天翼云

对象存储

(Object-Oriented Storage,OOS)

OOS C SDK 开发者指南 V5

天翼云科技有限公司

目录

前言.....4
使用条件.....5
先决条件.....5
下载及安装.....5
环境依赖.....6
第三方库下载以及安装.....6
libcurl（建议 7.32.0 及以上版本）.....6
apr （建议 1.5.2 及以上版本）.....6
apr-util （建议 1.5.4 及以上版本）.....7
minixml（建议版本：2.8、2.9、2.10、2.11、2.12）.....7
SDK 使用设置.....8
基本设置.....8
接口函数调用流程.....8
OOS 服务代码示例.....10
关于 Service 的操作.....10
GET Service(List Bucket).....10
关于 Bucket 的操作.....11
PUT Bucket.....11
GET Bucket acl.....12
GET Bucket (List Objects).....13
DELETE Bucket.....14
PUT Bucket Policy.....15
GET Bucket Policy.....15
DELETE Bucket Policy.....16
PUT Bucket WebSite.....16
GET Bucket WebSite.....17
DELETE Bucket WebSite.....18
List Multipart Uploads.....19
PUT Bucket Logging.....20
GET Bucket Logging.....21

HEAD Bucket 22
PUT Bucket Trigger 22
GET Bucket Trigger 23
DELETE Bucket Trigger 24
PUT Bucket Lifecycle 25
GET Bucket Lifecycle 26
DELETE Bucket Lifecycle 27
PUT Bucket cors 27
GET Bucket cors 29
DELETE Bucket cors 31
关于 Object 的操作 33
PUT Object 33
GET Object 34
DELETE Object 35
PUT Object - Copy 35
Initial Multipart Upload 36
Upload Part 37
Complete Multipart Upload 39
Abort Multipart Upload 41
List Part 41
Copy Part 42
断点续传 43
Delete Multiple Objects 43
生成共享链接 45
HEAD Object 46
关于 AccessKey 的操作 47
CreateAccessKey 47
DeleteAccessKey 48
UpdateAccessKey 48
ListAccessKey 49
STS 临时授权访问 50

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。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。

使用条件

先决条件

用户需要具备以下条件，然后才能够使用 OOS SDK C 版本：

- 一个 OOS 账户；
- 已经安装 libcurl、apr、apr-util、minixml；
- 已获取 AccessKeyId 和 SecretKey；
- 熟悉各接口的参数和响应参数的使用方法，详见《OOS 开发者文档》。

下载及安装

从官方渠道下载 C SDK 的压缩包，放到相应位置后并解压。命令可以参考：

```
tar -xzvf oos-c-sdk.tar.gz
cd oos-c-sdk/
```

用户可以使用下列两种方法进行安装使用：

方法一：直接使用 oos-sdk 源码：

假设用户使用源码文件为 main_new.c，其中 Makefile 为 Makefile_oostest，使用如下方式进行则编译：

```
make -f Makefile_oostest  //该命令生成 oos_test 可执行文件
```

方法二：使用 liboos. so 动态链接库

1. 使用命令 make 编译 oos-c-sdk。
2. 拷贝 liboos.so 到/usr/lib64/目录下

```
cp liboos.so /usr/lib64/
```

3. 创建文件夹/usr/include/oos/ ，并将下列头文件拷贝到该文件夹内：
   buffer.h、cjson.h、json_util.h、list.h、oos_api_define.h、oos_api.h、oos_define.h、oos_status.h、oos_string.h、response.h。
4. 修改/etc/ld.so.conf，增加以下内容，分别对应 curl 与 mxml、apr 的动态链接库：

```
/usr/local/lib/
/usr/local/apr/lib/
```

5. 执行 shell 命令：

```
ldconfig
```

6. 假设 test.c 为源码文件，其包含了头文件“#include <oos/oos_api.h>”，编译命令如下：

```
gcc  -I /usr/local/apr/include/apr-1/ -L/usr/local/apr/lib/ -lapr-1 -
loos -lcurl -lmxml -lm -pthread -lcrypto test.c -o main
```

环境依赖

OOS C SDK 使用 curl 进行网络操作，无论是作为客户端还是服务器端，都需要依赖 curl。OSS C SDK 使用 apr/apr-util 库解决内存管理以及跨平台问题，使用 minixml 库解 析请求返回的 xml。OOS C SDK 并没有带上这三个外部库，您需要确认这三个库已经安 装，并且将它们的头文件目录和库文件目录都加入到了项目中。

第三方库下载以及安装

libcurl（建议 7.32.0 及以上版本）

请从这里下载，并参考 libcurl 安装指南安装。典型的安装方式如下：

```
./configure
Make
make install
```

注意：执行./configure 时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用 ./configure --prefix=/your/install/path/

apr （建议 1.5.2 及以上版本）

请从官网下载，典型的安装方式如下：

```
./configure
Make
make install
```

注意：执行./configure 时默认是配置安装目录为/usr/local/apr/，如果需要指定安装目录，请使用 ./configure --prefix=/your/install/path/

apr-util （建议 1.5.4 及以上版本）

请从官网下载，安装时需要注意指定--with-apr 选项，典型的安装方式如下：

```
./configure –with-apr=/your/apr/install/path
make
make install
```

注意：

- 执行./configure 时默认是配置安装目录为/usr/local/apr/，如果需要指定安装目录，请使用 ./configure --prefix=/your/install/path/
- 需要通过--with-apr 指定 apr 安装目录，如果 apr 安装到系统目录下需要指定 -with-apr=/usr/local/apr/

minixml（建议版本：2.8、2.9、2.10、2.11、2.12）

请从这里下载，典型的安装方式如下:

```
./configure
make
make install
```

注意：执行./configure 时默认是配置安装目录为/usr/local/，如果需要指定安装目录，请使用 ./configure --prefix=/your/install/path/

SDK 使用设置

基本设置

使用 sdk 访问 OOS 的服务，需要设置正确的 AccessKeyId、SecretKey 和服务端 Endpoint，所有的服务可以使用同一 key 凭证来进行访问，但不同的服务需要使用不同的 endpoint 进行访问。OOS 的服务端 endpoint 请参见 https://www.ctyun.cn/help2/10000101/10474062，其他服务端 endpoint 详见各服务代码示例章节。

接口函数调用流程

每个接口调用需要完成同样的准备工作：

1.初始化内存池 oos_initialize；

2.创建内存池 oos_pool_create；

3.创建 options 并设置 AccessKey、SecretKey 和 HostName；

4.调用对应接口；

5.释放内初池。

访问 OOS 的服务示例代码如下：

```
int main() {
    // 初始化内存池
    oos_initialize();
    oos_pool_t *p = NULL;
    request_options_t *options = NULL;
    oos_status_t *status = NULL;
    // 创建内存池
    oos_pool_create(&p, NULL);
    options = (request_options_t *)oos_pcalloc(p,
sizeof(request_options_t));
    options->pool = p; options->is_v4 = OOS_TRUE;
    options->is_sign_payload = OOS_TRUE;
    // 设置 options 信息，用户的 AK、SK 以及需要访问的 endpoint
    oos_str_set(options->pool, &options->access_key, ACCESS_KEY);
    oos_str_set(options->pool, &options->secret_key, SECRET_KEY);
    oos_str_set(options->pool, &options->endpoint, HOST_NAME);
```

```
    // 调用接口
    list_all_bucket_test (options, status);
    // 释放内存池
    oos_pool_destroy(p);
    oos_terminate();
    return 0;
}
```

不同的服务需要使用不同的 endpoint 进行访问，上例进行了对象存储服务（OOS）配置设置。

OOS 服务代码示例

OOS 的服务代码示例是根目录的 **main_new.c** 文件。

关于 Service 的操作

GET Service(List Bucket)

此操作用来返回请求者拥有的所有 Bucket，其中“/”表示根目录。

该 API 只对验证用户有效，匿名用户不能执行该操作。

- 示例代码

```
void list_all_bucket_test() {
    oos_pool_t *p = NULL;
    request_options_t *options = NULL;
    oos_status_t *status = NULL;
    oos_list_bucket_t *buckets = NULL;
    oos_pool_create(&p, NULL);
    options = (request_options_t *)oos_pcalloc(p,
sizeof(request_options_t));

    options->pool = p;
    oos_str_set(options->pool,  &options->access_key, ACCESS_KEY);
    oos_str_set(options->pool,  &options->secret_key, SECRET_KEY);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    options->is_v4 = OOS_TRUE;
    options->is_sign_payload = OOS_TRUE;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    status = oos_list_buckets(options, &buckets, &resp_headers);
    if (status)
        printf("service code %d\n", status->code);

    if (buckets && buckets->root) {
        struct list_head *pos = NULL;
        list_for_each(pos, buckets->root) {
            oos_list_bucket_content_t* temp = list_entry(pos,struct
oos_list_bucket_content_t,node);
```

```
            PRINT_OOS_STRING(name, temp->name);
            PRINT_OOS_STRING(creationDate, temp->creationDate);
        }
        printf("owner_id:%s\n", buckets->owner_id.data);
        printf("display_name:%s\n", buckets->owner_DisplayName.data);
    }

    oos_pool_destroy(p);
}
```

关于 Bucket 的操作

PUT Bucket

此操作用来创建一个新的 Bucket。Bucket 的命名方式中并不是支持所有的字符，具体请参见《OOS 开发者文档》Bucket 命名规范。

- 示例代码

```
void create_bucket_tolocation_test(request_options_t
*options,oos_status_t *status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    // make location config
    struct location_configuration_t* config = NULL;
    config = oos_pcalloc(options->pool, sizeof(*config));

    oos_str_set(options->pool,  &config->medadata_location,"ShenZhen");
    config->data_location_type = LOCAL;
    config->data_schedule_strategy = ALLOWED;

    // add multiple data location to locationg config
    INIT_LIST_HEAD(&config->data_location_list.node);
    add_list_string_node(options, "ShenZhen",
&config->data_location_list);

    status = oos_create_bucket_tolocation(options, "test-oos5-bucket1",
BUCKET_TYPE_PUBLIC_READ, config, &resp_headers);
    if (status) {
```

```
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

GET Bucket acl

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

- 示例代码

```
void get_bucket_acl_test(request_options_t *options, oos_status_t
*status) {
    acl_configuration_t* config=NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_get_bucket_acl(options, "test-oos5-bucket1", &config,
&resp_headers);
    if (status)
        printf("service code %d\n", status->code);

    char result[2048] = {"\0"};
    if(config) {
        if(config->id.len > 0) {
            strcat(result, "owner Id:");
            strcat(result, config->id.data);
        }

        if(config->display_name.len > 0) {
            strcat(result, "owner DisplayName:");
            strcat(result, config->display_name.data);
        }

        strcat(result, "\n");
        acl_grant_info* tmp = NULL;
        list_for_each_entry(tmp, &config->grant_list.node, node) {
            // Grantee URI
            if(tmp->grantee_uri.len > 0) {
```

```
                strcat(result, "Grantee URI:");
                strcat(result, tmp->grantee_uri.data);
                strcat(result, " ");
            }

            // Permission
            if(tmp->permission.len > 0) {
                strcat(result, "Permission:");
                strcat(result, tmp->permission.data);
                strcat(result, "\n");
            }
        }
    }
    printf("==============result:\n%s\n", result);
}
```

GET Bucket (List Objects)

此操作返回 bucket 中部分或者全部（每次最多 1000）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。

- 示例代码

```
void list_objects_test(request_options_t *options, oos_status_t *status)
{
    oos_list_object_t *objects = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool, &options->endpoint, HOST_NAME);

    status = oos_list_objects(options, "test-oos5-js0", NULL, NULL,
NULL, "1", &objects, &resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

    if (objects) {
        struct list_head *pos = NULL;
        oos_list_object_content_t *tmp = NULL;
```

```
        list_for_each_entry(tmp, objects->root, node) {
            printf("name: %s, etag: %s\n", tmp->key.data,
tmp->eTag.data);
            PRINT_OOS_STRING(lastModified, tmp->lastModified);
            PRINT_OOS_STRING(size, tmp->size);
            PRINT_OOS_STRING(storageClass, tmp->storageClass);
            PRINT_OOS_STRING(owner_id, tmp->owner_id);
            PRINT_OOS_STRING(owner_DisplayName, tmp->owner_DisplayName);
        }

        printf("delimiter:%s\n", objects->delimiter.data);
        printf("prefix:%s\n", objects->prefix.data);
        printf("marker:%s\n", objects->marker.data);
        printf("maxKeys:%s\n", objects->maxKeys.data);
        printf("isTruncated:%s\n", objects->isTruncated.data);
        printf("name:%s\n", objects->name.data);
        printf("nextMarker:%s\n", objects->nextMarker.data);

        list_for_each(pos, objects->prefix_root) {
            oos_list_object_common_prefix_t* temp =
list_entry(pos,struct oos_list_object_common_prefix_t,node);
            printf("0000000 id %s\n", temp->prefix.data);
        }
    }
}
```

DELETE Bucket

该操作用来删除 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

```
void delete_bucket_test(request_options_t *options,oos_status_t *status)
{
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_bucket(options,"test-oos5-
bucket1",&resp_headers);
```

```
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

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。

- 示例代码

其中 **test-oos5-js0** 为用户自己的 Bucket 名字。

```
void put_bucket_policy_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    char text[] = "{\"Version\":\"2012-10-17\",\"Id\":\"http referer
policy example\","
                  "\"Statement\":[{\"Sid\":\"*\", "
                  "\"Effect\":\"Allow\",\"Principal\":{ \"AWS\":
[\"*\"] }, \"Action\":\"s3:*\", "
                  "\"Resource\":\"arn:aws:s3:::test-oos5-js0/*\",
\"Condition\":{\"StringLike\":"

"{\"aws:Referer\":[\"http://www.mysite.com/*\"]} }} ]}";
    status = oos_put_bucket_policy(options, "test-oos5-js0", text,
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

GET Bucket Policy

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

- 示例代码

```
void get_bucket_policy_test(request_options_t *options,oos_status_t
*status) {
    char* policy_text= NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_get_bucket_policy(options, "test-oos5-js0",
&policy_text,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

    printf("policy text %s\n", policy_text);
}
```

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

```
void delete_bucket_policy_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_bucket_policy(options,"test-oos5-
js0",&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

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。

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-cn.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-cn.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-cn.ctyunapi.cn”。

- **示例代码**

```
void put_bucket_website_test(request_options_t *options,oos_status_t
*status) {
    website_configuration_t* configweb = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    configweb = oos_pcalloc(options->pool,sizeof(*configweb));
    if (!configweb) {
        printf("oos_pcalloc failed\n");
        return;
    }

    oos_str_set(options->pool,
&configweb->suffix,"test.suffix.index.html");
    oos_str_set(options->pool,  &configweb->key,"test.key.index.html");

    status = oos_put_bucket_website(options, "test-oos5-
js0",configweb,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

GET Bucket WebSite

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

- **示例代码**

```
void get_bucket_website_test(request_options_t *options,oos_status_t
*status) {
    website_configuration_t* configweb = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_get_bucket_website(options,"test-oos5-js0",&configweb,
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

    if (configweb) {
        printf("website prefix %s\n", configweb->suffix.data);
        printf("website key %s\n", configweb->key.data);
    }
}
```

DELETE Bucket WebSite

在 DELETE 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

```
void delete_bucket_website_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_bucket_website(options, "test-oos5-js0",
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

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。

- 示例代码

```
void list_multipart_uploads_test(request_options_t *options,oos_status_t
*status) {
    oos_list_multipart_uploads_t* list_upload = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_list_multipart_uploads(options, "test-oos5-bucket0",
NULL, "/", "3", "test-multipart.txt", "1603781746783056328",
&list_upload, &resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }

    if (list_upload){
        struct list_head *pos;
        if (list_upload) {
            printf("bucket:%s ", list_upload->bucket.data);
            printf("isTruncated:%s ", list_upload ->isTruncated.data);
            printf("keyMarker:%s ", list_upload->keyMarker.data);
            printf("maxUploads:%s ", list_upload->maxUploads.data);
            printf("nextKeyMarker:%s ",
list_upload->nextKeyMarker.data);
            printf("nextUploadIdMarker:%s ",
list_upload->nextUploadIdMarker.data);
            printf("uploadIdMarker:%s\n",
list_upload->uploadIdMarker.data);
        }

        list_for_each(pos, list_upload->root) {
            oos_list_multipart_uploads_content_t* temp =
list_entry(pos,struct oos_list_multipart_uploads_content_t,node);
            printf("upload id %s\n", temp->uploadId.data);
            printf("key %s\n", temp->key.data);
```

```
            printf("initiator_id %s\n", temp->initiator_id.data);
            printf("initiator_displayName %s\n",
temp->initiator_displayName.data);
            printf("owner_id %s\n", temp->owner_id.data);
            printf("owner_displayName %s\n",
temp->owner_displayName.data);
            printf("storageClass %s\n", temp->storageClass.data);
            printf("initiated %s\n", temp->initiated.data);
        }

        list_for_each(pos, list_upload->prefix_root) {
            oos_list_object_common_prefix_t* temp = list_entry(pos,
struct oos_list_object_common_prefix_t, node);
            printf("commain prefix: %s\n", temp->prefix.data);
        }
    }
}
```

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。

- 示例代码

```
void put_bucket_logging_test(request_options_t *options,oos_status_t
*status) {
    logging_configuration_t* configlog = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    configlog = oos_pcalloc(options->pool,sizeof(*configlog));
    if (!configlog) {
        printf("%s: oos_pcalloc failed\n", __FUNCTION__);
        return;
    }

    oos_str_set(options->pool,  &configlog->targetBucket,"test-oos5-
js0");
    oos_str_set(options->pool,  &configlog->targetPrefix,"testw5");
```

```
    oos_str_set(options->pool,  &configlog->triggerTargetBucket,"test-
oos5-target");
    oos_str_set(options->pool,
&configlog->triggerTargetPrefix,"triggerTargetPrefix");
    oos_str_set(options->pool,  &configlog->triggerSourceBucket,"test-
oos5-source");
    oos_str_set(options->pool,
&configlog->triggerSourcePrefix,"triggerSourcePrefix");
    status = oos_put_bucket_logging(options,"test-oos5-
js0",configlog,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

GET Bucket Logging

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

- 示例代码

```
void get_bucket_logging_test(request_options_t *options,oos_status_t
*status) {
    logging_configuration_t* config = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_get_bucket_logging(options,"test-oos5-
js0",&config,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

    if (config) {
        PRINT_OOS_STRING(targetBucket, config->targetBucket);
        PRINT_OOS_STRING(targetPrefix, config->targetPrefix);
        PRINT_OOS_STRING(triggerTargetBucket,
config->triggerTargetBucket);
        PRINT_OOS_STRING(triggerTargetPrefix,
config->triggerTargetPrefix);
```

```
        PRINT_OOS_STRING(triggerSourceBucket,
config->triggerSourceBucket);
        PRINT_OOS_STRING(triggerSourcePrefix,
config->triggerSourcePrefix);
    }
}
```

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

```
void head_bucket_test(request_options_t *options,oos_status_t *status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_head_bucket(options,"test-oos5-js0",&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }
}
```

PUT Bucket Trigger

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

- 示例代码

```
void put_bucket_trigger_test(request_options_t *options,oos_status_t
*status){
```

```
    trigger_configuration_t* config = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    config = oos_pcalloc(options->pool,sizeof(*config));

    oos_str_set(options->pool,  &options->endpoint, "oos-
js.ctyunapi.cn");

    oos_str_set(options->pool,&config->triggerName,"testtrigger");
    oos_str_set(options->pool,&config->isDefault,"false");
    oos_str_set(options->pool,&config->remontEndPoint, "http://oos-
js.ctyunapi.cn");
    oos_str_set(options->pool,&config->remoteBucketName, BUCKET_NAME);
    oos_str_set(options->pool,&config->remoteAK, ACCESS_KEY);
    oos_str_set(options->pool,&config->remoteSK, SECRET_KEY);

    trigger_configuration_t configs[1];
    memcpy((char *)&configs[0],(char *)config,sizeof(*config));

    status = oos_put_bucket_trigger(options,
BUCKET_NAME,configs,1,&resp_headers);
    if (status) {
        printf("service code: %d\n", status->code);
        printf("error msg: %s\n", status->error_msg);
    }
}
```

GET Bucket Trigger

在 GET 操作的 url 中加上 trigger，可以查询某 bucket 中配置的所有 trigger。

- 示例代码

```
void get_bucket_trigger_test(request_options_t *options,oos_status_t
*status){
    trigger_configuration_t* config = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, "oos-
js.ctyunapi.cn");
```

```
    status = oos_get_bucket_trigger(options, BUCKET_NAME,
&config,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

    if (config) {
        struct list_head *pos = NULL;
        list_for_each(pos, &config->node){
            trigger_configuration_t* tmp = list_entry(pos,struct
trigger_configuration_t,node);
            printf("triggerName: %s, isDefault: %s\n",
tmp->triggerName.data, tmp->isDefault.data);
            PRINT_OOS_STRING(remontEndPoint, tmp->remontEndPoint);
            PRINT_OOS_STRING(replicaMode, tmp->replicaMode);
            PRINT_OOS_STRING(remoteBucketName, tmp->remoteBucketName);
            PRINT_OOS_STRING(remoteAK, tmp->remoteAK);
            PRINT_OOS_STRING(remoteSK, tmp->remoteSK);
        }
    }
}
```

DELETE Bucket Trigger

在 DELETE 操作的 url 中加上 trigger，可以删除 Bucket 中的指定 trigger。

- 示例代码

```
void delete_bucket_trigger_test(request_options_t *options,oos_status_t
*status){
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, "oos-
js.ctyunapi.cn");

    status = oos_delete_bucket_trigger(options, BUCKET_NAME,
"testtrigger",&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
```

```
    }
}
```

PUT Bucket Lifecycle

存储在 OOS 中的对象有时需要有生命周期。比如，用户可能上传了一些周期性的日志文件到 Bucket 中，一段时间后，用户可能不需要这些日志对象了。

- 示例代码

```
void put_bucket_lifecycle_test(request_options_t *options, oos_status_t
*status) {
    lifecycle_configuration_t* config = NULL;
    config = oos_pcalloc(options->pool, sizeof(*config));
    if (!config) {
        printf("%s[%d]: oos_pcalloc failed.\n", __FUNCTION__, __LINE__);
        return;
    }

    oos_str_set(options->pool,  &config->ruleid,"111");
    oos_str_set(options->pool,  &config->prefix,"www1/");
    oos_str_set(options->pool,  &config->status,"Enabled");
    //oos_str_set(options->pool,  &config->date,"2011");
    config->days=10;

    lifecycle_configuration_t* config1 = NULL;
    config1 = oos_pcalloc(options->pool, sizeof(*config1));
    if (!config1) {
        printf("%s[%d]: oos_pcalloc failed.\n", __FUNCTION__, __LINE__);
        return;
    }

    oos_str_set(options->pool,  &config1->ruleid,"222");
    oos_str_set(options->pool,  &config1->prefix,"www2/");
    oos_str_set(options->pool,  &config1->status,"Disabled");
    oos_str_set(options->pool,  &config1->date,"2020-10-
23T00:00:00.000Z");
    //config->days=100;

    lifecycle_configuration_t configs[2];
```

```
    // memcpy((char *)&configs[0],(char *)&config,sizeof(config));
    memcpy((char *)&configs[0],(char *)config,sizeof(*config));
    memcpy((char *)&configs[1],(char *)config1,sizeof(*config));
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_put_bucket_lifecycle(options,"test-oos5-
js0",configs,2,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service error msg %s\n", status->error_msg);
    }

}
```

GET Bucket Lifecycle

此接口用来返回配置的 Bucket 生命周期。

- 示例代码

```
void get_bucket_lifecycle_test(request_options_t *options, oos_status_t
*status) {
    lifecycle_configuration_t* config=NULL;
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    status = oos_get_bucket_lifecycle(options, "test-oos5-js0",
&config,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }

    if(config) {
        struct list_head *pos = NULL;
        list_for_each(pos, &config->node) {
            lifecycle_configuration_t* tmp = list_entry(pos,struct
lifecycle_configuration_t,node);
```

```
            printf("ruleid: %s, prefix: %s status:%s day:%d date:%s\n",
tmp->ruleid.data, tmp->prefix.data, tmp->status.data, tmp->days,
tmp->date.data);
        }
    }

}
```

DELETE Bucket Lifecycle

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

- 示例代码

```
void delete_bucket_lifecycle_test(request_options_t
*options,oos_status_t *status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_bucket_lifecycle(options, "test-oos5-js0",
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
}
```

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。

- 示例代码

```
void put_bucket_cors_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
```

```
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    // make location config
    struct cors_configuration_t* config = NULL;
    config = oos_pcalloc(options->pool, sizeof(*config));
    if (!config) {
        printf("%s[%d]: oos_pcalloc failed \n", __FUNCTION__, __LINE__);
        return;
    }

    INIT_LIST_HEAD(&config->cors_rule_list.node);

    // add cors rule1
    list_cors_rule_node_t* cors_rule_node = NULL;
    cors_rule_node = oos_pcalloc(options->pool,
sizeof(*cors_rule_node));
    if (!cors_rule_node) {
        printf("%s[%d]: oos_pcalloc failed \n", __FUNCTION__, __LINE__);
        return;
    }

    INIT_LIST_HEAD(&cors_rule_node->node);
    INIT_LIST_HEAD(&cors_rule_node->allowed_header_list.node);
    INIT_LIST_HEAD(&cors_rule_node->allowed_method_list.node);
    INIT_LIST_HEAD(&cors_rule_node->allowed_origin_list.node);
    INIT_LIST_HEAD(&cors_rule_node->expose_header_list.node);

    // add id
    oos_str_set(options->pool,  &cors_rule_node->id,"111");
    // add allowed orign1 orign2
    add_list_string_node(options, "www.test21.com",
&cors_rule_node->allowed_origin_list);
    add_list_string_node(options, "www.test22.com",
&cors_rule_node->allowed_origin_list);
    // add allowed method1 method2
    add_list_string_node(options, "PUT",
&cors_rule_node->allowed_method_list);
    add_list_string_node(options, "GET",
&cors_rule_node->allowed_method_list);
```

```
    // add allowed headers1 headers2
    add_list_string_node(options, "x-amz*",
&cors_rule_node->allowed_header_list);
    add_list_string_node(options, "*",
&cors_rule_node->allowed_header_list);
    // add expose_header1 expose_header2
    add_list_string_node(options, "test11",
&cors_rule_node->expose_header_list);
    add_list_string_node(options, "test12",
&cors_rule_node->expose_header_list);

    cors_rule_node->maxage_seconds = 10;
    list_add_tail(&cors_rule_node->node, &config->cors_rule_list.node);

    status = oos_put_bucket_cors(options,"test-oos5-js0", config,
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
}
```

GET Bucket cors

此操作用来返回 Bucket 的跨域配置信息。

- 示例代码

```
void get_bucket_cors_test(request_options_t *options, oos_status_t
*status) {
    cors_configuration_t* config = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_get_bucket_cors(options, "test-oos5-js0",
&config,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
```

```
    char result[20480+1] = {"\0"};
    strcat(result, "cors result:\n");

    if(config) {
        list_cors_rule_node_t* cors_rule_node_for;
        list_for_each_entry(cors_rule_node_for,
&config->cors_rule_list.node, node) {
            list_cors_rule_node_t* cors_rule_node = cors_rule_node_for;

            strcat(result, "====================================\n");
            // ID
            strcat(result, "rule ID:");

            if(cors_rule_node->id.len > 0) {
                strcat(result, cors_rule_node->id.data);
            }

            // MaxageSecond
            strcat(result, " MaxageSecond:");

            if(cors_rule_node->maxage_seconds > 0) {
                char max_age[64] = {"\0"};
                sprintf(max_age, "%d\n",
cors_rule_node->maxage_seconds);
                strcat(result, max_age);
            }

            // AllowedOrign
            list_string_node_t* tmp_node;
            list_for_each_entry(tmp_node,
&cors_rule_node->allowed_origin_list.node, node) {
                char tmp[128+1] = {"\0"};
                sprintf(tmp, "AllowedOrign:%s\n", tmp_node->text.data);
                strcat(result, tmp);
            }
            // AllowedMethod
            tmp_node = NULL;
            list_for_each_entry(tmp_node,
&cors_rule_node->allowed_method_list.node, node) {
```

```
                char tmp[128+1] = {"\0"};
                sprintf(tmp, "AllowedMethod:%s\n", tmp_node->text.data);
                strcat(result, tmp);
            }
            // AllowedHeader
            tmp_node = NULL;
            list_for_each_entry(tmp_node,
&cors_rule_node->allowed_header_list.node, node) {
                char tmp[128+1] = {"\0"};
                sprintf(tmp, "AllowedHeader:%s\n", tmp_node->text.data);
                strcat(result, tmp);
            }
            //ExposeHeader
            tmp_node = NULL;
            list_for_each_entry(tmp_node,
&cors_rule_node->expose_header_list.node, node) {
                char tmp[128+1] = {"\0"};
                sprintf(tmp, "ExposeHeader:%s\n", tmp_node->text.data);
                strcat(result, tmp);
            }

        }
    }
    printf("==============\nresult:%s\n", result);
}
```

DELETE Bucket cors

删除 Bucket 的跨域配置信息。

- 示例代码

```
void delete_bucket_cors_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_bucket_cors(options, "test-oos5-
js0",&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
```

```
    }
}
```

关于 Object 的操作

PUT Object

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

- 示例代码

```
void put_buf_object_test(request_options_t *options,oos_status_t
*status) {_table_make(options->pool, 1);
    oos_table_t* medadata = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    //oos_table_set(medadata, "x-amz-meta-media_key1", "value1");
    oos_table_set(medadata, "x-ctyun-data-location",
"type=Specified,location=ShenZhen,QingDao,scheduleStrategy=Allowed");

    status = oos_put_object_from_buffer(options,"test-oos5-
js0","your_object1","hello world!111",strlen("hello world!111"),
medadata,&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service msg %s\n", status->error_msg);
    }

    int pos = 0;
    const oos_array_header_t *tarr = NULL;
    const oos_table_entry_t *telts = NULL;
    if (resp_headers) {
        tarr = oos_table_elts(resp_headers);
        telts = (oos_table_entry_t*)tarr->elts;

        for (pos = 0; pos < tarr->nelts; ++pos) {
            printf("--!- %s:%s\n",telts[pos].key, telts[pos].val);
        }
    }
}
```

GET Object

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

- 示例代码

```
void get_buf_object_test(request_options_t *options,oos_status_t
*status) {
    oos_string_t* content = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_table_t* req_header = oos_table_make(options->pool, 1);
    int pos = 0;
    const oos_array_header_t *tarr = NULL;
    const oos_table_entry_t *telts = NULL;
    oos_table_t* medadata = NULL;

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    content = (oos_string_t*)oos_pcalloc(options->pool,
sizeof(*content));
    if (!content) {
        printf("%s[%d]: oos_pcalloc failed. \n", __FUNCTION__,
__LINE__);
        return;
    }

    //oos_table_set(req_header, "Range", "1-3");
    status = oos_get_object_to_buffer_ex(options,"test-oos5-
js0","your_object1", req_header, content, &medadata, &resp_headers);

    if (medadata) {
        tarr = oos_table_elts(medadata);
        telts = (oos_table_entry_t*)tarr->elts;

        for (pos = 0; pos < tarr->nelts; ++pos) {
            printf("--%s: %s\n",telts[pos].key, telts[pos].val);
        }
    }
    if(resp_headers) {
        tarr = oos_table_elts(resp_headers);
```

```
        telts = (oos_table_entry_t*)tarr->elts;

        for (pos = 0; pos < tarr->nelts; ++pos) {
            printf("--%s: %s\n",telts[pos].key, telts[pos].val);
        }
    }
    if (status) {
        printf("service code %d\n", status->code);
        printf("service msg %s\n", status->error_msg);
    }
    printf("content: %s\n", content->data);
}
```

DELETE Object

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

- 示例代码

```
void delete_object_test(request_options_t *options,oos_status_t *status)
{
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_object(options,"test-oos5-
js0","your_object_file",&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
}
```

PUT Object - Copy

此操作用来创建一个存储在 OOS 里的对象的拷贝。此操作类似于执行一个 GET 然后在执行一次 PUT。

- 示例代码

```
void copy_object_test(request_options_t *options,oos_status_t *status) {
    copy_object_result_t* result = NULL;
    oos_table_t* req_headers = oos_table_make(options->pool, 1);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
```

```
    // oos_table_set(req_headers, "x-amz-meta-media_key1", "value1");
    // oos_table_set(req_headers, "x-amz-meta-media_key2", "value2");
    // oos_table_set(req_headers, "x-amz-copy-source-if-modified-since",
"Fri, 28 Apr 2019 07:58:00 GMT");
    // oos_table_set(req_headers, "x-amz-copy-source-if-unmodified-
since", "Mon, 13 May 2019 07:58:00 GMT");
    // oos_table_set(req_headers, "x-amz-copy-source-if-none-match",
"c84c314e2d5696bad88669cf517f6852");
    // oos_table_set(req_headers, "x-amz-meta-v1", "testv1");
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_copy_object(options, "test-oos5-js0", "your_object1",
"test-oos5-js0", "your_object1_copy", &result, req_headers,
&resp_headers);

    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
    if (result) {
        printf("etag: %s\n", result->etag.data);
        printf("lastModifiedDate: %s\n", result->lastModifiedDate.data);
    }
}
```

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。当使用非对象存储网络时，不能够指定对象的存储 location。

- 示例代码

```
void init_multipart_upload_test(request_options_t *options,oos_status_t
*status) {
    initiate_multipart_upload_result_t* result;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
```

```
    oos_table_t* medadata = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    oos_table_set(medadata, "x-amz-meta-media_key1", "value1");
    oos_table_set(medadata, "x-amz-meta-media_key2", "value2");
    oos_table_set(medadata, "x-ctyun-data-location",
"type=Local,location=QingDao,scheduleStrategy=NotAllowed");

    status = oos_init_multipart_upload(options, "test-oos5-
bucket0","test-multipart.txt", &result, medadata, &resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
    if (result)
        printf("upload id %s\n", result->uploadId.data);
}
```

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。

```
void upload_part_test(request_options_t *options, oos_status_t *status)
{
    FILE* file = NULL;
    size_t file_size = 0;
    size_t part_size = 5*1024*1024;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_string_t* etag = (oos_string_t*)oos_pcalloc(options->pool,
sizeof(*etag));
    if (!etag) {
        printf("%s[%d]: oos_pcalloc failed. \n", __FUNCTION__,
__LINE__);
        return;
    }

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    file = fopen("/root/wuyq/test.txt", "rb");
    if (file == NULL) {
        fprintf(stderr, "open file %s failed : %m\n",
"/root/wuyq/test.txt");
        return ;
    }

    fseek(file, 0, SEEK_END);
    file_size = ftell(file);
    fseek(file, 0, SEEK_SET);
    char* buff;
    buff = (char*)oos_pcalloc(options->pool, part_size+1);
    if (!buff) {
        printf("%s[%d]: oos_pcalloc failed. \n", __FUNCTION__,
__LINE__);
        return;
    }

    int a = fread(buff, 1, part_size, file);

    status = oos_upload_part_from_buffer(options, "test-oos5-bucket0",
"test-multipart.txt","1603792616279271487",1, buff, a, etag,
&resp_headers);
```

```
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }

    if (etag)
        printf("====== fist part complete : etag - %s\n", etag->data);

    //第二片
    resp_headers = oos_table_make(options->pool, 1);
    fseek(file, part_size, SEEK_SET);
    buff = (char*)oos_pcalloc(options->pool, part_size+1);
    if (!buff) {
        printf("%s[%d]: oos_pcalloc failed. \n", __FUNCTION__,
__LINE__);
        return;
    }
    int n = fread(buff, 1, part_size, file);
    status = oos_upload_part_from_buffer(options, "test-oos5-bucket0",
"test-multipart.txt","1603792616279271487",2, buff, n, etag,
&resp_headers);
    if (etag)
        printf("====== second part complete : etag - %s\n", etag->data);

    fclose(file);
}
```

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。

- 示例代码

```
void complete_multipart_upload_test(request_options_t
*options,oos_status_t *status) {
    part_etags* tag1 = NULL;
    part_etags tags[3];
    tag1 = oos_pcalloc(options->pool,sizeof(*tag1));
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    tag1->partnumber =1;
    oos_str_set(options->pool,  &tag1→etag,
"356f237344fec11ff1cf9adf058e597c");
    memcpy((char *)&tags[0],(char *)tag1,sizeof(*tag1));

    part_etags* tag2 = NULL;
    tag2 = oos_pcalloc(options->pool,sizeof(*tag2));
    tag2->partnumber =2;
    oos_str_set(options->pool,  &tag2→etag,
"75a6adb826df33b49eeee50679268b18");
    memcpy((char *)&tags[1],(char *)tag2,sizeof(*tag2));

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    status = oos_complete_multipart_upload(options,"test-oos5-
bucket0","test-multipart.txt", "1603792616279271487", tags, 2,
&resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }

}
```

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

```
void abort_multipart_upload_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    status = oos_abort_multipart_upload(options, "test-oos5-bucket0",
"test-multipart.txt", "1604538379099999104", &resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
}
```

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。

- 示例代码

```
void list_parts_test(request_options_t *options,oos_status_t *status) {
    part_listing_t* listing = NULL;
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_list_parts(options,"test-oos5-bucket0", "test-
multipart.txt", "1603792616279271487","4", NULL, &listing,
&resp_headers);

    if(listing) {
        struct list_head *pos = NULL;
        list_for_each(pos, listing->root) {
            part_summary_t* temp = list_entry(pos,struct
part_summary_t,node);
```

```
            printf("0000000 id %s\n", temp->eTag.data);
        }
    }
}
```

Copy Part

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

- 示例代码

```
void copy_part_test(request_options_t *options, oos_status_t *status) {
    char InitUploadID[100] = {"\0"};
    strcpy(InitUploadID, "1603792616279271487");
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    oos_string_t* etag1 = (oos_string_t*)oos_pcalloc(options->pool,
sizeof(*etag1));
    oos_table_t* req_headers1 = oos_table_make(options->pool, 1);
    oos_table_t* resp_headers1 = oos_table_make(options->pool, 1);
    //oos_table_set(req_headers1, "x-amz-copy-source-if-modified-since",
"Fri, 28 Apr 2019 07:58:00 GMT");
    //oos_table_set(req_headers1, "x-amz-copy-source-if-unmodified-
since", "Fri, 28 Apr 2019 07:58:00 GMT");
    //oos_table_set(req_headers1, "x-amz-copy-source-if-match",
"fc3ff98e8c6a0d3087d515c0473f8677====");
    oos_table_set(req_headers1, "x-amz-copy-source-if-none-match",
"fc3ff98e8c6a0d3087d515c0473f8677======");

    status = oos_copy_part(options,"test-oos5-bucket0","test-
multipart.txt",InitUploadID, 3, "test-oos5-bucket0", "test.txt",
req_headers1, etag1, &resp_headers1);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
    if (etag1)
        printf("====== fist part complete : etag1 - %s\n", etag1->data);
}
```

断点续传

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

- 示例代码

```
void upload_part_break_point_test(request_options_t
*options,oos_status_t *status) {
    oos_table_t* resp_headers = NULL;

    oos_str_set(options->pool,  &options->access_key, ACCESS_KEY);
    oos_str_set(options->pool,  &options->secret_key, SECRET_KEY);
    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_upload_part_break_point(options, BUCKET_NAME, "CentOS-
7-x86_64-DVD-2003.iso",
                        "/path/to/your/file", 50, NULL, 5*1024*1024,
OOS_FALSE, &resp_headers);

    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n", status->code,
status->error_msg);
    }
}
```

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。

- 示例代码

```
void delete_multiple_object_test(request_options_t *options,oos_status_t
*status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    delete_multiple_list_t* list = oos_pcalloc(options->pool,
sizeof(*list));
    if (!list) {
```

```
        printf("%s[%d]: oos_pcalloc failed. \n", __FUNCTION__,
__LINE__);
        return;
    }
    INIT_LIST_HEAD(&list->delete_list.node);
    add_list_object_node_pool(options->pool, "your_object1",
&list->delete_list);
    add_list_object_node_pool(options->pool, "your_object1_copy",
&list->delete_list);

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);

    status = oos_delete_multiple_object(options,"test-oos5-
target",list,OOS_TRUE,&resp_headers);

    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);

        if(status->code < 300) {
            char result[2048] = {"\0"};
            list_delete_node_t* tmp = NULL;

            if(!list_empty(&list->delete_list.node)) {
                list_for_each_entry(tmp, &list->delete_list.node, node)
{
                    char object_result[64] = {"\0"};
                    strcat(object_result, tmp->object_key.data);
                    strcat(object_result, ":");
                    strcat(object_result, tmp->result.data);
                    strcat(object_result, "\n");
                    strcat(result, object_result);
                }
                printf("==============result:\n%s\n", result);
            }
        }
    }
}
```

生成共享链接

对于私有或只读 Bucket，可以通过生成 Object 的共享链接的方式，将 Object 分享给其他人，同时可以在链接中设置限速以对下载速度进行控制。

- **示例代码**

```
void genrate_object_presignedurl_test(request_options_t
*options,oos_status_t *status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    char* url_result = NULL;
    int expires = 3600;
    int pos = 0;
    const oos_array_header_t *tarr = NULL;
    const oos_table_entry_t *telts = NULL;

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    status = oos_object_generate_presigned_url(options,"test-oos5-
bucket0","your_object1", expires, &resp_headers, &url_result);

    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }

    if (url_result)
        printf("result url:%s\n", url_result);

    if (resp_headers) {
        tarr = oos_table_elts(resp_headers);
        telts = (oos_table_entry_t*)tarr->elts;

        for (pos = 0; pos < tarr->nelts; ++pos) {
            printf("--!- %s:%s\n",telts[pos].key, telts[pos].val);
        }
    }
}
```

HEAD Object

此操作用来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

```
void head_object_test(request_options_t *options,oos_status_t *status) {
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    oos_table_t* req_header = oos_table_make(options->pool, 1);

    oos_str_set(options->pool,  &options->endpoint, HOST_NAME);
    oos_table_set(req_header, "Range", "bytes=1-30000");
    // oos_table_set(req_header, "If-Match",
"fc3ff98e8c6a0d3087d515c0473f8677");
    oos_table_set(req_header, "If-None-Match",
"fc3ff98e8c6a0d3087d515c0473f8677");
    oos_table_set(req_header, "If-Modified-Since", "2019-05-
01T09:12:43.083Z");
    oos_table_set(req_header, "If-Unmodified-Since", "2019-05-
01T09:12:43.083Z");

    status = oos_head_object(options, "test-oos5-js0","your_object1",
req_header, &resp_headers);
    if (status) {
        printf("service code %d\n", status->code);
        printf("service err msg %s\n", status->error_msg);
    }
}
```

关于 AccessKey 的操作

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，每个账户最多创建 10 个 AccessKey。

- 示例代码

```
void create_ctyun_access_key_test(request_options_t
*options,oos_status_t *status) {
    oos_iam_result_t *iam_result = NULL;
    oos_str_set(options->pool, &options->endpoint, HOST_NAME_IAM);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);
    status = oos_ctyun_create_access_key(options, "user1", NULL,
&iam_result, &resp_headers);

    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n", status->code,
status->error_msg);
    }
    if (iam_result && iam_result->root) {
        struct list_head *pos = NULL;
        PRINT_OOS_STRING(requestId, iam_result->requestId);
        list_for_each(pos, iam_result->root) {
            oos_iam_access_key_t *temp = list_entry(pos, struct
oos_iam_access_key_t, node);
            PRINT_OOS_STRING(AccessKeyId, temp->accessKeyId);
            PRINT_OOS_STRING(userName, temp->userName);
            PRINT_OOS_STRING(Status, temp->status);
            PRINT_OOS_STRING(SecretAccessKey, temp->secretAccessKey);
            PRINT_OOS_STRING(createDate, temp->createDate);
        }
    }
```

```
}
```

DeleteAccessKey

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

- 示例代码

```
void delete_ctyun_user_access_key_test(request_options_t
*options,oos_status_t *status) {
    oos_iam_result_t *iam_result = NULL;
    oos_str_set(options->pool, &options->endpoint, HOST_NAME_IAM);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    status = oos_ctyun_delete_user_access_key(options, "user1", NULL,
"9daad04cee4e9bc234ff", &iam_result, &resp_headers);

    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n", status->code,
status->error_msg);
    }

    if (iam_result) {
        PRINT_OOS_STRING(requestId, iam_result->requestId);
    }
   }
```

UpdateAccessKey

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

- 示例代码

```
void update_ctyun_user_access_key_test(request_options_t
*options,oos_status_t *status) {
```

```
    oos_iam_result_t *iam_result = NULL;
    oos_str_set(options->pool, &options->endpoint, HOST_NAME_IAM);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    status = oos_ctyun_update_user_access_key(options, "user1", NULL,
"9daad04cee4e9bc234ff", key_inactive, OOS_TRUE, &iam_result,
&resp_headers);

    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n", status->code,
status->error_msg);
    }

    if (iam_result) {
        PRINT_OOS_STRING(requestId, iam_result->requestId);
    }
    }
```

ListAccessKey

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

- 示例代码

```
void list_ctyun_user_access_key_test(request_options_t
*options,oos_status_t *status) {
    oos_iam_result_t *iam_result = NULL;
    oos_str_set(options->pool, &options->endpoint, HOST_NAME_IAM);
    oos_table_t* resp_headers = oos_table_make(options->pool, 1);

    status = oos_ctyun_list_user_access_keys(options, "user1", NULL,
NULL, 4, &iam_result, &resp_headers);
```

```
    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n", status->code,
status->error_msg);
    }

    if (iam_result && iam_result->root) {
        struct list_head *pos = NULL;
        PRINT_OOS_STRING(requestId, iam_result->requestId);
        PRINT_OOS_STRING(marker, iam_result->marker);
        printf("0000 isTruncated: %d\n", iam_result->isTruncated);
        list_for_each(pos, iam_result->root) {
            printf("=======================================\n");
            oos_iam_access_key_t *temp = list_entry(pos, struct
oos_iam_access_key_t, node);
            PRINT_OOS_STRING(AccessKeyId, temp->accessKeyId);
            PRINT_OOS_STRING(userName, temp->userName);
            PRINT_OOS_STRING(Status, temp->status);
            PRINT_OOS_STRING(createDate, temp->createDate);
            printf("0000 isPrimary: %d\n", temp->isPrimary);
        }
    }
    }
```

STS 临时授权访问

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

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

- **示例代码**

```
void get_ctyun_security_token_service_test(request_options_t
*options,oos_status_t *status) {
    oos_iam_result_t *iam_result = NULL;
    oos_str_set(options->pool,  &options->access_key,
ACCESS_KEY);
    oos_str_set(options->pool,  &options->secret_key,
SECRET_KEY);
    oos_str_set(options->pool, &options->endpoint,
HOST_NAME_IAM);
    oos_table_t* resp_headers =
oos_table_make(options->pool, 1);

    status = oos_ctyun_get_security_token_service(options,
900, &iam_result, &resp_headers);

    if (status) {
        printf("status %p\n", status);
        printf("service code %d, errmsg: %s\n",
status->code, status->error_msg);
    }

    if (iam_result && iam_result->root) {
        struct list_head *pos = NULL;
        PRINT_OOS_STRING(requestId, iam_result->requestId);
        list_for_each(pos, iam_result->root) {
            oos_iam_security_token_t *temp = list_entry(pos,
struct oos_iam_security_token_t, node);
```

```
            PRINT_OOS_STRING(sessionToken,
temp->sessionToken);
            PRINT_OOS_STRING(accessKeyId,
temp->accessKeyId);
            PRINT_OOS_STRING(secretAccessKey,
temp->secretAccessKey);
            PRINT_OOS_STRING(expiration, temp->expiration);
        }
    }
}
```