对象存储

（Object-Oriented Storage,OOS）

开发者文档 V6

中国电信股份有限公司

云计算分公司

目录

1 对象存储（OOS）

1.1 产品介绍

对象存储（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

1) 容量几乎没有上限，弹性扩容；

2) 自动数据冗余和故障切换，确保高可用；

3) 流量按需计费，节省带宽成本；

4) 易于管理、维护和升级。

2 主要概念

面向对象存储的主要概念有：Account（账户）、Service（服务）、Bucket（对象容器）和 Object（对象）。它们之间的关系如图 1 所示。在使用 OOS 之前，首先需要在 www.ctyun.cn 注册一个账号（Account），注册成功之后，OOS 会为该账号提供服务（Service），在该服务下，用户可以创建 1 个或多个对象容器（Bucket），每个对象容器中可以存储不限数量的对象（Object）。

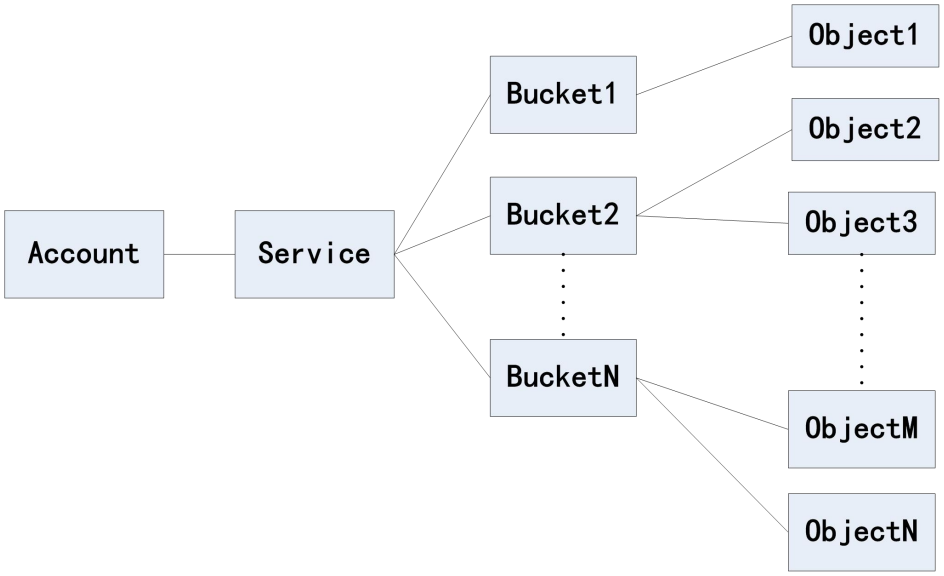


图 1　OOS 主要概念之间的关系

2.1 Account

在使用 OOS 之前，首先需要在 www.ctyun.cn 注册一个账号（Account）。注册账号时必须填写用户手机号码、邮箱、密码和联系方式，其他信息如邀请码等可以选填。正确填写注册信息后需要进行实名认证，只有实名认证通过的账号才可以继续开通并使用 OOS 服务。

2.2 Service

Service 是 OOS 为注册成功的用户提供的服务，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对象容器（Bucket）。

2.3 Bucket

Bucket 是存储 Object 的容器。面向对象存储的每个 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。

2.3.1 Bucket 的命名规范

1、 Bucket 名称必须全局唯一

2、 Bucket 名称长度介于 3 到 63 字节之间

3、 Bucket 名称可以由一个或者多个小节组成，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；

4、 Bucket名称不能是IP地址形式（如192.162.0.1）；

5、不允许使用敏感字符，例如“www”、“http”、“https”等作为开头；

6、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

身份验证通过的用户可以新建 Bucket、删除 Bucket 以及查看 Bucket 的属性。

新建 Bucket 时需要输入 Bucket 的名称，选择操作权限（Bucket 的默认操作权限是 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：只有不包含任何文件的 Bucket 才能删除。若 Bucket 中包含文件，则该 Bucket 不能删除，要想删除，需要先将 Bucket 中的文件全部删除。只有所有者可以删除 Bucket。

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

2.3.3 Bucket 的索引位置和数据位置

在创建 bucket 时，需要指定索引位置，索引位置创建之后，不能被更改。

Bucket 的数据位置是指对象存储的位置。用户可以选择就近写入，即 OOS 将数据写到离用户访问点最近的资源池中。用户也可以指定要写入的资源池，OOS 会将数据按顺序写入用户指定的资源池。用户也可以选择让 OOS 自动调度数据，OOS 可以根据用户选择地区的实际使用情况，自动进行数据存储位置的调度，以便为用户提供更快的访问速度。

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。对象存储（OOS）支持的单个文件的最大长度为 5T 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。您可以在对象存储的控制台—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，每个用户可以拥有多个主密钥和普通密钥，两者的区别是：

1. 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。
2. 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。
3. 用户可以将普通密钥设置成为主密钥。
4. 普通密钥不能创建，删除，修改 bucket 属性。

5. 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。

6. 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 通过 IPv6 访问 OOS

OOS 除了支持通过 IPv4 协议访问以外 ,还支持 IPv6 协议。当用户使用 IPv6 访问 bucket 时，有以下几点需要知晓。

- 客户端和网络必须都支持IPv6。
- 在bucket policy中设置ip地址的黑白名单时，可以使用IPv6。用户可以将bucket policy的Condition 元素同时设置为包含IPv4和IPv6地址。例如

```
"Condition": {
    "IpAddress": {
        "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
        ]
    }
}
```

- 当客户使用IPv6访问时，bucket日志中记录的IP地址（Remote IP字段），是IPv6格式的。

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、Policy 安全策略。

3.1 用户签名验证

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。如果用户的请求中没有 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 +
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

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，则根据标准 RFC 2616, 4.2 章进行合并（两个值之间只用逗号分隔）。如有

两个名为’x-amz-meta-name’的请求头，对应的值分别为’ fred’和’ barney’，则合并后为：’x-amz-meta-name:fred,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。

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
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.
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.
例子：/BucketName/ObjectName?response-content-type=ContentType
当发送批量删除对象请求时，delete 参数需要包含在 CanonicalizedResource 中。

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。

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。

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。

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，如果你在签名时设置 Date 请求头有困难，那么你可以使用 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（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。

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.ctyunapi.cn Date: Tue, 27 Mar 2007 19:36:42 +0000 *Authorization: AWS AKIAIOSFODNN7EXAMPLE:* *xXjDGYUmKxnwqr5KXNPGldn5LbA=*	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	PUT\n \n

Content-Length: 94328 Host: johnsmith.oos.ctyunapi.cn Date: Tue, 27 Mar 2007 21:15:45 +0000 *Authorization: AWS AKIAIOSFODNN7EX AMPLE:* *hcicpDDvL9SsO6AkvxqmIWkmOuQ=*	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 HTTP/1.1 User-Agent: Mozilla/5.0 Host: johnsmith.oos.ctyunapi.cn Date: Tue, 27 Mar 2007 19:42:41 +0000 *Authorization: AWS AKIAIOSFODNN7EXAMPLE:js Rt/rhG+Vtp88HrYL706QhE4w4=*	GET\n \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.ctyunapi.cn Date: Tue, 27 Mar 2007 19:44:46 +0000 *Authorization: AWS AKIAIOSFODNN7EX AMPLE:thdUi9VAkzhkniLj96JIrOPGi0g=*	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 HTTP/1.1 User-Agent: dotnet Host: oos.ctyunapi.cn Date: Tue, 27 Mar 2007 21:20:27 +0000 x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000 *Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5Ip83xlYzk=*	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; filename=database.dat Content-Encoding: gzip Content-Length: 5913339 *Authorization: AWS AKIAIOSFODNN7EXAMPLE:C0FlOtU8Ylb9KDTpZqYkZPX91iI=*	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。

7. List Buckets

请求	**StringToSign**
GET / HTTP/1.1 Host: oos.ctyunapi.cn Date: Wed, 28 Mar 2007 01:29:59 +0000 *Authorization: AWS AKIAIOSFODNN7EX* *AMPLE:Db+gepJSUbZKwpx1FR0DLtEYoZA=*	GET\n \n \n Wed, 28 Mar 2007 01:29:59 +0000\n /

8. 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.ctyunapi.cn Date: Wed, 28 Mar 2007 01:49:49 +0000 *Authorization: AWS* *AKIAIOSFODNN7EXAMPLE:* *dxhSBHoI6eVSPcXJqEghlUzZMnY=*	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对象存储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.ctyunapi.cn/quotes/nelson?AWSAccessKeyId=44CF9590006BF252F707&Expires=114

1889120&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	可以登录到对象存储的门户上，点击控制台—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 这个参数的值是一个 UNIX 时间（自 UTC 时间 1970 年 1 月 1 号开始的秒数，详见 wiki），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：

1) 通过 URL 包含签名时，之前的 Date 参数换成 Expires 参数。

2) 不支持同时在 URL 和 Head 中包含签名。

3) 如果传入的 Signature，Expires，AWSAccessKeyId 出现不止一次，以第一次为准。

4) 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 用户签名验证（V4）

OOS 除了支持 3.1 用户签名验证中所述的签名算法外，还兼容 Amazon S3 Version 4 签名算法。

3.2.1 Signature Version 4 的工作原理

客户端按照如下方式创建签名：

1．创建规范请求。

2．使用规范请求和其他信息创建待签名的字符串。

3．使用 SecretKey 派生签名密钥。然后将该签名密钥与在上一步中创建的字符串结合使用来创建签名。

4．将生成的签名添加到 HTTP 请求头中或者作为参数添加到 URL 中。

OOS 服务收到请求后，将执行相同步骤来计算请求中发送的签名。之后，OOS 会将计算得到的签名与用户在请求中发送的签名进行比较。如果签名匹配，则处理请求。如果签名不匹配，则拒绝请求。

3.2.2 认证方法

用户可以使用以下方法来发送身份验证信息：

1．HTTP 授权请求头 - 使用 HTTP Authorization 请求头是验证 OOS 请求的最常用方法。所有 OOS REST 操作（使用 POST 请求的基于浏览器的上传除外）都需要此请求头。

2．查询字符串参数 - 使用 URL 查询参数来提供请求信息，包括身份验证信息。由于请求签名是 URL 的一部分，因此这种类型的 URL 通常称为预签名 URL。

OOS 还支持使用基于浏览器的 HTTP POST 请求的上传方式。通过 HTTP POST 请求，用户可以直接通过浏览器将内容上传到 OOS。

3.2.3 使用 Authorization 请求头验证

3.2.3.1 概述

Authorization 请求头包含以下信息（增加换行是为了方便阅读，实际为空字符串）：

```
Authorization: AWS4-HMAC-SHA256

Credential=AKIAIOSFODNN7EXAMPLE/20180501/region/s3/aws4_request,

SignedHeaders=host;range;x-amz-date,

Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

组成字段说明如下：

部分	**描述**
AWS4-HMAC-SHA256	用于签名的算法，固定值
Credential	用户的 accessKeyId 和范围信息，范围信息包括请求日期、区域、服务、终止字符串 aws4_request，格式如下： *<your-access-key-id>/ <date>/ <region>/ <service>*/aws4_request 其中,date 格式为 YYYYMMDD 若使用 OOS API 服务，service 为 s3 若使用 IAM 服务，service 为 sts
SignedHeaders	已签名请求头的列表。该列表只需包含请求头名字，用分号分隔，必须全部小写，并按字符顺序对其进行排序，示例如下： host;range;x-amz-date
Signature	计算出的 256 位签名信息，以 64 个小写十六进制字符串形式表示。

有两种签名计算方式：

- 签名负载方式 - 用户可以选择计算整个负载（即请求体）的 checksum，并将其包含在签名计算中。这种方式提高了安全性 ,但用户需要读取两次负载或将其缓冲在内存中。我们建议用户包含负载 checksum 以增强安全性。
- 无签名负载方式 - 在签名计算中不包括负载的 checksum。

上述两种方式，都必须携带 x-amz-content-sha256 请求头，如果选择签名负载方式，请将 x-amz-content-sha256 请求头的值设置为负载的 checksum 值，否则将值设置为文本字符串 UNSIGNED-PAYLOAD。

3.2.3.2 签名过程

签名过程如下图所示：

下表描述了图中显示的功能。用户需要为这些函数实现代码。

功能	描述
Lowercase()	将字符串转换为小写。
Hex()	小写 16 进制编码。
SHA256Hash()	安全散列算法（SHA）加密散列函数。
HMAC-SHA256()	使用签名密钥，根据 SHA256 算法计算出的签名值。
Trim()	删除任何前导或尾随空格。
UriEncode()	URI 编码每个字节。UriEncode（ ）必须强制执行以下规则：

	• URI 编码除了下面字符之外的每个字节：'A' - 'Z'，'a' - 'z'，'0' - '9'，' - '，'.'，'_'和'~'。
	• 空格字符是保留字符，必须编码为“%20”（而不是“+”）。
	• 每个 URI 编码字节由'%'和两位十六进制值组成。
	• 十六进制值中的字母必须为大写，例如“%1A”。
	• 除了对象名之外，对正斜杠字符'/'进行编码。例如，如果对象名称为 photos/Jan/sample.jpg，则不对名称中的正斜杠进行编码。
	重要
	我们建议用户编写自己的自定义 UriEncode 函数，以确保您的编码可以正常工作。
	以下是 Java 中的示例 UriEncode（）函数。
	`public static String UriEncode(CharSequence input, boolean encodeSlash) {`
	`StringBuilder result = new StringBuilder();`
	`for (int i = 0; i < input.length(); i++) {`
	`char ch = input.charAt(i);`
	`if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9') || ch == '_' || ch == '-' || ch == '~' || ch == '.') {`
	`result.append(ch);`
	`} else if (ch == '/') {`
	`result.append(encodeSlash ? "%2F" : ch);`
	`} else {`
	`result.append(toHexUTF8(ch));`
	`}`
	`}`
	`return result.toString();`
	`}`

1．创建规范请求

将请求的内容（包括主机、操作、请求头等）组织为标准规范格式。规范请求是用于创建待签字符串的输入之一。伪代码如下：

```
CanonicalRequest =
  HTTPRequestMethod + '\n' +
  CanonicalURI + '\n' +
  CanonicalQueryString + '\n' +
  CanonicalHeaders + '\n' +
  SignedHeaders + '\n' +
  HexEncode(Hash(RequestPayload))
```

HTTPMethod 是 HTTP 方法，例如 GET，PUT，HEAD 和 DELETE。

CanonicalURI 是 URI 的绝对路径，以域名后面的“/”开头，直到字符串的末尾或者问号字符（'?'）截止。例如 /examplebucket/myphoto.jpg

CanonicalQueryString 是 URI 编码后的查询字符串参数。用户需要单独对参数名称和值进行 URI 编码。并需要按参数名称的字母顺序，对参数进行排序。排序在编码后进行。以下 URI 示例中的查询字符串是 `prefix=somePrefix&marker=someMarker&max-keys=20`：

```
http://oos.ctyunapi.cn/examplebucket?prefix=somePrefix&marker=someMarker&max-keys=20
```

CanonicalQueryString 的构造方式如下（为了便于阅读，添加了换行符）：

```
UriEncode("marker")+"="+UriEncode("someMarker")+"&"+
UriEncode("max-keys")+"="+UriEncode("20") + "&" +
UriEncode("prefix")+"="+UriEncode("somePrefix")
```

当请求的目标是子资源时，相应的查询参数的值设置为空字符串（ “” ）。例如，下面的请求用于设置 bucket 的 ACL 权限：

```
http://oos.ctyunapi.cn/examplebucket?acl
```

在这种情况下，CanonicalQueryString 为：

```
UriEncode("acl") + "=" + ""
```

如果 URI 中不包含“？”,则请求中不存在查询字符串,此时将 CanonicalQueryString 设置为空字符串（ “” ），但仍需要包含“\ n”。

CanonicalHeaders 是请求头的列表。各个请求头名称和值由换行符（ “\ n” ）分隔。请求头名称必须为小写。需要按字母顺序对请求头名称进行排序，示例如下：

```
Lowercase(<HeaderName1>)+":"+Trim(<value>)+"\n"

Lowercase(<HeaderName2>)+":"+Trim(<value>)+"\n"

...

Lowercase(<HeaderNameN>)+":"+Trim(<value>)+"\n"
```

CanonicalHeaders 列表必须包括以下内容：

- HTTP Host 标头
- 如果存在 Content-Type 请求头，则必须将其添加到 CanonicalHeaders 列表中。
- 所有 x-amz-*请求头。

以下是 CanonicalHeaders 的示例，请求头名称为小写并已排序。

```
host:s3.amazonaws.com

x-amz-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e464

9b934ca495991b7852b855

x-amz-date:20130708T220855Z
```

SignedHeaders 是按字母顺序排序的，以分号分隔的小写请求头名称列表。列表中的请求头与用户在 CanonicalHeaders 字符串中包含的请求头相同。例如，对于前面的示例，SignedHeaders 的值为：

```
host;x-amz-content-sha256;x-amz-date
```

HashedPayload 是请求负载的 SHA256 哈希的十六进制值。

如果请求中没有负载，则计算空字符串的哈希值，如下所示：

```
Hex(SHA256Hash(""))
```

哈希返回以下值：

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

例如，当用户使用 PUT 请求上传对象时，用户可以在请求体中提供对象数据。使用 GET 请求检索对象时，没有请求体，所以计算空字符串的哈希。

2．创建待签名字符串

待签名的字符串格式如下：

```
"AWS4-HMAC-SHA256" + "\n" +
timeStampISO8601Format + "\n" +<Scope> + "\n" +
Hex(SHA256Hash(<CanonicalRequest>))
```

常量字符串 AWS4-HMAC-SHA256 指定用户使用的哈希算法 HMAC-SHA256。

timeStamp 是 ISO 8601 格式的当前 UTC 时间（例如 20180501T000000Z）。

Scope 是将生成的签名绑定到指定日期，OOS 区域和服务。

```
date.Format(<YYYYMMDD>) + "/" + <region> + "/" + <service> +
"/aws4_request"
```

对于 OOS，服务字符串是 s3。Region 为域名 oos-xx.ctyunapi.cn 中的 xx 部分。

3．计算签名

使用 secretKey 作为初始哈希操作的密钥，对请求日期、区域和服务执行一系列加密哈希操作（HMAC 操作），从而派生签名密钥。伪代码如下：

```
DateKey = HMAC-SHA256("AWS4"+"<SecretAccessKey>", "<YYYYMMDD>")

DateRegionKey = HMAC-SHA256(<DateKey>, "<oos-region>")

DateRegionServiceKey = HMAC-SHA256(<DateRegionKey>, "<oos-service>")

SigningKey = HMAC-SHA256(<DateRegionServiceKey>, "aws4_request")
```

最终签名是使用签名密钥作为密钥，对待签名字符串计算得到 HMAC-SHA256 哈希值。伪代码如下：

```
HMAC-SHA256(SigningKey, StringToSign)
```

4．将签名信息添加到请求头

在计算签名后，将其添加到请求的 HTTP 请求头或查询字符串中。

将签名信息添加到 Authorization 标头的伪代码如下：

```
Authorization: algorithm Credential=ak/credential_scope,

SignedHeaders=SignedHeaders, Signature=signature
```

3.2.3.3 示例

以下是使用 V4 签名的示例。示例中使用的访问密钥如下：

参数	值
AWSAccessKeyId	2a948fd3f00ba0925806
AWSSecretAccessKey	ef2017c2e5ffa0b1761717ecbca021da16501384

Bucket 名称：examplebucket。

访问的域名是 oos-cn.ctyunapi.cn, region 是 cn

1. 示例：GET 对象

从存储桶 examplebucket 中获取对象 test.txt 的前 10 个字节。请求如下：

```
GET /test.txt HTTP/1.1
x-amz-content-sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Authorization: SignatureToBeCalculated
x-amz-date: 20190220T060724Z
Range: bytes=0-9
Host: examplebucket.oos-cn.ctyunapi.cn
```

由于此 GET 请求不提供任何请求体内容，因此该 x-amz-content-sha256 请求头的值是空请求体的哈希值。以下步骤显示 Authorization 请求头的计算的方法。

1) StringToSign

a. 创建规范请求

```
GET

/test.txt



host:examplebucket.oos-cn.ctyunapi.cn

range:bytes=0-9

x-amz-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934c

a495991b7852b855

x-amz-date:20190220T060724Z



host;range;x-amz-content-sha256;x-amz-date

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

其中，最后一行是空请求体的 hash 值。第三行是空，因为此请求不包含请求参数。

b. 待签名字符串

```
AWS4-HMAC-SHA256

20190220T060724Z

20190220/cn/s3/aws4_request

bca722269a76aadb00dfe5a50fefdbd5712065267e1692cc596cefd2681f5d14
```

2) 生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("AWS4" +

"<YourSecretAccessKey>","20190220"),"cn"),"s3"),"aws4_request")
```

3) 计算后的签名

```
be3f55b78165716c51ce37f588048f858fc27f7449d8fe74f887d999e5fc9193
```

4) Authorization 请求头

```
Authorization: AWS4-HMAC-SHA256

Credential=2a948fd3f00ba0925806/20190220/cn/s3/aws4_request,

SignedHeaders=host;range;x-amz-content-sha256;x-amz-date,

Signature=be3f55b78165716c51ce37f588048f858fc27f7449d8fe74f887d999e5fc9

193
```

2. 示例：PUT 对象

在存储桶 examplebucket 中上传对象 test.txt。

```
PUT /examplebucket/test.txt HTTP/1.1

x-amz-content-sha256:

7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9

Authorization:SignatureToBeCalculated

x-amz-date: 20190220T070722Z

x-amz-storage-class: STANDARD

Host: oos-cn.ctyunapi.cn

Content-Length: 12



hello world!
```

以下步骤显示 Authorization 请求头的计算的方法。

1) StringToSign

a. 创建规范请求

```
PUT

/examplebucket/test.txt


content-length:12

host:oos-cn.ctyunapi.cn

x-amz-content-sha256:7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5

e3df163be08e6ca9

x-amz-date:20190220T070722Z

x-amz-storage-class:STANDARD


content-length;host;x-amz-content-sha256;x-amz-date;x-amz-storage-cla

ss

7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9
```

其中，第三行是空，因为此请求不包含请求参数。最后一行是请求体的 hash 值，它应该与 x-amz-content-sha256 请求头的值相同。

b. 待签名字符串

```
AWS4-HMAC-SHA256

20190220T070722Z

20190220/cn/s3/aws4_request

66919f4f7f555dec8599c5894bbd5c104767bbf0180103d751653143f67a8d45
```

2) 生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("AWS4" +

"<YourSecretAccessKey>","20190220"),"cn"),"s3"),"aws4_request")
```

3) 计算后的签名

```
29407b3d2010ab3f86e313302a4d952d8ac0070364cd91ba3b113258a4d36b9b
```

4) Authorization 请求头

```
Authorization: AWS4-HMAC-SHA256
Credential=2a948fd3f00ba0925806/20190220/cn/s3/aws4_request,
SignedHeaders=content-length;host;x-amz-content-sha256;x-amz-date;x-amz
-storage-class,
Signature=29407b3d2010ab3f86e313302a4d952d8ac0070364cd91ba3b113258a4d36
b9b
```

3. 示例：列出存储桶中的对象

列出存储桶 examplebucket 中的对象，prefix 设置为“t”，最多返回 2 个对象。请求如下：

```
GET /?max-keys=2&prefix=t HTTP/1.1
x-amz-content-sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Authorization:  SignatureToBeCalculated
x-amz-date: 20190220T085955Z
Host: examplebucket.oos-cn.ctyunapi.cn
```

以下步骤显示 Authorization 请求头的计算的方法。

1) StringToSign

a. 创建规范请求

```
GET

/

max-keys=2&prefix=t

host:examplebucket.oos-cn.ctyunapi.cn

x-amz-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934c

a495991b7852b855

x-amz-date:20190220T085955Z



host;x-amz-content-sha256;x-amz-date

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

其中，最后一行是空请求体的 hash 值。

b. 待签名字符串

```
AWS4-HMAC-SHA256

20190220T085955Z

20190220/cn/s3/aws4_request

bc2b6af0cbbe17679b2697f7239b02dc21d4b62fc30e197441cf900d35d3b103
```

2) 生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("AWS4" +

"<YourSecretAccessKey>","20190220"),"cn"),"s3"),"aws4_request")
```

3) 计算后的签名

```
ce5ef3764d4a34b4e3c81d37b9a310432e5c4bf8bb4722c14877adba882fc559
```

4) Authorization 请求头

```
Authorization: AWS4-HMAC-SHA256
Credential=2a948fd3f00ba0925806/20190220/cn/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date,
Signature=ce5ef3764d4a34b4e3c81d37b9a310432e5c4bf8bb4722c14877adba882fc
559
```

3.2.4 使用查询参数验证

3.2.4.1 概述

用户可以使用查询字符串参数验证身份信息，此方法也称为预签名 URL。预签名 URL 的用例场景是用户可以授予对 OOS 资源的临时访问权限。例如，用户可以在网站上包含预先签名的 URL，或者在命令行客户端（例如 Curl）中使用它来下载对象。

以下是预签名 URL 的示例。

```
https://oos.ctyunapi.cn/examplebucket/test.txt
?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=<your-access-key-id>/20180721/<oos-region>/s3/aws4_request
&X-Amz-Date=20180721T201207Z
&X-Amz-Expires=86400
&X-Amz-SignedHeaders=host
&X-Amz-Signature=<signature-value>
```

在示例 URL 中，请注意以下事项：

- 添加换行符是为了便于阅读。
- 该 X-Amz-CredentialURL 中的值仅显示可读性“/”字符。在实践中，它应编码为%2F。例如：

```
&X-Amz-Credential=<your-access-key-id>%2F20180721%2F<oos-region>%2Fs3%2Faws
4_request
```

下表介绍了 URL 中提供身份验证信息的查询参数。

查询字符串参数	**说明**
X-Amz-Algorithm	请将此参数值设置为 AWS4-HMAC-SHA256。
X-Amz-Credential	用户的 accessKeyId 和范围信息，范围信息包括请求日期、区域、服务、终止字符串 aws4_request，格式如下： <your-access-key-id>/<date>/<region>/<service>/aws4_request 其中,date 格式为 YYYYMMDD，service 为 s3
X-Amz-Date	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”。
X-Amz-Expires	提供生成的预签名 URL 有效的时间段（以秒为单位）。例如，86400（24 小时）。该值是整数。您可以设置的最小值为 1，最大值为 604800（七天）。
X-Amz-SignedHeaders	列出用于计算签名的标头。签名计算中需要以下标头： ● HTTP host 标头。 ● x-amz-*请求头。
X-Amz-Signature	提供签名以验证您的请求。此签名必须与 OOS 计算的签名相匹配; 否则，OOS 拒绝该请求。

3.2.4.2 签名过程

下图说明了签名计算过程。

下表描述了图中显示的功能。用户需要为这些功能实现代码。

功能	描述
Lowercase()	将字符串转换为小写。
Hex()	小写十六进制编码。
SHA256Hash()	安全散列算法（SHA）加密散列函数。
HMAC-SHA256()	使用提供的签名密钥的 SHA256 算法计算 HMAC。这是最后的签名。
Trim()	删除任何前导或尾随空格。
UriEncode()	URI 编码每个字节。UriEncode（）必须强制执行以下规则： ● URI 编码除了下面字符之外的每个字节：'A' - 'Z'，'a' - 'z'，'0' - '9'，' - '，'.'，'_'和'~'。 ● 空格字符是保留字符，必须编码为“%20”（而不是“+”）。 ● 每个 URI 编码字节由'%'和两位十六进制值组成。 ● 十六进制值中的字母必须为大写，例如“%1A”。

	• 除了对象名之外，对正斜杠字符'/'进行编码。例如，如果对象名称为 photos/Jan/sample.jpg，则不对键名称中的正斜杠进行编码。
	我们建议您编写自己的自定义 UriEncode 函数，以确保您的编码可以正常工作。
	以下是 Java 中的示例 UriEncode（）函数。
	`public static String UriEncode(CharSequence input, boolean encodeSlash) {`
	`StringBuilder result = new StringBuilder();`
	`for (int i = 0; i < input.length(); i++) {`
	`char ch = input.charAt(i);`
	`if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9') || ch == '_' || ch == '-' || ch == '~' || ch == '.') {`
	`result.append(ch);`
	`} else if (ch == '/') {`
	`result.append(encodeSlash ? "%2F" : ch);`
	`} else {`
	`result.append(toHexUTF8(ch));`
	`}`
	`}`
	`return result.toString();`
	`}`

使用参数的签名过程与使用请求头的签名过程类似，如下所示：

- 由于创建预签名 URL 的时候，并不知道有效负载的内容，所以设置常量 UNSIGNED-PAYLOAD。
- 规范查询字符串（Canonical Query String）必须包括除了 X-Amz-Signature 之外的所有上述查询字符串。

- 规范标头必须包括 HTTP Host 标头。如果用户想包含 x-amz-*请求头，这些标头都将参与签名计算。用户可以选择其他的请求头是否参与签名计算。安全起见，让尽可能多的请求头参与签名计算。

3.2.4.3 示例

通过创建预签名 URL 的方式，与其他人共享 examplebucket 中 test.txt 对象，过期时间设置为 7 天（604800 秒）。

```
http://oos-cn.ctyunapi.cn/examplebucket/test.txt
?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=2a948fd3f00ba0925806/20190220/cn/s3/aws4_request
&X-Amz-Date=20190220T095256Z
&X-Amz-Expires=604800
&X-Amz-SignedHeaders=host
&X-Amz-Signature=<signature-value>
```

以下步骤首先说明如何计算签名和构建预签名 URL。示例中使用的访问密钥如下：

参数	值
AWSAccessKeyId	2a948fd3f00ba0925806
AWSSecretAccessKey	ef2017c2e5ffa0b1761717ecbca021da16501384

1) StringToSign

a. 创建规范请求

```
GET

/examplebucket/test.txt

X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=2a948fd3f00ba092580

6%2F20190220%2Fcn%2Fs3%2Faws4_request&X-Amz-Date=20190220T095256Z&X-A

mz-Expires=604800&X-Amz-SignedHeaders=host

host:oos-cn.ctyunapi.cn



host

UNSIGNED-PAYLOAD
```

b. 待签名字符串

```
AWS4-HMAC-SHA256

20190220T095256Z

20190220/cn/s3/aws4_request

023d2e0e5fba779afb9fe621e9413622e7f1aff9ffc7348e55d0805d97cb1571
```

2) 生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("AWS4" +

"<YourSecretAccessKey>","20190220"),"cn"),"s3"),"aws4_request")
```

3) 计算后的签名

```
f566134de06fb3daa22b9649baf82d15d6aa575e146b6ba9aff13a2bde63a1ec
```

4) 预签名 URL

```
http://oos-cn.ctyunapi.cn/examplebucket/test.txt?X-Amz-Algorithm=AWS4-H

MAC-SHA256&X-Amz-Credential=2a948fd3f00ba0925806/20190220/cn/s3/aws4_re

quest&X-Amz-Date=20190220T095256Z&X-Amz-Expires=604800&X-Amz-SignedHead

ers=host&X-Amz-Signature=f566134de06fb3daa22b9649baf82d15d6aa575e146b6b

a9aff13a2bde63a1ec
```

3.3 Bucket 权限控制

对象存储(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（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(包括 Put、Delete 和 Get Object）。其他人无法访问该 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 Object）。任何人（包括匿名访问）可以对该 Bucket 中的 Object 进行读操作（Get Object）。

- 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 Put，Get 和 Delete 操作。这些操作可能会造成 Bucket 所有者数据的增加或者丢失，且所有这些操作产生的费用由该 Bucket 的所有者承担，所以请慎用该权限。

  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。

3.4 Policy 安全策略

3.4.1 介绍

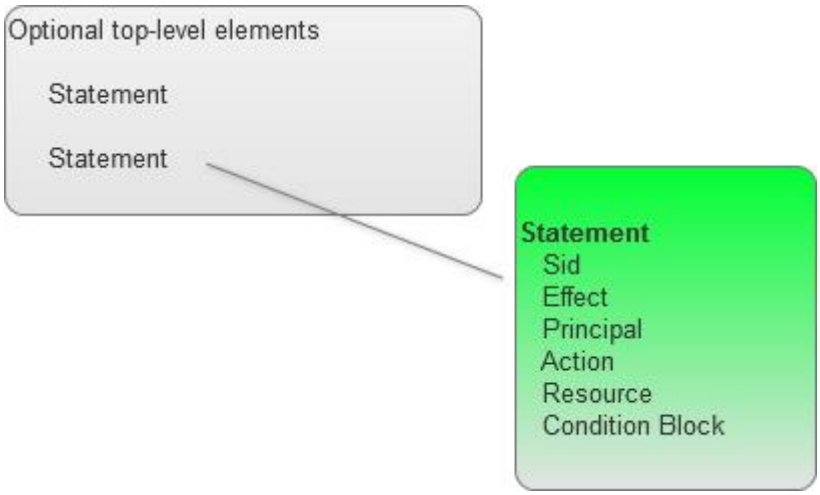
Bucket policy 用于定义 OOS 资源的访问权限。Policy 可以用于：

- ✓ 允许/拒绝 bucket 级别的权限
- ✓ 允许/拒绝 object 级别的权限

Policy 本身是一个 JSON 字符串，限制在 20KB。一个 policy 包括：

- ✓ 可选的 policy 基本信息
- ✓ 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。



3.4.2 Policy 元素

下面介绍 statements 中的各个元素。

3.3.2.1 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"
```

3.3.2.2 Id

Id 是 policy 的一个可选标识。我们推荐使用 UUID 来指明 policy。

3.3.2.3 Statement

Statement 是个重要的元素，它可以包含多个元素。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":[{...},{...},{...}]
```

1. 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"
```

2. 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"
```

3. 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。

4. 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 权限列表如下。

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 权限列表如下。

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。例如，以下 Action 元素适用于所有 OOS 操作。

```
"Action": "s3:*"
```

还可以使用通配符（*）作为操作名称的一部分。例如下面的 Action 元素，适用于 PutObject, GetObject, DeleteObject

```
"Action": "s3:*Object"
```

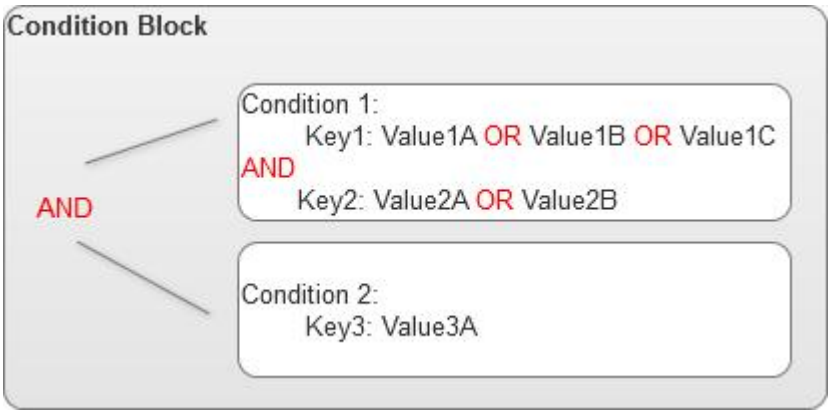
5. Resource

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*"
```

6. 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。



（1） 条件运算符

条件运算符是条件的“动词”形式，可指定 OOS 执行的比较类型。条件运算符可分为以下类别：

1) String Conditions

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.mysite.com/”开头的请求，可以这样配置：

```
"StringLike":{
         "aws:Referer":[
           "http://www.mysite.com/*"
         ]
       }
```

2) 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"}}
  }
}
```

3) 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

（2） 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。

3.4.3 示例

1. 下面是一个定义 Referer Policy 的例子

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

2. 下面是一个定义 IP Policy 的例子

```
{
   "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"
            }
         }
      }
   ]
}
```

3. 下面的例子可向匿名用户授予只读权限

下面的示例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。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AddPerm",
      "Effect":"Allow",
      "Principal": "*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::examplebucket/*"]
    }
  ]
}
```

4 HTTP REST 接口

该部分主要介绍的内容是 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。

OOS 的 API 服务端地址请参见 Endpoint 列表。

4.1 关于 Service 的操作

4.1.1 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.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

请求参数

该操作不使用请求参数。

返回结果

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.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
<?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>
```

4.1.2 GET Regions

获取资源池中的索引位置和数据位置列表。

- 请求语法

```
GET /?regions HTTP/1.1
HOST:oos.ctyunapi.cn
Date:date
Authorization: signatureValue
```

- 请求参数

regions，表示要获取索引位置和数据位置的列表。

- 返回结果

```
HTTP/1.1 200 OK
x-amz-request-id: 236A8905248E5A01
Date: Mon, 03Sep 2017 12:00:00 GMT
Content-Length: 100
Connection: close
Server: CTYUN

<BucketRegions>
    <MetadataRegions>
        <Region>ChengDu</Region>
        <Region>ShenYang</Region>
    </MetadataRegions>
    <DataRegions>
        <Region>ChengDu</Region>
        <Region>ShenYang</Region>
    </DataRegions>
</BucketRegions>
```

- 响应元素

名称	描述
BucketRegions	Bucket 的索引位置和数据位置 类型：容器
MetadataRegions	Bucket 的索引位置 类型：容器
DataRegions	Bucket 的数据位置 类型：容器
Region	索引位置和数据位置的值 类型：字符串

4.2 关于 Bucket 的操作

4.2.1 PUT Bucket

Put 操作用来创建一个新的 bucket。只有在 OOS 中注册的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 命名规范。

请求语法

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

<CreateBucketConfiguration
xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <MetadataLocationConstraint>
        <Location>MetadataRegion</Location>
    </MetadataLocationConstraint >
    <DataLocationConstraint>
        <Type>RegionType</Type>
        <LocationList>
            <Location>DataRegion</Location>
            <Location>DataRegion</Location>
        </LocationList>
        <ScheduleStrategy>Strategy</ScheduleStrategy>
    </DataLocationConstraint >
</CreateBucketConfiguration>
```

请求元素

名称	描述	是否必须
CreateBucketConfiguration	设置 bucket 索引位置和数据位置的容器。 类型：容器	是
MetadataLocationConstraint	设置 bucket 的索引位置。 类型：容器 父节点：CreateBucketConfiguration	是

DataLocationConstraint	设置 bucket 的数据位置。 类型：容器 父节点：CreateBucketConfiguration	否
Type	数据位置的类型 类型：枚举 有效值：Local(本地) |Specified（指定位置） 默认：Local 父节点：DataLocationConstraint	否
LocationList	指定的数据位置 类型：容器 父节点：DataLocationConstraint	否
Location	数据位置所在的地点 类型：String 有效值：例如 [ZhengZhou|ChengDu|ShenYang|...] 默认值：无 父节点：MetadataLocationConstraint| DataLocationConstraint	否
ScheduleStrategy	指定数据时的调度策略 类型：枚举 有效值：Allowed(允许 OOS 自动调 度)|NotAllowed（不允许 OOS 自动调度） 默认：Allowed 父节点：DataLocationConstraint	否

请求参数

名称	描述	是否必须
x-amz-acl	设置创建 bucket 的 ACL （Access Control List）	否

请求示例

请求创建一个名叫 picture 的 bucket，索引位置设置为 Beijing，数据位置设置为优先本地，bucket 权限设置为公开。

```
PUT / HTTP/1.1
Host: picture.oos.ctyunapi.cn
Content-Length:200
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: authorization string
x-amz-acl: public

<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <MetadataLocationConstraint>
        <Location>BeiJing</Location>
    </MetadataLocationConstraint>
    <DataLocationConstraint>
        <Type> Local </Type>
    </DataLocationConstraint>
</CreateBucketConfiguration >
```

返回示例

```
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
```

请求创建一个名叫 picture 的 bucket ,索引位置设置为 Beijing ,数据位置设置为北京、上海，调度策略是允许 OOS 自动调度：

```
PUT / HTTP/1.1
Host: picture.oos.ctyunapi.cn
Content-Length:200
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: authorization string

<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <MetadataLocationConstraint>
        <Location>BeiJing</Location>
    </MetadataLocationConstraint>
    <DataLocationConstraint>
        <Type> Specified </Type>
         <LocationList>
            <Location>BeiJing</Location>
            <Location>ShangHai</Location>
         </LocationList>
         <ScheduleStrategy>Allowed</ScheduleStrategy>
    </DataLocationConstraint>
</CreateBucketConfiguration >
```

4.2.2 GET Bucket location

这个 Get 操作用来获取 bucket 的索引位置和数据位置，只有 bucket 的所有者才能执行此操作。

请求语法

```
GET /?location HTTP/1.1
Host: BucketName.oos.ctyunapi.cn
Content-Length: length
Date: date
Authorization: signatureValue
```

响应元素

名称	描述
CreateBucketConfiguration	设置 bucket 索引位置和数据位置的容器。 类型：容器
MetadataLocationConstraint	设置 bucket 的索引位置。

	类型：容器 父节点：CreateBucketConfiguration
DataLocationConstraint	设置 bucket 的数据位置。 类型：容器 父节点：CreateBucketConfiguration
Type	数据位置的类型 类型：枚举 有效值：Local(本地) |Specified（指定位置） 默认：Local 父节点：DataLocationConstraint
LocationList	指定的数据位置 类型：容器 父节点：DataLocationConstraint
Location	数据位置所在的地点 类型：String 有效值：例如[ZhengZhou|ChengDu|ShenYang|...] 默认值：无 父节点：MetadataLocationConstraint| DataLocationConstraint
ScheduleStrategy	指定数据时的调度策略 类型：枚举 有效值 :Allowed(允许 OOS 自动调度)|NotAllowed(不允许 OOS 自动调度） 默认：Allowed 父节点：DataLocationConstraint

请求示例

获取一个名叫 picture 的 bucket，索引位置为 Beijing，数据位置为优先本地：

```
GET /?location HTTP/1.1
Host: picture.oos.ctyunapi.cn
Content-Length:200
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: authorization string
```

返回示例

```
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

<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <MetadataLocationConstraint>
        <Location>BeiJing</Location>
    </MetadataLocationConstraint>
    <DataLocationConstraint>
        <Type> Local </Type>
    </DataLocationConstraint>
</CreateBucketConfiguration >
```

4.2.3 GET Bucket acl

这个 Get 操作用来获取 bucket 的 ACL 信息，用户必须对改 bucket 有读权限。

请求语法

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

响应

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

请求一个指定 bucket 的 ACL 信息

```
GET ?acl HTTP/1.1
Host: bucket.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
HTTP/1.1 200 OK
x-amz-request-id: 318BC8BC148832E5
Date: Mon, 03Sep 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</ID>
    <DisplayName>CustomersName@email.com</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>
```

4.2.4 GET Bucket (List Objects)

这个 Get 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 的子集。

要执行该操作，需要对操作的 bucket 拥有读权限。

请求语法

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

请求参数

名称	描述	是否必须
Delimiter	一个 delimiter 是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。	否
Marker	指明在 bucket 中 list object 的起始位置，Amazon S3 中 list object 按照阿拉伯字母顺序	否
max-key	设置返回结果中最多显示的数量，如果查询结果超过最大数量，另一个变量是否翻页会标记为 true<IsTruncated>True<IsTruncated>,用来返回剩余的查询结果	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

返回元素

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。
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。
DisplayName	Object 的所有者显示名称
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
ID	对象拥有者的 ID
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
Owner	Bucket 的拥有者
Prefix	关键字以特定的前缀开始
Size	记录对象 object 的大小
StorageClass	STANDARD 或 REDUCED_REDUNDANCY 或 EC_N_M

请求示例

```
GET / HTTP/1.1
Host: BucketName.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 17:50:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
```

返回示例

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Name>bucket</Name>
<Prefix/>
<Marker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Contents>
    <Key>my-image.jpg</Key>
    <LastModified>2012-09-03T17:50:30.000Z</LastModified>
    <ETag>&amp;quot;fba9dede5f27731c9771645a39863328&amp;quot;</ETag>
    <Size>434234</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
        <ID>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06
    a</ID>
        <DisplayName>yourmail@mail.com</DisplayName>
    </Owner>
</Contents>
<Contents>
    <Key>my-third-image.jpg</Key>
    <LastModified>2012-09-03 T17:50:30.000Z</LastModified>
    <ETag>&amp;quot;1b2cf535f27731c974343645a3985328&amp;
    <Size>64994</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
        <ID>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06
    a</ID>
        <DisplayName>yourmail@mail.com</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.ctyunapi.cn
Date: Mon, 03 Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回的结果集为

```
<ListBucketResult xmlns="http://oos.ctyunapi.cn/doc/2012-09-03/">
  <Name>doc</Name>
  <Prefix>doc/</Prefix>
  <Marker>doc/F</Marker>
  <MaxKeys>40</MaxKeys>
  <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>canonical-user-id</ID>
      <DisplayName>display-name</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>Feb/</Prefix>
    <Prefix>Ant/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

4.2.5 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求示例

删除名叫 doc 的 bucket：

```
DELETE / HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

4.2.6 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 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求语法

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

Policy written in JSON
```

请求的内容是一个包含 Policy 语句的 JSON 串。

请求示例

```
PUT /?policy HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

{
"Version":"2012-10-17",
"Id":"aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect":"Allow",
        "Sid":"1",
        "Principal" : {
        "AWS":"*"
        },
        "Action":["s3:*"],
         "Resource":"arn:aws:s3:::bucket/*",
    }
 ]
}
```

返回示例

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

4.2.7 GET Bucket Policy

在 GET 操作的 url 中加上 policy，可以获得指定 bucket 的 policy。只有 bucket 的 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求示例

```
GET /?policy HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
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/*",
    }
 ]
}
```

4.2.8 DELETE Bucket Policy

在 DELETE 操作的 url 中加上 policy，可以删除指定 bucket 的 policy。只有 bucket 的 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 204 NoContent。

请求语法

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

请求示例

```
DELETE /?policy HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

4.2.9 PUT Bucket WebSite

在 PUT 操作的 url 中加上 website，可以设置 website 配置。如果 bucket 已经存在了 website，此操作会替换原有 website。只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

WebSite 功能可以让用户将静态网站存放到 OOS 上。对于已经设置了 WebSite 的 Bucket，当用户访问 http://bucketName.oos-website-cn.oos.ctyunapi.cn 时，会跳转到用户指定的主页，当出现 4XX 错误时，会跳转到用户指定的出错页面。

如果想通过自有域名的形式（例如 http://yourdomain.com/login.html）而非通过第三方域名的形式（例如 http://yourdomain.com.oos.ctyunapi.cn/login.html）访问，可以创建一个名为“yourdomain.com”的 bucket，并在域名管理系统中将“yourdomain.com”增加一个别名记录“oos.ctyunapi.cn”。

请求语法

```
PUT /?website HTTP/1.1
Host: BucketName.oos.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>
</WebsiteConfiguration>
```

请求元素

名称	描述	是否必须
WebsiteConfiguration	请求的容器	是
IndexDocument	Suffix 元素的容器	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	是
ErrorDocument	Key 的容器	否
Key	如果出现 4XX 错误，会返回指定的 Object	有条件的

请求示例

```
PUT /?website HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

<WebsiteConfiguration xmlns='http://s3.amazonaws.com/doc/2006-03-01/'>
    <IndexDocument>
        <Suffix>index.html</Suffix>
    </IndexDocument>
    <ErrorDocument>
        <Key>404.html</Key>
    </ErrorDocument>
</WebsiteConfiguration>
```

返回示例

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

4.2.10 GET Bucket WebSite

在 GET 操作的 url 中加上 website，可以获得指定 bucket 的 website。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

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

请求示例

```
GET /?website HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
HTTP/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03Sep 2012 12:00:00 GMT
Connection: close
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>404.html</Key>
    </ErrorDocument>
</WebsiteConfiguration>
```

4.2.11 DELETE Bucket WebSite

在 DELETE 操作的 url 中加上 website ,可以删除指定 bucket 的 website。只有 bucket 的 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求示例

```
DELETE /?website HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

4.2.12 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.ctyunapi.cn
Date: Date
Authorization: Signature
```

请求参数

名称	描述	是否必须
delimiter	delimiter 是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 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	否

	prefix 来对 key 进行分组）。	
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 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。	否

返回元素

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 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

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: Integer 父节点: Upload
UploadId	分片上传过程的 ID 号 类型: Integer 父节点: Upload
Initiator	指定执行此次分片上传过程的用户账号 子节点：ID，DisplayName 类型：容器 父节点：Upload
ID	OOS 账号的 ID 号 类型：String 父节点：Initiator，Owner
DisplayName	OOS 账号的账户名 类型：String 父节点：Initiator，Owner
Owner	用来标识对象的拥有者 子节点：ID，DisplayName 类型：容器 父节点：Upload
StorageClass	对象的存储类型（STANDARD 或 REDUCED_REDUDANCY 或 EC_N_M） 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	一个 delimiter 是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 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

下面的请求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.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

下面的示例表示这次 List 操作无法将分片上传过程列举完，需要指定 NextKeyMarker 和 NextUploadIdMarker 元素。用户需要在下一次 List 操作中指定这两个值。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: 1330
Connection: keep-alive
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</Ne
xtUploadIdMarker>
    <MaxUploads>3</MaxUploads>
    <IsTruncated>true</IsTruncated>
    <Upload>
        <Key>my-divisor</Key>
        <UploadId>XMgbGlrZSBlbHZpbmcncyBub3QgaGF2aW5nIG11Y2ggbHVjaw</UploadId
>
        <Initiator>
            <ID>mailaddress@email.com</ID>
            <DisplayName>user1-11111a31-17b5-4fb7-9df5-b111111f13de</DisplayName>
        </Initiator>
        <Owner>
            <ID>mailaddress@email.com</ID>
            <DisplayName>OwnerDisplayName</DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
        <Initiated>2010-11-10T20:48:33.000Z</Initiated>
    </Upload>
    <Upload>
        <Key>my-movie.m2ts</Key>
        <UploadId>VXBsb2FkIElEIGZvciBlbHZpbmcncyBteS1tb3ZpZS5tMnRzIHVwbG9hZA</Up
loadId>
        <Initiator>
            <ID>mailaddress@email.com</ID>
            <DisplayName>InitiatorDisplayName</DisplayName>
        </Initiator>
        <Owner>
            <ID>mailaddress@email.com</ID>
            <DisplayName>OwnerDisplayName</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@email.com</ID>
            <DisplayName>user1-22222a31-17b5-4fb7-9df5-b222222f13de</DisplayName>
        </Initiator>
        <Owner>
            <ID>mailaddress@email.com</ID>
            <DisplayName>OwnerDisplayName</DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
        <Initiated>2010-11-10T20:49:33.000Z</Initiated>
    </Upload>
</ListMultipartUploadsResult>
```

4.2.13 PUT Bucket Logging

在 PUT 操作的 url 中加上 logging，可以进行添加/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 的 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.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	否

	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。	
TargetPrefix	生成的 log 文件将以此为前缀命名	否

以下是启动日志的示例：

请求示例

```
PUT /?logging HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

<?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>
```

返回示例

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

以下是取消日志的示例：

请求示例

```
PUT /?logging HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

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

返回示例

```
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。唯一字符串部分没有实际含义，可以被忽略。

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@email.com	源 Bucket 的 Owner
Bucket 名称	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/Aug/2012:22:34:02 +0000]	请求到达的时间，格式是 [%d/%B/%Y:%H:%M:%S %z]
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@email.com	请求者的邮箱。如果是匿名访问，显示“Anonymoous”
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“-”

4.2.14 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 的 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

响应元素

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。
TargetPrefix	生成的 log 文件将以此为前缀命名

请求示例

```
GET /?logging HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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/>
```

4.2.15 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 200 OK。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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求示例

```
HEAD / HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

响应示例

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

4.2.16 PUT Bucket Lifecycle

存储在 OOS 中的对象有时需要有生命周期。比如，用户可能上传了一些周期性的日志文件到 bucket 中，一段时间后，用户可能不需要这些日志对象了。之前，用户需要自己手动删除这些不用的对象，现在可以使用对象到期功能来指定 bucket 中对象的生命周期。

当对象的生命周期结束时，OOS 会异步删除他们。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。对象到期后，用户将不会再为到期的对象付费。

用户可以通过在 bucket 中配置生命周期，来为对象设置到期时间。一个生命周期的配置中最多可以包含 100 条规则。每个规则指定了对象的前缀和生命周期，生命周期是指从对象创建开始到被删除之前的天数。生命周期的值必须是个正整数。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。比如，一个对象是 GMT 2016 年 1 月 15 日 10:30 分创建的，生命周期是 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，来重新计算到期时间。

当用户为 bucket 设置了生命周期时，这些规则将同时应用于已有对象和之后新创建的对象。比如，用户今天增加了一个生命周期的配置，指定某些前缀的对象 30 天后过期，那么 OOS 将会把满足条件的 30 天前创建的对象都加入到待删除队列中。

用户可以使用 GET, HEAD API 来查询对象的到期时间，这些接口会通过响应头来返回对象的到期时间信息。

用户可以使用 OOS access logs 来查询对象是何时被删除的。OOS 删除到期对象后，会在 access logs 中记录一条日志，操作项是"OOS.EXPIRE.OBJECT"。

Put Bucket Lifecycle 接口用于设置 bucket 的生命周期，如果生命周期的配置已经存在，将会被替换。用户可以通过设置生命周期，来让 OOS 删除过期的对象。

请求语法

```
PUT /?lifecycle HTTP/1.1
Host: BucketName.oos.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 100 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。 类型：字符串 父节点：Rule	否
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	是

Status	如果是 Enabled ,那么规则立即生效。如果是 Disabled，那么规则不会生效。 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

例子 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: doc.oos.ctyunapi.cn

Date: Tue, 13 Dec 2011 17:54:50 GMT

Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==

Authorization: AWS AKIAIOSFODNN7EXAMPLE:z6mvnXscCWad60vdmB9xZVVZn46=

Content-Length: 207

<LifecycleConfiguration>

 <Rule>

     <ID>delete-logs-in-30-days-rule</ID>

     <Prefix>logs</Prefix>

     <Status>Enabled</Status>

     <Expiration>

         <Days>30</Days>

     </Expiration>

 </Rule>
</LifecycleConfiguration>
```

返回示例

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

例子 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.ctyunapi.cn

Date: Tue, 13 Dec 2011 17:54:50 GMT

Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==

Authorization: AWS AKIAIOSFODNN7EXAMPLE:z6mvnXscCWad60vdmB9xZVVZn46=

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>
```

返回示例

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

例子 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.ctyunapi.cn
Date: Tue, 13 Dec 2011 17:54:50 GMT
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Authorization: AWS AKIAIOSFODNN7EXAMPLE:z6mvnXscCWad60vdmB9xZVVZn46=
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>
```

返回示例

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

例子 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.ctyunapi.cn

Date: Tue, 13 Dec 2011 17:54:50 GMT

Authorization: AWS AKIAIOSFODNN7EXAMPLE:z6mvnXscCWad60vdmB9xZVVZn46=

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>
```

4.2.17 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

响应元素

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	如果是 Enabled，那么规则立即生效。如果是 Disabled，那么规则不会生效。 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	描述	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.ctyunapi.cn

Date: Tue, 13 Dec 2011 17:54:50 GMT

Authorization: AWS AKIAIOSFODNN7EXAMPLE:z6mvnXscCWad60vdmB9xZVVZn46=
```

返回示例

```
HTTP/1.1 200OK
x-amz-request-id: BDC4B83DF5096BBE
Date:Tue, 13 Dec 2011 19:14:41 GMT
Connection: close
Content-Length: 267
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>
```

4.2.18 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求示例

```
DELETE /?lifecycle HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Wed, 14 Dec 2011 05:37:16 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

返回示例

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

4.2.19 PUT Bucket accelerate

在 PUT 操作的 url 中加上 accelerate，可以进行添加或修改 CDN IP 白名单的操作。如果 bucket 已经配置了 CDN 加速，此操作会替换原有配置。只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。一个 bucket 最多配置 5 个 IP 白名单地址段。

请求语法

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

<AccelerateConfiguration xmls="http://s3.amazonaws.com/doc/2006-03-01/">
    <Status>transfer acceleration state</Status>
    <IPWhiteLists>
        <IP>IP1</IP>
        <IP>IP2</IP>
    </IPWhiteLists>
</AccelerateConfiguration>
```

请求元素

名称	描述	是否必须
AccelerateConfiguration	配置 CDN 加速的容器 类型：容器	是
Status	设置 bucket 的 CDN 加速状态 类型：枚举 有效值：Enabled|Suspended 父节点：AccelerateConfiguration	是
IPWhiteLists	配置 IP 白名单的容器 类型：容器	否
IP	CDN 服务提供商的 IP 白名单 ,IP 值需符合 RFC 4632 中描述的 CIDR 表示法。	否

请求示例

启动 bucket 的 CDN 加速功能，设置 IP 白名单列表包含：36.111.88.0/24 和 114.80.1.136。

```
PUT /? accelerate HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS fad0e782cd5132563e38:xQE0diMbLRepdf3YB+FIEXAMPLE=

< AccelerateConfiguration xmls="http://s3.amazonaws.com/doc/2006-03-01/">
    <Status> Enabled</Status>
    <IPWhiteLists>
    <IP>36.111.88.0/24</IP>
    <IP>114.80.1.136</IP>
    </IPWhiteLists>
</AccelerateConfiguration>
```

返回示例

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

4.2.20 GET Bucket accelerate

在 GET 操作的 url 中加上 accelerate，可以获得指定 bucket 的 cdn 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。

如果 bucket 没有配置过 CDN 加速，那么将不会返回状态信息。

请求语法

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

请求示例

```
GET /?accelerate HTTP/1.1
Host: doc.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 12:00:00 GMT
Authorization: AWS fad0e782cd5132563e38:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

< AccelerateConfiguration
xmls="http://s3.amazonaws.com/doc/2006-03-01/">
    <Status> Enabled</Status>
    <IPWhiteLists>
        <IP>36.111.88.0/24</IP>
        <IP>114.80.1.136</IP>
    </IPWhiteLists>
</AccelerateConfiguration>
```

如果 bucket 没有配置过 CDN 加速，将返回以下信息：

```
<AccelerateConfiguration

xmlns="http://s3.amazonaws.com/doc/2006-03-01/"/>
```

4.2.21 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.example.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.example.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.example.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.example.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 的 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 64KB。

请求语法

```
PUT /?cors HTTP/1.1
Host: bucketname.oos.ctyunapi.cn
Content-Length: length
Date: date
Authorization: authorization string
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.	是

	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	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://*.example.com。 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 (秒) 父节点: CORSRule	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://www.example1.com 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://www.example2.com。

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.ctyunapi.cn
x-amz-date: Tue, 21 Aug 2012 17:54:50 GMT
Content-MD5: 8dYiLewFWZyGgV2Q5FNI4W==
Authorization: authorization string
Content-Length: 445

<CORSConfiguration>
 <CORSRule>
   <AllowedOrigin>http://www.example1.com</AllowedOrigin>

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

   <AllowedHeader>*</AllowedHeader>
 </CORSRule>
 <CORSRule>
   <AllowedOrigin>http://www.example2.com</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
Server:OOS
```

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。

4.2.22 GET Bucket cors

返回 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

```
GET /?cors HTTP/1.1
Host: bucketname.oos.ctyunapi.cn
Date: date
Authorization: authorization string
```

响应元素

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://*.example.com。 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 (秒) 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

下面的示例获取 bucket 的跨域配置信息。

```
GET /?cors HTTP/1.1
Host: bucketname.oos.ctyunapi.cn
Date: date
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200 OK
x-amz-request-id: 0CF038E9BCF63097
Date: Wed, 13 Dec 2017 19:14:42 GMT
Server: OOS
Content-Length: 280

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

4.2.23 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.ctyunapi.cn
Date: date
Authorization: authorization string
```

请求示例

```
DELETE /?cors HTTP/1.1
Host: examplebucket.oos.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
Server: OOS
```

4.3 关于 Object 的操作

4.3.1 PUT Object

Put 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。

请求语法

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

请求头格式

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。	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 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	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 默认值: STANDARD 可选值: STANDARD | REDUCED_REDUNDANCY|EC_N_M 其中，EC_N_M 表示用 Erasure Code 方式存储数据，表示将 N 份数据生成 M 个校验数据，在 N+M 个数据块中，丢失其中任意的 M 块数据，都可以将 M 块数据恢复出来。 如果上传对象时，不加 x-amz-storage-class 请求头，OOS 会使用动态副本策略，为对象指定副本模式。	
x-ctyun-data-location	设置bucket的数据位置。 类型：key-value形式 有效值： type=[Local|Specified],location=[ChengDu|ShenYang|...],scheduleStrategy=[Allowed|NotAllowed] type=local表示就近写入本地。type= Specified表示指定位置。location表示指定的数据位置，可以填写多个，以逗号分隔。scheduleStrategy表示调度策略，是否允许OOS自动调度数据存储位置。	否

请求示例

在名叫 myBucket 的 bucket 中，存储一张叫 my-image.jpg 的图片。

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 17:50:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: image/ipeg
Expect: 100-continue
Content-Length: 11434
```

返回示例

```
HTTP/1.1 100 Continue

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
```

副本模式

默认情况下，OOS 将采用动态冗余模式的方式来存储数据。但对于那些不太重要，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。如果用户想使用减少冗余策略，那么可以在请求头中设置 x-amz-storage-class 为 REDUCED_REDUNDANCY。用户也可以选择使用 Erasure Code 方式存储数据，即将 x-amz-storage-class 请求头设置为 EC_N_M。其中，EC_N_M 表示用 Erasure Code 方式存储数据，表示将 N 份数据生成 M 个校验数据，在 N+M 个数据块中，丢失其中任意的 M 块数据，都可以将 M 块数据恢复出来。此外，用户也可以将 x-amz-storage-class 请求头设置为 EC_N_M_MinReplicaNum，其中 MinReplicaNum 是最小副本数。例如，将 x-amz-storage-class 请求头设置为 EC_2_1_2，即表示使用 EC 2+1 的方式存储对象，最小副本数是 2。

4.3.2 GET Object

GET 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.ctyunapi.cn
Date: date
Authorization: signatureValue
```

请求变量

变量	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	否

	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”)	
response-content-encoding	设置返回头中的 Content-Encoding	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	下载指定范围内大小的一个对象	否
If-Modified-Since	只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。
x-ctyun-metadata-location	获取对象的索引位置。 类型：枚举 有效值： ChengDu|ShenYang|...
x-ctyun-data-location	获取 bucket 的数据位置。 类型：枚举 有效值： ChengDu|ShenYang|...

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.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
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]
```

4.3.3 DELETE Object

Delete 操作移除指定的对象，要求用户要对对象所在的 bucket 拥有写权限。

请求语法

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

请求示例

下一个例子中删除一个叫 my-image.jpg 的图片对象

```
DELETE /my-image.jpg HTTP/1.1
Host: bucket.oos.ctyunapi.cn
Date: Mon, 03 Sep 2012 17:50:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: image/ipeg
```

返回示例

```
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
```

4.3.4 PUT Object - Copy

通过 PUT 操作创建一个存储在 OOS 里的对象的拷贝。PUT 操作类似于执行一个 GET 然后在执行一次 PUT。增加请求头，x-awz-copy-source，使用 PUT 操作将源对象存入指定 bucket。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.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
<request metadata>
Authorization: signatureValue
Date: date
```

请求头格式

使用者可以通过以下请求头实现对应操作。

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	是
x-amz-metadata-directive	指明元数据是否是源对象的拷贝或者被请求头提供的元数据覆盖。如果是拷贝，元数据保持不变，否则所有原始元数据都被指定的元数据覆盖	否

	有效值：COPY | REPLACE	
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 412HTTP 状态码错误。	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	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	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	否
x-amz-storage-class	新对象的存储类型，可以采用标准存储模式、减少冗余模式和 EC 模式。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 默认值: STANDARD 可选值: STANDARD | REDUCED_REDUNDANCY|EC_N_M	否
x-ctyun-data-location	设置bucket的数据位置。 类型：key-value形式 有效值： type=[Local|Specified],location=[ChengDu|ShenYang|...],ScheduleStrategy=[Allowed|NotAllowed] Type=local表示就近写入本地。type= Specified表示指定位置。location表示指定的数据位置，可以填写多个，以逗号分隔。ScheduleStrategy表示调度策略，是否允许OOS自动调度数据存储位置。	否

默认情况下，OOS 采用源对象的副本模式来存储数据。用户也可以通过 x-amz-storage-class 请求头来指定新对象的副本模式。对于普通数据，用户可以选择 STANDARD 标准存储模式。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。如果用户想使用减少冗余策略，那么可以在请求头中设置 x-amz-storage-class 为 REDUCED_REDUNDANCY。用户也可以选择使用 Erasure Code 方式存储数据，即将 x-amz-storage-class 请求头设置为 EC_N_M。其中，EC_N_M 表示用 Erasure Code 方式存储数据，表示将 N 份数据生成 M 个校验数据，在 N+M 个数据块中，丢失其中任意的 M 块数据，都可以将 M 块数据恢复出来。此外，用户也可以将

x-amz-storage-class 请求头设置为 EC_N_M_MinReplicaNum，其中 MinReplicaNum 是最小副本数。例如 将 x-amz-storage-class 请求头设置为 EC_2_1_2 ,即表示使用 EC 2+1 的方式存储对象，最小副本数是 2。

返回元素

名称	描述
CopyObjectResult	包含所有返回元素的容器
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

```
PUT /my-second-image.jpg HTTP/1.1
Host: bucket-sample.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
x-amz-copy-source: /bucket-sample/my-image.jpg
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

示例中执行的操作是将对象 my-image.jpg 拷贝到叫 bucket-sample 的 bucket 中，重命名为 my-second-image.jpg。

返回示例

```
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>
```

4.3.5 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 Upload Part）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: oos.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	否

	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	
Content-Type	用来描述对象数据格式的标准 MIME 类型。 类型：String 默认值：binary/octel-stream 限制：仅 MIME 类型	否
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 可选值：STANDARD|REDUCED_REDUNDANCY 或 EC_N_M 默认值：STANDARD	否
x-ctyun-data-location	设置bucket的数据位置。 类型：key-value形式 有效值： type=[Local|Specified],location=[ChengDu|ShenYang|...],ScheduleStrategy=[Allowed|NotAllowed] Type=local表示就近写入本地。type= Specified表示指定位置。location表示指定的数据位置，可以填写多个，以逗号分隔。ScheduleStrategy表示调度策略，是否允许OOS自动调度数据存储位置。	否

响应头格式

无响应头信息

返回元素

名称	描述
InitiateMultipartUploadResult	包含所有返回元素的容器 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

示例中执行的操作是初始化一个名为“example-object”对象的分片上传操作。

```
POST /example-object?uploads HTTP/1.1
Host: example-bucket.oos.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 197
Connection: keep-alive
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</Uploa
dId>
</InitiateMultipartUploadResult>
```

4.3.6 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 Initial Multipart Upload 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.ctyunapi.cn
Date: date
Content-Length: Size
```

请求参数

无

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
Content-Length	该分片的大小，以字节为单位。 类型：Integer 默认值：None	是
Content-MD5	该分片数据的 128 位采用 base64 编	否

	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 默认值：None	
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 默认值：None 可选值：100-continue	否

返回元素

无

出错响应

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

示例中的 PUT 请求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.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 10485760
Content-MD5: pUNXr/BjKK5G2UKvaRRrOA==
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

***part data omitted***
```

返回示例

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: keep-alive
Server: CTYUN
```

4.3.7 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 Upload Part 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.ctyunapi.cn
Date: Date
Content-Length: Size
Authorization: Signature

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

请求参数

无

请求头格式

无

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 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

返回元素

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

出错响应

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

请求示例

下面的分片上传请求在 CompleteMultipartUpload 元素中指定了三个片段。

```
POST
/example-object?uploadId=AAAsb2FkIElEIGZvciBlbHZpbmcncyWeeS1tb3ZpZS5tMnRzI
R
RwbG9hZA HTTP/1.1
Host: example-bucket.oos.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 391
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

<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>
```

返回示例

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>
```

4.3.8 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.ctyunapi.cn
Date: Date
Authorization: Signature
```

出错响应

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	404Not Found

	ID 可能非法，分片上传过程可能被终止或者已经完成	

请求示例

下面的请求通过指定上传 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.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

4.3.9 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.ctyunapi.cn
Date: Date
Authorization: Signature
```

请求参数

此次 Get 请求通过使用下表中的参数来返回一个 uploadId 中的 parts 集合。

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 默认值：None	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 默认值：None	是

返回元素

名称	描述
ListPartsResult	包含整个响应的容器 类型：容器 子节点：Bucket, 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
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号 子节点：ID，DisplayName 类型：容器 父节点：ListPartsResult
ID	OOS 账号的 ID 号 类型：String 父节点：Initiator
DisplayName	OOS 账号的账户名 类型：String 父节点：Initiator
Owner	用来标识对象的拥有者 子节点：ID，DisplayName 类型：容器 父节点：ListPartsResult
StorageClass	对象的存储类型（STANDARD 或 REDUCED_REDUNDANCY 或 EC_N_M） 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

出错响应

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=XXBsb2FkIElEIGZvciBlbHZpbmcncyVcdS1tb3ZpZS5tMnRzE
EEw
bG9hZA&max-parts=2&part-number-marker=1 HTTP/1.1
Host: example-bucket.oos.ctyunapi.cn
Date: Mon, 1 Nov 2010 20:34:56 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

```
HTTP/1.1 200 OK
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 985
Connection: keep-alive
Server: CTYUN
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ListPartsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Bucket>example-bucket</Bucket>
    <Key>example-object</Key>
    <UploadId>XXBsb2FkIElEIGZvciBlbHZpbmcncyVcdS1tb3ZpZS5tMnRzEEEwbG9hZA</UploadId>
    <Initiator>
        <ID>mailaddress@email.com</ID>
        <DisplayName>umat-user-11116a31-17b5-4fb7-9df5-b288870f11xx</DisplayName>
    </Initiator>
    <Owner>
        <ID>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
        <DisplayName>someName</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>
```

4.3.10 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.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: Signature
```

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 默认值：None	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 默认值：None	否

以下是一些可选的头

名称	描述	是否必须
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 412HTTP 状态码错误。	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	否
x-amz-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	否
x-amz-copy-source-if-modified-since	如果对象在指定时间点之后被修改	否

	过则执行拷贝操作，否则返回 412 错误	

返回元素

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.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 mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

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

4.3.11 分段上传高级别 API

OOS JAVA SDK 公开了一个高级别 API，用于简化分段上传。用户可以从文件或流上传数据，也可以设置高级选项，例如，用于分段上传的片段大小，或在并发上传时使用的线程数。用户可以使用 PutObjectRequest 和 TransferManagerConfiguration 类来设置这些高级选项。Java API 的 TransferManager 类提供了高级别 API，用户可以使用它来上传数据。TransferManager 会尝试使用多个线程来一次性上传单个文件的多个分段片段。当对象大小很大，而且带宽也很大时，此操作可以大幅度地增加吞吐量。

高级别 API 文件上传过程如下：

1. 创建 TransferManager 类的实例。
2. 用户可以选择从文件还是从流上传数据 ,并执行 TransferManager.upload 方法。

以下 Java 代码示例演示了上述任务。

```
import java.io.File;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class UploadObjectMultipartUploadUsingHighLevelAPI {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";
        String keyName = "*** Provide object key ***";
        String filePath = "*** Path to and name of the file to upload ***";

        TransferManager tm = new TransferManager(new
        ProfileCredentialsProvider());
        System.out.println("Hello");
        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(
        existingBucketName, keyName, new File(filePath));
        System.out.println("Hello2");

        try {
            // Or you can block and wait for the upload to finish
            upload.waitForCompletion();
            System.out.println("Upload complete.");
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

TransferManager 类提供了 abortMultipartUploads 方法 ,用于终止正在上传的分段片段。用户可以在调用 API 时指定日期，删除所有在指定日期之前被初始化，且还在上传的分段片段。

高级别 API 终止分段片段过程如下：

1. 创建 TransferManager 类的实例。
2. 执行 TransferManager. abortMultipartUploads 方法 ,指定 bucket 名称和日期。

以下 Java 代码示例演示了上述任务。

```
import java.util.Date;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.transfer.TransferManager;

public class AbortMPUUsingHighLevelAPI {
    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";
        TransferManager tm = new TransferManager(new
ProfileCredentialsProvider());
        int sevenDays = 1000 * 60 * 60 * 24 * 7;
        Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);
        try {
            tm.abortMultipartUploads(existingBucketName, oneWeekAgo);
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

4.3.12 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。

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 /?deleteHTTP/1.1
Host: BucketName.oos.ctyunapi.cn
Content-Length: Size
Content-MD5: MD5
Authorization: Signature
Date: Date

<?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 默认值：None	是
Content-Length	请求体的长度，参考 RFC 2616 类型：String 默认值：None	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	是

	父节点：无	
Quiet	使用简明信息模式来返回响应，当使用此元素时，需要指定 true。 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 名 类型：String 父节点：*Object*	是

响应元素

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 值：AccessDenied, InternalError
Message	错误的描述 类型：String 父节点：Error

请求示例

下面的请求批量删除一些 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.ctyunapi.cn
Date: Mon, 11 Apr 2011 20:34:56 GMT
Content-MD5: p5/WA/oEr30qrEEl21PAqw==
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Length: 125
Connection: Keep-Alive

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

返回示例

```
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>
```

4.3.13 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。

参数设置

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 partSize，若 partSize 小于 5MB，则会将 partSize 调整至 5MB 默认：5MB
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 5MB，若 partSize 小于 5MB，除了最后一个分片以外，会
将 partSize 调整至 5MB
            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());
        }
    }
}
```

4.3.14 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。

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.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—
```

请求参数

本实现不使用请求参数。

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。 类型：String 默认值：无	条件
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 默认值：无	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 默认值：无	是
key	上传对象的名称。 可以使用${filename}变量来使用用户提供的文件名。例如，如果用户 Betty 上传的文件名为 lolcatz.jpg，字段值指定为 /user/betty/${filename},那么保存的对象名称将会是 /user/betty/lolcatz.jpg。 类型：String 默认值：无	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 的设置方法请查看 Policy 字段的构造部分。 类型：String 默认值：无	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。 如果未指定 success_action_redirect，OOS 将返回在	否

	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 默认值：无 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。 类型: String 默认值: STANDARD 可选值: STANDARD | REDUCED_REDUNDANCY|EC_N_M 更多信息请参考 PUT Object	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。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:	否

	http://www.example.com/ 注意：这个值必须以”/”、http://或 https://开头，且长度不能超过 2KB	
x-ctyun-data-location	设置bucket的数据位置。 类型：key-value形式 有效值： type=[Local|Specified],location=[ChengDu|ShenYang|...], ScheduleStrategy=[Allowed|NotAllowed] Type=local表示就近写入本地。type= Specified表示指定位置。location表示指定的数据位置，可以填写多个，以逗号分隔。ScheduleStrategy表示调度策略，是否允许OOS自动调度数据存储位置。	否

响应格式

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

简单请求

```
POST / HTTP/1.1
Content-Length: 4
Host: BucketName.oos.ctyunapi.cn
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Expect: the 100-continue HTTP status code
ObjectContent
```

简单响应

下面的代码演示了一个简单的响应头。

```
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.ctyunapi.cn/”。

method 必须是 POST。enctype 设置为“multipart/form-data”。

```
<form action="http:// BucketName.oos.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 MB 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。

4.3.15 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.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.example.com. 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

响应头格式

响应头	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.example.com 向名为 examplebucket 的 bucket，发送 PUT 请求。

```
OPTIONS /exampleobject HTTP/1.1
Host: examplebucket.oos.ctyunapi.cn
Origin: http://www.example.com
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.example.com
Access-Control-Allow-Methods: PUT
Access-Control-Expose-Headers: x-amz-request-id
Server: OOS
```

4.3.16 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://oos.ctyunapi.cn/mysite/a.png?Expires=1363760719&AWSAccessKeyId=

mailaddress %40email.cn&Signature=S7FTz%2BerLjBjirI7jKI4LmHXDRA%3D
```

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 KB/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-cn.ctyunapi.cn/test-20180604/6aa3df83gw1f35nhhp70pj20gj0r046l.j
pg?Signature=8l7F/pabWm2%2Bi8iXyExZIXm/eGY%3D&AWSAccessKeyId=08f17977afa1a
87736ac&Expires=1528438576&x-amz-limitrate=2048
```

4.3.17 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。

1、请求语法

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

2、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	否
If-Modified-Since	返回一个在指定时间点后被修改的对象，无结果则返回 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 ,无结果则返回 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

3、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。

4、请求示例

下面示例中返回对象 my-image.jpg

```
HEAD /my-image.jpg HTTP/1.1
Host: bucket.oos.ctyunapi.cn
Date: Mon, 03Sep 2012 22:32:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

5、返回示例

```
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
```

4.4 关于 AccessKey 的操作

OOS 的 IAM 服务端地址请参见 Endpoint 列表。

4.4.1 CreateAccessKey

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，每个账号最多创建 10 个 AccessKey。

请求语法

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

Action=CreateAccessKey
```

请求参数

名称	描述
Action	值是 CreateAccessKey

返回结果

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 ,默认生成的都是普通 key ,值是 false

请求示例

```
POST / HTTP/1.1
Host:oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

Action=CreateAccessKey
```

返回示例

```
<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>
```

4.4.2 DeleteAccessKey

删除一对 AccessKey 和 SecretKey。只有主 key 才能执行此操作。

请求语法

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

Action=DeleteAccessKey&AccessKeyId=accessKeyId
```

请求参数

名称	描述

Action	值是 DeleteAccessKey
AccessKeyId	要删除的 AccessKey

请求示例

```
POST / HTTP/1.1
Host:oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

Action=DeleteAccessKey&AccessKeyId=2d1d5625f6202b08c8db
```

返回示例

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

4.4.3 UpdateAccessKey

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-iam.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue

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

请求参数

名称	描述
Action	值是 UpdateAccessKey
AccessKeyId	要删除的 AccessKey
Status	Key 的状态，启用或禁止。值是 Active, Inactive

IsPrimary	是否是主 key，值是 true，false

请求示例

```
POST / HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

Action=UpdateAccessKey&AccessKeyId=2d1d5625f6202b08c8db&Status=active&IsPr
imary=false
```

返回示例

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

4.4.4 ListAccessKey

列出账号下的主 ke 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。为保证账号安全，list 操作时，不会返回 SecretKey。

请求语法

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

Action=ListAccessKey
```

请求参数

名称	描述
Action	值是 ListAccessKey
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。
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。

请求示例

```
POST /HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=

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。

返回示例

```
<ListAccessKeysResponse>
    <ListAccessKeysResult>
    <UserName>Bob</UserName>
    <AccessKeyMetadata>
    <member>
        <UserName>Bob</UserName>
        <AccessKeyId>2d1d5625f6202b08c8db</AccessKeyId>
        <Status>Active</Status>
        <IsPrimary>true</IsPrimary>
    </member>
    <member>
    <UserName>Bob</UserName>
        <AccessKeyId>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 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>
```

5 错误响应

5.1 REST 错误响应

当请求出错时，OOS 返回以下格式的错误响应信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
    <Code>NoSuchKey</Code>
    <Message>The resource you requested does not exist</Message>
    <Resource>/mybucket/myfoto.jpg</Resource>
    <RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

响应元素说明如下。

名称	描述
Code	错误码是描述出错信息的字符串。详细的错误码信息可以参见后面的列表。
Message	错误信息的简单描述。
RequestId	请求的 ID。
Resource	出错相关的 bucket 或 object 信息。

5.2 错误码列表

错误码	描述	HTTP 状态码
AccessDenied	权限被拒绝	403 Forbidden

AuthorizationHeaderMalformed	签名请求头不合法	400 Bad Request
AuthorizationQueryParametersError	签名请求参数不合法	400 Bad Request
BadDigest	用户指定的 Content-MD5 与 OOS 计算的不匹配	400 Bad Request
BucketAlreadExists	请求的 bucket 的 name 已经不可用，请选择一个不同的名字尝试。	409 Conflict
BucketNotEmpty	试图删除的的 bucket 非空	409 Conflict
CanNotModifyMetadataLocation	索引位置不允许修改	400 Bad Request
IncompleteBody	对象长度小于 Content-Length 请求头中指定的长度	400 Bad Request
Internal Error	OOS 服务错误，请重试	500 InternalServerError
InvalidAccessKeyId	你使用的 OOS ID 不存在	403 Forbidden
InvalidArgument	无效的请求参数	400 Bad Request
InvalidBucketName	指定的 bucket 无效	400 Bad Request
InvalidLocationConstraint	用户提供的配置索引位置和数据位置的 xml 格式不合法	400 Bad Request
InvalidObjectName	对象名称不合法	400 Bad Request
InvalidPolicyDocument	策略不合法	400 Bad Request
InvalidRequest	IP 白名单不合法	400 Bad Request
InvalidRequest	IP 白名单个数超过限制	400 Bad Request
InvalidStorageClass	副本模式不合法	400 Bad Request
InvalidURI	无法解析指定的 URI	400 Bad Request
MalformedXML	XML 格式不合法	400 Bad Request
MethodNotAllowed	指定的方法不允许操作该资源	405 Method Not Allowed

MissingContentLength	在 HTTP 头中必须提供 ContentLength	411 length required
NoSuchBucket	指定的 bucket 不存在	404 Not Found
NoSuchKey	指定对象的 Key 不存在	404 Not Found
NoSuchLifecycleConfiguration	生命周期的配置不存在	404 Not Found
NotVerify	用户未激活	403 Forbidden
PreconditionFailed	至少有一个指定的条件不满足	412 Precondition Failed
PutObjectTooFast	并发写同一个对象或分段片段时冲突	400 Bad Request
RequestTimeTooSkewed	请求的时间和服务器时间相差太大	403 Bad Request
SignatureDoesNotMatch	我们计算出的请求中的签名与用户提供的签名不一致。请检查 secretKey 和签名算法。	403 Forbidden
SlowDown	OOS 服务繁忙，请重试	503 Service Unavailable
TooManyBuckets	Bucket 个数超出限制	400 Bad Request

6 SDK 开发

OOS 目前支持通过多种语言的 SDK 来访问，SDK 支持的语言类型包括：Java，Android，iOS，C，Python。

以下主要介绍如何使用 OOS 的 Java SDK。

6.1 配置 SDK

导入 SDK 包到项目中

首先将 OOS Java SDK 添加到编译路径中。

下载第三方包，并添加到编译路径中，地址：https://oos-cn.ctyunapi.cn/docs/oos/sdk/java/third-party.zip

这样即可以在项目中调用 OOS SDK。

使用 SDK 访问服务器

首先，创建 Client 对象，用于向服务器端发送请求，java 代码如下：

```
AmazonS3 client= new AmazonS3Client(new
PropertiesCredentials(S3Sample.class.getResourceAsStream("AwsCrede
ntials.properties")));
client.setEndpoint(http://oos.ctyunapi.cn);
```

其中 AwsCredentials.properties 用于存储 accessKey 和 secretKey(此信息可以在控制台—账号管理—API 密钥管理中查看到)，例如：

```
accessKey =test
secretKey =test
```

http://oos.ctyunapi.cn 是 OOS 服务端的地址，详见 Endpoint 列表。

然后就可以调用 SDK 中的方法，对 Service,Bucket,Object 进行 PUT,GET,DELETE,HEAD 等操作。

SDK FAQ

1. 如何设置请求的重试功能

在创建 AmazonS3Client 时，可以配置客户端发送请求的最大重试次数，当服务端返回 5XX 错误，或连接超时等错误信息时，AmazonS3Client 会自动重发请求。配置方法如下：

```
ClientConfiguration cc = newClientConfiguration();
cc.setMaxErrorRetry(2);//这里设置重试两次，如果不设置的话，默认重试三次
AmazonS3Client oosclient = newAmazonS3Client(new AWSCredentials() {
public String getAWSAccessKeyId() {
return"yourAccessKey";
        }

public String getAWSSecretKey() {
return"yourSecretKey";
        }
 }, cc);
```

2. 如何设置超时时间

在创建 AmazonS3Client 时 ,可以配置客户端到服务器端的连接超时和 socket 超时时间，代码示例如下：

```
ClientConfiguration cc = newClientConfiguration();
cc.setConnectionTimeout(30*1000);//设置连接的超时时间，单位毫秒
cc.setSocketTimeout(30*1000) ;//设置socket超时时间，单位毫秒
AmazonS3Client oosclient = newAmazonS3Client(new AWSCredentials() {
public String getAWSAccessKeyId() {
return“yourAccessKey”;
        }

public String getAWSSecretKey() {
return“yourSecretKey”;
        }
 }, cc);
```

3. 如何设置 https 的 endpoint

通过 https 访问 OOS 服务的代码示例如下：

```
System.setProperty("com.amazonaws.sdk.disableCertChecking",
"true");
oosclient.setEndpoint(“https://oos.ctyunapi.cn”);//设置通过 https 方式访问 oos 服务
```

4. 下载分段上传且未合并的对象

OOS Java SDK 中增加了对下载文件 Etag 的校验，如果下载文件的 Etag 校验不通过，会抛出异常。因此，如果用户采用分段上传方式上传文件，并且没有合并文件就直接下载，直接使用 OOS Java SDK 是会抛出异常的，需要设置下面的参数来屏蔽此校验功能。

```
System.setProperty("com.amazonaws.services.s3.disableGetObjectMD5Validation","true");
```

6.2 代码示例

OOSCredentials.properties

用于存储用户名和密码，例如：

```
accessKey =yourAccessKey
secretKey =yourSecretKey
```

OOSSample.java 文件

```
package com.amazonaws.services.s3;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Writer;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectSummary;

public class OOSSample {
public static void main(String[] args) throws IOException {
    /* 创建 client,其中 OOSCredentials.properties 中存放着用户名和密码*/
    AmazonS3 oos = new AmazonS3Client(new PropertiesCredentials(
              OOSSample.class
                    .getResourceAsStream("OOSCredentials.properties")));
    // 设置 endpoint 到 OOS
    oos.setEndpoint("http://oos.ctyunapi.cn");
    String bucketName = "my-first-oos-bucket";
    String key = "MyObjectKey";
    System.out.println("===========================================");
    System.out.println("Getting Started with OOS");
    System.out.println("===========================================");
    try {
        /* 创建 bucket */
        System.out.println("Creating bucket " + bucketName + "\n");
        oos.createBucket(bucketName);

        /*列出账户内的所有 buckets */
        System.out.println("Listing buckets");
        for (Bucket bucket : oos.listBuckets()) {
       System.out.println(" - " + bucket.getName());
                }
       System.out.println();

       /* 上传一个 object 到 bucket 中 */
       System.out.println("Uploading a new object to OOS from a file\n");
       oos.putObject(new PutObjectRequest(bucketName, key,
       createSampleFile()));

       /* 下载 object */
       System.out.println("Downloading an object");
       /* 当使用getObject()方法时，需要非常小心。因为返回的S3Object对象包括一个从
```

```
    HTTP连接获得的数据流。底层的HTTP连接不会被关闭，直到用户读完了数据，并关闭了流。因此从
    S3Object中读取inputStream数据后，需要立刻关闭流。否则会导致客户端连接池用满 */
        S3Object object = oos.getObject(new GetObjectRequest(bucketName,
key));
         System.out.println("Content-Type: "
         + object.getObjectMetadata().getContentType());
         System.out.println("Content:");
         displayTextInputStream(object.getObjectContent());

         /* 拷贝 object */
         String destinationBucketName = "my-copy-oos-bucket";
         String destinationKey = "MyCopyKey";
         System.out.println("Copying an object ,from " + bucketName + "/"
                  + key + " to " + destinationBucketName + "/"
                  + destinationKey);
         oos.createBucket(destinationBucketName);
         oos.copyObject(bucketName, key, destinationBucketName,
         destinationKey);

         /* 下载拷贝的 object */
         System.out.println("Downloading the " + destinationKey + " object");
         object = oos.getObject(new GetObjectRequest(destinationBucketName,
destinationKey));
         System.out.println("Content-Type: "
                  + object.getObjectMetadata().getContentType());
         System.out.println("Content:");
         displayTextInputStream(object.getObjectContent());

         /* 列出 bucket 中的 object，支持 prefix,delimiter,marker,max-keys 等选项 */
         System.out.println("Listing objects");
         ObjectListing objectListing = oos
                  .listObjects(new ListObjectsRequest().withBucketName(
bucketName).withPrefix("My"));
         for (S3ObjectSummary objectSummary : objectListing
                  .getObjectSummaries()) {
          System.out.println(" - " + objectSummary.getKey() + "  "
                               + "(size = " + objectSummary.getSize() + ")");
          }
          System.out.println();

          /* 删除 object */
          System.out.println("Deleting objects\n");
         oos.deleteObject(bucketName, key);
```

```
        oos.deleteObject(destinationBucketName, destinationKey);

         /* 删除 bucket */
        System.out.println("Deleting bucket " + bucketName + "\n");
        oos.deleteBucket(bucketName);
        System.out.println("Deleting bucket " + destinationBucketName
                + "\n");
        oos.deleteBucket(destinationBucketName);
        } 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());
       }
    }

Private static File createSampleFile() throws IOException {
       File file = File.createTempFile("oos-java-sdk-", ".txt");
file.deleteOnExit();
       Writer writer = new OutputStreamWriter(new FileOutputStream(file));
        writer.write("abcdefghijklmnopqrstuvwxyz\n");
        writer.write("012345678901123456789012345\n");
        writer.write("!@#$%^&*()-=[]{};':',.<>/?\n");
        writer.write("012345678901123456789012345\n");
        writer.write("abcdefghijklmnopqrstuvwxyz\n");
        writer.close();
        return file;
    }

Private static void displayTextInputStream(InputStream input)
throws IOException {
    BufferedReader reader = newBufferedReader(new InputStreamReader(input));
    while (true) {
          String line = reader.readLine();
```

```
    if (line == null)
    break;
    System.out.println("    " + line);
       }
        /* 需要在这里关闭InputStream，原因参见getObject处 */
    input.close();
    System.out.println();
    }
}
```

6.3 SDK 版本说明

目前 OOS 提供以下几个版本的 SDK：

1. oos-sdk-2.0.0.jar

   此版本的 SDK 兼容亚马逊 SDK 的 1.3.15 版本。

2. oos-sdk-2.1.0.jar

   此版本的 SDK 兼容亚马逊 SDK 的 1.4.7 版本。

   增加了 AccessKey 和 SecretKey 的相关接口，这些方法以 ctyun 为开头命名。

3. oos-sdk-5.0.0.jar

   此版本的 SDK 兼容亚马逊 SDK 的 1.4.7 版本。

   增加了异地互备相关的接口说明，可以实现不同资源池之间的数据互备，参见 PUT/GET/DELETE Bucket Trigger 等接口。同时用户可以记录异地互备相关的日志，参见 PUT/GET Bucket Trigger 接口。

   增加了断点续传接口。

4. oos-sdk-6.0.0.jar

   增加 bucket 索引位置、数据位置的相关接口，参见 PUT Bucket 接口。

7 使用 HttpURLConnection 开发

```
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;
}
```

```
conn.setRequestProperty("Authorization", authorization);
conn.setConnectTimeout(CONN_TIMEOUT);
conn.setReadTimeout(READ_TIMEOUT);
conn.setDoInput(true);
conn.setRequestMethod("GET");
conn.connect();
intcode = conn.getResponseCode();
System.out.println("code=" + code);
try (InputStream in = conn.getInputStream()) {
byte[] buffer = newbyte[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 = newURL("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);
    }
}
```

8 Endpoint 列表

OOS 对象存储网络中的各个地区，使用统一的 OOS API、管理 API、IAM Endpoint，各域名都支持 http 和 https。用户也可以指定使用某个资源池的 Endpoint，这样可以直接定位到该资源池。Endpoint 列表如下：

OOS API Endpoint: oos-cn.ctyunapi.cn

IAM Endpoint: oos-cn-iam.ctyunapi.cn

地区	OOS API Endpoint
郑州	oos-hazz.ctyunapi.cn
沈阳	oos-lnsy.ctyunapi.cn
四川成都	oos-sccd.ctyunapi.cn
乌鲁木齐	oos-xjwlmq.ctyunapi.cn
甘肃兰州	oos-gslz.ctyunapi.cn
山东青岛	oos-sdqd.ctyunapi.cn
贵州贵阳	oos-gzgy.ctyunapi.cn
湖北武汉	oos-hbwh.ctyunapi.cn
西藏拉萨	oos-xzls.ctyunapi.cn
安徽芜湖	oos-ahwh.ctyunapi.cn
广东深圳	oos-gdsz.ctyunapi.cn