对象存储

（Object-Oriented Storage, OOS）

开发者文档 V2.0

中国电信股份有限公司

云计算分公司

2013-11

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 提供了基于各种开发语言的 SDK 和基于 HTTP REST 的访问接口两种访问方式，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 Ruby、.NET、PHP、Java、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 oos.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）

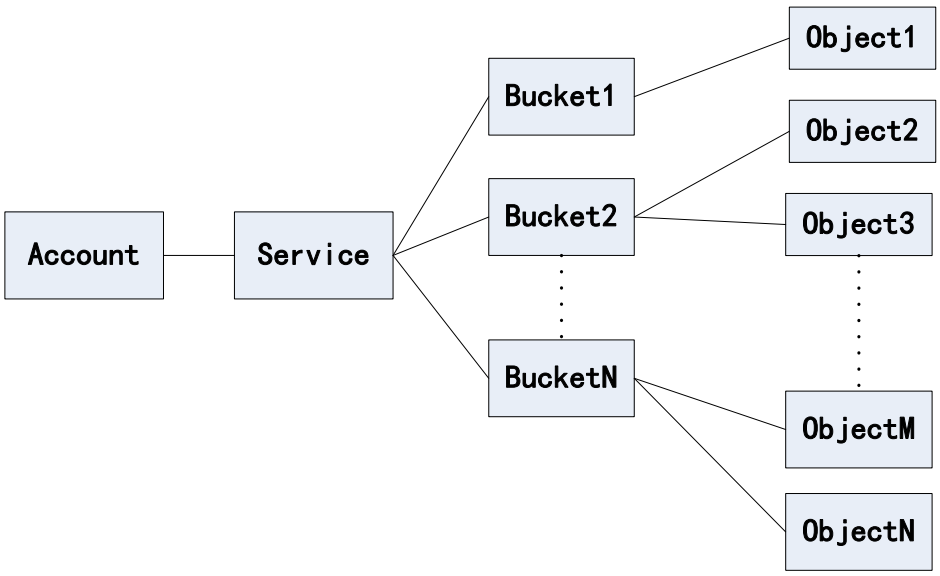


图 1　OOS 主要概念之间的关系

2.1 Account

在使用 OOS 之前，首先需要在 oos.ctyun.cn 注册一个账号（Account）。注册账号时必须填写邮箱、密码和联系方式，其他信息如公司名称、地址、联系人等可以选填。正确填写注册信息后需要激活账号。只有激活成功的账号才可以登录并使用 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

- Bucket名称必须全局唯一
- Bucket名称长度介于3到63字节之间
- Bucket名称可以由一个或者多个小节组成，小节之间用点（.）隔开，各个小节需要：
  - 只能包含小写字母、数字和短横线（-）
  - 必须以小写字母或者数字开始
  - 必须以小写字母或者数字结束
- Bucket名称不能是IP地址形式（如192.162.0.1）

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 的数量限制

每个用户最多可以建立 10 个 Bucket。

说明：Bucket 的数量及每个 Bucket 中存放的 Object 数量对面向对象存储的性能没有影响，10 万个 Object 存放在同一个 Bucket 中和分布在 100 个不同的 Bucket 中的性能是一样的。

应尽量避免频繁地创建、删除 Bucket 操作，推荐的方式是通过面向 Web 界面的控制台，或者只在应用程序初始化时进行创建，删除 Bucket 的操作。

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）支持的单个文件的大小从 1 字节到 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 oos-iam.ctyunapi.cn。

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（Authentication）

为了防止用户数据被他人盗取，所有发送到 OOS 的非匿名请求都需要经过签名验证。OOS 通过使用 Access Key ID/Secret Access Key 对称加密的方法来验证某个请求的发送者身份。Access Key ID 和 Secret Access Key 在 OOS 服务开通后自动随机生成。当用户想以个人身份向 OOS 发送请求时，需要首先将发送的请求按照 OOS 指定的格式生成签名字符串；然后使用 Secret Access Key 对签名字符串进行加密产生验证码。OOS 收到请求以后，会通过 Access Key ID 找到对应的 Secret Access Key，以同样的方法提取签名字符串和验证码，如果计算出来的验证码和提供的一样，即认为该请求时有效的；否则，OOS 将拒绝处理这次请求，并返回错误码。

3.1.1 Head 中包含签名

用户可以在 HTTP 请求中增加 Authorization（授权）的 Head 来包含签名信息，表明这个消息已被授权。如果用户的请求中没有 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;
```

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 AMZ 目标资源被称为 CanonicalizedResource。它的构建方法如下：

1) 将 CanonicalizedResource 置成空字符串（“”）；
2) 放入要访问的 AMZ 资源：“ /BucketName/ObjectName”（无 ObjectName 则不填）
3) 如果请求的资源包括子资源(sub-resource)，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 例子如：/BucketName/ObjectName?acl
4) 如果用户请求在查询字符串(query string)中指定了要重写(override)返回请求的 header，那么将这些查询字符串及其请求值按照字典序，从小到大排列，以’&’为分隔符，按参数的字典序添加到 CanonicalizedResource 中。此时的 CanonicalizedResource 例子：/BucketName/ObjectName?response-content-type=ContentType

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。对更多 Based64 编码信息，查看 Amazon S3 代码示例。

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。

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=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D
```

URL 中包含签名的示例如下：

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

请求参数

请求字符串参数名称	示例中的值	描述
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE	可以登录到对象存储的门户上，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 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（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。

3.3 Policy 安全策略

3.3.1 介绍

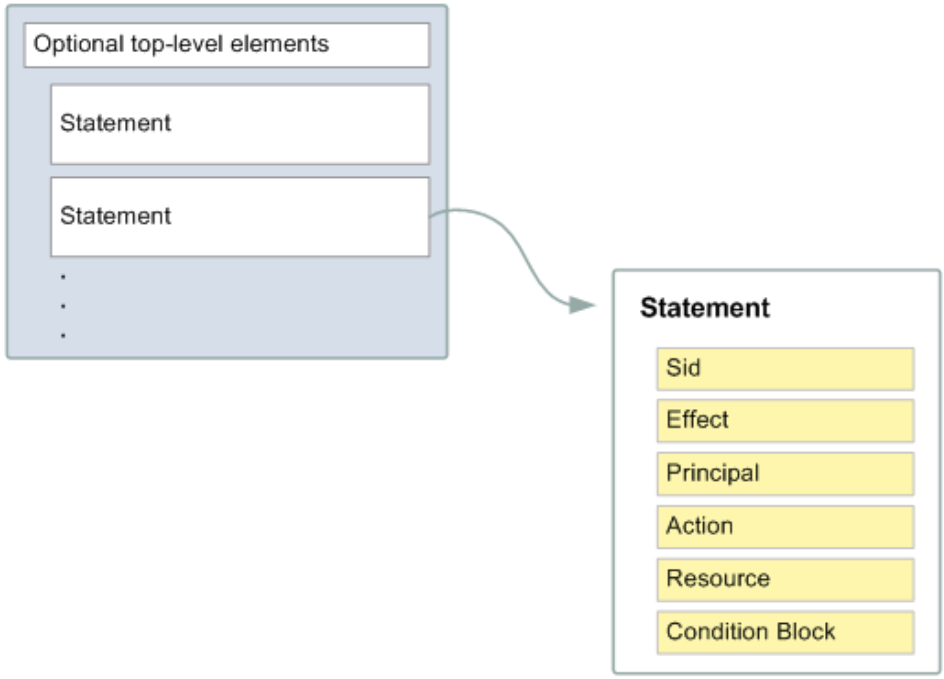
Bucket policy 用于定义 OOS 资源的访问权限。Policy 可以用于：

- ✓ 允许/拒绝 bucket 级别的权限
- ✓ 设置 bucket 中任何 object 的拒绝权限
- ✓ 为 bucket 中的 object 授权

Policy 本身是一个 JSON 字符串，一个 policy 包括：

- 可选的 policy 基本信息（在文档顶部）
- 一个或多个独立的 statements

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



3.3.2 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 2008-10-17。

```
"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，在版本中，Action 只能是"s3:*" 或 ["s3:*"]，表示所有 OOS 操作。

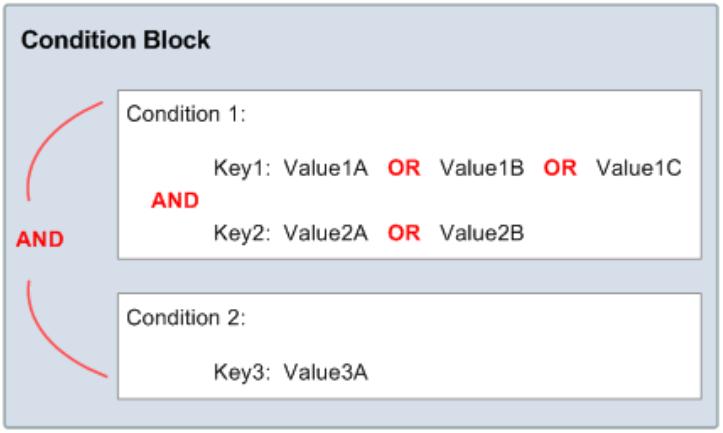
5. Resource

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） 支持的 Key

目前支持以下两个 Key。

aws:Referer — 同 HTTP referer 字段一样。

aws:SourceIp — 请求者的 IP 地址。

（2） Condition 类型

目前支持的类型有：String 和 IP address

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) 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

3.3.3 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"
            }
        }
    }
]
 }
```

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。

4.1 关于 Service 的操作

4.1.1 GetService(ListBucket)

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，用户需要使用有效的 Key ID 在 OOS 中注册，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>
```

4.2 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，OOS 中 bucket 的 name 长度为 63 个字符以内，只支持小写字母数字，点(.)，以及横杠（-）。

请求语法

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

请求参数

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

请求示例

请求创建一个名叫 picture 的 bucket：

```
PUT / HTTP/1.1
Host: picture.oos.ctyunapi.cn
Content-Length: 0
Date: Mon, 03 Sep 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: 236A8905248E5A01
Date: Mon, 03 Sep 2012 12:00:00 GMT
Location: /picture
Content-Length: 0
Connection: close
Server: CTYUN
```

创建 bucket 并设置 ACL

设置 bucket 名称是 picture 的 ACL 为 public。

```
PUT / HTTP/1.1
Host: picture.oos.ctyunapi.cn
Content-Length: 0
x-amz-acl: public
Date: Mon, 03 Sep 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: 236A8905248E5A01
Date: Mon, 03 Sep 2012 12:00:00 GMT
Location: /picture
Content-Length: 0
Connection: close
Server: CTYUN
```

4.2.2 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, 03 Sep 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 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>
```

4.2.3 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	使用 STANDARD 类型

请求示例

```
GET / HTTP/1.1
Host: BucketName.oos.ctyunapi.cn
Date: Mon, 03 Sep 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>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</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>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
      <DisplayName>yourmail@mail.com</DisplayName>
    </Owner>
  </Contents>
</ListBucketResult>
```

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>
```

4.2.4 Delete bucket

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

请求语法

```
DELETE / HTTP/1.1
Host: BucketName.oos.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, 03 Sep 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, 03 Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.2.5 Put Bucket Policy

在 PUT 操作的 url 中加上 policy，可以进行添加或修改 policy 的操作。如果 bucket 已经存在了 Policy，此操作会替换原有 Policy。只有 bucket 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, 03 Sep 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, 03 Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.2.6 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, 03 Sep 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, 03 Sep 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/*",
        }
 ]
}
```

4.2.7 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, 03 Sep 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, 03 Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.2.8 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 4**错误时，会跳转到用户指定的出错页面。

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	是

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

4.2.9 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, 03 Sep 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, 03 Sep 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>
```

4.2.10 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, 03 Sep 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, 03 Sep 2012 12:00:00 GMT
Connection: close
Server: CTYUN
```

4.2.11 List Multipart Uploads

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

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

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

请求语法

```
GET /?uploads HTTP/1.1
Host: BucketName.oos.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 prefix 来对 key 进行分组）。	否
upload-id-marker	与 key-marker 参数一起，指定列表操作从什么位置后面开始。如果 key-marker 没有被指定，upload-id-marker 参数也会被忽略。否则，任何 key 值等于或大于 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

请求头格式

无

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 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

出错响应

无

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</NextUp
loadIdMarker>
<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>
```

4.2.12 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 bucket 存储日志。可以设置任意一个你拥有的 bucket 作为 TargetBucket，包括启动日志的 bucket 本身。你也可以设置将多个 bucket 的日志存放到一个 TargetBucket 中，在这种情况下，你需要为每个源 bucket 设置不同的 TargetPrefix，以便不同 bucket 的 log 可以被区分出来。	否
TargetPrefix	生成的 log 文件将以此为前缀命名	否

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, 03 Sep 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/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03 Sep 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, 03 Sep 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/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03 Sep 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
Bucket Owner	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“-”

4.2.13 Get Bucket Logging

在 GET 操作的 url 中加上 logging，可以获得指定 bucket 的 logging。只有 bucket 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, 03 Sep 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/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03 Sep 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/1.1 200 OK
x-amz-request-id: 32FE2CEB32F5EE25
Date: Mon, 03 Sep 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/>
```

4.2.14 Head Bucket

此操作用于判断 bucket 是否存在，而且用户是否有权限访问。如果 bucket 存在，而且用户有权限访问时，此操作返回 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, 03 Sep 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

响应示例

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

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
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	否

	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。	

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, 03 Sep 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
```

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

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前未被修改的对象，无结果则返回 412 错误	否
If-Match	返回一个实体标签（Etag）和指定的那个一致的对象，无结果则返回 412 错误	否
If-None-Match	只返回与指定实体标签（etag）不同的对象，无结果则返回 304 错误	否

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, 03 Sep 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
```

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

```
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, 03 Sep 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

使用者可以通过一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
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

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

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

```
PUT /my-second-image.jpg HTTP/1.1
Host: bucket-sample.oos.ctyunapi.cn
Date: Mon, 03 Sep 2012 22:32:00 GMT
x-amz-copy-source: /bucket-sample/my-image.jpg
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
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?uploads HTTP/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被 Content-Type 头字段引用的 media-type 时采用何种解码方式。 类型：String 默认值：None	否
Content-Type	用来描述对象数据格式的标准 MIME 类型。 类型：String 默认值：binary/octel-stream 限制：仅 MIME 类型	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 默认值：STANDARD	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=
```

```
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>
```

返回示例

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	否
Expect	如果用户的应用中设置该头为 100-continue，则应用在接收到请求回应之前不会发送请求实体。如果基于头的消息被拒绝，消息的实体也不会被发送。 类型：String 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	404 Not 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	404 Not 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>
```

带指定错误头的返回示例

下面的响应表示在一个错误发生在 HTTP 响应头被发送前。

```
HTTP/1.1 403 Forbidden
x-amz-id-2: Uuag1LuByRx9e6j5Onimru9pO4ZVKnJ2Qz7/C1NPcfTWAtRPfTaOFg==
x-amz-request-id: 656c76696e6727732072657175657374
Date: Mon, 1 Nov 2010 20:34:56 GMT
Content-Length: 237
Connection: keep-alive
Server: CTYUN

<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>AccessDenied</Code>
<Message>Access Denied</Message>
<RequestId>656c76696e6727732072657175657374</RequestId>
<HostId>oos.ctyunapi.cn</HostId>
</Error>
```

带指定错误实体的返回示例

下面的响应表示在一个错误发生在 HTTP 响应头被发送之后。注意，当 HTTP 状态码为 200 ok 时，请求实际上出现了 Error 元素所描述的那种错误。

```
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"?>
<Error>
<Code>InternalError</Code>
<Message> Internal Error </Message>
<RequestId>656c76696e6727732072657175657374</RequestId>
<HostId>oos.ctyunapi.cn</HostId>
```

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
```

请求参数

无

请求头格式

无

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	404 Not Found

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

请求头格式

无

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） 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	404 Not 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过则执行拷贝操作，否则返回 412 错误	否

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: 996c76696e672773207265717565737 4
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 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 /?delete HTTP/1.1
Host: BucketName.oos.ctyunapi.cn
Content-Length: Size
Content-MD5: MD5
Authorization: Signature

<?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 父节点：无	是
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	是

	父节点：*Delete*	
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>
```

4.3.12 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。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
```

4.4 关于 AccessKey 的操作

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/?Action=CreateAccessKey HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

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

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

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

请求语法

```
POST/?Action=DeleteAccessKey&AccessKeyId=***** HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

请求参数

名称	描述
Action	值是 DeleteAccessKey
AccessKeyId	要删除的 AccessKey

请求示例

```
POST /?Action=DeleteAccessKey&AccessKeyId=2d1d5625f6202b08c8db HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

返回示例

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

4.4.3 UpdateAccessKey

更新普通的 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/?Action=UpdateAccessKey&AccessKeyId=***** &Status=active&IsPrimary=false HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

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 /?Action= UpdateAccessKey &AccessKeyId=2d1d5625f6202b08c8db&Status=active&IsPrimary=false
HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
```

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/?Action=ListAccessKey    HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: GMT Date
Authorization: SignatureValue
```

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 /?Action= ListAccessKey &MaxItem=100&Marker=2d1d5625f6202b08c8db HTTP/1.1
Host: oos-iam.ctyunapi.cn
Date: Wed, 01 Mar 2012 12:00:00 GMT
Authorization: AWS mailaddress@email.com:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

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 错误代码列表

错误码	描述	HTTP 状态码
NotVerify	用户未激活	403 Forbidden
InvalidArgument	无效的请求参数	400
BucketAlreadExists	请求的 bucket 的 name 已经不可用，请选择两一个不同的名字尝试。	409 Conflict
BucketNotEmpty	试图删除的的 bucket 非空	409 Conflict
InvalidAccessKeyId	你使用的 OOS ID 不存在	403 Forbidden
InvalidBucketName	指定的 bucket 无效	400 Bad Request
InvalidURI	无法解析指定的 URI	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
RequestTimeTooSkewed	请求的时间和服务器时间相差 15 分钟以上	403 Bad Request
NoSuchBucket	指定的 bucket 不存在	404 Not Found
NoSuchKey	指定对象的 Key 不存在	404 Not Found
SignatureDoesNotMatch	我们计算出的请求中的签名与用户提供的签名不一致。检查 OOS 账户安全	403 Forbidden
TooManyBuckets	你试图创建的 bucket 超出允许范围	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：PHP、Python、.NET、Ruby、Java。

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-1.0.0.jar 添加到编译路径中。

然后下载 Amazon S3 的 SDK，地址：http://aws.amazon.com/sdkforjava/，将 third-party 中的包添加到编译路径中。

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("AwsCredentials.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 服务器的地址。

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，云计算公司目前支持的 API 参见 SDK 包中的 com.amazonaws.services.s3.AmazonS3.java

使用别名支持网站

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”。

6.2 代码示例

OOSCredentials.properties

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

用于存储用户名和密码，例如：

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("012345678901123456789012345678901234\n");
            writer.write("!@#$%^&*()-=[]{};':',.<>/?\n");
            writer.write("012345678901123456789012345678901234\n");
            writer.write("abcdefghijklmnopqrstuvwxyz\n");
            writer.close();
            return file;
        }

        private static void displayTextInputStream(InputStream input)
                throws IOException {
            BufferedReader reader = new BufferedReader(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

OOS 提供的 REST 接口与 Amazon S3 兼容，以保障服务的稳定性。OOS 将一直坚持兼容 Amazon S3 协议的做法，不断地升级完善，满足客户更多的需求。开发者可以通过多种语言（Java，PHP, Python, Ruby, .NET）的 SDK 访问 OOS 服务。

OOS 官方提供了基于 Java 的 SDK。该 SDK 是基于 Amazon 官方 SDK 修改而成的。其唯一修改在于： 对目前暂不支持的 API 进行了标注（增加了 Javadoc 关键字 @Depreciated）， 以便帮助开发者轻松地知道哪些 API 是 OOS 暂不支

持的。开发者仅需将服务端点(End Point) 指向 OOS（http://oos.ctyunapi.cn/），即可轻松使用 OOS 服务。

这种“Depreciated Only”的做法使得开发者也可直接通过 Amazon 官方 SDK 使用 OOS 服务，只要不调用那些 OOS 暂不支持的 API 即可。由于带有标注的 SDK 更容易使用，因此推荐开发者使用 OOS 官方发布的 SDK。 该 SDK 与 Amazon 官方 SDK 并无本质差异，经过了严格的测试，具有非常好的正确性和稳定性，请放心使用。

目前 OOS 提供两个版本的 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。
   - ✓ 亚马逊 SDK 的 1.4.7 版本增加了对下载文件的 Etag 校验，如果下载的文件 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的话，直接使用 oos-sdk-2.1.0.jar 是会抛出异常的，需要设置 System.setProperty("com.amazonaws.services.s3.disableGetObjectMD5Validation","true"); 来屏蔽此校验功能。
   - ✓ Policy 的版本由 2008-10-17 变成了 2012-10-17，在设置 policy 的版本时需要注意。