对象存储 OSS

开发人员指南

MORE THAN JUST CLOUD | Alibaba Cloud

开发人员指南

OSS 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。
- 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。对象元信息是一个键值对，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。

根据不同的上传方式，对象的大小限制是不一样的。分片上传 最大支持 48.8TB 的对象大小，其他的上传方式最大支持 5GB。

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，对象信息不可变更。重复上传同名的对象会覆盖之前的对象，因此，OSS 不支持修改文件的部分内容等操作。

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“/”或者“\”字符开头。

注意：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（区域）

Region 表示 OSS 的数据中心所在的区域，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。详细请查看 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，并且数据的三份副本已经写成功。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。

强一致性方便了用户架构设计，可以使用跟传统存储设备同样的逻辑来使用OSS，修改立即可见，无需考虑最终一致性带来的各种问题。

OSS与文件系统的对比

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。用户可以根据 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。

文件系统是一种典型的树状索引结构，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。因此文件系统可以很轻易的支持文件夹的操作，比如重命名目录、删除目录、移动目录等，因为这些操作仅仅只是针对目录节点的操作。这种组织结构也决定了文件系统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。

对于 OSS 来说，可以通过一些操作来模拟类似的功能，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。

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。而文件系统的文件支持修改，比如修改指定偏移位置的内容、截断文件尾部等，这些特点也使得文件系统拥有广泛的适用性。但另外一方面，OSS 能支持海量的用户并发访问，而文件系统会受限于单个设备的性能。

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
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

OSS 术语表

英文	中文
Bucket	存储空间
Object	对象或者文件
Endpoint	OSS 访问域名
Region	区域或者数据中心
AccessKey	AccessKeyId 和 AccessKeySecret 的统称，访问密钥
Put Object	简单上传
Post Object	表单上传
Multipart Upload	分片上传
Append Object	追加上传
Get Object	简单下载
Callback	回调
Object Meta	文件元信息。用来描述文件信息，例如长度，类型等
Data	文件数据
Key	文件名
ACL (Access Control List)	存储空间或者文件的权限

注意：如果没有特殊说明，本文中出现和术语表中相同的英文和中文，表达的是相同的意思。有时候为了表述方便会混合使用。

访问与控制

访问域名

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针对具体Bucket的三级域名，由BucketName和Endpoint构成：BucketName.Endpoint。其中Endpoint根据Bucket所在Region（数据中心）的不同以及内外网访问方式的不同会有所区分。

外网Endpoint规则

这里的外网指的是互联网。

```
Region + .aliyuncs.com
```

内网Endpoint规则

这里的内网指的是阿里云内网网络。

```
Region + -internal + .aliyuncs.com
```

OSS开通Region和Endpoint对照表

请参考访问域名和数据中心。

OSS SDK 域名设置

OSS SDK中帮助用户对于每一个操作拼接了访问域名。但用户在对不同区域的Bucket进行操作的时候需要设置不同的Endpoint。

以Java SDK为例，准备对华东1节点的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 = "http://oss-cn-hangzhou.aliyuncs.com";
OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

在ECS中使用内网Endpoint访问OSS

在同一个Region的ECS和OSS之间可以通过内网地址来访问，不同Region的ECS和OSS之间则无法通过内网访问。

例如用户购买了北京的ECS，其OSS有两个Bucket：

- 其中一个Bucket叫 beijingres，Region为华北2，那么可以使用 beijingres.oss-cn-beijing-internal.aliyuncs.com 去访问 beijingres 的资源。
- 另外一个Bucket叫 qingdaores, Region为华北 1，那么从北京的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。

上面JAVA SDK的例子中使用了该Bucket的外网地址进行访问，如果要通过内网地址进行访问，需要对Endpoint进行简单修改：

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

访问OSS

OSS访问的URL

OSS是基于HTTP API的对象存储，您在进行所有的操作的时候都需要指定访问的OSS资源，这个资源可以是Bucket，也可以是Object。在访问的时候，是以URL的形式来表示OSS的资源的。OSS的URL构成如下：

```
<Schema>://<Bucket>.<Endpoint>/<Object> 三级域名访问方式

Schema：值为HTTP或者为HTTPS
Bucket：用户的OSS存储空间
Endpoint：用户的Bucket所在数据中心的访问域名
Object：用户上传在OSS上的文件
```

注意:

- 这里的Endpoint需要和Bucket所在的数据中心(Region)一致。也就是说：如果Bucket创建在杭州，就需要使用杭州的Endpoint，不能使用其他区域的Endpoint。
- ECS访问OSS可以使用同区域OSS的内网Endpoint。Region和对应Endpoint的列表可以参考 访问域名。

假如用户使用HTTPS请求杭州的OSS，Bucket名字叫oss-sample, Object名字叫mytest/oss-test-object，那么用户访问的三级域名为：

```
https://oss-sample.oss-cn-hangzhou.aliyuncs.com/mytest/oss-test-object
```

用户可以直接将Object的URL链接放入HTML中使用，如下所示：

```
<img src="https://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png" />
```

OSS访问的安全性

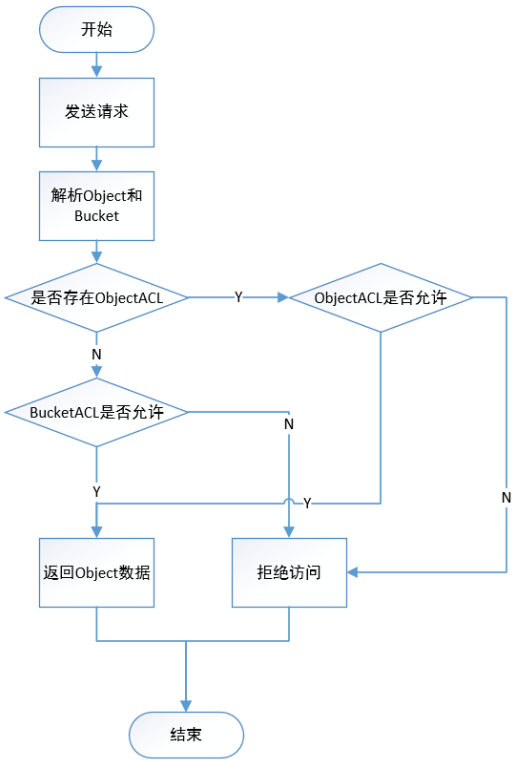
对OSS的HTTP请求可以根据是否携带身份验证信息分为两种请求。一种是带身份验证的请求，一种是不带身份验证的匿名请求。带身份验证的请求表示的是如下两种情况：

- 请求头部中带Authorization，格式为OSS + AccessKeyId + 签名字符串。
- 请求的URL中带OSS AccessKeyId和Signature字段。

OSS访问验证流程

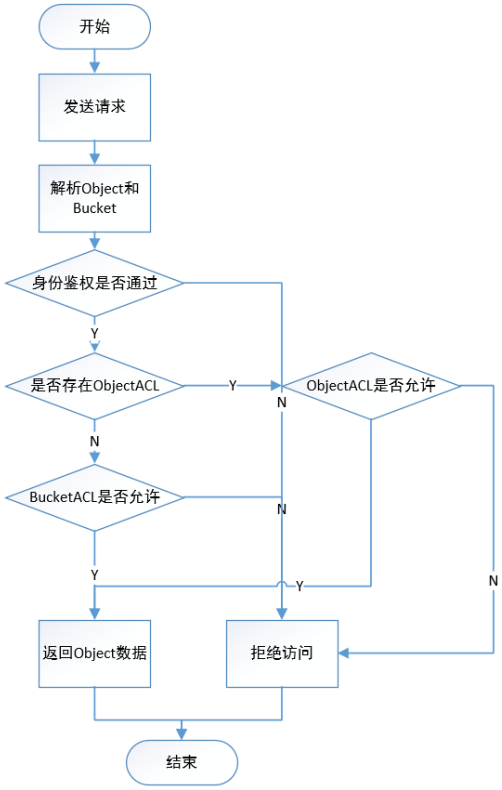
匿名请求访问流程

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。



带身份验证请求访问流程

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是否允许匿名用户访问。
        - 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是否允许匿名用户访问。
    - 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。



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。
- 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AK信息作为参数输入即可。
- 根据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。具体的签名算法可以参考API手册中的 在Header中包含签名 和 在URL中包含签名。

关于AccessKey相关的解释及更详细的身份验证的操作请参见 访问控制。

绑定自定义域名（CNAME）

您可以将自定义的域名访问绑定在属于自己的Bucket上面，即CNAME。这个操作必须通过OSS管理控制台来实现。按照中国《互联网管理条例》的要求，所有需要开通这项功能的用户，必须提供工信部备案号，域名持有者身份证等有效资料，经由阿里云审批通过后才可以使用。在开通CNAME功能后，OSS将自动处理对该域名的访问请求。

CNAME应用场景例子：

- 用户A拥有一个域名为abc.com的网站，网站的网页中的链接为http://img.abc.com/logo.png。
- 用户A此时需要将对img.abc.com图片的请求迁移到OSS上，并且不想修改任何网页的代码，也就是

对外链接不变，CNAME功能特别适合这种场景。

- 用户A通过OSS控制台，提交将img.abc.com这个自定义的域名绑定在abc-img上的申请，并提供相应的材料。
- 通过阿里云审核后，OSS后台会将img.abc.com做一个映射到abc-img（此处会做权限验证）。
- 用户A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。这样所有对用户img.abc.com域名的访问都将被转发到OSS的abc-img.oss-cn-hangzhou.aliyuncs.com。
- http://img.abc.com/logo.png请求到达OSS后，OSS找到img.abc.com和abc-img的映射，实际转换变成访问abc-img这个Bucket。这样一次对http://img.abc.com/logo.png的访问，经过OSS后，实际上访问的是http://abc-img.oss-cn-hangzhou.aliyuncs.com/logo.png。

	未做CNAME绑定前	做CNAME绑定后
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 的 logo.png。

使用CNAME注意:

1. 首先需要通过了阿里云的审批。
2. 用户在OSS上创建了Bucket，并且上传了相应的文件。
3. 在控制台上开通了CNAME，将用户自定义的域名映射到Bucket。
4. 用户在域名服务提供商那里将自定义的域名解析到Bucket.Endpoint这个域名。

控制台操作流程请参见域名管理。

OSS访问域名

OSS开通Region和Endpoint对照表

经典网络情况下Endpoint各区域的内外网设置如下：

Region中文名称	Region英文表示	外网Endpoint	ECS访问的内网Endpoint
华东 1（杭州）	oss-cn-hangzhou	oss-cn-hangzhou.aliyuncs.com	oss-cn-hangzhou-internal.aliyuncs.com
华东 2（上海）	oss-cn-shanghai	oss-cn-shanghai.aliyuncs.com	oss-cn-shanghai-internal.aliyuncs.com

华北 1（青岛）	oss-cn-qingdao	oss-cn-qingdao.aliyuncs.com	oss-cn-qingdao-internal.aliyuncs.com
华北 2（北京）	oss-cn-beijing	oss-cn-beijing.aliyuncs.com	oss-cn-beijing-internal.aliyuncs.com
华北 3 (张家口)	oss-cn-zhangjiakou	oss-cn-zhangjiakou.aliyuncs.com	oss-cn-zhangjiakou-internal.aliyuncs.com
华南 1（深圳）	oss-cn-shenzhen	oss-cn-shenzhen.aliyuncs.com	oss-cn-shenzhen-internal.aliyuncs.com
香港	oss-cn-hongkong	oss-cn-hongkong.aliyuncs.com	oss-cn-hongkong-internal.aliyuncs.com
美国西部 1（硅谷）	oss-us-west-1	oss-us-west-1.aliyuncs.com	oss-us-west-1-internal.aliyuncs.com
美国东部 1（弗吉尼亚)	oss-us-east-1	oss-us-east-1.aliyuncs.com	oss-us-east-1-internal.aliyuncs.com
亚太东南 1（新加坡）	oss-ap-southeast-1	oss-ap-southeast-1.aliyuncs.com	oss-ap-southeast-1-internal.aliyuncs.com
亚太东南 2（悉尼）	oss-ap-southeast-2	oss-ap-southeast-2.aliyuncs.com	oss-ap-southeast-2-internal.aliyuncs.com
亚太东北 1（日本）	oss-ap-northeast-1	oss-ap-northeast-1.aliyuncs.com	oss-ap-northeast-1-internal.aliyuncs.com
欧洲中部 1（法兰克福）	oss-eu-central-1	oss-eu-central-1.aliyuncs.com	oss-eu-central-1-internal.aliyuncs.com
中东东部 1（迪拜）	oss-me-east-1	oss-me-east-1.aliyuncs.com	oss-me-east-1-internal.aliyuncs.com

注意:

在分享链接或者做CNAME绑定域名的时候建议使用三级域名，即Bucket + Endpoint(以上海的oss-sample Bucket为例子，三级域名为oss-sample.oss-cn-shanghai.aliyuncs.com)的形式。

新版本SDK(除了C SDK)请使用http://和https:// + Ednpoint作为初始化的参数。以上海的Endpoint为例子，为http://oss-cn-shanghai.aliyuncs.com或者https://oss-cn-shanghai.aliyuncs.com。**不用三级域名作为初始化参数**（即不使用http://bucket.oss-cn-shanghai.aliyuncs.com作为初始化参数）。

由于旧版本SDK(例如C, PHP, Python SDK)中有的是直接使用Endpoint，例如oss-cn-shanghai.aliyuncs.com。请以实际使用的SDK版本文档或者代码说明的为准。

原地址oss.aliyuncs.com 默认指向杭州节点外网地址。

原内网地址oss-internal.aliyuncs.com 默认指向杭州节点内网地址。

VPC网络下Region和Endpoint对照表

在VPC网络下的ECS访问OSS只能使用如下的Endpoint：

Region中文名称	Region英文表示	VPC网络Endpoint
华东 1 (杭州)	oss-cn-hangzhou	vpc100-oss-cn-hangzhou.aliyuncs.com
华东 2 (上海)	oss-cn-shanghai	vpc100-oss-cn-shanghai.aliyuncs.com
华北 1 (青岛)	oss-cn-qingdao	vpc100-oss-cn-qingdao.aliyuncs.com
华北 2 (北京)	oss-cn-beijing	vpc100-oss-cn-beijing.aliyuncs.com
华北 3 (张家口)	oss-cn-zhangjiakou	oss-cn-zhangjiakou-internal.aliyuncs.com
华南 1 (深圳)	oss-cn-shenzhen	vpc100-oss-cn-shenzhen.aliyuncs.com
美国西部 1 （硅谷 ）	oss-us-west-1	vpc100-oss-us-west-1.aliyuncs.com
美国东部 1 （弗吉尼亚)	oss-us-east-1	oss-us-east-1-internal.aliyuncs.com
亚太东南 1 （新加坡 ）	oss-ap-southeast-1	vpc100-oss-ap-southeast-1.aliyuncs.com
亚太东南 2 （悉尼 ）	oss-ap-southeast-2	vpc100-oss-ap-southeast-2.aliyuncs.com
亚太东北 1（日本 ）	oss-ap-northeast-1	oss-ap-northeast-1-internal.aliyuncs.com
欧洲中部 1（法兰克福 ）	oss-eu-central-1	oss-eu-central-1-internal.aliyuncs.com
中东东部 1（迪拜 ）	oss-me-east-1	oss-me-east-1-internal.aliyuncs.com

接入OSS

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。

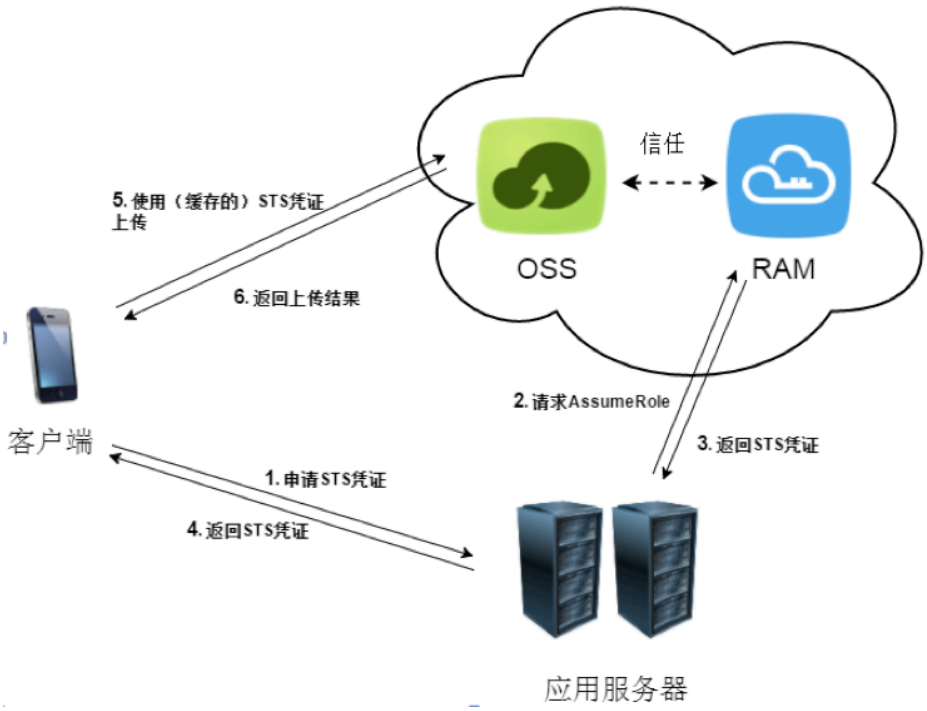
最佳实践：

- 30分钟快速搭建移动应用直传服务
- 移动端回调案例分析及代码下载）
- 轻松搞定移动端安全策略）

开发业务流程

临时凭证授权的上传

如图所示：



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。

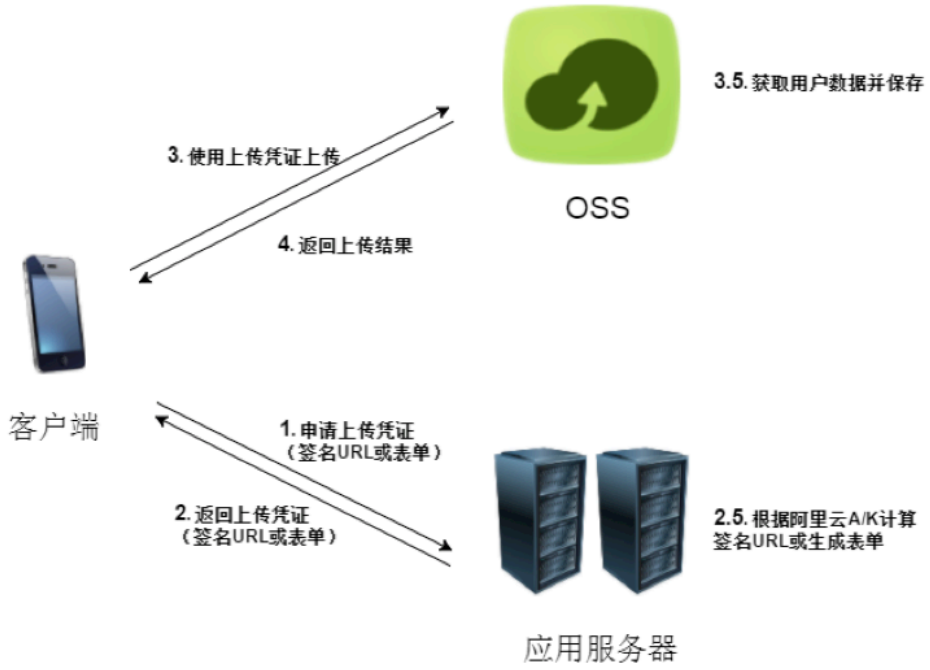
这里有几个要点：

- 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，从而可以实现不同客户端在OSS端上传的Object的完全隔离，极大的提高了安全系数。

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

如图所示：



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。

这里有几个要点：

- 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。

特别注意

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

- RAM和STS使用指南

功能使用参考

- SDK：Android SDK 文件操作
- SDK：iOS SDK 文件操作

OSS快速开始

基于控制台快速开始：

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

具体见控制台快速开始文档。

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、以及用于大文件上传的分片上传还有适用于视频监控等领域使用的追加上传。
- 同样也可以使用多种方法从OSS下载文件，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。

具体见SDK开发文档。

管理存储空间

创建存储空间

您可以选择在已有的区域创建存储空间。同时需要注意有下列限制：

- 用户的国内区域、美国、新加坡存储空间总数不能超过 10 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，名称和所处区域不能修改。
- OSS总存储容量没有限制，但单个存储空间的容量限制是 2PB。

OSS 提供 ACL（Access Control List）权限控制方法，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，默认值为私有读写。更多信息，请参见 设置存储空间读写权限。

功能使用参考

- 控制台：创建存储空间
- SDK：Java SDK-Bucket 中新建存储空间
- API：Put Bucket

除了在创建存储空间的时候能够对存储空间的 ACL 进行设置，也可以之后根据自己的业务需求对存储空间的ACL进行修改。这个操作只有该存储空间的创建者有权限执行。目前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）都可以对该存储空间中的文件进行读写操作，所有这些操作产生的费用由该存储空间的创建者承担，请慎用该权限。

public-read	公共读，私有写	只有该存储空间的拥有者或者授权对象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）可以对该存储空间中的文件进行读操作。
private	私有读写	只有资源拥有者或者授权对象可以对该存储空间内的文件进行读写操作，其他人在未经授权的情况下无法访问该存储空间内的文件。

详细解释请参见 访问权限。

功能使用参考

设置存储空间 ACL

- 控制台: 设置访问权限
- SDK：Java SDK-Bucket 中 **设置Bucket ACL**
- API：Put BucketACL

获取存储空间 ACL

- 控制台: 登录后可以在存储空间属性中查看
- SDK：Java SDK-Bucket 中 **获取Bucket ACL**
- API：Get BucketACL

查看存储空间列表

查看您创建的所有存储空间列表。

功能使用参考

- 控制台：进入控制台后默认显示您创建的存储空间列表
- API: GetService
- SDK：Java SDK-Bucket中列举Bucket

相关参考链接

- 创建Bucket

获取存储空间信息

您可以获取存储空间所属的地域，即数据中心的物理位置信息。返回的 Location 字段显示存储空间所在的地域信息，比如华东 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。

功能使用参考

- 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
- API: Get Bucket Location
- SDK: Java SDK-Bucket中**获取Bucket地址**

删除存储空间

您可以删除您创建的存储空间。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。

功能使用参考

- API: Delete Bucket
- SDK: Java SDK-Bucket中**删除Bucket**
- 控制台： 删除存储空间

上传文件

简单上传

适用场景

简单上传指的是用户使用OSS API中的Put Object方法上传单个Object，可以适用在任何一次HTTP请求交互即可完成上传的场景，比如小文件的上传。

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。具体请参考Object Meta。

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“/”或者“\”字符开头。

大文件上传

因为使用的是单次HTTP请求，Object过大会导致上传时间长。在这段时间出现网络原因造成超时或者链接断开等错误的时候，上传容易失败，可以考虑断点续传上传（分片上传）。当Object大于5GB，这种情况下只能使用断点续传上传（分片上传），具体参考断点续传上传。

上传的安全及授权

为了防止第三方往开发者的Bucket未经授权上传，OSS提供了Bucket和Object级别的访问权限控制，详细解释见访问控制。为了授权给第三方上传，OSS除了Bucket和Object级别的访问权限外，还提供了账号级别的授权，见上传安全之授权第三方。

上传后续操作

在文件上传到OSS上后，开发者可以使用上传后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。如果上传的是图片需要处理，可以使用上传图片后云端处理。如果上传的是音频或者视频文件也可以使用媒体转码。

功能使用参考：

- API：PutObject
- SDK：Java SDK-Object中PutObject
- 控制台：上传文件

最佳实践

- RAM和STS使用指南

- Web端直传实践及上传回调

相关参考链接：

- 上传后回调
- 移动端开发上传场景介绍
- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 上传安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

表单上传

适用场景

表单上传是一种适用于小文件上传的上传方式，指的是用户使用OSS API中的Post Object请求来完成Object的上传。非常适合嵌入在HTML网页中来上传Object。比较常见的应用场景是网站应用，以招聘网站为例子：

	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

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“/”或者“\”字符开头

使用表单上传的好处

- 从流程上，少了一步转发。
- 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，瓶颈在网站服务器，可能需要扩容网站服务器。采用表单上传后，上传都是直接从客户端发送到OSS。上传量过大时，压力都在OSS上

，由OSS来保障服务质量。

上传的安全及授权

为了防止第三方往开发者的Bucket未经授权上传，OSS提供了Bucket和Object级别的访问权限控制，详细解释见 访问控制。为了授权给第三方上传，这里使用的是Post Policy的机制，详细见 PostObject。

使用表单上传的基本流程

构建一个Post Policy。Policy的作用是网站开发者用来限制网站用户上传。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。
比如是这样的一个简单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
```

注意:

- 构建的表单中 "success_action_redirect" value="http://oss.aliyun.com" 表示的是上传成功后跳转的页面。可以替换。
- 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。

功能使用参考

- API：PostObject

最佳实践

- Web端直传实践
- 跨域资源共享（CORS）使用指南

相关参考链接

- 上传后回调
- 移动端开发上传场景介绍
- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 上传安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

断点续传

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，OSS提供了分片上传(Multipart Upload)来达到断点续传的效果。顾名思义，分片上传就是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。

相对于其他的上传方式，分片上传适用于以下场景：

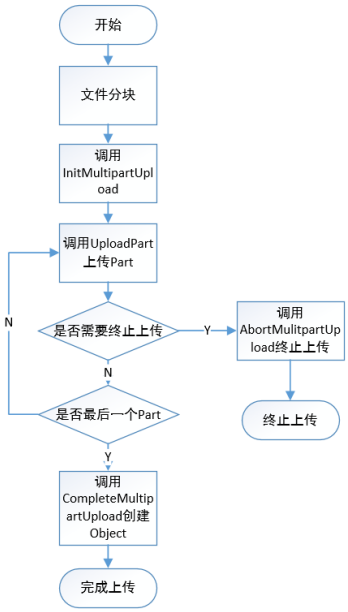
- **恶劣的网络环境**。如手机端，当出现上传失败的时候，可以对失败的Part进行独立的重试，而不需要

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。

基本流程

一般的流程如下：

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）。



在这个过程中有些要注意的地方：

- 除了最后一块Part，其他Part的大小不能小于100KB，否则会导致在调用CompleteMultipartUpload接口的时候失败。
- 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。
- 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。

断点续传

因为已经上传的Part的生命周期是永久的，因此很容易可以实现断点续传的功能。

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。通过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。

断点续传功能在移动设备和大文件上传中尤其有价值。

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，每块最小100KB（最后一块可以比100KB小），最大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“/”或者“\”字符开头

上传的安全及授权

为了防止第三方往开发者的Bucket未经授权上传，OSS提供了Bucket和Object级别的访问权限控制，详细解释见访问控制。为了授权给第三方上传，OSS除了Bucket和Object级别的访问权限外，还提供了账号级别的授权，见上传安全之授权第三方。

上传后续操作

- 在文件上传到OSS上后，开发者可以使用上传后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，可以使用图片服务进行后续处理。
- 如果上传的是音频或者视频文件，可以使用媒体转码进行后续处理。

功能使用参考：

- API：
    - MultipartUpload
    - InitiateMultipartUpload
    - UploadPart
    - UploadPartCopy
    - CompleteMultipartUpload
    - AbortMultipartUpload
    - ListMultipartUploads

    - ListParts
- SDK：Java SDK-MultipartUpload 中的分片上传

最佳实践

- RAM和STS使用指南
- Web端直传实践

相关参考链接：

- 上传后回调
- 移动端开发上传场景介绍
- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 上传安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

追加上传

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，典型比如视频监控、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。
- 需要有位置保存元数据，比如已经生成的Object列表等，然后每次请求都重复读取元数据来判断是否有新的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提供了用户通过追加上传（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。

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“/”或者“\”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

为了防止第三方往开发者的Bucket未经授权上传，OSS提供了Bucket和Object级别的访问权限控制，详细解释见访问控制。为了授权给第三方上传，OSS除了Bucket和Object级别的访问权限外，还提供了账号级别的授权，见上传安全之授权第三方。

上传后续操作

如果上传的是图片需要处理，可以使用上传图片后云端处理。如果上传的是音频或者视频文件也可以使用媒体转码。

功能使用参考

- API：Append Object
- SDK：Java SDK-追加文件示例

注意: 追加上传不支持上传回调操作。

最佳实践

- RAM和STS使用指南

相关参考链接

- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 上传安全之授权第三方

授权给第三方上传

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。

目前授权上传可以由两种实现方式：

URL签名

URL签名是授权访问的一种方式，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。具体的做法可以参考在URL中包含签名。

临时访问凭证

临时访问凭证是通过阿里云SecurityTokenService(STS)来实现授权的一种方式。其实现请参见STS Java SDK。临时访问凭证的流程如下：

1. 客户端向服务器端发起获得授权的请求。服务器端先验证客户端的合法性。如果是合法客户端，那么服务器端会使用自己的AccessKey来向STS发起一个请求授权的请求，具体可以参考访问控制。
2. 服务器端获取临时凭证之后返回给客户端。
3. 客户端使用获取的临时凭证来发起向OSS的上传请求，更详细的请求构造可以参考临时授权访问。客户端可以缓存该凭证用来上传，直到凭证失效再向服务器端请求新的凭证。

最佳实践

- RAM和STS使用指南
- Web端直传实践及上传回调

相关参考链接

- 上传后回调
- 移动端开发上传场景介绍
- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 表单上传

上传回调

适用场景

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。现在支持CallBack的API 接口有：PutObject、PostObject、CompleteMultipartUpload。

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；另外一种就是自定义的参数，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：



注意:

- 目前只有大陆地区支持上传回调功能。
- 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。

功能使用参考

- API：Callback
- SDK: iOS 上传后回调通知

最佳实践

Web端直传实践及上传回调

如何搭建一个回调应用服务器（包括示例代码下载）

相关参考链接

- 移动端直传服务
- 移动端权限管理
- 移动端开发上传场景介绍
- 上传后下载
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 上传安全之访问控制
- 上传安全之授权第三方
- 上传后对文件进行复制、删除等管理操作

下载文件

简单下载

简单下载即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。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。

当用户需要访问某个Object的时候，有两种情况：

- 该Object没有匿名读的权限，用户拥有对应的AccessKey，那么可以使用AccessKey对GET请求进行签名来访问。
- 该Object拥有匿名读的权限，那么所有的用户都可以直接使用GET请求来进行访问。

具体的Object和Bucket的访问权限控制请参考访问控制。

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。

如果需要实现断点下载请参考断点续传下载。

功能使用参考

- API：Get Object
- SDK：Java SDK-Object
- 控制台：获取文件访问地址

最佳实践

- RAM和STS使用指南

相关参考链接

- 上传文件方式
- 上传后回调
- 移动端开发下载场景介绍
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 下载安全之访问控制
- 下载安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

断点续传下载

OSS提供了从Object指定的位置开始下载的功能，这样在下载很大的Object的时候，可以分为多次下载。如果中途下载中断，重启的时候也可以从上次最后完成的地点开始继续下载。

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。如果不符合范围规范，则传送整个文件，并且不在结果中提及Content-Range。返回码为

206。

功能使用参考

- API：Get Object
- SDK：Java SDK-分段读取文件

相关参考链接

- 上传文件方式
- 上传后回调
- 移动端开发下载场景介绍
- 上传图片后云端处理
- 上传音频视频文件后云端处理
- 下载安全之访问控制
- 下载安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

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，而应该使用以下两种方法。

URL签名

OSS提供了签名下载的方法。在实现中，开发者在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：开发者的AccessKeyId。
- Expires：开发者期望URL过期的时间。
- Signature：开发者签名的字符串，具体请参考API文档签名部分。

注意：此连接需要进行URL编码。

功能使用参考

- API：Get Object
- SDK：Java SDK-使用URL签名授权访问
- 控制台：获取文件访问地址

注意: 在控制台中只有当Bucket处于私有读写权限的时候，获取的访问地址才是这种URL中签名的形式。否则则为不带签名的URL形式。

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和AccessKeySecret以及STS Token签名，去请求开发者的Object的资源。

功能使用参考

- API：临时访问凭证
- SDK：Java SDK-Object中**使用STS服务临时授权**
- 控制台：获取文件访问地址

最佳实践

- RAM和STS使用指南

相关参考链接

- 上传文件方式
- 上传后回调
- 移动端开发下载场景介绍
- 上传图片后云端处理
- 上传音频视频文件后云端处理

- 下载安全之访问控制
- 下载安全之授权第三方
- 上传后对文件进行复制，删除等管理操作

管理文件

设置对象/文件元信息

对象/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信息会在下载GetObject或者HeadObject的时候在HTTP头部中返回。

上传Object时设置Object Meta

当上传Object的时候，可以设置Object Meta。

功能使用参考：

- API：Put Object

- SDK：Java SDK-上传文档中的**设定Object的Http Header** 和 **用户自定义元信息**

分片上传（断点续传）的时候也可以设置Object Meta。

功能使用参考：

- API：InitiateMultipartUpload
- SDK：Java SDK-初始化Multipart Upload 中的 **初始化Multipart Upload**

上传Object后修改Object Meta

如果需要修改Object Meta而不修改Object本身的数据，那么应该使用拷贝Object的接口来实现这个功能，只需要将新的Meta信息（注意这个Meta必须是全量）放在HTTP头部中，然后将拷贝的源地址和目标地址都设为目标Object的地址即可。

功能使用参考：

- API：拷贝Object
- SDK：Java SDK-通过CopyObjectRequest拷贝Object

获取Object Meta

如果需要获取Object Meta而并不需要Object本身的数据，可以使用此功能。

功能使用参考：

- API：Head Object
- SDK：Java SDK-仅获取文件元信息

查看对象列表

查看对象列表即列出您在Bucket中上传的文件（Object），您可以通过OSS的接口调用一次性得到某一Bucket下最多1000个的Object列表。通过下面的四个参数，您可以完成多种拓展功能：

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，如果不设定，默认为100，MaxKeys取值不能大于1000。
Prefix	限定返回的Object key必须以Prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含Prefix。

文件夹模拟功能

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size为0的Object。对于这个Object照样可以上传下载，只是控制台会对以“/”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实现创建模拟文件夹。

您可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

- 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 “/” 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：
假如在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

1. 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。

2. 列出第二层目录A底下的目录和文件
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返回的是第一层的目录：“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。

功能使用参考

- API：Get Bucket
- SDK：Java SDK-列出存储空间中的文件

拷贝对象

拷贝对象即复制Bucket中的文件。在有些情况下，您可能需要仅仅只是将一些Object从一个Bucket复制到另外一个Bucket，不改变内容。这种情况一般的做法是将Object重新下载然后上传。但是因为数据实际上都是一样的，因此浪费了很多网络带宽。因此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，因此如果需要对Object进行重命名的话，最佳的方法就是调用OSS的CopyObject接口先将原来的数据拷贝成新的文件名，然后删除原Object。如果用户仅仅是需要修改某个Object的一些Object Meta信息，同样可以调用CopyObject的接口，将源Object地址和目标Object地址设置成相同的，这样OSS就会仅更新该Object Meta信息。Object Meta信息可以参考Object Meta。

该操作有以注意事项：

- 您需要有源Object的操作权限，否则会无法完成操作。
- 该操作不支持跨Region拷贝数据。比如：不支持将杭州Bucket里的Object拷贝到青岛。
- 该操作支持的最大Object大小为1GB。
- 该操作不能对追加上传产生的Object进行拷贝。

功能使用参考：

- API：Copy Object
- SDK：Java SDK-Object

拷贝大文件

当用户的Object很大的时候，就必须采用其他的方法来完成。和上传操作一样，OSS提供了类似断点续传上传的功能来完成大文件的拷贝。

基本操作步骤和断点续传上传内描述的基本一致，唯一需要注意的就是将UploadPart替换成UploadPartCopy。UploadPartCopy的语义和UploadPart基本一致，只是数据源从HTTP请求直接上传改成从源Object中获取。

功能使用参考：

- API：UploadPartCopy
- SDK：Java SDK-拷贝大文件

删除对象

删除对象即删除上传在Bucket中的文件（Object），OSS允许您做如下的删除动作：

- 单个删除。指定某个Object删除。
- 批量删除。批量删除一次最多可指定1000个Object。
- 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，这个时候推荐使用生命周期管理来完成。一旦设置了之后，OSS会根据规则自动删除已到期的Object，能大大减少您发送删除请求的次数，提高删除速度。

功能使用参考

- API：Delete Object和Delete Multiple Objects
- SDK：Java SDK 删除文件
- 控制台：删除文件

管理对象生命周期

OSS 提供 Object（文件）生命周期管理来为您管理Object。您可以为某个Bucket定义生命周期配置，来为该Bucket的Object定义各种规则。目前，您可以通过规则来删除相匹配的Object。每条规则都由如下几个部分组成：

- Object名称前缀
  只有匹配该前缀的Object才适用这个规则。
- 操作
  您希望对匹配的Object所执行的操作。
- 日期或天数
  您期望在特定日期或者在Object最后修改时间后多少天执行指定的操作。

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。

当前，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时，GET和HEAD该Object时，OSS在响应Header中加入x-oss-expiration头。它包含了两个键值对：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。

举例

您可以通过OSS开放接口来设置Bucket的生命周期配置。生命周期配置是由XML格式给出的，下面是一个具体的例子。

```
<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>2014-12-31T00:00:00.000Z</CreatedBeforeDate>
</Expiration>
</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>：Enabled或Disabled。OSS只会应用值为Enabled的规则。
- <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。
    - 这里CreatedBeforeDate表示的含义是：将最后修改时间早于2014-12-31T00:00:00.000Z的文件删除。晚于这个时间的Object不会被删除。
    - 这里Days表示的含义是：将相对最后修改时间10天之后的Object删除。

在这个例子中，第一条规则会删除前缀为logs/的，最后更新时间是10天前的Object。第二条规则虽然指定了删除2014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。

细节分析

- 前缀的命名规范和Object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，如果同一Bucket的两条规则，一个前缀是logs/，一个是logs/program，那么OSS会返回错误。
- 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2014-01-01T00:00:00.000Z的ISO8601格式。OSS会在当前时间超过2014-01-01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2014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 2014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。

功能使用参考

- API：Put Bucket Lifecycle

管理跨区域复制

跨区域复制（Bucket Cross-Region Replication）是跨不同OSS数据中心的Bucket自动、异步复制Object，它会将对源Bucket中的对象的改动（新建、覆盖、删除等）同步到目标Bucket。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）。

使用场景

您可能基于各种原因对Bucket配置Cross-Region Replication，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会有多份副本，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。通过跨区域数据同步，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。

使用说明

跨区域同步支持异名Bucket的同步，如果您拥有的两个Bucket分属不同区域，可以通过配置同步将源Bucket的数据实时同步到目的Bucket。现在能够支持以下特性：

- 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区域Bucket。对于2M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。

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可能会覆盖目的Bucket中同名的Object，使用中请注意。
- 由于Bucket Replication是采用异步复制，数据复制到目的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。举例来说，若 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必须分属两个区域，同区域的Bucket不能进行数据同步。
- 目前只支持中国区内的跨区域复制。

功能使用参考

- 控制台：跨区域复制

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。

通过规则的方式，对每条到OSS的Get请求的url进行匹配，然后按照特定的方式进行回源。规则最多配置5条

，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，则当Get一个不存在的文件时，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

镜像回写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ossimport2，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)，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，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，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，不要将镜像回写的配置删掉，这样可以保证切换到OSS或者CDN的流量还是能够拿到源站的数据。

使用细则

- 只有当GetObject()本应该返回404的情况下，OSS才会执行镜像回写，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/，其他与上例相同，那么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。

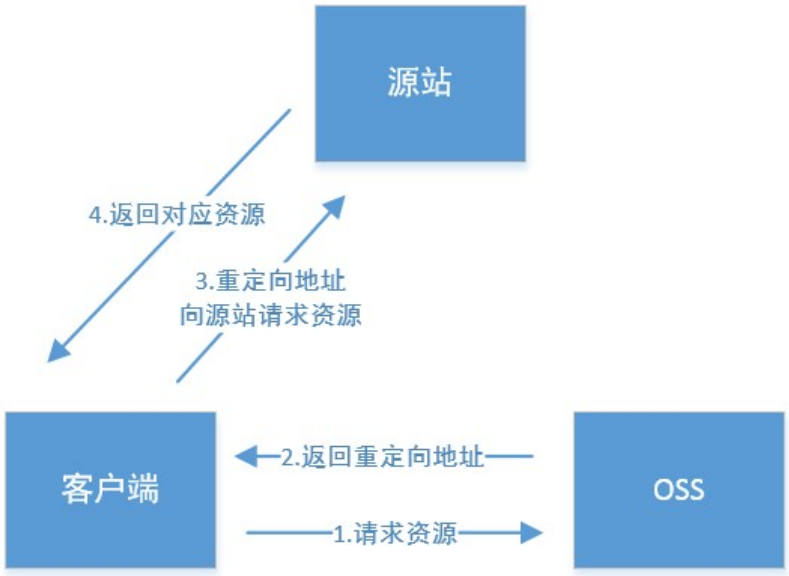
- 传给OSS的header信息以及querystring信息不会传递给源站。
- 如果源站是chunked编码返回，那么OSS返回给用户的也是chunked编码。
- OSS会将源站的以下头信息返回并存为OSS的头信息：

```
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），以上例为例，则为x-oss-tag:MIRROR http%3a%2f%2fwww.example-domain.com%2fdir1%2fimage%2fexample_object.jpg。文件回写到OSS上后，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了。
- 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将返回用户424，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：



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、或者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。

功能使用参考

- 控制台：**管理回源规则**

安全管理

发送访问OSS的请求

您可以直接使用OSS提供的RESTful API接口访问或者使用对API接口进行完整封装的SDK开发包。而每一次向

OSS的请求根据当前Bucket权限和操作不同要求用户进行身份验证或者直接匿名访问。对OSS的资源访问的分类如下：

- 按访问者的角色可分为**拥有者访问**和**第三方用户访问**。这里的拥有者指的是Bucket的Owner，也称为开发者。第三方用户是指访问Bucket里资源的用户。
- 按访问者的身份信息可分为**匿名访问**和**带签名访问**。对于OSS来说，如果请求中没有携带任何和身份相关的信息即为匿名访问。带签名访问指的是按照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。

AccessKey 类型

目前访问 OSS 使用的 AK（AccessKey）有三种类型，具体如下：

阿里云账号AccessKey

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：请谨慎直接使用阿里云账户的 AccessKey。

RAM子账号AccessKey

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AccessKey

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。

对于用户来说可以直接使用OSS提供的SDK，配合不同类型的AccessKey即可实现不同的身份验证。

权限控制

OSS对向存放在Bucket的Object访问提供了多种权限控制，主要有：

- Bucket级别权限
- Object级别权限
- 账号级别权限（RAM）
- 临时账号权限（STS）

Bucket级别权限

Bucket权限类型

OSS提供ACL（Access Control List）权限控制方法，OSS ACL提供Bucket级别的权限访问控制，Bucket目前有三种访问权限：public-read-write，public-read和private，它们的含义如下：

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。

Bucket权限设定和读取方法

功能使用参考：

- API：Put BucketACL
- SDK：Java SDK-设置Bucket ACL
- 控制台: 创建Bucket权限设置
- API：Get BucketACL
- SDK：Java SDK-获取Bucket ACL

Object级别权限

Object权限类型

OSS ACL也提供Object级别的权限访问控制。目前Object有四种访问权限：private, public-read, public-read-write, default。Put Object ACL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。

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

注意事项

- 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。举个例子，如果设置了Object的权限是public-read，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。

Object权限设定和读取方法

功能使用参考：

- API：Put Object ACL
- SDK：Java SDK-ObjectACL 中设定Object ACL
- API：Get Object ACL
- SDK：Java SDK-ObjectACL 中读取Object ACL

账号级别权限（RAM）

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。

解决方法：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。

具体实现

RAM详情，请参考RAM用户手册，里面有对如何授权，RAM账号生成以及分组权限管理等有详细的描述。对于授权中需要的Policy的配置方式可以参考本章最后一节**RAM和STS授权策略（Policy）配置**。

临时账号权限（STS）

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，也有直接访问OSS资源的可能，将这部分用户称为联盟用户。此外，用户还可以是您创建的能访问您的阿里云资源的应用程序。

对于这部分联盟用户，通过阿里云STS (Security Token Service) 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。

用户通过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。此外还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。

具体实现

STS安全令牌详情，请参考RAM用户指南中的**角色管理**。关键是调用STS服务接口AssumeRole来获取有效访问凭证即可。也可以直接使用STS SDK来调用该方法，**点击查看**。

角色管理与使用相关内容请参考RAM用户指南中的**角色管理**部分。

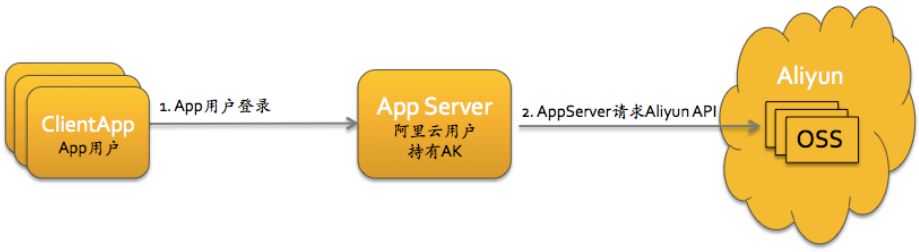
对于授权中需要的Policy的配置方式参考本章最后一节**RAM和STS授权策略（Policy）配置**。

RAM和STS应用场景实践

对于不同的应用场景，涉及到的访问身份验证方式可能存在差异。下面以几种典型的应用场景来说明访问身份验证中几种使用方式。

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。

方式一：使用AppServer来做数据中转和数据隔离

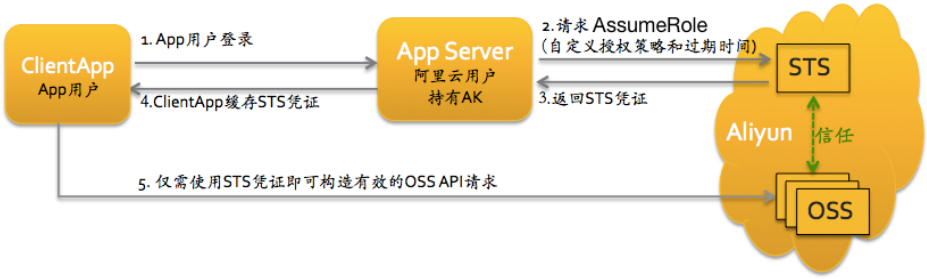


如上图所示，您需要开发一个AppServer。只有AppServer能访问云服务，ClientApp的每次读写数据都需要通过AppServer，AppServer来保证不同用户数据的隔离访问。

对于该种使用方式，使用阿里云账号或者RAM账号提供的密钥来进行签名验证访问。建议您尽量不要直接使用阿里云账号（根账号）的密钥访问OSS，避免出现安全问题。

方式二：使用STS让用户直接访问OSS

STS方案描述如下图所示：



方案的详细描述如下：

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，角色管理与使用相关内容请参考RAM用户指南中的角色管理。
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。

RAM和STS授权策略（Policy）配置

对于RAM或者STS授权中使用Policy，详细规则如下。

示例

先看下面的一个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:Delimiter": "/",
"oss:Prefix": "foo"
},
"IpAddress": {
"acs:SourceIp": "192.168.0.0"
}
}
},
```

```
{
"Action": [
"oss:PutObject",
"oss:GetObject",
"oss:DeleteObject"
],
"Resource": [
"acs:oss:*:1775305056529849:mybucket/file*"
],
"Effect": "Allow",
"Condition": {
"IpAddress": {
"acs:SourceIp": "192.168.0.0"
}
}
}
]
}
```

这是一个授权的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和Delimiter这两个Condtion是在GetBucket（ListObjects）的时候起作用的，关于这两个字段的解释详见OSS的API文档。

配置细则

Version

Version定义了Policy的版本，本文档中的配置方式，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
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
DeleteMultipartObjects	oss:DeleteObject
AbortMultipartUpload	oss:AbortMultipartUpload
ListParts	oss:ListParts
CopyObject	oss:GetObject,oss:PutObject
UploadPartCopy	oss:GetObject,oss:PutObject
AppendObject	oss:PutObject
GetObjectAcl	oss:GetObjectAcl
PutObjectAcl	oss:PutObjectAcl

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"
"oss:"
],
"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:Delimiter和oss:Prefix这两项用来在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
oss:Delimiter	用作ListObject时的delimiter	合法的delimiter值

更多示例

针对具体场景更多的授权策略配置示例，可以参考OSS授权常见问题；Policy在线图形化便捷配置工具，请点击这里。

最佳实践

- RAM和STS使用指南

设置访问日志记录

OSS提供自动保存访问日志记录功能。Bucket的拥有者可以通过OSS控制台为其所拥有的Bucket开启访问日志记录功能。当一个Bucket（源Bucket，Source Bucket）开启访问日志记录功能后，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写入用户指定的Bucket（目标Bucket，Target Bucket）。

日志记录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；YYYY, mm, DD, HH, MM和SS分别是该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。一个实际的用于存储OSS访问日志的Object名称例子如下：

```
MyLog-oss-example-2012-09-10-04-00-00-0000
```

上例中，“MyLog-”是用户指定的Object前缀；“oss-example”是源Bucket的名称；“2012-09-10-04-00-00”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	119.140.142.11	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	505B01695037C2AF032593A4	用于唯一标识该请求的UUID
LoggingFlag	true	是否开启了访问日志功能
Reserved	-	保留字段
Requester Aliyun ID	1657136103983691	请求者的阿里云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	1657136103983691	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“-”
Reserved	-	保留字段

细节分析

- 源Bucket和目标Bucket必须属于同一个用户。
- “TargetPrefix”表示存储访问日志记录的Object名字前缀，可以为空。
- 源Bucket和目标Bucket可以是同一个Bucket，也可以是不同的Bucket；用户也可以将多个源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生成的Log文件命名规则中的“UniqueString”仅仅是OSS为其生成的UUID，用于唯一标识该文件。

- 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，用户可以按照普通的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。如果你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。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。

功能使用参考

- 控制台：设置日志

设置防盗链

OSS是按使用收费的服务，为了防止您在OSS上的数据被其他人盗链，OSS支持基于HTTP header中表头字段referer的防盗链方法。您可以通过OSS管理控制台或者API的方式对一个Bucket设置referer字段的白名单和是否允许referer字段为空的请求访问。例如，对于一个名为oss-example的Bucket，设置其referer白名单为http://www.aliyun.com。则所有referer为http://www.aliyun.com的请求才能访问oss-example这个Bucket中的Object。

细节分析

- 用户只有通过URL签名或者匿名访问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。
- 一个Bucket可以支持多个referer参数，这些参数之间由“，”号分隔。
- Referer参数支持通配符“*”和“？”。
- 用户可以设置是否允许referer字段为空的请求访问。
- 白名单为空时，不会检查referer字段是否为空（不然所有的请求都会被拒绝）。
- 白名单不为空，且设置了不允许referer字段为空的规则；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，但设置了允许referer字段为空的规则；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。
- Bucket的三种权限（private，public-read，public-read-write）都会检查referer字段。

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开头的所有文件类型并以.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。如果输入love?,查找以love开头的一个字符结尾文件类型的文件，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文件类型并以.doc为扩展名的文件如lovey.doc、loveh.doc。

功能使用参考

- API：Put Bucket Referer
- 控制台：设置防盗链

设置跨域访问

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。当来自于A网站的页面中的JavaScript代码希望访问B网站的时候，浏览器会拒绝该访问，因为A、B两个网站是属于不同的域。

在实际应用中，经常会有跨域访问的需求，比如用户的网站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

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。具体的CORS规则可以参考W3C CORS规范。其实现如下：

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。如果允许的话，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*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。

这里有几个要点：

- 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。

功能使用参考

- API：跨域资源共享
- SDK：Java SDK-跨域资源共享
- 控制台：跨域资源共享

服务器端加密编码

OSS支持在服务器端对用户上传的数据进行加密编码（Server-Side Encryption）：用户上传数据时，OSS对收到的用户数据进行加密编码，然后再将编码得到的数据永久保存下来；用户下载数据时，OSS自动对保存的编码数据进行解码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编码。换句话说，下载一个进行服务器端加密编码的Object和下载一个普通的Object没有多少区别，因为OSS会为用户管理整个编解码过程。
OSS的服务器端加密编码是Object的一个属性。用户创建一个Object的时候，只需要在Put Object的请求中携带x-oss-server-side-encryption的HTTP Header，并指定其值为AES256，即可以实现该Object的服务器端加密编码存储。目前支持服务器端加密编码的操作包括：

- Put Object
- Copy Object
- Initiate Multipart Upload

细节分析

- 除了Put Object、Copy Object和Initiate Multipart Upload以外，其他OSS收到的请求中如果出现x-oss-server-side-encryption头，OSS会直接返回HTTP状态码：400；并在消息体内注明错误码是：InvalidArgument。

- 目前，OSS只支持AES256加密编码算法，如果用户指定x-oss-server-side-encryption头为其他值，OSS会直接返回HTTP状态码：400；并在消息体内注明错误码是：InvalidEncryptionAlgorithmError。
- 通过服务器端加密编码存储的Object，在下述API请求中OSS会返回x-oss-server-side-encryption头，其值为服务器端使用的熵编码加密算法。
    - Put Object
    - Copy Object
    - Initiate Multipart Upload
    - Upload Part
    - Complete Multipart Upload
    - Get Object
    - Head Object

具体实现

- API: Append Object
- API: Put Object
- API: Copy Object
- API: Post Object

静态网站托管

OSS支持静态网站托管。用户可以通过OSS 控制台将自己的存储空间配置成静态网站托管模式。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/
```

为了使用户更方便地管理在OSS上托管的静态网站，OSS支持两种功能：

- 索引页面支持（Index Document Support）
  索引文档是指当用户直接访问静态网站根域名时，OSS返回的默认索引文档（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，就必须指定一个索引文档。
- 错误页面支持（Error Document Support）
  错误文档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 FOUNT”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。

举个例子：用户设置了索引页面支持为index.html, 错误页面支持为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

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，APS.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CNAME来实现。
- 由于直接使用Bucket域名进行访问时OSS做了限制，您的文件无法直接在浏览器显示，建议您使用CNAME。
- 将一个Bucket设置成静态网站托管模式时，必须指定索引页面，错误页面是可选的。
- 将一个Bucket设置成静态网站托管模式时，指定的索引页面和错误页面必须是该Bucket内的一个Object。
- 在将一个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Get Bucket结果。
- Bucket设置静态网站托管模式后，对于静态网站根域名的访问或者访问不存在的Object会返回给用户设定的Object，对此返回的流量和请求将会计费。

功能使用参考

- API：PutBucketWebsite
- 控制台：静态网站托管

监控服务

OSS监控服务

OSS监控服务为用户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，帮助用户跟踪请求、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监控指标分类详细，主要可以归类为基础服务指标、性能指标和计量指标，详见OSS监控指标参考手册。

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，如下说明：

- 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2016-05-10 08:00:00到2016-05-10 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2016-05-10 08:00:00，这条数据将会在2016-05-10 09:30:00左右被输出。如果这条数据是从2016-05-01 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2016-05-10 09:00:00。如果2016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2016-05-01 00:00:00。

OSS报警服务

每个用户最多能够设置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。

- 报警服务相关概念参考报警服务概览。
- OSS报警服务使用指南详见OSS报警服务使用指南。
- OSS具体监控指标详见OSS监控指标参考手册。

监控数据保留策略

监控数据保留31天，过期自动清除，如果需要脱机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，请使参考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，参考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，使用方法可以参考如下文档:

- 云监控OpenAPI使用手册
- 云监控SDK使用手册
- OSS监控数据访问指南

监控、诊断和故障排除

监控诊断和故障排除中通过详细介绍以下各个方面的内容帮助用户更好的了解OSS服务的运行状态以及自主诊断和排除故障：

- 实时服务监控
  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；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。
- 问题排查
  提供常见的问题场景和故障排除方法。

注意事项

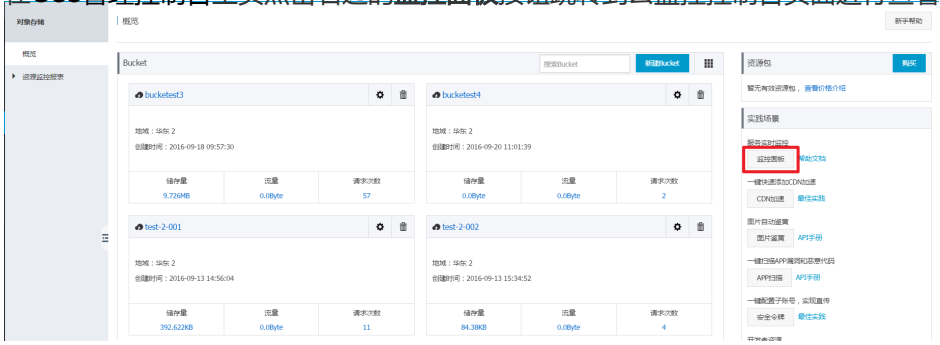
OSS Bucket全局唯一，如果删掉Bucket之后再创建同名的Bucket，那么被删掉的Bucket的监控以及报警规则会作用在新的同名Bucket上。

监控服务使用指南

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：

在OSS管理控制台主页点击右边的**监控面板**按钮跳转到云监控控制台页面进行查看。



直接进入云监控控制台查看OSS监控服务，如下图所示：



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，可以点击右上角的**刷新**按钮自动更新数据信息。

点击**前往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

该模块主要展示用户拥有的bucket总数以及相关的报警规则情况。



- 点击**Bucket数量**的数字，链接到Bucket列表Tab页。
- 点击**报警规则总数**的数字，链接到报警规则Tab页。
- 点击**处于告警状态**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。
- 点击**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。
- 点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该用户从当月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。



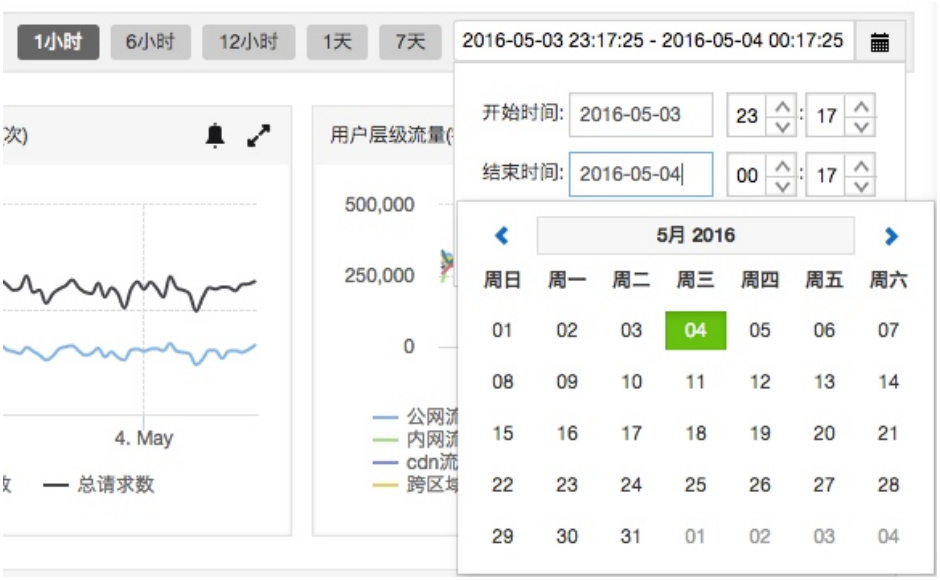
用户层级监控指标

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。

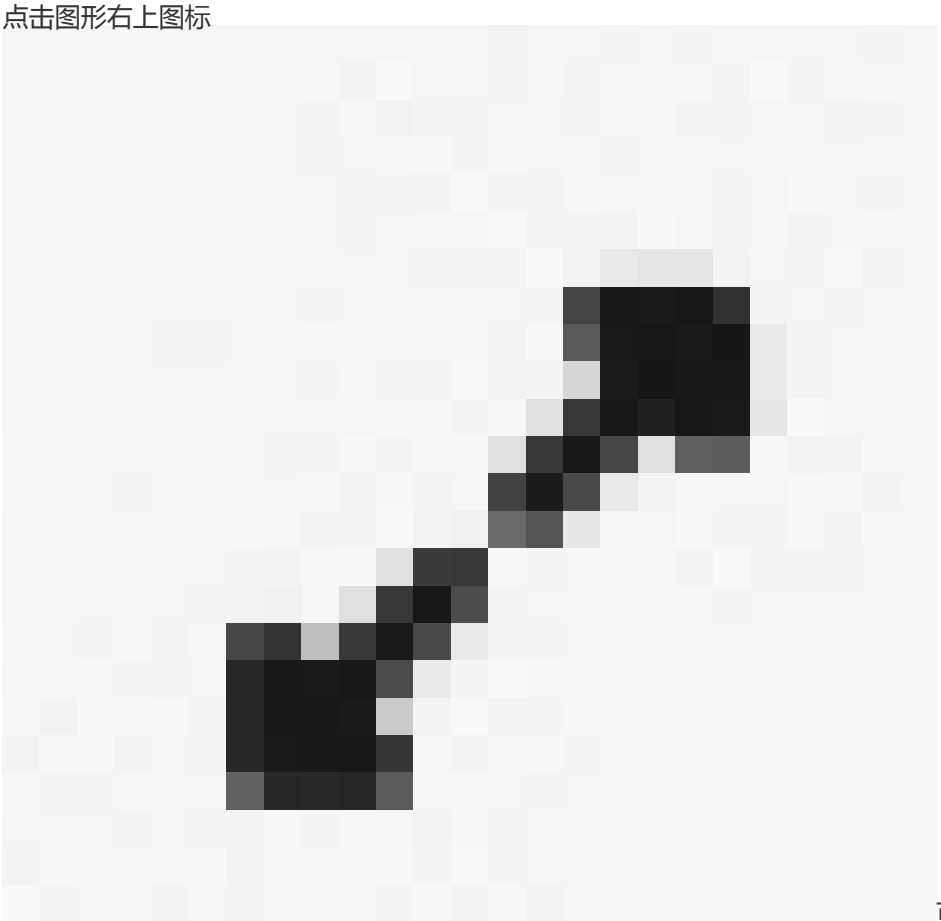


图表展示还支持以下功能:

点击相关图例可以将该指标曲线隐去，如下图：

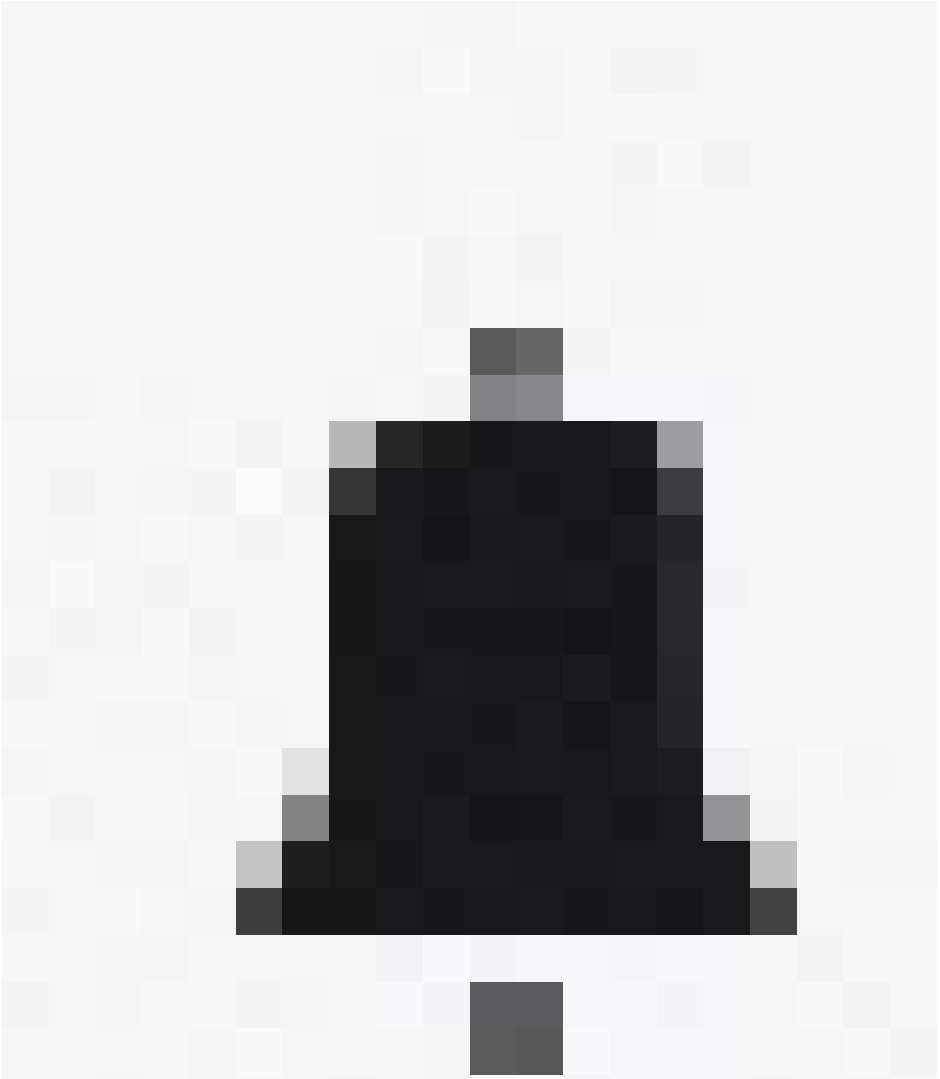


点击图形右上图标



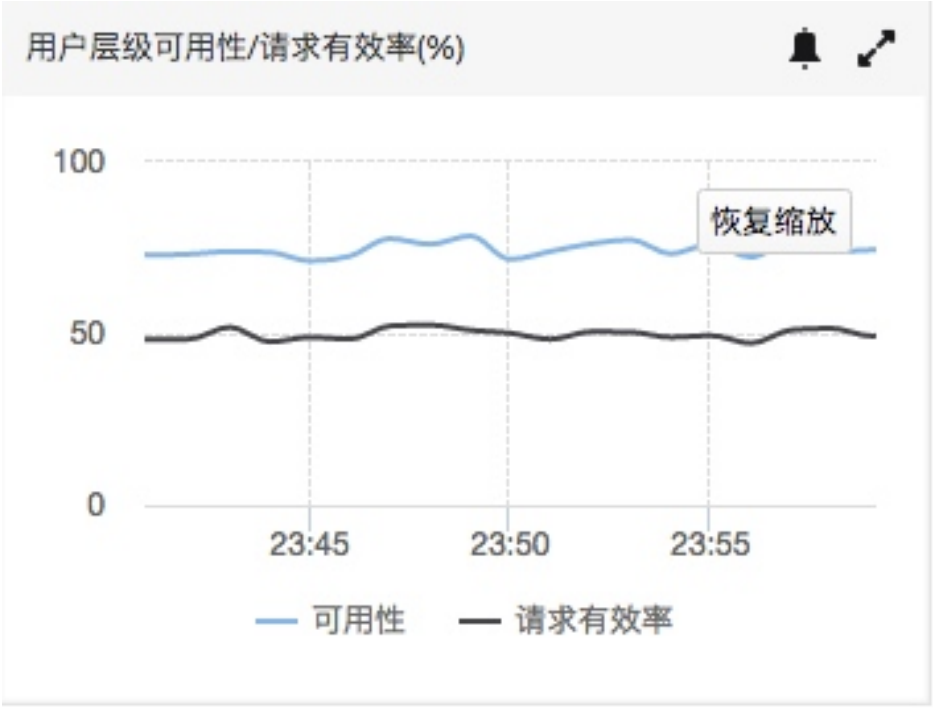
可以将图形放大展示。注意，表格不支持放大展示。

- 点击图形右上图标



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。

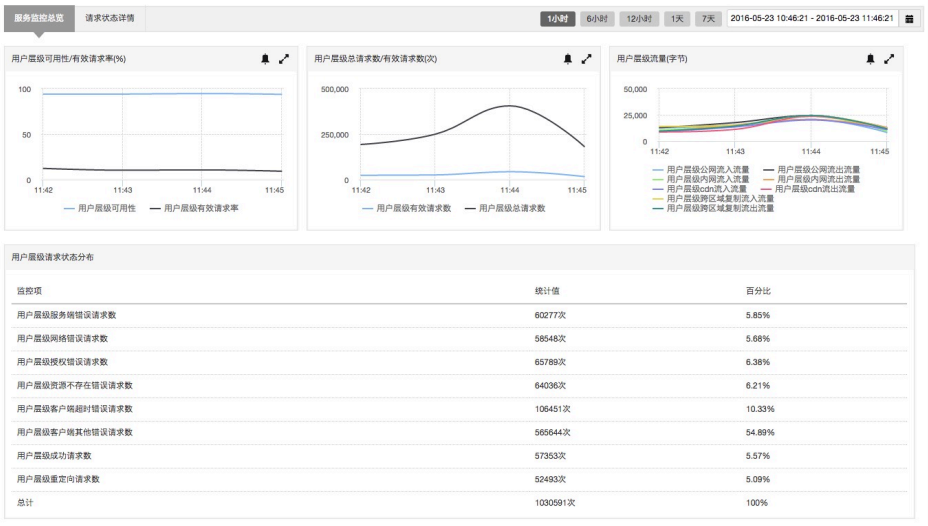
- 鼠标按住图形曲线区域拖放，可以进行时间范围快速调整放大，点击**恢复缩放**回归到拖放之前的时间范围。



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：

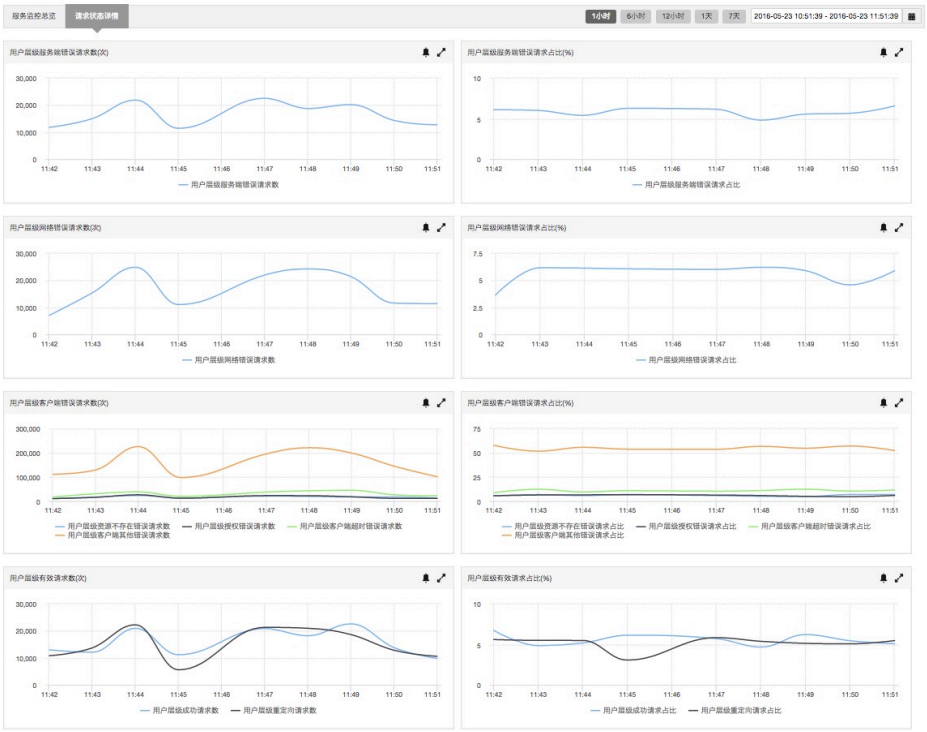
- 用户层级可用性/有效请求率: 包括可用性和有效请求率2项指标。
- 用户层级总请求数/有效请求数: 包括总请求数和有效请求数2项指标。
- 用户层级流量: 包括公网流出流量、公网流入流量、内网流出流量、内网流入流量、cdn流出流量、cdn流入流量、跨区域复制流出流量和跨区域复制流入流量8项指标。
- 用户层级请求状态分布: 该表格中展示选定时间范围内各个请求类型的个数以及占比。



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:

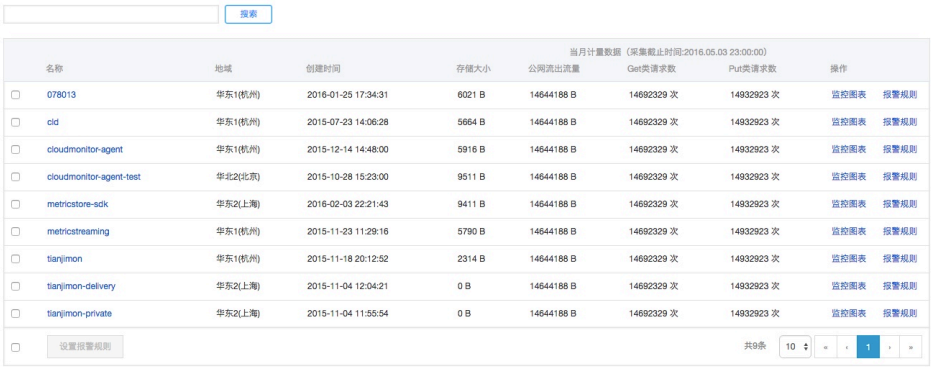
- 用户层级服务端错误请求数。
- 用户层级服务端错误请求占比。
- 用户层级网络错误请求数。
- 用户层级网络错误请求占比。
- 用户层级客户端错误请求数: 包括资源不存在错误请求数、授权错误请求数、客户端超时错误请求数和客户端其他错误请求数4项指标。
- 用户层级客户端错误请求占比: 包括资源不存在错误请求占比、授权错误请求占比、客户端超时错误请求占比和客户端其他错误请求占比4项指标。
- 用户层级有效请求数: 包括成功请求数和重定向请求数2项指标。
- 用户层级有效请求占比: 包括成功请求占比和重定向请求占比2项指标。



Bucket列表

Bucket列表信息

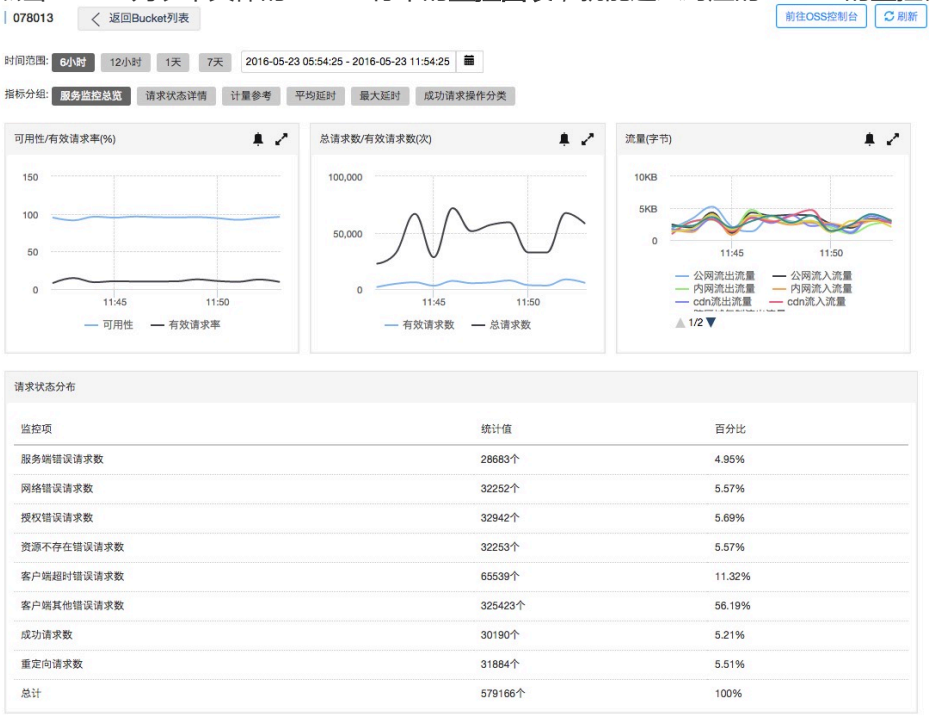
列表展现了用户拥有的所有Bucket的名称、所属地域、创建时间、当月计量数据统计信息以及相关操作，表中内容如下：



- 当月计量统计包括每个bucket各自的存储大小、公网流出流量、Put类请求数和Get类请求数。
- 点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。
- 点击报警规则，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，并点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

点击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6个指标分组:

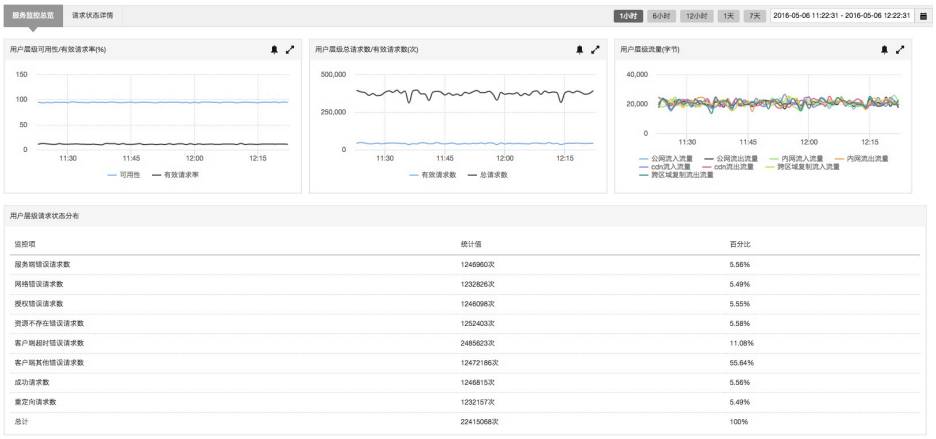
- 服务监控总览
- 请求状态详情
- 计量参考
- 平均延时
- 最大延时
- 成功请求操作分类

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。点击上方的**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2项指标。
- 总请求数/有效请求数: 包括总请求数和有效请求数2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8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。



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	1232626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	224150683次	100%

请求状态详情

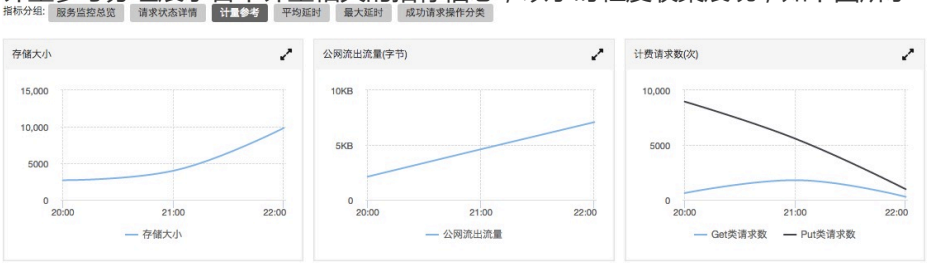
该指标分组同用户层级的请求状态详情，只是从具体的Bucket进行监控，主要包括下面监控指标图：

- 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4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4项指标。
- 有效请求数: 包括成功请求数和重定向请求数2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2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2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。



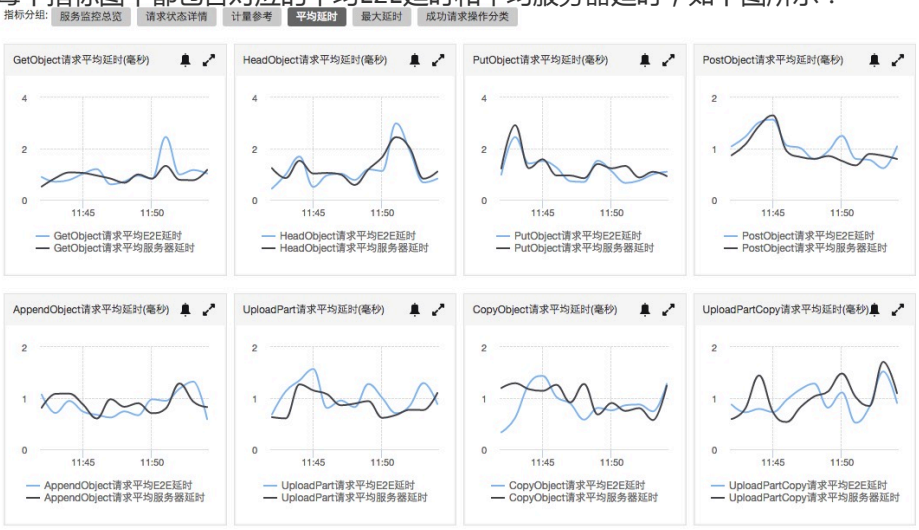
平均延时

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：

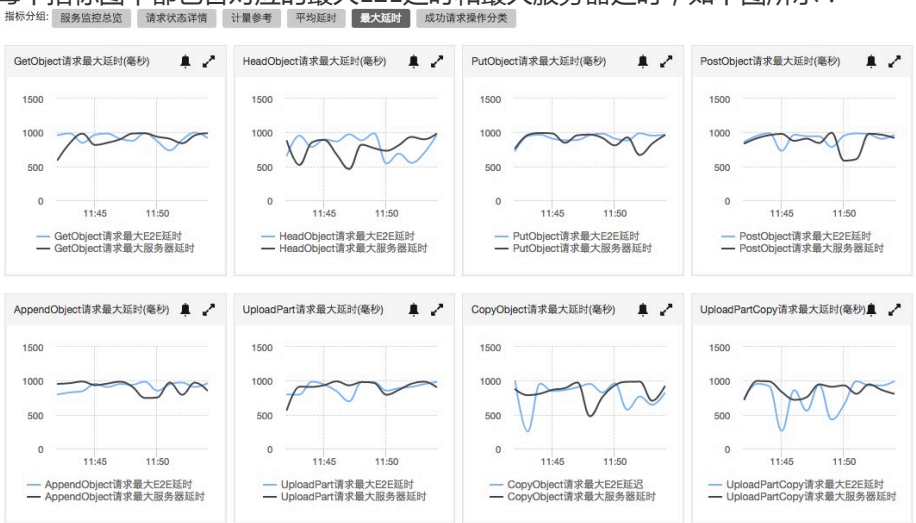


最大延时

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：

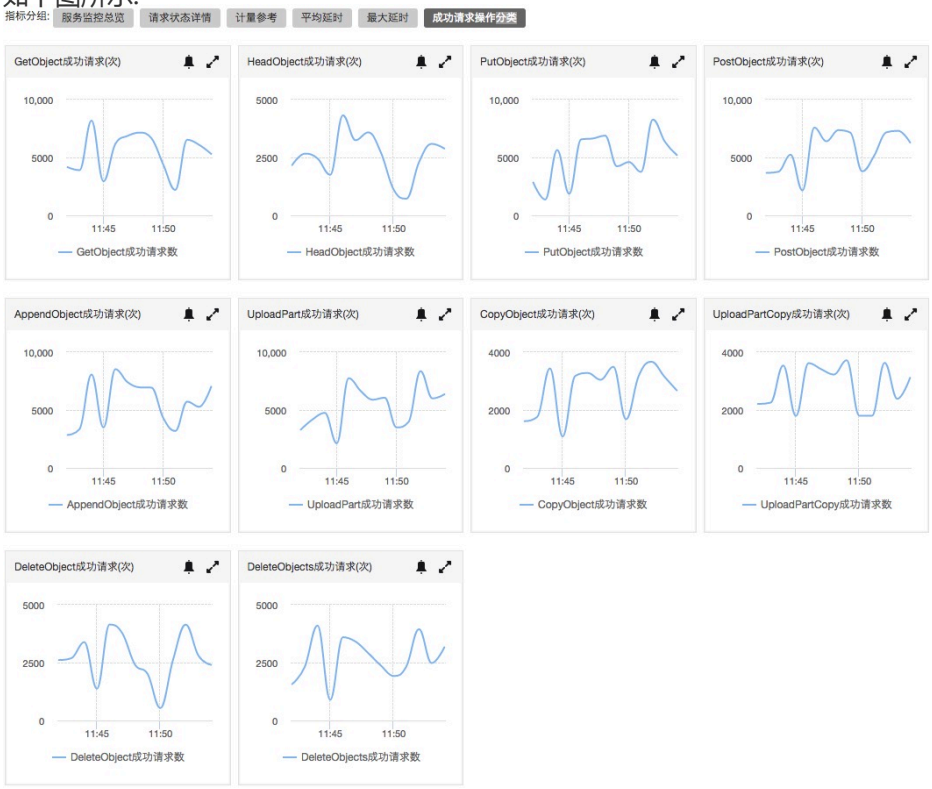


成功请求操作分类

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页能够展示和管理该用户的所有的报警规则，如下图所示



报警规则页的使用和相关说明详见报警服务使用指南。

监控关注事项以及使用指导

监控关注点以及使用指南可以参考监视诊断和故障排除的相关章节。

报警服务使用指南

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，而且能够查看该对应报警规则的历史报警情况。



- 点击对应报警规则的**修改**，就可以对该报警规则进行修改。
- 点击对应报警规则的**删除**，就可以删除该报警规则。选中多条报警规则，然后点击表格最下方的**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，点击该报警规则的**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，然后点击表格最下方的**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，点击该报警规则的**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，然后点击表格最下方的**启用**按钮，可以批量启用报警规则。
- 点击对应报警规则的**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，如下图所示:



这里解释一下报警历史的相关概念:

- 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。另外，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24小时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)。这个时候系统是不会再向用户发送告警信息进行通知的，一直持续到24小时之后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。

另外，点击具体报警规则的通知对象旁边的**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:



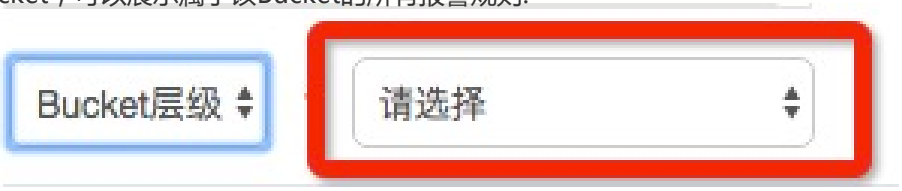
如何快速定位报警规则

根据报警规则页中的下面控件信息能够快速定位到被搜索的报警规则:

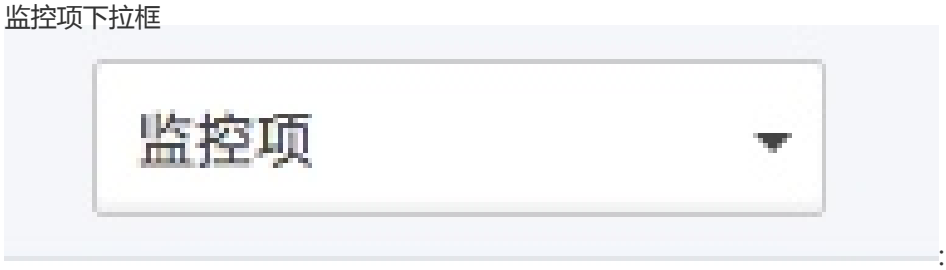
报警维度下拉框: 全部和Bucket层级。当选项为**全部**时，显示所有用户层级和Bucket层级的报警规则。



Bucket下拉框:当报警维度下拉框为**Bucket层级**时，这里可以罗列该用户的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**监控项**时，显示用户层级或者Bucket层级所有监控项的报警规则。

报警状态下拉框



: 报警状态、告警和正常。当为报警状态时显示所有状态的报警规则。

启用状态下来框



: 启

用状态、已禁用和已启用。当为启用状态时，显示所有状态的报警规则。

查看全部报警规则

点击**报警规则**Tab页进入报警规则也默认显示全部报警规则。将报警维度下拉框设置为**全部**，即能显示全部报警规则。配合**监控项**、**报警状态**和**启用状态**这三个下拉框能够更精确定位全部维度下的报警规则。

查看特定Bucket的报警规则

如果需要查看某个特定的Bucket的报警规则，需要设置报警维度下拉框为Bucket层级，然后在Bucket下拉框中选择对应的Bucket名称。通过**Bucket列表**中的对应Bucket的**报警规则**进入的报警页，就能显示该Bucket的所有的报警规则。配合**监控项**、**报警状态**和**启用状态**这三个，即能显示该报警维度下更精确的报警规则。

查看与具体监控项相关的报警规则

设置监控项下拉框为具体的某项监控项，就能显示对应监控项的所有报警规则。

查看指定报警状态的报警规则

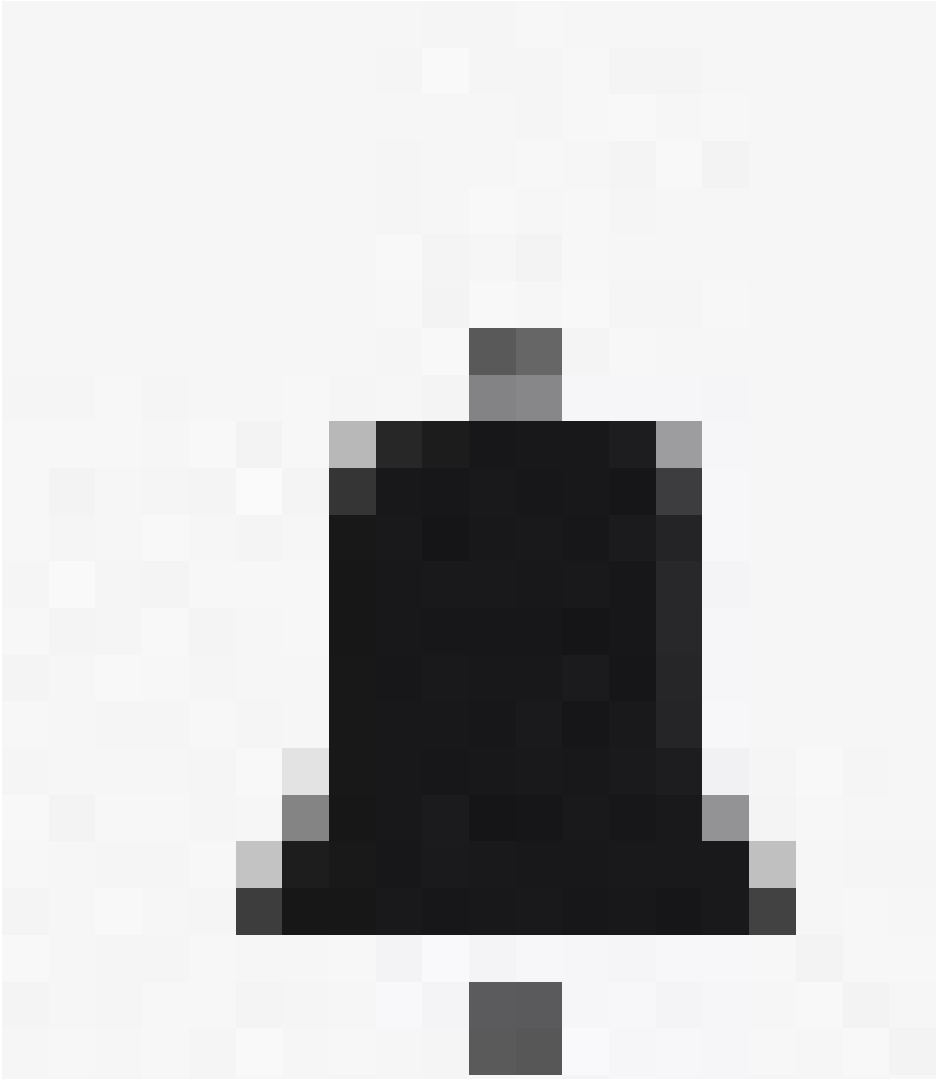
设置报警状态下拉框为指定报警状态，如告警，就能展示所有处于该告警状态的报警规则。

查看指定启用状态的报警规则

设置启动状态下拉框为指定启用状态，如已禁用，就能显示已经被禁用的所有的报警规则。

如何添加报警规则

在Bucket列表Tab页选中指定的Bucket之后点击**设置报警规则**按钮，或者在用户概况Tab页亦或某个Bucket监控视图页中的监控图形上的

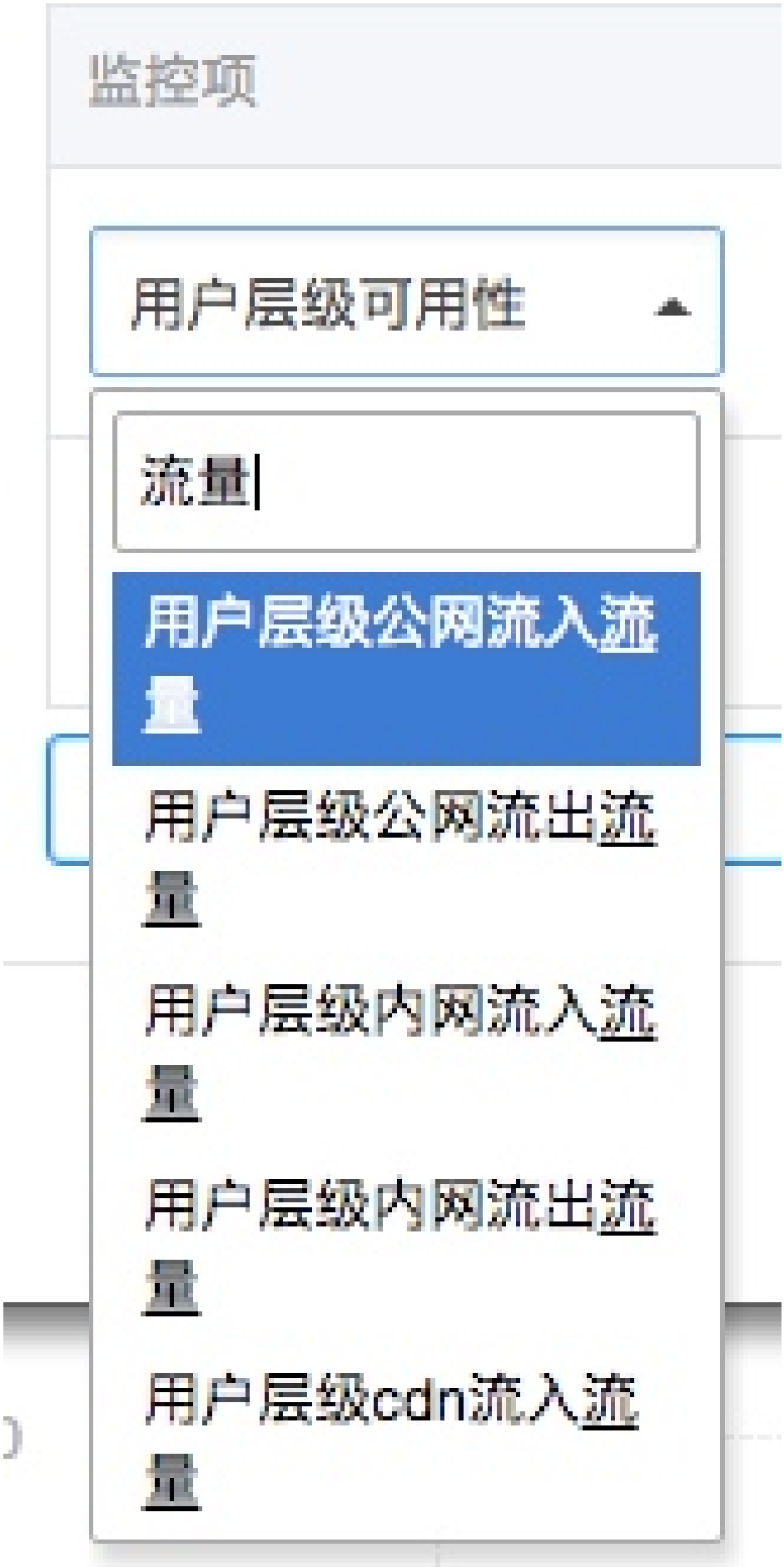


都可以打开**批量设置报警规则**窗口，进行报警规则的设置。下面演示一下报警规则设置的过程(注意，下面出现的名词概念请详见云监控**报警服务概览**:

1.首先进入如下**设置报警规则**步骤:



- 报警维度: 提示用户目前设置的报规则属于哪个维度的监控。如果是Bucket层级，则会提示目前为哪个具体的Bucket设置报警规则，图例中显示的是为用户层级。
- 监控项: 该多选框显示属于该报警维度的所有的监控项选项。可以通过快速搜索框迅速定位:


监控项
用户层级可用性
流量
用户层级公网流入流量
用户层级公网流出流量
用户层级内网流入流量
用户层级内网流出流量
用户层级cdn流入流量

- 统计周期: 间隔多久进行一次数据统计，默认为5min。
- 连续次数: 指连续几个统计周期监控项的值持续超过阈值后触发报警。
- 统计方法: 指定该监控项中的具体的统计指标，OSS监控服务的统计方法均为**监控值**。
- 添加报警规则按钮: 点击按钮可以设置更多监控项的报警规则。
- 删除: 点击对应报警规则后的**删除**，可以删除该报警规则。

2.设置完成之后，点击**下一步**，进入**设置通知对象**步骤，如下图:



如果事先已经按照**报警联系人和联系组**设置好报警联系组，那么就会像图例中展示出已经存在的报警联系人组。如果没有设置过报警联系人组，则点击**快创建联系人组**，按照提示创建即可。

3.点击**确定**，完成报警规则的设置。



Bucket列表添加报警规则

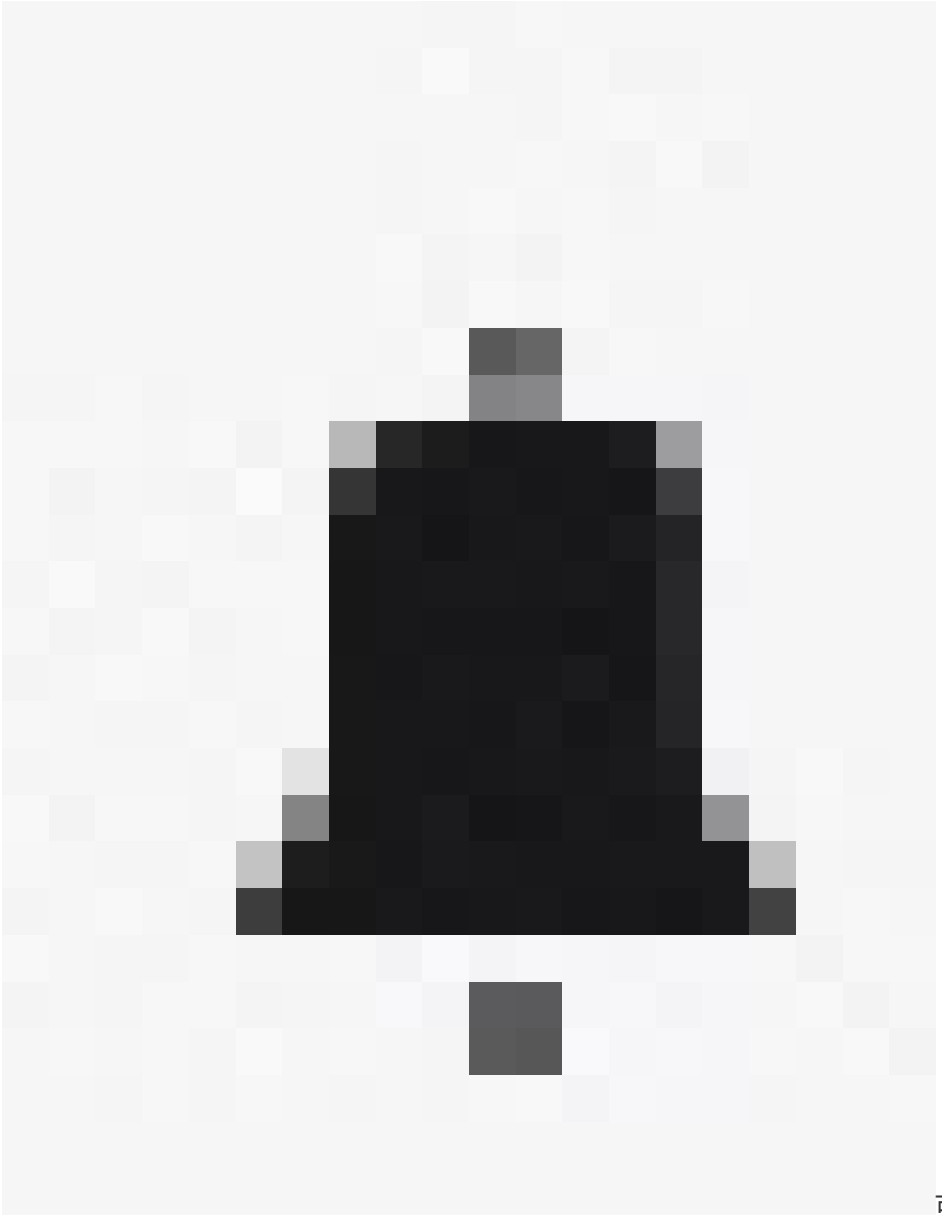
在Bucket列表Tab页中可以为多个Bucket批量添加相同的报警规则。选中需要配置报警规则的Bucket，点击**设置报警规则**按钮，进入报警规则设置页面。



注意，批量设置时报警维度是确定的，监控项都是Bucket层级的。

具体监控图表中添加报警规则

在**用户概况**Tab页或者具体Bucket的监控视图页中，点击具体的监控图右上方的



可以为该监控图用的监控指标项设置报警规则。

注意，通过点击监控图的报警图标进入报警规则页面时，报警维度都是确定的，而且只能设置该图中的监控项的报警规则。

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的报警规则依赖存在。建议用户在删除Bucket之前先删除对应的报警规则。

监控数据访问指南

本章内容为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Project

OSS监控服务指标项的数据都使用相同的Project名称：acs_oss。

使用JAVA SDK设置代码示例如下：

```
QueryMetricRequest request = new QueryMetricRequest();
request.setProject("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]，处于边界StartTime的数据不会被获取，而处于边界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。

其他时间参数信息详见云监控API接口说明。

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监控指标参考手册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

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
AvailabilityAvailability	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
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
ClientOtherErrorRate	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
HeadObjectE2eLatency	HeadObject请求平均E2E延时	毫秒	Bucket层级
HeadObjectServerLatency	HeadObject请求平均服务器延时	毫秒	Bucket层级
MaxHeadObjectE2eLatency	HeadObject请求最大E2E延时	毫秒	Bucket层级
MaxHeadObjectServerLatency	HeadObject请求最大服务器延时	毫秒	Bucket层级
PutObjectE2eLatency	PutObject请求平均E2E延时	毫秒	Bucket层级
PutObjectServerLatency	PutObject请求平均服务器延时	毫秒	Bucket层级
MaxPutObjectE2eLat	PutObject请求最大	毫秒	Bucket层级

ency	E2E延时		
MaxPutObjectServerLatency	PutObject请求最大服务器延时	毫秒	Bucket层级
PostObjectE2eLatency	PostObject请求平均E2E延时	毫秒	Bucket层级
PostObjectServerLatency	PostObject请求平均服务器延时	毫秒	Bucket层级
MaxPostObjectE2eLatency	PostObject请求最大E2E延时	毫秒	Bucket层级
MaxPostObjectServerLatency	PostObject请求最大服务器延时	毫秒	Bucket层级
AppendObjectE2eLatency	AppendObject请求平均E2E延时	毫秒	Bucket层级
AppendObjectServerLatency	AppendObject请求平均服务器延时	毫秒	Bucket层级
MaxAppendObjectE2eLatency	AppendObject请求最大E2E延时	毫秒	Bucket层级
MaxAppendObjectServerLatency	AppendObject请求最大服务器延时	毫秒	Bucket层级
UploadPartE2eLatency	UploadPart请求平均E2E延时	毫秒	Bucket层级
UploadPartServerLatency	UploadPart请求平均服务器延时	毫秒	Bucket层级
MaxUploadPartE2eLatency	UploadPart请求最大E2E延时	毫秒	Bucket层级
MaxUploadPartServerLatency	UploadPart请求最大服务器延时	毫秒	Bucket层级
UploadPartCopyE2eLatency	UploadPartCopy请求平均E2E延时	毫秒	Bucket层级
UploadPartCopyServerLatency	UploadPartCopy请求平均服务器延时	毫秒	Bucket层级
MaxUploadPartCopyE2eLatency	UploadPartCopy请求最大E2E延时	毫秒	Bucket层级
MaxUploadPartCopyServerLatency	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

AppendObjectCount	AppendObject成功请求数	次数	Bucket层级
UploadPartCount	UploadPart成功请求数	次数	Bucket层级
UploadPartCopyCount	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，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");
```

监控指标参考手册

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，详细介绍参见下文。

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）占总请求的百分比而得到
有效请求率	%	有效请求占总请求数的百分比，有效请求的介绍见下面说明
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线流量
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
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
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

监控想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
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	毫秒	请求API为UploadPart的成功请

时		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

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。

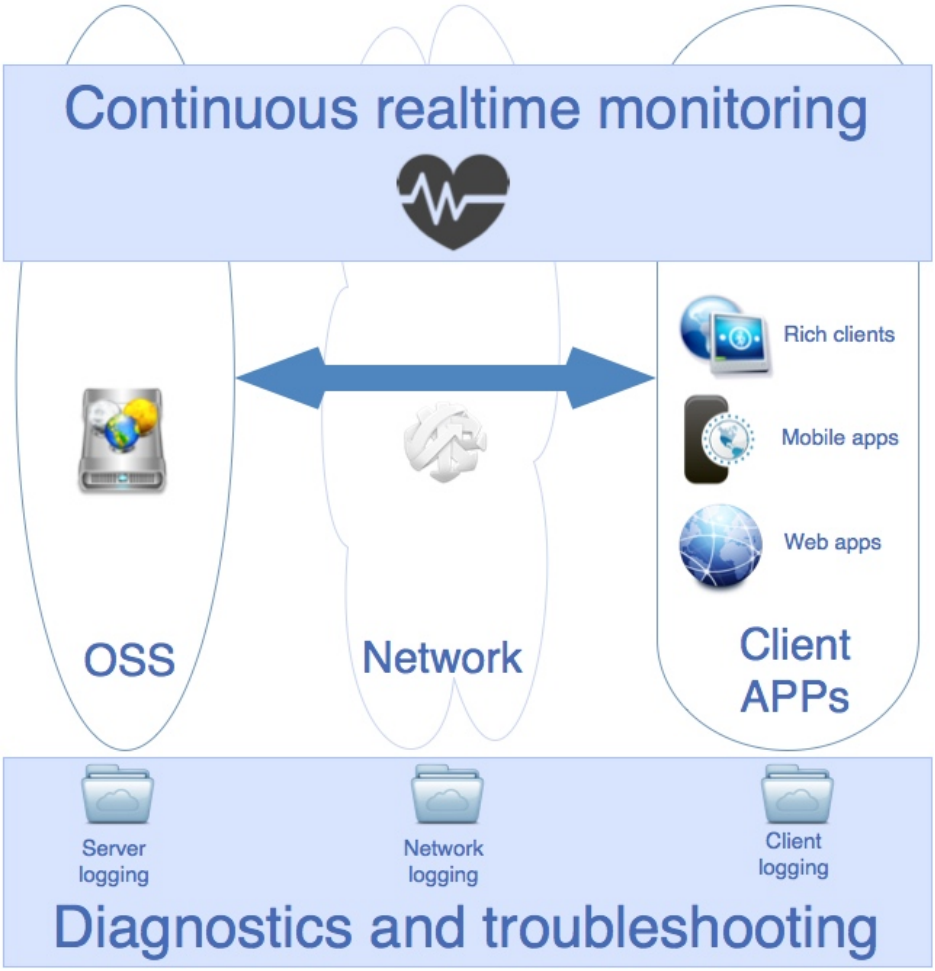
概述

本章主要描述如何使用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。

本章内容组织如下：

- 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。

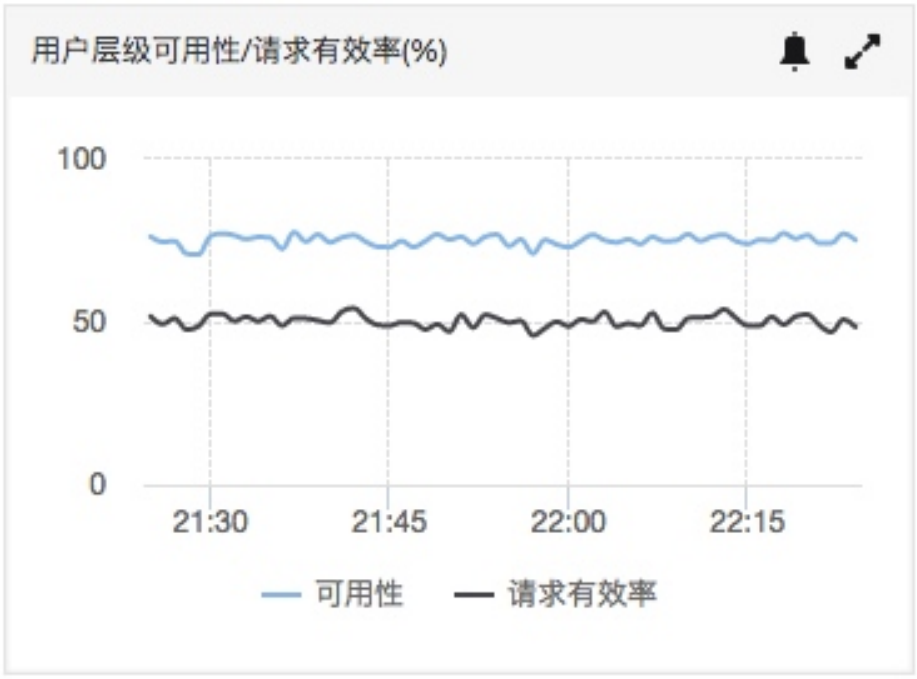


服务监控

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。



可用性也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、具体的监视某类请求。



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。

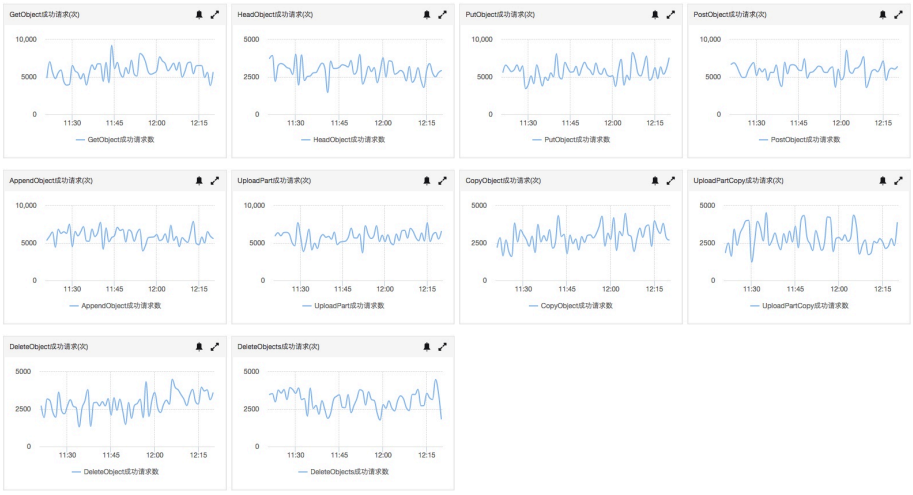
平均延时，包括E2E平均延时和服务器平均延时



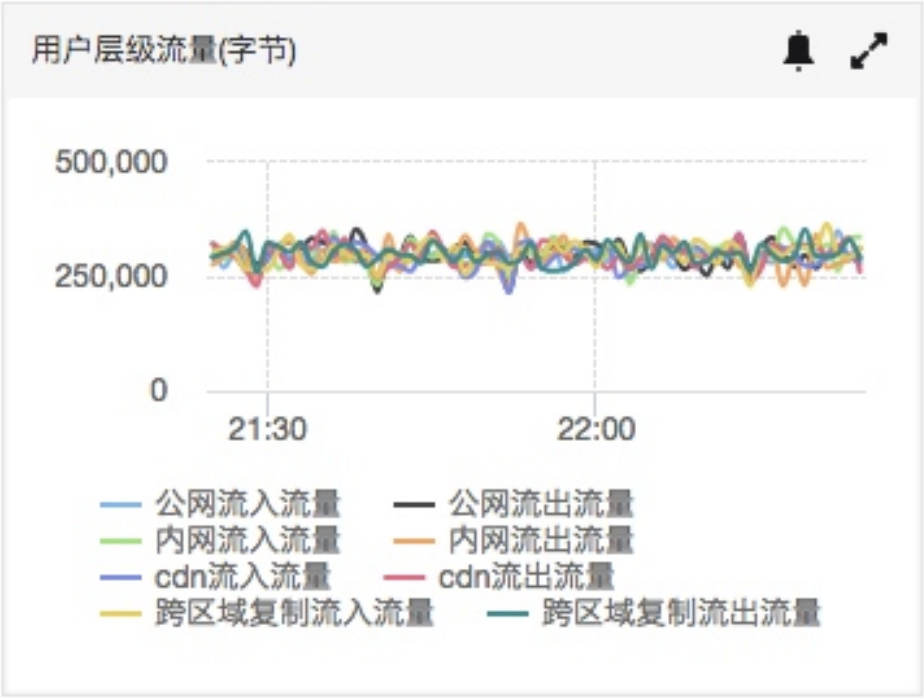
最大延时，包括E2E最大延时和服务器最大延时


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
GetObject请求最大延时(毫秒)
HeadObject请求最大延时(毫秒)
PutObject请求最大延时(毫秒)
PostObject请求最大延时(毫秒)
AppendObject请求最大延时(毫秒)
UploadPart请求最大延时(毫秒)
CopyObject请求最大延时(毫秒)
UploadPartCopy请求最大延时(毫秒)

成功请求操作分类


GetObject成功请求(次)
HeadObject成功请求(次)
PutObject成功请求(次)
PostObject成功请求(次)
AppendObject成功请求(次)
UploadPart成功请求(次)
CopyObject成功请求(次)
UploadPartCopy成功请求(次)
DeleteObject成功请求(次)
DeleteObjects成功请求(次)

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

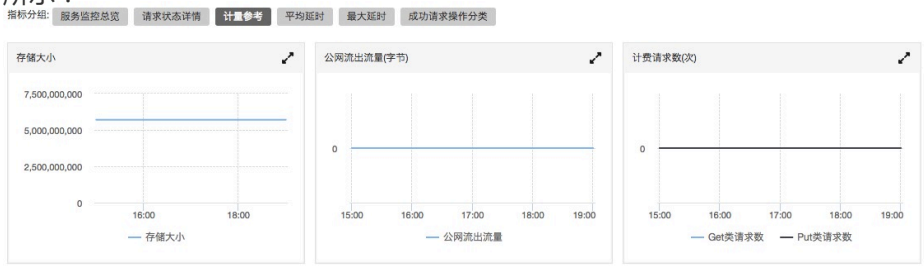
流量指标从用户或者具体的Bucket层级的总体情况进行监控，关注公网、内网、cdn回源以及跨域复制等场景中的网络资源占用状况。

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

攥写本文档时，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。能帮助用户实时了解本月使用资源的情况并计算消费，如下图所示:





> **注意:** 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，请参考日志设置。

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。

WireShark使用参考WireShark安装使用。

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服务始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。

调查客户端的性能问题

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

）。

调查网络延迟问题

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。

WireShark使用参考WireShark安装使用。

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。WireShark使用参考WireShark安装使用。

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。

WireShark使用请参考WireShark安装使用。

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:

i. 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。
ii. 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。