# 对象存储

（Object-Oriented Storage,OOS）

# JavaScript SDK V1.0

**中国电信股份有限公司**

**云计算分公司**

目录

# 1 开始使用

## 1.1 要求

在使用 OOS 之前，首先需要在 oos.ctyun.cn 注册一个账号（Account）。

创建 AccessKeyId 和 AccessSecretKey。AccessKeyId 和 AccessSecretKey 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。

## 1.2 使用方式

```
<!—引入本地资源-->
<script src="./oos-sdk-x.x.x.min.js"></script>
```

使用 new OOS 创建 oos 对象

OOS JS-SDK 目前只支持异步请求方式，通过 callback 方式处理，对于 err 处理，非 err 显示处理结果。下图示例初始化到返回结果流程：

```
var client = new OOS.S3({...});
var params;
client.listObjects(params, function (err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data);           // successful response
});
```

## 1.3 CORS 配置

从浏览器中直接访问 OOS 需要开通 Bucket 的 CORS：

- 将来源（*）设置成：*
- 将允许的方法（*）设置成：PUT, GET, POST, DELETE, HEAD
- 将允许的 Headers 设置成：*
- 将暴露的 Headers：设置成 ETag

**注意：请将该条 CORS 规则设置成所有 CORS 规则的第一条。**


由于浏览器的同源策略，在浏览器里调用 JS-SDK 时，部分功能无法实现，包括 Service 的操作，Bucket 的新建，AccessKey 以及 SecretKey 的操作。

## 1.4 options 配置项

OOS options 介绍

| 参数 | 描述 | 是否必须 |
| --- | --- | --- |
| accessKeyId | String 通过天翼云控制台创建的 access key | 是 |
| secretAccessKey | String 通过天翼云控制台创建的 secret access key; | 是 |
| endPoint | String OOS 域名 | 是 |
| signatureVersion | String 计算签名版本，目前支持的版本为“v2” | 是 |
| apiVersion | String 使用的 API 版本 默认 ‘2006-03-01’ | 是 |
| s3ForcePathStyle | Boolean 是否强把请求的 url 格式化为 s3 协议所需的风格，目前遵循 s3 协议需要填写 true; | 是 |

示例：

```
var client = new OOS.S3({
    accessKeyId: accessKeyId,
    secretAccessKey: secretAccessKey,
    endpoint: endPoint,
    signatureVersion: 'v2',
    apiVersion: '2006-03-01',
    s3ForcePathStyle: true
});
```

# 2 SDK 列表

该部分主要介绍的内容是 OOS JS-SDK 对支持的所有功能，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。

由于浏览器同源策略，需要在自服务门户把 Bucket 的 CORS 支持配置完成才可用，配置方法见 1.3 CORS 配置。

一个完整的操作包括 params，callback，基本用法：

```
var params = {
    Param1: 'Value1', /* 参数*/
    ...
};
client.OPERATE(params, function(err, data) {
 if (err) console.log(err, err.stack); // 显示错误信息
 else     console.log(data);           // 成功，可在此处进行下一步操作
});
```

以下所有操作均可按照此模板，在 params 里面放入所需参数，调用 client 的方法，通过 callback 进行下一步操作。

## 2.1 关于 Bucket 操作

### 2.1.1 GET Bucket Acl(params = {}, callback)

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。

**请求方法**

```
client.getBucketAcl(params, function(err, data) {
    ...
});
```

**请求参数**

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回参数

| 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 |

## 2.1.2 GET Bucket (params = {}, callback)

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

### 请求方法

```
client.listObjects(params, function(err, data) {
    ...
});
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| 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 |
| MaxKeys | 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 |

## 2.1.3 DELETE Bucket (params = {}, callback)

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。

### 请求方法

```
client.deleteBucket(params, function(err, data) {
    ...
});
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

**返回元素**

成功无返回值

### 2.1.4 PUT Bucket Policy (params = {}, callback)

policy 的添加或修改的操作。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。

**请求方法**

```
client.putBucketPolicy(params, function(err, data) {
    ...
})
```

**请求参数**

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| Policy | Policy 规则的字符串 | 是 |

**返回元素**

成功没有返回值

### 2.1.5 GET Bucket Policy (params = {}, callback)

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

**请求方法**

```
client.getBucketPolicy(params, function(err, data) {
    ...
})
```

**请求参数**

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |

| | | |
|---|---|---|
| Bucket | 用户 bucket 名称 | 是 |

**返回元素**

| 名称 | 描述 |
|---|---|
| Policy | policy 的 String 格式 |

### 2.1.6 DELETE Bucket Policy (params = {}, callback)

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。

**请求方法**

```
client.deleteBucketPolicy(params, function(err, data) {
    ...
})
```

**请求参数**

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 用户 bucket 名称 | 是 |

**返回元素**

成功没有返回值

### 2.1.7 PUT Bucket WebSite(params = {}, callback)

配置 Bucket 的 website，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。

## 请求方法

```
client.putBucketWebsite(params, function(err, data) {
    ...
})
```

## 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | Bucket name | 是 |
| 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 |

## 参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    WebsiteConfiguration: {
        ErrorDocument:{
         Key: 'err.html'
        },
        IndexDocument:{
         Suffix: 'index.html'
        }
    }
}
```

### 返回元素

成功没有返回值

## 2.1.8 GET Bucket WebSite(params = {}, callback)

### 请求方法

```
client.getBucketWebsite(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回元素

```
<WebsiteConfiguration >
    <IndexDocument>
        <Suffix></Suffix>
    </IndexDocument>
    <ErrorDocument>
        <Key></Key>
    </ErrorDocument>
</WebsiteConfiguration>
```

## 2.1.9 DELETE Bucket WebSite(params = {}, callback)

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.

### 请求方法

```
client.deleteBucketWebsite(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回元素

成功没有返回值

## 2.1.10 List Multipart Uploads (params = {}, callback)

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。

## 请求方法

```
client.listMultiparyUploads(params, function(err, data) {
    ...
})
```

## 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| 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<br>类型：String | 否 |
| MaxUploads | 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。<br>类型：Integer<br>默认值：1000 | 否 |
| KeyMarker | 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。<br>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 |
| UploadIdMarker | 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<br>类型：容器<br>子节点：Bucket, KeyMarker, UploadIdMarker, NextKeyMarker,NextUploadIdMarker,Maxuploads,Delimiter ,Prefix,Commonfixes,IsTruncated<br>父节点：无 |
| Bucket | 分片上传对应的对象名称 |

| | |
|---|---|
| | 类型：String<br>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<br>类型：String<br>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<br>类型：String<br>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。<br>类型：String<br>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。<br>类型：String<br>父节点：ListMultipartUploadsResult |
| MaxUploads | 响应中包含的上传过程的最大数目<br>类型：String<br>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<br>类型：Boolean<br>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<br>类型: 容器<br>子节点: Key, UploadId, InitiatorOwner, StorageClass,Initiated<br>父节点: ListMultipartUploadsResult |
| Key | 分片上传过程起始位置对象的 Key 值<br>类型: *Integer*<br>父节点: *Upload* |
| UploadId | 分片上传过程的 ID 号<br>类型: *Integer*<br>父节点: *Upload* |
| Initiator | 指定执行此次分片上传过程的用户账号<br>子节点：ID，DisplayName<br>类型：容器<br>父节点：Upload |
| ID | OOS 账号的 ID 号<br>类型：String<br>父节点：Initiator，Owner |

| | |
|---|---|
| DisplayName | OOS 账号的账户名<br>类型：String<br>父节点：Initiator，Owner |
| Owner | 用来标识对象的拥有者<br>子节点：ID，DisplayName<br>类型：容器<br>父节点：Upload |
| StorageClass | 对象的存储类型（STANDARD 或 REDUCED_REDUDANCY 或 EC_N_M）<br>类型：String<br>父节点：Upload |
| Initiated | 分片上传过程启动的时间<br>类型：Date<br>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。<br>类型：String<br>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<br>类型：String<br>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。<br>类型：String<br>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。<br>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。<br>类型：String<br>父节点：CommonPrefixes |

### 2.1.11 PUT Bucket Logging(params = {}, callback)

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。

#### 请求方法

```
client.putBucketLogging(params, function(err, data) {
    ...
})
```

#### 请求参数

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

### 参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    BucketLoggingStatus: {
        LoggingEnabled: {
         TargetBucket: Target_Bucket_Name,
         TargetPrefix: Target_Prefix
    }
} }
```

### 返回元素

成功没有返回值

## 2.1.12 GET Bucket Logging (params = {}, callback)

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。

### 请求方法

```
client.getBucketLogging(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回元素

| 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 |

| | |
|---|---|
| TriggerTargetBucket | 在异地互备过程中，目标资源池的 log 会保存到此 bucket 中，oos 会向此 bucket 存储日志 |
| TriggerTargetPrefix | 在异地互备过程中，目标资源池的 log 文件将以此为前缀命名 |
| TriggerSourceBucket | 在异地互备过程中，源资源池的 log 会保存到此 bucket 中，oos 会向此 bucket 存储日志 |
| TriggerSourcePrefix | 在异地互备过程中，源资源池的 log 文件将以此为前缀命名 |

### 2.1.13 Head Bucket (params = {}, callback)

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。

#### 请求方法

```
client.headBuck (params, function(err, data) {
    ...
})
```

#### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 用户 bucket 名称 | 是 |

#### 返回元素

成功没有返回值

### 2.1.14 PUT Bucket Trigger (params = {}, callback)

进行添加 trigger 的操作，即添加一个向异地资源池同步的触发器。当客户端向本地资源池的 bucket 上传对象时，OOS 可以根据配置的策略，自动将对象同步到异地资源池中。一个 bucket 可以配置多个触发器，但只能有一个是默认的触发器。如果客户端要使用非默认的触发器上传对象，需要在 put object 时，加上参数 Trigger，值是指定的

TriggerName，目前部分资源池支持此功能，使用前请与技术支持确认。只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

### 请求方法

```
client.putBucketTrigger(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | Bucket 名称 | 是 |
| TriggerName | Trigger 的名称，是字母或数字，不能包含特殊符号，最多 20 个字符。 | 是 |
| IsDefault | 是否是默认的 trigger | 是 |
| RemoteSite | 异地资源池的相关配置，可以配置向多个异地资源池同步数据，每个异地资源池对应一个 RemoteSite。如果不配置 RemoteSite，即不复制到其他资源池。 | 否 |
| RemontEndPoint | 异地资源池的 endpoint | 是 |
| ReplicaMode | 同步到异地资源池的副本模式，如果不填写，会使用动态副本模式 | 否 |
| RemoteBucketName | 异地资源池的 bucketName | 是 |
| RemoteAK | 异地资源池的 AccessKey | 是 |
| RemoteSK | 异地资源池的 secretKey | 是 |

### 参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    TriggerConfiguration: {
        Trigger: {
         TriggerName: Trigger_Name,
         IsDefault: false,
         RemoteSite: {
             RemontEndPoint: Remont_End_Point,
             RemoteBucketName: Remote_Bucket_Name,
             RemoteAK: Remote_AK,
             RemoteSK: Remote_SK
         }
        }
}
```

### 返回元素

成功没有返回值

## 2.1.15 GET BucketTrigger (params = {}, callback)

查询某 bucket 中配置的所有 trigger。只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

### 请求方法

```
client.getBucketTrigger(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回元素

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| TriggerName | Trigger 的名称 | 是 |

| | | |
|---|---|---|
| IsDefault | 是否是默认的 trigger | 是 |
| RemoteSite | 异地资源池的相关配置 | 否 |
| RemontEndPoint | 异地资源池的 endpoint | 是 |
| ReplicaMode | 同步到异地资源池的副本模式 | 否 |
| RemoteBucketName | 异地资源池的 bucketName | 是 |
| RemoteAK | 异地资源池的 AccessKey | 是 |
| RemoteSK | 异地资源池的 SecretKey | 是 |

### 2.1.16 DELETE Bucket Trigger (params = {}, callback)

删除某 bucket 中配置的所有 trigger，只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

#### 请求方法

```
client.deleteBucketTrigger(params, function(err, data) {
    ...
})
```

#### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 用户 bucket 名称 | 是 |

#### 返回元素

成功没有返回值

### 2.1.17 PUT Bucket Lifecycle (params = {}, callback)

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。

## 请求方法

```
client.putBucketLifecycle(params, function(err, data) {
    ...
})
```

## 请求参数

| 名称 | 描述 | 是否必需 |
| --- | --- | --- |
| Bucket | Bucket 名称 | 是 |
| LifecycleConfiguration | 容器，最多包含 100 个规则<br>类型：容器<br>子节点：Rule<br>父节点：无 | 是 |
| Rule | 生命周期规则的容器<br>类型：object 类型 Array<br>父节点：<br>LifecycleConfiguration | 是 |
| ID | 规则的唯一标识，最长 255 个字符。<br>类型：字符串<br>父节点：Rule | 否 |
| Filter | 配置过滤器<br>类型：object<br>父节点：Rule | 是 |
| Prefix | 指明要使用规则的对象前缀，最长 1024 个字符<br>类型：字符串<br>父节点：Filter | 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。<br>类型：字符串<br>父节点：Rule | 是 |
| Expiration | 描述过期动作的容器<br>类型：容器<br>子节点：Days<br>父节点：Rule | 是 |
| Days | 以天数来描述生命周期，值是正整数<br>类型：整数<br>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<br>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<br>类型：String<br>父节点：Expiration | Days 和 Date 二选一 |

### 参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    LifecycleConfiguration: {
        Rules: [{
         Expiration: {
            Days: 3650
         },
         Filter: {
            Prefix: 'documents/'
         },
         ID: 'TestOnly',
         Status: 'Enabled'
    }]
}}
```

### 返回元素

成功没有返回值

## 2.1.18 GET Bucket Lifecycle (params = {}, callback)

### 请求方法

```
client.getBucketLifecycle(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 用户 bucket 名称 | 是 |

**返回元素**

| 名称 | 描述 |
| --- | --- |
| LifecycleConfiguration | 容器，最多包含 100 个规则<br>类型：容器<br>子节点：Rule<br>父节点：无 |
| Rule | 生命周期规则的容器<br>类型：容器<br>父节点：LifecycleConfiguration |
| ID | 规则的唯一标示，最长 255 个字符。<br>类型：字符串<br>父节点：Rule |
| Prefix | 指明要使用规则的对象前缀<br>类型：字符串<br>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。<br>类型：字符串<br>父节点：Rule |
| Expiration | 描述过期动作的容器<br>类型：容器<br>子节点：Days<br>父节点：Rule |
| Days | 以天数来描述生命周期，值是正整数<br>类型：整数<br>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<br>日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。<br>例如：2002-10-11T00:00:00.000Z<br>类型：String<br>父节点：Expiration |

### 2.1.19 DELETE Bucket Lifecycle(params = {}, callback)

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

### 请求方法

```
client.deleteBucketLifecycle(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |

### 返回元素

成功没有返回值

## 2.2 关于 Object 的操作

### 2.2.1 PUT Object(params = {}, callback)

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。

### 请求方法

```
client.putObject(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| Body | File 内容 | 是 |
| Key | 文件名 | 是 |
| Trigger | 冗余的地址（使用此功能请与技术支持人员确认） | 否 |
| CacheControl | 按照请求/回应的方式用来定义缓存行为 | 否 |
| ContentDisposition | 指出对象的描述性的信息 | 否 |
| ContentEncoding | 指出对象所使用的编码格式 | 否 |

| | | |
|---|---|---|
| ContentMD5 | 按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值 | 否 |
| ContentType | 标准的 MIME 类型用来描述内容格式。 | 否 |
| Expires | new Date，对象不再被缓存的时间<br>类型：String | 否 |
| Metadata | 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 |
| StorageClass | 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。<br>类型: String<br>默认值: STANDARD<br>可选值: STANDARD \| REDUCED_REDUNDANCY\|EC_N_M<br>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。 | 否 |

## 参数格式

```
var params = {
    Bucket: 'STRING_VALUE', /* required */
    Key: 'STRING_VALUE', /* required */
    Body: new Buffer('...') || 'STRING_VALUE' || streamObject,
    CacheControl: 'STRING_VALUE',
    ContentDisposition: 'STRING_VALUE',
    ContentEncoding: 'STRING_VALUE',
    ContentLanguage: 'STRING_VALUE',
    ContentLength: 0,
    ContentMD5: 'STRING_VALUE',
    ContentType: 'STRING_VALUE',
    Expires: new Date,
    Metadata: {
        <MetadataKey>: 'STRING_VALUE',
        /* '<MetadataKey>': ... */
    },
    StorageClass: STANDARD | REDUCED_REDUNDANCY|EC_N_M,
    Trigger: 'STRING_VALUE'
};
```

### 返回元素

成功没有返回值

## 2.2.2 GET Object (params = {}, callback)

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。注：此方法并不能下载，只能获取到 Object 的 Uint8Array 编码字节流，若想下载，可通过 getSignedUrl 方法 return 出可下载的 url。

### 请求方法

```
client.getObject(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| Key | 文件名 | 是 |

### 返回元素

| 名称 | 描述 |
| --- | --- |
| Body | OOS 中的对象信息 |
| ContentType | 对象类型 |
| LastModified | 最后更改时间 |
| ContentLength | 对象大小 |
| Metadata | 对象的元数据 |

## 2.2.3 DELETE Object(params = {}, callback)

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。

### 请求方法

```
client.deleteObject(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| Key | 文件名 | 是 |

### 返回元素

成功没有返回值

## 2.2.4 PUT Object - Copy (params = {}, callback)

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。增加参数 CopySource，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。

CopySource 数据需要做 encodeURIComponent()处理。

### 请求方法

```
client.copyObject(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 目标 bucket 名称 | 是 |
| CopySource | 复制的源 Bucket/key | 是 |
| Key | 目标文件名 | 是 |
| CacheControl | 按照请求/回应的方式用来定义缓存行为 | 否 |
| ContentDisposition | 指出对象的描述性的信息 | 否 |
| ContentEncoding | 指出对象所使用的编码格式 | 否 |

| | | |
|---|---|---|
| ContentMD5 | 按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值 | 否 |
| ContentType | 标准的 MIME 类型用来描述内容格式。 | 否 |
| Expires | new Date，对象不再被缓存的时间<br>类型：String | 否 |
| Metadata | 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 |
| StorageClass | 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。<br>类型: String<br>默认值: STANDARD<br>可选值: STANDARD \| REDUCED_REDUNDANCY\|EC_N_M<br>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。 | 否 |

**返回元素**

成功没有返回值

### 2.2.5 Initial Multipart Upload (params = {}, callback)

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。

**请求方法**

```
client.createMultipartUpload(params, function(err, data) {
    ...
})
```

## 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 用户 bucket 名称 | 是 |
| Key | 文件名 | 是 |
| CacheControl | 按照请求/回应的方式用来定义缓存行为 | 否 |
| ContentDisposition | 指出对象的描述性的信息 | 否 |
| ContentEncoding | 指出对象所使用的编码格式 | 否 |
| ContentType | 标准的 MIME 类型用来描述内容格式。 | 否 |
| Expires | new Date，对象不再被缓存的时间<br>类型：String | 否 |
| Metadata | 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 |
| StorageClass | 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。<br>类型: String<br>默认值: STANDARD<br>可选值: STANDARD \| REDUCED_REDUNDANCY\|EC_N_M<br>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。 | 否 |

## 返回元素

| 名称 | 描述 |
| --- | --- |
| InitiateMultipartUploadResult | 包含所有返回元素的容器<br>类型：容器<br>子节点：Bucket，Key，UploadId<br>父节点：无 |
| Bucket | 分片上传对应的 Bucket 的名称<br>类型：Sring<br>父节点：InitiateMultipartUploadResult |
| Key | 分片上传对应的对象名称<br>类型：String<br>父节点：InitiateMultipartUploadResult |
| UploadId | 分片上传 ID<br>类型：String<br>父节点：InitiateMultipartUploadResult |

### 2.2.6 Upload Part (params = {}, callback)

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。

#### 请求方法

```
client.uploadPart(params, function(err, data) {
    ...
})
```

#### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 分片所属的 bucket 名称 | 是 |
| Key | 分片的名称 | 是 |
| PartNumber | 分片的 id | 是 |
| UploadId | 初始化分片后返回的 id | 是 |

| | | |
|---|---|---|
| Body | File 数据 | 否 |
| *ContentLength* | 该分片的大小，以字节为单位。<br>类型：Integer<br>默认值：None | 否 |
| *ContentMD5* | 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。<br>类型：String<br>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。<br>类型：String<br>默认值：None<br>可选值：100-continue | 否 |

### 返回元素

| | |
|---|---|
| ETag | 上传文件返回的实体标签<br>类型：String |

## 2.2.7 Complete Multipart Upload (params = {}, callback)

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。

### 请求方法

```
client.completeMultipartUpload(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|

| | | |
|---|---|---|
| Bucket | 分片 object 所属 bucket 名称 | 是 |
| Key | 分片的 object 名称 | 是 |
| UploadId | 初始化时返回的 uploadID | 是 |
| MultipartUpload | 请求的容器。<br>类型：object<br>子节点：1 个或多个 Part 元素 | 是 |
| Parts | 描述 part 的信息<br>父容器：MultipartUpload<br>类型 object 类型数组<br>包括：ETag: 上传每个分片时返回的 ETag（实体标签）<br>PartNumber: 上传的每个分片对应的编号 | 是 |

**返回元素**

| 名称 | 描述 |
|---|---|
| CompleteMultipartUploadResult | 包含整个响应的容器<br>类型：容器<br>子节点：*Location*, *Bucket*, *Key*, *ETag*<br>父节点：无 |
| Location | 新创建的对象的 URL 地址<br>类型：URI<br>父节点：*CompleteMultipartUploadResult* |
| Bucket | 分片上传对应的对象容器<br>类型：String<br>父节点：*CompleteMultipartUploadResult* |
| Key | 新创建的对象的 Key<br>类型：String<br>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。<br>类型：String<br>父节点：*CompleteMultipartUploadResult* |

### 2.2.8 Abort Multipart Upload (params = {}, callback)

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。

### 请求方法

```
client.abortMultipartUpload(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 分片 object 所属 bucket 名称 | 是 |
| Key | 分片的 object 名称 | 是 |
| UploadId | 初始化时返回的 uploadID | 是 |

### 返回元素

成功没有返回值

## 2.2.9 List Parts (params = {}, callback)

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。

### 请求方法

```
client.listParts(params, function(err, data) {
    ...
})
```

### 请求参数

| 名称 | 描述 | 是否必须 |
| --- | --- | --- |
| Bucket | 分片 object 所属 bucket 名称 | 是 |
| Key | 分片的 object 名称 | 是 |
| UploadId | 初始化时返回的 uploadID | 是 |
| MaxParts | Integer 设置返回的最大的分片数量 | 否 |
| PartNumberMarker | Intege 设定此值，只会返回分片号大于此值得分片 | 否 |

### 返回元素

| 名称 | 描述 |
| --- | --- |
| ListPartsResult | 包含整个响应的容器<br>类型：容器 |

| | |
|---|---|
| | 子节点：Bucket, Key, UploadId, Initiator, Owner, StorageClass,PartNumberMarker, NextPartNumberMarker, MaxParts,IsTruncated, Part<br>父节点：无 |
| Bucket | 分片上传对应的对象名称<br>类型：String<br>父节点：ListPartsResult |
| Key | 新创建的对象的 Key<br>类型：String<br>父节点：ListPartsResult |
| UploadId | 分片上传 ID<br>类型：String<br>父节点：ListPartsResult |
| Initiator | 指定执行此次分片上传过程的用户账号<br>子节点：ID，DisplayName<br>类型：容器<br>父节点：ListPartsResult |
| ID | OOS 账号的 ID 号<br>类型：String<br>父节点：Initiator |
| DisplayName | OOS 账号的账户名<br>类型：String<br>父节点：Initiator |
| Owner | 用来标识对象的拥有者<br>子节点：ID，DisplayName<br>类型：容器<br>父节点：ListPartsResult |
| StorageClass | 对象的存储类型（STANDARD 或 REDUCED_REDUNDANCY 或 EC_N_M）<br>类型：String<br>父节点：ListPartsResult |
| PartNumberMarker | 列表起始位置的片段的分片号<br>类型：Integer<br>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。<br>类型：Integer<br>父节点：ListPartsResult |
| MaxParts | 响应中片段的最大数目<br>类型：Integer<br>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<br>类型：Boolean<br>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<br>子节点：*PartNumber*, *LastModified*, *ETag*, *Size*<br>*类型：String*<br>*父节点：ListPartsResult* |
| PartNumber | 标识片段的分片号<br>类型：Integer<br>父节点：Part |
| LastModified | 片段上传完成的日期<br>类型：Date<br>父节点：Part |
| ETag | 片段上传完成时返回的 ETag 值<br>类型：String<br>父节点：Part |
| Size | 片段的数据大小<br>类型：Integer<br>父节点：Part |

.

### 2.2.10 Copy Part (params = {}, callback)

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。需要通过参数 CopySource 来定义拷贝源。如果想拷贝源 object 中的一部分，可以加参数 CopySourceRange，CopySource 数据需要做 encodeURIComponent()处理。

#### 请求方法

```
client.uploadPartCopy(params, function(err, data) {
    ...
})
```

#### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 分片 object 所属 bucket 名称 | 是 |
| Key | 分片的 object 名称 | 是 |

| | | |
|---|---|---|
| PartNumber | 分片的编号 | 是 |
| UploadId | 初始化时返回的 uploadID | 是 |
| CopySource | 已经存在的 object 地址 | 是 |

### 返回元素

| | |
|---|---|
| CopyPartResult | 包含整个响应的容器<br>类型：容器<br>父节点：无 |
| ETag | 新分片的 ETag<br>类型：String<br>父节点：*CopyPartResult* |
| LastModified | 分片的最后修改时间<br>类型：String<br>父节点：*CopyPartResult* |

## 2.2.11 DELETE Multiple Objects(params = {}, callback)

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。

### 请求方法

```
client.deleteObjects(params, function(err, data) {
    ...
})
```

### 请求参数

请求元素

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 需要删除 objects 的 bucket 名称 | 是 |
| Delete | 包含整个请求的容器<br>类型：容器<br>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。<br>类型：Boolean<br>父节点：*Delete* | 否 |
| Object | 包含被删除 object 的容器 | 是 |

| | | |
|---|---|---|
| | 类型：容器<br>父节点：*Delete* | |
| Key | 被删除 object 名<br>类型：String<br>父节点：*Object* | 是 |

## 参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    Delete: {
        Objects: [
         {
            Key: Key_1
         },
         {
            Key: Key_2
         },
         {
            Key: Key_3
         }],
        Quiet: true
    }
}
```

## 返回元素

| 名称 | 描述 |
|---|---|
| DeleteResult | 包含整个响应的容器<br>类型：容器<br>父节点：无 |
| Deleted | 成功删除的容器，包含成功删除的 object<br>类型：容器<br>父节点：*DeleteResult* |
| Key | 尝试删除的 object 名<br>类型：String<br>父节点：Deleted，或 Error |
| Error | 删除失败的容器，包含删除失败的 object 信息<br>类型：容器<br>父节点：*DeleteResult* |
| Code | 删除失败的状态码<br>类型：String<br>父节点：Error |

| | |
|---|---|
| | 值：AccessDenied, InternalError |
| Message | 错误的描述<br>类型：String<br>父节点：Error |

### 2.2.12 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。

#### 请求方法

```
var url = client.getSignedUrl('getObject', params);
return url;
```

#### 请求参数

| 名称 | 描述 | 是否必须 |
|---|---|---|
| Bucket | 要分享的 bucket 名称 | 是 |
| Key | 要分享的 object 名称 | 是 |
| Expires | 过期时间，单位 秒 ，默认 900 秒 | 否 |

#### 返回元素

返回一个非拥有者也可以下载的 url。