ALIBABA CLOUD

阿里云

对象存储 OSS
SDK 示例

文档版本：20210802

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.简介 29
2.Java 30
2.1. 前言 30
2.2. 安装 31
2.3. 初始化 32
2.4. 快速入门 37
2.5. 存储空间 40
2.5.1. 创建存储空间 40
2.5.2. 列举存储空间 41
2.5.3. 判断存储空间是否存在 44
2.5.4. 获取存储空间的地域 44
2.5.5. 获取存储空间的信息 45
2.5.6. 管理存储空间访问权限 45
2.5.7. 删除存储空间 47
2.5.8. 存储空间标签 47
2.5.9. 存储空间清单 49
2.5.10. 授权策略 54
2.5.11. 合规保留策略 56
2.5.12. 生命周期 59
2.5.13. 请求者付费模式 63
2.5.14. 防盗链 65
2.5.15. 访问日志 67
2.5.16. 静态网站托管 69
2.5.17. 跨区域复制 70
2.5.18. 跨域资源共享 73
2.5.19. 传输加速 76

2.6. 上传文件 77
2.6.1. 概述 77
2.6.2. 简单上传 77
2.6.3. 表单上传 81
2.6.4. 追加上传 84
2.6.5. 断点续传上传 85
2.6.6. 分片上传 87
2.6.7. 进度条 96
2.6.8. 上传回调 98
2.7. 下载文件 99
2.7.1. 概述 99
2.7.2. 流式下载 99
2.7.3. 下载到本地文件 100
2.7.4. 范围下载 101
2.7.5. 断点续传下载 103
2.7.6. 限定条件下载 105
2.7.7. 进度条 106
2.8. 管理文件 107
2.8.1. 概述 107
2.8.2. 判断文件是否存在 108
2.8.3. 管理文件访问权限 108
2.8.4. 管理文件元信息 109
2.8.5. 转换文件存储类型 113
2.8.6. 列举文件 115
2.8.7. 查询文件 138
2.8.8. 删除文件 140
2.8.9. 拷贝文件 143
2.8.10. 禁止覆盖同名文件 147

2.8.11. 解冻文件 152
2.8.12. 管理软链接 154
2.9. 版本控制 156
2.9.1. 管理版本控制 156
2.9.2. 上传文件 157
2.9.3. 下载文件 161
2.9.4. 拷贝文件 162
2.9.5. 列举文件 164
2.9.6. 删除文件 171
2.9.7. 解冻文件 175
2.9.8. 获取文件元信息 176
2.9.9. 管理文件访问权限 177
2.9.10. 管理软链接 178
2.10. 对象标签 180
2.10.1. 设置对象标签 180
2.10.2. 获取对象标签 190
2.10.3. 删除对象标签 192
2.10.4. 对象标签和生命周期管理 193
2.11. 单链接限速 195
2.12. 数据加密 197
2.12.1. 服务器端加密 197
2.12.2. 客户端加密 199
2.13. 授权访问 210
2.14. 数据安全性 219
2.15. 图片处理 222
2.16. 异常处理 228
2.17. 常见问题 231
3.PHP 246

3.1. 前言 246
3.2. 安装 246
3.3. 初始化 248
3.4. 快速入门 251
3.5. 存储空间 256
3.5.1. 创建存储空间 256
3.5.2. 列举存储空间 257
3.5.3. 判断存储空间是否存在 261
3.5.4. 获取存储空间的地域 262
3.5.5. 获取存储空间信息 263
3.5.6. 获取存储空间元信息 264
3.5.7. 管理存储空间访问权限 265
3.5.8. 删除存储空间 267
3.5.9. 存储空间标签 268
3.5.10. 生命周期 271
3.5.11. 授权策略 275
3.5.12. 合规保留策略 278
3.5.13. 请求者付费模式 283
3.5.14. 防盗链 286
3.5.15. 访问日志 289
3.5.16. 静态网站托管 292
3.5.17. 跨域资源共享 295
3.5.18. 自定义域名绑定 298
3.6. 上传文件 301
3.6.1. 概述 301
3.6.2. 简单上传 301
3.6.3. 追加上传 303
3.6.4. 分片上传 305

3.6.5. 上传回调 313
3.7. 下载文件 316
3.7.1. 概述 316
3.7.2. 下载到本地文件 316
3.7.3. 下载到本地内存 317
3.7.4. 范围下载 318
3.7.5. 限定条件下载 319
3.8. 管理文件 320
3.8.1. 概述 320
3.8.2. 判断文件是否存在 321
3.8.3. 管理文件访问权限 321
3.8.4. 管理文件元信息 323
3.8.5. 转换文件存储类型 326
3.8.6. 禁止覆盖同名文件 329
3.8.7. 列举文件 331
3.8.8. 删除文件 335
3.8.9. 拷贝文件 336
3.8.10. 解冻文件 338
3.8.11. 管理软链接 340
3.8.12. 开启MD5校验 342
3.9. 版本控制 343
3.9.1. 管理版本控制 343
3.9.2. 上传文件 346
3.9.3. 下载文件 350
3.9.4. 拷贝文件 351
3.9.5. 删除文件 354
3.9.6. 解冻文件 358
3.9.7. 获取文件元信息 359

3.9.8. 管理文件访问权限 360
3.9.9. 管理软链接 362
3.10. 对象标签 364
3.10.1. 设置对象标签 364
3.10.2. 获取对象标签 368
3.10.3. 删除对象标签 369
3.11. 单链接限速 370
3.12. 数据加密 372
3.12.1. 服务器端加密 372
3.13. 授权访问 375
3.14. 图片处理 378
3.15. 异常处理 383
4.Node.js 386
4.1. 安装 386
4.2. 初始化 387
4.3. 配置项 388
4.4. 快速入门 389
4.5. 存储空间 391
4.5.1. 创建存储空间 391
4.5.2. 列举存储空间 392
4.5.3. 判断存储空间是否存在 393
4.5.4. 获取存储空间的地域 394
4.5.5. 获取存储空间的信息 394
4.5.6. 管理存储空间访问权限 395
4.5.7. 删除存储空间 397
4.5.8. 请求者付费模式 398
4.5.9. 生命周期 400
4.5.10. 授权策略 405

4.5.11. 合规保留策略 407
4.5.12. 存储空间标签 410
4.5.13. 存储空间清单 412
4.5.14. 跨域资源共享 417
4.5.15. 访问日志 419
4.5.16. 防盗链 420
4.5.17. 静态网站托管 422
4.5.18. 使用自定义域名 423
4.6. 上传文件 424
4.6.1. 概述 424
4.6.2. 上传本地文件 424
4.6.3. 上传本地内存 425
4.6.4. 流式上传 426
4.6.5. 分片上传 426
4.6.6. 追加上传 432
4.6.7. 断点续传上传 433
4.6.8. 上传回调 434
4.7. 下载文件 435
4.7.1. 概述 435
4.7.2. 下载到本地文件 436
4.7.3. 下载到本地内存 436
4.7.4. 流式下载 437
4.7.5. 范围下载 438
4.7.6. 限定条件下载 440
4.8. 管理文件 444
4.8.1. 概述 444
4.8.2. 判断文件是否存在 444
4.8.3. 管理文件元信息 445

4.8.4. 管理文件访问权限 446
4.8.5. 列举文件 448
4.8.6. 拷贝文件 457
4.8.7. 禁止覆盖同名文件 459
4.8.8. 删除文件 462
4.8.9. 转换文件存储类型 465
4.8.10. 解冻归档文件 466
4.8.11. 管理软链接 467
4.9. 版本控制 468
4.9.1. 管理版本控制 468
4.9.2. 上传文件 469
4.9.3. 下载文件 471
4.9.4. 拷贝文件 472
4.9.5. 列举文件 474
4.9.6. 删除文件 479
4.9.7. 解冻文件 482
4.9.8. 获取文件元信息 483
4.9.9. 管理文件访问权限 484
4.9.10. 管理软链接 486
4.10. 对象标签 487
4.10.1. 设置对象标签 487
4.10.2. 获取对象标签 494
4.10.3. 删除对象标签 495
4.10.4. 对象标签和生命周期管理 496
4.11. 单链接限速 497
4.12. 服务器端加密 499
4.13. 授权访问 501
4.14. 图片处理 506

4.15. 异常处理 511
4.16. 常见问题 512
5.Python 514
5.1. 前言 514
5.2. 安装 514
5.3. 初始化 517
5.4. 快速入门 519
5.5. 存储空间 520
5.5.1. 创建存储空间 520
5.5.2. 列举存储空间 521
5.5.3. 获取存储空间的地域 522
5.5.4. 判断存储空间是否存在 522
5.5.5. 获取存储空间信息 523
5.5.6. 管理存储空间访问权限 523
5.5.7. 删除存储空间 524
5.5.8. 存储空间标签 525
5.5.9. 存储空间清单 527
5.5.10. 授权策略 532
5.5.11. 合规保留策略 533
5.5.12. 请求者付费模式 535
5.5.13. 生命周期 537
5.5.14. 防盗链 541
5.5.15. 访问日志 542
5.5.16. 静态网站托管 544
5.5.17. 跨域资源共享 545
5.5.18. 跨区域复制 546
5.6. 上传文件 549
5.6.1. 概述 549

5.6.2. 简单上传 549
5.6.3. 追加上传 552
5.6.4. 断点续传上传 553
5.6.5. 分片上传 554
5.6.6. 进度条 557
5.6.7. 上传回调 558
5.7. 下载文件 559
5.7.1. 概述 559
5.7.2. 流式下载 559
5.7.3. 下载到本地文件 561
5.7.4. 范围下载 561
5.7.5. 断点续传下载 563
5.7.6. 进度条 564
5.8. 管理文件 565
5.8.1. 概述 565
5.8.2. 判断文件是否存在 565
5.8.3. 管理文件访问权限 566
5.8.4. 管理文件元信息 567
5.8.5. 转换文件存储类型 569
5.8.6. 列举文件 570
5.8.7. 查询文件 576
5.8.8. 删除文件 581
5.8.9. 拷贝文件 582
5.8.10. 禁止覆盖同名文件 583
5.8.11. 解冻文件 586
5.8.12. 管理软链接 588
5.9. 版本控制 589
5.9.1. 管理版本控制 589

5.9.2. 上传文件 590
5.9.3. 下载文件 592
5.9.4. 拷贝文件 593
5.9.5. 列举文件 595
5.9.6. 删除文件 600
5.9.7. 解冻文件 604
5.9.8. 获取文件元信息 605
5.9.9. 管理文件访问权限 606
5.9.10. 管理软链接 607
5.10. 对象标签 608
5.10.1. 设置对象标签 608
5.10.2. 获取对象标签 617
5.10.3. 删除对象标签 618
5.10.4. 对象标签和生命周期管理 618
5.11. 单链接限速 621
5.12. 数据加密 623
5.12.1. 客户端加密 623
5.12.2. 服务器端加密 631
5.13. 授权访问 633
5.14. 数据安全性 635
5.15. 开启Python SDK日志 637
5.16. 图片处理 639
5.17. 中文和时间 643
5.18. 异常处理 645
5.19. 常见问题 646
6.Browser.js 650
6.1. 安装 650
6.2. 配置项 654

6.3. 快速开始 654
6.4. 上传文件 661
6.5. 下载文件 664
6.6. 管理文件 666
6.7. 自定义域名绑定 670
6.8. 授权访问 671
6.9. 访问权限 673
6.10. 图片处理 674
6.11. 浏览器应用 677
6.12. 异常处理 680
6.13. 常见问题 680
6.14. 日志收集 684
7..NET 686
7.1. 前言 686
7.2. 安装 687
7.3. 初始化 688
7.4. 快速入门 690
7.5. 存储空间 694
7.5.1. 创建存储空间 694
7.5.2. 列举存储空间 695
7.5.3. 判断存储空间是否存在 695
7.5.4. 获取存储空间的地域 696
7.5.5. 获取存储空间的信息 697
7.5.6. 管理存储空间访问权限 697
7.5.7. 删除存储空间 699
7.5.8. 存储空间标签 700
7.5.9. 存储空间清单 705
7.5.10. 生命周期 709

7.5.11. 请求者付费模式 713
7.5.12. 授权策略 716
7.5.13. 访问日志 719
7.5.14. 跨域资源共享 722
7.5.15. 静态网站托管 725
7.5.16. 防盗链 727
7.6. 上传文件 730
7.6.1. 概述 730
7.6.2. 简单上传 730
7.6.3. 追加上传 735
7.6.4. 断点续传上传 736
7.6.5. 分片上传 737
7.6.6. 进度条 744
7.6.7. 上传回调 746
7.7. 下载文件 747
7.7.1. 概述 747
7.7.2. 流式下载 747
7.7.3. 范围下载 748
7.7.4. 断点续传下载 750
7.7.5. 进度条 751
7.8. 管理文件 753
7.8.1. 概述 753
7.8.2. 判断文件是否存在 753
7.8.3. 管理文件访问权限 753
7.8.4. 管理文件元信息 755
7.8.5. 列举文件 757
7.8.6. 删除文件 762
7.8.7. 拷贝文件 764

7.8.8. 解冻文件 767
7.8.9. 管理软链接 770
7.9. 版本控制 771
7.9.1. 管理版本控制 771
7.9.2. 上传文件 773
7.9.3. 下载文件 777
7.9.4. 拷贝文件 778
7.9.5. 删除文件 781
7.9.6. 解冻文件 787
7.9.7. 获取文件元信息 788
7.9.8. 管理文件访问权限 789
7.9.9. 管理软链接 791
7.10. 对象标签 793
7.10.1. 设置对象标签 793
7.10.2. 获取对象标签 805
7.10.3. 删除对象标签 805
7.10.4. 对象标签和生命周期管理 806
7.11. 单链接限速 808
7.12. 服务器端加密 810
7.13. 授权访问 813
7.14. 图片处理 815
7.15. 异常处理 821
8.Android 824
8.1. 前言 824
8.2. 安装 824
8.3. 初始化 825
8.4. 快速入门 829
8.5. 存储空间 833

8.5.1. 创建存储空间 833
8.5.2. 获取存储空间的访问权限 834
8.5.3. 获取存储空间信息 835
8.5.4. 删除存储空间 836
8.5.5. 生命周期 837
8.5.6. 防盗链 839
8.5.7. 访问日志 841
8.6. 上传文件 843
8.6.1. 概述 843
8.6.2. 简单上传 843
8.6.3. 追加上传 847
8.6.4. 分片上传 848
8.6.5. 断点续传上传 852
8.6.6. 进度条 855
8.6.7. 上传回调 856
8.7. 下载文件 857
8.7.1. 概述 857
8.7.2. 流式下载 858
8.7.3. 范围下载 859
8.7.4. 下载到本地文件 860
8.7.5. 限定条件下载 861
8.7.6. 进度条 862
8.8. 管理文件 863
8.8.1. 概述 863
8.8.2. 判断文件是否存在 863
8.8.3. 获取文件访问权限 864
8.8.4. 拷贝文件 865
8.8.5. 列举文件 866

8.8.6. 删除文件 872
8.8.7. 设置Content-Type 874
8.8.8. 获取文件元信息 875
8.8.9. 解冻归档文件 875
8.8.10. 管理软链接 876
8.9. 数据安全性 877
8.10. 签名URL 878
8.11. 授权访问 878
8.12. 图片处理 882
8.13. 异常处理 883
9.Go 885
9.1. 前言 885
9.2. 安装 886
9.3. 初始化 887
9.4. 快速入门 889
9.5. 存储空间 893
9.5.1. 创建存储空间 893
9.5.2. 列举存储空间 894
9.5.3. 判断存储空间是否存在 896
9.5.4. 获取存储空间的地域 897
9.5.5. 获取存储空间的信息 898
9.5.6. 管理存储空间的访问权限 898
9.5.7. 删除存储空间 900
9.5.8. 存储空间标签 901
9.5.9. 存储空间清单 905
9.5.10. 传输加速 909
9.5.11. 授权策略 911
9.5.12. 生命周期 914

9.5.13. 请求者付费模式 917
9.5.14. 防盗链 919
9.5.15. 访问日志 921
9.5.16. 静态网站托管 924
9.5.17. 跨域资源共享 926
9.6. 上传文件 928
9.6.1. 概述 928
9.6.2. 简单上传 928
9.6.3. 追加上传 932
9.6.4. 断点续传上传 934
9.6.5. 分片上传 935
9.6.6. 进度条 943
9.7. 下载文件 945
9.7.1. 概述 945
9.7.2. 流式下载 945
9.7.3. 范围下载 948
9.7.4. 下载到本地文件 951
9.7.5. 断点续传下载 951
9.7.6. 限定条件下载 953
9.7.7. 文件压缩下载 955
9.7.8. 进度条 955
9.8. 管理文件 956
9.8.1. 概述 957
9.8.2. 判断文件是否存在 957
9.8.3. 管理文件访问权限 957
9.8.4. 管理文件元信息 959
9.8.5. 转换文件存储类型 963
9.8.6. 列举文件 966

9.8.7. 删除文件 983
9.8.8. 拷贝文件 987
9.8.9. 禁止覆盖同名文件 991
9.8.10. 解冻文件 994
9.8.11. 管理软链接 997
9.9. 版本控制 999
9.9.1. 管理版本控制 999
9.9.2. 上传文件 1002
9.9.3. 下载文件 1006
9.9.4. 拷贝文件 1006
9.9.5. 列举文件 1009
9.9.6. 删除文件 1021
9.9.7. 解冻文件 1027
9.9.8. 获取文件元信息 1029
9.9.9. 管理文件访问权限 1030
9.9.10. 管理软链接 1032
9.10. 对象标签 1034
9.10.1. 设置对象标签 1034
9.10.2. 获取对象标签 1043
9.10.3. 删除对象标签 1044
9.10.4. 对象标签和生命周期管理 1045
9.11. 数据加密 1047
9.11.1. 服务器端加密 1047
9.11.2. 客户端加密 1049
9.12. 单链接限速 1057
9.13. 数据安全性 1060
9.14. 授权访问 1062
9.15. 图片处理 1067

9.16. 错误处理 1072
10.iOS 1077
10.1. 前言 1077
10.2. 安装 1077
10.3. 快速入门 1078
10.4. 初始化 1081
10.5. 存储空间 1084
10.5.1. 创建存储空间 1084
10.5.2. 列举存储空间 1085
10.5.3. 获取存储空间访问权限 1087
10.5.4. 获取存储空间的信息 1088
10.5.5. 删除存储空间 1088
10.6. 上传文件 1089
10.6.1. 概述 1089
10.6.2. 简单上传 1089
10.6.3. 分片上传 1091
10.6.4. 追加上传 1097
10.6.5. 断点续传上传 1097
10.6.6. 上传回调 1099
10.6.7. 进度条 1100
10.7. 下载文件 1101
10.7.1. 概述 1101
10.7.2. 简单下载 1101
10.7.3. 流式下载 1102
10.7.4. 范围下载 1103
10.7.5. 断点续传下载 1104
10.7.6. 进度条 1111
10.8. 管理文件 1112

10.8.1. 概述 1112
10.8.2. 判断文件是否存在 1112
10.8.3. 获取文件元信息 1113
10.8.4. 列举文件 1113
10.8.5. 拷贝文件 1118
10.8.6. 删除文件 1118
10.8.7. 管理软链接 1119
10.8.8. 解冻归档文件 1120
10.8.9. 管理文件访问权限 1121
10.9. 数据安全性 1122
10.10. 签名URL 1124
10.11. 授权访问 1125
10.12. 图片处理 1128
10.13. 异常响应 1129
11.C++ 1131
11.1. 前言 1131
11.2. 安装 1131
11.3. 初始化 1134
11.4. 快速入门 1139
11.5. 存储空间 1143
11.5.1. 创建存储空间 1143
11.5.2. 列举存储空间 1144
11.5.3. 判断存储空间是否存在 1145
11.5.4. 获取存储空间的地域 1146
11.5.5. 获取存储空间的信息 1147
11.5.6. 管理存储空间访问权限 1148
11.5.7. 删除存储空间 1150
11.5.8. 存储空间标签 1151

11.5.9. 存储空间清单 1156
11.5.10. 授权策略 1161
11.5.11. 合规保留策略 1165
11.5.12. 生命周期 1170
11.5.13. 请求者付费模式 1174
11.5.14. 防盗链 1177
11.5.15. 访问日志 1180
11.5.16. 静态网站托管 1183
11.5.17. 跨域资源共享 1186
11.6. 上传文件 1189
11.6.1. 概述 1189
11.6.2. 简单上传 1189
11.6.3. 追加上传 1191
11.6.4. 断点续传上传 1193
11.6.5. 分片上传 1194
11.6.6. 中断上传 1201
11.6.7. 进度条 1202
11.6.8. 上传回调 1203
11.7. 下载文件 1204
11.7.1. 概述 1204
11.7.2. 下载到本地文件 1204
11.7.3. 下载到本地内存 1205
11.7.4. 范围下载 1206
11.7.5. 断点续传下载 1208
11.7.6. 进度条 1210
11.8. 管理文件 1211
11.8.1. 概述 1211
11.8.2. 判断文件是否存在 1212

11.8.3. 管理文件访问权限 1212
11.8.4. 管理文件元信息 1214
11.8.5. 转换文件存储类型 1217
11.8.6. 列举文件 1219
11.8.7. 删除文件 1234
11.8.8. 拷贝文件 1237
11.8.9. 禁止覆盖同名文件 1240
11.8.10. 解冻文件 1246
11.8.11. 管理软链接 1248
11.9. 版本控制 1250
11.9.1. 管理版本控制 1250
11.9.2. 上传文件 1253
11.9.3. 下载文件 1258
11.9.4. 拷贝文件 1259
11.9.5. 删除文件 1262
11.9.6. 解冻文件 1267
11.9.7. 获取文件元信息 1268
11.9.8. 管理文件访问权限 1270
11.9.9. 管理软链接 1271
11.10. 对象标签 1273
11.10.1. 设置对象标签 1273
11.10.2. 获取对象标签 1283
11.10.3. 删除对象标签 1284
11.10.4. 对象标签和生命周期管理 1285
11.11. 数据加密 1287
11.11.1. 服务器端加密 1287
11.11.2. 客户端加密 1291
11.12. 单链接限速 1303

11.13. 数据安全性 1305
11.14. 授权访问 1307
11.15. 图片处理 1310
12.C 1315
12.1. 前言 1315
12.2. 安装 1316
12.3. 初始化 1319
12.4. 快速入门 1324
12.5. 存储空间 1331
12.5.1. 创建存储空间 1331
12.5.2. 列举存储空间 1333
12.5.3. 获取存储空间的地域 1334
12.5.4. 获取存储空间的信息 1335
12.5.5. 管理存储空间访问权限 1336
12.5.6. 删除存储空间 1339
12.5.7. 生命周期 1340
12.5.8. 跨域资源共享 1345
12.5.9. 访问日志 1350
12.5.10. 防盗链 1353
12.5.11. 静态网站托管 1357
12.6. 上传文件 1361
12.6.1. 概述 1361
12.6.2. 简单上传 1361
12.6.3. 追加上传 1365
12.6.4. 断点续传上传 1369
12.6.5. 分片上传 1371
12.6.6. 进度条 1375
12.6.7. 上传回调 1376

12.7. 下载文件 1378
12.7.1. 概述 1378
12.7.2. 下载到本地文件 1379
12.7.3. 下载到本地内存 1380
12.7.4. 范围下载 1382
12.7.5. 断点续传下载 1383
12.7.6. 进度条 1385
12.8. 管理文件 1387
12.8.1. 概述 1387
12.8.2. 管理文件访问权限 1387
12.8.3. 管理文件元信息 1389
12.8.4. 列举文件 1390
12.8.5. 删除文件 1405
12.8.6. 拷贝文件 1409
12.8.7. 解冻归档文件 1413
12.8.8. 管理软链接 1415
12.9. 授权访问 1417
12.10. 图片处理 1422
12.11. 错误处理 1427
12.12. 常见问题 1429
13.Ruby 1430
13.1. 安装 1430
13.2. 快速入门 1432
13.3. 存储空间 1435
13.3.1. 创建存储空间 1435
13.3.2. 列举存储空间 1435
13.3.3. 判断存储空间是否存在 1435
13.3.4. 管理存储空间访问权限 1436

13.3.5. 删除存储空间 1437
13.3.6. 生命周期 1437
13.3.7. 防盗链 1438
13.3.8. 访问日志 1439
13.3.9. 跨域资源共享 1440
13.3.10. 静态网站托管 1441
13.3.11. 自定义域名绑定 1442
13.4. 上传文件 1443
13.5. 下载文件 1446
13.6. 管理文件 1448
13.7. 使用STS进行临时授权 1453
13.8. 异常处理 1454
13.9. Rails应用 1455

1.简介

本文介绍阿里云对象存储OSS提供的主流语言SDK。

前提条件

- 已开通OSS服务，详情请参见开通OSS服务。
- 已创建AccessKey，详情请参见创建AccessKey。

示例代码

请参见下表了解主流语言SDK的安装、各类操作、涉及参数、示例和常见问题等说明。

语言	参考文档
Java	Java SDK参考
Python	Python SDK参考
Go	Go SDK参考
C++	C++ SDK参考
PHP	PHP SDK参考
C	C SDK参考
Android	Android SDK参考
iOS	iOS SDK参考
.NET	.NET SDK参考
Node.js	Node.js SDK参考
Browser.js	Browser.js SDK参考
Ruby	Ruby SDK参考

2.Java

2.1. 前言

本文档基于OSS Java SDK 3.10.2版本编写。

兼容性

Java SDK版本兼容性说明如下：

- 对于3.x.x系列SDK：
  - 接口： 兼容。
  - 命名空间： 兼容。
- 对于2.x.x系列SDK：
  - 接口： 兼容。
  - 命名空间： 兼容。
- 对于1.0.x 系列SDK：
  - 接口：兼容。
  - 命名空间：不兼容。2.0.0版本移除了1.0.x版本中TableStore相关代码，将包名称 `com.aliyun.openservices.*` 和 `com.aliyun.openservices.oss.*` 更换为 `com.aliyun.oss.*` 。

SDK源码和API文档

SDK源码请参见GitHub。更多信息请参见OSS Java SDK API文档。

示例代码

OSS Java SDK提供丰富的示例代码，方便您参考或直接使用。示例代码包括以下内容：

示例文件	示例内容
GetStartedSample.java	快速入门
CreateFolderSample.java	简单上传中的创建文件夹
AppendObjectSample.java	追加上传
UploadSample.java	断点续传上传
MultipartUploadSample.java	分片上传
GetProgressSample.java	进度条
CallbackSample.java	上传回调
CRCSample.java	CRC64校验
SimpleGetObjectSample.java	下载文件
DownloadSample.java	断点续传下载

示例文件	示例内容
ObjectMetaSample.java	文件元信息
ListObjectsSample.java	列举文件
UploadPartCopySample.java	拷贝文件
DeleteObjectsSample.java	删除文件
BucketOperationsSample.java	设置存储空间的授权访问、生命周期、访问日志、防盗链、CORS等
ImageSample.java	图片处理
PostObjectSample.java	表单上传（PostObject）的实现不依赖Java SDK
ConcurrentGetObjectSample.java	并发下载文件，推荐使用断点续传下载
TrafficLimitSample.java	设置上传、下载文件时的单链接限速
SetRequestPaymentSample.java	存储空间的请求者付费模式
EncryptionClientRsaSample.java、EncryptionClientKmsSample.java	客户端加密

2.2. 安装

进行OSS各类操作前，您需要先安装Java SDK。本文提供了Java SDK的多种安装方式，请结合实际使用场景选用。

前提条件

使用Java 1.7及以上版本。

您可以通过命令**java -version**查看Java版本。

下载SDK

- SDK安装包
- 通过GitHub下载
- 历史版本下载

安装SDK

您可以通过以下三种方式安装SDK。

- 方式一：在Maven项目中加入依赖项（推荐方式）

  在Maven工程中使用OSS Java SDK，只需在pom.xml中加入相应依赖即可。以3.10.2版本为例，在<dependencies>中加入如下内容：

```
<dependency>
  <groupId>com.aliyun.oss</groupId>
  <artifactId>aliyun-sdk-oss</artifactId>
  <version>3.10.2</version>
</dependency>
```

如果使用的是Java 9及以上的版本，则需要添加jaxb相关依赖。添加jaxb相关依赖示例代码如下：

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>
<!-- no more than 2.3.3-->
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>2.3.3</version>
</dependency>
```

- 方式二：在Eclipse项目中导入JAR包

  以3.10.2版本为例，步骤如下：

  i. 下载Java SDK 开发包。
  ii. 解压该开发包。
  iii. 将解压后文件夹中的文件*aliyun-sdk-oss-3.10.2.jar*以及lib文件夹下的所有文件拷贝到您的项目中。
  iv. 在Eclipse中选择您的工程，右键选择**Properties > Java Build Path > Add JARs**。
  v. 选中拷贝的所有JAR文件，导入到Libraries中。

- 方式三：在IntelliJ IDEA项目中导入JAR包

  以3.10.2版本为例，步骤如下：

  i. 下载Java SDK 开发包。
  ii. 解压该开发包。
  iii. 将解压后文件夹中的文件*aliyun-sdk-oss-3.10.2.jar*以及lib文件夹下的所有JAR文件拷贝到您的项目中。
  iv. 在IntelliJ IDEA中选择您的工程，右键选择**File > Project Structure > Modules > Dependencies > + > JARs or directories** 。
  v. 选中拷贝的所有JAR文件，导入到External Libraries中。

2.3. 初始化

OSSClient是OSS的Java客户端，用于管理存储空间和文件等OSS资源。使用Java SDK发起OSS请求，您需要初始化一个OSSClient实例，并根据需要修改ClientConfiguration的默认配置项。

新建OSSClient

新建OSSClient时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

- 使用OSS域名新建OSSClient

  以下代码用于使用OSS域名新建OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 关闭OSSClient。
ossClient.shutdown();
```

  您的工程中可以有一个或多个OSSClient。OSSClient可以并发使用。

- 使用自定义域名新建OSSClient

  以下代码用于使用自定义域名新建OSSClient：

```
String endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建ClientConfiguration实例，您可以按照实际情况修改默认参数。
ClientBuilderConfiguration conf = new ClientBuilderConfiguration();
// 设置是否支持CNAME。CNAME用于将自定义域名绑定到目标Bucket。
conf.setSupportCname(true);
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, conf);
// 关闭OSSClient。
ossClient.shutdown();
```

  说明 使用自定义域名时无法使用ossClient.listBuckets方法。

- 专有云或专有域环境新建OSSClient

  以下代码用于使用专有云或专有域环境新建OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建ClientConfiguration实例，按照您的需要修改默认参数。
ClientBuilderConfiguration conf = new ClientBuilderConfiguration();
// 关闭CNAME选项。
conf.setSupportCname(false);
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, conf);
// 关闭OSSClient。
ossClient.shutdown();
```

- 使用IP新建OSSClient

  以下代码用于使用IP新建OSSClient：

```
// 某些特殊情况（比如专有域）下，您需要将IP地址作为Endpoint，请按照实际IP地址填写。
String endpoint = "https://10.10.10.10";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建ClientConfiguration。
ClientBuilderConfiguration conf = new ClientBuilderConfiguration();
// 开启二级域名访问OSS，默认不开启。OSS Java SDK 2.1.2及之前的版本需要设置此值，OSS Java SDK 2.1.2及之后以后的版本会自动检测到IP地址，不需要再设置此值。
conf.setSLDEnabled(true);
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, conf);
// 关闭OSSClient。
ossClient.shutdown();
```

- 使用STS新建OSSClient

  以下代码用于使用STS新建OSSClient：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String securityToken = "<yourSecurityToken>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 关闭OSSClient。
ossClient.shutdown();
```

  更多信息，请参见授权访问。

- 使用STSAssumeRole新建OSSClient

  以下代码用于使用STSAssumeRole新建OSSClient：

```
// 授权STSAssumeRole访问的Region。以杭州为例，其它Region请按实际情况填写。
String region = "cn-hangzhou";
// 填写RAM用户的访问密钥（AccessKeyId和AccessKeySecret）。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 角色的ARN，即需要扮演的角色ID。
String roleArn = "<yourRoleArn>";
// 创建STSAssumeRoleSessionCredentialsProvider实例。
STSAssumeRoleSessionCredentialsProvider provider = CredentialsProviderFactory.newSTSAssumeRoleS
essionCredentialsProvider(
region, accessKeyId, accessKeySecret, roleArn);
// 通过STSAssumeRole访问OSS的Endpoint。以杭州为例，其它endpoint请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 创建ClientConfiguration实例。
ClientBuilderConfiguration conf = new ClientBuilderConfiguration();
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, provider, conf);
// 关闭OSSClient。
ossClient.shutdown();
```

更多信息，请参见使用STS临时访问凭证访问OSS。

- 使用EcsRamRole新建OSSClient

  在云服务器ECS上，您可以通过实例RAM角色的方式访问OSS。实例RAM角色允许您将一个角色关联到云服务器实例，在实例内部基于临时凭证STS访问OSS。临时凭证由系统自动生成和更新，应用程序可以使用指定的实例元数据URL获取临时凭证，无需特别管理。

  以下代码用于使用EcsRamRole新建OSSClient：

```
// 通过ECS RAM角色访问OSS。
// 以杭州为例，其它endpoint请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 通过ECS RAM角色获取访问凭证，例如以角色名（ecs-ram-role）访问为例。
InstanceProfileCredentialsProvider provider = CredentialsProviderFactory.newInstanceProfileCredential
sProvider("ecs-ram-role");
// 创建OSSClient实例。
 OSS ossClient = new OSSClientBuilder().build(endpoint, provider, new ClientBuilderConfiguration());
// 关闭OSSClient。
ossClient.shutdown();
```

配置OSSClient

ClientConfiguration是OSSClient的配置类，您可通过此类来配置代理、连接超时、最大连接数等参数。可设置的参数如下：

参数	描述	方法
MaxConnections	允许打开的最大HTTP连接数。默认为1024。	ClientConfiguration.setMaxConnections
SocketTimeout	Socket层传输数据的超时时间（单位：毫秒）。默认为50000毫秒。	ClientConfiguration.setSocketTimeout

参数	描述	方法
ConnectionTimeout	建立连接的超时时间（单位：毫秒）。默认为50000毫秒。	ClientConfiguration.setConnectionTimeout
ConnectionRequestTimeout	从连接池中获取连接的超时时间（单位：毫秒）。默认不超时。	ClientConfiguration.setConnectionRequestTimeout
IdleConnectionTime	连接空闲超时时间，超时则关闭连接（单位：毫秒）。默认为60000毫秒。	ClientConfiguration.setIdleConnectionTime
MaxErrorRetry	请求失败后最大的重试次数。默认3次。	ClientConfiguration.setMaxErrorRetry
SupportCname	是否支持CNAME作为Endpoint，默认支持CNAME。	ClientConfiguration.setSupportCname
SLDEnabled	是否开启二级域名（Second Level Domain）的访问方式，默认不开启。	ClientConfiguration.setSLDEnabled
Protocol	连接OSS所采用的协议（HTTP或HTTPS），默认为HTTP。	ClientConfiguration.setProtocol
UserAgent	用户代理，指HTTP的User-Agent头。默认为 `aliyun-sdk-java` 。	ClientConfiguration.setUserAgent
ProxyHost	代理服务器主机地址。	ClientConfiguration.setProxyHost
ProxyPort	代理服务器端口。	ClientConfiguration.setProxyPort
ProxyUsername	代理服务器验证的用户名。	ClientConfiguration.setProxyUsername
ProxyPassword	代理服务器验证的密码。	ClientConfiguration.setProxyPassword
RedirectEnable	是否开启HTTP重定向。 **说明** Java SDK 3.10.1及以上版本支持设置是否开启HTTP重定向，默认开启。	ClientConfiguration.setRedirectEnable
VerifySSLEnable	是否开启SSL证书校验。 **说明** Java SDK 3.10.1及以上版本支持设置是否开启SSL证书校验，默认开启。	ClientConfiguration.setVerifySSLEnable

以下代码用于使用ClientConfiguration设置OSSClient参数：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建ClientConfiguration。ClientConfiguration是OSSClient的配置类，可配置代理、连接超时、最大连接数等参数。
ClientBuilderConfiguration conf = new ClientBuilderConfiguration();
// 设置OSSClient允许打开的最大HTTP连接数，默认为1024个。
conf.setMaxConnections(200);
// 设置Socket层传输数据的超时时间，默认为50000毫秒。
conf.setSocketTimeout(10000);
// 设置建立连接的超时时间，默认为50000毫秒。
conf.setConnectionTimeout(10000);
// 设置从连接池中获取连接的超时时间（单位：毫秒），默认不超时。
conf.setConnectionRequestTimeout(1000);
// 设置连接空闲超时时间。超时则关闭连接，默认为60000毫秒。
conf.setIdleConnectionTime(10000);
// 设置失败请求重试次数，默认为3次。
conf.setMaxErrorRetry(5);
// 设置是否支持将自定义域名作为Endpoint，默认支持。
conf.setSupportCname(true);
// 设置是否开启二级域名的访问方式，默认不开启。
conf.setSLDEnabled(true);
// 设置连接OSS所使用的协议（HTTP或HTTPS），默认为HTTP。
conf.setProtocol(Protocol.HTTP);
// 设置用户代理，指HTTP的User-Agent头，默认为aliyun-sdk-java。
conf.setUserAgent("aliyun-sdk-java");
// 设置代理服务器端口。
conf.setProxyHost("<yourProxyHost>");
// 设置代理服务器验证的用户名。
conf.setProxyUsername("<yourProxyUserName>");
// 设置代理服务器验证的密码。
conf.setProxyPassword("<yourProxyPassword>");
// 设置是否开启HTTP重定向，默认开启。
conf.setRedirectEnable(true);
// 设置是否开启SSL证书校验，默认开启。
conf.setVerifySSLEnable(true);
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, conf);
// 关闭OSSClient。
ossClient.shutdown();
```

更多信息，请参见阿里云OSS Java SDK超时时间设置。

2.4. 快速入门

本文介绍如何快速使用OSS Java SDK完成常见操作，如创建存储空间（Bucket）、上传文件（Object）、下载文件等。

示例工程

OSS Java SDK提供了基于Maven和Ant的示例工程。您可以在本地设备上编译和运行示例工程，或者以示例工程为基础开发您的应用。工程的编译和运行方法，请参见工程目录下的README.md。

- Maven示例工程：aliyun-oss-java-sdk-demo-mvn.zip
- Ant示例工程：aliyun-oss-java-sdk-demo-ant.zip

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建存储空间。
ossClient.createBucket(bucketName);
// 关闭OSSClient。
ossClient.shutdown();
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 上传文件到指定的存储空间（bucketName）并将其保存为指定的文件名称（objectName）。
String content = "Hello OSS";
ossClient.putObject(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
// 关闭OSSClient。
ossClient.shutdown();
```

上传文件详情请参见上传文件。

下载文件

以下代码用于下载文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// <yourObjectName>从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 调用ossClient.getObject返回一个OSSObject实例，该实例包含文件内容及文件元信息。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// 调用ossObject.getObjectContent获取文件输入流，可读取此输入流获取其内容。
InputStream content = ossObject.getObjectContent();
if (content != null) {
  BufferedReader reader = new BufferedReader(new InputStreamReader(content));
  while (true) {
    String line = reader.readLine();
    if (line == null) break;
    System.out.println("\n" + line);
  }
  // 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
  content.close();
}
// 关闭OSSClient。
ossClient.shutdown();
```

下载文件详情请参见下载文件。

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// ossClient.listObjects返回ObjectListing实例，包含此次listObject请求的返回结果。
ObjectListing objectListing = ossClient.listObjects(bucketName);
// objectListing.getObjectSummaries获取所有文件的描述信息。
for (OSSObjectSummary objectSummary : objectListing.getObjectSummaries()) {
   System.out.println(" - " + objectSummary.getKey() + "  " +
         "(size = " + objectSummary.getSize() + ")");
}
// 关闭OSSClient。
ossClient.shutdown();
```

列举文件详情请参见列举文件。

删除文件

删除文件完整代码请参见GitHub。

以下代码用于删除指定文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// <yourObjectName>表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除文件。
ossClient.deleteObject(bucketName, objectName);
// 关闭OSSClient。
ossClient.shutdown();
```

删除文件详情请参见删除文件。

2.5. 存储空间

2.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

以下代码用于创建examplebucket存储空间。

您可以在创建存储空间时指定存储类型和存储空间读写权限。更多信息，请分别参见存储类型介绍和设置存储空间读写权限（ACL）。

在支持资源组的地域创建存储空间时，您可以为Bucket配置资源组。关于资源组的更多信息，请参见资源组。

> **说明** 配置资源组时，您可以通过资源管理的控制台或者ListResourceGroups接口获取资源组ID。具体操作，请分别参见查看资源组基本信息和ListResourceGroups。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写资源组ID。如果未填写资源组ID，则创建的Bucket属于默认资源组。
//String rsId = "rg-aek27tc********";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建CreateBucketRequest对象。
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucketName);
// 如果创建存储空间的同时需要指定存储类型和数据容灾类型, 请参考如下代码。
// 此处以设置存储空间的存储类型为标准存储为例介绍。
//createBucketRequest.setStorageClass(StorageClass.Standard);
// 数据容灾类型默认为本地冗余存储，即DataRedundancyType.LRS。如果需要设置数据容灾类型为同城冗余存储，
请设置为DataRedundancyType.ZRS。
//createBucketRequest.setDataRedundancyType(DataRedundancyType.ZRS);
// 设置存储空间的权限为公共读，默认为私有。
//createBucketRequest.setCannedACL(CannedAccessControlList.PublicRead);
// 在支持资源组的地域创建Bucket时，您可以为Bucket配置资源组。
//createBucketRequest.setResourceGroupId(rsId);
// 创建存储空间。
ossClient.createBucket(createBucketRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

存储空间的命名规范请参见存储空间（Bucket）。

2.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

> **说明** 关于列举存储空间的更多信息，请参见GetService (ListBuckets)。如果有低频类型或归档类型的存储空间，请使用Java SDK 2.6.0及以上版本。

列举所有存储空间

以下代码用于列举所有存储空间。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举存储空间。
List<Bucket> buckets = ossClient.listBuckets();
for (Bucket bucket : buckets) {
  System.out.println(" - " + bucket.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

列举资源组中的存储空间

以下代码用于列举资源组中的存储空间。

> **说明** 一个云账户包含一个默认资源组和多个自定义的资源组。如果在创建存储空间时未指定资源组ID，则创建的存储空间均属于默认资源组。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写资源组ID。如果未填写资源组ID，则列举默认资源组中的存储空间。
String rsId = "rg-aek27tc********";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举存储空间。
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定资源组中的存储空间。
listBucketsRequest.setResourceGroupId(rsId);
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
  System.out.println(" - " + bucket.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

列举指定前缀的存储空间

以下代码用于列举以example为前缀（prefix）的存储空间。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定前缀的存储空间。
// 填写前缀。
listBucketsRequest.setPrefix("example");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
   System.out.println(" - " + bucket.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

列举指定marker之后的存储空间

以下代码用于列举examplebucket之后的存储空间。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定marker之后的存储空间。
// 填写marker。从marker之后按字母排序的第一个开始返回存储空间。
listBucketsRequest.setMarker("examplebucket");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
   System.out.println(" - " + bucket.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

列举指定个数的存储空间

以下代码用于最多列举500个存储空间。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 限定此次列举存储空间的最大个数为500。默认值为100，最大值为1000。
listBucketsRequest.setMaxKeys(500);
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
  System.out.println(" - " + bucket.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

以下代码用于判断存储空间examplebucket是否存在。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 判断存储空间examplebucket是否存在。如果返回值为true，则存储空间存在，否则存储空间不存在。
boolean exists = ossClient.doesBucketExist("examplebucket");
System.out.println(exists);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
String location = ossClient.getBucketLocation("<yourBucketName>");
System.out.println(location);
// 关闭OSSClient。
ossClient.shutdown();
```

有关地域的详细信息，请参见开发指南基本概念中的地域介绍以及GetBucketLocation API。

2.5.5. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期等。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 存储空间的信息包括地域（Region或Location）、创建日期（CreationDate）、拥有者（Owner）等。
// 填写Bucket名称，例如examplebucket。
BucketInfo info = ossClient.getBucketInfo("examplebucket");
// 获取地域。
info.getBucket().getLocation();
// 获取创建日期。
info.getBucket().getCreationDate();
// 获取拥有者信息。
info.getBucket().getOwner();
// 获取容灾类型。
info.getDataRedundancyType();
// 关闭OSSClient。
ossClient.shutdown();
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

2.5.6. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	CannedAccessControlList.Private
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	CannedAccessControlList.PublicRead
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	CannedAccessControlList.PublicReadWrite

以下代码用于设置存储空间的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置存储空间的访问权限为私有。
ossClient.setBucketAcl("<yourBucketName>", CannedAccessControlList.Private);
// 关闭OSSClient。
ossClient.shutdown();
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间访问权限

以下代码用于获取存储空间的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取存储空间的访问权限。
AccessControlList acl = ossClient.getBucketAcl("<yourBucketName>");
System.out.println(acl.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

2.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> 说明
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用Bucket.ListMultipartUploads列举出所有碎片，然后使用Bucket.AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除存储空间。
ossClient.deleteBucket("<yourBucketName>");
// 关闭OSSClient。
ossClient.shutdown();
```

删除存储空间的更多详情，请参见DeleteBucket。

2.5.8. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

> 说明
> - 只有Bucket的拥有者及授权子账户才能为Bucket设置用户标签，否则返回403 Forbidden错误，错误码：AccessDenied。
> - 最多可设置20对Bucket用户标签（Key-Value对）。
> - Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
> - Value最大长度为128字节，可以为空。
> - Key和Value必须为UTF-8编码。
> - PutBucketTagging是覆盖语义，即新设置的标签会完全覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置Bucket标签。
SetBucketTaggingRequest request = new SetBucketTaggingRequest("<yourBucketName>");
request.setTag("<yourTagkey1>", "<yourTagValue1>");
request.setTag("<yourTagkey2>", "<yourTagValue2>");
ossClient.setBucketTagging(request);
// 关闭OSSClient。
ossClient.shutdown();
```

获取Bucket标签

以下代码用于获取Bucket标签：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取Bucket标签信息。
TagSet tagSet = ossClient.getBucketTagging(new GenericRequest("<yourBucketName>"));
Map<String, String> tags = tagSet.getAllTags();
// 关闭OSSClient。
ossClient.shutdown();
```

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举带指定标签的Bucket。
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
listBucketsRequest.setTag("<yourTagKey>", "<yourTagValue>");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket o : bucketList.getBucketList()) {
 System.out.println("list result bucket: " + o.getName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

删除Bucket标签

以下代码用于删除Bucket标签：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除Bucket标签。
ossClient.deleteBucketTagging(new GenericRequest("<yourBucketName>"));
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.9. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **说明** 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

> **说明**
> - 单个Bucket最多只能有1000条清单配置。
> - 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单（Inventory）配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String destBucketName ="<yourDestinationBucketName>";
String accountId ="<yourDestinationBucketAccountId>";
String roleArn ="<yourDestinationBucketRoleArn>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建清单配置。
InventoryConfiguration inventoryConfiguration = new InventoryConfiguration();
// 设置清单配置id。
String inventoryId = "testid";
inventoryConfiguration.setInventoryId(id);
// 设置清单中包含的object属性。
List<String> fields = new ArrayList<String>();
fields.add(InventoryOptionalFields.Size);
fields.add(InventoryOptionalFields.LastModifiedDate);
fields.add(InventoryOptionalFields.IsMultipartUploaded);
fields.add(InventoryOptionalFields.StorageClass);
fields.add(InventoryOptionalFields.ETag);
fields.add(InventoryOptionalFields.EncryptionStatus);
inventoryConfiguration.setOptionalFields(fields);
// 设置清单的生成计划，以下示例为每周一次。其中，Weekly对应每周一次，Daily对应每天一次。
inventoryConfiguration.setSchedule(new InventorySchedule().withFrequency(InventoryFrequency.Weekly
));
// 设置清单中包含的object的版本为当前版本。如果设置为InventoryIncludedObjectVersions.All则表示object的所有版本，在版本控制状态下生效。
inventoryConfiguration.setIncludedObjectVersions(InventoryIncludedObjectVersions.Current);
// 清单配置是否启用的标识, true或false。
inventoryConfiguration.setEnabled(true);
// 设置清单筛选规则，指定筛选object的前缀。
InventoryFilter inventoryFilter = new InventoryFilter().withPrefix("obj-prefix");
inventoryConfiguration.setInventoryFilter(inventoryFilter);
// 创建清单的bucket目的地配置。
InventoryOSSBucketDestination ossInvDest = new InventoryOSSBucketDestination();
// 设置产生清单结果的存储路径前缀。
ossInvDest.setPrefix("destination-prefix");
// 设置清单格式。
ossInvDest.setFormat(InventoryFormat.CSV);
// 目的地bucket的用户accountId。
ossInvDest.setAccountId(accountId);
// 目的地bucket的roleArn。
ossInvDest.setRoleArn(roleArn);
// 目的地bucket的名称。
ossInvDest.setBucket(destBucketName);
// 如果需要使用KMS加密清单，请参考如下设置。
// InventoryEncryption inventoryEncryption = new InventoryEncryption();
```

```
// InventoryServerSideEncryptionKMS serverSideKmsEncryption = new InventoryServerSideEncryptionKMS
().withKeyId("test-kms-id");
// inventoryEncryption.setServerSideKmsEncryption(serverSideKmsEncryption);
// ossInvDest.setEncryption(inventoryEncryption);
// 如果需要使用OSS服务端加密清单，请参考如下设置。
// InventoryEncryption inventoryEncryption = new InventoryEncryption();
// inventoryEncryption.setServerSideOssEncryption(new InventoryServerSideEncryptionOSS());
// ossInvDest.setEncryption(inventoryEncryption);
// 设置清单的目的地。
InventoryDestination destination = new InventoryDestination();
destination.setOssBucketDestination(ossInvDest);
inventoryConfiguration.setDestination(destination);
// 上传清单配置。
ossClient.setBucketInventoryConfiguration(bucketName, inventoryConfiguration);
// 关闭ossClient。
ossClient.shutdown();
```

有关添加存储空间清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String inventoryId = "<yourInventoryConfigurationId>";
// 查看指定id的清单配置信息。
GetBucketInventoryConfigurationRequest request = new GetBucketInventoryConfigurationRequest(bucketName, inventoryId);
GetBucketInventoryConfigurationResult getResult = ossClient.getBucketInventoryConfiguration(request);
# 打印清单配置信息。
InventoryConfiguration config = getResult.getInventoryConfiguration();
System.out.println("=====Inventory configuration=====");
System.out.println("inventoryId:" + config.getInventoryId());
System.out.println("isenabled:" + config.isEnabled());
System.out.println("includedVersions:" + config.getIncludedObjectVersions());
System.out.println("schdule:" + config.getSchedule().getFrequency());
if (config.getInventoryFilter().getPrefix() != null) {
  System.out.println("filter, prefix:" + config.getInventoryFilter().getPrefix());
}
List<String> fields = config.getOptionalFields();
for (String field : fields) {
  System.out.println("field:" + field);
}
System.out.println("===bucket destination config===");
InventoryOSSBucketDestination destin = config.getDestination().getOssBucketDestination();
System.out.println("format:" + destin.getFormat());
System.out.println("bucket:" + destin.getBucket());
System.out.println("prefix:" + destin.getPrefix());
System.out.println("accountId:" + destin.getAccountId());
System.out.println("roleArn:" + destin.getRoleArn());
if (destin.getEncryption() != null) {
  if (destin.getEncryption().getServerSideKmsEncryption() != null) {
    System.out.println("server-side kms encryption, key id:" + destin.getEncryption().getServerSideKmsEncryption().getKeyId());
  } else if (destin.getEncryption().getServerSideOssEncryption() != null) {
    System.out.println("server-side oss encryption.");
  }
}
// 关闭ossClient。
ossClient.shutdown();
```

有关查看清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明** 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举清单配置项。默认每次最多列举100条结果，如果配置超过100条，结果将会分页返回，通过传入Token方式列举下一页。
String continuationToken = null;
while (true) {
  ListBucketInventoryConfigurationsRequest listRequest = new ListBucketInventoryConfigurationsRequest(bucketName, continuationToken);
  ListBucketInventoryConfigurationsResult result = ossClient.listBucketInventoryConfigurations(listRequest);
  System.out.println("=======List bucket inventory configuration=======");
  System.out.println("istruncated:" + result.isTruncated());
  System.out.println("continuationToken:" + result.getContinuationToken());
  System.out.println("nextContinuationToken:" + result.getNextContinuationToken());
  System.out.println("list size :" + result.getInventoryConfigurationList
  if (result.getInventoryConfigurationList() != null && !result.getInventoryConfigurationList().isEmpty()) {
    for (InventoryConfiguration config : result.getInventoryConfigurationList()) {
      System.out.println("===Inventory configuration===");
      System.out.println("inventoryId:" + config.getInventoryId());
      System.out.println("isenabled:" + config.isEnabled());
      System.out.println("includedVersions:" + config.getIncludedObjectVersions());
      System.out.println("schdule:" + config.getSchedule().getFrequency());
      if (config.getInventoryFilter().getPrefix() != null) {
        System.out.println("filter, prefix:" + config.getInventoryFilter().getPrefix());
      }
      List<String> fields = config.getOptionalFields();
      for (String field : fields) {
        System.out.println("field:" + field);
      }
      System.out.println("===bucket destination config===");
      InventoryOSSBucketDestination destin = config.getDestination().getOssBucketDestination();
      System.out.println("format:" + destin.getFormat());
      System.out.println("bucket:" + destin.getBucket());
      System.out.println("prefix:" + destin.getPrefix());
      System.out.println("accountId:" + destin.getAccountId());
      System.out.println("roleArn:" + destin.getRoleArn());
      if (destin.getEncryption() != null) {
        if (destin.getEncryption().getServerSideKmsEncryption() != null) {
          System.out.println("server-side kms encryption key id:" + destin.getEncryption().getServerSideKmsEncryption().getKeyId());
        } else if (destin.getEncryption().getServerSideOssEncryption() != null) {
          System.out.println("server-side oss encryption.");
        }
      }
    }
  if (result.isTruncated()) {
    continuationToken = result.getNextContinuationToken();
```

```
    } else {
      break;
    }
  }
  // 关闭ossClient。
  ossClient.shutdown();
```

有关批量列举清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String inventoryId = "<yourInventoryConfigurationId>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除指定id的清单配置。
ossClient.deleteBucketInventoryConfiguration(bucketName, inventoryId);
// 关闭ossClient。
ossClient.shutdown();
```

有关删除存储空间清单配置的详情，请参见DeleteBucketInventory。

2.5.10. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 填写policyText。
String policyText = "{\"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"oss:GetObject\", \"oss:ListObjects\"], \"Resource\": [\"acs:oss:*:*:*/user1/*\"]}], \"Version\": \"1\"}";
// 设置授权策略。
ossClient.setBucketPolicy(bucketName, policyText);
// 关闭ossClient。
ossClient.shutdown();
```

有关设置Bucket Policy详情，请参见Put Bucket Policy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取授权策略。
GetBucketPolicyResult result = ossClient.getBucketPolicy(bucketName);
System.out.println("policyText:"+ result.getPolicyText());
// 关闭ossClient。
ossClient.shutdown();
```

有关获取Bucket Policy信息详情，请参见Get Bucket Policy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除授权策略。
ossClient.deleteBucketPolicy(bucketName);
// 关闭ossClient。
ossClient.shutdown();
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

2.5.11. 合规保留策略

对象存储OSS允许针对存储空间（Bucket）设置基于时间的合规保留策略，保护周期为1天到70年。本文介绍如何新建、获取、锁定合规保留策略等。

背景信息

OSS支持WORM（Write Once Read Many）特性，允许您以不可删除、不可篡改的方式保存和使用数据。

当基于时间的合规保留策略创建24小时后未提交锁定，则该策略自动失效。当合规保留策略锁定后，您可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，不允许删除Object及合规保留策略，且无法缩短策略保护周期，仅允许延长Object的保留时间。有关合规保留策略的详情，请参见合规保留策略。

新建合规保留策略

以下代码用于新建合规保留策略：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建InitiateBucketWormRequest对象。
InitiateBucketWormRequest initiateBucketWormRequest = new InitiateBucketWormRequest(bucketName);
// 指定Object保护天数为1天。
initiateBucketWormRequest.setRetentionPeriodInDays(1);
// 创建合规保留策略。
InitiateBucketWormResult initiateBucketWormResult = ossClient.initiateBucketWorm(initiateBucketWormRequest);
// 查看合规保留策略Id。
String wormId = initiateBucketWormResult.getWormId();
System.out.println(wormId);
// 关闭ossClient。
ossClient.shutdown();
```

有关新建合规保留策略的详情，请参见InitiateBucketWorm。

取消未锁定的合规保留策略

以下代码用于取消未锁定的合规保留策略：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 取消合规保留策略。
ossClient.abortBucketWorm(bucketName);
// 关闭ossClient。
ossClient.shutdown();
```

有关取消未锁定的合规保留策略的详情，请参见AbortBucketWorm。

锁定合规保留策略

以下代码用于锁定合规保留策略：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String wormId = "<yourWormId>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 锁定合规保留策略。
ossClient.completeBucketWorm(bucketName, wormId);
// 关闭ossClient。
ossClient.shutdown();
```

有关锁定合规保留策略的详情，请参见CompleteBucketWorm。

获取合规保留策略

以下代码用于获取合规保留策略：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取合规保留策略。
GetBucketWormResult getBucketWormResult = ossClient.getBucketWorm(bucketName);
// 查看合规保留策略Id。
System.out.println(getBucketWormResult.getWormId());
// 查看合规保留策略状态。未锁定状态下为"InProgress", 锁定状态下为"Locked"。
System.out.println(getBucketWormResult.getWormState());
// 查看Object的保护时间。
System.out.println(getBucketWormResult.getRetentionPeriodInDays());
// 查看合规保留策略的创建时间。
System.out.println(getBucketWormResult.getCreationDate());
// 关闭ossClient。
ossClient.shutdown();
```

有关获取合规保留策略的详情，请参见GetBucketWorm。

延长Object的保留天数

以下代码用于延长已锁定的合规保留策略中Object的保留天数：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String wormId = "<yourWormId>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 延长已锁定的合规保留策略中Object的保留天数。
ossClient.extendBucketWorm(bucketName, wormId, 2);
// 关闭ossClient。
ossClient.shutdown();
```

有关延长已锁定的合规保留策略中Object的保留天数详情，请参见ExtendBucketWorm。

2.5.12. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

生命周期规则包含如下信息：

- 前缀或标签匹配策略：生命周期规则匹配的Object和碎片。
  - 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。
  - 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，OSS对所有拥有这些标签的对象执行生命周期规则。标签匹配不可以作用于碎片。

    > **说明** 对象标签功能详情请参见对象标签。

  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。
  - 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。
- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Object成为非当前版本天数：指定一个过期天数N，并指定非当前版本Object过期后执行什么操作。Object会在其成为非当前版本的N天后过期，并执行指定的操作。

    > **说明** 您可以将过期Object转换为低频访问类型或归档类型，也可以选择删除过期Object。详情请参见生命周期配置元素。

- 碎片过期策略：设置碎片的过期时间及操作。
  - 过期天数：可指定一个过期天数N，碎片会在其最后修改时间的N天后被删除。

- 过期日期：指定一个过期日期，最后修改时间在该日期之前的碎片会被全部删除。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建SetBucketLifecycleRequest。
SetBucketLifecycleRequest request = new SetBucketLifecycleRequest(bucketName);
// 设置规则ID、文件匹配前缀与标签。
String ruleId0 = "rule0";
String matchPrefix0 = "A0/";
Map<String, String> matchTags0 = new HashMap<String, String>();
matchTags0.put("key0", "value0");
String ruleId1 = "rule1";
String matchPrefix1 = "A1/";
Map<String, String> matchTags1 = new HashMap<String, String>();
matchTags1.put("key1", "value1");
String ruleId2 = "rule2";
String matchPrefix2 = "A2/";
String ruleId3 = "rule3";
String matchPrefix3 = "A3/";
String ruleId4 = "rule4";
String matchPrefix4 = "A4/";
String ruleId5 = "rule5";
String matchPrefix5 = "A5/";
String ruleId6 = "rule6";
String matchPrefix6 = "A6/";
// 距最后修改时间3天后过期。
LifecycleRule rule = new LifecycleRule(ruleId0, matchPrefix0, LifecycleRule.RuleStatus.Enabled, 3);
rule.setTags(matchTags0);
request.AddLifecycleRule(rule);
// 指定日期之前创建的文件过期。
rule = new LifecycleRule(ruleId1, matchPrefix1, LifecycleRule.RuleStatus.Enabled);
rule.setCreatedBeforeDate(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
rule.setTags(matchTags1);
request.AddLifecycleRule(rule);
// 分片3天后过期。
rule = new LifecycleRule(ruleId2, matchPrefix2, LifecycleRule.RuleStatus.Enabled);
LifecycleRule.AbortMultipartUpload abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setExpirationDays(3);
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);
```

```
request.AddLifecycleRule(rule);
// 指定日期之前的分片过期。
rule = new LifecycleRule(ruleId3, matchPrefix3, LifecycleRule.RuleStatus.Enabled);
abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setCreatedBeforeDate(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);
// 距最后修改时间10天后转低频访问存储类型，距最后修改时间30天后转归档存储类型。
rule = new LifecycleRule(ruleId4, matchPrefix4, LifecycleRule.RuleStatus.Enabled);
List<StorageTransition> storageTransitions = new ArrayList<StorageTransition>();
StorageTransition storageTransition = new StorageTransition();
storageTransition.setStorageClass(StorageClass.IA);
storageTransition.setExpirationDays(10);
storageTransitions.add(storageTransition);
storageTransition = new StorageTransition();
storageTransition.setStorageClass(StorageClass.Archive);
storageTransition.setExpirationDays(30);
storageTransitions.add(storageTransition);
rule.setStorageTransition(storageTransitions);
request.AddLifecycleRule(rule);
// 指定最后修改日期在2022-10-12之前的文件转为归档存储。
rule = new LifecycleRule(ruleId5, matchPrefix5, LifecycleRule.RuleStatus.Enabled);
storageTransitions = new ArrayList<StorageTransition>();
storageTransition = new StorageTransition();
storageTransition.setCreatedBeforeDate(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
storageTransition.setStorageClass(StorageClass.Archive);
storageTransitions.add(storageTransition);
rule.setStorageTransition(storageTransitions);
request.AddLifecycleRule(rule);
// rule6针对版本控制状态下的Bucket。
rule = new LifecycleRule(ruleId6, matchPrefix6, LifecycleRule.RuleStatus.Enabled);
// 设置Object相对最后修改时间365天之后自动转为归档文件。
storageTransitions = new ArrayList<StorageTransition>();
storageTransition = new StorageTransition();
storageTransition.setStorageClass(StorageClass.Archive);
storageTransition.setExpirationDays(365);
storageTransitions.add(storageTransition);
rule.setStorageTransition(storageTransitions);
// 设置自动移除过期删除标记。
rule.setExpiredDeleteMarker(true);
// 设置非当前版本的object距最后修改时间10天之后转为低频访问类型。
NoncurrentVersionStorageTransition noncurrentVersionStorageTransition =
    new NoncurrentVersionStorageTransition().withNoncurrentDays(10).withStrorageClass(StorageClass.IA
);
// 设置非当前版本的Object距最后修改时间20天之后转为归档类型。
NoncurrentVersionStorageTransition noncurrentVersionStorageTransition2 =
    new NoncurrentVersionStorageTransition().withNoncurrentDays(20).withStrorageClass(StorageClass.A
rchive);
// 设置非当前版本Object 30天后删除。
NoncurrentVersionExpiration noncurrentVersionExpiration = new NoncurrentVersionExpiration().withNonc
urrentDays(30);
List<NoncurrentVersionStorageTransition> noncurrentVersionStorageTransitions = new ArrayList<Noncurr
entVersionStorageTransition>();
noncurrentVersionStorageTransitions.add(noncurrentVersionStorageTransition2);
rule.setStorageTransition(storageTransitions);
```

```
rule.setNoncurrentVersionExpiration(noncurrentVersionExpiration);
rule.setNoncurrentVersionStorageTransitions(noncurrentVersionStorageTransitions);
request.AddLifecycleRule(rule);
// 发起设置生命周期规则请求。
ossClient.setBucketLifecycle(request);
// 关闭OSSClient。
ossClient.shutdown();
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取生命周期规则。
List<LifecycleRule> rules = ossClient.getBucketLifecycle(bucketName);
// 查看生命周期规则。
for (LifecycleRule r : rules) {
  System.out.println("================");
  // 查看规则id。
  System.out.println("rule id: " + r.getId());
  // 查看规则状态。
  System.out.println("rule status: " + r.getStatus());
  // 查看规则前缀。
  System.out.println("rule prefix: " + r.getPrefix());
  // 查看规则标签。
  if (r.hasTags()) {
    System.out.println("rule tagging: "+ r.getTags().toString());
  }
  // 查看过期天数规则。
  if (r.hasExpirationDays()) {
    System.out.println("rule expiration days: " + r.getExpirationDays());
  }
  // 查看过期日期规则。
  if (r.hasCreatedBeforeDate()) {
    System.out.println("rule expiration create before days: " + r.getCreatedBeforeDate());
  }
  // 查看过期分片规则。
  if(r.hasAbortMultipartUpload()) {
    if(r.getAbortMultipartUpload().hasExpirationDays()) {
      System.out.println("rule abort uppart days: " + r.getAbortMultipartUpload().getExpirationDays());
    }
    if (r.getAbortMultipartUpload().hasCreatedBeforeDate()) {
      System.out.println("rule abort uppart create before date: " + r.getAbortMultipartUpload().getCreatedB
eforeDate());
    }
  }
```

```
    }
    // 查看存储类型转换规则。
    if (r.getStorageTransition().size() > 0) {
      for (StorageTransition transition : r.getStorageTransition()) {
        if (transition.hasExpirationDays()) {
          System.out.println("rule storage trans days: " + transition.getExpirationDays() +
              " trans storage class: " + transition.getStorageClass());
        }
        if (transition.hasCreatedBeforeDate()) {
          System.out.println("rule storage trans before create date: " + transition.getCreatedBeforeDate());
        }
      }
    }
    // 查看是否自动删除过期删除标记。
    if (r.hasExpiredDeleteMarker()) {
      System.out.println("rule expired delete marker: " + r.getExpiredDeleteMarker());
    }
    // 查看非当前版本Object存储类型转换规则。
    if (r.hasNoncurrentVersionStorageTransitions()) {
      for (NoncurrentVersionStorageTransition transition : r.getNoncurrentVersionStorageTransitions()) {
        System.out.println("rule noncurrent versions trans days:" + transition.getNoncurrentDays() +
            " trans storage class: " + transition.getStorageClass());
      }
    }
    // 查看非当前版本Object过期规则。
    if (r.hasNoncurrentVersionExpiration()) {
      System.out.println("rule noncurrent versions expiration days:" + r.getNoncurrentVersionExpiration().ge
tNoncurrentDays());
    }
  }
  // 关闭OSSClient。
  ossClient.shutdown();
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ossClient.deleteBucketLifecycle(bucketName);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.13. 请求者付费模式

请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

  如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

  数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置请求者付费模式。
Payer payer = Payer.Requester;
ossClient.setBucketRequestPayment(bucketName, payer);
// 关闭OSSClient。
ossClient.shutdown();
```

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取请求者付费模式。
GetBucketRequestPaymentResult result = ossClient.getBucketRequestPayment(bucketName);
System.out.println("Payer:" + result.getPayer());
// 关闭OSSClient。
ossClient.shutdown();
```

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、GetObject和DeleteObject为例，用于指定第三方付费访问Object。

其他用于指定第三方付费的Object读写操作接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
Payer payer = Payer.Requester;
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// putObject接口指定付费者。
String content = "hello";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName, new ByteArrayInp
utStream(content.getBytes()));
putObjectRequest.setRequestPayer(payer);
ossClient.putObject(putObjectRequest);
// getObject接口指定付费者。
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
getObjectRequest.setRequestPayer(payer);
OSSObject ossObject = ossClient.getObject(getObjectRequest);
ossObject.close();
// deleteObject接口指定付费者。
GenericRequest genericRequest = new GenericRequest(bucketName, objectName);
genericRequest.setRequestPayer(payer);
ossClient.deleteObject(genericRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.14. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链

以下代码用于设置防盗链：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
List<String> refererList = new ArrayList<String>();
// 添加Referer白名单。Referer参数支持通配符星号（*）和问号（？）。
refererList.add("http://www.aliyun.com");
refererList.add("http://www.*.com");
refererList.add("http://www.?.aliyuncs.com");
// 设置存储空间Referer列表。设为true表示Referer字段允许为空。
BucketReferer br = new BucketReferer(true, refererList);
ossClient.setBucketReferer(bucketName, br);
// 关闭OSSClient。
ossClient.shutdown();
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取存储空间Referer白名单列表。
BucketReferer br = ossClient.getBucketReferer(bucketName);
List<String> refererList = br.getRefererList();
for (String referer : refererList) {
   System.out.println(referer);
}
// 关闭OSSClient。
ossClient.shutdown();
```

清空防盗链

以下代码用于清空防盗链：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
BucketReferer br = new BucketReferer();
ossClient.setBucketReferer(bucketName, br);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.15. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

关于访问日志的更多信息，请参见开发指南中的日志转存。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
SetBucketLoggingRequest request = new SetBucketLoggingRequest("<yourSourceBucketName>");
// 设置存放日志文件的存储空间。
request.setTargetBucket("<yourTargetBucketName>");
// 设置日志文件存放的目录。
request.setTargetPrefix("<yourTargetPrefix>");
ossClient.setBucketLogging(request);
// 关闭OSSClient。
ossClient.shutdown();
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
BucketLoggingResult result = ossClient.getBucketLogging("<yourSourceBucketName>");
System.out.println(result.getTargetBucket());
System.out.println(result.getTargetPrefix());
// 关闭OSSClient。
ossClient.shutdown();
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
SetBucketLoggingRequest request = new SetBucketLoggingRequest("<yourSourceBucketName>");
request.setTargetBucket(null);
request.setTargetPrefix(null);
ossClient.setBucketLogging(request);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.16. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
SetBucketWebsiteRequest request = new SetBucketWebsiteRequest("<yourBucketName>");
request.setIndexDocument("index.html");
request.setErrorDocument("error.html");
ossClient.setBucketWebsite(request);
// 关闭OSSClient。
ossClient.shutdown();
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
BucketWebsiteResult result = ossClient.getBucketWebsite("<yourBucketName>");
System.out.println(result.getIndexDocument());
System.out.println(result.getErrorDocument());
// 关闭OSSClient。
ossClient.shutdown();
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ossClient.deleteBucketWebsite("<yourBucketName>");
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.17. 跨区域复制

跨区域复制是在不同OSS地域之间自动、异步复制文件，将源存储空间中文件的改动（新建、覆盖、删除操作）同步到目标存储空间中。该功能用于满足异地容灾和数据复制的需求。

更多跨区域复制的内容请参见开发指南中的管理跨区域复制。

开启跨区域复制

以下代码用于开启跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
AddBucketReplicationRequest request = new AddBucketReplicationRequest(bucketName);
request.setReplicationRuleID("<yourRuleId>");
request.setTargetBucketName("<yourTargetBucketName>");
// 目标Endpoint以北京为例。
request.setTargetBucketLocation("oss-cn-beijing");
// 设置禁止同步历史数据。默认会同步历史数据。
request.setEnableHistoricalObjectReplication(false);
ossClient.addBucketReplication(request);
// 关闭OSSClient。
ossClient.shutdown();
```

查看跨区域复制

以下代码用于查看存储空间已开启的跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
List<ReplicationRule> rules = ossClient.getBucketReplication(bucketName);
for (ReplicationRule rule : rules) {
  System.out.println(rule.getReplicationRuleID());
  System.out.println(rule.getTargetBucketLocation());
  System.out.println(rule.getTargetBucketName());
}
// 关闭OSSClient。
ossClient.shutdown();
```

查看跨区域复制进度

跨区域复制进度分为历史数据同步进度和实时数据同步进度。

- 历史数据同步进度用百分比表示，仅对开启了历史数据同步的存储空间有效。
- 实时数据同步进度用新写入数据的时间点表示，代表这个时间点之前的数据已同步完成。

以下代码用于查看跨区域复制进度：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
BucketReplicationProgress process = ossClient.getBucketReplicationProgress(bucketName, "<yourRuleId>")
;
System.out.println(process.getReplicationRuleID());
// 是否开启了历史数据同步。
System.out.println(process.isEnableHistoricalObjectReplication());
// 历史数据同步进度。
System.out.println(process.getHistoricalObjectProgress());
// 实时数据同步进度。
System.out.println(process.getNewObjectProgress());
// 关闭OSSClient。
ossClient.shutdown();
```

查看可同步的目标地域

以下代码用于获取存储空间所能同步到的地域列表：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
List<String> locations = ossClient.getBucketReplicationLocation(bucketName);
for (String loc : locations) {
  System.out.println(loc);
}
// 关闭OSSClient。
ossClient.shutdown();
```

关闭跨区域复制

以下代码用于关闭已开启的跨区域复制：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 关闭跨区域复制。关闭后目标存储空间内的文件依然存在，只是不再同步源存储空间内文件的所有改动。
ossClient.deleteBucketReplication(bucketName, "<yourRuleId>");
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.18. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
SetBucketCORSRequest request = new SetBucketCORSRequest(bucketName);
// 跨域资源共享规则的容器，每个存储空间最多允许10条规则。
ArrayList<CORSRule> putCorsRules = new ArrayList<CORSRule>();
CORSRule corRule = new CORSRule();
ArrayList<String> allowedOrigin = new ArrayList<String>();
// 指定允许跨域请求的来源。
allowedOrigin.add( "http://www.b.com");
ArrayList<String> allowedMethod = new ArrayList<String>();
// 指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)。
allowedMethod.add("GET");
ArrayList<String> allowedHeader = new ArrayList<String>();
// 是否允许预取指令（OPTIONS）中Access-Control-Request-Headers头中指定的Header。
allowedHeader.add("x-oss-test");
ArrayList<String> exposedHeader = new ArrayList<String>();
// 指定允许用户从应用程序中访问的响应头。
exposedHeader.add("x-oss-test1");
// AllowedOrigins和AllowedMethods最多支持一个星号（*）通配符。星号（*）表示允许所有的域来源或者操作。
corRule.setAllowedMethods(allowedMethod);
corRule.setAllowedOrigins(allowedOrigin);
// AllowedHeaders和ExposeHeaders不支持通配符。
corRule.setAllowedHeaders(allowedHeader);
corRule.setExposeHeaders(exposedHeader);
// 指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，单位为秒。
corRule.setMaxAgeSeconds(10);
// 最多允许10条规则。
putCorsRules.add(corRule);
// 已存在的规则将被覆盖。
request.setCorsRules(putCorsRules);
ossClient.setBucketCORS(request);
// 关闭OSSClient。
ossClient.shutdown();
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ArrayList<CORSRule> corsRules;
// 获取跨域资源共享规则列表。
corsRules =  (ArrayList<CORSRule>) ossClient.getBucketCORSRules(bucketName);
for (CORSRule rule : corsRules) {
  for (String allowedOrigin1 : rule.getAllowedOrigins()) {
    // 获取允许的跨域请求源。
    System.out.println(allowedOrigin1);
  }
  for (String allowedMethod1 : rule.getAllowedMethods()) {
    // 获取允许的跨域请求方法。
    System.out.println(allowedMethod1);
  }
  if (rule.getAllowedHeaders().size() > 0){
    for (String allowedHeader1 : rule.getAllowedHeaders()) {
      // 获取允许的头部列表。
      System.out.println(allowedHeader1);
    }
  }
  if (rule.getExposeHeaders().size() > 0) {
    for (String exposeHeader : rule.getExposeHeaders()) {
      // 获取允许的头部。
      System.out.println(exposeHeader);
    }
  }
  if ( null != rule.getMaxAgeSeconds()) {
    System.out.println(rule.getMaxAgeSeconds());
  }
}
// 关闭OSSClient。
ossClient.shutdown();
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除存储空间的跨域资源共享规则。
ossClient.deleteBucketCORSRules(bucketName);
// 关闭OSSClient。
ossClient.shutdown();
```

2.5.19. 传输加速

传输加速可提升全球各地用户对OSS的访问速度，适用于远距离数据传输、GB或TB级大文件上传和下载的场景。

关于传输加速的更多信息，请参见开发指南中的传输加速。

开启传输加速

以下代码用于开启目标存储空间examplebucket的传输加速功能。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置Bucket的传输加速状态。
// 当设置enabled为true时，表示开启传输加速；当设置enabled为false时，表示关闭传输加速。
boolean enabled = true;
ossClient.setBucketTransferAcceleration(bucketName, enabled);
// 关闭ossClient。
ossClient.shutdown();
```

开启传输加速功能的接口信息，请参见API参考中的PutBucketTransferAcceleration。

查询传输加速状态

以下代码用于查询目标存储空间examplebucket的传输加速状态。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 查询Bucket的传输加速状态。
// 如果返回值为true，则Bucket已开启传输加速功能；如果返回值为false，则Bucket的传输加速功能为关闭状态。
TransferAcceleration  result = ossClient.getBucketTransferAcceleration(bucketName);
System.out.println("Is transfer acceleration enabled:"+ result.isEnabled());
// 关闭ossClient。
ossClient.shutdown();
```

查询传输加速状态的接口信息，请参见API参考中的GetBucketTransferAcceleration。

2.6. 上传文件

2.6.1. 概述

本文介绍对象存储OSS Java SDK的多种文件上传方式。

在OSS中，操作的基本数据单元是文件（Object）。OSS Java SDK提供了以下几种文件上传方式：

- 简单上传：包括流式上传和文件上传。最大不能超过5GB。
- 表单上传：最大不能超过5GB。
- 追加上传：最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

2.6.2. 简单上传

简单上传是指通过PutObject方法上传单个文件（Object）。简单上传包括流式上传和文件上传，流式上传使用InputStream作为OSS文件的数据源，文件上传使用本地文件作为OSS文件的数据源。本文介绍如何使用流式上传和文件上传方式上传文件。

说明

- 有关简单上传的完整代码，请参见GitHub。
- 有关上传文件的详细介绍，请参见开发指南中的简单上传。

流式上传

使用流式上传，您可以将数据流上传到OSS文件。

- 上传字符串

  以下代码用于将字符串上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 填写字符串。
String content = "Hello OSS";
// 创建PutObjectRequest对象。
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
PutObjectRequest putObjectRequest = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", new ByteArrayInputStream(content.getBytes()));
// 如果需要上传时设置存储类型和访问权限，请参考以下示例代码。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setHeader(OSSHeaders.OSS_STORAGE_CLASS, StorageClass.Standard.toString());
// metadata.setObjectAcl(CannedAccessControlList.Private);
// putObjectRequest.setMetadata(metadata);
// 上传字符串。
ossClient.putObject(putObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

- 上传Byte数组

  以下代码用于将Byte数组上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,accessKeySecret);
// 填写Byte数组。
byte[] content = "Hello OSS".getBytes();
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
ossClient.putObject("examplebucket", "exampledir/exampleobject.txt", new ByteArrayInputStream(content));
// 关闭OSSClient。
ossClient.shutdown();
```

- 上传网络流

  以下代码用于将网络流上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 填写网络流地址。
InputStream inputStream = new URL("https://www.aliyun.com/").openStream();
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
ossClient.putObject("examplebucket", "exampledir/exampleobject.txt", inputStream);
// 关闭OSSClient。
ossClient.shutdown();
```

- 上传文件流

  以下代码用于将文件流上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件流。
InputStream inputStream = new FileInputStream("D:\\localpath\\examplefile.txt");
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
ossClient.putObject("examplebucket", "exampledir/exampleobject.txt", inputStream);
// 关闭OSSClient。
ossClient.shutdown();
```

文件上传

使用文件上传，您可以将本地文件上传到OSS文件。

以下代码用于将本地文件examplefile.txt上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建PutObjectRequest对象。
// 依次填写Bucket名称（例如examplebucket）、Object完整路径（例如exampledir/exampleobject.txt）和本地文件的完整路径。Object完整路径中不能包含Bucket名称。
// 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
PutObjectRequest putObjectRequest = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", new File("D:\\localpath\\examplefile.txt"));
// 如果需要上传时设置存储类型和访问权限，请参考以下示例代码。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setHeader(OSSHeaders.OSS_STORAGE_CLASS, StorageClass.Standard.toString());
// metadata.setObjectAcl(CannedAccessControlList.Private);
// putObjectRequest.setMetadata(metadata);
// 上传文件。
ossClient.putObject(putObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

常见问题

- 上传Object后如何获取访问URL?
- OSS如何限制上传文件类型及大小?

- OSS怎样上传下载文件夹（目录）？

2.6.3. 表单上传

表单上传是使用HTML表单形式上传文件（Object）到指定存储空间（Bucket）中，上传文件最大不能超过5GB。

以下代码用于将本地路径D:\localpath下的examplefile.txt，通过表单方式上传到指定存储空间examplebucket中的test\examplefile.txt：

```
public class PostObjectSample {
  // 上传文件。
  // 填写文档上传的本地路径。
  private String localFilePath = "D:\localpath\examplefile.txt";
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  private String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  private String accessKeyId = "<yourAccessKeyId>";
  private String accessKeySecret = "<yourAccessKeySecret>";
  // 设置存储空间名称。
  private String bucketName = "examplebucket";
  // 填写Object完整路径，完整路径中不能包含Bucket名称。
  private String objectName = "test\examplefile.txt";
  /**
   * 表单上传。
   * @throws Exception
   */
  private void PostObject() throws Exception {
    // 在URL中添加存储空间名称，添加后URL如下：http://yourBucketName.oss-cn-hangzhou.aliyuncs.com。
    String urlStr = endpoint.replace("http://", "http://examplebucket.");
    // 设置表单Map。
    Map<String, String> formFields = new LinkedHashMap<String, String>();
    // 设置文件名称。
    formFields.put("key", this.objectName);
    // 设置Content-Disposition。
    formFields.put("Content-Disposition", "attachment;filename="
        D:\localpath\examplefile.txt);
    // 设置回调参数。
    Callback callback = new Callback();
    // 设置回调服务器地址，如http://oss-demo.aliyuncs.com:23450或http://127.0.0.1:9090。
    callback.setCallbackUrl("<yourCallbackServerUrl>");
    // 设置回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
    callback.setCallbackHost("<yourCallbackServerHost>");
    // 设置发起回调时请求body的值。
    callback.setCallbackBody("{\\\"mimeType\\\":${mimeType},\\\"size\\\":${size}}");
    // 设置发起回调请求的Content-Type。
    callback.setCalbackBodyType(CalbackBodyType.JSON);
    // 设置发起回调请求的自定义参数，由Key和Value组成，Key必须以x:开始，且必须小写。
    callback.addCallbackVar("x:var1", "value1");
    callback.addCallbackVar("x:var2", "value2");
    // 在表单Map中设置回调参数。
    setCallBack(formFields, callback);
    // 设置OSSAccessKeyId。
```

```
    formFields.put("OSSAccessKeyId", accessKeyId);
    String policy = "{\"expiration\": \"2120-01-01T12:00:00.000Z\",\"conditions\": [[\"content-length-range\",
0, 104857600]]}";
    String encodePolicy = new String(Base64.encodeBase64(policy.getBytes()));
    // 设置policy。
    formFields.put("policy", encodePolicy);
    // 生成签名。
    String signaturecom = com.aliyun.oss.common.auth.ServiceSignature.create().computeSignature(acces
sKeySecret, encodePolicy);
    // 设置签名。
    formFields.put("Signature", signaturecom);
    String ret = formUpload(urlStr, formFields, localFilePath);
    System.out.println("Post Object ["test\examplefile.txt"] to bucket ["examplebucket"]");
    System.out.println("post reponse:" + ret);
  }
  private static String formUpload(String urlStr, Map<String, String> formFields, String localFile)
      throws Exception {
    String res = "";
    HttpURLConnection conn = null;
    String boundary = "9431149156168";
    try {
      URL url = new URL(urlStr);
      conn = (HttpURLConnection) url.openConnection();
      conn.setConnectTimeout(5000);
      conn.setReadTimeout(30000);
      conn.setDoOutput(true);
      conn.setDoInput(true);
      conn.setRequestMethod("POST");
      conn.setRequestProperty("User-Agent",
          "Mozilla/5.0 (Windows; U; Windows NT 6.1; zh-CN; rv:1.9.2.6)");
      // 设置MD5值。MD5值由整个body计算得出。
      conn.setRequestProperty("Content-MD5", "<yourContentMD5>");
      conn.setRequestProperty("Content-Type",
          "multipart/form-data; boundary=" + boundary);
      OutputStream out = new DataOutputStream(conn.getOutputStream());
      // 遍历读取表单Map中的数据，将数据写入到输出流中。
      if (formFields != null) {
        StringBuffer strBuf = new StringBuffer();
        Iterator<Entry<String, String>> iter = formFields.entrySet().iterator();
        int i = 0;
        while (iter.hasNext()) {
          Entry<String, String> entry = iter.next();
          String inputName = entry.getKey();
          String inputValue = entry.getValue();
          if (inputValue == null) {
            continue;
          }
          if (i == 0) {
            strBuf.append("--").append(boundary).append("\r\n");
            strBuf.append("Content-Disposition: form-data; name=\""
                + inputName + "\"\r\n\r\n");
            strBuf.append(inputValue);
          } else {
            strBuf.append("\r\n").append("--").append(boundary).append("\r\n");
            strBuf.append("Content-Disposition: form-data; name=\""
```

```
                strBuf.append("Content-Disposition: form-data; name=\"
                    + inputName + "\"\r\n\r\n");
                strBuf.append(inputValue);
            }
            i++;
        }
        out.write(strBuf.toString().getBytes());
    }
    // 读取文件信息，将要上传的文件写入到输出流中。
    File file = new File(localFile);
    String filename = file.getName();
    String contentType = new MimetypesFileTypeMap().getContentType(file);
    if (contentType == null || contentType.equals("")) {
        contentType = "application/octet-stream";
    }
    StringBuffer strBuf = new StringBuffer();
    strBuf.append("\r\n").append("--").append(boundary)
            .append("\r\n");
    strBuf.append("Content-Disposition: form-data; name=\"file\"; "
            + "filename=\"" + filename + "\"\r\n");
    strBuf.append("Content-Type: " + contentType + "\r\n\r\n");
    out.write(strBuf.toString().getBytes());
    DataInputStream in = new DataInputStream(new FileInputStream(file));
    int bytes = 0;
    byte[] bufferOut = new byte[1024];
    while ((bytes = in.read(bufferOut)) != -1) {
        out.write(bufferOut, 0, bytes);
    }
    in.close();
    byte[] endData = ("\r\n--" + boundary + "--\r\n").getBytes();
    out.write(endData);
    out.flush();
    out.close();
    // 读取返回数据。
    strBuf = new StringBuffer();
    BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String line = null;
    while ((line = reader.readLine()) != null) {
        strBuf.append(line).append("\n");
    }
    res = strBuf.toString();
    reader.close();
    reader = null;
  } catch (Exception e) {
    System.err.println("Send post request exception: " + e);
    throw e;
  } finally {
    if (conn != null) {
        conn.disconnect();
        conn = null;
    }
  }
  return res;
}
private static void setCallBack(Map<String, String> formFields, Callback callback) {
```

```
        if (callback != null) {
            String jsonCb = OSSUtils.jsonizeCallback(callback);
            String base64Cb = BinaryUtil.toBase64String(jsonCb.getBytes());
            formFields.put("callback", base64Cb);
            if (callback.hasCallbackVar()) {
                Map<String, String> varMap = callback.getCallbackVar();
                for (Entry<String, String> entry : varMap.entrySet()) {
                    formFields.put(entry.getKey(), entry.getValue());
                }
            }
        }
    }
    public static void main(String[] args) throws Exception {
        PostObjectSample ossPostObject = new PostObjectSample();
        ossPostObject.PostObject();
    }
}
```

有关表单上传的适用场景及详细介绍，请参见表单上传和PostObject。有关回调的详细介绍，请参见Callback。

2.6.4. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。有关追加上传完整代码的更多信息，请参见GitHub。

示例代码

以下代码用于追加上传文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String content1 = "Hello OSS A \n";
String content2 = "Hello OSS B \n";
String content3 = "Hello OSS C \n";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");
// 通过AppendObjectRequest设置多个参数。
AppendObjectRequest appendObjectRequest = new AppendObjectRequest("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content1.getBytes()),meta);
// 通过AppendObjectRequest设置单个参数。
// 设置Bucket名称。
//appendObjectRequest.setBucketName("<yourBucketName>");
// 设置Object名称。即不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
//appendObjectRequest.setKey("<yourObjectName>");
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为InputStream类型。
//appendObjectRequest.setInputStream(new ByteArrayInputStream(content1.getBytes()));
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为File类型。
//appendObjectRequest.setFile(new File("<yourLocalFile>"));
// 指定文件的元信息，第一次追加时有效。
//appendObjectRequest.setMetadata(meta);
// 第一次追加。
// 设置文件的追加位置。
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 文件的64位CRC值。此值根据ECMA-182标准计算得出。
System.out.println(appendObjectResult.getObjectCRC());
// 第二次追加。
// nextPosition指明下一次请求中应当提供的Position，即文件当前的长度。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content2.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 第三次追加。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content3.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

2.6.5. 断点续传上传

通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录点。上传过程中，如果出现网络异常或程序崩溃导致文件上传失败时，将从断点记录处继续上传未上传完成的部分。

参数

您可以通过ossClient.uploadFile方法实现断点续传上传。此方法的uploadFileRequest请求包含的参数请参见下表。

参数	描述
BucketName	存储空间名称。
Key	上传到OSS的文件名称。
UploadFile	待上传的本地文件路径。
TaskNum	上传并发线程数，默认值为1。
PartSize	上传的分片大小，单位为Byte，取值范围为100 KB~5 GB。默认值为100 KB。
EnableCheckpoint	是否开启断点续传功能，默认关闭。
CheckpointFile	记录本地分片上传结果的文件。上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。如果未设置该值，默认与待上传的本地文件同路径，名称为${uploadFile}.ucp。
Callback	使用上传回调。关于上传回调的更多信息，请参见Callback和上传回调。

示例

以下代码用于断点续传上传。

```
/// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");
// 文件上传时设置访问权限ACL。
// meta.setObjectAcl(CannedAccessControlList.Private);
// 通过UploadFileRequest设置多个参数。
// 填写Bucket名称和Object完整路径。Object完整路径中不能包含Bucket名称。
UploadFileRequest uploadFileRequest = new UploadFileRequest("examplebucket","exampleobject.txt");
// 通过UploadFileRequest设置单个参数。
// 填写Bucket名称。
//uploadFileRequest.setBucketName("examplebucket");
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
//uploadFileRequest.setKey("exampleobject.txt");
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
uploadFileRequest.setUploadFile("D:\\localpath\\examplefile.txt");
// 指定上传并发线程数，默认值为1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小。
uploadFileRequest.setPartSize(1 * 1024 * 1024);
// 开启断点续传，默认关闭。
uploadFileRequest.setEnableCheckpoint(true);
// 记录本地分片上传结果的文件。上传过程中的进度信息会保存在该文件中。
uploadFileRequest.setCheckpointFile("yourCheckpointFile");
// 文件的元数据。
uploadFileRequest.setObjectMetadata(meta);
// 设置上传成功回调，参数为Callback类型。
//uploadFileRequest.setCallback("yourCallbackEvent");
// 断点续传上传。
ossClient.uploadFile(uploadFileRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

关于断点续传上传的更多信息，请参见分片上传和断点续传。完整代码请参见GitHub。

2.6.6. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用ossClient.initiateMultipartUpload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

   调用ossClient.uploadPart方法上传分片数据。

   > 说明
   > - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
   > - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
   > - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用ossClient.completeMultipartUpload方法将所有分片合并成完整的文件。

关于分片上传的说明及适用场景，请参见分片上传。分片上传的完整代码请参见GitHub。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建InitiateMultipartUploadRequest对象。
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectName);
// 如果需要在初始化分片时设置文件存储类型，请参考以下示例代码。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setHeader(OSSHeaders.OSS_STORAGE_CLASS, StorageClass.Standard.toString());
// request.setObjectMetadata(metadata);
// 初始化分片。
InitiateMultipartUploadResult upresult = ossClient.initiateMultipartUpload(request);
// 返回uploadId，它是分片上传事件的唯一标识。您可以根据该uploadId发起相关的操作，例如取消分片上传、查询
分片上传等。
String uploadId = upresult.getUploadId();
// partETags是PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags =  new ArrayList<PartETag>();
// 每个分片的大小，用于计算文件有多少个分片。单位为字节。
final long partSize = 1 * 1024 * 1024L;  //1 MB。
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
final File sampleFile = new File("D:\\localpath\\examplefile.txt");
```

```
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
    partCount++;
}
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
    long startPos = i * partSize;
    long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
    InputStream instream = new FileInputStream(sampleFile);
    // 跳过已经上传的分片。
    instream.skip(startPos);
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(objectName);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setInputStream(instream);
    // 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100 KB。
    uploadPartRequest.setPartSize(curPartSize);
    // 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出此范围，OSS将返回InvalidArgu
ment错误码。
    uploadPartRequest.setPartNumber( i + 1);
    // 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。
    UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
    // 每次上传分片之后，OSS的返回结果包含PartETag。PartETag将被保存在partETags中。
    partETags.add(uploadPartResult.getPartETag());
}
// 创建CompleteMultipartUploadRequest对象。
// 在执行完成分片上传操作时，需要提供所有有效的partETags。OSS收到提交的partETags后，会逐一验证每个分片
的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
// 如果需要在完成文件上传的同时设置文件访问权限，请参考以下示例代码。
// completeMultipartUploadRequest.setObjectACL(CannedAccessControlList.PublicRead);
// 完成上传。
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.completeMultipartUpload(co
mpleteMultipartUploadRequest);
System.out.println(completeMultipartUploadResult.getETag());
// 关闭OSSClient。
ossClient.shutdown();
```

取消分片上传事件

您可以调用ossClient.abortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用该uploadId进行任何操作，已上传的分片数据会被删除。

以下代码用于取消分片上传事件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写uploadId，例如0004B999EF518A1FE585B0C9360D****。uploadId来自于InitiateMultipartUpload返回的
结果。
String uploadId = "0004B999EF518A1FE585B0C9360D****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 取消分片上传。
AbortMultipartUploadRequest abortMultipartUploadRequest =
    new AbortMultipartUploadRequest(bucketName, objectName, uploadId);
ossClient.abortMultipartUpload(abortMultipartUploadRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

列举已上传的分片

调用ossClient.listParts方法列举出指定uploadId下所有已经上传成功的分片。

- 简单列举已上传的分片

  以下代码用于简单列举已上传的分片：

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写uploadId，例如0004B999EF518A1FE585B0C9360D****。uploadId来自于InitiateMultipartUpload返回的结果。
String uploadId = "0004B999EF518A1FE585B0C9360D****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举已上传的分片。
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, objectName, uploadId);
// 设置uploadId。
//listPartsRequest.setUploadId(uploadId);
// 设置分页时每一页中分片数量为100个。默认列举1000个分片。
listPartsRequest.setMaxParts(100);
// 指定List的起始位置。只有分片号大于此参数值的分片会被列举。
listPartsRequest.setPartNumberMarker(2);
PartListing partListing = ossClient.listParts(listPartsRequest);
for (PartSummary part : partListing.getParts()) {
  // 获取分片号。
  part.getPartNumber();
  // 获取分片数据大小。
  part.getSize();
  // 获取分片的ETag。
  part.getETag();
  // 获取分片的最后修改时间。
  part.getLastModified();
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举所有已上传的分片

  默认情况下，listParts只能一次列举1000个分片。当分片数量大于1000时，请使用以下代码列举所有已上传的分片。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写uploadId，例如0004B999EF518A1FE585B0C9360D****。uploadId来自于InitiateMultipartUpload返回的结果。
String uploadId = "0004B999EF518A1FE585B0C9360D****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举所有已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, objectName, uploadId);
do {
  partListing = ossClient.listParts(listPartsRequest);
  for (PartSummary part : partListing.getParts()) {
    // 获取分片号。
    part.getPartNumber();
    // 获取分片数据大小。
    part.getSize();
    // 获取分片的ETag。
    part.getETag();
    // 获取分片的最后修改时间。
    part.getLastModified();
  }
// 指定List的起始位置，只有分片号大于此参数值的分片会被列出。
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举所有已上传的分片

  以下代码用于指定每页分片的数量、分页列举所有分片。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写uploadId，例如0004B999EF518A1FE585B0C9360D****。uploadId来自于InitiateMultipartUpload返回的结果。
String uploadId = "0004B999EF518A1FE585B0C9360D****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 分页列举已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, objectName, uploadId);
// 设置分页列举时，每页列举100个分片。
listPartsRequest.setMaxParts(100);
do {
  partListing = ossClient.listParts(listPartsRequest);
  for (PartSummary part : partListing.getParts()) {
    // 获取分片号。
    part.getPartNumber();
    // 获取分片数据大小。
    part.getSize();
    // 获取分片的ETag。
    part.getETag();
    // 获取分片的最后修改时间。
    part.getLastModified();
  }
  listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

列举分片上传事件

调用ossClient.listMultipartUploads方法列举出所有执行中的分片上传事件，即已初始化但尚未完成或已取消的分片上传事件。可设置的参数请参见下表。

参数	作用	设置方法
prefix	限定返回的文件名称必须以指定的prefix作为前缀。注意使用prefix查询时，返回的文件名称中仍会包含prefix。	ListMultipartUploadsRequest.setPrefix(String prefix)
delimiter	用于对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素。	ListMultipartUploadsRequest.setDelimiter(String delimiter)

参数	作用	设置方法
maxUploads	限定此次返回分片上传事件的最大个数，默认值和最大值均为1000。	ListMultipartUploadsRequest.setMaxUploads(Integer maxUploads)
keyMarker	所有文件名称的字典序大于keyMarker参数值的分片上传事件。与uploadIdMarker参数一同使用，用于指定返回结果的起始位置。	ListMultipartUploadsRequest.setKeyMarker(String keyMarker)
uploadIdMarker	与keyMarker参数一同使用，用于指定返回结果的起始位置。 如果未设置keyMarker参数，则此参数无效。如果设置了keyMarker参数，则查询结果中包含： • 文件名称的字典序大于keyMarker参数值的所有文件。 • 文件名称等于keyMarker参数值且uploadId比uploadIdMarker参数值大的所有分片上传事件。	ListMultipartUploadsRequest.setUploadIdMarker(String uploadIdMarker)

- 简单列举分片上传事件

  以下代码用于列举分片上传事件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举分片上传事件。默认列举1000个分片。
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
MultipartUploadListing multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);
for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
    // 获取uploadId。
    multipartUpload.getUploadId();
    // 获取Key。
    multipartUpload.getKey();
    // 获取分片上传的初始化时间。
    multipartUpload.getInitiated();
}
// 关闭OSSClient。
ossClient.shutdown();
```

  返回结果中isTruncated为false，返回nextKeyMarker和nextUploadIdMarker作为下次读取的起点。如果无法一次性获取所有的上传事件，可以采用分页列举的方式。

- 列举全部分片上传事件

默认情况下，listMultipartUploads只能一次列举1000个分片。当分片数量大于1000时，请使用以下代码列举全部分片上传事件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举分片上传事件。
MultipartUploadListing multipartUploadListing;
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
do {
  multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);
  for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
    // 获取uploadId。
    multipartUpload.getUploadId();
    // 获取文件名称。
    multipartUpload.getKey();
    // 获取分片上传的初始化时间。
    multipartUpload.getInitiated();
  }
  listMultipartUploadsRequest.setKeyMarker(multipartUploadListing.getNextKeyMarker());
  listMultipartUploadsRequest.setUploadIdMarker(multipartUploadListing.getNextUploadIdMarker());
} while (multipartUploadListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举全部上传事件

  以下代码用于分页列举所有上传事件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举分片上传事件。
MultipartUploadListing multipartUploadListing;
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
// 设置每页列举的分片上传事件个数。
listMultipartUploadsRequest.setMaxUploads(50);
do {
  multipartUploadListing = ossClient.listMultipartUploads(listMultipartUploadsRequest);
  for (MultipartUpload multipartUpload : multipartUploadListing.getMultipartUploads()) {
    // 获取uploadId。
    multipartUpload.getUploadId();
    // 获取Key。
    multipartUpload.getKey();
    // 获取分片上传的初始化时间。
    multipartUpload.getInitiated();
  }
  listMultipartUploadsRequest.setKeyMarker(multipartUploadListing.getNextKeyMarker());
  listMultipartUploadsRequest.setUploadIdMarker(multipartUploadListing.getNextUploadIdMarker());
} while (multipartUploadListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

2.6.7. 进度条

进度条用于指示上传或下载文件的进度。本文以ossClient.putObject方法为例，介绍如何使用进度条。

以下代码用于演示通过ossClient.putObject上传文件时使用进度条的方法：

```
public class PutObjectProgressListener implements ProgressListener {
  private long bytesWritten = 0;
  private long totalBytes = -1;
  private boolean succeed = false;
  @Override
  public void progressChanged(ProgressEvent progressEvent) {
    long bytes = progressEvent.getBytes();
    ProgressEventType eventType = progressEvent.getEventType();
    switch (eventType) {
    case TRANSFER_STARTED_EVENT:
      System.out.println("Start to upload.....");
      break;
    case REQUEST_CONTENT_LENGTH_EVENT:
      this.totalBytes = bytes;
      System.out.println(this.totalBytes + " bytes in total will be uploaded to OSS");
```

```
        break;
      case REQUEST_BYTE_TRANSFER_EVENT:
        this.bytesWritten += bytes;
        if (this.totalBytes != -1) {
          int percent = (int)(this.bytesWritten * 100.0 / this.totalBytes);
          System.out.println(bytes + " bytes have been written at this time, upload progress: " + percent + "%("
+ this.bytesWritten + "/" + this.totalBytes + ")");
        } else {
          System.out.println(bytes + " bytes have been written at this time, upload ratio: unknown" + "(" + this
.bytesWritten + "/...)");
        }
        break;
      case TRANSFER_COMPLETED_EVENT:
        this.succeed = true;
        System.out.println("Succeed to upload, " + this.bytesWritten + " bytes have been transferred in total")
;
        break;
      case TRANSFER_FAILED_EVENT:
        System.out.println("Failed to upload, " + this.bytesWritten + " bytes have been transferred");
        break;
      default:
        break;
      }
    }
    public boolean isSucceed() {
      return succeed;
    }
    public static void main(String[] args) {
      // Endpoint以杭州为例，其它Region请按实际情况填写。
      String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
      // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日
常运维，请登录RAM控制台创建RAM账号。
      String accessKeyId = "<yourAccessKeyId>";
      String accessKeySecret = "<yourAccessKeySecret>";
      String bucketName = "<yourBucketName>";
      String objectName = "<yourObjectName>";
      // 创建OSSClient实例。
      OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
      try {
        // 上传文件的同时指定了进度条参数。
        ossClient.putObject(new PutObjectRequest(bucketName, objectName, new File("<yourLocalFile>")).
            <PutObjectRequest>withProgressListener(new PutObjectProgressListener()));
      } catch (Exception e) {
        e.printStackTrace();
      }
      // 关闭OSSClient。
      ossClient.shutdown();
    }
}
```

ossClient.putObject、ossClient.getObject、ossClient.uploadPart、ossClient.uploadFile以及ossClient.downloadFile方法均支持进度条功能，使用方法与ossClient.putObject类似。

上传文件时进度条使用方法的完整代码请参见GitHub。

2.6.8. 上传回调

本文介绍如何使用上传回调。

> **说明** 上传回调的完整代码请参见GitHub。关于上传回调的更多信息，请参见开发指南中的上传回调和API参考中的Callback。

以下代码用于上传回调（callback）。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 您的回调服务器地址，例如http://oss-demo.aliyuncs.com:23450。
String callbackUrl = "<yourCallbackServerUrl>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
String content = "Hello OSS";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName,new ByteArrayInp
utStream(content.getBytes()));
// 设置上传回调参数。
Callback callback = new Callback();
callback.setCallbackUrl(callbackUrl);
//（可选）设置回调请求消息头中Host的值，即您的服务器配置Host的值。
// callback.setCallbackHost("yourCallbackHost");
// 设置发起回调时请求body的值。
callback.setCallbackBody("{\\\"mimeType\\\":${mimeType},\\\"size\\\":${size}}");
// 设置发起回调请求的Content-Type。
callback.setCalbackBodyType(CalbackBodyType.JSON);
// 设置发起回调请求的自定义参数，由Key和Value组成，Key必须以x:开始。
callback.addCallbackVar("x:var1", "value1");
callback.addCallbackVar("x:var2", "value2");
putObjectRequest.setCallback(callback);
PutObjectResult putObjectResult = ossClient.putObject(putObjectRequest);
// 读取上传回调返回的消息内容。
byte[] buffer = new byte[1024];
putObjectResult.getResponse().getContent().read(buffer);
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
putObjectResult.getResponse().getContent().close();
// 关闭OSSClient。
ossClient.shutdown();
```

2.7. 下载文件

2.7.1. 概述

OSS Java SDK提供了丰富的文件下载方式：

- 流式下载
- 下载到本地文件
- 范围下载
- 断点续传下载
- 限定条件下载

下载过程中，您还可以通过下载进度条功能查看下载进度。

2.7.2. 流式下载

当下载的文件太大或者一次性下载耗时太长时，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

ossObject对象使用完毕后必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。关闭方法如下：

```
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
ossObject.close();
```

以下代码用于流式下载examplebucket中的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object的完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// ossObject包含文件所在的存储空间名称、文件名称、文件元信息以及一个输入流。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// 读取文件内容。
System.out.println("Object content:");
BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
while (true) {
  String line = reader.readLine();
  if (line == null) break;
  System.out.println("\n" + line);
}
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
reader.close();
// 关闭OSSClient。
ossClient.shutdown();
```

流式下载的完整代码请参见GitHub。

2.7.3. 下载到本地文件

本文介绍如何将存储空间（Bucket）中的文件（Object）下载到本地文件。

以下代码用于将examplebucket中testfolder目录下的exampleobject.txt下载到本地*D:\localpath*路径下的examplefile.txt。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写不包含Bucket名称在内的Object完整路径，例如testfolder/exampleobject.txt。
String objectName = "testfolder/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 下载Object到本地文件，并保存到指定的本地路径中。如果指定的本地文件存在会覆盖，不存在则新建。
// 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
ossClient.getObject(new GetObjectRequest(bucketName, objectName), new File("D:\\localpath\\examplefile
.txt"));
// 关闭OSSClient。
ossClient.shutdown();
```

2.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

指定正常的下载范围

以下代码用于指定正常的下载范围来下载文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
//<yourObjectName>表示从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
// 对于大小为1000 Bytes的文件，正常的字节范围为0~999。
// 获取0~999字节范围内的数据，包括0和999，共1000个字节的数据。如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
getObjectRequest.setRange(0, 999);
// 范围下载。
OSSObject ossObject = ossClient.getObject(getObjectRequest);
// 读取数据。
byte[] buf = new byte[1024];
InputStream in = ossObject.getObjectContent();
for (int n = 0; n != -1; ) {
    n = in.read(buf, 0, buf.length);
}
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
in.close();
// 关闭OSSClient。
ossClient.shutdown();
```

流式读取一次可能无法读取全部数据。如果您需要流式读取64 KB的数据，请使用如下的方式多次读取，直到读取到64 KB或者文件结束。详情请参见InputStream.read。

```
byte[] buf = new byte[1024];
InputStream in = ossObject.getObjectContent();
for (int n = 0; n != -1; ) {
    n = in.read(buf, 0, buf.length);
}
in.close();
```

指定异常的下载范围

假设现有大小为1000 Bytes的Object，则指定的正常下载范围应为0~999。如果指定范围不在有效区间，会导致Range不生效，响应返回值为200，并传送整个Object的内容。请求不合法的示例及返回说明如下：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。

兼容行为范围下载

在请求中增加请求头x-oss-range-behavior:standard，则改变指定范围不在有效区间时OSS的下载行为。假设现有大小为1000 Bytes的Object：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回HTTP Code为416，错误码为InvalidRange。

以下代码用于兼容行为范围下载：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 指定大小为1000 Bytes的文件。
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定兼容行为。
// 范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。
getObjectRequest = new GetObjectRequest(bucketName, objectName);
getObjectRequest.setRange(500, 2000);
getObjectRequest.addHeader("x-oss-range-behavior", "standard");
OSSObject ossObject = ossClient.getObject(getObjectRequest);
ossObject.close();
System.out.println("standard get " + "500~2000 "+ "statusCode:"  + ossObject.getResponse().getStatusCode
());
System.out.println("standard get " + "500~2000 " + "contentLength:" + ossObject.getResponse().getContent
Length());
try {
   // 范围首端取值不在有效区间，以下代码会抛出异常。返回HTTP Code为416，错误码为InvalidRange。
   getObjectRequest = new GetObjectRequest(bucketName, objectName);
   getObjectRequest.setRange(1000, 2000);
   getObjectRequest.addHeader("x-oss-range-behavior", "standard");
   ossClient.getObject(getObjectRequest);
} catch (OSSException e) {
   System.out.println("standard get " + "1000~2000:" + "error code:" + e.getErrorCode());
}
// 关闭OSSClient。
ossClient.shutdown();
```

2.7.5. 断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

您可以通过ossClient.downloadFile方法实现断点续传下载。此方法的DownloadFileRequest请求包含以下参数：

参数	描述	是否必选	默认值	如何设置
bucketName	存储空间名称。	是	无	通过构造方法设置。
key	Object名称。	是	无	通过构造方法设置。
downloadFile	本地文件。OSS文件将下载到该文件。	否	OSS文件名称	通过构造方法或setDownloadFile设置。
partSize	分片大小，取值范围为1 B~5 GB。	否	文件大小/10000	通过setPartSize设置。
taskNum	分片下载的并发数。	否	1	通过setTaskNum设置。
enableCheckpoint	是否开启断点续传功能。	否	关闭	通过setEnableCheckpoint设置。
checkpointFile	记录本地分片下载结果的文件。开启断点续传功能时需要设置此参数。下载过程中的进度信息会保存在该文件中，如果某一分片下载失败，再次下载时会根据文件中记录的点继续下载。下载完成后，该文件会被删除。	否	downloadFile.ucp（与DownloadFile同目录）。	通过setCheckpointFile设置。

以下代码用于断点续传下载：

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
String bucketName = "yourBucketName";
String objectName = "yourObjectName";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 下载请求，10个任务并发下载，启动断点续传。
DownloadFileRequest downloadFileRequest = new DownloadFileRequest(bucketName, objectName);
downloadFileRequest.setDownloadFile("<yourDownloadFile>");
downloadFileRequest.setPartSize(1 * 1024 * 1024);
downloadFileRequest.setTaskNum(10);
downloadFileRequest.setEnableCheckpoint(true);
downloadFileRequest.setCheckpointFile("<yourCheckpointFile>");
// 下载文件。
DownloadFileResult downloadRes = ossClient.downloadFile(downloadFileRequest);
// 下载成功时，会返回文件元信息。
downloadRes.getObjectMetadata();
// 关闭OSSClient。
ossClient.shutdown();
```

2.7.6. 限定条件下载

本文介绍如何使用限定条件下载OSS文件。

下载文件时，可以指定一个或多个限定条件。满足限定条件则下载，不满足则返回错误，不下载。可以使用的限定条件如下：

参数	描述
If-Modified-Since	如果指定的时间早于实际修改时间，则正常传输文件，否则返回错误（304 Not modified）。
If-Unmodified-Since	如果指定的时间等于或者晚于文件实际修改时间，则正常传输文件，否则返回错误（412 Precondition failed）。
If-Match	如果指定的ETag和OSS文件的ETag匹配，则正常传输文件，否则返回错误（412 Precondition failed）。
If-None-Match	如果指定的ETag和OSS文件的ETag不匹配，则正常传输文件，否则返回错误（304 Not modified）。

If-Modified-Since和If-Unmodified-Since可以同时存在。If-Match和If-None-Match可以同时存在。

ETag可以通过ossClient.getObjectMeta方法获取。

以下代码用于限定条件下载：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
// 设置限定条件。
request.setModifiedSinceConstraint(new Date());
// 下载OSS文件到本地文件。
ossClient.getObject(request, new File("<yourLocalFile>"));
// 关闭OSSClient。
ossClient.shutdown();
```

ossClient.getObject和ossClient.downloadFile方法支持限定条件下载。

2.7.7. 进度条

进度条用于指示上传或下载文件的进度。本文以ossClient.getObject方法为例，介绍如何使用进度条。

以下代码用于演示通过ossClient.getObject下载文件时使用进度条的方法。

```
static class GetObjectProgressListener implements ProgressListener {
    private long bytesRead = 0;
    private long totalBytes = -1;
    private boolean succeed = false;
    @Override
    public void progressChanged(ProgressEvent progressEvent) {
      long bytes = progressEvent.getBytes();
      ProgressEventType eventType = progressEvent.getEventType();
      switch (eventType) {
      case TRANSFER_STARTED_EVENT:
        System.out.println("Start to download.....");
        break;
      case RESPONSE_CONTENT_LENGTH_EVENT:
        this.totalBytes = bytes;
        System.out.println(this.totalBytes + " bytes in total will be downloaded to a local file");
        break;
      case RESPONSE_BYTE_TRANSFER_EVENT:
        this.bytesRead += bytes;
        if (this.totalBytes != -1) {
          int percent = (int)(this.bytesRead * 100.0 / this.totalBytes);
          System.out.println(bytes + " bytes have been read at this time, download progress: " +
              percent + "%(" + this.bytesRead + "/" + this.totalBytes + ")");
        } else {
          System.out.println(bytes + " bytes have been read at this time, download ratio: unknown" +
              "(" + this.bytesRead + "/...)");
        }
        break;
```

```
            break;
        case TRANSFER_COMPLETED_EVENT:
            this.succeed = true;
            System.out.println("Succeed to download, " + this.bytesRead + " bytes have been transferred in total
");
            break;
        case TRANSFER_FAILED_EVENT:
            System.out.println("Failed to download, " + this.bytesRead + " bytes have been transferred");
            break;
        default:
            break;
        }
    }
    public boolean isSucceed() {
        return succeed;
    }
  }
public static void main(String[] args) {
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常
运维，请登录RAM控制台创建RAM账号。
  String accessKeyId = "<yourAccessKeyId>";
  String accessKeySecret = "<yourAccessKeySecret>";
  String bucketName = "<yourBucketName>";
  String objectName = "<yourObjectName>";
  OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
  try {
      // 下载文件的同时指定了进度条参数。
      ossClient.getObject(new GetObjectRequest(bucketName, objectName).
          <GetObjectRequest>withProgressListener(new GetObjectProgressListener()),
          new File("<yourLocalFile>"));
  } catch (Exception e) {
    e.printStackTrace();
  }
  // 关闭OSSClient。
  ossClient.shutdown();
}
```

ossClient.putObject、ossClient.getObject、ossClient.uploadPart、ossClient.uploadFile以及ossClient.downloadFile方法均支持进度条功能，使用方法与ossClient.getObject类似。

下载文件的进度条使用方法的完整代码请参见Git Hub。

2.8. 管理文件

2.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息

- 转换文件存储类型
- 列举文件
- 查询文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理软链接

2.8.2. 判断文件是否存在

本文介绍如何判断文件（Object）是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 判断文件是否存在。如果返回值为true，则文件存在，否则存储空间或者文件不存在。
// 设置是否进行重定向或者镜像回源。默认值为true，表示忽略302重定向和镜像回源；如果设置isINoss为false，则进
行302重定向或者镜像回源。
//boolean isINoss = true;
// 填写Bucket名称和Object完整路径。Object完整路径中不能包含Bucket名称。
boolean found = ossClient.doesObjectExist("examplebucket", "exampleobject.txt");
//boolean found = ossClient.doesObjectExist("examplebucket", "exampleobject.txt", isINoss);
System.out.println(found);
// 关闭OSSClient。
ossClient.shutdown();
```

2.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	CannedAccessControlList.Default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	CannedAccessControlList.Private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	CannedAccessControlList.PublicRead

访问权限	描述	访问权限值
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	CannedAccessControlList.PublicReadWrite

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于设置指定文件的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置文件的访问权限为公共读。
ossClient.setObjectAcl("<yourBucketName>", "<yourObjectName>", CannedAccessControlList.PublicRead);
// 关闭OSSClient。
ossClient.shutdown();
```

设置文件访问权限更多详情，请参见PutObjectACL。

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
// 获取文件的访问权限。
ObjectAcl objectAcl = ossClient.getObjectAcl("<yourBucketName>", "<yourObjectName>");
System.out.println(objectAcl.getPermission().toString());
// 关闭OSSClient。
ossClient.shutdown();
```

获取文件访问权限更多详情，请参见GetObjectACL。

2.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息。

> **说明** 有关文件元信息的更多详情，请参见开发指南中的文件元信息。

设置文件元信息

以下为设置HTTP header和自定义元信息的详细示例。

- 设置HTTP header

  以下代码用于设置HTTP header：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String content = "Hello OSS";
// 创建上传文件的元信息，可以通过文件元信息设置HTTP header。
ObjectMetadata meta = new ObjectMetadata();
String md5 = BinaryUtil.toBase64String(BinaryUtil.calculateMd5(content.getBytes()));
// 开启文件内容MD5校验。开启后OSS会把您提供的MD5与文件的MD5比较，不一致则抛出异常。
meta.setContentMD5(md5);
// 指定上传的内容类型。内容类型决定浏览器将以什么形式、什么编码读取文件。如果没有指定则根据文件的扩展名生成，如果没有扩展名则为默认值application/octet-stream。
meta.setContentType("text/plain");
// 设置内容被下载时的名称。
meta.setContentDisposition("attachment; filename=\"DownloadFilename\"");
// 设置上传文件的长度。如超过此长度，则上传文件会被截断，上传的文件长度为设置的长度。如小于此长度，则为上传文件的实际长度。
meta.setContentLength(content.length());
// 设置内容被下载时网页的缓存行为。
meta.setCacheControl("Download Action");
// 设置缓存过期时间，格式是格林威治时间（GMT）。
meta.setExpirationTime(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
// 设置内容被下载时的编码格式。
meta.setContentEncoding("utf-8");
// 设置header。
meta.setHeader("<yourHeader>", "<yourHeaderValue>");
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 上传文件。
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()), meta);
// 关闭OSSClient。
ossClient.shutdown();
```

  HTTP header详情请参见RFC2616。

- 设置自定义元信息

  您可以自定义文件的元信息来对文件进行描述。

  以下代码用于设置文件的自定义元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String content = "Hello OSS";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建文件元信息。
ObjectMetadata meta = new ObjectMetadata();
// 设置自定义元信息property值为property-value。建议使用Base64编码。
meta.addUserMetadata("property", "property-value");
// 上传文件。
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()), meta);
// 获取文件元信息。
ossClient.getObjectMetadata("<yourBucketName>", "<yourObjectName>");
// 关闭OSSClient。
ossClient.shutdown();
```

下载文件时，文件元信息也会同时下载。 一个文件可以有多个元信息，总大小不能超过8 KB。

修改文件元信息

以下代码用于修改文件的元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,accessKeySecret);
// 设置源文件与目标文件相同，调用ossClient.copyObject方法修改文件元信息。
CopyObjectRequest request = new CopyObjectRequest(sourceBucketName, sourceObjectName, destination
BucketName, destinationObjectName);
ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。内容类型决定浏览器将以什么形式、什么编码读取文件。如果没有指定则根据文件的扩展名生成，如果没有扩展名则为默认值application/octet-stream。
meta.setContentType("text/plain");
// 设置内容被下载时的名称。
meta.setContentDisposition("Download File Name");
// 设置内容被下载时网页的缓存行为。
meta.setCacheControl("Download Action");
// 设置缓存过期时间，格式是格林威治时间（GMT）。
meta.setExpirationTime(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
// 设置内容被下载时的编码格式。
meta.setContentEncoding("utf-8");
// 设置header。
meta.setHeader("<yourHeader>", "<yourHeaderValue>");
// 设置自定义元信息property值为property-value。
meta.addUserMetadata("property", "property-value");
request.setNewObjectMetadata(meta);
// 修改元信息。
ossClient.copyObject(request);
// 关闭OSSClient。
ossClient.shutdown();
```

获取文件元信息

您可以通过以下两种方法获取文件元信息：

方法	描述	优势
ossClient.getSimplifiedObjectMeta	获取文件的ETag、Size（文件大小）、LastModified（最后修改时间）。	更轻量、更快
ossClient.getObjectMetadata	获取文件的全部元信息。	无

以下代码用于获取文件元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取文件的部分元信息。
SimplifiedObjectMeta objectMeta = ossClient.getSimplifiedObjectMeta("<yourBucketName>", "<yourObjectName>");
System.out.println(objectMeta.getSize());
System.out.println(objectMeta.getETag());
System.out.println(objectMeta.getLastModified());
// 获取文件的全部元信息。
ObjectMetadata metadata = ossClient.getObjectMetadata("<yourBucketName>", "<yourObjectName>");
System.out.println(metadata.getContentType());
System.out.println(metadata.getLastModified());
System.out.println(metadata.getExpirationTime());
// 关闭OSSClient。
ossClient.shutdown();
```

2.8.5. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何转换文件（Object）的存储类型。

有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

以下提供了详细的示例代码用于Object存储类型的相互转换。

- 以下代码用于将Object的存储类型从标准或低频访问转换为归档类型：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 本示例中的Bucket与Object需提前创建好，且Object类型为标准或低频访问存储类型。
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建CopyObjectRequest对象。
CopyObjectRequest request = new CopyObjectRequest(bucketName, objectName, bucketName, objectName) ;
// 创建ObjectMetadata对象。
ObjectMetadata objectMetadata = new ObjectMetadata();
// 封装header，此处以设置存储类型为归档类型为例。
objectMetadata.setHeader("x-oss-storage-class", StorageClass.Archive);
request.setNewObjectMetadata(objectMetadata);
// 更改文件存储类型。
CopyObjectResult result = ossClient.copyObject(request);
// 关闭OSSClient。
ossClient.shutdown();
```

- 以下代码用于将Object的存储类型从归档转换为低频访问类型：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 本示例中的Bucket与Object需提前创建好，且Object类型为归档存储类型。
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取文件元信息。
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);
// 检查目标文件是否为归档类型。如果是，则需要先解冻才能更改存储类型。
StorageClass storageClass = objectMetadata.getObjectStorageClass();
System.out.println("storage type:" + storageClass);
if (storageClass == StorageClass.Archive) {
  // 解冻文件。
  ossClient.restoreObject(bucketName, objectName);
  // 等待解冻完成。
  do {
    Thread.sleep(1000);
    objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);
  } while (!objectMetadata.isRestoreCompleted());
}
 // 创建CopyObjectRequest对象。
CopyObjectRequest request = new CopyObjectRequest(bucketName, objectName, bucketName, objectNa
me) ;
// 创建ObjectMetadata对象。
objectMetadata = new ObjectMetadata();
// 封装header，此处以设置存储类型为低频访问类型为例。
objectMetadata.setHeader("x-oss-storage-class", StorageClass.IA);
request.setNewObjectMetadata(objectMetadata);
// 更改文件存储类型。
CopyObjectResult result = ossClient.copyObject(request);
// 关闭OSSClient。
ossClient.shutdown();
```

各种存储类型的存储费用介绍，请参见计量项和计费项的计量计费概述一节。

2.8.6. 列举文件

本文介绍如何列举指定存储空间下（Bucket）的所有文件（Object）、指定个数的文件、指定前缀的文件等。

背景信息

您可以调用GetBucket (ListObjects)或GetBucketV2 (ListObjectsV2)接口，一次性列举某个Bucket下最多1000个Object。您可以通过指定参数实现多种列举功能，例如通过指定参数列举指定起始位置后的所有文件、列举指定目录下的文件和子目录、以及列举结果分页。这两个接口的主要区别如下：

- 使用GetBucket (ListObjects)接口列举文件时，默认返回owner信息。
- 使用GetBucketV2 (ListObjectsV2)接口列举文件时，需通过fetchOwner指定是否在返回结果中包含owner

信息。

> **说明** 对于开启版本控制的Bucket，建议使用GetBucketV2 (ListObjectsV2)接口列举文件。

以下分别介绍通过GetBucket (ListObjects)以及GetBucketV2 (ListObjectsV2)方法列举文件时涉及的参数说明。

- 通过GetBucket (ListObjects)方法列举文件

  GetBucket (ListObjects)有以下三类参数格式：

  - ObjectListing listObjects(String bucketName)：列举存储空间下的文件。单次请求默认最多列举100个文件。
  - ObjectListing listObjects(String bucketName, String prefix)： 列举存储空间下指定前缀的文件。单次请求默认最多列举100个文件。
  - ObjectListing listObjects(ListObjectsRequest listObjectsRequest)：提供多种过滤功能，实现灵活的查询功能。

  GetBucket (ListObjects)涉及参数说明如下：

参数	描述	方法
objectSummaries	限定返回的文件元信息。	List<OSSObjectSummary> getObjectSummaries()
prefix	本次查询结果的前缀。	String getPrefix()
delimiter	对文件名称进行分组的字符。	String getDelimiter()
marker	标明本次列举文件的起点。	String getMarker()
maxKeys	列举文件的最大个数。	int getMaxKeys()
nextMarker	下一次列举文件的起点。	String getNextMarker()
isTruncated	指明列举文件是否被截断。 ○ 列举完没有截断，返回值为false。 ○ 没列举完就有截断，返回值为true。	boolean isTruncated()
commonPrefixes	以delimiter结尾，且有共同前缀的文件集合。	List<String> getCommonPrefixes()
encodingType	指明返回结果中编码使用的类型。	String getEncodingType()

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  GetBucketV2 (ListObjectsV2)有以下三类参数格式：

  - ListObjectsV2Result listObjectsV2(String bucketName)：列举存储空间下的文件。单次请求默认最多列举100个文件。
  - ListObjectsV2Result listObjectsV2(String bucketName, String prefix)： 列举存储空间下指定前缀的文件。单次请求默认最多列举100个文件。

- ListObjectsV2Result listObjectsV2(ListObjectsRequest listObjectsRequest)：提供多种过滤功能，实现灵活的查询功能。

GetBucketV2 (ListObjectsV2)涉及参数说明如下：

参数	描述	方法
objectSummaries	限定返回的文件元信息。	List<OSSObjectSummary> getObjectSummaries()
prefix	本次查询结果的前缀。	String getPrefix()
delimiter	对文件名称进行分组的字符。	String getDelimiter()
startAfter	此次列举文件的起点。	String getStartAfter()
maxKeys	列举文件的最大个数。	int getMaxKeys()
continuationToken	此次列举文件操作使用的continuationToken。	String getContinuationToken()
nextContinuationToken	下次列举文件的continuationToken。	String getNextContinuationToken()
isTruncated	指明列举文件是否被截断。 ○ 列举完没有截断，返回值为false。 ○ 没列举完就有截断，返回值为true。	boolean isTruncated()
commonPrefixes	以delimiter结尾，且有共同前缀的文件集合。	List<String> getCommonPrefixes()
encodingType	指明返回结果中编码使用的类型。	String getEncodingType()
fetchOwner	指定是否在返回结果中包含owner信息。 ○ true：表示返回结果中包含owner信息。 ○ false：表示返回结果中不包含owner信息。	String getFetchOwner()

简单列举文件

您可以通过GetBucket (ListObjects)方法列举或GetBucketV2 (ListObjectsV2)方法列举指定存储空间下的文件。

- 通过GetBucket (ListObjects)方法

  以下代码用于通过GetBucket (ListObjects)方法列举指定存储空间下的文件，默认列举100个文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String KeyPrefix = "<yourKeyPrefix>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举文件。如果不设置KeyPrefix，则列举存储空间下的所有文件。如果设置KeyPrefix，则列举包含指定前缀的文件。
ObjectListing objectListing = ossClient.listObjects(bucketName, KeyPrefix);
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过GetBucketV2 (ListObjectsV2)方法

  以下代码用于通过GetBucketV2 (ListObjectsV2)方法列举指定存储空间下的文件，默认列举100个文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举文件。如果不设置KeyPrefix，则列举存储空间下的所有文件。如果设置KeyPrefix，则列举包含指定前缀的文件。
ListObjectsV2Result result = ossClient.listObjectsV2(bucketName);
List<OSSObjectSummary> ossObjectSummaries = result.getObjectSummaries();
for (OSSObjectSummary s : ossObjectSummaries) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

通过ListObjectsRequest列举文件

通过设置ListObjectsRequest的参数实现各种灵活的查询功能。ListObjectsRequest的参数如下：

参数	描述	方法
prefix	限定返回的文件必须以prefix作为前缀。	setPrefix(String prefix)

参数	描述	方法
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。	setDelimiter(String delimiter)
marker	列举指定marker之后的文件。	setMarker(String marker)
maxKeys	限定此次列举文件的最大个数。默认值为100，最大值为1000。	setMaxKeys(Integer maxKeys)
encodingType	请求响应体中文件名称采用的编码方式，目前仅支持URL。	setEncodingType(String encodingType)

- 列举指定个数的文件

  以下代码用于列举指定个数的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置最大个数。
final int maxKeys = 200;
// 列举文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withMaxKeys(maxKeys));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举指定前缀的文件

  以下代码用于列举包含指定前缀（prefix）的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定前缀。
final String keyPrefix = "<yourkeyPrefix>";
// 列举包含指定前缀的文件。默认列举100个文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withPrefix(keyPrefix));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举指定marker之后的文件

  参数marker代表文件名称。以下代码用于列举指定marker之后的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
final String marker = "<yourMarker>";
// 列举指定marker之后的文件。默认列举100个文件。
ObjectListing objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withMarker(marker));
List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举所有文件

  以下代码用于分页列举指定存储空间下的所有文件。每页列举的文件个数通过maxKeys指定。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
final int maxKeys = 200;
String nextMarker = null;
ObjectListing objectListing;
do {
  objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).withMarker(nextMarker).withMaxKeys(maxKeys));
  List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
  for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
  }
  nextMarker = objectListing.getNextMarker();
} while (objectListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举指定前缀的文件

  以下代码用于分页列举包含指定前缀的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定每页200个文件。
final int maxKeys = 200;
final String keyPrefix = "<yourkeyPrefix>";
String nextMarker = "<yourNextMarker>";
ObjectListing objectListing;
do {
  objectListing = ossClient.listObjects(new ListObjectsRequest(bucketName).
      withPrefix(keyPrefix).withMarker(nextMarker).withMaxKeys(maxKeys));
  List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
  for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
  }
  nextMarker = objectListing.getNextMarker();
} while (objectListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 指定文件名称编码

  如果文件名称含有以下特殊字符，需要进行编码传输。OSS目前仅支持URL编码。

  - 单引号（' '）
  - 双引号（" "）
  - and符号（&）
  - 尖括号（< >）
  - 顿号（、）
  - 中文

  以下代码用于指定文件名称编码。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
final int maxKeys = 200;
final String keyPrefix = "<yourkeyPrefix>";
String nextMarker = "<yourNextMarker>";
ObjectListing objectListing;
do {
  ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
  listObjectsRequest.setPrefix(keyPrefix);
  listObjectsRequest.setMaxKeys(maxKeys);
  listObjectsRequest.setMarker(nextMarker);
  // 指定文件名称编码。
  listObjectsRequest.setEncodingType("url");
  objectListing = ossClient.listObjects(listObjectsRequest);
  // 文件解码。
  for (OSSObjectSummary objectSummary: objectListing.getObjectSummaries()) {
    System.out.println("Key:" + URLDecoder.decode(objectSummary.getKey(), "UTF-8"));
  }
  // commonPrefixes解码。
  for (String commonPrefixes: objectListing.getCommonPrefixes()) {
    System.out.println("CommonPrefixes:" + URLDecoder.decode(commonPrefixes, "UTF-8"));
  }
  // nextMarker解码。
  if (objectListing.getNextMarker() != null) {
    nextMarker = URLDecoder.decode(objectListing.getNextMarker(), "UTF-8");
  }
} while (objectListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

通过ListObjectsV2Request列举文件

通过设置ListObjectsV2Request参数实现灵活的列举文件功能，例如列举指定个数的文件、指定前缀（prefix）的文件、分页列举所有文件等。

ListObjectsV2Request参数说明如下：

参数	描述	方法
prefix	限定返回的文件必须以prefix作为前缀。	setPrefix(String prefix)
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。	setDelimiter(String delimiter)

参数	描述	方法
maxKeys	限定此次列举文件的最大个数。默认值为100，最大值为1000。	setMaxKeys(Integer maxKeys)
startAfter	此次列举文件的起点。	setStartAfter(String startAfter)
continuationToken	此次列举文件使用的continuationToken。	setContinuationToken(String continuationToken)
encodingType	请求响应体中文件名称采用的编码方式，目前仅支持URL。	setEncodingType(String encodingType)
fetchOwner	本次列举结果中是否包含文件的owner的信息。	setFetchOwner(boolean fetchOwner)

- 列举指定个数的文件

  以下代码用于列举指定个数的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置最大个数。
final int maxKeys = 200;
// 列举文件，每次请求默认返回100个文件，此次指定设置最多返回文件数为200。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
listObjectsV2Request.setMaxKeys(maxKeys);
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
List<OSSObjectSummary> ossObjectSummaries = result.getObjectSummaries();
for (OSSObjectSummary s : ossObjectSummaries) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举指定前缀的文件

  以下代码用于列举包含指定前缀（prefix）的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定前缀。
String prefix = "<yourKeyPrefix>";
// 列举指定前缀的文件。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
listObjectsV2Request.setPrefix(prefix);
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
List<OSSObjectSummary> ossObjectSummaries = result.getObjectSummaries();
for (OSSObjectSummary s : ossObjectSummaries) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举指定startAfter之后的文件

  以下代码用于列举文件名称字典序中排在指定字符串startAfter之后的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举指定名称之后的文件，传入的字符串即可以是已存在的文件的名称也可以是其它值。
// 以传入字符串"test-obj"为例，则返回字典序在"test-obj"之后的文件。即使存储空间中已经存在"test-obj"文件，返回结果中依然不会返回"test-obj"文件。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
listObjectsV2Request.setStartAfter("test-obj");
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
List<OSSObjectSummary> ossObjectSummaries = result.getObjectSummaries();
for (OSSObjectSummary s : ossObjectSummaries) {
    System.out.println("\t" + s.getKey());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 列举结果中返回文件的owner信息

  以下代码用于列举结果中返回文件的owner信息。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 默认列举的文件信息中不包含owner信息。如果需要包含owner信息，将fetchOwner参数设置为true。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
listObjectsV2Request.setFetchOwner(true);
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
List<OSSObjectSummary> ossObjectSummaries = result.getObjectSummaries();
for (OSSObjectSummary s : ossObjectSummaries) {
  System.out.println("\t" + s.getKey());
  if (s.getOwner() != null) {
    System.out.println("owner id:" + s.getOwner().getId());
    System.out.println("name:" + s.getOwner().getDisplayName());
  }
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举所有文件

  以下代码用于分页列举指定存储空间下的所有文件。每页列举的文件个数通过maxKeys指定。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置最大个数。
final int maxKeys = 200;
String nextContinuationToken = null;
ListObjectsV2Result result = null;
// 分页列举，每次传入上次返回结果中的nextContinuationToken。
do {
  ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName).withMaxKeys(maxKeys);
  listObjectsV2Request.setContinuationToken(nextContinuationToken);
  result = ossClient.listObjectsV2(listObjectsV2Request);
  List<OSSObjectSummary> sums = result.getObjectSummaries();
  for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
  }
  nextContinuationToken = result.getNextContinuationToken();
} while (result.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 分页列举指定前缀的文件

  以下代码用于分页列举包含指定前缀的文件。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String KeyPrefix = "<yourKeyPrefix>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置最大个数。
final int maxKeys = 200;
String nextContinuationToken = null;
ListObjectsV2Result result = null;
final String keyPrefix = "<yourkeyPrefix>";
// 分页列举指定前缀的文件。
do {
  ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName).withMaxKeys(maxKeys);
  listObjectsV2Request.setPrefix(keyPrefix);
  listObjectsV2Request.setContinuationToken(nextContinuationToken);
  result = ossClient.listObjectsV2(listObjectsV2Request);
  List<OSSObjectSummary> sums = result.getObjectSummaries();
  for (OSSObjectSummary s : sums) {
    System.out.println("\t" + s.getKey());
  }
  nextContinuationToken = result.getNextContinuationToken();
} while (result.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 指定文件名称编码

> **说明** 有关文件名称包含的特殊字符详情，请参见文件名称特殊字符说明。

以下代码用于指定文件名称编码。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置最大个数。
final int maxKeys = 200;
String nextContinuationToken = null;
ListObjectsV2Result result = null;
final String keyPrefix = "<yourkeyPrefix>";
// 指定返回结果使用URL编码，则您需要对结果中的prefix、delemiter、startAfter、key和commonPrefix进行URL解码。
do {
  ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName).withMaxKeys(maxKeys);
  listObjectsV2Request.setPrefix(keyPrefix);
  listObjectsV2Request.setEncodingType("url");
  listObjectsV2Request.setContinuationToken(nextContinuationToken);
  result = ossClient.listObjectsV2(listObjectsV2Request);
  // prefix解码。
  if (result.getPrefix() != null) {
    String prefix = URLDecoder.decode(result.getPrefix(), "UTF-8");
    System.out.println("prefix: " + prefix);
  }
  // delimiter解码。
  if (result.getDelimiter() != null) {
    String delimiter = URLDecoder.decode(result.getDelimiter(), "UTF-8");
    System.out.println("delimiter: " + delimiter);
  }
  // startAfter解码。
  if (result.getStartAfter() != null) {
    String startAfter = URLDecoder.decode(result.getStartAfter(), "UTF-8");
    System.out.println("startAfter: " + startAfter);
  }
  // 文件名称解码。
  for (OSSObjectSummary s : result.getObjectSummaries()) {
    String decodedKey = URLDecoder.decode(s.getKey(), "UTF-8");
    System.out.println("key: " + decodedKey);
  }
  // commonPrefixes解码。
  for (String commonPrefix: result.getCommonPrefixes()) {
    String decodeCommonPrefix = URLDecoder.decode(commonPrefix, "UTF-8");
    System.out.println("CommonPrefix:" + decodeCommonPrefix);
  }
  nextContinuationToken = result.getNextContinuationToken();
} while (result.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为objects。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见文件夹功能。创建文件夹的完整代码请参见GitHub。

假设存储空间中包含文件 `oss.jpg` 、 `fun/test.jpg` 、 `fun/movie/001.avi` 和 `fun/movie/007.avi` ，以正斜线（/）作为文件夹的分隔符。以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举存储空间下的所有文件
  - 通过GetBucket (ListObjects)列举存储空间下的所有文件

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
// 列举文件。
ObjectListing listing = ossClient.listObjects(listObjectsRequest);
// 遍历所有文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}
// 遍历所有commonPrefix。
System.out.println("CommonPrefixes:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过GetBucketV2 (ListObjectsV2)列举指定存储空间下的所有文件

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举文件。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
// 遍历文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}
// 遍历commonPrefix。
System.out.println("CommonPrefixes:");
for (String commonPrefix : result.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 返回结果

  通过以上两种方法列举指定存储空间下所有文件的返回结果如下。

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg
CommonPrefixes:
```

- 列举指定目录下所有文件

- 通过GetBucket (ListObjects)方法列举指定目录下的所有文件

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
// 设置prefix参数来获取fun目录下的所有文件。
listObjectsRequest.setPrefix("fun/");
// 递归列举fun目录下的所有文件。
ObjectListing listing = ossClient.listObjects(listObjectsRequest);
// 遍历所有文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}
// 遍历所有commonPrefix。
System.out.println("\nCommonPrefixes:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过GetBucketV2 (ListObjectsV2)方法列举指定目录下所有文件

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 构造ListObjectsV2Request请求。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
// 设置prefix参数来获取fun目录下的文件。
listObjectsV2Request.setPrefix("fun/");
// 发起列举请求。
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
// 遍历文件。
System.out.println("Objects:");
for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}
// 遍历commonPrefix。
System.out.println("\nCommonPrefixes:");
for (String commonPrefix : result.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过以上两种方法列举指定目录下所有文件的返回结果如下。

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
CommonPrefixes:
```

- 列举目录下的文件和子目录

- 通过GetBucket (ListObjects)方法列举目录下的文件和子目录

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);
// 设置正斜线（/）为文件夹的分隔符。
listObjectsRequest.setDelimiter("/");
// 列出fun目录下的所有文件和文件夹。
listObjectsRequest.setPrefix("fun/");
ObjectListing listing = ossClient.listObjects(listObjectsRequest);
// 遍历所有文件。
System.out.println("Objects:");
// objectSummaries的列表中给出的是fun目录下的文件。
for (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {
    System.out.println(objectSummary.getKey());
}
// 遍历所有commonPrefix。
System.out.println("\nCommonPrefixes:");
// commonPrefixs列表中显示的是fun目录下的所有子文件夹。由于fun/movie/001.avi和fun/movie/007.avi属于fun文件夹下的movie目录，因此这两个文件未在列表中。
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过GetBucketV2 (ListObjectsV2)方法列举目录下的文件和子目录

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 构造ListObjectsV2Request请求。
ListObjectsV2Request listObjectsV2Request = new ListObjectsV2Request(bucketName);
// 设置prefix参数来获取fun目录下的所有文件与文件夹。
listObjectsV2Request.setPrefix("fun/");
// 设置正斜线（/）为文件夹的分隔符。
listObjectsV2Request.setDelimiter("/");
// 发起列举请求。
ListObjectsV2Result result = ossClient.listObjectsV2(listObjectsV2Request);
// 遍历文件。
System.out.println("Objects:");
// objectSummaries的列表中给出的是fun目录下的文件。
for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
  System.out.println(objectSummary.getKey());
}
// 遍历commonPrefix。
System.out.println("\nCommonPrefixes:");
// commonPrefixs列表中显示的是fun目录下的所有子文件夹。由于fun/movie/001.avi和fun/movie/007.avi属于fun文件夹下的movie目录，因此这两个文件未在列表中。
for (String commonPrefix : result.getCommonPrefixes()) {
  System.out.println(commonPrefix);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过以上两种方法列举指定目录下的文件和子目录的返回结果如下：

```
Objects:
fun/test.jpg
CommonPrefixes:
fun/movie/
```

- 获取指定目录下的文件大小
  - 通过GetBucket (ListObjects)方法获取指定目录下的文件大小

```
// 获取某个存储空间下指定目录（文件夹）下的文件大小。
private static long calculateFolderLength(OSS ossClient, String bucketName, String folder) {
    long size = 0L;
    ObjectListing objectListing = null;
    do {
      // MaxKey默认值为100，最大值为1000。
      ListObjectsRequest request = new ListObjectsRequest(bucketName).withPrefix(folder).withMaxKey
s(1000);
      if (objectListing != null) {
          request.setMarker(objectListing.getNextMarker());
      }
      objectListing = ossClient.listObjects(request);
      List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
      for (OSSObjectSummary s : sums) {
          size += s.getSize();
      }
   } while (objectListing.isTruncated());
    return size;
}
public void Calculate() {
    // Endpoint以杭州为例，其它Region请按实际情况填写。
    String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
    // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问
或日常运维，请登录RAM控制台创建RAM账号。
    String accessKeyId = "<yourAccessKeyId>";
    String accessKeySecret = "<yourAccessKeySecret>";
    String bucketName = "<yourBucketName>";
    // 创建OSSClient实例。
    OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
    // 指定前缀。若您希望遍历主目录下的文件夹，则将此值置空。
    final String keyPrefix = "<YourPrefix>";
    ObjectListing objectListing = null;
    do {
      // 默认情况下，每次列举100个文件或目录。
      ListObjectsRequest request = new ListObjectsRequest(bucketName).withDelimiter("/").withPrefix(
keyPrefix);
      if (objectListing != null) {
          request.setMarker(objectListing.getNextMarker());
      }
      objectListing = ossClient.listObjects(request);
      List<String> folders = objectListing.getCommonPrefixes();
      for (String folder : folders) {
          System.out.println(folder + " : " + (calculateFolderLength(ossClient, bucketName, folder) / 1024) +
"KB");
      }
      List<OSSObjectSummary> sums = objectListing.getObjectSummaries();
      for (OSSObjectSummary s : sums) {
          System.out.println(s.getKey() + " : " + (s.getSize() / 1024) + "KB");
      }
    } while (objectListing.isTruncated());
    ossClient.shutdown();
}
```

- 通过GetBucketV2 (ListObjectsV2)方法获取指定目录下的文件大小

```
// 获取某个存储空间下指定目录（文件夹）下的文件大小。
private static long calculateFolderLength(OSS ossClient, String bucketName, String folder) {
    long size = 0L;
    ListObjectsV2Result result = null;
    do {
        // MaxKey默认值为100，最大值为1000。
        ListObjectsV2Request request = new ListObjectsV2Request(bucketName).withPrefix(folder).withMa
xKeys(1000);
        if (result != null) {
            request.setContinuationToken(result.getNextContinuationToken());
        }
        result = ossClient.listObjectsV2(request);
        List<OSSObjectSummary> sums = result.getObjectSummaries();
        for (OSSObjectSummary s : sums) {
            size += s.getSize();
        }
    } while (result.isTruncated());
    return size;
}
public void Calculate() {
    // Endpoint以杭州为例，其它Region请按实际情况填写。
    String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
    // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或
日常运维，请登录RAM控制台创建RAM账号。
    String accessKeyId = "<yourAccessKeyId>";
    String accessKeySecret = "<yourAccessKeySecret>";
    String bucketName = "<yourBucketName>";
    // 创建OSSClient实例。
    OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
    // 指定前缀。若您希望遍历主目录文件夹，将该值置空。
    final String keyPrefix = "<YourPrefix>";
    ListObjectsV2Result result = null;
    do {
        // 默认情况下，每次列举100个文件或目录。
        ListObjectsV2Request request = new ListObjectsV2Request(bucketName).withDelimiter("/").withPr
efix(keyPrefix);
        if (result != null) {
            request.setContinuationToken(result.getNextContinuationToken());
        }
        result = ossClient.listObjectsV2(request);
        List<String> folders = result.getCommonPrefixes();
        for (String folder : folders) {
            System.out.println(folder + " : " + (calculateFolderLength(ossClient, bucketName, folder) / 1024) +
"KB");
        }
        List<OSSObjectSummary> sums = result.getObjectSummaries();
        for (OSSObjectSummary s : sums) {
            System.out.println(s.getKey() + " : " + (s.getSize() / 1024) + "KB");
        }
    } while (result.isTruncated());
    ossClient.shutdown();
}
```

2.8.7. 查询文件

本文介绍如何使用Java SDK的SelectObject查询CSV和JSON文件。

> **说明** 有关SelectObject的更多详情，请参考开发指南使用SelectObject查询文件及SelectObject API。

以下代码用于查询CSV和JSON文件：

```
package samples;
import com.aliyun.oss.model.*;
import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClientBuilder;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
/**
 * Examples of create select object metadata and select object.
 *
 */
public class SelectObjectSample {
  private static String endpoint = "<endpoint, http://oss-cn-hangzhou.aliyuncs.com>";
  private static String accessKeyId = "<yourAccessKeyId>";
  private static String accessKeySecret = "<yourAccessKeySecret>";
  private static String bucketName = "<yourBucketName>";
  public static void main(String[] args) throws Exception {
    OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
    selectCsvSample("test.csv", ossClient);
    selectJsonSample("test.json", ossClient);
    ossClient.shutdown();
  }
  private static void selectCsvSample(String key, OSS ossClient) throws Exception {
    String content = "name,school,company,age\r\n" +
        "Lora Francis,School A,Staples Inc,27\r\n" +
        "Eleanor Little,School B,\"Conectiv, Inc\",43\r\n" +
        "Rosie Hughes,School C,Western Gas Resources Inc,44\r\n" +
        "Lawrence Ross,School D,MetLife Inc.,24";
    ossClient.putObject(bucketName, key, new ByteArrayInputStream(content.getBytes()));
    SelectObjectMetadata selectObjectMetadata = ossClient.createSelectObjectMetadata(
        new CreateSelectObjectMetadataRequest(bucketName, key)
            .withInputSerialization(
                new InputSerialization().withCsvInputFormat(
                    new CSVFormat().withHeaderInfo(CSVFormat.Header.Use).withRecordDelimiter("\r\n"))));
    System.out.println(selectObjectMetadata.getCsvObjectMetadata().getTotalLines());
    System.out.println(selectObjectMetadata.getCsvObjectMetadata().getSplits());
    SelectObjectRequest selectObjectRequest =
        new SelectObjectRequest(bucketName, key)
            .withInputSerialization(
                new InputSerialization().withCsvInputFormat(
                    new CSVFormat().withHeaderInfo(CSVFormat.Header.Use).withRecordDelimiter("\r\n")))
            .withOutputSerialization(new OutputSerialization().withCsvOutputFormat(new CSVFormat()));
    selectObjectRequest.setExpression("select * from ossobject where _4 > 40");
    OSSObject ossObject = ossClient.selectObject(selectObjectRequest);
```

```
    // read object content from ossObject
    BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
    while (true) {
        String line = reader.readLine();
        if (line == null) {
            break;
        }
        System.out.println(line);
    }
    reader.close();
    ossClient.deleteObject(bucketName, key);
}
private static void selectJsonSample(String key, OSS ossClient) throws Exception {
    final String content = "{\n" +
            "\t\"name\": \"Lora Francis\",\n" +
            "\t\"age\": 27,\n" +
            "\t\"company\": \"Staples Inc\"\n" +
            "}\n" +
            "{\n" +
            "\t\"name\": \"Eleanor Little\",\n" +
            "\t\"age\": 43,\n" +
            "\t\"company\": \"Conectiv, Inc\"\n" +
            "}\n" +
            "{\n" +
            "\t\"name\": \"Rosie Hughes\",\n" +
            "\t\"age\": 44,\n" +
            "\t\"company\": \"Western Gas Resources Inc\"\n" +
            "}\n" +
            "{\n" +
            "\t\"name\": \"Lawrence Ross\",\n" +
            "\t\"age\": 24,\n" +
            "\t\"company\": \"MetLife Inc.\"\n" +
            "}";
    ossClient.putObject(bucketName, key, new ByteArrayInputStream(content.getBytes()));
    SelectObjectRequest selectObjectRequest =
            new SelectObjectRequest(bucketName, key)
                    .withInputSerialization(new InputSerialization()
                            .withCompressionType(CompressionType.NONE)
                            .withJsonInputFormat(new JsonFormat().withJsonType(JsonType.LINES)))
                    .withOutputSerialization(new OutputSerialization()
                            .withCrcEnabled(true)
                            .withJsonOutputFormat(new JsonFormat()))
                    .withExpression("select * from ossobject as s where s.age > 40");
    OSSObject ossObject = ossClient.selectObject(selectObjectRequest);
    // read object content from ossObject
    BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
    while (true) {
        String line = reader.readLine();
        if (line == null) {
            break;
        }
        System.out.println(line);
    }
    reader.close();
```

```
    ossClient.deleteObject(bucketName, key);
  }
}
```

2.8.8. 删除文件

您可以根据需要删除单个文件（Object）、删除指定的多个文件、删除指定前缀的文件或者删除指定目录及目录下的所有文件。

> **警告** 请您谨慎使用删除操作，文件删除后将无法恢复。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写文件完整路径。文件完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除文件或目录。如果要删除目录，目录必须为空。
ossClient.deleteObject(bucketName, objectName);
// 关闭OSSClient。
ossClient.shutdown();
```

批量删除文件

批量删除文件时，每次最多删除1000个文件。您可以删除指定的多个文件、删除指定前缀的文件或者删除指定目录及目录下的所有文件。

OSS还支持通过设置生命周期规则来自动删除文件。更多信息，请参见开发指南中的生命周期规则介绍。

- 参数说明

  请求中的参数说明请参见下表。

参数	描述	方法
Keys	需要删除的文件。	setKeys(List<String>)

参数	描述	方法
quiet	返回结果包括如下两种模式，默认返回模式为详细模式，请根据实际选择返回模式。 ○ 详细模式（verbose）：未设置quiet或者设置quiet为false，表示返回所有删除的文件列表。 ○ 简单模式（quiet）：设置quiet为true，表示只返回删除失败的文件列表。	setQuiet(boolean)
encodingType	对返回的文件名称进行编码。编码类型目前仅支持url。	setEncodingType(String)

返回结果中的参数说明请参见下表。

参数	描述	方法
deletedObjects	删除结果。详细模式下为删除成功的文件列表，简单模式下为删除失败的文件列表。	List<String> getDeletedObjects()
encodingType	deletedObjects中文件名称的编码，返回为空表示没有编码。	getEncodingType()

- 示例
  - 删除指定的多个文件

    以下代码用于删除examplebucket中指定的多个文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除文件。
// 填写需要删除的多个文件完整路径。文件完整路径中不能包含Bucket名称。
List<String> keys = new ArrayList<String>();
keys.add("exampleobjecta.txt");
keys.add("testfolder/sampleobject.txt");
keys.add("exampleobjectb.txt");
DeleteObjectsResult deleteObjectsResult = ossClient.deleteObjects(new DeleteObjectsRequest(bucketName).withKeys(keys));
List<String> deletedObjects = deleteObjectsResult.getDeletedObjects();
// 关闭OSSClient。
ossClient.shutdown();
```

- 删除指定前缀（prefix）的文件

  以下代码用于删除examplebucket中以file为前缀的文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日
常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 指定前缀。
final String prefix = "file";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举所有包含指定前缀的文件并删除。
String nextMarker = null;
ObjectListing objectListing = null;
do {
  ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName)
      .withPrefix(prefix)
      .withMarker(nextMarker);
  objectListing = ossClient.listObjects(listObjectsRequest);
  if (objectListing.getObjectSummaries().size() > 0) {
    List<String> keys = new ArrayList<String>();
    for (OSSObjectSummary s : objectListing.getObjectSummaries()) {
      System.out.println("key name: " + s.getKey());
      keys.add(s.getKey());
    }
    DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName).withKeys(
keys);
    ossClient.deleteObjects(deleteObjectsRequest);
  }
  nextMarker = objectListing.getNextMarker();
} while (objectListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

- 删除指定目录及目录下的所有文件

  以下代码用于删除examplebucket中testdir目录及目录下的所有文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写不包含Bucket名称在内的目录完整路径。例如Bucket下testdir目录的完整路径为testdir/。
final String prefix = "testdir/";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除目录及目录下的所有文件。
String nextMarker = null;
ObjectListing objectListing = null;
do {
  ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName)
      .withPrefix(prefix)
      .withMarker(nextMarker);
  objectListing = ossClient.listObjects(listObjectsRequest);
  if (objectListing.getObjectSummaries().size() > 0) {
    List<String> keys = new ArrayList<String>();
    for (OSSObjectSummary s : objectListing.getObjectSummaries()) {
      System.out.println("key name: " + s.getKey());
      keys.add(s.getKey());
    }
    DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName).withKeys(keys);
    ossClient.deleteObjects(deleteObjectsRequest);
  }
  nextMarker = objectListing.getNextMarker();
} while (objectListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

批量删除文件的完整代码请参见GitHub。

2.8.9. 拷贝文件

本文介绍如何将一个存储空间（源存储空间）中的文件复制到同一地域下相同或不同存储空间（目标存储空间）中。

拷贝文件时注意事项如下：

- 用户必须有源Object的读权限及目标Bucket的读写权限。
- 不支持跨地域拷贝。例如不能将华东1（杭州）地域存储空间中的文件拷贝到华北1（青岛）地域。

拷贝小文件

对于小于1 GB的文件，您可以通过ossClient.copyObject方法拷贝文件。此方法有两种参数指定方式：

参数指定方式	描述
CopyObjectResult copyObject(String sourceBucketName, String sourceKey, String destinationBucketName, String destinationKey)	允许指定源存储空间和源文件，以及目标存储空间和目标文件。拷贝后，目标文件的内容及元信息与源文件相同，称为简单拷贝。
CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)	允许指定目标文件的元信息和拷贝的限制条件。如果拷贝操作的源文件地址和目标文件地址相同，则直接替换源文件的元信息。

CopyObjectRequest可设置的参数请参见下表。

参数	描述	方法
sourceBucketName	源存储空间名称。	setSourceBucketName(String sourceBucketName)
sourceKey	源文件名称。	setSourceKey(String sourceKey)
destinationBucketName	目标存储空间名称。	setDestinationBucketName(String destinationBucketName)
destinationKey	目标文件名称。	setDestinationKey(String destinationKey)
newObjectMetadata	目标文件元信息。	setNewObjectMetadata(ObjectMetadata newObjectMetadata)
matchingETagConstraints	拷贝的限制条件。如果源文件的ETag值和用户提供的ETag值相等，则执行拷贝操作，否则返回错误。	setMatchingETagConstraints(List<String> matchingETagConstraints)
nonmatchingEtagConstraints	拷贝的限制条件。如果源文件的ETag值和用户提供的ETag值不相等，则执行拷贝操作，否则返回错误。	setNonmatchingETagConstraints(List<String> nonmatchingEtagConstraints)
unmodifiedSinceConstraint	拷贝的限制条件。如果传入参数中的时间等于或者晚于文件实际修改时间，则执行拷贝操作，否则返回错误。	setUnmodifiedSinceConstraint(Date unmodifiedSinceConstraint)
modifiedSinceConstraint	拷贝的限制条件。如果源文件在指定的时间之后被修改过，则执行拷贝操作，否则返回错误。	setModifiedSinceConstraint(Date modifiedSinceConstraint)

CopyObjectRequest的参数说明请参见下表。

参数	描述	方法
etag	OSS文件唯一性标识。	String getETag()
lastModified	文件最后修改时间。	Date getLastModified()

拷贝小文件有以下几种拷贝形式。

- 简单拷贝

  以下代码用于通过简单拷贝将srcexamplebucket中的srcexampleobject.txt文件拷贝到目标存储空间desexamplebucket中的desexampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写源Bucket名称。
String sourceBucketName = "srcexamplebucket";
// 填写源Object的完整路径。Object完整路径中不能包含Bucket名称。
String sourceKey = "srcexampleobject.txt";
// 填写与源Bucket处于同一地域的目标Bucket名称。
String destinationBucketName = "desexamplebucket";
// 填写目标Object的完整路径。Object完整路径中不能包含Bucket名称。
String destinationKey = "desexampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 拷贝文件。
CopyObjectResult result = ossClient.copyObject(sourceBucketName, sourceKey, destinationBucketName, destinationKey);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
// 关闭OSSClient。
ossClient.shutdown();
```

- 通过CopyObjectRequest拷贝

  以下代码用于通过CopyObjectRequest将srcexamplebucket中的srcexampleobject.txt文件拷贝到目标存储空间desexamplebucket中的desexampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写源Bucket名称。
String sourceBucketName = "srcexamplebucket";
// 填写源Object的完整路径。Object完整路径中不能包含Bucket名称。
String sourceKey = "srcexampleobject.txt";
// 填写与源Bucket处于同一地域的目标Bucket名称。
String destinationBucketName = "desexamplebucket";
// 填写目标Object的完整路径。Object完整路径中不能包含Bucket名称。
String destinationKey = "desexampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建CopyObjectRequest对象。
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(sourceBucketName, sourceKey, destinationBucketName, destinationKey);
// 设置新的文件元信息。
ObjectMetadata meta = new ObjectMetadata();
meta.setContentType("text/txt");
copyObjectRequest.setNewObjectMetadata(meta);
// 复制文件。
CopyObjectResult result = ossClient.copyObject(copyObjectRequest);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
// 关闭OSSClient。
ossClient.shutdown();
```

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

1. 通过ossClient.initiateMultipartUpload初始化分片拷贝任务。
2. 通过ossClient.uploadPartCopy进行分片拷贝。除最后一个分片外，其它分片都要大于100KB。
3. 通过ossClient.completeMultipartUpload提交分片拷贝任务。

分片拷贝的完整代码请参见GitHub。

以下代码用于通过分片拷贝将srcexamplebucket中的srcexampleobject.txt文件拷贝到目标存储空间desexamplebucket中的desexampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写源Bucket名称。
String sourceBucketName = "srcexamplebucket";
// 填写源Object的完整路径。Object完整路径中不能包含Bucket名称。
String sourceKey = "srcexampleobject.txt";
// 填写与源Bucket处于同一地域的目标Bucket名称。
```

```
// 填写与源Bucket处于同一地域的目标Bucket名称。
String destinationBucketName = "desexamplebucket";
// 填写目标Object的完整路径。Object完整路径中不能包含Bucket名称。
String destinationKey = "desexampleobject.txt";
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(sourceBucketName, sourceKey);
// 获取被拷贝文件的大小。
long contentLength = objectMetadata.getContentLength();
// 设置分片大小为10 MB。单位为字节。
long partSize = 1024 * 1024 * 10;
// 计算分片总数。
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
    partCount++;
}
System.out.println("total part count:" + partCount);
// 初始化拷贝任务。可以通过InitiateMultipartUploadRequest指定目标文件元信息。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(des
tinationBucketName, destinationKey);
InitiateMultipartUploadResult initiateMultipartUploadResult = ossClient.initiateMultipartUpload(initiateMul
tipartUploadRequest);
String uploadId = initiateMultipartUploadResult.getUploadId();
// 分片拷贝。
List<PartETag> partETags = new ArrayList<PartETag>();
for (int i = 0; i < partCount; i++) {
    // 计算每个分片的大小。
    long skipBytes = partSize * i;
    long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;
    // 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
    UploadPartCopyRequest uploadPartCopyRequest =
        new UploadPartCopyRequest(sourceBucketName, sourceKey, destinationBucketName, destinationKey);
    uploadPartCopyRequest.setUploadId(uploadId);
    uploadPartCopyRequest.setPartSize(size);
    uploadPartCopyRequest.setBeginIndex(skipBytes);
    uploadPartCopyRequest.setPartNumber(i + 1);
    UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPartCopy(uploadPartCopyRequest);
    // 将返回的分片ETag保存到partETags中。
    partETags.add(uploadPartCopyResult.getPartETag());
}
// 提交分片拷贝任务。
CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRequ
est(
            destinationBucketName, destinationKey, uploadId, partETags);
ossClient.completeMultipartUpload(completeMultipartUploadRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

关于分片拷贝的更多信息，请参见UploadPartCopy。

2.8.10. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传、拷贝文件及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String content = "Hello OSS!";
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建PutObjectRequest对象。
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
// 指定上传文件操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setHeader("x-oss-forbid-overwrite", "true");
putObjectRequest.setMetadata(metadata);
// 上传文件。
ossClient.putObject(putObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

简单上传的更多详情，请参见PutObject。

拷贝文件

- 通过CopyObjectRequest拷贝

  以下代码用于通过CopyObjectRequest拷贝小文件时禁止覆盖同名文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建CopyObjectRequest对象。
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(sourceBucketName, sourceObjectNa
me, destinationBucketName, destinationObjectName);
// 设置新的文件元信息。
ObjectMetadata metadata = new ObjectMetadata();
meta.setContentType("text/html");
// 指定拷贝文件操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
metadata.setHeader("x-oss-forbid-overwrite", "true");
copyObjectRequest.setNewObjectMetadata(metadata);
// 拷贝文件。
CopyObjectResult result = ossClient.copyObject(copyObjectRequest);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
// 关闭OSSClient。
ossClient.shutdown();
```

- 拷贝大文件

  以下代码用于拷贝大文件（分片拷贝）时禁止覆盖同名文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(sourceBucketName, sourceObjectName);
// 获取被拷贝文件的大小。
long contentLength = objectMetadata.getContentLength();
// 设置分片大小为10MB。
long partSize = 1024 * 1024 * 10;
// 计算分片总数。
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
    partCount++;
}
System.out.println("total part count:" + partCount);
```

```
// 初始化拷贝任务。可以通过InitiateMultipartUploadRequest指定目标文件元信息。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(d
estinationBucketName, destinationObjectName);
// 指定拷贝文件操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setHeader("x-oss-forbid-overwrite", "true");
initiateMultipartUploadRequest.setObjectMetadata(metadata);
InitiateMultipartUploadResult initiateMultipartUploadResult = ossClient.initiateMultipartUpload(initiate
MultipartUploadRequest);
String uploadId = initiateMultipartUploadResult.getUploadId();
// 分片拷贝。
List<PartETag> partETags = new ArrayList<PartETag>();
for (int i = 0; i < partCount; i++) {
    // 计算每个分片的大小。
   long skipBytes = partSize * i;
   long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;
   // 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
   UploadPartCopyRequest uploadPartCopyRequest =
      new UploadPartCopyRequest(sourceBucketName, sourceObjectName, destinationBucketName, desti
nationObjectName);
   uploadPartCopyRequest.setUploadId(uploadId);
   uploadPartCopyRequest.setPartSize(size);
   uploadPartCopyRequest.setBeginIndex(skipBytes);
   uploadPartCopyRequest.setPartNumber(i + 1);
   UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPartCopy(uploadPartCopyRequest);
   // 将返回的分片ETag保存到partETags中。
   partETags.add(uploadPartCopyResult.getPartETag());
}
// 完成分片拷贝任务。
CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRe
quest(
            destinationBucketName, destinationObjectName, uploadId, partETags);
// 指定拷贝文件操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
completeMultipartUploadRequest.addHeader("x-oss-forbid-overwrite", "true");
ossClient.completeMultipartUpload(completeMultipartUploadRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

拷贝文件的更多详情，请参见CopyObject。

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
```

```
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建InitiateMultipartUploadRequest对象。
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectName);
// 指定分片上传操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setHeader("x-oss-forbid-overwrite", "true");
request.setObjectMetadata(metadata);
// 初始化分片。
InitiateMultipartUploadResult upresult = ossClient.initiateMultipartUpload(request);
// 返回uploadId，它是分片上传事件的唯一标识，您可以根据这个ID来发起相关的操作，如取消分片上传、查询分片上传等。
String uploadId = upresult.getUploadId();
// partETags是PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags =  new ArrayList<PartETag>();
// 计算文件有多少个分片。
final long partSize = 1 * 1024 * 1024L;  // 1MB
final File sampleFile = new File("<localFile>");
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
    partCount++;
}
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
    long startPos = i * partSize;
    long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
    InputStream instream = new FileInputStream(sampleFile);
    // 跳过已经上传的分片。
    instream.skip(startPos);
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(objectName);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setInputStream(instream);
    // 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100KB。
    uploadPartRequest.setPartSize(curPartSize);
    // 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出这个范围，OSS将返回InvalidArgument的错误码。
    uploadPartRequest.setPartNumber( i + 1);
    // 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。
    UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
    // 每次上传分片之后，OSS的返回结果会包含一个PartETag。PartETag将被保存到partETags中。
    partETags.add(uploadPartResult.getPartETag());
}
// 创建CompleteMultipartUploadRequest对象。
// 完成分片上传操作时，需要提供所有有效的partETags。OSS收到提交的partETags后，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
```

```
CompleteMultipartUploadRequest completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
// 指定分片上传操作时是否覆盖同名Object。
// 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
// 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
// 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
completeMultipartUploadRequest.addHeader("x-oss-forbid-overwrite", "true");
// 完成上传。
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.completeMultipartUpload(co
mpleteMultipartUploadRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

2.8.11. 解冻文件

归档或冷归档类型的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档和冷归档文件。

> **说明** 非归档或冷归档类型的文件，不要调用RestoreObject方法。
>
> 归档及冷归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

解冻归档文件的完整代码请参见GitHub。

以下代码用于解冻归档文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);
// 校验文件是否为归档文件。
StorageClass storageClass = objectMetadata.getObjectStorageClass();
if (storageClass == StorageClass.Archive) {
  // 解冻文件。
  ossClient.restoreObject(bucketName, objectName);
  // 等待解冻完成。
  do {
    Thread.sleep(1000);
    objectMetadata = ossClient.getObjectMetadata(bucketName, objectName);
  } while (!objectMetadata.isRestoreCompleted());
}
// 获取解冻文件。
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
ossObject.getObjectContent().close();
// 关闭OSSClient。
ossClient.shutdown();
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。
   - 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
   - 批量（Bulk）：表示5~12小时内完成解冻。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourColdArchiveObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 如需创建冷归档类型的文件，请参考以下代码。
// 创建PutObjectRequest对象。
// PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName, new ByteArrayInputStream("<yourContent>".getBytes()));
// 在metadata中指定文件的存储类型为冷归档类型。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setHeader(OSSHeaders.OSS_STORAGE_CLASS, StorageClass.ColdArchive.toString());
// putObjectRequest.setMetadata(metadata);
// 上传文件的同时设置文件的存储类型。
// ossClient.putObject(putObjectRequest);
// 设置解冻冷归档文件的优先级。
// RestoreTier.RESTORE_TIER_EXPEDITED 表示1小时内完成解冻。
// RestoreTier.RESTORE_TIER_STANDARD 表示2~5小时内完成解冻。
// RestoreTier.RESTORE_TIER_BULK 表示5~12小时内完成解冻。
RestoreJobParameters jobParameters = new RestoreJobParameters(restoreTier);
// 配置解冻参数，以设置5小时内解冻完成，解冻状态保持2天为例。
// 第一个参数表示保持解冻状态的天数，默认是1天，此参数适用于解冻Archive（归档）与ColdArchive（冷归档）类型文件。
// 第二个参数jobParameters表示解冻优先级，只适用于解冻ColdArchive类型文件。
RestoreConfiguration configuration = new RestoreConfiguration(2, jobParameters);
// 发起解冻请求。
ossClient.restoreObject(bucketName, objectName, configuration);
// 关闭ossClient。
ossClient.shutdown();
```

2.8.12. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String symLink = "<yourSymLink>";
String destinationObjectName = "<yourDestinationObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建上传文件元信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("text/plain");
// 设置自定义元信息property的值为property-value。
metadata.addUserMetadata("property", "property-value");
// 创建CreateSymlinkRequest。
CreateSymlinkRequest createSymlinkRequest = new CreateSymlinkRequest(bucketName, symLink, destinationObjectName);
// 设置元信息。
createSymlinkRequest.setMetadata(metadata);
// 创建软链接。
ossClient.createSymlink(createSymlinkRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String symLink = "<yourSymLink>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取软链接。
OSSSymlink symbolicLink = ossClient.getSymlink(bucketName, symLink);
// 打印软链接指向的文件内容。
System.out.println(symbolicLink.getSymlink());
System.out.println(symbolicLink.getTarget());
System.out.println(symbolicLink.getRequestId());
// 关闭OSSClient。
ossClient.shutdown();
```

获取软链接的详细信息请参见GetSymlink。

2.9. 版本控制

2.9.1. 管理版本控制

版本控制应用于存储空间（Bucket）内的所有文件（Object）。通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。

版本控制的详情请参见开发指南的版本控制介绍。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置存储空间版本控制状态为Enabled。
BucketVersioningConfiguration configuration = new BucketVersioningConfiguration();
configuration.setStatus(BucketVersioningConfiguration.ENABLED);
SetBucketVersioningRequest request = new SetBucketVersioningRequest(bucketName, configuration);
ossClient.setBucketVersioning(request);
// 关闭OSSClient。
ossClient.shutdown();
```

设置Bucket版本控制状态的详情请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取存储空间版本控制状态信息。
BucketVersioningConfiguration versionConfiguration = ossClient.getBucketVersioning("<yourBucketName>");
System.out.println("bucket versioning status: " + versionConfiguration.getStatus());
// 关闭OSSClient。
ossClient.shutdown();
```

获取Bucket版本控制状态的详情请参见GetBucketVersioning。

2.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本的ID为“null”。

以下代码用于简单上传：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 以上传字符串为例。
String content = "Hello OSS";
PutObjectResult result = ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()));
// 查看此次上传object的版本id。
System.out.println("result.versionid: " + result.getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

> **说明**
>
> - 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
> - 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
> - 不支持对当前版本为非Appendable类型的Object（包括 Normal Object、Delete Marker等）执行AppendObject 操作。

以下代码用于追加上传：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String content1 = "Hello OSS A \n";
String content2 = "Hello OSS B \n";
String content3 = "Hello OSS C \n";
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");
// 通过AppendObjectRequest设置多个参数。
AppendObjectRequest appendObjectRequest = new AppendObjectRequest("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content1.getBytes()),meta);
// 通过AppendObjectRequest设置单个参数。
// 设置存储空间名称。
//appendObjectRequest.setBucketName("<yourBucketName>");
// 设置文件名称。
//appendObjectRequest.setKey("<yourObjectName>");
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为InputStream类型。
//appendObjectRequest.setInputStream(new ByteArrayInputStream(content1.getBytes()));
// 设置待追加的内容。有两种可选类型：InputStream类型和File类型。这里为File类型。
//appendObjectRequest.setFile(new File("<yourLocalFile>"));
// 指定文件的元信息，第一次追加时有效。
//appendObjectRequest.setMetadata(meta);
// 第一次追加。
// 设置文件的追加位置。
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 文件的64位CRC值。此值根据ECMA-182标准计算得出。
System.out.println(appendObjectResult.getObjectCRC());
// 第二次追加。
// nextPosition指明下一次请求中应当提供的Position，即文件当前的长度。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content2.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 第三次追加。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content3.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
/* 步骤1：初始化一个分片上传事件。
*/
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectName);
InitiateMultipartUploadResult upresult = ossClient.initiateMultipartUpload(request);
// 返回uploadId，它是分片上传事件的唯一标识，您可以根据此uploadId来发起相关的操作，如取消分片上传、查询分片上传等。
String uploadId = upresult.getUploadId();
/* 步骤2：上传分片。
*/
// partETags是PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags =  new ArrayList<PartETag>();
// 计算文件有多少个分片。
final long partSize = 1 * 1024 * 1024L;  // 1MB
final File sampleFile = new File("<localFile>");
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
   partCount++;
 }
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
   long startPos = i * partSize;
   long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
   InputStream instream = new FileInputStream(sampleFile);
   // 跳过已经上传的分片。
   instream.skip(startPos);
   UploadPartRequest uploadPartRequest = new UploadPartRequest();
   uploadPartRequest.setBucketName(bucketName);
   uploadPartRequest.setKey(objectName);
   uploadPartRequest.setUploadId(uploadId);
   uploadPartRequest.setInputStream(instream);
   // 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100 KB。
   uploadPartRequest.setPartSize(curPartSize);
   // 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出这个范围，OSS将返回InvalidArgument的错误码。
   uploadPartRequest.setPartNumber( i + 1);
 // 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。
   UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
 // 每次上传分片之后，OSS的返回结果会包含一个PartETag。PartETag将被保存到partETags中。
   partETags.add(uploadPartResult.getPartETag());
}
/* 步骤3：完成分片上传。
*/
// 在执行该操作时，需要提供所有有效的partETags。OSS收到提交的partETags后，会逐一验证每个分片的有效性。
```

```
当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
CompleteMultipartUploadRequest completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.completeMultipartUpload(co
mpleteMultipartUploadRequest);
// 查看上传文件的版本id。
System.out.println("restore object versionid: " + completeMultipartUploadResult.getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

分片上传的详细信息请参见CompleteMultipartUpload。

2.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String versionid = "<yourObjectVersionid>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 封装GetObject请求。
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
// 指定object版本id。
getObjectRequest.setVersionId(versionid);
// ossObject包含文件所在的存储空间名称、文件名称、文件元信息以及一个输入流。
OSSObject ossObject = ossClient.getObject(getObjectRequest);
// 查看下载文件的版本id。
System.out.println("Get Object versionid:" +  ossObject.getObjectMetadata().getVersionId());
// 读取文件内容。
System.out.println("Object content:");
BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
while (true) {
  String line = reader.readLine();
  if (line == null) break;
  System.out.println("\n" + line);
}
// 关闭OSSClient。
ossClient.shutdown();
```

下载文件的详细信息请参见GetObject。

2.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的版本ID，此版本ID将会在响应header 的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为”null“的版本，且会覆盖原先versionId为”null“的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";
String versionid = "<yourSourceObjectVersionid>";
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(sourceBucketName, sourceObjectName,
destinationBucketName, destinationObjectName);
copyObjectRequest.setSourceVersionId(versionid);
CopyObjectResult copyObjectResult = ossClient.copyObject(copyObjectRequest);
System.out.println("ETag: " + copyObjectResult.getETag() + " LastModified: " + copyObjectResult.getLastMo
dified());
System.out.println("dest object versionid: " + copyObjectResult.getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String sourceBucketName = "<yourSourceBucketName>";
String sourceObjectName = "<yourSourceObjectName>";
String destinationBucketName = "<yourDestinationBucketName>";
String destinationObjectName = "<yourDestinationObjectName>";
String sourceObjectVersionid = "<yourSourceObjectVersionid>";
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(sourceBucketName, sourceObjectName, so
urceObjectVersionid);
```

```
// 获取被拷贝文件的大小。
long contentLength = objectMetadata.getContentLength();
// 设置分片大小为10 MB。
long partSize = 1024 * 1024 * 10;
// 计算分片总数。
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
   partCount++;
}
System.out.println("total part count:" + partCount);
// 初始化拷贝任务。可以通过InitiateMultipartUploadRequest指定目标文件元信息。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(des
tinationBucketName, destinationObjectName);
InitiateMultipartUploadResult initiateMultipartUploadResult = ossClient.initiateMultipartUpload(initiateMul
tipartUploadRequest);
String uploadId = initiateMultipartUploadResult.getUploadId();
// 分片拷贝。
List<PartETag> partETags = new ArrayList<PartETag>();
for (int i = 0; i < partCount; i++) {
    // 计算每个分片的大小。
   long skipBytes = partSize * i;
   long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;
   // 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
   UploadPartCopyRequest uploadPartCopyRequest =
      new UploadPartCopyRequest(sourceBucketName, sourceObjectName, destinationBucketName, destinat
ionObjectName);
   uploadPartCopyRequest.setUploadId(uploadId);
   uploadPartCopyRequest.setPartSize(size);
   uploadPartCopyRequest.setBeginIndex(skipBytes);
   uploadPartCopyRequest.setPartNumber(i + 1);
   // 指定源文件的versionId。
   uploadPartCopyRequest.setSourceVersionId(sourceObjectVersionid);
   UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPartCopy(uploadPartCopyRequest);
   // 将返回的分片ETag保存到partETags中。
   partETags.add(uploadPartCopyResult.getPartETag());
}
// 提交分片拷贝任务。
CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRequ
est(
            destinationBucketName, destinationObjectName, uploadId, partETags);
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.completeMultipartUpload(co
mpleteMultipartUploadRequest);
// 查看目标文件的versionId。
System.out.println("object versionid: " + completeMultipartUploadResult.getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

分片拷贝的详细信息请参见UploadPartCopy。

2.9.5. 列举文件

本文介绍如何在开启版本控制状态下列举存储空间下（Bucket）的所有文件（Object）、指定个数的文件、指定前缀的文件等。

列举Bucket中所有Object的信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举包括删除标记在内的所有Object的版本信息。
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker);
  versionListing = ossClient.listVersions(listVersionsRequest);
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

列举指定前缀Object的版本信息

以下代码用于列举指定前缀Object的版本信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定列举以"test-"为前缀的Object的版本信息。
String prefix = "test-";
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker)
      .withEncodingType(OSSConstants.URL_ENCODING)
      .withPrefix(prefix);
  versionListing = ossClient.listVersions(listVersionsRequest);
  // 查看Object的版本信息。
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

列举指定个数Object的版本信息

以下代码用于列举指定个数Object的版本信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定最多返回200个结果。
ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
        .withBucketName(bucketName)
        .withMaxResults(200);
versionListing = ossClient.listVersions(listVersionsRequest);
for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
   System.out.println("key name: " + ossVersion.getKey());
   // 在未开启版本控制的情况下，VersionId为none。
   System.out.println("versionid: " + ossVersion.getVersionId());
   System.out.println("Is latest: " + ossVersion.isLatest());
   System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
}
// 关闭OSSClient。
ossClient.shutdown();
```

分页列举所有Object的版本信息

以下代码用于分页列举所有Object的版本信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 分页列举所有Object的版本信息。
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker);
  versionListing = ossClient.listVersions(listVersionsRequest);
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

列举名称中包含特殊字符的Object

如果文件名称含有以下特殊字符，需要进行编码传输。OSS目前仅支持URL编码。

- 单引号（' '）
- 双引号（" "）
- and符号（&）
- 尖括号（< >）
- 顿号（、）
- 中文

以下代码用于列举名称中包含特殊字符的Object：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定返回结果以URL编码进行传输。
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
        .withBucketName(bucketName)
        .withKeyMarker(nextKeyMarker)
        .withVersionIdMarker(nextVersionMarker)
        .withEncodingType(OSSConstants.URL_ENCODING)
  versionListing = ossClient.listVersions(listVersionsRequest);
   // 查看Object的版本信息。
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为Object。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

假设存储空间中包含文件*oss.jpg*、*fun/test.jpg*、*fun/movie/001.avi*和*fun/movie/007.avi*，以正斜线（/）作为文件夹的分隔符。以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举根目录下的Object的版本信息

  以下代码用于列举根目录下的Object的版本信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 指定delimiter参数为正斜线（/），将会列举根目录下的Object的版本信息以及文件夹名称。
String delimiter = "/";
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker)
      .withEncodingType(OSSConstants.URL_ENCODING)
      .withDelimiter(delimiter);
  versionListing = ossClient.listVersions(listVersionsRequest);
  // 查看Object的版本信息。
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  // 查看根目录下以正斜线（/）结尾的文件夹名称。
  for (String commonPrefix : versionListing.getCommonPrefixes()) {
    System.out.println("commonPrefix: " + commonPrefix);
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

返回结果

```
key name: oss.jpg
versionid: CAEQEhiBgMCw8Y7FqBciIGIzMDE3MTEzOWRiMDRmZmFhMmRlMjljZWI0MWU4****
Is latest: true
Is delete marker: false
commonPrefix: fun/
```

- 列举目录下的文件和子目录

  以下代码用于列举指定目录下的文件和子目录：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置prefix参数来获取fun目录下的所有文件与文件夹，同时设置delimiter参数为正斜线（/）作为文件夹的分隔符。
String prefix = "fun/";
String delimiter = "/";
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker)
      .withEncodingType(OSSConstants.URL_ENCODING)
      .withDelimiter(delimiter)
      .withPrefix(prefix);
  versionListing = ossClient.listVersions(listVersionsRequest);
  // 查看Object的版本信息。
  for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
    System.out.println("key name: " + ossVersion.getKey());
    System.out.println("versionid: " + ossVersion.getVersionId());
    System.out.println("Is latest: " + ossVersion.isLatest());
    System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
  }
  // 查看fun目录下的子目录名称。
  for (String commonPrefix : versionListing.getCommonPrefixes()) {
    System.out.println("commonPrefix: " + commonPrefix);
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

返回结果：

```
key name: fun/test.jpg
versionid: CAEQEhiBgIC48Y7FqBciIDU4NjAzMjczMTY5NDRjYmVhNGY4NTM2YTdjY2Ji****
Is latest: true
Is delete marker: false
commonPrefix: fun/movie/
```

2.9.6. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

  如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

  `x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object进行永久删除：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，例如example/test.jpg。
String objectName = "<yourObjectName>";
// 可以传入Object的版本id，也可以传入删除标记的版本id。
String versionid = "<yourObjectVersionidOrDelMarkerVersionid>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除指定版本的Object。
ossClient.deleteVersion(bucketName, objectName , versionid);
// 关闭OSSClient。
ossClient.shutdown();
```

- 临时删除

  以下代码用于不指定versionId对Object进行临时删除：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
// 开启版本控制状态下，此方法为临时删除，将会给Object添加删除标记。
ossClient.deleteObject(bucketName, objectName);
// 关闭OSSClient。
ossClient.shutdown();
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于指定versionId对多个Object及删除标记进行永久删除：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除指定版本的Object与删除标记。
List<KeyVersion> keyVersionsList = new ArrayList<KeyVersion>();
keyVersionsList.add(new KeyVersion("<yourObject3Name>","<yourObject3NameVersionid>"));
keyVersionsList.add(new KeyVersion("<yourObject4Name>","<yourObject4DelMarkerVersionid>"));
DeleteVersionsRequest delVersionsRequest = new DeleteVersionsRequest(bucketName);
delVersionsRequest.setKeys(keyVersionsList);
// 发起deleteVersions请求。
DeleteVersionsResult delVersionsResult = ossClient.deleteVersions(delVersionsRequest);
// 查看删除结果。
for (DeletedVersion delVer : delVersionsResult.getDeletedVersions()) {
 String keyName = URLDecoder.decode(delVer.getKey(), "UTF-8");
 String keyVersionId = delVer.getVersionId();
 String keyDelMarkerId = delVer.getDeleteMarkerVersionId();
 System.out.println("delete key: " + keyName);
 System.out.println("delete key versionid: " + keyVersionId);
 if(keyDelMarkerId != null && keyDelMarkerId.length() != 0){
  System.out.println("delete key del_marker versionid: " + keyDelMarkerId);
 }
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String versionid = "<yourObjectVersionid>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 批量删除Object。
List<String> KeysList = new ArrayList<String>();
KeysList.add("<yourObject1Name>");
KeysList.add("<yourObject2Name>");
DeleteObjectsRequest request = new DeleteObjectsRequest(bucketName);
request.setKeys(KeysList);
// 发起deleteObjects请求。
DeleteObjectsResult delObjResult = ossClient.deleteObjects(request);
// 查看删除结果。
for (String o : delObjResult.getDeletedObjects()) {
    String keyName = URLDecoder.decode(o, "UTF-8");
    System.out.println("delete key name: " + keyName);
}
// 关闭OSSClient。
ossClient.shutdown();
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

删除指定前缀（prefix）的文件

以下代码用于删除指定前缀的文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 指定前缀。
final String prefix = "<yourkeyPrefix>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 列举所有指定前缀文件的版本信息并删除这些文件。
String nextKeyMarker = null;
String nextVersionMarker = null;
VersionListing versionListing = null;
do {
  ListVersionsRequest listVersionsRequest = new ListVersionsRequest()
      .withBucketName(bucketName)
      .withKeyMarker(nextKeyMarker)
      .withVersionIdMarker(nextVersionMarker)
      .withPrefix(prefix);
  versionListing = ossClient.listVersions(listVersionsRequest);
  if (versionListing.getVersionSummaries().size() > 0) {
    List<KeyVersion> keyVersionsList = new ArrayList<KeyVersion>();
    for (OSSVersionSummary ossVersion : versionListing.getVersionSummaries()) {
      System.out.println("key name: " + ossVersion.getKey());
      System.out.println("versionid: " + ossVersion.getVersionId());
      System.out.println("Is delete marker: " + ossVersion.isDeleteMarker());
      keyVersionsList.add(new KeyVersion(ossVersion.getKey(), ossVersion.getVersionId()));
    }
    DeleteVersionsRequest delVersionsRequest = new DeleteVersionsRequest(bucketName).withKeys(keyVersionsList);
    ossClient.deleteVersions(delVersionsRequest);
  }
  nextKeyMarker = versionListing.getNextKeyMarker();
  nextVersionMarker = versionListing.getNextVersionIdMarker();
} while (versionListing.isTruncated());
// 关闭OSSClient。
ossClient.shutdown();
```

2.9.7. 解冻文件

在受版本控制的存储空间（Bucket）中，Object的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 指定归档类文件的版本id
String versionid = "<yourArchiveObjectVersionid>";
// 获取指定版本的文件的部分元信息。
GenericRequest getObjectMetadataRequest = new GenericRequest(bucketName,objectName, versionid);
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(getObjectMetadataRequest);
// 校验指定版本的文件是否为归档文件。
StorageClass storageClass = objectMetadata.getObjectStorageClass();
System.out.println("object storage class:" + objectMetadata.getObjectStorageClass());
if (storageClass == StorageClass.Archive) {
  // 解冻文件。
  GenericRequest genericRequest = new GenericRequest(bucketName, objectName, versionid);
  RestoreObjectResult restoreObjectResult = ossClient.restoreObject(genericRequest);
  System.out.println("restor versionid: " + restoreObjectResult.getVersionId());
  // 等待解冻完成。
  do {
    Thread.sleep(1000);
    objectMetadata = ossClient.getObjectMetadata(getObjectMetadataRequest);
    System.out.println("is competed:" + objectMetadata.isRestoreCompleted());
  } while (!objectMetadata.isRestoreCompleted());
}
// 获取指定版本解冻文件。
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
getObjectRequest.setVersionId(versionid);
OSSObject ossObject = ossClient.getObject(getObjectRequest);
ossObject.getObjectContent().close();
```

解冻文件的详细信息请参见RestoreObject。

2.9.8. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的meta信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的meta信息。

以下代码用于获取文件元信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取指定版本文件的部分元信息。
GenericRequest getSimplifiedObjectMetaRequest = new GenericRequest("<yourBucketName>", "<yourObjectName>", "<yourVersionId>");
SimplifiedObjectMeta objectMeta = ossClient.getSimplifiedObjectMeta(getSimplifiedObjectMetaRequest);
System.out.println(objectMeta.getSize());
System.out.println(objectMeta.getETag());
System.out.println(objectMeta.getLastModified());
// 获取指定版本文件的全部元信息。
GenericRequest getObjectMetadataRequest = new GenericRequest("<yourBucketName>", "<yourObjectName>", "<yourObjectVersionid>");
ObjectMetadata metadata = ossClient.getObjectMetadata(getObjectMetadataRequest);
System.out.println(metadata.getContentType());
System.out.println(metadata.getLastModified());
System.out.println(metadata.getExpirationTime());
// 关闭OSSClient。
ossClient.shutdown();
```

获取文件元信息的详情请参见HeadObject、GetObjectMeta。

2.9.9. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String versionid = "<yourObjectVersionid>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建SetObjectAclRequest对象，此示例中设置文件的访问权限为公共读。
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest(bucketName, objectName,
    versionid, CannedAccessControlList.PublicRead);
// 设置指定版本文件的权限。
ossClient.setObjectAcl(setObjectAclRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（DeleteMarker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String versionid = "<yourObjectVersionid>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取文件的访问权限。
GenericRequest genericRequest = new GenericRequest(bucketName, objectName, versionid);
ObjectAcl objectAcl = ossClient.getObjectAcl(genericRequest);
System.out.println("get object acl: " + objectAcl.getPermission().toString());
System.out.println("object versionid: " + objectAcl.getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

获取文件访问权限的详细信息请参考GetObjectACL。

2.9.10. 管理软链接

本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

您可以通过创建软链接指向Target Object的当前版本。

> **说明** 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

软链接本身也可以有多个版本，每个不同的版本可以指向不同的Target Object，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。

以下代码用于创建软链接：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String symLink = "<yourSymLink>";
String destinationObjectName = "<yourDestinationObjectName>";
// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);
// 创建上传文件元信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("text/plain");
// 设置自定义元信息property的值为property-value。
metadata.addUserMetadata("property", "property-value");
// 创建CreateSymlinkRequest。
CreateSymlinkRequest createSymlinkRequest = new CreateSymlinkRequest(bucketName, symLink, destinationObjectName);
// 设置元信息。
createSymlinkRequest.setMetadata(metadata);
// 创建软链接。
ossClient.createSymlink(createSymlinkRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
String symLink = "<yourSymLink>";
String versionid = "<yourSymLinkVersionid>";
// 获取指定版本的软链接的内容。
GenericRequest genericRequest = new GenericRequest(bucketName, symLink, versionid);
OSSSymlink symbolicLink = ossClient.getSymlink(genericRequest);
// 打印软链接指向的文件内容。
System.out.println(symbolicLink.getSymlink());
System.out.println(symbolicLink.getTarget());
System.out.println(symbolicLink.getRequestId());
// 打印软链接的版本id
System.out.println(symbolicLink.getMetadata().getVersionId());
// 关闭OSSClient。
ossClient.shutdown();
```

获取软链接的详细信息请参见GetSymlink。

2.10. 对象标签

2.10.1. 设置对象标签

OSS支持使用对象标签（Object Tagging）对存储空间（Bucket）中的文件（Object）进行分类，您可以针对相同标签的Object设置生命周期规则、访问权限等。

背景信息

设置对象标签时，请注意以下事项：

- 您可以在上传Object时设置对象标签，也可以对已上传Object设置对象标签。设置对象标签时，如果对象已有标签，则覆盖原标签。关于设置对象标签的更多信息，请参见PutObjectTagging。
- 设置对象标签时，您要有PutObjectTagging权限。
- 更改标签时不会更新Object的Last-Modified时间。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/

> **说明** 通过HTTP header的方式设置标签且标签中包含任意字符时，您可以对标签的Key和Value做URL编码。

对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见对象标签。

上传Object时添加对象标签

以下分别介绍简单上传、分片上传、追加上传以及断点续传上传场景下为上传的Object添加对象标签的示例。

- 简单上传时添加对象标签

  以下代码用于简单上传时添加对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
// 在HTTP header中设置标签信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags);
// 上传文件的同时设置标签信息。
String content = "<yourtContent>";
ossClient.putObject(bucketName, objectName, new ByteArrayInputStream(content.getBytes()), metadata);
// 关闭OSSClient。
ossClient.shutdown();
```

- 分片上传时添加对象标签

  以下代码用于分片上传时添加对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
String localFile = "D:\\localpath\\examplefile.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
/*
  步骤1：初始化一个分片上传事件。
*/
```

```
// 在HTTP header中设置标签信息。
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags);
// 发起InitiateMultipartUploadRequest请求的同时设置标签信息。
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectName
, metadata);
InitiateMultipartUploadResult result = ossClient.initiateMultipartUpload(request);
// 返回uploadId，它是分片上传事件的唯一标识。您可以根据该ID来发起相关的操作，例如取消分片上传、查询分片
上传等。
String uploadId = result.getUploadId();
/*
  步骤2：上传分片。
*/
// partETags是PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags =  new ArrayList<PartETag>();
// 计算文件有多少个分片。
final long partSize = 1 * 1024 * 1024L;  // 1 MB。
final File sampleFile = new File(localFile);
long fileLength = sampleFile.length();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
   partCount++;
 }
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
   long startPos = i * partSize;
   long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
   InputStream instream = null;
   try {
      instream = new FileInputStream(sampleFile);
      // 跳过已上传的分片。
      instream.skip(startPos);
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
   UploadPartRequest uploadPartRequest = new UploadPartRequest();
   uploadPartRequest.setBucketName(bucketName);
   uploadPartRequest.setKey(objectName);
   uploadPartRequest.setUploadId(uploadId);
   uploadPartRequest.setInputStream(instream);
   // 设置分片大小。除了最后一个分片没有大小限制，其他的分片最小为100 KB。
   uploadPartRequest.setPartSize(curPartSize);
   // 设置分片号。每一个上传的分片都有一个分片号，取值范围是1~10000，如果超出该范围，OSS将返回InvalidAr
gument的错误码。
   uploadPartRequest.setPartNumber( i + 1);
   // 每个分片不需要按顺序上传，甚至可以在不同客户端上传，OSS会按照分片号排序组成完整的文件。
   UploadPartResult uploadPartResult = ossClient.uploadPart(uploadPartRequest);
   // 每次上传分片之后，OSS的返回结果会包含一个PartETag。PartETag将被保存到partETags中。
```

```
    partETags.add(uploadPartResult.getPartETag());
}
/* 步骤3：完成分片上传。
*/
// partETags必须按分片号升序排列。
Collections.sort(partETags, new Comparator<PartETag>() {
    public int compare(PartETag p1, PartETag p2) {
        return p1.getPartNumber() - p2.getPartNumber();
    }
});
// 在执行该操作时，需要提供所有有效的partETags。OSS收到提交的partETags后，会逐一验证每个分片的有效性
。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
ossClient.completeMultipartUpload(completeMultipartUploadRequest);
// 查看文件标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, objectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("object tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

- 追加上传时添加对象标签

  以下代码用于追加上传时添加对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
String content1 = "Hello OSS A \n";
String content2 = "Hello OSS B \n";
String content3 = "Hello OSS C \n";
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
ObjectMetadata meta = new ObjectMetadata();
// 设置上传文件的标签。
 meta.setObjectTagging(tags);
// 指定上传的内容类型。
meta.setContentType("text/plain");
// 通过AppendObjectRequest设置多个参数。
AppendObjectRequest appendObjectRequest = new AppendObjectRequest(bucketName, objectName, ne
w ByteArrayInputStream(content1.getBytes()), meta);
// 通过AppendObjectRequest设置单个参数。
//appendObjectRequest.setBucketName(bucketName);
```

```
//appendObjectRequest.setBucketName(bucketName);
//appendObjectRequest.setKey(objectName);
// 设置待追加的内容。可选类型包括InputStream类型和File类型两种。此处为InputStream类型。
//appendObjectRequest.setInputStream(new ByteArrayInputStream(content1.getBytes()));
// 设置待追加的内容。可选类型包括InputStream类型和File类型两种。此处为File类型。
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
//appendObjectRequest.setFile(new File("D:\\localpath\\examplefile.txt"));
// 指定文件的元信息，第一次追加时有效。
//appendObjectRequest.setMetadata(meta);
// 第一次追加。只有第一次追加上传时设置的标签生效。
// 设置文件的追加位置。
appendObjectRequest.setPosition(0L);
AppendObjectResult appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 文件的64位CRC值。此值根据ECMA-182标准计算得出。
System.out.println(appendObjectResult.getObjectCRC());
// 第二次追加。
// nextPosition表示下一次请求中应当提供的Position，即文件当前的长度。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content2.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 第三次追加。
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
appendObjectRequest.setInputStream(new ByteArrayInputStream(content3.getBytes()));
appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 查看上传文件的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, objectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("object tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

- 断点续传上传时添加对象标签

  以下代码用于断点续传上传时添加对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
String localFile = "D:\\localpath\\examplefile.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置文件的标签信息。
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
ObjectMetadata meta = new ObjectMetadata();
```

```
ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");
// 设置文件标签。
meta.setObjectTagging(tags);
// 通过UploadFileRequest设置多个参数。
UploadFileRequest uploadFileRequest = new UploadFileRequest(bucketName,objectName);
// 通过UploadFileRequest设置单个参数。
// 指定Bucket名称。
//uploadFileRequest.setBucketName(bucketName);
// 指定Object完整路径。
//uploadFileRequest.setKey(objectName);
// 指定待上传的本地文件。
uploadFileRequest.setUploadFile(localFile);
// 指定上传并发线程数，默认值为1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小，取值范围为100 KB~5 GB，默认上传的分片大小是整个文件大小的1/10000。
uploadFileRequest.setPartSize(1 * 1024 * 1024);
// 开启断点续传，默认为关闭。
uploadFileRequest.setEnableCheckpoint(true);
// 记录本地分片上传结果的文件。开启断点续传功能时需要设置此参数，上传过程中的进度信息会保存在该文件中，
如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。默认与待上传
的本地文件同目录，为uploadFile.ucp。
uploadFileRequest.setCheckpointFile("yourCheckpointFile");
// 文件的元数据。
uploadFileRequest.setObjectMetadata(meta);
// 设置上传成功回调，参数为Callback类型。
//uploadFileRequest.setCallback("yourCallbackEvent");
// 断点续传上传, 同时设置文件标签。
ossClient.uploadFile(uploadFileRequest);
// 查看文件的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, objectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("object tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

为已上传Object添加或更改对象标签

如果上传Object时未添加对象标签或者添加的对象标签不满足使用需求，您可以在上传Object后为Object添加或更改对象标签。

以下代码用于为已上传Object添加或更改对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
// 为文件设置标签。
ossClient.setObjectTagging(bucketName, objectName, tags);
// 关闭OSSClient。
ossClient.shutdown();
```

为Object指定版本添加或更改对象标签

在已开启版本控制的Bucket中，通过指定Object的版本ID（versionId），您可以为Object指定版本添加或更改对象标签。

以下代码用于为Object指定版本添加或更改对象标签。

> **说明** 关于获取versionId的具体操作，请参见列举文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写Object的版本ID，例如CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****
。
String versionId = "CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
Map<String, String> tags = new HashMap<String, String>(1);
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
SetObjectTaggingRequest setObjectTaggingRequest = new SetObjectTaggingRequest(bucketName, objectN
ame, tags);
setObjectTaggingRequest.setVersionId(versionId);
ossClient.setObjectTagging(setObjectTaggingRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 简单拷贝时设置对象标签

  以下代码用于简单拷贝1 GB以下的Object时设置对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写源Bucket名称，例如srcexamplebucket。
String sourceBucketName = "srcexamplebucket";
// 填写源Object完整路径，例如srcexampledir/exampleobject.txt。
String sourceObjectName = "srcexampledir/exampleobject.txt";
// 填写目标Bucket名称，例如destexamplebucket。
String destinationBucketName = "destexamplebucket";
// 填写目标Object完整路径，例如destexampledir/exampleobject.txt。
String destinationObjectName = "destexampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建CopyObjectRequest对象。
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(sourceBucketName, sourceObjectNa
me, destinationBucketName, destinationObjectName);
// 设置目标文件的标签。如果不设置headers，目标文件默认复制源文件的标签。
Map<String, String> headers = new HashMap<String, String>();
headers.put(OSSHeaders.COPY_OBJECT_TAGGING_DIRECTIVE, "REPLACE");
headers.put(OSSHeaders.OSS_TAGGING, "key1=value1&key2=value2");
copyObjectRequest.setHeaders(headers);
// 复制文件。
CopyObjectResult result = ossClient.copyObject(copyObjectRequest);
System.out.println("ETag: " + result.getETag() + " LastModified: " + result.getLastModified());
// 查看目标文件的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, destinationObjectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("dest object tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

- 分片拷贝时设置对象标签

  以下代码用于分片拷贝1 GB以上的Object时设置对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写源Bucket名称，例如srcexamplebucket。
String sourceBucketName = "srcexamplebucket";
// 填写源Object完整路径，例如srcexampledir/exampleobject.txt。
String sourceObjectName = "srcexampledir/exampleobject.txt";
// 填写目标Bucket名称，例如destexamplebucket。
String destinationBucketName = "destexamplebucket";
// 填写目标Object完整路径，例如destexampledir/exampleobject.txt。
String destinationObjectName = "destexampledir/exampleobject.txt";
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(sourceBucketName, sourceObjectName);
```

```
// 获取被拷贝文件的大小。
long contentLength = objectMetadata.getContentLength();
// 设置分片大小为10 MB。
long partSize = 1024 * 1024 * 10;
// 计算分片总数。
int partCount = (int) (contentLength / partSize);
if (contentLength % partSize != 0) {
   partCount++;
}
System.out.println("total part count:" + partCount);
// 在HTTP header中设置标签信息。
Map<String, String> tags2 = new HashMap<String, String>();
// 依次填写对象标签的键（owner）和值（例如Lily）。
tags2.put("owner", "Lily");
tags2.put("type", "document");
ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags2);
// 初始化拷贝任务。同时给要拷贝的目标文件设置标签。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(d
estinationBucketName, destinationObjectName, metadata);
InitiateMultipartUploadResult initiateMultipartUploadResult = ossClient.initiateMultipartUpload(initiate
MultipartUploadRequest);
String uploadId = initiateMultipartUploadResult.getUploadId();
// 分片拷贝。
List<PartETag> partETags = new ArrayList<PartETag>();
for (int i = 0; i < partCount; i++) {
    // 计算每个分片的大小。
   long skipBytes = partSize * i;
   long size = partSize < contentLength - skipBytes ? partSize : contentLength - skipBytes;
   // 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
   UploadPartCopyRequest uploadPartCopyRequest =
      new UploadPartCopyRequest(sourceBucketName, sourceObjectName, destinationBucketName, desti
nationObjectName);
   uploadPartCopyRequest.setUploadId(uploadId);
   uploadPartCopyRequest.setPartSize(size);
   uploadPartCopyRequest.setBeginIndex(skipBytes);
   uploadPartCopyRequest.setPartNumber(i + 1);
   UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPartCopy(uploadPartCopyRequest);
   // 将返回的分片ETag保存到partETags中。
   partETags.add(uploadPartCopyResult.getPartETag());
}
// 提交分片拷贝任务。
CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRe
quest(
            destinationBucketName, destinationObjectName, uploadId, partETags);
CompleteMultipartUploadResult completeMultipartUploadResult = ossClient.completeMultipartUpload(c
ompleteMultipartUploadRequest);
// 查看源文件的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, sourceObjectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("src object tagging: "+ getTags.toString());
// 查看目标文件的标签信息。
tagSet = ossClient.getObjectTagging(bucketName, destinationObjectName);
getTags = tagSet.getAllTags();
```

```
System.out.println("dest object tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

为软链接文件设置对象标签

以下代码用于为软链接文件设置对象标签。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写软链接完整路径，例如shortcut/myobject.txt。
String symLink = "shortcut/myobject.txt";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String destinationObjectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置软链接的标签信息。
Map<String, String> tags = new HashMap<String, String>();
// 依次填写对象标签的键（例如owner）和值（例如John）。
tags.put("owner", "John");
tags.put("type", "document");
// 创建上传文件元信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags);
// 创建CreateSymlinkRequest。
CreateSymlinkRequest createSymlinkRequest = new CreateSymlinkRequest(bucketName, symLink, destinat
ionObjectName);
// 设置元信息。
createSymlinkRequest.setMetadata(metadata);
// 创建软链接。
ossClient.createSymlink(createSymlinkRequest);
// 查看软链接的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, symLink);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("symLink tagging: "+ getTags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

2.10.2. 获取对象标签

设置对象标签后，您可以根据需要获取Object的标签信息。当存储空间（Bucket）已开启版本控制时，OSS默认只获取Object当前版本的标签信息，您可以通过指定Object的版本ID（versionId）来获取Object指定版本的标签信息。

说明

- 对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见开发指南中的对象标签。
- 关于获取对象标签的更多信息，请参见GetObjectTagging。

获取Object标签信息

当存储空间（Bucket）未开启版本控制时，您可以根据需要获取Object标签信息。当Bucket已开启版本控制时，OSS默认只获取Object当前版本标签信息。

以下代码用于获取目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件的标签信息。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取文件的标签。
TagSet tagSet = ossClient.getObjectTagging(bucketName, objectName);
Map<String, String> tags = tagSet.getAllTags();
// 查看标签信息。
System.out.println("object tagging: "+ tags.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

获取Object指定版本标签信息

当Bucket已开启版本控制时，通过指定Object的版本ID（versionId），您可以获取Object指定版本标签信息。

以下代码用于获取目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件指定版本的标签信息。

说明 关于获取versionId的具体操作，请参见列举文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写Object的版本ID，例如CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****
。
String versionId = "CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
GenericRequest genericRequest = new GenericRequest(bucketName, objectName, versionId);
TagSet tagSet = ossClient.getObjectTagging(genericRequest);
// 查看标签信息。
System.out.println("object tagging: "+ tagSet.toString());
// 关闭OSSClient。
ossClient.shutdown();
```

2.10.3. 删除对象标签

您可以根据需要删除不需要的标签信息。当存储空间（Bucket）已开启版本控制时，OSS默认只删除Object当前版本的标签信息，您可以通过指定Object的版本ID（versionId）来删除Object指定版本标签信息。

> **说明**
> - 对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见开发指南中的对象标签。
> - 关于删除对象标签的更多信息，请参见DeleteObjectTagging。

删除Object标签信息

当存储空间（Bucket）未开启版本控制时，您可以根据需要删除Object标签信息。当Bucket已开启版本控制时，OSS默认只删除Object当前版本标签信息。

以下代码用于删除目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件的对象标签信息。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除文件的标签信息。
ossClient.deleteObjectTagging(bucketName, objectName);
```

删除Object指定版本标签信息

当Bucket已开启版本控制时，通过指定Object的版本ID，您可以删除Object指定版本的对象标签。

以下代码用于删除目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件指定版本的对象标签信息。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampledir/exampleobject.txt";
// 填写Object的版本ID，例如CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****
。
String versionId = "CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
GenericRequest genericRequest = new GenericRequest(bucketName, objectName, versionId);
ossClient.deleteObjectTagging(genericRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

2.10.4. 对象标签和生命周期管理

生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为生命周期规则生效的条件。

> **说明** 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于生命周期规则中添加标签匹配规则：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建SetBucketLifecycleRequest
SetBucketLifecycleRequest request = new SetBucketLifecycleRequest(bucketName);
// 设置规则ID、文件前缀与标签。
String ruleId0 = "rule0";
String matchPrefix0 = "A0/";
Map<String, String> matchTags0 = new HashMap<String, String>();
matchTags0.put("key0", "value0");
String ruleId1 = "rule1";
String matchPrefix1 = "A1/";
Map<String, String> matchTags1 = new HashMap<String, String>();
matchTags1.put("key1", "value1");
String ruleId2 = "rule2";
String matchPrefix2 = "A2/";
String ruleId3 = "rule3";
String matchPrefix3 = "A3/";
// 距最后修改时间3天后过期。
LifecycleRule rule = new LifecycleRule(ruleId0, matchPrefix0, RuleStatus.Enabled, 3);
rule.setTags(matchTags0)
request.AddLifecycleRule(rule);
// 指定日期之前创建的文件过期。
rule = new LifecycleRule(ruleId1, matchPrefix1, RuleStatus.Enabled);
rule.setCreatedBeforeDate(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
rule.setTags(matchTags1)
request.AddLifecycleRule(rule);
// 分片3天后过期。
rule = new LifecycleRule(ruleId2, matchPrefix2, RuleStatus.Enabled);
LifecycleRule.AbortMultipartUpload abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setExpirationDays(3);
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);
// 指定日期之前的分片过期。
rule = new LifecycleRule(ruleId3, matchPrefix3, RuleStatus.Enabled);
abortMultipartUpload = new LifecycleRule.AbortMultipartUpload();
abortMultipartUpload.setCreatedBeforeDate(DateUtil.parseIso8601Date("2022-10-12T00:00:00.000Z"));
rule.setAbortMultipartUpload(abortMultipartUpload);
request.AddLifecycleRule(rule);
ossClient.setBucketLifecycle(request);
// 关闭OSSClient。
ossClient.shutdown();
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取生命周期规则。
List<LifecycleRule> rules = ossClient.getBucketLifecycle(bucketName);
// 查看生命周期规则。
for (LifecycleRule rule1 : rules) {
  // 查看规则id。
  System.out.println("rule id: " + rule1.getId());
  // 查看规则前缀。
  System.out.println("rule prefix: " + rule1.getPrefix());
  // 查看规则标签。
  if (rule1.hasTags()) {
   System.out.println("rule tagging: "+ rule1.getTags().toString());
  }
  //查看过期天数规则。
  if (rule1.hasExpirationDays()) {
   System.out.println("rule expiration days: " + rule1.getExpirationDays());
  }
  //查看过期日期规则。
  if (rule1.hasCreatedBeforeDate()) {
    System.out.println("rule expiration create before days: " + rule1.getCreatedBeforeDate());
  }
  //查看过期分片规则。
  if(rule1.hasAbortMultipartUpload()) {
    if(rule1.getAbortMultipartUpload().hasExpirationDays()) {
      System.out.println("rule abort uppart days: " + rule1.getAbortMultipartUpload().getExpirationDays());
    }
    if (rule1.getAbortMultipartUpload().hasCreatedBeforeDate()) {
      System.out.println("rule abort uppart create before date: " + rule1.getAbortMultipartUpload().getCreat
edBeforeDate());
    }
  }
}
// 关闭OSSClient。
ossClient.shutdown();
```

2.11. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

说明 有关单链接限速的使用场景及注意事项的更多信息，请参考开发指南的单链接限速文档。

普通上传下载限速

以下代码用于普通上传、下载文件时设置单链接限速：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>"
String objectName = "<yourObjectName>"
String localFileName = "<yourLocalFileName>"
String downLoadFileName = "<yourDownLoadFileName>"
// 限速100KB/s，即819200bit/s。
int limitSpeed = 100 * 1024 * 8;
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 限速上传。
InputStream inputStream = new FileInputStream(localFileName);
PutObjectRequest PutObjectRequest = new PutObjectRequest(bucketName, key, inputStream);
PutObjectRequest.setTrafficLimit(limitSpeed);
ossClient.putObject(PutObjectRequest);
// 限速下载。
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
getObjectRequest.setTrafficLimit(limitSpeed);
File localFile = new File(downLoadFileName);
ossClient.getObject(getObjectRequest, localFile);
 ossClient.shutdown();
```

使用签名URL方式上传下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String localFileName = "<yourLocalFileName>";
String downLoadFileName = "<yourDownLoadFileName>";
// 限速100KB/s，即819200bit/s。
int limitSpeed = 100 * 1024 * 8;
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 创建限速上传的url, 有效期60s。
Date date = new Date();
date.setTime(date.getTime() + 60 * 1000);
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName, Http
Method.PUT);
request.setExpiration(date);
request.setTrafficLimit(limitSpeed);
URL signedUrl = ossClient.generatePresignedUrl(request);
System.out.println("put object url" = signedUrl);
// 限速上传。
InputStream inputStream = new FileInputStream(localFileName);
ossClient.putObject(signedUrl, instream, -1, null, true);
// 创建限速下载的url, 有效期60s。
date = new Date();
date.setTime(date.getTime() + 60 * 1000);
request = new GeneratePresignedUrlRequest(bucketName, key, HttpMethod.GET);
request.setExpiration(date);
request.setTrafficLimit(limitSpeed);
signedUrl = ossClient.generatePresignedUrl(request);
System.out.println("get object url" = signedUrl);
// 限速下载。
GetObjectRequest getObjectRequest =  new GetObjectRequest(signedUrl, null);
ossClient.getObject(getObjectRequest, downLoadFileName);
ossClient.shutdown();
```

2.12. 数据加密

2.12.1. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

> **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

> **注意**
> - 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
> - 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
> - 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 设置Bucket加密。
ServerSideEncryptionByDefault applyServerSideEncryptionByDefault = new ServerSideEncryptionByDefaul
t(SSEAlgorithm.KMS);
applyServerSideEncryptionByDefault.setKMSMasterKeyID("<yourTestKmsId>");
ServerSideEncryptionConfiguration sseConfig = new ServerSideEncryptionConfiguration();
sseConfig.setApplyServerSideEncryptionByDefault(applyServerSideEncryptionByDefault);
SetBucketEncryptionRequest request = new SetBucketEncryptionRequest("<yourBucketName>", sseConfig)
;
ossClient.setBucketEncryption(request);
// 关闭OSSClient。
ossClient.shutdown();
```

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 获取Bucket加密配置。
ServerSideEncryptionConfiguration sseConfig = ossClient.getBucketEncryption("<yourBucketName>");
System.out.println("get Algorithm: " + sseConfig.getApplyServerSideEncryptionByDefault().getSSEAlgorithm());
System.out.println("get kmsid: " + sseConfig.getApplyServerSideEncryptionByDefault().getKMSMasterKeyID());
// 关闭OSSClient。
ossClient.shutdown();
```

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 删除Bucket加密配置。
ossClient.deleteBucketEncryption("<yourBucketName>");
// 关闭OSSClient。
ossClient.shutdown();
```

2.12.2. 客户端加密

客户端加密是指将数据发送到OSS之前在用户本地进行加密。

免责声明

- 使用客户端加密功能时，您需要对主密钥的完整性和正确性负责。因您维护不当导致主密钥用错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。
- 在对加密数据进行复制或者迁移时，您需要对加密元信息的完整性和正确性负责。因您维护不当导致加密元信息出错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。

背景信息

使用客户端加密时，会为每个Object生成一个随机数据加密密钥，用该随机数据加密密钥明文对Object的数据进行对称加密。主密钥用于生成随机的数据加密密钥，加密后的内容会当作Object的meta信息保存在服务端。解密时先用主密钥将加密后的随机密钥解密出来，再用解密出来的随机数据加密密钥明文解密Object的数据。主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。

注意

- 客户端加密支持分片上传超过5 GB的文件。在使用分片方式上传文件时，需要指定上传文件的总大小和分片大小， 除了最后一个分片外，每个分片的大小要一致，且分片大小目前必须是16的整数倍。
- 调用客户端加密上传文件后，加密元数据会被保护，无法通过CopyObject修改Object meta信息。

加密方式

对于主密钥的使用，目前支持如下两种方式：

- 使用KMS托管用户主密钥

  当使用KMS托管用户主密钥用于客户端数据加密时，需要将KMS用户主密钥ID（即CMK ID）传递给SDK。

- 使用用户自主管理的主密钥（RSA）

  主密钥信息由用户提供，需要用户将主密钥的公钥、私钥信息当做参数传递给SDK。

使用以上两种加密方式能够有效地避免数据泄漏，保护客户端数据安全。即使数据泄漏，其他人也无法解密得到原始数据。

加密元信息

参数	描述	是否必须
x-oss-meta-client-side-encryption-key	加密后的密钥。 经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-start	随机产生的用于加密数据的初始值。经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-cek-alg	数据的加密算法。	是
x-oss-meta-client-side-encryption-wrap-alg	数据密钥的加密算法。	是
x-oss-meta-client-side-encryption-matdesc	主密钥的描述信息。JSON格式。 警告 强烈建议为每个主密钥都配置描述信息，并保存好主密钥和描述信息之间的对应关系。否则加密之后不支持更换主密钥进行加密。	否
x-oss-meta-client-side-encryption-unencrypted-content-length	加密前的数据长度。若未指定content-length，则不生成该参数。	否

参数	描述	是否必须
x-oss-meta-client-side-encryption-unencrypted-content-md5	加密前数据的MD5。若未指定MD5，则不生成该参数。	否
x-oss-meta-client-side-encryption-data-size	若加密Multipart文件，则需要在init_multipart时传入整个大文件的总大小。	是（分片上传）
x-oss-meta-client-side-encryption-part-size	若加密Multipart文件，则需要在init_multipart时传入分片大小。 **注意** 目前分片大小必须是16的整数倍。	是（分片上传）

创建加密客户端

说明 使用客户端加密时，需确保您使用了OSS Java SDK 3.9.1及以上版本，同时在工程里添加Bouncy Castle Crypto包，例如在pom.xml中加入如下依赖。

```
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk15on</artifactId>
  <version>1.62</version>
</dependency>
```

如果运行时报 `java.security.InvalidKeyException: Illegal key size or default parameters` 异常，则需要补充Oracle的JCE文件，将其部署在JRE的环境中。

请根据使用的JDK版本分别下载对应的文件，将其解压后保存在*jre/lib/security*目录下。

- JDK6 JCE补充包
- JDK7 JCE补充包
- JDK8 JCE补充包

以下提供了创建RSA、KMS加密客户端的完整示例。

- 创建RSA加密客户端

  创建RSA加密客户端之前，需要创建非对称密钥KeyPair对象。OSS Java SDK提供了从PKCS1编码或PKCS8编码的pem格式私钥字符串到RSAPrivateKey对象的转换，以及从X509编码pem格式公钥字符串到RSAPublicKey对象的转换。

  上述密钥对应的转换方法如下：

  ```
  RSAPrivateKey SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(String privateKeyStr);
  RSAPrivateKey SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS8(String privateKeyStr);
  RSAPublicKey SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(String publicKeyStr);
  ```

  创建RSA加密客户端示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 您可以使用以下命令分别生成私钥与公钥pem文件，然后复制pem文件中的字符串到PRIVATE_PKCS1_PEM，PUBLIC_X509_PEM变量中。
// openssl genrsa -out private_key.pem 2048
// openssl rsa -in private_key.pem -out rsa_public_key.pem -pubout
// 填写您的RSA私钥字符串。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请将其他主密钥及其描述信息添加到加密材料中。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建RSA加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
            build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
```

- 创建KMS加密客户端

  创建KMS加密客户端示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 用户主密钥。
String cmk = "<yourCmkId>";
// cmk所在的region。
String region = "<yourKmsRegion>";
// 创建主密钥KMS的描述信息，创建后不允许修改。主密钥描述信息、region以及用户主密钥（cmk)是一一对应关系。
// 如果所有的object都使用相同的cmk，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置主密钥描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建KMS加密材料。
KmsEncryptionMaterials encryptionMaterials = new KmsEncryptionMaterials(region, cmk, matDesc);
// 如果要下载并解密其他cmk加密的文件，请把cmk的region名称以及描述信息添加到KMS加密材料中。
// encryptionMaterials.addKmsDescMaterial(<otherKmsRegion>, <otherKmsMatDesc>);
// 如果要下载并解密其他cmk加密的文件，并且kms的账号不同于OSS客户端账号，请把Kms的region名称、凭证以及描述信息添加到加密材料中。
// encryptionMaterials.addKmsDescMaterial(<otherKmsRegion>, <otherKmsCredentialsProvider>, <otherKmsMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
            build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
```

以下提供了如何使用用户自主管理的主密钥（RSA）进行普通上传和下载文件、分片上传、范围下载等场景的完整示例。

> **说明** 使用主密钥KMS与使用主密钥RSA的方式，区别仅在于OSSEncryptionClient的创建过程。

普通上传和下载文件

使用主密钥RSA进行普通上传和下载Object示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String content = "Hello OSS!";
// 填写您的RSA私钥字符串。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息。创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请将其他主密钥及其描述信息添加到加密材料中。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
                build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
// 加密上传文件。
ossEncryptionClient.putObject(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
// 下载文件时自动解密。
OSSObject ossObject = ossEncryptionClient.getObject(bucketName, objectName);
BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
StringBuffer buffer = new StringBuffer();
String line;
while ((line = reader.readLine()) != null) {
    buffer.append(line);
}
reader.close();
// 查看解密后的内容是否与上传的明文一致。
System.out.println("Put plain text: " + content);
System.out.println("Get and decrypted text: " + buffer.toString());
```

分片上传

使用主密钥RSA进行分片上传的示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
```

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String localFile = "<yourLocalFilePath>";
// 填写您的RSA私钥字符串。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请将其他主密钥及其描述信息添加到加密材料中。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
                build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
// 创建MultipartUploadCryptoContext对象，指定分片大小与文件大小。分片大小需16字节对齐。
File file = new File(localFile);
long fileLength = file.length();
final long partSize = 100 * 1024L;  // 100K
MultipartUploadCryptoContext context = new MultipartUploadCryptoContext();
context.setPartSize(partSize);
context.setDataSize(fileLength);
// 初始化一个分片上传事件。
InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(bucketName, objectName);
// 传入MultipartUploadCryptoContext对象。
InitiateMultipartUploadResult upresult = ossEncryptionClient.initiateMultipartUpload(initiateMultipartUploadRequest, context);
String uploadId = upresult.getUploadId();
// 创建PartETag的集合。PartETag由分片的ETag和分片号组成。
List<PartETag> partETags =  new ArrayList<PartETag>();
int partCount = (int) (fileLength / partSize);
if (fileLength % partSize != 0) {
    partCount++;
}
// 遍历分片上传。
for (int i = 0; i < partCount; i++) {
    long startPos = i * partSize;
```

```
    long curPartSize = (i + 1 == partCount) ? (fileLength - startPos) : partSize;
    InputStream instream = new FileInputStream(file);
    instream.skip(startPos);
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(objectName);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setInputStream(instream);
    uploadPartRequest.setPartSize(curPartSize);
    uploadPartRequest.setPartNumber( i + 1);
    // 传入MultipartUploadCryptoContext对象。
    UploadPartResult uploadPartResult = ossEncryptionClient.uploadPart(uploadPartRequest, context);
    partETags.add(uploadPartResult.getPartETag());
}
// 完成分片上传。
CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest(bucketName, objectName, uploadId, partETags);
ossEncryptionClient.completeMultipartUpload(completeMultipartUploadRequest);
// 关闭OSSEncryptionClient。
ossEncryptionClient.shutdown();
```

断点续传上传

使用主密钥RSA进行断点续传上传的示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String localFile = "<yourLocalFilePath>";
// 填写您的RSA私钥字符串，可以使用OpenSSL工具生成。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串，可以使用OpenSSL工具生成。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请将RSA密钥加密的文件及其描述信息添加至加密材料中。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
                build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
// 创建UploadFileRequest对象。
UploadFileRequest uploadFileRequest = new UploadFileRequest(bucketName, key);
// 设置要上传的文件的路径。
uploadFileRequest.setUploadFile(localFile));
// 指定上传的分片大小，范围为100 KB~5 GB，默认为文件大小的1/10000。
uploadFileRequest.setPartSize(100 * 1024);
// 开启断点续传，默认关闭。
uploadFileRequest.setEnableCheckpoint(true);
// 设置断点记录文件。如未指定，则默认名称为localfile + ".ucp"，与localfile同目录。
// 上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。
uploadFileRequest.setCheckpointFile("test-upload.ucp");
// 开始断点续传上传。
ossEncryptionClient.uploadFile(uploadFileRequest);
// 关闭OSSEncryptionClient。
ossEncryptionClient.shutdown();
```

断点续传下载

使用主密钥RSA进行断点续传下载的示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 填写您的RSA私钥字符串，可以使用OpenSSL工具生成。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串，可以使用OpenSSL工具生成。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请把它以及其描述信息添加到加密材料里。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
            build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
// 发起下载请求，指定10个任务并发下载，启动断点续传下载。
DownloadFileRequest downloadFileRequest = new DownloadFileRequest(bucketName, objectName);
downloadFileRequest.setDownloadFile("<yourDownloadFile>");
downloadFileRequest.setPartSize(1 * 1024 * 1024);
downloadFileRequest.setTaskNum(10);
downloadFileRequest.setEnableCheckpoint(true);
downloadFileRequest.setCheckpointFile("<yourCheckpointFile>");
// 开始断点续传下载。
DownloadFileResult downloadRes = ossEncryptionClient.downloadFile(downloadFileRequest);
// 关闭OSSEncryptionClient。
ossEncryptionClient.shutdown();
```

范围下载

使用主密钥RSA对范围下载的文件进行解密的示例代码如下：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String content = "test-range-get-content-82042795hlnf12s8yhfs976y2nfoshhnsdfsf235bvsmnhtskbcfd!";
// 填写您的RSA私钥字符串。
final String PRIVATE_PKCS1_PEM = "<yourPrivateKeyWithPKCS1PemString>";
// 填写您的RSA公钥字符串。
final String PUBLIC_X509_PEM = "<yourPublicKeyWithX509PemString>";
// 创建一个RSA密钥对。
RSAPrivateKey privateKey = SimpleRSAEncryptionMaterials.getPrivateKeyFromPemPKCS1(PRIVATE_PKCS1_PEM);
RSAPublicKey publicKey = SimpleRSAEncryptionMaterials.getPublicKeyFromPemX509(PUBLIC_X509_PEM);
KeyPair keyPair = new KeyPair(publicKey, privateKey);
// 创建主密钥RSA的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断文件使用的是哪个主密钥进行加密。
// 强烈建议为每个主密钥都配置描述信息，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
Map<String, String> matDesc = new HashMap<String, String>();
matDesc.put("<yourDescriptionKey>", "<yourDescriptionValue>");
// 创建RSA加密材料。
SimpleRSAEncryptionMaterials encryptionMaterials = new SimpleRSAEncryptionMaterials(keyPair, matDesc);
// 如果要下载并解密其他RSA密钥加密的文件，请将其他主密钥及其描述信息添加到加密材料中。
// encryptionMaterials.addKeyPairDescMaterial(<otherKeyPair>, <otherKeyPairMatDesc>);
// 创建加密客户端。
OSSEncryptionClient ossEncryptionClient = new OSSEncryptionClientBuilder().
                build(endpoint, accessKeyId, accessKeySecret, encryptionMaterials);
// 加密上传文件。
ossEncryptionClient.putObject(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
// 范围下载。
int start = 17;
int end = 35;
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
getObjectRequest.setRange(start, end);
ossObject = ossEncryptionClient.getObject(getObjectRequest);
reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
buffer = new StringBuffer();
while ((line = reader.readLine()) != null) {
    buffer.append(line);
}
reader.close();
// 查看范围下载结果。
System.out.println("Range-Get plain text:" + content.substring(start, end + 1));
System.out.println("Range-Get decrypted text: " + buffer.toString());
// 关闭OSSEncryptionClient。
ossEncryptionClient.shutdown();
```

2.13. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

> **说明** 如果需要生成HTTPS协议的签名URL，请将Endpoint中的通信协议设置为HTTPS。

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

> **说明** 关于搭建STS服务的具体操作，请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。临时访问凭证有效时间单位为秒，最小值为900，最大值以当前角色设定的最大会话时间为准。更多信息，请参见设置角色最大会话时间。

以下代码用于使用STS凭证构造签名请求。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
//String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
//String objectName = "exampleobject.txt";
// 从STS服务获取临时访问凭证后，您可以通过临时访问密钥和安全令牌生成OSSClient。
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 执行OSS相关操作，例如上传、下载文件等。
// 上传文件，此处以上传本地文件为例介绍。
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
//PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objectName, new File("D:\\loc
alpath\\examplefile.txt"));
//ossClient.putObject(putObjectRequest);
// 下载OSS文件到本地文件。如果指定的本地文件存在则覆盖，不存在则新建。
// 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
//ossClient.getObject(new GetObjectRequest(bucketName, objectName), new File("D:\\localpath\\examplefil
e.txt"));
// 关闭OSSClient。
ossClient.shutdown();
```

使用签名URL进行临时授权

以下介绍使用签名URL临时授权的常见示例。

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

- 生成签名URL

  您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

  在URL中加入签名信息，以便将该URL转给第三方实现授权访问。具体操作，请参见在URL中包含签名。

- 生成以GET方法访问的签名URL

  您可以根据需要一次生成单个或者多个以GET方法访问的签名URL。

- 生成单个以GET方法访问的签名URL

  以下代码用于一次生成单个以GET方法访问的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
// 生成以GET方法访问的签名URL，访客可以直接通过浏览器访问相关内容。
URL url = ossClient.generatePresignedUrl(bucketName, objectName, expiration);
System.out.println(url);
// 关闭OSSClient。
ossClient.shutdown();
```

- 生成多个以GET方法访问的签名URL

  以下代码用于一次生成多个以GET方法访问的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
// 此处请填写多个Object完整路径，用于一次性获取多个Object的签名URL。
String objectNameList [] = {"exampleobject.txt","exampleimage.jpg"};
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
List<URL> urlList = new ArrayList<URL>();
for(int i=0; i<objectNameList.length; i++){
  URL url = ossClient.generatePresignedUrl(bucketName, objectNameList[i], expiration);
  urlList.add(url);
}
// 打印签名URL。
for(URL url:urlList){
  System.out.println(url);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 生成以其他HTTP方法访问的签名URL

  如果您要授权其他用户临时执行其他操作（例如上传、删除文件等），需要生成对应的签名URL，例如生成以PUT方法访问的签名URL来上传文件。您可以根据需要一次生成单个或者多个以其他HTTP方法访问的签名URL。

- 生成单个以其他HTTP方法访问的签名URL

  以下代码用于一次生成单个以其他HTTP方法访问的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 从STS服务获取临时访问凭证后，您可以通过临时访问密钥和安全令牌生成OSSClient。
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.PUT);
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
request.setExpiration(expiration);
// 设置ContentType。
request.setContentType("text/plain");
// 设置自定义元信息。
request.addUserMetadata("author", "aliy");
// 生成签名URL。
URL signedUrl = ossClient.generatePresignedUrl(request);
System.out.println(signedUrl);
Map<String, String> requestHeaders = new HashMap<String, String>();
// 设置ContentType，必须和生成签名URL时设置的ContentType一致。
requestHeaders.put(HttpHeaders.CONTENT_TYPE, "text/plain");
// 设置自定义元信息。
requestHeaders.put(OSS_USER_METADATA_PREFIX + "author", "aliy");
// 使用签名URL上传文件。
ossClient.putObject(signedUrl, new ByteArrayInputStream("Hello OSS".getBytes()), -1, requestHeaders
, true);
// 关闭OSSClient。
ossClient.shutdown();
```

- 生成多个以其他HTTP方法访问的签名URL

  以下代码用于一次生成多个以其他HTTP方法访问的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
// 此处请填写多个Object完整路径，用于一次性获取多个Object的签名URL。
String objectNameList [] = {"exampleobject.txt","exampleimage.jpg"};
String upLoadNameArray [] = {"D:\\localpath\\examplefile1.txt","D:\\localpath\\examplefile2.jpg"};
// 从STS服务获取临时访问凭证后，您可以通过临时访问密钥和安全令牌生成OSSClient。
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
for(int i=0; i<objectNameList.length; i++){
  GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectNam
eList[i], HttpMethod.PUT);
  request.setExpiration(expiration);
  // 设置ContentType。
  request.setContentType(DEFAULT_OBJECT_CONTENT_TYPE);
  // 设置自定义元信息。
  request.addUserMetadata("author", "aliy");
  // 生成签名URL。
  URL signedUrl = ossClient.generatePresignedUrl(request);
  // 打印签名URL。
  System.out.println(signedUrl);
  Map<String, String> requestHeaders = new HashMap<String, String>();
  requestHeaders.put(HttpHeaders.CONTENT_TYPE, DEFAULT_OBJECT_CONTENT_TYPE);
  requestHeaders.put(OSS_USER_METADATA_PREFIX + "author", "aliy");
  // 如果要上传字符串，请使用如下方式。
  //ossClient.putObject(signedUrl, new ByteArrayInputStream("Hello OSS".getBytes()), -1, requestHea
ders, true);
  // 使用签名URL上传文件。
  try {
   ossClient.putObject(signedUrl, new FileInputStream(new File(upLoadNameArray[i])), -1, requestHea
ders, true);
  } catch (FileNotFoundException e) {
   e.printStackTrace();
  }
}
// 关闭OSSClient。
ossClient.shutdown();
```

通过传入HttpMethod.PUT参数，访客可以使用生成的签名URL上传文件。

- 生成带有指定参数的签名URL

  您可以根据需要一次生成单个或者多个带有指定参数的签名URL。

- 生成单个带有指定参数的签名URL

  以下代码用于一次生成单个带有指定参数的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 创建请求。
GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresignedUrlRequest(bucketName, objectName);
// 设置HttpMethod为PUT。
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
// 添加用户自定义元信息。
generatePresignedUrlRequest.addUserMetadata("author", "baymax");
// 设置ContentType。
generatePresignedUrlRequest.setContentType("application/txt");
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
generatePresignedUrlRequest.setExpiration(expiration);
// 生成签名URL。
URL url = ossClient.generatePresignedUrl(generatePresignedUrlRequest);
System.out.println(url);
// 关闭OSSClient。
ossClient.shutdown();
```

- 生成多个带有指定参数的签名URL

  以下代码用于一次生成多个带有指定参数的签名URL。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
// 此处请填写多个Object完整路径，用于一次性获取多个Object的签名URL。
String objectNameList [] = {"exampleobject.txt","exampleimage.jpg"};
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 设置签名URL过期时间为3600秒（1小时）。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
for(int i=0; i<objectNameList.length; i++){
  // 创建请求。
  GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresignedUrlRequest(b
ucketName, objectNameList[i]);
  // 设置HttpMethod为PUT。
  generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
  // 添加用户自定义元信息。
  generatePresignedUrlRequest.addUserMetadata("author", "baymax");
  // 设置ContentType。
  generatePresignedUrlRequest.setContentType("application/txt");
  generatePresignedUrlRequest.setExpiration(expiration);
  // 生成签名URL。
  URL url = ossClient.generatePresignedUrl(generatePresignedUrlRequest);
  // 打印签名URL。
  System.out.println(url);
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 使用签名URL上传或获取文件
  - 使用签名URL上传文件

    以下代码用于使用签名URL上传文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 填写签名URL的过期时间。
Date expiration = null;
try {
   expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2022 14:20:00 GMT");
} catch (ParseException e) {
   e.printStackTrace();
}
// 生成签名URL。
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.PUT);
// 设置过期时间。
request.setExpiration(expiration);
// 设置ContentType。
request.setContentType("application/txt");
// 添加用户自定义元信息。
request.addUserMetadata("author", "aliy");
// 通过HTTP PUT请求生成签名URL。
URL signedUrl = ossClient.generatePresignedUrl(request);
System.out.println("signed url for putObject: " + signedUrl);
// 使用签名URL发送请求。
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
File f = new File("D:\\localpath\\examplefile.txt");
FileInputStream fin = null;
try {
   fin = new FileInputStream(f);
} catch (FileNotFoundException e) {
   e.printStackTrace();
}
// 添加PutObject请求头。
Map<String, String> customHeaders = new HashMap<String, String>();
customHeaders.put("Content-Type", "application/txt");
customHeaders.put("x-oss-meta-author", "aliy");
PutObjectResult result = ossClient.putObject(signedUrl, fin, f.length(), customHeaders);
// 关闭OSSClient。
ossClient.shutdown();
```

- 使用签名URL获取文件

  以下代码用于使用签名URL获取指定文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-c
n-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
String securityToken = "yourSecurityToken";
// 填写Bucket名称，例如examplebucket。
String bucketName = "examplebucket";
// 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.txt";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret, securityToken);
// 填写签名URL的过期时间。
Date expiration = null;
try {
   expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2022 14:20:00 GMT");
} catch (ParseException e) {
   e.printStackTrace();
}
// 生成签名URL。
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, objectName,
HttpMethod.GET);
// 设置过期时间。
request.setExpiration(expiration);
// 通过HTTP GET请求生成签名URL。
URL signedUrl = ossClient .generatePresignedUrl(request);
System.out.println("signed url for getObject: " + signedUrl);
// 使用签名URL发送请求。
Map<String, String> customHeaders = new HashMap<String, String>();
// 添加GetObject请求头。
customHeaders.put("Range", "bytes=100-1000");
OSSObject object = ossClient.getObject(signedUrl,customHeaders);
// 关闭OSSClient。
ossClient.shutdown();
```

2.14. 数据安全性

OSS提供基于MD5和CRC64的数据校验，确保上传、下载和拷贝文件（Object）过程中的数据完整性。

MD5校验

如果上传文件时设置了Content-MD5，OSS会根据接收的内容计算MD5。OSS计算的MD5值和上传提供的MD5值不一致时，则返回InvalidDigest异常，从而保证数据的完整性。返回InvalidDigest异常后，您需要重新上传文件。

以下代码用于上传文件时进行MD5校验：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 上传字符串。
String content = "Hello OSS";
ObjectMetadata meta = new ObjectMetadata();
// 设置MD5校验。
String md5 = BinaryUtil.toBase64String(BinaryUtil.calculateMd5(content.getBytes()));
meta.setContentMD5(md5);
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(content.getBytes()), meta);
// 关闭OSSClient。
ossClient.shutdown();
```

说明 putObject、getObject、appendObject、postObject、uploadPart支持MD5校验。

CRC64校验

上传、下载和拷贝文件时默认开启CRC数据校验，确保数据的完整性。

说明

- putObject、getObject、appendObject、uploadPart支持CRC64校验。上传时默认开启CRC校验，如果客户端计算的CRC值与服务端返回的CRC值不一致， 则会抛出InconsistentException异常。
- 范围下载不支持CRC64校验。
- CRC64校验会占用一定的CPU，对上传、下载速度均会有影响。

- 下载文件时CRC64校验

  以下代码用于下载文件时进行CRC64数据完整性校验：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 流式下载。
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectName);
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// 读取文件内容，只有读取文件内容之后才能获取clientCrc。
System.out.println("Object content:");
BufferedReader reader = new BufferedReader(new InputStreamReader(ossObject.getObjectContent()));
while (true) {
  String line = reader.readLine();
  if (line == null) break;
  System.out.println("\n" + line);
}
// 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
reader.close();
// 查看客户端是否开启了crc校验，默认是开启状态。
Boolean isCrcCheckEnabled = ((OSSClient)ossClient).getClientConfiguration().isCrcCheckEnabled();
// 查看是否是范围下载请求。范围下载方式不支持校验crc。
Boolean isRangGetRequest = getObjectRequest.getHeaders().get(OSSHeaders.RANGE) != null;
// 校验crc，只有读取文件内容之后才能获取clientCrc。
if (isCrcCheckEnabled && !isRangGetRequest) {
  Long clientCRC = IOUtils.getCRCValue(ossObject.getObjectContent());
  OSSUtils.checkChecksum(clientCRC, ossObject.getServerCRC(), ossObject.getRequestId());
}
// 关闭OSSClient。
ossClient.shutdown();
```

- 追加上传时CRC64校验

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";
String firstAppendContent = "<yourFirstAppendContent>";
String secondAppendContent = "<yourSecondAppendContent>";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 第一次追加。
AppendObjectRequest appendObjectRequest = new AppendObjectRequest(bucketName, objectName, new ByteArrayInputStream(firstAppendContent.getBytes()));
appendObjectRequest.setPosition(0L);
// 初始化crc。初始化crc之后，SDK内部默认会对上传结果进行crc校验。
appendObjectRequest.setInitCRC(0L);
AppendObjectResult appendObjectResult = ossClient.appendObject(appendObjectRequest);
// 第二次追加。
appendObjectRequest = new AppendObjectRequest(bucketName, objectName, new ByteArrayInputStream(secondAppendContent.getBytes()));
appendObjectRequest.setPosition(appendObjectResult.getNextPosition());
// 初始化crc设置为已上传数据的crc。初始化crc之后，SDK内部默认会对上传结果进行crc校验。
appendObjectRequest.setInitCRC(appendObjectResult.getClientCRC());
ossClient.appendObject(appendObjectRequest);
// 关闭OSSClient。
ossClient.shutdown();
```

2.15. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

> **说明** 图片处理支持的参数请参见处理参数。图片处理的完整代码请参见GitHub。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.jpg";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 将图片缩放为固定宽高100 px。
String style = "image/resize,m_fixed,w_100,h_100";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-resize.jpg并保存到本地。
// 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
// 如果未指定本地路径，则文件默认保存到示例程序所属项目对应本地路径中。
ossClient.getObject(request, new File("D:\\localpath\\example-resize.jpg"));
// 关闭OSSClient。
ossClient.shutdown();
```

- 使用不同的图片处理参数处理图片并分别保存为本地图片

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.jpg";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 将图片缩放为固定宽高100 px。
String style = "image/resize,m_fixed,w_100,h_100";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-resize.jpg并保存到本地。
// 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
// 如果未指定本地路径，则文件默认保存到示例程序所属项目对应本地路径中。
ossClient.getObject(request, new File("D:\\localpath\\example-resize.jpg"));
// 从坐标（100,100）开始，将图片裁剪为宽高100 px。
style = "image/crop,w_100,h_100,x_100,y_100";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-crop.jpg并保存到本地。
ossClient.getObject(request, new File("D:\\localpath\\example-crop.jpg"));
// 将图片旋转90°。
style = "image/rotate,90";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-rotate.jpg并保存到本地。
ossClient.getObject(request, new File("D:\\localpath\\example-rotate.jpg"));
// 在图片中添加文字水印。
// 文字水印的文字内容经过Base64编码后，再将编码结果中的加号（+）替换成短划线（-），正斜线（/）替换成下划线（_）并去掉尾部的等号（=），从而得到水印字符串。
// 指定文字水印的文字内容为Hello World，文字内容进行编码处理后得到的水印字符串为SGVsbG8gV29ybGQ。
String style = "image/watermark,text_SGVsbG8gV29ybGQ";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-watermarktext.jpg并保存到本地。
ossClient.getObject(request, new File("D:\\localpath\\example-watermarktext.jpg"));
// 在图片中添加图片水印。请确保水印图片已保存在图片所在Bucket中。
// 水印图片的完整路径经过Base64编码后，再将编码结果中的加号（+）替换成短划线（-），正斜线（/）替换成下划线（_）并去掉尾部的等号（=），从而得到水印字符串。
// 指定水印图片的完整路径为panda.jpg，完整路径进行编码处理后得到的水印字符串为cGFuZGEuanBn。
String style = "image/watermark,image_cGFuZGEuanBn";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-watermarkimage.jpg并保存到本地。
ossClient.getObject(request, new File("D:\\localpath\\example-watermarkimage.jpg"));
// 关闭OSSClient。
ossClient.shutdown();
```

- 同时使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.jpg";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 将图片缩放为固定宽高100 px后，再旋转90°。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-new.jpg并保存到本地。
// 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
// 如果未指定本地路径，则文件默认保存到示例程序所属项目对应本地路径中。
ossClient.getObject(request, new File("D:\\localpath\\example-new.jpg"));
// 关闭OSSClient。
ossClient.shutdown();
```

使用图片样式处理图片

您可以通过OSS管理控制台创建图片样式将多个图片处理参数封装在一个样式中，然后使用样式批量处理图片。具体操作，请参见图片样式。

以下代码展示了使用图片样式处理图片。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.jpg";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 使用自定义样式处理图片。
// yourCustomStyleName填写通过OSS管理控制台创建的图片样式名称。
String style = "style/yourCustomStyleName";
GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
// 将处理后的图片命名为example-new.jpg并保存到本地。
// 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
// 如果未指定本地路径，则文件默认保存到示例程序所属项目对应本地路径中。
ossClient.getObject(request, new File("D:\\localpath\\example-new.jpg"));
// 关闭OSSClient。
ossClient.shutdown();
```

图片处理持久化

图片处理服务默认不保存处理后的图片，您可以通过ImgSaveAs接口将图片保存到原图片所在存储空间。

以下代码展示了图片处理持久化操作。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String sourceImage = "exampleimage.png";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
try {
  // 将图片缩放为固定宽高100 px。
  StringBuilder sbStyle = new StringBuilder();
  Formatter styleFormatter = new Formatter(sbStyle);
  String styleType = "image/resize,m_fixed,w_100,h_100";
  // 将处理后的图片命名为example-resize.png并保存到当前Bucket。
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  String targetImage = "example-resize.png";
  styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
      BinaryUtil.toBase64String(targetImage.getBytes()),
      BinaryUtil.toBase64String(bucketName.getBytes()));
  System.out.println(sbStyle.toString());
  ProcessObjectRequest request = new ProcessObjectRequest(bucketName, sourceImage, sbStyle.toString(
));
  GenericResult processResult = ossClient.processObject(request);
  String json = IOUtils.readStreamAsString(processResult.getResponse().getContent(), "UTF-8");
  processResult.getResponse().getContent().close();
  System.out.println(json);
} catch (Exception e) {
  e.printStackTrace();
}
// 关闭OSSClient。
ossClient.shutdown();
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
String endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
String bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
String objectName = "exampleobject.jpg";
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
// 将图片缩放为固定宽高100 px后，再旋转90°。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
// 指定签名URL过期时间为10分钟。
Date expiration = new Date(new Date().getTime() + 1000 * 60 * 10 );
GeneratePresignedUrlRequest req = new GeneratePresignedUrlRequest(bucketName, objectName, HttpMet
hod.GET);
req.setExpiration(expiration);
req.setProcess(style);
URL signedUrl = ossClient.generatePresignedUrl(req);
System.out.println(signedUrl);
// 关闭OSSClient。
ossClient.shutdown();
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

2.16. 异常处理

OSS Java SDK包含两类异常，一类是客户端异常ClientException，另一类是服务器端异常OSSException，它们均继承自RuntimeException。

异常处理示例

以下代码用于展示异常处理：

```
try {
  // OSS操作，例如上传文件
  ossClient.putObject(...);
} catch (OSSException oe) {
  System.out.println("Caught an OSSException, which means your request made it to OSS, "
      + "but was rejected with an error response for some reason.");
  System.out.println("Error Message: " + oe.getErrorMessage());
  System.out.println("Error Code:     " + oe.getErrorCode());
  System.out.println("Request ID:     " + oe.getRequestId());
  System.out.println("Host ID:        " + oe.getHostId());
} catch (ClientException ce) {
  System.out.println("Caught an ClientException, which means the client encountered "
      + "a serious internal problem while trying to communicate with OSS, "
      + "such as not being able to access the network.");
  System.out.println("Error Message: " + ce.getMessage());
} finally {
  if (ossClient != null) {
    ossClient.shutdown();
  }
}
```

ClientException

ClientException指客户端尝试向OSS发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用，则会抛出ClientException。当上传文件时发生IO异常，也会抛出ClientException。

OSSException

OSSException指服务器端异常，它来自于对服务器错误信息的解析。OSSException包含OSS返回的错误码和错误信息，便于定位问题，并做出适当的处理。

OSSException通常包含以下错误信息：

参数	描述
Code	OSS返回的错误码。
Message	OSS返回的详细错误信息。
RequestId	用于唯一标识该请求的UUID。当您无法解决问题时，可以提供RequestId来请求OSS开发工程师的帮助。
HostId	用于标识访问的OSS集群，与请求时使用的Host一致。

OSS常见错误码

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	存储空间已经存在	409

错误码	描述	HTTP状态码
BucketNotEmpty	存储空间非空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
FilePartNotExist	文件分片不存在	400
FilePartStale	文件分片过时	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的存储空间名称	400
InvalidDigest	无效的摘要	400
InvalidObjectName	无效的文件名称	400
InvalidPart	无效的分片	400
InvalidPartOrder	无效的分片顺序	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标存储空间	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	存储空间不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	分片上传ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403

错误码	描述	HTTP状态码
RequestTimeout	请求超时	400
SignatureDoesNotMatch	签名错误	403
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

2.17. 常见问题

本文介绍使用OSS Java SDK的常见问题及解决方法。

包冲突

- 错误原因

  使用OSS Java SDK时，报类似如下错误，说明工程中可能有包冲突。

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/ssl/TrustStrategy
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
    at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:77)
Caused by: java.lang.ClassNotFoundException: org.apache.http.ssl.TrustStrategy
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
    ... 3 more
```

  或

```
Exception in thread "main" java.lang.NoSuchFieldError: INSTANCE
 at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<init>(DefaultHttpRequestWriterFactory.ja
va:52)
 at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<init>(DefaultHttpRequestWriterFactory.ja
va:56)
 at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<clinit>(DefaultHttpRequestWriterFactory.j
ava:46)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnecti
onFactory.java:82)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnecti
onFactory.java:95)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<init>(ManagedHttpClientConnecti
onFactory.java:104)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<clinit>(ManagedHttpClientConnec
tionFactory.java:62)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager$InternalConnectionFactory.<init>(P
oolingHttpClientConnectionManager.java:572)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnection
Manager.java:174)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnection
Manager.java:158)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnection
Manager.java:149)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnection
Manager.java:125)
 at com.aliyun.oss.common.comm.DefaultServiceClient.createHttpClientConnectionManager(DefaultServ
iceClient.java:237)
 at com.aliyun.oss.common.comm.DefaultServiceClient.<init>(DefaultServiceClient.java:78)
 at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
 at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
 at OSSManagerImpl.upload(OSSManagerImpl.java:42)
 at OSSManagerImpl.main(OSSManagerImpl.java:63)
```

错误原因是OSS Java SDK使用了Apache httpclient 4.4.1，而您的工程使用了与Apache httpclient 4.4.1冲突的Apache httpclient或commons-httpclient jar包。要查看工程使用的jar包及版本，请在您的工程目录下执行 `mvn dependency:tree` 。如下图所示，您的工程里使用了Apache httpclient 4.3：

```
[INFO] --- maven-dependency-plugin:2.2:tree (default-cli) @ maven-demo ---
[INFO] com.aliyun.oss:maven-demo:jar:0.1.1-SNAPSHOT
[INFO] +- junit:junit:jar:4.10:test
[INFO] |  \- org.hamcrest:hamcrest-core:jar:1.1:test
[INFO] +- org.apache.httpcomponents:httpclient:jar:4.3:compile
[INFO] |  +- org.apache.httpcomponents:httpcore:jar:4.3:compile
[INFO] |  +- commons-logging:commons-logging:jar:1.1.3:compile
[INFO] |  \- commons-codec:commons-codec:jar:1.6:compile
[INFO] \- com.aliyun.oss:aliyun-sdk-oss:jar:2.2.1:compile
[INFO]    +- org.jdom:jdom:jar:1.1:compile
[INFO]    \- net.sf.json-lib:json-lib:jar:jdk15:2.4:compile
[INFO]       +- commons-beanutils:commons-beanutils:jar:1.8.0:compile
[INFO]       +- commons-collections:commons-collections:jar:3.2.1:compile
[INFO]       +- commons-lang:commons-lang:jar:2.5:compile
[INFO]       \- net.sf.ezmorph:ezmorph:jar:1.0.6:compile
```

- 解决方法

包冲突有以下两种解决方法：

- 使用统一版本。如果您的工程使用与Apache httpclient 4.4.1冲突的版本，请您使用4.4.1版本，并在pom.xml删除其它版本的Apache httpclient依赖。如果您的工程使用了commons-httpclient，也可能存在冲突，请删除commons-httpclient。
- 解除依赖冲突。如果您的工程依赖多个第三方包，而第三方包又依赖不同版本的Apache httpclient，您的工程里会有依赖冲突，请使用exclusion解除。更多信息，请参见Maven Guides。

OSS Java SDK依赖以下版本的包，冲突解决办法与httpclient类似。

```
[INFO] com.aliyun.oss:maven-demo:jar:0.1.1-SNAPSHOT
[INFO] +- junit:junit:jar:4.10:test
[INFO] |  \- org.hamcrest:hamcrest-core:jar:1.1:test
[INFO] \- com.aliyun.oss:aliyun-sdk-oss:jar:2.2.1:compile
[INFO]    +- org.apache.httpcomponents:httpclient:jar:4.4.1:compile
[INFO]    |  +- org.apache.httpcomponents:httpcore:jar:4.4.1:compile
[INFO]    |  +- commons-logging:commons-logging:jar:1.2:compile
[INFO]    |  \- commons-codec:commons-codec:jar:1.9:compile
[INFO]    +- org.jdom:jdom:jar:1.1:compile
[INFO]    \- net.sf.json-lib:json-lib:jar:jdk15:2.4:compile
[INFO]       +- commons-beanutils:commons-beanutils:jar:1.8.0:compile
[INFO]       +- commons-collections:commons-collections:jar:3.2.1:compile
[INFO]       +- commons-lang:commons-lang:jar:2.5:compile
[INFO]       \- net.sf.ezmorph:ezmorph:jar:1.0.6:compile
```

缺少包

- 错误原因

使用OSS Java SDK时，报类似如下错误，说明您的工程中可能缺少编译或运行OSS Java SDK所必需的包。

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/auth/Credentials
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
    at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:76)
Caused by: java.lang.ClassNotFoundException: org.apache.http.auth.Credentials
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
    ... 3 more
```

或

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/http/protocol/HttpContext
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:268)
    at com.aliyun.oss.OSSClient.<init>(OSSClient.java:193)
    at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:76)
Caused by: java.lang.ClassNotFoundException: org.apache.http.protocol.HttpContext
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
    ... 3 more
```

或

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/jdom/input/SAXBuilder
    at com.aliyun.oss.internal.ResponseParsers.getXmlRootElement(ResponseParsers.java:645)
    at … …
    at com.aliyun.oss.OSSClient.doesBucketExist(OSSClient.java:471)
    at com.aliyun.oss.OSSClient.doesBucketExist(OSSClient.java:465)
    at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:82)
Caused by: java.lang.ClassNotFoundException: org.jdom.input.SAXBuilder
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
    ... 11 more
```

OSS Java SDK依赖下列包：

- aliyun-sdk-oss-2.2.1.jar
- hamcrest-core-1.1.jar
- jdom-1.1.jar
- commons-codec-1.9.jar
- httpclient-4.4.1.jar
- commons-logging-1.2.jar
- httpcore-4.4.1.jar
- log4j-1.2.15.jar

其中log4j-1.2.15.jar是可选的，需要日志功能的时候加入该包，其它包都是必需的。

- 解决方法

  在您的工程中加入OSS Java SDK依赖的包。加入方法如下：

  - 如果您的工程在Eclipse中，请参见Java SDK使用手册中的安装方式二。
  - 如果您的工程在Ant中，请把OSS Java SDK依赖的包放入工程的lib目录中。

- 如果您直接使用.javac或.java文件，请使用 `-classpath` 或 `-cp` 命令指定OSS Java SDK依赖的包路径，或把OSS Java SDK依赖的包放入classpath路经下。

连接超时

- 错误原因

  运行OSS Java SDK程序时报如下类似错误，可能原因是Endpoint错误或者网络不通。

```
com.aliyun.oss.ClientException: SocketException
  at com.aliyun.oss.common.utils.ExceptionFactory.createNetworkException(ExceptionFactory.java:71)
  at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:116
)
  at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:121)
  at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:67)
  at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:92)
  at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:140)
  at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:111)
  at com.aliyun.oss.internal.OSSBucketOperation.getBucketInfo(OSSBucketOperation.java:1152)
  at com.aliyun.oss.OSSClient.getBucketInfo(OSSClient.java:1220)
  at com.aliyun.oss.OSSClient.getBucketInfo(OSSClient.java:1214)
  at com.aliyun.oss.demo.HelloOSS.main(HelloOSS.java:94)
Caused by: org.apache.http.conn.HttpHostConnectException: Connect to oss-test.oss-cn-hangzhou-inter
nal.aliyuncs.com:80 [oss-test.oss-cn-hangzhou-internal.aliyuncs.com/10.84.135.99] failed: Connection tim
ed out: connect
  at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnec
tionOperator.java:151)
  at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.connect(PoolingHttpClientConnec
tionManager.java:353)
  at org.apache.http.impl.execchain.MainClientExec.establishRoute(MainClientExec.java:380)
  at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:236)
  at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
  at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
  at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
  at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
  at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:113
)
  ... 9 more
```

- 解决方法

  您可以使用ossutil工具快速定位错误原因并解决问题。

报错SignatureDoesNotMatch

- 错误原因1

  AccessKey ID和AccssKey Secret不一致。

  有关获取AccessKey ID和AccssKey Secret的操作步骤，请参见创建AccessKey。

- 错误原因2

  签名URL使用不正确。错误示例如下：

```
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucketName, object);
request.setExpiration( new Date(new Date().getTime() + 3600 * 1000));
request.addUserMetadata("author");
URL url = ossClient.generatePresignedUrl(request);
Map<String, String> header = new HashMap<String, String>();
header.put("author");
ossClient.putObject(url, new ByteArrayInputStream("Hello OSS".getBytes()), -1, header);
```

未指定Method参数时，默认使用GET方法。以上为PutObject请求，应指定Method参数并设置为PUT方法。

通过PutObject发送请求时，请求Header中自定义的元数据必须以 `x-oss-meta-` 为前缀。以上示例中自定义元数据应改为 `x-oss-meta-author` 。

解决方法：

指定Method，并修改Header：

```
request.addUserMetadata("author");
request.setMethod(HttpMethod.PUT);
URL url = ossClient.generatePresignedUrl(request);
Map<String, String> header = new HashMap<String, String>();
header.put("x-oss-meta-" + "author");
ossClient.putObject(url, new ByteArrayInputStream("Hello OSS".getBytes()), -1, header);
```

- 错误原因3
  - 使用了低于3.7.0版本的OSS SDK，项目中引入了4.5.9及以上版本的httpclient。
  - 上传的文件名中包含 `+` 字符，而4.5.9版本的httpclient不会对 `+` 进行URLEncode编码，从而造成客户端与服务端计算的签名不一致而报错。

```
 320 POST /a HTTP/1.1  (application/json)
 107 HTTP/1.1 200   (application/json)
 453 PUT /%E4%B8%AD%E6%96%87%2B%E6%B5%8B%E8%AF%95 HTTP/1.1
 375 HTTP/1.1 200 OK
 364 POST /p HTTP/1.1  (application/json)
1108 HTTP/1.1 200   (application/json)
1017 POST /a HTTP/1.1  (application/json)
 107 HTTP/1.1 200   (application/json)
 450 PUT /%E4%B8%AD%E6%96%87+%E6%B5%8B%E8%AF%95 HTTP/1.1
1183 HTTP/1.1 403 Forbidden
 325 POST /a HTTP/1.1  (application/json)
 107 HTTP/1.1 200   (application/json)
```

解决方法：

  - OSS SDK建议升级为3.11.1及以上版本， 以兼容4.5.9版本的httpclient。
  - 移除多余的httpclient依赖。引入OSS SDK时会自动引入httpclient依赖，如果是第三方库另外引入了httpclient，请参见包冲突解决方案。

- 错误原因4

HttpClient 4.5.10版本不支持Header中包含ISO/9959-1标准以外的字符，但在项目中引入了4.5.10以上的 httpclient，并在请求Header中包含了ISO/9959-1标准以外的字符，例如 `x-oss-meta-` 开头的自定义元数据中传入了中文字符。

```
String content = "123";
ObjectMetadata metadata = new ObjectMetadata();
metadata.setHeader("x-oss-meta-filename", "测试中文header");
PutObjectRequest request = new PutObjectRequest(bucketName, objectName, new ByteArrayInputStream(content.getBytes()));
request.setMetadata(metadata);
ossClient.putObject(request);
```

解决方法：

- 参见包冲突解决方案，移除冲突的httpclient版本。
- 在请求Header中传入符合ISO/9959-1标准的字符。

报异常“Failed to parse the response result”

```
com.aliyun.oss.OSSException: Failed to parse the response result.
ErrorCode]: InvalidResponse
RequestId]: null
HostId]: null
        at com.aliyun.oss.common.utils.ExceptionFactory.createOSSException(ExceptionFactory.java:109)
        at com.aliyun.oss.common.utils.ExceptionFactory.createInvalidResponseException(ExceptionFactory.java:91)
        at com.aliyun.oss.common.utils.ExceptionFactory.createInvalidResponseException(ExceptionFactory.java:81)
        at com.aliyun.oss.internal.OSSErrorResponseHandler.handle(OSSErrorResponseHandler.java:71)
        at com.aliyun.oss.common.comm.ServiceClient.handleResponse(ServiceClient.java:248)
        at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:130)
        at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
        at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
        at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:149)
        at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
        at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:273)
        at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:301)
        at com.aliyun.oss.OSSClient.getObject(OSSClient.java:545)
```

- 错误原因

  客户端某些特殊的软件拦截了HTTP请求，或者公网路由劫持了HTTP请求。

  在Java 11上使用OSS Java SDK，且未在pom.xml文件中添加JAXB相关依赖。

- 解决方法

  切换为HTTPS请求。

  添加JAXB相关依赖。操作步骤，请参见安装SDK。

org.apache.http.NoHttpResponseException: The target server failed to respond

- 错误原因

  运行OSS Java SDK程序时，报类似如下错误：

```
com.aliyun.oss.ClientException: Unknown
        at com.aliyun.oss.common.utils.ExceptionFactory.createNetworkException(ExceptionFactory.java:68) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:115) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:121) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:67) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:92) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:140) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:111) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.internal.OSSMultipartOperation.initiateMultipartUpload(OSSMultipartOperation.java:206) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.aliyun.oss.OSSClient.initiateMultipartUpload(OSSClient.java:765) ~[aliyun-sdk-oss-2.1.0.jar:na]
        at com.taobao.agoo.dump.client.OssTools.multipartUpload(OssTools.java:79) ~[agoo-dump-client-2.0.0-SNAPSHOT.jar:na]
        at com.taobao.agoo.dump.biz.manager.TaskExcutorManager$UploadTask.run(TaskExcutorManager.java:114) ~[agoo-dump-biz-2.0.0-SNAPSHOT.jar:na]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471) [na:1.7.0_51]
        at java.util.concurrent.FutureTask.run(FutureTask.java:262) [na:1.7.0_51]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145) [na:1.7.0_51]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) [na:1.7.0_51]
        at java.lang.Thread.run(Thread.java:744) [na:1.7.0_51]
Caused by: org.apache.http.NoHttpResponseException: The target server failed to respond
        at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:143) ~[httpclient-4.4.jar:4.4]
        at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:57) ~[httpclient-4.4.jar:4.4]
        at org.apache.http.impl.io.AbstractMessageParser.parse(AbstractMessageParser.java:261) ~[httpcore-4.4.jar:4.4]
        at org.apache.http.impl.DefaultBHttpClientConnection.receiveResponseHeader(DefaultBHttpClientConnection.java:165) ~[httpcore-4.4.jar:4.4]
        at org.apache.http.impl.conn.CPoolProxy.receiveResponseHeader(CPoolProxy.java:167) ~[httpclient-4.4.jar:4.4]
        at org.apache.http.protocol.HttpRequestExecutor.doReceiveResponse(HttpRequestExecutor.java:272) ~[httpcore-4.4.jar:4.4]
```

  使用过期的连接会导致上述错误，该错误仅在Java SDK 2.1.2之前的版本出现。

- 解决方法

  请升级OSS Java SDK到2.1.2及以后版本。

JVM中存在大量org.apache.http.impl.conn.PoolingHttpClientConnectionManager实例

- 错误原因

  ossClient没有关闭导致。

- 解决方法

  主动关闭已执行完毕的ossClient或使用单例模式。

调用OSS Java SDK不响应

- 错误原因

  调用OSS Java SDK不响应。通过 `jstack -l pid` 命令查看堆栈，问题出现在如下的位置：

```
"main" prio=6 tid=0x000000000291e000 nid=0xc40 waiting on condition [0x0000000002dae000]
java.lang.Thread.State: WAITING (parking)
   at sun.misc.Unsafe.park(Native Method)
   - parking to wait for  <0x00000007d85697f8> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$
ConditionObject)
   at java.util.concurrent.locks.LockSupport.park(LockSupport.java:186)
   at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSyn
chronizer.java:2043)
   at org.apache.http.pool.PoolEntryFuture.await(PoolEntryFuture.java:138)
   at org.apache.http.pool.AbstractConnPool.getPoolEntryBlocking(AbstractConnPool.java:306)
   at org.apache.http.pool.AbstractConnPool.access$000(AbstractConnPool.java:64)
   at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(AbstractConnPool.java:192)
   at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(AbstractConnPool.java:185)
   at org.apache.http.pool.PoolEntryFuture.get(PoolEntryFuture.java:107)
   at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.leaseConnection(PoolingHttpClien
tConnectionManager.java:276)
   at org.apache.http.impl.conn.PoolingHttpClientConnectionManager$1.get(PoolingHttpClientConnectio
nManager.java:263)
   at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:190)
   at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
   at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
   at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
   at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
   at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:113
)
   at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:123)
   at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
   at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
   at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:146)
   at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
   at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:229)
   at com.aliyun.oss.OSSClient.getObject(OSSClient.java:629)
   at com.aliyun.oss.OSSClient.getObject(OSSClient.java:617)
   at samples.HelloOSS.main(HelloOSS.java:49)
```

原因是连接池中连接泄漏，可能是使用ossObject后没有关闭。

- 解决方法

  请检查您的程序，确保没有连接泄漏。关闭方法如下：

```
// 读取文件
OSSObject ossObject = ossClient.getObject(bucketName, objectName);
// OSS操作
// 关闭ossObject
ossObject.close();
```

问题排查的具体步骤，请参见OSS Java SDK不响应问题排查。

连接关闭

- 错误原因

  如果您在使用ossClient.getObject时，报类似如下错误：

```
Exception in thread "main" org.apache.http.ConnectionClosedException: Premature end of Content-Leng
th delimited message body (expected: 11990526; received: 202880)
    at org.apache.http.impl.io.ContentLengthInputStream.read(ContentLengthInputStream.java:180)
    at org.apache.http.impl.io.ContentLengthInputStream.read(ContentLengthInputStream.java:200)
    at org.apache.http.impl.io.ContentLengthInputStream.close(ContentLengthInputStream.java:103)
    at org.apache.http.impl.execchain.ResponseEntityProxy.streamClosed(ResponseEntityProxy.java:128)
    at org.apache.http.conn.EofSensorInputStream.checkClose(EofSensorInputStream.java:228)
    at org.apache.http.conn.EofSensorInputStream.close(EofSensorInputStream.java:174)
    at java.io.FilterInputStream.close(FilterInputStream.java:181)
    at java.io.FilterInputStream.close(FilterInputStream.java:181)
    at com.aliyun.oss.event.ProgressInputStream.close(ProgressInputStream.java:147)
    at java.io.FilterInputStream.close(FilterInputStream.java:181)
    at samples.HelloOSS.main(HelloOSS.java:39)
```

原因是两次读取数据间隔时间超过1分钟。OSS会关闭超过1分钟没有发送或接收数据的连接。

- 解决方法

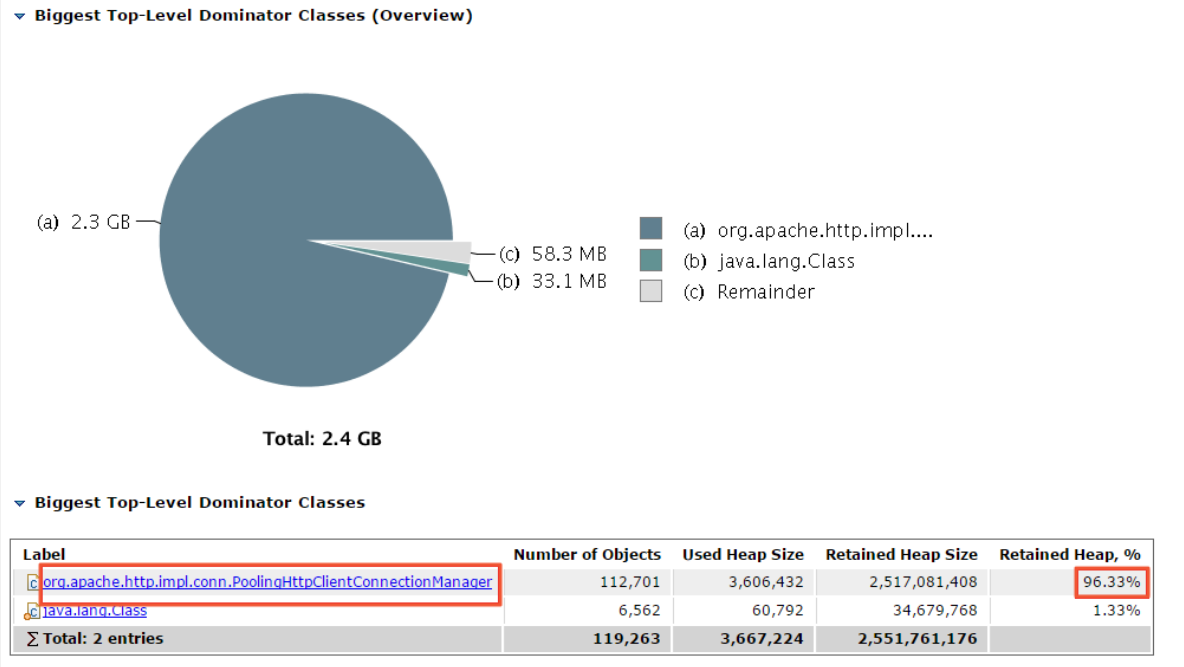
  如果您每次仅读取部分数据，且处理数据的时间不固定，建议使用指定范围读取，避免数据读取时连接关闭。更多信息，请参见范围下载。

内存泄露

- 错误原因

  调用OSS Java SDK的程序，运行一段时间（根据业务量，几小时到几天不等）后内存泄露。 推荐使用Eclipse Memory Analyzer (MAT)分析内存使用情况。更多信息，请参见使用MAT进行堆转储文件分析。

  如果分析结果类似下图所示（PoolingHttpClientConnectionManager占96%的内存），原因是程序中可能多次执行new OSSClient，但是没有调用ossClient.shutdown，造成内存泄漏。



- 解决方法

  new OSSClient操作完成后，请通过shutdown进行关闭，保证new OSSClient和ossClient.shutdown成对使用。

调用ossClient.shutdown报异常InterruptedException

- 错误原因

  OSS Java SDK 2.3.0之前的版本在调用ossClient.shutdown时报如下异常：

```
java.lang.InterruptedException: sleep interrupted
    at java.lang.Thread.sleep(Native Method)
    at com.aliyun.oss.common.comm.IdleConnectionReaper.run(IdleConnectionReaper:76)
```

  原因是ossClient后台线程IdleConnectionReaper会定时关闭闲置连接。IdleConnectionReaper在Sleep时，调用ossClient.shutdown，就会报上面的异常。

- 解决方法

  使用如下代码，忽略该异常：

```
try {
  ossClient.shutdown();
} catch(Exception e) {
}
```

请求出现异常“SDK.ServerUnreachable：Speicified endpoint or uri is not valid”

```
SunshineE/android/01040181-2018-05-07173747-860769-online-1.jpg</cloudUrl><fileSize>4096</fileSize><lossName/><loss
...skipping...
com.aliyuncs.exceptions.ClientException: SDK.ServerUnreachable : Speicified endpoint or uri is not valid.
        at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:201)
        at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:151)
        at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:59)
        at com.aliyuncs.DefaultAcsClient.getAcsResponse(DefaultAcsClient.java:103)
        at com.sunyard.insurance.oss.aliyun.GetSts.assumeRole(GetSts.java:50)
        at com.sunyard.insurance.oss.aliyun.GetSts.getSts(GetSts.java:60)
        at com.sunyard.insurance.ecm.web.action.oss.OssInfoAction.getUpOssInfo(OssInfoAction.java:259)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
```

- 错误原因
  - 用户端并发请求STS过高。
  - 网络到Server端超时。
  - 所使用的STS SDK以及SDK core不是最新版本。
- 解决方法
  - 用户端并发请求STS过高，而用户端的ECS或者本地PC不足以承载当时的并发，降低OSS并发。
  - 用户的网络到Server端有超时现象可以进行抓包验证。
  - 建议将STS SDK及SDK core升级至最新版本。

NoSuchKey

```
case:java.lang.Exception: com.aliyun.oss.OSSException:
Not Foundn[ErrorCode]: NoSuchKeyn
```

- 错误原因

  源文件不存在。

- 解决方法

  参见404错误。

SocketException

```
2018-10-10 16:21:11,127 ERROR com.qunhe.instdeco.plan.maxservice.convert.Converter
downloadOssObject:taskId:LO63HXQKN4BMWM4BAU888888;
com.aliyun.oss.ClientException: SocketException
at com.aliyun.oss.common.utils.ExceptionFactory.createNetworkException(ExceptionFactory.java:71)
at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:128)
at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:123)
at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:149)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:273)
```

- 错误原因

  可能是socket在init阶段就失败了，导致请求没有到达OSS。

- 解决方法

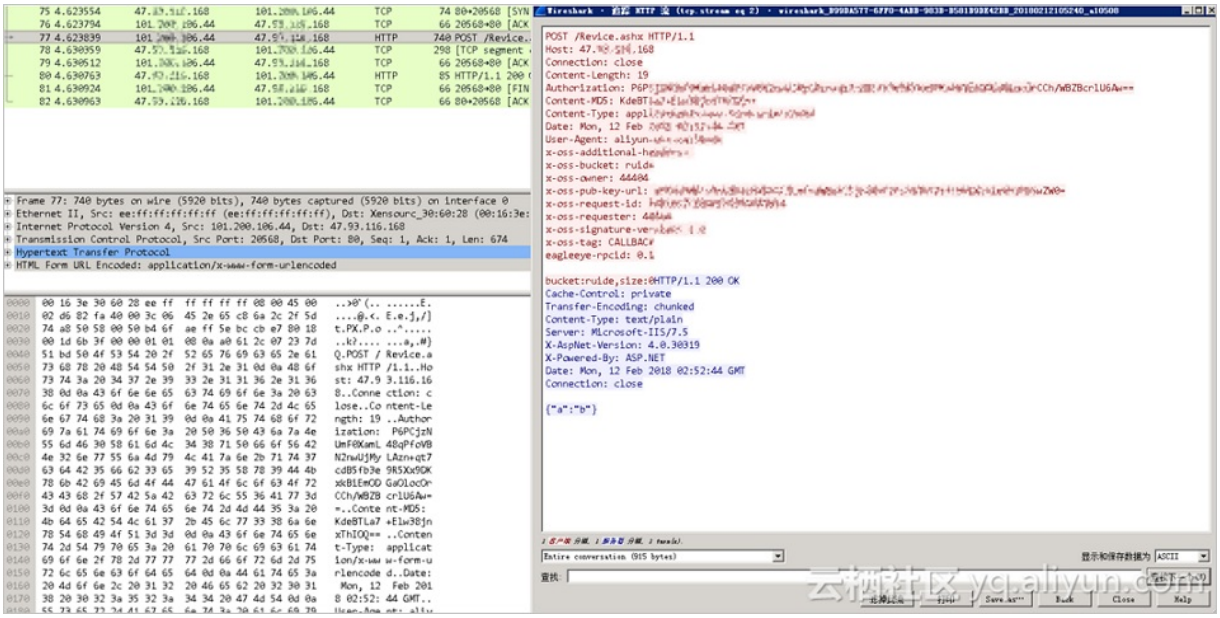
  建议从以下几个方面进行排查：

  - 出现问题时是否出现网络抖动。
  - 主机的socket连接数是否占满。
  - 确认出现问题时连接数是否超过SDK中设置的maxconnection，如果连接数超过maxconnection设置，也会出现socket异常。

  如果以上都没有问题，建议您部署tcpdump或者wireshark抓包，问题复现后再分析数据包。

使用OSS PostObject的callback没有触发回调

使用OSS PostObject的callback没有触发回调，但是通过PutObject用同样的callback触发了回调。一般情况下，如果JSON格式有误或者回调失败，都会返回相应的消息，这里需要分别测试Put和Post回调效果：



- 错误原因

发送请求时callback参数在file下面。

```
LTAIVryeg2DYBhvo
--9431149156168
Content-Disposition: form-data; name="policy"

eyJleHBpcmF0aW9uIjogIjIxMjAtMDEtMDFUMTI6MDA6MDAuMDAwWiIsImNvbmRpdGlvbnMiOiBbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDAsIDEwNDg1NzYwMDAwMF1dfQ==
--9431149156168
Content-Disposition: form-data; name="Signature"

gncf07Apde0vqaMWBtol8XxzoH0=
--9431149156168
Content-Disposition: form-data; name="file"; filename="1.txt"
Content-Type: text/plain


--9431149156168
Content-Disposition: form-data; name="callback"

eyJjYWxsYmFja1VybCI6Imh0dHA6Ly80Ny45My4xMTYuMTY4L1JldmljZS5hc2h4IiwiY2FsbGJhY2tCb2R5Ijoie1wiYnVja2V0XCI9JHtidWNrZXR9LFwic216ZVwiPSR7c216ZX19In0=

12345678910
--9431149156168--
HTTP/1.1 204 No Content
Server: AliyunOSS
```

- 解决方法

  调整callback参数与file的位置。

```
eyJleHBpcmF0aW9uIjogIjIxMjAtMDEtMDFUMTI6MDA6MDAuMDAwWiIsImNvbmRpdGlvbnMiOiBbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDAsIDEwNDg1NzYwMDAwMF1dfQ==
--9431149156168
Content-Disposition: form-data; name="Signature"

gncf07Apde0vqaMWBtol8XxzoH0=
--9431149156168
Content-Disposition: form-data; name="callback"

eyJjYWxsYmFja1VybCI6Imh0dHA6Ly80Ny45My4xMTYuMTY4L1JldmljZS5hc2h4IiwiY2FsbGJhY2tCb2R5Ijoie1wiYnVja2V0XCI9JHtidWNrZXR9LFwic216ZVwiPSR7c216ZX19In0=

--9431149156168
Content-Disposition: form-data; name="file"; filename="1.txt"
Content-Type: text/plain

12345678910
--9431149156168--
HTTP/1.1 200 OK
Server: AliyunOSS
Date: Mon, 12 Feb 2018 06:39:22 GMT
Content-Type: application/json
```

此时测试结果显示业务服务器成功抓取请求。

```
POST /Revice.ashx HTTP/1.1
Host: 47.93.116.168
Connection: close
Content-Length: 26
Authorization: kw8kVDjOdt9CHtRX1
Content-MD5: bXbGjahEaOLNnrG/qOvA
Content-Type: application/x-www-
Date: Mon, 12 Feb 2018 06:40:54
User-Agent: aliyun-oss-callback
x-oss-additional-headers:
x-oss-bucket: ruide
x-oss-owner: 44404
x-oss-pub-key-url: aHR0cHM6Ly9nb
x-oss-request-id: 5A8136F600261B
x-oss-requester: 232379788537179
x-oss-signature-version: 1.0
x-oss-tag: CALLBACK
eagleeye-rpcid: 0.1

{"bucket"=ruide,"size"=11}HTTP/1.1 200 OK
Cache-Control: private
Transfer-Encoding: chunked
Content-Type: text/plain
Server: Microsoft-IIS/7.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 12 Feb 2018 06:40:54 GMT
Connection: close

{"a":"b"}
```

Connection pool shut down

```
Caused by: java.lang.IllegalStateException: Connection pool shut down
 at org.apache.http.util.Asserts.check(Asserts.java:34)
 at org.apache.http.pool.AbstractConnPool.lease(AbstractConnPool.java:184)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.requestConnection(PoolingHttpClient
ConnectionManager.java:251)
 at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:175)
 at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
 at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
 at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
 at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
 at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:124)
 at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:133)
 ... 8 more
```

- 错误原因

  调用ossClient.shutdown()接口后，还继续通过ossClient发送请求。

- 解决方法

  请检查调用逻辑，确保调用了ossClient.shutdown()接口之后，不再通过ossClient发送请求。

使用Java SDK的generatePresignedUrl生成的请求报错Request has expired

- 错误原因

  int类型溢出，导致2038年时间戳问题。

  超出URL设置的过期时间后发起上传请求。

- 解决方法

  如果是int类型溢出，建议Java SDK中过期时长不要超过2038年。

  如果因超出URL设置的过期时间后发起上传请求，建议设置合理的过期时间，确保过期时间大于您的发起请求时间。

报错Invalid Response或Implementation of JAXB-API has not been found on module path or classpath

- 错误原因

  使用了Java 9以上的版本，并且没有添加JAXB依赖。

- 解决方法

  关于如何添加JAXB依赖的更多信息，请参见安装SDK。

如何开启或关闭SDK日志

Apache Log4j定义了不同级别的日志，包括OFF、FATAL、ERROR、WARN、INFO、DEBUG、TRACE、 ALL。

通过配置log4j的属性选择开启或者关闭SDK日志：

```
t > aliyun-oss-java-sdk-demo-mvn > conf > log4j.properties
#

log4j.appender.DRFA=org.apache.log4j.DailyRollingFileAppender
log4j.appender.DRFA.File=${ossdemo.log.dir}/${ossdemo.log.file}

# Rollver at midnight
log4j.appender.DRFA.DatePattern=.yyyy-MM-dd

# 30-day backup
#log4j.appender.DRFA.MaxBackupIndex=30
log4j.appender.DRFA.layout=org.apache.log4j.PatternLayout

# Pattern format: Date LogLevel LoggerName LogMessage
log4j.appender.DRFA.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
# Debugging Pattern format
#log4j.appender.DRFA.layout.ConversionPattern=%d{ISO8601} %-5p %c{2} (%F:%M(%L)) - %m%n


#
# console
# Add "console" to rootlogger above if you want to use this
#

log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{2}: %m%n

log4j.logger.org.apache.http=off

# oss log level
log4j.logger.com.aliyun.oss=DEBUG
```

其他错误

更多错误排查，请参见OSS错误响应。

3.PHP

3.1. 前言

本文介绍对象存储OSS的PHP SDK各种使用场景下的示例代码。

源码地址

请访问GitHub获取源码地址。

示例代码

OSS PHP SDK提供丰富的示例代码，方便您参考或直接使用。示例代码包括以下内容：

示例文件	示例内容
Object.php	文件的相关操作，包括上传文件、下载文件和管理文件等
MultipartUpload.php	分片上传
Signature.php	URL签名授权访问
Callback.php	上传回调
Image.php	图片处理
LiveChannel.php	LiveChannel的相关操作
Bucket.php	存储空间的相关操作，包括创建存储空间、列举存储空间、删除存储空间等。
BucketLifecycle.php	设置、读取和清除存储空间的生命周期
BucketLogging.php	设置、读取和清除存储空间的访问日志
BucketReferer.php	设置、读取和清除存储空间的防盗链
BucketWebsite.php	设置、读取和清除存储空间的静态网站托管
BucketCors.php	设置、读取和清除存储空间的跨域资源访问

3.2. 安装

本文介绍如何安装PHP SDK。

环境准备

OSS PHP SDK适用于PHP 5.3以上版本。本文以PHP 5.6.22为例。

- 安装环境

  您需要安装PHP和cURL扩展：

- 在Windows系统中，请参见Windows下编译使用阿里云 OSS PHP SDK来安装PHP和cURL扩展。在Windows环境中，如果提示找不到指定模块，请在php.ini文件中指定extension_dir为 `C:/Windows/System32/` 。
- 在Ubuntu系统中，请使用apt-get包管理器安装PHP的cURL扩展 `sudo apt-get install php-curl` 。
- 在CentOS系统中，请使用yum包管理器安装PHP的cURL扩展 `sudo yum install php-curl` 。

- 查看版本
  - 通过 `php -v` 命令查看当前的PHP版本。
  - 通过 `php -m` 命令查看cURL扩展是否已经安装好。

下载SDK

- 通过GitHub下载
- 历史版本下载

更多信息请参见OSS API文档。

> **说明** 建议您使用最新版本的SDK。OSS PHP SDK 2.0.0以下版本的文档请从此处下载。

安装SDK

您可以使用以下三种方式安装SDK:

- composer方式
  i. 在项目的根目录运行 `composer require aliyuncs/oss-sdk-php` ，或者在 `composer.json` 文件中添加如下依赖关系。

```
"require": {
  "aliyuncs/oss-sdk-php": "~2.4"
}
```

  ii. 运行 `composer install` ，安装依赖。安装完成后，目录结构如下:

```
.
├── app.php
├── composer.json
├── composer.lock
└── vendor
```

  其中 `app.php` 是您的应用程序， `vendor/` 目录下包含了所依赖的库。您需要在 `app.php` 中添加依赖关系如下:

```
require_once __DIR__ . '/vendor/autoload.php';
```

> **说明**
> - 如果您的项目中已经引用过 `autoload.php` ，则添加了SDK的依赖关系之后，不需要再次引入。
> - 如果使用composer出现网络错误，可以使用composer中国区的镜像源。方法是在命令行执行 `composer config -g repositories.packagist composer http://packagist.phpcomposer.com` 。

- phar方式
  i. 在GitHub中选择相应的版本并下载打包好的phar文件。
  ii. 在代码中引入phar文件：

```
require_once '/path/to/oss-sdk-php.phar';
```

- 源码方式
  i. 在GitHub中选择相应版本并下载打包好的zip文件。
  ii. 解压后的根目录中包含一个 `autoload.php` 文件，在代码中引入此文件：

```
require_once '/path/to/oss-sdk/autoload.php';
```

3.3. 初始化

OssClient是OSS的PHP客户端，用于管理存储空间和文件等OSS资源。

新建OssClient

新建OSSClient时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

- 使用OSS域名新建OssClient

  以下代码用于使用OSS域名新建OSSClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
} catch (OssException $e) {
  print $e->getMessage();
}
```

- 使用自定义域名新建OssClient

  以下代码用于使用自定义域名新建OSSClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
$endpoint = "<yourEndpoint>";
try {
  // true为开启CNAME。CNAME是指将自定义域名绑定到存储空间上。
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, true);
} catch (OssException $e) {
  print $e->getMessage();
}
```

> **说明** 使用自定义域名时，无法使用listBuckets方法。

- 使用STS新建OssClient

  以下代码用于使用STS新建OSSClient：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$securityToken = "<yourSecurityToken>";
try {
  $ossClient = new OssClient(
    $accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
} catch (OssException $e) {
  print $e->getMessage();
}
```

更多信息请参见授权访问。

- 使用代理服务器新建OssClient

  PHP 5.3以上版本支持使用代理服务器新建OssClient，代码如下：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// 代理服务器地址，例如http://<用户名>:<密码>@<代理ip>:<代理端口>。
$requestProxy = "<yourRequestProxy>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, $requestProxy);
} catch (OssException $e) {
  print $e->getMessage();
}
```

配置网络参数

以下代码用于配置OssClient网络参数：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
try {
  $ossClient = new OssClient(
    $accessKeyId, $accessKeySecret, $endpoint);
  // 设置Socket层传输数据的超时时间，单位秒，默认5184000秒。
  $ossClient->setTimeout(3600);
  // 设置建立连接的超时时间，单位秒，默认10秒。
  $ossClient->setConnectTimeout(10);
} catch (OssException $e) {
  print $e->getMessage();
}
```

3.4. 快速入门

本节介绍如何快速使用OSS PHP SDK完成常见操作，如创建存储空间（Bucket）、上传/下载文件（Object）等。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket = "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $ossClient->createBucket($bucket);
} catch (OssException $e) {
   print $e->getMessage();
}
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= " <yourBucketName>";
// <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg
$object = " <yourObjectName>";
$content = "Hi, OSS.";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putObject($bucket, $object, $content);
} catch (OssException $e) {
  print $e->getMessage();
}
```

上传文件详情请参见上传文件。

下载文件

以下代码用于下载文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// <yourObjectName>从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
$object = "<yourObjectName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $content = $ossClient->getObject($bucket, $object);
   print("object content: " . $content);
} catch (OssException $e) {
   print $e->getMessage();
}
```

下载文件详情请参见下载文件。

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $listObjectInfo = $ossClient->listObjects($bucket);
  $objectList = $listObjectInfo->getObjectList();
  if (!empty($objectList)) {
    foreach ($objectList as $objectInfo) {
    print($objectInfo->getKey() . "\t" . $objectInfo->getSize() . "\t" . $objectInfo->getLastModified() . "\n");
    }
  }
} catch (OssException $e) {
  print $e->getMessage();
}
```

列举功能详情请参见管理文件中的列举文件。

删除文件

以下代码用于删除指定文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称
$bucket= "<yourBucketName>";
// <yourObjectName>表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
$object = "<yourObjectName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteObject($bucket, $object);
} catch (OssException $e) {
  print $e->getMessage();
}
```

删除文件详情请参见管理文件中的删除文件。

3.5. 存储空间

3.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

创建存储空间的完整代码请参见GitHub。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 设置存储空间的存储类型为低频访问类型，默认是标准类型。
  $options = array(
    OssClient::OSS_STORAGE => OssClient::OSS_STORAGE_IA
  );
  // 设置存储空间的权限为公共读，默认是私有读写。
  $ossClient->createBucket($bucket, OssClient::OSS_ACL_TYPE_PUBLIC_READ, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.2. 列举存储空间

存储空间（Bucket）按照字母序排列。您可以列举所有的Bucket或指定前缀的Bucket等。

列举所有的Bucket

以下代码用于列举所有的Bucket：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM用户创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 列举所有的Bucket。
  $bucketListInfo = $ossClient->listBuckets();
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
$bucketList = $bucketListInfo->getBucketList();
foreach($bucketList as $bucket) {
  print($bucket->getLocation() . "\t" . $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
```

列举指定前缀的Bucket

以下代码用于列举指定前缀的Bucket：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM用户创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 指定Bucket前缀。
$prefix = "<yourPrefix>";
try{
  $options = array(OssClient::OSS_QUERY_STRING => array(OssClient::OSS_PREFIX => $prefix));
  // 列举指定前缀的Bucket。
  $bucketListInfo = $ossClient->listBuckets($options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
$bucketList = $bucketListInfo->getBucketList();
foreach($bucketList as $bucket) {
  print($bucket->getLocation() . "\t" . $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
```

列举指定Marker之后的Bucket

Marker代表Bucket名称，指定Marker后，列举结果从Marker之后按字母序的第一个开始返回。

以下代码用于列举指定Marker之后的Bucket：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM用户创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 指定Marker。
$marker = "<yourMarker>";
// 列举字母序排在指定Marker之后的Bucket。
try{
  $options = array(OssClient::OSS_QUERY_STRING => array(OssClient::OSS_MARKER => $marker));
  $bucketListInfo = $ossClient->listBuckets($options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
$bucketList = $bucketListInfo->getBucketList();
foreach($bucketList as $bucket) {
  print($bucket->getLocation() . "\t" . $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
```

列举指定个数的Bucket

MAX_KEYS用于限定此次返回Bucket的最大个数，MAX_KEYS取值不能大于1000。如果不指定MAX_KEYS，则默认返回100个Bucket。

以下代码用于列举指定个数的Bucket：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM用户创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 指定列举10个Bucket。
try{
  $options = array(OssClient::OSS_QUERY_STRING => array(OssClient::OSS_MAX_KEYS => 10));
  $bucketListInfo = $ossClient->listBuckets($options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
$bucketList = $bucketListInfo->getBucketList();
foreach($bucketList as $bucket) {
  print($bucket->getLocation() . "\t" . $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
```

3.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

以下代码用于判断存储空间examplebucket是否存在。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $res = $ossClient->doesBucketExist($bucket);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
if ($res === true) {
   print(__FUNCTION__ . ": OK" . "\n");
} else {
   print(__FUNCTION__ . ": FAILED" . "\n");
}
```

3.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $Regions = $ossClient->getBucketLocation($bucket);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Regions);
```

有关地域的详细信息，请参见开发指南基本概念中的地域介绍以及GetBucketLocation API。

3.5.5. 获取存储空间信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息（Info）。

以下代码用于获取存储空间的信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取存储空间的信息，包括存储空间名称（Name）、所在地域（Location）、创建日期（CreateDate）、存储类型（StorageClass）、外网访问域名（ExtranetEndpoint）以及内网访问域名（IntranetEndpoint）等。
  $info = $ossClient->getBucketInfo($bucket);
  printf("bucket name:%s\n", $info->getName());
  printf("bucket location:%s\n", $info->getLocation());
  printf("bucket creation time:%s\n", $info->getCreateDate());
  printf("bucket storage class:%s\n", $info->getStorageClass());
  printf("bucket extranet endpoint:%s\n", $info->getExtranetEndpoint());
  printf("bucket intranet endpoint:%s\n", $info->getIntranetEndpoint());
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取存储空间信息的详情，请参见GetBucketInfo。

3.5.6. 获取存储空间元信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间元信息。

以下代码用于获取存储空间元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $Metas = $ossClient->getBucketMeta($bucket);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Metas);
```

3.5.7. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	OssClient::OSS_ACL_TYPE_PRIVATE
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	OssClient::OSS_ACL_TYPE_PUBLIC_READ
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	OssClient::OSS_ACL_TYPE_PUBLIC_READ_WRITE

以下代码用于设置存储空间的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
// 设置存储空间的权限为私有。
$acl = OssClient::OSS_ACL_TYPE_PRIVATE;
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putBucketAcl($bucket, $acl);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $res = $ossClient->getBucketAcl($bucket);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
print('acl: ' . $res);
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

3.5.8. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
>
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 存储空间名称。
$bucket= "<yourBucketName>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteBucket($bucket);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除存储空间的更多详情，请参见DeleteBucket。

3.5.9. 存储空间标签

您可以通过标签（Tags）标记不同用途的存储空间（Bucket），并对Bucket进行分类管理。

注意事项

Bucket标签使用一组键值对（Key-Value）来标记Bucket。使用Bucket标签时，有如下注意事项：

- 只有Bucket的拥有者及授权的RAM用户才能为Bucket设置标签，否则返回403 Forbidden错误，错误码为AccessDenied。
- 单个Bucket最多可设置20个标签，Key和Value必须为UTF-8编码。
- Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
- Value最大长度为128字节，允许为空。
- PutBucketTags是覆盖语义，即新设置的标签会完全覆盖已有的标签。

有关Bucket标签的详情，请参见存储空间标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\TaggingConfig;
use OSS\Model\Tag;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 设置Bucket标签。
  $config = new TaggingConfig();
  $config->addTag(new Tag("key1", "value1"));
  $config->addTag(new Tag("key2", "value2"));
  $ossClient->putBucketTags($bucket, $config);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关设置Bucket标签的详情，请参见PutBucketTags。

获取Bucket标签

以下代码用于获取Bucket标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\TaggingConfig;
use OSS\Model\Tag;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取Bucket标签信息。
  $config = $ossClient->getBucketTags($bucket);
  print_r($config->getTags());
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取Bucket标签信息的详情，请参见GetBucketTags。

删除Bucket标签

以下代码用于删除Bucket标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\Tag;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 删除Bucket指定标签。
  $tags = array();
  $tags[] = new Tag("key1", "value1");
  $tags[] = new Tag("key2", "value2");
  $ossClient->deleteBucketTags($bucket, $tags);
  // 删除Bucket所有标签。
  $ossClient->deleteBucketTags($bucket);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除Bucket标签的详情，请参见DeleteBucketTags。

3.5.10. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

每条生命周期规则包含以下内容：

- 生命周期规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 生命周期策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
  - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
  - 配置到整个存储空间。此种方式只能创建一条规则。
- 指定文件过期时间。有两种指定方式：
  - 指定一个过期天数N，文件会在其最近更新时间点的N天后过期。
  - 指定一个过期时间点，最近更新时间在该时间点之前的文件全部过期。

- 生命周期规则是否生效。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

生命周期管理的完整代码请参见GitHub。

设置生命周期规则

以下代码用于设置生命周期规则：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\LifecycleConfig;
use OSS\Model\LifecycleRule;
use OSS\Model\LifecycleAction;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 设置规则ID和文件前缀。
$ruleId0 = "rule0";
$matchPrefix0 = "A0/";
$ruleId1 = "rule1";
$matchPrefix1 = "A1/";
$lifecycleConfig = new LifecycleConfig();
$actions = array();
// 距最后修改时间3天后过期。
$actions[] = new LifecycleAction(OssClient::OSS_LIFECYCLE_EXPIRATION, OssClient::OSS_LIFECYCLE_TIMING_DAYS, 3);
$lifecycleRule = new LifecycleRule($ruleId0, $matchPrefix0, "Enabled", $actions);
$lifecycleConfig->addRule($lifecycleRule);
$actions = array();
// 指定日期之前创建的文件过期。
$actions[] = new LifecycleAction(OssClient::OSS_LIFECYCLE_EXPIRATION, OssClient::OSS_LIFECYCLE_TIMING_DATE, '2022-10-12T00:00:00.000Z');
$lifecycleRule = new LifecycleRule($ruleId1, $matchPrefix1, "Enabled", $actions);
$lifecycleConfig->addRule($lifecycleRule);
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $ossClient->putBucketLifecycle($bucket, $lifecycleConfig);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$lifecycleConfig = null;
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $lifecycleConfig = $ossClient->getBucketLifecycle($bucket);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($lifecycleConfig->serializeToXml() . "\n");
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteBucketLifecycle($bucket);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.11. 授权策略

Bucket Policy是对象存储OSS推出的针对Bucket的授权策略，您可以通过Bucket Policy授权其他用户访问您的OSS资源。本文介绍如何设置、获取和删除Bucket Policy。

使用场景

Bucket Policy是基于资源的授权策略。Bucket Policy常见的使用场景如下：

- 向其他账号的RAM用户授权访问。

  您可以授予其他账号的RAM用户访问您的OSS资源的权限。

- 向匿名用户授予带特定IP条件限制的访问权限。

  某些场景下，您需要向匿名用户授予带IP限制的访问策略。例如，企业内部的机密文档，只允许在企业内部访问，不允许在其他区域访问。此时，您可以基于Bucket Policy设置带IP限制的访问策略，从而高效方便地进行授权。

  有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 填写授权策略。
$policy = <<< BBBB
{
 "Version":"1",
 "Statement":[
 {
   "Action":[
   "oss:PutObject",
   "oss:GetObject"
 ],
   "Effect":"Allow",
   "Resource":["acs:oss:*:*:*/user1/*"]
 }
 ]
}
BBBB;
try {
   // 设置授权策略。
   $ossClient->putBucketPolicy($bucket, $policy);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取授权策略。
  $policy = $ossClient->getBucketPolicy($bucket);
  // 打印授权策略内容。
  print($policy);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取Bucket Policy信息详情，请参见Get Bucket Policy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 删除授权策略。
  $ossClient->deleteBucketPolicy($bucket);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

3.5.12. 合规保留策略

对象存储OSS允许针对存储空间（Bucket）设置基于时间的合规保留策略，保护周期为1天到70年。本文介绍如何新建、获取、锁定合规保留策略等。

背景信息

OSS支持WORM（Write Once Read Many）特性，允许您以不可删除、不可篡改的方式保存和使用数据。

当基于时间的合规保留策略创建24小时后未提交锁定，则该策略自动失效。当合规保留策略锁定后，您可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，不允许删除Object及合规保留策略，且无法缩短策略保护周期，仅允许延长Object的保留时间。有关合规保留策略的详情，请参见合规保留策略。

新建合规保留策略

以下代码用于新建合规保留策略：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 创建合规保留策略，指定Object保护天数为30天。
  $wormId = $ossClient->initiateBucketWorm($bucket, 30);
  // 查看合规保留策略Id。
  print($wormId);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关新建合规保留策略的详情，请参见InitiateBucketWorm。

取消未锁定的合规保留策略

以下代码用于取消未锁定的合规保留策略：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 取消合规保留策略。
  $ossClient->abortBucketWorm($bucket);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关取消未锁定的合规保留策略的详情，请参见AbortBucketWorm。

锁定合规保留策略

以下代码用于锁定合规保留策略：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 创建合规保留策略，指定Object保护天数为30天。
  $wormId = $ossClient->initiateBucketWorm($bucket, 30);
  // 锁定合规保留策略。
  $ossClient->completeBucketWorm($bucket, $wormId);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关锁定合规保留策略的详情，请参见CompleteBucketWorm。

获取合规保留策略

以下代码用于获取合规保留策略：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取合规保留策略。
  $config = $ossClient->getBucketWorm($bucket);
  // 打印合规保留策略信息。
  printf("WormId:%s\n", $config->getWormId());
  printf("State:%s\n", $config->getState());
  printf("Day:%d\n", $config->getDay());
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取合规保留策略的详情，请参见GetBucketWorm。

延长Object的保留天数

以下代码用于延长已锁定的合规保留策略中Object的保留天数：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$wormId = "<yourWormId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 将已锁定的合规保留策略中Object的保留天数延长至120天。
  $ossClient->ExtendBucketWorm($bucket, $wormId, 120);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关延长已锁定的合规保留策略中Object的保留天数详情，请参见ExtendBucketWorm。

3.5.13. 请求者付费模式

请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

  如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

  数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 设置请求者付费模式。
  $ossClient->putBucketRequestPayment($bucket, "Requester");
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关设置请求者付费模式的详情，请参见PutBucketRequestPayment。

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取请求者付费模式配置。
  $payer = $ossClient->getBucketRequestPayment($bucket);
  // 打印结果。
  print($payer);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取请求者付费模式配置的详情，请参见GetBucketRequestPayment。

第三方付费访问Object

第三方操作Object时需在HTTP Header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、GetObject和DeleteObject为例，用于指定第三方付费访问Object。其他用于指定第三方付费的Object读写接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 指定为请求者付费模式。
$options = array(
  OssClient::OSS_HEADERS => array(
  OssClient::OSS_REQUEST_PAYER => 'requester',
));
try {
   // PutObject接口指定付费者。
   $content = "hello";
   $ossClient->putObject($bucket, $object, $content, $options);
   // GetObject接口指定付费者。
   $ossClient->getObject($bucket, $object, $options);
   // DeleteObject接口指定付费者。
   $ossClient->deleteObject($bucket, $object, $options);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.14. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。防盗链的完整代码请参见GitHub。

设置防盗链

以下代码用于设置防盗链：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\RefererConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$refererConfig = new RefererConfig();
// 设置允许空Referer。
$refererConfig->setAllowEmptyReferer(true);
// 添加Referer白名单。Referer参数支持通配符星号（*）和问号（？）。
$refererConfig->addReferer("www.aliiyun.com");
$refererConfig->addReferer("www.aliiyuncs.com");
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putBucketReferer($bucket, $refererConfig);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\RefererConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$refererConfig = null;
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $refererConfig = $ossClient->getBucketReferer($bucket);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($refererConfig->serializeToXml() . "\n");
```

清空防盗链

以下代码用于清空防盗链：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\RefererConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$refererConfig = new RefererConfig();
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
  $ossClient->putBucketReferer($bucket, $refererConfig);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.15. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

更多关于访问日志的介绍，请参见开发指南中的设置访问日志记录。访问日志的完整代码请参见GitHub。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$option = array();
// 设置存放日志文件的存储空间。
$targetBucket = "<yourBucketName>";
$targetPrefix = "access.log";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   // 开启访问日志记录。
   $ossClient->putBucketLogging($bucket, $targetBucket, $targetPrefix, $option);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$loggingConfig = null;
$options = array();
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 查看访问日志设置。
  $loggingConfig = $ossClient->getBucketLogging($bucket, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($loggingConfig->serializeToXml() . "\n");
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   // 关闭访问日志记录。
   $ossClient->deleteBucketLogging($bucket);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.16. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

以下场景的完整代码请参见GitHub。

设置静态网站托管

以下代码用于设置静态网站托管：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\WebsiteConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 设置静态网站托管：索引页面为index.html, 错误页面为error.html。
$websiteConfig = new WebsiteConfig("index.html", "error.html");
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $ossClient->putBucketWebsite($bucket, $websiteConfig);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$websiteConfig = null;
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $websiteConfig = $ossClient->getBucketWebsite($bucket);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($websiteConfig->serializeToXml() . "\n");
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $ossClient->deleteBucketWebsite($bucket);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.17. 跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

> **说明** 跨域资源共享的完整代码请参见GitHub。关于跨域资源共享的更多信息，请参见开发指南中的设置跨域访问和API参考中的PutBucketcors。

设置跨域资源共享规则

以下代码用于设置目标存储空间examplebucket的跨域资源共享规则。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\CorsConfig;
use OSS\Model\CorsRule;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
$corsConfig = new CorsConfig();
$rule = new CorsRule();
// 设置允许跨域请求的响应头。AllowedHeader可以设置多个，每个AllowedHeader中最多只能使用一个通配符星号
（*）。
// 建议无特殊需求时设置AllowedHeader为星号（*）。
$rule->addAllowedHeader("*");
// 设置允许用户从应用程序中访问的响应头。ExposeHeader可以设置多个，ExposeHeader中不支持使用通配符星号
（*）。
$rule->addExposeHeader("x-oss-header");
// 设置允许的跨域请求的来源。AllowedOrigin可以设置多个，每个AllowedOrigin中最多只能使用一个通配符星号（*
）。
$rule->addAllowedOrigin("https://www.example.com:8080");
$rule->addAllowedOrigin("https://*.aliyun.com");
// 设置AllowedOrigin为星号（*）时，表示允许所有域的来源。
//$rule->addAllowedOrigin("*");
// 设置允许的跨域请求方法。
$rule->addAllowedMethod("POST");
// 设置浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，单位为秒。
$rule->setMaxAgeSeconds(10);
// 每个Bucket最多支持添加10条规则。
$corsConfig->addRule($rule);
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 已存在的规则将被覆盖。
  $ossClient->putBucketCors($bucket, $corsConfig);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取跨域资源共享规则

以下代码用于获取目标存储空间examplebucket的跨域资源共享规则。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
$corsConfig = null;
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $corsConfig = $ossClient->getBucketCors($bucket);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
print($corsConfig->serializeToXml() . "\n");
```

删除跨域资源共享规则

以下代码用于删除目标存储空间examplebucket的所有跨域资源共享规则。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteBucketCors($bucket);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.5.18. 自定义域名绑定

本文介绍如何进行自定义域名绑定。

您可以通过添加CNAME记录将自定义域名绑定到指定的存储空间上，从而使用自己的域名访问OSS资源。例如您的域名是my-domain.com，之前访问所有图片都是通过 `http://img.my-domain.com/x.jpg` 的格式访问，在将数据迁移到OSS之后，通过绑定自定义域名的方式，您仍然可以使用原来的地址访问图片。

详情请参见开发指南中的绑定自定义域名。

添加CNAME记录

以下代码用于为存储空间添加CNAME记录：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 设置自定义域名。
$myDomain = '<yourDomainName>';
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
 // 添加CNAME记录。
   $ossClient->addBucketCname($bucket, $myDomain);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

查看CNAME记录

以下代码用于获取存储空间已添加的CNAME记录：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
 // 查看CNAME记录。
   $cnameConfig = $ossClient->getBucketCname($bucket);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
var_dump($cnameConfig);
print(__FUNCTION__ . ": OK" . "\n");
```

删除CNAME记录

以下代码用于删除存储空间的CNAME记录：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 设置自定义域名。
$myDomain = '<yourDomainName>';
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
 // 删除CNAME记录。
  $ossClient->deleteBucketCname($bucket, $myDomain);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.6. 上传文件

3.6.1. 概述

OSS中，操作的基本数据单元是文件（Object）。OSS PHP SDK提供了丰富的文件上传方式：

- 简单上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 分片上传：文件最大不能超过48.8TB。当文件较大时，请使用分片上传。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。上传完成后，您还可以进行上传回调。

3.6.2. 简单上传

简单上传分为字符串上传和文件上传。本文介绍如何进行字符串上传和文件上传。

字符串上传

以下代码用于将字符串上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 设置存储空间名称。
$bucket= "<yourBucketName>";
// 设置文件名称。
$object = "<yourObjectName>";
// 配置文件内容。
$content = "Hello OSS";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putObject($bucket, $object, $content);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
// 上传时可以设置相关的headers，例如设置访问权限为private和自定义元信息。
$options = array(
  OssClient::OSS_HEADERS => array(
    'x-oss-object-acl' => 'private',
    'x-oss-meta-info' => 'your info'
  ),
);
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putObject($bucket, $object, $content, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

文件上传

以下代码用于上传文件到OSS：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 设置存储空间名称。
$bucket= "<yourBucketName>";
// 设置文件名称。
$object = "<yourObjectName>";
// <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
$filePath = "<yourLocalFile>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->uploadFile($bucket, $object, $filePath);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

追加上传本地文件

以下代码用于通过追加上传的方式将本地文件examplefilea.txt、examplefileb.txt和examplefilec.txt的内容依次上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
// 设置Object完整路径。Object完整路径中不能包含Bucket名称。
$object = "exampleobject.txt";
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
$filePath = "D:\\localpath\\examplefilea.txt";
$filePath1 = "D:\\localpath\\examplefileb.txt";
$filePath1 = "D:\\localpath\\examplefilec.txt";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 第一次追加上传。第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
  $position = $ossClient->appendFile($bucket, $object, $filePath, 0);
  $position = $ossClient->appendFile($bucket, $object, $filePath1, $position);
  $position = $ossClient->appendFile($bucket, $object, $filePath2, $position);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

关于Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。关于Object名称命名规范的更多信息，请参见对象（Object）。

追加上传字符串

以下代码用于通过追加上传的方式将字符串依次上传到目标存储空间examplebucket中的strexampleobject.txt文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
// 设置Object完整路径。Object完整路径中不能包含Bucket名称。
$object = "strexampleobject.txt";
// 设置依次追加上传的字符串。第一次、第二次以及第三次追加上传后获取的文件内容分别为Hello OSS、Hi OSS以及
OSS OK。
$content_array = array('Hello OSS', 'Hi OSS', 'OSS OK');
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 第一次追加上传。第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
  $position = $ossClient->appendObject($bucket, $object, $content_array[0], 0);
  $position = $ossClient->appendObject($bucket, $object, $content_array[1], $position);
  $position = $ossClient->appendObject($bucket, $object, $content_array[2], $position);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.6.4. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用$ossClient->initiateMultipartUpload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

   调用$ossClient->uploadPart方法上传分片数据。

> **说明**
>
> - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
> - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
> - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   调用$ossClient->completeMultipartUpload方法将所有分片合并成完整的文件。

分片上传的完整代码请参见GitHub。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Core\OssUtil;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadFile = "<yourLocalFile>";
/**
 * 步骤1：初始化一个分片上传事件，获取uploadId。
 */
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  //返回uploadId。uploadId是分片上传事件的唯一标识，您可以根据uploadId发起相关的操作，如取消分片上传、查询分片上传等。
  $uploadId = $ossClient->initiateMultipartUpload($bucket, $object);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": initiateMultipartUpload FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": initiateMultipartUpload OK" . "\n");
/*
 * 步骤2：上传分片。
 */
```

```
$partSize = 10 * 1024 * 1024;
$uploadFileSize = sprintf('%u',filesize($uploadFile));
$pieces = $ossClient->generateMultiuploadParts($uploadFileSize, $partSize);
$responseUploadPart = array();
$uploadPosition = 0;
$isCheckMd5 = true;
foreach ($pieces as $i => $piece) {
  $fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
  $toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
  $upOptions = array(
    // 上传文件。
    $ossClient::OSS_FILE_UPLOAD => $uploadFile,
    // 设置分片号。
    $ossClient::OSS_PART_NUM => ($i + 1),
    // 指定分片上传起始位置。
    $ossClient::OSS_SEEK_TO => $fromPos,
    // 指定文件长度。
    $ossClient::OSS_LENGTH => $toPos - $fromPos + 1,
    // 是否开启MD5校验，true为开启。
    $ossClient::OSS_CHECK_MD5 => $isCheckMd5,
  );
  // 开启MD5校验。
  if ($isCheckMd5) {
    $contentMd5 = OssUtil::getMd5SumForFile($uploadFile, $fromPos, $toPos);
    $upOptions[$ossClient::OSS_CONTENT_MD5] = $contentMd5;
  }
  try {
    // 上传分片。
    $responseUploadPart[] = $ossClient->uploadPart($bucket, $object, $uploadId, $upOptions);
  } catch(OssException $e) {
    printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} FAILED\n");
    printf($e->getMessage() . "\n");
    return;
  }
  printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} OK\n");
}
// $uploadParts是由每个分片的ETag和分片号（PartNumber）组成的数组。
$uploadParts = array();
foreach ($responseUploadPart as $i => $eTag) {
  $uploadParts[] = array(
    'PartNumber' => ($i + 1),
    'ETag' => $eTag,
  );
}
/**
 * 步骤3：完成上传。
 */
try {
  // 执行completeMultipartUpload操作时，需要提供所有有效的$uploadParts。OSS收到提交的$uploadParts后
，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
  $ossClient->completeMultipartUpload($bucket, $object, $uploadId, $uploadParts);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": completeMultipartUpload FAILED\n");
  printf($e->getMessage() . "\n");
```

```
    return;
}
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

分片上传本地文件

以下代码用于分片上传本地文件到OSS。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
    require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
    require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$file = "<yourLocalFile>";
$options = array(
    OssClient::OSS_CHECK_MD5 => true,
    OssClient::OSS_PART_SIZE => 1,
);
try{
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
    $ossClient->multiuploadFile($bucket, $object, $file, $options);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

分片上传目录

以下代码用于分片上传本地目录（包含此目录下的所有文件）到OSS。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$localDirectory = ".";
$prefix = "samples/codes";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->uploadDir($bucket, $prefix, $localDirectory);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

取消分片上传事件

您可以调用$ossClient->abortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用此uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$upload_id = "<yourUploadId>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->abortMultipartUpload($bucket, $object, $upload_id);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

列举已上传的分片

以下代码用于列举已上传的分片。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadId = "<yourUploadId>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $listPartsInfo = $ossClient->listParts($bucket, $object, $uploadId);
  foreach ($listPartsInfo->getListPart() as $partInfo) {
    print($partInfo->getPartNumber() . "\t" . $partInfo->getSize() . "\t" . $partInfo->getETag() . "\t" . $partInfo
->getLastModified() . "\n");
  }
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

列举分片上传事件

以下代码用于列举分片上传事件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$options = array(
   'delimiter' => '/',
   'max-uploads' => 100,
   'key-marker' => '',
   'prefix' => '',
   'upload-id-marker' => ''
);
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $listMultipartUploadInfo = $ossClient->listMultipartUploads($bucket, $options);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": listMultipartUploads FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
printf(__FUNCTION__ . ": listMultipartUploads OK\n");
$listUploadInfo = $listMultipartUploadInfo->getUploads();
var_dump($listUploadInfo);
```

$options的参数说明请参见下表。

参数	说明
delimiter	用于对文件名称进行分组的一个字符。所有文件名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素。
key-marker	所有文件名称的字典序大于key-marker参数值的分片上传事件。与upload-id-marker参数一同使用，用于指定返回结果的起始位置。
max-uploads	限定此次返回分片上传事件的最大个数，默认值和最大值均为1000。

参数	说明
prefix	限定返回的文件名称必须以指定的prefix作为前缀。 **说明** 使用prefix查询时，返回的文件名称中仍会包含prefix。
upload-id-marker	与key-marker参数一同使用，用于指定返回结果的起始位置。 如果未设置key-marker参数，则此参数无效。如果设置了key-marker参数，则查询结果中包含： • 文件名称的字典序大于key-marker参数值的所有文件。 • 文件名称等于key-marker参数值且uploadId比upload-id-marker参数值大的所有分片上传事件。

3.6.5. 上传回调

本文介绍如何在简单上传及分片上传中使用上传回调。

简单上传设置回调

以下代码用于在简单上传设置回调。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 上传文件时设置回调。
// callbackUrl为回调服务器地址，如http://oss-demo.aliyuncs.com:23450或http://127.0.0.1:9090。
//（可选）callbackHost为回调请求消息头中Host的值，即您的服务器配置Host的值。
$url =
  '{
    "callbackUrl":"<yourCallbackServerUrl>",
    "callbackHost":"<yourCallbackServerHost>",
    "callbackBody":"bucket=${bucket}&object=${object}&etag=${etag}&size=${size}&mimeType=${mimeType}&imageInfo.height=${imageInfo.height}&imageInfo.width=${imageInfo.width}&imageInfo.format=${imageInfo.format}&my_var1=${x:var1}&my_var2=${x:var2}",
    "callbackBodyType":"application/x-www-form-urlencoded"
  }';
// 设置发起回调请求的自定义参数，由Key和Value组成，Key必须以x:开始。
$var =
  '{
    "x:var1":"value1",
    "x:var2":"值2"
  }';
$options = array(OssClient::OSS_CALLBACK => $url,
  OssClient::OSS_CALLBACK_VAR => $var
);
$result = $ossClient->putObject($bucket, $object, file_get_contents(__FILE__), $options);
print_r($result['body']);
print_r($result['info']['http_code']);
```

分片上传设置回调

以下代码用于分片上传设置回调。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
```

```
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Core\OssUtil;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadFile = "<yourLocalFile>";
/**
 * 步骤1：初始化一个分片上传事件，获取uploadId。
 */
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
//返回uploadId，它是分片上传事件的唯一标识，您可以根据这个ID来发起相关的操作，如取消分片上传、查询分片上传等。
$uploadId = $ossClient->initiateMultipartUpload($bucket, $object);
print(__FUNCTION__ . ": initiateMultipartUpload OK" . "\n");
/*
 * 步骤2：上传分片。
 */
$partSize = 10 * 1024 * 1024;
$uploadFileSize = sprintf('%u',filesize($uploadFile));
$pieces = $ossClient->generateMultiuploadParts($uploadFileSize, $partSize);
$responseUploadPart = array();
$uploadPosition = 0;
$isCheckMd5 = true;
foreach ($pieces as $i => $piece) {
  $fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
  $toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
  $upOptions = array(
    $ossClient::OSS_FILE_UPLOAD => $uploadFile,
    $ossClient::OSS_PART_NUM => ($i + 1),
    $ossClient::OSS_SEEK_TO => $fromPos,
    $ossClient::OSS_LENGTH => $toPos - $fromPos + 1,
    $ossClient::OSS_CHECK_MD5 => $isCheckMd5,
  );
  // MD5校验。
  if ($isCheckMd5) {
    $contentMd5 = OssUtil::getMd5SumForFile($uploadFile, $fromPos, $toPos);
    $upOptions[$ossClient::OSS_CONTENT_MD5] = $contentMd5;
  }
  try {
    // 上传分片。
    $responseUploadPart[] = $ossClient->uploadPart($bucket, $object, $uploadId, $upOptions);
  } catch(OssException $e) {
    printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} FAILED\n");
    printf($e->getMessage() . "\n");
    return;
  }
  printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} OK\n");
}
// $uploadParts是由每个分片的ETag和分片号（PartNumber）组成的数组。
```

```
$uploadParts = array();
foreach ($responseUploadPart as $i => $eTag) {
  $uploadParts[] = array(
    'PartNumber' => ($i + 1),
    'ETag' => $eTag,
  );
}
/**
 * 步骤3：完成上传。
 */
// callbackUrl为回调服务器地址，如http://oss-demo.aliyuncs.com:23450或http://127.0.0.1:9090。
//（可选）callbackHost为回调请求消息头中Host的值，即您的服务器配置Host的值。
$json =
  '{
    "callbackUrl":"<yourCallbackServerUrl>",
    "callbackHost":"<yourCallbackServerHost>",
    "callbackBody":"{\"mimeType\":${mimeType},\"size\":${size},\"x:var1\":${x:var1},\"x:var2\":${x:var2}}",
    "callbackBodyType":"application/json"
  }';
// 设置发起回调请求的自定义参数，由Key和Value组成，Key必须以x:开始。
$var =
  '{
    "x:var1":"value1",
    "x:var2":"值2"
  }';
$options = array(OssClient::OSS_CALLBACK => $json,
  OssClient::OSS_CALLBACK_VAR => $var);
// 在执行该操作时，需要提供所有有效的$uploadParts。OSS收到提交的$uploadParts后，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
$ossClient->completeMultipartUpload($bucket, $object, $uploadId, $uploadParts, $options);
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

3.7. 下载文件

3.7.1. 概述

文件下载的完整代码请参见GitHub。

OSS PHP SDK提供了丰富的文件下载方式：

- 下载OSS文件到本地文件
- 下载OSS文件到本地内存
- 范围下载
- 限定条件下载

3.7.2. 下载到本地文件

本文介绍如何使用PHP SDK将指定的文件（Object）下载到本地。

以下代码用于把指定的OSS文件下载到本地：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// <yourObjectName>表示您从OSS下载文件时，需要指定文件所在存储空间的完整名称，即包含文件后缀在内的完整路径，例如abc/123.jpg。
$object = "<yourObjectName>";
// <yourLocalFile>本地指定的文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
$localfile = "<yourLocalFile>";
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $localfile
  );
// 使用try catch捕获异常，如果捕获到异常，则说明下载失败；如果没有捕获到异常，则说明下载成功。
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK, please check localfile: 'upload-test-object-name.txt'" . "\n");
```

3.7.3. 下载到本地内存

本文介绍如何使用PHP SDK将指定的文件下载到本地内存。

以下代码用于把指定的OSS文件下载到本地内存并打印出文件内容：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 文件名称。
$object = "<yourObjectName>";
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $content = $ossClient->getObject($bucket, $object);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print($content);
print(__FUNCTION__ . ": OK" . "\n");
```

说明 由于内存断电数据就会丢失，若想将下载文件保存在本地磁盘请参照下载到本地文件。

3.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

以下代码用于下载文件[0, 4]的内容到本地内存：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
// 获取0~4字节（包括0和4），共5个字节的数据。如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
$options = array(OssClient::OSS_RANGE => '0-4');
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $content = $ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print ($content);
print(__FUNCTION__ . ": OK" . "\n");
```

3.7.5. 限定条件下载

本文介绍如何使用限定条件下载OSS文件。

下载文件时，可以指定一个或多个限定条件。满足限定条件则下载，条件不满足则返回错误且不会触发下载行为。可用的限定条件如下：

参数	描述	如何设置
If-Modified-Since	如果指定的时间早于实际修改时间，则正常传输文件，否则返回错误（304 Not modified）。	OssClient::OSS_IF_MODIFIED_SINCE
If-Unmodified-Since	如果指定的时间等于或者晚于文件实际修改时间，则正常传输文件，否则返回错误（412 Precondition failed）。	OssClient::OSS_IF_UNMODIFIED_SINCE
If-Match	如果指定的ETag和OSS文件的ETag匹配，则正常传输文件，否则返回错误（412 Precondition failed）。	OssClient::OSS_IF_MATCH

参数	描述	如何设置
If-None-Match	如果指定的ETag和OSS文件的ETag不匹配，则正常传输文件，否则返回错误（304 Not modified）。	OssClient::OSS_IF_NONE_MATCH

说明

- If-Modified-Since和If-Unmodified-Since可以同时存在。If-Match和If-None-Match也可以同时存在。
- ETag可以通过$ossClient->getObjectMeta方法获取。

条件下载既可以下载OSS文件到本地内存，也可以下载到本地文件。以下代码用于限定条件下载：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
try{
  $options = array(
    OssClient::OSS_HEADERS => array(
      OssClient::OSS_IF_MODIFIED_SINCE => "Fri, 13 Nov 2015 14:47:53 GMT"),
  );
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $content = $ossClient->getObject($bucket, $object, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print ($content);
print(__FUNCTION__ . ": OK" . "\n");
```

3.8. 管理文件

3.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理软链接
- 开启MD5验证

3.8.2. 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $exist = $ossClient->doesObjectExist($bucket, $object);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($exist);
```

3.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	public-read-write

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于设置指定文件的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
// 设置文件的访问权限为公共读，默认为继承Bucket。
$acl = "public-read";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putObjectAcl($bucket, $object, $acl);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $objectAcl = $ossClient->getObjectAcl($bucket, $object);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($objectAcl);
```

3.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP Header和自定义元信息。本文介绍如何设置、修改、以及获取文件元信息。

文件元信息详情请参见文件元信息。

设置文件元信息

以下代码用于设置文件元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// yourObjectName表示想要设置文件元信息的Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，如填写为example/test.jpg。
$object = "<yourObjectName>";
$content = file_get_contents(__FILE__);
$options = array(
  OssClient::OSS_HEADERS => array(
    'Expires' => '2012-10-01 08:00:00',
    'Content-Disposition' => 'attachment; filename="xxxxxx"',
    'x-oss-meta-self-define-title' => 'user define meta info',
  ));
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putObject($bucket, $object, $content, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

修改文件元信息

以下代码用于修改文件元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
/// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$fromBucket = "<yourFromBucketName>";
$fromObject = "<yourFromObjectName>";
$toBucket = $fromBucket;
$toObject = $fromObject;
$copyOptions = array(
  OssClient::OSS_HEADERS => array(
    'Expires' => '2018-10-01 08:00:00',
    'Content-Disposition' => 'attachment; filename="xxxxxx"',
    'x-oss-meta-location' => 'location',
    // 指定设置目标Object元信息的方式，此处以指定为'REPLACE'为例，表示忽略源Object的元数据，直接采用请求中指定的元数据。若设置为COPY，表示复制源Object的元数据到目标Object。
    'x-oss-metadata-directive' => 'REPLACE',
  ),
);
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->copyObject($fromBucket, $fromObject, $toBucket, $toObject, $copyOptions);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取文件元信息

以下代码用于获取文件元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 获取文件的全部元信息。
  $objectMeta = $ossClient->getObjectMeta($bucket, $object);
  print_r($objectMeta);
  // 获取文件的部分元信息。
  $objectMeta = $ossClient->getSimplifiedObjectMeta($bucket, $object);
  print_r($objectMeta);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.8.5. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何通过CopyObject接口将文件（Object）转换为指定的存储类型。

> **说明** 有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

转换标准或低频访问为归档

以下代码用于将标准或低频访问转换为归档类型：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 填写Bucket名称。
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如destfolder/exampleobject.txt。
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 指定转换后的文件存储类型，例如指定为归档存储类型（Archive）。
  $copyOptions = array(
    OssClient::OSS_HEADERS => array(
      'x-oss-storage-class' => 'Archive',
      'x-oss-metadata-directive' => 'REPLACE',
    ),
  );
  $ossClient->copyObject($bucket, $object, $bucket, $object, $copyOptions);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关Bucket命名规范的详情，请参见存储空间（Bucket）。有关Object命名规范的详情，请参见对象（Object）。

转换归档为低频访问或标准

以下代码用于将归档转换为低频访问或标准存储类型：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 填写Bucket名称。
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如destfolder/exampleobject.txt。
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 解冻归档文件。
  $ossClient->restoreObject($bucket, $object);
  // 等待解冻完成。
  $waitTime = 100;
  while ($waitTime > 0) {
    $meta = $ossClient->getObjectMeta($bucket, $object);
    if (array_key_exists("x-oss-restore", $meta) &&
      strcmp($meta["x-oss-restore"], "ongoing-request=\"true\"") == 0){
      print("In restore status, ". $meta["x-oss-restore"]. "\n");
      sleep(5);
    } else {
      break;
    }
    $waitTime -= 5;
  }
  // 指定转换后的文件存储类型，例如指定为低频访问（IA）或标准存储类型（Standard）。
  $copyOptions = array(
    OssClient::OSS_HEADERS => array(
      'x-oss-storage-class' => 'IA',
      'x-oss-metadata-directive' => 'REPLACE',
    ),
  );
  $ossClient->copyObject($bucket, $object, $bucket, $object, $copyOptions);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.8.6. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$content = "Hello OSS";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 指定PutObject操作时是否覆盖同名Object。
  // 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
  // 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
  // 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，则报错FileAlreadyExists。
  $options = array(
    OssClient::OSS_HEADERS => array(
      'x-oss-forbid-overwrite' => 'true'
    ),
  );
  $ossClient->putObject($bucket, $object, $content, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
```

```
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Core\OssUtil;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$uploadFile = "<yourLocalFile>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 指定初始化分片上传操作时是否覆盖同名Object。
  // 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
  // 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
  // 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，则报错FileAlreadyExists。
  $options = array(
    OssClient::OSS_HEADERS => array(
      'x-oss-forbid-overwrite' => 'true'
    ),
  );
  $uploadId = $ossClient->initiateMultipartUpload($bucket, $object, $options);
  // 上传分片。
  $partSize = 1 * 1024 * 1024;
  $uploadFileSize = sprintf('%u',filesize($uploadFile));
  $pieces = $ossClient->generateMultiuploadParts($uploadFileSize, $partSize);
  $responseUploadPart = array();
  $uploadPosition = 0;
  $isCheckMd5 = true;
  foreach ($pieces as $i => $piece) {
    $fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
    $toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
    $upOptions = array(
      $ossClient::OSS_FILE_UPLOAD => $uploadFile,
      $ossClient::OSS_PART_NUM => ($i + 1),
      $ossClient::OSS_SEEK_TO => $fromPos,
      $ossClient::OSS_LENGTH => $toPos - $fromPos + 1,
    );
    $responseUploadPart[] = $ossClient->uploadPart($bucket, $object, $uploadId, $upOptions);
  }
  $uploadParts = array();
  foreach ($responseUploadPart as $i => $eTag) {
    $uploadParts[] = array(
      'PartNumber' => ($i + 1),
      'ETag' => $eTag,
    );
```

```
    );
  }
  // 指定完成分片上传操作时是否覆盖同名Object。
  // 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
  // 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
  // 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，则报错FileAlreadyE
xists。
  $options = array(
    OssClient::OSS_HEADERS => array(
      'x-oss-forbid-overwrite' => 'true'
    ),
  );
  $ossClient->completeMultipartUpload($bucket, $object, $uploadId, $uploadParts, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.8.7. 列举文件

本文介绍如何列举指定存储空间（Bucket）下的所有文件或者符合条件的文件（Object）。

列举所有文件

以下代码用于列举指定存储空间下的所有文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$nextMarker = '';
while (true) {
  try {
    $options = array(
      'delimiter' => '',
      'marker' => $nextMarker,
    );
    $listObjectInfo = $ossClient->listObjects($bucket, $options);
  } catch (OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
  }
  // 得到nextMarker，从上一次listObjects读到的最后一个文件的下一个文件开始继续获取文件列表。
  $nextMarker = $listObjectInfo->getNextMarker();
  $listObject = $listObjectInfo->getObjectList();
  $listPrefix = $listObjectInfo->getPrefixList();
  if (!empty($listObject)) {
    print("objectList:\n");
    foreach ($listObject as $objectInfo) {
      print($objectInfo->getKey() . "\n");
    }
  }
  if (!empty($listPrefix)) {
    print("prefixList: \n");
    foreach ($listPrefix as $prefixInfo) {
      print($prefixInfo->getPrefix() . "\n");
    }
  }
  if ($listObjectInfo->getIsTruncated() !== "true") {
    break;
  }
}
```

说明

- 上述示例代码中的$options为可选参数，可以用来列举整个bucket或者某个目录下的所有文件。
- 使用listObjects方法主要用于获取文件列表，如需获取其他属性，如文件名，文件大小等，请参见文档管理文件元信息。

列举指定条件的文件

以下代码用于列举指定条件的文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
$prefix = 'dir/';
$delimiter = '/';
$nextMarker = '';
$maxkeys = 10;
$options = array(
  'delimiter' => $delimiter,
  'prefix' => $prefix,
  'max-keys' => $maxkeys,
  'marker' => $nextMarker,
);
try {
  $listObjectInfo = $ossClient->listObjects($bucket, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
$objectList = $listObjectInfo->getObjectList(); // object list
$prefixList = $listObjectInfo->getPrefixList(); // directory list
if (!empty($objectList)) {
  print("objectList:\n");
  foreach ($objectList as $objectInfo) {
    print($objectInfo->getKey() . "\n");
  }
}
if (!empty($prefixList)) {
  print("prefixList: \n");
  foreach ($prefixList as $prefixInfo) {
    print($prefixInfo->getPrefix() . "\n");
  }
}
```

上述例子中的$options包含的参数如下：

参数	说明	是否必需
delimiter	对文件名称进行分组的一个字符。CommonPrefixes是以delimiter结尾，并有共同前缀的文件集合。	否
prefix	本次查询结果的前缀。	否
max-keys	列举文件的最大个数。默认为100，最大值为1000。	否
marker	标明本次列举文件的起点。	否

3.8.8. 删除文件

本文介绍如何删除文件。

警告 请您谨慎使用删除操作，文件一旦删除将无法恢复。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteObject($bucket, $object);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除多个文件

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$objects = array();
$objects[] = "<yourObjectName1>";
$objects[] = "<yourObjectName2>";
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->deleteObjects($bucket, $objects);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

说明 如果使用 try{}catch{} 获取错误时抛出 Exception 则表示没有成功删除单个或多个文件。同时，您还可以根据 $e->getMessage 来分析错误原因。

3.8.9. 拷贝文件

本文介绍如何拷贝文件。

拷贝小文件

对于小于 1GB 的文件，您可以通过 $ossClient->copyObject 方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

说明

以下代码用于拷贝小文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$from_bucket = "<yourFromBucketName>";
$from_object = "<yourFromObjectName>";
$to_bucket = $bucket;
$to_object = $from_object . '.copy';
try{
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $ossClient->copyObject($from_bucket, $from_object, $to_bucket, $to_object);
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关拷贝小文件的详情，请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

1. 通过$ossClient->initiateMultipartUpload初始化分片拷贝任务。
2. 通过$ossClient->uploadPartCopy进行分片拷贝。除最后一个分片外，其它分片都要大于100KB。
3. 通过$ossClient->completeMultipartUpload提交分片拷贝任务。

以下代码用于分片拷贝：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
```

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$src_bucket = "<yourSourceBucketName>";
$src_object = "<yourSourceObjectName>";
$dst_bucket = "<yourDestinationBucketName>";
$dst_object = "<yourDestinationObjectName>";
// 根据实际情况设置分片大小。
$part_size = 256*1024*1024;
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $objectMeta = $ossClient->getObjectMeta($src_bucket, $src_object);
  $length = $objectMeta['content-length'] + 0;
  // 初始化分片。
  $upload_id = $ossClient->initiateMultipartUpload($dst_bucket, $dst_object);
  // 逐个分片拷贝。
  $pieces = $ossClient->generateMultiuploadParts($length, $part_size);
  $response_upload_part = array();
  $copyId = 1;
  $upload_position = 0;
  foreach ($pieces as $i => $piece) {
    $from_pos = $upload_position + (integer)$piece['seekTo'];
    $to_pos = (integer)$piece['length'] + $from_pos - 1;
    $up_options = array(
      'start' => $from_pos,
      'end' => $to_pos,
    );
    $response_upload_part[] = $ossClient->uploadPartCopy( $src_bucket, $src_object, $dst_bucket, $dst_o
bject, $copyId, $upload_id, $up_options);
    $copyId = $copyId + 1;
  }
  // 完成分片拷贝。
  $upload_parts = array();
  foreach ($response_upload_part as $i => $etag) {
    $upload_parts[] = array(
      'PartNumber' => ($i + 1),
      'ETag' => $etag,
    );
  }
  $result = $ossClient->completeMultipartUpload($dst_bucket, $dst_object, $upload_id, $upload_parts);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关拷贝大文件的详情，请参见UploadPartCopy。

3.8.10. 解冻文件

归档或冷归档类型的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档和冷归档文件。

注意 非归档或冷归档类型的文件，请勿调用RestoreObject方法。

归档及冷归档类型的详细说明请参见存储类型介绍。解冻文件的详情请参见RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->restoreObject($bucket, $object);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。

- 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
- 批量（Bulk）：表示5~12小时内完成解冻。

3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\RestoreConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object= "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 此处以设置解冻天数为5天，优先级为Standard为例。
$config = new RestoreConfig(5, "Standard");
$options = array(
  OssClient::OSS_RESTORE_CONFIG => $config
);
try {
  // 解冻文件。
  $ossClient->restoreObject($bucket, $object, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.8.11. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$symlink = "<yourSymlink>";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->putSymlink($bucket, $symlink, $object);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$symlink = "<yourSymlink>";
try {
   $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
   $Symlinks = $ossClient->getSymlink($bucket, $symlink);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
var_dump($Symlinks);
```

获取软链接的详细信息请参见GetSymlink。

3.8.12. 开启MD5校验

本文介绍如何开启MD5校验。

MD5校验用于确保数据传输的完整性。使用MD5校验时，性能会有所损失。上传文件时默认关闭MD5校验。

以下代码用于上传文件时开启MD5校验：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$options = array(OssClient::OSS_CHECK_MD5 => true);
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  $ossClient->uploadFile($bucket, $object, __FILE__, $options);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

putObject、uploadFile、appendObject、appendFile、multiuploadFile方法支持开启MD5校验。

3.9. 版本控制

3.9.1. 管理版本控制

版本控制应用于存储空间（Bucket） 内的所有文件（Object）。通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

Bucket的版本状态包括未开启版本控制（默认）、开启版本控制及暂停版本控制三种。有关版本控制的详情，请参见开发指南的版本控制介绍。

> **注意** 一旦Bucket开启了版本控制，将无法返回至未开启状态，但允许暂停版本控制。

设置Bucket版本控制状态

以下代码用于设置Bucket版本控制状态：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 设置存储空间版本控制为开启版本控制（Enabled）或暂停版本控制（Suspended）。
  $ossClient->putBucketVersioning($bucket, "Enabled");
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

设置Bucket版本控制状态的详情请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 获取Bucket版本控制状态信息。
  $status = $ossClient->getBucketVersioning($bucket);
  print($status);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取Bucket版本控制状态的详情请参见GetBucketVersioning。

列举Bucket中所有Object的版本信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  $maxKey = 100;
  $nextKeyMarker = '';
```

```
    $nextVersionIdMarker = '';
    while (true) {
        $options = array(
            'delimiter' => '',
            'key-marker' => $nextKeyMarker,
            'max-keys' => $maxKey,
            'version-id-marker' => $nextVersionIdMarker,
        );
        $result = $ossClient->listObjectVersions($bucket, $options);
        $nextKeyMarker = $result->getNextKeyMarker();
        $nextVersionIdMarker = $result->getNextVersionIdMarker();
        $objectList = $result->getObjectVersionList();
        $deleteMarkerList = $result->getDeleteMarkerList();
        // 打印Object版本信息。
        if (!empty($objectList)) {
            print("objectList:\n");
            foreach ($objectList as $objectInfo) {
                print($objectInfo->getKey() . ",");
                print($objectInfo->getVersionId() . ",");
                print($objectInfo->getLastModified() . ",");
                print($objectInfo->getETag() . ",");
                print($objectInfo->getSize() . ",");
                print($objectInfo->getIsLatest() . "\n");
            }
        }
        // 打印删除标记版本信息。
        if (!empty($deleteMarkerList)) {
            print("deleteMarkerList: \n");
            foreach ($deleteMarkerList as $deleteMarkerInfo) {
                print($deleteMarkerInfo->getKey() . ",");
                print($deleteMarkerInfo->getVersionId() . ",");
                print($deleteMarkerInfo->getLastModified() . ",");
                print($deleteMarkerInfo->getIsLatest() . "\n");
            }
        }
                if ($result->getIsTruncated() !== "true") {
                break;
        }
    }
} catch (OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息的详情请参见GetBucketVersions(ListObjectVersions)。

3.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS将为新上传的Object生成全局唯一的随机字符串版本ID，并在响应Header中通过 `x-oss-version-id` 形式返回。

在暂停了版本控制的Bucket中，OSS将为新上传的Object生成特殊字符串为“null”的版本ID。上传同名Object时，后一次会覆盖前一次上传的文件内容，即同一个Object仅有一个版本的ID为“null”。

以下代码用于简单上传：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
$content = "hello world";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 在受版本控制的Bucket中上传Object。
  $ret = $ossClient->putObject($bucket, $object, $content);
  // 查看Object的版本信息。
  print("versionId:" .$ret[OssClient::OSS_HEADER_VERSION_ID]);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中对Object进行追加上传（AppendObject）操作时，有如下注意事项：

- 仅支持对当前版本为Appendable类型的Object执行AppendObject操作。
- 不支持对历史版本为Appendable类型或当前版本为非Appendable类型的Object（包括Normal Object、Delete Marker等）执行AppendObject操作。
- 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
- 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable

类型的Object保留为历史版本，且该Object不允许继续追加。

以下代码用于追加上传：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
// 表示第一次、第二次以及第三次追加上传后获取的文件内容分别为Hello OSS、Hi OSS以及OSS OK。
$content_array = array('Hello OSS', 'Hi OSS', 'OSS OK');
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 第一次追加上传。第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
  $position = $ossClient->appendObject($bucket, $object, $content_array[0], 0);
  $position = $ossClient->appendObject($bucket, $object, $content_array[1], $position);
  $position = $ossClient->appendObject($bucket, $object, $content_array[2], $position);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应Header中以 `x-oss-version-id` 的形式返回。

以下代码用于分片上传：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Core\OssUtil;
```

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
// 填写本地文件的完整路径。
$uploadFile = "<yourLocalFile>";
/**
 * 步骤1：初始化一个分片上传事件，获取uploadId。
 */
try{
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
  // 返回uploadId。uploadId是分片上传事件的唯一标识。您可以根据uploadId发起相关的操作，如取消分片上传、查询分片上传等。
  $uploadId = $ossClient->initiateMultipartUpload($bucket, $object);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": initiateMultipartUpload FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": initiateMultipartUpload OK" . "\n");
/*
 * 步骤2：上传分片。
 */
$partSize = 10 * 1024 * 1024;
$uploadFileSize = sprintf('%u',filesize($uploadFile));
$pieces = $ossClient->generateMultiuploadParts($uploadFileSize, $partSize);
$responseUploadPart = array();
$uploadPosition = 0;
$isCheckMd5 = true;
foreach ($pieces as $i => $piece) {
  $fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
  $toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
  $upOptions = array(
    // 上传文件。
    $ossClient::OSS_FILE_UPLOAD => $uploadFile,
    // 设置分片号。
    $ossClient::OSS_PART_NUM => ($i + 1),
    // 指定分片上传起始位置。
    $ossClient::OSS_SEEK_TO => $fromPos,
    // 指定文件长度。
    $ossClient::OSS_LENGTH => $toPos - $fromPos + 1,
    // 是否开启MD5校验，true表示开启，false表示未开启。
    $ossClient::OSS_CHECK_MD5 => $isCheckMd5,
  );
  // 开启MD5校验。
  if ($isCheckMd5) {
    $contentMd5 = OssUtil::getMd5SumForFile($uploadFile, $fromPos, $toPos);
    $upOptions[$ossClient::OSS_CONTENT_MD5] = $contentMd5;
  }
  try {
    // 上传分片
```

```
        // 上传分片。
        $responseUploadPart[] = $ossClient->uploadPart($bucket, $object, $uploadId, $upOptions);
    } catch(OssException $e) {
        printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }
    printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPart - part#{$i} OK\n");
}
// $uploadParts是由每个分片的ETag和分片号（PartNumber）组成的数组。
$uploadParts = array();
foreach ($responseUploadPart as $i => $eTag) {
    $uploadParts[] = array(
        'PartNumber' => ($i + 1),
        'ETag' => $eTag,
    );
}
/**
 * 步骤3：完成上传。
 */
try {
    // 执行completeMultipartUpload操作时，需要提供所有有效的$uploadParts。OSS收到提交的$uploadParts后，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
    $ret = $ossClient->completeMultipartUpload($bucket, $object, $uploadId, $uploadParts);
    // 查看Object版本信息。
    print("versionId:" .$ret[OssClient::OSS_HEADER_VERSION_ID]);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": completeMultipartUpload FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

分片上传的详细信息请参见InitiateMultipartUpload、UploadPart、CompleteMultipartUpload等接口。

3.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
// 指定Object的版本ID。
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 下载指定版本ID的Object。
  $content = $ossClient->getObject($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
  print($content);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

下载文件的详细信息请参见Get Object。

3.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从源Bucket复制到同一地域的目标Bucket。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定版本的Object，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝的Object自动生成唯一的版本ID，此版本ID将会在响应Header的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为”null“的版本，且会覆盖原versionId为”null“的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
$versionId = "<yourObjectVersionId>";
$to_bucket = $bucket;
$to_object = $object . '.copy';
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 拷贝指定版本的Object。
  $ret = $ossClient->copyObject($bucket, $object, $to_bucket, $to_object, array(OssClient::OSS_VERSION_ID => $versionId));
  print("versionId:" .$ret['x-oss-version-id']);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在 `x-oss-copy-source` 的请求Header中附带versionId的子条件，实现从Object的指定版本进行拷贝，例如 `x-oss-copy-source:/SourceBucketName/SourceObjectName?versionId=111111` 。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
<?php
```

```
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$versionId = "<yourObjectVersionId>";
$to_bucket = $bucket;
$to_object = $object . '.copy';
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
/**
 * 步骤1：初始化一个分片上传事件，获取uploadId。
 */
try {
  // 返回uploadId。uploadId是分片上传事件的唯一标识，您可以根据uploadId发起相关的操作，如取消分片上传、
查询分片上传等。
  $uploadId = $ossClient->initiateMultipartUpload($to_bucket, $to_object);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": initiateMultipartUpload FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": initiateMultipartUpload OK" . "\n");
/*
 * 步骤2：上传分片。
 */
$partSize = 10 * 1024 * 1024;
try {
  $meta = $ossClient->getSimplifiedObjectMeta($bucket, $object);
  $uploadSize = $meta[strtolower('Content-Length')];
} catch(OssException $e) {
  printf(__FUNCTION__ . ": getSimplifiedObjectMeta FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
$pieces = $ossClient->generateMultiuploadParts($uploadSize, $partSize);
$responseUploadPart = array();
$uploadPosition = 0;
foreach ($pieces as $i => $piece) {
  $fromPos = $uploadPosition + (integer)$piece[$ossClient::OSS_SEEK_TO];
  $toPos = (integer)$piece[$ossClient::OSS_LENGTH] + $fromPos - 1;
  $upOptions = array(
    // 指定源Object的起始位置。
    'start' => $fromPos,
```

```
        // 指定源Object的结束位置。
        'end' => $toPos,
        // 指定源Object的版本ID。
        OssClient::OSS_VERSION_ID => $versionId
    );
    try {
        // 上传分片。
        $responseUploadPart[] = $ossClient->uploadPartCopy($bucket, $object, $to_bucket, $to_object, ($i + 1),
$uploadId, $upOptions);
    } catch(OssException $e) {
        printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPartCopy - part#{$i} FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }
    printf(__FUNCTION__ . ": initiateMultipartUpload, uploadPartCopy - part#{$i} OK\n");
}
// $uploadParts是由每个分片的ETag和分片号（PartNumber）组成的数组。
$uploadParts = array();
foreach ($responseUploadPart as $i => $eTag) {
    $uploadParts[] = array(
        'PartNumber' => ($i + 1),
        'ETag' => $eTag,
    );
}
/**
 * 步骤3：完成分片上传。
 */
try {
    // 执行completeMultipartUpload操作时，需要提供所有有效的$uploadParts。OSS收到提交的$uploadParts后
，会逐一验证每个分片的有效性。当所有的数据分片验证通过后，OSS将把这些分片组合成一个完整的文件。
    $ret = $ossClient->completeMultipartUpload($to_bucket, $to_object, $uploadId, $uploadParts);
    // 查看Object的版本信息。
    print("versionId:" .$ret[OssClient::OSS_HEADER_VERSION_ID]);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": completeMultipartUpload FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

分片拷贝的详细信息请参见UploadPartCopy。

3.9.5. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

- 未指定versionId（临时删除）：

如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS会将“null”字符串当成"null"的versionId，从而删除versionId为"null"的Object。

删除单个文件

以下提供了永久删除和临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object进行永久删除：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 删除指定versionId的Object。
  $ossClient->deleteObject($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

- 临时删除

  以下代码用于不指定versionId对Object进行临时删除：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 不指定versionId对Object进行临时删除，此操作会为Object添加删除标记。
  $ret = $ossClient->deleteObject($bucket, $object);
  print("delete marker:" .$ret['x-oss-delete-marker'] ."\n");
  print("version id:" .$ret['x-oss-version-id']);
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于指定versionId对多个Object进行永久删除：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\DeleteObjectInfo;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 删除多个指定versionId的Object。
  $objects = array();
  $objects[] = new DeleteObjectInfo("<yourObject1Name>", "<object1VersionId>");
  $objects[] = new DeleteObjectInfo("<yourObject2Name>", "<object2VersionId>");
  $deletedObjectList = $ossClient->deleteObjectVersions($bucket, $objects);
  // 查看删除结果。
  if (!empty($deletedObjectList)) {
      print("deletedObjectList:\n");
      foreach ($deletedObjectList as $deletedObjectInfo) {
        print($deletedObjectInfo->getKey() . ",");
        print($deletedObjectInfo->getVersionId() . ",");
        print($deletedObjectInfo->getDeleteMarker() . ",");
        print($deletedObjectInfo->getDeleteMarkerVersionId() . "\n");
      }
  }
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\DeleteObjectInfo;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
  // 不指定versionId删除多个Object，此操作会为Object添加删除标记。
    $objects = array();
  $objects[] = new DeleteObjectInfo("<yourObject1Name>");
  $objects[] = new DeleteObjectInfo("<yourObject2Name>");
  $deletedObjectList = $ossClient->deleteObjectVersions($bucket, $objects);
  // 查看删除结果。
  if (!empty($deletedObjectList)) {
      print("deletedObjectList:\n");
      foreach ($deletedObjectList as $deletedObjectInfo) {
        print($deletedObjectInfo->getKey() . ",");
        print($deletedObjectInfo->getVersionId() . ",");
        print($deletedObjectInfo->getDeleteMarker() . ",");
        print($deletedObjectInfo->getDeleteMarkerVersionId() . "\n");
      }
  }
} catch(OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

3.9.6. 解冻文件

在受版本控制的存储空间（Bucket）中，文件（Object）的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\DeleteObjectInfo;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object= "<yourObjectName>";
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try{
   // 解冻指定版本的Object。
   $result = $ossClient->restoreObject($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
} catch(OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

解冻文件的详细信息请参见RestoreObject。

3.9.7. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的元信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的元信息。

以下代码用于获取文件元信息：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 获取指定版本Object的全部元信息。
  $objectMeta = $ossClient->getObjectMeta($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
  print_r($objectMeta);
  // 获取指定版本Object的部分元信息。
  $objectMeta = $ossClient->getSimplifiedObjectMeta($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
  print_r($objectMeta);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取文件元信息的详情请参见GetObjectMeta。

3.9.8. 管理文件访问权限

OSS提供了公共读写、公共读、私有读写以及默认权限共四种文件（Object）级别的访问权限（ACL）。本文介绍如何在受版本控制的存储空间（Bucket）中管理Object ACL。

Object ACL说明

Object ACL的四种访问权限含义如下：

权限值	中文名称	权限控制
public-read-write	公共读写	该ACL表明某个Object是公共读写资源，即任何人（包括匿名访问）拥有该Object的读写权限。

权限值	中文名称	权限控制
public-read	公共读，私有写	该ACL表明某个Object是公共读资源，即非Object Owner只有该Object的读权限，而Object Owner拥有该Object的读写权限。
private	私有读写	该ACL表明某个Object是私有资源，即Object Owner拥有该Object的读写权限，其他人在未经授权的情况下无法访问该Object。
default	默认权限	该ACL表明某个Object遵循Bucket的读写权限。

Object ACL优先于Object所属Bucket的ACL。例如Bucket ACL设置为（public-read），Object ACL设置为private，则只有Object Owner拥有该Object的读写权限，其他人在未经授权的情况下无法访问该Object。

设置Object ACL

调用 `putObjectAcl` 方法默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置Object ACL:

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 填写Bucket名称。
$bucket= "<yourBucketName>";
// 填写Object名称，即不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
// 指定Object的版本Id。
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 设置指定版本Object的访问权限为公共读。
  $ossClient->putObjectAcl($bucket, $object, 'public-read', array(OssClient::OSS_VERSION_ID => $versionId)
);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关设置Object ACL的更多信息，请参见Put Object ACL。

获取Object ACL

调用 `getObjectAcl` 方法默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取Object ACL:

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 填写Bucket名称。
$bucket= "<yourBucketName>";
// 填写Object名称，即不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
// 指定Object的版本Id。
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 获取文件的访问权限。
  $acl = $ossClient->getObjectAcl($bucket, $object, array(OssClient::OSS_VERSION_ID => $versionId));
  printf($acl . "\n");
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取Object ACL的更多信息，请参见Get Object ACL。

3.9.9. 管理软链接

软链接类似Windows快捷方式。通过软链接，您可以快速访问常用的目标文件（Object）。在受版本控制的存储空间（Bucket）中，不同版本的软链接可以指向不同的目标文件。

创建软链接

在受版本控制的Bucket中创建软链接时，软链接默认指向Object的当前版本。不允许对删除标记（Delete Marker）创建软链接。

以下代码用于创建软链接：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 指定Bucket名称。
$bucket= "<yourBucketName>";
// 指定Object名称，即不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
// 指定软链接名称。
$symLink = "<yourSymLink>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 创建软链接。
  $sym = $ossClient->putSymlink($bucket, $symlink, $object);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关创建软链接的更多信息，请参见PutSymlink。

获取软链接

调用 `getSymlink` 方法默认获取软链接的当前版本，允许通过指定versionId来获取指定版本的软链接。如果软链接的当前版本为删除标记，OSS会返回 `404 Not Found` ，在响应Header中返回 `x-oss-delete-marker = true` 以及 `x-oss-version-id` 的版本ID。

以下代码用于获取软链接：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 指定Bucket名称。
$bucket= "<yourBucketName>";
// 指定Object名称，即不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
$object = "<yourObjectName>";
// 指定Object的版本Id。
$versionId = "<yourObjectVersionId>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
try {
  // 获取指定版本软链接的内容。
  $sym = $ossClient->getSymlink($bucket, $symlink, array(OssClient::OSS_VERSION_ID => $versionId));
  printf($sym['x-oss-version-id'] . "\n");
  printf($sym[OssClient::OSS_SYMLINK_TARGET] . "\n");
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取软链接的更多信息，请参见GetSymlink。

3.10. 对象标签

3.10.1. 设置对象标签

OSS支持使用对象标签（Object Tagging）对存储的Object进行分类，您可以针对同标签的Object设置生命周期规则、访问权限等。

注意事项

对象标签使用一组键值对（Key-Value）标记对象。使用对象标签时，有如下注意事项：

- 只有Bucket拥有者和授权的RAM用户拥有读写对象标签的权限，该权限不受对象ACL控制。
- 您可以在上传Object时添加对象标签，也可以对已上传Object添加或更改对象标签。

  有关设置对象标签的详情，请参见PutObjectTagging。
- 单个Object最多可设置10个标签，Key不可重复。

- 每个Key长度不超过128字节，每个Value长度不超过256字节。
- Key和Value区分大小写。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/
- 更改标签时不会更新Object的Last-Modified时间。

有关对象标签的详情，请参见对象标签。

上传Object时添加对象标签

以下分别介绍如何在简单上传、分片上传以及追加上传场景中添加对象标签的示例。

- 简单上传时添加对象标签

  以下代码用于简单上传时添加对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// yourObjectName表示即将上传的Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，如填写为example/test.jpg。
$object= "<yourObjectName>";
$content= "hello world";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 设置对象标签。
$options = array(
    OssClient::OSS_HEADERS => array(
        'x-oss-tagging' => 'key1=value1&key2=value2&key3=value3',
));
try {
    // 通过简单上传的方式上传Object。
    $ossClient->putObject($bucket, $object, $content, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

- 分片上传时添加对象标签

  以下代码用于分片上传时添加对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object= "<yourObjectName>";
$file = "<yourFilePath>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 设置对象标签。
$options = array(
  OssClient::OSS_CHECK_MD5 => true,
  OssClient::OSS_PART_SIZE => 1,
  OssClient::OSS_HEADERS => array(
      'x-oss-tagging' => 'key1=value1&key2=value2&key3=value3',
  ),
);
try {
  // 通过分片上传的方式上传Object。
  $ossClient->multiuploadFile($bucket, $object, $file, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

- 追加上传时添加对象标签

  以下代码用于追加上传时添加对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object= "<yourObjectName>";
$content_array = array('Hello OSS', 'Hi OSS');
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 设置对象标签。
$options = array(
    OssClient::OSS_HEADERS => array(
        'x-oss-tagging' => 'key1=value1&key2=value2',
));
try {
  // 通过追加上传的方式上传Object。
  $position = $ossClient->appendObject($bucket, $object, $content_array[0], 0, $options);
  $position = $ossClient->appendObject($bucket, $object, $content_array[1], $position);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

对已上传Object添加或更改对象标签

以下代码用于对已上传Object添加或更改对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\TaggingConfig;
use OSS\Model\Tag;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// yourObjectName表示已上传Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，如填写为example
/test.jpg。
$object= "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 设置对象标签。
$config = new TaggingConfig();
$config->addTag(new Tag("key1", "value1"));
$config->addTag(new Tag("key2", "value2"));
try {
   $ossClient->putObjectTagging($bucket, $object, $config);
} catch (OssException $e) {
   printf(__FUNCTION__ . ": FAILED\n");
   printf($e->getMessage() . "\n");
   return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

3.10.2. 获取对象标签

OSS支持使用对象标签（Object Tagging）对存储的Object进行分类。本文介绍如何获取对象标签。

对象标签使用一组键值对（Key-Value）来标记对象。有关对象标签的详情，请参见开发指南的对象标签。

以下代码用于获取对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，例如example/test.jpg。
$object= "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取对象标签。
  $config = $ossClient->getObjectTagging($bucket, $object);
  print_r($config);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

获取对象标签详情请参考GetObjectTagging。

3.10.3. 删除对象标签

OSS支持使用对象标签（Object Tagging）对存储的Object进行分类。本文介绍如何删除对象标签。

对象标签使用一组键值对（Key-Value）来标记对象。有关对象标签的详情，请参见开发指南的对象标签。

以下代码用于删除对象标签：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
// 填写Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，例如example/test.jpg。
$object= "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 删除对象标签。
  $ossClient->deleteObjectTagging($bucket, $object);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

删除对象标签的详情，请参见DeleteObjectTagging。

3.11. 单链接限速

本文介绍如何在上传和下载文件（Object）时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

> **说明** 有关单链接限速的使用场景及注意事项的更多信息，请参见开发指南的单链接限速文档。

普通上传和下载限速

以下代码用于通过PutObject方式普通上传和通过GetObject方式下载文件时设置单链接限速：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object= "<yourObjectName>";
$content = "hello world";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 限速100 KB/s，即819200 bit/s。
$options = array(
    OssClient::OSS_HEADERS => array(
        OssClient::OSS_TRAFFIC_LIMIT => 819200,
));
try {
  // 限速上传。
  $ossClient->putObject($bucket, $object, $content, $options);
  // 限速下载。
  $ossClient->getObject($bucket, $object, $options);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

签名URL方式上传和下载限速

以下代码用于使用签名URL方式上传和下载文件时设置单链接限速：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object= "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 限速100 KB/s，即819200 bit/s。
$options = array(
    OssClient::OSS_TRAFFIC_LIMIT => 819200,
);
// 创建限速上传的URL，有效期为60s。
$timeout = 60;
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "PUT", $options);
print($signedUrl);
// 创建限速下载的URL，有效期为120s。
$timeout = 120;
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "GET", $options);
print($signedUrl);
```

3.12. 数据加密

3.12.1. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来。下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

加密方式

OSS提供了以下两种服务器端加密方式：

- 使用KMS托管密钥进行加密（SSE-KMS）

  上传文件（Object）时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

  使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参考KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

说明

- 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
- 如果配置了服务器端加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。

更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

Bucket默认加密方式设置成功后，所有上传至该Bucket但未设置加密方式的Object，均使用Bucket默认加密方式进行加密。

以下代码用于设置Bucket默认加密方式：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\ServerSideEncryptionConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 将Bucket默认的服务器端加密方式设置为OSS完全托管加密（SSE-OSS）。
  $config = new ServerSideEncryptionConfig("AES256");
  $ossClient->putBucketEncryption($bucket, $config);
  // 将Bucket默认的服务器端加密方式设置为KMS，且不指定CMK ID。
  $config = new ServerSideEncryptionConfig("KMS");
  $ossClient->putBucketEncryption($bucket, $config);
  // 将Bucket默认的服务器端加密方式设置为KMS，且指定了CMK ID。
  $config = new ServerSideEncryptionConfig("KMS", "your kms id");
  $ossClient->putBucketEncryption($bucket, $config);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关配置Bucket加密详情，请参见PutBucketEncryption。

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Model\ServerSideEncryptionConfig;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 获取Bucket加密配置。
  $config = $ossClient->getBucketEncryption($bucket);
  // 打印Bucket加密配置。
  print($config->getSSEAlgorithm());
  print($config->getKMSMasterKeyID());
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关获取Bucket加密配置详情，请参见GetBucketEncryption。

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
try {
  // 删除Bucket加密配置。
  $ossClient->deleteBucketEncryption($bucket);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

有关删除Bucket加密配置详情，请参见DeleteBucketEncryption。

3.13. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

以下代码用于使用STS临时授权上传文件。

说明 您可以通过调用STS服务AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
$securityToken = "yourSecurityToken";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
// 填写不包含Bucket名称在内的Object完整路径。
$object = "exampleobject.txt";
// 填写上传的字符串。
$content = "Hello OSS";
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
  // 使用STS临时授权上传文件。
  $ossClient->putObject($bucket, $object, $content);
} catch (OssException $e) {
  print $e->getMessage();
}
```

使用签名URL进行临时授权

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

授权访问的完整代码请参见GitHub。

- 生成上传的签名URL

  以下代码用于生成上传（PutObject）的签名URL，并使用生成的签名URL上传文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Http\RequestCore;
use OSS\Http\ResponseCore;
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
$securityToken = "yourSecurityToken";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
// 填写不包含Bucket名称在内的Object完整路径。
$object = "exampleobject.txt";
// 设置签名URL的有效时长为3600秒。
$timeout = 3600;
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
  // 生成签名URL。
  $signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "PUT");
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": signedUrl: " . $signedUrl . "\n");
// 使用签名URL上传文件。
// 填写上传的字符串。
$content = "Hello OSS";
$request = new RequestCore($signedUrl);
// 生成的签名URL以PUT的方式访问。
$request->set_method('PUT');
$request->add_header('Content-Type', '');
$request->add_header('Content-Length', strlen($content));
$request->set_body($content);
$request->send_request();
$res = new ResponseCore($request->get_response_header(),
  $request->get_response_body(), $request->get_response_code());
if ($res->isOK()) {
  print(__FUNCTION__ . ": OK" . "\n");
} else {
  print(__FUNCTION__ . ": FAILED" . "\n");
};
```

- 生成下载的签名URL

以下代码用于生成下载文件（GetObject）的签名URL，并使用生成的签名URL下载文件。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
use OSS\Core\OssException;
use OSS\Http\RequestCore;
use OSS\Http\ResponseCore;
// 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
$accessKeyId = "yourAccessKeyId";
$accessKeySecret = "yourAccessKeySecret";
// 从STS服务获取的安全令牌（SecurityToken）。
$securityToken = "yourSecurityToken";
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
$endpoint = "yourEndpoint";
// 填写Bucket名称。
$bucket= "examplebucket";
// 填写不包含Bucket名称在内的Object完整路径。
$object = "exampleobject.txt";
// 设置签名URL的有效时长为3600秒。
$timeout = 3600;
try {
  $ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false, $securityToken);
  // 生成GetObject的签名URL。
  $signedUrl = $ossClient->signUrl($bucket, $object, $timeout);
} catch (OssException $e) {
  printf(__FUNCTION__ . ": FAILED\n");
  printf($e->getMessage() . "\n");
  return;
}
print(__FUNCTION__ . ": signedUrl: " . $signedUrl . "\n");
// 您可以使用代码来访问签名URL，也可以输入到浏览器地址栏中进行访问。
$request = new RequestCore($signedUrl);
// 生成的签名URL默认以GET方式访问。
$request->set_method('GET');
$request->add_header('Content-Type', '');
$request->send_request();
$res = new ResponseCore($request->get_response_header(), $request->get_response_body(), $request-
>get_response_code());
if ($res->isOK()) {
  print(__FUNCTION__ . ": OK" . "\n");
} else {
  print(__FUNCTION__ . ": FAILED" . "\n");
};
```

3.14. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的完整代码请参见Git Hub。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
// 指定处理后的图片名称。
$download_file = "<LocalFileName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
// $ossClient->uploadFile($bucket, $object, "<yourLocalFile>");
// 将图片缩放为固定宽高100 px，并保存在本地。
$options = array(
  OssClient::OSS_FILE_DOWNLOAD => $download_file,
  OssClient::OSS_PROCESS => "image/resize,m_fixed,h_100,w_100" );
$ossClient->getObject($bucket, $object, $options);
// 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
// $ossClient->deleteObject($bucket, $object);
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
// 指定处理后的图片名称。
$download_file = "<LocalFileName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
// $ossClient->uploadFile($bucket, $object, "<yourLocalFile>");
// 将图片缩放为固定宽高100 px后，再旋转90°，之后保存到本地。
$style = "image/resize,m_fixed,w_100,h_100/rotate,90";
$options = array(
  OssClient::OSS_FILE_DOWNLOAD => $download_file,
  OssClient::OSS_PROCESS => $style);
$ossClient->getObject($bucket, $object, $options);
// 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
// $ossClient->deleteObject($bucket, $object);
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket = "<yourBucketName>";
$object = "<yourObjectName>";
// 指定处理后的图片名称。
$download_file = "<LocalFileName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
// $ossClient->uploadFile($bucket, $object, "<yourLocalFile>");
// 指定图片样式。
$style = "style/<yourCustomStyleName>";
// 将图片保存在本地。
$options = array(
  OssClient::OSS_FILE_DOWNLOAD => $download_file,
  OssClient::OSS_PROCESS => $style);
$ossClient->getObject($bucket, $object, $options);
// 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
// $ossClient->deleteObject($bucket, $object);
```

图片处理持久化

您可以通过ImgSaveAs接口将图片保存至原图片所在存储空间。以下代码展示了图片处理持久化操作：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
   require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
   require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
// 指定处理后的图片名称。
$save_object = "<saveObjectName>";
function base64url_encode($data)
{
   return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
}
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint, false);
// 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
// $ossClient->uploadFile($bucket, $object, "<yourLocalFile>");
// 将图片缩放为固定宽高100 px后，再旋转90°。
$style = "image/resize,m_fixed,w_100,h_100/rotate,90";
// 将处理后的图片转存到当前Bucket。
$process = $style.
        '|sys/saveas'.
        ',o_'.base64url_encode($save_object).
        ',b_'.base64url_encode($bucket);
$result = $ossClient->processObject($bucket, $object, $process);
// 打印处理结果。
print($result);
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
<?php
if (is_file(__DIR__ . '/../autoload.php')) {
  require_once __DIR__ . '/../autoload.php';
}
if (is_file(__DIR__ . '/../vendor/autoload.php')) {
  require_once __DIR__ . '/../vendor/autoload.php';
}
use OSS\OssClient;
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
$accessKeyId = "<yourAccessKeyId>";
$accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
$endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
$bucket= "<yourBucketName>";
$object = "<yourObjectName>";
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $endpoint);
// 生成一个带图片处理参数的签名的URL，有效期是3600秒，可以直接使用浏览器访问。
$timeout = 3600;
$options = array(
  OssClient::OSS_PROCESS => "image/resize,m_lfit,h_100,w_100" );
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout, "GET", $options);
print("rtmp url: \n" . $signedUrl);
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

3.15. 异常处理

OSS PHP SDK异常（OssException）包括参数无效、文件不存在等错误。您可以通过getMessage方法获取错误信息。

OssException的详细信息请参见GitHub 。

异常处理示例

以下代码展示了创建一个已存在的存储空间时的异常处理，并打印出错误信息（Message）。

```
  try {
    $ossClient->createBucket($bucket);
  } catch (OssException $e) {
 print("Exception:" . $e->getMessage() . "\n");
  }
```

您还可以获取以下信息：

参数	说明
HTTPStatus	HTTP状态码，通过方法getHTTPStatus获取。
ErrorCode	OSS返回的错误码，通过方法getErrorCode获取。

参数	说明
ErrorMessage	OSS返回的错误信息，通过方法getErrorMessage获取。
RequestId	用于唯一标识该请求的UUID。当您无法解决问题时，可以提供RequestId来请求OSS开发工程师的帮助。通过方法getRequestId获取。
Details	OSS返回的错误信息描述。通过方法getDetails获取。

OSS常见错误码

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	存储空间已经存在	409
BucketNotEmpty	存储空间不为空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
FilePartNotExist	文件分片不存在	400
FilePartStale	文件分片过时	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的存储空间名称	400
InvalidDigest	无效的摘要	400
InvalidObjectName	无效的文件名称	400
InvalidPart	无效的分片	400
InvalidPartOrder	无效的分片顺序	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411

错误码	描述	HTTP状态码
MissingContentLength	缺少内容长度	411
NoSuchBucket	存储空间不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	分片上传ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403
RequestTimeout	请求超时	400
SignatureDoesNotMatch	签名错误	403
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

4.Node.js

4.1. 安装

本文主要介绍如何安装和使用Node.js SDK。

前提条件

- 已开通阿里云对象存储OSS服务。
- 已创建RAM用户的AccessKey ID和AccessKey Secret。

  由于云账号AccessKey拥有所有API的访问权限，建议使用RAM用户的AccessKey。如果部署在服务端，请使用RAM或STS的方式进行API访问或日常运维管控操作；如果部署在客户端，请使用STS方式进行API访问。详情请参见访问控制

背景信息

OSS Node.js SDK基于Node.js环境构建。

目前官网文档中的demo基于SDK 6.X，SDK版本低于6.X，请参见5.X开发文档，如需升级至6.X，请参见升级详情。

SDK源码请参见Github，更多信息请参见API文档。

安装

OSS支持8.0.0及以上的Node.js版本。如果需要在Node.js低于8.0.0的环境中使用，请使用ali-oss 4.x版本。

使用npm安装SDK开发包，安装命令为**npm install ali-oss --save**。

如果使用npm遇到网络问题，建议使用淘宝提供的npm镜像cnpm。

使用方式

Node.js SDK支持同步和异步的使用方式。无论同步方式还是异步方式，均使用 `new OSS()` 创建client。

- 同步方式：基于 `async` 和 `await` 方式，异步编程同步化。
- 异步方式：类似Callback的方式，API接口返回Promise，使用 `.then()` 处理返回结果，使用 `.catch()` 处理错误。

先使用同步方式上传文件，再使用异步方式下载文件的示例代码如下：

- 同步方式

```
let client = new OSS(...);
async function put () {
  try {
    // 'object'填写上传至OSS的object名称,即不包括Bucket名称在内的Object的完整路径。
    // 'localfile'填写上传至OSS的本地文件完整路径。
    let r1 = await client.put('object','localfile');
    console.log('put success: %j', r1);
    let r2 = await client.get('object');
    console.log('get success: %j', r2);
  } catch(e) {
    console.error('error: %j', e);
  }
}
await put();
```

- 异步方式

```
let client = new OSS(...);
// 'object'填写从OSS下载的object名称，即不包括Bucket名称在内的Object的完整路径。
// 'localfile'填写下载到本地文件的完整路径。
client.put('object', 'localfile').then(function (r1) {
  console.log('put success: %j', r1);
  return client.get('object');
}).then(function (r2) {
  console.log('get success: %j', r2);
}).catch(function (err) {
  console.error('error: %j', e);
});
```

4.2. 初始化

本文介绍如何初始化Node.js SDK。

创建一个 `app.js` 文件并写入如下内容：

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
```

其中 `region` 是指您申请OSS服务时的地域，例如 `oss-cn-hangzhou` 。完整的地域列表请参见访问域名和数据中心。

说明 如果使用的endpoint不在上述列表中，可以通过以下参数指定endpoint：

- internal：配合 `region` 使用。如果指定 `internal` 为 `true` ，则访问内网节点。
- secure：配合 `region` 使用。如果指定 `secure` 为 `true` ，则使用HTTPS访问。
- endpoint：如果指定了 `endpoint` ，如 `http://oss-cn-hangzhou.aliyuncs.com` ，则 `region` 会被忽略， `endpoint` 可以指定为HTTPS，也可以是IP的形式。
- cname：配合 `endpoint` 使用。如果指定 `cname` 为 `true` ，则将 `endpoint` 视为用户绑定的自定义域名。
- bucket：如果未指定 `bucket` ，则进行object相关的操作时需要先调用 `useBucket` 接口（仅需要调用一次）。
- timeout：指定访问OSS的API的超时时间，默认值为60000，单位为毫秒。

4.3. 配置项

本文介绍初始化时如何使用配置项。

OSS（options）中的各个配置项说明请参见下表。

配置项	类型	描述
accessKeyId	String	通过阿里云控制台创建的AccessKey ID。
accessKeySecret]	String	通过阿里云控制台创建的AccessKey Secret。
stsToken	String	使用临时授权方式。更多信息，请参见使用STS进行临时授权。
bucket	String	通过控制台或PutBucket创建的Bucket。
endpoint	String	OSS域名。
region	String	Bucket所在的区域， 默认值为oss-cn-hangzhou。
internal	Boolean	是否使用阿里云内网访问，默认值为false。例如通过ECS访问OSS，则设置internal为true，采用internal的endpoint可节省费用。
cname	Boolean	是否支持上传自定义域名，默认值为false。如果设置cname为true，则endpoint传入自定义域名时，自定义域名需要先和Bucket进行绑定。
isRequestPay	Boolean	Bucket是否开启请求者付费模式，默认值为false。更多信息，请参见请求者付费模式。
secure	Boolean	设置secure为true，则使用HTTPS；设置secure为false，则使用HTTP。更多信息，请参见常见问题。
timeout	String|Number	超时时间，默认值为60000，单位为毫秒。

Node.js示例如下：

```
var oss = require('ali-oss');
var client = oss({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 填写Bucket名称。
  bucket: 'examplebucket'
});
```

4.4. 快速入门

本文介绍如何在Node.js环境中快速使用OSS服务，包括查看存储空间（Bucket）列表、上传文件（Object）等。

> **说明** 为了方便修改，本文会新建一个 `app.js` 文件。以下将以同步的方式说明各个操作的示例代码。示例代码中client的生成请参见初始化。

查看Bucket列表

在 `app.js` 末尾添加如下内容，使用listBuckets接口查看Bucket列表。

```
async function listBuckets () {
  try {
    let result = await client.listBuckets();
  } catch(err) {
    console.log(err)
  }
}
listBuckets();
```

您可以使用 `node app.js` 运行并查看结果。

关于Bucket接口的更多信息，请参见Git Hub。

查看文件列表

修改 `app.js` ，使用list接口查看文件列表。

```
client.useBucket('examplebucket');
async function list () {
 try {
  let result = await client.list({
   'max-keys': 5
  })
  console.log(result)
 } catch (err) {
  console.log (err)
 }
}
list();
```

您可以使用 `node app.js` 运行并查看结果。

上传文件

修改 `app.js` ，使用put接口上传单个文件。

```
client.useBucket('examplebucket');
async function put () {
 try {
  let result = await client.put('exampleobject.txt', 'D:\\localpath\\examplefile.txt');
  console.log(result);
 } catch (err) {
  console.log (err);
 }
}
put();
```

下载文件

修改 `app.js` ，使用get接口下载单个文件。

```
client.useBucket('examplebucket');
async function get () {
 try {
  let result = await client.get('exampleobject.txt');
  console.log(result);
 } catch (err) {
  console.log (err);
 }
}
get();
```

删除文件

修改 `app.js` ，使用delete接口删除单个文件。

```
client.useBucket('examplebucket');
async function delete () {
 try {
  let result = await client.delete('exampleobject.txt');
  console.log(result);
 } catch (err) {
  console.log (err);
 }
}
delete();
```

关于Object接口的更多信息，请参见Git Hub。

4.5. 存储空间

4.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

以下代码用于创建examplebucket存储空间。

您可以在创建存储空间时指定存储类型和存储空间读写权限。更多信息，请分别参见存储类型介绍和设置存储空间读写权限（ACL）。

```
let OSS = require('ali-oss');
let client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourregion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret'
});
// 创建存储空间。
async function putBucket() {
 try {
  const options = {
   storageClass: 'Standard', // 存储空间的默认存储类型为标准存储，即Standard。如果需要设置存储空间的存储类型为归档存储，请替换为Archive。
   acl: 'private', // 存储空间的默认读写权限为私有，即private。如果需要设置存储空间的读写权限为公共读，请替换为public-read。
   dataRedundancyType: 'LRS' // 存储空间的默认数据容灾类型为本地冗余存储，即LRS。如果需要设置数据容灾类型为同城冗余存储，请替换为ZRS。
  }
  // 填写Bucket名称。
  const result = await client.putBucket('examplebucket', options);
  console.log(result);
 } catch (err) {
  console.log(err);
 }
}
putBucket();
```

关于创建存储空间的更多信息，请参见PutBucket。

4.5.2. 列举存储空间

存储空间（Bucket）是存储对象（Object）的容器。您可以列举所有的存储空间，或符合指定条件的存储空间。

列举所有的存储空间

以下代码用于列举所有存储空间。

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
async function listBuckets() {
 try {
  const result = await client.listBuckets();
  console.log(result);
 } catch (err) {
  console.log(err);
 }
}
listBuckets();
```

列举指定前缀的存储空间

以下代码用于列举以example为前缀（prefix）的存储空间。

```
const OSS = require('ali-oss');
const client = new OSS({
 region: '<Your region>',
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
async function listBuckets() {
 try {
  const result = await client.listBuckets({
   prefix: 'prefix' // 指定需要列举的存储空间的前缀。
  });
  console.log(result);
 } catch (err) {
  console.log(err);
 }
}
listBuckets();
```

列举指定marker之后的存储空间

以下代码用于列举examplebucket之后的存储空间。

```
const OSS = require('ali-oss');
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function listBuckets() {
  try {
    const result = await client.listBuckets({
      marker: 'marker' // 列举指定marker之后的存储空间。
    });
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}
listBuckets();
```

列举指定个数的存储空间

以下代码用于最多列举500个存储空间。

```
const OSS = require('ali-oss');
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function listBuckets() {
  try {
    const result = await client.listBuckets({
      'max-keys': 20 // 限定此次列举存储空间的个数为20个。
    });
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}
listBuckets();
```

4.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

以下代码用于判断存储空间examplebucket是否存在。

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>'
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
async function bucketisExist() {
 try {
  const result = await client.getBucketInfo('<Your bucket>')
  console.log('bucketInfo: ', result.bucket)
 } catch (error) {
  // 指定的存储空间不存在。
  if (error.name === 'NoSuchBucketError') {
   console.log
  } else {
   console.log(error)
  }
 }
}
bucketisExist()
```

4.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>'
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>'
});
async function getLocation() {
 const result = await client.getBucketInfo
 console.log(result.bucket.Location)
}
getLocation()
```

有关地域的详细信息，请参见开发指南基本概念中的地域介绍以及GetBucketLocation API。

4.5.5. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、存储类型、版本控制状态等。

```
const OSS = require('ali-oss')
const client = new OSS({
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your region>'
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketInfo() {
  const result = await client.getBucketInfo('<Your bucket>')
  // 获取存储空间所在的地域。
  console.log(result.bucket.Location)
  // 获取存储空间的名称。
  console.log(result.bucket.Name)
  // 获取存储空间的拥有者ID。
  console.log(result.bucket.Owner.ID)
  // 获取存储空间的拥有者名称，目前和拥有者ID一致。
  console.log(result.bucket.Owner.DisplayName)
  // 获取存储空间的创建时间。
  console.log(result.bucket.CreationDate)
  // 获取存储空间的存储类型。
  console.log(result.bucket.StorageClass)
  // 获取存储空间的版本控制状态。
  console.log(result.bucket.Versioning)
}
getBucketInfo()
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

4.5.6. 管理存储空间访问权限

您可以在创建存储空间（Bucket）时设置存储空间的访问权限（ACL），也可以在创建存储空间后根据自己的业务需求修改存储空间的访问权限。本文介绍如何设置和获取存储空间的访问权限。

设置存储空间的访问权限

存储空间的访问权限有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	private

访问权限	描述	访问权限值
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	public-read-write

- 创建存储空间的同时设置其访问权限

  以下代码用于创建存储空间的同时设置其访问权限：

```
const OSS = require('ali-oss');

const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
// 此处以创建Bucket的同时设置其访问权限为public-read为例。
async function putBucket() {
 const acl = 'public-read';  try {
  await client.putBucket('<Your Bucket Name>', { acl });
 } catch (error) {
  console.log(error)
 }
}

putBucket()
```

- 创建存储空间后修改其访问权限

  以下代码用于创建存储空间后修改其访问权限：

```
const OSS = require('ali-oss');

const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});


async function putBucketACL() {
// 此处以创建Bucket后修改其访问权限为private为例。
 const acl = 'private'
 try {
  await client.putBucketACL('<Your Bucket Name>', acl)
 } catch (error) {
  console.log(error)
 }
}

putBucketACL()
```

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
const OSS = require('ali-oss');

const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});

// 获取存储空间的访问权限。
async function getBucketAcl() {
 const result = await client.getBucketACL('<Your Bucket Name>')
 console.log('acl: ', result.acl)
}

getBucketAcl()
```

4.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

说明

- 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
- 如果该Bucket下还有未完成的上传请求，则需要通过 listUploads 和 abortMultipartUpload 取消请求后才能删除Bucket。

使用 deleteBucket 接口删除Bucket，您需要指定Bucket的名称：

```
let OSS = require('ali-oss');
let client = new OSS({
 region: '<Your region>',
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
async function deleteBucket() {
 try {
  const result = await client.deleteBucket('your bucket name');
  console.log(result);
 } catch (err) {
  console.log(err);
 }
}
deleteBucket();
```

删除存储空间的更多详情，请参见DeleteBucket。

4.5.8. 请求者付费模式

请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

  如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

  数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
const OSS = require('ali-oss')
const client = new OSS({
  region: '<Your region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
async function setBucketRequestPayment(bucket, Payer) {
  try {
    // bucket填写需要设置请求者付费模式的bucket名称。
    // Payer取值为Requester或BucketOwner。
    // Payer设置为Requester，表明该存储空间已开启请求者付费模式，由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用。
    // Payer设置为BucketOwner，表明该存储空间不开启请求者付费模式（默认状态），即请求产生的费用由数据拥有者（BucketOwner）来支付。
    let result = await client.putBucketRequestPayment(bucket, Payer);
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
setBucketRequestPayment('bucketName', 'Requester')
```

设置存储空间请求者付费模式更多详情，请参见PutBucketRequestPayment。

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
const OSS = require('ali-oss')
const client = new OSS({
  region: '<Your region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
async function getBucketRequestPayment(bucket) {
  try {
    // 获取存储空间请求者付费模式配置信息。
    let result = await client.getBucketRequestPayment(bucket);
    console.log(result.payer);
  } catch (e) {
    console.log(e);
  }
}
getBucketRequestPayment('bucketName')
```

获取存储空间请求者付费模式配置信息的更多详情，请参见GetBucketRequestPayment。

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、GetObject和DeleteObject为例，用于指定第三方付费访问Object。其他用于指定第三方付费的Object读写操作接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
const OSS = require('ali-oss')
const bucket = 'bucket-name'
const payer = 'Requester'
const client = new OSS({
  region: '<Your region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: bucket
});
async function put() {
  await client.putBucketRequestPayment(bucket, payer)
  // putObject接口指定付费者。
  await client.put('fileName', 'fileContent', {
    headers: {
      'x-oss-request-payer': 'requester'
    }
  })
}
async function get() {
  await client.putBucketRequestPayment(bucket, payer)
  // getObject接口指定付费者。
  await client.get('fileName', {
    headers: {
      'x-oss-request-payer': 'requester'
    }
  })
}
async function del() {
  await client.putBucketRequestPayment(bucket, payer)
  // deleteObject接口指定付费者。
  await client.delete('fileName', {
    headers: {
      'x-oss-request-payer': 'requester'
    }
  })
}
```

4.5.9. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

生命周期规则包含如下信息：

- 前缀或标签匹配策略：生命周期规则匹配的Object和碎片。
  - 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。
  - 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，OSS对所有拥有这些标签的对象执行生命周期规则。标签匹配不可以作用于碎片。

    说明 对象标签功能详情请参见对象标签。

  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。
  - 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。
- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Object成为非当前版本天数：指定一个过期天数N，并指定非当前版本Object过期后执行什么操作。Object会在其成为非当前版本的N天后过期，并执行指定的操作。

    说明 您可以将过期Object转换为低频访问类型或归档类型，也可以选择删除过期Object。详情请参见生命周期配置元素。

- 碎片过期策略：设置碎片的过期时间及操作。
  - 过期天数：可指定一个过期天数N，碎片会在其最后修改时间的N天后被删除。
  - 过期日期：指定一个过期日期，最后修改时间在该日期之前的碎片会被全部删除。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

通过 `putBucketLifecycle` 来设置生命周期规则：

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>'
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>'
});
async function putBucketLifecycle(lifecycle) {
 try {
  const result = await client.putBucketLifecycle('<bucket-name>', [
  lifecycle
 ]);
  console.log(result);
```

```
 } catch (e) {
   console.log(e);
 }
}
const lifecycle1 = {
 id: 'rule1',
 status: 'Enabled',
 prefix: 'foo/',
 expiration: {
   days: 3 // 指定当前版本Object距其最后修改时间3天后过期。
 }
}
putBucketLifecycle(lifecycle1)
const lifecycle2 = {
 id: 'rule2',
 status: 'Enabled',
 prefix: 'foo/',
 expiration: {
   createdBeforeDate: '2020-02-18T00:00:00.000Z' // 指定日期之前创建的文件过期。
 },
}
putBucketLifecycle(lifecycle2)
const lifecycle3 = {
 id: 'rule3',
 status: 'Enabled',
 prefix: 'foo/',
 abortMultipartUpload: {
   days: 3 // 指定分片3天后过期。
 },
}
putBucketLifecycle(lifecycle3)
const lifecycle4 = {
 id: 'rule4',
 status: 'Enabled',
 prefix: 'foo/',
 abortMultipartUpload: {
   createdBeforeDate: '2020-02-18T00:00:00.000Z' // 指定日期之前创建的分片过期。
 },
}
putBucketLifecycle(lifecycle4)
const lifecycle5 = {
 id: 'rule5',
 status: 'Enabled',
 prefix: 'foo/',
 transition: {
   // 指定当前版本Object距其最后修改时间20天后转归档存储类型。
   days: 20,
   storageClass: 'Archive'
 },
 expiration: {
   days: 21 // 指定当前版本Object距其最后修改时间21天后过期。
 },
}
putBucketLifecycle(lifecycle5)
```

```
const lifecycle6 = {
 id: 'rule6',
 status: 'Enabled',
 prefix: 'foo/',
 transition: {
  // 指定日期之前自动将文件转为归档类型。
  createdBeforeDate: '2020-02-18T00:00:00.000Z', // 指定日期应早于文件过期时间。
  storageClass: 'Archive'
 },
 expiration: {
  createdBeforeDate: '2020-02-19T00:00:00.000Z' // 指定日期之前创建的文件过期。
 },
}
putBucketLifecycle(lifecycle6)
const lifecycle7 = {
 id: 'rule7',
 status: 'Enabled',
 prefix: 'foo/',
 noncurrentVersionExpiration: {
  noncurrentDays: 1 // 设置Object成为非当前版本1天后过期。
 },
}
putBucketLifecycle(lifecycle7)
const lifecycle8 = {
 id: 'rule8',
 status: 'Enabled',
 prefix: 'foo/',
 expiredObjectDeleteMarker: true // 设置自动移除过期删除标记。
}
putBucketLifecycle(lifecycle8)
const lifecycle9 = {
 id: 'rule9',
 status: 'Enabled',
 prefix: 'foo/',
 // 设置非当前版本的Object距其最后修改时间10天之后转为低频访问类型。
 noncurrentVersionTransition: {
  noncurrentDays: '10',
  storageClass: 'IA'
 }
}
putBucketLifecycle(lifecycle9)
const lifecycle10 = {
 id: 'rule10',
 status: 'Enabled',
 prefix: 'foo/',
 // 设置非当前版本的Object距其最后修改时间10天之后转为低频访问类型。
 noncurrentVersionTransition: {
  noncurrentDays: '10',
  storageClass: 'IA'
 },
 // 指定规则所适用的对象标签。
 tag: [{
  key: 'key1',
  value: 'value1'
 }
```

```
  },
  {
    key: 'key2',
    value: 'value2'
  }]
}
putBucketLifecycle(lifecycle10)
```

查看生命周期规则

通过 `getBucketLifecycle` 来查看生命周期规则：

```
const OSS = require('ali-oss')
const client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketLifecycle () {
  try {
    let result = await client.getBucketLifecycle('<bucket-name>');
    console.log(result.rules); // 获取生命周期规则。
    rules.forEach(rule => {
      console.log(rule.id) //  查看生命周期规则id。
      console.log(rule.status) // 查看生命周期规则状态。
      console.log(rule.tags) // 查看规则标签。
      console.log(rule.expiration.days) // 查看过期天数规则。
      console.log(rule.expiration.createdBeforeDate) // 查看过期日期规则。
      // 查看过期分片规则。
      console.log(rule.abortMultipartUpload.days || rule.abortMultipartUpload.createdBeforeDate)
      // 查看存储类型转换规则。
      console.log(rule.transition.days || rule.transition.createdBeforeDate) // 查看存储类型转换时间。
      console.log(rule.transition.storageClass) // 转换存储类型。
      // 查看是否自动删除过期删除标记。
      console.log(rule.transition.expiredObjectDeleteMarker)
      // 查看非当前版本Object存储类型转换规则。
      console.log(rule.noncurrentVersionTransition.noncurrentDays) // 查看非当前版本Object存储类型转换时间
。
      console.log(rule.noncurrentVersionTransition.storageClass) // 查看非当前版本Object转换的存储类型。
      // 查看非当前版本Object的过期规则。
      console.log(rule.noncurrentVersionExpiration.noncurrentDays)
    })
  } catch (e) {
    console.log(e);
  }
}
getBucketLifecycle();
```

清空生命周期规则

通过 `deleteBucketLifecycle` 来清空生命周期规则：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function deleteBucketLifecycle () {
  try {
    let result = await client.deleteBucketLifecycle('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
deleteBucketLifecycle();
```

4.5.10. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
//设置授权策略。
const policy = {
  Version: '1',
  Statement: [
  {
    Action: ['oss:PutObject', 'oss:GetObject'],
    Effect: 'Deny',
    Principal: ['1234567890'],
    Resource: ['acs:oss:*:1234567890:*/*']
   }
  ]
};
async function putPolicy() {
  const result = await client.putBucketPolicy('<Your Bucket Name>', policy);
  console.log(result)
}
putPolicy()
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 获取授权策略。
async function getPolicy() {
  const result = await client.getBucketPolicy('<Your Bucket Name>');
  console.log(result.policy)
}
getPolicy()
```

有关获取Bucket Policy信息详情，请参见GetBucketPolicy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
//删除授权策略。
async function deletePolicy() {
  const result = await client.deleteBucketPolicy('<Your Bucket Name>');
  console.log(result)
}
deletePolicy()
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

4.5.11. 合规保留策略

对象存储OSS允许针对存储空间（Bucket）设置基于时间的合规保留策略，保护周期为1天到70年。本文介绍如何新建、获取、锁定合规保留策略等。

背景信息

OSS支持WORM（Write Once Read Many）特性，允许您以不可删除、不可篡改的方式保存和使用数据。

当基于时间的合规保留策略创建24小时后未提交锁定，则该策略自动失效。当合规保留策略锁定后，您可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，不允许删除Object及合规保留策略，且无法缩短策略保护周期，仅允许延长Object的保留时间。有关合规保留策略的详情，请参见合规保留策略。

新建合规保留策略

以下代码用于新建合规保留策略：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
// 创建合规保留策略。
async function initiateBucketWorm() {
 const bucket = '<Your Bucket Name>'
 // 指定Object保护天数。
 const days = '<Retention Days>'
 const res = await client.initiateBucketWorm(bucket, days)
 console.log(res.wormId)
}
initiateBucketWorm()
```

有关新建合规保留策略的详情，请参见InitiateBucketWorm。

取消未锁定的合规保留策略

以下代码用于取消未锁定的合规保留策略：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
// 取消合规保留策略。
async function abortBucketWorm() {
 const bucket = '<Your Bucket Name>'
 try {
  await client.abortBucketWorm(bucket)
  console.log('abort success')
 } catch(err) {
 console.log('err: ', err)
 }
}
abortBucketWorm()
```

有关取消未锁定的合规保留策略的详情，请参见AbortBucketWorm。

锁定合规保留策略

以下代码用于锁定合规保留策略：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
// 锁定合规保留策略。
async function completeBucketWorm() {
 const bucket = '<Your Bucket Name>'
 const wormId = '<Your Worm Id>'
 try {
 await client.completeBucketWorm(bucket, wormId)
 } catch(err) {
  console.log(err)
 }
}
completeBucketWorm()
```

有关锁定合规保留策略的详情，请参见CompleteBucketWorm。

获取合规保留策略

以下代码用于获取合规保留策略：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
// 获取合规保留策略。
async function getBucketWorm() {
 const bucket = '<Your Bucket Name>'
 try {
 const res = await client.getBucketWorm(bucket)
  // 查看合规保留策略Id。
  console.log(res.wormId)
  // 查看合规保留策略状态。未锁定状态为"InProgress"，锁定状态为"Locked"。
  console.log(res.state)
  // 查看Object的保护时间。
  console.log(res.days)
 } catch(err) {
  console.log(err)
 }
}
getBucketWorm()
```

有关获取合规保留策略的详情，请参见GetBucketWorm。

延长Object的保留天数

以下代码用于延长已锁定的合规保留策略中Object的保留天数：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
// 延长已锁定的合规保留策略中Object的保留天数。
async function extendBucketWorm() {
 const bucket = '<Your Bucket Name>'
 const wormId = '<Your Worm Id>'
 const days = '<Retention Days>'
 try {
 const res = await client.extendBucketWorm(bucket, wormId, days)
  console.log(res)
 } catch(err) {
  console.log(err)
 }
}
extendBucketWorm()
```

有关延长已锁定的合规保留策略中Object的保留天数详情，请参见ExtendBucketWorm。

4.5.12. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

> **说明**
> - 只有Bucket的拥有者及授权子账户才能为Bucket设置用户标签，否则返回403 Forbidden错误，错误码：AccessDenied。
> - 最多可设置20对Bucket用户标签（Key-Value对）。
> - Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
> - Value最大长度为128字节，可以为空。
> - Key和Value必须为UTF-8编码。
> - PutBucketTagging是覆盖语义，即新设置的标签会完全覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 设置Bucket标签。
async function putBucketTags(bucketName, tag) {
 try {
  let result = await client.putBucketTags(bucketName, tag);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
const tag = { a: '1', b: '2' };
putBucketTags('bucketName', tag)
```

获取Bucket标签

以下代码用于获取Bucket标签：

```
const OSS = require('ali-oss')
const client = new OSS({
 //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
//获取Bucket标签。
async function getBucketTags(bucketName) {
 try {
  let result = await client.getBucketTags(bucketName);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
getBucketTags('bucketName')
```

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
const OSS = require('ali-oss')
const client = new OSS({
 //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 列举带指定标签的Bucket。
async function listBucketsTags() {
 try {
  const params = {
   // 填充tag-key、tag-value字段到listBuckets接口的params参数中。
   'tag-key': '<yourTagKey>',
   'tag-value': '<yourTagValue>'
  }
  const result = await client.listBuckets(params);
  console.log(result);
 } catch (err) {
  console.log(err);
 }
}
listBucketTags('bucketName')
```

删除Bucket标签

以下代码用于删除Bucket标签：

```
const OSS = require('ali-oss')
const client = new OSS({
 //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 删除Bucket标签。
async function deleteBucketTags(bucketName) {
 try {
  let result = await client.deleteBucketTags(bucketName);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
deleteBucketTags('bucketName')
```

4.5.13. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **注意**
>
> 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

添加清单配置时，有如下注意事项：

- 单个Bucket最多只能配置1000条清单规则。
- 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单配置：

```
const OSS = require('ali-oss');

const client = new OSS({
 bucket: '<Your BucketName>',
 // Region以杭州为例，其他Region按实际情况填写。
 region: '<oss-cn-hangzhou>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});

const inventory = {
 // 设置清单配置ID。
 id: 'default',
 // 清单配置是否启用的标识，取值为true或false。
 isEnabled: false,
 //（可选）设置清单筛选规则，指定筛选object的前缀。
 prefix: 'ttt',
 OSSBucketDestination: {
  // 设置清单格式。
  format: 'CSV',
  // 目标Bucket拥有者的账号ID。
  accountId: '<Your AccountId>',
  // 目标Bucket的角色名称。
  rolename: 'AliyunOSSRole',
  // 目标Bucket的名称。
  bucket: '<Your BucketName>',
  //（可选）清单结果的存储路径前缀。
  prefix: '<Your Prefix>',
  // 如果需要使用SSE-OSS加密清单，请参考以下代码。
  //encryption: {'SSE-OSS': ''},
  // 如果需要使用SSE-KMS加密清单，请参考以下代码。
  /*
   encryption: {
    'SSE-KMS': {
     keyId: 'test-kms-id';
    };,
   */
 }
```

```
  },
  // 设置清单的生成计划，WEEKLY对应每周一次，DAILY对应每天一次。
  frequency: 'Daily',
  // 设置清单结果中包含了Object的所有版本, 如果设置为Current，则表示仅包含Object的当前版本。
  includedObjectVersions: 'All',
  optionalFields: {
    //（可选）设置清单中包含的Object属性。
    field: ["Size", "LastModifiedDate", "ETag", "StorageClass", "IsMultipartUploaded", "EncryptionStatus"]
  },
}

async function putInventory(){
  // 需要添加清单配置的Bucket名称。
  const bucket = '<Your BucketName>';
  try {
    await client.putBucketInventory(bucket, inventory);
    console.log('清单配置添加成功')
  } catch(err) {
    console.log('清单配置添加失败: ', err);
  }
}

putInventory()
```

有关添加Bucket清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
const OSS = require('ali-oss');

const client = new OSS({
 bucket: '<Your BucketName>',
 // Region以杭州为例，其他Region按实际情况填写。
 region: '<oss-cn-hangzhou>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});

async function getBucketInventoryById() {
 // 指定获取清单配置的Bucket名称。
 const bucket = '<Your BucketName>';
 try {
  // 查看清单配置。
  const result = await client.getBucketInventory(bucket, 'inventoryid');
  console.log(result.inventory);
 } catch (err) {
  console.log(err)
 }
}

getBucketInventoryById();
```

有关查看Bucket清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明**
>
> 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
const OSS = require('ali-oss');

const client = new OSS({
  bucket: '<Your BucketName>',
  // Region以杭州为例，其他Region按实际情况填写。
  region: '<oss-cn-hangzhou>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});

async function listBucketInventory() {
  const bucket = '<Your Bucket Name>';
  // 列举清单配置。
  const result = await client.listBucketInventory(bucket);
  console.log(result.inventoryList);
  // 默认每次最多列举100条结果，如果配置超过100条，结果将会分页返回，通过传入nextContinuationToken方式列举下一页。
  const { nextContinuationToken } = result;
  const resultNext = await client.listBucketInventory(bucket, nextContinuationToken);
  console.log(result.inventoryList);
}

listBucketInventory();
```

有关批量列举Bucket清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
const OSS = require('ali-oss');

const client = new OSS({
 bucket: '<Your BucketName>',
 // Region以杭州为例，其他Region按实际情况填写。
 region: '<oss-cn-hangzhou>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});

async function deleteBucketInventoryById() {
 const bucket = '<Your Bucket Name>';
 // 删除指定ID的清单配置。
 const inventory_id = '<Your InventoryId>';
 try {
  await client.deleteBucketInventory(bucket, inventory_id);
 } catch (err) {
  console.log(err)
 }
}
deleteBucketInventoryById();
```

有关删除Bucket清单配置的详情，请参见DeleteBucketInventory。

4.5.14. 跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

> **说明** 更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域资源共享及GitHub说明。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则。

```
let oss = require('ali-oss');
const client = oss({
  accessKeyId: 'your access key',
  accessKeySecret: 'your access secret',
  bucket: 'your bucket name',
  region: 'oss-cn-hangzhou'
})
client.putBucketCORS('hello', [
  {
    allowedOrigin: '*',
    allowedMethod: [
      'GET',
      'HEAD',
    ],
  }
]).then((result) => {});
```

更多关于设置跨域资源共享规则的介绍，请参见PutBucketCors。

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则。

```
let OSS = require('ali-oss');
const client = new OSS({
  region: 'your region',
  accessKeyId: 'your accessKeyId',
  accessKeySecret: 'your accessKeySecret',
})
client.getBucketCORS('bucketName').then((res) => {
  console.log(res);
}).catch(e => {
  console.log(e)
})
```

更多关于获取跨域资源共享规则的介绍，请参见GetBucketCors。

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则。

```
let OSS = require('ali-oss');
const client = new OSS({
  region: 'your region',
  accessKeyId: 'your accessKeyId',
  accessKeySecret: 'your accessKeySecret',
})
client.deleteBucketCORS('bucketName').then((res) => {
  console.log(res);
}).catch(e => {
  console.log(e)
})
```

更多关于删除跨域资源共享规则的介绍，请参见DeleteBucketCors。

4.5.15. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

关于访问日志的更多信息，请参见开发指南中的设置访问日志记录。

开启访问日志记录

通过 `putBucketLogging` 来开启存储空间的访问日志记录。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function putBucketLogging () {
  try {
    let result = await client.putBucketLogging('bucket-name', 'logs/');
    console.log(result)
  } catch (e) {
    console.log(e)
  }
}
putBucketLogging();
```

查看访问日志设置

通过 `getBucketLogging` 来查看存储空间的访问日志设置。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketLogging() {
  try {
    let result = await client.getBucketLogging('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
})
getBucketLogging();
```

关闭访问日志记录

通过 `deleteBucketLogging` 来关闭存储空间的访问日志记录。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function deleteBucketLogging () {
  try {
    let result = await client.deleteBucketLogging('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
deleteBucketLogging();
```

4.5.16. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置Referer白名单

通过 `putBucketReferer` 设置Referer白名单。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function putBucketReferer () {
  try {
  let result = await client.putBucketReferer('bucket-name', true, [
    'my-domain.com',
    '*.example.com'
  ]);
  console.log(result);
  } catch (e) {
    console.log(e);
  }
}
putBucketReferer();
```

查看Referer白名单

通过 `getBucketReferer` 查看Referer白名单。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketReferer () {
  try {
    let result = await client.getBucketReferer('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
getBucketReferer();
```

清空Referer白名单

通过 `deleteBucketReferer` 清空Referer白名单。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function deleteBucketReferer () {
  try {
    let result = await client.deleteBucketReferer('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
deleteBucketReferer();
```

4.5.17. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

通过 `putBucketWebsite` 来设置静态网站托管：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function putBucketWebsite () {
  try {
    let result = await client.putBucketWebsite('bucket-name', {
    index: 'index.html',
    error: 'error.html'
  });
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
putBucketWebsite();
```

查看静态网站托管配置

通过 `getBucketWebsite` 来查看静态网站托管配置：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketWebsite () {
  try {
    let result = await client.getBucketWebsite('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
getBucketWebsite();
```

删除静态网站托管配置

通过 `deleteBucketWebsite` 来删除静态网站托管配置：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function deleteBucketWebsite() {
  try {
    let result = await client.deleteBucketWebsite('bucket-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
deleteBucketWebsite();
```

4.5.18. 使用自定义域名

通过添加CNAME记录将自定义域名绑定到指定的存储空间（Bucket）后，您可以使用自有域名访问OSS资源。本文介绍如何使用自定义域名。

假设您的自有域名为my-domain.com，之前访问所有图片都是通过 `http://img.my-domain.com/x.jpg` 的格式访问，将资源迁移到OSS后，通过绑定自定义域名的方式，您仍然可以使用原来的地址访问图片。更多信息，请参见开发指南中的绑定自定义域名。

在使用SDK时，您也可以将自定义域名作为endpoint，此时需要设置 `cname` 参数为true。

```
let OSS = require('ali-oss')
let client = new OSS({
  endpoint: 'http://img.my-domain.com', // 使用自定义域名作为endpoint。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  cname: true
});
client.useBucket('examplebucket')
```

注意 自定义域名已绑定到某个特定的Bucket，因此使用CNAME时无法使用list_buckets接口。

4.6. 上传文件

4.6.1. 概述

在对象存储OSS中，操作的基本数据单元是文件（Object）。OSS Node.js SDK提供了丰富的文件上传方式。

OSS Node.js SDK文件上传方式如下：

- 上传本地文件：最大不能超过5 GB。
- 上传缓存中的内容：最大不能超过5 GB。
- 流式上传：最大不能超过5 GB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8 TB。
- 追加上传：使用AppendObject方法在已上传的Appendable Object类型文件后面直接追加内容。最大不能超过5 GB。
- 断点续传上传：支持并发上传、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8 TB。

4.6.2. 上传本地文件

通过put接口将本地文件上传到OSS。

以下代码用于将本地文件examplefile.txt上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
const OSS = require('ali-oss')
const path = require("path")
const client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourregion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
async function put () {
 try {
  // 填写OSS文件完整路径和本地文件的完整路径。OSS文件完整路径中不能包含Bucket名称。
  // 如果本地文件的完整路径中未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
  const result = await client.put('exampleobject.txt', path.normalize('D:\\localpath\\examplefile.txt'));
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
put();
```

4.6.3. 上传本地内存

本文档介绍如何将本地内存中的内容上传到 OSS。

说明

您可以通过 `put` 接口简单地将本地内存中的内容上传到 OSS：

```
let OSS = require('ali-oss');
let client = new OSS({
 region: '<Your region>',
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
async function putBuffer () {
 try {
  let result = await client.put('object-name', new Buffer('hello world'));
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
putBuffer();
```

4.6.4. 流式上传

通过putStream接口将文件流、网络流等数据流上传到存储空间（Bucket）中的文件（Object）。本文以文件流为例介绍如何进行流式上传。

> **说明** stream参数可以是任何实现了Readable Stream的对象，包含文件流，网络流等。

以下代码用于将文件流上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
let OSS = require('ali-oss');
let fs = require('fs');
let client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称，例如examplebucket。
 bucket: 'examplebucket',
});
async function putStream () {
 try {
 // 使用'chunked encoding'。使用putStream接口时，SDK默认会发起一个'chunked encoding'的HTTP PUT请求。
 // 填写本地文件的完整路径，从本地文件中读取数据流。
 // 如果本地文件的完整路径中未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
 let stream = fs.createReadStream('D:\\localpath\\examplefile.txt');
 // 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
 let result = await client.putStream('exampledir/exampleobject.txt', stream);
 console.log(result);
 // 不使用'chunked encoding'。如果在options指定了contentLength参数，则不会使用chunked encoding。
 let stream = fs.createReadStream('D:\\localpath\\examplefile.txt');
 let size = fs.statSync('D:\\localpath\\examplefile.txt').size;
 let result = await client.putStream(
  'exampledir/exampleobject.txt', stream, {contentLength: size});
 console.log(result);
 } catch (e) {
  console.log(e)
 }
}
putStream();
```

4.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

背景信息

当需要上传的文件较大时，您可以通过MultipartUpload接口进行分片上传。分片上传是指将要上传的文件分成多个数据块（Part）来分别上传。当其中一些分片上传失败后，OSS将保留上传进度记录，再次重传时只需要上传失败的分片，而不需要重新上传整个文件。一般大于100 MB的文件，建议采用分片上传的方法，通过断点续传和重试，提高上传成功率。

在使用MultipartUpload接口时，如果遇到 `ConnectionTimeoutError` 超时问题，业务方需自行处理超时逻辑。例如通过缩小分片大小、增加超时时间、重试请求或者捕获 `ConnectionTimeoutError` 错误等方法处理超时。更多信息，请参见网络错误处理。

分片上传涉及的相关参数说明请参见下表。

类型	参数	说明
必选参数	name {String}	Object的名称。即不包括Bucket名称在内的Object的完整路径。
	file {String|File}	表示文件路径或者HTML5文件。
[options] {Object} 可选参数	[checkpoint] {Object}	记录本地分片上传结果的文件。开启断点续传功能时需要设置此参数，上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。
	[parallel] {Number}	并发上传的分片个数，默认为5。无特殊需求，不需要手动设置此参数。
	[partSize] {Number}	指定上传的每个分片的大小，范围为100 KB~5 GB。单个分片默认大小为1 * 1024 * 1024（即1 MB）。无特殊需求，不需要手动设置此参数。
	[progress] {Function}	表示进度回调函数，用于获取上传进度，可以是async的函数形式。回调函数包含三个参数： • percentage {Number}：进度百分比（0~1之间的小数）。 • checkpoint {Object}：与[options] {Object}中的[checkpoint] {Object}定义相同。 • res {Object}：单个分片上传成功返回的response。
	[meta] {Object}	用户自定义的Header meta信息，Header前缀为 `x-oss-meta-` 。
	[mime] {String}	设置Content-Type请求头。

类型	参数	说明
	[headers] {Object}	其他Header。更多信息，请参见RFC 2616。其中： • 'Cache-Control'：通用消息头被用在HTTP请求和响应中通过指定指令来实现缓存机制，例如 `Cache-Control: public, no-cache` 。 • 'Content-Disposition'：指示回复的内容该以何种形式展示，是以网页预览的形式，还是以附件的形式下载并保存到本地，例如 `Content-Disposition: some name` 。 • 'Content-Encoding'：用于对特定媒体类型的数据进行压缩，例如 `Content-Encoding: gzip` 。 • 'Expires'：过期时间，单位为毫秒。

以下示例代码中的catch语法，请自行学习es6 promise、async/await。 SDK的使用方式，请参见安装。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
const progress = (p, _checkpoint) => {
 console.log(p); // Object的上传进度。
 console.log(_checkpoint); // 分片上传的断点信息。
};
// 开始分片上传。
async function multipartUpload() {
 try {
  // object-name可以定义为文件名（例如file.txt）或目录（例如abc/test/file.txt）的形式，实现将文件上传至Bucket根目录或Bucket下的指定目录。
  const result = await client.multipartUpload('<Object Name>', 'file-object', {
   progress,
   // meta是用户自定义的元数据，通过head接口可以获取到Object的meta数据。
   meta: {
    year: 2020,
    people: 'test',
   },
  });
  console.log(result);
  const head = await client.head('object-name');
  console.log(head);
 } catch (e) {
  // 捕获超时异常。
  if (e.code === 'ConnectionTimeoutError') {
   console.log('TimeoutError');
   // do ConnectionTimeoutError operation
  }
  console.log(e);
 }
}
multipartUpload();
```

注意 Node.js分片上传过程中不支持MD5校验。建议分片上传完成后调用CRC64库自行判断是否进行CRC64校验。

取消分片上传事件

您可以调用 `client.abortMultipartUpload` 方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用这个uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件。

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
async function abortMultipartUpload() {
 const name = '<Object Name>'; // Object所在Bucket的完整路径。
 const uploadId = '<Upload Id>'; // 分片上传uploadId。
 const result = await client.abortMultipartUpload(name, uploadId);
 console.log(result);
}
abortMultipartUpload();
```

列举分片上传事件

调用 `client.listUploads` 方法列举出分片上传事件示例代码如下：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
async function listUploads(query = {}) {
 // query参数支持prefix、marker、delimiter、upload-id-marker和max-uploads。
 const result = await client.listUploads(query);
 result.uploads.forEach(upload => {
  console.log(upload.uploadId); // 分片上传的uploadId。
  console.log(upload.name); // 将所有上传完成后的分片（Part）组合为一个完整的Object，并指定Object所在Bucket的完整路径。
 });
}
const query = {
 'max-uploads': 1000, // 限定此次返回Multipart Uploads事件的最大数目。max-uploads默认值和最大值均为1000。
};
listUploads(query);
```

列举已上传的分片

调用 `client.listParts` 方法列举指定uploadId下所有已经上传成功的分片。

- 简单列举已上传的分片

  以下代码用于简单列举已上传的分片：

```
const OSS = require('ali-oss');
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
async function listParts() {
 const query = {
  'max-parts': 1000, // 指定OSS响应中的最大分片（Part）数目。max-parts默认值和最大值均为1000。
 };
 const result = await client.listParts('<Object Name>', '<Upload Id>', query);
 result.parts.forEach(part => {
  console.log(part.PartNumber);
  console.log(part.LastModified);
  console.log(part.ETag);
  console.log(part.Size);
 });
}
listParts();
```

- 列举所有已上传的分片

  默认情况下，`client.listParts` 只能一次列举1000个分片。当分片数量大于1000时，请使用以下代码列举所有已上传的分片。

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
async function listParts() {
 const query = {
  'max-parts': 1000, // 设置OSS响应中的最大分片（Part）数目。max-parts默认值和最大值均为1000。
 };
 let result;
 do {
  result = await client.listParts('<Object Name>', '<Upload Id>', query);
  // 设置列举分片的起始位置。
  query['part-number-marker'] = result.nextPartNumberMarker;
  result.parts.forEach(part => {
   console.log(part.PartNumber);
   console.log(part.LastModified);
   console.log(part.ETag);
   console.log(part.Size);
  });
 } while (result.isTruncated === "true");
}
listParts();
```

4.6.6. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

示例代码

以下代码用于追加上传：

```
let OSS = require('ali-oss')
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 目标Bucket名称。
 bucket: '<Your bucket name>',
});
async function append () {
 // 第一次追加上传。返回值为下一次追加的位置。
 // object-name表示不包含Bucket名称在内的Object的完整路径，例如destfolder/examplefile.txt。
 // local-file填写包含后缀在内的本地文件完整路径，例如/users/local/examplefile.txt。
 let result = await client.append('object-name', 'local-file')
 // 第二次追加。后续追加的位置（position）是追加前文件的长度（Content-Length）。
 result = await client.append('object-name', 'local-file', {
  position: result.nextAppendPosition
 })
}
append();
```

有关Bucket命名规范的详情，请参见存储空间（Bucket）。有关Object命名规范的详情，请参见对象（Object）。

4.6.7. 断点续传上传

断点续传上传将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

> **说明** 以下示例代码中的catch语法，请自行学习es6 promise、async/await。SDK的使用方式，请参见安装。

断点续传详情请参见开发指南中的断点续传。您还可以通过设置生命周期规则来定时清理不需要的Part，详情请参考管理碎片。

分片上传提供 progress 参数方便用户传递进度回调，在回调中 SDK 将当前已经上传成功的比例和断点信息作为参数。为了实现断点上传，可以在上传过程中保存断点信息（checkpoint），发生错误后，再将已保存的 checkpoint 作为参数传递给 multipartUpload，此时将从上次失败的地方继续上传。

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
let checkpoint;
async function resumeUpload() {
  // retry 5 times
  for (let i = 0; i < 5; i++) {
    try {
      const result = await client.multipartUpload('object-name', filePath, {
        checkpoint,
        async progress(percentage, cpt) {
          checkpoint = cpt;
        },
      });
      console.log(result);
      break; // break if success
    } catch (e) {
      console.log(e);
    }
  }
}
resumeUpload();
```

上述示例代码将 checkpoint 保存在变量中，如果程序崩溃，则 checkpoint 信息会丢失。建议将 checkpoint 保存在文件中，在程序重启后则可以从文件中读取 checkpoint 信息。

4.6.8. 上传回调

本文介绍如何使用上传回调。

> **说明** 上传回调的完整代码请参见GitHub。关于上传回调的更多信息，请参见开发指南中的上传回调和API参考中的Callback。

以下代码用于在上传本地文件examplefile.txt到目标存储空间examplebucket中的exampleobject.txt文件时使用上传回调（callback）。

```
const oss = require('ali-oss');
var path = require('path');
const client = oss({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourregion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket'
});
const options = {
 callback: {
  // 设置回调请求的服务器地址，例如http://oss-demo.aliyuncs.com:23450。
  url: 'http://oss-demo.aliyuncs.com:23450',
  //（可选）设置回调请求消息头中Host的值，即您的服务器配置Host的值。
  //host: 'yourCallbackHost',
  // 设置发起回调时请求body的值。
  body: 'bucket=${bucket}&object=${object}&var1=${x:var1}',
  // 设置发起回调请求的Content-Type。
  contentType: 'application/x-www-form-urlencoded',
  // 设置发起回调请求的自定义参数。
  customValue: {
   var1: 'value1',
   var2: 'value2'
  }
 }
}
async function put () {
 try {
  // 填写Object完整路径和本地文件的完整路径。Object完整路径中不能包含Bucket名称。
  // 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
  let result = await client.put('exampleobject.txt', path.normalize('/localpath/examplefile.txt'), options);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
put();
```

4.7. 下载文件

4.7.1. 概述

OSS Node.js SDK提供了丰富的文件下载方式。

- 下载到本地文件
- 下载到本地内存
- 流式下载
- 范围下载

- 限定条件下载

4.7.2. 下载到本地文件

本文介绍如何将存储空间（Bucket）中的文件（Object）下载到本地文件。

以下代码用于将examplebucket中的exampleobject.txt文件下载到本地*D:\localpath*路径下的examplefile.txt。

```
let OSS = require('ali-oss');
let client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket'
});
async function get () {
 try {
  // 填写Object完整路径和本地文件的完整路径。Object完整路径中不能包含Bucket名称。
  // 如果指定的本地文件存在会覆盖，不存在则新建。
  // 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
  let result = await client.get('exampleobject.txt', 'D:\\localpath\\examplefile.txt');
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
get();
```

4.7.3. 下载到本地内存

本文介绍如何将 OSS 文件下载到本地内存。

您可以通过 `get` 接口简单地将 OSS 文件下载到本地内存：

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function getBuffer () {
  try {
    let result = await client.get('object-name');
    console.log(result.content);
  } catch (e) {
    console.log(e);
  }
}
getBuffer();
```

4.7.4. 流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

以下代码用于流式下载examplebucket中的exampleobject.txt文件到本地*D:\localpath*路径下的examplefile.txt文件。

> **说明** 使用 `getStream` 下载文件时，返回的 `Readable Stream` 用于流式地处理文件内容。

```
const OSS = require('ali-oss');
const fs = require('fs');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
async function getStream () {
 try {
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  const result = await client.getStream('exampleobject.txt');
  console.log(result);
  // 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
  // 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
  const writeStream = fs.createWriteStream('D:\\localpath\\examplefile.txt');
  result.stream.pipe(writeStream);
 } catch (e) {
  console.log(e);
 }
}
getStream()
```

4.7.5. 范围下载

如果仅需要文件（Object）中的部分数据，您可以使用范围下载，下载指定范围内的数据。

指定正常的下载范围

当您指定的范围首端和末端都在文件的有效区间内，则按指定区间正常下载文件。例如现有大小为1000字节的Object，则指定的有效区间应为0~999。

以下代码用于指定正常的下载范围来下载文件：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
 let start = 1, end = 900;
 // yourObjectName表示不包含Bucket名称在内的Object的完整路径，例如destfolder/examplefile.txt。
 // 获取目标Object的1~900字节范围内的数据，包含1和900，共900字节的数据。
 // 如果指定范围的首端或末端不在有效区间，则下载整个文件的内容，返回HTTP Code为200。
 let result = await client.get("<yourObjectName>", {
  headers: {
   Range: `bytes=${start}-${end}`,
  },
 })
 console.log(result.content.toString())
};
main();
```

有关Bucket命名规范的详情，请参见存储空间（Bucket）。有关Object命名规范的详情，请参见对象（Object）。

兼容行为范围下载

通过在请求中增加请求头 `x-oss-range-behavior:standard` ，改变指定范围不在有效区间时OSS的下载行为。

以下代码用于兼容行为范围下载：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyun
cs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
 // 上传大小为10字节的文件exampleobject.txt。
 let buf = Buffer.from("abcdefghij");
 await client.put("exampleobject.txt", buf);
 let result = await client.get("exampleobject.txt", {
 //  指定Range: bytes=5-15，此时范围末端取值不在有效区间，返回6~10字节范围内容，且HTTP Code为206。
  headers: {
   Range: "bytes=5-15",
   "x-oss-range-behavior": "standard",
  },
 });
 console.log(result.content.toString() === 'fghij')
 console.log(result.res.status === 206)
 try{
  await client.get("exampleobject.txt", {
   // 指定Range: bytes=15-25，此时范围首端取值不在有效区间，返回HTTP Code为416，错误码为InvalidRange。
   headers: {
    Range: "bytes=15-25",
    "x-oss-range-behavior": "standard",
   },
  })
 }catch(e) {
  console.log(e.status === 416);
  console.log(e.name === 'InvalidRangeError')
 }
}
main();
```

4.7.6. 限定条件下载

下载文件（Object）时允许指定条件。满足条件则下载，不满足条件则返回错误且不下载。

下载早于指定时间修改的文件

以下代码用于下载早于指定时间修改的文件：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyun
cs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
 // 向目标Bucket上传名为exampleobject.txt的文件，文件内容自定义。
  await client.put('exampleobject.txt', Buffer.from('contenttest'))
 // 在请求头If-Modified-Since中指定时间，如果指定的时间早于文件实际修改时间，则下载文件。
let result = await client.get("exampleobject.txt", {
  headers: {
    "If-Modified-Since": new Date("1970-01-01").toGMTString(),
  },
});
console.log(result.content.toString() === "contenttest");
console.log(result.res.status === 200);
// 如果指定的时间等于或者晚于文件实际修改时间，则返回304 Not Modified。
result = await client.get("exampleobject.txt", {
  headers: {
    "If-Modified-Since": new Date().toGMTString(),
  },
});
console.log(result.content.toString() === "");
console.log(result.res.status === 304);
console.log(e.code === "Not Modified");
}
main();
```

有关Bucket命名规范的详情，请参见存储空间（Bucket）。有关Object命名规范的详情，请参见对象（Object）。

下载不早于指定时间修改的文件

以下代码用于下载不早于（即等于或晚于）指定时间修改的文件：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
 // 向目标Bucket上传名为exampleobject.txt的文件，文件内容自定义。
  await client.put('exampleobject.txt', Buffer.from('contenttest'))
// 在请求头If-Unmodified-Since中指定时间，如果指定的时间等于或者晚于文件实际修改时间，则下载文件。
let result = await client.get("exampleobject.txt", {
  headers: {
    "If-Unmodified-Since": new Date().toGMTString(),
  },
});
console.log(result.content.toString() === "contenttest");
console.log(result.res.status === 200);
// 如果指定的时间早于文件实际修改时间，则返回412 Precondition Failed。
try {
  await client.get("exampleobject.txt", {
    headers: {
      "If-Unmodified-Since": new Date("1970-01-01").toGMTString(),
    },
  });
} catch (e) {
  console.log(e.status === 412);
  console.log(e.code === "PreconditionFailed");
}
main();
```

下载与指定ETag匹配的文件

ETag用于标识文件内容是否发生变化。以下代码用于下载与指定ETag匹配的文件：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
  // 向目标Bucket上传名为exampleobject.txt的文件，并获取文件的ETag。
  let {res: {headers: {etag}}} = await client.put('exampleobject.txt', Buffer.from('contenttest'))
}
// 在请求头If-Match中传入ETag，如果传入的ETag和文件的ETag匹配，则下载文件。
let result = await client.get("exampleobject.txt", {
  headers: {
    "If-Match": etag,
  },
});
console.log(result.content.toString() === "contenttest");
console.log(result.res.status === 200);
// 如果传入的ETag和文件的ETag不匹配，则返回412 Precondition Failed。
try {
  await client.get("exampleobject.txt", {
    headers: {
      "If-Match": etag,
    },
  });
} catch (e) {
  console.log(e.status === 412);
  console.log(e.code === "PreconditionFailed");
}
main();
```

下载与指定ETag不匹配的文件

以下代码用于下载与指定ETag不匹配的文件：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
const accessKeyId = "yourAccessKeyId";
const accessKeySecret = "yourAccessKeySecret";
// 填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyun
cs.com。
const endpoint = "yourEndpoint";
// 填写目标Bucket名称。
const bucket = "yourBucketName";
async function main() {
  // 向目标Bucket上传名为exampleobject.txt的文件，并获取文件的ETag。
  let {res: {headers: {etag}}} = await client.put('exampleobject.txt', Buffer.from('contenttest'))
}
// 通过在请求头If-None-Match中传入ETag，如果传入的ETag和文件的ETag不匹配，则下载文件。
let result = await client.get("exampleobject.txt", {
  headers: {
    "If-None-Match": etag,
  },
});
console.log(result.content.toString() === "contenttest");
console.log(result.res.status === 200);
// 如果传入的ETag和文件的ETag匹配，则返回304 Not Modified。
result = await client.get("exampleobject.txt", {
  headers: {
    "If-None-Match": etag,
  },
});
console.log(result.content.toString() === "");
console.log(result.res.status === 304);
console.log(e.code === "Not Modified");
}
main();
```

4.8. 管理文件

4.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object）。

包括以下操作：

- 判断文件是否存在
- 管理文件元信息
- 管理文件访问权限
- 列举文件
- 拷贝文件
- 删除文件
- 解冻归档文件

4.8.2. 判断文件是否存在

对存储空间（Bucket）中的任意文件（Object）进行相关操作前，您需要先判断该文件是否存在。

以下代码用于判断文件是否存在：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function isExistObject(name, options = {}) {
  try {
    await client.head(name, options);
    console.log('文件存在')
  } catch (error) {
    if (error.code === 'NoSuchKey') {
      console.log('文件不存在')
    }
  }
}
// 用于判断未开启版本控制状态的Bucket中的Object是否存在。
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
const name = '<Your ObjectName>'
isExistObject(name)
// 用于判断受版本控制Bucket中指定versionId的Object是否存在。
const options = {
   // 填写Object的versionId。
   versionId: '<Your Object VersionId>'
}
isExistObject(name, options)
```

4.8.3. 管理文件元信息

文件元信息（Object Meta）包括 HTTP header 和自定义元信息，详情请参见开发指南中的文件元信息。

向OSS上传文件时，您可以指定文件的一些属性信息，称为“元信息”。这些元信息在上传时与文件一起存储，在下载时与文件一起返回。

文件元信息会附在HTTP请求头部，而HTTP协议规定请求头部不能包含复杂字符，所以元信息只能是简单的ASCII可见字符且不能包含换行。

> **说明** 所有元信息的总大小不能超过 8KB。

使用 `put` 、 `putStream` 和 `multipartUpload` 时，都可以通过指定meta参数来指定文件的元信息：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function put () {
  try {
    let result = await client.put('object-name', 'local-file', {
    meta: {
      year: 2016,
      people: 'mary'
    }
  });
  console.log(result);
  } catch (e) {
    console.log(e);
  }
}
put();
```

您还可以通过putMeta接口来更新文件元信息：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function putMeta () {
  try {
    let result = await client.putMeta('object-name', {
    meta: {
      year: 2015,
      people: 'mary'
    }
  });
  console.log(result);
  } catch (e) {
    console.log(e);
  }
}
putMeta();
```

4.8.4. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	public-read-write

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于设置指定文件的访问权限：

```
const oss = require('ali-oss');
const store = oss({
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<your access key>',
  accessKeySecret: '<your access secret>',
  bucket: '<your bucket name>',
  // 获取当前bucket所在的region。
  region: 'oss-cn-hangzhou'
});
  //设置指定文件的访问权限为公共读。
await store.putACL('<object name>', 'public-read');
```

设置文件访问权限更多详情，请参见PutObjectACL。

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
const oss = require('ali-oss');
const store = oss({
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<your access key>',
  accessKeySecret: '<your access secret>',
  bucket: '<your bucket name>',
  // 获取当前bucket所在的region。
  region: 'oss-cn-hangzhou'
});
  // 获取文件访问权限。
const result = await store.getACL('<object name>);
console.log(result.acl);
```

获取文件访问权限更多详情，请参见GetObjectACL。

4.8.5. 列举文件

本文介绍如何列举指定存储空间下（Bucket）的所有文件（Object）、指定个数的文件、指定前缀的文件等。

列举方法

您可以调用 `list` 或 `listV2` 方法，一次性列举某个Bucket下最多1000个Object。您可以通过指定参数实现多种列举功能，例如通过指定参数列举指定起始位置后的所有文件、列举指定目录下的文件和子目录、以及列举结果分页。这两个接口的主要区别如下：

- 使用 `list` 方法列举文件时，默认返回owner信息。
- 使用 `listV2` 方法列举文件时，需通过fetchOwner指定是否在返回结果中包含owner信息。

> **说明** 对于开启版本控制的Bucket，建议使用 `listV2` 接口列举文件。

以下分别介绍通过 `list` 以及 `listV2` 方法列举文件时涉及的参数说明。

- 通过list方法列举文件

  list涉及参数说明如下：

参数	类型	描述
prefix	string	列举符合特定前缀的文件。
delimiter	string	对文件名称进行分组的字符。
marker	string	列举文件名在marker之后的文件。
max-keys	number | string	指定最多返回的文件个数。
encoding-type	'url' | ''	对返回的内容进行编码并指定编码类型为URL。

- 通过listV2方法列举文件

  listV2涉及参数说明如下：

参数	类型	描述
prefix	string	列举符合特定前缀的文件。
continuation-token	string	从此token开始获取文件列表。
delimiter	string	对文件名称进行分组的字符。
max-keys	number | string	指定最多返回的文件个数。
start-after	string	设定从start-after之后按字母排序开始返回Object
fetch-owner	boolean	是否在返回结果中包含owner信息 。
encoding-type	'url' | ''	对返回的内容进行编码并指定编码类型为URL。

简单列举文件

以下代码用于列举指定Bucket的文件，默认列举100个文件。

- 通过list方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
  // 不带任何参数，默认最多返回100个文件。
  let result = await client.list();
  console.log(result);
}
list();
```

- 通过listV2方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
  // 不带任何参数，默认最多返回100个文件。
  let result = await client.listV2();
  console.log(result);
}
list();
```

列举指定个数的文件

以下代码用于通过max-keys参数列举Bucket下指定个数的文件。

- 通过list方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
  let result = await client.list({
     // 设置按字母排序最多返回前10个文件。
    "max-keys": 10
 });
  console.log(result);
}
list();
```

- 通过listV2方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
  let result = await client.listV2({
    // 设置按字母排序最多返回前10个文件。
   "max-keys": 10
 });
  console.log(result);
}
list();
```

列举指定前缀的文件

以下代码用于通过prefix参数列举Bucket中包含指定前缀的文件。

- 通过list方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
 let result = await client.list({
  // 列举10个文件。
  "max-keys": 10,
  // 列举文件名中包含前缀foo/的文件。
  prefix: 'foo/'
 });
 console.log(result);
}
list();
```

- 通过listV2方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
 let result = await client.listV2({
  // 列举10个文件。
  "max-keys": 10,
  // 列举文件名中包含前缀foo/的文件。
  prefix: 'foo/'
 });
 console.log(result);
}
list();
```

列举指定文件名后的文件

以下代码用于列举文件名称字典序排在指定字符串（marker或startAfter）之后的文件。

- 通过list方法

  参数marker代表文件名称。

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
// 列举字典序排在test之后的文件。默认列举100个文件。
const marker = 'test'
async function list () {
 let result = await client.list({
  marker
 });
 console.log(result);
}
list();
```

- 通过listV2方法

  startAfter代表文件名称。

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
 let result = await client.listV2({
  // 列举a/文件夹下名称在a/b之后的文件及子文件夹。
  delimiter: '/',
  prefix: 'a/',
  'start-after': 'a/b'
 });
 console.log(result.objects， result.prefixes);
}
list();
```

分页列举所有文件

以下代码用于分页列举指定Bucket下的所有文件。每页列举的文件个数通过max-keys指定。

- 通过list方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
let marker = null;
// 每页列举20个文件。
const maxKeys = 20;
async function list () {
 do {
  let result = await client.list({
      marker,
    'max-keys': maxKeys
    });
  marker = result.nextMarker;
     console.log(result);
 }while(marker)
}
list();
```

- 通过listV2方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
async function list () {
 let continuationToken = null;
 // 每页列举20个文件。
 const maxKeys = 20;
 do {
  let result = await client.listV2({
   'continuation-token': continuationToken,
   'max-keys': maxKeys
   });
  continuationToken = result.nextContinuationToken;
    console.log(result);
 }while(continuationToken)
}
list();
```

列举结果返回文件owner信息

以下代码用于列举结果中返回文件的owner信息：

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
// 默认列举的文件信息中不包含owner信息。如需包含owner信息，fetch-owner参数取值为true。
async function list () {
 let result = await client.listV2({
  'fetch-owner': true
 });
 console.log(result.objects);
}
list();
```

列举指定目录下的文件和子目录

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为objects。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

假设Bucket中有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

以下代码用于通过 `list` 或 `listV2` 的方法列举指定目录下的文件和子目录。

- 通过list方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
// 调用listDir函数，通过设置不同的文件前缀列举不同的目标文件。
async function listDir(dir) {
 const result = await client.list({
  prefix: dir,
  // 设置正斜线（/）为文件夹的分隔符。
  delimiter: '/'
 });
 result.prefixes.forEach(subDir => {
  console.log('SubDir: %s', subDir);
 });
 result.objects.forEach(obj => {
  console.log('Object: %s', obj.name);
 });
}
listDir('foo/');
// SubDir: foo/bar/
// SubDir: foo/hello/
// Object: foo/x
// Object: foo/y
listDir('foo/bar/');
// Object: foo/bar/a
// Object: foo/bar/b
listDir('foo/hello/C/');
// Object: foo/hello/C/1
// Object: foo/hello/C/2
// ...
// Object: foo/hello/C/9999
```

- 通过listV2方法

```
let OSS = require('ali-oss');
let client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 // 填写Bucket名称。
 bucket: '<Your bucket name>',
});
// 调用listV2Dir函数，通过设置不同的文件前缀列举不同的目标文件。
async function listV2Dir(dir) {
 const result = await client.listV2({
  prefix: dir,
  // 设置正斜线（/）为文件夹的分隔符。
  delimiter: '/'
 });
 result.prefixes.forEach(subDir => {
  console.log('SubDir: %s', subDir);
 });
 result.objects.forEach(obj => {
  console.log('Object: %s', obj.name);
 });
}
listDir('foo/');
// SubDir: foo/bar/
// SubDir: foo/hello/
// Object: foo/x
// Object: foo/y
listDir('foo/bar/');
// Object: foo/bar/a
// Object: foo/bar/b
listDir('foo/hello/C/');
// Object: foo/hello/C/1
// Object: foo/hello/C/2
// ...
// Object: foo/hello/C/9999
```

4.8.6. 拷贝文件

本文介绍如何拷贝文件（Object）。拷贝文件时，您可以根据实际设置目标文件的HTTP头以及自定义目标文件的元信息。

> **说明** 关于文件元信息的更多信息，请参见开发指南中的管理文件元信息。

以下代码用于在同一个Bucket中拷贝文件。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
 // 设置是否启用HTTPS。设置secure为true时，表示启用。
 //secure: true
})
// 拷贝同一个Bucket中的文件。
// 填写拷贝后和拷贝前的文件完整路径。文件完整路径中不能包含Bucket名称。
client.copy('dstexampleobject.txt', 'srcexampleobject.txt',
// 设置目标文件的HTTP头和自定义目标文件的元信息。
//{
  // 指定headers参数，设置目标文件的HTTP头。如果未指定headers参数，则目标文件与源文件的HTTP头相同，即拷贝源文件的HTTP头。
  //headers:{
    //'Cache-Control': 'no-cache',
  //},
  // 指定meta参数，自定义目标文件的元信息。如果未指定meta参数，目标文件与源文件的元信息相同，即拷贝源文件的元信息。
  //meta:{
    //location: 'hangzhou',
    //year: 2015,
    //people: 'mary'
  //}
//}
).then((res) => {
  console.log(res);
}).catch(e => {
 console.log(e);
})
```

以下代码用于在同一个Region下的两个不同Bucket中拷贝文件。

```
const OSS = require('ali-oss')
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写目标Bucket名称。
 bucket: 'dstexamplebucket',
});
async function copy () {
 try {
  // 拷贝同一个Region下的两个不同Bucket中的文件时，源文件的地址格式为'/bucketname/objectname'。
  const result = await client.copy('dstexampleobjectnew.txt', '/srcexamplebucket/srcexampleobject.txt',
  // 设置目标文件的HTTP头和自定义目标文件的元信息。
  //{
    // 指定headers参数，设置目标文件的HTTP头。如果未指定headers参数，则目标文件与源文件的HTTP头相同，即拷贝源文件的HTTP头。
    //headers:{
      //'Cache-Control': 'no-cache',
    //},
    // 指定meta参数，自定义目标文件的元信息。如果未指定meta参数，目标文件与源文件的元信息相同，即拷贝源文件的元信息。
    //meta:{
      //location: 'hangzhou',
      //year: 2015,
      //people: 'mary'
    //}
  //}
  );
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
copy()
```

4.8.7. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传、拷贝文件及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
const OSS = require("ali-oss")
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function put () {
  try {
    //object-name可以自定义为文件名（例如file.txt）或目录（例如abc/test/file.txt）的形式，实现将文件上传至当前Bucket或Bucket下的指定目录。
    //指定Put操作时是否覆盖同名Object。
    //不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
    //指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
    //指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
    let result = await client.put('object-name', 'local-file',{headers: { 'x-oss-forbid-overwrite': true }});
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
put();
```

简单上传的更多详情，请参见PutObject。

拷贝文件

- 拷贝小文件

  以下代码用于拷贝小文件时禁止覆盖同名文件：

```
const OSS = require("ali-oss")
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
//拷贝同一个Bucket内的文件。
//指定Copy操作时是否覆盖同名Object。
//不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
//指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
//指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
client
  .copy("test_copy", "test", { headers: { "x-oss-forbid-overwrite": true } })
  .then(res => {
    console.log(res);
  })
  .catch(e => {
    console.log(e);
  });
```

- 拷贝大文件

以下代码用于拷贝大文件（分片拷贝）时禁止覆盖同名文件：

```
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function put() {
  try {
    let result = await client.multipartUploadCopy(
      copyName,
      {
        sourceKey: objectkey, // 拷贝源文件名。
        sourceBucketName: bucket // 拷贝源bucket。
      },
      {
        //指定multipartUploadCopy操作时是否覆盖同名Object。
        //不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
        //指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
        //指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
        headers: { "x-oss-forbid-overwrite": true }
      }
    );
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
```

拷贝文件的更多详情，请参见CopyObject。

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
const OSS = require("ali-oss")
const client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
//指定multipartUpload操作时是否覆盖同名Object。
//不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
//指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
//指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
 client
  .multipartUpload('object-name', 'local-file', { headers: { "x-oss-forbid-overwrite": true } })
  .then(res => {
    console.log(res);
  })
  .catch(e => {
    console.log(e);
  });
```

分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

4.8.8. 删除文件

本文介绍如何单个或者批量删除文件（Object）。

> **警告** 文件一旦删除将无法恢复，请谨慎使用删除操作。

注意事项

删除文件时，您需要具有对Object所在Bucket的写权限。

单个删除文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
let OSS = require('ali-oss')
let client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
async function deleteFile () {
 try {
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  let result = await client.delete('exampleobject.txt');
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
deleteFile();
```

批量删除文件

批量删除文件时，您可以删除指定的多个文件或者指定前缀的文件。

- 删除指定的多个文件

  以下代码用于删除examplebucket中指定的多个文件。

  返回结果包括如下两种模式，默认返回模式为简单模式，请根据实际选择返回模式。

  - 详细模式（verbose）：返回所有删除的文件列表。
  - 简单模式（quiet）：只返回删除失败的文件列表。

```
let OSS = require('ali-oss')
let client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
async function deleteMulti () {
 try {
  // 填写需要删除的多个Object完整路径并设置返回模式为详细模式。Object完整路径中不能包含Bucket名称。
  //let result = await client.deleteMulti(['exampleobject-1', 'exampleobject-2', 'testfolder/sampleobject.txt']);
  //console.log(result);
  // 填写需要删除的多个Object完整路径并设置返回模式为简单模式。Object完整路径中不能包含Bucket名称。
  let result = await client.deleteMulti(['exampleobject-1', 'exampleobject-2', 'testfolder/sampleobject.txt'], {quiet: true});
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
deleteMulti();
```

- 删除指定前缀的文件

  以下代码用于删除examplebucket中以file为前缀的文件，返回结果中只能返回删除失败的文件列表。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourregion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
// 处理请求失败的情况，防止promise.all中断，并返回失败原因和失败文件名。
async function handleDel(name, options) {
 try {
  await client.delete(name);
 } catch (error) {
  error.failObjectName = name;
  return error;
 }
}
// 删除指定前缀的文件。
async function deletePrefix(prefix) {
 const list = await client.list({
  prefix: prefix,
 });
 list.objects = list.objects || [];
 const result = await Promise.all(list.objects.map((v) => handleDel(v.name)));
 console.log(result);
}
deletePrefix('file')
```

4.8.9. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何转换文件（Object）的存储类型。

有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

以下提供了详细的示例代码用于Object存储类型的相互转换。

- 以下代码用于将Object的存储类型从标准或低频访问转换为归档类型：

```
let OSS = require('ali-oss');
let client = new OSS({
  bucket: '<your bucket>',
  region: '<your region>',
  accessKeyId: '<your accessKeyId>',
  accessKeySecret: '<your accessKeySecret>'
})
let options = {
  headers:{'x-oss-storage-class':'Archive'}
}
client.copy('Objectname','Objectname',options).then((res) => {
  console.log(res);
}).catch(err => {
  console.log(err)
})
```

- 以下代码用于将Object的存储类型从归档转换为低频访问或标准类型：

```
let OSS = require('ali-oss');
let client = new OSS({
  bucket: '<your bucket>',
  region: '<your region>',
  accessKeyId: '<your accessKeyId>',
  accessKeySecret: '<your accessKeySecret>'
})
// 以下以转换为低频访问类型（IA）为例。
var options = {
  headers:{'x-oss-storage-class':'IA'}
}
client.copy('Objectname','Objectname',options).then((res) => {
  console.log(res);
}).catch(err => {
  console.log(err)
})
```

各种存储类型的存储费用介绍，请参见计量项和计费项的计量计费概述一节。

4.8.10. 解冻归档文件

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档文件。

解冻归档文件的完整示例请参考Git Hub。

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档文件：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
client.restore('objectName').then((res) => {
   console.log(res);
}).catch(err => {
   console.log(err);
})
```

归档存储类型的详细说明请参见存储类型介绍。归档类型涉及的各参数详细说明请参见API文档RestoreObject。

4.8.11. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
const oss = require('ali-oss');
const store = oss({
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<your access key>',
  accessKeySecret: '<your access secret>',
  bucket: '<your bucket name>',
  // 获取当前bucket所在的region。
  region: 'oss-cn-hangzhou'
});
const options = {
  storageClass: 'IA',
  meta: {
    uid: '1',
    slus: 'test.html'
  }
}
const result = await store.putSymlink('<object name>', '<target object mame>', options)
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
const oss = require('ali-oss');
const store = oss({
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId: '<your access key>',
  accessKeySecret: '<your access secret>',
  bucket: '<your bucket name>',
  // 获取当前bucket所在的region。
  region: 'oss-cn-hangzhou'
});
const result = await store.getSymlink('<object name>')
console.log(result.targetName)
```

获取软链接的详细信息请参见GetSymlink。

4.9. 版本控制

4.9.1. 管理版本控制

版本控制应用于存储空间（Bucket）内的所有文件（Object）。通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。版本控制的详情请参见开发指南的版本控制介绍。

> **说明** 版本控制目前支持6.8.0及以上版本SDK。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function putBucketVersioning() {
  // 设置存储空间版本控制状态为Enabled或Suspended。
  const status = 'Enabled'; // `Enabled` or `Suspended`
  const result = await client.putBucketVersioning('<Bucket Name>', status);
  console.log(result);
}
putBucketVersioning();
```

有关设置Bucket版本控制状态的更多信息，请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function getBucketVersioning() {
  // 获取存储空间版本控制状态信息。
  const result = await client.getBucketVersioning('Bucket Name');
  console.log(result.versionStatus);
}
getBucketVersioning();
```

有关获取Bucket版本控制状态的更多信息，请参见GetBucketVersioning。

列举Bucket中所有Object的版本信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function getBucketVersions() {
  // 列举包括删除标记（Delete Marker）在内的所有Object的版本信息。
  const result = await client.getBucketVersions();
  console.log(result.objects);
  console.log(result.deleteMarker);
}
getBucketVersions();
```

4.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本ID为“null”。

以下代码用于简单上传：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function put() {
  const result = await client.put('fileName', 'file');
  console.log(result.res.headers['x-oss-version-id']); // 查看此次上传object的版本ID。
}
put();
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

> **说明**
>
> - 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
> - 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
> - 不支持对当前版本为非Appendable类型的Object（包括Normal Object、Delete Marker等）执行AppendObject操作。

以下代码用于追加上传：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function append() {
  await client.append('<file name>', '<upload file>', {
    partSize: 100 * 1024
  });
}
append();
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用MultipartUpload方法来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function multipartUpload() {
  const result = await client.multipartUpload('<file name>', '<upload file>', {
    partSize: 100 * 1024
  });
  // 查看此次上传object的版本ID。
  console.log(result.res.headers['x-oss-version-id']);
}
multipartUpload();
```

分片上传的详细信息请参见CompleteMultipartUpload。

4.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function get() {
  const get = await client.get(name, {
    //查看下载文件的版本Id。
    versionId: 'versionId',
  });
  console.log(result.content);
}
get();
```

下载文件的详细信息请参见Get Object。

4.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的版本ID，此版本ID将会在响应header 的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为”null“的版本，且会覆盖原先versionId为”null“的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 指定拷贝Object的versionId。
const versionId = 'your versionId';
async function copy() {
  const result = await client.copy(target, origin, { versionId });
  console.log(result.res.headers);
}
copy();
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（multipartUploadCopy）的方法。

分片拷贝默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID： x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
// 指定拷贝Object的versionId。
versionId = 'your versionId';
async function multipartUploadCopy() {
  const result = await client.multipartUploadCopy(
    copyName,
    {
      sourceKey: name,
      sourceBucketName: bucket,
    },
    {
      versionId
    }
  );
  console.log(result);
}
multipartUploadCopy();
```

分片拷贝的详细信息请参见UploadPartCopy。

4.9.5. 列举文件

本文介绍如何在开启版本控制状态下列举存储空间下（Bucket）的所有文件（Object）、指定个数的文件、指定前缀的文件等。

列举Bucket中所有Object的信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息：

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "<yourAccessKeyId>";
const accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
const endpoint = "httpss://oss-cn-hangzhou.aliyuncs.com";
const bucket = "<yourBucketName>";
// 创建OSSClient实例。
const ossClient = new OSS({
  accessKeyId,
 accessKeySecret,
 endpoint,
 bucket
});
// 列举包括删除标记在内的所有Object的版本信息。
async function main () {
 let nextKeyMarker = null;
 let nextVersionMarker = null;
 let versionListing = null;
 do {
  versionListing = await client.getBucketVersions({
   keyMarker: nextKeyMarker,
   versionIdMarker: nextVersionMarker
  })
  versionListing.objects.forEach(o => {
   console.log(` ${o.name}, ${o.versionId}` )
  })
  versionListing.deleteMarker.forEach(o => {
   console.log(` ${o.name}, ${o.versionId}` )
  })
  nextKeyMarker = versionListing.NextKeyMarker;
  nextVersionMarker = versionListing.NextVersionIdMarker;
 } while (versionListing.isTruncated);
}
main()
```

列举指定前缀Object的版本信息

以下代码用于列举指定前缀Object的版本信息：

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "<yourAccessKeyId>";
const accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
const endpoint = "httpss://oss-cn-hangzhou.aliyuncs.com";
const bucket = "<yourBucketName>";
// 创建OSSClient实例。
const ossClient = new OSS({
  accessKeyId,
 accessKeySecret,
 endpoint,
 bucket
});
// 指定列举以"test-"为前缀的Object的版本信息。
async function main () {
 let nextKeyMarker = null;
 let nextVersionMarker = null;
 let versionListing = null;
 const prefix = 'test-'
 do {
  versionListing = await client.getBucketVersions({
   keyMarker: nextKeyMarker,
   versionIdMarker: nextVersionMarker,
   prefix
  })
  versionListing.objects.forEach(o => {
   console.log(` ${o.name}, ${o.versionId}` )
  })
  versionListing.deleteMarker.forEach(o => {
   console.log(` ${o.name}, ${o.versionId}` )
  })
  nextKeyMarker = versionListing.NextKeyMarker;
  nextVersionMarker = versionListing.NextVersionIdMarker;
 } while (versionListing.isTruncated);
}
main()
```

列举指定个数Object的版本信息

以下代码用于列举指定个数Object的版本信息：

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "<yourAccessKeyId>";
const accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
const endpoint = "httpss://oss-cn-hangzhou.aliyuncs.com";
const bucket = "<yourBucketName>";
// 创建OSSClient实例。
const ossClient = new OSS({
  accessKeyId,
 accessKeySecret,
 endpoint,
 bucket
});
// 指定最多列举100个Object的版本信息。
const versionListing = await client.getBucketVersions({
 'max-keys': 100
})
// 获取Object的版本信息。在未开启版本控制的情况下，VersionId为none。
versionListing.objects.forEach(o => {
 console.log(`${o.name}, ${o.versionId}`)
})
versionListing.deleteMarker.forEach(o => {
 console.log(`${o.name}, ${o.versionId}`)
})
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为Object。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

假设存储空间中包含*oss.jpg*、*fun/test.jpg*、*fun/movie/001.av*和*fun/movie/007.avi*4个文件，正斜线（/）作为文件夹的分隔符。以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举根目录下的Object的版本信息

  以下代码用于列举根目录下的Object的版本信息：

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "<yourAccessKeyId>";
const accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
const endpoint = "httpss://oss-cn-hangzhou.aliyuncs.com";
const bucket = "<yourBucketName>";
// 创建OSSClient实例。
const ossClient = new OSS({
 accessKeyId,
  accessKeySecret,
  endpoint,
  bucket
});
// 指定delimiter参数为正斜线（/），将会列举根目录下的Object的版本信息以及文件夹名称。
async function main () {
  let nextKeyMarker = null;
  let nextVersionMarker = null;
  let versionListing = null;
  do {
    versionListing = await client.getBucketVersions({
      keyMarker: nextKeyMarker,
      versionIdMarker: nextVersionMarker,
      delimiter: '/'
    })
    nextKeyMarker = versionListing.NextKeyMarker;
    nextVersionMarker = versionListing.NextVersionIdMarker;
  } while (versionListing.isTruncated);
}
main()
```

返回结果：

```
{
  res: {},
  objects: [
    {
      name: 'oss.jpg',
      versionId: 'xxx',
    }
  ],
  prefixes: ["fun/"],
  NextKeyMarker: null,
  NextVersionIdMarker: null,
  isTruncated: false
}
```

- 列举目录下的文件和子目录

  以下代码用于列举指定目录下的文件和子目录：

```
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
const accessKeyId = "<yourAccessKeyId>";
const accessKeySecret = "<yourAccessKeySecret>";
// Endpoint以杭州为例，其它Region请按实际情况填写。
const endpoint = "httpss://oss-cn-hangzhou.aliyuncs.com";
const bucket = "<yourBucketName>";
// 创建OSSClient实例。
const ossClient = new OSS({
 accessKeyId,
  accessKeySecret,
  endpoint,
  bucket
});
// 设置prefix参数来获取fun目录下的所有文件与文件夹，同时设置delimiter参数为正斜线（/）作为文件夹的分隔符。
async function main () {
  let nextKeyMarker = null;
  let nextVersionMarker = null;
  let versionListing = null;
  let prefix = 'fun/'
  do {
    versionListing = await client.getBucketVersions({
      keyMarker: nextKeyMarker,
      versionIdMarker: nextVersionMarker,
      prefix,
      delimiter: '/'
    })
    nextKeyMarker = versionListing.NextKeyMarker;
    nextVersionMarker = versionListing.NextVersionIdMarker;
  } while (versionListing.isTruncated);
}
main()
```

返回结果：

```
{
  res: {},
  objects: [
    {
      name: 'test.jpg',
      versionId: 'xxx',
    }
  ],
  prefixes: ["fun/movie/"],
  NextKeyMarker: null,
  NextVersionIdMarker: null,
  isTruncated: false
}
```

4.9.6. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

  如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

  `x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object进行永久删除：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
// 删除指定versionId的Object。
const versionId = 'your versionId';
async function deleteVersionObject() {
  const result = await client.delete(obj.name, {
    versionId
  });
}
deleteVersionObject();
```

- 临时删除

  以下代码用于不指定versionId对Object进行临时删除：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
// 不指定versionId对Object进行临时删除，此操作会为Object添加删除标记。
async function deleteObject() {
  const result = await client.delete(obj.name);
}
deleteObject();
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于指定versionId对多个Object及删除标记进行永久删除：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
// 删除多个指定versionId的Object，或删除指定versionId的删除标记关联的Object。
const names = [
  { key: 'key1.js', versionId: 'versionId1' },
  { key: 'key2.js', versionId: 'versionId2' }
];
async function deleteMulti() {
  const result = await client.deleteMulti(names);
  console.log(result);
}
deleteMulti();
```

  删除多个文件的详细信息请参见DeleteMultipleObjects。

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除。

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
const names = ['key1.js', 'key2.js'];
async function deleteMulti() {
  // 不指定versionId删除多个Object。
  const result = await client.deleteMulti(names);
  console.log(result);
}
deleteMulti();
```

删除指定前缀的文件

以下代码用于删除指定前缀的文件：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function deleteMutiPrefix(prefix) {
  // 获取指定前缀的Object的versionId信息。
  const list = client.getBucketVersions({
    prefix: prefix,
  });
  list.objects = list.objects || [];
  for (let i = 0; i < result.objects.length; i++) {
    const obj = result.objects[i];
    // 删除指定前缀的Object。
    const versionId = obj.versionId;
    await client.delete(obj.name, {
      versionId
    });
  }
}
```

4.9.7. 解冻文件

在受版本控制的存储空间（Bucket）中，文件（Object）的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
const name = '<your objectName>'
// 解冻指定versionId的object。
const versionId = 'your versionId'
async function restore() {
  const result = await client.restore(name, {
    versionId
  });
  console.log(result);
}
restore();
```

解冻文件的详细信息请参见RestoreObject。

4.9.8. 获取文件元信息

默认情况下，在已开启版本控制的存储空间（Bucket）中调用HeadObject接口只能获取文件（Object）当前版本的元信息（meta）。通过指定文件的版本ID（versionId），您可以获取文件指定版本的元信息。

> **说明** 如果文件的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回文件指定版本的元信息。获取文件versionId的具体操作，请参见列举文件。

以下代码用于获取目标存储空间examplebucket中exampledir目录下exampleobject.txt文件指定版本的元信息。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称，例如examplebucket。
 bucket: 'examplebucket',
});
async function headInfo() {
 // 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称。
 const name = 'exampledir/exampleobject.txt'
 // 填写文件的版本ID，获取文件指定版本的元信息。
 const versionId = 'CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****'
 const options = {
  versionId
 };
 const result = client.head(name, options);
 console.log(result);
}
headInfo();
```

关于获取文件元信息的更多信息，请参见HeadObject。

4.9.9. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function putACL() {
  const name = '<your objectName>'
  const acl = '<your acl>'
  const versionId = 'your versionId' // 设置指定versionId的Object的访问权限（ACL）。
 const options = {
   versionId
  };
  const result = client.putACL(name, acl, options);
  console.log(result);
}
putACL();
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
async function getACL() {
  const name = '<your objectName>'
  const versionId = 'your versionId' //查看此次执行获取访问权限操作的object的versionId。
  const options = {
   versionId
  };
  const result = client.getACL(name, options);
  console.log(result);
}
getACL();
```

获取文件访问权限的详细信息请参见GetObjectACL。

4.9.10. 管理软链接

本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

软链接本身也可以有多个版本，每个不同的版本可以指向不同的Target Object，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。您可以通过创建软链接指向Target Object的当前版本。

> **说明** 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

以下代码用于创建软链接：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
async function putSymlink() {
  // 创建软链接。
 const name = '<your objectName>'
 const targetName = '<your target objectName>'
 const result = await client.putSymlink(name, targetName);
 console.log(result.res.headers['x-oss-version-id']);
}
putSymlink();
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 //阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>'
});
async function getSymlink() {
 const name = '<your objectName>'
 const versionId = '<your versionId>'
 // 获取指定versionId的软链接的内容。
 const result = await client.getSymlink(name, {
  versionId
 });
 console.log(result);
}
getSymlink();
```

获取软链接的详细信息请参见GetSymlink。

4.10. 对象标签

4.10.1. 设置对象标签

OSS支持使用对象标签（Object Tagging）对存储空间（Bucket）中的文件（Object）进行分类，您可以针对相同标签的Object设置生命周期规则、访问权限等。

背景信息

设置对象标签时，请注意以下事项：

- 您可以在上传Object时设置对象标签，也可以对已上传Object设置对象标签。设置对象标签时，如果对象已有标签，则覆盖原标签。关于设置对象标签的更多信息，请参见PutObjectTagging。
- 设置对象标签时，您要有PutObjectTagging权限。
- 更改标签时不会更新Object的Last-Modified时间。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/

  > **说明** 通过HTTP header的方式设置标签且标签中包含任意字符时，您可以对标签的Key和Value做URL编码。

对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见对象标签。

上传Object时添加对象标签

- 简单上传时添加对象标签

  以下代码用于简单上传时（即通过PutObject方法）添加对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B',
}
client.put('fileName', 'file', {
  headers
})
```

- 分片上传时添加对象标签

  以下代码用于分片上传时（即通过multipartUpload方法）添加对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B',
}
async function setTag() {
  await store.multipartUpload('<file name>', '<upload file>', {
    partSize: 100 * 1024,
    headers
  });
  const tag = await store.getObjectTagging(name);
  console.log(tag);
}
setTag()
```

- 追加上传时添加对象标签

  以下代码用于追加上传时（即通过AppendObject方法）添加对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B'
}
// 追加上传文件，append接口指定header时，将会为文件设置标签。
// 只有第一次调用append接口设置的标签才会生效，后续再次调用append设置的标签不生效。
async function setTag() {
  await store.append('<file name>', '<upload file>', {
    partSize: 100 * 1024,
    headers
  });
  const tag = await store.getObjectTagging(name);
  console.log(tag);
}
setTag()
```

- 断点续传上传时添加对象标签

  以下代码用于断点续传上传时（即通过multipartUpload方法）添加对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B'
}
// 设置断点信息。
const checkponit = {
  file,
  name,
  fileSize,
  partSize,
  uploadId,
  doneParts: []
}
async function setTag() {
  await store.multipartUpload('<file name>', '<upload file>', {
    checkponit,
    headers
  });
  const tag = await store.getObjectTagging(name);
  console.log(tag);
}
setTag()
```

对已上传Object添加或更改对象标签

以下代码用于对已上传Object添加或更改对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
async function putObjectTagging(objectName, tag) {
  try {
    let result = await client.putObjectTagging(objectName, tag);
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
const tag = { a: '1', b: '2' };
putObjectTagging('objectName', tag)
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object、及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 以下代码用于简单拷贝1 GB以下的Object时设置对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B',
  // 指定如何设置目标Object的对象标签。取值为Copy或Replace。其中Copy为默认值，表示复制源Object的对象标签到目标Object。Replace表示忽略源Object的对象标签，直接采用请求中指定的对象标签。
  'x-oss-tagging-directive': 'Replace'
}
async function setTag() {
  const targetObjectName = `${prefix}ali-sdk/oss/copy-tag.js`;
  const result = await store.copy(targetObjectName, sourceObjectName, {
    headers
  });
  const tag = await store.getObjectTagging(targetObjectName)
  console.log(tag)
}
setTag()
```

- 以下代码用于分片拷贝1 GB以上的Object时设置对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  //region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B',
}
async function setTag() {
  await client.multipartUploadCopy(copyName, {
    sourceKey: name,
    sourceBucketName: bucket
  }, {
    partSize: 256 * 1024,
    headers
  });
  const tag = await store.getObjectTagging(targetObjectName)
  console.log(tag)
}
setTag()
```

为软链接文件设置标签

以下代码用于为软链接文件设置标签：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  region: '<Your Region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置请求头信息。
const headers = {
  'x-oss-tagging': 'TagA=A&TagB=B'
}
async function setTag() {
  await store.putSymlink(name, targetName, {
    storageClass: 'IA',
    meta: {
      uid: '1',
      slus: 'test.html'
    },
    headers
  });
  const tag = await store.getObjectTagging(targetObjectName)
  console.log(tag)
}
setTag()
```

4.10.2. 获取对象标签

设置对象标签后，您可以根据需要获取Object的标签信息。当存储空间（Bucket）已开启版本控制时，OSS默认只获取Object当前版本的标签信息，您可以通过指定Object的版本ID（versionId）来获取Object指定版本的标签信息。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于获取对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 获取object的标签信息。
async function getObjectTagging(objectName) {
 try {
  let result = await client.getObjectTagging(objectName);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
getObjectTagging('objectName')
```

获取对象标签详情请参考GetObjectTagging。

4.10.3. 删除对象标签

您可以根据需要删除不需要的标签信息。当存储空间（Bucket）已开启版本控制时，OSS默认只删除Object当前版本的标签信息，您可以通过指定Object的版本ID（versionId）来删除Object指定版本标签信息。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于删除对象标签：

```
const OSS = require('ali-oss')
const client = new OSS({
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 删除object的标签信息。
async function deleteObjectTagging(objectName) {
 try {
  let result = await client.deleteObjectTagging(objectName);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
deleteObjectTagging('objectName')
```

删除对象标签详情请参考DeleteObjectTagging。

4.10.4. 对象标签和生命周期管理

在生命周期规则配置中，您可以指定生命周期规则生效的条件。生命周期规则可针对前缀（Prefix）或对象标签（Tag）生效，您也可以同时指定两者作为条件。

> **说明** 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于生命周期规则中添加标签匹配规则：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 设置匹配的标签。
const tag = [{
  key: 'key1',
  value: 'value1'
},{
  key: 'key2',
  value: 'value2'
}]
client.putBucketLifecycle('<yourBucketName>', [{
  // 指定生命周期规则id为rule1。
  id: 'rule1',
  // 将规则应用于前缀为“one”的object中。
  prefix: 'one',
  status: 'Enabled',
  expiration: {
  // 指定生命周期规则在object最后更新过后60天过期。
    days: 60
  },
  // 指定生命周期规则在object最后更新过后10天转为IA存储类型。
  transition: {
    days: 10,
    StorageClass: 'IA'
  },
  tag
}]);
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
try {
  const result = client.getBucketLifecycle('<yourBucketName>')
  // 查看生命周期规则中匹配的标签信息。
  result.rules.map(rule => {
    if (rule.tag) {
      rule.tag.map(_ => {
        console.log(` key: ${_.key}, value: ${_.value}` )
      })
    }
  })
} catch (error) {
  console.log(error)
}
```

4.11. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

简单上传下载限速

以下代码用于简单上传、下载文件时设置单链接限速：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 通过请求头设置限速。
const headers = {
 'x-oss-traffic-limit': 8 * 1024 * 100 // 设置限速，最小100KB/s。
}
// 限速上传。
async function put() {
  const result = await client.putStream('<file name>', '<file stream>', {
    headers,
    timeout: 60000 // 默认超时时长为60000ms。超时直接报错，限速上传时注意修改超时时长。
  })
  console.log(result)
}
put()
// 限速下载。
async function get() {
  const result = await client.get('<file name>', {
    headers,
    timeout: 60000 // 默认超时时长为60000ms。超时直接报错，限速下载时注意修改超时时长。
  })
  console.log(result)
}
get()
```

使用签名URL方式上传下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
const OSS = require('ali-oss')
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
// 通过URL Query方式限速上传。
async function putByQuery() {
  const url = client.signatureUrl('<file name>', {
    trafficLimit: 8 * 1024 * 100, // 设置限速，最小100KB/s。
    method: 'PUT' // 设置PUT请求方法。
  })
  const result = await client.urllib.request(url, {
    method: 'PUT',
    stream: '<file stream>'
    timeout: 60000, // 默认超时时长为60000ms。限速后注意设置超时时长，否则请求失败。
  });
  console.log(result)
}
putByQuery()
// 通过URL Query方式限速下载。
async function getByQuery() {
  const url = client.signatureUrl('<file name>', {
    trafficLimit: 8 * 1024 * 100, // 设置限速，最小100KB/s。
  })
  const result = await client.urllib.request(url, {
    timeout: 60000, // 默认超时时长为60000ms。限速后注意设置超时时长，否则请求失败。
  });
  console.log(result)
}
getByQuery()
```

4.12. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

  上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

> **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

> **注意**
>
> - 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
> - 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
> - 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
const OSS = require("ali-oss");
const store = new OSS({
 region: "<yourRegion>",
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: "<yourAccessKeyId>",
 accessKeySecret: "<yourAccessKeySecret>",
 bucket: "<yourBucketName>"
});
async function putBucketEncryption() {
 try {
  // 配置Bucket加密方式。
  let result = await store.putBucketEncryption("bucket-name", {
   SSEAlgorithm: "AES256", // 此处以设置AES256加密为例。若使用KMS加密，需添加KMSMasterKeyID属性。
   // KMSMasterKeyID：“<yourKMSMasterKeyId>”，设置KMS密钥ID，加密方式为KMS可设置此项。当SSEAlgorithm值为KMS，且使用指定的密钥加密时，需输入密钥ID。其他情况下，必须为空。
  });
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
putBucketEncryption();
```

有关配置Bucket加密规则的更多信息，请参见Put Bucket Encryption。

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
const OSS = require("ali-oss");
const store = new OSS({
  region: "<yourRegion>",
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  accessKeyId: "<yourAccessKeyId>",
  accessKeySecret: "<yourAccessKeySecret>",
  bucket: "<yourBucketName>"
});
async function getBucketEncryption() {
  try {
    let result = await store.getBucketEncryption("bucket-name");
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
getBucketEncryption();
```

有关获取Bucket加密配置的更多信息，请参见GetBucketEncryption。

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
const OSS = require("ali-oss");
const store = new OSS({
  region: "<yourRegion>",
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  accessKeyId: "<yourAccessKeyId>",
  accessKeySecret: "<yourAccessKeySecret>",
  bucket: "<yourBucketName>"
});
async function deleteBucketEncryption() {
  try {
    // 删除Bucket的加密配置。
    let result = await store.deleteBucketEncryption("bucket-name");
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
deleteBucketEncryption();
```

有关删除Bucket加密配置的更多信息，请参见DeleteBucketEncryption。

4.13. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

通过STS服务生成临时访问凭证，并在客户端使用临时访问凭证初始化OSSClient，用于临时授权访问OSS资源。示例代码如下：

```
// 通过STS服务生成临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。
const app = express();
const { STS } = require('ali-oss');
app.get('/sts', (req, res) => {
 let sts = new STS({
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret'
});
  sts.assumeRole('roleArn', 'policy', 'expiration', 'sessionName').then((result) => {
    console.log(result);
    res.set('Access-Control-Allow-Origin', '*');
    res.set('Access-Control-Allow-METHOD', 'GET');
    res.json({
      AccessKeyId: result.credentials.AccessKeyId,
      AccessKeySecret: result.credentials.AccessKeySecret,
      SecurityToken: result.credentials.SecurityToken,
      Expiration: result.credentials.Expiration
    });
  }).catch((err) => {
    console.log(err);
    res.status(400).json(err.message);
  });
});
// 在客户端使用临时访问凭证初始化OSS客户端，用于临时授权访问OSS资源。
const fetch = require('node-fetch');
const token = await fetch('https:/xxx/sts');
const client = new OSS({
    // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
    region: 'yourRegion',
    accessKeyId: token.AccessKeyId,
    accessKeySecret: token.AccessKeySecret,
    stsToken: token.SecurityToken,
    // 填写Bucket名称。
    bucket: 'examplebucket',
    // 刷新临时访问凭证。
    refreshSTSToken: async () => {
      const refreshToken = await fetch('http://127.0.0.1/sts');
      return {
        accessKeyId: refreshToken.AccessKeyId,
        accessKeySecret: refreshToken.AccessKeySecret,
        stsToken: refreshToken.SecurityToken,
      }
    }
});
```

STS相关参数说明请参见下表。更多信息，请参见AssumeRole。

参数	类型	是否必选	示例值	描述
roleArn	String	是	acs:ram::1746495********:role/osssts test	指定角色的ARN。格式：`acs:ram::$accountID:role/$roleName`。其中$accountID为阿里云账号ID，$roleName为RAM角色名称。 您可以登录RAM控制台，在**RAM角色管理**界面，搜索创建的RAM角色后，单击RAM角色名称，在RAM角色详情界面查看和复制角色的ARN信息。
policy	String	否	{"Version": "1", "Statement": [{"Action": ["oss:GetObject"], "Effect": "Allow", "Resource": ["acs:oss:*:*:examplebucket/*"]}]}	权限策略。 生成STS临时访问凭证时可以指定一个额外的权限策略，以进一步限制STS临时访问凭证的权限。如果不指定则返回的STS临时访问凭证拥有指定角色的所有权限。 长度为1~1024个字符。
expiration	Number	否	3600	临时访问凭证的过期时间，单位为秒，默认值为3600秒。 过期时间最小值为900秒，最大值为MaxSessionDuration设置的时间。 **说明** 您可以通过CreateRole或UpdateRole接口设置角色最大会话时间MaxSessionDuration。更多信息，请参见CreateRole或UpdateRole。
sessionName	String	是	Alice	用户自定义参数。此参数用来区分不同的令牌，可用于用户级别的访问审计。 长度为2~32个字符，可包含英文字母、数字、英文句点（.）、at（@）、短划线（-）和下划线（_）。

使用签名URL进行临时授权

以下介绍使用签名URL进行临时授权的常见示例。

- 生成签名URL

  您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

- 使用签名URL上传和下载文件

  说明 name {String}表示存放在OSS的Object名称，[expires] {Number}表示URL过期时间，默认值为1800秒。其他参数相关说明，请参见 Github。

以下代码用于生成对应签名URL来上传和下载文件。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket'
});
// 获取下载exampleobject.txt文件的签名URL，使用浏览器访问时默认直接预览要下载的文件。
// 填写不包含Bucket名称在内的Object完整路径。
const url = client.signatureUrl('exampleobject.txt');
console.log(url);
// 获取下载exampleobject.txt文件的签名URL，配置响应头实现浏览器访问时自动下载文件，并自定义下载后的文件名称。
const filename = 'ossdemo.txt' // 自定义下载后的文件名称。
const response = {
 'content-disposition': `attachment; filename=${encodeURIComponent(filename)}`
}
// 填写不包含Bucket名称在内的Object完整路径。
const url = client.signatureUrl('exampleobject.txt', { response });
console.log(url);
// 获取上传exampleobject.txt文件的签名URL，并设置过期时间。
// 填写不包含Bucket名称在内的Object完整路径。
const url = client.signatureUrl('exampleobject.txt', {
 expires: 3600, // 设置过期时间，默认值为1800秒。
 method: 'PUT'  // 设置请求方式为PUT。默认请求方式为GET。
});
console.log(url);
// 获取上传exampleobject.txt文件的签名URL，并设置Content-Type。
// 填写不包含Bucket名称在内的Object完整路径。
const url = client.signatureUrl('exampleobject.txt', {
 expires: 3600,
 method: 'PUT',
 'Content-Type': 'text/plain; charset=UTF-8',
});
console.log(url);
```

- 生成带图片处理参数的签名URL

  以下代码用于生成带图片处理参数的签名URL。

```
let OSS = require('ali-oss');
let store = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 填写Bucket名称。
 bucket: 'examplebucket'
})
// 获取处理exampleobject.png图片的签名URL。
// 填写不包含Bucket名称在内的Object完整路径。
const url = store.signatureUrl('exampleobject.png', {
 process: 'image/resize,w_200' // 设置图片处理参数。
});
console.log(url);
// 获取处理exampleobject.png图片的签名URL，并设置过期时间。
// 填写不包含Bucket名称在内的Object完整路径。
const url = store.signatureUrl('exampleobject.png', {
 expires: 3600, // 设置过期时间，默认值为1800秒。
 process: 'image/resize,w_200' // 设置图片处理参数。
});
console.log(url);
```

4.14. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理使用

图片处理使用标准的HTTP GET请求。您可以在URL的QueryString中设置处理参数。

如果图片文件的访问权限为私有，必须通过授权才能进行访问。

- 匿名访问

  如果图片文件（Object）的访问权限是 `公共读` ，如下表所示的权限，则可以匿名访问图片服务。

Bucket权限	Object权限
公共读私有写（public-read）或公共读写（public-read-write）	默认（default）
任意权限	公共读私有写（public-read）或公共读写（public-read-write）

  您可以通过如下格式的三级域名匿名访问处理后的图片：

```
http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=image/<yourAction>,<yourParamValue>
```

参数	描述
bucket	存储空间名称
endpoint	存储空间所在地域的访问域名
object	图片文件名称
image	图片处理的保留标志符
action	对图片做的操作，如缩放、裁剪、旋转等
param	对图片做的操作所对应的参数

- 基础操作

  例如，将图缩略成宽度为100，高度按比例处理：

  ```
  http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
  ```

- 自定义样式

  使用如下格式的三级域名匿名访问图片处理：

  ```
  http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=style/<yourStyleName>
  ```

  - style：自定义样式的保留标志符。
  - yourStyleName：自定义样式名称，即通过控制台自定义样式的规则名称。

  例如：

  ```
  http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=style/oss-pic-style-w-100
  ```

- 级联处理

  通过级联处理，可以对一张图片顺序进行多个操作，格式如下：

  ```
  http://<yourBucketName>.<yourEndpoint>/<yourObjectName>?x-oss-process=image/<yourAction1>,<yourParamValue1>/<yourAction2>,<yourParamValue2>/...
  ```

  例如：

  ```
  http://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100/rotate,90
  ```

- 支持HTTPS访问

  图片服务支持HTTPS访问，例如：

  ```
  https://image-demo.oss-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_100
  ```

- 授权访问

  对私有权限的文件（Object），如下表所示的权限，必须通过授权才能访问图片服务。

Bucket权限	Object权限
私有（private）	默认权限（default）
任意权限	私有（private）

以下代码用于生成带签名的图片处理URL：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 过期时间10分钟, 图片处理式样"image/resize,w_300"
let signUrl = client.signatureUrl('example.jpg', {expires: 600, 'process' : 'image/resize,w_300'});
console.log("signUrl="+signUrl);
```

说明

- 授权访问支持自定义样式、 HTTPS、 以及级联处理。
- 指定过期时间expires的单位是秒。

- SDK访问

  对于任意权限的图片文件，都可以直接使用SDK访问和处理。

  SDK处理图片文件支持自定义样式、HTTPS和级联处理。

  - 基础操作

    以下代码展示了图片处理的基础操作：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 缩放
async function scale () {
 try {
  let result = await client.get('example.jpg', './example-resize.jpg', { process: 'image/resize,m_fixed,w_100,h_100'});
 } catch (e) {
  console.log(e);
 }
```

```
}
// 裁剪
async function cut () {
 try {
   let result = await client.get('example.jpg', './example-crop.jpg', { process: 'image/crop,w_100,h_100,x_
100,y_100,r_1'});
 } catch (e) {
  console.log(e)
 }
}
// 旋转
async function rotate () {
 try {
  let result = await client.get('example.jpg', './example-rotate.jpg', { process: 'image/rotate,90'});
 } catch (e) {
  console.log(e);
 }
}
// 锐化
async function sharpen () {
 try {
  let result = yield client.get('example.jpg', './example-sharpen.jpg', { process: 'image/sharpen,100'});
 } catch (e) {
  console.log(e);
 }
}
// 水印
async function watermark () {
 try {
  let result = yield client.get('example.jpg', './example-watermark.jpg', { process: 'image/watermark,text
_SGVsbG8g5Zu-54mH5pyN5YqhIQ'});
 } catch (e) {
  console.log(e);
 }
}
// 格式转换
async function format () {
 try {
  let result = await client.get('example.jpg', './example-format.jpg', { process: 'image/format,png'});
 } catch (e) {
  console.log(e);
 }
}
// 图片信息
async function info () {
 try {
  let result = await client.get('example.jpg', './example-info.txt', {process: 'image/info'});
 } catch (e) {
  console.log(e);
 }
}
```

- 自定义样式

> **说明** 需要事先到OSS控制台添加自定义式样example-style。

以下代码用于自定义图片样式：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 自定义样式
async function style () {
 try {
  let result = await client.get('example.jpg', './example-custom-style.jpg', {process: 'style/example-style"'});
 } catch (e) {
  console.log(e);
 }
}
style();
```

- 级联处理

以下代码用于级联处理图片：

```
const OSS = require('ali-oss');
const client = new OSS({
 bucket: '<Your BucketName>',
 // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
 region: '<Your Region>',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
});
// 级联处理
async function cascade () {
 try {
  let result = await client.get('example.jpg', './example-cascade.jpg', {process: 'image/resize,m_fixed,w_100,h_100/rotate,90'});
 } catch (e) {
  console.log(e);
 }
}
cascade();
```

图片处理持久化

以下代码用于图片处理持久化：

```
const OSS = require('ali-oss');
const client = new OSS({
  bucket: '<Your BucketName>',
  // region以杭州为例（oss-cn-hangzhou），其他region按实际情况填写。
  region: '<Your Region>',
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
});
const sourceImage = 'sourceObject.png';
const targetImage = 'targetObject.jpg';
async function processImage(processStr, targetBucket) {
  const result = await client.processObjectSave(
    sourceImage,
    targetImage,
    processStr,
    targetBucket
  );
  console.log(result.res.status);
}
// 图片处理持久化：缩放
processImage("image/resize,m_fixed,w_100,h_100")
// 图片处理持久化：裁剪
processImage("image/crop,w_100,h_100,x_100,y_100,r_1")
// 图片处理持久化：旋转
processImage("image/rotate,90")
// 图片处理持久化：锐化
processImage("image/sharpen,100")
// 图片处理持久化：水印
processImage("image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ")
// 图片处理持久化：格式转换
processImage("image/format,jpg")
// 图片处理持久化：格式转换，并设置保存图片处理持久化结果的目标bucket。
processImage("image/format,jpg", "<target bucket>")
```

图片处理工具

可视化图片处理工具 ImageStyleViewer，可以直观的看到OSS图片处理的结果。

4.15. 异常处理

使用Node.js SDK时如果请求出错，会有相应的异常抛出，同时在log（默认为程序运行目录下oss_sdk.log）中也会记录详细的出错信息。

ClientError

ClientError指SDK内部出现的异常，例如参数设置错误、断点续传上传或断点续传下载过程中出现的文件被修改的错误。

RequestError

当网络出现中断或者异常时，Node.js SDK会抛出RequestError。出现此错误时，请检查网络连通性并确保网络正常后再重试。

ServerError

ServerError指服务器端错误，它来自于对服务器错误信息的解析。ServerError有以下几个属性：

- status：出错请求的HTTP状态码。
- code：OSS的错误码。
- message：OSS的错误信息。
- requestId：标识该次请求的UUID。当您无法解决问题时，可以凭requestId来寻求OSS开发工程师的帮助。

OSS中常见的错误信息请参见OSS错误响应。

调试

当您使用Node.js遇到客户端或者服务端错误时，您可以通过设置 `DEBUG` 环境变量来开启调试。

```
DEBUG=ali-oss node app.js
```

在浏览器环境中，您可以通过在console中设置 `localStorage.debug` 变量来开启调试。

```
localStorage.debug = 'ali-oss'
```

4.16. 常见问题

本文介绍使用 OSS Node.js SDK 的常见问题及解决方法。

如何进行 HTTPS 访问

初始化 SDK 时，只需要指定 secure 的值为 true，则默认进行 HTTPS 访问。

如何获取上传进度

使用分片上传时，可通过progress参数获取上传进度。

如何获取下载进度

Node.js SDK 中可根据下载流的大小来计算进度。

如何上传base64编码的图片

将 base64 内容转换成 File 对象，再调用接口上传至 OSS 服务器。

```
function dataURLtoFile(dataurl, filename) {
  let arr = dataurl.split(','), mime = arr[0].match(/:(.*?);/)[1],
    bstr = atob(arr[1]), n = bstr.length, u8arr = new Uint8Array(n);
  while(n--){
    u8arr[n] = bstr.charCodeAt(n);
  }
  return new File([u8arr], filename, {type:mime});
}
let file = dataURLtoFile('<base64 content>', '');
client.multipartUpload('<oss file name>', file).then( (res)=> {
  console.log(res)
}).catch((err) => {
  console.log(err)
});
```

如何上传文件到指定目录

给要上传的 object 名称前加指定目录前缀即可，可参考OSS 和文件系统对比。

```
let OSS = require('ali-oss')
let client = new OSS({
 region: '<Your region>',
 accessKeyId: '<Your AccessKeyId>',
 accessKeySecret: '<Your AccessKeySecret>',
 bucket: '<Your bucket name>',
});
client.multipartUpload('base-dir/' +'object-name', 'local-file', {
   progress: async function (p) {
    console.log('Progress: ' + p);
   }
 });
 console.log(result);
}).then((res) => {
 console.log(res)
}). catch((err) => {
 console.log(err);
});
```

如何获取 Object 的签名 URL

可调用 `signatureUrl` 方法，获取下载地址，可查看 Github 中 signatureUrl 部分。

常见错误参考

- SDK 开启异常日志
- OSS 常见错误

5.Python

5.1. 前言

本文介绍对象存储OSS的Python SDK各种使用场景下的示例代码。

源码地址

请访问GitHub获取源码。

示例代码

OSS Python SDK提供丰富的示例代码，方便您参考或直接使用。示例包括以下内容：

示例文件	示例内容
object_basic.py	快速入门，包括创建存储空间、上传、下载、列举、删除文件等
object_extra.py	上传文件和管理文件，包括设置自定义元信息、拷贝文件、追加上传文件等
upload.py	上传文件，包括断点续传上传、分片上传等
download.py	下载文件，包括流式下载、范围下载、断点续传下载等
object_check.py	上传和下载时数据校验的用法，包括MD5和CRC
object_progress.py	上传进度条和下载进度条
object_callback.py	上传文件中的上传回调
object_post.py	表单上传的相关操作
sts.py	STS的用法，包括角色扮演获取临时用户的密钥，并使用临时用户的密钥访问OSS
live_channel.py	LiveChannel的相关操作
image.py	图片处理的相关操作
bucket.py	存储空间基本操作，包括创建、删除、列举存储空间，以及设置静态网站托管，设置生命周期规则等
object_request_payment.py	存储空间的请求者付费模式
traffic_limit.py	设置上传、下载文件时的单链接限速

5.2. 安装

本文介绍如何安装Python SDK。

环境准备

OSS Python SDK适用于Python 2.6、2.7、3.3、3.4、3.5、3.6、3.7、3.8版本。

- 安装环境

  根据Python官网的引导安装合适的Python版本。

- 查看版本

  执行命令 `python -V` 查看Python版本。

  Windows环境中，如果提示“不是内部或外部命令”，请在环境变量中编辑 `Path` ，增加Python和pip的安装路径。pip的安装路径一般为Python所在目录的Scripts文件夹。

  > **说明** 您可能需要重启电脑使环境变量生效。

下载SDK

- 通过GitHub下载
- 历史版本下载

更多信息请参见OSS API文档。

安装python-devel

由于SDK需要crcmod库计算CRC校验码，而crcmod依赖Python.h文件，如果系统缺少这个头文件，安装SDK不会失败，但crcmod的C扩展模式安装会失败，因此导致上传、下载等操作效率非常低下。如果python-devel包不存在，则首先要安装这个包。

- 对于Windows和Mac OS X系统，由于安装Python的时候会将Python依赖的头文件一并安装，因此您无需安装python-devel。
- 对于CentOS、RHEL、Fedora系统，请执行以下命令安装python-devel：

```
yum install python-devel
```

- 对于Debian，Ubuntu系统，请执行以下命令安装python-devel：

```
apt-get install python-dev
```

安装SDK

- 通过pip安装

  执行命令如下：

```
pip install oss2
```

  > **说明** 如果您的环境尚未安装pip，请参见pip官网安装。

- 通过源码安装

  从GitHub下载相应版本的SDK包，解压后进入目录，确认目录下有setup.py文件，然后执行命令如下：

```
python setup.py install
```

验证

- 验证SDK版本
  i. 在命令行输入 `python` 并回车，进入Python环境。
  ii. 执行以下命令检查SDK版本：

  ```
  >>> import oss2
  >>> oss2.__version__
  '2.0.0'
  ```

  上面的输出表明您已经成功安装了Python SDK 2.0.0。

- 验证crcmod

  OSS的CRC数据校验有两种方式：C扩展模式（通过SDK的依赖库crcmod调用扩展库_crcfunext.so计算CRC）和纯Python模式。前者的性能远优于后者。安装oss2时会自动安装crcmod。crcmod详情请参见crcmod introduction。

  如果发现安装SDK之后调用上传、下载接口较其他工具如ossutil或者其他SDK慢了很多，有可能是安装SDK的依赖库crcmod的C扩展模式失败，导致在上传、下载计算CRC校验码时采用了纯Python模式。

  验证crcmod的C扩展模式是否安装成功的方法如下：

  i. 在命令行输入 `python` 并回车，进入Python环境。
  ii. 输入 `import crcmod._crcfunext` 并回车。
    - 如果没有出现错误提示，则表明crcmod库的C扩展模式安装成功。

- 如果出现以下错误提示，则表明crcmod库的C扩展模式安装失败。

```
>>> import crcmod._crcfunext
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named _crcfunext
```

出现这种情况的原因是编译crcmod时，由于_crcfunext.so依赖Python.h文件，而系统中缺少这个头文件，因此_crcfunext.so库生成失败。CRC数据校验就会使用纯Python方式。虽然SDK安装成功，但是上传、下载等操作的效率非常低下。

如果出现该问题，请执行以下步骤：

a. 执行以下命令卸载crcmod：

```
pip uninstall crcmod
```

b. 安装python-devel

c. 执行以下命令重新安装crcmod：

```
pip install crcmod
```

如果执行上述步骤依然安装失败，请卸载crcmod，然后执行以下命令查看安装失败的详细原因：

```
pip install crcmod -v
```

卸载SDK

如果安装失败，建议通过pip卸载然后重装。卸载命令如下：

```
pip uninstall oss2
```

5.3. 初始化

本文介绍如何初始化Python SDK。

使用Python SDK时，大部分操作都是通过oss2.Service和oss2.Bucket两个类进行。

- oss2.Service类用于列举存储空间。
- oss2.Bucket类用于上传、下载、删除文件以及对存储空间进行各种配置。

初始化这两个类时，需要指定Endpoint。其中oss2.Service类不支持自定义域名访问。关于Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

初始化oss2.Service类

具体操作，请参见列举存储空间。

初始化oss2.Bucket类

- 使用OSS域名初始化

  以下代码用于使用OSS域名初始化。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
endpoint = 'yourEndpoint'
# 填写Bucket名称。
bucket = oss2.Bucket(auth, endpoint, 'examplebucket')
```

- 使用自定义域名初始化

  下面的代码用于使用自定义域名初始化：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# 自定义域名以my-domain.com为例。
cname = 'http://my-domain.com'
# 填写Bucket名称，并设置is_cname=True来开启CNAME。CNAME是指将自定义域名绑定到存储空间。
bucket = oss2.Bucket(auth, cname, 'examplebucket', is_cname=True)
```

- 设置连接超时时间

  以下代码用于设置连接超时时间：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
endpoint = 'yourEndpoint'
# 填写Bucket名称，并设置连接超时时间为30秒。
bucket = oss2.Bucket(auth, endpoint, 'examplebucket', connect_timeout=30)
```

- 关闭CRC数据校验

  上传和下载文件时默认开启CRC数据校验，确保上传和下载过程的数据完整性。

  > **警告** 强烈建议您不要关闭CRC数据校验功能。如果您关闭此功能，则阿里云不保证上传和下载过程数据的完整性。

  以下代码用于关闭CRC数据校验。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
endpoint = 'yourEndpoint'
# 填写Bucket名称，并设置enable_crc=False来关闭CRC数据校验。
bucket = oss2.Bucket(auth, endpoint, 'examplebucket', enable_crc=False)
```

5.4. 快速入门

本文介绍如何快速使用OSS Python SDK完成常见操作，如创建存储空间（Bucket）、上传和下载文件（Object）等。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置存储空间为私有读写权限。
bucket.create_bucket(oss2.models.BUCKET_ACL_PRIVATE)
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
# <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
bucket.put_object_from_file('<yourObjectName>', '<yourLocalFile>')
```

下载文件

以下代码用于将指定的OSS文件下载到本地文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# <yourObjectName>从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
# <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>')
```

列举文件

以下代码用于列举指定存储空间下的10个文件：

```
# -*- coding: utf-8 -*-
import oss2
from itertools import islice
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# oss2.ObjectIterator用于遍历文件。
for b in islice(oss2.ObjectIterator(bucket), 10):
    print(b.key)
```

删除文件

以下代码用于删除指定文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# <yourObjectName>表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
bucket.delete_object('<yourObjectName>')
```

删除文件详情请参见管理文件中的删除文件。

5.5. 存储空间

5.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# 通过指定Endpoint和存储空间名称，您可以在指定的地域创建新的存储空间。Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建存储空间。
# 如果需要在创建存储空间时设置存储类型、存储空间访问权限、数据容灾类型，请参考以下代码。
# 以下以配置存储空间为标准存储类型，访问权限为私有，数据容灾类型为同城冗余存储为例。
# bucketConfig = oss2.models.BucketCreateConfig(oss2.BUCKET_STORAGE_CLASS_STANDARD, oss2.BUCKET_DATA_REDUNDANCY_TYPE_ZRS)
# bucket.create_bucket(oss2.BUCKET_ACL_PRIVATE, bucketConfig)
bucket.create_bucket()
```

创建低频类型的存储空间示例代码如下：

```
# 设置存储空间的存储类型为低频访问类型，访问权限为公共读。
bucket.create_bucket(oss2.BUCKET_ACL_PUBLIC_READ, oss2.models.BucketCreateConfig(oss2.BUCKET_STORAGE_CLASS_IA))
```

创建存储空间的更多详情，请参见Put Bucket。

5.5.2. 列举存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何列举所有的存储空间、列举指定前缀（prefix）的存储空间等。

列举所有的存储空间

以下代码用于列举所有存储空间。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')
# 列举所有的存储空间。
for b in oss2.BucketIterator(service):
    print(b.name)
```

列举指定前缀的存储空间

以下代码用于列举指定前缀（prefix）的存储空间：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')
# 列举前缀为test-的存储空间。
for b in oss2.BucketIterator(service, prefix='test-'):
    print(b.name)
```

列举指定marker之后的存储空间

以下代码用于列举examplebucket之后的存储空间。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')
# 列举按字典序排列在test-bucket1之后的存储空间。列举结果中不包含名为test-bucket1的存储空间。
for b in oss2.BucketIterator(service, marker='test-bucket1'):
    print(b.name)
```

5.5.3. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取存储空间的地域信息。
result = bucket.get_bucket_location()
print('location: ' + result.location)
```

5.5.4. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

以下代码用于判断存储空间examplebucket是否存在。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
def does_bucket_exist(bucket):
  try:
    bucket.get_bucket_info()
  except oss2.exceptions.NoSuchBucket:
    return False
  except:
    raise
  return True
exist = does_bucket_exist(bucket)
# 返回值为true表示已存在同名bucket，false表示不存在同名bucket。
if exist:
  print('bucket exist')
else:
  print('bucket not exist')
```

5.5.5. 获取存储空间信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期等。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取存储空间相关信息，包括存储空间的存储类型、创建日期、访问权限、数据容灾类型等。
bucket_info = bucket.get_bucket_info()
print('name: ' + bucket_info.name)
print('storage class: ' + bucket_info.storage_class)
print('creation date: ' + bucket_info.creation_date)
print('intranet_endpoint: ' + bucket_info.intranet_endpoint)
print('extranet_endpoint ' + bucket_info.extranet_endpoint)
print('owner: ' + bucket_info.owner.id)
print('grant: ' + bucket_info.acl.grant)
print('data_redundancy_type:' + bucket_info.data_redundancy_type)
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

5.5.6. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	oss2.BUCKET_ACL_PRIVATE
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	oss2.BUCKET_ACL_PUBLIC_READ
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	oss2.BUCKET_ACL_PUBLIC_READ_WRITE

以下代码用于设置存储空间的访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置存储空间访问权限为私有。
bucket.put_bucket_acl(oss2.BUCKET_ACL_PRIVATE)
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间访问权限

以下代码用于获取存储空间的访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
print(bucket.get_bucket_acl().acl)
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

5.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> 说明
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
try:
  # 删除存储空间。
  bucket.delete_bucket()
except oss2.exceptions.BucketNotEmpty:
  print('bucket is not empty.')
except oss2.exceptions.NoSuchBucket:
  print('bucket does not exist')
```

对于非空的存储空间，可以通过边列举边删除（对于分片上传则是终止上传）的方法清空存储空间，然后再删除。

删除存储空间的更多详情，请参见DeleteBucket。

5.5.8. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

> 说明
> - 只有Bucket的拥有者及授权子账户才能为Bucket设置用户标签，否则返回403 Forbidden错误，错误码：AccessDenied。
> - 最多可设置20对Bucket用户标签（Key-Value对）。
> - Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
> - Value最大长度为128字节，可以为空。
> - Key和Value必须为UTF-8编码。
> - PutBucketTagging是覆盖语义，即新设置的标签会完全覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import Tagging, TaggingRule
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建标签规则。
rule = TaggingRule()
rule.add('key1', 'value1')
rule.add('key2', 'value2')
# 创建标签。
tagging = Tagging(rule)
# 设置Bucket标签。
result = bucket.put_bucket_tagging(tagging)
# 查看HTTP返回码。
print('http status:', result.status)
```

获取Bucket标签

以下代码用于获取Bucket标签：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取Bucket标签信息。
result = bucket.get_bucket_tagging()
# 查看获取到的标签规则。
tag_rule = result.tag_set.tagging_rule
print('tag rule:', tag_rule)
```

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
# -*- coding: utf-8 -*-
import oss2
#创建Server对象。
#阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')
#填充tag-key，tag-value字段到list_buckets接口的params参数中。
params = {}
params['tag-key'] = '<yourTagging_key>'
params['tag-value'] = '<yourTagging_value>'
#列举出带指定标签的Bucket。
result = service.list_buckets(params=params)
#查看列举结果。
for bucket in result.buckets:
    print('result bucket_name:', bucket.name)
```

删除Bucket标签

以下代码用于删除Bucket标签：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 删除Bucket标签。
result = bucket.delete_bucket_tagging()
# 查看HTTP返回码。
print('http status:', result.status)
```

5.5.9. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **说明** 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

> **说明**
> - 单个Bucket最多只能有1000条清单配置。
> - 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单（Inventory）配置：

```
# -*- coding: utf-8 -*-
```

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import (InventoryConfiguration,
                         InventoryFilter,
                         InventorySchedule,
                         InventoryDestination,
                         InventoryBucketDestination,
                         InventoryServerSideEncryptionKMS,
                         InventoryServerSideEncryptionOSS,
                         INVENTORY_INCLUDED_OBJECT_VERSIONS_CURRENT,
                         INVENTORY_INCLUDED_OBJECT_VERSIONS_ALL,
                         INVENTORY_FREQUENCY_DAILY,
                         INVENTORY_FREQUENCY_WEEKLY,
                         INVENTORY_FORMAT_CSV,
                         FIELD_SIZE,
                         FIELD_LAST_MODIFIED_DATE,
                         FIELD_STORAG_CLASS,
                         FIELD_ETAG,
                         FIELD_IS_MULTIPART_UPLOADED,
                         FIELD_ENCRYPTION_STATUS)
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
account_id = '<yourtBucketDestinationAccountId>'
role_arn = '<yourBucketDestinationRoleArn>'
dest_bucket_name = '<yourDestinationBucketName>'
# 设置清单配置id。
inventory_id = "test-config-id"
# 清单中包含的object属性。
optional_fields = [FIELD_SIZE, FIELD_LAST_MODIFIED_DATE, FIELD_STORAG_CLASS,
    FIELD_ETAG, FIELD_IS_MULTIPART_UPLOADED, FIELD_ENCRYPTION_STATUS]
# 设置清单bucket目的地信息。
bucket_destination = InventoryBucketDestination(
    # 目的地bucket的用户account_id。
    account_id=account_id,
    # 目的地bucket的role_arn。
    role_arn=role_arn,
    # 目的地bucket的名称。
    bucket=dest_bucket_name,
    # 清单格式。
    inventory_format=INVENTORY_FORMAT_CSV,
    # 清单结果的存储路径前缀。
    prefix='destination-prefix',
    # 如果需要使用kms加密清单，请参考以下代码。
    # sse_kms_encryption=InventoryServerSideEncryptionKMS("test-kms-id"),
    # 如果需要使用OSS服务端加密清单，请参考以下代码。
    # sse_oss_encryption=InventoryServerSideEncryptionOSS()
    )
# 创建清单配置。
inventory_configuration = InventoryConfiguration(
    # 设置清单的配置id。
    inventory_id=inventory_id,
    # 清单配置是否启用的标识，true或false。
```

```
    # 清单配置是否启用的标识，true或false。
    is_enabled=True,
    # 设置清单的生成计划，以下示例为每天一次。其中，WEEKLY对应每周一次，DAILY对应每天一次。
    inventory_schedule=InventorySchedule(frequency=INVENTORY_FREQUENCY_DAILY),
    # 设置清单中包含的object的版本为当前版本。如果设置为INVENTORY_INCLUDED_OBJECT_VERSIONS_ALL则表示object的所有版本，在版本控制状态下生效。
    included_object_versions=INVENTORY_INCLUDED_OBJECT_VERSIONS_CURRENT,
    # 设置清单清筛选object的前缀。
    inventory_filter=InventoryFilter(prefix="obj-prefix"),
    # 设置清单中包含的object属性。
    optional_fields=optional_fields,
    # 设置清单的接收目的地。
    inventory_destination=InventoryDestination(bucket_destination=bucket_destination))
# 上传清单配置。
bucket.put_bucket_inventory_configuration(inventory_configuration)
```

有关添加存储空间清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
# -*- coding: utf-8 -*-
import oss2
import os
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置清单配置id。
inventory_id = "test-config-id"
# 获取清单配置。
result = bucket.get_bucket_inventory_configuration(inventory_id=inventory_id);
# 打印清单配置信息。
print('======inventory configuration======')
print('inventory_id', result.inventory_id)
print('is_enabled', result.is_enabled)
print('frequency', result.inventory_schedule.frequency)
print('included_object_versions', result.included_object_versions)
print('inventory_filter prefix', result.inventory_filter.prefix)
print('fields', result.optional_fields)
bucket_destin = result.inventory_destination.bucket_destination
print('===bucket destination===')
print('account_id', bucket_destin.account_id)
print('role_arn', bucket_destin.role_arn)
print('bucket', bucket_destin.bucket)
print('format', bucket_destin.inventory_format)
print('prefix', bucket_destin.prefix)
if bucket_destin.sse_kms_encryption is not None:
  print('server side encryption by kms, key id:', bucket_destin.sse_kms_encryption.key_id)
elif bucket_destin.sse_oss_encryption is not None:
  print('server side encryption by oss.')
```

有关查看清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明** 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
# -*- coding: utf-8 -*-
import oss2
import os
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 打印清单配置信息。
def print_inventory_configuration(configuration):
    print('======inventory configuration======')
    print('inventory_id', configuration.inventory_id)
    print('is_enabled', configuration.is_enabled)
    print('frequency', configuration.inventory_schedule.frequency)
    print('included_object_versions', configuration.included_object_versions)
    print('inventory_filter prefix', configuration.inventory_filter.prefix)
    print('fields', configuration.optional_fields)
    bucket_destin = configuration.inventory_destination.bucket_destination
    print('===bucket destination===')
    print('account_id', bucket_destin.account_id)
    print('role_arn', bucket_destin.role_arn)
    print('bucket', bucket_destin.bucket)
    print('format', bucket_destin.inventory_format)
    print('prefix', bucket_destin.prefix)
    if bucket_destin.sse_kms_encryption is not None:
        print('server side encryption by kms, key id:', bucket_destin.sse_kms_encryption.key_id)
    elif bucket_destin.sse_oss_encryption is not None:
        print('server side encryption by oss.')
# 列举所有的清单配置。
# 如果存在超过100条配置，列举结果将会分页，分页信息保存在class: <oss2.models.ListInventoryConfigurationResult>中。
continuation_token = None
while 1:
    result = bucket.list_bucket_inventory_configurations(continuation_token=continuation_token)
    # 本次列举结果是否分页。
    print('is truncated', result.is_truncated)
    # 本次列举携带的token。
    print('continuaiton_token', result.continuaiton_token)
    # 下次列举需要携带的token。
    print('next_continuation_token', result.next_continuation_token)
    # 打印清单配置信息。
    for inventory_config in result.inventory_configurations:
        print_inventory_configuration(inventory_config)
    # 如果本次列举为分页列举，则继续列举，且需要携带分页token。
    if result.is_truncated:
        continuation_token = result.next_continuation_token
    else:
        break
```

有关批量列举清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
# -*- coding: utf-8 -*-
import oss2
import os
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置清单配置id。
inventory_id = "test-config-id"
# 删除指定id的清单配置。
bucket.delete_bucket_inventory_configuration(inventory_id)
```

有关删除存储空间清单配置的详情，请参见DeleteBucketInventory。

5.5.10. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
# -*- coding: utf-8 -*-
import oss2
import json
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建policy_text。
policy_text = '{"Statement": [{"Effect": "Allow", "Action": ["oss:GetObject", "oss:ListObjects"], "Resource": ["acs:oss:*:*:*/user1/*"]}], "Version": "1"}'
# 上传授权策略。
bucket.put_bucket_policy(policy_text)
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取授权策略。
result = bucket.get_bucket_policy()
policy_json = json.loads(result.policy)
print("Get policy text: ", policy_json)
```

有关获取Bucket Policy信息详情，请参见GetBucketPolicy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 删除授权策略。
result = bucket.delete_bucket_policy()
assert int(result.status)//100 == 2
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

5.5.11. 合规保留策略

对象存储OSS允许针对存储空间（Bucket）设置基于时间的合规保留策略，保护周期为1天到70年。本文介绍如何新建、获取、锁定合规保留策略等。

背景信息

OSS支持WORM（Write Once Read Many）特性，允许您以不可删除、不可篡改的方式保存和使用数据。

当基于时间的合规保留策略创建24小时后未提交锁定，则该策略自动失效。当合规保留策略锁定后，您可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，不允许删除Object及合规保留策略，且无法缩短策略保护周期，仅允许延长Object的保留时间。有关合规保留策略的详情，请参见合规保留策略。

新建合规保留策略

以下代码用于新建合规保留策略：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 新建合规保留策略，并指定Object保护天数为1天。
result = bucket.init_bucket_worm(1)
# 查看合规保留策略ID。
print(result.worm_id)
```

有关新建合规保留策略的详情，请参见InitiateBucketWorm。

取消未锁定的合规保留策略

以下代码用于取消未锁定的合规保留策略：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 取消未锁定的合规保留策略。
bucket.abort_bucket_worm()
```

有关取消未锁定的合规保留策略的详情，请参见AbortBucketWorm。

锁定合规保留策略

以下代码用于锁定合规保留策略：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 锁定合规保留策略。
bucket.complete_bucket_worm('<yourWromId>')
```

有关锁定合规保留策略的详情，请参见CompleteBucketWorm。

获取合规保留策略

以下代码用于获取合规保留策略：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取合规保留策略。
result = self.bucket.get_bucket_worm()
# 查看合规保留策略ID。
print(result.worm_id)
# 查看合规保留策略状态。未锁定状态下为"InProgress"，锁定状态下为"Locked"。
print(result.state)
# 查看Object的保护时间。
print(result.retention_period_days)
# 查看合规保留策略创建时间。
print(result.creation_date)
```

有关获取合规保留策略的详情，请参见GetBucketWorm。

延长Object的保留天数

以下代码用于延长已锁定的合规保留策略中Object的保留天数：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 延长已锁定的合规保留策略中Object的保留天数。
bucket.extend_bucket_worm('<yourWormId>', 2)
```

有关延长已锁定的合规保留策略中Object的保留天数详情，请参见ExtendBucketWorm。

5.5.12. 请求者付费模式

阿里云对象存储OSS的请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import PAYER_BUCKETOWNER, PAYER_REQUESTER
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置请求者付费模式，默认的付费者为PAYER_BUCKETOWNER。
result = bucket.put_bucket_request_payment(PAYER_REQUESTER)
print("http respon status: ", result.status)
```

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 查看付费者模式。
result = bucket.get_bucket_request_payment()
print('payer:', result.payer)
```

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject和DeleteObject为例，用于指定第三方付费访问Object。

其他用于指定第三方付费的Object读写操作接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_REQUEST_PAYER
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourRequestBucketName>')
object_name = '<yourRequestobjectName>'
headers = dict()
headers[OSS_REQUEST_PAYER] = "requester"
# 上传文件, 需要指定header。
result = bucket.put_object(object_name, 'test-content', headers=headers)
# 删除文件, 需要指定header。
result = bucket.delete_object(object_name, headers=headers);
```

5.5.13. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

生命周期规则包含如下信息：

- 前缀或标签匹配策略：生命周期规则匹配的Object和碎片。
  - 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。
  - 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，OSS对所有拥有这些标签的对象执行生命周期规则。标签匹配不可以作用于碎片。

    > **说明** 对象标签功能详情请参见对象标签。

  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。
  - 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。
- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Object成为非当前版本天数：指定一个过期天数N，并指定非当前版本Object过期后执行什么操作。Object会在其成为非当前版本的N天后过期，并执行指定的操作。

    > **说明** 您可以将过期Object转换为低频访问类型或归档类型，也可以选择删除过期Object。详情请参见生命周期配置元素。

- 碎片过期策略：设置碎片的过期时间及操作。
  - 过期天数：可指定一个过期天数N，碎片会在其最后修改时间的N天后被删除。
  - 过期日期：指定一个过期日期，最后修改时间在该日期之前的碎片会被全部删除。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
# -*- coding: utf-8 -*-
import oss2
import datetime
from oss2.models import (LifecycleExpiration, LifecycleRule,
                BucketLifecycle,AbortMultipartUpload,
                TaggingRule, Tagging, StorageTransition,
                NoncurrentVersionStorageTransition,
                NoncurrentVersionExpiration)
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置object距其最后修改时间3天后过期。
rule1 = LifecycleRule('rule1', 'tests/',
            status=LifecycleRule.ENABLED,
            expiration=LifecycleExpiration(days=3))
# 设置object过期规则，指定日期之前创建的文件过期。
rule2 = LifecycleRule('rule2', 'tests2/',
            status=LifecycleRule.ENABLED,
expiration=LifecycleExpiration(created_before_date=datetime.date(2018, 12, 12)))
# 设置分片过期规则，分片3天后过期。
rule3 = LifecycleRule('rule3', 'tests3/',
            status=LifecycleRule.ENABLED,
        abort_multipart_upload=AbortMultipartUpload(days=3))
# 设置分片过期规则，指定日期之前的分片过期。
rule4 = LifecycleRule('rule4', 'tests4/',
            status=LifecycleRule.ENABLED,
            abort_multipart_upload = AbortMultipartUpload(created_before_date=datetime.date(2018, 12, 12)))
# 设置存储类型转换规则，指定Object在其最后修改时间20天之后转为低频访问类型，在其最后修改时间30天之后转为归档类型。
rule5 = LifecycleRule('rule5', 'tests5/',
            status=LifecycleRule.ENABLED,
            storage_transitions=[StorageTransition(days=20,storage_class=oss2.BUCKET_STORAGE_CLASS_IA),
                StorageTransition(days=30,storage_class=oss2.BUCKET_STORAGE_CLASS_ARCHIVE)])
# 设置匹配的标签。
tagging_rule = TaggingRule()
tagging_rule.add('key1', 'value1')
tagging_rule.add('key2', 'value2')
tagging = Tagging(tagging_rule)
# 设置存储类型转换规则，指定Object在其最后修改时间超过365天后转为ARCHIVE类型。
# rule6与以上几个规则不同的是它指定了匹配的标签，同时拥有key1=value1,key2=value2两个标签的object才会匹配此规则。
rule6 = LifecycleRule('rule6', 'tests6/',
```

```
            status=LifecycleRule.ENABLED,
            storage_transitions=[StorageTransition(created_before_date=datetime.date(2018, 12, 12),storag
e_class=oss2.BUCKET_STORAGE_CLASS_IA)],
            tagging = tagging)
# rule7针对版本控制状态下的Bucket。
# 设置object在其最后修改时间365天之后自动转为ARCHIVE类型。
# 设置自动移除过期删除标记。
# 设置非当前版本object 12天后转为IA类型。
# 设置非当前版本object 20天后转为ARCHIVE类型。
# 设置非当前版本object 30天后删除。
rule7 = LifecycleRule('rule7', 'tests7/',
        status=LifecycleRule.ENABLED,
        storage_transitions=[StorageTransition(days=365, storage_class=oss2.BUCKET_STORAGE_CLASS_AR
CHIVE)],
        expiration=LifecycleExpiration(expired_detete_marker=True),
        noncurrent_version_sotrage_transitions =
            [NoncurrentVersionStorageTransition(12, oss2.BUCKET_STORAGE_CLASS_IA),
             NoncurrentVersionStorageTransition(20, oss2.BUCKET_STORAGE_CLASS_ARCHIVE)],
        noncurrent_version_expiration = NoncurrentVersionExpiration(30))
lifecycle = BucketLifecycle([rule1, rule2, rule3, rule4, rule5, rule6, rule7])
bucket.put_bucket_lifecycle(lifecycle)
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 查看生命周期规则。
lifecycle = bucket.get_bucket_lifecycle()
for rule in lifecycle.rules:
  print('==========')
  print('id:', rule.id)
  print('prefix:', rule.prefix)
  print('status', rule.status)
  if rule.tagging is not None:
    print('tagging:', rule.tagging)
  if rule.abort_multipart_upload is not None:
    if rule.abort_multipart_upload.days is not None:
      print('abort_multipart_upload days:', rule.abort_multipart_upload.days)
    else:
      print('abort_multipart_upload created_before_date:', rule.abort_multipart_upload.created_before_date)
  if rule.expiration is not None:
    if rule.expiration.days is not None:
      print('expiration days:', rule.expiration.days)
    elif rule.expiration.expired_detete_marker is not None:
      print('expiration expired_detete_marker:', rule.expiration.expired_detete_marker)
    elif rule.expiration.created_before_date is not None:
      print('expiration created_before_date:', rule.expiration.created_before_date)
  if len(rule.storage_transitions) > 0:
    storage_info = ''
    for storage_rule in rule.storage_transitions:
      if storage_rule.days is not None:
        storage_info += 'days={0}, storage_class={1} *** '.format(
            storage_rule.days, storage_rule.storage_class)
      else:
        storage_info += 'created_before_date={0}, storage_class={1} *** '.format(
            storage_rule.created_before_date, storage_rule.storage_class)
    print('storage_transitions:', storage_info)
  if len(rule.noncurrent_version_sotrage_transitions) > 0:
    noncurrent_storage_info = ''
    for storage_rule in rule.noncurrent_version_sotrage_transitions:
      noncurrent_storage_info += 'days={0}, storage_class={1} *** '.format(
        storage_rule.noncurrent_days, storage_rule.storage_class)
    print('noncurrent_version_sotrage_transitions:', noncurrent_storage_info)
  if rule.noncurrent_version_expiration is not None:
    print('noncurrent_version_expiration days:', rule.noncurrent_version_expiration.noncurrent_days)
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
#清空生命周期规则。
bucket.delete_bucket_lifecycle()
# 再次查看生命周期规则会抛出异常。
try:
  lifecycle = bucket.get_bucket_lifecycle()
except oss2.exceptions.NoSuchLifecycle:
  print('lifecycle is not configured')
```

5.5.14. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链

以下代码用于设置防盗链：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketReferer
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置允许空Referer（True表示允许空Referer，False表示不允许空Referer），并设置Referer白名单。
bucket.put_bucket_referer(BucketReferer(True, ['http://aliyun.com', 'http://*.aliyuncs.com']))
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
config = bucket.get_bucket_referer()
print('allow empty referer={0}, referers={1}'.format(config.allow_empty_referer, config.referers))
```

清空防盗链

以下代码用于清空防盗链：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 不能直接清空防盗链，需要新建一个允许空Referer的规则来覆盖之前的规则。
bucket.put_bucket_referer(BucketReferer(True, []))
```

5.5.15. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

关于访问日志的更多信息，请参见开发指南中的日志转存。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketLogging
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启日志记录。把日志保存在当前存储空间，设置日志文件存放的目录为 `logging/` 。
logging = bucket.put_bucket_logging(BucketLogging(bucket.bucket_name, 'logging/'))
if logging.status == 200:
    print("Enable access logging")
else:
    print("request_id： ", logging.request_id)
    print("resp : ", logging.resp.response)
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
logging = bucket.get_bucket_logging()
print('TargetBucket={0}, TargetPrefix={1}'.format(logging.target_bucket, logging.target_prefix))
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
logging = bucket.delete_bucket_logging()
if logging.status == 204:
    print("Disable access logging")
else:
    print("request_id： ", logging.request_id)
    print("resp : ", logging.resp.response)
```

5.5.16. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketWebsite
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启静态网站托管模式，并把索引页面设置为index.html，404错误页面设置为error.html。
bucket.put_bucket_website(BucketWebsite('index.html', 'error.html'))
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
try:
  # 当静态网站托管模式没有开启时，get_bucket_website会抛出NoSuchWebsite异常。
  website = bucket.get_bucket_website()
  print('Index file is {0}, error file is {1}'.format(website.index_file, website.error_file))
except oss2.exceptions.NoSuchWebsite as e:
  print('Website is not configured, request_id={0}'.format(e.request_id))
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
bucket.delete_bucket_website()
```

5.5.17. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketCors, CorsRule
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
rule = CorsRule(allowed_origins=['*'],
            allowed_methods=['GET', 'HEAD'],
            allowed_headers=['*'],
            max_age_seconds=1000)
# 已存在的规则将被覆盖。
bucket.put_bucket_cors(BucketCors([rule]))
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
try:
  cors = bucket.get_bucket_cors()
except oss2.exceptions.NoSuchCors:
  print('cors is not set')
else:
  for rule in cors.rules:
    print('AllowedOrigins={0}'.format(rule.allowed_origins))
    print('AllowedMethods={0}'.format(rule.allowed_methods))
    print('AllowedHeaders={0}'.format(rule.allowed_headers))
    print('ExposeHeaders={0}'.format(rule.expose_headers))
    print('MaxAgeSeconds={0}'.format(rule.max_age_seconds))
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
bucket.delete_bucket_cors()
```

5.5.18. 跨区域复制

跨区域复制是在不同OSS地域之间自动、异步复制文件，将源存储空间（Bucket）中文件的改动（新建、覆盖、删除操作）同步到目标存储空间中。该功能用于满足异地容灾和数据复制的需求。

> **说明** 关于跨区域复制的更多信息，请参见开发指南中的管理跨区域复制。

开启跨区域复制

以下代码用于开启跨区域复制，将华东1（杭州）地域下源存储空间srcexamplebucket中的数据复制到华北2（北京）地域下的目标存储空间dstexamplebucket。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ReplicationRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# 依次填写源Bucket所在地域对应的Endpoint和源Bucket名称。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
bucket = oss2.Bucket(auth, 'https://oss-cn-hangzhou.aliyuncs.com', 'srcexamplebucket')
# 依次填写复制规则ID、目标Bucket名称、目标Bucket所在地域和是否同步历史数据。
# 如果未设置rule_id或者设置rule_id为空，则OSS会为该复制规则生成唯一值。
# 目标Bucket所在地域以华北2（北京）为例，target_bucket_location填写为oss-cn-beijing。
# OSS默认会同步历史数据。如果设置is_enable_historical_object_replication为False，表示禁止同步历史数据。
replica_config = ReplicationRule(rule_id='test_replication_1',
                 target_bucket_name='dstexamplebucket',
                 target_bucket_location='oss-cn-beijing',
                 is_enable_historical_object_replication=False
                 )
# 开启跨区域复制。
bucket.put_bucket_replication(replica_config)
```

查看跨区域复制信息

以下代码用于查看存储空间examplebucket已开启的跨区域复制信息。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ReplicationRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称，例如examplebucket。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 查看跨区域复制信息。
result = bucket.get_bucket_replication()
# 打印返回的信息。
for rule in result.rule_list:
  print(rule.rule_id)
  print(rule.target_bucket_name)
  print(rule.target_bucket_location)
```

查看数据同步进度

数据同步进度包括历史数据同步进度和实时数据同步进度。

- 历史数据同步进度用百分比表示，仅对开启了历史数据同步的存储空间有效。
- 实时数据同步进度用新写入数据的时间点表示，代表这个时间点之前的数据已同步完成。

以下代码用于查看存储空间examplebucket的test_replication_1复制规则的数据同步进度。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ReplicationRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称，例如examplebucket。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 查看跨区域复制进度。
# 填写复制规则ID，例如test_replication_1。
result = bucket.get_bucket_replication_progress('test_replication_1')
print(result.progress.rule_id)
# 是否开启了历史数据同步。
print(result.progress.is_enable_historical_object_replication)
# 历史数据同步进度。
print(result.progress.historical_object_progress)
# 实时数据同步进度。
print(result.progress.new_object_progress)
```

查看可同步的目标地域

以下代码用于查看存储空间examplebucket的数据所能同步到的地域列表。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ReplicationRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称，例如examplebucket。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 查看可同步的目标地域。
result = bucket.get_bucket_replication_location()
for location in result.location_list:
    print(location)
```

关闭跨区域复制关系

通过删除存储空间的复制规则，您可以关闭源存储空间到目标存储空间的跨区域复制关系。

以下代码用于删除存储空间examplebucket的test_replication_1复制规则。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ReplicationRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称，例如examplebucket。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 关闭跨区域复制。
# 填写复制规则ID，例如test_replication_1。
result = bucket.delete_bucket_replication('test_replication_1')
```

5.6. 上传文件

5.6.1. 概述

在OSS中，操作的基本数据单元是文件（Object）。OSS Python SDK提供了丰富的文件上传方式：

- 简单上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

5.6.2. 简单上传

简单上传是指通过 `put_object` 方法上传单个文件（Object）。简单上传包括上传字符串、上传Bytes、上传Unicode字符、上传网络流和上传本地文件五种形式。

> **说明** 关于上传文件的更多信息，请参见PutObject。

注意事项

上传文件（Object）时，如果存储空间（Bucket）中已存在同名文件且用户对该文件有访问权限，则新添加的文件将覆盖原有文件。

上传文件时涉及填写的公共参数如下：

参数	说明

参数	说明
bucket_name	Bucket名称。 Bucket名称的命名规范如下： • 只能包括小写字母、数字和短划线（-）。 • 必须以小写字母或者数字开头和结尾。 • 长度必须在3~63字符之间。
object_name	Object完整路径。Object完整路径中不能包含Bucket名称。 Object命名规范如下： • 使用UTF-8编码。 • 长度必须在1~1023字符之间。 • 不能以正斜线（/）或者反斜线（\）开头。

上传字符串

以下代码用于将字符串上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 上传文件。
# 如果需要在上传文件时设置文件存储类型（x-oss-storage-class）和访问权限（x-oss-object-acl），请在put_object中设置相关Header。
# headers = dict()
# headers["x-oss-storage-class"] = "Standard"
# headers["x-oss-object-acl"] = oss2.OBJECT_ACL_PRIVATE
# 填写Object完整路径和字符串。Object完整路径中不能包含Bucket名称。
# result = bucket.put_object('exampleobject.txt', 'Hello OSS', headers=headers)
result = bucket.put_object('exampleobject.txt', 'Hello OSS')
# HTTP返回码。
print('http status: {0}'.format(result.status))
# 请求ID。请求ID是本次请求的唯一标识，强烈建议在程序日志中添加此参数。
print('request_id: {0}'.format(result.request_id))
# ETag是put_object方法返回值特有的属性，用于标识一个Object的内容。
print('ETag: {0}'.format(result.etag))
# HTTP响应头部。
print('date: {0}'.format(result.headers['date']))
```

上传Bytes

以下代码用于将Bytes上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径和Bytes内容。Object完整路径中不能包含Bucket名称。
bucket.put_object('exampleobject.txt', b'Hello OSS')
```

上传Unicode字符

以下代码用于将Unicode字符上传到目标存储空间examplebucket中的exampleobject.txt文件。上传Unicode字符时，OSS会将Unicode字符自动转换为UTF-8编码的Bytes进行上传。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径和Unicode字符。Object完整路径中不能包含Bucket名称。
bucket.put_object('exampleobject.txt', u'Hello OSS')
```

上传网络流

以下代码用于将网络流上传到目标存储空间examplebucket中的exampleobject.txt文件。OSS将网络流视为可迭代对象（Iterable），并以Chunked Encoding的方式上传。

```
# -*- coding: utf-8 -*-
import oss2
import requests
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# requests.get返回的是一个可迭代对象（Iterable），此时Python SDK会通过Chunked Encoding方式上传。
# 填写网络流地址。
input = requests.get('http://www.aliyun.com')
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
bucket.put_object('exampleobject.txt', input)
```

上传本地文件

以下代码用于将本地文件examplefile.txt上传到目标存储空间examplebucket中的exampleobject.txt文件。OSS将本地文件视为文件对象（File Object），上传时必须以二进制方式打开（例如rb模式）。

```
# -*- coding: utf-8 -*-
import oss2
import os
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 必须以二进制的方式打开文件。
# 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
with open('D:\\localpath\\examplefile.txt', 'rb') as fileobj:
  # Seek方法用于指定从第1000个字节位置开始读写。上传时会从您指定的第1000个字节位置开始上传，直到文件结束。
  fileobj.seek(1000, os.SEEK_SET)
  # Tell方法用于返回当前位置。
  current = fileobj.tell()
  # 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  bucket.put_object('exampleobject.txt', fileobj)
```

Python SDK还提供了更便捷的方法用于上传本地文件。

```
# 填写Object完整路径和本地文件的完整路径。Object完整路径中不能包含Bucket名称。
# 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
bucket.put_object_from_file('exampleobject.txt', 'D:\\localpath\\examplefile.txt')
```

5.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

示例代码

以下代码用于追加上传文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'https://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置首次上传的追加位置（Position参数）为0。
# <yourObjectName>填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
result = bucket.append_object('<yourObjectName>', 0, 'content of first append')
# 如果不是首次上传，可以通过bucket.head_object方法或上次追加返回值的next_position属性，获取追加位置。
bucket.append_object('<yourObjectName>', result.next_position, 'content of second append')
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

5.6.4. 断点续传上传

通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录点。上传过程中，如果出现网络异常或程序崩溃导致文件上传失败时，将从断点记录处继续上传未上传完成的部分。

背景信息

您可以通过oss2.resumable_upload方法断点续传上传指定文件，该方法包含以下参数：

参数	描述	是否必须	默认值
bucket	存储空间名称	是	无
key	上传到OSS的文件名称	是	无
filename	待上传的本地文件名称	是	无
store	指定保存断点信息的目录	否	*HOME*目录下建立的*.py-oss-upload*目录
headers	HTTP头部	否	无
multipart_threshold	文件长度大于该值时，则使用分片上传	否	10 MB
part_size	分片大小	否	自动计算
progress_callback	上传进度回调函数	否	无
num_threads	并发上传线程数	否	1

有关断点续传上传的详情，请参见分片上传和断点续传。

代码实现

以下代码用于断点续传上传。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 当文件长度大于或等于可选参数multipart_threshold（默认值为10 MB）时，则使用分片上传。如未使用参数store指定目录，则会在HOME目录下建立.py-oss-upload目录来保存断点信息。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>')
```

Python SDK 2.1.0以上版本支持断点续传上传时设置可选参数，代码如下：

```
# 若使用store指定了目录，则断点信息将保存在指定目录中。若使用num_threads设置并发上传线程数，请将oss2.defaults.connection_pool_size设置为大于或等于并发上传线程数。默认并发上传线程数为1。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>',
  store=oss2.ResumableStore(root='/tmp'),
  multipart_threshold=100*1024,
  part_size=100*1024,
  num_threads=4)
```

说明

- 断点续传内部使用了多线程，调用时无需在外部封装多线程，否则可能导致数据重复传输。
- 网络情况较好时，建议增加分片大小。反之，减小分片大小。

5.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用bucket.init_multipart_upload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

   调用bucket.upload_part方法上传分片数据。

   说明

   - 对于同一个uploadId，分片号（partNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，那么OSS上这个分片已有的数据将会被覆盖。
   - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
   - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用bucket.complete_multipart_upload方法将所有分片合并成完整的文件。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
# -*- coding: utf-8 -*-
import os
from oss2 import SizedFileAdapter, determine_part_size
from oss2.models import PartInfo
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
filename = '<yourLocalFile>'
total_size = os.path.getsize(filename)
# determine_part_size方法用于确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 初始化分片。
# 如需在初始化分片时设置文件存储类型，请在init_multipart_upload中设置相关headers，参考如下。
# headers = dict()
# headers["x-oss-storage-class"] = "Standard"
# upload_id = bucket.init_multipart_upload(key, headers=headers).upload_id
upload_id = bucket.init_multipart_upload(key).upload_id
parts = []
# 逐个上传分片。
with open(filename, 'rb') as fileobj:
  part_number = 1
  offset = 0
  while offset < total_size:
    num_to_upload = min(part_size, total_size - offset)
    # 调用SizedFileAdapter(fileobj, size)方法会生成一个新的文件对象，重新计算起始追加位置。
    result = bucket.upload_part(key, upload_id, part_number,
                    SizedFileAdapter(fileobj, num_to_upload))
    parts.append(PartInfo(part_number, result.etag))
    offset += num_to_upload
    part_number += 1
# 完成分片上传。
# 如需在完成分片上传时设置文件访问权限ACL，请在complete_multipart_upload函数中设置相关headers，参考如下。
# headers = dict()
# headers["x-oss-object-acl"] = oss2.OBJECT_ACL_PRIVATE
# bucket.complete_multipart_upload(key, upload_id, parts, headers=headers)
bucket.complete_multipart_upload(key, upload_id, parts)
# 验证分片上传。
with open(filename, 'rb') as fileobj:
  assert bucket.get_object(key).read() == fileobj.read()
```

说明 网络情况较好时，建议增大分片大小。反之，减小分片大小。

有关分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

取消分片上传事件

您可以调用bucket.abort_multipart_upload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用此uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件。

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
# 由init_multipart_upload接口返回的upload_id。
upload_id = '<yourUploadId>'
# 取消指定upload_id的分片上传事件，已上传的分片会被删除。
bucket.abort_multipart_upload(key, upload_id)
```

取消分片上传事件的更多详情，请参见AbortMultipartUpload。

列举已上传的分片信息

以下代码用于列举已上传的分片信息：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
# 由init_multipart_upload接口返回的upload_id。
upload_id = '<yourUploadId>'
# 列举指定upload_id对应的已上传分片信息。
for part_info in oss2.PartIterator(bucket, key, upload_id):
    print('part_number:', part_info.part_number)
    print('etag:', part_info.etag)
    print('size:', part_info.size)
```

列举已上传分片的更多详情，请参见ListParts。

列举分片上传事件

- 列举指定Object的分片上传事件

以下代码用于列举指定Object的分片上传事件：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
# 列举Object的所有分片上传事件。对于同一个Object，每次调用init_multipart_upload均会返回不同的upload_id。
# 一个upload_id对应一个分片上传事件。
for upload_info in oss2.ObjectUploadIterator(bucket, key):
    print('key:', upload_info.key)
    print('upload_id:', upload_info.upload_id)
```

- 列举存储空间下的所有分片事件

  以下代码用于列举存储空间下的所有分片上传事件：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举存储空间下的所有分片上传事件。
for upload_info in oss2.MultipartUploadIterator(bucket):
    print('key:', upload_info.key)
    print('upload_id:', upload_info.upload_id)
```

- 列举存储空间下指定前缀Object的分片上传事件

  以下代码用于列举存储空间下指定前缀Object的分片上传事件：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举存储空间下以'test'为前缀的Object的分片上传事件。
for upload_info in oss2.MultipartUploadIterator(bucket, prefix='test'):
    print('key:', upload_info.key)
    print('upload_id:', upload_info.upload_id)
```

列举分片上传事件的更多详情，请参见ListMultipartUploads。

5.6.6. 进度条

进度条用于指示上传或下载的进度。

进度条的完整示例代码请参见GitHub。

下面的代码以bucket.put_object方法为例，介绍如何使用进度条。

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import os, sys
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 当无法确定待上传的数据长度时，total_bytes的值为None。
def percentage(consumed_bytes, total_bytes):
  if total_bytes:
    rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
    print('\r{0}% '.format(rate), end='')
    sys.stdout.flush()
# progress_callback为可选参数，用于实现进度条功能。
bucket.put_object('<yourObjectName>', 'a'*1024*1024, progress_callback=percentage)
```

5.6.7. 上传回调

本文介绍如何使用上传回调。在简单上传（put_object和put_object_from_file）以及完成分片上传（complete_multipart_upload）时支持使用上传回调。

> **说明** 上传回调的完整代码请参见GitHub。关于上传回调的更多信息，请参见开发指南中的上传回调和API参考中的Callback。

在上传字符串时使用上传回调（callback），目标存储空间为examplebucket，上传的目标文件为examplefiles文件夹下的exampleobject.txt文件，具体代码如下。

```
# -*- coding: utf-8 -*-
import json
import base64
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 定义回调参数Base64编码函数。
def encode_callback(callback_params):
    cb_str = json.dumps(callback_params).strip()
    return oss2.compat.to_string(base64.b64encode(oss2.compat.to_bytes(cb_str)))
# 设置上传回调参数。
callback_params = {}
# 设置回调请求的服务器地址，例如http://oss-demo.aliyuncs.com:23450。
callback_params['callbackUrl'] = 'http://oss-demo.aliyuncs.com:23450'
#（可选）设置回调请求消息头中Host的值，即您的服务器配置Host的值。
#callback_params['callbackHost'] = 'yourCallbackHost'
# 设置发起回调时请求body的值。
callback_params['callbackBody'] = 'bucket=${bucket}&object=${object}'
# 设置发起回调请求的Content-Type。
callback_params['callbackBodyType'] = 'application/x-www-form-urlencoded'
encoded_callback = encode_callback(callback_params)
# 设置发起回调请求的自定义参数，由Key和Value组成，Key必须以x:开始。
callback_var_params = {'x:my_var1': 'my_val1', 'x:my_var2': 'my_val2'}
encoded_callback_var = encode_callback(callback_var_params)
# 上传并回调。
params = {'callback': encoded_callback, 'callback-var': encoded_callback_var}
# 填写Object完整路径和字符串。Object完整路径中不能包含Bucket名称。
result = bucket.put_object('examplefiles/exampleobject.txt', 'a'*1024*1024, params)
```

5.7. 下载文件

5.7.1. 概述

OSS Python SDK提供了丰富的文件下载方式：

- 流式下载
- 下载到本地文件
- 范围下载
- 断点续传下载

下载过程中，您还可以通过下载进度条功能查看下载进度。

5.7.2. 流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

以下代码用于流式下载examplebucket中的exampleobject.txt文件。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# bucket.get_object的返回值是一个类文件对象（File-Like Object），同时也是一个可迭代对象（Iterable）。
# 填写Object的完整路径。Object完整路径中不能包含Bucket名称。
object_stream = bucket.get_object('exampleobject.txt')
print(object_stream.read())
# 由于get_object接口返回的是一个stream流，需要执行read()后才能计算出返回Object数据的CRC checksum，因此需要在调用该接口后进行CRC校验。
if object_stream.client_crc != object_stream.server_crc:
    print("The CRC checksum between client and server is inconsistent!")
```

以下代码用于将exampleobject.txt文件的流式数据下载到本地*D:\localpath*路径下的examplefile.txt。

```
import shutil
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# object_stream是类文件对象，您可以使用shutil.copyfileobj方法，将流式数据下载到本地文件中。
# 填写Object的完整路径。Object完整路径中不能包含Bucket名称。
object_stream = bucket.get_object('exampleobject.txt')
# 下载Object到本地文件，并保存到指定的本地路径中。如果指定的本地文件存在会覆盖，不存在则新建。
# 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
with open('D:\\localpath\\examplefile.txt', 'wb') as local_fileobj:
    shutil.copyfileobj(object_stream, local_fileobj)
```

以下代码用于将exampleobject.txt文件流式拷贝到另一个文件exampleobjectnew.txt中。

```
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# object_stream是一个可迭代对象，您可以将流式数据拷贝到同一Bucket中的另一个文件中。
# 填写Object的完整路径。Object完整路径中不能包含Bucket名称。
object_stream = bucket.get_object('exampleobject.txt')
# 填写另一个Object的完整路径。Object完整路径中不能包含Bucket名称。
bucket.put_object('exampleobjectnew.txt', object_stream)
```

5.7.3. 下载到本地文件

本文介绍如何将OSS文件（Object）下载到本地。

以下代码用于将指定的OSS文件下载到本地文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 下载OSS文件到本地文件。如果指定的本地文件存在会覆盖，不存在则新建。
#  <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
#  <yourObjectName>表示下载的OSS文件的完整名称，即包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>')
```

5.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

指定正常的下载范围

以下代码用于指定正常的下载范围来下载文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 对于1000字节大小的文件，正常的下载范围取值为0~999。
# 获取0~999字节范围内的数据，包括0和999，共1000个字节的数据。如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
object_stream = bucket.get_object('<yourObjectName>', byte_range=(0, 999))
```

指定异常的下载范围

假设现有大小为1000 Bytes的Object，则指定的正常下载范围应为0~999。如果指定范围不在有效区间，会导致Range不生效，响应返回值为200，并传送整个Object的内容。请求不合法的示例及返回说明如下：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。

兼容行为范围下载

在请求中增加请求头x-oss-range-behavior:standard，则改变指定范围不在有效区间时OSS的下载行为。假设现有大小为1000 Bytes的Object：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回HTTP Code为416，错误码为InvalidRange。

以下代码用于兼容行为范围下载：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建大小为1000 Bytes的object。
object_name = 'rangeTest.txt'
content = 'a' * 1000
bucket.put_object(object_name, content)
headers = {'x-oss-range-behavior':'standard'}
# 指定兼容行为。
# 若范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。
object_stream = bucket.get_object(object_name, byte_range=(500, 2000), headers=headers)
print('standard get 500~2000 http status code:', object_stream.status)
print('standard get 500~2000 contnet_length:', object_stream.content_length)
try:
  # 若范围首端取值不在有效区间，则抛出异常，返回HTTP Code为416，错误码为InvalidRange。
  object_stream = bucket.get_object(object_name, byte_range=(1000, 2000), headers=headers)
except oss2.exceptions.ServerError as e:
  print('standard get 1000~2000 http status code:', e.status)
  print('standard get 1000~2000 error code:', e.code)
```

5.7.5. 断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载的流程如下：

1. 在本地创建一个临时文件，文件名由原文件名加上一个随机的后缀组成。
2. 通过指定HTTP请求的Range头，按照范围读取OSS文件，并写入到临时文件里相应的位置。
3. 下载完成之后，把临时文件重命名为目标文件。如目标文件已存在会覆盖，不存在则新建。

您可以通过oss2.resumable_download方法断点续传下载，该方法中包含以下参数：

参数	描述	是否必需	默认值
bucket	存储空间名称。	是	无
key	OSS文件名称。	是	无
filename	本地文件。OSS文件将下载到该文件中。	是	无
store	记录本地分片下载结果的文件。下载过程中，断点信息会保存在此文件中，如果下载中断了，再次下载时会根据文件中记录的点继续下载。	否	HOME目录下建立的.py-oss-download目录。

参数	描述	是否必需	默认值
multipart_threshold	文件长度大于该值时，则使用断点续传下载。	否	10MB
part_size	分片大小。	否	自动计算
progress_callback	下载进度回调函数。	否	无
num_threads	并发下载的线程数。	否	1

以下代码用于断点续传下载：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
oss2.resumable_download(bucket, '<yourObjectName>', '<yourLocalFile>')
```

Python SDK 2.1.0以上版本支持设置可选参数进行断点续传下载，代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 请将oss2.defaults.connection_pool_size设成大于或等于线程数，并将part_size参数设成大于或等于oss2.defaults.multiget_part_size。
oss2.resumable_download(bucket, '<yourObjectName>', '<yourLocalFile>',
 store=oss2.ResumableDownloadStore(root='/tmp'),
 multiget_threshold=20*1024*1024,
 part_size=10*1024*1024,
 num_threads=3)
```

说明 避免多个程序（线程）同时调用该方法下载同一个源文件到同一个目标文件中。因为断点信息会在本地磁盘上互相覆盖，且临时文件名可能会冲突。

5.7.6. 进度条

进度条用于指示上传或下载的进度。

进度条的完整示例代码请参见GitHub。

下面的代码以bucket.get_object_to_file方法为例，介绍如何使用进度条。

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import os, sys
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 当HTTP响应头部没有Content-Length时，total_bytes的值为None。
def percentage(consumed_bytes, total_bytes):
  if total_bytes:
    rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
    print('\r{0}% '.format(rate), end='')
    sys.stdout.flush()
# progress_callback是可选参数，用于实现进度条功能。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>', progress_callback=percentage)
```

5.8. 管理文件

5.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理软链接

5.8.2. 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', 'examplebucket')
# 填写Object的完整路径，Object完整路径中不能包含Bucket名称。
exist = bucket.object_exists('exampleobject.txt')
# 返回值为true表示文件存在，false表示文件不存在。
if exist:
  print('object exist')
else:
  print('object not exist')
```

5.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	oss2.OBJECT_ACL_DEFAULT
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	oss2.OBJECT_ACL_PRIVATE
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	oss2.OBJECT_ACL_PUBLIC_READ
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	oss2.OBJECT_ACL_PUBLIC_READ_WRITE

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于设置指定文件的访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置文件的访问权限。
bucket.put_object_acl('<yourObjectName>', oss2.OBJECT_ACL_PUBLIC_READ)
```

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
print(bucket.get_object_acl('<yourObjectName>').acl)
```

5.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息。

详情请参见开发指南中的文件元信息。

设置HTTP header

以下代码用于设置HTTP header:

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
bucket.put_object('<yourObjectName>', '{"age": 1}', headers={'Content-Type': 'application/json; charset=utf-8'})
```

HTTP header详情请参见RFC2616。

设置自定义元信息

您可以自定义文件的元信息来对文件进行描述。

以下代码用于设置文件的自定义元信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
bucket.put_object('<yourObjectName>', 'a novel', headers={'x-oss-meta-author': 'O. Henry', 'Content-Type': 'application/json; charset=utf-8'})
```

说明 OSS使用以x-oss-meta-为前缀的参数作为用户自定义元信息header，详情请参见PutObject。

修改文件元信息

以下代码用于修改文件的元信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-author': 'O. Henry'})
# 每次调用bucket.update_object_meta都会清空用户自定义元信息，重新写入。
bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-price': '100 dollar'})
```

以下代码用于更改Content-Type等元信息：

```
bucket.update_object_meta('<yourObjectName>', {'x-oss-meta-author': 'O. Henry'})
# 每次调用bucket.update_object_meta都会清空用户自定义元信息，重新写入。
bucket.update_object_meta('<yourObjectName>', {'Content-Type': 'text/plain'})
```

获取文件元信息

您可以通过SDK提供的API获取文件元信息。

方法	描述	优势
get_object_meta	获取文件的ETag、Size（文件大小）、LastModified（最后修改时间）。	更轻量、更快
head_object	获取文件的全部元信息。	无

以下代码用于获取文件元信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取文件的部分元信息
simplifiedmeta = bucket.get_object_meta("<yourObjectName>")
print(simplifiedmeta.headers['Last-Modified'])
print(simplifiedmeta.headers['Content-Length'])
print(simplifiedmeta.headers['ETag'])
# 获取文件的全部元信息
objectmeta = bucket.head_object("<yourObjectName>")
print(objectmeta.headers['Content-Type'])
print(objectmeta.headers['Last-Modified'])
print(objectmeta.headers['x-oss-object-type'])
```

5.8.5. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何转换文件（Object）的存储类型。

有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

以下提供了详细的示例代码用于Object存储类型的相互转换。

- 以下代码用于将Object的存储类型从标准或低频访问转换为归档类型：

```
# -*- coding: utf-8 -*-
import oss2
import os
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。本示例中的Bucket与Object需提前创建好，且Object类型为标准或低频访问存储类型。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = '<yourObjectName>'
# 添加存储类型header，此处以更改文件存储类型为归档类型为例。
headers = {'x-oss-storage-class': oss2.BUCKET_STORAGE_CLASS_ARCHIVE}
# 更改文件存储类型。
bucket.copy_object(bucket.bucket_name, object_name, object_name, headers)
```

- 以下代码用于将Object的存储类型从归档转换为低频访问类型：

```
# -*- coding: utf-8 -*-
import oss2
import os
import time
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。本示例中的Bucket与Object需提前创建好，且Object类型为归档存储类型。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = '<yourObjectName>'
# 获取文件元信息。
meta = bucket.head_object(object_name)
# 查看文件存储类型是否为归档类型。如果是，则需要先解冻才能修改文件存储类型。解冻时间预计1分钟。
if meta.resp.headers['x-oss-storage-class'] == oss2.BUCKET_STORAGE_CLASS_ARCHIVE:
    bucket.restore_object(object_name)
    while True:
        meta = bucket.head_object(object_name)
        if meta.resp.headers['x-oss-restore'] == 'ongoing-request="true"':
            time.sleep(5)
        else:
            break
# 添加存储类型header，此处以更改文件存储类型为低频访问类型为例。
headers = {'x-oss-storage-class': oss2.BUCKET_STORAGE_CLASS_IA}
# 更改文件存储类型。
bucket.copy_object(bucket.bucket_name, object_name, object_name, headers)
```

各种存储类型的存储费用介绍，请参见计量项和计费项的计量计费概述一节。

5.8.6. 列举文件

本文介绍如何列举指定存储空间下（Bucket）的所有文件（Object）、指定前缀的文件、指定目录下的文件和子目录等。

背景信息

您可以调用GetBucket (ListObjects)或GetBucketV2 (ListObjectsV2)接口，一次性列举某个Bucket下最多1000个Object。您可以通过指定参数实现多种列举功能，例如通过指定参数列举指定起始位置后的所有文件、列举指定目录下的文件和子目录、以及列举结果分页。这两个接口的主要区别如下：

- 使用GetBucket (ListObjects)接口列举文件时，默认返回owner信息。
- 使用GetBucketV2 (ListObjectsV2)接口列举文件时，需通过fetchOwner指定是否在返回结果中包含owner信息。

> **说明** 对于开启版本控制的Bucket，建议使用GetBucketV2 (ListObjectsV2)接口列举文件。

以下分别介绍通过GetBucket (ListObjects)以及GetBucketV2 (ListObjectsV2)方法列举文件时涉及的参数说明。

- 通过GetBucket (ListObjects)方法列举文件

  GetBucket (ListObjects)涉及参数说明如下：

参数	描述
prefix	本次查询结果的前缀。
delimiter	对文件名称进行分组的字符。
marker	此次列举文件的起点。

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  GetBucketV2 (ListObjectsV2)涉及参数说明如下：

参数	描述
prefix	本次查询结果的前缀。
delimiter	对文件名称进行分组的字符。
startAfter	此次列举文件的起点。
fetchOwner	指定是否在返回结果中包含owner信息。 ○ true：表示返回结果中包含owner信息。 ○ false：表示返回结果中不包含owner信息。

简单列举

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定存储空间下的10个文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
from itertools import islice
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
for b in islice(oss2.ObjectIterator(bucket), 10):
    print(b.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  通过GetBucketV2 (ListObjectsV2)方法列举指定存储空间下的10个文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
from itertools import islice
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
for obj in islice(oss2.ObjectIteratorV2(bucket), 10):
    print(obj.key)
```

列举存储空间下所有文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定存储空间下所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举存储空间下所有文件。
for obj in oss2.ObjectIterator(bucket):
    print(obj.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  通过GetBucketV2 (ListObjectsV2)方法列举指定存储空间下所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举存储空间下所有文件。
for obj in oss2.ObjectIteratorV2(bucket):
    print(obj.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举文件，并返回文件的owner信息

  通过GetBucketV2 (ListObjectsV2)方法列举指定存储空间下所有文件，并通过指定fetch_owner参数返回文件的owner信息的完整示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举存储空间下所有文件，并通过指定fetch_owner参数返回owner信息。
for obj in oss2.ObjectIteratorV2(bucket, fetch_owner=True):
    print(obj.key)
    print('file owner display name: ' + obj.owner.display_name)
    print('file owner id: ' + obj.owner.id)
```

列举指定前缀的所有文件

假设存储空间中有4个文件： oss.jpg、fun/test.jpg、fun/movie/001.avi、fun/movie/007.avi，正斜线（/）作为文件夹的分隔符。

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定前缀的所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举fun文件夹下的所有文件，包括子目录下的文件。
for obj in oss2.ObjectIterator(bucket, prefix='fun/'):
    print(obj.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定前缀的所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举fun文件夹下的所有文件，包括子目录下的文件。
for obj in oss2.ObjectIteratorV2(bucket, prefix='fun/'):
    print(obj.key)
```

- 返回结果

```
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

列举指定起始位置后的所有文件

- 通过GetBucket (ListObjects)方法列举

  通过设置marker参数可以指定列举的起始位置，OSS将会返回字符串marker的字典序后的所有文件。假设存储空间中包含4个文件，分别为x1.txt、x2.txt、z1.txt和z2.txt。

  通过GetBucket (ListObjects)方法列举指定起始位置后的所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举指定字符串之后的所有文件。即使存储空间中存在marker的同名object，返回结果中也不会包含这个object。
for obj in oss2.ObjectIterator(bucket, marker="x2.txt"):
    print(obj.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过设置start_after参数可以指定列举的起始位置，OSS将会返回字符串start_after的字典序后的所有文件。假设存储空间中包含4个文件，分别为x1.txt、x2.txt、z1.txt和z2.txt。

  通过GetBucketV2 (ListObjectsV2)方法列举指定起始位置后的所有文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举指定字符串之后的所有文件。即使存储空间中存在start_after的同名object，返回结果中也不会包含这个object。
for obj in oss2.ObjectIteratorV2(bucket, start_after="x2.txt"):
    print(obj.key)
```

列举指定目录下的文件和子目录

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）。
- 如果再设置delimiter为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录）名称，子文件夹下的文件和文件夹不显示。

以下分别介绍如何通过GetBucket (ListObjects)和GetBucketV2 (ListObjectsV2)方法列举指定目录下的文件和子目录。

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定目录下的文件和子目录的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举fun文件夹下的文件与子文件夹名称，不列举子文件夹下的文件。
for obj in oss2.ObjectIterator(bucket, prefix = 'fun/', delimiter = '/'):
  # 通过is_prefix方法判断obj是否为文件夹。
  if obj.is_prefix():  # 判断obj为文件夹。
    print('directory: ' + obj.key)
  else:          # 判断obj为文件。
    print('file: ' + obj.key)
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定目录下的文件和子目录的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 列举fun文件夹下的文件与子文件夹名称，不列举子文件夹下的文件。如果不需要返回owenr信息可以不设置fetch_owner参数。
for obj in oss2.ObjectIteratorV2(bucket, prefix = 'fun/', delimiter = '/', fetch_owner=True):
  # 通过is_prefix方法判断obj是否为文件夹。
  if obj.is_prefix():  # 判断obj为文件夹。
    print('directory: ' + obj.key)
  else:          # 判断obj为文件。
    print('file: ' + obj.key)
    print('file owner display name: ' + obj.owner.display_name)
    print('file owner id: ' + obj.owner.id)
```

- 返回结果

```
directory: fun/movie/
file: fun/test.jpg
```

获取指定目录下的文件大小

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定目录下的文件大小的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
def CalculateFolderLength(bucket, folder):
  length = 0
  for obj in oss2.ObjectIterator(bucket, prefix=folder):
    length += obj.size
  return length
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
for obj in oss2.ObjectIterator(bucket, delimiter='/'):
  if obj.is_prefix():  # 判断obj为文件夹。
    length = CalculateFolderLength(bucket, obj.key)
    print('directory: ' + obj.key + '  length:' + str(length) + "Byte.")
  else: # 判断obj为文件。
    print('file:' + obj.key + '  length:' + str(obj.size) + "Byte.")
```

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  通过GetBucketV2 (ListObjectsV2)方法列举指定目录下的文件大小的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
def CalculateFolderLength(bucket, folder):
  length = 0
  for obj in oss2.ObjectIteratorV2(bucket, prefix=folder):
    length += obj.size
  return length
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
for obj in oss2.ObjectIteratorV2(bucket, delimiter='/'):
  if obj.is_prefix():  # 判断obj为文件夹。
    length = CalculateFolderLength(bucket, obj.key)
    print('directory: ' + obj.key + '  length:' + str(length) + "Byte.")
  else: # 判断obj为文件。
    print('file:' + obj.key + '  length:' + str(obj.size) + "Byte.")
```

5.8.7. 查询文件

本文主要介绍如何使用Python SDK的SelectObject查询CSV和JSON文件。

> **说明** 有关SelectObject的更多详情，请参见开发指南使用SelectObject查询文件及SelectObject API。

Python SDK示例

以下代码用于查询CSV和JSON文件：

```
import os2
```

```
import oss2
def select_call_back(consumed_bytes, total_bytes = None):
    print('Consumed Bytes:' + str(consumed_bytes) + '\n')
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourtBucketName>'
endpoint = '<yourEndpoint>'
# 创建存储空间，所有文件相关的方法都需要通过存储空间来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
key = 'python_select.csv'
content = 'Tom Hanks,USA,45\r\n'*1024
filename = 'python_select.csv'
# 上传CSV文件。
bucket.put_object(key, content)
# Select API的参数。
csv_meta_params = {'RecordDelimiter': '\r\n'}
select_csv_params = {'CsvHeaderInfo': 'None',
            'RecordDelimiter': '\r\n',
            'LineRange': (500, 1000)}
csv_header = bucket.create_select_object_meta(key, csv_meta_params)
print(csv_header.rows)
print(csv_header.splits)
result = bucket.select_object(key, "select * from ossobject where _3 > 44", select_call_back, select_csv_para
ms)
select_content = result.read()
print(select_content)
result = bucket.select_object_to_file(key, filename,
   "select * from ossobject where _3 > 44", select_call_back, select_csv_params)
bucket.delete_object(key)
###JSON DOCUMENT
key = 'python_select.json'
content = "{\"contacts\":[{\"key1\":1,\"key2\":\"hello world1\"},{\"key1\":2,\"key2\":\"hello world2\"}]}"
filename = 'python_select.json'
# 上传JSON DOCUMENT。
bucket.put_object(key, content)
select_json_params = {'Json_Type': 'DOCUMENT'}
result = bucket.select_object(key, "select s.key2 from ossobject.contacts[*] s where s.key1 = 1", None, select
_json_params)
select_content = result.read()
print(select_content)
result = bucket.select_object_to_file(key, filename,
   "select s.key2 from ossobject.contacts[*] s where s.key1 = 1", None, select_json_params)
bucket.delete_object(key)
###JSON LINES
key = 'python_select_lines.json'
content = "{\"key1\":1,\"key2\":\"hello world1\"}\n{\"key1\":2,\"key2\":\"hello world2\"}"
filename = 'python_select.json'
# 上传JSON LINE。
bucket.put_object(key, content)
select_json_params = {'Json_Type': 'LINES'}
json_header = bucket.create_select_object_meta(key,select_json_params)
print(json_header.rows)
print(json_header.splits)
result = bucket.select_object(key, "select s.key2 from ossobject s where s.key1 = 1", None, select_json_para
ms)
```

```
select_content = result.read()
print(select_content)
result = bucket.select_object_to_file(key, filename,
        "select s.key2 from ossobject s where s.key1 = 1", None, select_json_params)
bucket.delete_object(key)
```

Python Select API

以下内容是对Python Select API中的select_object、select_object_to_file、create_select_object_meta等元素的详细介绍。

- select_object
  - select_object 示例：

```
def select_object(self, key, sql,
            progress_callback=None,
            select_params=None
            byte_range=None
            headers=None
            ):
```

以上示例用于对指定Key的文件运行SQL，并返回查询结果。

- sql是原始的SQL字符串，无需做base64编码。
- Progress_callback是可选的用来汇报进度的回调函数。
- select_params用于指定select执行的各种参数以及行为。
- headers用于指定请求中附带的header信息，其行为和get-object一致。比如，对于CSV文件，在某些情况下可以用bytes来指定SQL查询在文件中的范围。

  - select_params支持的参数

参数名称	描述
Json_Type	■ 当Json_Type没有指定时，默认为CSV文件。 ■ 当Json_Type指定为DOCUMENT时，该文件为JSON DOCUMENT文件。 ■ 当Json_Type指定为LINES时，该文件为JSON LINE文件。
CsvHeaderInfo	CSV的header信息。 合法值为*None*、*Ignore*以及*Use*。 ■ *None*：该文件没有header信息。 ■ *Ignore*：该文件有header信息但未在SQL中使用。 ■ *Use*：该文件有Header信息且在sql语句中使用了Header中的列名。
CommentCharacter	CSV中的注释字符。仅支持一个字符，默认为None表示没有注释字符。
RecordDelimiter	CSV中的行分隔符，仅支持一个或两个字符。默认为\n。
OutputRecordDelimiter	Select输出结果中的行分隔符。默认为\n。
FieldDelimiter	CSV的列分隔符，仅支持一个字符，默认为逗号（,）。

参数名称	描述
OutputFieldDelimiter	Select输出结果中的列分隔符，默认为逗号（,）。
QuoteCharacter	CSV 列的引号字符，只支持一个字符，默认为双引号。引号内的行列分隔符被当做普通字符处理。
SplitRange	使用Split做分片查询。格式为(start, end)，此处为闭区间，表示查询范围从Split start#到end#。
LineRange	使用行做分片查询。格式为(start, end)，此处为闭区间，表示查询范围从行号start#到end#。
CompressionType	压缩类型，可以为GZIP。默认为None。
KeepAllColumns	该参数设置为true时表示原CSV文件中未在select列中出现的列将以空值输出（但保留列的位置）。默认为False。 如果CSV文件中的列为 firstname, lastname, age，SQL为 `select firstname, age from ossobject` 。 ■ 如果KeepAllColumns为*true*，则输出为firstname,,age（中间多一个逗号）。 ■ 如果KeepAllColumns为*false*，则输出为firstname,age。 **说明** 引入该选项的原因是让原本处理GetObject返回数据的代码可以在不用修改的情况下平移切换到SelectObject。
OutputRawData	■ 该参数为*True*时表示输出为Select数据，没有Frame的包装，表明很长时间不返回数据时会引起超时。 ■ 该参数为*False*时表示输出为Frame包装的数据。默认是False。
EnablePayloadCrc	为每个Frame计算CRC校验值，默认为False。
OutputHeader	仅用于CSV文件，表示输出结果中第一行是Header信息。
SkipPartialDataRecord	该参数为True时，如果CSV中某一列值不存在或者JSON中某一个Key不存在，则直接跳过整个记录。该参数为False时，则把该列当做null来处理。 假如某一行的列为firstname,lastname,age。SQL为 `select _1, _4 from ossobject` 。 ■ 如果为True，则直接跳过此行。 ■ 如果为False，则返回firstname,\n。
MaxSkippedRecordsAllowed	允许跳过的行的最大值。默认为0，表示一旦有一行跳过就返回错误。

参数名称	描述
ParseJsonNumberAsString	■ 当该参数为True时：表示JSON文件中的数字都解析为字符串。 ■ 当参数为False时：按照整数或者浮点数进行解析，False为默认值。 当JSON文件中含有高精度的浮点数时，直接解析为浮点数会丢失精度。如果想保留原始的精度，则可以设置该参数为True，并且在SQL语句中将该列Cast为decimal类型即可。

- select_object 返回值：返回SelectObjectResult对象，该对象支持read()函数以获得所有select结果。同时也支持__iter__方法。

  > **说明** 如果Select结果偏大，调用read()函数会阻塞直到select结果完全返回，同时占用过多的内存。 建议使用__iter__方法（foreach chunk in result），然后对每个chunk进行处理。__iter__方法不仅可以降低内存使用，且OSS服务器端每处理一个请求chunk客户端都能及时处理，不必等到全部结果返回后才处理。

- select_object_to_file

```
def select_object_to_file(self, key, filename, sql,
            progress_callback=None,
            select_params=None
            headers=None
            ):
```

以上示例用于对指定Key的文件运行SQL，将查询结果写入指定的文件。

涉及的参数描述与select_object相同。

- create_select_object_meta
  - create_select_object_meta的语法结构

```
def create_select_object_meta(self, key, select_meta_params=None, header=None):
```

以上示例用于对指定Key的文件进行创建或者获取select meta。select meta是指该文件的总行数、总列数（CSV文件）、总的Split数。

如果该文件已经创建过meta，则调用该函数不会重新创建，除非在参数中指定OverwriteIfExists为true。

创建select meta需要扫描整个文件。

- select_meta_params 中支持的参数

参数名称	描述
Json_Type	■ 当Json_Type没有指定时，默认为CSV文件。 ■ 若指定，必须为LINES，表示文件是JSON LINES。 说明 JSON DOCUMENT不支持该操作。
RecordDelimiter	CSV文件换行符。
FieldDelimiter	CSV文件列分隔符。
QuoteCharacter	CSV文件列引号符。引号符内的行列分隔符按普通字符处理。
CompressionType	压缩类型。目前不支持任何压缩类型，故只能为None。
OverwriteIfExists	覆盖原有的Select Meta。正常情况无需使用该选项。

- create_select_object_meta 返回值： GetSelectObjectMetaResult对象，包括rows、 splits 两个属性。对于CSV文件，其内部的select_resp对象还包括columns值，表示CSV文件的列数。

5.8.8. 删除文件

本文介绍如何删除文件。

警告 请您谨慎使用删除操作，文件一旦删除将无法恢复。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 删除文件。<yourObjectName>表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
# 如需删除文件夹，请将<yourObjectName>设置为对应的文件夹名称。如果文件夹非空，则需要将文件夹下的所有object删除后才能删除该文件夹。
bucket.delete_object('<yourObjectName>')
```

删除多个文件

```
# 批量删除3个文件。每次最多删除1000个文件。
result = bucket.batch_delete_objects(['<yourObjectName-a>', '<yourObjectName-b>', '<yourObjectName-c>
'])
# 打印成功删除的文件名。
print('\n'.join(result.deleted_keys))
```

删除指定前缀（prefix）的文件

以下代码用于删除指定前缀的文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运
维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
prefix = "<yourKeyPrefix>"
# 删除指定前缀的文件。
for obj in oss2.ObjectIterator(bucket, prefix=prefix):
    bucket.delete_object(obj.key)
```

5.8.9. 拷贝文件

本文介绍如何将一个存储空间（源存储空间）中的文件复制到同一地域下相同或不同存储空间（目标存储空间）中。

拷贝文件时注意事项如下：

- 用户必须有源Object的读权限及目标Bucket的读写权限。
- 不支持跨地域拷贝。例如不能将华东1（杭州）地域存储空间中的文件拷贝到华北1（青岛）地域。

简单拷贝

对于小于1GB的文件，您可以使用简单拷贝。以下代码用于通过简单拷贝将源存储空间srcexamplebucket中的srcexampleobject.txt文件拷贝到目标存储空间desexamplebucket中的desexampleobject.txt文件。：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录https://ram.console.aliyun.com创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
# 填写目标Bucket名称。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', 'desexamplebucket')
# 填写与目标Bucket处于同一地域的源Bucket名称。
# 填写源Object的完整路径。Object完整路径中不能包含Bucket名称。
# 填写目标Object的完整路径。Object完整路径中不能包含Bucket名称。
bucket.copy_object('srcexamplebucket', 'srcexampleobject.txt', 'desexampleobject.txt')
```

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

1. 通过bucket.init_multipart_upload初始化分片拷贝任务。
2. 通过bucket.upload_part_copy进行分片拷贝。除最后一个分片外，其它分片都要大于100KB。
3. 通过bucket.complete_multipart_upload提交分片拷贝任务。

以下代码用于将同一存储空间examplebucket中的srcexampleobject.txt文件拷贝到desexampleobject.txt文件：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import PartInfo
from oss2 import determine_part_size
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', 'examplebucket')
src_object = 'srcexampleobject.txt'
dst_object = 'desexampleobject.txt'
total_size = bucket.head_object(src_object).content_length
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 初始化分片。
upload_id = bucket.init_multipart_upload(dst_object).upload_id
parts = []
# 逐个分片拷贝。
part_number = 1
offset = 0
while offset < total_size:
    num_to_upload = min(part_size, total_size - offset)
    byte_range = (offset, offset + num_to_upload - 1)
    result = bucket.upload_part_copy(bucket.bucket_name, src_object, byte_range,dst_object, upload_id, part_number)
    parts.append(PartInfo(part_number, result.etag))
    offset += num_to_upload
    part_number += 1
# 完成分片拷贝。
bucket.complete_multipart_upload(dst_object, upload_id, parts)
```

关于分片拷贝的更多信息，请参见UploadPartCopy。

5.8.10. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传、拷贝文件及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 上传文件。
# 指定PutObject操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers["x-oss-forbid-overwrite"] = "true"
result = bucket.put_object('<yourObjectName>', 'content of object', headers=headers)
# HTTP返回码。
print('http status: {0}'.format(result.status))
# 请求ID。请求ID是请求的唯一标识，强烈建议在程序日志中添加此参数。
print('request_id: {0}'.format(result.request_id))
# ETag是put_object方法返回值特有的属性。
print('ETag: {0}'.format(result.etag))
# HTTP响应头部。
print('date: {0}'.format(result.headers['date']))
```

简单上传的更多详情，请参见PutObject。

拷贝文件

- 拷贝小文件

  以下代码用于拷贝小文件时禁止覆盖同名文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourDestinationBucketName>')
# 指定copy_object操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers = dict()
headers["x-oss-forbid-overwrite"] = "true"
bucket.copy_object('<yourSourceBucketName>', '<yourSourceObjectName>', '<yourDestinationObjectName>', headers=headers)
```

- 拷贝大文件

  以下代码用于拷贝大文件（分片拷贝）时禁止覆盖同名文件：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import PartInfo
from oss2 import determine_part_size
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
src_object = '<yourSourceObjectName>'
dst_object = '<yourDestinationObjectName>'
total_size = bucket.head_object(src_object).content_length
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 初始化分片。
# 指定拷贝文件操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers = dict()
headers["x-oss-forbid-overwrite"] = "true"
upload_id = bucket.init_multipart_upload(dst_object, headers=headers).upload_id
parts = []
# 逐个分片拷贝。
part_number = 1
offset = 0
while offset < total_size:
  num_to_upload = min(part_size, total_size - offset)
  byte_range = (offset, offset + num_to_upload - 1)
  result = bucket.upload_part_copy(bucket.bucket_name, src_object, byte_range,dst_object, upload_id, part_number)
  parts.append(PartInfo(part_number, result.etag))
  offset += num_to_upload
  part_number += 1
# 完成分片拷贝。
# 指定拷贝文件操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers = dict()
headers["x-oss-forbid-overwrite"] = "true"
bucket.complete_multipart_upload(dst_object, upload_id, parts, headers=headers)
```

拷贝文件的更多详情，请参见CopyObject。

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
# -*- coding: utf-8 -*-
import os
from oss2 import SizedFileAdapter, determine_part_size
from oss2.models import PartInfo
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
filename = '<yourLocalFile>'
total_size = os.path.getsize(filename)
# determine_part_size方法用来确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 初始化分片。
# 指定分片上传操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers["x-oss-forbid-overwrite"] = "true"
upload_id = bucket.init_multipart_upload(key, headers=headers).upload_id
parts = []
# 逐个上传分片。
with open(filename, 'rb') as fileobj:
  part_number = 1
  offset = 0
  while offset < total_size:
    num_to_upload = min(part_size, total_size - offset)
    # SizedFileAdapter(fileobj, size)方法会生成一个新的文件对象，重新计算起始追加位置。
    result = bucket.upload_part(key, upload_id, part_number,
                    SizedFileAdapter(fileobj, num_to_upload))
    parts.append(PartInfo(part_number, result.etag))
    offset += num_to_upload
    part_number += 1
# 完成分片上传。
# 指定分片上传操作时是否覆盖同名Object。
# 不指定x-oss-forbid-overwrite时，默认覆盖同名Object。
# 指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object。
# 指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
headers["x-oss-forbid-overwrite"] = "true"
bucket.complete_multipart_upload(key, upload_id, parts, headers=headers)
# 验证分片上传。
with open(filename, 'rb') as fileobj:
  assert bucket.get_object(key).read() == fileobj.read()
```

分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

5.8.11. 解冻文件

归档或冷归档类型的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档和冷归档文件。

> **说明** 非归档或冷归档类型的文件，不要调用RestoreObject方法。
>
> 归档及冷归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
objectName = '<yourObjectName>'
bucket.restore_object(objectName)
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。
   - 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
   - 批量（Bulk）：表示5~12小时内完成解冻。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import (RestoreJobParameters,
                         RestoreConfiguration,
                         RESTORE_TIER_EXPEDITED,
                         RESTORE_TIER_STANDARD,
                         RESTORE_TIER_BULK)
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = "<yourColdArchiveObjectName>"
# 如需上传文件的同时指定文件的存储类型为冷归档类型，请参考以下代码。
# bucket.put_object(object_name, '<yourContent>', headers={"x-oss-storage-class": oss2.BUCKET_STORAGE_CLASS_COLD_ARCHIVE})
# 指定解冻ColdArchive（冷归档）类型文件的优先级。
# RESTORE_TIER_EXPEDITED: 1个小时之内解冻完成。
# RESTORE_TIER_STANDARD: 2~5小时之内解冻完成。
# RESTORE_TIER_BULK: 5~12小时之内解冻完成。
job_parameters = RestoreJobParameters(RESTORE_TIER_STANDARD)
# 配置解冻参数，以设置5小时内解冻完成，解冻状态保持2天为例。
# days表示解冻之后保持解冻状态的天数，默认为1天，此参数适用于解冻Archive与ColdArchive类型文件。
# job_parameters表示解冻优先级配置，此参数只适用于解冻ColdArchive类型的文件。
restore_config= RestoreConfiguration(days=2, job_parameters=job_parameters)
# 发起解冻请求。
bucket.restore_object(object_name, input=restore_config)
```

5.8.12. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
objectName = '<yourObjectName>'
symlink = "<yourSymlink>";
bucket.put_symlink(objectName, symlink)
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
symlink = "<yourSymlink>";
bucket.get_symlink(symlink)
```

获取软链接的详细信息请参见GetSymlink。

5.9. 版本控制

5.9.1. 管理版本控制

版本控制应用于存储空间（Bucket）内的所有文件（Object）。通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。有关版本控制状态的更多信息，请参见版本控制介绍。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import BucketVersioningConfig
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建bucket版本控制配置。
config = BucketVersioningConfig()
# 状态配置为Enabled或Suspended。
config.status = oss2.BUCKET_VERSIONING_ENABLE
# 设置bucket版本控制状态。
result = bucket.put_bucket_versioning(config)
# 查看http返回码。
print('http response code:', result.status)
```

有关设置Bucket版本控制状态的更多信息，请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取bucket版本控制状态信息。
versioning_info = bucket.get_bucket_versioning()
# 查看bucket版本控制状态, 如果曾开启过版本控制则返回Enabled或Suspended, 如果从未开启过版本控制则返回None。
print('bucket versioning status:', versioning_info.status)
```

有关获取Bucket版本控制状态的更多信息，请参见Get Bucket Versioning。

5.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中使用简单上传、追加上传或者分片上传的方式来上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本ID为“null”。

以下代码用于简单上传：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 上传文件。
result = bucket.put_object('<yourObjectName>', 'content of object')
# HTTP返回码。
print('http response code: {0}'.format(result.status))
# 查看本次上传Object的版本ID。
print('put object version:', result.versionid)
```

简单上传的详细信息请参见Put Object。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

说明

- 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
- 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
- 不支持对当前版本为非Appendable类型的Object（包括Normal Object、Delete Marker等）执行AppendObject操作。

以下代码用于追加上传：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 设置首次上传的追加位置（Position参数）为0。
result = bucket.append_object('<yourObjectName>', 0, 'content of first append')
# 查看本次执行追加文件的Object的版本ID。
print('append object versionid:', result.versionid)
# 如果不是首次上传，可以通过bucket.head_object方法或上次追加返回值的next_position属性，得到追加位置。
bucket.append_object('<yourObjectName>', result.next_position, 'content of second append')
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2 import SizedFileAdapter, determine_part_size
from oss2.models import PartInfo
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key = '<yourObjectName>'
filename = '<yourLocalFile>'
total_size = os.path.getsize(filename)
# determine_part_size方法用来确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 初始化分片。
upload_id = bucket.init_multipart_upload(key).upload_id
parts = []
# 逐个上传分片。
with open(filename, 'rb') as fileobj:
  part_number = 1
  offset = 0
  while offset < total_size:
    num_to_upload = min(part_size, total_size - offset)
  # SizedFileAdapter(fileobj, size)方法会生成一个新的文件对象，重新计算起始追加位置。
    result = bucket.upload_part(key, upload_id, part_number,
                    SizedFileAdapter(fileobj, num_to_upload))
    parts.append(PartInfo(part_number, result.etag))
    offset += num_to_upload
    part_number += 1
# 完成分片上传。
result = bucket.complete_multipart_upload(key, upload_id, parts)
# 查看response中携带的上传文件的versionid
print('result.versionid:', result.versionid)
# 验证分片上传。
with open(filename, 'rb') as fileobj:
  assert bucket.get_object(key).read() == fileobj.read()
```

分片上传的详细信息请参见CompleteMultipartUpload。

5.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 下载指定版本的object。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
object_stream = bucket.get_object('<yourObjectName>', params=params)
# 读取下载的object内容。
read_content = object_stream.read()
print('get object content:', read_content)
# 查看本次下载的object的版本id。
print('get object versionid:', object_stream.versionid)
# 由于get_object接口返回的是一个stream流，需要执行read()后才能计算出返回Object数据的CRC checksum，因此需要在调用该接口后做CRC校验。
if object_stream.client_crc != object_stream.server_crc:
  print "The CRC checksum between client and server is inconsistent!"
```

下载文件的详细信息请参见Get Object。

5.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的版本ID，此版本ID将会在响应header的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为“null”的版本，且会覆盖原先versionId为“null”的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 拷贝指定版本的文件。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
result = bucket.copy_object('<yourBucketName>', '<yourSourceObjectName>', '<yourDestinationObjectName>', params=params)
#查看目的地object的versionid。
print('result.versionid:', result.versionid)
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2 import determine_part_size
from oss2.models import PartInfo
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
src_object_name = '<yourSrcObject>'
dest_object_name = '<yourDestObject>'
params = dict()
params['versionId'] = '<yourObjectVersionId>'
# 获取指定版本的源文件的文件大小。
head_info = bucket.head_object(src_object_name, params=params)
total_size = head_info.content_length
print('src object size:', total_size)
# determine_part_size方法用来确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
print('part_size:', part_size)
# 初始化分片。
upload_id = bucket.init_multipart_upload(dest_object_name).upload_id
parts = []
# 逐个上传分片。
part_number = 1
offset = 0
while offset < total_size:
  num_to_upload = min(part_size, total_size - offset)
  end = offset + num_to_upload - 1;
  result = bucket.upload_part_copy(bucket.bucket_name, src_object_name, (offset, end), dest_object_name, upload_id, part_number, params=params)
  # 保存part信息。
  parts.append(PartInfo(part_number, result.etag))
  offset += num_to_upload
  part_number += 1
# 完成分片上传。
result = bucket.complete_multipart_upload(dest_object_name, upload_id, parts)
# 查看拷贝生成的目标文件的versionId。
print('result version id:', result.versionid)
# 获取文件元信息。
head_info = bucket.head_object(dest_object_name)
# 查看目标文件大小。
dest_object_size = head_info.content_length
print('dest object size:', dest_object_size)
# 对比源文件大小。
assert dest_object_size == total_size
```

分片拷贝的详细信息请参见UploadPartCopy。

5.9.5. 列举文件

本文介绍如何在开启版本控制状态下列举存储空间下（Bucket）的所有文件（Object）、指定个数的文件、指定前缀的文件等。

列举Bucket中所有Object的信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启Bucket版本控制后，调用list_object_versions接口返回不同版本的Object信息。
# 列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。
result = bucket.list_object_versions()
# 列举所有Object的版本信息。
next_key_marker = None
next_versionid_marker = None
while True:
  result = bucket.list_object_versions(key_marker=next_key_marker, versionid_marker=next_versionid_marker)
  # 查看列举Object的版本信息。
  for version_info in result.versions:
    print('version_info.versionid:', version_info.versionid)
    print('version_info.key:', version_info.key)
    print('version_info.is_latest:', version_info.is_latest)
  # 查看列举删除标记的版本信息。
  for del_maker_Info in result.delete_marker:
    print('del_maker.key:', del_maker_Info.key)
    print('del_maker.versionid:', del_maker_Info.versionid)
    print('del_maker.is_latest:', version_info.is_latest)
  is_truncated = result.is_truncated
  # 查看列举结果是否完整。如果结果不完整，则继续罗列。如果结果已完整，则退出循环。
  if is_truncated:
    next_key_marker = result.next_key_marker
    next_versionid_marker = result.next_versionid_marker
  else:
    break
```

列举指定前缀Object的版本信息

以下代码用于列举指定前缀Object的版本信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM账号创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启Bucket版本控制后，调用list_object_versions接口返回不同版本的Object信息。
# 列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。
result = bucket.list_object_versions()
# 指定列举以test-为前缀的Object的版本信息。
prefix = 'test-'
next_key_marker = None
next_versionid_marker = None
while True:
  result = bucket.list_object_versions(prefix=prefix, key_marker=next_key_marker, versionid_marker=next_versionid_marker)
  # 查看列举的Object版本信息。
  for version_info in result.versions:
    print('version_info.versionid:', version_info.versionid)
    print('version_info.key:', version_info.key)
    print('version_info.is_latest:', version_info.is_latest)
  # 查看列举的删除标记版本信息。
  for del_maker_Info in result.delete_marker:
    print('del_maker.key:', del_maker_Info.key)
    print('del_maker.versionid:', del_maker_Info.versionid)
    print('del_maker.is_latest:', version_info.is_latest)
  is_truncated = result.is_truncated
  # 查看列举结果是否完整。如果结果不完整，则继续罗列。如果结果已完整，则退出循环。
  if is_truncated:
    next_key_marker = result.next_key_marker
    next_versionid_marker = result.next_versionid_marker
  else:
    break
```

列举指定个数Object的版本信息

以下代码用于列举指定个数Object的版本信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM账号创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启Bucket版本控制后，调用list_object_versions接口返回不同版本的Object信息。
# 列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。
result = bucket.list_object_versions()
# 指定本次最多返回200个结果。
max_keys = 200
result = bucket.list_object_versions(max_keys=max_keys)
# 查看列举的Object版本信息。
for version_info in result.versions:
  print('version_info.versionid:', version_info.versionid)
  print('version_info.key:', version_info.key)
  print('version_info.is_latest:', version_info.is_latest)
# 查看列举的Object删除标记的版本信息。
for del_maker_Info in result.delete_marker:
  print('del_maker.key:', del_maker_Info.key)
  print('del_maker.versionid:', del_maker_Info.versionid)
  print('del_maker.is_latest:', version_info.is_latest)
# 查看列举结果是否有截断。
# 由于指定本次列举最多返回200个结果，若Bucket中Object的个数超过200，则列举结果存在截断，即is_truncated为True。若Bucket中Object的个数少于200，则列举结果没有截断，即is_truncated为False。
print('is truncated', result.is_truncated)
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为Object。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

假设存储空间（examplebucket）中包含文件*oss.jpg*、*fun/test.jpg*、*fun/movie/001.avi*和*fun/movie/007.txt*，以正斜线（/）作为文件夹的分隔符。文件结构如下：

```
examplebucket
 └──── oss.jpg
 └──── fun
    └──── test.jpg
    └──── movie
       └──── 001.avi
       └──── 007.txt
```

以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举根目录下的Object的版本信息

  以下代码用于列举根目录下的Object的版本信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM用户创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启Bucket版本控制后，调用list_object_versions接口返回不同版本的Object信息。
# 列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。
result = bucket.list_object_versions()
# 指定delimiter为正斜线（/）。
delimiter = "/"
next_key_marker = None
next_versionid_marker = None
while True:
  result = bucket.list_object_versions(delimiter=delimiter, key_marker=next_key_marker, versionid_marker=next_versionid_marker)
  # 查看列举的Object版本信息。
  for version_info in result.versions:
    print('version_info.versionid:', version_info.versionid)
    print('version_info.key:', version_info.key)
    print('version_info.is_latest:', version_info.is_latest)
  # 查看列举的删除标记版本信息。
  for del_maker_Info in result.delete_marker:
    print('del_maker.key:', del_maker_Info.key)
    print('del_maker.versionid:', del_maker_Info.versionid)
    print('del_maker.is_latest:', version_info.is_latest)
  # 查看以正斜线（/）结尾的目录名称。
  for common_prefix in result.common_prefix:
    print("common_prefix:", common_prefix)
  is_truncated = result.is_truncated
  # 查看列举结果是否完整。如果结果不完整，则继续罗列。如果结果已完整，则退出循环。
  if is_truncated:
    next_key_marker = result.next_key_marker
    next_versionid_marker = result.next_versionid_marker
  else:
    break
```

  返回结果：

```
('version_info.versionid:', 'CAEQEhiBgMCw8Y7FqBciIGIzMDE3MTEzOWRiMDRmZmFhMmRlMjljZWI0MWU4*****')
('version_info.key:', 'oss.jpg')
('version_info.is_latest:', True)
('common_prefix:', 'fun/')
```

- 列举目录下的文件和子目录

  以下代码用于列举指定目录下的文件和子目录：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM账号创建RAM用户。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 开启Bucket版本控制后，调用list_object_versions接口返回不同版本的Object信息。
# 列出Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。
result = bucket.list_object_versions()
# 指定delimiter为正斜线（/），prefix为fun/。
prefix = "fun/"
delimiter = "/"
next_key_marker = None
next_versionid_marker = None
while True:
  result = bucket.list_object_versions(prefix=prefix, delimiter=delimiter, key_marker=next_key_marker, versionid_marker=next_versionid_marker)
  # 查看列举的Object版本信息。
  for version_info in result.versions:
    print('version_info.versionid:', version_info.versionid)
    print('version_info.key:', version_info.key)
    print('version_info.is_latest:', version_info.is_latest)
  # 查看列举的删除标记的版本信息。
  for del_maker_Info in result.delete_marker:
    print('del_maker.key:', del_maker_Info.key)
    print('del_maker.versionid:', del_maker_Info.versionid)
    print('del_maker.is_latest:', version_info.is_latest)
  # 查看以正斜线（/）结尾的文件夹名称。
  for common_prefix in result.common_prefix:
    print("common_prefix:", common_prefix)
  is_truncated = result.is_truncated
  # 查看列举结果是否完整。如果结果不完整，则继续罗列。如果结果已完整，则退出循环。
  if is_truncated:
    next_key_marker = result.next_key_marker
    next_versionid_marker = result.next_versionid_marker
  else:
    break
```

返回结果：

```
('version_info.versionid:', 'CAEQFRiBgMCh9JDkrxciIGE3OTNkYzFhYTc2YzQzOTQ4Y2MzYjg2YjQ4ODg*****'
)
('version_info.key:', 'fun/test.jpg')
('version_info.is_latest:', True)
('commonPrefix:', 'fun/movie/')
```

5.9.6. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

  如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

  `x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object进行永久删除：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
object_name = '<yourObjectName>'
# 指定Object的versionId，也可以是删除标记的versionId。
params = dict()
params['versionId'] = '<yourObjectVersionIdOrDeleteMarkerVersionId>'
# 删除指定versionId的Object或指定versionId的删除标记关联的Object。
result = bucket.delete_object(object_name, params=params)
print("delete object name: ", object_name)
# 如果指定的是Object的versionId，则返回的delete_marker为None且返回的versionId为指定Object的versionId。
# 如果指定的是删除标记的versionId，则返回的delete_marker为True且返回的versionId为指定删除标记的versionId。
if result.delete_marker:
    print("delete del-marker versionid: ",result.versionid)
else:
    print("delete object versionid:", result.versionid)
```

- 临时删除

  以下代码用于不指定versionId对Object进行临时删除：

```
# -*- coding: utf-8 -*-
import os
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = '<yourObjectName>'
# 不指定versionId对Object进行临时删除，此操作会为Object添加删除标记。
result = bucket.delete_object(object_name)
# 查看删除标记。
print("delete marker: ", result.delete_marker)
# 查看返回删除标记的versionId。
print("delete marker versionid: ", result.versionid)
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于指定versionId对多个Object及删除标记进行永久删除：

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2.models import BatchDeleteObjectVersion
from oss2.models import BatchDeleteObjectVersionList
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
version_list = BatchDeleteObjectVersionList()
# 可以传入Object或者删除标记的versionId。
obj1_versionid = '<yourObject1VersionId>'
obj1_del_marker_versionid = '<yourObject1DelMarkerVersionId>'
obj2_versionid = '<yourObject2VersionId>'
obj2_del_marker_versionid = '<yourObject2DelMarkerVersionId>'
version_list.append(BatchDeleteObjectVersion(key='<yourObject1Name>', versionid=obj1_versionid))
version_list.append(BatchDeleteObjectVersion(key='<yourObject1Name>', versionid=obj1_del_marker_versionid))
version_list.append(BatchDeleteObjectVersion(key='<yourObject2Name>', versionid=obj2_versionid))
version_list.append(BatchDeleteObjectVersion(key='<yourObject2Name>', versionid=obj2_del_marker_versionid))
# 批量删除指定versionId的Object或删除标记。
result = bucket.delete_object_versions(version_list)
# 查看被删除Object或删除标记的versionId。
for del_version in result.delete_versions:
  print('del object name:', del_version.key)
  # 查看删除的是否是删除标记。
  print('Is del marker:', del_version.delete_marker)
  # 如果是删除标记，则打印删除的删除标记的versionId，否则打印删除Object的versionId。
  if del_version.delete_marker :
    print('del object del_marker.versionid', del_version.delete_marker_versionid)
  else:
    print('del object versionid:', del_version.versionid)
```

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除：

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2.models import BatchDeleteObjectVersion
from oss2.models import BatchDeleteObjectVersionList
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
key_list = ['<yourObject1Name>', '<yourObject2Name>']
# 执行不指定versionId的删除操作后，将为Object添加删除标记。
result = bucket.batch_delete_objects(key_list)
for del_version in result.delete_versions:
    print('key name:', del_version.key)
    # 打印返回的删除标记。
    print('Is del marker:', del_version.delete_marker)
    print('key del_marker.versionid', del_version.delete_marker_versionid)
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

删除指定前缀（prefix）的文件

以下代码用于删除指定前缀的文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
prefix = "<yourKeyPrefix>"
# 列举所有指定前缀文件的versionId并删除这些文件。
next_key_marker = None
next_versionid_marker = None
while True:
    result = bucket.list_object_versions(prefix=prefix, key_marker=next_key_marker, versionid_marker=next_versionid_marker)
    for version_info in result.versions:
        bucket.delete_object(version_info.key, params={'versionId': version_info.versionid})
    for del_marker_info in result.delete_marker:
        bucket.delete_object(del_marker_info.key, params={'versionId': del_marker_info.versionid})
    is_truncated = result.is_truncated
    if is_truncated:
        next_key_marker = result.next_key_marker
        next_versionid_marker = result.next_versionid_marker
    else:
        break
```

5.9.7. 解冻文件

在受版本控制的存储空间（Bucket）中，Object的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 填充versionId字段到param。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
#填写一个归档类型的object的名称，如果没有请先调用put_objec接口创建。
object_name = 'yourArchiveObjectName'
# 获取指定版本的object元信息。
meta = bucket.head_object(object_name, params=params)
# 判断文件存储类型是否为归档类型。
if meta.resp.headers['x-oss-storage-class'] == oss2.BUCKET_STORAGE_CLASS_ARCHIVE:
    # 解冻指定版本的object。
    result = bucket.restore_object(object_name, params=params)
    # 查看执行解冻操作的object的版本id。
    print('restore object version id:', result.versionid)
    while True:
        meta = bucket.head_object(object_name, params=params)
        print('x-oss-restore:', meta.resp.headers['x-oss-restore'])
        # 判断解冻状态。
        if meta.resp.headers['x-oss-restore'] == 'ongoing-request="true"':
            time.sleep(5)
        else:
            break
```

解冻文件的详细信息请参见RestoreObject。

5.9.8. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的meta信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的meta信息。

以下代码用于获取文件元信息：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 填充versionId字段到param。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
# 获取文件的部分元信息。
simplifiedmeta = bucket.get_object_meta("<yourObjectName>", params=params)
print(simplifiedmeta.headers['Last-Modified'])
print(simplifiedmeta.headers['Content-Length'])
print(simplifiedmeta.headers['ETag'])
# 获取文件的全部元信息。
objectmeta = bucket.head_object("<yourObjectName>", params=params)
print(objectmeta.headers['Content-Type'])
print(objectmeta.headers['Last-Modified'])
print(objectmeta.headers['x-oss-object-type'])
```

获取文件元信息的详情请参见HeadObject、GetObjectMeta。

5.9.9. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 给指定版本的object修改权限，此处以修改为公共读权限为例。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
result = bucket.put_object_acl(<'yourObjectName'>, oss2.OBJECT_ACL_PUBLIC_READ, params = params)
# 查看本次修改权限操作的object的版本id。
print('set acl object versionid:', result.versionid)
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取指定版本的object权限。
params = dict()
params['versionId'] = '<yourObjectVersionId>'
result = bucket.get_object_acl(<'yourObjectName'>, params = params)
# 查看获取到的指定版本object的权限。
print('get object acl :', result.acl)
# 查看本次执行获取权限操作的object的版本id。
print('object version id:', result.versionid)
```

获取文件访问权限的详细信息请参见GetObjectACL。

5.9.10. 管理软链接

软链接功能用于快速访问对象存储空间内的常用文件。设置软链接后，您可以通过软链接文件快速打开源文件，类似于Windows的快捷方式。本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

软链接本身也可以有多个版本，每个不同的版本可以指向不同的TargetObject，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。您可以通过创建软链接指向TargetObject的当前版本。

> **说明** 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

以下代码用于创建软链接：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
objectName = '<yourTargetObjectName>'
symlink = '<yourSymlink>'
# 上传软链接。
result = bucket.put_symlink(objectName, symlink)
# 查看上传的软链接的版本id。
print('symlink versionid:', result.versionid)
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
symlink = '<yourSymlink>'
# 获取指定版本的软链接文件。
params = dict()
params['versionId'] = ‘yourSymlinkVersionId>’
result = bucket.get_symlink(symlink, params=params)
# 查看返回的软链接文件的版本id。
print('get symlink versionid:', result.versionid)
```

获取软链接的详细信息请参见GetSymlink。

5.10. 对象标签

5.10.1. 设置对象标签

OSS支持使用对象标签（Object Tagging）对存储空间（Bucket）中的文件（Object）进行分类，您可以针对相同标签的Object设置生命周期规则、访问权限等。

背景信息

设置对象标签时，请注意以下事项：

- 您可以在上传Object时设置对象标签，也可以对已上传Object设置对象标签。设置对象标签时，如果对象已有标签，则覆盖原标签。关于设置对象标签的更多信息，请参见PutObjectTagging。
- 设置对象标签时，您要有PutObjectTagging权限。
- 更改标签时不会更新Object的Last-Modified时间。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/

  > **说明** 通过HTTP header的方式设置标签且标签中包含任意字符时，您可以对标签的Key和Value做URL编码。

对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见对象标签。

上传Object时添加对象标签

- 简单上传时添加对象标签

  以下代码用于简单上传Object时添加对象标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&” + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 调用put_object接口时指定headers，将会为上传的文件添加标签。
result = bucket.put_object(object_name, 'content', headers=headers)
print('http response status: ', result.status)
# 查看Object的标签信息。
result = bucket.get_object_tagging(object_name)
for key in result.tag_set.tagging_rule:
    print('tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

- 分片上传时添加对象标签

  以下代码用于通过multipart_upload方式上传Object时添加对象标签。

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2 import SizedFileAdapter, determine_part_size
from oss2.models import PartInfo
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
filename = 'D:\\localpath\\examplefile.txt'
total_size = os.path.getsize(filename)
# determine_part_size方法用于确定分片大小。
```

```
# determine_part_size方法用于确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&" + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 初始化分片。
# 调用init_multipart_upload接口时指定headers，将会给上传的文件添加标签。
upload_id = bucket.init_multipart_upload(object_name, headers=headers).upload_id
parts = []
# 逐个上传分片。
with open(filename, 'rb') as fileobj:
  part_number = 1
  offset = 0
  while offset < total_size:
    num_to_upload = min(part_size, total_size - offset)
    # SizedFileAdapter(fileobj, size)方法会生成一个新的文件对象，重新计算起始追加位置。
    result = bucket.upload_part(object_name, upload_id, part_number,
                    SizedFileAdapter(fileobj, num_to_upload))
    parts.append(PartInfo(part_number, result.etag))
    offset += num_to_upload
    part_number += 1
# 完成分片上传。
result = bucket.complete_multipart_upload(object_name, upload_id, parts)
print('http response status: ', result.status)
# 查看Object的标签信息。
result = bucket.get_object_tagging(object_name)
for key in result.tag_set.tagging_rule:
  print('tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
# 验证分片上传。
with open(filename, 'rb') as fileobj:
  assert bucket.get_object(object_name).read() == fileobj.read()
```

- 追加上传时添加对象标签

  以下代码用于通过append_object方式上传Object时添加对象标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&” + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 追加上传文件。调用append_object接口时指定headers，将会给文件设置标签。
# 只有第一次调用append_object设置的标签才会生效，后续使用此种方式添加的标签不生效。
result = bucket.append_object(object_name, 0, '<yourContent>', headers=headers)
# 查看Object的标签信息。
result = bucket.get_object_tagging(object_name)
for key in result.tag_set.tagging_rule:
    print('tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

- 断点续传上传时添加对象标签

  以下代码用于通过resumable_upload方式上传Object时添加对象标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
local_file = 'D:\\localpath\\examplefile.txt'
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&" + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 当文件长度大于或等于可选参数multipart_threshold（默认值为10 MB）时使用分片上传。如果未使用参数store指定目录，则会在HOME目录下建立.py-oss-upload目录来保存断点信息。
# 调用resumable_upload接口时指定headers，将会给上传的文件添加标签。
oss2.resumable_upload(bucket, object_name, local_file, headers=headers)
result = bucket.get_object_tagging(object_name)
for key in result.tag_set.tagging_rule:
    print('object tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

为已上传Object添加或更改对象标签

如果上传Object时未添加对象标签或者添加的对象标签不满足使用需求，您可以在上传Object后为Object添加或更改对象标签。

以下代码用于为已上传Object添加或更改对象标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import Tagging, TaggingRule
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
#填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 创建标签规则。
rule = TaggingRule()
rule.add('key1', 'value1')
rule.add('key2', 'value2')
# 创建标签。
tagging = Tagging(rule)
# 设置标签。
result = bucket.put_object_tagging(object_name, tagging)
# 查看HTTP返回码。
print('http response status:', result.status)
```

为Object指定版本添加或更改对象标签

在已开启版本控制的Bucket中，通过指定Object的版本ID（versionId），您可以为Object指定版本添加或更改对象标签。

以下代码用于为Object指定版本添加或更改对象标签。

> **说明** 关于获取versionId的具体操作，请参见列举文件。

```
import oss2
from oss2.models import Tagging
from oss2.models import BucketVersioningConfig
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
#填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 填写Object的版本ID，例如CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****
。
version_id = 'CAEQMxiBgICAof2D0BYiIDJhMGE3N2M1YTI1NDQzOGY5NTkyNTI3MGYyMzJm****'
tagging = Tagging()
tagging.tag_set.add('key1', 'value1')
tagging.tag_set.add('key2', 'value2')
# 启用Bucket版本控制
config = BucketVersioningConfig()
config.status = 'Enabled'
bucket.put_bucket_versioning(config)
params = dict()
params['versionId'] = '<yourVersionId>'
bucket.put_object_tagging(object_name, tagging, params=params)
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object、及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 简单拷贝时设置对象标签

  以下代码用于简单拷贝1 GB以下的Object时设置对象标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_OBJECT_TAGGING, OSS_OBJECT_TAGGING_COPY_DIRECTIVE
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写源Object完整路径，例如srcexampledir/exampleobject.txt。
src_object_name = 'srcexampledir/exampleobject.txt'
# 填写目标Object完整路径，例如destexampledir1/exampleobject.txt。
dest_object_name1 = 'destexampledir1/exampleobject.txt。'
# 填写目标Object完整路径，例如destexampledir2/exampleobject.txt。
dest_object_name2 = 'destexampledir2/exampleobject.txt。'
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&” + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中指定OSS_OBJECT_TAGGING_COPY_DIRECTIVE参数为COPY或者默认不指定，则dest_object_name1将拥有与源文件相同的标签信息。
headers=dict()
headers[OSS_OBJECT_TAGGING_COPY_DIRECTIVE] = 'COPY'
bucket.copy_object(bucket.bucket_name, src_object_name, dest_object_name1, headers=headers)
# 在HTTP header中指定OSS_OBJECT_TAGGING_COPY_DIRECTIVE参数为REPLACE，则dest_object_name2的标签信息将会设置为headers[OSS_OBJECT_TAGGING]指定的标签信息。
headers[OSS_OBJECT_TAGGING_COPY_DIRECTIVE] = 'REPLACE'
headers[OSS_OBJECT_TAGGING] = tagging
bucket.copy_object(bucket.bucket_name, src_object_name, dest_object_name2, headers=headers)
# 查看src_object_name的标签信息。
result = bucket.get_object_tagging(src_object_name)
for key in result.tag_set.tagging_rule:
    print('src tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
# 查看dest_object_name1的标签信息。dest_object_name1的标签信息与src_object_name的标签信息相同。
result = bucket.get_object_tagging(dest_object_name1)
for key in result.tag_set.tagging_rule:
    print('dest1 object tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
# 查看dest_object_name2的标签信息。dest_object_name2的标签信息为headers[OSS_OBJECT_TAGGING]指定的标签信息。
result = bucket.get_object_tagging(dest_object_name2)
for key in result.tag_set.tagging_rule:
    print('dest2 object tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

- 分片拷贝时设置对象标签

  以下代码用于分片拷贝1 GB以上的Object时设置对象标签。

```
# -*- coding: utf-8 -*-
import os
import oss2
from oss2 import determine_part_size
from oss2.models import PartInfo
```

```
from oss2.models import PartInfo
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写源Object完整路径，例如srcexampledir/exampleobject.txt。
src_object_name = 'srcexampledir/exampleobject.txt'
# 填写目标Object完整路径，例如destexampledir/exampleobject.txt。
dest_object_name = 'destexampledir/exampleobject.txt'
# 获取源文件的文件大小。
head_info = bucket.head_object(src_object_name)
total_size = head_info.content_length
print('src object size:', total_size)
# determine_part_size方法用于确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 * 1024)
print('part_size:', part_size)
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&" + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 初始化分片。
# 调用init_multipart_upload接口时指定headers，将会给目标文件添加标签。
upload_id = bucket.init_multipart_upload(dest_object_name, headers=headers).upload_id
parts = []
# 逐个上传分片。
part_number = 1
offset = 0
while offset < total_size:
  num_to_upload = min(part_size, total_size - offset)
  end = offset + num_to_upload - 1;
  result = bucket.upload_part_copy(bucket.bucket_name, src_object_name, (offset, end), dest_object_name, upload_id, part_number)
  #保存part信息
  parts.append(PartInfo(part_number, result.etag))
  offset += num_to_upload
  part_number += 1
# 完成分片上传。
result = bucket.complete_multipart_upload(dest_object_name, upload_id, parts)
# 获取文件元信息。
head_info = bucket.head_object(dest_object_name)
# 查看目标文件大小。
dest_object_size = head_info.content_length
print('dest object size:', dest_object_size)
# 对比源文件大小。
assert dest_object_size == total_size
# 查看源文件的标签信息。
```

```
result = bucket.get_object_tagging(src_object_name)
for key in result.tag_set.tagging_rule:
    print('src tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
# 查看目标文件的标签信息。
result = bucket.get_object_tagging(dest_object_name)
for key in result.tag_set.tagging_rule:
    print('dest tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

为软链接文件设置标签

以下代码用于为软链接文件设置标签。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.headers import OSS_OBJECT_TAGGING
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 填写目标Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
object_name = 'exampledir/exampleobject.txt'
# 填写软链接完整路径，例如shortcut/myobject.txt。
symlink_name = 'shortcut/myobject.txt'
# 设置tagging字符串。
tagging = "k1=v1&k2=v2&k3=v3"
# 如果标签中包含了任意字符，则需要对标签的Key和Value做URL编码。
k4 = "k4+-="
v4 = "+-=._:/"
tagging += "&" + oss2.urlquote(k4) + "=" + oss2.urlquote(v4)
# 在HTTP header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = tagging
# 添加软链接。
# 调用put_symlink接口时指定headers，将会给软链接文件添加标签。
result = bucket.put_symlink(object_name, symlink_name, headers=headers)
print('http response status: ', result.status)
# 查看软链接文件的标签信息。
result = bucket.get_object_tagging(symlink_name)
for key in result.tag_set.tagging_rule:
    print('tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

5.10.2. 获取对象标签

本文介绍如何获取对象标签（Object Tagging）。

说明

- 对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见开发指南中的对象标签。
- 关于获取对象标签的更多信息，请参见GetObjectTagging。

以下代码用于获取目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件的对象标签信息。

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import Tagging, TaggingRule
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取文件的标签。
result = bucket.get_object_tagging('<yourObjectName>')
# 查看标签信息。
for key in result.tag_set.tagging_rule:
    print('tagging key: {}, value: {}'.format(key, result.tag_set.tagging_rule[key]))
```

5.10.3. 删除对象标签

本文介绍如何删除对象标签（Object Tagging）。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于删除对象标签：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import Tagging, TaggingRule
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 删除文件的标签。
result = bucket.delete_object_tagging('<yourObjectName>')
print('http response status: ', result.status)
```

删除对象标签详情请参考DeleteObjectTagging。

5.10.4. 对象标签和生命周期管理

生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为生命周期规则生效的条件。

说明 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于在生命周期规则中添加标签匹配规则。

```
# -*- coding: utf-8 -*-
import oss2
import datetime
from oss2.models import (LifecycleExpiration, LifecycleRule,
                 BucketLifecycle,AbortMultipartUpload,
                 TaggingRule, Tagging, StorageTransition)
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 设置Object过期规则，距最后修改时间3天后过期。
# 设置过期规则名称和Object的匹配前缀。
rule1 = LifecycleRule('rule1', 'tests/',
              # 启用过期规则。
              status=LifecycleRule.ENABLED,
              # 设置过期规则为距最后修改时间3天后过期。
              expiration=LifecycleExpiration(days=3))
# 设置Object过期规则，指定日期之前创建的文件过期。
# 设置过期规则名称和Object的匹配前缀。
rule2 = LifecycleRule('rule2', 'logging-',
              # 不启用过期规则。
              status=LifecycleRule.DISABLED,
              # 设置过期规则为指定日期之前创建的文件过期。
              expiration=LifecycleExpiration(created_before_date=datetime.date(2018, 12, 12)))
# 设置分片过期规则，分片3天后过期。
rule3 = LifecycleRule('rule3', 'tests1/',
              status=LifecycleRule.ENABLED,
              abort_multipart_upload=AbortMultipartUpload(days=3))
# 设置分片过期规则，指定日期之前的分片过期。
rule4 = LifecycleRule('rule4', 'logging1-',
              status=LifecycleRule.DISABLED,
              abort_multipart_upload = AbortMultipartUpload(created_before_date=datetime.date(2018, 12, 1
2)))
# 设置Object的匹配标签。
tagging_rule = TaggingRule()
tagging_rule.add('key1', 'value1')
tagging_rule.add('key2', 'value2')
tagging = Tagging(tagging_rule)
# 设置转储规则，最后修改时间超过365天后转为归档存储（ARCHIVE）。
# rule5中指定了Object的匹配标签，只有同时拥有key1=value1和key2=value2两个标签的Object才会匹配此规则。
rule5 = LifecycleRule('rule5', 'logging2-',
              status=LifecycleRule.ENABLED,
              storage_transitions=[StorageTransition(days=365, storage_class=oss2.BUCKET_STORAGE_CLASS
_ARCHIVE)],
              tagging = tagging)
lifecycle = BucketLifecycle([rule1, rule2, rule3, rule4, rule5])
bucket.put_bucket_lifecycle(lifecycle)
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 查看生命周期规则。
lifecycle = bucket.get_bucket_lifecycle()
for rule in lifecycle.rules:
  # 查看分片过期规则。
  if rule.abort_multipart_upload is not None:
    print('id={0}, prefix={1}, tagging={2}, status={3}, days={4}, created_before_date={5}'
        .format(rule.id, rule.prefix, rule.tagging, rule.status,
          rule.abort_multipart_upload.days,
          rule.abort_multipart_upload.created_before_date))
  # 查看Object过期规则。
  if rule.expiration is not None:
    print('id={0}, prefix={1}, tagging={2}, status={3}, days={4}, created_before_date={5}'
        .format(rule.id, rule.prefix, rule.tagging, rule.status,
          rule.expiration.days,
          rule.expiration.created_before_date))
  # 查看转储规则。
  if len(rule.storage_transitions) > 0:
    storage_trans_info = ''
    for storage_rule in rule.storage_transitions:
      storage_trans_info += 'days={0}, created_before_date={1}, storage_class={2} **** '.format(
        storage_rule.days, storage_rule.created_before_date, storage_rule.storage_class)
    print('id={0}, prefix={1}, tagging={2}, status={3},, StorageTransition={4}'
        .format(rule.id, rule.prefix, rule.tagging, rule.status, storage_trans_info))
```

5.11. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

> **说明** 有关单链接限速的使用场景及注意事项的更多信息，请参考开发指南的单链接限速文档。

普通上传下载限速

以下代码用于普通上传、下载文件时设置单链接限速：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import OSS_TRAFFIC_LIMIT
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourRequestBucketName>')
object_name = '<yourobjectName>'
local_file_name = '<yourLocalFileName>'
down_file_name = '<yourDownloadFileName>'
# 限速100KB/s，即819200bit/s, 在headers中设置。
limit_speed = (100 * 1024 * 8)
headers = dict()
headers[OSS_TRAFFIC_LIMIT] = str(limit_speed);
# 限速上传文件。
result = bucket.put_object_from_file(object_name, local_file_name, headers=headers)
print('http response status:', result.status)
# 限速下载文件到本地。
result = bucket.get_object_to_file(object_name, down_file_name, headers=headers)
print('http response status:', result.status)
```

使用签名URL方式上传下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import OSS_TRAFFIC_LIMIT
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录https://ram.console.aliyun.com创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourRequestBucketName>')
object_name = '<yourobjectName>'
local_file_name = '<yourLocalFileName>'
down_file_name = '<yourDownloadFileName>'
# 限速100KB/s，即819200bit/s，在params中设置。
limit_speed = (100 * 1024 * 8)
params = dict()
params[OSS_TRAFFIC_LIMIT] = str(limit_speed);
# 创建限速上传文件的签名url, 有效期60s。
url = bucket.sign_url('PUT', object_name, 60, params=params)
print('put object url:', url)
# 限速上传。
result = bucket.put_object_with_url_from_file(url, local_file_name)
print('http response status:', result.status)
# 创建限速下载文件的签名url, 有效期60s。
url = bucket.sign_url('GET', object_name, 60, params=params)
print('get object url:', url)
# 限速下载。
result = bucket.get_object_with_url_to_file(url, down_file_name)
print('http response status:', result.status)
```

5.12. 数据加密

5.12.1. 客户端加密

客户端加密是指将数据发送到OSS之前在用户本地进行加密。

免责声明

- 使用客户端加密功能时，您需要对主密钥的完整性和正确性负责。因您维护不当导致主密钥用错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。
- 在对加密数据进行复制或者迁移时，您需要对加密元信息的完整性和正确性负责。因您维护不当导致加密元信息出错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。

背景信息

使用客户端加密时，会为每个Object生成一个随机数据加密密钥，用该随机数据加密密钥明文对Object的数据进行对称加密。主密钥用于生成随机的数据加密密钥，加密后的内容会当作Object的meta信息保存在服务端。解密时先用主密钥将加密后的随机密钥解密出来，再用解密出来的随机数据加密密钥明文解密Object的数据。主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。

加密方式

对于主密钥的使用，目前支持如下两种方式：

- 使用KMS托管用户主密钥

  当使用KMS托管用户主密钥用于客户端数据加密时，需要将KMS用户主密钥ID（即CMK ID）传递给SDK。

- 使用用户自主管理的主密钥（RSA）

  主密钥信息由用户提供，需要用户将主密钥的公钥、私钥信息当做参数传递给SDK。

使用以上两种加密方式能够有效地避免数据泄漏，保护客户端数据安全。即使数据泄漏，其他人也无法解密得到原始数据。

客户端加密详细信息请参见开发指南中的客户端加密。完整的示例代码请参见GitHub。

V2版本客户端加密（推荐）

> **注意**
>
> - 客户端加密支持分片上传超过5 GB的文件。在使用分片方式上传文件时，需要指定上传文件的总大小和分片大小， 除了最后一个分片外，每个分片的大小要一致，且分片大小目前必须是16的整数倍。
> - 调用客户端加密上传文件后，加密元数据会被保护，无法通过CopyObject等接口修改加密相关元数据信息。

- 加密元信息

参数	描述	是否必需
x-oss-meta-client-side-encryption-key	加密后的密钥。 经过RSA或KMS加密后再经过base64编码的字符串。	是

参数	描述	是否必需
x-oss-meta-client-side-encryption-start	随机产生的加密数据的初始向量 。经过RSA或KMS加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-cek-alg	数据的加密算法。	是
x-oss-meta-client-side-encryption-wrap-alg	数据密钥的加密算法。	是
x-oss-meta-client-side-encryption-matdesc	内容加密密钥（CEK）描述，JSON格式。	否
x-oss-meta-client-side-encryption-unencrypted-content-length	加密前的数据长度。如未指定content-length，则不生成该参数。	否
x-oss-meta-client-side-encryption-unencrypted-content-md5	加密前的数据的MD5。如未指定MD5，则不生成该参数。	否
x-oss-meta-client-side-encryption-data-size	分片上传文件的总大小。	否（分片上传时必须指定）
x-oss-meta-client-side-encryption-part-size	分片上传中每个part的大小。	否（分片上传时必须指定）

- 创建加密Bucket

  同普通上传、下载等操作一样，在使用客户端加密上传、下载文件之前需要先初始化Bucket实例。通过Bucket实例的上传、下载等接口进行文件的上传、下载操作。客户端加密通过CryptoBucket这个类继承了普通Bucket的接口，需要使用到客户端加密时，只需要传入相应的参数，同初始化Bucket实例一样，初始化一个类似的CryptoBucket实例即可。

- 初始化非对称加密主密钥方式（RSA）的Bucket

> **注意** 使用RSA加密方式时，需要您自己管理加密的密钥对，一旦丢失密钥或者密钥数据出现损坏，可能会导致数据无法解密，推荐使用KMS托管方式进行加密。如果由于业务场景一定要使用RSA加密方式，建议您做好密钥数据的备份。

初始化非对称加密主密钥方式（RSA）的Bucket示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
key_pair = {'private_key': '<yourPrivateKey>', 'public_key': '<yourPublicKey>'}
bucket = oss2.CryptoBucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name,
                 crypto_provider=RsaProvider(key_pair))
```

- 初始化KMS主密钥加密方式的Bucket

初始化KMS主密钥加密方式的Bucket的示例代码如下：

```
import os
import oss2
from oss2.crypto import AliKMSProvider
kms_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccessKeySecret>', '<yourRegion>', '<yourCMKID>')
bucket = oss2.CryptoBucket(oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>'), '<yourEndpoint>', '<yourBucket>', crypto_provider = kms_provider)
```

- 使用和管理多个密钥

  对于同一个bucket，您在上传或者下载不同的数据时，可能会使用不同的密钥进行加密。您可以给不同的密钥配置不同的描述信息，并将这些密钥和描述信息添加到bucket的加密信息中，待您再解密数据时，SDK内部会根据加密数据的描述信息自动匹配密钥，从而实现数据无缝解密。示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
# 创建一个RSA密钥对。
key_pair_1 = {'private_key': '<yourPrivateKey_1>', 'public_key': '<yourPublicKey_1>'}
mat_desc_1 = {'key1': 'value1'}
# 创建另一个RSA密钥对。
key_pair_2 = {'private_key': '<yourPrivateKey_2>', 'public_key': '<yourPublicKey_2>'}
mat_desc_2 = {'key2': 'value2'}
# 将key_pair_1的描述信息添加到provider。
provider = RsaProvider(key_pair={'private_key': private_key_str_2, 'public_key': public_key_str_2}, ma
t_desc=mat_desc_2)
encryption_materials = oss2.EncryptionMaterials(mat_desc_1, key_pair=key_pair_1)
provider.add_encryption_materials(encryption_materials)
# 使用provider初始化crypto_bucket，这样可以使用crypto_bucket下载使用描述信息为mat_desc_1的主密
钥加密的对象数据。
crypto_bucket = oss2.CryptoBucket(oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>'), '<yourE
ndpoint>', '<yourBucketName>', crypto_provider=provider)
```

- 普通上传和下载文件

  使用主密钥KMS普通上传和下载文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
kms_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccessKeySecret>', '<yourRegion>', '<yourCMKID>')
bucket = oss2.CryptoBucket(oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>'), '<yourEndpoint>', '<yourBucket>', crypto_provider = kms_provider)
key = 'motto.txt'
content = b'a' * 1024 * 1024
filename = 'download.txt'
# 上传文件。
bucket.put_object(key, content, headers={'content-length': str(1024 * 1024)})
# 下载OSS文件到本地内存。
result = bucket.get_object(key)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
content_got = b''
for chunk in result:
  content_got += chunk
assert content_got == content
# 下载OSS文件到本地文件。
result = bucket.get_object_to_file(key, filename)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
with open(filename, 'rb') as fileobj:
  assert fileobj.read() == content
```

- 分片上传

> **说明**
>
> - 分片上传文件时，一旦上传中断（进程退出）后，加密分片上传上下文可能会丢失。上传中断后如果要继续上传该文件，则必须再次上传整个文件。
> - 推荐您直接使用OSS封装好的断点续传上传接口上传大文件，该接口已将加密分片上传上下文保存在用户本地了，即使是上传出现中断，该上下文信息也不会丢失。

使用主密钥KMS分片上传文件示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
kms_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccessKeySecret>', '<yourRegion>', '<yourCMKID>')
bucket = oss2.CryptoBucket(oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>'), '<yourEndpoint>', '<yourBucket>', crypto_provider = kms_provider)
"""
分片上传
"""
# 初始化上传分片。
part_a = b'a' * 1024 * 100
part_b = b'b' * 1024 * 100
part_c = b'c' * 1024 * 100
multi_content = [part_a, part_b, part_c]
parts = []
data_size = 100 * 1024 * 3
part_size = 100 * 1024
multi_key = "test_crypto_multipart"
# 初始化加密分片上传上下文。
context = models.MultipartUploadCryptoContext(data_size, part_size)
res = bucket.init_multipart_upload(multi_key, upload_context=context)
upload_id = res.upload_id
# 分片上传示例中使用顺序上传，实际使用中为了加快上传速度也可以支持多个线程并发上传。
for i in range(3):
  # context的值不允许修改，否则将导致数据上传失败。
  result = bucket.upload_part(multi_key, upload_id, i + 1, multi_content[i], upload_context=context)
  parts.append(oss2.models.PartInfo(i + 1, result.etag, size=part_size, part_crc=result.crc))
# 完成分片上传。
result = bucket.complete_multipart_upload(multi_key, upload_id, parts)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
result = bucket.get_object(multi_key)
content_got = b''
for chunk in result:
  content_got += chunk
assert content_got[0:102400] == part_a
assert content_got[102400:204800] == part_b
assert content_got[204800:307200] == part_c
```

- 断点续传上传

  使用主密钥RSA断点续传上传文件示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
key = 'motto.txt'
content = b'a' * 1024 * 1024 * 100
file_name_put = 'upload.txt'
# 将content的内容写入文件。
with open(file_name_put, 'wb') as fileobj:
  fileobj.write(content)
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# 创建存储空间，使用用户自主管理（RSA）方式加密，此方式只支持文件整体上传下载操作。
# 如果您使用2.9.0及以上版本的SDK，不建议使用LocalRsaProvider初始化bucket。
# bucket = oss2.CryptoBucket(auth,'<yourEndpoint>', '<yourBucketName>', crypto_provider=LocalRsaProvider())
key_pair = {'private_key': '<yourPrivateKey>', 'public_key': '<yourPublicKey>'}
# 初始化bucket将upload_contexts_flag设置为True，调用upload_part接口不用传入加密分片上传上下文参数。
bucket = oss2.CryptoBucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name,
                 crypto_provider=RsaProvider(key_pair))
# 此示例为了演示方便，将multipart_threshold的值设置为100*1024，实际使用过程中可以根据使用场景设置，默认值为10 MB。
# multipart_threshold表示文件超过这个阈值就是用分片上传方式上传问题，如果文件大小小于这个值，建议使用简单的put_object接口上传文件。
# part_size为使用分片上传时分片的大小，默认值为10 MB。
# num_threads为并发上传线程的个数，默认值为1。
oss2.resumable_upload(bucket, key, file_name_put, multipart_threshold=10 * 1024 * 1024, part_size=1024 * 1024, num_threads=3)
```

- 断点续传下载

  使用主密钥KMS断点续传下载文件示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import RsaProvider
key = 'motto.txt'
content = b'a' * 1024 * 1024 * 100
file_name_get = 'download.txt'
kms_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccessKeySecret>', '<yourRegion>', '<yourCMKID>')
bucket = oss2.CryptoBucket(oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>'), '<yourEndpoint>', '<yourBucket>', crypto_provider = kms_provider)
# 断点续传下载。
oss2.resumable_download(bucket, key, file_name_get, multiget_threshold=10 * 1024 * 1024, part_size=1024 * 1024, num_threads=3)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
with open(file_name_get, 'rb') as fileobj:
  assert fileobj.read() == content
```

V1版本客户端加密（不推荐）

说明

- V1版本的客户端加密只支持通过PutObject接口上传5 GB以下的文件，不支持分片上传、断点续传上传和断点续传下载接口。
- 通过客户端加密接口上传文件后，不允许通过CopyObject等接口修改对象的加密元信息。如果修改了这些元信息，可能会导致数据无法解密。
- V1版本的客户端加密仅在Python SDK上支持，其它语言的SDK无法解密通过V1版本客户端加密上传的数据。
- 自2.11.0版本后开始支持V2版本的客户端加密，V2版本支持的功能更加完善，条件允许的情况下建议升级到新版本。

- 加密元信息

参数	描述	是否必需
x-oss-meta-oss-crypto-key	加密后的密钥。 经过RSA加密后再经过base64编码的字符串。	是
x-oss-meta-oss-crypto-start	随机产生的加密数据的初始值 。经过RSA加密后再经过base64编码的字符串。	是
x-oss-meta-oss-cek-alg	数据的加密算法。	是
x-oss-meta-oss-wrap-alg	数据密钥的加密算法。	是
x-oss-meta-oss-matdesc	内容加密密钥（CEK）描述，JSON格式。暂未生效。	否
x-oss-meta-unencrypted-content-length	加密前的数据长度。如未指定content-length则不生成该参数。	否
x-oss-meta-unencrypted-content-md5	加密前的数据的MD5。如未指定MD5则不生成该参数。	否

- 使用RSA方式上传和下载文件

  使用RSA方式上传和下载文件示例代码如下：

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import LocalRsaProvider
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# 创建存储空间，使用用户自主管理（RSA）方式加密，此方式只支持文件整体上传下载操作。
bucket = oss2.CryptoBucket(auth,'<yourEndpoint>', '<yourBucketName>', crypto_provider=LocalRsaProvider())
key = 'motto.txt'
content = b'a' * 1024 * 1024
filename = 'download.txt'
# 上传文件。
bucket.put_object(key, content, headers={'content-length': str(1024 * 1024)})
# 下载OSS文件到本地内存。
result = bucket.get_object(key)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
content_got = b''
for chunk in result:
    content_got += chunk
assert content_got == content
# 下载OSS文件到本地文件。
result = bucket.get_object_to_file(key, filename)
# 验证获取到的文件内容跟上传时的文件内容是否一致。
with open(filename, 'rb') as fileobj:
    assert fileobj.read() == content
```

5.12.2. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

  上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

  > **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

注意

- 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
- 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
- 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
# -*- coding: utf-8 -*-
import oss2
from oss2.models import ServerSideEncryptionRule
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 创建Bucket加密配置，以AES256加密为例。
rule = ServerSideEncryptionRule()
rule.sse_algorithm = oss2.SERVER_SIDE_ENCRYPTION_AES256
#设置KMS密钥ID，加密方式为KMS可设置此项。如需使用指定的密钥加密，需输入指定的CMK ID；若使用OSS托管的CMK进行加密，此项为空。使用AES256进行加密时，此项必须为空。
rule.kms_master_keyid = ""
# 设置Bucket加密。
result = bucket.put_bucket_encryption(rule)
# 查看HTTP返回码。
print('http response code:', result.status)
```

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 获取Bucket加密配置。
result = bucket.get_bucket_encryption()
# 打印获取到的加密配置。
print('sse_algorithm:', result.sse_algorithm)
print('kms_master_keyid:', result.kms_master_keyid) #如果Bucket加密方式为AES256，那么kms_master_keyid为None。
```

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
# 删除Bucket加密配置。
result = bucket.delete_bucket_encryption()
# 查看HTTP返回码。
print('http status:', result.status)
```

5.13. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

> **说明** 关于搭建STS服务的具体操作，请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。临时访问凭证有效时间单位为秒，最小值为900，最大值以当前角色设定的最大会话时间为准。更多信息，请参见设置角色最大会话时间。

执行pip install aliyun-python-sdk-sts命令，安装官方的Python STS客户端。关于STS应用的完整示例代码，请参见GitHub。

请确保使用2.0.6及以上版本SDK。以下代码用于使用STS临时授权下载文件。

```
# -*- coding: utf-8 -*-
from aliyunsdkcore import client
from aliyunsdksts.request.v20150401 import AssumeRoleRequest
import json
import oss2
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-han
gzhou.aliyuncs.com。
endpoint = 'yourEndpoint'
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
access_key_id = 'yourAccessKeyId'
access_key_secret = 'yourAccessKeySecret'
# 填写Bucket名称。
bucket_name = 'examplebucket'
# 填写Object完整路径和字符串。Object完整路径中不能包含Bucket名称。
object_name = 'exampleobject.txt'
# 您可以登录RAM控制台，在“RAM角色管理”界面，搜索创建的RAM角色后，单击RAM角色名称，在RAM角色详情界
面查看和复制角色的ARN信息。
# 填写角色的ARN信息。格式为acs:ram::$accountID:role/$roleName。
# $accountID为阿里云账号ID。您可以通过登录阿里云控制台，将鼠标悬停在右上角头像的位置，直接查看和复制账号
ID，或者单击基本资料查看账号ID。
# $roleName为RAM角色名称。您可以通过登录RAM控制台，单击左侧导航栏的RAM角色管理，在RAM角色名称列表
下进行查看。
role_arn = 'acs:ram::17464958********:role/ossststest'
# 创建权限策略。
# 只允许对名称为examplebucket的Bucket下的所有资源执行GetObject操作。
policy_text = '{"Version": "1", "Statement": [{"Action": ["oss:GetObject"], "Effect": "Allow", "Resource": ["acs
:oss:*:*:examplebucket/*"]}]}'
clt = client.AcsClient(access_key_id, access_key_secret, 'cn-hangzhou')
req = AssumeRoleRequest.AssumeRoleRequest()
# 设置返回值格式为JSON。
req.set_accept_format('json')
req.set_RoleArn(role_arn)
# 用户自定义参数。此参数用来区分不同的令牌，可用于用户级别的访问审计。
req.set_RoleSessionName('session-name')
req.set_Policy(policy_text)
body = clt.do_action_with_exception(req)
# 使用RAM用户的AccessKeyId和AccessKeySecret向STS申请临时访问凭证。
token = json.loads(oss2.to_unicode(body))
# 使用临时访问凭证中的认证信息初始化StsAuth实例。
auth = oss2.StsAuth(token['Credentials']['AccessKeyId'],
            token['Credentials']['AccessKeySecret'],
            token['Credentials']['SecurityToken'])
# 使用StsAuth实例初始化存储空间。
bucket = oss2.Bucket(auth, endpoint, bucket_name)
# 下载object。
read_obj = bucket.get_object(object_name)
print(read_obj.read())
```

使用签名URL进行临时授权

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

在URL中加入签名信息，以便将该URL转给第三方实现授权访问。具体操作，请参见在URL中包含签名。

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

- 使用签名URL临时授权上传文件

  使用签名URL临时授权上传文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 生成上传文件的签名URL，有效时间为60秒。
print(bucket.sign_url('PUT', 'exampleobject.txt', 60))
```

- 使用签名URL临时授权下载文件

  使用签名URL临时授权下载文件的示例代码如下：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
auth = oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret')
# yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
# 填写Bucket名称。
bucket = oss2.Bucket(auth, 'yourEndpoint', 'examplebucket')
# 生成下载文件的签名URL，有效时间为60秒。
print(bucket.sign_url('GET', 'exampleobject.txt', 60))
```

5.14. 数据安全性

OSS提供基于MD5和CRC64的数据校验，确保上传、下载和拷贝文件（Object）过程中的数据完整性。

MD5校验

如果上传文件时设置了Content-MD5，OSS会根据接收的内容计算MD5。OSS计算的MD5值和上传提供的MD5值不一致时，则返回InvalidDigest异常，从而保证数据的完整性。返回InvalidDigest异常后，您需要重新上传文件。

以下代码用于上传文件时进行MD5校验：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = '<yourObjectName>'
content = '<yourContent>'
# 计算上传内容的md5。
content_md5 = oss2.utils.content_md5(content);
print('content_md5', content_md5)
# 上传请求中携带'Content-MD5' header。服务器将会校验上传内容的md5。
headers = dict()
headers['Content-MD5'] = content_md5
bucket.put_object(object_name, content, headers=headers)
```

说明 put_object、append_Object、post_Object、upload_part均支持MD5校验。

CRC64校验

使用CRC校验数据时，有如下注意事项：

说明

- put_object、get_object、append_object、upload_part支持CRC64校验。上传文件时默认开启CRC校验，如果客户端计算的CRC值与服务端返回的CRC值不一致， 则会抛出InconsistentError异常。
- 范围下载不支持CRC64校验。
- CRC64校验会占用一定的CPU，对上传、下载速度均会有影响。

- 下载文件时CRC64校验

  以下代码用于下载文件时进行CRC64数据完整性校验：

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = '<yourObjectName>'
# 查看是否已默认开启crc校验。
print('bucket.enable-crc:', bucket.enable_crc)
# bucket.get_object的返回值是一个类文件对象（File-Like Object），同时也是一个可迭代对象（Iterable）。
object_stream = bucket.get_object(object_name)
print(object_stream.read())
# 由于get_object接口返回的是一个stream流，需要执行read()后才能计算出返回Object数据的CRC checksum，因此需要在调用该接口后做CRC校验。
if object_stream.client_crc != object_stream.server_crc:
    print "The CRC checksum between client and server is inconsistent!"
```

- 追加上传时CRC64校验

  追加上传时，如果指定了init_crc参数，则默认开启CRC64校验。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
object_name = "<yourAppendObjectName>"
first_content = "<yourFirstContent>"
second_content = "<yourSecondContent>"
# 第一次追加上传。
# 在指定了init_crc的情况下，SDK默认会对返回结果进行crc校验。
result = bucket.append_object(object_name, 0, first_content, init_crc=0)
# 第二次追加上传。
# 指定init_crc为已上传数据的crc。
result = bucket.append_object(object_name, result.next_position, second_content, init_crc=result.crc)
```

5.15. 开启Python SDK日志

为方便追查问题，Python SDK提供了日志记录功能，该功能默认处于关闭状态。

Python SDK日志记录功能可以收集定位各类OSS操作的日志信息，并以日志文件的形式存储在本地。

> **说明** 开启Python SDK日志记录功能需要 Python SDK 2.6.x 以上版本。

日志文件的格式为：

```
<time><name><level><threadId><message>
```

日志级别：CRITICAL > ERROR > WARNING > INFO > DEBUG > NOTSET

> **说明** 指定日志级别后，本地日志文件只会记录指定级别及高于指定级别的信息。例如，指定日志级别为INFO，则日志文件会记录CRITICAL 、ERROR、WARNING及INFO级别的相关信息。

开启Python SDK日志记录功能

以下代码用于开启Python SDK日志记录功能：

```
# -*- coding: utf-8 -*-
import os
import logging
import oss2
from itertools import islice
# 初始化AccessKeyId、AccessKeySecret、Endpoint等信息。
# 请将<AccessKeyId>、<AccessKeySecret>、<Bucket>及<访问域名>分别替换成对应的具体信息。
access_key_id = '<AccessKeyId>'
access_key_secret = '<AccessKeySecret>'
bucket_name = '<Bucket>'
endpoint = '<访问域名>'
log_file_path = "log.log"
# 开启日志
oss2.set_file_logger(log_file_path, 'oss2', logging.INFO)
# 创建Bucket对象。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 遍历文件目录
for b in islice(oss2.ObjectIterator(bucket), 10):
    print(b.key)
# 获取文件元信息
object_meta = bucket.get_object_meta('object')
```

相应的日志信息如下：

```
2018-11-20 16:37:46,325 oss2.api [INFO] 26716 : Init oss bucket, endpoint: oss-cn-shenzhen.aliyuncs.com, isC
name: False, connect_timeout: None, app_name: , enabled_crc: True
2018-11-20 16:37:46,326 oss2.api [INFO] 26716 : Start to List objects, bucket: funrily-hello, prefix: , delimiter: ,
marker: , max-keys: 100
2018-11-20 16:37:46,430 oss2.api [INFO] 26716 : List objects done, req_id: 5BF3C7DA236B3A201CE645F7, stat
us_code: 200
2018-11-20 16:37:46,430 oss2.api [INFO] 26716 : Start to get object metadata, bucket: bucket-hello, key: objec
t
2018-11-20 16:37:46,437 oss2.api [ERROR] 26716 : Exception: {
    'status': 404,
    'x-oss-request-id': '5BF3C7DA236B3A201CE64679',
    'details': {
     'HostId': 'bucket-hello.oss-cn-shenzhen.aliyuncs.com',
       'Message': 'The specified key does not exist.',
       'Code': 'NoSuchKey',
       'RequestId': '5BF3C7DA236B3A201CE64679',
       'Key': 'object'
    }
}
```

由以上日志信息可知，初始化一个Bucket对象会遍历其中所有的Object，当请求的Object不存在时会发生错误，错误信息为：`NoSuchKey`。

5.16. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的完整代码请参见：GitHub。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
# -*- coding: utf-8 -*-
import os
import oss2
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
# 目标Bucket名称。
bucket_name = '<yourBucketName>'
# 目标图片名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
key = '<yourObjectName>'
# 指定处理后的图片名称。
new_pic = '<LocalFileName>'
# 指定Bucket实例，所有文件相关的方法都需要通过Bucket实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
# bucket.put_object_from_file(key, '<yourLocalFile>')
# 将图片缩放为固定宽高100 px后保存到本地。
style = 'image/resize,m_fixed,w_100,h_100'
bucket.get_object_to_file(key, new_pic, process=style)
# 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
# bucket.delete_object(key)
# 若处理后的图片不再需要，可以删除处理后的图片。
# os.remove(new_pic)
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
# -*- coding: utf-8 -*-
import os
import oss2
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
# 目标Bucket名称。
bucket_name = '<yourBucketName>'
# 目标图片名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
key = '<yourObjectName>'
# 指定处理后的图片名称。
new_pic = '<LocalFileName>'
# 指定Bucket实例，所有文件相关的方法都需要通过Bucket实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
# bucket.put_object_from_file(key, '<yourLocalFile>')
# 将图片缩放为固定宽高100 px后，再旋转90°，之后保存在本地。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'
bucket.get_object_to_file(key, new_pic, process=style)
# 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
# bucket.delete_object(key)
# 若处理后的图片不再需要，可以删除处理后的图片。
# os.remove(new_pic)
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
# -*- coding: utf-8 -*-
import os
import oss2
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
# 目标Bucket名称。
bucket_name = '<yourBucketName>'
# 目标图片名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
key = '<yourObjectName>'
# 指定处理后的图片名称。
new_pic = '<LocalFileName>'
# 指定Bucket实例，所有文件相关的方法都需要通过Bucket实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
# bucket.put_object_from_file(key, '<yourLocalFile>')
# 指定图片样式。
style = 'style/oss-pic-style-w-100'
# 将处理后的图片保存在本地。
bucket.get_object_to_file(key, new_pic, process=style)
# 图片处理完成后，若Bucket内的原图不再需要，可以删除原图。
# bucket.delete_object(key)
# 若处理后的图片不再需要，可以删除处理后的图片。
# os.remove(new_pic)
```

图片处理持久化

您可以通过ImgSaveAs接口将处理后的图片保存至指定Bucket。以下代码展示了图片处理持久化操作：

```
# -*- coding: utf-8 -*-
import os
import base64
import oss2
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
# 原图所在Bucket名称。
source_bucket_name = '<yourBucketName>'
# 指定处理后图片保存的Bucket，该Bucket需与源Bucket在同一地域。
taget_bucket_name = '<yourTargetBucketName>'
# 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
source_image_name = '<sourceObjectName>'
# 指定Bucket实例，所有文件相关的方法都需要通过Bucket实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, source_bucket_name)
# 将图片缩放为固定宽高100 px。
style = 'image/resize,m_fixed,w_100,h_100'
# 指定处理后图片名称，若携带访问路径，会将图片存储在指定路径。例如example/example-resize.jpg。
target_image_name = '<targetObjectName>'
process = "{0}|sys/saveas,o_{1},b_{2}".format(style,
  oss2.compat.to_string(base64.urlsafe_b64encode(oss2.compat.to_bytes(target_image_name))),
  oss2.compat.to_string(base64.urlsafe_b64encode(oss2.compat.to_bytes(taget_bucket_name))))
result = bucket.process_object(source_image_name, process)
print(result)
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
# -*- coding: utf-8 -*-
import oss2
# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
# 目标Bucket名称。
bucket_name = '<yourBucketName>'
# 目标图片名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
key = '<yourObjectName>'
# 指定Bucket实例，所有文件相关的方法都需要通过Bucket实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 若目标图片不在指定Bucket内，需上传图片到目标Bucket。
# bucket.put_object_from_file(key, '<yourLocalFile>')
# 将图片缩放为固定宽高100 px后，再旋转90°。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'
# 生成带签名的URL，并指定过期时间为10分钟。过期时间单位为秒。
url = bucket.sign_url('GET', key, 10 * 60, params={'x-oss-process': style})
print(url)
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

5.17. 中文和时间

本文介绍使用Python SDK时所用到的中文和时间知识。

中文

在Python代码中如果使用了中文字符，运行时会出错。因此，您需要在代码的开头部分加入字符编码的声明，例如：

```
# -*- coding: utf-8 -*-
```

- 数据类型

  Python 2.x支持以下两种数据类型：

数据类型	描述
str	字符串。对应Python 3.x中的bytes类型。
unicode	unicode流。其长度是字符数，如 `u'中文'` 的长度是2。

  Python 3.x支持以下两种数据类型：

数据类型	描述
str	字符串。对应Python 2.x中的unicode类型。
bytes	字节流。其长度是字节数，如 `b'中文'` 的长度取决于编码，如果是UTF-8编码，则为6。

- 输入、输出类型约定

  输入类型约定如下：

输入	类型	备注
OSS文件名	str	如为bytes，要求是UTF-8编码。
本地文件名	str, unicode	如为bytes，要求是UTF-8编码，例如bucket.get_object_to_file里的yourLocalFile参数。
输入数据流	bytes	例如bucket.put_object里的data参数。

  输出类型约定如下：

输出	类型	备注
解析XML得到的结果	str	例如通过bucket.list_object得到结果中的字符串。
下载内容	bytes	Python SDK默认bytes类型经过UTF-8编码，请确保Python源文件也是UTF-8编码。

- 类型转换函数

  Python SDK提供了三个用于类型转换的函数：

函数	描述
to_bytes	- Python 2.x中，把unicode转换为str。其他类型则原值返回。 - Python 3.x中，把str转换为bytes。其他类型则原值返回。
to_unicode	- Python 2.x中，把str转换为unicode。其他类型则原值返回。 - Python 3.x中，把bytes转换为str。其他类型则原值返回。
to_string	Python 2.x中相当于to_bytes。Python 3.x中相当于to_unicode。

时间

Python SDK会把从服务器获得的时间戳字符串（datetime.datetime类型的时间）都转换为Unix Time类型的时间，即自1970年1月1日UTC零点以来的秒数。例如bucket.get_object方法返回结果中的last_modified就是一个int类型的Unix Time。

您可以通过datetime.datetime.fromtimestamp()方法进行时间转换，得到时间戳字符串。

5.18. 异常处理

OSS Python SDK异常（OssError）分为三类：ClientError、RequestError和ServerError，这些异常定义在oss2.exceptions子模块中。

异常的变量、类型及描述如下表所示：

变量	类型	描述
status	int	• 如果为ServerError异常，则status为HTTP状态码。 • 如果为ClientError和RequestError异常，则status为固定值。
request_id	str	• 如果为ServerError异常，则OSS服务器返回请求ID。 • 如果为ClientError和RequestError异常，则request_id返回的是空字符串。
code和message	str	对应OSS的错误响应格式里的Code和Message两个XML Tag中的文本。

异常处理示例

以下代码展示了下载一个不存在文件时的异常处理，并打印出错误信息的HTTP状态码和请求ID。

```
# -*- coding: utf-8 -*-
import oss2
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以杭州为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com', '<yourBucketName>')
try:
  stream = bucket.get_object('random-key.txt')
except oss2.exceptions.NoSuchKey as e:
  print('status={0}, request_id={1}'.format(e.status, e.request_id))
```

ClientError

ClientError是由于客户端输入有误引起的。例如，使用bucket.batch_delete_objects方法时，如果收到空的文件列表，会抛出该异常。ClientError的status值是oss2.exceptions.OSS_CLIENT_ERROR_STATUS。

RequestError

当HTTP库抛出异常时，Python SDK会将其转换为RequestError。RequestError的status值是oss2.exceptions.OSS_REQUEST_ERROR_STATUS。

ServerError

当OSS服务器返回HTTP错误码时，Python SDK会将其转换为ServerError。ServerError根据HTTP状态码和OSS错误码派生出多个子类。其中NotFound子类对应所有404异常，Conflict子类对应所有409异常。

下表列出了常见的错误码：

异常类	HTTP状态码	OSS错误码	描述
NotModified	304	空	没有修改
AccessDenied	403	AccessDenied	拒绝访问
NoSuchBucket	404	NoSuchBucket	存储空间不存在
NoSuchKey	404	NoSuchKey	文件不存在
NoSuchUpload	404	NoSuchUpload	分片上传不存在
NoSuchWebsite	404	NoSuchWebsiteConfiguration	静态网站托管未配置
NoSuchLifecycle	404	NoSuchLifecycle	生命周期规则未配置
NoSuchCors	404	NoSuchCORSConfiguration	跨域资源共享未配置
BucketNotEmpty	409	BucketNotEmpty	存储空间非空
PositionNotEqualToLength	409	PositionNotEqualToLength	设置的追加位置和文件长度不等
ObjectNotAppendable	409	ObjectNotAppendable	不是可追加文件

5.19. 常见问题

本文介绍使用OSS Python SDK的常见问题及解决方法。

OSS Python SDK分片上传失败

解决方法如下：

- 先确认是直接上传到OSS，还是通过其他proxy传输到OSS（类似CDN）。如果经过CDN再上传到OSS，需要在OSS中配置跨域的HTTP Header，例如 `Access-Control-Allow-Origin` 、 `Access-Control-Allow-Mehtods` 、 `Access-Control-Allow-Headers` 等，并暴露ETag。更多信息，请参见PutBucketcors。
- 如果是网络超时导致上传失败，建议使用断点续传来替代普通上传。断点续传支持并行上传以及自定义分片大小。如果捕获到异常，需要详细查看并分析捕获到的SDK异常信息。
- 清理上传失败的碎片文件再重新上传。

如果以上方案仍旧没有解决您的问题，需要将以下信息提供给阿里云：

- SDK返回异常中的requestID。
- 客户端部署tcpdump，然后重新运行代码上传，并保存抓包。

```
tcpdump -i <网卡出口名称> -s0 host <访问oss的域名> -w faild.pcap
```

同台机器使用ossutil上传下载很快，而使用Python SDK上传下载很慢

- 错误原因

  ossutil工具基于Go SDK进行开发，并发上传的性能较好。如果使用Python SDK上传、下载时性能远不如ossutil，通常是因为没有正确安装crcmod 。

- 解决方法

  关于安装crcmod的更多信息，请参见安装Python SDK。如仍未能解决您的问题，请提交工单。

在Centos机器上调用SDK中的分片上传函数正常，但在Ubuntu机器上总是报403错误。

客户端部署tcpdump抓包，然后通过tcp报文排查是否由于Header信息不正确导致计算签名与服务端不匹配。

```
POST /ttsservice%2Fpasswd?uploadId=D468E486D1D94D90A1AB8885A4E32AE0 HTTP/1.1
Host: rokid.oss-cn-hangzhou.aliyuncs.com
Accept-Encoding: identity
Accept: text/html
Content-Length: 137
date: Sat, 29 Dec 2018 07:32:34 GMT
authorization: OSS LTAIknFr:r2KPR0y4E0G5tnU/MYdcvXHP****
Content-Type: application/x-www-form-urlencoded
User-Agent: aliyun-sdk-python/2.6.0(Linux/4.4.0-31-generic/x86_64;3.4.3)
<CompleteMultipartUpload><Part><PartNumber>1</PartNumber><ETag>"3195544E19D99658706D5****"</
ETag></Part></CompleteMultipartUpload>HTTP/1.1 403 Forbidden
Server: AliyunOSS
Date: Sat, 29 Dec 2018 07:33:43 GMT
Content-Type: application/xml
Content-Length: 1122
Connection: keep-alive
x-oss-request-id: 5C2723573183****
x-oss-server-time: 0
<?xml version="1.0" encoding="UTF-8"?>
<Error>
 <Code>SignatureDoesNotMatch</Code>
 <Message>The request signature we calculated does not match the signature you provided. Check your key
and signing method.</Message>
 <RequestId> 5C2723573183A12D </RequestId>
 <HostId>rokid.oss-cn-hangzhou.aliyuncs.com</HostId>
 <OSSAccessKeyId> LTAXXX </OSSAccessKeyId>
 <SignatureProvided>r2KPR0y4E0G5tnU/MYdc****</SignatureProvided>
 <StringToSign>POST
application/x-www-form-urlencoded
Sat, 29 Dec 2018 07:32:34 GMT
/rokid/ttsservice/passwd?uploadId=D468E486D1D94D90A1AB8885A4E3****</StringToSign>
 <StringToSignBytes>50 4F 53 54 0A 0A 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 2D 77 77 77 2D 66 6F 72 6D 2D 75
72 6C 65 6E 63 6F 64 65 64 0A 53 61 74 2C 20 32 39 20 44 65 63 20 32 30 31 38 20 30 37 3A 33 32 3A 33 34 20 47 4D
54 0A 2F 72 6F 6B 69 64 2D 6F 70 73 2D 6D 6F 64 65 6C 2F 74 74 73 73 65 72 76 69 63 65 2F 70 61 73 73 77 64 3F 75
70 6C 6F 61 64 49 64 3D 44 34 36 38 45 34 38 36 44 31 44 39 34 44 39 30 41 31 41 42 38 38 38 35 41 34 45 33 32 41 4
5 30 </StringToSignBytes>
</Error>
```

如果服务端收到的签名（Signature）和客户端计算的签名信息不一致，说明请求的内容已被改动，建议使用HTTPS的方式上传。

以上是客户端抓取的报文信息。请将获取的请求头信息带入以下脚本，并将计算结果与SDK进行比较。

```
import base64
import hmac
import sha
mac = hmac.new("<Secretkey>","POST\n\napplication/x-www-form-urlencoded\nSat, 29 Dec 2018 07:32:34
GMT\n/rokid/ttsservice/passwd?uploadId=D468E486D1D94D90A1AB8885A4E3****", sha)
Signature = base64.b64encode(mac.digest())
print(Signature)
```

如果抓包数据和脚本计算的结果一致，则说明SDK计算正确。如果抓包数据和脚本计算的结果不一致，可能是因为SDK在Ubuntu平台编译的适配问题导致MD5值不一样。

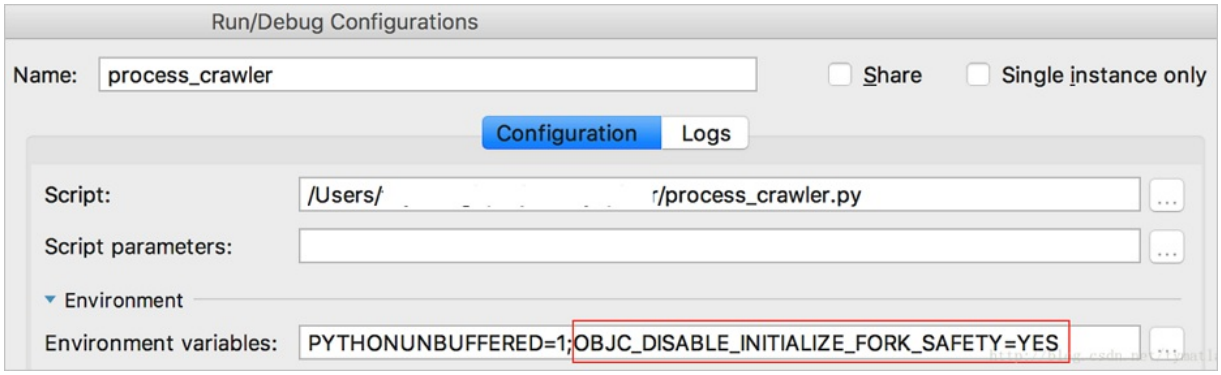
在Mac环境中用Python启动多线程并在子线程中使用OSS时，import tensorflow会报错，没有import tensorflow则不会报错。如果没有启动多线程，使用OSS时import tensorflow不会报错。

Python多线程运行时，报如下错误：

```
objc[2483]: +[__NSPlaceholderDate initialize] may have been in progress in another thread when fork() was c
alled.
objc[2483]: +[__NSPlaceholderDate initialize] may have been in progress in another thread when fork() was c
alled. We cannot safely call it or ignore it in the fork() child process. Crashing instead. Set a breakpoint on obj
c_initializeAfterForkError to debug.
```

解决方法：

添加环境变量 `OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES` ，如下图所示：



更多信息，请参见macOS 10.13系统使用多线程的错误说明。

如何添加重试策略

Python SDK自身没有重试机制。在网络状况不好的情况下，可能会出现请求失败的情况。此时，建议您参考如下代码自行添加重试策略。

```
def test_get_object():
  MAX_RETRIES = 3
  retry_count = 0
  while True:
    try:
      retry_count += 1
      # yourObjectName表示不包含Bucket名称在内的OSS文件的完整路径，例如abc/efg/example.jpg。
      # yourFileName表示下载到本地文件的完整路径，例如/users/local/example.jpg。
      bucket.get_object_to_file("yourObjectName", "yourFileName")
      break
    except Exception:
      if retry_count >= MAX_RETRIES:
        raise
```

6.Browser.js

6.1. 安装

本文档介绍OSS Browser.js SDK的安装。

准备工作

- 使用RAM用户或STS方式访问

  由于阿里云账号AccessKey拥有所有API访问权限，建议遵循阿里云安全最佳实践。如果部署在服务端可以使用RAM用户或STS来进行API访问或日常运维管控操作，如果部署在客户端请使用STS方式来进行API访问。更多信息，请参见访问控制。

- 设置跨域资源共享（CORS）

  具体操作，请参见设置跨域资源共享。

  通过浏览器直接访问OSS时，CORS配置规则要求如下：

  - **来源**：设置精准域名（例如 `www.aliyun.com` ）或带有通配符星号（*）的域名（例如 `*.aliyun.com` ）。
  - **允许Methods**：请根据实际使用场景，选择不同的Methods。例如分片上传时，设置为**PUT**；删除文件时，设置为**DELETE**。
  - **允许Headers**：设置为 `*` 。
  - **暴露Headers**：设置为 `ETag` 、 `x-oss-request-id` 和 `x-oss-version-id` 。

注意事项

OSS Browser.js SDK通过Browserify和Babel产生适用于浏览器的代码。由于浏览器环境的特殊性，无法使用以下功能：

- 流式上传：浏览器中无法设置chunked编码，建议使用分片上传替代。
- 操作本地文件：浏览器中不能直接操作本地文件系统，建议使用签名URL的方式下载文件。
- 由于OSS暂时不支持Bucket相关的跨域请求，建议在控制台执行Bucket相关操作。

下载SDK

目前官网文档中的demo都是基于SDK 6.X，版本低于6.X的可参考 5.X开发文档，如果要升级到6.X请参考升级文档。

- Github地址
- API文档
- ChangeLog

安装SDK

- 支持的浏览器

- IE(>=10)和Edge
- 主流版本的Chrome、Firefox、Safari
- 主流版本的Android、iOS、WindowsPhone系统默认浏览器

- 安装方式

  Browser.js SDK的安装方式有以下几种：

  - 浏览器引入

> **注意** 因为部分浏览器不支持promise，需要引入promise兼容库。 例如IE10和IE11需要引入promise-polyfill。

```
<!-- 引入在线资源 -->
<script src="https://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></script>
```

```
<!-- 引入本地资源 -->
<script src="./aliyun-oss-sdk-x.x.x.min.js"></script>
```

> **说明**
> - x.x.x代表版本号，具体版本信息请参考访问查看 。
> - 引入在线资源方式依赖于CDN服务器的稳定性，推荐用户使用离线方式引入，即通过本地资源或自行构建的方式。

在代码中使用OSS对象：

> **说明** 搭建STS服务的具体操作请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。

```
<script type="text/javascript">
 let client = new OSS({
  // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 从STS服务获取的安全令牌（SecurityToken）。
  stsToken: 'yourSecurityToken',
  // 填写Bucket名称。
  bucket: 'examplebucket'
 });
</script>
```

- 使用npm安装SDK开发包

```
npm install ali-oss
```

成功安装完成后，即可使用import或require进行引用。由于浏览器中原生不支持 `require` 模式，因此需要在开发环境中配合相关的打包工具，例如 `webpack` 、 `browserify` 等。

```
let OSS = require('ali-oss');
let client = new OSS({
  // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 从STS服务获取的安全令牌（SecurityToken）。
  stsToken: 'yourSecurityToken',
  // 填写Bucket名称。
  bucket: 'examplebucket'
});
```

使用方式

OSS Browser.js SDK支持同步和异步的使用方式。

- 同步方式：基于ES7规范的 `async/await` ，使得异步方式同步化。
- 异步方式：与Callback方式类似，API接口返回Promise，使用 `then()` 处理返回结果，使用 `catch()` 处理错误。

> **说明** 无论同步方式还是异步方式，均使用 `new OSS()` 创建client。

以下是使用同步或者异步方式上传文件，然后下载文件的示例说明。

- 同步方式

```
// 实例化OSS Client。
let client = new OSS(...);
async function put () {
 try {
  // object表示上传到OSS的文件名称。
  // file表示浏览器中需要上传的文件，支持HTML5 file和Blob类型。
  let r1 = await client.put('object', file);
  console.log('put success: %j', r1);
  let r2 = await client.get('object');
  console.log('get success: %j', r2);
 } catch (e) {
  console.error('error: %j', e);
 }
}
put();
```

- 异步方式

```
// 实例化OSS Client。
let client = new OSS(...);
// object表示上传到OSS的文件名称。
// file表示浏览器中需要上传的文件，支持HTML5 file和Blob类型。
client.put('object', file).then(function (r1) {
  console.log('put success: %j', r1);
  return client.get('object');
}).then(function (r2) {
  console.log('get success: %j', r2);
}).catch(function (err) {
  console.error('error: %j', err);
});
```

6.2. 配置项

本文介绍如何使用配置项。

OSS（options）中的各个配置项说明如下：

- [accessKeyId] {String}：通过阿里云控制台创建的access key。
- [accessKeySecret] {String}：通过阿里云控制台创建的access secret。
- [stsToken] {String}：使用临时授权方式，详情请参见使用STS访问。
- [bucket] {String}：通过控制台创建的bucket。
- [endpoint] {String}：OSS域名。
- [region] {String}：bucket 所在的区域，默认 oss-cn-hangzhou。
- [internal] {Boolean}：是否使用阿里云内网访问，默认false。比如通过ECS访问OSS，则设置为true，采用internal的endpoint可节约费用。
- [cname] {Boolean}：是否支持上传自定义域名，默认false。如果cname为true，endpoint传入自定义域名时，自定义域名需要先同bucket进行绑定。
- [isRequestPay] {Boolean}：bucket是否开启请求者付费模式，默认false。具体可查看请求者付费模式。
- [secure] {Boolean}：(secure: true) 则使用 HTTPS，(secure: false) 则使用 HTTP，详情请查看常见问题。
- [timeout] {String|Number}：超时时间，默认 60s。

Browser.js 示例如下：

```
// 假设浏览器环境当中已经引入OSS对象 可以是通过`script`或者`npm`方式引入
let store = new OSS({
  accessKeyId: 'your access key',
  accessKeySecret: 'your access secret',
  bucket: 'your bucket name',
  region: 'oss-cn-hangzhou'
});
```

6.3. 快速开始

本文介绍如何在Browser.js SDK中快速使用OSS服务，包括上传文件、下载文件和查看文件列表等。

前提条件

已配置跨域资源共享（CORS）规则。详情请参见安装。

背景信息

云账号AccessKey拥有所有API访问权限，建议使用STS方式给客户端授权。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。详情请参见使用STS临时访问凭证访问OSS。

浏览器的使用请参见浏览器应用。

示例工程请参见example。

使用SDK

目前浏览器中不允许执行Bucket相关的操作，例如Get Bucket (List Objects)、Get Bucket Logging、Get Bucket Referer等。但允许您执行Object相关的操作，例如Put Object 、Get Object 等。

> **注意** 浏览器是不受信任的环境，如果把AccessKeyId和AccessKeySecret直接保存在浏览器端，存在极高的风险。建议只在测试时使用明文设置，业务应用中请使用STS鉴权模式或自签名模式，详细说明请参见授权访问和快速搭建移动应用直传服务。

- 引入SDK

  在HTML文件的 `<head>` 中包含如下信息：

  ```
  <script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></script>
  ```

  有关其他引入方式，请参见安装。

  通过 `new OSS` 来创建client，创建后返回Promise，类似于callback的方式，在 `.then()` 中处理返回的结果，在 `.catch()` 中处理错误。

- 搭建STS Server并从客户端获取临时授权信息

  i. 借助STS的SDK或API应用专属的STS签名服务。为方便您快速理解STS，OSS提供了多个语言版本的demo：

     - Node.js
     - PHP
     - Java
     - Ruby

     > **说明** 以上demo仅供参考，在具体的生产环境中，请自行开发鉴权等业务所需代码。

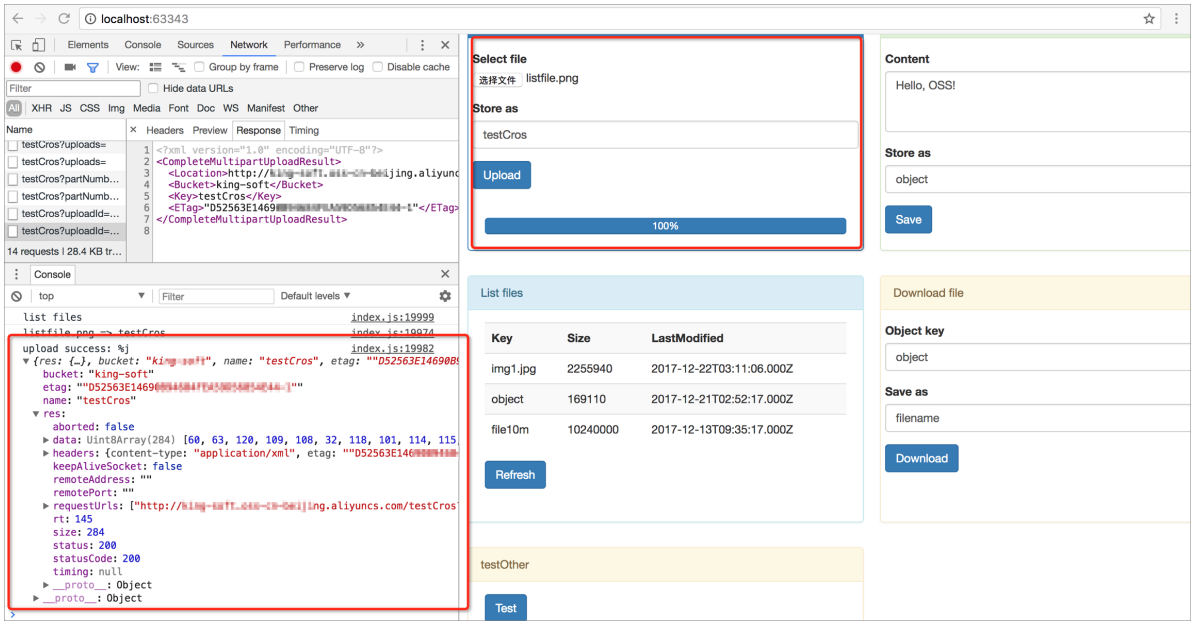
  ii. 通过浏览器向搭建的STS服务发起请求，获取STS临时授权信息。

```
<script type="text/javascript">
 // OSS.urlib是SDK内部封装的发送请求的逻辑，开发者可以使用任何发送请求的库向sts-server发送请求。
 OSS.urllib.request("http://your_sts_server/", {method: 'GET'}, (err, response) => {
   if (err) {
    return alert(err);
   }
   try {
    result = JSON.parse(response);
   } catch (e) {
     errmsg = 'parse sts response info error: ' + e.message;
     return alert(errmsg);
   }
  // console.log(result)
 })
</script>
```

- 上传文件

  使用multipartUpload接口来上传文件。有关multipartUpload详情，请参见分片上传。

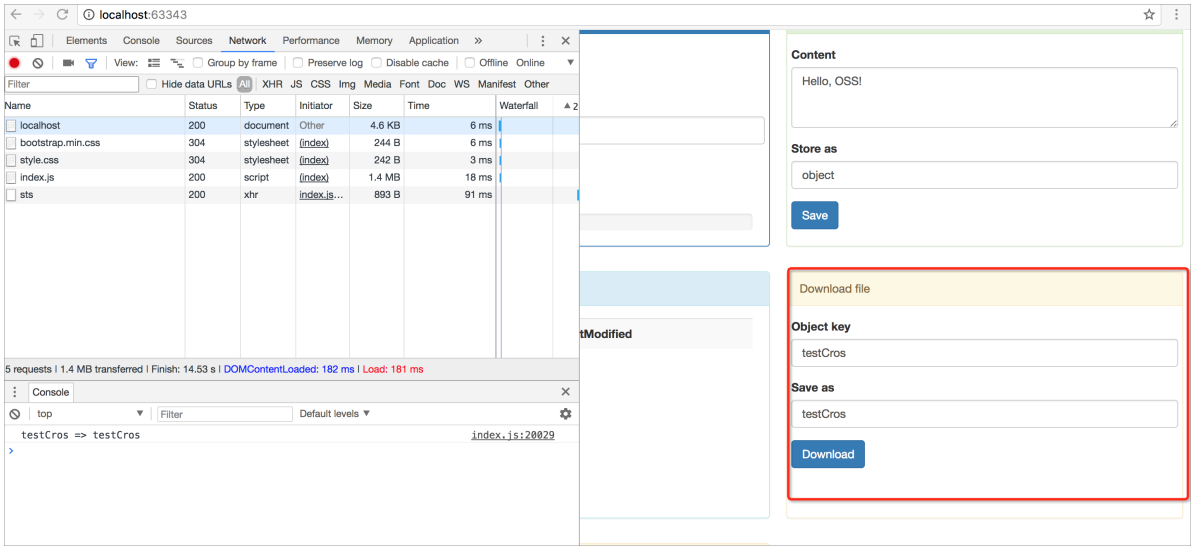
```
<body>
 <input type="file" id="file" />
 <script type="text/javascript">
  document.getElementById('file').addEventListener('change', function (e) {
   let file = e.target.files[0];
   let storeAs = 'upload-file';
   console.log(file.name + ' => ' + storeAs);
   // OSS.urlib是SDK内部封装的发送请求的逻辑，开发者可以使用任何发送请求的库向sts-server发送请求。
   OSS.urllib.request("http://your_sts_server/", {method: 'GET'}, (err, response) => {
     if (err) {
      return alert(err);
     }
     try {
      result = JSON.parse(response);
     } catch (e) {
      return alert('parse sts response info error: ' + e.message);
     }
     let client = new OSS({
      accessKeyId: result.AccessKeyId,
      accessKeySecret: result.AccessKeySecret,
      stsToken: result.SecurityToken,
      // region表示您申请OSS服务所在的地域，例如oss-cn-hangzhou。
      region: '<Your region>',
      bucket: '<Your bucket name>'
     });
     // storeAs可以自定义为文件名（例如file.txt）或目录（例如abc/test/file.txt）的形式，实现将文件上传至当
前Bucket或Bucket下的指定目录。
     // file可以自定义为File对象、Blob数据以及OSS Buffer。
     client.multipartUpload(storeAs, file).then(function (result) {
      console.log(result);
     }).catch(function (err) {
      console.log(err);
     });
    });
  });
 </script>
</body>
```



- 下载文件

  采用签名URL的方式生成文件的下载链接，您只需要单击链接即可下载文件。

```
<body>
 <input type="button" id="download" value="Download" />
 <script type="text/javascript">
  document.getElementById('download').addEventListener('click', function (e) {
   let objectKey = 'test/download_file';
   let saveAs = 'download_file';
   console.log(objectKey + ' => ' + saveAs);
   // OSS.urlib是SDK内部封装的发送请求的逻辑，开发者可以使用任何发送请求的库向sts-server发送请求。
   OSS.urllib.request("http://your_sts_server/", {method: 'GET'}, (err, response) => {
     if (err) {
      return alert(err);
     }
     try {
      result = JSON.parse(response);
     } catch (e) {
      return alert('parse sts response info error: ' + e.message);
     }
     let client = new OSS({
      accessKeyId: result.AccessKeyId,
      accessKeySecret: result.AccessKeySecret,
      stsToken: result.SecurityToken,
      // region表示您申请OSS服务所在的地域，例如oss-cn-hangzhou。
      region: '<Your region>',
      bucket: '<Your bucket name>'
     });
     let result = client.signatureUrl(objectKey, {
      expires: 3600,
      response: {
       'content-disposition': 'attachment; filename="' + saveAs + '"'
      }
     });
     console.log(result);
     window.location = result;
    });
  });
 </script>
</body>
```

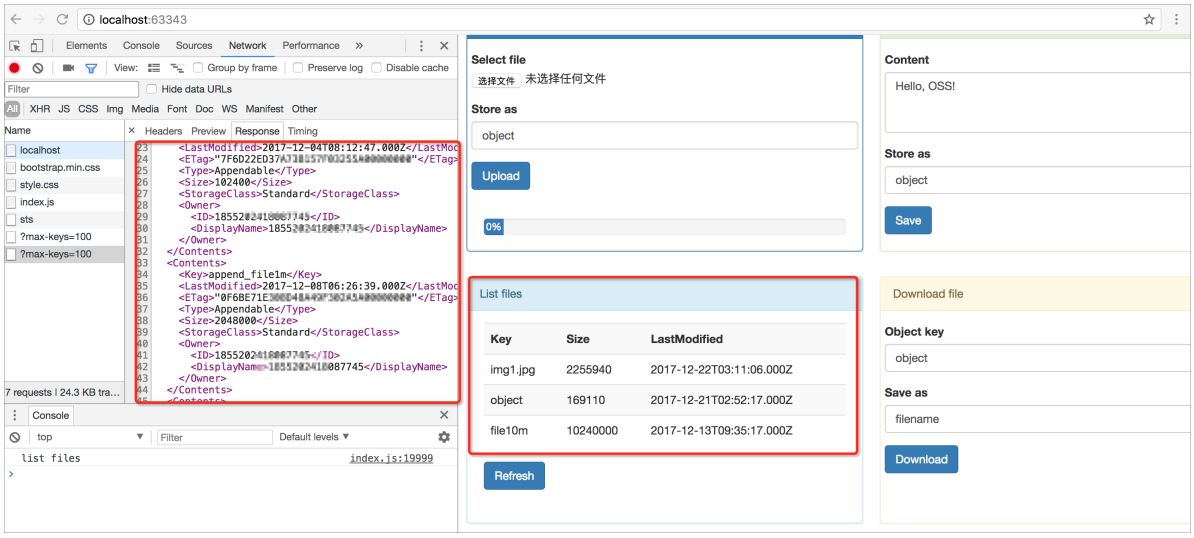


- 查看文件列表

  以下代码用于查看文件列表。

```
<script type="text/javascript">
OSS.urllib.request("http://your_sts_server/", {method: 'GET'}, (err, response) => {
 if (err) {
  return alert(err);
 }
 try {
  result = JSON.parse(response);
 } catch (e) {
  return alert('parse sts response info error: ' + e.message);
 }
 let client = new OSS({
  accessKeyId: result.AccessKeyId,
  accessKeySecret: result.AccessKeySecret,
  stsToken: result.SecurityToken,
  // region表示您申请OSS服务所在的地域，例如oss-cn-hangzhou。
  region: '<Your region>',
  bucket: '<Your bucket name>'
 });
 client.list({
  'max-keys': 10
 }).then(function (result) {
  console.log(result);
 }).catch(function (err) {
  console.log(err);
 });
});
</script>
```

在浏览器中打开HTML文件，然后打开Chrome开发者工具，即可查看列举文件的结果。



6.4. 上传文件

您可以通过简单上传或分片上传的方式将文件或数据上传到OSS。

> **说明** 关于搭建STS服务的具体操作，请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。

简单上传

您可以通过简单上传（即putObject方式）将File对象、Blob数据以及OSS Buffer上传到OSS文件。简单上传时不支持使用进度函数。

> **说明** Blob（Binary Large Object）代表二进制类型的大对象。

以下代码用于上传数据到examplebucket中exampledir目录下的exampleobject.txt文件。

```
const OSS = require('ali-oss');
const client = new OSS({
  // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 从STS服务获取的安全令牌（SecurityToken）。
  stsToken: 'yourSecurityToken',
  // 填写Bucket名称。
  bucket: 'examplebucket'
 });
// 从输入框获取file对象，例如<input type="file" id="file" />。
const data = document.getElementById('file').files[0];
// 创建并填写Blob数据。
//const data = new Blob('Hello OSS');
// 创建并填写OSS Buffer内容。
//const data = new OSS.Buffer('Hello OSS');
async function putObject () {
 try {
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  // 您可以通过自定义文件名（例如exampleobject.txt）或文件完整路径（例如exampledir/exampleobject.txt）的形式实现将数据上传到当前Bucket或Bucket中的指定目录。
  // data对象可以自定义为file对象、Blob数据或者OSS Buffer。
  const result = await client.put('exampledir/exampleobject.txt', data);
  console.log(result);
 } catch (e) {
  console.log(e);
 }
}
putObject();
```

分片上传

当需要上传的文件较大时，您可以通过MultipartUpload接口进行分片上传。分片上传是指将要上传的文件分成多个数据块（Part）来分别上传。当其中一些分片上传失败后，OSS将保留上传进度记录，再次重传时只需要上传失败的分片，而不需要重新上传整个文件。一般大于100 MB的文件，建议采用分片上传的方法，通过断点续传和重试，提高上传成功率。

在使用MultipartUpload接口时，如果遇到 `ConnectionTimeoutError` 超时问题，业务方需自行处理超时逻辑。例如通过缩小分片大小、增加超时时间、重试请求或者捕获 `ConnectionTimeoutError` 错误等方法处理超时。更多信息，请参见网络错误处理。

> **说明**
>
> - checkpoint参数用于记录上传进度，断点续传上传时将记录的checkpoint参数传入即可。
> - 每次进行分片上传时建议使用一个新的OSS实例。
> - 相关参数说明请参见GitHub。

以下代码用于断点续传上传。

```
const OSS = require('ali-oss')
const client = new OSS({
```

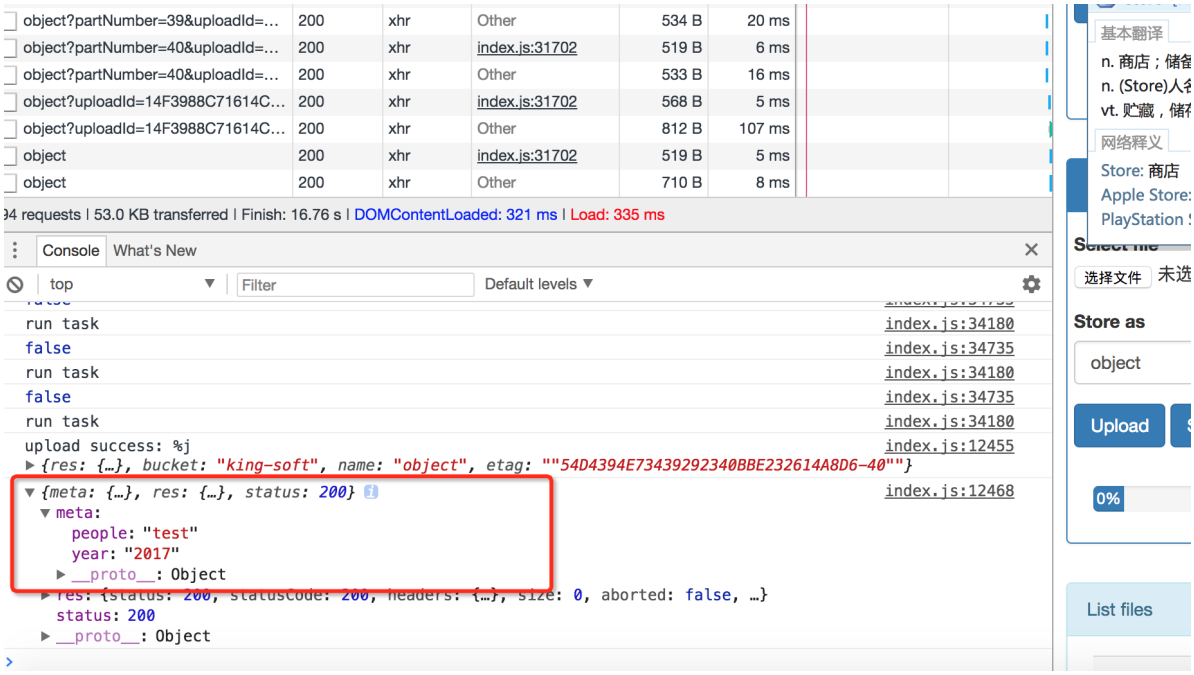
```
  // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 从STS服务获取的安全令牌（SecurityToken）。
  stsToken: 'yourSecurityToken',
  // 填写Bucket名称，例如examplebucket。
  bucket: 'examplebucket'
 });
const client = new OSS(ossConfig);
// 从输入框获取file对象，例如<input type="file" id="file" />。
const file = document.getElementById('file').files[0];
let tempCheckpoint;
// 定义上传方法。
async function multipartUpload () {
 try {
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  // 您可以通过自定义文件名（例如exampleobject.txt）或目录（例如exampledir/exampleobject.txt）的形式，实现将文件上传到当前Bucket或Bucket中的指定目录。
  const result = await client.multipartUpload('exampleobject.txt', file, {
   progress: function (p, checkpoint) {
    // 断点记录点。浏览器重启后无法直接继续上传，您需要手动触发上传操作。
    tempCheckpoint = checkpoint;
   },
   meta: { year: 2020, people: 'test' },
   mime: 'text/plain'
  })
 } catch(e){
  console.log(e);
 }
}
// 开始分片上传。
multipartUpload();
// 暂停分片上传。
client.cancel();
// 恢复上传。
const resumeclient = new OSS(ossConfig);
async function resumeUpload () {
 try {
  const result = await resumeclient.multipartUpload('exampleobject.txt', file, {
   progress: function (p, checkpoint) {
    tempCheckpoint = checkpoint;
   },
   checkpoint: tempCheckpoint,
   meta: { year: 2020, people: 'test' },
   mime: 'text/plain'
 })
 } catch (e) {
  console.log(e);
 }
}
resumeUpload();
```

- progress参数为进度回调函数，用于获取上传进度。

```
const progress = function progress(p, checkpoint) {
  console.log(p)
};
```

- meta参数是用户自定义的元数据，通过HeadObject接口可以获取到Object的元数据。同时您需要在OSS控制台**权限管理**页签的**跨域设置**中设置暴露Header。具体操作，请参见设置跨域访问。

  请求成功后的返回结果如下：



- 在options参数中添加 `callback` 即可实现上传回调通知，代码如下：

```
callback: {
  url: 'http://oss-demo.aliyuncs.com:23450',
  host: 'oss-cn-hangzhou.aliyuncs.com',
  /* eslint no-template-curly-in-string: [0] */
  body: 'bucket=${bucket}&object=${object}&var1=${x:var1}',
  contentType: 'application/x-www-form-urlencoded',
  customValue: {
   var1: 'value1',
   var2: 'value2',
  },
},
```

6.5. 下载文件

在浏览器中使用 `signatureUrl` 方法生成用于预览或下载的文件URL。您可以通过HTML中<a>标签的download属性、Web API中的window.open等方式使用获取的文件URL。

> **说明**
>
> 搭建STS服务的具体操作请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKey ID和AccessKey Secret）和安全令牌（SecurityToken）。

获取文件的预览URL

以下代码用于获取examplebucket中exampleobject.txt文件的预览URL。

> **说明** 如果您要直接在浏览器中预览文件，请设置文件HTTP头中的Content-Disposition为inline并使用Bucket绑定的自定义域名进行访问。具体操作，请分别参见设置文件元信息和自定义域名绑定。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 从STS服务获取的安全令牌（SecurityToken）。
 stsToken: 'yoursecurityToken',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
let url;
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。文件URL的有效时长默认为1800秒，即30分钟。
url = client.signatureUrl('exampleobject.txt');
console.log(url);
// 设置URL的有效时长为3600，单位为秒。如果不设置有效时长，则默认值为1800。
//url = client.signatureUrl('exampleobject.txt', {expires: 3600});
//console.log(url);
```

获取文件的下载URL

以下代码用于获取examplebucket中exampleobject.txt文件的下载URL。URL的有效时长默认为1800秒。

```
const OSS = require('ali-oss');
const client = new OSS({
 // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
 region: 'yourRegion',
 // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
 accessKeyId: 'yourAccessKeyId',
 accessKeySecret: 'yourAccessKeySecret',
 // 从STS服务获取的安全令牌（SecurityToken）。
 stsToken: 'yoursecurityToken',
 // 填写Bucket名称。
 bucket: 'examplebucket',
});
// 配置响应头实现通过URL访问时自动下载文件，并设置下载后的文件名。
const filename = 'examplefile.txt' // 自定义下载后的文件名。
const response = {
 'content-disposition': `attachment; filename=${encodeURIComponent(filename)}`
}
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
const url = client.signatureUrl('exampleobject.txt', { response });
console.log(url);
```

6.6. 管理文件

OSS Browser.js SDK提供一系列的接口方便用户管理某个Bucket下的多个文件。

查看所有文件

通过 `list` 来列出当前Bucket下的所有文件。主要的参数如下：

- prefix指定只列出符合特定前缀的文件
- marker指定只列出文件名大于marker之后的文件
- delimiter用于获取文件的公共前缀
- max-keys用于指定最多返回的文件个数

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function list (dir) {
  try {
    // 不带任何参数，默认最多返回1000个文件
    let result = await client.list();
    console.log(result);
    // 根据nextMarker继续列出文件
    if (result.isTruncated) {
      let result = await client.list({ marker : result.nextMarker });
    }
    // 列出前缀为'my-'的文件
    let result = await client.list({
      prefix: 'my-'
    });
    console.log(result);
    // 列出前缀为'my-'且在'my-object'之后的文件
    let result = await client.list({
      prefix: 'my-',
      marker: 'my-object'
    });
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
list();
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在一个Bucket中的所有文件都是通过文件的key唯一标识，并没有层级的结构。这种结构可以让OSS的存储非常高效，但是用户管理文件时希望能够像传统的文件系统一样把文件分门别类放到不同的“目录”下面。通过OSS提供的“公共前缀”的功能，也可以很方便地模拟目录结构。公共前缀的概念请参考列举Object。

假设Bucket中已有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

接下来我们实现一个函数叫 `listDir` ，列出指定目录下的文件和子目录：

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function listDir (dir) {
  try {
    let result = await client.list({
      prefix: dir,
      delimiter: '/'
  });
  result.prefixes.forEach(function (subDir) {
    console.log('SubDir: %s', subDir);
  });
  result.objects.forEach(function (obj) {
    console.log('Object: %s', obj.name);
  });
 } catch (e) {
   console.log(e);
 }
}
```

运行结果如下：

```
> listDir('foo/')
=> SubDir: foo/bar/
  SubDir: foo/hello/
  Object: foo/x
  Object: foo/y
> listDir('foo/bar/')
=> Object: foo/bar/a
  Object: foo/bar/b
> listDir('foo/hello/C/')
=> Object: foo/hello/C/1
  Object: foo/hello/C/2
  ...
  Object: foo/hello/C/9999
```

文件元信息

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，称为“元信息”。 这些信息在上传时与文件一起存储。

文件元信息会附在HTTP Headers中， 而HTTP协议规定请求头部不能包含复杂字符，所以元信息只能是简单的ASCII可见字符且不能包含换行。

说明 所有元信息的总大小不能超过8KB。

使用 put 和 multipartUpload 时都可以通过指定 meta 参数来指定文件的元信息：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function multipartUploadWithMeta () {
  try {
    let result = await client.multipartUpload('object-key', 'local-file', {
    meta: {
      year: 2017,
      people: 'mary'
    }
  });
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
multipartUploadWithMeta();
```

删除文件

通过 delete 来删除某个文件：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function delete () {
  try {
    let result = await client.delete('object-key');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
delete();
```

批量删除文件

通过 deleteMulti 来删除一批文件，用户可以通过 quiet 参数来指定是否返回删除的结果：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function deleteMulti () {
  try {
    let result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3']);
    console.log(result);
    let  result = await client.deleteMulti(['obj-1', 'obj-2', 'obj-3'],{
    quiet: true
  });
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
deleteMulti();
```

6.7. 自定义域名绑定

本文介绍如何使用自定义域名绑定。

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是形如 `http://img.my-domain.com/x.jpg` 的格式，用户将图片存储迁移到OSS之后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

- 开通OSS服务并创建Bucket。
- 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）。
- 在官网控制台将img.my-domain.com与创建的Bucket绑定。
- 将图片上传到OSS的这个Bucket中。

这样就可以通过原地址 `http://img.my-domain.com/x.jpg` 访问到存储在OSS上的图片。 绑定自定义域名请参考自定义域名绑定 。

在使用SDK时，也可以使用自定义域名作为endpoint，这时需要将 `cname` 参数设置为true，如下面的例子：

```
let OSS = require('ali-oss')
let client = new OSS({
  endpoint: '<Your endpoint>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  cname: true
});
```

> **注意** 自定义域名已经绑定到某个特定的 Bucket，因此使用 CNAME 时无法使用 list_buckets 接口。

6.8. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS进行临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

> **说明** 关于搭建STS服务的具体操作，请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。临时访问凭证有效时间单位为秒，最小值为900，最大值以当前角色设定的最大会话时间为准。更多信息，请参见设置角色最大会话时间。

以下代码用于客户端使用STS凭证构造签名请求。

```
// 向您搭建的STS服务获取临时访问凭证。
fetch('http://your_sts_server/')
 .then(resp => resp.json())
 .then(result => {
  const store = new OSS({
   // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
   accessKeyId: result.AccessKeyId,
   accessKeySecret: result.AccessKeySecret,
   // 从STS服务获取的安全令牌（SecurityToken）。
   stsToken: result.SecurityToken,
   // 填写Bucket所在地域。以华东1（杭州）为例，设置region为oss-cn-hangzhou。
   region: 'oss-cn-hangzhou',
   // 填写Bucket名称，例如examplebucket。
   bucket: 'examplebucket'
  });
  // 生成签名URL。
  // 填写Object完整路径，例如ossdemo.txt。Object完整路径中不能包含Bucket名称。
  const url = store.signatureUrl('ossdemo.txt');
  console.log(url);
 })
```

使用签名URL进行临时授权

> **注意** 如果要在前端使用带可选参数的签名URL，请确保在服务端生成该签名URL时设置的Content-Type与在前端使用时设置的Content-Type一致，否则可能出现SignatureDoesNotMatch错误。设置Content-Type的具体操作，请参见如何设置Content-Type（MIME）？。

- 生成签名URL

  您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为1800秒，最大值为32400秒。

- 生成对象签名URL

  > **说明** name {String}表示存放在OSS的Object名称，[expires] {Number}表示URL过期时间，默认值为1800秒。其他参数相关说明，请参见 Github。

  以下代码用于生成对象签名URL。

```
const url = store.signatureUrl('ossdemo.txt');
console.log(url);
// ------------------------------------------------
const url = store.signatureUrl('ossdemo.txt', {
  expires: 3600,
  method: 'PUT'
});
console.log(url);
// put object with signatureUrl
// ------------------------------------------------
const url = store.signatureUrl('ossdemo.txt', {
  expires: 3600,
  method: 'PUT',
  'Content-Type': 'text/plain; charset=UTF-8',
});
console.log(url);
// ------------------------------------------------
const url = store.signatureUrl('ossdemo.txt', {
  expires: 3600,
  response: {
    'content-type': 'text/custom',
    'content-disposition': 'attachment'
  }
});
console.log(url);
// put operation
```

- 生成图片处理的签名URL

```
const url = store.signatureUrl('ossdemo.png', {
  process: 'image/resize,w_200'
});
console.log(url);
// ----------------------------------------------
const url = store.signatureUrl('ossdemo.png', {
  expires: 3600,
  process: 'image/resize,w_200'
});
console.log(url);
```

6.9. 访问权限

OSS允许您对文件（Object）设置访问权限，方便您控制资源访问的方式。

更多关于访问权限控制的内容请参考访问控制。

对于Object，有四种访问权限：

- default：继承所属的Bucket的访问权限，即与所属Bucket的访问权限一样。
- public-read-write：允许匿名用户读写该Object。
- public-read：允许匿名用户读该Object。
- private：不允许匿名访问，所有的访问都要经过签名。

创建Object时，默认为**继承Bucket**权限。之后您可以通过 `putACL` 来设置Object的其他访问权限。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getACL () {
  try {
    let result = await client.getACL('my-object');
    console.log(result.acl); // default
    await client.putACL('my-object', 'public-read');
    let result = await client.getACL('my-object');
    console.log(result.acl); // public-read
  } catch (e) {
    console.log(e);
  }
}
getACL();
```

注意

- 如果没有设置Object的权限，即Object的ACL为继承Bucket，此时Object 的权限和Bucket权限一致。
- 如果Object的权限为继承Bucket以外的三种权限时，访问该Object进行权限认证时会优先判断Object的权限，此时Bucket的权限设置会被忽略。
- 允许匿名访问时（即设置了 public-read 或者 public-read-write 权限），则可以直接通过浏览器访问，例如 `http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg` 。

6.10. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片

```
let OSS = require('ali-oss');
let client = new OSS({
// 拥有目标Bucket访问权限的AccessKey。阿里云账号的AccessKey拥有所有资源的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
  accessKeyId: 'YourAccessKeyId',
  accessKeySecret: 'YourAccessKeySecret',
// 目标Bucket名称。
  bucket: 'YourBucketName',
// 目标Bucket所在地域的Endpoint，以华东1（杭州）地域为例。
  endpoint: 'http://oss-cn-hangzhou.aliyuncs.com'
});
// 将目标图片缩放为固定宽高100 px。
async function scale () {
  try {
    let result = await client.signatureUrl('YourObjectName', {expires: 3600, process: 'image/resize,m_fixed,w_100,h_100'})
  } catch (e) {
    console.log(e);
  }
}
scale();
```

- 使用多个图片处理参数处理图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
let OSS = require('ali-oss');
let client = new OSS({
// 拥有目标Bucket访问权限的AccessKey。阿里云账号的AccessKey拥有所有资源的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
  accessKeyId: 'YourAccessKeyId',
  accessKeySecret: 'YourAccessKeySecret',
// 目标Bucket名称。
  bucket: 'YourBucketName',
// 目标Bucket所在地域的Endpoint，以华东1（杭州）地域为例。
  endpoint: 'http://oss-cn-hangzhou.aliyuncs.com'
});
// 将目标图片缩放为固定宽高100 px后，再旋转90°。
async function resize () {
  try {
    let result = await client.signatureUrl('YourObjectName', {expires: 3600, process: 'image/resize,m_fixed,w_100,h_100/rotate,90'})
  } catch (e) {
    console.log(e);
  }
}
resize();
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。如何创建图片样式，请参见图片样式。

```
let OSS = require('ali-oss');
let client = new OSS({
// 拥有目标Bucket访问权限的AccessKey。阿里云账号的AccessKey拥有所有资源的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
  accessKeyId: 'YourAccessKeyId',
  accessKeySecret: 'YourAccessKeySecret',
// 目标Bucket名称。
  bucket: 'YourBucketName',
// 目标Bucket所在地域的Endpoint，以华东1（杭州）地域为例。
  endpoint: 'http://oss-cn-hangzhou.aliyuncs.com'
});
// 使用指定图片样式处理目标图片。
async function style () {
  try {
    let result = await client.signatureUrl('YourObjectName', {expires: 3600, process: 'style/YourStyleName'});
  } catch (e) {
    console.log(e);
  }
}
style();
```

图片处理持久化

以下代码用于图片处理持久化：

```
let OSS = require('ali-oss');
let client = new OSS({
 // 原图所在Bucket名称。
 bucket: 'SourceBucketName',
 // 源Bucket所在地域的Endpoint，以华东1（杭州）地域为例。
 endpoint: 'http://oss-cn-hangzhou.aliyuncs.com',
 // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
 accessKeyId: 'YourAccessKeyId',
 accessKeySecret: 'YourAccessKeySecret',
});
// 原图名称。
let sourceImage = 'SourceObjectName';
// 处理后的图片名称。
let targetImage = 'TargetObjectName';
async function processImage(processStr, targetBucket) {
 let result = await client.processObjectSave(
  sourceImage,
  targetImage,
  processStr,
  targetBucket
 );
 console.log(result.res.status);
}
// 将原图缩放为固定宽高100 px后，存储至指定Bucket。
processImage("image/resize,m_fixed,w_100,h_100", "TargetBucketName")
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
let OSS = require('ali-oss');
let client = new OSS({
// 拥有目标Bucket访问权限的AccessKey。阿里云账号的AccessKey拥有所有资源的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。
 accessKeyId: 'YourAccessKeyId',
 accessKeySecret: 'YourAccessKeySecret',
// 目标Bucket名称。
 bucket: 'YourBucketName',
// 目标Bucket所在地域的Endpoint，以华东1（杭州）地域为例。
 endpoint: 'http://oss-cn-hangzhou.aliyuncs.com'
});
// 生成带图片处理参数的文件签名URL，过期时间600秒。
let signUrl = client.signatureUrl('example.jpg', {expires: 600, 'process' : 'image/resize,w_300'});
console.log("signUrl="+signUrl);
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

6.11. 浏览器应用

本文主要介绍浏览器的应用。

目前支持的浏览器如下：

- IE(>=10)和Edge
- 主流版本的Chrome/Firefox/Safari
- 主流版本的Android/iOS/WindowsPhone系统默认浏览器

> **注意** 在IE中使用需要在引入SDK之前引入promise库。

低版本浏览器

SDK是通过File API进行文件操作，在一些较低版本的浏览器中运行会出现问题。建议使用第三方插件，通过对OSS API的调用，实现上传文件、下载文件等操作。具体范例请参见Web端直传实践。

- 相关设置
  - Bucket设置

    从浏览器中直接访问OSS，需要对Bucket的CORS属性进行设置。

    - 将来源设置成 `*`
    - 将Method设置成 `GET, POST, PUT, DELETE, HEAD`
    - 将Allowed Header设置成 `*`
    - 将Expose Header设置成 `etag`

    > **注意** 请将该条CORS规则设置成所有CORS规则的第一条。

  - STS设置

    为了不在网页中暴露AccessKeyId和AccessKeySecret，我们采用STS进行临时授权。授权服务器由用户维护，只有认证的用户才能向授权服务器申请临时权限。设置子账号和STS的Policy，请参考Node.js STS AppServer搭建自己的授权服务器。

- 示例

  使用SDK开发一个网页应用，包含上传文件、上传文本、列举文件和下载文件。完整的示例代码可参考oss-in-browser。

  以上传文件为例：

  i. 创建网页

  - 一个文件选择框，用于选择需要上传的文件。
  - 一个文本框，用于输入上传到OSS Object的名字。
  - 一个按钮，用于开始上传。
  - 一个进度条，用于显示上传的进度。

  下面的代码创建了这4个控件，其中 `class="xxx"` 是为了使用Bootstrap样式。

```
<div class="form-group">
 <label>Select file</label>
 <input type="file" id="file" />
</div>
<div class="form-group">
 <label>Store as</label>
 <input type="text" class="form-control" id="object-key-file" value="object" />
</div>
<div class="form-group">
 <input type="button" class="btn btn-primary" id="file-button" value="Upload" />
</div>
<div class="form-group">
 <div class="progress">
  <div id="progress-bar"
     class="progress-bar"
     role="progressbar"
     aria-valuenow="0"
     aria-valuemin="0"
     aria-valuemax="100" style="min-width: 2em;">
   0%
  </div>
 </div>
</div>
```

ii. 添加样式

添加样式时用到了Bootstrap。在网页的 `<head>` 标签中加入样式：

```
<head>
 <title>OSS in Browser</title>
 <link rel="stylesheet" href="bootstrap.min.css" />
</head>
```

其中 `bootstrap.min.css` 可以在Bootstrap的官网下载。

iii. 添加脚本

此时网页仍然是静态的。您需要为该网页添加JavaScript脚本，从而实现上传文件、下载文件和列举文件等操作。

首先在 `<head>` 标签中引入SDK的开发包：

```
<head>
 <title>OSS in Browser</title>
 <link rel="stylesheet" href="bootstrap.min.css" />
 <script type="text/javascript" src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.js"></scrip
t>
 <script type="text/javascript" src="app.js"></script>
</head>
```

其中 `app.js` 为执行上传文件的代码，内容如下：

```
const appServer = 'http://localhost:3000';
const bucket = 'js-sdk-bucket-sts';
```

```
const bucket = 'js-sdk-bucket-sts';
const region = 'oss-cn-hangzhou';
const urllib = OSS.urllib;
const applyTokenDo = function (func) {
  const url = appServer;
  return urllib.request(url, {
    method: 'GET'
  }).then(function (result) {
    const creds = JSON.parse(result.data);
    const client = new OSS({
      region: region,
      accessKeyId: creds.AccessKeyId,
      accessKeySecret: creds.AccessKeySecret,
      stsToken: creds.SecurityToken,
      bucket: bucket
    });
    return func(client);
  });
};
let currentCheckpoint;
const progress = async function progress(p, checkpoint) {
  currentCheckpoint = checkpoint;
  const bar = document.getElementById('progress-bar');
  bar.style.width = `${Math.floor(p * 100)}%`;
  bar.innerHTML = `${Math.floor(p * 100)}%`;
};
let uploadFileClient;
const uploadFile = function (client) {
  if (!uploadFileClient || Object.keys(uploadFileClient).length === 0) {
    uploadFileClient = client;
  }
  const file = document.getElementById('file').files[0];
  const key = document.getElementById('object-key-file').value.trim() || 'object';
  console.log(`${file.name} => ${key}`);
  const options = {
    progress,
    partSize: 100 * 1024,
    meta: {
      year: 2017,
      people: 'test',
    },
  };
  return client.multipartUpload(key, file, options).then( (res) => {
    console.log('upload success: %j', res);
    currentCheckpoint = null;
    uploadFileClient = null;
  }).catch((err) => {
    if (uploadFileClient && uploadFileClient.isCancel()) {
      console.log('stop-upload!');
    } else {
      console.error(err);
    }
  });
};
window.onload = function () {
```

```
document.getElementById('file-button').onclick = function () {
  applyTokenDo(uploadFile);
}
};
```

上传文件分为以下几个步骤：

a. 向授权服务器申请临时权限。其中授权服务器是网站开发者构建用于向终端用户提供临时授权的服务。开发者可以在授权时为不同的用户提供不同的权限。授权服务器可以参考GitHub，GitHub示例中授权服务器直接向用户返回临时凭证。

b. 使用临时密钥创建OSS Client。

c. 通过 `multipartUpload` 上传选中的文件，并通过 `progress` 参数设置进度条。

上传文本内容、获取文件列表、下载文件等功能请参考代码示例OSS in Browser。

6.12. 异常处理

使用SDK时如果请求出错，会有相应的异常抛出。

服务器端异常通常会包含以下信息：

- status：出错请求的HTTP状态码
- code：OSS的错误码
- message：OSS的错误信息
- requestId：标识该次请求的UUID。当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的帮助

OSS中常见的错误信息请参考OSS错误响应。

调试

在浏览器环境中您可以通过在console中设置 `localStorage.debug` 变量来开启调试：

```
localStorage.debug = 'ali-oss'
```

6.13. 常见问题

本文主要介绍Browser.js SDK中的常见问题与解决方法。

如何调用STS

浏览器是不受信任的环境，如果把AccessKey ID和AccessKey Secret直接保存在浏览器端，存在极高的风险。建议在浏览器环境下使用STS模式进行OSS接口调用。

浏览器中使用STS Server相关文档。

获取STS token后，就可进行SDK初始化操作。

```
<script type="text/javascript">
 $.ajax("http://your_sts_server/",{method: 'GET'},function (err, result) {
  let client = new OSS({
    accessKeyId: result.AccessKeyId,
    accessKeySecret: result.AccessKeySecret,
    stsToken: result.SecurityToken,
    endpoint: '<oss endpoint>',
    bucket: '<Your bucket name>'
  });
 });
</script>
```

如何开启HTTPS访问

初始化SDK时，可传入以下几个参数：

- region：指您申请OSS服务时的区域，例如 `oss-cn-hangzhou` 。完整的区域列表可以在OSS服务节点查看。
- internal：配合 `region` 使用，如果指定 `internal` 为 `true` ，则访问内网节点。
- secure：配合 `region` 使用，如果指定了 `secure` 为 `true` ，则使用HTTPS访问。

```
const client = new OSS({
region,
accessKeyId: creds.AccessKeyId,
accessKeySecret: creds.AccessKeySecret,
stsToken: creds.SecurityToken,
bucket,
secure:true
});
```

- endpoint：例如 `http://oss-cn-hangzhou.aliyuncs.com` ，如果指定了 `endpoint` ，则 `region` 会被忽略， `endpoint` 可以指定HTTPS，也可以是IP形式。

浏览器跨域问题如何解决

在浏览器中使用SDK前，先要设置Bucket的CORS属性。具体操作，请参见相关文档。

如何设置上传文件的用户自定义数据（meta）、文件类型（mime）和请求头（header）

请参见浏览器分片上传。

关于浏览器端断点续传的说明

可以将checkpoint保存到浏览器的localstorage，下次再调用的时候传入checkpoint参数，就可以实现断点续传功能。

如何上传文件到指定目录

给要上传的Object名称前加指定目录前缀即可。更多信息，请参见OSS和文件系统对比。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: 'Your bucket name'
});
client.multipartUpload('base-dir/' +'object-key', 'local-file', {
    progress: async function (p) {
      console.log('Progress: ' + p);
    }
  });
  console.log(result);
}).catch((err) => {
  console.log(err);
});
```

如何上传base64编码的图片

base64先转码成指定格式图片，然后调用OSS上传接口进行上传。更多信息，请参见Github示例。

```
/**
 * base64 to file
 * @param dataurl  base64 content
 * @param filename  set up a meaningful suffix, or you can set mime type in options
 * @returns {File|*}
 */
const dataURLtoFile = function dataURLtoFile(dataurl, filename) {
  const arr = dataurl.split(',');
  const mime = arr[0].match(/:(.*?);/)[1];
  const bstr = atob(arr[1]);
  let n = bstr.length;
  const u8arr = new Uint8Array(n);
  while (n--) {
    u8arr[n] = bstr.charCodeAt(n);
  }
  return new Blob([u8arr], { type: mime });// if env support File, also can use this: return new File([u8arr], filen
ame, { type: mime });
};
// client表示OSS client实例
const uploadBase64Img = function uploadBase64Img(client) {
  // base64格式的内容
  const base64Content = 'data:image:xxxxxxxxxxxxx';
  const filename =  'img.png';
  const imgfile = dataURLtoFile(base64Content, filename);
  //key表示上传的object key ,imgFile表示dataURLtoFile处理后返回的图片
  client.multipartUpload(key, imgfile).then((res) => {
    console.log('upload success: %j', res);
  }).catch((err) => {
    console.error(err);
  });
};
```

如何限制上传文件的大小

在浏览器中可以根据document.getElementById(“file”).files[0].size获取上传文件的大小（字节数）。更多信息，请参见Web端直传实践的post请求。

如何获取上传进度

使用分片上传时，可获取上传进度。

如何获取下载进度

浏览器中无法获取进度，可调用 `signatureUrl` 方法，获取下载地址。更多信息，请参见相关文档。

如何获取Object的签名URL

可调用 `signatureUrl` 方法，获取下载地址。更多信息，请参见相关文档。

如何使用SDK生成的签名URL并进行资源上传

签名URL常用于授权给第三方进行资源的下载和上传操作。下载请参见上一条。SDK中提供signatureUrl API，用于返回一个经过签名的URL，用户直接使用该URL上传或者下载资源即可。利用签名URL上传资源请参见SDK工程示例签名URL上传资源示例。

如何使用表单上传方式上传资源到OSS服务器

请参见Web端直传实践。

如何运行示例工程

进入ali-oss/example执行 `npm run start` 。

如何添加上传回调

```
const uploadFile = function uploadFile(client) {
if (!uploadFileClient || Object.keys(uploadFileClient).length === 0) {
uploadFileClient = client;
}
const file = document.getElementById('file').files[0];
const key = document.getElementById('object-key-file').value.trim() || 'object';
console.log(` ${file.name} => ${key}` );
const options = {
progress,
partSize: 500 * 1024,
timeout:60000,
meta: {
year: 2017,
people: 'test',
},
callback: {
//此处是添加上传回调的位置。
url: 'https://uploadtest.xueersi.com/v2/sync',
/* host: 'oss-cn-shenzhen.aliyuncs.com', */
/* eslint no-template-curly-in-string: [0] */
body: 'bucket=${bucket}&object=${object}&var1=${x:var1}',
contentType: 'application/x-www-form-urlencoded',
customValue: {
var1: 'value1',
var2: 'value2',
},
},
```

关于上传回调的更多信息，请参见原理介绍。

常见错误参考

- SDK开启异常日志
- OSS常见错误

6.14. 日志收集

本文介绍如何使用 OSS js SDK 进行日志收集。

开发者使用 OSS js SDK 开发应用产品时，一旦产品面向广大用户，由于产品运行在不同的终端，用户的操作千差万异，用户遇到问题时如果没有相关日志信息，很难定位问题根源。因此我们提供一套方案协助开发者解决这种问题，如何收集和使用日志收集请参考文档OSS js SDK浏览器端如何收集用户日志。

7..NET

7.1. 前言

OSS C# SDK适用于 .NET Framework 2.0及以上版本。本文档基于OSS C# SDK 2.8.0编写。

兼容性

- 对于2.x.x 系列SDK：
  - 接口：兼容。
  - 命名空间：兼容。
- 对于1.0.x 系列SDK：
  - 接口：兼容。
  - 命名空间：不兼容。Aliyun.OpenServices.OpenStorageService变更为Aliyun.OSS。

SDK源码和API文档

SDK源码请参见GitHub地址：GitHub。更多信息请参见API Doc。

示例代码

OSS C# SDK提供丰富的示例代码。您可以从GitHub获取示例代码。示例代码包括以下内容：

示例文件	示例内容
PutObjectSample.cs	上传文件
AppendObjectSample.cs	追加上传
DoesObjectExistSample.cs	判断文件是否存在
DeleteObjectsSample.cs	删除文件
CopyObjectSample.cs	拷贝文件
ModifyObjectMetaSample.cs	管理文件元信息
MultipartUploadSample.cs	分片上传
ResumableSample.cs	断点续传上传
GetObjectSample.cs	下载文件
GetObjectByRangeSample.cs	范围下载
GetObjectAclSample.cs	管理文件访问权限
SetObjectAclSample.cs	管理文件访问权限
ListObjectsSample.cs	列举文件

示例文件	示例内容
UrlSignatureSample.cs	授权访问
UploadCallbackSample.cs	上传回调
ProgressSample.cs	上传进度条和下载进度条
CNameSample.cs	使用自定义域名访问OSS（CNAME）
PostPolicySample.cs	表单上传
CreateBucketSample.cs	创建存储空间
DeleteBucketSample.cs	删除存储空间
DoesBucketExistSample.cs	判断存储空间是否存在
ListBucketsSample.cs	列举存储空间
SetBucketAclSample.cs	设置存储空间的访问权限
SetBucketLifecycleSample.cs	生命周期
SetBucketLoggingSample.cs	访问日志
SetBucketRefererSample.cs	防盗链
SetBucketWetbsiteSample.cs	静态网站托管
SetBucketCorsSample.cs	跨域资源共享
ImageProcessSample.cs	图片处理

7.2. 安装

本文介绍如何安装.NET SDK。

环境准备

- Windows
  - 适用于.NET 2.0及以上版本
  - 适用于Visual Studio 2010及以上版本
- Linux/Mac

  适用于Mono 3.12及以上版本

下载SDK

- 直接下载
- 通过GitHub下载
- 历史版本下载

安装SDK

- Windows环境安装
  - NuGet方式安装
    a. 如果您的Visual Studio没有安装NuGet，请先安装NuGet。
    b. 在Visual Studio中新建或者打开已有的项目，选择**工具 > NuGet程序包管理器 > 管理解决方案的NuGet程序包**。
    c. 搜索aliyun.oss.sdk，在结果中找到Aliyun.OSS.SDK（适用于.NET Framework）或Aliyun.OSS.SDK.NetCore（适用于.Net Core），选择最新版本，单击**安装**。
  - DLL引用方式安装
    a. 下载并解压.NET SDK开发包。
    b. 打开Visual Studio的**解决方案资源管理器**，选择您的项目，右击项目名称，选择 **引用 > 添加引用**，在弹出的对话框中选择**浏览**。
    c. 找到SDK解压包的bin目录，选择*Aliyun.OSS.dll*文件，单击**确定**。
  - 项目引入方式安装

    如果是下载了SDK包或者从GitHub上下载了源码，并通过源码安装，操作步骤如下：
    a. 在Visual Studio中，右击选择**解决方案**，在弹出的菜单中单击**添加现有项目**。
    b. 在弹出的对话框中选择*aliyun-oss-sdk.csproj*文件，单击**打开**。
    c. 右击项目名称，选择**引用 > 添加引用**，在弹出的对话框中选择**项目选项卡**，选中aliyun-oss-sdk项目，单击**确定**。
- Unix/Mac环境安装
  - NuGet方式安装
    a. 在Xamarin中新建或者打开已有的项目，选择工具**Add NuGet Packages**。
    b. 搜索到Aliyun.OSS.SDK或Aliyun.OSS.SDK.NetCore，选择最新版本，单击**Add Package**添加到项目应用中。
  - DLL引用方式安装
    a. 下载并解压.NET SDK开发包。
    b. 在Xamarin的**解决方案**中选择您的项目，右击选择**引用**，在弹出的菜单中选择**Edit References**。
    c. 在Edit References对话框中，选择*.Net Assembly*单击**浏览**，找到SDK解压包的bin目录，选中*Aliyun.OSS.dll*文件，单击**Open** 。

7.3. 初始化

OssClient是OSS服务的C#客户端，用于管理存储空间和文件等OSS资源。

新建OSSClient

新建OSSClient时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

- 使用OSS域名新建OSSClient

  以下代码用于使用OSS域名新建OSSClient：

```
using Aliyun.OSS;
const string accessKeyId = "<yourAccessKeyId>";
const string accessKeySecret = "<yourAccessKeySecret>";
const string endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 由用户指定的OSS访问地址、阿里云颁发的AccessKeyId/AccessKeySecret构造一个新的OssClient实例。
var ossClient = new OssClient(endpoint, accessKeyId, accessKeySecret);
```

- 使用自定义域名新建OssClient

  以下代码用于使用自定义域名新建OssClient：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
const string accessKeyId = "<yourAccessKeyId>";
const string accessKeySecret = "<yourAccessKeySecret>";
const string endpoint = "<yourDomain>";
// 创建ClientConfiguration实例，按照您的需要修改默认参数。
var conf = new ClientConfiguration();
// 开启支持CNAME。CNAME是指将自定义域名绑定到存储空间上。
conf.IsCname = true;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

> **说明** 使用自定义域名时无法使用ossClient.listBuckets方法。

配置OssClient

ClientConfiguration是OSSClient的配置类，您可通过此类来配置代理、连接超时、最大连接数等参数。可设置的参数如下：

参数	描述	默认值
ConnectionLimit	最大并发连接数	512
MaxErrorRetry	请求失败后最大的重试次数。	3
ConnectionTimeout	设置连接超时时间，单位：毫秒，默认不超时。	-1
IsCname	Endpoint是否支持Cname。	false
ProgressUpdateInterval	进度条更新间隔，单位：字节。	8096

示例如下：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var conf = new ClientConfiguration();
conf.ConnectionLimit = 512;
conf.MaxErrorRetry = 3;
conf.ConnectionTimeout = 300;
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

- 数据校验

  以下代码用于MD5数据校验：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var conf = new ClientConfiguration();
// 打开MD5校验。设置EnalbeMD5Check对上传下载的数据自动进行MD5校验，默认关闭。
conf.EnalbeMD5Check = true;
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

> **说明** 使用MD5校验时会有一定的性能开销。

- 代理网络

  如果您使用的网络是代理（Proxy）网络，可以配置以下参数访问OSS：

参数	描述	默认值
ProxyHost	代理服务器，如 `8.8.8.8` 或 `abc.def.com` 。	空
ProxyPort	代理端口，如 `3128` 或 `8080` 。	空
ProxyUserName	代码服务账号，可选参数。	无
ProxyPassword	代码服务密码，可选参数。	无

  无账号密码的代理网络访问示例：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var conf = new ClientConfiguration();
conf.ProxyHost = "8.8.8.8";
conf.ProxyPort = 3128;
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

  带账号密码的代理网络访问示例：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var conf = new ClientConfiguration();
conf.ProxyHost = "8.8.8.8";
conf.ProxyPort = 3128;
conf.ProxyUserName = "user";
conf.ProxyPassword = "6666";
var client = new OssClient(endpoint, accessKeyId, accessKeySecret, conf);
```

7.4. 快速入门

本节介绍如何快速使用OSS .NET SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 创建存储空间。
  var bucket = client.CreateBucket(bucketName);
  Console.WriteLine("Create bucket succeeded, {0} ", bucket.Name);
}
catch (Exception ex)
{
  Console.WriteLine("Create bucket failed, {0}", ex.Message);
}
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 上传文件。
  var result = client.PutObject(bucketName, objectName, localFilename);
  Console.WriteLine("Put object succeeded, ETag: {0} ", result.ETag);
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

详情请参见上传文件 。

下载文件

以下代码用于将指定的OSS文件下载到本地文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
var downloadFilename = "<yourDownloadFilename>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 下载文件。
  var result = client.GetObject(bucketName, objectName);
  using (var requestStream = result.Content)
  {
    using (var fs = File.Open(downloadFilename, FileMode.OpenOrCreate))
    {
      int length = 4 * 1024;
      var buf = new byte[length];
      do
      {
        length = requestStream.Read(buf, 0, length);
        fs.Write(buf, 0, length);
      } while (length != 0);
    }
  }
  Console.WriteLine("Get object succeeded");
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var objects = new List<string>();
  ObjectListing result = null;
  string nextMarker = string.Empty;
  do
  {
    var listObjectsRequest = new ListObjectsRequest(bucketName)
    {
      Marker = nextMarker,
    };
    // 列举文件。
    result = client.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
      Console.WriteLine(summary.Key);
      objects.Add(summary.Key);
    }
    nextMarker = result.NextMarker;
  } while (result.IsTruncated);
  Console.WriteLine("List objects of bucket:{0} succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除文件

以下代码用于删除指定文件：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除文件。
  client.DeleteObject(bucketName, objectName);
  Console.WriteLine("Delete object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Delete object failed, {0}", ex.Message);
}
```

7.5. 存储空间

7.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

创建存储空间的完整代码请参见Git Hub。

```
using Aliyun.OSS;
// 初始化OssClient。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 创建存储空间。
public void CreateBucket(string bucketName)
{
  try
  {
    var request = new CreateBucketRequest(bucketName);
    //设置存储空间访问权限ACL。
    request.ACL = CannedAccessControlList.PublicReadWrite;
    //设置数据容灾类型。
    request.DataRedundancyType = DataRedundancyType.ZRS;
    // 创建存储空间。存储空间名称必须全局唯一。
    client.CreateBucket(request);
    Console.WriteLine("Create bucket succeeded");
  }
  catch (Exception ex)
  {
    Console.WriteLine("Create bucket failed. {0}", ex.Message);
  }
}
```

有关存储空间的命名规范，请参见存储空间（Bucket）。创建存储空间的更多详情，请参见PutBucket。

7.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

列举存储空间的完整代码请参见Git Hub。

以下代码用于列举所有存储空间。

```
using Aliyun.OSS;
// 初始化OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 列出账户下所有的存储空间信息。
public void ListBuckets()
{
  try
  {
    var buckets = client.ListBuckets();
    Console.WriteLine("List bucket succeeded");
    foreach (var bucket in buckets)
    {
      Console.WriteLine("Bucket name：{0}，Location：{1}，Owner：{2}", bucket.Name, bucket.Location, b
ucket.Owner);
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("List bucket failed. {0}", ex.Message);
  }
}
```

7.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

判断存储空间的完整代码请参见Git Hub。

以下代码用于判断存储空间examplebucket是否存在。

```
using Aliyun.OSS;
// 初始化OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 判断存储空间是否存在。
public void DoesBucketExist(string bucketName)
{
  try
  {
    var exist = client.DoesBucketExist(bucketName);
    Console.WriteLine("Check object Exist succeeded");
    Console.WriteLine("exist ? {0}", exist);
  }
  catch (Exception ex)
  {
    Console.WriteLine("Check object Exist failed. {0}", ex.Message);
  }
}
```

7.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域（Location）。

以下代码用于获取存储空间所在的地域：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
// Endpoint以杭州为例，其它Region请按实际情况填写。
var endpoint = "<https://oss-cn-hangzhou.aliyuncs.com>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";

// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取存储空间所在的地域。
  var result = client.GetBucketLocation(bucketName);
  Console.WriteLine("Get bucket:{0} Info succeeded ", bucketName);
  Console.WriteLine("bucket Location: {0}", result.Location);

}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

7.5.5. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期、权限信息等。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
// Endpoint以杭州为例，其它Region请按实际情况填写。
var endpoint = "<https://oss-cn-hangzhou.aliyuncs.com>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";

// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 存储空间的信息包括地域（Region或Location）、创建日期（CreationDate）、拥有者（Owner）、权限（Grants）等。
  var bucketInfo = client.GetBucketInfo(bucketName);
  Console.WriteLine("Get bucket:{0} Info succeeded ", bucketName);
  // 获取存储空间所在的地域。
  Console.WriteLine("bucketInfo Location: {0}", bucketInfo.Bucket.Location);
  // 获取存储空间的创建日期。
  Console.WriteLine("bucketInfo CreationDate: {0}", bucketInfo.Bucket.CreationDate);
  // 获取存储空间的数据容灾类型。
  Console.WriteLine("bucketInfo DataRedundancyType: {0}", bucketInfo.Bucket.DataRedundancyType);
  // 获取存储空间的权限信息。
  Console.WriteLine("bucketInfo Grant: {0}", bucketInfo.Bucket.AccessControlList.Grant);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

7.5.6. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间访问权限

设置访问权限的完整代码请参见Git Hub。

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	CannedAccessControlList.Private
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	CannedAccessControlList.PublicRead
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	CannedAccessControlList.PublicReadWrite

以下代码用于设置存储空间的访问权限：

```
using Aliyun.OSS;
// 初始化OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 设置存储空间的访问权限。
public void SetBucketAcl(string bucketName)
{
  try
  {
    // 设置存储空间的访问权限为公共读。
    client.SetBucketAcl(bucketName, CannedAccessControlList.PublicRead);
    Console.WriteLine("Set bucket ACL succeeded");
  }
  catch (Exception ex)
  {
    Console.WriteLine("Set bucket ACL failed. {0}", ex.Message);
  }
}
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间访问权限

获取访问权限的完整代码请参见GitHub。

以下代码用于获取存储空间的访问权限：

```
using Aliyun.OSS;
// 初始化OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 获取存储空间的访问权限。
public void GetBucketAcl(string bucketName)
{
  try
  {
    string bucketName = "your-bucket";
    var acl = client.GetBucketAcl(bucketName);
    Console.WriteLine("Get bucket ACL success"，acl.ACL.ToString());
  }
  catch (Exception ex)
  {
    Console.WriteLine("Get bucket ACL failed. {0}", ex.Message);
  }
}
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

7.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

删除存储空间的完整代码请参见GitHub。

> **说明**
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
using Aliyun.OSS;
// 初始化OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 删除存储空间。
public void DeleteBucket(string bucketName)
{
  try
  {
    client.DeleteBucket(bucketName);
    Console.WriteLine("Delete bucket succeeded");
  }
  catch (Exception ex)
  {
    Console.WriteLine("Delete bucket failed. {0}", ex.Message);
  }
}
```

删除存储空间的更多详情，请参见DeleteBucket。

7.5.8. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

注意事项

使用存储空间标签时，有如下注意事项：

- 只有Bucket的拥有者及授权子账户才能为Bucket设置用户标签，否则返回403 Forbidden错误，错误码：AccessDenied。
- 最多可设置20对Bucket用户标签（Key-Value对）。Key和Value必须为UTF-8编码。
- Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
- Value最大长度为128字节，可以为空。
- Put Bucket Tags是覆盖语义，即新设置的标签将覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  //设置Bucket标签。
  var setRequest = new SetBucketTaggingRequest(bucketName);
  var tag1 = new Tag
  {
    Key = "project",
    Value = "projectone"
  };
  var tag2 = new Tag
  {
    Key = "user",
    Value = "jsmith"
  };
  setRequest.AddTag(tag1);
  setRequest.AddTag(tag2);
  client.SetBucketTagging(setRequest);
  Console.WriteLine("Set bucket:{0} Tagging succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关设置Bucket标签的详情请参见PutBucketTags。

获取Bucket标签

以下代码用于获取Bucket标签：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取Bucket标签。
  var result = client.GetBucketTagging(bucketName);
  Console.WriteLine("Get bucket:{0} tagging succeeded ", bucketName);
  for (var i = 0; i < result.Tags.Count; i++) {
    Console.WriteLine("bucket tagging key: {0}; bucket tagging value: {1}", result.Tags[i].Key, result.Tags[i].V
alue);
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关获取Bucket标签的详情请参见GetBucketTags。

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 列举带指定标签的Bucket。
  var tag = new Tag
  {
    Key = "tag",
    Value = "value"
  };
  var listRequest = new ListBucketsRequest();
  listRequest.Tag = tag;
  var result = client.ListBuckets(listRequest);
  Console.WriteLine("list bucket:{0} succeeded ", bucketName);
  foreach (var bucket in result.Buckets) {
    Console.WriteLine("bucket name: {0}", bucket.Name);
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除Bucket所有标签

以下代码用于删除Bucket所有的标签：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除Bucket所有的标签。
  client.DeleteBucketTagging(bucketName);
  Console.WriteLine("Delete bucket:{0} Tagging succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关删除Bucket标签详情请参见DeleteBucketTags。

删除Bucket指定标签

以下代码用于删除Bucket指定标签：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var tag = new Tag
  {
    Key = "tag1",
    Value = "value1"
  };
  var tags = new List<Tag>();
   tags.Add(tag);
  // 通过指定key的方法删除Bucket指定标签。
  var request = new DeleteBucketTaggingRequest(bucketName, tags);
  var result = client.DeleteBucketTagging(request);
  Console.WriteLine("delete bucket:{0} Tagging by key succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

7.5.9. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **说明** 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

添加清单配置时，有如下注意事项：

- 单个Bucket最多只能配置1000条清单规则。
- 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单（Inventory）配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var accountId ="<yourDestinationBucketAccountId>";
```

```
var accountId = "<yourDestinationBucketAccountId>";
var roleArn ="<yourDestinationBucketRoleArn>";
var destBucketName ="<yourDestinationBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  //设置Bucket清单。
  var config = new InventoryConfiguration();
  // 设置清单配置id。
  config.Id = "report1";
  // 清单配置是否启用的标识, true或false。
  config.IsEnabled = true;
  // 设置清单筛选规则，指定筛选object的前缀。
  config.Filter = new InventoryFilter("filterPrefix");
  // 创建清单的bucket目的地配置。
  config.Destination = new InventoryDestination();
  config.Destination.OSSBucketDestination = new InventoryOSSBucketDestination();
  // 设置清单格式。
  config.Destination.OSSBucketDestination.Format = InventoryFormat.CSV;
  // 目的地bucket的用户accountId。
  config.Destination.OSSBucketDestination.AccountId = accountId;
  // 目的地bucket的roleArn。
  config.Destination.OSSBucketDestination.RoleArn = roleArn;
  // 目的地bucket的名称。
  config.Destination.OSSBucketDestination.Bucket = destBucketName;
  // 设置产生清单结果的存储路径前缀。
  config.Destination.OSSBucketDestination.Prefix = "prefix1";
  // 设置清单的生成计划，以下示例为每周一次。其中，Weekly对应每周一次，Daily对应每天一次。
  config.Schedule = new InventorySchedule(InventoryFrequency.Daily);
  // 设置清单中包含的object的版本为当前版本。如果设置为InventoryIncludedObjectVersions.All则表示object的
所有版本，在版本控制状态下生效。
  config.IncludedObjectVersions = InventoryIncludedObjectVersions.All;
  // 设置清单中包含的object属性。
  config.OptionalFields.Add(InventoryOptionalField.Size);
  config.OptionalFields.Add(InventoryOptionalField.LastModifiedDate);
  config.OptionalFields.Add(InventoryOptionalField.StorageClass);
  config.OptionalFields.Add(InventoryOptionalField.IsMultipartUploaded);
  config.OptionalFields.Add(InventoryOptionalField.EncryptionStatus);
  config.OptionalFields.Add(InventoryOptionalField.ETag);
  var req = new SetBucketInventoryConfigurationRequest(bucketName, config)
  client.SetBucketInventoryConfiguration(req);
  Console.WriteLine("Set bucket:{0} InventoryConfiguration succeeded", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

有关添加Bucket清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var id = "<BucketInventoryConfigurationId>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 查看指定id的清单配置信息。
  var request = new GetBucketInventoryConfigurationRequest(bucketName, id);
  // 获取Bucket清单。
  var result = client.GetBucketInventoryConfiguration(request);
  Console.WriteLine("Get bucket:{0} BucketInventoryConfiguration succeeded ", bucketName);
  // 打印Bucket清单信息。
  Console.WriteLine("bucket InventoryConfiguration id: {0}; bucket InventoryConfiguration IsEnabled: {1}", r
esult.Configuration.Id, result.Configuration.IsEnabled);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

有关查看Bucket清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明** 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var id = "<BucketInventoryConfigurationId>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 列举清单配置项。默认每次最多列举100条结果，如果配置超过100条，结果将会分页返回，通过传入Token方式列
举下一页。
  string continuationToken = null;
  bool isTruncated = false;
  do {
    var request = new ListBucketInventoryConfigurationRequest(bucketName, continuationToken)
    var result = client.ListBucketInventoryConfiguration(request);
    Console.WriteLine("List bucket:{0} BucketInventoryConfiguration succeeded ", bucketName);
    //打印bucket清单信息
    for (var i = 0; i < result.Configurations.Count; i++) {
      Console.WriteLine("bucket InventoryConfiguration id: {0}; bucket InventoryConfiguration IsEnabled: {1
}", result.Configurations[i].Id, result.Configurations[i].IsEnabled);
    }
    continuationToken = result.NextContinuationToken;
    isTruncated = result.IsTruncated;
  } while (isTruncated)
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

有关批量列举Bucket清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var id = "<BucketInventoryConfigurationId>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除指定id的清单配置。
  var request = new DeleteBucketInventoryConfigurationRequest(bucketName, id);
  var result = client.DeleteBucketInventoryConfiguration(request);
  Console.WriteLine("delete bucket:{0} BucketInventoryConfiguration succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
```

有关删除Bucket清单配置的详情，请参见DeleteBucketInventory。

7.5.10. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

> **说明** 关于生命周期的更多信息，请参见开发指南中的生命周期规则介绍。

设置生命周期规则

以下代码用于设置生命周期规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var setBucketLifecycleRequest = new SetBucketLifecycleRequest(bucketName);
  // 创建第1条生命周期规则。
  LifecycleRule lcr1 = new LifecycleRule()
  {
    ID = "delete obsoleted files",
    Prefix = "obsoleted/",
    Status = RuleStatus.Enabled,
```

```
        ExpriationDays = 3,
        Tags = new Tag[1]
    };
    // 设置标签。
    var tag1 = new Tag
    {
        Key = "project",
        Value = "projectone"
    };
    lcr1.Tags[0] = tag1;
    // 创建第2条生命周期规则。
    LifecycleRule lcr2 = new LifecycleRule()
    {
        ID = "delete temporary files",
        Prefix = "temporary/",
        Status = RuleStatus.Enabled,
        ExpriationDays = 20,
        Tags = new Tag[1]
    };
    // 设置标签。
    var tag2 = new Tag
    {
        Key = "user",
        Value = "jsmith"
    };
    lcr2.Tags[0] = tag2;
    // 设置碎片在距最后修改时间30天后过期。
    lcr2.AbortMultipartUpload = new LifecycleRule.LifeCycleExpiration()
    {
        Days = 30
    };
    LifecycleRule lcr3 = new LifecycleRule();
    lcr3.ID = "only NoncurrentVersionTransition";
    lcr3.Prefix = "test1";
    lcr3.Status = RuleStatus.Enabled;
    lcr3.NoncurrentVersionTransitions = new LifecycleRule.LifeCycleNoncurrentVersionTransition[2]
    {
        // 设置非当前版本的Object距最后修改时间90天之后转为低频访问类型。
        new LifecycleRule.LifeCycleNoncurrentVersionTransition(){
            StorageClass = StorageClass.IA,
            NoncurrentDays = 90
        },
        // 设置非当前版本的Object距最后修改时间180天之后转为归档类型。
        new LifecycleRule.LifeCycleNoncurrentVersionTransition(){
            StorageClass = StorageClass.Archive,
            NoncurrentDays = 180
        }
    };
    setBucketLifecycleRequest.AddLifecycleRule(lcr1);
    setBucketLifecycleRequest.AddLifecycleRule(lcr2);
    setBucketLifecycleRequest.AddLifecycleRule(lcr3);
    // 设置生命周期规则。
    client.SetBucketLifecycle(setBucketLifecycleRequest);
    Console.WriteLine("Set bucket:{0} Lifecycle succeeded ", bucketName);
}
```

```
}
catch (OssException ex)
{
    Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
        ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
    Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 查看生命周期规则。
  var rules = client.GetBucketLifecycle(bucketName);
  Console.WriteLine("Get bucket:{0} Lifecycle succeeded ", bucketName);
  foreach (var rule in rules)
  {
    Console.WriteLine("ID: {0}", rule.ID);
    Console.WriteLine("Prefix: {0}", rule.Prefix);
    Console.WriteLine("Status: {0}", rule.Status);
    // 查看标签信息。
    foreach (var tag in rule.Tags)
    {
      Console.WriteLine("key:{0}, value:{1}", tag.Key, tag.Value);
    }
    // 查看非当前版本Object过期规则。
    foreach (var version in rule.NoncurrentVersionTransitions)
    {
      Console.WriteLine("expiration day:{0}, storage class:{1}", version.NoncurrentDays, version.StorageClas
s);
    }
    if (rule.ExpriationDays.HasValue)
      Console.WriteLine("ExpirationDays: {0}", rule.ExpriationDays);
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 清空生命周期规则。
  client.DeleteBucketLifecycle(bucketName);
  Console.WriteLine("Delete bucket:{0} Lifecycle succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.5.11. 请求者付费模式

请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

  如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

  数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 设置请求者付费模式。
  var request = new SetBucketRequestPaymentRequest(bucketName, RequestPayer.Requester);
  client.SetBucketRequestPayment(request);
  Console.WriteLine("Set bucket:{0} RequestPayment succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取请求者付费模式。
  var result = client.GetBucketRequestPayment(bucketName);
  Console.WriteLine("Set bucket:{0} RequestPayment succeeded; RequestPayment: {1}", bucketName, result
.Payer);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、GetObject和DeleteObject为例，用于指定第三方付费访问Object。其他用于指定第三方付费的Object读写操作接口设置方法类似。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectContent = "More than just cloud.";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  // PutObject接口指定付费者。
  var putRequest =  new PutObjectRequest(bucketName, objectName, requestContent);
  putRequest.RequestPayer = RequestPayer.Requester;
  var result = client.PutObject(putRequest);
  // GetObject接口指定付费者。
  var getRequest = new GetObjectRequest(bucketName, objectName);
  getRequest.RequestPayer = RequestPayer.Requester;
  var getResult = client.GetObject(getRequest);
  // DeleteObject接口指定付费者。
  var delRequest = new DeleteObjectRequest(bucketName, objectName);
  delRequest.RequestPayer = RequestPayer.Requester;
  client.DeleteObject(delRequest);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.5.12. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  //设置Bucket Policy。
  string policy = "{\"Version\":\"1\",\"Statement\":[{\"Action\":[\"oss:PutObject\",\"oss:GetObject\"],\"Resou
rce\": \"acs:oss:*:*:*\",\"Effect\": \"Deny\"}]}\n";
  var request = new SetBucketPolicyRequest(bucketName, policy);
  client.SetBucketPolicy(request);
  Console.WriteLine("Set bucket:{0} Policy succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取Bucket Policy。
  var result = client.GetBucketPolicy(bucketName);
  Console.WriteLine("Get bucket:{0} Policy succeeded ", bucketName);
  Console.WriteLine("Policy: {0}", result.Policy);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关获取Bucket Policy信息详情，请参见GetBucketPolicy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除Bucket Policy。
  client.DeleteBucketPolicy(bucketName);
  Console.WriteLine("Delete bucket:{0} Policy succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

7.5.13. 访问日志

您可以开启存储空间的访问日志记录功能。开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

更多关于访问日志的介绍，请参见开发指南中的设置访问日志记录。访问日志的完整代码请参见GitHub。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var targetBucketName = "<yourTargetBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // targetBucketName为存放日志文件的存储空间，logging-为被保存的访问日志文件前缀。bucketName和target
BucketName可以是同一存储空间。
  var request = new SetBucketLoggingRequest(bucketName, targetBucketName, "logging-");
  // 开启访问日志记录。
  client.SetBucketLogging(request);
  Console.WriteLine("Set bucket:{0} Logging succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

查看访问日志设置

查看访问日志设置完整代码请参见GitHub。

以下代码用于查看存储空间的访问日志设置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 查看存储空间的访问日志设置。
  var result = client.GetBucketLogging(bucketName);
  Console.WriteLine("Get bucket:{0} Logging succeeded, prefix:{1}", bucketName, result.TargetPrefix);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

关闭访问日志记录

关闭访问日志记录完整代码请参见GitHub。

以下代码用于关闭存储空间的访问日志记录：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 关闭访问日志记录，已经存在的日志不受影响。
  client.DeleteBucketLogging(bucketName);
  Console.WriteLine("Delete bucket:{0} Logging succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.5.14. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

跨域资源共享的完整代码请参见GitHub。

设置跨域资源共享规则

设置跨域资源共享规则的完整代码请参见GitHub。

以下代码用于设置指定存储空间的跨域资源共享规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var request = new SetBucketCorsRequest(bucketName);
  var rule1 = new CORSRule();
  // 指定允许跨域请求的来源。
  rule1.AddAllowedOrigin("http://www.a.com");
  // 指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)。
  rule1.AddAllowedMethod("POST");
  // AllowedHeaders和ExposeHeaders不支持通配符。
  rule1.AddAllowedHeader("*");
  // 指定允许用户从应用程序中访问的响应头。
  rule1.AddExposeHeader("x-oss-test");
  // 最多允许10条规则。
  request.AddCORSRule(rule1);
  var rule2 = new CORSRule();
  // AllowedOrigins和AllowedMethods最多支持一个星号（*）通配符。星号（*）表示允许所有的域来源或者操作。
  rule2.AddAllowedOrigin("http://www.b.com");
  rule2.AddAllowedMethod("GET");
  // 是否允许预取指令（OPTIONS）中Access-Control-Request-Headers头中指定的Header。
  rule2.AddExposeHeader("x-oss-test2");
  // 指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，单位为秒。
  rule2.MaxAgeSeconds = 100;
  request.AddCORSRule(rule2);
  // 设置跨域资源共享规则。
  client.SetBucketCors(request);
  Console.WriteLine("Set bucket:{0} Cors succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取跨域资源共享规则

获取跨域资源共享规则完整代码请参见Git Hub。

以下代码用于获取跨域资源共享规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取跨域资源共享规则。
  var result = client.GetBucketCors(bucketName);
  Console.WriteLine("Get bucket:{0} Cors succeeded ", bucketName);
  foreach (var rule in result)
  {
    foreach (var origin in rule.AllowedOrigins)
    {
      Console.WriteLine("Allowed origin:{0}", origin);
    }
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除跨域资源共享规则

删除跨域资源共享规则的完整代码请参见Git Hub。

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除跨域资源共享规则。
  client.DeleteBucketCors(bucketName);
  Console.WriteLine("Delete bucket:{0} Cors succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0} ; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.5.15. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 设置静态网站托管。
  var request = new SetBucketWebsiteRequest(bucketName, "index.html", "error.html");
  client.SetBucketWebsite(request);
  Console.WriteLine("Set bucket:{0} Wetbsite succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 查看静态网站托管配置。
  var result = client.GetBucketWebsite(bucketName);
  Console.WriteLine("Get bucket:{0} Wetbsite succeeded, index doc:{1}, error doc:{2}",
            bucketName, result.IndexDocument, result.ErrorDocument);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除静态网站托管配置。
  client.DeleteBucketWebsite(bucketName);
  Console.WriteLine("Delete bucket:{0} Wetbsite succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error info: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.5.16. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链完整代码请参见GitHub。

设置防盗链

以下代码用于设置防盗链：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var refererList = new List<string>();
  // 添加Referer白名单。Referer参数支持通配符星号（*）和问号（？）。
  refererList.Add(" http://www.aliyun.com");
  refererList.Add(" http://www.*.com");
  refererList.Add(" http://www.?.aliyuncs.com");
  var srq = new SetBucketRefererRequest(bucketName, refererList);
  // 设置防盗链。
  client.SetBucketReferer(srq);
  Console.WriteLine("Set bucket:{0} Referer succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取防盗链信息

获取防盗链信息完整代码请参见Git Hub。

以下代码用于获取防盗链信息：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取防盗链信息。
  var rc = client.GetBucketReferer(bucketName);
  Console.WriteLine("Get bucket:{0} Referer succeeded ", bucketName);
  Console.WriteLine("allow?  " + (rc.AllowEmptyReferer ? "yes" : "no"));
  if (rc.RefererList.Referers != null)
  {
    for (var i = 0; i < rc.RefererList.Referers.Length; i++)
      Console.WriteLine(rc.RefererList.Referers[i]);
  }
  else
  {
    Console.WriteLine("Empty Referer List");
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
finally
{
  client.SetBucketReferer(new SetBucketRefererRequest(bucketName));
}
```

清空防盗链

以下代码用于清空防盗链：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
  var srq = new SetBucketRefererRequest(bucketName);
  client.SetBucketReferer(srq);
  Console.WriteLine("Set bucket:{0} Referer succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.6. 上传文件

7.6.1. 概述

在OSS中，操作的基本数据单元是文件（Object）。OSS .NET SDK提供了丰富的文件上传方式：

- 简单上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

7.6.2. 简单上传

简单上传是指通过PutObject方法上传单个文件（Object）。简单上传包括上传字符串和上传本地文件，您可以使用同步方式或者异步方式进行上传。您也可以使用MD5校验来确保上传过程中的数据完整性。

上传字符串

说明 上传字符串的完整代码请参见GitHub。

以下代码用于将字符串上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
using System.Text;
using Aliyun.OSS;
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
var endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
var accessKeyId = "yourAccessKeyId";
var accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
var bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
var objectName = "exampleobject.txt";
// 填写字符串。
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  // 上传文件。
  client.PutObject(bucketName, objectName, requestContent);
  Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

上传本地文件

说明 上传本地文件的完整代码请参见GitHub。

以下代码用于将本地文件examplefile.txt上传到目标存储空间examplebucket中的exampleobject.txt文件。

```
using Aliyun.OSS;
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
var endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
var accessKeyId = "yourAccessKeyId";
var accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
var bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
var objectName = "exampleobject.txt";
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
var localFilename = "D:\\localpath\\examplefile.txt";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 上传文件。
  client.PutObject(bucketName, objectName, localFilename);
  Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

上传文件并且带MD5校验

> **说明** 上传文件并且带MD5校验的完整代码请参见GitHub。

为保证客户端发送的数据和OSS服务端接收到的数据一致，您可以在ObjectMeta中增加Content-Md5值，OSS服务端会使用该MD5值做校验。

> **注意** 使用MD5校验时，性能会有所下降。

以下代码用于上传本地文件时进行MD5校验。

```
using Aliyun.OSS;
using Aliyun.OSS.Util;
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
var endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
var accessKeyId = "yourAccessKeyId";
var accessKeySecret = "yourAccessKeySecret";
// 填写Bucket名称。
var bucketName = "examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
var objectName = "exampleobject.txt";
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
var localFilename = "D:\\localpath\\examplefile.txt";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 计算MD5。
  string md5;
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    md5 = OssUtils.ComputeContentMd5(fs, fs.Length);
  }
  var objectMeta = new ObjectMetadata
  {
    ContentMd5 = md5
  };
  // 上传文件。
  client.PutObject(bucketName, objectName, localFilename, objectMeta);
  Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

异步上传

> **说明** 异步上传文件的完整代码请参见GitHub。

以下代码用于以异步方式将本地文件examplefile.txt上传到目标存储空间examplebucket中的exampleobject.txt文件。使用异步上传时，您需要实现自己的回调处理函数。

```
using System;
using System.IO;
using System.Threading;
using Aliyun.OSS;
using Aliyun.OSS.Common;
// 异步上传。
namespace AsyncPutObject
{
```

```
{
  class Program
  {
    // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-
hangzhou.aliyuncs.com。
    static string endpoint = "yourEndpoint";
    // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常
运维，请登录RAM控制台创建RAM用户。
    static string accessKeyId = "yourAccessKeyId";
    static string accessKeySecret = "yourAccessKeySecret";
    // 填写Bucket名称。
    static string bucketName = "examplebucket";
    // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
    static string objectName = "exampleobject.txt";
    // 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
    static string localFilename = "D:\\localpath\\examplefile.txt";
    static AutoResetEvent _event = new AutoResetEvent(false);
    // 创建OssClient实例。
    static OssClient client = new OssClient(endpoint, accessKeyId, accessKeySecret);
    private static void PutObjectCallback(IAsyncResult ar)
    {
      try
      {
        client.EndPutObject(ar);
        Console.WriteLine(ar.AsyncState as string);
        Console.WriteLine("Put object succeeded");
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
      }
      finally
      {
        _event.Set();
      }
    }
    public static void AsyncPutObject()
    {
      try
      {
        using (var fs = File.Open(localFilename, FileMode.Open))
        {
          var metadata = new ObjectMetadata();
          // 增加自定义元信息。
          metadata.UserMetadata.Add("mykey1", "myval1");
          metadata.UserMetadata.Add("mykey2", "myval2");
          metadata.CacheControl = "No-Cache";
          metadata.ContentType = "text/txt";
          string result = "Notice user: put object finish";
          // 异步上传。
          client.BeginPutObject(bucketName, objectName, fs, metadata, PutObjectCallback, result.ToCharAr
ray());
          _event.WaitOne();
        }
      }
```

```
        }
        catch (OssException ex)
        {
          Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
            ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
        }
        catch (Exception ex)
        {
          Console.WriteLine("Failed with error info: {0}", ex.Message);
        }
      }
      static void Main(string[] args)
      {
        Program.AsyncPutObject();
      }
    }
  }
```

7.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。有关追加上传完整代码的更多信息，请参见GitHub。

示例代码

以下代码用于追加上传：

```
using Aliyun.OSS;
// Endpoint以杭州为例，其它Region请按实际情况填写。
var endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
// 设置Bucket名称。
var bucketName = "<yourBucketName>";
// 设置Object名称。即不包含Bucket名称在内的Object的完整路径，例如example/folder/abc.jpg。
var objectName = "<yourObjectName>";
// 设置本地文件名称。即填写本地文件的完整路径，例如/users/local/abc.jpg。
var localFilename = "<yourLocalFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
```

```
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
long position = 0;
try
{
  var metadata = client.GetObjectMetadata(bucketName, objectName);
  position = metadata.ContentLength;
}
catch (Exception) { }
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    // 追加文件。
    var result = client.AppendObject(request);
    // 设置文件的追加位置。
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}", position);
  }
  // 获取追加位置，再次追加文件。
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    var result = client.AppendObject(request);
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}", position);
  }
}
catch (Exception ex)
{
  Console.WriteLine("Append object failed, {0}", ex.Message);
}
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

7.6.4. 断点续传上传

断点续传上传是指将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

断点续传完整代码请参见GitHub。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
string checkpointDir = "<yourCheckpointDir>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 通过UploadFileRequest设置多个参数。
  UploadObjectRequest request = new UploadObjectRequest(bucketName, objectName, localFilename)
  {
    // 指定上传的分片大小。
    PartSize = 8 * 1024 * 1024,
    // 指定并发线程数。
    ParallelThreadCount = 3,
    // checkpointDir保存断点续传的中间状态，用于失败后继续上传。如果checkpointDir为null，断点续传功能不会
生效，每次失败后都会重新上传。
    CheckpointDir = checkpointDir,
  };
  // 断点续传上传。
  client.ResumableUploadObject(request);
  Console.WriteLine("Resumable upload object:{0} succeeded", objectName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用InitiateMultipartUploadRequest方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

调用UploadPartRequest方法上传分片数据。

> 说明
> - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
> - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
> - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用CompleteMultipartUploadRequest方法将所有分片合并成完整的文件。

分片上传的完整代码请参见GitHub。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 初始化分片上传。
var uploadId = "";
try
{
  // 定义上传的文件及所属存储空间的名称。您可以在InitiateMultipartUploadRequest中设置ObjectMeta，但不必指定其中的ContentLength。
  var request = new InitiateMultipartUploadRequest(bucketName, objectName);
  var result = client.InitiateMultipartUpload(request);
  uploadId = result.UploadId;
  // 打印UploadId。
  Console.WriteLine("Init multi part upload succeeded");
  Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
  Console.WriteLine("Init multi part upload failed, {0}", ex.Message);
}
// 计算分片总数。
var partSize = 100 * 1024;
var fi = new FileInfo(localFilename);
var fileSize = fi.Length;
var partCount = fileSize / partSize;
if (fileSize % partSize != 0)
{
  partCount++;
```

```
}
// 开始分片上传。PartETags是保存PartETag的列表，OSS收到用户提交的分片列表后，会逐一验证每个分片数据的有效性。当所有的数据分片通过验证后，OSS会将这些分片组合成一个完整的文件。
var partETags = new List<PartETag>();
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    for (var i = 0; i < partCount; i++)
    {
      var skipBytes = (long)partSize * i;
      // 定位到本次上传的起始位置。
      fs.Seek(skipBytes, 0);
      // 计算本次上传的分片大小，最后一片为剩余的数据大小。
      var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
      var request = new UploadPartRequest(bucketName, objectName, uploadId)
      {
        InputStream = fs,
        PartSize = size,
        PartNumber = i + 1
      };
      // 调用UploadPart接口执行上传功能，返回结果中包含了这个数据片的ETag值。
      var result = client.UploadPart(request);
      partETags.Add(result.PartETag);
      Console.WriteLine("finish {0}/{1}", partETags.Count, partCount);
    }
    Console.WriteLine("Put multi part upload succeeded");
  }
}
catch (Exception ex)
{
  Console.WriteLine("Put multi part upload failed, {0}", ex.Message);
}
// 完成分片上传。
try
{
  var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, objectName, uploadId);
  foreach (var partETag in partETags)
  {
    completeMultipartUploadRequest.PartETags.Add(partETag);
  }
  var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);
  Console.WriteLine("complete multi part succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("complete multi part failed, {0}", ex.Message);
}
```

取消分片上传事件

您可以调用client.AbortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用这个uploadId做任何操作，已经上传的分片数据会被删除。

取消分片上传事件的完整代码请参见Git Hub。

以下代码用于取消分片上传事件。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var uploadId = "";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 初始化分片上传。
try
{
  var request = new InitiateMultipartUploadRequest(bucketName, objectName);
  var result = client.InitiateMultipartUpload(request);
  uploadId = result.UploadId;
  // 打印UploadId。
  Console.WriteLine("Init multi part upload succeeded");
  Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
  Console.WriteLine("Init multi part upload failed, {0}", ex.Message);
}
// 取消分片上传。
try
{
  var request = new AbortMultipartUploadRequest(bucketName, objectName, uploadId);
  client.AbortMultipartUpload(request);
  Console.WriteLine("Abort multi part succeeded， {0}", uploadId);
}
catch (Exception ex)
{
  Console.WriteLine("Abort multi part failed, {0}", ex.Message);
}
```

列举已上传的分片

列举已上传分片的完整代码请参见Git Hub。

以下代码用于列举已上传的分片：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var uploadId = "<yourUploadId>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  PartListing listPartsResult = null;
  var nextMarker = 0;
  do
  {
    var listPartsRequest = new ListPartsRequest(bucketName, objectName, uploadId)
    {
      PartNumberMarker = nextMarker,
    };
    // 列举已上传的分片。
    listPartsResult = client.ListParts(listPartsRequest);
    Console.WriteLine("List parts succeeded");
    // 遍历所有分片。
    var parts = listPartsResult.Parts;
    foreach (var part in parts)
    {
      Console.WriteLine("partNumber: {0}, ETag: {1}, Size: {2}", part.PartNumber, part.ETag, part.Size);
    }
    nextMarker = listPartsResult.NextPartNumberMarker;
  } while (listPartsResult.IsTruncated);
}
catch (Exception ex)
{
  Console.WriteLine("List parts failed, {0}", ex.Message);
}
```

列举分片上传事件

调用ossClient.listMultipartUploads方法列举出所有执行中的分片上传事件，即已初始化但尚未完成或已取消的分片上传事件。可设置的参数如下：

参数	作用	设置方法
prefix	限定返回的文件名称必须以指定的prefix作为前缀。注意使用prefix查询时，返回的文件名称中仍会包含prefix。	ListMultipartUploadsRequest.setPrefix(String prefix)
delimiter	用于对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素。	ListMultipartUploadsRequest.setDelimiter(String delimiter)

参数	作用	设置方法
maxUploads	限定此次返回分片上传事件的最大数目，默认值和最大值均为1000。	ListMultipartUploadsRequest.setMaxUploads(Integer maxUploads)
keyMarker	所有文件名称的字母序大于keyMarker参数值的分片上传事件。可以与uploadIdMarker参数一同使用来指定返回结果的起始位置。	ListMultipartUploadsRequest.setKeyMarker(String keyMarker)
uploadIdMarker	与keyMarker参数一同使用来指定返回结果的起始位置。 如果未设置keyMarker参数，则此参数无效。如果设置了keyMarker参数，则查询结果中包含： • 名称的字母序大于keyMarker参数值的所有文件。 • 文件名称等于keyMarker参数值且uploadId比uploadIdMarker参数值大的所有分片上传事件。	ListMultipartUploadsRequest.setUploadIdMarker(String uploadIdMarker)

列举分片上传事件的完整代码请参见GitHub。

以下代码用于列举分片上传事件。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  MultipartUploadListing multipartUploadListing = null;
  var nextMarker = string.Empty;
  do
  {
    // 列举分片上传事件。
    var request = new ListMultipartUploadsRequest(bucketName)
    {
      KeyMarker = nextMarker,
    };
    multipartUploadListing = client.ListMultipartUploads(request);
    Console.WriteLine("List multi part succeeded");
    // 列举各个分片上传事件的信息。
    foreach (var mu in multipartUploadListing.MultipartUploads)
    {
      Console.WriteLine("Key: {0}, UploadId: {1}", mu.Key, mu.UploadId);
    }
    // 返回结果中isTruncated为false，nextKeyMarker和nextUploadIdMarker作为下次读取的起点。
    nextMarker = multipartUploadListing.NextKeyMarker;
  } while (multipartUploadListing.IsTruncated);
}
catch (Exception ex)
{
  Console.WriteLine("List multi part uploads failed, {0}", ex.Message);
}
```

指定前缀和最大返回条数

以下代码通过指定前缀和最大返回条数来进行分片上传：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 定义前缀。
var prefix = "<yourObjectPrefix>";
// 定义最大返回条数为100。
var maxUploads = 100;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  MultipartUploadListing multipartUploadListing = null;
  var nextMarker = string.Empty;
  do
  {
    // 列举分片上传事件。默认列举1000个分片。
    var request = new ListMultipartUploadsRequest(bucketName)
    {
      KeyMarker = nextMarker,
      // 指定前缀。
      Prefix = prefix,
      // 指定最大返回条数。
      MaxUploads = maxUploads,
    };
    multipartUploadListing = client.ListMultipartUploads(request);
    Console.WriteLine("List multi part succeeded");
    // 列举各个分片上传事件的信息。
    foreach (var mu in multipartUploadListing.MultipartUploads)
    {
      Console.WriteLine("Key: {0}, UploadId: {1}", mu.Key, mu.UploadId);
    }
    nextMarker = multipartUploadListing.NextKeyMarker;
  } while (multipartUploadListing.IsTruncated);
}
catch (Exception ex)
{
  Console.WriteLine("List multi part uploads failed, {0}", ex.Message);
}
```

7.6.6. 进度条

进度条用于指示上传或下载的进度。

进度条的完整代码请参见 Git Hub。

下面的代码以Put Object方法为例，介绍如何使用进度条。

```
using System;
using System.IO;
using System.Text;
using Aliyun.OSS;
using Aliyun.OSS.Common;
namespace PutObjectProgress
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.PutObjectProgress();
      Console.ReadKey();
    }
    public static void PutObjectProgress()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      var localFilename = "<yourLocalFilename>";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      // 带进度条的上传。
      try
      {
        using (var fs = File.Open(localFilename, FileMode.Open))
        {
          var putObjectRequest = new PutObjectRequest(bucketName, objectName, fs);
          putObjectRequest.StreamTransferProgress += streamProgressCallback;
          client.PutObject(putObjectRequest);
        }
        Console.WriteLine("Put object:{0} succeeded", objectName);
      }
      catch (OssException ex)
      {
        Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2}\tHostID: {3}",
          ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
      }
      catch (Exception ex)
      {
        Console.WriteLine("Failed with error info: {0}", ex.Message);
      }
    }
    // 获取上传进度。
    private static void streamProgressCallback(object sender, StreamTransferProgressArgs args)
    {
      System.Console.WriteLine("ProgressCallback - Progress: {0}%, TotalBytes:{1}, TransferredBytes:{2} ",
        args.TransferredBytes * 100 / args.TotalBytes, args.TotalBytes, args.TransferredBytes);
    }
  }
}
```

7.6.7. 上传回调

本文介绍如何使用上传回调。

上传回调的完整代码请参见Git Hub。

以下代码用于上传回调（callback）：

```
using System;
using System.IO;
using System.Text;
using Aliyun.OSS;
using Aliyun.OSS.Common;
using Aliyun.OSS.Util;
namespace Callback
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.PutObjectCallback();
      Console.ReadKey();
    }
    public static void PutObjectCallback()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      var localFilename = "<yourLocalFilename>";
      // 设置回调请求的服务器地址。
      const string callbackUrl = "http://oss-demo.aliyuncs.com:23450";
      // 设置发起回调时请求body的值。
      const string callbackBody = "bucket=${bucket}&object=${object}&etag=${etag}&size=${size}&mimeTyp
e=${mimeType}&" +
                    "my_var1=${x:var1}&my_var2=${x:var2}";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      try
      {
        string responseContent = "";
        var metadata = BuildCallbackMetadata(callbackUrl, callbackBody);
        using (var fs = File.Open(localFilename, FileMode.Open))
        {
          var putObjectRequest = new PutObjectRequest(bucketName, objectName, fs, metadata);
          var result = client.PutObject(putObjectRequest);
          responseContent = GetCallbackResponse(result);
        }
        Console.WriteLine("Put object:{0} succeeded, callback response content:{1}", objectName, responseC
ontent);
      }
      catch (OssException ex)
      {
        Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2}\tHostID: {3}",
```

```
            Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2}\tHostID: {3}",
              ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Failed with error info: {0}", ex.Message);
        }
    }
    // 设置上传回调。
    private static ObjectMetadata BuildCallbackMetadata(string callbackUrl, string callbackBody)
    {
        string callbackHeaderBuilder = new CallbackHeaderBuilder(callbackUrl, callbackBody).Build();
        string CallbackVariableHeaderBuilder = new CallbackVariableHeaderBuilder().
            AddCallbackVariable("x:var1", "x:value1").AddCallbackVariable("x:var2", "x:value2").Build();
        var metadata = new ObjectMetadata();
        metadata.AddHeader(HttpHeaders.Callback, callbackHeaderBuilder);
        metadata.AddHeader(HttpHeaders.CallbackVar, CallbackVariableHeaderBuilder);
        return metadata;
    }
    // 读取上传回调返回的消息内容。
    private static string GetCallbackResponse(PutObjectResult putObjectResult)
    {
        string callbackResponse = null;
        using (var stream = putObjectResult.ResponseStream)
        {
            var buffer = new byte[4 * 1024];
            var bytesRead = stream.Read(buffer, 0, buffer.Length);
            callbackResponse = Encoding.Default.GetString(buffer, 0, bytesRead);
        }
        return callbackResponse;
    }
  }
}
```

> **说明** 上传回调详情请参见开发指南中的上传回调。

7.7. 下载文件

7.7.1. 概述

OSS .NET SDK提供了丰富的文件下载方式：

- 流式下载
- 范围下载
- 断点续传下载

下载过程中，您还可以通过进度条功能查看下载进度。

7.7.2. 流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

流式下载的完整代码请参见GitHub。

以下代码用于将123.jpg文件的流式数据下载到本地D:\localpath路径下的123.jpg：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 填写Object的完整路径，完整路径中不能包含Bucket名称，例如123.jpg。
var objectName = "123.jpg";
// 填写本地文件路径。
var downloadFilename = "D:\\localpath\\examplefile.txt";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 下载文件到流。OssObject包含了文件的各种信息，如文件所在的存储空间、文件名、元信息以及一个输入流。
  var obj = client.GetObject(bucketName, 123.jpg);
  using (var requestStream = obj.Content)
  {
    byte[] buf = new byte[1024];
    var fs = File.Open("D:\\localpath\\examplefile.txt", FileMode.OpenOrCreate);
    var len = 0;
    // 通过输入流将文件的内容读取到文件或者内存中。
    while ((len = requestStream.Read(buf, 0, 1024)) != 0)
    {
      fs.Write(buf, 0, len);
    }
    fs.Close();
  }
  Console.WriteLine("Get object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Get object failed. {0}", ex.Message);
}
```

7.7.3. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

范围下载的完整代码请参见GitHub。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var downloadFilename = "<yourDownloadFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var getObjectRequest = new GetObjectRequest(bucketName, objectName);
  // 设置Range，取值范围为第20至第100字节。
  getObjectRequest.SetRange(20, 100);
  // 范围下载。getObjectRequest的setRange可以实现文件的分段下载和断点续传。
  var obj = client.GetObject(getObjectRequest);
  // 下载数据并写入文件。
  using (var requestStream = obj.Content)
  {
    byte[] buf = new byte[1024];
    var fs = File.Open(downloadFilename, FileMode.OpenOrCreate);
    var len = 0;
    while ((len = requestStream.Read(buf, 0, 1024)) != 0)
    {
      fs.Write(buf, 0, len);
    }
    fs.Close();
  }
  Console.WriteLine("Get object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Get object failed. {0}", ex.Message);
}
```

GetObjectRequest可以设置以下参数：

参数	说明
Range	指定文件传输的范围。
ModifiedSinceConstraint	如果指定的时间早于实际修改时间，则正常传送文件。否则抛出304 Not Modified异常。
UnmodifiedSinceConstraint	如果传入参数中的时间等于或者晚于文件实际修改时间，则正常传输文件。否则抛出412 precondition failed异常。
MatchingETagConstraints	传入一组ETag，如果传入期望的ETag和文件的 ETag匹配，则正常传输文件。否则抛出412 precondition failed异常。

参数	说明
NonmatchingEtagConstraints	传入一组ETag，如果传入的ETag值和文件的ETag不匹配，则正常传输文件。否则抛出304 Not Modified异常。
ResponseHeaderOverrides	自定义OSS返回请求中的一些Header。

7.7.4. 断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

以下代码用于断点续传下载：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
var downloadFilename = "<yourDownloadFilename>";
var checkpointDir = "<yourCheckpointDir>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 通过DownloadObjectRequest设置多个参数。
  DownloadObjectRequest request = new DownloadObjectRequest(bucketName, objectName, downloadFil
ename)
  {
    // 指定下载的分片大小。
    PartSize = 8 * 1024 * 1024,
    // 指定并发线程数。
    ParallelThreadCount = 3,
    // checkpointDir用于保存断点续传进度信息。如果某一分片下载失败，再次下载时会根据文件中记录的点继续下载
。如果checkpointDir为null，断点续传功能不会生效，每次失败后都会重新下载。
    CheckpointDir = checkpointDir,
  };
  // 断点续传下载。
  client.ResumableDownloadObject(request);
  Console.WriteLine("Resumable download object:{0} succeeded", objectName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

断点续传下载详情请参见断点续传下载。

7.7.5. 进度条

进度条用于指示上传或下载的进度。

下载进度条的完整代码请详见GitHub。

下面的代码以GetObject方法为例，介绍如何使用进度条。

```
using System;
using System.IO;
using Aliyun.OSS;
using Aliyun.OSS.Common;
```

```
namespace GetObjectProgress
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.GetObjectProgress();
      Console.ReadKey();
    }
    public static void GetObjectProgress()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      try
      {
        var getObjectRequest = new GetObjectRequest(bucketName, objectName);
        getObjectRequest.StreamTransferProgress += streamProgressCallback;
        // 下载文件。
        var ossObject = client.GetObject(getObjectRequest);
        using (var stream = ossObject.Content)
        {
          var buffer = new byte[1024 * 1024];
          var bytesRead = 0;
          while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)
          {
            // 处理读取的数据（此处代码省略）。
          }
        }
        Console.WriteLine("Get object:{0} succeeded", objectName);
      }
      catch (OssException ex)
      {
        Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
          ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
      }
      catch (Exception ex)
      {
        Console.WriteLine("Failed with error info: {0}", ex.Message);
      }
    }
    private static void streamProgressCallback(object sender, StreamTransferProgressArgs args)
    {
      System.Console.WriteLine("ProgressCallback - Progress: {0}%, TotalBytes:{1}, TransferredBytes:{2} ",
        args.TransferredBytes * 100 / args.TotalBytes, args.TotalBytes, args.TransferredBytes);
    }
  }
}
```

7.8. 管理文件

7.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理软链接

7.8.2. 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 判断文件是否存在。
  var exist = client.DoesObjectExist(bucketName, objectName);
  Console.WriteLine("Object exist ? " + exist);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	CannedAccessControlList.Default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	CannedAccessControlList.Private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	CannedAccessControlList.PublicRead
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	CannedAccessControlList.PublicReadWrite

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限的完整代码请参见GitHub。 获取文件访问权限的完整代码请参见GitHub。

以下是设置并获取文件访问权限的完整代码：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 设置文件权限。
try
{
  // 通过SetObjectAcl设置文件权限。
  client.SetObjectAcl(bucketName, objectName, CannedAccessControlList.PublicRead);
  Console.WriteLine("Set Object:{0} ACL succeeded ", objectName);
}
catch (Exception ex)
{
  Console.WriteLine("Set Object ACL failed with error info: {0}", ex.Message);
}
// 获取文件权限。
try
{
  // 通过GetObjectAcl获取文件权限。
  var result = client.GetObjectAcl(bucketName, objectName);
  Console.WriteLine("Get Object ACL succeeded, Id: {0}  ACL: {1}",
    result.Owner.Id, result.ACL.ToString());
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2}\tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

文件权限的详细说明请参见权限控制。

7.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息，详情请参见开发指南中的文件元信息。

设置文件元信息的完整代码请参见GitHub。 修改文件元信息的完整代码请参见GitHub。

以下代码用于设置、修改和获取文件元信息：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
```

```
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    // 创建上传文件的元信息，可以通过文件元信息设置HTTP header。
    var metadata = new ObjectMetadata()
    {
      // 指定文件类型。
      ContentType = "text/html",
      // 设置缓存过期时间，格式是格林威治时间（GMT）。
      ExpirationTime = DateTime.Parse("2025-10-12T00:00:00.000Z"),
    };
    // 设置上传文件的长度。如超过此长度，则会被截断为设置的长度。如不足，则为上传文件的实际长度。
    metadata.ContentLength = fs.Length;
    // 设置文件被下载时网页的缓存行为。
    metadata.CacheControl = "No-Cache";
    // 设置元信息mykey1值为myval1。
    metadata.UserMetadata.Add("mykey1", "myval1");
    // 设置元信息mykey2值为myval2。
    metadata.UserMetadata.Add("mykey2", "myval2");
    var saveAsFilename = "文件名测试123.txt";
    var contentDisposition = string.Format("attachment;filename*=utf-8''{0}", HttpUtils.EncodeUri(saveAsFi
lename, "utf-8"));
    // 把请求所得的内容存为一个文件的时候提供一个默认的文件名。
    metadata.ContentDisposition = contentDisposition;
    // 上传文件并设置文件元信息。
    client.PutObject(bucketName, objectName, fs, metadata);
    Console.WriteLine("Put object succeeded");
    // 获取文件元信息。
    var oldMeta = client.GetObjectMetadata(bucketName, objectName);
    // 设置新的文件元信息。
    var newMeta = new ObjectMetadata()
    {
      ContentType = "application/octet-stream",
      ExpirationTime = DateTime.Parse("2035-11-11T00:00:00.000Z"),
      // 指定文件被下载时的内容编码格式。
      ContentEncoding = null,
      CacheControl = ""
    };
    // 增加自定义元信息。
    newMeta.UserMetadata.Add("author", "oss");
    newMeta.UserMetadata.Add("flag", "my-flag");
    newMeta.UserMetadata.Add("mykey2", "myval2-modified-value");
    // 通过ModifyObjectMeta方法修改文件元信息。
    client.ModifyObjectMeta(bucketName, objectName, newMeta);
  }
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
```

```
}
```

说明

- HTTP header详情请参见RFC2616。
- 带自定义Header文件元信息，总大小不超过8KB。

7.8.5. 列举文件

本文介绍如何列举文件。

列举文件的完整代码请参见GitHub。

OSS文件按照字母顺序排列。您可以通过ListObjects列出存储空间下的文件。主要的参数如下：

参数	说明
Delimiter	对文件名称进行分组的一个字符。CommonPrefixes是以delimiter结尾，并有共同前缀的文件集合。
Marker	标明本次列举文件的起点。
MaxKeys	列举文件的最大个数。默认为100，最大值为1000。
Prefix	本次查询结果的前缀。

简单列举文件

简单列举文件的完整代码请参见GitHub。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var listObjectsRequest = new ListObjectsRequest(bucketName);
  // 简单列举存储空间下的文件，默认返回100条记录。
  var result = client.ListObjects(listObjectsRequest);
  Console.WriteLine("List objects succeeded");
  foreach (var summary in result.ObjectSummaries)
  {
    Console.WriteLine("File name:{0}", summary.Key);
  }
}
catch (Exception ex)
{
  Console.WriteLine("List objects failed. {0}", ex.Message);
}
```

列举指定个数的文件

以下代码用于列举指定个数的文件。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var listObjectsRequest = new ListObjectsRequest(bucketName)
  {
    // 最大返回200条记录。
    MaxKeys = 200,
  };
  var result = client.ListObjects(listObjectsRequest);
  Console.WriteLine("List objects succeeded");
  foreach (var summary in result.ObjectSummaries)
  {
    Console.WriteLine(summary.Key);
  }
}
catch (Exception ex)
{
  Console.WriteLine("List objects failed, {0}", ex.Message);
}
```

列举指定前缀的文件

以下代码用于列举包含指定前缀（prefix）的文件。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var prefix = "<yourObjectPrefix>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var keys = new List<string>();
  ObjectListing result = null;
  string nextMarker = string.Empty;
  do
  {
    var listObjectsRequest = new ListObjectsRequest(bucketName)
    {
      Marker = nextMarker,
      MaxKeys = 100,
      Prefix = prefix,
    };
    result = client.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
      Console.WriteLine(summary.Key);
      keys.Add(summary.Key);
    }
    nextMarker = result.NextMarker;
  } while (result.IsTruncated);
  Console.WriteLine("List objects of bucket:{0} succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

列举指定marker之后的文件

参数marker代表文件名称。以下代码用于列举指定marker之后的文件。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var marker = "<yourObjectMarker>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var keys = new List<string>();
  ObjectListing result = null;
  string nextMarker = marker;
  do
  {
    var listObjectsRequest = new ListObjectsRequest(bucketName)
    // 若想增大返回文件数目，可以修改MaxKeys参数，或者使用Marker参数分次读取。
    {
      Marker = nextMarker,
      MaxKeys = 100,
    };
    result = client.ListObjects(listObjectsRequest);
    foreach (var summary in result.ObjectSummaries)
    {
      Console.WriteLine(summary.Key);
      keys.Add(summary.Key);
    }
    nextMarker = result.NextMarker;
  // 如果IsTruncated为 true， NextMarker将作为下次读取的起点。
  } while (result.IsTruncated);
  Console.WriteLine("List objects of bucket:{0} succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

通过异步方式列举文件

通过异步方式列举文件完整代码请参见Git Hub。

以下代码用于异步方式列举文件：

```
using System;
using System.IO;
using System.Threading;
using Aliyun.OSS;
namespace AsyncListObjects
{
  class Program
  {
    static string endpoint = "<yourEndpoint>";
    static string accessKeyId = "<yourAccessKeyId>";
    static string accessKeySecret = "<yourAccessKeySecret>";
    static string bucketName = "<yourBucketName>";
    static AutoResetEvent _event = new AutoResetEvent(false);
    // 创建OssClient实例。
    static OssClient client = new OssClient(endpoint, accessKeyId, accessKeySecret);
    static void Main(string[] args)
    {
      Program.AsyncListObjects();
      Console.ReadKey();
    }
    public static void AsyncListObjects()
    {
      try
      {
        var listObjectsRequest = new ListObjectsRequest(bucketName);
        client.BeginListObjects(listObjectsRequest, ListObjectCallback, null);
        _event.WaitOne();
      }
      catch (Exception ex)
      {
        Console.WriteLine("Async list objects failed. {0}", ex.Message);
      }
    }
    // 通过ListObjectCallback方法异步调用结束后执行回调，如果使用异步类型的接口，都需要实现类似接口。
    private static void ListObjectCallback(IAsyncResult ar)
    {
      try
      {
        var result = client.EndListObjects(ar);
        foreach (var summary in result.ObjectSummaries)
        {
          Console.WriteLine("文件名称: {0}", summary.Key);
        }
        _event.Set();
        Console.WriteLine("Async list objects succeeded");
      }
      catch (Exception ex)
      {
        Console.WriteLine("Async list objects failed. {0}", ex.Message);
      }
    }
  }
}
```

列举所有文件

以下代码用于列举指定存储空间下的所有文件：

```
using Aliyun.OSS;
// 初始化OssClient。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 列举所有文件。
public void ListObject(string bucketName)
{
  try
  {
    ObjectListing result = null;
    string nextMarker = string.Empty;
    do
    {
      // 每页列举的文件个数通过maxKeys指定，超过指定数将进行分页显示。
      var listObjectsRequest = new ListObjectsRequest(bucketName)
      {
        Marker = nextMarker,
        MaxKeys = 100
      };
      result = client.ListObjects(listObjectsRequest);
      Console.WriteLine("File:");
      foreach (var summary in result.ObjectSummaries)
      {
        Console.WriteLine("Name:{0}", summary.Key);
      }
      nextMarker = result.NextMarker;
    } while (result.IsTruncated);
  }
  catch (Exception ex)
  {
    Console.WriteLine("List object failed. {0}", ex.Message);
  }
}
```

7.8.6. 删除文件

本文介绍如何单个或者批量删除文件。

> **警告** 请您谨慎使用删除操作，文件删除后将无法恢复。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
using Aliyun.OSS;
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
var endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
var accessKeyId = "yourAccessKeyId";
var accessKeySecret = "yourAccessKeySecret";
var bucketName = "examplebucket";
var objectName = "exampleobject.txt";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除文件。
  client.DeleteObject(bucketName, objectName);
  Console.WriteLine("Delete object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Delete object failed. {0}", ex.Message);
}
```

批量删除文件

批量删除文件时，每次最多删除1000个文件。

返回结果包括如下两种模式，默认返回模式为详细模式，请根据实际选择返回模式。

- 详细模式（verbose）：未设置quietMode或者设置quietMode为false，表示返回所有删除的文件列表。
- 简单模式（quiet）：设置quietMode为true，表示只返回删除失败的文件列表。

以下代码用于删除examplebucket中指定的多个文件。

```
using Aliyun.OSS;
// yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-ha
ngzhou.aliyuncs.com。
var endpoint = "yourEndpoint";
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维
，请登录RAM控制台创建RAM用户。
var accessKeyId = "yourAccessKeyId";
var accessKeySecret = "yourAccessKeySecret";
var bucketName = "examplebucket";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 设置需要删除的多个文件完整路径。文件完整路径中不能包含Bucket名称。
  var keys = new List<string>();
  keys.Add("exampleobject.txt");
  keys.Add("testdir/sampleobject.txt");
  // 设置为详细模式，返回所有删除的文件列表。
  var quietMode = false;
  var request = new DeleteObjectsRequest(bucketName, keys, quietMode);
  // 删除多个文件。
  var result = client.DeleteObjects(request);
  if ((!quietMode) && (result.Keys != null))
  {
    foreach (var obj in result.Keys)
    {
      Console.WriteLine("Delete successfully : {0} ", obj.Key);
    }
  }
  Console.WriteLine("Delete objects succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Delete objects failed. {0}", ex.Message);
}
```

批量删除文件的完整代码请参见Git Hub。

7.8.7. 拷贝文件

本文介绍如何拷贝文件。

拷贝小文件

拷贝小文件的完整代码请参见Git Hub。

> 说明

以下代码用于拷贝小文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var metadata = new ObjectMetadata();
  metadata.AddHeader("mk1", "mv1");
  metadata.AddHeader("mk2", "mv2");
  var req = new CopyObjectRequest(sourceBucket, sourceObject, targetBucket, targetObject)
  {
    // 如果NewObjectMetadata为null则为COPY模式（即拷贝源文件的元信息），非null则为REPLACE模式（覆盖源
文件的元信息）。
    NewObjectMetadata = metadata
  };
  // 拷贝文件。
  client.CopyObject(req);
  Console.WriteLine("Copy object succeeded");
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

拷贝大文件

拷贝大文件完整代码请参见GitHub。

- 分片拷贝

  对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

  i. 使用 InitiateMultipartUploadRequest 方法来初始化一个分片上传事件。
  ii. 使用 UploadPartCopy 方法进行分片拷贝。
  iii. 使用 CompleteMultipartUpload 方法完成文件拷贝。

  以下代码用于分片拷贝文件：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
```

```
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
var uploadId = "";
var partSize = 50 * 1024 * 1024;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 初始化拷贝任务。可以通过InitiateMultipartUploadRequest指定目标文件元信息。
  var request = new InitiateMultipartUploadRequest(targetBucket, targetObject);
  var result = client.InitiateMultipartUpload(request);
  // 打印uploadId。
  uploadId = result.UploadId;
  Console.WriteLine("Init multipart upload succeeded， Upload Id: {0}", result.UploadId);
  // 计算分片数。
  var metadata = client.GetObjectMetadata(sourceBucket, sourceObject);
  var fileSize = metadata.ContentLength;
  var partCount = (int)fileSize / partSize;
  if (fileSize % partSize != 0)
  {
    partCount++;
  }
  // 开始分片拷贝。
  var partETags = new List<PartETag>();
  for (var i = 0; i < partCount; i++)
  {
    var skipBytes = (long)partSize * i;
    var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    // 创建UploadPartCopyRequest。可以通过UploadPartCopyRequest指定限定条件。
    var uploadPartCopyRequest = new UploadPartCopyRequest(targetBucket, targetObject, sourceBucke
t, sourceObject, uploadId)
      {
        PartSize = size,
        PartNumber = i + 1,
        // BeginIndex用来定位此次上传分片开始所对应的位置。
        BeginIndex = skipBytes
      };
    // 调用uploadPartCopy方法来拷贝每一个分片。
    var uploadPartCopyResult = client.UploadPartCopy(uploadPartCopyRequest);
    Console.WriteLine("UploadPartCopy : {0}", i);
    partETags.Add(uploadPartCopyResult.PartETag);
  }
  // 完成分片拷贝。
  var completeMultipartUploadRequest =
  new CompleteMultipartUploadRequest(targetBucket, targetObject, uploadId);
  // partETags为分片上传中保存的partETag的列表，OSS收到用户提交的此列表后，会逐一验证每个数据分片的有
效性。全部验证通过后，OSS会将这些分片合成一个完整的文件。
  foreach (var partETag in partETags)
  {
    completeMultipartUploadRequest.PartETags.Add(partETag);
  }
  var completeMultipartUploadResult = client.CompleteMultipartUpload(completeMultipartUploadReque
st);
```

```
    Console.WriteLine("CompleteMultipartUpload succeeded");
}
catch (OssException ex)
{
    Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
        ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
    Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

- 断点续传拷贝

  如果拷贝中断，可通过断点续传继续拷贝。 断点续传拷贝的完整代码请参见GitHub。

  以下代码用于断点续传拷贝：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
var checkpointDir = @"<yourCheckpointDir>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
    var request = new CopyObjectRequest(sourceBucket, sourceObject, targetBucket, targetObject);
    // checkpointDir目录中会保存断点续传的中间状态，用于失败后，下次继续拷贝。如果checkpointDir为null，
断点续传功能不会生效，每次都会重新拷贝。
    client.ResumableCopyObject(request, checkpointDir);
    Console.WriteLine("Resumable copy new object:{0} succeeded", request.DestinationKey);
}
catch (Exception ex)
{
    Console.WriteLine("Resumable copy new object failed, {0}", ex.Message);
}
```

7.8.8. 解冻文件

本文介绍如何解冻归档（Archive）及冷归档（Cold Archive）类型的文件（Object）。

> **说明** 非归档或冷归档类型的文件，不要调用RestoreObject方法。
>
> 归档及冷归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

解冻归档文件的完整代码请参见GitHub。

以下代码用于解冻归档文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Model;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
int maxWaitTimeInSeconds = 600;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 创建归档存储空间。
  var bucket = client.CreateBucket(bucketName, StorageClass.Archive);
  Console.WriteLine("Create Archive bucket succeeded, {0} ", bucket.Name);
}
catch (Exception ex)
{
  Console.WriteLine("Create Archive bucket failed, {0}", ex.Message);
}
// 上传文件。
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  client.PutObject(bucketName, objectName, requestContent);
  Console.WriteLine("Put object succeeded, {0}", objectName);
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
var metadata = client.GetObjectMetadata(bucketName, objectName);
string storageClass = metadata.HttpMetadata["x-oss-storage-class"] as string;
if (storageClass != "Archive")
{
  Console.WriteLine("StorageClass is {0}", storageClass);
  return;
}
// 解冻文件。
RestoreObjectResult result = client.RestoreObject(bucketName, objectName);
Console.WriteLine("RestoreObject result HttpStatusCode : {0}", result.HttpStatusCode);
```

```
if (result.HttpStatusCode != HttpStatusCode.Accepted)
{
  throw new OssException(result.RequestId + ", " + result.HttpStatusCode + " ,");
}
while (maxWaitTimeInSeconds > 0)
{
  var meta = client.GetObjectMetadata(bucketName, objectName);
  string restoreStatus = meta.HttpMetadata["x-oss-restore"] as string;
  if (restoreStatus != null && restoreStatus.StartsWith("ongoing-request=\"false\"", StringComparison.Inva
riantCultureIgnoreCase))
  {
    break;
  }
  Thread.Sleep(1000);
  maxWaitTimeInSeconds--;
}
if (maxWaitTimeInSeconds == 0)
{
  Console.WriteLine("RestoreObject is timeout. ");
  throw new TimeoutException();
}
else
{
  Console.WriteLine("RestoreObject is successful. ");
}
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。
   - 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
   - 批量（Bulk）：表示5~12小时内完成解冻。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Model;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 解冻冷归档文件。
var request = new RestoreObjectRequest(bucketName, objectName);
// 设置解冻冷归档文件的优先级。
// TierType.Expedited 表示1小时内完成解冻。
// TierType.Standard 表示2~5小时内完成解冻。
// TierType.Bulk 表示5~12小时内完成解冻。
// 配置解冻参数，以设置1小时内完成解冻，解冻状态保持2天为例。
// 第一个参数Days表示保持解冻状态的天数，默认是1天，此参数适用于解冻Archive（归档）与ColdArchive（冷归档
）类型文件。
// 第二个参数Tier表示解冻优先级，只适用于解冻ColdArchive类型文件。
request.Days = 2;
request.Tier = TierType.Expedited;
RestoreObjectResult result = client.RestoreObject(request);
Console.WriteLine("RestoreObject result HttpStatusCode : {0}", result.HttpStatusCode);
```

7.8.9. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

> **说明** 获取软链接要求您对该软链接有读权限。软链接的详细信息请参见Get Symlink。

以下代码用于创建软链接和获取软链接指向的目标文件（Object）名称：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var targetObjectName = "<yourTargetObjectName>";
var symlinkObjectName = "<yourSymlinkObjectName>";
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 上传目标文件。
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  client.PutObject(bucketName, targetObjectName, requestContent);
  // 创建软链接。
  client.CreateSymlink(bucketName, symlinkObjectName, targetObjectName);
  // 获取软链接指向的目标文件名称。
  var ossSymlink = client.GetSymlink(bucketName, symlinkObjectName);
  Console.WriteLine("Target object is {0}", ossSymlink.Target);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接的详细信息请参见GetSymlink。

7.9. 版本控制

7.9.1. 管理版本控制

版本控制应用于存储空间（Bucket）内的所有文件（Object）。通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。有关版本控制状态的更多信息，请参见版本控制介绍。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 初始化OssClient。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 设置存储空间版本控制状态为Enabled。
  client.SetBucketVersioning(new SetBucketVersioningRequest(bucketName, VersioningStatus.Enabled));;
  Console.WriteLine("Create bucket Version succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Create bucket Version failed. {0}", ex.Message);
}
```

有关设置Bucket版本控制状态的更多信息，请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取存储空间版本控制状态信息。
  var result = client.GetBucketVersioning(bucketName);
  Console.WriteLine("Get bucket:{0} Version succeeded ", bucketName);
  Console.WriteLine("bucket version status: {0}", result.Status);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

有关获取Bucket版本控制状态的更多信息，请参见GetBucketVersioning。

列举Bucket中所有Object的版本信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  ObjectVersionList result = null;
  var request = new ListObjectVersionsRequest(bucketName);
  do {
    result = client.ListObjectVersions(request);
    Console.WriteLine("ListObjectVersions succeeded");
    //查看列出的object删除标记的各版本信息。
    foreach (var objectversion in result.ObjectVersionSummaries)
    {
      Console.WriteLine("objectversion key: {0}; objectversion versionid: {1}", objectversion.Key, objectversi
on.VersionId);
    }
    //查看列出的object各版本信息。
    foreach (var deletemarker in result.DeleteMarkerSummaries)
    {
      Console.WriteLine("deletemarker key: {0}; deletemarker versionid: {1}", deletemarker.Key, deletemark
er.VersionId);
    }
    request.KeyMarker = result.NextKeyMarker;
    request.NextVersionIdMarker = result.NextVersionIdMarker ;
  } while (result.IsTruncated)
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本ID为“null”。

以下代码用于简单上传：

```
using System.Text;
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  // 上传文件。
  var result = client.PutObject(bucketName, objectName, requestContent);
  Console.WriteLine("Put object succeeded versionid : {0}", result.VersionId);
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

> **说明**
>
> - 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
> - 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
> - 不支持对当前版本为非Appendable类型的Object（包括Normal Object、Delete Marker等）执行AppendObject操作。

以下代码用于追加上传：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
long position = 0;
try
{
  var metadata = client.GetObjectMetadata(bucketName, objectName);
  position = metadata.ContentLength;
}
catch (Exception) { }
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    // 追加文件。
    var result = client.AppendObject(request);
    // 设置文件的追加位置。
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}, vesionid:{1}", position, result.Ve
rsionId);
  }
  // 获取追加位置，再次追加文件。
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    var result = client.AppendObject(request);
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}, vesionid:{1}", position, result.Ve
rsionId);
  }
}
catch (Exception ex)
{
  Console.WriteLine("Append object failed, {0}", ex.Message);
}
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 初始化分片上传。
var uploadId = "";
try
{
  // 定义上传文件的名字和所属存储空间。在InitiateMultipartUploadRequest中，可以设置ObjectMeta，但不必指定其中的ContentLength。
  var request = new InitiateMultipartUploadRequest(bucketName, objectName);
  var result = client.InitiateMultipartUpload(request);
  uploadId = result.UploadId;
  // 打印UploadId。
  Console.WriteLine("Init multi part upload succeeded");
  Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
  Console.WriteLine("Init multi part upload failed, {0}", ex.Message);
}
// 计算分片总数。
var partSize = 100 * 1024;
var fi = new FileInfo(localFilename);
var fileSize = fi.Length;
var partCount = fileSize / partSize;
if (fileSize % partSize != 0)
{
  partCount++;
}
// 开始分片上传。partETags是保存partETag的列表，OSS收到用户提交的分片列表后，会逐一验证每个分片数据的有效性。 当所有的数据分片通过验证后，OSS会将这些分片组合成一个完整的文件。
var partETags = new List<PartETag>();
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    for (var i = 0; i < partCount; i++)
    {
      var skipBytes = (long)partSize * i;
      // 定位到本次上传起始位置。
```

```
        fs.Seek(skipBytes, 0);
        // 计算本次上传的片大小，最后一片为剩余的数据大小。
        var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
        var request = new UploadPartRequest(bucketName, objectName, uploadId)
        {
          InputStream = fs,
          PartSize = size,
          PartNumber = i + 1
        };
        // 调用UploadPart接口执行上传功能，返回结果中包含了这个数据片的ETag值。
        var result = client.UploadPart(request);
        partETags.Add(result.PartETag);
        Console.WriteLine("finish {0}/{1}", partETags.Count, partCount);
      }
      Console.WriteLine("Put multi part upload succeeded");
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("Put multi part upload failed, {0}", ex.Message);
  }
  // 完成分片上传。
  try
  {
    var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, objectName,
uploadId);
    foreach (var partETag in partETags)
    {
      completeMultipartUploadRequest.PartETags.Add(partETag);
    }
    var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);
    Console.WriteLine("CompleteMultipartUpload succeeded, vesionid:{0}", result.VersionId);
  }
  catch (Exception ex)
  {
    Console.WriteLine("complete multi part failed, {0}", ex.Message);
  }
```

分片上传的详细信息请参见CompleteMultipartUpload。

7.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// objectName 表示您在下载文件时需要指定的文件名称，如abc/efg/123.jpg。
var objectName = "<yourObjectName>";
var downloadFilename = "<yourDownloadFilename>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 下载文件到流。OssObject包含了文件的各种信息，如文件所在的存储空间、文件名、元信息以及一个输入流。
  var request = new GetObjectRequest(bucketName, objectName)
  {
    // 指定Object的版本id。
    VersionId = versionid
  };
  var obj = client.GetObject(request);
  using (var requestStream = obj.Content)
  {
    byte[] buf = new byte[1024];
    var fs = File.Open(downloadFilename, FileMode.OpenOrCreate);
    var len = 0;
    // 通过输入流将文件的内容读取到文件或者内存中。
    while ((len = requestStream.Read(buf, 0, 1024)) != 0)
    {
      fs.Write(buf, 0, len);
    }
    fs.Close();
  }
  Console.WriteLine("Get object succeeded, vesionid:{0}", obj.VersionId);
}
catch (Exception ex)
{
  Console.WriteLine("Get object failed. {0}", ex.Message);
}
```

下载文件的详细信息请参见Get Object。

7.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。

- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的版本ID，此版本ID将会在响应header的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为“null”的版本，且会覆盖原先versionId为“null”的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var metadata = new ObjectMetadata();
  metadata.AddHeader("mk1", "mv1");
  metadata.AddHeader("mk2", "mv2");
  var req = new CopyObjectRequest(sourceBucket, sourceObject, targetBucket, targetObject)
  {
    // 如果NewObjectMetadata为null，则为COPY模式（即拷贝源文件的元信息），非null则为REPLACE模式（即覆盖源文件的元信息）。
    NewObjectMetadata = metadata,
    // 指定Object的版本ID。
    SourceVersionId = versionid
  };
  // 拷贝文件。
  var result = client.CopyObject(req);
  Console.WriteLine("Copy object succeeded, vesionid:{0}", result.VersionId);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
var sourceObjectVersionid = "<yourSourceObjectVersionid>";
var uploadId = "";
var partSize = 50 * 1024 * 1024;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 初始化拷贝任务。您可以通过InitiateMultipartUploadRequest指定目标文件元信息。
  var request = new InitiateMultipartUploadRequest(targetBucket, targetObject);
  var result = client.InitiateMultipartUpload(request);
  // 打印uploadId。
  uploadId = result.UploadId;
  Console.WriteLine("Init multipart upload succeeded， Upload Id: {0}", result.UploadId);
  // 计算分片数。
  var request = new GetObjectMetadataRequest(sourceBucket, sourceObject)
  {
    // 指定Object的版本ID。
    VersionId = sourceObjectVersionid
  };
  var metadata = client.GetObjectMetadata(request);
  var fileSize = metadata.ContentLength;
  var partCount = (int)fileSize / partSize;
  if (fileSize % partSize != 0)
  {
    partCount++;
  }
  // 开始分片拷贝。
  var partETags = new List<PartETag>();
  for (var i = 0; i < partCount; i++)
  {
    var skipBytes = (long)partSize * i;
    var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    // 创建UploadPartCopyRequest。您可以通过UploadPartCopyRequest指定限定条件。
    var uploadPartCopyRequest = new UploadPartCopyRequest(targetBucket, targetObject, sourceBucket, s
```

```
ourceObject, uploadId)
        {
            PartSize = size,
            PartNumber = i + 1,
            // BeginIndex用来定位此次上传分片开始所对应的位置。
            BeginIndex = skipBytes,
            // 指定Object的版本ID。
            VersionId = sourceObjectVersionid
        };
      // 调用uploadPartCopy方法来拷贝每一个分片。
      var uploadPartCopyResult = client.UploadPartCopy(uploadPartCopyRequest);
      Console.WriteLine("UploadPartCopy : {0}", i);
      partETags.Add(uploadPartCopyResult.PartETag);
    }
    // 完成分片拷贝。
    var completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(targetBucket, targetObject, uploadId);
    // partETags为分片上传中保存的partETag的列表，OSS收到用户提交的此列表后，会逐一验证每个数据分片的有效
性。全部验证通过后，OSS会将这些分片合成一个完整的文件。
    foreach (var partETag in partETags)
    {
      completeMultipartUploadRequest.PartETags.Add(partETag);
    }
    var completeMultipartUploadResult = client.CompleteMultipartUpload(completeMultipartUploadRequest)
;
    Console.WriteLine("CompleteMultipartUpload succeeded, vesionid:{0}", completeMultipartUploadResult.
VersionId);
}
catch (OssException ex)
{
    Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
      ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
    Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

分片拷贝的详细信息请参见UploadPartCopy。

7.9.5. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

`x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object进行永久删除：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var versionid = "<yourObjectVersionidOrDelMarkerVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 指定Object的versionId，也可以是删除标记的versionId。
  var request = new DeleteObjectRequest(bucketName, key)
  {
    VersionId = versionid
  };
  client.DeleteObject(request);
  Console.WriteLine("Delete object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Delete object failed. {0}", ex.Message);
}
```

- 临时删除

  以下代码用于不指定versionId对Object进行临时删除：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 不指定versionId对Object进行临时删除，此操作会为Object添加删除标记。
  var result = client.DeleteObject(bucketName, objectName);
  Console.WriteLine("Delete object succeeded, versionid: {0}, DeleteMarker: {1}", result.VersionId, result.D
eleteMarker);
}
catch (Exception ex)
{
  Console.WriteLine("Delete object failed. {0}", ex.Message);
}
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于指定versionId对多个Object及删除标记进行永久删除：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除指定版本的object或指定删除标记版本关联的Object。
  var obj1 = new ObjectIdentifier
  {
    Key = "yourObject1Name",
    VersionId  = "yourObject1NameVersionid"
  };
  var obj2 = new ObjectIdentifier
  {
    Key = "yourObject2Name",
    VersionId  = "yourObject2DelMarkerVersionid"
  };
  IList<ObjectIdentifier> objects = new List<ObjectIdentifier>();
  objects.Add(obj1);
  objects.Add(obj2);
  var request = new DeleteObjectVersionsRequest(bucketName, objects);
  // 发起deleteVersions请求。
  client.DeleteObjectVersions(request);
  Console.WriteLine("DeleteObjectVersions succeeded ");
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var keys = new List<string>();
  var listResult = client.ListObjects(bucketName);
  foreach (var summary in listResult.ObjectSummaries)
  {
    keys.Add(summary.Key);
  }
  // quietMode为true表示简单模式，即返回删除失败的文件列表。quietMode为false表示详细模式，即返回删除成功的文件列表。默认为详细模式。
  var quietMode = false;
  // 通过DeleteObjectsRequest的quietMode参数指定返回模式。
  var request = new DeleteObjectsRequest(bucketName, keys, quietMode);
  // 执行不指定versionId的删除操作后，将为Object添加删除标记。
  var result = client.DeleteObjects(request);
  if ((!quietMode) && (result.Keys != null))
  {
    foreach (var obj in result.Keys)
    {
      Console.WriteLine("Delete successfully : {0} ", obj.Key);
    }
  }
  Console.WriteLine("Delete objects succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Delete objects failed. {0}", ex.Message);
}
```

删除指定前缀的文件

以下代码用于删除指定前缀（prefix）的文件。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var prefix = "<yourkeyPrefix>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  ObjectVersionList result = null;
  var request = new ListObjectVersionsRequest(bucketName)
  {
    //指定前缀。
    Prefix = prefix;
  };
  // 列举所有指定前缀的文件的版本信息并删除这些文件。
  do {
    result = client.ListObjectVersions(request);
    Console.WriteLine("ListObjectVersions succeeded");
    foreach (var deleteversion in result.DeleteMarkerSummaries)
    {
      var request = new DeleteObjectRequest(bucketName, deleteversion.Key)
      {
        VersionId = deleteversion.VersionId
      };
      client.DeleteObject(request);
    }
    foreach (var objectversion in result.ObjectVersionSummaries)
    {
      var request = new DeleteObjectRequest(bucketName, objectversion.Key)
      {
        VersionId = objectversion.VersionId
      };
      client.DeleteObject(request);
    }
    request.KeyMarker = result.NextKeyMarker;
    request.NextVersionIdMarker = result.NextVersionIdMarker ;
  } while (result.IsTruncated)
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.9.6. 解冻文件

在受版本控制的存储空间（Bucket）中，文件（Object）的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Model;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
var versionid = "<yourArchiveObjectVersionid>";
int maxWaitTimeInSeconds = 600;
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
var getrequest = new GetObjectMetadataRequest(bucketName, objectName)
{
  // 指定归档类型Object的版本id。
  VersionId = versionid
};
var metadata = client.GetObjectMetadata(getrequest);
string storageClass = metadata.HttpMetadata["x-oss-storage-class"] as string;
if (storageClass != "Archive")
{
  Console.WriteLine("StorageClass is {0}", storageClass);
  return;
}
// 解冻文件。
var request = new RestoreObjectRequest(bucketName, objectName)
{
  // 指定归档类型Object的版本id。
  VersionId = versionid
};
RestoreObjectResult result = client.RestoreObject(request);
Console.WriteLine("RestoreObject result HttpStatusCode : {0}, versionid: {1}", result.HttpStatusCode, result
.VersionId);
if (result.HttpStatusCode != HttpStatusCode.Accepted)
{
  throw new OssException(result.RequestId + ", " + result.HttpStatusCode + " ,");
}
while (maxWaitTimeInSeconds > 0)
{
  var req = new GetObjectMetadataRequest(bucketName, objectName)
  {
    // 指定归档类型Object的版本id。
    VersionId = versionid
  };
  var meta = client.GetObjectMetadata(req);
  string restoreStatus = meta.HttpMetadata["x-oss-restore"] as string;
```

```
    if (restoreStatus != null && restoreStatus.StartsWith("ongoing-request=\"false\"", StringComparison.Inva
riantCultureIgnoreCase))
    {
        break;
    }
    Thread.Sleep(1000);
    maxWaitTimeInSeconds--;
}
if (maxWaitTimeInSeconds == 0)
{
    Console.WriteLine("RestoreObject is timeout. ");
    throw new TimeoutException();
}
else
{
    Console.WriteLine("RestoreObject is successful. ");
}
```

解冻文件的详细信息请参见RestoreObject。

7.9.7. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的meta信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的meta信息。

以下代码用于获取文件元信息：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 填写不包含Bucket名称在内的Object的完整路径，例如example/test.jpg。
var objectName = "<yourObjectName>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取文件元信息。
  var request = new GetObjectMetadataRequest(bucketName, objectName)
  {
    // 获取指定版本文件的元信息。
    VersionId = versionid
  };
  // 获取指定版本文件的全部元信息。
  var oldMeta = client.GetObjectMetadata(request);
  // 获取指定版本文件的部分元信息。
  var meta = client.GetSimplifiedObjectMetadata(request);
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

关于如何获取文件元信息的详情，请参见GetObjectMeta和HeadObject。

7.9.8. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 设置文件访问权限。
try
{
  // 通过SetObjectAcl设置文件访问权限。
  var request = new SetObjectAclRequest(bucketName, objectName, CannedAccessControlList.PublicRead)
  {
    // 设置指定版本文件的访问权限。
    VersionId = versionid
  };
  client.SetObjectAcl(request);
  Console.WriteLine("Set Object:{0} ACL succeeded ", objectName);
}
catch (Exception ex)
{
  Console.WriteLine("Set Object ACL failed with error info: {0}", ex.Message);
}
}
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（DeleteMarker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 获取文件访问权限。
try
{
  // 通过GetObjectAcl获取文件访问权限。
  var request = new GetObjectAclRequest(bucketName, objectName)
  {
    // 获取指定版本文件的访问权限。
    VersionId = versionid
  };
  var result = client.GetObjectAcl(request);
  Console.WriteLine("Get Object ACL succeeded, Id: {0}  ACL: {1}",
    result.Owner.Id, result.ACL.ToString());
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2}\tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取文件访问权限的详细信息请参见GetObjectACL。

7.9.9. 管理软链接

软链接功能用于快速访问对象存储空间内的常用文件。设置软链接后，您可以通过软链接文件快速打开源文件，类似于Windows的快捷方式。本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

软链接本身也可以有多个版本，每个不同的版本可以指向不同的TargetObject，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。您可以通过创建软链接指向TargetObject的当前版本。

> **说明** 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

以下代码用于创建软链接：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var targetObjectName = "<yourTargetObjectName>";
var symlinkObjectName = "<yourSymlinkObjectName>";
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 上传TargetObject。
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  client.PutObject(bucketName, targetObjectName, requestContent);
  // 创建软链接。
  client.CreateSymlink(bucketName, symlinkObjectName, targetObjectName);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var symlinkObjectName = "<yourSymlinkObjectName>";
var versionid = "<yourArchiveObjectVersionid>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取软链接指向的目标文件名称。
  var request = new GetSymlinkRequest(bucketName, symlinkObjectName)
  {
    // 获取指定版本的软链接的内容。
    VersionId = versionid
  };
  var ossSymlink = client.GetSymlink(request);
  Console.WriteLine("Target object is {0}", ossSymlink.Target);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取软链接的详细信息请参见GetSymlink。

7.10. 对象标签

7.10.1. 设置对象标签

OSS支持使用对象标签（Object Tagging）对存储空间（Bucket）中的文件（Object）进行分类，您可以针对相同标签的Object设置生命周期规则、访问权限等。

背景信息

设置对象标签时，请注意以下事项：

- 您可以在上传Object时设置对象标签，也可以对已上传Object设置对象标签。设置对象标签时，如果对象已有标签，则覆盖原标签。关于设置对象标签的更多信息，请参见PutObjectTagging。
- 设置对象标签时，您要有PutObjectTagging权限。
- 更改标签时不会更新Object的Last-Modified时间。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/

  > **说明** 通过HTTP header的方式设置标签且标签中包含任意字符时，您可以对标签的Key和Value做URL编码。

对象标签使用一组键值对（Key-Value）来标记对象。关于对象标签的更多信息，请参见对象标签。

上传Object时添加对象标签

以下分别介绍简单上传、分片上传、追加上传以及断点续传上传场景下为上传的Object添加对象标签的示例。

- 简单上传时添加对象标签

  以下代码用于简单上传时添加对象标签。

```
using System.Text;
using Aliyun.OSS;
using System.Text;
using Aliyun.OSS.Util;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  var meta = new ObjectMetadata();
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + Url
EncodeKey("key2");
  meta.AddHeader("x-oss-tagging", str);
  var putRequest = new PutObjectRequest(bucketName, objectName, requestContent);
  putRequest.Metadata = meta;
  // 上传文件的同时对文件设置标签。
  client.PutObject(putRequest);
```

```
  Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

- 分片上传时添加对象标签

  以下代码用于分片上传时添加对象标签。

```
using Aliyun.OSS;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 初始化分片上传。
var uploadId = "";
try
{
  // 设置上传Object及所属存储空间的名称。您可以在InitiateMultipartUploadRequest中设置ObjectMeta，但不必指定ContentLength。
  var request = new InitiateMultipartUploadRequest(bucketName, objectName);
  var meta = new ObjectMetadata();
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + UrlEncodeKey("key2");
```

```
    meta.AddHeader("x-oss-tagging", str);
    request.Metadata = meta;
    var result = client.InitiateMultipartUpload(request);
    uploadId = result.UploadId;
    // 打印UploadId。
    Console.WriteLine("Init multi part upload succeeded");
    Console.WriteLine("Upload Id:{0}", result.UploadId);
}
catch (Exception ex)
{
    Console.WriteLine("Init multi part upload failed, {0}", ex.Message);
}
// 计算分片总数。
var partSize = 100 * 1024;
var fi = new FileInfo(localFilename);
var fileSize = fi.Length;
var partCount = fileSize / partSize;
if (fileSize % partSize != 0)
{
    partCount++;
}
// 开始分片上传。partETags是保存partETag的列表，OSS收到您提交的分片列表后，会逐一验证每个分片数据的有效性。当所有的数据分片通过验证后，OSS会将这些分片组合成一个完整的文件。
var partETags = new List<PartETag>();
try
{
    using (var fs = File.Open(localFilename, FileMode.Open))
    {
        for (var i = 0; i < partCount; i++)
        {
            var skipBytes = (long)partSize * i;
            // 定位到本次上传的起始位置。
            fs.Seek(skipBytes, 0);
            // 计算本次上传的分片大小，最后一个分片为剩余的数据大小。
            var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
            var request = new UploadPartRequest(bucketName, objectName, uploadId)
            {
                InputStream = fs,
                PartSize = size,
                PartNumber = i + 1
            };
            // 调用UploadPart接口上传分片，返回结果中包含了分片的ETag值。
            var result = client.UploadPart(request);
            partETags.Add(result.PartETag);
            Console.WriteLine("finish {0}/{1}", partETags.Count, partCount);
        }
        Console.WriteLine("Put multi part upload succeeded");
    }
}
catch (Exception ex)
{
    Console.WriteLine("Put multi part upload failed, {0}", ex.Message);
}
// 完成分片上传。
try
```

```
try
{
  var completeMultipartUploadRequest = new CompleteMultipartUploadRequest(bucketName, objectNa
me, uploadId);
  foreach (var partETag in partETags)
  {
    completeMultipartUploadRequest.PartETags.Add(partETag);
  }
  var result = client.CompleteMultipartUpload(completeMultipartUploadRequest);
  Console.WriteLine("complete multi part succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("complete multi part failed, {0}", ex.Message);
}
```

- 追加上传时添加对象标签

  以下代码用于追加上传时添加对象标签。

```
using Aliyun.OSS;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
// 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
long position = 0;
try
```

```
{
  var metadata = client.GetObjectMetadata(bucketName, objectName);
  position = metadata.ContentLength;
}
catch (Exception) { }
try
{
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    var meta = new ObjectMetadata();
    // 在HTTP header中设置标签信息。
     string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + U
rlEncodeKey("key2");
    meta.AddHeader("x-oss-tagging", str);
    request.Metadata = meta;
    // 第一次追加。只有第一次追加上传时设置的标签生效。
    var result = client.AppendObject(request);
    // 设置文件的追加位置。
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}", position);
  }
  // 获取追加位置，再次追加文件。
  using (var fs = File.Open(localFilename, FileMode.Open))
  {
    var request = new AppendObjectRequest(bucketName, objectName)
    {
      ObjectMetadata = new ObjectMetadata(),
      Content = fs,
      Position = position
    };
    var result = client.AppendObject(request);
    position = result.NextAppendPosition;
    Console.WriteLine("Append object succeeded, next append position:{0}", position);
  }
}
catch (Exception ex)
{
  Console.WriteLine("Append object failed, {0}", ex.Message);
}
```

- 断点续传上传时添加对象标签

  以下代码用于断点续传上传时添加对象标签。

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
```

```
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var localFilename = "<yourLocalFilename>";
string checkpointDir = "<yourCheckpointDir>";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 通过UploadFileRequest设置多个参数。
  UploadObjectRequest request = new UploadObjectRequest(bucketName, objectName, localFilename)
  {
    // 指定上传的分片大小。
    PartSize = 8 * 1024 * 1024,
    // 指定并发线程数。
    ParallelThreadCount = 3,
    // checkpointDir用于记录本地分片上传结果的文件。上传过程中的进度信息会保存在此文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。如果checkpointDir设置为null，断点续传功能不会生效，每次上传失败后都会重新上传。
    CheckpointDir = checkpointDir,
  };
  var meta = new ObjectMetadata();
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + UrlEncodeKey("key2");
  meta.AddHeader("x-oss-tagging", str);
  request.Metadata = meta;
  // 断点续传上传。
  client.ResumableUploadObject(request);
  Console.WriteLine("Resumable upload object:{0} succeeded", objectName);
}
```

```
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

对已上传Object添加或更改对象标签

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 设置标签信息。
  var setRequest = new SetObjectTaggingRequest(bucketName, objectName);
  var tag1 = new Tag
  {
    Key = "project",
    Value = "projectone"
  };
  var tag2 = new Tag
  {
    Key = "user",
    Value = "jsmith"
  };
  setRequest.AddTag(tag1);
  setRequest.AddTag(tag2);
  client.SetObjectTagging(setRequest);
  Console.WriteLine("set object tagging succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("set object tagging failed. {0}", ex.Message);
}
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 以下代码用于简单拷贝1 GB以下的Object时设置对象标签：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var metadata = new ObjectMetadata();
  metadata.AddHeader("mk1", "mv1");
  metadata.AddHeader("mk2", "mv2");
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + Url
EncodeKey("key2");
  metadata.AddHeader("x-oss-tagging", str);
  var req = new CopyObjectRequest(sourceBucket, sourceObject, targetBucket, targetObject)
  {
    // 如果NewObjectMetadata为null，则表示Copy模式（即拷贝源文件的元信息）；非null则表示Replace模式
（即覆盖源文件的元信息）。
    NewObjectMetadata = metadata
  };
  // 拷贝文件。
  client.CopyObject(req);
  Console.WriteLine("Copy object succeeded");
}
```

```
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

- 以下代码用于分片拷贝1 GB以上的Object时设置对象标签：

```
using Aliyun.OSS;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var sourceBucket = "<yourSourceBucketName>";
var sourceObject = "<yourSourceObjectName>";
var targetBucket = "<yourDestBucketName>";
var targetObject = "<yourDestObjectName>";
var uploadId = "";
var partSize = 50 * 1024 * 1024;
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 初始化拷贝任务。您可以通过InitiateMultipartUploadRequest指定目标文件元信息。
  var request = new InitiateMultipartUploadRequest(targetBucket, targetObject);
  var meta = new ObjectMetadata();
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + Url
```

```
EncodeKey("key2");
  meta.AddHeader("x-oss-tagging", str);
  request.Metadata = meta;
  var result = client.InitiateMultipartUpload(request);
  // 打印uploadId。
  uploadId = result.UploadId;
  Console.WriteLine("Init multipart upload succeeded， Upload Id: {0}", result.UploadId);
  // 计算分片数。
  var metadata = client.GetObjectMetadata(sourceBucket, sourceObject);
  var fileSize = metadata.ContentLength;
  var partCount = (int)fileSize / partSize;
  if (fileSize % partSize != 0)
  {
    partCount++;
  }
  // 开始分片拷贝。
  var partETags = new List<PartETag>();
  for (var i = 0; i < partCount; i++)
  {
    var skipBytes = (long)partSize * i;
    var size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    // 创建UploadPartCopyRequest并通过UploadPartCopyRequest指定限定条件。
    var uploadPartCopyRequest = new UploadPartCopyRequest(targetBucket, targetObject, sourceBucke
t, sourceObject, uploadId)
      {
        PartSize = size,
        PartNumber = i + 1,
        // BeginIndex用来定位此次上传分片开始所对应的位置。
        BeginIndex = skipBytes
      };
    // 调用uploadPartCopy方法来拷贝每一个分片。
    var uploadPartCopyResult = client.UploadPartCopy(uploadPartCopyRequest);
    Console.WriteLine("UploadPartCopy : {0}", i);
    partETags.Add(uploadPartCopyResult.PartETag);
  }
  // 完成分片拷贝。
  var completeMultipartUploadRequest =
  new CompleteMultipartUploadRequest(targetBucket, targetObject, uploadId);
  // partETags为分片上传中保存的partETag的列表，OSS收到用户提交的此列表后，会逐一验证每个数据分片的有
效性。全部验证通过后，OSS会将这些分片合成一个完整的文件。
  foreach (var partETag in partETags)
  {
    completeMultipartUploadRequest.PartETags.Add(partETag);
  }
  var completeMultipartUploadResult = client.CompleteMultipartUpload(completeMultipartUploadReque
st);
  Console.WriteLine("CompleteMultipartUpload succeeded");
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID: {2} \tHostID: {3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
```

```
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

为软链接文件设置标签

以下代码用于为软链接文件设置对象标签。

```
using Aliyun.OSS;
using System.Text;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var targetObjectName = "<yourTargetObjectName>";
var symlinkObjectName = "<yourSymlinkObjectName>";
var objectContent = "More than just cloud.";
String UrlEncodeKey(String key)
{
const string CharsetName = "utf-8";
const char separator = '/';
var segments = key.Split(separator);
var encodedKey = new StringBuilder();
encodedKey.Append(HttpUtils.EncodeUri(segments[0], CharsetName));
for (var i = 1; i < segments.Length; i++)
  encodedKey.Append(separator).Append(HttpUtils.EncodeUri(segments[i], CharsetName));
  if (key.EndsWith(separator.ToString()))
  {
    // String#split ignores trailing empty strings, e.g., "a/b/" will be split as a 2-entries array,
    // so we have to append all the trailing slash to the uri.
    foreach (var ch in key)
    {
      if (ch == separator)
        encodedKey.Append(separator);
      else
        break;
    }
  }
return encodedKey.ToString();
}
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 上传目标文件。
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  client.PutObject(bucketName, targetObjectName, requestContent);
  var meta = new ObjectMetadata();
  // 在HTTP header中设置标签信息。
  string str = UrlEncodeKey("key1") + "=" + UrlEncodeKey("key1") + "&" + UrlEncodeKey("key2") + "=" + UrlEn
codeKey("key2");
  meta.AddHeader("x-oss-tagging", str);
  var request = new CreateSymlinkRequest(bucketName, symlinkObjectName, targetObjectName);
  request.ObjectMetadata = meta;
```

```
    request.ObjectMetadata = meta;
    // 创建软链接。
    client.CreateSymlink(request);
    // 获取软链接指向的目标文件名称。
    var ossSymlink = client.GetSymlink(bucketName, symlinkObjectName);
    Console.WriteLine("Target object is {0}", ossSymlink.Target);
}
catch (Exception ex)
{
    Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.10.2. 获取对象标签

对象标签使用一组键值对（Key-Value）来标记对象。本文介绍如何获取对象标签。

以下代码用于获取对象标签：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
    // 获取对象标签信息。
    var result = client.GetObjectTagging(bucketName,objectName);
    Console.WriteLine("get objects tagging succeeded");
    foreach (var tag in result.Tags)
    {
        Console.WriteLine("key:{0}, value:{1}", tag.Key, tag.Value);
    }
}
catch (Exception ex)
{
    Console.WriteLine("get objects tagging failed. {0}", ex.Message);
}
```

有关获取对象标签的详情请参见GetObjectTagging。有关对象标签的详情请参见对象标签。

7.10.3. 删除对象标签

对象标签使用一组键值对（Key-Value）来标记对象。本文介绍如何删除对象标签。

以下代码用于删除对象标签：

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除对象标签。
  var result = client.DeleteObjectTagging(bucketName,objectName);
  Console.WriteLine("delete objects tagging succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("delete objects failed. {0}", ex.Message);
}
```

有关删除对象标签的详情请参见DeleteObjectTagging。有关对象标签的详情请参见对象标签。

7.10.4. 对象标签和生命周期管理

生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为生命周期规则生效的条件。

> **说明** 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于生命周期规则中添加标签匹配规则：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var setBucketLifecycleRequest = new SetBucketLifecycleRequest(bucketName);
  // 创建第一条生命周期规则。
  LifecycleRule lcr1 = new LifecycleRule()
  {
    ID = "delete obsoleted files",
    Prefix = "obsoleted/",
    Status = RuleStatus.Enabled,
    ExpriationDays = 3,
    Tags = new Tag[1]
  };
```

```
    // 设置标签信息。
    var tag1 = new Tag
    {
      Key = "project",
      Value = "projectone"
    };
    lcr1.Tags[0] = tag1;
    // 创建第二条生命周期规则。
    LifecycleRule lcr2 = new LifecycleRule()
    {
      ID = "delete temporary files",
      Prefix = "temporary/",
      Status = RuleStatus.Enabled,
      ExpriationDays = 20,
      Tags = new Tag[1]
    };
    // 设置标签信息。
    var tag2 = new Tag
    {
      Key = "user",
      Value = "jsmith"
    };
    lcr2.Tags[0] = tag2;
    setBucketLifecycleRequest.AddLifecycleRule(lcr1);
    setBucketLifecycleRequest.AddLifecycleRule(lcr2);
    // 配置生命周期规则。
    client.SetBucketLifecycle(setBucketLifecycleRequest);
    Console.WriteLine("Set bucket:{0} Lifecycle succeeded ", bucketName);
  }
  catch (OssException ex)
  {
    Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
      ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
  }
  catch (Exception ex)
  {
    Console.WriteLine("Failed with error info: {0}", ex.Message);
  }
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 查看生命周期规则。
  var rules = client.GetBucketLifecycle(bucketName);
  Console.WriteLine("Get bucket:{0} Lifecycle succeeded ", bucketName);
  foreach (var rule in rules)
  {
    Console.WriteLine("ID: {0}", rule.ID);
    Console.WriteLine("Prefix: {0}", rule.Prefix);
    Console.WriteLine("Status: {0}", rule.Status);
    // 查看标签信息。
    foreach (var tag in rule.Tags)
    {
      Console.WriteLine("key:{0}, value:{1}", tag.Key, tag.Value);
    }
    if (rule.ExpriationDays.HasValue)
      Console.WriteLine("ExpirationDays: {0}", rule.ExpriationDays);
  }
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.11. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

> **说明** 有关单链接限速的使用场景及注意事项的更多信息，请参考开发指南的单链接限速文档。

简单上传和下载限速

以下代码用于简单上传（PutObject）和下载文件（GetObject）时设置单链接限速：

```
using System.Text;
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  byte[] binaryData = Encoding.ASCII.GetBytes(objectContent);
  MemoryStream requestContent = new MemoryStream(binaryData);
  // 上传文件限速100KB/s，即819200bit/s。
  var putRequest = new PutObjectRequest(bucketName, objectName, requestContent)
  {
    TrafficLimit = 100*1024*8
  };
  client.PutObject(putRequest);
  Console.WriteLine("Put object succeeded");
  // 下载文件限速100KB/s，即819200bit/s。
  var getRequest = new GetObjectRequest(bucketName, objectName)
  {
    TrafficLimit = 100 * 1024 * 8
  };
  var getResult = client.GetObject(getRequest);
  Console.WriteLine("Get object succeeded");
}
catch (Exception ex)
{
  Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

使用签名URL方式上传和下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
using System.Text;
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 生成上传签名URL。
  var generatePresignedUriRequest = new GeneratePresignedUriRequest(bucketName, objectName, SignHtt
pMethod.Put)
  {
```

```
    {
      Expiration = DateTime.Now.AddHours(1),
    };
    // 限速100KB/s，即819200bit/s。
    generatePresignedUriRequest.AddQueryParam("x-oss-traffic-limit", "819200");
    var signedUrl = client.GeneratePresignedUri(generatePresignedUriRequest);
    // 使用签名URL上传文件。
    var buffer = Encoding.UTF8.GetBytes(objectContent);
    using (var ms = new MemoryStream(buffer))
    {
      client.PutObject(signedUrl, ms);
    }
    Console.WriteLine("Put object by signatrue succeeded. {0} ", signedUrl.ToString());
    var metadata = client.GetObjectMetadata(bucketName, objectName);
    var etag = metadata.ETag;
    // 生成下载签名URL。
    // 限速100KB/s，即819200bit/s。
    var req = new GeneratePresignedUriRequest(bucketName, objectName, SignHttpMethod.Get);
    req.AddQueryParam("x-oss-traffic-limit", "819200");
    var uri = client.GeneratePresignedUri(req);
    // 使用签名URL下载文件。
    OssObject ossObject = client.GetObject(uri);
    using (var file = File.Open(downloadFilename, FileMode.OpenOrCreate))
    {
      using (Stream stream = ossObject.Content)
      {
        int length = 4 * 1024;
        var buf = new byte[length];
        do
        {
          length = stream.Read(buf, 0, length);
          file.Write(buf, 0, length);
        } while (length != 0);
      }
    }
    Console.WriteLine("Get object by signatrue succeeded. {0} ", uri.ToString());
  }
  catch (Exception ex)
  {
    Console.WriteLine("Put object failed, {0}", ex.Message);
  }
```

7.12. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

> **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

> **注意**
> - 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
> - 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
> - 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  //配置Bucket加密。
  var request = new SetBucketEncryptionRequest(bucketName, "KMS", null);
  client.SetBucketEncryption(request);
  Console.WriteLine("Set bucket:{0} Encryption succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 获取Bucket加密配置。
  var result = client.GetBucketEncryption(bucketName);
  Console.WriteLine("Get bucket:{0} Encryption succeeded ", bucketName);
  Console.WriteLine("SSEAlgorithm: {0}", rules.SSEAlgorithm);
  Console.WriteLine("KMSMasterKeyID: {0}", rules.KMSMasterKeyID);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 删除Bucket加密配置。
  client.DeleteBucketEncryption(bucketName);
  Console.WriteLine("Delete bucket:{0} Encryption succeeded ", bucketName);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.13. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

以下代码用于使用STS凭证构造签名请求。

```
using Aliyun.OSS;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var securityToken = "<yourSecurityToken>";
// 获取STS临时凭证后，您可以通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和AccessKey
Secret）生成OSSClient。
// 创建OSSClient实例。
var ossStsClient = new OssClient(endpoint, accessKeyId, accessKeySecret, securityToken);
// 执行OSS相关操作。
```

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

使用签名URL临时授权

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

使用签名URL进行临时授权的完整代码请参见Git Hub。

以下介绍使用签名URL临时授权的常见示例。

- 使用签名URL上传文件

  以下代码用于使用签名URL上传文件：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var objectContent = "More than just cloud.";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 生成签名URL。
  var generatePresignedUriRequest = new GeneratePresignedUriRequest(bucketName, objectName, Sign
HttpMethod.Put)
  {
    Expiration = DateTime.Now.AddHours(1),
  };
  var signedUrl = client.GeneratePresignedUri(generatePresignedUriRequest);
  // 使用签名URL上传文件。
  var buffer = Encoding.UTF8.GetBytes(objectContent);
  using (var ms = new MemoryStream(buffer))
  {
    client.PutObject(signedUrl, ms);
  }
  Console.WriteLine("Put object by signatrue succeeded. {0} ", signedUrl.ToString());
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

- 使用签名URL下载文件

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
var downloadFilename = "<yourDownloadFilename>";
// 创建OSSClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  var metadata = client.GetObjectMetadata(bucketName, objectName);
  var etag = metadata.ETag;
  // 生成签名URL。
  var req = new GeneratePresignedUriRequest(bucketName, objectName, SignHttpMethod.Get);
  var uri = client.GeneratePresignedUri(req);
  // 使用签名URL下载文件。
  OssObject ossObject = client.GetObject(uri);
  using (var file = File.Open(downloadFilename, FileMode.OpenOrCreate))
  {
    using (Stream stream = ossObject.Content)
    {
      int length = 4 * 1024;
      var buf = new byte[length];
      do
      {
        length = stream.Read(buf, 0, length);
        file.Write(buf, 0, length);
      } while (length != 0);
    }
  }
  Console.WriteLine("Get object by signatrue succeeded. {0} ", uri.ToString());
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

7.14. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
using System;
```

```
using System;
using System.IO;
using Aliyun.OSS;
using Aliyun.OSS.Common;
using Aliyun.OSS.Util;
namespace ImageProcess
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.ImageProcess();
      Console.ReadKey();
    }
    public static void ImageProcess()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      // 本地文件名称，上传本地图片文件时使用。
      // var localImageFilename = "<yourLocalImageFilename>";
      // 指定保存处理后的图片的本地文件夹。
      var downloadDir = "<yourDownloadDir>";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      try
      {
        // 若目标图片不在目标Bucket内，需上传图片到目标Bucket。
        // client.PutObject(bucketName, objectName, localImageFilename);
        // 将图片缩放为固定宽高100 px。
        var process = "image/resize,m_fixed,w_100,h_100";
        var ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
        // 指定处理后的图片名称。
        WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
      }
      catch (OssException ex)
      {
        Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
          ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
      }
      catch (Exception ex)
      {
        Console.WriteLine("Failed with error info: {0}", ex.Message);
      }
    }
    private static void WriteToFile(string filePath, Stream stream)
    {
      using (var requestStream = stream)
      {
        using (var fs = File.Open(filePath, FileMode.OpenOrCreate))
        {
          IoUtils.WriteTo(stream, fs);
        }
```

```
        }
      }
    }
  }
```

- 使用不同的图片处理参数处理图片并分别保存为本地图片

```
using System;
using System.IO;
using Aliyun.OSS;
using Aliyun.OSS.Common;
using Aliyun.OSS.Util;
namespace ImageProcess
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.ImageProcess();
      Console.ReadKey();
    }
    public static void ImageProcess()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      // 本地文件名称，上传本地图片文件时使用。
      // var localImageFilename = "<yourLocalImageFilename>";
      // 指定保存处理后的图片的本地文件夹。
      var downloadDir = "<yourDownloadDir>";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      try
      {
        // 若目标图片不在目标Bucket内，需上传图片到目标Bucket。
        // client.PutObject(bucketName, objectName, localImageFilename);
        // 将图片缩放为固定宽高100 px。
        var process = "image/resize,m_fixed,w_100,h_100";
        var ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
        // 指定处理后的图片名称。
        WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
        // 从坐标（100,100）开始，将图片裁剪为宽高100 px。
        process = "image/crop,w_100,h_100,x_100,y_100";
        ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
        WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
        // 将图片旋转90°。
        process = "image/rotate,90";
        ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
        WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
      }
      catch (OssException ex)
      {
        Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
```

```
            Console.WriteLine("Failed with error code: {0}; Error info: {1}. {RequestId:{2} {HostId:{3}}",
              ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
        }
        catch (Exception ex)
        {
          Console.WriteLine("Failed with error info: {0}", ex.Message);
        }
      }
      private static void WriteToFile(string filePath, Stream stream)
      {
        using (var requestStream = stream)
        {
          using (var fs = File.Open(filePath, FileMode.OpenOrCreate))
          {
            IoUtils.WriteTo(stream, fs);
          }
        }
      }
    }
  }
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
using System;
using System.IO;
using Aliyun.OSS;
using Aliyun.OSS.Common;
using Aliyun.OSS.Util;
namespace ImageProcessCascade
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.ImageProcessCascade();
      Console.ReadKey();
    }
    public static void ImageProcessCascade()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
      var bucketName = "<yourBucketName>";
      var objectName = "<yourObjectName>";
      // 本地文件名称，上传本地图片文件时使用。
      // var localImageFilename = "<yourLocalImageFilename>";
      // 指定保存处理后的图片的本地文件夹。
      var downloadDir = "<yourDownloadDir>";
      // 创建OssClient实例。
      var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
      try
      {
        // 若目标图片不在目标Bucket内，需上传图片到目标Bucket。
        // client.PutObject(bucketName, objectName, localImageFilename);
```

```
            // 将图片缩放为固定宽高100 px后，再旋转90°。
            var process = "image/resize,m_fixed,w_100,h_100/rotate,90";
            var ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
            // 指定处理后的图片名称。
            WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
            Console.WriteLine("Get Object:{0} with process:{1} succeeded ", objectName, process);
          }
          catch (OssException ex)
          {
            Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
              ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
          }
          catch (Exception ex)
          {
            Console.WriteLine("Failed with error info: {0}", ex.Message);
          }
        }
        private static void WriteToFile(string filePath, Stream stream)
        {
          using (var requestStream = stream)
          {
            using (var fs = File.Open(filePath, FileMode.OpenOrCreate))
            {
              IoUtils.WriteTo(stream, fs);
            }
          }
        }
      }
    }
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
using System;
using System.IO;
using Aliyun.OSS;
using Aliyun.OSS.Common;
using Aliyun.OSS.Util;
namespace ImageProcessCustom
{
  class Program
  {
    static void Main(string[] args)
    {
      Program.ImageProcessCustomStyle();
      Console.ReadKey();
    }
    public static void ImageProcessCustomStyle()
    {
      var endpoint = "<yourEndpoint>";
      var accessKeyId = "<yourAccessKeyId>";
      var accessKeySecret = "<yourAccessKeySecret>";
```

```
        var bucketName = "<yourBucketName>";
        var objectName = "<yourObjectName>";
        // 本地文件名称，上传本地图片文件时使用。
        // var localImageFilename = "<yourLocalImageFilename>";
        // 指定保存处理后的图片的本地文件夹。
        var downloadDir = "<yourDownloadDir>";
        // 创建OssClient实例。
        var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
        try
        {
          // 若目标图片不在目标Bucket内，需上传图片到目标Bucket。
          // client.PutObject(bucketName, objectName, localImageFilename);
          // 使用指定图片样式处理图片。
          var process = "style/<yourCustomStyleName>";
          var ossObject = client.GetObject(new GetObjectRequest(bucketName, objectName, process));
          // 指定处理后的图片名称。
          WriteToFile(downloadDir + "/<LocalFilename>", ossObject.Content);
          Console.WriteLine("Get Object:{0} with process:{1} succeeded ", objectName, process);
        }
        catch (OssException ex)
        {
          Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
            ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
        }
        catch (Exception ex)
        {
          Console.WriteLine("Failed with error info: {0}", ex.Message);
        }
      }
      private static void WriteToFile(string filePath, Stream stream)
      {
        using (var requestStream = stream)
        {
          using (var fs = File.Open(filePath, FileMode.OpenOrCreate))
          {
            IoUtils.WriteTo(stream, fs);
          }
        }
      }
    }
  }
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;
var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";
// 创建OssClient实例。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);
try
{
  // 将图片缩放为固定宽高100 px。
  var process = "image/resize,m_fixed,w_100,h_100";
  var req = new GeneratePresignedUriRequest(bucketName, objectName, SignHttpMethod.Get)
  {
    Expiration = DateTime.Now.AddHours(1),
    Process = process
  };
  // 生成带有签名的URI。
  var uri = client.GeneratePresignedUri(req);
  Console.WriteLine("Generate Presigned Uri:{0} with process:{1} succeeded ", uri, process);
}
catch (OssException ex)
{
  Console.WriteLine("Failed with error code: {0}; Error info: {1}. \nRequestID:{2}\tHostID:{3}",
    ex.ErrorCode, ex.Message, ex.RequestId, ex.HostId);
}
catch (Exception ex)
{
  Console.WriteLine("Failed with error info: {0}", ex.Message);
}
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

7.15. 异常处理

OSS .NET SDK 包含两类异常，一类是客户端异常ClientException，另一类是服务器端异常OSSException，它们均继承自RuntimeException。

ClientException

ClientException指客户端尝试向OSS发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用，则会抛出ClientException。当上传文件时发生IO异常，也会抛出ClientException。

OSSException

OSSException指服务器端异常，它来自于对服务器错误信息的解析。OSSException包含OSS返回的错误码和错误信息，便于定位问题，并做出适当的处理。

OSSException通常包含以下错误信息：

参数	描述
Code	OSS返回的错误码。
Message	OSS返回的详细错误信息。
RequestId	用于唯一标识该请求的UUID。当您无法解决问题时，可以提供RequestId来请求OSS开发工程师的帮助。
HostId	用于标识访问的OSS集群，与请求时使用的Host一致。

OSS常见错误码

错误码	描述
AccessDenied	拒绝访问
BucketAlreadyExists	存储空间已经存在
BucketNotEmpty	存储空间非空
EntityTooLarge	实体过大
EntityTooSmall	实体过小
FileGroupTooLarge	文件组过大
FilePartNotExist	文件分片不存在
FilePartStale	文件分片过时
InvalidArgument	参数格式错误
InvalidAccessKeyId	AccessKeyId不存在
InvalidBucketName	无效的存储空间名称
InvalidDigest	无效的摘要
InvalidObjectName	无效的文件名称
InvalidPart	无效的分片
InvalidPartOrder	无效的分片顺序
InvalidTargetBucketForLogging	Logging操作中有无效的目标存储空间
InternalError	OSS内部错误
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法

错误码	描述
MissingArgument	缺少参数
MissingContentLength	缺少内容长度
NoSuchBucket	存储空间不存在
NoSuchKey	文件不存在
NoSuchUpload	分片上传ID不存在
NotImplemented	无法处理的方法
PreconditionFailed	预处理错误
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟
RequestTimeout	请求超时
SignatureDoesNotMatch	签名错误
TooManyBuckets	用户的存储空间数目超过限制

8.Android

8.1. 前言

本文档基于Android SDK 2.9.5编写。

兼容性

Android SDK具有向下兼容的特性，新版本完全兼容老版本。

SDK源码

SDK源码请参见GitHub。

示例代码

OSS Android SDK提供丰富的示例代码，方便您参考或直接使用。

示例代码包括以下内容：

示例文件	示例内容
BaseTestCase.java	初始化
OSSPutObjectTest.java	上传文件
OSSGetObjectTest.java	下载文件
ManageObjectTest.java	管理文件
OSSBucketTest.java	存储空间
ResumableUploadTest.java	断点续传
CRC64Test.java	数据安全性
ImagePersistTest.java	图片处理

8.2. 安装

本文介绍如何安装Android SDK及相关配置说明。

> **说明** 使用Android SDK时，建议指定Android SDK的版本号。

前提条件

- Android系统为2.3及以上版本。
- 已开通OSS服务。

下载SDK

- 通过添加依赖项下载

在Android Studio环境中添加如下依赖：

```
dependencies { compile 'com.aliyun.dpa:oss-android-sdk:2.9.5' }
```

- SDK安装包
- 通过GitHub下载

安装SDK

您可以通过如下两种方式安装SDK：

- 方式一：在Maven项目中加入依赖项（推荐方式）

```
dependencies {
  compile 'com.aliyun.dpa:oss-android-sdk:2.9.5'}
```

- 方式二：源码编译jar包
  i. clone工程源码，运行gradle命令生成jar包。

```
# clone工程源码。
$ git clone https://github.com/aliyun/aliyun-oss-android-sdk.git
# 进入目录。
$ cd aliyun-oss-android-sdk/oss-android-sdk/
# 执行打包脚本，要求jdk为1.7及以上版本。
$ ../gradlew releaseJar
# 进入打包生成目录，jar包将在此目录下生成。
$ cd build/libs && ls
```

  ii. 直接引入编译好的jar包。

  将aliyun-oss-sdk-android-2.9.5.jar、okhttp-3.x.x.jar、okio-1.x.x.jar 3个jar包导入libs目录。

相关配置

以下介绍使用OSS Android SDK前的相关配置。

- 权限配置

  OSS Android SDK所需的Android权限如下：

> **注意** 请确保已在AndroidManifest.xml文件中配置所需权限，否则SDK将无法正常工作。

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
```

- 混淆配置

  请在混淆配置中加入以下内容：

```
-keep class com.alibaba.sdk.android.oss.** { *; }
-dontwarn okio.**
-dontwarn org.apache.commons.codec.binary.**
```

8.3. 初始化

OSSClient 是 OSS 服务的 Android 客户端，它为调用者提供了一系列的方法进行操作、管理存储空间（Bucket）和文件（Object）等。在使用 SDK 发起对 OSS 的请求前，您需要初始化一个 OSSClient 实例，并对它进行一些必要设置。

> **说明** 生命周期和应用生命周期保持一致即可。在应用启动时创建一个全局的OSSClient，在应用结束时销毁即可。

确定 Endpoint

Endpoint 是阿里云 OSS 服务在各个区域的地址，目前支持以下两种形式：

Endpoint类型	解释
OSS区域地址	使用OSS Bucket所在区域地址，各个区域Endpoint参考访问域名和数据中心
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

- OSS区域地址

  使用OSS Bucket所在区域地址，Endpoint查询可以有下面两种方式：

  - 查询Endpoint与区域对应关系详情请参见：访问域名和数据中心。
  - 您可以登录 阿里云OSS控制台，进入Bucket概览页，Bucket域名的后缀部分：如 `bucket-1.oss-cn-hangzhou.aliyuncs.com` 的 `oss-cn-hangzhou.aliyuncs.com` 部分为该Bucket的外网Endpoint。

- Cname

  您可以将自己拥有的域名通过Cname绑定到某个存储空间上，然后通过自身域名访问存储空间内的文件。

  比如您要将域名new-image.xxxxx.com绑定到深圳区域的名称为image的存储空间上：您需要到您域名xxxxx.com托管商那里设定一个新的域名解析，将 `http://new-image.xxxxx.com` 解析到 `http://image.oss-cn-shenzhen.aliyuncs.com` ，类型为CNAME。

设置EndPoint和凭证

移动终端是一个不受信任的环境，把 `AccessKeyId` 和 `AccessKeySecret` 直接保存在终端用来加签请求，存在极高的风险。推荐使用STS鉴权模式或自签名模式，详情请参见：授权访问、快速搭建移动应用直传服务。

> **说明** 如果使用STS鉴权模式，推荐使用OSSAuthCredentialsProvider方式直接访问鉴权应用服务器，token过期后可以自动更新。

关于示例工程更多信息请参见GitHub。

设置EndPoint和CredentialProvider示例如下：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
String stsServer = "STS应用服务器地址，例如http://abc.com"
// 推荐使用OSSAuthCredentialsProvider。token过期可以及时更新。
OSSCredentialProvider credentialProvider = new OSSAuthCredentialsProvider(stsServer);
// 配置类如果不设置，会有默认配置。
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒。
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒。
conf.setMaxConcurrentRequest(5); // 最大并发请求数，默认5个。
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次。
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

说明 更多鉴权方式请参见授权访问。

设置EndPoint为cname

如果您已经在Bucket上绑定cname，将该cname直接设置到endpoint即可。如：

```
String endpoint = "http://new-image.xxxxx.com";
...
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

启用日志

移动端的使用环境比较复杂。会出现部分区域或某一个时段无法正常使用OSS SDK的情况。为了进一步方便开发者定位问题，OSS SDK在开启日志记录功能后，会将一些LOG信息记录在本地。如需开启需要在OSSClient使用之前进行初始化，调用方法如下。

```
//日志的样式
//通过调用OSSLog.enableLog()开启可以在控制台看到日志，
//并且会支持写入手机sd卡中的一份日志文件位置在内置sd卡路径\OSSLog\logs.csv 默认不开启
//日志会记录oss操作行为中的请求数据，返回数据，异常信息
//例如requestId,response header等，下边是一个日志记录case
//android_version：5.1 android版本
//mobile_model：XT1085 android手机型号
//network_state：connected 网络状况
//network_type：WIFI 网络连接类型
//具体的操作行为信息:
//[2017-09-05 16:54:52] - Encounter local execpiton: //java.lang.IllegalArgumentException: The bucket name is
invalid.
//A bucket name must:
//1) be comprised of lower-case characters, numbers or dash(-);
//2) start with lower case or numbers;
//3) be between 3-63 characters long.
//------>end of log
OSSLog.enableLog(); //调用此方法即可开启日志
```

> **说明**
> - 日志文件内置sd卡路径\OSSLog\logs.csv。
> - 您可以自行选择将文件上传至服务器，进一步追踪问题。
> - 您还可以接入阿里云SLS日志服务进行日志文件上传，详情请参考日志服务。

设置网络参数

也可以在初始化的时候设置详细的ClientConfiguration：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 在移动端建议使用STS的方式初始化OSSClient。
OSSCredentialProvider credentialProvider = new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.SecurityToken>");
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒。
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒。
conf.setMaxConcurrentRequest(5); // 最大并发请求书，默认5个。
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次。
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider, conf);
```

对 SDK 中同步接口、异步接口的一些说明

考虑到移动端开发场景下不允许在UI线程执行网络请求的编程规范，SDK大多数接口都提供了同步、异步两种调用方式，同步接口调用后会阻塞等待结果返回，而异步接口需要在请求时传入回调函数，请求的执行结果将在回调中处理。

同步接口不能在UI线程调用。遇到异常时，将直接抛出ClientException或者ServiceException异常，前者指本地遇到的异常如网络异常、参数非法等；后者指OSS返回的服务异常，如鉴权失败、服务器错误等。

异步请求遇到异常时，异常会在回调函数中处理。

调用异步接口时，函数会直接返回一个Task，Task可以取消、等待直到完成、或者直接获取结果。如：

```
OSSAsyncTask task = oss.asyncGetObejct(...);
task.cancel(); // 可以取消任务
task.waitUntilFinished(); // 等待直到任务完成
GetObjectResult result = task.getResult(); // 阻塞等待结果返回
```

> **说明** 接口支持同步和异步两种调用方式，考虑到简洁性，本文档中只有部分重要接口会同时提供同步、异步两种调用的示例，其他接口暂时以异步调用的示例为主。

设置是否开启DNS配置

```
ClientConfiguration conf = new ClientConfiguration();
conf.setHttpDnsEnable(true);//默认为true 表开启，需要关闭可以设置为false。
```

设置自定义user-agent

```
ClientConfiguration conf = new ClientConfiguration();
//设置的ua会默认添加到sdk默认ua的最后边，取得时候自行去取最后一个。
conf.setUserAgentMark("customUserAgent");
```

8.4. 快速入门

本节介绍如何快速使用 OSS Android SDK 完成常见操作，如创建存储空间、上传文件、下载文件等。

本工程的更多用法请参考以下两种方式：

- 查看 sample 目录（包含上传本地文件、下载文件、断点续传、设置回调等示例），详情请点击查看。
- 直接 git clone 工程。

运行本工程前，您需要配置必要参数 Config，配置 Config 示例代码如下：

```
public class Config {
  // To run the sample correctly, the following variables must have valid values.
  // The endpoint value below is just the example. Please use proper value according to your region
  // 访问的endpoint地址
  public static final String OSS_ENDPOINT = "http://oss-cn-shanghai.aliyuncs.com";
  //callback 测试地址
  public static final String OSS_CALLBACK_URL = "http://oss-demo.aliyuncs.com:23450";
  // STS 鉴权服务器地址。
  // 或者根据工程sts_local_server目录中本地鉴权服务脚本代码启动本地STS鉴权服务器。
  public static final String STS_SERVER_URL = "http://****/sts/getsts";//STS 地址
  public static final String BUCKET_NAME = "请输入bucket名称";
  public static final String OSS_ACCESS_KEY_ID = "<yourAccessKeyId>";;
  public static final String OSS_ACCESS_KEY_SECRET = "<yourAccessKeySecret>";
  public static final int DOWNLOAD_SUC = 1;
  public static final int DOWNLOAD_Fail = 2;
  public static final int UPLOAD_SUC = 3;
  public static final int UPLOAD_Fail = 4;
  public static final int UPLOAD_PROGRESS = 5;
  public static final int LIST_SUC = 6;
  public static final int HEAD_SUC = 7;
  public static final int RESUMABLE_SUC = 8;
  public static final int SIGN_SUC = 9;
  public static final int BUCKET_SUC = 10;
  public static final int GET_STS_SUC = 11;
  public static final int MULTIPART_SUC = 12;
  public static final int STS_TOKEN_SUC = 13;
  public static final int FAIL = 9999;
  public static final int REQUESTCODE_AUTH = 10111;
  public static final int REQUESTCODE_LOCALPHOTOS = 10112;
}
```

说明

- 有关如何配置 STS 鉴权服务器地址，请参考快速搭建移动应用直传服务。
- sts_local_server 脚本内容，请参考GitHub。

运行工程 demo 如下：

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest("<bucketName>");
// 设置存储空间的访问权限为公共读，默认为私有读写。
createBucketRequest.setBucketACL(CannedAccessControlList.PublicRead);
// 指定存储空间所在的地域。
createBucketRequest.setLocationConstraint("oss-cn-hangzhou");
OSSAsyncTask createTask = oss.asyncCreateBucket(createBucketRequest, new OSSCompletedCallback<Cre
ateBucketRequest, CreateBucketResult>() {
  @Override
  public void onSuccess(CreateBucketRequest request, CreateBucketResult result) {
    Log.d("locationConstraint", request.getLocationConstraint());
    }
  @Override
  public void onFailure(CreateBucketRequest request, ClientException clientException, ServiceException ser
viceException) {
    // 请求异常。
    if (clientException != null) {
      // 本地异常，如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于将指定的本地文件上传到OSS：

```
// 构造上传请求。
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectName>", "<uploadFilePath>");
// 异步上传时可以设置进度回调。
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
  @Override
  public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
    Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectR
esult>() {
  @Override
  public void onSuccess(PutObjectRequest request, PutObjectResult result) {
    Log.d("PutObject", "UploadSuccess");
    Log.d("ETag", result.getETag());
    Log.d("RequestId", result.getRequestId());
  }
  @Override
  public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 请求异常。
    if (clientExcepion != null) {
      // 本地异常，如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.cancel(); // 可以取消任务。
// task.waitUntilFinished(); // 等待上传完成。
```

下载指定文件

以下代码用于将指定的OSS文件下载到本地文件：

```
// 构造下载文件请求。
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectName>");
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectRe
sult>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
    // 请求成功。
    Log.d("asyncGetObject", "DownloadSuccess");
    Log.d("Content-Length", "" + result.getContentLength());
    InputStream inputStream = result.getObjectContent();
    byte[] buffer = new byte[2048];
    int len;
    try {
      while ((len = inputStream.read(buffer)) != -1) {
        // 您可以在此处编写代码来处理下载的数据。
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  @Override
  // GetObject请求成功，将返回GetObjectResult，其持有一个输入流的实例。返回的输入流，请自行处理。
  public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 请求异常。
    if (clientExcepion != null) {
      // 本地异常，如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.cancel(); // 可以取消任务。
// task.waitUntilFinished(); // 等待任务完成。
```

说明 返回数据的输入流需自行处理。

8.5. 存储空间

8.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest("<bucketName>");
// 指定Bucket的ACL权限。
createBucketRequest.setBucketACL(CannedAccessControlList.PublicRead);
// 指定Bucket所在的数据中心。
createBucketRequest.setLocationConstraint("oss-cn-hangzhou");
// 异步创建存储空间。
OSSAsyncTask createTask = oss.asyncCreateBucket(createBucketRequest, new OSSCompletedCallback<Cre
ateBucketRequest, CreateBucketResult>() {
  @Override
  public void onSuccess(CreateBucketRequest request, CreateBucketResult result) {
    Log.d("asyncCreateBucket", "Success");
  }
  @Override
  public void onFailure(CreateBucketRequest request, ClientException clientException, ServiceException ser
viceException) {
    // 请求异常。
    if (clientException != null) {
      // 本地异常，如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

上述代码在创建Bucket时，指定了Bucket的ACL和所在地域。

- 同一阿里云账号在同一地域内创建的Bucket总数不能超过100个。
- 创建Bucket时可以选择Bucket ACL权限，如果不设置ACL，默认是private。
- 每个Bucket的名称全局唯一，不能出现同名Bucket，否则会创建失败。
- 创建成功后，结果返回至Bucket所在地域。

8.5.2. 获取存储空间的访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间访问权限（ACL）。

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	private

访问权限	描述	访问权限值
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	public-read-write

以下代码用于获取存储空间的访问权限：

```
GetBucketACLRequest getBucketACLRequest = new GetBucketACLRequest("<bucketName>");
// 获取存储空间访问权限。
OSSAsyncTask getBucketAclTask = oss.asyncGetBucketACL(getBucketACLRequest, new OSSCompletedCallb
ack<GetBucketACLRequest, GetBucketACLResult>() {
  @Override
  public void onSuccess(GetBucketACLRequest request, GetBucketACLResult result) {
    Log.d("asyncGetBucketACL", "Success!");
    Log.d("BucketAcl", result.getBucketACL());
    Log.d("Owner", result.getBucketOwner());
    Log.d("ID", result.getBucketOwnerID());
  }
  @Override
  public void onFailure(GetBucketACLRequest request, ClientException clientException, ServiceException se
rviceException) {
    // 请求异常。
    if (clientException != null) {
      // 本地异常，如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

说明

- 目前Bucket有三种访问权限：public-read-write，public-read和private。
- 只有Bucket的拥有者才能使用Get Bucket ACL这个接口。
- 获取的结果中返回Bucket拥有者ID、拥有者名称（和ID保持一致）和权限。
- 获取存储空间访问权限的更多详情，请参见Get Bucket Acl。

8.5.3. 获取存储空间信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info）：

```
GetBucketInfoRequest request = new GetBucketInfoRequest("<bucketName>");
// 获取存储空间信息。
OSSAsyncTask task = oss.asyncGetBucketInfo(request, new OSSCompletedCallback<GetBucketInfoRequest,
GetBucketInfoResult>() {
  @Override
  public void onSuccess(GetBucketInfoRequest request, GetBucketInfoResult result) {
    OSSLog.logInfo("code: " + result.getStatusCode());
  }
  @Override
  public void onFailure(GetBucketInfoRequest request, ClientException clientException, ServiceException se
rviceException) {
    // 请求异常。
    if (clientException != null) {
      // 本地异常，如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

8.5.4. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

说明

- 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
- 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest("<bucketName>");
// 异步删除存储空间。
OSSAsyncTask deleteBucketTask = oss.asyncDeleteBucket(deleteBucketRequest, new OSSCompletedCallba
ck<DeleteBucketRequest, DeleteBucketResult>() {
  @Override
  public void onSuccess(DeleteBucketRequest request, DeleteBucketResult result) {
    Log.d("asyncDeleteBucket", "Success!");
  }
  @Override
  public void onFailure(DeleteBucketRequest request, ClientException clientException, ServiceException ser
viceException) {
    // 请求异常。
    if (clientException != null) {
      // 本地异常，如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

注意

- 为防止误删除，OSS不允许用户删除一个非空的存储空间。
- 存储空间删除后不可恢复，请谨慎操作。
- 删除存储空间的更多详情，请参见DeleteBucket。

8.5.5. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

每条生命周期规则包含以下内容：

- 生命周期规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 生命周期策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
  - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
  - 配置到整个存储空间。此种方式只能创建一条规则。
- 指定文件过期时间。有两种指定方式：
  - 指定一个过期天数N，文件会在其最近更新时间点的N天后过期。
  - 指定一个过期时间点，最近更新时间在该时间点之前的文件全部过期。

- 生命周期规则是否生效。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
PutBucketLifecycleRequest request = new PutBucketLifecycleRequest();
request.setBucketName("yourBucketName");
BucketLifecycleRule rule1 = new BucketLifecycleRule();
// 设置规则ID和文件前缀。
rule1.setIdentifier("1");
rule1.setPrefix("A");
//设置是否执行生命周期规则。如果值为true，则OSS会定期执行该规则；如果值为false，则OSS会忽略该规则。
rule1.setStatus(true);
//距最后修改时间200天后过期。
rule1.setDays("200");
//30天后自动转为归档存储类型（Archive）
rule1.setArchiveDays("30");
//未完成分片3天后过期。
rule1.setMultipartDays("3");
//15天后自动转为低频存储类型（IA）。
rule1.setIADays("15");
BucketLifecycleRule rule2 = new BucketLifecycleRule();
rule2.setIdentifier("2");
rule2.setPrefix("B");
rule2.setStatus(true);
rule2.setDays("300");
rule2.setArchiveDays("30");
rule2.setMultipartDays("3");
rule2.setIADays("15");
ArrayList<BucketLifecycleRule> lifecycleRules = new ArrayList<BucketLifecycleRule>();
lifecycleRules.add(rule1);
lifecycleRules.add(rule2);
request.setLifecycleRules(lifecycleRules);
OSSAsyncTask task = oss.asyncPutBucketLifecycle(request, new OSSCompletedCallback<PutBucketLifecycl
eRequest, PutBucketLifecycleResult>() {
  @Override
  public void onSuccess(PutBucketLifecycleRequest request, PutBucketLifecycleResult result) {
    OSSLog.logInfo("code::"+result.getStatusCode());
  }
  @Override
  public void onFailure(PutBucketLifecycleRequest request, ClientException clientException, ServiceExcepti
on serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

有关设置生命周期的更多信息，请参见PutBucketLifecycle。

查看生命周期规则

以下代码用于查看生命周期规则：

```
GetBucketLifecycleRequest request = new GetBucketLifecycleRequest();
request.setBucketName("yourBucketName");
OSSAsyncTask task = oss.asyncGetBucketLifecycle(request, new OSSCompletedCallback<GetBucketLifecycl
eRequest, GetBucketLifecycleResult>() {
  @Override
  public void onSuccess(GetBucketLifecycleRequest request, GetBucketLifecycleResult result) {
    ArrayList<BucketLifecycleRule> list = result.getlifecycleRules();
    for (BucketLifecycleRule rule : list){
      OSSLog.logInfo("info: " + rule.getIdentifier());
    }
  }
  @Override
  public void onFailure(GetBucketLifecycleRequest request, ClientException clientException, ServiceExcepti
on serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

有关查看生命周期的更多信息，请参见GetBucketLifecycle。

清空生命周期规则

以下代码用于清空生命周期规则：

```
DeleteBucketLifecycleRequest request = new DeleteBucketLifecycleRequest();
request.setBucketName("yourBucketName");
OSSAsyncTask task = oss.asyncDeleteBucketLifecycle(request,"yourBucketName" new OSSCompletedCallb
ack<DeleteBucketLifecycleRequest, DeleteBucketLifecycleResult>() {
  @Override
  public void onSuccess(DeleteBucketLifecycleRequest request, DeleteBucketLifecycleResult result) {
    OSSLog.logInfo("code : "+result.getStatusCode());
  }
  @Override
  public void onFailure(DeleteBucketLifecycleRequest request, ClientException clientException, ServiceExce
ption serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

有关清空生命周期规则的更多信息，请参见DeleteBucketLifecycle。

8.5.6. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。

- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链

以下代码用于设置防盗链：

```
PutBucketRefererRequest request = new PutBucketRefererRequest();
request.setBucketName("yourBucketName");
//添加Referer白名单。Referer参数支持通配符星号（*）和问号（？）。
ArrayList<String> referers = new ArrayList<String>();
referers.add("http://www.aaa.com");
referers.add("http://www.*.com");
referers.add("http://www.?.com");
request.setReferers(referers);
OSSAsyncTask task = oss.asyncPutBucketReferer(request, new OSSCompletedCallback<PutBucketRefererR
equest, PutBucketRefererResult>() {
  @Override
  public void onSuccess(PutBucketRefererRequest request, PutBucketRefererResult result) {
    OSSLog.logInfo("code: " + result.getStatusCode());
  }
  @Override
  public void onFailure(PutBucketRefererRequest request, ClientException clientException, ServiceExceptio
n serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
GetBucketRefererRequest request = new GetBucketRefererRequest();
request.setBucketName("yourBucketName");
OSSAsyncTask task = oss.asyncGetBucketReferer(request, new OSSCompletedCallback<GetBucketRefererR
equest, GetBucketRefererResult>() {
  @Override
  public void onSuccess(GetBucketRefererRequest request, GetBucketRefererResult result) {
    // 获取存储空间Referer白名单列表。
    ArrayList<String> list = result.getReferers();
    for (String ref : list){
      OSSLog.logInfo("info: " + ref);
    }
  }
  @Override
  public void onFailure(GetBucketRefererRequest request, ClientException clientException, ServiceExceptio
n serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

清空防盗链

以下代码用于清空防盗链：

```
PutBucketRefererRequest request = new PutBucketRefererRequest();
request.setBucketName("yourBucketName");
request.setAllowEmpty(true);
// 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
ArrayList<String> referers = new ArrayList<String>();
request.setReferers(referers);
OSSAsyncTask task = oss.asyncPutBucketReferer(request, new OSSCompletedCallback<PutBucketRefererR
equest, PutBucketRefererResult>() {
  @Override
  public void onSuccess(PutBucketRefererRequest request, PutBucketRefererResult result) {
    OSSLog.logInfo("code: " + result.getStatusCode());
  }
  @Override
  public void onFailure(PutBucketRefererRequest request, ClientException clientException, ServiceExceptio
n serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

8.5.7. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

关于访问日志的更多信息，请参见开发指南中的日志转存。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
PutBucketLoggingRequest request = new PutBucketLoggingRequest();
//指定要开启访问日志记录的源Bucket。
request.setBucketName("<yourSourceBucketName>");
//指定存储访问日志的目标Bucket。
//目标Bucket和源Bucket必须属于同一地域。源Bucket和目标Bucket可以是同一个Bucket，也可以是不同的Bucket
。
request.setTargetBucketName("<yourTargetBucketName>");
//设置日志文件存放的目录。
request.setTargetPrefix("<yourTargetPrefix>");
OSSAsyncTask task = oss.asyncPutBucketLogging(request, new OSSCompletedCallback<PutBucketLogging
Request, PutBucketLoggingResult>() {
  @Override
  public void onSuccess(PutBucketLoggingRequest request, PutBucketLoggingResult result) {
    OSSLog.logInfo("code::"+result.getStatusCode());
  }
  @Override
  public void onFailure(PutBucketLoggingRequest request, ClientException clientException, ServiceExceptio
n serviceException) {
     OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

开启存储空间访问日志记录的更多信息，请参见PutBucketLogging。

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
GetBucketLoggingRequest request = new GetBucketLoggingRequest();
request.setBucketName("<yourSourceBucketName>");
OSSAsyncTask task = oss.asyncGetBucketLogging(request, new OSSCompletedCallback<GetBucketLogging
Request, GetBucketLoggingResult>() {
  @Override
  public void onSuccess(GetBucketLoggingRequest request, GetBucketLoggingResult result) {
    OSSLog.logInfo("info: " + result.getTargetBucketName()+"*"+result.getTargetPrefix());
  }
  @Override
  public void onFailure(GetBucketLoggingRequest request, ClientException clientException, ServiceExceptio
n serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

查看存储空间的访问日志配置的更多信息，请参见GetBucketLogging。

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
DeleteBucketLoggingRequest request = new DeleteBucketLoggingRequest();
request.setBucketName("<yourSourceBucketName>");
OSSAsyncTask task = oss.asyncDeleteBucketLogging(request, new OSSCompletedCallback<DeleteBucketLo
ggingRequest, DeleteBucketLoggingResult>() {
  @Override
  public void onSuccess(DeleteBucketLoggingRequest request, DeleteBucketLoggingResult result) {
    OSSLog.logInfo("code::"+result.getStatusCode());
  }
  @Override
  public void onFailure(DeleteBucketLoggingRequest request, ClientException clientException, ServiceExcep
tion serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

关闭存储空间的访问日志记录功能的更多信息，请参见DeleteBucketLogging。

8.6. 上传文件

8.6.1. 概述

本文档介绍 OSS Android SDK 上传文件的方式。

在OSS中，操作的基本数据单元是文件（Object）。OSS Android SDK提供了以下三种文件上传方式：

- 简单上传：包括上传本地文件和二进制byte[]数组。最大不能超过5GB。
- 追加上传：最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以通过进度条查看上传进度。上传完成后，您还可以进行上传回调。

8.6.2. 简单上传

本文介绍如何使用简单上传。

上传本地文件

您可以通过同步方式或者异步方式上传本地文件到OSS。

- 调用同步接口上传

  以下代码用于以同步方式上传本地*D:\\localpath*路径下的examplefile.txt到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// 构造上传请求。
// 依次填写Bucket名称（例如examplebucket）、文件完整路径（例如exampledir/exampleobject.txt）和本地完整路径（例如D:\\localpath\\examplefile.txt）。
PutObjectRequest put = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", "D:\\localpath\\examplefile.txt");
// 设置文件元信息为可选操作。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setContentType("application/octet-stream"); // 设置content-type。
// metadata.setContentMD5(BinaryUtil.calculateBase64Md5(uploadFilePath)); // 校验MD5。
// put.setMetadata(metadata);
try {
  PutObjectResult putResult = oss.putObject(put);
  Log.d("PutObject", "UploadSuccess");
  Log.d("ETag", putResult.getETag());
  Log.d("RequestId", putResult.getRequestId());
} catch (ClientException e) {
  // 客户端异常，例如网络异常等。
  e.printStackTrace();
} catch (ServiceException e) {
  // 服务端异常。
  Log.e("RequestId", e.getRequestId());
  Log.e("ErrorCode", e.getErrorCode());
  Log.e("HostId", e.getHostId());
  Log.e("RawMessage", e.getRawMessage());
}
```

说明 在Android中，只能在子线程调用、而不能在UI线程调用同步接口，否则将出现异常。如果希望直接在UI线程中上传，请使用异步接口。

- 调用异步接口上传
  - Android10之前版本

    以下代码用于以异步方式上传本地*D:\\localpath*路径下的examplefile.txt到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// 构造上传请求。
// 依次填写Bucket名称（例如examplebucket）、文件完整路径（例如exampledir/exampleobject.txt）和本地完整路径（例如D:\\localpath\\examplefile.txt）。
PutObjectRequest put = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", "D:\\localpath\\examplefile.txt");
// 异步上传时可以设置进度回调。
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
  @Override
  public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
    Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectResult>() {
  @Override
  public void onSuccess(PutObjectRequest request, PutObjectResult result) {
    Log.d("PutObject", "UploadSuccess");
    Log.d("ETag", result.getETag());
    Log.d("RequestId", result.getRequestId());
  }
  @Override
  public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException serviceException) {
    // 请求异常。
    if (clientExcepion != null) {
      // 客户端异常，例如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.cancel(); // 可以取消任务。
// task.waitUntilFinished(); // 等待任务完成。
```

- Android10及之后版本

  以下代码用于以异步方式上传本地文件到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// 构造上传请求。
//
PutObjectRequest put = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", file
Uri);
// 异步上传时可以设置进度回调。
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
  @Override
  public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
    Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObj
ectResult>() {
  @Override
  public void onSuccess(PutObjectRequest request, PutObjectResult result) {
    Log.d("PutObject", "UploadSuccess");
    Log.d("ETag", result.getETag());
    Log.d("RequestId", result.getRequestId());
  }
  @Override
  public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException se
rviceException) {
    // 请求异常。
    if (clientExcepion != null) {
      // 客户端异常，例如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.cancel(); // 可以取消任务。
// task.waitUntilFinished(); // 等待任务完成。
```

上传二进制byte[]数组

以下代码用于以异步方式上传二进制byte[]数组到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
byte[] uploadData = new byte[100 * 1024];
new Random().nextBytes(uploadData);
// 构造上传请求。
// 依次填写Bucket名称（例如examplebucket）和文件完整路径（例如exampledir/exampleobject.txt）。
PutObjectRequest put = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", upload
Data);
try {
  PutObjectResult putResult = oss.putObject(put);
  Log.d("PutObject", "UploadSuccess");
  Log.d("ETag", putResult.getETag());
  Log.d("RequestId", putResult.getRequestId());
} catch (ClientException e) {
  // 客户端异常，例如网络异常等。
  e.printStackTrace();
} catch (ServiceException e) {
  // 服务端异常。
  Log.e("RequestId", e.getRequestId());
  Log.e("ErrorCode", e.getErrorCode());
  Log.e("HostId", e.getHostId());
  Log.e("RawMessage", e.getRawMessage());
}
```

8.6.3. 追加上传

Append Object以追加写的方式上传文件。

通过Append Object操作创建的Object类型为Appendable Object，而通过Put Object上传的Object是Normal Object。

```
AppendObjectRequest append = new AppendObjectRequest("<bucketName>", "<objectName>", "<uploadFilePath>");
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("application/octet-stream");
append.setMetadata(metadata);
// 设置追加位置。
append.setPosition(0);
// 设置回调。
append.setProgressCallback(new OSSProgressCallback<AppendObjectRequest>() {
  @Override
  public void onProgress(AppendObjectRequest request, long currentSize, long totalSize) {
    Log.d("AppendObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
// 异步追加上传。
OSSAsyncTask task = oss.asyncAppendObject(append, new OSSCompletedCallback<AppendObjectRequest, AppendObjectResult>() {
  @Override
  public void onSuccess(AppendObjectRequest request, AppendObjectResult result) {
    Log.d("AppendObject", "AppendSuccess");
    Log.d("NextPosition", "" + result.getNextPosition());
  }
  @Override
  public void onFailure(AppendObjectRequest request, ClientException clientExcepion, ServiceException serviceException) {
    // 异常处理。
  }
});
```

当您使用Append方式上传，注意对追加位置（Position）这个参数进行正确的设置：

- 当您创建一个Appendable Object时，追加位置设为0。
- 当对Appendable Object进行内容追加时，追加位置设为Object当前长度。

  有两种方式获取该Object长度：

  - 通过上传追加后的返回内容获取。
  - 通过head object获取文件长度。

8.6.4. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用oss.initMultipartUpload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

调用oss.uploadPart方法上传分片数据。

> **说明**
>
> - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
> - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
> - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用oss.CompleteMultipartUpload方法将所有分片合并成完整的文件。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
//初始化分片上传。
//InitiateMultipartUploadRequest用于指定上传文件的名称以及上传文件所属的存储空间的名称。
//objectKey等同于objectName，表示上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jp
g。
InitiateMultipartUploadRequest init = new InitiateMultipartUploadRequest("<bucketName>", "<objectKey>"
);
InitiateMultipartUploadResult initResult = oss.initMultipartUpload(init);
//initResult返回的结果中包含UploadId，UploadId用于唯一标识本次Multipart Upload事件。
String uploadId = initResult.getUploadId();
//设置单个Part的大小，单位为Byte，取值范围为100 KB~5 GB。
int partCount = 100*1024;
//分片上传。
for (int i = 1; i < 5; i++) {
  byte[] data = new byte[partCount];
  RandomAccessFile raf = new RandomAccessFile("path", "r");
  long skip = (i-1) * partCount;
  raf.seek(skip);
  raf.readFully(data, 0, partCount);
  UploadPartRequest uploadPart = new UploadPartRequest();
  uploadPart.setBucketName(mBucketName);
  uploadPart.setObjectKey(objectKey);
  uploadPart.setUploadId(uploadId);
  uploadPart.setPartNumber(i); //上传的Part对应的PartNumber，从1开始标识。
  uploadPart.setPartContent(data);
  try {
    oss.uploadPart(uploadPart);
  } catch (ServiceException serviceException) {
    OSSLog.logError(serviceException.getErrorCode());
  }
}
//完成分片上传。
CompleteMultipartUploadRequest complete = new CompleteMultipartUploadRequest("<bucketName>", "<o
bjectName>", "<uploadId>", "<partETagList>";
//上传回调。完成分片上传请求时可以设置CALLBACK_SERVER参数，请求完成后会向指定的Server Address发送回调
请求。可通过返回结果的completeResult.getServerCallbackReturnBody()查看servercallback结果。
complete.setCallbackParam(new HashMap<String, String>() {
  {
    put("callbackUrl", CALLBACK_SERVER); //修改为您的服务器地址。
    put("callbackBody", "test");
  }
});
CompleteMultipartUploadResult completeResult = oss.completeMultipartUpload(complete);
OSSLog.logError("------------- serverCallback: " + completeResult.getServerCallbackReturnBody());
```

上述代码调用uploadPart来上传每一个分片。

- 每一个分片上传请求均需指定UploadId和PartNumber。PartNumber的范围是1~10000。如果超出该范围，OSS将返回InvalidArgument的错误码。
- uploadPart要求除最后一个Part外，其他的Part大小都要大于100 KB。uploadPart仅在完成分片上传时校验Part的大小。
- 每次上传Part时都要将流定位至此次上传片开头所对应的位置。
- 每次上传Part之后，OSS的返回结果会包含一个分片的ETag值，ETag值为Part数据的MD5值，您需要将

ETag值和块编号组合成PartEtag并保存，用于后续完成分片上传。

本地文件分片上传

通过获取文件路径进行本地文件分片上传有以下两种方法。

> **说明** 分片上传完整示例是按照分片上传流程逐步实现的完整代码，本地文件分片上传的代码是将分片上传完整示例中的代码进行了封装，您只需要使用MultipartUploadRequest即可实现分片上传。

- 方法一

```
ObjectMetadata meta = new ObjectMetadata();
//设置文件元信息等。
meta.setHeader("x-oss-object-acl", "public-read-write");
MultipartUploadRequest rq = new MultipartUploadRequest("<bucketName>", "<objkey>",
    "<filepath>", meta);
//设置PartSize。PartSize默认值为256 KB，最小值为100 KB。
rq.setPartSize(1024 * 1024);
rq.setProgressCallback(new OSSProgressCallback<MultipartUploadRequest>() {
  @Override
  public void onProgress(MultipartUploadRequest request, long currentSize, long totalSize) {
    OSSLog.logDebug("[testMultipartUpload] - " + currentSize + " " + totalSize, false);
  }
});
CompleteMultipartUploadResult result = oss.multipartUpload(rq);
```

- 方式二

```
MultipartUploadRequest request = new MultipartUploadRequest("<bucketName>", "<objkey>",
        "<filepath>");
request.setProgressCallback(new OSSProgressCallback<MultipartUploadRequest>() {
  @Override
  public void onProgress(MultipartUploadRequest request, long currentSize, long totalSize) {
    OSSLog.logDebug("[testMultipartUpload] - " + currentSize + " " + totalSize, false);
  }
});
OSSAsyncTask task = oss.asyncMultipartUpload(request, new OSSCompletedCallback<MultipartUploadR
equest, CompleteMultipartUploadResult>() {
  @Override
  public void onSuccess(MultipartUploadRequest request, CompleteMultipartUploadResult result) {
    OSSLog.logInfo(result.getServerCallbackReturnBody());
  }
  @Override
  public void onFailure(MultipartUploadRequest request, ClientException clientException, ServiceExcepti
on serviceException) {
    OSSLog.logError(serviceException.getRawMessage());
  }
});
// Thread.sleep(100);
//取消分片上传。
// task.cancel();
task.waitUntilFinished();
```

列举已上传分片

调用oss.listParts方法获取某个上传事件所有已上传的分片。

以下代码用于列举已上传分片。

```
//列举分片。
ListPartsRequest listParts = new ListPartsRequest("<bucketName>", "<objectName>", "<uploadId>");
ListPartsResult result = oss.listParts(listParts);
List<PartETag> partETagList = new ArrayList<PartETag>();
for (PartSummary part : result.getParts()) {
  partETagList.add(new PartETag(part.getPartNumber(), part.getETag()));
}
```

> **注意** 默认情况下，如果存储空间中的分片上传事件的数量大于1000，则OSS仅返回1000个Multipart Upload信息，且返回结果中IsTruncated的值为false，并返回NextPartNumberMarker作为下此读取的起点。

取消分片上传

调用oss.abortMultipartUpload方法取消对应UploadId的分片上传请求。当一个分片上传事件被取消后，无法再使用此uploadId做任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传：

```
//取消分片上传。
AbortMultipartUploadRequest abort = new AbortMultipartUploadRequest("<bucketName>", "<objectName
>", "<uploadId>");
AbortMultipartUploadResult abortResult = oss.abortMultipartUpload(abort);
```

8.6.5. 断点续传上传

在无线网络下，上传比较大的文件持续时间长，可能会遇到因为网络条件差、用户切换网络等原因导致上传中途失败，整个文件需要重新上传。为此，Android SDK提供了断点续传上传功能。

背景信息

对于移动端来说，如果不是大文件（例如小于5 GB的文件），不建议使用此种方式上传。断点续传上传是通过分片上传实现的，上传单个文件需要进行多次网络请求，效率不高。以下介绍断点续传上传前以及断点续传上传过程中的注意事项。

- 断点续传上传前

  通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录的保存文件夹。断点续传上传仅在本次上传生效。

  - 如果未指定断点记录的保存文件夹，假设某个分片因为网络原因等导致文件上传失败时，将耗用大量的重试时间及流量来重新上传整个大文件。
  - 如果指定了断点记录的保存文件夹，在文件上传失败时，将从断点记录处继续上传未上传完成的部分。

- 断点续传上传时

  使用断点续传上传时，您需要了解以下信息。

  - 断点续传上传仅支持上传本地文件。断点续传上传支持上传回调，使用方法与常见的上传回调类似，详情请参见Callback。

- 断点续传上传依赖InitMultipartUpload、UploadPart、ListParts、CompleteMultipartUpload、AbortMultipartUpload等接口来实现。如果您需要通过STS鉴权模式来使用断点续传上传，则需要保证您拥有访问上述API接口的权限。
- 断点续传上传默认已开启每个分片上传时的MD5校验，因此无需在request中设置 `Content-Md5` 头部。
- 如果同一任务一直得不到续传，可能会在OSS上积累无用碎片。此时，您可以为Bucket设置lifeCycle规则的方式来定时清理碎片，详情请参见生命周期管理。

示例代码

以下为断点续传上传的常见示例。

- 断点记录不在本地持久保存时，调用 `ResumableUploadRequest` 方法实现断点续传上传的过程如下：

```
// 创建断点上传请求。
// objectKey等同于objectName，表示断点上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");
// 设置上传过程回调。
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
  @Override
  public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
    Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
// 异步调用断点上传。
OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult>() {
  @Override
  public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
    Log.d("resumableUpload", "success!");
  }
  @Override
  public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceException serviceException) {
    // 异常处理。
  }
});
// 等待完成断点上传任务。
resumableTask.waitUntilFinished();
```

- 断点记录在本地持久保存时，调用 `ResumableUploadRequest` 方法实现断点续传上传的过程如下：

```
String recordDirectory = Environment.getExternalStorageDirectory().getAbsolutePath() + "/oss_record/";
File recordDir = new File(recordDirectory);
// 确保断点记录的保存文件夹已存在，如果不存在则新建断点记录的保存文件夹。
if (!recordDir.exists()) {
  recordDir.mkdirs();
}
// 创建断点上传请求，并指定断点记录文件的保存路径，保存路径为断点记录文件的绝对路径。
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>", "<
uploadFilePath>", recordDirectory);
// 设置上传回调。
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
  @Override
  public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
    Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new OSSCompletedCallback<Resu
mableUploadRequest, ResumableUploadResult>() {
  @Override
  public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
    Log.d("resumableUpload", "success!");
  }
  @Override
  public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceExcepti
on serviceException) {
    // 异常处理。
  }
});
// 等待完成断点上传任务。
resumableTask.waitUntilFinished();
```

- 断点续传上传的完整示例代码如下：

```
String recordDirectory = Environment.getExternalStorageDirectory().getAbsolutePath() + "/oss_record/";
File recordDir = new File(recordDirectory);
// 确保断点记录的保存文件夹已存在，如果不存在则新建断点记录的保存文件夹。
if (!recordDir.exists()) {
  recordDir.mkdirs();
}
// 创建断点续传上传请求，并指定断点记录文件的保存路径，保存路径为断点记录文件的绝对路径。
ResumableUploadRequest request = new ResumableUploadRequest("<bucketName>", "<objectKey>", "<
uploadFilePath>", recordDirectory);
// 调用OSSAsyncTask cancel()方法时，DeleteUploadOnCancelling设置为false时，则不删除断点记录文件。如
果不设置此参数，则默认值为true，表示删除断点记录文件，下次再上传用一个文件时则重新上传。
request.setDeleteUploadOnCancelling(false);
// 设置上传回调。
request.setProgressCallback(new OSSProgressCallback<ResumableUploadRequest>() {
  @Override
  public void onProgress(ResumableUploadRequest request, long currentSize, long totalSize) {
    Log.d("resumableUpload", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new OSSCompletedCallback<Resu
mableUploadRequest, ResumableUploadResult>() {
  @Override
  public void onSuccess(ResumableUploadRequest request, ResumableUploadResult result) {
    Log.d("resumableUpload", "success!");
  }
  @Override
  public void onFailure(ResumableUploadRequest request, ClientException clientExcepion, ServiceExcepti
on serviceException) {
    // 异常处理。
  }
});
// 等待完成断点上传任务。
resumableTask.waitUntilFinished();
```

8.6.6. 进度条

进度条用于指示上传或下载的进度。

下面的代码以asyncPutObject方法为例，介绍如何使用进度条。

```
// 构造上传请求。
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectName>", "<uploadFilePath>");
// 设置进度回调函数打印进度条。
put.setProgressCallback(new OSSProgressCallback<PutObjectRequest>() {
  @Override
  public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
    Log.d("PutObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
// 异步上传。
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectR
esult>() {
  @Override
  public void onSuccess(PutObjectRequest request, PutObjectResult result) {
    Log.d("PutObject", "UploadSuccess");
    Log.d("ETag", result.getETag());
    Log.d("RequestId", result.getRequestId());
  }
  @Override
  public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 请求异常。
    if (clientExcepion != null) {
      // 本地异常，如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.cancel(); // 可以取消任务。
// task.waitUntilFinished(); // 等待任务完成。
```

8.6.7. 上传回调

本文介绍如何使用上传回调。

OSS在上传文件完成的时候可以提供回调（Callback）给应用服务器。您只需要在发送给OSS的请求中携带相应的Callback参数，即能实现回调。因为加入了回调请求和响应的过程，相比简单上传，使用回调通知机制一般会导致客户端花费更多的等待时间。

```
PutObjectRequest put = new PutObjectRequest(testBucket, testObject, uploadFilePath);
put.setCallbackParam(new HashMap<String, String>() {
  {
    put("callbackUrl", "110.75.82.106/callback");
    put("callbackHost", "yourCallbackHost");
    put("callbackBodyType", "application/json");
    put("callbackBody", "{\"mimeType\":${mimeType},\"size\":${size}}");
  }
});
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectR
esult>() {
  @Override
  public void onSuccess(PutObjectRequest request, PutObjectResult result) {
    Log.d("PutObject", "UploadSuccess");
    // 只有设置了servercallback，这个值才有数据
    String serverCallbackReturnJson = result.getServerCallbackReturnBody();
    Log.d("servercallback", serverCallbackReturnJson);
  }
  @Override
  public void onFailure(PutObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 异常处理
  }
});
```

如果需要支持自定义参数，参考如下设置：

```
put.setCallbackParam(new HashMap<String, String>() {
  {
    put("callbackUrl", "http://182.92.192.125/leibin/notify.php");
    put("callbackHost", "yourCallbackHost");
    put("callbackBodyType", "application/json");
    put("callbackBody", "{\"object\":${object},\"size\":${size},\"my_var1\":${x:var1},\"my_var2\":${x:var2}}");
  }
});
put.setCallbackVars(new HashMap<String, String>() {
  {
    put("x:var1", "value1");
    put("x:var2", "value2");
  }
});
```

有关Callback的更多信息，请参考Callback API文档说明。

8.7. 下载文件

8.7.1. 概述

OSS Android SDK提供了丰富的文件下载方式。

您可以根据业务场景，选择适当的文件下载方式。

- 流式下载
- 范围下载
- 下载到本地文件
- 限定条件下载

8.7.2. 流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

下载指定文件后将获得文件的输入流，此操作要求用户对该Object有读权限。

同步调用：

```
//构造下载文件请求
//objectKey等同于objectName，表示从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");
//设置下载进度回调
get.setProgressListener(new OSSProgressCallback<GetObjectRequest>() {
        @Override
        public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
          OSSLog.logDebug("getobj_progress: " + currentSize+"  total_size: " + totalSize, false);
        }
      });
try {
  // 同步执行下载请求，返回结果
  GetObjectResult getResult = oss.getObject(get);
  Log.d("Content-Length", "" + getResult.getContentLength());
  // 获取文件输入流
  InputStream inputStream = getResult.getObjectContent();
  byte[] buffer = new byte[2048];
  int len;
  while ((len = inputStream.read(buffer)) != -1) {
    // 处理下载的数据，比如图片展示或者写入文件等
  }
  // 下载后可以查看文件元信息
  ObjectMetadata metadata = getResult.getMetadata();
  Log.d("ContentType", metadata.getContentType());
} catch (ClientException e) {
  // 本地异常如网络异常等
  e.printStackTrace();
} catch (ServiceException e) {
  // 服务异常
  Log.e("RequestId", e.getRequestId());
  Log.e("ErrorCode", e.getErrorCode());
  Log.e("HostId", e.getHostId());
  Log.e("RawMessage", e.getRawMessage());
} catch (IOException e) {
  e.printStackTrace();
}
```

异步调用：

```
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");
//设置下载进度回调
get.setProgressListener(new OSSProgressCallback<GetObjectRequest>() {
        @Override
        public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
          OSSLog.logDebug("getobj_progress: " + currentSize+"  total_size: " + totalSize, false);
        }
      });
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectRe
sult>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
    // 请求成功
    InputStream inputStream = result.getObjectContent();
    byte[] buffer = new byte[2048];
    int len;
    try {
      while ((len = inputStream.read(buffer)) != -1) {
        // 处理下载的数据
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  @Override
  public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 请求异常
    if (clientExcepion != null) {
      // 本地异常如网络异常等
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

8.7.3. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

```
//objectKey等同于objectName，表示下载文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jp
g
GetObjectRequest get = new GetObjectRequest("<bucketName>", "<objectKey>");
// 设置范围
get.setRange(new Range(0, 99)); // 下载0到99字节共100个字节，文件范围从0开始计算
// get.setRange(new Range(100, Range.INFINITE)); // 下载从100个字节到结尾
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectRe
sult>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
    // 请求成功
    InputStream inputStream = result.getObjectContent();
    byte[] buffer = new byte[2048];
    int len;
    try {
      while ((len = inputStream.read(buffer)) != -1) {
        // 处理下载的数据
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  @Override
  public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException service
Exception) {
    // 请求异常
    if (clientExcepion != null) {
      // 本地异常，如网络异常等
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

8.7.4. 下载到本地文件

本文档介绍如何将OSS文件（Object）下载到本地文件。

以下代码用于将指定的OSS文件下载到本地文件。

```
//下载文件。
//objectKey等同于objectname，表示从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
GetObjectRequest get = new GetObjectRequest("BucketName", "objectKey");
oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectResult>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
//开始读取数据。
    long length = result.getContentLength();
    byte[] buffer = new byte[(int) length];
    int readCount = 0;
    while (readCount < length) {
      try{
        readCount += result.getObjectContent().read(buffer, readCount, (int) length - readCount);
      }catch (Exception e){
        OSSLog.logInfo(e.toString());
      }
    }
//将下载后的文件存放在指定的本地路径。
    try {
      FileOutputStream fout = new FileOutputStream("download_filePath");
      fout.write(buffer);
      fout.close();
    } catch (Exception e) {
      OSSLog.logInfo(e.toString());
    }
  }
  @Override
  public void onFailure(GetObjectRequest request, ClientException clientException,
              ServiceException serviceException) {
  }
});
```

8.7.5. 限定条件下载

下载文件时，您可以指定一个或多个限定条件。如果满足限定条件，则下载文件；如果不满足限定条件，则返回错误，且不下载文件。

您可以使用的限定条件如下：

参数	描述
If-Modified-Since	如果指定的时间早于实际修改时间，则正常传输文件，否则返回错误（304 Not modified）。
If-Unmodified-Since	如果指定的时间等于或者晚于文件实际修改时间，则正常传输文件，否则返回错误（412 Precondition failed）。
If-Match	如果指定的ETag和OSS文件的ETag匹配，则正常传输文件，否则返回错误（412 Precondition failed）。

参数	描述
If-None-Match	如果指定的ETag和OSS文件的ETag不匹配，则正常传输文件，否则返回错误（304 Not modified）。

If-Modified-Since和If-Unmodified-Since可以同时存在。If-Match和If-None-Match可以同时存在。

ETag可以通过ossClient.getObjectMeta方法获取。

以下代码用于限定条件下载：

```
public void testGetObjectWithHeaders() throws Exception {
    GetObjectRequest request = new GetObjectRequest(bucketName, objectName);
    request.setRequestHeaders(...);
    OSSAsyncTask task = oss.asyncGetObject(request, getCallback);
    task.waitUntilFinished();
    GetObjectRequest rq = getCallback.request;
    GetObjectResult result = getCallback.result;
  }
```

8.7.6. 进度条

进度条用于指示上传或下载文件的进度。本文以GetObject接口为例介绍如何打印下载文件（Object）的进度条。

以下代码用于打印从examplebucket下载exampleobject.txt文件的进度条。

```
// 构造下载请求。
// 填写Bucket名称和Object完整路径。Object完整路径中不能包含Bucket名称。
GetObjectRequest get = new GetObjectRequest("examplebucket", "exampleobject.txt");
// 设置进度回调函数打印进度条。
get.setProgressListener(new OSSProgressCallback<GetObjectRequest>() {
  @Override
  public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
    Log.d("GetObject", "currentSize: " + currentSize + " totalSize: " + totalSize);
  }
});
// 异步下载。
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObjectRe
sult>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
    Log.d("GetObject", "UploadSuccess");
  }
  @Override
  public void onFailure(GetObjectRequest request, ClientException clientException, ServiceException servic
eException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

8.8. 管理文件

8.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 获取文件元信息
- 列举文件
- 拷贝文件
- 删除文件

8.8.2. 判断文件是否存在

Android SDK提供了方便的同步接口以检测Bucket中是否存在指定的文件。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
try {
  if (oss.doesObjectExist("<bucketName>", "<objectKey>")) {
    Log.d("doesObjectExist", "object exist.");
  } else {
    Log.d("doesObjectExist", "object does not exist.");
  }
} catch (ClientException e) {
  // 本地异常如网络异常等
  e.printStackTrace();
} catch (ServiceException e) {
  // 服务异常
  Log.e("ErrorCode", e.getErrorCode());
  Log.e("RequestId", e.getRequestId());
  Log.e("HostId", e.getHostId());
  Log.e("RawMessage", e.getRawMessage());
}
```

8.8.3. 获取文件访问权限

文件访问权限包括私有、公共读和公共读写三种。本文介绍如何获取文件（Object）的访问权限。

文件访问权限

文件访问权限包括如下三种：

文件访问权限	描述
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。

关于文件访问权限的更多信息，请参见基于读写权限ACL的权限控制。

示例

以下代码用于获取examplebucket存储空间中exampleobject.txt文件的访问权限。

```
// 填写Bucket名称和Object完整路径。Object完整路径中不能包含Bucket名称。
GetObjectACLRequest request = new GetObjectACLRequest("examplebucket", "exampleobject.txt");
// 获取文件访问权限。
oss.asyncGetObjectACL(request, new OSSCompletedCallback<GetObjectACLRequest, GetObjectACLResult>
() {
  @Override
  public void onSuccess(GetObjectACLRequest request, GetObjectACLResult result) {
    Log.d("GetObjectACL", "Success!");
    Log.d("ObjectAcl", result.getObjectACL());
    Log.d("Owner", result.getObjectOwner());
    Log.d("ID", result.getObjectOwnerID());
  }
  @Override
  public void onFailure(GetObjectACLRequest request, ClientException clientException, ServiceException se
rviceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

关于获取文件访问权限的更多信息，请参见GetObjectACL。

8.8.4. 拷贝文件

您可以将文件从一个存储空间（源存储空间）拷贝到同一地域的另一个存储空间（目标存储空间）中。

以下代码用于拷贝文件。

```
// 创建copy请求
CopyObjectRequest copyObjectRequest = new CopyObjectRequest("<srcBucketName>", "<srcObjectKey>",
    "<destBucketName>", "<destObjectKey>");
// 可选设置copy文件元信息
// ObjectMetadata objectMetadata = new ObjectMetadata();
// objectMetadata.setContentType("application/octet-stream");
// copyObjectRequest.setNewObjectMetadata(objectMetadata);
// 异步copy
OSSAsyncTask copyTask = oss.asyncCopyObject(copyObjectRequest, new OSSCompletedCallback<CopyObj
ectRequest, CopyObjectResult>() {
  @Override
  public void onSuccess(CopyObjectRequest request, CopyObjectResult result) {
      Log.d("copyObject", "copy success!");
  }
  @Override
  public void onFailure(CopyObjectRequest request, ClientException clientExcepion, ServiceException servic
eException) {
    // 请求异常
    if (clientExcepion != null) {
      // 本地异常如网络异常等
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

说明

- 源Object和目标Object必须属于同一地域。
- 如果拷贝操作的源Object地址和目标Object地址相同，可以修改已有文件元信息。
- 拷贝文件大小不能超过1G，超过1G需使用Multipart Upload等操作。

8.8.5. 列举文件

本文介绍如何列举存储空间下（Bucket）中的所有文件（Object）、指定个数的文件、指定前缀的文件等。

列举指定个数的文件

以下代码用于列举examplebucket中最多20个文件。

```
// 填写Bucket名称。
ListObjectsRequest request = new ListObjectsRequest("examplebucket");
// 填写返回文件的最大个数。如果不设置此参数，则默认值为100，maxkeys的取值不能大于1000。
request.setMaxKeys(20);
oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, ListObjectsResult>() {
  @Override
  public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
    for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
      Log.i("ListObjects", objectSummary.getKey());
    }
  }
  @Override
  public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException servi
ceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

列举指定前缀的文件

以下代码用于列举examplebucket中以file为前缀的文件。

```
// 填写Bucket名称。
ListObjectsRequest request = new ListObjectsRequest("examplebucket");
// 填写前缀。
request.setPrefix("file");
oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, ListObjectsResult>() {
  @Override
  public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
    for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
      Log.i("ListObjects", objectSummary.getKey());
    }
  }
  @Override
  public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException servi
ceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

列举指定marker之后的文件

以下代码用于列举examplebucket中exampleobject.txt之后的文件。

```
// 填写Bucket名称。
ListObjectsRequest request = new ListObjectsRequest("examplebucket");
// 填写marker。从marker之后按字母排序的第一个开始返回文件。
// 如果marker在存储空间中不存在，则会从符合marker字母排序的下一个开始返回文件。
request.setMarker("exampleobject.txt");
oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, ListObjectsResult>() {
  @Override
  public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
    for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
      Log.i("ListObjects", objectSummary.getKey());
    }
  }
  @Override
  public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException servi
ceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

分页列举所有文件

以下代码用于分页列举examplebucket中的所有文件，每页最多返回20个文件。

```
private String marker = null;
private boolean isCompleted = false;
// 分页列举所有object
public void getAllObject() {
  do {
    OSSAsyncTask task = getObjectList();
    // 阻塞等待请求完成获取NextMarker，请求下一页时需要将请求的marker设置为上一页请求返回的NextMarker。
第一页无需设置。
    // 示例中通过循环分页列举数据，因此需要阻塞等待请求完成获取NextMarker才能请求下一页数据，实际使用时可
根据实际场景判断是否需要阻塞。
    task.waitUntilFinished();
  } while (!isCompleted);
}
// 列举一页文件。
public OSSAsyncTask getObjectList() {
  //填写Bucket名称。
  ListObjectsRequest request = new ListObjectsRequest("examplebucket");
  // 填写每页返回文件的最大个数。如果不设置此参数，则默认值为100，maxkeys的取值不能大于1000。
  request.setMaxKeys(20);
  request.setMarker(marker);
```

```
  request.setMarker(marker);
  OSSAsyncTask task = oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, List
ObjectsResult>() {
    @Override
    public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
      for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
        Log.i("ListObjects", objectSummary.getKey());
      }
      // 最后一页。
      if (!result.isTruncated()) {
        isCompleted = true;
        return;
      }
      // 下一次列举文件的marker。
      marker = result.getNextMarker();
    }
    @Override
    public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException ser
viceException) {
      isCompleted = true;
      // 请求异常。
      if (clientException != null) {
        // 客户端异常，例如网络异常等。
        clientException.printStackTrace();
      }
      if (serviceException != null) {
        // 服务端异常。
        Log.e("ErrorCode", serviceException.getErrorCode());
        Log.e("RequestId", serviceException.getRequestId());
        Log.e("HostId", serviceException.getHostId());
        Log.e("RawMessage", serviceException.getRawMessage());
      }
    }
  });
  return task;
}
```

分页列举指定前缀的文件

以下代码用于分页列举examplebucket中以file为前缀的文件，每页最多返回20个文件。

```
private String marker = null;
private boolean isCompleted = false;
// 分页列举所有文件。
public void getAllObject() {
  do {
    OSSAsyncTask task = getObjectList();
    // 阻塞等待请求完成获取NextMarker，请求下一页时需要将请求的marker设置为上一页请求返回的NextMarker。
第一页无需设置。
    // 示例中通过循环分页列举数据，因此需要阻塞等待请求完成获取NextMarker才能请求下一页数据，实际使用时可
根据实际场景判断是否需要阻塞。
    task.waitUntilFinished();
  } while (!isCompleted);
}
// 列举一页文件。
```

```
// 列举一页文件。
public OSSAsyncTask getObjectList() {
  // 填写Bucket名称。
  ListObjectsRequest request = new ListObjectsRequest("examplebucket");
  // 填写每页返回文件的最大个数。如果不设置此参数，则默认值为100，maxkeys的取值不能大于1000。
  request.setMaxKeys(20);
  // 填写前缀。
  request.setPrefix("file");
  request.setMarker(marker);
  OSSAsyncTask task = oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, List
ObjectsResult>() {
    @Override
    public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
      for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
        Log.i("ListObjects", objectSummary.getKey());
      }
      // 最后一页。
      if (!result.isTruncated()) {
        isCompleted = true;
        return;
      }
      // 下一次列举文件的marker。
      marker = result.getNextMarker();
    }
    @Override
    public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException ser
viceException) {
      isCompleted = true;
      // 请求异常。
      if (clientException != null) {
        // 客户端异常，例如网络异常等。
        clientException.printStackTrace();
      }
      if (serviceException != null) {
        // 服务端异常。
        Log.e("ErrorCode", serviceException.getErrorCode());
        Log.e("RequestId", serviceException.getRequestId());
        Log.e("HostId", serviceException.getHostId());
        Log.e("RawMessage", serviceException.getRawMessage());
      }
    }
  });
  return task;
}
```

列举名称包含特殊字符的文件

如果文件名称包含以下特殊字符，需要进行编码传输。OSS目前仅支持URL编码。

- 单引号（' '）
- 双引号（" "）
- and符号（&）
- 尖括号（< >）

- 顿号（、）
- 中文

以下代码用于列举examplebucket中名称包含特殊字符的文件。

```
// 填写Bucket名称。
ListObjectsRequest request = new ListObjectsRequest("examplebucket");
// 指定文件名称编码。
request.setEncodingType("url");
oss.asyncListObjects(request, new OSSCompletedCallback<ListObjectsRequest, ListObjectsResult>() {
  @Override
  public void onSuccess(ListObjectsRequest request, ListObjectsResult result) {
    for (OSSObjectSummary objectSummary : result.getObjectSummaries()) {
      Log.i("ListObjects", URLDecoder.decode(objectSummary.getKey(), "UTF-8"));
    }
  }
  @Override
  public void onFailure(ListObjectsRequest request, ClientException clientException, ServiceException servi
ceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

关于列举文件的更多信息，请参见GetBucket (ListObjects)。

8.8.6. 删除文件

本文介绍如何单个或者批量删除文件。

> **警告** 文件一旦删除将无法恢复，请谨慎使用删除操作。

注意事项

删除文件时，您需要具有对Object所在Bucket的写权限。

单个删除文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
// 创建删除请求。
// 填写Bucket名称和Object完整路径。Object完整路径中不能包含Bucket名称。
DeleteObjectRequest delete = new DeleteObjectRequest("examplebucket", "exampleobject.txt");
// 异步删除。
OSSAsyncTask deleteTask = oss.asyncDeleteObject(delete, new OSSCompletedCallback<DeleteObjectRequ
est, DeleteObjectResult>() {
  @Override
  public void onSuccess(DeleteObjectRequest request, DeleteObjectResult result) {
    Log.d("asyncDeleteObject", "success!");
  }
  @Override
  public void onFailure(DeleteObjectRequest request, ClientException clientExcepion, ServiceException serv
iceException) {
    // 请求异常。
    if (clientExcepion != null) {
      // 客户端异常，例如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

批量删除文件

批量删除文件时，每次最多删除1000个文件。

返回结果包括如下两种模式，默认返回模式为详细模式，请根据实际选择返回模式。

- 详细模式（verbose）：未设置isQuiet或者设置isQuiet为false，表示返回所有删除的文件列表。
- 简单模式（quiet）：设置isQuiet为true，表示只返回删除失败的文件列表。

以下代码用于删除examplebucket中指定的多个文件且只返回删除失败的文件列表。

```
// 设置需要删除的多个Object完整路径。Object完整路径中不能包含Bucket名称。
List<String> objectKeys = new ArrayList<String>();
objectKeys.add("exampleobject.txt");
objectKeys.add("testfolder/sampleobject.txt");
// 设置为简单模式，只返回删除失败的文件列表。
DeleteMultipleObjectRequest request = new DeleteMultipleObjectRequest("examplebucket", objectKeys, tr
ue);
mOss.asyncDeleteMultipleObject(request, new OSSCompletedCallback<DeleteMultipleObjectRequest, Dele
teMultipleObjectResult>() {
  @Override
  public void onSuccess(DeleteMultipleObjectRequest request, DeleteMultipleObjectResult result) {
    Log.i("DeleteMultipleObject", "success");
  }
  @Override
  public void onFailure(DeleteMultipleObjectRequest request, ClientException clientException, ServiceExcep
tion serviceException) {
    // 请求异常。
    if (clientException != null) {
      // 客户端异常，例如网络异常等。
      clientException.printStackTrace();
    }
    if (serviceException != null) {
      // 服务端异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
```

8.8.7. 设置Content-Type

在Web服务中Content-Type用于设定文件的类型，决定以哪种形式、什么编码读取这个文件。

某些情况下，对于上传的文件需要设置Content-Type，否则文件不能以需要的形式和编码来读取。如果使用SDK上传文件时没有指定Content-Type，SDK会帮您根据后缀自动添加Content-Type。

```
// 构造上传请求
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<objectKey>", "<uploadFilePath>");
ObjectMetadata metadata = new ObjectMetadata();
// 指定Content-Type
metadata.setContentType("application/octet-stream");
// user自定义metadata
metadata.addUserMetadata("x-oss-meta-name1", "value1");
put.setMetadata(metadata);
OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<PutObjectRequest, PutObjectR
esult>() {
  ...
});
```

8.8.8. 获取文件元信息

通过HeadObject方法可以只获取文件元信息而不获取文件的实体。

获取文件元信息代码如下：

```
// 创建同步获取文件元信息请求。
HeadObjectRequest head = new HeadObjectRequest("<bucketName>", "<objectKey>");
// 获取文件元信息。
OSSAsyncTask task = oss.asyncHeadObject(head, new OSSCompletedCallback<HeadObjectRequest, HeadO
bjectResult>() {
  @Override
  public void onSuccess(HeadObjectRequest request, HeadObjectResult result) {
 // 获取文件长度。
    Log.d("headObject", "object Size: " + result.getMetadata().getContentLength());
 // 获取文件类型。
    Log.d("headObject", "object Content Type: " + result.getMetadata().getContentType());
  }
  @Override
  public void onFailure(HeadObjectRequest request, ClientException clientExcepion, ServiceException servic
eException) {
    // 请求异常。
    if (clientExcepion != null) {
      // 本地异常，如网络异常等。
      clientExcepion.printStackTrace();
    }
    if (serviceException != null) {
      // 服务异常。
      Log.e("ErrorCode", serviceException.getErrorCode());
      Log.e("RequestId", serviceException.getRequestId());
      Log.e("HostId", serviceException.getHostId());
      Log.e("RawMessage", serviceException.getRawMessage());
    }
  }
});
// task.waitUntilFinished(); //等待任务完成。
```

8.8.9. 解冻归档文件

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档文件。

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档文件：

```
//解冻归档文件。
RestoreObjectRequest restore = new RestoreObjectRequest();
restore.setBucketName("yourBucketName");
restore.setObjectKey("yourObjectName");
OSSAsyncTask task = oss.asyncRestoreObject(restore, new OSSCompletedCallback<RestoreObjectRequest,
    RestoreObjectResult>() {
  @Override
  public void onSuccess(RestoreObjectRequest request, RestoreObjectResult result) {
    OSSLog.logInfo("code::"+result.getStatusCode());
  }
  @Override
  public void onFailure(RestoreObjectRequest request, ClientException clientException,
              ServiceException serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

归档存储类型的详细说明请参见存储类型介绍。归档类型涉及的各参数详细说明请参见API文档RestoreObject。

8.8.10. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
PutSymlinkRequest putSymlink = new PutSymlinkRequest();
putSymlink.setBucketName("yourBucketName");
//设置软链接文件名。
putSymlink.setObjectKey("yourSymLink");
//设置目标文件名。
putSymlink.setTargetObjectName("yourTargetObjectName");
OSSAsyncTask task = oss.asyncPutSymlink(putSymlink, new OSSCompletedCallback<PutSymlinkRequest,
    PutSymlinkResult>() {
  @Override
  public void onSuccess(PutSymlinkRequest request, PutSymlinkResult result) {
    OSSLog.logInfo("code:"+result.getStatusCode());
  }
  @Override
  public void onFailure(PutSymlinkRequest request, ClientException clientException,
              ServiceException serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
//获取软链接指向的文件信息。
GetSymlinkRequest getSymlink = new GetSymlinkRequest();
getSymlink.setBucketName("yourBucketName");
getSymlink.setObjectKey("yourSymLink");
OSSAsyncTask task = oss.asyncGetSymlink(getSymlink, new OSSCompletedCallback<GetSymlinkRequest,
    GetSymlinkResult>() {
  @Override
  public void onSuccess(GetSymlinkRequest request, GetSymlinkResult result) {
    OSSLog.logInfo("targ::"+result.getTargetObjectName());
  }
  @Override
  public void onFailure(GetSymlinkRequest request, ClientException clientException,
              ServiceException serviceException) {
    OSSLog.logError("error: "+serviceException.getRawMessage());
  }
});
task.waitUntilFinished();
```

获取软链接的详细信息请参见GetSymlink。

8.9. 数据安全性

OSS Android SDK提供了数据完整性校验方法，保证您在上传、下载和拷贝过程中数据的安全性。

由于移动端网络环境的复杂性，数据在客户端和服务器之间传输时可能会出错。为此，OSS Android SDK 提供了基于CRC端到端以及MD5两种数据完整性校验方式。

- CRC校验

  在读取下载数据流的时候，如果开启了CRC校验，会在数据流读取完毕后自动验证数据完整性。

  以下代码用于开启CRC校验：

```
GetObjectRequest request = new GetObjectRequest(OSSTestConfig.ANDROID_TEST_BUCKET, testFile);
//开启CRC效验。
request.setCRC64(OSSRequest.CRC64Config.YES);
//....
try{
 GetObjectResult result = oss.getObject(request);
 InputStream in = result.getObjectContent();
 ByteArrayOutputStream output = new ByteArrayOutputStream();
   byte[] buffer = new byte[BUFFER_SIZE];
   int len;
   while ((len = in.read(buffer)) > -1) {
        output.write(buffer, 0, len);
   }
   output.flush();
 in.close();
}catch(ClientException e){
 //...
}catch(InconsistentException e){
   //....
}
```

- MD5校验

  如果要校验分片上传到OSS的文件和本地文件是否一致，可以在上传分片时携带分片的Content-MD5值，OSS服务器会帮助用户进行MD5校验。只有OSS服务器接收到的分片MD5值和Content-MD5一致时才可以上传成功，从而保证上传分片的一致性。

  以下代码用于设置MD5验证：

```
UploadPartRequest uploadPart = new UploadPartRequest("<bucketName>", "<objectKey>", uploadId, cur
rentIndex);
// 设置分片内容
uploadPart.setPartContent(partData);
//设置MD5内容
uploadPart.setMd5Digest(BinaryUtil.calculateBase64Md5(data));
```

8.10. 签名URL

Android SDK支持签名私有资源的指定有效时长或者公开的访问URL，用于转给第三方实现授权访问。

- 签名私有资源的指定有效时长的访问URL

  如果Bucket或Object的权限为私有，则需要调用presignConstrainedObjectURL接口获取签名后的URL：

```
String url = oss.presignConstrainedObjectURL("<bucketName>", "<objectKey>", 30 * 60);
```

- 签名公开的访问URL

  如果Bucket或Object的权限为公共读或公共读写，则需要调用presignPublicObjectURL接口获取可公开访问Object的URL：

```
String url = oss.presignPublicObjectURL("<bucketName>", "<objectKey>");
```

8.11. 授权访问

Android SDK提供了STS鉴权模式和自签名模式来保障移动终端的安全性。

背景信息

无论是STS鉴权模式还是自签名模式，您实现的回调函数都需要保证调用时的返回结果。如果您需要实现向业务Server获取Token、Signature的网络请求，建议调用网络库的同步接口。回调都是在SDK发起具体请求时，在请求的子线程中执行，所以不会阻塞主线程。

STS鉴权模式

> **说明** 使用STS模式授权需要先开通阿里云访问控制RAM服务。

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。
- 设置StsToken

  您可以在App中，预先通过某种方式（如通过网络请求从您的业务Server上）获取StsToken，然后用StsToken初始化SDK。通过这种方式获取StsToken时需要注意StsToken的过期时间，并在StsToken即将过期时手动更新StsToken到SDK中。

  初始化SDK示例代码如下：

  ```
  String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
  OSSCredentialProvider credentialProvider = new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.SecurityToken>");
  OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
  ```

  当判断StsToken即将过期时，您可以重新构造OSSClient，也可以通过如下方式更新CredentialProvider：

  ```
  oss.updateCredentialProvider(new OSSStsTokenCredentialProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.SecurityToken>"));
  ```

- 获取StsToken回调

  如果您期望SDK自动更新StsToken，那么您需要在SDK的应用中实现回调。通过您实现回调的方式去获取Federation Token（即StsToken），SDK会使用此StsToken来进行加签处理，并在需要更新时主动调用此回调来获取StsToken：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
OSSCredentialProvider credentialProvider = new OSSFederationCredentialProvider() {
  @Override
  public OSSFederationToken getFederationToken() {
  // 获取FederationToken，并将其构造为OSSFederationToken对象返回。如果因某种原因FederationToken获取失败，服务器则直接返回null。
    OSSFederationToken * token;
    // 从您的服务器中获取token。
    ...
    return token;
  }
};
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

说明 此外，如果您已经通过别的方式获取了StsToken所需的各个字段，也可以在回调中直接返回StsToken。但您需要手动处理StsToken的更新，且更新后重新设置该OSSClient实例的OSSCredentialProvider。

假设您访问的Server地址为：http://localhost:8080/distribute-token.json，则返回的数据如下：

```
{
  "StatusCode": 200,
  "AccessKeyId":"STS.iA645eTOXEqP3cg3****",
  "AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojU****",
  "Expiration":"2015-11-03T09:52:59Z",
  "SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS/p3OgaUC+oJl3XSlbJ7StKpQ*****"}
```

实现 `OSSFederationCredentialProvider` 的示例如下：

```
OSSCredentialProvider credetialProvider = new OSSFederationCredentialProvider() {
  @Override
  public OSSFederationToken getFederationToken() {
    try {
      URL stsUrl = new URL("http://localhost:8080/distribute-token.json");
      HttpURLConnection conn = (HttpURLConnection) stsUrl.openConnection();
      InputStream input = conn.getInputStream();
      String jsonText = IOUtils.readStreamAsString(input, OSSConstants.DEFAULT_CHARSET_NAME);
      JSONObject jsonObjs = new JSONObject(jsonText);
      String ak = jsonObjs.getString("AccessKeyId");
      String sk = jsonObjs.getString("AccessKeySecret");
      String token = jsonObjs.getString("SecurityToken");
      String expiration = jsonObjs.getString("Expiration");
      return new OSSFederationToken(ak, sk, token, expiration);
    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }
};
```

自签名模式

您可以把AccessKey ID和AccessKey Secret保存在自有服务器中后，通过自有服务器对客户端信息进行签名。具体步骤如下：

1. 在客户端获取并发送待签名的字符串到自有服务器。
   i. 构造请求时通过SDK中OSSCustomSignerCredentialProvider的signContent方法获取待签名的字符串。
   ii. 发送待签名的字符串到自有服务器。
2. 在自有服务器进行签名并返回签名后的字符串到客户端。
   i. 按照OSS规定的签名算法对待签名字符串进行签名。有关签名算法的更多信息，请参见在Header中包含签名。

      签名算法的格式为 `signature = "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeySecret, content))` ，其中content是已根据请求参数拼接后的字符串。示例代码如下：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
credentialProvider = new OSSCustomSignerCredentialProvider() {
  @Override
  public String signContent(String content) {
  // 按照OSS规定的签名算法加签字符串，并将得到的加签字符串拼接AccessKeyId后返回。
  // 将加签的字符串传给您的服务器，然后返回签名。如果因某种原因加签失败，服务器描述错误信息后返回null。
    return "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeySecret, content));
  }
};
OSS oss = new OSSClient(getApplicationContext(), endpoint, credentialProvider);
```

   ii. 返回签名后的字符串给客户端。

3. 在客户端发送签名后的字符串到OSS服务器进行鉴权。

8.12. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理使用

- 匿名访问

```
String url = oss.presignPublicObjectURL(testBucket, testObject);
    OSSLog.logDebug("signPublicURL", "get url: " + url);
然后对生成的url追加x-oss-process:operation 的参数，operation代表的是图片处理操作
```

- 授权访问

  SDK中使用图片处理时，只需要在下载图片时调用 `request.setxOssProcess()` 方法设置处理参数。示例如下：

```
// 构造图片下载请求。
GetObjectRequest get = new GetObjectRequest("<bucketName>", "example.jpg");
// 图片处理。
request.setxOssProcess("image/resize,m_fixed,w_100,h_100");
OSSAsyncTask task = oss.asyncGetObject(get, new OSSCompletedCallback<GetObjectRequest, GetObject
Result>() {
  @Override
  public void onSuccess(GetObjectRequest request, GetObjectResult result) {
    // 请求成功。
    InputStream inputStream = result.getObjectContent();
    byte[] buffer = new byte[2048];
    int len;
    try {
      while ((len = inputStream.read(buffer)) != -1) {
        // 处理下载的数据。
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  @Override
  public void onFailure(GetObjectRequest request, ClientException clientExcepion, ServiceException servi
ceException) {
    // 处理异常，用户可自行填写。
  }
});
```

> **说明** 如需对图片进行其它处理，只需替换 `request.setxOssProcess()` 的相关参数。

- SDK访问

```
GetObjectRequest request = new GetObjectRequest("bucket-name", "image-name");
request.setxOssProcess("image/resize,m_lfit,w_100,h_100"); // 设置图片处理
OSSAsyncTask task = ossClient.asyncGetObject(request, getCallback);
```

图片处理持久化

以下代码用于图片处理持久化：

```
// fromBucket和toBucket分别表示源Bucket和目的Bucket名称。
// fromObjectKey和toObjectkey分别表示源Object和目标Object名称，其填写格式为指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
// action表示图片处理操作。
ImagePersistRequest request = new ImagePersistRequest(fromBucket,fromObjectKey,toBucket,toObjectkey,action);
    OSSAsyncTask task = mOss.asyncImagePersist(request, new OSSCompletedCallback<ImagePersistRequest, ImagePersistResult>() {
        @Override
        public void onSuccess(ImagePersistRequest request, ImagePersistResult result) {
          // sucess callback
        }
        @Override
        public void onFailure(ImagePersistRequest request, ClientException clientException, ServiceException serviceException) {
            // errror callback
        }
    });
```

图片处理工具

您可以通过可视化图片处理工具 ImageStyleViewer直观地看到OSS图片处理结果。

8.13. 异常处理

OSS Android SDK中有两种异常，分别为ClientException以及ServiceException 。

ClientException

ClientException指客户端尝试向OSS发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用，则会抛出ClientException。当上传文件时发生IO异常，也会抛出ClientException。

ServiceException

ServiceException指服务器端错误，来源于对服务器端错误信息的解析。OSSException包含OSS返回的错误码和错误信息，便于定位问题，并做出适当的处理。

ServiceException通常包含以下错误信息：

参数	描述
Code	OSS返回的错误码。
Message	OSS返回的详细错误信息。
RequestId	用于唯一标识该请求的UUID。您可以凭借此RequestId请求协助，排查并解决您遇到的问题。
HostId	用于标识访问的OSS集群，与请求时使用的Host一致。

参数	描述
rawMessage	HTTP响应的原始Body文本。

OSS常见错误码

有关OSS常见错误码汇总的更多信息，请参见错误响应。

9.Go

9.1. 前言

本文介绍对象存储OSS的Go SDK各种使用场景下的示例代码。

SDK源码和API文档

请访问Git Hub获取OSS Go SDK源码。更多信息请参见OSS Go SDK API文档。

示例程序

OSS Go SDK提供丰富的示例程序，方便您参考或直接使用。示例包括以下内容：

示例文件	示例内容
new_bucket.go	初始化Client
put_object.go	上传文件，包括简单上传、断点续传上传等
append_object.go	追加上传
get_object.go	下载文件，包括流式下载、限定条件下载、文件压缩下载等
delete_object.go	删除文件
copy_object.go	同一存储空间内拷贝文件、跨存储空间拷贝文件、限定条件拷贝
list_objects.go	列举文件，包括指定前缀的文件列举、指定个数的文件列举等
object_meta.go	设置和读取文件元信息
object_acl.go	设置和读取文件访问权限（ACL）
sign_url.go	生成带签名的URL
cname_sample.go	绑定自定义域名（CNAME）
create_bucket.go	创建存储空间
list_buckets.go	列举存储空间，包括默认参数列举和指定参数列举
bucket_acl.go	设置存储空间的访问权限（ACL）
bucket_referer.go	设置、读取、清除存储空间的防盗链
bucket_logging.go	设置、读取、清除存储空间的访问日志
bucket_lifecycle.go	设置、读取、清除文件的生命周期

示例文件	示例内容
bucket_cors.go	设置、读取、清除存储空间的跨域访问
object_tagging.go	设置、获取、删除对象标签，及对象标签和生命周期管理。
bucket_requestpayment.go	存储空间的请求者付费模式
bucket_encryption.go	设置、获取、删除Bucket加密

9.2. 安装

本文介绍如何安装Go SDK。

环境准备

- 环境要求

  使用Golang 1.6及以上版本。

  请参考Golang安装下载和安装Go编译运行环境。Go安装完毕后请新建系统变量GOPATH，并将其指向您的代码目录。要了解更多GOPATH相关信息，请执行命令 `go help gopath` 。

- 查看语言版本

  执行命令 `go version` 查看Go语言版本。

下载SDK

- 通过GitHub下载
- 历史版本下载

安装SDK

执行以下命令安装OSS Go SDK：

```
go get github.com/aliyun/aliyun-oss-go-sdk/oss
```

> **说明** 安装过程中，界面不会打印提示，请耐心等待。如果安装超时，请再次执行以上命令。

查看SDK版本

运行以下代码查看OSS Go SDK版本：

```
package main
import (
 "fmt"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 fmt.Println("OSS Go SDK Version: ", oss.Version)
}
```

9.3. 初始化

Client是OSS的Go客户端，用于管理存储空间和文件等OSS资源。

新建Client时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

可选的参数如下：

参数	说明	默认值
UseCname	是否使用自定义域名CNAME	否
Timeout	请求超时时间，包括连接超时、Socket读写超时，单位秒	连接超时30秒，读写超时60秒
SecurityToken	临时用户的SecurityToken	空
EnableMD5	是否开启MD5校验。推荐使用CRC校验，CRC的效率高于MD5	否
EnableCRC	是否开启CRC校验	是
Proxy	代理服务器，如 `http://8.8.8.8:3128`	空
AuthProxy	带账号密码的代理服务器	空

使用OSS域名新建Client

下面的代码用于使用OSS域名新建Client：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
```

使用自定义域名新建Client

自定义域名的完整代码请参见GitHub。

以下代码用于使用自定义域名新建Client：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
// oss.UseCname(true)为开启CNAME。CNAME是指将自定义域名绑定到存储空间上。
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>", oss.UseCname(tr
ue))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
```

注意 使用CNAME时，无法进行列举存储空间的操作，因为自定义域名已经绑定到某个特定的存储空间。

使用STS新建Client

以下代码用于使用STS新建一个Client：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>", oss.SecurityToken("<yourSecurityToken>"))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
```

更多信息请参见授权访问。

设置连接超时时间

以下代码用于设置连接超时时间：

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
// oss.Timeout(10, 120)表示设置HTTP连接超时时间为10秒（默认值为30秒），HTTP读写超时时间为120秒（默认值为60秒）。0表示永不超时（不推荐使用）。
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>", oss.Timeout(10, 120))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
```

CRC数据校验

上传、下载文件时默认开启CRC数据校验，确保上传、下载过程的数据完整性。以下代码用于关闭CRC数据校验：

警告 强烈建议您不要关闭CRC数据校验功能。如果您关闭此功能，则阿里云不保证上传、下载过程数据的完整性。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>", oss.EnableCRC(false))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
```

9.4. 快速入门

本节介绍如何快速使用OSS Go SDK完成常见操作，如创建存储空间（Bucket）、上传/下载文件（Object）等。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func handleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  endpoint := "http://oss-cn-hangzhou.aliyuncs.com"
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId := "<yourAccessKeyId>"
  accessKeySecret := "<yourAccessKeySecret>"
  bucketName := "<yourBucketName>"
  // 创建OSSClient实例。
  client, err := oss.New(endpoint, accessKeyId, accessKeySecret)
  if err != nil {
    handleError(err)
  }
  // 创建存储空间。
  err = client.CreateBucket(bucketName)
  if err != nil {
    handleError(err)
  }
}
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func handleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  endpoint := "http://oss-cn-hangzhou.aliyuncs.com"
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId := "<yourAccessKeyId>"
  accessKeySecret := "<yourAccessKeySecret>"
  bucketName := "<yourBucketName>"
  // <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
  objectName := "<yourObjectName>"
  // <yourLocalFileName>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。
  localFileName := "<yourLocalFileName>"
  // 创建OSSClient实例。
  client, err := oss.New(endpoint, accessKeyId, accessKeySecret)
  if err != nil {
    handleError(err)
  }
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    handleError(err)
  }
  // 上传文件。
  err = bucket.PutObjectFromFile(objectName, localFileName)
  if err != nil {
    handleError(err)
  }
}
```

下载文件

以下代码用于下载文件到本地：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func handleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  endpoint := "http://oss-cn-hangzhou.aliyuncs.com"
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId := "<yourAccessKeyId>"
  accessKeySecret := "<yourAccessKeySecret>"
  bucketName := "<yourBucketName>"
  // <yourObjectName>从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
  objectName := "<yourObjectName>"
  downloadedFileName := "<yourDownloadedFileName>"
  // 创建OSSClient实例。
  client, err := oss.New(endpoint, accessKeyId, accessKeySecret)
  if err != nil {
    handleError(err)
  }
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    handleError(err)
  }
  // 下载文件。
  err = bucket.GetObjectToFile(objectName, downloadedFileName)
  if err != nil {
    handleError(err)
  }
}
```

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 列举文件。
  marker := ""
  for {
    lsRes, err := bucket.ListObjects(oss.Marker(marker))
    if err != nil {
      HandleError(err)
    }
    // 打印列举文件，默认情况下一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println("Bucket: ", object.Key)
    }
    if lsRes.IsTruncated {
      marker = lsRes.NextMarker
    } else {
      break
    }
  }
}
```

删除文件

以下代码用于删除指定文件：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func handleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // Endpoint以杭州为例，其它Region请按实际情况填写。
  endpoint := "http://oss-cn-hangzhou.aliyuncs.com"
  // 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。
  accessKeyId := "<yourAccessKeyId>"
  accessKeySecret := "<yourAccessKeySecret>"
  bucketName := "<yourBucketName>"
  // <yourObjectName>表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
  objectName := "<yourObjectName>"
  // 创建OSSClient实例。
  client, err := oss.New(endpoint, accessKeyId, accessKeySecret)
  if err != nil {
    handleError(err)
  }
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    handleError(err)
  }
  // 删除文件。
  err = bucket.DeleteObject(objectName)
  if err != nil {
    handleError(err)
  }
}
```

删除文件详情请参见管理文件中的删除文件。

9.5. 存储空间

9.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

创建存储空间的完整代码请参见GitHub。

```
package main
  import (
    "fmt"
    "os"
    "github.com/aliyun/aliyun-oss-go-sdk/oss"
  )
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 创建存储空间（默认为标准存储类型），并设置存储空间的权限为公共读（默认为私有）。
  err = client.CreateBucket("<yourBucketName1>", oss.ACL(oss.ACLPublicRead))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 创建存储空间，并设置数据容灾类型为同城区域冗余存储。
  err = client.CreateBucket("<yourBucketName2>", oss.RedundancyType(oss.RedundancyZRS))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

以下代码用于创建归档或低频访问类型的存储空间：

```
// 创建归档类型的存储空间。如要创建低频访问类型存储空间，请将oss.StorageArchive替换为oss.StorageIA。
  err = client.CreateBucket("<yourBucketName>", oss.StorageClass(oss.StorageArchive))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
```

创建存储空间的更多详情，请参见Put Bucket。

9.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

列举存储空间的完整代码请参见Git Hub。

列举所有的存储空间

存储空间按照字母顺序排列。您可以列举所有的存储空间，或符合指定条件的存储空间。

以下代码用于列举所有存储空间。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 列举存储空间。
  marker := ""
  for {
    lsRes, err := client.ListBuckets(oss.Marker(marker))
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    // 默认情况下一次返回100条记录。
    for _, bucket := range lsRes.Buckets {
      fmt.Println("Bucket: ", bucket.Name)
    }
    if lsRes.IsTruncated {
      marker = lsRes.NextMarker
    } else {
      break
    }
  }
}
```

列举指定前缀的存储空间

以下代码用于列举以example为前缀（prefix）的存储空间。

```
// 列举指定前缀的存储空间。
  lsRes, err = client.ListBuckets(oss.Prefix("<yourBucketPrefix>"))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印存储空间列表。
  fmt.Println("Buckets with prefix: ", lsRes.Buckets)
  for _, bucket := range lsRes.Buckets {
    fmt.Println("Bucket with prefix: ", bucket.Name)
  }
```

列举指定marker之后的存储空间

以下代码用于列举examplebucket之后的存储空间。

```
// 列举指定marker之后的存储空间。
lsRes, err = client.ListBuckets(oss.Marker("<yourBucketMarker>"))
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
// 打印存储空间列表。
fmt.Println("My buckets with marker :", lsRes.Buckets)
for _, bucket := range lsRes.Buckets {
    fmt.Println("Bucket with marker: ", bucket.Name)
}
```

列举指定个数的存储空间

以下代码用于最多列举500个存储空间。

```
// 限定此次列举存储空间的个数为500。默认值为100，最大值为1000。
lsRes, err = client.ListBuckets(oss.MaxKeys(500))
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
// 打印存储空间列表。
fmt.Println("My buckets max num:", lsRes.Buckets)
for _, bucket := range lsRes.Buckets {
    fmt.Println("Bucket with maxKeys: ", bucket.Name)
}
```

9.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

以下代码用于判断存储空间examplebucket是否存在。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 判断存储空间是否存在。
  isExist, err := client.IsBucketExist("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("IsBucketExist result : ", isExist)
}
```

9.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间的地域。
  loc, err := client.GetBucketLocation("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Bucket Location:", loc)
}
```

有关地域的详细信息，请参见开发指南基本概念中的地域介绍以及GetBucketLocation API。

9.5.5. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期等。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间的信息，包括地域（Region或Location）、创建日期（CreationDate）、访问权限（ACL）、拥有者（Owner）、存储类型（StorageClass）、容灾类型（RedundancyType）等。
  res, err := client.GetBucketInfo("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("BucketInfo.Location: ", res.BucketInfo.Location)
  fmt.Println("BucketInfo.CreationDate: ", res.BucketInfo.CreationDate)
  fmt.Println("BucketInfo.ACL: ", res.BucketInfo.ACL)
  fmt.Println("BucketInfo.Owner: ", res.BucketInfo.Owner)
  fmt.Println("BucketInfo.StorageClass: ", res.BucketInfo.StorageClass)
  fmt.Println("BucketInfo.RedundancyType: ", res.BucketInfo.RedundancyType)
  fmt.Println("BucketInfo.ExtranetEndpoint: ", res.BucketInfo.ExtranetEndpoint)
  fmt.Println("BucketInfo.IntranetEndpoint: ", res.BucketInfo.IntranetEndpoint)
}
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

9.5.6. 管理存储空间的访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

更多关于访问权限的内容请参见访问控制。存储空间访问权限的完整代码请参见GitHub。

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有文件的读写权限，其他用户没有权限操作文件。	oss.ACLPrivate
公共读	存储空间的拥有者和授权用户有文件的读写权限，其他用户只有读权限。请谨慎使用该权限。	oss.ACLPublicRead
公共读写	所有用户都有文件的读写权限。请谨慎使用该权限。	oss.ACLPublicReadWrite

以下代码用于设置存储空间的访问权限：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置存储空间的访问权限为公共读。
  err = client.SetBucketACL("<yourBucketName>", oss.ACLPublicRead)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间的访问权限。
  aclRes, err := client.GetBucketACL("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Bucket ACL:", aclRes.ACL)
}
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

9.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 删除存储空间。
  err = client.DeleteBucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

删除存储空间的更多详情，请参见DeleteBucket。

9.5.8. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

> **说明**
>
> - 只有Bucket的拥有者及授权子账户才能为Bucket设置用户标签，否则返回403 Forbidden错误，错误码：AccessDenied。
> - 最多可设置20对Bucket用户标签（Key-Value对）。
> - Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
> - Value最大长度为128字节，可以为空。
> - Key和Value必须为UTF-8编码。
> - Put Bucket Tagging是覆盖语义，即新设置的标签会完全覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 初始化标签。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 // 设置存储空间的标签。
 err = client.SetBucketTagging("yourBucketName", tagging)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

获取Bucket标签

以下代码用于获取Bucket标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间的标签。
 ret, err := client.GetBucketTagging("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印标签个数。
 fmt.Println("Tag length: ", len(ret.Tags))
}
```

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 根据TagKey来查找Bucket。
 ret, err := client.ListBuckets(oss.TagKey("yourTaggingKey"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印Bucket信息。
 for _, bucket := range ret.Buckets{
  fmt.Println("bucket:", bucket)
 }
 // 根据TagKey和TagValue一起来查找Bucket。
 // TagValue参数必须和TagKey一起使用，可以不设定。不设定时，不过滤TagValue信息。
 res, err := client.ListBuckets(oss.TagKey("yourTaggingKey"), oss.TagValue("yourTaggingValue"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印Bucket信息。
 for _, b := range res.Buckets{
  fmt.Println("bucket:", b)
 }
}
```

删除Bucket标签

以下代码用于删除Bucket标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 删除标签。
 err = client.DeleteBucketTagging("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

9.5.9. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **说明** 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

> **说明**
> - 单个Bucket最多只能有1000条清单配置。
> - 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单（Inventory）配置：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bEnabled := true
```

```
    bEnabled := true
    // 如果清单需要加密，请参考以下代码。
    //var invEncryption oss.InvEncryption
    //var invSseOss oss.InvSseOss
    //var invSseKms oss.InvSseKms
    //invSseKms.KmsId = "<yourKmsId>"
    //invEncryption.SseOss = &invSseOss // 使用OSS完全托管加密（SSE-OSS）方式进行加密。
    //invEncryption.SseKms = &invSseKms // 使用KMS托管密钥（SSE-KMS）的方式进行加密。
    invConfig := oss.InventoryConfiguration{
      // 由用户指定的清单名称，清单名称在当前Bucket下必须全局唯一。
      Id: "<yourInventoryId2>",
      // 清单是否启用的标识。
      IsEnabled: &bEnabled,
      // 筛选规则的匹配前缀。
      Prefix: "<yourFilterPrefix>",
      OSSBucketDestination: oss.OSSBucketDestination{
        // 导出清单文件的文件格式。
        Format: "CSV",
        // 存储空间所有者授予的账户ID，比如109885487000****。
        AccountId: "<yourGrantAccountId>",
        // 存储空间所有者授予操作权限的角色名，比如acs:ram::109885487000****:role/ram-test。
        RoleArn: "<yourRoleArn>",
        // 存放导出的清单文件的存储空间。
        Bucket: "acs:oss:::" + "<yourDestBucketName>",
        // 清单文件的存储路径前缀。
        Prefix: "<yourDestPrefix>",
        // 如果清单需要加密，请参考以下代码。
        //Encryption:    &invEncryption,
      },
      // 清单文件导出的周期。
      Frequency: "Daily",
      // 是否在清单中包含Object的所有版本信息。
      IncludedObjectVersions: "All",
      OptionalFields: oss.OptionalFields{
        // 清单结果中包含的配置项。
        Field: []string{
          "Size", "LastModifiedDate", "ETag", "StorageClass", "IsMultipartUploaded", "EncryptionStatus",
        },
      },
    }
    err = client.SetBucketInventory("<yourBucketName>", invConfig)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
  }
```

有关添加存储空间清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
package main
import (
  "encoding/xml"
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  result, err := client.GetBucketInventory("<yourBucketName>", "<yourInventoryId>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印清单信息。
  bs, err := xml.MarshalIndent(result, "  ", "    ")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println(string(bs))
}
```

有关查看清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明** 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
package main
import (
  "encoding/xml"
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  var sumResult oss.ListInventoryConfigurationsResult
  vmarker := ""
  for {
    listResult, err := client.ListBucketInventory("<yourBucketName>", vmarker)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    sumResult.InventoryConfiguration = append(sumResult.InventoryConfiguration, listResult.InventoryCo
nfiguration...)
    if listResult.IsTruncated != nil && *listResult.IsTruncated {
      vmarker = listResult.NextContinuationToken
    } else {
      break
    }
  }
  // 打印所有清单信息。
  bs, err := xml.MarshalIndent(sumResult, "  ", "    ")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println(string(bs))
}
```

有关批量列举清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 删除清单配置。
  err = client.DeleteBucketInventory("<yourBucketName>", "<yourInventoryId>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

有关删除存储空间清单配置的详情，请参见DeleteBucketInventory。

9.5.10. 传输加速

传输加速可提升全球各地用户对OSS的访问速度，适用于远距离数据传输、GB或TB级大文件上传和下载的场景。

关于传输加速的更多信息，请参见开发指南中的传输加速。

开启传输加速

以下代码用于开启目标存储空间examplebucket的传输加速功能：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
 fmt.Println("Error:", err)
 os.Exit(-1)
}
func main() {
 // 创建OSSClient实例。
 // Endpoint为Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
 // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
 client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
 if err != nil {
  HandleError(err)
 }
 // 填写Bucket名称。
 bucketName := "examplebucket"
 // 开启Bucket的传输加速状态。
 // Enabled表示传输加速的开关，取值为true表示开启传输加速，取值为false表示关闭传输加速。
 accConfig := oss.TransferAccConfiguration{}
 accConfig.Enabled = true
 err = client.SetBucketTransferAcc(bucketName, accConfig)
 if err != nil {
  HandleError(err)
 }
 fmt.Printf("set bucket transfer accelerate success\n")
}
```

开启传输加速功能的接口信息，请参见PutBucketTransferAcceleration。

查询传输加速

以下代码用于查询目标存储空间examplebucket的传输加速状态：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
 fmt.Println("Error:", err)
 os.Exit(-1)
}
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
 if err != nil {
  HandleError(err)
 }
 // 填写Bucket名称。
 bucketName := "examplebucket"
 // 查询目标Bucket的传输加速状态。
 accConfig, err := client.GetBucketTransferAcc(bucketName)
 if err != nil {
  HandleError(err)
 }
 fmt.Printf("accConfigRes.Enabled:%T\n", accConfig.Enabled)
}
```

查询传输加速状态的接口信息，请参见GetBucketTransferAcceleration。

9.5.11. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 配置授权策略。
  policyConfig := `
  {
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect" : "Allow",
        "Resource" : ["acs:oss:*:*:*/user1/*"]
      }
    ],
    "Version": "1"
  }`
  // 设置bucket授权策略。
  err = client.SetBucketPolicy("<yourBucketName>", policyConfig)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("SetBucketPolicy success")
}
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间授权策略。
  strPolicy, err := client.GetBucketPolicy("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Bucket policy:", strPolicy)
}
```

有关获取Bucket Policy信息详情，请参见GetBucketPolicy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 删除存储空间授权策略。
  err = client.DeleteBucketPolicy("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("DeleteBucketPolicy success")
}
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

9.5.12. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

生命周期规则包含如下信息：

- 前缀或标签匹配策略：生命周期规则匹配的Object和碎片。
  - 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。
  - 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，OSS对所有拥有这些标签的对象执行生命周期规则。标签匹配不可以作用于碎片。

    说明　对象标签功能详情请参见对象标签。

  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。
  - 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。
- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Object成为非当前版本天数：指定一个过期天数N，并指定非当前版本Object过期后执行什么操作。Object会在其成为非当前版本的N天后过期，并执行指定的操作。

    说明　您可以将过期Object转换为低频访问类型或归档类型，也可以选择删除过期Object。详情请参见生命周期配置元素。

- 碎片过期策略：设置碎片的过期时间及操作。
  - 过期天数：可指定一个过期天数N，碎片会在其最后修改时间的N天后被删除。
  - 过期日期：指定一个过期日期，最后修改时间在该日期之前的碎片会被全部删除。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
```

```
    client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    // 设置生命周期规则1(id:"rule1", enable:true, prefix:"foo/", expiry:Days 3)，表示前缀为foo的文件距最后修改时间3天后过期。
    rule1 := oss.BuildLifecycleRuleByDays("rule1", "foo/", true, 3)
    // 在受版本控制状态下的object仅有删除标记的情况下，自动删除删除标记。
    deleteMark := true
    expiration := oss.LifecycleExpiration{
        ExpiredObjectDeleteMarker: &deleteMark,
    }
    // 非当前版本Object超过30天后过期删除。
    versionExpiration := oss.LifecycleVersionExpiration{
        NoncurrentDays: 30,
    }
    // 非当前版本Object超过10天后转为IA存储类型。
    versionTransition := oss.LifecycleVersionTransition{
        NoncurrentDays: 10,
        StorageClass:   "IA",
    }
    // 设置生命周期规则2。
    rule2 := oss.LifecycleRule{
        ID:             "rule2",
        Prefix:         "<yourObjectPrefix>",
        Status:         "Enabled",
        Expiration:     &expiration,
        NonVersionExpiration: &versionExpiration,
        NonVersionTransition: &versionTransition,
    }
    // 设置生命周期规则3，对标签键为tagA、标签值为A的文件，距文件最后修改时间3天后过期。
    rule3 := oss.LifecycleRule{
        ID:    "rule3",
        Prefix: "",
        Status: "Enabled",
        Tags: []oss.Tag{
            oss.Tag{
                Key:   "tagA",
                Value: "A",
            },
        },
        Expiration: &oss.LifecycleExpiration{Days: 3},
    }
    // 设置生命周期规则。
    rules := []oss.LifecycleRule{rule1, rule2, rule3}
    bucketName := "<yourBucketName>"
    err = client.SetBucketLifecycle(bucketName, rules)
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
}
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 查看生命周期规则。
  lcRes, err := client.GetBucketLifecycle(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Lifecycle Rules:", lcRes.Rules)
}
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 清空生命周期规则。
  err = client.DeleteBucketLifecycle(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
```

9.5.13. 请求者付费模式

阿里云对象存储OSS的请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

  如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

  数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 初始化请求者付费模式。
 reqPayConf := oss.RequestPaymentConfiguration{
  Payer: "Requester",
 }
 // 设置bucket请求者付费模式。
 err = client.SetBucketRequestPayment("<yourBucketName>", reqPayConf)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
  }
 // 获取bucket请求者付费模式。
 ret, err := client.GetBucketRequestPayment("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印bucket请求者付费模式。
 fmt.Println("Bucket request payer:", ret.Payer)
}
```

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、ListObjects、GetObject及DeleteObject为例，用于指定第三方付费访问Object。

其他用于指定第三方付费的Object读写操作接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
package main
import (
 "fmt"
 "os"
 "io/ioutil"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
 "strings"
)
func main() {
 // 创建OSSClient实例。
 // <Endpoint> 待访问bucket所在的域。
 // <yourAccessKeyId>  <yourAccessKeySecret> 外部访问者账户密码与密钥。
 payerClient, err := oss.New("<Endpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("New Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 // <bucketName> 该bucket拥有者开启了请求者付费模式，并授权给了外部访问者
```

```
  // <bucketName> 该bucket拥有者开启了请求者付费模式，并授权给了外部访问者。
  payerBucket, err := payerClient.Bucket("<bucketName>")
  if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
  }
   // 当bucket拥有者开启付费模式时，必须设置oss.RequestPayer(oss.Requester)才可以访问授权的内容。
   // 当bucket拥有者没有开启付费模式时，外部访问者可以不用携带oss.RequestPayer(oss.Requester)参数。
   // bucket付费模式设置，可参考SetBucketRequestPayment。
  // 外部访问者上传文件，该外部访问者已取得bucket相应权限。
  key := "youObjectName"
  err = payerBucket.PutObject(key, strings.NewReader("<objectValue>"), oss.RequestPayer("requester"))
  if err != nil {
   fmt.Println("put Error:", err)
   os.Exit(-1)
  }
  // 列举bucket下的所有object，该外部访问者已取得bucket相应权限。
  // 作为外部授权的访问者，必须设置oss.RequestPayer(oss.Requester)才可以访问授权的内容。
  lor, err := payerBucket.ListObjects(oss.RequestPayer(oss.Requester))
  if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
  }
  // 打印object名字
  for _, l := range lor.Objects {
   fmt.Println("the Key name is :", l.Key)
  }
  // 外部访问者下载文件，该外部访问者已取得bucket相应权限。
  body, err := payerBucket.GetObject(key, oss.RequestPayer(oss.Requester))
  if err != nil {
   fmt.Println("Get Error:", err)
   os.Exit(-1)
  }
  // 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
  defer body.Close()
  // 读取并打印获取的内容。
  data, err := ioutil.ReadAll(body)
  if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
  }
  fmt.Println("data:", string(data))
  // 外部访问者删除文件，该外部访问者已取得bucket相应权限。
  err = payerBucket.DeleteObject(key, oss.RequestPayer(oss.Requester))
  if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
  }
 }
```

9.5.14. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。设置防盗链的完整代码请参见GitHub。

设置防盗链

以下代码用于设置防盗链：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 添加Referer白名单，且不允许空Referer。Referer参数支持通配符星号（*）和问号（？）。
  referers := []string{"http://www.aliyun.com",
              "http://www.???.aliyuncs.com",
              "http://www.*.com"}
  err = client.SetBucketReferer(bucketName, referers, false)
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取防盗链信息。
  refRes, err := client.GetBucketReferer(bucketName)
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Referers: ", refRes.RefererList)
  fmt.Println("AllowEmptyReferer: ", refRes.AllowEmptyReferer)
}
```

清空防盗链

以下代码用于清空防盗链：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 清空防盗链。防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。
  err = client.SetBucketReferer(bucketName, []string{}, true)
  if err!=nil{
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.5.15. 访问日志

您可以开启存储空间的访问日志记录功能。开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

更多关于访问日志的介绍，请参见开发指南中的设置访问日志记录。访问日志的完整代码请参见GitHub。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  targetBucketName := "<yourTargetBucketName>"
  targetPrefix := "<yourTargetPrefix>"
  // 开启存储空间的访问日志记录。targetBucketName为存放日志文件的存储空间，targetPrefix为被保存的访问日志文件前缀，即日志文件存放的目录。
  err = client.SetBucketLogging(bucketName, targetBucketName, targetPrefix, true)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 查看存储空间的访问日志设置。
  logRes, err := client.GetBucketLogging(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Target Bucket: ", logRes.LoggingEnabled.TargetBucket)
  fmt.Println("Target Prefix: ", logRes.LoggingEnabled.TargetPrefix)
}
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 关闭访问日志记录。
  err = client.DeleteBucketLogging(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.5.16. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 设置静态网站托管： 索引页面为index.html, 错误页面为error.html。
  err = client.SetBucketWebsite(bucketName, "index.html", "error.html")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 查看静态网站托管配置。
  wsRes, err := client.GetBucketWebsite(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("indexWebsite: ", wsRes.IndexDocument.Suffix)
  fmt.Println("errorWebsite: ", wsRes.ErrorDocument.Key)
}
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 删除静态网站托管配置。
  err = client.DeleteBucketWebsite(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.5.17. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

跨域资源共享的完整代码请参见GitHub。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  rule1 := oss.CORSRule{
    AllowedOrigin: []string{"*"},
    AllowedMethod: []string{"PUT", "GET"},
    AllowedHeader: []string{},
    ExposeHeader:  []string{},
    MaxAgeSeconds: 200,
  }
  rule2 := oss.CORSRule{
    AllowedOrigin: []string{"http://www.a.com", "http://www.b.com"},
    AllowedMethod: []string{"POST"},
    AllowedHeader: []string{"Authorization"},
    ExposeHeader:  []string{"x-oss-test", "x-oss-test1"},
    MaxAgeSeconds: 100,
  }
  // 设置跨域资源共享规则。
  err = client.SetBucketCORS(bucketName, []oss.CORSRule{rule1, rule2})
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取跨域资源共享规则。
  corsRes, err := client.GetBucketCORS(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Bucket CORS:", corsRes.CORSRules)
}
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 删除跨域资源共享规则。
  err = client.DeleteBucketCORS(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.6. 上传文件

9.6.1. 概述

在OSS中，操作的基本数据单元是文件（Object）。OSS Go SDK提供了丰富的文件上传方式：

- 简单上传：文件最大不能超过5GB。
- 追加上传：文件最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。

9.6.2. 简单上传

本文介绍如何使用简单上传。

简单上传的完整代码请参见GitHub。

上传字符串

以下代码用于将字符串上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 指定存储类型为标准存储，缺省也为标准存储。
  storageType := oss.ObjectStorageClass(oss.StorageStandard)
  // 指定存储类型为归档存储。
  // storageType := oss.ObjectStorageClass(oss.StorageArchive)
  // 指定访问权限为公共读，缺省为继承bucket的权限。
  objectAcl := oss.ObjectACL(oss.ACLPublicRead)
  // 上传字符串。
  err = bucket.PutObject("<yourObjectName>", strings.NewReader("yourObjectValue"), storageType, object
Acl)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

上传Byte数组

以下代码用于上传Byte数组：

```
package main
import (
  "fmt"
  "os"
  "bytes"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 上传Byte数组。
  err = bucket.PutObject("<yourObjectName>", bytes.NewReader([]byte("yourObjectValueByteArrary")))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

上传文件流

以下代码用于上传文件流：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 读取本地文件。
  fd, err := os.Open("<yourLocalFile>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  defer fd.Close()
  // 上传文件流。
  err = bucket.PutObject("<yourObjectName>", fd)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

上传本地文件

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 上传本地文件。
  err = bucket.PutObjectFromFile("<yourObjectName>", "<yourLocalFile>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

示例代码

以下代码用于追加上传文件：

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  var nextPos int64 = 0
  // 第一次追加上传的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
  // <yourObjectName>填写不包含Bucket名称在内的Object的完整路径，例如example/test.txt。
  nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("YourObjectAppendValue1
"), nextPos)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 如果不是第一次追加上传，可以通过bucket.GetObjectDetailedMeta方法或上次追加返回值的X-Oss-Next-Appe
nd-Position的属性，获取追加位置。
  //props, err := bucket.GetObjectDetailedMeta("<yourObjectName>")
  //if err != nil {
  //   fmt.Println("Error:", err)
  //   os.Exit(-1)
  //}
  //nextPos, err = strconv.ParseInt(props.Get("X-Oss-Next-Append-Position"), 10, 64)
  //if err != nil {
  //   fmt.Println("Error:", err)
  //   os.Exit(-1)
  //}
  // 第二次追加上传。
  nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("YourObjectAppendValue2
"), nextPos)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 您可以进行多次追加上传操作。
}
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

9.6.4. 断点续传上传

通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录点。上传过程中，如果出现网络异常或程序崩溃导致文件上传失败时，将从断点记录处继续上传未上传完成的部分。

背景信息

使用断点续传上传时，文件上传的进度信息会记录在Checkpoint文件中，如果上传过程中某一分片上传失败，再次上传时会从Checkpoint文件中记录的点继续上传，从而达到断点续传的效果。上传完成后，Checkpoint文件会被删除。

> **注意**
> - SDK会将上传的状态信息记录在Checkpoint文件中，所以要确保程序对Checkpoint文件有写权限。请勿修改Checkpoint文件中携带的校验信息。如果Checkpoint文件损坏，则会重新上传所有分片。
> - 如果上传过程中本地文件发生了改变，则会重新上传所有分片。

您可以使用Bucket.UploadFile方法实现断点续传上传。可设置的参数如下：

参数	说明
objectKey	上传到OSS的文件名称。
filePath	待上传的本地文件路径。
partSize	上传的分片大小，取值范围为100 KB~5 GB。默认值为100 KB。
options	包含如下可选项： • Routines：指定分片上传的并发数。默认值是1，即不使用并发上传。 • Checkpoint：指定是否开启断点续传上传功能并设置Checkpoint文件。断点续传上传默认处于关闭状态。 例如：`oss.Checkpoint(true, "")` 表示开启断点续传上传功能，且Checkpoint文件设置为与本地文件同目录下的 `file.cp` ，其中 `file` 为本地文件名称。您也可以使用 `oss.Checkpoint(true, "your-cp-file.cp")` 指定Checkpoint文件。 **说明** 其它元信息请参见设置文件元信息。

代码实现

以下代码用于断点续传上传。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置分片大小为100 KB，指定分片上传并发数为3，并开启断点续传上传。
  // 其中<yourObjectName>与objectKey是同一概念，表示断点续传上传文件到OSS时需要指定包含文件后缀在内的
完整路径，例如abc/efg/123.jpg。
  // "LocalFile"为filePath，100*1024为partSize。
  err = bucket.UploadFile("<yourObjectName>", "LocalFile", 100*1024, oss.Routines(3), oss.Checkpoint(true,
""))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用Bucket.InitiateMultipartUpload方法返回OSS创建的全局唯一的uploadID。

2. 上传分片。

   调用Bucket.UploadPart方法上传分片数据。

> **说明**
>
> - 对于同一个uploadID，分片号（partNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，那么OSS上该分片已有的数据将会被覆盖。
> - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
> - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用Bucket.CompleteMultipartUpload方法将所有分片合并成完整的文件。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  objectName := "exampleobject.txt"
  // 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
  locaFilename := "D:\\localpath\\examplefile.txt"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 将本地文件分片，且分片数量指定为3。
  chunks, err := oss.SplitFileByPartNum(locaFilename, 3)
  fd, err := os.Open(locaFilename)
  defer fd.Close()
  // 设置存储类型为标准存储。
  storageType := oss.ObjectStorageClass(oss.StorageStandard)
  // 步骤1：初始化一个分片上传事件，并指定存储类型为标准存储。
  imur, err := bucket.InitiateMultipartUpload(objectName, storageType)
  // 步骤2：上传分片
```

```
    // 步骤2：上传分片。
    var parts []oss.UploadPart
    for _, chunk := range chunks {
        fd.Seek(chunk.Offset, os.SEEK_SET)
        // 调用UploadPart方法上传每个分片。
        part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number)
        if err != nil {
            fmt.Println("Error:", err)
            os.Exit(-1)
        }
        parts = append(parts, part)
    }
    // 指定Object的读写权限为公共读，默认为继承Bucket的读写权限。
    objectAcl := oss.ObjectACL(oss.ACLPublicRead)
    // 步骤3：完成分片上传，指定文件读写权限为公共读。
    cmur, err := bucket.CompleteMultipartUpload(imur, parts, objectAcl)
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    fmt.Println("cmur:", cmur)
}
```

取消分片上传事件

您可以调用bucket.AbortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用同一个uploadID执行任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-h
angzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  // 填写Bucket名称。
  bucket, err := client.Bucket("examplebucket")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 初始化一个分片上传事件。
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  imur, err := bucket.InitiateMultipartUpload("exampleobject.txt")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 取消分片上传。
  err = bucket.AbortMultipartUpload(imur)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

列举已上传的分片

以下代码用于列举某个分片上传事件中已经上传成功的分片：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-h
angzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
```

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
 // 填写Bucket名称。
bucketName := "examplebucket"
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
objectName := "exampleobject.txt"
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
locaFilename := "D:\\localpath\\examplefile.txt"
// 填写uploadID。
uploadID := "EBF2CAC46F1748B99376D795********"
// 获取存储空间。
bucket, err := client.Bucket(bucketName)
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
// 将本地文件分片，且分片数量指定为3。
chunks, err := oss.SplitFileByPartNum(locaFilename, 3)
fd, err := os.Open(locaFilename)
defer fd.Close()
// 初始化一个分片上传事件。
imur, err := bucket.InitiateMultipartUpload(objectName)
uploadID = imur.UploadID
fmt.Println("InitiateMultipartUpload UploadID: ", uploadID)
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
// 上传分片。
var parts []oss.UploadPart
for _, chunk := range chunks {
    fd.Seek(chunk.Offset, os.SEEK_SET)
    // 调用UploadPart方法上传每个分片。
    part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number)
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    fmt.Println("UploadPartNumber: ", part.PartNumber, ", ETag: ", part.ETag)
    parts = append(parts, part)
}
// 根据InitiateMultipartUploadResult列举已上传的分片。
lsRes, err := bucket.ListUploadedParts(imur)
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
}
// 打印已上传的分片。
fmt.Println("\nParts:", lsRes.UploadedParts)
for _, upload := range lsRes.UploadedParts {
```

```
    fmt.Println("List PartNumber:  ", upload.PartNumber, ", ETag: " ,upload.ETag, ", LastModified: ", upload.L
astModified)
  }
  // 根据objectName和uploadID生成InitiateMultipartUploadResult，然后列举所有已上传的分片。
  var imur_with_uploadid oss.InitiateMultipartUploadResult
  imur_with_uploadid.Key = objectName
  imur_with_uploadid.UploadID = uploadID
  // 列举已上传的分片。
  lsRes, err = bucket.ListUploadedParts(imur_with_uploadid)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印已上传的分片。
  fmt.Println("\nListUploadedParts by UploadID: ", uploadID)
  for _, upload := range lsRes.UploadedParts {
    fmt.Println("List PartNumber:  ", upload.PartNumber, ", ETag: " ,upload.ETag, ", LastModified: ", upload.L
astModified)
  }
  // 完成分片上传。
  cmur, err := bucket.CompleteMultipartUpload(imur, parts)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("cmur:", cmur)
}
```

列举分片上传事件

您可以使用 `Bucket.ListMultipartUploads` 方法列举所有执行中的分片上传事件，即已初始化但尚未完成或已取消的分片上传事件。可设置的参数请参见下表。

参数	说明
Delimiter	用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现Delimiter字符之间的Object作为一组元素。
MaxUploads	限定此次返回分片上传事件的最大数目，默认值和最大值均为1000。
KeyMarker	所有文件名称的字母序大于KeyMarker参数值的分片上传事件，可以与UploadIDMarker参数一同使用来指定返回结果的起始位置。
Prefix	限定返回的文件名称必须以指定的Prefix作为前缀。注意使用Prefix查询时，返回的文件名称中仍会包含Prefix。

参数	说明
UploadIDMarker	与KeyMarker参数一同使用来指定返回结果的起始位置。 ● 如果KeyMarker参数未设置，则OSS忽略该参数。 ● 如果KeyMarker参数被设置，查询结果中包含： ○ 所有Object名字的字典序大于KeyMarker参数值的分片上传事件。 ○ Object名字等于KeyMarker参数值，但是UploadID比UploadIDMarker参数值大的分片上传事件。

- 使用默认参数

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn
-hangzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常
运维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 列举所有分片上传事件。
  keyMarker := ""
  uploadIdMarker := ""
  for {
    // 默认情况一次最多返回1000条记录。
    lsRes, err := bucket.ListMultipartUploads(oss.KeyMarker(keyMarker), oss.UploadIDMarker(uploadIdMa
rker))
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    // 打印分片上传事件。
    for _, upload := range lsRes.Uploads {
      fmt.Println("Upload: ", upload.Key, ", UploadID: ",upload.UploadID)
    }
    if lsRes.IsTruncated {
      keyMarker = lsRes.NextKeyMarker
      uploadIdMarker = lsRes.NextUploadIDMarker
    } else {
      break
    }
  }
}
```

- 指定前缀为file

```
lsRes, err := bucket.ListMultipartUploads(oss.Prefix("file"))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

- 指定最多返回100条结果数据

```
lsRes, err := bucket.ListMultipartUploads(oss.MaxUploads(100))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

- 指定前缀为file且最多返回100条结果数据

```
lsRes, err := bucket.ListMultipartUploads(oss.Prefix("file"), oss.MaxUploads(100))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
fmt.Println("Uploads:", lsRes.Uploads)
```

9.6.6. 进度条

进度条用于指示上传或下载的进度。

下面的代码以Bucket.PutObjectFromFile方法为例，介绍如何使用进度条。

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
// 定义进度条监听器。
type OssProgressListener struct {
}
// 定义进度变更事件处理函数。
func (listener *OssProgressListener) ProgressChanged(event *oss.ProgressEvent) {
 switch event.EventType {
 case oss.TransferStartedEvent:
  fmt.Printf("Transfer Started, ConsumedBytes: %d, TotalBytes %d.\n",
   event.ConsumedBytes, event.TotalBytes)
 case oss.TransferDataEvent:
  fmt.Printf("\rTransfer Data, ConsumedBytes: %d, TotalBytes %d, %d%%.",
   event.ConsumedBytes, event.TotalBytes, event.ConsumedBytes*100/event.TotalBytes)
 case oss.TransferCompletedEvent:
  fmt.Printf("\nTransfer Completed, ConsumedBytes: %d, TotalBytes %d.\n",
   event.ConsumedBytes, event.TotalBytes)
 case oss.TransferFailedEvent:
  fmt.Printf("\nTransfer Failed, ConsumedBytes: %d, TotalBytes %d.\n",
   event.ConsumedBytes, event.TotalBytes)
 default:
 }
}
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 localFile := "<yourLocalFile>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 带进度条的上传。
 err = bucket.PutObjectFromFile(objectName, localFile, oss.Progress(&OssProgressListener{}))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

9.7. 下载文件

9.7.1. 概述

下载文件的完整示例代码请参见Git Hub。

OSS Go SDK提供了丰富的文件下载方式：

- 流式下载
- 下载到本地文件
- 断点续传下载
- 限定条件下载
- 文件压缩下载

下载过程中，您还可以通过进度条功能查看下载进度。

9.7.2. 流式下载

本文介绍如何使用流式下载。

下载文件到流

以下代码用于把指定的OSS文件下载到流：

```
package main
import (
 "fmt"
 "os"
 "io/ioutil"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 下载文件到流。
 body, err := bucket.GetObject("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
 defer body.Close()
 data, err := ioutil.ReadAll(body)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("data:", string(data))
}
```

下载文件到缓存

以下代码用于把指定的OSS文件下载到本地缓存：

```
package main
import (
 "fmt"
 "os"
 "io"
 "bytes"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 下载文件到缓存。
 body, err := bucket.GetObject("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer body.Close()
 buf := new(bytes.Buffer)
 io.Copy(buf, body)
 fmt.Println("buf:", buf)
}
```

下载文件到本地文件流

以下代码用于把指定的OSS文件下载到本地文件流：

```
package main
import (
 "fmt"
 "os"
 "io"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 下载文件到本地文件流。
 body, err := bucket.GetObject("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer body.Close()
 fd, err := os.OpenFile("LocalFile", os.O_WRONLY|os.O_CREATE, 0660)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 io.Copy(fd, body)
}
```

9.7.3. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

指定正常的下载范围

以下代码用于指定正常的下载范围来下载文件：

```
package main
import (
 "fmt"
 "os"
 "io/ioutil"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取15~35字节范围内的数据，包含15和35，共21个字节的数据。
 // 如果指定的范围无效（比如开始或结束位置的指定值为负数，或指定值大于文件大小），则下载整个文件。
 body, err := bucket.GetObject("<yourObjectName>", oss.Range(15, 35))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
 defer body.Close()
 data, err := ioutil.ReadAll(body)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("data:", string(data))
}
```

指定异常的下载范围

假设现有大小为1000 Bytes的Object，则指定的正常下载范围应为0~999。如果指定范围不在有效区间，会导致Range不生效，响应返回值为200，并传送整个Object的内容。请求不合法的示例及返回说明如下：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。

兼容行为范围下载

在请求中增加请求头x-oss-range-behavior:standard，则改变指定范围不在有效区间时OSS的下载行为。假设现有大小为1000 Bytes的Object：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。

- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回HTTP Code为416，错误码为InvalidRange。

以下代码用于兼容行为范围下载：

```
package main
import (
  "fmt"
  "io/ioutil"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 上传1000个字节内容。
  strContent := ""
  for i := 0; i < 100; i++ {
    strContent += "abcdefghij"
  }
  fmt.Printf("content len:%d\n", len(strContent))
  // 上传字符串。
  err = bucket.PutObject("<yourObjectName>", strings.NewReader(strContent))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  //若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code
为206。
  //若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回HTTP Code为416 ，错误码为Invalid
Range。
  body, err := bucket.GetObject("<yourObjectName>", oss.Range(500, 2000), oss.RangeBehavior("standard")
)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 数据读取完成后，获取的流必须关闭，否则会造成连接泄漏，导致请求无连接可用，程序无法正常工作。
  defer body.Close()
  data, err := ioutil.ReadAll(body)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
```

```
  if len(data) != 500 {
    fmt.Println("read data error, len:%d", len(data))
    os.Exit(-1)
  }
  fmt.Println("data:", string(data))
}
```

9.7.4. 下载到本地文件

本文介绍如何将存储空间（Bucket）中的文件（Object）下载到本地文件。

以下代码用于将examplebucket中exampledir目录下的exampleobject.txt下载到本地*D:\localpath*路径下的examplefile.txt。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 填写Bucket名称，例如examplebucket。
  bucket, err := client.Bucket("examplebucket")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 下载文件到本地文件，并保存到指定的本地路径中。如果指定的本地文件存在会覆盖，不存在则新建。
  // 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
  // 依次填写Object完整路径（例如exampledir/exampleobject.txt）和本地文件的完整路径(例如D:\\localpath\\examplefile.txt)。Object完整路径中不能包含Bucket名称。
  err = bucket.GetObjectToFile("exampledir/exampleobject.txt", "D:\\localpath\\examplefile.txt")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.7.5. 断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

在下载的过程中会在checkpoint文件中记录当前下载的进度信息，如果下载过程中某一分片下载失败，再次下载时会从checkpoint文件中记录的点继续下载，从而达到断点续传下载的效果。下载完成后，checkpoint文件会被删除。

> **说明**
>
> - SDK会将下载的中间状态信息记录在Checkpoint文件中，所以要确保程序对Checkpoint文件有写权限。Checkpoint携带了校验信息，请不要修改。如果Checkpoint文件损坏则会重新下载文件。
> - 如果下载过程中待下载的OSS文件发生了改变（ETag改变），或者part文件丢失或被修改，则会重新下载文件。

您可以使用 `Bucket.DownloadFile` 实现断点续传下载。可设置的参数如下：

参数	说明
objectKey	要下载的OSS文件名称。
filePath	下载到本地文件的路径。
partSize	下载分片大小，取值范围为1B~5GB。
options	可选项，包括： - Routines：指定分片下载的并发数。默认是1，即不使用并发下载。 - Checkpoint：指定是否开启断点续传下载功能以及设置Checkpoint文件。默认关闭断点续传下载功能。例如 `oss.Checkpoint(true, "")` ，表示开启断点续传下载功能，并且Checkpoint文件为与本地文件同目录下的 `file.cp` ，其中 `file` 是本地文件名称。您也可以使用 `oss.Checkpoint(true, "your-cp-file.cp")` 指定Checkpoint文件。 - 下载时限定条件，请参见限定条件下载。

以下代码用于断点续传下载：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 分片下载。3个协程并发下载分片，开启断点续传下载。
  // 其中"<yourObjectName>"为objectKey，表示从OSS断点下载文件到时需要指定包含文件后缀在内的完整路径，
例如abc/efg/123.jpg。
  // "LocalFile"为filePath，100*1024为partSize。
  err = bucket.DownloadFile("<yourObjectName>", "LocalFile", 100*1024, oss.Routines(3), oss.Checkpoint(tr
ue, ""))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.7.6. 限定条件下载

本文介绍如何使用限定条件下载OSS文件。

下载文件时，可以指定一个或多个限定条件。满足限定条件则下载，不满足则返回错误，不下载。可以使用的限定条件如下：

参数	描述	如何设置
IfModifiedSince	如果指定的时间早于实际修改时间，则正常传输文件，否则返回错误（304 Not modified）。	oss.IfModifiedSince
IfUnmodifiedSince	如果指定的时间等于或者晚于文件实际修改时间，则正常传输文件，否则返回错误（412 Precondition failed）。	oss.IfUnmodifiedSince

参数	描述	如何设置
IfMatch	如果指定的ETag和OSS文件的ETag匹配，则正常传输文件，否则返回错误（412 Precondition failed）。	oss.IfMatch
IfNoneMatch	如果指定的ETag和OSS文件的ETag不匹配，则正常传输文件，否则返回错误（304 Not modified）。	oss.IfNoneMatch

If-Modified-Since和If-Unmodified-Since可以同时存在。If-Match和If-None-Match可以同时存在。

ETag可以通过Bucket.GetObjectDetailedMeta方法获取。

以下代码用于限定条件下载：

```
package main
import (
  "fmt"
  "os"
  "time"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置时间限定条件。
  date := time.Date(2015, time.November, 10, 23, 0, 0, 0, time.UTC)
  // 限定条件不满足，不下载文件。
  err = bucket.GetObjectToFile("<yourObjectName>", "LocalFile", oss.IfModifiedSince(date))
  if err == nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 满足限定条件，下载文件。
  err = bucket.GetObjectToFile("<yourObjectName>", "LocalFile", oss.IfUnmodifiedSince(date))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.7.7. 文件压缩下载

本文介绍如何使用文件压缩下载。

文件可以压缩下载，目前支持GZIP压缩。Bucket.GetObject和Bucket.GetObjectToFile支持压缩功能。

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 文件压缩下载。
 err = bucket.GetObjectToFile("<yourObjectName>", "LocalFile.gzip", oss.AcceptEncoding("gzip"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

9.7.8. 进度条

进度条用于指示上传或下载的进度。本文以Bucket.GetObjectToFile方法为例介绍如何打印下载文件（Object）的进度条。

以下代码用于打印从examplebucket下载exampleobject.txt文件的进度条。

```
package main
import (
   "fmt"
   "os"
   "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
// 定义进度条监听器。
type OssProgressListener struct {
}
// 定义进度变更事件处理函数。
// TotalBytes表示下载文件所需的总字节数，ConsumedBytes表示下载文件已经消耗的字节数。
func (listener *OssProgressListener) ProgressChanged(event *oss.ProgressEvent) {
   switch event.EventType {
```

```
  case oss.TransferStartedEvent:
    fmt.Printf("Transfer Started, ConsumedBytes: %d, TotalBytes %d.\n",
      event.ConsumedBytes, event.TotalBytes)
  case oss.TransferDataEvent:
    if event.TotalBytes != 0 {
      fmt.Printf("\rTransfer Data, ConsumedBytes: %d, TotalBytes %d, %d%%.",
        event.ConsumedBytes, event.TotalBytes, event.ConsumedBytes*100/event.TotalBytes)
    }
  case oss.TransferCompletedEvent:
    fmt.Printf("\nTransfer Completed, ConsumedBytes: %d, TotalBytes %d.\n",
      event.ConsumedBytes, event.TotalBytes)
  case oss.TransferFailedEvent:
    fmt.Printf("\nTransfer Failed, ConsumedBytes: %d, TotalBytes %d.\n",
      event.ConsumedBytes, event.TotalBytes)
  default:
  }
}
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-h
angzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  // 填写Object完整路径。Object完整路径中不能包含Bucket名称。
  objectName := "exampleobject.txt"
  // 填写本地文件的完整路径。如果指定的本地文件存在会覆盖，不存在则新建。
  // 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
  localFile := "D:\\localpath\\examplefile.txt"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 带进度条的下载。
  err = bucket.GetObjectToFile(objectName, localFile, oss.Progress(&OssProgressListener{}))
  if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
  }
  fmt.Println("Transfer Completed.")
}
```

9.8. 管理文件

9.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 删除文件
- 拷贝文件
- 解冻归档文件
- 管理软链接

9.8.2. 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 判断文件是否存在。
  isExist, err := bucket.IsObjectExist("<yourObjectName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Exist:", isExist)
}
```

9.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件访问权限的完整示例代码请参见GitHub。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	oss.ACLDefault
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	oss.ACLPrivate
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	oss.ACLPublicRead
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	oss.PublicReadWrite

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

以下是设置并获取文件访问权限的完整代码：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置文件的访问权限。
  err = bucket.SetObjectACL("<yourObjectName>", oss.ACLPublicReadWrite)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取文件的访问权限。
  aclRes, err := bucket.GetObjectACL("<yourObjectName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Object ACL:", aclRes.ACL)
}
```

9.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息。本文介绍如何设置、修改和获取文件元信息。

有关文件元信息（Object Meta）的更多信息，请参见文件元信息。有关文件元信息完整代码的更多信息，请参见GitHub。

设置文件元信息

您可以在上传文件时设置文件元信息。可设置的文件元信息如下：

参数	说明
CacheControl	指定该文件被下载时的网页的缓存行为。
ContentDisposition	指定该文件被下载时的名称。

参数	说明
ContentEncoding	指定该文件被下载时的内容编码格式。
Expires	设置缓存过期时间，格式是格林威治时间（GMT）。
ServerSideEncryption	指定OSS创建文件时的服务器端加密编码算法。有效值：AES256。
ObjectACL	指定OSS创建的文件的访问权限。
Meta	自定义元信息，以 `X-Oss-Meta-` 为前缀的参数。

以下代码用于设置文件元信息：

```
package main
import (
  "fmt"
  "os"
  "time"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置文件元信息：过期时间为2049年1月10日 23:00:00 GMT，访问权限为公共读，自定义元信息为MyProp（取值
MyPropVal）。
  expires := time.Date(2049, time.January, 10, 23, 0, 0, 0, time.UTC)
  options := []oss.Option{
    oss.Expires(expires),
    oss.ObjectACL(oss.ACLPublicRead),
    oss.Meta("MyProp", "MyPropVal"),
  }
  // 使用数据流上传文件。
  err = bucket.PutObject("<yourObjectName>", strings.NewReader("MyObjectValue"), options...)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取文件元信息。
  props, err := bucket.GetObjectDetailedMeta("<yourObjectName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Object Meta:", props)
}
```

修改文件元信息

您可以一次修改一条或多条元信息，代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 一次修改一条元信息。
  err = bucket.SetObjectMeta(objectName, oss.Meta("MyMeta", "MyMetaValue1"))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 一次修改多条元信息。
  options := []oss.Option{
    oss.Meta("MyMeta", "MyMetaValue2"),
    oss.Meta("MyObjectLocation", "HangZhou"),
  }
  err = bucket.SetObjectMeta(objectName, options...)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取文件元信息。
  props, err := bucket.GetObjectDetailedMeta(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Object Meta:", props)
}
```

获取文件元信息

以下代码用于获取文件元信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取文件元信息。
  props, err := bucket.GetObjectDetailedMeta(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Object Meta:", props)
}
```

9.8.5. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何转换文件（Object）的存储类型。

有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

以下提供了详细的示例代码用于Object存储类型的相互转换。

- 以下代码用于将Object的存储类型从标准或低频访问转换为归档类型：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  //更改文件存储类型，例如修改为归档存储类型。
  _, err = bucket.CopyObject(objectName, objectName, oss.ObjectStorageClass(oss.StorageArchive))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

- 以下代码用于将Object的存储类型从归档转换为标准类型：

```
package main
import (
  "fmt"
  "os"
  "time"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 检查目标文件是否为归档类型。如果是，则需要先解冻才能更改存储类型。
  meta, err := bucket.GetObjectDetailedMeta(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("X-Oss-Storage-Class : ", meta.Get(oss.HTTPHeaderOssStorageClass))
  if meta.Get(oss.HTTPHeaderOssStorageClass) == string(oss.StorageArchive) {
    //解冻文件。
    err = bucket.RestoreObject(objectName)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    //等待解冻结束，预计1分钟。
    meta, err = bucket.GetObjectDetailedMeta(objectName)
    for meta.Get("X-Oss-Restore") == "ongoing-request=\"true\"" {
      fmt.Println("X-Oss-Restore:" + meta.Get("X-Oss-Restore"))
      time.Sleep(1 * time.Second)
      meta, err = bucket.GetObjectDetailedMeta(objectName)
    }
  }
  //更改已解冻文件的存储类型，例如修改为标准存储类型。
  _, err = bucket.CopyObject(objectName, objectName, oss.ObjectStorageClass(oss.StorageStandard))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

各种存储类型的存储费用介绍，请参见计量项和计费项的计量计费概述一节。

9.8.6. 列举文件

本文介绍如何列举指定存储空间下（Bucket）的所有文件（Object）、指定前缀的文件、指定目录下的文件和子目录等。

背景信息

您可以调用GetBucket (ListObjects)或GetBucketV2 (ListObjectsV2)接口，一次性列举某个Bucket下最多1000个Object。您可以通过指定参数实现多种列举功能，例如通过指定参数列举指定起始位置后的所有文件、列举指定目录下的文件和子目录以及列举结果分页。这两个接口的主要区别如下：

- 使用GetBucket (ListObjects)接口列举文件时，默认返回Owner信息。
- 使用GetBucketV2 (ListObjectsV2)接口列举文件时，需通过fetchOwner指定是否在返回结果中包含Owner信息。

> **说明** 对于开启版本控制的Bucket，建议使用GetBucketV2 (ListObjectsV2)接口列举文件。

以下分别介绍通过GetBucket (ListObjects)以及GetBucketV2 (ListObjectsV2)方法列举文件时涉及的参数说明。

- 通过GetBucket (ListObjects)方法列举文件

  GetBucket (ListObjects)涉及参数说明如下：

参数	说明
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。
prefix	限定返回的文件必须以prefix作为前缀。
maxKeys	限定此次列举文件的最大个数。默认值为100，最大值为1000。
marker	此次列举文件的起点。

  更多信息，请参见GetBucket (ListObjects)。

- 通过GetBucketV2 (ListObjectsV2)方法列举文件

  GetBucketV2 (ListObjectsV2)涉及参数说明如下：

参数	描述
prefix	限定返回的文件必须以prefix作为前缀。
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。
startAfter	此次列举文件的起点。
fetchOwner	指定是否在返回结果中包含Owner信息。 ○ true：表示返回结果中包含Owner信息。 ○ false：表示返回结果中不包含Owner信息。

更多信息，请参见GetBucketV2 (ListObjectsV2)。

简单列举文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定Bucket下的100个文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 列举所有文件。
  marker := ""
  for {
    lsRes, err := bucket.ListObjects(oss.Marker(marker))
    if err != nil {
      HandleError(err)
    }
    // 打印列举结果。默认情况下，一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println("Bucket: ", object.Key)
    }
    if lsRes.IsTruncated {
      marker = lsRes.NextMarker
    } else {
      break
    }
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定Bucket下的100个文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  continueToken := ""
  for {
    lsRes, err := bucket.ListObjectsV2(oss.ContinuationToken(continueToken))
    if err != nil {
      HandleError(err)
    }
    // 打印列举结果。默认情况下，一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageCla
ss)
    }
    if lsRes.IsTruncated {
      continueToken = lsRes.NextContinuationToken
    } else {
      break
    }
  }
}
```

列举指定个数的文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定个数文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 设置列举文件的最大个数，并列举文件。
  lsRes, err := bucket.ListObjects(oss.MaxKeys(200))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  fmt.Println("Objects:", lsRes.Objects)
  for _, object := range lsRes.Objects {
    fmt.Println("Object:", object.Key)
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定个数文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 通过MaxKeys设置文件列举的最大个数，并列举文件。
  lsRes, err := bucket.ListObjectsV2(oss.MaxKeys(200))
  if err != nil {
    HandleError(err)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  for _, object := range lsRes.Objects {
    fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageClas
s, object.Owner.ID, object.Owner.DisplayName)
  }
}
```

列举指定前缀的文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定前缀的文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 列举包含指定前缀的文件。默认列举100个文件。
  lsRes, err := bucket.ListObjects(oss.Prefix("my-object-"))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  fmt.Println("Objects:", lsRes.Objects)
  for _, object := range lsRes.Objects {
    fmt.Println("Object:", object.Key)
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定前缀的文件的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 通过Prefix参数设置列举的文件前缀为my-object-。
  lsRes, err := bucket.ListObjectsV2(oss.Prefix("my-object-"))
  if err != nil {
    HandleError(err)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  for _, object := range lsRes.Objects {
    fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageClas
s, object.Owner.ID, object.Owner.DisplayName)
  }
}
```

列举指定起始位置之后的文件

- 通过GetBucket (ListObjects)方法列举

  通过设置Marker参数指定列举的起始位置，OSS将返回Marker字典序后的所有文件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 指定列举Marker之后的文件。默认列举100个文件。
  lsRes, err := bucket.ListObjects(oss.Marker("my-object-xx"))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  fmt.Println("Objects:", lsRes.Objects)
  for _, object := range lsRes.Objects {
    fmt.Println("Object:", object.Key)
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过设置StartAfter参数可以指定列举的起始位置，OSS将返回StartAfter字典序后的所有文件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 指定列举StartAfter之后的文件。默认列举100个文件。
  lsRes, err := bucket.ListObjectsV2(oss.StartAfter("my-object-"))
  if err != nil {
    HandleError(err)
  }
  // 打印列举结果。默认情况下，一次返回100条记录。
  for _, object := range lsRes.Objects {
    fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageClas
s, object.Owner.ID, object.Owner.DisplayName)
  }
}
```

分页列举所有文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举分页列举指定Bucket下的所有文件。每页列举的文件个数通过MaxKeys指定。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 分页列举所有文件。每页列举100个。
  marker := oss.Marker("")
  for {
    lsRes, err := bucket.ListObjects(oss.MaxKeys(100), marker)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    marker = oss.Marker(lsRes.NextMarker)
    // 打印列举结果。默认情况下，一次返回100条记录。
    fmt.Println("Objects:", lsRes.Objects)
    if !lsRes.IsTruncated {
      break
    }
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举分页列举指定Bucket下的所有文件。每页列举的文件个数通过MaxKeys指定。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 分页列举所有文件。每页列举100个。
  continueToken := ""
  for {
    lsRes, err := bucket.ListObjectsV2(oss.MaxKeys(100), oss.ContinuationToken(continueToken))
    if err != nil {
      HandleError(err)
    }
    // 打印列举结果。默认情况下，一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageCla
ss)
    }
    if lsRes.IsTruncated {
      continueToken = lsRes.NextContinuationToken
    } else {
      break
    }
  }
}
```

分页列举指定前缀的文件

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法分页列举指定前缀的文件。每页列举的文件个数通过MaxKeys指定。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 分页列举包含指定前缀的文件。每页列举80个。
  prefix := oss.Prefix("my-object-")
  marker := oss.Marker("")
  for {
    lsRes, err := bucket.ListObjects(oss.MaxKeys(80), marker, prefix)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    prefix = oss.Prefix(lsRes.Prefix)
    marker = oss.Marker(lsRes.NextMarker)
    // 打印列举结果。默认情况下，一次返回100条记录。
    fmt.Println("Objects:", lsRes.Objects)
    if !lsRes.IsTruncated {
      break
    }
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法分页列举指定前缀的文件。每页列举的文件个数通过MaxKeys指定。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 分页列举指定前缀的文件。每页列举80个。
  prefix := "my-object-"
  continueToken := ""
  for {
    lsRes, err := bucket.ListObjectsV2(oss.Prefix(prefix), oss.MaxKeys(80), oss.ContinuationToken(continu
eToken))
    if err != nil {
      HandleError(err)
    }
    // 打印列举结果。默认情况下，一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageCla
ss)
    }
    if lsRes.IsTruncated {
      continueToken = lsRes.NextContinuationToken
    } else {
      break
    }
  }
}
```

列举指定目录下所有文件的信息

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定前缀（目录）下所有文件的信息，包括文件大小、文件最后修改时间以及文件名等。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 遍历文件。
  marker := oss.Marker("")
  prefix := oss.Prefix("<yourObjectPrefix>")
  for {
    lor, err := bucket.ListObjects(marker, prefix)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    for _, object := range lor.Objects {
      fmt.Printf("%s %d %s\n", object.LastModified, object.Size, object.Key)
    }
    prefix = oss.Prefix(lor.Prefix)
    marker = oss.Marker(lor.NextMarker)
    if !lor.IsTruncated {
      break
    }
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定前缀（目录）下所有文件的信息，包括文件大小、文件最后修改时间以及文件名等。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  continueToken := ""
  prefix := "my-object-"
  for {
    lsRes, err := bucket.ListObjectsV2(oss.Prefix(prefix), oss.ContinuationToken(continueToken))
    if err != nil {
      HandleError(err)
    }
    // 打印列举文件，默认情况下，一次返回100条记录。
    for _, object := range lsRes.Objects {
      fmt.Println(object.Key, object.Type, object.Size, object.ETag, object.LastModified, object.StorageCla
ss)
    }
    if lsRes.IsTruncated {
      continueToken = lsRes.NextContinuationToken
    } else {
      break
    }
  }
}
```

列举指定目录下所有子目录的信息

- 通过GetBucket (ListObjects)方法列举

  通过GetBucket (ListObjects)方法列举指定前缀（目录）下所有子目录信息的示例代码如下：

```
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  marker := oss.Marker("")
  prefix := oss.Prefix("<yourDirPrefix>")
  for {
    lor, err := bucket.ListObjects(marker, prefix, oss.Delimiter("/"))
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    for _, dirName := range lor.CommonPrefixes {
      fmt.Println(dirName)
    }
    prefix = oss.Prefix(lor.Prefix)
    marker = oss.Marker(lor.NextMarker)
    if !lor.IsTruncated {
      break
    }
  }
}
```

- 通过GetBucketV2 (ListObjectsV2)方法列举

  通过GetBucketV2 (ListObjectsV2)方法列举指定前缀（目录）下所有子目录信息的示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  continueToken := ""
  prefix := "<yourDirPrefix>"
  for {
    lsRes, err := bucket.ListObjectsV2(oss.Prefix(prefix), oss.ContinuationToken(continueToken), oss.Deli
miter("/"))
    if err != nil {
      HandleError(err)
    }
    for _, dirName := range lsRes.CommonPrefixes {
      fmt.Println(dirName)
    }
    if lsRes.IsTruncated {
      continueToken = lsRes.NextContinuationToken
    } else {
      break
    }
  }
}
```

列举文件并返回Owner信息

通过GetBucketV2 (ListObjectsV2)方法列举文件并返回Owner信息示例代码如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 获取Owner信息。
  lsRes, err := bucket.ListObjectsV2(oss.FetchOwner(true))
  if err != nil {
    HandleError(err)
  }
  // 打印列举文件，默认情况下一次返回100条记录。
  for _, object := range lsRes.Objects {
    fmt.Println(object.Key, object.Owner.ID, object.Owner.DisplayName)
  }
}
```

9.8.7. 删除文件

本文介绍如何删除文件。

> **警告** 请您谨慎使用删除操作，文件删除后将无法恢复。

删除文件的完整代码请参见GitHub。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 删除单个文件。objectName表示删除OSS文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
  // 如需删除文件夹，请将objectName设置为对应的文件夹名称。如果文件夹非空，则需要将文件夹下的所有object删除后才能删除该文件夹。
  err = bucket.DeleteObject(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

删除多个文件

您可以通过 `Bucket.DeleteObjects` 来删除多个文件，并通过 `DeleteObjectsQuiet` 参数来指定是否返回删除的结果。默认返回删除成功的文件。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 返回删除成功的文件。
  delRes, err := bucket.DeleteObjects([]string{"my-object-1", "my-object-2"})
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("Deleted Objects:", delRes.DeletedObjects)
  // 不返回删除的结果。
  _, err = bucket.DeleteObjects([]string{"my-object-3", "my-object-4"},
  oss.DeleteObjectsQuiet(true))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

删除指定前缀（prefix）的文件

以下代码用于删除指定前缀的文件：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 列举所有包含指定前缀的文件并删除。
  marker := oss.Marker("")
  prefix := oss.Prefix("<yourObjectPrefix>")
  count := 0
  for {
    lor, err := bucket.ListObjects(marker, prefix)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    objects := []string{}
    for _, object := range lor.Objects {
      objects = append(objects, object.Key)
    }
    delRes, err := bucket.DeleteObjects(objects, oss.DeleteObjectsQuiet(true))
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    if len(delRes.DeletedObjects) > 0 {
      fmt.Println("these objects deleted failure,", delRes.DeletedObjects)
      os.Exit(-1)
    }
    count += len(objects)
    prefix = oss.Prefix(lor.Prefix)
    marker = oss.Marker(lor.NextMarker)
    if !lor.IsTruncated {
      break
    }
  }
  fmt.Printf("success,total delete object count:%d\n", count)
}
```

9.8.8. 拷贝文件

本文介绍如何拷贝文件。

拷贝文件的完整代码请参见Git Hub。

同一存储空间内拷贝文件

以下代码用于在同一个存储空间内拷贝文件：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 destObjectName := "<yourDestObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 拷贝文件到同一个存储空间的另一个文件。
 _, err = bucket.CopyObject(objectName, destObjectName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

跨存储空间拷贝文件

您可以将其它存储空间的文件拷贝到当前存储空间，也可以将当前存储空间的文件拷贝到其它存储空间。代码如下：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 srcBucketName := "<yourSrcBucketName>"
 dstBucketName := "<yourDstBucketName>"
 srcObjectName := "<yourSrcObjectName>"
 dstObjectName := "<yourDstObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 从其它存储空间（srcBucketName）拷贝源文件（srcObjectName）到本存储空间。
 bucket.CopyObjectFrom(srcBucketName, srcObjectName, dstObjectName)
 if err != nil {
  fmt.Println("CopyObjectFrom Error:", err)
  os.Exit(-1)
 }
 // 从本存储空间拷贝源文件（srcObjectName）到其它存储空间（dstBucketName）。
 bucket.CopyObjectTo(dstBucketName, dstObjectName, srcObjectName)
 if err != nil {
  fmt.Println("CopyObjectTo Error:", err)
  os.Exit(-1)
 }
}
```

拷贝时处理文件元信息

您可以在拷贝文件时通过MetadataDirective参数来处理文件元信息。MetadataDirective的取值如下：

- oss.MetaCopy：默认值。目标文件的元信息与源文件的元信息相同，即拷贝源文件的元信息。
- oss.MetaReplace：使用指定的元信息覆盖源文件的元信息。

> **注意** oss.MetaReplace将覆盖源文件的全部元信息，且无法恢复，请谨慎操作。

MetadataDirective取值为oss.MetaReplace时，可以指定的元信息如下：

参数	说明
CacheControl	指定目标文件被下载时的网页的缓存行为。

参数	说明
ContentDisposition	指定目标文件被下载时的名称。
ContentEncoding	指定目标文件被下载时的内容编码格式。
Expires	设置缓存过期时间，格式是格林威治时间（GMT）。
ServerSideEncryption	指定OSS创建目标文件时的服务器端加密编码算法。有效值：AES256。
ObjectACL	指定OSS创建的目标文件的访问权限。
Meta	自定义元信息，以 `X-Oss-Meta-` 为前缀的参数。

以下代码用于在拷贝时处理文件元信息：

```
package main
import (
 "fmt"
 "os"
 "time"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 destObjectName := "<yourDestObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 指定目标文件的元信息。
 expires := time.Date(2049, time.January, 10, 23, 0, 0, 0, time.UTC)
 options := []oss.Option{
  oss.MetadataDirective(oss.MetaReplace),
  oss.Expires(expires),
  oss.ObjectACL(oss.ACLPublicRead),
  oss.Meta("MyMeta", "MyMetaValue")}
 // 使用指定的元信息覆盖源文件的元信息。
 _, err = bucket.CopyObject(objectName, destObjectName, options...)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

限定条件拷贝

拷贝文件时，可以指定一个或多个限定条件。满足限定条件则拷贝，不满足则返回错误，不拷贝。可以使用的限定条件如下：

参数	说明
CopySourceIfMatch	如果源文件的ETag和指定的ETag匹配，则执行拷贝操作；否则返回错误。
CopySourceIfNoneMatch	如果源文件的ETag和指定的ETag不匹配，则执行拷贝操作；否则返回错误。
CopySourceIfModifiedSince	如果指定的时间早于实际修改时间，则执行拷贝操作；否则返回错误。

参数	说明
CopySourceIfUnmodifiedSince	如果指定的时间等于或者晚于文件实际修改时间，则执行拷贝操作；否则返回错误。

CopySourceIfMatch和CopySourceIfNoneMatch可以同时存在。CopySourceIfModifiedSince和CopySourceIfUnmodifiedSince可以同时存在。

以下代码用于限定条件拷贝：

```
package main
import (
 "fmt"
 "os"
 "time"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // bucketName := "<yourBucketName>"
 // objectName := "<yourObjectName>"
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 destMatchObjectName := "<yourMatchDestObjectName>"
 destUnMatchObjectName := "<yourUnmatchDestObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 date := time.Date(2011, time.November, 10, 23, 0, 0, 0, time.UTC)
 // 满足限定条件，执行拷贝。
 _, err = bucket.CopyObject(objectName, destMatchObjectName, oss.CopySourceIfModifiedSince(date))
 if err != nil {
  fmt.Println("CopyObject CopySourceIfModifiedSince Error:", err)
 }
 // 不满足限定条件，不执行拷贝。
 _, err = bucket.CopyObject(objectName, destUnMatchObjectName, oss.CopySourceIfUnmodifiedSince(date
))
 if err != nil {
  fmt.Println("CopyObject CopySourceIfUnmodifiedSince Error:", err)
 }
}
```

9.8.9. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传、拷贝文件及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 指定是否覆盖同名文件。
  // 不指定oss.ForbidOverWrite时，默认覆盖同名Object。
  // 指定oss.ForbidOverWrite为false时，表示允许覆盖同名Object。
  // 指定oss.ForbidOverWrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
  forbidWrite := oss.ForbidOverWrite(true)
  // 上传字符串。
  err = bucket.PutObject("<yourObjectName>", strings.NewReader("yourObjectValue"), forbidWrite)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

简单上传的更多详情，请参见PutObject。

拷贝文件

以下代码用于拷贝文件时禁止覆盖同名文件：

拷贝文件的更多详情，请参见CopyObject。

```
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  destObjectName := "<yourDestObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 指定是否覆盖同名目标文件。
  // 不指定oss.ForbidOverWrite时，默认覆盖同名目标Object。
  // 指定oss.ForbidOverWrite为false时，表示允许覆盖同名目标Object。
  // 指定oss.ForbidOverWrite为true时，表示禁止覆盖同名目标Object，如果同名目标Object已存在，程序将报错。
  forbidWrite := oss.ForbidOverWrite(true)
  // 拷贝文件到同一个存储空间的另一个文件。
  _, err = bucket.CopyObject(objectName, destObjectName, forbidWrite)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  locaFilename := "<yourLocalFilename>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  chunks, err := oss.SplitFileByPartNum(locaFilename, 3)
  fd, err := os.Open(locaFilename)
  defer fd.Close()
  // 指定是否覆盖同名文件。
  // 不指定oss.ForbidOverWrite时，默认覆盖同名Object。
  // 指定oss.ForbidOverWrite为false时，表示允许覆盖同名Object。
  // 指定oss.ForbidOverWrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错。
  forbidWrite := oss.ForbidOverWrite(true)
  // 步骤1：初始化一个分片上传事件
  imur, err := bucket.InitiateMultipartUpload(objectName, forbidWrite)
  // 步骤2：上传分片。
  var parts []oss.UploadPart
  for _, chunk := range chunks {
    fd.Seek(chunk.Offset, os.SEEK_SET)
    // 对每个分片调用UploadPart方法上传。
    part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    parts = append(parts, part)
  }
  // 步骤3：完成分片上传，禁止覆盖同名文件
  cmur, err := bucket.CompleteMultipartUpload(imur, parts, forbidWrite)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("cmur:", cmur)
}
```

分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

9.8.10. 解冻文件

归档或冷归档类型的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档和冷归档文件。

> **说明** 非归档或冷归档类型的文件，不要调用RestoreObject方法。
>
> 归档及冷归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

解冻归档文件的完整代码请参见GitHub。

以下代码用于解冻归档文件：

```
package main
import (
  "fmt"
  "os"
  "time"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  localFilename := "<yourLocalFilename>"
  // 创建存储空间。
  err = client.CreateBucket(bucketName, oss.StorageClass(oss.StorageArchive))
  if err != nil {
    HandleError(err)
  }
  // 获取存储空间。
  archiveBucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 上传归档文件。
  var val = "More than just cloud."
```

```
    err = archiveBucket.PutObject(objectName, strings.NewReader(val))
    if err != nil {
        HandleError(err)
    }
    // 检查是否为归档类型文件。
    meta, err := archiveBucket.GetObjectDetailedMeta(objectName)
    if err != nil {
        HandleError(err)
    }
    fmt.Println("X-Oss-Storage-Class : ", meta.Get("X-Oss-Storage-Class"))
    if meta.Get("X-Oss-Storage-Class") == string(oss.StorageArchive) {
        // 解冻归档类型文件。
        err = archiveBucket.RestoreObject(objectName)
        if err != nil {
            HandleError(err)
        }
        // 等待解冻结束。
        meta, err = archiveBucket.GetObjectDetailedMeta(objectName)
        for meta.Get("X-Oss-Restore") == "ongoing-request=\"true\"" {
            fmt.Println("x-oss-restore:" + meta.Get("X-Oss-Restore"))
            time.Sleep(1000 * time.Second)
            meta, err = archiveBucket.GetObjectDetailedMeta(objectName)
        }
    }
    // 下载已解冻的文件。
    err = archiveBucket.GetObjectToFile(objectName, localFilename)
    if err != nil {
        HandleError(err)
    }
    // 再次解冻文件。
    err = archiveBucket.RestoreObject(objectName)
    fmt.Println("ArchiveSample completed")
}
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。
   - 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
   - 批量（Bulk）：表示5~12小时内完成解冻。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  var restoreConfig oss.RestoreConfiguration
  // 表示1小时内完成解冻。
  restoreConfig.Tier = string(oss.RestoreExpedited)
  // 表示2~5小时内完成解冻。
  //restoreConfig.Tier = string(oss.RestoreStandard)
  // 表示5~12小时内完成解冻。
  //restoreConfig.Tier = string(oss.RestoreBulk)
  // 解冻状态保持1天。
  restoreConfig.Days = 1
  err = bucket.RestoreObjectDetail("<yourObjectName>", restoreConfig)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.8.11. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 设置软链接名称。
  objectName := "<yourSymlinkName>"
  // 设置软链接指向的目标文件名称。
  targetObjectName := "<yourSymlinkTargetName>"
  // 获取存储空间名称。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 创建软链接。
  err = bucket.PutSymlink(objectName, targetObjectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 上传软链接指向的目标文件。
  err = bucket.PutObject(targetObjectName, strings.NewReader("target"))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取软链接指向的文件内容。
  meta, err := bucket.GetSymlink(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println(meta.Get(oss.HTTPHeaderOssSymlinkTarget))
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  // 获取软链接名称。
  objectName := "<yourSymlinkObjectName>"
  // 获取存储空间名称。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取软链接指向的目标文件名称。
  meta, err := bucket.GetSymlink(objectName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println(meta.Get(oss.HTTPHeaderOssSymlinkTarget))
}
```

获取软链接的详细信息请参见GetSymlink。

9.9. 版本控制

9.9.1. 管理版本控制

版本控制应用于存储空间（Bucket） 内的所有文件（Object）。

通过Object的版本管理，在错误覆盖或者删除Object后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。

版本控制的详情请参见开发指南的版本控制介绍。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 创建一个存储空间。
 err = client.CreateBucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 设置存储空间版本控制状态为Enabled或Suspended，此处以设置存储空间版本状态为Enabled为例。
 config := oss.VersioningConfig{Status: "Enabled"}
 err = client.SetBucketVersioning("<yourBucketName>", config)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

设置Bucket版本控制状态的详情请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间版本控制状态信息。
 ret, err := client.GetBucketVersioning("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印存储空间版本控制状态信息。
 fmt.Println("The bucket Versioning status:", ret.Status)
}
```

获取Bucket版本控制状态的详情请参见GetBucketVersioning。

列举Bucket中所有Object的版本信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 列举包含指定前缀的文件。
 ret, err := bucket.ListObjectVersions(oss.Prefix("yourObjectPrefix"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印文件信息。
 for _, object := range ret.ObjectVersions {
  fmt.Println("Object:", object)
 }
 // 打印删除标记。
 for _, marker := range ret.ObjectDeleteMarkers {
  fmt.Println("Delete Markers:", marker)
 }
}
```

列举Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息的详情请参见GetBucketVersions(ListObjectVersions)。

9.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本的ID为“null”。

以下代码用于简单上传：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 var retHeader http.Header
 // 上传字符串。用oss.GetResponseHeader获取返回header。
 err = bucket.PutObject("<yourObjectName>", strings.NewReader("yourObjectValue"), oss.GetResponseHea
der(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
}
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

> **说明**
>
> - 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
> - 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
> - 不支持对当前版本为非Appendable类型的Object（包括 Normal Object、Delete Marker等）执行AppendObject 操作。

以下代码用于追加上传：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
 var retHeader http.Header
 var nextPos int64 = 0
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("YourObjectAppendValue1"
), nextPos, oss.GetResponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", retHeader.Get("x-oss-version-id"))
 // 第二次追加，用oss.GetResponseHeader获取返回header。
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("YourObjectAppendValue2"
), nextPos, oss.GetResponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
 // 您可以进行多次Append。
}
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  locaFilename := "<yourLocalFilename>"
  // 用oss.GetResponseHeader获取返回header。
  var retHeader http.Header
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  chunks, err := oss.SplitFileByPartNum(locaFilename, 3)
  fd, err := os.Open(locaFilename)
  defer fd.Close()
  // 步骤1：初始化一个分片上传事件。
  imur, err := bucket.InitiateMultipartUpload(objectName)
  // 步骤2：上传分片。
  var parts []oss.UploadPart
  for _, chunk := range chunks {
    fd.Seek(chunk.Offset, os.SEEK_SET)
    // 对每个分片调用UploadPart方法上传。
    part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    parts = append(parts, part)
  }
  // 步骤3：完成分片上传。
  cmur, err := bucket.CompleteMultipartUpload(imur, parts, oss.GetResponseHeader(&retHeader))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  fmt.Println("cmur:", cmur)
  // 打印x-oss-version-id。
  fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
}
```

分片上传的详细信息请参见CompleteMultipartUpload。

9.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null“的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 var retHeader http.Header
 // 下载yourObjectVersionId版本的文件到缓存。
 _, err = bucket.GetObject("youObjectName", oss.VersionId("yourObjectVersionId"), oss.GetResponseHeade
r(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
}
```

下载文件的详细信息请参见GetObject。

9.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的versionId，此versionId将会在响应header 的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成versionId为”null“的版本，且会覆盖原先versionId为”null“的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 destObjectName := "<yourDestObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 拷贝指定版本文件到同一个存储空间的另一个文件，并获取版本信息。
 var retHeader http.Header
 _, err = bucket.CopyObject(objectName, destObjectName, oss.VersionId("yourObjectVersionId"), oss.GetRe
sponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印版本信息。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
 fmt.Println("x-oss-copy-source-version-id:", oss.GetCopySrcVersionId(retHeader))
}
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝Object的版本id：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "time"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "yourBucketName"
 objectName := "yourObjectName"
 fileName := "yourFileName"
 futureDate := time.Date(2049, time.January, 10, 23, 0, 0, 0, time.UTC)
 var retHeader http.Header
 chunks, err := oss.SplitFileByPartNum(fileName, 3)
 // 上传一个待拷贝大文件。
 err = bucket.PutObjectFromFile(objectName, fileName, oss.GetResponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 取得上传源文件的versionId。
 versionId := oss.GetVersionId(retHeader)
 options := []oss.Option{
  oss.Expires(futureDate), oss.Meta("my", "myprop"),
 }
 // 目标文件名。
```

```
 objectDest := objectName + "dest"
 var parts []oss.UploadPart
 // 设置拷贝文件参数。
 copyOptions := []oss.Option{
   oss.VersionId(versionId), oss.GetResponseHeader(&retHeader),
 }
 // 进行大文件拷贝。
 imu, err := bucket.InitiateMultipartUpload(objectDest, options...)
 for _, chunk := range chunks {
   part, err := bucket.UploadPartCopy(imu, bucketName, objectName, chunk.Offset, chunk.Size, (int)(chunk.
Number), copyOptions...)
   if err != nil {
     fmt.Println("Error:", err)
     os.Exit(-1)
   }
   parts = append(parts, part)
   // 打印x-oss-copy-source-version-id。
   fmt.Println("x-oss-copy-source-version-id:", oss.GetCopySrcVersionId(retHeader))
 }
 _, err = bucket.CompleteMultipartUpload(imu, parts, oss.GetResponseHeader(&retHeader))
 if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
}
```

分片拷贝的详细信息请参见UploadPartCopy。

9.9.5. 列举文件

当存储空间（Bucket）处于版本控制状态时，您可以列举该Bucket中包含的所有文件（Object）、指定前缀的文件、指定目录下的文件和子目录等。

场景说明

假设您有一个名为examplebucket的存储空间，存储空间内的文件结构如下所示：

```
examplebucket
  └──── fun
      └──── exampleobject.jpg
      └──── examplefile.txt
      └──── destfolder
          └──── image1.jpg
          └──── image2.png
  └──── srcfile.txt
  └──── oss.jpg
```

以下示例分别说明如何通过对examplebucket设置不同的列举条件，获取不同的返回结果。

有关列举文件涉及的各个参数的更多信息，请参见GetBucketVersions(ListObjectVersions)。

列举Bucket中所有Object的信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  // yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-h
angzhou.aliyuncs.com。
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运
维，请登录RAM控制台创建RAM用户。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 列举包括删除标记在内的所有Object。
  keyMarker := oss.KeyMarker("")
  // VersionIdMarker与KeyMarker参数一同使用，以指定列举的起点。
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(keyMarker,versionIdMarker)
    if err != nil {
      HandleError(err)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest",dirName.IsLatest)
    }
    // 查看删除标记的版本信息。
     for _, marker  := range lor.ObjectDeleteMarkers {
      fmt.Println(marker.VersionId)
      fmt.Println(marker.Key)
    }
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    keyMarker = oss.KeyMarker(lor.NextKeyMarker)
    versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    if !lor.IsTruncated {
      break
```

```
        break
    }
  }
}
```

有关Endpoint的更多信息，请参见访问域名和数据中心。有关Bucket命名规范的更多信息，请参见存储空间（Bucket）。

返回结果：

```
Versionid: CAEQChiBgIDHzNPEthciIDYzZGQ5M2VjOTU2ODRmMzA4ZWM4ODk0NjVmMWEx****
Key: fun/
Is Latest true
Versionid: CAEQChiBgID61tnEthciIDNmOGM2MTQ3ZjU1NTQ2MGFhYWJjNjQxMTQxZWZh****
Key: fun/destfolder/
Is Latest true
Versionid: CAEQChiBgMDczt3EthciIDEwYjliZmQ4MDNiMDQ2Njk4YWMxM2NhM2E5NzQ3****
Key: fun/destfolder/image1.jpg
Is Latest true
Versionid: CAEQChiBgIDszt3EthciIGU5OWU0ZTllMGY3NTRmMmU5NzVjZmJkYmE3ZWYy****
Key: fun/destfolder/image2.png
Is Latest true
Versionid: CAEQChiBgICditzEthciIDBiNTg1ZTZkMDRlMzRjNDdiMjRjMTBlOGUzMTM0****
Key: fun/examplefile.txt
Is Latest true
Versionid: CAEQChiBgMC.itzEthciIDEzNWVlYjNhYzNmMjQ4NWM5Nzc2NzllY2FiYmQ3****
Key: fun/exampleobject.jpg
Is Latest true
Versionid: CAEQChiBgIDMgdbEthciIDUyMGI0NmZlNThkODQwY2ZhNmZhNTQ1Njk4ZTdj****
Key: oss.jpg
Is Latest true
Versionid: CAEQChiBgICIgdbEthciIDdlM2Q1YjYxZDIyZDQyMzI4MTRkNzVmYzdiMTBh****
Key: srcfile.txt
Is Latest true
```

列举指定前缀Object的版本信息

以下代码用于列举指定前缀Object的版本信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 通过Prefix参数列举前缀为fun的Object。
  prefix := oss.Prefix("fun")
  keyMarker := oss.KeyMarker("")
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(prefix,keyMarker,versionIdMarker)
    if err != nil {
      HandleError(err)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest",dirName.IsLatest)
    }
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    keyMarker = oss.KeyMarker(lor.NextKeyMarker)
    versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    if !lor.IsTruncated {
      break
    }
  }
}
```

返回结果：

```
Versionid: CAEQChiBgIDHzNPEthciIDYzZGQ5M2VjOTU2ODRmMzA4ZWM4ODk0NjVmMWEx****
Key: fun/
Is Latest true
Versionid: CAEQChiBgID61tnEthciIDNmOGM2MTQ3ZjU1NTQ2MGFhYWJjNjQxMTQxZWZh****
Key: fun/destfolder/
Is Latest true
Versionid: CAEQChiBgMDczt3EthciIDEwYjliZmQ4MDNiMDQ2Njk4YWMxM2NhM2E5NzQ3****
Key: fun/destfolder/image1.jpg
Is Latest true
Versionid: CAEQChiBgIDszt3EthciIGU5OWU0ZTllMGY3NTRmMmU5NzVjZmJkYmE3ZWYy****
Key: fun/destfolder/image2.png
Is Latest true
Versionid: CAEQChiBgICditzEthciIDBiNTg1ZTZkMDRlMzRjNDdiMjRjMTBlOGUzMTM0****
Key: fun/examplefile.txt
Is Latest true
Versionid: CAEQChiBgMC.itzEthciIDEzNWVlYjNhYzNmMjQ4NWM5Nzc2NzllY2FiYmQ3****
Key: fun/exampleobject.jpg
Is Latest true
```

列举指定个数Object的版本信息

以下代码用于列举指定个数Object的版本信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 通过MaxKeys参数指定最多返回4个结果，按Object名称的字母序依次返回。
  maxkey := oss.MaxKeys(4)
  keyMarker := oss.KeyMarker("")
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(maxkey,keyMarker,versionIdMarker)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest",dirName.IsLatest)
    }
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    keyMarker = oss.KeyMarker(lor.NextKeyMarker)
    versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    if !lor.IsTruncated {
      break
    }
  }
}
```

返回结果：

```
Versionid: CAEQChiBgIDHzNPEthciIDYzZGQ5M2VjOTU2ODRmMzA4ZWM4ODk0NjVmMWEx****
Key: fun/
Is Latest true
Versionid: CAEQChiBgID61tnEthciIDNmOGM2MTQ3ZjU1NTQ2MGFhYWJjNjQxMTQxZWZh****
Key: fun/destfolder/
Is Latest true
Versionid: CAEQChiBgMDczt3EthciIDEwYjliZmQ4MDNiMDQ2Njk4YWMxM2NhM2E5NzQ3****
Key: fun/destfolder/image1.jpg
Is Latest true
Versionid: CAEQChiBgIDszt3EthciIGU5OWU0ZTllMGY3NTRmMmU5NzVjZmJkYmE3ZWYy****
Key: fun/destfolder/image2.png
Is Latest true
```

分页列举所有Object的版本信息

以下代码用于分页列举所有Object的版本信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 分页列举所有Object。
  keyMarker := oss.KeyMarker("")
  // 通过MaxKeys参数指定最多返回4个结果，按Object名称的字母序依次返回。
  maxkey := oss.MaxKeys(4)
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(maxkey,keyMarker,versionIdMarker)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest",dirName.IsLatest)
    }
    fmt.Println("---------------")
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    if lor.IsTruncated {
      keyMarker = oss.KeyMarker(lor.NextKeyMarker)
      versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    }else{
      break
    }
  }
}
```

返回结果：

```
Versionid: CAEQChiBgIDHzNPEthciIDYzZGQ5M2VjOTU2ODRmMzA4ZWM4ODk0NjVmMWEx****
Key: fun/
Is Latest true
Versionid: CAEQChiBgID61tnEthciIDNmOGM2MTQ3ZjU1NTQ2MGFhYWJjNjQxMTQxZWZh****
Key: fun/destfolder/
Is Latest true
Versionid: CAEQChiBgMDczt3EthciIDEwYjliZmQ4MDNiMDQ2Njk4YWMxM2NhM2E5NzQ3****
Key: fun/destfolder/image1.jpg
Is Latest true
Versionid: CAEQChiBgIDszt3EthciIGU5OWU0ZTllMGY3NTRmMmU5NzVjZmJkYmE3ZWYy****
Key: fun/destfolder/image2.png
Is Latest true
---------------
Versionid: CAEQChiBgICditzEthciIDBiNTg1ZTZkMDRlMzRjNDdiMjRjMTBlOGUzMTM0****
Key: fun/examplefile.txt
Is Latest true
Versionid: CAEQChiBgMC.itzEthciIDEzNWVlYjNhYzNmMjQ4NWM5Nzc2NzllY2FiYmQ3****
Key: fun/exampleobject.jpg
Is Latest true
Versionid: CAEQChiBgIDMgdbEthciIDUyMGI0NmZlNThkODQwY2ZhNmZhNTQ1Njk4ZTdj****
Key: oss.jpg
Is Latest true
Versionid: CAEQChiBgICIgdbEthciIDdlM2Q1YjYxZDIyZDQyMzI4MTRkNzVmYzdiMTBh****
Key: srcfile.txt
Is Latest true
---------------
```

模拟文件夹列举文件

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为Object。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举根目录下的Object的版本信息

  以下代码用于列举根目录下的Object的版本信息：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 通过指定delimiter为正斜线（/），实现列举根目录下的Object的版本信息以及文件夹名称。
  delimiter := oss.Delimiter("/")
  keyMarker := oss.KeyMarker("")
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(keyMarker,delimiter,versionIdMarker)
    if err != nil {
      HandleError(err)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest",dirName.IsLatest)
    }
    // 查看以正斜线（/）结尾的文件夹名称。
    for _,common_prefix := range lor.CommonPrefixes{
      fmt.Println("common_prefix:",common_prefix)
    }
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    if lor.IsTruncated {
      keyMarker = oss.KeyMarker(lor.NextKeyMarker)
      versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    }else{
      break
    }
  }
}
```

返回结果：

```
Versionid: CAEQChiBgIDMgdbEthciIDUyMGI0NmZlNThkODQwY2ZhNmZhNTQ1Njk4ZTdj****
Key: oss.jpg
Is Latest true
Versionid: CAEQChiBgICIgdbEthciIDdlM2Q1YjYxZDIyZDQyMzI4MTRkNzVmYzdiMTBh****
Key: srcfile.txt
Is Latest true
common_prefix: fun/
```

- 列举目录下的文件和子目录

  以下代码用于列举指定目录下的文件和子目录：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("yourEndpoint","yourAccessKeyId","yourAccessKeySecret")
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称。
  bucketName := "examplebucket"
  bucket,err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 设置Prefix参数来获取fun目录下的所有文件与文件夹，同时设置delimiter参数为正斜线（/）作为文件夹的分隔符。
  prefix := oss.Prefix("fun/")
  delimiter := oss.Delimiter("/")
  keyMarker := oss.KeyMarker("")
  versionIdMarker := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(prefix,delimiter,keyMarker,versionIdMarker)
    if err != nil {
      HandleError(err)
    }
    // 查看Object的版本信息。
    for _, dirName := range lor.ObjectVersions{
      fmt.Println("Versionid:",dirName.VersionId)
      fmt.Println("Key:",dirName.Key)
      fmt.Println("Is Latest:",dirName.IsLatest)
    }
    // 查看以正斜线（/）结尾的文件夹名称。
    for _,common_prefix := range lor.CommonPrefixes{
      fmt.Println("common_prefix:",common_prefix)
    }
    // 查看列举结果是否完整。如果结果不完整，则继续列举。如果结果已完整，则退出循环。
    if lor.IsTruncated {
      keyMarker = oss.KeyMarker(lor.NextKeyMarker)
      versionIdMarker = oss.VersionIdMarker(lor.NextVersionIdMarker)
    }else{
      break
    }
  }
}
```

返回结果：

```
Versionid: CAEQChiBgIDHzNPEthciIDYzZGQ5M2VjOTU2ODRmMzA4ZWM4ODk0NjVmMWEx****
Key: fun/
Is Latest: true
Versionid: CAEQChiBgICditzEthciIDBiNTg1ZTZkMDRlMzRjNDdiMjRjMTBlOGUzMTM0****
Key: fun/examplefile.txt
Is Latest: true
Versionid: CAEQChiBgMC.itzEthciIDEzNWVlYjNhYzNmMjQ4NWM5Nzc2NzllY2FiYmQ3****
Key: fun/exampleobject.jpg
Is Latest: true
common_prefix: fun/destfolder/
```

9.9.6. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object））以及指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

  如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

  `x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 指定versionId删除Object

  以下代码用于指定versionId对Object进行永久删除：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 指定versionId对Object进行永久删除。
 err = bucket.DeleteObject(key, oss.VersionId("yourObjectVersionId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

- 不指定versionId删除Object

  以下代码用于不指定versionId对Object进行临时删除：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 不指定versionId对Object进行临时删除，此操作将会为Object添加删除标记。
 err = bucket.DeleteObject("youObjectName", oss.GetResponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印删除标记信息。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
 fmt.Println("x-oss-delete-marker:", oss.GetDeleteMark(retHeader))
}
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object或删除标记的示例。

- 指定versionId删除多个Object

  以下代码用于指定versionId对多个Object进行永久删除：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 指定需永久删除Object的versionId。
 keyArray := []oss.DeleteObject{
  oss.DeleteObject{Key: "objectName1", VersionId:"objectVersionId1"},
  oss.DeleteObject{Key: "objectName2", VersionId:"objectVersionId2"},
 }
 // 删除指定版本的Object。
 ret, err := bucket.DeleteObjectVersions(keyArray)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印删除的Object信息。
 for _, object := range ret.DeletedObjectsDetail {
  fmt.Println("Key:", object.Key, "\n VersionId:", object.VersionId)
 }
}
```

- 指定versionId删除多个删除标记

  以下代码用于指定versionId对多个删除标记进行永久删除：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 指定Object的版本信息。
 keyArray := []oss.DeleteObject{
  oss.DeleteObject{Key: "objectName1", VersionId:"objectVersionId1"},
  oss.DeleteObject{Key: "objectName2", VersionId:"objectVersionId2"},
 }
 // 指定versionId对多个删除标记进行永久删除。
 ret, err := bucket.DeleteObjectVersions(keyArray)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印删除的Object以及删除标记。
 for _, object := range ret.DeletedObjectsDetail {
  fmt.Println("key:", object.Key, "\n versionId:", object.VersionId, "\n DeleteMarker:", object.DeleteMarker
, "\nDeleteMarkerVersionId", object.DeleteMarkerVersionId)
 }
}
```

- 不指定versionID删除多个Object

  以下代码用于不指定versionId对多个Object进行临时删除：

```
package main
import (
 "fmt"
 "net/http"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 不指定versionId对多个Object进行临时删除，此操作会为Object添加删除标记。
 keyArray := []oss.DeleteObject{
  oss.DeleteObject{Key: "objectName1"},
  oss.DeleteObject{Key: "objectName2"},
 }
 res, err := bucket.DeleteObjectVersions(keyArray)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印Object的删除标记。
 for _, object := range res.DeletedObjectsDetail {
  fmt.Println("key:", object.Key, "\n DeleteMarker:", object.DeleteMarker, "\nDeleteMarkerVersionId", obj
ect.DeleteMarkerVersionId)
 }
}
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

删除指定前缀（prefix）的文件

以下代码用于删除指定前缀的文件：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
   fmt.Println("Error:", err)
```

```
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 列举所有包含指定前缀的文件并删除这些文件。
  prefix := oss.Prefix("<yourObjectPrefix>")
  marker := oss.KeyMarker("")
  version := oss.VersionIdMarker("")
  for {
    lor, err := bucket.ListObjectVersions(marker, prefix, version)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    objects := []oss.DeleteObject{}
    for _, object := range lor.ObjectDeleteMarkers {
      objects = append(objects, oss.DeleteObject{
        Key:     object.Key,
        VersionId: object.VersionId,
      })
    }
    for _, object := range lor.ObjectVersions {
      objects = append(objects, oss.DeleteObject{
        Key:     object.Key,
        VersionId: object.VersionId,
      })
    }
    delRes, err := bucket.DeleteObjectVersions(objects, oss.DeleteObjectsQuiet(true))
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    if len(delRes.DeletedObjectsDetail) > 0 {
      fmt.Println("these objects deleted failure,", delRes.DeletedObjectsDetail)
      os.Exit(-1)
    }
    prefix = oss.Prefix(lor.Prefix)
    marker = oss.KeyMarker(lor.NextKeyMarker)
    version = oss.VersionIdMarker(lor.NextVersionIdMarker)
    if !lor.IsTruncated {
      break
    }
  }
  fmt.Printf("delete success\n")
}
```

9.9.7. 解冻文件

在受版本控制的存储空间（Bucket）中，Object的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
package main
import (
 "fmt"
 "os"
 "time"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
 fmt.Println("Error:", err)
 os.Exit(-1)
}
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  HandleError(err)
 }
 bucketName := "<yourBucketName>"
 objectName := "<yourObjectName>"
 localFilename := "<yourLocalFilename>"
 // 创建归档类型的存储空间。
 err = client.CreateBucket(bucketName, oss.StorageClass(oss.StorageArchive))
 if err != nil {
  HandleError(err)
 }
 // 设置存储空间为开启版本控制状态。
 config := oss.VersioningConfig{Status: "Enabled"}
 err = client.SetBucketVersioning(bucketName, config)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 archiveBucket, err := client.Bucket(bucketName)
 if err != nil {
  HandleError(err)
 }
 var retHeader http.Header
 // 上传字符串。用oss.GetResponseHeader获取返回header。
 err = archiveBucket.PutObject("<yourObjectName>", strings.NewReader("yourObjectValue"), oss.GetRespo
nseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取上传解冻文件的versionId。
 versionId := oss.GetVersionId(retHeader)
 // 检查是否为归档类型文件。
 meta, err := archiveBucket.GetObjectDetailedMeta(objectName, oss.VersionId(versionId))
```

```
  if err != nil {
    HandleError(err)
  }
  if meta.Get("X-Oss-Storage-Class") == string(oss.StorageArchive) {
    err = archiveBucket.RestoreObject(objectName, oss.VersionId(versionId))
    if err != nil {
      HandleError(err)
    }
    // 等待解冻结束。
    meta, err = archiveBucket.GetObjectDetailedMeta(objectName, oss.VersionId(versionId))
    for meta.Get("X-Oss-Restore") == "ongoing-request=\"true\"" {
      fmt.Println("x-oss-restore:" + meta.Get("X-Oss-Restore"))
      time.Sleep(1000 * time.Second)
      meta, err = archiveBucket.GetObjectDetailedMeta(objectName, oss.VersionId(versionId))
    }
  }
  // 下载已解冻的文件。
  err = archiveBucket.GetObjectToFile(objectName, localFilename, oss.VersionId(versionId))
  if err != nil {
    HandleError(err)
  }
}
```

解冻文件的详细信息请参见RestoreObject。

9.9.8. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的meta信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的meta信息。

以下代码用于获取文件元信息：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 objectName := "<yourObjectName>"
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取指定版本文件的部分元信息。
 props, err := bucket.GetObjectMeta(objectName, oss.VersionId("youObjectVersionId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("Object Meta:", props)
 // 获取指定版本文件的所有元信息。
 props, err := bucket.GetObjectDetailedMeta(objectName, oss.VersionId("youObjectVersionId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("Object Meta:", ret)
}
```

获取文件元信息的详情请参见HeadObject、GetObjectMeta。

9.9.9. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取一个存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 设置指定版本文件的访问权限。
 err = bucket.SetObjectACL("<yourObjectName>", oss.ACLPublicReadWrite, oss.VersionId("youObjectVersio
nId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取一个存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取指定版本文件的访问权限。
 aclRes, err := bucket.GetObjectACL("<yourObjectName>", oss.VersionId("youObjectVersionId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("Object ACL:", aclRes.ACL)
}
```

获取文件访问权限的详细信息请参考GetObjectACL。

9.9.10. 管理软链接

本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

您可以通过创建软链接指向TargetObject的当前版本。

> **说明** 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

软链接本身也可以有多个版本，每个不同的版本可以指向不同的TargetObject，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。

以下代码用于创建软链接：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 // 目标文件。
 objectName := "<yourSymlinkName>"
 // 目标文件的软链接。
 targetObjectName := "<yourSymlinkTargetName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 上传文件到目标文件。
 err = bucket.PutObject(targetObjectName, strings.NewReader("target"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 创建软链接，并获取版本信息。
 var retHeader http.Header
 err = bucket.PutSymlink(objectName, targetObjectName, oss.GetResponseHeader(&retHeader))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印x-oss-version-id。
 fmt.Println("x-oss-version-id:", oss.GetVersionId(retHeader))
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 bucketName := "<yourBucketName>"
 // 目标文件。
 objectName := "<yourSymlinkName>"
 // 获取存储空间。
 bucket, err := client.Bucket(bucketName)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取指定版本软链接指向的文件内容。
 meta, err := bucket.GetSymlink(objectName,oss.VersionId("youObjectVersionId"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印软链接。
 fmt.Println(meta.Get(oss.HTTPHeaderOssSymlinkTarget))
}
```

获取软链接的详细信息请参见GetSymlink。

9.10. 对象标签

9.10.1. 设置对象标签

本文介绍如何设置对象标签（Object Tagging）。

说明

上传Object时添加对象标签

- 简单上传时添加对象标签

  以下代码用于简单上传时添加对象标签：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 // 设置Object Tagging。
 err = bucket.PutObject("<yourObjectKey>", strings.NewReader("<yourObjectValue>"), oss.SetTagging(ta
gging))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

- 分片上传时添加对象标签

  以下代码用于分片上传时添加对象标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
```

```
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
// 获取存储空间。
bucket, err := client.Bucket("<yourBucketName>")
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
  // 设置Tagging规则。
tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
}
tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
}
tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
}
// 获取待上传的文件位置和名称。
fileName := "<youFileNameAndPath>"
// 将文件分成3个分片进行上传，具体分片数可以试文件大小而定。
chunks, err := oss.SplitFileByPartNum(fileName, 3)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
// 打开文件。
fd, err := os.Open(fileName)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
defer fd.Close()
// 初始化上传的文件，并设置tagging。
imur, err := bucket.InitiateMultipartUpload("<yourObjectName>", oss.SetTagging(tagging))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
// 分片上传。
var parts []oss.UploadPart
for _, chunk := range chunks {
  fd.Seek(chunk.Offset, os.SEEK_SET)
  part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  parts = append(parts, part)
}
_, err = bucket.CompleteMultipartUpload(imur, parts)
```

```
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

- 追加上传时添加对象标签

  以下代码用于追加上传时添加对象标签：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
  // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 var nextPos int64
 // 第一次上传可追加的文件。只有第一次调用AppendObject设置的标签才会生效，后续再调用AppendObject设置的标签不生效。
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("<yourObjectValue>"), nextPos, oss.SetTagging(tagging))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 第二次上传。
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.NewReader("<yourObjectAppandValue>"), nextPos)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

- 断点续传上传时添加对象标签

  以下代码用于断点续传上传时添加对象标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
  // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 // 将文件分成多个分片，每个分片大小为100KB，使用3个协程并发上传分片，上传时设置Object Tagging。
 err = bucket.UploadFile("<yourObjectName>", "<yourNeedPuttingFile>", 100*1024, oss.Routines(3), oss.
SetTagging(tagging))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

对已上传Object添加或更改对象标签

以下代码用于对已上传Object添加或更改对象标签：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 // 设置Object Tagging。
 err = bucket.PutObjectTagging("<yourObjectName>", tagging)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object、及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 以下代码用于简单拷贝1GB以下的Object时设置对象标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
  // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 // 复制object时只设置tagging参数，Object Tagging不生效。
 _, err = bucket.CopyObject("<yourSourceObjectName>", "<yourDirectObjectName>", oss.SetTagging(tag
ging))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 复制object时需同时设置TaggingReplace及tagging两个参数，Object Tagging才生效。
 _, err = bucket.CopyObject("<yourSourceObjectName>", "<yourDirectObjectName>", oss.SetTagging(tag
ging), oss.TaggingDirective(oss.TaggingReplace))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

- 以下代码用于分片拷贝1GB以上的Object时设置对象标签：

```
package main
import (
 "fmt"
 "os"
 "strings"
```

```
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
    // 设置Tagging规则。
  tag1 := oss.Tag{
    Key:   "key1",
    Value: "value1",
  }
  tag2 := oss.Tag{
    Key:   "key2",
    Value: "value2",
  }
  tagging := oss.Tagging{
    Tags: []oss.Tag{tag1, tag2},
  }
  // 上传一个object，用于分片拷贝。
  content := "this your object value"
  err = bucket.PutObject("<yourObjectName>", strings.NewReader(content))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 初始化上传的文件，文件名是<desObjectName>，并设置tagging。
  imur, err := bucket.InitiateMultipartUpload("<desObjectName>", oss.SetTagging(tagging))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // yourSourObject表示拷贝的源Object。将Object分成1个分片进行上传。
  part, err := bucket.UploadPartCopy(imur, "<yourBucketName>", "<yourSourObject>", 0, int64(len(conte
nt)), 1)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  parts := []oss.UploadPart{part}
  _, err = bucket.CompleteMultipartUpload(imur, parts)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

为软链接文件设置标签

以下代码用于为软链接文件设置标签：

```
package main
import (
 "fmt"
 "os"
 "strings"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
  // 设置Tagging规则。
 tag1 := oss.Tag{
  Key:   "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:   "key2",
  Value: "value2",
 }
 tagging := oss.Tagging{
  Tags: []oss.Tag{tag1, tag2},
 }
 err = bucket.PutObject("<yourObjectName>", strings.NewReader("<yourObjectValue>"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 err = bucket.PutSymlink("<yourObjectName>", "<yourObjectSymLinkName>", oss.SetTagging(tagging))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

9.10.2. 获取对象标签

本文介绍如何获取对象标签（Object Tagging）。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于获取对象标签信息：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取object的Tagging信息。
 taggingResult, err := bucket.GetObjectTagging("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Printf("Object Tagging: %v\n", taggingResult)
}
```

获取对象标签详情请参考GetObjectTagging。

9.10.3. 删除对象标签

本文介绍如何删除对象标签（Object Tagging）。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于删除对象标签：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 删除Object Tagging。
 err = bucket.DeleteObjectTagging("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

删除对象标签详情请参考DeleteObjectTagging。

9.10.4. 对象标签和生命周期管理

生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为生命周期规则生效的条件。

> **说明** 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于生命周期规则中添加标签匹配规则：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 设置lifecycle的规则id为"rule2"，应用在前缀为"one"的object上。
 // 该前缀的object将在3天后转为IA的存储类型，30天后转为archive存储类型。
 transitionIA := oss.LifecycleTransition{
  Days: 3,
  StorageClass: oss.StorageIA,
 }
 transitionArch := oss.LifecycleTransition{
  Days: 30,
  StorageClass: oss.StorageArchive,
 }
 // 设置Tagging。
 tag1 := oss.Tag{
  Key:  "key1",
  Value: "value1",
 }
 tag2 := oss.Tag{
  Key:  "key2",
  Value: "value2",
 }
 // 设置bucket的lifecycle，其中lifecycle规则中包含设置的tagging信息。
 rule1 := oss.LifecycleRule{
  ID: "rule1",
  Prefix: "one",
  Status: "Enabled",
  Transitions: []oss.LifecycleTransition{transitionIA, transitionArch},
  Tags:[]oss.Tag{tag1,tag2},
 }
 rules := []oss.LifecycleRule{rule1}
 err = client.SetBucketLifecycle("<yourBucketName>", rules)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取bucket的lifecycle。
 lc, err := client.GetBucketLifecycle("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印lifecycle信息，其中包含tagging信息。
 for i, v := range lc.Rules {
  fmt.Printf("Bucket %d Lifecycle: %v", i, v)
  if v.Expiration != nil {
   fmt.Printf(", Expiration:%v", *v.Expiration)
  }
  fmt.Printf("\n")
 }
}
```

9.11. 数据加密

9.11.1. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

  上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

  > **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

注意

- 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
- 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
- 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 初始化一个加密规则，加密方式以AES256为例。
 config := oss.ServerEncryptionRule{SSEDefault: oss.SSEDefaultRule{SSEAlgorithm: "AES256"}}
 err = client.SetBucketEncryption("yourBucketName", config)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
package main
import (
 "fmt"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取加密规则。
 ret, err := client.GetBucketEncryption("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打印加密规则。
 fmt.Println("Encrypt rule", ret.SSEDefault.SSEAlgorithm)
}
```

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
package main
import (
 "fmt"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 删除加密规则。
 err = client.DeleteBucketEncryption("yourBucketName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

9.11.2. 客户端加密

客户端加密是指将数据发送到OSS之前在用户本地进行加密。

免责声明

- 使用客户端加密功能时，您需要对主密钥的完整性和正确性负责。因您维护不当导致主密钥用错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。
- 在对加密数据进行复制或者迁移时，您需要对加密元信息的完整性和正确性负责。因您维护不当导致加密元信息出错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。

背景信息

使用客户端加密时，会为每个Object生成一个随机数据加密密钥，用该随机数据加密密钥明文对Object的数据进行对称加密。主密钥用于生成随机的数据加密密钥，加密后的内容会当作Object的meta信息保存在服务端。解密时先用主密钥将加密后的随机密钥解密出来，再用解密出来的随机数据加密密钥明文解密Object的数据。主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。

> **注意**
>
> - 客户端加密支持分片上传超过5 GB的文件。在使用分片方式上传文件时，需要指定上传文件的总大小和分片大小， 除了最后一个分片外，每个分片的大小要一致，且分片大小目前必须是16的整数倍。
> - 调用客户端加密上传文件后，加密元数据会被保护，无法通过CopyObject修改Object meta信息。

加密方式

对于主密钥的使用，目前支持如下两种方式：

- 使用KMS托管用户主密钥

  当使用KMS托管用户主密钥用于客户端数据加密时，需要将KMS用户主密钥ID（即CMK ID）传递给SDK。

- 使用用户自主管理的主密钥（RSA）

  主密钥信息由用户提供，需要用户将主密钥的公钥、私钥信息当做参数传递给SDK。

使用以上两种加密方式能够有效地避免数据泄漏，保护客户端数据安全。即使数据泄漏，其他人也无法解密得到原始数据。

加密元信息

参数	描述	是否必需
x-oss-meta-client-side-encryption-key	加密后的密钥。 经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-start	随机产生的用于加密数据的初始值 。经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-cek-alg	数据的加密算法。	是
x-oss-meta-client-side-encryption-wrap-alg	数据密钥的加密算法。	是

参数	描述	是否必需
x-oss-meta-client-side-encryption-matdesc	主密钥的描述信息。JSON格式。 **警告** 强烈建议为每个主密钥都配置描述信息，并保存好主密钥和描述信息之间的对应关系。否则不支持更换主密钥进行加密。	否
x-oss-meta-client-side-encryption-unencrypted-content-length	加密前的数据长度。如未指定content-length则不生成该参数。	否
x-oss-meta-client-side-encryption-unencrypted-content-md5	加密前数据的MD5。如未指定MD5，则不生成该参数。	否
x-oss-meta-client-side-encryption-data-size	若加密Multipart文件需要在init_multipart时传入整个大文件的总大小。	是（分片上传）
x-oss-meta-client-side-encryption-part-size	若加密Multipart文件需要在init_multipart时传入分片大小。 **说明** 目前分片大小必须是16的整数倍。	是（分片上传）

使用主密钥RSA普通上传和下载Object

使用主密钥RSA普通上传和下载Object示例代码如下：

```
package main
import (
 "bytes"
 "fmt"
 "io/ioutil"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
 "github.com/aliyun/aliyun-oss-go-sdk/oss/crypto"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 创建一个主密钥的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
 // 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
 // 如果主密钥描述信息为空，解密时无法判断使用的是哪个主密钥。
 // 强烈建议为每个主密钥都配置主密钥描述信息(json字符串), 由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
 // 由主密钥描述信息(json字符串)转换的map
```

```
  // 由主密钥描述信息(json字符串)转换的map。
  materialDesc := make(map[string]string)
 materialDesc["desc"] = "<your master encrypt key material describe information>"
 // 根据主密钥描述信息创建一个主密钥对象。
 masterRsaCipher, err := osscrypto.CreateMasterRsa(materialDesc, "<your rsa public key>", "<your rsa priva
te key>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 根据主密钥对象创建一个用于加密的接口, 使用aes ctr模式加密。
 contentProvider := osscrypto.CreateAesCtrCipher(masterRsaCipher)
 // 获取一个用于客户端加密的已创建bucket。
 // 客户端加密bucket和普通bucket具有相似的用法。
 cryptoBucket, err := osscrypto.GetCryptoBucket(client, "<yourBucketName>", contentProvider)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // put object时自动加密。
 err = cryptoBucket.PutObject("<yourObjectName>", bytes.NewReader([]byte("yourObjectValueByteArrary
")))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // get object时自动解密。
 body, err := cryptoBucket.GetObject("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer body.Close()
 data, err := ioutil.ReadAll(body)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("data:", string(data))
}
```

使用主密钥RSA分片上传Object

使用主密钥RSA分片上传Object示例代码如下：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
 "github.com/aliyun/aliyun-oss-go-sdk/oss/crypto"
)
func main() {
 // 创建OSSClient实例
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
```

```
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
// 创建一个主密钥的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
// 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
// 如果主密钥描述信息为空，解密时无法判断使用的是哪个主密钥。
// 强烈建议为每个主密钥都配置主密钥描述信息(json字符串)，由客户端保存主密钥和描述信息之间的对应关系（服务端不保存两者之间的对应关系）。
  // 由主密钥描述信息(json字符串)转换的map。
materialDesc := make(map[string]string)
materialDesc["desc"] = "<your master encrypt key material describe information>"
// 根据主密钥描述信息创建一个主密钥对象。
masterRsaCipher, err := osscrypto.CreateMasterRsa(materialDesc, "<your rsa public key>", "<your rsa private key>")
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
// 根据主密钥对象创建一个用于加密的接口，使用aes ctr模式加密。
contentProvider := osscrypto.CreateAesCtrCipher(masterRsaCipher)
// 获取一个用于客户端加密的已创建bucket。
// 客户端加密bucket和普通bucket具有相似的用法。
cryptoBucket, err := osscrypto.GetCryptoBucket(client, "<yourBucketName>", contentProvider)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
fileName := "<yourLocalFilePath>"
fileInfo, err := os.Stat(fileName)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
fileSize := fileInfo.Size()
// 加密上下文信息。
var cryptoContext osscrypto.PartCryptoContext
cryptoContext.DataSize = fileSize
// 期望的分片数，实际分片数以后续计算出来的为准。
expectPartCount := int64(10)
// 目前aes ctr加密分片大小需16个字节对齐。
cryptoContext.PartSize = (fileSize / expectPartCount / 16) * 16
imur, err := cryptoBucket.InitiateMultipartUpload("<yourObjectName>", &cryptoContext)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
chunks, err := oss.SplitFileByPartSize(fileName, cryptoContext.PartSize)
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
var partsUpload []oss.UploadPart
for _, chunk := range chunks {
```

```
  for _, chunk := range chunks {
    part, err := cryptoBucket.UploadPartFromFile(imur, fileName, chunk.Offset, chunk.Size, (int)(chunk.Numb
er), cryptoContext)
    if err != nil {
      fmt.Println("Error:", err)
      os.Exit(-1)
    }
    partsUpload = append(partsUpload, part)
  }
  // Complete
  _, err = cryptoBucket.CompleteMultipartUpload(imur, partsUpload)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

解密使用主密钥为不同的RSA加密的Object

解密使用主密钥为不同的RSA加密的Object示例代码如下：

```
package main
import (
  "bytes"
  "fmt"
  "io/ioutil"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
  "github.com/aliyun/aliyun-oss-go-sdk/oss/crypto"
)
// 根据主密钥描述信息查询主密钥。如果需要解密不同的主密钥加密的object，需要提供此接口。
type MockRsaManager struct {
}
func (mg *MockRsaManager) GetMasterKey(matDesc map[string]string) ([]string, error) {
  keyList := []string{"<yourRsaPublicKey", "yourRsaPrivatKey"}
  return keyList, nil
}
// 解密不同主密钥加密的object。
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 创建一个主密钥的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
  // 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
  // 如果主密钥描述信息为空，解密时无法判断使用的是哪个主密钥。
  // 强烈建议为每个主密钥都配置主密钥描述信息(json字符串)，由客户端保存主密钥和描述信息之间的对应关系（服务
端不保存两者之间的对应关系）。
    // 由主密钥描述信息(json字符串)转换的map。
  materialDesc := make(map[string]string)
  materialDesc["desc"] = "<your master encrypt key material describe information>"
  // 根据主密钥描述信息创建一个主密钥对象。
  masterRsaCipher, err := osscrypto.CreateMasterRsa(materialDesc, "<your rsa public key>", "<your rsa priva
```

```
 masterRsaCipher, err := osscrypto.CreateMasterRsa(materialDesc, "<yourRsaPublicKey>", "<yourRsaPrivate key>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 根据主密钥对象创建一个用于加密的接口，使用aes ctr模式加密。
 contentProvider := osscrypto.CreateAesCtrCipher(masterRsaCipher)
 // 如果需要解密不同主密钥加密的object，需要提供此接口。
 var mockRsaManager MockRsaManager
 var options []osscrypto.CryptoBucketOption
 options = append(options, osscrypto.SetMasterCipherManager(&mockRsaManager))
 // 获取一个用于客户端加密的已创建bucket。
 // 客户端加密bucket和普通bucket具有相似的用法。
 cryptoBucket, err := osscrypto.GetCryptoBucket(client, "<yourBucketName>", contentProvider, options...)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // put object时自动加密。
 err = cryptoBucket.PutObject("<yourObjectName>", bytes.NewReader([]byte("yourObjectValueByteArrary")))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // get object时自动解密。
 body, err := cryptoBucket.GetObject("<otherObjectNameEncryptedWithOtherRsa>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer body.Close()
 data, err := ioutil.ReadAll(body)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 fmt.Println("data:", string(data))
}
```

使用主密钥KMS普通上传和下载Object

使用主密钥KMS普通上传和下载Object示例代码如下：

```
package main
import (
 "bytes"
 "fmt"
 "io/ioutil"
 "os"
 kms "github.com/aliyun/alibaba-cloud-sdk-go/services/kms"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
 "github.com/aliyun/aliyun-oss-go-sdk/oss/crypto"
)
```

```
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 创建kms client实例。
 kmsClient, err := kms.NewClientWithAccessKey("<yourKmsRegion>", "<yourKmsAccessKeyId>", "<yourKms
AccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 创建一个主密钥的描述信息，创建后不允许修改。主密钥描述信息和主密钥一一对应。
 // 如果所有的Object都使用相同的主密钥，主密钥描述信息可以为空，但后续不支持更换主密钥。
 // 如果主密钥描述信息为空，解密时无法判断使用的是哪个主密钥。
 // 强烈建议为每个主密钥都配置主密钥描述信息(json字符串)，由客户端保存主密钥和描述信息之间的对应关系（服务
端不保存两者之间的对应关系）。
  // 由主密钥描述信息(json字符串)转换的map。
 materialDesc := make(map[string]string)
 materialDesc["desc"] = "<your kms encrypt key material describe information>"
 // 根据主密钥描述信息创建一个主密钥对象。
 masterkmsCipher, err := osscrypto.CreateMasterAliKms(materialDesc, "<YourKmsId>", kmsClient)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 根据主密钥对象创建一个用于加密的接口，使用aes ctr模式加密。
 contentProvider := osscrypto.CreateAesCtrCipher(masterkmsCipher)
 // 获取一个用于客户端加密的已创建bucket。
 // 客户端加密bucket和普通bucket具有相似的用法。
 cryptoBucket, err := osscrypto.GetCryptoBucket(client, "<yourBucketName>", contentProvider)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // put object时自动加密。
 err = cryptoBucket.PutObject("<yourObjectName>", bytes.NewReader([]byte("yourObjectValueByteArrary
")))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // get object时自动解密。
 body, err := cryptoBucket.GetObject("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer body.Close()
 data, err := ioutil.ReadAll(body)
 if err != nil {
  fmt.Println("Error:", err)
```

```
  os.Exit(-1)
 }
 fmt.Println("data:", string(data))
}
```

9.12. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

> **说明** 有关单链接限速的使用场景及注意事项的更多信息，请参考开发指南的单链接限速文档。

普通上传下载限速

以下代码用于普通上传、下载文件时设置单链接限速：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 //
 // 单链接限速支持以下几个接口：
 // PutObject/AppendObject/PostObject/CopyObject/UploadPart/UploadPartCopy
 // 以下是各接口的使用示例。
 //
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 待上传本地文件。
 fd, err := os.Open("<yourLocalFile>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 // 上传速度上限 参数的内容为数字，默认单位为bit/s。
 // 41943040 表示此请求限速40Mb/s，即5MB/s。
 var traffic int64 = 41943040
 // 示例一、限速上传。
 err = bucket.PutObject("<youObjectName>", fd, oss.TrafficLimitHeader(traffic))
```

```
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 示例二、限速下载, newFile表示下载后文件的名称。
 err = bucket.GetObjectToFile("<youObjectName>", "newFile", oss.TrafficLimitHeader(traffic))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 示例三、追加上传，localFileOne和localFileTwo表示本地的两个文件，将localFileOne上传后，继续追加localFile
Two上传。
 localFileOne := "<localFileOne>"
 localFileTwo := "<localFileTwo>"
 fd1, err := os.Open(localFileOne)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 fd2, err := os.Open(localFileTwo)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 var nextPos int64
 nextPos, err = bucket.AppendObject("<yourObjectName>", fd1, nextPos, oss.TrafficLimitHeader(traffic))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 nextPos, err = bucket.AppendObject("<yourObjectName>", fd2, nextPos, oss.TrafficLimitHeader(traffic))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 示例四、大文件分片上传
 // 将文件分成3片上传，具体分片数可以视文件大小而定。
 chunks, err := oss.SplitFileByPartNum("localFile", 3)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 打开文件。
 fd, err = os.Open("fileName")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 // 初始化上传的文件，并设置限速大小。
 imur, err := bucket.InitiateMultipartUpload("<yourObjectName>")
 if err != nil {
  fmt.Println("Error:", err)
```

```
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 分片上传并限速。
 var parts []oss.UploadPart
 for _, chunk := range chunks {
  fd.Seek(chunk.Offset, os.SEEK_SET)
  part, err := bucket.UploadPart(imur, fd, chunk.Size, chunk.Number, oss.TrafficLimitHeader(traffic))
 if err != nil {
 fmt.Println("Error:", err)
 os.Exit(-1)
 }
  parts = append(parts, part)
 }
 // 上传完成。
 _, err = bucket.CompleteMultipartUpload(imur, parts)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
}
```

使用签名URL方式上传下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
package main
import (
 "fmt"
 "os"
 "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
 // 创建OSSClient实例。
 client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 获取存储空间。
 bucket, err := client.Bucket("<yourBucketName>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 // 示例一、使用签名后的URL进行上传,
 //
 // 打开一个本地文件
 fd, err := os.Open("<localfile>")
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
 defer fd.Close()
 // 设置上传限速，参数格式为数字，默认单位为bit/s。
```

```
  // 设置上传限速，参数格式为数字，默认单位为bit/s。
  // 41943040表示此请求限速40Mb/s，即5MB/s。
  var traffic int64 = 41943040
  // 获取一个上传文件的url。
  strURL, err := bucket.SignURL("<yourObjectName>", oss.HTTPPut, 60, oss.TrafficLimitParam(traffic))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 上传本地文件。
  err = bucket.PutObjectWithURL(strURL, fd)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 示例二、使用签名后的URL进行下载。
  // 获取一个下载文件的url。
  strURL, err = bucket.SignURL("<yourObjectName>", oss.HTTPGet, 60, oss.TrafficLimitParam(traffic))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 下载文件到本地, newFile表示下载后文件的名称。
  err = bucket.GetObjectToFileWithURL(strURL, "newFile")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

9.13. 数据安全性

OSS提供基于MD5和CRC64的数据校验，确保上传、下载和拷贝文件（Object）过程中的数据完整性。

MD5校验

如果上传文件时设置了Content-MD5，OSS会根据接收的内容计算MD5。OSS计算的MD5值和上传提供的MD5值不一致时，则返回InvalidDigest异常，从而保证数据的完整性。返回InvalidDigest异常后，您需要重新上传文件。

以下代码用于上传文件时进行MD5校验：

```
package main
import (
  "crypto/md5"
  "encoding/base64"
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取文件内容。
  content := "<yourObjectValue>"
  // 计算md5值。
  h := md5.New()
  h.Write([]byte(content))
  strMd5 := base64.StdEncoding.EncodeToString(h.Sum(nil))
  // 简单上传。
  err = bucket.PutObject("<yourObjectName>", strings.NewReader(content), oss.ContentMD5(strMd5))
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
}
```

说明 putObject、getObject、appendObject、postObject、uploadPart支持MD5校验。

CRC64校验

上传、下载和拷贝文件时默认开启CRC数据校验，确保数据的完整性。

使用CRC64校验，有如下注意事项：

- putObject、getObject、appendObject、uploadPart支持CRC64校验。上传时默认开启CRC校验，如果客户端计算的CRC值与服务端返回的CRC值不一致， 则会返回错误。
- 范围下载不支持CRC64校验。
- CRC64校验会占用一定的CPU，对上传、下载速度均会有影响。

以下代码用于追加上传文件时进行CRC64数据完整性校验：

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 获取存储空间。
  bucket, err := client.Bucket("<yourBucketName>")
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。
  request := &oss.AppendObjectRequest{
    ObjectKey: "<yourObjectName>",
    Reader:   strings.NewReader("<YourObjectAppendValue1>"),
    Position:  0,
  }
  // 第一次追加时，初始化crc的值为0。
  options := []oss.Option{oss.InitCRC(0)}
  result, err := bucket.DoAppendObject(request, options)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 第二次追加的位置从第一次的返回长度开始。
  request = &oss.AppendObjectRequest{
    ObjectKey: "<yourObjectName>",
    Reader:   strings.NewReader("<YourObjectAppendValue2>"),
    Position:  result.NextPosition,
  }
  // 第二次追加的初始化crc值为第一次返回值。
  options = []oss.Option{oss.InitCRC(result.CRC)}
  result, err = bucket.DoAppendObject(request, options)
  if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
  }
  // 您可以进行多次AppendObject操作。
}
```

9.14. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

> **说明** 关于搭建STS服务的具体操作，请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKeyId和AccessKeySecret）和安全令牌（SecurityToken）。

以下代码用于使用STS凭证构造签名请求。

```
import "github.com/aliyun/aliyun-oss-go-sdk/oss"
// 获取STS临时凭证后，您可以通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和AccessKey
Secret）生成OSSClient。
// 创建OSSClient实例。
client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret", oss.SecurityToken("your
SecurityToken"))
if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
// OSS操作。
}
```

使用签名URL临时授权

以下介绍使用签名URL临时授权的常见示例。使用签名URL进行临时授权的完整代码请参见GitHub。

> **说明** 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

- 生成签名URL

  您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

  在URL中加入签名信息，以便将该URL转给第三方实现授权访问。具体操作，请参见在URL中包含签名。

- 使用签名URL上传文件

  以下代码用于使用签名URL上传本地*D:\\localpath*路径下的examplefile.txt到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

  > **注意** 如果要在前端使用带可选参数的签名URL，请确保在服务端生成该签名URL时设置的ContentType与在前端使用时设置的ContentType一致。

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 获取STS临时凭证后，您可以通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和Access
KeySecret）生成OSSClient。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret", oss.SecurityToken("
yourSecurityToken"))
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称，例如examplebucket。
  bucketName := "examplebucket"
  // 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称。
  objectName := "exampledir/exampleobject.txt"
  // 填写本地文件完整路径，例如D:\\localpath\\examplefile.txt，其中localpath为本地文件examplefile.txt所在
本地路径。
  localFilename := "D:\\localpath\\examplefile.txt"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 签名直传。
  signedURL, err := bucket.SignURL(objectName, oss.HTTPPut, 60)
  if err != nil {
    HandleError(err)
  }
  var val = "上云就上阿里云"
  err = bucket.PutObjectWithURL(signedURL, strings.NewReader(val))
  if err != nil {
    HandleError(err)
  }
  // 带可选参数的签名直传。请确保设置的ContentType值与在前端使用时设置的ContentType值一致。
  options := []oss.Option{
    oss.Meta("myprop", "mypropval"),
    oss.ContentType("text/plain"),
  }
  signedURL, err = bucket.SignURL(objectName, oss.HTTPPut, 60, options...)
  if err != nil {
    HandleError(err)
  }
  err = bucket.PutObjectFromFileWithURL(signedURL, localFilename, options...)
  if err != nil {
    HandleError(err)
  }
```

```
}
```

> **说明** 关于签名URL中的可选参数的更多信息，请参见文件元信息。

- 使用签名URL下载文件

  以下代码用于下载目标存储空间examplebucket中exampledir目录下的exampleobject.txt到本地*D:\\localpath*路径下的examplefile.txt文件。

```
package main
import (
  "fmt"
  "os"
  "io/ioutil"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 获取STS临时凭证后，您可以通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和Access
KeySecret）生成OSSClient。
  client, err := oss.New("yourEndpoint", "yourAccessKeyId", "yourAccessKeySecret", oss.SecurityToken("
yourSecurityToken"))
  if err != nil {
    HandleError(err)
  }
  // 填写Bucket名称，例如examplebucket。
  bucketName := "examplebucket"
  // 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称。
  objectName := "exampledir/exampleobject.txt"
  // 下载OSS文件到本地文件，并保存到指定的本地路径中。如果指定的本地文件存在会覆盖，不存在则新建。
  // 如果未指定本地路径，则下载后的文件默认保存到示例程序所属项目对应本地路径中。
  localDownloadedFilename := "D:\\localpath\\examplefile.txt"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 使用签名URL将OSS文件下载到流。
  signedURL, err := bucket.SignURL(objectName, oss.HTTPGet, 60)
  if err != nil {
    HandleError(err)
  }
  body, err := bucket.GetObjectWithURL(signedURL)
  if err != nil {
    HandleError(err)
  }
  // 读取内容。
  data, err := ioutil.ReadAll(body)
  body.Close()
  data = data // use data。
  // 使用签名URL将OSS文件下载到本地文件。
  err = bucket.GetObjectToFileWithURL(signedURL, localDownloadedFilename)
  if err != nil {
    HandleError(err)
  }
}
```

9.15. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
  HandleError(err)
  }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  sourceImageName := "<yourObjectName>"
  // 指定处理后的图片名称。
  targetImageName := "<LocalFileName>"
  // 将图片缩放为固定宽高100 px后保存在本地。
  style := "image/resize,m_fixed,w_100,h_100"
  err = bucket.GetObjectToFile(sourceImageName, targetImageName, oss.Process(style))
  if err != nil {
  HandleError(err)
  }
}
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
  HandleError(err)
   }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  sourceImageName := "<yourObjectName>"
  // 指定处理后的图片名称。
  targetImageName := "<LocalFileName>"
  // 将图片缩放为固定宽高100 px后，再旋转90°，之后保存在本地。
  style := "image/resize,m_fixed,w_100,h_100/rotate,90"
  err = bucket.GetObjectToFile(sourceImageName, targetImageName, oss.Process(style))
  if err != nil {
   HandleError(err)
  }
}
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
   HandleError(err)
  }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  sourceImageName := "<yourObjectName>"
  // 指定处理后的图片名称。
  targetImageName := "<LocalFileName>"
  // 指定图片样式。
  style := "style/<yourCustomStyleName>"
  // 将处理后的图片保存在本地。
  err = bucket.GetObjectToFile(sourceImageName, targetImageName, oss.Process(style))
  if err != nil {
  HandleError(err)
  }
}
```

图片处理持久化

您可以通过ImgSaveAs接口将处理后的图片保存至指定Bucket。以下代码展示了图片处理持久化操作：

- 图片处理后转存到当前存储空间

```
package main
import (
  "fmt"
  "os"
  "encoding/base64"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
   HandleError(err)
  }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  sourceImageName := "<yourObjectName>"
  // 指定处理后的图片名称。
  targetImageName := "<targetObjectName>"
  // 将图片缩放为固定宽高100 px后转存到当前存储空间。
  style := "image/resize,m_fixed,w_100,h_100"
  process := fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.URLEncoding.EncodeToString([]byte(targetI
mageName)))
  result, err := bucket.ProcessObject(sourceImageName, process)
  if err != nil {
  HandleError(err)
  } else {
    fmt.Println(result)
  }
}
```

- 图片处理后转存到指定存储空间

```
package main
import (
  "fmt"
  "os"
  "encoding/base64"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourSourceBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
   HandleError(err)
  }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  sourceImageName := "<yourObjectName>"
  // 指定处理后图片存放的Bucket，该Bucket需与源Bucket在相同地域。
  targetBucketName := "<TargetBucketName>"
  // 指定处理后的图片名称。
  targetImageName = "<TargetObjectName>"
  // 将图片缩放为固定宽高100 px后转存到指定存储空间。
  style = "image/resize,m_fixed,w_100,h_100"
  process = fmt.Sprintf("%s|sys/saveas,o_%v,b_%v", style, base64.URLEncoding.EncodeToString([]byte(t
argetImageName)), base64.URLEncoding.EncodeToString([]byte(targetBucketName)))
  result, err = bucket.ProcessObject(sourceImageName, process)
  if err != nil {
  HandleError(err)
  } else {
    fmt.Println(result)
  }
}
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
package main
import (
  "fmt"
  "os"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
  HandleError(err)
  }
  // 指定原图所在Bucket。
  bucketName := "<yourBucketName>"
  bucket, err := client.Bucket(bucketName)
  if err != nil {
  HandleError(err)
  }
  // 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。
  ossImageName := "<yourObjectName>"
  // 生成带签名的URL，并指定过期时间为600s。
  signedURL, err := bucket.SignURL(ossImageName, oss.HTTPGet, 600, oss.Process("image/format,png"))
  if err != nil {
  HandleError(err)
  } else {
  fmt.Println(signedURL)
  }
}
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

9.16. 错误处理

本文介绍Go SDK的错误处理。

客户端错误

Go SDK中调用出错会统一返回error接口，该接口定义如下：

```
type error interface {
  Error() string
}
```

其它错误继承该接口。比如HTTP错误请求返回的错误如下：

```
//net.Error
type Error interface {
  error
  Timeout() bool  // Is the error a timeout
  Temporary() bool // Is the error temporary
}
```

服务器端错误

使用OSS Go SDK时如果请求出错，会返回相应的error。HTTP请求、IO等错误会返回Go语言自定义错误。OSS Server处理请求出错，则返回如下的错误，该错误实现了Error接口。

```
type ServiceError struct {
  Code     string   // OSS返回给用户的错误码
  Message   string   // OSS给出的详细错误信息
  RequestId  string   // 用于唯一标识该次请求的UUID
  HostId    string   // 用于标识访问的OSS集群
  StatusCode int     // HTTP状态码
}
```

如果OSS返回的HTTP状态码与预期不符，则返回如下错误，该错误也实现了Error接口。

```
type UnexpectedStatusCodeError struct {
  allowed []int   // 预期OSS返回HTTP状态码
  got    int    // OSS实际返回HTTP状态码
}
```

错误处理示例

以下代码用于展示错误处理：

```
package main
import (
  "fmt"
  "os"
  "strings"
  "github.com/aliyun/aliyun-oss-go-sdk/oss"
)
// 错误处理函数。
func HandleError(err error) {
  fmt.Println("Error:", err)
  os.Exit(-1)
}
func main() {
  // 创建OSSClient实例。
  client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<yourAccessKeySecret>")
  if err != nil {
    HandleError(err)
  }
  bucketName := "<yourBucketName>"
  objectName := "<yourObjectName>"
  // 获取存储空间。
  bucket, err := client.Bucket(bucketName)
  if err != nil {
    HandleError(err)
  }
  // 简单上传：字符串上传。
  err = bucket.PutObject(objectName, strings.NewReader("yourObjectValueContentString"))
  if err != nil {
    HandleError(err)
}
```

OSS错误码

OSS错误码列表如下：

错误码	描述	HTTP状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	Bucket已经存在	409
BucketNotEmpty	Bucket不为空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
InvalidLinkName	Object Link与指向的Object同名	400
LinkPartNotExist	Object Link中指向的Object不存在	400

错误码	描述	HTTP状态码
ObjectLinkTooLarge	Object Link中Object个数过多	400
FieldItemTooLong	Post请求中表单域过大	400
FilePartInterity	文件Part已改变	400
FilePartNotExist	文件Part不存在	400
FilePartStale	文件Part过时	400
IncorrectNumberOfFilesInPOSTRequest	Post请求中文件个数非法	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的Bucket名称	400
InvalidDigest	无效的摘要	400
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400
InvalidObjectName	无效的Object名称	400
InvalidPart	无效的Part	400
InvalidPartOrder	无效的Part顺序	400
InvalidPolicyDocument	无效的Policy文档	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标Bucket	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MalformedPOSTRequest	Post请求的body格式非法	400
MaxPOSTPreDataLengthExceededError	Post请求上传文件内容之外的body过大	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	Bucket不存在	404
NoSuchKey	文件不存在	404

错误码	描述	HTTP状态码
NoSuchUpload	Multipart Upload ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403
RequestTimeout	请求超时	400
RequestIsNotMultiPartContent	Post请求Content-Type非法	400
SignatureDoesNotMatch	签名错误	403
TooManyBuckets	用户的Bucket数目超过限制	400
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

说明 上表中的错误码即OssServiceError.Code，HTTP状态码即OssServiceError.StatusCode。

10.iOS

10.1. 前言

本文档基于 iOS SDK 8.0编写。

简介

本文档假设您已经开通了阿里云OSS 服务，并创建了AccessKeyId 和AccessKeySecret。文中的ID 指的是AccessKeyId，KEY 指的是AccessKeySecret。

如果您还没有开通或者还不了解OSS，请登录OSS产品主页获取更多的帮助。

环境要求

- iOS系统版本：iOS 8.0及以上
- 必须注册有Aliyun.com用户账户，并开通OSS服务。

SDK源码和API文档

SDK源码请参见GitHub。更多信息请参见OSS iOS SDK API文档。

下载SDK

- 通过GitHub下载
- pod依赖： pod 'AliyunOSSiOS'
- demo地址： https://github.com/aliyun/aliyun-oss-ios-sdk

10.2. 安装

本文介绍如何安装OSS iOS SDK。

环境要求

- iOS系统版本：iOS 8.0及以上
- macOS版本：10.10及以上

直接引入Framework

如何获取OSS iOS SDK Framework，请参见GitHub。

在Xcode中，直接把Framework拖入您对应的Target下即可，在弹出框选中**Copy items if needed**。

Pod依赖

如果工程是通过Pod管理依赖，只需在Podfile中加入以下依赖，不需要再导入Framework。

```
pod 'AliyunOSSiOS'
```

> **说明** 您可以选择直接引入Framework或者Pod依赖两种方式中的任意一种。

工程中引入头文件

```
#import <AliyunOSSiOS/OSSService.h>
```

> **注意** 引入Framework后，需要在工程 `Build Settings` 的 `Other Linker Flags` 中加入 `-ObjC` 。如果工程已设置了 `-force_load` 选项，则需要加入 `-force_load <framework path>/AliyunOSSiOS` 。

兼容IPv6-Only网络

为了解决无线网络下域名解析容易遭到劫持的问题，OSS移动端SDK引入了HTTPDNS进行域名解析，直接使用IP请求OSS服务端。在IPv6-Only 的网络下可能会遇到兼容性问题。苹果官方近期发布了关于IPv6-only网络环境兼容的App审核要求。为此，SDK从2.5.0版本开始已经做了兼容性处理。在新版本中，除了-ObjC的设置，还需要引入如下两个系统库：

```
libresolv.tbd
CoreTelephony.framework
SystemConfiguration.framework
```

关于苹果ATS政策

WWDC 2016开发者大会上，苹果宣布从2017年1月1日起，苹果App Store中的所有App都必须启用App Transport Security(ATS)安全功能。即所有新提交的App默认不允许使用 `NSAllowsArbitraryLoads` 绕过ATS限制。此外，还需保证App的所有网络请求都必须通过HTTPS加密，否则可能会在应用审核时遇到麻烦。

OSS iOS SDK在 `2.6.0` 以上版本中对此做出支持。SDK不会自行发出任何非HTTPS请求，同时SDK支持 `https://` 前缀的 `Endpoint` ，只需要设置正确的HTTPS `Endpoint` ，即可保证发出的网络请求均符合要求。

> **注意**
> - 设置 `Endpoint` 时，需使用 `https://` 前缀的URL。
> - 在实现加签、获取STSToken等回调时，需确保不会发出非HTTPS的请求。

10.3. 快速入门

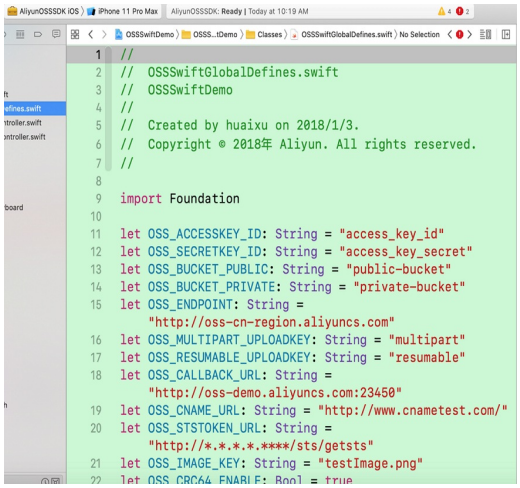
本文介绍如何使用iOS SDK完成上传、下载文件等基础操作。

背景信息

关于iOS SDK的更多信息，请参见如下示例：

- iOS demo示例
- Mac demo示例
- Swift demo示例
- 测试用例

您还可以直接git clone工程，并配置如下必要的参数，例如OSS_ACCESSKEY_ID、OSS_SECRETKEY_ID等。

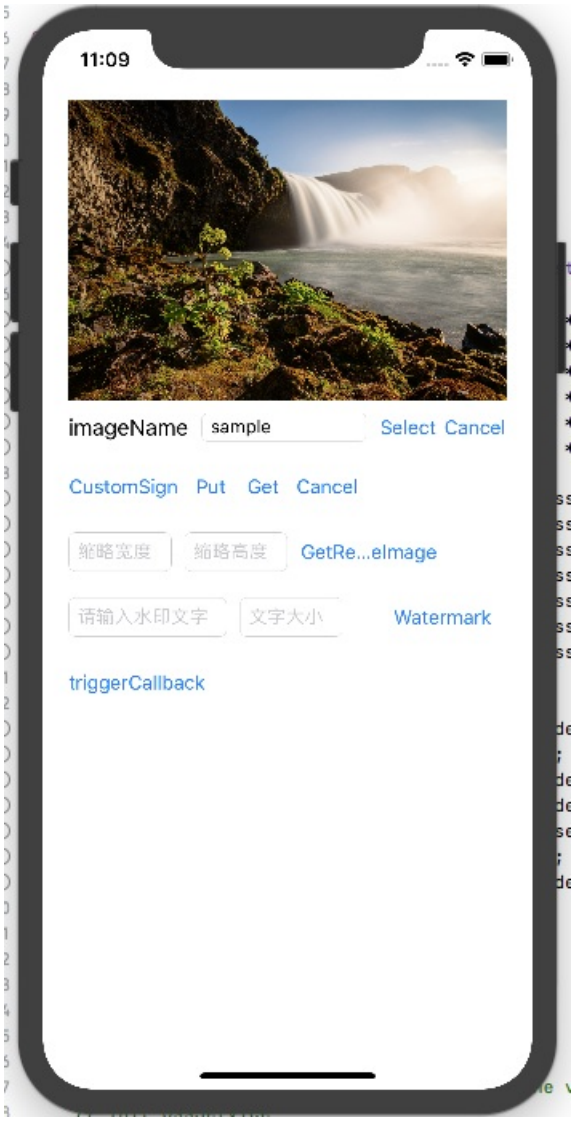


```swift
//
//  OSSSwiftGlobalDefines.swift
//  OSSSwiftDemo
//
//  Created by huaixu on 2018/1/3.
//  Copyright © 2018年 Aliyun. All rights reserved.
//

import Foundation

let OSS_ACCESSKEY_ID: String = "access_key_id"
let OSS_SECRETKEY_ID: String = "access_key_secret"
let OSS_BUCKET_PUBLIC: String = "public-bucket"
let OSS_BUCKET_PRIVATE: String = "private-bucket"
let OSS_ENDPOINT: String =
    "http://oss-cn-region.aliyuncs.com"
let OSS_MULTIPART_UPLOADKEY: String = "multipart"
let OSS_RESUMABLE_UPLOADKEY: String = "resumable"
let OSS_CALLBACK_URL: String =
    "http://oss-demo.aliyuncs.com:23450"
let OSS_CNAME_URL: String = "http://www.cnametest.com/"
let OSS_STSTOKEN_URL: String =
    "http://*.*.*.*.****/sts/getsts"
let OSS_IMAGE_KEY: String = "testImage.png"
let OSS_CRC64_ENABLE: Bool = true
```

运行工程demo如下：



示例代码

以下演示了在iOS平台以及Mac平台上传和下载文件的流程。

1. 需要添加的引用

```
#import <AliyunOSSiOS/OSSService.h>
```

2. 初始化OSSClient

初始化OSSClient时需要完成Endpoint设置、鉴权方式设置和Client参数设置。其中鉴权方式包含明文设置模式、自签名模式以及STS鉴权模式。如果需要使用STS鉴权模式，请参见授权访问。

完善脚本文件中的AccessKeyId、SecretKeyId以及RoleArn参数信息。通过Python启动本机HTTP服务。在客户端代码中访问本地服务以获取StsToken.AccessKeyId、StsToken.SecretKeyId以及StsToken.SecurityToken。

> **说明**
>
> 搭建STS服务的具体操作请参见开发指南中的使用STS临时访问凭证访问OSS。您可以通过调用STS服务的AssumeRole接口或者使用各语言STS SDK来获取临时访问凭证。临时访问凭证包括临时访问密钥（AccessKey ID和AccessKey Secret）和安全令牌（SecurityToken）。

```
NSString *endpoint = @"https://oss-cn-hangzhou.aliyuncs.com";
// 移动端建议使用STS方式初始化OSSClient。
id<OSSCredentialProvider> credential = [[OSSFederationCredentialProvider alloc] initWithFederationTokenGetter:^OSSFederationToken * _Nullable{
  OSSFederationToken *token = [OSSFederationToken new];
  // 从STS服务获取的临时访问密钥（AccessKey ID和AccessKey Secret）。
  token.tAccessKey = @"AccessKeyId";
  token.tSecretKey = @"AccessKeySecret";
  // 从STS服务获取的安全令牌（SecurityToken）。
  token.tToken = @"SecurityToken";
  // 临时访问凭证的过期时间。
  token.expirationTimeInGMTFormat = @"Expiration";
  return token;
}];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

您可以通过OSSClient发起多线程的上传和下载请求，并发执行多个任务。

3. 上传文件

如果在OSS控制台有已创建的Bucket。SDK的所有操作都会返回 `OSSTask` ，您可以为 `OSSTask` 设置延续动作，等待其异步完成，也可以通过调用 `waitUntilFinished` 阻塞等待其完成。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 填写Bucket名称，例如examplebucket。
put.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称，例如exampledir/testdir/exampleobject.txt
。
put.objectKey = @"exampledir/testdir/exampleobject.txt";
// 直接上传NSData。
put.uploadingData = <NSData *>;
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
OSSTask * putTask = [client putObject:put];
[putTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"upload object success!");
  } else {
    NSLog(@"upload object failed, error: %@" , task.error);
  }
  return nil;
}];
// 等待任务完成。
// [putTask waitUntilFinished];
```

4. 下载指定文件

   以下代码用于下载指定Object为 `NSData` 。

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// 填写Bucket名称，例如examplebucket。
request.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称，例如exampledir/testdir/exampleobject.txt
。
request.objectKey = @"exampledir/testdir/exampleobject.txt";
request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpe
ctedToWrite) {
  NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download object success!");
    OSSGetObjectResult * getResult = task.result;
    NSLog(@"download result: %@", getResult.downloadedData);
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
// 阻塞等待任务完成。
// [task waitUntilFinished];
```

10.4. 初始化

OSSClient 是OSS 服务的 iOS 客户端，它为调用者提供了一系列的方法，可以用来操作，管理存储空间（Bucket）和文件（Object）等。在使用 SDK 发起 OSS 请求前，您需要初始化一个 OSSClient 实例，并对其进行一些必要设置。

> **说明**
>
> OSSClient 的生命周期和应用程序的生命周期保持一致即可。在应用程序启动时创建一个 OSSClient 实例，在应用程序结束时释放即可。

确定 Endpoint

Endpoint 是阿里云 OSS 服务在各个区域的地址，目前支持两种形式：

Endpoint 类型	解释
OSS 区域地址	使用 OSS Bucket 所在区域地址，各个区域Endpoint参考访问域名和数据中心。
用户自定义域名	用户自定义域名，且 CNAME 指向 OSS 域名。

- OSS 区域地址

  使用 OSS Bucket 所在区域地址时，您可以通过以下两种方式查询 Endpoint：

  - 查询 Endpoint 与区域对应关系详情，可以参考访问域名和数据中心。
  - 您可以登录 阿里云 OSS 管理控制台，进入 Bucket 概览页，Bucket 域名 `bucket-1.oss-cn-hangzhou.aliyuncs.com` 的后缀部分 `oss-cn-hangzhou.aliyuncs.com` ，即为该 Bucket 的外网 Endpoint。

- 用户自定义域名

  通过 CNAME 将自定义域名绑定到某个存储空间上，然后通过该域名访问存储空间内的文件。

  假设您要将域名 new-image.xxxxx.com 绑定到深圳区域名称为 image 的存储空间。您需要到您的域名 xxxxx.com托管商处设定一个新的域名解析，并将 `http://new-image.xxxxx.com` 解析到 `http://image.oss-cn-shenzhen.aliyuncs.com` ，类型为 CNAME。

设置 EndPoint 和凭证

移动终端是一个不受信任的环境，把 `AccessKeyId` 和 `AccessKeySecret` 直接保存在终端用来加签请求，存在极高的风险。推荐使用STS鉴权模式或自签名模式，详情请参考访问控制、 移动端直传。

> **说明** 如果用 STS 鉴权模式，推荐使用 OSSAuthCredentialsProvider 方式直接访问鉴权应用服务器，token 过期后将自动更新。

设置 EndPoint 和 CredentialProvider 示例如下：

```
NSString *endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 由阿里云颁发的AccessKeyId/AccessKeySecret构造一个CredentialProvider。
// 推荐使用OSSAuthCredentialProvider，token过期后会自动刷新。
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:@"应用服务器地址，例如http://abc.com:8080"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential];
```

设置 EndPoint 为 CNAME

如果您已经在 Bucket 上绑定 CNAME，将该 Endpoint 设置为 CNAME 即可。

```
NSString *endpoint = "http://new-image.xxxxx.com";
// 推荐使用OSSAuthCredentialProvider，token过期后会自动刷新。
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:@"应用服务器地址，例如http://abc.com:8080"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential];
```

注意 苹果要求支持 ATS 标准后，所有 `Endpoint` URL 都必须为 HTTPS URL，而 `CNAME` 域名暂不支持证书设置，因此暂时不能用 `CNAME` 设置 `Endpoint`。

启用日志

移动端的使用环境比较复杂。部分区域或某个时段会出现无法正常使用 OSS SDK 的情况。为了进一步定位开发者遇到的问题，OSS SDK 在开启日志记录功能后，会将日志信息记录在本地。在使用 OSSClient 前进行初始化，并调用如下方法开启日志记录。

```
//日志的样式
//2017/10/25 11:05:43:863 [Debug]: 第17次：<NSThread: 0x7f8099108580>{number = 3, name = (null)}
//2017/10/25 11:05:43:863 [Debug]: 第15次：<NSThread: 0x7f80976052c0>
//2017/10/25 11:05:43:863 [Debug]: ----------TestDebug----------
[OSSLog enableLog];//执行该方法，开启日志记录
```

说明

- 文件存储在沙盒的 Caches/OSSLogs 文件夹内。
- 您可以自行选择将文件上传至服务器，便于进一步追踪问题。

您还可以接入阿里云日志服务 LOG 进行日志文件上传。

设置 ClientConfiguration

以下代码用于初始化时设置详细的 ClientConfiguration：

```
NSString *endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
// 推荐使用OSSAuthCredentialProvider，token过期后会自动刷新。
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:@"应用服务器地址，例如http://abc.com:8080"];
OSSClientConfiguration * conf = [OSSClientConfiguration new];
conf.maxRetryCount = 3; // 网络请求遇到异常失败后的重试次数
conf.timeoutIntervalForRequest = 30; // 网络请求的超时时间
conf.timeoutIntervalForResource = 24 * 60 * 60; // 允许资源传输的最长时间
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential clientConfiguration:conf];
```

OSSTask

所有调用 API 的操作都会立即获得一个 OSSTask，如：

```
OSSTask * task = [client getObject:get];
```

您可以为这个 Task 设置延续 (continution) 以实现异步回调，如：

```
[task continueWithBlock: ^(OSSTask *task) {
  // do something
  ...
  return nil;
}];
```

您还可以等待这个 Task 完成以实现同步等待，如：

```
[task waitUntilFinished];
```

10.5. 存储空间

10.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

背景信息

创建存储空间时，您需要了解如下信息：

- 同一阿里云账号在同一地域内创建的存储空间总数不能超过100个。
- 每个存储空间的名称全局唯一，否则会创建失败。
- 存储空间的名称需要符合命名规范。存储空间的命名规范请参见存储空间（Bucket）。
- 存储空间一旦创建成功，名称和所处地域不能修改。
- 创建存储空间时，您可以指定存储空间的权限。如果未指定存储空间权限，则存储空间权限默认为私有（private）。
- 创建存储空间时，您可以指定存储类型。如果未指定存储类型，则存储空间的存储类型默认为标准存储（standard）。

示例

创建一个名为examplebucket的存储空间，且存储空间的ACL为公共读（public-read）。

```
id<OSSCredentialProvider> credentialProvider = [[OSSAuthCredentialProvider alloc] initWithAuthServerUrl:
@"<StsServer>"];
OSSClientConfiguration *cfg = [[OSSClientConfiguration alloc] init];
cfg.maxRetryCount = 3;
cfg.timeoutIntervalForRequest = 15;
// 由于client在释放时会取消持有的session中的所有任务并将session置为无效，因此请保证client的生命周期在请求
完成前不会被释放。
// Endpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzh
ou.aliyuncs.com。
OSSClient *client = [[OSSClient alloc] initWithEndpoint:@"Endpoint" credentialProvider:credentialProvider
clientConfiguration:cfg];
OSSCreateBucketRequest * create = [OSSCreateBucketRequest new];
create.bucketName = @"examplebucket";
create.xOssACL = @"public-read";
OSSTask * createTask = [client createBucket:create];
[createTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"create bucket success!");
  } else {
    NSLog(@"create bucket failed, error: %@", task.error);
  }
  return nil;
}];
```

10.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何列举所有存储空间。指定前缀的存储空间、指定个数的存储空间等。

列举所有存储空间

以下代码用于列举所有存储空间。

```
OSSGetServiceRequest * getService = [OSSGetServiceRequest new];
OSSTask * getServiceTask = [client getService:getService];
[getServiceTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetServiceResult * result = task.result;
    NSLog(@"buckets: %@", result.buckets);
    NSLog(@"owner: %@, %@", result.ownerId, result.ownerDispName);
    [result.buckets enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
      NSDictionary * bucketInfo = obj;
      NSLog(@"BucketName: %@", [bucketInfo objectForKey:@"Name"]);
      NSLog(@"CreationDate: %@", [bucketInfo objectForKey:@"CreationDate"]);
      NSLog(@"Location: %@", [bucketInfo objectForKey:@"Location"]);
    }];
  } else {
    NSLog(@"get service failed, error: %@", task.error);
  }
  return nil;
}];
```

列举指定前缀的存储空间

以下代码用于列举以bucket为前缀的存储空间。

```
OSSGetServiceRequest * getService = [OSSGetServiceRequest new];
getService.prefix = @"bucket";
OSSTask * getServiceTask = [client getService:getService];
[getServiceTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetServiceResult * result = task.result;
    NSLog(@"buckets: %@", result.buckets);
    NSLog(@"owner: %@, %@", result.ownerId, result.ownerDispName);
    [result.buckets enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
      NSDictionary * bucketInfo = obj;
      NSLog(@"BucketName: %@", [bucketInfo objectForKey:@"Name"]);
      NSLog(@"CreationDate: %@", [bucketInfo objectForKey:@"CreationDate"]);
      NSLog(@"Location: %@", [bucketInfo objectForKey:@"Location"]);
    }];
  } else {
    NSLog(@"get service failed, error: %@", task.error);
  }
  return nil;
}];
```

列举指定marker之后的存储空间

参数marker表示存储空间名称。以下代码用于列举examplebucket之后的存储空间。

```
OSSGetServiceRequest * getService = [OSSGetServiceRequest new];
getService.marker = @"examplebucket";
OSSTask * getServiceTask = [client getService:getService];
[getServiceTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetServiceResult * result = task.result;
    NSLog(@"buckets: %@", result.buckets);
    NSLog(@"owner: %@, %@", result.ownerId, result.ownerDispName);
    [result.buckets enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
      NSDictionary * bucketInfo = obj;
      NSLog(@"BucketName: %@", [bucketInfo objectForKey:@"Name"]);
      NSLog(@"CreationDate: %@", [bucketInfo objectForKey:@"CreationDate"]);
      NSLog(@"Location: %@", [bucketInfo objectForKey:@"Location"]);
    }];
  } else {
    NSLog(@"get service failed, error: %@", task.error);
  }
  return nil;
}];
```

列举指定个数的存储空间

以下代码用于列举最多20个存储空间。

```
OSSGetServiceRequest * getService = [OSSGetServiceRequest new];
getService.maxKeys = 20;
OSSTask * getServiceTask = [client getService:getService];
[getServiceTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetServiceResult * result = task.result;
    NSLog(@"buckets: %@", result.buckets);
    NSLog(@"owner: %@, %@", result.ownerId, result.ownerDispName);
    [result.buckets enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
      NSDictionary * bucketInfo = obj;
      NSLog(@"BucketName: %@", [bucketInfo objectForKey:@"Name"]);
      NSLog(@"CreationDate: %@", [bucketInfo objectForKey:@"CreationDate"]);
      NSLog(@"Location: %@", [bucketInfo objectForKey:@"Location"]);
    }];
  } else {
    NSLog(@"get service failed, error: %@", task.error);
  }
  return nil;
}];
```

10.5.3. 获取存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间访问权限（ACL）。

存储空间访问权限

存储空间的访问权限（ACL）包括如下三种：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	private
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	public-read-write

示例

以下代码用于获取examplebucket存储空间的访问权限。

```
OSSGetBucketACLRequest *request = [OSSGetBucketACLRequest new];
request.bucketName = @"examplebucket";
OSSTask * getBucketACLTask = [client getBucketACL:request];
[getBucketACLTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetBucketACLResult *result = task.result;
    NSLog(@"权限：%@", result.aclGranted);
  } else {
    NSLog(@"get bucket ACL failed, error: %@", task.error);
  }
  return nil;
}];
```

10.5.4. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取examplebucket存储空间的信息（Info），包括存储空间所在地域、创建日期、权限信息等。

```
OSSGetBucketInfoRequest *request = [OSSGetBucketInfoRequest new];
request.bucketName = @"examplebucket";
OSSTask * getBucketInfoTask = [client getBucketInfo:request];
[getBucketInfoTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetBucketInfoResult *result = task.result;
    NSLog(@"创建时间：%@", result.creationDate);
    NSLog(@"地域：%@", result.location);
    NSLog(@"类型：%@", result.storageClass);
    NSLog(@"拥有者信息：%@", result.owner.userName);
    NSLog(@"权限：%@", result.acl.grant);
  } else {
    NSLog(@"get bucket info failed, error: %@", task.error);
  }
  return nil;
}];
```

10.5.5. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
>
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用Bucket.ListMultipartUploads列举出所有碎片，然后使用Bucket.AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间examplebucket。

```
OSSDeleteBucketRequest * delete = [OSSDeleteBucketRequest new];
delete.bucketName = @"examplebucket";
OSSTask * deleteTask = [client deleteBucket:delete];
[deleteTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"delete bucket success!");
  } else {
    NSLog(@"delete bucket failed, error: %@", task.error);
  }
  return nil;
}];
```

关于删除存储空间的更多信息，请参见DeleteBucket。

10.6. 上传文件

10.6.1. 概述

本文档介绍 OSS iOS SDK 上传文件的方式。

在 OSS 中，操作的基本数据单元是文件（Object）。OSS iOS SDK提供了以下三种文件上传方式：

- 简单上传：包括从内存中上传或上传本地文件。最大不能超过 5GB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。
- 追加上传：最大不能超过 5GB。
- 断点续传上传：支持并发上传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过 48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传完成后，您还可以进行上传回调。

10.6.2. 简单上传

本文介绍如何从内存中上传文件或者上传本地文件。您也可以使用MD5校验来确保上传过程中的数据完整性。

从内存中上传文件或上传本地文件

上传文件时可以直接上传OSSData或者通过NSURL上传文件。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 填写Bucket名称，例如examplebucket。
put.bucketName = @"examplebucket";
// 填写文件完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
put.objectKey = @"exampledir/exampleobject.txt";
put.uploadingFileURL = [NSURL fileURLWithPath:@"<filePath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData。
// （可选）设置上传进度。
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  // 指定当前上传长度、当前已经上传总长度、待上传的总长度。
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
// 配置可选字段。
// put.contentType = @"application/octet-stream";
// put.contentMd5 = @"";
// put.contentEncoding = @"";
// put.contentDisposition = @"";
// 可以在上传文件时设置文件元数据或者HTTP头部。
// put.objectMeta = [NSMutableDictionary dictionaryWithObjectsAndKeys:@"value1", @"x-oss-meta-name1
", nil];
OSSTask * putTask = [client putObject:put];
[putTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"upload object success!");
  } else {
    NSLog(@"upload object failed, error: %@" , task.error);
  }
  return nil;
}];
// [putTask waitUntilFinished];
// [put cancel];
```

关于contentType、contentMd5、contentEncoding、contentDisposition、objectMeta等可选参数的含义，请参见PutObject。

上传到文件目录

OSS没有文件夹的概念，所有元素都是以文件来存储。OSS提供了创建模拟文件夹的方式。创建模拟文件夹本质上是创建了一个名称以正斜线（/）结尾的文件以实现将文件上传至目录，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

例如上传文件时，如果把objectKey设置为 `folder/subfolder/file` ，表示将file文件上传到 `folder/subfolder/` 目录下。

> **说明** 路径默认是根目录，不需要以正斜线（/）开头。

上传文件时设置contentType并带有MD5校验

为保证客户端发送的数据和OSS服务端接收到的数据一致，您可以在上传文件时增加contentMd5值，OSS服务端会使用该MD5值做校验。上传文件时还可以显式指定contentType，如果未显式指定contentType，则SDK会根据文件名或者上传的objectKey自行判断。

注意 使用MD5校验时，性能会有所下降。

SDK提供了便捷的Base64和contentMd5的计算方法。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 填写Bucket名称，例如examplebucket。
put.bucketName = @"examplebucket";
// 填写文件完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
put.objectKey = @"exampledir/exampleobject.txt";
put.uploadingFileURL = [NSURL fileURLWithPath:@"<filePath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData。
// （可选）设置contentType。
put.contentType = @"application/octet-stream";
// （可选）设置contentMd5校验。
// 设置文件路径的contentMd5校验。
put.contentMd5 = [OSSUtil base64Md5ForFilePath:@"<filePath>"];
// 设置二进制数据的contentMd5校验。
// put.contentMd5 = [OSSUtil base64Md5ForData:<NSData *>];
// （可选）设置上传进度。
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  // 指定当前上传长度、当前已经上传总长度、待上传的总长度。
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
OSSTask * putTask = [client putObject:put];
[putTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"upload object success!");
  } else {
    NSLog(@"upload object failed, error: %@" , task.error);
  }
  return nil;
}];
// [putTask waitUntilFinished];
// [put cancel];
```

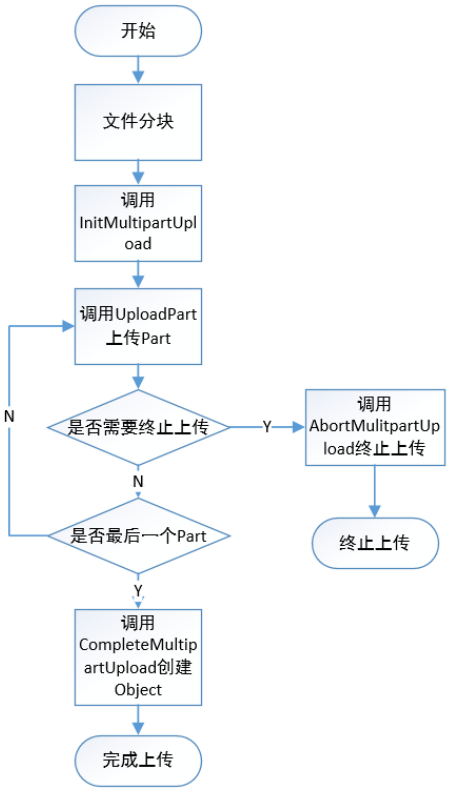
10.6.3. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大的文件（Object）分成多个数据块（OSS中称之为Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传的基本流程如下：

1. 将要上传的文件按照一定的大小分片。
2. 初始化一个分片上传任务（InitiateMultipartUpload）。
3. 逐个或并行上传分片（UploadPart）。
4. 完成上传（CompleteMultipartUpload）。



该过程需注意以下几点：

- 除了最后一块Part，其他Part的大小不能小于100 KB，否则会导致调用CompleteMultipartUpload接口失败。
- 要上传的文件切分成Part之后，文件顺序是通过上传过程中指定的partNumber来确定的，实际执行中并没有顺序要求，因此可以实现并发上传。

  具体的并发个数并不是越多速度越快，要结合用户自身的网络情况和设备负载综合考虑。网络情况较好时，建议增大分片大小。反之，减小分片大小。
- 默认情况下，已经上传但还没有调用CompleteMultipartUpload的Part是不会被自动回收的，因此如果要终止上传并删除占用的空间请调用AbortMultipartUpload。如果需要自动回收上传的Part，请参见生命周期管理。

初始化分片上传

以下代码用于初始化分片上传。

```
__block NSString * uploadId = nil;
__block NSMutableArray * partInfos = [NSMutableArray new];
NSString * uploadToBucket = @"<bucketName>";
// objectKey等同于objectName，表示上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.j
pg
NSString * uploadObjectkey = @"<objectKey>";
// OSSInitMultipartUploadRequest用于指定上传文件的名称以及上传文件所属的存储空间的名称。
OSSInitMultipartUploadRequest * init = [OSSInitMultipartUploadRequest new];
init.bucketName = uploadToBucket;
init.objectKey = uploadObjectkey;
// init.contentType = @"application/octet-stream";
// multipartUploadInit返回的结果中包含UploadId，UploadId是分片上传的唯一标识。
OSSTask * initTask = [client multipartUploadInit:init];
[initTask waitUntilFinished];
if (!initTask.error) {
  OSSInitMultipartUploadResult * result = initTask.result;
  uploadId = result.uploadId;
} else {
  NSLog(@"multipart upload failed, error: %@", initTask.error);
  return;
}
```

上传分片

以下代码用于上传分片。

```
// 指定要上传的文件。
NSString * filePath = [docDir stringByAppendingPathComponent:@"***"];
// 获取文件大小。
uint64_t fileSize = [[[NSFileManager defaultManager] attributesOfItemAtPath:filePath error:nil] fileSize];
// 设置分片上传数量。
int chuckCount = *;
// 设置分片大小。
uint64_t offset = fileSize/chuckCount;
for (int i = 1; i <= chuckCount; i++) {
  OSSUploadPartRequest * uploadPart = [OSSUploadPartRequest new];
  uploadPart.bucketName = uploadToBucket;
  uploadPart.objectkey = uploadObjectkey;
  uploadPart.uploadId = uploadId;
  uploadPart.partNumber = i; // part number start from 1
  NSFileHandle* readHandle = [NSFileHandle fileHandleForReadingAtPath:filePath];
  [readHandle seekToFileOffset:offset * (i -1)];
  NSData* data = [readHandle readDataOfLength:offset];
  uploadPart.uploadPartData = data;
  OSSTask * uploadPartTask = [client uploadPart:uploadPart];
  [uploadPartTask waitUntilFinished];
  if (!uploadPartTask.error) {
    OSSUploadPartResult * result = uploadPartTask.result;
    uint64_t fileSize = [[[NSFileManager defaultManager] attributesOfItemAtPath:uploadPart.uploadPartFile
URL.absoluteString error:nil] fileSize];
    [partInfos addObject:[OSSPartInfo partInfoWithPartNum:i eTag:result.eTag size:fileSize]];
  } else {
    NSLog(@"upload part error: %@", uploadPartTask.error);
    return;
  }
}
```

上述代码调用 `uploadPart` 来上传每一个分片。

- 每一个分片上传请求需指定UploadId和PartNum。
- `uploadPart` 接口并不会立即校验上传。只有完成分片上传时才会校验Part的大小。
- Part编号范围是1~10000。如果超出该范围，OSS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传片开头所对应的位置。
- 每次上传Part之后，OSS的返回结果中会包含每个分片的ETag值，ETag值为Part数据MD5值，您需要将ETag值和Part编号组合成PartETag并保存，用于后续完成分片上传。

完成分片上传

以下代码中的 `partInfos` 为分片上传过程中保存的PartETag列表，OSS收到您提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，OSS会将这些Part组合成一个完整的文件。

以下代码用于完成分片上传。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipartUploadRequest new];
complete.bucketName = uploadToBucket;
complete.objectKey = uploadObjectkey;
complete.uploadId = uploadId;
complete.partInfos = partInfos;
OSSTask * completeTask = [client completeMultipartUpload:complete];
[[completeTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSCompleteMultipartUploadResult * result = task.result;
    // ...
  } else {
    // ...
  }
  return nil;
}] waitUntilFinished];
```

完成分片上传请求时可以设置 `servercallback` 函数，请求完成后会向指定的Server Address发送回调请求。您可以通过返回结果中的 `result.serverReturnJsonString` 查看 `servercallback` 结果。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipartUploadRequest new];
complete.bucketName = @"<bucketName>";
complete.objectKey = @"<objectKey>";
complete.uploadId = uploadId;
complete.partInfos = partInfos;
complete.callbackParam = @{
                @"callbackUrl": @"<server address>",
                @"callbackBody": @"<test>"
                };
complete.callbackVar = @{
               @"var1": @"value1",
               @"var2": @"value2"
               };
OSSTask * completeTask = [client completeMultipartUpload:complete];
[[completeTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSCompleteMultipartUploadResult * result = task.result;
    NSLog(@"server call back return : %@", result.serverReturnJsonString);
  } else {
    // ...
  }
  return nil;
}] waitUntilFinished];
```

如果要校验分片上传到OSS的文件和本地文件是否一致，可以在上传文件时携带文件的Content-MD5值，OSS服务器会帮助您进行Content-MD5校验。只有在OSS服务器接收到的文件MD5值和Content-MD5一致时才可以上传成功，从而保证上传文件的数据完整性。

以下代码用于MD5校验设置。

```
OSSUploadPartRequest * uploadPart = [OSSUploadPartRequest new];
uploadPart.bucketName = TEST_BUCKET;
uploadPart.uploadId = uploadId;
....
uploadPart.contentMd5 = [OSSUtil fileMD5String:filepath];
```

列举分片

调用 `listParts` 方法获取某个上传事件所有已上传的分片。

```
OSSListPartsRequest * listParts = [OSSListPartsRequest new];
listParts.bucketName = @"<bucketName>";
listParts.objectKey = @"<objectkey>";
listParts.uploadId = @"<uploadid>";
OSSTask * listPartTask = [client listParts:listParts];
[listPartTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"list part result success!");
    OSSListPartsResult * listPartResult = task.result;
    for (NSDictionary * partInfo in listPartResult.parts) {
      NSLog(@"each part: %@", partInfo);
    }
  } else {
    NSLog(@"list part result error: %@", task.error);
  }
  return nil;
}];
```

注意 默认情况下，如果存储空间中的分片上传事件的数量大于1000，则OSS仅返回1000个Multipart Upload信息，且返回结果中IsTruncated的值为false，并返回NextPartNumberMarker作为下此读取的起点。

取消分片上传

以下代码用于取消了对应 `UploadId` 的分片上传请求。

```
OSSAbortMultipartUploadRequest * abort = [OSSAbortMultipartUploadRequest new];
abort.bucketName = @"<bucketName>";
abort.objectKey = @"<objectKey>";
abort.uploadId = uploadId;
OSSTask * abortTask = [client abortMultipartUpload:abort];
[abortTask waitUntilFinished];
if (!abortTask.error) {
  OSSAbortMultipartUploadResult * result = abortTask.result;
  uploadId = result.uploadId;
} else {
  NSLog(@"multipart upload failed, error: %@", abortTask.error);
  return;
}
```

10.6.4. 追加上传

追加上传指的是使用OSS API中的AppendObject在已上传的Appendable Object类型文件后直接追加内容。

有关追加上传的更多详情，请参见追加上传

以下代码用于追加上传：

```
OSSAppendObjectRequest * append = [OSSAppendObjectRequest new];
// 配置以下必填字段，其中bucketName为存储空间名称；objectKey等同于objectName，表示追加上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
append.bucketName = @"<bucketName>";
append.objectKey = @"<objectKey>";
// 指定首次进行追加上传的位置。
append.appendPosition = 0;
NSString * docDir = [self getDocumentDirectory];
append.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// 以下为可选字段。
append.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
// append.contentType = @"";
// append.contentMd5 = @"";
// append.contentEncoding = @"";
// append.contentDisposition = @"";
OSSTask * appendTask = [client appendObject:append];
[appendTask continueWithBlock:^id(OSSTask *task) {
  NSLog(@"objectKey: %@", append.objectKey);
  if (!task.error) {
    NSLog(@"append object success!");
    OSSAppendObjectResult * result = task.result;
    NSString * etag = result.eTag;
    long nextPosition = result.xOssNextAppendPosition;
  } else {
    NSLog(@"append object failed, error: %@" , task.error);
  }
  return nil;
}];
```

有关可选字段的详情，请参见AppendObject。

10.6.5. 断点续传上传

在无线网络下，上传比较大的文件持续时间长，可能会遇到因为网络条件差、用户切换网络等原因导致上传中途失败，整个文件需要重新上传。为此，iOS SDK提供了断点续传上传功能。

背景信息

对于移动端来说，如果不是大文件（例如小于5 GB的文件），不建议使用此种方式上传。断点续传上传是通过分片上传实现的，上传单个文件需要进行多次网络请求，效率不高。以下介绍断点续传上传前以及断点续传上传过程中的注意事项。

- 断点续传上传前

通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录的保存文件夹。断点续传上传仅在本次上传生效。

- 如果未指定断点记录的保存文件夹，假设某个分片因为网络原因等导致文件上传失败时，将耗用大量的重试时间及流量来重新上传整个大文件。
- 如果指定了断点记录的保存文件夹，在文件上传失败时，将从断点记录处继续上传未上传完成的部分。

- 断点续传上传时

  使用断点续传上传时，您需要了解以下信息。

  - 断点续传上传仅支持上传本地文件。断点续传上传支持上传回调，使用方法与常见的上传回调类似，详情请参见Callback。
  - 断点续传上传依赖InitMultipartUpload、UploadPart、ListParts、CompleteMultipartUpload、AbortMultipartUpload等接口来实现。如果您需要通过STS鉴权模式来使用断点续传上传，则需要保证您拥有访问上述API接口的权限。
  - 断点续传上传默认已开启每个分片上传时的MD5校验，因此无需在request中设置 `Content-Md5` 头部。
  - 如果同一任务一直得不到续传，可能会在OSS上积累无用碎片。此时，您可以为Bucket设置lifeCycle规则的方式来定时清理碎片，详情请参见生命周期管理。

    注意 出于碎片管理的原因，如果在断点续传时取消当前上传任务，默认会同步清理已经上传到服务器的分片。取消上传任务时如果仍希望保留断点上传记录，则需要指定断点记录的保存文件夹并修改deleteUploadIdOnCancelling参数。如果服务端保留记录时间过长，且Bucket已设置lifeCycle规则定时清理了服务端分片，会出现服务端和移动端记录不一致的问题。

示例代码

以下为断点续传上传的常见示例。

- 断点记录在本地持久保存时，调用 `resumableUpload` 方法实现断点续传上传的过程如下：

```
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRequest new];
resumableUpload.bucketName = OSS_BUCKET_PRIVATE;
//...
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask,
YES) firstObject];
resumableUpload.recordDirectoryPath = cachesDir;
```

  说明 使用断点续传上传文件失败时会生成断点记录文件，然后从断点记录处继续上传直到上传完成。上传成功后会自动删除断点记录文件，默认情况下不会在本地持久保存断点记录。

- 断点续传上传的完整示例代码如下：

```
// 获取UploadId上传文件。
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRequest new];
resumableUpload.bucketName = <bucketName>;
// objectKey等同于objectName，表示断点上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg
/123.jpg
resumableUpload.objectKey = <objectKey>;
resumableUpload.partSize = 1024 * 1024;
resumableUpload.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpect
edToSend) {
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask,
YES) firstObject];
// 设置断点记录保存路径。
resumableUpload.recordDirectoryPath = cachesDir;
// 将参数deleteUploadIdOnCancelling设置为NO，表示不删除断点记录文件，上传失败后将从断点记录处继续上
传直到文件上传完成。如果不设置此参数，即保留默认值YES，表示删除断点记录文件，下次再上传同一文件时则重
新上传。
resumableUpload.deleteUploadIdOnCancelling = NO;
resumableUpload.uploadingFileURL = [NSURL fileURLWithPath:<your file path>];
OSSTask * resumeTask = [client resumableUpload:resumableUpload];
[resumeTask continueWithBlock:^id(OSSTask *task) {
  if (task.error) {
    NSLog(@"error: %@", task.error);
    if ([task.error.domain isEqualToString:OSSClientErrorDomain] && task.error.code == OSSClientErrorC
odeCannotResumeUpload) {
      // 此任务无法续传，需获取新的uploadId重新上传。
    }
  } else {
    NSLog(@"Upload file success");
  }
  return nil;
}];
// [resumeTask waitUntilFinished];
// [resumableUpload cancel];
```

10.6.6. 上传回调

本文介绍如何使用上传回调。

OSS在上传文件完成的时候可以提供回调（Callback）给应用服务器。您只需要在发送给OSS的请求中携带相应的Callback参数，即能实现回调。因为加入了回调请求和响应的过程，相比简单上传，使用回调通知机制一般会导致客户端花费更多的等待时间。

具体说明参考：Callback

以下代码用于上传回调：

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
request.uploadingFileURL = [NSURL fileURLWithPath:@<filepath>"];
// 设置回调参数
request.callbackParam = @{
                 @"callbackUrl": @"<your server callback address>",
                 @"callbackBody": @"<your callback body>"
                 };
// 设置自定义变量
request.callbackVar = @{
                @"<var1>": @"<value1>",
                @"<var2>": @"<value2>"
                };
request.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
OSSTask * task = [client putObject:request];
[task continueWithBlock:^id(OSSTask *task) {
  if (task.error) {
    OSSLogError(@"%@", task.error);
  } else {
    OSSPutObjectResult * result = task.result;
    NSLog(@"Result - requestId: %@, headerFields: %@, servercallback: %@",
        result.requestId,
        result.httpResponseHeaderFields,
        result.serverReturnJsonString);
  }
  return nil;
}];
```

10.6.7. 进度条

进度条用于指示上传或下载文件的进度。本文以PutObject接口为例介绍如何打印上传文件（Object）的进度条。

将本地文件examplefile.txt上传到目标存储空间examplebucket中的exampleobject.txt文件时，您可以使用以下代码获取上传文件的进度条。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 填写Bucket名称。
put.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
put.objectKey = @"exampleobject.txt";
// 填写本地文件的完整路径。
// 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
put.uploadingFileURL = [NSURL fileURLWithPath:@"D:\\localpath\\examplefile.txt"];
// 设置进度回调函数打印进度条。
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent, int64_t totalBytesExpectedToSend) {
  // 当前上传长度、当前已上传总长度、待上传的总长度。
  NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytesExpectedToSend);
};
OSSTask * putTask = [client putObject:put];
[putTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"upload object success!");
  } else {
    NSLog(@"upload object failed, error: %@" , task.error);
  }
  return nil;
}];
```

10.7. 下载文件

10.7.1. 概述

本文档介绍 OSS iOS SDK 下载文件的方式。

OSS iOS SDK 提供了如下几种下载文件的方式：

- 简单下载
- 流式下载
- 范围下载
- 断点续传下载

10.7.2. 简单下载

本文档介绍如何进行简单下载。

下载文件时，您可以指定下载为本地文件或者下载为NSData。

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// 必填字段。
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
// 可选字段。
request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpected
ToWrite) {
  // 当前下载段长度、当前已经下载总长度、一共需要下载的总长度。
  NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};
// request.range = [[OSSRange alloc] initWithStart:0 withEnd:99]; // bytes=0-99，指定范围下载。
// request.downloadToFileURL = [NSURL fileURLWithPath:@"<filepath>"]; // 如果需要直接下载到文件，需要指
明目标文件地址。
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download object success!");
    OSSGetObjectResult * getResult = task.result;
    NSLog(@"download result: %@", getResult.downloadedData);
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
// [getTask waitUntilFinished];
// [request cancel];
```

10.7.3. 流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载，一次处理部分内容，直到完成文件的下载。

OSS iOS SDK没有提供stream类型的下载接口，但是提供了类似 `NSURLSession` 库的 `didRecieveData` 函数的分段回调功能。如果设置了分段回调，下载的结果中将不再包含实际数据。

以下代码用于流式下载：

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// 以下为必选字段。其中objectKey等同于objectName，表示从OSS下载文件时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
// 设置分段回调函数
request.onRecieveData = ^(NSData * data) {
  NSLog(@"Recieve data, length: %ld", [data length]);
};
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download object success!");
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
// [getTask waitUntilFinished];
// [request cancel];
```

10.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

范围下载适用于下载较大的 Object 。如果在请求头中使用 Range 参数，则返回消息中会包含整个文件的长度和此次返回的范围。

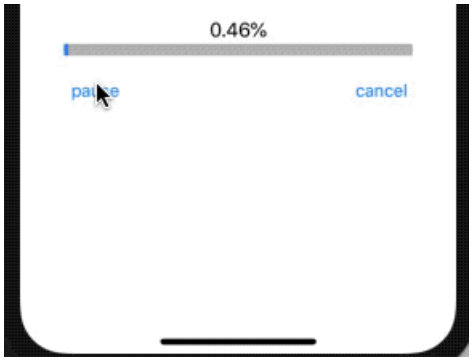
```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"<bucketName>;
request.objectKey = @"<objectKey>";
request.range = [[OSSRange alloc] initWithStart:1 withEnd:99]; // bytes=1-99
// request.range = [[OSSRange alloc] initWithStart:-1 withEnd:99]; // bytes=-99
// request.range = [[OSSRange alloc] initWithStart:10 withEnd:-1]; // bytes=10-
request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpectedToWrite) {
  NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download object success!");
    OSSGetObjectResult * getResult = task.result;
    NSLog(@"download result: %@", getResult.dowloadedData);
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
// [getTask waitUntilFinished];
// [request cancel];
```

10.7.5. 断点续传下载

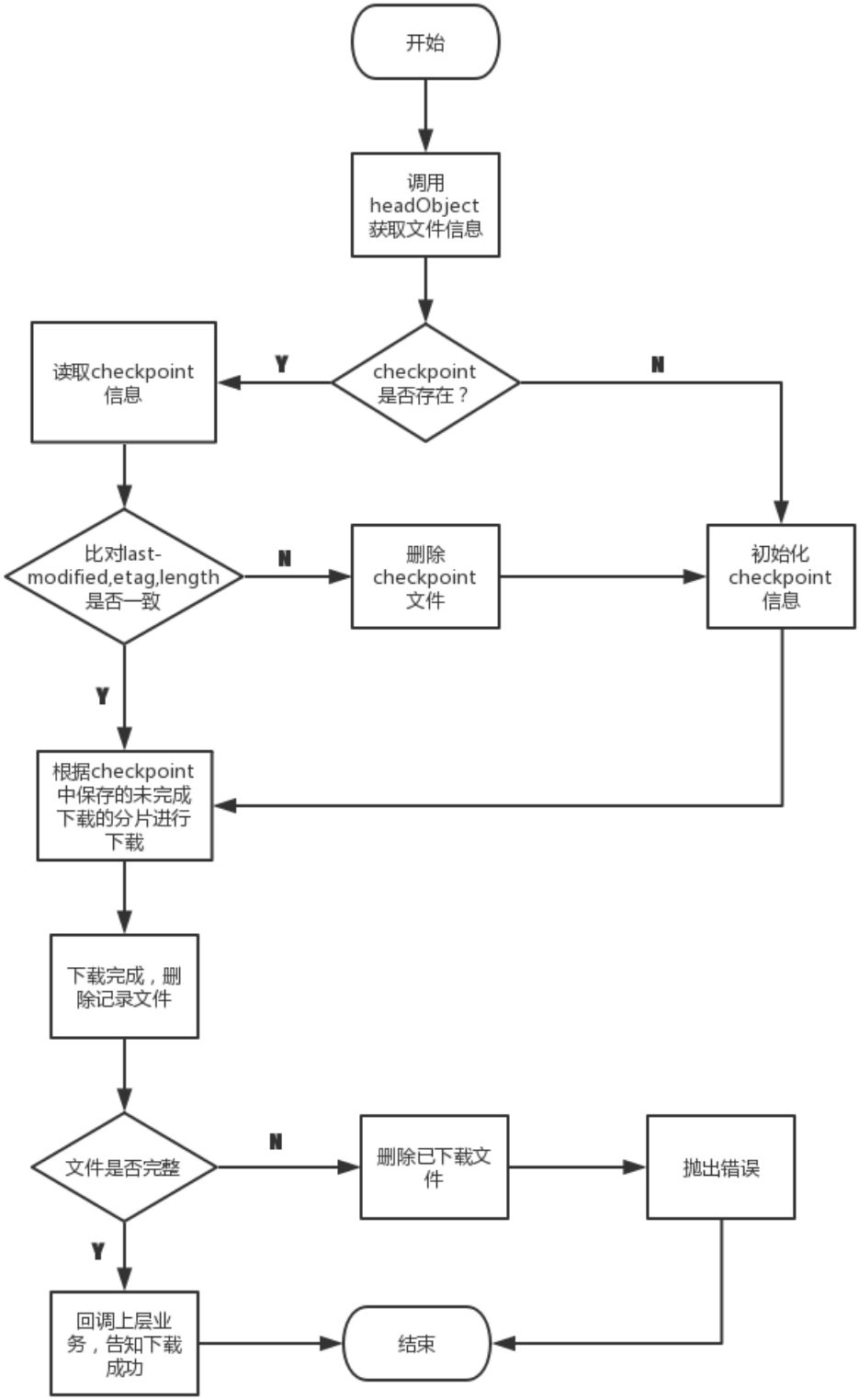
断点续传下载是指客户端在从网络上下载资源时，由于某种原因中断下载。再次开启下载时可以从已下载完成的部分开始继续下载未完成的部分，从而节省时间和流量。

当我们在手机端使用视频软件下载视频时，下载期间网络模式从 Wifi 切换到移动网络，此时 App 默认都会中断下载。当再次切换到 Wifi 网络时，由用户手动重新开启下载任务，即开始断点续传下载。

执行断点续传下载操作时，效果图如下所示。



断点续传下载流程如下所示。



细节分析

- HTTP1.1中新增了Range头的支持，用于指定获取数据的范围。Range的格式一般分为以下几种：
  - `Range: bytes=100-` ：从101 bytes之后开始传，一直传到最后。

- `Range: bytes=100-200`：指定开始到结束这一段的长度。Range是从0计数的，如果指定`Range: bytes=100-200`，则要求服务器从101字节开始传，一直到201字节结束。通常用于较大的文件分片传输，例如视频。
- `Range: bytes=-100`：表示范围没有指定开始位置，则要求服务器传递倒数100字节的内容，而不是从0开始的100字节。
- `Range: bytes=0-100, 200-300`：用于同时指定多个范围的内容，这种情况并不常见。

- 断点续传下载时需要使用`If-Match`头来验证服务器上的文件是否发生变化，`If-Match`对应的是`ETag`的值。
- 客户端发起请求时在Header中携带`Range`、`If-Match`，OSS Server收到请求后会验证If-Match中的ETag值。如果不匹配，则返回412 precondition状态码。
- OSS Server针对GetObject目前已支持`Range`、`If-Match`、`If-None-Match`、`If-Modified-Since`、`If-Unmodified-Since`，因此您可以在移动端实践OSS资源的断点续传下载功能。

具体实现

以iOS实现断点续传下载为例：

> **说明** 以下代码仅供参考了解下载流程，不建议在生产项目中使用。如需下载，可自行实现或使用第三方开源的下载框架。

```
#import "DownloadService.h"
#import "OSSTestMacros.h"
@implementation DownloadRequest
@end
@implementation Checkpoint
- (instancetype)copyWithZone:(NSZone *)zone {
  Checkpoint *other = [[[self class] allocWithZone:zone] init];
  other.etag = self.etag;
  other.totalExpectedLength = self.totalExpectedLength;
  return other;
}
@end
@interface DownloadService()<NSURLSessionTaskDelegate, NSURLSessionDataDelegate>
@property (nonatomic, strong) NSURLSession *session;      //网络会话。
@property (nonatomic, strong) NSURLSessionDataTask *dataTask;  //数据请求任务。
@property (nonatomic, copy) DownloadFailureBlock failure;   //请求出错。
@property (nonatomic, copy) DownloadSuccessBlock success;   //请求成功。
@property (nonatomic, copy) DownloadProgressBlock progress; //下载进度。
@property (nonatomic, copy) Checkpoint *checkpoint;      //检查节点。
@property (nonatomic, copy) NSString *requestURLString;   //文件资源地址，用于下载请求。
@property (nonatomic, copy) NSString *headURLString;      //文件资源地址，用于head请求。
@property (nonatomic, copy) NSString *targetPath;    //文件存储路径。
@property (nonatomic, assign) unsigned long long totalReceivedContentLength; //已下载大小。
@property (nonatomic, strong) dispatch_semaphore_t semaphore;
@end
@implementation DownloadService
- (instancetype)init
{
  self = [super init];
  if (self) {
    NSURLSessionConfiguration *conf = [NSURLSessionConfiguration defaultSessionConfiguration];
```

```
        conf.timeoutIntervalForRequest = 15;
        NSOperationQueue *processQueue = [NSOperationQueue new];
        _session = [NSURLSession sessionWithConfiguration:conf delegate:self delegateQueue:processQueue];
        _semaphore = dispatch_semaphore_create(0);
        _checkpoint = [[Checkpoint alloc] init];
    }
    return self;
}
+ (instancetype)downloadServiceWithRequest:(DownloadRequest *)request {
    DownloadService *service = [[DownloadService alloc] init];
    if (service) {
        service.failure = request.failure;
        service.success = request.success;
        service.requestURLString = request.sourceURLString;
        service.headURLString = request.headURLString;
        service.targetPath = request.downloadFilePath;
        service.progress = request.downloadProgress;
        if (request.checkpoint) {
            service.checkpoint = request.checkpoint;
        }
    }
    return service;
}
/**
 * head文件信息，取出来文件的ETag和本地checkpoint中保存的ETag进行对比，并且将结果返回。
 */
- (BOOL)getFileInfo {
    __block BOOL resumable = NO;
    NSURL *url = [NSURL URLWithString:self.headURLString];
    NSMutableURLRequest *request = [[NSMutableURLRequest alloc]initWithURL:url];
    [request setHTTPMethod:@"HEAD"];
    NSURLSessionDataTask *task = [[NSURLSession sharedSession] dataTaskWithRequest:request completio
nHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
        if (error) {
            NSLog(@"获取文件meta信息失败,error : %@", error);
        } else {
            NSHTTPURLResponse *httpResponse = (NSHTTPURLResponse *)response;
            NSString *etag = [httpResponse.allHeaderFields objectForKey:@"Etag"];
            if ([self.checkpoint.etag isEqualToString:etag]) {
                resumable = YES;
            } else {
                resumable = NO;
            }
        }
        dispatch_semaphore_signal(self.semaphore);
    }];
    [task resume];
    dispatch_semaphore_wait(self.semaphore, DISPATCH_TIME_FOREVER);
    return resumable;
}
/**
 * 用于获取本地文件的大小
 */
- (unsigned long long)fileSizeAtPath:(NSString *)filePath {
    unsigned long long fileSize = 0;
```

```
  unsigned long long fileSize = 0;
  NSFileManager *dfm = [NSFileManager defaultManager];
  if ([dfm fileExistsAtPath:filePath]) {
    NSError *error = nil;
    NSDictionary *attributes = [dfm attributesOfItemAtPath:filePath error:&error];
    if (!error && attributes) {
      fileSize = attributes.fileSize;
    } else if (error) {
      NSLog(@"error: %@", error);
    }
  }
  return fileSize;
}
- (void)resume {
  NSURL *url = [NSURL URLWithString:self.requestURLString];
  NSMutableURLRequest *request = [[NSMutableURLRequest alloc]initWithURL:url];
  [request setHTTPMethod:@"GET"];
  BOOL resumable = [self getFileInfo];  // 如果resumable为NO，则证明不能断点续传，否则走续传逻辑。
  if (resumable) {
    self.totalReceivedContentLength = [self fileSizeAtPath:self.targetPath];
    NSString *requestRange = [NSString stringWithFormat:@"bytes=%llu-", self.totalReceivedContentLengt
h];
    [request setValue:requestRange forHTTPHeaderField:@"Range"];
  } else {
    self.totalReceivedContentLength = 0;
  }
  if (self.totalReceivedContentLength == 0) {
    [[NSFileManager defaultManager] createFileAtPath:self.targetPath contents:nil attributes:nil];
  }
  self.dataTask = [self.session dataTaskWithRequest:request];
  [self.dataTask resume];
}
- (void)pause {
  [self.dataTask cancel];
  self.dataTask = nil;
}
- (void)cancel {
  [self.dataTask cancel];
  self.dataTask = nil;
  [self removeFileAtPath: self.targetPath];
}
- (void)removeFileAtPath:(NSString *)filePath {
  NSError *error = nil;
  [[NSFileManager defaultManager] removeItemAtPath:self.targetPath error:&error];
  if (error) {
    NSLog(@"remove file with error : %@", error);
  }
}
#pragma mark - NSURLSessionDataDelegate
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(nullable NSError *)error {
  NSHTTPURLResponse *httpResponse = (NSHTTPURLResponse *)task.response;
  if ([httpResponse isKindOfClass:[NSHTTPURLResponse class]]) {
    if (httpResponse.statusCode == 200) {
      self.checkpoint.etag = [[httpResponse allHeaderFields] objectForKey:@"Etag"];
```

```
        self.checkpoint.totalExpectedLength = httpResponse.expectedContentLength;
      } else if (httpResponse.statusCode == 206) {
        self.checkpoint.etag = [[httpResponse allHeaderFields] objectForKey:@"Etag"];
        self.checkpoint.totalExpectedLength = self.totalReceivedContentLength + httpResponse.expectedCon
tentLength;
      }
    }
    if (error) {
      if (self.failure) {
        NSMutableDictionary *userInfo = [NSMutableDictionary dictionaryWithDictionary:error.userInfo];
        [userInfo oss_setObject:self.checkpoint forKey:@"checkpoint"];
        NSError *tError = [NSError errorWithDomain:error.domain code:error.code userInfo:userInfo];
        self.failure(tError);
      }
    } else if (self.success) {
      self.success(@{@"status": @"success"});
    }
}
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask didReceiveResp
onse:(NSURLResponse *)response completionHandler:(void (^)(NSURLSessionResponseDisposition))comple
tionHandler
{
    NSHTTPURLResponse *httpResponse = (NSHTTPURLResponse *)dataTask.response;
    if ([httpResponse isKindOfClass:[NSHTTPURLResponse class]]) {
      if (httpResponse.statusCode == 200) {
        self.checkpoint.totalExpectedLength = httpResponse.expectedContentLength;
      } else if (httpResponse.statusCode == 206) {
        self.checkpoint.totalExpectedLength = self.totalReceivedContentLength +  httpResponse.expectedCon
tentLength;
      }
    }
    completionHandler(NSURLSessionResponseAllow);
}
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask didReceiveData:
(NSData *)data {
    NSFileHandle *fileHandle = [NSFileHandle fileHandleForWritingAtPath:self.targetPath];
    [fileHandle seekToEndOfFile];
    [fileHandle writeData:data];
    [fileHandle closeFile];
    self.totalReceivedContentLength += data.length;
    if (self.progress) {
      self.progress(data.length, self.totalReceivedContentLength, self.checkpoint.totalExpectedLength);
    }
}
@end
```

以上示例代码展示的是从网络接收数据的处理逻辑，其中：

- DownloadService是下载逻辑的核心。
- 在 URLSession:dataTask:didReceiveData中将接收到的网络数据按照追加的方式写入到文件中，并更新下载进度。
- 在 URLSession:task:didCompleteWithError中判断下载任务是否完成，然后将结果返回给上层业务。
- 在 URLSession:dataTask:didReceiveResponse:completionHandler处理Object相关信息，例如ETag用于

断点续传时进行precheck，content-length用于计算下载进度。

DownloadRequest定义如下：

```
#import <Foundation/Foundation.h>
typedef void(^DownloadProgressBlock)(int64_t bytesReceived, int64_t totalBytesReceived, int64_t totalBytesExpectToReceived);
typedef void(^DownloadFailureBlock)(NSError *error);
typedef void(^DownloadSuccessBlock)(NSDictionary *result);
@interface Checkpoint : NSObject<NSCopying>
@property (nonatomic, copy) NSString *etag;   // 资源的ETag值。
@property (nonatomic, assign) unsigned long long totalExpectedLength;   //文件总大小。
@end
@interface DownloadRequest : NSObject
@property (nonatomic, copy) NSString *sourceURLString;    // 用于下载的URL。
@property (nonatomic, copy) NSString *headURLString;     // 用于获取文件原信息的URL。
@property (nonatomic, copy) NSString *downloadFilePath;   // 文件的本地存储地址。
@property (nonatomic, copy) DownloadProgressBlock downloadProgress; // 下载进度。
@property (nonatomic, copy) DownloadFailureBlock failure;  // 下载成功的回调。
@property (nonatomic, copy) DownloadSuccessBlock success;  // 下载失败的回调。
@property (nonatomic, copy) Checkpoint *checkpoint;      // checkpoint用于存储文件的ETag。
@end
@interface DownloadService : NSObject
+ (instancetype)downloadServiceWithRequest:(DownloadRequest *)request;
/**
 * 启动下载
 */
- (void)resume;
/**
 * 暂停下载
 */
- (void)pause;
/**
 * 取消下载
 */
- (void)cancel;
@end
```

上层业务调用如下：

```
- (void)initDownloadURLs {
  OSSPlainTextAKSKPairCredentialProvider *pCredential = [[OSSPlainTextAKSKPairCredentialProvider alloc] initWithPlainTextAccessKey:OSS_ACCESSKEY_ID secretKey:OSS_SECRETKEY_ID];
  _mClient = [[OSSClient alloc] initWithEndpoint:OSS_ENDPOINT credentialProvider:pCredential];
  // 生成用于get请求的带签名的url
  OSSTask *downloadURLTask = [_mClient presignConstrainURLWithBucketName:@"aliyun-dhc-shanghai" withObjectKey:OSS_DOWNLOAD_FILE_NAME withExpirationInterval:1800];
  [downloadURLTask waitUntilFinished];
  _downloadURLString = downloadURLTask.result;
  // 生成用于head请求的带签名的url
  OSSTask *headURLTask = [_mClient presignConstrainURLWithBucketName:@"aliyun-dhc-shanghai" withObjectKey:OSS_DOWNLOAD_FILE_NAME httpMethod:@"HEAD" withExpirationInterval:1800 withParameters:nil];
  [headURLTask waitUntilFinished];
```

```
  _headURLString = headURLTask.result;
}
- (IBAction)resumeDownloadClicked:(id)sender {
  _downloadRequest = [DownloadRequest new];
  _downloadRequest.sourceURLString = _downloadURLString;    // 设置资源的url
  _downloadRequest.headURLString = _headURLString;
  NSString *documentPath = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomai
nMask, YES).firstObject;
  _downloadRequest.downloadFilePath = [documentPath stringByAppendingPathComponent:OSS_DOWNL
OAD_FILE_NAME];  //设置下载文件的本地保存路径
  __weak typeof(self) wSelf = self;
  _downloadRequest.downloadProgress = ^(int64_t bytesReceived, int64_t totalBytesReceived, int64_t tota
lBytesExpectToReceived) {
    // totalBytesReceived是当前客户端已经缓存了的字节数，totalBytesExpectToReceived是总共需要下载的字节
数。
    dispatch_async(dispatch_get_main_queue(), ^{
      __strong typeof(self) sSelf = wSelf;
      CGFloat fProgress = totalBytesReceived * 1.f / totalBytesExpectToReceived;
      sSelf.progressLab.text = [NSString stringWithFormat:@"%.2f%%", fProgress * 100];
      sSelf.progressBar.progress = fProgress;
    });
  };
  _downloadRequest.failure = ^(NSError *error) {
    __strong typeof(self) sSelf = wSelf;
    sSelf.checkpoint = error.userInfo[@"checkpoint"];
  };
  _downloadRequest.success = ^(NSDictionary *result) {
    NSLog(@"下载成功");
  };
  _downloadRequest.checkpoint = self.checkpoint;
  NSString *titleText = [[_downloadButton titleLabel] text];
  if ([titleText isEqualToString:@"download"]) {
    [_downloadButton setTitle:@"pause" forState: UIControlStateNormal];
    _downloadService = [DownloadService downloadServiceWithRequest:_downloadRequest];
    [_downloadService resume];
  } else {
    [_downloadButton setTitle:@"download" forState: UIControlStateNormal];
    [_downloadService pause];
  }
}
- (IBAction)cancelDownloadClicked:(id)sender {
  [_downloadButton setTitle:@"download" forState: UIControlStateNormal];
  [_downloadService cancel];
}
```

说明 暂停或取消下载时均能从 failure 回调中获取 checkpoint。重启下载时可以将 checkpoint 传到 DownloadRequest，然后 DownloadService 将使用checkpoint 做一致性校验。

10.7.6. 进度条

进度条用于指示上传或下载文件的进度。本文以GetObject接口为例介绍如何打印下载文件（Object）的进度条。

以下代码用于打印从examplebucket下载exampleobject.txt文件的进度条。

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
request.objectKey = @"exampleobject.txt";
// 设置进度回调函数打印进度条。
request.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t totalBytesExpected
ToWrite) {
  // 当前下载长度、当前已下载总长度、待下载的总长度。
  NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
};
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download object success!");
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
```

10.8. 管理文件

10.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 判断文件是否存在
- 获取文件元信息
- 列举文件
- 拷贝文件
- 删除文件

10.8.2. 判断文件是否存在

OSS iOS SDK 提供了方便的同步接口以检测 Bucket 中是否存在指定的文件。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
NSError * error = nil;
BOOL isExist = [client doesObjectExistInBucket:TEST_BUCKET withObjectKey:@"file1m" withError:&error];
if (!error) {
  if(isExist) {
    NSLog(@"File exists.");
  } else {
    NSLog(@"File not exists.");
  }
} else {
  NSLog(@"Error!");
}
```

10.8.3. 获取文件元信息

通过HeadObject方法可以只获取文件元信息而不获取文件的实体。

> **说明**
>
> - 文件元信息（Object Meta）包括HTTP header和自定义元信息，详情请参见开发指南中的文件元信息。
> - iOS SDK当前不支持设置和修改文件元信息。

以下代码用于获取文件元信息：

```
OSSHeadObjectRequest * request = [OSSHeadObjectRequest new];
request.bucketName = @"<bucketName>;
//objectKey表示文件名称，即包含文件后缀在内的完整路径，例如abc/efg/123.jpg。
request.objectKey = @"<objectKey>";
OSSTask * headTask = [client headObject:request];
[headTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"head object success!");
    OSSHeadObjectResult * result = task.result;
    NSLog(@"header fields: %@", result.httpResponseHeaderFields);
    for (NSString * key in result.objectMeta) {
      NSLog(@"ObjectMeta: %@ - %@", key, [result.objectMeta objectForKey:key]);
    }
  } else {
    NSLog(@"head object failed, error: %@" ,task.error);
  }
  return nil;
}];
```

10.8.4. 列举文件

本文介绍如何列举存储空间下（Bucket）中的所有文件（Object）、指定个数的文件、指定前缀的文件等。

说明 关于列举文件的更多信息，请参见Get Bucket (List Objects)。

列举指定个数的文件

以下代码用于列举examplebucket中最多20个文件。

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
// 填写Bucket名称。
getBucket.bucketName = @"examplebucket";
// 填写返回文件的最大个数。如果不设置此参数，则默认值为100，maxkeys的取值不能大于1000。
getBucket.maxKeys = 20;
OSSTask * getBucketTask = [client getBucket:getBucket];
[getBucketTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetBucketResult * result = task.result;
    NSLog(@"get bucket success!");
    for (NSDictionary * objectInfo in result.contents) {
      NSLog(@"list object: %@", objectInfo);
    }
  } else {
    NSLog(@"get bucket failed, error: %@", task.error);
  }
  return nil;
}];
```

列举指定前缀的文件

以下代码用于列举examplebucket中以file为前缀的文件。

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
// 填写Bucket名称。
getBucket.bucketName = @"examplebucket";
// 填写前缀。
getBucket.prefix = @"file";
OSSTask * getBucketTask = [client getBucket:getBucket];
[getBucketTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetBucketResult * result = task.result;
    NSLog(@"get bucket success!");
    for (NSDictionary * objectInfo in result.contents) {
      NSLog(@"list object: %@", objectInfo);
    }
  } else {
    NSLog(@"get bucket failed, error: %@", task.error);
  }
  return nil;
}];
```

列举指定marker之后的文件

以下代码用于列举examplebucket中exampleobject.txt之后的文件。

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
// 填写Bucket名称。
getBucket.bucketName = @"examplebucket";
// 填写marker。从marker之后按字母排序的第一个开始返回文件。
// 如果marker在存储空间中不存在，则会从符合marker字母排序的下一个开始返回文件。
getBucket.marker = @"exampleobject.txt";
OSSTask * getBucketTask = [client getBucket:getBucket];
[getBucketTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetBucketResult * result = task.result;
    NSLog(@"get bucket success!");
    for (NSDictionary * objectInfo in result.contents) {
      NSLog(@"list object: %@", objectInfo);
    }
  } else {
    NSLog(@"get bucket failed, error: %@", task.error);
  }
  return nil;
}];
```

分页列举所有文件

以下代码用于分页列举examplebucket中的所有文件，每页最多返回20个文件。

```
@interface ... {
  NSString *_marker;
  BOOL _isCompleted;
}
@end
@implementation ...
// 分页列举所有文件。
- (void)getAllObjects {
  do {
    OSSTask *task = [self getObjectList];
    // 阻塞等待请求完成获取NextMarker，请求下一页时需要将请求的marker设置为上一页请求返回的NextMarker。第一页无需设置。
    // 示例中通过循环分页列举数据，因此需要阻塞等待请求完成获取NextMarker才能请求下一页数据，实际使用时可根据实际场景判断是否需要阻塞。
    [task waitUntilFinished];
  } while (!_isCompleted);
}
// 列举一页。
- (OSSTask *)getObjectList {
  OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
  // 填写Bucket名称。
  getBucket.bucketName = @"examplebucket";
  // 填写每页返回文件的最大个数。如果不设置此参数，则默认值为100，maxKeys的取值不能大于1000。
  getBucket.maxKeys = 20;
  // marker为全局属性。
  getBucket.marker = self.marker;
  OSSTask * getBucketTask = [client getBucket:getBucket];
  [getBucketTask continueWithBlock:^id(OSSTask *task) {
    if (!task.error) {
      OSSGetBucketResult * result = task.result;
      NSLog(@"get bucket success!");
      for (NSDictionary * objectInfo in result.contents) {
        NSLog(@"list object: %@", objectInfo);
      }
      if (result.isTruncated) {
        _marker = result.nextMarker;
      } else {
        _isCompleted = YES;
      }
    } else {
      _isCompleted = YES;
      NSLog(@"get bucket failed, error: %@", task.error);
    }
    return nil;
  }];
  return getBucketTask;
}
@end
```

分页列举指定前缀的文件

以下代码用于分页列举examplebucket中以file为前缀的文件，每页最多返回20个文件。

```
@interface ... {
  NSString *_marker;
  BOOL _isCompleted;
}
@end
@implementation ...
// 分页列举所有文件。
- (void)getAllObjects {
  do {
    OSSTask *task = [self getObjectList];
    // 阻塞等待请求完成获取NextMarker，请求下一页时需要将请求的marker设置为上一页请求返回的NextMarker。第一页无需设置。
    // 示例中通过循环分页列举数据，因此需要阻塞等待请求完成获取NextMarker才能请求下一页数据，实际使用时可根据实际场景判断是否需要阻塞。
    [task waitUntilFinished];
  } while (!_isCompleted);
}
// 列举一页。
- (OSSTask *)getObjectList {
  OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
  // 填写Bucket名称。
  getBucket.bucketName = @"examplebucket";
  // 填写每页返回文件的最大个数。如果不设置此参数，则默认值为100，maxKeys的取值不能大于1000。
  getBucket.maxKeys = 20;
  // 填写前缀。
  getBucket.prefix = @"file";
  // marker为全局属性。
  getBucket.marker = self.marker;
  OSSTask * getBucketTask = [client getBucket:getBucket];
  [getBucketTask continueWithBlock:^id(OSSTask *task) {
    if (!task.error) {
      OSSGetBucketResult * result = task.result;
      NSLog(@"get bucket success!");
      for (NSDictionary * objectInfo in result.contents) {
        NSLog(@"list object: %@", objectInfo);
      }
      if (result.isTruncated) {
        _marker = result.nextMarker;
      } else {
        _isCompleted = YES;
      }
    } else {
      _isCompleted = YES;
      NSLog(@"get bucket failed, error: %@", task.error);
    }
    return nil;
  }];
  return getBucketTask;
}
@end
```

10.8.5. 拷贝文件

本文介绍如何将一个存储空间（源存储空间）中的文件复制到同一地域下相同或不同存储空间（目标存储空间）中。

拷贝文件时注意事项如下：

- 用户必须有源Object的读权限及目标Bucket的读写权限。
- 不支持跨地域拷贝。例如不能将华东1（杭州）地域存储空间中的文件拷贝到华北1（青岛）地域。

以下代码用于拷贝文件。

```
OSSCopyObjectRequest * copy = [OSSCopyObjectRequest new];
// 填写源Bucket名称。
copy.bucketName = @<bucketName>;
// objectKey等同于objectName，表示从源Bucket拷贝文件时需要指定包含文件后缀在内的完整路径，完整路径中不能包含Bucket名称，例如abc/efg/123.jpg
copy.objectKey = @"<objectKey>";
copy.sourceCopyFrom = [NSString stringWithFormat:@"/%@/%@", @"<bucketName>, @"<objectKey_copyFrom>"];
OSSTask * task = [client copyObject:copy];
[task continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    // ...
  }
  return nil;s
}];
```

10.8.6. 删除文件

本文介绍如何单个或者批量删除文件。

> **警告** 文件一旦删除将无法恢复，请谨慎使用删除操作。

注意事项

删除文件时，您需要具有对Object所在Bucket的写权限。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
OSSDeleteObjectRequest * delete = [OSSDeleteObjectRequest new];
// 填写Bucket名称。
delete.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
delete.objectKey = @"exampleobject.txt";
OSSTask * deleteTask = [client deleteObject:delete];
[deleteTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    // ...
  }
  return nil;
}];
// [deleteTask waitUntilFinished];
```

批量删除文件

批量删除文件时，每次最多删除1000个文件。

返回结果包括如下两种模式，默认返回模式为简单模式，请根据实际选择返回模式。

- 详细模式（verbose）：设置quiet为NO，表示返回所有删除的文件列表。
- 简单模式（quiet）：未设置quiet或者设置quiet为YES，表示只返回删除失败的文件列表。

以下代码用于删除examplebucket中指定的多个文件且只返回删除失败的文件列表。

```
OSSDeleteMultipleObjectsRequest *request = [OSSDeleteMultipleObjectsRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写需要删除的多个Object完整路径。Object完整路径中不能包含Bucket名称。
request.keys = @[@"exampleobject.txt", @"testfolder/sampleobject.txt"];
// 设置为简单模式，只返回删除失败的文件列表。
request.quiet = YES;
OSSTask * deleteMultipleObjectsTask = [client deleteMultipleObjects:request];
[deleteMultipleObjectsTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSDeleteMultipleObjectsResult *result = task.result;
    NSLog(@"delete objects: %@", result.deletedObjects);
  } else {
    NSLog(@"delete objects failed, error: %@", task.error);
  }
  return nil;
}];
```

10.8.7. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于为examplebucket中的exampleobject.txt文件创建名为examplesymlink的软链接。

```
OSSPutSymlinkRequest *request = [OSSPutSymlinkRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写软链接名称。
request.objectKey = @"examplesymlink";
// 填写软链接指定的Object完整路径。Object完整路径中不能包含Bucket名称。
request.targetObjectName = @"exampleobject.txt";
OSSTask *putSymlinkTask = [client putSymlink:request];
[putSymlinkTask continueWithBlock:^id _Nullable(OSSTask * _Nonnull task) {
  if (!task.error) {
    NSLog(@"put symlink success");
  } else {
    NSLog(@"put symlink failed, error: %@", task.error);
  }
  return nil;
}];
```

获取软链接指向的目标文件名称

获取软链接要求您对该软链接具有读权限。

以下代码用于获取examplebucket存储空间中软链接examplesymlink指向的目标文件名称。

```
OSSGetSymlinkRequest *request = [OSSGetSymlinkRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写软链接名称。
request.objectKey = @"examplesymlink";
OSSTask *getSymlinkTask = [client getSymlink:request];
[getSymlinkTask continueWithBlock:^id _Nullable(OSSTask * _Nonnull task) {
  if (!task.error) {
    OSSGetSymlinkResult *result = task.result;
    NSLog(@"get symlink: %@", result.httpResponseHeaderFields[@"x-oss-symlink-target"]);
  } else {
    NSLog(@"get symlink failed, error: %@", task.error);
  }
  return nil;
}];
```

关于获取软链接的更多信息， 请参见GetSymlink。

10.8.8. 解冻归档文件

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。本文介绍如何解冻归档文件。

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。

4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻examplebucket存储空间中的归档文件exampleobject.txt。

```
OSSRestoreObjectRequest *request = [OSSRestoreObjectRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
request.objectKey = @"exampleobject.txt";
OSSTask *restoreObjectTask = [client restoreObject:request];
[restoreObjectTask continueWithBlock:^id _Nullable(OSSTask * _Nonnull task) {
  if (!task.error) {
    NSLog(@"restore object success");
  } else {
    NSLog(@"restore object failed, error: %@", task.error);
  }
  return nil;
}];
```

关于归档存储类型的更多信息，请参见存储类型介绍。关于归档类型的参数的更多信息，请参见API文档RestoreObject。

10.8.9. 管理文件访问权限

文件访问权限包括继承Bucket、私有、公共读和公共读写四种。本文介绍如何获取文件（Object）的访问权限。

文件访问权限

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	public-read
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	public-read-write

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于将examplebucket中exampleobject.txt的文件访问权限设置为私有（private）。

```
OSSPutObjectACLRequest *request = [OSSPutObjectACLRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
request.objectKey = @"exampleobject.txt";
/**
 * 设置权限。
 * public-read：公共读
 * private：私有
 * public-read-write：公共读写
 * default：继承bucket
 */
request.acl = @"private";
OSSTask * putObjectACLTask = [client putObjectACL:request];
[putObjectACLTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"put object ACL success!");
  } else {
    NSLog(@"put object ACL failed, error: %@", task.error);
  }
  return nil;
}];
```

关于设置文件访问权限更多信息，请参见PutObjectACL。

获取文件访问权限

以下代码用于获取examplebucket存储空间中exampleobject.txt文件的访问权限。

```
OSSGetObjectACLRequest *request = [OSSGetObjectACLRequest new];
// 填写Bucket名称。
request.bucketName = @"examplebucket";
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
request.objectName = @"exampleobject.txt";
OSSTask * getObjectACLTask = [client getObjectACL:request];
[getObjectACLTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    OSSGetObjectACLResult *result = task.result;
    NSLog(@"objectACL: %@", result.grant);
  } else {
    NSLog(@"get object ACL failed, error: %@", task.error);
  }
  return nil;
}];
```

关于获取文件访问权限更多信息，请参见GetObjectACL。

10.9. 数据安全性

iOS SDK提供了数据完整性校验来保证您在上传、下载过程中数据的安全性。

上传文件的数据完整性校验

由于移动端网络环境的复杂性，数据在客户端和服务器之间传输时有可能会出错。OSS提供了基于MD5和CRC64的端到端的数据完整性验证功能。

- MD5校验

  需要在上传文件时提供文件的Content-MD5值，OSS服务器会帮助用户进行MD5校验，仅当OSS服务器接收到文件的MD5值和上传提供的MD5一致时才可以上传成功，以此保证上传数据的完整性。

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = BUCKET_NAME;
...
request.contentMd5 = [base64Md5ForFilePath];
```

- CRC校验

  与MD5相比，CRC64可以在文件上传的同时计算CRC值。

```
// 构造上传请求。
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = BUCKET_NAME;
///....
request.crcFlag = OSSRequestCRCOpen;
// 开启CRC校验。
OSSTask * task = [_client putObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
  // 如果在传输过程中数据不一致，则CRC64校验失败，并提示OSSClientErrorCodeInvalidCRC错误。
  XCTAssertNil(task.error);
  return nil;
}] waitUntilFinished];
```

下载文件的数据完整性校验

iOS SDK在下载过程中提供了基于CRC的端到端的数据完整性效验功能。

在读取数据流时，如果开启了CRC校验，会在数据流读取完成时自动验证数据完整性。

以下代码用于开启CRC校验：

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = BUCKET_NAME;
// 开启CRC校验。
request.crcFlag = OSSRequestCRCOpen;
OSSTask * task = [testProxyClient getObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
  // 开启CRC校验后，如果在传输过程中出现数据错误，则提示OSSClientErrorCodeInvalidCRC。
  XCTAssertNil(task.error);
  return nil;
}] waitUntilFinished];
// 开启CRC校验后，如果设置了onReceiveData block，需自行比较CRC值是否一致。
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = BUCKET_NAME;
request.crcFlag = OSSRequestCRCOpen;
....
__block uint64_t localCrc64 = 0;
NSMutableData *receivedData = [NSMutableData data];
request.onRecieveData = ^(NSData *data) {
  if (data)
  {
    NSMutableData *mutableData = [data mutableCopy];
    void *bytes = mutableData.mutableBytes;
    localCrc64 = [OSSUtil crc64ecma:localCrc64 buffer:bytes length:data.length];
    [receivedData appendData:data];
  }
};
__block uint64_t remoteCrc64 = 0;
OSSTask * task = [_client getObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
  XCTAssertNil(task.error);
    OSSGetObjectResult *result = task.result;
  if (result.remoteCRC64ecma)
  {
    NSScanner *scanner = [NSScanner scannerWithString:result.remoteCRC64ecma];
    [scanner scanUnsignedLongLong:&remoteCrc64];
    if (remoteCrc64 == localCrc64)
    {
      NSLog(@"crc64校验成功!");
    }
    else
    {
      NSLog(@"crc64校验失败!");
    }
  }
  return nil;
}] waitUntilFinished];
```

10.10. 签名URL

OSS iOS SDK 支持签名特定有效时长或者公开的 URL，用于转给第三方实现授权访问。

签名私有资源指定有效时长的访问URL

如果 Bucket 或 Object 为私有，您需要调用以下接口获取签名后的 URL：

```
NSString * constrainURL = nil;
// sign constrain url
OSSTask * task = [client presignConstrainURLWithBucketName:@"<bucket name>"
                                withObjectKey:@"<object key>"
                        withExpirationInterval: 30 * 60];
if (!task.error) {
  constrainURL = task.result;
} else {
  NSLog(@"error: %@", task.error);
}
```

签名公开的访问URL

如果 Bucket 或 Object 为公共可读，您需要调用以下接口获取可公开访问 Object 的URL：

```
NSString * publicURL = nil;
// sign public url
task = [client presignPublicURLWithBucketName:@"<bucket name>"
     withObjectKey:@"<object key>"];
if (!task.error) {
 publicURL = task.result;
} else {
 NSLog(@"sign url error: %@", task.error);
}
```

10.11. 授权访问

iOS SDK提供了STS鉴权模式和自签名模式来保障移动终端的安全性。

背景信息

无论是STS鉴权模式还是自签名模式，您实现的回调函数都需要保证调用时的返回结果。如果您需要实现向业务Server获取Token、Signature的网络请求，建议调用网络库的同步接口。回调都是在SDK发起具体请求时，在请求的子线程中执行，所以不会阻塞主线程。

STS鉴权模式

> **说明** 使用STS模式授权需要先开通阿里云访问控制RAM服务。

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。
- 设置StsToken

您可以在App中，预先通过某种方式（如通过网络请求从您的业务Server上）获取StsToken，然后用StsToken初始化SDK。通过这种方式获取StsToken时需要注意StsToken的过期时间，并在StsToken即将过期时手动更新StsToken到SDK中。

初始化SDK示例代码如下：

> **说明** 如果您需要使用OSSAuthCredentialProvider初始化SDK，请参见初始化。

```
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentialProvider alloc] initWithAccessKeyId:@"<StsToken.AccessKeyId>" secretKeyId:@"<StsToken.SecretKeyId>" securityToken:@"<StsToken.SecurityToken>"];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

当判断StsToken即将过期时，您可以重新构造OSSClient，也可以通过如下方式更新CredentialProvider：

```
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentialProvider alloc] initWithAccessKeyId:@"<StsToken.AccessKeyId>" secretKeyId:@"<StsToken.SecretKeyId>" securityToken:@"<StsToken.SecurityToken>"];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

- 实现获取StsToken回调

如果您期望SDK自动更新StsToken，那么您需要在SDK的应用中实现回调。通过您实现回调的方式去获取Federation Token（即StsToken），SDK会使用此StsToken来进行加签处理，并在需要更新时主动调用此回调来获取StsToken：

```
id<OSSCredentialProvider> credential = [[OSSFederationCredentialProvider alloc] initWithFederationTokenGetter:^OSSFederationToken * {
  // 您需要在此处实现获取一个FederationToken，并构造成OSSFederationToken对象返回。
  // 如果由于某种原因获取失败，直接返回nil。
   OSSFederationToken * token;
  // 从您的服务器中获取token。
  ...
  return token;
}];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

> **说明** 此外，如果您已经通过别的方式获取了StsToken所需的各个字段，也可以在回调中直接返回StsToken。但您需要手动处理StsToken的更新，且更新后重新设置该OSSClient实例的OSSCredentialProvider。

使用示例：

假设您搭建的server地址为http://localhost:8080/distribute-token.json，并且要访问该地址，则返回数据的格式如下：

```
{
  "StatusCode": 200,
  "AccessKeyId":"STS.iA645eTOXEqP3cg3****",
  "AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojU****",
  "Expiration":"2015-11-03T09:52:59Z",
  "SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS/p3OgaUC+oJl3XSlbJ7StKpQ**
**"
}
```

实现 `OSSFederationCredentialProvider` 的示例如下：

```
id<OSSCredentialProvider> credential2 = [[OSSFederationCredentialProvider alloc] initWithFederationTo
kenGetter:^OSSFederationToken * {
  // 构造请求访问您的业务Server。
  NSURL * url = [NSURL URLWithString:@"http://localhost:8080/distribute-token.json"];
  NSURLRequest * request = [NSURLRequest requestWithURL:url];
  OSSTaskCompletionSource * tcs = [OSSTaskCompletionSource taskCompletionSource];
  NSURLSession * session = [NSURLSession sharedSession];
  // 发送请求。
  NSURLSessionTask * sessionTask = [session dataTaskWithRequest:request
                                completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {
                                  if (error) {
                                    [tcs setError:error];
                                    return;
                                  }
                                  [tcs setResult:data];
                                }];
  [sessionTask resume];
  // 需要阻塞等待请求返回。
  [tcs.task waitUntilFinished];
  // 解析结果。
  if (tcs.task.error) {
    NSLog(@"get token error: %@", tcs.task.error);
    return nil;
  } else {
    // 返回数据为JSON格式，需要解析返回数据得到token的各个字段。
    NSDictionary * object = [NSJSONSerialization JSONObjectWithData:tcs.task.result
                                        options:kNilOptions
                                         error:nil];
    OSSFederationToken * token = [OSSFederationToken new];
    token.tAccessKey = [object objectForKey:@"AccessKeyId"];
    token.tSecretKey = [object objectForKey:@"AccessKeySecret"];
    token.tToken = [object objectForKey:@"SecurityToken"];
    token.expirationTimeInGMTFormat = [object objectForKey:@"Expiration"];
    NSLog(@"get token: %@", token);
    return token;
  }
}];
```

自签名模式

您可以把AccessKey ID和AccessKey Secret保存在自有服务器中后，通过自有服务器对客户端信息进行签名。具体步骤如下：

1. 在客户端获取并发送待签名的字符串到自有服务器。
   i. 构造请求时通过SDK中OSSCustomSignerCredentialProvider的signContent方法获取待签名的字符串。
   ii. 发送待签名的字符串到自有服务器。
2. 在自有服务器进行签名并返回签名后的字符串到客户端。
   i. 按照OSS规定的签名算法对待签名字符串进行签名。有关签名算法的更多信息，请参见在Header中包含签名。

      签名算法的格式为 `signature = "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeySecret, content))` ，其中content是已根据请求参数拼接后的字符串。示例代码如下：

```
id<OSSCredentialProvider> credential = [[OSSCustomSignerCredentialProvider alloc] initWithImplementedSigner:^NSString *(NSString *contentToSign, NSError *__autoreleasing *error) {
  // 按照OSS规定的签名算法加签字符串，并将得到的加签字符串拼接AccessKeyId后返回。
  // 将加签的字符串传给您的服务器，然后返回签名。
  // 如果因某种原因加签失败，服务器描述错误信息后返回nil。
NSString *signature = [OSSUtil calBase64Sha1WithData:contentToSign withSecret:@"<your accessKeySecret>"]; // 此处为用SDK内的工具函数进行本地加签，建议您通过业务server实现远程加签。
  if (signature != nil) {
    *error = nil;
  } else {
    *error = [NSError errorWithDomain:@"<your domain>" code:-1001 userInfo:@"<your error info>"];
    return nil;
  }
  return [NSString stringWithFormat:@"OSS %@:%@", @"<your accessKeyId>", signature];
}];
client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

   ii. 返回签名后的字符串给客户端。
3. 在客户端发送签名后的字符串到OSS服务器进行鉴权。

10.12. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理支持的参数请参见处理参数。

使用方法

图片处理使用标准的HTTP GET请求。您可以在URL的QueryString中设置处理参数。

如果图片文件的访问权限为私有读写，必须通过授权才能进行访问。

- 匿名访问

```
OSSTask * task = [_client presignPublicURLWithBucketName:_publicBucketName
                    withObjectKey:@"hasky.jpeg"
                   withParameters:@{@"x-oss-process": @"image/resize,w_50"}];
```

- 授权访问

SDK中使用图片处理时，只需要在下载图片时调用 `request.setxOssProcess()` 方法设置处理参数。示例如下：

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"example.jpg";
// 图片处理。
request.xOssProcess = @"image/resize,m_lfit,w_100,h_100";
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
  if (!task.error) {
    NSLog(@"download image success!");
    OSSGetObjectResult * getResult = task.result;
    NSLog(@"download image data: %@", getResult.dowloadedData);
  } else {
    NSLog(@"download object failed, error: %@" ,task.error);
  }
  return nil;
}];
// [getTask waitUntilFinished];
// [request cancel];
```

- SDK访问

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"bucket-name";   // 设置Bucket名称。
request.objectKey = @"image-name";    // 设置图片名称。
request.xOssProcess = @"image/resize,m_lfit,w_100,h_100";  // 将图片缩放为固定宽高100 px。
OSSTask * task = [ossClient getObject:request];
```

图片处理持久化

以下代码用于图片处理持久化：

```
OSSImagePersistRequest *request = [OSSImagePersistRequest new];
request.fromBucket = _privateBucketName;
request.fromObject = OSS_IMAGE_KEY;
request.toBucket = _privateBucketName;
request.toObject = @"image_persist_key";
request.action = @"image/resize,w_100";
//request.action = @"resize,w_100";
[[[ossClient imageActionPersist:request] continueWithBlock:^id _Nullable(OSSTask * _Nonnull task) {
  return nil;
}] waitUntilFinished];
```

图片处理工具

可视化图片处理工具 ImageStyleViewer ，可以直观的看到OSS图片处理的结果。

10.13. 异常响应

iOS SDK 中发生的异常分为两类：Client Error 和 ServerError。

Client Error 指参数错误、网络错误等。ServerError 指 OSS Server 返回的异常响应。

Error类型	Error Domain	Code	UserInfo	描述	解决方法
ClientError	com.aliyun.oss.clientError	0	OSSClientErrorCodeNetworkingFailWithResponseCode0	连接异常	请检查网络连接后重试。
		1	OSSClientErrorCodeSignFailed	签名失败	请参见签名常见问题进行排查。
		2	OSSClientErrorCodeFileCantWrite	文件无法写入	可能是指定的断点记录文件的路径或者下载的文件路径不合法。请修改对应的文件路径后重试。
		3	OSSClientErrorCodeInvalidArgument	参数非法	参数格式不符合要求，请参见API概览中相应的API，填写正确的参数格式。
		4	OSSClientErrorCodeNilUploadid	未获取到断点续传任务的uploadId	检查参数，例如objectMeta无误后，请尝试重新获取uploadId。
		5	OSSClientErrorCodeTaskCancelled	任务被取消	请检查代码中任务取消逻辑是否正确，或网络连接是否异常。
		6	OSSClientErrorCodeNetworkError	网络异常	请检查网络连接后重试。
		7	OSSClientErrorCodeInvalidCRC	CRC校验失败	传输过程中数据不一致。请检查文件是否被修改。
		8	OSSClientErrorCodeCannotResumeUpload	断点续传上传失败，无法继续上传	上传过程中文件发生了更改、导致文件大小不一致。因此文件上传过程中请勿修改文件。
		9	OSSClientErrorCodeExcepetionCatched	异常捕获	请结合具体的报错信息进行排查。
ServerError	com.aliyun.oss.serverError	(-1 * httpResponse.statusCode)	dict	解析响应XML得到的Dictionary	可能是服务端遇到了错误无法完成请求，请参见错误响应进行排查。

11.C++

11.1. 前言

本文档基于C++ SDK 1.8.2版本编写。

SDK 源码

SDK源码请参见 GitHub。

示例代码

OSS C++ SDK提供丰富的示例代码，方便您参考或直接使用。示例代码包括以下内容：

示例文件	示例内容
PutObjectFromFile	上传文件
GetObjectToFile	下载文件
AppendObject	追加上传
MultipartUploadObject	分片上传
HeadObject	管理文件元信息
ListObjects	列举文件
CopyObject	拷贝文件
DeleteObject	删除文件
PutObjectProgress	进度条上传
DownloadObjectProcess	进度条下载
RestoreArchiveObject	解冻归档文件
BucketSample.cc	设置存储空间的静态网站托管、CORS、生命周期、访问日志、授权访问等
ObjectTrafficLimitTest.cc	设置上传、下载文件时的单链接限速
ObjectRequestPaymentTest.cc	存储空间的请求者付费模式
ObjectTaggingTest.cc	设置、获取、删除对象标签以及对象标签和生命周期管理

11.2. 安装

本文介绍如何在不同的操作系统中安装OSS C++ SDK以及编译选项的使用说明。

前提条件

- C++11及以上版本的编译器
- Visual Studio 2013及以上版本
- GCC 4.8及以上版本
- Clang 3.3及以上版本

下载SDK

- 下载SDK安装包
- 通过GitHub下载

安装SDK

您可以通过Linux 、Windows 、Android及macOS系统安装SDK。

- Linux系统
  i. 安装CMake。

     请安装CMake3.1及以上版本。打开SDK安装包，通过CMake编译生成所需文件。

     编译命令如下：

     ```
     cd <path/to/aliyun-oss-cpp-sdk>
     mkdir build
     cd build
     cmake ..
     ```

  ii. 安装第三方库libcurl、openssl。

     RedHat或Centos：

     ```
     yum –y install libcurl-devel openssl-devel
     ```

     Fedora：

     ```
     sudo dnf install libcurl-devel openssl-devel
     ```

  iii. 安装SDK。

     ```
     make && make install
     ```

  > **注意** C++ SDK默认关闭rtti属性。因此使用g++编译运行时，请添加-std=c++11、-fno-rtti、-lalibabacloud-oss-cpp-sdk、-lcurl、-lcrypto和-lpthread。

  示例如下：

  ```
  g++ test.cpp -std=c++11 -fno-rtti -lalibabacloud-oss-cpp-sdk -lcurl -lcrypto -lpthread -o test.bin
  ```

- Windows系统：
  i. 安装CMake。

     打开cmd进入SDK文件目录，创建build文件夹，运行**cmake ..**生成所需文件，如下图所示。

```
D:\Item\C++\aliyun-oss-cpp-sdk\build>cmake ..
-- Selecting Windows SDK version 10.0.17134.0 to target Windows 6.1.7601.
-- Project version: 1.1.0
-- TARGET_OS: WINDOWS
-- Configuring done
-- Generating done
-- Build files have been written to: D:/Item/C++/aliyun-oss-cpp-sdk/build
```

说明

- 下载的SDK中不直接提供alibabacloud-oss-cpp-sdk.sln工程文件，您需要通过cmake生成所需的工程文件。
- 如果要构建x64体系结构，可以使用命令cmake -A x64 ..来实现。

ii. 请以管理员身份运行VS开发人员命令提示符，在build目录文件下运行以下命令进行编译安装。

```
msbuild ALL_BUILD.vcxproj
msbuild INSTALL.vcxproj
```

或者用Visual Studio打开alibabacloud-oss-cpp-sdk.sln生成解决方案。

- Android系统

  Linux环境下，基于android-ndk-r16工具链构建工程。请在*$ANDROID_NDK/sysroot*路径下安装libcurl和libopenssl第三方库，并运行以下命令进行编译安装。

```
cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build/cmake/android.toolchain.cmake \
    -DANDROID_NDK=$ANDROID_NDK   \
    -DANDROID_ABI=armeabi-v7a   \
    -DANDROID_TOOLCHAIN=clang   \
    -DANDROID_PLATFORM=android-21 \
    -DANDROID_STL=c++_shared ..
make
```

- macOS系统

  在macOS系统中，您可以使用brew方式来安装依赖库。

  在macOS系统中需要指定OpenSSL安装路径。假设OpenSSL安装在*/usr/local/Cellar/openssl/1.0.2p*目录中，运行以下命令进行编译安装。

```
cmake -DOPENSSL_ROOT_DIR=/usr/local/Cellar/openssl/1.0.2p \
    -DOPENSSL_LIBRARIES=/usr/local/Cellar/openssl/1.0.2p/lib \
    -DOPENSSL_INCLUDE_DIRS=/usr/local/Cellar/openssl/1.0.2p/include/ ..
make
```

编译选项

您可以根据具体的业务场景，设置不同的编译选项，编译选项格式为cmake -D${OptionName}=ON|OFF ..。

可选的编译选项请参见下表。

选项名称	描述
BUILD_SHARED_LIBS	构建动态库，默认值为OFF。打开此选项时，会同时构建静态库和动态库， 且静态库名字增加-static后缀。 结合编译选项格式，如果要构建动态库，则使用**cmake -DBUILD_SHARED_LIBS=ON ..**。
ENABLE_RTTI	支持的实时类型信息，默认值为ON。
OSS_DISABLE_BUCKET	不构建Bucket相关设置接口，默认值为OFF。如果需要裁剪代码大小，可将其置为ON。
OSS_DISABLE_LIVECHANNEL	不构建LiveChannel接口，默认值为OFF。如果需要裁剪代码大小，可将其置为ON。
OSS_DISABLE_RESUAMABLE	不构建断点续传接口，默认值为OFF。如果需要裁剪代码大小，可将其置为ON。
OSS_DISABLE_ENCRYPTION	不构建客户端加密接口，默认值为OFF。如果需要裁剪代码大小，可将其置为ON。

11.3. 初始化

OssClient用于管理存储空间（Bucket）和文件（Object）等OSS资源。使用C++ SDK发起OSS请求时，您需要初始化一个OssClient实例，并根据需要修改ClientConfiguration的默认配置项。

配置OssClient

ClientConfiguration是OssClient的配置类，您可通过此配置类来配置代理、连接超时、最大连接数等参数。可设置的参数如下：

参数	描述
userAgent	用户代理，指HTTP的User-Agent头。默认代理为aliyun-sdk-cpp/1.X.X。
maxConnections	连接池数。默认为16个。
requestTimeoutMs	请求超时时间。请求超时没有收到数据将会关闭连接，默认为10000ms。
connectTimeoutMs	建立连接的超时时间。默认为5000ms。
retryStrategy	自定义失败重试策略。
proxyScheme	代理协议，默认为HTTP。
proxyPort	代理服务器端口。
proxyPassword	代理服务器验证的密码。
proxyUserName	代理服务器验证的用户名。

参数	描述
verifySSL	是否开启SSL证书校验，默认关闭。 说明 C++ SDK 1.8.2及以上版本默认开启SSL证书校验。
caPath	用于设置CA证书根路径，当verifySSL为true时有效，默认为空。
caFile	用于设置CA证书路径，当verifySSL为true时有效，默认为空。
enableCrc64	是否开启CRC64校验，默认开启。
enableDateSkewAdjustment	是否开启HTTP请求时间自动修正，默认开启。
sendRateLimiter	上传限速（单位KB/s）。
recvRateLimiter	下载限速（单位KB/s）。

设置超时时间

以下代码用于设置超时时间：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  /* 设置连接池数，默认为16个。*/
  conf.maxConnections = 20;
  /* 设置请求超时时间，超时没有收到数据就关闭连接，默认为10000ms。*/
  conf.requestTimeoutMs = 8000;
  /* 设置建立连接的超时时间，默认为5000ms。*/
  conf.connectTimeoutMs = 8000;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

设置SSL证书校验

C++ SDK 1.8.2及以上的版本默认自动开启SSL证书校验。如果出现SSL证书校验失败，您需要根据实际情况设置正确的证书路径，或者关闭SSL证书校验。

以下代码用于设置SSL证书校验：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  /* 设置SSL证书校验开关，默认为true，即开启证书校验。*/
  conf.verifySSL = true;
  /* 设置CA证书根路径，当verifySSL为true时有效，默认为空。*/
  conf.caPath = "/etc/ssl/certs/";
  /* 设置CA证书路径，当verifySSL为true时有效，默认为空。*/
  conf.caFile = "/etc/ssl/certs/ca-certificates.crt";;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

设置限速处理

以下代码用于设置上传或下载限速：

```
#include <alibabacloud/oss/OssClient.h>
#include <alibabacloud/oss/client/RateLimiter.h>
using namespace AlibabaCloud::OSS;
class  UserRateLimiter : public RateLimiter
{
public:
   DefaultRateLimiter() :rate_(0) {};
   ~DefaultRateLimiter() {};
   virtual void setRate(int rate) { rate_ = rate; };
   virtual int Rate() const { return rate_; };
private:
   int rate_;
};
int main(void)
{
   /* 初始化OSS账号信息。*/
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName";
   /* 初始化网络等资源。*/
   InitializeSdk();
   ClientConfiguration conf;
   auto sendrateLimiter = std::make_shared<UserRateLimiter>();
   auto recvrateLimiter = std::make_shared<UserRateLimiter>();
   conf.sendRateLimiter = sendrateLimiter;
   conf.recvRateLimiter = recvrateLimiter;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 设置下载限速（单位KB/s）。*/
   recvrateLimiter->setRate(256);
   /* 设置上传限速（单位KB/s）。*/
   sendrateLimiter->setRate(256);
   /* 上传文件。*/
   auto outcome = client.PutObject(BucketName, ObjectName,"yourLocalFilename");
   /* 上传过程中更新上传限速（单位KB/s）。*/
   sendrateLimiter->setRate(300);
   /* 释放网络等资源。*/
   ShutdownSdk();
   return 0;
}
```

设置重试策略

以下代码用于设置重试策略：

```
#include <alibabacloud/oss/OssClient.h>
#include <alibabacloud/oss/client/RetryStrategy.h>
using namespace AlibabaCloud::OSS;
class UserRetryStrategy : public RetryStrategy
{
public:
   /* maxRetries表示最大重试次数，scaleFactor为重试等待时间的尺度因子。*/
   UserRetryStrategy(long maxRetries = 3, long scaleFactor = 300) :
```

```
  UserRetryStrategy(long maxRetries = 3, long scaleFactor = 300) :
    m_scaleFactor(scaleFactor), m_maxRetries(maxRetries)
  {}
  /* 您可以自定义shouldRetry函数，该函数用于判断是否进行重试。*/
  bool shouldRetry(const Error & error, long attemptedRetries) const;
  /* 您可以自定义calcDelayTimeMs函数，该函数用于计算重试的延迟等待时间。*/
  long calcDelayTimeMs(const Error & error, long attemptedRetries) const;
private:
  long m_scaleFactor;
  long m_maxRetries;
};
bool UserRetryStrategy::shouldRetry(const Error & error, long attemptedRetries) const
{
  if (attemptedRetries >= m_maxRetries)
    return false;
  long responseCode = error.Status();
  //http code
  if ((responseCode == 403 && error.Message().find("RequestTimeTooSkewed") != std::string::npos) ||
    (responseCode > 499 && responseCode < 599)) {
    return true;
  }
  else {
    switch (responseCode)
    {
    //curl error code
    case (ERROR_CURL_BASE + 7):  //CURLE_COULDNT_CONNECT
    case (ERROR_CURL_BASE + 18): //CURLE_PARTIAL_FILE
    case (ERROR_CURL_BASE + 23): //CURLE_WRITE_ERROR
    case (ERROR_CURL_BASE + 28): //CURLE_OPERATION_TIMEDOUT
    case (ERROR_CURL_BASE + 52): //CURLE_GOT_NOTHING
    case (ERROR_CURL_BASE + 55): //CURLE_SEND_ERROR
    case (ERROR_CURL_BASE + 56): //CURLE_RECV_ERROR
      return true;
    default:
      break;
    };
  }
  return false;
}
long UserRetryStrategy::calcDelayTimeMs(const Error & error, long attemptedRetries) const
{
  UNUSED_PARAM(error);
  return (1 << attemptedRetries) * m_scaleFactor;
}
int main(void)
{
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  /* 设置失败请求重试次数，默认为3次。*/
  auto defaultRetryStrategy = std::make_shared<UserRetryStrategy>(5);
  conf.retryStrategy = defaultRetryStrategy;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 释放网络等资源。*/
  ShutdownSdk();
```

```
  return 0;
}
```

11.4. 快速入门

本节介绍如何快速使用OSS C++ SDK完成常见操作，如创建存储空间（Bucket）、上传文件、下载文件（Object）等。

> **说明** 示例代码中的conf为ClientConfiguration的实例，ClientConfiguration是OssClient的配置类，您可以通过此配置类来配置代理、连接超时、最大连接数等参数。更多信息，请参见初始化。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。

> **说明** 关于存储空间命名规范的更多信息，请参见基本概念中的命名规范。关于获取Endpoint的更多信息，请参见访问域名和数据中心。

以下代码用于创建examplebucket存储空间。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 指定新创建Bucket的名称、存储类型和ACL */
  CreateBucketRequest request(BucketName, StorageClass::IA, CannedAccessControlList::PublicReadWrite);
  /* 创建Bucket */
  auto outcome = client.CreateBucket(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CreateBucket fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

关于创建存储空间的更多信息，请参见控制台用户指南中的创建存储空间。

上传文件

以下代码用于上传文件到OSS。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称 */
  std::string ObjectName = "exampledir/exampleobject.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 上传文件 */
  /* 填写本地文件完整路径，例如D:\\localpath\\examplefile.txt，其中localpath为本地文件examplefile.txt所在本地路径 */
  auto outcome = client.PutObject(BucketName, ObjectName,"D:\\localpath\\examplefile.txt");
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

下载文件

以下代码用于将指定的OSS文件下载到本地文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称 */
  std::string ObjectName = "exampledir/exampleobject.txt";
  /* 填写本地文件完整路径，例如D:\\localpath\\examplefile.txt，其中localpath为本地文件examplefile.txt所在本地路径 */
  std::string FileNametoSave = "D:\\localpath\\examplefile.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取文件到本地文件 */
  GetObjectRequest request(BucketName, ObjectName);
  request.setResponseStreamFactory([=]() {return std::make_shared<std::fstream>(FileNametoSave, std::ios_base::out | std::ios_base::in | std::ios_base::trunc| std::ios_base::binary); });
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
    std::cout << "GetObjectToFile success, ContentLength is" << outcome.result().Metadata().ContentLength() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetObjectToFile fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

列举文件

以下代码用于列举指定存储空间下的文件。默认列举100个文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 列举文件 */
  ListObjectsRequest request(BucketName);
  auto outcome = client.ListObjects(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "ListObjects fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  else {
    for (const auto& object : outcome.result().ObjectSummarys()) {
      std::cout << "object"<<
      ",name:" << object.Key() <<
      ",size:" << object.Size() <<
      ",last modified time:" << object.LastModified() << std::endl;
    }
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

删除文件

以下代码用于删除指定文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称 */
  std::string ObjectName = "exampledir/exampleobject.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  DeleteObjectRequest request(BucketName, ObjectName);
  /* 删除文件 */
  auto outcome = client.DeleteObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "DeleteObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

关于删除文件的更多信息，请参见管理文件中的删除文件。

11.5. 存储空间

11.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*指定新创建bucket的名称、存储类型和ACL*/
  CreateBucketRequest request(BucketName, StorageClass::IA, CannedAccessControlList::PublicReadWrite);
  /*设置同城冗余存储属性*/
  //request.setDataRedundancyType(DataRedundancyType::ZRS);
  /*创建bucket*/
  auto outcome = client.CreateBucket(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CreateBucket fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

11.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

以下代码用于列举所有存储空间。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 列举存储空间 */
  ListBucketsRequest request;
  auto outcome = client.ListBuckets(request);
  if (outcome.isSuccess()) {
    /* 打印存储空间信息 */
    std::cout <<" success, and bucket count is" << outcome.result().Buckets().size() << std::endl;
    std::cout << "Bucket name is" << std::endl;
    for (auto result : outcome.result().Buckets())
    {
      std::cout << result.Name() << std::endl;
    }
  }
  else {
    /* 异常处理 */
    std::cout << "ListBuckets fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.5.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 判断存储空间是否存在 */
  auto outcome = client.DoesBucketExist(BucketName);
  if (outcome) {
    std::cout << " The Bucket exists" << std::endl;
  }
  else {
    std::cout << "The Bucket does not exist" << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.5.4. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取存储空间地域 */
  GetBucketLocationRequest request(BucketName);
  auto outcome = client.GetBucketLocation(request);
  if (outcome.isSuccess()) {
    std::cout << "getBucketLocation success, location: " << outcome.result().Location() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "getBucketLocation fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

有关地域的详细信息，请参见开发指南基本概念中的地域介绍以及GetBucketLocation API。

11.5.5. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期等。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取存储空间信息*/
  GetBucketInfoRequest request(BucketName);
  auto outcome = client.GetBucketInfo(request);
  /*查看存储空间冗余类型*/
  if (outcome.isSuccess()) {
    std::cout << "GetBucketInfo success, DataRedundancyType:" << outcome.result().DataRedundancyType(
) << std::endl;
  }
  else {
    /*异常处理*/
    std::cout << "GetBucketInfo fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

11.5.6. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间的访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	OSS_ACL_PRIVATE

访问权限	描述	访问权限值
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ_WRITE

更多关于访问权限的内容请参见开发指南中的访问控制。

以下代码用于设置存储空间的访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置存储空间访问权限 */
  SetBucketAclRequest request(BucketName, CannedAccessControlList::Private);
  auto outcome = client.SetBucketAcl(request);
  if (outcome.isSuccess()) {
    std::cout << " setBucketAcl to private success " << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "SetBucketAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取存储空间访问权限 */
  GetBucketAclRequest request(BucketName);
  auto outcome = client.GetBucketAcl(request);
  if (outcome.isSuccess()) {
    std::cout << "getBucketAcl success, acl: " << outcome.result().Acl() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "getBucketAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

11.5.7. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
>
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用ListMultipartUploads列举出所有碎片，然后使用AbortMultipartUpload删除这些碎片。

以下代码用于删除存储空间：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 指定要删除的bucket名称 */
  DeleteBucketRequest request(BucketName);
  /* 删除bucket */
  auto outcome = client.DeleteBucket(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "DeleteBucket fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

删除存储空间的更多详情，请参见DeleteBucket。

11.5.8. 存储空间标签

您可以通过存储空间（Bucket）的标签功能， 对Bucket进行分类管理，如List Bucket时只显示带有指定标签的Bucket。

注意事项

使用Bucket标签功能时，有如下注意事项：

- 只有Bucket的拥有者及授权的RAM用户才能为Bucket设置标签，否则返回403 Forbidden错误，错误码：AccessDenied。
- 最多可设置20对Bucket标签（Key-Value对）。
  - Key最大长度为64字节，不能以 `http://` 、 `https://` 、 `Aliyun` 为前缀，且不能为空。
  - Value最大长度为128字节，允许为空。
  - Key和Value必须为UTF-8编码。
- PutBucketTags是覆盖语义，即新设置的标签完全覆盖已有的标签。

设置Bucket标签

以下代码用于设置Bucket标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*设置Bucket标签*/
  SetBucketTaggingRequest request(BucketName);
  Tag tag1("yourTagkey1","yourTagValue1");
  Tag tag2("yourTagkey2", "yourTagValue2");
  TagSet tagset;
  tagset.push_back(tag1);
  tagset.push_back(tag2);
  Tagging taging;
  taging.setTags(tagset);
  request.setTagging(taging);
  auto outcome = client.SetBucketTagging(request);
  if (outcome.isSuccess()) {
      std::cout << " SetBucketTagging success " << std::endl;
  }
  else {
    /*异常处理*/
    std::cout << "SetBucketTagging fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

获取Bucket标签

以下代码用于获取Bucket标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取Bucket标签*/
  GetBucketTaggingRequest request(BucketName);
  auto outcome = client.GetBucketTagging(request);
  if (outcome.isSuccess()) {
    std::cout << " GetBucketTagging success " << std::endl;
    for (const auto& tag : outcome.result().Tagging().Tags()) {
      std::cout << "tag key:" << tag.Key() <<
      "tag value:" << tag.Value() << std::endl;
    }
  }
  else {
    /*异常处理*/
    std::cout << "GetBucketTagging fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

列举带指定标签的Bucket

以下代码用于列举带指定标签的Bucket：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*列举带指定标签的Bucket*/
  ListBucketsRequest request;
  Tag tag1("yourTagkey1","yourTagValue1");
  request.setTag(tag1);
  auto outcome = client.ListBuckets(request);
  if (outcome.isSuccess()) {
    std::cout << "ListBuckets success" << std::endl;
    for (const auto& bucket : outcome.result().Buckets()) {
      std::cout << "bucket name:" << bucket.Name() << std::endl;
    }
  }
  else {
    /*异常处理*/
    std::cout << "ListBuckets fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

删除Bucket所有标签

以下代码用于删除Buckey的所有标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*删除Bucket标签*/
  DeleteBucketTaggingRequest request(BucketName);
  auto outcome = client.DeleteBucketTagging(request);
  if (outcome.isSuccess()) {
      std::cout << " DeleteBucketTagging success " << std::endl;
  }
  else {
    /*异常处理*/
    std::cout << "DeleteBucketTagging fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

删除Bucket指定标签

以下代码用于删除Bucket指定标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /*指定key删除Bucket标签*/
    DeleteBucketTaggingRequest request(BucketName);
    Tagging tagging;
    TagSet tagset;
    Tag tag("project","projectone");
    tagset.push_back(tag);
    tagging.setTags(tagset);
    request.setTagging(tagging);
    auto outcome = client.DeleteBucketTagging(request);
    if (outcome.isSuccess()) {
        std::cout << " DeleteBucketTagging success " << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "DeleteBucketTagging fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

11.5.9. 存储空间清单

本文档介绍如何添加、查看、列举和删除存储空间（Bucket）的清单（Inventory）配置。

> **说明** 请确保您拥有调用以上操作的权限。Bucket所有者默认拥有此类权限，若您无此类权限，请先向Bucket所有者申请对应操作的权限。

添加清单配置

说明

- 单个Bucket最多只能有1000条清单配置。
- 配置清单的源Bucket与存放导出的清单文件所在的目标Bucket必须位于同一个Region。

以下代码用于为某个Bucket添加清单（Inventory）配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  InventoryConfiguration inventoryConf;
  /* 指定的清单名称，清单名称在当前Bucket下必须全局唯一 */
  inventoryConf.setId("inventoryId");
  /* 清单配置是否启用的标识, true或false */
  inventoryConf.setIsEnabled(true);
  /* (可选)清单筛选的前缀。指定前缀后，清单将筛选出符合前缀设置的对象 */
  inventoryConf.setFilter(InventoryFilter("objectPrefix"));
  InventoryOSSBucketDestination dest;
  /* 导出清单文件的文件格式 */
  dest.setFormat(InventoryFormat::CSV);
  /* 存储空间所有者授予的账户ID */
  dest.setAccountId("10988548********");
  /* 存储空间所有者授予操作权限的角色名 */
  dest.setRoleArn("acs:ram::10988548********:role/inventory-test");
  /* 存放导出的清单文件的存储空间 */
  dest.setBucket("yourDstBucketName");
  /* 清单文件的存储路径前缀 */
  dest.setPrefix("yourPrefix");
  /* (可选)清单文件的加密方式, 可选SSEOSS或者SSEKMS方式加密 */
  //dest.setEncryption(InventoryEncryption(InventorySSEOSS()));
  //dest.setEncryption(InventoryEncryption(InventorySSEKMS("yourKmskeyId")));
  inventoryConf.setDestination(dest);
  /* 清单文件导出的周期, 可选为Daily或者Weekly */
  inventoryConf.setSchedule(InventoryFrequency::Daily);
  /* 是否在清单中包含Object版本信息, 可选为All或者Current */
  inventoryConf.setIncludedObjectVersions(InventoryIncludedObjectVersions::All);
  /* (可选)设置清单结果中应包含的配置项, 请按需配置 */
  InventoryOptionalFields field {
    InventoryOptionalField::Size, InventoryOptionalField::LastModifiedDate,
    InventoryOptionalField::ETag, InventoryOptionalField::StorageClass,
    InventoryOptionalField::IsMultipartUploaded, InventoryOptionalField::EncryptionStatus
  };
  inventoryConf.setOptionalFields(field);
```

```
    /* 设置清单配置 */
    auto outcome = client.SetBucketInventoryConfiguration(
      SetBucketInventoryConfigurationRequest(BucketName, inventoryConf));
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "Set Bucket Inventory fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
  }
```

有关添加存储空间清单配置的详情，请参见PutBucketInventory。

查看清单配置

以下代码用于查看某个Bucket的清单配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 清单名称*/
  std::string InventoryId = "yourInventoryId";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取清单配置 */
  auto outcome = client.GetBucketInventoryConfiguration(
    GetBucketInventoryConfigurationRequest(BucketName, InventoryId));
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "Get Bucket Inventory fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 打印清单配置信息*/
  const auto& inventoryConf = outcome.result().InventoryConfiguration();
  std::cout << inventoryConf.Id() << std::endl;
  std::cout << inventoryConf.IsEnabled() << std::endl;
  std::cout << inventoryConf.Filter().Prefix() << std::endl;
  std::cout << inventoryConf.Destination().OSSBucketDestination().AccountId() << std::endl;
  std::cout << inventoryConf.Destination().OSSBucketDestination().RoleArn() << std::endl;
  std::cout << inventoryConf.Destination().OSSBucketDestination().Bucket() << std::endl;
  std::cout << inventoryConf.Destination().OSSBucketDestination().Prefix() << std::endl;
  if (inventoryConf.Destination().OSSBucketDestination().Encryption().hasSSEKMS()) {
    std::cout << inventoryConf.Destination().OSSBucketDestination().Encryption().SSEKMS().KeyId() << std::
endl;
  }
  else if (inventoryConf.Destination().OSSBucketDestination().Encryption().hasSSEOSS()) {
    std::cout << "has sse-oss" << std::endl;
  }
  std::cout << static_cast<int>(inventoryConf.Schedule()) << std::endl;
  std::cout << static_cast<int>(inventoryConf.IncludedObjectVersions()) << std::endl;
  for (const auto& fiedl: inventoryConf.OptionalFields()) {
    std::cout << static_cast<int>(fiedl) << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

有关查看清单配置的详情，请参见GetBucketInventory。

批量列举清单配置

> **说明** 单次请求最多可获取100条清单配置项内容。若需获取超过100条清单配置项，则需发送多次请求，并保留相应的Token，作为下一次请求的参数。

以下代码用于批量列举某个Bucket的清单配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置请求参数*/
  std::string nextToken = "";
  bool isTruncated = false;
  do {
    /*列举清单配置，每次请求最多返回100条记录*/
    auto request = ListBucketInventoryConfigurationsRequest(BucketName);
    request.setContinuationToken(nextToken);
    auto outcome = client.ListBucketInventoryConfigurations(request);
    if (!outcome.isSuccess()) {
      /*异常处理*/
      std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
      break;
    }
    for (const auto& inventoryConf : outcome.result().InventoryConfigurationList()) {
      std::cout << inventoryConf.Id() << std::endl;
      std::cout << inventoryConf.IsEnabled() << std::endl;
      std::cout << inventoryConf.Filter().Prefix() << std::endl;
      std::cout << inventoryConf.Destination().OSSBucketDestination().AccountId() << std::endl;
      std::cout << inventoryConf.Destination().OSSBucketDestination().RoleArn() << std::endl;
      std::cout << inventoryConf.Destination().OSSBucketDestination().Bucket() << std::endl;
      std::cout << inventoryConf.Destination().OSSBucketDestination().Prefix() << std::endl;
      if (inventoryConf.Destination().OSSBucketDestination().Encryption().hasSSEKMS()) {
        std::cout << inventoryConf.Destination().OSSBucketDestination().Encryption().SSEKMS().KeyId() << s
td::endl;
      }
      else if (inventoryConf.Destination().OSSBucketDestination().Encryption().hasSSEOSS()) {
        std::cout << "has sse-oss" << std::endl;
      }
      std::cout << static_cast<int>(inventoryConf.Schedule()) << std::endl;
```

```
        std::cout << static_cast<int>(inventoryConf.IncludedObjectVersions()) << std::endl;
        for (const auto& fiedl: inventoryConf.OptionalFields()) {
          std::cout << static_cast<int>(fiedl) << std::endl;
        }
      }
      nextToken = outcome.result().NextContinuationToken();
      isTruncated = outcome.result().IsTruncated();
    } while (isTruncated);
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

有关批量列举清单配置的详情，请参见ListBucketInventory。

删除清单配置

以下代码用于删除某个Bucket的清单配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    /* 清单名称*/
    std::string InventoryId = "yourInventoryId";
    /* 初始化网络等资源 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /* 删除清单配置 */
    auto outcome = client.DeleteBucketInventoryConfiguration(
        DeleteBucketInventoryConfigurationRequest(BucketName, InventoryId));;
    if (!outcome.isSuccess()) {
        /* 异常处理 */
        std::cout << "Delete Bucket Inventory fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

有关删除存储空间清单配置的详情，请参见DeleteBucketInventory。

11.5.10. 授权策略

本文介绍如何设置、获取和删除指定存储空间（Bucket）的授权策略（Policy）。

背景信息

Bucket Policy是基于资源的授权策略。Bucket Policy常见的应用场景如下：

有关Bucket Policy的配置详情及场景案例，请参见通过Bucket Policy授权用户访问指定资源。有关Policy语法，请参见权限策略语法和结构。

设置Bucket Policy

以下代码用于设置Bucket Policy：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置存储空间授权策略，例如该存储空间下，可列举或下载前缀为user1/的对象 */
  std::string policy =
    R"(
    {
      "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
          "Effect" : "Allow",
            "Resource" : ["acs:oss:*:*:*/user1/*"]
      }
      ],
        "Version": "1"
    }
    )";
  SetBucketPolicyRequest request(BucketName);
  request.setPolicy(policy);
  auto outcome = client.SetBucketPolicy(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "Set Bucket Policy fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

有关设置Bucket Policy详情，请参见PutBucketPolicy。

获取Bucket Policy

以下代码用于获取Bucket Policy信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取存储空间授权策略 */
  GetBucketPolicyRequest request(BucketName);
  auto outcome = client.GetBucketPolicy(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "Get Bucket Policy fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  /* 打印配置信息*/
  std::cout << outcome.result().Policy() << std::endl;
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

有关获取Bucket Policy信息详情，请参见GetBucketPolicy。

删除Bucket Policy

以下代码用于删除Bucket Policy：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 删除存储空间授权策略 */
  DeleteBucketPolicyRequest request(BucketName);
  auto outcome = client.DeleteBucketPolicy(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "Delete Bucket Policy fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

有关删除Bucket Policy详情，请参见DeleteBucketPolicy。

11.5.11. 合规保留策略

对象存储OSS允许针对存储空间（Bucket）设置基于时间的合规保留策略，保护周期为1天到70年。本文介绍如何新建、获取、锁定合规保留策略等。

背景信息

OSS支持WORM（Write Once Read Many）特性，允许您以不可删除、不可篡改的方式保存和使用数据。

当基于时间的合规保留策略创建24小时后未提交锁定，则该策略自动失效。当合规保留策略锁定后，您可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，不允许删除Object及合规保留策略，且无法缩短策略保护周期，仅允许延长Object的保留时间。有关合规保留策略的详情，请参见合规保留策略。

新建合规保留策略

以下代码用于新建合规保留策略：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
     /*创建合规保留策略，指定Object保护天数为10天*/
    auto outcome = client.InitiateBucketWorm(InitiateBucketWormRequest(BucketName, 10));
    if (outcome.isSuccess()) {
        std::cout << " InitiateBucketWorm success " << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "InitiateBucketWorm fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

有关新建合规保留策略的详情，请参见InitiateBucketWorm。

取消未锁定的合规保留策略

以下代码用于取消未锁定的合规保留策略：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /*取消未锁定的合规保留策略*/
    auto outcome = client.AbortBucketWorm(AbortBucketWormRequest(BucketName));
    if (outcome.isSuccess()) {
     std::cout << " AbortBucketWorm success " << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "AbortBucketWorm fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

有关取消未锁定的合规保留策略的详情，请参见AbortBucketWorm。

锁定合规保留策略

以下代码用于锁定合规保留策略：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string WormId = "yourWormId";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
     /*锁定合规保留策略*/
    auto outcome = client.CompleteBucketWorm(CompleteBucketWormRequest(BucketName, WormId));
    if (outcome.isSuccess()) {
     std::cout << " CompleteBucketWorm success " << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "CompleteBucketWorm fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

有关锁定合规保留策略的详情，请参见CompleteBucketWorm。

获取合规保留策略

以下代码用于获取合规保留策略：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";

    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
     /*获取合规保留策略*/
    auto outcome = client.GetBucketWorm(GetBucketWormRequest(BucketName));
    if (outcome.isSuccess()) {
        std::cout << " GetBucketWorm success " << std::endl;
        std::cout << " CreationDate:" << outcome.result().CreationDate() <<
        ",State:" << outcome.result().State() <<
        ",WormId:" << outcome.result().WormId() << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "GetBucketWorm fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

有关获取合规保留策略的详情，请参见GetBucketWorm。

延长Object的保留天数

以下代码用于延长已锁定的合规保留策略中Object的保留天数：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string WormId = "yourWormId";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
     /*延长已锁定的合规保留策略中Object的保留天数*/
    auto outcome = client.ExtendBucketWormWorm(ExtendBucketWormRequest(BucketName, WormId, 20));
    if (outcome.isSuccess()) {
     std::cout << " ExtendBucketWormWorm success " << std::endl;
    }
    else {
     /*异常处理*/
     std::cout << "ExtendBucketWormWorm fail" <<
     ",code:" << outcome.error().Code() <<
     ",message:" << outcome.error().Message() <<
     ",requestId:" << outcome.error().RequestId() << std::endl;
     ShutdownSdk();
     return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

有关延长已锁定的合规保留策略中Object的保留天数详情，请参见ExtendBucketWorm。

11.5.12. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

生命周期规则包含如下信息：

- 前缀或标签匹配策略：生命周期规则匹配的Object和碎片。
  - 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。
  - 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，OSS对所有拥有这些标签的对象执行生命周期规则。标签匹配不可以作用于碎片。

    > **说明** 对象标签功能详情请参见对象标签。

  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。

- 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。

- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定非版本状态下的所有Object、以及版本控制状态下的当前版本Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Object成为非当前版本天数：指定一个过期天数N，并指定非当前版本Object过期后执行什么操作。Object会在其成为非当前版本的N天后过期，并执行指定的操作。

> **说明** 您可以将过期Object转换为低频访问类型或归档类型，也可以选择删除过期Object。详情请参见生命周期配置元素。

- 碎片过期策略：设置碎片的过期时间及操作。
  - 过期天数：可指定一个过期天数N，碎片会在其最后修改时间的N天后被删除。
  - 过期日期：指定一个过期日期，最后修改时间在该日期之前的碎片会被全部删除。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  SetBucketLifecycleRequest request(BucketName);
  std::string date("2022-10-12T00:00:00.000Z");
  /* 设置标签。*/
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  /* 指定生命周期规则。*/
  auto rule1 = LifecycleRule();
  rule1.setID("rule1");
  rule1.setPrefix("test1/");
  rule1.setStatus(RuleStatus::Enabled);
  rule1.setExpiration(3);
  rule1.setTags(tagging.Tags());
  /* 指定过期时间。*/
```

```
    auto rule2 = LifecycleRule();
    rule2.setID("rule2");
    rule2.setPrefix("test2/");
    rule2.setStatus(RuleStatus::Disabled);
    rule2.setExpiration(date);
    /* rule3为针对版本控制状态下的Bucket的生命周期规则。*/
    auto rule3 = LifecycleRule();
    rule3.setID("rule3");
    rule3.setPrefix("test3/");
    rule3.setStatus(RuleStatus::Disabled);
    /* 设置Object距其最后修改时间365天之后自动转为归档类型。*/
    auto transition = LifeCycleTransition();
    transition.Expiration().setDays(365);
    transition.setStorageClass(StorageClass::Archive);
    rule3.addTransition(transition);
    /* 设置自动移除过期删除标记。*/
    rule3.setExpiredObjectDeleteMarker(true);
    /* 设置非当前版本的Object距最后修改时间10天之后转为低频访问类型。*/
    auto transition1 = LifeCycleTransition();
    transition1.Expiration().setDays(10);
    transition1.setStorageClass(StorageClass::IA);
    /* 设置非当前版本的object距最后修改时间20天之后转为归档类型。*/
    auto transition2 = LifeCycleTransition();
    transition2.Expiration().setDays(20);
    transition2.setStorageClass(StorageClass::Archive);
    /* 设置Object在其成为非当前版本30天之后删除。*/
    auto expiration  = LifeCycleExpiration(30);
    rule3.setNoncurrentVersionExpiration(expiration);
    LifeCycleTransitionList noncurrentVersionStorageTransitions{transition1, transition2};
    rule3.setNoncurrentVersionTransitionList(noncurrentVersionStorageTransitions);
    /* 设置生命周期规则。*/
    LifecycleRuleList list{rule1, rule2, rule3};
    request.setLifecycleRules(list);
    auto outcome = client.SetBucketLifecycle(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "SetBucketLifecycle fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /* 释放网络等资源。*/
    ShutdownSdk();
    return 0;
}
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
#include <alibabacloud/oss/OssClient.h>
#include "../src/utils/Utils.h"
```

```
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 查看生命周期规则。*/
  auto outcome = client.GetBucketLifecycle(BucketName);
  if (outcome.isSuccess()) {
    std::cout << "GetBucketLifecycle success," << std::endl;
    for (auto const rule : outcome.result().LifecycleRules()) {
      std::cout << "rule:" << rule.ID() << "," << rule.Prefix() << "," << rule.Status() << ","
      "hasExpiration:" << rule.hasExpiration() << "," <<
      "hasTransitionList:" << rule.hasTransitionList() << "," << std::endl;
      auto taglist = rule.Tags();
      for (const auto& tag : taglist)
      {
        std::cout <<"GetBucketLifecycle tag success, Key:"
        << tag.Key() << "; Value:" << tag.Value() << std::endl;
      }
      /* 查看是否自动删除过期删除标记。*/
      if (rule.ExpiredObjectDeleteMarker()) {
        std::cout << "rule expired delete marker: " << rule.ExpiredObjectDeleteMarker() << std::endl;
      }
      /* 查看非当前版本object存储类型转换规则。*/
      if (rule.hasNoncurrentVersionTransitionList()) {
        for (auto const lifeCycleTransition : rule.NoncurrentVersionTransitionList()) {
          std::cout << "rule noncurrent versions trans days:" << lifeCycleTransition.Expiration() <<
          " trans storage class: " << ToStorageClassName(lifeCycleTransition.StorageClass()) << std::endl;
        }
      }
      /* 查看非当前版本object过期规则。*/
      if (rule.hasNoncurrentVersionExpiration()) {
        std::cout << "rule noncurrent versions expiration days:" << rule.NoncurrentVersionExpiration().Days(
) << std::endl;
      }
    }
  }
  else {
    /* 异常处理。*/
    std::cout << "GetBucketLifecycle fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
```

```
  ShutdownSdk();
  return 0;
}
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
   /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 清空生命周期规则。*/
  DeleteBucketLifecycleRequest request(BucketName);
  auto outcome = client.DeleteBucketLifecycle(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "DeleteBucketLifecycle fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

11.5.13. 请求者付费模式

阿里云对象存储OSS的请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量费用和请求费用，而Bucket拥有者仅支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

请求方式说明

- 不允许匿名访问

  如果您的Bucket启用了请求者付费模式，则不允许匿名访问该Bucket。请求方必须提供身份验证信息，以便OSS能够识别请求方，从而对请求方而非Bucket拥有者收取请求所产生的费用。

  当请求者是通过扮演阿里云RAM角色来请求数据时，该角色所属的账户将为此请求付费。

- 申请方需携带x-oss-request-payer信息

如果您的Bucket启用了请求者付费模式，请求方必须在PUT、POST、GET和HEAD等请求的Header信息中包含x-oss-request-payer:requester，以表明请求方知道请求和数据下载将产生费用。否则，请求方无法通过验证。

数据拥有者访问该Bucket时，可以不携带x-oss-request-payer请求头。数据拥有者作为请求者访问该Bucket时，请求产生的费用由数据拥有者（也是请求者）来支付。

有关请求者付费模式的详情，请参见开发指南的请求者付费模式。

设置请求者付费模式

以下代码用于设置请求者付费模式：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置请求者付费模式 */
  SetBucketRequestPaymentRequest request(BucketName);
  request.setRequestPayer(RequestPayer::Requester);
  auto outcome = client.SetBucketRequestPayment(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketRequestPayment fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取请求者付费模式配置

以下代码用于获取请求者付费模式配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取请求者付费模式 */
  GetBucketRequestPaymentRequest request(BucketName);
  auto outcome = client.GetBucketRequestPayment(request);
  if (outcome.isSuccess())
  {
    std::cout << "GetBucketRequestPayment success Payer:" << outcome.result().Payer() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetBucketPayment fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

第三方付费访问Object

第三方操作Object时需在http header中携带x-oss-request-payer:requester参数，否则会报错。

以下代码以PutObject、GetObject及DeleteObject为例，用于指定第三方付费访问Object。

其他用于指定第三方付费的Object读写操作接口设置方法类似。

以下代码用于设置第三方付费访问Object：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string PayerAccessKeyId = "yourPayerAccessKeyId";
  std::string PayerAccessKeySecret = "yourPayerAccessKeySecret";
  std::string PayerEndpoint = "yourPayerEndpoint";
  std::string BucketName = "yourBucketName";
  std::string PayerUID = "PayerUID";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  OssClient payerClient(PayerEndpoint, PayerAccessKeyId, PayerAccessKeySecret, conf);
  /* 创建bucket */
  CreateBucketOutcome outCome = client.CreateBucket(CreateBucketRequest(BucketName));
  /* 设置请求者付费模式 */
  SetBucketRequestPaymentRequest request(BucketName);
  request.setRequestPayer(RequestPayer::Requester);
  auto payoutcome = client.SetBucketRequestPayment(request);
  /* 上传文件，并设置请求者付费 */
  *content << "test cpp sdk";
  PutObjectRequest putrequest(BucketName, ObjectName, content);
  putrequest.setRequestPayer(RequestPayer::Requester);
  auto putoutcome = payerClient.PutObject(putrequest);
  /* 获取文件到本地内存，并设置请求者付费 */
  GetObjectRequest getrequest(BucketName, ObjectName);
  getrequest.setRequestPayer(RequestPayer::Requester);
  auto getoutcome = payerClient.GetObject(getrequest);
  /* 删除文件，并设置请求者付费 */
  DeleteObjectRequest delrequest(BucketName, ObjectName);
  delrequest.setRequestPayer(RequestPayer::Requester);
  auto deloutcome = payerClient.DeleteObject(delrequest);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.5.14. 防盗链

本文主要介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链

以下代码用于设置防盗链：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret,  conf) ;
   /* 设置防盗链 */
   SetBucketRefererRequest request(BucketName);
   request.addReferer("http://www.referersample.com");
   request.addReferer("https://www.referersample.com");
   request.addReferer("https://www.?.referersample.com");
   request.addReferer("https://www.*.cn");
   request.setAllowEmptyReferer(false);
   auto outcome = client.SetBucketReferer(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "SetBucketReferer fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 获取防盗链信息 */
   GetBucketRefererRequest request(BucketName);
   auto outcome = client.GetBucketReferer(request);
   if (outcome.isSuccess()) {
      std::cout << " GetBucketReferer success, AllowEmptyReferer: " << outcome.result().AllowEmptyReferer()
<<
      " ,Referer size: " << outcome.result().RefererList().size() << std::endl;
   }
   else {
      /* 异常处理 */
      std::cout << "GetBucketReferer fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

清空防盗链

以下代码用于清空防盗链：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 清空防盗链，防盗链不能直接清空，需要新建一个允许空referer的规则来覆盖之前的规则 */
   SetBucketRefererRequest request(BucketName);
   request.setAllowEmptyReferer(true);
   auto outcome = client.SetBucketReferer(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "CleanBucketReferer fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.5.15. 访问日志

您可以开启存储空间的访问日志记录功能。开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

更多关于访问日志的介绍，请参见开发指南中的设置访问日志记录。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string TargetBucketName = "yourTargetBucketName";
  std::string TargetPrefix  ="yourTargetPrefix";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret ,  conf);
  /* 开启访问日志记录 */
  SetBucketLoggingRequest request(BucketName, TargetBucketName, TargetPrefix);
  auto outcome = client.SetBucketLogging(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketLogging fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 查看访问日志设置*/
  GetBucketLoggingRequest request(BucketName);
  auto outcome = client.GetBucketLogging(request);
  if (outcome.isSuccess()) {
    std::cout <<" GetBucketLogging success, TargetBucket: " << outcome.result().TargetBucket() <<
    ",TargetPrefix: " << outcome.result().TargetPrefix() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetBucketLogging fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 关闭访问日志记录 */
   DeleteBucketLoggingRequest request(BucketName);
   auto outcome = client.DeleteBucketLogging(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "DeleteBucketLogging fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.5.16. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret,  conf);
  /* 设置静态网站托管 */
  SetBucketWebsiteRequest request(BucketName);
  request.setIndexDocument("index.html");
  request.setErrorDocument("error.html");
  auto outcome = client.SetBucketWebsite(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketWebsite fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取静态网站托管配置 */
  GetBucketWebsiteRequest request(BucketName);
  auto outcome = client.GetBucketWebsite(request);
  if (outcome.isSuccess()) {
    std::cout << "GetBucketWebsite success,IndexDocument: " << outcome.result().IndexDocument() <<
    " ,ErrorDocument: " << outcome.result().ErrorDocument() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetBucketWebsite fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret,  conf);
   /* 删除静态网站托管配置 */
   DeleteBucketWebsiteRequest request(BucketName);
   auto outcome = client.DeleteBucketWebsite(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "DeleteBucketWebsite fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.5.17. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  SetBucketCorsRequest request(BucketName);
  /* 设置跨域资源共享规则 */
  auto rule1 = CORSRule();
  /* 指定允许跨域请求的来源 */
  rule1.addAllowedOrigin("http://www.a.com");
  /* 指定允许跨域请求方法（GET/PUT/POST/DELETE/HEAD）*/
  rule1.addAllowedMethod("POST");
  /* 是否允许预取指令（OPTIONS）中Access-Control-Request-Headers头中指定的Header */
  rule1.addAllowedHeader("*");
  /* 指定允许用户从应用程序中访问的响应头 */
  rule1.addExposeHeader("x-oss-test");
  /* 最多允许10条规则 */
  request.addCORSRule(rule1);
  auto rule2 = CORSRule();
  rule2.addAllowedOrigin("http://www.b.com");
  rule2.addAllowedMethod("GET");
  rule2.addExposeHeader("x-oss-test2");
  rule2.setMaxAgeSeconds(100);
  request.addCORSRule(rule2);
  auto outcome = client.SetBucketCors(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketCors fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, c onf);
   /* 获取跨域资源共享规则 */
   auto outcome = client.GetBucketCors(BucketName);
   if (outcome.isSuccess()) {
      std::cout << "GetBucketCors success" << std::endl;
      for (auto const rule : outcome.result().CORSRules()) {
         std::cout << "Get Bucket Cors List:" << std::endl;
         for (auto const origin : rule.AllowedOrigins()) {
            std::cout << "Allowed origin:" << origin << std::endl;
         }
      }
   }
   else {
      /* 异常处理 */
      std::cout << "GetBucketCors fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret,  conf);
   /* 删除跨域资源共享规则 */
   auto outcome = client.DeleteBucketCors(BucketName);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "DeleteBucketCors fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.6. 上传文件

11.6.1. 概述

在OSS中，操作的基本数据单元是文件（Object）。OSS C++ SDK提供了丰富的文件上传方式：

- 简单上传：包括从内存上传和从本地上传。最大不能超过5GB。
- 追加上传：最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

11.6.2. 简单上传

本文主要介绍如何从内存中或从本地磁盘上传文件（Object）。

从内存中上传文件

以下代码用于从内存中上传内容到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称 */
  std::string ObjectName = "exampledir/exampleobject.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "Thank you for using Alibaba Cloud Object Storage Service!";
  PutObjectRequest request(BucketName, ObjectName, content);
  /*（可选）请参见如下示例设置存储类型及访问权限ACL* /
  //request.MetaData().addHeader("x-oss-object-acl", "private");
  //request.MetaData().addHeader("x-oss-storage-class", "Standard");
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

上传本地文件

以下代码用于上传本地*D:\\localpath*路径下的examplefile.txt到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
#include <alibabacloud/oss/OssClient.h>
#include <fstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写文件完整路径，例如exampledir/exampleobject.txt。文件完整路径中不能包含Bucket名称 */
  std::string ObjectName = "exampledir/exampleobject.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 填写本地文件完整路径，例如D:\\localpath\\examplefile.txt，其中localpath为本地文件examplefile.txt所在本地路径 */
  std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>("D:\\localpath\\examplefile.txt", std::ios::in | std::ios::binary);
  PutObjectRequest request(BucketName, ObjectName, content);
  /*（可选）请参见如下示例设置存储类型及访问权限ACL */
  //request.MetaData().addHeader("x-oss-object-acl", "private");
  //request.MetaData().addHeader("x-oss-storage-class", "Standard");
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
    return 0;
}
```

11.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出

ObjectNotAppendable异常。

- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

示例代码

以下代码用于追加上传文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  /* 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  /* Endpoint以杭州为例，其它Region请按实际情况填写。
  std::string Endpoint = "yourEndpoint";
  /* 设置Bucket名称。*/
  std::string BucketName = "yourBucketName";
  /* 设置Object名称。即不包含Bucket名称在内的Object的完整路径，例如example/folder/abc.jpg。*/
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  /* 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度。*/
  std::shared_ptr<std::iostream> content1 = std::make_shared<std::stringstream>();
  *content1 <<"Thank you for using Aliyun Object Storage Service!";
  AppendObjectRequest request(BucketName, ObjectName, content1, meta);
  request.setPosition(0L);
  /* 第一次追加文件。*/
  auto result = client.AppendObject(request);
  if (!result.isSuccess()) {
    /* 异常处理。*/
    std::cout << "AppendObject fail" <<
    ",code:" << result.error().Code() <<
    ",message:" << result.error().Message() <<
    ",requestId:" << result.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  std::shared_ptr<std::iostream> content2 = std::make_shared<std::stringstream>();
  *content2 <<"Thank you for using Aliyun Object Storage Service!";
  auto position = result.result().Length();
  AppendObjectRequest appendObjectRequest(BucketName, ObjectName, content2);
  appendObjectRequest.setPosition(position);
  /* 第二次追加文件。*/
  auto outcome = client.AppendObject(appendObjectRequest);
  if (!outcome.isSuccess()) {
    /* 异常处理。*/
```

```
        std::cout << "AppendObject fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        ShutdownSdk();
        return -1;
    }
    /* 释放网络等资源。*/
    ShutdownSdk();
    return 0;
}
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

11.6.4. 断点续传上传

通过断点续传上传的方式将文件上传到OSS前，您可以指定断点记录点。上传过程中，如果出现网络异常或程序崩溃导致文件上传失败时，将从断点记录处继续上传未上传完成的部分。

背景信息

使用断点续传上传时，文件上传的进度信息会记录在CheckpointDir文件中，如果上传过程中某一分片上传失败，再次上传时会从CheckpointDir文件中记录的点继续上传，从而达到断点续传的效果。上传完成后，CheckpointDir文件会被删除。

您可以通过client.ResumableUploadObject方法实现断点续传上传。此方法中的UploadObjectRequest请求包含以下参数：

参数	描述	是否必须	默认值	设置方法
bucket	存储空间名称。	是	无	通过构造方法设置
key	上传到OSS的文件名称。	是	无	
filePath	上传的本地文件名称。	否	无	
partSize	上传的分片大小，取值范围为100 KB~5 GB。	否	8 MB	通过setPartSize设置
threadNum	分片上传的并发数。	否	3	通过构造函数或者setThreadNum设置
checkpointDir	记录本地分片上传结果的文件。上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。	否	与DownloadFile同目录	通过构造函数或者setCheckpointDir设置

断点续传上传详情请参见分片上传和断点续传。完整代码请参见GitHub。

代码实现

以下代码用于断点续传上传。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName";
   std::string UploadFilePath = "yourUploadfilePath";
   std::string CheckpointFilePath = "yourCheckpointFilepath"
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 断点续传上传 */
   UploadObjectRequest request(BucketName, ObjectName, UploadFilePath, CheckpointFilePath);
   auto outcome = client.ResumableUploadObject(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "ResumableUploadObject fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用InitiateMultipartUpload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

   调用UploadPart方法上传分片数据。

> **说明**
>
> - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
> - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
> - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用CompleteMultipartUpload方法将所有分片合并成完整的文件。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
#include <alibabacloud/oss/OssClient.h>
int64_t getFileSize(const std::string& file)
{
  std::fstream f(file, std::ios::in | std::ios::binary);
  f.seekg(0, f.end);
  int64_t size = f.tellg();
  f.close();
  return size;
}
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName);
  /*（可选）请参见如下示例设置存储类型 */
  //initUploadRequest.MetaData().addHeader("x-oss-storage-class", "Standard");
  /* 初始化分片上传事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  std::string fileToUpload = "yourLocalFilename";
  int64_t partSize = 100 * 1024;
  PartList partETagList;
  auto fileSize = getFileSize(fileToUpload);
  int partCount = static_cast<int>(fileSize / partSize);
  /* 计算分片个数 */
  if (fileSize % partSize != 0) {
    partCount++;
  }
  /* 对每一个分片进行上传 */
```

```
  /* 对每一个分片进行上传 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>(fileToUpload, std::ios::in|std::
ios::binary);
    content->seekg(skipBytes, std::ios::beg);
    UploadPartRequest uploadPartRequest(BucketName, ObjectName, content);
    uploadPartRequest.setContentLength(size);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setPartNumber(i);
    auto uploadPartOutcome = client.UploadPart(uploadPartRequest);
    if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
    }
    else {
      std::cout << "uploadPart fail" <<
      ",code:" << uploadPartOutcome.error().Code() <<
      ",message:" << uploadPartOutcome.error().Message() <<
      ",requestId:" << uploadPartOutcome.error().RequestId() << std::endl;
    }
  }
  /* 完成分片上传 */
  CompleteMultipartUploadRequest request(BucketName, ObjectName);
  request.setUploadId(uploadId);
  request.setPartList(partETagList);
  /*（可选）请参见如下示例设置读写权限ACL */
  //request.setAcl(CannedAccessControlList::Private);
  auto outcome = client.CompleteMultipartUpload(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CompleteMultipartUpload fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取已经上传的分片过程中调用CompleteMultipartUpload接口时，需提供每个分片的ETag值。您可以通过以下两种方式获取ETag值：

- 上传每个分片时，返回结果中会包含这个分片的ETag值，供您直接保存使用。上述示例采用的是此种方式。
- 通过调用ListParts方法获取已经上传的分片的ETag值。

列举已上传的分片

调用ListParts方法列举出指定UploadID下所有已上传成功的分片。

- 列举所有已上传的分片

> **说明** 默认情况下，ListParts只能一次列举1000个分片。当分片数量大于1000时，请分页列举所有已上传分片。

以下代码用于列举所有已上传的分片：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 列举已上传分片，默认列举1000个分片 */
  ListPartsRequest listuploadrequest(BucketName, ObjectName);
  listuploadrequest.setUploadId(uploadId);
  do {
    auto listuploadresult = client.ListParts(listuploadrequest);
    if (!listUploadResult.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListParts fail" <<
      ",code:" << listuploadresult.error().Code() <<
      ",message:" << listuploadresult.error().Message() <<
      ",requestId:" << listuploadresult.error().RequestId() << std::endl;
      break;
    }
    else {
      for (const auto& part : listuploadresult.result().PartList()) {
        std::cout << "part"<<
        ",name:" << part.PartNumber() <<
        ",size:" << part.Size() <<
        ",etag:" << part.ETag() <<
        ",lastmodify time:" << part.LastModified() << std::endl;
      }
    }
    listuploadrequest.setPartNumberMarker(listuploadresult.result().NextPartNumberMarker());
  } while (listuploadresult.result().IsTruncated());
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 分页列举所有已上传的分片

以下代码用于指定每页分片的数量，并分页列举所有分片：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 分页列举全部上传的分片 */
  /* 设置每页列举最大上传分片数目 */
  ListPartsRequest listuploadrequest(BucketName, ObjectName);
  listuploadrequest.setMaxParts(50);
  listuploadrequest.setUploadId(uploadId);
  do {
    listuploadresult = client.ListParts(listuploadrequest);
    if (!listUploadResult.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListParts fail" <<
      ",code:" << listuploadresult.error().Code() <<
      ",message:" << listuploadresult.error().Message() <<
      ",requestId:" << listuploadresult.error().RequestId() << std::endl;
      break;
    }
    else {
      for (const auto& part : listuploadresult.result().PartList()) {
        std::cout << "part"<<
        ",name:" << part.PartNumber() <<
        ",size:" << part.Size() <<
        ",etag:" << part.ETag() <<
        ",lastmodify time:" << part.LastModified() << std::endl;
      }
    }
    listuploadrequest.setPartNumberMarker(listuploadresult.result().NextPartNumberMarker());
  } while (listuploadresult.result().IsTruncated());
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

列举分片上传事件

调用ListMultipartUploads方法列举出所有执行中的分片上传事件，即已初始化但尚未完成或已取消的分片上传事件。

- 列举全部分片上传事件

> **说明** 默认情况下，ListMultipartUploads只能一次列举1000个分片上传事件。当分片数量大于1000时，请分页列举所有上传事件。

以下代码用于列举所有已上传事件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 列举已上传事件，默认列举1000个分片 */
  ListMultipartUploadsRequest listmultiuploadrequest(BucketName);
  do {
    auto listresult = client.ListMultipartUploads(listmultiuploadrequest);
    if (!listresult.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListMultipartUploads fail" <<
      ",code:" << listresult.error().Code() <<
      ",message:" << listresult.error().Message() <<
      ",requestId:" << listresult.error().RequestId() << std::endl;
      break;
    }
    else {
      for (const auto& part : listresult.result().MultipartUploadList()) {
        std::cout << "part"<<
        ",name:" << part.Key <<
        ",uploadid:" << part.UploadId <<
        ",initiated time:" << part.Initiated << std::endl;
      }
    }
    listmultiuploadrequest.setKeyMarker(listresult.result().NextKeyMarker());
    listmultiuploadrequest.setUploadIdMarker(listresult.result().NextUploadIdMarker());
  } while (listresult.result().IsTruncated());
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 分页列举所有上传事件

  以下代码用于分页列举所有上传事件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 分页列举全部上传事件 */
  /* 设置每页列举最大上传事件数目 */
  ListMultipartUploadsRequest  listmultiuploadrequest(BucketName);
  listmultiuploadrequest.setMaxUploads(50);
  do {
    listresult = client.ListMultipartUploads(listmultiuploadrequest);
    if (!listresult.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListMultipartUploads fail" <<
      ",code:" << listresult.error().Code() <<
      ",message:" << listresult.error().Message() <<
      ",requestId:" << listresult.error().RequestId() << std::endl;
      break;
    }
    else {
      for (const auto& part : listresult.result().MultipartUploadList()) {
        std::cout << "part"<<
        ",name:" << part.Key <<
        ",uploadid:" << part.UploadId <<
        ",initiated time:" << part.Initiated << std::endl;
      }
    }
    listmultiuploadrequest.setKeyMarker(listresult.result().NextKeyMarker());
    listmultiuploadrequest.setUploadIdMarker(listresult.result().NextUploadIdMarker());
  } while (listresult.result().IsTruncated());
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

取消分片上传事件

您可以调用client.AbortMultipartUpload方法来取消分片上传事件。当一个分片上传事件被取消后，无法再使用同一个uploadId执行任何操作，已经上传的分片数据会被删除。

以下代码用于取消分片上传事件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName);
  /* 初始化分片上传事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  /* 取消分片上传 */
  AbortMultipartUploadRequest  abortUploadRequest(BucketName, ObjectName, uploadId);
  auto abortUploadIdResult = client.AbortMultipartUpload(abortUploadRequest);
  if (!abortUploadIdResult.isSuccess()) {
    /* 异常处理 */
    std::cout << "AbortMultipartUpload fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.6.6. 中断上传

您可以在上传过程中的任意时间节点中断上传。

以下代码用于中断上传：

```
#include <alibabacloud/oss/OssClient.h>
#include <fstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName ";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   std::vector<PutObjectOutcomeCallable> Callables;
   for (int i=0; i < 4; i++)  {
      std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>("yourLocalFilename", std::ios
::in|std::ios::binary);
      PutObjectRequest request(BucketName, ObjectName, content);
      auto outcomeCallable = client.PutObjectCallable(request);
      Callables.emplace_back(std::move(outcomeCallable));
   }
   std::this_thread::sleep_for(std::chrono::milliseconds(1000));
   /* 中断上传 */
   client.DisableRequest();
   for (size_t i = 0; i < Callables.size(); i++) {
      auto outcome = Callables[i].get();
      if (outcome.error().Code() == "ClientError:100002" ||
      outcome.error().Code() == "ClientError:200042") {
         std::cout << "disable putobject success" << std::endl;
      }
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

说明 中断上传与取消分片上传区别如下：

- 中断上传是指开发者调用上传接口向远程 OSS 上传文件时，停掉了该调用任务，即取消了本次上传任务。取消分片上传是指通知 OSS 不再接受该 UploadID 的分片上传任务。
- 若进行分片上传任务，中断上传将导致本次分片上传任务被取消，您仍然可以调用该 UploadID 重新上传。若取消分片上传后，则该 UploadID 将不可用。
- 中断上传对所有接口均有效。取消分片上传只针对分片上传有效。

11.6.7. 进度条

进度条用于指示上传或下载的进度。

下面的代码以PutObject方法为例，介绍如何使用进度条。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
void ProgressCallback(size_t increment, int64_t transfered, int64_t total, void* userData)
{
  std::cout << "ProgressCallback[" << userData << "] => " <<
  increment <<" ," << transfered << "," << total << std::endl;
}
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>("yourLocalFilename", std::ios::i
n|std::ios::binary);
  PutObjectRequest request(BucketName, ObjectName, content);
  TransferProgress progressCallback = { ProgressCallback , nullptr };
  request.setTransferProgress(progressCallback);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.6.8. 上传回调

本文介绍如何使用上传回调。

以下代码用于上传回调（Callback）：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName";
   /* 您的回调服务器地址，如http://oss-demo.aliyunc s.com:23450 */
   std::string ServerName = "yourServerName";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
   *content << "Thank you for using Aliyun Object Storage Service!";
   /* 设置上传回调参数 */
   std::string callbackBody = "bucket=${bucket}&object=${object}&etag=${etag}&size=${size}&mimeType=${
mimeType}&my_var1=${x:var1}";
   ObjectCallbackBuilder builder(ServerName, callbackBody, "", ObjectCallbackBuilder::Type::URL);
   std::string value = builder.build();
   ObjectCallbackVariableBuilder varBuilder;
   varBuilder.addCallbackVariable("x:var1", "value1");
   std::string varValue = varBuilder.build();
   PutObjectRequest request(BucketName, ObjectName, content);
   request.MetaData().addHeader("x-oss-callback", value);
   request.MetaData().addHeader("x-oss-callback-var", varValue);
   auto outcome = client.PutObject(request);
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

上传回调详情请参见开发指南中的上传回调，调试方法和错误排除请参见上传回调错误及排除。有关上传回调的实现原理，请参见 Callback API 文档。

11.7. 下载文件

11.7.1. 概述

OSS C++ SDK 提供了丰富的文件下载方式：

- 下载到本地文件
- 下载到本地内存
- 范围下载
- 断点续传下载

下载过程中，您还可以通过进度条功能查看下载进度。

11.7.2. 下载到本地文件

本文介绍如何将OSS文件下载到本地文件。

以下代码用于将指定的OSS文件下载到本地文件。

```
#include <alibabacloud/oss/OssClient.h>
#include <memory>
#include <fstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*设置下载文件的文件名*/
  std::string FileNametoSave = "yourFileName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取文件到本地文件*/
  GetObjectRequest request(BucketName, ObjectName);
  request.setResponseStreamFactory([=]() {return std::make_shared<std::fstream>(FileNametoSave, std::io
s_base::out | std::ios_base::in | std::ios_base::trunc| std::ios_base::binary); });
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
    std::cout << "GetObjectToFile success" << outcome.result().Metadata().ContentLength() << std::endl;
  }
  else {
    /*异常处理*/
    std::cout << "GetObjectToFile fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

11.7.3. 下载到本地内存

本文介绍如何将OSS文件下载到本地内存。

以下代码用于把指定的OSS文件下载到本地内存：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息。*/
  /*阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  /*填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com。*/
  std::string Endpoint = "yourEndpoint";
  /*填写Bucket名称。*/
  std::string BucketName = "yourBucketName";
  /*填写不包含Bucket名称在内的Object的完整路径，例如desrfolder/exampleobject.txt。*/
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取文件到本地内存。*/
  GetObjectRequest request(BucketName, ObjectName);
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
   std::cout << "getObjectToBuffer" << " success, Content-Length:" << outcome.result().Metadata().ContentLength() << std::endl;
    /*通过read接口读取数据。*/
    auto& stream = outcome.result().Content();
    char buffer[256];
    while (stream->good()) {
      stream->read(buffer, 256);
      auto count = stream->gcount();
      /*根据实际情况处理数据。*/
  }
  }
  else {
    /*异常处理。*/
    std::cout << "getObjectToBuffer fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

11.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

指定正常的下载范围

以下代码用于指定正常的下载范围来下载文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObectName ";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf );
   /* 获取文件 */
   GetObjectRequest request(BucketName, ObjectName);
   /* 设置下载范围 */
   request.setRange(0, 1);
   auto outcome = client.GetObject(request);
   if (!outcome.isSuccess ()) {
      /* 异常处理 */
      std::cout << "getObject fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
       return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

指定异常的下载范围

假设现有大小为1000 Bytes的Object，则指定的正常下载范围应为0~999。如果指定范围不在有效区间，会导致Range不生效，响应返回值为200，并传送整个Object的内容。请求不合法的示例及返回说明如下：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。
- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回整个文件的内容，且HTTP Code为200。

兼容行为范围下载

在请求中增加请求头x-oss-range-behavior:standard，则改变指定范围不在有效区间时OSS的下载行为。假设现有大小为1000 Bytes的Object：

- 若指定了Range: bytes=500~2000，此时范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206。

- 若指定了Range: bytes=1000~2000，此时范围首端取值不在有效区间，返回HTTP Code为416，错误码为InvalidRange。

以下代码用于兼容行为范围下载：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
    /* 指定大小为1000 Bytes的文件 */
   std::string ObjectName = "yourObectName ";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf );
   /* 获取文件 */
   GetObjectRequest request(BucketName, ObjectName);
   /* 设置下载范围 */
   /* 指定兼容行为 */
   /* 范围末端取值不在有效区间，返回500~999字节范围内容，且HTTP Code为206 */
   request.setRange(500, 2000, true);
   auto outcome = client.GetObject(request);
   /* 范围首端取值不在有效区间，则会抛出异常。返回HTTP Code为416，错误码为InvalidRange */
   request.setRange(1000, 2000, true);
   outcome = client.GetObject(request);
   if (!outcome.isSuccess ()) {
      /* 异常处理 */
      std::cout << "getObject fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
       return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

11.7.5. 断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此 OSS 提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

您可以通过 OssClient .ResumableDownloadObject 方法实现断点续传下载。此方法的 DownloadObjectRequest 请求包含以下参数：

参数	描述	是否必需	默认值	如何设置
bucket	存储空间名称。	是	无	通过构造方法设置。
key	OSS文件名称。	是	无	通过构造方法设置。
filePath	本地文件。OSS文件将下载到该文件。	否	OSS文件名称	通过构造方法设置。
partSize	分片大小，取值范围为100KB~5GB。	否	8MB	通过setPartSize设置。
threadNum	分片下载的并发数。	否	3	通过构造方法或者setThreadNum 设置。
checkpointDir	记录本地分片下载结果的文件。开启断点续传功能时需要设置此参数。下载过程中的进度信息会保存在该文件中，如果某一分片下载失败，再次下载时会根据文件中记录的点继续下载。下载完成后，该文件会被删除。	否	与DownloadFile同目录。	通过构造方法或者setCheckpointDir设置。

以下代码用于断点续传下载：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string DownloadFilePath = "yourDownloadFilePath";
  std::string CheckpointFilePath = "yourCheckpointFilepath";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 断点续传下载 */
  DownloadObjectRequest request(BucketName, ObjectName, DownloadFilePath, CheckpointFilePath);
  auto outcome = client.ResumableDownloadObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "ResumableDownloadObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.7.6. 进度条

进度条用于指示上传或下载的进度。

下载进度条的完整代码请参见Git Hub。

以下代码展示如何使用进度条：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
void ProgressCallback(size_t increment, int64_t transfered, int64_t total, void* userData)
{
  std::cout << "ProgressCallback[" << userData << "] => " <<
          increment <<" ," << transfered << "," << total << std::endl;
}
int main(void )
{
   /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName" ;
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeyS ecre t, conf);
  /* 获取文件 */
  GetObjectRequest request(BucketName, ObjectName);
  TransferProgress progressCallback = { ProgressCallback , nullptr };
  request.setTransferProgress(progressCallback);
  auto outcome = client.GetObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "getObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk ();
     return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.8. 管理文件

11.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 拷贝文件
- 删除文件
- 判断文件是否存在
- 解冻归档文件

- 管理软链接

11.8.2. 判断文件是否存在

本文主要介绍判断文件是否存在。

以下代码用于判断examplebucket中的exampleobject.txt文件是否存在。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 判断文件是否存在 */
  auto outcome = client.DoesObjectExist(BucketName, ObjectName);
  if (outcome) {
   std::cout << "The Object exists!" << std::endl;
  }
  else {
   std::cout << "The Object does not exist!" << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.8.3. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	CannedAccessControlList::Default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	CannedAccessControlList::Private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	CannedAccessControlList::PublicRead

访问权限	描述	访问权限值
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	CannedAccessControlList::PublicReadWrite

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

设置文件访问权限

以下代码用于设置指定文件的访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置文件访问权限 */
  SetObjectAclRequest request(BucketName, ObjectName);
  request.setAcl(CannedAccessControlList::Private);
  auto outcome = client.SetObjectAcl(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetObjectAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取文件访问权限

以下代码用于获取指定文件的访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取文件访问权限 */
  GetObjectAclRequest request(BucketName, ObjectName);
  auto outcome = client.GetObjectAcl(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "GetObjectAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  else {
    std::cout << " GetObjectAcl success, Acl:" << outcome.result().Acl() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.8.4. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息。

> **说明** 有关文件元信息的更多详情，请参见开发指南中的文件元信息。

设置文件元信息

以下代码用于设置文件元信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret , conf);
  /* 设置HTTP header */
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  meta.setCacheControl("max-ag e=3");
  /* 设置自定义文件元信息 */
  meta.UserMetaData()["meta"] = "meta-value";
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "Thank you for using Aliyun Object Storage Service!";
  client.PutObject(BucketName, ObjectName, content, meta);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取文件元信息

您可以通过以下两种方法获取文件元信息：

方法	描述	优势
GetObjectMeta	获取文件的ETag、Size（文件大小）、LastModified（最后修改时间）。	更轻量、更快
HeadObject	获取文件的全部元信息。	无

以下代码用于获取文件元信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 获取文件的部分元信息 */
   auto outcome = client.GetObjectMeta(BucketName, ObjectName);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "GetObjectMeta fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   else {
      auto metadata = outcome.result();
      std::cout << " get metadata success, ETag:" << metadata.ETag() << "; LastModified:"
      << metadata.LastModified() << "; Size:" << metadata.ContentLength() << std::endl;
   }
   /* 获取文件的全部元信息 */
   outcome = client.HeadObject(BucketName, ObjectName);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "HeadObject fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   else {
      auto headMeta = outcome.result();
      std::cout <<"headMeta success, ContentType:"
      << headMeta.ContentType() << "; ContentLength:" << headMeta.ContentLength()
      << "; CacheControl:" << headMeta.CacheControl() << std::endl;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

说明 HTTP header详情请参见RFC2616。

11.8.5. 转换文件存储类型

OSS提供标准、低频访问、归档和冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。本文主要介绍如何转换文件（Object）的存储类型。

有关存储类型的更多信息，请参见开发指南的存储类型介绍及存储类型转换。

以下提供了详细的示例代码用于Object存储类型的相互转换。

- 以下代码用于将Object的存储类型从标准或低频访问转换为归档类型：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置修改后的文件存储类型，例如将修改后的文件存储类型设置为归档 */
  ObjectMetaData objectMeta;
  objectMeta.addHeader("x-oss-storage-class", "Archive");
  CopyObjectRequest request(SourceBucketName, SourceBucketName,objectMeta);
  request.setCopySource(SourceBucketName, SourceObjectName);
  /* 修改为上述指定的文件存储类型 */
  auto outcome = client.CopyObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CopyObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 以下代码用于将Object的存储类型从归档转换为低频访问类型：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
```

```
/* 初始化OSS账号信息 */
std::string AccessKeyId = "yourAccessKeyId";
std::string AccessKeySecret = "yourAccessKeySecret";
std::string Endpoint = "yourEndpoint";
std::string SourceBucketName = "yourSourceBucketName";
std::string SourceObjectName = "yourSourceObjectName";
/* 初始化网络等资源 */
InitializeSdk();
ClientConfiguration conf;
OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
/* 检查目标文件是否为归档类型。如果是，则需要先解冻才能更改存储类型 */
auto restoreOutcome = client.RestoreObject(SourceBucketName, SourceObjectName);
if (!restoreOutcome.isSuccess()) {
  /* 异常处理 */
  std::cout << "RestoreObject fail" <<
  ",code:" << restoreOutcome.error().Code() <<
  ",message:" << restoreOutcome.error().Message() <<
  ",requestId:" << restoreOutcome.error().RequestId() << std::endl;
  ShutdownSdk();
  return -1;
}
std::string onGoingRestore("ongoing-request=\"false\"");
int maxWaitTimeInSeconds = 600;
while (maxWaitTimeInSeconds > 0)
{
  auto meta = client.HeadObject(SourceBucketName, SourceObjectName);
  std::string restoreStatus = meta.result().HttpMetaData()["x-oss-restore"];
  std::transform(restoreStatus.begin(), restoreStatus.end(), restoreStatus.begin(), ::tolower);
  if (!restoreStatus.empty() &&
  restoreStatus.compare(0, onGoingRestore.size(), onGoingRestore)==0) {
    std::cout << " success, restore status:" << restoreStatus << std::endl;
    /* 成功解冻归档文件*/
    break;
  }
  std::cout << " info, WaitTime:" << maxWaitTimeInSeconds
  << "; restore status:" << restoreStatus << std::endl;
  std::this_thread::sleep_for(std::chrono::seconds(10));
  maxWaitTimeInSeconds--;
}
/* 设置修改后的文件存储类型，例如将修改后的文件存储类型设置为低频 */
ObjectMetaData objectMeta;
objectMeta.addHeader("x-oss-storage-class", "IA");
CopyObjectRequest request(SourceBucketName, SourceBucketName,objectMeta);
request.setCopySource(SourceBucketName, SourceObjectName);
/* 修改为上述指定的文件存储类型 */
auto outcome = client.CopyObject(request);
if (!outcome.isSuccess()) {
  /* 异常处理 */
  std::cout << "CopyObject fail" <<
  ",code:" << outcome.error().Code() <<
  ",message:" << outcome.error().Message() <<
  ",requestId:" << outcome.error().RequestId() << std::endl;
  ShutdownSdk();
  return -1;
}
```

```
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

各种存储类型的存储费用介绍，请参见计量项和计费项的计量计费概述一节。

11.8.6. 列举文件

本文介绍如何列举指定存储空间（Bucket）的所有文件、指定个数的文件以及指定目录下的文件大小等。

OSS 文件按照字母顺序排列。您可以通过OssClient.ListObjects列出存储空间下的文件。ListObjects有以下三类参数格式：

- ListObjectOutcome ListObjects(const std::string& bucket) const：列举存储空间下的文件。最多列举1000个文件。
- ListObjectOutcome ListObjects(const std::string& bucket, const std::string& prefix) const： 列举存储空间下指定前缀的文件。最多列举 1000 个文件。
- ListObjectOutcome ListObjects(const ListObjectsRequest& request) const：提供多种过滤功能，实现灵活的查询功能。

主要参数及说明如下：

参数	描述
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组（commonPrefixes）。
prefix	限定返回的文件必须以prefix作为前缀。
maxKeys	限定此次列举文件的最大个数。默认值为100，最大值为1000。
marker	列举指定marker之后的文件。

简单列举文件

以下代码用于列举指定存储空间下的文件，默认列举100个文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息*/
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 列举文件 */
   ListObjectsRequest request(BucketName);
   auto outcome = client.ListObjects(request);
   if (!outcome.isSuccess())  {
      /* 异常处理 */
      std::cout << "ListObjects fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   else {
      for (const auto& object : outcome.result().ObjectSummarys()) {
         std::cout << "object"<<
         ",name:" << object.Key() <<
         ",size:" << object.Size() <<
         ",lastmodify time:" << object.LastModified() << std:: endl;
      }
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

列举指定个数的文件

以下代码用于列举指定个数的文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf) ;
   /* 列举文件 */
   ListObjectsRequest request(BucketName);
   /* 设置最大个数 */
   request.setMaxKeys(200);
   auto outcome = client.ListObjects(request);
   if (!outcome.isSuccess( )) {
      /* 异常处理 */
      std::cout << "ListObjects fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   else {
      for (const auto& object : outcome.result().ObjectSummarys()) {
         std::cout << "object"<<
         ",name:" << object.Key() <<
         ",size:" << object.Size() <<
         ",lastmodify time:" << object.LastModified() << std ::endl;
      }
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

列举指定目录下的文件和子目录

以下代码用于列举指定目录下的文件和子目录：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string keyPrefix = "yourkeyPrefix ";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   std::string nextMarker = "";
   bool isTruncated = false;
   do {
      /* 列举文件 */
      ListObjectsRequest request(BucketName);
       /* 设置正斜线（/）为文件夹的分隔符 */
      request.setDelimiter("/");
      request.setPrefix(keyPrefix);
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess ()) {
         /* 异常处理 */
         std::cout << "ListObjects fail" <<
         ",code:" << outcome.error().Code() <<
         ",message:" << outcome.error().Message() <<
         ",requestId:" << outcome.error().RequestId() << std::endl;
         break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
         std::cout << "object"<<
         ",name:" << object.Key() <<
         ",size:" << object.Size() <<
         ",lastmodify time:" << object.LastModified() << std::endl;
      }
      for (const auto& commonPrefix : outcome.result().CommonPrefixes()) {
         std::cout << "commonPrefix" << ",name:" << commonPrefix << std::endl;
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
   } while (isTruncated);
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

列举指定目录下的文件大小

以下代码用于获取指定目录下的文件大小：

```
#include <alibabacloud/oss/OssClient.h>
```

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  static int64_t calculateFolderLength (const OssClient &client, const std::string &bucketName, const std::st
ring &folder)
  {
    std::string nextMarker = "";
    bool isTruncated = false;
    int64_t size = 0;
    do {
      /* 列举文件 */
      ListObjectsRequest request(bucketName);
      request.setPrefix(folder);
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess()) {
        /* 异常处理 */
        std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        size += object.Key()
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
   } while (isTruncated);
   return size;
  }
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string keyPrefix = "yourkeyPrefix";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  bool isTruncated = false;
  do {
    /* 列举文件 */
    ListObjectsRequest request(BucketName);
    /* 设置正斜线（/）为文件夹的分隔符 */
    request.setDelimiter("/");
    request.setPrefix(keyPrefix);
    request.setMarker(nextMarker);
    auto outcome = client.ListObjects(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListObjects fail" <<
```

```
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        std::cout << "object" <<
        ",name:" << object.Key() <<
        ",size:" << object.Size() << std::endl;
      }
      for (const auto& commonPrefix : outcome.result().CommonPrefixes()) {
        int64_t foldersize = calculateFolderLength(client, BucketName, commonPrefix);
        std::cout << "folder" <<
        ",name:" << commonPrefix <<
        ",size:" << foldersize << std::endl;
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
    } while (isTruncated);
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

列举指定前缀的文件

以下代码用于列举包含指定前缀（prefix）的文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string keyPrefix = "yourkeyPrefix";
    /* 初始化网络等资源 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    std::string nextMarker = "";
    bool isTruncated = false;
    do {
        /* 列举文件 */
        ListObjectsRequest request(BucketName);
        /* 指定前缀 */
        request.setPrefix(keyPrefix);
        request.setMarker(nextMarker);
        auto outcome = client.ListObjects(request);
        if (!outcome.isSuccess()) {
            /* 异常处理 */
            std::cout << "ListObjects fail" <<
         ",code:" << outcome.error().Code() <<
         ",message:" << outcome.error().Message() <<
         ",requestId:" << outcome.error().RequestId() << std::endl;
                break;
        }
       for (const auto& object : outcome.result().ObjectSummarys()) {
           std::cout << "object"<<
        ",name:" << object.Key() <<
        ",size:" << object.Size() <<
        ",lastmodify time:" << object.LastModified() << std::endl;
       }
        nextMarker = outcome.result().NextMarker();
        isTruncated = outcome.result().IsTruncated();
    } while (isTruncated);
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

列举指定marker之后的文件

参数marker代表文件名称。以下代码用于列举指定marker之后的文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string YourMarker = "yourMarker";
    /* 初始化网络等资源 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   ListObjectOutcome outcome;
    do {
        /* 列举文件 */
        ListObjectsRequest request(BucketName);
        /* 列举指定marker之后的文件 */
        request.setMarker(YourMarker);
        outcome = client.ListObjects(request);
        if (!outcome.isSuccess()) {
            /* 异常处理  */
            std::cout << "ListObjects fail" <<
         ",code:" << outcome.error().Code() <<
         ",message:" << outcome.error().Message() <<
         ",requestId:" << outcome.error().RequestId() << std::endl;
            break;
        }
       YourMarker = outcome.result().NextMarker();
       for (const auto& object : outcome.result().ObjectSummarys()) {
              std::cout << "object"<<
           ",name:" << object.Key() <<
         ",size:" << object.Size() <<
         ",lastmodify time:" << object.LastModified() << std::endl;
       }
    } while (outcome.result().IsTruncated());
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

指定文件名称编码

如果文件名称含有以下特殊字符，需要进行编码传输。OSS目前仅支持URL编码。

- 单引号（''）
- 双引号（""）
- and符号（&）
- 尖括号（< >）
- 顿号（、）
- 中文

以下代码用于指定文件名称编码：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   std::string nextMarker = "";
   bool isTruncated = false;
   do {
     /* 列举文件 */
     ListObjectsRequest request(BucketName) ;
     /* 指定文件名称编码 */
     request.setEncodingType("url");
     request.setMarker(nextMarker);
     auto outcome = client.ListObjects(request);
     if (!outcome.isSuccess() ) {
       /* 异常处理 */
       std::cout << "ListObjects fail" <<
       ",code:" << outcome.error().Code() <<
       ",message:" << outcome.error().Message() <<
       ",requestId:" << outcome.error().RequestId() << std::endl;
       break;
     }
     for (const auto& object : outcome.result().ObjectSummarys()) {
       std::cout << "object"<<
       ",name:" << object.Key() <<
       ",size:" << object.Size() <<
       ",lastmodify time:" << object.LastModified() << std::endl;
     }
     nextMarker = outcome.result().NextMarker();
     isTruncated = outcome.result().IsTruncated();
   } while (isTruncated);
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为objects。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见文件夹功能。创建文件夹的完整代码请参见GitHub。

假设存储空间中包含文件 `oss.jpg` 、 `fun/test.jpg` 、 `fun/movie/001.avi` 和 `fun/movie/007.avi` ，以正斜线（/）作为文件夹的分隔符。以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举存储空间下所有文件

  以下代码用于列举指定存储空间下的所有文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  bool isTruncated = false;
  do {
    /* 列举文件 */
    ListObjectsRequest request(BucketName);
    request.setMarker(nextMarker);
    auto outcome = client.ListObjects(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListObjects fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    else {
      for (const auto& object : outcome.result().ObjectSummarys()) {
        std::cout << "object"<<
        ",name:" << object.Key() <<
        ",size:" << object.Size() <<
        ",lastmodify time:" << object.LastModified() << std::endl;
      }
    }
    nextMarker = outcome.result().NextMarker();
    isTruncated = outcome.result().IsTruncated();
  } while (isTruncated);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

返回结果如下：

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg
CommonPrefixes:
```

- 列举指定目录下所有文件

  以下代码用于列举指定目录下的所有文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  bool isTruncated = false;
  do {
      /* 列举文件 */
      ListObjectsRequest request(BucketName);
      request.setPrefix("fun/");
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess()) {
        /* 异常处理 */
        std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        std::cout << "object"<<
        ",name:" << object.Key() <<
        ",size:" << object.Size() <<
        ",lastmodify time:" << object.LastModified() << std::endl;
      }
      for (const auto& commonPrefix : outcome.result().CommonPrefixes()) {
        std::cout << "commonPrefix" << ",name:" << commonPrefix << std::endl;
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
  } while (isTruncated);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

  返回结果如下：

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
CommonPrefixes:
```

- 列举目录下的文件和子目录

  以下代码用于列举指定目录下的文件和子目录。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  bool isTruncated = false;
  do {
      /* 列举文件 */
      ListObjectsRequest request(BucketName);
      /* 设置正斜线（/）为文件夹的分隔符 */
      request.setDelimiter("/");
      request.setPrefix("fun/");
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess()) {
        /* 异常处理 */
        std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        std::cout << "object"<<
        ",name:" << object.Key() <<
        ",size:" << object.Size() <<
        ",lastmodify time:" << object.LastModified() << std::endl;
      }
      for (const auto& commonPrefix : outcome.result().CommonPrefixes()) {
        std::cout << "commonPrefix" << ",name:" << commonPrefix << std::endl;
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
  } while (isTruncated);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

返回结果如下：

```
Objects:
fun/test.jpg
CommonPrefixes:
fun/movie/
```

- 列举指定目录下的文件大小

  以下代码用于获取指定目录下的文件大小：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
static int64_t calculateFolderLength (const OssClient &client, const std::string &bucketName, const std::s
tring &folder)
{
  std::string nextMarker = "";
  bool isTruncated = false;
  int64_t size = 0;
  do {
      /* 列举文件 */
      ListObjectsRequest request(bucketName);
      request.setPrefix(folder);
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess()) {
        /* 异常处理 */
        std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        size += object.Size();
      }
      nextMarker = outcome.result().NextMarker();
      isTruncated = outcome.result().IsTruncated();
    } while (isTruncated);
   return size;
}
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  bool isTruncated = false;
  do {
    /* 列举文件 */
    ListObjectsRequest request(BucketName);
```

```
    ListObjectsRequest request(BucketName);
    /* 设置正斜线（/）为文件夹的分隔符 */
    request.setDelimiter("/");
    request.setPrefix("fun/");
    request.setMarker(nextMarker);
    auto outcome = client.ListObjects(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "ListObjects fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      break;
    }
    for (const auto& object : outcome.result().ObjectSummarys()) {
      std::cout << "object" <<
      ",name:" << object.Key() <<
      ",size:" << object.Size() << std::endl;
    }
    for (const auto& commonPrefix : outcome.result().CommonPrefixes()) {
      int64_t foldersize = calculateFolderLength(client, BucketName, commonPrefix);
      std::cout << "folder" <<
      ",name:" << commonPrefix <<
      ",size:" << foldersize << std::endl;
    }
    nextMarker = outcome.result().NextMarker();
    isTruncated = outcome.result().IsTruncated();
  } while (isTruncated);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.8.7. 删除文件

您可以根据需要删除单个文件（Object）、删除指定的多个文件或者删除指定前缀的文件。

> **警告** 请您谨慎使用删除操作，文件删除后将无法恢复。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  /* 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  /* yourEndpoint填写Bucket所在地域对应的Endpoint。以华东1（杭州）为例，Endpoint填写为https://oss-cn-hangzhou.aliyuncs.com */
  std::string Endpoint = "yourEndpoint";
  /* 填写Bucket名称，例如examplebucket */
  std::string BucketName = "examplebucket";
  /* 填写Object完整路径，例如exampleobject.txt。Object完整路径中不能包含Bucket名称 */
  /* 当要删除目录时，请将ObjectName设置为对应的目录名称。如果目录非空，则需要将目录下的所有Object删除后才能删除该目录 */
  std::string ObjectName = "exampleobject.txt";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  DeleteObjectRequest request(BucketName, ObjectName);
  /* 删除文件 */
  auto outcome = client.DeleteObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "DeleteObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

批量删除文件

批量删除文件时，每次最多删除1000个文件。您可以删除指定的多个文件或者删除指定前缀的文件。

OSS还支持通过设置生命周期规则来自动删除文件。更多信息，请参见开发指南中的生命周期规则介绍。

返回结果包括如下两种模式，默认返回模式为详细模式，请根据实际选择返回模式。

- 详细模式（verbose）：返回所有删除的文件列表。
- 简单模式（quiet）：只返回删除失败的文件列表。
- 删除指定的多个文件

  以下代码用于删除examplebucket中指定的多个文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string ObjectName = "yourObjectName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   DeleteObjectsRequest request(BucketName);
   /* 添加要删除文件的文件名 */
   request.addKey(ObjectName);
   /* 删除文件 */
   auto outcome = client.DeleteObjects(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "DeleteObjects fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

- 删除指定前缀的文件

  以下代码用于删除examplebucket中以file为前缀的文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string keyPrefix = "yourkeyPrefix";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::string nextMarker = "";
  do {
      /*列举文件*/
      ListObjectsRequest request(BucketName);
      /*指定前缀*/
      request.setPrefix(keyPrefix);
      request.setMarker(nextMarker);
      auto outcome = client.ListObjects(request);
      if (!outcome.isSuccess()) {
        /*异常处理*/
        std::cout << "ListObjects fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      for (const auto& object : outcome.result().ObjectSummarys()) {
        DeleteObjectRequest request(BucketName, object.Key());
        /* 删除文件 */
        auto delResult = client.DeleteObject(request);
      }
      nextMarker = outcome.result().NextMarker();
  } while (outcome.result().IsTruncated());
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

11.8.8. 拷贝文件

拷贝文件分为拷贝小文件和拷贝大文件。

拷贝小文件

对于小于 1GB 的文件，您可以通过 client.CopyObject 方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。此方法参数指定方式如下

参数指定方式	描述
CopyObjectOutcome OssClient::CopyObject(const CopyObjectRequest &request)	允许指定目标文件的元信息和拷贝的限制条件。如果拷贝操作的源文件地址和目标文件地址相同，则直接替换源文件的元信息。

说明

以下代码用于拷贝小文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string SourceBucketName = "yourSourceBucketName";
   std::string CopyBucketName = "yourCopyBucketName";
   std::string SourceObjectName = "yourSourceObjectName";
   std::string CopyObjectName = "yourCopyObjectName";
   /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   CopyObjectRequest request(CopyBucketName, CopyObjectName);
   request.setCopySource(SourceBucketName, SourceObjectName);
   /* 拷贝文件 */
   auto outcome = client.CopyObject(request);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "CopyObject fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：

1. 通过 client.InitiateMultipartUpload 初始化分片拷贝任务。
2. 通过 client.UploadPartCopy 进行分片拷贝。除最后一个分片外，其它分片都要大于 100KB。
3. 通过 client.CompleteMultipartUpload 提交分片拷贝任务。

说明 拷贝大文件也不支持跨地域拷贝。例如，不支持将北京存储空间里的文件拷贝到青岛。

以下代码用于拷贝大文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto getObjectMetaReq = GetObjectMetaRequest(SourceBucketName, SourceObjectName);
  auto getObjectMetaResult = GetObjectMeta(getObjectMetaReq);
  if (!getObjectMetaResult.isSuccess()) {
    std::cout << "GetObjectMeta fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 获取被拷贝文件大小 */
  auto objectSize = getObjectMetaResult.result().ContentLength();
  /* 拷贝大文件 */
  InitiateMultipartUploadRequest initUploadRequest(CopyBucketName, CopyObjectName);
  /* 初始化分片拷贝事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  int64_t partSize = 100 * 1024;
  PartList partETagList;
  int partCount = static_cast<int>(objectSize / partSize);
  /* 计算分片个数*/
  if (objectSize % part Size != 0) {
    partCount++;
  }
  /* 对每一个分片进行拷贝 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < objectSize - skipBytes) ? partSize : (objectSize - skipBytes);
    auto uploadPartCopyReq = UploadPartCopyRequest(CopyBucketName, CopyObjectName, SourceBucket
Name, SourceObjectName,uploadId, i + 1);
    uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes + size -1);
    auto uploadPartOutcome = client.UploadPartCopy(uploadPartCopyReq);
    if (uploadPartOutcome.isSuccess()) {
```

```
        Part part(i, uploadPartOutcome.result().ETag());
        partETagList.push_back(part);
      }
      else {
        std::cout << "UploadPartCopy fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
      }
    }
    /* 完成分片拷贝 */
    CompleteMultipartUploadRequest request(CopyBucketName, CopyObjectName, partETagList, uploadId);
    auto outcome = client.CompleteMultipartUpload(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "CompleteMultipartUpload fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
  }
```

关于分片拷贝的更多信息，请参见UploadPartCopy。

11.8.9. 禁止覆盖同名文件

默认情况下，如果新添加文件与现有文件（Object）同名且对该文件有访问权限，则新添加的文件将覆盖原有的文件。本文介绍如何通过设置请求头x-oss-forbid-overwrite在简单上传、拷贝文件及分片上传等场景中禁止覆盖同名文件。

简单上传

以下代码用于简单上传时禁止覆盖同名文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  /*指定上传文件操作时是否覆盖同名Object*/
  /*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
  /*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
  /*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
  ObjectMetaData meta;
  meta.addHeader("x-oss-forbid-overwrite", "true");
  PutObjectRequest request(BucketName, ObjectName, content, meta);
  /*上传文件*/
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
      /*异常处理*/
      std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

简单上传的更多详情，请参见PutObject。

拷贝文件

- 拷贝小文件

  以下代码用于拷贝小文件时禁止覆盖同名文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*指定上传文件操作时是否覆盖同名Object*/
  /*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
  /*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
  /*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
  ObjectMetaData meta;
  meta.addHeader("x-oss-forbid-overwrite", "true");
  CopyObjectRequest request(CopyBucketName, CopyObjectName, meta);
  request.setCopySource(SourceBucketName, SourceObjectName);
  /*拷贝文件*/
  auto outcome = client.CopyObject(request);
  if (!outcome.isSuccess()) {
      /*异常处理*/
      std::cout << "CopyObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

- 拷贝大文件

  以下代码用于拷贝大文件（分片拷贝）时禁止覆盖同名文件：

```
#include <alibabacloud/oss/OssClient.h>
#include <sstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
```

```
std::string SourceObjectName = "yourSourceObjectName";
std::string CopyObjectName = "yourCopyObjectName";
/*初始化网络等资源*/
InitializeSdk();
ClientConfiguration conf;
OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
auto getObjectMetaReq = GetObjectMetaRequest(SourceBucketName, SourceObjectName);
auto getObjectMetaResult = GetObjectMeta(getObjectMetaReq);
if (!getObjectMetaResult.isSuccess()) {
  std::cout << "GetObjectMeta fail" <<
  ",code:" << outcome.error().Code() <<
  ",message:" << outcome.error().Message() <<
  ",requestId:" << outcome.error().RequestId() << std::endl;
  ShutdownSdk();
  return -1;
}
  /*获取被拷贝文件大小*/
auto objectSize = getObjectMetaResult.result().ContentLength();
/*拷贝大文件*/
ObjectMetaData metaData;
std::stringstream ssDesc;
ssDesc << "/" << SourceBucketName << "/" << SourceObjectName;
std::string value = ssDesc.str();
metaData.addHeader("x-oss-copy-source", value);
/*指定上传文件操作时是否覆盖同名Object*/
/*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
/*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
/*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
metaData.addHeader("x-oss-forbid-overwrite", "true");
InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName, metaData);
/*初始化分片拷贝事件*/
auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
auto uploadId = uploadIdResult.result().UploadId();
int64_t partSize = 100 * 1024;
PartList partETagList;
int partCount = static_cast<int>(objectSize / partSize);
/*计算分片个数*/
if (objectSize % partSize != 0) {
  partCount++;
}
/*对每一个分片进行拷贝*/
for (int i = 1; i <= partCount; i++) {
  auto skipBytes = partSize * (i - 1);
  auto size = (partSize < objectSize - skipBytes) ? partSize : (objectSize - skipBytes);
  auto uploadPartCopyReq = UploadPartCopyRequest(CopyBucketName, CopyObjectName, SourceBuc
ketName, SourceObjectName,uploadId, i + 1);
  uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes + size -1);
  auto uploadPartOutcome = client.UploadPartCopy(uploadPartCopyReq);
  if (uploadPartOutcome.isSuccess()) {
    Part part(i, uploadPartOutcome.result().ETag());
    partETagList.push_back(part);
  }
  else {
    std::cout << "UploadPartCopy fail" <<
```

```
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
      }
    }
    /*完成分片拷贝*/
    CompleteMultipartUploadRequest request(CopyBucketName, CopyObjectName, partETagList, uploadId
  );
    /*指定上传文件操作时是否覆盖同名Object*/
    /*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
    /*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
    /*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
    request.MetaData().addHeader("x-oss-forbid-overwrite", "true");
    auto outcome = client.CompleteMultipartUpload(request);
    if (!outcome.isSuccess()) {
      /*异常处理*/
      std::cout << "CompleteMultipartUpload fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
  }
```

拷贝文件的更多详情，请参见CopyObject。

分片上传

以下代码用于分片上传时禁止覆盖同名文件：

```
#include <alibabacloud/oss/OssClient.h>
int64_t getFileSize(const std::string& file)
{
      std::fstream f(file, std::ios::in | std::ios::binary);
    f.seekg(0, f.end);
    int64_t size = f.tellg();
    f.close();
    return size;
}
using namespace AlibabaCloud::OSS;
int main(void)
{
    /*初始化OSS账号信息*/
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string ObjectName = "yourObjectName";
    /*初始化网络等资源*/
    InitializeSdk();
    ClientConfiguration conf;
```

```
OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
/*指定上传文件操作时是否覆盖同名Object*/
/*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
/*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
/*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
ObjectMetaData meta;
data.addHeader("x-oss-forbid-overwrite", "true");
InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName, data);
/*初始化分片上传事件*/
auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
auto uploadId = uploadIdResult.result().UploadId();
std::string fileToUpload = "yourLocalFilename";
int64_t partSize = 100 * 1024;
PartList partETagList;
auto fileSize = getFileSize(fileToUpload);
int partCount = static_cast<int>(fileSize / partSize);
/*计算分片个数*/
if (fileSize % partSize != 0) {
  partCount++;
}
/*对每一个分片进行上传*/
for (int i = 1; i <= partCount; i++) {
  auto skipBytes = partSize * (i - 1);
  auto size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
  std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>(fileToUpload, std::ios::in|std::
ios::binary);
  content->seekg(skipBytes, std::ios::beg);
  UploadPartRequest uploadPartRequest(BucketName, ObjectName, content);
  uploadPartRequest.setContentLength(size);
  uploadPartRequest.setUploadId(uploadId);
  uploadPartRequest.setPartNumber(i);
  auto uploadPartOutcome = client.UploadPart(uploadPartRequest);
  if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
  }
  else {
      std::cout << "uploadPart fail" <<
    ",code:" << uploadPartOutcome.error().Code() <<
    ",message:" << uploadPartOutcome.error().Message() <<
    ",requestId:" << uploadPartOutcome.error().RequestId() << std::endl;
  }
}
/*完成分片上传*/
CompleteMultipartUploadRequest request(BucketName, ObjectName);
request.setUploadId(uploadId);
request.setPartList(partETagList);
/*指定上传文件操作时是否覆盖同名Object*/
/*不指定x-oss-forbid-overwrite时，默认覆盖同名Object*/
/*指定x-oss-forbid-overwrite为false时，表示允许覆盖同名Object*/
/*指定x-oss-forbid-overwrite为true时，表示禁止覆盖同名Object，如果同名Object已存在，程序将报错*/
request.MetaData().addHeader("x-oss-forbid-overwrite", "true");
auto outcome = client.CompleteMultipartUpload(request);
if (!outcome.isSuccess()) {
```

```
        /*异常处理*/
        std::cout << "CompleteMultipartUpload fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
        ShutdownSdk();
        return -1;
    }
     /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

分片上传的更多详情，请参见InitiateMultipartUpload以及CompleteMultipartUpload。

11.8.10. 解冻文件

本文介绍如何解冻归档（Archive）及冷归档（Cold Archive）类型的文件（Object）。

> **说明** 非归档或冷归档类型的文件，不要调用RestoreObject方法。
>
> 归档及冷归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

解冻归档文件

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档文件：

```
#include <alibabacloud/oss/OssClient.h>
#include <thread>
#include <chrono>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
    std::string AccessKeyId = "yourAccessKeyId";
    std::string AccessKeySecret = "yourAccessKeySecret";
    std::string Endpoint = "yourEndpoint";
    std::string BucketName = "yourBucketName";
    std::string ObjectName = "yourObjectName";
    /* 初始化网络等资源 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /* 解冻归档文件 */
```

```
  /* 解冻归档文件 */
  auto outcome = client.RestoreObject(BucketName, ObjectName);
  /* 非归档文件不能解冻 */
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "RestoreObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  std::string onGoingRestore("ongoing-request=\"false\"");
  int maxWaitTimeInSeconds = 600;
  while (maxWaitTimeInSeconds > 0)
  {
    auto meta = client.HeadObject(BucketName, ObjectName);
    std::string restoreStatus = meta.result().HttpMetaData()["x-oss-restore"];
    std::transform(restoreStatus.begin(), restoreStatus.end(), restoreStatus.begin(), ::tolower);
    if (!restoreStatus.empty() &&
    restoreStatus.compare(0, onGoingRestore.size(), onGoingRestore)==0) {
      std::cout << " success, restore status:" << restoreStatus << std::endl;
      /* 成功解冻归档文件*/
      break;
    }
    std::cout << " info, WaitTime:" << maxWaitTimeInSeconds
    << "; restore status:" << restoreStatus << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(10));
    maxWaitTimeInSeconds--;
  }
  if (maxWaitTimeInSeconds == 0)
  {
    std::cout << "RestoreObject fail, TimeoutException" << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

解冻冷归档文件

冷归档类型的Object在执行解冻前后的状态变换过程如下：

1. 冷归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态。不同优先级的首字节取回时间如下：
   - 高优先级（Expedited）：表示1小时内完成解冻。
   - 标准（Standard）：表示2~5小时内完成解冻。如果不传入JobParameters节点，则默认为Standard。
   - 批量（Bulk）：表示5~12小时内完成解冻。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。您可以指定解冻的天数，最短是1天，最长是7天。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻冷归档文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 如果上传文件的同时指定文件存储类型为冷归档，请参考如下代码 */
  //auto content = std::make_shared<std::stringstream>("test");
  //PutObjectRequest putRequest(BucketName, ObjectName, content);
  //putRequest.MetaData().addHeader("x-oss-storage-class", "ColdArchive");
  //auto putOutcome = client.PutObject(putRequest);
  RestoreObjectRequest request(BucketName, ObjectName);
  /* 设置解冻之后保持解冻状态的天数，默认值为1天 */
  request.setDays(2);
  /* 设置解冻冷归档类型文件的优先级 */
  request.setTierType(TierType::Expedited);
  auto outcome = client.RestoreObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "Delete Bucket Inventory fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.8.11. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string LinkName = "yourLinkName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置HTTP header */
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  /* 设置自定义文件元信息 */
  meta.UserMetaData()["meta"] = "meta-value";
  /* 创建软链接 */
  CreateSymlinkRequest request(BucketName, ObjectName, meta);
  request.SetSymlinkTarget(LinkObjectName);
  auto outcome = client.CreateSymlink(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CreateSymlink fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string LinkName = "yourLinkName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取object软链接 */
  GetSymlinkRequest request(BucketName, LinkName);
  auto outcome = client.GetSymlink(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "GetSymlink fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  else {
    std::cout << " GetSymlink success Symlink name:" << outcome.result().SymlinkTarget() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取软链接的详细信息请参见GetSymlink。

11.9. 版本控制

11.9.1. 管理版本控制

版本控制应用于存储空间（Bucket）内的所有文件（Object）。

Bucket的版本状态包括非版本化（默认）、开启版本控制及暂停版本控制三种。有关版本控制状态的更多信息，请参见版本控制介绍。

设置Bucket版本控制状态

以下代码用于设置Bucket为开启版本控制（Enabled）或暂停版本控制（Suspended）状态。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*创建bucket版本配置，状态设置为Enabled或Suspended*/
  SetBucketVersioningRequest setrequest(BucketName, VersioningStatus::Enabled);
  auto outcome = client.SetBucketVersioning(setrequest);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketVersioning fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

有关设置Bucket版本控制状态的更多信息，请参见PutBucketVersioning。

获取Bucket版本控制状态信息

以下代码用于获取Bucket的版本控制状态信息。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取bucket版本状态信息*/
  auto outcome = client.GetBucketVersioning(GetBucketVersioningRequest(BucketName));
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "GetBucketVersioning fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

有关获取Bucket版本控制状态的更多信息，请参见GetBucketVersioning。

列举Bucket中所有Object的版本信息

以下代码用于列举指定Bucket中包括删除标记（Delete Marker）在内的所有Object的版本信息。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  ListObjectVersionsRequest request(BucketName);
  bool IsTruncated = false;
  do {
      auto outcome = client.ListObjectVersions(request);
      if (outcome.isSuccess()) {
        /*查看列出的object删除标记的各版本信息*/
        for (auto const &marker : outcome.result().DeleteMarkerSummarys()) {
          std::cout << "marker key:" << marker.Key() << ",marker versionid:" << marker.VersionId() << std::endl
;
        }
        /*查看列出的object各版本信息*/
        for (auto const &obj : outcome.result().ObjectVersionSummarys()) {
           std::cout << "object key:" << obj.Key() << ",object versionid:" << obj.VersionId() << std::endl;
        }
      }
      else {
        std::cout << "ListObjectVersions fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      request.setKeyMarker(outcome.result().NextKeyMarker());
      request.setVersionIdMarker(outcome.result().NextVersionIdMarker());
      IsTruncated = outcome.result().IsTruncated();
  } while (IsTruncated);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.9.2. 上传文件

本文介绍如何在受版本控制的存储空间（Bucket）中上传文件（Object）。

简单上传

在已开启版本控制的Bucket中，OSS会为新添加的Object自动生成唯一的版本ID，并在响应header中通过x-oss-version-id形式返回。在暂停了版本控制的Bucket中，新添加的Object的版本ID为“null”，上传同名Object，后一次会覆盖前一次上传的文件内容。OSS保证同一个Object只会有一个版本ID为“null”。

以下代码用于简单上传：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

简单上传的详细信息请参见PutObject。

追加上传

在受版本控制的Bucket中，仅支持对于当前版本为Appendable类型的Object执行追加（AppendObject）操作，不支持对于历史版本为Appendable类型的Object执行AppendObject操作。

说明

- 对当前版本为Appendable类型的Object执行AppendObject操作时，OSS不会为该Appendable类型的Object生成历史版本。
- 对当前版本为Appendable类型的Object执行PutObject或DeleteObject操作时，OSS会将该Appendable类型的Object保留为历史版本，且该Object不允许继续追加。
- 不支持对当前版本为非Appendable类型的Object（包括Normal Object、Delete Marker等）执行AppendObject操作。

以下代码用于追加上传：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  /* 第一次追加的位置是0，返回值为下一次追加的位置。后续追加的位置是追加前文件的长度*/
  std::shared_ptr<std::iostream> content1 = std::make_shared<std::stringstream>();
  *content1 <<"Thank you for using Aliyun Object Storage Service!";
  AppendObjectRequest request(BucketName, ObjectName, content1, meta);
  request.setPosition(0L);
  /*第一次追加文件*/
  auto result = client.AppendObject(request);
  /*查看本次执行追加文件的object的版本ID*/
  if (result.isSuccess()) {
    std::cout << "versionid:" << result.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "AppendObject fail" <<
    ",code:" << result.error().Code() <<
    ",message:" << result.error().Message() <<
    ",requestId:" << result.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  std::shared_ptr<std::iostream> content2 = std::make_shared<std::stringstream>();
  *content2 <<"Thank you for using Aliyun Object Storage Service!";
  auto position = result.result().Length();
  AppendObjectRequest appendObjectRequest(BucketName, ObjectName, content2);
  appendObjectRequest.setPosition(position);
  /*第二次追加文件*/
```

```
  auto outcome = client.AppendObject(appendObjectRequest);
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "AppendObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

追加上传的详细信息请参见AppendObject。

分片上传

在受版本控制的Bucket中，调用CompleteMultipartUpload接口来完成整个文件的分片上传，OSS会为整个文件生成唯一的版本ID，并在响应header中以x-oss-version-id的形式返回。

以下代码用于分片上传：

```
#include <alibabacloud/oss/OssClient.h>
int64_t getFileSize(const std::string& file)
{
  std::fstream f(file, std::ios::in | std::ios::binary);
  f.seekg(0, f.end);
  int64_t size = f.tellg();
  f.close();
  return size;
}
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName);
  /*初始化分片上传事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  std::string fileToUpload = "yourLocalFilename";
```

```
    int64_t partSize = 100 * 1024;
    PartList partETagList;
    auto fileSize = getFileSize(fileToUpload);
    int partCount = static_cast<int>(fileSize / partSize);
    /* 计算分片个数*/
    if (fileSize % partSize != 0) {
      partCount++;
    }
    /*对每一个分片进行上传 */
    for (int i = 1; i <= partCount; i++) {
      auto skipBytes = partSize * (i - 1);
      auto size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
      std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>(fileToUpload, std::ios::in|std::
ios::binary);
      content->seekg(skipBytes, std::ios::beg);
      UploadPartRequest uploadPartRequest(BucketName, ObjectName, content);
      uploadPartRequest.setContentLength(size);
      uploadPartRequest.setUploadId(uploadId);
      uploadPartRequest.setPartNumber(i);
      auto uploadPartOutcome = client.UploadPart(uploadPartRequest);
      if (uploadPartOutcome.isSuccess()) {
          Part part(i, uploadPartOutcome.result().ETag());
          partETagList.push_back(part);
      }
      else {
          std::cout << "uploadPart fail" <<
        ",code:" << uploadPartOutcome.error().Code() <<
        ",message:" << uploadPartOutcome.error().Message() <<
        ",requestId:" << uploadPartOutcome.error().RequestId() << std::endl;
      }
    }
    /* 完成分片上传 */
    CompleteMultipartUploadRequest request(BucketName, ObjectName);
    request.setUploadId(uploadId);
    request.setPartList(partETagList);
    auto outcome = client.CompleteMultipartUpload(request);
    if (outcome.isSuccess()) {
      std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
    }
    else {
      /* 异常处理 */
      std::cout << "CompleteMultipartUpload fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

分片上传的详细信息请参见CompleteMultipartUpload。

11.9.3. 下载文件

默认情况下，在受版本控制的存储空间（Bucket）中调用GetObject接口仅返回文件（Object）的当前版本。

对某个Bucket执行GetObject操作时，分以下三种情况：

- 如果该Bucket中Object的当前版本是删除标记（Delete Marker），则返回404 Not Found。
- 如果在查询参数中指定Object的versionId，则返回指定的Object版本。当versionId指定为“null”时，则返回versionId为“null”的Object版本。
- 通过指定versionId的方式来获取删除标记时，则返回405 Method Not Allowed。

以下代码用于下载文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取文件到本地内存*/
  GetObjectRequest request(BucketName, ObjectName);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
    std::cout << "getObjectToBuffer" << " success, Content-Length:" << outcome.result().Metadata().Content
Length() << std::endl;
    /*读取下载的object内容*/
    std::string content;
    *(outcome.result().Content()) >> content;
    std::cout << "getObjectToBuffer" << "content:" << content << std::endl;
    /*查看本次下载object的版本Id*/
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
      /*异常处理*/
      std::cout << "getObjectToBuffer fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

下载文件的详细信息请参见Get Object。

11.9.4. 拷贝文件

本文介绍如何在受版本控制的存储空间（Bucket）中拷贝文件（Object）。您可以通过CopyObject的方法拷贝小于1 GB的文件，通过分片拷贝（UploadPartCopy）的方法拷贝大于1 GB的文件。

拷贝小文件

对于小于1 GB的文件，您可以通过CopyObject方法将文件从一个存储空间（源存储空间）复制到同一地域的另一个存储空间（目标存储空间）。

- x-oss-copy-source默认拷贝Object的当前版本。如果当前版本是删除标记，则返回404表示该Object不存在。您可以在x-oss-copy-source中加入versionId来拷贝指定的Object版本，删除标记不能被拷贝。
- 您可以将Object的早期版本拷贝到同一个Bucket中，拷贝Object的历史版本将会成为一个新的当前版本，达到恢复Object早期版本的目的。
- 如果目标Bucket已开启版本控制，OSS将会为新拷贝出来的Object自动生成唯一的版本ID，此版本ID将会在响应header 的x-oss-version-id中返回。如果目标Bucket未曾开启或者暂停了版本控制，OSS将会为新拷贝的Object自动生成version ID为”null“的版本，且会覆盖原先versionId为”null“的版本。
- 目标Bucket在开启或暂停版本控制状态下，不支持对Appendable类型Object执行拷贝操作。

以下代码用于拷贝小文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  CopyObjectRequest request(CopyBucketName, CopyObjectName);
  request.setCopySource(SourceBucketName, SourceObjectName);
  request.setVersionId("yourSourceObjectVersionId");
  /* 拷贝指定版本的文件 */
  auto outcome = client.CopyObject(request);
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "CopyObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

拷贝小文件的详细说明请参见CopyObject。

拷贝大文件

对于大于1 GB的文件，需要使用分片拷贝（UploadPartCopy）。

UploadPartCopy默认从一个已存在的Object的当前版本中拷贝数据来上传一个Part。允许通过在请求header：x-oss-copy-source中附带versionId的子条件，实现从Object的指定版本进行拷贝，如x-oss-copy-source：/SourceBucketName/SourceObjectName?versionId=111111。

> **说明** SourceObjectName要进行URL编码。响应中将会返回被拷贝的Object版本ID：x-oss-copy-source-version-id。

如果未指定versionId且拷贝Object的当前版本为删除标记，OSS将返回404 Not Found。通过指定versionId来拷贝删除标记时，OSS将返回400 Bad Request。

以下代码用于分片拷贝：

```
#include <alibabacloud/oss/OssClient.h>
#include <sstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto getObjectMetaReq = GetObjectMetaRequest(SourceBucketName, SourceObjectName);
  getObjectMetaReq.setVersionId("yourSourceObjectVersionid");
  auto getObjectMetaResult = GetObjectMeta(getObjectMetaReq);
  if (!getObjectMetaResult.isSuccess()) {
    std::cout << "GetObjectMeta fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 获取被拷贝文件大小 */
  auto objectSize = getObjectMetaResult.result().ContentLength();
  /* 拷贝大文件 */
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName, metaData);
  /* 初始化分片拷贝事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  int64_t partSize = 100 * 1024;
  PartList partETagList;
  int partCount = static_cast<int>(objectSize / partSize);
  /* 计算分片个数*/
  if (objectSize % partSize != 0) {
    partCount++;
```

```
    partCount++;
  }
  /*对每一个分片进行拷贝 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < objectSize - skipBytes) ? partSize : (objectSize - skipBytes);
    auto uploadPartCopyReq = UploadPartCopyRequest(CopyBucketName, CopyObjectName, SourceBucket
Name, SourceObjectName,uploadId, i + 1);
    uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes + size -1);
    uploadPartCopyReq.setVersionId("yourSourceObjectVersionid");
    auto uploadPartOutcome = client.UploadPartCopy(uploadPartCopyReq);
    if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
    }
    else {
      std::cout << "UploadPartCopy fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
    }
  }
  /* 完成分片拷贝 */
  CompleteMultipartUploadRequest request(CopyBucketName, CopyObjectName, partETagList, uploadId);
  auto outcome = client.CompleteMultipartUpload(request);
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "CompleteMultipartUpload fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

分片拷贝的详细信息请参见UploadPartCopy。

11.9.5. 删除文件

本文介绍如何在受版本控制的存储空间（Bucket）中删除单个或多个文件（Object）以及删除指定前缀的（Prefix）的文件。

版本控制下的删除行为

版本控制下的删除行为说明如下：

- 未指定versionId（临时删除）：

如果在未指定versionId的情况下执行删除操作时，默认不会删除Object的当前版本，而是对当前版本插入删除标记（Delete Marker）。当执行GetObject操作时，OSS会检测到当前版本为删除标记，并返回 `404 Not Found` 。此外，响应中会返回 `header：x-oss-delete-marker = true` 以及新生成的删除标记的版本号 `x-oss-version-id` 。

`x-oss-delete-marker` 的值为true，表示与返回的 `x-oss-version-id` 对应的版本为删除标记。

- 指定versionId（永久删除）：

  如果在指定versionId的情况下执行删除操作时，OSS会根据 `params` 中指定的 `versionId` 参数永久删除该版本。如果要删除ID为“null”的版本，请在 `params` 参数中添加 `params['versionId'] = “null”` ，OSS将“null”字符串当成“null”的versionId，从而删除versionId为“null”的Object。

删除单个文件

以下提供了永久删除及临时删除单个Object的示例。

- 永久删除

  以下代码用于指定versionId对Object的指定版本进行永久删除。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 删除指定versionId的Object或指定versionId的删除标记关联的Object。*/
  auto outcome = client.DeleteObject(DeleteObjectRequest(BucketName, ObjectName, "yourObjectVersio
nIdOrDeleteMarkerVersionId"));
  /* 如果指定的是Object的versionId，则返回的delete_marker为None且返回的versionId为指定Object的versio
nId。*/
  /* 如果指定的是删除标记的versionId，则返回的delete_marker为True且返回的versionId为指定删除标记的ver
sionId。*/
  if (!outcome.isSuccess()) {
    /* 异常处理。*/
    std::cout << "DeleteObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

- 临时删除

以下代码用于不指定versionId对Object进行临时删除。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  DeleteObjectRequest request(BucketName, ObjectName);
  /* 不指定versionId对Object进行临时删除，此操作会为Object添加删除标记。*/
  auto outcome = client.DeleteObject(request);
  /* 查看返回删除标记的versionId。*/
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << ",DeleteMarker:" << outcome.result().Delet
eMarker() << std::endl;
  }
  else {
    /* 异常处理。*/
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

删除单个文件的详细信息请参见DeleteObject。

删除多个文件

以下提供了永久删除以及临时删除多个Object的示例。

- 永久删除

  以下代码用于永久删除多个指定versionId的Object或指定versionId的删除标记关联的Object。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  DeleteObjectVersionsRequest  request(BucketName);
  /* 添加需要删除的Object或删除标记的versionId。*/
  ObjectIdentifier obj1("yourObject1Name");
  obj1.setVersionId("yourVersionId");
  ObjectIdentifier obj2("yourObject2Name");
  obj2.setVersionId("obj2_del_marker_versionid");
  request.addObject(obj1);
  request.addObject(obj2);
  /* 批量删除指定versionId的Object或指定删除标记versionId关联的Object。*/
  auto outcome = client.DeleteObjectVersions(request);
  if (!outcome.isSuccess()) {
    /* 异常处理。*/
    std::cout << "DeleteObjectVersions fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

删除多个文件的详细信息请参见DeleteMultipleObjects。

- 临时删除

  以下代码用于不指定versionId对多个Object进行临时删除。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  DeleteObjectVersionsRequest request(BucketName);
  /* 填写要删除Object的名称。*/
  ObjectIdentifier obj1(ObjectName1);
  ObjectIdentifier obj2(ObjectName2);
  ObjectIdentifier obj3(ObjectName3);
  request.addObject(obj1);
  request.addObject(obj2);
  request.addObject(obj3);
  /* 删除Object。*/
  auto outcome = client.DeleteObjectVersions(request);
  if (outcome.isSuccess()) {
    for (auto const &obj : outcome.result().DeletedObjects()) {
      std::cout << "versionid:" << obj.VersionId() << ",DeleteMarker:" << obj.DeleteMarker() << std::endl;
    }
  }
  else {
    /* 异常处理。*/
    std::cout << "DeleteObjectVersions fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

删除指定前缀的文件

以下代码用于删除指定前缀（prefix）的文件。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息。*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源。*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  ListObjectVersionsRequest request(BucketName);
  bool IsTruncated = false;
  do {
      request.setPrefix("yourkeyPrefix");
      auto outcome = client.ListObjectVersions(request);
      if (outcome.isSuccess()) {
        /* 查看列举的Object以及删除标记的各版本信息。*/
        for (auto const &marker : outcome.result().DeleteMarkerSummarys()) {
          client.DeleteObject(DeleteObjectRequest(bucket, marker.Key(), marker.VersionId()));
        }
        /* 列举所有指定前缀的Object的版本信息并删除这些文件。*/
        for (auto const &obj : outcome.result().ObjectVersionSummarys()) {
           client.DeleteObject(DeleteObjectRequest(bucket, obj.Key(), obj.VersionId()));
        }
      }
      else {
        std::cout << "ListObjectVersions fail" <<
        ",code:" << outcome.error().Code() <<
        ",message:" << outcome.error().Message() <<
        ",requestId:" << outcome.error().RequestId() << std::endl;
        break;
      }
      request.setKeyMarker(outcome.result().NextKeyMarker());
      request.setVersionIdMarker(outcome.result().NextVersionIdMarker());
      IsTruncated = outcome.result().IsTruncated();
  } while (IsTruncated);
  /* 释放网络等资源。*/
  ShutdownSdk();
  return 0;
}
```

11.9.6. 解冻文件

在受版本控制的存储空间（Bucket）中，文件（Object）的各个版本可以对应不同的存储类型。RestoreObject接口默认解冻Object的当前版本，允许通过指定versionId的方式来解冻指定版本的Object。

以下代码用于解冻文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 解冻归档文件 */
  RestoreObjectRequest request(BucketName, ObjectName);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.RestoreObject(request);
  /*查看执行解冻操作的object的版本id*/
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "RestoreObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

解冻文件的详细信息请参见RestoreObject。

11.9.7. 获取文件元信息

默认情况下，在受版本控制的存储空间（Bucket）中调用HeadObject接口仅获取文件（Object）当前版本的meta信息。

> **说明** 如果Object的当前版本为删除标记，则返回404 Not Found。请求参数中指定versionId则返回指定版本Object的meta信息。

以下代码用于获取文件元信息：

```
#include <alibabacloud/oss/OssClient.h>
#include <alibabacloud/oss/model/ObjectMetaData.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取文件的部分元信息 */
  GetObjectMetaRequest request(BucketName, ObjectName);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.GetObjectMeta(request);
  if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "GetObjectMeta fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
  }
  else {
      auto metadata = outcome.result();
      std::cout << " get metadata success, VersionId:" << metadata.VersionId() << std::endl;
  }
  /* 获取文件的全部元信息 */
  HeadObjectRequest request1(BucketName, ObjectName);
  request1.setVersionId("yourObjectVersionId");
  outcome = client.HeadObject(request1);
  if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "HeadObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
ShutdownSdk();
      return -1;
  }
  else {
      auto headMeta = outcome.result();
      std::cout << " get headMeta success, VersionId:" << headMeta.VersionId() << std::endl;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取文件元信息的详情请参见HeadObject、GetObjectMeta。

11.9.8. 管理文件访问权限

本文介绍如何在受版本控制的存储空间（Bucket）中管理文件（Object）的访问权限（ACL）。

设置文件访问权限

PutObjectACL默认设置Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以设置指定Object版本的ACL权限。

以下代码用于设置文件访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置文件访问权限 */
  SetObjectAclRequest request(BucketName, ObjectName);
  request.setAcl(CannedAccessControlList::Private);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.SetObjectAcl(request);
  /*查看本次修改权限操作的object的版本id*/
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "SetObjectAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

设置文件访问权限的详细信息请参见PutObjectACL。

获取文件访问权限

GetObjectACL默认获取Object当前版本的ACL权限。如果Object的当前版本是删除标记（Delete Marker），OSS将返回404 Not Found。请求参数中指定versionId可以获取指定Object版本的ACL权限。

以下代码用于获取文件访问权限：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取文件访问权限 */
  GetObjectAclRequest request(BucketName, ObjectName);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.GetObjectAcl(request);
  /* 查看本次执行获取权限操作的object的版本id */
  if (outcome.isSuccess()) {
    std::cout << "GetObjectAcl success versionid:"
    << outcome.result().VersionId()
    << "Acl:" << outcome.result().Acl() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetObjectAcl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取文件访问权限的详细信息请参见GetObjectACL。

11.9.9. 管理软链接

本文介绍如何在受版本控制的存储空间（Bucket）中管理软链接。

创建软链接

软链接本身也可以有多个版本，每个不同的版本可以指向不同的TargetObject，版本ID由OSS自动生成，在响应header中返回x-oss-version-id。您可以通过创建软链接指向TargetObject的当前版本。

说明 在受版本控制的Bucket中，无法为删除标记（Delete Marker）创建软链接。

以下代码用于创建软链接：

```
#include <alibabacloud/oss/OssClient.h>
#include <alibabacloud/oss/model/ObjectMetaData.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string LinkName = "yourLinkName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*设置HTTP header*/
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  /*设置自定义文件元信息*/
  meta.UserMetaData()["meta"] = "meta-value";
  /* 创建软链接 */
  CreateSymlinkRequest request(BucketName, ObjectName, meta);
  request.SetSymlinkTarget(LinkObjectName);
  auto outcome = client.CreateSymlink(request);
  /*查看上传的软链接的版本id*/
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "CreateSymlink fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接

GetSymlink接口默认获取软链接的当前版本。允许通过指定versionId来获取指定版本。如果软链接的当前版本为删除标记，OSS会返回404 Not Found，在响应header中返回x-oss-delete-marker = true以及版本ID：x-oss-version-id。

> **说明** 获取软链接操作需要您对该软链接有读权限。

以下代码用于获取软链接：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string LinkName = "yourLinkName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 获取指定版本的软链接文件 */
  GetSymlinkRequest request(BucketName, LinkName);
  request.setVersionId("yourObjectVersionId");
  auto outcome = client.GetSymlink(request);
  /* 查看返回的软链接文件的版本id */
  if (outcome.isSuccess()) {
    std::cout << "versionid:" << outcome.result().VersionId()
    << "Symlink name:" << outcome.result().SymlinkTarget() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetSymlink fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取软链接的详细信息请参见GetSymlink。

11.10. 对象标签

11.10.1. 设置对象标签

本文介绍如何设置对象标签（Object Tagging）。

说明

上传Object时添加对象标签

- 简单上传时添加对象标签

  以下代码用于简单上传时添加对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 设置x-oss-tagging */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  request.setTagging(tagging.toQueryParameters());
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 分片上传时添加对象标签

  以下代码用于分片上传时添加对象标签：

```
#include <alibabacloud/oss/OssClient.h>
int64_t getFileSize(const std::string& file)
{
```

```
  std::fstream f(file, std::ios::in | std::ios::binary);
  f.seekg(0, f.end);
  int64_t size = f.tellg();
  f.close();
  return size;
}
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName);
  /* 设置x-oss-tagging */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  initUploadRequest.setTagging(tagging.toQueryParameters());
  /* 初始化分片上传事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  std::string fileToUpload = "yourLocalFilename";
  int64_t partSize = 100 * 1024;
  PartList partETagList;
  auto fileSize = getFileSize(fileToUpload);
  int partCount = static_cast<int>(fileSize / partSize);
  /* 计算分片个数 */
  if (fileSize % partSize != 0) {
    partCount++;
  }
  /* 对每一个分片进行上传 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>(fileToUpload, std::ios::in|st
d::ios::binary);
    content->seekg(skipBytes, std::ios::beg);
    UploadPartRequest uploadPartRequest(BucketName, ObjectName, content);
    uploadPartRequest.setContentLength(size);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setPartNumber(i);
    auto uploadPartOutcome = client.UploadPart(uploadPartRequest);
    if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
    }
    else {
```

```
        std::cout << "uploadPart fail" <<
        ",code:" << uploadPartOutcome.error().Code() <<
        ",message:" << uploadPartOutcome.error().Message() <<
        ",requestId:" << uploadPartOutcome.error().RequestId() << std::endl;
      }
    }
    /* 完成分片上传 */
    CompleteMultipartUploadRequest request(BucketName, ObjectName);
    request.setUploadId(uploadId);
    request.setPartList(partETagList);
    auto outcome = client.CompleteMultipartUpload(request);
    if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "CompleteMultipartUpload fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
    }
    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
  }
```

- 追加上传时添加对象标签

  以下代码用于追加上传时添加对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto meta = ObjectMetaData();
  meta.setContentType("text/plain");
  /* 追加文件，默认追加position为0 */
  std::shared_ptr<std::iostream> content1 = std::make_shared<std::stringstream>();
  *content1 <<"Thank you for using Aliyun Object Storage Service!";
  AppendObjectRequest request(BucketName, ObjectName, content1, meta);
  /* 设置标签 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  request.setTagging(tagging.toQueryParameters());
  /* 追加文件 */
  auto result = client.AppendObject(request);
  if (!result.isSuccess()) {
    /* 异常处理 */
    std::cout << "AppendObject fail" <<
    ",code:" << result.error().Code() <<
    ",message:" << result.error().Message() <<
    ",requestId:" << result.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 断点续传上传时添加对象标签

  以下代码用于断点续传上传时添加对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string UploadFilePath = "yourUploadfilePath";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 断点续传上传 */
  UploadObjectRequest request(BucketName, ObjectName, YourUploadFileName);
  /* 设置对象标签 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  request.setTagging(tagging.toQueryParameters());
  auto outcome = client.ResumableUploadObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "ResumableUploadObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

对已上传Object添加或更改对象标签

以下代码用于对已上传Object添加或更改对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  /* 设置对象标签 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  auto putTaggingOutcome = client.SetObjectTagging(SetObjectTaggingRequest(BucketName, ObjectName,
tagging));
  if (!putTaggingOutcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetObjectTagging fail" <<
    ",code:" << putTaggingOutcome.error().Code() <<
    ",message:" << putTaggingOutcome.error().Message() <<
    ",requestId:" << putTaggingOutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

拷贝Object时设置对象标签

拷贝Object时，可以指定如何设置目标Object的对象标签。取值如下：

- Copy（默认值）：复制源Object的对象标签到目标Object。
- Replace：忽略源Object的对象标签，直接采用请求中指定的对象标签。

以下分别提供了简单拷贝1 GB以下的Object、及分片拷贝1 GB以上的Object时设置对象标签的详细示例。

- 以下代码用于简单拷贝1GB以下的Object时设置对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 设置x-oss-tagging */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  request.setTagging(tagging.toQueryParameters());
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  /* 拷贝文件，设置标签 */
  CopyObjectRequest cpRequest(yourBucketName, CopyObjectName);
  cpRequest.setCopySource(yourBucketName, yourObjectName);
  Tagging tagging2;
  tagging2.addTag(Tag("key1-2", "value1-2"));
  tagging2.addTag(Tag("key2-2", "value2-2"));
  tagging2.addTag(Tag("key3-2", "value3-2"));
  cpRequest.setTagging(tagging2.toQueryParameters());
  /* 忽略源Object的对象标签，直接替换为请求中指定的对象标签 */
  cpRequest.setTaggingDirective(CopyActionList::Replace);
  /* 拷贝文件 */
  auto copyoutcome = client.CopyObject(cpRequest);
  if (!copyoutcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CopyObject fail" <<
    ",code:" << copyoutcome.error().Code() <<
    ",message:" << copyoutcome.error().Message() <<
    ",requestId:" << copyoutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

- 以下代码用于分片拷贝1GB 以上的Object时设置对象标签：

```
#include <alibabacloud/oss/OssClient.h>
```

```
#include <sstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SourceBucketName = "yourSourceBucketName";
  std::string CopyBucketName = "yourCopyBucketName";
  std::string SourceObjectName = "yourSourceObjectName";
  std::string CopyObjectName = "yourCopyObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  auto getObjectMetaReq = GetObjectMetaRequest(SourceBucketName, SourceObjectName);
  auto getObjectMetaResult = GetObjectMeta(getObjectMetaReq);
  if (!getObjectMetaResult.isSuccess()) {
    std::cout << "GetObjectMeta fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 获取被拷贝文件的大小 */
  auto objectSize = getObjectMetaResult.result().ContentLength();
  /* 拷贝大文件 */
  ObjectMetaData metaData;
  std::stringstream ssDesc;
  ssDesc << "/" << SourceBucketName << "/" << SourceObjectName;
  std::string value = ssDesc.str();
  metaData.addHeader("x-oss-copy-source", value);
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName, metaData);
  /* 设置x-oss-tagging */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  initUploadRequest.setTagging(tagging.toQueryParameters());
  /* 初始化分片拷贝事件 */
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest);
  auto uploadId = uploadIdResult.result().UploadId();
  int64_t partSize = 100 * 1024;
  PartList partETagList;
  int partCount = static_cast<int>(objectSize / partSize);
  /* 计算分片个数*/
  if (objectSize % partSize != 0) {
    partCount++;
  }
  /* 对每一个分片进行拷贝 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < objectSize - skipBytes) ? partSize : (objectSize - skipBytes);
    auto uploadPartCopyReq = UploadPartCopyRequest(CopyBucketName, CopyObjectName, SourceBuc
```

```
    auto uploadPartCopyReq = UploadPartCopyRequest(CopyBucketName, CopyObjectName, SourceBuc
ketName, SourceObjectName,uploadId, i + 1);
    uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes + size -1);
    auto uploadPartOutcome = client.UploadPartCopy(uploadPartCopyReq);
    if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
    }
    else {
      std::cout << "UploadPartCopy fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
    }
  }
  /* 完成分片拷贝 */
  CompleteMultipartUploadRequest request(CopyBucketName, CopyObjectName, partETagList, uploadId
);
  auto outcome = client.CompleteMultipartUpload(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CompleteMultipartUpload fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

为软链接文件设置标签

以下代码用于为软链接文件设置标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string LinkName = "yourLinkName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  /* 创建软链接 */
  CreateSymlinkRequest csRequest(BucketName, ObjectName);
  csRequest.SetSymlinkTarget(LinkName);
  /* 设置x-oss-tagging */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  csRequest.setTagging(tagging.toQueryParameters());
  auto csoutcome = client.CreateSymlink(csRequest);
  if (!csoutcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CreateSymlink fail" <<
    ",code:" << csoutcome.error().Code() <<
    ",message:" << csoutcome.error().Message() <<
    ",requestId:" << csoutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.10.2. 获取对象标签

本文介绍如何获取对象标签（Object Tagging）。

对象标签使用一组键值对（Key-Value）来标记对象。对象标签的详情请参考开发指南的对象标签。

以下代码用于获取对象标签信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  /* 设置对象标签 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  auto putTaggingOutcome = client.SetObjectTagging(SetObjectTaggingRequest(BucketName, ObjectName,
tagging));
  /* 获取对象标签 */
  getTaggingOutcome = client.GetObjectTagging(GetObjectTaggingRequest(BucketName, key));
  if (!getTaggingOutcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "getTaggingOutcome fail" <<
    ",code:" << getTaggingOutcome.error().Code() <<
    ",message:" << getTaggingOutcome.error().Message() <<
    ",requestId:" << getTaggingOutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  else {
    auto taglist = getTaggingOutcome.result().Tagging();
    for (const auto& tag : taglist)
    {
     std::cout <<"GetObjectTagging success, Key:"
     << tag.Key() << "; Value:" << tag.Value() << std::endl;
    }
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

获取对象标签详情请参考GetObjectTagging。

11.10.3. 删除对象标签

本文介绍如何删除对象标签（Object Tagging）。

以下代码用于删除对象标签：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  /* 设置对象标签 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  auto putTaggingOutcome = client.SetObjectTagging(SetObjectTaggingRequest(BucketName, ObjectName,
tagging));
  /* 删除对象标签 */
  auto delTaggingOutcome = client.DeleteObjectTagging(DeleteObjectTaggingRequest(BucketName, Object
Name));
  if (!delTaggingOutcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "delTaggingOutcome fail" <<
    ",code:" << delTaggingOutcome.error().Code() <<
    ",message:" << delTaggingOutcome.error().Message() <<
    ",requestId:" << delTaggingOutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

删除对象标签详情请参考DeleteObjectTagging。

11.10.4. 对象标签和生命周期管理

生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为生命周期规则生效的条件。

说明 标签条件中标签的Key和Value必须同时匹配。同一个规则中，如果同时配置了前缀和多个对象标签，则只有同时满足前缀且匹配规则中所有对象标签的对象（Object），才视为适用于该规则。

生命周期规则中添加标签匹配规则

以下代码用于生命周期规则中添加标签匹配规则：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  SetBucketLifecycleRequest request(BucketName);
  /* 设置标签规则 */
  Tagging tagging;
  tagging.addTag(Tag("key1", "value1"));
  tagging.addTag(Tag("key2", "value2"));
  /* 指定生命周期 */
  auto rule1 = LifecycleRule();
  rule1.setID("rule1");
  rule1.setPrefix("test1/");
  rule1.setStatus(RuleStatus::Enabled);
  rule1.setExpiration(3);
  rule1.setTags(tagging.Tags());
  /* 在生命周期规则中设置标签信息 */
  LifecycleRuleList list{rule1};
  request.setLifecycleRules(list);
  auto outcome = client.SetBucketLifecycle(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketLifecycle fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

查看生命周期规则中匹配的标签信息

以下代码用于查看生命周期规则中匹配的标签信息：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 查看生命周期规则，包含涉及的标签信息 */
  auto outcome = client.GetBucketLifecycle(BucketName);
  if (outcome.isSuccess()) {
    std::cout << "GetBucketLifecycle success," << std::endl;
    for (auto const rule : outcome.result().LifecycleRules()) {
      std::cout << "rule:" << rule.ID() << "," << rule.Prefix() << "," << rule.Status() << ","
      "hasExpiration:" << rule.hasExpiration() << "," <<
      "hasTransitionList:" << rule.hasTransitionList() << "," << std::endl;
     auto taglist = rule.Tags();
     for (const auto& tag : taglist)
     {
       std::cout <<"GetBucketLifecycle tag success, Key:"
       << tag.Key() << "; Value:" << tag.Value() << std::endl;
     }
    }
  }
  else {
    /* 异常处理 */
    std::cout << "GetBucketLifecycle fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.11. 数据加密

11.11.1. 服务器端加密

OSS支持在服务器端对上传的数据进行加密编码（Server-Side Encryption）。上传数据时，OSS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

背景信息

OSS有如下两种服务器端加密方式：

- 使用OSS托管的CMK进行加密（SSE-KMS）

  上传文件时，可以使用指定的CMK ID或者OSS托管的CMK进行加密操作。数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。

  > **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用，费用详情请参见KMS计费标准。

- 使用OSS完全托管加密（SSE-OSS）

  上传文件时，OSS服务端使用完全托管的AES256进行加密操作。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

> **注意**
>
> - 同一个Object在同一时间内仅可以使用一种服务器端加密方式。
> - 如果配置了Bucket加密，仍然可以在上传或拷贝Object时单独对Object配置加密方式，且以Object配置的加密方式为准。详情请参见Put Object。
> - 更多关于服务器端加密的介绍请参见服务器端加密。

配置Bucket加密

您可以通过以下代码设置Bucket默认加密方式，设置成功之后，所有上传至该Bucket但未设置加密方式的Object都会使用Bucket默认加密方式进行加密：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  SetBucketEncryptionRequest setrequest(BucketName);
  setrequest.setSSEAlgorithm(SSEAlgorithm::KMS);
undefined  /*设置KMS服务端加密*/
  auto outcome = client.SetBucketEncryption(setrequest);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "SetBucketEncryption fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

获取Bucket加密配置

以下代码用于获取Bucket加密配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取服务端加密*/
  GetBucketEncryptionRequest request(BucketName);
  auto outcome = client.GetBucketEncryption(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "GetBucketEncryption fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

删除Bucket加密配置

以下代码用于删除Bucket加密配置：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*删除服务端加密*/
  DeleteBucketEncryptionRequest request(BucketName);
  auto outcome = client.DeleteBucketEncryption(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "DeleteBucketEncryption fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

11.11.2. 客户端加密

客户端加密是指将数据发送到OSS之前在用户本地进行加密。

免责声明

- 使用客户端加密功能时，您需要对主密钥的完整性和正确性负责。因您维护不当导致主密钥用错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。
- 在对加密数据进行复制或者迁移时，您需要对加密元信息的完整性和正确性负责。因您维护不当导致加密元信息出错或丢失，从而导致加密数据无法解密所引起的一切损失和后果均由您自行承担。

背景信息

使用客户端加密时，会为每个Object生成一个随机数据加密密钥，用该随机数据加密密钥明文对Object的数据进行对称加密。主密钥用于生成随机的数据加密密钥，加密后的内容会当作Object的meta信息保存在服务端。解密时先用主密钥将加密后的随机密钥解密出来，再用解密出来的随机数据加密密钥明文解密Object的数据。主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。

注意

- 客户端加密支持分片上传超过5 GB的文件。在使用分片方式上传文件时，需要指定上传文件的总大小和分片大小， 除了最后一个分片外，每个分片的大小要一致，且分片大小目前必须是16的整数倍。
- 调用客户端加密上传文件后，加密元数据会被保护，无法通过CopyObject修改Object meta信息。

加密方式

对于主密钥的使用，目前支持如下两种方式：

- 使用KMS托管用户主密钥

  当使用KMS托管用户主密钥用于客户端数据加密时，需要将KMS用户主密钥ID（即CMK ID）传递给SDK。

- 使用用户自主管理的主密钥（RSA）

  主密钥信息由用户提供，需要用户将主密钥的公钥、私钥信息当做参数传递给SDK。

使用以上两种加密方式能够有效地避免数据泄漏，保护客户端数据安全。即使数据泄漏，其他人也无法解密得到原始数据。

加密元信息

参数	描述	是否必需
x-oss-meta-client-side-encryption-key	加密后的密钥。 经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-start	随机产生的加密数据的初始值 。经过主密钥加密后再经过base64编码的字符串。	是
x-oss-meta-client-side-encryption-cek-alg	数据的加密算法。	是
x-oss-meta-client-side-encryption-wrap-alg	数据密钥的加密算法。	是
x-oss-meta-client-side-encryption-matdesc	主密钥的描述信息。JSON格式。 警告 强烈建议为每个主密钥都配置描述信息，并保存好主密钥和描述信息之间的对应关系。否则不支持更换主密钥进行加密。	否
x-oss-meta-client-side-encryption-unencrypted-content-length	加密前的数据长度。如未指定content-length则不生成该参数。	否

参数	描述	是否必需
x-oss-meta-client-side-encryption-unencrypted-content-md5	加密前数据的MD5。如未指定MD5，则不生成该参数。	否
x-oss-meta-client-side-encryption-data-size	若加密Multipart文件需要在init_multipart时传入整个大文件的总大小。	是（分片上传）
x-oss-meta-client-side-encryption-part-size	若加密Multipart文件需要在init_multipart时传入分片大小。 **说明** 目前分片大小必须是16的整数倍。	是（分片上传）

以下提供了如何使用用户自主管理的主密钥（RSA）方式从内存中上传文件、上传本地文件、断点续传上传、分片上传、下载到本地文件等场景的完整示例代码。

从内存中上传文件

以下代码用于使用主密钥RSA加密内存中上传的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 主密钥及描述信息 */
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "Thank you for using Aliyun Object Storage Service!";
  PutObjectRequest request(BucketName, ObjectName, content);
  /* 上传文件 */
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

上传本地文件

以下代码用于使用主密钥RSA加密本地上传的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 上传文件 */
  auto outcome = client.PutObject(BucketName, ObjectName, "yourLocalFilename");
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

断点续传上传

以下代码用于使用主密钥RSA加密断点续传上传的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string UploadFilePath = "yourUploadfilePath";
  std::string CheckpointFilePath = "yourCheckpointFilepath";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 断点续传上传 */
  UploadObjectRequest request(BucketName, ObjectName, UploadFilePath, CheckpointFilePath);
  auto outcome = client.ResumableUploadObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "ResumableUploadObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

分片上传

以下代码用于使用主密钥RSA加密分片上传的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
#include <fstream>
using namespace AlibabaCloud::OSS;
static int64_t getFileSize(const std::string& file)
{
  std::fstream f(file, std::ios::in | std::ios::binary);
  f.seekg(0, f.end);
  int64_t size = f.tellg();
  f.close();
  return size;
```

```
  return size;
}
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string fileToUpload = "yourLocalFilename";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 初始化分片加密上下文 */
  /* 需要16字节对齐 */
  int64_t partSize = 100 * 1024;
  auto fileSize = getFileSize(fileToUpload);
  MultipartUploadCryptoContext cryptoCtx;
  cryptoCtx.setPartSize(partSize);
  cryptoCtx.setDataSize(fileSize);
  /* 初始化分片上传事件 */
  InitiateMultipartUploadRequest initUploadRequest(BucketName, ObjectName);
  auto uploadIdResult = client.InitiateMultipartUpload(initUploadRequest, cryptoCtx);
  auto uploadId = uploadIdResult.result().UploadId();
  PartList partETagList;
  int partCount = static_cast<int> (fileSize / partSize);
  /* 计算分片个数 */
  if (fileSize % partSize != 0)  {
    partCount++;
  }
  /* 对每一个分片进行上传 */
  for (int i = 1; i <= partCount; i++) {
    auto skipBytes = partSize * (i - 1);
    auto size = (partSize < fileSize - skipBytes) ? partSize : (fileSize - skipBytes);
    std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>(fileToUpload, std::ios::in|std::
ios::binary);
    content->seekg(skipBytes, std::ios::beg);
    UploadPartRequest uploadPartRequest(BucketName, ObjectName, content);
    uploadPartRequest.setContentLength(size);
    uploadPartRequest.setUploadId(uploadId);
    uploadPartRequest.setPartNumber(i);
    auto uploadPartOutcome = client.UploadPart(uploadPartRequest, cryptoCtx);
    if (uploadPartOutcome.isSuccess()) {
      Part part(i, uploadPartOutcome.result().ETag());
      partETagList.push_back(part);
    }
```

```
    else {
      std::cout << "uploadPart fail" <<
      ",code:" << uploadPartOutcome.error().Code() <<
      ",message:" << uploadPartOutcome.error().Message() <<
      ",requestId:" << uploadPartOutcome.error().RequestId() << std::endl;
    }
  }
  /* 完成分片上传 */
  CompleteMultipartUploadRequest request(BucketName, ObjectName);
  request.setUploadId(uploadId);
  request.setPartList(partETagList);
  auto outcome = client.CompleteMultipartUpload(request, cryptoCtx);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "CompleteMultipartUpload fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

下载到本地文件

以下代码用于使用主密钥RSA解密下载到本地的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
#include <fstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string FileNametoSave = "yourFileName";
  /* 主密钥及描述信息 */
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  /* 如果需要解密不同主密钥加密的内容, 需要传对应的密钥信息 */
  //std::string RSAPublicKey2 =  "your rsa public key";
  //std::string RSAPrivateKey2 = "your rsa private key";
  //std::map<std::string, std::string> desc2;
  //desc2["comment"] = "your comment";
  //materials.addEncryptionMaterial(RSAPublicKey2, RSAPrivateKey2, desc2);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 获取文件到本地文件 */
  GetObjectRequest request(BucketName, ObjectName);
  request.setResponseStreamFactory([=]() {return std::make_shared<std::fstream>(FileNametoSave, std::io
s_base::out | std::ios_base::in | std::ios_base::trunc| std::ios_base::binary); });
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
    std::cout << "GetObjectToFile success" << outcome.result().Metadata().ContentLength() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GetObjectToFile fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

下载到本地内存

以下代码用于使用主密钥RSA解密下载到本地内存的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  /* 如果需要解密不同主密钥加密的内容, 需要传对应的密钥信息 */
  //std::string RSAPublicKey2 =  "your rsa public key";
  //std::string RSAPrivateKey2 = "your rsa private key";
  //std::map<std::string, std::string> desc2;
  //desc2["comment"] = "your comment";
  //materials.addEncryptionMaterial(RSAPublicKey2, RSAPrivateKey2, desc2);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 获取文件到本地内存 */
  GetObjectRequest request(BucketName, ObjectName);
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
   std::cout << "getObjectToBuffer" << " success, Content-Length:" << outcome.result().Metadata().Content
Length() << std::endl;
    /* 打印下载内容 */
    std::string content;
    *(outcome.result().Content()) >> content;
    std::cout << "getObjectToBuffer" << "content:" << content << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "getObjectToBuffer fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

范围下载

以下代码用于使用主密钥RSA解密范围下载的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObectName ";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  /* 如果需要解密不同主密钥加密的内容, 需要传对应的密钥信息 */
  //std::string RSAPublicKey2 =  "your rsa public key";
  //std::string RSAPrivateKey2 = "your rsa private key";
  //std::map<std::string, std::string> desc2;
  //desc2["comment"] = "your comment";
  //materials.addEncryptionMaterial(RSAPublicKey2, RSAPrivateKey2, desc2);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 设置下载范围 */
  GetObjectRequest request(BucketName,  ObjectName);
  request.setRange(0, 1);
  auto outcome = client.GetObject(request);
  if (!outcome.isSuccess ()) {
    /* 异常处理 */
    std::cout << "getObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
     return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

断点续传下载

以下代码用于使用主密钥RSA解密断点续传下载的文件：

```
#include <alibabacloud/oss/OssEncryptionClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  std::string DownloadFilePath = "yourDownloadFilePath";
  std::string CheckpointFilePath = "yourCheckpointFilepath";
  /* 主密钥及描述信息*/
  std::string RSAPublicKey = "your rsa public key";
  std::string RSAPrivateKey = "your rsa private key";
  std::map<std::string, std::string> desc;
  desc["comment"] = "your comment";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  CryptoConfiguration cryptoConf;
  auto materials = std::make_shared<SimpleRSAEncryptionMaterials>(RSAPublicKey, RSAPrivateKey, desc);
  /* 如果需要解密不同主密钥加密的内容, 需要传对应的密钥信息 */
  //std::string RSAPublicKey2 =  "your rsa public key";
  //std::string RSAPrivateKey2 = "your rsa private key";
  //std::map<std::string, std::string> desc2;
  //desc2["comment"] = "your comment";
  //materials.addEncryptionMaterial(RSAPublicKey2, RSAPrivateKey2, desc2);
  OssEncryptionClient client(Endpoint, AccessKeyId, AccessKeySecret, conf, materials, cryptoConf);
  /* 断点续传下载 */
  DownloadObjectRequest request(BucketName, ObjectName, DownloadFilePath, CheckpointFilePath);
  auto outcome = client.ResumableDownloadObject(request);
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "ResumableDownloadObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.12. 单链接限速

本文介绍如何在上传、下载文件时，通过在请求中携带x-oss-traffic-limit参数并设置限速值，以保证其他应用的正常带宽。

说明 有关单链接限速的使用场景及注意事项的更多信息，请参考开发指南的单链接限速文档。

普通上传下载限速

以下代码用于普通上传、下载文件时设置单链接限速：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 上传文件 */
  std::shared_ptr<std::iostream> content = std::make_shared<std::fstream>("yourLocalFilename", std::ios::i
n|std::ios::binary);
  PutObjectRequest putrequest(BucketName, ObjectName,content);
  /* 设置上传流量速度限制为100KB/s */
  getrequest.setTrafficLimit(819200);
  auto putoutcome = client.PutObject(putrequest);
  /* 获取文件到本地内存 */
  GetObjectRequest getrequest(BucketName, ObjectName);
  /* 设置下载流量速度限制为100KB/s */
  getrequest.setTrafficLimit(819200);
  auto getoutcome = client.GetObject(getrequest);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

使用签名URL方式上传下载限速

以下代码用于使用签名URL方式上传、下载文件时设置单链接限速：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string PutobjectUrlName = "yourPutobjectUrlName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置签名URL有效时间 */
  std::time_t t = std::time(nullptr) + 1200;
  /* 生成上传文件URL */
  GeneratePresignedUrlRequest putrequest(BucketName, PutobjectUrlName, Http::Put);
  putrequest.setExpires(expires);
  /* 设置上传流量速度限制为100KB/s */
  putrequest.setTrafficLimit(819200);
  auto genOutcome = client.GeneratePresignedUrl(putrequest);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  /* 使用签名URL上传文件 */
  auto outcome = client.PutObjectByUrl(genOutcome.result(), content);
  /* 生成下载文件URL */
  GeneratePresignedUrlRequest getrequest(BucketName, PutobjectUrlName, Http::Get);
  getrequest.setExpires(expires);
  /* 设置下载流量速度限制为100KB/s */
  getrequest.setTrafficLimit(819200);
  genOutcome = client.GeneratePresignedUrl(getrequest);
  /* 使用签名URL下载文件 */
  outcome = client.GetObjectByUrl(genOutcome.result());
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.13. 数据安全性

OSS提供基于MD5和CRC64的数据校验，确保上传、下载和拷贝文件过程中的数据完整性。

MD5校验

如果上传文件时设置了Content-MD5，OSS会根据接收的内容计算MD5。OSS计算的MD5值和上传提供的MD5值不一致时，则返回InvalidDigest异常，从而保证数据的完整性。返回InvalidDigest异常后，您需要重新上传文件。

以下代码用于上传文件时进行MD5校验：

```
#include <alibabacloud/oss/OssClient.h>
#include "../src/utils/Utils.h"
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
  *content << "test cpp sdk";
  PutObjectRequest request(BucketName, ObjectName, content);
  /*设置MD5校验*/
  std::string contentMd5 = ComputeContentMD5(*content);
  request.setContentMd5(contentMd5);
  /*上传文件*/
  auto outcome = client.PutObject(request);
  if (!outcome.isSuccess()) {
    /*异常处理*/
    std::cout << "PutObject fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

说明 putObject、getObject、appendObject、postObject、uploadPart支持MD5校验。

CRC64校验

上传、下载和拷贝文件时默认开启CRC数据校验，确保数据的完整性。

说明

- putObject、getObject、appendObject、uploadPart支持CRC64校验。上传时默认开启CRC校验，如果客户端计算的CRC值与服务端返回的CRC值不一致， 则会返回错误。
- 范围下载不支持CRC64校验。
- CRC64校验会占用一定的CPU，对上传、下载速度均会有影响。

以下代码用于下载文件时进行CRC64数据完整性校验：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /*初始化OSS账号信息*/
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string ObjectName = "yourObjectName";
  /*初始化网络等资源*/
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /*获取文件到本地内存*/
  GetObjectRequest request(BucketName, ObjectName);
  /*C++ SDK默认开启CRC64校验*/
  /*CRC64校验失败会报错Crc64 validation failed。*/
  auto outcome = client.GetObject(request);
  if (outcome.isSuccess()) {
    std::cout << "getObjectToBuffer" << "server crc:" << outcome.result().Metadata().HttpMetaData().at("x-o
ss-hash-crc64ecma-by-client") << std::endl;
  }
  else {
    /*异常处理*/
    std::cout << "getObjectToBuffer fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /*释放网络等资源*/
  ShutdownSdk();
  return 0;
}
```

11.14. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

以下代码用于使用STS凭证构造签名请求。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string SecurityToken  = "securityToken";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, SecurityToken, conf);
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

说明 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

使用签名URL临时授权

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

在URL中加入签名信息，以便将该URL转给第三方实现授权访问。具体操作，请参见在URL中包含签名。

- 使用签名URL上传文件

  以下代码用于使用签名URL上传文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息 */
   std::string AccessKeyId = "yourAccessKeyId";
   std::string AccessKeySecret = "yourAccessKeySecret";
   std::string Endpoint = "yourEndpoint";
   std::string BucketName = "yourBucketName";
   std::string PutobjectUrlName = "yourPutobjectUrlName";
    /* 初始化网络等资源 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 设置签名URL有效时长 */
   std::time_t t = std::time(nullptr) + 1200;
   /* 生成签名URL */
   auto genOutcome = client.GeneratePresignedUrl(BucketName, PutobjectUrlName, t, Http::Put);
   if (genOutcome.isSuccess()) {
      std::cout << "GeneratePresignedUrl success, Gen url:" << genOutcome.result().c_str() << std::endl;
   }
   else {
      /* 异常处理 */
      std::cout << "GeneratePresignedUrl fail" <<
      ",code:" << genOutcome.error().Code() <<
      ",message:" << genOutcome.error().Message() <<
      ",requestId:" << genOutcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   std::shared_ptr<std::iostream> content = std::make_shared<std::stringstream>();
   *content << "test cpp sdk";
   /* 使用签名URL上传文件 */
   auto outcome = client.PutObjectByUrl(genOutcome.result(), content);
   if (!outcome.isSuccess()) {
      /* 异常处理 */
      std::cout << "PutObjectByUrl fail" <<
      ",code:" << outcome.error().Code() <<
      ",message:" << outcome.error().Message() <<
      ",requestId:" << outcome.error().RequestId() << std::endl;
      ShutdownSdk();
      return -1;
   }
   /* 释放网络等资源 */
   ShutdownSdk();
   return 0;
}
```

- 使用签名URL下载文件

  以下代码用于使用签名URL下载文件：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
  /* 初始化OSS账号信息 */
  std::string AccessKeyId = "yourAccessKeyId";
  std::string AccessKeySecret = "yourAccessKeySecret";
  std::string Endpoint = "yourEndpoint";
  std::string BucketName = "yourBucketName";
  std::string GetobjectUrlName = "yourGetobjectUrlName";
  /* 初始化网络等资源 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 设置签名URL有效时长*/
  std::time_t t = std::time(nullptr) + 1200;
  /* 生成签名URL */
  auto genOutcome = client.GeneratePresignedUrl(BucketName, GetobjectUrlName, t, Http::Get);
  if (genOutcome.isSuccess()) {
    std::cout << "GeneratePresignedUrl success, Gen url:" << genOutcome.result().c_str() << std::endl;
  }
  else {
    /* 异常处理 */
    std::cout << "GeneratePresignedUrl fail" <<
    ",code:" << genOutcome.error().Code() <<
    ",message:" << genOutcome.error().Message() <<
    ",requestId:" << genOutcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 使用签名URL下载文件 */
  auto outcome = client.GetObjectByUrl(genOutcome.result());
  if (!outcome.isSuccess()) {
    /* 异常处理 */
    std::cout << "GetObjectByUrl fail" <<
    ",code:" << outcome.error().Code() <<
    ",message:" << outcome.error().Message() <<
    ",requestId:" << outcome.error().RequestId() << std::endl;
    ShutdownSdk();
    return -1;
  }
  /* 释放网络等资源 */
  ShutdownSdk();
  return 0;
}
```

11.15. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息。 */
    std::string AccessKeyId = "<yourAccessKeyId>";
    std::string AccessKeySecret = "<yourAccessKeySecret>";
    std::string Endpoint = "<yourEndpoint>";
    std::string BucketName = "<yourBucketName>";
    std::string ObjectName = "<yourObjectName>";
    /* 初始化网络等资源。 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /* 将图片缩放为固定宽高100 px后保存在本地。 */
    std::string Process = "image/resize,m_fixed,w_100,h_100";
    GetObjectRequest request(BucketName, ObjectName);
    request.setProcess(Process);
    auto outcome = client.GetObject(request);
    /* 释放网络等资源。 */
    ShutdownSdk();
    return 0;
}
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息。 */
    std::string AccessKeyId = "<yourAccessKeyId>";
    std::string AccessKeySecret = "<yourAccessKeySecret>";
    std::string Endpoint = "<yourEndpoint>";
    std::string BucketName = "<yourBucketName>";
    std::string ObjectName = "<yourObjectName>";
    /* 初始化网络等资源。 */
    InitializeSdk();
    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
    /* 将图片缩放为固定宽高100 px后，再旋转90°，之后保存在本地。 */
    std::string Process = "image/resize,m_fixed,w_100,h_100/rotate,90";
    GetObjectRequest request(BucketName, ObjectName);
    request.setProcess(Process);
    auto outcome = client.GetObject(request);
    /* 释放网络等资源。 */
    ShutdownSdk();
    return 0;
}
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息。 */
   std::string AccessKeyId = "<yourAccessKeyId>";
   std::string AccessKeySecret = "<yourAccessKeySecret>";
   std::string Endpoint = "<yourEndpoint>";
   std::string BucketName = "<yourBucketName>";
   std::string ObjectName = "<yourObjectName>";
    /* 初始化网络等资源。 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf) ;
   /* 使用指定图片样式处理图片后保存在本地。 */
   std::string Process = "style/<yourCustomStyleName>";
   GetObjectRequest request(BucketName, ObjectName);
   request.setProcess(Process);
   auto outcome = client.GetObject(request);
   /* 释放网络等资源。 */
   ShutdownSdk();
   return 0;
}
```

图片处理持久化

您可以通过ImgSaveAs接口将图片保存至当前存储空间。以下代码展示了图片处理持久化操作：

```
#include <alibabacloud/oss/OssClient.h>
#include <sstream>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息。 */
   std::string AccessKeyId = "<yourAccessKeyId>";
   std::string AccessKeySecret = "<yourAccessKeySecret>";
   std::string Endpoint = "<yourEndpoint>";
   std::string BucketName = "<yourBucketName>";
    /* 原图名称。若图片不在Bucket根目录，需携带文件访问路径，例如example/example.jpg。 */
   std::string SourceObjectName = "<yourSourceObjectName>";
    /* 指定处理后的图片名称。 */
   std::string TargetObjectName = "<yourTargetObjectName>";
    /* 初始化网络等资源。 */
   InitializeSdk();
   ClientConfiguration conf;
   OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
   /* 将图片缩放为固定宽高100 px后转存到当前存储空间。 */
   std::string Process = "image/resize,m_fixed,w_100,h_100";
   std::stringstream ss;
   ss  << Process
   <<"|sys/saveas"
   << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
   << ",b_" << Base64EncodeUrlSafe(BucketName);
   ProcessObjectRequest request(BucketName, SourceObjectName, ss.str());
   auto outcome = client.ProcessObject(request);
   /* 释放网络等资源。 */
   ShutdownSdk();
   return 0;
}
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
#include <alibabacloud/oss/OssClient.h>
using namespace AlibabaCloud::OSS;
int main(void)
{
    /* 初始化OSS账号信息。 */
  std::string AccessKeyId = "<yourAccessKeyId>";
  std::string AccessKeySecret = "<yourAccessKeySecret>";
  std::string Endpoint = "<yourEndpoint>";
  std::string BucketName = "<yourBucketName>";
  std::string ObjectName = "<yourObjectName>";
   /* 初始化网络等资源。 */
  InitializeSdk();
  ClientConfiguration conf;
  OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);
  /* 生成带图片处理参数的文件签名URL。 */
  std::string Process = "image/resize,m_fixed,w_100,h_100";
  GeneratePresignedUrlRequest request(BucketName, ObjectName, Http::Get);
  request.setProcess(Process);
  auto outcome = client.GeneratePresignedUrl(request);
  /* 释放网络等资源。 */
  ShutdownSdk();
  return 0;
}
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

12.C

12.1. 前言

本文档基于OSS C SDK 3.9.2编写。

兼容性

OSS C SDK版本兼容性说明如下：

- 对于3.X.X系列SDK：兼容。
- 对于 2.X.X系列SDK：
  - 兼容Windows。
  - 兼容Linux，链表（aos_list_t）遍历接口不兼容。
    - aos_list_for_each_entry
    - aos_list_for_each_entry_reverse
    - aos_list_for_each_entry_safe
    - aos_list_for_each_entry_safe_reverse
- 对于 1.0.0 系列SDK，除以下结构体和接口不兼容外，其余均兼容。
  - oss_request_options_t
  - oss_get_object_to_buffer
  - oss_get_object_to_file
  - oss_get_object_to_buffer_by_url
  - oss_get_object_to_file_by_url
  - oss_init_multipart_upload
  - oss_complete_multipart_upload
- 对于 0.0.X系列SDK：不兼容。

示例代码

OSS C SDK提供丰富的示例代码，方便您参考或直接使用。示例代码包括以下内容：

示例文件	示例内容
oss_put_object_sample	上传文件
oss_get_object_sample.c	下载文件
oss_append_object_sample.c	追加上传
oss_multipart_upload_sample.c	分片上传
oss_resumable_sample.c	断点续传上传、断点续传下载
oss_head_object_sample.c	管理文件元信息
oss_list_object_sample.c	列举文件

示例文件	示例内容
oss_delete_object_sample.c	删除文件
oss_callback_sample.c	上传回调
oss_progress_sample.c	进度条上传、进度条下载
oss_crc_sample.c	上传、下载时进行CRC校验
oss_image_sample.c	图片处理

说明 C SDK源码请参见GitHub。

12.2. 安装

本文介绍如何安装OSS C SDK。

Linux环境下的安装

- 安装CMake和第三方库

  OSS C SDK安装时，需要安装编译工具CMake和第三方库curl、apr、apr-util、minixml。

  安装环境时所需参数如下：

名称	描述	版本要求
CMake	编译安装工具。	2.6.0及以上版本
curl	主要解决网络方面的问题。	7.32.0 及以上版本
apr-util	解决内存管理以及跨平台问题。	1.5.2 及以上版本
minixml	解析请求返回的xml。	推荐使用 v2.9 版本

  请选择对应的系统安装。

  - Ubuntu/Debian
    - 安装CMake

```
sudo apt-get install cmake
```

    - 安装第三方库

```
sudo apt-get install libcurl4-openssl-dev libapr1-dev libaprutil1-dev libmxml-dev
```

- RedHat/Aliyun/CentOS
  - 安装CMake

    ```
    sudo yum install cmake
    ```

  - 安装第三方库

    ```
    sudo yum install curl-devel apr-devel apr-util-devel  mxml mxml-devel
    ```

    如果yum源中没有mxml安装包，可采用rpm安装。请下载系统对应的rpm包：

    - 64位：mxml-2.9-1.x86_64.rpm
    - 32位：mxml-2.9-1.i386.rpm

    rpm包安装mxml：

    ```
    rpm -ivh mxml-2.9-1.x86_64.rpm
    ```

- SuSE
  - 安装CMake

    ```
    zypper install cmake
    ```

  - 安装第三方库

    ```
    zypper install libcurl-devel libapr1-devel libapr-util1-devel mxml-devel
    ```

- 其它Linux
  - 安装CMake：下载地址

    常用的安装方式如下：

    ```
    ./configure
    make
    make install
    ```

    说明 执行./configure时，默认安装路径为/usr/local/，如果需要指定安装路径，请使用./configure --prefix选项。

  - 安装libcurl：下载地址

    常用的安装方式如下：

    ```
    ./configure
    make
    make install
    ```

  - 安装apr：下载地址

    常用的安装方式如下：

    ```
    ./configure
    make
    make install
    ```

  - 安装apr-util：下载地址

    常用的安装方式如下：

    ```
    // 安装时需要指定--with-apr选项。
    ./configure --with-apr=/your/apr/install/path
    make
    make install
    ```

  - 安装minixml：下载地址

    说明 请使用 2.x 版本的 minixml。

    常用的安装方式如下：

    ```
    ./configure
    make
    sudo make install
    ```

- 下载SDK
  - 直接下载
  - 通过GitHub下载

- 历史版本下载

- 安装SDK
  - 如果 curl、apr、apr-util 和 mxml 第三方库安装在默认路径下时，安装方式如下：

```
cmake .
make
make install
```

  - 如果 curl、apr、apr-util 和 mxml 第三方库不是安装在默认路径下，安装 SDK时，需要指定其安装路径，安装方式如下：

```
cmake -f CMakeLists.txt
// 编译类型为Release。常用的编译类型为：Debug、Release、RelWithDebInfo和MinSizeRel，默认使用Debu
g。
-DCMAKE_BUILD_TYPE=Release
// 自定义安装目录。
-DCMAKE_INSTALL_PREFIX=/usr/local/
// 指定curl、apr、apr-util和xml第三方库头文件和库文件的所在目录。
-DCURL_INCLUDE_DIR=/usr/include/curl
-DCURL_LIBRARY=/usr/lib64/libcurl.so
-DAPR_INCLUDE_DIR=/usr/include/apr-1
-DAPR_LIBRARY=/usr/lib64/libapr-1.so
-DAPR_UTIL_INCLUDE_DIR=/usr/include/apr-1
-DAPR_UTIL_LIBRARY=/usr/lib64/libaprutil-1.so
-DMINIXML_INCLUDE_DIR=/usr/include
-DMINIXML_LIBRARY=/usr/lib64/libmxml.so
// 编译时，如果报`Could not find apr-config/apr-1-config`，原因是在默认路径里面找不到apr-1-config文件
，请添加该选项。
-DAPR_CONFIG_BIN=/path/to/bin/apr-1-config
// 如果报：Could not find apu-config/apu-1-config，请添加该选项。
-DAPU_CONFIG_BIN=/path/to/bin/apu-1-config
```

Windows环境下的安装

- 下载SDK
  - 直接下载
  - 通过GitHub下载
  - 历史版本下载

- 安装SDK

  使用Visual Studio编译OSS C SDK的详细步骤及常见问题，请参见Windows下编译使用Aliyun OSS C SDK。

> **说明** 如果您使用Visual Studio 2012及其以后版本打开时，会提示是否将项目升级成最新版的编译器和库，这里最好和您自己的项目保持一致。如果项目使用了最新版本的编译器和库，就选择升级，否则可以不升级。

12.3. 初始化

使用OSS C SDK时，需要初始化请求选项（oss_request_options_t），并指定Endpoint。

有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

使用OSS域名初始化请求选项

以下代码用于OSS域名初始化请求选项：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，其中这个函数的第二个参数表示ctl的归属，默认为0。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main() {
  aos_pool_t *p;
  oss_request_options_t *options;
  /* 初始化全局变量。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    return -1;
  }
  /* 初始化内存池和options。*/
  aos_pool_create(&p, NULL);
  options = oss_request_options_create(p);
  init_options(options);
  /* 逻辑代码，此处省略。*/
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(p);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

> **说明** 关于请求选项的设置，请参见curl_easy_setopt。

使用自定义域名初始化请求选项

以下代码用于自定义域名初始化请求选项：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourCustomEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
void init_options(oss_request_options_t *options) {
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 开启CNAME，将自定义域名绑定到存储空间上。*/
  options->config->is_cname = 1;
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main() {
  aos_pool_t *p;
  oss_request_options_t *options;
  /* 初始化全局变量。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    return -1;
  }
  /* 初始化内存池和options。*/
  aos_pool_create(&p, NULL);
  options = oss_request_options_create(p);
  init_options(options);
  /* 逻辑代码，此处省略。*/
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(p);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

使用STS初始化请求选项

以下代码用于STS初始化请求选项：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *sts_token = "<yourSecurityToken>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 设置STS。*/
  aos_str_set(&options->config->sts_token, sts_token);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main() {
  aos_pool_t *p;
  oss_request_options_t *options;
  /* 初始化全局变量。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    return -1;
  }
  /* 初始化内存池和options。*/
  aos_pool_create(&p, NULL);
  options = oss_request_options_create(p);
  init_options(options);
  /* 逻辑代码，此处省略。*/
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(p);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

设置超时时间

以下代码用于设置超时时间：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，其中这个函数的第二个参数表示ctl的归属，默认为0。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
  /* 设置链接超时，默认为10秒。*/
  options->ctl->options->connect_timeout = 10;
  /* 设置DNS超时，默认为60秒。*/
  options->ctl->options->dns_cache_timeout = 60;
  /*
  设置请求超时。
  通过speed_limit设置可接受的最小速率，默认为1024，即1 KB/s。
  通过speed_time设置可接受的最长时间，默认为15秒。
  如果传输速率连续15秒小于1 KB/s，则表明请求超时。
  */
  options->ctl->options->speed_limit = 1024;
  options->ctl->options->speed_time = 15;
}
```

设置SSL证书校验

C SDK 3.9.2及以上版本默认开启SSL证书校验。如果出现SSL证书校验失败，您需要设置正确的SSL证书路径，或者关闭SSL证书验证功能。

以下代码用于设置SSL证书校验：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，其中这个函数的第二个参数表示ctl的归属，默认为0。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
  /*设置SSL证书校验。
  通过verify_ssl设置是否开启SSL证书校验。可选值为1或0，默认值为1，即开启SSL证书校验。
  通过ca_path设置CA证书的根路径，当verify_ssl为1时有效，默认值为空。
  通过ca_file设置CA证书的路径，当verify_ssl为1时有效，默认值为空。*/
  //开启SSL证书校验，并设置CA证书路径。
  //options->ctl->options->verify_ssl = 1;
  //options->ctl->options->ca_path = "/etc/ssl/certs/";
  //options->ctl->options->ca_file = "/etc/ssl/certs/ca-certificates.crt";
  //关闭SSL证书校验。
  //options->ctl->options->verify_ssl = 0;
}
```

12.4. 快速入门

本节介绍如何快速使用OSS C SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

示例工程

- Linux示例工程

  基于Linux的示例工程 aliyun-oss-c-sdk-demo.tar.gz 。

  i. 下载并解压示例工程。

  ii. 安装oss-c-sdk-demo-specified-installation时需要指定安装目录**/home/your/oss/csdk**。其它示例工程基于OSS C SDK及依赖的第三方库默认安装，不需要指定安装目录。

  iii. 将示例代码中的OSS_ENDPOINT、ACCESS_KEY_ID、ACCESS_KEY_SECRET、BUCKET NAME更换成有效值。如果OSS C SDK及依赖库的动态库不在系统目录下，执行时请使用 `LD_LIBRARY_PATH` 指定。

  iv. 进入工程目录（oss-c-sdk-demo-xxx），执行 `make` ，编译示例工程；执行 `./main` 运行可执行程序；如需重新编译，请执行 `make clean` 再进行编译。

  示例工程安装详情请参见Linux下使用Aliyun OSS C SDK 。

- Windows示例工程

  基于Windows示例工程：aliyun-oss-c-sdk-sample.zip。更多示例工程下载请参见GitHub。

  i. 下载并解压示例工程。

ii. 使用Visual Studio直接打开对应的示例工程，将oss_config.c中的OSS_ENDPOINT、ACCESS_KEY_ID、ACCESS_KEY_SECRET、BUCKET_NAME、OBJECT_NAME、MULTIPART_UPLOAD_FILE_PATH和DIR_NAME更换成有效值，编译运行，也可以在示例工程基础上开发您的项目。

> **说明** 使用Visual Studio运行示例工程的详细步骤及常见问题，请参见Windows下编译使用Aliyun OSS C SDK。

创建存储空间

存储空间是OSS的全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于创建存储空间：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 创建存储空间。*/
  aos_status_t *resp_status;
  aos_table_t *resp_headers;
  oss_acl_e oss_acl = OSS_ACL_PRIVATE;
  aos_string_t bucket;
  /* 将类型为char*的数据赋值给bucket。*/
  aos_str_set(&bucket, bucket_name);
  resp_status = oss_create_bucket(oss_client_options, &bucket, oss_acl, &resp_headers);
```

```
  resp_status = oss_create_bucket(oss_client_options, &bucket, oss_acl, &resp_headers);
  /* 判断请求是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("create bucket succeeded\n");
  } else {
    printf("create bucket failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

存储空间的命名规范详情请参见基本概念中的命名规范。创建存储空间详情请参见创建存储空间。

获取endpoint详情请参见访问域名和数据中心文档。

上传文件

以下代码用于上传文件至OSS：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
```

```
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_status_t *resp_status;
  aos_table_t *headers;
  aos_table_t *resp_headers;
  aos_list_t buffer;
  aos_buf_t *content = NULL;
  const char *data = "More than just cloud.";
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  headers = aos_table_make(pool, 0);
  /* 将char*类型的数据转换为aos_list_t类型。*/
  aos_list_init(&buffer);
  content = aos_buf_pack(oss_client_options->pool, data, strlen(data));
  aos_list_add_tail(&content->node, &buffer);
  /* 上传文件。*/
  resp_status = oss_put_object_from_buffer(oss_client_options, &bucket, &object, &buffer, headers, &resp
_headers);
  /* 判断上传文件的请求是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("put file succeeded\n");
  } else {
    printf("put file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

下载文件

以下代码用于将指定的OSS文件下载到本地文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *download_filename = "<yourDownloadedFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
```

```
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_status_t *resp_status;
  aos_table_t *headers = NULL;
  aos_table_t *params = NULL;
  aos_table_t *resp_headers = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, download_filename);
  headers = aos_table_make(pool, 0);
  /* 下载文件。*/
  resp_status = oss_get_object_to_file(oss_client_options, &bucket, &object, headers, params, &file, &resp_
headers);
  /* 判断下载文件的请求是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("get object to local file succeeded\n");
  } else {
    printf("get object to local file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

列举文件

以下代码用于列举指定存储空间的文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool = NULL;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options = NULL;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  /* 列举文件。*/
  resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
  /* 判断列举文件的请求是否成功。*/
  if (!aos_status_is_ok(resp_status))
  {
    printf("list object failed\n");
  }
  else{
    /* 打印文件。*/
    printf("Object\tSize\tLastModified\n");
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
```

```
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    printf("Total %d\n", size);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

删除文件

以下代码用于删除指定文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool = NULL;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options = NULL;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
```

```
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_status_t *resp_status = NULL;
  aos_table_t *resp_headers = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 删除文件。*/
  resp_status = oss_delete_object(oss_client_options, &bucket, &object, &resp_headers);
  /* 判断删除文件的请求是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("delete object succeeded\n");
  } else {
    printf("delete object failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.5. 存储空间

12.5.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。 */
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。 */
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
```

```
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  oss_acl_e oss_acl = OSS_ACL_PRIVATE;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。 */
  aos_str_set(&bucket, bucket_name);
  /* 创建存储空间。*/
  resp_status = oss_create_bucket(oss_client_options, &bucket, oss_acl, &resp_headers);
  /* 判断请求是否成功。 */
  if (aos_status_is_ok(resp_status)) {
    printf("create bucket succeeded\n");
  } else {
    printf("create bucket failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

以下代码用于在创建存储空间时指定存储空间的访问权限：

```
  /* 创建存储空间，并设置存储空间的权限为公共读（默认是私有）。*/
  resp_status = oss_create_bucket(oss_client_options, &bucket, OSS_ACL_PUBLIC_READ, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("create bucket succeeded\n");
  } else {
    printf("create bucket failed\n");
  }
```

以下代码用于创建低频访问类型的存储空间：

如需创建归档存储类型的存储空间，请将如下示例中的OSS_STORAGE_CLASS_IA替换为OSS_STORAGE_CLASS_ARCHIVE。

```
  resp_status = oss_create_bucket_with_storage_class(oss_client_options, &bucket, OSS_ACL_PRIVATE, O
SS_STORAGE_CLASS_IA);
  /* 判断请求是否成功。 */
  if (aos_status_is_ok(resp_status)) {
    printf("create bucket succeeded\n");
  } else {
    printf("create bucket failed\n");
  }
```

创建存储空间的更多详情，请参见Put Bucket。

12.5.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

以下代码用于列举所有存储空间。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。 */
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。 */
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。 */
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_table_t *resp_headers = NULL;
```

```
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_list_buckets_params_t *params = NULL;
  oss_list_bucket_content_t *content = NULL;
  int size = 0;
  params = oss_create_list_buckets_params(pool);
  /* 列举存储空间。 */
  resp_status = oss_list_bucket(oss_client_options, params, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("list buckets succeeded\n");
  } else {
    printf("list buckets failed\n");
  }
  /* 打印存储空间。 */
  aos_list_for_each_entry(oss_list_bucket_content_t, content, &params->bucket_list, node) {
    printf("BucketName: %s\n", content->name.data);
    ++size;
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.3. 获取存储空间的地域

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间所在的地域。

以下代码用于获取存储空间所在的地域（称为Region或Location）：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
```

```
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t oss_location;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。 */
  aos_str_set(&bucket, bucket_name);
  /* 获取存储空间的地域。*/
  resp_status = oss_get_bucket_location(oss_client_options, &bucket, &oss_location, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket location succeeded : %s \n", oss_location.data);
  } else {
    printf("get bucket location failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.4. 获取存储空间的信息

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何获取存储空间的信息。

以下代码用于获取存储空间的信息（Info），包括存储空间所在地域、创建日期等。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
```

```
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  oss_bucket_info_t bucket_info;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。*/
  aos_str_set(&bucket, bucket_name);
  /* 获取存储空间的信息。*/
  resp_status = oss_get_bucket_info(oss_client_options, &bucket, &bucket_info, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket info succeeded\n");
    printf("location: %s\n", bucket_info.location.data);
    printf("owner_id: %s\n", bucket_info.owner_id.data);
    printf("owner_name: %s\n", bucket_info.owner_name.data);
    printf("created_date: %s\n", bucket_info.created_date.data);
    printf("location: %s\n", bucket_info.location.data);
  } else {
    printf("get bucket info failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

关于获取存储空间信息的更多信息，请参见GetBucketInfo。

12.5.5. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

设置存储空间访问权限

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	OSS_ACL_PRIVATE
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ_WRITE

以下代码用于设置存储空间的访问权限：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
```

```
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。*/
  aos_str_set(&bucket, bucket_name);
  /* 设置存储空间权限为公共读(OSS_ACL_PUBLIC_READ)。
  resp_status = oss_put_bucket_acl(oss_client_options, &bucket, OSS_ACL_PUBLIC_READ, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("set bucket acl succeeded\n");
  } else {
    printf("set bucket acl failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间的访问权限

以下代码用于获取存储空间的访问权限：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
```

```
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t oss_acl;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。 */
  aos_str_set(&bucket, bucket_name);
  /* 获取存储空间权限。*/
  resp_status = oss_get_bucket_acl(oss_client_options, &bucket, &oss_acl, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket acl succeeded : %s \n", oss_acl.data);
  } else {
    printf("get bucket acl failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

12.5.6. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
>
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用oss_list_upload_part列举出碎片，然后使用oss_abort_multipart_upload删除碎片。详情请参见分片上传。

以下代码用于删除存储空间：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
```

```
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的存储空间。*/
  aos_str_set(&bucket, bucket_name);
  /* 删除存储空间。 */
  resp_status = oss_delete_bucket (oss_client_options, &bucket, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("delete bucket succeeded\n");
  } else {
    printf("delete bucket failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

删除存储空间的更多详情，请参见DeleteBucket。

12.5.7. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

每条生命周期规则包含以下内容：

- 生命周期规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 生命周期策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
  - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
  - 配置到整个存储空间。此种方式只能创建一条规则。
- 指定文件过期时间。有两种指定方式：
  - 指定一个过期天数N，文件会在其最近更新时间点的N天后过期。
  - 指定一个过期时间点，最近更新时间在该时间点之前的文件全部过期。
- 生命周期规则是否生效。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用cname域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
```

```
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_list_t lifecycle_rule_list;
  /* 给存储空间创建生命周期规则。*/
  aos_str_set(&bucket, bucket_name);
  aos_list_init(&lifecycle_rule_list);
  /* 指定过期天数。*/
  oss_lifecycle_rule_content_t *rule_content_days = oss_create_lifecycle_rule_content(pool);
  aos_str_set(&rule_content_days->id, "rule-1");
  aos_str_set(&rule_content_days->prefix, "obsoleted");
  aos_str_set(&rule_content_days->status, "Enabled");
  rule_content_days->days = 3;
  aos_list_add_tail(&rule_content_days->node, &lifecycle_rule_list);
  /* 指定过期时间。*/
  oss_lifecycle_rule_content_t *rule_content_date = oss_create_lifecycle_rule_content(pool);
  aos_str_set(&rule_content_date->id, "rule-2");
  aos_str_set(&rule_content_date->prefix, "delete");
  aos_str_set(&rule_content_date->status, "Enabled");
  aos_str_set(&rule_content_date->date, "2022-10-11T00:00:00.000Z");
  aos_list_add_tail(&rule_content_date->node, &lifecycle_rule_list);
  /* 设置生命周期规则。*/
  resp_status = oss_put_bucket_lifecycle(oss_client_options, &bucket, &lifecycle_rule_list, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put bucket lifecycle succeeded\n");
  } else {
    printf("put bucket lifecycle failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
```

```
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用cname域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_list_t lifecycle_rule_list;
  oss_lifecycle_rule_content_t *rule_content;
  char *rule_id;
  char *prefix;
  char *status;
  int days = INT_MAX;
  char* date = "";
  aos_str_set(&bucket, bucket_name);
  /* 获取存储空间生命周期规则并打印。*/
  aos_list_init(&lifecycle_rule_list);
  resp_status = oss_get_bucket_lifecycle(oss_client_options, &bucket, &lifecycle_rule_list, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket lifecycle succeeded\n");
    aos_list_for_each_entry(oss_lifecycle_rule_content_t, rule_content, &lifecycle_rule_list, node) {
      rule_id = apr_psprintf(pool, "%.*s", rule_content->id.len, rule_content->id.data);
      prefix = apr_psprintf(pool, "%.*s", rule_content->prefix.len, rule_content->prefix.data);
      status = apr_psprintf(pool,"%.*s", rule_content->status.len, rule_content->status.data);
      date = apr_psprintf(pool, "%.*s", rule_content->date.len, rule_content->date.data);
      days = rule_content->days;
    }
    printf("rule_id: %s\n", rule_id);
```

```
    printf("rule_id: %s \n", rule_id);
    printf("prefix: %s \n", prefix);
    printf("status: %s \n", status);
    printf("date: %s \n", date);
    printf("days: %d \n", days);
  } else {
    printf("get bucket lifecycle failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用cname域名访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
```

```
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  /* 删除存储空间生命周期规则。*/
  aos_str_set(&bucket, bucket_name);
  resp_status = oss_delete_bucket_lifecycle(oss_client_options, &bucket, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("delete bucket lifecycle succeeded\n");
  } else {
    printf("delete bucket lifecycle failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.8. 跨域资源共享

本文介绍如何进行跨域资源共享。

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

设置跨域资源共享规则

以下代码用于设置指定存储空间的跨域资源共享规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
```

```
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_list_t cors_rule_list;
  oss_cors_rule_t *cors_rule1 = NULL, *cors_rule2 = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_list_init(&cors_rule_list);
  cors_rule1 = oss_create_cors_rule(pool);
  aos_list_add_tail(&cors_rule1->node, &cors_rule_list);
  oss_create_sub_cors_rule(pool, &cors_rule1->allowed_origin_list, "allowed_origin_1_1");
  oss_create_sub_cors_rule(pool, &cors_rule1->allowed_origin_list, "allowed_origin_1_1");
  oss_create_sub_cors_rule(pool, &cors_rule1->allowed_method_list, "PUT");
  oss_create_sub_cors_rule(pool, &cors_rule1->allowed_method_list, "GET");
  oss_create_sub_cors_rule(pool, &cors_rule1->allowed_head_list, "Authorization");
  oss_create_sub_cors_rule(pool, &cors_rule1->expose_head_list, "expose_head_1_1");
  oss_create_sub_cors_rule(pool, &cors_rule1->expose_head_list, "expose_head_1_1");
  cors_rule2 = oss_create_cors_rule(pool);
  aos_list_add_tail(&cors_rule2->node, &cors_rule_list);
  oss_create_sub_cors_rule(pool, &cors_rule2->allowed_origin_list, "allowed_origin_2_1");
  oss_create_sub_cors_rule(pool, &cors_rule2->allowed_origin_list, "allowed_origin_2_2");
  oss_create_sub_cors_rule(pool, &cors_rule2->allowed_method_list, "PUT");
  oss_create_sub_cors_rule(pool, &cors_rule2->allowed_method_list, "GET");
  oss_create_sub_cors_rule(pool, &cors_rule2->allowed_head_list, "Authorization");
  oss_create_sub_cors_rule(pool, &cors_rule2->expose_head_list, "expose_head_2_1");
  oss_create_sub_cors_rule(pool, &cors_rule2->expose_head_list, "expose_head_2_2");
  /* 设置跨域资源共享规则。*/
  resp_status = oss_put_bucket_cors(oss_client_options, &bucket, &cors_rule_list, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put bucket cors succeeded\n");
  } else {
    printf("put bucket cors failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

获取跨域资源共享规则

以下代码用于获取跨域资源共享规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_list_t cors_rule_list;
  oss_cors_rule_t *cors_rule = NULL;
  oss_sub_cors_rule_t *sub_cors_rule = NULL;
  aos_str_set(&bucket, bucket_name);
  /* 获取跨域资源共享规则。*/
  aos_list_init(&cors_rule_list);
  resp_status = oss_get_bucket_cors(oss_client_options, &bucket, &cors_rule_list, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket cors succeeded\n");
    aos_list_for_each_entry(oss_cors_rule_t, cors_rule, &cors_rule_list, node) {
```

```
        printf("max_age_seconds: %d\n", cors_rule->max_age_seconds);
        aos_list_for_each_entry(oss_sub_cors_rule_t, sub_cors_rule, &cors_rule->allowed_origin_list, node) {
          printf("allowed_origin_list: %s \n", sub_cors_rule->rule.data);
        }
        aos_list_for_each_entry(oss_sub_cors_rule_t, sub_cors_rule, &cors_rule->allowed_method_list, node
) {
          printf("allowed_method_list: %s \n", sub_cors_rule->rule.data);
        }
        aos_list_for_each_entry(oss_sub_cors_rule_t, sub_cors_rule, &cors_rule->allowed_head_list, node) {
          printf("allowed_head_list: %s \n", sub_cors_rule->rule.data);
        }
        aos_list_for_each_entry(oss_sub_cors_rule_t, sub_cors_rule, &cors_rule->expose_head_list, node) {
          printf("expose_head_list: %s \n", sub_cors_rule->rule.data);
        }
      }
    } else {
      printf("get bucket cors failed\n");
    }
    /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
    aos_pool_destroy(pool);
    /* 释放之前分配的全局资源。*/
    aos_http_io_deinitialize();
    return 0;
}
```

删除跨域资源共享规则

以下代码用于删除指定存储空间的所有跨域资源共享规则：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  /* 删除跨域资源共享规则。*/
  resp_status = oss_delete_bucket_cors(oss_client_options, &bucket, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket cors succeeded\n");
  } else {
    printf("get bucket cors failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.9. 访问日志

您可以开启存储空间的访问日志记录功能。开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

```
<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString
```

关于访问日志的更多信息，请参见开发指南中的日志转存。

开启访问日志记录

以下代码用于开启存储空间的访问日志记录：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *target_bucket_name = "<yourTargetBucketName>";
const char *target_logging_prefix = "<yourTargetPrefix>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool）， 等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数 */
  aos_string_t bucket;
  oss_logging_config_content_t *content;
```

```
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  content = oss_create_logging_rule_content(pool);
  aos_str_set(&content->target_bucket, target_bucket_name);
  aos_str_set(&content->prefix, target_logging_prefix);
  /* 开启存储空间的访问日志记录。 */
  resp_status = oss_put_bucket_logging(oss_client_options, &bucket, content, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put symlink succeeded\n");
  } else {
    printf("put symlink failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

查看访问日志设置

以下代码用于查看存储空间的访问日志设置：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool）， 等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配
```

```
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  oss_logging_config_content_t *content;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  content = oss_create_logging_rule_content(pool);
  /* 查看存储空间的访问日志设置。*/
  resp_status = oss_get_bucket_logging(oss_client_options, &bucket, content, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket logging succeeded\n");
  } else {
    printf("get bucket logging failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  printf("bucket:%s, prefix:%s\n", content->target_bucket.data, content->prefix.data);
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

关闭访问日志记录

以下代码用于关闭存储空间的访问日志记录：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
```

```
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  /* 关闭访问日志记录。*/
  resp_status = oss_delete_bucket_logging(oss_client_options, &bucket, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("delete bucket logging succeeded\n");
  } else {
    printf("delete bucket logging failed, code:%d, error_code:%s, error_msg:%s, request_id:%s\n",
      resp_status->code, resp_status->error_code, resp_status->error_msg, resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.10. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置防盗链

以下代码用于设置防盗链：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
```

```
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_referer_config_t referer_config;
  aos_str_set(&bucket, bucket_name);
  aos_list_init(&referer_config.referer_list);
  oss_create_and_add_refer(pool, &referer_config, "http://www.aliyun.com");
  oss_create_and_add_refer(pool, &referer_config, "https://www.aliyun.com");
  referer_config.allow_empty_referer = 0;
  /* 添加Referer白名单。Referer参数支持通配符星号（*）和问号（？）。*/
  resp_status = oss_put_bucket_referer(oss_client_options, &bucket, &referer_config, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put bucket referer succeeded\n");
  } else {
    printf("put bucket referer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

获取防盗链信息

以下代码用于获取防盗链信息：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_referer_config_t referer_config;
  oss_referer_t *referer;
  aos_str_set(&bucket, bucket_name);
  aos_list_init(&referer_config.referer_list);
  /* 获取Referer白名单列表。*/
  resp_status = oss_get_bucket_referer(oss_client_options, &bucket, &referer_config, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket referer succeeded\n");
    aos_list_for_each_entry(oss_referer_t, referer, &referer_config.referer_list, node) {
      printf("get referer %s\n", referer->referer.data);
    }
  } else {
```

```
  } else {
    printf("get bucket referer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

清空防盗链

以下代码用于清空防盗链：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_referer_config_t referer_config;
  aos_str_set(&bucket, bucket_name);
```

```
  aos_str_set(&bucket, bucket_name);
  aos_list_init(&referer_config.referer_list);
  referer_config.allow_empty_referer = 1;
  /* 防盗链不能直接清空，需要新建一个允许空Referer的规则来覆盖之前的规则。*/
  resp_status = oss_put_bucket_referer(oss_client_options, &bucket, &referer_config, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("delete bucket referer succeeded\n");
  } else {
    printf("delete bucket referer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.5.11. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

以下代码用于设置静态网站托管：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
```

```
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_website_config_t website_config;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&website_config.suffix_str, "index.html");
  aos_str_set(&website_config.key_str, "error.html");
  /* 设置静态网站托管：索引页面为index.html，错误页面为error.html。*/
  resp_status = oss_put_bucket_website(oss_client_options, &bucket, &website_config, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put bucket website succeeded\n");
  } else {
    printf("put bucket website failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

查看静态网站托管配置

以下代码用于查看静态网站托管配置：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
```

```
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_website_config_t website_config;
  aos_str_set(&bucket, bucket_name);
  /* 查看静态网站托管配置。*/
  resp_status = oss_get_bucket_website(oss_client_options, &bucket, &website_config, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get bucket website succeeded\n");
    printf("website_config: %s  %s \n", website_config.suffix_str.data, website_config.key_str.data);
  } else {
    printf("get bucket website failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

删除静态网站托管配置

以下代码用于删除静态网站托管配置：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  /* 删除静态网站托管配置。*/
  resp_status = oss_delete_bucket_website(oss_client_options, &bucket, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("delete bucket website succeeded\n");
  } else {
    printf("delete bucket website failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.6. 上传文件

12.6.1. 概述

在OSS中，操作的基本数据单元是文件（Object）。OSS C SDK提供了丰富的文件上传方式：

- 简单上传：包括流式上传和文件上传。最大不能超过5GB。
- 追加上传：最大不能超过5GB。
- 断点续传上传：支持并发、断点续传、自定义分片大小。大文件上传推荐使用断点续传。最大不能超过48.8TB。
- 分片上传：当文件较大时，可以使用分片上传，最大不能超过48.8TB。

> **说明** 各种上传方式的适用场景请参见开发指南中的上传文件。

上传过程中，您可以设置文件元信息，也可以通过进度条功能查看上传进度。上传完成后，您还可以进行上传回调。

12.6.2. 简单上传

本文介绍如何使用简单上传。

简单上传的完整代码请参见：Git Hub 。

从内存中上传数据

以下代码用于从内存中上传数据：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *object_content = "More than just cloud.";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
```

```
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  aos_buf_t *content = NULL;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_list_init(&buffer);
  content = aos_buf_pack(oss_client_options->pool, object_content, strlen(object_content));
  aos_list_add_tail(&content->node, &buffer);
  /* 上传文件。*/
  resp_status = oss_put_object_from_buffer(oss_client_options, &bucket, &object, &buffer, headers, &resp_headers);
  /* 判断上传是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("put object from buffer succeeded\n");
  } else {
    printf("put object from buffer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

上传本地文件

以下代码用于上传本地文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
```

```
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  /* 上传文件。*/
  resp_status = oss_put_object_from_file(oss_client_options, &bucket, &object, &file, headers, &resp_headers);
  /* 判断上传是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("put object from file succeeded\n");
  } else {
    printf("put object from file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

常见问题：无法上传名称包含中文字符的文件

以 Windows 系统为例，您可以通过将包含中文字符的文件名称进行 UTF-8 编码后再上传该文件，示例代码如下：

```
#include "oss_api.h"
#include "aos_http_io.h"
#include <wchar.h>
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename_ch = "c:\\测试.txt";
void init_options(oss_request_options_t *options)
{
    options->config = oss_config_create(options->pool);
    /* 用char*类型的字符串初始化aos_string_t类型。*/
    aos_str_set(&options->config->endpoint, endpoint);
    aos_str_set(&options->config->access_key_id, access_key_id);
    aos_str_set(&options->config->access_key_secret, access_key_secret);
    /* 是否使用了CNAME。0表示不使用。*/
    options->config->is_cname = 0;
    /* 设置网络相关参数，比如超时时间等。*/
    options->ctl = aos_http_controller_create(options->pool, 0);
}
char* multibyte_to_utf8(const char * ch, char *str, int size)
{
    if (!ch || !str || size <= 0)
     return NULL;
    int chlen = strlen(ch);
    int multi_byte_cnt = 0;
    for (int i = 0; i < chlen - 1; i++) {
  if ((ch[i] & 0x80) && (ch[i + 1] & 0x80)) {
      i++;
      multi_byte_cnt++;
  }
    }
    if (multi_byte_cnt == 0)
  return ch;
    int len = MultiByteToWideChar(CP_ACP, 0, ch, -1, NULL, 0);
    if (len <= 0)
  return NULL;
    wchar_t *wch = malloc(sizeof(wchar_t) * (len + 1));
    if (!wch)
  return NULL;
    wmemset(wch, 0, len + 1);
    MultiByteToWideChar(CP_ACP, 0, ch, -1, wch, len);
    len = WideCharToMultiByte(CP_UTF8, 0, wch, -1, NULL, 0, NULL, NULL);
    if ((len <= 0) || (size < len + 1)) {
  free(wch);
  return NULL;
    }
    memset(str, 0, len + 1);
    WideCharToMultiByte(CP_UTF8, 0, wch, -1, str, len, NULL, NULL);
```

```
    WideCharToMultiByte(CP_UTF8, 0, wch, -1, str, len, NULL, NULL);
    free(wch);
    return str;
}
int main(int argc, char argv[])
{
    /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
    if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
     exit(1);
    }
    /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
    aos_pool_t pool;
    /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
    aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
    oss_request_options_t oss_client_options;
    /* 在内存池中分配内存给options。*/
    oss_client_options = oss_request_options_create(pool);
    /* 初始化Client的选项oss_client_options。*/
    init_options(oss_client_options);
    /* 初始化参数。*/
    aos_string_t bucket;
    aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 将多字节中文名字进行UTF8编码。*/
  char local_filename_buf[1024];
    char * local_filename = multibyte_to_utf8(local_filename_ch, local_filename_buf, 1024);
    aos_str_set(&file, local_filename);
    /* 上传文件。*/
  resp_status = oss_put_object_from_file(oss_client_options, &bucket, &object, &file, headers, &resp_headers);
    /* 判断上传是否成功。*/
    if (aos_status_is_ok(resp_status)) {
   printf("put object from file succeeded\n");
  } else {
   printf("put object from file failed\n");
    }
    /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
    aos_pool_destroy(pool);
    /* 释放之前分配的全局资源。*/
    aos_http_io_deinitialize();
    return 0;
}
```

12.6.3. 追加上传

追加上传是指通过AppendObject方法在已上传的追加类型文件（Appendable Object）末尾直接追加内容。

使用限制

通过追加上传的方式上传文件时，有如下限制：

- 如果文件不存在，则调用AppendObject接口会创建一个可追加类型的文件。
- 如果文件已存在，且为可追加类型文件时，若设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。如果文件已存在，且为不可追加类型文件时，则抛出ObjectNotAppendable异常。
- 追加类型的文件暂不支持CopyObject操作。

有关追加上传的更多信息，请参见AppendObject。

追加上传内存数据

通过追加上传的方式将内存数据上传至指定Bucket的示例代码如下：

```
#include "oss_api.h"
#include "aos_http_io.h"
/* Endpoint以杭州为例，其它Region请按实际情况填写。*/
const char *endpoint = "<yourEndpoint>";
/* 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。*/
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
/* 设置Bucket名称。*/
const char *bucket_name = "<yourBucketName>";
/* 设置Object名称。即不包含Bucket名称在内的Object的完整路径，例如example/folder/abc.jpg。*/
const char *object_name = "<yourObjectName>";
const char *object_content = "More than just cloud.";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
```

```
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  int64_t position = 0;
  char *next_append_position = NULL;
  aos_buf_t *content = NULL;
  aos_table_t *headers1 = NULL;
  aos_table_t *headers2 = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  headers1 = aos_table_make(pool, 0);
  /* 获取起始追加位置。*/
  resp_status = oss_head_object(oss_client_options, &bucket, &object, headers1, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    next_append_position = (char*)(apr_table_get(resp_headers, "x-oss-next-append-position"));
    position = atoi(next_append_position);
  }
  /* 追加文件。*/
  headers2 = aos_table_make(pool, 0);
  aos_list_init(&buffer);
  content = aos_buf_pack(pool, object_content, strlen(object_content));
  aos_list_add_tail(&content->node, &buffer);
  resp_status = oss_append_object_from_buffer(oss_client_options, &bucket, &object, position, &buffer, h
eaders2, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("append object from buffer succeeded\n");
  } else {
    printf("append object from buffer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

有关Bucket名称命名规范的更多信息，请参见存储空间（Bucket）。有关Object名称命名规范的更多信息，请参见对象（Object）。

追加上传本地文件

通过追加上传的方式将本地文件上传至指定Bucket的示例代码如下：

```
#include "oss_api.h"
#include "aos_http_io.h"
```

```
/* Endpoint以杭州为例，其它Region请按实际情况填写。*/
const char *endpoint = "<yourEndpoint>";
/* 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。*/
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
/* 设置Bucket名称。*/
const char *bucket_name = "<yourBucketName>";
/* 设置Object名称。即填写不包含Bucket名称在内的Object的完整路径，例如example/folder/abc.jpg。*/
const char *object_name = "<yourObjectName>";
/* 设置本地文件名称。即填写本地文件的完整路径，例如/users/local/abc.jpg。
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  int64_t position = 0;
  char *next_append_position = NULL;
  aos_table_t *headers1 = NULL;
  aos_table_t *headers2 = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  /* 获取起始追加位置。*/
```

```
  /* 获取追加追加位置。 */
  resp_status = oss_head_object(oss_client_options, &bucket, &object, headers1, &resp_headers);
  if(aos_status_is_ok(resp_status)) {
    next_append_position = (char*)(apr_table_get(resp_headers, "x-oss-next-append-position"));
    position = atoi(next_append_position);
  }
  /* 追加文件。 */
  resp_status = oss_append_object_from_file(oss_client_options, &bucket, &object, position, &file, headers
2, &resp_headers);
  /* 判断追加是否成功。 */
  if (aos_status_is_ok(resp_status)) {
    printf("append object from file succeeded\n");
  } else {
    printf("append object from file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。 */
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。 */
  aos_http_io_deinitialize();
  return 0;
}
```

12.6.4. 断点续传上传

断点续传上传是指将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

您可以通过 `oss_resumable_clt_params_t` 实现断点续传。该方法常见参数如下：

参数	说明
thread_num	并发线程数，默认为1。
enable_checkpoint	是否开启断点续传。默认不开启。
partSize	分片上传大小，取值范围为100KB~5GB。
checkpoint_path	checkpoint文件的路径，默认为{upload_file_path}.cp。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。 */
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
```

```
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_list_t resp_body;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_resumable_clt_params_t *clt_params;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  aos_list_init(&resp_body);
  /* 断点续传。*/
  clt_params = oss_create_resumable_clt_params_content(pool, 1024 * 100, 3, AOS_TRUE, NULL);
  resp_status = oss_resumable_upload_file(oss_client_options, &bucket, &object, &file, headers, NULL, clt_
params, NULL, &resp_headers, &resp_body);
  if (aos_status_is_ok(resp_status)) {
    printf("resumable upload succeeded\n");
  } else {
    printf("resumable upload failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.6.5. 分片上传

OSS提供的分片上传（Multipart Upload）功能，将要上传的较大文件（Object）分成多个数据块（Part）来分别上传，上传完成后再调用CompleteMultipartUpload接口将这些Part组合成一个Object来达到断点续传的效果。

分片上传流程

分片上传（Multipart Upload）分为以下三个步骤：

1. 初始化一个分片上传事件。

   调用oss_init_multipart_upload方法返回OSS创建的全局唯一的uploadId。

2. 上传分片。

   调用oss_upload_part_from_file方法上传分片数据。

   > 说明
   > - 对于同一个uploadId，分片号（PartNumber）标识了该分片在整个文件内的相对位置。如果使用同一个分片号上传了新的数据，则OSS上该分片已有的数据将会被覆盖。
   > - OSS将收到的分片数据的MD5值放在ETag头内返回给用户。
   > - OSS计算上传数据的MD5值，并与SDK计算的MD5值比较，如果不一致则返回InvalidDigest错误码。

3. 完成分片上传。

   所有分片上传完成后，调用oss_complete_multipart_upload方法将所有分片合并成完整的文件。

分片上传完整示例

以下通过一个完整的示例对分片上传的流程进行逐步解析：

```
#include "oss_api.h"
#include "aos_http_io.h"
#include <sys/stat.h>
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
```

```
int64_t get_file_size(const char *file_path)
{
  int64_t filesize = -1;
  struct stat statbuff;
  if(stat(file_path, &statbuff) < 0){
    return filesize;
  } else {
    filesize = statbuff.st_size;
  }
  return filesize;
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  oss_upload_file_t *upload_file = NULL;
  aos_string_t upload_id;
  aos_table_t *headers = NULL;
  aos_table_t *complete_headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_null(&upload_id);
  headers = aos_table_make(pool, 1);
  complete_headers = aos_table_make(pool, 1);
  int part_num = 1;
  /* 初始化分片上传，获取一个上传ID(upload_id)。*/
  resp_status = oss_init_multipart_upload(oss_client_options, &bucket, &object, &upload_id, headers, &res
p_headers);
  /* 判断分片上传初始化是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("Init multipart upload succeeded, upload_id:%.*s\n",
        upload_id.len, upload_id.data);
  } else {
    printf("Init multipart upload failed, upload_id:%.*s\n",
        upload_id.len, upload_id.data);
  }
  /* 上传分片。*/
```

```
  /* 上传分片。 */
  int64_t file_length = 0;
  int64_t pos = 0;
  aos_list_t complete_part_list;
    oss_complete_part_content_t* complete_content = NULL;
  char* part_num_str = NULL;
  char* etag = NULL;
  aos_list_init(&complete_part_list);
  file_length = get_file_size(local_filename);
  while(pos < file_length) {
    upload_file = oss_create_upload_file(pool);
    aos_str_set(&upload_file->filename, local_filename);
    upload_file->file_pos = pos;
    pos += 100 * 1024;
    upload_file->file_last = pos < file_length ? pos : file_length;
    resp_status = oss_upload_part_from_file(oss_client_options, &bucket, &object, &upload_id, part_num+
+, upload_file, &resp_headers);
    /* 保存分片号和ETag。*/
    complete_content = oss_create_complete_part_content(pool);
    part_num_str = apr_psprintf(pool, "%d", part_num-1);
    aos_str_set(&complete_content->part_number, part_num_str);
    etag = apr_pstrdup(pool,
    (char*)apr_table_get(resp_headers, "ETag"));
    aos_str_set(&complete_content->etag, etag);
    aos_list_add_tail(&complete_content->node, &complete_part_list);
    if (aos_status_is_ok(resp_status)) {
      printf("Multipart upload part from file succeeded\n");
    } else {
      printf("Multipart upload part from file failed\n");
    }
  }
  /* 完成分片上传。*/
  resp_status = oss_complete_multipart_upload(oss_client_options, &bucket, &object, &upload_id,
      &complete_part_list, complete_headers, &resp_headers);
  /* 判断分片上传是否完成。*/
  if (aos_status_is_ok(resp_status)) {
    printf("Complete multipart upload from file succeeded, upload_id:%.*s\n",
        upload_id.len, upload_id.data);
  } else {
    printf("Complete multipart upload from file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

获取已经上传的分片，调用oss_complete_multipart_upload接口时需要每个分片的ETag值。您可以通过以下两种方式获取ETag值：

- 上传每个分片时，返回结果中会包含这个分片的ETag值，供您直接保存使用。
- 通过调用oss_list_upload_part接口获取已经上传的分片的ETag值。上述示例采用的此种方式。

取消分片上传事件

以下代码用于取消分片上传事件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t upload_id;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_null(&upload_id);
  /* 初始化分片上传，获取一个上传ID(upload_id)。*/
  resp_status = oss_init_multipart_upload(oss_client_options, &bucket, &object, &upload_id, headers, &res
p_headers);
  /* 判断是否初始化分片上传成功。 */
  if (aos_status_is_ok(resp_status)) {
```

```
  if (aos_status_is_ok(resp_status)) {
    printf("Init multipart upload succeeded, upload_id:%.*s\n",
        upload_id.len, upload_id.data);
  } else {
    printf("Init multipart upload failed, upload_id:%.*s\n",
        upload_id.len, upload_id.data);
  }
  /* 取消这次分片上传。*/
  resp_status = oss_abort_multipart_upload(oss_client_options, &bucket, &object, &upload_id, &resp_hea
ders);
  /* 判断取消分片上传是否成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("Abort multipart upload succeeded, upload_id::%.*s\n",
        upload_id.len, upload_id.data);
  } else {
    printf("Abort multipart upload failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

> **说明** 当一个分片上传事件被取消后，就不能再使用upload_id做任何操作，已经上传的分片数据也会被删除。

12.6.6. 进度条

进度条用于指示上传或下载的进度。

上传进度条的完整代码请参见Git Hub。

下面的代码以Put Object方法为例，介绍如何使用进度条。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
```

```
  options->ctl = aos_http_controller_create(options->pool, 0);
}
void percentage(int64_t consumed_bytes, int64_t total_bytes)
{
  assert(total_bytes >= consumed_bytes);
  printf("%%%" APR_INT64_T_FMT "\n", consumed_bytes * 100 / total_bytes);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_list_t resp_body;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  /* 带进度条的上传。*/
  resp_status = oss_do_put_object_from_file(oss_client_options, &bucket, &object, &file, NULL, NULL, perc
entage, &resp_headers, &resp_body);
  if (aos_status_is_ok(resp_status)) {
    printf("put object from file succeeded\n");
  } else {
    printf("put object from file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

oss_put_object_from_file、oss_put_object_from_buffer不支持进度条功能。

12.6.7. 上传回调

本文介绍如何使用上传回调。

上传回调的完整代码请参见GitHub。

以下代码用于上传回调（callback）。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *object_content = "More than just cloud.";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  aos_pool_t *p = NULL;
  aos_status_t *s = NULL;
  aos_string_t bucket;
  aos_string_t object;
  aos_table_t *headers = NULL;
  oss_request_options_t *options = NULL;
  aos_table_t *resp_headers = NULL;
  aos_list_t resp_body;
  aos_list_t buffer;
  aos_buf_t *content;
  char *buf = NULL;
  int64_t len = 0;
  int64_t size = 0;
  int64_t pos = 0;
  char b64_buf[1024];
  int b64_len;
  /* 采用JSON格式。*/
  /* （可选）设置回调请求消息头中Host的值，如您的服务器配置Host的值。*/
  char *callback = "{"
    "\"callbackUrl\":\"http://oss-demo.aliyuncs.com:23450\","
     "\"callbackHost\":\"yourCallbackHost\","
    "\"callbackBody\":\"bucket=${bucket}&object=${object}&size=${size}&mimeType=${mimeType}\","
    "\"callbackBodyType\":\"application/x-www-form-urlencoded\""
    "}";
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
```

```
    }
    /* 初始化参数。*/
    aos_pool_create(&p, NULL);
    options = oss_request_options_create(p);
    init_options(options);
    aos_str_set(&bucket, bucket_name);
    aos_str_set(&object, object_name);
    aos_list_init(&resp_body);
    aos_list_init(&buffer);
    content = aos_buf_pack(options->pool, object_content, strlen(object_content));
    aos_list_add_tail(&content->node, &buffer);
    /* 将callback放入header。*/
    b64_len = aos_base64_encode((unsigned char*)callback, strlen(callback), b64_buf);
    b64_buf[b64_len] = '\0';
    headers = aos_table_make(p, 1);
    apr_table_set(headers, OSS_CALLBACK, b64_buf);
    /* 上传回调。*/
    s = oss_do_put_object_from_buffer(options, &bucket, &object, &buffer,
      headers, NULL, NULL, &resp_headers, &resp_body);
    if (aos_status_is_ok(s)) {
      printf("put object from buffer succeeded\n");
    } else {
      printf("put object from buffer failed\n");
    }
    /* 获取buffer长度。*/
    len = aos_buf_list_len(&resp_body);
    buf = (char *)aos_pcalloc(p, (apr_size_t)(len + 1));
    buf[len] = '\0';
    /* 拷贝buffer内容到内存。*/
    aos_list_for_each_entry(aos_buf_t, content, &resp_body, node) {
      size = aos_buf_size(content);
      memcpy(buf + pos, content->pos, (size_t)size);
      pos += size;
    }
    /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
    aos_pool_destroy(p);
    /* 释放之前分配的全局资源。*/
    aos_http_io_deinitialize();
    return 0;
}
```

关于上传回调的更多信息，请参见上传回调。调试方法和错误排除请参见上传回调错误及排除。

12.7. 下载文件

12.7.1. 概述

文件下载的完整代码请参见GitHub。

OSS C SDK提供了丰富的文件下载方式：

- 下载到本地文件
- 下载到本地内存

- 范围下载
- 断点续传下载

下载过程中，您还可以通过进度条功能查看下载进度。

12.7.2. 下载到本地文件

本文介绍如何将OSS文件下载到本地。

下载OSS文件的完整代码请详见GitHub。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *params;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
```

```
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  params = aos_table_make(pool, 0);
  /* 下载文件。如果指定的本地文件存在会覆盖，不存在则新建。*/
  resp_status = oss_get_object_to_file(oss_client_options, &bucket, &object, headers, params, &file, &resp_
headers);
  if (aos_status_is_ok(resp_status)) {
    printf("Get object from file succeeded\n");
  } else {
    printf("Get object from file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

说明 当前版本与1.0.0版本相比，oss_get_object_to_file接口新增params参数，headers和params参数允许为NULL。

12.7.3. 下载到本地内存

本文介绍如何将OSS文件下载到本地内存。

下载OSS文件的完整代码请参见GitHub。

以下代码用于把指定的OSS文件下载到内存中：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
```

```
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  aos_buf_t *content = NULL;
  aos_table_t *params = NULL;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  char *buf = NULL;
  int64_t len = 0;
  int64_t size = 0;
  int64_t pos = 0;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_list_init(&buffer);
  /* 下载文件到本地内存。*/
  resp_status = oss_get_object_to_buffer(oss_client_options, &bucket, &object,
                    headers, params, &buffer, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to buffer succeeded\n");
    /* 将下载内容拷贝到buffer中。*/
    len = aos_buf_list_len(&buffer);
    buf = aos_pcalloc(pool, len + 1);
    buf[len] = '\0';
    aos_list_for_each_entry(aos_buf_t, content, &buffer, node) {
    size = aos_buf_size(content);
      memcpy(buf + pos, content->pos, size);
      pos += size;
    }
  }
  else {
    printf("get object to buffer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

```
}
```

12.7.4. 范围下载

如果仅需要文件中的部分数据，您可以使用范围下载，下载指定范围内的数据。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  aos_buf_t *content = NULL;
  aos_table_t *params = NULL;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  char *buf = NULL;
  int64_t len = 0;
  int64_t size = 0;
```

```
  int64_t pos = 0;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_list_init(&buffer);
  headers = aos_table_make(pool, 1);
  /* 设置Range，取值范围为第20至第100字节。*/
  apr_table_set(headers, "Range", "bytes=20-100");
  /* 下载文件到内存。*/
  resp_status = oss_get_object_to_buffer(oss_client_options, &bucket, &object, headers, params, &buffer,
&resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to buffer succeeded\n");
    /* 获取buffer长度。*/
    aos_list_for_each_entry(aos_buf_t, content, &buffer, node) {
      len += aos_buf_size(content);
    }
    buf = aos_pcalloc(pool, (apr_size_t)(len + 1));
    buf[len] = '\0';
    /* 拷贝buffer内容到内存。*/
    aos_list_for_each_entry(aos_buf_t, content, &buffer, node) {
      size = aos_buf_size(content);
      memcpy(buf + pos, content->pos, (size_t)size);
      pos += size;
    }
  }
  else {
    printf("get object to buffer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.7.5. 断点续传下载

大文件下载时，如果网络不稳定或者发生其它异常，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

您可以通过 `oss_resumable_clt_params_t` 方法实现断点续传下载， `oss_resumable_clt_params_t` 包含以下参数：

参数	说明
thread_num	并发线程数，默认为1。
enable_checkpoint	是否开启断点续传。默认不开启。
partSize	分片上传大小，取值范围为1B~5GB，单位为byte。

参数	说明
checkpoint_path	checkpoint文件的路径，默认在{upload_file_path}.cp目录下。

下载过程中，会将当前下载的进度信息记录在 `checkpoint` 文件中，如果下载失败，再次下载将匹配 `checkpoint` 文件中的记录，继续之前的下载。下载完成后，该 `checkpoint` 文件会被删除。

以下代码用于断点续传下载：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
```

```
  oss_resumable_clt_params_t *clt_params;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  /* 断点续传文件。*/
  clt_params = oss_create_resumable_clt_params_content(pool, 1024 * 100, 3, AOS_TRUE, NULL);
  resp_status = oss_resumable_download_file(oss_client_options, &bucket, &object, &file, headers, NULL, c
lt_params, NULL, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("download succeeded\n");
  } else {
    printf("download failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.7.6. 进度条

进度条用于指示上传或下载的进度。

下载进度条的完整代码请参见Git Hub。

以下代码展示如何使用进度条：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
const char *download_filename = "<yourDownloadFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
void percentage(int64_t consumed_bytes, int64_t total_bytes)
{
  assert(total_bytes >= consumed_bytes);
  printf("%%%" APR_INT64_T_FMT "\n", consumed_bytes * 100 / total_bytes);
}
```

```
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *resp_headers = NULL;
  aos_list_t resp_body;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  /* 带进度条的文件上传。*/
  resp_status = oss_do_put_object_from_file(oss_client_options, &bucket, &object, &file, NULL, NULL, percentage, &resp_headers, &resp_body);
  if (aos_status_is_ok(resp_status)) {
    printf("put object from file succeeded\n");
  } else {
    printf("put object from file failed\n");
  }
  /* 带进度条的文件下载。*/
  aos_pool_create(&pool, NULL);
  oss_client_options = oss_request_options_create(pool);
  init_options(oss_client_options);
  aos_str_set(&file, download_filename);
  resp_status = oss_do_get_object_to_file(oss_client_options, &bucket, &object, NULL, NULL, &file, percentage, NULL);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to file succeeded\n");
  } else {
    printf("get object to file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

oss_get_object_from_file、oss_get_object_from_buffer不支持进度条功能。

12.8. 管理文件

12.8.1. 概述

您可以通过一系列的接口管理存储空间（Bucket）下的文件（Object），包括以下操作：

- 管理文件访问权限
- 管理文件元信息
- 列举文件
- 拷贝文件
- 删除文件
- 解冻归档文件
- 管理软链接

12.8.2. 管理文件访问权限

本文介绍如何管理文件访问权限。

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	OSS_ACL_DEFAULT
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	OSS_ACL_PRIVATE
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ
公共读写	所有用户都有该文件的读写权限。请谨慎使用该权限。	OSS_ACL_PUBLIC_READ_WRITE

文件的访问权限优先级高于存储空间的访问权限。例如存储空间的访问权限是私有，而文件的访问权限是公共读写，则所有用户都有该文件的读写权限。如果某个文件没有设置过访问权限，则遵循存储空间的访问权限。

以下是设置并获取文件访问权限的完整代码：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
    options->config = oss_config_create(options->pool);
```

```
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  oss_acl_e oss_acl = OSS_ACL_PUBLIC_READ;
  /* 设置文件权限。*/
  resp_status = oss_put_object_acl(oss_client_options, &bucket, &object, oss_acl, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put object acl success!\n");
  } else {
    printf("put object acl failed!\n");
  }
  /* 获取文件权限。*/
  aos_string_t oss_acl_string;
  resp_status = oss_get_object_acl(oss_client_options, &bucket, &object, &oss_acl_string, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object acl success!\n");
    printf("acl: %s \n", oss_acl_string.data);
  } else {
    printf("get object acl failed!\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
```

```
  return 0;
}
```

12.8.3. 管理文件元信息

文件元信息（Object Meta）包括HTTP header和自定义元信息，详情请参见开发指南中的文件元信息。

在OSS中上传文件后或者读取文件前，您可以通过 `oss_get_object_meta` 接口获取文件的一些元信息，比如长度，文件类型等。 以下代码用于设置和获取文件元信息：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_table_t *headers;
  aos_list_t buffer;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
```

```
  aos_buf_t *content = NULL;
  char *content_length_str = NULL;
  char *object_type = NULL;
  char *object_author = NULL;
  int64_t content_length = 0;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  headers = aos_table_make(pool, 2);
  /* 设置用户自定义元信息。*/
  apr_table_set(headers, "Expires", "Fri, 28 Feb 2032 05:38:42 GMT");
  apr_table_set(headers, "x-oss-meta-author", "oss");
  aos_list_init(&buffer);
  content = aos_buf_pack(oss_client_options->pool, object_content, strlen(object_content));
  aos_list_add_tail(&content->node, &buffer);
  /* 设置文件权限，从缓存上传文件。*/
  resp_status = oss_put_object_from_buffer(oss_client_options, &bucket, &object,
        &buffer, headers, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put object from buffer with md5 succeeded\n");
  } else {
    printf("put object from buffer with md5 failed\n");
  }
  /* 获取文件权限。*/
  resp_status = oss_get_object_meta(oss_client_options, &bucket, &object, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    content_length_str = (char*)apr_table_get(resp_headers, OSS_CONTENT_LENGTH);
    if (content_length_str != NULL) {
      content_length = atol(content_length_str);
    }
    object_author = (char*)apr_table_get(resp_headers, OSS_AUTHORIZATION);
    object_type = (char*)apr_table_get(resp_headers, OSS_OBJECT_TYPE);
    printf("get object meta succeeded, object author:%s, object type:%s, content_length:%ld\n", object_aut
hor, object_type, content_length);
  } else {
    printf("req:%s, get object meta failed\n", resp_status->req_id);
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

说明 HTTP header详情请参见RFC2616。

12.8.4. 列举文件

本文介绍如何列举文件。

列举文件完整代码请参见GitHub。

OSS文件按照字母顺序排列。您可以通过 `oss_list_object` 列出存储空间下的文件。主要的参数如下：

参数	说明
delimiter	对文件名称进行分组的一个字符。所有名称包含指定的前缀且第一次出现delimiter字符之间的文件作为一组元素（commonPrefixes）。
prefix	限定返回的文件必须以prefix作为前缀。
max_ret	限定此次列举文件的最大个数。默认值为100，最大值为1000。
marker	列举指定marker之后的文件。

列举所有文件

以下代码用于列出当前存储空间下的所有文件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、access_key_secret、is_cname、curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
```

```
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  char *prefix = "";
  char *nextMarker = "";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

说明 默认列举文件数最多1000，若超过，则只返回前1000个文件， 且返回结果中的 truncated 为 true，并返回 next_marker 作为下次读取的起点。若想调整返回文件数，可修改 max_ret 参数，或使用marker参数分次读取。

列举指定个数的文件

以下代码用于列举指定个数的文件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
```

```
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，这个方法内部会做一些全局资源的初始化，涉及网络、内存等部分。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, curl
参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  char *prefix = "";
  char *nextMarker = "";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  /* 设置列举文件的最大个数为200。*/
  params->max_ret = 200;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
  if (!aos_status_is_ok(resp_status))
  {
    printf("list object failed\n");
  }
  aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
```

```
    ++size;
    line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
      content->size.len, content->size.data,
      content->last_modified.len, content->last_modified.data);
    printf("%s", line);
  }
  printf("Total %d\n", size);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(pool);
  /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

列举指定前缀的文件

以下代码用于列举包含指定前缀（prefix）的文件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，这个方法内部会做一些全局资源的初始化，涉及网络，内存等部分。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, curl
参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
```

```
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  /* 列举包含指定前缀的文件。*/
  char *prefix = "<yourObjectPefix>";
  char *nextMarker = "";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(pool);
  /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

列举指定marker之后的文件

参数marker代表文件名称。以下代码用于列举指定marker之后的文件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
```

```
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，这个方法内部会做一些全局资源的初始化，涉及网络，内存等部分。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, curl
参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  char *prefix = "";
  /* 列举指定marker之后的文件。*/
  char *nextMarker = "<yourObjectMarker>";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
```

```
        line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
          content->size.len, content->size.data,
          content->last_modified.len, content->last_modified.data);
        printf("%s", line);
      }
      nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
      aos_str_set(&params->marker, nextMarker);
      aos_list_init(&params->object_list);
      aos_list_init(&params->common_prefix_list);
    } while (params->truncated == AOS_TRUE);
    printf("Total %d\n", size);
    /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
    aos_pool_destroy(pool);
    /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源。*/
    aos_http_io_deinitialize();
    return 0;
}
```

文件夹功能

OSS没有文件夹的概念，所有元素都是以文件来存储。创建文件夹本质上来说是创建了一个大小为0并以正斜线（/）结尾的文件。这个文件可以被上传和下载，控制台会对以正斜线（/）结尾的文件以文件夹的方式展示。

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为objects。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见文件夹功能。创建文件夹的完整代码请参见GitHub。

假设存储空间中包含文件 `oss.jpg` 、 `fun/test.jpg` 、 `fun/movie/001.avi` 和 `fun/movie/007.avi` ，以正斜线（/）作为文件夹的分隔符。以下示例说明了如何通过模拟文件夹的方式列举文件。

- 列举存储空间下所有文件

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
```

```
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、access_key_secret、is_cname、curl等全
局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  char *nextMarker = "";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
```

```
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
oss.jpg
CommonPrefixes:
```

- 列举指定目录下所有文件

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* aos_str_set是用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，该方法内部会做某些全局资源的初始化，涉及网络、内存等部分。*
/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, cu
rl参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
```

```
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  int size = 0;
  char *line = NULL;
  /* 列举包含指定前缀的文件。*/
  char *prefix = "fun/";
  char *nextMarker = "";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  printf("Object\tSize\tLastModified\n");
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(pool);
  /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源 */
  aos_http_io_deinitialize();
  return 0;
}
```

```
Objects:
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
CommonPrefixes:
```

- 列举目录下的文件和子目录

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
```

```
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* aos_str_set是用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，该方法内部会做某些全局资源的初始化，涉及网络、内存等部分。*
/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, cu
rl参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  oss_list_object_common_prefix_t *commonPrefix = NULL;
  int size = 0;
  char *line = NULL;
  /* 列举包含指定前缀的文件。*/
  char *prefix = "fun/";
  char *nextMarker = "";
  /* 设置正斜线（/）为文件夹的分隔符。*/
  char *delimiter = "/";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  aos_str_set(&params->delimiter, delimiter);
  printf("Object\tSize\tLastModified\n");
  /*列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
```

```
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    aos_list_for_each_entry(oss_list_object_common_prefix_t, commonPrefix, &params->common_prefi
x_list, node) {
      line = apr_psprintf(pool, "%.*s", commonPrefix->prefix.len, commonPrefix->prefix.data);
      printf("%s", line);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(pool);
  /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

```
Objects:
fun/test.jpg
CommonPrefixes:
fun/movie/
```

- 列举指定目录下的文件大小

  以下代码用于获取指定目录下的文件大小：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* aos_str_set是用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否设置了CNAME。0表示不设置。*/
  options->config->is_cname = 0;
```

```
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int64_t calculateFolderLength (aos_string_t bucketName, char* folder)
{
  int64_t size = 0;
  oss_list_object_params_t *params = NULL;
  char *nextMarker = "";
  aos_status_t *resp_status = NULL;
  aos_pool_t *p = NULL;
  oss_request_options_t *options = NULL;
  oss_list_object_content_t* content = NULL;
  aos_pool_create(&p, NULL);
  options = oss_request_options_create(p);
  params = oss_create_list_object_params(p);
  params->max_ret = 10;
  aos_str_set(&params->prefix, folder);
  aos_str_set(&params->marker, nextMarker);
  /* 列举所有文件。*/
  do {
    resp_status = oss_list_object(options, &bucketName, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      size += content->size.len;
    }
    nextMarker = apr_psprintf(p, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(p);
  return size;
}
int main(int argc, char *argv[])
{
  /* 程序入口调用aos_http_io_initialize方法，该方法内部会做某些全局资源的初始化，涉及网络、内存等部分。*
/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 等价于apr_pool_t，用于内存管理的内存池，实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个新的内存池，第二个参数值是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,access_key_secret，is_cname, cu
rl参数等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* options的内存是由pool分配的，后续释放pool后，相当于释放了options的内存，不需要单独释放内存。*/
  oss_client_options = oss_request_options_create(pool);
```

```
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_status_t *resp_status = NULL;
  oss_list_object_params_t *params = NULL;
  oss_list_object_content_t *content = NULL;
  oss_list_object_common_prefix_t *commonPrefix = NULL;
  int size = 0;
  char *line = NULL;
  /* 列举包含指定前缀的文件。*/
  char *prefix = "fun/";
  char *nextMarker = "";
  /* 设置正斜线（/）为文件夹的分隔符。*/
  char *delimiter = "/";
  aos_str_set(&bucket, bucket_name);
  params = oss_create_list_object_params(pool);
  params->max_ret = 100;
  aos_str_set(&params->prefix, prefix);
  aos_str_set(&params->marker, nextMarker);
  aos_str_set(&params->delimiter, delimiter);
  printf("Object\tSize\tLastModified\n");
  /*列举所有文件。*/
  do {
    resp_status = oss_list_object(oss_client_options, &bucket, params, NULL);
    if (!aos_status_is_ok(resp_status))
    {
      printf("list object failed\n");
      break;
    }
    aos_list_for_each_entry(oss_list_object_content_t, content, &params->object_list, node) {
      ++size;
      line = apr_psprintf(pool, "%.*s\t%.*s\t%.*s\n", content->key.len, content->key.data,
        content->size.len, content->size.data,
        content->last_modified.len, content->last_modified.data);
      printf("%s", line);
    }
    aos_list_for_each_entry(oss_list_object_common_prefix_t, commonPrefix, &params->common_prefi
x_list, node) {
      line = apr_psprintf(pool, "%.*s", commonPrefix->prefix.len, commonPrefix->prefix.data);
      int64_t size = calculateFolderLength(bucket, commonPrefix->prefix.data);
      printf("Total %d\n", size);
    }
    nextMarker = apr_psprintf(pool, "%.*s", params->next_marker.len, params->next_marker.data);
    aos_str_set(&params->marker, nextMarker);
    aos_list_init(&params->object_list);
    aos_list_init(&params->common_prefix_list);
  } while (params->truncated == AOS_TRUE);
  printf("Total %d\n", size);
  /* 执行完一个请求后，释放这个内存池，相当于释放了这个请求过程中各个部分分配的内存。*/
  aos_pool_destroy(pool);
  /* 程序结束前，调用aos_http_io_deinitialize方法释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

```
}
```

12.8.5. 删除文件

本文介绍如何删除文件。

> **警告** 请您谨慎使用删除操作，文件删除后将无法恢复。

删除文件的完整代码请参见Git Hub。

删除单个文件

以下代码用于删除examplebucket中的exampleobject.txt文件。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_table_t *resp_headers = NULL;
```

```
  aos_status_t *resp_status = NULL;
  /* 将char*类型数据赋值给aos_string_t类型的bucket。*/
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 删除文件。*/
  resp_status = oss_delete_object(oss_client_options, &bucket, &object, &resp_headers);
  /* 判断是否删除成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("delete object succeed\n");
  } else {
    printf("delete object failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

删除多个文件

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name1 = "<yourObjectName1>";
const char *object_name2 = "<yourObjectName2>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
```

```
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object1;
  aos_string_t object2;
  int is_quiet = 1;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object1, object_name1);
  aos_str_set(&object2, object_name2);
  /* 构建待删除文件列表。*/
  aos_list_t object_list;
  aos_list_t deleted_object_list;
  oss_object_key_t *content1;
  oss_object_key_t *content2;
  aos_list_init(&object_list);
  aos_list_init(&deleted_object_list);
  content1 = oss_create_oss_object_key(pool);
  aos_str_set(&content1->key, object_name1);
  aos_list_add_tail(&content1->node, &object_list);
  content2 = oss_create_oss_object_key(pool);
  aos_str_set(&content2->key, object_name2);
  aos_list_add_tail(&content2->node, &object_list);
  /* 删除文件列表中的文件。`is_quiet`表示是否返回删除的结果。*/
  resp_status = oss_delete_objects(oss_client_options, &bucket, &object_list, is_quiet, &resp_headers, &de
leted_object_list);
  if (aos_status_is_ok(resp_status)) {
    printf("delete objects succeeded\n");
  } else {
    printf("delete objects failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

批量删除指定前缀的文件

批量删除指定前缀的文件可以实现删除目录的功能，如prefix的值为 dir/，表示删除dir目录。 以下代码用于批量删除指定前缀的文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_prefix = "<yourObjectNamePrefix>";
```

```
const char *object_prefix = "<yourObjectNamePrefix>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t prefix;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&prefix, object_prefix);
  /* 删除指定前缀的文件。*/
  resp_status = oss_delete_objects_by_prefix(oss_client_options, &bucket, &prefix);
  if (aos_status_is_ok(resp_status)) {
    printf("delete objects by prefix succeeded\n");
  } else {
    printf("delete objects by prefix failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

警告 如果前缀prefix的值为空字符串或者NULL，将会删除整个存储空间里的文件，请谨慎使用。

12.8.6. 拷贝文件

本文介绍如何拷贝文件。

拷贝小文件

说明

以下代码用于拷贝小文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *source_bucket_name = "<yourSourceBucketName>";
const char *source_object_name = "<yourSourceObjectName>";
const char *dest_bucket_name = "<yourDestBucketName>";
const char *dest_object_name = "<yourDestObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t source_bucket;
  aos_string_t source_object;
```

```
  aos_string_t dest_bucket;
  aos_string_t dest_object;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&source_bucket, source_bucket_name);
  aos_str_set(&source_object, source_object_name);
  aos_str_set(&dest_bucket, dest_bucket_name);
  aos_str_set(&dest_object, dest_object_name);
  headers = aos_table_make(pool, 0);
  /* 拷贝文件。*/
  resp_status = oss_copy_object(oss_client_options, &source_bucket, &source_object, &dest_bucket, &des
t_object, headers, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("copy object succeeded\n");
  } else {
    printf("copy object failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

拷贝大文件

对于大于1GB的文件，建议通过oss_upload_part_copy接口来进行分片拷贝（UploadPartCopy）。 以下代码用于分片拷贝文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
#include <sys/stat.h>
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *source_bucket_name = "<yourSourceBucketName>";
const char *source_object_name = "<yourSourceObjectName>";
const char *dest_bucket_name = "<yourDestBucketName>";
const char *dest_object_name = "<yourDestObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int64_t get_file_size(const char *file_path)
```

```
{
  int64_t filesize = -1;
  struct stat statbuff;
  if(stat(file_path, &statbuff) < 0){
    return filesize;
  } else {
    filesize = statbuff.st_size;
  }
  return filesize;
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t dest_bucket;
  aos_string_t dest_object;
  aos_string_t upload_id;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  oss_list_upload_part_params_t *list_upload_part_params;
  oss_upload_part_copy_params_t *upload_part_copy_params1;
  oss_upload_part_copy_params_t *upload_part_copy_params2;
  oss_list_part_content_t *part_content;
  aos_list_t complete_part_list;
  oss_complete_part_content_t *complete_content;
  aos_table_t *list_part_resp_headers = NULL;
  aos_table_t *complete_resp_headers = NULL;
  int part1 = 1;
  int part2 = 2;
  int64_t range_start1 = 0;
  int64_t range_end1 = 6000000;//not less than 5MB
  int64_t range_start2 = 6000001;
  int64_t range_end2;
  int max_ret = 1000;
  headers = aos_table_make(pool, 0);
  aos_str_set(&dest_bucket, dest_bucket_name);
  aos_str_set(&dest_object, dest_object_name);
  resp_status = oss_init_multipart_upload(oss_client_options, &dest_bucket, &dest_object, &upload_id, headers, &resp_headers);
```

```
  if (aos_status_is_ok(resp_status)) {
    printf("init multipart upload succeeded, upload_id is %s\n", upload_id.data);
  } else {
    printf("init multipart upload failed\n");
  }
  /* 拷贝第一个分片数据。*/
  upload_part_copy_params1 = oss_create_upload_part_copy_params(pool);
  aos_str_set(&upload_part_copy_params1->source_bucket, source_bucket_name);
  aos_str_set(&upload_part_copy_params1->source_object, source_object_name);
  aos_str_set(&upload_part_copy_params1->dest_bucket, dest_bucket_name);
  aos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
  aos_str_set(&upload_part_copy_params1->upload_id, upload_id.data);
  upload_part_copy_params1->part_num = part1;
  upload_part_copy_params1->range_start = range_start1;
  upload_part_copy_params1->range_end = range_end1;
  headers = aos_table_make(pool, 0);
  resp_status = oss_upload_part_copy(oss_client_options, upload_part_copy_params1, headers, &resp_he
aders);
  if (aos_status_is_ok(resp_status)) {
    printf("upload part 1 copy succeeded\n");
  } else {
    printf("upload part 1 copy failed\n");
  }
  /* 拷贝第二个分片数据。*/
  range_end2 = get_file_size(local_filename) - 1;
  upload_part_copy_params2 = oss_create_upload_part_copy_params(pool);
  aos_str_set(&upload_part_copy_params2->source_bucket, source_bucket_name);
  aos_str_set(&upload_part_copy_params2->source_object, source_object_name);
  aos_str_set(&upload_part_copy_params2->dest_bucket, dest_bucket_name);
  aos_str_set(&upload_part_copy_params2->dest_object, dest_object_name);
  aos_str_set(&upload_part_copy_params2->upload_id, upload_id.data);
  upload_part_copy_params2->part_num = part2;
  upload_part_copy_params2->range_start = range_start2;
  upload_part_copy_params2->range_end = range_end2;
  headers = aos_table_make(pool, 0);
  resp_status = oss_upload_part_copy(oss_client_options, upload_part_copy_params2, headers, &resp_he
aders);
  if (aos_status_is_ok(resp_status)) {
    printf("upload part 2 copy succeeded\n");
  } else {
    printf("upload part 2 copy failed\n");
  }
  /* 列出分片。*/
  headers = aos_table_make(pool, 0);
  list_upload_part_params = oss_create_list_upload_part_params(pool);
  list_upload_part_params->max_ret = max_ret;
  aos_list_init(&complete_part_list);
  resp_status = oss_list_upload_part(oss_client_options, &dest_bucket, &dest_object, &upload_id, list_upl
oad_part_params, &list_part_resp_headers);
  aos_list_for_each_entry(oss_list_part_content_t, part_content, &list_upload_part_params->part_list, no
de) {
    complete_content = oss_create_complete_part_content(pool);
    aos_str_set(&complete_content->part_number, part_content->part_number.data);
    aos_str_set(&complete_content->etag, part_content->etag.data);
```

```
    aos_list_add_tail(&complete_content->node, &complete_part_list);
  }
  /* 完成分片拷贝。*/
  resp_status = oss_complete_multipart_upload(oss_client_options, &dest_bucket, &dest_object, &upload
_id, &complete_part_list, headers, &complete_resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("complete multipart upload succeeded\n");
  } else {
    printf("complete multipart upload failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.8.7. 解冻归档文件

归档类型的文件（Object）需要解冻（Restore）之后才能读取。本文介绍如何解冻归档类型的文件。

归档类型的Object在执行解冻前后的状态变换过程如下：

1. 归档类型的Object初始时处于冷冻状态。
2. 提交一次解冻请求后，Object处于解冻中的状态，完成解冻任务通常需要1分钟。
3. 服务端完成解冻任务后，Object进入解冻状态，此时您可以读取Object。解冻状态默认持续24小时，24小时内再次调用RestoreObject接口则解冻状态会自动延长24小时。对于同份归档文件，一次解冻流程内可有效调用7次RestoreObject接口达到最长7天的解冻持续时间。
4. 解冻状态结束后，Object再次返回到冷冻状态。

以下代码用于解冻归档类型Object：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *object_content = "More than just cloud.";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
```

```
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  oss_acl_e oss_acl = OSS_ACL_PRIVATE;
  aos_buf_t *content = NULL;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  headers = aos_table_make(pool, 0);
  /* 创建归档存储空间。*/
  resp_status = oss_create_bucket_with_storage_class(oss_client_options, &bucket, oss_acl, OSS_STORAGE_CLASS_ARCHIVE, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("create bucket succeeded\n");
  } else {
    printf("create bucket failed\n");
  }
  aos_list_init(&buffer);
  content = aos_buf_pack(oss_client_options->pool, object_content, strlen(object_content));
  aos_list_add_tail(&content->node, &buffer);
  /* 上传文件。*/
  resp_status = oss_put_object_from_buffer(oss_client_options, &bucket, &object, &buffer, headers, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put object from buffer succeeded\n");
  } else {
    printf("put object from buffer failed\n");
  }
  /* 解冻文件。*/
  do {
    headers = aos_table_make(pool, 0);
    resp_status = oss_restore_object(oss_client_options, &bucket, &object, headers, &resp_headers);
    printf("restore object resp_status->code: %d \n", resp_status->code);
    if (resp_status->code != 409) {
      break;
    } else {
```

```
    } else {
      printf("restore object is already in progress, resp_status->code: %d \n", resp_status->code);
      apr_sleep(5000);
    }
  } while (1);
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

归档类型的详细说明请参见存储类型介绍。相关状态码的详细解释请参见API文档RestoreObject。

12.8.8. 管理软链接

本文介绍如何创建软链接并获取软链接指向的目标文件（Object）名称。

创建软链接

软链接是一种特殊的文件，它指向某个具体的文件，类似于Windows上使用的快捷方式。软链接支持自定义元信息。

以下代码用于创建软链接：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *link_object_name = "<yourLinkObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于创建内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
```

```
  /* 创建并初始化options，该参数包括endpoint、access_key_id、access_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t sym_object;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&sym_object, link_object_name);
  resp_status = oss_put_symlink(oss_client_options, &bucket, &sym_object, &object, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("put symlink succeeded\n");
  } else {
    printf("put symlink failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

创建软链接的详细信息请参见PutSymlink。

获取软链接指向的目标文件名称

获取软链接要求您对该软链接有读权限。以下代码用于获取软链接指向的目标文件名称：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *link_object_name = "<yourLinkObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
```

```
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于创建内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、access_key_secret、is_cname、curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t link_object;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  char *target_object_name = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&link_object, link_object_name);
  resp_status = oss_get_symlink(oss_client_options, &bucket, &link_object, &resp_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get symlink succeeded\n");
    target_object_name = (char*)(apr_table_get(resp_headers, OSS_CANNONICALIZED_HEADER_SYMLINK));
    printf("target_object_name: %s \n", target_object_name);
  } else {
    printf("get symlink failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

获取软链接的详细信息请参见GetSymlink。

12.9. 授权访问

本文介绍如何使用STS以及签名URL临时授权访问OSS资源。

使用STS临时授权

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

以下代码用于使用STS凭证构造签名请求。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *sts_token = "<yourStsToken>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *object_content = "More than just cloud.";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  aos_str_set(&options->config->sts_token, sts_token);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，例如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_list_t buffer;
  aos_buf_t *content = NULL;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  /* 将类型为char*的数据赋值给bucket。*/
  aos_str_set(&bucket, bucket_name);
```

```
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_list_init(&buffer);
  content = aos_buf_pack(oss_client_options->pool, object_content, strlen(object_content));
  aos_list_add_tail(&content->node, &buffer);
  /* 上传文件。*/
  resp_status = oss_put_object_from_buffer(oss_client_options, &bucket, &object, &buffer, headers, &resp
_headers);
  /* 判断文件是否上传成功。*/
  if (aos_status_is_ok(resp_status)) {
    printf("put object from buffer succeeded\n");
  } else {
    printf("put object from buffer failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

说明 由于STS临时账号以及签名URL均需设置有效时长，当您使用STS临时账号生成签名URL执行相关操作（例如上传、下载文件）时，以最小的有效时长为准。例如您的STS临时账号的有效时长设置为1200秒、签名URL设置为3600秒时，当有效时长超过1200秒后，您无法使用此STS临时账号生成的签名URL上传文件。

使用签名URL临时授权

您可以将生成的签名URL提供给访客进行临时访问。生成签名URL时，您可以通过指定URL的过期时间来限制访客的访问时长。签名URL的默认过期时间为3600秒，最大值为32400秒。

以下介绍使用签名URL临时授权的常见示例。

- 使用签名URL上传文件

  以下代码用于使用签名URL上传文件：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
```

```
  options->config->is_cname = 0;
  /* 设置网络相关参数，例如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_http_request_t *req;
  apr_time_t now;
  char *url_str;
  aos_string_t url;
  int64_t expire_time;
  int one_hour = 3600;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  expire_time = now / 1000000 + one_hour;
  headers = aos_table_make(pool, 0);
  req = aos_http_request_create(pool);
  req->method = HTTP_PUT;
  now = apr_time_now(); /* 单位：微秒 */
  expire_time = now / 1000000 + one_hour;
  /* 生成签名URL。*/
  url_str = oss_gen_signed_url(oss_client_options, &bucket, &object, expire_time, req);
  aos_str_set(&url, url_str);
  printf("临时上传URL: %s\n", url_str);
  /* 使用签名URL上传文件。*/
  resp_status = oss_put_object_from_file_by_url(oss_client_options, &url, &file, headers, &resp_headers
);
  if (aos_status_is_ok(resp_status)) {
    printf("put objects by signed url succeeded\n");
  } else {
    printf("put objects by signed url failed\n");
  }
```

```
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

- 使用签名URL下载文件

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
const char *local_filename = "<yourLocalFilename>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用CNAME访问OSS服务。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，例如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *params = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
```

```
  aos_http_request_t *req;
  apr_time_t now;
  char *url_str;
  aos_string_t url;
  int64_t expire_time;
  int one_hour = 3600;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  aos_str_set(&file, local_filename);
  expire_time = now / 1000000 + one_hour;
  headers = aos_table_make(pool, 0);
  params = aos_table_make(pool, 0);
  req = aos_http_request_create(pool);
  req->method = HTTP_GET;
  now = apr_time_now(); /* 单位：微秒 */
  expire_time = now / 1000000 + one_hour;
  /* 生成签名URL。*/
  url_str = oss_gen_signed_url(oss_client_options, &bucket, &object, expire_time, req);
  aos_str_set(&url, url_str);
  /* 使用签名URL下载文件。*/
  resp_status = oss_get_object_to_file_by_url(oss_client_options, &url, headers, params, &file, &resp_he
aders);
  if (aos_status_is_ok(resp_status)) {
    printf("get object succeeded\n");
  } else {
    printf("get object failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

12.10. 图片处理

图片处理是OSS提供的海量、安全、低成本、高可靠的图片处理服务。原始图片上传到OSS后，您可以通过简单的RESTful接口，在任何时间、任何地点、任何互联网设备上对图片进行处理。

图片处理的完整代码请参见：GitHub。

使用图片处理参数处理图片

- 使用单个图片处理参数处理图片并保存为本地图片

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
```

```
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *params = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 将图片缩放为固定宽高100 px。*/
  params = aos_table_make(pool, 1);
  apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");
  /* 将处理后的图片保存到本地。*/
  aos_str_set(&file, "<LocalFileName>");
  resp_status = oss_get_object_to_file(oss_client_options, &bucket, &object, headers, params, &file, &res
p_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to file succeeded\n");
  } else {
    printf("get object to file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

- 使用多个图片处理参数处理图片并保存为本地图片

  使用多个图片处理参数处理图片时，多个参数之间以正斜线（/）分隔。

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局
配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *params = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 将图片缩放为固定宽高100 px后，再旋转90°。*/
  params = aos_table_make(pool, 1);
  apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100/rotate,90");
  /* 将处理后的图片保存到本地。*/
```

```
  aos_str_set(&file, "<LocalFileName>");
  resp_status = oss_get_object_to_file(oss_client_options, &bucket, &object, headers, params, &file, &res
p_headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to file succeeded\n");
  } else {
    printf("get object to file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

使用图片样式处理图片

您可以将多个图片处理参数封装在一个样式中，之后使用样式批量处理图片。详情请参见图片样式。以下代码展示了使用图片样式处理图片：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 用于设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配
置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
```

```
  /* 初始化client的选项oss_client_options。 */
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_string_t file;
  aos_table_t *headers = NULL;
  aos_table_t *params = NULL;
  aos_table_t *resp_headers = NULL;
  aos_status_t *resp_status = NULL;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 指定图片样式。*/
  params = aos_table_make(pool, 1);
  apr_table_set(params, OSS_PROCESS, "style/<yourCustomStyleName>");
  /* 将处理后的图片保存到本地。*/
  aos_str_set(&file, "<LocalFileName>");
  resp_status = oss_get_object_to_file(oss_client_options, &bucket, &object, headers, params, &file, &resp_
headers);
  if (aos_status_is_ok(resp_status)) {
    printf("get object to file succeeded\n");
  } else {
    printf("get object to file failed\n");
  }
  /* 释放内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

生成带图片处理参数的文件签名URL

私有文件的访问URL带有签名。OSS不支持在带签名的URL后直接添加图片处理参数。如果您想要对私有文件进行图片处理，需要将图片处理参数加入到签名中，相关的代码示例如下：

```
#include "oss_api.h"
#include "aos_http_io.h"
const char *endpoint = "<yourEndpoint>";
const char *access_key_id = "<yourAccessKeyId>";
const char *access_key_secret = "<yourAccessKeySecret>";
const char *bucket_name = "<yourBucketName>";
const char *object_name = "<yourObjectName>";
void init_options(oss_request_options_t *options)
{
  options->config = oss_config_create(options->pool);
  /* 用char*类型的字符串初始化aos_string_t类型。*/
  aos_str_set(&options->config->endpoint, endpoint);
  aos_str_set(&options->config->access_key_id, access_key_id);
  aos_str_set(&options->config->access_key_secret, access_key_secret);
  /* 是否使用了CNAME。0表示不使用。*/
  options->config->is_cname = 0;
  /* 设置网络相关参数，比如超时时间等。*/
  options->ctl = aos_http_controller_create(options->pool, 0);
}
```

```
}
int main(int argc, char *argv[])
{
  /* 在程序入口调用aos_http_io_initialize方法来初始化网络、内存等全局资源。*/
  if (aos_http_io_initialize(NULL, 0) != AOSE_OK) {
    exit(1);
  }
  /* 用于内存管理的内存池（pool），等价于apr_pool_t。其实现代码在apr库中。*/
  aos_pool_t *pool;
  /* 重新创建一个内存池，第二个参数是NULL，表示没有继承其它内存池。*/
  aos_pool_create(&pool, NULL);
  /* 创建并初始化options，该参数包括endpoint、access_key_id、acces_key_secret、is_cname、 curl等全局配置信息。*/
  oss_request_options_t *oss_client_options;
  /* 在内存池中分配内存给options。*/
  oss_client_options = oss_request_options_create(pool);
  /* 初始化Client的选项oss_client_options。*/
  init_options(oss_client_options);
  /* 初始化参数。*/
  aos_string_t bucket;
  aos_string_t object;
  aos_table_t *params = NULL;
  aos_http_request_t *req;
  char *url_str;
  apr_time_t now;
  int64_t expire_time;
  aos_str_set(&bucket, bucket_name);
  aos_str_set(&object, object_name);
  /* 将图片缩放为固定宽高100 px。*/
  params = aos_table_make(pool, 1);
  apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100,h_100");
  req = aos_http_request_create(pool);
  req->method = HTTP_GET;
  req->query_params = params;
  /* 指定过期时间（expire_time），单位为秒。*/
  now = apr_time_now();
  expire_time = now / 1000000 + 10 * 60;
  /* 生成签名url。*/
  url_str = oss_gen_signed_url(oss_client_options, &bucket, &object, expire_time, req);
  printf("url: %s\n", url_str);
  /* 释放该内存池，相当于释放了请求过程中各资源分配的内存。*/
  aos_pool_destroy(pool);
  /* 释放之前分配的全局资源。*/
  aos_http_io_deinitialize();
  return 0;
}
```

图片处理工具

您可以通过可视化图片处理工具ImageStyleViewer直观地看到OSS图片处理结果。

12.11. 错误处理

本文介绍OSS C SDK的错误处理。

异常处理

使用OSS C SDK时，如果请求出错，会在aos_status_s中输出相应的错误信息。aos_status_s有以下几种属性：

错误码	描述	字符类型
code	出错请求的HTTP状态码。	整形
error_code	OSS的错误码。	字符串
error_msg	OSS的错误信息。	字符串
req_id	标识该次请求的UUID，可以提供req_id来寻求OSS开发工程师的帮助。	字符串

超时处理

如果返回的aos_status_t中的code不等于2XX，且error_code为-992或者-995时，表示链接超时或请求超时，可以重试。

使用aos_status.h中的aos_should_retry(aos_status_t *)，判断返回的错误码是否需要重试。如果返回1，表示需要重试。

常见错误码

错误码	描述	HTTP 状态码
AccessDenied	拒绝访问	403
BucketAlreadyExists	存储空间已经存在	409
BucketNotEmpty	存储空间非空	409
EntityTooLarge	实体过大	400
EntityTooSmall	实体过小	400
FileGroupTooLarge	文件组过大	400
FilePartNotExist	文件分片不存在	400
FilePartStale	文件分片过时	400
InvalidArgument	参数格式错误	400
InvalidAccessKeyId	AccessKeyId不存在	403
InvalidBucketName	无效的存储空间名称	400
InvalidDigest	无效的摘要	400
InvalidObjectName	无效的文件名称	400

错误码	描述	HTTP 状态码
InvalidPart	无效的分片	400
InvalidPartOrder	无效的分片顺序	400
InvalidTargetBucketForLogging	Logging操作中有无效的目标存储空间	400
InternalError	OSS内部错误	500
MalformedXML	XML格式非法	400
MethodNotAllowed	不支持的方法	405
MissingArgument	缺少参数	411
MissingContentLength	缺少内容长度	411
NoSuchBucket	存储空间不存在	404
NoSuchKey	文件不存在	404
NoSuchUpload	分片上传ID不存在	404
NotImplemented	无法处理的方法	501
PreconditionFailed	预处理错误	412
RequestTimeTooSkewed	客户端本地时间和OSS服务器时间相差超过15分钟	403
RequestTimeout	请求超时	400
SignatureDoesNotMatch	签名错误	403
TooManyBuckets	用户的存储空间数超过限制	400

12.12. 常见问题

本文介绍使用 OSS C SDK 的常见问题与解决方法。

- OSS C SDK 在嵌入式环境下如何编译
- OSS C SDK 如何设置程序日志的输出
- OSS C SDK 如何设置通信时和 CURL 相关的一些参数
- OSS C SDK 利用 CURL 提供的 Callback 实现上传
- OSS C SDK 利用 CURL 提供的 Callback 实现数据下载
- Windows 版本 OSS C SDK 如何上传和下载包含中文名的文件

13.Ruby

13.1. 安装

本文介绍如何安装OSS Ruby SDK。

前提条件

- 已开通阿里云对象存储OSS服务

  如果您还没有开通或者还不了解OSS服务，请查看对象存储OSS产品详情页。

- 已创建AccessKey

  如果还没有创建AccessKey（AccessKey Id和AccessKey Secret），请参见。

安装

您可以通过以下两种方式安装Ruby SDK。

- 方式一

  执行 `gem install aliyun-sdk` 命令安装SDK。

  如果无法访问*https://rubygems.org*，建议直接使用淘宝的镜像源。

  ```
  gem install aliyun-sdk --clear-sources --source https://gems.ruby-china.org
  ```

- 方式二

  通过bundler安装。

  i. 在应用程序的 `Gemfile` 中添加 `gem 'aliyun-sdk', '~> 0.6.0'` 。

  ii. 选择淘宝镜像源进行安装。

  ```
  # 选择淘宝镜像源。
  bundle config mirror.https://rubygems.org https://ruby.taobao.org
  bundle install
  ```

  > **说明** https://ruby.taobao.org是完整的rubygems.org镜像，并与官方源自动同步。不方便访问rubygems.org的用户可以使用此镜像源。

依赖

- Ruby版本 >= 1.9.3
- 支持Ruby运行环境的Linux/Windows/OS X系统

> **注意**
>
> - SDK依赖的某些gem是本地扩展的形式，因此需要安装ruby-dev以支持编译本地扩展的gem。
> - SDK依赖处理XML的gem(nokogiri)环境中要求包含zlib库。

在Linux、Windows和OS X系统中安装依赖的步骤如下。

- Linux

以Ubuntu为例，执行如下命令安装上述依赖。

```
sudo apt-get install ruby ruby-dev zlib1g-dev
```

各个Linux发行版都有自身的包管理工具，且安装方法类似。

- Windows

  i. 前往RubyInstaller下载RubyInstaller，双击安装，在安装时选择**Add Ruby executables to your PATH**。

  > **说明** 请下载2.1及以下版本。现因某些原因还无法安装2.2版本，详情请参见GitHub。

  ii. 前往RubyInstaller下载DEVELOPMENT KIT，注意选择相应的版本。下载后是一个压缩包，在解压前首先在C盘根目录建立一个文件夹 `C:\RubyDev` ，然后将压缩包解压到此文件夹。

  iii. 同时按下 `Windows + R` 键，并输入 `cmd` 后回车进入到命令行窗口，输入以下命令。

  ```
  cd C:\RubyDev
  ruby dk.rb init
  ruby dk.rb install
  ```

  如果install时显示config.yml配置错误，建议用文本编辑器打开 `C:\RubyDev\config.yml` ，将其内容改为：

  ```
  ---
  - C:/Ruby21
  ```

  保存config.yml。然后继续使用命令 `ruby dk.rb install` 完成安装。其中，Ruby21是Ruby的安装目录，可根据具体的名字填写。安装完成后关闭此命令行窗口。

  iv. 输入命令 `gem install aliyun-sdk` 。

  安装完成后，输入 `irb` 进入Ruby交互式命令行。在交互式命令行中输入 `require 'aliyun/oss'` ，如果显示true，则表明SDK已完成安装。

- OS X

  OS X系统默认已安装Ruby。为了方便使用和管理，建议您再安装一个供开发使用的Ruby版本。

  i. 在终端输入 `xcode-select --install` 安装Xcode command line tools。 如果安装失败，建议手动下载并安装。

  > **说明** 使用您的Apple ID登录后从苹果开发者网站下载Xcode command line tools。注意选择与您的系统匹配的版本。下载完成后双击加载dmg文件，然后在打开的窗口中双击安装程序进行安装，在安装的过程中需要输入您的系统密码。

  ii. 在终端输入以下命令安装brew。

  ```
  ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
  ```

  iii. 在终端输入以下命令安装Ruby。

```
brew install ruby
exec $SHELL -l
```

iv. 在终端输入以下命令安装SDK。

```
gem install aliyun-sdk
```

v. 在终端输入以下命令验证SDK是否已安装成功。如果显式true，则表明SDK已完成安装。

```
irb
> require 'aliyun/oss'
=> true
```

相关文档

- github地址
- API文档地址
- ChangeLog

13.2. 快速入门

下面介绍如何使用OSS Ruby SDK来访问OSS服务，包括查看Bucket列表、查看文件列表、上传文件、下载文件和删除文件。

为了方便使用，下面的操作都是在Ruby的交互式命令行 `irb` 中进行。

初始化Client

在命令行中输入irb并回车。

在Ruby的交互式命令行模式下，输入 `require` 引入SDK包。

```
> require 'aliyun/oss'
=> true
```

> **说明** 在接下来的演示中，`>` 符号后面的内容是用户输入的命令，`=>` 后面的内容是程序返回的内容。

接下来创建Client：

```
> client = Aliyun::OSS::Client.new(
>   endpoint: 'endpoint',
>   access_key_id: 'AccessKeyId',
>   access_key_secret: 'AccessKeySecret')
=> #<Aliyun::OSS::Client...
```

将其中的参数替换成您实际的endpoint、AccessKeyId和AccessKeySecret。

查看Bucket列表

通过以下命令查看Bucket列表：

```
> buckets = client.list_buckets
=> #<Enumerator...
> buckets.each { |b| puts b.name }
=> bucket-1
=> bucket-2
=> ...
```

如果Bucket列表为空，则可以用以下命令创建一个Bucket：

```
> client.create_bucket('my-bucket')
=> true
```

说明

- Bucket的命名规范请查看OSS 基本概念。
- Bucket名字不能与OSS服务中其他用户已有的Bucket重复，所以你需要选择一个独特的Bucket名字以避免创建失败。

查看文件列表

通过以下命令查看Bucket中的文件列表：

```
> bucket = client.get_bucket('my-bucket')
=> #<Aliyun::OSS::Bucket...
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> object-1
=> object-2
=> ...
```

上传文件

通过以下命令向Bucket中上传一个文件：

```
> bucket.put_object('my-object', :file => 'local-file')
=> true
```

其中 `local-file` 是需要上传的本地文件的路径。上传成功后，可以通过 `list_objects` 来查看：

```
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> my-object
=> ...
```

下载文件

通过以下命令从Bucket中下载一个文件：

```
> bucket.get_object('my-object', :file => 'local-file')
=> #<Aliyun::OSS::Object...
```

其中 `local-file` 是文件保存的路径。下载成功后，可以打开文件查看其内容。

删除文件

通过以下命令从Bucket中删除一个文件：

```
> bucket.delete_object('my-object')
=> true
```

删除文件后可以通过 `list_objects` 来查看文件确实已经被删除：

```
> objects = bucket.list_objects
=> #<Enumerator...
> objects.each { |obj| puts obj.key }
=> object-1
=> ...
```

了解更多

- 存储空间
- 上传文件
- 下载文件
- 管理文件
- Rails应用
- 自定义域名绑定
- 使用STS访问
- 生命周期
- 设置访问日志
- 静态网站托管
- 防盗链
- 设置跨域资源共享

- 异常处理

13.3. 存储空间

13.3.1. 创建存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

使用 `Client#create_bucket` 接口创建存储空间时，需要指定存储空间名称：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
client.create_bucket('my-bucket')
```

> **说明**
> - 存储空间的命名规范请参见存储空间（Bucket）。
> - 存储空间的名称在OSS范围内必须是全局唯一的，一旦创建之后无法修改名称。
> - 获取endpoint信息，请参见访问域名和数据中心文档。
> - 创建存储的更多详情，请参见Put Bucket。

13.3.2. 列举存储空间

存储空间（Bucket）是用来存储对象（Object）的容器。对象都隶属于存储空间。存储空间按照字母顺序排列。您可以列举所有存储空间、列举资源组下的所有存储空间以及符合指定条件的存储空间。

使用 `Client#list_buckets` 接口列出当前用户下的所有Bucket，用户还可以指定 `:prefix` 参数，列出包含特定前缀的所有Bucket：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
buckets = client.list_buckets
buckets.each { |b| puts b.name }
buckets = client.list_buckets(:prefix => 'my-')
buckets.each { |b| puts b.name }
```

13.3.3. 判断存储空间是否存在

存储空间（Bucket）是存储对象（Object）的容器。对象均隶属于存储空间。本文介绍如何判断存储空间是否存在。

用户可以通过 `Client#bucket_exists?` 接口判断某个存储空间是否存在：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
puts client.bucket_exists?('my-bucket')
```

13.3.4. 管理存储空间访问权限

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何设置和获取存储空间访问权限（ACL）。

存储空间的访问权限（ACL）有以下三类：

访问权限	描述	访问权限值
私有	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户没有权限操作该存储空间内的文件。	Aliyun::OSS::ACL::PRIVATE
公共读	存储空间的拥有者和授权用户有该存储空间内的文件的读写权限，其他用户只有该存储空间内的文件的读权限。请谨慎使用该权限。	Aliyun::OSS::ACL::PUBLIC_READ
公共读写	所有用户都有该存储空间内的文件的读写权限。请谨慎使用该权限。	Aliyun::OSS::ACL::PUBLIC_READ_WRITE

设置存储空间访问权限

通过 `Bucket#acl=` 设置存储空间访问权限：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.acl = Aliyun::OSS::ACL::PUBLIC_READ
puts bucket.acl
```

设置存储空间访问权限的更多详情，请参见PutBucketAcl。

获取存储空间访问权限

通过 `Bucket#acl` 查看Bucket的ACL：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
puts bucket.acl
```

获取存储空间访问权限的更多详情，请参见GetBucketAcl。

13.3.5. 删除存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何删除存储空间。

> **说明**
> - 删除存储空间之前，必须先删除存储空间下的所有文件、LiveChannel和分片上传产生的碎片。
> - 要删除分片上传产生的碎片，首先使用list_upload列举出所有碎片，然后使用abort_upload删除这些碎片。

使用 `Client#delete_bucket` 接口删除一个Bucket，用户需要指定Bucket的名字：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
client.delete_bucket('my-bucket')
```

删除存储空间的更多详情，请参见DeleteBucket。

13.3.6. 生命周期

OSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储费用。本文介绍如何管理生命周期规则。

背景信息

每条生命周期规则包含以下内容：

- 生命周期规则ID。用于标识一条规则，同一存储空间内规则ID不能重复。
- 生命周期策略。有以下两种设置方式。同一存储空间内仅支持一种设置方式。
  - 按前缀匹配。此种方式允许创建多条规则，前缀不能重复。
  - 配置到整个存储空间。此种方式只能创建一条规则。
- 指定文件过期时间。有两种指定方式：
  - 指定一个过期天数N，文件会在其最近更新时间点的N天后过期。
  - 指定一个过期时间点，最近更新时间在该时间点之前的文件全部过期。
- 生命周期规则是否生效。

通过uploadPart方法上传的分片也支持设置生命周期规则。文件最后修改时间以初始化分片上传事件的时间为准。

更多关于生命周期的内容请参见管理对象生命周期。

设置生命周期规则

以下代码用于设置生命周期规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket.lifecycle = [
  LifeCycleRule.new(
    :id => 'rule1', :enabled => true, :prefix => 'foo/', :expiry => 3),
  LifeCycleRule.new(
    :id => 'rule2', :enabled => false, :prefix => 'bar/', :expiry => Date.new(2016, 1, 1))
]
```

查看生命周期规则

以下代码用于查看生命周期规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
rules = bucket.lifecycle
puts rules
```

清空生命周期规则

以下代码用于清空生命周期规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket.lifecycle = []
```

13.3.7. 防盗链

本文介绍如何使用防盗链。

为了防止您在OSS上的数据被其他人盗链而产生额外费用，您可以设置防盗链功能，包括以下参数：

- Referer白名单。仅允许指定的域名访问OSS资源。
- 是否允许空Referer。如果不允许空Referer，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。

更多关于防盗链的介绍，请参见开发指南中的设置防盗链。

设置Referer白名单

通过 `Bucket#referer=` 设置Referer白名单。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.referer = BucketReferer.new(
  allow_empty: true, whitelist: ['my-domain.com', '*.example.com'])
```

查看Referer白名单

通过 `Bucket#referer=` 查看Referer白名单。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
ref = bucket.referer
puts ref.to_s
```

清空Referer白名单

通过 `Bucket#referer=` 清空Referer白名单。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.referer = BucketReferer.new(allow_empty: true, whitelist: [])
```

13.3.8. 访问日志

您可以开启存储空间的访问日志记录功能，开启后对于此存储空间的访问会被记录成日志文件，保存在指定的存储空间中。

日志文件的格式为：

`<TargetPrefix><SourceBucket>YYYY-mm-DD-HH-MM-SS-UniqueString`

更多关于访问日志的介绍，请参见开发指南中的设置访问日志记录。

开启访问日志记录

通过 `Bucket#logging=` 来开启存储空间的访问日志记录。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.logging = BucketLogging.new(
  enable: true, target_bucket: 'logging_bucket', target_prefix: 'my-log')
```

查看访问日志设置

通过 `Bucket#logging` 来查看存储空间的访问日志设置。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
log = bucket.logging
puts log.to_s
```

关闭访问日志记录

通过 `Bucket#logging=` 来关闭存储空间的访问日志记录。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.logging = BucketLogging.new(enable: false)
```

13.3.9. 跨域资源共享

跨域资源共享（Cross-origin resource sharing，简称CORS）允许Web端的应用程序访问不属于本域的资源。OSS提供跨域资源共享接口，方便您控制跨域访问的权限。本文介绍如何进行跨域资源共享。

更多关于跨域资源共享的介绍，请参见开发指南中的设置跨域访问和API参考中PutBucketcors。

OSS的跨域共享设置由一条或多条CORS规则组成，每条CORS规则包含以下设置：

- allowed_origins，允许的跨域请求的来源，如www.my-domain.com, *
- allowed_methods，允许的跨域请求的HTTP方法（PUT、POST、GET、DELETE、HEAD）
- allowed_headers，在OPTIONS预取指令中允许的header，如x-oss-test, *
- expose_headers，允许用户从应用程序中访问的响应头
- max_age_seconds，浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间

设置CORS规则

通过 `bucket.cors` 设置CORS规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.cors = [
  CORSRule.new(
    :allowed_origins => ['aliyun.com', 'http://www.taobao.com'],
    :allowed_methods => ['PUT', 'POST', 'GET'],
    :allowed_headers => ['Authorization'],
    :expose_headers => ['x-oss-test'],
    :max_age_seconds => 100)
]
```

查看CORS规则

通过 `bucket.cors` 查看CORS规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
cors = bucket.cors
puts cors.map(&:to_s)
```

清空CORS规则

通过 `bucket.cors` 清空CORS规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.cors = []
```

13.3.10. 静态网站托管

您可以将存储空间配置成静态网站托管模式。配置生效后，访问网站相当于访问存储空间，并且能够自动跳转至指定的索引页面和错误页面。

更多关于静态网站托管的介绍，请参见开发指南中的配置静态网站托管。

设置静态网站托管

通过 `Bucket#website=` 来设置静态网站托管：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.website = BucketWebsite.new(index: 'index.html', error: 'error.html')
```

查看静态网站托管配置

通过 `Bucket#website` 来查看静态网站托管配置：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
web = bucket.website
puts web.to_s
```

删除静态网站托管配置

通过 `Bucket#website=` 来删除静态网站托管配置：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.website = BucketWebsite.new(enable: false)
```

13.3.11. 自定义域名绑定

本文介绍如何使用自定义域名绑定。

OSS支持用户将自定义的域名绑定到OSS服务上，这样能够支持用户无缝地将存储迁移到OSS上。例如用户的域名是my-domain.com，之前用户的所有图片资源都是形如 `http://img.my-domain.com/x.jpg` 的格式，用户将图片存储迁移到OSS之后，通过绑定自定义域名，仍可以使用原来的地址访问到图片：

1. 开通OSS服务并创建Bucket。
2. 将img.my-domain.com与创建的Bucket绑定。
3. 将图片上传到OSS的这个Bucket中。
4. 修改域名的DNS配置，增加一个CNAME记录，将img.my-domain.com指向OSS服务的endpoint（如my-bucket.oss-cn-hangzhou.aliyuncs.com）。

这样就可以通过原地址 `http://img.my-domain.com/x.jpg` 访问到存储在OSS上的图片。 绑定自定义域名请参考自定义域名绑定。

在使用SDK时，也可以使用自定义域名作为endpoint，这时需要将 `:cname` 参数设置为true，如下面的例子：

```
require 'aliyun/oss'
include Aliyun::OSS
client = Client.new(
  endpoint: 'ENDPOINT',
  access_key_id: 'ACCESS_KEY_ID',
  access_key_secret: 'ACCESS_KEY_SECRET',
  cname: true)
bucket = client.get_bucket('my-bucket')
```

说明 自定义域名已经绑定到某个特定的Bucket，因此使用CNAME时无法使用list_buckets接口。

13.4. 上传文件

在OSS中，操作的基本数据单元是文件（Object）。OSS Ruby SDK提供了丰富的文件上传方式。

上传本地文件

通过 `Bucket#put_object` 接口，并指定 `:file` 参数来上传一个本地文件到OSS。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.put_object('my-object', :file => 'local-file')
```

流式上传

如果要上传的文件太大，或者一次性上传耗时太长，您可以通过流式上传的方式，一次处理部分内容，直到完成整个文件的上传。

通过 `Bucket#put_object` 接口，并指定 `block` 参数将流式生成的内容上传到OSS：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.put_object('my-object') do |stream|
  100.times { |i| stream << i.to_s }
end
```

断点续传上传

断点续传上传将要上传的文件分成若干个分片（Part）分别上传，所有分片都上传完成后，将所有分片合并成完整的文件，完成整个文件的上传。

通过 `Bucket#resumable_upload` 接口实现断点续传上传，包含以下参数：

- key上传到OSS的Object名字。
- file待上传的本地文件路径。
- opts可选项，主要包括：
  - :cpt_file指定checkpoint文件的路径。如果不指定，则默认是本地文件同目录下的 `file.cpt` ，其中 `file` 是本地文件的名字。
  - :disable_cpt如果指定为true，则上传过程中不会记录上传进度，上传失败后也无法进行续传。
  - :part_size指定每个分片的大小，默认为4MB。
  - &block如果调用时传递了block，则上传进度会由block来处理。

详细的参数说明请参考API文档。

> **说明** 如果上传过程中某一分片上传失败，再次上传时会从 checkpoint文件中记录的点继续上传。再次调用 `Bucket#resumable_upload` 时需指定与上次相同的checkpoint文件。文件上传完成后，checkpoint文件将被删除。

以下代码用于断点续传上传：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.resumable_upload('my-object', 'local-file') do |p|
 puts "Progress: #{p}"
end
bucket.resumable_upload(
 'my-object', 'local-file',
 :part_size => 100 * 1024, :cpt_file => '/tmp/x.cpt') { |p|
 puts "Progress: #{p}"
}
```

> **说明**
>
> - SDK会将上传的中间状态信息记录在cpt文件中，用户需确保对cpt文件有写权限。
> - cpt文件不仅记录了上传的中间状态信息，还附带校验功能。cpt文件不允许编辑。如果cpt文件损坏则无法继续上传，整个文件上传完成后cpt文件会被删除。
> - 上传过程中如果本地文件发生了改变，会导致上传失败。

追加上传

通过 `Bucket#append_object` 来上传可追加的文件。调用该接口时需要指定文件追加的position，首次追加操作的position必须为0，后续追加操作的position是Object的当前大小。

- 文件不存在时，调用 `append_object` 会创建一个可追加的文件。

- 文件存在时，调用 `append_object` 会向文件末尾追加内容。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
# 创建可追加的文件
bucket.append_object('my-object', 0) {}
# 向文件末尾追加内容
next_pos = bucket.append_object('my-object', 0) do |stream|
 100.times { |i| stream << i.to_s }
end
next_pos = bucket.append_object('my-object', next_pos, :file => 'local-file-1')
next_pos = bucket.append_object('my-object', next_pos, :file => 'local-file-2')
```

> **说明**
> - 只能向Appendable类型的Object追加内容。
> - 不支持对Appendable类型的Object进行拷贝。

上传回调

用户在上传文件时可以指定上传回调。文件上传成功后，OSS会向用户提供的服务器地址发起HTTP POST请求，用户可以在收到回调时执行相应的操作。更多有关上传回调的内容，请参考 OSS 上传回调。

> **说明** 目前OSS支持上传回调的接口只有 `put_object` 和 `resumable_upload` 。有关上传回调的详细示例，请参考callback.rb。

使用 `put_object` 上传文件时指定了上传回调，并将此次上传的Bucket和Object信息添加在body中。如果应用服务器收到这个回调，则表明文件已成功上传到OSS了。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
callback = Aliyun::OSS::Callback.new(
 url: 'http://10.101.168.94:1234/callback',
 query: {user: 'put_object'},
 body: 'bucket=${bucket}&object=${object}'
)
begin
 bucket.put_object('files/hello', file: '/tmp/x', callback: callback)
rescue Aliyun::OSS::CallbackError => e
 puts "Callback failed: #{e.message}"
end
```

`resumable_upload` 的调用方法与 `put_object` 类似。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
callback = Aliyun::OSS::Callback.new(
  url: 'http://10.101.168.94:1234/callback',
  query: {user: 'put_object'},
  body: 'bucket=${bucket}&object=${object}'
)
begin
  bucket.resumable_upload('files/hello', '/tmp/x', callback: callback)
rescue Aliyun::OSS::CallbackError => e
  puts "Callback failed: #{e.message}"
end
```

说明

- callback的URL不能包含query string，须在 `:query` 参数中指定。
- 如果出现文件上传成功，但是执行回调失败时，client会抛出 `CallbackError` 。用户如果忽略此错误，需要显式接住这个异常。
- 接收回调的server请参见callback_server.rb。

13.5. 下载文件

本文档介绍OSS Ruby SDK提供的多种文件下载方式。

下载到本地文件

通过 `Bucket#get_object` 接口，并指定 `:file` 参数将指定的OSS文件下载到本地文件：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.get_object('my-object', :file => 'local-file')
```

流式下载

如果要下载的文件太大，或者一次性下载耗时太长，您可以通过流式下载的方式，一次处理部分内容，直到完成整个文件的下载。

通过 `Bucket#get_object` 接口，并指定 `block` 参数来流式处理下载的内容：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.get_object('my-object') do |chunk|
 # handle_data(chunk)
 puts "Got a chunk, size: #{chunk.size}."
end
```

断点续传下载

当下载大文件时，如果网络不稳定或者程序异常退出，会导致下载失败，甚至重试多次仍无法完成下载。为此OSS提供了断点续传下载功能。

断点续传下载将需要下载的文件分成若干个分片分别下载，所有分片都下载完成后，将所有分片合并成完整的文件。

通过Bucket#resumable_download接口来实现断点续传下载，包含以下参数：

- key要下载的文件名称。
- file下载到本地文件的路径。
- opts可选项，主要包括：
  - :cpt_file指定checkpoint文件的路径，如果不指定，则默认是本地文件同目录下的 `file.cpt` ，其中 `file` 是本地文件的名字。
  - :disable_cpt如果指定为true，则下载过程中不会记录下载进度，失败后也无法进行续传。
  - :part_size指定每个分片的大小，默认为10MB。
  - &block如果调用时候传递了block，则下载进度会交由block处理。

详细的参数请参考API文档。

在下载文件的过程中会记录当前下载的进度信息，并记录在checkpoint文件中。已下载的分片将保存为 `file.part.N` ，其中 `file` 表示下载的本地文件的名称。如果下载过程中某一分片下载失败，再次下载时会从checkpoint文件中记录的点继续下载。再次调用Bucket#resumable_download时要指定与上次相同的checkpoint文件。下载完成后，part文件和checkpoint文件都会被删除。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.resumable_download('my-object', 'local-file') do |p|
 puts "Progress: #{p}"
end
bucket.resumable_download(
 'my-object', 'local-file',
 :part_size => 100 * 1024, :cpt_file => '/tmp/x.cpt') { |p|
 puts "Progress: #{p}"
}
```

说明

- SDK会将下载的中间状态信息记录在cpt文件中，您需确保对cpt文件有写权限。
- SDK会将已下载的分片保存在part文件中，您需确保对 `file` 所在的目录有创建文件的权限。
- cpt文件不仅记录了下载的中间状态信息，还附带校验功能。cpt文件不允许编辑。如果cpt文件损坏则无法继续下载，整个文件下载完成后cpt文件会被删除。
- 下载过程中待下载的文件、ETag值发生改变，part文件丢失或被修改均会导致下载失败。

HTTP下载

在不用SDK的情况下，您也可以直接使用HTTP下载存放在OSS中的文件。HTTP下载包括直接使用浏览器下载，或者使用 `wget` `curl` 等命令行工具下载，此时文件的URL需要由SDK生成。

使用 `Bucket#object_url` 方法生成可下载的HTTP地址， 包含以下参数：

- key待下载的文件名称。
- sign是否生成带签名的URL。

  说明

  - 对于拥有公共读（public-read）或者公共读写（public-read-write）权限的文件，允许通过不带签名的URL进行访问。
  - 对于私有（private）权限的文件，则必须通过带签名的URL进行访问。

- expiresURL的有效时间，默认为60s。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
# 生成URL，默认带签名，有效时间为60秒
puts bucket.object_url('my-object')
# http://my-bucket.oss-cn-hangzhou.aliyuncs.com/my-object?Expires=1448349966&OSSAccessKeyId=5vxxxx
&Signature=aM2HpBLeMq1aec6JCd7BBAKYiwI%3D
# 不带签名的URL
puts bucket.object_url('my-object', false)
# http://my-bucket.oss-cn-hangzhou.aliyuncs.com/my-object
# 指定URL过期时间为1小时（3600秒）
puts bucket.object_url('my-object', true, 3600)
```

13.6. 管理文件

OSS Ruby SDK提供一系列的接口方便用户管理某个存储空间（Bucket）下的多个文件（Object）。

列举所有文件

通过 `bucket.list_objects` 接口来列举指定Bucket下的所有文件，包括以下主要参数：

- :prefix指定只列出符合特定前缀的文件。

- :marker指定只列出文件名大于marker之后的文件。
- :delimiter用于获取文件的公共前缀。

以下代码用于列举指定Bucket下符合条件的文件：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
# 列出所有文件
objects = bucket.list_objects
objects.each { |o| puts o.key }
# 列出前缀为'my-'的所有文件
objects = bucket.list_objects(:prefix => 'my-')
objects.each { |o| puts o.key }
# 列出前缀为'my-'且在'my-object'之后的所有文件
objects = bucket.list_objects(:prefix => 'my-', :marker => 'my-object')
objects.each { |o| puts o.key }
```

模拟目录结构

OSS是基于对象的存储服务，没有目录的概念。存储在某个Bucket中的所有文件都是通过文件的key来唯一标识，并没有层级的结构。通过OSS提供的公共前缀的功能，您可以方便地模拟目录结构，将文件分门别类地存放在不同的目录下。公共前缀的概念请参考查看对象列表。

假设某个Bucket中有如下文件：

```
foo/x
foo/y
foo/bar/a
foo/bar/b
foo/hello/C/1
foo/hello/C/2
...
foo/hello/C/9999
```

然后通过调用 `list_dir` 函数来列举指定目录下的文件和子目录：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
def list_dir(dir)
  objects = bucket.list_objects(:prefix => dir, :delimiter => '/')
  objects.each do |obj|
    if obj.is_a?(OSS::Object) # object
      puts "Object: #{obj.key}"
    else # common prefix
      puts "SubDir: #{obj}"
    end
  end
end
```

运行结果如下：

```
> list_dir('foo/')
=> SubDir: foo/bar/
   SubDir: foo/hello/
   Object: foo/x
   Object: foo/y
> list_dir('foo/bar/')
=> Object: foo/bar/a
   Object: foo/bar/b
> list_dir('foo/hello/C/')
=> Object: foo/hello/C/1
   Object: foo/hello/C/2
   ...
   Object: foo/hello/C/9999
```

文件元信息

向OSS上传文件时，除了文件内容，还可以指定文件的一些属性信息，这些属性信息称为元信息。 元信息在上传时与文件一起存储，在下载时与文件一起返回。

在Ruby SDK中，文件元信息用 `Hash` 来表示，其他key和value均为 `String` 类型。文件元信息会附在HTTP Headers中， 而HTTP协议规定请求头部不能包含复杂字符，所以元信息只能是简单的ASCII可见字符且不能包含换行。

> **说明** 所有元信息的总大小不能超过8KB。

使用 `bucket.put_object` 、 `bucket.append_object` 和 `bucket.resumable_upload` 时都可以通过指定 `:metas` 参数来指定文件的元信息。

以下代码用于指定文件元信息：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.put_object(
 'my-object-1',
 :file => 'local-file',
 :metas => {'year' => '2016', 'people' => 'mary'})
bucket.append_object(
 'my-object-2', 0,
 :file => 'local-file',
 :metas => {'year' => '2016', 'people' => 'mary'})
bucket.resumable_upload(
 'my-object',
 'local-file',
 :metas => {'year' => '2016', 'people' => 'mary'})
```

通过 `bucket.update_object_metas` 命令来更新文件元信息：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.update_object_metas('my-object', {'year' => '2017'})
```

拷贝文件

使用 `bucket.copy_object` 拷贝一个文件，并通过 `:meta_directive` 参数指定文件元信息。拷贝时对文件元信息的处理有两种选择：

- 如果没有指定meta参数，则与源文件相同，即拷贝源文件的元信息。
- 如果指定了meta参数，则使用新的元信息覆盖源文件的信息。

以下代码用于拷贝文件：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
# 拷贝文件元信息。
bucket.copy_object(
  'my-object', 'copy-object',
  :meta_directive => Aliyun::OSS::MetaDirective::COPY)
# 覆盖文件元信息。
bucket.copy_object(
  'my-object', 'copy-object',
  :metas => {'year' => '2017'},
  :meta_directive => Aliyun::OSS::MetaDirective::REPLACE)
```

删除文件

通过 `bucket.delete_object` 来删除某个文件：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
bucket.delete_object('my-object')
```

批量删除文件

通过 `bucket.batch_delete_objects` 批量删除文件，并通过 `:quiet` 参数来指定是否返回删除的结果：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
objs = ['my-object-1', 'my-object-2']
result = bucket.batch_delete_objects(objs)
# 默认返回删除成功的文件。
puts result #['my-object-1', 'my-object-2']
objs = ['my-object-3', 'my-object-4']
result = bucket.batch_delete_objects(objs, :quiet => true)
# 不返回删除的结果。
puts result #[]
```

设置文件访问权限

对于Object，有四种访问权限：

- default：继承所属的Bucket的访问权限，即与所属Bucket的访问权限一样。

- public-read-write：允许匿名用户读写该Object。
- public-read：允许匿名用户读该Object。
- private：不允许匿名访问，所有的访问都要经过签名。

创建Object时，默认为default权限。之后您可以通过 `bucket.set_object_acl` 来设置Object的权限。

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
 endpoint: 'endpoint',
 access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket = client.get_bucket('my-bucket')
acl = bucket.get_object_acl('my-object')
puts acl # default
bucket.set_object_acl('my-object', Aliyun::OSS::ACL::PUBLIC_READ)
acl = bucket.get_object_acl('my-object')
puts acl # public-read
```

- 如果没有设置Object的权限，即Object的ACL为继承Bucket，此时Object的权限和Bucket权限一致。
- 如果Object的权限为继承Bucket以外的三种权限时，访问该Object进行权限认证时会优先判断Object的权限，此时Bucket的权限设置会被忽略。
- 允许匿名访问时（设置了public-read或者public-read-write权限），您可以直接通过浏览器访问，例如 `http://bucket-name.oss-cn-hangzhou.aliyuncs.com/object.jpg` 。
- 您也可以通过创建匿名的Client来访问具有public-read或者public-read-write权限的Object。

```
require 'aliyun/oss'
# 不填access_key_id和access_key_secret，将创建匿名Client，只能访问具有public权限的Object。
client = Aliyun::OSS::Client.new(endpoint: 'endpoint')
bucket.get_object('my-object', :file => 'local_file')
```

13.7. 使用STS进行临时授权

OSS 可以通过阿里云 STS (Security Token Service) 进行临时授权访问。

OSS可以通过阿里云STS (Security Token Service) 进行临时授权访问。阿里云STS是为云计算用户提供临时访问令牌的Web服务。通过STS，您可以为第三方应用或子用户（即用户身份由您自己管理的用户）颁发一个自定义时效和权限的访问凭证。关于STS的更多信息，请参见STS介绍。

STS的优势如下：

- 您无需透露您的长期密钥（AccessKey）给第三方应用，只需生成一个访问令牌并将令牌交给第三方应用。您可以自定义这个令牌的访问权限及有效期限。
- 您无需关心权限撤销问题，访问令牌过期后自动失效。

使用STS访问OSS时，需要设置:sts_token参数：

```
require 'aliyun/sts'
require 'aliyun/oss'
sts = Aliyun::STS::Client.new(
  access_key_id: '<子账号的AccessKeyId>',
  access_key_secret: '<子账号的AccessKeySecret>')
token = sts.assume_role('<role-arn>', '<session-name>')
client = Aliyun::OSS::Client.new(
  endpoint: '<endpoint>',
  access_key_id: token.access_key_id,
  access_key_secret: token.access_key_secret,
  sts_token: token.security_token)
bucket = client.get_bucket('my-bucket')
```

```
require 'aliyun/sts'
require 'aliyun/oss'
sts = Aliyun::STS::Client.new(
  access_key_id: '<子账号的AccessKeyId>',
  access_key_secret: '<子账号的AccessKeySecret>')
policy = Aliyun::STS::Policy.new
policy.allow(['oss:Get*'], ['acs:oss:*:*:my-bucket/*'])
token = sts.assume_role('<role arc>', '<session name>', policy, 15 * 60)
client = Aliyun::OSS::Client.new(
  endpoint: 'ENDPOINT',
  access_key_id: token.access_key_id,
  access_key_secret: token.access_key_secret,
  sts_token: token.security_token)
bucket = client.get_bucket('my-bucket')
```

更详细的用法和参数说明请参考API文档。

13.8. 异常处理

使用SDK时如果请求出错，会有相应的异常抛出，同时在log（默认为程序运行目录下oss_sdk.log）中也会记录详细的出错信息。

OSS Ruby SDK中有ClientError和ServerError两种异常，它们都是RuntimeError的子类。

ClientError

ClientError指SDK内部出现的异常，比如参数设置错误、断点续传上传或断点续传下载过程中出现的文件被修改的错误。

ServerError

ServerError指服务器端错误，它来自于对服务器错误信息的解析。ServerError 有以下几个属性：

- http_code：出错请求的HTTP状态码
- error_code：OSS的错误码
- message：OSS的错误信息
- request_id：标识该次请求的UUID；当您无法解决问题时，可以凭这个RequestId来请求OSS开发工程师的

帮助

OSS中常见的错误信息请参考OSS错误响应。

13.9. Rails应用

本文主要介绍如何在Rails应用中使用OSS Ruby SDK来列举存储空间（Bucket）、上传文件（Object）、下载文件等。

准备工作

在Rails应用中使用OSS Ruby SDK，需要在 `Gemfile` 中添加以下依赖。

```
gem 'aliyun-sdk', '~> 0.3.0
```

然后在使用OSS时引入以下依赖。

```
require 'aliyun/oss'
```

与Rails集成

OSS Ruby SDK与Rails集成的流程如下。

1. 创建项目
   i. 安装Rails，然后创建一个oss-manager的Rails应用。

   ```
   gem install rails
   rails new oss-manager
   ```

   > **说明** 本示例中的oss-manager是OSS的文件管理器，包含列举所有Bucket、按目录层级列举Bucket下所有文件、上传文件和下载文件等功能。

   ii. 使用Git管理项目代码。

   ```
   cd oss-manager
   git init
   git add .
   git commit -m "init project"
   ```

2. 添加SDK依赖
   i. 编辑 `oss-manager/Gemfile` ，并加入SDK的依赖。

   ```
   gem 'aliyun-sdk', '~> 0.3.0'
   ```

   ii. 在 `oss-manager/` 中执行 `bundle install` 。

iii. 保存更改。

```
git add .
git commit -m "add aliyun-sdk dependency"
```

初始化OSS Client

为了避免在项目中用到OSS Client时都需要初始化，建议在项目中添加一个初始化文件。

```
# oss-manager/config/initializers/aliyun_oss_init.rb
require 'aliyun/oss'
module OSS
 def self.client
  unless @client
   Aliyun::Common::Logging.set_log_file('./log/oss_sdk.log')
   @client = Aliyun::OSS::Client.new(
    endpoint:
     Rails.application.secrets.aliyun_oss['endpoint'],
    access_key_id:
     Rails.application.secrets.aliyun_oss['access_key_id'],
    access_key_secret:
     Rails.application.secrets.aliyun_oss['access_key_secret']
   )
  end
  @client
 end
end
```

以上代码可以在SDK的rails/目录下找到。初始化后，可以在项目中方便地使用OSS Client。

```
buckets = OSS.client.list_buckets
```

其中endpoint、access_key_id和access_key_secret都保存在 `oss-manager/conf/secrets.yml` 中。

```
development:
 secret_key_base: xxxx
 aliyun_oss:
  endpoint: xxxx
  access_key_id: aaaa
  access_key_secret: bbbb
```

保存更改。

```
git add .
git commit -m "add aliyun-sdk initializer"
```

列举所有的Bucket

您可以按照如下步骤列举所有的Bucket。

1. 用Rails生成管理Bucket的controller。

```
rails g controller buckets index
```

2. 在 `oss-manager` 中生成以下文件。
   - app/controller/buckets_controller.rb Buckets相关的逻辑代码
   - app/views/buckets/index.html.erb Buckets相关的展示代码
   - app/helpers/buckets_helper.rb辅助函数
3. 编辑buckets_controller.rb，调用OSS Client将 `list_buckets` 的结果存放在 `@buckets` 变量中。

```
class BucketsController < ApplicationController
 def index
  @buckets = OSS.client.list_buckets
 end
end
```

4. 编辑views/buckets/index.html.erb，列举Bucket列表。

```
<h1>Buckets</h1>
<table class="table table-striped">
 <tr>
  <th>Name</th>
  <th>Location</th>
  <th>CreationTime</th>
 </tr>
 <% @buckets.each do |bucket| %>
 <tr>
  <td><%= link_to bucket.name, bucket_objects_path(bucket.name) %></td>
  <td><%= bucket.location %></td>
  <td><%= bucket.creation_time.localtime.to_s %></td>
 </tr>
 <% end %>
</table>
```

5. 在app/helpers/buckets_helper.rb中定义辅助函数 `bucket_objects_path` 。

```
module BucketsHelper
 def bucket_objects_path(bucket_name)
  "/buckets/#{bucket_name}/objects"
 end
end
```

以上步骤完成后即可列出所有的Bucket。

此外，您需要配置Rails的路由，使得在浏览器中输入的地址能够调用正确的逻辑。编辑 `config/routes.rb` 时需增加如下内容。

```
resources :buckets do
 resources :objects
end
```

在 `oss-manager/` 下输入 `rails s` 以启动Rails服务器，然后在浏览器中输入 `http://localhost:3000/buckets/` 即可查看Bucket列表。

6. 保存更改。

```
git add .
git commit -m "add list buckets feature"
```

按目录层级列举Bucket下的所有文件

按目录层级列举Bucket下的所有文件的步骤如下。

1. 生成一个管理Object的controller。

```
rails g controller objects index
```

2. 编辑app/controllers/objects_controller.rb。

```
class ObjectsController < ApplicationController
 def index
  @bucket_name = params[:bucket_id]
  @prefix = params[:prefix]
  @bucket = OSS.client.get_bucket(@bucket_name)
  @objects = @bucket.list_objects(:prefix => @prefix, :delimiter => '/')
 end
end
```

您可以从URL的参数中获取Bucket名字。您需要添加一个前缀来实现只按目录层级列出Object，然后调用OSS Client的list_objects接口获取文件列表。

> **说明** 这里获取的是指定前缀下并且以正斜线（/）为分界的文件。具体操作，请参见管理文件。

3. 编辑app/views/objects/index.html.erb。

```
<h1>Objects in <%= @bucket_name %></h1>
<p> <%= link_to 'Upload file', new_object_path(@bucket_name, @prefix) %></p>
<table class="table table-striped">
 <tr>
  <th>Key</th>
  <th>Type</th>
  <th>Size</th>
  <th>LastModified</th>
 </tr>
 <tr>
  <td><%= link_to '../', with_prefix(upper_dir(@prefix)) %></td>
  <td>Directory</td>
  <td>N/A</td>
  <td>N/A</td>
 </tr>
 <% @objects.each do |object| %>
 <tr>
  <% if object.is_a?(Aliyun::OSS::Object) %>
  <td><%= link_to remove_prefix(object.key, @prefix),
       @bucket.object_url(object.key) %></td>
  <td><%= object.type %></td>
  <td><%= number_to_human_size(object.size) %></td>
  <td><%= object.last_modified.localtime.to_s %></td>
  <% else  %>
  <td><%= link_to remove_prefix(object, @prefix), with_prefix(object) %></td>
  <td>Directory</td>
  <td>N/A</td>
  <td>N/A</td>
  <% end  %>
 </tr>
 <% end %>
</table>
```

将文件按目录结构显示的实现逻辑是：

i. 总是在第一个出现'../'时，指向上级目录。

ii. 对于Common prefix，显示为目录。

iii. 对于Object，显示为文件。

上面的代码中使用了 `with_prefix` 、 `remove_prefix` 等辅助函数，这些辅助函数定义在app/helpers/objects_helper.rb中。

```
module ObjectsHelper
 def with_prefix(prefix)
  "?prefix=#{prefix}"
 end
 def remove_prefix(key, prefix)
  key.sub(/^#{prefix}/, '')
 end
 def upper_dir(dir)
  dir.sub(/[^\/]+\/$/, '') if dir
 end
 def new_object_path(bucket_name, prefix = nil)
  "/buckets/#{bucket_name}/objects/new/#{with_prefix(prefix)}"
 end
 def objects_path(bucket_name, prefix = nil)
  "/buckets/#{bucket_name}/objects/#{with_prefix(prefix)}"
 end
end
```

4. 完成之后运行 `rails s` ，然后在浏览器中输入地址 `http://localhost:3000/buckets/my-bucket/objects/` 即可查看文件列表。
5. 保存更改。

```
git add .
git commit -m "add list objects feature"
```

下载文件

在上述显示文件列表步骤中已为每个文件添加了URL，您可以直接使用该文件URL来下载文件。

```
<td><%= link_to remove_prefix(object.key, @prefix),
    @bucket.object_url(object.key) %></td>
```

您还可以使用 `bucket.object_url` 为文件生成临时的URL。具体操作，请参见下载文件。

上传文件

在Rails服务端应用中，您可以通过以下两种方式上传文件。

- 第一种上传方式与普通上传文件类似。您需要先将文件上传到Rails服务器，服务器再将文件上传到OSS。
- 第二种上传方式是使用服务器生成的表单和临时凭证将文件直接上传到OSS。
  i. 在app/controllers/objects_controller.rb中增加 `#new` 方法，用于生成上传表单。

```
def new
 @bucket_name = params[:bucket_id]
 @prefix = params[:prefix]
 @bucket = OSS.client.get_bucket(@bucket_name)
 @options = {
  :prefix => @prefix,
  :redirect => 'http://localhost:3000/buckets/'
 }
end
```

ii. 编辑app/views/objects/new.html.erb。

```
<h2>Upload object</h2>
<%= upload_form(@bucket, @options) do %>
 <table class="table table-striped">
  <tr>
   <td><label>Bucket:</label></td>
   <td><%= @bucket.name %></td>
  </tr>
  <tr>
   <td><label>Prefix:</label></td>
   <td><%= @prefix %></td>
  </tr>
  <tr>
   <td><label>Select file:</label></td>
   <td><input type="file" name="file" style="display:inline" /></td>
  </tr>
  <tr>
   <td colspan="2">
    <input type="submit" class="btn btn-default" value="Upload" />
    <span>  </span>
    <%= link_to 'Back', objects_path(@bucket_name, @prefix) %>
   </td>
  </tr>
 </table>
<% end %>
```

说明 `upload_form` 是SDK提供的一个便于生成上传表单的辅助函数，该辅助函数存放在rails/aliyun_oss_helper.rb中。您需要将其拷贝到app/helpers/目录下，完成后运行 `rails s` ，然后通过访问 `http://localhost:3000/buckets/my-bucket/objects/new` 即可将文件上传到OSS。

iii. 保存更改。

```
git add .
git commit -m "add upload object feature"
```

添加样式

您可以按照如下步骤为界面添加样式。

1. 下载bootstrap，解压后将 `bootstrap.min.css` 拷贝到 `app/assets/stylesheets/` 下。
2. 修改 `app/views/layouts/application.html.erb` ，并将 `yield` 这一行改成。

```
<div id="main">
 <%= yield %>
</div>
```

3. 为每个页面添加一个ID为main的 `<div>` ，然后修改 `app/assets/stylesheets/application.css` ，并添加以下内容。

```
body {
  text-align: center;
}
div#main {
  text-align: left;
  width: 1024px;
  margin: 0 auto;
}
```

完整的demo代码请参见OSS Rails Demo。