天翼云

对象存储

(Object-Oriented Storage,OOS)

IOS SDK 开发者指南 V6

天翼云科技有限公司

目录

1. 前言

对象存储（Object-Oriented Storage，OOS）为客户提供一种海量、弹性、廉价、高可用的存储服务。OOS 提供了基于 Web 门户和基于 HTTP REST 接口两种访问方式，用户可以在任何地方通过互联网对数据进行管理和访问，也可以通过 OOS 提供的 SDK 来调用 OOS 服务。

2.使用条件

2.1.使用前需具备的知识

- 已经熟悉 OOS 的基本概念，如 Bucket、Object、AccessKey 等。具体介绍可参见《OOS 开发者文档-v6》。
- 了解 iOS 的开发基础知识。

2.2.获取 OOS 访问凭证

AccessKeyId 和 SecretKey 是用户访问 OOS 的密钥（即 OOS 访问凭证），OOS 会通过它来验证用户的资源请求，请务必妥善保管。关于 AccessKeyId 和 SecretKey 的介绍详见《OOS 开发者文档-v6》。

2.3.基础条件

用户需要具备以下条件，然后才能够使用 OOS IOS SDK：

- ➢ 一个 OOS 账户；
- ➢ 获取 OOS 访问凭证；
- ➢ 获取 OOS IOS SDK；
- ➢ macos 10.15.5，xcode 11.5，cocoapod 1.9.3。

2.4.安装 SDK

1. 安装 CocoaPod（版本 1.9.3），具体安装方法参考官网。
2. 解压 **OOS IOS SDK** 文件，进入 samples/CocoaPod 文件夹。
3. 执行 pod install。
4. 打开 CocoaPod.xcworkspace。
5. 编译并运行 CocoaPod。

2.5.使用 SDK 前设置

无

3.OOS 服务代码示例

samples 文件下为示例工程，其中 Demo 为 Objective-C 语言示例工程，SwiftDemo 为使用 Swift 语言调用 Objective-C 示例工程，CocoaPod 为使用 CocoaPod 工具的工程文件。

3.1.关于 Service 的操作

3.1.1.GET Service(List Bucket)

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

注意：只有验证用户可以执行该操作，匿名用户不能执行该操作。

示例代码

```
OOS *client = [OOS defaultOOS];

// 默认不需要设置任何参数
[[client listBuckets:[OOSRequest new]] continueWithBlock:^id(OOSTask
*task) {
    // 结果在 OOSListBucketsOutput 中 buckets 属性中
    OSListBucketsOutput *listBucketOutput = task.result;
    NSLog(@" testListBucket ========= responseObject is:
================  %@", [listBucketOutput description]);
    return nil;
}];
```

3.1.2.GET Regions

获取资源池中的索引位置和数据位置列表。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetRegionsRequest *req = [OOSGetRegionsRequest new];
[[client getRegions:req] continueWithBlock:^id(OOSTask *task) {
    // 在 output.bucketRegions.dataRegions.regions 中查看结果
    OOSGetRegionsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.关于 Bucket 的操作

3.2.1.PUT Bucket

Put Bucket 用来创建一个 Bucket。只有根用户和拥有相应权限的子用户才能执行此操作。

Bucket 的命名方式如下：

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

示例代码

```
OOS *client = [OOS defaultOOS];

// 设置 Bucket 的索引位置
OOSMetaDataLocationConstraint *metadataLocation =
[OOSMetaDataLocationConstraint new];
metadataLocation.location = @"ShenYang";

// 设置 Bucket 的数据位置，是个数组
OOSDataLocationList *locationList = [OOSDataLocationList new];
locationList.locations = @[ @"ZhengZhou" ];

// 如果 type 设定为 Specified，需要设置 locationList
OOSDataLocationConstraint *dataLocation = [OOSDataLocationConstraint
new];
```

```
dataLocation.type = @"Specified";
dataLocation.scheduleStrategy = @"Allowed";
dataLocation.locationList = locationList;

// 打包设置
OOSCreateBucketConfiguration *config = [OOSCreateBucketConfiguration
new];
config.metadataLocationConstraint = metadataLocation;
config.dataLocationConstraint = dataLocation;

OOSCreateBucketRequest *createBucketReq = [OOSCreateBucketRequest
new];
createBucketReq.bucket = @"bucket name";
// 设置 Bucket 的权限
createBucketReq.ACL = OOSBucketCannedACLPublicReadWrite;
createBucketReq.createBucketConfiguration = config;

[[client createBucket:createBucketReq] continueWithBlock:^id(OOSTask
*task) {
    // 只有一个 location 的返回值
    OOSCreateBucketOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.2.GET Bucket location

该操作用来获取 Bucket 的索引位置和数据位置，只有具有该权限的用户才能执行此操作。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetBucketLocationRequest *getReq = [OOSGetBucketLocationRequest
new];
getReq.bucket = @"bucket name";

[[client getBucketLocation:getReq] continueWithBlock:^id(OOSTask *task)
{
    // 返回值中的内容格式和创建时的内容格式一致
    OOSGetBucketLocationOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.3.GET Bucket ACL

此操作用来获取 Bucket ACL 信息，只有拥有 GET Bucket ACL 权限的用户才可以执行此操作

示例代码

```
OOSGetBucketAclRequest *getBucketAclReq = [OOSGetBucketAclRequest new];
getBucketAclReq.bucket = @"bucket name";

OOS *client = [OOS defaultOOS];
[[client getBucketAcl:getBucketAclReq] continueWithBlock:^id(OOSTask
*task) {
    // grants 是 OOSGrant 数组
    // owner 保存着 Bucket 拥有者的信息
    OOSGetBucketAclOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.4.Get Bucket(List Objects)

此操作用来返回 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。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSListObjectsRequest *listObjectsReq = [OOSListObjectsRequest new];
listObjectsReq.bucket = @"bucket name";
// 返回的 object 名称前都需要有 prefix 值
listObjectsReq.prefix = @"prefix";
// 一次最多返回的 object 数量
listObjectsReq.maxKeys = [NSNumber numberWithInt:100];

[[client listObjects:listObjectsReq] continueWithBlock:^id(OOSTask
*task) {
    // contents 中保存 OOSObject 数组
    OOSListObjectsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
        return nil;
}];
```

3.2.5.DELETE Bucket

此操作用来删除 Bucket，但要求被删除 Bucket 中无 Object，即该 Bucket 中的所有 Object 都已被删除。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSDeleteBucketRequest *deleteBucketReq = [OOSDeleteBucketRequest new];
deleteBucketReq.bucket = @"bucket name";

[[client deleteBucket:deleteBucketReq] continueWithBlock:^id(OOSTask
*task) {
    // 删除失败时 task.error 不为空
    return nil;
}];
```

3.2.6.PUT Bucket Policy

此操作用来在 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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 包含 Policy 语句的 JSON 串
NSDictionary *dic = @{
                    @"Statement": @[@{
                                        @"Effect": @"Allow",
                                        @"Principal": @{
                                                @"AWS": @[@"*"]
                                                },
                                        @"Action": @"s3:*",
                                        @"Resource": @"arn:aws:s3:::*/*"
                                        }]
                    };

OOSPutBucketPolicyRequest *putReq = [OOSPutBucketPolicyRequest new];
putReq.bucket = @"bucket name";
putReq.policy = dic;

[[client putBucketPolicy:putReq] continueWithBlock:^id(OOSTask *task)
{
    // 操作失败时 task.error 不为空
    return nil;
}];
```

3.2.7.GET Bucket Policy

在 GET 操作的 url 中加上 Policy，获得指定 Bucket 的 Policy。如果 Bucket 没有 Policy，返回 404，NoSuchPolicy 错误

示例代码

```
OOS *client = [OOS defaultOOS];


OOSGetBucketPolicyRequest *getReq = [OOSGetBucketPolicyRequest new];
getReq.bucket = @"bucket name";


[[client getBucketPolicy:getReq] continueWithBlock:^id(OOSTask *task) {
    // policy 是一个 JSON 字符串
    OOSGetBucketPolicyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.8.DELETE Bucket Policy

在 DELETE 操作的 url 中加上 Policy，可以删除指定 Bucket 的 Policy。

- 如果 Bucket 配置了 Policy，删除成功，返回 200 0K。
- 如果 Bucket 没有配置 Policy，返回 204 NoContent。。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSDeleteBucketPolicyRequest *deleteReq = [OOSDeleteBucketPolicyRequest
new];
deleteReq.bucket = @"bucket name";

[[client deleteBucketPolicy:deleteReq] continueWithBlock:^id(OOSTask
*task) {
    // 操作失败时 task.error 不为空
    return nil;
}];
```

3.2.9.PUT Bucket Website

在 PUT Bucket 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。

WebSite 功能可以让用户将静态网站存放到 OOS 上。对于已经设置了 WebSite 的 Bucket，当用户访问 http://bucketName.oos-website-cn.oos.ctyunapi.cn 时，会跳转到用户指定的主页，当出现 4XX 错误时，会跳转到用户指定的出错页面。如果想通过自有域名的形式（例如 http://yourdomain.com/login.html）而非通过第三方域名的形式（例如 http://yourdomain.com.oos.ctyunapi.cn/login.html）访问，可以创建一个名为“yourdomain.com”的 bucket，并在域名管理系统中将“yourdomain.com”增加一个别名记录“oos.ctyunapi.cn”。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSWebsiteConfiguration *config = [OOSWebsiteConfiguration new];

// 设置出错页面
OOSErrorDocument *errorDocument = [OOSErrorDocument new];
errorDocument.key = @"error.html";
config.errorDocument = errorDocument;

// 设置首页
OOSIndexDocument *indexDocument = [OOSIndexDocument new];
indexDocument.suffix = @"index.html";
config.indexDocument = indexDocument;

// 配置静态页面
OOSPutBucketWebsiteRequest *putReq = [OOSPutBucketWebsiteRequest new];
putReq.bucket = @"bucket name";
putReq.websiteConfiguration = config;

[[client putBucketWebsite:putReq] continueWithBlock:^id(OOSTask *task)
{
    return nil;
```

```
}];
```

3.2.10.GET Bucket Website

在 GET Bucket 的 url 中加上 website，获得指定 Bucket 的 website。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetBucketWebsiteRequest *getReq = [OOSGetBucketWebsiteRequest new];
getReq.bucket = @"bucket name";

[[client getBucketWebsite:getReq] continueWithBlock:^id(OOSTask *task)
{
    // 返回值只有 errorDocument 和 indexDocument
    OOSGetBucketWebsiteOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.11.DELETE Bucket Website

在 Delete Bucket 操作的 url 中加上 website，可以删除指定 Bucket 的 website。

如果 Bucket 没有 website，返回 200 OK。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSDeleteBucketWebsiteRequest *deleteReq =
[OOSDeleteBucketWebsiteRequest new];
deleteReq.bucket = @"bucket name";

[[client deleteBucketWebsite:deleteReq] continueWithBlock:^id(OOSTask
*task) {
    // 操作失败时 task.error 不为空
    return nil;
}];
```

3.2.12.List Multipart Uploads

该操作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。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSListMultipartUploadsRequest *listReq = [OOSListMultipartUploadsRequest new];
listReq.bucket = @"bucket name";
// 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。
listReq.keyMarker = @"1";
// 设置返回的数量，从 1 到 1000
listReq.maxUploads = [NSNumber numberWithInteger:10];
// 指定 uploadId
listReq.uploadIdMarker = @"uploadId";

[[client listMultipartUploads:listReq] continueWithBlock:^id(OOSTask *task) {
    OOSListMultipartUploadsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.13.PUT Bucket Logging

在 PUT Bucket 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。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSLoggingEnabled *enabled = [OOSLoggingEnabled new];
enabled.targetBucket = @"bucket name";
// TargetPrefix<RegionName>/UTCYYYY-MM-DD-hh-mm-ss-唯一字符串
enabled.targetPrefix = @"nnn";

OOSBucketLoggingStatus *status = [OOSBucketLoggingStatus new];
// 如果将 loggingEnabled 设置为空，就代表删除 log
status.loggingEnabled = enabled;

OOSPutBucketLoggingRequest *putReq = [OOSPutBucketLoggingRequest new];
putReq.bucket = @"bucket name";
// 必须设置，不能为空
putReq.bucketLoggingStatus = status;

[[client putBucketLogging:putReq] continueWithBlock:^id(OOSTask *task) {
    return nil;
}];
```

3.2.14.GET Bucket Logging

在 GET Bucket 操作的 url 中加上 logging，可以获得指定 Bucket 的 logging。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetBucketLoggingRequest *getReq = [OOSGetBucketLoggingRequest new];
getReq.bucket = @"bucket name";

[[client getBucketLogging:getReq] continueWithBlock:^id(OOSTask *task) {
    // 和设置时的格式一致
    OOSGetBucketLoggingOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.15.HEAD Bucket

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

示例代码

```
OOS *client = [OOS defaultOOS];

OOSHeadBucketRequest *headBucketReq = [OOSHeadBucketRequest new];
headBucketReq.bucket = @"bucket name";

[[client headBucket:headBucketReq] continueWithBlock:^id(OOSTask *task)
{
    // task.error 为空时表示该 Bucket 存在
    return nil;
}];
```

3.2.16.PUT Bucket Lifecycle

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

示例代码

```
OOS *client = [OOS defaultOOS];

// 以天数来描述生命周期，值是正整数
OOSLifecycleExpiration *expire = [OOSLifecycleExpiration new];
expire.days = @10;
// 生命周期规则的容器
OOSRule *myRule = [OOSRule new];
// 指明要使用规则的对象前缀，最长 1024 个字符
myRule.prefix = @"key-prefix";
myRule.status = OOSExpirationStatusEnabled;
myRule.expiration = expire;

// 可以设置多个规则
OOSLifecycleConfiguration *cycleConf = [OOSLifecycleConfiguration new];
cycleConf.rules = @[myRule];

OOSPutBucketLifecycleRequest *putBucketLifeCycleReq =
[OOSPutBucketLifecycleRequest new];
putBucketLifeCycleReq.bucket = @"bucket name”;
// 设置配置
putBucketLifeCycleReq.lifecycleConfiguration = cycleConf;

[[client putBucketLifecycle:putBucketLifeCycleReq]
continueWithBlock:^id(OOSTask *task) {
    return nil;
    }];
```

3.2.17.GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期规则。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetBucketLifecycleRequest *getBucketLifeCycle =
[OOSGetBucketLifecycleRequest new];
getBucketLifeCycle.bucket = @"bucket name";

[[client getBucketLifecycle:getBucketLifeCycle]
continueWithBlock:^id(OOSTask *task) {
    // 内容格式和设置时一致
    OOSGetBucketLifecycleOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.18.DELETE Buckete Lifecycle

此操作用来删除指定 Bucket 所有生命周期。

注意：删除 Bucket 的生命周期后，用户的对象将永远不会到期，OOS 也不会再自动删除对象。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSDeleteBucketLifecycleRequest *deleteBucketLifecycleReq =
[OOSDeleteBucketLifecycleRequest new];
deleteBucketLifecycleReq.bucket = @"bucket name";

[[client deleteBucketLifecycle:deleteBucketLifecycleReq]
continueWithBlock:^id(OOSTask *task) {
    // 操作失败时 task.error 不为空
    return nil;
}];
```

3.2.19.PUT Bucket CORS

跨域资源共享(Cross-Origin Resource Sharing，CORS)定义了客户端 Web 应用程序在一个域中与另一个域中的资源进行交互的方式，是浏览器出于安全考虑而设置的一个限制，即同源策略。例如，当来自于 A 网站的页面中的 JavaScript 代码希望访问 B 网站的时候，浏览器会拒绝该访问，因为 A、B 两个网站是属于不同的域。

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

示例代码

```
OOS *client = [OOS defaultOOS];

// 可以存储多个
NSMutableArray *corsRules = [NSMutableArray new];

OOSCORSRule *rule = [OOSCORSRule new];
// 每个规则都有一个 ID
rule.identifier = @“A";
// 设定 HTTP Method
rule.allowedMethods = @[@“PUT",@"POST",@"DELETE"];
// 允许跨域的源。最多包含一个 * 通配 符。
rule.allowedOrigins = @[@"*"];
[corsRules addObject:rule];

OOSCORSRule *rule2 = [OOSCORSRule new];
rule2.identifier = @“B";
rule2.allowedOrigins = @[@“*"];
// 只允许执行 get 请求
rule2.allowedMethods = @[@"GET"];
rule2.allowedHeaders = @[@"*"];
[corsRules addObject:rule2];

OOSCORSConfiguration *cors = [OOSCORSConfiguration new];;
cors.CORSRules = corsRules;
```

```
OOSPutBucketCorsRequest *putBucketCorsReq = [OOSPutBucketCorsRequest
new];
putBucketCorsReq.bucket = @"bucket name";
putBucketCorsReq.CORSConfiguration = cors;

[[client putBucketCors:putBucketCorsReq] continueWithBlock:^id(OOSTask
*task) {
    return nil;
}];
```

3.2.20.GET Bucket CORS

此操作用来返回 Bucket 的跨域配置信息。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSGetBucketCorsRequest *getBucketCorsReq = [OOSGetBucketCorsRequest
new];
getBucketCorsReq.bucket = @"bucket name";

[[client getBucketCors:getBucketCorsReq] continueWithBlock:^id(OOSTask
*task) {
    // 内容格式和设置时一致
    OOSGetBucketCorsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.2.21.DELETE Bucket CORS

删除 Bucket 的跨域配置信息。

- 如果 Bucket 配置了 CORS，删除成功，返回 200 0K。
- 如果 Bucket 没有配置 CORS，返回 204 NoContent。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSDeleteBucketCorsRequest *deleteBucketCorsReq =
[OOSDeleteBucketCorsRequest new];
deleteBucketCorsReq.bucket = @"bucket name";

[[client deleteBucketCors:deleteBucketCorsReq]
continueWithBlock:^id(OOSTask *task) {
    // task.error 为空时表示操作成功
    return nil;
}];
```

3.3.关于 Object 的操作

3.3.1.PUT Object

此操作用来向指定 Bucket 中添加一个对象。

示例代码

```
OOS *client = [OOS defaultOOS];

// 生成一个测试数据
NSString *testObjectStr = @"a test object string.";
NSData *testObjectData = [testObjectStr
dataUsingEncoding:NSUTF8StringEncoding];

OOSPutObjectRequest *putObjectRequest = [OOSPutObjectRequest new];
putObjectRequest.bucket = @"bucket name";
putObjectRequest.key = @"object name";
// 设置数据内容
putObjectRequest.body = testObjectData;
// 设置过期时间
putObjectRequest.expires = [NSDate dateWithTimeIntervalSinceNow:100];
// 设置数据长度
putObjectRequest.contentLength = [NSNumber
numberWithUnsignedInteger:[testObjectData length]];
// 设置 contentType
putObjectRequest.contentType = @"text/html";

// 添加自定义元数据
NSDictionary *userMetaData = @{@"user-data-1": @"user-metadata-
value1",
                               @"user-data-2": @"user-metadata-value2",
                           };
putObjectRequest.metadata = userMetaData;

[[client putObject:putObjectRequest] continueWithBlock:^id(OOSTask
*task) {
    OOSPutObjectOutput *output = task.result;
```

```
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

此操作用来检索在 OOS 中的对象信息，执行此操作，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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 指定 Bucket 和 Object 名称
OOSGetObjectRequest *getObjectRequest = [OOSGetObjectRequest new];
getObjectRequest.bucket = @"bucket name";
getObjectRequest.key = @"object name";

[[client getObject:getObjectRequest] continueWithBlock:^id(OOSTask
*task) {
    // 返回对象的元数据
    OOSGetObjectOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.2.DELETE Object

此操作用来删除指定的对象，要求用户要对对象所在的 Bucket 拥有写权限。

示例代码

```
OOS *client = [OOS defaultOOS];


OOSDeleteObjectRequest *deleteObjectRequest = [OOSDeleteObjectRequest
new];
deleteObjectRequest.bucket = @"bucket name";
deleteObjectRequest.key = @"object name";


[[client deleteObject:deleteObjectRequest] continueWithBlock:^id(OOSTask
*task) {
    // 返回值暂时可以忽略
    OOSDeleteObjectOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.3.PUT Object - Copy

此操作用来创建一个存储在 OOS 里对象的拷贝。PUT Object - Copy 操作类似于执行一个 GET Object，然后再执行一次 PUT Object。用户对源对象有读权限，对目标 Bucket 有写权限，才能执行 PUT Object - Copy 操作。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSCopyObjectRequest *copyReq = [OOSCopyObjectRequest new];
// 目标 Bucket，可以支持不同的桶之间 copy 对象
copyReq.bucket = @"bucket name";
// 新对象名称
copyReq.key = @"object name";
// 要 copy 的目标对象名称，写法为：bucket/key
copyReq.replicateSource = @“sourceBucket/sourceObjectName";

[[client copyObject:copyReq] continueWithBlock:^id(OOSTask *task) {
    OOSCopyObjectOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.4.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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 指定某个 Object 使用分片上传的方式来生成
OOSCreateMultipartUploadRequest *createReq =
[OOSCreateMultipartUploadRequest new];
createReq.bucket = @"bucket name";
createReq.key = @"object name";
// 设置过期时间，操作该时间如果还完成 complete 操作则自动终止
createReq.expires = [NSDate dateWithTimeIntervalSinceNow:100];

[[client createMultipartUpload:createReq] continueWithBlock:^id(OOSTask
*task) {
    // 返回一个 upload id
    // 后续操作都需要使用这个 upload id
    OOSCreateMultipartUploadOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.5.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。

示例代码

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。

```
OOS *client = [OOS defaultOOS];

// 生成一个测试数据
NSData *testData = [@"this is a test string"
dataUsingEncoding:NSUTF8StringEncoding];
NSInteger dataLength = [testData length]/10;
NSData *partData = [testData subdataWithRange:NSMakeRange(0,
dataLength)];

OOSUploadPartRequest *uploadPartRequest = [OOSUploadPartRequest new];
uploadPartRequest.bucket = @"bucket name";
uploadPartRequest.key = @"object name";
// 设置分片序号
uploadPartRequest.partNumber = @(0);
// 设置分片数据内容
uploadPartRequest.body = partData;
```

```
// 设置分片数据长度
uploadPartRequest.contentLength = @(dataLength);
// 设置此次上传的 id
uploadPartRequest.uploadId = @"upload id";

[[client uploadPart:uploadPartRequest]
continueWithSuccessBlock:^id(OOSTask *task) {
    OOSUploadPartOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.6.Complete Multipart Upload

该接口通过合并之前的上传片段来完成一次分片上传过程。

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

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

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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 调用 uploadPart 请求后，会得到一个对应的 ETag
// 需要讲每个 part 的 number 和 ETag 保存起来
NSMutableArray *completedParts = [NSMutableArray
arrayWithCapacity:2];

OOSCompletedPart *completedPart1 = [OOSCompletedPart new];
completedPart1.partNumber = @(1);
```

```
// 替换为真实值
completedPart1.ETag = @"xxxxx";
[completedParts addObject:completedPart1];

OOSCompletedPart *completedPart2 = [OOSCompletedPart new];
completedPart2.partNumber = @(2);
// 替换为真实值
completedPart2.ETag = @"yyyyy";
[completedParts addObject:completedPart2];

OOSCompletedMultipartUpload *multipartUpload =
[OOSCompletedMultipartUpload new];
multipartUpload.parts = completedParts;

OOSCompleteMultipartUploadRequest *compReq =
[OOSCompleteMultipartUploadRequest new];
compReq.bucket = @"bucket name";
compReq.key = @"object name";
// 替换为真实值
compReq.uploadId = @"upload id";
compReq.multipartUpload = multipartUpload;

[[[client completeMultipartUpload:compReq]
continueWithBlock:^id(OOSTask *task) {

    OOSCompleteMultipartUploadOutput *compOutput = task.result;
    NSLog(@"result  %@", [output description]);

    return nil;
}] waitUntilFinished];
```

3.3.7.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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 设置要终止的 upload 信息，必须设置 uploadId
OOSAbortMultipartUploadRequest *abortReq = [OOSAbortMultipartUploadRequest new];
abortReq.bucket = @"bucket name";
abortReq.key = @"object name";
abortReq.uploadId = @"upload id";

[[client abortMultipartUpload:abortReq] continueWithSuccessBlock:^id(OOSTask *task) {
    OOSAbortMultipartUploadOutput *output = task.result;
    NSLog(@"result    %@", [output description]);
    return nil;
}];
```

3.3.8.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。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSListMultipartUploadsRequest *listReq = [OOSListMultipartUploadsRequest
new];
listReq.bucket = @"bucket name";
// 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。
listReq.keyMarker = @"1";
// 设置返回的数量，从 1 到 1000
listReq.maxUploads = [NSNumber numberWithInteger:10];
// 指定 uploadId
listReq.uploadIdMarker = @"uploadId";

[[client listMultipartUploads:listReq] continueWithBlock:^id(OOSTask
*task) {
    OOSListMultipartUploadsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.9.Copy Part

此操作将已经存在的 Object 作为分段上传的片段，拷贝生成一个新的片段。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSUploadPartCopyRequest *copyReq = [OOSUploadPartCopyRequest new];
copyReq.bucket = @"bucket name";
copyReq.key = @"object name";
// 设置 upload id
copyReq.uploadId = @"upload id";
// 设置 part number
copyReq.partNumber = [NSNumber numberWithInteger:1];
// 要 copy 的数据源，bucket 和 object 名称之间必须用/区分
copyReq.replicateSource = [NSString stringWithFormat:@"%@/%@", @"bucket
name", @"temp.txt"];

[[client uploadPartCopy:copyReq] continueWithSuccessBlock:^id(OOSTask
*task) {
    OOSUploadPartCopyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.10.Delete Multiple Objects

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。对于要删除的每个 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。

示例代码

```
OOS *client = [OOS defaultOOS];

// 生成一个存放 OOSObjectIdentifier 的数组
NSMutableArray *objectsArray = [NSMutableArray new];
for (int i=0;i<5;i++) {
    NSString *keyName = [NSString stringWithFormat:@"key%d",i];
    OOSObjectIdentifier *obj = [OOSObjectIdentifier new];
    // 要删除的 object 名称
    obj.key = keyName;
    [objectsArray addObject:obj];
}

OOSRemove *removes = [OOSRemove new];
// 使用简明信息模式来返回响应，有效 值为:true 和 false
removes.quiet = [NSNumber numberWithBool:false];
// 设置要删除的数组
removes.objects = objectsArray;

OOSDeleteObjectsRequest *multipleObjectsDeleteReq =
[OOSDeleteObjectsRequest new];
// 只能在一个 bucket 中批量删除
```

```
multipleObjectsDeleteReq.bucket = @"bucket name";
multipleObjectsDeleteReq.remove = removes;

[[client deleteObjects:multipleObjectsDeleteReq ]
continueWithBlock:^id(OOSTask *task) {
    OOSDeleteObjectsOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.3.11.断点续传

通过 OOSTransferManager 辅助类上传文件时，如果文件过大（超过 5Mb），会自动实现断点续传。

上传过程中会在本地存储一个临时文件，用来保存上传成功的分片信息，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件会被删除。该文件名称可以通过 cacheIdentifier 属性自定义。

注意：如果用户在上传文件过程中强制退出，或者 App 崩溃，需要继续之前的断点续传，在客户端需要主动保存四个临时变量：currentUploadingPartNumber、completedPartsArray、totalSuccessfullySentPartsDataLength、uploadId，等 App 重启后重新生成一个具有相同的 cacheIdentifier 的 request，再重新赋值之前保存的四个参数，就可以保存之前上传的进度。

示例代码

```
OOSTransferManager *transferManager = [OOSTransferManager
defaultTransferManager];

OOSTransferManagerUploadRequest *uploadRequest =
[OOSTransferManagerUploadRequest new];
// 配置必填字段，指定 Bucket 和 Object 名称
uploadRequest.bucket = @"bucket name";
uploadRequest.key = @"object name";
// 指定需要上传的本地文件
NSURL *dataURL = [NSURL
fileURLWithPath:[[NSSearchPathForDirectoriesInDomains(NSCachesDirectory,
NSUserDomainMask, YES) objectAtIndex:0] stringByAppendingPathComponent:@"file
name"]];
uploadRequest.body = dataURL;
// 可以自定义缓存名称，建议不设置
uploadRequest.cacheIdentifier = @"自定义名称.ucp";

__block int64_t accumulatedUploadBytes = 0;
__block int64_t totalUploadedBytes = 0;
```

```
__block int64_t totalExpectedUploadBytes = 0;
uploadRequest.uploadProgress = ^(int64_t bytesSent, int64_t totalBytesSent,
int64_t totalBytesExpectedToSend) {
    // 指定当前上传长度、当前已经上传总长度、待上传的总长度。
    NSLog(@"bytesSent: %lld, totalBytesSent: %lld,
totalBytesExpectedToSend: %lld",bytesSent,totalBytesSent,totalBytesExpectedToSe
nd);
    accumulatedUploadBytes += bytesSent;
    totalUploadedBytes = totalBytesSent;
    totalExpectedUploadBytes = totalBytesExpectedToSend;
};

[[transferManager upload:uploadRequest] continueWithBlock:^id(OOSTask *task) {
    // 成功后无返回值
    return nil;
}];
```

3.3.12.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 分享给其他用户，同时可以在链接中设置限速以对下载速度进行控制。

示例代码

```
OOSPreSignedURLBuilder *builder = [OOSPreSignedURLBuilder
defaultPreSignedURLBuilder];

OOSGetPreSignedURLRequest *getPreSignedURLRequest =
[OOSGetPreSignedURLRequest new];
getPreSignedURLRequest.bucket = @"bueckt name";
getPreSignedURLRequest.key = @"object name";
// 允许使用的 Method
getPreSignedURLRequest.HTTPMethod = OOSHTTPMethodGET;
// 设置过期时间
getPreSignedURLRequest.expires = [NSDate
dateWithTimeIntervalSinceNow:3600];
// 设置限速
getPreSignedURLRequest.limitrate = @(2048);

[[builder getPreSignedURL:getPreSignedURLRequest]
continueWithBlock:^id(OOSTask *task) {
    // result 就是地址
    NSURL *presignedURL = task.result;
    NSLog(@"presignedURL = %@", presignedURL);
    return nil;
}];
```

3.3.13.HEAD Object

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

示例代码

```
OOS *client = [OOS defaultOOS];

OOSHeadObjectRequest *req = [OOSHeadObjectRequest new];
req.bucket = @"bucket name";
req.key = @"object name";

[[[client headObject:req] continueWithBlock:^id(OOSTask *task) {
   // 获取元数据，ETag / contentLength / contentType
    OOSHeadObjectOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}] waitUntilFinished];
```

3.4.关于 AccessKey 的操作

3.4.1.CreateAccessKey

此操作用来为指定的 IAM 用户创建新的 AccessKey。

说明：

- 新密钥的默认状态是 Active。
- 如果未指明 IAM 用户名，则为请求者创建新的 AccessKey。

示例代码

```
OOS *client = [OOS defaultOOS];


OOSCreateAccessKeyRequest *createReq = [OOSCreateAccessKeyRequest
new];
[[client createAccessKey:createReq] continueWithBlock:^id(OOSTask
*task) {
    // 通过如下操作获取 id， accessKeyId =
output.createAccessKeyResponse.createAccessKeyResult.accessKey.accessK
eyId;
    OOSCreateAccessKeyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.4.2.DeleteAccessKey

此操作用来删除指定 IAM 用户关联的 AccessKey。

说明：

- 如果未指定用户名，IAM 将根据签名请求的 OOS AccessKeyID 确定用户名。
- 此操作也可以用来删除根用户的 AccessKey。

示例代码

```
OOS *client = [OOS defaultOOS];


OOSDeleteAccessKeyRequest *deleteReq = [OOSDeleteAccessKeyRequest new];
deleteReq.accessKeyId = @"access key id";
[[client deleteAccessKey:deleteReq] continueWithBlock:^id(OOSTask
*task) {
    // 详情需要查看 responseMetadata 属性
    OOSDeleteAccessKeyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.4.3.UpdateAccessKey

此操作用来更新指定访问密钥（AK）的状态，从 Active 到 Inactive，或者从 Inactive 到 Active。

说明：

- 如果请求中未携带 IAM 用户名，则更新请求者的密钥状态。

- 此操作可以管理根用户的密钥。

示例代码

```
OOS *client = [OOS defaultOOS];


OOSUpdateAccessKeyRequest *updateReq = [OOSUpdateAccessKeyRequest new];
// 需要先获取到 ID 再更改
updateReq.accessKeyId = @"access key id";
// 字符串，Active｜Inactive
updateReq.status = @"Inactive";
// 字符串，不是 bool
updateReq.isPrimary = @"true";


[[client updateAccessKey:updateReq] continueWithBlock:^id(OOSTask
*task) {
    OOSUpdateAccessKeyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```

3.4.4.ListAccessKey

此操作用来返回指定 IAM 用户的 AK 的详细信息。

说明：

- 如果指定用户没有 AK，则操作返回空列表。
- 如果未指定 IAM 用户，则返回请求者的 AK。

示例代码

```
OOS *client = [OOS defaultOOS];

OOSListAccessKeyRequest *listReq = [OOSListAccessKeyRequest new];
// 设置最多返回数量
listReq.maxItems = [NSNumber numberWithInteger:10];

[[client listAccessKey:listReq] continueWithBlock:^id(OOSTask *task) {
    OOSListAccessKeyOutput *output = task.result;
    NSLog(@"result  %@", [output description]);
    return nil;
}];
```