阿里云
对象存储

开发指南

文档版本：20190912

法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 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。	`swich` *`{stand | slave}`*

目录

1 使用前须知

本文列出您在使用阿里云对象存储OSS前需要了解的内容。

如果您初次使用阿里云OSS，请参见阿里云OSS快速入门系列文档，帮助您了解OSS并快速使用OSS。

如果您已经充分了解OSS，您也可以通过下列资源快速使用OSS的其他各项功能：

资源	描述
阿里云OSS开发人员指南	本文档为您讲解阿里云OSS服务的核心概念、所有功能介绍与操作步骤，以及如何使用API和SDK的有效示例。
阿里云OSS最佳实践	详细介绍阿里云OSS的各种使用场景与配置实践。
阿里云OSS SDK参考	介绍主流语言的SDK开发操作和参数。
阿里云OSS API参考	详细探讨了阿里云OSS支持的RESTful API操作和相关的示例。
阿里云OSS 控制台用户指南	阿里云OSS管理控制台可让您通过界面执行OSS的部分功能。本文档为您介绍了基于阿里云OSS管理控制台的所有操作。
阿里云OSS图片处理指南	详细探讨了阿里云OSS提供的图片处理服务的详细内容与操作方式。
阿里云OSS迁移工具	您可能需要将您本地或第三方云存储服务上的文件同步到阿里云OSS上，阿里云为此提供了完善的迁移解决方案。

2 基本概念介绍

本部分将向您介绍本产品中涉及的几个基本概念，以便于您更好地理解对象存储 OSS 产品。

存储空间（Bucket）

存储空间是您用于存储对象（Object）的容器，所有的对象都必须隶属于某个存储空间。您可以设置和修改存储空间属性用来控制地域、访问权限、生命周期等，这些属性设置直接作用于该存储空间内所有对象，因此您可以通过灵活创建不同的存储空间来完成不同的管理功能。

- 同一个存储空间的内部是扁平的，没有文件系统的目录等概念，所有的对象都直接隶属于其对应的存储空间。
- 每个用户可以拥有多个存储空间。
- 存储空间的名称在 OSS 范围内必须是全局唯一的，一旦创建之后无法修改名称。
- 存储空间内部的对象数目没有限制。

存储空间的命名规范如下：

- 只能包括小写字母、数字和短横线（-）。
- 必须以小写字母或者数字开头和结尾。
- 长度必须在 3–63 字节之间。

对象/文件（Object）

对象是 OSS 存储数据的基本单元，也被称为 OSS 的文件。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由存储空间内部唯一的 Key 来标识。对象元信息是一对键值对，表示了对象的一些属性，比如最后修改时间、大小等信息，同时用户也可以在元信息中存储一些自定义的信息。

对象的生命周期是从上传成功到被删除为止。在整个生命周期内，对象内容无法编辑，您可以通过重复上传同名的对象来覆盖之前的对象。

OSS 提供了追加上传功能，用户可以使用该功能不断地在 Object 尾部追加写入数据。

对象的命名规范如下：

- 使用 UTF-8 编码。
- 长度必须在 1–1023 字节之间。
- 不能以正斜线（/）或者反斜线（\）开头。

> **说明：**
> 对象名称需要区分大小写。如无特殊说明，本文档中的对象、文件称谓等同于 Object。

Region（地域）

Region 表示 OSS 的数据中心所在的地域，物理位置。用户可以根据费用、请求来源等综合选择数据存储的 Region。一般来说，距离用户更近的 Region 访问速度更快。详情请查看 OSS 已经开通的 Region。

Region是在创建 Bucket 的时候指定的，一旦指定之后就不允许更改。该 Bucket 下所有的 Object 都存储在对应的数据中心，目前不支持 Object 级别的 Region 设置。

Endpoint（访问域名）

Endpoint 表示 OSS 对外服务的访问域名。OSS 以 HTTP RESTful API 的形式对外提供服务，当访问不同的 Region 的时候，需要不同的域名。通过内网和外网访问同一个 Region 所需要的 Endpoint 也是不同的。例如杭州 Region 的外网 Endpoint 是 oss-cn-hangzhou.aliyuncs.com，内网 Endpoint 是 oss-cn-hangzhou-internal.aliyuncs.com。具体的内容请参见各个 Region 对应的 Endpoint。

AccessKey（访问密钥）

AccessKey（简称 AK）指的是访问身份验证中用到的 AccessKeyId 和 AccessKeySecret。OSS 通过使用 AccessKeyId 和 AccessKeySecret 对称加密的方法来验证某个请求的发送者身份。AccessKeyId 用于标识用户，AccessKeySecret 是用户用于加密签名字符串和 OSS 用来验证签名字符串的密钥，其中 AccessKeySecret 必须保密。对于 OSS 来说，AccessKey 的来源有：

- Bucket 的拥有者申请的 AccessKey。
- 被 Bucket 的拥有者通过 RAM 授权给第三方请求者的 AccessKey。
- 被 Bucket 的拥有者通过 STS 授权给第三方请求者的 AccessKey。

更多 AccessKey 介绍请参见访问控制。

强一致性

Object 操作在 OSS 上具有原子性，操作要么成功要么失败，不会存在有中间状态的 Object。OSS 保证用户一旦上传完成之后读到的 Object 是完整的，OSS 不会返回给用户一个部分上传成功的 Object。

Object 操作在 OSS 上同样具有强一致性，用户一旦收到了一个上传（PUT）成功的响应，该上传的 Object 就已经立即可读，并且 Object 的冗余数据已经写成功。不存在一种上传的中间状态，即 read-after-write 却无法读取到数据。对于删除操作也是一样的，用户删除指定的 Object 成功之后，该 Object 立即变为不存在。

数据冗余机制

OSS 采用数据冗余存储机制，将每个对象的不同冗余存储在同一个区域内多个设施的多个设备上，确保硬件失效时的数据可靠性和可用性。

- OSS Object 操作具有强一致性，用户一旦收到了上传/复制成功的响应，则该上传的 Object 就已经立即可读，且数据已经冗余写入到多个设备中。
- OSS 会通过计算网络流量包的校验和，验证数据包在客户端和服务端之间传输中是否出错，保证数据完整传输。
- OSS 的冗余存储机制，可支持两个存储设施并发损坏时，仍维持数据不丢失。
  - 当数据存入 OSS 后，OSS 会检测和修复丢失的冗余，确保数据可靠性和可用性。
  - OSS 会周期性地通过校验等方式验证数据的完整性，及时发现因硬件失效等原因造成的数据损坏。当检测到数据有部分损坏或丢失时，OSS 会利用冗余的数据，进行重建并修复损坏数据。

OSS 与文件系统的对比

对比项	OSS	文件系统
数据模型	OSS 是一个分布式的对象存储服务，提供的是一个 Key-Value 对形式的对象存储服务。	文件系统是一种典型的树状索引结构。
数据获取	根据 Object 的名称（Key）唯一的获取该 Object 的内容。 虽然用户可以使用类似 test1/test.jpg 的名字，但是这并不表示用户的 Object 是保存在 test1 目录下面的。对于 OSS 来说，test1/test.jpg 仅仅只是一个字符串，和a.jpg 这种并没有本质的区别。因此不同名称的 Object 之间的访问消耗的资源是类似的。	一个名为 test1/test.jpg 的文件，访问过程需要先访问到 test1 这个目录，然后再在该目录下查找名为 test.jpg 的文件。
优势	支持海量的用户并发访问。	支持文件的修改，比如修改指定偏移位置的内容、截断文件尾部等。也支持文件夹的操作，比如重命名目录、删除目录、移动目录等非常容易。

对比项	OSS	文件系统
劣势	OSS 保存的 Object 不支持修改（追加写 Object 需要调用特定的接口，生成的 Object 也和正常上传的 Object 类型上有差别）。用户哪怕是仅仅需要修改一个字节也需要重新上传整个 Object。 OSS 可以通过一些操作来模拟类似文件夹的功能，但是代价非常昂贵。比如重命名目录，希望将 test1 目录重命名成 test2，那么 OSS 的实际操作是将所有以 test1/ 开头的 Object 都重新复制成以 test2/ 开头的 Object，这是一个非常消耗资源的操作。因此在使用 OSS 的时候要尽量避免类似的操作。	受限于单个设备的性能。访问越深的目录消耗的资源也越大，操作拥有很多文件的目录也会非常慢。

因此，将 OSS 映射为文件系统是非常低效的，也是不建议的做法。如果一定要挂载成文件系统的话，建议尽量只做写新文件、删除文件、读取文件这几种操作。使用 OSS 应该充分发挥其优点，即海量数据处理能力，优先用来存储海量的非结构化数据，比如图片、视频、文档等。

以下是 OSS 与文件系统的概念对比：

对象存储 OSS	文件系统
Object	文件
Bucket	主目录
Region	无
Endpoint	无
AccessKey	无
无	多级目录
GetService	获取主目录列表
GetBucket	获取文件列表
PutObject	写文件

对象存储 OSS	文件系统
AppendObject	追加写文件
GetObject	读文件
DeleteObject	删除文件
无	修改文件内容
CopyObject （目的和源相同）	修改文件属性
CopyObject	复制文件
无	重命名文件

3 访问域名（Endpoint）

3.1 访问域名和数据中心

Region表示OSS的数据中心所在的地域，Endpoint表示OSS对外服务的访问域名。本文主要介绍Region与Endpoint的对应关系。

OSS开通Region和Endpoint对照表

经典网络情况下各地域Endpoint的内外网设置如下：

Region中文名称	Region英文表示	外网Endpoint	外网支持HTTPS	ECS访问的内网Endpoint	内网支持HTTPS
华东 1（杭州）	oss-cn-hangzhou	oss-cn-hangzhou.aliyuncs.com	是	oss-cn-hangzhou-internal.aliyuncs.com	是
华东 2（上海）	oss-cn-shanghai	oss-cn-shanghai.aliyuncs.com	是	oss-cn-shanghai-internal.aliyuncs.com	是
华北 1（青岛）	oss-cn-qingdao	oss-cn-qingdao.aliyuncs.com	是	oss-cn-qingdao-internal.aliyuncs.com	是
华北 2（北京）	oss-cn-beijing	oss-cn-beijing.aliyuncs.com	是	oss-cn-beijing-internal.aliyuncs.com	是
华北 3（张家口）	oss-cn-zhangjiakou	oss-cn-zhangjiakou.aliyuncs.com	是	oss-cn-zhangjiakou-internal.aliyuncs.com	是

Region中文名称	Region英文表示	外网Endpoint	外网支持HTTPS	ECS访问的内网Endpoint	内网支持HTTPS
华北 5（呼和浩特）	oss-cn-huhehaote	oss-cn-huhehaote.aliyuncs.com	是	oss-cn-huhehaote-internal.aliyuncs.com	是
华南 1（深圳）	oss-cn-shenzhen	oss-cn-shenzhen.aliyuncs.com	是	oss-cn-shenzhen-internal.aliyuncs.com	是
西南 1（成都）	oss-cn-chengdu	oss-cn-chengdu.aliyuncs.com	是	oss-cn-chengdu-internal.aliyuncs.com	是
中国（香港）	oss-cn-hongkong	oss-cn-hongkong.aliyuncs.com	是	oss-cn-hongkong-internal.aliyuncs.com	是
美国西部 1（硅谷）	oss-us-west-1	oss-us-west-1.aliyuncs.com	是	oss-us-west-1-internal.aliyuncs.com	是
美国东部 1（弗吉尼亚）	oss-us-east-1	oss-us-east-1.aliyuncs.com	是	oss-us-east-1-internal.aliyuncs.com	是
亚太东南 1（新加坡）	oss-ap-southeast-1	oss-ap-southeast-1.aliyuncs.com	是	oss-ap-southeast-1-internal.aliyuncs.com	是
亚太东南 2（悉尼）	oss-ap-southeast-2	oss-ap-southeast-2.aliyuncs.com	是	oss-ap-southeast-2-internal.aliyuncs.com	是

Region中文名称	Region英文表示	外网Endpoint	外网支持HTTPS	ECS访问的内网Endpoint	内网支持HTTPS
亚太东南 3（吉隆坡）	oss-ap-southeast-3	oss-ap-southeast-3.aliyuncs.com	是	oss-ap-southeast-3-internal.aliyuncs.com	是
亚太东南 5（雅加达）	oss-ap-southeast-5	oss-ap-southeast-5.aliyuncs.com	是	oss-ap-southeast-5-internal.aliyuncs.com	是
亚太东北 1（日本）	oss-ap-northeast-1	oss-ap-northeast-1.aliyuncs.com	是	oss-ap-northeast-1-internal.aliyuncs.com	是
亚太南部 1（孟买）	oss-ap-south-1	oss-ap-south-1.aliyuncs.com	是	oss-ap-south-1-internal.aliyuncs.com	是
欧洲中部 1（法兰克福）	oss-eu-central-1	oss-eu-central-1.aliyuncs.com	是	oss-eu-central-1-internal.aliyuncs.com	是
英国（伦敦）	oss-eu-west-1	oss-eu-west-1.aliyuncs.com	是	oss-eu-west-1-internal.aliyuncs.com	是
中东东部 1（迪拜）	oss-me-east-1	oss-me-east-1.aliyuncs.com	是	oss-me-east-1-internal.aliyuncs.com	是

说明:

- 在分享链接或者做自定义域名绑定（CNAME）的时候建议使用三级域名，即`Bucket + Endpoint`的形式。以华东2地域内名为oss-sample的Bucket为例，三级域名为oss-sample.oss-cn-shanghai.aliyuncs.com。

- 使用SDK时，请将 `http://` 或 `https://` + `Endpoint` 作为初始化的参数。以华东2的Endpoint为例，建议将初始化参数设置为`http://oss-cn-shanghai.aliyuncs.com` 或者 `https://oss-cn-shanghai.aliyuncs.com`，不建议将三级域名（即`http://bucket.oss-cn-shanghai.aliyuncs.com`）作为初始化参数。
- 原地址oss.aliyuncs.com 默认指向华东1 地域外网地址。原内网地址oss-internal.aliyuncs.com 默认指向华东 1 地域内网地址。

VPC网络下Region和Endpoint对照表

在VPC网络下的ECS访问OSS可以使用如下的Endpoint：

Region中文名称	Region英文表示	VPC网络Endpoint	支持HTTPS
华东 1（杭州）	oss-cn-hangzhou	oss-cn-hangzhou-internal.aliyuncs.com	是
华东 2（上海）	oss-cn-shanghai	oss-cn-shanghai-internal.aliyuncs.com	是
华北 1（青岛）	oss-cn-qingdao	oss-cn-qingdao-internal.aliyuncs.com	是
华北 2（北京）	oss-cn-beijing	oss-cn-beijing-internal.aliyuncs.com	是
华北 3（张家口）	oss-cn-zhangjiakou	oss-cn-zhangjiakou-internal.aliyuncs.com	是
华北 5（呼和浩特）	oss-cn-huhehaote	oss-cn-huhehaote-internal.aliyuncs.com	是
华南 1（深圳）	oss-cn-shenzhen	oss-cn-shenzhen-internal.aliyuncs.com	是
西南 1（成都）	oss-cn-chengdu	oss-cn-chengdu-internal.aliyuncs.com	是
中国（香港）	oss-cn-hongkong	oss-cn-hongkong-internal.aliyuncs.com	是

Region中文名称	Region英文表示	VPC网络Endpoint	支持HTTPS
美国西部 1 （硅谷）	oss-us-west-1	oss-us-west-1-internal.aliyuncs.com	是
美国东部 1 （弗吉尼亚）	oss-us-east-1	oss-us-east-1-internal.aliyuncs.com	是
亚太东南 1 （新加坡）	oss-ap-southeast-1	oss-ap-southeast-1-internal.aliyuncs.com	是
亚太东南 2 （悉尼）	oss-ap-southeast-2	oss-ap-southeast-2-internal.aliyuncs.com	是
亚太东南 3 （吉隆坡）	oss-ap-southeast-3	oss-ap-southeast-3-internal.aliyuncs.com	是
亚太东南 5 （雅加达）	oss-ap-southeast-5	oss-ap-southeast-5-internal.aliyuncs.com	是
亚太东北 1 （日本）	oss-ap-northeast-1	oss-ap-northeast-1-internal.aliyuncs.com	是
亚太南部 1 （孟买）	oss-ap-south-1	oss-ap-south-1-internal.aliyuncs.com	是
欧洲中部 1 （法兰克福）	oss-eu-central-1	oss-eu-central-1-internal.aliyuncs.com	是
英国（伦敦）	oss-eu-west-1	oss-eu-west-1-internal.aliyuncs.com	是
中东东部 1 （迪拜）	oss-me-east-1	oss-me-east-1-internal.aliyuncs.com	是

如何使用访问域名

- 关于OSS域名的构成规则以及如何使用内网和外网访问OSS，请参见OSS访问域名使用规则。
- 如果您是ECS用户，需要使用OSS内网地址，请参见 ECS 用户如何正确使用OSS内网地址。

3.2 OSS访问域名使用规则

OSS会为每一个存储空间（Bucket）分配默认的访问域名，本文介绍OSS访问域名的构成规则及使用方式。

OSS域名构成规则

针对OSS的网络请求，除了GetService这个API以外，其他所有请求的域名都是带有指定Bucket信息的三级域名组成的。

访问域名结构：BucketName.Endpoint。BucketName为您的存储空间名称，Endpoint为存储空间对应的地域域名。

> 说明:
>
> - OSS以HTTP RESTful API的形式对外提供服务，当访问不同的地域（Region）时，需要不同的访问域名。
> - Endpoint分内网和外网访问域名。例如，华东1（杭州）地域的外网Endpoint是`oss-cn-hangzhou.aliyuncs.com`，内网Endpoint是`oss-cn-hangzhou-internal.aliyuncs.com`。Region和Endpoint对照表请参考访问域名和数据中心。
> - 您也可以通过绑定自定义域名或绑定CDN加速域名，将OSS的外网访问域名替换为您的自有域名。

通过外网访问OSS服务

外网指的是互联网。通过外网访问产生的流入流量（写）是免费的，流出流量（读）是收费的。

> 说明:
> OSS费用详情请参见OSS服务价格页。

外网访问OSS有如下两种方式：

- 访问方式一：访问时以URL的形式来表示OSS的资源。OSS的URL构成如下：

  ```
  <Schema>://<Bucket>.<外网Endpoint>/<Object>
  ```

  - Schema：HTTP或者为HTTPS
  - Bucket：OSS存储空间
  - Endpoint：Bucket所在数据中心的访问域名，您需要填写外网Endpoint
  - Object：上传到OSS上的文件

  示例：如您的Region为华东1（oss-cn-hangzhou），Bucket名称为abc，Object访问路径为*myfile/aaa.txt*，那么您的外网访问地址为：

  ```
  abc.oss-cn-hangzhou.aliyuncs.com/myfile/aaa.txt
  ```

  注意：

  OSS访问域名需携带Object访问路径才可以被访问，仅访问域名，如`abc.oss-cn-hangzhou.aliyuncs.com`，会有报错提示。若您希望直接访问OSS访问域名，可以通过配置#unique_20来实现。

  您还可以直接将Object的URL放入HTML中使用，如下所示：

  ```
  <img src="https://abc.oss-cn-hangzhou.aliyuncs.com/mypng/aaa.png" />
  ```

- 访问方式二： 通过OSS SDK配置外网访问域名。

  OSS SDK会对您的每一个操作拼接访问域名。但您在对不同地域的Bucket进行操作的时候需要设置不同的Endpoint。

  以Java SDK为例，对华东1的Bucket进行操作时，需要在对类实例化时设置Endpoint：

  ```
  String accessKeyId = "<key>";
    String accessKeySecret = "<secret>";
    String endpoint = "oss-cn-hangzhou.aliyuncs.com";
    OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeyS
  ecret);
  ```

通过内网访问OSS服务

内网指的是阿里云产品之间的内网通信网络，例如您通过ECS云服务器访问OSS服务。内网产生的流入和流出流量均免费，但是请求次数仍会计费。

内网访问OSS有如下两种方式：

- 访问方式一：在访问的时候以URL的形式来表示OSS的资源。OSS的URL构成如下。

```
<Schema>://<Bucket>.<内网Endpoint>/<Object>
```

  - Schema：HTTP或者为HTTPS
  - Bucket：OSS存储空间
  - Endpoint：Bucket所在数据中心的访问域名，您需要填写内网Endpoint
  - Object：上传到OSS上的文件

  示例：如您的Bucket名称为abc，Region为华东1，Object名称为myfile/aaa.txt，那么您的内网访问地址为：

```
abc.oss-cn-hangzhou-internal.aliyuncs.com/myfile/aaa.txt
```

- 访问方式二：通过ECS使用OSS SDK配置内网访问域名。

  以Java SDK为例，对华东1地域的Bucket进行操作时，需要在对类实例化时设置Endpoint：

```
String accessKeyId = "<key>";
  String accessKeySecret = "<secret>";
  String endpoint = "oss-cn-hangzhou-internal.aliyuncs.com";
  OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeyS
ecret);
```

   注意：

  同一个Region的ECS和OSS之间内网互通，不同Region的ECS和OSS之间内网不互通。

  例如，您的OSS有两个Bucket，并且购买了华北2（oss-cn-beijing）的ECS：

  - 其中一个Bucket名称为*beijingres*，Region为华北2，那么在华北2的ECS中可以使用`beijingres.oss-cn-beijing-internal.aliyuncs.com`来访问 beijingres 的资源。
  - 另外一个Bucket名称为*qingdaores*，Region为华北1，那么在华北2的ECS用内网地址`qingdaores.oss-cn-qingdao-internal.aliyuncs.com`是无法访问OSS的，必须使用外网地址`qingdaores.oss-cn-qingdao.aliyuncs.com`。

4 存储类型

4.1 存储类型介绍

对象存储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。

标准存储类型（Standard）

OSS标准存储类型提供高可靠、高可用、高性能的对象存储服务，能够支持频繁的数据访问。OSS高吞吐和低延时的服务响应能力能够有效支持各种热点类型数据的访问。标准存储类型是各种社交、分享类的图片、音视频应用、大型网站、大数据分析的合适选择。

- 支持的冗余类型
  - 本地冗余：采用数据冗余存储机制，将每个对象的不同冗余存储在同一个区域内多个设施的多个设备上，确保硬件失效时的数据可靠性和可用性。

    说明:
    标准存储（本地冗余）即为同城冗余发布之前的标准存储类型。
  - 同城冗余：采用多可用区（AZ）机制，将用户的数据分散存放在同一地域（Region）的3个可用区。当某个可用区不可用时，仍然能够保障数据的正常访问。
- 关键特性
  - 99.9999999999%（12个9）的数据可靠性
  - 按照99.995%服务可用性设计
  - 低延时、高吞吐的访问性能
  - 支持HTTPS加密传输
  - 支持图片处理

低频访问存储类型（Infrequent Access）

OSS低频访问存储类型适合长期保存不经常访问的数据（平均每月访问频率1到2次）。存储单价低于标准类型，适合各类移动应用、智能设备、企业数据的长期备份，支持实时数据访问。低频访问存储类型的Object有最短存储时间，存储时间短于30天的Object提前被删除会产生一定费用。低频访问存储Object有最小计量空间，Object大小低于64KB，会按照64KB计算存储空间。数据获取会产生数据取回费用。

- 支持的冗余类型
  - 本地冗余：采用数据冗余存储机制，将每个对象的不同冗余存储在同一个区域内多个设施的多个设备上，确保硬件失效时的数据可靠性和可用性。

    说明:
    低频访问存储（本地冗余）即为同城冗余发布之前的低频访问存储类型。
  - 同城冗余：采用多可用区（AZ）机制，将用户的数据分散存放在同一地域（Region）的3个可用区。当某个可用区不可用时，仍然能够保障数据的正常访问。
- 关键特性
  - 99.9999999999%（12个9）的数据可靠性
  - 按照99.995%服务可用性设计
  - 支持实时访问
  - 支持HTTPS加密传输
  - 支持图片处理
  - 有最短存储时间和最小计量空间

归档存储类型（Archive）

OSS归档存储类型在三种存储类型中单价最低，适合需要长期保存（建议半年以上）的归档数据，在存储周期内极少被访问，数据进入到可读取状态需要1分钟的解冻时间。适合需要长期保存的档案数据、医疗影像、科学资料、影视素材。归档存储类型的Object有最短存储时间，存储时间短于60天的Object提前删除会产生一定费用。归档类型存储Object有最小计量空间，Object大小低于64KB，会按照64KB计算存储空间。数据获取会产生数据取回费用。

关键特性：

- 99.999999999%（11个9）的数据可靠性
- 按照99.99%服务可用性设计
- 已经存储的数据从冷冻状态恢复到可读取状态需要1分钟的等待时间
- 支持HTTPS加密传输
- 支持图片处理，但需要先解冻
- 有最短存储时间和最小计量空间

存储类型对比

对比指标	标准存储类型	低频访问存储类型	归档存储类型
支持的冗余类型	本地冗余和同城冗余	本地冗余和同城冗余	-

对比指标	标准存储类型	低频访问存储类型	归档存储类型
数据可靠性	99.9999999999%（12个9）	99.9999999999%（12个9）	99.999999999%（11个9）
服务设计的可用性	99.995%	99.995%	99.99%（数据解冻之后）
对象最小计量大小	按照对象实际大小计算	64KB	64KB
最少存储时间	无最短存储时间要求	30天	60天
数据取回费用	不收取数据取回费用	按实际获取的数据量收取，单位GB	按实际解冻的数据量收取，单位GB
数据访问特点	实时访问ms延迟	实时访问ms延迟	数据需要先解冻，解冻完成后才能读取，解冻时间需要1分钟
图片处理	支持	支持	支持，但需要先解冻

说明:

“数据取回费用”中的数据是从底层分布式存储系统读取的数据量，在公网传输的数据量会计入到流出流量的计费项中。

存储类型支持的API

API	标准存储类型	低频访问存储类型	归档存储类型
Bucket创建、删除、查询			
PutBucket	支持	支持	支持
GetBucket	支持	支持	支持
DeleteBucket	支持	支持	支持
Bucket ACL设置相关			
PutBucketAcl	支持	支持	支持
GetBucketAcl	支持	支持	支持
Bucket日志功能	支持	支持	支持
PutBucketLogging	支持	支持	支持
GetBucketLogging	支持	支持	支持
Bucket缺省静态页面设置			
PutBucketWebsite	支持	支持	不支持
GetBucketWebsite	支持	支持	不支持

API	标准存储类型	低频访问存储类型	归档存储类型
Bucket Referer防盗链	支持	支持	支持
PutBucketReferer	支持	支持	支持
GetBucketReferer	支持	支持	支持
Bucket生命周期			
PutBucketLifecycle	支持	支持	支持，仅支持数据回收
GetBucketLifecycle	支持	支持	支持
DeleteBucketLifecycle	支持	支持	支持
Bucket跨区域复制	支持	支持	支持，仅复制已解冻的对象
PutBucketReplication	支持	支持	支持
Bucket跨域设置			
PutBucketcors	支持	支持	支持
GetBucketcors	支持	支持	支持
DeleteBucketcors	支持	支持	支持
Object操作			
PutObject	支持	支持	支持
PutObjectACL	支持	支持	支持
GetObject	支持	支持	支持，需要先解冻
GetObjectACL	支持	支持	支持
GetObjectMeta	支持	支持	支持
HeadObject	支持	支持	支持
CopyObject	支持	支持	支持
OptionObject	支持	支持	支持
DeleteObject	支持	支持	支持
DeleteMultipleObjects	支持	支持	支持
PostObject	支持	支持	支持
PutSymlink	支持	支持	支持

API	标准存储类型	低频访问存储类型	归档存储类型
GetSymlink	支持	支持	支持
RestoreObject	不支持	不支持	支持
Multipart操作			
InitiateMultipartUpload	支持	支持	支持
UploadPart	支持	支持	支持
UploadPartCopy	支持	支持	支持
CompleteMultipartUpload	支持	支持	支持
AbortMultipartUpload	支持	支持	支持
ListMultipartUpload	支持	支持	支持
ListParts	支持	支持	支持
图片处理	支持	支持	支持

4.2 存储类型转换

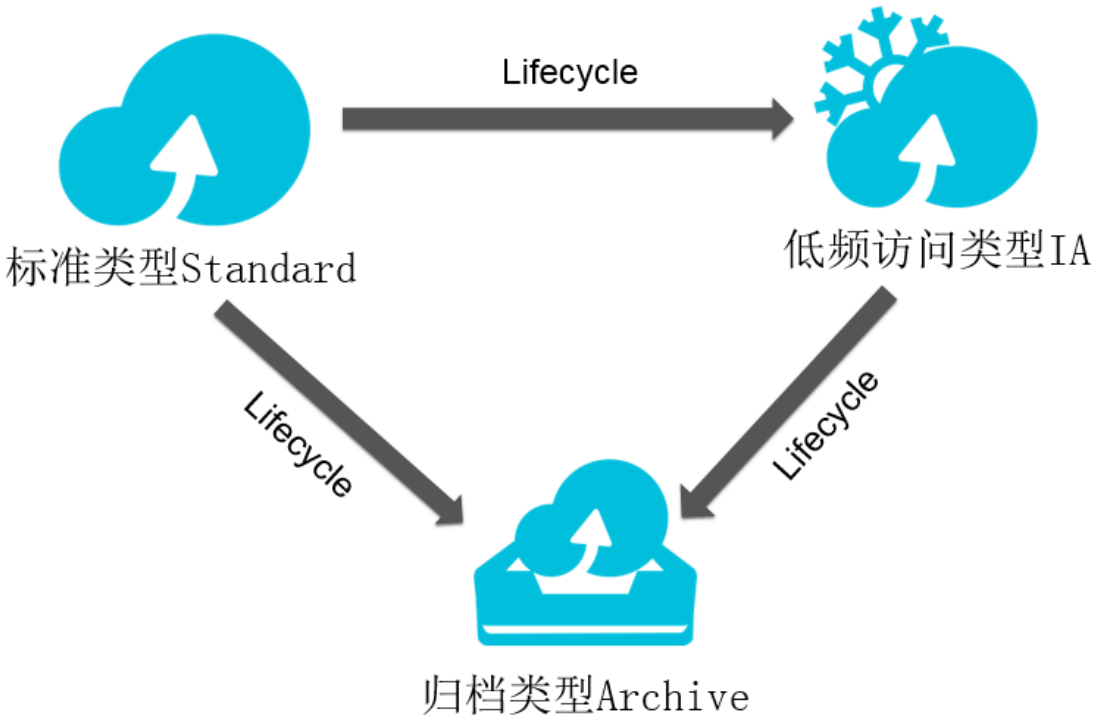
本文主要介绍标准存储类型、低频访问存储类型及归档存储类型之间的相互转换。

通过生命周期设置存储类型自动转换

OSS支持三种存储类型：标准类型、低频访问类型、归档类型。

OSS生命周期管理（Lifecycle）提供Object Transition机制，支持以下存储类型的自动转换：

- 标准类型转换为低频访问类型
- 标准类型转换为归档类型
- 低频访问类型转换为归档类型



例如，对一个Bucket里特定前缀的Object配置Lifecycle策略如下：

- **存储30天后，自动转换到低频访问类型。**
- **180天后，自动转换到归档类型。**
- **超过360天后，文件会被自动删除。**

您可以通过控制台完成以上Lifecycle策略的配置。具体操作请参见设置生命周期。

说明:

如果一个Bucket同时配置了Object保留指定天数后转换到低频访问、Object保留指定天数后转换到归档冷备和Object保留指定天数后删除，各规则设置的天数必须满足以下规则：

转换到低频的天数 < 转换到归档的天数 < 指定天数后删除

费用说明

Object类型转换后，会按照转换后的存储类型的存储单价计算存储费用。低频访问类型和归档存储类型需要特别注意：

- 最小计量空间

  对于小于64KB的Object，会按照64KB计算空间大小。

- 最短存储周期

  存储的数据需要至少保存30天，删除短于30天的文件，会收取提前删除费用。

- 归档类型的Restore时间

  归档类型Object恢复到可读取状态（Restore）需要1分钟的解冻时间。所以如果业务场景上需要实时读取，建议只转换成低频访问存储类型，而不应转换成归档存储类型，避免转换成归档类型后，数据无法实时读取。

- 数据获取费用

  低频访问类型和归档类型会收取数据获取费用，与流出流量是独立计费项，如果每个Object平均访问频率高于每月1次，不建议转换成低频访问或者归档类型。

其他方式的存储类型转换

您可以手动将归档类型重新转换为标准类型或低频类型，低频类型转换为标准类型。手动修改文件存储类型实际是通过覆写操作，将文件修改为指定的存储类型。所以，若修改的文件是低频访问或归档存储类型，且存储未满指定天数的，会产生数据提前删除费用，详情请参见计量项和计费项。

对于归档类型的Object，需要先执行Restore操作，解冻成可读取状态后，才可以修改存储类型。详情请参见#unique_27。

4.3 创建和使用归档存储类型

OSS提供三种存储类型，本文介绍归档存储类型（Archive）的存储空间的创建与使用。

创建归档存储类型的存储空间

您可以通过控制台、API/SDK和命令行工具创建归档存储类型的存储空间。

- **通过控制台创建**

  通过控制台创建归档存储类型的存储空间，存储类型选择归档，如下图所示。

- **通过API/SDK创建**

  以Java SDK为例：

```
OSSClient ossClient = new OSSClient(endpoint, accessKeyId,
accessKeySecret);
```

```
CreateBucketRequest createBucketRequest=new CreateBucketRequest(
bucketName);
// 设置bucket权限为公共读，默认是私有读写 createBucketRequest.setCannedA
CL(CannedAccessControlList.PublicRead);
// 设置bucket存储类型为归档类型，默认是标准类型
createBucketRequest.setStorageClass(StorageClass.Archive);
ossClient.createBucket(createBucketRequest);
```

createBucketRequest.setStorageClass(StorageClass.Archive);即设置创建的存储空间的存储类型为归档存储类型。

- 通过OSS命令行工具创建

  以OSSUtil为例：

  ```
  ./ossutil mb oss://[bucket name] --storage-class=Archive
  ```

  [bucket name]为需要创建的存储空间名称。指定--storage-class的参数为Archive，用来创建归档存储类型的存储空间。

使用归档存储类型

- 上传数据

  归档存储类型存储空间支持PutObject和MultipartUpload两种上传方式，不支持AppendObject。基于PutObject和MultipartUpload开发的上传应用可以直接使用归档存储类型。

- 下载数据

  归档存储类型的数据读取方式与标准存储类型和低频访问类型的数据读取方式有所区别。归档类型数据在读取前需要先执行restore操作解冻到可读取状态，解冻过程需要1分钟时间。

  归档文件的状态变换过程如下：

  1. 归档类型的文件初始时处于冷冻状态。
  2. 提交解冻（restore）操作后，服务端执行解冻，文件处于解冻中状态。
  3. 完成解冻后，可以读取文件。
  4. 解冻状态默认持续1天，最多延长7天，之后文件又回到冷冻状态。

解冻方式有如下几种：

- **使用控制台解冻**

对需要读取的文件，执行解冻操作，解冻过程预计花费1分钟。期间可以查询到object处于解冻中状态。

- **使用API/SDK解冻**

  以Java SDK举例，调用restoreObject方法进行object解冻：

```
ObjectMetadata objectMetadata = ossClient.getObjectMetadata(
bucketName, key);
// check whether the object is archive class
StorageClass storageClass = objectMetadata.getObjectStorageClass();
if (storageClass == StorageClass.Archive) {
    // restore object
    ossClient.restoreObject(bucketName, key);
```

```
    // wait for restore completed
    do {
        Thread.sleep(1000);
        objectMetadata = ossClient.getObjectMetadata(bucketName, key
);
    } while (!objectMetadata.isRestoreCompleted());
}
// get restored object
OSSObject ossObject = ossClient.getObject(bucketName, key);
ossObject.getObjectContent().close();
```

- **使用OSS命令行工具解冻**

  以OSSUtil为例：

  ```
  ./ossutil restore oss://[Bucket name]/[Object name]
  ```

  `[Bucket name]`**和**`[Object name]`**为需要做解冻操作的bucket和object名称。**

5 接入OSS

5.1 快速开始

本文主要介绍如何快速使用OSS，如创建Bucket、上传文件、下载文件等。

1. 登录OSS管理控制台，开通OSS。
2. 创建一个Bucket。
3. 上传和下载文件。

具体请参见开始使用阿里云 OSS。

快速了解OSS的上传下载

开始使用SDK之前，请先参考开发人员指南关于上传下载文件的功能介绍。

OSS是使用RESTful API来操作的，所有的请求都是标准的HTTP请求。

- OSS提供了多种类型的上传文件的方法，如使用单次PUT请求完成的简单上传，使用网页表单直接上传的表单上传，用于大文件上传的分片上传，以及适用于视频监控等领域的追加上传。
- 同样也可以使用多种方法从OSS下载文件，如简单下载和下载大文件的断点续传下载。

基于SDK快速开始

1. 开通OSS后，从控制台上获取AccessKeyId和AccessKeySecret。
2. 下载各种开发语言SDK。
3. 根据SDK的文档描述，完成上传、下载文件等操作。

具体请参见SDK文档。

5.2 基于OSS的移动开发

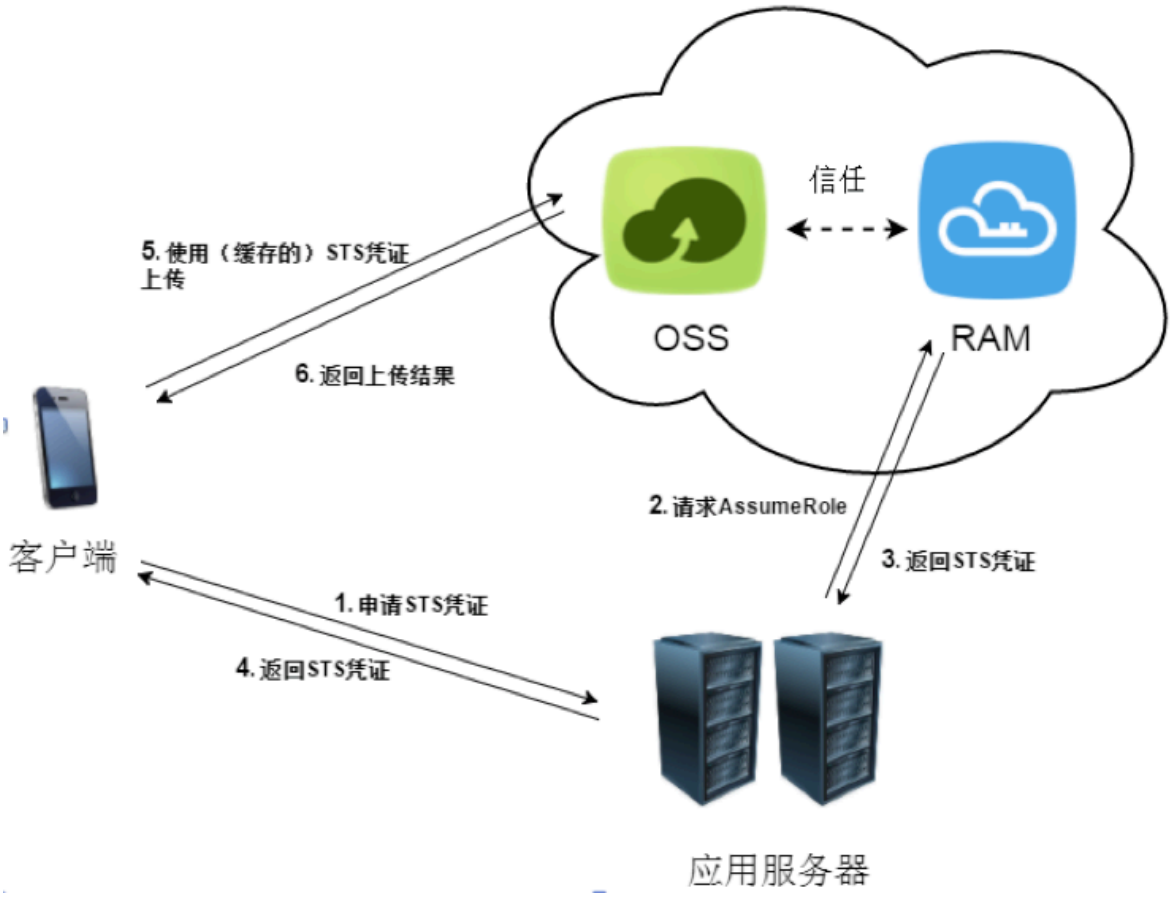
开发架构图

典型的基于OSS的移动开发有四个组件：

- OSS：提供上传、下载、上传回调等功能。
- 开发者的移动客户端（app或者网页应用），简称客户端：通过开发者提供的服务，间接使用OSS。
- 应用服务器：客户端交互的服务器，也是开发者的业务服务器。
- 阿里云STS：颁发临时凭证。

开发业务流程

- 临时凭证授权的上传



具体步骤如下：

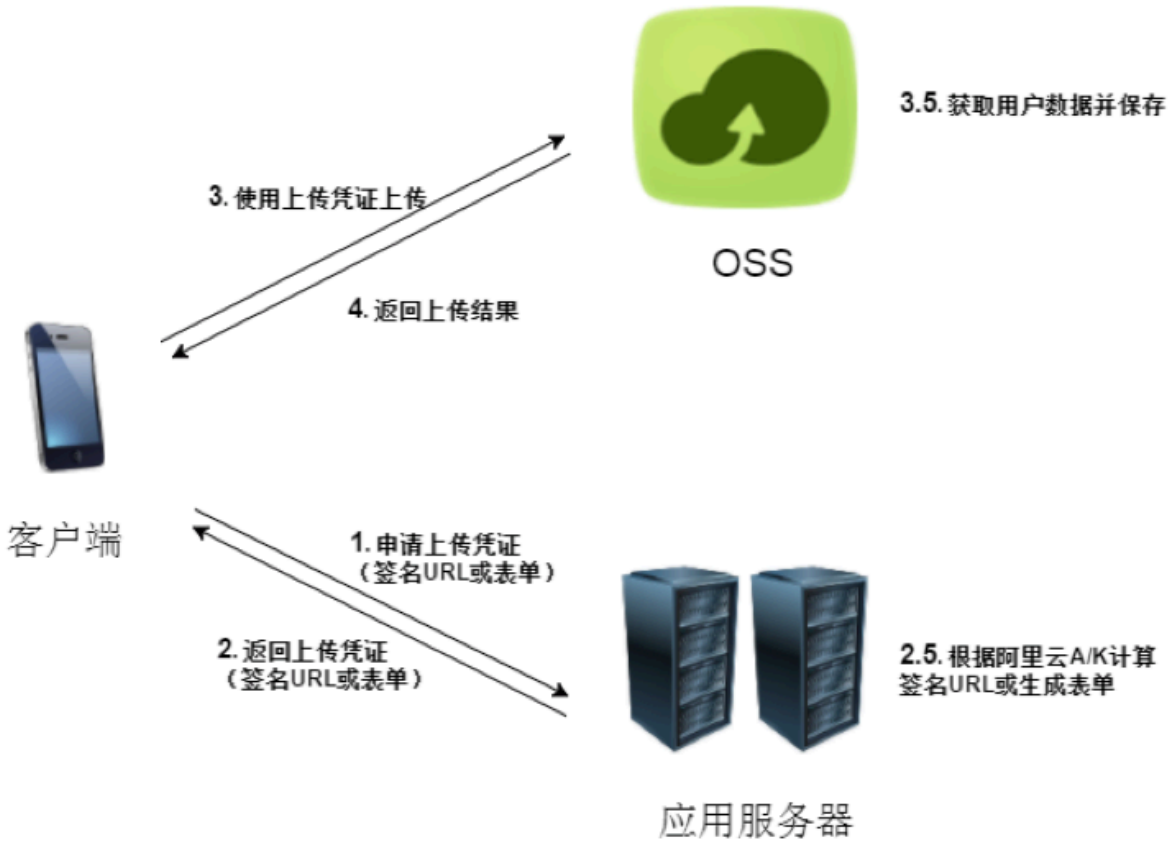
1. 客户端向应用服务器发出上传文件到OSS的请求。
2. 应用服务器向STS服务器请求一次，获取临时凭证。
3. 应用服务器回复客户端，将临时凭证返回给客户端。
4. 客户端获取上传到OSS的授权（STS的AccessKey以及Token），调用OSS提供的移动端SDK上传。
5. 客户端成功上传数据到OSS。如果没有设置回调，则流程结束。如果设置了回调功能，OSS会调用相关的接口。

> **说明：**
>
> - 客户端不需要每次都向应用服务器请求授权，在第一次授权完成之后可以缓存STS返回的临时凭证直到超过失效时间。

- STS提供了强大的权限控制功能，可以将客户端的访问权限限制到Object级别，从而实现不同客户端在OSS端上传的Object的完全隔离，极大提高了安全性。

更多信息请参见授权给第三方上传。

· 签名URL授权的上传和表单上传



具体步骤如下：

1. 客户端向应用服务器发出上传文件到OSS的请求。
2. 应用服务器回复客户端，将上传凭证（签名URL或者表单）返回给客户端。
3. 客户端获取上传到OSS的授权（签名URL或者表单），调用OSS提供的移动端SDK上传或者直接表单上传。
4. 客户端成功上传数据到OSS。如果没有设置回调，则流程结束。如果设置了回调功能，OSS会调用相关的接口。

更多信息请参见授权给第三方上传。

- 临时凭证授权的下载

  跟临时凭证授权的上传过程类似：

  1. 客户端向应用服务器发出下载OSS文件的请求。
  2. 应用服务器向STS服务器请求一次，获取临时凭证。
  3. 应用服务器回复客户端，将临时凭证返回给客户端。
  4. 客户端获取下载OSS文件的授权（STS的AccessKey以及Token），调用OSS提供的移动端SDK下载。
  5. 客户端成功从OSS下载文件。

  > 说明：
  > - 和上传类似，客户端对于临时凭证也可以进行缓存从而提高访问速度。
  > - STS同样也提供了精确到Object的下载权限控制，和上传权限控制结合在一起可以实现各移动端在OSS上存储空间的完全隔离。

- 签名URL授权的下载

  跟签名URL授权的上传类似：

  1. 客户端向应用服务器发出下载OSS文件的请求。
  2. 应用服务器回复客户端，将签名URL返回给客户端。
  3. 客户端获取下载OSS文件的授权（签名URL），调用OSS提供的移动端SDK下载。
  4. 客户端成功从OSS下载文件。

  > 说明：
  > 客户端不能存储开发者的AccessKey，只能获取应用服务器签名的URL或者是通过STS颁发的临时凭证（也就是STS的AccessKey和Token）。

最佳实践

- 快速搭建移动应用直传服务
- 快速搭建移动应用上传回调服务
- 权限控制
- RAM和STS使用指南

功能使用参考

- Android SDK上传文件
- iOS SDK上传文件

6 合规保留策略

对象存储OSS支持WORM特性，允许用户以“不可删除、不可篡改”方式保存和使用数据，符合美国证券交易委员会（SEC）和金融业监管局（FINRA）的合规要求。

说明:

- 合规保留策略正在公测中，您可以联系售后技术支持申请试用此功能。
- OSS目前仅支持针对Bucket级别设置合规保留策略。
- OSS是目前国内唯一通过Cohasset Associates审计认证的云服务，可满足严格的电子记录保留要求，例如EC Rule 17a-4（f）、FINRA 4511、 CFTC 1.31等合规要求。详情请参见 OSS Cohasset Assessment Report。

OSS提供强合规策略，用户可针对存储空间（Bucket）设置基于时间的合规保留策略。当策略锁定后，用户可以在Bucket中上传和读取文件（Object），但是在Object的保留时间到期之前，任何用户都无法删除Object和策略。Object的保留时间到期后，才可以删除Object。OSS支持的WORM特性，适用于金融、保险、医疗、证券等行业。您可以基于OSS搭建“云上数据合规存储空间”。

说明:

Bucket内的Object在合规保留策略生效期间，可通过设置生命周期规则进行存储类型转化，在保证合规性的前提下，降低存储成本。

操作方式

控制台：设置合规保留策略

规则说明

OSS允许添加一条基于时间的合规保留策略，保护周期为1天到70年。

假设您在2013年6月1日创建一个名为examplebucket的Bucket，并且在不同时间上传了file1.txt、file2.txt、file3.txt三个Object。随后，在2014年7月1日创建了保护周期为5年的合规保留策略。有关这三个Object的具体上传时间以及对应的Object到期时间如下：

Object名称	上传时间	Object到期时间
file1.txt	2013年6月1日	2018年5月31日
file2.txt	2014年7月1日	2019年6月30日
file3.txt	2018年9月30日	2023年9月29日

- 生效规则

  当基于时间的合规保留策略创建后，该策略默认处于“InProgress”状态，且该状态的有效期为 24 小时。在有效期24小时内，此策略对应的Bucket资源处于保护状态。

  - 启动合规保留策略24小时内：若该策略未提交锁定，则Bucket所有者以及授权用户可以删除该策略；若该保留策略已提交锁定，则不允许删除该策略，且无法缩短策略保护周期，仅可以延长保护周期。
  - 启动合规保留策略24小时后：若超过24小时该保留策略未提交锁定，则该策略自动失效。
  - Bucket内的数据处于被保护状态时，若您尝试删除或修改这些数据，OSS API将返回`409 FileImmutable`的错误信息。

- 删除规则

  - 基于时间的合规保留策略是Bucket的一种Metadata属性。当删除某个Bucket时，该Bucket对应的合规保留策略以及访问策略也会被删除。因此当Bucket为空时，Bucket的所有者可以删除该Bucket，从而间接删除该Bucket的保留策略。
  - 启动保留策略24小时内，若该保留策略未提交锁定，则Bucket所有者以及授权用户可以删除该策略。
  - 若Bucket内有文件处于保护周期内，那么您将无法删除保留策略，同时也无法删除Bucket。

常见问题

- 合规保留策略的优势

  合规保留策略可提供数据合规存储，数据在合规保留策略保护周期内，任何用户都不能删除和修改。而通过RAM policy和Bucket Policy保护的数据，则存在被修改和删除可能。

- 什么情况下需要配置合规保留策略

  您需要长期存储且不允许修改或删除的重要数据，如医疗档案、技术文件、合同文书等，可以存放在指定的Bucket内，并通过开启合规保留策略保护您的重要数据。

- 是否支持针对Object设置合规保留策略

  目前仅支持针对Bucket设置保留策略，不支持针对目录以及单个对象设置合规保留策略。

- 如何删除已开启合规保留策略的Bucket

  - 若该Bucket内未存储文件，可以直接删除该Bucket。
  - 若该Bucket内已存储文件，且所有文件均已过了保护期，删除该Bucket会提示失败。此时，您可以先删除该Bucket内所有文件，再删除Bucket。
  - 若该Bucket内已存储文件，且还有文件处于保护期内，无法删除该Bucket。

- 如果OSS欠费，但仍有文件处于合规保留策略的保护期内，这些文件会被保留么

  在未付款的情况下，阿里云会根据您签署的合同条款和条件，应用对应的数据保留策略。

- 授权的子账号是否可以设置合规保留策略

  合规保留策略相关API接口已全部对外开放，并且相关API操作已支持接入RAM policy。通过RAM Policy授权的子账号可以通过控制台、API、SDK等方式创建、删除合规保留策略。

7 数据容灾

7.1 同城冗余存储

OSS采用多可用区（AZ）机制，将用户的数据分散存放在同一地域（Region）的3个可用区。当某个可用区不可用时，仍然能够保障数据的正常访问。OSS同城冗余存储能够提供99.9999999999%（12个9）的数据可靠性以及99.995%的服务设计可用性。

OSS的同城冗余存储能够提供机房级容灾能力。当断网、断电或者发生灾难事件导致某个机房不可用时，仍然能够确保继续提供强一致性的服务能力，整个故障切换过程用户无感知，业务不中断、数据不丢失，可以满足关键业务系统对于“恢复时间目标（RTO）”以及“恢复点目标（RPO）”等于0的强需求。

支持的存储类型

目前OSS的同城冗余存储支持标准存储类型、低频访问存储类型。这两种存储类型的各项对比指标详情如下：

对比指标	标准存储类型	低频访问存储类型
数据可靠性	99.9999999999%	99.9999999999%
服务设计的可用性	99.995%	99.995%
对象最小计量大小	按照对象实际大小计算	64KB
最短存储时间	无最短存储时间要求	30天
数据取回费用	不收取数据取回费用	按实际获取的数据收取，单位GB
数据访问	实时访问毫秒延迟	实时访问毫秒延迟
图片处理	支持	支持

修改属性

目前仅支持在创建Bucket时，设置同城冗余存储属性。不支持针对现有的Bucket开启同城冗余存储属性。

对于现有的Bucket，可以利用迁移工具，如OSSImport、SDK等方式，将原有Bucket中的数据复制到开启了OSS同城冗余属性的Bucket中。

7.2 管理跨区域复制

本文主要介绍跨区域复制的使用场景及使用限制。

跨区域复制（Bucket Cross-Region Replication）是跨不同OSS数据中心（地域）的Bucket自动、异步复制Object，它会将Object的创建、更新和删除等操作从源存储空间复制到不同区域的目标存储空间。该功能能够很好的提供Bucket跨区域容灾或满足用户数据复制的需求。目标Bucket中的对象是源Bucket中对象的精确副本，它们具有相同的对象名、元数据以及内容，例如创建时间、拥有者、用户定义的元数据、Object ACL、对象内容等。

说明:

- 开启跨区域复制功能后，主从两个区域的Bucket在复制文件时会产生跨区域间的数据流量，OSS会收取跨区域复制流量费用。每同步1个Object，OSS会累计计算请求次数并进行按量计费。
- 跨区域复制功能支持的地域及费用请参见对象存储OSS产品定价中跨区域复制费用部分。

使用场景

您可能基于各种原因对Bucket配置跨区域复制，这些原因包括：

- 合规性要求：虽然 OSS 默认对每个存储的对象在物理盘上有多份副本，但合规性要求所规定的数据需要跨一定距离保存一份副本。通过跨区域复制，可以在远距离的 OSS 数据中心之间复制数据以满足这些合规性要求。
- 最大限度减少延迟：客户处于两个地理位置。为了最大限度缩短访问对象时的延迟，可以在地理位置与用户较近的 OSS 数据中心中维护对象副本。
- 数据备份与容灾：您对数据的安全性和可用性有极高的要求，对所有写入的数据，都希望在另一个数据中心显式地维护一份副本，以备发生特大灾难，如地震、海啸等导致一个OSS数据中心损毁时，还能启用另一个OSS数据中心的备份数据。
- 数据复制：由于业务原因，需要将数据从OSS的一个数据中心迁移到另一个数据中心。
- 操作原因：您在两个不同数据中心中具有分析同一组对象的计算集群。您可能选择在这些区域中维护对象副本。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
Java SDK	丰富、完整的各类语言SDK demo

使用说明

跨区域复制支持异名Bucket的同步。如果您拥有的两个Bucket分属不同地域，可以通过配置同步将源Bucket的数据实时同步到目标Bucket。目前支持以下特性：

- 实时同步数据：数据实时同步，对于数据的增加、删除、修改能够实时监控同步到目标地域Bucket。对于2MB文件，能够做到分钟级别信息同步，保证两边数据的最终一致。
- 历史数据迁移：迁移历史数据，让原来Bucket中历史数据也能进行同步，形成相同的两份数据。
- 实时获取同步进度：能够针对实时同步数据展示最近同步的时间节点，针对历史数据的迁移展示迁移的百分比。
- 便捷配置：OSS控制台提供便捷的界面管理配置。
- 互通同步：您可以配置Bucket A到Bucket B的同步，然后配置Bucket B到Bucket A的同步，实现两个Bucket之间的互通同步。

限制说明

- 对于处于同步状态的两个Bucket，由于您可以同时操作这两个Bucket，但源Bucket复制过去的Object可能会覆盖目标Bucket中同名的Object，使用中请注意。
- 由于跨区域复制采用异步复制，数据复制到目标Bucket需要一定的时间，通常几分钟到几小时不等，取决于数据的大小。
- 开启跨区域同步的条件是同步的两个Bucket没有开启与其他Bucket的同步配置，同时不能被其他Bucket同步。例如，Bucket A 开启了到 Bucket B 的同步，那么您就不能再为 Bucket A 开启到 Bucket C 的同步，除非先删除 Bucket A到Bucket B 的同步配置。同理，若 Bucket A 开启了到 Bucket B 的同步，这时候再开启 Bucket C 到 Bucket B 的同步也是不允许的。
- 开启数据同步的两个Bucket必须分属两个地域，同地域的Bucket之间不能进行数据同步。

8 存储空间（Bucket）

8.1 创建存储空间

在上传文件（Object）到 OSS 之前，您需要使用 OSS API 中的 PutBucket 接口来创建一个用于存储文件的存储空间（Bucket），存储空间具有各种配置属性，包括地域、访问权限以及其他元数据。

说明:

创建存储空间的 API 接口的详细信息请参见 #unique_53。

操作方式

操作方式	说明
控制台	Web 应用程序，直观易用
图形化工具 ossbrowser	图形化工具，易操作
命令行工具 ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言 SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	
Node.js SDK	
Ruby SDK	

使用限制

- 同一用户创建的存储空间总数不能超过 30 个。
- 每个存储空间的名字全局唯一，否则会创建失败。
- 存储空间的名称需要符合命名规范。
- 存储空间一旦创建成功，名称和所处地域不能修改。

权限控制

您可以在创建存储空间的时候设置相应的存储空间权限（ACL），也可以在创建之后修改 ACL。如果不设置 ACL，默认值为私有。更多信息，请参见设置存储空间读写权限。

更多参考

- #unique_68
- #unique_69
- #unique_70
- #unique_71

8.2 设置存储空间读写权限（ACL）

您可以在创建存储空间（Bucket）时设置存储空间的访问权限（ACL），也可以在创建Bucket后根据自己的业务需求修改存储空间的ACL，该操作只有存储空间的拥有者可以执行。

存储空间有三种访问权限：

权限值	中文名称	权限对访问者的限制
public-read-write	公共读写	任何人（包括匿名访问者）都可以对该存储空间内文件进行读写操作。 警告： 互联网上任何用户都可以对该 Bucket 内的文件进行访问，并且向该 Bucket 写入数据。这有可能造成您数据的外泄以及费用激增，若被人恶意写入违法信息还可能会侵害您的合法权益。除特殊场景外，不建议您配置公共读写权限。
public-read	公共读，私有写	只有该存储空间的拥有者可以对该存储空间内的文件进行写操作，任何人（包括匿名访问者）都可以对该存储空间中的文件进行读操作。 警告： 互联网上任何用户都可以对该 Bucket 内文件进行访问，这有可能造成您数据的外泄以及费用激增，请谨慎操作。
private	私有读写	只有该存储空间的拥有者可以对该存储空间内的文件进行读写操作，其他人无法访问该存储空间内的文件。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

更多参考

- #unique_71
- #unique_74

8.3 获取存储空间地域信息

您可以通过OSS API的GetBucketLocation接口获取存储空间（Bucket）所属的地域，即数据中心的物理位置信息。

说明:

获取存储空间地域信息的API详情请参考GetBucketLocation。

您可以通过返回的Location字段查看存储空间所在的地域信息，例如：华东 1（杭州）的Location字段信息显示为`oss-cn-hangzhou`。请参见 OSS 提供的访问域名。

操作方式

操作方式	说明
控制台	进入控制台后在存储空间属性中直接显示地域信息
图形化工具ossbrowser	图形化工具，易操作

操作方式	说明
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

8.4 查看存储空间列表

存储空间创建之后，您可以通过OSS API的GetService接口获取存储空间列表信息。

说明:

查看存储空间列表的API接口详细信息请参见GetService。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

更多参考

- 创建Bucket

8.5 请求者付费模式

阿里云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。

说明:

请求者付费模式目前只在华南1（深圳）开放。

使用案例

- 共享大型数据集 (如邮政编码目录、参考数据、地理空间信息或网络爬取数据)。例如，研究机构提供公开数据集，希望所有客户都能访问该数据，但请求产生的流量费用和请求次数费用由请求者支付。配置步骤如下：

  1. 将Bucket开启请求者付费模式。详细配置步骤请参考设置请求者付费模式。
  2. 通过Bucket Policy，将该Bucket授权给您客户的阿里云RAM子账号。配置详情请参考使用Bucket Policy授权其他用户访问OSS资源。

- 将数据交付给您的客户或合作伙伴。例如，某公司需要将生产数据交付给他的合作伙伴，下载数据产生的流量费用和请求次数费用需要由合作伙伴支付。

  配置步骤如下：

  1. 将Bucket开启请求者付费模式。
  2. 将Bucket ACL设置为私有。
  3. 利用Bucket Policy，将该Bucket授权给您的合作伙伴的阿里云RAM子账号。配置详情请参考#unique_84。

  注意:

  您需要将Bucket授权给对方的RAM子账号，而不是将您账号下的RAM子账号的AK提供给对方。因为，当对方通过您账号下的RAM子账号访问时，请求者仍是您自身，则请求费用需要由您（请求者）付费。

请求方式说明

- 不允许匿名访问

  如果您在Bucket上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

  如果您在Bucket上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 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。

费用详解

请求者付费模式下，请求者支付流量费用和请求费用，Bucket拥有者支付存储费用。但是在以下情况下，请求会失败（返回HTTP 403错误），将对Bucket拥有者收取请求费用：

- 请求者未在请求中 (GET、HEAD 或 POST) 包含参数 x-oss-request-payer，或未在请求中将其作为参数 (REST) 。
- 请求身份验证失败。
- 请求是匿名请求 。

更多参考

- 控制台：设置请求者付费模式
- ossutil：访问开启“请求者付费模式”的Bucket

8.6 绑定自定义域名

您的文件上传到阿里云 OSS 的 Bucket 后，会自动生成该文件的访问地址，您可以使用此地址访问 Bucket 内的文件。若您希望通过自定义域名访问这些文件，需要将自定义域名绑定到文件所在的存储空间，并添加 CNAME 记录指向存储空间对应的外网域名。

操作方式

操作方式	说明
控制台	Web 应用程序，直观易用

操作方式	说明
PHP SDK	丰富、完整的各类语言 SDK demo
Node.js SDK	
Browser.js SDK	
Ruby SDK	

常见概念

绑定自定义域名，您需要了解以下概念：

- 用户域名、自定义域名、自有域名：您在域名服务商处购买的域名。
- OSS 访问域名或 Bucket 域名：OSS 为您的 Bucket 分配的的访问域名。您可以使用此域名访问您 Bucket 里的资源。如果您想使用您自己的用户域名访问 OSS Bucket，必须将您的用户域名绑定到 OSS 域名，即在云解析中添加CNAME记录。
- 阿里云 CDN 域名：阿里云 CDN 为您的用户域名分配的 CDN 加速域名。将您的用户域名绑定到 CDN 加速域名，即在云解析中添加 CNAME 记录，才可以加速访问 OSS Bucket 里的资源。
- CDN 缓存自动刷新：如果您更新了 OSS Bucket 里的 Object，而旧 Object 在 CDN 节点上的缓存没有到期，此时终端用户访问 Object 还是未更新的内容，则您需要手工去刷新缓存，这很麻烦。因此，OSS 服务提供了 CDN 缓存自动刷新功能，开启此动能后，您在 OSS Bucket 里做的所有更新都会自动刷新到 CDN 节点。具体操作请参见开启 CDN 缓存自动刷新。

绑定自定义域名应用场景

例如，用户 A 拥有一个域名为 `img.abc.com` 的网站，网站的一个图片链接为`http://img.abc.com/logo.png`。为方便后续管理，用户 A 想要访问网站中图片的请求转到 OSS，并且不想修改任何网页的代码，也就是对外链接不变。绑定自定义域名可以满足这个需求。流程如下：

1. 在 OSS 上创建一个名为 *abc-img* 的 Bucket，并将其网站上的图片上传至 Bucket。
2. 通过 OSS 控制台，将 `img.abc.com` 这个自定义的域名绑定在 *abc-img*。
3. 绑定成功之后，OSS 后台会将 `img.abc.com` 做一个映射到 *abc-img*。
4. 在自己的域名服务器上，添加一条 CNAME 规则，将 `img.abc.com` 映射成 `abc-img.oss-cn-hangzhou.aliyuncs.com`（即 *abc-img* 的 OSS 域名）。
5. `http://img.abc.com/logo.png` 请求到达 OSS 后，OSS 通过 `img.abc.com` 和 *abc-img* 的映射关系，将访问转到 *abc-img* 这个 Bucket。也就是说，对`http://img.abc.com/logo.png`的访问，实际上访问的是`http://abc-img.oss-cn-hangzhou.aliyuncs.com/logo.png`。

绑定自定义域名前后流程对比如下：

	绑定自定义域名前	绑定自定义域名后
流程对比	1. 访问 `http://img.abc.com/logo.png`。 2. DNS 解析到用户服务器 IP。 3. 访问用户服务器上的logo.png。	1. 访问 `http://img.abc.com/logo.png` 。 2. DNS 解析到 `abc-img.oss-cn-hangzhou.aliyuncs.com`。 3. 访问 OSS 上 *abc-img* 里的 *logo.png*。

更多参考

- 如果您希望设置 CDN 加速，可参考绑定 CDN 加速域名 。
- 如果您希望通过静态网页访问 OSS 资源，可参考设置静态网站托管和教程示例：使用自定义域名设置静态网站托管。
- 如果您希望使用 HTTPS 方式访问访问 OSS 服务，可参考#unique_95。

8.7 设置防盗链

防盗链是指调用 OSS API 的 PutBucketReferer 接口设置 Referer， 用以防止他人盗用 OSS 数据。

> 说明:
> 防盗链的 OSS API 接口详细信息请参见 PutBucketReferer。

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。

例如，对于一个名为 oss-example 的 Bucket，设置其 Referer 白名单为`https://www.aliyun.com/`。则所有 Referer 为`https://www.aliyun.com/`的请求才能访问 oss-example 这个 Bucket 中的 Object。

操作方式

操作方式	说明
控制台	Web 应用程序，直观易用
Java SDK	丰富、完整的各类语言 SDK demo

操作方式	说明
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

细节分析

- Referer 验证
  - 用户只有通过签名 URL 或者匿名访问 Object 时，才会做防盗链验证。请求的 Header 中有 `Authorization` 字段的，不会做防盗链验证。
  - Bucket 的三种权限（private，public-read，public-read-write）都会做防盗链验证。
- Referer 配置
  - 一个 Bucket 可以支持多个 Referer 参数。通过控制台设置时使用回车作为换行符分隔，通过 API 设置时使用英文逗号（,）分隔。
  - Referer 参数支持通配符星号（*）和问号（?）。
- Referer 效果
  - Referer 白名单为空时，不会检查 Referer 字段是否为空（否则所有的请求都会被拒绝）。
  - 如果 Referer 白名单不为空，且不允许 Referer 字段为空，则只有 Referer 属于白名单的请求被允许，其他请求（包括 Referer 为空的请求）会被拒绝。
  - 如果白名单不为空，但允许 Referer 字段为空，则 Referer 为空的请求和符合白名单的请求会被允许，其他请求都会被拒绝。

通配符详解

- 星号（*）：可以使用星号代替 0 个或多个字符。如果正在查找以 AEW 开头的一个文件，但不记得文件名其余部分，可以输入 `AEW*`，查找以 AEW 开头的所有文件类型的文件，如 *`AEWT.txt#AEWU.EXE#AEWI.dll`* 等。要缩小范围可以输入`AEW*.txt`，查找以 AEW 开头并以 .txt 为扩展名的文件，如 *`AEWIP.txt#AEWDF.txt`*。
- 问号（?）：可以使用问号代替一个字符。例如输入`love?`查找以 love 开头的一个字符结尾的文件，如 lovey、lovei 等。要缩小范围可以输入 `love?.doc`，查找以 love 开头的一个字符结尾并以 .doc 为扩展名的文件，如 *`lovey.doc#loveh.doc`*。

更多参考

- 防盗链原理介绍请参考#unique_107。
- 防盗链功能常见问题请参考OSS 防盗链（Referer）配置及错误排除。

8.8 设置跨域资源共享

跨域资源共享（Cross-Origin Resource Sharing），简称 CORS，是 HTML5 提供的标准跨域解决方案，OSS 支持 CORS 标准来实现跨域访问。您可以通过 OSS API 的 PutBucketcors 接口设置跨域资源共享。

说明:

- 跨域访问相关 API 接口详细信息请参考跨域访问。
- 具体的 CORS 规则可以参考W3C CORS 规范。

跨域访问，或者说 JavaScript 的跨域访问问题，是浏览器出于安全考虑而设置的一个限制，即同源策略。当 A、B 两个网站属于不同域的时候，来自于 A 网站页面中的 JavaScript 代码希望访问 B 网站时，浏览器会拒绝该访问。

操作方式

操作方式	说明
控制台	Web 应用程序，直观易用
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	

使用场景

在实际应用中，经常会有跨域访问的需求。

例如，用户的网站 www.a.com，后端使用了 OSS。在网页中提供了使用JavaScript实现的上传功能，但是在该页面中，只能向 www.a.com 发送请求，向其他网站发送的请求都会被浏览器拒绝。这样就导致用户上传的数据必须从 www.a.com 中转。如果设置了跨域访问的话，用户就可以直接上传到OSS而无需从 www.a.com 中转。

跨域资源共享的实现

实现如下：

1. CORS 通过 HTTP 请求中附带 Origin 的 Header 来表明自己来源域，比如上面那个例子，Origin 的 Header 就是 www.a.com。
2. 服务器端接收到这个请求之后，会根据一定的规则判断是否允许该来源域的请求。如果允许，服务器在返回的响应中会附带上 Access-Control-Allow-Origin 这个 Header，内容为 www.a.com 来表示允许该次跨域访问。如果服务器允许所有的跨域请求，将 Access-Control-Allow-Origin 的 Header 设置为星号（*）即可。
3. 浏览器根据是否返回了对应的 Header 来决定该跨域请求是否成功，如果没有附加对应的 Header，浏览器将会拦截该请求。如果是非简单请求，浏览器会先发送一个 OPTIONS 请求来获取服务器的 CORS 配置，如果服务器不支持接下来的操作，浏览器也会拦截接下来的请求。

OSS 提供了 CORS 规则的配置，从而根据需求允许或者拒绝相应的跨域请求。该规则是配置在 Bucket 级别的。详情可以参考 PutBucketCORS。

细节分析

- CORS 相关的 Header 附加等都是浏览器自动完成的，用户不需要有任何额外的操作。CORS操作也只在浏览器环境下有意义。
- CORS 请求的通过与否和 OSS 的身份验证是完全独立的，即 OSS 的 CORS 规则仅仅是用来决定是否附加 CORS 相关的Header的一个规则。是否拦截该请求完全由浏览器决定。
- 使用跨域请求的时候需要关注浏览器是否开启了 Cache 功能。当运行在同一个浏览器上分别来源于 www.a.com 和 www.b.com 的两个页面都同时请求同一个跨域资源的时候，如果 www.a.com 的请求先到达服务器，服务器将资源带上 Access-Control-Allow-Origin 的 Header 为 www.a.com 返回给用户。这个时候 www.b.com 又发起了请求，浏览器会将 Cache 的上一次请求返回给用户，此时 Header 的内容和 CORS 的要求不匹配，就会导致后面的请求失败。

常见问题

功能使用常见问题请参考OSS 跨域资源共享（CORS）错误及排除。

8.9 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。

Bucket标签使用一组键值对（Key-Value）对标记存储空间，您可以通过Bucket标签标记不同用途的Bucket，并进行分类管理。

- 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。

操作方式

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
Go SDK	

使用说明

添加Bucket标签后，您可以对拥有相同标签的Bucket进行批量管理，例如列举拥有相同标签的Bucket、授权RAM用户管理拥有相同标签的Bucket等。

- 列举所有带指定标签的Bucket

  您可以在列举Bucket时仅列举带指定标签的Bucket，详情请参见如下SDK Demo：

  - Java SDK
  - Python SDK
  - Go SDK

- 授权RAM用户管理拥有指定标签的Bucket

  当您的Bucket较多时，您可以用Bucket标签对您的Bucket进行分类，并通过RAM策略授权指定用户可以管理拥有指定标签的Bucket。例如授权用户A可以列举所有拥有keytest=valuetest标签的Bucket，RAM策略如下：

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "oss:ListBuckets"
            ],
            "Resource": [
                "acs:oss:*:1932487924256138:*"
            ],
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "oss:BucketTag/keytest": "valuetest"
```

```
                    }
                }
            }
        ]
}
```

8.10 删除存储空间

您可以通过 OSS API 的 DeleteBucket 接口删除您创建的存储空间。

说明:

删除存储空间的 API 详细信息可参考DeleteBucket

如果存储空间不为空（存储空间中有文件或者是尚未完成的分片上传），则存储空间无法删除，必须删除存储空间中的所有文件和未完成的分片文件后，存储空间才能成功删除。如果想删除存储空间内部所有的文件，推荐使用生命周期管理。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	
Node.js SDK	
Ruby SDK	

9 对象/文件（Object）

9.1 上传文件（Object）

9.1.1 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）。简单上传适用于一次HTTP请求交互即可完成上传的场景，比如小文件（小于5GB）的上传。

说明:

- 简单上传的API接口的详细信息请参见PutObject。
- 大文件（大于5GB）的上传请使用断点续传上传。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	
Node.js SDK	
Browser.js SDK	
Ruby SDK	

上传限制

- 大小限制：Object的大小不能超过5GB。

- 命名限制：
  - 使用UTF-8编码。
  - 长度必须在1–1023字节之间。
  - 不能以正斜线（/）或者反斜线（\）字符开头。

上传文件时设置Object Meta

在使用简单上传的情况下，可以携带Object Meta信息对Object进行描述，比如可以设定Content-Type等标准HTTP头，也可以设定自定义信息。详情请参见设置文件元信息。

上传的安全及授权

为了防止第三方未经授权往您的Bucket里上传数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。

为了授权给第三方上传，OSS还提供了账号级别的授权。详情请参见授权给第三方上传。

上传后续操作

- 在文件上传到OSS上后，您可以通过上传回调来向指定的应用服务器发起回调请求，进行下一步操作。
- 如果上传的是图片，您还可以进行图片处理。
- 如果上传是音频或者视频文件，您还可以进行媒体处理。

9.1.2 表单上传

表单上传是指使用OSS API中的PostObject请求来完成Object的上传，上传的Object不能超过5GB。

> 说明：
> 表单上传的API接口详细信息请参见PostObject。

适用场景

表单上传非常适合嵌入在HTML网页中来上传Object，比较常见的场景是网站应用，以招聘网站为例：

	不使用表单上传	表单上传
流程对比	1. 网站用户上传简历。 2. 网站服务器回应上传页面。 3. 简历被上传到网站服务器。 4. 网站服务器再将简历上传到OSS。	1. 网站用户上传简历。 2. 网站服务器回应上传页面。 3. 简历上传到OSS。

- 从流程上来说，使用表单上传，少了一步转发流程，更加方便。
- 从架构上来说，原来的上传都统一走网站服务器，上传量过大时，需要扩容网站服务器。采用表单上传后，直接从客户端上传数据到OSS，上传量过大时，压力都在OSS上，由OSS来保障服务质量。

SDK demo

请参见Java SDK。

上传限制

- 大小限制：使用表单上传时，Object不能超过5GB。
- 命名限制：
  - 使用UTF-8编码。
  - 长度必须在1–1023字节之间。
  - 不能以正斜线（/）或者反斜线（\）字符开头。

流程解析

1. 构建一个Post Policy。

   Post请求的Policy表单域用于验证请求的合法性。例如可以指定上传的大小，可以指定上传的Object名字等，上传成功后客户端跳转到的URL，上传成功后客户端收到的状态码。具体请参考Post Policy。

   例如如下Policy，网站用户能上传的过期时间是2115-01-27T10:56:19Z（这里为了测试成功写的过期时间很长，实际使用中不建议这样设置），能上传的文件最大为104857600字节。

   ```
   以Python代码为例子，Policy是json格式的字符串。
    policy="{\"expiration\":\"2115-01-27T10:56:19Z\",\"conditions\":
   [[\"content-length-range\", 0, 104857600]]}"
   ```

2. 将Policy字符串进行base64编码。
3. 用OSS的AccessKeySecret对base64编码后的Policy进行签名。
4. 构建上传的HTML页面。
5. 打开HTML页面，选择文件上传。

完整Python代码示例：

```
#coding=utf8
import md5
import hashlib
import base64
import hmac
from optparse import OptionParser
def convert_base64(input):
    return base64.b64encode(input)
```

```
def get_sign_policy(key, policy):
    return base64.b64encode(hmac.new(key, policy, hashlib.sha1).digest
())
def get_form(bucket, endpoint, access_key_id, access_key_secret, out):
    #1 构建一个Post Policy
    policy="{\"expiration\":\"2115-01-27T10:56:19Z\",\"conditions\":
[[\"content-length-range\", 0, 1048576]]}"
    print("policy: %s" % policy)
    #2 将Policy字符串进行base64编码
    base64policy = convert_base64(policy)
    print("base64_encode_policy: %s" % base64policy)
    #3 用OSS的AccessKeySecret对编码后的Policy进行签名
    signature = get_sign_policy(access_key_secret, base64policy)
    #4 构建上传的HTML页面
    form = '''
    <html>
        <meta http-equiv=content-type content="text/html; charset=UTF-
8">
        <head><title>OSS表单上传(PostObject)</title></head>
        <body>
            <form  action="http://%s.%s" method="post" enctype="
multipart/form-data">
                <input type="text" name="OSSAccessKeyId" value="%s">
                <input type="text" name="policy" value="%s">
                <input type="text" name="Signature" value="%s">
                <input type="text" name="key" value="upload/${filename
}">
                <input type="text" name="success_action_redirect"
value="http://oss.aliyun.com">
                <input type="text" name="success_action_status" value
="201">
                <input name="file" type="file" id="file">
                <input name="submit" value="Upload" type="submit">
            </form>
        </body>
    </html>
    ''' % (bucket, endpoint, access_key_id, base64policy, signature)
    f = open(out, "wb")
    f.write(form)
    f.close()
    print("form is saved into %s" % out)
if __name__ == '__main__':
    parser = OptionParser()
    parser.add_option("", "--bucket", dest="bucket", help="specify ")
    parser.add_option("", "--endpoint", dest="endpoint", help="specify
")
    parser.add_option("", "--id", dest="id", help="access_key_id")
    parser.add_option("", "--key", dest="key", help="access_key_secret
")
    parser.add_option("", "--out", dest="out", help="out put form")
    (opts, args) = parser.parse_args()
    if opts.bucket and opts.endpoint and opts.id and opts.key and opts
.out:
        get_form(opts.bucket, opts.endpoint, opts.id, opts.key, opts.
out)
    else:
        print "python %s --bucket=your-bucket --endpoint=oss-cn-
hangzhou.aliyuncs.com --id=your-access-key-id --key=your-access-key-
secret --out=out-put-form-name" % __file__
```

将此段代码保存为post_object.py，然后用python post_object.py来运行。

用法：

```
python post_object.py --bucket=您的Bucket --endpoint=Bucket对应的OSS域名
 --id=您的AccessKeyId --key=您的AccessKeySecret --out=输出的文件名
示例:
python post_object.py --bucket=oss-sample --endpoint=oss-cn-hangzhou.
aliyuncs.com --id=tphpxp --key=ZQNJzf4QJRkrH4 --out=post.html
```

说明:

- 构建的表单中`success_action_redirect value=http://oss.aliyun.com`表示上传成功后跳转的页面。可以替换成您自己的页面。
- 构建的表单中`success_action_status value=201`表示的是上传成功后返回的状态码为201。可以替换。
- 假如指定生成的HTML文件为post.html，打开post.html，选择文件上传。在这个例子中如果成功则会跳转到OSS的主页面。

上传的安全及授权

为了防止第三方未经授权往您的Bucket里上传数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。

为了授权给第三方上传，OSS还提供了账号级别的授权。详情请参见授权给第三方上传。

9.1.3 分片上传和断点续传

阿里云OSS提供的分片上传（Multipart Upload）和断点续传功能，可以将要上传的文件分成多个数据块（OSS里又称之为Part）来分别上传，上传完成之后再调用OSS的接口将这些Part组合成一个Object来达到断点续传的效果。

适用场景

当使用简单上传（PutObject）功能来上传较大的文件到OSS的时候，如果上传的过程中出现了网络错误，那么此次上传失败，重试必须从文件起始位置上传。针对这种情况，您可以使用分片上传来达到断点续传的效果。

相对于其他的上传方式，分片上传适用于以下场景：

- 恶劣的网络环境：如手机端，当出现上传失败的时候，可以对失败的Part进行独立的重试，而不需要重新上传其他的Part。
- 断点续传：中途暂停之后，可以从上次上传完成的Part的位置继续上传。
- 加速上传：要上传到OSS的本地文件很大的时候，可以并行上传多个Part以加快上传。
- 流式上传：可以在需要上传的文件大小还不确定的情况下开始上传。这种场景在视频监控等行业应用中比较常见。

分片上传操作方式

操作方式	说明
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	

断点续传操作方式

在使用分片上传的过程中，如果系统意外崩溃，可以在重启的时候通过ListMultipartUploads和ListParts两个接口来获取某个Object上的所有的分片上传任务和每个分片上传任务中上传成功的Part列表。这样就可以从最后一块成功上传的Part开始继续上传，从而达到断点续传的效果。暂停和恢复上传实现原理也是一样的。

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	

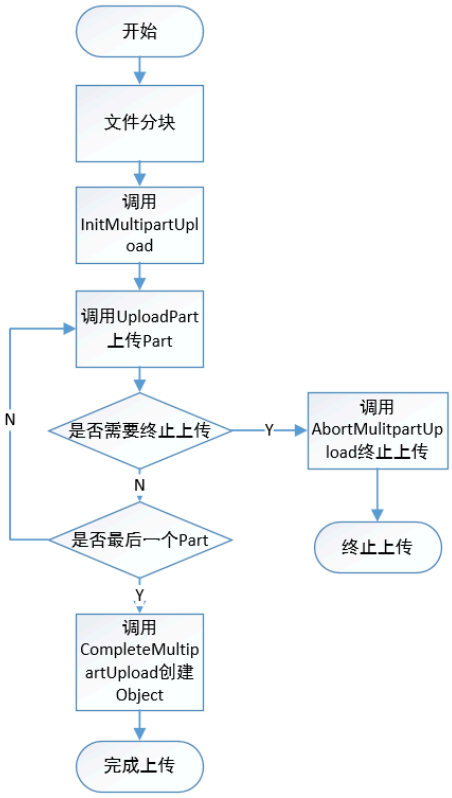
上传限制

- 大小限制：在这种上传方式下，Object的大小是由Part来决定的，最大支持10000块Part。每块Part最小100KB（最后一块可以比100KB小），最大5GB。Object的大小不能超过48.8TB。
- 命名限制
  - 使用UTF-8编码。
  - 长度必须在1–1023字节之间。
  - 不能以正斜线（/）或者反斜线（\）字符开头。

分片上传流程

分片上传的基本流程如下：

1. 将要上传的文件按照一定的大小分片。
2. 初始化一个分片上传任务（InitiateMultipartUpload）。
3. 逐个或并行上传分片（UploadPart）。
4. 完成上传（CompleteMultipartUpload）。



该过程需注意以下几点：

- 除了最后一块Part，其他Part的大小不能小于100KB，否则会导致调用CompleteMultipartUpload接口时失败。
- 要上传的文件切分成Part之后，文件顺序是通过上传过程中指定的partNumber来确定的，实际执行中并没有顺序要求，因此可以实现并发上传。

  具体的并发个数并不是越多速度越快，要结合用户自身的网络情况和设备负载综合考虑。网络情况较好时，建议增大分片大小。反之，减小分片大小。
- 默认情况下，已经上传但还没有调用CompleteMultipartUpload的Part是不会自动回收的，因此如果要终止上传并删除占用的空间请调用AbortMultipartUpload。如果需要自动回收上传的Part，请参考Object生命周期管理。

上传的安全及授权

为了防止第三方未经授权往您的Bucket里上传数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。

为了授权给第三方上传，OSS还提供了账号级别的授权。详情请参见授权给第三方上传。

上传后续操作

- 在文件上传到OSS上后，您可以通过上传回调来向指定的应用服务器发起回调请求，进行下一步操作。
- 如果上传的是图片，您还可以进行图片处理。
- 如果上传是音频或者视频文件，您还可以进行媒体处理。

API参考

- 分片上传API：
  - MultipartUpload
  - InitiateMultipartUpload
  - UploadPart
  - UploadPartCopy
  - CompleteMultipartUpload
  - AbortMultipartUpload
  - ListMultipartUploads
  - ListParts

9.1.4 追加上传

追加上传指的是使用OSS API中的AppendObject方法在已上传的Appendable Object类型文件后面直接追加内容。

说明：

追加上传的API接口的详细信息请参见AppendObject。

适用场景

之前提到的上传方式，比如简单上传，表单上传，断点续传上传等，创建的Object都是Normal类型，这种Object在上传结束之后内容就是固定的，只能读取，不能修改。如果Object内容发生了改变，只能重新上传同名的Object来覆盖之前的内容，这也是OSS和普通文件系统使用的一个重大区别。

正因为这种特性，在很多应用场景下会很不方便，比如视频监控、视频直播领域等，视频数据在实时的不断产生。如果使用其他上传方式，只能将视频流按照一定规律切分成小块然后不断的上传新的Object。这种方式在实际使用上存在很明显的缺点：

- 软件架构比较复杂，需要考虑文件分块等细节问题。
- 需要有位置保存元信息，比如已经生成的Object列表等，然后每次请求都重复读取元信息来判断是否有新的Object生成。这样对服务器的压力很大，而且客户端每次都需要发送两次网络请求，延时上也会有一定的影响。
- 如果Object切分的比较小的话，延时比较低，但是众多Object会导致管理起来很复杂。如果Object切分的比较大的话，数据的延时又会很高。

为了简化这种场景下的开发成本，OSS提供了追加上传（Append Object）的方式在一个Object后面直接追加内容的功能。通过这种方式操作的Object的类型为Appendable Object，而其他的方式上传的Object类型为Normal Object。每次追加上传的数据都能够即时可读。

如果使用追加上传，那么上述场景的架构就变得很简单。视频数据产生之后即时地通过追加上传到同一个Object，而客户端只需要定时获取该Object的长度与上次读取的长度进行对比，如果发现有新的数据可读，那么就触发一次读操作来获取新上传的数据部分即可。通过这种方式可以很大的简化架构，增强扩展性。

不仅在视频场景，在日志追加上传的场景下，追加上传也能发挥作用。

操作方式

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	

上传限制

- 大小限制：在这种上传方式下，Object不能超过5GB。

- 命名限制：
  - 使用UTF-8编码。
  - 长度必须在1–1023字节之间。
  - 不能以正斜线（/）或者反斜线（\）字符开头。
- 文件类型：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。
- 后续操作限制：通过追加上传的文件，不能被复制，可以修改文件本身的meta信息。

上传的安全及授权

为了防止第三方未经授权往您的Bucket里上传数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。

为了授权给第三方上传，OSS还提供了账号级别的授权。详情请参见授权给第三方上传。

上传后续操作

- 如果上传的是图片，您还可以进行图片处理。
- 如果上传是音频或者视频文件，您还可以进行媒体处理。

说明：
追加上传不支持上传回调操作。

9.1.5 授权给第三方上传

本文介绍如何授权第三方直接上传文件到OSS而不通过服务器端转发。

适用场景

在典型的C/S系统架构中，服务器端负责接收并处理客户端的请求。那么考虑一个使用OSS作为后端的存储服务，客户端将要上传的文件发送给服务器端，然后服务器端再将数据转发上传到OSS。在这个过程中，一份数据需要在网络上传输两次，一次从客户端到服务器端，一次从服务器端到OSS。当访问量很大的时候，服务器端需要有足够的带宽资源来满足多个客户端的同时上传的需求，这对架构的伸缩性提出了挑战。

为了解决这种场景带来的挑战，OSS提供了授权给第三方上传的功能。使用这个功能，每个客户端可以直接将文件上传到OSS而不是通过服务器端转发，节省了自建服务器的成本，并且充分利用了OSS的海量数据处理能力，无需考虑带宽和并发限制等，可以让客户专心于业务处理。

目前授权上传有两种实现方式：URL签名和临时访问凭证。

URL签名

URL签名即在请求的URL中带OSS AccessKeyId和Signature字段，这样用户就可以直接使用该URL来进行上传。每个URL签名中携带有过期时间以保证安全。

- 操作方法

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	

- 更多详情请参考在URL中包含签名。

临时访问凭证

临时访问凭证是通过阿里云Security Token Service（STS）来实现授权的一种方式。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。

- 操作方法

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	

- 操作示例请参考STS临时授权访问OSS。

9.1.6 上传回调

阿里云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。

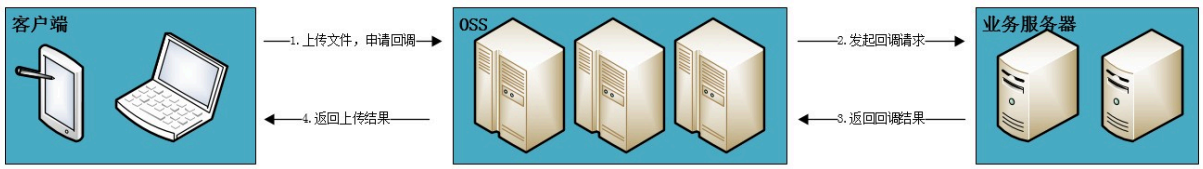
> 说明:
>
> · Callback的OSS API接口详细信息请参见Callback。
> · 支持CallBack的API接口有：PutObject、PostObject、CompleteMultipartUpload。

适用场景

上传回调的一种典型应用场景是与授权第三方上传同时使用，客户端在上传文件到OSS的时候指定到服务器端的回调，当客户端的上传任务在OSS执行完毕之后，OSS会向应用服务器端主动发起HTTP请求进行回调，这样服务器端就可以及时得到上传完成的通知从而可以完成诸如数据库修改等操作，当回调请求接收到服务器端的响应之后OSS才会将状态返回给客户端。

OSS在向应用服务器发送POST回调请求的时候，会在POST请求的Body中包含一些参数来携带特定的信息。这些参数有两种，一种是系统定义的参数，如Bucket名称、Object名称等；另外一种是自定义的参数，您可以在发送带回调的请求给OSS的时候根据应用逻辑的需要指定这些参数。您可以通过使用自定义参数来携带一些和应用逻辑相关的信息，比如发起请求的用户id等。具体使用自定义参数的方法可以参考Callback。

通过适当的使用上传回调机制，能很好的降低客户端的逻辑复杂度和网络消耗。流程如下：



> 说明:
> 目前只有简单上传（PutObject）、表单上传（PostObject）、分片上传完成（Complete Multipart Upload）操作支持上传回调功能。

操作方式

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
C SDK	

操作方式	说明
.NET SDK	

更多参考

- 上传回调错误及排除
- Web 端直传实践及上传回调
- 快速搭建移动应用上传回调服务

9.1.7 RTMP推流上传

OSS支持使用RTMP协议推送H264编码的视频流和AAC编码的音频流到OSS。推送到OSS的音视频数据可以点播播放；在对延迟不敏感的应用场景，也可以做直播用途。

通过RTMP协议上传音视频数据有以下限制：

- 只能使用RTMP推流的方式，不支持拉流。
- 必须包含视频流，且视频流格式为H264。
- 音频流是可选的，并且只支持AAC格式，其他格式的音频流会被丢弃。
- 转储只支持HLS协议。
- 一个LiveChannel同时只能有一个客户端向其推流。

下面分别介绍如何推送音视频流到OSS，以及如何点播和直播播放。

向OSS推送音视频数据

- 获得推流地址

  使用SDK调用PutLiveChannel接口，创建一个LiveChannel，并获取对应的推流地址。如果Bucket的权限控制（ACL）为 公共读写（public-read-write），那么可以直接使用得到的推流地址进行推流；否则需要进行签名操作。

  以Python SDK为例，获取未签名以及签名推流地址的代码如下：

```
from oss2 import *
from oss2.models import *
host = "oss-cn-hangzhou.aliyuncs.com" #just for example
accessid = "your-access-id"
accesskey = "your-access-key"
bucket_name = "your-bucket"
channel_name = "test-channel"
auth = Auth(accessid, accesskey)
bucket = Bucket(auth, host, bucket_name)
channel_cfg = LiveChannelInfo(target = LiveChannelInfoTarget())
channel = bucket.create_live_channel(channel_name, channel_cfg)
publish_url = channel.publish_url
```

```
signed_publish_url = bucket.sign_rtmp_url("test-channel", "playlist.
m3u8", 3600)
```

获得的推流地址示例如下：

```
publish_url = rtmp://your-bucket.oss-cn-hangzhou.aliyuncs.com/live/
test-channel
signed_publish_url = rtmp://your-bucket.oss-cn-hangzhou.aliyuncs.com
/live/your-channel?OSSAccessKeyId=LGarxxxxxxHjKWg6&playlistName=t.
m3u8&Expires=1472201595&Signature=bjKraZTTyzz9%2FpYoomDx4Wgh%2FlM%3D
"
```

- 使用ffmpeg进行推流

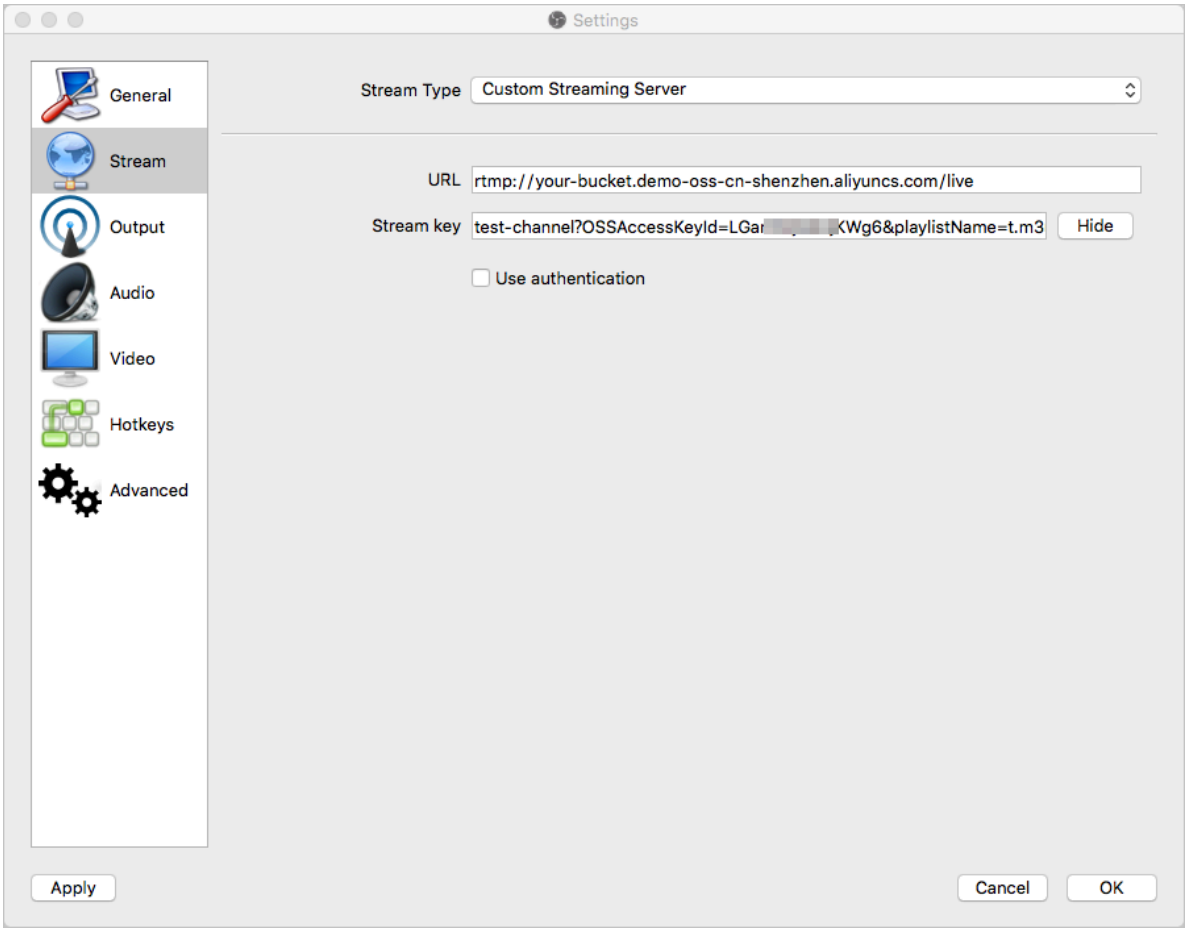
  可以使用ffmpeg推送本地的视频文件到OSS，命令如下：

  ```
  ffmpeg -i 1.flv -c copy -f flv "rtmp://your-bucket.oss-cn-hangzhou
  .aliyuncs.com/live/test-channel?OSSAccessKeyId=LGarxxxxxxHjKWg6&
  Expires=1472199095&Signature=%2FAvRo7FTss1InBKgwn7Gz%2FUlp9w%3D"
  ```

- 使用OBS进行推流

  首先单击Settings，在URL文本框中输入前面步骤获取的推流地址，然后单击OK开始推流。

  如下图所示，请注意推流地址的拆分方式：



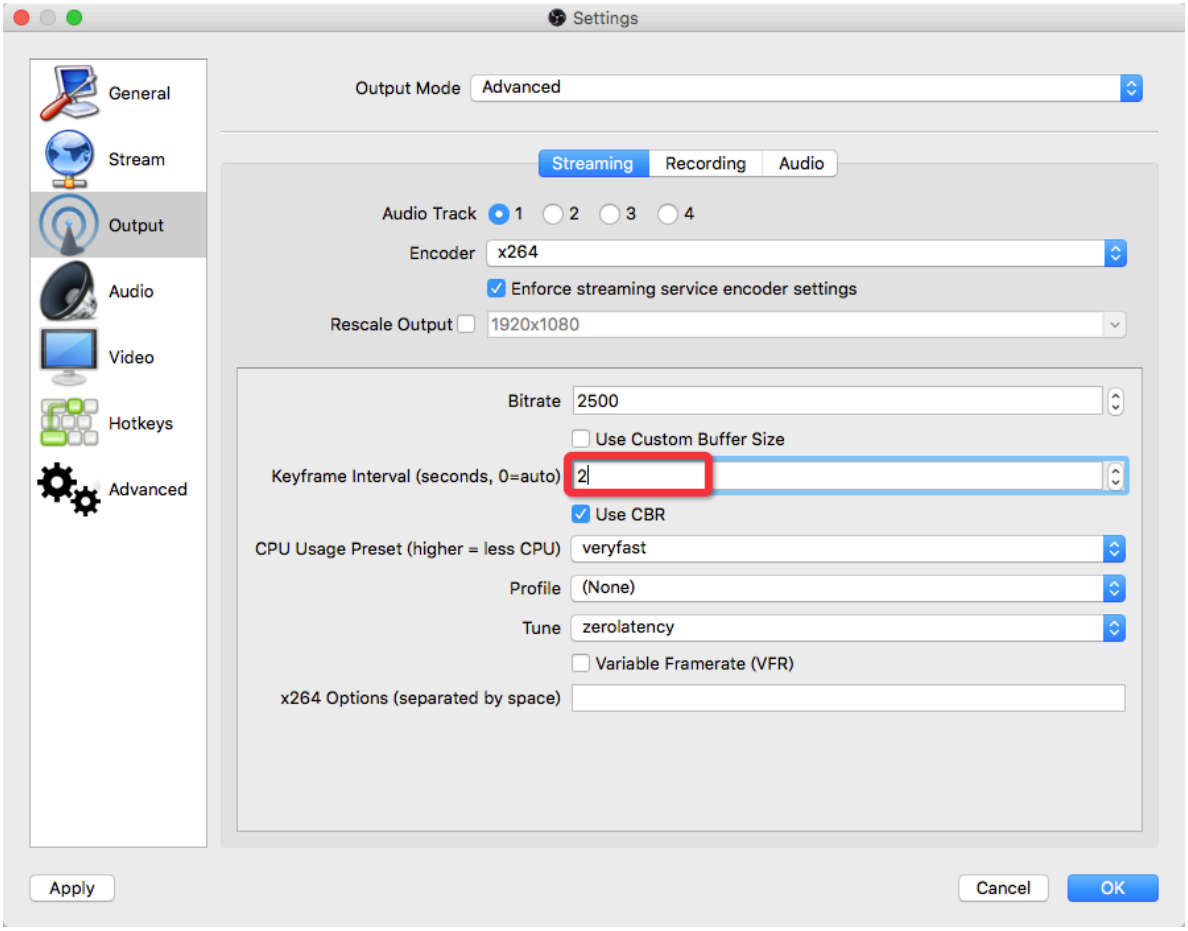
播放推送到OSS的音视频数据

- 直播场景

  在推流的过程中，可以通过HLS协议播放正在推送的内容。各个平台的播放方法如下：

  - 在Android、iOS等移动平台，直接在浏览器输入LiveChannel对应的播放地址即可。
  - Mac OS可以使用safari浏览器进行播放。
  - PC端可以安装vlc播放器进行播放。

  为了直播流畅，可以设置比较小的FragDuration，例如2s；另外，GOP的大小最好固定且与LiveChannel的FragDuration配置一致。OBS的GOP（即`keyframe Interval`）设置方法如下：



- 点播场景

  推流的过程中，OSS总是以直播流的方式推送/更新M3U8。所以对于点播的场景，需要在推流结束后，调用PostVodPlaylist接口来组装一个点播用的m3u8文件，然后使用该文件地址来播放。

  对于点播的场景，可以设置较大的GOP来减少ts文件数，降低码率。

9.2 下载文件

9.2.1 简单下载

简单下载是通过OSS API的GetObject接口，下载已经上传的文件（Object），Object下载是使用HTTP的GET请求来完成的。

说明:

- 简单下载的API接口详细信息请参见GetObject。
- Object的URL生成规则请参考OSS的访问。
- 如果需要使用自定义域名来访问Object，请参考自定义域名访问OSS。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
Android SDK	
iOS SDK	
Node.js SDK	
Browser.js SDK	
Ruby SDK	

下载的安全及授权

- 为了防止第三方未经授权从您的Bucket里下载数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。
- 如果希望将私有Bucket的Object提供给第三方进行下载，请参考授权给第三方下载。

更多参考

如果需要实现断点下载请参考断点续传下载。

9.2.2 断点续传下载

OSS提供了从Object指定的位置开始下载的功能，在下载大的Object的时候，可以分多次下载。如果下载中断，重启的时候也可以从上次完成的位置开始继续下载。

和简单上传类似，您也需要对该Object有读权限。通过设置参数Range来支持断点续传，对于比较大的Object建议使用该功能。Range的定义可参考HTTP RFC。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。例如：Content-Range: bytes 0–9/44，表示整个文件长度为44，此次返回的范围为0–9。如果不在范围内，则传送整个文件，并且不在结果中提及Content-Range，返回码为206。

操作方式

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
Go SDK	
C SDK	
Android SDK	
iOS SDK	

下载的安全及授权

- 为了防止第三方未经授权从您的Bucket里下载数据，OSS提供了Bucket和Object级别的访问权限控制。详情请参见权限控制。
- 如果希望将私有Bucket的Object提供给第三方进行下载，请参考授权给第三方下载。

9.2.3 授权给第三方下载

当您希望将私有Bucket内部的Object授权给第三方下载的时候，不应该直接将AccessKey提供给下载者，而应该使用URL签名或临时访问凭证两种方式授权给第三方下载。

URL签名

OSS提供了签名下载的方法。您可以在URL中加入签名信息，把该URL转给第三方实现授权访问。第三方用户只需要使用HTTP的GET请求访问此URL即可下载Object。

- 实现方式

  URL中包含签名示例如下：

  ```
  http://<bucket>.<region>.aliyuncs.com/<object>?OSSAccessKeyId=<user
  access_key_id>&Expires=<unix time>&Signature=<signature_string>
  ```

  在URL中实现签名，必须至少包含Signature、Expires、OSSAccessKeyId三个参数。

  - OSSAccessKeyId：您的AccessKeyId。
  - Expires：您期望URL过期的时间。
  - Signature：您签名的字符串，具体请参见在URL中包含签名。

    说明：

    此链接需要进行URL编码。

  - 操作方法

操作方式	说明
控制台	Web应用程序，直观易用
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	

临时访问凭证

OSS通过STS（Security Token Service）提供了临时凭证给第三方用户，第三方用户以在请求头部中带签名的方式去访问Object。这种授权方式适合移动场景的下载。临时访问凭证实现请参见STS Java SDK。

- 实现方式

  第三方用户向app服务器请求，获取了STS颁发的AccessKeyId、AccessKeySecret以及STS Token。第三方用户以STS AccessKeyId、AccessKeySecret以及STS Token去请求开发者的Object资源。

- 操作方法

操作方式	说明
控制台	Web应用程序，直观易用

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Android SDK	
iOS SDK	

最佳实践

- RAM和STS使用指南

9.2.4 选取内容（OSS Select）

通过 OSS Select，您可以使用简单的结构化查询语言 (SQL) 语句选取 OSS 中文件的内容，仅获取需要的数据。使用 OSS Select 可以减少从 OSS 传输的数据量，从而提高数据获取效率，节约时间。

说明:

您可以在请求中将 SQL 表达式传递给 OSS。有关 OSS Select 支持的 SQL 语句和限制，请参考#unique_232。

操作方式

OSS SDK（目前 Java 和 Python 的 SDK 已支持 OSS Select）。

限制

- 文件格式：OSS Select 支持编码为 UTF-8 且符合 RFC 4180 标准的 CSV 文件（包括TSV 等类 CSV 文件）。文件的行列分隔符及 Quote 字符都可以自由定义。
- 加密方式：OSS Select 支持 OSS 完全托管和通过 KMS 托管主密钥进行加密的文件。
- 地域：OSS Select 目前只在华南1（深圳）开放。

9.3 管理文件

9.3.1 管理文件元信息

文件元信息（Object Meta）是对上传到OSS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和 User Meta（用户自定义元信息）。文件元信息可以在各种方式上传时或者拷贝文件时进行设置。

- HTTP标准属性

名称	描述
Cache-Control	指定该 Object 被下载时的网页的缓存行为
Content-Disposition	指定该 Object 被下载时的名称
Content-Encoding	指定该 Object 被下载时的内容编码格式
Content-Language	指定该 Object 被下载时的内容语言编码
Expires	过期时间
Content-Length	该 Object 大小
Content-Type	该 Object 文件类型
Last-Modified	最近修改时间

- User Meta

  为了便于用户对 Object 进行更多描述，OSS 中规定所有以 x-oss-meta- 为前缀的参数视为 User Meta，比如 x-oss-meta-location。一个 Object 可以有多个类似的参数，但所有的 User Meta 总大小不能超过 8KB。这些 User Meta 信息会在 GetObject 或者 HeadObject 的时候在 HTTP 头部中返回。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	

操作方式	说明
.NET SDK	
Android SDK	

更多参考

您也可以在以下操作中添加文件元信息：

- #unique_68
- #unique_243
- #unique_244
- #unique_245

9.3.2 查看文件列表

您可以通过 OSS API 中的 GetBucket 接口列出您在存储空间（Bucket）中上传的文件（Object）。

说明：

查看文件列表的 API 详细信息请参考 GetBucket。

您可以调用通过 GetBucket 接口，一次性得到某一 Bucket 下最多 1000 个 Object 的列表。通过下面的四个参数，您可以实现多种列举功能：

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。默认为100，最大值为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。

操作方式

操作方式	说明
控制台	可在控制台对应 Bucket 的文件管理内直接查看
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Browser.js SDK	
Ruby SDK	

文件夹功能

OSS服务是没有文件夹这个概念的，所有元素都是以Object来存储。创建模拟文件夹本质上来说是创建了一个大小为0的Object。这个Object可以被上传和下载，只是控制台会对以正斜线（/）结尾的Object以文件夹的方式展示。所以您可以使用上述方式来创建文件夹。

您可以通过 Delimiter 和 Prefix 参数的配合实现文件夹功能：

- 如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。
- 如果再把 Delimiter 设置为正斜线（/）时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

举个例子：

在OSS的Bucket `oss-sample`下有如下Object：

```
文件D
目录A/文件C
目录A/文件D
目录A/目录B/文件B
目录A/目录B/目录C/文件A
目录A/目录C/文件A
目录A/目录D/文件B
目录B/文件A
```

- 列出第一层目录和文件

  根据API中请求约定，需要设置Prefix为“”，Delimiter为“/”。返回结果如下：

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

```
<ListBucketResult>
  <Name>oss-sample</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>文件D</Key>
    <LastModified>2015-11-06T10:07:11.000Z</LastModified>
    <ETag>"8110930DA5E04B1ED5D84D6CC4DC9080"</ETag>
    <Type>Normal</Type>
    <Size>3340</Size>
    <StorageClass>Standard</StorageClass>
    <Owner>
      <ID>oss</ID>
      <DisplayName>oss</DisplayName>
    </Owner>
  </Contents>
  <CommonPrefixes>
    <Prefix>目录A/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>目录B/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

可以看到：

Contents返回的是第一层的文件：“文件D”。 CommonPrefixes返回的是第一层的目录：“目录A/” 和 “目录B/”，而“目录A/” 和 “目录B/”下的文件名不显示。

- **列出目录A底下的第二层目录和文件**

  根据API中请求约定，需要设置Prefix为“目录A”，Delimiter为“/”。返回结果如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult>
  <Name>oss-sample</Name>
  <Prefix>目录A/</Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>目录A/文件C</Key>
    <LastModified>2015-11-06T09:36:00.000Z</LastModified>
    <ETag>"B026324C6904B2A9CB4B88D6D61C81D1"</ETag>
    <Type>Normal</Type>
    <Size>2</Size>
    <StorageClass>Standard</StorageClass>
    <Owner>
      <ID>oss</ID>
      <DisplayName>oss</DisplayName>
    </Owner>
  </Contents>
  <Contents>
    <Key>目录A/文件D</Key>
    <LastModified>2015-11-06T09:36:00.000Z</LastModified>
    <ETag>"B026324C6904B2A9CB4B88D6D61C81D1"</ETag>
    <Type>Normal</Type>
```

```
    <Size>2</Size>
    <StorageClass>Standard</StorageClass>
    <Owner>
      <ID>oss</ID>
      <DisplayName>oss</DisplayName>
    </Owner>
  </Contents>
  <CommonPrefixes>
    <Prefix>目录A/目录B/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>目录A/目录C/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>目录A/目录D/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

可以看到：

Contents返回的是第二层的文件：“目录A/文件C”，“目录A/文件D”。CommonPrefixes返回的是第二层的目录：“目录A/目录B/”，“目录A/目录C/”和“目录A/目录D/”。而目录下的文件名不会被显示。

9.3.3 拷贝文件

拷贝文件是指在不改变文件内容的情况下，将某个存储空间（Bucket）内的一些文件（Object）复制到另外一个 Bucket 中。

之前，拷贝文件通常是将 Object 重新下载然后上传，但这种做法会浪费很多网络带宽。为此，OSS 提供了 CopyObject 的功能来实现 OSS 的内部拷贝，这样在用户和 OSS 之间就无需传输大量的数据。

另外，由于 OSS 不提供重命名功能，如果需要对 Object 进行重命名，建议您调用 OSS 的 CopyObject 接口先将原来的数据拷贝成新的文件名，然后删除源 Object。如果您仅需要修改某个 Object 的一些 Object Meta 信息，同样可以调用 CopyObject 接口，将源 Object 和目标 Object 设置成相同的地址，这样 OSS 就会仅更新 Object Meta 信息。Object Meta 信息可以参考 Object Meta。

操作方式

操作方式	说明
图形化工具ossbrowser	图形化工具，易操作 注意: 仅可以用于拷贝 5GB 以下的文件。
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言 SDK demo

操作方式	说明
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Browser.js SDK	
Ruby SDK	

注意事项

拷贝文件时，有以下注意事项：

- 您需要有源 Object 的操作权限，否则会无法完成操作。
- 不支持跨 Region 拷贝数据。比如，不支持将杭州 Bucket 里的 Object 拷贝到青岛的 Bucket 中。
- 不能对追加上传产生的 Object 进行拷贝。
- 1GB 以下的文件使用简单拷贝，OSS API 接口为 CopyObject。
- 1GB 以上的文件使用分片拷贝，OSS API 接口为 UploadPartCopy。

9.3.4 删除文件

删除文件即删除上传到存储空间（Bucket）中的文件（Object）。

OSS允许您执行如下删除动作：

- 单个删除：指定某个Object进行删除。
- 批量删除：一次最多可指定1000个Object进行删除。
- 自动删除：如果需要删除的Object数目很多，而且删除的Object有一定的规律，比如定期删除某些天之前的Object，或者是要清空整个Bucket，推荐使用生命周期管理来自动删除Object。设置了生命周期规则之后，OSS会根据规则自动删除到期的Object，从而极大减少您发送删除请求的次数，提高删除效率。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用

操作方式	说明
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Browser.js SDK	
Ruby SDK	

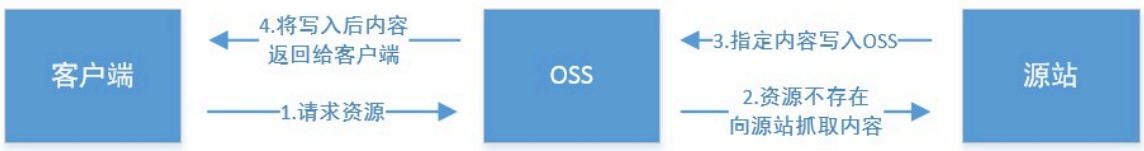
9.3.5 管理回源设置

通过回源设置，对于获取数据的请求以多种方式进行回源读取，满足您对于数据热迁移、特定请求重定向等需求。

> 说明:
> 详细的配置步骤请参考管理回源规则。

对OSS的每条GET请求的URL按设定的规则进行匹配，然后按照设定的规则进行回源。最多配置5条规则，顺序匹配，直到匹配到有效规则。回源类型分为镜像方式和重定向的方式。

镜像方式



如果配置了镜像回写，则对一个不存在的文件进行GET操作时，会向源地址请求这个文件，返回给用户，并同时写入到OSS。

使用场景

镜像回源主要用于无缝迁移数据到OSS，即服务已经在自己建立的源站或者在其他云产品上运行，需要迁移到OSS上，但是又不能停止服务，此时可利用镜像回写功能实现。具体场景分析如下：

- 源站有一批冷数据，同时在不断的生成新的热数据。

  可以先通过迁移工具将冷数据迁移到OSS上，迁移工具为OssImport，同时配置镜像回写，将源站的地址配置到OSS上。当将域名切换到OSS上（或者阿里云的CDN，回源到OSS）之后，就算有一部分新生成的数据没有迁移过来，依然可以在OSS上正常访问到，且访问一次后文件就会存入到OSS。域名切换后，源站已经没有新的数据产生了，此时再扫描一次，将还没有导过来的数据一次性导入到OSS，然后将镜像回写配置删除。

  如果配置的源站是IP地址，那么将域名迁移到OSS后还可以继续镜像到源站，但是如果配置的是一个域名，由于域名本身会解析到OSS或者CDN，那么镜像就失去作用了，在这种情况下，可以另外申请一个域名作为镜像的源站，这个域名与正在服务的域名解析到同一个IP地址，这样服务域名迁移的时候就可以继续镜像到源站了。
- 只切换源站的部分流量到OSS或者CDN，源站本身还在不断的产生数据。

  迁移方式与上述方式类似，只是流量切换到OSS后，不要将镜像回写配置删掉，这样可以保证切换到OSS或者CDN的流量还是能够取到源站的数据。

细节分析

- 只有当GetObject()本应该返回404的情况下，OSS才会执行镜像回源，向源站请求文件。
- 向源站请求的URL为`MirrorURL+object`，回写到OSS的文件名为“object”，例如bucket为`example-bucket`，配置了镜像回写，MirrorURL为`http://www.example-domain.com/`，文件`image/example_object.jpg`不在这个bucket里面，此时去下载这个文件时，OSS将向`http://www.example-domain.com/image/example_object.jpg`发起GET请求，并将结果同时返回给用户以及写入到OSS，当下载完成后，这个文件就已经存在OSS上了，文件名为`image/example_object.jpg`，此时相当于将源站的文件同名的迁移到了OSS上。如果MirrorURL带有path信息，比如`http://www.example-domain.com/dir1/`，则与上例相同，OSS回源的URL为`http://www.example-domain.com/dir1/image/example_object.jpg`，写入到OSS的object依然是`image/example_object.jpg`，此时相当于将源站的某一个目录下的文件迁移到OSS上。
- 传给OSS的header信息不会传递给源站，querystring信息是否会传递给源站取决于控制台回源规则中的配置。
- 如果源站是chunked编码返回，那么OSS返回给用户的也是chunked编码。

- OSS会将源站的以下头信息返回并存为OSS的头信息：

```
Content-Type
Content-Encoding
Content-Disposition
Cache-Control
Expires
Content-Language
Access-Control-Allow-Origin
```

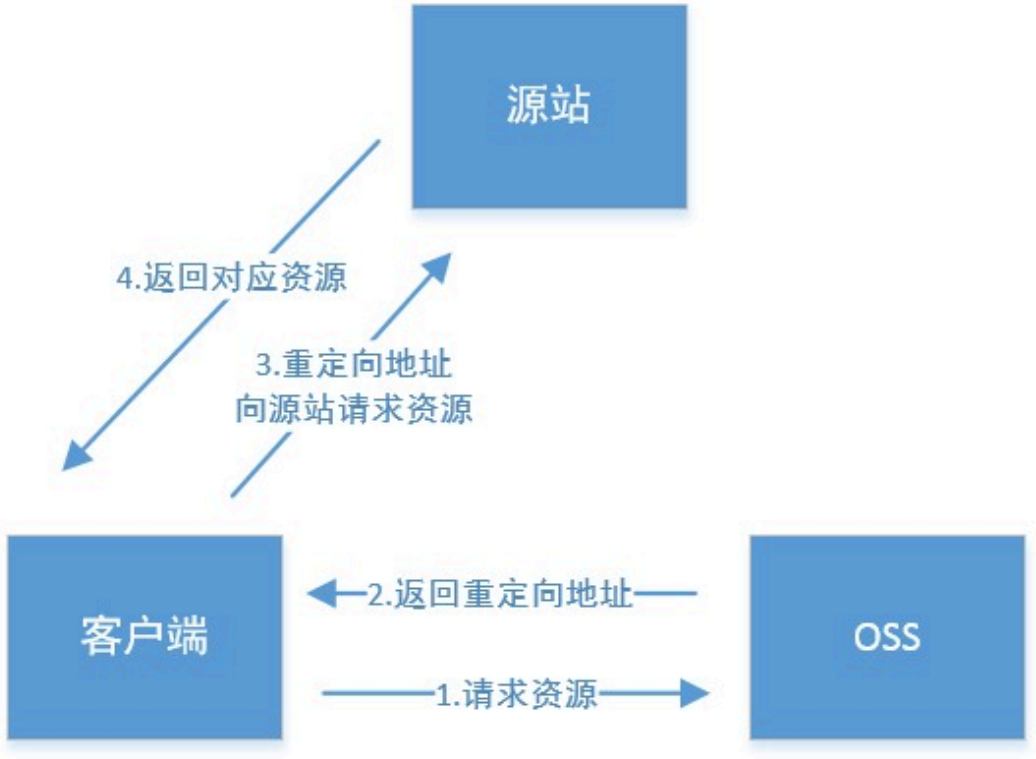
- 通过镜像回写的文件会添加一个回应头x-oss-tag，值为MIRROR + 空格 + url_decode（回源URL），例如：

```
x-oss-tag:MIRROR http%3a%2f%2fwww.example-domain.com%2fdir1%2fimage%
2fexample_object.jpg
```

  文件回写到OSS后，只要文件不被重新覆盖，每次下载这个文件都会添加这个头部，用于表示这个文件来源于镜像。
- 假设文件已经通过镜像回写到了OSS，如果源站的相同文件发生了变化，那OSS不会更新已经存在于OSS上的文件，因为此时文件已经在OSS上，不符合镜像回写的条件。
- 如果镜像源也不存在此文件，即镜像源返回给OSS的http status为404，那么OSS也返回404给用户，如果是其他非200的状态码（包括因为网络原因等获取不到文件的错误情况），OSS将返回424给用户，错误码为“MirrorFailed”。

重定向

URL重定向功能的作用是根据用户设置的条件，以及相应的跳转的配置，向用户返回一个3xx跳转。用户可以利用这种跳转的功能对文件做重定向以及在此基础之上的各种业务。其流程如下图所示。



使用场景

- **其他数据源向OSS的无缝迁移。**

  用户异步的从自己的数据源向OSS迁移数据，在此过程中未迁移到OSS的数据通过URL rewrite的方式返回给用户一个302重定向请求，用户的客户端根据302中的Location从自己的数据源读回数据。

- **配置页面跳转功能。**

  比如用户希望隐藏自己的某些前缀开头的object，给访问者返回一个特殊的页面。

- **配置发生404或500错误时的跳转页面。**

  发生以上错误的时候用户可以看到一个预先设定的页面，不至于系统发生错误的时候向用户完全暴露OSS的错误。

9.3.6 SelectObject

SelectObject典型的应用场景是做日志文件分析，还可以和大数据产品结合使用。本文主要介绍如何使用Python SDK以及Java SDK实现以上应用场景。

功能介绍

对象存储（Object Storage Service，简称OSS）是基于阿里云飞天分布式系统的海量、安全和高可靠的云存储服务，是一种面向互联网的大规模、低成本、通用存储，提供RESTful API，具备容量和处理的弹性扩展能力。OSS不仅非常适合存储海量的媒体文件，也适合作为数据仓库存储海量的数据文件。目前Hadoop 3.0已经支持OSS，在EMR上运行Spark、Hive、Presto等服务，以及阿里自研的MaxCompute、HybridDB以及新上线的Data Lake Analytics都支持从OSS直接处理数据。

然而，目前OSS提供的GetObject接口决定了大数据平台只能把OSS数据全部下载到本地然后进行分析过滤，在很多查询场景下浪费了大量带宽和客户端资源。

SelectObject接口是对上述问题的解决方案。其核心思想是大数据平台将条件、Projection下推到OSS层，让OSS做基本的过滤，从而只返回有用的数据。客户端一方面可以减少网络带宽，另一方面也减少了数据的处理量，从而节省了CPU和内存用来做其他更多的事情。这使得基于OSS的数据仓库、数据分析成为一种更有吸引力的选择。

SelectObject提供了Java和Python 的SDK，其他语言SDK也会很快推出。目前支持RFC 4180标准的CSV（包括TSV等类CSV文件，文件的行列分隔符以及Quote字符都可自定义）和Json文件，且文件编码为UTF-8。支持标准存储类型和低频访问存储类型的文件（归档文件需要先取回）。支持加密文件（OSS完全托管加密、KMS托管主密钥）。

JSON支持DOCUMENT和LINES两种文件。DOCUMENT是指整个文件是单一的JSON对象，LINES表示整个文件由一行行的JSON对象组成，每一行是一个JSON对象（但整个文件本身并不是一个合法的JSON对象），行与行之间以换行分隔符隔开。OSS Select可以支持常见的\n，\r\n等分隔符，且无需用户指定。

- 支持的SQL语法
  - SQL 语句： Select From Where
  - 数据类型：string、int(64bit)、double(64bit), decimal(128) 、timestamp、bool
  - 操作： 逻辑条件（AND,OR,NOT)， 算术表达式（+-*/%)， 比较操作(>,=, <, >=, <=, !=), String 操作 (LIKE, ||)
- 分片查询

  和GetObject提供了基于Byte的分片下载类似，SelectObject也提供了分片查询的机制，包括两种分片方式：按行分片、按Split分片。按行分片是常用的分片方式，然而对于稀疏数据来

说，按行分片可能会导致分片时负载不均衡。Split是OSS用于分片的一个概念，一个Split包含多行数据，每个Split的数据大小大致相等。按Spit分片比按行分片要更加高效。

- 数据类型

  关于数据类型，OSS中的CSV数据默认都是String类型，用户可以使用CAST函数实现数据转换，比如下面的SQL查询将_1和_2转换为int后进行比较。

  ```
  Select * from OSSOBject where cast (_1 as int) > cast(_2 as int)
  ```

  同时，对于SelectObject支持在Where条件中进行隐式的转换，比如下面的语句中第一列和第二列将被转换成int：

  ```
  Select _1 from ossobject where _1 + _2 > 100
  ```

  对于JSON文件，如果在SQL中未指定cast函数，则其类型根据Json数据的实际类型而定，标准Json内建的数据类型包括null、bool、int64、double、string等类型。

Python SDK 样例

```
import os
import oss2

def  select_call_back(consumed_bytes, total_bytes =  None):
 print('Consumed Bytes:'  +  str(consumed_bytes) +  '\n')

# 首先初始化AccessKeyId、AccessKeySecret、Endpoint等信息。
# 通过环境变量获取，或者把诸如“<yourAccessKeyId>”替换成真实的AccessKeyId等。
#
# 以杭州区域为例，Endpoint可以是：
# http://oss-cn-hangzhou.aliyuncs.com
# https://oss-cn-hangzhou.aliyuncs.com

access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '<yourAccessKeyId
>')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '<
yourAccessKeySecret>')
bucket_name = os.getenv('OSS_TEST_BUCKET', '<yourBucket>')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '<yourEndpoint>')

# 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)
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 = {'CsvHeaderInfo': 'None',
'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_jso
n_params)

bucket.delete_object(key)
```

Python Select API详解

- **select_object**
  - **select_object 示例：**

```
def select_object(self, key, sql,
                  progress_callback=None,
```

```
                    select_params=None                    ):
```

上述示例的功能：对指定Key的文件运行SQL，返回查询结果。

- 参数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是该API的重点，它指定了select执行的各种参数以及行为。
- headers可以让用户指定请求中附带的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为DOCUMENT时，该文件为JSON文件。 ■ 当Json_Type为LINES时，该文件为JSON LINE文件。
CsvHeaderInfo	CSV的header信息，合法值为None、Ignore、 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，默认为None，可以为GZIP。

参数名称	描述
KeepAllColumns	该参数为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。 示例如下： 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（中间多了一个逗号）。如果KeepAllColumns为false，则输出为firstname,age。 引入该选项的原因是让原本处理Get-Object返回数据的代码可以不用修改的情况下平移切换到用Select object。
OutputRawData	■ 该参数为True时表示输出为Select数据，没有Frame的包装，也意味着当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，当CSV中某一列值不存在或者Json中某一个Key不存在时，直接跳过整个记录。 ■ 该参数为False时，则把该列当做null来处理。 示例如下： 假如某一行的列为firstname,lastname,age。SQL为`select _1, _4 from ossobject`。如果为True，则该行被完全跳过，如果为False，则返回firstname,\n。
MaxSkippedRecordsAllowed	允许跳过的行的最大值。默认为0，表示一旦有一行跳过就返回错误。

参数名称	描述
ParseJsonNumberAsString	当该参数为True时表示Json文件中的数字都解析为字符串；为False时则仍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。如果Select结果非常大，用read()函数则不是最佳的办法，因为该调用会阻塞直到select结果完全返回，同时占用过多的内存。

  推荐的做法是用__iter__方法（foreach chunk in result），然后对每个chunk进行处理。此做法的优势是一方面降低内存使用，另一方面OSS服务器端每处理一个请求chunk客户端就能及时处理，不必等到全部返回后才处理。

· select_object_to_file

```
def select_object_to_file(self, key, filename, sql,
                          progress_callback=None,
                          select_params=None
                          ):
```

上述示例的功能：对指定Key的文件运行SQL，将查询结果写入指定的文件。

其他参数同select_object相同。

- create_select_object_meta
  - select_meta_params 的语法结构

    ```
    def create_select_object_meta(self, key, select_meta_params=None):
    ```

    上述示例的功能：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。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，若指定，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，它包括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。

Java SDK 样例

```
package samples;

import com.aliyun.oss.ClientBuilderConfiguration;
import com.aliyun.oss.model.*;
import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClientBuilder;


import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
```

```
import com.aliyun.oss.common.auth.*;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.http.ProtocolType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.aliyuncs.sts.model.v20150401.AssumeRoleRequest;
import com.aliyuncs.sts.model.v20150401.AssumeRoleResponse;

import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
 * Examples of create select object metadata and select object.
 *
 */
class MulipartSelector implements Callable<Integer> {

    private  OSS client;
    private  String bucket;
    private  String key;
    private  int start;
    private  int end;
    private  String sql;

    public MulipartSelector(OSS client, String bucket, String key, int
 start, int end, String sql){
        this.client = client;
        this.bucket = bucket;
        this.key = key;
        this.start = start;
        this.end = end;
        this.sql = sql;
    }

    @Override
    public Integer call() throws Exception {
        SelectObjectRequest selectObjectRequest =
                new SelectObjectRequest(bucket, key)
                        .withInputSerialization(
                                new InputSerialization().withCsvInp
utFormat(
                                        new CSVFormat().withHeaderInfo
(CSVFormat.Header.None).withRecordDelimiter("\n")
                                                .withFieldDelimiter
("|")))
                        .withSplitRange(start, end)
                        .withOutputSerialization(new OutputSeri
alization().withCsvOutputFormat(new CSVFormat()).withCrcEnabled(true
));

        selectObjectRequest.setExpression(sql);
        OSSObject ossObject = client.selectObject(selectObjectRequest
);
        byte[] buffer = new byte[4096];
        int bytesRead;
        int totalSize = 0;
        try {
            while ((bytesRead = ossObject.getObjectContent().read(
buffer)) != -1) {
                totalSize += bytesRead;
            }
            String result = new String(buffer, 0, totalSize - 1);
```

```
            return new Integer(Integer.parseInt(result));
        }
        catch (IOException e){
            System.out.println(e.toString());
            return new Integer(0);
        }
    }
}
class RoleCredentialProvider {
    public static final String REGION_CN_HANGZHOU = "cn-hangzhou";
    // 当前 STS API 版本
    public static final String STS_API_VERSION = "2015-04-01";
    public static final String serviceAccessKeyId = "<access Key Id
that can do assume role>";
    public static final String serviceAccessKeySecret = "<access key
secret>";

    public static final long DurationSeconds = 15 * 60;

    private Credentials credential;
    private Calendar expireTime;

    private String roleArn;
    private DefaultAcsClient client;

    public RoleCredentialProvider(String roleArn) throws InvalidCre
dentialsException {
        this.roleArn = roleArn;
    }

    private AssumeRoleResponse assumeRole (String accessKeyId, String
accessKeySecret, String roleArn, String roleSessionName, String policy
, ProtocolType protocolType, long durationSeconds) throws ClientExce
ption {
        try {
            // create Aliyun Acs Client for sending OpenAPI requests
            if (this.client == null) {
                IClientProfile profile = DefaultProfile.getProfile(
REGION_CN_HANGZHOU, accessKeyId, accessKeySecret);
                this.client = new DefaultAcsClient(profile);
            }
            // create an instance of AssumeRoleRequest and set
properties
            final AssumeRoleRequest request = new AssumeRoleRequest();
            request.setVersion(STS_API_VERSION);
            request.setMethod(MethodType.POST);
            request.setProtocol(protocolType);
            request.setRoleArn(roleArn);
            request.setRoleSessionName(roleSessionName);
            request.setPolicy(policy);
            request.setDurationSeconds(durationSeconds);
            // send request and get the response
            final AssumeRoleResponse response = client.getAcsResponse(
request);
            return response;
        } catch (ClientException e) {
            throw e;
        }
    }


    public CredentialsProvider GetCredentialProvider()
            throws IOException {
```

```
        // AssumeRole API 请求参数: RoleArn, RoleSessionName, Policy,
and DurationSeconds
        // RoleArn 需要在 RAM 控制台上获取
        // RoleSessionName 是临时Token的会话名称，自己指定用于标识你的用户，主
要用于审计，或者用于区分Token颁发给谁
        // 但是注意RoleSessionName的长度和规则，不要有空格，只能有'-' '_' 字
母和数字等字符
        // 具体规则请参考API文档中的格式要求
        SimpleDateFormat timeFormat = new SimpleDateFormat("yyyy-MM-dd
");
        String roleSessionName = "AssumingRole" + timeFormat.format(
Calendar.getInstance().getTime());
        // read OSS data
        String policy = "{\n" +
                "    \"Version\": \"1\", \n" +
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Action\": \"oss:*\", \n" +
                "            \"Resource\": [\n" +
                "                \"acs:oss:*:*:*\"\n" +
                "            ], \n" +
                "            \"Effect\": \"Allow\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
        // 此处必须为 HTTPS
        ProtocolType protocolType = ProtocolType.HTTPS;
        try {
            final AssumeRoleResponse response = assumeRole(serviceAcc
essKeyId, serviceAccessKeySecret,
                    roleArn, roleSessionName, policy, protocolType,
DurationSeconds);
            String ossAccessId = response.getCredentials().getAccessK
eyId();
            String ossAccessKey = response.getCredentials().getAccessK
eySecret();
            String ossSts = response.getCredentials().getSecurityToken
();

            return new DefaultCredentialProvider(ossAccessId,
ossAccessKey, ossSts);

        } catch (ClientException e) {
            throw new InvalidCredentialsException("Unable tp get the
temporary AK:" + e);
        }
    }

    public void setClient(DefaultAcsClient client) {
        this.client = client;
    }

    public void setCredentials(Credentials creds) {
        this.credential = creds;
    }

    public Credentials getCredentials() {
        if (credential != null && expireTime.after(Calendar.getInstanc
e())) {
            return credential;
        }

        try {
            CredentialsProvider provider = GetCredentialProvider();
```

```
            credential = provider.getCredentials();
            expireTime = Calendar.getInstance();
            expireTime.add(Calendar.SECOND, (int) DurationSeconds - 60
);
            return credential;
        } catch (IOException e) {
            throw new InvalidCredentialsException("Unable tp get the
temporary AK:" + e);
        }
    }
}
public class SelectObjectSample {
    private static String endpoint = "<oss endpoint>";
    private static String bucketName = "<bucket>”;
    private static String key = "<object>";
    private static String roleArn = "<service role’s ARN>";// 访问控制台
中可以获得一个RAM角色的ARN，该角色必须有访问OSS的权限
    private static String recordDelimiter = "\n";
    private static int threadCount = 10;

    public static void main(String[] args) throws Exception {
        ClientBuilderConfiguration config = new ClientBuilderConfigu
ration();
        RoleCredentialProvider provider = new RoleCredentialProvider(
roleArn);
        Credentials credentials = provider.getCredentials();
        //OSS client = new OSSClientBuilder().build(endpoint,
accessKeyId, accessKeySecret, config);
        System.out.println("Id " + credentials.getAccessKeyId());
        System.out.println("Key " + credentials.getSecretAccessKey());
        System.out.println("Token " + credentials.getSecurityToken());
        OSS client = new OSSClientBuilder().build(endpoint, credential
s.getAccessKeyId(), credentials.getSecretAccessKey(), credentials.
getSecurityToken(), config);
        int totalSplits = 1;
        try {
            SelectObjectMetadata selectObjectMetadata = client.
createSelectObjectMetadata(
                    new CreateSelectObjectMetadataRequest(bucketName,
key)
                            .withInputSerialization(
                                    new InputSerialization().
withCsvInputFormat(
                                            new CSVFormat().withHeader
Info(CSVFormat.Header.None).withRecordDelimiter(recordDelimiter))));

            totalSplits = selectObjectMetadata.getCsvObjectMetadata().
getSplits();
            System.out.println(selectObjectMetadata.getCsvObje
ctMetadata().getTotalLines());
            System.out.println(totalSplits);

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

        String sql = "select count(*) from ossobject";

        ExecutorService executor = Executors.newFixedThreadPool(
threadCount);
        long startTime = System.currentTimeMillis();
        List<Future<Integer>> list = new ArrayList<Future<Integer>>();
```

```
        int n = threadCount < totalSplits ? threadCount: totalSplits;
        for(int i = 0; i < n; i++) {
            int start = i * totalSplits/n;
            int end = i == n-1 ? totalSplits - 1 : (i+1)* totalSplits
 /n - 1;
            System.out.println("start:" + start + " end:" + end);
            Callable<Integer> task = new MulipartSelector(client,
bucketName, key, start, end, sql);
            Future<Integer> future = executor.submit(task);
            list.add(future);
        }

        long totalLines = 0;
        for(Future<Integer> task : list){
            totalLines += task.get().longValue();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("total lines:" + totalLines);
        System.out.printf("Total time %dms\n" , (endTime - startTime
));

    }
}
```

常见的SQL用例

- 常见的SQL用例（Csv）

应用场景	SQL语句
返回前10行数据	select * from ossobject limit 10
返回第1列和第3列的整数，并且第1列大于第3列	select _1, _3 from ossobject where cast(_1 as int) > cast(_3 as int)
返回第1列以'陈'开头的记录的个数(注：此处like后的中文需要用UTF-8编码)	select count(*) from ossobject where _1 like '陈%'
返回所有第2列时间大于2018-08-09 11:30:25且第3列大于200的记录	select * from ossobject where _2 > cast('2018-08-09 11:30:25' as timestamp) and _3 > 200
返回第2列浮点数的平均值，总和，最大值，最小值	select AVG(cast(_2 as double)), SUM(cast(_2 as double)), MAX(cast(_2 as double)), MIN(cast(_2 as double))
返回第1列和第3列连接的字符串中以'Tom'为开头以’Anderson‘结尾的所有记录	select * from ossobject where (_1 || _3) like 'Tom%Anderson'
返回第1列能被3整除的所有记录	select * from ossobject where (_1 % 3) == 0
返回第1列大小在1995到2012之间的所有记录	select * from ossobject where _1 between 1995 and 2012

应用场景	SQL语句
返回第5列值为N,M,G,L的所有记录	select * from ossobject where _5 in ('N', 'M', 'G', 'L')
返回第2列乘以第3列比第5列大100以上的所有记录	select * from ossobject where _2 * _3 > _5 + 100

- 常见的SQL用例（Json）

  假如我们有如下Json文档。

```
{
  "contacts":[
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}, …… #此处省略其他类似的节点
]}
```

  SQL用例如下：

应用场景	SQL语句
返回所有age是27的记录	select * from ossobject.contacts[*] s where s.age = 27
返回所有的家庭电话	select s.number from ossobject.contacts[*].phoneNumbers[*] s where s.type = “home”
返回所有单身的记录	select * from ossobject s where s.spouse is null

应用场景	SQL语句
返回所有没有孩子的记录	select * from ossobject s where s.children[0] is null 说明: 目前没有专用的空数组的表示方法，用以上语句代替。

最佳实践

- 大文件分片查询

  如果确定该CSV文件列中不含有换行符，则基于Bytes的分片由于不需要创建Meta，其使用最为简便。如果列中包含换行符或者是Json文件时，则使用以下步骤：

  1. 调用Create Select Object Meta API获得该文件的总的Split数。理想情况下如果该文件需要用SelectObject，则该API最好在查询前进行异步调用，这样可以节省扫描时间。
  2. 根据客户端资源情况选择合适的并发度n，用总的Split数除以并发度n得到每个分片查询应该包含的Split个数。
  3. 在请求Body中用诸如split-range=1-20的形式进行分片查询。
  4. 如果需要最后可以合并结果。
- SelectObject和Normal类型文件配合性能更佳。Multipart 以及Appendable类型的文件由于其内部结构差异导致性能较差。
- 查询Json文件时，在SQL的From 语句中尽可能的缩小From后的Json Path 范围。

  示例：有如下Json文件。

```
{ contacts:[
        {“firstName”:”John”, “lastName”:”Smith”, “phoneNumbers”:[{“
type”:”home”, “number”:”212-555-1234”}, {“type”:”office”, “number
”:”646-555-4567”}, {“type”:”mobile”, “number”:”123 456-7890”}], ”
address”:{“streetAddress”: “21 2nd Street”, “city”:”New York”, “
state”:NY, “postalCode”:”10021-3100”}
         }
```

```
]}
```

如果要查找所有postalCode为 10021 开头的streetAddress，可以这样写SQL：

```
select s.address.streetAddress from ossobject.contacts[*] s where s.address.postalCode like ‘10021%’
```

也可以这样写SQL：

```
select s.streetAddress from ossobject.contacts[*].address s where s.postalCode like ‘10021%’
```

第二个SQL由于Json Path更加精确，所以性能会更好。

- 在Json文件中处理高精度浮点数

  在Json文件中需要进行高精度浮点数的数值计算时，推荐的做法是使用ParseJsonNumberAsString选项为true, 同时将该值cast成Decimal。比如一个属性a值为123456789.123456789，用`select s.a from ossobject s where cast(s.a as decimal) > 123456789.12345`就可以保持原始数据的精度不丢失。

9.3.7 对象标签

OSS支持使用对象标签（Object Tagging）对存储的对象（Object）进行分类，您可以针对同标签的Object设置生命周期规则。

> 说明:
> 使用对象标签会收取一定的费用，详情请参见OSS产品定价。

对象标签使用一组键值对（Key-Value）标记对象，您可以在上传文件时添加标签，也可以为现有文件添加标签。

- 单个文件最多可设置10个标签，Key不可重复。
- 每个Key长度不超过128字节，每个Value长度不超过256字节。
- Key和Value区分大小写。
- 标签合法字符集包括大小写字母、数字、空格和以下符号：

  +-=._:/
- 只有Bucket拥有者和授权用户拥有读写对象标签的权限，该权限#受对象ACL控制。
- 跨区域复制时，对象标签也会复制到目的端。

使用场景

对象标签不受文件目录限制，您可以批量操作拥有指定标签的对象，例如：

- 设置针对指定标签的生命周期规则。例如，周期性生成的非长期保存的文件，可以在上传时设置指定的标签，之后通过生命周期规则，将拥有这个标签的文件定期删除。
- 设置RAM权限，允许访问拥有指定标签的Object。

操作方式

操作方式	说明
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
Go SDK	
C++ SDK	

使用说明

- 对象标签涉及的API接口
  - #unique_281：设置对象的标签。若对象已有标签，则覆盖原标签。
  - #unique_282：读取对象的标签。
  - #unique_283：删除某个对象关联的标签。
  - #unique_284：用户可在上传对象时通过`x-oss-tagging`请求头指定对象标签。
  - #unique_285：用户可在初始化分片上传时，通过`x-oss-tagging`请求头指定对象标签。
  - #unique_286：复制对象时可通过`x-oss-tagging-directive`请求头指定是否复制源对象标签，通过`x-oss-tagging`请求头指定目标对象标签。
  - #unique_287：当用户拥有读取对象标签的权限时，响应头中会带有`x-oss-tagging-count`标识对象的标签个数。
  - #unique_288：当用户拥有读取对象标签的权限时，响应头中会带有`x-oss-tagging-count`标识对象的标签个数。
- 权限说明

  能够进行Tag相关操作的用户、角色、服务，必须具有如下相关权限：
  - GetObjectTagging：获取对象标签的权限。拥有此权限，可以查看到文件的已有标签。
  - PutObjectTagging：设置对象标签的权限。拥有此权限，可以为文件设置标签。
  - DeleteObjectTagging：删除对象标签的权限。拥有此权限，可以删除文件的标签。

对象标签和生命周期管理

在生命周期规则配置中，您可以指定生命周期规则生效的条件。生命周期规则可针对前缀或对象标签生效，您也可以同时指定两者作为条件。

- 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。

示例：

```
<LifecycleConfiguration>
<Rule>
<ID>r1</ID>
<Prefix>rule1</Prefix>
<Tag><Key>xx</Key><Value>1</Value></Tag>
<Tag><Key>yy</Key><Value>2</Value></Tag>
<Status>Enabled</Status>
<Expiration>
<Days>30</Days>
</Expiration>
</Rule>
<Rule>
<ID>r2</ID>
<Prefix>rule2</Prefix>
<Tag><Key>xx</Key><Value>1</Value></Tag>
<Status>Enabled</Status>
<Transition>
<Days>60</Days>
<StorageClass>Archive</StorageClass>
</Transition>
</Rule>
</LifecycleConfiguration>
```

以上规则中：

- 以rule1为前缀且同时拥有标签xx=1和yy=2的对象在30天后被删除。
- 以rule2为前缀且拥有标签xx=1的对象在60天后被转换为Archive（归档存储）。

 说明：

更多信息请参见#unique_289。

对象标签和RAM Policy

您可以授权您的RAM用户拥有管理对象标签权限，也可以授权RAM用户管理拥有指定对象标签的Object的权限。

- 授予RAM用户管理对象标签的权限

  您可以授予RAM用户操作所有对象标签的权限，也可以授予RAM用户仅可以操作指定对象标签的权限。例如，授予用户A可以设置的对象标签为allow=yes，则该用户仅可以给对象增加allow=yes的标签， RAM策略如下。

```
{
  "Version": "1",
  "Statement": [
    {
```

```
      "Action": "oss:PutObjectTagging",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "oss:RequestObjectTag/allow": [
            "yes"
          ]
        }
      }
    }
  ]
}
```

注意:

授予RAM用户可以设置指定对象标签后，该用户仅可针对已有对象设置指定标签，无法在上传对象时设置指定标签。

- **授予RAM用户管理拥有指定标签的对象**

  您可以授权RAM用户操作所有拥有指定标签的Object。例如，授予用户A可以访问对象标签为allow=yes的所有Object，RAM策略如下。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "oss:GetObject",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "oss:ExistingObjectTag/allow": [
            "yes"
          ]
        }
      }
    }
  ]
}
```

10 版本控制介绍

开启存储空间（Bucket）版本控制特性后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。通过文件（Object）的版本控制，用户在错误覆盖或者删除 Object 后，能够将 Bucket 中存储的 Object 恢复至任意时刻的历史版本。

> 说明:
> 版本控制目前已在印度区域支持白名单开通，其他区域即将陆续开通，敬请期待。

应用场景

- 用户数据误删除

  当前 OSS 不提供回收站功能。当用户删除 OSS 数据时，无法找回已删除的数据，必须依赖线下或者第三方的备份产品。

- 文件被覆盖

  针对网盘、在线协作类产品，该类产品里的文档被频繁的修改。在线办公场景下，针对文件的编辑会产生大量的临时版本，用户经常需要找回某个时间点的版本。

原理介绍

版本控制应用于 Bucket 内的所有 Object，而不是某些指定的 Object。当第一次针对 Bucket 开启版本控制后，该 Bucket 中所有的 Object 将在之后一直受到版本控制，并且每个版本都具有唯一的版本 ID。

- Bucket 的版本状态包括：
  - 非版本化（默认）
  - 启用版本控制
  - 暂停版本控制

  > 说明:
  > 开启 Bucket 版本控制后，则无法返回到非版本化状态，但允许暂停版本控制。

- 可以配置 Bucket 版本控制的用户：
  - 根账号
  - 授权 OSSFullAccess 权限的子账号或者角色
  - 授权 PutBucketVersioning 权限的子账号或者角色

说明:

Bucket 开启版本控制后，针对文件的每次覆盖都会生成 一个历史版本，并且针对每个版本进行收费。您可以通过 lifecycle 自动删除过期版本。

11 文件生命周期

11.1 管理文件生命周期

您可以通过OSS的PutBucketLifecycle接口设置生命周期规则（Lifecycle），自动删除过期的对象（Object）和碎片或将到期的Object转储为低频或归档存储类型，从而节省存储费用。

说明:

设置生命周期的API详细信息可参考PutBucketLifecycle。

生命周期规则包含如下信息：

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

    说明:

    对象标签功能正在公测中，您可以联系售后技术支持申请体验此功能。详情请参见对象标签。
  - 按前缀+标签匹配：按指定前缀和一个或多个标签的筛选条件匹配对象。
  - 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。此种方式只能创建一条规则。
- 文件过期策略：设置Object的过期时间及操作。
  - 过期天数：指定一个过期天数N，并指定Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。
  - 过期日期：指定一个过期日期，并指定Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。

    说明:

    针对过期Object可执行的操作为：转换到低频访问型存储、转换到归档型存储、删除。

- 碎片过期策略：设置碎片的过期时间及操作
  - 过期天数：可指定一个过期天数N，文件碎片会在其最后修改时间的的N天后被删除。
  - 过期日期：指定一个过期日期，最后修改时间在该日期之前的文件碎片会被全部删除。

只要Object名称前缀和一条规则的前缀匹配，那么该规则就适用于它。例如，一个Bucket有如下几个Object：

```
logs/program.log.1
logs/program.log.2
logs/program.log.3
doc/readme.txt
```

如果一个规则指定的前缀是logs/，那么该规则就适用于前三个以logs/开头的Object；如果前缀是doc/readme.txt，那么这条规则就只对doc/readme.txt起作用。

规则支持过期删除操作。例如，您可以设置这样的规则：当前缀为logs/的Object的最后一次更新是30天前，就删除它们；也可以指定在某年某月某日删除doc/readme.txt。

当一个Object匹配到某个过期规则，执行GET和HEAD操作时，OSS在响应Header中加入x-oss-expiration头。它包含了两个参数：expiry-date的值表示Object的过期日期；rule-id的值表示相匹配的规则ID。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
Java SDK	丰富、完整的各类语言 SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

API 配置举例

您可以通过OSS接口来设置Bucket的生命周期。生命周期配置为XML格式，举例如下：

```
<LifecycleConfiguration>
<Rule>
<ID>delete logs after 10 days</ID>
<Prefix>logs/</Prefix>
```

```
<Status>Enabled</Status>
<Expiration>
<Days>10</Days>
</Expiration>
</Rule>
<Rule>
<ID>delete doc</ID>
<Prefix>doc/</Prefix>
<Status>Disabled</Status>
<Expiration>
<CreatedBeforeDate>2017-12-31T00:00:00.000Z</CreatedBeforeDate>
</Expiration>
</Rule>
<Rule>
<ID>delete xx=1</ID>
<Prefix>rule2</Prefix>
<Tag><Key>xx</Key><Value>1</Value></Tag>
<Status>Enabled</Status>
<Transition>
<Days>60</Days>
<StorageClass>Archive</StorageClass>
</Transition>
</Rule>
</LifecycleConfiguration>
```

在这个例子中，各个元素的含义如下：

- `<ID>`：每个规则唯一的标识。
- `<Status>`：取值为Enabled或Disabled。OSS只会应用取值为Enabled的生命周期规则。
- `<Prefix>`：前缀。
- `<Expiration>`：过期操作。子元素`<CreatedBeforeDate>`或`<Days>`指定绝对和相对过期时间。
  - CreatedBeforeDate：设置过期时间。指定一个过期日期，并指定Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - Days：设置过期天数。指定一个过期天数N，并指定Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。

在这个例子中，第一条规则会删除前缀为logs/，最后更新时间是10天前的Object。第二条规则虽然指定了删除2017年12月31日之前被修改的前缀为doc/的Object，但是由于它的Status是Disabled状态，所以该规则并不会生效。第三条规则会将标签为xx=1，最后更新时间是60天前的的Object存储类型修改为Archive（归档存储）。

细节分析

- 前缀、标签
  - 前缀的命名规范和Object的命名规范一样。
  - 当前缀为空时，表明该规则适用于Bucket里的所有Object。
  - 任意两个前缀不能有重叠。例如，同一Bucket配置了两条规则，一条前缀是logs/，一条前缀是logs/program，那么OSS会返回错误。
  - 标签允许字符包括大小写字母、数字、空格和符号（+-=._:/），且Key#可为空。
  - 前缀+标签规则可以有前缀重叠。例如，同一Bucket配置了两条规则，规则1前缀是logs/，标签是K1=V1；规则2前缀是logs/program，标签是K1=V2。OSS会对logs/program前缀下标签为K1=V1的对象执行规则1，标签为K1=V2的对象执行规则2，logs/前缀下对象标签为K1=V1的对象执行规则1。
- 规则生效时间
  - 当规则设置为在指定日期删除Object，该日期必须是UTC午夜零点，并且符合形如2017-01-01T00:00:00.000Z的ISO8601格式。OSS会在当前时间超过2017-01-01午夜零点时删除匹配的Object。
  - 当规则设定为天数时，OSS把Object最后更新时间（Last-Modified）加上天数，再取整到下一个UTC午夜零点。例如，一个Object的最后更新时间是UTC的2017年4月12日上午1点，相匹配的规则定义的天数是3天，那么过期时间就是UTC 2017年4月16日0点整。
  - OSS会在指定时间删除与规则相匹配的Object。请注意，通常Object会在指定时间稍稍延后一段时间才被删除。
  - 通常Object的最后更新时间和创建时间相差无几。当一个Object被多次Put时，最后更新时间是最后一次Put的时间；当一个Object被Copy到自身时，最后更新时间是Copy发生时的时间。
- 费用

  成功的生命周期异步请求操作会记录在访问日志中并产生相关的请求次数费用，失败的不会被记录和收费。
- 规则冲突行为
  - 前缀+标签相同

rule	prefix	tag	action
rule1	abc	a=1	20天后删除

rule	prefix	tag	action
rule2	abc	a=1	20天后转为Archive

执行结果：所有前缀为abc，标签为a=1的Object会在20天被删除（优先执行删除操作）。此时，文件已不存在，所以第二条规则已经没有意义。

rule	prefix	tag	action
rule1	abc	a=1	365天后转为IA（低频存储）
rule2	abc	a=1	2018-03-01前转为Archive

执行结果：前缀为abc且标签为a=1的Object如果同时满足两个规则，则转换为Archive；若仅满足其中一条，则按指定规则执行。

- 前缀重叠+标签相同

rule	prefix	tag	action
rule1		a=1	20天后转为IA
rule2	abc	a=1	120天后被删除

执行结果：所有标签为a=1的Object在20天后转为IA，前缀为abc且标签为a=1的Object会在120天后被删除。

rule	prefix	tag	action
rule1		a=1	10天后转为Archive
rule2	abc	a=1	20天后转为IA

执行结果：所有标签为a=1的Object会在10天后被转为Archive，前缀为abc且标签为a=1的Object在20天后转为IA的规则失效，因为Archive的文件无法转为IA。

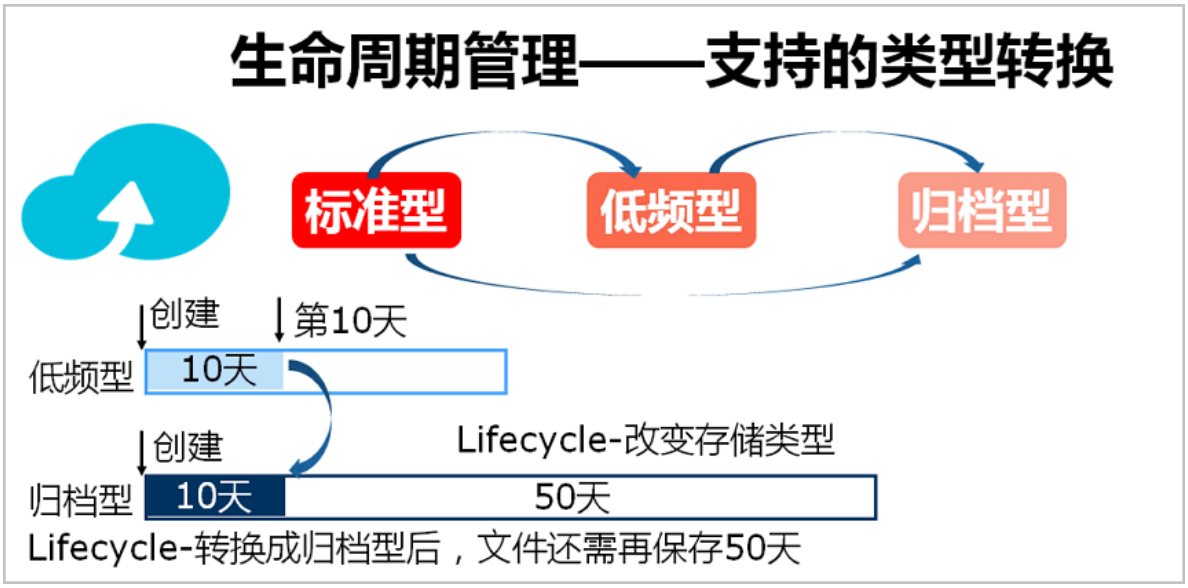
常见问题

- 通过设置生命周期转换文件类型或过期删除，是否有最小存储天数限制?

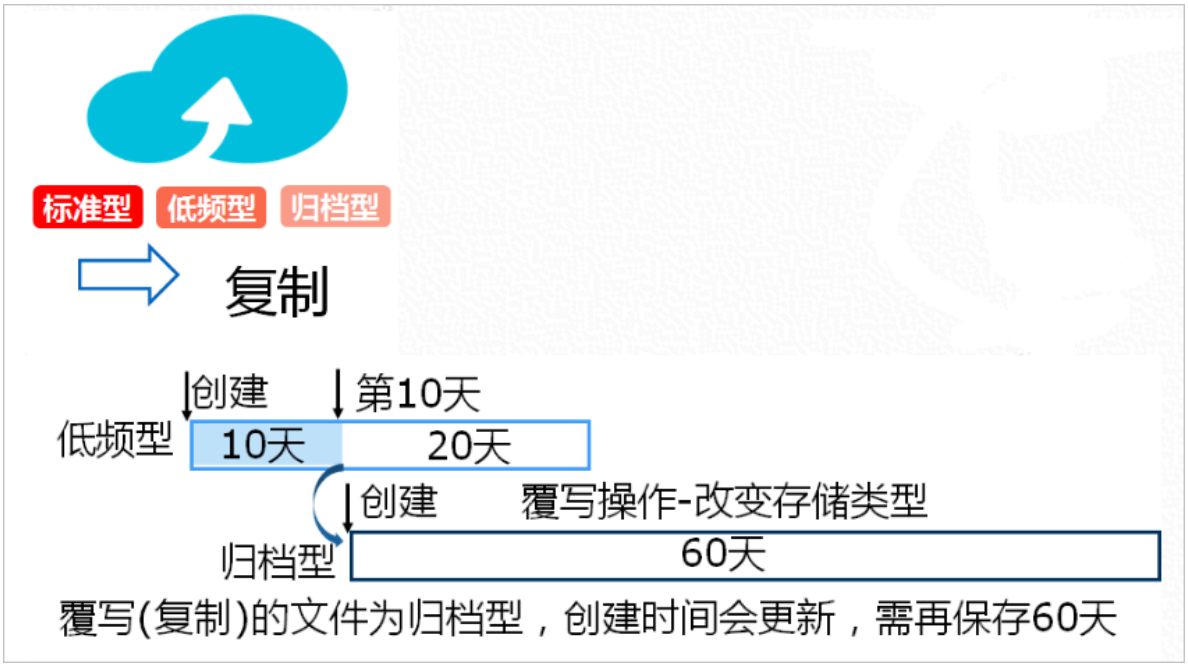
  生命周期的文件类型转换没有最小存储天数限制（从Standard转IA、Archive，或从IA转Archive），但是数据过期删除有最小存储天数限制。数据过期删除最小存储天数按删除时文件存储类型来计算，低频存储文件至少保存30天，归档存储文件至少保存60天。

  数据过期删除的最小存储天数是指从文件被创建开始到被删除时这段时间。若中间修改了文件，如通过CopyObject覆写、Append追加文件等操作时，存储天数从最后修改时间开始计算。

  举例1：一个低频型文件，在创建10天后，通过生命周期转换为归档型。文件的创建时间不会变化，转换后的归档型文件，需再保存至少50天 。



  举例2：一个低频型文件，在创建10天后，通过CopyObject覆写，将存储类型转换为归档型时，会产生20天的低频型文件提前删除费用；文件的创建时间会更新，转换后的归档型文件，需再保存至少60天。



- 请求次数计费逻辑

  生命周期的类型转换或过期删除，会产生请求次数。OSS 按照请求次数收取标准费用。例如：

  - 通过设置生命周期， 1000个Object，从标准转成归档，会产生1000次写类别请求次数（POST）。
  - 通过设置生命周期， 1000个Object，过期删除，会产生1000次写类别请求次数（Delete）。

  详细收费标准请参见计量计费。

- 通过生命周期的类型转换、过期删除，是否有日志记录?

  通过生命周期的类型转换、过期删除，都有日志记录，日志记录字段如下：

  - Operation
    - CommitTransition：类型转换
    - ExpireObject：过期删除
  - Sync Request
    - lifecycle: 生命周期操作

11.2 生命周期配置示例

本文介绍文件生命周期的配置示例。

您可以通过OSS接口来设置Bucket中文件的生命周期。生命周期配置为XML格式，举例如下：

```
<LifecycleConfiguration>
```

```
<Rule>
<ID>delete logs after 10 days</ID>
<Prefix>logs/</Prefix>
<Status>Enabled</Status>
<Expiration>
<Days>10</Days>
</Expiration>
</Rule>
<Rule>
<ID>delete doc</ID>
<Prefix>doc/</Prefix>
<Status>Disabled</Status>
<Expiration>
<CreatedBeforeDate>2017-12-31T00:00:00.000Z</CreatedBeforeDate>
</Expiration>
</Rule>
<Rule>
<ID>delete xx=1</ID>
<Prefix>rule2</Prefix>
<Tag><Key>xx</Key><Value>1</Value></Tag>
<Status>Enabled</Status>
<Transition>
<Days>60</Days>
<StorageClass>Archive</StorageClass>
</Transition>
</Rule>
</LifecycleConfiguration>
```

上述示例中，有三条规则，含义如下：

- 第一条规则会删除前缀为logs/，且最后更新时间是10天前的Object。
- 第二条规则虽然指定了删除2017年12月31日之前被修改的前缀为doc/的Object，但是由于它的Status是Disabled状态，所以该规则并不会生效。
- 第三条规则会将标签为xx=1，且最后更新时间是60天前的的Object存储类型修改为Archive（归档存储）。

配置生命周期涉及的各个元素定义如下：

- <ID>：每个规则唯一的标识。
- <Status>：取值为Enabled或Disabled。OSS只会应用取值为Enabled的生命周期规则。
- <Prefix>：前缀。
- <Expiration>：过期操作。子元素<CreatedBeforeDate>或<Days>指定绝对和相对过期时间。
  - <CreatedBeforeDate>：设置过期时间。指定一个过期日期，并指定Object过期后执行什么操作。最后修改时间在该日期之前的Object全部过期，并执行指定的操作。
  - <Days>：设置过期天数。指定一个过期天数N，并指定Object过期后执行什么操作。Object会在其最后修改时间的N天后过期，并执行指定的操作。

11.3 常见问题

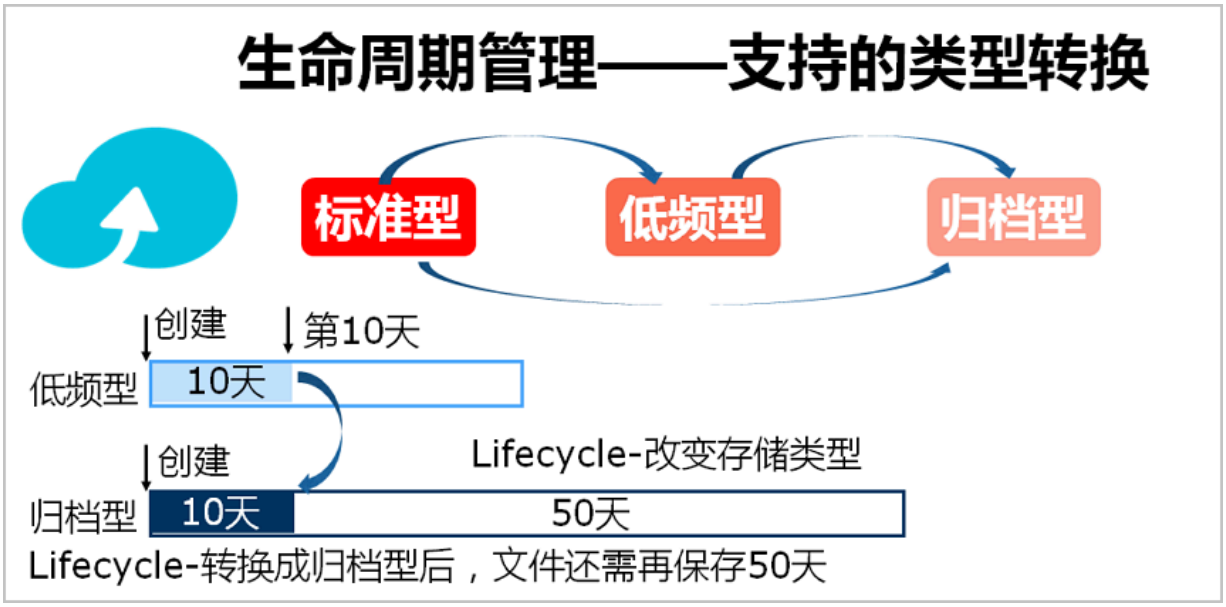
本文主要介绍使用生命周期管理文件时可能遇到的问题。

通过设置生命周期转换文件类型或过期删除，是否有最小存储天数限制?

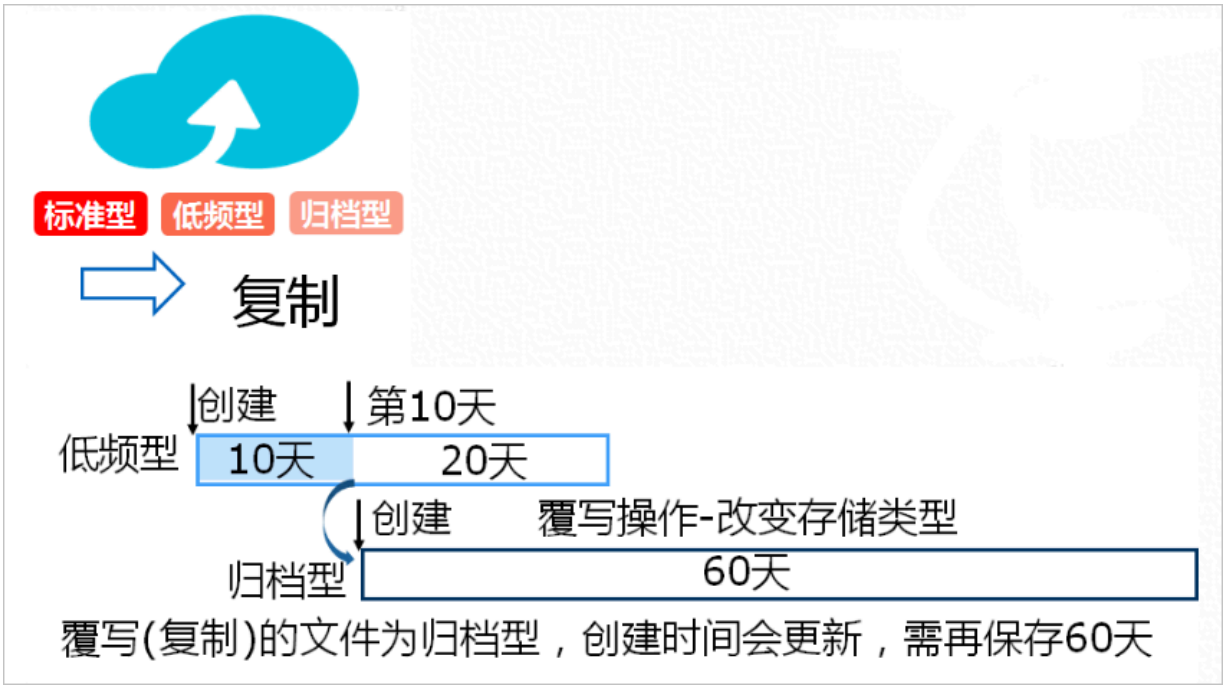
生命周期的文件类型转换没有最小存储天数限制（从Standard转IA、Archive，或从IA转Archive），但是数据过期删除有最小存储天数限制。数据过期删除最小存储天数按删除时文件存储类型来计算，低频存储文件至少保存30天，归档存储文件至少保存60天。

数据过期删除的最小存储天数是指从文件被创建开始到被删除时这段时间。若中间修改了文件，如通过CopyObject覆写、Append追加文件等操作时，存储天数从最后修改时间开始计算。

举例1：一个低频型文件，在创建10天后，通过生命周期转换为归档型。文件的创建时间不会变化，转换后的归档型文件，需再保存至少50天 。



举例2：一个低频型文件，在创建10天后，通过CopyObject覆写，将存储类型转换为归档型时，会产生20天的低频型文件提前删除费用；文件的创建时间会更新，转换后的归档型文件，需再保存至少60天。



请求次数计费逻辑

生命周期的类型转换或过期删除，会产生请求次数。OSS 按照请求次数收取标准费用。例如：

- 通过设置生命周期，1000个Object，从标准转成归档，会产生1000次写类别请求次数（POST）。
- 通过设置生命周期，1000个Object，过期删除，会产生1000次写类别请求次数（Delete）。

详细收费标准请参见计量计费。

通过生命周期的类型转换、过期删除，是否有日志记录?

通过生命周期的类型转换、过期删除，都有日志记录，日志记录字段如下：

- Operation
  - CommitTransition：类型转换
  - ExpireObject：过期删除
- Sync Request

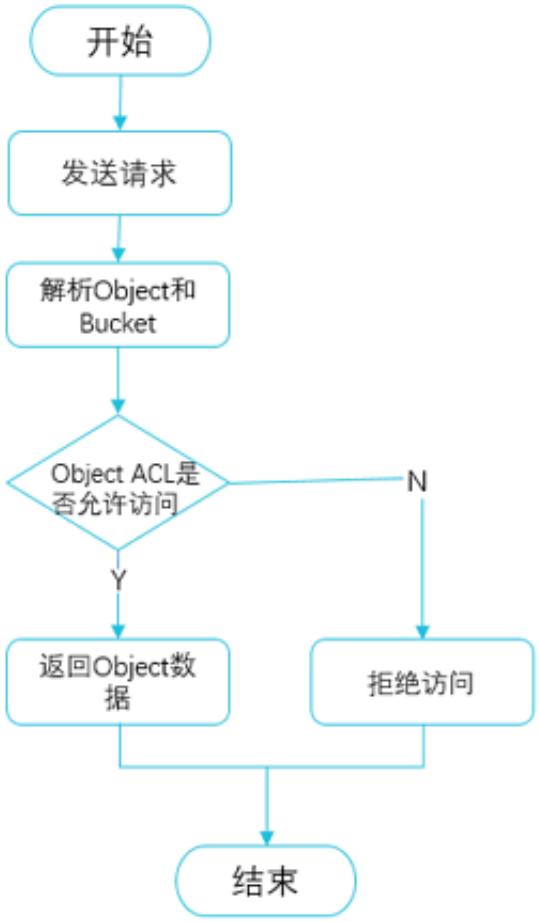
  lifecycle：生命周期操作

12 签名

12.1 OSS请求流程

对OSS的HTTP请求可以根据是否携带身份验证信息分为匿名请求和带身份验证的请求。匿名请求指的是请求中没有携带任何和身份相关的信息；带身份验证的请求指的是按照 OSS API 文档中规定的在请求头部或者在请求 URL 中携带签名的相关信息。

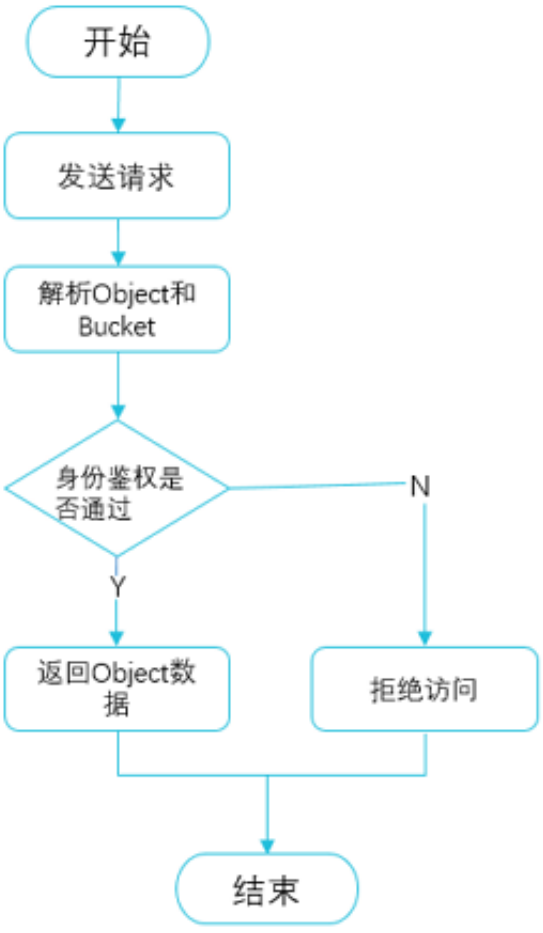
匿名请求流程



1. 用户的请求被发送到OSS的HTTP服务器上。
2. OSS根据URL解析出Bucket和Object。

3. OSS检查Object是否设置了ACL。
   - 如果没有设置ACL，那么继续4。
   - 如果设置了ACL，则判断Object的ACL是否允许匿名用户访问。
     - 允许则跳到5。
     - 不允许则拒绝请求，请求结束。
4. OSS判断Bucket的ACL是否允许匿名用户访问。
   - 允许则继续5。
   - 不允许则返回，请求结束。
5. 权限验证通过，返回Object的内容给用户。

带身份验证的请求流程



1. 用户的请求被发送到OSS的HTTP服务器上。
2. OSS根据URL解析出Bucket和Object。

3. OSS根据请求的OSS的AccessKeyId获取请求者的相关身份信息，进行身份鉴权。
   - 如果未获取成功，则返回，请求结束。
   - 如果获取成功，但请求者不被允许访问此资源，则返回，请求结束。
   - 如果获取成功，但OSS端根据请求的HTTP参数计算的签名和请求发送的签名字符串不匹配，则返回，请求结束。
   - 如果身份鉴权成功，那么继续4。
4. OSS检查Object是否设置了ACL。
   - 如果Object没有设置ACL，那么继续5。
   - 如果Object设置了ACL，OSS判断Object的ACL是否允许此用户访问。
     - 允许则跳到6。
     - 不允许则拒绝请求，请求结束。
5. OSS判断Bucket的ACL是否允许此用户访问。
   - 允许则继续6。
   - 不允许则返回，请求结束。
6. 权限验证通过，返回Object的内容给用户。

AccessKey 类型

目前访问 OSS 使用的 AccessKey（AK）有三种类型，具体如下：

- 阿里云账号 AK

  阿里云账号 AK 特指 Bucket 拥有者的 AK，每个阿里云账号提供的 AccessKey 对拥有的资源有完全的权限。每个阿里云账号能够同时拥有不超过5个 active 或者 inactive 的 AK 对（AccessKeyId 和 AccessKeySecret）。

  用户可以登录AccessKey 管理控制台，申请新增或删除 AK 对。

  每个 AK 对都有active/inactive两种状态。

  - Active 表明用户的 AK 处于激活状态，可以在身份验证的时候使用。
  - Inactive 表明用户的 AK 处于非激活状态，不能在身份验证的时候使用。

  > 注意:
  > 请避免直接使用阿里云账号AccessKey。

- RAM 子账号 AK

  RAM (Resource Access Management) 是阿里云提供的资源访问控制服务。RAM 账号 AK 指的是通过 RAM 被授权的 AK。这组 AK 只能按照 RAM 定义的规则去访问Bucket 里的资

源。通过 RAM，您可以集中管理您的用户（比如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。比如能够限制您的用户只拥有对某一个 Bucket 的读权限。子账号是从属于主账号的，并且这些账号下不能拥有实际的任何资源，所有资源都属于主账号。

- STS 账号 AK

  STS（Security Token Service）是阿里云提供的临时访问凭证服务。STS 账号 AK 指的是通过 STS 颁发的 AK。这组 AK 只能按照 STS 定义的规则去访问 Bucket 里的资源。

身份验证具体实现

目前主要有三种身份验证方式：

- AK 验证
- RAM 验证
- STS 验证

当用户以个人身份向 OSS 发送请求时，其身份验证的实现如下：

1. 用户将发送的请求按照OSS指定的格式生成签名字符串。
2. 用户使用 AccessKeySecret 对签名字符串进行加密产生验证码。
3. OSS 收到请求以后，通过 AccessKeyId 找到对应的 AccessKeySecret，以同样的方法提取签名字符串和验证码。
   - 如果计算出来的验证码和提供的一样即认为该请求是有效的。
   - 否则，OSS 将拒绝处理这次请求，并返回 HTTP 403错误。

带身份验证访问OSS的三种方法

- 使用控制台访问OSS：控制台中对用户隐藏了身份验证的细节，使用控制台访问OSS的用户无需关注细节。更多信息请参见下载文件。
- 使用SDK访问OSS：OSS提供了多种开发语言的SDK，SDK中实现了签名算法，只需要将AccessKey信息作为参数输入即可。详情请参见各语言SDK文档中的访问控制章节，例如Java SDK：使用签名URL进行临时授权、Python SDK：使用签名URL进行临时授权。
- 使用API访问OSS：如果您想用自己喜欢的语言来封装调用RESTful API接口，您需要实现签名算法来计算签名。具体请参见API手册中的在Header中包含签名和在URL中包含签名。

12.2 在Header中包含签名

用户可以在HTTP请求中增加 `Authorization` 的Header来包含签名（Signature）信息，表明这个消息已被授权。

SDK 签名实现

OSS SDK已经实现签名，用户使用OSS SDK不需要关注签名问题。如果您想了解具体语言的签名实现，请参考OSS SDK的代码。OSS SDK签名实现的文件如下表：

SDK	签名实现
Java SDK	OSSRequestSigner.java
Python SDK	auth.py
.Net SDK	OssRequestSigner.cs
PHP SDK	OssClient.php
C SDK	oss_auth.c
JavaScript SDK	client.js
Go SDK	auth.go
Ruby SDK	util.rb
iOS SDK	OSSModel.m
Android SDK	OSSUtils.java

Authorization字段计算的方法

```
Authorization = "OSS " + AccessKeyId + ":" + Signature
Signature = base64(hmac-sha1(AccessKeySecret,
            VERB + "\n"
            + Content-MD5 + "\n"
            + Content-Type + "\n"
            + Date + "\n"
            + CanonicalizedOSSHeaders
            + CanonicalizedResource))
```

- `AccessKeySecret` 表示签名所需的密钥。
- `VERB`表示HTTP 请求的Method，主要有PUT、GET、POST、HEAD、DELETE等。
- `\n` 表示换行符。
- `Content-MD5` 表示请求内容数据的MD5值，对消息内容（不包括头部）计算MD5值获得128比特位数字，对该数字进行base64编码而得到。该请求头可用于消息合法性的检查（消息内容是否与发送时一致），如”eB5eJF1ptWaXm4bijSPyxw==”，也可以为空。详情请参见RFC2616 Content-MD5。
- `Content-Type` 表示请求内容的类型，如”application/octet-stream”，也可以为空。
- `Date` 表示此次操作的时间，且必须为GMT格式，如”Sun, 22 Nov 2015 08:16:38 GMT”。
- `CanonicalizedOSSHeaders` 表示以 x-oss- 为前缀的HTTP Header的字典序排列。
- `CanonicalizedResource` 表示用户想要访问的OSS资源。

其中，Date和CanonicalizedResource不能为空；如果请求中的Date时间和OSS服务器的时间差15分钟以上，OSS服务器将拒绝该服务，并返回HTTP 403错误。

构建CanonicalizedOSSHeaders的方法

所有以 x-oss- 为前缀的HTTP Header被称为CanonicalizedOSSHeaders。它的构建方法如下：

1. 将所有以 x-oss- 为前缀的HTTP请求头的名字转换成小写 。如`X-OSS-Meta-Name: TaoBao`转换成`x-oss-meta-name: TaoBao`。
2. 如果请求是以STS获得的AccessKeyId和AccessKeySecret发送时，还需要将获得的security-token值以 `x-oss-security-token:security-token` 的形式加入到签名字符串中。
3. 将上一步得到的所有HTTP请求头按照名字的字典序进行升序排列。
4. 删除请求头和内容之间分隔符两端出现的任何空格。如`x-oss-meta-name: TaoBao`转换成：`x-oss-meta-name:TaoBao`。
5. 将每一个头和内容用 `\n` 分隔符分隔拼成最后的CanonicalizedOSSHeaders。

说明：

- CanonicalizedOSSHeaders可以为空，无需添加最后的 `\n`。
- 如果只有一个，则如 `x-oss-meta-a\n`，注意最后的`\n`。
- 如果有多个，则如 `x-oss-meta-a:a\nx-oss-meta-b:b\nx-oss-meta-c:c\n`，注意最后的`\n`。

构建CanonicalizedResource的方法

用户发送请求中想访问的OSS目标资源被称为CanonicalizedResource。它的构建方法如下：

1. 将CanonicalizedResource置成空字符串 `""`。
2. 放入要访问的OSS资源 `/BucketName/ObjectName`（如果没有ObjectName则CanonicalizedResource为”/BucketName/“，如果同时也没有BucketName则为“/”）。
3. 如果请求的资源包括子资源(SubResource)，那么将所有的子资源按照字典序，从小到大排列并以 & 为分隔符生成子资源字符串。在CanonicalizedResource字符串尾添加 ? 和子资源字符串。此时的CanonicalizedResource如：`/BucketName/ObjectName?acl&uploadId=UploadId`。

说明：

- OSS目前支持的子资源(sub-resource)包括：acl，uploads，location，cors，logging，website，referer，lifecycle，delete，append，tagging，objectMeta，uploadId，partNumber，security-token，position，img，style，styleName，replication，replicationProgress，replicationLocation，cname，bucketInfo，comp，qos，live，status，vod，startTime，endTime，symlink，x-oss-process，response-content-type，response-content-language，response-expires，response-cache-control，response-content-disposition，response-content-encoding等。
- 子资源(sub-resource)有三种类型：
  - 资源标识，如子资源中的acl、append、uploadId、symlink等，详见关于Bucket的操作和关于Object的操作。
  - 指定返回Header字段，如 `response-***`，详见GetObject的`Request Parameters`。
  - 文件（Object）处理方式，如 `x-oss-process`，用于文件的处理方式，如图片处理。

计算签名头规则

- 签名的字符串必须为 `UTF-8` 格式。含有中文字符的签名字符串必须先进行 `UTF-8` 编码，再与`AccessKeySecret`计算最终签名。
- 签名的方法用RFC 2104中定义的HMAC-SHA1方法，其中Key为 AccessKeySecret。
- `Content-Type` 和 `Content-MD5` 在请求中不是必须的，但是如果请求需要签名验证，空值的话以换行符 `\n` 代替。
- 在所有非HTTP标准定义的header中，只有以 `x-oss-` 开头的header，需要加入签名字符串；其他非HTTP标准header将被OSS忽略（如下方签名示例中的x-oss-magic是需要加入签名字符串的）。
- 以 `x-oss-` 开头的header在签名验证前需要符合以下规范：
  - header的名字需要变成小写。
  - header按字典序自小到大排序。
  - 分割header name和value的冒号前后不能有空格。
  - 每个Header之后都有一个换行符“\n”，如果没有Header，CanonicalizedOSSHeaders就设置为空。

签名示例

请求	签名字符串计算公式	签名字符串
PUT /nelson HTTP/1.0 Content-MD5: eB5eJF1ptWaXm4bijSPyxw== Content-Type: text/html Date: Thu, 17 Nov 2005 18:49:58 GMT Host: oss-example.oss-cn-hangzhou.aliyuncs.com X-OSS-Meta-Author: foo@bar.com X-OSS-Magic: abracadabra	Signature = base64(hmac-sha1(AccessKeySecret,VERB + “\n” + Content-MD5 + “\n” + Content-Type + “\n” + Date + “\n” + CanonicalizedOSSHeaders + CanonicalizedResource))	“PUT\n eB5eJF1ptWaXm4bijSPyxw==\n text/html\n Thu, 17 Nov 2005 18:49:58 GMT\n x-oss-magic:abracadabra\nx-oss-meta-author:foo@bar.com\n/oss-example/nels

假如AccessKeyId是“44CF959******252F707”，AccessKeySecret是“OtxrzxIsfpFjA7Sw******8Bw21TLhquhboDYROV”，可用以下方法计算签名(Signature)：

python示例代码：

```
import base64
import hmac
import sha
h = hmac.new("OtxrzxIsfpFjA7Sw******8Bw21TLhquhboDYROV",
             "PUT\nODBGOERFMDMzQTczRUY3NUE3NzA5QzdFNUYzMDQxNEM=\ntext
/html\nThu, 17 Nov 2005 18:49:58 GMT\nx-oss-magic:abracadabra\nx-oss-
meta-author:foo@bar.com\n/oss-example/nelson", sha)
Signature = base64.b64encode(h.digest())
print("Signature: %s" % Signature)
```

签名(Signature)计算结果应该为 26NBxoKd******Dv6inkoDft/yA=，因为Authorization = “OSS”+ AccessKeyId + “:” + Signature，所以最后Authorization为 “OSS 44CF95900***BF252F707:26NBxoKd******Dv6inkoDft/yA=”，然后加上Authorization头来组成最后需要发送的消息：

```
PUT /nelson HTTP/1.0
Authorization:OSS 44CF95900***BF252F707:26NBxoKd******Dv6inkoDft/yA=
Content-Md5: eB5eJF1ptWaXm4bijSPyxw==
Content-Type: text/html
Date: Thu, 17 Nov 2005 18:49:58 GMT
Host: oss-example.oss-cn-hangzhou.aliyuncs.com
X-OSS-Meta-Author: foo@bar.com
X-OSS-Magic: abracadabra
```

细节分析如下：

- 如果传入的AccessKeyId不存在或inactive，返回403 Forbidden。错误码：InvalidAccessKeyId。

- 若用户请求头中Authorization值的格式不对，返回400 Bad Request。错误码：InvalidArgument。
- OSS所有的请求都必须使用HTTP 1.1协议规定的GMT时间格式。其中，日期的格式为：

```
date1 = 2DIGIT SP month SP 4DIGIT; day month year (e.g., 02 Jun
        1982)
```

上述日期格式中，“天”所占位数都是“2 DIGIT”。因此，“Jun 2”、“2 Jun 1982”和“2-Jun-82”都是非法日期格式。

- 如果签名验证的时候，头中没有传入Date或者格式不正确，返回403 Forbidden错误。错误码：AccessDenied。
- 传入请求的时间必须在OSS服务器当前时间之后的15分钟以内，否则返回403 Forbidden。错误码：RequestTimeTooSkewed。
- 如果AccessKeyId是active的，但OSS判断用户的请求发生签名错误，则返回403 Forbidden，并在返回给用户的response中告诉用户正确的用于验证加密的签名字符串。用户可以根据OSS的response来检查自己的签名字符串是否正确。

返回示例：

```
<?xml version="1.0" ?>
<Error>
 <Code>
     SignatureDoesNotMatch
 </Code>
 <Message>
     The request signature we calculated does not match the
signature you provided. Check your key and signing method.
 </Message>
 <StringToSignBytes>
     47 45 54 0a 0a 0a 57 65 64 2c 20 31 31 20 4d 61 79 20 32 30 31
31 20 30 37 3a 35 39 3a 32 35 20 47 4d 54 0a 2f 75 73 72 65 61 6c 74
 65 73 74 3f 61 63 6c
 </StringToSignBytes>
 <RequestId>
     1E446260FF9B10C2
 </RequestId>
 <HostId>
     oss-cn-hangzhou.aliyuncs.com
 </HostId>
 <SignatureProvided>
     y5H7yzPsA/tP4+0tH1HHvPEwUv8=
 </SignatureProvided>
 <StringToSign>
     GET
Wed, 11 May 2011 07:59:25 GMT
/oss-example?acl
 </StringToSign>
 <OSSAccessKeyId>
     AKIAIVAKMSMOY7VOMRWQ
 </OSSAccessKeyId>
```

```
</Error>
```

常见问题

Content-MD5的计算方法

```
Content-MD5的计算
以消息内容为"123456789"来说，计算这个字符串的Content-MD5
正确的计算方式：
标准中定义的算法简单点说就是：
1．先计算MD5加密的二进制数组（128位）。
2．再对这个二进制进行base64编码（而不是对32位字符串编码）。
以Python为例子：
正确计算的代码为：
>>> import base64,hashlib
>>> hash = hashlib.md5()
>>> hash.update("0123456789")
>>> base64.b64encode(hash.digest())
'eB5eJF1ptWaXm4bijSPyxw=='
需要注意
正确的是：hash.digest()，计算出进制数组（128位）
>>> hash.digest()
'x\x1e^$]i\xb5f\x97\x9b\x86\xe2\x8d#\xf2\xc7'
常见错误是直接对计算出的32位字符串编码进行base64编码。
例如，错误的是：hash.hexdigest()，计算得到可见的32位字符串编码
>>> hash.hexdigest()
'781e5e245d69b566979b86e28d23f2c7'
错误的MD5值进行base64编码后的结果：
>>> base64.b64encode(hash.hexdigest())
'NzgxZTVlMjQ1ZDY5YjU2Njk3OWI4NmUyOGQyM2YyYzc='
```

12.3 在URL中包含签名

除了使用Authorization Header，用户还可以在URL中加入签名信息，这样用户就可以把该URL转给第三方实现授权访问。

示例代码

URL中添加签名的python示例代码：

```
import base64
import hmac
import sha
import urllib
h = hmac.new("OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV",
             "GET\n\n\n1141889120\n/oss-example/oss-api.pdf",
             sha)
urllib.quote (base64.encodestring(h.digest()).strip())
```

OSS SDK中提供了实现URL签名的方法，使用方法请参看SDK参考中的授权访问文档。

OSS SDK的URL签名实现，请参看下表：

SDK	URL签名方法	实现文件
Java SDK	OSSClient.generatePresignedUrl	OSSClient.java
Python SDK	Bucket.sign_url	api.py
.Net SDK	OssClient.GeneratePresignedUri	OssClient.cs
PHP SDK	OssClient.signUrl	OssClient.php
JavaScript SDK	signatureUrl	Object.js
C SDK	oss_gen_signed_url	oss_object.c

实现方式

URL签名示例:

```
http://oss-example.oss-cn-hangzhou.aliyuncs.com/oss-api.pdf?OSSAccessKeyId=nz2pc56s936**9l&Expires=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D
```

URL签名必须至少包含Signature、Expires和OSSAccessKeyId三个参数。

- `Expires` 参数的值是一个Unix time（自UTC时间1970年1月1号开始的秒数），用于标识该URL的超时时间。如果OSS接收到这个URL请求的时候晚于签名中包含的Expires参数时，则返回请求超时的错误码。例如：当前时间是1141889060，开发者希望创建一个60秒后自动失效的URL，则可以设置Expires时间为1141889120。URL的有效时间默认为3600秒，最大为64800秒。
- `OSSAccessKeyId` 即密钥中的AccessKeyId。
- `Signature` 表示签名信息。所有的OSS支持的请求和各种Header参数，在URL中进行签名的算法和在Header中包含签名的算法基本一样。

```
Signature = urlencode(base64(hmac-sha1(AccessKeySecret,
          VERB + "\n"
          + CONTENT-MD5 + "\n"
          + CONTENT-TYPE + "\n"
          + EXPIRES + "\n"
          + CanonicalizedOSSHeaders
```

```
            + CanonicalizedResource)))
```

与Header中包含签名相比主要区别如下：

- 通过URL包含签名时，之前的Date参数换成Expires参数。
- 不支持同时在URL和Header中包含签名。
- 如果多次传入Signature、Expires、OSSAccessKey参数Id，以第一次为准。
- 先验证请求时间是否晚于Expires时间，然后再验证签名。
- 将签名字符串放到URL时，注意要对URL进行urlencode。

· 临时用户URL签名时，需要携带`security-token`，格式如下：

```
http://oss-example.oss-cn-hangzhou.aliyuncs.com/oss-api.pdf?
OSSAccessKeyId=nz2pc56s936**9l&Expires=1141889120&Signature=
vjbyPxybdZaNmGa%2ByT272YEAiv4%3D&security-token=SecurityToken
```

细节分析

· 使用在URL中签名的方式，会将你授权的数据在过期时间内曝露在互联网上，请预先评估使用风险。
· PUT和GET请求都支持在URL中签名。
· 在URL中添加签名时，Signature，Expires，OSSAccessKeyId顺序可以交换，但是如果Signature，Expires，OSSAccessKeyId缺少其中的一个或者多个，返回403 Forbidden消息。错误码：AccessDenied。
· 如果访问的当前时间晚于请求中设定的Expires时间或时间格式错误，返回403 Forbidden消息。错误码：AccessDenied。
· 如果URL中包含参数Signature，Expires，OSSAccessKeyId中的一个或者多个，并且Header中也包含签名消息，返回400 Bad Request消息 。错误码：InvalidArgument。
· 生成签名字符串时，除Date被替换成Expires参数外，仍然包含content-type、content-md5等上节中定义的Header（请求中虽然仍有Date这个请求Header，但不需要将Date加入签名字符串中）。

13 身份认证

13.1 RAM和STS介绍

RAM （Resource Access Management）和STS（Security Token Service）是阿里云提供的权限管理系统。

RAM主要的作用是控制账号系统的权限。您可以使用RAM在主账号的权限范围内创建子用户，给不同的子用户分配不同的权限从而达到授权管理的目的。STS是一个安全凭证（Token）的管理系统。您可以使用STS来完成对于临时用户的访问授权。

为什么要使用RAM和STS

RAM和STS需要解决的一个核心问题是如何在不暴露主账号的AccessKey的情况下安全的授权别人访问。因为一旦主账号的AccessKey暴露出去的话会带来极大的安全风险，别人可以随意操作该账号下所有的资源，盗取重要信息等。

RAM提供一种长期有效的权限控制机制，通过分出不同权限的子账号，将不同的权限分给不同的用户，这样一旦子账号泄露也不会造成全局的信息泄露。但是，由于子账号在一般情况下是长期有效的，因此，子账号的AccessKey也是不能泄露的。

相对于RAM提供的长效控制机制，STS提供的是一种临时访问授权。通过STS可以返回临时的AccessKey和Token，这些信息可以直接发给临时用户用来访问OSS。一般来说，从STS获取的权限会受到更加严格的限制，并且拥有时间限制，因此这些信息泄露之后对于系统的影响也很小。

基本概念

以下是一些基本概念的简单解释：

- 子账号（RAM account）：从阿里云的主账号中创建出来的子账号，在创建的时候可以分配独立的密码和权限，每个子账号拥有自己AccessKey，可以和阿里云主账号一样正常的完成有权限的操作。一般来说，这里的子账号可以理解为具有某种权限的用户，可以被认为是一个具有某些权限的操作发起者。
- 角色（Role）：表示某种操作权限的虚拟概念，但是没有独立的登录密码和AccessKey。

  说明：

  子账号可以扮演角色，扮演角色时候的权限是该角色自身的权限。
- 授权策略（Policy）：用来定义权限的规则，比如允许用户读取或写入某些资源。

- 资源（Resource）：代表用户可访问的云资源，比如OSS所有的Bucket、OSS的某个Bucket，或OSS的某个Bucket下面的某个Object等。

子账号和角色可以类比为某个个人和其身份的关系，某人在公司的角色是员工，在家里的角色是父亲，在不同的场景扮演不同的角色，但是还是同一个人。在扮演不同的角色的时候也就拥有对应角色的权限。单独的员工或者父亲概念并不能作为一个操作的实体，只有有人扮演了之后才是一个完整的概念。这里还可以体现一个重要的概念，那就是角色可以被多个不同的个人同时扮演。

说明:

完成角色扮演之后，该个人就自动拥有该角色的所有权限。

这里再用一个例子解释一下。

- 某个阿里云用户，名为Alice，其在OSS下有两个私有的Bucket，alice_a和alice_b。alice对这两个Bucket都拥有完全的权限。
- 为了避免阿里云账号的AccessKey泄露导致安全风险，Alice使用RAM创建了两个子账号bob和carol，bob对alice_a拥有读写权限，carol对alice_b拥有读写权限。bob和carol都拥有独立的AccessKey，这样万一泄露了也只会影响其中一个Bucket，而且Alice可以很方便的在控制台取消泄露用户的权限。
- 现在因为某些原因，需要授权给别人读取alice_a中的Object，这种情况下不应该直接把bob的AccessKey透露出去，那么，这个时候可以新建一个角色，比如AliceAReader，给这个角色赋予读取alice_a的权限。但是请注意，这个时候AliceAReader还是没法直接用的，因为并不存在对应AliceAReader的AccessKey，AliceAReader现在仅仅表示一个拥有访问alice_a权限的一个虚拟实体。
- 为了能获取临时授权，这个时候可以调用STS的AssumeRole接口，告诉STS，bob将要扮演AliceAReader这个角色，如果成功，STS会返回一个临时的AccessKeyId、AccessKeySecret还有SecurityToken作为访问凭证。将这个凭证发给需要访问的临时用户就可以获得访问alice_a的临时权限了。凭证过期的时间在调用AssumeRole的时候指定。

13.2 RAM子账号

在您的阿里云账号下面，通过 RAM 可以创建具有自己 AccessKey 的子用户。您的阿里云账号被称为主账号，创建出来的账号被称为子账号，使用子账号的 AccessKey 只能使用主账号授权的操作和资源。

使用场景

如果您购买了云资源，您的组织里有多个用户需要使用这些云资源，这些用户只能共享使用您的云账号 AccessKey。这里有两个问题：

- 您的密钥由多人共享，泄露的风险很高。
- 您无法控制特定用户能访问哪些资源（比如 Bucket）的权限。

此时，您可以创建 RMA 子账号，并授予子账号对应的权限。之后，让您的用户通过子账号访问或管理您的资源。

具体实现

关于RAM的详细介绍和RAM子账号的创建，请参见RAM用户指南。通过构建RAM Policy实现对OSS资源访问的授权，详情请参见RAM Policy。

13.3 STS临时授权访问OSS

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。

> 说明:
> 关于STS的更多信息，请参见RAM和STS介绍。

使用场景

对于您本地身份系统所管理的用户，比如您的 App 的用户、您的企业本地账号、第三方 App的用户，将这部分用户称为联盟用户。此外，联盟用户还可以是您创建的能访问您的阿里云资源应用程序的用户。这些联盟用户可能需要直接访问 OSS 资源。

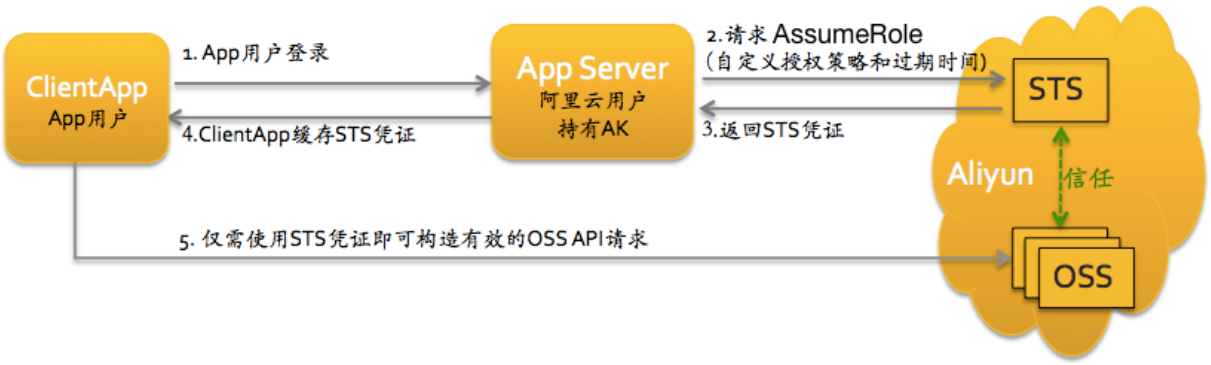
对于这部分联盟用户，通过阿里云 STS 服务为阿里云账号（或 RAM 用户）提供短期访问权限管理。您不需要透露云账号（或 RAM 用户）的长期密钥（如登录密码、AccessKey），只需要生成一个短期访问凭证给联盟用户使用即可。这个凭证的访问权限及有效期限都可以由您自定义。您不需要关心权限撤销问题，访问凭证过期后会自动失效。

通过 STS 生成的凭证包括安全令牌（SecurityToken）、临时访问密钥（AccessKeyId, AccessKeySecret）。使用AccessKey 方法与您在使用阿里云账户或 RAM 用户 AccessKey 发送请求时的方法相同。需要注意的是在每个向 OSS 发送的请求中必须携带安全令牌。

实现原理

以一个移动 App 举例。假设您是一个移动 App 开发者，打算使用阿里云 OSS 服务来保存App 的终端用户数据，并且要保证每个 App 用户之间的数据隔离，防止一个 App 用户获取到其它 App 用户的数据。你可以使用 STS 授权用户直接访问 OSS。

使用 STS 授权用户直接访问 OSS 的流程如下：



1. App 用户登录。App 用户和云账号无关，它是 App 的终端用户，AppServer 支持 App 用户登录。对于每个有效的 App 用户来说，需要 AppServer 能定义出每个 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。然后通过扮演角色（AssumeRole）来获取一个代表角色身份的安全令牌。
3. STS 返回给 AppServer 一个有效的访问凭证，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 将访问凭证返回给 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。比如，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 使用本地缓存的访问凭证去请求 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，正确响应用户请求。

STS 安全令牌、角色管理和使用相关内容详情，请参考 RAM 角色管理。调用 STS 服务接口AssumeRole来获取有效访问凭证即可。

操作步骤

假设有一个名为 ram-test 的 Bucket 用于存储用户数据，现将利用用户子账号结合 STS 实现 OSS 权限控制访问。

您可以通过 OSS SDK 与 STS SDK 的结合使用，实现使用 STS 临时授权访问 OSS 实例。

1. 创建用户子账号。

   a. 登录 RAM 访问控制管理控制台。

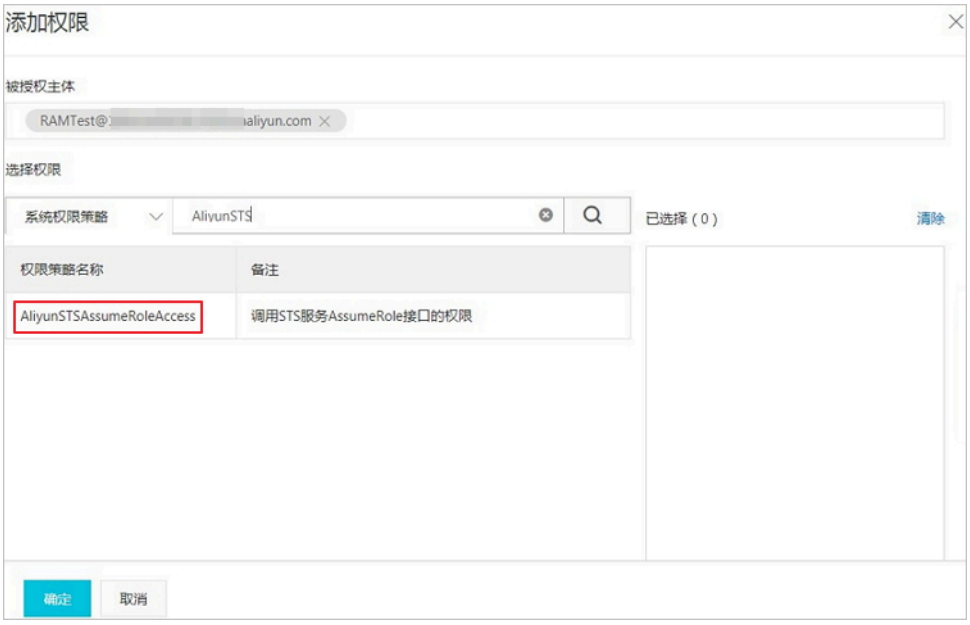
   b. 在 RAM 访问控制页面，单击用户。

   c. 在用户页面，单击新建用户。

   d. 在新建用户页面，用户账号信息填写登录名称、显示名称，访问方式下勾选编程访问，并单击确定。

   e. 单击权限管理 > 添加权限。

   f. 在添加权限页面，为已创建子账号添加 AliyunSTSAssumeRoleAccess 权限。



说明:

尽量不要赋予子账号其他任意权限，因为在扮演角色的时候会自动获得被扮演角色的所有(部分)权限。

2. **创建权限策略。**

   a. **登录 RAM 访问控制管理控制台。**

   b. **在 RAM 访问控制页面，单击权限策略管理。**

   c. **单击新建权限策略。**

   d. **在新建自定义权限策略页面，填写策略名称、备注，配置模式选择可视化配置或脚本配置。**

   以脚本配置为例，对 ram-test 添加 ListObjects 与 GetObject 等只读权限，在策略内容中配置脚本如下:

```
{
 "Version": "1",
 "Statement": [
  {
      "Effect": "Allow",
      "Action": [
        "oss:ListObjects",
        "oss:GetObject"
      ],
      "Resource": [
        "acs:oss:*:*:ram-test",
        "acs:oss:*:*:ram-test/*"
      ]
  }
 ]
```

```
}
```


新建自定义权限策略
策略名称
RAMtest
备注
Fortest
配置模式
可视化配置
脚本配置
策略内容
导入已有系统策略
{
"Version": "1",
"Statement": [
{
"Effect": "Allow",
"Action": [
"oss:ListObjects",
"oss:GetObject"
],
"Resource": [
确定
返回

3. 创建角色。

   a. 登录 RAM 访问控制管理控制台。

   b. 在 RAM 访问控制页面，单击RAM 角色管理。

   c. 在 RAM 角色管理页面，单击新建 RAM 角色。

   d. 在新建 RAM 角色页面，填写 RAM 角色名称，本示例 RAM 角色名称为 RamOssTest，选择可信实体类型及受信云账号 ID 保留默认选项。

   e. 单击已创建 RAM 角色 RamOssTest 右侧对应的添加权限 。

   f. 在添加权限页面，选择自定义权限策略，添加步骤 2 中创建的策略 Ramtest。

   添加策略后，页面如下图所示：

说明:

ARN 代表需要扮演角色的 ID。

4. 通过 STS API 获取 STS AK 与 SecurityToken。

   通过调用 STS SDK 请求 STS 服务获取一个安全令牌。STS SDK 的安装及使用详见STS Java SDK安装及使用。

   以 STS Java SDK 为例:

```
public class StsServiceSample {
    public static void main(String[] args) {
        String endpoint = "sts.aliyuncs.com";
        String accessKeyId = "<access-key-id>";
        String accessKeySecret = "<access-key-secret>";
        String roleArn = "<role-arn>";
        String roleSessionName = "session-name";
        String policy = "{\n" +
                "    \"Version\": \"1\", \n" +
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Action\": [\n" +
                "                \"oss:*\"\n" +
                "            ], \n" +
                "            \"Resource\": [\n" +
                "                \"acs:oss:*:*:*\" \n" +
                "            ], \n" +
                "            \"Effect\": \"Allow\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
        try {
            // 添加endpoint（直接使用STS endpoint，前两个参数留空，无需添加
region ID)
            DefaultProfile.addEndpoint("", "", "Sts", endpoint);
            // 构造default profile（参数留空，无需添加region ID)
            IClientProfile profile = DefaultProfile.getProfile("",
accessKeyId, accessKeySecret);
            // 用profile构造client
            DefaultAcsClient client = new DefaultAcsClient(profile);
            final AssumeRoleRequest request = new AssumeRoleRequest
();
            request.setMethod(MethodType.POST);
            request.setRoleArn(roleArn);
```

```
            request.setRoleSessionName(roleSessionName);
            request.setPolicy(policy); // 若policy为空，则用户将获得该角
色下所有权限
            request.setDurationSeconds(1000L); // 设置凭证有效时间
            final AssumeRoleResponse response = client.getAcsResp
onse(request);
            System.out.println("Expiration: " + response.getCredent
ials().getExpiration());
            System.out.println("Access Key Id: " + response.
getCredentials().getAccessKeyId());
            System.out.println("Access Key Secret: " + response.
getCredentials().getAccessKeySecret());
            System.out.println("Security Token: " + response.
getCredentials().getSecurityToken());
            System.out.println("RequestId: " + response.getRequestId
());
        } catch (ClientException e) {
            System.out.println("Failed: ");
            System.out.println("Error code: " + e.getErrCode());
            System.out.println("Error message: " + e.getErrMsg());
            System.out.println("RequestId: " + e.getRequestId());
        }
    }
}
```

参数说明如下：

- AccessKeyId、AccessKeySecret：子账号 AK 信息。
- RoleArn：需要扮演的角色 ID。
- RoleSessionName：用来标识临时凭证的名称，建议使用不同的应用程序用户来区分。
- Policy：在扮演角色的时候额外添加的权限限制。

> **说明：**
> 这里传入的 Policy 是用来限制扮演角色之后的临时凭证的权限。临时凭证最后获得的权限是角色的权限和这里传入的 Policy 的交集。扮演角色的时候传入Policy的原因是为了灵活性，比如上传文件的时候可以根据不同的用户添加对于上传文件路径的限制等。

- DurationSeconds：设置临时凭证的有效期，单位是 s，最小为 900，最大为 3600。

5. 通过 STS AK 与 SecurityToken 访问 OSS。

   获取 STS AK 与 SecurityToken 后，您可以使用 STS 凭证构造签名请求。

   以下是 Java SDK 的示例，其他示例请参见Python、PHP、Go SDK 授权访问的“使用 STS 进行临时授权”一节。

```
// 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。注意，这里使用到了上一步生成的STS凭证
OSSClient ossClient = new OSSClient(endpoint, accessKeyId,
accessKeySecret, securityToken);

// OSS操作。

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

14 权限控制

14.1 权限控制概述

针对存放在 Bucket 的 Object 的访问，OSS 提供了多种权限控制方式，包括ACL、RAM Policy和Bucket Policy。

- ACL：OSS 为权限控制提供访问控制列表（ACL）。ACL是基于资源的授权策略，可授予Bucket和Object访问权限。 您可以在创建Bucket或上传Object时设置ACL，也可以在创建Bucket或上传Object后的任意时间内修改ACL。
- RAM Policy：RAM （Resource Access Management）是阿里云提供的资源访问控制服务。RAM Policy是基于用户的授权策略。通过设置RAM Policy，您可以集中管理您的用户（比如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。比如能够限制您的用户只拥有对某一个 Bucket 的读权限。子账号是从属于主账号的，并且这些账号下不能拥有实际的任何资源，所有资源都属于主账号。
- Bucket Policy：Bucket Policy是基于资源的授权策略。相比于RAM Policy，Bucket Policy操作简单，支持在控制台直接进行图形化配置，并且Bucket拥有者直接可以进行访问授权，无需具备RAM操作权限。Bucket Policy支持向其他账号的RAM用户授予访问权限，以及向匿名用户授予带特定IP条件限制的访问权限。

14.2 基于读写权限ACL的权限控制

OSS 为权限控制提供访问控制列表（ACL）。ACL 是授予 Bucket 和 Object 访问权限的访问策略。 您可以在创建存储空间（Bucket）或上传文件（Object）时配置 ACL，也可以在创建 Bucket 或上传 Object 后的任意时间内修改 ACL。

说明:

基于读写权限 ACL 的 OSS API 接口详细信息请参考：

- 设置 Bucket ACL：PutBucketACL
- 获取 Bucket ACL：GetBucketACL
- 设置 Object ACL：PutObjectACL
- 获取 Object ACL：GetObjectACL

Bucket ACL

- Bucket ACL 介绍

  Bucket ACL是 Bucket 级别的权限访问控制。目前有三种访问权限：public-read-write，public-read 和 private，含义如下：

权限值	中文名称	权限对访问者的限制
public-read-write	公共读写	任何人（包括匿名访问）都可以对该 Bucket 中的 Object 进行读/写/删除操作；所有这些操作产生的费用由该 Bucket 的 Owner 承担，请慎用该权限。
public-read	公共读，私有写	只有该 Bucket 的 Owner 或者授权对象可以对存放在其中的 Object 进行写/删除操作；任何人（包括匿名访问）可以对 Object 进行读操作。
private	私有读写	只有该 Bucket 的 Owner 或者授权对象可以对存放在其中的 Object 进行读/写/删除操作；其他人在未经授权的情况下无法访问该 Bucket 内的 Object。

- 操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

Object ACL

- Object ACL 介绍

  Object ACL是Object 级别的权限访问控制。目前有四种访问权限：private, public-read, public-read-write, default。PutObjectACL 操作通过 Put 请求中的`x-oss-object-acl`头来设置，这个操作只有 Bucket Owner 有权限执行。

  Object ACL 的四种访问权限含义如下：

权限值	中文名称	权限对访问者的限制
public-read-write	公共读写	该 ACL 表明某个 Object 是公共读写资源，即所有用户拥有对该 Object 的读写权限。
public-read	公共读，私有写	该 ACL 表明某个 Object 是公共读资源，即非 Object Owner 只有该 Object 的读权限，而 Object Owner 拥有该 Object 的读写权限。
private	私有读写	该 ACL 表明某个 Object 是私有资源，即只有该 Object 的 Owner 拥有该 Object 的读写权限，其他的用户没有权限操作该 Object。
default	默认权限	该 ACL 表明某个 Object 是遵循 Bucket 读写权限的资源，即 Bucket 是什么权限，Object 就是什么权限。

  说明:

  - 如果没有设置 Object 的权限，即 Object 的 ACL 为 default，Object 的权限和 Bucket 权限一致。
  - 如果设置了 Object 的权限，Object 的权限大于 Bucket 权限。例如，设置了 Object 的权限是 public-read，则无论 Bucket 是什么权限，该 Object 都可以被身份验证访问和匿名访问。

- 操作方式

操作方式	说明
控制台	Web应用程序，直观易用
图形化工具ossbrowser	图形化工具，易操作
命令行工具ossutil	命令行工具，性能好
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	

操作方式	说明
Go SDK	
C SDK	
.NET SDK	

更多参考

如果您希望仅允许指定用户访问您的文件，可参考：

- #unique_84
- 跨账号授权OSS资源

14.3 基于RAM Policy的权限控制

14.3.1 教程示例：使用RAM Policy控制存储空间和文件夹的访问权限

本教程示例详细演示了如何使用 RAM Policy 控制用户对 OSS 存储空间和文件夹的访问。

在示例中，我们首先创建一个存储空间和文件夹，然后使用阿里云主账号创建访问管理（RAM）用户，并通过构建不同的 RAM Policy 为这些用户授予对所创建 OSS 存储空间及文件夹的增量权限。

存储空间和文件夹的基本概念

阿里云 OSS 的数据模型为扁平型结构，所有文件都直接隶属于其对应的存储空间。因此，OSS 缺少文件系统中类似于目录与子文件夹的层次结构。但是，您可以在 OSS 控制台上模拟文件夹层次结构。在该控制台中，您可以按文件夹对相关文件进行分组、分类和管理，如下图所示。

OSS 提供使用键值（key）对格式的分布式对象存储服务。用户根据其唯一的key（对象名）检索对象的内容。例如，名为 example-company 的存储空间有三个文件夹：*Development#* *Marketing* 和 *Private*，以及一个对象 *oss-dg.pdf*。

- 在创建 *Development* 文件夹时，控制台会创建一个 key 为`Development/`的对象。注意，文件夹的 key 包括分隔符 `/`。

- 当您将名为 *ProjectA.docx* 的对象上传到 *Development* 文件夹中时，控制台会上传该对象并将其key设置为`Development/ProjectA.docx`。

  在该key中，`Development`为前缀，而`/`为分隔符。您可以从存储空间中获取具有特定前缀和分隔符的所有对象的列表。在控制台中，单击 *Development* 文件夹时，控制台会列出文件夹中的对象，如下图所示。

  说明:
  当控制台列出 example-company 存储空间中的 *Development* 文件夹时，它会向 OSS 发送一个用于指定前缀 `Development` 和分隔符`/`的请求。控制台的响应与文件系统类似，会显示文件夹列表。上例说明，存储空间 example-company 有三个对象，其 key 分别为 `Development/Alibaba Cloud.pdf`、`Development/ProjectA.docx` 及 `Development/ProjectB.docx`。

控制台通过对象的 key 推断逻辑层次结构。当您创建对象的逻辑层次结构时，您可以管理对个别文件夹的访问，如本教程后面描述的那样。

在本教程开始之前，您还需要知道“根级”存储空间内容的概念。假设 example-company 存储空间包含以下对象：

- Development/Alibaba Cloud.pdf
- Development/ProjectA.docx

- Development/ProjectB.docx
- Marketing/data2016.xlsx
- Marketing/data2016.xlsx
- Private/2017/images.zip
- Private/2017/promote.pptx
- oss-dg.pdf

这些对象的key构建了一个以 *Development#Marketing* 和 *Private* 作为根级文件夹并以 oss-dg.pdf 作为根级对象的逻辑层次结构。当您单击 OSS 控制台中的存储空间名时，控制台会将一级前缀和一个分隔符（*Development/#Marketing/* 和 *Private/*）显示为根级文件夹。对象 *oss-dg.pdf* 没有前缀，因此显示为根级别项。

OSS 的请求和响应逻辑

在授予权限之前，我们需要清楚，当用户单击某个存储空间的名字时控制台向 OSS 发送的是什么请求、OSS 返回的是什么响应，以及控制台如何解析该响应。

当用户单击某个存储空间名时，控制台会将 GetBucket 请求发送至 OSS。此请求包括以下参数：

- `prefix`，其值为空字符串。
- `delimiter`，其值为`/`。

请求示例如下所示：

```
GET /?prefix=&delimiter=/ HTTP/1.1
Host: example-company.oss-cn-hangzhou.aliyuncs.com
Date: Fri, 24 Feb 2012 08:43:27 GMT
Authorization: OSS qn6qrrqxo2oawuk53otfjbyc:DNrnx7xHk3sgysx7I8U9
I9IY1vY=
```

OSS 返回的响应包括`ListBucketResult`**元素：**

```
HTTP/1.1 200 OK
x-oss-request-id: 534B371674E88A4D8906008B
Date: Fri, 24 Feb 2012 08:43:27 GMT
Content-Type: application/xml
Content-Length: 712
Connection: keep-alive
Server: AliyunOSS
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns=¡±http://doc.oss-cn-hangzhou.aliyuncs.com¡±>
<Name>example-company</Name>
<Prefix></Prefix>
<Marker></Marker>
<MaxKeys>100</MaxKeys>
<Delimiter>/</Delimiter>
    <IsTruncated>false</IsTruncated>
    <Contents>
        <Key>oss-dg.pdf</Key>
        ...
    </Contents>
   <CommonPrefixes>
        <Prefix>Development</Prefix>
   </CommonPrefixes>
      <CommonPrefixes>
        <Prefix>Marketing</Prefix>
   </CommonPrefixes>
      <CommonPrefixes>
        <Prefix>Private</Prefix>
   </CommonPrefixes>
</ListBucketResult>
```

由于 *oss-dg.pdf* **不包含**`/`**分隔符，因此 OSS 在**`<Contents/>`**元素中返回该 key。存储空间 example-company 中的所有其他 key 都包含**`/`**分隔符，因此 OSS 会将这些 key 分组，并为每个前缀值** *Development/#Marketing/* **和** *Private/* **返回一个** `<CommonPrefixes/>` **元素。该元素是一个字符串，包含从这些 key 的第一个字符开始到第一次出现指定的**`/`**分隔符之间的字符。**

控制台会解析此结果并显示如下的根级别项：

现在，如果用户单击 *Development* 文件夹，控制台会将 GetBucket 请求发送至 OSS。此请求包括以下参数：

- `prefix`，其值为 `Development/`。
- `delimiter`，其值为`/`。

请求示例如下所示：

```
GET /?prefix=Development/&delimiter=/ HTTP/1.1
Host: oss-example.oss-cn-hangzhou.aliyuncs.com
Date: Fri, 24 Feb 2012 08:43:27 GMT
Authorization: OSS qn6qrrqxo2oawuk53otfjbyc:DNrnx7xHk3sgysx7I8U9
I9IY1vY=
```

作为响应，OSS 返回以指定前缀开头的 key：

```
HTTP/1.1 200 OK
x-oss-request-id: 534B371674E88A4D8906008B
Date: Fri, 24 Feb 2012 08:43:27 GMT
Content-Type: application/xml
Content-Length: 712
Connection: keep-alive
Server: AliyunOSS
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns=¡±http://doc.oss-cn-hangzhou.aliyuncs.com¡±>
<Name>example-company</Name>
<Prefix>Development/</Prefix>
<Marker></Marker>
<MaxKeys>100</MaxKeys>
<Delimiter>/</Delimiter>
```

```
    <IsTruncated>false</IsTruncated>
    <Contents>
        <Key>ProjectA.docx</Key>
        ...
    </Contents>
    <Contents>
        <Key>ProjectB.docx</Key>
        ...
    </Contents>
</ListBucketResult>
```

控制台会解析此结果并显示如下的 key：

教程示例

本教程示例如下所示：

- 创建一个存储空间 example-company，然后向其中添加三个文件夹（*Development*、*Marketing* 和 *Private*）。
- 您有 Anne 和 Leo 两个用户。您希望 Anne 只能访问 *Development* 文件夹而 Leo 则只能访问 *Marketing* 文件夹，并且希望将 *Private* 文件夹保持私有。在教程示例中，通过创建访问控制（RAM）用户（Anne 和 Leo）来管理访问权限，并授予他们必要的权限。
- RAM 还支持创建用户组并授予适用于组中所有用户的组级别权限。这有助于更好地管理权限。在本示例中，Anne 和 Leo 都需要一些公共权限。因此，您还要创建一个名为 Staff 的组，然后将 Anne 和 Leo 添加到该组中。首先，您需要给该组分配策略授予权限。然后，将策略分配给特定用户，添加特定用户的权限。

> **说明:**
> 本教程示例使用 example-company 作为存储空间名、使用 Anne 和 Leo 作为 RAM 用户名并使用 Staff 作为组名。由于阿里云 OSS 要求存储空间名全局唯一，所以您需要用自己的存储空间名称替换本教程中的存储空间名。

示例准备

本示例使用阿里云主账号创建 RAM 用户。最初，这些用户没有任何权限。您将逐步授予这些用户执行特定 OSS 操作的权限。为了测试这些权限，您需要使用每个用户的 RAM 账号登录到控制台。当您作为主账号所有者逐步授予权限并作为 RAM 用户测试权限时，您需要每次使用不同账号进行登录和注销。您可以使用一个浏览器来执行此测试。如果您可以使用两个不同的浏览器，则该测试过程用时将会缩短：一个浏览器用于使用主账号连接到阿里云控制台，另一个浏览器用于使用 RAM 账号进行连接。

要使用您的主账号登录到阿里云控制台。 RAM 用户不能使用相同的链接登录。他们必须使用 RAM 用户登录链接。作为主账号所有者，您可以向 RAM 用户提供此链接。

> **说明:**
> 有关 RAM 的详细信息，请参见使用 RAM 用户账号登录。

为 RAM 用户提供登录链接：

1. 使用主账号登录 RAM 控制台。
2. 在左侧导航栏中，单击概览。
3. 在账号管理区域，将 RAM 用户登录地址的 URL 提供给 RAM 用户，以便其使用 RAM 用户名和密码登录控制台。

步骤 1：创建存储空间

在此步骤中，您可以使用主账号登录到 OSS 控制台、创建存储空间、将文件夹（`Development#Marketing#Private`）添加到存储空间中，并在每个文件夹中上传一个或两个示例文档。

1. 使用主账号登录 OSS 控制台。
2. 创建名为 example-company 的存储空间。

   有关详细过程，请参见 OSS 控制台用户指南中的创建存储空间 。
3. 将一个文件上传到存储空间中。

   本示例假设您将文件 `oss-dg.pdf` 上传到存储空间的根级别。您可以用不同的文件名上传自己的文件。

   有关详细过程，请参见 OSS 控制台用户指南中的上传文件。

4. 添加名为 *Development#Marketing* 和 *Private* 的三个文件夹。

   有关详细过程，请参见 OSS 控制台用户指南 中的创建文件夹。

5. 将一个或两个文件上传到每个文件夹中。

   本例假设您将具有以下对象键的对象上传到存储空间中：

   - Development/Alibaba Cloud.pdf
   - Development/ProjectA.docx
   - Development/ProjectB.docx
   - Marketing/data2016.xlsx
   - Marketing/data2016.xlsx
   - Private/2017/images.zip
   - Private/2017/promote.pptx
   - oss-dg.pdf

步骤 2：创建 RAM 用户和组

在此步骤中，您使用 RAM 控制台将两个 RAM 用户 Anne 和 Leo 添加到主账号中。您还将创建一个名为 Staff 的组，然后将这两个用户添加到该组中。

说明：

在此步骤中，不要分配任何授予这些用户权限的策略。在以下步骤中，您将逐步为其授予权限。

有关创建 RAM 用户的详细过程，请参见 RAM 快速入门中的创建 RAM 用户。请为每个 RAM 用户创建登录密码。

有关创建组的详细过程，请参见 RAM 用户指南中的创建组。

步骤 3：确认 RAM 用户没有任何权限

如果您使用两个浏览器，现在可以在另一个浏览器中使用其中一个 RAM 用户账号登录到控制台。

1. 打开 RAM 用户登录链接，并用 Anne 或 Leo 的账号登录到 RAM 控制台。
2. 打开 OSS 控制台。

   您发现控制台中没有任何存储空间，这意味着 Anne 不具有对存储空间 example-company 的任何权限。

步骤 4：授予组级别权限

我们希望 Anne 和 Leo 都能执行以下操作：

- 列出主账号所拥有的所有存储空间。

  为此，Anne 和 Leo 必须具有执行 `oss:ListBuckets` 操作的权限。

- 列出 example-company 存储空间中的根级别项、文件夹和对象。

  为此，Anne 和 Leo 必须具有对 example-company 存储空间执行 `oss:ListObjects` 操作的权限。

步骤 4.1.授予列出所有存储空间的权限

在此步骤中，创建一个授予用户最低权限的策略。凭借最低权限，用户可列出主账号所拥有的所有存储空间。您还将此策略分配给 Staff 组，以便授予获得主账号拥有的存储空间列表的组权限。

1. 使用主账号登录 RAM 控制台。
2. 创建策略 `AllowGroupToSeeBucketListInConsole`。

   a. 在左侧导航窗格中，单击权限管理 > 权限策略管理 > 新建权限策略。

   b. 在策略名称字段中，输入 `AllowGroupToSeeBucketListInConsole`。

   c. 选择配置模式为脚本配置。

   d. 在策略内容字段中，复制并粘贴以下策略。

```
{
"Version": "1",
"Statement": [
 {
   "Effect": "Allow",
       "Action": [
         "oss:ListBuckets",
         "oss:GetBucketStat",
          "oss:GetBucketInfo",
         "oss:GetBucketAcl"
       ],
   "Resource": [
     "acs:oss:*:*:*"
   ]
 }
]
}
```

说明:

- RAM 策略为 JSON 格式。各字段定义如下：
  - Statement：对象数组，每个对象使用名/值对的集合来描述权限。
  - Effect：决定是允许还是拒绝特定的权限。
  - Action：指定访问权限的类型。在本策略中，`oss:ListBuckets` 是预定义的 OSS 操作，可返回经过身份验证的发送者所拥有的所有储存空间的列表。

- 推荐您使用 RAM 策略编辑器快速生成 RAM 策略。更详细的 RAM 策略介绍请参见如何构建 RAM Policy。

3. 将 `AllowGroupToSeeBucketListInConsole` 策略分配给 Staff 组。

   有关分配策略的详细过程，请参见 RAM 快速入门为 RAM 用户授权中的“为用户所属的用户组授权”。

   可以将策略分配给 RAM 控制台中的 RAM 用户和组。在本例中，我们将策略分配给组，因为我们希望 Anne 和 Leo 都能够列出这些存储空间。

4. 测试权限。

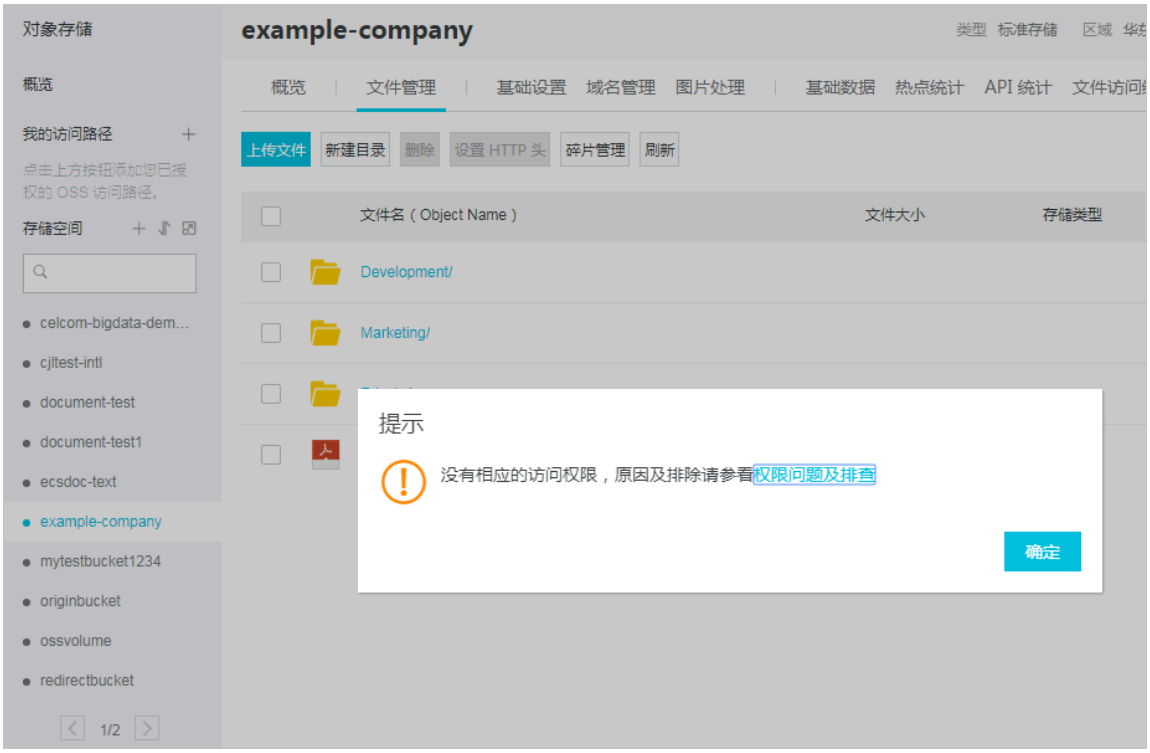
   a. 打开 RAM 用户登录链接，并用 Anne 或 Leo 的账号登录到 RAM 控制台。

   b. 打开 OSS 控制台。

      控制台列出所有存储空间。

   c. 单击 example-company 存储空间，然后单击文件选项卡。

      此时将显示一个消息框，表明您没有相应的访问权限。



步骤 4.2.授予列出存储空间根级内容的权限

在此步骤中，您授予权限，允许所有用户列出存储空间 example-company 中的所有项目。当用户在 OSS 控制台中单击 example-company 时，能够看到存储空间中的根级别项。

1. 使用主账号登录RAM 控制台。
2. 用以下策略取代分配给 Staff 组的现有策略 `AllowGroupToSeeBucketListInConsole`，该策略还允许 `oss:ListObjects` 操作。请用您的存储空间名替换策略资源中的 example-company。

   有关详细过程，请参见 RAM 用户指南中授权策略的“修改自定义授权策略”部分。注意，您最多可对 RAM 策略进行五次修改。如果超过了五次，则需要删除该策略并创建一个新的策略，然后再次将新策略分配给 Staff 组。

```
{
   "Version": "1",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "oss:ListBuckets",
         "oss:GetBucketStat",
         "oss:GetBucketInfo",
         "oss:GetBucketAcl"
       ],
       "Resource": [
         "acs:oss:*:*:*"
       ],
       "Condition": {}
     },
     {
       "Effect": "Allow",
       "Action": [
         "oss:ListObjects"
```

```
        ],
        "Resource": [
          "acs:oss:*:*:example-company"
        ],
        "Condition": {
          "StringLike": {
            "oss:Prefix": [
              ""
            ],
            "oss:Delimiter": [
              "/"
            ]
          }
        }
      }
    ]
}
```

说明:

- **要列出存储空间内容，用户需要调用 `oss:ListObjects` 操作的权限。为了确保用户仅看到根级内容，我们添加了一个条件：用户必须在请求中指定一个空前缀，也就是说，他们不能单击任何根级文件夹。我们还通过要求用户请求包含分隔符参数和值 `/` 来添加需要文件夹样式访问的条件。**
- **当用户登录到 OSS 控制台时，控制台检查用户的身份是否有访问 OSS 服务的权限。要在控制台中支持存储空间操作，我们还需要添加 `oss:GetBucketAcl` 操作。**

3. 测试更新的权限。

   a. 打开 RAM 用户登录链接，并用 Anne 或 Leo 的账号登录到 RAM 控制台。

   b. 打开 OSS 控制台。

      控制台列出所有存储空间。

   c. 单击 example-company 存储空间，然后单击文件选项卡。

      控制台列出所有根级别项。

   d. 单击任何文件夹或对象 `oss-dg.pdf`。

      此时将显示一个消息框，表明您没有相应的访问权限。

组策略摘要

添加组策略的最终结果是授予 RAM 用户 Anne 和 Leo 以下最低权限：

- 列出主账号所拥有的所有存储空间。
- 查看 example-company 存储空间中的根级别项。

然而，他们可以进行的操作仍然有限。在以下部分中，我们将授予用户以下特定权限：

- 允许 Anne 在 `Development` 文件夹中获取和放入对象。
- 允许 Leo 在 `Finance` 文件夹中获取和放入对象。

对于用户特定的权限，您需要将策略分配给特定用户，而非分配给组。以下部分授予 Anne 在 *Development* 文件夹中操作的权限。您可以重复这些步骤，授予 Leo 在 *Finance* 文件夹中进行类似操作的权限。

步骤 5：授予 RAM 用户 Anne 特定权限

在此步骤中，我们向 Anne 授予额外的权限，使她可以看到 *Development* 文件夹的内容，并将对象放入文件夹中。

步骤 5.1.授予 RAM 用户 Anne 权限以列出 *Development* 文件夹内容

若要 Anne 能够列出 *Development* 文件夹内容，您必须为其分配策略。该策略必须能够授予其对 example-company 存储空间执行 `oss:ListObjects` 操作的权限，还必须包括要求用户在请求中指定前缀 *Development/* 的条件。

1. 使用主账号登录RAM 控制台。
2. 创建策略 `AllowListBucketsIfSpecificPrefixIsIncluded`，授予 RAM 用户 Anne 权限以列出 *Development* 文件夹内容。
   a. 在左侧导航窗格中，单击权限管理 > 权限策略管理 > 新建权限策略。
   b. 在策略名称字段中，输入 `AllowGroupToSeeBucketListInConsole`。
   c. 选择配置模式为脚本配置。
   d. 在策略内容字段中，复制并粘贴以下策略。

```
{
"Version": "1",
"Statement": [
 {
   "Effect": "Allow",
   "Action": [
     "oss:ListObjects"
   ],
   "Resource": [
     "acs:oss:*:*:example-company"
   ],
   "Condition": {
     "StringLike": {
       "oss:Prefix": [
         "Development/*"
       ]
     }
   }
 }
]
}
```

3. 将策略分配给 RAM 用户 Anne。

   有关分配策略的详细过程，请参见 RAM 快速入门中的为 RAM 用户授权。

4. 测试 Anne 的权限。

   a. 打开 RAM 用户登录链接，并用 Anne 的账号登录到 RAM 控制台。

   b. 打开 OSS 控制台，控制台列出所有存储空间。

   c. 单击 example-company 存储空间，然后单击文件选项卡，控制台列出所有根级别项。

   d. 单击 *Development/* 文件夹。控制台列出文件夹中的对象。

步骤 5.2 授予 RAM 用户 Anne 在 *Development* 文件夹中获取和放入对象的权限。

若要 Anne 能够在 *Development* 文件夹中获取和放入对象，您必须授予她调用 `oss:GetObject` 和 `oss:PutObject` 操作的权限，包括用户必须在请求中指定前缀 *Development/* 的条件。

1. 使用主账号登录 RAM 控制台。
2. 用以下策略取代您在之前步骤中创建的策略 `AllowListBucketsIfSpecificPrefixIsIncluded`。

   有关详细过程，请参见 RAM 用户指南中授权策略的“修改自定义授权策略”部分。注意，您最多可对 RAM 策略进行五次修改。如果超过了五次，则需要删除该策略并创建一个新的策略，然后再次将新策略分配给 Staff 组。

```
{
   "Version": "1",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "oss:ListObjects"
       ],
       "Resource": [
         "acs:oss:*:*:example-company"
       ],
       "Condition": {
         "StringLike": {
           "oss:Prefix": [
             "Development/*"
           ]
         }
       }
     },
     {
       "Effect": "Allow",
       "Action": [
         "oss:GetObject",
         "oss:PutObject",
         "oss:GetObjectAcl"
       ],
       "Resource": [
         "acs:oss:*:*:example-company/Development/*"
       ],
       "Condition": {}
     }
   ]
```

```
  }
```

说明:

当用户登录到 OSS 控制台时，控制台检查用户的身份是否有访问 OSS 服务的权限。要在控制台中支持存储空间操作，我们还需要添加 `oss:GetObjectAcl` 操作。

3. **测试更新的策略。**

   a. **打开 RAM 用户登录链接，并用 Anne 的账号登录到 RAM 控制台。**

   b. **打开 OSS 控制台。**

      控制台列出所有存储空间。

   c. **在 OSS 控制台中，确认 Anne 现在可以在 *Development* 文件夹中添加对象并下载对象。**

步骤 5.3 显式拒绝 RAM 用户 Anne 访问存储空间中任何其他文件夹的权限

RAM 用户 Anne 现在可以在 example-company 存储空间中列出根级内容，并将对象放入 *Development* 文件夹中。如果要严格限制访问权限，您可以显式拒绝 Anne 对存储空间中任何其他文件夹的访问。如果有授予 Anne 访问存储空间中任何其他文件夹的其他策略，则此显式策略将替代这些权限。

您可以将以下语句添加到 RAM 用户 Anne 的策略 `AllowListBucketsIfSpecificPrefixIsIncluded`，以要求 Anne 发送到 OSS 的所有请求包含前缀参数，该参数的值可以是 *Development/ 或空字符串。**

```
{
      "Effect": "Deny",
      "Action": [
        "oss:ListObjects"
      ],
      "Resource": [
        "acs:oss:*:*:example-company"
      ],
      "Condition": {
        "StringNotLike": {
          "oss:Prefix": [
            "Development/*",
            ""
          ]
        }
      }
    }
```

按照之前的步骤更新您为 RAM 用户 Anne 创建的策略 `AllowListBucketsIfSpecificPrefixIsIncluded`。复制并粘贴以下策略以替换现有策略。

```
{
  "Version": "1",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "oss:ListObjects"
      ],
      "Resource": [
        "acs:oss:*:*:example-company"
      ],
      "Condition": {
        "StringLike": {
          "oss:Prefix": [
            "Development/*"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "oss:GetObject",
        "oss:PutObject",
        "oss:GetObjectAcl"
      ],
      "Resource": [
        "acs:oss:*:*:example-company/Development/*"
      ],
      "Condition": {}
    },
    {
      "Effect": "Deny",
      "Action": [
        "oss:ListObjects"
      ],
      "Resource": [
        "acs:oss:*:*:example-company"
      ],
      "Condition": {
        "StringNotLike": {
          "oss:Prefix": [
            "Development/*",
            ""
          ]
        }
      }
    }
  ]
}
```

步骤 6：授予 RAM 用户 Leo 特定权限

现在，您希望授予 Leo 访问 *Marketing* 文件夹的权限。请遵循之前用于向 Anne 授予权限的步骤，但应将 *Development* 文件夹替换为 *Marketing* 文件夹。有关详细过程，请参见步骤 5：授予 RAM 用户 Anne 特定权限。

步骤 7：确保 Private 文件夹安全

在本例中，您仅拥有两个用户。您在组级别授予两个用户所有所需的最小权限，只有当您真正需要单个用户级别上的权限时，才授予用户级别权限。此方法有助于最大限度地减少管理权限的工作量。随着用户数量的增加，我们希望确保不意外地授予用户对 *Private* 文件夹的权限。因此，我

们需要添加一个显式拒绝访问 *Private* 文件夹的策略。显式拒绝策略会取代任何其他权限。若要确保 *Private* 文件夹保持私有，可以向组策略添加以下两个拒绝语句：

- 添加以下语句以显式拒绝对 *Private* 文件夹 (*example-company/Private/**) 中的资源执行任何操作。

```
{
    "Effect": "Deny",
    "Action": [
      "oss:*"
    ],
    "Resource": [
      "acs:oss:*:*:example-company/Private/*"
    ],
    "Condition": {}
  }
```

- 您还要在请求指定了 *Private/prefix* 时拒绝执行 `ListObjects` 操作的权限。在控制台中，如果 Anne 或 Leo 单击 *Private* 文件夹，则此策略将导致 OSS 返回错误响应。

```
{
    "Effect": "Deny",
    "Action": [
      "oss:ListObjects"
    ],
    "Resource": [
      "acs:oss:*:*:*"
    ],
    "Condition": {
      "StringLike": {
        "oss:Prefix": [
          "Private/"
        ]
      }
    }
  }
```

- 用包含前述拒绝语句的更新策略取代 Staff 组策略 `AllowGroupToSeeBucketListInConsole`。在应用更新策略后，组中的任何用户都不能访问您的存储空间中的 Private 文件夹。

  1. 使用主账号登录RAM 控制台。
  2. 用以下策略取代分配给 Staff 组的现有策略 `AllowGroupToSeeBucketListInConsole`。请用您的存储空间名替换策略资源中的 example-company。

```
{
"Version": "1",
"Statement": [
 {
   "Effect": "Allow",
       "Action": [
         "oss:ListBuckets",
         "oss:GetBucketStat",
         "oss:GetBucketInfo",
         "oss:GetBucketAcl"
       ],
```

```
    "Resource": [
      "acs:oss:*:*:*"
    ],
    "Condition": {}
  },
  {
    "Effect": "Allow",
    "Action": [
      "oss:ListObjects"
    ],
    "Resource": [
      "acs:oss:*:*:example-company"
    ],
    "Condition": {
      "StringLike": {
        "oss:Prefix": [
           ""
         ],
        "oss:Delimiter": [
          "/"
        ]
      }
    }
  },
  {
    "Effect": "Deny",
    "Action": [
      "oss:*"
    ],
    "Resource": [
      "acs:oss:*:*:example-company/Private/*"
    ],
    "Condition": {}
  },
  {
    "Effect": "Deny",
    "Action": [
      "oss:ListObjects"
    ],
    "Resource": [
      "acs:oss:*:*:*"
    ],
    "Condition": {
      "StringLike": {
        "oss:Prefix": [
          "Private/"
        ]
      }
    }
  }
]
}
```

清理

要进行清理，您需要在 RAM 控制台中删除用户 Anne 和 Leo。

有关详细过程，请参见 RAM 用户指南中用户的“删除 RAM 用户”部分。

为了确保您不再因存储而继续被收取费用，您还需要删除为本示例创建的对象和存储空间。

14.3.2 如何构建RAM Policy

RAM （Resource Access Management）是阿里云提供的资源访问控制服务。RAM Policy是基于用户的授权策略。通过设置RAM Policy，您可以集中管理您的用户（比如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。比如能够限制您的用户只拥有对某一个Bucket 的读权限。子账号是从属于主账号的，并且这些账号下不能拥有实际的任何资源，所有资源都属于主账号。

说明:

您可以使用RAM策略编辑器快速生成RAM Policy。

常见Policy示例

- 完全授权的Policy

  完全授权的Policy表示允许应用可以对OSS进行任何操作。

  警告:

  完全授权的Policy对移动应用来说是不安全的授权，不推荐使用。

```
{
  "Statement": [
    {
      "Action": [
        "oss:*"
      ],
      "Effect": "Allow",
      "Resource": ["acs:oss:*:*:*"]
    }
  ],
  "Version": "1"
}
```

对OSS的操作	结果
列举所有创建的Bucket	成功
上传不带前缀的Object，test.txt	成功
下载不带前缀的Object，test.txt	成功
上传带前缀的Object，user1/test.txt	成功
下载带前缀的Object，user1/test.txt	成功
列举Object，test.txt	成功
带前缀的Object，user1/test.txt	成功

- 不限制前缀的只读不写Policy

  此Policy表示应用对Bucket`app-base-oss`下所有的Object可列举，可下载。

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object，test.txt	成功
上传带前缀的Object，user1/test.txt	失败
下载带前缀的Object，user1/test.txt	成功
列举不带前缀的Object，test.txt	成功
列举带前缀的Object，user1/test.txt	成功

- 限制前缀的只读不写Policy

  此Policy表示应用对Bucket`app-base-oss`下带有前缀`user1/`的Object可列举、可下载，但无法下载其他前缀的Object。采用此种Policy，如果不同的应用对应不同的前缀，就可以达到在同一个Bucket中空间隔离的效果。

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
```

```
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object，test.txt	失败
上传带前缀的Object，user1/test.txt	失败
下载带前缀的Object，user1/test.txt	成功
列举Object，test.txt	成功
带前缀的Object，user1/test.txt	成功

- 不限制前缀的只写不读Policy

  此Policy表示应用可以对Bucket`app-base-oss`下所有的Object进行上传。

```
{
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	成功
下载不带前缀的Object，test.txt	失败
上传带前缀的Object，user1/test.txt	成功
下载带前缀的Object，user1/test.txt	成功
列举Object，test.txt	成功
带前缀的Object，user1/test.txt	成功

- 限制前缀的只写不读Policy

  此Policy表示应用可以对Bucket`app-base-oss`下带有前缀`user1/`的Object进行上传。但无法上传其他前缀的Object。采用此种Policy，如果不同的应用对应不同的前缀，就可以达到在同一个Bucket中空间隔离的效果。

```
{
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object，test.txt	失败
上传带前缀的Object，user1/test.txt	成功
下载带前缀的Object，user1/test.txt	失败
列举Object，test.txt	失败
带前缀的Object，user1/test.txt	失败

- 不限制前缀的读写Policy

  此Policy表示应用可以对Bucket`app-base-oss`下所有的Object进行列举、下载、上传和删除。

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
```

```
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	成功
下载不带前缀的Object，test.txt	成功
上传带前缀的Object，user1/test.txt	成功
下载带前缀的Object，user1/test.txt	成功
列举Object，test.txt	成功
带前缀的Object，user1/test.txt	成功

- 限制前缀的读写Policy

  此Policy表示应用可以对Bucket`app-base-oss`下带有前缀`user1/`的Object进行列举、下载、上传和删除，但无法对其他前缀的Object进行读写。采用此种Policy，如果不同的应用对应不同的前缀，就可以达到在同一个Bucket中空间隔离的效果。

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
```

对OSS的操作	结果
列举所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object，test.txt	失败
上传带前缀的Object，user1/test.txt	成功
下载带前缀的Object，user1/test.txt	成功
列举Object，test.txt	成功
带前缀的Object，user1/test.txt	成功

复杂Policy示例

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "oss:GetBucketAcl",
                "oss:ListObjects"
            ],
            "Resource": [
                "acs:oss:*:1775305056529849:mybucket"
            ],
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "acs:UserAgent": "java-sdk",
                    "oss:Prefix": "foo"
                },
                "IpAddress": {
                    "acs:SourceIp": "192.168.0.1"
                }
            }
        },
        {
            "Action": [
                "oss:PutObject",
                "oss:GetObject",
                "oss:DeleteObject"
            ],
            "Resource": [
                "acs:oss:*:1775305056529849:mybucket/file*"
            ],
            "Effect": "Allow",
            "Condition": {
                "IpAddress": {
                    "acs:SourceIp": "192.168.0.1"
                }
            }
        }
    ]
}
```

这是一个较为复杂的授权 Policy，用户用这样的一个 Policy 通过 RAM 或 STS 服务向其他用户授权。Policy 当中有一个 Statement（一条 Policy 当中可以有多条 Statement）。Statement 里面规定了相应的 Action、Resource、Effect 和 Condition。

这条 Policy 把用户自己名下的`mybucket`和`mybucket/file*`这些资源授权给相应的用户，并且支持 GetBucketAcl、GetBucket、PutObject、GetObject 和 DeleteObject 这几种操作。Condition 中的条件表示 UserAgent为 java-sdk，源IP为192.168.0.1的时候鉴权才能通过，被授权的用户才能访问相关的资源。Prefix 这个 Condition 是在 GetBucket（ListObjects）的时候起作用的，关于这个字段的解释详见 OSS的 API 文档。

Version

Version定义了Policy的版本，本文档中 Version 设置为`1`。

Statement

通过Statement描述授权语义，其中可以根据业务场景包含多条语义，每条包含对 Action、Effect、Resource 和 Condition 的描述。每次请求系统会逐条依次匹配检查，所有匹配成功的 Statement 会根据 Effect 的设置不同分为通过（Allow）、禁止（Deny），其中禁止（Deny）的优先。如果匹配成功的都为通过，该条请求即鉴权通过。如果匹配成功有一条禁止，或者没有任何条目匹配成功，该条请求被禁止访问。

Action

Action 分为三大类：

- Service 级别操作，对应的是 GetService 操作，用来列出所有属于该用户的 Bucket 列表。
- Bucket 级别操作，对应类似于 oss:PutBucketAcl、oss:GetBucketLocation之类的操作，操作的对象是 Bucket，它们的名称和相应的接口名称一一对应。
- Object 级别操作，分为 oss:GetObject、oss:PutObject、oss:DeleteObject和oss:AbortMultipartUpload，操作对象是 Object。

如想授权某一类的 Object 的操作，可以选择这几种的一种或几种。另外，所有的 Action 前面都必须加上`oss:`，如上面例子所示。Action 是一个列表，可以有多个 Action。具体的 Action 和 API 接口的对应关系如下：

- Service级别

API	Action
GetService（ListBuckets）	oss:ListBuckets

- Bucket 级别

API	Action
PutBucket	oss:PutBucket
GetBucket（ListObjects）	oss:ListObjects
PutBucketAcl	oss:PutBucketAcl
DeleteBucket	oss:DeleteBucket
GetBucketLocation	oss:GetBucketLocation
GetBucketAcl	oss:GetBucketAcl
GetBucketLogging	oss:GetBucketLogging
PutBucketLogging	oss:PutBucketLogging
DeleteBucketLogging	oss:DeleteBucketLogging

API	Action
GetBucketWebsite	oss:GetBucketWebsite
PutBucketWebsite	oss:PutBucketWebsite
DeleteBucketWebsite	oss:DeleteBucketWebsite
GetBucketReferer	oss:GetBucketReferer
PutBucketReferer	oss:PutBucketReferer
GetBucketLifecycle	oss:GetBucketLifecycle
PutBucketLifecycle	oss:PutBucketLifecycle
DeleteBucketLifecycle	oss:DeleteBucketLifecycle
ListMultipartUploads	oss:ListMultipartUploads
PutBucketCors	oss:PutBucketCors
GetBucketCors	oss:GetBucketCors
DeleteBucketCors	oss:DeleteBucketCors
PutBucketReplication	oss:PutBucketReplication
GetBucketReplication	oss:GetBucketReplication
DeleteBucketReplication	oss:DeleteBucketReplication
GetBucketReplicationLocation	oss:GetBucketReplicationLocation
GetBucketReplicationProgress	oss:GetBucketReplicationProgress

- Object 级别

API	Action
GetObject	oss:GetObject
HeadObject	oss:GetObject
PutObject	oss:PutObject
PostObject	oss:PutObject
InitiateMultipartUpload	oss:PutObject
UploadPart	oss:PutObject
CompleteMultipart	oss:PutObject
DeleteObject	oss:DeleteObject
DeleteMultipleObjects	oss:DeleteObject
AbortMultipartUpload	oss:AbortMultipartUpload

API	Action
ListParts	oss:ListParts
CopyObject	oss:GetObject,oss:PutObject
UploadPartCopy	oss:GetObject,oss:PutObject
AppendObject	oss:PutObject
GetObjectAcl	oss:GetObjectAcl
PutObjectAcl	oss:PutObjectAcl
RestoreObject	oss:RestoreObject

Resource

Resource 指代的是 OSS 上面的某个具体的资源或者某些资源（支持*通配），resource的规则是`acs:oss:{region}:{bucket_owner}:{bucket_name}/{object_name}`。对于所有 Bucket 级别的操作来说不需要最后的斜杠和{object_name}，即`acs:oss:{region}:{bucket_owner}:{bucket_name}`。Resource 也是一个列表，可以有多个 Resource。其中的 region 字段暂时不做支持，设置为*。

Effect

Effect 代表本条的Statement的授权的结果，分为 Allow 和 Deny，分别指代通过和禁止。多条 Statement 同时匹配成功时，禁止（Deny）的优先级更高。

例如，期望禁止用户对某一目录进行删除，但对于其他文件有全部权限：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "oss:*"
      ],
      "Resource": [
        "acs:oss:*:*:bucketname"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "oss:DeleteObject"
      ],
      "Resource": [
        "acs:oss:*:*:bucketname/index/*",
      ]
    }
  ]
```

```
}
```

Condition

Condition 代表 Policy 授权的一些条件，上面的示例里面可以设置对于 acs:UserAgent 的检查、acs:SourceIp 的检查，还有 oss:Prefix 项用来在 GetBucket 的时候对资源进行限制。

OSS 支持的 Condition 如下：

condition	功能	合法取值
acs:SourceIp	指定 ip 网段	普通的 ip，支持*通配
acs:UserAgent	指定 http useragent 头	字符串
acs:CurrentTime	指定合法的访问时间	ISO8601格式
acs:SecureTransport	是否是 https 协议	“true”或者”false”
oss:Prefix	用作 ListObjects 时的 prefix	合法的object name

最佳实践

OSS提供了RAM策略编辑器帮助您快速生成RAM Policy。您也可以使用图形化管理工具 ossbrowser 的简化 Policy 授权，一键完成对子账号授予特定 Bucket 或特定目录的权限 。

针对具体场景更多的授权策略配置示例，可以参考教程示例：控制存储空间和文件夹的访问权限和 OSS 授权常见问题。

14.4 基于Bucket Policy的权限控制

Bucket Policy是基于资源的授权策略。相比于RAM Policy，Bucket Policy支持在控制台直接进行图形化配置操作，并且Bucket拥有者直接可以进行访问授权。

Bucket Policy常见的应用场景如下：

- 向其他账号的RAM用户授权访问。

  您可以授予其他账号的RAM用户访问您的OSS资源的权限。
- 向匿名用户授予带特定IP条件限制的访问权限。

  某些场景下，您需要向匿名用户授予带IP限制的访问策略。例如，企业内部的机密文档，只允许在企业内部访问，不允许在其他区域访问。由于企业内部人员较多，如果针对每个人配置RAM Policy，工作量非常大。此时，您可以基于Bucket Policy设置带IP限制的访问策略，从而高效方便地进行授权。

Bucket Policy的配置方法和教程视频请参见使用Bucket Policy授权其他用户访问OSS资源。

14.5 跨账号授权

14.5.1 跨账号授权概述

针对跨账号授权访问 OSS 资源的需求，OSS 提供了多种跨账号授权访问的方式。

OSS 资源默认都是私有的，若拥有者希望将资源共享给他人，可以通过跨账号授权的方式，将 OSS 资源授权给他人访问，下面提供几种跨账号授权访问 OSS 资源的方式。

- 基于 Bucket Policy 实现跨账号访问OSS：Bucket Policy 是基于资源的授权策略。相比于 RAM Policy，Bucket Policy 操作简单，支持在控制台直接进行图形化配置，并且 Bucket 拥有者直接可以进行访问授权，无需具备 RAM 操作权限。Bucket Policy 支持向其他账号的 RAM 用户、匿名用户等授予带特定 IP 条件限制的访问权限。
- 跨账号授权OSS资源：RAM 管理员可以配置 RAM 角色，并将合作伙伴的云账号 ID 添加为受信任云账号 ID。通过授予该 RAM 角色配置 OSS 访问权限的方式，可以将 OSS 资源共享给合作伙伴。

14.5.2 教程示例：基于Bucket Policy实现跨账号访问OSS

阿里云 OSS 的资源默认都是私有的，若您希望您的合作伙伴可以访问您的 OSS 资源，可以通过 Bucket Policy 授予合作伙伴访问 Bucket 的权限。

案例：公司 A 希望其合作公司 B 可以访问自己的 OSS 资源，但又不方便开放子账号给 B 公司。此时，A 公司可以通过 Bucket Policy 授予合作伙伴访问 Bucket 的权限。B 公司账号获得授权之后，可以在控制台添加 A 公司 OSS 资源的访问路径进行访问。

添加 Bucket Policy

- B 公司账号：
  1. 登录 RAM 访问控制台，创建 RAM 子账号，详细配置方法请参见创建 RAM 用户。
  2. 在 RAM 访问控制台，单击用户。
  3. 单击刚刚创建的 RAM 用户用户名，查看并记录 RAM 用户的 UID 号。
- A 公司账号
  1. 登录阿里云 OSS 控制台。
  2. 在左侧菜单栏选择您需要授权访问的 Bucket。
  3. 单击文件管理 > 授权 > 新增授权。
  4. 在新增授权的对话框，填写授权策略。其中，授权用户选择其他账号，并填写 B 公司子账号的 UID 号。其他参数请参考 Bucket Policy。
  5. 单击确定。

登录 RAM 子账号并添加访问路径

Bucket Policy 添加完成之后，您还需要登录 B 公司子账号，添加 A 公司的 Bucket 访问路径。配置步骤如下：

1. 通过 RAM 账号登录链接登录 B 公司子账号。
2. 打开 OSS 控制台。
3. 单击左侧菜单栏我的访问路径后的加号（+），添加 A 公司授权访问的 Bucket 路径信息。
   - 区域：下拉选择 A 公司允许访问的 Bucket 所在地域。
   - 访问路径：添加 A 公司允许访问的 Bucket 访问路径，格式为「bucket/object-prefix」。例如，仅允许访问名为 `aliyun` 的 Bucket 下的 `abc` 文件夹，则添加 `aliyun/abc`。

您也可以创建子账号的 AccessKey，并通过 AccessKey 使用 ossutil、ossbrowser 等工具访问被授权的 Bucket。

参考文档

您也可以通过以下方式授权合作伙伴访问您的 OSS 资源：

- 教程示例：基于 RAM 角色实现跨账号访问 OSS

15 日志管理

15.1 访问日志存储

用户在访问 OSS 的过程中，会产生大量的访问日志。日志存储功能，可将 OSS 的访问日志，以小时为单位，按照固定的命名规则，生成一个 Object 写入您指定的 Bucket（目标 Bucket，Target Bucket）。您可以使用阿里云 DataLakeAnalytics 或搭建 Spark 集群等方式对这些日志文件进行分析。同时，您可以配置目标 Bucket 的生命周期管理规则，将这些日志文件转成归档存储，长期归档保存。

说明:

日志存储相关 API 接口请参考：

- 设置日志存储功能：PutBucketLogging
- 删除日志存储配置：DeleteBucketLogging
- 查看日志存储配置：GetBucketLogging

操作方式

操作方式	说明
控制台	Web 应用程序，直观易用
Java SDK	丰富、完整的各类语言 SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

日志存储 Object 命名规则

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

命名规则中，

- `TargetPrefix` 由用户指定，表示存储访问日志记录的 Object 名字前缀，可以为空。

- `YYYY-mm-DD-HH-MM-SS`表示该 Object 被创建时的阿拉伯数字的年、月、日、小时、分钟和秒（注意位数）。
- `UniqueString` 为 OSS 系统生成的字符串（UUID），用于唯一标识该 Log 文件。

存储 OSS 访问日志的 Object 的名称示例如下：

```
MyLog-oss-example2017-09-10-04-00-00-0000
```

上例中，

- `MyLog-` 是用户指定的 Object 前缀。
- `oss-example` 是源 Bucket 的名称。
- `2017-09-10-04-00-00` 是该 Object 的创建时间。
- `0000` 是 OSS 系统生成的字符串。

Log 文件格式

Log 文件的格式组成：以下名称从左至右，以空格分隔。

名 称	例 子	含 义
Remote IP	119.xxx.xx.11	请求发起的 IP 地址（Proxy 代理或用户防火墙可能会屏蔽该字段）
Reserved	-	保留字段
Reserved	-	保留字段
Time	[02/May/2012:00:00:04 +0800]	OSS 收到请求的时间
Request-URI	“GET /aliyun-logo.png HTTP/1.1”	用户请求的 URI（包括 query-string）
HTTP Status	200	OSS 返回的 HTTP 状态码
SentBytes	5576	用户从 OSS 下载的流量
RequestTime (ms)	71	完成本次请求的时间（毫秒）
Referer	`http://www.aliyun.com/product/oss`	请求的 HTTP Referer
User-Agent	curl/7.15.5	HTTP 的 User-Agent 头
HostName	oss-example.oss-cn-hangzhou.aliyuncs.com	请求访问域名
Request ID	505B016950xxxxxx032593A4	用于唯一标识该请求的 UUID

名称	例子	含义
LoggingFlag	true	是否开启了访问日志功能
Requester Aliyun ID	16571xxxxxx83691	RAM User ID；匿名访问为“-”
Operation	GetObject	请求类型
Bucket	oss-example	请求访问的 Bucket 名字
Key	/aliyun-logo.png	用户请求的 Key
ObjectSize	5576	Object 大小
Server Cost Time (ms)	17	OSS 服务器处理本次请求所花的时间（毫秒）
Error Code	NoSuchBucket	OSS 返回的错误码
Request Length	302	用户请求的长度（Byte）
UserID	16571xxxxxx83691	Bucket 拥有者 ID
Delta DataSize	280	Bucket 大小的变化量；若没有变化为-
Sync Request	-	是否是 CDN 回源请求；若不是为-
StorageClass	Standard	显示当前 Object 的存储类型，当前取值为 Standard/IA/Archive/-，-表示操作的是Bucket等无法获取相应存储类型的情况
TargetStorageClass	Standard	显示通过 Lifecycle 或 CopyObject 转换存储类型后的目标 Object 存储类型，当前取值为Standard/IA/Archive/-，-表示该操作无目标 Object 存储类型信息（未通过Lifecycle或者CopyObject转换存储类型）

细节分析

- 源 Bucket 和目标 Bucket 可以是同一个 Bucket，也可以是不同的 Bucket，但必须属于同一个账号下的同一地域内。您也可以将多个源 Bucket 的 Log 都保存在同一个目标 Bucket 内（建议指定不同的 TargetPrefix）。

- OSS 以小时为单位生成 Bucket 访问的 Log 文件，但并不表示这个小时的所有请求都记录在这个小时的 Log 文件内，也有可能出现在上一个或者下一个 Log 文件中。
- OSS 生成一个 Bucket 访问的 Log 文件，算作一次 PUT 操作，并记录其占用的空间，但不会记录产生的流量。Log 生成后，您可以按照普通的 Object 来操作这些 Log 文件。
- OSS 会忽略掉所有以 `x-` 开头的 query-string 参数，但这个 query-string 会被记录在访问 Log 中。如果您想从海量的访问日志中标识一个特殊的请求，可以在 URL 中添加一个 `x-` 开头的 query-string 参数。例如：

  ```
  http://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png
  ```

  ```
  http://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png?x-user=admin
  ```

  OSS 处理上面两个请求，结果是一样的。但是在访问 Log 中，您可以通过搜索 `x-user=admin`，方便地定位出经过标记的这个请求。
- OSS 的 Log 中的任何一个字段，都可能出现`-`，用于表示未知数据或对于当前请求该字段无效。
- 根据需求，OSS 的 Log 格式将来会在尾部添加一些字段，请开发者开发 Log 处理工具时考虑兼容性的问题。

15.2 实时日志查询

用户在访问 OSS 的过程中，会产生大量的访问日志。实时日志查询功能将 OSS 与日志服务（LOG）相结合， 允许您在 OSS 控制台直接查询 OSS 访问日志，帮助您完成 OSS 访问的操作审计、访问统计、异常事件回溯和问题定位等工作，提升您的工作效率并更好地帮助您基于数据进行决策。

实时日志查询与访问日志存储对比

- 实时日志查询：
  - 3分钟内将日志实时推送到日志服务实例中，您可在 OSS 控制台直接查看实时日志。
  - 提供日志分析服务，定制了常用的分析报表，数据查询更方便。
  - 可实时查询和分析原始日志并可按 Bucket、Object 名称、API 操作、时间等条件过滤日志。

- 访问日志存储：
  - 开启日志存储功能后，OSS 自动将访问这个 Bucket 的请求日志以小时为单位，按照固定的命名规则，生成一个 Object 写入您指定的 Bucket（目标Bucket，Target Bucket）。
  - 使用阿里云 DataLakeAnalytics 或搭建 Spark 集群等方式进行分析。
  - 可配置目标 Bucket 的生命周期管理规则，将这些日志文件转成归档存储，长期归档保存。

配置方法

控制台：开启实时日志查询

实时日志查询方式

OSS 实时日志查询功能，提供了以下三种查询方式：

- 原始日志实时查询

  用户对原始日志，指定时间段和查询语句进行实时查询，轻松实现如下功能：

  - 可快速分析某一个字段（如某个 API 操作）在一段时间内的分布情况。
  - 按字段过滤出需要分析的记录：例如，按 Bucket、Object 名称、API 名称等过滤出近一天的文件删除操作，并可查询删除操作的时间和访问IP。
  - 统计 OSS 访问记录：例如，统计一段时间内，某个 Bucket 的 PV、UV 或最高延时等。
- 日志报表查询

  提供了4个开箱即用的报表：

  - 访问中心：展示总体运营状况的信息，包括PV、UV、流量以及外网访问地图分布等。
  - 审计中心：展示文件操作的统计信息，包括文件读、写、删等操作统计。
  - 运维中心：展示针对访问日志的统计信息，包括请求数量、失败操作的分布统计。
  - 性能中心：展示针对性能的统计信息，包括外网下载/上传性能分布、不同网络与文件大小的传输性能、文件下载差异列表等信息。
- 通过日志服务控制台查询

  您可以在日志服务控制台查看 OSS 的访问日志 。

实时日志查询收费详情

OSS 实时日志查询，免费提供最近7天的日志查询。若您设置的日志存储时间大于7天，则超过7天的部分，由日志服务单独收费。当您在外网读写日志服务时也会产生的额外费用。

具体收费标准，请参考日志服务计费方式。

16 数据加密

16.1 服务器端加密

本文介绍如何使用OSS的服务器端加密功能，对持久化在OSS上的数据进行加密保护。

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编码，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。

应用场景

OSS通过服务器端加密机制，提供静态数据保护。适合于对于文件存储有高安全性或者合规性要求的应用场景。例如，深度学习样本文件的存储、在线协作类文档数据的存储。针对不同的应用场景，OSS有如下两种服务器端加密方式：

- 使用KMS托管密钥进行加解密（SSE-KMS）

  上传文件时，可以使用指定的CMK ID或者默认KMS托管的CMK进行加解密操作。这种场景适合于大量的数据加解密。数据无需通过网络发送到KMS服务端进行加解密，这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。

- 使用OSS完全托管加密（SSE-OSS）

  基于OSS完全托管的加密方式，是Object的一种属性。OSS服务器端加密使用AES256加密每个对象，并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。该方式适合于批量数据的加解密。

> 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。

操作方式

- 操作步骤请参考#unique_376。

- SDK示例请参考：
  - Java SDK
  - Python SDK
  - Go SDK

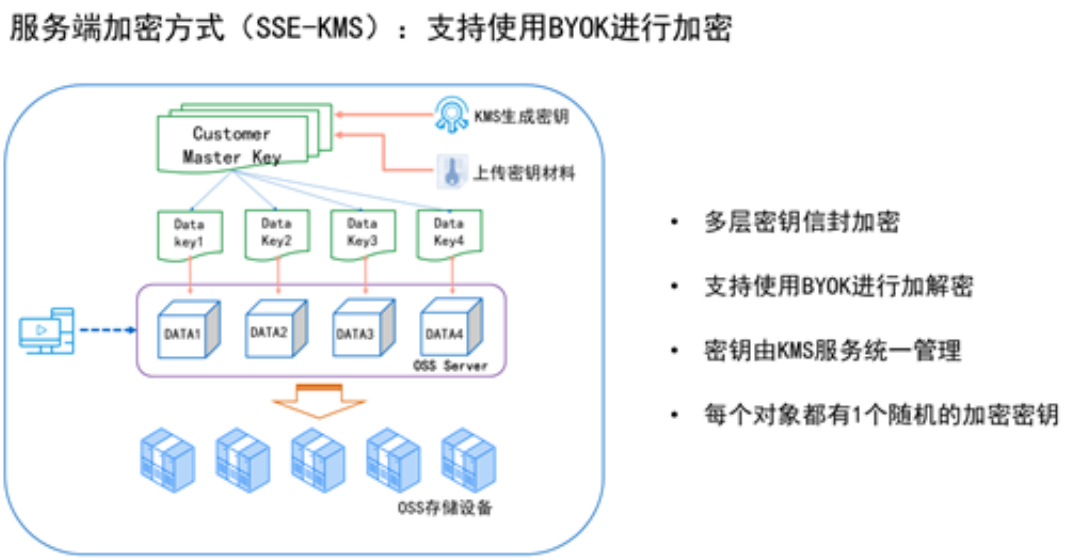
原理介绍

- 使用KMS托管密钥进行加解密

  KMS(Key Management Service)是阿里云提供的一款安全、易用的管理类服务。用户无需花费大量成本来保护密钥的保密性、完整性和可用性，借助密钥管理服务，用户可以安全、便捷的使用密钥，专注于开发加解密功能场景。用户可以通过KMS控制台中查看和管理KMS密钥。

  除了采用AES-256加密算法外，KMS负责保管用户主密钥CMK（对数据密钥进行加密的密钥），以及生成数据加密的密钥，通过信封加密机制，进一步防止未经授权的数据访问。

  SSE-KMS服务器端加密的逻辑示意图如下。



  关于Customer Master Key的生成方式有多种：

  - 使用OSS默认托管的KMS密钥

    您可以将Bucket默认的服务器端加密方式设置为KMS且不指定具体的CMK ID，也可以在上传Object或修改Object的meta信息时，在请求中携带`X-OSS-server-side-`

encryption并指定其值为KMS且不指定具体的CMK ID。OSS将使用默认托管的CMK生成不同的密钥来加密不同的对象，并且在下载时自动解密。

- 使用BYOK进行加密

  服务器端加密支持使用BYOK进行加密，您可以将Bucket默认的服务器端加密方式设置为KMS并指定具体的CMK ID，也可以在上传Object或修改Object的meta信息时，在请求中携带X-OSS-server-side-encryption，指定其值为KMS，并指定X-OSS-server-side-encryption-key-id为具体的CMK ID。OSS将使用指定的CMK生成不同的密钥来加密不同的对象，并将加密Object的CMK ID记录到对象的元数据中，因此具有解密权限的用户下载对象时会自动解密。

  使用的BYOK材料来源有两种：

  ■ 由阿里云提供的BYOK材料：在KMS平台创建密钥时，选择密钥材料来源为阿里云KMS。

  ■ 使用用户自有的BYOK材料：在KMS平台创建密钥时，选择密钥材料来源为外部，并按照要求导入外部密钥材料。导入外部密钥可参考文档#unique_380。

  说明：

  ■ 目前使用BYOK材料进行加密处于公测中，请联系技术支持添加权限。

  ■ 原数据中用于加密数据的数据密钥也会被加密，并且作为Object的元数据信息一并存储。

  ■ KMS托管密钥的服务器端加密方式仅加密对象数据，不会加密任何对象的元数据。

  ■ 使用子账号对数据使用指定的KMS密钥加密时，子账号需取得KMS相关权限，详情请参考#unique_381。

· 使用OSS完全托管加密

  数据加密密钥的生成和管理，由OSS负责，并采用高强度、多因素的安全措施进行保护。数据加密的算法采用使用行业标准的强加密算法AES-256（即256位高级加密标准）。

  OSS的服务器端加密是Object的一个属性。您可以将Bucket默认的服务器端加密方式设置为AES256，也可以在上传Object或修改Object的meta信息时，在请求中携带X-OSS-server-side-encryption并指定其值为AES256，即可以实现该Object的服务器端加密存储。

17 静态网站托管

17.1 配置静态网站托管

您可以通过 OSS API 的 PutBucketWebsite 接口将自己的存储空间配置成静态网站托管模式，并通过存储空间域名访问该静态网站。

说明:

配置静态网站托管的 API 详细接口信息请参考PutBucketWebsite。

静态网站托管配置生效后，假如这个Bucket在杭州，那么这个静态网站的访问域名为:

```
http://<Bucket>.oss-cn-hangzhou.aliyuncs.com/
```

说明:

针对中国大陆、中国香港地区的 OSS，如果直接使用OSS默认域名从互联网访问OSS上网页类型文件，Response Header 中会自动加上 `Content-Disposition:'attachment=filename;'`。即从浏览器访问网页类型文件时，会以附件形式进行下载。若用户使用自有域名访问OSS的请求，Response Header 中不会加上此信息。使用自有域名访问 OSS，请参考绑定自定义域名。

为了使您更方便地管理在 OSS 上托管的静态网站，OSS 支持两种功能:

- 静态页面支持（Index Document Support）

  静态页是指当用户直接访问静态网站根域名时，OSS 返回的默认静态页（相当于网站的 index.html）。如果您为一个 Bucket 配置了静态网站托管模式，就必须指定一个静态页。
- 错误页面支持（Error Document Support）

  错误页面是指在用户访问该静态网站时，如果遇到 HTTP 4XX 错误时（最典型的是 404 NOT FOUND 错误），OSS 返回给用户的错误页面。通过指定错误页面，您可以为您的用户提供恰当的出错提示。

例如：设置索引页面为 index.html，错误页面为 error.html，Bucket 为 oss-sample，Endpoint 为 oss-cn-hangzhou.aliyuncs.com，那么:

- 用户访问 `http://oss-sample.oss-cn-hangzhou.aliyuncs.com/` 和 `http://oss-sample.oss-cn-hangzhou.aliyuncs.com/directory/` 的时候，相当于访问 `http://oss-sample.oss-cn-hangzhou.aliyuncs.com/index.html`。

- 用户访问 `http://oss-sample.oss-cn-hangzhou.aliyuncs.com/object` 的时候，如果 object 不存在，OSS 会返回`http://oss-sample.oss-cn-hangzhou.aliyuncs.com/error.html`。

操作方式

操作方式	说明
控制台	Web应用程序，直观易用
Java SDK	丰富、完整的各类语言SDK demo
Python SDK	
PHP SDK	
Go SDK	
C SDK	
.NET SDK	
Node.js SDK	
Ruby SDK	

细节分析

- 所谓静态网站是指所有的网页都由静态内容构成，包括客户端执行的脚本，例如 JavaScript。OSS 不支持涉及到需要服务器端处理的内容，例如 PHP，JSP，ASP.NET等。
- 如果您想使用自己的域名来访问基于 Bucket 的静态网站，可以通过绑定自定义域名来实现。
- 将一个 Bucket 设置成静态网站托管模式时：
  - 索引页面是必须指定的，错误页面是可选的。
  - 指定的索引页面和错误页面必须是该 Bucket 内的 Object。

  说明:
  若您的存储空间为归档类型，需保证这两个 Object 为标准存储，或处于解冻状态，否则会导致静态网页无法被访问。
- 将一个 Bucket 设置成静态网站托管模式后：
  - 对静态网站根域名的匿名访问，OSS 将返回索引页面；对静态网站根域名的签名访问，OSS 将返回 GetBucket 结果。
  - 访问静态网站根域名或者访问不存在的 Object 时，OSS 会返回给用户设定的 Object，对此返回的流量和请求将会计费。

更多参考

- #unique_94
- 教程示例：配置静态网站托管

17.2 教程示例：使用自定义域名设置静态网站托管

假设您要在阿里云 OSS 上托管静态网站。您注册了域名（例如，examplewebsite.com）并且想要从 OSS 内容响应对 `http://examplewebsite.com` 和 `http://www.examplewebsite.com` 的请求。不论您是已经有了要在 OSS 上托管的静态网站，还是要从头开始创建，都可以通过以下示例了解如何在阿里云 OSS 上托管网站。

准备工作

本教程涉及到以下服务：

- 域名注册

  如果您没有注册域名（如`examplewebsite.com`），请选择一个注册商进行注册。阿里云也提供域名注册服务。详情请参考 阿里云域名服务。

- 阿里云 OSS

  您可以使用阿里云 OSS 创建 bucket，上传示例网页，配置权限允许所有人查看内容，然后配置 bucket 的网站托管功能。在本示例中，因为要允许对 `http://examplewebsite.com` 和 `http://www.examplewebsite.com` 的请求，所以需要创建两个 bucket。您可以仅在一个 bucket 中托管内容，再配置另一个 bucket 将请求重定向到托管内容的 bucket。

- 云解析

  您可以将阿里云解析配置为域名解析服务 (DNS) 提供商。在本示例中，您可以将域名添加到云解析并定义 CNAME 记录，这样能够使用您的域名而不是 OSS 分配的访问域名来访问 OSS bucket。

  - 在本示例中，我们使用阿里云解析。但您可以通过大多数注册商定义指向 OSS bucket 的 CNAME 记录。

说明：

本教程使用 `examplewebsite.com` 作为域名。请使用您注册的域名替换此域名。

步骤 1：注册域名

如果您已有注册域名，可以跳过该步骤。如果您从未托管过网站，请先注册一个域名，例如 `examplewebsite.com`。您可以使用阿里云域名服务注册一个域名。

详情请参考购买域名。

步骤 2：创建和配置 bucket 并上传数据

您可以创建两个 bucket，以同时支持来自根域名（如 `examplewebsite.com`）和子域名（如 `http://www.examplewebsite.com`）的请求。一个 bucket 用于存储内容，另一个 bucket 用于将请求重定向到存储内容的 bucket。

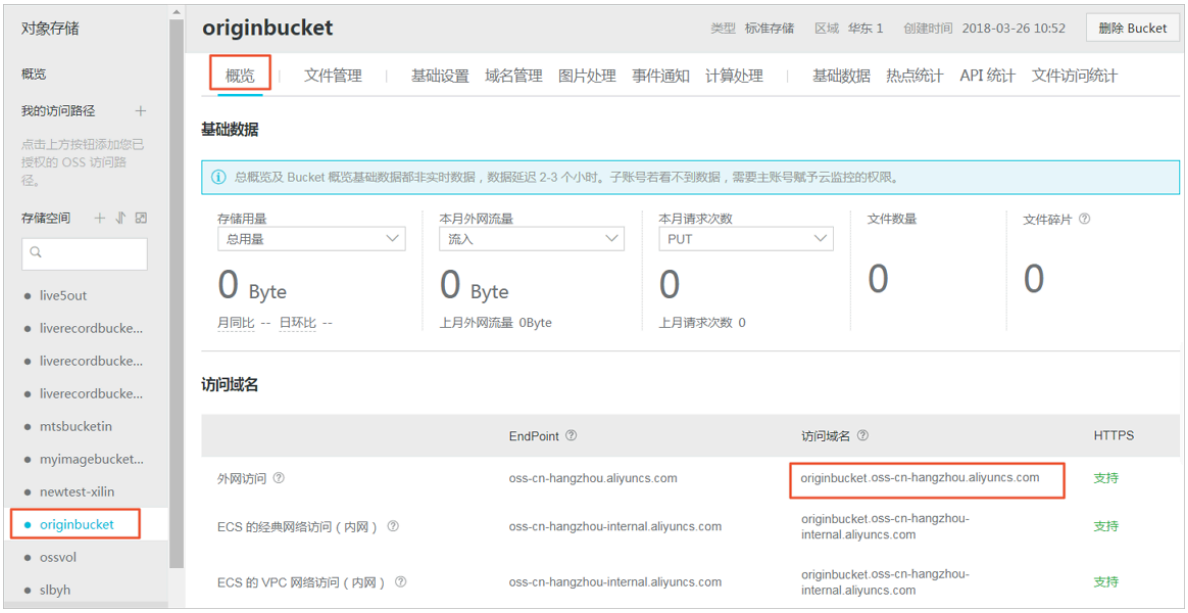
步骤 2.1：创建两个 bucket

在该步骤中，使用阿里云帐户登录 OSS 控制台，并创建以下两个 bucket：

- originbucket：用于存储内容
- redirectbucket：用于将请求重定向到originbucket

1. 登录OSS 控制台。
2. 创建两个 bucket（例如originbucket和redirectbucket）。将两个 bucket 的读写权限设置为公共读取，以便所有人都可以查看 bucket 的内容。

   详细流程，请参考创建 bucket。
3. 记下originbucket和redirectbucket的访问域名。在后续步骤中将会用到它们。您可以在 bucket 的概览页面上找到 bucket 的访问域名，如下图所示。



4. 将网站数据上传到 originbucket。

   您将在根域 bucket originbucket 外托管内容，并且将针对子域 bucket redirectbucket 的请求重定向到根域 bucket originbucket。您可以在任一 bucket 中存储内容。

   本示例将内容托管在originbucket bucket 中。内容可以是任何类型的文件，例如文本文件、照片和视频。如果您尚未创建网站，您在本示例中仅需要两个文件。一个文件用作网站首页，另一个文件用作网站的错误页面。

   例如，使用以下 HTML 创建一个名为 index.html 的文件，并将其上传到 bucket。在后续步骤中，将使用此文件名作为网站的默认首页。

```
<html>
   <head>
       <title>Hello OSS!</title>
       <meta charset="utf-8">
   </head>
   <body>
       <p>开始阿里云 OSS 托管</p>
       <p>这是索引页面</p>
   </body>
 </html>
```

   您可以使用以下 HTML 创建另一个名为error.html 的文件，并将其上传到bucket。此文件用作网站的 404 错误页面。在后续步骤中，将使用此文件名作为网站的默认404 页面。

```
<html>
<head>
   <title>Hello OSS!</title>
   <meta charset="utf-8">
</head>
<body>
   <p>这是 404 错误页面</p>
</body>
</html>
```

步骤 2.2：配置 bucket 的网站托管功能

在配置 bucket 的网站托管功能后，可以通过 OSS 分配的访问域名访问该网站。

在此步骤中，将 originbucket 配置为网站。

1. 登录OSS 控制台。
2. 从 bucket 名称列表中，选择 originbucket。
3. 单击基础设置页签，找到静态页面区域。

4. 单击设置，然后输入以下信息：
   - 默认首页：索引页面（相当于网站的 index.html）。只能使用已存储在 bucket 中的 HTML 文件。在本示例中，输入*index.html* 。
   - 默认 404 页：当访问错误路径时，返回默认的 404 页面。只能使用已存储在 bucket 中的 HTML 和图像文件。在本示例中，输入 *error.html*。
5. 单击保存。

步骤 2.3：配置索引页面的重定向功能

配置 originbucket的默认首页和错误页面之后，您还需要配置redirectbucket 的默认首页。

参考以下步骤配置索引页面的重定向功能：

1. 登录OSS 控制台。
2. 从 bucket 名称列表中，选择 `redirectbucket`。
3. 单击基础设置页签，找到静态页面区域。
4. 单击设置，然后在默认首页文本框中输入 *index.html*。
5. 单击保存。

步骤 3：将域名绑定到 OSS bucket

现在，您已有了根域名`examplewebsite.com`和OSS bucketoriginbucket，接下来您需要将域名绑定到 OSS bucket，以便能够使用您的域名而不是 OSS 分配的域名来访问 OSS bucket。

在本示例中，在将域名`examplewebsite.com`绑定到OSS bucketoriginbucket之前，必须使用 OSS 分配的域名 originbucket.oss-cn-beijing.aliyuncs.com 访问 bucket originbucket。在绑定域名 `examplewebsite.com` 后，可以使用此 `examplewebsite.com` 来访问 OSS bucket。

同样，还需要将子域名`www.examplewebsite.com`绑定到OSS bucketredirectbucket，以便能够使用`www.examplewebsite.com`而不是 OSS 分配的域名 originbucket.oss-cn-beijing.aliyuncs.com 来访问 OSS bucket。

参考以下步骤将根域名 `examplewebsite.com`绑定到OSS bucketoriginbucket：

1. 登录OSS 控制台。
2. 从 bucket 名称列表中，选择 `originbucket`。
3. 单击域名管理页签。
4. 单击绑定用户域名。
5. 在用户域名文本框中，输入根域名`examplewebsite.com`。

6. 将 CNAME 记录定义为 originbucket。
   - 如果已在阿里云帐户下解析域名，则可以打开自动添加 CNAME 记录开关。然后单击提交。
   - 如果未在阿里云主帐户下解析域名，自动添加 CNAME 记录开关处于禁用状态。执行以下步骤，手动添加 CNAME 记录，然后单击提交。
     a. 在云解析中添加域名。
        - 如果域名是在阿里云中注册的，它会自动添加到云解析列表中。您可以跳过该步骤。
     b. 在云解析控制台中，找到域名。
     c. 单击域名。
     d. 单击添加解析。
     e. 在添加解析对话框中，从记录类型下拉框中选择CNAME，然后在记录值文本框中输入 bucket 的 OSS 域名。在本示例中，输入originbucket.oss-cn-beijing.aliyuncs.com 。
     f. 单击确认。
7. 参考上述步骤，将子域名`www.examplewebsite.com`绑定到 OSS bucketredirectbucket。

步骤 4：配置网站重定向功能

您已配置 bucket 的网站托管功能并将自定义域名绑定到 OSS bucket，接下来需要配置 redirectbucket，将`http://www.examplewebsite.com`的所有请求重定向到 `http://examplewebsite.com`。

参考以下步骤配置网站的重定向功能：

1. 登录OSS 控制台。
2. 从 bucket 列表中，选择`redirectbucket`。
3. 单击基础设置页签，找到镜像回源区域。
4. 单击设置，然后单击创建规则。
5. 创建 404 重定向规则，具体步骤如下：
   a. 在回源类型区域，选择重定向。
   b. 在回源条件区域，设置HTTP 状态码为404。
   c. 在回源地址区域，选择添加前后缀，输入originbucket的域名。在本示例中，输入 examplewebsite.com。
   d. 单击确定。

步骤 5：（可选）使用阿里云 CDN 加快网站速度

您可以使用阿里云 CDN 改善网站性能。CDN 让您的网站文件（如 HTML、图像和视频）可供全球各地的数据中心（即，边缘节点）使用。当访问者从您的网站请求文件时，CDN 自动将请求重定向到最近边缘节点上的文件副本。因此下载速度要快于访问者从较远的数据中心请求内容。

CDN 在边缘节点缓存内容的时间长度由您指定。如果访问者请求的内容的缓存时间超过了到期日期，CDN 会检查源服务器，看看是否有新版本的内容可用。如果有新版本，CDN 将新版本复制到边缘节点。您对原始内容所做的更改会在访问者请求内容时复制到边缘节点。但在到期日期前，内容仍为之前的版本。我们建议打开CDN 缓存自动刷新开关，以便您对原始内容所做的更改可以在 CDN 缓存中自动实时刷新。

参考以下步骤使用 CDN 加快originbucket速度：

1. 在 CDN 控制台中添加 CDN 加速域名，并复制该域名的 CNAME 地址。详情请参考CDN 快速入门中的步骤 2：添加加速域名。
2. 在云解析控制台中定义 CNAME 记录。详情请参考配置CNAME流程。
3. 打开CDN 缓存自动刷新开关。
   a. 登录OSS 控制台。
   b. 从 bucket 列表中，选择 `originbucket`。
   c. 单击域名管理页签。
   d. 打开CDN 缓存自动刷新开关。
4. 参考上述步骤，使用 CDN 为`redirectbucket`加速。

步骤 6：测试网站

要验证网站是否正常运行，请在浏览器中尝试以下 URL：

URL	结果
`http://examplewebsite.com`	显示originbucket中的索引文档。
originbucket中不存在的文件的 URL，例如 `http://examplewebsite.com/abc`	显示originbucket中的错误页面。
`http://www.examplewebsite.com`	将您的请求重定向到`http://examplewebsite.com`。
`http://www.examplewebsite.com/abc`	将您的请求重定向到`http://examplewebsite.com/abc`。

说明：

某些情况下，可能需要清除 Web 浏览器的缓存才能看到预期行为。

清理

如果您创建静态网站只是为了练习，别忘了删除分配的阿里云资源以免产生不必要的费用。删除阿里云资源后，网站不再可用。

参考以下步骤进行清理：

1. 禁用阿里云 CDN 控制台中的域名，然后将其删除。
2. 删除云解析控制台中的 CNAME 记录。
3. 删除阿里云 OSS 控制台中的 OSS 文件和 bucket。

18 OSS沙箱

当您的 OSS Bucket 遭受攻击或通过 Bucket 分享违法内容，OSS 会自动将 Bucket 切入沙箱。沙箱中的 Bucket 仍可以正常响应请求，但服务质量将被降级，您的应用可能会有明显感知。

- 对于被攻击的 Bucket，OSS 会将其切入沙箱。若您的 Bucket 遭受攻击，您需要自行承担因攻击而产生的全额费用。
- 若您的用户通过您的 Bucket 分享涉黄、涉政、涉恐等违法内容，也会导致您的 Bucket 被切入沙箱。情节严重者，将被追究法律责任。

说明:

当您的 Bucket 切入到沙箱中后，您将会收到短信和邮件通知。

针对攻击的预防措施

为防止您的 Bucket 因攻击原因被切入沙箱，您可使用高防 IP 来抵御 DDoS 攻击和 CC 攻击。如果您的业务有可能遭受攻击，可以按照如下两种方案添加预防措施。

- 方案一：绑定域名并配置高防 IP

  1. OSS 绑定自定义域名，配置步骤请参考绑定自定义域名。
  2. 根据业务情况，购买合适的高防 IP。
  3. 将高防 IP 绑定到您已设置好的自定义域名上。

  - 防护网站：填写您绑定的 OSS 自定义域名。
  - 协议类型：按照您的实际访问方式选择。
  - 源站IP/域名：选择源站域名，并填写您 OSS 的默认访问域名。

  其他步骤及参数填写，请参考文档设置高防 IP。

- 方案二：配置 ECS 反向代理并配置高防 IP

  基于安全考虑，Bucket 默认域名解析的 IP 是随机变化的，若您期望使用固定 IP 方式访问，推荐使用 ECS 搭建反向代理的方式进行访问 OSS。ECS 上的 EIP 可以绑定高防 IP 以抵御 DDoS 攻击和 CC 攻击。具体可以按照如下方式进行配置：

  1. 创建一个 CentOS 或 Ubuntu 的 ECS 实例，详情请参考创建 ECS 实例。

  > 注意：

若 Bucket 有很大的网络流量或访问请求，请提高 ECS 硬件配置或者搭建 ECS 集群。

2. 配置 ECS 反向代理方式访问 OSS，详细配置可参考配置 OSS 反向代理。
3. 根据业务情况，购买合适的高防 IP。
4. 参考方案一的步骤3。其中， 防护网站填写您 ECS 绑定的域名，源站 IP/域名填写 ECS 的外网 IP。

方案优劣势分析

方案名称	优势	劣势
方案一：绑定域名并配置高防 IP	配置简单：支持控制台图形化设置。	应用场景有局限性：只能对未进入沙箱的 Bucket 提供防护。
方案2：配置 ECS 反向代理并配置高防 IP	· 解决方案具有通用性：能够为已进入沙箱和未进入沙箱的 Bucket 提供防护。 · 适合通过固定 IP 访问 OSS 的场景。	· 配置复杂:需要用户自定搭建 Nginx 反向代理。 · 成本高：需要额外购买 ECS 搭建反向代理。

针对违法内容的预防措施

为防止您的 Bucket 因分享涉黄、涉政、涉恐等违法内容被切入沙箱，建议您开通内容安全服务。阿里云内容安全服务将定期针对您选中的 Bucket 进行检测，有效降低涉黄、涉政、涉恐的风险。

Bucket 已进入沙箱如何处理

针对已经进入沙箱的 Bucket，阿里云不提供迁出服务。若您的 Bucket 被切入沙箱，请按照以下操作执行。

· 因攻击原因导致Bucket被切入沙箱

- 针对已经进入沙箱的 Bucket，请按照方案二配置安全防护措施。

  注意:
  请在 Bucket 所在的 Region 搭建 ECS，并且 proxy_pass 填写 Bucket 内网域名地址。

- 若您的账号下的 Bucket 曾多次遭受攻击。那么，您后续新建的 Bucket 默认也会进入沙箱。此时，针对新建 Bucket 的安全访问措施如下：

  1. 购买高防 IP。
  2. 通过工单系统提交“新建 Bucket 默认不进入沙箱申请”。
  3. 申请通过后，参照方案一进行配置。

- 因发布违法内容导致 Bucket 被切入沙箱
  - 针对已经入沙箱的 Bucket，请按照如下操作执行：
    1. 开通内容安全服务，定期检测您的 Bucket，保证不再发布违规内容。
    2. 请按照方案二配置 ECS 反向代理，通过代理服务器进行访问。

       注意：
       请在 Bucket 所在的 Region 搭建 ECS，并且 proxy_pass 填写 Bucket 内网域名地址。
  - 若您多个 Bucket 同时发布违法内容，或单个 Bucket 多次发布违法内容。那么，您后续新建的 Bucket 默认也会进入沙箱。此时，您需按如下操作执行：
    1. 购买内容安全产品。
    2. 通过工单系统提交“新建 Bucket 默认不进入沙箱申请”。
    3. 申请通过后，配置内容安全检测，定期检测您新建的 Bucket。

19 监控服务

19.1 监控服务概览

OSS监控服务为您提供系统基本运行状态、性能以及计量等方面的监控数据指标，并且提供自定义报警服务，帮助您跟踪请求、分析使用情况、统计业务趋势，及时发现以及诊断系统的相关问题。

OSS监控指标主要分为基础服务指标、性能指标和计量指标，详见OSS监控指标参考。

高实时性

高实时性能够暴露可能隐藏的峰谷问题，显示出实际的波动情况，有助于分析和评估业务场景。OSS监控指标的实时性（除了计量指标）是按照分钟粒度采集聚合的，输出延时不超过1分钟，即每分钟内的用户信息都会聚合成一个值，并在一分钟内输出，代表这一分钟的监控情况。

计量指标相关说明

为了保持和计费策略的统一，计量指标的收集和展现存在一定的特殊性，说明如下：

- 计量指标数据是按照小时粒度输出的，即每个小时内的资源计量信息都会聚合成一个值，代表这个小时总的计量情况。
- 计量指标数据会有近半个小时的延时输出。
- 计量指标数据的数据时间是指该数据所统计时间区间的开始时间。
- 计量采集截止时间是当月最后一条计量数据所统计时间区间的结束时间，如果当月没有产生任何一条计量监控数据，那么计量数据采集截止时间为当月1号0点。
- 计量指标数据的展示都是尽最大可能推送的，准确计量请参考费用中心—使用记录。

举个例子，假设用户只使用PutObject这个请求上传数据，每分钟平均10次。那么在2018-05-10 08:00:00到2018-05-10 09:00:00这一个小时时间区间内，用户的PUT类请求数的计量数据值为600次（10*60分钟），数据时间为2018-05-10 08:00:00，这条数据将会在2018-05-10 09:30:00左右被输出。如果这条数据是从2018-05-01 00:00:00开始到现在的最后一条计量监控数据，那么当月的计量数据采集截止时间就是2018-05-10 09:00:00。如果2018年5月该用户没有产生任何的计量数据，那么计量采集截止时间为2018-05-01 00:00:00。

OSS报警服务

每个账号最多能够设置1000项报警规则。除计量指标和统计指标，其他的监控指标均可配置为报警规则加入报警监控，并且一个监控指标可以配置为多个不同的报警规则。

- 报警服务相关概念请参见报警服务概览。

- OSS报警服务使用指南请参见使用报警服务。
- OSS具体监控指标请参见监控指标参考。

监控数据保留策略

监控数据保留31天，过期自动清除，如果需要离线分析监控数据或者长期下载并保存历史监控，需要使用工具或者编写代码来读取云监控数据存储，请参见OpenAPI访问监控数据。

控制台展示最近7天的数据，如果希望查询7天以上的历史数据，建议使用云监控提供的SDK进行查询，请参见OpenAPI访问监控数据。

OpenAPI访问监控数据

OSS服务的相关监控指标数据可以通过云监控提供的OpenAPI访问，使用方法请参见：

- 云监控SDK参考
- 访问监控数据

监控、诊断和故障排除

监控诊断和故障排除通过详细介绍以下各个方面的内容帮助您更好的了解OSS服务的运行状态并进行自主诊断和故障排除：

- 服务监控

  介绍如何使用监控服务持续监控OSS存储服务的运行状况和性能。
- 跟踪诊断

  介绍如何使用OSS监控服务和logging记录功能诊断问题，以及如何关联各种日志文件中的相关信息进行跟踪诊断。
- 故障排除

  提供常见的问题场景和故障排除方法。

注意事项

OSS Bucket全局唯一，如果删掉Bucket之后再创建同名的Bucket，那么被删掉的Bucket的监控以及报警规则会作用在新的同名Bucket上。

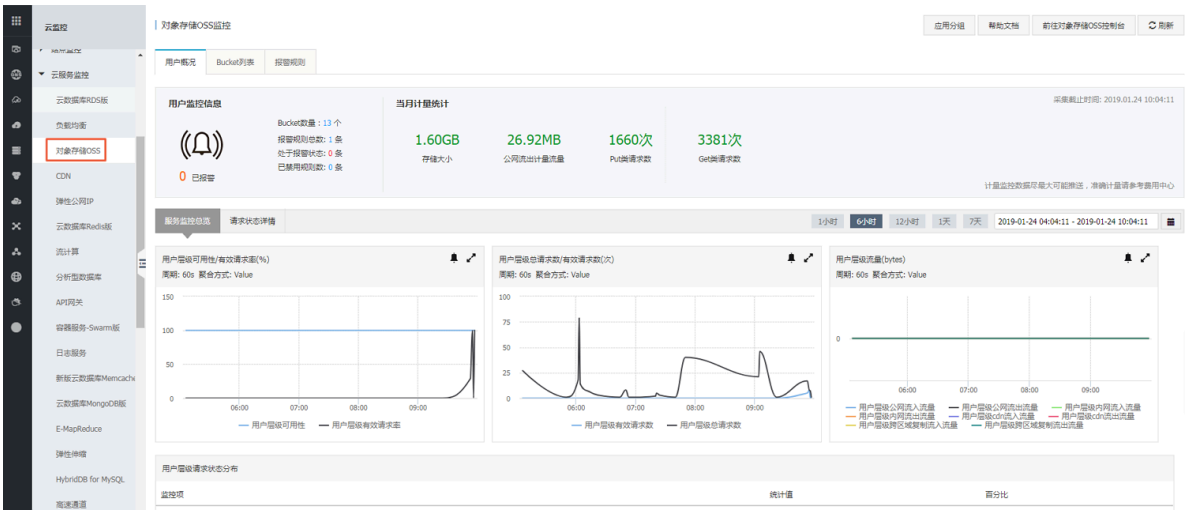
19.2 使用监控服务

OSS监控服务入口

OSS监控服务处于云监控控制台中。可以通过如下两种方式进入。

- 登录OSS管理控制台，在OSS概览页右边单击设置报警规则，进入OSS监控服务。

- 直接进入云监控控制台查看OSS监控服务。



OSS监控服务页面

OSS监控服务主页的主体由如下三部分组成。

- 用户概况
- Bucket列表
- 报警规则

该页面没有自动刷新功能，可以单击右上角的刷新按钮自动更新数据信息。

单击前往对象存储OSS控制台可以直接进入OSS控制台界面。

用户概况

用户概况页面从用户层级监控用户相关的信息。主要包括用户监控信息、当月计量统计和用户层级监控指标三大部分。

- 用户监控信息

  该模块主要展示该账号所拥有的Bucket总数以及相关的报警规则情况。

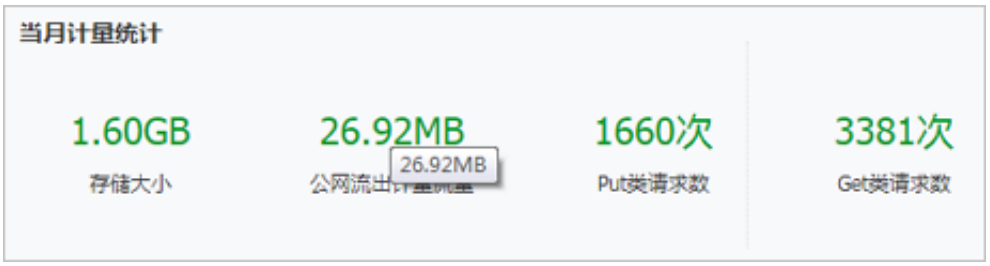
  - 单击Bucket数量的数字，链接到Bucket列表Tab页。
  - 单击报警规则总数的数字，链接到报警规则Tab页。
  - 单击处于报警状态的数字，链接到报警规则Tab页，并且此时该页展示的报警规则均处于告警状态。
  - 单击已禁用规则数的数字，链接到报警规则Tab页，并且此时该页展示的报警规则均被禁用。
  - 单击警铃图标下面的数字，链接到报警规则Tab页，并且此时该页展示的报警规则均处于告警状态。

- 当月计量统计

  当月计量统计展示了从当月1号0点开始，到采集截止时间为止，这段时间内所使用的OSS服务的计费相关的资源信息，包括如下指标：

  - 存储大小
  - 公网流出流量
  - Put类请求数
  - Get类请求数

各个计量框中展示的数据根据量级自动调整单位，鼠标停留在数字上方会显示精确的数值。

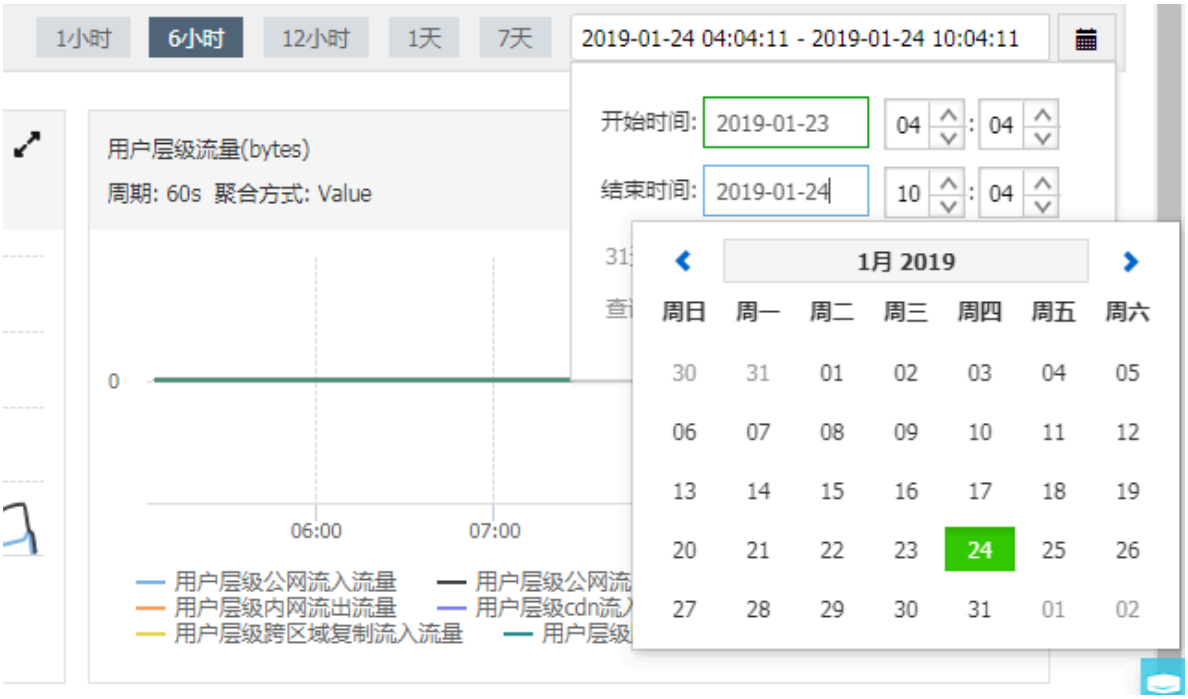


- 用户层级监控指标

该模块主要展示具体的用户层级的监控图表，主要包括服务监控总览和请求状态详情两部分，下面会详细介绍。

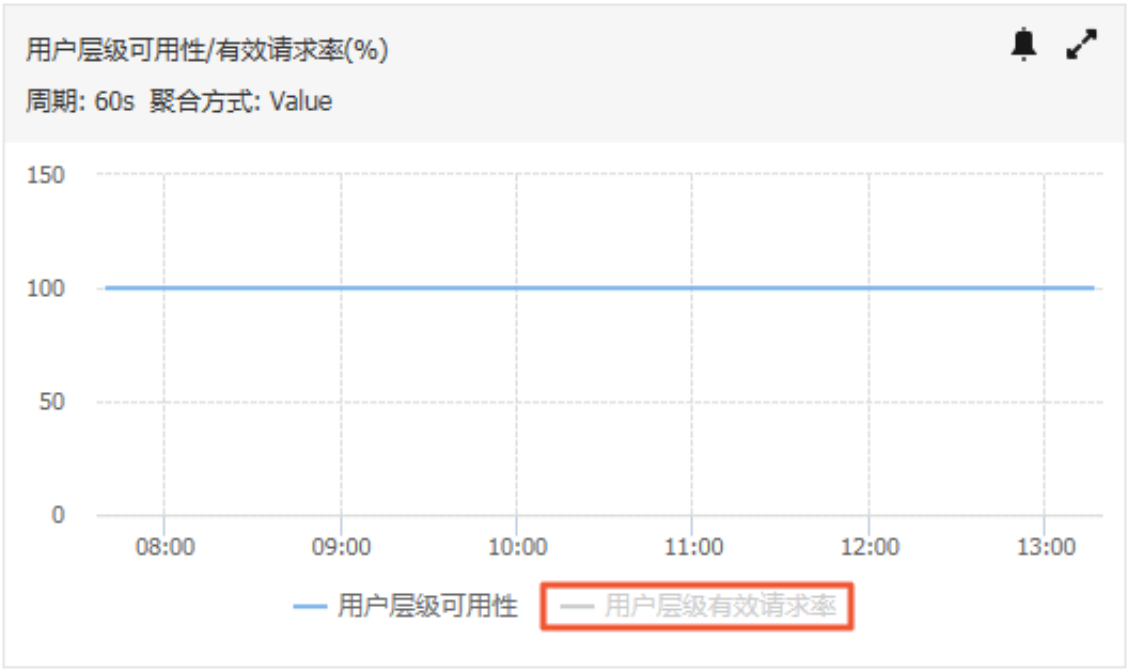
图表展现提供了快速时间范围选择按钮和自定义时间框。

- 快速时间范围选择按钮提供1小时、6小时、12小时、1天和7天的时间范围选择，默认为1小时。
- 自定义时间框可以自定义起始时间和结束时间，精确到分钟级别。注意，不支持查询8天以前的数据。



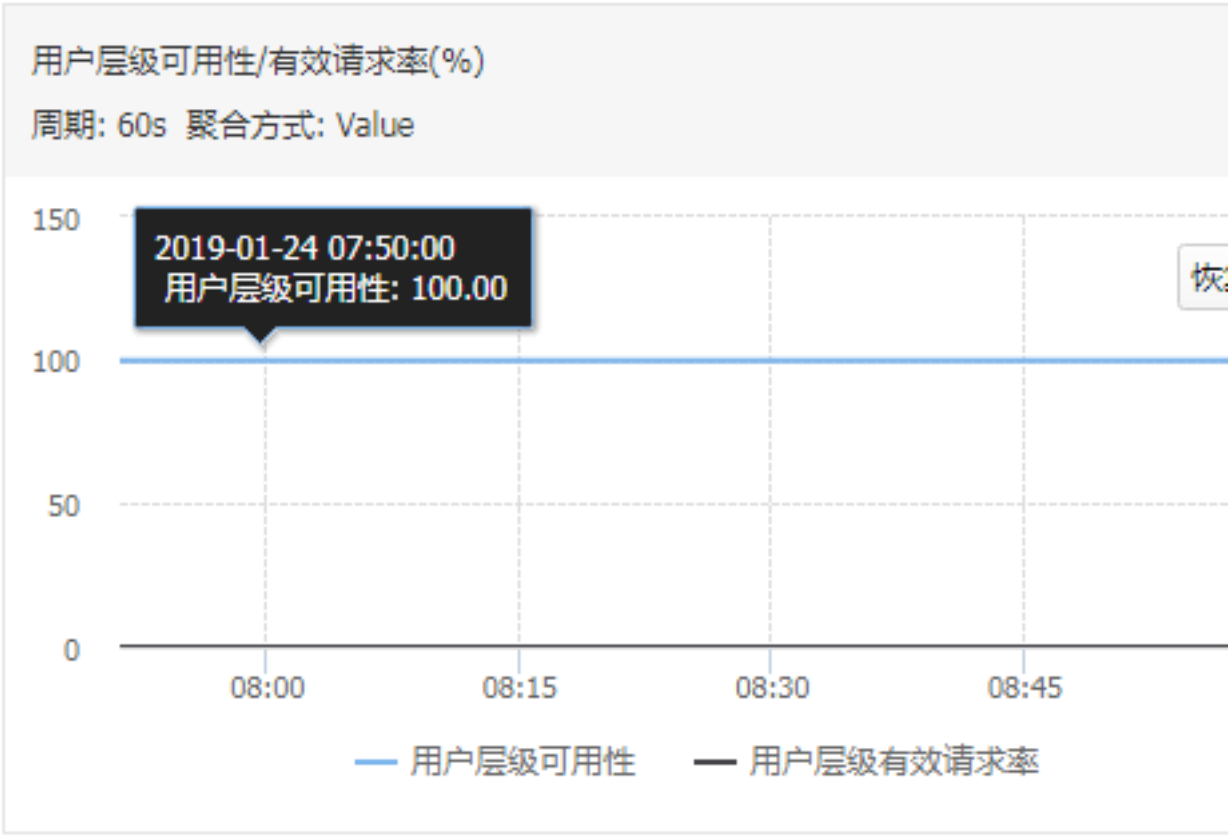
图表展示还支持以下功能：

- 单击相关图例可以将该指标曲线隐去，如下图：



- 单击图形右上图标 可以将图形放大展示。注意，表格不支持放大展示。
- 单击图形右上图标 可以对该图中展示的指标项设置相关报警规则。详见报警服务使用指南。注意，表格和计量参考指标不支持报警设置。

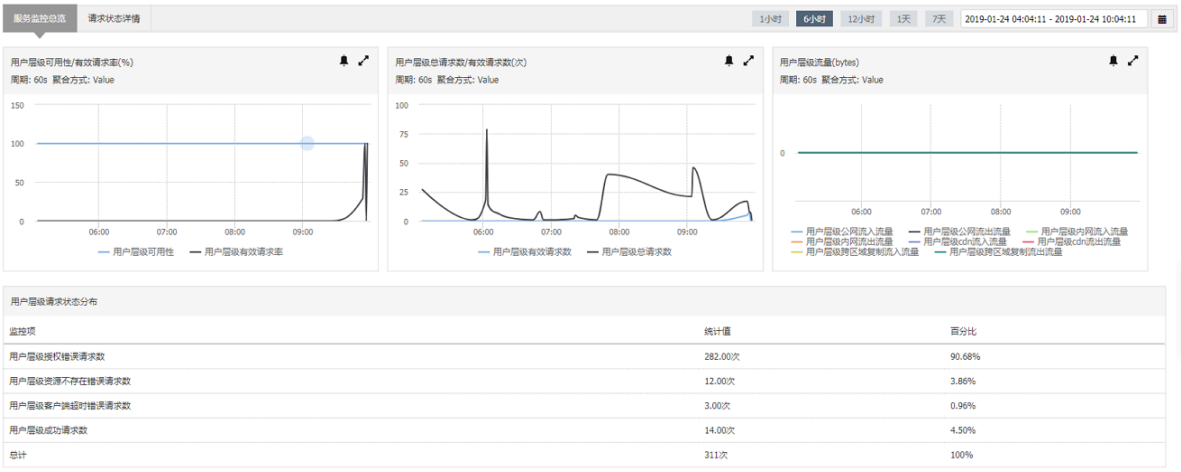
- **鼠标按住图形曲线区域拖放，可以进行时间范围快速调整放大，单击恢复缩放回归到拖放之前的时间范围。**



- **服务监控总览**

  服务监控总览页面主要包括下面监控指标图：

  - **用户层级可用性/有效请求率： 包括可用性和有效请求率两项指标。**
  - **用户层级总请求数/有效请求数： 包括总请求数和有效请求数两项指标。**
  - **用户层级流量： 包括公网流出流量、公网流入流量、内网流出流量、内网流入流量、CDN流出流量、CDN流入流量、跨区域复制流出流量和跨区域复制流入流量八项指标。**
  - **用户层级请求状态分布： 该表格中展示选定时间范围内各个请求类型的个数以及占比。**



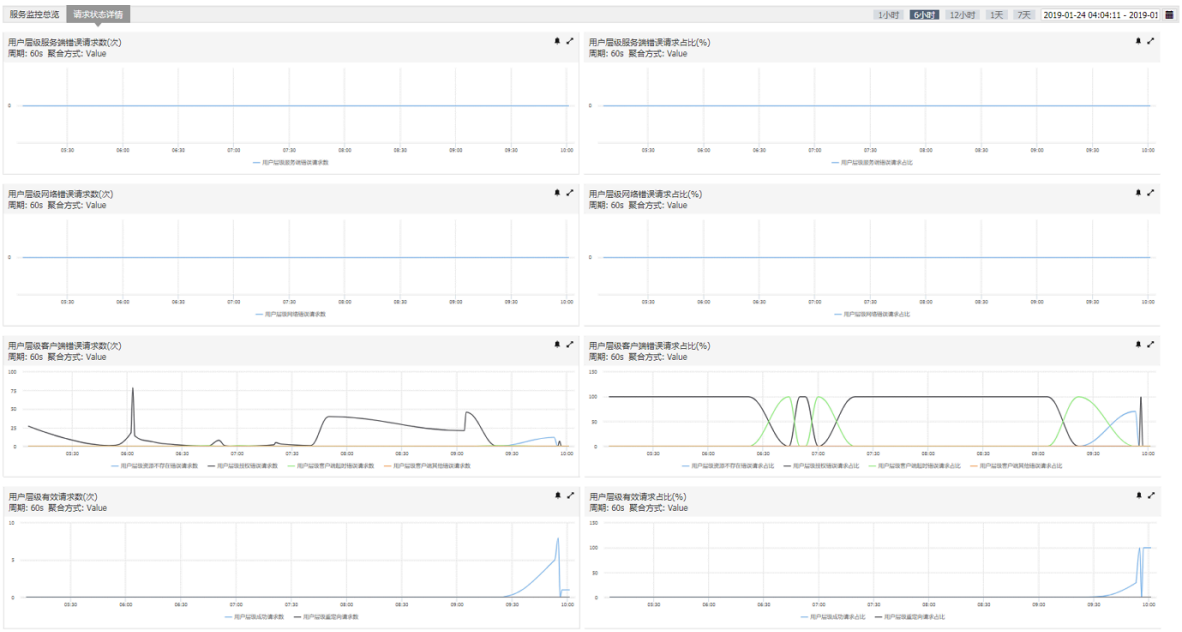
用户层级请求状态分布

监控项	统计值	百分比
用户层级授权错误请求数	282.00次	90.68%
用户层级资源不存在错误请求数	12.00次	3.86%
用户层级客户端超时错误请求数	3.00次	0.96%
用户层级成功请求数	14.00次	4.50%
总计	311次	100%

- **请求状态详情**

  请求状态详情是对请求状态分布统计的一个具体监控，主要包括下面的监控指标图：

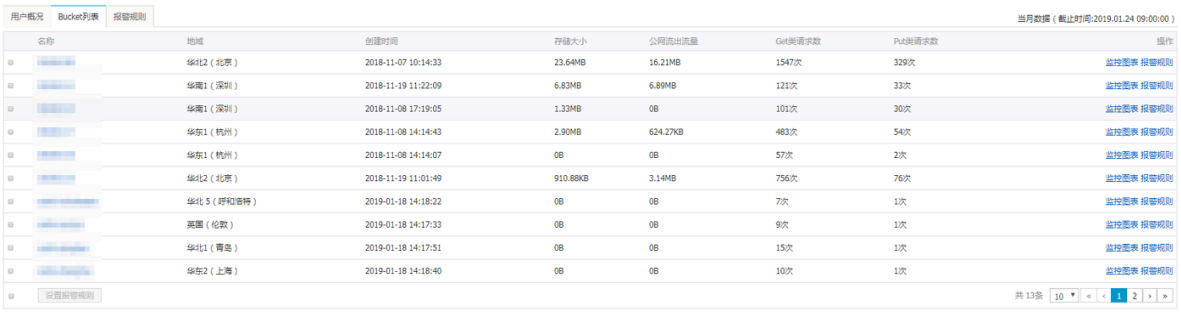
  - **用户层级服务端错误请求数。**
  - **用户层级服务端错误请求占比。**
  - **用户层级网络错误请求数。**
  - **用户层级网络错误请求占比。**
  - **用户层级客户端错误请求数：包括资源不存在错误请求数、授权错误请求数、客户端超时错误请求数和客户端其他错误请求数四项指标。**
  - **用户层级客户端错误请求占比：包括资源不存在错误请求占比、授权错误请求占比、客户端超时错误请求占比和客户端其他错误请求占比四项指标。**
  - **用户层级有效请求数： 包括成功请求数和重定向请求数两项指标。**
  - **用户层级有效请求占比： 包括成功请求占比和重定向请求占比两项指标。**



Bucket列表

- **Bucket列表信息**

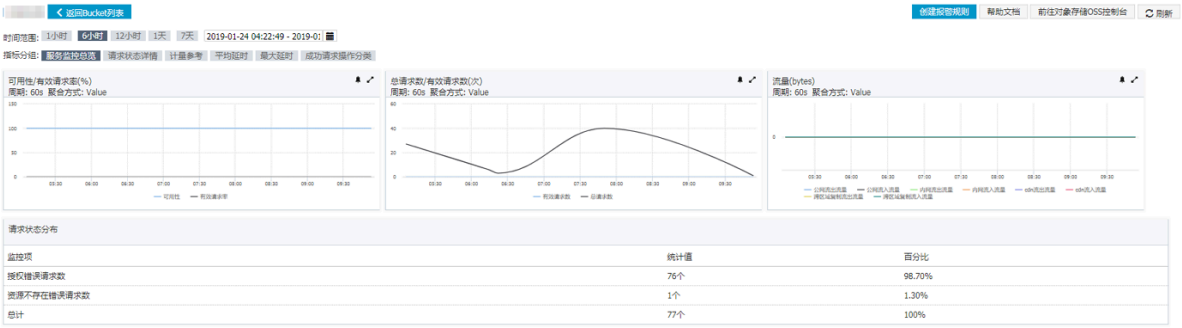
  列表展现该账号所拥有的Bucket的名称、所属地域、创建时间、当月计量数据统计信息以及相关操作。



  - 当月计量统计包括每个Bucket的存储量、公网流出流量、Put类请求数和Get类请求数。
  - 单击监控图表或者对应的Bucket名称，能够进入具体的Bucket监控视图页。
  - 单击报警规则，进入报警规则Tab页，并且展现所有属于该Bucket的报警规则。
  - 通过上面的搜索框能够模糊匹配快速找到具体的Bucket。
  - 选中Bucket复选框，并单击设置报警规则可以批量设置报警规则，详见报警服务使用指南。

- Bucket层级监控视图

  单击Bucket列表中具体的Bucket行中的监控图表，就能进入对应的Bucket的监控视图页，如下图：



  Bucket监控视图页按指标分组进行展示监控图，主要包含六个指标分组：

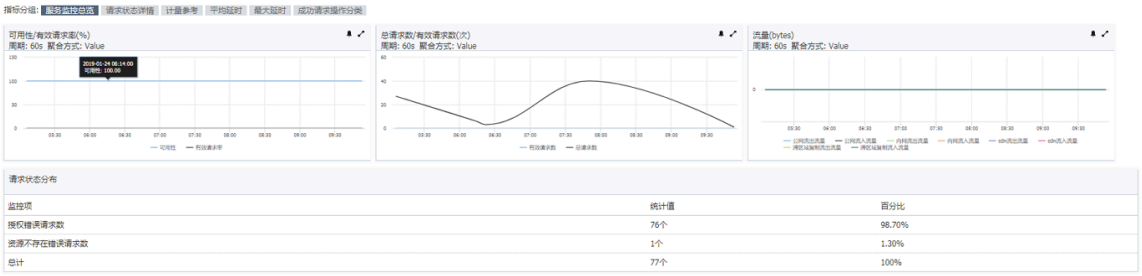
  - 服务监控总览
  - 请求状态详情
  - 计量参考
  - 平均延时
  - 最大延时
  - 成功请求操作分类

  除了计量参考，所有的指标项都是分钟级别聚合展示的。不同于用户层级默认时间展现为最近1小时，Bucket层级的监控展示默认为6小时。单击上方的返回Bucket列表能够回到Bucket列表Tab页。

  - 服务监控总览

    该指标分组同用户层级的服务监控总览，只是从具体的Bucket进行监控，主要包括下面监控指标图：

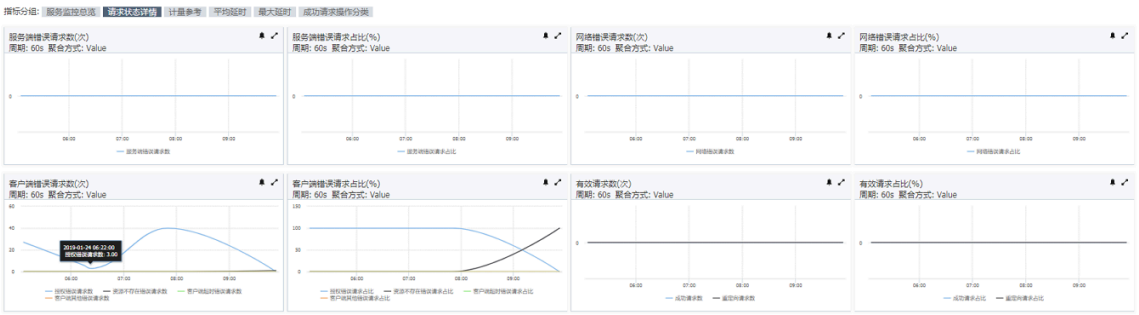
    - 可用性/有效请求率： 包括可用性和有效请求率两项指标。
    - 总请求数/有效请求数： 包括总请求数和有效请求数两项指标。
    - 流量： 包括公网流出流量、公网流入流量、内网流出流量、内网流入流量、cdn流出流量、cdn流入流量、跨区域复制流出流量和跨区域复制流入流量八项指标。
    - 请求状态分布： 该表格中展示选定时间范围内各个请求类型的个数以及占比。



- **请求状态详情**

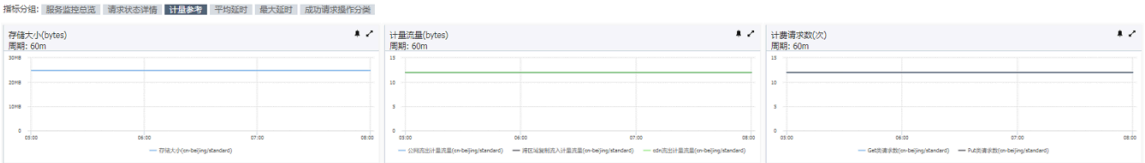
  该指标分组同用户层级的请求状态详情，只是从具体的Bucket进行监控，主要包括下面监控指标图：

  - **服务端错误请求数**
  - **服务端错误请求占比**
  - **网络错误请求数**
  - **网络错误请求占比**
  - **客户端错误请求数：包括资源不存在错误请求数、授权错误请求数、客户端超时错误请求数和客户端其他错误请求数四项指标。**
  - **客户端错误请求占比：包括资源不存在错误请求占比、授权错误请求占比、客户端超时错误请求占比和客户端其他错误请求占比四项指标。**
  - **有效请求数：包括成功请求数和重定向请求数两项指标。**
  - **有效请求占比：包括成功请求占比和重定向请求占比两项指标。**



- **计量参考**

  计量参考分组展示各个计量相关的指标信息，以小时粒度收集展现，如下图所示：



包含以下计量指标监控图：

■ 存储大小

■ 公网流出流量

■ 计费请求数：包括Get类请求数和Put类请求数两项指标项。

如果新建Bucket，需要到当前时间点的下一个整小时点才会采集到新数据，然后在半个小时内展示出来。

- 平均延时

该分组包含分API类型监控的各项平均延时指标，包含如下几个指标图：

■ GetObject请求平均延时

■ HeadObject请求平均延时

■ PutObject请求平均延时

■ PostObject请求平均延时

■ AppendObject请求平均延时

■ UploadPart请求平均延时

■ UploadPartCopy请求平均延时

每个指标图中都包含对应的平均E2E延时和平均服务器延时，如下图所示：

- 最大延时

该分组包含分API类型监控的各项最大延时指标，包含如下几个指标图：

■ GetObject请求最大延时

■ HeadObject请求最大延时

■ PutObject请求最大延时

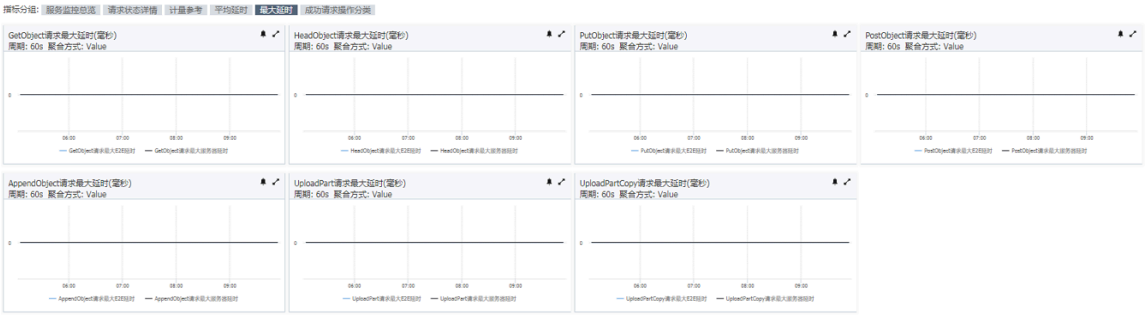
■ PostObject请求最大延时

■ AppendObject请求最大延时

■ UploadPart请求最大延时

■ UploadPartCopy请求最大延时

每个指标图中都包含对应的最大E2E延时和最大服务器延时，如下图所示：



- **成功请求操作分类**

  该分组包含分API类型监控的各项成功请求数指标，包含如下几个指标图：

  - **GetObject成功请求**
  - **HeadObject成功请求**
  - **PutObject成功请求**
  - **PostObject成功请求**
  - **AppendObject成功请求**
  - **UploadPart成功请求**
  - **UploadPartCopy成功请求**
  - **DeleteObject成功请求**
  - **DeleteObjects成功请求**

  如下图所示：

报警规则

报警规则Tab页能够展示和管理报警规则，如下图所示：

报警规则页的使用和相关说明请参见使用报警服务。

监控关注事项以及使用指导

监控关注点以及使用指南请参见监视诊断和故障排除。

19.3 使用报警服务

本文主要介绍OSS监控服务控制台中报警规则的概览及配置方法。

在介绍OSS监控服务控制台之前，请先阅读云监控提供的监控服务文档，了解基本概念并进行报警联系人和报警联系组的配置。

- 报警服务概览
- 报警联系人和联系组

因为OSS的报警规则是根据OSS监控项设置的，所以类似于OSS监控项的维度分类，将其分成两个报警维度：用户层级和Bucket层级。

报警规则页

报警规则页是OSS监控报警相关的规则管理页面，您可以查看、修改、启用、禁用和删除对应的报警规则，而且能够查看该报警规则对应的历史报警情况。

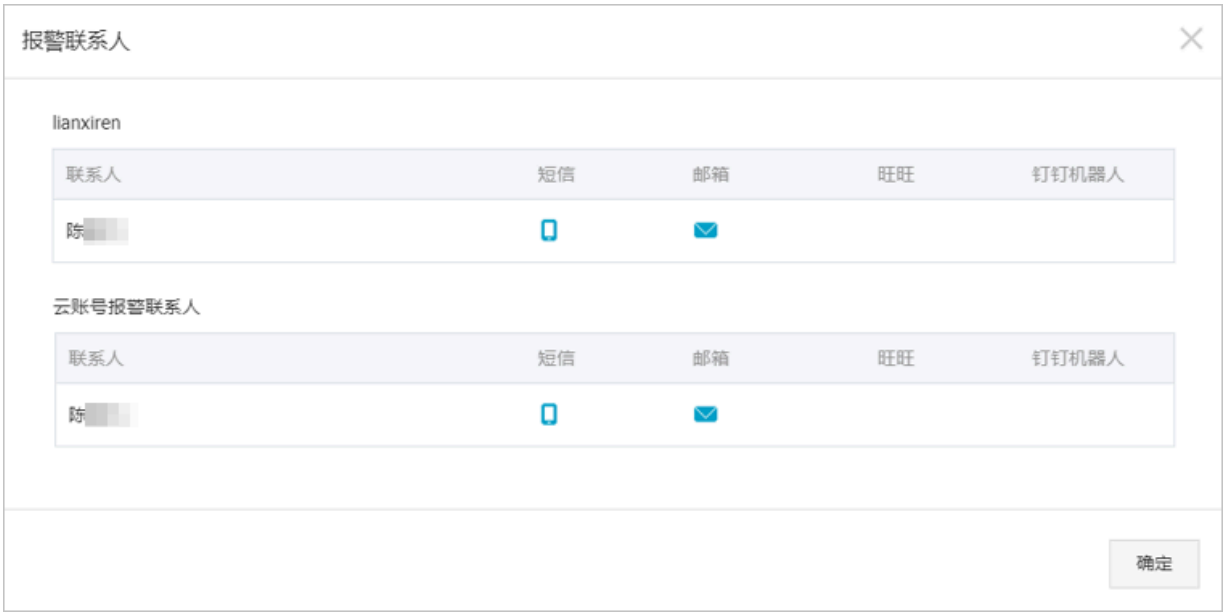
- 单击对应报警规则的查看，可以查看该报警规则的内容。
- 单击对应报警规则的修改，就可以对该报警规则进行修改。
- 单击对应报警规则的删除，就可以删除该报警规则。选中多条报警规则，然后单击表格最下方的删除按钮，可以批量删除报警规则。
- 如果报警规则处于已启用状态，单击该报警规则的禁用，可以禁用该报警规则，报警规则失效，用户不能再收到对应的告警信息。选中多条报警规则，然后单击表格最下方的禁用按钮，可以批量禁用报警规则。

- 如果报警规则处于已禁用状态，单击该报警规则的启用，可以启用该报警规则，报警规则重新生效，能检测并发出对应的告警信息。选中多条报警规则，然后单击表格最下方的启用按钮，可以批量启用报警规则。
- 单击对应报警规则的报警历史，可以查看该报警规则历史发生的所有的告警情况。

相关概念:

- 报警历史指的是该报警规则的状态变化历史，比如从正常变成告警状态，是一个状态变化；从告警变成正常也是状态变化；还有一个特殊的状态变化：通道沉默。
- 当通知对象为通道沉默时，表示该报警规则触发告警之后的指定时间内一直满足报警触发状态(即一直在告警，没有恢复到正常状态)。此时，系统不向通知对象发送告警信息，直到通道沉默时间结束，才会有新的报警信息发送到通知对象。
- 报警历史信息能够保存一个月，即一个月之前的告警信息会被自动清理。查询时一次最多只能查询3天的数据，但不支持查询31天前的数据。

单击具体报警规则的通知对象后的查看，可以显示该通知对象（报警联系组）的成员以及每个成员接收告警信息的方式（短信、邮箱或者旺旺），如下图所示：



查看报警规则

- 快速定位报警规则

  根据报警规则页中下面的控件信息能够快速定位到被搜索的报警规则:

  - 报警维度下拉框: 全部和Bucket层级。当选项为全部时，显示所有用户层级和Bucket层级的报警规则。

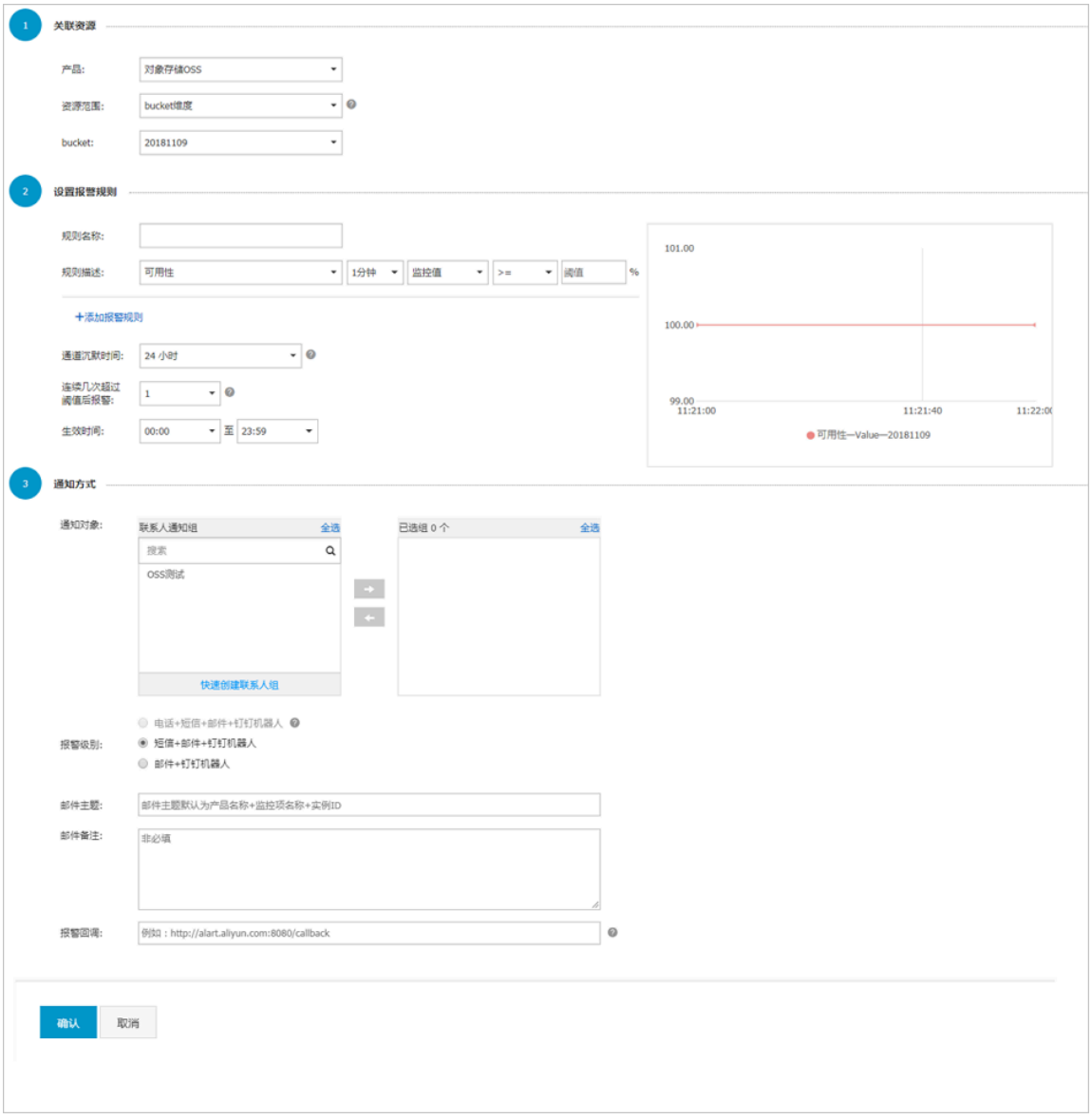
  - Bucket下拉框：当报警维度下拉框为Bucket层级时，这里可以罗列该账号下所有的Bucket。选择对应的Bucket，可以展示属于该Bucket的所有报警规则。

  - 监控项下拉框：罗列所有的OSS的监控项，包括用户层级和Bucket层级的监控项。当选项为全部时，显示用户层级或者Bucket层级所有监控项的报警规则。
  - 状态下拉框：可选择显示处于指定状态的报警规则，如全部、正常状态、报警状态、数据不足、启用、禁用。选择全部时，显示所有状态的报警规则。
  - 维度下拉框：可分维度显示报警规则，如全部用户维度、分组维度、实例维度。

添加报警规则

1. 进入创建报警规则页面，您可以通过如下方式进入：
   - 在用户概况Tab页单击服务监控总览任意图表内的 按钮。
   - 在Bucket列表Tab页选中指定的Bucket，之后单击创建报警规则按钮。
   - 在Bucket列表Tab页选中指定的Bucket，之后单击服务监控总览任意图表内的 按钮。



2. 根据需求配置报警规则。

- 关联资源
  - 产品：选择对象存储OSS。
  - 资源范围：根据您的需求选择全部资源或Bucket维度。
  - Bucket（针对Bucket维度）：选择指定的Bucket，可一次选中多个Bucket。
- 设置报警规则
  - 规则名称：自定义。
  - 规则描述：根据需要选择监控的内容、时间及数值。
  - +添加报警规则：单击可添加多条规则。
  - 通道沉默时间：报警发生后如果未恢复正常，间隔多久重复发送一次报警通知。
  - 连续几次超过阈值后报警：即连续几次报警的探测结果符合您设置的规则描述，才会触发报警。例如：设置的规则描述为“一分钟内公网流出量大于100MBytes，连续3次超过阈值后报警”。则连续出现3次一分钟内公网流出量大于100MBytes的情况，才会触发报警。
  - 生效时间：选择报警规则的生效时间。
- 通知方式
  - 通知对象：若事先已经按照报警联系人和联系组设置好报警联系组，直接选择对应的联系人组；若没有设置过报警联系人组，则单击快创建联系人组，按照提示创建即可。
  - 报警级别：设置报警规则通知的方式。
  - 邮件备注（可选）:设置邮件备注信息。
  - 报警回调：填写公网可访问的URL，云监控会将报警信息通过POST请求推送到该地址，目前仅支持HTTP协议。

3. 单击确认，完成报警规则的设置。

注意事项

目前属于某个Bucket的报警规则存在性并没有与该Bucket的存在性强关联，即如果删除了某个Bucket，属于这个Bucket的报警规则依然存在。建议您在删除Bucket之前先删除对应的报警规则。

19.4 访问监控数据

本文内容为OpenAPI（或者云监控提供的SDK）查询OSS监控服务指标数据的使用提供相关参数依据。

关于Space

OSS监控服务指标项的数据都使用相同的Namespace：acs_oss。

使用Java SDK设置代码示例如下：

```
QueryMetricRequest request = new QueryMetricRequest();
request.setNamespace("acs_oss");
```

关于StartTime和EndTime

云监控的时间参数取值范围采用左开右闭的形式，即(StartTime, EndTime]，即StartTime的数据不会被查询到，而EndTime的数据会被查询到。

另外，云监控数据保留时间为31天，设置的StartTime和EndTime的时间间距不能大于31天，并且不能够查询31天以前的数据。

使用Java SDK设置代码示例如下：

```
request.setStartTime("2016-05-15 08:00:00");
request.setEndTime("2015-05-15 09:00:00");
```

关于Dimensions

OSS监控服务根据用户使用场景，将监控项分为用户和Bucket两个不同的层级进行监控。针对不同的层级监控数据的访问，Dimensions不同：

- 用户层级数据不需要设置Dimensions。
- Bucket层级数据的Dimensions设置为：

  ```
  {"BucketName": "your_bucket_name"}
  ```

  其中，your_bucket_name为属于该用户的某个需要查询的Bucket名称。

  注意，Dimenisons是一个JSON字符串，OSS监控指标的Dimensions只有一对Key-Value。

  使用Java SDK设置代码示例如下：

  ```
  request.setDimensions("{\"BucketName\":\"your_bucket_name\"}");
  ```

关于Period

OSS监控指标除了计量指标，其他所有的指标的聚合粒度均为60s。计量指标的聚合粒度为3600s。

使用Java SDK设置代码示例如下：

```
request.setPeriod("60");
```

关于Metric

OSS监控指标参考中详细介绍的各项指标项对应的Metric名称和层级如下表所示：

Metric	对应指标项名称	单位	层级
UserAvailability	用户层级可用性	%	用户层级
UserRequestValidRate	用户层级有效请求率	%	用户层级
UserTotalRequestCount	用户层级总请求数	次数	用户层级
UserValidRequestCount	用户层级有效请求数	次数	用户层级
UserInternetSend	用户层级公网流出流量	字节	用户层级
UserInternetRecv	用户层级公网流入流量	字节	用户层级
UserIntranetSend	用户层级内网流出流量	字节	用户层级
UserIntranetRecv	用户层级内网流入流量	字节	用户层级
UserCdnSend	用户层级cdn流出流量	字节	用户层级
UserCdnRecv	用户层级cdn流入流量	字节	用户层级
UserSyncSend	用户层级跨区域复制流出流量	字节	用户层级
UserSyncRecv	用户层级跨区域复制流入流量	字节	用户层级
UserServerErrorCount	用户层级服务端错误请求总数	次数	用户层级
UserServerErrorRate	用户层级服务端错误请求占比	%	用户层级
UserNetworkErrorCount	用户层级网络错误请求总数	次数	用户层级
UserNetworkErrorRate	用户层级网络错误请求占比	%	用户层级
UserAuthorizationErrorCount	用户层级客户端授权错误请求总数	次数	用户层级

Metric	对应指标项名称	单位	层级
UserAuthorizationErrorRate	用户层级客户端授权错误请求占比	%	用户层级
UserResourceNotFoundErrorCount	用户层级客户端资源不存在错误请求总数	次数	用户层级
UserResourceNotFoundErrorRate	用户层级客户端资源不存在错误请求占比	%	用户层级
UserClientTimeoutErrorCount	用户层级客户端超时错误请求总数	次数	用户层级
UserClientOtherErrorRate	用户层级客户端超时错误请求占比	%	用户层级
UserClientOtherErrorCount	用户层级客户端其他错误请求总数	次数	用户层级
UserClientOtherErrorRate	用户层级客户端其他错误请求占比	%	用户层级
UserSuccessCount	用户层级成功请求总数	次数	用户层级
UserSuccessRate	用户层级成功请求占比	%	用户层级
UserRedirectCount	用户层级重定向请求总数	次数	用户层级
UserRedirectRate	用户层级重定向请求占比	%	用户层级
Availability	可用性	%	Bucket层级
RequestValidRate	有效请求率	%	Bucket层级
TotalRequestCount	总请求数	次数	Bucket层级
ValidRequestCount	有效请求数	次数	Bucket层级
InternetSend	公网流出流量	字节	Bucket层级
InternetRecv	公网流入流量	字节	Bucket层级
IntranetSend	内网流出流量	字节	Bucket层级
IntranetRecv	内网流入流量	字节	Bucket层级
CdnSend	cdn流出流量	字节	Bucket层级
CdnRecv	cdn流入流量	字节	Bucket层级
SyncSend	跨区域复制流出流量	字节	Bucket层级

Metric	对应指标项名称	单位	层级
SyncRecv	跨区域复制流入流量	字节	Bucket层级
ServerErrorCount	服务端错误请求总数	次数	Bucket层级
ServerErrorRate	服务端错误请求占比	%	Bucket层级
NetworkErrorCount	网络错误请求总数	次数	Bucket层级
NetworkErrorRate	网络错误请求占比	%	Bucket层级
AuthorizationErrorCount	客户端授权错误请求总数	次数	Bucket层级
AuthorizationErrorRate	客户端授权错误请求占比	%	Bucket层级
ResourceNotFoundErrorCount	客户端资源不存在错误请求总数	次数	Bucket层级
ResourceNotFoundErrorRate	客户端资源不存在错误请求占比	%	Bucket层级
ClientTimeoutErrorCount	客户端超时错误请求总数	次数	Bucket层级
ClientTimeoutErrorRate	客户端超时错误请求占比	%	Bucket层级
ClientOtherErrorCount	客户端其他错误请求总数	次数	Bucket层级
ClientOtherErrorRate	客户端其他错误请求占比	%	Bucket层级
SuccessCount	成功请求总数	次数	Bucket层级
SuccessRate	成功请求占比	%	Bucket层级
RedirectCount	重定向请求总数	次数	Bucket层级
RedirectRate	重定向请求占比	%	Bucket层级
GetObjectE2eLatency	GetObject请求平均E2E延时	毫秒	Bucket层级
GetObjectServerLatency	GetObject请求平均服务器延时	毫秒	Bucket层级
MaxGetObjectE2eLatency	GetObject请求最大E2E延时	毫秒	Bucket层级
MaxGetObjectServerLatency	GetObject请求最大服务器延时	毫秒	Bucket层级

Metric	对应指标项名称	单位	层级
HeadObject E2eLatency	HeadObject请求平均E2E延时	毫秒	Bucket层级
HeadObject ServerLatency	HeadObject请求平均服务器延时	毫秒	Bucket层级
MaxHeadObj ectE2eLatency	HeadObject请求最大E2E延时	毫秒	Bucket层级
MaxHeadObj ectServerLatency	HeadObject请求最大服务器延时	毫秒	Bucket层级
PutObjectE 2eLatency	PutObject请求平均E2E延时	毫秒	Bucket层级
PutObjectS erverLatency	PutObject请求平均服务器延时	毫秒	Bucket层级
MaxPutObje ctE2eLatency	PutObject请求最大E2E延时	毫秒	Bucket层级
MaxPutObje ctServerLatency	PutObject请求最大服务器延时	毫秒	Bucket层级
PostObject E2eLatency	PostObject请求平均E2E延时	毫秒	Bucket层级
PostObject ServerLatency	PostObject请求平均服务器延时	毫秒	Bucket层级
MaxPostObj ectE2eLatency	PostObject请求最大E2E延时	毫秒	Bucket层级
MaxPostObj ectServerLatency	PostObject请求最大服务器延时	毫秒	Bucket层级
AppendObje ctE2eLatency	AppendObject请求平均E2E延时	毫秒	Bucket层级
AppendObje ctServerLatency	AppendObject请求平均服务器延时	毫秒	Bucket层级
MaxAppendO bjectE2eLatency	AppendObject请求最大E2E延时	毫秒	Bucket层级
MaxAppendO bjectServerLatency	AppendObject请求最大服务器延时	毫秒	Bucket层级
UploadPart E2eLatency	UploadPart请求平均E2E延时	毫秒	Bucket层级

Metric	对应指标项名称	单位	层级
UploadPart ServerLatency	UploadPart请求平均服务器延时	毫秒	Bucket层级
MaxUploadP artE2eLatency	UploadPart请求最大E2E延时	毫秒	Bucket层级
MaxUploadP artServerLatency	UploadPart请求最大服务器延时	毫秒	Bucket层级
UploadPart CopyE2eLatency	UploadPartCopy请求平均E2E延时	毫秒	Bucket层级
UploadPart CopyServerLatency	UploadPartCopy请求平均服务器延时	毫秒	Bucket层级
MaxUploadP artCopyE2eLatency	UploadPartCopy请求最大E2E延时	毫秒	Bucket层级
MaxUploadP artCopySer verLatency	UploadPartCopy请求最大服务器延时	毫秒	Bucket层级
GetObjectCount	GetObject成功请求数	次数	Bucket层级
HeadObjectCount	HeadObject成功请求数	次数	Bucket层级
PutObjectCount	PutObject成功请求数	次数	Bucket层级
PostObjectCount	PostObject成功请求数	次数	Bucket层级
AppendObje ctCount	AppendObject成功请求数	次数	Bucket层级
UploadPartCount	UploadPart成功请求数	次数	Bucket层级
UploadPart CopyCount	UploadPartCopy成功请求数	次数	Bucket层级
DeleteObjectCount	DeleteObject成功请求数	次数	Bucket层级
DeleteObjectsCount	DeleteObjects成功请求数	次数	Bucket层级

计量类指标的Metric如下表所示，注意聚合粒度为3600s。

Metric	对应指标项名称	单位	层级
MeteringStorageUtilization	存储大小	字节	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。
MeteringGetRequest	Get类请求数	次数	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。
MeteringPutRequest	Put类请求数	次数	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。
MeteringInternetTX	公网流出计量流量	字节	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。
MeteringCdnTX	cdn流出计量流量	字节	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。
MeteringSyncRX	跨区域复制流入计量流量	字节	若设置Dimensions，则为Bucket层级；如果不设置，那么返回的指标为整个用户层级数据。

使用Java SDK设置代码示例如下：

```
request.setMetric("UserAvailability");
```

19.5 监控指标参考

根据用户使用场景，将OSS的指标分为用户层级和Bucket（存储空间）层级两个层级维度进行监控。

另外，为了更好地观察监控数据以及匹配计费策略，除了一般监控项的时间序指标外，OSS对现有的监控指标项进行统计分析，提供了一段时间内的统计指标，如请求状态分布统计和当月计量统计。

除了计量指标和统计指标，所有的指标（时间序指标）都是分钟级别的数据汇总（如求和、求最大值或者求均值等等）。而计量指标是按小时的数据进行汇总的时间序指标。

用户层级指标

用户层级指标是指从用户的账户级别对OSS系统使用的总体情况进行监控的指标信息，是对该账户下的所有的Bucket相关监控数据的汇总。其中包括当月计量统计、服务监控总览和请求状态详情三个方面。

- 服务监控总览

  服务监控总览指标属于基础服务指标，具体指标项详见下表。注意，下面所有的指标都是在用户层级监控的。

服务监控总览指标名称	单位	描述
可用性	%	存储服务的系统可用性衡量指标。通过公式 `1 - 服务端错误请求（返回状态码为5xx）占总请求的百分比`获取
有效请求率	%	有效请求占总请求数的百分比
总请求数	次数	被OSS服务端接收并处理的请求总数
有效请求数	次数	返回状态码为2xx和3xx的请求总数
公网流出流量	字节	通过互联网网络的下行流量
公网流入流量	字节	通过互联网网络的上行流量
内网流出流量	字节	通过服务系统内部网络的下行流量
内网流入流量	字节	通过服务系统内部网络的上行流量
cdn流出流量	字节	开通cdn加速服务之后，通过cdn产生的下行流量，即回源流量
cdn流入流量	字节	开通cdn加速服务之后，通过cdn产生的上行流量

服务监控总览指标名称	单位	描述
跨区域复制流出流量	字节	开通跨区域复制功能之后，数据复制过程产生的下行流量
跨区域复制流入流量	字节	开通跨区域复制功能之后，数据复制过程产生的上行流量

除了以上具体的监控指标，还提供一段时间内的请求状态分布统计，主要是根据返回的状态码或者OSS错误码进行分类的请求的统计信息（被观察时间段内的请求次数总和以及占比），相关的监控指标项信息详见以下请求状态详情的介绍。

- 请求状态详情

请求状态详情指标是指根据请求返回状态码或者OSS错误码进行分类的请求的监控信息，属于基础服务指标，具体指标项详见下表。注意，下面所有的指标都是在用户层级监控的。

请求状态详情指标名称	单位	描述
服务端错误请求总数	次数	返回状态码为5xx的系统级错误请求总数
服务端错误请求占比	%	服务端错误请求总数占总请求数的百分比
网络错误请求总数	次数	HTTP状态码为499的请求总数
网络错误请求占比	%	网络错误请求数占总请求数的百分比
客户端授权错误请求总数	次数	返回状态码403的请求总数
客户端授权错误请求占比	%	授权错误请求数占总请求数的百分比
客户端资源不存在错误请求总数	次数	返回状态码为404的请求总数
客户端资源不存在错误请求占比	%	资源不存在错误请求数占总请求数百分比
客户端超时错误请求总数	次数	返回状态码为408或者返回的OSS错误码为RequestTimeout的请求总数
客户端超时错误请求占比	%	网络错误请求总数占总请求数的百分比
客户端其他错误请求总数	次数	除了以上提到的客户端错误请求之外的其他返回状态码为4xx的请求总数

请求状态详情指标名称	单位	描述
客户端其他错误请求占比	%	客户端其他错误请求数占总请求数的百分比
成功请求总数	次数	返回状态码为2xx的请求总数
成功请求占比	%	成功请求数占总请求数的百分比
重定向请求总数	次数	返回状态码为3xx的请求总数
重定向请求占比	%	重定向请求数占总请求数的百分比

- 当月计量统计

  当月计量统计指标是指从当月的1号0点开始，到当月计量采集截止时间为止，这段时间内计量指标的统计数据。

  目前统计的计量指标如下：

当月计量统计指标名称	单位	描述
存储大小	字节	在计量采集截止时间前属于该用户的所有Bucket占用的存储总大小
公网流出流量	字节	从本月1号0点开始累积到计量采集截止时间为止，用户所使用的所有公网流出流量的总和。
Put类请求数	次数	从本月1号0点开始累积到计量采集截止时间为止，用户所使用的所有Put类请求的总和。
Get类请求数	次数	从本月1号0点开始累积到计量采集截止时间为止，用户所使用的所有Get类请求的总和。

Bucket层级指标

Bucket层级指标是指对具体的存储空间的OSS操作情况进行监控的指标信息，具有更强的业务场景，所以除了类似从用户层面可以监控的服务监控总览和请求状态详情这些基础服务指标项和当月计量统计之外，还有计量参考、延时和成功请求操作分类等计量指标和性能指标。

- 服务监控总览

  监控项指标含义同用户层级的服务监控总览，从具体的Bucket进行监控。

- 请求状态详情

  监控项指标含义同用户层级的请求状态详情，从具体的Bucket进行监控。

- 当月计量统计

  统计方式同用户层级的当月计量统计，从具体的Bucket资源使用情况进行统计。

当月计量统计指标名称	单位	描述
存储大小	字节	在计量采集截止时间前该Bucket占用的存储大小
公网流出流量	字节	从本月1号0点开始累积到计量采集截止时间为止，该Bucket的公网流出流量的总和。
Put类请求数	次数	从本月1号0点开始累积到计量采集截止时间为止，该Bucket的所有Put类请求的总和。
Get类请求数	次数	从本月1号0点开始累积到计量采集截止时间为止，该Bucket的所有Get类请求的总和。

- 计量参考

  计量指标的时间序监控，具体如下：

当月计量统计指标名称	单位	描述
存储大小	字节	该Bucket每小时使用的平均存储大小
公网流出流量	字节	该Bucket每小时的公网流出流量的总和
Put类请求数	次数	该Bucket每小时的Put类请求的总和
Get类请求数	次数	该Bucket每小时的Get类请求的总和

- 延时

  请求延时是系统性能的直观反映。监控服务提供了分钟级别的平均延时和最大延时两类指标，分别反映系统平均响应能力和系统抖动情况。并且根据OSS API请求操作类型进行分类，更细粒度

地反应系统应对不同操作的性能状况。目前只对关于Bucket的操作并且涉及数据操作（不包含对meta操作）的API进行监控。

另外，延时监控指标分别从E2E和服务器两条不同的链路进行收集，便于分析性能热点以及环境问题，其中：

- E2E延时是指向OSS系统发出的成功请求的端到端滞后时间，包括在OSS系统中读取请求、发送响应以及接受响应确认所需的处理时间。
- 服务器延时是指OSS系统成功处理请求所使用的滞后时间，不包括E2E延时中的网络滞后时间。

注意，性能相关指标都是对成功请求（返回状态码为2xx）进行的监控。

具体的监控指标项如下表：

延时指标名称	单位	描述
GetObject请求平均E2E延时	毫秒	请求API为GetObject的成功请求的平均端到端延时
GetObject请求平均服务器延时	毫秒	请求API为GetObject的成功请求的平均服务器延时
GetObject请求最大E2E延时	毫秒	请求API为GetObject的成功请求的最大端到端延时
GetObject请求最大服务器延时	毫秒	请求API为GetObject的成功请求的最大服务器延时
HeadObject请求平均E2E延时	毫秒	请求API为HeadObject的成功请求的平均端到端延时
HeadObject请求平均服务器延时	毫秒	请求API为HeadObject的成功请求的平均服务器延时
HeadObject请求最大E2E延时	毫秒	请求API为HeadObject的成功请求的最大端到端延时
HeadObject请求最大服务器延时	毫秒	请求API为HeadObject的成功请求的最大服务器延时
PutObject请求平均E2E延时	毫秒	请求API为PutObject的成功请求的平均端到端延时
PutObject请求平均服务器延时	毫秒	请求API为PutObject的成功请求的平均服务器延时
PutObject请求最大E2E延时	毫秒	请求API为PutObject的成功请求的最大端到端延时

延时指标名称	单位	描述
PutObject请求最大服务器延时	毫秒	请求API为PutObject的成功请求的最大服务器延时
PostObject请求平均E2E延时	毫秒	请求API为PostObject的成功请求的平均端到端延时
PostObject请求平均服务器延时	毫秒	请求API为PostObject的成功请求的平均服务器延时
PostObject请求最大E2E延时	毫秒	请求API为PostObject的成功请求的最大端到端延时
PostObject请求最大服务器延时	毫秒	请求API为PostObject的成功请求的最大服务器延时
AppendObject请求平均E2E延时	毫秒	请求API为AppendObject的成功请求的平均端到端延时
AppendObject请求平均服务器延时	毫秒	请求API为AppendObject的成功请求的平均服务器延时
AppendObject请求最大E2E延时	毫秒	请求API为AppendObject的成功请求的最大端到端延时
AppendObject请求最大服务器延时	毫秒	请求API为AppendObject的成功请求的最大服务器延时
UploadPart请求平均E2E延时	毫秒	请求API为UploadPart的成功请求的平均端到端延时
UploadPart请求平均服务器延时	毫秒	请求API为UploadPart的成功请求的平均服务器延时
UploadPart请求最大E2E延时	毫秒	请求API为UploadPart的成功请求的最大端到端延时
UploadPart请求最大服务器延时	毫秒	请求API为UploadPart的成功请求的最大服务器延时
UploadPartCopy请求平均E2E延时	毫秒	请求API为UploadPartCopy的成功请求的平均端到端延时
UploadPartCopy请求平均服务器延时	毫秒	请求API为UploadPartCopy的成功请求的平均服务器延时
UploadPartCopy请求最大E2E延时	毫秒	请求API为UploadPartCopy的成功请求的最大端到端延时
UploadPartCopy请求最大服务器延时	毫秒	请求API为UploadPartCopy的成功请求的最大服务器延时

· 成功请求操作分类

配合延时监控，成功请求的监控一定程度上反应了系统处理访问请求的能力。目前只监控关于Bucket的操作中涉及数据操作的API。详细的指标项如下：

成功请求操作分类指标名称	单位	描述
GetObject成功请求数	次数	请求API为GetObject的成功请求数
HeadObject成功请求数	次数	请求API为HeadObject的成功请求数
PutObject成功请求数	次数	请求API为PutObject的成功请求数
PostObject成功请求数	次数	请求API为PostObject的成功请求数
AppendObject成功请求数	次数	请求API为AppendObject的成功请求数
UploadPart成功请求数	次数	请求API为UploadPart的成功请求数
UploadPartCopy成功请求数	次数	请求API为UploadPartCopy的成功请求数
DeleteObject成功请求数	次数	请求API为DeleteObject的成功请求数
DeleteObjects成功请求数	次数	请求API为DeleteObjects的成功请求数

19.6 监控、诊断和故障排除

相对于传统应用程序，开发云端应用虽然降低了用户在基础设施搭建、运维等方面的成本，但却增大了监控、诊断和故障排查的难度。OSS存储服务为您提供了丰富的监控和日志信息，帮助您深刻洞察程序行为，及时发现并快速定位问题。

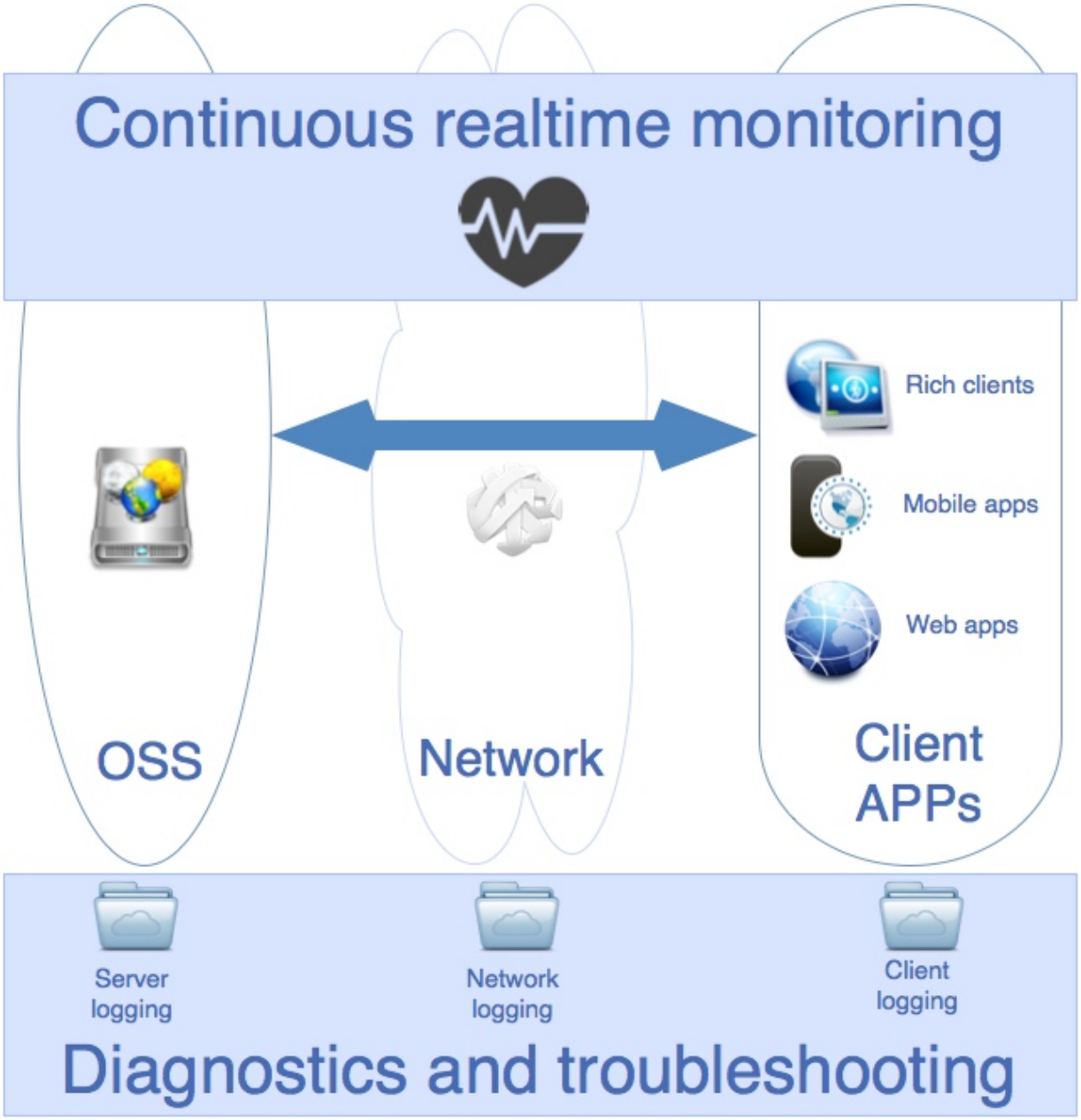
概述

本文主要描述如何使用OSS监控服务、日志记录功能以及其他第三方工具来监控、诊断和排查应用业务使用OSS存储服务时遇到的相关问题，帮助您达到如下目标：

· 实时监控OSS存储服务的运行状况和性能，并及时报警通知。
· 获取有效的方法和工具来定位问题。
· 根据相关问题的处理和操作指南，快速解决与OSS相关的问题。

本文包括如下内容：

- 服务监控，介绍如何使用OSS监控服务持续监控OSS存储服务的运行状况和性能。
- 跟踪诊断，介绍如何使用OSS监控服务和logging记录功能诊断问题；另外，还介绍如何关联各种日志文件中的相关信息进行跟踪诊断。
- 故障排除，提供常见的问题场景和故障排除方法。

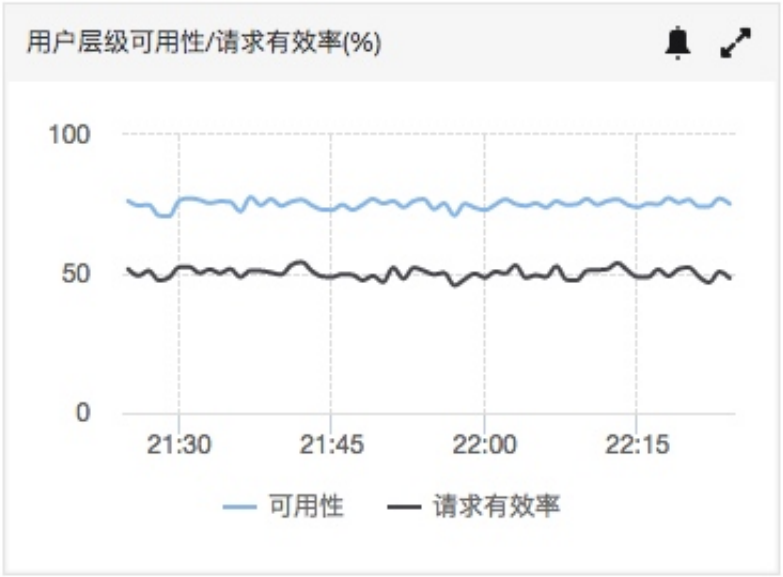


服务监控

监视总体运行状况

- 可用性和有效请求率

  可用性和有效请求率是有关系统稳定性和用户是否正确使用系统的最重要指标，指标小于100%说明某些请求失败。



可能因为一些系统优化因素出现暂时性的低于100%，比如为了负载均衡而出现分区迁移，此时OSS的SDK能够提供相关的重试机制无缝处理这类间歇性的失败情况，使得业务端无感知。

另外，对于有效请求率低于100%的情况，用户需要根据自己的使用情况进行分析，可以通过请求分布统计或者请求状态详情确定错误请求的具体类型，跟踪诊断确定原因，并故障排除。当然，对于一些业务场景，出现有效请求率低于100%是符合预期的，比如，用户需要先check访问的object是否存在，然后根据object的存在性采取一定的操作，如果object不存在，检测object存在性的读取请求必然会收到404错误（资源不存在错误），那么必然会导致该指标项低于100%。

对于系统可用性指标要求较高的业务，可以设置其报警规则，当低于预期阈值时，进行报警。

- 总请求数和有效请求数

  该指标从总访问量角度来反应系统运行状态。当有效请求数不等于总请求数时表明某些请求失败。

  用户可以关注总请求数或者有效请求数的波动状况，特别是突然上升或者下降的情况，需要进行跟进调查，可以通过设置报警规则进行及时通知，对于存在周期性状况的业务，可以设置为环比报警规则（环比报警方式即将上线），详见报警服务使用指南。

- 请求状态分布统计

  当可用性或者有效请求率低于100%（或者有效请求数不等于总请求数时），可以通过查看请求状态分布统计快速确定请求的错误类型，该统计监控指标的详细介绍参见OSS监控指标参考手册。

用户层级请求状态分布

监控项	统计值	百分比
服务端错误请求数	1246960次	5.56%
网络错误请求数	1232826次	5.49%
授权错误请求数	1246098次	5.55%
资源不存在错误请求数	1252403次	5.58%
客户端超时错误请求数	2485623次	11.08%
客户端其他错误请求数	12472186次	55.64%
成功请求数	1246815次	5.56%
重定向请求数	1232157次	5.49%
总计	22415068次	100%

监视请求状态详情

请求状态详情在请求状态分布统计的基础上进一步细化请求监控状态，可以更加深入、具体地监视某类请求。

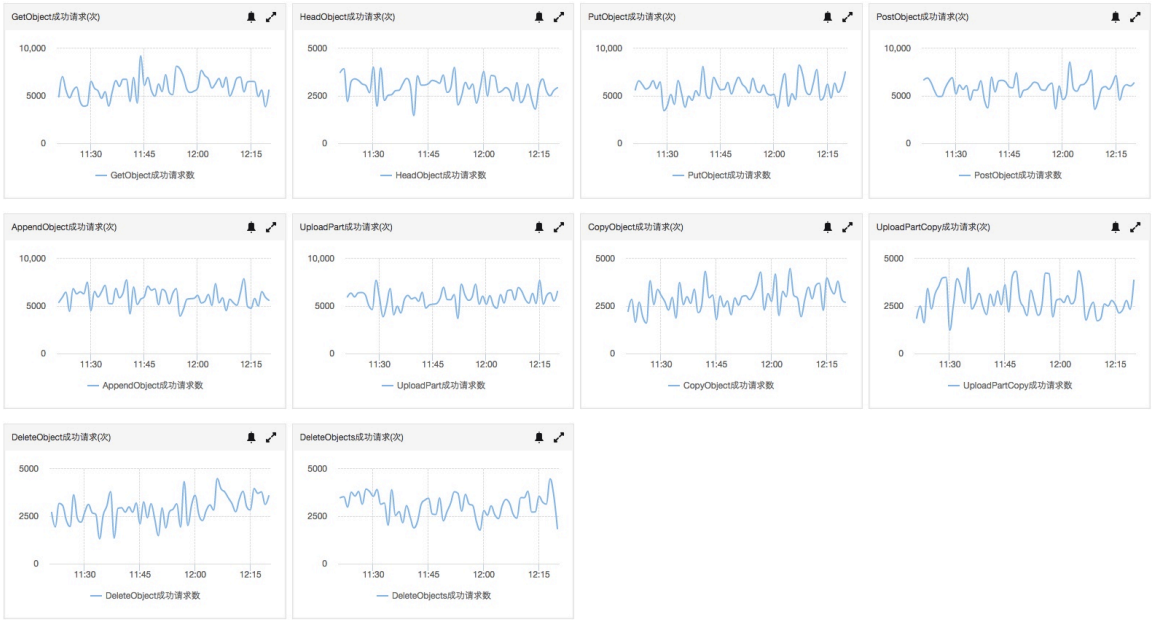
监视性能

监控服务提供了以下几个监控项用来监控性能相关的指标。

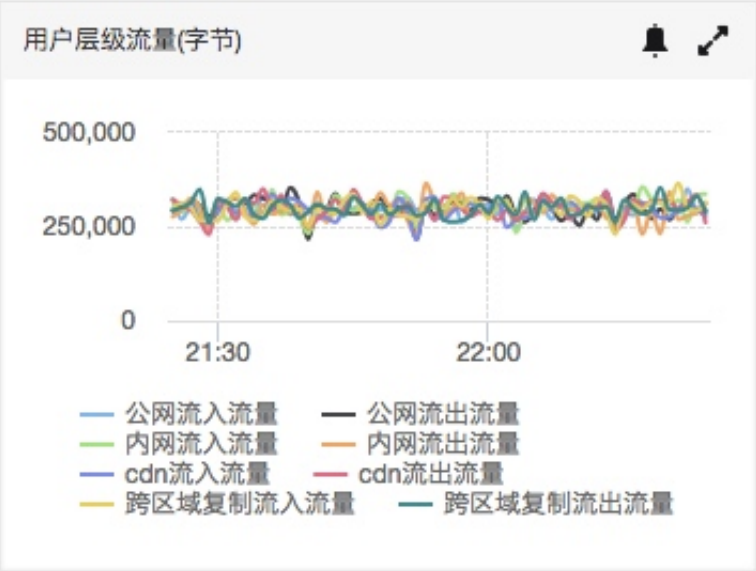
- **平均延时，包括E2E平均延时和服务器平均延时**

- **最大延时，包括E2E最大延时和服务器最大延时**

- **成功请求操作分类**



- 流量



以上各个监控项（除流量）都分别从API操作类型进行分类监控：

- GetObject
- HeadObject
- PutObject
- PostObject
- AppendObject
- UploadPart
- UploadPartCopy

延时指标显示API操作类型处理请求所需的平均和最大时间。其中E2E延时是端到端延迟指标，除了包括处理请求所需的时间外，还包括读取请求和发送响应所需的时间以及请求在网络上传输的延时；而服务器延时只是请求在服务器端被处理的时间，不包括与客户端通信的网络延时。所以当出现E2E延时突然升高的情况下，如果服务器延时并没有很大的变化，那么可以判定是网络的不稳定因素造成的性能问题，排除OSS系统内部故障。

成功请求操作分类除了以上提到的API之外，还提供以下两个API操作类型请求数量的监控：

- DeleteObject
- DeleteObjects

流量指标从用户或者具体的Bucket层级的总体情况进行监控，关注公网、内网、cdn回源以及跨域复制等场景中的网络资源占用状况。

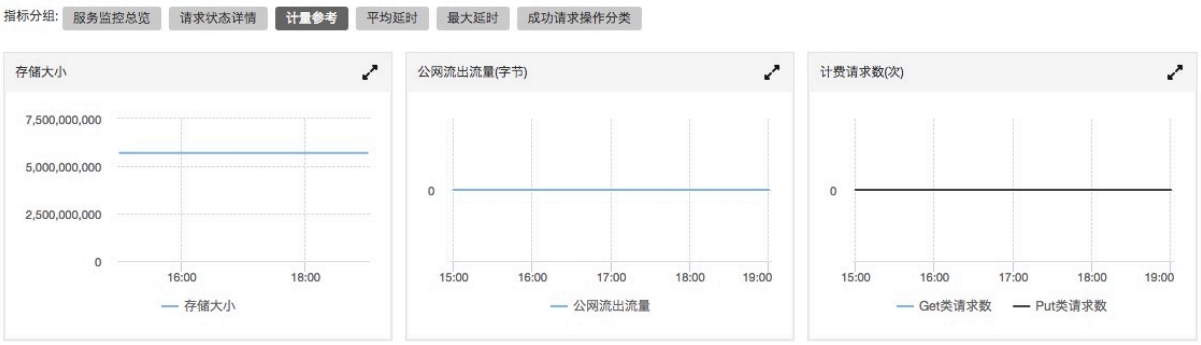
对于性能类指标项，需要重点关注其突发的异常变化，例如，平均延时突然出现尖峰，或者长时间处于高出正常请求延时的基线上方等等。用户可以通过对性能指标设置对应的报警规则，对

于低于或者超过阈值时及时通知到相关人员。对于有业务周期性的业务，可以设置环比报警规则(环比一周、环比一天或者环比一个小时，环比报警方式即将上线)。

监视计量

撰写本文档时，OSS监控服务只支持监控存储大小、公网流出流量、Put类请求数和Get类请求数（不包括跨区域复制流出流量和cdn流出流量），而且不支持对计量数据的报警设置和OpenAPI的读取。

OSS监控服务对Bucket层级的计量监控数据进行小时级别的粒度采集，可以在具体的Bucket监控视图中查看其持续的监控趋势图。用户可以根据监控视图分析其业务的存储服务资源使用趋势并且进行成本的预估等。如下所示：



OSS监控服务还提供了用户层级和Bucket层级两个不同维度的当月消费的资源数统计。即从当月1号开始到目前的账户或者某个Bucket所消耗的OSS资源总量，每小时更新，帮助用户实时了解本月使用资源的情况并计算消费。

OSS的消费计算请参考计量项和计费项。

> 说明:
> 监控服务中提供的计量数据是尽最大可能推送的，可能会与实际消费账单中产生差异，请以费用中心的数据为准。

跟踪诊断

问题诊断

- 诊断性能

  对于应用程序的性能判断多带有主观因素，用户需要根据具体的业务场景确定满足业务需求的基准线，来判定性能问题。另外，从客户端发起的请求，能引起性能问题的因素贯穿整个请求链路。比如OSS存储服务load过大、客户端tcp配置问题或网络基础结构中存在流量瓶颈等。

  因此诊断性能问题首先需要设置合理的基准线，然后通过监控服务提供的性能指标确定性能问题可能的根源位置，然后根据日志查到详细的信息以便进一步诊断并且排除故障。

  在下文故障排除中例举了常见的性能问题和排查措施，可以参考。

- 诊断错误

  客户端应用程序会在请求发生错误时接收到服务端返回的相关错误信息，监控服务也会记录并显示各种错误类型请求的计数和占比。用户也可以通过检查服务器端日志、客户端日志和网络日志来获取相关单个请求的详细信息。通常，响应中返回的HTTP状态代码和OSS错误码以及OSS错误信息都指示请求失败的原因。

  相关错误响应信息请参考 OSS错误响应。

- 使用Logging功能

  OSS存储服务为用户的请求提供服务端日志记录功能，能帮助用户记录端到端的详细请求日志跟踪。

  有关如何开启并使用logging功能，请参考设置日志。

  有关日志服务的命名规则以及记录格式等详细信息，请参见设置访问日志记录。

- 使用网络日志记录工具

  在许多情况下，通过Logging功能记录的存储日志和客户端应用程序的日志数据已足以诊断问题，但在某些情况下，可能需要更详细的信息，这时需要使用网络日志记录工具。

  捕获客户端和服务器之间的流量，可以更详细地获取客户端和服务器之间交换的数据以及底层网络状况的详细信息，帮助问题的调查。例如，在某些情况下，用户请求可能会报告一个错误，而服务器端日志中却看不到任何该请求的访问情况，这时就可以使用OSS的日志服务功能记录的日志来调查该问题的原因是否出在客户端上，或者使用网络监视工具来调查网络问题。

  最常用的网络日志分析工具之一是Wireshark。该免费的协议分析器运行在数据包级别，能够查看各种网络协议的详细数据包信息，从而可以排查丢失的数据包和连接问题。

  更详细的WireShark操作请参见WireShark用户指南。

E2E跟踪诊断

请求从客户端应用进程发起，通过网络环境，进入OSS服务端被处理，然后响应从服务端回到网络环境，被客户端接收。整个过程，是一个端到端的跟踪过程。关联客户端应用程序日志、网络跟踪日志和服务端日志，有助于排查问题的详细信息根源，发现潜在的问题。

在OSS存储服务中，提供了RequestID作为关联各处日志信息的标识符。另外，通过日志的时间戳，不仅可以迅速查找和定位日志范围，还能够了解在请求发生时间点范围内，客户端应用、网络或者服务系统发生的其他事件，有利于问题的分析和调查。

- RequestID

  OSS服务会为接收的每个请求分配唯一的服务器请求ID，即RequestID。在不同的日志中，RequestID位于不同的字段中:

  - 在OSS logging功能记录的服务端日志记录中，RequestID出现在“Request ID”列中。
  - 在网络跟踪（如WireShark捕获的数据流）中，RequestID作为x-oss-request-id标头值出现在响应消息中。
  - 在客户端应用中，需要客户端code实现的时候将请求的RequestID自行打印到客户端日志中。在攥写本文的时候，最新版本的JAVA SDK已经支持打印正常请求的RequestID信息，可以通过各个API接口返回的Result结果的getRequestId这个方法获取。OSS的各个版本SDK都支持打印出异常请求的RequestID，可以通过调用OSSException的getRequestId方法获取。

- 时间戳

  使用时间戳来查找相关日志项，需要注意客户端和服务器之间可能存在的时间偏差。在客户端上基于时间戳搜索logging功能记录的服务器端日志条目时，应加/减15分钟。

故障排除

性能相关常见问题

- 平均E2E延时高，而平均服务端延时低

  前面介绍了平均E2E延时与平均服务器延时的区别。所以产生高E2E延时、低服务器延时可能的原因有两个：

  - 客户端应用程序响应慢。
  - 网络原因导致。

  客户端应用程序响应速度慢的可能原因包括：

  - 可用连接数或可用线程数有限。
    - 对于可用连接数问题，可以使用相关命令确定系统是否存在大量TIME_WAIT状态的连接。如果是，可以通过调整内核参数解决。
    - 对于可用线程数有限，可以先查看客户端CPU、内存、网络等资源是否已经存在瓶颈，如果没有，适当调大并发线程数。
    - 如果还不能解决问题，那么就需要通过优化客户端代码，比如，使用异步访问方式等。也可以使用性能分析功能分析客户端应用程序热点，然后具体优化。
  - CPU、内存或网络带宽等资源不足。
    - 对于这类问题，需要先使用相关系统的资源监控查看客户端具体的资源瓶颈在哪里，然后通过优化代码使其对资源的使用更为合理，或者扩容客户端资源（使用更多的内核或者内存）。

  网络延迟的可能原因是：

  通常，因网络导致的端到端高延迟是由暂时状况导致的。可以使用Wireshark调查临时和持久网络问题，例如数据包丢失问题。

- 平均E2E延时低，平均服务端延时低，但客户端请求延时高

  客户端出现请求延时高的情况，最可能的原因是请求还未达到服务端就出现了延时。所以应该调查来自客户端的请求为什么未到达服务器。

  对于客户端延迟发送请求，可能的客户端的原因有两个：

  - 可用连接数或可用线程数有限，在上面章节已经描述过解决方案。
  - 客户端请求出现多次重试，如果遇到这种情况，需要根据重试信息具体调查重试的原因再解决。可以通过下面方式确定客户端是否出现重试：
    - 检查客户端日志。详细日志记录会指示重试已发生过。以OSS的JAVA SDK为例，可以搜索如下日志提示，warn或者info的级别。如果存在该日志，说明可能出现了重试。

      ```
      [Server]Unable to execute HTTP request:
        或者
      ```

```
[Client]Unable to execute HTTP request:
```

■ 如果客户端的日志级别为debug，以OSS的JAVA SDK为例，可以搜索如下日志，如果存在，那么肯定出现过重试。

```
Retrying on
```

如果客户端没有问题，则应调查潜在的网络问题，例如数据包丢失。可以使用工具（如Wireshark）调查网络问题。

· 平均服务端延时高

对于下载或者上传出现服务端高延时的情况，可能的原因有2个:

- 大量客户端频繁访问同一个小Object。

  这种情况，可以通过查看logging功能记录的服务端日志信息来确定是否在一段时间内，某个或某组小Object被频繁访问。

  对于下载场景，建议用户为该Bucket开通CDN服务，利用CDN来提升性能，并且可以节约流量费用；对于上传场景，用户可以考虑在不影响业务需求的情况下，收回Object或Bucket的写访问权限。

- 系统内部因素。

  对于系统内部问题或者不能通过优化方式解决的问题，请提供客户端日志或者Loggging功能记录的日志信息中的RequestID，联系系统工作人员协助解决。

服务端错误问题

对于服务端错误的增加，可以分为两个场景考虑:

· 暂时性的增加

对于这一类问题，用户需要调整客户端程序中的重试策略，采用合理的退让机制，例如指数退避。这样不仅可以有效避免因为优化或者升级等系统操作（如为了系统负载均衡进行分区迁移等）暂时导致的服务不可用问题，还可以避开业务峰值的压力。

· 永久性的增加

如果服务端错误持续在一个较高的水平，那么请提供客户端日志或者Logging功能记录的RequestID，联系后台工作人员协助调查。

网络错误问题

网络错误是指服务端正在处理请求时，连接在非服务器端断开而来不及返回HTTP请求头的情况。此时系统会记录该请求的HTTP状态码为499。以下几种情况会导致服务器记录请求的状态码变为499:

- 服务器在收到读写请求处理之前，会检查连接是否可用，不可用则为499.
- 服务器正在处理请求时，客户端提前关闭了连接，此时请求被记录为499.

总之，在请求过程中，客户端主动关闭请求或者客户端网络断掉都会产生网络错误。对于客户端主动关闭请求的情况，需要调查客户端中的代码，了解客户端断开与存储服务连接的原因和时间。对于客户端网络断掉的情况，用户可以使用工具（如Wireshark）调查网络连接问题。

客户端错误问题

- 客户端授权错误请求增加

  当监控中的客户端授权错误请求数增加，或者客户端程序接收到大量的403请求错误，那么最常见的可能原因有如下几个:

  - 用户访问的Bucket域名不正确。
    - 如果用户直接用三级域名或者二级域名访问，那么可能的原因就是用户的Bucket并不属于该域名所指示的region内，比如，用户创建的Bucket的地域为杭州，但是访问的域名却为Bucket.oss-cn-shanghai.aliyuncs.com。这时需要确认Bucket的所属区域，然后更正域名信息。
    - 如果用户开启了cdn加速服务，那么可能的原因是cdn绑定的回源域名错了，请检查cdn回源域名是否为用户Bucket的三级域名。
  - 如果用户使用javascript客户端遇到403错误，可能的原因就是CORS（跨域资源共享）的设置问题，因为Web浏览器实施了“同源策略”的安全限制。用户需要先检查所属Bucket的CORS设置是否正确，并进行相应的更正。设置CORS参考跨域资源共享。
  - 访问控制问题，可以分为下面四种:
    - 用户使用主AK访问，那么用户需要检查是否AK设置出错，使用了无效AK。
    - 用户使用RAM子账号访问，那么用户需要确定RAM子账号是否使用了正确的子AK，或者对应子账号的相关操作是否已经授权。
    - 用户使用STS临时Token访问，那么用户需要确认一下这个临时Token是否已经过期。如果过期，需要重新申请。
    - 如果Bucket或者Object设置了访问控制，这个时候需要查看用户所访问的Bucket或者Object是否支持相关的操作。
  - 授权第三方下载，即用户使用签名URL进行OSS资源访问，如果之前访问正常而突然遇到403错误，最大可能的原因是URL已经过期。
  - RAM子账号使用OSS周边工具的情况也会出现403错误。这类周边工具如ossftp、ossbrowser、OSS控制台客户端等，在填写相关的AK信息登入时就抛出错误，此时如

果您的AK是正确填写的，那么您需要查看使用的AK是否为子账号AK，该子账号是否有GetService等操作的授权等。

- 客户端资源不存在错误请求增加

  客户端收到404错误说明用户试图访问不存在的资源信息。当看到监控服务上资源不存在错误请求增加，那么最大可能是以下问题导致的:

  - 用户的业务使用方式，比如，用户需要先检查Object是否存在来进行下一步动作，这时他会调用doesObjectExist（以JAVA SDK为例）方法，如果Object不存在，在客户端则收到false值，但是这时在服务器端实际上会产生一个404的请求信息。所以，这种业务场景下，出现404是正常的业务行为。
  - 客户端或其他进程以前删除了该对象。这种情况可以通过搜索logging功能记录的服务端日志信息中的相关对象操作即可确认。
  - 网络故障引起丢包重试。举个例子，客户端发起一个删除操作删除某个Object，此时请求达到服务端，执行删除成功，但是响应在网络环境中丢包，然后客户端发起重试，第二次的删除操作可能就会遇到404错误。这种由于网络问题引起的404错误可以通过客户端日志和服务端日志确定:

    ■ 查看客户端应用日志是否出现重试请求。

    ■ 查看服务端日志是否对该Object有两次删除操作，前一次的删除操作HTTP Status为2xx。

- 有效请求率低且客户端其他错误请求数高

  有效请求率为请求返回的HTTP状态码为2xx/3xx的请求数占总请求的比例。状态码为4XX和5XX范围内的请求将计为失败并降低该比例。客户端其他错误请求是指除服务端错误请求（5xx）、网络错误请求（499）、客户端授权错误请求（403）、客户端资源不存在错误请求（404）和客户端超时错误请求（408或者OSS错误码为RequestTimeout的400请求）这些错误请求之外的请求。

  可以通过查看Logging功能记录的服务端日志确定这些错误的具体类型，可以在OSS错误响应上找到存储服务返回的常见错误码的列表，然后检查客户端代码中查找具体原因进行解决。

存储容量异常增加

存储容量异常增加，如果不是上载类请求量增多，一般常见的原因应该是清理操作出现了问题，可以根据下面两个方面进行调查:

- 客户端应用使用特定的进程定期清理来释放空间。针对这种请求的调查步骤是:
  1. 查看有效请求率指标是否下降，因为失败的删除请求会导致清理操作没能按预期完成。
  2. 定位请求有效率降低的具体原因，查看具体是什么错误类型的请求导致。然后还可以结合具体的客户端日志定位更详细的错误信息（例如，用于释放空间的STS临时Token已过期）。
- 客户端通过设置LifeCycle来清理空间：针对这种请求，需要通过控制台或者API接口查看目前Bucket的LifeCycle是否为之前设置的预期值。如果不是，可以直接更正目前配置；进一步的调查可以通过Logging功能记录的服务端日志记录查询以前修改的具体信息。如果LifeCycle是正常的，但是却没有生效，请联系OSS系统管理员协助调查。

其他存储服务问题

如果前面的故障排除章节未包括您遇到的存储服务问题，则可以采用以下方法来诊断和排查您的问题:

1. 查看OSS监控服务，了解与预期的基准行为相比是否存在任何更改。监控视图可能能够确定此问题是暂时的还是永久性的，并可确定此问题影响哪些存储操作。
2. 使用监控信息来帮助您搜索Logging功能记录的服务端日志数据，获取相关时间点发生的任何错误信息。此信息可能会帮助您排查和解决该问题。
3. 如果服务器端日志中的信息不足以成功排查此问题，则可以使用客户端日志来调查客户端应用程序，或者配合使用网络工具（Wireshark等）调查您的网络问题。

20 云端数据处理

图片处理

图片处理（Image Processing，简称IMG）是阿里云OSS对外提供的海量、安全、低成本、高可靠的图片处理服务。您可以将原始图片上传保存在OSS上，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。IMG提供了图片处理功能，还提供了图片水印、管道、图片样式等操作。图片处理服务提供图片处理接口，图片上传请使用OSS上传接口。基于IMG，您可以搭建出跟图片相关的服务。

图片服务提供以下功能：

- 图片缩放、裁剪、旋转
- 图片添加图片、文字、图文混合水印
- 图片格式转换
- 自定义图片处理样式
- 通过管道顺序调用多种图片处理功能
- 获取图片信息

更多功能及详细介绍请参见图片处理文档。

媒体处理

媒体处理是为多媒体数据提供的转码计算服务。它以经济、易用、弹性和高可扩展的音视频转换方法，帮助您将存储于OSS的音视频转码成适合在PC、TV以及移动终端上播放的格式。

媒体处理基于阿里云云计算服务构建，它改变了以往进行转码时需要购买、搭建、管理转码软硬件的高昂投入以及配置优化、转码参数适配等复杂性问题。同时，借助云计算服务的弹性伸缩特性，可以按需提供转码能力，从而最大限度的满足业务转码需求、避免资源浪费。

媒体处理功能包括Web管理控制台、服务API和软件开发工具包。您可以通过它们使用和管理媒体处理服务，也可以将媒体处理功能集成到您自己的应用和服务中。

媒体处理功能包括：

- 转码
- 管道
- 截图
- 媒体信息
- 水印
- 预置模版

- 自定义模版
- 剪辑输出
- 分辨率按比例缩放
- M3U8输出自定义切片时长
- 音视频抽取
- 视频画面旋转
- 视频转GIF

更多功能及详细介绍请参见媒体处理文档。