阿里云
对象存储 OSS

SDK 参考

文档版本：20191125

为了无法计算的价值 | 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. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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
	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。	禁止： 重置操作将丢失用户配置数据。
	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。
	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。
	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。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
##	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

法律声明 I
通用约定 I
1 SDK 文档简介 1
2 Java 2
2.1 前言 2
2.2 安装 3
2.3 快速入门 5
2.4 初始化 8
2.5 存储空间 12
2.6 请求者付费模式 17
2.7 单链接限速 19
2.8 存储空间标签 21
2.9 对象标签 23
2.9.1 设置对象标签 23
2.9.2 获取对象标签 32
2.9.3 删除对象标签 32
2.9.4 对象标签和生命周期管理 33
2.10 服务器端加密 35
2.11 上传文件 37
2.11.1 概述 38
2.11.2 简单上传 38
2.11.3 表单上传 41
2.11.4 追加上传 44
2.11.5 断点续传上传 45
2.11.6 分片上传 46
2.11.7 进度条 54
2.11.8 上传回调 56
2.12 下载文件 57
2.12.1 概述 57
2.12.2 流式下载 57
2.12.3 下载到本地文件 58
2.12.4 范围下载 59
2.12.5 断点续传下载 60
2.12.6 限定条件下载 61
2.12.7 进度条 62
2.13 管理文件 64
2.13.1 概述 64
2.13.2 判断文件是否存在 64
2.13.3 管理文件访问权限 65
2.13.4 管理文件元信息 66

2.13.5 转换文件存储类型 69
2.13.6 列举文件 71
2.13.7 查询文件 81
2.13.8 删除文件 84
2.13.9 拷贝文件 86
2.13.10 解冻归档文件 91
2.13.11 管理软链接 92
2.14 数据安全性 93
2.15 授权访问 95
2.16 生命周期 99
2.17 跨域资源共享 102
2.18 访问日志 104
2.19 静态网站托管 106
2.20 防盗链 107
2.21 跨区域复制 109
2.22 图片处理 111
2.23 异常处理 118
2.24 常见问题 120
3 Python 134
3.1 前言 134
3.2 安装 135
3.3 快速入门 138
3.4 初始化 140
3.5 存储空间 142
3.6 请求者付费模式 145
3.7 单链接限速 147
3.8 存储空间标签 148
3.9 对象标签 150
3.9.1 设置对象标签 150
3.9.2 获取对象标签 157
3.9.3 删除对象标签 157
3.9.4 对象标签和生命周期管理 158
3.10 上传文件 160
3.10.1 概述 160
3.10.2 简单上传 160
3.10.3 追加上传 163
3.10.4 断点续传上传 163
3.10.5 分片上传 164
3.10.6 进度条 166
3.10.7 上传回调 167
3.11 下载文件 167
3.11.1 概述 167
3.11.2 流式下载 168
3.11.3 下载到本地文件 169
3.11.4 范围下载 169

3.11.5 断点续传下载.....169
3.11.6 进度条.....171
3.12 管理文件.....171
3.12.1 概述.....171
3.12.2 判断文件是否存在.....172
3.12.3 管理文件访问权限.....172
3.12.4 管理文件元信息.....173
3.12.5 转换文件存储类型.....175
3.12.6 列举文件.....177
3.12.7 查询文件.....178
3.12.8 删除文件.....184
3.12.9 拷贝文件.....185
3.12.10 解冻归档文件.....186
3.12.11 管理软链接.....187
3.13 授权访问.....188
3.14 开启Python SDK日志.....189
3.15 数据加密.....191
3.15.1 客户端加密.....191
3.15.2 服务器端加密.....194
3.16 静态网站托管.....196
3.17 生命周期.....197
3.18 跨域资源共享.....199
3.19 访问日志.....200
3.20 防盗链.....202
3.21 图片处理.....203
3.22 中文和时间.....210
3.23 异常处理.....212
3.24 常见问题.....214
4 Go.....217
4.1 前言.....217
4.2 安装.....218
4.3 快速入门.....219
4.4 初始化.....223
4.5 存储空间.....225
4.6 请求者付费模式.....232
4.7 单链接限速.....235
4.8 存储空间标签.....238
4.9 服务器端加密.....241
4.10 对象标签.....243
4.10.1 设置对象标签.....244
4.10.2 获取对象标签.....252
4.10.3 删除对象标签.....253
4.10.4 对象标签和生命周期管理.....254
4.11 上传文件.....255
4.11.1 概述.....255

4.11.2 简单上传 256
4.11.3 追加上传 259
4.11.4 断点续传上传 260
4.11.5 分片上传 262
4.11.6 进度条 267
4.12 下载文件 269
4.12.1 概述 269
4.12.2 流式下载 269
4.12.3 范围下载 271
4.12.4 下载到本地文件 272
4.12.5 断点续传下载 273
4.12.6 限定条件下载 275
4.12.7 文件压缩下载 276
4.12.8 进度条 277
4.13 管理文件 278
4.13.1 概述 278
4.13.2 判断文件是否存在 278
4.13.3 管理文件访问权限 279
4.13.4 管理文件元信息 280
4.13.5 转换文件存储类型 283
4.13.6 列举文件 285
4.13.7 删除文件 293
4.13.8 拷贝文件 296
4.13.9 解冻归档文件 300
4.13.10 管理软链接 302
4.14 授权访问 304
4.15 生命周期 307
4.16 访问日志 309
4.17 静态网站托管 311
4.18 防盗链 313
4.19 跨域资源共享 315
4.20 图片处理 317
4.21 错误处理 325
5 C++ 329
5.1 前言 329
5.2 安装 330
5.3 快速入门 332
5.4 初始化 336
5.5 存储空间 340
5.6 请求者付费模式 346
5.7 服务器端加密 349
5.8 存储空间标签 351
5.9 单链接限速 355
5.10 对象标签 357
5.10.1 设置对象标签 357

5.10.2 获取对象标签.....366
5.10.3 删除对象标签.....367
5.10.4 对象标签和生命周期管理.....368
5.11 上传文件.....370
5.11.1 概述.....370
5.11.2 简单上传.....371
5.11.3 追加上传.....373
5.11.4 断点续传上传.....374
5.11.5 分片上传.....376
5.11.6 中断上传.....383
5.11.7 进度条.....384
5.11.8 上传回调.....385
5.12 下载文件.....386
5.12.1 概述.....386
5.12.2 下载到本地文件.....386
5.12.3 下载到本地内存.....387
5.12.4 范围下载.....388
5.12.5 断点续传下载.....389
5.12.6 进度条.....390
5.13 管理文件.....391
5.13.1 概述.....391
5.13.2 管理文件访问权限.....392
5.13.3 管理文件元信息.....394
5.13.4 转换文件存储类型.....396
5.13.5 列举文件.....398
5.13.6 删除文件.....411
5.13.7 拷贝文件.....413
5.13.8 判断文件是否存在.....416
5.13.9 解冻归档文件.....417
5.13.10 管理软链接.....418
5.14 授权访问.....420
5.15 生命周期.....423
5.16 跨域资源共享.....426
5.17 访问日志.....429
5.18 静态网站托管.....431
5.19 防盗链.....433
5.20 图片处理.....436
6 PHP.....444
6.1 前言.....444
6.2 安装.....444
6.3 快速入门.....446
6.4 初始化.....450
6.5 存储空间.....453
6.6 上传文件.....459
6.6.1 概述.....459

6.6.2 简单上传 459
6.6.3 追加上传 460
6.6.4 分片上传 462
6.6.5 上传回调 469
6.7 下载文件 472
6.7.1 概述 472
6.7.2 下载到本地文件 472
6.7.3 下载到本地内存 473
6.7.4 范围下载 474
6.7.5 限定条件下载 474
6.8 管理文件 476
6.8.1 概述 476
6.8.2 判断文件是否存在 476
6.8.3 管理文件访问权限 477
6.8.4 管理文件元信息 479
6.8.5 列举文件 481
6.8.6 删除文件 483
6.8.7 拷贝文件 485
6.8.8 解冻归档文件 487
6.8.9 管理软链接 488
6.8.10 开启MD5校验 489
6.9 授权访问 490
6.10 静态网站托管 493
6.11 生命周期 495
6.12 访问日志 498
6.13 跨域资源共享 500
6.14 防盗链 502
6.15 自定义域名绑定 505
6.16 图片处理 507
6.17 异常处理 513
7 C 516
7.1 前言 516
7.2 安装 517
7.3 快速入门 521
7.4 初始化 527
7.5 存储空间 530
7.6 上传文件 539
7.6.1 概述 539
7.6.2 简单上传 539
7.6.3 追加上传 543
7.6.4 断点续传上传 546
7.6.5 分片上传 548
7.6.6 进度条 552
7.6.7 上传回调 554
7.7 下载文件 555

7.7.1 概述 555
7.7.2 下载到本地文件 556
7.7.3 下载到本地内存 557
7.7.4 范围下载 558
7.7.5 断点续传下载 560
7.7.6 进度条 562
7.8 管理文件 563
7.8.1 概述 563
7.8.2 管理文件访问权限 564
7.8.3 管理文件元信息 565
7.8.4 列举文件 567
7.8.5 删除文件 581
7.8.6 拷贝文件 585
7.8.7 解冻归档文件 589
7.8.8 管理软链接 591
7.9 授权访问 593
7.10 生命周期 598
7.11 图片处理 602
7.12 跨域资源共享 611
7.13 访问日志 615
7.14 防盗链 618
7.15 静态网站托管 622
7.16 错误处理 625
7.17 常见问题 627
8 .NET 628
8.1 前言 628
8.2 安装 629
8.3 快速入门 631
8.4 初始化 634
8.5 存储空间 637
8.6 上传文件 640
8.6.1 概述 640
8.6.2 简单上传 641
8.6.3 追加上传 644
8.6.4 断点续传上传 645
8.6.5 分片上传 646
8.6.6 进度条 653
8.6.7 上传回调 654
8.7 下载文件 656
8.7.1 概述 656
8.7.2 流式下载 656
8.7.3 范围下载 657
8.7.4 断点续传下载 658
8.7.5 进度条 659
8.8 管理文件 660

8.8.1 概述 660
8.8.2 判断文件是否存在 660
8.8.3 管理文件访问权限 661
8.8.4 管理文件元信息 662
8.8.5 列举文件 664
8.8.6 删除文件 668
8.8.7 拷贝文件 670
8.8.8 解冻归档文件 673
8.8.9 管理软链接 675
8.9 授权访问 675
8.10 生命周期 678
8.11 访问日志 680
8.12 跨域资源共享 682
8.13 静态网站托管 684
8.14 防盗链 686
8.15 图片处理 688
8.16 异常处理 695
9 Android 698
9.1 前言 698
9.2 安装 698
9.3 快速入门 700
9.4 初始化 705
9.5 存储空间 709
9.6 上传文件 712
9.6.1 概述 712
9.6.2 简单上传 712
9.6.3 追加上传 714
9.6.4 分片上传 715
9.6.5 断点续传上传 719
9.6.6 进度条 722
9.6.7 上传回调 723
9.7 下载文件 724
9.7.1 概述 724
9.7.2 流式下载 724
9.7.3 范围下载 726
9.7.4 下载到本地文件 727
9.7.5 限定条件下载 727
9.7.6 断点续传下载 728
9.8 管理文件 730
9.8.1 概述 730
9.8.2 判断文件是否存在 730
9.8.3 拷贝文件 731
9.8.4 列举文件 732
9.8.5 删除文件 734
9.8.6 设置Content-Type 734

9.8.7 获取文件元信息 735
9.8.8 解冻归档文件 736
9.8.9 管理软链接 737
9.9 数据安全性 738
9.10 生命周期 739
9.11 防盗链 741
9.12 签名URL 744
9.13 授权访问 744
9.14 访问日志 748
9.15 图片处理 750
9.16 异常响应 752
10 iOS 754
10.1 前言 754
10.2 安装 754
10.3 快速入门 756
10.4 初始化 759
10.5 存储空间 762
10.6 上传文件 764
10.6.1 概述 764
10.6.2 简单上传 765
10.6.3 分片上传 766
10.6.4 追加上传 770
10.6.5 断点续传上传 771
10.6.6 上传回调 773
10.7 下载文件 774
10.7.1 概述 774
10.7.2 简单下载 774
10.7.3 流式下载 775
10.7.4 范围下载 775
10.7.5 断点续传下载 776
10.8 管理文件 785
10.8.1 概述 785
10.8.2 判断文件是否存在 785
10.8.3 获取文件元信息 786
10.8.4 列举文件 786
10.8.5 拷贝文件 787
10.8.6 删除文件 788
10.9 数据安全性 788
10.10 签名URL 790
10.11 授权访问 791
10.12 图片处理 794
10.13 异常响应 796
11 Node.js 798
11.1 安装 798

11.2 配置项 799
11.3 快速入门 800
11.4 初始化 802
11.5 存储空间 803
11.6 上传文件 805
11.6.1 概述 805
11.6.2 上传本地文件 806
11.6.3 上传本地内存 806
11.6.4 流式上传 807
11.6.5 分片上传 808
11.6.6 追加上传 810
11.6.7 断点续传上传 811
11.6.8 上传回调 812
11.7 下载文件 813
11.7.1 概述 813
11.7.2 下载到本地文件 813
11.7.3 下载到本地内存 813
11.7.4 流式下载 814
11.7.5 HTTP 下载 814
11.8 管理文件 815
11.8.1 概述 815
11.8.2 判断文件是否存在 815
11.8.3 管理文件元信息 816
11.8.4 管理文件访问权限 817
11.8.5 列举指定Bucket下的所有文件 818
11.8.6 列举指定目录下的文件和子目录 819
11.8.7 拷贝文件 820
11.8.8 删除文件 821
11.8.9 解冻归档文件 822
11.8.10 管理软链接 823
11.9 自定义域名绑定 824
11.10 跨域资源共享 825
11.11 授权访问 826
11.12 访问权限 829
11.13 生命周期 830
11.14 访问日志 832
11.15 静态网站托管 833
11.16 防盗链 835
11.17 图片处理 836
11.18 异常处理 842
11.19 常见问题 843
12 Browser.js 845
12.1 安装 845
12.2 快速开始 848
12.3 配置项 854

12.4 上传文件.....855
12.5 下载文件.....860
12.6 管理文件.....861
12.7 自定义域名绑定.....865
12.8 授权访问.....866
12.9 访问权限.....868
12.10 图片处理.....869
12.11 浏览器应用.....875
12.12 异常处理.....879
12.13 常见问题.....879
12.14 日志收集.....883
13 Ruby.....884
13.1 安装.....884
13.2 快速入门.....887
13.3 存储空间.....890
13.4 上传文件.....892
13.5 下载文件.....896
13.6 管理文件.....898
13.7 自定义域名绑定.....902
13.8 使用STS进行临时授权.....903
13.9 访问权限.....904
13.10 生命周期.....906
13.11 访问日志.....907
13.12 静态网站托管.....908
13.13 防盗链.....909
13.14 跨域资源共享.....910
13.15 异常处理.....911
13.16 Rails应用.....912
14 Media-C.....920
14.1 前言.....920
14.2 安装.....921
14.3 初始化.....923
14.4 客户端.....924
14.5 服务端.....931
14.6 HLS基础接口.....940
14.7 HLS封装接口.....945
14.8 使用场景.....948
14.9 常见问题.....951
15 conref.....953

1 SDK 文档简介

本文介绍对象存储OSS 包含的主流语言SDK。

使用OSS SDK之前，您需要：

- 了解并开通阿里云OSS服务。
- 创建AccessKey。

本文档详细介绍了以下主流语言SDK的安装、开发操作、参数、示例和常见问题处理。

语言	参考文档
Java	Java SDK参考
Python	Python 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
React Native	React Native SDK参考
PHP	PHP SDK参考
C	C SDK参考
Ruby	Ruby SDK参考
Go	Go SDK参考
Media-C	Media-C SDK参考

2 Java

2.1 前言

本文档基于OSS Java SDK 3.8.0编写。

兼容性

- **对于3.×.×系列SDK：**
  - **接口： 兼容。**
  - **命名空间： 兼容。**
- **对于2.×.×系列SDK：**
  - **接口： 兼容。**
  - **命名空间： 兼容。**
- **对于1.0.× 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Table Store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**

示例文件	示例内容
CRCSample.java	上传时进行CRC校验
SimpleGetObjectSample.java	下载文件
DownloadSample.java	断点续传下载
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	PostObject，该实现不依赖于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

2.2 安装

本文介绍如何安装Java SDK。

环境准备

- 环境要求

  使用Java 1.8及以上版本。

- 查看版本

  执行命令`java -version`查看Java版本。

下载SDK

- 直接下载
- 通过*GitHub*下载
- 历史版本下载

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 3.8.0 版本为例，在 <dependencies> 内加入如下内容：

```
<dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>3.8.0</version>
</dependency>
```

- **方式二：在Eclipse项目中导入JAR包**

  以 3.8.0 版本为例，步骤如下：

  1. **下载***Java SDK* **开发包**。
  2. **解压该开发包**。
  3. **将解压后文件夹中的文件** *aliyun-sdk-oss-3.8.0.jar* **以及 lib 文件夹下的所有文件拷贝到您的项目中**。
  4. **在 Eclipse 中选择您的工程，右击选择 Properties > Java Build Path > Add JARs**。
  5. **选中您在第3步拷贝的所有JAR文件，导入到Libraries中**。

- **方式三：在Intellij IDEA项目中导入JAR包**

  以 3.8.0 版本为例，步骤如下：

  1. **下载***Java SDK* **开发包**。
  2. **解压该开发包**。
  3. **将解压后文件夹中的文件** *aliyun-sdk-oss-3.8.0.jar* **以及lib文件夹下的所有JAR文件拷贝到您的项目中**。
  4. **在 Intellij IDEA中选择您的工程，选择File > Project Structure > Modules > Dependencies > + > JARs or directories** 。
  5. **选中您在第3步拷贝的所有JAR文件，导入到External Libraries中**。

2.3 快速入门

本节介绍如何快速使用OSS Java SDK完成常见操作，如创建存储空间（Bucket）、上传/下载文件（Object）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，也可以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。 以下代码用于新建一个存储空间：

```
// 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>";
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，请参见**基本概念**中的命名规范。创建存储空间详情，请参见*#unique_41*。

获取endpoint信息，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 = "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>";
// <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);

// 上传内容到指定的存储空间（bucketName）并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

以下代码用于获取文件的文本内容：

```
// 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>";
// <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 = "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);

// 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();
```

列举功能详情请参见管理文件中的列举文件。

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 = "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>";
// <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管理文件中的删除文件。

2.4 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 = "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);

// 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 https://ram.console.aliyun.com 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();
// 开启支持CNAME。CNAME是指将自定义域名绑定到存储空间上。
```

```
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/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 = "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地址做作为Endpoint，请按照实际IP地址
填写。
String endpoint = "http://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 https://ram.console.aliyun.com 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及以前的版本需要设置此
值，OSS Java SDK 2.1.2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新建一个OSSClient：

```
// 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 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*RAM和STS介绍*和授权访问。

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
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

参数	描述	方法
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/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

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 = "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/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>");

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret, conf);

// 关闭OSSClient。
ossClient.shutdown();
```

更多详情请参见阿里云*OSS Java SDK*超时时间设置。

2.5 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。

创建存储空间

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 = "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);

// 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以及数据容灾类型，可以参考以下代码。
// 此处以设置存储空间的存储类型为标准存储为例。
// createBucketRequest.setStorageClass(StorageClass.Standard);
// 默认情况下，数据容灾类型为本地冗余存储，即DataRedundancyType.LRS。如果需要设
置数据容灾类型为同城冗余存储，请替换为DataRedundancyType.ZRS。
// createBucketRequest.setDataRedundancyType(DataRedundancyType.ZRS)

// 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#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb。

您可以在创建存储空间时指定存储空间的权限和存储类型。

如果需要创建低频或归档类型的存储空间，请使用Java SDK 2.6.0及以上版本。示例代码如下：

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest(
bucketName);
// 设置存储空间的权限为公共读，默认是私有。
createBucketRequest.setCannedACL(CannedAccessControlList.PublicRead);
// 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。
createBucketRequest.setStorageClass(StorageClass.IA);
ossClient.createBucket(createBucketRequest);
```

列举存储空间

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。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。

- 列举所有的存储空间

  以下代码用于列举所有的存储空间：

```
// 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);

// 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();
```

- 列举指定前缀的存储空间

  以下代码用于列举包含指定前缀（prefix）的存储空间：

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定前缀的存储空间。
listBucketsRequest.setPrefix("<yourBucketPrefix>");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
```

```
for (Bucket bucket : bucketList.getBucketList()) {
    System.out.println(" - " + bucket.getName());
}
```

- **列举指定marker之后的存储空间**

  参数marker代表存储空间名称。以下代码用于列举指定marker之后的存储空间：

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 列举指定marker之后的存储空间。
listBucketsRequest.setMarker("<yourBucketMarker>");
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
    System.out.println(" - " + bucket.getName());
}
```

- **列举指定个数的存储空间**

  以下代码用于列举指定个数（maxKeys）的存储空间：

```
ListBucketsRequest listBucketsRequest = new ListBucketsRequest();
// 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。
listBucketsRequest.setMaxKeys(500);
BucketList bucketList = ossClient.listBuckets(listBucketsRequest);
for (Bucket bucket : bucketList.getBucketList()) {
    System.out.println(" - " + bucket.getName());
}
```

判断存储空间是否存在

以下代码用于判断指定的存储空间是否存在：

```
// 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);

boolean exists = ossClient.doesBucketExist("<yourBucketName>");

// 关闭OSSClient。
ossClient.shutdown();
```

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。	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();
```

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：

```
// 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();
```

获取存储空间的地域

以下代码用于获取存储空间的地域（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();
```

关于地域的详细信息请参见基本概念中的地域。

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）：

```
// 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);

// 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。
BucketInfo info = ossClient.getBucketInfo("<yourBucketName>");
// 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();
// 获取权限信息。
info.getGrants();
// 获取容灾类型。
info.getDataRedundancyType();
```

```
// 关闭OSSClient。
ossClient.shutdown();
```

删除存储空间

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。

> **说明:**
>
> 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();
```

2.6 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，请求方必须在其请求中包含x-oss-request-payer: requester（在PUT、POST、GET和HEAD请求的Head信息中），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#unique_55。

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.getBucketRequestPaym
ent(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 ByteArrayInputStream(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();
```

2.7 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#unique_56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, HttpMethod.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();
```

2.8 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ListBucket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();
```

2.9 对象标签

2.9.1 设置对象标签

本文介绍如何设置对象标签（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#unique_60。

说明：

- **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，若对象已有标签，则覆盖原标签。设置对象标签的详情请参考#unique_61。**
- **设置对象标签时要求请求者有PutObjectTagging权限。**

- 更改标签时#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：

  +-=._:/

上传Object时添加对象标签

- 普通上传时添加对象标签

  以下代码用于普通上传时添加对象标签：

```
// 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>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

Map<String, String> tags = new HashMap<String, String>();
tags.put("key1", "value1");
tags.put("key2", "value2");

// 在http header中设置标签信息。
ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags);

// 上传文件，同时给文件设置标签。
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：

```
// 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 localFile = "<yourLocalFile>";

// 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>();
tags.put("key1", "value1");
tags.put("key2", "value2");

ObjectMetadata metadata = new ObjectMetadata();
metadata.setObjectTagging(tags);

// 发起InitiateMultipartUploadRequest请求，同时设置文件标签信息。
InitiateMultipartUploadRequest request = new InitiateMultipartUpl
oadRequest(bucketName, objectName);
InitiateMultipartUploadResult result = ossClient.initiateMu
ltipartUpload(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 = 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;   // 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());
}

/* 步骤3：完成分片上传。
*/
// 排序。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：

```
// 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);

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>();
tags.put("key1", "value1");
tags.put("key2", "value2");

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, 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。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);

// 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：

```
// 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);

// 设置文件的标签信息。
Map<String, String> tags = new HashMap<String, String>();
tags.put("key1", "value1");
tags.put("key2", "value2");

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("<
yourBucketName>","<yourObjectName>");

// 通过UploadFileRequest设置单个参数。
// 设置存储空间名称。
//uploadFileRequest.setBucketName("<yourBucketName>");
// 设置文件名称。
//uploadFileRequest.setKey("<yourObjectName>");
// 指定上传的本地文件。
uploadFileRequest.setUploadFile("<yourLocalFile>");
// 指定上传并发线程数，默认为1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小，范围为100KB~5GB，默认为文件大小/10000。
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。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();
```

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：

```
// 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);

Map<String, String> tags = new HashMap<String, String>();
tags.put("key1", "value1");
tags.put("key2", "value2");
```

```
// 给文件设置标签。
ossClient.setObjectTagging(bucketName, objectName, tags);

// 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1GB以下的Object、及分片拷贝1GB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：

```
// 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 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, sourceObjectName, 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, destinatio
nObjectName);
Map<String, String> getTags = tagSet.getAllTags();
System.out.println("dest object tagging: "+ getTags.toString());

// 关闭OSSClient。
```

```
ossClient.shutdown();
```

- 以下代码用于分片拷贝1GB 以上的的Object时设置对象标签：

```
// 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>";

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);

// 在http header中设置标签信息。
Map<String, String> tags2 = new HashMap<String, String>();
tags2.put("key3", "value3");
tags2.put("key4", "value4");

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(destinationBucketName, destinatio
nObjectName, metadata);
InitiateMultipartUploadResult initiateMultipartUploadResult =
ossClient.initiateMultipartUpload(initiateMultipartUploadRequest);
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, destinationObjectName);
    uploadPartCopyRequest.setUploadId(uploadId);
    uploadPartCopyRequest.setPartSize(size);
    uploadPartCopyRequest.setBeginIndex(skipBytes);
```

```
    uploadPartCopyRequest.setPartNumber(i + 1);

    UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPart
Copy(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
CompleteMultipartUploadRequest(
                    destinationBucketName, destinationObjectName,
uploadId, partETags);
CompleteMultipartUploadResult completeMultipartUploadResult =
ossClient.completeMultipartUpload(completeMultipartUploadRequest);

// 查看源文件的标签信息。
TagSet tagSet = ossClient.getObjectTagging(bucketName, sourceObje
ctName);
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, destinationObjectNam
e);
getTags = tagSet.getAllTags();
System.out.println("dest object tagging: "+ getTags.toString());

// 关闭OSSClient。
ossClient.shutdown();
```

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：

```
// 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);

// 设置软链接的标签信息。
Map<String, String> tags = new HashMap<String, String>();
tags.put("key1", "value1");
tags.put("key2", "value2");

// 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, destinationObjectName);
```

```
// 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();
```

2.9.2 获取对象标签

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。对象标签的详情请参考开发指南的#unique_60。

以下代码用于获取对象标签：

```
// 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>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 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();
```

获取对象标签详情请参考#unique_63。

2.9.3 删除对象标签

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。对象标签的详情请参考开发指南的#unique_60。

以下代码用于删除对象标签：

```
// 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>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 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);
```

删除对象标签详情请参考#unique_65。

2.9.4 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：**
>
> **Tag条件中，标签的Key和Value必须同时匹配。同一个规则中，若同时配置了前缀和多个对象标签，则对象需满足前缀，且同时匹配规则中所有对象标签，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.getExpirat
ionDays());
    }

    //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().getCreatedBeforeDate());
        }
    }
}

// 关闭OSSClient。
ossClient.shutdown();
```

2.10 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。

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。

> **注意:**
> - 同一对象同一时间仅可以使用一种服务器器端加密方式。
> - 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*PutObject*。
> - 更多关于服务器端加密的介绍请参见*#unique_70*。

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 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加密。
ServerSideEncryptionByDefault applyServerSideEncryptionByDefault = new
 ServerSideEncryptionByDefault(SSEAlgorithm.KMS);
applyServerSideEncryptionByDefault.setKMSMasterKeyID("<yourTestKmsId
>");
ServerSideEncryptionConfiguration sseConfig = new ServerSideEncryption
Configuration();
sseConfig.setApplyServerSideEncryptionByDefault(applyServerSideEncry
ptionByDefault);
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 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加密配置。
ServerSideEncryptionConfiguration sseConfig = ossClient.getBucketE
ncryption("<yourBucketName>");
System.out.println("get Algorithm: " + sseConfig.getApplyServerSideEn
cryptionByDefault().getSSEAlgorithm());
System.out.println("get kmsid: " + sseConfig.getApplyServerSideEn
cryptionByDefault().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 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加密配置。
ossClient.deleteBucketEncryption("<yourBucketName>");

// 关闭OSSClient。
ossClient.shutdown();
```

2.11 上传文件

2.11.1 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。

2.11.2 简单上传

简单上传指的是使用OSS API中的PutObject方法上传单个文件（Object），本文主要介绍通过Java SDK进行简单上传。

简单上传分为流式上传和文件上传。流式上传使用InputStream作为文件的数据源。文件上传使用本地文件作为OSS文件的数据源。

> 说明：
> 简单上传的完整代码请参见*GitHub*。

流式上传

使用ossClient.putObject上传数据流到OSS。

- 上传字符串

  以下代码用于上传字符串：

```
// 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);

// 创建PutObjectRequest对象。
String content = "Hello OSS";
// <yourObjectName>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.jpg。
PutObjectRequest putObjectRequest = new PutObjectRequest("<
yourBucketName>", "<yourObjectName>", new ByteArrayInputStream(
content.getBytes()));

// 如果需要上传时设置存储类型与访问权限，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**

  以下代码用于上传Byte数组：

```
// 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);

// 上传Byte数组。
byte[] content = "Hello OSS".getBytes();
ossClient.putObject("<yourBucketName>", "<yourObjectName>", new
ByteArrayInputStream(content));

// 关闭OSSClient。
ossClient.shutdown();
```

- **上传网络流**

  以下代码用于上传网络流：

```
// 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);

// 上传网络流。
InputStream inputStream = new URL("https://www.aliyun.com/").
openStream();
ossClient.putObject("<yourBucketName>", "<yourObjectName>",
inputStream);

// 关闭OSSClient。
ossClient.shutdown();
```

- **上传文件流**

  以下代码用于上传文件流：

```
// 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";
// 云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用RAM
子账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 上传文件流。
InputStream inputStream = new FileInputStream("<yourlocalFile>");
ossClient.putObject("<yourBucketName>", "<yourObjectName>",
inputStream);

// 关闭OSSClient。
ossClient.shutdown();
```

文件上传

以下代码用于上传本地文件：

```
// 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>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 创建PutObjectRequest对象。
PutObjectRequest putObjectRequest = new PutObjectRequest("<yourBucket
Name>", "<yourObjectName>", new File("<yourLocalFile>"));

// 如果需要上传时设置存储类型与访问权限，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();
```

2.11.3 表单上传

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

表单上传的适用场景介绍请参见开发指南中的表单上传。

PostObject更多详情请参见*Java模拟PostObject表单上传OSS*。

以下代码用于表单上传：

```
public class PostObjectSample {
    // 上传文件
    private String localFilePath = "<yourLocalFile>";
    // 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 https://ram.console.aliyun.com 创建
RAM账号。
    private String accessKeyId = "<yourAccessKeyId>";
    private String accessKeySecret = "<yourAccessKeySecret>";
    // 存储空间名称
    private String bucketName = "<yourBucketName>";
    // 文件名称
    private String objectName = "<yourObjectName>";
    /**
     * 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://" +
bucketName+ ".");
        // 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="
                + localFilePath);
        // 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
必须为小写。
        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.ServiceSig
nature.create().computeSignature(accessKeySecret, encodePolicy);
        // 设置签名。
        formFields.put("Signature", signaturecom);
        String ret = formUpload(urlStr, formFields, localFilePath);
        System.out.println("Post Object [" + this.objectName + "] to
bucket [" + bucketName + "]");
        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值是由整个body计算得出的。
            conn.setRequestProperty("Content-MD5", "<yourContentMD5
>");
            conn.setRequestProperty("Content-Type",
                    "multipart/form-data; boundary=" + boundary);
            OutputStream out = new DataOutputStream(conn.getOutputS
tream());
            // 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=\""
                                + 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().getContent
Type(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 FileInputS
tream(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 InputStrea
mReader(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();
    }
}
```

说明:

更多回调详情请参见*Callback*。

2.11.4 追加上传

本文介绍如何使用追加上传。

追加类型的文件（Append Object）暂时不支持copyObject操作。

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 = "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 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();
```

追加上传详情请参见开发指南中的**追加上传**。追加上传的完整代码请参见*GitHub*。

2.11.5 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。

断点续传详情请参见开发指南中的**断点续传**。完整代码请参见*GitHub*。

以下代码用于断点续传上传：

```
// 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 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);

ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");

// 通过UploadFileRequest设置多个参数。
UploadFileRequest uploadFileRequest = new UploadFileRequest("<
yourBucketName>","<yourObjectName>");

// 通过UploadFileRequest设置单个参数。
// 设置存储空间名称。
//uploadFileRequest.setBucketName("<yourBucketName>");
// 设置文件名称。
//uploadFileRequest.setKey("<yourObjectName>");
// 指定上传的本地文件。
uploadFileRequest.setUploadFile("<yourLocalFile>");
// 指定上传并发线程数，默认为1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小，范围为100KB~5GB，默认为文件大小/10000。
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。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。
ossClient.uploadFile(uploadFileRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

2.11.6 分片上传

本文介绍如何使用分片上传。

分片上传的完整代码请参见*GitHub*。

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，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ossClient.completeMultipartUpload方法将所有分片合并成完整的文件。

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：

```
// 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>";
// <yourObjectName>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.jpg。
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 创建InitiateMultipartUploadRequest对象。
InitiateMultipartUploadRequest request = new InitiateMultipartUpl
oadRequest("<yourBucketName>", "<yourObjectName>");

// 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.initiateMu
ltipartUpload(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。
// 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(completeMultipartUploadRequest);

// 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：

```
// 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);

// 取消分片上传，其中uploadId来自于InitiateMultipartUpload。
AbortMultipartUploadRequest abortMultipartUploadRequest =
        new AbortMultipartUploadRequest("<yourBucketName>", "<
yourObjectName>", "<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：

```
// 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);

// 列举已上传的分片，其中uploadId来自于InitiateMultipartUpload返回的结果。
ListPartsRequest listPartsRequest = new ListPartsRequest("<
yourBucketName>", "<yourObjectName>", "<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。只有分片号大于该参数值的分片会被列出。
 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。

```
// 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);

// 列举所有已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest("<
yourBucketName>", "<yourObjectName>", "<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，只有分片号大于该参数值的分片会被列出。
listPartsRequest.setPartNumberMarker(partListing.getNextPar
tNumberMarker());
} 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。

```
// 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);

// 分页列举已上传的分片。
PartListing partListing;
ListPartsRequest listPartsRequest = new ListPartsRequest("<
yourBucketName>", "<yourObjectName>", "<uploadId>");
// 设置分页时每一页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.getNextPar
tNumberMarker());

} 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。可设置的参数如下：

参数	作用	如何设置
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。	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)
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)

参数	作用	如何设置
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)

· 简单列举分片上传事件

以下代码用于列举分片上传事件：

```
// 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>";

// 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.listMultip
artUploads(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。如果没有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。

```
// 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);

// 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(multipartU
ploadListing.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：

```
// 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);

// 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数目。
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(multipartU
ploadListing.getNextUploadIdMarker());

} while (multipartUploadListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

2.11.7 进度条

进度条用于指示上传或下载的进度。

下面的代码以ossClient.putObject方法为例，介绍如何使用进度条。

```
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.bytesWritt
en + " bytes have been transferred in total");
            break;
        case TRANSFER_FAILED_EVENT:
            System.out.println("Failed to upload, " + this.bytesWritt
en + " 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 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);

        try {
            // 带进度条的上传。
```

```
            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和ossClient.uploadPart方法支持进度条功能。

ossClient.uploadFile和ossClient.downloadFile方法不支持进度条功能。

上传进度条的完整代码请参见*GitHub*。

2.11.8 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）：

```
// 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>";
// 您的回调服务器地址，如http://oss-demo.aliyuncs.com:23450或http://127.0.
0.1:9090。
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 ByteArrayInputStream(content.getBytes()));

// 上传回调参数。
Callback callback = new Callback();
callback.setCallbackUrl(callbackUrl);
// 设置回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
callback.setCallbackHost("oss-cn-hangzhou.aliyuncs.com");
// 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();
```

上传回调详情请参见开发指南中的上传回调。

2.12 下载文件

2.12.1 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。

2.12.2 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。

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：

```
// 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);

// 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*。

2.12.3 下载到本地文件

以下代码用于把指定的OSS文件下载到本地文件：

```
// 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>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 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。
ossClient.getObject(new GetObjectRequest(bucketName, objectName), new
File("<yourLocalFile>"));

// 关闭OSSClient。
ossClient.shutdown();
```

2.12.4 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 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 getObjectRequest = new GetObjectRequest(bucketName,
objectName);
// 获取0~1000字节范围内的数据，包括0和1000，共1001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, 1000);

// 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64KB的数据，请使用如下的方式多次读取，直到读取到64KB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();
```

2.12.5 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。
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1B~5GB。	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：

```
// 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);

// 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(downloadFi
leRequest);
// 下载成功时，会返回文件元信息。
downloadRes.getObjectMetadata();

// 关闭OSSClient。
ossClient.shutdown();
```

2.12.6 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）。

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：

```
// 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。

2.12.7 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*GitHub*。

下面的代码以ossClient.getObject方法为例，介绍如何使用进度条。

```
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;
            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 https://ram.console.aliyun.com 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 {
            // 带进度条的下载。
            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和ossClient.uploadPart方法支持进度条功能。ossClient.uploadFile和ossClient.downloadFile方法不支持进度条功能。

2.13 管理文件

2.13.1 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

2.13.2 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断指定的文件是否存在：

```
// 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);

// 判断文件是否存在。doesObjectExist还有一个参数isOnlyInOSS，如果为true则忽略
302重定向或镜像；如果
为false，则考虑302重定向或镜像。
```

```
boolean found = ossClient.doesObjectExist("<yourBucketName>", "<
yourObjectName>");
System.out.println(found);

// 关闭OSSClient。
ossClient.shutdown();
```

2.13.3 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

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();
```

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, accessKeyS
ecret);

// 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();
```

2.13.4 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

- **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 https://ram.console.aliyun.com 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，则会被截断，为设置的长度。如不足，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 https://ram.console.aliyun.com 创建RAM账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

String content = "Hello OSS";

// 创建OSSClient实例。
OSSClient ossClient = new OSSClient(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8KB。

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 https://ram.console.aliyun.com 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。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId,accessKeyS
ecret);

// 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, destinationBucketName, 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 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, accessKeyS
ecret);

// 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();
```

2.13.5 转换文件存储类型

OSS提供标准、低频访问、归档三种存储类型，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。

有关标准存储、低频访问、归档存储三种存储类型的更多信息，请参考开发指南的#unique_51及#unique_90。

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 https://ram.console.aliyun.com 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 https://ram.console.aliyun.com 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, objectName) ;

// 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，请参考计量项和计费项的*#unique_91/unique_91_Connect_42_section_uz2_m2t_lgb*一节。

2.13.6 列举文件

本文介绍如何列举文件。

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：

- ObjectListing listObjects(String bucketName)：列举存储空间下的文件。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。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。

ObjectListing的参数如下：

参数	描述	方法
objectSummaries	限定返回的文件元信息。	List<OSSObjectSummary> getObjectSummaries()

参数	描述	方法
prefix	本次查询结果的前缀。	String getPrefix()
delimiter	对文件名称进行分组的一个字符。	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()

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。

```
// 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 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，则列举存储空间下所有的文件。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ListObjectsRequest列举文件

通过设置ListObjectsReques的参数实现各种灵活的查询功能。ListObjectsReques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)
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 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;
// 列举文件。
ObjectListing objectListing = ossClient.listObjects(new ListObject
sRequest(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 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);

// 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 ListObject
sRequest(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 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);
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 ListObject
sRequest(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 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);

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 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);

// 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 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);

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.getObjectS
ummaries()) {
        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();
```

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），显示为Objects。
- 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），该文件夹下的子文件夹（目录）则显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见*#unique_92/unique_92_Connect_42_section_bby_5yy_5db*。创建文件夹的完整代码请参见*GitHub*。

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。下面的示例展示了如何模拟文件夹功能。

- **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：

```
// 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);

// 构造ListObjectsRequest请求。
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(
bucketName);

// 列出文件。
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();
```

  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：

```
// 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);

// 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/");

// 递归列出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();
```

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：

```
// 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);

// 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列表中给出的是fun目录下的所有子文件夹。fun/movie/001.avi和
fun/movie/007.avi两个文件没有被列出来，因为它们属于fun文件夹下的movie目录。
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}

// 关闭OSSClient。
ossClient.shutdown();
```

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**

  以下代码用于获取指定目录下的文件大小：

```
// 获取bucket下某个文件夹大小。
    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).withMaxKeys(1000);
            if (objectListing != null) {
                request.setMarker(objectListing.getNextMarker());
            }
            objectListing = ossClient.listObjects(request);
            List<OSSObjectSummary> sums = objectListing.getObjectS
ummaries();
            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 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);
        // 指定前缀，若希望遍历主目录文件夹，则将该值置空。
        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 + " : " + (calculateF
olderLength(ossClient, bucketName, folder) / 1024) + "KB");
            }
            List<OSSObjectSummary> sums = objectListing.getObjectS
ummaries();
            for (OSSObjectSummary s : sums) {
                System.out.println(s.getKey() + " : " + (s.getSize
() / 1024) + "KB");
            }
        } while (objectListing.isTruncated());
        ossClient.shutdown();
    }
```

2.13.7 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*#unique_93*及*#unique_94*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().withCsvInp
utFormat(
                                        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().withCsvInp
utFormat(
                                        new CSVFormat().withHeaderInfo
(CSVFormat.Header.Use).withRecordDelimiter("\r\n")))
                        .withOutputSerialization(new OutputSeri
alization().withCsvOutputFormat(new CSVFormat()));
        selectObjectRequest.setExpression("select * from ossobject
where _4 > 40");
        OSSObject ossObject = ossClient.selectObject(selectObje
ctRequest);

        // read object content from ossObject
        BufferedReader reader = new BufferedReader(new InputStrea
mReader(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 OutputSeri
alization()
                                .withCrcEnabled(true)
                                .withJsonOutputFormat(new JsonFormat
()))
                        .withExpression("select * from ossobject as s
where s.age > 40");

        OSSObject ossObject = ossClient.selectObject(selectObje
ctRequest);

        // read object content from ossObject
        BufferedReader reader = new BufferedReader(new InputStrea
mReader(ossObject.getObjectContent()));
        while (true) {
            String line = reader.readLine();
            if (line == null) {
                break;
            }
            System.out.println(line);
        }
```

```
        reader.close();

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

2.13.8 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

以下代码用于删除单个文件：

```
// 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 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);

// 删除文件。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。
ossClient.deleteObject(bucketName, objectName);

// 关闭OSSClient。
ossClient.shutdown();
```

删除多个文件

每次最多删除1000个文件。有两种返回模式：

- 详细（verbose）模式：返回删除成功的文件列表。默认为详细模式。
- 简单（quiet）模式：返回删除失败的文件列表。

DeleteObjectsRequest的参数如下：

参数	描述	方法
Keys	需要删除的文件。	setKeys(List<String>)
quiet	返回模式。true表示简单模式，false表示详细模式。默认为详细模式。	setQuiet(boolean)

参数	描述	方法
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)

DeleteObjectsResult的参数如下：

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，为空表示没有编码。	getEncodingType()

以下代码用于批量删除文件：

```
// 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>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 删除文件。key等同于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。
List<String> keys = new ArrayList<String>();
keys.add("key0");
keys.add("key1");
keys.add("key2");

DeleteObjectsResult deleteObjectsResult = ossClient.deleteObjects(new
DeleteObjectsRequest(bucketName).withKeys(keys));
List<String> deletedObjects = deleteObjectsResult.getDeletedObjects();

// 关闭OSSClient。
ossClient.shutdown();
```

批量删除文件的完整代码请参见*GitHub*。

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 = "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>";

// 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 DeleteObje
ctsRequest(bucketName).withKeys(keys);
        ossClient.deleteObjects(deleteObjectsRequest);
    }

    nextMarker = objectListing.getNextMarker();
} while (objectListing.isTruncated());

// 关闭OSSClient。
ossClient.shutdown();
```

2.13.9 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）。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。

参数指定方式	描述
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可设置的参数如下：

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的参数如下：

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

说明:

此操作的限制条件如下:

- 用户有源文件的读写权限。
- 不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。
- 文件的大小不能超过1GB。

拷贝小文件有以下几种拷贝形式。

- 简单拷贝

  以下代码用于简单拷贝:

```
// 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 https://ram.console.aliyun.com 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);

// 拷贝文件。
CopyObjectResult result = ossClient.copyObject(sourceBucketName,
sourceObjectName, destinationBucketName, destinationObjectName);
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拷贝文件:

```
// 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 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, sourceObjectName, destinationBucketName,
destinationObjectName);

// 设置新的文件元信息。
ObjectMetadata meta = new ObjectMetadata();
meta.setContentType("text/html");
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1GB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 https://ram.console.aliyun.com 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(sourceBuck
etName, 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(destinationBucketName, destinatio
nObjectName);
InitiateMultipartUploadResult initiateMultipartUploadResult =
ossClient.initiateMultipartUpload(initiateMultipartUploadRequest);
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, destinationObjectName);
    uploadPartCopyRequest.setUploadId(uploadId);
    uploadPartCopyRequest.setPartSize(size);
    uploadPartCopyRequest.setBeginIndex(skipBytes);
    uploadPartCopyRequest.setPartNumber(i + 1);
    UploadPartCopyResult uploadPartCopyResult = ossClient.uploadPart
Copy(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
CompleteMultipartUploadRequest(
                    destinationBucketName, destinationObjectName,
uploadId, partETags);
ossClient.completeMultipartUpload(completeMultipartUploadRequest);

// 关闭OSSClient。
ossClient.shutdown();
```

分片拷贝的详细说明请参见*UploadPartCopy*。

2.13.10 解冻归档文件

本文介绍如何解冻归档文件。

解冻归档文件的完整代码请参见*GitHub*。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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：

```
// 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);

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();
```

归档存储类型的详细说明请参见存储类型介绍。

2.13.11 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：

```
// 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*。

2.14 数据安全性

Java SDK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，从而保证数据的完整性。

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 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);

// 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。

以下代码用于上传文件时查看CRC数据校验是否生效：

```
// 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);

// 上传字符串。
String content = "Hello OSS";

PutObjectResult result = ossClient.putObject("<yourBucketName>", "<
yourObjectName>", new ByteArrayInputStream(content.getBytes()));

// 判断是否开启CRC校验，true为开启，false为关闭。
if (ossClient.getClientConfiguration().isCrcCheckEnabled()) {
    // 通过响应结果查看CRC校验是否生效。
    OSSUtils.checkChecksum(result.getClientCRC(), result.getServerCRC
(), result.getRequestId());
 }
// 关闭OSSClient。
ossClient.shutdown();
```

说明:

- **putObject、getObject、appendObject、uploadPart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。**

2.15 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

以下代码用于使用STS凭证构造签名请求：

```
// 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 securityToken = "<yourSecurityToken>";

// 用户拿到STS临时凭证后，通过其中的安全令牌（SecurityToken）和临时访问密钥（
AccessKeyId和AccessKeySecret）生成OSSClient。
// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret, securityToken);

// OSS相关操作。例如上传、下载文件等。
// 上传文件。
// ossClient.putObject(putObjectRequest);

// 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。
// ossClient.getObject(new GetObjectRequest(bucketName, objectName),
new File("<yourLocalFile>"));

// 关闭OSSClient。
ossClient.shutdown();
```

使用签名URL进行临时授权

- 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，您可以指定URL的过期时间，来限制访客的访问时长。

- **生成以GET方法访问的签名URL**

  以下代码用于生成以GET方法访问的签名URL：

```
// 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);

// 设置URL过期时间为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);

// 关闭OSSClient。
ossClient.shutdown();
```

- **生成以其他HTTP方法访问的签名URL**

  如果您想允许访客临时进行其他操作（比如上传或删除文件），需要生成对应的签名URL，例如使用生成PUT的签名URL上传文件：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。
String endpoint = "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 securityToken = "<yourSecurityToken>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 用户拿到STS临时凭证后，通过其中的安全令牌（SecurityToken）和临时访问密钥（AccessKeyId和AccessKeySecret）生成OSSClient。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret, securityToken);

GeneratePresignedUrlRequest request = new GeneratePresignedUrl
Request(bucketName, objectName, HttpMethod.PUT);
// 设置URL过期时间为1小时。
Date expiration = new Date(new Date().getTime() + 3600 * 1000);
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PUT方式的签名URL。
URL signedUrl = ossClient.generatePresignedUrl(request);
```

```
Map<String, String> requestHeaders = new HashMap<String, String>();
requestHeaders.put(HttpHeaders.CONTENT_TYPE, DEFAULT_OBJECT_CONTE
NT_TYPE);
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();
```

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

- 生成指定参数的签名URL

  以下代码用于生成指定参数的签名URL：

```
// 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);

// 创建请求。
GeneratePresignedUrlRequest generatePresignedUrlRequest = new
GeneratePresignedUrlRequest(bucketName, objectName);
// HttpMethod为PUT。
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
// 添加用户自定义元信息。
generatePresignedUrlRequest.addUserMetadata("author", "baymax");
// 添加Content-Type。
generatePresignedUrlRequest.setContentType("application/octet-stream
");
// 设置URL过期时间为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
);

// 关闭OSSClient。
ossClient.shutdown();
```

- 使用签名URL上传/获取文件
  - 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：

```
// 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);

Date expiration = DateUtil.parseRfc822Date("Wed, 18 Mar 2022 14:20
:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrl
Request(bucketName, objectName, HttpMethod.GET);
// 设置过期时间。
request.setExpiration(expiration);
// 生成签名URL（HTTP GET请求）。
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();
```

  - 使用签名URL上传文件

    以下代码用于使用签名URL上传文件：

```
// 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>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId
, accessKeySecret);

// 生成签名URL。
```

```
Date expiration = DateUtil.parseRfc822Date("Thu, 19 Mar 2019 18:00
:00 GMT");
GeneratePresignedUrlRequest request = new GeneratePresignedUrl
Request(bucketName, objectName, HttpMethod.PUT);
// 设置过期时间。
request.setExpiration(expiration);
// 设置Content-Type。
request.setContentType("application/octet-stream");
// 添加用户自定义元信息。
request.addUserMetadata("author", "aliy");
// 生成签名URL（HTTP PUT请求）。
URL signedUrl = ossClient.generatePresignedUrl(request);
System.out.println("signed url for putObject: " + signedUrl);

// 使用签名URL发送请求。
File f = new File("<yourLocalFile>");
FileInputStream fin = new FileInputStream(f);
// 添加PutObject请求头。
Map<String, String> customHeaders = new HashMap<String, String>();
customHeaders.put("Content-Type", "application/octet-stream");
customHeaders.put("x-oss-meta-author", "aliy");

PutObjectResult result = ossClient.putObject(signedUrl, fin, f.
length(), customHeaders);

// 关闭OSSClient。
ossClient.shutdown();
```

2.16 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- 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。
- 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。
- 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。
- 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：

```
// 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);

SetBucketLifecycleRequest request = new SetBucketLifecycleRequest(
bucketName);

// 设置规则ID和文件前缀。
String ruleId0 = "rule0";
String matchPrefix0 = "A0/";
String ruleId1 = "rule1";
String matchPrefix1 = "A1/";
String ruleId2 = "rule2";
String matchPrefix2 = "A2/";
String ruleId3 = "rule3";
String matchPrefix3 = "A3/";

// 距最后修改时间3天后过期。
request.AddLifecycleRule(new LifecycleRule(ruleId0, matchPrefix0,
RuleStatus.Enabled, 3));

// 指定日期之前创建的文件过期。
LifecycleRule rule = new LifecycleRule(ruleId1, matchPrefix1,
RuleStatus.Enabled);
rule.setCreatedBeforeDate(DateUtil.parseISO8601Date("2022-10-12T00:00:
00.000Z"));
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

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 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<LifecycleRule> rules = ossClient.getBucketLifecycle(bucketName);
for (LifecycleRule rule : rules) {
    System.out.println(rule.getId());
    System.out.println(rule.getPrefix());
    System.out.println(rule.getExpirationDays());
    System.out.println(rule.getCreatedBeforeDate());
    if(rule.hasAbortMultipartUpload()) {
        System.out.println(rule.getAbortMultipartUpload().getExpirat
ionDays());
        System.out.println(rule.getAbortMultipartUpload().getCreated
BeforeDate());
    }
}

// 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 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);

ossClient.deleteBucketLifecycle(bucketName);

// 关闭OSSClient。
ossClient.shutdown();
```

2.17 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：

```
// 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();
```

2.18 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：

```
// 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("<yourSource
BucketName>");
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();
```

2.19 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("<yourBucket
Name>");
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();
```

2.20 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();
```

2.21 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.getBucketReplication
Progress(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();
```

2.22 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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理的完整代码请参见：*GitHub*。

图片处理功能

OSS图片处理提供以下功能：

- **获取图片信息**
- **图片格式转换**

- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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读写，必须通过授权才能进行访问。

- 匿名访问

  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

参数	描述
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

  授权访问支持自定义样式、HTTPS和级联处理。

  以下代码用于生成带签名的图片处理URL：

  ```
  // 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);
// 设置图片处理样式。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
// 指定过期时间为10分钟。
Date expiration = new Date(new Date().getTime() + 1000 * 60 * 10 );
GeneratePresignedUrlRequest req = new GeneratePresignedUrlRequest(
bucketName, objectName, HttpMethod.GET);
req.setExpiration(expiration);
req.setProcess(style);
URL signedUrl = ossClient.generatePresignedUrl(req);
System.out.println(signedUrl);
// 关闭OSSClient。
ossClient.shutdown();sClient.shutdown();
```

- **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：

```
// 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);
// 缩放
String style = "image/resize,m_fixed,w_100,h_100";
GetObjectRequest request = new GetObjectRequest(bucketName,
objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-resize.jpg"));
// 裁剪
style = "image/crop,w_100,h_100,x_100,y_100,r_1";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-crop.jpg"));
// 旋转
style = "image/rotate,90";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-rotate.jpg"));
// 锐化
```

```
style = "image/sharpen,100";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-sharpen.jpg"));
// 水印
style = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-watermark.jpg"));
// 格式转换
style = "image/format,png";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-format.png"));
// 获取图片信息
style = "image/info";
request = new GetObjectRequest(bucketName, objectName);
request.setProcess(style);
ossClient.getObject(request, new File("example-info.txt"));
// 关闭OSSClient。
ossClient.shutdown();
```

- 自定义样式

  以下代码用于自定义图片样式：

```
// 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>";
String bucketName = "<yourBucketName>";
String objectName = "<yourObjectName>";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId
, accessKeySecret);

// 自定义样式。
String style = "style/<yourCustomStyleName>";
GetObjectRequest request = new GetObjectRequest(bucketName,
objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-new.jpg"));

// 关闭OSSClient。
ossClient.shutdown();
```

- 级联处理

  以下代码用于级联处理图片：

```
// 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);

// 级联处理。
String style = "image/resize,m_fixed,w_100,h_100/rotate,90";
GetObjectRequest request = new GetObjectRequest(bucketName,
objectName);
request.setProcess(style);

ossClient.getObject(request, new File("example-new.jpg"));

// 关闭OSSClient。
ossClient.shutdown();
```

图片处理持久化

以下代码用于图片处理持久化：

```
// 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 sourceImage = "<yourSourceImageName>";

try {
    // 创建OSSClient实例。
    OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId
, accessKeySecret);

    // 图片处理持久化 ： 缩放
    StringBuilder sbStyle = new StringBuilder();
    Formatter styleFormatter = new Formatter(sbStyle);
    String styleType = "image/resize,m_fixed,w_100,h_100";
    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);

    // 图片处理持久化 ： 裁剪
    sbStyle.delete(0, sbStyle.length());
    styleType = "image/crop,w_100,h_100,x_100,y_100,r_1";
    targetImage = "example-crop.png";
    styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
            BinaryUtil.toBase64String(targetImage.getBytes()),
            BinaryUtil.toBase64String(bucketName.getBytes()));
    System.out.println(sbStyle.toString());
```

```
    request = new ProcessObjectRequest(bucketName, sourceImage,
sbStyle.toString());
    processResult = ossClient.processObject(request);
    json = IOUtils.readStreamAsString(processResult.getResponse().
getContent(), "UTF-8");
    processResult.getResponse().getContent().close();
    System.out.println(json);

    // 图片处理持久化 : 旋转
    sbStyle.delete(0, sbStyle.length());
    styleType = "image/rotate,90";
    targetImage = "example-rotate.png";
    styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
            BinaryUtil.toBase64String(targetImage.getBytes()),
            BinaryUtil.toBase64String(bucketName.getBytes()));
    request = new ProcessObjectRequest(bucketName, sourceImage,
sbStyle.toString());
    processResult = ossClient.processObject(request);
    json = IOUtils.readStreamAsString(processResult.getResponse().
getContent(), "UTF-8");
    processResult.getResponse().getContent().close();
    System.out.println(json);

    // 图片处理持久化 : 锐化
    sbStyle.delete(0, sbStyle.length());
    styleType = "image/sharpen,100";
    targetImage = "example-sharpen.png";
    styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
            BinaryUtil.toBase64String(targetImage.getBytes()),
            BinaryUtil.toBase64String(bucketName.getBytes()));
    request = new ProcessObjectRequest(bucketName, sourceImage,
sbStyle.toString());
    processResult = ossClient.processObject(request);
    json = IOUtils.readStreamAsString(processResult.getResponse().
getContent(), "UTF-8");
    processResult.getResponse().getContent().close();
    System.out.println(json);

    // 图片处理持久化 : 水印
    sbStyle.delete(0, sbStyle.length());
    styleType = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
    targetImage = "example-watermark.png";
    styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
            BinaryUtil.toBase64String(targetImage.getBytes()),
            BinaryUtil.toBase64String(bucketName.getBytes()));
    request = new ProcessObjectRequest(bucketName, sourceImage,
sbStyle.toString());
    processResult = ossClient.processObject(request);
    json = IOUtils.readStreamAsString(processResult.getResponse().
getContent(), "UTF-8");
    processResult.getResponse().getContent().close();
    System.out.println(json);

    // 图片处理持久化 : 格式转换
    sbStyle.delete(0, sbStyle.length());
    styleType = "image/format,jpg";
    targetImage = "example-formatconvert.jpg";
    styleFormatter.format("%s|sys/saveas,o_%s,b_%s", styleType,
            BinaryUtil.toBase64String(targetImage.getBytes()),
            BinaryUtil.toBase64String(bucketName.getBytes()));
    request = new ProcessObjectRequest(bucketName, sourceImage,
sbStyle.toString());
    processResult = ossClient.processObject(request);
```

```
    json = IOUtils.readStreamAsString(processResult.getResponse().
getContent(), "UTF-8");
    processResult.getResponse().getContent().close();
    System.out.println(json);

} catch (Exception e) {
    e.printStackTrace();
}
```

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。

2.23 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
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

错误码	描述	HTTP状态码
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
InvalidEncryptionAlgorithmError	指定的熵编码加密算法错误	400

2.24 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，说明您的工程里可能有包冲突。

```
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.java:52)
 at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<init>(
DefaultHttpRequestWriterFactory.java:56)
 at org.apache.http.impl.io.DefaultHttpRequestWriterFactory.<clinit
>(DefaultHttpRequestWriterFactory.java:46)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<
init>(ManagedHttpClientConnectionFactory.java:82)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<
init>(ManagedHttpClientConnectionFactory.java:95)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<
init>(ManagedHttpClientConnectionFactory.java:104)
 at org.apache.http.impl.conn.ManagedHttpClientConnectionFactory.<
clinit>(ManagedHttpClientConnectionFactory.java:62)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager$
InternalConnectionFactory.<init>(PoolingHttpClientConnectionManager.
java:572)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<
init>(PoolingHttpClientConnectionManager.java:174)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<
init>(PoolingHttpClientConnectionManager.java:158)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<
init>(PoolingHttpClientConnectionManager.java:149)
 at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<
init>(PoolingHttpClientConnectionManager.java:125)
 at com.aliyun.oss.common.comm.DefaultServiceClient.createHttp
ClientConnectionManager(DefaultServiceClient.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
```

- 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。详细请参见*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：在Eclipse项目中导入工程依赖的包。**
  - **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，报类似如下错误，可能的原因是Endpoint错误或者网络不通。

```
com.aliyun.oss.ClientException: SocketException
    at com.aliyun.oss.common.utils.ExceptionFactory.createNetw
orkException(ExceptionFactory.java:71)
```

```
    at com.aliyun.oss.common.comm.DefaultServiceClient.sendReques
tCore(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-internal.aliyuncs.com:80 [oss-test.oss-
cn-hangzhou-internal.aliyuncs.com/10.84.135.99] failed: Connection
timed out: connect
    at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator
.connect(DefaultHttpClientConnectionOperator.java:151)
    at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.
connect(PoolingHttpClientConnectionManager.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.sendReques
tCore(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，从而解决问题。

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。

调用OSS Java SDK夯住

- **错误原因**

  调用OSS Java SDK夯住（hang）。通过`jstack -l pid`命令查看堆栈，夯在如下的位置：

```
"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(AbstractQueuedSynchronizer.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(AbstractCo
nnPool.java:64)
    at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(
AbstractConnPool.java:192)
    at org.apache.http.pool.AbstractConnPool$2.getPoolEntry(
AbstractConnPool.java:185)
    at org.apache.http.pool.PoolEntryFuture.get(PoolEntryFuture.java
:107)
    at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.
leaseConnection(PoolingHttpClientConnectionManager.java:276)
    at org.apache.http.impl.conn.PoolingHttpClientConnectionManager$
1.get(PoolingHttpClientConnectionManager.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.sendReques
tCore(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();
```

  问题排查的详细步骤，请参见*OSS Java SDK***夯住问题排查。**

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-Length 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.streamClos
ed(ResponseEntityProxy.java:128)
    at org.apache.http.conn.EofSensorInputStream.checkClose(
EofSensorInputStream.java:228)
```

```
    at org.apache.http.conn.EofSensorInputStream.close(EofSensorI
nputStream.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(ProgressIn
putStream.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**

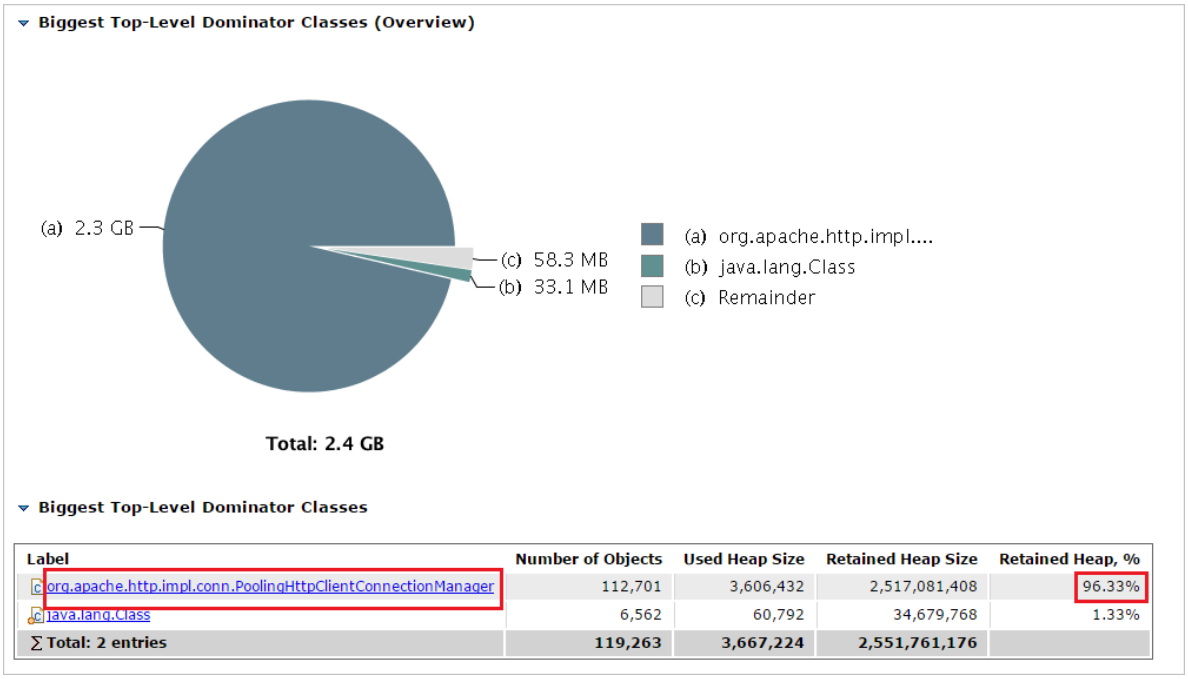
  如果您每次读取部分数据进行处理，且处理数据的时间不固定，建议使用指定范围读取，避免数据读取时连接关闭。

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， 使用方法请参见使用*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一定要和ossClient.shutdown成对使用。

调用ossClient.shutdown抛异常InterruptedException

- **错误原因**

  调用ossClient.shutdown抛如下异常：

```
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，就会抛出上面的异常。

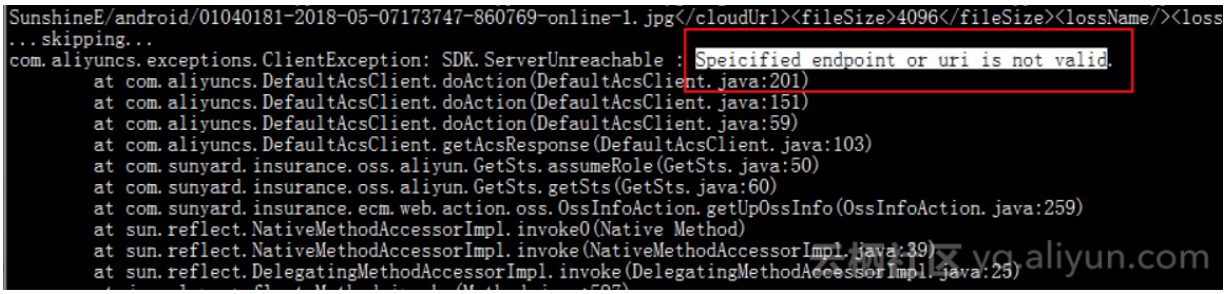
- **解决方法**

  OSS Java SDK 2.3.0已经修复该问题。

  OSS Java SDK 2.3.0之前的版本，请使用如下代码，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”



- **错误原因**

  出现此类报错是因为用户端并发请求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，导致用户无法连接到阿里云网关。

- **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**

  出现此类报错原因是源文件不存在。

- **解决方法**

  参考404问题排查。

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。

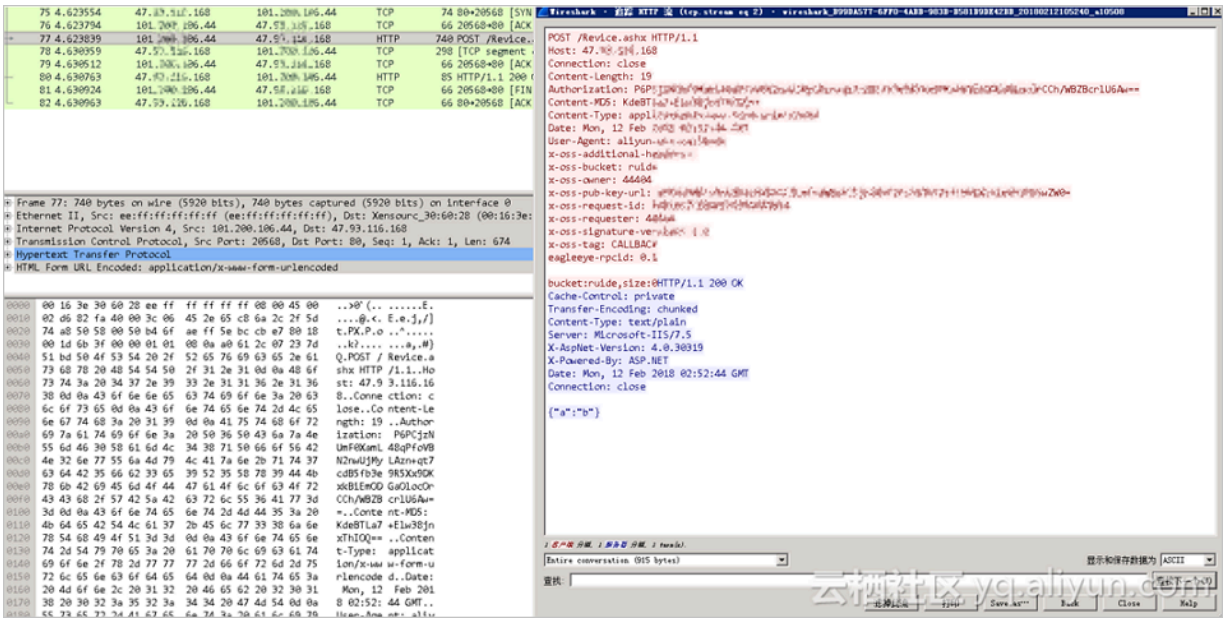
- **解决方法**

  建议从以下几个方面进行排查：

  - **当时是否出现网络抖动。**
  - **主机的socket连接数是否占满。**
  - **SDK中设置的maxconnection是多大。如果当时连接数超过maxconnection设置，也会出现socket异常。**
  - **如果以上都没有问题，建议用户部署tcpdump或者wireshark抓包，复现问题后分析下数据包。**

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msg，这里需要分别测试下Put和Post回调效果：



- **错误原因**

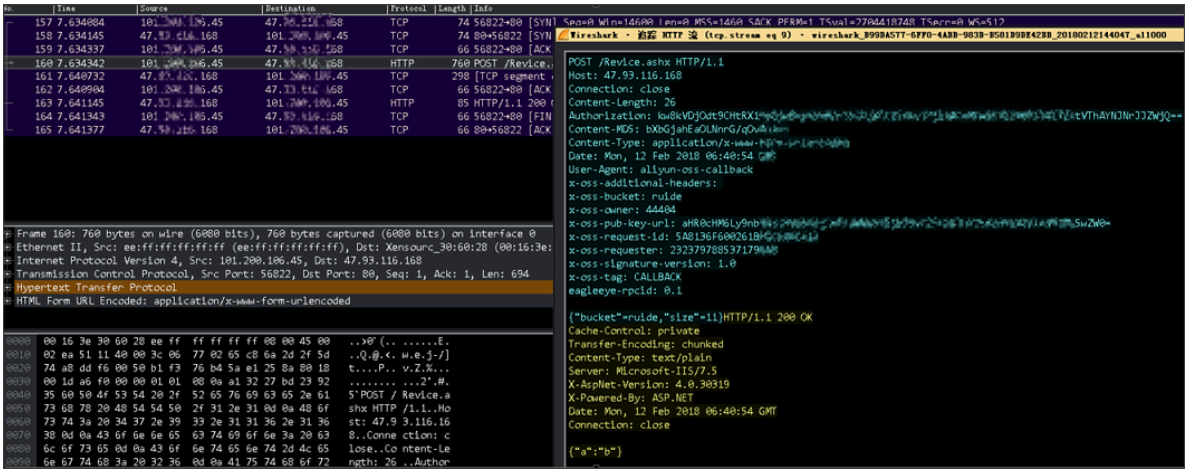
  发送请求时callback参数在file下面。

```
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"

eyJjYWxsYmFja1VybCI6Imh0dHA6Ly80Ny45My4xMTYuMTY4L1JldmljZS5hc2h4IiwiY2FsbGJhY2tCb2R5Ijoie1wiYnVja2V0XCI9JHtidWNrZXR9LFwic2l6ZVwiPSR7c2l6ZX19In0=

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

- **解决方法**

  调整callback参数与file的位置。

  此时测试结果显示业务服务器成功抓取请求。



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(PoolingHttpClientConnectionManager.java:251)
  at org.apache.http.impl.execchain.MainClientExec.execute(MainClient
Exec.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。

其它错误

OSS返回的其它错误排查，请参见常见错误及排查。

3 Python

3.1 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	管理存储空间，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

3.2 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**

  1. 在命令行输入`python`并回车，进入Python环境。

  2. 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：

  1. **在命令行输入`python`并回车，进入Python环境。**
  2. **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
```

3.3 快速入门

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

创建存储空间

存储空间是OSS全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于新建一个存储空间：

```
# -*- 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>')

# 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，请参见基本概念中的命名规范。创建存储空间详情，请参见#unique_41。

获取endpoint信息，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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

# <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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')
# <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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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.ObjectIteratorr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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

# <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。

3.4 初始化

本文介绍如何初始化Python SDK 。

Python SDK中的大部分操作都是通过oss2.Service和oss2.Bucket这两个类进行的。

- 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。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。

初始化oss2.Service类

详情请参管理存储空间中的列举存储空间。

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
```

```
# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>')
```

- **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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

# 自定义域名以my-domain.com为例。is_cname=True为开启CNAME。CNAME是指将自定义域名绑定到存储空间上。
cname = 'http://my-domain.com'
bucket = oss2.Bucket(auth, cname, '<yourBucketName>', 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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

# 设置连接超时时间为30秒。
bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', connect_timeout=30)
```

- **关闭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。**

```
# -*- 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。
endpoint = 'http://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', enable_crc=
False)
```

3.5 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。

创建存储空间

以下代码用于创建存储空间：

```
# -*- 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>')

# 新建存储空间默认为标准存储类型，私有访问权限。
bucket.create_bucket()
```

存储空间的命名规范，请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定存储空间的权限和存储类型。

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))
```

列举存储空间

以下代码用于列举存储空间：

```
# -*- 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。
service = oss2.Service(auth, 'http://oss-cn-hangzhou.aliyuncs.com')
```

```
print([b.name for b in oss2.BucketIterator(service)])
```

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。	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)
```

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：

```
# -*- 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)
```

获取存储空间的信息

以下代码用于获取存储空间的信息（Info）：

```
# -*- 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
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)
```

删除存储空间

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。

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。

3.6 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，请求方必须在其请求中包含x-oss-request-payer: requester（在PUT、POST、GET和HEAD请求的Head信息中），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*#unique_55*。

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)
```

```
printt("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);
```

3.7 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#unique_56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)
```

3.8 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ListBucket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)
```

3.9 对象标签

3.9.1 设置对象标签

本文介绍如何设置对象标签（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#unique_60。

说明:

- **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，若对象已有标签，则覆盖原标签。设置对象标签的详情请参考**#unique_61。
- **设置对象标签时要求请求者有PutObjectTagging权限。**
- **更改标签时#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：**

  +-=._:/

上传Object时添加对象标签

- **普通上传时添加对象标签**

  以下代码用于通过put_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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

object_name = '<yourObjectName>'

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = "k1=v1&k2=v2&k3=v3"

# put_object接口指定headers，将会给上传的Object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('yourObjectName')
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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

object_name = '<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)

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = 'k1=v1&k2=v2'

# 初始化分片。
# init_multipart_upload接口指定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)

# 查看文件的的标签信息。
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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

object_name = '<yourObjectName>'

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = "k1=v1&k2=v2&k3=v3"

# 追加上传文件，append_object接口指定header，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，后续再调用append_object设置
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('yourObjectName')
```

```
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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

object_name = '<yourObjectName>'
local_file = '<yourLocalFileName>'

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = "k1=v1&k2=v2&k3=v3"

# 当文件长度大于或等于可选参数multipart_threshold（默认值为10MB）时，会使用分
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。
# resumable_upload接口指定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]))
```

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：

```
# -*- 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)

# 设置标签。
result = bucket.put_object_tagging('<yourObjectName>', tagging)
# 查看HTTP返回码。
print('http response status:', result.status)
```

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1GB以下的Object、及分片拷贝1GB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：**

```
# -*- coding: utf-8 -*-

import oss2
from oss2.headers import OSS_OBJECT_TAGGING, OSS_OBJECT_TAGGING_C
OPY_DIRECTIVE

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

# 源文件。
src_object_name = '<yourtSrcObjectName>'
# 目标文件1。
dest_object_name1 = '<yourtDestObjectName1>'
# 目标文件2。
dest_object_name2 = '<yourtDestObjectName2>'

# http header中指定OSS_OBJECT_TAGGING_COPY_DIRECTIVE参数为COPY或者默认不
指定，则目标文件1将会拥有源文件同样的标签信息。
headers=dict()
headers[OSS_OBJECT_TAGGING_COPY_DIRECTIVE] = 'COPY'
bucket.copy_object(bucket.bucket_name, src_object_name, dest_objec
t_name1, headers=headers)

# http header中指定OSS_OBJECT_TAGGING_COPY_DIRECTIVE参数为REPLACE，则目
标文件2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] = "key4=value4&key5=value5"
bucket.copy_object(bucket.bucket_name, src_object_name, dest_objec
t_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]))
```

```
# 查看目标文件1的标签信息，将会跟源文件的标签信息保持一致。
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]))

# 查看目标文件2的标签信息，将会为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]))
```

- **以下代码用于分片拷贝1GB 以上的的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.headers import OSS_OBJECT_TAGGING

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>')

# 源文件。
src_object_name = '<yourSrcObject>'
# 目标文件。
dest_object_name = '<yourDestObject>'

# 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方法用来确定分片大小。
part_size = determine_part_size(total_size, preferred_size=100 *
1024)
print('part_size:', part_size)

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = 'k3=v3'

# 初始化分片。
# init_multipart_upload接口指定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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

object_name = '<yourObjectName>'
symlink_name = '<yourSymlinkName>'

# http header中设置标签信息。
headers = dict()
headers[OSS_OBJECT_TAGGING] = "k1=v1&k2=v2&k3=v3"

#添加软链接。
# put_symlink接口指定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]))
```

3.9.2 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#unique_60。

以下代码用于获取对象标签信息：

```
# -*- 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]))
```

获取对象标签详情请参考#unique_63。

3.9.3 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#unique_60。

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#unique_65。

3.9.4 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:**
>
> **Tag条件中，标签的Key和Value必须同时匹配。同一个规则中，若同时配置了前缀和多个对象标签，则对象需满足前缀，且同时匹配规则中所有对象标签，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：

```
# -*- coding: utf-8 -*-
import oss2
import datetime
from oss2.models import (LifecycleExpiration, LifecycleRule,
                         BucketLifecycle,AbortMultipartUpload,
                         TaggingRule, Tagging, StorageTransition)

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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object过期规则，距最后修改时间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', 'logging-',
                      status=LifecycleRule.DISABLED,
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, 12)))

# 设置匹配的标签。
tagging_rule = TaggingRule()
tagging_rule.add('key1', 'value1')
tagging_rule.add('key2', 'value2')
tagging = Tagging(tagging_rule)
```

```
# 设置转储规则，最后修改时间超过356天后转为ARCHIVE，
# rule5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。
rule5 = LifecycleRule('rule5', 'logging2-',
                      status=LifecycleRule.ENABLED,
                      storage_transitions=[StorageTransition(days=356
, 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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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))
```

3.10 上传文件

3.10.1 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。

3.10.2 简单上传

简单上传提供上传字符串、上传Bytes、上传Unicode、上传网络流以及上传本地文件五种形式。

通过bucket.put_object方法上传文件。上传方法支持多种类型的输入源，输入源有如下几种类型：

类型	上传方式
字符串	直接上传
Bytes	直接上传
Unicode	自动转换为UTF-8编码的Bytes进行上传
本地文件	文件对象（File Object），必须以二进制方式打开（如“rb”模式）
网络流	可迭代对象（Iterable），以Chunked Encoding的方式上传

说明:

试图添加文件（Object）时，如果存储空间（Bucket）中已经存在同名文件且对该文件有访问权限，则新添加的文件将覆盖原来的文件，文件添加成功后返回200 OK。

PutObject的更多详情，请参见细节分析。

上传字符串

以下代码用于上传字符串：

```
# -*- 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
# 如果需要上传文件时设置文件存储类型与访问权限，请在put_object中设置相关headers
，参考如下。
# headers = dict()
# headers["x-oss-storage-class"] = "Standard"
# headers["x-oss-object-acl"] = oss2.OBJECT_ACL_PRIVATE
# result = bucket.put_object('<yourObjectName>', 'content of object',
headers=headers)
result = bucket.put_object('<yourObjectName>', 'content of object')

# 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']))
```

上传Bytes

以下代码用于上传Bytes：

```
# -*- 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.put_object('<yourObjectName>', b'content of object')
```

上传Unicode

以下代码用于上传Unicode：

```
# -*- 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.put_object('<yourObjectName>', u'content of object')
```

上传网络流

以下代码用于上传网络流：

```
# -*- coding: utf-8 -*-
import oss2
import requests

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

# 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。
input = requests.get('http://www.aliyun.com')
bucket.put_object('<yourObjectName>', input)
```

上传本地文件

以下代码用于上传本地文件：

```
# -*- 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 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>')

# 必须以二进制的方式打开文件，因为需要知道文件包含的字节数。
with open('<yourLocalFile>', '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()
    bucket.put_object('<yourObjectName>', fileobj)
```

Python SDK还提供了一个更加便捷的方法用于上传本地文件：

```
bucket.put_object_from_file('<yourObjectName>', '<yourLocalFile>')
```

3.10.3 追加上传

本文介绍如何使用追加上传。

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>')

# 设置首次上传的追加位置（Position参数）为0。
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_posit
ion属性，得到追加位置。
bucket.append_object('<yourObjectName>', result.next_position, '
content of second append')
```

如果文件已经存在，如下两种情况将会抛出异常：

- 不可追加文件，则抛出ObjectNotAppendable异常。
- 可追加文件，但设置的追加位置和文件当前长度不等，则抛出PositionNotEqualToLength异常。

3.10.4 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。

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	是否必需	默认值
bucket	存储空间名称	是	无
key	文件名称	是	无
filename	待上传的本地文件名称	是	无

参数	描述	是否必需	默认值
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，则用分片上传	否	10MB
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	并发上传的线程数	否	1

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 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>')
# 当文件长度大于或等于可选参数multipart_threshold（默认值为10MB）时，会使用分片上传。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以上版本支持设置可选参数进行断点续传上传，代码如下：

```
# 如使用store指定了目录，则保存断点信息在指定目录中。如使用num_threads设置上传并发数，请将oss2.defaults.connection_pool_size设成大于或等于线程数。默认线程数为1。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>',
    store=oss2.ResumableStore(root='/tmp'),
    multipart_threshold=100*1024,
    part_size=100*1024,
    num_threads=4)
```

断点续传详情请参见开发指南中的断点续传。

说明：

- 断点续传内部使用了多线程，调用时无需在外面封装多线程，否则可能导致数据重复传输。
- 网络情况较好时，建议增大分片大小。反之，减小分片大小。

3.10.5 分片上传

本文介绍如何使用分片上传。

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。

以下代码用于分片上传文件：

```
# -*- 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。
# 如果需要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)
        # 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。
# 如果需要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.put_object(key, fileobj, headers=headers)
# 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。

3.10.6 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)
```

3.10.7 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*。

put_object、put_object_from_file、complete_multipart_upload支持上传回调功能。

以下代码用于上传回调（callback）：

```
# -*- coding: utf-8 -*-
import json
import base64
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 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>')

# 准备回调参数。
callback_dict = {}
# 设置回调请求的服务器地址，如http://oss-demo.aliyuncs.com:23450或http://
127.0.0.1:9090。
callback_dict['callbackUrl'] = 'http://oss-demo.aliyuncs.com:23450'
# 设置回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
callback_dict['callbackHost'] = 'oss-cn-hangzhou.aliyuncs.com'
# 设置发起回调时请求body的值。
callback_dict['callbackBody'] = 'filename=${object}&size=${size}&
mimeType=${mimeType}'
# 设置发起回调请求的Content-Type。
callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded
'
# 回调参数是JSON格式，并且需要Base64编码。
callback_param = json.dumps(callback_dict).strip()
base64_callback_body = base64.b64encode(callback_param)
# 回调参数编码后放在Header中发送给OSS。
headers = {'x-oss-callback': base64_callback_body}

# 上传并回调。
result = bucket.put_object('<yourObjectName>', 'a'*1024*1024, headers)
```

上传回调详情请参见开发指南中的上传回调。

3.11 下载文件

3.11.1 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。

3.11.2 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>')

# 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('<yourObjectName>')
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
!"
```

以下代码用于将数据下载到本地文件：

```
import shutil

# object_stream是类文件对象，您可以使用shutil.copyfileobj方法，将数据下载到本地文件中。
object_stream = bucket.get_object('<yourObjectName>')
with open('<yourLocalFile>', 'wb') as local_fileobj:
    shutil.copyfileobj(object_stream, local_fileobj)
```

以下代码用于流式拷贝到另一个文件：

```
# object_stream是一个可迭代对象，您可以将它流式拷贝到另一个文件中。
object_stream = bucket.get_object('<yourObjectName>')
```

```
bucket.put_object('<yourBackupObjectName>', object_stream)
```

3.11.3 下载到本地文件

本文介绍如何将OSS文件下载到本地。

以下代码用于把指定的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 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>')

# 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。
bucket.get_object_to_file('<yourObjectName>', '<yourLocalFile>')
```

3.11.4 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 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0~99字节范围内的数据，包括0和99，共100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,
99))
```

3.11.5 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。
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。

3.11.6 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)
```

3.12 管理文件

3.12.1 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

3.12.2 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断指定的文件是否存在：

```
# -*- 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>')

exist = bucket.object_exists('<yourObjectName>')
# 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')
```

3.12.3 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>')

print(bucket.get_object_acl('<yourObjectName>').acl)
```

3.12.4 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 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.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 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.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>')

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 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'])
```

3.12.5 转换文件存储类型

OSS提供标准、低频访问、归档三种存储类型，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。

有关标准存储、低频访问、归档存储三种存储类型的更多信息，请参考开发指南的*#unique_51*及*#unique_90*。

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 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。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 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。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_STO
RAGE_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，请参考计量项和计费项的#unique_91/unique_91_Connect_42_section_uz2_m2t_lgb一节。

3.12.6 列举文件

OSS文件按照字母顺序排列。Python SDK提供了一系列列举文件及获取指定目录下文件大小的方法。

简单列举

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

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

for b in islice(oss2.ObjectIterator(bucket), 10):
    print(b.key)
```

列举指定前缀（目录）下的所有文件

以下代码用于列举指定前缀（目录）下的所有文件：

```
# -*- 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>')

# 列举包含指定前缀的文件。默认列举100个文件。
for obj in oss2.ObjectIterator(bucket, prefix = 'img-'):
    print(obj.key)
```

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：

```
# -*- 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Delimiter参数为正斜线（/）。
for obj in oss2.ObjectIterator(bucket, delimiter = '/'):
    # 通过is_prefix方法判断obj是否为文件夹。
    if obj.is_prefix():  # 文件夹
        print('directory: ' + obj.key)
    else:                # 文件
        print('file: ' + obj.key)
```

获取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：

```
import oss2
def CalculateFolderLength(bucket, folder):
    length = 0
    for obj in oss2.ObjectIterator(bucket, prefix=folder, max_keys=
1000):
        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 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>')
# ObjectIterator 会列举所有的文件或目录，但每次循环列举100个，可以通过
max_keys 来调整数目，最大值为 1000
for obj in oss2.ObjectIterator(bucket, delimiter='/'):
    if obj.is_prefix():  # 文件夹
        length = CalculateFolderLength(bucket, obj.key)
        print('directory: ' + obj.key + '  length:' + str(length /
1024) + "KB")
    else: # 文件
        print('file:' + obj.key + '  length:' + str(obj.size / 1024) +
 "KB")
```

3.12.7 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，请参考开发指南*#unique_93*及*#unique_94*API。**

Python SDK示例

以下代码用于查询CSV和JSON文件：

```
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_params)
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_params)
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。

参数名称	描述
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（,）。
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。

参数名称	描述
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。
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。**

3.12.8 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

以下代码用于删除单个文件：

```
# -*- 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

以下代码用于批量删除文件：

```
# 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>"

# 列举所有指定前缀的文件。
for obj in oss2.ObjectIterator(bucket, prefix=prefix):
```

```
    bucket.delete_object(obj.key)
```

3.12.9 拷贝文件

您可以通过拷贝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）中。

拷贝时限制条件如下：

- **用户有源文件的读写权限。**
- **不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。**

简单拷贝

对于小于1GB的文件，您可以使用简单拷贝。以下代码用于简单拷贝：

```
# -*- 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>')

bucket.copy_object('<yourSourceBucketName>', '<yourSourceObjectName>',
 '<yourDestinationObjectName>')
```

拷贝大文件

对于大于1GB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。**

以下代码用于分片拷贝。

```
# -*- 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。
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)
```

分片拷贝的详细说明请参见*UploadPartCopy*。

3.12.10 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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 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>'

bucket.restore_object(objectName)
```

归档存储类型的详细说明请参见存储类型介绍。

3.12.11 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：

```
# -*- 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*。

3.13 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

首先您需要安装官方的Python STS客户端：

```
pip install aliyun-python-sdk-sts
```

请使用2.0.6及以上版本。以下代码用于使用STS临时授权上传文件：

```
# -*- coding: utf-8 -*-

from aliyunsdkcore import client
from aliyunsdksts.request.v20150401 import AssumeRoleRequest
import json
import oss2

# Endpoint以杭州为例，其它Region请按实际情况填写。
endpoint = '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。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
# role_arn是角色的资源名称。
role_arn = '<yourRoleArn>'

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)
req.set_RoleSessionName('session-name')
```

```
body = clt.do_action(req)

# 使用RAM账号的AccessKeyId和AccessKeySecret向STS申请临时token。
token = json.loads(body)

# 使用临时token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)

# 上传一个字符串。
bucket.put_object('object-name.txt', b'hello world')
```

STS应用完整的示例代码请参见*GitHub*。

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，您可以指定URL的过期时间，来限制访客的访问时长。

以下代码用于使用签名URL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 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>')

# 设置此签名URL在60秒内有效。
print(bucket.sign_url('GET', '<yourObjectName>', 60))
```

3.14 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, isCname: False, connect_ti
meout: 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, status_code: 200
2018-11-20 16:37:46,430 oss2.api [INFO] 26716 : Start to get object
metadata, bucket: bucket-hello, key: object
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`。

3.15 数据加密

3.15.1 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。

注意:

- 客户端加密当前只支持5GB以下的文件。
- 调用客户端加密上传文件后，不允许再调用Python SDK的update_object_meta接口或#unique_177修改object meta信息。

完整的示例代码请参见GitHub。

数据加密密钥的保存方式有如下两种：

- 用户自主管理（RSA）
- KMS托管

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SDK介绍。

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

参数	描述	是否必需
x-oss-meta-oss-cek-alg	数据的加密算法，取值为AES/GCM/NoPadding。	是
x-oss-meta-oss-wrap-alg	数据密钥的加密算法。取值为rsa和kms。	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

使用用户自主管理方式上传和下载文件

您可以使用用户自主管理方式上传和下载文件。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)})

# 下载文件
result = bucket.get_object(key)

# 验证
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
with open(filename, 'rb') as fileobj:
    assert fileobj.read() == content
```

```
os.remove(filename)
```

使用KMS托管方式上传和下载文件

- **前提条件**

  1. **已开通阿里云密钥管理服务。**
  2. **已生成您所在地域的密钥ID。**
  3. **已创建自定义授权策略（操作路径：控制台 > 访问控制 > 策略管理 > 自定义授权策略 > 新建授权策略），并为对应用户关联此策略。**

  参考以下格式创建自定义策略：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateKey",
        "kms:GenerateDataKey",
        "kms:ListKeys",
        "kms:Encrypt",
        "kms:Decrypt"
      ],
      "Resource": [
        "acs:kms:<yourRegion>:<yourloginUserId>:key/*"
      ]
    }
  ]
}
```

- **示例代码**

```
# -*- coding: utf-8 -*-
import os
import oss2
from  oss2.crypto import AliKMSProvider

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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KMS托管方式加密，此方式只支持文件整体上传下载操作。
bucket = oss2.CryptoBucket(auth,'<yourEndpoint>', 'liusiman123456
',crypto_provider=AliKMSProvider('<yourAccessKeyId>', '<yourAccess
KeySecret>', '<yourRegion>', '<yourCMK>', '1234'))

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)})

# 下载文件
result = bucket.get_object(key)

# 验证
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
with open(filename, 'rb') as fileobj:
    assert fileobj.read() == content

os.remove(filename)
```

3.15.2 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。

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。

  > **注意：**
  > - 同一对象同一时间仅可以使用一种服务器器端加密方式。
  > - 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*PutObject*。
  > - 更多关于服务器端加密的介绍请参见*#unique_70*。

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 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 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 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加密配置。
result = bucket.delete_bucket_encryption()
# 查看HTTP返回码。
print('http status:', result.status)
```

3.16 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()
```

3.17 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- **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。**
- **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。**
- **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。**
- **是否生效。**

通过upload_part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
from oss2.models import LifecycleExpiration, LifecycleRule, BucketLife
cycle,AbortMultipartUpload
```

```
import datetime

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>')

# 距最后修改时间3天后过期。
rule1 = LifecycleRule('rule1', 'tests/',
                      status=LifecycleRule.ENABLED,
                      expiration=LifecycleExpiration(days=3))

# 指定日期之前创建的文件过期。
rule2 = LifecycleRule('rule2', 'logging-',
                      status=LifecycleRule.DISABLED,
expiration=LifecycleExpiration(created_before_date=datetime.date(2018
, 12, 12)))

# 分片3天后过期。
rule3 = LifecycleRule('rule3', 'tests1/',
                      status=LifecycleRule.ENABLED,
            abort_multipart_upload=AbortMultipartUpload(days=3))

# 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, 12)))

lifecycle = BucketLifecycle([rule1, rule2,rule3,rule4])

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 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>')

lifecycle = bucket.get_bucket_lifecycle()

for rule in lifecycle.rules:
    if rule.abort_multipart_upload is None:
        print('id={0}, prefix={1}, status={2}, days={3}, created_be
fore_date={4}'
              .format(rule.id, rule.prefix, rule.status,
                      rule.expiration.days,
                      rule.expiration.created_before_date))
    else:
        print('id={0}, prefix={1}, status={2}, days={3}, created_be
fore_date={4}'
              .format(rule.id, rule.prefix, rule.status,
```

```
                        rule.abort_multipart_upload.days,
                        rule.abort_multipart_upload.created_before_date
))
```

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 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_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')
```

3.18 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参考中的*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()
```

3.19 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：

```
# -*- 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_buc
ket, 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)
```

3.20 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_empt
y_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, []))
```

3.21 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。

图片处理的详细信息请参见*OSS图片处理指南*。

图片处理的完整代码请参见：*GitHub*。

图片处理功能

OSS图片处理提供以下功能：

- **获取图片信息**
- **图片格式转换**
- **图片缩放**
- **图片裁剪**
- **图片旋转**
- **图片效果**
- **图片水印，包括添加图片、文字、图文混合水印**
- **自定义图片处理样式**
- **级联处理，调用多个图片处理功能**

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读写，必须通过授权才能进行访问。

- **匿名访问**

  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

参数	描述
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

  授权访问支持自定义样式、HTTPS和级联处理。

  以下代码用于生成带签名的图片处理URL：

  ```
  # -*- coding: utf-8 -*-
  import oss2
  ```

```
# 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。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 生成带签名的URL，并指定过期时间为10分钟。过期时间单位是秒。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'
url = bucket.sign_url('GET', key, 10 * 60, params={'x-oss-process':
style})
print(url)
```

- **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**

    get_object和get_object_to_file支持图片处理。

    以下代码展示了图片处理的基础操作：

```
# -*- 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 = '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。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 缩放
style = 'image/resize,m_fixed,w_100,h_100'
bucket.get_object_to_file(key, new_pic, process=style)
```

```
# 裁剪
style = 'image/crop,w_100,h_100,x_100,y_100,r_1'
bucket.get_object_to_file(key, new_pic, process=style)

# 旋转
style = 'image/rotate,90'
bucket.get_object_to_file(key, new_pic, process=style)

# 锐化
style = 'image/sharpen,100'
bucket.get_object_to_file(key, new_pic, process=style)

# 水印
style = 'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ'
bucket.get_object_to_file(key, new_pic, process=style)

# 格式转换
style = 'image/format,png'
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
os.remove(new_pic)
```

- **自定义样式**

  以下代码用于自定义图片样式：

```
# -*- 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 = '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。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 自定义样式。
style = 'style/oss-pic-style-w-100'

# 图片处理。
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
```

```
os.remove(new_pic)
```

- **级联处理**

  以下代码用于级联处理图片：

```
# -*- 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 = '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。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
bucket_name = '<yourBucketName>'
key = 'example.jpg'
new_pic = 'example-new.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)

# 上传示例图片。
bucket.put_object_from_file(key, 'example.jpg')

# 级联处理。
style = 'image/resize,m_fixed,w_100,h_100/rotate,90'

# 图片处理。
bucket.get_object_to_file(key, new_pic, process=style)

# 删除示例图片。
bucket.delete_object(key)
# 清除本地文件。
os.remove(new_pic)
```

图片处理持久化

我们提供图片转存的数据处理操作，将处理结果作为资源保存到指定Bucket内，并赋以指定Key。保存成功后，下一次可直接通过指定Bucket来访问该资源，以达到提升下载速度的效果。比较适用于超大图的切割或者其他高延时操作。

以下代码用于图片处理持久化：

```
# -*- 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 = '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。
# source_bucket_name与taget_bucket_name为同一Bucket。
access_key_id = '<yourAccessKeyId>'
access_key_secret = '<yourAccessKeySecret>'
source_bucket_name = '<yourBucketName>'
```

```
taget_bucket_name = '<yourTargetBucketName>'
source_image_name = 'example.jpg'

# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, source_bucket_name)

# 缩放
style = 'image/resize,m_fixed,w_100,h_100'
target_image_name = 'example-resize.jpg'
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)

# 裁剪
style = 'image/crop,w_100,h_100,x_100,y_100,r_1'
target_image_name = 'example-crop.jpg'
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)

# 旋转
style = 'image/rotate,90'
target_image_name = 'example-rotate.jpg'
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)

# 锐化
style = 'image/sharpen,100'
target_image_name = 'example-sharpen.jpg'
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)

# 水印
style = 'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ'
target_image_name = 'example-watermark.jpg'
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)

# 格式转换
style = 'image/format,png'
```

```
target_image_name = 'example-formatconvert.png'
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)
```

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。

3.22 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。

函数	描述
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。

3.23 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

异常类	HTTP状态码	OSS错误码	描述
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

3.24 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

用户使用 Python SDK 分片上传函数，上传数据到 OSS失败，且出现很多文件碎片。

文件名	上传 ID	文件碎片
accesslog_2018_09.tar.gz	E609B68C0F0647BF902DF6032902BBD5	点击统计
accesslog_2018_10.tar.gz	EA9E2EDFBA00450688127E17D2AED4CA	点击统计

解决方法：

- 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，Access-Control-Allow-Origin 、Access-Control-Allow-Mehtods、Access-Control-Allow-Headers，并且将 Etag 暴露出去，详情请参考*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、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

ossutil 使用 Go 语言开发 ，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 。有关 crcmod 的更多详情，请参考 *Python SDK 安装*。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。

解决方法：

- **客户端部署 tcpdump 抓包，可以通过 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: */*
Content-Length: 137
date: Sat, 29 Dec 2018 07:32:34 GMT
authorization: OSS LTAIknFr:r2KPR0y4E0G5tnU/MYdcvXHPQQ4=
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>"
3195544E19D99658706D51EF5"</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: 5C2723573183A12D
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/MYdcvXHPQQ4=</SignatureProvided>
 <StringToSign>POST
application/x-www-form-urlencoded
Sat, 29 Dec 2018 07:32:34 GMT
/rokid/ttsservice/passwd?uploadId=D468E486D1D94D90A1AB8885A4E32AE0</
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 45 30 </StringToSi
gnBytes>
</Error>
```

以上是客户端抓到的报文信息。请将获取的请求头信息带入以下脚本，并比较计算结果是否和 SDK 一致。

```
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=D468E486D1D94D90A1AB8885A4E32AE0", 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。**
- **如果服务端收到的 signature 和客户端计算的 signature 不一致，说明请求的内容已经被改动过，建议使用 https 的方式上传。**

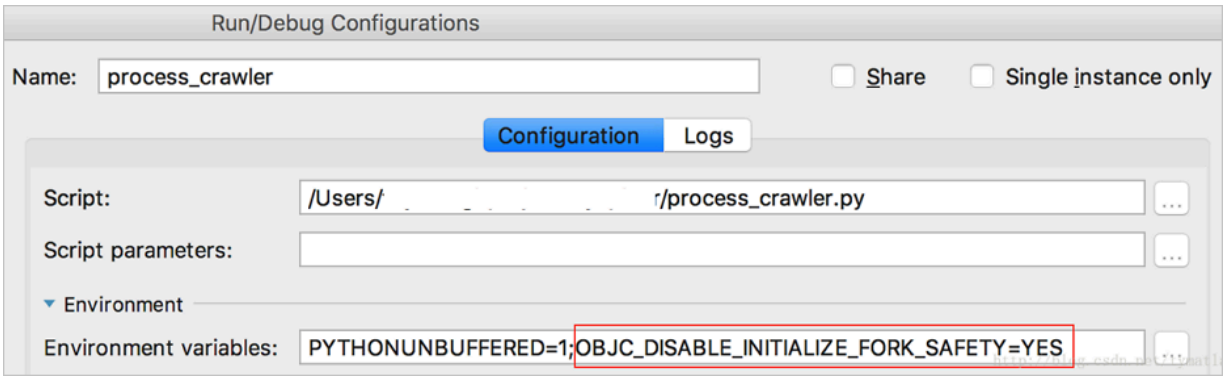
在 Mac 环境中用 Python 启动多线程并在子线程中使用 OSS 时，import tensorflow 会报错，没有 import tensorflow 则不会报错。如果没有启动多线程，使用 OSS 时 import tensorflow 不会报错。

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 called.
objc[2483]: +[__NSPlaceholderDate initialize] may have been in progress in another thread when fork() was called. We cannot safely call it or ignore it in the fork() child process. Crashing instead. Set a breakpoint on objc_initializeAfterForkError to debug.
```

解决方法：

添加环境变量`OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES`**，如下图所示：**



更多详情，请参考 *macOS 10.13***系统使用多线程的错误说明**。

其他错误

有关安装过程中遇到的问题排查，请参见 *Python SDK* **安装文档**。

4 Go

4.1 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*GitHub***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**

示例文件	示例内容
bucket_lifecycle.go	设置、读取、清除文件的生命周期
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

4.2 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)
}
```

4.3 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。 以下代码用于新建一个存储空间：

```
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，请参见基本概念中的命名规范。创建存储空间详情，请参见*#unique_41*。

获取endpoint信息，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。

4.4 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(true))
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*RAM*和*STS*介绍和授权访问。

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)
}
```

4.5 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。

创建存储空间

创建存储空间的完整代码请参见*GitHub*。

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 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，私有权限。
    err = client.CreateBucket("<yourBucketName>")
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
}
```

存储空间的命名规范，请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定存储空间的权限和存储类型。

以下代码用于在创建存储空间时指定存储空间的权限：

```
    // 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）。
    err = client.CreateBucket("<yourBucketName>", oss.ACL(oss.
ACLPublicRead))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
```

以下代码用于创建归档类型的存储空间：

```
    err = client.CreateBucket("<yourBucketName>", oss.StorageClass(oss
.StorageArchive))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
```

以下代码用于创建低频访问类型的存储空间：

```
    err = client.CreateBucket("<yourBucketName>", oss.StorageClass(oss
.StorageIA))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
```

列举存储空间

列举存储空间的完整代码请参见*GitHub*。

- 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的存储空间：

```
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**

  以下代码用于列举包含指定前缀（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**

  参数marker代表存储空间名称。以下代码用于列举指定marker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**

  以下代码用于列举指定个数（maxKeys）的存储空间：

```
    // 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)
    }
```

判断存储空间是否存在

以下代码用于判断指定的存储空间是否存在：

```
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)
}
```

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

更多关于访问权限的内容请参见开发指南中访问控制。存储空间访问权限的完整代码请参见*GitHub*。

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)
    }
}
```

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)
}
```

获取存储空间的地域

以下代码用于获取存储空间的地域（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)
}
```

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）：

```
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)
    }

    // 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）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.CreationDa
te)
    fmt.Println("BucketInfo.ACL: ", res.BucketInfo.ACL)
    fmt.Println("BucketInfo.Owner: ", res.BucketInfo.Owner)
    fmt.Println("BucketInfo.StorageClass: ", res.BucketInfo.StorageCla
ss)
    fmt.Println("BucketInfo.ExtranetEndpoint: ", res.BucketInfo.
ExtranetEndpoint)
    fmt.Println("BucketInfo.IntranetEndpoint: ", res.BucketInfo.
IntranetEndpoint)
}
```

删除存储空间

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。

说明:

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：

```
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)
    }
}
```

4.6 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，请求方必须在其请求中包含x-oss-request-payer:requester（在PUT、POST、GET和HEAD请求的Head信息中），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#unique_55。

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。
  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.RequestPay
er(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)
  }
}
```

4.7 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*#unique_56*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localFileTwo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)
    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。
  // 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)
  }
}
```

4.8 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ListBucket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：

```
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)
  }
}
```

4.9 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。

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。

> **注意：**
>
> - **同一对象同一时间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***PutObject*。
> - **更多关于服务器端加密的介绍请参见***#unique_70*。

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)
  }
}
```

4.10 对象标签

4.10.1 设置对象标签

本文介绍如何设置对象标签（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#unique_60。

说明:

- **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，若对象已有标签，则覆盖原标签。设置对象标签的详情请参考#unique_61。**
- **设置对象标签时要求请求者有PutObjectTagging权限。**
- **更改标签时#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：**

  +-=._:/

上传Object时添加对象标签

- **普通上传时添加对象标签**

  以下代码用于普通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(tagging))
  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 {
    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1GB以下的Object、及分片拷贝1GB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>", "<yourDirect
ObjectName>", oss.SetTagging(tagging))
  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>", "<yourDirect
ObjectName>", oss.SetTagging(tagging), 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(content)), 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)
  }
}
```

4.10.2 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#unique_60。

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#unique_63。

4.10.3 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#unique_60。

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#unique_65。

4.10.4 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:**
>
> **Tag条件中，标签的Key和Value必须同时匹配。同一个规则中，若同时配置了前缀和多个对象标签，则对象需满足前缀，且同时匹配规则中所有对象标签，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, transition
Arch},
    Tags:[]oss.Tag{tag1,tag2},
```

```
  }
  rules := []oss.LifecycleRule{rule}
  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")
  }
}
```

4.11 上传文件

4.11.1 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。

4.11.2 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

以下代码用于上传字符串：

```
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)
    }

    // 指定存储类型为归档存储。
    storageType := oss.ObjectStorageClass(oss.StorageArchive)

    // 指定访问权限为公共读。
    objectAcl := oss.ObjectACL(oss.ACLPublicRead)

    // 上传字符串。
    err = bucket.PutObject("<yourObjectName>", strings.NewReader("
yourObjectValue"), storageType, objectAcl)
    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

以下代码用于上传本地文件：

```
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)
    }
```

```
}
```

4.11.3 追加上传

本文介绍如何使用追加上传。

追加类型的文件（Append Object）暂时不支持copyObject操作。文件不存在时，调用`AppendObject`会创建一个可追加的文件；文件存在时，调用`AppendObject`会向文件末尾追加内容。

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。
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.
NewReader("YourObjectAppendValue1"), nextPos)
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }

 // 第二次追加。
 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Append。
```

```
}
```

第一次追加（即开始位置是0的追加）可以指定文件元信息。后续追加不能指定文件元信息。

```
 // 第一次追加指定文件元信息。
 nextPos, err = bucket.AppendObject("<yourObjectName>", strings.
NewReader("YourObjectValue"), 0, oss.Meta("MyProp", "MyPropVal"))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
```

4.11.4 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。

在上传的过程中会在Checkpoint文件中记录当前上传的进度信息，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。

说明:

- SDK会将上传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`实现断点续传。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	分片上传大小，取值范围为100KB~5GB。

参数	说明
options	可选项，包括： · Routines：指定分片上传的并发数。默认是1，即不使用并发上传。 · Checkpoint：指定是否开启断点续传功能以及设置Checkpoint文件。默认关闭断点续传功能。例如`oss.Checkpoint(true, "")`，表示开启断点续传功能，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。 · 其它元信息：请参见设置文件元信息。

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)
 }

 // 分片大小100K，3个协程并发上传分片，使用断点续传。
 // 其中"<yourObjectName>"为objectKey， "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)
 }
```

```
}
```

4.11.5 分片上传

本文介绍如何使用分片上传。

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.CompleteMultipartUpload方法将所有分片合并成完整的文件。

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。
    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()

    // 指定存储类型为归档存储。
    storageType := oss.ObjectStorageClass(oss.StorageArchive)

    // 步骤1：初始化一个分片上传事件，指定存储类型为归档存储。
    imur, err := bucket.InitiateMultipartUpload(objectName, storageTyp
e)
    // 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)
    }

    // 指定访问权限为公共读。
    objectAcl := oss.ObjectACL(oss.ACLPublicRead)

    // 步骤3：完成分片上传，指定访问权限为公共读。
    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：

```
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)
    }
    // 初始化一个分片上传事件。
    imur, err := bucket.InitiateMultipartUpload("<yourObjectName>")
    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。
    client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<
yourAccessKeySecret>")
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    bucketName := "<yourBucketName>"
    objectName := "<yourObjectName>"
    locaFilename := "<yourLocalFilename>"
    uploadID := "<yourUploadID>"
    // 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，此处分为3片。
    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)
        // 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)
        }
        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.LastModified)
    }
    // 根据objectName和UploadID生成InitiateMultipartUploadResult，然后列
举所有已上传的分片，这种情况适用于已知objectName和UploadID的情况。
    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.LastModified)
    }
    // 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。可设置的参数如下：

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。
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Upload ID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。
    client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<
yourAccessKeySecret>")
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    // 获取存储空间。
    bucketName := "<yourBucketName>"
    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 {
        // 默认情况下一次返回1000条记录。
        lsRes, err := bucket.ListMultipartUploads(oss.KeyMarker(
keyMarker), oss.UploadIDMarker(uploadIdMarker))
        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**

```
    lsRes, err := bucket.ListMultipartUploads(oss.Prefix("<
yourObjectNamePrefix>"))
    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)
```

- **同时指定前缀和最大返回条数**

```
    lsRes, err := bucket.ListMultipartUploads(oss.Prefix("<
yourObjectNamePrefix>"), oss.MaxUploads(100))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    fmt.Println("Uploads:", lsRes.Uploads)
```

4.11.6 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)
 }
```

```
}
```

4.12 下载文件

4.12.1 概述

下载文件的完整示例代码请参见*GitHub*。

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。

4.12.2 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)
}
```

4.12.3 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。

以下代码用于范围下载：

```
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))
}
```

4.12.4 下载到本地文件

本文介绍如何把OSS文件下载到本地。

以下代码用于把指定的OSS文件下载到本地文件：

```
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)
  }

  // 下载文件到本地文件。
  err = bucket.GetObjectToFile("<yourObjectName>", "LocalFile")
  if err != nil {
   fmt.Println("Error:", err)
   os.Exit(-1)
  }
}
```

4.12.5 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（part）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，完成整个文件的下载。

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。

参数	说明
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， "LocalFile"为filePath，100*1024
为partSize。
 err = bucket.DownloadFile("<yourObjectName>", "LocalFile", 100*1024,
oss.Routines(3), oss.Checkpoint(true, ""))
 if err != nil {
  fmt.Println("Error:", err)
  os.Exit(-1)
 }
```

```
}
```

4.12.6 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
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

IfModifiedSince和IfUnmodifiedSince可以同时存在。IfMatch和IfNoneMatch可以同时存在。

ETag可以通过Bucket.GetObjectDetailedMeta方法获取。

```
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)
 }
}
```

4.12.7 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)
 }
}
```

4.12.8 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.GetObjectToFile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)
 }

 // 带进度条的下载。
 err = bucket.GetObjectToFile(objectName, localFile, oss.Progress(&
OssProgressListener{}))
 if err != nil {
 fmt.Println("Error:", err)
 os.Exit(-1)
 }
 fmt.Println("Transfer Completed.")
}
```

4.13 管理文件

4.13.1 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**

4.13.2 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断指定的文件是否存在：

```
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)
}
```

4.13.3 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。

```
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)
}
```

4.13.4 管理文件元信息

本文介绍如何使用对象存储OSS的Go SDK设置、修改和获取文件元信息。

文件元信息（Object Meta）详情请参见开发指南中的文件元信息。文件元信息的完整代码请参见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。
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）。

参数	说明
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)
}
```

4.13.5 转换文件存储类型

OSS提供标准、低频访问、归档三种存储类型，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。

有关标准存储、低频访问、归档存储三种存储类型的更多信息，请参考开发指南的#unique_51及#unique_90。

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.ObjectStor
ageClass(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.HTTPHeader
OssStorageClass))
 if meta.Get(oss.HTTPHeaderOssStorageClass) == string(oss.StorageArc
hive) {
  //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.ObjectStor
ageClass(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，请参考计量项和计费项的*#unique_91/unique_91_Connect_42_section_uz2_m2t_lgb*一节。

4.13.6 列举文件

本文介绍如何列举文件。

OSS文件按照字母顺序排列。您可以通过`Bucket.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。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。

参数	说明
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。

ListObjects的完整代码请参见*Github*。

简单列举文件

您可以使用默认参数获取存储空间的文件列表，默认返回100条记录。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
        }
    }
```

```
}
```

列举指定个数的文件

以下代码用于列举指定个数的文件：

```
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)
    }

    // 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)
    }

    // 打印结果。
    fmt.Println("Objects:", lsRes.Objects)
    for _, object := range lsRes.Objects {
        fmt.Println("Object:", object.Key)
    }
}
```

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：

```
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)
    }

    // 打印结果。
    fmt.Println("Objects:", lsRes.Objects)
    for _, object := range lsRes.Objects {
        fmt.Println("Object:", object.Key)
    }
}
```

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:

```
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)
    }
```

```
    // 列举指定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)
    }

    // 打印结果。
    fmt.Println("Objects:", lsRes.Objects)
    for _, object := range lsRes.Objects {
        fmt.Println("Object:", object.Key)
    }
}
```

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。

```
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200个。
    marker := oss.Marker("")
    for {
        lsRes, err := bucket.ListObjects(oss.MaxKeys(200), marker)
        if err != nil {
            fmt.Println("Error:", err)
            os.Exit(-1)
        }
        marker = oss.Marker(lsRes.NextMarker)

        fmt.Println("Objects:", lsRes.Objects)

        if !lsRes.IsTruncated {
            break
        }
    }
}
```

分页列举指定marker之后的文件

以下代码用于分页列举指定maker之后的文件。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)
    }

    // 分页列举指定maker之后的文件。每页列举50个。
    marker = oss.Marker("my-object-xx")
    for {
        lsRes, err := bucket.ListObjects(oss.MaxKeys(50), marker)
        if err != nil {
            fmt.Println("Error:", err)
            os.Exit(-1)
        }

        marker = oss.Marker(lsRes.NextMarker)

        fmt.Println("Objects:", lsRes.Objects)

        if !lsRes.IsTruncated {
            break
        }
    }
}
```

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。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)

        fmt.Println("Objects:", lsRes.Objects)

        if !lsRes.IsTruncated {
            break
        }
    }
}
```

列举指定前缀（目录）下所有文件的信息

以下代码用于列举指定前缀（目录）下所有文件的信息，包含文件大小、文件最后被修改的时间及文件名等：

```
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
        }
    }
}
```

列举指定前缀（目录）下所有子目录的信息

以下代码用于列举指定目录下所有子目录的信息：

```
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
        }
    }
}
```

4.13.7 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，文件一旦删除将无法恢复。**

删除文件的完整代码请参见*GitHub*。

删除单个文件

以下代码用于删除单个文件：

```
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。

以下代码用于批量删除文件：

```
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.DeleteObje
ctsQuiet(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)
}
```

4.13.8 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*GitHub*。

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。
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。
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)
 }
}
```

4.13.9 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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。

归档存储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，进入读就绪状态。
```

```
        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")
}
```

4.13.10 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：

```
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>"
    // 软链接。
    objectName := "<yourSymlinkName>"
```

```
    // 软链接目标文件。
    targetObjectName := "<yourSymlinkTargetName>"

    // 获取存储空间。
    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)
    }

    // 上传文件到软链接目标文件。
    err = bucket.PutObject(targetObjectName, strings.NewReader("target
"))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }

    err = bucket.PutSymlink(objectName, targetObjectName)
    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>"
    // 软链接名称。
    objectName := "<yourSymlinkObjectName>"

    // 获取存储空间。
    bucket, err := client.Bucket(bucketName)
    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))
}
```

获取软链接的详细信息请参见*GetSymlink*。

4.14 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

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"

// 用户拿到STS临时凭证后，通过其中的安全令牌（SecurityToken）和临时访问密钥（
AccessKeyId和AccessKeySecret）生成OSSClient。
// 创建OSSClient实例。
client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<
yourAccessKeySecret>", oss.SecurityToken("<yourSecurityToken>"))
if err != nil {
    fmt.Println("Error:", err)
    os.Exit(-1)
```

```
// OSS操作。
}
```

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，您可以指定URL的过期时间，来限制访客的访问时长。

使用签名URL进行临时授权的完整代码请参见*GitHub*。

- **使用签名URL上传文件**

  以下代码用于使用签名URL上传文件：

```
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() {
    client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<
yourAccessKeySecret>")
    if err != nil {
        HandleError(err)
    }

    bucketName := "<yourBucketName>"
    objectName := "<yourObjectName>"
    localFilename := "<yourLocalFilename>"

    // 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。
    options := []oss.Option{
        oss.Meta("myprop", "mypropval"),
        oss.ContentType("image/tiff"),
    }

    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:

可选参数详情请参见管理文件中的文件元信息。

- **使用签名URL下载文件**

  以下代码用于使用签名URL下载文件:

```
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() {
    client, err := oss.New("<yourEndpoint>", "<yourAccessKeyId>", "<
yourAccessKeySecret>")
    if err != nil {
        HandleError(err)
    }

    bucketName := "<yourBucketName>"
    objectName := "<yourObjectName>"
    localDownloadedFilename := "<yourDownloadedFilename>"

    // 获取存储空间。
    bucket, err := client.Bucket(bucketName)
    if err != nil {
        HandleError(err)
    }

    // 签名直传，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

    // 签名直传，下载到文件。
    err = bucket.GetObjectToFileWithURL(signedURL, localDownl
oadedFilename)
    if err != nil {
        HandleError(err)
    }
}
```

4.15 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- 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。
- 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。
- 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。
- 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。管理生命周期的完整代码请参见*GitHub*。

设置生命周期规则

以下代码用于设置生命周期规则：

```
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)
    }

    // 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)
    rules := []oss.LifecycleRule{rule1}

    // 设置生命周期规则。
    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)
    }
```

4.16 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)
    }
}
```

4.17 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

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)
 }
}
```

4.18 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)
 }
}
```

4.19 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)
 }
}
```

4.20 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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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读写，必须通过授权才能进行访问。

- **匿名访问**

  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

参数	描述
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

  授权访问支持自定义样式、HTTPS和级联处理。

  以下代码用于生成带签名的图片处理URL：

  ```
  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)
    }

    ossImageName := "<yourImageName>"
    signedURL, err := bucket.SignURL(ossImageName, oss.HTTPGet, 600
, oss.Process("image/format,png"))
    if err != nil {
 HandleError(err)
    } else {
 fmt.Println(signedURL)
    }
}
```

- **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：

```
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)
    }

    // 图片处理：缩放。
    sourceImageName := "example.png"
    targetImageName := "example-resize.png"
    style := "image/resize,m_fixed,w_100,h_100"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }

    // 图片处理：裁剪。
    targetImageName = "example-crop.png"
    style = "image/crop,w_100,h_100,x_100,y_100,r_1"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }

    // 图片处理：旋转。
    targetImageName = "example-rotate.png"
    style = "image/rotate,90"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }

    // 图片处理：锐化。
    targetImageName = "example-sharpen.png"
    style = "image/sharpen,100"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }

    // 图片处理：水印。
    targetImageName = "example-watermark.png"
    style = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }

    // 图片处理：格式转换。
    targetImageName = "example-formatconvert.jpg"
    style = "image/format,jpg"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
```

```
    if err != nil {
 HandleError(err)
    }
}
```

- **自定义样式**

  以下代码用于自定义图片样式：

```
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)
    }

    // 图片处理：自定义样式。
    sourceImageName := "example.png"
    targetImageName := "example-custom.png"
    style := "style/<yourCustomStyleName>"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
 HandleError(err)
    }
}
```

- **级联处理**

  以下代码用于级联处理图片：

```
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)
    }

    // 获取存储空间。
    bucketName := "<yourBucketName>"
    bucket, err := client.Bucket(bucketName)
    if err != nil {
 HandleError(err)
     }

    // 图片处理：自定义样式。
    sourceImageName := "example.png"
    targetImageName := "example-cascade.png"
    style := "image/resize,m_fixed,w_100,h_100/rotate,90"
    err = bucket.GetObjectToFile(sourceImageName, targetImageName
, oss.Process(style))
    if err != nil {
  HandleError(err)
    }
}
```

图片处理持久化

以下代码用于图片处理持久化：

```
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)
    }

    // 获取存储空间。
    bucketName := "<yourBucketName>"
    bucket, err := client.Bucket(bucketName)
    if err != nil {
  HandleError(err)
    }
```

```
    // 图片处理持久化：缩放。目标图片默认会存放到相同的bucket中。
    sourceImageName := "example.png"
    targetImageName := "example-resize.png"
    style := "image/resize,m_fixed,w_100,h_100"
    process := fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err := bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }

    // 图片处理持久化：缩放。将目标图片存放到不同的bucket中。
    targetBucketName := "<yourTargetBucketName>"
    targetImageName = "example-crop-different-bucket.png"
    style = "image/resize,m_fixed,w_100,h_100"
    process = fmt.Sprintf("%s|sys/saveas,o_%v,b_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)), base64.URLEncodin
g.EncodeToString([]byte(targetBucketName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }

    // 图片处理持久化：裁剪。
    targetImageName = "example-crop.png"
    style = "image/crop,w_100,h_100,x_100,y_100,r_1"
    process = fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }

    // 图片处理持久化：旋转。
    targetImageName = "example-rotate.png"
    style = "image/rotate,90"
    process = fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }

    // 图片处理持久化：锐化。
    targetImageName = "example-sharpen.png"
    style = "image/sharpen,100"
    process = fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
       fmt.Println(result)
    }
```

```
    // 图片处理持久化：水印。
    targetImageName = "example-watermark.png"
    style = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ"
    process = fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }

    // 图片处理持久化：格式转换。
    targetImageName = "example-formatconvert.jpg"
    style = "image/format,jpg"
    process = fmt.Sprintf("%s|sys/saveas,o_%v", style, base64.
URLEncoding.EncodeToString([]byte(targetImageName)))
    result, err = bucket.ProcessObject(sourceImageName, process)
    if err != nil {
 HandleError(err)
    } else {
        fmt.Println(result)
    }
}
```

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。

4.21 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("yourObject
ValueContentString"))
    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**

错误码	描述	HTTP状态码
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
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

错误码	描述	HTTP状态码
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
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。

5 C++

5.1 前言

本文档基于 C++ SDK 1.3.0 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

5.2 安装

本文介绍如何安装 OSS C++ SDK。

环境准备

请安装支持 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

直接下载

通过 *GitHub* 下载

安装SDK

可以通过 Linux 、Windows 、Android 及 Mac 系统安装 SDK。

- Linux 系统：

  1. 安装 CMake

     请安装 CMake3.1 及以上版本。进入 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 ..
     ```

  2. 安装第三方库 libcurl、openssl

     RedHat/Centos：

     ```
     yum -y install libcurl-devel openssl-devel
     ```

     Fedora：

     ```
     sudo dnf install libcurl-devel openssl-devel
     ```

  3. 安装 SDK

     ```
     make && make install
     ```

     说明：

C++ SDK 默认关闭 rtti 属性。所以使用 g++ 编译运行时，请添加 -std=c++11、-fno-rtti、-lalibabacloud-oss-cpp-sdk。示例如下：

```
g++ test.cpp -std=c++11 -fno-rtti -lalibabacloud-oss-cpp-sdk -o
test.bin
```

- Windows 系统：

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。

1. 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：

如果要构建 x64 体系结构，可以使用命令cmake -A x64 ..来实现。

2. 请以管理员身份运行VS开发人员命令提示符，进入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
```

- **Mac 系统：**

  在 Mac 上，请指定 OpenSSL 安装路径。

  示例如下，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
```

5.3 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。

创建存储空间

存储空间是OSS全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于新建一个存储空间：

```
#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、存储级别和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;
}
```

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

获取endpoint信息，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 <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 */
    auto outcome = client.PutObject(BucketName, ObjectName,"yourLocalF
ilename");

    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

以下代码用于下载文件到本地文件：

```
#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";
    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::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";
    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() <<
            ",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";
    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);

    /* 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;
}
```

删除文件详情请参见管理文件中的删除文件。

5.4 初始化

OssClient 是用于管理存储空间（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。默认为 aliyun-sdk-cpp/1.X.X
maxConnections	连接池数。默认是 16 个
requestTimeoutMs	请求超时时间。超时没有收到数据会关闭连接，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
verifySSL	是否开启 SSL 验证，默认关闭
enableCrc64	是否开启 CRC64 检查，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;
}
```

设置限速处理

以下代码用于设置上传、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,"yourLocalF
ilename");

    /*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最多重试次数，scaleFactor重试等待时间的尺度因子 */
    DefaultRetryStrategy(long maxRetries = 3, long scaleFactor = 300
) :
        m_scaleFactor(scaleFactor), m_maxRetries(maxRetries)
    {}

    /* 用户可自定义shouldRetry函数，用于判断是否进行重试 */
    bool shouldRetry(const Error & error, long attemptedRetries) const
;

    /* 用户可自定义calcDelayTimeMs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 DefaultRetryStrategy::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("RequestTim
eTooSkewed")) ||
        (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 DefaultRetryStrategy::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;
```

```
}
```

5.5 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。

创建存储空间

以下代码用于创建存储空间：

```
#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;
}
```

存储空间的命名规范请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定存储空间的权限和存储类型。

列举存储空间

以下代码用于列举所有的存储空间：

```
#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;
}
```

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;
```

```
}
```

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;
}
```

获取存储空间的地域

以下代码用于获取存储空间的地域（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;
}
```

判断存储空间是否存在

以下代码用于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;
}
```

删除存储空间

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。

> **说明：**
> **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：

```
#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;
```

```
}
```

5.6 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，请求方必须在其请求中包含x-oss-request-payer: requester（在PUT、POST、GET和HEAD请求的Head信息中），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#unique_55。

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, PayerAcces
sKeySecret, conf);

    /* 创建bucket */
    CreateBucketOutcome outCome = client.CreateBucket(CreateBuck
etRequest(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;
}
```

5.7 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。

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。

> **注意：**
> - 同一对象同一时间仅可以使用一种服务器器端加密方式。
> - 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*PutObject*。
> - 更多关于服务器端加密的介绍请参见*#unique_70*。

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;
}
```

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ListBucket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：

```
#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(BucketName);
    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标签

以下代码用于删除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);

    /*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;
}
```

5.9 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*#unique_56*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::in|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, PutobjectU
rlName, 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, PutobjectU
rlName, 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;
}
```

5.10 对象标签

5.10.1 设置对象标签

本文介绍如何设置对象标签（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#unique_60。

说明:

- **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，若对象已有标签，则覆盖原标签。设置对象标签的详情请参考#unique_61。**
- **设置对象标签时要求请求者有PutObjectTagging权限。**
- **更改标签时#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：**

  +-=._:/

上传Object时添加对象标签

- **普通上传时添加对象标签**

  以下代码用于普通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(initUpload
Request);
    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()) {
        /* 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, YourUpload
FileName);

    /* 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(SetObjectT
aggingRequest(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1GB以下的Object、及分片拷贝1GB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(initUpload
Request);
    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(CopyBucket
Name, CopyObjectName, SourceBucketName, SourceObjectName,uploadId, i
 + 1);
        uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes +
size -1);
        auto uploadPartOutcome = client.UploadPartCopy(uploadPart
CopyReq);
        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;
}
```

5.10.2 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#unique_60。

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(SetObjectT
aggingRequest(BucketName, ObjectName, tagging));

    /* 获取对象标签 */
    getTaggingOutcome = client.GetObjectTagging(GetObjectTaggingRequ
est(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#unique_63。

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#unique_60。

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(SetObjectT
aggingRequest(BucketName, ObjectName, tagging));

    /* 删除对象标签 */
    auto delTaggingOutcome = client.DeleteObjectTagging(DeleteObje
ctTaggingRequest(BucketName, ObjectName));

    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#unique_65。

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:

Tag条件中，标签的Key和Value必须同时匹配。同一个规则中，若同时配置了前缀和多个对象标签，则对象需满足前缀，且同时匹配规则中所有对象标签，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;
}
```

5.11 上传文件

5.11.1 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。

5.11.2 简单上传

本文主要介绍如何从内存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";
    /* yourObjectName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.jpg */
    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 << "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";
    std::string BucketName = "yourBucketName";
    /* yourObjectName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.jpg*/
    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::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;
```

```
}
```

5.11.3 追加上传

本文介绍如何使用追加上传。

以下代码用于从本地进行追加上传：

```
#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);

    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;
}
```

如果文件存在，会出现以下两种情况：

- 不可追加文件，则抛出 ObjectNotAppendable 异常。
- 可追加文件，通过 PutObject 操作上传，将覆盖原有文件，文件类型变为 Normal Object。HeadObject 操作会返回 x-oss-object-type，用于表明 Object 的类型。

5.11.4 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。

断点续传详情请参见开发指南中的断点续传。完整代码请参见*GitHub*。

您可以通过 OssClient .ResumableUploadObject 方法实现断点续传上传。此方法的 UploadObjectRequest 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将上传该文件。	否	本地文件名称	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	分片上传的并发数。	否	3	通过构造函数或者setThreadNum 设置。

参数	描述	是否必需	默认值	如何设置
checkpointDir	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。	否	与DownloadFile同目录。	通过构造函数或者 setCheckpointDir 设置。

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;
```

```
}
```

5.11.5 分片上传

本文主要介绍如何使用分片上传。

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，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 CompleteMultipartUpload 方法将所有分片合并成完整的文件。

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(initUpload
Request);
    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);
    /*（可选）请参见如下示例设置访问权限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;
}
```

说明:

获取已经上传的分片，调用`CompleteMultipartUpload`**接口时需要每个分片的 ETag 值。**

ETag 值的获取有两种途径：

- **第一种是上传每个分片的时候，返回结果里面会包含这个分片的 ETag 值，可以保存使用。**
- **第二种是通过调用 ListParts 接口获取已经上传的分片的 ETag 值。上述示例采用的是第一种方式。**

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，请使用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，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，请使用以下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(listmultiu
ploadrequest);
        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().MultipartU
ploadList()) {
                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(listmultiu
ploadrequest);
        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().MultipartU
ploadList()) {
                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

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(initUpload
Request);
    auto uploadId = uploadIdResult.result().UploadId();

    /* 取消分片上传 */
    AbortMultipartUploadRequest  abortUploadRequest(BucketName, key,
uploadId);
    auto abortUploadIdResult = client.AbortMultipartUpload(abortUploa
dRequest);
```

```
    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;
}
```

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。

5.11.6 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。**

5.11.7 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::in|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;
}
```

5.11.8 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。

5.12 下载文件

5.12.1 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。

5.12.2 下载到本地文件

本文介绍如何将 OSS 文件下载到本地文件。

以下代码用于把指定的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::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" << 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;
}
```

5.12.3 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*/
    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);
    auto outcome = client.GetObject(request);
    if (outcome.isSuccess()) {
      std::cout << "getObjectToBuffer" << " success, Content-Length:"
 << outcome.result().Metadata().ContentLength() << 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;
}
```

5.12.4 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";
    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;
```

```
}
```

5.12.5 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, DownloadFi
lePath, 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;
}
```

5.12.6 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*GitHub*。

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;
}
```

5.13 管理文件

5.13.1 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**

5.13.2 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。	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

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);
    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;
```

```
}
```

5.13.3 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。

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.ContentLen
gth() << 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*。

5.13.4 转换文件存储类型

OSS提供标准、低频访问、归档三种存储类型，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。

有关标准存储、低频访问、归档存储三种存储类型的更多信息，请参考开发指南的*#unique_51***及***#unique_90***。**

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 */
    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, SourceObje
ctName);
```

```
        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(), onGoingRes
tore)==0) {
            std::cout << " success, restore status:" << restoreSta
tus << 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，请参考计量项和计费项的*#unique_91/unique_91_Connect_42_section_uz2_m2t_lgb*一节。

5.13.5 列举文件

本文介绍如何列举文件。

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.isSucces s())  {
        /* 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 = "";
    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().CommonPref
ixes()) {
            std::cout << "commonPrefix" << ",name:" << commonPrefix
 << std::endl;
        }
        nextMarker = outcome.result().NextMarker();
    } while (outcome.result().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>
using namespace AlibabaCloud::OSS;

int main(void)
{
    static int64_t calculateFolderLength (const OssClient &client,
const std::string &bucketName, const std::string &folder)
    {
        std::string nextMarker = "";
        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();
      } while (outcome.result().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 = "";
    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().CommonPref
ixes()) {
            int64_t foldersize = calculateFolderLength(client,
BucketName, commonPrefix);
            std::cout << "folder" <<
            ",name:" << commonPrefix <<
            ",size:" << foldersize << std::endl;
        }
        nextMarker = outcome.result().NextMarker();
    } while (outcome.result().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 = "";
    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();
    } while (outcome.result().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);

    do {
        /* 列举文件 */
        ListObjectsRequest request(BucketName);
         /* 列举指定marker之后的文件 */
        request.setMarker(YourMarker);
        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;
        }
        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 = "";
    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();
    } while (outcome.result().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），显示为Objects。
- 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），该文件夹下的子文件夹（目录）则显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见*#unique_92/unique_92_Connect_42_section_bby_5yy_5db*。创建文件夹的完整代码请参见*GitHub*。

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。下面的示例展示了如何模拟文件夹功能。

- 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);


    ListObjectOutcome outcome;

    std::string nextMarker = "";

    do {
        /*列举文件*/
        ListObjectsRequest request(BucketName);
        request.setMarker(nextMarker);
        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().ObjectSumm
arys()) {
                std::cout << "object"<<
                ",name:" << object.Key() <<
                ",size:" << object.Size() <<
                ",lastmodify time:" << object.LastModified() << std
::endl;
            }
        }
        nextMarker = outcome.result().NextMarker();
    } while (outcome.result().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);

    ListObjectOutcome outcome;
    std::string nextMarker = "";
    do {
            /*列举文件*/
            ListObjectsRequest request(BucketName);
            request.setPrefix("fun/");
            request.setMarker(nextMarker);
            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().ObjectSumm
arys()) {
                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();
    } while (outcome.result().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);

    ListObjectOutcome outcome;
    std::string nextMarker = "";
    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);
            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().ObjectSumm
arys()) {
                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();
    } while (outcome.result().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::string &folder)
{
    std::string nextMarker = "";
    int64_t size = 0;
    ListObjectOutcome outcome;
    do {
            /*列举文件*/
            ListObjectsRequest request(bucketName);
            request.setPrefix(folder);
            request.setMarker(nextMarker);
            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().ObjectSumm
arys()) {
                size += object.Size();
            }

            nextMarker = outcome.result().NextMarker();
        } while (outcome.result().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);

    ListObjectOutcome outcome;
    std::string nextMarker = "";
    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);
        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().CommonPref
ixes()) {
            int64_t foldersize = calculateFolderLength(client,
BucketName, commonPrefix);
            std::cout << "folder" <<
            ",name:" << commonPrefix <<
            ",size:" << foldersize << std::endl;
        }
        nextMarker = outcome.result().NextMarker();
    } while (outcome.result().IsTruncated());

    /*释放网络等资源*/
    ShutdownSdk();
    return 0;
}
```

5.13.6 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

以下代码用于删除单个文件：

```
#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);

    /* 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 */
    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;
}
```

删除多个文件

以下代码用于批量删除文件：

```
#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;
}
```

5.13.7 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 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）。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：

CopyObject 的限制条件如下：

- **用户有源文件的读写权限。**
- **不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。**
- **拷贝的文件大小不能超过 1GB。**

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 1GB 的文件，需要使用分片拷贝（UploadPartCopy）。分片拷贝分为三步：通过 ossClient.initiateMultipartUpload 初始化分片拷贝任务。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。通过 ossClient.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(initUpload
Request);
    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, SourceBucketName, SourceObjectName,uploadId, i + 1);
        uploadPartCopyReq.setCopySourceRange(skipBytes, skipBytes +
size -1);
        auto uploadPartOutcome = client.UploadPartCopy(uploadPart
CopyReq);
        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, CopyObject
Name, 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;
}
```

5.13.8 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;
```

```
}
```

5.13.9 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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 */
    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;
}
```

归档存储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*。

5.13.10 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：

```
#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*。

5.14 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

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;
}
```

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，您可以指定URL的过期时间，来限制访客的访问时长。

- **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有效时间 */
    std::time_t t = std::time(nullptr) + 1200;
    /* 生成上传文件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有效时间*/
    std::time_t t = std::time(nullptr) + 1200;
    /* 生成下载文件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;
}
```

5.15 生命周期

本文介绍如何管理生命周期规则（Lifecycle）。

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。每条规则包含：

- 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。
- 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。
- 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。
- 是否生效。

通过oss_upload_file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");

    /* 指定过期天数 */
    auto rule1 = LifecycleRule();
    rule1.setID("rule1");
    rule1.setPrefix("test1/");
    rule1.setStatus(RuleStatus::Enabled);
    rule1.setExpiration(3);

    /* 设置转换存储类型的时间 */
    auto transition = LifeCycleTransition();
    transition.Expiration().setDays(180);
    transition.setStorageClass(StorageClass::IA);
    rule1.addTransition(transition);

    transition.Expiration().setDays(365);
    transition.setStorageClass(StorageClass::Archive);
    rule1.addTransition(transition);

    /* 设置生命周期规则 */
    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

以下代码用于查看生命周期规则：

```
#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, c onf);

    /* 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;
        }
    }
    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;
}
```

5.16 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;
}
```

查看跨域资源共享规则

以下代码用于查看跨域资源共享规则：

```
#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);


    /* 查看跨域资源共享规则 */
    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;
```

```
}
```

5.17 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;
}
```

5.18 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;
}
```

5.19 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;
}
```

5.20 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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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读写，必须通过授权才能进行访问。

- **匿名访问**

  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

参数	描述
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

  授权访问支持自定义样式、HTTPS和级联处理。

  以下代码用于生成带签名的图片处理URL：

  ```
  #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;
}
```

- **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：

```
#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
);

    /* 缩放 */
    std::string Process = "image/resize,m_fixed,w_100,h_100";
    GetObjectRequest request(BucketName, ObjectName);
    request.setProcess(Process);
```

```
    auto outcome = client.GetObject(request);

    /* 裁剪 */
    Process = "image/crop,w_100,h_100,x_100,y_100,r_1";
    request.setProcess(Process);
    outcome = client.GetObject(request);

    /* 旋转 */
    Process = "image/rotate,90";
    request.setProcess(Process);
    outcome = client.GetObject(request);

    /* 锐化 */
    Process = "image/sharpen,100";
    request.setProcess(Process);
    outcome = client.GetObject(request);

    /* 水印 */
    Process = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
    request.setProcess(Process);
    outcome = client.GetObject(request);

    /* 格式转换 */
    Process = "image/format,png";
    request.setProcess(Process);
    outcome = client.GetObject(request);

    /* 获取图片信息 */
    request = request(BucketName, ObjectName, "image/info") ;
    outcome = client.GetObject(request);

    /* 释放网络等资源 */
    ShutdownSdk();
    return 0;
}
```

- **自定义样式**

以下代码用于自定义图片样式：

```
#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";
    std::string ObjectName = "yourObjectName";

     /* 初始化网络等资源 */
    InitializeSdk();

    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf
) ;

    /* 自定义样式 */
    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;
}
```

- **级联处理**

  以下代码用于级联处理图片：

```
#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
);

    /* 级联处理 */
    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;
}
```

图片处理持久化

以下代码用于图片处理持久化：

```
#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";
    std::string SourceObjectName = "yourSourceObjectName";
    std::string TargetObjectName = "yourTargetObjectName";

     /* 初始化网络等资源 */
```

```
    InitializeSdk();

    ClientConfiguration conf;
    OssClient client(Endpoint, AccessKeyId, AccessKeySecret, conf);

    /* 缩放 */
    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);

    /* 裁剪 */
    Process = "image/crop,w_100,h_100,x_100,y_100,r_1";
    ss.str("");
    ss  << Process
    <<"|sys/saveas"
    << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
    << ",b_" << Base64EncodeUrlSafe(BucketName);
    request = request(BucketName, SourceObjectName, ss.str());
    outcome = client.ProcessObject(request);

    /* 旋转 */
    Process = "image/rotate,90";
    ss.str("");
    ss  << Process
    <<"|sys/saveas"
    << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
    << ",b_" << Base64EncodeUrlSafe(BucketName);
    request = request(BucketName, SourceObjectName, ss.str());
    outcome = client.ProcessObject(request);

    /* 锐化 */
    Process = "image/sharpen,100";
    ss.str("");
    ss  << Process
    <<"|sys/saveas"
    << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
    << ",b_" << Base64EncodeUrlSafe(BucketName);
    request = request(BucketName, SourceObjectName, ss.str());
    outcome = client.ProcessObject(request);

    /* 水印 */
    Process = "image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ";
    ss.str("");
    ss  << Process
    <<"|sys/saveas"
    << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
    << ",b_" << Base64EncodeUrlSafe(BucketName);
    request = request(BucketName, SourceObjectName, ss.str());
    outcome = client.ProcessObject(request);

    /* 格式转换 */
    Process = "image/format,png";
    ss.str("");
    ss  << Process
    <<"|sys/saveas"
    << ",o_" << Base64EncodeUrlSafe(TargetObjectName)
    << ",b_" << Base64EncodeUrlSafe(BucketName);
    request = request(BucketName, SourceObjectName, ss.str());
```

```
    outcome = client.ProcessObject(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。

6 PHP

6.1 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、删除、列举、权限等
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

6.2 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**

  1. **在项目的根目录运行`composer require aliyuncs/oss-sdk-php`，或者在`composer.json`文件中添加依赖关系如下：**

     ```
     "require": {
         "aliyuncs/oss-sdk-php": "~2.x.x"
     }
     ```

  2. **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

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

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

- 源码方式

  1. 在GitHub中选择相应版本并下载打包好的zip文件。
  2. 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';
```

6.3 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。 以下代码用于新建一个存储空间：

```
<?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，请参见基本概念中的命名规范。创建存储空间详情，请参见#unique_41。

获取endpoint信息，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。

6.4 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*RAM*和*STS*介绍和授权访问。

- **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://<您的用户名>:<您的密码>@<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();
}
```

6.5 存储空间

存储空间（Bucket）是对象（Object）的容器，对象必须隶属于某个存储空间。

以下场景的完整代码请参见*GitHub*。

创建存储空间

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);
    // 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");
```

存储空间的命名规范，请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定存储空间的权限和存储类型。

判断存储空间是否存在

以下代码用于判断指定的存储空间是否存在：

```
<?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");
}
```

列举存储空间

以下代码用于列举存储空间：

```
<?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";
try{
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

    $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");
}
```

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");
```

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);
```

获取存储空间的地域

以下代码用于获取存储空间的地域（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);
```

关于地域的详细信息请参见基本概念中的地域。

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);
```

删除存储空间

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。

说明:

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：

```
<?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");
```

6.6 上传文件

6.6.1 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。

6.6.2 简单上传

本文介绍如何使用简单上传。

简单上传分为字符串上传和文件上传两部分。

字符串上传

以下代码用于上传字符串到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 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>";
// 文件名称
$object = "<yourObjectName>";
// 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");
```

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 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>";
// 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");
```

6.6.3 追加上传

追加上传支持两种上传类型，分别为字符串和文件。

追加类型的文件（Append Object）暂时不支持copyObject操作。

追加上传字符串

以下代码用于追加上传字符串到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 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>";
// 文件名称
$object = "<yourObjectName>";
// 文件内容
$content_array = array('Hello OSS', 'Hi OSS', 'OSS OK');
try{
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

    $position = $ossClient->appendObject($bucket, $object, $content_ar
ray[0], 0);
    $position = $ossClient->appendObject($bucket, $object, $content_ar
ray[1], $position);
    $position = $ossClient->appendObject($bucket, $object, $content_ar
ray[2], $position);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

追加上传文件

以下代码用于追加上传本地文件到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 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>";
// 文件名称
$object = "<yourObjectName>";
// 本地文件1
$filePath = "<yourLocalFile1>";
// 本地文件2
$filePath1 = "<yourLocalFile2>";
try{
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

    $position = $ossClient->appendFile($bucket, $object, $filePath, 0
);
    $position = $ossClient->appendFile($bucket, $object, $filePath1, $
position);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

6.6.4 分片上传

本文介绍如何使用分片上传。

分片上传分为以下三个步骤：

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。

   > **注意：**
   >
   > - 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。
   > - 除了最后一块Part以外，其他的Part最小为100KB。最后一块Part没有大小限制。

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*。

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 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。
 */
try{
    $ossClient = new OssClient($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);
} 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;
$uploadFileSize = filesize($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' => $eTag,
    );
}
/**
 * 步骤3：完成上传。
 */
try {
 // 在执行该操作时，需要提供所有有效的$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);
}  catch(OssException $e) {
    printf(__FUNCTION__ . ": completeMultipartUpload FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
printf(__FUNCTION__ . ": completeMultipartUpload OK\n");
```

上述例子中的$options包含的参数如下：

参数	描述
$ossClient::OSS_FILE_UPLOAD	上传文件
OssClient::OSS_PART_NUM	分片号
OssClient::OSS_SEEK_TO	指定开始位置
OssClient::OSS_LENGTH	文件长度
OssClient::OSS_PART_SIZE	分片大小
OssClient::OSS_CHECK_MD5	是否开启MD5校验，true为开启

取消分片上传

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：

```
<?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>";
$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");
```

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 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>";
$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 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>";
$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");
```

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 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>";
$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
<?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>";

$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的参数说明如下：

参数	说明
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。
key-marker	所有文件名称的字母序大于key-marker参数值的分片上传事件。可以与upload-id-marker参数一同使用来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数目，默认值和最大值均为1000。
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
upload-id-marker	与key-marker参数一同使用来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： · 名称的字母序大于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。

6.6.5 上传回调

本文介绍如何使用上传回调。

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。
// callbackHost为回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
$url =
    '{
        "callbackUrl":"<yourCallbackServerUrl>",
        "callbackHost":"<yourCallbackServerHost>",
        "callbackBody":"bucket=${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\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;
$uploadFileSize = filesize($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。
// callbackHost为回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
$json =
    '{
        "callbackUrl":"<yourCallbackServerUrl>",
        "callbackHost":"<yourCallbackServerHost>",
        "callbackBody":"{\"mimeType\":${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");
```

6.7 下载文件

6.7.1 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**

6.7.2 下载到本地文件

本文介绍如何使用PHP SDK将指定的文件下载到本地。

以下代码用于把指定的OSS文件下载到本地文件：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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 表示您在下载文件时需要指定的文件名称，如abc/efg/123.jpg。
$object = "<yourObjectName>";
// 指定文件下载路径。
$localfile = "<yourLocalFile>";
$options = array(
        OssClient::OSS_FILE_DOWNLOAD => $localfile
    );

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");
```

6.7.3 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。

6.7.4 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");
```

6.7.5 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

参数	描述	如何设置
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
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");
```

6.8 管理文件

6.8.1 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**

6.8.2 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断指定的文件是否存在：

```
<?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);
```

6.8.3 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);
```

6.8.4 管理文件元信息

本文介绍如何管理文件元信息。

文件元信息（Object Meta）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：

```
<?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>";
$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 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";
$fromBucket = "<yourFromBucketName>";
$fromObject = "<yourFromObjectName>";
$toBucket = "<yourToBucketName>";
$toObject = "<yourToObjectName>";
$copyOptions = array(
    OssClient::OSS_HEADERS => array(
        'Expires' => '2018-10-01 08:00:00',
        'Content-Disposition' => 'attachment; filename="xxxxxx"',
        'x-oss-meta-location' => 'location',
    ),
);
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 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);

    $objectMeta = $ossClient->getObjectMeta($bucket, $object);
} catch (OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
if (isset($objectMeta[strtolower('Content-Disposition')]) &&
    'attachment; filename="xxxxxx"' === $objectMeta[strtolower('
Content-Disposition')]
) {
    print(__FUNCTION__ . ": ObjectMeta checked OK" . "\n");
} else {
    print(__FUNCTION__ . ": ObjectMeta checked FAILED" . "\n");
}
```

6.8.5 列举文件

本文介绍如何列举文件。

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。

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**

6.8.6 删除文件

本文介绍如何删除文件。

删除单个文件

以下代码用于删除单个文件：

```
<?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>";
$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

以下代码用于批量删除文件：

```
<?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。

6.8.7 拷贝文件

本文介绍如何拷贝文件。

简单拷贝

以下代码用于拷贝小于1GB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";
$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");
```

拷贝大文件

对于大于1GB的文件，需要使用分片拷贝。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>";

try{
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

    // 初始化分片。
    $upload_id = $ossClient->initiateMultipartUpload($dst_bucket, $
dst_object);

    $copyId = 1;
    // 逐个分片拷贝。
    $eTag = $ossClient->uploadPartCopy( $src_bucket, $src_object, $
dst_bucket, $dst_object,$copyId, $upload_id);
    $upload_parts[] = array(
        'PartNumber' => $copyId,
        'ETag' => $eTag,
    );

    // 完成分片拷贝。
    $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");
```

6.8.8 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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 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);

    $ossClient->restoreObject($bucket, $object);
} catch (OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

存储类型详情请参见存储类型。

6.8.9 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。

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*。

6.8.10 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。

6.9 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

以下代码用于使用STS临时授权上传文件：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>";
$securityToken = "<yourSecurityToken>";

$content = "Hi, OSS.";

try {
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint, false, $securityToken);

    $ossClient->putObject($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，您可以指定URL的过期时间，来限制访客的访问时长。

授权访问的完整代码请参见*GitHub*。

- **生成下载的签名URL**

  以下代码用于生成下载的签名URL：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>";
$securityToken = "<yourSecurityToken>";

// 设置URL的有效期为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");

// 可以使用代码来访问签名的URL，也可以输入到浏览器中进行访问。
$request = new RequestCore($signedUrl);
// 生成的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");
};
```

- **生成上传的签名URL**

  以下代码用于生成上传的签名URL：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用
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>";
$securityToken = "<yourSecurityToken>";

$timeout = 3600;
try {
    $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint, false, $securityToken);

    // 生成PutObject的签名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");

$content = "Hello OSS.";
$request = new RequestCore($signedUrl);
// 生成的URL以PUT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");
};
```

6.10 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。

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");
```

6.11 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- 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。
- 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。
- 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。
- 是否生效。

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 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 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>";

$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 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);

    $ossClient->deleteBucketLifecycle($bucket);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
```

```
print(__FUNCTION__ . ": OK" . "\n");
```

6.12 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");
```

6.13 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跨域访问。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：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>";

$corsConfig = new CorsConfig();
$rule = new CorsRule();
```

```
// AllowedHeaders和ExposeHeaders不支持通配符。
$rule->addAllowedHeader("x-oss-header");
// AllowedOlowedMethods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。
$rule->addAllowedOrigin("http://www.b.com");
$rule->addAllowedMethod("POST");
$rule->setMaxAgeSeconds(10);
// 每个存储空间最多允许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;

/// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>";

$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

以下代码用于删除指定存储空间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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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);

    $ossClient->deleteBucketCors($bucket);
} catch(OssException $e) {
    printf(__FUNCTION__ . ": FAILED\n");
    printf($e->getMessage() . "\n");
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
```

6.14 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");
```

6.15 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");
```

6.16 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。

图片处理的详细信息请参见*OSS图片处理指南*。图片处理的完整代码请参见*GitHub*。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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读写，必须通过授权才能进行访问。

- **匿名访问**

  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

参数	描述
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

  授权访问支持自定义样式、 HTTPS和级联处理。

  以下代码用于生成带签名的图片处理URL：

```
<?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
);

// 生成一个带签名的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);
```

- **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：

```
<?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>";
$download_file = "download.jpg";
```

```
$ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

// 上传示例图片。
$ossClient->uploadFile($bucket, $object, "example.jpg");

// 图片缩放
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/resize,m_fixed,h_100,w_100
" );
$ossClient->getObject($bucket, $object, $options);

// 图片裁剪
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/crop,w_100,h_100,x_100,y_100,
r_1" );
$ossClient->getObject($bucket, $object, $options);

// 图片旋转
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/rotate,90" );
$ossClient->getObject($bucket, $object, $options);

// 图片锐化
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/sharpen,100" );
$ossClient->getObject($bucket, $object, $options);

// 图片水印
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/watermark,text_SGVsbG8g5Zu-
54mH5pyN5YqhIQ" );
$ossClient->getObject($bucket, $object, $options);

// 图片格式转换
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/format,png" );
$ossClient->getObject($bucket, $object, $options);

// 获取图片信息。
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => "image/info" );
$ossClient->getObject($bucket, $object, $options);

// 删除示例图片。
$ossClient->deleteObject($bucket, $object);
```

- **自定义样式**

  以下代码用于自定义图片样式：

```
<?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;

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用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>";
$download_file = "download.jpg";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);

// 上传示例图片。
$ossClient->uploadFile($bucket, $object, "example.jpg");

// 自定义样式。
$style = "style/oss-pic-style-w-300";
$options = array(
    OssClient::OSS_FILE_DOWNLOAD => $download_file,
    OssClient::OSS_PROCESS => $style);
$ossClient->getObject($bucket, $object, $options);

// 删除示例图片。
$ossClient->deleteObject($bucket, $object);
```

- **级联处理**

  以下代码用于级联处理图片：

```
<?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 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>";
$download_file = "download.jpg";

$ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint, false);

// 上传示例图片。
$ossClient->uploadFile($bucket, $object, "example.jpg");
```

```
// 级联处理。
$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);

// 删除示例图片。
$ossClient->deleteObject($bucket, $object);
```

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。

6.17 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。
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
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

7 C

7.1 前言

本文档基于OSS C SDK 3.6.0编写。

兼容性

- 对于3.*.*系列SDK：兼容。
- 对于 2.*.*系列SDK：
  - Windows兼容。
  - Linux接口兼容，链表(aos_list_t)遍历接口不兼容。
    - os_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：以下结构体和接口不兼容，其余都兼容。
  - 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.*系列SDK：不兼容。

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
oss_put_object_sample	上传文件

示例文件	示例内容
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
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

7.2 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

名称	描述	版本要求
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

  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Debug。
-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。**

7.3 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* 。

  1. **下载并解压示例工程。**
  2. **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。**
  3. **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。**
  4. **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***。**

  1. **下载并解压示例工程。**
  2. **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_N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。 以下代码用于新建一个存储空间：

```
#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);
    /* 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;
}
```

获取endpoint信息，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

以下代码用于下载指定文件：

```
#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>";
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) {
            ++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;
}
```

7.4 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);
}
int main() {
    aos_pool_t *p;
    oss_request_options_t *options;
    /* 全局变量初始化，程序启动时，最先运行。*/
    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详见*OSS C SDK*如何设置通信时和*CURL*相关的一些参数。

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;
    /* 全局变量初始化，程序启动时，最先运行。*/
    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);
    /* 是否使用了CNAME。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;
    /* 全局变量初始化，程序启动时，最先运行。*/
    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);
    /* 设置链接超时，默认是10秒。*/
    options->ctl->options->connect_timeout = 10;
    /* 设置DNS超时，默认是60秒。*/
    options->ctl->options->dns_cache_timeout = 60;
    /*
    设置请求超时：
    通过设置speed_limit的值控制能容忍的最小速率，默认是1024，即1KB/s。
    通过设置speed_time的值控制能容忍的最长时间，默认是15秒。
    表示如果传输速率连续15秒小于1KB/s，则超时。
    */
    options->ctl->options->speed_limit = 1024;
    options->ctl->options->speed_time = 15;
}
```

7.5 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。

创建存储空间

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;
    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;
}
```

存储空间的命名规范，请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定**存储空间的权限**和**存储类型**。

以下代码用于在创建存储空间时指定存储空间的权限：

```
    /* 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");
    }
```

以下代码用于创建归档类型的存储空间：

```
    resp_status = oss_create_bucket_with_storage_class(oss_client
_options, &bucket, OSS_ACL_PRIVATE, OSS_STORAGE_CLASS_ARCHIVE);
    /* 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");
    }
```

以下代码用于创建低频访问类型的存储空间：

```
    resp_status = oss_create_bucket_with_storage_class(oss_client
_options, &bucket, OSS_ACL_PRIVATE, OSS_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");
    }
```

列举存储空间

以下代码用于列举所有的存储空间：

```
#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。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_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;
}
```

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
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 "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。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;
}
```

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。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;
}
```

获取存储空间的地域

以下代码用于获取存储空间的地域（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。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;
}
```

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）：

```
#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。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);
    /* 获取存储空间的地域。*/
    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;
}
```

删除存储空间

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。

说明:

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;
    /* 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;
```

```
}
```

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 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。

7.6.2 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* 。

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);
    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;
}
```

7.6.3 追加上传

本文介绍如何使用追加上传。

追加类型的文件（Append Object）暂时不支持copyObject操作。

从内存中追加数据

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;
    /* 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, headers2, &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;
}
```

从本地文件追加数据

以下代码用于从本地文件追加数据：

```
#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 *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。*/
    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, headers2, &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;
}
```

如果文件存在，会出现以下两种情况：

- 不可追加文件，则抛出ObjectNotAppendable异常。
- 可追加文件，通过Put Object操作上传，将覆盖原有文件，文件类型变为Normal Object。Head Object操作会返回x-oss-object-type，用于表明Object的类型。

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。

在上传的过程中会在Checkpoint文件中记录当前上传的进度信息，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。

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。

以下代码用于断点续传上传：

```
#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;
}
```

7.6.5 分片上传

本文介绍如何使用分片上传。

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，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oss_complete_multipart_upload方法将所有分片合并成完整的文件。

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);
}
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(p, 1);
    complete_headers = aos_table_make(p, 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, &resp_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。*/
    int64_t file_length = 0;
    int64_t pos = 0;
    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);
        if (aos_status_is_ok(resp_status)) {
            printf("Multipart upload part from file succeeded\n");
        } else {
            printf("Multipart upload part from file failed\n");
        }
    }
    oss_list_upload_part_params_t *params = NULL;
    aos_list_t complete_part_list;
    /* 获取已经上传的分片。*/
    params = oss_create_list_upload_part_params(pool);
    params->max_ret = 1000;
    aos_list_init(&complete_part_list);
    resp_status = oss_list_upload_part(oss_client_options, &bucket, &
object, &upload_id, params, &resp_headers);
    /* 判断分片列表是否获取成功。*/
    if (aos_status_is_ok(resp_status)) {
        printf("List multipart succeeded\n");
    } else {
        printf("List multipart failed\n");
    }
    oss_complete_part_content_t *complete_part_content = NULL;
    oss_list_part_content_t *part_content = NULL;
    aos_list_for_each_entry(oss_list_part_content_t, part_content, &
params->part_list, node) {
        complete_part_content = oss_create_complete_part_content(pool
);
        aos_str_set(&complete_part_content->part_number, part_content-
>part_number.data);
        aos_str_set(&complete_part_content->etag, part_content->etag.
data);
        aos_list_add_tail(&complete_part_content->node, &complete_p
art_list);
    }
    /* 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;
}
```

说明:

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。ETag值获取有两种途径：第一种是上传每个分片的时候，返回结果里面会包含这个分片的ETag值，可以保存使用。第二种是通过调用oss_list_upload_part接口获取已经上传的分片的ETag值。上述示例采用的是第二种方式。

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 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, &resp_headers);
    /* 判断是否初始化分片上传成功。 */
    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);
    }
    /* 取消这次分片上传。*/
    resp_status = oss_abort_multipart_upload(oss_client_options, &
bucket, &object, &upload_id, &resp_headers);
    /* 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。

7.6.6 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*GitHub*。

下面的代码以PutObject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, 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");
    }
    /* 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。

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*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。*/
    char *callback =  "{"
        "\"callbackUrl\":\"http://oss-demo.aliyuncs.com:23450\","
        "\"callbackHost\":\"oss-cn-hangzhou.aliyuncs.com\","
        "\"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;
}
```

上传回调详情请参见上传回调，调试方法和错误排除请参见上传回调错误及排除。

7.7 下载文件

7.7.1 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。

7.7.2 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*。

以下代码用于把指定的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 *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_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。

7.7.3 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。*/
    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_heade
rs);
    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;
}
```

7.7.4 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。

以下代码用于范围下载：

```
#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>";
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;
}
```

7.7.5 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。
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, clt_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;
```

```
}
```

7.7.6 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*GitHub*。

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。

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：

- **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**

7.8.2 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。

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 "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;
    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;
}
```

7.8.3 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。

获取文件元信息

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_AUTHOR
IZATION);
        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_author, object_type, content_le
ngth);
    } 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*。

7.8.4 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_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);
    /* 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
);
    /* 是否使用了CNAME */
    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,
acces_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);
    /* 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
);
    /* 是否使用了CNAME */
    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,
acces_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);
    /* 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
);
    /* 是否使用了CNAME */
    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,
acces_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），显示为Objects。
- 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），该文件夹下的子文件夹（目录）则显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

文件夹详情请参见*#unique_92/unique_92_Connect_42_section_bby_5yy_5db*。创建文件夹的完整代码请参见*GitHub*。

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。下面的示例展示了如何模拟文件夹功能。

- 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 "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_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;
}
```

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 "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);
    /* 是否使用了CNAME。*/
    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,
acces_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 = "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;
}
```

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 "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);
    /* 是否使用了CNAME。*/
    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,
acces_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;
    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_prefix_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;
}
```

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 "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);
    /* 是否使用了CNAME。*/
    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,
acces_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;
    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_prefix_list, node) {
            line = apr_psprintf(pool, "%.*s", commonPrefix->prefix.
len, commonPrefix->prefix.data);
            int64_t size = calculateFolderLength(bucket, commonPref
ix->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;
}
```

7.8.5 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，文件一旦删除将无法恢复。**

删除文件的完整代码请参见*GitHub*。

删除单个文件

以下代码用于删除单个文件：

```
#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_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

以下代码用于批量删除文件：

```
#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;
    /* 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, &deleted_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>";
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。

7.8.6 拷贝文件

本文介绍如何拷贝文件。

拷贝小文件

此操作的限制条件如下：

- **用户有源文件的读写权限。**
- **不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。**
- **文件的大小不能超过1GB。**

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, &dest_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_buc
ket_name);
    aos_str_set(&upload_part_copy_params1->source_object, source_obj
ect_name);
    aos_str_set(&upload_part_copy_params1->dest_bucket, dest_bucke
t_name);
    aos_str_set(&upload_part_copy_params1->dest_object, dest_objec
t_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_par
t_copy_params1, headers, &resp_headers);
    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_buc
ket_name);
    aos_str_set(&upload_part_copy_params2->source_object, source_obj
ect_name);
    aos_str_set(&upload_part_copy_params2->dest_bucket, dest_bucke
t_name);
    aos_str_set(&upload_part_copy_params2->dest_object, dest_objec
t_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_par
t_copy_params2, headers, &resp_headers);
    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_bucke
t, &dest_object, &upload_id, list_upload_part_params, &list_part_
resp_headers);
    aos_list_for_each_entry(oss_list_part_content_t, part_content, &
list_upload_part_params->part_list, node) {
        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;
}
```

7.8.7 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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 "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 {
            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;
}
```

归档存储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*。

7.8.8 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 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 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*。

7.9 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*RAM和STS应用场景实践*。

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;
 /* 将类型为char*的数据赋值给bucket。*/
 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;
}
```

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，您可以指定URL的过期时间，来限制访客的访问时长。

- **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_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**

  以下代码用于使用签名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 *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_headers);
    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;
}
```

7.10 生命周期

本文介绍如何管理生命周期规则。

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。每条规则包含：

- 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。
- 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。
- 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。
- 是否生效。

通过oss_upload_file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_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 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("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_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;
}
```

7.11 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。

图片处理的详细信息请参见*OSS***图片处理指南**。

图片处理的完整代码请参见：*GitHub*。

图片处理功能

OSS图片处理提供以下功能：

- **获取图片信息**
- **图片格式转换**
- **图片缩放**
- **图片裁剪**
- **图片旋转**
- **图片效果**
- **图片水印，包括添加图片、文字、图文混合水印**
- **自定义图片处理样式**
- **级联处理，调用多个图片处理功能**

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读写，必须通过授权才能进行访问。

- **匿名访问**

  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

参数	描述
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

授权访问支持自定义样式、HTTPS和级联处理。

以下代码用于生成带签名的图片处理URL：

```
#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);
    /* 图片处理。*/
    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;
    /* 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;
}
```

- **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：

```
#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);
    /* 缩放 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100
,h_100");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-resize.jpg");
    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");
    }
    /* 裁剪 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/crop,w_100,h_100,
x_100,y_100,r_1");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-crop.jpg");
    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");
    }
    /* 旋转 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/rotate,90");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-rotate.jpg");
    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");
    }
    /* 锐化 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/sharpen,100");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-sharpen.jpg");
    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");
    }
    /* 水印 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/watermark,text_SGVsb
G8g5Zu-54mH5pyN5YqhIQ");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-watermark.jpg");
    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");
    }
```

```
    /* 格式转换 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/format,png");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-format.png");
    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");
    }
    /* 获取图片信息 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/info");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-info.txt");
    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;
}
```

- **自定义样式**

  以下代码用于自定义图片样式：

```
#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);
    /* 自定义样式 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "style/<yourCustomStyleName
>");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-custom-style.jpg");
    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;
}
```

- **级联处理**

  以下代码用于级联处理图片：

```
#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。*/
    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);
    /* 级联处理 */
    params = aos_table_make(pool, 1);
    apr_table_set(params, OSS_PROCESS, "image/resize,m_fixed,w_100
,h_100/rotate,90");
    /* 下载处理后的图片到本地文件。*/
    aos_str_set(&file, "example-cascade.jpg");
    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;
}
```

图片处理工具

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

7.12 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;
}
```

7.13 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。

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;
}
```

7.14 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：

```
#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_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_co
nfig.referer_list, node) {
            printf("get referer %s\n", referer->referer.data);
```

```
        }
    } 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_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;
}
```

7.15 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：

```
#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;
}
```

7.16 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
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
TooManyBuckets	用户的存储空间数超过限制	400

7.17 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

8 .NET

8.1 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**

示例文件	示例内容
SetObjectAclSample.cs	管理文件访问权限
ListObjectsSample.cs	列举文件
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

8.2 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**
    1. **如果您的Visual Studio没有安装NuGet，请先安装*NuGet*。**
    2. **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。**
    3. **搜索 aliyun.oss.sdk，在结果中找到 Aliyun.OSS.SDK 或 Aliyun.OSS.SDK.NetCore，选择最新版本，单击安装。**
  - **DLL引用方式安装**
    1. **下载并解压C# SDK开发包。**
    2. **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。**
    3. **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，希望源码安装，操作步骤如下：

    1. **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。**
    2. **在弹出的对话框中选择`aliyun-oss-sdk.csproj`文件，单击打开。**
    3. **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**
    1. **在Xamarin中新建或者打开已有的项目，选择工具Add NuGet Packages。**
    2. **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**
    1. **下载并解压C# SDK开发包。**
    2. **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。**
    3. **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 。**

8.3 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。

以下代码用于新建一个存储空间：

```
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，请参见基本概念中的命名规范。

获取endpoint信息，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, localFilen
ame);
    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
{
    client.PutObject(bucketName, objectName, localFilename);
    // 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：

```
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);
}
```

8.4 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);
```

8.5 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。

创建存储空间

创建存储空间的完整代码请参见*GitHub*。

以下代码用于创建存储空间：

```
using Aliyun.OSS;

// 初始化OssClient。
var client = new OssClient(endpoint, accessKeyId, accessKeySecret);

// 创建存储空间（Bucket）。
public void CreateBucket(string bucketName)
{
    try
    {
        // 创建存储空间。BucketName具有全局唯一性。
        client.CreateBucket(bucketName);
        Console.WriteLine("Create bucket succeeded");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Create bucket failed. {0}", ex.Message);
    }
}
```

存储空间的命名规范，请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定存储空间的权限和存储类型。

列举存储空间

列举存储空间的完整代码请参见*GitHub*。

以下代码用于列举所有的存储空间：

```
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, bucket.Owner);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("List bucket failed. {0}", ex.Message);
    }
}
```

判断存储空间是否存在

判断存储空间的完整代码请参见GitHub。

以下代码用于判断指定的存储空间是否存在：

```
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
);
    }
}
```

设置存储空间访问权限

设置访问权限的完整代码请参见GitHub。

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文件的读写权限，其他用户没有权限操作文件。	oss.ACLPrivate

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：

```
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);
    }
}
```

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);
    }
}
```

删除存储空间

删除存储空间的完整代码请参见*GitHub*。

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。

> **说明：**
> **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：

```
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);
    }
}
```

8.6 上传文件

8.6.1 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。

8.6.2 简单上传

本文介绍如何使用简单上传。

上传字符串

上传字符串的完整代码请参见*GitHub*。

以下代码用于上传字符串：

```
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。
    client.PutObject(bucketName, objectName, requestContent);
    Console.WriteLine("Put object succeeded");
}
catch (Exception ex)
{
    Console.WriteLine("Put object failed, {0}", ex.Message);
}
```

上传指定的本地文件

上传本地文件的完整代码请参见*GitHub*。

以下代码用于上传本地文件：

```
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。
    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

为保证SDK发送的数据和OSS服务端接收到的数据一致，可以在ObjectMeta中增加Content-Md5值，这样服务端就会使用这个MD5值做校验。

上传文件并且带MD5校验的完整代码请参见*GitHub*。

```
using Aliyun.OSS;
using Aliyun.OSS.Util;
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
{
    // 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);
}
```

说明:

使用MD5时，性能会有所下降。

异步上传

异步上传的完整代码请参见*GitHub*。

以下代码用于异步上传：

```
using System;
using System.IO;
using System.Threading;

using Aliyun.OSS;
using Aliyun.OSS.Common;

// 异步上传。
namespace AsyncPutObject
{
    class Program
    {
        static string endpoint = "<yourEndpoint>";
        static string accessKeyId = "<yourAccessKeyId>";
        static string accessKeySecret = "<yourAccessKeySecret>";
        static string bucketName = "<yourBucketName>";
        static string objectName = "<yourObjectName>";
        static string localFilename = "<yourLocalFilename>";

        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/html";
                    string result = "Notice user: put object finish";

                    // 异步上传。
                    client.BeginPutObject(bucketName, objectName, fs,
metadata, PutObjectCallback, result.ToCharArray());
```

```
                    _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();
        }
    }
}
```

说明:

使用异步上传时，您需要实现自己的回调处理函数。

8.6.3 追加上传

本文介绍如何使用追加上传。

追加上传的完整代码请参见*GitHub*。

追加类型的文件（Append Object）暂时不支持copyObject操作。文件不存在时，调用AppendObject会创建一个可追加的文件；文件存在时，调用AppendObject会向文件末尾追加内容。

以下代码用于追加上传文件：

```
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}", 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);
}
```

8.6.4 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。

文件上传时，会在Checkpoint文件中记录当前上传的进度信息，如果上传过程中某一分片上传失败，再次上传时会从Checkpoint记录处继续上传，从而达到断点续传的效果。上传完成后，Checkpoint文件会被删除。

断点续传完整代码请参见*GitHub*。

以下代码用于断点续传上传：

```
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);
}
```

8.6.5 分片上传

本文介绍如何进行分片上传。

分片上传的完整代码请参见*GitHub*。

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，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CompleteMultipartUploadRequest方法将所有分片合并成完整的文件。

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
);
}
// 列举已上传的分片。
try
{
    var listPartsRequest = new ListPartsRequest(bucketName, objectName
, uploadId);
    var 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);
    }
}
catch (Exception ex)
{
    Console.WriteLine("List parts failed, {0}", ex.Message);
}
// 完成分片上传。
try
{
    var completeMultipartUploadRequest = new CompleteMultipartUpl
oadRequest(bucketName, objectName, uploadId);
    foreach (var partETag in partETags)
    {
        completeMultipartUploadRequest.PartETags.Add(partETag);
    }
    var result = client.CompleteMultipartUpload(completeMultipartUpl
oadRequest);
    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

取消分片上传事件的完整代码请参见gitHub。

以下代码用于取消分片上传：

```
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GitHub。

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	如何设置
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。	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)
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)

参数	作用	如何设置
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*。

以下代码用于列举所有分片上传事件：

```
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nextUpload
IdMarker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);
```

```
}
```

8.6.6 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 *GitHub*。

下面的代码以PutObject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);
        }
    }
}
```

8.6.7 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：

```
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}&mimeType=${mimeType}&" +
                                        "my_var1=${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, responseContent);
            }
            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);
            }
        }
        // 设置上传回调。
        private static ObjectMetadata BuildCallbackMetadata(string
callbackUrl, string callbackBody)
        {
            string callbackHeaderBuilder = new CallbackHeaderBuilder(
callbackUrl, callbackBody).Build();
            string CallbackVariableHeaderBuilder = new CallbackVa
riableHeaderBuilder().
                AddCallbackVariable("x:var1", "x:value1").AddCallbac
kVariable("x:var2", "x:value2").Build();
            var metadata = new ObjectMetadata();
            metadata.AddHeader(HttpHeaders.Callback, callbackHe
aderBuilder);
            metadata.AddHeader(HttpHeaders.CallbackVar, CallbackVa
riableHeaderBuilder);
            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。

8.7 下载文件

8.7.1 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。

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。

流式下载的完整代码请参见*GitHub*。

以下代码用于把指定的OSS文件下载到流：

```
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>";
// 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, objectName);
    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");
}
catch (Exception ex)
{
    Console.WriteLine("Get object failed. {0}", ex.Message);
```

```
}
```

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。

范围下载的完整代码请参见*GitHub*。

以下代码用于范围下载：

```
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。

参数	说明
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。
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。

8.7.4 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, downloadFilename)
    {
        // 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);
}
```

断点续传详情请参见开发指南中的断点续传。

8.7.5 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 += streamProg
ressCallback;
                // 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);
        }
    }
}
```

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**
- **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**

8.8.2 判断文件是否存在

本文介绍如何判断文件是否存在。

以下代码用于判断指定的文件是否存在：

```
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);
}
```

8.8.3 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*。

以下代码用于设置和获取文件访问权限：

```
using Aliyun.OSS;
using Aliyun.OSS.Common;

var endpoint = "<yourEndpoint>";
var accessKeyId = "<yourAccessKeyId>";
```

```
var accessKeySecret = "<yourAccessKeySecret>";
var bucketName = "<yourBucketName>";
var objectName = "<yourObjectName>";

// 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, CannedAccessControlL
ist.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。

8.8.4 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(saveAsFilename, "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。**

8.8.5 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*。

以下代码用于列举指定存储空间下的文件：

```
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);
    // 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*GitHub*。

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, ListObject
Callback, 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);
    }
}
```

8.8.6 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

以下代码用于删除单个文件：

```
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);
}
```

删除多个文件

每次最多删除1000个文件。有两种返回模式：

- **详细模式：返回删除成功的文件列表。默认为详细模式。**
- **简单模式：返回删除失败的文件列表。**

以下代码用于删除多个文件：

```
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 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，为false表示详细模式。默认为详细模式。
    var quietMode = false;
    // DeleteObjectsRequest的第三个参数指定返回模式。
    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);
}
```

8.8.7 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*GitHub*。

此操作的限制条件如下：

- **用户有源文件的读写权限。**
- **不支持跨地域拷贝。例如，不支持将杭州存储空间里的文件拷贝到青岛。**
- **文件的大小不能超过1GB。**

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1GB的文件，建议使用分片拷贝，步骤如下：

  1. **使用 InitiateMultipartUploadRequest 方法来初始化一个分片上传事件。**
  2. **使用 UploadPartCopy 方法进行分片拷贝。**
  3. **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 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, sourceObje
ct);
    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, sourceBucket, 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(uploadPart
CopyRequest);
        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.CompleteMultipartUpl
oad(completeMultipartUploadRequest);
    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);
}
```

8.8.8 解冻归档文件

本文介绍如何解冻归档文件。

解冻归档文件的完整代码请参见*GitHub*。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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：

```
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.HttpStatus
Code + " ,");
}
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.InvariantCultureIgnoreCase))
    {
        break;
    }
    Thread.Sleep(1000);
    maxWaitTimeInSeconds--;
}
if (maxWaitTimeInSeconds == 0)
{
    Console.WriteLine("RestoreObject is timeout. ");
    throw new TimeoutException();
}
else
{
    Console.WriteLine("RestoreObject is successful. ");
}
```

归档存储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*。

8.8.9 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*GetSymlink*。

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, targetObje
ctName);
    // 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*。

8.9 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的#unique_100。

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>";
// 拿到STS临时凭证后，通过其中的安全令牌（SecurityToken）和临时访问密钥（
AccessKeyId和AccessKeySecret）生成OSSClient。
// 创建OSSClient实例。
var ossStsClient = new OssClient(endpoint, accessKeyId, accessKeyS
ecret, securityToken);
// 进行OSS操作。
```

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，您可以指定URL的过期时间，来限制访客的访问时长。

使用签名URL进行临时授权的完整代码请参见GitHub。

- 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
{
    // 生成上传签名URL。
    var generatePresignedUriRequest = new GeneratePresignedUri
Request(bucketName, objectName, SignHttpMethod.Put)
    {
        Expiration = DateTime.Now.AddHours(1),
    };
    var signedUrl = client.GeneratePresignedUri(generatePresignedUri
Request);
    // 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**

  以下代码用于使用签名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;
    // 生成下载签名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);
}
```

8.10 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- 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。
- 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。
- 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。
- 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：

```
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，3天内有效。
    LifecycleRule lcr1 = new LifecycleRule()
    {
        ID = "delete obsoleted files",
        Prefix = "obsoleted/",
```

```
        Status = RuleStatus.Enabled,
        ExpriationDays = 3
    };
    //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
    };
    setBucketLifecycleRequest.AddLifecycleRule(lcr1);
    setBucketLifecycleRequest.AddLifecycleRule(lcr2);
    // 设置生命周期。
    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

查看生命周期规则完整代码请参见*GitHub*。

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);
        if (rule.ExpriationDays.HasValue)
            Console.WriteLine("ExpirationDays: {0}", rule.Expriation
Days);
    }
}
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);
}
```

8.11 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targetBucketName可以是同一存储空间。
    var request = new SetBucketLoggingRequest(bucketName, targetBuck
etName, "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);
}
```

8.12 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*GitHub*。

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*GitHub*。

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);
}
```

8.13 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);
}
```

8.14 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*GitHub*。

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);
}
```

8.15 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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- **获取图片信息**

- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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读写，必须通过授权才能进行访问。

- 匿名访问

  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

参数	描述
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

授权访问支持自定义样式、HTTPS和级联处理。

以下代码用于生成带签名的图片处理URL：

```
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
{
    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);
}
```

- **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：

```
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>";
            var localImageFilename = "<yourLocalImageFilename>";
            var downloadDir = "<yourDownloadDir>";
            // 创建OssClient实例。
            var client = new OssClient(endpoint, accessKeyId,
accessKeySecret);
            try
            {
                client.PutObject(bucketName, objectName,
localImageFilename);
                // 图片缩放
                var process = "image/resize,m_fixed,w_100,h_100";
                var ossObject = client.GetObject(new GetObjectR
equest(bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-resize.jpg",
ossObject.Content);
                // 图片裁剪
                process = "image/crop,w_100,h_100,x_100,y_100,r_1
";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-crop.jpg",
ossObject.Content);
                // 图片旋转
                process = "image/rotate,90";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-rotate.jpg",
ossObject.Content);
                // 图片锐化
                process = "image/sharpen,100";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-sharpen.jpg",
ossObject.Content);
                // 图片加水印
                process = "image/watermark,text_SGVsbG8g5Zu-
54mH5pyN5YqhIQ";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-watermark.jpg
", ossObject.Content);
                // 图片格式转换
                process = "image/format,png";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-format.png",
ossObject.Content);
                // 图片信息
                process = "image/info";
                ossObject = client.GetObject(new GetObjectRequest(
bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-info.txt",
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);
                }
            }
        }
    }
}
```

- **自定义样式**

以下代码用于自定义图片样式：

```
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>";
            var localImageFilename = "<yourLocalImageFilename>";
            var downloadDir = "<yourDownloadDir>";
            // 创建OssClient实例。
            var client = new OssClient(endpoint, accessKeyId,
accessKeySecret);
            try
            {
                client.PutObject(bucketName, objectName,
localImageFilename);
                // 自定义样式："style/<yourCustomStyleName>"。
                var process = "style/oss-pic-style-w-100";
                var ossObject = client.GetObject(new GetObjectR
equest(bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-style.jpg",
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);
                }
            }
        }
    }
}
```

- **级联处理**

以下代码用于级联处理图片：

```
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>";
            var localImageFilename = "<yourLocalImageFilename>";
            var downloadDir = "<yourDownloadDir>";
            // 创建OssClient实例。
            var client = new OssClient(endpoint, accessKeyId, 
accessKeySecret);
            try
            {
                client.PutObject(bucketName, objectName, 
localImageFilename);
```

```
                // 级联处理。
                var process = "image/resize,m_fixed,w_100,h_100/
rotate,90";
                var ossObject = client.GetObject(new GetObjectR
equest(bucketName, objectName, process));
                WriteToFile(downloadDir + "/sample-style.jpg",
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);
                }
            }
        }
    }
}
```

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。

8.16 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

错误码	描述
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法
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

9 Android

9.1 前言

本文档基于Android SDK 2.9.1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**

9.2 安装

本文介绍如何安装Android SDK。

环境要求

- **Android系统版本：2.3及以上**
- **有阿里云账户，并开通OSS服务。**

下载SDK

- **Android SDK开发包**

  Android Studio环境下：

```
dependencies { compile 'com.aliyun.dpa:oss-android-sdk:+' }
```

- **Github地址：点击查看**

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:+'
}
```

- **方式二：源码编译jar包**

  1. clone 工程源码，运行gradle命令打成jar包：

```
# clone工程
$ 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 1.7
$ ../gradlew releaseJar

# 进入打包生成目录，jar包生成在该目录下
$ cd build/libs && ls
```

  2. 直接引入上面编译好的jar包：

  将aliyun-oss-sdk-android-x.x.x.jar、*okhttp-3.x.x.jar* 、 *okio-1.x.x.jar* 3 个 jar 包导入 libs 目录。

相关设置

- **权限设置**

  OSS Android SDK 所需 Android 权限如下：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NET
WORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE
" />
<uses-permission android:name="android.permission.WRITE_EXTE
RNAL_STORAGE" />
```

```
<uses-permission android:name="android.permission.MOUNT_UNMO
UNT_FILESYSTEMS" />
```

说明:

请确保您的 AndroidManifest.xml 文件中配置了以上权限，否则 SDK 将无法正常工作。

- **混淆设置**

  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.**
```

9.3 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 = "<yourAccess
KeySecret>";

    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:

10:53
请先设置正确的STS鉴权服务器地址，否则以下服务无法直接使用。
STS鉴权服务器:
http://0.0.0.0:0000/sts/getsts
上传Bucket:
测试bucket名称
区域:
杭州
设置
选择图片
请输入OSS文件名
普通上传下载
下载
上传
缩略宽度
缩略高度
图片缩略
请输入水印文字
文字大小
文字水印
OSSSIGN
MANAGE
LISTOBJECTS
HEADOBJECT
MULTIPART
RESUMABLE
CUTOMSIGN
BATCH
CALLBACK

创建存储空间

存储空间是OSS全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于新建一个存储空间：

```
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<CreateBucketRequest, CreateBucketResult>() {
    @Override
    public void onSuccess(CreateBucketRequest request, CreateBuck
etResult result) {
        Log.d("locationConstraint", request.getLocationConstraint());
        }
    @Override
    public void onFailure(CreateBucketRequest request, ClientException
 clientException, ServiceException serviceException) {
        // 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，请参见基本概念中的命名规范。创建存储空间详情，请参见#unique_41。

获取endpoint信息，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, 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) {
            // 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, GetObjectResult>() {
    @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 serviceException) {
        // 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。

9.4 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 OSSAuthCredentialsPr
ovider(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, credential
Provider);
```

> **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, credential
Provider);
```

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 OSSStsTokenCredentia
lProvider("<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, credential
Provider, 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");
```

9.5 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。

创建存储空间

以下代码用于创建存储空间：

```
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<CreateBucketRequest, CreateBucketResult>() {
    @Override
    public void onSuccess(CreateBucketRequest request, CreateBuck
etResult result) {
        Log.d("asyncCreateBucket", "Success");
    }
    @Override
    public void onFailure(CreateBucketRequest request, ClientException
 clientException, ServiceException serviceException) {
        // 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请参见*#unique_40/unique_40_Connect_42_section_yxy_jmt_tdb*。

您可以在创建存储空间时指定**存储空间的权限**和**存储类型**。

上述代码在创建Bucket时，指定了Bucket的ACL和所在地域。

- 每个用户的Bucket数量不能超过30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。

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：

```
// 构造上传请求。
GetBucketACLRequest getBucketACLRequest = new GetBucketACLRequest("<
bucketName>");

// 获取存储空间访问权限。
OSSAsyncTask getBucketAclTask = oss.asyncGetBucketACL(getBucketA
CLRequest, new OSSCompletedCallback<GetBucketACLRequest, GetBucketA
CLResult>() {
    @Override
    public void onSuccess(GetBucketACLRequest request, GetBucketA
CLResult 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 serviceException) {
        // 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。

获取存储空间信息

以下代码用于获取存储空间的信息（Info）：

```
// 构造查看请求。
```

```
GetBucketInfoRequest request = new GetBucketInfoRequest("<bucketName
>");

// 获取存储空间信息。
OSSAsyncTask task = oss.asyncGetBucketInfo(request, new OSSComplet
edCallback<GetBucketInfoRequest, GetBucketInfoResult>() {
    @Override
    public void onSuccess(GetBucketInfoRequest request, GetBucketI
nfoResult result) {
        OSSLog.logInfo("code: " + result.getStatusCode());
    }

    @Override
    public void onFailure(GetBucketInfoRequest request, ClientExce
ption clientException, ServiceException serviceException) {
        // 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());
        }
    }
});
```

删除存储空间

以下代码用于删除单个存储空间：

```
// 构造上传请求。
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest("<
bucketName>");

// 异步删除存储空间。
OSSAsyncTask deleteBucketTask = oss.asyncDeleteBucket(deleteBuck
etRequest, new OSSCompletedCallback<DeleteBucketRequest, DeleteBuck
etResult>() {
    @Override
    public void onSuccess(DeleteBucketRequest request, DeleteBuck
etResult result) {
        Log.d("asyncDeleteBucket", "Success!");
    }
    @Override
    public void onFailure(DeleteBucketRequest request, ClientException
 clientException, ServiceException serviceException) {
        // 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Bucket。
- 只有Bucket的拥有者才能删除这个Bucket。

9.6 上传文件

9.6.1 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。

9.6.2 简单上传

本文介绍如何使用简单上传。

上传本地文件

- 调用同步接口上传：

```
// 构造上传请求。
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<
objectName>", "<uploadFilePath>");

// 文件元信息的设置是可选的。
// ObjectMetadata metadata = new ObjectMetadata();
// metadata.setContentType("application/octet-stream"); // 设置
content-type。
// metadata.setContentMD5(BinaryUtil.calculateBase64Md5(uploadFile
Path)); // 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) {
    // 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());
}
```

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：**

```
// 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 currentSiz
e, 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) {
            // 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。
```

上传二进制byte[]数组

调用异步接口上传二进制byte[]数组代码如下：

```
byte[] uploadData = new byte[100 * 1024];
new Random().nextBytes(uploadData);

// 构造上传请求。
PutObjectRequest put = new PutObjectRequest("<bucketName>", "<
objectName>", uploadData);

try {
    PutObjectResult putResult = oss.putObject(put);

    Log.d("PutObject", "UploadSuccess");
    Log.d("ETag", putResult.getETag());
    Log.d("RequestId", putResult.getRequestId());
} 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());
}
```

9.6.3 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。

以下代码用于追加上传：

```
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 OSSComplet
edCallback<AppendObjectRequest, AppendObjectResult>() {
    @Override
    public void onSuccess(AppendObjectRequest request, AppendObje
ctResult 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。

9.6.4 分片上传

本文介绍如何使用分片上传。

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，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oss.CompleteMultipartUpload方法将所有分片合并成完整的文件。

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。
    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MultipartUpload事件。
    String uploadId = initResult.getUploadId();

    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part number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>", "<objectName>", "<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。

> **注意:**
>
> - 每一个分片上传请求需指定 UploadId 和 PartNum 。
> - uploadPart 要求除最后一个 Part 外，其他的 Part 大小都要大于100KB。但是 Upload Part 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 号码的范围是1~10000。如果超出这个范围，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。

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，则只会返回 1000 个 Multipart Upload 信息，且返回结果中 IsTruncated 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：

```
    //取消分片上传。
```

```
    AbortMultipartUploadRequest abort = new AbortMultipartUpload
Request("<bucketName>", "<objectName>", "<uploadId>");
    AbortMultipartUploadResult abortResult = oss.abortMultipartUpload(
abort);
```

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。

- **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256KB，最小值为100KB。
        rq.setPartSize(1024 * 1024);
        rq.setProgressCallback(new OSSProgressCallback<MultipartU
ploadRequest>() {
            @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<MultipartU
ploadRequest>() {

        @Override
        public void onProgress(MultipartUploadRequest request, long
currentSize, long totalSize) {
            OSSLog.logDebug("[testMultipartUpload] - " + currentSize
 + " " + totalSize, false);
        }
    });

    OSSAsyncTask task = oss.asyncMultipartUpload(request, new
OSSCompletedCallback<MultipartUploadRequest, CompleteMultipartUpl
oadResult>() {
    @Override
    public void onSuccess(MultipartUploadRequest request, CompleteMu
ltipartUploadResult result) {
        OSSLog.logInfo(result.getServerCallbackReturnBody());
    }

    @Override
    public void onFailure(MultipartUploadRequest request, ClientExce
ption clientException, ServiceException serviceException) {
```

```
            OSSLog.logError(serviceException.getRawMessage());

        }
    });

    // Thread.sleep(100);
    //取消分片上传。
    // task.cancel();

     task.waitUntilFinished();
```

9.6.5 断点续传上传

本文介绍如何使用断点续传上传。

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，SDK提供了断点续传上传功能。

注意:

- 断点续传上传暂时只支持上传本地文件。
- 对于移动端来说，如果不是比较大的文件，不建议使用这种方式上传，因为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。

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。

- 断点续传上传只在本次上传生效。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，您可以通过为Bucket设置lifeCycle规则的方式来定时清理碎片。详情请参考：生命周期管理。

说明:

- 断点续传上传需要依赖`InitMultipartUpload`、`UploadPart`、`ListParts`、`CompleteMultipartUpload`、`AbortMult`
  些接口来实现。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。
- 断点续传上传也支持上传后回调通知，使用用法与常见的上传回调通知一致。
- 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。

- 断点记录不在本地持久保存时的调用方式如下：

```
// 创建断点上传请求
ResumableUploadRequest request = new ResumableUploadRequest("<
bucketName>", "<objectKey>", "<uploadFilePath>");
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableU
ploadRequest>() {
    @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, ResumableU
ploadResult result) {
        Log.d("resumableUpload", "success!");
    }

    @Override
    public void onFailure(ResumableUploadRequest request, ClientExce
ption clientExcepion, ServiceException serviceException) {
        // 异常处理
    }
});

// resumableTask.waitUntilFinished(); // 可以等待直到任务完成
```

- 断点记录在本地持久保存时的调用方式如下：

```
String recordDirectory = Environment.getExternalStorageDirectory().
getAbsolutePath() + "/oss_record/";

File recordDir = new File(recordDirectory);

// 要保证目录存在，如果不存在则主动创建
if (!recordDir.exists()) {
    recordDir.mkdirs();
}

// 创建断点上传请求，参数中给出断点记录文件的保存位置，需是一个文件夹的绝对路径
ResumableUploadRequest request = new ResumableUploadRequest("<
bucketName>", "<objectKey>", "<uploadFilePath>", recordDirectory);
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableU
ploadRequest>() {
    @Override
    public void onProgress(ResumableUploadRequest request, long
currentSize, long totalSize) {
        Log.d("resumableUpload", "currentSize: " + currentSize + "
totalSize: " + totalSize);
    }
});
```

```
OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new
OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult
>() {
    @Override
    public void onSuccess(ResumableUploadRequest request, ResumableU
ploadResult result) {
        Log.d("resumableUpload", "success!");
    }

    @Override
    public void onFailure(ResumableUploadRequest request, ClientExce
ption clientExcepion, ServiceException serviceException) {
        // 异常处理
    }
});

// resumableTask.waitUntilFinished();
```

- 以下代码用于实现断点续传上传：

```
//调用OSSAsyncTask cancel()方法时是否需要删除断点记录文件的设置
String recordDirectory = Environment.getExternalStorageDirectory().
getAbsolutePath() + "/oss_record/";

File recordDir = new File(recordDirectory);

// 要保证目录存在，如果不存在则主动创建
if (!recordDir.exists()) {
    recordDir.mkdirs();
}

// 创建断点上传请求，参数中给出断点记录文件的保存位置，需是一个文件夹的绝对路径
ResumableUploadRequest request = new ResumableUploadRequest("<
bucketName>", "<objectKey>", "<uploadFilePath>", recordDirectory);
//设置false，取消时，不删除断点记录文件，如果不进行设置，默认true，是会删除断点
记录文件，下次再进行上传时会重新上传。
request.setDeleteUploadOnCancelling(false);
// 设置上传过程回调
request.setProgressCallback(new OSSProgressCallback<ResumableU
ploadRequest>() {
    @Override
    public void onProgress(ResumableUploadRequest request, long
currentSize, long totalSize) {
        Log.d("resumableUpload", "currentSize: " + currentSize + "
totalSize: " + totalSize);
    }
});


OSSAsyncTask resumableTask = oss.asyncResumableUpload(request, new
OSSCompletedCallback<ResumableUploadRequest, ResumableUploadResult
>() {
    @Override
    public void onSuccess(ResumableUploadRequest request, ResumableU
ploadResult result) {
        Log.d("resumableUpload", "success!");
    }

    @Override
    public void onFailure(ResumableUploadRequest request, ClientExce
ption clientExcepion, ServiceException serviceException) {
        // 异常处理
```

```
    }
});

// resumableTask.waitUntilFinished();
```

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, 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) {
            // 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。
```

9.6.7 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。

以下代码用于上传回调：

```
PutObjectRequest put = new PutObjectRequest(testBucket, testObject,
uploadFilePath);

put.setCallbackParam(new HashMap<String, String>() {
 {
  put("callbackUrl", "110.75.82.106/callback");
        put("callbackHost", "oss-cn-hangzhou.aliyuncs.com");
        put("callbackBodyType", "application/json");
  put("callbackBody", "{\"mimeType\":${mimeType},\"size\":${size}}");
 }
});

OSSAsyncTask task = oss.asyncPutObject(put, new OSSCompletedCallback<
PutObjectRequest, PutObjectResult>() {
 @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 serviceException) {
  // 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", "oss-cn-hangzhou.aliyuncs.com");
        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。

9.7 下载文件

9.7.1 概述

OSS Android SDK提供了丰富的文件下载方式：

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

9.7.2 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
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, GetObjectResult>() {
 @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 serviceException) {
  // 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());
  }
 }
});
```

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。

以下代码用于范围下载：

```
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, GetObjectResult>() {
 @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 serviceException) {
  // 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());
  }
 }
```

```
});
```

9.7.4 下载到本地文件

本文档介绍如何将OSS文件下载到本地。

以下代码用于把指定的OSS文件下载到本地文件：

```
public void testDownloadObjectToFile() throws Exception {
        // upload file
        PutObjectRequest put = new PutObjectRequest(mBucketName, "
file1m",
                OSSTestConfig.FILE_DIR + "file1m");
        OSSTestConfig.TestPutCallback putCallback = new OSSTestConfig.
TestPutCallback();
        String srcFileBase64Md5 = BinaryUtil.toBase64String(BinaryUtil
.calculateMd5(OSSTestConfig.FILE_DIR + "file1m"));
        OSSAsyncTask putTask = oss.asyncPutObject(put, putCallback);
        putTask.waitUntilFinished();
        assertEquals(200, putCallback.result.getStatusCode());

        // download object to file
        GetObjectRequest get = new GetObjectRequest(mBucketName, "
file1m");
        OSSTestConfig.TestGetCallback getCallback = new OSSTestConfig.
TestGetCallback();
        OSSAsyncTask getTask = oss.asyncGetObject(get, getCallback);
        getTask.waitUntilFinished();
        assertEquals(200, getCallback.result.getStatusCode());
        long length = getCallback.result.getContentLength();
        byte[] buffer = new byte[(int) length];
        int readCount = 0;
        while (readCount < length) {
            readCount += getCallback.result.getObjectContent().read(
buffer, readCount, (int) length - readCount);
        }
        try {
            FileOutputStream fout = new FileOutputStream(OSSTestConfig
.FILE_DIR + "download_file1m");
            fout.write(buffer);
            fout.close();
        } catch (Exception e) {
            OSSLog.logInfo(e.toString());
        }
        String downloadFileBase64Md5 = BinaryUtil.toBase64String(
BinaryUtil.calculateMd5(OSSTestConfig.FILE_DIR + "download_file1m"));
        assertEquals(srcFileBase64Md5, downloadFileBase64Md5);

    }
```

9.7.5 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，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：

```
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;

    }
```

9.7.6 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。

细节分析

If-Range HTTP 请求头字段使得 Range 头字段在一定条件下起作用：

- 如果字段值中的条件得到满足时，Range 头字段才会起作用。同时服务器返回 206 部分内容状态码，以及 Range 头字段请求的相应部分。
- 如果字段值中的条件没有得到满足，服务器将会返回 200 OK 状态码，并返回完整的请求资源。
- 字段值中既可以用 Last-Modified 时间值用作验证，也可以用 ETag 标记作为验证，但不能同时使用 Last-Modified 时间值及 ETag 标记用于验证。

If-Range 头字段通常用于断点续传下载过程中，确保上次下载中断后下载的资源没有发生改变。

使用`If-Range`时，客户端只需发起一次网络请求，而`If-Unmodified-Since`或者`If-Match`在条件判断失败时，会返回 412 前置条件检查失败状态码，客户端需再次开启请求来获取资源。

具体实现

有关 Android 中如何实现对 OSS Object 的断点续传下载，请参考以下 iOS 中的代码实现。如需下载，可自行实现或使用第三方开源的下载框架。

iOS 中断点续传下载 OSS Object 示例代码如下：

```
//1. 首先使用sdk获取object的下载链接
String signedURLString = ossClient.presignConstrainedObjectURL(bucket
, object, expires);

//2. 添加下载任务

        mDownloadManager = DownloadManager.getInstance();
        mDownloadManager.add(signedURLString, new DownloadListner() {
            @Override
            public void onFinished() {
                Toast.makeText(MainActivity.this, "下载完成!", Toast.
LENGTH_SHORT).show();
            }

            @Override
            public void onProgress(float progress) {
                pb_progress1.setProgress((int) (progress * 100));
                tv_progress1.setText(String.format("%.2f", progress *
100) + "%");
            }

            @Override
            public void onPause() {
                Toast.makeText(MainActivity.this, "暂停了!", Toast.
LENGTH_SHORT).show();
            }

            @Override
            public void onCancel() {
                tv_progress1.setText("0%");
                pb_progress1.setProgress(0);
                btn_download1.setText("下载");
                Toast.makeText(MainActivity.this, "下载已取消!", Toast.
LENGTH_SHORT).show();
            }
        });

//3.开启下载
```

```
        mDownloadManager.download(signedURLString);

//4.暂停下载
        mDownloadManager.cancel(signedURLString);

//5.继续下载
        mDownloadManager.download(signedURLString);
```

说明:

目前 OSS Server 不支持`If-Range`字段。所以还不能使用`NSURLSessionDownloadTask`在 Android 中实现断点续传下载。而在 iOS 中使用`NSURLSessionDownloadTask`实现断点续传下载时，会发送`If-Range`给服务器以确认下载文件是否发生更改。

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**
- **获取文件元信息**
- **列举文件**
- **拷贝文件**
- **删除文件**

9.8.2 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。

以下代码用于判断指定文件是否存在：

```
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());
```

```
}
```

9.8.3 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<CopyObjectRequest, CopyObjectResult>() {
    @Override
    public void onSuccess(CopyObjectRequest request, CopyObjectResult
result) {
            Log.d("copyObject", "copy success!");
    }

    @Override
    public void onFailure(CopyObjectRequest request, ClientException
clientExcepion, ServiceException serviceException) {
        // 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。**

9.8.4 列举文件

本文介绍如何列举存储空间中的所有文件及指定前缀的文件。

列举存储空间中所有文件

以下代码用于列举存储空间中所有文件：

```
ListObjectsRequest listObjects = new ListObjectsRequest("<bucketName
>");

// 设置成功、失败回调，发送异步列举请求。
OSSAsyncTask task = oss.asyncListObjects(listObjects, new OSSComplet
edCallback<ListObjectsRequest, ListObjectsResult>() {
 @Override
 public void onSuccess(ListObjectsRequest request, ListObjectsResult
result) {
  Log.d("AyncListObjects", "Success!");
  for (int i = 0; i < result.getObjectSummaries().size(); i++) {
   Log.d("AyncListObjects", "object: " + result.getObjectSummaries().
get(i).getKey() + " "
     + result.getObjectSummaries().get(i).getETag() + " "
     + result.getObjectSummaries().get(i).getLastModified());
  }
 }

 @Override
 public void onFailure(ListObjectsRequest request, ClientException
clientExcepion, ServiceException serviceException) {
  // 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());
  }
 }
});
// 等待任务完成。
task.waitUntilFinished();
```

列举存储空间中指定前缀的文件

以下代码用于列出存储空间中以“file”为前缀的所有文件。

```
 ListObjectsRequest listObjects = new ListObjectsRequest("<bucketName
>");

// 设定前缀
listObjects.setPrefix("file");

// 设置成功、失败回调，发送异步列举请求
OSSAsyncTask task = oss.asyncListObjects(listObjects, new OSSComplet
edCallback<ListObjectsRequest, ListObjectsResult>() {
 @Override
```

```
 public void onSuccess(ListObjectsRequest request, ListObjectsResult
result) {
  Log.d("AyncListObjects", "Success!");
  for (int i = 0; i < result.getObjectSummaries().size(); i++) {
   Log.d("AyncListObjects", "object: " + result.getObjectSummaries().
get(i).getKey() + " "
     + result.getObjectSummaries().get(i).getETag() + " "
     + result.getObjectSummaries().get(i).getLastModified());
  }
 }

 @Override
 public void onFailure(ListObjectsRequest request, ClientException
clientExcepion, ServiceException serviceException) {
  // 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();
```

各参数说明如下：

名称	描述
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: CommonPrefixes。
marker	设定结果从marker之后按字母排序的第一个开始返回。
maxkeys	限定此次返回object的最大数，如果不设定，默认为100，maxkeys取值不能大于1000。
prefix	限定返回的object key必须以prefix作为前缀。 **说明:** 使用prefix查询时，返回的key中仍会包含prefix。

9.8.5 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，文件一旦删除将无法恢复。

以下代码用于删除文件。

```
// 创建删除请求。
DeleteObjectRequest delete = new DeleteObjectRequest("<bucketName>",
 "<objectKey>");
// 异步删除。
OSSAsyncTask deleteTask = oss.asyncDeleteObject(delete, new OSSComplet
edCallback<DeleteObjectRequest, DeleteObjectResult>() {
    @Override
    public void onSuccess(DeleteObjectRequest request, DeleteObje
ctResult result) {
        Log.d("asyncCopyAndDelObject", "success!");
    }

    @Override
    public void onFailure(DeleteObjectRequest request, ClientException
 clientExcepion, ServiceException serviceException) {
        // 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());
        }
    }
});
```

> **注意:**
>
> DeleteObject要求对Object所在的Bucket有写权限。

9.8.6 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, PutObjectResult>() {
    ...
});
```

9.8.7 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, HeadObjectResult>() {
    @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 serviceException) {
        // 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。
```

9.8.8 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。
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 OSSComplet
edCallback<RestoreObjectRequest,
        RestoreObjectResult>() {
    @Override
    public void onSuccess(RestoreObjectRequest request, RestoreObj
ectResult result) {
        OSSLog.logInfo("code::"+result.getStatusCode());

    }

    @Override
    public void onFailure(RestoreObjectRequest request, ClientExce
ption clientException,
                          ServiceException serviceException) {
        OSSLog.logError("error: "+serviceException.getRawMessage());

    }
});

task.waitUntilFinished();
```

9.8.9 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。

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 OSSComplet
edCallback<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 OSSComplet
edCallback<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*。

9.9 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, currentIndex);
// 设置分片内容
uploadPart.setPartContent(partData);
//设置MD5内容
uploadPart.setMd5Digest(BinaryUtil.calculateBase64Md5(data));
```

9.10 生命周期

本文介绍如何管理生命周期。

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。每条规则包含：

- 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。
- 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。
- 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。
- 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。

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<PutBucketLifecycleRequest, PutBucketLifecycleRe
sult>() {
    @Override
    public void onSuccess(PutBucketLifecycleRequest request,
PutBucketLifecycleResult result) {
        OSSLog.logInfo("code::"+result.getStatusCode());

    }

    @Override
    public void onFailure(PutBucketLifecycleRequest request,
ClientException clientException, ServiceException serviceException) {
        OSSLog.logError("error: "+serviceException.getRawMessage());

    }
});

task.waitUntilFinished();
```

有关设置生命周期的更多信息，请参见#unique_438。

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<GetBucketLifecycleRequest, GetBucketLifecycleRe
sult>() {
    @Override
    public void onSuccess(GetBucketLifecycleRequest request,
GetBucketLifecycleResult result) {
         ArrayList<BucketLifecycleRule> list = result.getlifecyc
leRules();
```

```
            for (BucketLifecycleRule rule : list){
                OSSLog.logInfo("info: " + rule.getIdentifier());
            }
        }

        @Override
        public void onFailure(GetBucketLifecycleRequest request,
ClientException clientException, ServiceException serviceException) {
            OSSLog.logError("error: "+serviceException.getRawMessage());

        }
});
task.waitUntilFinished();
```

有关查看生命周期的更多信息，请参见#unique_439。

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 DeleteBucketLifecycl
eRequest();
request.setBucketName("yourBucketName");
OSSAsyncTask task = oss.asyncDeleteBucketLifecycle(request,"yourBucket
Name" new OSSCompletedCallback<DeleteBucketLifecycleRequest,
DeleteBucketLifecycleResult>() {
    @Override
    public void onSuccess(DeleteBucketLifecycleRequest request,
DeleteBucketLifecycleResult result) {
        OSSLog.logInfo("code :  "+result.getStatusCode());

    }

    @Override
    public void onFailure(DeleteBucketLifecycleRequest request,
ClientException clientException, ServiceException 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#unique_440。

9.11 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 OSSComplet
edCallback<PutBucketRefererRequest, PutBucketRefererResult>() {
    @Override
    public void onSuccess(PutBucketRefererRequest request, PutBucketR
efererResult result) {
        OSSLog.logInfo("code: " + result.getStatusCode());
    }
    @Override
    public void onFailure(PutBucketRefererRequest request, ClientExce
ption clientException, ServiceException 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 OSSComplet
edCallback<GetBucketRefererRequest, GetBucketRefererResult>() {
    @Override
    public void onSuccess(GetBucketRefererRequest request, GetBucketR
efererResult 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, ClientExce
ption clientException, ServiceException 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 OSSComplet
edCallback<PutBucketRefererRequest, PutBucketRefererResult>() {
    @Override
    public void onSuccess(PutBucketRefererRequest request, PutBucketR
efererResult result) {
        OSSLog.logInfo("code: " + result.getStatusCode());

    }
    @Override
    public void onFailure(PutBucketRefererRequest request, ClientExce
ption clientException, ServiceException serviceException) {
        OSSLog.logError("error: "+serviceException.getRawMessage());

    }
});
```

```
task.waitUntilFinished();
```

9.12 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>");
  ```

9.13 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。

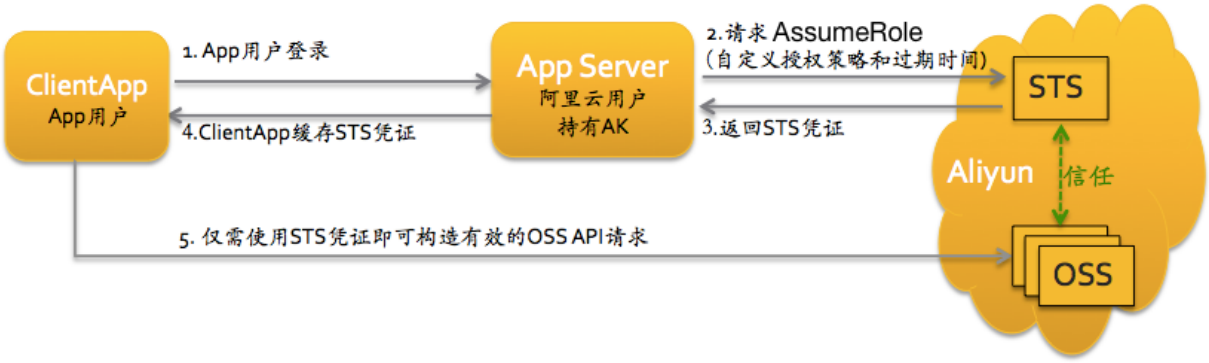
STS鉴权模式

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。STS更详细的解释请参见*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APP应用为例，交互流程如图所示：



方案的详细描述如下：

1. App用户登录。App用户身份由您自己管理。您可以自定义身份管理系统，也可以使用外部Web账号或OpenID。对于每个有效的App用户来说，AppServer可以确切地定义出每个App用户的最小访问权限。
2. AppServer请求STS服务获取一个安全令牌（SecurityToken）。在调用STS之前，AppServer需要确定App用户的最小访问权限（用Policy语法描述）以及授权的过期时间。 然后通过调用STS的AssumeRole（扮演角色）接口来获取安全令牌。角色管理与使用相关内容请参考RAM使用指南中的角色管理。
3. STS返回给AppServer一个有效的访问凭证，App端称为FederationToken，包括一个安全令牌（SecurityToken）、临时访问密钥（AccessKeyId, AccessKeySecret）以及过期时间。
4. AppServer将FederationToken返回给ClientApp。ClientApp可以缓存这个凭证。当凭证失效时，ClientApp需要向AppServer申请新的有效访问凭证。例如访问凭证有效期为1小时，那么ClientApp可以每30分钟向AppServer请求更新访问凭证。
5. ClientApp使用本地缓存的FederationToken去请求Aliyun Service API。云服务会感知STS访问凭证，并会依赖STS服务来验证访问凭证，并正确响应用户请求。

使用STS访问OSS的流程请参见开发指南中的*#unique_100*。

STS安全令牌详情请参见RAM使用指南中的角色管理。您可以通过调用STS服务接口*AssumeRole*来获取有效访问凭证，也可以直接使用STS SDK来调用该方法，点击查看。

> **说明:**
> 使用STS模式授权需要先开通阿里云RAM服务。

STS使用手册：点击查看

OSS授权策略配置：点击查看

- **直接设置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）获取一对StsToken，然后用它来初始化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新的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 OSSStsTokenCredentia
lProvider("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<
StsToken.SecurityToken>");

OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider);
```

  在您判断到Token即将过期时，您可以重新构造新的OSSClient，也可以通过如下方式更新CredentialProvider：

```
oss.updateCredentialProvider(new OSSStsTokenCredentialProvider
("<StsToken.AccessKeyId>", "<StsToken.SecretKeyId>", "<StsToken.
SecurityToken>"));
```

- **实现获取StsToken回调**

  如果您期望SDK自动更新Token，则要求您在SDK的应用中实现回调。这个回调通过您实现的方式去获取一个Federation Token（即StsToken），然后返回。SDK会利用这个Token来进行加签处理，并在需要更新时主动调用这个回调来获取Token：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

OSSCredentialProvider credentialProvider = new OSSFederationCredent
ialProvider() {

    @Override
    public OSSFederationToken getFederationToken() {
    // 您需要在这里实现获取一个FederationToken，并构造成OSSFederationToken
对象返回
        // 如果因为某种原因获取失败，可直接返回nil

        OSSFederationToken * token;
        // 下面是一些获取token的代码，比如从您的server获取
        ...
        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token所需的各个字段，也可以在这个回调中直接返回。但您需要手动处理token的更新，且更新后重新设置该OSSClient实例的OSSCredentialProvider。

假设您访问搭建的server地址为：http://localhost:8080/distribute-token.json，则返回的数据如下：

```
{
    "StatusCode": 200,
    "AccessKeyId":"STS.iA645eTOXEqP3cg3VeHf",
    "AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojUBZCf",
    "Expiration":"2015-11-03T09:52:59Z",
    "SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS/
p3OgaUC+oJl3XSlbJ7StKpQ...."}
```

`OSSFederationCredentialProvider`**的实现示例如下：**

```
OSSCredentialProvider credetialProvider = new OSSFederationCredent
ialProvider() {
    @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AccessKeyId/AccessKeySecret保存在您的业务server，然后在SDK实现回调，将需要加签的签名串POST到server，您在业务server按照OSS规定的签名算法对这个签名串完成签名后，返回给该回调函数，再由回调返回给server。

签名算法请参考在*Header*中包含签名。

content是已经根据请求各个参数拼接后的字符串，所以算法为：

```
signature = "OSS " + AccessKeyId + ":" + base64(hmac-sha1(AccessKeyS
ecret, content))
```

代码如下：

```
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";

credentialProvider = new OSSCustomSignerCredentialProvider() {
    @Override
    public String signContent(String content) {
    // 您需要在这里依照OSS规定的签名算法，实现加签一串字符内容，并把得到的签名传拼
接上AccessKeyId后返回
        // 一般实现是，将字符内容post到您的业务服务器，然后返回签名
        // 如果因为某种原因加签失败，描述error信息后，返回nil

        // 以下是用本地算法进行的演示
        return "OSS " + AccessKeyId + ":" + base64(hmac-sha1(
AccessKeySecret, content));
    }
};

OSS oss = new OSSClient(getApplicationContext(), endpoint, credential
Provider);
```

注意：

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。如果您在其中实现了向业务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具体请求的时候，在请求的子线程中执行，所以不会阻塞主线程。

9.14 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*#unique_105*。

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 OSSComplet
edCallback<PutBucketLoggingRequest, PutBucketLoggingResult>() {
    @Override
    public void onSuccess(PutBucketLoggingRequest request, PutBucketL
oggingResult result) {
        OSSLog.logInfo("code::"+result.getStatusCode());
    }

    @Override
    public void onFailure(PutBucketLoggingRequest request, ClientExce
ption clientException, ServiceException 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#unique_448。

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 OSSComplet
edCallback<GetBucketLoggingRequest, GetBucketLoggingResult>() {
    @Override
    public void onSuccess(GetBucketLoggingRequest request, GetBucketL
oggingResult result) {
        OSSLog.logInfo("info: " + result.getTargetBucketName()+"*"+
result.getTargetPrefix());

    }

    @Override
    public void onFailure(GetBucketLoggingRequest request, ClientExce
ption clientException, ServiceException 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#unique_449。

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<DeleteBucketLoggingRequest, DeleteBucketLoggingR
esult>() {
    @Override
    public void onSuccess(DeleteBucketLoggingRequest request,
DeleteBucketLoggingResult result) {
        OSSLog.logInfo("code::"+result.getStatusCode());

    }

    @Override
    public void onFailure(DeleteBucketLoggingRequest request,
ClientException clientException, ServiceException 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*#unique_450*。

9.15 图片处理

OSS图片处理是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。

用户将原始图片上传保存到OSS，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。基于OSS图片处理，用户可以搭建自己的图片处理服务。**

图片处理功能

图片处理的详细信息请参见*OSS***图片处理指南**。

OSS**图片处理****提供以下功能：**

- **图片缩放**
- **图片裁剪**
- **图片旋转**
- **图片效果**
- **格式转换**
- **图片水印，包括添加图片、文字、图文混合水印**

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, GetObjectResult>() {
    @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 serviceException) {
        // 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：

```
OSSImagePersistRequest *request = [OSSImagePersistRequest new];
```

```
request.fromBucket = _privateBucketName;
request.fromObject = OSS_IMAGE_KEY;
request.toBucket = _privateBucketName;
request.toObject = @"image_persist_key";
// 缩放
request.action = @"image/resize,w_100";

[[[ossClient imageActionPersist:request] continueWithBlock:^id
_Nullable(OSSTask * _Nonnull task) {

    return nil;
}] waitUntilFinished];
```

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 异常响应

OSS Android SDK 中有两种异常 ClientException 以及 ServiceException，它们都是受检异常。

ClientException

ClientException指SDK内部出现的异常，比如参数错误，网络无法到达，主动取消等等。

ServiceException

OSSException指服务器端错误，它来自于对服务器错误信息的解析。OSSException一般有以下几个成员：

- Code： OSS返回给用户的错误码。
- Message： OSS给出的详细错误信息。
- RequestId： 用于唯一标识该次请求的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。
- HostId： 用于标识访问的OSS集群。
- rawMessage： HTTP响应的原始Body文本。

下面是OSS中常见的异常：

错误码	描述
AccessDenied	拒绝访问
BucketAlreadyExists	Bucket已经存在
BucketNotEmpty	Bucket不为空
EntityTooLarge	实体过大
EntityTooSmall	实体过小

错误码	描述
FileGroupTooLarge	文件组过大
FilePartNotExist	文件Part不存在
FilePartStale	文件Part过时
InvalidArgument	参数格式错误
InvalidAccessKeyId	AccessKeyId不存在
InvalidBucketName	无效的Bucket名字
InvalidDigest	无效的摘要
InvalidObjectName	无效的Object名字
InvalidPart	无效的Part
InvalidPartOrder	无效的part顺序
InvalidTargetBucketForLogging	Logging操作中有无效的目标bucket
InternalError	OSS内部发生错误
MalformedXML	XML格式非法
MethodNotAllowed	不支持的方法
MissingArgument	缺少参数
MissingContentLength	缺少内容长度
NoSuchBucket	Bucket不存在
NoSuchKey	文件不存在
NoSuchUpload	Multipart Upload ID不存在
NotImplemented	无法处理的方法
PreconditionFailed	预处理错误
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟
RequestTimeout	请求超时
SignatureDoesNotMatch	签名错误
TooManyBuckets	用户的Bucket数目超过限制

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。

运行环境要求

- 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，在弹出框勾选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'
```

有关管理工具 Cocoapods 的更多详情，请参考文档*CocoaPods*安装和使用教程。

> **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。App 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`来绕过 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`，就能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 iOS SDK的快速入门。

更多细节用法可以参考本工程的：

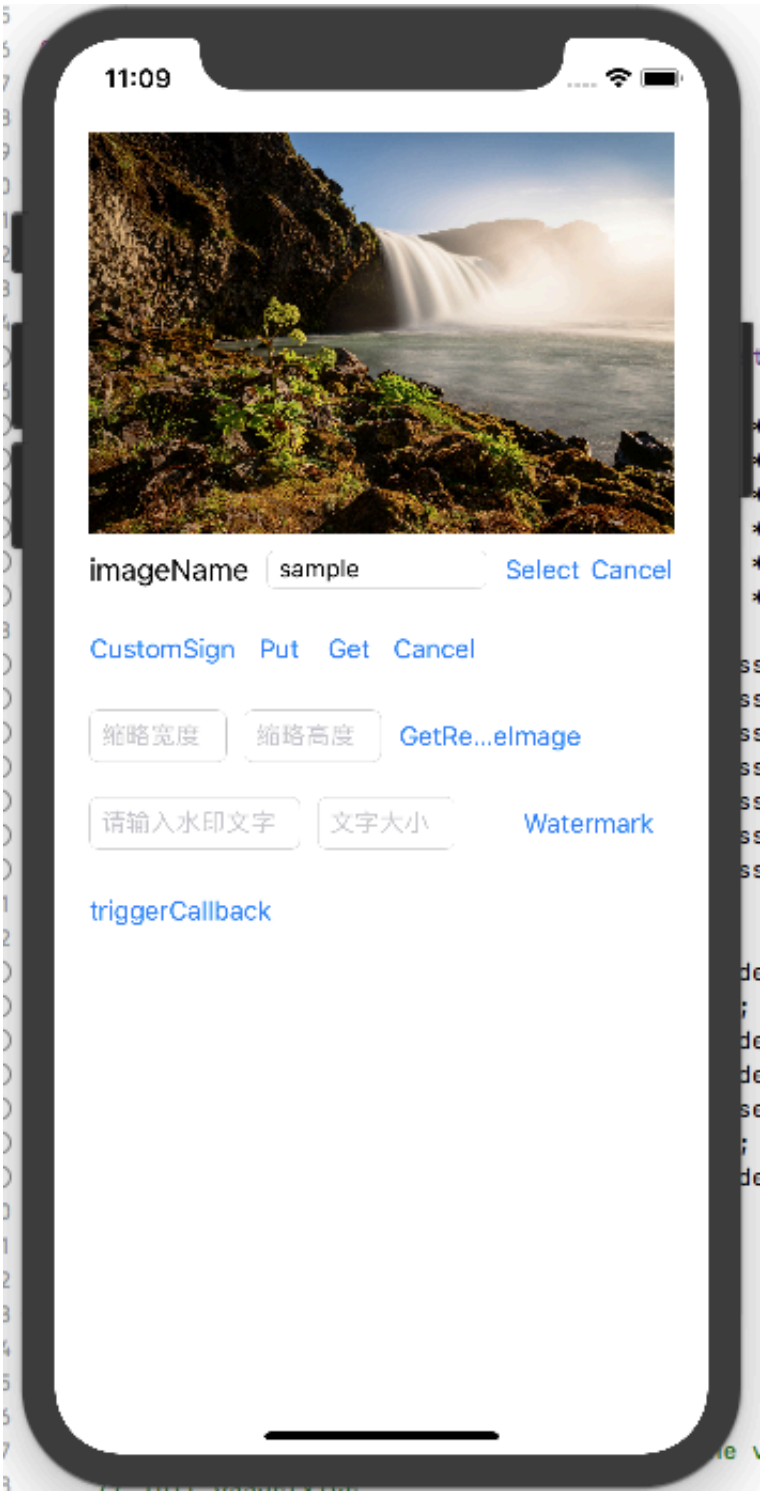
- *iOS demo* **示例**
- *Mac demo* **示例**
- *Swift demo* **示例**
- **测试用例**

您还可以直接 git clone 工程，并配置如下必要的参数：

AliyunOSSSDK-Example 〉 AliyunOSSSDK-iOS-Example 〉 Classes 〉 OSSConstants.m 〉 No Selection

```
//
//  Constant.m
//  AliyunOSSiOS
//
//  Created by jingdan on 2017/9/8.
//  Copyright © 2017年 zhouzhuo. All rights reserved.
//

#import "OSSConstants.h"

NSString * const BUCKET_NAME = @"bucket_name";
NSString * const UPLOAD_OBJECT_KEY = @"upload_object_key";
NSString * const DOWNLOAD_OBJECT_KEY = @"download_object_key";
NSString * const endPoint = @"http://oss-cn-shenzhen.aliyuncs.com";
NSString * const imageEndPoint = @"http://img-cn-shenzhen.aliyuncs.com";
NSString * const callbackAddress = @"http://oss-demo.aliyuncs.com:23450";
NSString * const STS_AUTH_URL = @"http://*.*.*.*:****/sts/getsts";

```

运行工程 demo 如下：



示例

以下演示了在 iOS 平台以及 Mac 平台上传和下载文件的基本流程。

- iOS 平台

1. 需要添加的引用

```
#import <AliyunOSSiOS/OSSService.h>
```

2. 初始化 OSSClient

初始化 OSSClient 需完成 Endpoint 设置、鉴权方式设置、Client 参数设置。其中，鉴权方式包含明文设置模式、自签名模式、STS 鉴权模式。 如果要使用 STS鉴权请先阅读访问控制章节以了解 RAM 的基础知识。以下内容假设您已开通 RAM 服务并了解 RAM 相关内容，比如如何获取子账户 AccessKeyId，SecretKeyId 以及 RoleArn 信息。

完善脚本文件中 AccessKeyId，SecretKeyId 以及 RoleArn 参数信息。通过 Python 启动一个本机 http 服务。在客户端代码中访问本地服务从而获得 StsToken.AccessKeyId、StsToken.SecretKeyId 以及 StsToken.SecurityToken。

更多信息可查看 sample 中的*STS* 使用说明。

```
NSString *endpoint = "https://oss-cn-hangzhou.aliyuncs.com";

// 移动端建议使用STS方式初始化OSSClient。可以通过sample中STS使用说明了解更多(https://github.com/aliyun/aliyun-oss-ios-sdk/tree/master/DemoByOC)
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentialProvider alloc] initWithAccessKeyId:@"AccessKeyId" secretKeyId:@"AccessKeySecret" securityToken:@"SecurityToken"];

client = [[OSSClient alloc] initWithEndpoint:endpoint credentialProvider:credential];
```

通过 OSSClient 发起上传、下载请求是线程安全的，您可以并发执行多个任务。

3. 上传文件

假设您在控制台有已创建的 Bucket。SDK 的所有操作都会返回一个`OSSTask`，您可以为这个 Task 设置一个延续动作，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];
put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";
put.uploadingData = <NSData *>; // 直接上传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;
}];
// 可以等待任务完成
// [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];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
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.downloadedData);
    } else {
        NSLog(@"download object failed, error: %@" ,task.error);
    }
    return nil;
}];
// 如果需要阻塞等待任务完成
// [task waitUntilFinished];
```

- MAC 平台

  在MAC 平台中，除引入方式不一样以外，其他操作示例与 iOS 平台一致。

```
import <AliyunOSSOSX/AliyunOSSiOS.h>
```

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 credential
Provider: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 credential
Provider: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 credential
Provider: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

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。

创建存储空间

以下代码用于创建存储空间：

```
OSSCreateBucketRequest * create = [OSSCreateBucketRequest new];
create.bucketName = @"<bucketName>";
create.xOssACL = @"public-read";
create.location = @"oss-cn-hangzhou";

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;
}];
```

说明：

- **每个用户的 Bucket 数量不能超过 30 个。**
- **每个 Bucket 的名字全局唯一，即创建的 Bucket 不能和其他用户已经在使用的 Bucket 同名，否则会创建失败。**

- **创建的时候可以设置 Bucket ACL 权限，如果不设置 ACL，默认是 Private。**
- **创建成功结果返回 Bucket 所在数据中心。**

列举存储空间

以下代码用于列举请求者拥有的所有存储空间：

```
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
"]);
        }];
    }
    return nil;
}];
```

说明:

匿名访问不支持该操作。

列举存储空间中的文件

以下代码用于列举存储空间中的文件：

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
getBucket.bucketName = @"<bucketName>";
// getBucket.marker = @"";
// getBucket.prefix = @"";
// getBucket.delimiter = @"";
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;
}];
```

说明:

- 列举操作必须具备访问该存储空间的权限。
- 列举时可以通过 prefix，marker，delimiter 和 max-keys 对 list 做限定，指定仅返回部分结果。

删除存储空间

以下代码用于删除单个存储空间：

```
OSSDeleteBucketRequest * delete = [OSSDeleteBucketRequest new];
delete.bucketName = @"<bucketName>";
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;
}];
```

- 只有 Bucket 的拥有者才能删除这个Bucket。
- 为了防止误删除，OSS 不允许用户删除一个非空的 Bucket。

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

本文介绍如何进行简单上传。

从内存中上传或上传本地文件

上传文件时可以直接上传 OSSData，或者通过 NSURL 上传一个文件：

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 必填字段
put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";
put.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData
// 可选字段，可不设置
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent,
int64_t totalBytesExpectedToSend) {
    // 当前上传段长度、当前已经上传总长度、一共需要上传的总长度
    NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytes
ExpectedToSend);
};
// 以下可选字段的含义参考： https://docs.aliyun.com/#/pub/oss/api-
reference/object&PutObject
// put.contentType = @"";
// put.contentMd5 = @"";
// put.contentEncoding = @"";
// put.contentDisposition = @"";
// put.objectMeta = [NSMutableDictionary dictionaryWithObjectsAndKeys
:@"value1", @"x-oss-meta-name1", nil]; // 可以在上传时设置元信息或者其他
HTTP头部
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];
```

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 给用户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字以“/”结尾的文件以实现上传下载，只是控制台会对以“/”结尾的文件以文件夹的方式展示。

上传文件时，如果把 ObjectKey 写为`"folder/subfolder/file"`，即是模拟了把文件上传到`folder/subfolder/`下的`file`文件。注意，路径默认是根目录，不需要以正斜线 / 开头。

上传时设置 Content-Type 和开启校验 MD5

上传时可以显式指定 ContentType，如果没有指定，SDK 会根据文件名或者上传的ObjectKey 自动判断。另外，上传 Object 时如果设置了 Content-MD5，那么 OSS 会用 Content-MD5 检查消息内容是否与发送时一致。SDK 提供了方便的 Base64 和 MD5 计算方法。

```
OSSPutObjectRequest * put = [OSSPutObjectRequest new];
// 必填字段
put.bucketName = @"<bucketName>";
put.objectKey = @"<objectKey>";
put.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// put.uploadingData = <NSData *>; // 直接上传NSData
// 设置Content-Type，可选
put.contentType = @"application/octet-stream";
// 设置MD5校验，可选
put.contentMd5 = [OSSUtil base64Md5ForFilePath:@"<filePath>"]; // 如果
是文件路径
// put.contentMd5 = [OSSUtil base64Md5ForData:<NSData *>]; // 如果是二进
制数据
// 进度设置，可选
put.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent,
int64_t totalBytesExpectedToSend) {
    // 当前上传段长度、当前已经上传总长度、一共需要上传的总长度
    NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytes
ExpectedToSend);
};
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

本文档介绍分片上传文件的完整流程。

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>";
NSString * uploadObjectkey = @"<objectKey>";
OSSInitMultipartUploadRequest * init = [OSSInitMultipartUploadRequest
new];
init.bucketName = uploadToBucket;
init.objectKey = uploadObjectkey;
// init.contentType = @"application/octet-stream";
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;
}
```

说明:

- **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。**
- **multipartUploadInit 返回的结果中包含 UploadId，UploadId 是区分分片上传的唯一标示。**

上传分片

以下代码用于上传分片：

```
NSString * filePath = [docDir stringByAppendingPathComponent:@"***"];
//要上传的文件
int chuckCount = *;
//分片上传数量
uint64_t offset = fileSie/chuckCount;
//分片大小
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] attributes
OfItemAtPath:uploadPart.uploadPartFileURL.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。

注意:

- 每一个分片上传请求需指定 UploadId 和 PartNum 。
- uploadPart 要求除最后一个 Part 外，其他的 Part 大小都要大于100KB。但是 Upload Part 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 号码的范围是1~10000。如果超出这个范围，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。

完成分片上传

下面代码中的 partInfos 就是进行分片上传中保存的 partETag 的列表，OSS 收到用户提交的 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。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipart
UploadRequest 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 Server Callback 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 result.serverReturnJsonString 查看servercallback 结果。

```
OSSCompleteMultipartUploadRequest * complete = [OSSCompleteMultipart
UploadRequest 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.serverRetu
rnJsonString);
    } else {
        // ...
    }
    return nil;
}] waitUntilFinished];
```

说明:

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 服务器会帮助用户进行 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一致性。

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，则只会返回 1000 个 Multipart Upload 信息，且返回结果中 IsTruncated 为 false，并返回 NextPartNumberMarker 作为下此读取的起点。

取消分片上传

下面代码取消了对应 UploadId 的分片上传请求。

```
OSSAbortMultipartUploadRequest * abort = [OSSAbortMultipartUpl
oadRequest 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

本文档介绍如何使用追加上传。

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。

以下代码用于追加上传：

```
OSSAppendObjectRequest * append = [OSSAppendObjectRequest new];
// 必填字段
append.bucketName = @"<bucketName>";
append.objectKey = @"<objectKey>";
append.appendPosition = 0; // 指定从何处进行追加
NSString * docDir = [self getDocumentDirectory];
append.uploadingFileURL = [NSURL fileURLWithPath:@"<filepath>"];
// 可选字段
append.uploadProgress = ^(int64_t bytesSent, int64_t totalByteSent,
int64_t totalBytesExpectedToSend) {
    NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytes
ExpectedToSend);
};
// 以下可选字段的含义参考：https://docs.aliyun.com/#/pub/oss/api-reference
/object&AppendObject
// 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;
}];
```

10.6.5 断点续传上传

本文介绍如何使用断点续传上传。

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，SDK提供了断点续传上传功能。

注意:

- 断点续传上传暂时只支持上传本地文件。
- 对于移动端来说，如果不是比较大的文件，不建议使用这种方式上传，因为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。

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。

- 断点续传上传只在本次上传生效。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，您可以通过为Bucket设置lifeCycle规则的方式来定时清理碎片。详情请参考：生命周期管理。

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。如果本地保留记录时间过长，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。

说明:

- 断点续传上传需要依赖InitMultipartUpload、UploadPart、ListParts、CompleteMultipartUpload、AbortMult
  些接口来实现。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。
- 断点续传上传也支持上传后回调通知，使用用法与常见的上传回调通知一致。

- 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。

- 在本地持久保存断点记录的调用方式：

```
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRe
quest 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。

- 以下代码用于实现断点续传上传：

```
// 获得UploadId进行上传，如果任务失败并且可以续传，利用同一个UploadId可以上传
同一文件到同一个OSS上的存储对象
OSSResumableUploadRequest * resumableUpload = [OSSResumableUploadRe
quest new];
resumableUpload.bucketName = <bucketName>;
resumableUpload.objectKey = <objectKey>;
resumableUpload.partSize = 1024 * 1024;
resumableUpload.uploadProgress = ^(int64_t bytesSent, int64_t
totalByteSent, int64_t totalBytesExpectedToSend) {
    NSLog(@"%lld, %lld, %lld", bytesSent, totalByteSent, totalBytes
ExpectedToSend);
};
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(
NSCachesDirectory, NSUserDomainMask, YES) firstObject];
//设置断点记录文件
resumableUpload.recordDirectoryPath = cachesDir;
//设置NO，取消时，不删除断点记录文件，如果不进行设置，默认YES，是会删除断点记录
文件，下次再进行上传时会重新上传。
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 == OSSClientErrorCodeCannotResumeUpload) {
            // 该任务无法续传，需要获取新的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, totalBytes
ExpectedToSend);
};
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: %@, servercall
back: %@",
              result.requestId,
              result.httpResponseHeaderFields,
              result.serverReturnJsonString);
    }
    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 totalBytes
Written, int64_t totalBytesExpectedToWrite) {
    // 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.dowloadedData);
    } 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`的分段回调功能。如果设置了这个回调，下载的结果将不再包含实际数据。

以下代码用于流式下载：

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
// required
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
// 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。

以下代码用于范围下载：

```
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 totalBytes
Written, int64_t totalBytesExpectedToWrite) {
    NSLog(@"%lld, %lld, %lld", bytesWritten, totalBytesWritten,
totalBytesExpectedToWrite);
};
OSSTask * getTask = [client getObject:request];
[getTask continueWithBlock:^id(OSSTask *task) {
```

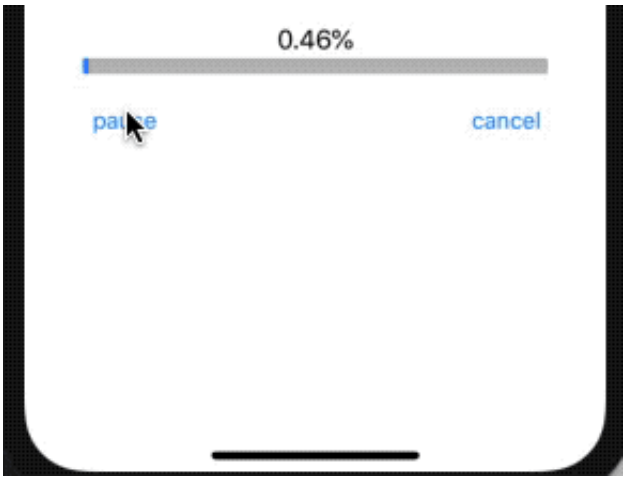
```
    if (!task.error) {
        NSLog(@"download object success!");
        OSSGetObjectResult * getResult = task.result;
        NSLog(@"download result: %@", getResult.dowloadedData);
    } else {
        NSLog(@"download object failed, error: %@" ,task.error);
    }
    return nil;
}];
// [getTask waitUntilFinished];
// [request cancel];
```

10.7.5 断点续传下载

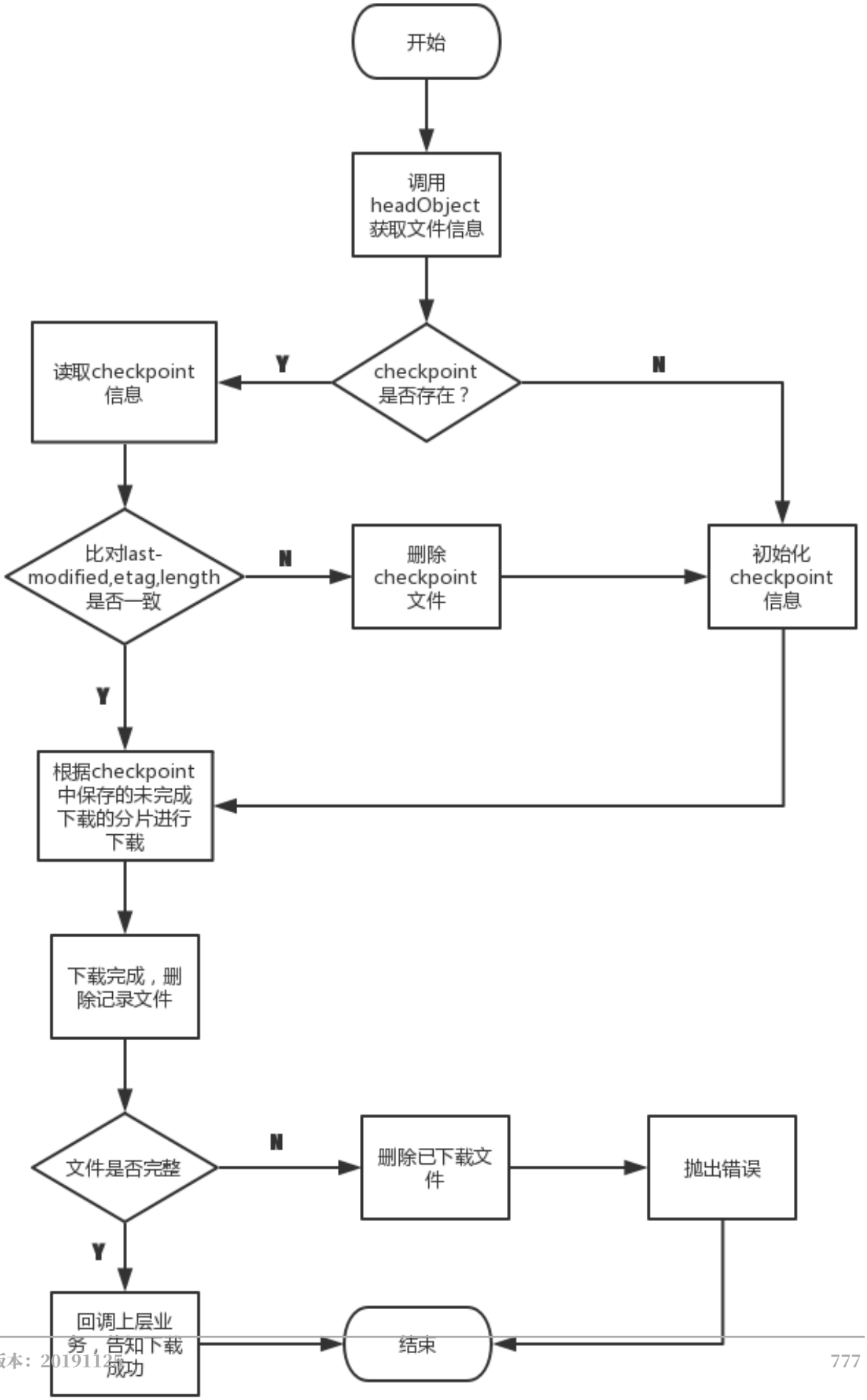
断点续传下载是指客户端在从网络上下载资源时，由于某种原因中断下载。再次开启下载时可以从已下载完成的部分开始继续下载未完成的部分，从而节省时间和流量。

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：


开始
调用 headObject 获取文件信息
checkpoint 是否存在？
Y
N
读取checkpoint 信息
比对last-modified,etag,length 是否一致
N
删除 checkpoint 文件
初始化 checkpoint 信息
Y
根据checkpoint 中保存的未完成下载的分片进行下载
下载完成，删除记录文件
文件是否完整
N
删除已下载文件
抛出错误
Y
回调上层业务，告知下载成功
结束

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，比如视频。**
  - `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, NSURLSessi
onDataDelegate>

@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 totalReceivedContent
Length; //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 completionHandler:^(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 objectForK
ey:@"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;
    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.totalReceivedContentLength];
        [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.totalRecei
vedContentLength + httpResponse.expectedContentLength;
        }
    }

    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:(NSURLSessi
onDataTask *)dataTask didReceiveResponse:(NSURLResponse *)response
 completionHandler:(void (^)(NSURLSessionResponseDisposition))
completionHandler
{
    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.totalRecei
vedContentLength +  httpResponse.expectedContentLength;
        }
    }

    completionHandler(NSURLSessionResponseAllow);
}

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessi
onDataTask *)dataTask didReceiveData:(NSData *)data {

    NSFileHandle *fileHandle = [NSFileHandle fileHandleForWriting
AtPath: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 = [[OSSPlainTe
xtAKSKPairCredentialProvider 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 presignConstrainURLW
ithBucketName:@"aliyun-dhc-shanghai" withObjectKey:OSS_DOWNLO
AD_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 presignConstrainURLWithBucketN
ame:@"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, NSUserDomainMask, YES).firstObject;
    _downloadRequest.downloadFilePath = [documentPath stringByAp
pendingPathComponent:OSS_DOWNLOAD_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 totalBytesExpectToReceived) {
        // totalBytesReceived是当前客户端已经缓存了的字节数，totalBytes
ExpectToReceived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 / totalBytes
ExpectToReceived;
            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: UIControlS
tateNormal];
        _downloadService = [DownloadService downloadServiceWithRequest
:_downloadRequest];
        [_downloadService resume];
    } else {
        [_downloadButton setTitle:@"download" forState: UIControlS
tateNormal];
        [_downloadService pause];
    }
}

- (IBAction)cancelDownloadClicked:(id)sender {
    [_downloadButton setTitle:@"download" forState: UIControlS
tateNormal];
    [_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。

10.8 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。

以下代码用于判断指定文件是否存在：

```
NSError * error = nil;
BOOL isExist = [client doesObjectExistInBucket:TEST_BUCKET withObject
Key:@"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

本文档介绍如何列举存储空间（Bucket）中的所有文件（Object）。

以下代码用于列举存储空间中的所有文件：

```
OSSGetBucketRequest * getBucket = [OSSGetBucketRequest new];
getBucket.bucketName = @"<bucketName>";

// 可选参数，具体含义参考：https://docs.aliyun.com/#/pub/oss/api-reference
/bucket&GetBucket
// getBucket.marker = @"";
// getBucket.prefix = @"";
// getBucket.delimiter = @"";

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;
}];
```

列举操作具体可设置的参数名称和作用如下：

名称	作用
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: CommonPrefixes。
marker	设定结果从marker之后按字母排序的第一个开始返回。
maxkeys	限定此次返回object的最大数，如果不设定，默认为100，maxkeys取值不能大于1000。
prefix	限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix。

10.8.5 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：

```
OSSCopyObjectRequest * copy = [OSSCopyObjectRequest new];
copy.bucketName = @<bucketName>;
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
}];
```

注意:

- 源 Object 和目标 Object 必须属于同一个数据中心。
- 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 Object 的文件元信息。
- 通过 CopyObject 拷贝文件大小不能超过 1G，超过 1G 需使用 Multipart Upload 。

10.8.6 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。

以下代码用于删除文件：

```
OSSDeleteObjectRequest * delete = [OSSDeleteObjectRequest new];
delete.bucketName = @"<bucketName>";
delete.objectKey = @"<objectKey>";

OSSTask * deleteTask = [client deleteObject:delete];

[deleteTask continueWithBlock:^id(OSSTask *task) {
    if (!task.error) {
        // ...
    }
    return nil;
}];

// [deleteTask waitUntilFinished];
```

说明:

DeleteObject 要求对 Object 所在的 Bucket 有写权限。

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 iOS SDK 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，只有在 OSS 服务器计算接收到文件的 MD5 值和上传提供的 MD5 一致时才可以上传成功，从而保证上传数据的完整性。

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
```

```
request.bucketName = BUCKET_NAME;
...
request.contentMd5 = [OSSUtil fileMD5String:filepath];
```

- **CRC 校验**

  与 MD5 相比，CRC64 可以同时上传并计算 CRC 值。

```
// 构造上传请求
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = OSS_BUCKET_PRIVATE;
///....
request.crcFlag = OSSRequestCRCOpen;
// 开启crc效验后。如果在传输中数据不一致，会提示OSSClientErrorCodeIn
validCRC 错误。
OSSTask * task = [_client putObject:request];
[[task continueWithBlock:^id(OSSTask *task) {
    //如果crc效验失败，会有error
    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。

在读取下载数据流时，如果开启了 CRC 效验，会在读取完数据流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 = ...;
//开启crc效验
request.crcFlag = OSSRequestCRCOpen;

OSSTask * task = [testProxyClient getObject:request];

[[task continueWithBlock:^id(OSSTask *task) {
 //如果开启了crc效验，并有在传输中有数据错误的情况。会提示OSSClientErrorCodeIn
validCRC的错误
    XCTAssertNil(task.error);
    return nil;
}] waitUntilFinished];

注：如果设置onReceiveData block。开启CRC效验后，需要自行比较CRC数值是否一致。范
例如下
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = ....
request.crcFlag = OSSRequestCRCOpen;
....

__block uint64_t localCrc64 = 0;
 //如果设置onReceiveData block
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。

STS鉴权模式

OSS可以通过阿里云STS服务，临时进行授权访问。阿里云STS (Security Token Service) 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联邦用户（用户身份由您自己管理）颁发一个自定义时效和权限的访问凭证，App端称为FederationToken。 第三方应用或联邦用户可以使用该访问凭证直接调用阿里云产品API，或者使用阿里云产品提供的SDK来访问云产品API。

- **使用STS的好处：**
  - **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详细步骤**
  1. **App用户登录。App用户身份由您自己管理。您可以自定义身份管理系统，也可以使用外部Web账号或OpenID。对于每个有效的App用户来说，AppServer可以确切地定义出每个App用户的最小访问权限。**
  2. **AppServer请求STS服务获取一个安全令牌（SecurityToken）。在调用STS之前，AppServer需要确定App用户的最小访问权限（用Policy语法描述）以及授权的过期时间。 然后通过调用STS的AssumeRole（扮演角色）接口来获取安全令牌。角色管理与使用相关内容请参考RAM使用指南中的角色管理。**
  3. **STS返回给AppServer一个有效的访问凭证，App端称为FederationToken，包括一个安全令牌（SecurityToken）、临时访问密钥（AccessKeyId, AccessKeySecret）以及过期时间。**
  4. **AppServer将FederationToken返回给ClientApp。ClientApp可以缓存这个凭证。当凭证失效时，ClientApp需要向AppServer申请新的有效访问凭证。例如访问凭证有效期为1小时，那么ClientApp可以每30分钟向AppServer请求更新访问凭证。**
  5. **ClientApp使用本地缓存的FederationToken去请求Aliyun Service API。云服务会感知STS访问凭证，并会依赖STS服务来验证访问凭证，并正确响应用户请求。**

注意:

使用这种模式授权需要先开通阿里云RAM服务。

- 直接设置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）获取一对StsToken，然后用它来初始化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新的StsToken到SDK中。

  初始化代码为：

```
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentia
lProvider alloc] initWithAccessKeyId:@"<StsToken.AccessKeyId>"
secretKeyId:@"<StsToken.SecretKeyId>" securityToken:@"<StsToken.
SecurityToken>"];
client = [[OSSClient alloc] initWithEndpoint:endpoint credential
Provider:credential];
```

  在您判断到Token即将过期时，您可以重新构造新的OSSClient，也可以通过如下方式更新CredentialProvider：

```
id<OSSCredentialProvider> credential = [[OSSStsTokenCredentia
lProvider alloc] initWithAccessKeyId:@"<StsToken.AccessKeyId>"
secretKeyId:@"<StsToken.SecretKeyId>" securityToken:@"<StsToken.
SecurityToken>"];
client = [[OSSClient alloc] initWithEndpoint:endpoint credential
Provider:credential];
```

- 实现获取StsToken回调

  如果您期望SDK自动更新Token，则要求您在SDK的应用中实现回调。这个回调通过您实现的方式去获取一个Federation Token（即StsToken），然后返回。SDK会利用这个Token来进行加签处理，并在需要更新时主动调用这个回调来获取Token：

```
id<OSSCredentialProvider> credential = [[OSSFederationCredent
ialProvider alloc] initWithFederationTokenGetter:^OSSFederat
ionToken * {
    // 您需要在这里实现获取一个FederationToken，并构造成OSSFederat
ionToken对象返回
    // 如果因为某种原因获取失败，可直接返回nil
      OSSFederationToken * token;
    // 下面是一些获取token的代码，比如从您的server获取
    ...
    return token;
}];
client = [[OSSClient alloc] initWithEndpoint:endpoint credential
Provider:credential];
```

  说明：

此外，如果您已经通过别的方式获取了token所需的各个字段，也可以在这个回调中直接返回。但您需要手动处理token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，并假设访问这个地址，返回的数据如下：

```
{
    "StatusCode": 200,
    "AccessKeyId":"STS.iA645eTOXEqP3cg3VeHf",
    "AccessKeySecret":"rV3VQrpFQ4BsyHSAvi5NVLpPIVffDJv4LojUBZCf",
    "Expiration":"2015-11-03T09:52:59Z",
    "SecurityToken":"CAES7QIIARKAAZPlqaN9ILiQZPS+JDkS/GSZN45RLx4YS
/p3OgaUC+oJl3XSlbJ7StKpQ...."}
```

那么，您可以这么实现一个`OSSFederationCredentialProvider`实例：

```
id<OSSCredentialProvider> credential2 = [[OSSFederationCredent
ialProvider alloc] initWithFederationTokenGetter:^OSSFederat
ionToken * {
    // 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 {
        // 返回数据是json格式，需要解析得到token的各个字段
        NSDictionary * object = [NSJSONSerialization JSONObject
WithData: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

```
id<OSSCredentialProvider> credential = [[OSSCustomSignerCrede
ntialProvider alloc] initWithImplementedSigner:^NSString *(NSString *
contentToSign, NSError *__autoreleasing *error) {
    // 您需要在这里依照OSS规定的签名算法，实现加签一串字符内容，并把得到的签名传拼
接上AccessKeyId后返回
    // 一般实现是，将字符内容post到您的业务服务器，然后返回签名
    // 如果因为某种原因加签失败，描述error信息后，返回nil
NSString *signature = [OSSUtil calBase64Sha1WithData:contentToSign
withSecret:@"<your accessKeySecret>"]; // 这里是用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 accessKeyI
d>", signature];
}];

client = [[OSSClient alloc] initWithEndpoint:endpoint credential
Provider:credential];
```

注意:

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。如果您在其中实现了向业务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具体请求的时候，在请求的子线程中执行，所以不会阻塞主线程。

10.12 图片处理

OSS图片处理是OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。 用户将原始图片上传保存到OSS，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。 图片处理提供图片处理接口，图片上传请使用上传接口。基于OSS图片处理，用户可以搭建自己的图片处理服务。

图片处理的详细信息请参见*OSS图片处理指南*。

图片处理功能

OSS图片处理提供以下功能：

- **图片缩放**
- **图片裁剪**
- **图片旋转**
- **图片效果**
- **格式转换**
- **图片水印，包括添加图片、文字、图文混合水印**

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读写，必须通过授权才能进行访问。

- **匿名访问**

```
OSSTask * task = [_client presignPublicURLWithBucketName:_publicBuc
ketName
                                                 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];
```

说明：

需要对图片进行其它处理，只需替换request.setxOssProcess()的相关参数。

- SDK 访问

```
OSSGetObjectRequest * request = [OSSGetObjectRequest new];
request.bucketName = @"bucket-name";     // 设置存储桶名称
request.objectKey = @"image-name";       // 设置图片名称
request.xOssProcess = @"image/resize,m_lfit,w_100,h_100";   // 设置图片处理
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";     // 图片处理效果
//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：ClientError 和 ServerError。

ClientError 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#unique_482进行排查。

Error类型	Error Domain	Code	UserInfo	描述	解决方法
		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#unique_483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	OSSClientErrorCodeExcpetionCatched	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#unique_184/unique_184_Connect_42_section_zzm行排查。

11 Node.js

11.1 安装

本文介绍如何安装Node.js。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*，升级6.X请移步升级文档

- *Github*地址
- *API*文档
- *ChangeLog*

环境准备

- 要求
  - 开通阿里云OSS服务，如果您还不了解阿里云OSS服务，请登录OSS产品主页了解。
  - 创建AccessKeyId和AccessKeySecret，由于云账号AccessKey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子账号或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。详情请参见访问控制。
- 环境要求

  OSS NodeJS SDK基于*Node.js*环境构建。

使用方式

OSS NodeJS SDK同时支持同步和异步的使用方式

- 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

无论同步方式还是异步方式，均使用`new OSS()`创建client。

下面分别举例，先上传一个文件，然后立即下载这个文件：

- 同步方式

```
let client = new OSS(...);
async function put () {
  try {
    // object表示上传到OSS的Object名称，localfile表示本地文件或者文件路径
    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', err);
  }
}
put();
```

- **异步方式**

```
let client = new OSS(...);

// object表示上传到OSS的Object名称，localfile表示本地文件或者文件路径
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', err);
});
```

安装

- **使用Node.js**

  支持的Node.js版本：

  Node.js >= 8.0.0 如果需要在 Node.js < 8 的环境中使用，请使用 ali-oss 4.x版本。

  首先使用*npm*安装SDK的开发包：

```
npm install ali-oss
```

  然后在Vue、React等程序中依赖注入即可使用：

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<oss region>',
  //云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，部署在服务端使用RAM子账号或STS，部署在客户端使用STS。
  accessKeyId: '<Your accessKeyId>',
  accessKeySecret: '<Your accessKeySecret>',
  bucket: '<Your bucket name>'
});
```

- **如果使用npm遇到网络问题，可以使用淘宝提供的npm镜像：** *cnpm*

11.2 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AccessKey。
- [accessKeySecret] {String}：通过阿里云控制台创建的AccessSecret。
- [stsToken] {String}：使用临时授权方式，详情请参见使用 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。

Node.js示例如下：

```
var oss = require('ali-oss');

var store = oss({
  accessKeyId: 'your access key',
  accessKeySecret: 'your access secret',
  bucket: 'your bucket name',
  region: 'oss-cn-hangzhou'
});
```

11.3 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。

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。

更多 Bucket 接口详情请参见*GitHub*。

查看文件列表

修改`app.js`，使用`list`接口查看文件列表：

```
client.useBucket('Your bucket name');
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('Your bucket name');

async function put () {
  try {
    let result = await client.put('object-name', 'local file');
    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：

```
async function get () {
  try {
    let result = await client.get('object-name');
    console.log(result);
  } catch (err) {
    console.log (err);
  }
```

```
}

get();
```

删除文件

修改`app.js`，使用`delete`接口删除某个文件：

```
async function delete () {
  try {
    let result = await client.delete('object-name');
    console.log(result);
  } catch (err) {
    console.log (err);
  }
}

delete();
```

更多 Object 接口详情请参见*GitHub*。

11.4 初始化

本文档介绍如何初始化 Node.js SDK。

创建一个文件`app.js`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 服务时的区域，例如`oss-cn-hangzhou`。完整的区域列表请参考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：默认为 60 秒，指定访问 OSS 的 API 的超时时间。

11.5 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。

创建 Bucket

使用putBucket接口创建一个 Bucket 时，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>'
});

// putBucket
async function putBucket() {
  try {
    const result = await client.putBucket('your bucket name');
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}

putBucket();
```

存储空间的命名规范，请参见基本概念中的命名规范。创建存储空间详情，请参见管理存储空间。

获取 endpoint 信息，请参见访问域名和数据中心文档。

查看所有 Bucket

使用listBuckets接口列出当前用户下的所有 Bucket。您还可以指定prefix参数，列出特定前缀的所有 Bucket：

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});

async function listBuckets() {
  try {
    const result = await client.listBuckets();
    console.log(result);
    const result2 = await client.listBuckets({
      prefix: 'prefix',
```

```
    });
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}

listBuckets();
```

删除 Bucket

使用`deleteBucket`接口删除一个 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();
```

注意：

- **如果该 Bucket 下还有文件，则需要先删除所有文件才能删除 Bucket。**
- **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。**

Bucket访问权限

用户可以设置 Bucket 的访问权限，允许或者禁止匿名用户对其内容进行读写。更多关于访问权限的内容请参考访问权限。

- **获取 Bucket 的访问权限（ACL）**

  通过`getBucketACL`查看 Bucket 的 ACL：

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});
```

```
async function getBucketACL() {
  try {
    const result = await client.getBucketACL('luozhang002');
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}

getBucketACL();
```

- **设置 Bucket 的访问权限（ACL）**

  通过`putBucketACL`设置 Bucket 的 ACL：

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>'
});

async function putBucketACL() {
  try {
    const result = await client.putBucketACL('bucket name', 'public-
read');
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}

putBucketACL();
```

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 Node.js SDK 提供了丰富的文件上传方式。

- **上传本地文件：最大不能超过 5GB。**
- **上传缓存中的内容：最大不能超过 5GB。**
- **流式上传：文件大小不限制。**
- **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。**
- **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。**

11.6.2 上传本地文件

本文档介绍如何将本地文件上传到 OSS。

> **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，请参考安装。

通过put接口将本地文件上传到 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 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。
    let result = await client.put('object-name', 'local-file');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}

put();
```

11.6.3 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:**
> 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，请参考安装。

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();
```

11.6.4 流式上传

本文档介绍如何进行流式上传。

> **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，请参考安装。**

通过putStream**接口来上传一个Stream中的内容。**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。当使用**putStream**接口时，SDK 默认会发起一个**chunked encoding**的 HTTP PUT 请求。如果在**options**指定了**contentLength**参数，则不会使用**chunked encoding**。**

```
let OSS = require('ali-oss');
let fs = require('fs');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function putStream () {
  try {
  // use 'chunked encoding'
  let stream = fs.createReadStream('local-file');
  let result = await client.putStream('object-name', stream);
  console.log(result);

  // don't use 'chunked encoding'
  let stream = fs.createReadStream('local-file');
  let size = fs.statSync('local-file').size;
  let result = await client.putStream(
    'object-name', stream, {contentLength: size});
  console.log(result);
  } catch (e) {
    console.log(e)
  }
}
```

```
putStream();
```

11.6.5 分片上传

本文档介绍如何使用分片上传。

> **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，请参考安装。

在需要上传的文件较大时，可以通过`multipartUpload`接口进行分片上传。分片上传的好处是将一个大请求分成多个小请求来执行，这样当其中一些请求失败后，不需要重新上传整个文件，只需要上传失败的分片即可。对于大于 100MB 的文件，建议采用分片上传的方法。

在使用 multipartUpload 接口时如果遇到`ConnectionTimeoutError`超时问题，业务方需要自行处理超时逻辑。您可以通过缩小分片大小、加大超时时间、重试请求或者捕获`ConnectionTimeoutError`错误来分析具体原因。

相关参数说明如下：

- name {String}： object 名称
- file {String|File}： 文件路径或者 HTML5 文件

- [options] {Object}：额外参数
  - [checkpoint] {Object}：断点记录点。如果设置这个参数，上传会从断点开始，如果没有设置这个参数，则会重新上传
  - [parallel] {Number}：并发上传的分片个数
  - [partSize] {Number}：分片大小
  - [progress] {Function}： async 函数形式。回调函数包含三个参数：
    - percentage {Number}：进度百分比（0-1之间小数）
    - checkpoint {Object}：断点记录点
    - res {Object}：单个分片成功返回的 response
  - [meta] {Object}：用户自定义 header meta 信息。header 前缀`x-oss-meta-`
  - [mime] {String}：设置 Content-Type 请求头
  - [headers] {Object}：其他 headers，详情请参见*RFC 2616*
    - '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。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。如 `Content-Disposition: somename`
    - 'Content-Encoding'：用于对特定媒体类型的数据进行压缩。如`Content-Encoding: gzip`
    - 'Expires'：过期时间，单位为毫秒。如`Expires: 3600000`

以下代码用于分片上传：

```
let OSS = require('ali-oss')

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function multipartUpload () {
  try {
    let result = await client.multipartUpload('object-name', 'local-
file', {
    progress,
 meta: {
   year: 2017,
   people: 'test'
 }
  });
  console.log(result);
  let head = await client.head('object-name');
  console.log(head);
```

```
  } catch (e) {
    // 捕获超时异常
 if (e.code === 'ConnectionTimeoutError') {
   console.log("Woops,超时啦!");
   // do ConnectionTimeoutError operation
 }
    console.log(e)
  }
}
```

说明:

meta参数是一个用户自定义的元数据，通过 head 接口可以获取到 object 的 meta 数据。

```
const progress = async function (p) {
  console.log(p);
};
```

说明:

progress参数是一个进度回调函数，用于获取上传进度。progress可以是一个 async 函数。

11.6.6 追加上传

本文介绍如何使用追加上传。

追加上传是指使用AppendObject方法在已上传的Appendable Object类型文件后面直接追加内容。

说明:

- 只有通过追加上传创建的文件才可以后续继续被追加上传。也就是说，其他通过简单上传、表单上传、分片上传得到的文件，不能在这些文件后面追加上传新的内容。
- 追加类型的文件（Append Object）暂时不支持CopyObject操作。

追加上传详情请参考开发指南中的#unique_75。

以下代码用于追加上传文件：

```
let OSS = require('ali-oss')

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function append () {
  try {
    let result = await client.append('object-name', 'local-file');
    console.log(result);
  } catch (e) {
    console.log(e);
```

```
  }
}

append();
```

说明:

追加上传的`[position] {String}`**参数表示最近上传Object的Content-Length所在的位置，其他参数设置可参考***GitHub*。

11.6.7 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，请参考安装。

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*#unique_499*。

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。

11.6.8 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）：

```
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 = {
  callback: {
    // 您的回调服务器地址，如http://oss-demo.aliyuncs.com:23450或http://
127.0.0.1:9090。
    url: '<callbackUrl>',
    // 设置回调请求消息头中Host的值，如oss-cn-hangzhou.aliyuncs.com。
    host: '<callbackHost>',
    // 设置发起回调请求的Content-Type。
    body: 'bucket=${bucket}&object=${object}&var1=${x:var1}',
    contentType: 'application/x-www-form-urlencoded',
    // callback-var 设置发起回调请求的自定义参数。
    customValue: {
      var1: 'value1',
      var2: 'value2'
    }
  }
}
const result = await store.put('<object name>', options)
```

上传回调的各参数详情请参见*#unique_274*。上传回调原理介绍请参见开发指南的*#unique_77*。

11.7 下载文件

11.7.1 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**
- *HTTP* **下载**

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。

您可以通过`get`**接口将 OSS 文件下载到本地文件：**

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function get () {
  try {
    let result = await client.get('object-name', 'local-file');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}

get();
```

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。

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();
```

11.7.4 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。

使用`getStream`下载文件时，返回的`Readable Stream`用于流式地处理文件内容。

```
let OSS = require('ali-oss');
let fs = require('fs');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function getStream () {
  try {
    let result = await client.getStream('object-name');
    console.log(result);
    let writeStream = fs.createWriteStream('local-file');
    result.stream.pipe(writeStream);
  } catch (e) {
    console.log(e);
  }
}

getStream()
```

11.7.5 HTTP 下载

您可以在不使用SDK的情况下，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。文件的URL由SDK生成。

> **说明:**
>
> **使用`signatureUrl`方法生成可下载的HTTP地址，URL的有效时间默认设置为3600秒，无最大有效时间限制，但出于安全考虑，建议设置的URL有效时间不宜过长。**

使用HTTP下载OSS中的文件的示例代码如下：

```
let OSS = require('ali-oss');

let client = new OSS({
```

```
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

let url = client.signatureUrl('object-name');
console.log(url);

let url = client.signatureUrl('object-name', {expires: 3600});
console.log(url);

// signed URL for PUT
let url = client.signatureUrl('object-name', {method: 'PUT'});
console.log(url);
```

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：

- **判断文件是否存在**
- **管理文件元信息**
- **列举指定 *Bucket* 下的所有文件**
- **列举指定目录下的文件和子目录**
- **拷贝文件**
- **删除文件**

11.8.2 判断文件是否存在

本文介绍如何判断文件是否存在。

通过get接口判断文件是否存在：

```
client.get(object).then((result) => {
      if (result.res.status == 200) {
        return true
      }
  }).catch((e)=> {
    if (e.code == 'NoSuchKey') {
      return false
    }
  })
```

说明：

get**接口的更多应用，请参见**下载文件。

11.8.3 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();
```

11.8.4 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#unique_516。

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#unique_517。

11.8.5 列举指定Bucket下的所有文件

本文介绍如何列举指定 Bucket 下的所有文件。

通过`list`接口列举当前 Bucket 下的所有文件。主要参数说明如下：

- prefix：只列出符合特定前缀的文件。
- marker：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。

以下代码用于列举指定 Bucket下的所有文件：

```
let OSS = require('ali-oss');
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});
async function list () {
  {
    // 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
      });
    }
  // 列举前缀为'my-'的文件。
  let result = await client.list({
     prefix: 'my-'
  });
  console.log(result);
  // 列举前缀为'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();
```

11.8.6 列举指定目录下的文件和子目录

本文介绍如何列举指定目录下的文件和子目录。

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 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，像传统的文件一样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见*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
```

接下来我们调用函数`listDir`来列举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)
  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);
  });
end
```

运行结果如下：

```
> await listDir('foo/')
=> SubDir: foo/bar/
   SubDir: foo/hello/
   Object: foo/x
   Object: foo/y

> await listDir('foo/bar/')
=> Object: foo/bar/a
   Object: foo/bar/b

> await listDir('foo/hello/C/')
=> Object: foo/hello/C/1
   Object: foo/hello/C/2
   ...
   Object: foo/hello/C/9999
```

11.8.7 拷贝文件

本文介绍如何拷贝文件（Object）。

拷贝文件时，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元信息。**

以下代码用于拷贝同一个Bucket内的文件：

```
let OSS = require('ali-oss');
var client = new OSS({
  bucket: 'bucket',
  region: 'oss-cn-xxx',
  accessKeyId: 'ak',
  accessKeySecret: 'aksecret',
//   secure: true
})
// 拷贝同一个Bucket内的文件。
client.copy('test_copy', 'test').then((res) => {
  console.log(res);
}).catch(e => {
  console.log(e);
})
```

以下代码用于拷贝同一个Region下的两个不同Bucket内的文件：

```
let OSS = require('ali-oss')
```

```
let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
});

async function copy () {
  try {
     // 拷贝同一个Region下的两个不同Bucket内的文件时，源Object名字应为 '/
bucket/object' 的形式。
    let result = await client.copy('to', '/from-bucket/from');
    console.log(result);

    // 拷贝元信息。
    let result = await client.copy('to', 'from');
    console.log(result);

    // 覆盖元信息。
    let result = await client.copy('to', 'from', {
      meta: {
        year: 2015,
        people: 'mary'
      }
    });
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
```

11.8.8 删除文件

本文介绍如何删除文件。

删除单个文件

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-name');
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}

delete();
```

删除多个文件

通过`deleteMulti`来删除多个文件，并通过`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();
```

11.8.9 解冻归档文件

本文介绍如何解冻归档文件。

归档类型（Archive）的文件需要解冻（Restore）之后才能读取。非归档类型的文件，不要调用RestoreObject方法。

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，最长等待任务完成时间为4小时。**
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：

说明:

解冻归档文件的相关参数说明请参考*GitHub*。

```
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。

11.8.10 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：

```
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*。

11.9 自定义域名绑定

本文介绍如何使用自定义域名绑定。

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，在将资源迁移到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。

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，示例代码如下：**

```
let OSS = require('ali-oss')

let client = new OSS({
  endpoint: '<Your endpoint>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  cname: true
});

client.useBucket('my-bucket')
```

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。

11.10 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*#unique_102*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*#unique_103*。

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*#unique_522*。

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*#unique_523*。

11.11 授权访问

本文介绍如何进行授权访问。

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 STS， 详情请参见*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 的流程请参见开发指南中的 *STS* **临时授权访问** *OSS*。

在使用 STS 访问 OSS 时，需要设置`stsToken`**参数，如下面的例子所示：**

```
let OSS = require('ali-oss');
let STS = OSS.STS;
let sts = new STS({
  accessKeyId: '<子账号的AccessKeyId>',
  accessKeySecret: '<子账号的AccessKeySecret>'
});
async function assumeRole () {
  try {
    let token = await sts.assumeRole(
    '<role-arn>', '<policy>', '<expiration>', '<session-name>');
    let client = new OSS({
      region: '<region>',
      accessKeyId: token.credentials.AccessKeyId,
      accessKeySecret: token.credentials.AccessKeySecret,
      stsToken: token.credentials.SecurityToken,
      bucket: '<bucket-name>'
```

```
    });
  } catch (e) {
    console.log(e);
  }
}
assumeRole();
```

向 STS 申请临时 token 时，还可以指定自定义的 STS Policy。这样申请的临时权限是所扮演角色的权限与`Policy`指定的权限的交集。下面的例子将通过指定 STS Policy 申请对`my-bucket`的只读权限，并指定临时 token 的过期时间为 15 分钟。

```
let OSS = require('ali-oss');
let STS = OSS.STS;
let sts = new STS({
  accessKeyId: '<子账号的AccessKeyId>',
  accessKeySecret: '<子账号的AccessKeySecret>'
});
let policy = {
  "Statement": [
    {
      "Action": [
        "oss:Get*"
      ],
      "Effect": "Allow",
      "Resource": ["acs:oss:*:*:my-bucket/*"]
    }
  ],
  "Version": "1"
};
async function assumeRole () {
  try {
    let token = await sts.assumeRole(
    '<role-arn>', policy, 15 * 60, '<session-name>');
    let client = new OSS({
      region: '<region>',
      accessKeyId: token.credentials.AccessKeyId,
      accessKeySecret: token.credentials.AccessKeySecret,
      stsToken: token.credentials.SecurityToken,
      bucket: '<bucket-name>'
    });
  } catch (e) {
    console.log(e);
  }
}
assumeRole();
```

使用签名 URL 进行临时授权

- **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，您可以指定 URL 的过期时间来限制访客的访问时长。

- **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，默认是 1800 秒。其他参数相关说明，请参考 *Github*。

以下代码用于生成对象签名 URL：

```
let OSS = require('ali-oss');
let store = new OSS({
    bucket: '<your bucket>',
    region: '<your region>',
    accessKeyId: '<your accessKeyId>',
    accessKeySecret: '<your accessKeySecret>'
})
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
```

- **带有图片处理的签名 URL**

```
let OSS = require('ali-oss');
let store = new OSS({
    bucket: '<your bucket>',
    region: '<your region>',
    accessKeyId: '<your accessKeyId>',
    accessKeySecret: '<your accessKeySecret>'
})
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);
```

11.12 访问权限

OSS允许对Bucket和Object分别设置访问权限，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。

Bucket访问权限

对于Bucket，有三种访问权限：

- **Public-read-write：允许匿名用户对该存储空间内文件进行读写操作。**
- **Public-read：允许匿名用户对该存储空间内文件进行读操作。**
- **Private：不允许匿名访问，所有的访问都要经过签名。**

创建Bucket时，默认是private权限。创建Bucket后您可以通过`putBucketACL`来设置Bucket的权限，还可以通过`getBucketACL`来获取Bucket的权限。

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function bucketACL () {
  try {
     let result = await client.getBucketACL('bucket-name');
     console.log(result);
     let result = await client.putBucketACL('bucket-name', 'acl');
     console.log(result);
  } catch (e) {
    console.log(e);
  }
}

bucketACL();
```

Object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，默认为继承Bucket权限。之后您可以通过`putACL` 来设置Object的权限。

```
let OSS = require('ali-oss')
```

```
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function bucketACL () {
  try {
     let result = await client.getACL('my-object');
     console.log(result.acl); // default
     await client.putACL('my-object', 'public-read');
     result = await client.getACL('my-object');
     console.log(result.acl); // public-read
  } catch (e) {
    console.log(e);
  }
}
```

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`。

11.13 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。

每条规则包含：

- 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。
- 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。
- 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。
- 是否生效。

更多关于生命周期的内容请参考文件生命周期。

设置生命周期规则

通过putBucketLifecycle来设置生命周期规则：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function putBucketLifecycle () {
  try {
    let result = yield client.putBucketLifecycle('bucket-name', [
    {
      id: 'rule1',
      status: 'Enabled',
      prefix: 'foo/',
      days: 3
    },
    {
      id: 'rule2',
      status: 'Disabled',
      date: '2022-10-11T00:00:00.000Z'
    }
  ]);
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
putBucketLifecycle();
```

查看生命周期规则

通过getBucketLifecycle来查看生命周期规则：

```
let OSS = require('ali-oss')
let client = new OSS({
  region: '<Your region>'
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getBucketLifecycle () {
  try {
    let result = await client.getBucketLifecycle('bucket-name');
  console.log(result);
  } catch (e) {
    console.log(e);
  }
}
getBucketLifecycle();
```

清空生命周期规则

通过`deleteBucketLifecycle`设置一个空的Rule数组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();
```

11.14 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`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();
```

11.15 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

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();
```

11.16 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();
```

11.17 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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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

参数	描述
action	对图片做的操作，如缩放、裁剪、旋转等

参数	描述
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）

Bucket权限	Object权限
任意权限	私有（private）

以下代码用于生成带签名的图片处理URL：

```
let OSS = require('ali-oss');
let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});
// 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、 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：

```
let OSS = require('ali-oss');
let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});
// 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:

```
let OSS = require('ali-oss');
let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
```

```
});
// 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：

```
let OSS = require('ali-oss');
let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});
// 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。**

11.18 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，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：

- **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

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'
```

11.19 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**

12 Browser.js

12.1 安装

本文档介绍OSS Browser.js SDK的安装。

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*，如要升级至6.X请参考升级文档。

- *Github地址*
- *API文档*
- *ChangeLog*

环境准备

- 前提条件
  - 开通阿里云OSS服务。如果您还不了解阿里云OSS服务，请登录OSS产品主页了解。详情请参见产品介绍。
  - 创建AccessKeyId和AccessKeySecret。由于云账号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子账号或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。详情请参见访问控制。
- 环境要求

  OSS Browser.js SDK基于*Node.js*环境构建，通过*Browserify*和*Babel*产生适用于浏览器的代码。

  但是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，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，用签名URL的方式下载。
  - Bucket相关的操作（listBuckets/BucketACL/BucketLogging等），由于OSS暂时不支持Bucket相关的跨域请求，Bucket相关的操作可以在控制台进行。

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，具体的参数可参照文档配置项
let client = new OSS(...);

async function put () {
  try {
    // object表示上传到OSS的名字，可自己定义
    // file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，具体的参数可参照文档配置项
let client = new OSS(...);

// object表示上传到OSS的名字，可自己定义
// file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);
```

```
});
```

安装

- **在浏览器中使用**

  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**

  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="http://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：

```
<script type="text/javascript">
  let client = new OSS({
    region: '<oss region>',
 //云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用
STS方式来进行API访问
    accessKeyId: '<Your accessKeyId(STS)>',
    accessKeySecret: '<Your accessKeySecret(STS)>',
    stsToken: '<Your securityToken(STS)>',
    bucket: '<Your bucket name>'
  });
```

```
</script>
```

- 使用*npm*安装SDK的开发包：

```
npm install ali-oss
```

成功安装完成之后，即可使用 import 或 require 进行引用。由于浏览器中原生不支持`require`这种模式的，因此需要在开发环境中配合相关的打包工具，比如`webpack`、`browserify`等。

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<oss region>',
 //云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用STS方式来进行API访问
    accessKeyId: '<Your accessKeyId(STS)>',
    accessKeySecret: '<Your accessKeySecret(STS)>',
    stsToken: '<Your securityToken(STS)>',
  bucket: '<Your bucket name>'
});
```

- 使用*Bower*：

对于Web项目，你也可以通过*Bower*安装SDK：

```
bower install ali-oss
```

然后在网页中添加`<script>`标签：

```
<script src="bower_components/ali-oss/dist/aliyun-oss-sdk.min.js
"></script>
```

12.2 快速开始

下面介绍如何在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。

> **注意:**
> 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，在客户端不要使用会泄露配置的AK（AccessKeyId 和 AccessKeySecret），所以请使用STS方式（临时账号权限）给客户端授权。

STS场景说明：**临时账号权限（*STS*）**。

更多有关浏览器的使用请参考：**浏览器应用**

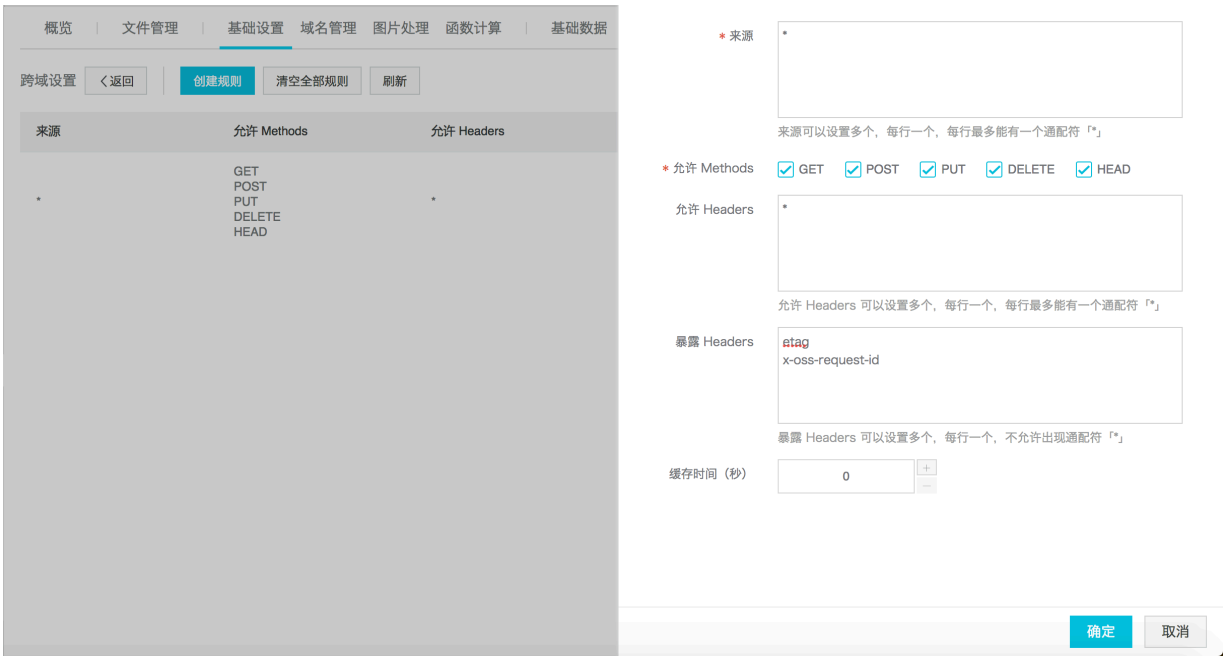
示例工程请参考：*example*

Bucket设置

从浏览器中直接访问OSS需要开通Bucket的*CORS*设置：

- **将allowed origins设置成** `*`
- **将allowed methods设置成** `PUT, GET, POST, DELETE, HEAD`
- **将allowed headers设置成** `*`
- **将expose headers设置成** `etag` `x-oss-request-id`

> **注意：**
> **请将该条CORS规则设置成所有CORS规则的第一条。**



使用SDK

目前浏览器中不能直接进行Bucket相关的操作，例如GetBucket (ListObjects)、GetBucketLogging、GetBucketReferer等。但是可以进行Object相关的操作，例如PutObject、GetObject等。

- **包含SDK**

  在html文件的`<head>`中包含如下标签：

```
<script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-x.x.x.min.
js"></script>
```

  想了解其他引入方式可参考安装介绍。

  通过`new OSS`来创建client，创建后返回的是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**

  1. **为账号开通STS服务，如果已开通可以跳过，开通STS请参见*STS开通说明*，具体OSS授权Policy配置参见*RAM和STS授权策略（Policy）配置***
  2. **借助STS的SDK或API应用专属的STS签名服务。为了方便大家快速理解，OSS提供了多个语言版本的Demo程序供参考：**
     - **Node.js: 下载地址**
     - **PHP: 下载地址**
     - **Java: 下载地址**
     - **Ruby: 下载地址**

     说明:
     Demo程序仅供STS功能实现参考。在具体的生产环境使用时，请自行开发鉴权等业务所需代码。
  3. **在前端（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)
   })
```

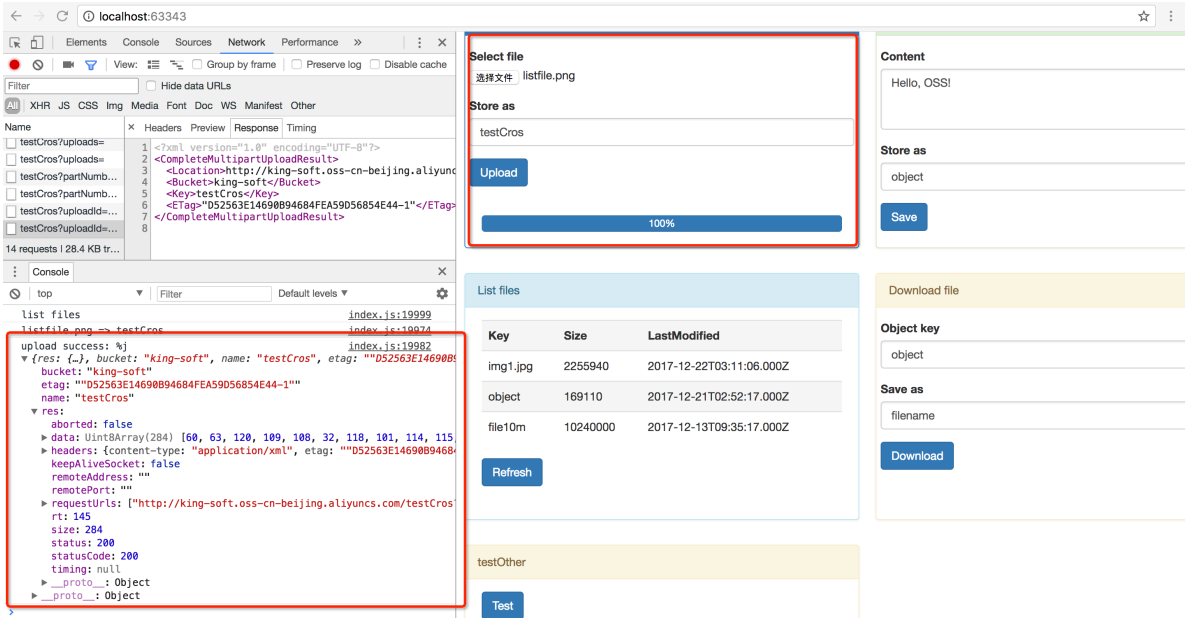
```
</script>
```

- **上传文件**

  下面使用`multipartUpload`接口来上传文件，`multipartUpload`的使用请参考*multipartUpload*。

```
<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,
   endpoint: '<oss endpoint>',
   bucket: '<Your bucket name>'
    });
          //storeAs表示上传的object name , file表示上传的文件
    client.multipartUpload(storeAs, file).then(function (result) {
   console.log(result);
    }).catch(function (err) {
   console.log(err);
    });
  });
 });
  </script>
```

```
</body>
```

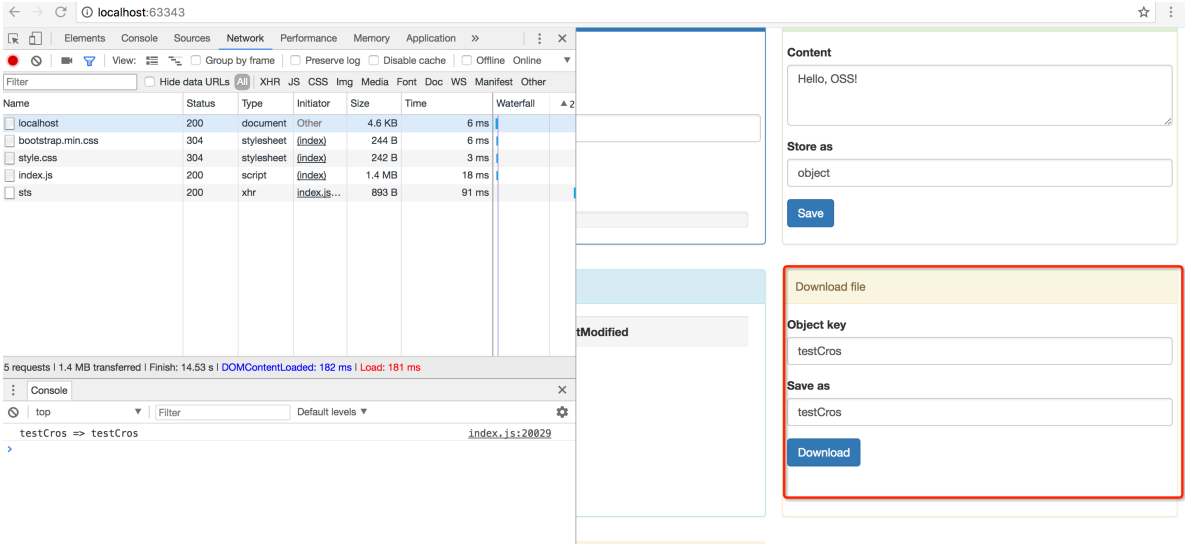


- **下载文件**

在浏览器中不能直接操作文件，因此采用签名URL的方式来生成文件的下载链接， 用户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，开发者完全可以使用任何发请求的库向`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,
   endpoint: '<oss endpoint>',
   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>
```



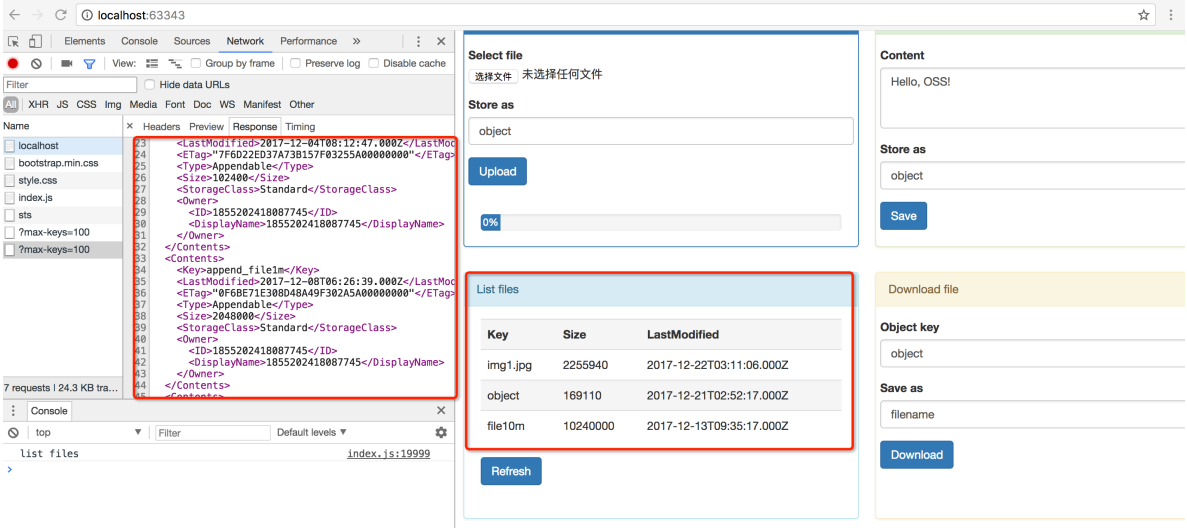
- **查看文件列表**

```
<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,
 endpoint: '<oss endpoint>',
 bucket: '<Your bucket name>'
  });

  client.list({
    'max-keys': 10
  }).then(function (result) {
    console.log(result);
  }).catch(function (err) {
    console.log(err);
  });
});
```

```
</script>
```

其中`region`参数是指您申请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。

在浏览器中打开html文件，然后打开Chrome的开发者控制台，即可查看列举文件的结果。



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鉴权模式和自签名模式，详细说明请参考访问控制、移动端直传。

12.3 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'
});
```

12.4 上传文件

您可以通过上传Blob数据或者断点续传上传的方式将文件上传到OSS。

上传Blob数据

Blob (Binary Large Object) 代表二进制类型的大对象。在一些数据库中会使用到Blob的概念。例如MySQL中的Blob类型表示二进制数据的容器。

用户也可以通过`put`接口简单地将Blob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 putBlob () {
  try {
    let result = await client.put('object-key', new Blob(['content'],
{ type: 'text/plain' }));
    console.log(result);
  } catch (e) {
    console.log(e);
  }
}
putBlob();
```

断点续传上传

当需要上传的文件较大时，可以通过*MultipartUpload*接口进行分片上传。分片上传的好处是将一个大请求分成多个小请求来执行。这样当其中一些分片请求失败后，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。一般对于大于100MB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，业务方需要自己处理超时逻辑。可以通过缩小分片大小、加大超时时间、重试请求或者业务上捕获`ConnectionTimeoutError`错误等方法处理超时，然后给用户提示。详情请参见**网络错误处理**。

相关参数说明如下：

- name {String} Object名称。
- file {File} HTML5网络文件或file {Blob} Blob的数据格式。

- [options] {Object} 额外参数 。
  - [checkpoint] {Object} 断点记录点。如果设置这个参数，上传会从断点开始；如果没有设置，则重新上传。
    - file {File} 用户选取的文件对象，浏览器重启后需要用户手动触发进行设置。
    - name {String} 上传的object key。
    - fileSize {Number} 文件大小。
    - partSize {Number} 分片大小。
    - uploadId {String} 分片上传的ID。
    - doneParts {Array} 已完成分片的数组，包含的对象结构如下：
      - number {Number} 分片的number。
      - etag {String} 分片的etag。
  - [parallel] {Number} 并发上传的分片个数。
  - [partSize] {Number} 分片大小。
  - [progress] {Function} `function`、 `async` 、`promise` 形式，回调函数包含三个参数：
    - percentage {Number} 进度百分比（0-1之间小数）。
    - checkpoint {Object} 断点记录点。
    - res {Object}) 单次part成功返回的response。
  - [meta] {Object} 用户自定义header meta信息，header前缀 `x-oss-meta-`。
  - [mime] {String} 用户自定义`Content-Type header` 。
  - [headers] {Object} http额外的头字段，详情请看 *RFC 2616* 。
    - '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' 指定回复的内容该以何种形式展示。是以内联的形式（即网页或者页面的一部分），还是以附件的形式下载并保存到本地，例如`Content-Disposition : somename`。
    - '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，例如`Expires: 3600000`。
  - [callback] {Object} callback回调设置。
    - url {String} 和oss server交互的回调服务器地址，对应CallBack参数中的callbackUrl。

- body {String} 发起回调时请求body的值，对应CallBack参数中的callbackBody。body值为JSON格式。
- [host] {String} 发起回调请求时Host头的值，对应CallBack参数中的callbackHost。
- [contentType] {String} 发起回调请求的Content-Type，对应CallBack参数中的callbackBodyType。
- [customValue] {Object} 发起回调请求自定义参数，对应CallBack参数中的callback-var。

以下代码用于断点续传上传：

```
let OSS = require('ali-oss')

let ossConfig = {
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: 'Your bucket name'
}

let client = new OSS(ossConfig);

let tempCheckpoint;

// 定义上传方法
async function multipartUpload () {
  try {
    let result = await client.multipartUpload('object-key', 'local-
file', {
      progress: async function (p, checkpoint) {
        // 断点记录点。 浏览器重启后无法直接继续上传，需用户手动触发进行设置。
        tempCheckpoint = checkpoint;
      }
      meta: { year: 2017, people: 'test' },
      mime: 'image/jpeg'
   })
  } catch(e){
    console.log(e);
  }
}

//开始上传
multipartUpload();

// 暂停分片上传方法
client.cancel();

// 恢复上传
let resumeclient = new OSS(ossConfig);
async function resumeUpload () {
  try {
    let result = await resumeclient.multipartUpload('object-key', '
local-file', {
    progress: async function (p, checkpoint) {
          tempCheckpoint = checkpoint;
        },
        checkpoint: tempCheckpoint
        meta: { year: 2017, people: 'test' },
```

```
        mime: 'image/jpeg'
  })
  } catch (e) {
    console.log(e);
  }
}

resumeUpload();
```

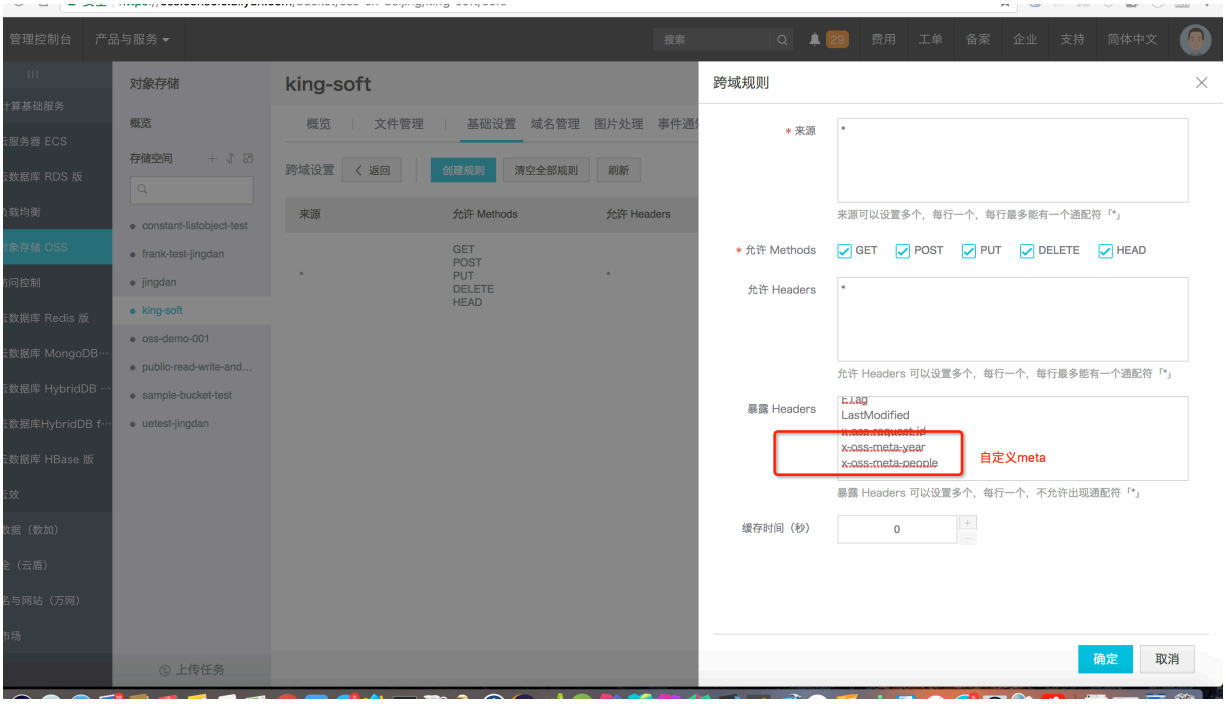
说明:

- **记录上传进度。再次上传时将之前记录的checkpoint参数传入即可。**
- **每次进行分片上传时建议用一个新的OSS实例。**

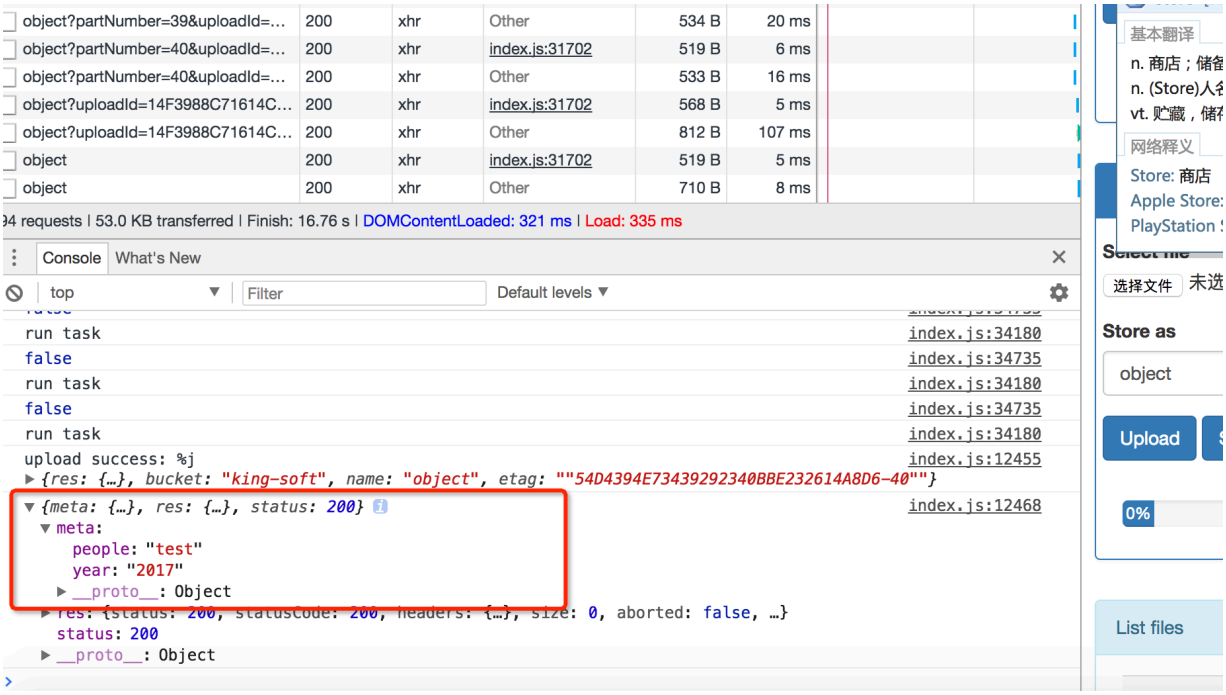
上述示例中的`progress`参数是一个进度回调函数，用于获取上传进度。

```
const progress = async function progress(p, checkpoint) {
  console.log(p)
};
```

`meta`参数是一个用户自定义的元数据，通过head接口可以获取到Object的meta数据。您需要在OSS控制台基础设置页签下的跨域设置中设置暴露header，如下图所示：



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',
  },
},
```

12.5 下载文件

本文介绍如何使用Browser.js下载对象存储OSS中的文件（Object）。

您可以通过HTTP下载的方式从OSS中下载文件。

HTTP下载，即浏览器下载。浏览器中使用`signatureUrl`方法生成可下载的HTTP地址，URL的有效时间默认为半个小时。

以下代码用于HTTP下载：

```
let OSS = require('ali-oss');

let client = new OSS({
  region: '<Your region>',
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>',
```

```
});

let url = client.signatureUrl('object-key');
console.log(url);

let url = client.signatureUrl('object-key', {expires: 3600});
console.log(url);
```

12.6 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();
```

12.7 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。

12.8 授权访问

本文介绍如何进行授权访问。

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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

以下代码用于使用STS凭证构造签名请求：

```
const express = require('express');
const { STS } = require('ali-oss');
const fs = require('fs');

const app = express();
const path = require('path');
const conf = require('./config');

app.get('/sts', (req, res) => {
  console.log(conf);
  let policy;
  if (conf.PolicyFile) {
    policy = fs.readFileSync(path.resolve(__dirname, conf.PolicyFile
)).toString('utf-8');
  }

  const client = new STS({
    accessKeyId: conf.AccessKeyId,
    accessKeySecret: conf.AccessKeySecret
  });

  client.assumeRole(conf.RoleArn, policy, conf.TokenExpireTime).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);
  });
```

```
});

app.use('/static', express.static('public'));
app.get('/', (req, res) => {
  res.sendFile(path.resolve(__dirname, '../index.html'));
});

app.listen(9000, () => {
  console.log('App started.');
});
```

使用签名 URL 进行临时授权

- **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，您可以指定 URL 的过期时间来限制访客的访问时长。

- **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，默认是 1800 秒。其他参数相关说明，请参考 *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);
```

12.9 访问权限

OSS允许您对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`。

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。

图片处理的详细信息请参见*OSS*图片处理指南。

图片处理功能

OSS图片处理提供以下功能：

- 获取图片信息
- 图片格式转换
- 图片缩放
- 图片裁剪
- 图片旋转
- 图片效果
- 图片水印，包括添加图片、文字、图文混合水印
- 自定义图片处理样式
- 级联处理，调用多个图片处理功能

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中设置并处理参数。

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

  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

参数	描述
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
私有读写（private）	默认权限（default）

Bucket权限	Object权限
任意权限	私有读写（private）

以下代码用于生成带签名的图片处理URL：

```
let OSS = require('ali-oss');
let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});
// 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`、 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` 访问图片、处理图片。

  说明：

  SDK处理图片文件支持 自定义样式、 `HTTPS`、 级联处理。

  - 基础操作

    以下代码展示了图片处理的基础操作：

```
let OSS = require('ali-oss');

let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 缩放
async function scale () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/resize,m_fixed,w_100,h_100'})
  } catch (e) {
    console.log(e);
  }
}

scale();
```

```
// 裁剪
async function crop () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/crop,w_100,h_100,x_100,y_100,r_1'})
  } catch (e) {
    console.log(e);
  }
}

crop();

// 旋转
async function rotate() {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/rotate,90'})
  } catch (e) {
    console.log(e);
  }
}

rotate();

// 锐化
async function sharpen () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/sharpen,100'})
  } catch (e) {
    console.log(e);
  }
}

sharpen();

// 水印
async function watermark () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/watermark,text_SGVsbG8g5Zu-54mH5pyN5YqhIQ
'})
  } catch (e) {
    console.log(e);
  }
}

watermark();

// 格式转换
async function format () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/format,png'})
  } catch (e) {
    console.log(e);
  }
}

format();

// 图片信息
async function info () {
  try {
```

```
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/info})
  } catch (e) {
    console.log(e)
  }
}

info();
```

- **自定义样式**

以下代码用于自定义图片样式：

```
let OSS = require('ali-oss');

let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 自定义样式
async function style () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'style/example-style'});
  } catch (e) {
    console.log(e);
  }
}

style();
```

说明：

使用以上示例自定义图片样式前，需在OSS控制台中添加自定义样式example-style。

- **级联处理**

以下代码用于级联处理图片：

```
let OSS = require('ali-oss');

let client = new OSS({
  accessKeyId: '<accessKeyId>',
  accessKeySecret: '<accessKeySecret>',
  bucket: '<bucketName>',
  endpoint: '<endpoint, 例如http://oss-cn-hangzhou.aliyuncs.com>'
});

// 级联处理
async function resize () {
  try {
    let result = await client.signatureUrl('example.jpg', {expires
: 3600, process: 'image/resize,m_fixed,w_100,h_100/rotate,90'})
  } catch (e) {
    console.log(e);
  }
```

```
}
resize();
```

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 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：

1. 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>
```

2. 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。

3. 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"></script>
  <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 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，详情请参考*RAM和STS介绍*。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*。

12.12 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'
```

12.13 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 AccessKeyId 和 AccessKeySecret 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: 参数是指您申请 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。具体步骤，请查看相关文档。

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)

参考浏览器分片上传

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，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-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，具体可参考*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], filename, { 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），可参考*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，可查看相关文档。

如何获取object的签名URL

可调用 `signatureUrl` 方法，获取下载地址，可查看相关文档。

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，用户直接使用这个url上传或者下载资源即可。利用签名URL上传资源请参考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

可参考*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: {


//这里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',
},
},
```

更多上传回调请参见原理介绍。

常见错误参考

- *SDK* **开启异常日志**

- *OSS* **常见错误**

12.14 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。**

13 Ruby

13.1 安装

本文介绍如何安装OSS Ruby SDK。

前提条件

- 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阿里云对象存储OSS服务，请查看对象存储OSS产品详情页。

- 创建AccessKey

  如果还没有创建AccessKey（AccessKeyId和AccessKeySecret），请到RAM访问控制管理控制台创建AccessKey。

安装

执行`gem`命令安装SDK。

```
gem install aliyun-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
```

或者通过bundler安装。

1. 在应用程序的`Gemfile`中添加`gem 'aliyun-sdk', '~> 0.3.0'`。
2. 选择淘宝镜像源进行安装。

```
# 可选的淘宝镜像源
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

  1. 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*。

  2. 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。

  3. 同时按下`Windows + R`，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，则用文本编辑器打开`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。

4. 输入以下命令`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，建议用户再安装一个开发使用的Ruby版本。

1. 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。

2. 在终端输入以下命令安装brew。

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

3. 在终端输入以下命令安装Ruby。

```
brew install ruby
exec $SHELL -l
```

4. 在终端输入以下命令安装SDK。

```
gem install aliyun-sdk
```

5. 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
=> ...
```

上传一个文件

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
=> ...
```

下载一个文件

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。

删除一个文件

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**
- **设置访问权限**
- **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

存储空间（Bucket）是OSS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

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Bucket名字为特定前缀的所有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 }
```

创建Bucket

使用`Client#create_bucket`接口创建一个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.create_bucket('my-bucket')
```

说明：

- **Bucket的命名规范请查看*OSS* 基本概念。**
- **由于存储空间的名字是全局唯一的，所以必须保证您的Bucket名字不与别人的重复。**
- **获取endpoint信息，请参见访问域名和数据中心文档。**

删除Bucket

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')
```

说明:

- **如果该Bucket下还有文件存在，则需要先删除所有文件才能删除Bucket。**
- **如果该Bucket下还有未完成的上传请求，则需要通过`list_uploads`和`abort_upload`先取消那些请求才能删除Bucket。用法请参考*API文档*。**

查看Bucket是否存在

用户可以通过`Client#bucket_exists?`接口查看当前用户的某个Bucket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')
```

Bucket访问权限

用户可以设置Bucket的访问权限，允许或者禁止匿名用户对其内容进行读写。更多关于访问权限的内容请参考访问权限。

- **获取Bucket的访问权限（ACL）**

  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
  ```

- **设置Bucket的访问权限（ACL）**

  通过`Bucket#acl=`设置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')
  bucket.acl = Aliyun::OSS::ACL::PUBLIC_READ
  ```

```
puts bucket.acl
```

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请参考*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`wgetcurl`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。

- `expires`URL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下的所有文件：

```
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，称为“元信息”。 这些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_object`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_object(objs)
puts result #['my-object-1', 'my-object-2']，默认返回删除成功的文件

objs = ['my-object-3', 'my-object-4']
result = bucket.batch_delete_object(objs, :quiet => true)
puts result #[]，不返回删除的结果
```

13.7 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：

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。

13.8 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。STS更详细的解释请参见*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的流程请参见开发指南中的*#unique_100*。

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')
```

向 STS 申请临时 token 时，还可以指定自定义的 STS Policy。这样申请的临时权限是所扮演角色的权限与Policy指定的权限的交集。下面的例子将通过指定 STS Policy 申请对`my-bucket`的只读权限，并指定临时 token 的过期时间为 15 分钟。

```
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。

13.9 访问权限

OSS允许对Bucket和Object分别设置访问权限，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。

Bucket访问权限

对于Bucket，有三种访问权限：

- Public-read-write：允许匿名用户对该存储空间内文件进行读写操作。
- Public-read：允许匿名用户对该存储空间内文件进行读操作。

- **Private：不允许匿名访问，所有的访问都要经过签名。**

创建Bucket时，默认是private权限。之后您可以通过`puts bucket.acl`来设置Bucket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')
puts bucket.acl
```

Object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
```

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（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`。
- 访问具有public-read或者public-read-write权限的Bucket或Object时，也可以通过创建匿名的Client来进行：

有

```
require 'aliyun/oss'

# 不填access_key_id和access_key_secret，将创建匿名Client，只能访问具
# public权限的Bucket/Object
client = Aliyun::OSS::Client.new(endpoint: 'endpoint')
bucket = client.get_bucket('my-bucket')

bucket.get_object('my-object', :file => 'local_file')
```

13.10 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。

每条规则包含：

- 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。
- 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。
- 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。
- 是否生效。

更多关于生命周期的内容请参考文件生命周期。

设置生命周期规则

通过`Bucket#lifecycle=`来设置生命周期规则：

```
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

通过`Bucket#lifecycle`来查看生命周期规则：

```
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

通过`Bucket#lifecycle=`设置一个空的Rule数组来清空生命周期规则：

```
require 'aliyun/oss'
client = Aliyun::OSS::Client.new(
  endpoint: 'endpoint',
  access_key_id: 'AccessKeyId', access_key_secret: 'AccessKeySecret')
bucket.lifecycle = []
```

13.11 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)
```

13.12 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)
```

13.13 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: [])
```

13.14 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跨域访问。

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 = []
```

13.15 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。

13.16 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

a. 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。**

b. 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

a. 编辑`oss-manager/Gemfile`，并加入SDK的依赖。

```
gem 'aliyun-sdk', '~> 0.3.0'
```

b. 在`oss-manager/`中执行`bundle install`。

c. 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、AccessKeyId和AccessKeySecret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 一些辅助函数
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。

说明:

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。详情请参考管理文件。

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：

a. 总是在第一个显示'../'指向上级目录

b. 对于Common prefix，显示为目录

c. 对于Object，显示为文件

上面的代码中用到了`with_prefixremove_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，详情请参考 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。**
- **第二种上传方式是用户使用服务器生成的表单和临时凭证将文件直接上传到OSS。**

1. **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
```

2. **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提供的一个方便用户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。用户需要将其拷贝到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。

3. 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代码请参看 *OSS Rails Demo*。

14 Media-C

14.1 前言

OSS Media-C SDK构建于OSS C SDK版本之上，本文主要介绍OSS Media-C SDK 2.0.1版本的安装和使用。

背景信息

将摄像头拍摄的视频快速存储到阿里云对象存储OSS时，您需要考虑以下几方面的因素：

- **设备上不能永久存储AccessKeyId和AccessKeySecret。**
- **设备上只允许上传文件、下载文件，但不允许删除或修改配置。**
- **可以提供网页让用户去管理自己的视频。**
- **对设备的权限精准控制。**
- **对设备的权限存在有效期，不能让设备永久持有某种权限。**
- **可以直接通过HLS协议直接查看摄像机输出的音视频。**

针对以上考虑，阿里云推出的OSS Media-C SDK可以方便地解决上述问题，为音视频行业提供更完善易用的解决方案。

前提条件

- **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阿里云对象存储OSS服务，请查看对象存储OSS产品详情页。
- **创建AccessKey**

  如果还没有创建AccessKey（AccessKeyId和AccessKeySecret），请到RAM访问控制管理控制台创建AccessKey。

下载SDK

- **Linux V2.0.2**
  - **SDK包：** *aliyun-media-c-sdk-2.0.2.tar.gz*
  - **源代码：** *GitHub*
  - **版本迭代：** *GitHub*

> **说明：**
> **OSS Media-C SDK不支持Windows。**

- **兼容性**

  对于1.x.x 系列的Media-C SDK，除C SDK中list相关接口发生变化后不兼容以外，其他接口均兼容。

14.2 安装

本文介绍如何安装 Media-C SDK。

版本依赖

- **Linux**

  OSS C SDK = 3.x.x

- **Windows**

  不支持

Linux环境安装

- **安装OSS C SDK，安装步骤请参考安装*C SDK* 。**
- **单击此处下载SDK包或者下载源代码，包括src、sample、test三个目录和CMakeList.txt文件。**
- **安装到系统目录**
  - **如果OSS C SDK和其依赖都是安装在系统目录下（/usr/local/或/usr/），且希望OSS Media-C SDK也安装到系统目录下，执行下列命令编译安装：**

    ```
    cmake .
    make
    make install
    ```

  - **上面命令执行成功后，OSS Media-C SDK会自动安装到/usr/local/下面。**
- **安装到自定义目录（依赖包安装到系统目录）**
  - **如果OSS C SDK及其依赖都是安装到系统目录下（/usr/local/或/usr/），但希望将OSS Media-C SDK安装到自定义目录，例如/home/user/aliyun/oss/install/，执行下列命令编译安装：**

    ```
    cmake . -DCMAKE_INSTALL_PREFIX=/home/user/aliyun/oss/install/
usr/local/
    make
    make install
    ```

  - **上面命令执行成功后，OSS Media-C SDK会自动安装到/home/user/aliyun/oss/install/usr/local/下面。**

- 安装到自定义目录（依赖包安装在自定义目录）
  - 如果OSS C SDK或某些依赖包安装到了自定义目录，此时编译OSS Media-C SDK时默认是找不到这些包的头文件和库文件，所以需要在执行cmake时指定路径，例如已经将OSS C SDK安装到了/home/user/aliyun/oss/install/目录，则执行下列命令编译安装：

```
    cmake . -DCMAKE_INSTALL_PREFIX=/home/user/aliyun/oss/install/
usr/local/ -DOSS_C_SDK_INCLUDE_DIR=/home/user/aliyun/oss/install
/usr/local/include/ -DOSS_C_SDK_LIBRARY=/home/user/aliyun/oss/
install/usr/local/lib/liboss_c_sdk.so
    make
    make install
```

  - 上面命令执行成功后，OSS Media-C SDK会自动安装到/home/user/aliyun/oss/install/usr/local/下面。
  - 其他依赖包相关参数名称：APR_UTIL_LIBRARY，APR_LIBRARY，CURL_LIBRARY，APR_INCLUDEDIRS，APU_INCLUDEDIRS，OSS_C_SDK_INCLUDE_DIR，CURL_INCLUDEDIRS等。
- 仅编译安装客户端SDK
  - 默认是同时编译安装客户端和服务端的sdk的，如果仅需要编译安装客户端的SDK，则执行下列命令编译安装。

```
    cmake . -DONLY_BUILD_CLIENT=ON
    make
    make install
```

  - 如果仅需要编译安装服务端，将ONLY_BUILD_CLIENT修改为ONLY_BUILD_SERVER即可。
  - 只有同时编译客户端和服务端时才会编译测试用例。
- 其他编译安装方式和问题
  - 编译模式：目前支持四种，分别是Debug，Release，MinSizeRef，RelWithDebInfo，指定使用某种编译类型，使用参数-DCMAKE_BUILD_TYPE，例如使用Debug模式编译，则在cmake后增加参

数-DCMAKE_BUILD_TYPE=Debug：cmake . -DCMAKE_BUILD_TYPE=Debug，默认是Release模式。

- Debug：没有做任何代码优化，支持gdb，一般用来调试程序。
- Release：使用了更高级别的优化，一般适用于生产环境。
- MinSizeRef：生成最小的库文件，一般用于嵌入式环境。
- RelWithDebInfo：使用了更高级的优化，但附带了调试信息，一般也用于生产环境。

- 执行cmake时出现“Targets may link only to libraries. CMake is dropping the item”的warnning，原因是指定的library路径不对，library路径应该指定到*.so，例如/path/to/xxx.so。
- 如果需要使用OSS C SDK的静态库，则在执行cmake时指定-DOSS_C_SDK_LIBRARY=/home/user/aliyun/oss/install/usr/local/lib/liboss_c_sdk_static.a即可。其他库类似。
- 执行cmake时出现“CMake Error: The following variables are used in this project, but they are set to NOTFOUND.”，原因是相应的库无法从默认路径中找到，需要用户指定，参考<安装到自定义目录>。

14.3 初始化

初始化 Media-C SDK环境。

确定Endpoint

Endpoint是阿里云OSS服务在各个区域的域名地址，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参考这里
用户自定义域名	用户自定义域名，且CNAME指向OSS域名

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，可以参考：点击查看。
- 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。

配置密钥

要接入阿里云OSS，您需要拥有一个有效的 Access Key（包括AccessKeyId和AccessKeySecret）用来进行签名认证。可以通过如下步骤获得：

- 注册阿里云账号
- 申请*AccessKey*

使用RAM和STS服务

如果您的产品需要从设备端（例如手机、平板电脑、摄像机等）上传、下载文件，设备端应该只获取到临时授权，而非永久授权。这时候，需要用到RAM和STS服务。

1. 前往*RAM*，开通RAM服务。
2. 创建一个角色，可以给这个角色授予AliyunOSSFullAccess和AliyunSTSAssumeRoleAccess等权限。
3. 创建成功后，在角色详情里面有一个Arn，类似于acs:ram:xxxx:role/yyyy，这个就是role_arn，在后续获取临时token时需要使用。

初始化

- 客户端的初始化参考：客户端操作
- 服务端的初始化参考：服务端操作

14.4 客户端

OSS Media-C SDK分为客户端、服务端和HLS三部分。本文主要介绍客户端的相关操作。

接口

客户端相关的操作接口都位于oss_media_file.h中，目前提供以下接口：

- oss_media_file_open
- oss_media_file_stat
- oss_media_file_tell
- oss_media_file_seek
- oss_media_file_read
- oss_media_file_write
- oss_media_file_close

下面会详细介绍各个接口的功能和注意事项。

基础结构体介绍

```
/**
 * OSS MEDIA FILE的元数据，包括文件长度，位置和类型
 */
typedef struct {
    int64_t length;
    int64_t pos;
    char    *type;
} oss_media_file_stat_t;


/**
 * OSS MEDIA FILE的属性信息
 */
typedef struct oss_media_file_s {
    void   *ipc;
    char   *endpoint;
    int8_t is_cname;
    char   *bucket_name;
    char   *object_key;
    char   *access_key_id;
    char   *access_key_secret;
    char   *token;
    char   *mode;
    oss_media_file_stat_t _stat;

    time_t expiration;
    auth_fn_t auth_func;
} oss_media_file_t;
```

说明:

- **type：文件类型，Normal或者Appendable。**
- **ipc：用于设置设备的唯一标识，根据用户的需要可以在auth func中使用，如果用不到则可以忽略。**
- **endpoint：比如oss-cn-hangzhou.aliyuncs.com。**
- **is_cname：是否使用了CNAME。**
- **bucket_name：OSS上存储空间的名称。**
- **object_key：OSS上文件的名称。**
- **access_key_id：阿里云提供的访问控制的access key id，有以下两种使用方式。**
  - **使用主账号或者子账号的永久access key id，此时后面的token需要设置为NULL。**
  - **使用通过get_token获取的临时access key id。**
- **access_key_secret：阿里云提供的访问控制的access key secret，有以下两种使用方式。**
  - **使用主账号或者子账号的永久access key secret，此时后面的token需要设置为NULL。**
  - **使用通过get_token获取到的临时access key secret。access_key_id和access_key_secret必须同时使用同种方式。**

- token：
  - 如果使用了主账号或子账号的永久access key id和access key secret，这里需要设置为NULL。
  - 如果是终端设备上传、下载资源，可以通过服务端的get_token获取临时的access_key_id、access_key_secret和token。
  - 如果设置token不为NULL，则用户使用的是临时access key id、临时access key secrety和临时token。
- expiration：授权失效的时间，超出失效时间后才会再次授权。
- auth：授权函数，用户需要实现的一个函数，且在这个函数内为host、bucket、token等赋值。
- mode：读写模式。

初始化

```
/**
 *  @brief   初始化oss media
 *  @note    在程序开始的时候应该首先调用此接口，初始化OSS MEDIA C SDK
 *  @return:
 *       返回0时表示成功
 *       否则，表示出现了错误，可能导致失败的原因包括：内存不足，apr、curl版本太低等
 */
int oss_media_init(aos_log_level_e log_level);
```

说明:

示例代码请参考*GitHub* 。

销毁

```
/**
 *  @brief   销毁oss meida
 *  @note    在程序结束的时候应该最后调用此接口，销毁OSS MEDIA C SDK
 */
void oss_media_destroy();
```

说明:

示例代码请参考*GitHub*。

打开文件

```
/**
 *  @brief   打开一个oss文件
 *  @param[in]  bucket_name oss上存储文件的存储空间名称
 *  @param[in]  object_key  oss上的文件名称
 *  @param[in]  mode:
```

```
 *       'r': 读模式
 *       'w': 覆盖写模式
 *       'a': 追加写模式
 *      notes: 不允许组合使用
 *  @param[in]  auth_func    授权函数，设置access_key_id/access_key
_secret等
 *  @return:
 *      返回非NULL时成功，否则失败
 */
oss_media_file_t* oss_media_file_open(char *bucket_name,
                                      char *object_key,
                                      char *mode,
                                      auth_fn_t auth_func);
```

说明:

- **mode支持只读、覆盖写、追加写三种模式，不支持组合模式。**
  - **只读模式：打开文件后，一次读取部分文件内容，完整内容一般需要多次读取。**
  - **覆盖写：打开文件后，一次写入完整的文件内容，可以多次写入，且最后一次写入的内容会覆盖前面写入的内容。**
  - **追加写：打开文件后，可以多次追加写文件。**
- **示例代码请参考***GitHub*

关闭文件

```
/**
 *  @brief  关闭oss文件
 */
void oss_media_file_close(oss_media_file_t *file);
```

说明:

示例代码请参考*GitHub*。

写文件

```
/**
 *  @brief  写文件到oss上
 *  @return:
 *      成功时返回写入的数据大小，返回-1表示写失败
 */
int64_t oss_media_file_write(oss_media_file_t *file, const void *buf,
int64_t nbyte);
```

示例程序:

```
/* 授权函数 */
static void auth_func(oss_media_file_t *file) {
    file->endpoint = "your endpoint";
    file->is_cname = 0;
```

```
    file->access_key_id = "阿里云提供的access key id或者临时access key id
";
    file->access_key_secret = "阿里云提供的access key secret或者临时
access key secret";
    file->token = "通过get_token接口获取到得临时token";

    /* 本次授权的有效时间 */
    file->expiration = time(NULL) + 300;
}

static void write_file() {
    oss_clean(g_filename);

    int64_t write_size;
    oss_media_file_t *file;
    char *bucket_name = "<your bucket name>";
    char *key = "<your object key>";
    char *content = "aliyun oss media c sdk";

    /* 打开一个文件 */
    file = oss_media_file_open(bucket_name, key, "w", auth_func);
    if (!file) {
        printf("open media file failed\n");
        return;
    }

    /* 写文件 */
    write_size = oss_media_file_write(file, content, strlen(content));
    if (-1 != write_size) {
         printf("write %" PRId64 " bytes succeeded\n", write_size);
    } else {
        oss_media_file_close(file);
        printf("write failed\n");
        return;
    }

    /* 关闭文件，释放资源 */
    oss_media_file_close(file);
}
```

说明:

- 上述示例中，打开文件的模式为覆盖写("w")，如果多次写入，后一次写入的内容会覆盖前一次写入的内容。如果需要追加写，则指定打开文件的模式为追加写("a")。
- 示例代码请参考*GitHub*。

读文件

```
/**
 *  @brief   读取固定数目nbyte的数据
 *  @note    buf的大小应该大于等于nbyte + 1
 *  @return:
 *       返回0时表示成功
 *       否则，返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文
件，无法连接OSS，无权限读OSS等
 */
int64_t oss_media_file_read(oss_media_file_t *file, void *buf, int64_t
 nbyte);
```

示例程序：

```
    int ntotal, nread, nbuf = 16;
    char buf[nbuf];
    char *content;
    char *bucket_name = "<your bucket name>";
    char *key = "<your object key>";

    oss_media_file_t *file;

    /* 打开文件 */
    file = oss_media_file_open(bucket_name, key, "r", auth_func);
    if (!file) {
        printf("open media file failed\n");
        return;
    }

    /* 读文件 */
    content = malloc(stat.length + 1);
    ntotal = 0;
    while ((nread = oss_media_file_read(file, buf, nbuf)) > 0) {
        memcpy(content + ntotal, buf, nread);
        ntotal += nread;
    }
    content[ntotal] = '\0';

    /* 关闭文件 */
    oss_media_file_close(file);
    free(content);

    printf("oss media c sdk read object succeeded\n");
}
```

说明：

示例代码请参考*GitHub*。

管理文件

OSS Media File也支持tell、seek和stat操作。

```
/**
 *  @brief  获取当前OSS MEDIA FILE的位置
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文件，无法连接OSS，无权限读OSS等
 */
int64_t oss_media_file_tell(oss_media_file_t *file);

/**
 *  @brief  设置OSS MEDIA FILE的指针到指定位置
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：不是以读模式打开的文件，无法连接OSS，无权限读OSS等
 */
```

```
int64_t oss_media_file_seek(oss_media_file_t *file, int64_t offset);

/**
 *  @brief 获取OSS MEDIA FILE的元数据
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
读OSS等
 */
int oss_media_file_stat(oss_media_file_t *file, oss_media_file_stat_t
 *stat);
```

示例程序：

```
static void seek_tell_stat_file() {
    int ntotal, nread, nbuf = 16;
    char buf[nbuf];
    char *bucket_name = "<your bucket name>";
    char *key = "<your object key>";

    oss_media_file_t *file;

    /* 打开文件 */
    file = oss_media_file_open(bucket_name, key, "r", auth_func);
    if (!file) {
        printf("open media file failed\n");
        return;
    }

    /* 获取文件meta信息 */
    oss_media_file_stat_t stat;
    if (0 != oss_media_file_stat(file, &stat)) {
        oss_media_file_close(file);
        oss_media_file_free(file);

        printf("stat media file[%s] failed\n", file->object_key);
        return;
    }

    printf("file [name=%s, length=%ld, type=%s]",
           file->object_key, stat.length, stat.type);

    /* tell */
    printf("file [position=%" PRId64 "]", oss_media_file_tell(file));

    /* seek */
    oss_media_file_seek(file, stat.length / 2);

    /* 关闭文件 */
    oss_media_file_close(file);

    printf("oss media c sdk seek object succeeded\n");
}
```

说明：

示例代码请参考*GitHub*。

14.5 服务端

OSS Media-C SDK分为客户端，服务端和HLS三部分。本文主要介绍服务端的相关操作。

接口

服务端相关的操作接口位于oss_media.h中，目前提供的接口有：

- **oss_media_create_bucket**
- **oss_media_delete_bucket**
- **oss_media_create_bucket_lifecycle**
- **oss_media_get_bucket_lifecycle**
- **oss_media_delete_bucket_lifecycle**
- **oss_media_delete_file**
- **oss_media_list_files**
- **oss_media_get_token**
- **oss_media_get_token_from_policy**

下面会详细介绍各个接口的功能和注意事项。

基础结构体介绍

```
typedef struct oss_media_config_s {
    char *endpoint;
    int is_cname;
    char *access_key_id;
    char *access_key_secret;
    char *role_arn;
} oss_media_config_t;

typedef struct oss_media_files_s {
    char *path;
    char *marker;
    int  max_size;

    char *next_marker;
    int  size;
    char **file_names;
} oss_media_files_t;
```

说明:

- **endpoint：比如oss-cn-hangzhou.aliyuncs.com。**
- **is_cname：是否使用了CNAME。**
- **access_key_id：阿里云提供的访问控制的access key id。**

- access_key_secret：阿里云提供的访问控制的access key secret。
- role_arn：阿里云访问控制中创建的角色的Arn，具体位置在：访问控制RAM控制台 -> 角色管理 -> 点击相应角色名称 -> 基本信息 -> Arn，格式类似于acs:ram::xxxxxx:role/yyyy。如果还没有角色，需要创建一个新的角色，并赋予AliyunOSSFullAccess和AliyunSTSAssumeRoleAccess等权限。
- marker：设定结果从marker之后按字母排序的第一个开始返回。
- max_size：设定返回的最大数，取值不能大于1000。
- next_marker：下次执行时开始的位置。

初始化

```
/**
 *  @brief   初始化oss media
 *  @note    在程序开始的时候应该首先调用此接口，初始化OSS MEDIA C SDK
 *  @return:
 *       返回0时表示成功
 *       否则，表示出现了错误，可能导致失败的原因包括：内存不足，apr、curl版本太低等
 */
int oss_media_init();
```

说明:

示例代码参考：*GitHub*。

销毁

```
/**
 *  @brief   销毁oss meida
 *  @note    在程序结束的时候应该最后调用此接口，销毁OSS MEDIA C SDK
 */
void oss_media_destroy();
```

说明:

示例代码参考：*GitHub*。

创建存储空间

```
/**
 *  @brief   创建新的存储空间
 *  @paramp[in] oss_media_acl_t
 *      OSS_ACL_PRIVATE    私有读写
 *      OSS_ACL_PUBLIC_READ   公共读，私有写
 *      OSS_ACL_PUBLIC_READ_WRITE 公共读写
 *  @return:
 *       返回0时表示成功
 *       否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限等
 */
```

```
int oss_media_create_bucket(oss_media_config_t *config, const char *
bucket_name, oss_media_acl_t acl);
```

示例程序：

```
static void init_media_config(oss_media_config_t *config) {
    config->endpoint = "your endpoint";
    config->access_key_id = "阿里云提供的access key id";
    config->access_key_secret = "阿里云提供的access key secret";
    config->role_arn = "阿里云访问控制RAM提供的role arn";
    config->is_cname = 0;
}

void create_bucket() {
    int ret;
    char *bucket_name;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 创建存储空间 */
    ret = oss_media_create_bucket(&config, bucket_name, OSS_ACL_PR
IVATE);

    if (0 == ret) {
        printf("create bucket[%s] succeeded.\n", bucket_name);
    } else {
        printf("create bucket[%s] failed.\n", bucket_name);
    }
}
```

说明：

示例代码参考： *GitHub*。

删除存储空间

```
/*
 *  @brief   删除存储空间
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
等
 */
int oss_media_delete_bucket(oss_media_config_t *config, const char *
bucket_name);
```

示例程序：

```
void delete_bucket() {
    int ret;
    char *bucket_name;
```

```
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除存储空间 */
    ret = oss_media_delete_bucket(&config, bucket_name);

    if (0 == ret) {
        printf("delete bucket[%s] succeeded.\n", bucket_name);
    } else {
        printf("delete bucket[%s] failed.\n", bucket_name);
    }
}
```

说明:

示例代码参考： *GitHub*。

创建存储生命周期规则

```
/**
 *  @brief   创建存储空间生命周期规则
 *  @note    这些规则可以控制文件的自动删除时间
 *  @return:
 *       返回0时表示成功
 *       否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
等
 */
int oss_media_create_bucket_lifecycle(oss_media_config_t *config,
const char *bucket_name, oss_media_lifecycle_rules_t *rules);
```

示例程序：

```
void create_bucket_lifecycle() {
    int ret;
    char *bucket_name;
    oss_media_lifecycle_rules_t *rules;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 创建生命周期规则 */
    rules = oss_media_create_lifecycle_rules(2);
    oss_media_lifecycle_rule_t rule1;
    rule1.name = "example-1";
    rule1.path = "/example/1";
    rule1.status = "Enabled";
    rule1.days = 1;
    oss_media_lifecycle_rule_t rule2;
    rule2.name = "example-2";
    rule2.path = "/example/2";
    rule2.status = "Disabled";
    rule2.days = 2;
```

```
    rules->rules[0] = &rule1;
    rules->rules[1] = &rule2;

    /* 设置存储空间的生命周期规则 */
    ret = oss_media_create_bucket_lifecycle(&config,
            bucket_name, rules);

    if (0 == ret) {
        printf("create bucket[%s] lifecycle succeeded.\n", bucket_name
);
    } else {
        printf("create bucket[%s] lifecycle failed.\n", bucket_name);
    }

    /* 释放资源 */
    oss_media_free_lifecycle_rules(rules);
}
```

说明:

示例代码参考： *GitHub*。

获取存储空间的生命周期规则

```
/**
 *  @brief   获取存储空间的生命周期规则
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS等
 */
int oss_media_get_bucket_lifecycle(oss_media_config_t *config, const
char *bucket_name, oss_media_lifecycle_rules_t *rules);
```

示例程序：

```
void get_bucket_lifecycle() {
    int ret, i;
    char *bucket_name;
    oss_media_lifecycle_rules_t *rules;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 获取生命周期规则 */
    rules = oss_media_create_lifecycle_rules(0);
    ret = oss_media_get_bucket_lifecycle(&config, bucket_name, rules);

    if (0 == ret) {
        printf("get bucket[%s] lifecycle succeeded.\n", bucket_name);
    } else {
        printf("get bucket[%s] lifecycle failed.\n", bucket_name);
    }

    for (i = 0; i < rules->size; i++) {
        printf(">>>> rule: [name:%s, path:%s, status=%s, days=%d]\n",
               rules->rules[i]->name, rules->rules[i]->path,
```

```
                rules->rules[i]->status, rules->rules[i]->days);
    }

    /* 释放资源 */
    oss_media_free_lifecycle_rules(rules);
}
```

说明:

示例代码参考： *GitHub*。

删除存储空间的生命周期规则

```
/**
 *  @brief   删除存储空间的生命周期规则
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
等
 */
int oss_media_delete_bucket_lifecycle(oss_media_config_t *config,
const char *bucket_name);
```

示例程序：

```
void delete_bucket_lifecycle()
{
    int ret;
    char *bucket_name;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除生命周期规则 */
    ret = oss_media_delete_bucket_lifecycle(&config, bucket_name);

    if (0 == ret) {
        printf("delete bucket[%s] lifecycle succeeded.\n", bucket_name
);
    } else {
        printf("delete bucket[%s] lifecycle failed.\n", bucket_name);
    }
}
```

说明:

示例代码参考： *GitHub* 。

删除文件

```
/**
 *  @brief   删除存储空间中特定的文件
 *  @return:
```

```
 *       返回0时表示成功
 *       否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
等
 */
int oss_media_delete_file(oss_media_config_t *config, const char *
bucket_name, const char *key);
```

示例程序：

```
void delete_file() {
    int ret;
    oss_media_config_t config;
    char *file;
    char *bucket_name;

    /* 初始化变量 */
    file = "oss_media_file";
    bucket_name = "<your bucket name>";
    init_media_config(&config);

    /* 删除文件 */
    ret = oss_media_delete_file(&config, bucket_name, file);

    if (0 == ret) {
        printf("delete file[%s] succeeded.\n", file);
    } else {
        printf("delete file[%s] lifecycle failed.\n", file);
    }
}
```

说明：

示例代码参考： *GitHub*。

列出文件

```
/**
 *  @brief   列出特定存储空间内的文件
 *  @return:
 *       返回0时表示成功
 *       否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接OSS，无权限
等
 */
int oss_media_list_files(oss_media_config_t *config, const char *
bucket_name, oss_media_files_t *files);
```

示例程序：

```
void list_files() {
    int ret, i;
    char *bucket_name;
    oss_media_files_t *files;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>>";
```

```
    init_media_config(&config);

    files = oss_media_create_files();
    files->max_size = 50;

    /* 列举文件 */
    ret = oss_media_list_files(&config, bucket_name, files);

    if (0 == ret) {
        printf("list files succeeded.\n");
    } else {
        printf("list files lifecycle failed.\n");
    }

    for (i = 0; i < files->size; i++) {
        printf(">>>>file name: %s\n", files->file_names[i]);
    }

    /* 释放资源 */
    oss_media_free_files(files);
}
```

说明:

示例代码参考： *GitHub* 。

获取临时token

```
/*
 *  @brief   获取临时token
 *  @param[in]:
      mode:
 *      'r': 读模式
 *      'w': 覆盖写模式
 *      'a': 追加写模式
 *    expiration: 临时token的有效时间，最小15分钟，最长1小时，单位秒
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接阿里云STS服务，没有开通RAM、STS，没有创建Role，参数不合法等
 */
int oss_media_get_token(oss_media_config_t *config,
                        const char *bucket_name,
                        const char *path,
                        const char *mode,
                        int64_t expiration,
                        oss_media_token_t *token);
```

示例程序：

```
void get_token() {
    int ret;
    char *bucket_name;
    oss_media_token_t token;
    oss_media_config_t config;

    /* 初始化变量 */
    bucket_name = "<your bucket name>";
```

```
    init_media_config(&config);

    /* 获取临时token */
    ret = oss_media_get_token(&config, bucket_name, "/*", "rwa",
                              3600, &token);

    if (0 == ret) {
        printf("get token succeeded, access_key_id=%s, access_key
_secret=%s, token=%s\n",
                token.tmpAccessKeyId, token.tmpAccessKeySecret, token.
securityToken);
    } else {
        printf("get token failed.\n");
    }
}
```

说明:

示例代码参考： *GitHub*。

通过自定义policy获取token

```
/**
 *  @brief   通过指定自定义的policy获取token
 *  @param[in]:
 *      expiration:  临时token的有效时间，最小15分钟，最长1小时，单位秒
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误，可能导致失败的原因包括：无法连接阿里云STS服
务，没有开通RAM、STS，没有创建Role，参数不合法等
 */
int oss_media_get_token_from_policy(oss_media_config_t *config,
                                    const char *policy,
                                    int64_t expiration,
                                    oss_media_token_t *token);
```

示例程序：

```
void get_token_from_policy() {
    int ret;
    oss_media_token_t token;
    char *policy;
    oss_media_config_t config;

    /* 初始化变量 */
    init_media_config(&config);

    /* 设置自定义policy */
    policy = "{\"Version\":\"1\",\"Statement\":[{\"Effect\":\"Allow\",
 "
             "\"Action\":\"*\", \"Resource\":\"*\"}]}";

    /* 获取token */
    ret = oss_media_get_token_from_policy(&config, policy, 3600, &
token);

    if (0 == ret) {
```

```
        printf("get token succeeded, access_key_id=%s, access_key
_secret=%s, token=%s\n",
               token.tmpAccessKeyId, token.tmpAccessKeySecret, token.
securityToken);
    } else {
        printf("get token failed.\n");
    }
}
```

说明:

示例代码参考： *GitHub*。

14.6 HLS基础接口

OSS MEDIA C SDK 客户端部分支持将接收到的H.264、AAC格式封装为TS、M3U8格式，然后写到OSS上，用户通过对应的m3u8地址就可以欣赏视频音频了。

接口

HLS相关基础接口都位于oss_media_hls.h中，目前提供的接口有：

- **oss_media_hls_open**
- **oss_media_hls_write_frame**
- **oss_media_hls_begin_m3u8**
- **oss_media_hls_write_m3u8**
- **oss_media_hls_end_m3u8**
- **oss_media_hls_flush**
- **oss_media_hls_close**

下面详细介绍各个接口的功能和注意事项。

- **基础结构体介绍**

```
/**
 *  OSS MEDIA HLS FRAME的元数据
 */
typedef struct oss_media_hls_frame_s {
    stream_type_t stream_type;
    frame_type_t frame_type;
    uint64_t pts;
    uint64_t dts;
    uint32_t continuity_counter;
    uint8_t  key:1;
    uint8_t *pos;
    uint8_t *end;
} oss_media_hls_frame_t;

/**
 *  OSS MEDIA HLS的描述信息
 */
```

```
typedef struct oss_media_hls_options_s {
    uint16_t video_pid;
    uint16_t audio_pid;
    uint32_t hls_delay_ms;
    uint8_t encrypt:1;
    char    key[OSS_MEDIA_HLS_ENCRYPT_KEY_SIZE];
    file_handler_fn_t handler_func;
    uint16_t pat_interval_frame_count;
} oss_media_hls_options_t;

/**
 *  OSS MEDIA HLS FILE的描述信息
 */
typedef struct oss_media_hls_file_s {
    oss_media_file_t *file;
    oss_media_hls_buf_t *buffer;
    oss_media_hls_options_t options;
    int64_t frame_count;
} oss_media_hls_file_t;
```

说明:

- **stream_type：流类型，目前支持st_h264和st_aac。**
- **frame_type：帧类型，目前支持ft_non_idr、ft_idr、ft_sei、ft_sps、ft_pps、ft_aud等。**
- **pts：显示时间戳。**
- **dts：解码时间戳。**
- **continuity_counter：递增计数器，从0-15，起始值不一定取0，但必须是连续的。**
- **key：是否为关键帧。**
- **pos：当前帧数据的起始位置（包含）。**
- **end：当前帧数据的结束位置（不包含）。**
- **video_pid：视频的pid。**
- **audio_pid：音频的pid。**
- **hls_delay_ms：延迟毫秒数。**
- **encrypt：是否使用AES-128加密。目前暂不支持使用AES-128加密。**
- **key：使用加密时的密钥。目前暂不支持使用加密时的密钥。**
- **handler_func：文件操作回调函数。**
- **pat_interval_frame_count：隔多少帧插入一个pat、mpt表。**

· **打开HLS文件**

```
/**
 *  @brief  打开一个OSS HLS文件
 *  @param[in]  bucket_name oss上存储文件的存储空间名称
 *  @param[in]  object_key  oss上的文件名称
 *  @param[in]  auth_func   授权函数，设置access_key_id/access_key
_secret等
```

```
 *  @return:
 *      返回非NULL时成功，否则失败
 */
oss_media_hls_file_t* oss_media_hls_open(char *bucket_name, char *
object_key, auth_fn_t auth_func);
```

说明:

示例代码请参考*GitHub*。

- **关闭HLS文件**

```
/**
 *  @brief  关闭OSS HLS文件
 */
int oss_media_hls_close(oss_media_hls_file_t *file);
```

说明:

示例代码请参考*GitHub* 。

- **写HLS文件**

```
/**
 *  @brief  写H.264或者AAC的一帧数据到oss上
 *  @param[in]  frame         h.264或者aac格式的一帧数据
 *  @param[out] file          hls file
 *  @return:
 *      返回0时表示成功
 *      否则，表示出现了错误
 */
int oss_media_hls_write_frame(oss_media_hls_frame_t *frame,
oss_media_hls_file_t *file);
```

示例程序:

```
static void write_frame(oss_media_hls_file_t *file) {
    oss_media_hls_frame_t frame;
    FILE    *file_h264;
    uint8_t *buf_h264;
    int     len_h264, i;
    int     cur_pos = -1;
    int     last_pos = -1;
    int     video_frame_rate = 30;
    int     max_size = 10 * 1024 * 1024;
    char    *h264_file_name = "/path/to/example.h264";

    /* 读取H.264文件 */
    buf_h264 = calloc(max_size, 1);
    file_h264 = fopen(h264_file_name, "r");
    len_h264 = fread(buf_h264, 1, max_size, file_h264);

    /* 初始化frame结构体 */
    frame.stream_type = st_h264;
    frame.pts = 0;
    frame.continuity_counter = 1;
```

```
    frame.key = 1;

    /* 遍历H.264的数据，抽取出每帧数据，然后写入oss */
    for (i = 0; i < len_h264; i++) {
        /*  判断当前位置是否下一帧数据的开头，也就是当前帧的结尾 */
        if ((buf_h264[i] & 0x0F)==0x00 && buf_h264[i+1]==0x00
            && buf_h264[i+2]==0x00 && buf_h264[i+3]==0x01)
        {
            cur_pos = i;
        }

    /* 如果获取到完整的一帧数据，就调用接口转为HLS格式后写入OSS */
        if (last_pos != -1 && cur_pos > last_pos) {
            frame.pts += 90000 / video_frame_rate;
            frame.dts = frame.pts;

            frame.pos = buf_h264 + last_pos;
            frame.end = buf_h264 + cur_pos;

            oss_media_hls_write_frame(&frame, file);
        }

        last_pos = cur_pos;
    }

    /* 关闭文件，释放资源 */
    fclose(file_h264);
    free(buf_h264);
}
```

说明:

- 示例代码请参考*GitHub*
- 如果H.264的数据中缺少Access Unit Delimiter NALs（00 00 00 01 09 xx）时，需要添加这个NAL，否则无法在ipad，iphone，safari上播放。
- H.264的帧是通过0xX0，0x00，0x00，0x01分隔的；AAC的帧是通过0xFF，0x0X分隔的。
- 当前帧为关键帧时，frame.key需要设置为1。

- **写M3U8文件**

```
/**
 *  @brief   写M3U8文件的头部数据
 *  @param[in]  max_duration  TS文件最长持续时间
 *  @param[in]  sequence      TS文件起始编号
 *  @param[out] file          m3u8 file
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误
 */
void oss_media_hls_begin_m3u8(int32_t max_duration,
                              int32_t sequence,
                              oss_media_hls_file_t *file);

/**
 *  @brief   写M3U8文件数据
```

```
 *  @param[in]  size          m3u8 item个数
 *  @param[in]  m3u8          m3u8 item的详细数据
 *  @param[out] file          m3u8 file
 *  @return:
 *      返回0时表示成功
 *      否则，返回-1时表示出现了错误
 */
int oss_media_hls_write_m3u8(int size,
                             oss_media_hls_m3u8_info_t m3u8[],
                             oss_media_hls_file_t *file);

/**
 *  @brief  写M3U8文件的结束符等数据
 *  @param[out] file          m3u8 file
 */
void oss_media_hls_end_m3u8(oss_media_hls_file_t *file);
```

示例程序：

```
static void write_m3u8() {
    char *bucket_name;
    char *key;
    oss_media_hls_file_t *file;

    bucket_name = "<your bucket name>";
    key = "<your m3u8 file name>";

    /* 打开一个HLS文件用来写M3U8格式的数据，文件名必须以.m3u8结尾 */
    file = oss_media_hls_open(bucket_name, key, auth_func);
    if (file == NULL) {
        printf("open m3u8 file[%s] failed.", key);
        return;
    }

    /* 构造3个ts格式文件的信息 */
    oss_media_hls_m3u8_info_t m3u8[3];
    m3u8[0].duration = 9;
    memcpy(m3u8[0].url, "video-0.ts", strlen("video-0.ts"));
    m3u8[1].duration = 10;
    memcpy(m3u8[1].url, "video-1.ts", strlen("video-1.ts"));

    /* 写入M3U8文件
    oss_media_hls_begin_m3u8(10, 0, file);
    oss_media_hls_write_m3u8(2, m3u8, file);
    oss_media_hls_end_m3u8(file);

    /* 关闭HLS文件 */
    oss_media_hls_close(file);

    printf("write m3u8 to oss file succeeded\n");
}
```

说明:

- 目前使用的M3U8版本号是3。
- 如果是录播，需要在结束的时候调用oss_media_hls_end_m3u8(file)接口写入结束符，否则可能无法播放；如果是直播，则不能调用此接口。

- 示例代码请参考*GitHub*
- 可以通过示例程序观看效果。
- Windows平台可以通过*VLC播放器*观看，iPhone、iPad、Mac等可以直接使用Safari观看。

14.7 HLS封装接口

OSS MEDIA C SDK 客户端部分支持将接收到的H.264、AAC封装为TS、M3U8格式后写入OSS，除了基础接口外，还提供封装好的录播、直播接口。

接口

HLS相关封装接口都位于oss_media_hls_stream.h中，目前提供的接口有：

- **oss_media_hls_stream_open**
- **oss_media_hls_stream_write**
- **oss_media_hls_stream_close**

下面详细介绍各个接口的功能和注意事项。

- **基础结构体介绍**

```
/**
 *  OSS MEDIA HLS STREAM OPTIONS的描述信息
 */
typedef struct oss_media_hls_stream_options_s {
    int8_t is_live;
    char *bucket_name;
    char *ts_name_prefix;
    char *m3u8_name;
    int32_t video_frame_rate;
    int32_t audio_sample_rate;
    int32_t hls_time;
    int32_t hls_list_size;
} oss_media_hls_stream_options_t;

/**
 *  OSS MEDIA HLS STREAM的描述信息
 */
typedef struct oss_media_hls_stream_s {
    const oss_media_hls_stream_options_t *options;
    oss_media_hls_file_t *ts_file;
    oss_media_hls_file_t *m3u8_file;
    oss_media_hls_frame_t *video_frame;
    oss_media_hls_frame_t *audio_frame;
    oss_media_hls_m3u8_info_t *m3u8_infos;
    int32_t ts_file_index;
    int64_t current_file_begin_pts;
    int32_t has_aud;
    aos_pool_t *pool;
} oss_media_hls_stream_t;
```

说明:

- is_live：是否为直播模式。直播模式时，M3U8文件里面只包含最新的几个ts文件信息。
- bucket_name：存储HLS文件的存储空间名称。
- ts_name_prefix：TS文件名称的前缀。
- m3u8_name：M3U8文件名称。
- video_frame_rate：视频数据的帧率。
- audio_sample_rate：音频数据的采样率。
- hls_time：每个ts文件最大持续时间。
- hls_list_size：直播模式时在M3U8文件中最多保留的ts文件个数。

- 打开HLS stream文件

```
/**
 *  @brief   打开一个oss hls文件
 *  @param[in]  auth_func    授权函数，设置access_key_id/access_key
_secret等
 *  @param[in]  options      配置信息
 *  @return:
 *      返回非NULL时成功，否则失败
 */
oss_media_hls_stream_t* oss_media_hls_stream_open(auth_fn_t
auth_func,
                        const oss_media_hls_stream_options_t *
options);
```

说明:

示例代码请参考*GitHub*。

- 关闭HLS stream文件

```
/**
 *  @brief   关闭HLS stream文件
 */
int oss_media_hls_stream_close(oss_media_hls_stream_t *stream);
```

说明:

示例代码请参考*GitHub*。

- 写HLS stream文件

```
/**
 *  @brief   写入视频和音频数据
 *  @param[in]  video_buf 视频数据
```

```
 *  @param[in]  video_len 视频数据的长度，可以为0
 *  @param[in]  audio_buf 音频数据
 *  @param[in]  audio_len 音频数据的长度，可以为0
 *  @param[in]  stream    HLS stream
 *  @return:
 *      返回0时表示成功
 *      否则，表示出现了错误
 */
int oss_media_hls_stream_write(uint8_t *video_buf,
                               uint64_t video_len,
                               uint8_t *audio_buf,
                               uint64_t audio_len,
                               oss_media_hls_stream_t *stream);
```

示例程序：

```
static void write_video_audio_vod() {
    int ret;
    int max_size = 10 * 1024 * 1024;
    FILE *h264_file, *aac_file;
    uint8_t *h264_buf, *aac_buf;
    int h264_len, aac_len;

    oss_media_hls_stream_options_t options;
    oss_media_hls_stream_t *stream;

    /* 设置HLS stream的参数值 */
    options.is_live = 0;
    options.bucket_name = "<your buckete name>";
    options.ts_name_prefix = "vod/video_audio/test";
    options.m3u8_name = "vod/video_audio/vod.m3u8";
    options.video_frame_rate = 30;
    options.audio_sample_rate = 24000;
    options.hls_time = 5;

    /* 打开HLS stream */
    stream = oss_media_hls_stream_open(auth_func, &options);
    if (stream == NULL) {
        printf("open hls stream failed.\n");
        return;
    }

    /* 创建两个buffer用来存储音频和视频数据 */
    h264_buf = malloc(max_size);
    aac_buf = malloc(max_size);

    /* 读取一段视频数据和音频数据，然后调用接口写入OSS */
    {
        h264_file = fopen("/path/to/video/1.h264", "r");
        h264_len = fread(h264_buf, 1, max_size, h264_file);
        fclose(h264_file);

        aac_file = fopen("/path/to/audio/1.aac", "r");
        aac_len = fread(aac_buf, 1, max_size, aac_file);
        fclose(aac_file);

        ret = oss_media_hls_stream_write(h264_buf, h264_len,
                aac_buf, aac_len, stream);
        if (ret != 0) {
            printf("write vod stream failed.\n");
            return;
        }
```

```
    }

    /* 再读取一段视频数据和音频数据，然后调用接口写入OSS */
    {
        h264_file = fopen("/path/to/video/2.h264", "r");
        h264_len = fread(h264_buf, 1, max_size, h264_file);
        fclose(h264_file);

        aac_file = fopen("/path/to/audio/1.aac", "r");
        aac_len = fread(aac_buf, 1, max_size, aac_file);
        fclose(aac_file);

        ret = oss_media_hls_stream_write(h264_buf, h264_len,
                aac_buf, aac_len, stream);
        if (ret != 0) {
            printf("write vod stream failed.\n");
            return;
        }
    }

    /* 写完数据后，关闭HLS stream */
    ret = oss_media_hls_stream_close(stream);
    if (ret != 0) {
        printf("close vod stream failed.\n");
        return;
    }

    /* 释放资源 */
    free(h264_buf);
    free(aac_buf);

    printf("convert H.264 and aac to HLS vod succeeded\n");
}
```

说明:

- 目前的录播、直播接口都支持只有视频、只有音频或同时有音视频等。您可以通过示例程序观看效果。
- 示例代码请参考*GitHub*
- 目前仅提供初级的录播、直播接口。如果您有高级需求，建议模拟这两个接口，并使用基础接口自助实现高级定制功能。

14.8 使用场景

本文主要介绍OSS Media-C SDK的使用场景。

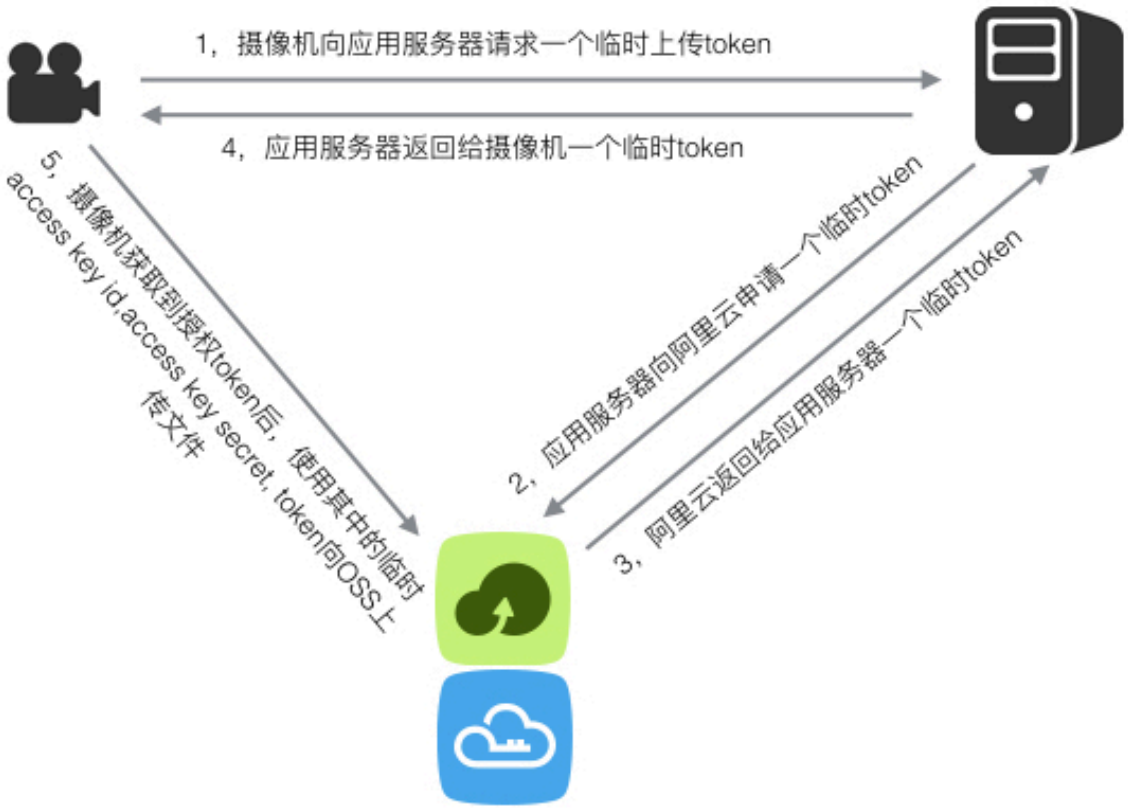
注意:

请先了解客户端和服务端的相关操作。

上传视频文件到OSS

在前言章节中介绍了OSS Media-C SDK可以方便地用于网络摄像头等设备。

使用OSS Media-C SDK将网络摄像机拍摄的视频上传到OSS，步骤如下。



说明:

网络摄像机对应OSS Media-C SDK的客户端（client）部分，应用服务器对应OSS Media-C SDK的服务端（server）部分。

1. **网络摄像机向应用服务器请求获取上传视频到OSS的权限。**
2. **应用服务器收到请求后，通过OSS Media-C SDK的get_token接口，向阿里云请求一个在特定时间内有效、只有上传权限的token。**
3. **阿里云接收到应用服务器的获取token请求后，通过检查用户的配置，颁发一个临时token（包括临时access key id、临时access key secret和临时sts token），然后发送给应用服务器。该临时token只有上传OSS的权限，且仅在特定时间内有效。**
4. **应用服务器收到临时token后，将其转发给请求获取token的网络摄像机。**
5. **网络摄像机获取token后，通过OSS Media-C SDK client部分的oss_media_write接口将视频文件上传到OSS。**

说明:

您还可以在应用服务器中使用OSS Media-C SDK的server部分，或者通过Java、Go、Python、Ruby等SDK实现HTTP服务，便于直接在网页上查看并管理各个视频文件

示例代码

以下示例代码用于简单模拟客户端和服务端操作：

```
char* global_temp_access_key_id = NULL;
char* global_temp_access_key_secret = NULL;
char* global_temp_token = NULL;
/* 授权函数 */
static void auth_func(oss_media_file_t *file) {
    file->endpoint = "your endpoint";
    file->is_cname = 0;
    file->access_key_id = global_temp_access_key_id;
    file->access_key_secret = global_temp_access_key_secret;
    file->token = global_temp_token;
    /* 本次授权的有效时间  */
    file->expiration = time(NULL) + 300;
}
/* 模拟服务端发送token给客户端 */
static void send_token_to_client(oss_media_token_t token) {
    global_temp_access_key_id = token.tmpAccessKeyId;
    global_temp_access_key_secret = token.tmpAccessKeySecret;
    global_temp_token = token.securityToken;
}
void get_and_use_token() {
    oss_media_token_t token;
    /* 服务端逻辑：从阿里云获取到临时token后发送给客户端 */
    {
        int ret;
        char *policy = NULL;
        oss_media_config_t config;

        policy = "{\n"
                 "\"Statement\": [\n"
                 "{"
                 "\"Action\": \"oss:*\",\n"
                 "\"Effect\": \"Allow\",\n"
                 "\"Resource\": \"*\"\n"
                 "}\n"
                 "],\n"
                 "\"Version\": \"1\"\n"
                 "}\n";
        init_media_config(&config);
 /* 从阿里云请求一个临时授权token */
        ret = oss_media_get_token_from_policy(&config, policy,
                 17 * 60, &token);
 if (ret != 0) {
    printf ("Get token failed.");
    return;
 }
 /* 模拟将临时token发送给客户端 */
        send_token_to_client(token);
    }
    /* 客户端逻辑：从服务端获取到临时token后，使用临时token操作文件 */
    {
        int ret;
        int64_t write_size = 0;
        oss_media_file_t *file = NULL;
```

```
        char *content = NULL;
 char *bucket_name;
 char *object_key;
        oss_media_file_stat_t stat;
        content = "hello oss media file\n";
 bucket_name = "<your bucket name>";
 object_key = "key";
        /* 打开文件 */
        file = oss_media_file_open(bucket_name, object_key, "w",
auth_func);
        if (file != NULL) {
     printf ("open file failed.");
     return;
 }
        /* 写文件 */
        write_size = oss_media_file_write(file, content, strlen(
content));
        if (write_size != strlen(content)) {
     printf ("write file failed.");
     return;
 }
        /* 关闭文件释放资源 */
        oss_media_file_close(file);
    }
}
```

说明:

- **Policy可以从阿里云的*RAM角色管理*的任意RAM角色下的策略内容中获取。**
- **如果不需要精准控制权限，可以使用更简单的oss_media_get_token接口，其中path参数可以是/*，mode参数可以是rwa。**
- **关于使用RAM、STS的更多信息，请参考*RAM和STS指南*。**

14.9 常见问题

本文介绍使用OSS Media-C SDK的常见问题及解决方法。

OSS Media-C SDK和OSS C SDK有怎样的联系？

OSS Media-C SDK依赖于OSS C SDK，OSS Media-C SDK中的上传和下载等功能是通过调用OSS C SDK的接口实现的。

OSS Media-C SDK是否支持Windows？

目前还不支持。

是否支持追加写文件？

支持，调用oss_media_file_open时使用“a”模式，然后通过多次调用oss_media_file_write接口实现追加写。

什么是role arn？如何获取role arn？

- **role arn表示需要扮演角色的id。**
- **您可以在*RAM*访问控制管理控制台，选择RAM角色管理，然后单击已经创建的角色名称。在基本信息区域中查看ARN，其值格式为acs:ram::xxxx:role/yyyyy。如果还没有已创建的角色，需要在角色管理页面创建一个新的用户角色和其他相应角色，并赋予新的用户角色AliyunSTSAssumeRoleAccess权限，更详细的介绍请参考*RAM*的文档。**

如何运行sample?

修改sample/config.c文件，赋予access key id、access key secret、bucket等值，完成编译后在bin目录下可以找到sample的可执行文件。

运行sample时报错：error:Couldn't resolve host name 和[code=-990, message=HttpIoError]

修改sample/config.c文件，配置相关参数值，然后重新编译即可。

客户端和服务端的access key id，access key secret，token配置之间的差异和注意点?

- **服务端只需要配置access key id和access key secret，这两个值可以是主账号的AccessKey，也可以是主账号生成的子账号的AccessKey。**
- **客户端有两种配置方式：第一种是和服务端一样，只配置主账号或者子账号的access key id、access key secret；第二种是配置access key id、access key secret和token三个值，这三个值都是服务端通过oss_media_get_token或者oss_media_get_token_from_policy获取的临时AccessKey和token，该AccessKey和token有时间期限，超过有效期后，不能再次使用。**

执行sample获取token的时候出现以下错误：http_code=500, error_code=GetSTSTokenError, error_message=Internal Error

- **原因：安装的libcurl不支持HTTPS协议，导致无法访问sts服务。具体过程是机器上没有安装openssl-devel等ssl的开发包，在编译libcurl时找不到ssl，libcurl会自动禁止HTTPS协议，导致编译出来的libcurl库不支持HTTPS，最终访问STS失败。**
- **解决办法：先安装openssl-devel等ssl开发包，然后重新安装libcurl。安装libcurl时执行了./configure后，检查并确认最后一行Protocols里已包含HTTPS。**

15 conref