天翼云
State Cloud

对象存储（经典版）I 型

(Object-Oriented Storage,OOS)

开发者文档 V5

天翼云科技有限公司

目录

1 产品简介

1.1 产品介绍

对象存储（经典版）I型（Object-Oriented Storage，OOS）是中国电信为客户提供的一种海量、弹性、廉价、高可用的存储服务。客户只需花极少的钱就可以获得一个几乎无限的存储空间，可以随时根据需要调整对资源的占用，并只需为真正使用的资源付费。

OOS 提供了基于 Web 门户和基于 HTTP REST 接口两种访问方式，用户可以在任何地方通过互联网对数据进行管理和访问。OOS 提供的 REST 接口与 Amazon S3 兼容，因此基于 OOS 的业务可以非常轻松的与 Amazon S3 对接。您也可以通过 OOS 提供的 SDK 来调用 OOS 服务，开发语言目前支持 Java, Android, iOS, C，Python。

1.2 产品优势

OOS 具有以下特点：

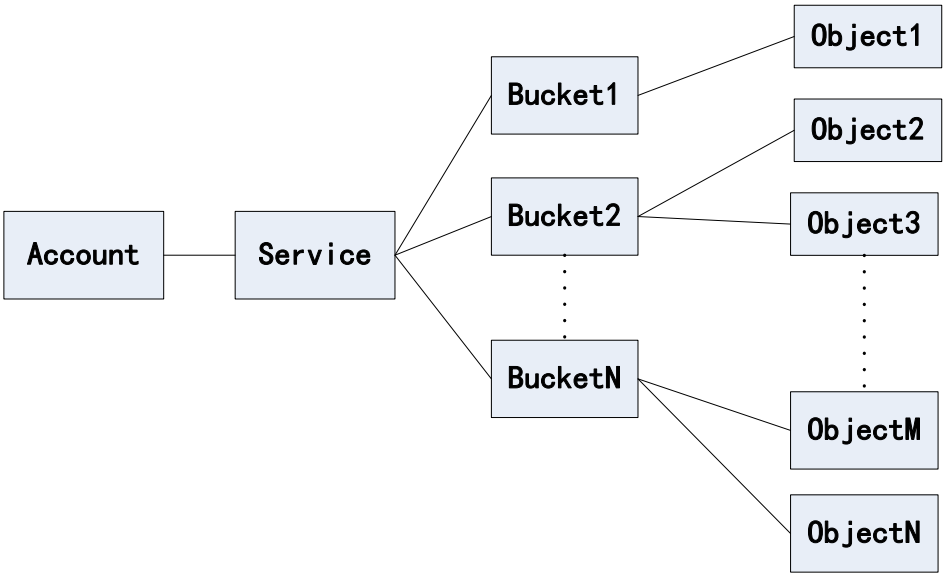
- 容量几乎没有上限，弹性扩容；
- 自动数据冗余和故障切换，确保高可用；
- 流量按需计费，节省带宽成本；
- 易于管理、维护和升级。

2 OOS 主要概念

OOS 的主要概念有：

- Account（账户）：用户登录时 OOS 使用的账户。
- Service（服务）：OOS 为用户提供的服务。
- Object（对象）：用户存储在 OOS 上的每个对象都是一个 Object，也称为 OOS 的文件。
- Bucket（容器）：存储 Object 的容器。

它们之间的关系如下所示。



在使用 OOS 之前，首先需要在天翼云网站开通一个 Account（账户），然后联系天翼云客服工作人员开通 OOS 服务，OOS 会为该账户提供服务（Service），在该服务下，用户可以创建 1 个或多个 Bucket（容器），每个存储桶中可以存储不限数量的 Object（对象）。

2.1 Account

在使用 OOS 之前，需要已开通天翼云账户，联系天翼云客服人员（客服电话：400-810-9889）申请开通 OOS 服务。开通 OOS 服务成功之后，用户可以用该账户登录并使用 OOS 服务。

2.2 Service

本文中的 Service 是 OOS 为用户提供的 OOS 服务，该服务为用户提供弹性可扩展的存储空间，用户可以根据自己的业务需要建立一个或者多个的容器（Bucket）。

2.3 Bucket

Bucket 是存储 Object 的容器。OOS 的每个 Object 都必须包含在一个 Bucket 中。Bucket 不能嵌套，每个 Bucket 中只能存放 Object，不能再存放 Bucket，每个 bucket 下可以创建目录。

2.3.1 Bucket 的命名规范

Put Bucket 用来创建一个 Bucket。只有已注册 OOS 账户才能创建 Bucekt，匿名请求无效，创建 Bucket 的用户将是 Bucket 的拥有者。

Bucket 的命名规则如下：

- Bucket 名称必须全局唯一；
- Bucket 名称长度介于 3 到 63 字节之间；
- Bucket 名称只能由小写字母、数字、短横线（-）和点（.）组成；
- Bucket 名称可以由一个或者多个小节组成，小节之间用点（.）隔开，各个小节需要：
  - 只能包含小写字母、数字和短横线（-）；
  - 必须以小写字母或者数字开始；
  - 必须以小写字母或者数字结束。
- Bucket 名称不能是一组或多组“数字.数字”的组合（如 192.162.0.1）；
- Bucket 名称中不能包含双点（..）、横线点（-.）和点横线（.-）；
- 不允许使用非法敏感字符，例如暴恐涉政相关信息等。

2.3.2 Bucket 的基本操作

成功开通 OOS 的用户可以新建 Bucket、删除 Bucket 以及查看 Bucket 的属性。

新建 Bucket：需要输入 Bucket 的名称，选择操作权限（Bucket 的推荐及默认操作权限是 private）。Bucket 一旦创建成功，其名将不可更改。所有者可以更改 Bucket 的操作权限。

删除 Bucket：只有不包含任何对象的 Bucket 才能删除。若 Bucket 中包含对象，则需要将 Bucket 内的所有对象删除，该 Bucket 才能被删除。只有所有者可以删除 Bucket。

查看 Bucket 属性：查看 Bucket 的属性时，用户可以更改 Bucket 的操作权限。

2.4 Object

用户存储在 OOS 上的每个文件都是一个 Object。文件可以是文本、图片、音频、视频或者网页。OOS 支持的单个文件的最大长度为 5TiB 字节。

用户可以上传、下载和删除 Object。此外用户还可以对 Object 的组织形式进行管理，将 Object 移动或者复制到目标目录下。

2.5 AccessKeyId 和 SecretKey

AccessKeyId 和 SecretKey 是您访问 OOS 的密钥，OOS 会通过它来验证您的资源请求，请妥善保管。您可以在 OOS 的控制台—账户管理—API 密钥管理页面中查看到 AccessKeyId 和 SecretKey。密钥分为主密钥和普通密钥两种类型，每个用户可以拥有最多 10 个主密钥和普通密钥，两者的区别是：

- 主密钥用于生成普通密钥，每个账户必须至少拥有一个主密钥。
- 密钥可以被禁止使用，或者启用。当账户的主密钥只剩下一个时，该密钥不能被禁用或者删除。
- 用户可以将普通密钥设置成为主密钥。
- 普通密钥不能创建，删除，修改 bucket 属性。
- 普通密钥只能操作以自己 AccessKey 开头的 Object，包括创建，删除，下载 Object 等操作。例如：普通 AccessKey 为 e67057e798af03040565，那么该 AccessKey 只能创建以 e67057e798af03040565 开头的 Object 名。
- 普通密钥可以 list objects，但是参数 prefix 必须以 AccessKey 开头，即普通密钥只能 list 以自己的 AccessKey 开头的 Object。
- 在使用 SDK 访问 AccessKey 相关的 API 时，需要 setEndpoint 为 IAM 的 endpoint。

2.6 访问域名（Endpoint）

当您使用 API 访问时，您必须指定您的访问域名（Endpoint），不同的地域有不同的访问域名，访问域名详见 **Endpoint 列表**。

本文档中涉及到域名的都以江苏（oos-js.ctyunapi.cn）为例，其他的访问域名参见 **Endpoint 列表**。

3 安全策略

为了保证用户数据的安全，OOS 提供三种安全策略来保障用户数据的安全性：用户签名验证、Bucket 权限控制、Bucket Policy 安全策略。

3.1 用户签名验证（Authentication）

为了防止用户数据被他人盗取，所有发送到 OOS 的非匿名请求都需要经过签名验证。OOS 通过使用 Access Key ID/Secret Access Key 对称加密的方法来验证某个请求的发送者身份。Access Key ID 和 Secret Access Key 在 OOS 服务开通后自动随机生成。当用户想以个人身份向 OOS 发送请求时，需要首先将发送的请求按照 OOS 指定的格式生成签名字符串；然后使用 Secret Access Key 对签名字符串进行加密产生验证码。OOS 收到请求以后，会通过 Access Key ID 找到对应的 Secret Access Key，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，即认为该请求时有效的；否则，OOS 将拒绝处理这次请求，并返回错误码。

3.1.1 Head 中包含签名

用户可以在 HTTP 请求中增加 Authorization（授权）的 Head 来包含签名信息，表明这个消息已被授权。如果用户请求中没有 Authentication 字段，则认为是匿名访问。

验证码计算方法如下：

```
Authorization: "AWS " + AccessId + ":" + Signature
Signature =Base64( HMAC-SHA1( YourSecretAccessKey, UTF-8-Encoding-
Of( StringToSign ) ) ) ;
StringToSign = HTTP-VERB + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n" +
CanonicalizedAmzHeaders + "\n" +
CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
```

```
<HTTP-Request-URI > +
[ sub-resource, 如果存在的话。例如 "?acl", "?logging", or "?website"];
CanonicalizedAmzHeaders = <在下面描述>
```

HMAC-SHA1 是由 RFC 2104 定义的算法。该算法需要输入两个字符串：一个 key 和一个 message。OOS 在验证请求时，使用您的 Secret Access Key 作为 key，使用 UTF-8 编码的 StringToSign 作为 message。HMAC-SHA1 的输出同样是个字符串，被称为摘要。Signature 请求参数是这个摘要的 Base64 编码。

说明：

（1） Content-Md5 表示请求内容数据的 MD5 值；

（2） CONTENT-TYPE 表示请求内容的类型；

（3） DATE 表示此次操作的时间，且必须为 HTTP1.1 中支持的 GMT 格式；

（4） CanonicalizedAmzHeaders 表示 http 中的 object user meta 组合；

（5） CanonicalizedResource 表示用户想要访问的 OOS 资源

1、 构建 CanonicalizedAMZHeaders 的方法

所有以“x-amz-”为前缀的 HTTP Header 被称为 CanonicalizedAMZHeaders。它的构建方法如下：

（1） 将所有以“x-amz-”为前缀的 HTTP 请求头的名字转换成小写字母。如’X-AMZ-Meta-Name: fred’转换成’x-amz-meta-name: fred。

（2） 将上一步得到的所有 HTTP 请求头按照字典序进行升序排列。

（3） 如果有相同名字的请求头，则以最后一个值为准。如有两个名为‘x-amz-meta-name’的请求头，对应的值分别为‘fred’和‘barney’，则最后为：‘x-amz-meta-name:barney’。

（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-amz-meta-name: fred,barney’转换成：’x-amz-meta-name:fred,barney’。

（5） 将所有的头和内容用’\n’分隔符分隔拼成最后的 CanonicalizedAMZHeader。

2、 构建 CanonicalizedResource 的方法

用户发送请求中想访问的 OOS 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） 如果请求通过 Host 请求头来指定 Bucket，那么增加 Bucket 名，并在其前面加上“/”(例如/bucketname)。对于 Bucket 名称在 path 中的请求，或者没有指定 Bucket 的请求，不做任何操作。

（3） 在 path 中增加未编码的 HTTP 请求 URI，到请求参数处截止，不包含请求参数。

（4） 如果请求包含子资源，例如?acl, ?website, ?logging，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 例如：

/BucketName/ObjectName?acl

（5） 在构造 CanonicalizedResource 时，必须包含的子资源包括：

```
acl, lifecycle, location, logging, notification, partNumber, policy, requestPayment,
torrent, uploadId, uploads, versionId, versioning, versions and website
```

（6） 如果请求通过参数指定了要覆盖的响应头，那么在 CanonicalizedResource 后加上该请求参数和值。当进行签名时，不要对这些值进行编码。但是当发送请求的时候，用户必须对这些参数值进行编码。GET 请求中的这些参数包括：

```
response-content-type,response-content-language, response-expires, response-cache-
control,response-content-disposition, response-content-encoding
```

例如：/BucketName/ObjectName?response-content-type=ContentType

（7） 当发送批量删除对象请求时，delete 参数需要包含在 CanonicalizedResource 中。

（8） StringToSign 中不包含 Content-Type, Date, Content-MD5 这些请求头的名字，只包含这些请求头的值。但是以“x-amz”开头的请求头的名字和值都包含在 StringToSign 中。

（9） 如果在请求中，Content-Type, Content-MD5 等请求头不存在，那么该位置用空串（""）来代替。

3、 使用 Base64 编码

HMAC 请求签名必须使用 Base64 编码。Base64 编码将签名转换为简单的能附加到请求中的 ASCII 编码字符串。加号和斜杠这两个字符不能直接在 URL 中使用，必须经过编码。比如，如果授权编码包括加号，在 URL 中把它编码成为%2B。

4、 时间戳说明

在签名时必须使用时间戳，可以用 HTTP 的 Date 请求头，也可以用 x-amz-date 请求头。时间戳中的时间和 OOS 的系统时间不能相差 15 分钟以上，否则，OOS 会返回错误码

RequestTimeTooSkewed。

有些 HTTP 客户端不能设置 Date 请求头，则可以使用 x-amz-date 请求头来传送时间戳。x-amz-date 请求头需要符合 RFC 2616 的格式(详见 http://www.ietf.org/rfc/rfc2616.txt)。当请求中包含 x-amz-date 头时，OOS 在计算签名时会忽略 Date 头。所以如果请求中有 x-amz-date 头，在客户端计算签名时，Date 头的值应设置为空串。

5、 示例

（1） GET Object：从名为 johnsmith 的 bucket 中 get 对象

请求	StringToSign
GET /photos/puppy.jpg HTTP/1.1 Host: johnsmith.oos-js.ctyunapi.cn Date: Tue, 27 Mar 2007 19:36:42 +0000 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g	GET\n \n \n Tue, 27 Mar 2007 19:36:42 +0000\n /johnsmith/photos/puppy.jpg

注意：CanonicalizedResource 中包含 Bucket 名称，但是 HTTP 请求 URI 中没有，因为 Bucket 名称是在 Host 请求头中指定的。

（2） PUT Object：向名为 johnsmith 的 bucket 中上传一个对象

请求	StringToSign
PUT /photos/puppy.jpg HTTP/1.1 Content-Type: image/jpeg Content-Length: 94328 Host: johnsmith.oos-js.ctyunapi.cn Date: Tue, 27 Mar 2007 21:15:45 +0000 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g	PUT\n \n image/jpeg\n Tue, 27 Mar 2007 21:15:45 +0000\n /johnsmith/photos/puppy.jpg

注意：Content-Type 请求头包含在请求中，也包含在 StringToSign 中。但请求中没有 Content-MD5 请求头，所以 StringToSign 中是空行。

（3） List Objects：list 名为 johnsmith 的 bucket 的对象。

请求	StringToSign
GET /?prefix=photos&max-keys=50&marker=puppy	GET\n

HTTP/1.1 User-Agent: Mozilla/5.0 Host: johnsmith.oos-js.ctyunapi.cn Date: Tue, 27 Mar 2007 19:42:41 +0000 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi 0g	\n \n Tue, 27 Mar 2007 19:42:41 +0000\n /johnsmith/

注意：CanonicalizedResource 的结尾要有斜杠/，查询字符串为空。

（4） 获取 ACL：下面的例子是获取名为 johnsmith 的 bucket 的访问控制权限配置信息。

请求	StringToSign
GET /?acl HTTP/1.1 Host: johnsmith.oos-js.ctyunapi.cn Date: Tue, 27 Mar 2007 19:44:46 +0000 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrO PGi0g	GET\n \n \n Tue, 27 Mar 2007 19:44:46 +0000\n /johnsmith/?acl

注意：在 CanonicalizedResource 中是如何包含子资源查询字符串参数的。

（5） Delete Object：从名为 johnsmith 的 bucket 中删除对象。bucket 在 path 中指定，并使用 x-amz-date 请求头。

请求	StringToSign
DELETE /johnsmith/photos/puppy.jpg HT TP/1.1 User-Agent: dotnet Host: oos-js.ctyunapi.cn Date: Tue, 27 Mar 2007 21:20:27 +0000 x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrO PGi0g	DELETE\n \n \n \n x-amz-date:Tue, 27 Mar 2007 21:20:26 +0000\n /johnsmith/photos/puppy.jpg

注意：此请求使用 x-amz-date 请求头来替代 Date 请求头，在 StringToSign 中，实际的 Date 请求头被设置成空行。

（6） 使用 CNAME 形式上传对象：下面的例子通过 CNAME 形式上传对象，并使用自定义元数据。

请求	StringToSign
PUT /db-backup.dat.gz HTTP/1.1 User-Agent: curl/7.15.5 Host: static.johnsmith.net:8080 Date: Tue, 27 Mar 2007 21:06:08 +0000 x-amz-acl: public-read content-type: application/x-download Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-Amz-Meta-ReviewedBy: joe@johnsmith.net X-Amz-Meta-ReviewedBy: jane@johnsmith.net X-Amz-Meta-FileChecksum: 0x02661779 X-Amz-Meta-ChecksumAlgorithm: crc32 Content-Disposition: attachment; file name=database.dat Content-Encoding: gzip Content-Length: 5913339 Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g	PUT\n 4gJE4saaMU4BqNR0kLY+lw==\n application/x-download\n Tue, 27 Mar 2007 21:06:08 +0000\n x-amz-acl:public-read\n x-amz-meta-checksumalgorithm:crc32\n x-amz-meta-filechecksum:0x02661779\n x-amz-meta-reviewedby: joe@johnsmith.net,jane@johnsmith.net\n /static.johnsmith.net/dbbackup.dat.gz

注意：“x-amz -”请求头被排序了，空白行被删除，转换为小写字符，多个同名的请求头使用逗号分隔的方式被加入。只有 Content-Type, Content-MD5 请求头被加入到了 StringToSign 中，但是其他的 Content-*请求头没有被加入。

List Buckets：

请求	StringToSign
GET / HTTP/1.1 Host: oos-js.ctyunapi.cn Date: Wed, 28 Mar 2007 01:29:59 +0000 *Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g*	GET\n \n \n Wed, 28 Mar 2007 01:29:59 +0000\n /

对象名被编码：

请求	StringToSign
GET /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re HT TP/1.1 Host: oos-js.ctyunapi.cn Date: Wed, 28 Mar 2007 01:49:49 +0000 *Authorization: AWS* *81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrO* *PGi0g*	GET\n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re

StringToSign 中从请求 URI 获取的元素，是按原样获取的，包括 URL 编码和大小写。

3.1.2 URL 中包含签名

除了授权头以外，用户可以通过 URL 中加入签名信息的方式，将该 URL 转给第三方实现授权访问。这对于使用第三方浏览器获取 OOS 数据的用户非常有用，不需要代理请求。这种方式通过构造并加密一个终端用户浏览器可以检索到的预签名的请求来实现，同时设置过期时间来标明预签名的有效期。

URL 中包含签名的示例如下：

```
http://oos-
js.ctyunapi.cn/quotes/nelson?AWSAccessKeyId=44CF9590006BF252F707&Expires=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D
```

URL 中包含签名的示例如下：

在 URL 中实现签名，必须至少包含 Signature，Expires，AWSAccessKeyId 三个参数。请求参数

请求字符串参数名称	示例中的值	描述
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE	可以登录到 OOS 的控制台，在账户管理—API 密钥管理，查看到 AccessKeyId 和 SecretKey
Expires	1141889120	定义里签名过期时间，按照秒来计算的。服务器端超过这个时间的请求将被拒绝

Signature	vjbyPxybdZaNmGa%2ByT272YEAiv4%3D	URL 是按照 HMAC-SHA1 的 StringToSign 的 Base64 编码方式编码

Expires 参数的值是一个 UNIX 时间（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 OOS 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。

所有的 OOS 支持的请求和各种 Head 参数，在 URL 中进行签名的方法和上节介绍的签名算法基本一样。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 Head 中包含签名。
- 如果传入的 Signature，Expires，AWSAccessKeyId 出现不止一次，以第一次为准。
- 请求先验证请求时间是否晚于 Expires 时间，然后再验证签名。

3.2 Bucket 权限控制

OOS 提供 Bucket 级别的权限控制，Bucket 目前有 3 种访问权限：private（私有），public-read（只读）和 public-read-write（公有）。各自含义如下：

- private（私有）

  只有该 Bucket 的所有者可以对该 Bucket 内的 Object 进行读写操作。其他人无法访操作 Bucket 内的 Object。

- public-read（只读）

  只有该 Bucket 的所有者可以对该 Bucket 内的 Object 进行写操作（包括 Put 和 Delete）。任何人（包括匿名访问）可以对该 Bucket 中的 Object 进行读操作（Get Object），这有可能造成您的数据外泄以及费用激增，请慎用该权限。

- public-read-write（公有）

  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。这有可能造成您的数据外泄以及费用激增，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。

注意：如果想使用访问权限为公有的容器，请联系天翼云客服协助开通此功能。

说明：新建 Bucket 时，默认权限是 private，用户可以根据需要修改为其他权限。

3.3 Bucket Policy 安全策略

3.3.1 介绍

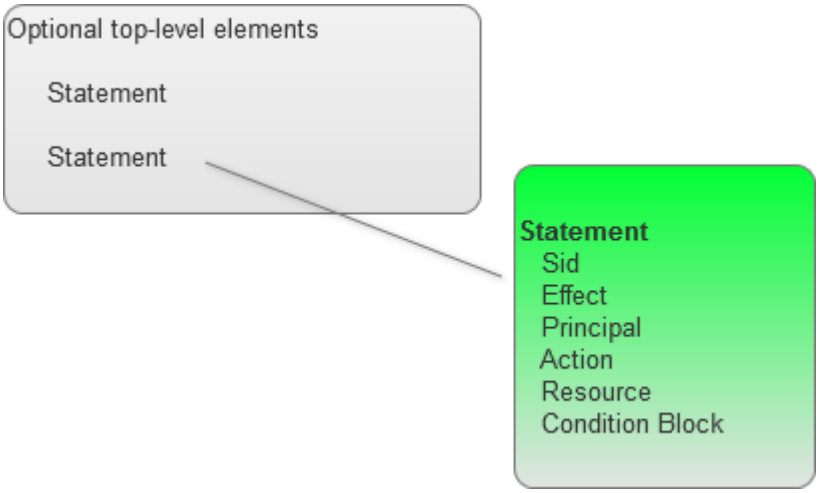
Bucket policy 用于定义 OOS 资源的访问权限。Policy 可以用于：

- 允许/拒绝 Bucket 级别的权限；
- 设置 Bucket 中任何 Object 的拒绝权限；
- 为 Bucket 中的 Object 授权。

Policy 本身是一个 JSON 字符串，一个 Policy 包括：

- 可选的 Policy 基本信息 (在文档顶部)；
- 一个或多个独立的 Statements。

每个 statement 包括一个权限的核心信息。如果一个 Policy 包括多个 Statements,那么 Statements 之间是逻辑或关系。



3.3.2 Bucket Policy 元素

- **Version**

Version 是 policy 语言的版本，它是个可选项，目前只允许填写 2012-10-17。

```
"Version":"2012-10-17"
```

- **ID**

  ID 是 Policy 的一个可选标识，我们推荐使用 UUID 来指明 Policy。

- **Statement**

  可以包含多个元素。Statement 元素可以包含一组独立的 statements，每个独立的 Statement 是一个包含在大括号（{}）内的严格的 JSON 块。

```
"Statement":[{...},{...},{...}]
```

  - **Sid**

    Sid（statement ID）是 Statement 的一个可选标识，它可以看作是 Policy ID 的子 ID。

```
"Sid" : "1"
```

  - **Effect**

    Effect 是一个必填元素，用于表示允许访问或拒绝访问，有效值只能是 Allow 或 Deny。

```
"Effect":"Allow"
```

  - **Principal**

    Principal 用于指定被允许或被拒绝访问的用户，在本版本中，principal 只能是 `AWS:*` 或 `AWS:"*"` 或 `AWS:["*"]`，表示所有用户，暂不支持设置某个用户 ID。

  - **Action**

    Action 用于指定被允许或拒绝访问的操作，必填项。只有 Bucke owner 能执行 Bucket 相关写操作，以及 Bucket 子资源的相关操作。Action 中可以配置以下权限。

与 Bucket 相关的 OOS 权限列表：

权限关键字	OOS 操作
`s3:ListBucket`	GET Bucket（列出对象），HEAD Bucket
`s3:ListBucketMultipartUploads`	列出分段上传

对象操作的 OOS 权限列表

权限	OOS 操作

s3:AbortMultipartUpload	中止分段上传
s3:DeleteObject	DELETE Object
s3:GetObject	GET Object、HEAD Object
s3:ListMultipartUploadParts	列出分段
s3:PutObject	PUT Object、POST Object、启动分段上传、上传分段、完成分段上传、PUT Object - Copy

可使用通配符 (*) 来访问 OOS 产品提供的所有操作。还可以使用通配符 (*) 作为操作名称的一部分。

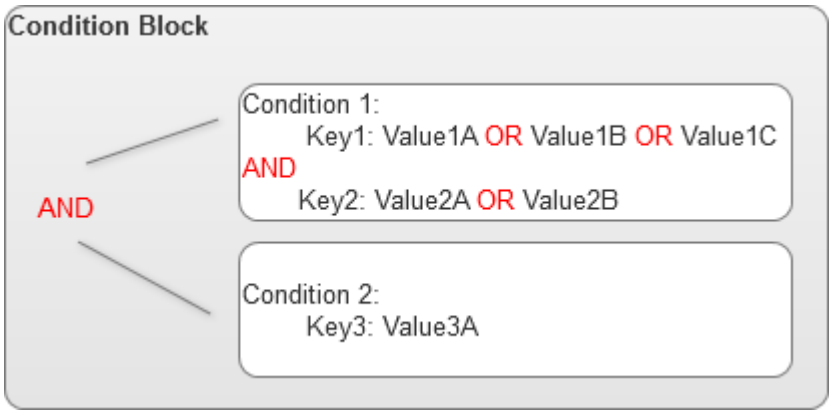
- **Resource**

Resource 是指 policy 覆盖的对象或对象集合。可以使用通配符*和？，必须以 arn:aws:s3:::bucketName/开头。例如要使 Policy 对于所有以 image 开头的 Object 生效，可以这样配置：

```
"Resource":"arn:aws:s3:::example-bucket/image*"
```

- **Condition**

Condition 是 Statement 中最复杂的一个元素，我们把它称之为 Condition 块，虽然只有一个 Condition 元素，但是它可以包含多个 Conditions，而且每个 Condition 可以包含多个 Key-value 对。如下图所示，Condition 块中的各个 Condition 之间是逻辑与关系，Condition Key 之间也是逻辑与关系，各个 value 之间是逻辑或关系。



- 条件运算符

a. String conditions 可以允许设置 string 的匹配规则

Condition	描述
StringEquals	严格匹配
StringNotEquals	严格不匹配
StringEqualsIgnoreCase	严格匹配，忽略大小写
StringNotEqualsIgnoreCase	严格不匹配，忽略大小写
StringLike	模糊匹配，支持使用通配符，*表示多个字符，？表示单个字符。
StringNotLike	模糊不匹配，支持使用通配符，*表示多个字符，？表示单个字符。

例如，要匹配 HTTP Referer 头是以“http://www.ctyun.cn/”开头的请求，可以这样配置：

```
"StringLike":{
         "aws:Referer":[
           "http://www.ctyun.cn/*"
         ]
       }
```

b. 布尔值条件运算符

利用布尔值条件，用户可以通过与“正确”或“错误”的对比，来设置 Condition 元素。

条件运算符	说明
Bool	布尔值匹配

例如：下列声明使用 Bool 条件运算符与 aws:SecureTransport 键来指定必须使用 SSL 发出请求。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:PutObject",
    "Resource": " arn:aws:s3:::example_bucket ",
    "Condition": {"Bool": {"aws:SecureTransport": "true"}}
  }
}
```

c. IP Address

IP address conditions 允许设置 IP 地址的匹配规则，与 aws:SourceIp key 配合使用。此项的值必须符合标准的 CIDR 格式（例如：10.52.176.0/24），参考 RFC 4632。

Condition	描述
IpAddress	IP 地址或范围的白名单。
NotIpAddress	IP 地址或范围的黑名单。

- 条件键
  - aws:Referer：与字符串运算符结合使用，用于检查请求中的 Referer 请求头。
  - aws:SecureTransport：与布尔值运算符结合使用，检查请求是否是使用 SSL 发送的。
  - aws:SourceIp：与 IP 地址运算符结合使用，用于检查请求者的 IP 地址。
  - aws:UserAgent：与字符串运算符结合使用，用于检查请求者的客户端应用程序。

3.3.3 示例

- **Referer Policy**

```
{
  "Version":"2012-10-17",
  "Id":"http referer policy example",
  "Statement":[
    {
      "Sid":"Allow get requests referred by www.ctyun.cn, ctyun.cn",
      "Effect":"Allow",
      "Principal":{ "AWS": ["*"] },
      "Action":"s3:*",
      "Resource":"arn:aws:s3:::example-bucket/*",
      "Condition":{
        "StringLike":{
          "aws:Referer":[
            "https://www.ctyun.cn/*",
            "https://ctyun.cn/*"
```

```
        ]
      }
    }
  }
 ]
}
```

- IP Policy

```
{
    "Version": "2012-10-17",
    "Id": "S3PolicyId1",
    "Statement": [
       {
           "Sid": "IPAllow",
           "Effect": "Allow",
           "Principal": {
               "AWS": "*"
           },
           "Action": "s3:*",
           "Resource": "arn:aws:s3:::example-bucket /*",
           "Condition" : {
               "IpAddress" : {
                   "aws:SourceIp": "192.168.143.0/24"
               },
               "NotIpAddress" : {
                   "aws:SourceIp": "192.168.143.188/32"
               }
           }
       }
    ]
}
```

- 向匿名用户授予只读权限

该策略向任何公用匿名用户授予 s3:GetObject 权限。此权限允许任何人读取对象数据，当用户将 Bucket 配置为网站并且希望每个人都能读取存储桶中的对象时，此配置十分有用。可以将 Bucket 设置为私有，然后配置以下 Bucket 策略授予任何人读取数据的权限。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
```

```
      "Sid":"AddPerm",
      "Effect":"Allow",
      "Principal":{ "AWS": ["*"] },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example-bucket/*"]
    }
  ]
}
```

4 HTTP REST 接口

本章主要介绍 OOS 对外开放的接口，当用户发送请求给 OOS 时，可以通过签名认证的方式请求，也可以匿名访问。

4.1 OOS API 请求结构

- **接入地址**

  接入地址详见 **Endpoint 列表**。

- **通信协议**

  为了保证通信的安全性，统计支持 HTTP 和 HTTPS。

- **请求方法**

  OOS API 支持 GET、POST、PUT、HEAD、DELETE 和 OPTION 请求方法发送请求。

- **请求参数**

  每个请求都需要指定如下信息：

  - 每个操作接口都需要包含的公共请求参数。
  - 操作接口所特有的请求参数。

  本节主要描述公共请求参数和请求结果。

说明：在后续提到具体 API 时，举例中都会有**公共请求头**、**公共响应头**，但是不对其进行描述和解释。

4.1.1 公共请求头

以下是 OOS API 的公共请求头。

名称	描述	是否必须
Authorization	用于身份验证的请求头。	是
Content-Length	请求体的长度。	否

Content-MD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值。此请求头可以用作数据完整性检查，以验证数据是否与客户端发送的数据相同。	否
Content-Type	描述请求内容的标准 MIME 类型。	否
Date	请求的日期和时间，GMT 时间。 **注意**：Date 和 x-amz-date 必须填写一个。	否
x-amz-date	请求的日期和时间。日期和时间格式必须遵循 ISO 8601 标准，并且必须使用“yyyyMMddT HHmmssZ”格式进行格式化。例如，如果日期和时间是“08/01/2018 15：32：41.982-700”，则必须首先将其转换为 UTC（协调世界时），然后提交为“20180801T083241Z”。 **注意**：Date 和 x-amz-date 必须填写一个。	否
Host	请求的域名，详见错误!未找到引用源。。	是
Connection	客户端与 OOS 服务器之间的连接状态。 取值： ● keep-alive：长连接，请求结束后继续保持连接； ● close：短连接，请求结束后关闭连接。 默认值为：close。	否

4.1.2 公共响应头

每个 OOS API 响应结果中都包含公共响应头。

名称	描述
Content-Length	响应体的长度。
Content-Type	响应内容的 MIME 类型。
Date	OOS 返回响应的日期和时间。
ETag	对象的哈希值。ETag 只反映了对象的内容，而不是其元数据。此响应头可以用作数据完整性检查，以验证数据是否与客户端发送的数据相

	同。
Server	服务端名称，默认值是 CTYUN。
x-amz-request-id	用于唯一标示请求的 ID。
Connection	客户端与 OOS 服务器之间的连接状态。 ● 如果请求时 Connection 值为 keep-alive，请求结束后继续保持连接，不返回此响应头。 ● 如果请求时 Connection 值为 close，请求结束后关闭连接，返回此响应头 Connection: close。

4.2 关于 Service 的操作

4.2.1 GET Service (List Bucket)

对于做 Get 请求的服务，返回请求者拥有的所有 Bucket，其中“/”表示根目录。

该 API 只对验证用户有效，匿名用户不能执行该操作。

- **请求语法**

```
GET / HTTP/1.1
Host: oos-js.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

- **请求参数**

  无

- **响应结果**

名称	描述
Bucket	存储 bucket 信息的容器。
Buckets	存储一个或多个 Bucket 的容器。
CreationDate	Bucket 的创建日期。
DisplayName	Bucket 拥有者的用户显示姓名。
ID	Bucket 拥有者的用户 ID。
Name	Bucket 的名称。
Owner	Bucket 的拥有者的信息。

- **请求示例**

```
GET / HTTP/1.1
Host: oos-js.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

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

```
<ListAllMyBucketsResult xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Owner>
    <ID>bcaf1ffd86f461ca5fb16fd081034f</ID>
    <DisplayName>displayName</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>sample1</Name>
      <CreationDate>2012-09-01T16:45:09.000Z</CreationDate>
    </Bucket>
    <Bucket>
      <Name>sample2</Name>
      <CreationDate>2012-09-01T16:41:58.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

4.3 关于 Bucket 的操作

4.3.1 PUT Bucket

此操作用来创建一个新的 Bucket，或者对已有 Bucket 的 Bucket ACL 进行修改。

只有 OOS 注册用户才能创建一个新的 Bucekt，匿名请求无效，创建 Bucket 的用户将是 Bucket 的拥有者。

Bucket 的命名方式中并不是支持所有的字符，具体请参见 **Bucket 的命名规范**。

- **请求语法**

```
PUT / HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Content-Length: length
Date: date
Authorization: signatureValue
```

- **请求参数**

名称	描述	是否必须
x-amz-acl	设置 Bucket 的 ACL（Access Control List）。 **类型**：字符串。 **有效值**： ● private：私有。只有该 Bucket 的所有者可以对该 Bucket 内的 Object 进行读写操作。其他人无法操作该 Bucket 内的 Object。 ● public-read：只读。只有该 Bucket 的所有者可以对该 Bucket 内的 Object 进行写操作（包括 Put 和 Delete）。任何人（包括匿名访问）可以对该 Bucket 中的 Object 进行读操作（Get Object），这有可能造成您的数据外泄以及费用激增，请慎用该权限。	否

	• public-read-write：公有。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。这有可能造成您的数据外泄以及费用激增，若被人恶意写入违法信息，还可能侵害您的合法权益，除特殊场景外，不建议您配置公共读写权限。 **注意**：如果想使用访问权限为公有的 Bucket，请联系天翼云客服协助开通此功能。 默认值为 private。	

- **请求示例 1**

请求创建名为 picture 的 Bucket。

```
PUT / HTTP/1.1
Host: picture.oos-js.ctyunapi.cn
Content-Length: 0
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例 1**

```
HTTP/1.1 200 OK
x-amz-request-id: 236A8905248E5A01
Date: Mon, 03Sep 2012 12:00:00 GMT
Location: /picture
Content-Length: 0
Connection: close
Server: CTYUN
```

- **请求示例 2**

设置 Bucket picture 的 ACL 为 private。

```
PUT / HTTP/1.1
Host: picture.oos-js.ctyunapi.cn
Content-Length: 0
x-amz-acl: private
```

```
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- 响应示例 2

```
HTTP/1.1 200 OK
x-amz-request-id: 236A8905248E5A01
Date: Mon, 03Sep 2012 12:00:00 GMT
Location: /picture
Content-Length: 0
Connection: close
Server: CTYUN
```

4.3.2 GET Bucket ACL

该操作用来获取 Bucket 的 ACL 信息，用户必须对该 Bucket 拥有读权限。

- **请求语法**

```
GET /?acl HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应结果**

名称	描述
DisplayName	Bucket 拥有者的显示名称。
Grant	存储 Permission 和 Grantee 的容器。
Grantee	用来存储 Display 和拥有 ID 的用户被承认的许可的容器。
ID	Bucket 拥有者的 ID 信息。
Owner	存储 Bucket 的拥有者信息的容器。
Permission	对一个 Bucket 认可的许可信息。

- **请求示例**

获取指定 Bucket 的 ACL 信息

```
GET ?acl HTTP/1.1
Host: bucket.oos-js.ctyunapi.cn
Date: Mon, 03 Sep 2012 22:32:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 318BC8BC148832E5
Date: Mon, 03 Sep 2012 22:32:00 GMT
Last-Modified: Sat, 01 Sep 2012 12:00:00 GMT
```

```
Content-Length: 124
Content-Type: text/plain
Connection: close
Server: CTYUN

<AccessControlPolicy>
  <Owner>
    <ID>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</I D>
    <DisplayName>CustomersName@ctyun.cn</DisplayName>
  </Owner>
  <AccessControlList>
     <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
       <URI>http://acs.amazonaws.com/groups/global/AllUser</URI>
       <Permission>FULL_CONTROL</Permission>
     </Grantee>
  </AccessControlList>
</AccessControlPolicy>
```

4.3.3 GET Bucket (List Objects)

此操作返回 Bucket 中部分或者全部（最多 1000）的 Object 信息。用户可以在请求元素中设置选择条件来获取 Bucket 中的 Object 子集。

对该 Bucket 拥有读权限的用户才可以执行该操作。

- **请求语法**

```
GET / HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求参数**

名称	描述	是否必须
BucketName	存储桶名称。 类型：字符串。	是
delimiter	分隔符，一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。delimiter 只支持”/”，不支持其他分隔符。 类型：字符串。	否
encoding-type	指定响应中 Delimiter、Marker、Prefix、NextMarker、Key 的编码类型。如果 Delimiter、Marker、Prefix、NextMarker、Key 包含 xml 1.0 标准不支持的控制字符，可通过设置该参数对响应中的 Delimiter、Marker、Prefix、NextMarker、Key 进行编码。 取值：url，字母不区分大小写。	否

marker	分页标识。还有需要返回的用户时，上条响应结果中会返回该参数。查看未显示项时，请求参数中需要携带此参数。 类型：字符串。 取值：与上条响应中返回的结果值相同。	否
max-keys	设置响应中最多返回的条数。如果存在超出您指定的返回项，则响应元 IsTruncated 为 true，表示需要再次发送请求查看未显示的项。查看未显示的项时，需要携带响应参数 Marker 的值。 类型：整型。 取值：1~1000，默认值为 100。	否
prefix	前缀用来限制返回的结果必须以这个前缀开始，可以通过前缀将 bucket 分为若干个组。	否

- **响应结果**

名称	描述
Contents	每个 Object 返回的元数据。
CommonPrefixes	当定义 delimiter 之后，返回结果中会包含 CommonPrefixes，CommonPrefix 中包含以 prefix 开头，delimiter 结束的字符串组合，比如 prefix 是 note/,同时 delimiter 是斜杠（/），结果中的 note/summer/july/lotus.jpg 将返回 note/summer/，其余结果将按照 maxkey 要求返回。
Delimiter	将 prefix 和第一次出现 delimiter 的所有查询结果滚动的存入 CommonPrefix 组中。
ETag	通过 MD5 的方式计算出标签，ETag 主要用来反映对象内容的信息发生了改变，并不反映元数据的变化。
IsTruncated	通过是（True）或否（false）来表示返回的结果是否为所有要求的结果。
Key	对象的关键字。

LastModified	记录的对象最后一个被修改的日期和时间。
Marker	标记从哪个位置开始罗列出 bucket 中的 object。
Name	Bucket 的名称。
Prefix	关键字以特定的前缀开始。
EncodingType	Delimiter、Marker、Prefix、NextMarker、Key 的编码类型。
Size	记录对象 object 的大小。
StorageClass	STANDARD。
NextMarker	若 IsTruncated 返回 true，用户可以将 NextMarker 的值设置为下次请求中的 marker 参数，以便获取后续对象。

- **请求示例**

```
GET / HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 17:50:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Type: text/plain
```

- **响应示例**

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>bucket</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <EncodingType></EncodingType>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>my-image.jpg</Key>
    <LastModified>2012-09-03T17:50:30.000Z</LastModified>
    <ETag>&quot;fba9dede5f27731c9771645a39863328&quot;</ETag>
    <Size>434234</Size>
    <StorageClass>STANDARD</StorageClass>
```

```
    <Owner>
        <ID></ID>
        <DisplayName></DisplayName>
    </Owner>
    </Contents>
    <Contents>
    <Key>my-third-image.jpg</Key>
    <LastModified>2012-09-03 T17:50:30.000Z</LastModified>
    <ETag>&quot;1b2cf535f27731c974343645a3985328&
    <Size>64994</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
        <ID></ID>
        <DisplayName></DisplayName>
    </Owner>
    </Contents>
</ListBucketResult>
```

使用请求变量的示例：

假设数据库中存储数据如下所示：

```
doc/Sample.pdf
doc/Ant/sample.doc
doc/Feb/sample2.doc
doc/Feb/sample3.doc
doc/Feb/sample4.doc
```

要查询 prefix 为 doc/,delimiter 为/，同时 marker 为 doc/F，max-key 为 40 的结果，请求头为：

```
GET ?prefix=doc/&marker=doc/F&max-keys=40&delimiter=/ HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

返回的结果集为

```
<ListBucketResult xmlns="http://oos-js.ctyunapi.cn/doc/2012-09-03/">
```

```
  <Name>doc</Name>
  <Prefix>doc/</Prefix>
  <Marker>doc/F</Marker>
  <MaxKeys>40</MaxKeys>
  <EncodingType></EncodingType>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>sample.pdf</Key>
    <LastModified>2012-09-01T01:56:20.000Z</LastModified>
    <ETag>&amp;quot;bf1d737a4d46a19f3bced6905cc8b902&amp;quot;</ETag>
    <Size>142863</Size>
    <Owner>
      <ID></ID>
      <DisplayName></DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>Feb/</Prefix>
    <Prefix>Ant/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

4.3.4 DELETE Bucket

该操作用来执行删除 Bucket 的操作，但要求所有 bucket 中的 object 都必须被删除。

- **请求语法**

```
DELETE / HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

删除名为 doc 的 Bucket：

```
DELETE / HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 204 No Content
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.5 PUT Bucket Policy

在 PUT 操作的 url 中加上 Policy，可以进行添加或修改 Policy 的操作。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

注意：如果 Bucket 的属性为私有或者只读，使用该接口配置允许任何用户可以向该 Bucket 写对象的策略时，需要联系天翼云客服进行备案。

- **请求语法**

请求的内容是一个包含 Policy 语句的 JSON 串

```
PUT /?policy HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue

Policy written in JSON
```

- **请求示例**

```
PUT /?policy HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

{
"Version":"2012-10-17",
"Id":"aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect":"Allow",
        "Sid":"1",
        "Principal" : {
        "AWS":"*"
        },
```

```
        "Action":["s3:*"],
        "Resource":"arn:aws:s3:::bucket/*",
    }
 ]
}
```

- **响应示例**

```
HTTP/1.1 204 No Content
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.6 GET Bucket Policy

在 GET 操作的 url 中加上 Policy，可以获得指定 Bucket Policy。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 bucket 没有 policy，返回 404 NoSuchPolicy 错误。

- **请求语法**

```
GET /?policy HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
GET /?policy HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN

{
"Version":"2012-10-17",
"Id":"aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect":"Allow",
        "Sid":"1",
        "Principal" : {
        "AWS":"*"
```

```
        },
    "Action":["s3:*"],
    "Resource":"arn:aws:s3:::bucket/*",
    }
 ]
}
```

4.3.7 DELETE Bucket Policy

在 DELETE 操作的 url 中加上 Policy，可以删除指定 Bucket Policy。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果没有 Bucket Policy，返回 204 NoContent。

- **请求语法**

```
DELETE /?policy HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
DELETE /?policy HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 204 No Content
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.8 PUT Bucket WebSite

此操作用来配置网站托管属性。如果 Bucket 已经存在了 website，此操作会替换原有 website。

注意:

- OOS 自有网站托管域名不支持 HTTPS 访问，用户自定义域名支持 HTTPS 访问。
- 如果配置静态网站托管后，当匿名用户直接访问 Bucket 的域名，会将静态网站文件下载到本地。如果您需要实现访问静态网站时，是预览网站内容而非下载静态网站文件，您需要为桶绑定已通过备案的自定义域名，请联系天翼云客服申请绑定自定义域名。
- 设置 Bucket 的网络配置请求消息体的上限是 10KiB。
- 尽量避免目标 Bucket 名中带有“.”，否则通过 HTTPS 访问时可能出现客户端校验证书出错。

网站托管配置步骤如下：

1) 创建一个只读属性的对象容器（Bucket）。

2) 向天翼云客服提交工单，申请客户自定义域名添加白名单。

3) 在域名管理中添加别名。

- 如果不使用 CDN 加速，将 Bucket 的 CNAME Record Value（*bucketname*.oos-website-cn.oos-*xx*.ctyunapi.cn）作为别名添加到域名管理系统中。
- 如果使用 CDN 加速，将 CDN 厂商提供的别名添加到域名管理系统中，然后在 CDN 回源地址中配置 OOS 侧的 CNAME Record Value，并将回源 host 配置为您的自定义域名（如 *your***domain*.com）。

4) 上传文件

将网站的所有文件（html、CSS、js、图片等）上传到之前创建的 Bucket 中， 注意要保持文件之间的相对路径。

5) 配置 Bucket 网站属性：可以通过控制台或者调用本接口配置。

- **请求语法**

托管模式为当前容器

```
PUT /?website HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Content-Length: ContentLength
Authorization: signatureValue

<WebsiteConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
      <IndexDocument>
             <Suffix>index.html</Suffix>
       </IndexDocument>
       <ErrorDocument>
             <Key>errorDocument.html</Key>
        </ErrorDocument>
        <RoutingRules>
             <RoutingRule>
                  <Condition>
                    <HttpErrorCodeReturnedEquals>string</HttpErrorCodeReturnedEquals>
                    <KeyPrefixEquals>string</KeyPrefixEquals>
                  </Condition
                  <Redirect>
                     <HostName>string</HostName>
                     <Protocol>string</Protocol>
                     <ReplaceKeyPrefixWith>string</ReplaceKeyPrefixWith>
                     <ReplaceKeyWith>string</ReplaceKeyWith>
                  </Redirect>
            </RoutingRule>
    </RoutingRules>
</WebsiteConfiguration>
```

托管模式为重定向请求

```
PUT /?website HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Content-Length: ContentLength
Authorization: signatureValue
```

```
<WebsiteConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
      <RedirectAllRequestsTo>
            <HostName>string</HostName>
            <Protocol>string</Protocol>
      </RedirectAllRequestsTo>
</WebsiteConfiguration>
```

- **请求元素**

名称	描述	是否必须
WebsiteConfiguration	请求的容器。 类型：容器 子节点：IndexDocument、ErrorDocument、RoutingRules 或 RedirectAllRequestsTo	是
IndexDocument	Suffix 元素的容器。 类型：容器 父节点：WebsiteConfiguration 子节点：Suffix	是
Suffix	在请求 website endpoint 上的路径时，Suffix 会被加在请求的后面。例如，如果 suffix 是 Index.html，而你请求的是 bucket/images/，那么返回的响应是名为 images/index.html 的 Object。 类型：字符串 父节点：IndexDocument	是
ErrorDocument	Key 的容器。 类型：容器 父节点：WebsiteConfiguration 子节点：Key	否
Key	如果出现 4XX 错误，会返回指定的 Object。	是

	类型：字符串 有效值：长度为 1~1024 的字符串。 父节点：ErrorDocument	
RoutingRules	托管模式配置到当前容器的重定向规则容器。 **注意**：如果 RoutingRules 下有多个 RoutingRule，各 RoutingRule 之间无影响，按照配置的先后顺序向下执行，当有一个满足条件时，就不再继续向后匹配。如果都没有匹配，就不使用重定向规则。 类型：容器 父节点：WebsiteConfiguration 子节点：RoutingRule	否
RoutingRule	重定向规则的容器。一条重定向规则包含一个 Condition 和一个 Redirect，当 Condition 匹配时，Redirect 生效。容器中至少要有一个重定向规则。 **注意**：一个 RoutingRule 下，出现多个 Condition 和 Redirect 时，以最后一个为准。 类型：容器 父节点：RoutingRules 子节点：Condition、Redirect	是
Condition	描述重定向规则匹配的条件的容器。如果重定向规则匹配的条件未配置，则重定向规则将应用于所有请求。 **注意**：该容器可以不配置，如果配置，则至少应该包含 HttpErrorCodeReturnedEquals、KeyPrefixEquals 中的一个。 类型：容器	否

	父节点：RoutingRule 子节点：HttpErrorCodeReturnedEquals、KeyPrefixEquals	
HttpErrorCodeReturnedEquals	指定 Redirect 生效时的 HTTP 错误码。当发生错误时，如果错误码等于这个值，那么 Redirect 生效。 **注意**：HttpErrorCodeReturnedEquals 和 KeyPrefixEquals 同时存在时，只有都匹配时，Redirect 才生效。 类型：字符串 有效值：[400, 417]，[500, 505]。 例如：当返回的 http 错误码为 404 时重定向到 NotFound.html，可以将 Condition 中的 HttpErrorCodeReturnedEquals 设置为 404，Redirect 中的 ReplaceKeyWith 设置为 NotFound.html。 父节点：Condition	否
KeyPrefixEquals	重定向规则生效时的对象名的前缀。 **注意**：HttpErrorCodeReturnedEquals 和 KeyPrefixEquals 同时存在时，只有都匹配时，Redirect 才生效。 类型：字符串 有效值：长度为 0-1024 的字符串。 父节点：Condition	否
Redirect	重定信息容器。 **注意**：Redirect 配置包含的元素可以为空，也可以包含以下元素:Protocol、HostName、	是

	ReplaceKeyPrefixWith、ReplaceKeyWith。当某一元素存在多条值时以最后一条为准。 类型：容器 父节点：RoutingRule 子节点：Protocol、HostName、ReplaceKeyPrefixWith、ReplaceKeyWith	
Protocol	重定向请求时使用的协议。 类型：字符串 有效值：http、https，默认值为 http。 父节点：Redirect 或者 RedirectAllRequestsTo	否
HostName	重定向请求时使用的站点名。 如果父节点为 RedirectAllRequestsTo，此项必须填写。 类型：字符串 有效值：1~1024 个字符，不能包含空格。父节点为 Redirect，也不能包含斜杠(/)。 父节点：Redirect 或者 RedirectAllRequestsTo	否
ReplaceKeyPrefixWith	重定向请求时使用的对象名前缀。 **注意**：ReplaceKeyPrefixWith 与 ReplaceKeyWith 不能同时存在。 类型：字符串 有效值：0~1024 个字符。 父节点：Redirect	否
ReplaceKeyWith	指定重定向请求时使用的对象名。 **注意**：ReplaceKeyPrefixWith 与 ReplaceKeyWith 不能同时存在。 类型：字符串 有效值：0~1024 个字符。	否

	父节点：Redirect	
RedirectAllRequestsTo	托管模式为重定向请求的容器。 父节点：WebsiteConfiguration 子节点：HostName、Protocol	否

- **请求示例 1**

  指定 index.html 为首页，error.html 为错误页。

```
PUT /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 08:40:33 GMT
Host: example.oos-js.ctyunapi.cn
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 268

<WebsiteConfiguration xmlns='http://s3.amazonaws.com/doc/2006-03-01/'>
       <IndexDocument>
              <Suffix>index.html</Suffix>
       </IndexDocument>
       <ErrorDocument>
              <Key>error.html</Key>
       </ErrorDocument>
</WebsiteConfiguration>
```

- **响应示例 1**

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2022 08:40:33 GMT
x-amz-request-id: 91dd8b753eba4237
Content-Length: 0
Server: CTYUN
```

- **请求示例 2**

  将所有的请求重定向到主机 ctyun.cn，且使用 https 协议。

```
PUT /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 08:45:39 GMT
Host: www.example-bucket.com.oos-js.ctyunapi.cn
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 252

<WebsiteConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
       <RedirectAllRequestsTo>
            <HostName>ctyun.cn</HostName>
            <Protocol>https</Protocol>
      </RedirectAllRequestsTo>
</WebsiteConfiguration>
```

- **响应示例 2**

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2022 08:45:39 GMT
x-amz-request-id: 98449be04a8e4abb
Content-Length: 0
Server: CTYUN
```

- **请求示例 3**

  指定 index.html 为首页，error.html 为错误页。将所有前缀为“docs”的请求，全部重定向至本 Bucket 中前缀为“documents/”的对象上。

```
PUT /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 08:51:46 GMT
Host: www.example.com.oos-js.ctyunapi.cn
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 471

<WebsiteConfiguration xmlns='http://s3.amazonaws.com/doc/2006-03-01/'>
  <IndexDocument>
```

```
    <Suffix>index.html</Suffix>
  </IndexDocument>
  <ErrorDocument>
    <Key>error.html</Key>
  </ErrorDocument>

  <RoutingRules>
    <RoutingRule>
    <Condition>
      <KeyPrefixEquals>docs</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyPrefixWith>documents/</ReplaceKeyPrefixWith>
    </Redirect>
    </RoutingRule>
  </RoutingRules>
</WebsiteConfiguration>
```

- **响应示例 3**

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2022 08:51:46 GMT
x-amz-request-id: 01413e0eca2e4cbf
Content-Length: 0
Server: CTYUN
```

- **请求示例 4**

  指定 index.html 为首页，error.html 为错误页。如果发生 404 错误，重定向到主机“ctyun,cn”上前缀为“report-404/”的对象。

```
PUT /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 09:07:39 GMT
Host: www.example.com.oos-js.ctyunapi.cn
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 538
```

```
<WebsiteConfiguration xmlns='http://s3.amazonaws.com/doc/2006-03-01/'>
  <IndexDocument>
    <Suffix>index.html</Suffix>
  </IndexDocument>
  <ErrorDocument>
    <Key>error.html</Key>
  </ErrorDocument>
  <RoutingRules>
    <RoutingRule>
    <Condition>
      <HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals>
    </Condition>
    <Redirect>
       <HostName>www.ctyun.cn</HostName>
      <ReplaceKeyPrefixWith>report-404/</ReplaceKeyPrefixWith>
    </Redirect>
    </RoutingRule>
  </RoutingRules>
</WebsiteConfiguration>
```

- **响应示例 4**

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2022 09:07:39 GMT
x-amz-request-id: 2814d402a9f64041
Content-Length: 0
Server: CTYUN
```

- **请求示例 5**

  指定 index.html 为首页，error.html 为错误页。将前缀为“images/”的请求，重定向至本 Bucket 内的“errorpage.html”对象。

```
PUT /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 09:11:38 GMT
Host: test-bucket.oos-js.ctyunapi.cn
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 465

<WebsiteConfiguration xmlns='http://s3.amazonaws.com/doc/2006-03-01/'>
  <IndexDocument>
    <Suffix>index.html</Suffix>
  </IndexDocument>
  <ErrorDocument>
    <Key>error.html</Key>
  </ErrorDocument>
  <RoutingRules>
    <RoutingRule>
    <Condition>
      <KeyPrefixEquals>images/</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyWith>errorpage.html</ReplaceKeyWith>
    </Redirect>
    </RoutingRule>
  </RoutingRules>
</WebsiteConfiguration>
```

- **响应示例 5**

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2022 09:11:38 GMT
x-amz-request-id: 2ef16ade344d4c11
Content-Length: 0
Server: CTYUN
```

4.3.9 GET Bucket WebSite

此操作用来获得指定 Bucket 的 website。

- **请求语法**

```
GET /?website HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求参数**

名称	描述	是否必须
BucketName	存储桶名称。	是

- **响应结果**

名称	描述
WebsiteConfiguration	请求的容器。 类型：容器 子节点：IndexDocument、ErrorDocument、RoutingRules 或 RedirectAllRequestsTo
IndexDocument	Suffix 元素的容器。 类型：容器 父节点：WebsiteConfiguration 子节点：Suffix
Suffix	在请求 website endpoint 时，Suffix 会被加在请求的后面。 类型：字符串 父节点：IndexDocument
ErrorDocument	Key 的容器。 类型：容器

	父节点：WebsiteConfiguration 子节点： Key
Key	如果出现 4XX 错误，返回的 Object。 类型：字符串 父节点：ErrorDocument
RoutingRules	托管模式配置到当前容器的重定向规则容器。 类型：容器 父节点：WebsiteConfiguration 子节点：RoutingRule
RoutingRule	具体重定向规则的容器。 父节点：RoutingRules 子节点：Condition、Redirect
Condition	描述重定向规则匹配的条件的容器。 类型：容器 父节点：RoutingRule 子节点：HttpErrorCodeReturnedEquals、KeyPrefixEquals
HttpErrorCodeReturnedEquals	Redirect 生效时的 HTTP 错误码。 类型：字符串 父节点：Condition
KeyPrefixEquals	重定向规则生效时的对象名的前缀。 类型：字符串 父节点：Condition
Redirect	重定信息容器。 类型：容器 父节点：RoutingRule 子节点：Protocol、HostName、ReplaceKeyPrefixWith、ReplaceKeyWith

Protocol	描述重定向请求时使用的协议。 类型：字符串 父节点：Redirect 或者 RedirectAllRequestsTo
HostName	重定向请求时使用的站点名。 类型：字符串 父节点：Redirect 或者 RedirectAllRequestsTo
ReplaceKeyPrefixWith	重定向请求时使用的对象名前缀。 类型：字符串 父节点：Redirect
ReplaceKeyWith	重定向请求时使用的对象名。 类型：字符串 父节点：Redirect
RedirectAllRequestsTo	托管模式为重定向请求的容器。 父节点：WebsiteConfiguration 子节点：HostName、Protocol

- **请求示例**

```
GET /?website HTTP/1.1
Authorization: Authorization
Date: Mon, 18 Jul 2022 09:15:30 GMT
Host: test-bucket.oos-js.ctyunapi.cn
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

- **响应示例**

```
HTTP/1.1 200 OK
Content-Type: application/xml;charset=UTF-8
Date: Mon, 18 Jul 2022 09:15:30 GMT
x-amz-request-id: 88d52b33bef4440b
Content-Length: 430
Server: CTYUN
```

```
<?xml version="1.0" encoding="UTF-8"?>
<WebsiteConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <IndexDocument>
    <Suffix>index.html</Suffix>
  </IndexDocument>
  <ErrorDocument>
    <Key>error.html</Key>
  </ErrorDocument>
  <RoutingRules>
    <RoutingRule>
      <Condition>
        <KeyPrefixEquals>images/</KeyPrefixEquals>
      </Condition>
      <Redirect>
        <ReplaceKeyWith>errorpage.html</ReplaceKeyWith>
      </Redirect>
    </RoutingRule>
  </RoutingRules>
</WebsiteConfiguration>
```

4.3.10 DELETE Bucket WebSite

在 DELETE 操作的 url 中加上 website，可以删除指定 Bucket website。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 bucket 没有 website，返回 200 OK。

- **请求语法**

```
DELETE /?website HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
DELETE /?website HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.11 List Multipart Uploads

该接口用于列出所有已经通过 Initiate Multipart Upload 请求初始化，但未完成或未终止的分片上传过程。

响应中最多返回 1000 个分片上传过程的信息，它既是响应能返回的最大分片上传过程数目，也是请求的默认值。用户也可以通过设置 max-uploads 参数来限制响应中的分片上传过程数目。如果当前的分片上传过程数超出了这个值，则响应中会包含一个值为 true 的 IsTruncated 元素。如果用户要列出多于这个值的分片上传过程信息，则需要继续调用 List Multipart Uploads 请求，并在请求中设置 key-marker 和 upload-id-marker 参数。

在响应体中，分片上传过程的信息通过 key 来排序。如果用户的应用程序中启动了多个使用同一 key 对象开头的分片上传过程，那么响应体中分片上传过程首先是通过 key 来排序，在相同 key 的分片上传内部则是按上传启动的起始时间的升序来进行排列。

- **请求语法**

```
GET /?uploads HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Date: Date
Authorization: Signature
```

- **请求参数**

名称	描述	是否必须
delimiter	分隔符，一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。delimiter 只支持”/”，不支持其他分隔符。 类型：String	否

max-uploads	设置返回的分片上传过程的最大数目，范围从 1 到 1000，1000 是响应体中能返回的分片上传信息的最大值。 类型：Integer 默认值：1000	否
key-marker	与 upload-id-marker 参数一起，该参数指定列表操作从什么位置后面开始。如果 upload-id-marker 参数没有被指定，那么只有比 key-marker 参数指定的 key 更大的 key 会被列出。如果 upload-id-marker 参数被指定，任何包含与 key-marker 指定值相等或大于 key 的分片上传过程都会被包括，假设这些分片上传过程包含了比 upload-id-marker 指定值更大的 ID。 类型：String	否
Prefix	该参数用于列出那些以 prefix 为前缀的正在进行的上传过程，用户可以使用多个 prefix 来把一个 bucket 分成不同的组（可以考虑采用类似文件系统中的文件夹的作用那样，使用 prefix 来对 key 进行分组）	否
encoding-type	指定响应中 KeyMarker、NextKeyMarker、Key、Prefix、Delimiter 的编码类型。如果 KeyMarker、NextKeyMarker、Key、Prefix、Delimiter 包含 xml 1.0 标准不支持的控制字符，可通过设置该参数对响应中的 KeyMarker、NextKeyMarker、Key、Prefix、Delimiter 进行编码。 类型：String 取值：url，字母不区分大小写。	否
upload-id-marker	与 key-marker 参数一起，指定列表操作从什么位置后面开始。如果 key-marker 没有被指定，upload-id-marker 参数也会被忽略。否则，任何 key 值等于或大于	否

	key-marker 的上传过程都会被包含在列表中，只要 ID 大于 upload-id-marker 指定的值。	

- **响应结果**

名称	描述
ListMultipartUploadsResult	包含整个响应的容器。 类型：容器 子节点：Bucket、KeyMarker、UploadIdMarker、NextKeyMarker、NextUploadIdMarker、Maxuploads、Delimiter、Prefix、Commonfixes、IsTruncated、EncodingType 父节点：无
Bucket	分片上传对应的对象名称。 类型：String 父节点：ListMultipartUploadsResult
KeyMarker	指定 key 值，在这个 key 当前位置或它之后开始列表操作。 类型：String 父节点：ListMultipartUploadsResult
UploadIdMarker	指定分片上传 ID，在这个 ID 所在位置之后开始列表操作。 类型：String 父节点：ListMultipartUploadsResult
NextKeyMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker 作为下一次列表请求的 key-marker 参数的值。 类型：String 父节点：ListMultipartUploadsResult

NextUploadIdMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker 作为下一次列表请求的 upload-id-marker 参数的值。 类型：String 父节点：ListMultipartUploadsResult
MaxUploads	响应中包含的上传过程的最大数目。 类型：String 父节点：ListMultipartUploadsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为 true 则表示没有全部列出。如果分片上传过程的片段数超过了 MaxParts 元素指定的最大数，则会导致一次列表请求无法将所有片段数列出。 类型：Boolean 父节点：ListMultipartUploadsResult
EncodingType	KeyMarker、NextKeyMarker、Key、Prefix、Delimiter 的编码类型。 类型：String 父节点：ListMultipartUploadsResult。
Upload	某个分片上传过程的容器，响应体中可能包含 0 个或多个 Upload 元素。 类型：容器 子节点: Key, UploadId, InitiatorOwner, StorageClass,Initiated 父节点: ListMultipartUploadsResult
Key	分片上传过程起始位置对象的 Key 值。 类型：String 父节点：Upload
UploadId	分片上传过程的 ID 号。

	类型：String 父节点：Upload
Initiator	指定执行此次分片上传过程的用户账户。 子节点：ID，DisplayName 类型：容器 父节点：Upload
ID	OOS 账户的 ID 号。 类型：String 父节点：Initiator，Owner
StorageClass	对象的存储类型：STANDARD。 类型：String 父节点：Upload
Initiated	分片上传过程启动的时间。 类型：Date 父节点：Upload
ListMultipartUploadsResult.Prefix	如果请求中包含了 Prefix 参数，则这个字段会包含 Prefix 的值。返回的结果只包含那些 key 值以 Prefix 开头的对象。 类型：String 父节点：ListMultipartUploadsResult
Delimiter	分割符，用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。 类型：String 父节点：ListMultipartUploadsResult

CommonPrefixes	当定义 delimiter 之后，返回结果中会包含 CommonPrefixes，CommonPrefix 中包含以 prefix 开头，delimiter 结束的左右字符串组合，比如 prefix 是 note/，同时 delimiter 是斜杠（/），结果中的 note/summer/ju 将返回 note/summer/,其余结果将按照 maxkey 要求返回。 类型：String 父节点：ListMultipartUploadsResult
CommonPrefixes.Prefix	如果请求中不包含 Prefix 参数，那么这个元素只显示那些在 delimiter 字符第一次出现之前的 key 的子字符串，且这些 key 不在响应的其它位置出现。 如果请求中包含 Prefix 参数，那么这个元素显示在 prefix 之后，到第一次出现 delimiter 之间的子串。 类型：String 父节点：CommonPrefixes

- **请求示例**

  列出正在进行的三个分片上传过程。请求指定了 max-uploads 参数来设置响应中返回的分片上传过程的最大数目。

```
GET /?uploads&max-uploads=3 HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Connection: keep-alive
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

  下面的示例表示这次 List 操作无法将分片上传过程列举完，用户在下次 List 操作时需要指定 NextKeyMarker 和 NextUploadIdMarker 元素。也就是说，在下一次 List 操作中指定 key-marker=my-movie2.m2ts（NextKeyMarker 的值）和 upload-id-

marker=YW55IGlkZWEgd2h5IGVsdmluZydzIHVwbG9hZCBmYWlsZWQ

（NextUploadIdMarker 的值）。

响应示例中也显示了一个两个分片上传过程拥有同一个 key（my-movie.m2ts）的例子。响应包含了两个通过 key 来排序的上传过程，在每个 key 内部上传过程是根据上传启动的起始时间来排序的。

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 1359
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<ListMultipartUploadsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Bucket>bucket</Bucket>
    <KeyMarker></KeyMarker>
    <UploadIdMarker></UploadIdMarker>
    <NextKeyMarker>my-movie.m2ts</NextKeyMarker>
    <NextUploadIdMarker>YW55IGlkZWEgd2h5IGVsdmluZydzIHVwbG9hZCBmYWlsZWQ</NextUploadIdMarker>
    <MaxUploads>3</MaxUploads>
    <IsTruncated>true</IsTruncated>
    <EncodingType></EncodingType>
    <Upload>
        <Key>my-divisor</Key>
        <UploadId>XMgbGlrZSBlbHZpbmcncyBub3QgaGF2aW5nIG11Y2ggbHVjaw</UploadId>
        <Initiator>
            <ID>mailaddress@ctyun.cn</ID>
            <DisplayName>user1-11111a31-17b5-4fb7-9df5-b111111f13de</DisplayName>
        </Initiator>
        <Owner>
            <ID></ID>
            <DisplayName></DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
        <Initiated>2010-11-10T20:48:33.000Z</Initiated>
    </Upload>
```

```
    <Upload>
        <Key>my-movie.m2ts</Key>
        <UploadId>VXBsb2FkIElEIGZvciBlbHZpbmcncyBteS1tb3ZpZS5tMnRzIHVwbG9hZA</UploadId>
        <Initiator>
            <ID>mailaddress@ctyun.cn</ID>
            <DisplayName>InitiatorDisplayName</DisplayName>
        </Initiator>
        <Owner>
            <ID></ID>
            <DisplayName></DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
        <Initiated>2010-11-10T20:48:33.000Z</Initiated>
    </Upload>
    <Upload>
        <Key>my-movie.m2ts</Key>
        <UploadId>YW55IGlkZWEgd2h5IGVsdmluZydzIHVwbG9hZCBmYWlsZWQ</UploadId>
        <Initiator>
            <ID>mailaddress@ctyun.cn</ID>
            <DisplayName>user1-22222a31-17b5-4fb7-9df5-b222222f13de</DisplayName>
        </Initiator>
        <Owner>
            <ID></ID>
            <DisplayName></DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
        <Initiated>2010-11-10T20:49:33.000Z</Initiated>
    </Upload>
</ListMultipartUploadsResult>
```

4.3.12 PUT Bucket Logging

此操作用来添加/修改/删除 logging 的操作。如果 Bucket 已经存在了 logging，此操作会替换原有 logging。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
PUT /?logging HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue

Request elements vary depending on what you're setting.
```

- **请求元素**

名称	描述	是否必须
BucketLoggingStatus	请求的容器。	是
LoggingEnabled	日志信息的容器，当启动日志时，需要包含这个元素。	否
TargetBucket	指定要保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。可以设置任意一个你拥有的 Bucket 作为 TargetBucket，包括启动日志的 Bucket 本身。你也可以设置将多个 Bucket 的日志存放到一个 TargetBucket 中，在这种情况下，你需要为每个源 Bucket 设置不同的 TargetPrefix，以便不同 Bucket 的 log 可以被区分出来。	否
TargetPrefix	生成的 log 文件将以此为前缀命名。	否
TriggerTargetBucket	在异地互备过程中，目标资源池的 log 会保存到此 Bucket 中，OOS 会向此 Bucket 存储日志。	否
TriggerTargetPrefix	在异地互备过程中，目标资源池的 log 文件将以此为前缀命名。	否

TriggerSourceBucket	在异地互备过程中，源资源池的 log 会保存到此 Bucket 中，OOS 会向此 Bucket 存储日志。	否
TriggerSourcePrefix	在异地互备过程中，源资源池的 log 文件将以此为前缀命名。	否

- **请求示例 1**

  启动日志。

```
PUT /?logging HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
    <LoggingEnabled>
        <TargetBucket>mybucketlogs</TargetBucket>
        <TargetPrefix>mybucket-access_log-/</TargetPrefix>
    </LoggingEnabled>
</BucketLoggingStatus>
```

- **响应示例 1**

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

- **请求示例 2**

  取消日志。

```
PUT /?logging HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

```
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns=http://doc.s3.amazonaws.com/2006-03-01/>
```

- 响应示例 2

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

- 日志名称

```
TargetPrefixGMTYYYY-mm-DD-HH-MM-SS-唯一字符串
```

其中 YYYY，mm，DD，HH，MM，SS 分别代表日志发送时的年，月，日，小时，分钟，秒。

TargetPrefix 是用户在启动 Bucket 日志时配置的，如果未配置，则该项不存在。唯一字符串是服务端生成的，某些场景下不会生成。

系统不会删除旧的日志文件。用户可以自己删除以前的日志文件，可以在 List Object 时指定 prefix 参数，挑选出旧的日志文件，然后删除。

当客户端的请求到达后，日志记录不会被立刻推送到 TargetBucket 中，会延迟一段时间。

- 日志格式

字段名称	示例	备注
BucketOwner	mailaddress@ctyun.cn	源 Bucket Owner。
Bucket name	mybucket	请求的 Bucket，如果 OOS 收到一个错误的 bucket 名称，那么这个请求不会出现在任何 log 中。
Time	04/08/2012 22:34:02 zz	请求到达的时间，格式是dd/MM/yyyy HH:mm:ss zz。

Remote IP	72.21.206.5	请求者的 IP 地址。中间的代理和防火墙可能会隐藏实际发出请求的机器的地址。
Requester	mailaddress@ctyun.cn	请求者的邮箱。如果是匿名访问，显示 Anonymous。
Request ID	e4dd82b2f0994896	Request ID 是 OOS 生成的一个字符串，用于唯一标示每个请求
Operation	REST.PUT.OBJECT	REST.HTTP_method. resource_type
Key	/photos/2012/08/puppy.jpg	请求的“Key”部分，URL 编码。如果请求中没有指定对象，显示“-”
Request-URI	"GET /mybucket/photos/2012/08/puppy.jpg HTTP/1.1"	HTTP 请求中的 Request-URI 部分
HTTP status	200	响应的 HTTP 状态码
Error Code	NoSuchBucket	OOS 错误码，如果没有错误，显示“-”
Bytes Sent	2662992	写请求或读响应发送的字节数，不包括 HTTP 协议的头。如果是 0，显示“-”
Object Size	3462992	Object 的总大小
Time	70	从收到请求到发出响应的时间
Referer	"http://oos.ctyun.cn"	HTTP Referer 请求头的值
User-Agent	"curl/7.15.1"	HTTP User-Agent 请求头的值
Version Id	3HL4kqtJvjVBH40Nrjfkd	请求中的 versionId 参数，如果没有，显示“-”

4.3.13 GET Bucket Logging

在 GET 操作的 url 中加上 logging，可以获得指定 Bucket 的 logging。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
GET /?logging HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应结果**

名称	描述
BucketLoggingStatus	响应的容器。
LoggingEnabled	日志信息的容器，当启动日志时，包含这个元素；否则此元素及其子元素都不显示。
TargetBucket	保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。
TriggerTargetBucket	在异地互备过程中，目标资源池的 log 会保存到此 Bucket 中，OOS 会向此 Bucket 存储日志。
TriggerTargetPrefix	在异地互备过程中，目标资源池的 log 文件将以此为前缀命名。
TriggerSourceBucket	在异地互备过程中，源资源池的 log 会保存到此 Bucket 中，OOS 会向此 Bucket 存储日志。
TriggerSourcePrefix	在异地互备过程中，源资源池的 log 文件将以此为前缀命名。

- **请求示例**

```
GET /?logging HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

以下是设置了日志的响应示例

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
    <LoggingEnabled>
        <TargetBucket>mybucketlogs</TargetBucket>
        <TargetPrefix>mybucket-access_log-/</TargetPrefix>
    </LoggingEnabled>
</BucketLoggingStatus>
```

以下是没有设置日志时的响应示例

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns=http://doc.s3.amazonaws.com/2006-03-01/>
```

4.3.14 HEAD Bucket

此操作用于判断 Bucket 是否存在，而且用户是否有权限访问。如果 Bucket 存在，而且用户有权限访问时，此操作返回 200OK。否则，返回 404 不存在，或者 403 没有权限。

- **请求语法**

```
HEAD / HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
HEAD / HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.15 PUT Bucket Trigger

在 PUT 操作的 url 中加上 trigger，可以进行添加 trigger 的操作，即添加一个向异地资源池同步的触发器。当客户端向本地资源池的 Bucket 上传对象时，OOS 可以根据配置的策略，自动将对象同步到异地资源池中。一个 Bucket 可以配置多个触发器，但只能有一个是默认的触发器。如果客户端要使用非默认的触发器上传对象，需要在 PUT Object 时，加上请求头 `x-ctyun-trigger`，值是指定的 `TriggerName`。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
PUT /?trigger  HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
 Date: date
Content-Length: Length
Authorization: signatureValue

<TriggerConfiguration>
  <Trigger>
    <TriggerName>trigger1</TriggerName>
    <IsDefault>false</IsDefault>
    <RemoteSite>
      <RemontEndPoint>http://oos-
      js.ctyunapi.cn</RemontEndPoint><ReplicaMode>STANDARD</ReplicaMode>
      <RemoteBucketName>bucket2</RemoteBucketName>
      <RemoteAK> ak2</RemoteAK>
      <RemoteSK>sk2</RemoteSK>
    </RemoteSite>
  </Trigger>
</TriggerConfiguration>
```

- **请求元素**

名称	描述	是否必须
TriggerName	Trigger 的名称，是字母或数字，不能包含特殊符号，最多	是

	20 个字符。	
IsDefault	是否是默认的 trigger。	是
RemoteSite	异地资源池的相关配置，可以配置向多个异地资源池同步数据，每个异地资源池对应一个 RemoteSite。如果不配置 RemoteSite，即不复制到其他资源池。	否
RemontEndPoint	异地资源池的 endpoint。	是
ReplicaMode	同步到异地资源池的副本模式，如果不填写，会使用动态副本模式。	否
RemoteBucketName	异地资源池的 bucketName。	是
RemoteAK	异地资源池的 AccessKey。	是
RemoteSK	异地资源池的 secretKey。	是

4.3.16 GET Bucket Trigger

在 GET 操作的 url 中加上 trigger，可以查询指定 Bucket 中配置的所有 trigger。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
GET /?trigger HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Date: date

Authorization: signatureValue
```

- **响应结果**

名称	描述
TriggerName	Trigger 的名称
IsDefault	是否是默认的 trigger
RemoteSite	异地资源池的相关配置
RemontEndPoint	异地资源池的 endpoint
ReplicaMode	同步到异地资源池的副本模式
RemoteBucketName	异地资源池的 bucketName
RemoteAK	异地资源池的 AccessKey
RemoteSK	异地资源池的 SecretKey

- **请求示例**

```
GET /jsbuckettest/?trigger HTTP/1.1
Host: oos-js.ctyunapi.cn
Authorization: signatureValue
Date: Mon, 06 Jun 2022 08:11:58 GMT
Content-Type: application/xml
Connection: Keep-alive
```

- 响应示例

```
HTTP/1.1 200 OK
Content-Type: application/xml;charset=UTF-8
Date: Mon, 06 Jun 2022 08:11:58 GMT
x-amz-request-id: 870498e4282345c3
Content-Length: 485
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<TriggerConfiguration
    xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Trigger>
        <TriggerName>tiggerName1214</TriggerName>
        <IsDefault>true</IsDefault>
        <RemoteSite>
            <RemontEndPoint>http://oos-js.ctyunapi.cn</RemontEndPoint>
            <ReplicaMode>STANDARD</ReplicaMode>
            <RemoteBucketName>jsbuckettest</RemoteBucketName>
            <RemoteAK>f18e108a91f151ff412e</RemoteAK>
            <RemoteSK>95ccc07e04ec499e6805780e82aca08e72777a45</RemoteSK>
        </RemoteSite>
    </Trigger>
</TriggerConfiguration>
```

4.3.17 DELETE Bucket Trigger

在 DELETE 操作的 url 中加上 trigger，可以删除某 Bucket 中的指定 trigger。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
DELETE /?trigger=triggerName HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应示例**

```
HTTP/1.1 204 No Content
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.3.18 PUT Bucket Lifecycle

此操作用来设置 Bucket 生命周期规则。

生命周期是指对象从更新开始到被删除的时间。

通过设置存储桶的生命周期规则，可以删除与生命周期规则匹配的对象。当对象的生命周期到期时，OOS 会异步删除它们。生命周期中配置的到期时间和实际删除时间之间可能会有一段延迟。但对象到期被删除后，用户将不需要为到期的对象付费。OOS 删除到期对象后，会在 Bucket log 中记录一条日志，操作项是"OOS.EXPIRE.OBJECT"。

注意：

- 如果对象的生命周期规则设置的是到期后删除，对象到期后将被永久删除，无法恢复。
- 如果 Bucket 内的生命周期规则正在执行时被修改配置，则修改后的配置并不立即生效，需等原生命周期规则执行完成后才能生效。
- 如果 Bucket 没有配置过生命周期规则，执行该操作将创建新的生命周期规则；如果 Bucket 已存在生命周期规则，则执行此操作将覆盖原有规则。
- 每个 Bucket 最多创建 1000 条生命周期规则。
- 同一 Bucket 的生命周期规则不能存在叠加前缀，例如已创建到期删除对象的生命周期规则的前缀是 ABC，则无法再创建前缀为 ABCD 或 AB 或 A 的到期删除对象的生命周期规则。
- 当用户为 Bucket 设置了生命周期规则，这些规则将同时应用于已有对象和后续新创建的对象。例如，用户今天增加了一个生命周期，指定某些前缀的对象 30 天后过期，那么 OOS 将会将满足条件的 30 天前创建的对象都加入到待删除队列中。

OOS 通过将对象的创建时间加上生命周期时间来计算到期时间，并且将时间近似到下一天的 GMT 零点时间。例如，一个对象于 GMT 2016 年 1 月 15 日 10:30 创建，生命周期为 3 天，那么对象的到期时间是 GMT 2016 年 1 月 19 日 00:00。当重写一个对象时，OOS 将以最后更新时间为准，来重新计算该对象的到期时间。

可以通过 GET Object、HEAD Object 查询对象的到期时间。

- **请求语法**

```
PUT /?lifecycle HTTP/1.1
```

```
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
Content-MD5: MD5

<LifecycleConfiguration>
    <Rule>
        <ID>UniqueIdentifier</ID>
        <Prefix>Prefix</Prefix>
        <Status>LifecycleStatus</Status>
        <Expiration>
            <Days>NumberOfDays</Days>
        </Expiration>
</Rule>
</LifecycleConfiguration>
```

- **请求头**

名称	描述	是否必需
Content-MD5	数据的 base64 编码的 128 位 MD5。此请求头必填，以便校验数据的完整性。	是

- **请求元素**

名称	描述	是否必需
LifecycleConfiguration	容器，最多包含 1000 个规则 类型：容器 子节点：Rule 父节点：无	是
Rule	生命周期规则的容器 类型：容器 父节点：LifecycleConfiguration	是
ID	规则的唯一标识，最长 255 个字符。	否

	类型：字符串 父节点：Rule	
Prefix	指明要使用规则的对象前缀，最长 1024 个字符 类型：字符串 父节点：Rule	否
Status	指定生命周期规则的状态： ● Enabled：生命周期规则立即生效。 ● Disabled：生命周期规则不生效。 类型：字符串 父节点：Rule	是
Expiration	描述过期动作的容器。 类型：容器 子节点：Days 父节点：Rule	是
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration	Days 和 Date 二选一
Date	生成时间早于此时间的对象将被认为是过期对象 日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。 例如：2002-10-11T00:00:00.000Z 类型：String 父节点：Expiration	Days 和 Date 二选一

- **请求示例 1：单一规则**

  下面的生命周期只包含一个规则，这个规则指定所有以 logs/为前缀的对象将在创建后的 30 天过期。因为规则的状态是 Enabled，OOS 会每隔一段时间来评估这个规则，删除过期的对象。

```
PUT /?lifecycle HTTP/1.1
```

```
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
Content-MD5: MD5

<LifecycleConfiguration>
    <Rule>
        <ID>UniqueIdentifier</ID>
        <Prefix>Prefix</Prefix>
        <Status>LifecycleStatus</Status>
        <Expiration>
            <Days>NumberOfDays</Days>
        </Expiration>
</Rule>
</LifecycleConfiguration>
```

- **响应示例 1**

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Tue, 13 Dec 2011 19:14:41 GMT
Connection: close
Server: CTYUN
```

- **请求示例 2：多规则**

  下面的生命周期包含 2 个规则，第一个规则指定所有以 logs/为前缀的对象将在创建后的 30 天过期。第二个规则指定所有以 documents/为前缀的对象将在创建后的 365 天过期，但第二个规则的状态是 Disabled，即不生效。

```
PUT /?lifecycle HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Tue, 13 Dec 2011 17:54:50 GMT
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Length: 413
<LifecycleConfiguration>
```

```
  <Rule>
    <ID>delete-logs-rule</ID>
    <Prefix>logs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
        <Days>30</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>delete-documents-rule</ID>
    <Prefix>documents/</Prefix>
    <Status>Disabled</Status>
    <Expiration>
        <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

- **响应示例 2**

```
HTTP/1.1 200OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Tue, 13 Dec 2011 19:14:41 GMT
Connection: close
Server: CTYUN
```

- **请求示例 3：定义所有对象的规则**

  下面的规则指定所有对象将在创建后的 3650 天被删除。

  注意：当指定空的前缀时，规则将对 Bucket 内的所有对象生效。OOS 将会删除满足此规则的所有对象。

```
PUT /?lifecycle HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Tue, 13 Dec 2011 17:54:50 GMT
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

```
Content-Length: 226

<LifecycleConfiguration>
    <Rule>
        <ID>10 years all objects expire rule </ID>
        <Prefix></Prefix>
        <Status>Enabled</Status>
        <Expiration>
            <Days>3650</Days>
        </Expiration>
    </Rule>
</LifecycleConfiguration>
```

- **响应示例 3**

```
HTTP/1.1 200OK
x-amz-request-id: BDC4B83DF5096BBE
Date:Tue, 13 Dec 2011 19:14:41 GMT
Connection: close
Server: CTYUN
```

- **请求示例 4：避免设置冲突的规则**

  在设置多个规则时，请注意各规则之间不要互相冲突。比如，下面的生命周期中配置了一个规则，指定所有以 documents 为前缀的对象在 30 天后过期。而另一个规则指定所有以 documents/2011 为前缀的对象在 365 天过期。在这种情况下，OOS 将返回错误信息。

```
PUT /?lifecycle HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Tue, 13 Dec 2011 17:54:50 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Content-Length: 226

<LifecycleConfiguration>
    <Rule>
        <ID>111</ID>
```

```
        <Prefix>documents</Prefix>
        <Status>Enabled</Status>
        <Expiration>
            <Days>30</Days>
        </Expiration>
    </Rule>
    <Rule>
        <ID>222</ID>
        <Prefix>documents/2011</Prefix>
        <Status>Enabled</Status>
        <Expiration>
            <Days>365</Days>
        </Expiration>
    </Rule>
</LifecycleConfiguration>
```

4.3.19 GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期。

- **请求语法**

```
GET /?lifecycle HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应结果**

名称	描述
LifecycleConfiguration	容器，最多包含 100 个规则。 类型：容器 子节点：Rule 父节点：无
Rule	生命周期规则的容器。 类型：容器 父节点：LifecycleConfiguration
ID	规则的唯一标示，最长 255 个字符。 类型：字符串 父节点：Rule
Prefix	指明要使用规则的对象前缀。 类型：字符串 父节点：Rule
Status	生命周期规则的状态： ● Enabled：生命周期规则立即生效。 ● Disabled：生命周期规则不生效。 类型：字符串

	父节点：Rule
Expiration	描述过期动作的容器。 类型：容器 子节点：Days 父节点：Rule
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration
Date	生成时间早于此时间的对象将被认为是过期对象。 日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。例如：2002-10-11T00:00:00.000Z。 类型：String 父节点：Expiration

- **错误信息**

错误码	错误信息	HTTP 响应码
NoSuchLifecycleConfiguration	The lifecycle configuration does notexist.	404 Not Found

- **请求示例**

下面的 GET 请求从指定的 Bucket 中获取生命周期的配置信息，OOS 将生命周期的配置返回在响应体中。下面的例子显示所有以 logs 开头的对象将在创建后的 30 天到期。

```
GET /?lifecycle HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Tue, 13 Dec 2011 17:54:50 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200OK
x-amz-request-id: BDC4B83DF5096BBE
```

```
Date:Tue, 13 Dec 2011 19:14:41 GMT
Content-Length: 267
Connection: close
Server: CTYUN

    <?xml version="1.0" encoding="UTF-8"?>
    <LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Rule>
    <ID>30-day-log-deletion-rule</ID>
    <Prefix>logs</Prefix>
    <Status>Enabled</Status>
    <Expiration>
        <Days>30</Days>
    </Expiration>
    </Rule>
</LifecycleConfiguration>
```

4.3.20 DELETE Bucket Lifecycle

此接口用于删除配置的 Bucket 生命周期，OOS 将会删除指定 Bucket 的所有生命周期配置规则。用户的对象将永远不会到期，OOS 也不会再自动删除对象。

- **请求语法**

```
DELETE /?lifecycle HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
DELETE /?lifecycle HTTP/1.1
Host: doc.oos-js.ctyunapi.cn
Date: Wed, 14 Dec 2011 05:37:16 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 204No Content
x-amz-request-id: BDC4B83DF5096BBE
Date:Tue, 13 Dec 2011 19:14:41 GMT
Connection: close
Server: CTYUN
```

4.3.21 PUT Bucket CORS

跨域资源共享(Cross-Origin Resource Sharing，CORS)定义了客户端 Web 应用程序在一个域中与另一个域中的资源进行交互的方式，是浏览器出于安全考虑而设置的一个限制，即同源策略。例如，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。

通过 CORS，客户可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨域访问 OOS 资源。

以下是有关使用 CORS 的示例场景：

- 场景 1：比如用户的网站 www.exam***ple.com，后端使用了 OOS。在 web 应用中提供了使用 JavaScript 实现的上传对象功能，但是在该 web 应用中，只能向 www.exam***ple.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.exam***ple.com 中转。如果设置了跨域访问的话，用户就可以直接上传到 OOS，而无需从 www.exam***ple.com 中转。
- 场景 2：假设用户在名为 website 的 Bucket 中托管网站，网站的 Endpoint 是 http://website.oos-website-cn.oos-xx.ctyunapi.cn。现在，用户想要使用网页上的 JavaScript (存储在此 Bucket 中)，通过 OOS API endpoint oos-xx.ctyunapi.cn 向 Bucket 发送 GET 和 PUT 请求。浏览器通常会阻止 JavaScript 发送这些请求，但借助 CORS，用户可以配置 Bucket 支持来自 website.oos-website-cn.oos-xx.ctyunapi.cn 的跨域请求。

设置 Bucket 的跨域请求。如果配置已经存在，OOS 会覆盖它。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。在配置跨域请求时，用户可以通过 XML 来配置允许跨域的源和 HTTP 方法。XML 请求体不能超过 64KiB。

OOS 收到来自浏览器的预检请求后，它将为 Bucket 评估 CORS 配置，并使用第一个与浏览器请求相匹配的 CORSRule 规则来实现跨域请求。要使规则实现匹配，必须满足以下条件：

- 请求的 Origin 标头必须匹配一个 `AllowedOrigin` 元素。
- 请求方法(例如，GET 或 PUT)，或者预检 OPTIONS 请求中的 `Access-Control-Request-Method` 请求头，必须是某个 `AllowedMethod` 元素。
- 在预检请求中，`Access-Control-Request-Headers` 请求头中列出的每个请求头，必须匹配一个 `AllowedHeader` 元素。

- **请求语法**

```
PUT /?cors HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Content-Length: length
Date: date
Authorization: signatureValue
Content-MD5: MD5

<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>Origin you want to allow cross-domain requests from</AllowedOrigin>
        <AllowedOrigin>...</AllowedOrigin>
        ...
        <AllowedMethod>HTTP method</AllowedMethod>
        <AllowedMethod>...</AllowedMethod>
        ...
        <MaxAgeSeconds>Time in seconds your browser to cache the pre-flight OPTIONS response
for a resource</MaxAgeSeconds>
        <AllowedHeader>Headers that you want the browser to be allowed to
send</AllowedHeader>
        <AllowedHeader>...</AllowedHeader>
        ...
        <ExposeHeader>Headers in the response that you want accessible from client
application</ExposeHeader>
        <ExposeHeader>...</ExposeHeader>
        ...
    </CORSRule>
    <CORSRule>
        ...
    </CORSRule>
</CORSConfiguration>
```

- **请求元素**

名称	描述	是否必需

CORSConfiguration	最多包含100个CORSRules元素的容器。 类型：容器	是
CORSRule	用户允许跨域的源和方法。 类型：容器 子节点: AllowedOrigin, AllowedMethod, MaxAgeSeconds,ExposeHeader, ID. 父节点: CORSConfiguration	是
ID	规则的唯一标示。最长255个字符。 类型：String 父节点: CORSRule	否
AllowedMethod	允许跨域的HTTP方法。每个CORSRule应至少包含一个源和一个方法。 类型: 枚举 (GET, PUT, HEAD, POST, DELETE) 父节点：CORSRule	是
AllowedOrigin	允许跨域的源。最多包含一个 * 通配符。比如：http://*.ctyun.cn。 用户也可以只指定 * 表示允许所有源跨域访问。 类型: String 父节点: CORSRule	是
AllowedHeader	控制在预检OPTIONS请求中Access-Control-Request-Headers头中指定的header是否允许。Access-Control-Request-Headers 中的每个请求头名称，必须在规则中有匹配的条目。 规则中的每个AllowedHeader最多可以包含一个 * 通配符字符。例如，<AllowedHeader>x-amz-*</AllowedHeader>。 类型: String 父节点: CORSRule	否

MaxAgeSeconds	指定浏览器对特定资源的预检（OPTIONS）请求返回结果的缓存时间，单位为秒。通过缓存响应，在需要重复原始请求时，浏览器无需向 OOS 发送预检请求。 一个CORSRule最多包含一个MaxAgeSeconds元素。 类型: Integer 父节点: CORSRule 取值：大于等于-1 的整数。-1 表示禁用缓存。	否
ExposeHeader	指定客户应用程序 (例如：JavaScript XMLHttpRequest对象) 能够访问的响应头。 类型: String 父节点: CORSRule	否

- **请求示例**

  第一个规则允许来自 https://docs.oos-cn.ctyunapi.cn 源的跨源 PUT、POST 和 DELETE 请求。该规则还通过 Access-Control-Request-Headers 标头允许预检 OPTIONS 请求中的所有标头。作为对任何预检 OPTIONS 请求的响应，OOS 将返回请求的任意请求头。

  第二个规则允许与第一个规则具有相同的跨源请求，但第二个规则应用于另一个源 https://example.bucket.oos-cn.ctyunapi.cn。

  第三个规则允许来自所有源的跨源 GET 请求。“*”通配符字符是指所有的源。

```
PUT /?cors HTTP/1.1
Host: examplebucket.oos-js.ctyunapi.cn
x-amz-date: Tue, 21 Aug 2012 17:54:50 GMT
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Authorization: signatureValue
Content-Length: 445

<CORSConfiguration>
 <CORSRule>
```

```
    <AllowedOrigin>https://docs.oos-cn.ctyunapi.cn</AllowedOrigin>

    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>

    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
  <CORSRule>
    <AllowedOrigin>https://example.bucket.oos-cn.ctyunapi.cn</AllowedOrigin>

    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>

    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
  </CORSRule>
</CORSConfiguration>
```

- 响应示例

```
HTTP/1.1 200 OK
x-amz-request-id: BDC4B83DF5096BBE
Date: Mon, 21 Aug 2017 17:54:50 GMT
Connection: close
Server:OOS
```

4.3.22 GET Bucket CORS

返回 Bucket 的跨域配置信息。只有 Bucket owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
GET /?cors HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应结果**

名称	描述
CORSConfiguration	最多包含100个CORSRules 元素的容器 类型：容器
CORSRule	用户允许跨域的源和方法。 类型：容器 子节点: AllowedOrigin、AllowedMethod、MaxAgeSeconds、ExposeHeader、ID 父节点： CORSConfiguration
ID	规则的唯一标示。最长255个字符。 类型：String 父节点：CORSRule
AllowedMethod	用户允许跨域的HTTP方法。每个CORSRule应至少包含一个源和一个方法。 类型：枚举 (GET、PUT、HEAD、POST、DELETE) 父节点：CORSRule
AllowedOrigin	用户允许跨域的源。最多包含一个 * 通配符。比如： http://*.ctyun.cn。

	用户也可以只指定 * 表示允许所有源跨域访问。 类型：String 父节点：CORSRule
AllowedHeader	通过Access-Control-Request-Headers请求头，指定预检OPTIONS请求中允许的请求头。Access-Control-Request-Headers中的每个请求头名称，必须在规则中有匹配的相应条目。 OOS 将仅发送允许的响应头。 规则中的每个AllowedHeader字符串可以最多包含一个 * 通配符字符。例如：<AllowedHeader>x-amz-*</AllowedHeader>。 类型: String 父节点: CORSRule
MaxAgeSeconds	指定浏览器为预检请求缓存响应的时间，以秒为单位。 一个CORSRule最多包含一个MaxAgeSeconds元素。 类型：Integer 父节点：CORSRule
ExposeHeader	指定客户应用程序（例如：JavaScript XMLHttpRequest对象）能够访问的响应头。 类型: String 父节点: CORSRule

- **请求示例**

```
GET /?cors HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 0CF038E9BCF63097
```

```
Date: Wed, 13 Dec 2017 19:14:42 GMT
Connection: close
Server: OOS
Content-Length: 280

<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>http://www.ctyun.cn</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSec>
        <ExposeHeader>x-amz-server-side-encryption</ExposeHeader>
    </CORSRule>
</CORSConfiguration>
```

4.3.23 DELETE Bucket CORS

删除 Bucket 的跨域配置信息。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

- **请求语法**

```
DELETE /?cors HTTP/1.1
Host: bucketname.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求示例**

```
DELETE /?cors HTTP/1.1
Host: examplebucket.oos-js.ctyunapi.cn
Date: Tue, 13 Dec 2011 19:14:42 GMT
Authorization: signatureValue
```

- **响应示例**

```
HTTP/1.1 204 No Content
x-amz-request-id: 0CF038E9BCF63097
Date: Tue, 13 Dec 2011 19:14:42 GMT
Connection: close
Server: OOS
```

4.4 关于 Object 的操作

4.4.1 PUT Object

此操作用来向指定 Bucket 中添加一个对象，要求发送请求者对该 Bucket 有写权限，用户必须添加完整的对象。

说明：对象名称不能包含 ASCII 码为 0 的字符（NUL）。

- **请求语法**

```
PUT /ObjectName HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求参数**

名称	描述	是否必须
Cache-Control	按照请求/回应的方式用来定义缓存行为	否
Content-Disposition	指出对象的描述性的信息	否
Content-Encoding	指出对象所使用的编码格式	否
Content-Length	用字节的方式定义对象的大小	是
Content-MD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值	否
Content-Type	标准的 MIME 类型用来描述内容格式。 如果要在浏览器中预览对象，需要在 put 对象时，增加 Content-Type 请求头，例如 jpg 图片的 Content-Type 是 image/jpeg；pdf 文件的 Content-Type 是 application/pdf。	否
Expires	对象不再被缓存的时间 类型：String	否

x-amz-meta-	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和对象一起被存储并返回。PUT 请求头大小限制为 8KiB。在 PUT 请求头中，用户定义的元数据大小限制为 2KiB。	否
x-amz-limit	对像上传限制的速率。 格式为：x-amz-limit:rate=xxx，取值为大于 0 的整数，单位是 KiB/s。当取值是大于 0 小于 128 的整数时，按速率等于 128KiB/s 处理。	否
x-amz-storage-class	数据的存储类型，默认采用动态副本模式存储。用户也可选择使用标准模式进行存储。对于那些不太重要，可以重复生成的数据，用户可以选择减少冗余策略来降低成本。 类型: String 取值：STANDARD 默认值为STANDARD。	否

- **请求示例**

在 Bucket myBucket 中，存储图片 my-image.jpg。

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 17:50:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Type: image/ipeg
Content-Length: 11434
[11434 bytes of object data]
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 0A49CE4060975EAC
Date: Mon, 03 Sep 2012 17:50:00 GMT
```

```
ETag: "1b2cf535f27731c974343645a3985328"
Content-Length: 0
Connection: close
Server: CTYUN
```

4.4.2 GET Object

此操作用来检索在 OOS 中的对象信息，执行 GET 操作，用户必须对 Object 所在的 Bucket 有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通过非授权的方式进行读操作。

- **请求语法**

```
GET /ObjectName HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求变量**

名称	描述	是否必须
response-content-type	设置返回头中的 Content-Type。	否
response-content-language	设置返回头中的 Content-Language。	否
response-cache-control	设置返回头中的 Cache-Control。	否
response-content-disposition	设置返回头中的 Content-Disposition。 注：OOS 会把 response-content-disposition 中的值设置到响应头 Content-Disposition 中。对于不同的浏览器，此值的编码方式可能不同，此工作由客户端来完成。例如对于 IE 浏览器，要设置下载的文件名为”文件.txt”，那么 response-content-disposition 要设置为 attachment;filename=URLEncoder.encode(URLEncoder.encode("文件.txt","UTF-8"), "UTF-8")。	否

response-content-encoding	设置返回头中的 Content-Encoding。	否
x-amz-limit	对象下载限制的速率。 格式为： ● x-amz-limit:rate=*xxx* ● x-amz-limit:concurrency=*xxx* ● x-amz-limit:rate=*xxx*, concurrency=*xxx* 其中 ● rate 为速率，取值为大于 0 的整数，单位是 KiB/s。当取值是大于 0 小于 128 的整数时，按速率等于 128KiB/s 处理。 ● concurrency 为并发连接数。取值为大于 0 的正整数。	否
response-expires	设置返回头中的 Expires。	否

- **请求头格式**

名称	描述	是否必须
Range	下载一个对象的指定字节范围。表达格式为 Range: bytes=n1-n2。n1，n2 为大于等于 0 的整数。	否
If-Modified-Since	返回一个在指定时间点后被修改的对象，否则返回 304 错误	否
If-Unmodified-Since	返回一个在指定时间点后未被修改的对象，否则返回 412 错误	否
If-Match	当对象的 ETag 与指定值一致时，返回此对象。否则返回 412 错误	否
If-None-Match	当对象的 ETag 与指定值不一致时，返回此对象。否则返回 304 错误	否

- **响应结果**

名称	描述
x-amz-expiration	如果对象被配置了到期时间，那么 OOS 返回此响应头。这个响应头包含键值对 expiry-date 和 rule-id。rule-id 的值是 URL 编码的。

- **请求示例**

下面示例中返回对象 my-image.jpg

```
GET /my-image.jpg HTTP/1.1
Host: bucket.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 318BC8BC148832E5
Date: Mon, 03 Sep 2012 22:32:00 GMT
Last-Modified: Sat, 01 Sep 2012 17:50:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: image/ipeg
Connection: close
Server: CTYUN
[434234 bytes of object data]
```

4.4.3 DELETE Object

Delete 操作移除指定的对象，具有对象所在 Bucket 写权限的用户才可以执行此操作。

- **请求语法**

```
DELETE /ObjectName HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Content-Length: length
Authorization: signatureValue
```

- **请求示例**

删除一个图片对象 my-image.jpg。

```
DELETE /my-image.jpg HTTP/1.1
Host: bucket.oos-js.ctyunapi.cn
Date: Mon, 03 Sep 2012 17:50:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Type: image/ipeg
```

- **响应示例**

```
HTTP/1.1 204 NoContent
x-amz-request-id: 0A49CE4060975EAC
Date: Mon, 03Sep 2012 17:50:00 GMT
Content-Length: 0
Connection: close
Server: CTYUN
```

4.4.4 PUT Object - Copy

此操作用来创建一个存储在 OOS 里的对象拷贝。类似于执行一个 GET，然后再执行一次 PUT。要执行拷贝请求，用户需要对源对象有读权限，对目标 Bucket 有写权限。

- **请求语法**

```
PUT /destinationObject HTTP/1.1
Host: destinationBucket.oos-js.ctyunapi.cn
x-amz-copy-source: /source_bucket/sourceObject
x-amz-metadata-directive: metadata_directive
x-amz-copy-source-if-match: etag
x-amz-copy-source-if-none-match: etag
x-amz-copy-source-if-unmodified-since: time_stamp
x-amz-copy-source-if-modified-since: time_stamp
x-amz-meta-paramater: value
Authorization: signatureValue
Date: date
```

- **请求头格式**

名称	描述	是否必须
x-amz-copy-source	源 Bucket 和对象的名称，用斜杠(/)分割。 类型：字符串	是
x-amz-metadata-directive	指明元数据是源对象的拷贝或者元数据被请求头提供的元数据覆盖。 **类型：**字符串 **取值：** ● COPY：除存储类型（x-amz-storage-class）外的其他元数据保持不变，拷贝源对象的元数据； ● REPLACE：所有原始元数据都被指定的元数据覆盖。	否

	默认值为 COPY。 **注意**：如果取值为 COPY，源对象和目的对象相同，则必须携带 x-amz-storage-class，否则不能拷贝，返回 400 错误码。	
x-amz-copy-source-if-match	只有当源对象的 Etag 与给定 Etag 匹配时，才能执行对象拷贝操作，否则返回 412 HTTP 状态码错误。 **类型**：字符串	否
x-amz-copy-source-if-none-match	只有当源对象的 Etag 与给定 Etag 不匹配时，才能执行对象拷贝操作，否则返回 412 HTTP 状态码错误。 **类型**：字符串	否
x-amz-copy-source-if-unmodified-since	只有源对象在指定时间点之后没有修改，才执行对象拷贝操作，否则返回 412 错误。 **类型**：符合 https://tools.ietf.org/html/rfc7232 规定格式的 HTTP 时间字符串。	否
x-amz-copy-source-if-modified-since	只有源对象在指定时间点之后修改过，才执行对象拷贝操作，否则返回 412 错误。 **类型**：符合 https://tools.ietf.org/html/rfc7232 规定格式的 HTTP 时间字符串。	否
x-amz-storage-class	新对象的存储类型。 **类型**：String **取值**：STANDARD：标准存储； 默认值为 STANDARD。	否
x-amz-meta-*paramater*	用户自定义的元数据，用户可以根据需要，自定义一些元数据的参数。 **类型**：字符串	否

- **响应结果**

名称	描述
CopyObjectResult	包含所有响应结果的容器。
ETag	返回新对象的 ETag。ETag 只反映对象内容发生了改变，元数据未改变。
LastModified	返回对象最后一次修改的日期。

- **请求示例**

  将存储桶 bucket-sample 中的对象 my-image.jpg 复制一份到本存储桶，新对象命名为 my-second-image.jpg。

```
PUT /my-second-image.jpg HTTP/1.1
Host: bucket-sample.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
x-amz-copy-source: /bucket-sample/my-image.jpg
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 318BC8BC148832E5
Date: Mon, 03Sep 2012 22:32:00 GMT
Connection: close
Server: CTYUN

<CopyObjectResult>
  <LastModified>2012-09-01T22:32:00</LastModified>
  <ETag>9b2cf535f27731c974343645a3985328</ETag>
</CopyObjectResult>
```

4.4.5 Initial Multipart Upload

本接口初始化一个分片上传（Multipart Upload）操作，并返回一个上传 ID，此 ID 用来将此次分片上传操作中上传的所有片段合并成一个对象。用户在执行每一次子上传请求（见**错误！未找到引用源。**）时都应该指定该 ID。用户也可以在表示整个分片上传完成的最后一个请求中指定该 ID。或者在用户放弃该分片上传操作时指定该 ID。

- **请求语法**

```
POST /ObjectName?uploadsHTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: signatureValue
```

- **请求头格式**

使用者可以通过一下请求头实现对应操作

名称	描述	是否必须
Cache-Control	可以用来指定请求或响应中的缓存操作。 类型：String 默认值：None	否
Content-Disposition	指定对象的描述性信息。 类型：String 默认值：None	否
Content-Encoding	指定对象的描述性信息采用何种编码方式以及在获取被 Content-Type 头字段引用的 media-type 时采用何种解码方式。 类型：String 默认值：None	否
Content-Type	用来描述对象数据格式的标准 MIME 类型。 类型：String	否

	默认值：binary/octel-stream 限制：仅 MIME 类型	
Expires	对象不再被缓存的时间。 类型：String	否
x-amz-meta-	任何以 x-amz-meta-为前缀的头都被当作用户元数据，它和对象一起存储，当用户获取该对象的时候作为响应的一部分被返回。	否
x-amz-storage-class	对象的存储类型，针对那些在成功完成分片上传后被创建的对象。 类型：String 取值：STANDARD。 默认值为 STANDARD。	否

- **响应结果**

名称	描述
InitiateMultipartUploadResult	包含所有响应结果的容器。 类型：容器 子节点：Bucket，Key，UploadId 父节点：无
Bucket	分片上传对应的 Bucket 的名称。 类型：Sring 父节点：InitiateMultipartUploadResult
Key	分片上传对应的对象名称。 类型：String 父节点：InitiateMultipartUploadResult
UploadId	分片上传 ID 类型：String 父节点：InitiateMultipartUploadResult

- **请求示例**

初始化对象 example-object 的分片上传操作。

```
POST /example-object?uploads HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 197
Connection: close
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<InitiateMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Bucket>example-bucket</Bucket>
<Key>example-object</Key>
<UploadId>VXBsb2FkIElEIGZvciA2aWWpbmcncyBteS1tb3ZpZS5tMnRzIHVwbG9hZA</UploadId>
</InitiateMultipartUploadResult>
```

4.4.6 Upload Part

该接口用于实现分片上传操作中片段的上传。

在上传任何一个分片之前，必须执行错误!未找到引用源。操作来初始化分片上传操作，初始化成功后，OOS 会返回一个上传 ID，这是一个唯一的标识，用户必须在调用 Upload Part 接口时加入该 ID。

分片号 PartNumber 可以唯一标识一个片段并且定义该分片在对象中的位置，范围从 1 到 10000。如果用户用之前上传过的片段的分片号来上传新的分片，之前的分片将会被覆盖。

除了最后一个分片外，所有分片的大小都应该不小于 5M，最后一个分片的大小不受限制。

为了确保数据不会由于网络传输而毁坏，需要在每个分片上传请求中指定 Content-MD5 头，OOS 通过提供的 Content-MD5 值来检查数据的完整性，如果不匹配，则会返回一个错误信息。

- **请求语法**

```
PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Content-Length: Size
Authorization: Signature
```

- **请求参数**

名称	描述	是否必须
PartNumber	标识片段的分片号。整数形式，取值范围是[1,10000]。	是

- **请求头格式**

名称	描述	是否必须
Content-Length	该分片的大小，以字节为单位。 类型：Integer	是

Content-MD5	该分片数据的 128 位采用 base64 编码的 MD5 值。这个头可以用来验证该分片数据是否与原始数据值保持一致。尽管这个值是可选的，我们仍然推荐使用 Content-MD5 机制来执行端到端的一致性校验。 类型：String	否
Expect	如果用户的应用中设置该头为 100-continue，则应用在接收到请求回应之前不会发送请求实体。如果基于头的消息被拒绝，消息的实体也不会被发送。 类型：String 可选值：100-continue	否
x-amz-limit	对像上传限制的速率。 格式为： ● x-amz-limit:rate=*xxx* ● x-amz-limit:concurrency=*xxx* ● x-amz-limit:rate=xxx, concurrency=*xxx* 其中 ● rate 为速率，取值为大于 0 的整数，单位是 KiB/s。当取值是大于 0 小于 128 的整数时，按速率等于 128KiB/s 处理。 ● concurrency 为并发连接数。取值为大于 0 的正整数。	否

- **响应结果**

名称	描述
Etag	片段上传完成时返回的 ETag 值。字符串形式。

- **错误码**

响应代码	描述	HTTP 状态码

NoSuchUpload	指定的分片上传过程不存在，上传 ID 可能非法，分片上传过程可能被终止或者已经完成	404Not Found

- **请求示例**

  执行一次分片上传过程中的片段上传操作，该请求要求包含在 Initial Multipart Upload 操作中获取到的上传 ID。

```
PUT /my-movie.m2ts?partNumber=1&uploadId=VCVsb2FkIElEIGZvciBlbZZpbm
cncyBteS1tb3ZpZS5tMnRzIHVwbG9hZR HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 10485760
Content-MD5: pUNXr/BjKK5G2UKvaRRrOA==
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
***part data omitted***
```

- **响应示例**

  响应中包含 Etag 头，用户需要在最后发送完成分片上传过程请求的时候包含该 Etag 值。

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
ETag: "b54357faf0632cce46e942fa68356b38"
Content-Length: 0
Connection: close
Server: CTYUN
```

4.4.7 Complete Multipart Upload

该接口通过合并之前的上传片段来完成一次分片上传过程。

用户首先初始化分片上传过程，然后通过**错误!未找到引用源。**接口上传所有分片。在成功将一次分片上传过程的所有相关片段上传之后，调用这个接口来结束分片上传过程。当收到这个请求的时候，OOS 会以分片号升序排列的方式将所有片段依次拼接来创建一个新的对象。在这个 Complete Multipart Upload 请求中，用户需要提供一个片段列表。同时，必须确保这个片段列表中的所有片段必须是已经上传完成的，Complete Multipart Upload 操作会将片段列表中提供的片段拼接起来。对片段列表中的每个片段，需要提供该片段上传完成时返回的 ETag 头的值和对应的分片号。

处理一次 Complete Multipart Upload 请求可能需要花费几分钟时间。OOS 在处理这个请求之前会发送一个值为 200 响应头。在处理这个请求的过程中，OOS 每隔一段时间就会发送一个空格字符来防止连接超时。因为一个请求在初始的 200 响应已经发出之后仍可能失败，用户需要检查响应体内容以判断请求是否成功。

注意：如果 Complete Multipart Upload 请求失败，应用程序应该尝试重新发送该请求。

由于 Complete Multipart Upload 请求可能需要花费几分钟时间，所以 OOS 提供了不合并片段也可以读取 Object 内容的功能。在没有调用 Complete Multipart Upload 接口合并片段时，也可以通过调用 Get Object 接口来获取文件内容，OOS 会根据最近一次创建的 uploadId，以分片号升序的方式顺序读取片段内容，返回给客户端。但此时不能返回整个文件的 ETag 值。

- **请求语法**

```
POST /ObjectName?uploadId=UploadId HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: Date
Content-Length: Size
Authorization: signatureValue

<CompleteMultipartUpload>
    <Part>
        <PartNumber>PartNumber</PartNumber>
        <ETag>ETag</ETag>
```

```
    </Part>
...
</CompleteMultipartUpload>
```

- **请求元素**

名称	描述	是否必须
CompleteMultipartUpload	请求的容器。 父节点：无 类型：容器 子节点：1 个或多个 Part 元素	是
Part	一个片段的容器。 父节点：CompleteMultipartUpload 类型：容器 子节点：PartNumber，ETag	是
PartNumber	标识片段的分片号，取值范围是 1~10000。 父节点：Part 类型：Integer	是
ETag	片段上传完成时返回的 Etag 内容。 父节点：Part 类型：String	是

- **响应结果**

名称	描述
CompleteMultipartUploadResult	包含整个响应的容器。 类型：容器 子节点：Location，Bucket，Key，ETag 父节点：无
Location	新创建的对象的 URL 地址。

	类型：URI 父节点：CompleteMultipartUploadResult
Bucket	分片上传对应的对象容器 类型：String 父节点：CompleteMultipartUploadResult
Key	新创建的对象的 Key 类型：String 父节点：CompleteMultipartUploadResult
ETag	Tag 用来标识新创建的对象数据，拥有不同数据的对象，它的 Tag 值也不同。ETag 值是一个不透明的字符串，它可以是也可以不是一个对象数据的 MD5 值，如果 ETag 值不是一个对象的 MD5 值，它将会包含一个或者多个非十六进制字符串，并且/或者包含少于 32 位/多于 32 位的十六进制数字。 类型：String 父节点：CompleteMultipartUploadResult

- **错误码**

响应代码	描述	HTTP 状态码
InvalidPart	一个或者多个指定片段无法找到，片段可能没有被上传，或者指定的 tag 值跟片段的 tag 值不匹配。	400 Bad Request
InvalidPartOrder	片段列表不以升序排列，片段列表必须根据分片号按顺序排列。	400 Bad Request
NoSuchUpload	指定的分片上传过程不存在，上传 ID 可能非法，分片上传过程可能被终止或者已经完成。	404Not Found
InvalidPartSize	如果小于 5M 的分片为 2 片或者 2 片以上，会终止合并分片上传。	400 Bad Request

- **请求示例**

  在 CompleteMultipartUpload 元素中指定了三个片段。

```
POST /example-object?uploadId=AAAsb2FkIElEIGZvciBlbHZpbmcncyWeeS1tb3ZpZS5tMnRzIR
RwbG9hZA HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 391
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

<CompleteMultipartUpload>
    <Part>
        <PartNumber>1</PartNumber>
        <ETag>"a54357aff0632cce46d942af68356b38"</ETag>
    </Part>
    <Part>
        <PartNumber>2</PartNumber>
        <ETag>"0c78aef83f66abc1fa1e8477f296d394"</ETag>
    </Part>
    <Part>
        <PartNumber>3</PartNumber>
        <ETag>"acbd18db4cc2f85cedef654fccc4a4d8"</ETag>
    </Part>
</CompleteMultipartUpload>
```

- **响应示例**

  下面的响应中表示一个对象被拼接成功。

```
HTTP/1.1 200 OK
x-amz-id-2: Uuag1LuByRx9e6j5Onimru9pO4ZVKnJ2Qz7/C1NPcfTWAtRPfTaOFg==
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Connection: close
Server: CTYUN

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

```
<CompleteMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Location>http://example-Bucket.s3.amazonaws.com/example-Object</Location>
    <Bucket>example-Bucket</Bucket>
    <Key>example-Object</Key>
    <ETag>"3858f62230ac3c915f300c664312c11f-9"</ETag>
</CompleteMultipartUploadResult>
```

4.4.8 Abort Multipart Upload

该接口用于终止一次分片上传操作。分片上传操作被终止后，用户不能再通过上传 ID 上传其它片段，之前已上传完成的片段所占用的存储空间将被释放。如果此时任何片段正在上传，该上传过程可能会也可能不会成功。所以，为了释放所有片段所占用的存储空间，可能需要多次终止分片上传操作。

- **请求语法**

```
DELETE /ObjectName?uploadId=UploadId HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: Date
Authorization: signatureValue
```

- **错误码**

响应代码	描述	HTTP 状态码
NoSuchUpload	指定的分片上传过程不存在，上传 ID 可能非法，分片上传过程可能被终止或者已经完成。	404Not Found

- **请求示例**

通过指定上传 ID 来终止一次分片上传操作。

```
DELETE /example-object?uploadId=VXBsb2FkIElEIGZvciBlbHZpbmcncyBteS1tb3ZpZS5tM
nRzIHVwbG9hZ HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g =
```

- **响应示例**

```
HTTP/1.1 204 OK
x-amz-request-id: 996c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 0
```

```
Connection: close
Server: CTYUN
```

4.4.9 List Part

该操作用于列出一次分片上传过程中已经上传完成的所有片段。

该操作必须包含一个通过 Initial Multipart Upload 操作获取的上传 ID。该请求最多返回 1000 个上传片段信息，默认返回的片段数是 1000。用户可以通过指定 max-parts 参数来指定一次请求返回的片段数。如果用户的分片上传过程超过 1000 个片段，响应中的 IsTruncated 字段的值则被设置成 true，并且指定一个 NextPartNumberMarker 元素。用户可以在下一个连续的 List Part 请求中加入 part-number-marker 参数，并把它的值设置成上一个请求返回的 NextPartNumberMarker 值。

- **请求语法**

```
GET /ObjectName?uploadId=UploadId HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: Date
Authorization: signatureValue
```

- **请求参数**

名称	描述	是否必须
uploadId	该 ID 用来标识一个分片上传过程。 类型：String	是
max-parts	设置响应体中返回的片段的最大数目。 类型：String 默认值：1000	是
part-number-marker	指定此次列表的起始片段的分片号，只有比该片段的分片号更高的片段才会被列举出来。 类型：String	是
encoding-type	指定响应中 Key 的编码类型。如果 Key 包含 xml 1.0 标准不支持的字符，可通过设置该参数对响应中的 Key 进行编码。	否

	类型：String 取值：url，字母不区分大小写。	

- **响应结果**

名称	描述
ListPartsResult	包含整个响应的容器。 类型：容器 子节点：Bucket、EncodingType、Key、UploadId、Initiator、Owner、StorageClass、PartNumberMarker、NextPartNumberMarker、MaxParts、IsTruncated、Part 父节点：无
Bucket	分片上传对应的对象名称。 类型：String 父节点：ListPartsResult
EncodingType	Key 的编码类型。 类型：String 父节点：ListPartsResult
Key	新创建的对象的 Key。 类型：String 父节点：ListPartsResult
UploadId	分片上传 ID。 类型：String 父节点：ListPartsResult
Initiator	指定执行此次分片上传过程的用户账户。 子节点：ID，DisplayName 类型：容器 父节点：ListPartsResult
ID	OOS 账户的 ID。

	类型：String 父节点：Initiator
StorageClass	对象的存储类型：STANDARD。 类型：String 父节点：ListPartsResult
PartNumberMarker	列表起始位置的片段的分片号。 类型：Integer 父节点：ListPartsResult
NextPartNumberMarker	当此次请求没有将所有片段列举完时，此元素指定列表中的最后一个片段的分片号。此分片号用于作为下一次连续列表请求的 part-number-marker 参数的值。 类型：Integer 父节点：ListPartsResult
MaxParts	响应中片段的最大数目。 类型：Integer 父节点：ListPartsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为 true 则表示没有全部列出。如果分片上传过程的片段数超过了 MaxParts 元素指定的最大数，则会导致一次列表请求无法将所有片段数列出 类型：Boolean 父节点：ListPartsResult
Part	与某个片段对应的容器，响应中可能包含 0 个或多个 Part 元素 子节点：PartNumber, LastModified, ETag, Size 类型：String 父节点：ListPartsResult
PartNumber	标识片段的分片号 类型：Integer

	父节点：Part
LastModified	片段上传完成的日期 类型：Date 父节点：Part
ETag	片段上传完成时返回的 ETag 值 类型：String 父节点：Part
Size	片段的数据大小 类型：Integer 父节点：Part

- **错误码**

响应代码	描述	HTTP 状态码
NoSuchUpload	指定的分片上传过程不存在，上传 ID 可能非法，分片上传过程可能被终止或者已经完成。	404Not Found

- **请求示例**

假设用户上传了一系列以 1 开头的有连续分片号的片段，下面的 List Part 请求指定了 max-parts 和 part-number-marker 查询参数。此次请求列出了紧随片段 1 的两个片段，从请求响应体中可以获取到片段 2 和片段 3。如果有更多的片段存在，则片段列表操作没有完成，响应体中将会包含值为 true 的 IsTruncated 元素。同时，响应体中还会包含值为 3 的 NextPartNumberMarker 元素，这个值用来指定在下一次连续的列表操作中 part-number-marker 参数的值。

```
GET /example-object?uploadId=XXBsb2FkIElEIGZvciBlbHZpbmcncyVcdS1tb3ZpZS5tMnRzEEEw
bG9hZA&max-parts=2&part-number-marker=1 HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- 响应示例

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 1014
Connection: close
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<ListPartsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>example-bucket</Bucket>
  <EncodingType></EncodingType>
  <Key>example-object</Key>
  <UploadId>XXBsb2FkIElEIGZvciBlbHZpbmcncyVcdS1tb3ZpZS5tMnRzEEEwbG9hZA</UploadId>
  <Initiator>
    <ID>mailaddress@ctyun.cn</ID>
    <DisplayName>umat-user-11116a31-17b5-4fb7-9df5-b288870f11xx</DisplayName>
  </Initiator>
  <Owner>
    <ID></ID>
    <DisplayName></DisplayName>
  </Owner>
  <StorageClass>STANDARD</StorageClass>
  <PartNumberMarker>1</PartNumberMarker>
  <NextPartNumberMarker>3</NextPartNumberMarker>
  <MaxParts>2</MaxParts>
  <IsTruncated>true</IsTruncated>
  <Part>
    <PartNumber>2</PartNumber>
    <LastModified>2010-11-10T20:48:34.000Z</LastModified>
    <ETag>"7778aef83f66abc1fa1e8477f296d394"</ETag>
    <Size>10485760</Size>
  </Part>
  <Part>
    <PartNumber>3</PartNumber>
    <LastModified>2010-11-10T20:48:33.000Z</LastModified>
```

```
    <ETag>"aaaa18db4cc2f85cedef654fccc4a4x8"</ETag>
    <Size>10485760</Size>
  </Part>
</ListPartsResult>
```

4.4.10 Copy Part

可以将已经存在的 Object 作为分段上传的片段，拷贝生成一个新的片段。需要指定请求头 x-amz-copy-source 来定义拷贝源。如果想拷贝源 Object 中的一部分，可以加请求头 x-amz-copy-source-range。

- **请求语法**

```
PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
x-amz-copy-source: /source_bucket/sourceObject
x-amz-copy-source-range:bytes=first-last
x-amz-copy-source-if-match: etag
x-amz-copy-source-if-none-match: etag
x-amz-copy-source-if-unmodified-since: time_stamp
x-amz-copy-source-if-modified-since: time_stamp
Date: Date
Authorization: SignatureValue
```

- **请求头**

名称	描述	是否必须
x-amz-copy-source	指定源 Bucketname 和 Objectname，用斜杠/分隔。 类型：String	是
x-amz-copy-source-range	要从源 object 拷贝的 bytes 范围。Rage 值的格式是 bytes=第一个字节-最后一个字节。第一个字节从 0 开始。例如要拷贝前 10 个字节，bytes=0-9。只允许对大于 5G 的源 object 进行部分拷贝的操作。 如果要拷贝整个 object，不需要这个头。 类型：String	否

x-amz-copy-source-if-match	如果对象的实体标签与给定标签匹配则执行拷贝对象的操作，否则请求返回 412 错误码。	否
x-amz-copy-source-if-none-match	如果对象实体标签和指定实体标签不同则执行拷贝操作，否则返回 412 错误码。	否
x-amz-copy-source-if-unmodified-since	如果对象在指定时间点之后没有修改过则执行拷贝操作，否则返回 412 错误码。	否
x-amz-copy-source-if-modified-since	如果对象在指定时间点之后被修改过则执行拷贝操作，否则返回 412 错误码。	否

- **响应结果**

名称	描述
CopyPartResult	包含整个响应的容器。 类型：容器 父节点：无
ETag	新分片的 ETag。 类型：String 父节点：CopyPartResult
LastModified	分片的最后修改时间。 类型：String 父节点：CopyPartResult

- **请求示例**

下面的请求通过从源 Object 中指定范围，拷贝生成一个新片段。

```
PUT /newobject?partNumber=2&uploadId=VCVsb2FkIElEIGZvciBlbZZpbm
cncyBteS1tb3ZpZS5tMnRzIHVwbG9hZR HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 11 Apr 2011 20:34:56 GMT
x-amz-copy-source: /source-bucket/sourceobject
x-amz-copy-source-range:bytes=500-6291456
```

```
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- 响应示例

```
HTTP/1.1 200 OK
x-amz-request-id: 996c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Connection: close
Server: CTYUN

<CopyPartResult>
    <LastModified>2009-10-28T22:32:00</LastModified>
    <ETag>9b2cf535f27731c974343645a3985328</ETag>
</CopyPartResult>
```

4.4.11 Delete Multiple Objects

批量删除 Object 功能支持用一个 HTTP 请求删除一个 Bucket 中的多个 Object。如果你知道你想删除的 object 名字，此功能可以批量删除这些 object，而不用发送多个单独的删除请求。

批量删除请求包含一个不超过 1000 个 object 的 XML 列表。在这个 xml 中，你需要指定要删除的 Object 的名字。对于每个 Object，OOS 都会返回删除的结果，成功或者失败。注意，如果请求中的 Object 不存在，那么 OOS 也会返回删除成功。

批量删除功能支持两种格式的响应，全面信息和简明信息。默认情况下，OOS 在响应中会显示全面信息，即包含每个 object 的删除结果。在简明信息模式下，OOS 只返回删除出错的 Object 的结果。对于成功删除的 Object，在响应中将不返回任何信息。

说明：在删除不存在的对象时，也会返回删除成功。

最后，批量删除功能必须使用 Content-MD5 请求头，OOS 使用此头来保证请求体在传输过程中没有被修改。

- **请求语法**

```
POST /?delete HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
Content-Length: Length
Content-MD5: MD5
Authorization: SignatureValue

<?xml version="1.0" encoding="UTF-8"?>
<Delete>
    <Quiet>true</Quiet>
    <Object>
        <Key>Key</Key>
    </Object>
    <Object>
        <Key>Key</Key>
    </Object>
    ...
</Delete>
```

- **请求头**

名称	描述	是否必须
Content-MD5	Base64 编码，128 位的 MD5，这个请求头必须被使用，以保证数据在传输过程中没有被篡改。参考 RFC 1864。 类型：String	是
Content-Length	请求体的长度，参考 RFC 2616 类型：String	是

- **请求元素**

名称	描述	是否必须
Delete	包含整个请求的容器。 类型：容器 父节点：无	是
Quiet	使用简明信息模式来返回响应，有效值为：true 和 false，默认为 false。 类型：Boolean 父节点：Delete	否
Object	包含被删除 object 的容器。 类型：容器 父节点：Delete	是
Key	被删除 object 的名称。 类型：String 父节点：Object	是

- **响应结果**

名称	描述
DeleteResult	包含整个响应的容器。

	类型：容器 父节点：无
Deleted	成功删除的容器，包含成功删除的 object。 类型：容器 父节点：DeleteResult
Key	尝试删除的 object 名。 类型：String 父节点：Deleted，或 Error
Error	删除失败的容器，包含删除失败的 object 信息。 类型：容器 父节点：DeleteResult
Code	删除失败的状态码。 类型：String 父节点：Error 取值：AccessDenied, InternalError
Message	错误的描述。 类型：String 父节点：Error

- **请求示例**

批量删除一些 object，有些删除成功，有些失败（例如没有权限删除）。

```
POST /?delete HTTP/1.1
Host: example-bucket.oos-js.ctyunapi.cn
Date: Mon, 11 Apr 2011 20:34:56 GMT
Content-MD5: p5/WA/oEr30qrEEl21PAqw==
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Length: 125
Connection: Keep-alive

<Delete>
```

```
    <Object>
        <Key>sample1.txt</Key>
    </Object>
    <Object>
        <Key>sample2.txt</Key>
    </Object>
</Delete>
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 996c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Server: CTYUN
Content-Length: 251

<?xml version="1.0" encoding="UTF-8"?>
<DeleteResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Deleted>
        <Key>sample1.txt</Key>
    </Deleted>
    <Error>
        <Key>sample2.txt</Key>
        <Code>AccessDenied</Code>
        <Message>AccessDenied</Message>
    </Error>
</DeleteResult>
```

4.4.12 断点续传

通过 MultipleUpload 类以及文件上传请求 UploadFileRequest 类，实现基于分段上传的断点续传的功能。

- **请求参数**

名称	描述
EnableCheckpoint	是否开启断点续传功能。 默认：关闭
PartSize	每个分段的大小。若 partSize 小于 5MiB，则会将 partSize 调整至 5MiB。 默认为 5MiB。
UploadFile	上传的本地文件。
checkpointFile	记录本地分片上传结果的文件，默认为上传的本地文件同路径下 uploadFile.ucp。
objectMetadata	文件的元数据。
ProgressListener	上传状态监听器。
TaskNum	分片上传并发线程数，默认为 1，最多为 1000。

用 checkpoint 文件来记录所有分片的状态。上传过程中的进度信息会保存在该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。checkpoint 文件默认与待上传的本地文件同目录，为 uploadFile.ucp。

- **Java 上传代码示例**

```
public static void main(String[] args) throws IOException {

    // 创建 client

    AmazonS3 oos = new AmazonS3Client(new AWSCredentials() {
        @Override
        public String getAWSSecretKey() {
            return secretKey;
```

```
        }
        @Override
        public String getAWSAccessKeyId() {
            return accessKey;
        }
    });
oos.setEndpoint(endpoint);
        try {
            // 通过 UploadFileRequest 设置多个参数
            UploadFileRequest request = new UploadFileRequest(bucketName, key);
            // 上传的本地文件
            request.setUploadFile(uploadFile);
            // 分片上传并发线程数，默认为 1，最多为 1000
            request.setTaskNum(1);
/* 每个分片的大小，默认 5MiB，若 partSize 小于 5MiB，除了最后一个分片以外，会将 partSize 调整至
5MiB */
            request.setPartSize(5*1024*1024);
            // 开启断点续传功能，默认关闭
            request.setEnableCheckpoint(true);
/* 记录本地分片上传结果的文件。开启断点续传功能时需要设置此参数，上传过程中的进度信息会保存在该
文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。*/
// 上传完成后，该文件会被删除。默认与待上传的本地文件同目录，为 uploadFile.ucp
            request.setCheckpointFile("<yourCheckpointFile>");
            MultipleUpload multiUpload = new MultipleUpload(request, oos);
            // 断点续传上传
            CompleteMultipartUploadResult result = multiUpload.upload();
System.out.println("Upload complete. Upload object name:" + result.getKey());
        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means your
request made it "
```

```
                        + "to OOS, but was rejected with an error response for some
reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("OOS Error Code:   " + ase.getErrorCode());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means the client
encountered "
                        + "a serious internal problem while trying to communicate with
OOS, "
                        + "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

4.4.13 POST Object

POST 操作使用 HTML 表单将对象上传到指定的 Bucket。POST 是另一种形式的 PUT 操作，POST 可以让使用者通过 Browser-based 的方式，将对象上传到指定 bucket 中。PUT 的参数是通过 HTTP Header 提交的，而 POST 通过使用 multipart/form-data 编码的消息体中的字段进行提交。用户必须对操作的 Bucket 有写权限。OOS 不存储部分对象：如果收到成功的响应，那么对象就是存储成功了。

OOS 是一个分布式系统。如果 OOS 同时收到针对同一个对象的多个写请求，所有的请求都可以成功执行，前面的写请求上传的文件会被最后一个覆盖。

为了保证数据在网络传输过程中没有损坏，可以使用 Content-MD5 字段进行校验。如果请求参数中有 Content-MD5，OOS 将会计算用户提交的对象的 MD5 值。如果计算出的值与用户提供的值不一致，OOS 将会返回一个错误给用户。或者，用户可以在上传对象到 OOS 时计算对象的 MD5 值，并与 OOS 在响应中返回的 ETag 进行比较。ETag 是对象内容的 MD5 值，不包括 metadata。

- **请求语法**

```
POST /HTTP/1.1
Host: BucketName.oos-js.ctyunapi.cn
User-Agent: browser_data
Accept: file_types
Accept-Language: Regions
Accept-Encoding: encoding
Accept-Charset: character_set
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=9431149156168
Content-Length: Length
--9431149156168
Content-Disposition: form-data; name="key"
Key
--9431149156168
Content-Disposition: form-data; name="success_action_redirect"
```

```
success_redirect
--9431149156168
Content-Disposition: form-data; name="Content-Type"
content_type
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-uuid"
uuid
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-tag"
Metadata
--9431149156168
Content-Disposition: form-data; name="AWSAccessKeyId"
access-key-id
--9431149156168
Content-Disposition: form-data; name="Policy"
encoded_policy
--9431149156168
Content-Disposition: form-data; name="Signature"
signature=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"
Content-Type: image/jpeg
file_content
--9431149156168
Content-Disposition: form-data; name="submit"
Upload to OOS
--9431149156168–
```

- **表单字段**

可以使用以下字段

名称	描述	是否必须
AWSAccessKeyId	Bucket 拥有者的 AWS 访问密钥 ID，该拥有者将授予匿名用户对满足策略中一组约束的请求的访问权限。如果请求包含 Policy，则此字段为必填字段。	否

	类型：String	
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	特定于 REST 的请求头。有关更多信息，请参阅 PUT Object。 类型：String	否
file	文件或文本内容。文件或文本内容必须是 Form 表单的最后一个字段。一次只能上传一个文件。 类型：文件或文本内容	是
key	上传对象的名称。 ${*filename*}为用户提供的文件名。例如，如果用户 Betty 上传的文件名为 lolcatz.jpg，字段值指定为 /user/betty/${filename},那么保存的对象名称将会是 /user/betty/lolcatz.jpg。 类型：String	是
policy	描述请求中允许的内容的安全策略。不带安全策略的请求被认为是匿名的，只能对公开的 Bucket 进行操作。 Policy 的设置方法参见 **Policy 字段的构造**。 类型：String	否
signature	使用 AWSAccessKeyId 对应的秘钥对 policy 的签名。如果请求包含 Policy，则此字段为必填字段。 signature = Base64(HMAC-SHA1(YourSecrectKey,policy)),具体计算方法请查看 Signature 字段的构造步骤。	条件
success_action_redirect,redirect	上传成功后客户端重定向到的 URL。OOS 将 Bucket、对象名和 etag 值作为查询字符串参数附加到 URL。	否

	如果未指定 success_action_redirect，OOS 将返回在 success_action_status 字段中指定的空文档类型。 如果 OOS 无法识别用户提供的 URL，将忽略该字段。 如果上传失败，OOS 将显示错误且不会执行重定向操作。 类型：String **注意**：redirect 将在可能会移除，建议使用 success_action_redirect。	
success_action_status	如果没有指定 success_action_redirect，上传成功后状态代码将返回到客户端。 有效值为 200、201 或 204（默认）。 如果值被设置为 200 或 204，OOS 将返回一个空文档和一个 200 或 204 状态代码。 如果值被设置为 201，OOS 将返回一个 XML 文档和一个 201 状态代码。有关 XML 文档 内容的信息，请参阅 POST 对象。 如果没有设置值或者设置了无效的值，OOS 将返回一个空文档和一个 204 状态代码。 **注意**：某些版本的 AdobeFlashplayer 无法正确处理使用空白正文的 HTTP 响应。要通过 AdobeFlash 支持上传，建议您将 success_action_status 设置为 201。	否
x-amz-storage-class	数据的存储类型，默认采用动态副本模式存储。用户也可选择使用标准模式进行存储。对于那些不太重要，可以重复生成的数据，用户可以选择减少冗余策略来降低成本。	否

	类型: String 取值: STANDARD。	
x-amz-meta-*	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和对象一起被存储并返回。更多信息请参考 PUT Object。 类型：String 默认值：无	否
x-amz-website-redirect-location	如果 Bucket 配置为网站，重定向对这个对象的请求到相同 Bucket 的另外一个对象或者到一个其他的 URL。OOS 会保存这个值到对象的 metadata。对象的 metadata 信息请参考 Object Key 和 Metadata 部分。 下面这个例子表示请求头设置重定向到相同 Bucket 的另一个对象(anotherPage.html) x-amz-website-redirect-location: /anotherPage.html。 重定向到其他网站的例子： x-amz-website-redirect-location: http://www.ctyun.cn/ 注意：这个值必须以“/”、https://或 http://开头，且长度不能超过 2KiB。	否

- **响应头**

除了通用的响应头以外，本操作可以包括以下响应头。

名称	描述
x-amz-expiration	如果对象设置了过期时间（参考 PUT Bucket Lifecycle 章节），响应头会增加过期信息。过期信息包括过期日期和 rule-id 的 key 和 value。rule-id 的 value 是经过 URL 编码的。 类型：String

success_action_redirect , redirect	上传成功重定向的 URL。 类型：String

- **响应体**

名称	描述
Bucket	存储 Object 的 Bucket 名称。 类型：字符串 父元素：PostResponse
ETag	ETag 是对象内容经过 MD5 哈希后得到的值。用户可以在 GET 请求中使用 If-Match 请求头。ETag 仅反映对象内容的变化，不包括元数据。 类型：String 父元素：PostResponse
Key	对象名称。 类型：String 父元素：PostResponse
Location	对象的 URL。 类型：String 父元素: PostResponse

- **请求示例**

```
POST / HTTP/1.1
Content-Length: 4
Host: BucketName.oos-js.ctyunapi.cn
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
Content-Type: text/plain
Expect: the 100-continue HTTP status code
ObjectContent
```

- **响应结果**

```
HTTP/1.1 200 OK
x-amz-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2009 17:50:00 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Content-Length: 0
Connection: close
Server: OOS
```

说明

- 表单声明包含三个部分：action、method 和 enctype。如果这些值当中的任意一个设置不正确，请求将失败。action 指定处理请求的 URL，必须将它设置为 Bucket 的 URL。例如，如果 Bucket 的名称是“BucketName”，则 URL 为“http://BucketName.oos-js.ctyunapi.cn/”。
- method 必须是 POST。enctype 设置为“multipart/form-data”。

```
<form action="http:// BucketName.oos-js.ctyunapi.cn /"
method="post"enctype="multipart/form-data">
</form>
```

- **Policy 字段的构造**

Policy 是使用 UTF-8 和 Base64 编码的 JSON 文档，它指定了请求必须满足的条件并且用于对内容进行身份验证。根据您设计策略文档的方式，您可以对每次上传、每个用户、所有上传或根据其他能够满足您需要的设计来使用它们。

下面是一个简单的 Policy 示例

```
{
    "expiration": "2007-12-01T12:00:00.000Z",
    "conditions": [
        {
            "bucket": "johnsmith"
        },
        [
            "starts-with",
            "$key",
            "user/eric/"
        ]
```

```
    ]
}
```

策略文档由过期(expiration)和条件(conditions)两部分构成。

- 过期

过期元素采用 ISO 8601UTC 日期格式来指定策略的过期日期。例如，“2007-12-01T12:00:00.000Z”指定策略在 2007 年 12 月 1 日午夜 UTC 之后失效。在策略文档中过期字段是必需的。

- 条件

策略文档中的条件验证上传的对象的内容。您在表单中指定的每个表单字段（AWSAccessKeyId、签名、文件、策略和带 x-ignore-前缀的字段名称除外）必须包含在条件列表中。如果您有多个具有相同名称的字段，使用逗号进行分隔。例如，如果您有两个名为“x- amz-meta-tag”的字段，第一个字段的值为“Ninja”，第二个字段的值为“Stallman”，您可以将策略文档设置为 Ninja,Stallman。

应该针对扩展的字段执行前缀匹配。例如，如果您将对象名称字段设置为 user/betty/${filename}，您的策略可能是["starts-with", "$key", "user/betty/"]。请勿输入["starts-with", "$key", "user/betty/${filename}"]。

元素名称	说明
bucket	指定上传内容允许的 Bucket。 支持精确匹配和 starts-with。
content-length-range	指定已上传内容允许的最小和最大大小。 支持范围匹配。
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	特定于 REST 的标头。 支持精确匹配和 starts-with。
Key	已上传的密钥的名称。 支持精确匹配和 starts-with。
success_action_redirect, redirect	上传成功后客户端重定向到的 URL。 支持精确匹配和 starts-with。

success_action_status	如果没有指定 success_action_redirect，上传成功后状态代码将返回到客户端。 支持精确匹配。
其他以 x-amz-meta- 为前缀的字段名称	特定于用户的元数据。 支持精确匹配和 starts-with。

注意：如果您的工具包添加了其他字段（例如，Flash 添加了文件名），您必须将它们添加到策略文档。如果您可以控制此功能，将 x-ignore-添加为字段的前缀以使 OOS 忽略此功能并使其不影响此功能的未来版本。

条件匹配

下表介绍条件匹配类型。尽管您必须为您在表单中指定的每个表单字段指定一个条件，您也可以通过为某个表单字段指定多个条件来创建更复杂的匹配条件。

条件	说明
精确匹配	精确匹配将验证字段是否匹配特定的值。此示例指示 bucket 必须设置为 BucketName： {"bucket": "BucketName"} 也可写为： ["eq", "bucket": "BucketName"]
Starts With	如果值必须从某个特定的值开始，请使用 starts-with。本示例指示密钥必须从 user/betty 开始： ["starts-with", "$key", "user/betty/"]
匹配任何内容	要配置策略以允许字段中的任何内容，请使用 starts-with 和一个空值。本示例允许任何 success_action_redirect： ["starts-with", "$success_action_redirect", ""]
指定范围	对于接受范围的字段，请使用逗号来分隔上限和下限值。本示例允许 1 到 10 MiB 的文件大小： ["content-length-range", 1048579, 10485760]

字符串转义

转义序列	描述
\\	反斜杠
\$	美元符号
\b	退格键
\f	换页
\n	新建行
\r	回车
\t	水平选项卡
\v	垂直选项卡
\uxxxx	所有 Unicode 字符

Signature 字段的构造步骤

步骤	说明
1	使用 UTF-8 对策略进行编码。
2	使用 Base64 对这些 UTF-8 字节进行编码。
3	使用 HMAC SHA-1，通过您的秘密访问密钥签署策略。
4	使用 Base64 对 SHA-1 签名进行编码。

4.4.14 OPTIONS Object

浏览器可以向 OOS 发送预检请求，来判断其是否可以发送特定源、HTTP 方法和头的实际请求。当浏览器发送预检请求时，OOS 根据 bucket 的跨域配置来返回响应信息。如果 bucket 没有配置跨域，那么 OOS 返回响应 403 Forbidden。

- **请求语法**

```
OPTIONS /ObjectName HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Origin: Origin
Access-Control-Request-Method: HTTPMethod
Access-Control-Request-Headers: RequestHeader
```

- **请求头格式**

名称	描述	是否必需
Origin	标示访问OOS的跨域请求的源 例如：http://www.ctyun.cn. 类型: String	是
Access-Control-Request-Method	标示在实际请求中，将被用到的HTTP方法。 类型: String	是
Access-Control-Request-Headers	在实际请求中将被发送的HTTP请求头，用逗号分隔。 类型: String	否

- **响应头**

名称	描述
Access-Control-Allow-Origin	用户请求中发送的源。如果此源不被允许访问，那么 OOS 不会返回此响应头。 类型: String
Access-Control-Max-Age	预检请求可以被缓存的时间，单位为秒。

	类型: String
Access-Control-Allow-Methods	用户请求中发送的 HTTP 方法。如果此方法不被允许，那么 OOS 不会返回此响应头。 类型: String
Access-Control-Allow-Headers	在实际请求中，浏览器可以发送的 HTTP 请求头列表，以逗号分隔。如果没有任何请求头被允许，OOS 不会返回此响应头，也不会返回任何以 Access-Control 开头的响应头。 类型: String
Access-Control-Expose-Headers	在实际响应中，JavaScript客户端可以访问的响应头列表，以逗号分隔。 类型: String

- **请求示例**

浏览器可以向 OOS 发送预检请求，来判断其是否可以从源 http://www.ctyun.cn 向名为 examplebucket 的 bucket，发送 PUT 请求。

```
OPTIONS /exampleobject HTTP/1.1
Host: examplebucket.oos-js.ctyunapi.cn
Origin: http://www.ctyun.cn
Access-Control-Request-Method: PUT
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: BDC4B83DF5096BBE
Date: Wed, 21 Aug 2012 23:09:55 GMT
Access-Control-Allow-Origin: http://www.ctyun.cn
Access-Control-Allow-Methods: PUT
Access-Control-Expose-Headers: x-amz-request-id
Connection: close
Server: OOS
```

4.4.15 生成共享链接

- **生成共享链接**

对于私有或只读 Bucket，可以通过生成 Object 的共享链接的方式，将 Object 分享给其他人，同时可以在链接中设置限速以对下载速度进行控制。在 SDK 中调用 AmazonS3 中的 generatePresignedUrl(String bucketName, String key, Date expiration)方法，可以生成共享链接 URL。参数分别是 Bucket 名称，Object 名称和过期时间，如果过期时间传 Null 的话，默认的过期时间是 15 分钟。超出过期时间后，共享链接失效，不能再通过链接下载 Object。以下是一个共享链接的例子：

```
http://mybucket.oos-
js.ctyunapi.cn/example/example.zip?Signature=QzVGqUY3n/ypmdQK6YD4h7I3bic%3D&AWSAccessK
eyId=3073c6a8ede206655738&Expires=1586226782
```

- **共享链接限速**

如果需要为链接设置下载速度限制，需要新增加自定义参数“x-amz-limitrate”，调用 generatePresignedUrlRequest.addRequestParameter("x-amz-limitrate", value)方法，value 值为限速带宽(单位 KiB/s)，将参数加到 generatePresignedUrlRequest 对象中，参与共享链接生成，以下为增加了下载速度限制的共享链接的示例：

```
http://oos-js.ctyunapi.cn/test-
20180604/6aa3df83gw1f35nhhp70pj20gj0r046l.jpg?Signature=8l7F/pabWm2%2Bi8iXyExZIXm/eGY%
3D&AWSAccessKeyId=08f17977afa1a87736ac&Expires=1528438576&x-amz-limitrate=2048
```

4.4.16 HEAD Object

Head 操作用于获取对象的元数据信息，而不返回数据本身。当只希望获取对象的属性信息时，可以使用此操作。

- **请求语法**

```
HEAD /ObjectName HTTP/1.1
Host: bucketName.oos-js.ctyunapi.cn
Date: date
Authorization: SignatureValue
```

- **响应头**

变量	描述
x-amz-expiration	如果对象被配置了到期时间，那么 OOS 返回此响应头。这个响应头包含键值对 expiry-date 和 rule-id。rule-id 的值是 URL 编码的。

- **请求示例**

返回对象 my-image.jpg。

```
HEAD /my-image.jpg HTTP/1.1
Host: bucket.oos-js.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- **响应示例**

```
HTTP/1.1 200 OK
x-amz-request-id: 318BC8BC148832E5
Date: Mon, 03 Sep 2012 22:32:00 GMT
Last-Modified: Sat, 01 Sep 2012 17:50:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: image/ipeg
```

```
Connection: close
Server: CTYUN
```

4.5 关于 AccessKey 的操作

OOS 的 IAM 服务端地址请参加 **Endpoint 列表**。

4.5.1 CreateAccessKey

此操作用来创建一对普通的 AccessKey 和 SecretKey，默认的状态是 Active。只有主 key 才能执行此操作。

为保证账户的安全，SecretKey 只在创建的时候会被显示。请把 key 保存起来，比如保存到一个文本文件中。如果 SecretKey 丢失了，你可以删除 AccessKey，并创建一对新的 key。默认情况下，每个账户最多创建 10 个 AccessKey。

- **请求语法**

```
POST/ HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: date
Authorization: SignatureValue

Action=CreateAccessKey
```

- **请求参数**

名称	描述	是否必须
Action	CreateAccessKey。	是

- **响应结果**

名称	描述
UserName	用户名称。
AccessKeyId	生成的 AccessKey。
Status	默认生成的是 Active 状态。
SecretAccessKey	生成的 SecretKey。

IsPrimary	是否是主 key： ● true：是主 key； ● false：不是主 key，是普通 key。

- **请求示例**

```
POST / HTTP/1.1
Host:oos-js-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

Action=CreateAccessKey
```

- **响应示例**

```
<CreateAccessKeyResponse>
    <CreateAccessKeyResult>
    <AccessKey>
      <UserName>Bob</UserName>
      <AccessKeyId>2d1d5625f6202b08c8db</AccessKeyId>
      <Status>Active</Status>
      <SecretAccessKey>18ae0013ee15f6f879b26ca18e8a</SecretAccessKey>
      <IsPrimary>false</IsPrimary>
    </AccessKey>
</CreateAccessKeyResult>
<ResponseMetadata>
    <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
</ResponseMetadata>
</CreateAccessKeyResponse>
```

4.5.2 DeleteAccessKey

此操作用来删除密钥对 AccessKey/SecretKey。只有主 key 才能执行此操作。

- **请求语法**

```
POST/ HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Date
Authorization: SignatureValue

Action=DeleteAccessKey&AccessKeyId=accessKeyId
```

- **请求参数**

名称	描述	是否必须
Action	值是 DeleteAccessKey。	是
AccessKeyId	要删除的 AccessKey。	是

- **请求示例**

```
POST / HTTP/1.1
Host:oos-js-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

Action=DeleteAccessKey&AccessKeyId=2d1d5625f6202b08c8db
```

- **响应示例**

```
<DeleteAccessKeyResponse>
    <ResponseMetadata>
        <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
    </ResponseMetadata>
</DeleteAccessKeyResponse>
```

4.5.3 UpdateAccessKey

此操作用来更新 AccessKey 的状态，或将普通 key 设置成为主 key，反之亦然。只有主 key 才能执行此操作。

- **请求语法**

```
POST/ HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Date
Authorization: SignatureValue

Action=UpdateAccessKey&AccessKeyId=accessKeyId&Status=status&IsPrimary=isPrimary
```

- **请求参数**

名称	描述	是否必须
Action	值是 UpdateAccessKey。	是
AccessKeyId	要更改状态的 AccessKey。	是
Status	Key 的状态： ● Active：启用； ● Inactive：禁止。	是
IsPrimary	是否是主 key： ● true：是主 key； ● false：不是主 key，是普通 key。	否

- **请求示例**

```
POST / HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

Action=UpdateAccessKey&AccessKeyId=2d1d5625f6202b08c8db&Status=active&IsPrimary=false
```

- **响应示例**

```
<UpdateAccessKeyResponse>
    <ResponseMetadata>
        <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
    </ResponseMetadata>
</UpdateAccessKeyResponse>
```

4.5.4 ListAccessKey

此操作用来列出账户下的主 key 和普通 key。只有主 key 才能执行此操作。可以通过 MaxItems 参数指定返回的结果数量，默认返回 100 个 key。可以通过 Marker 参数设置返回的起始位置，该参数可以从前一次请求的响应体中获得。为保证账户安全，list 操作时，不会返回 SecretKey。

- **请求语法**

```
POST/  HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Date
Authorization: SignatureValue

Action=ListAccessKey
```

- **请求参数**

名称	描述	是否必须
Action	值是 ListAccessKey。	是
MaxItems	设置返回结果集的最大数量，如果实际的 key 的数量超过了 MaxItem，那么在响应结果中，IsTruncated 的值是 true。	否
Marker	可以通过 Marker 参数设置返回的起始位置，该参数可以从前一次请求的响应体中获得。该参数可选。	否

- **请求示例**

```
POST / HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g

Action=ListAccessKey&MaxItem=100&Marker=2d1d5625f6202b08c8db
```

- **返回结果**

名称	描述
AccessKeyMetadata	Access Key 元数据的 list
IsTruncated	返回的结果是否被截断，如果是 true，表示还有结果没有返回，可以通过 Marker 参数，继续请求，以便获得后续的数据。
Marker	如果 IsTruncated 是 true，那么会返回 Marker，可以用做下次请求的参数。

- **响应示例**

```
<ListAccessKeysResponse>
    <ListAccessKeysResult>
    <UserName>Bob</UserName>
    <AccessKeyMetadata>
    <member>
        <UserName>Bob</UserName>
        <AccessKeyId>2d1d5625f6202b08c8db</AccessKeyId>
        <Status>Active</Status>
        <IsPrimary>true</IsPrimary>
    </member>
    <member>
    <UserName>Bob</UserName>
        <AcceseyId>2d1d5625f6202b08c8dd</AccessKeyId>
        <Status>Inactive</Status>
        <IsPrimary>false</IsPrimary>
        </member>
    </AccessKeyMetadata>
    <IsTruncated>false</IsTruncated>
    </ListAccessKeysResult>
<ResponseMetadata>
    <RequestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</RequestId>
</ResponseMetadata>
</ListAccessKeysResponse>
```

4.5.5 STS 临时授权访问

OOS 为用户提供临时授权访问。STS（Security Token Service）是为云计算用户提供临时访问令牌的 Web 服务。通过 STS，可以为第三方应用或用户颁发一个自定义时效的访问凭证。第三方应用或用户可以使用该访问凭证直接调用 OOS API，或者使用 OOS 提供的 SDK 来访问 OOS API。

临时授权访问 OOS API 时，用户需要将安全令牌（SessionToken）携带在请求 header 中或者以请求参数的形式放入 URI 中，标头为 `X-Amz-Security-Token`。

- **请求语法**

```
POST/?Action=GetSessionToken&DurationSeconds=seconds HTTP/1.1
Host: oos-js-iam.ctyunapi.cn
Date: Date
Authorization: SignatureValue
```

- **请求参数**

名称	描述	是否必须
Action	值是GetSessionToken。	是
DurationSeconds	令牌的有效期限，单位秒，范围 15 分钟至 36 小时。若用户没有设置有效期限，报错 400 InvalidDurationSeconds。	是

- **响应结果**

响应头	描述
Credentials	用户访问凭证信息
SessionToken	安全令牌
AccessKeyId	临时访问 AK
SecretAccessKey	临时访问 SK
Expiration	过期时间
RequestId	请求ID

- 请求示例

```
POST /?Action=GetSessionToken&DurationSeconds=7200 HTTP/1.1
Host:oos-js-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS 81ebc16ddc8d2af82c90:thdUi9VAkzhkniLj96JIrOPGi0g
```

- 响应示例

```
<GetSessionTokenResponse>
    <GetSessionTokenResult>
        <Credentials>
            <SessionToken>53732f2313e795ba5a649c40d64441b618f194d7c0c08
            33c337dbc954b113eb37b66abea6751b890320fc5f7253482e87a0ca50f7bcb
            221fdc9f4620e232721e
            </SessionToken>
            <AccessKeyId>sts.1310458e7d0d73109ba7</AccessKeyId>
            <SecretAccessKey>9af52496afb6c0f5ba6542
            ec1bec4e52c19093e2
            </SecretAccessKey>
            <Expiration>2018-10-29 17:38:21</Expiration>
        </Credentials>
    </GetSessionTokenResult>
    <ResponseMetadata>
        <RequestId>cb4cbf7cdd3f4572</RequestId>
    </ResponseMetadata>
</GetSessionTokenResponse>
```

4.6 Backoff 说明

当 OOS 系统服务繁忙，暂时不可使用时，OOS 会返回客户端 503 响应状态码和 Retry-After 响应头。示例如下：

```
HTTP/1.1 503 Service Unavailable
x-ctyun-request-id: abdcf4945d14406a-30383a4b4e4948542d6b6e696854
Retry-After: 90
Date: Tue, 6 May 2014 08:02:58 GMT
Content-Length: 133
Content-Type: text/xml
Connection: close
Server: CTYUN

<?xml version='1.0' encoding='UTF-8'?>
<Error>
    <Code>SlowDown</Code>
    <Message>Please reduce your request rate.</Message>
</Error>
```

4.7 错误码列表

状态码	错误码	错误信息	描述
400 Bad Request	DefalutTriggerAlreadyExists	The defalut trigger is already exists	trigger 已经存在。
400 Bad Request	IncompleteBody	You did not provide the number of bytes specified by the Content-Length HTTP header	请求头 Content-Length 不对。
400 Bad Request	InvaildBucketCorsConfigure	The maximum size of a CORS XML you provided is 64KiB.	XML 文档的长度不能能超过 64KiB。
400 Bad Request	InvaildBucketCorsConfigure	InvaildBucketCorsConfigure	CORS 配置无效。
400 Bad Request	InvaildBucketCorsConfigure	The XML you provided is not valid.	必须是标准的 XML 格式。
400 Bad Request	InvalidArgument		请求携带 host 请求头
400 Bad Request	InvalidArgument		对象名称不合格。
400 Bad Request	InvalidArgument	Bucket POST must contain a field named 'policy'. If it is specified, please check the order of the fields.	必须配置 policy。
400 Bad Request	InvalidArgument	Bucket POST must contain a field named 'signature'. If it is specified, please check the order of the fields.	必须配置 signature。
400 Bad	InvalidArgument	The IndexDocument Suffix is	IndexDocument 和 Suffix 不

Request		not well formed	能为空。
400 Bad Request	InvalidArgument		在调用 PUT Bucket Logging 接口时，如果 TargetBucket 为空，则 TargetPrefix 也必须为空。
400 Bad Request	InvalidArgument		在调用 PUT Bucket Logging 接口时，TargetPrefix 的长度不能超过了 900B。
400 Bad Request	InvalidArgument	The trigger name is invalid	Trigger 名字无效。
400 Bad Request	InvalidArgument	the bucket name is:xx	Bucket 名字无效。
400 Bad Request	InvalidArgument	The endpoint is not exists	Endpoint 不存在。
400 Bad Request	InvalidArgument	The replica mode is invalid	设置的 replca mode 无效。
400 Bad Request	InvalidArgument	The endpoint is duplicate	Endpoint 重复。
400 Bad Request	InvalidArgument	ID length should not exceed allowed limit of 255B	rule id 长度不能超过 255B。
400 Bad Request	InvalidArgument	At least one action needs to be specified in a rule	策略必须配置过期时间，Days 或 Date。
400 Bad Request	InvalidArgument	Days' for Expiration action must be a positive integer	Days 必须为正整数。
400 Bad Request	InvalidArgument	Date' must be at midnight GMT	Date 必须为 ISO8601 格式，并且为 UTC 的零点。
400 Bad Request	InvalidArgument	maxUploads should between 0~1000	最大上传的取值范围为 [0,1000]。

400 Bad Request	InvalidArgument	The length of delimiter must be 1.	delimiter 长度不能超过 1。并且只能是‘/’或空。
400 Bad Request	InvalidArgument	maxKeys should between 0~1000	maxKeys 取值范围为[0,1000]。
400 Bad Request	InvalidArgument	x-amz-limitrate invalid.The input should be positive integer.	x-amz-limitrate 取值必须为正整数。
400 Bad Request	InvalidArgument	The 'Content-Length' Field Format Invalid	Content-Length 必须是正整数。
400 Bad Request	InvalidArgument	The 'Expires' Field Format Invalid	Expires 格式不正确。
400 Bad Request	InvalidArgument	the object size should be less than 5T	待上传的对象不能超过 5TiB。
400 Bad Request	InvalidArgument	The metadata size should be less than 2K	元数据不能超过 2KiB。
400 Bad Request	InvalidArgument	Insufficient information. Origin request header needed.	跨域预检时，请求头不正确。
400 Bad Request	InvalidBucketName	the bucket name is:xx	Bucket 名字无效。
400 Bad Request	InvalidObjectName	the object name is:xxx	对象名字无效。
400 Bad Request	InvalidPartNumber		PartNumber 的取值是[1, 10000]。
400 Bad Request	InvalidPartSize		合并分片对象时，只允许最后一片小于 5MiB。
400 Bad Request	InvalidPolicyDocument	Invalid Policy: JSON INVALID:xxx	策略必须是 json 格式的。
400 Bad	InvalidPolicyDocument	Invalid according to Policy:	policy 内包含的域要与参数

Request		Extra input fields:xxx	里的一一对应。
400 Bad Request	InvalidPolicyDocument	The maximum size of a policy document you provided is 20KiB.	检查策略长度不能超过 20KiB。
400 Bad Request	InvalidPolicyDocument	Policy could not be parsed as a valid JSON string	策略必须是 json 格式。
400 Bad Request	InvalidPolicyDocument	Policy is missing required element - Statement	策略必须配置 Statement。
400 Bad Request	InvalidPolicyDocument	Policy is missing required element - Principal	策略必须配置 Principal。
400 Bad Request	InvalidPolicyDocument	Policy element has invalid value - Principal	Principal 值无效。
400 Bad Request	InvalidPolicyDocument	Policy is missing required element - Resource	策略必须配置 Resource。
400 Bad Request	InvalidPolicyDocument	Policy is missing required element - Resource content	策略的 Resource 元素必须赋值。
400 Bad Request	InvalidPolicyDocument	Policy has invalid resource - xx	Resource 无效。
400 Bad Request	InvalidPolicyDocument	Policy has an invalid condition key - xx	condition key 无效。
400 Bad Request	InvalidPolicyDocument	No such condition type - xx	condition type 无效。
400 Bad Request	InvalidPolicyDocument	Action does not apply to any resource(s) in statement	Action 无效。
400 Bad Request	InvalidPolicyDocument	Policy is missing required element - Effect	策略必须配置 Effect。
400 Bad Request	InvalidPolicyDocument	Policy element has invalid value - Effect : xxx	Effect 无效。

400 Bad Request	InvalidPolicyDocument	Policy element has invalid value - Version : xxx	Version 无效。
400 Bad Request	InvalidRedirectLocation	The website redirect location must be non-empty.	x-amz-website-redirect-location 头不能为空。
400 Bad Request	InvalidRedirectLocation	The website redirect location must have a prefix of 'http://' or 'https://' or '/'.	x-amz-website-redirect-location 必须有前缀 'http://' 、 'https://' 或'/'.。
400 Bad Request	InvalidRedirectLocation	The length of website redirect location cannot exceed 2,048 characters.	x-amz-website-redirect-location 长度不能超过 2048。
400 Bad Request	InvalidRequest	Missing required header for this request: Content-MD5	请求头必须包含 Content-MD5。
400 Bad Request	InvalidRequest	The maximum size of a prefix is 1024	Prefix 最大长度为 1024。
400 Bad Request	InvalidRequest	Rule ID must be unique. Found same ID for more than one rule	Rule ID 不能重复。
400 Bad Request	InvalidRequest	Found two rules with same prefix 'xx' and 'xxx' for same action type Expiration	同一 Bucket 的两条生命周期规则不能有相同的 prefix。
400 Bad Request	InvalidRequest	Found overlapping prefixes 'xx' and 'xx' for same action type Expiration	同一 Bucket 内的不同生命周期规则的前缀有重叠。
400 Bad Request	InvalidRequest	The number of concurrent connections exceeds the limit，please slow down.	当前时间的请求总数超过了允许的范围，请稍后再试。
400 Bad Request	InvalidRequest		如果源对象取 range 进行拷贝，那么源对象大小不能小

			于 5TiB。
400 Bad Request	InvalidRequest	This copy request is illegal because it is trying to copy an object to itself without changing the object's metadata or storage class.	复制对象时必须修改元数据或者存储类型。
400 Bad Request	InvalidStorageClass	Can not specify the storage class.	未合并的分段对象不能指定 x-amz-storage-class 进行拷贝。
400 Bad Request	InvalidTargetBucketForLogging	the target bucket name is: xx	目标 Bucket 不存在或者无目标 Bucket 的权限。
400 Bad Request	InvalidTriggerSourceBucketForLogging	the target bucket name is: xx	目标 Bucket 不存在或者无目标 Bucket 的权限。
400 Bad Request	InvalidURI		需要使用标准 http 请求的 url。
400 Bad Request	InvalidURI	the uri is:*xx*	urldecode 错误，需要使用 url 标准的 encode 方法对 uri 转码。
400 Bad Request	InvalidURI	The specified URI couldn't be parsed.	URI 解析错误。
400 Bad Request	MalformedXML	The XML you provided was not well-formed or did not validate against our published schema.	请求体 XML 不合规。
400 Bad Request	MalformedXML	The XML you provided was not well-formed or did not validate against our published schema.	请求体 XML 不合规。

400 Bad Request	MalformedXML	CORS Rule is empty.	CORS Rule 不能为空。
400 Bad Request	MalformedXML	The number of CORS Rules you provided cannot exceed 100.	CORS Rules 个数不能超过 100。
400 Bad Request	MalformedXML	The length of CORS Rule ID you provided cannot exceed 255 characters.	CORS Rule ID 长度不能超 255 个字符。
400 Bad Request	MalformedXML	AllowedOrigin does not exist.	AllowedOrigin 不存在。
400 Bad Request	MalformedXML	AllowedMethod does not exist.	AllowedMethod 不存在。
400 Bad Request	MalformedXML	AllowedOrigin xx can not have more than one wildcard.	AllowedOrigin 最多只能携带一个通配符。
400 Bad Request	MalformedXML	AllowedHeader xx can not have more than one wildcard.	AllowedHeader 最多只能携带一个通配符。
400 Bad Request	MalformedXML	ExposeHeader xx contains wildcard. We currently do not support wildcard for ExposeHeader.	ExposeHeader 不能携带通配符。
400 Bad Request	MalformedXML	Please enter an integer not less than -1.	MaxAgeSeconds 不能小于 0。
400 Bad Request	MalformedXML	CORS Rule ID is not unique.	CORSRule ID 必须是唯一的。
400 Bad Request	MalformedXML	The XML you provided was not well-formed or did not validate against our published schema.	400 Bad Request

400 Bad Request	TooManyBuckets		Bucket 数量已经到达上限。
400 Bad Request	TriggerAlreadyExists	The trigger is already exists	trigger 已经存在。
403 Forbidden	AccessDenied		bucket 创建操作不支持匿名访问
403 Forbidden	AccessDenied		用户没有写数据的权限，
403 Forbidden	AccessDenied		未配置 bucket website，不允许使用 cname 方式请求 oos。
403 Forbidden	AccessDenied	The OOS service has been closed due to overdue payment. If the overdue payment exceed 15 days, the data will be deleted. Please recharge it in time.	账户欠费，须及时补齐费用。
403 Forbidden	AccessDenied	the user's balance is not enough	账户欠费，须及时补齐费用。
403 Forbidden	AccessDenied	the user's permission is not enough	权限不足。
403 Forbidden	AccessDenied	please use primary access key	需要使用主 accesskey。
403 Forbidden	AccessDenied	Access Denied	非主 access key 只能 list 以自己 access key 为前缀的对象。
403 Forbidden	AccessDenied	Access Denied	非主 access key 不能执行 bucket 级别的操作。

403 Forbidden	AccessDenied	Access Denied	非主 key 只能对自己 key 为前缀的对象进行操作
403 Forbidden	AccessDenied	Invalid according to Policy: Policy Condition failed:xxx	签名策略不对。
403 Forbidden	AccessDenied	Access Denied	预签名时，Signature Expires 和 AWSAccessKeyId 都不能为空。
403 Forbidden	AccessDenied	the secret token is expired. expiration:	secret token 过期。
403 Forbidden	Forbidden		预检请求的 Bucket 不存在。
403 Forbidden	Forbidden		Access-Control-Request-Method 不正确。
403 Forbidden	Forbidden		Access-Control-Request-Method 不能为空。
403 Forbidden	Forbidden		预检请求与 CORSRule 不匹配。
403 Forbidden	Forbidden		CORSRule 不存在。
403 Forbidden	IllegalObject	The object "%s/%s" is illegal, forbid copy.	源对象是非法对象，不允许拷贝。
403 Forbidden	InvalidAccessKeyId	InvalidAccessKeyId	Accesskeyid 无效。
403 Forbidden	InvalidAccessKeyId		Accesskeyid 无效。
403 Forbidden	permissionIsNotAllow		权限不足。
403	RequestTimeTooSkewed	The difference between the	客户端时间和服务器端时间

Forbidden		request time and the server's time is too large.	不一致。
403 Forbidden	SignatureDoesNotMatch		签名计算错误，需按 OOS 的签名规则检查客户端签名计算方法
403 Forbidden	SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your key and signing method.	签名不正确。
403 Forbidden	UnaccessibleRegion		用户无当前资源池的权限。
404 Not Found	NoSuchBucket		没有此 Bucket。
404 Not Found	NoSuchBucket	the request bucketname is xx	此 Bucket 不存在。
404 Not Found	NotSuchBucketPolicy		没有此策略。
404 Not Found	NoSuchLifecycleConfiguration	The lifecycle configuration does not exist.	生命周期规则的配置不存在。
404 Not Found	NoSuchUpload		请求的分段对象不存在。
404 Not Found	NoSuchUpload	The resource you requested does not exist	请求的资源不存在。
405 Method Not Allowed	MethodNotAllowed		OOS 不支持该请求。
405 Method Not Allowed	MethodNotAllowed	The specified method is not allowed against this resource.	不支持此操作。
409 Conflict	BucketAlreadyExists	BucketAlreadyExists	Bucket 名字已存在。

409 Conflict	BucketOwnerAlreadyExists	bucket|owner already exists in pool	Bucket 名称已存在。
412 Precondition Failed	PreconditionFailed	The pre-conditions x-amz-copy-source-if-modified-since you specified did not hold	指定的前置条件 x-amz-copy-source-if-modified-since 不成立。
500 Internal Server Error	GetObjectFailed		检索对象失败。
500 Internal Server Error	Upload Part Failed		分片上传失败。
500 Internal Server Error	Copy Object Fail		复制对象失败。
500 Internal Server Error	Copy Upload Part Failed		复制上传片段失败。
500 Internal Server Error	Abort Multipart Upload Failed		终止分片上传失败。
500 Internal Server Error	Delete Object Failed		删除对象失败。
503 Service Unavailable	SlowDown	Please reduce your request rate.	请降低请求频率。

5 附录

5.1 使用 HttpURLConnection 开发

代码示例

```
package cn.ctyun.oos.sample;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.util.TimeZone;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;

public class HttpUrlConnectionSample {
private static final int CONN_TIMEOUT = 10000;
private static final int READ_TIMEOUT = 30000;
private static final String DATE_STR = "EEE, d MMM yyyy HH:mm:ss 'GMT'";
private static final SimpleDateFormat DATE_FMT = new SimpleDateFormat(DATE_STR,
           Locale.ENGLISH);
static {
        TimeZone gmt = TimeZone.getTimeZone("GMT");
        DATE_FMT.setTimeZone(gmt);
    }

Private final String host;
Private final int port;
Private final String ak;
Private final String sk;
```

```
public HttpUrlConnectionSample(String host, intport, String ak, String sk)
{
    this.host = host;
    this.port = port;
    this.ak = ak;
    this.sk = sk;
    }

  private String authorize(String httpVerb, String date, String bucket, String
          objectName) throws Exception {
    String stringToSign = httpVerb + "\n\n\n" + date + "\n/" + bucket + "/" +
objectName;
     Mac mac = Mac.getInstance("HmacSHA1");
    mac.init(new SecretKeySpec(sk.getBytes("UTF-8"), "HmacSHA1"));
      byte[] macResult = mac.doFinal(stringToSign.getBytes("UTF-8"));
      String signature = newString(Base64.encodeBase64(macResult), "UTF-8");
      String authorization = "AWS " + ak + ":" + signature;
returnauthorization;
    }

Public void put(String bucket, String objName, byte[] data) throws Exception
{
        String date = DATE_FMT.format(new Date());
        String authorization = authorize("PUT", date, bucket, objName);
        URL url = new URL("http", host, port, "/" + bucket + "/" + objName);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setUseCaches(false);
conn.setFixedLengthStreamingMode(data.length);
conn.setRequestProperty("Date", date);
conn.setRequestProperty("Authorization", authorization);
conn.setConnectTimeout(CONN_TIMEOUT);
conn.setReadTimeout(READ_TIMEOUT);
conn.setDoOutput(true);
conn.setRequestMethod("PUT");
conn.connect();
try (OutputStream out = conn.getOutputStream()) {
out.write(data);
```

```
out.flush();
        }
intcode = conn.getResponseCode();
System.out.println("code=" + code);
    }

Public void get(String bucket, String objName) throws Exception {
        String date = DATE_FMT.format(new Date());
        String authorization = authorize("GET", date, bucket, objName);
        URL url = new URL("http", host, port, "/" + bucket + "/" + objName);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setUseCaches(false);
conn.setRequestProperty("Date", date);
conn.setRequestProperty("Authorization", authorization);
conn.setConnectTimeout(CONN_TIMEOUT);
conn.setReadTimeout(READ_TIMEOUT);
conn.setDoInput(true);
conn.setRequestMethod("GET");
conn.connect();
int code = conn.getResponseCode();
System.out.println("code=" + code);
try (InputStream in = conn.getInputStream()) {
byte[] buffer = new byte[1024 * 8];
while (in.read(buffer) != -1) {
            }
        }
    }

Public void delete(String bucket, String objName) throws Exception {
        String date = DATE_FMT.format(new Date());
        String authorization = authorize("DELETE", date, bucket, objName);
        URL url = new URL("http", host, port, "/" + bucket + "/" + objName);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestProperty("Date", date);
conn.setRequestProperty("Authorization", authorization);
conn.setConnectTimeout(CONN_TIMEOUT);
conn.setReadTimeout(READ_TIMEOUT);
```

```
conn.setRequestMethod("DELETE");
conn.connect();
int code = conn.getResponseCode();
System.out.println("code=" + code);
    }

Public static void main(String[] args) throws Exception {
        String host = "oos-nm2.ctyunapi.cn";
        Int port = 80;
        String ak = "your_ak";
        String sk = "your_sk";
        String bucket = "bucket_name";
        String objName = "object_name";
byte[] data = new byte[1024 * 1024 * 10];
        Random rand = new Random();
rand.nextBytes(data);
        HttpUrlConnectionSample s = new HttpUrlConnectionSample(host, port, ak, sk);
s.put(bucket, objName, data);
s.get(bucket, objName);
s.delete(bucket, objName);
    }
}
```

5.2 Endpoint 列表

以下是各个地区的 OOS API Endpoint 和 IAM Endpoint 列表，在访问各个地区的数据时，请使用对应的 Endpoint。

地区	OOS API Endpoint	OOS IAM Endpoint
北京 2	oos-bj2.ctyunapi.cn	oos-bj2-iam.ctyunapi.cn
内蒙	oos-nm2.ctyunapi.cn	oos-nm2-iam.ctyunapi.cn
杭州	oos-hz.ctyunapi.cn	oos-hz-iam.ctyunapi.cn
江苏	oos-js.ctyunapi.cn	oos-js-iam.ctyunapi.cn
广州	oos-gz.ctyunapi.cn	oos-gz-iam.ctyunapi.cn
北京	oos-hq-bj.ctyunapi.cn	oos-hq-bj-iam.ctyunapi.cn
上海	oos-hq-sh.ctyunapi.cn	oos-hq-sh-iam.ctyunapi.cn