c linux sdk 示例文档

环境配置

s3 c sdk需要自己在环境中进行编译，故确保您的linux环境中包含以下基础组件

- GCC 4.8 或更新版本
- 以下库及其相关头文件，可在对应linux发行版的包管理器中安装，括号中给出的是经过验证的版本
    - libxml2 (2.9.1-6.el7)
    - libcurl (7.29.0-59.el7)
    - openssl (1.0.2k-26.el7_9)

如您的环境中没有这三个依赖，可直接通过包管理工具直接进行安装

```
sudo yum install -y libxml2*
sudo yum install -y libcurl*
sudo yum install openssl openssl-devel
```

初始化sdk

获取AccessKey, SecretAccessKey和Endpoint

可从自己创建的天翼云对象存储控制台获取

创建工程

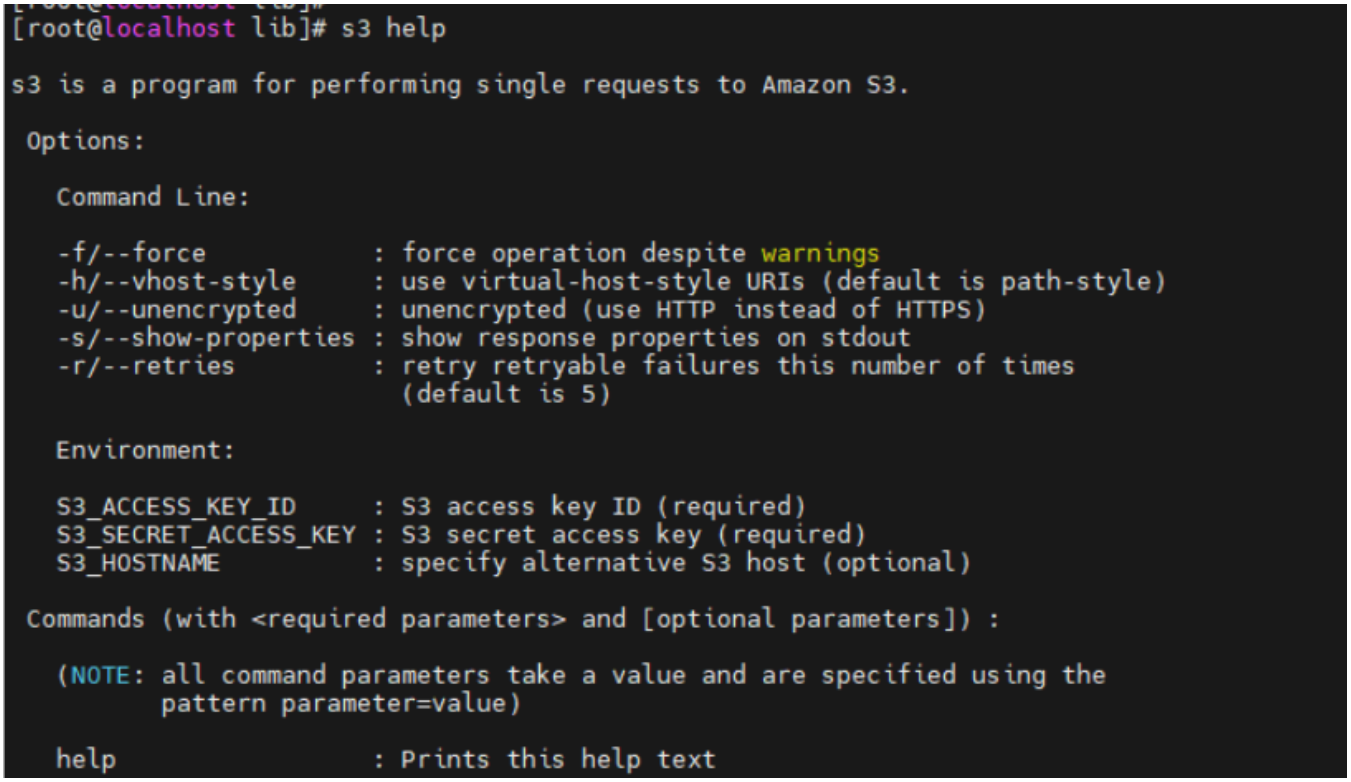
c sdk 名字为libs3，目前仅支持从天翼云官网下载libs3源代码压缩包。

代码下载成功以后，进入libs3 目录下，执行

```
make clean & make install
```

编译成功后，在libs3/build/bin/目录下会生产一个s3二进制文件。可直接执行验证环境是否验证

成功,s3能正常执行的打印如下：



- tips：执行s3二进制文件执行时报 "error while loading shared libraries: libs3.so.2：cannot open shared object file xxxxxxxxxxxxxxxx" 类的报错。
    - 解决方案：在明确已经安装该库的情况下还是报这个错误，可能是因为这个库目录没有添加到环境变量中，解决方案如下：
        - 如果编译成功了，在libs3/build/lib/目前下有一个 libs3.so.2 文件，将该文件cp一份到 /usr/local/lib目录下
        - 将动态库文件加入配置中：执行vi /etc/ld.so.conf，在最后一行增加 '/usr/local/lib'
        - 保存后，在命令行终端执行

```
/sbin/ldconfig -v
ldconfig
```

本文后续所有的用例全部在libs3/src/testsimplexml.c文件中，编译时会在build/bin/目录下会生成一个api_s3可执行文件，您如果想测试后续的接口，也可以在libs3/src/testsimplexml.c文件中进行修改验证。

设置service client

使用c sdk访问对象存储，需要配置您的AK，SK以及host信息到环境变量中，在Linux环境下，直接执行以下代码

```
    export S3_HOSTNAME="" /* 你的host网址 */
    export S3_ACCESS_KEY_ID="" /* 你的AK */
    export S3_SECRET_ACCESS_KEY="" /* 你的SK */
```

签名相关

libs3 c sdk支持aws v2和v4签名算法，您可以在调用接口前执行以下函数:

```
    setRequestSignVersion(S3_SIGN_VERSION_V2);
```

其中, S3_SIGN_VERSION_V2对应枚举变量:

```
    typedef enum {
    S3_SIGN_VERSION_V2, /* 对应v2签名算法 */
    S3_SIGN_VERSION_V4, /* 对应v4签名算法 */
    S3_SIGN_VERSION_NR
} S3SignVersion;
```

示例

```c
int list_bucket_test(int argc, char **argv)
{
    if (argc || argv) {
        /* do noting */
    }
    list_service_data data;
    int s3_status = 0;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    S3Protocol protocolG = S3ProtocolHTTP;
    int allDetails = 0;

    data.headerPrinted = 0;
    data.allDetails = allDetails;

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }

    /* 2. 定义回调函数 */
    S3ListServiceHandler listServiceHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &listServiceCallback
    };

    /* 指定签名算法版本 */
    setRequestSignVersion(S3_SIGN_VERSION_V2);

    /* 3. 创建list Bucket请求 */
    S3_list_service(protocolG, access_key, secret_key, 0, 0,
                        &listServiceHandler, &data);
    s3_status = get_statusG();

    /* 4. 返回成功，打印bucket 信息 */
```

```
    if (s3_status == S3StatusOK) {
        if (!data.headerPrinted) {
            printListServiceHeader(allDetails);
        }
    }

    S3_deinitialize();

    return s3_status;
}
```

桶操作

创建桶

CreateBucket

CreateBucket 操作用于创建桶. 每个用户可以拥有多个桶, 创建者默认为桶的拥有者, 对桶拥有 FULL_CONTROL 权限。

api接口展示

```
void S3_create_bucket(S3Protocol protocol, const char *accessKeyId,
                      const char *secretAccessKey, const char *hostName,
                      const char *bucketName, S3CannedAcl cannedAcl,
                      const char *locationConstraint,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
protocol	S3Protocol	访问协议	是
accessKeyId	const char *	访问账户的AK	是

参数	类型	说明	是否必要
secretAccessKey	const char *	访问账户的SK	是
hostName	const char *	访问的网址	否
bucketName	const char *	需要创建的bucket名称	是
cannedAcl	S3CannedAcl	创建桶的acl信息	否
locationConstraint	const char *	桶的位置信息	否
requestContext	S3RequestContext *	提供的请求信息，如没有，可填写为0	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidBucketName	创建桶的名称不合法
403	AccessDenied	用户没有权限执行操作
409	BucketAlreadyExists	当前创建的桶的名字已经被其他用户使用

完整代码展示

```c
#include <stdio.h>
#include <stdlib.h>
#include "simplexml.h"

int create_bucket(int argc, char **argv)
{
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *locationConstraint = 0;
    S3Protocol protocolG = S3ProtocolHTTP;
    S3Status status;
    if (argc < 2) {
        printf("please input your bucket name\n");
        return -1;
    }
    bucket = argv[1];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }
    /* 2. s3 response handle */
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback, &responseCompleteCallback
    };
    /* 3. create bucket req */
    S3_create_bucket(protocolG, access_key, secret_key,
        0, bucket, S3CannedAclPrivate, locationConstraint, 0,
        &responseHandler, 0);
    if (get_statusG() == S3StatusOK) {
        printf("Bucket %s successfully created. \n", bucket);
    } else {
       // printError();
    }
    S3_deinitialize();
    return get_statusG();
```

```
}

int main(int argc, char **argv)
{
    create_bucket(argc, argv);

    return 0;
}
```

完整的代码示例即是用main函数调用创建桶的函数接口，完整的代码读者可参考以上代码，后续将不再展示完整代码，仅展示API接口如何进行调用的demo。

获取桶列表

ListBuckets

ListBuckets 操作用于显示可用的桶

api接口展示

```
void S3_list_service(S3Protocol protocol, const char *accessKeyId,
                     const char *secretAccessKey, const char *hostName,
                     S3RequestContext *requestContext,
                     const S3ListServiceHandler *handler, void *callbackData)
```

API接口参数说明

参数	类型	说明	是否必要
protocol	S3Protocol	访问协议	是
accessKeyId	const char *	访问账户的AK	是
secretAccessKey	const char *	访问账户的SK	是
hostName	const char *	访问的网址	是
requestContext	S3RequestContext *	提供的请求信息，如没有，可填写为0	否

参数	类型	说明	是否必要
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作

代码示例

```c
int list_bucket_test(void)
{
    list_service_data data;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    S3Status status;
    S3Protocol protocolG = S3ProtocolHTTP;
    int allDetails = 0;
    data.headerPrinted = 0;
    data.allDetails = allDetails;
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }
    /* 2. 定义回调函数 */
    S3ListServiceHandler listServiceHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &listServiceCallback
    };
    /* 3. 创建list Bucket请求 */
    S3_list_service(protocolG, access_key, secret_key, 0, 0,
                        &listServiceHandler, &data);
    s3_status = get_statusG();
    /* 4. 返回成功，打印bucket 信息 */
    if (s3_status == S3StatusOK) {
        if (!data.headerPrinted) {
            printListServiceHeader(allDetails);
        }
    }
    S3_deinitialize();
    return s3_status;
}
```

获取桶信息

HeadBuckets

HeadBucket 方法查询桶是否存在, 用户可以使用该操作查询一个桶是否存在且是否拥有对其访问权限.
执行 HeadBucket 操作需要具有执行 ListBucket 操作的权限, 桶的拥有者默认拥有该权限并且可以将该权限授予其他用户

api接口展示

```
void S3_test_bucket(S3Protocol protocol, S3UriStyle uriStyle,
                    const char *accessKeyId, const char *secretAccessKey,
                    const char *hostName, const char *bucketName,
                    int locationConstraintReturnSize,
                    char *locationConstraintReturn,
                    S3RequestContext *requestContext,
                    const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
protocol	S3Protocol	访问协议	是
uriStyle	S3UriStyle	访问的url style,可使用默认值	是
accessKeyId	const char *	访问账户的AK	是
secretAccessKey	const char *	访问账户的SK	是
hostName	const char *	访问的网址	是
bucketName	const char *	需要访问的bucket名称	是
locationConstraint ReturnSize	int	桶信息中携带的位置信息大小	是
locationConstraint Return	char *	桶位置信息	是

参数	类型	说明	是否必要
requestContext	S3RequestContext *	提供的请求信息，如没有，可填写为0	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	指定的桶不存在

代码示例

```c
int head_bucket(int argc, char **argv)
{
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *result = NULL;
    char locationConstraint[64];
    S3Protocol protocolG = S3ProtocolHTTP;
    S3Status status;
    int s3_status = 0;
    if (argc < 1) {
        printf("please input your bucket name\n");
        return -1;
    }
    bucket = argv[1];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }
    /* 2. 定义回调函数 */
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback, &responseCompleteCallback
    };
    /* 3. 创建headBucket请求 */
    S3_test_bucket(protocolG, S3UriStylePath, access_key, secret_key,
        0, bucket, sizeof(locationConstraint), locationConstraint, 0, &responseHandler, 0);
    s3_status = get_https_status();
    if (s3_status >= 200 && s3_status < 300) {
        result = "NORMAL";
    } else if (s3_status == 404) {
        result = "Does Not Exist";
    } else if (s3_status == 403) {
        result = "Access Denied";
    } else {
        result = 0;
    }
```

```
    if (result) {
        printf("%-56s  %-20s\n", "                          Bucket",
               "       Status");
        printf("------------------------------------------------------  "
               "--------------------\n");
        printf("%-56s  %-20s\n", bucket, result);
    }
    S3_deinitialize();
    return s3_status;
}
```

删除桶

DeleteBucket

DeleteBucket 操作用于删除桶. 删除一个桶前, 需要先删除该桶中的全部对象(包括对象的版本和删除标记)

api接口展示

```
void S3_delete_bucket(S3Protocol protocol, S3UriStyle uriStyle,
                      const char *accessKeyId, const char *secretAccessKey,
                      const char *hostName, const char *bucketName,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
protocol	S3Protocol	访问协议	是
uriStyle	S3UriStyle	url type，分为S3UriStyleVirtualHost以及S3UriStylePath, 根据你的请求url类型来填写	是
accessKeyId	const char *	访问账户的AK	是

参数	类型	说明	是否必要
secretAccessKey	const char *	访问账户的SK	是
hostName	const char *	访问的网址	否
bucketName	const char	需要删除的bucket名称	是
requestContext	S3Request Context *	提供的请求信息，如没有，可填写为0	否
handler	const S3Response Handler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidBucketName	创建桶的名称不合法
403	AccessDenied	用户没有权限执行操作
409	BucketAlreadyExists	当前创建的桶的名字已经被其他用户使用

代码示例

```c
int delete_bucket(int argc, char **argv)
{
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Protocol protocolG = S3ProtocolHTTP;
    S3Status status;
    if (argc < 2) {
        printf("please input your bucket name\n");
        return -1;
    }
    bucket = argv[1];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }
    /* 2. 定义回调函数 */
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback, &responseCompleteCallback
    };
    /* 3. 创建删除Bucket请求 */
    S3_delete_bucket(protocolG, S3UriStylePath, access_key, secret_key,
        0, bucket, 0, &responseHandler, 0);
    if (get_statusG() == S3StatusOK) {
        printf("Bucket %s successfully deleted. \n", bucket);
    }
    S3_deinitialize();
    return get_statusG();
}
```

设置桶访问权限

PutBucketAcl

PutBucketAcl 操作可以通过ACL设置一个桶的访问权限. 用户在设置桶的ACL之前需要具备WRITE_ACP权限

Bucket的权限说明:

权限类型	说明
READ	可以对桶进行list操作
READ_ACP	可以读取桶的ACL信息. 桶的拥有者默认具有桶的 READ_ACP 权限
WRITE	可以在桶中创建对象, 修改原有对象数据和删除对象
WRITE_ACP	可以修改桶的ACL信息, 授予该权限相当于授予 FULL_CONTROL 权限, 因为具有WRITE_ACP 权限的用户可以配置桶任意权限. 桶的拥有者默认具有桶的WRITE_ACP 权限
FULL_CONTROL	同时授予 READ , READ_ACP , WRITE 和 WRITE_ACP 权限

api接口展示

```
void S3_set_acl(const S3BucketContext *bucketContext, const char *key,
                const char *ownerId, const char *ownerDisplayName,
                int aclGrantCount, const S3AclGrant *aclGrants,
                S3RequestContext *requestContext,
                const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
key	const char *	将上传对象的名 设置桶ACL可不用填写	否
ownerId	const char *	操作者自己的id标识符, 可先通过get acl获取	是
ownerDisplayName	const char *	操作者自己的owner名称，可先通过get acl获取	是

参数	类型	说明	是否必要
aclGrantCount	int	返回S3AclGrant的个数	是
aclGrants	const S3AclGrant *	权限信息结构图指针	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

ownerId和ownerDisplayName可以通过下一小节的get_acl获取
响应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidArgument	参数错误
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	操作指定的桶不存在

代码示例

```
int set_bucket_acl(int argc, char **argv)
{
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    S3Status status;
    S3Protocol protocolG = S3ProtocolHTTPS;
    S3AclGrant aclGrants[2];
    const char *key = NULL;
    char *email = "nocompatible@outlook.com";
    const char *id = "21f1c2f7-ee85-5ba7-9355-dfc3c008eb21";
    const char *disPlayName = "胡豪";
    int aclGrantCount = 1, i = 0;
     if (argc < 2) {
        printf("please input your bucket name\n");
        return -1;
    }
    bucket = argv[1];
    /* 设置granteeType需要设置的权限 */
    aclGrants[0].granteeType = S3GranteeTypeAmazonCustomerByEmail;
    aclGrants[0].permission = S3PermissionRead;
    if (aclGrants[0].granteeType == S3GranteeTypeAmazonCustomerByEmail) {
        i = 0;
        while(email[i] != '\0') {
        aclGrants[0].grantee.amazonCustomerByEmail.emailAddress[i] = email[i];
        i++;
        }
    } else {
        /*  如果想要使用账号进行测试，请创建您自己的账户 */
        i = 0;
        while (id[i] != '\0') {
            aclGrants[0].grantee.canonicalUser.id[i] = id[i];
            i++;
        }
        i = 0;
        while(disPlayName[i] != '\0') {
            aclGrants[0].grantee.canonicalUser.displayName[i] = disPlayName[i];
```

```
            i++;
        }
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        exit(-1);
    }
    /* 2. 基本信息设置 */
    S3BucketContext bucketContext =
    {
        hostname,
        bucket,
        protocolG,
        S3UriStylePath,
        access_key,
        secret_key
    };
    /* 3. 回调信息设置 */
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    /* 4. 调用set acl接口 */
    S3_set_acl(&bucketContext, key, id, disPlayName,
        aclGrantCount, aclGrants, 0, &responseHandler, 0);
    s3_status = get_statusG();

    S3_deinitialize();
    return s3_status;
}
```

获取桶访问权限

GetBucketAcl

GetBucketAcl 操作可以让用户获取桶的 access control list (ACL)信息, 查询ACL信息需要用户具有该桶的 READ_ACP (读取桶的ACL信息)权限

api接口展示

```
void S3_get_acl(const S3BucketContext *bucketContext, const char *key,
                char *ownerId, char *ownerDisplayName,
                int *aclGrantCountReturn, S3AclGrant *aclGrants,
                S3RequestContext *requestContext,
                const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
key	const char *	将上传对象的名	否
ownerId	char *	获取出来桶所属的owner信息, 需要定义结构体, 调用接口前可不用赋值	是
ownerDisplay Name	char *	获取出来桶所属的owner Name信息, 需要定义结构体, 调用接口前可不用赋值	是
aclGrantCount Return	int *	返回S3AclGrant的个数, 调用接口前可不用赋值	是
aclGrants	S3AclGrant *	ack中返回的权限信息 结构图指针,调用接口前 可不用赋值	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例代码

```
int get_object_acl_test(int argc, char **argv)
{
    const char *bucket = NULL, *key = NULL; /* 要访问的bucket以及Key name */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* acl相关信息变量 */
    int aclGrantCount;
    S3AclGrant aclGrants[S3_MAX_ACL_GRANT_COUNT];
    char ownerId[S3_MAX_GRANTEE_USER_ID_SIZE];
    char ownerDisplayName[S3_MAX_GRANTEE_DISPLAY_NAME_SIZE];
    if (argc < 2) {
        printf("please inpur your bucket / key,so we can get it \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_acl(&bucketContext, key, ownerId, ownerDisplayName,
```

```
                &aclGrantCount, aclGrants, 0, &responseHandler, 0);
    s3_status = get_https_status();
    if (s3_status >= 200 && s3_status< 300) {
        printf( "OwnerID %s %s\n\n", ownerId, ownerDisplayName);
        printf("%-6s  %-90s  %-12s\n", " Type",
                "                                       User Identifier",
                " Permission");
        printf("------  "
                "------------------------------------------------------------"
                "------------------------------  ------------\n");
        int i;
        for (i = 0; i < aclGrantCount; i++) {
            S3AclGrant *grant = &(aclGrants[i]);
            const char *type;
            char composedId[S3_MAX_GRANTEE_USER_ID_SIZE +
                            S3_MAX_GRANTEE_DISPLAY_NAME_SIZE + 16];
            const char *id;
            switch (grant->granteeType) {
            case S3GranteeTypeAmazonCustomerByEmail:
                type = "Email";
                id = grant->grantee.amazonCustomerByEmail.emailAddress;
                break;
            case S3GranteeTypeCanonicalUser:
                type = "UserID";
                snprintf(composedId, sizeof(composedId),
                         "%s (%s)", grant->grantee.canonicalUser.id,
                         grant->grantee.canonicalUser.displayName);
                id = composedId;
                break;
            case S3GranteeTypeAllAwsUsers:
                type = "Group";
                id = "Authenticated AWS Users";
                break;
            case S3GranteeTypeAllUsers:
                type = "Group";
                id = "All Users";
                break;
            default:
                type = "Group";
```

```
                id = "Log Delivery";
                break;
            }
            const char *perm;
            switch (grant->permission) {
            case S3PermissionRead:
                perm = "READ";
                break;
            case S3PermissionWrite:
                perm = "WRITE";
                break;
            case S3PermissionReadACP:
                perm = "READ_ACP";
                break;
            case S3PermissionWriteACP:
                perm = "WRITE_ACP";
                break;
            default:
                perm = "FULL_CONTROL";
                break;
            }
            printf("%-6s  %-90s  %-12s\n", type, id, perm);
        }
    }
    else {
    //    printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

设置桶策略

桶策略(bucket policy)可以灵活地配置用户各种操作和访问资源的权限. ACL只能对单一对象设置权限, 而桶策略可以基于各种条件对一个桶内的全部或者一组对象配置权限. 桶的拥有者自带PutBucketPolicy操作权限. 若桶已被设置policy, 则新的policy将覆盖原有policy

PutBucketPolicy

PutBucketPolicy 操作用于设置桶策略, 描述桶策略的信息以JSON格式的字符串通过Policy参数传入.
一个可用的Policy参数如下方代码示例中的 policy 变量所示, 其中 Statement 的内容说明如下:

元素	描述	是否必要
Sid	即statement Id, 是对本条策略的描述	否
Principal	指定允许访问资源的主体, 支持用通配符 "* "表示所有用户(包括匿名用户)	可选, Principal和NotPrincipal 选其一
NotPrincipal	指定不允许访问资源的主体, 取值同Principal	可选, Principal和NotPrincipal 选其一
Action	描述将允许的特定操作, 支持用通配符"* "表示允许所有操作	可选, Action 与NotAction 其一
NotAction	描述将拒绝的特定操作, 取值同 Action	可选, Action 与NotAction 选其一
Effect	指定声明所产生的效果是"允许"还是"显式拒绝", 取值为"Allow"或"Deny"	必选
Resource	指定策略作用的一组资源, 支持用通配符"* "表示所有资源	可选, Resource与NotResource选其一
NotResource	指定策略不作用的一组资源, 策略将作用于除此之外的其他资源, 取值同Resource	可选, Resource与NotResource选其一
Conditon	指定策略生效的条件	可选

api接口

```
void S3_set_bucket_policy(const S3BucketContext *bucketContext,
                          const char *json_data, S3RequestContext *requestContext,
                          const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
json_data	const char *	需要传入的policy数据， 需要严格按照Json格式给出	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

相应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	ErrorInvalidArgument	用户给出的policy 数据不符合json格式或者参数配置设置有误
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int put_bucket_policy_test(void)
{
    const char *bucket = "bucket-tt-test"; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* 桶策略 数据，请按照c语言格式给出json数据，否则可能无法正确解析，按照以下示例赋值 */
    const char policy[] =  "{"
        "\"Version\": \"2012-10-17\","
        "\"Statement\":["
            "{"
                "\"Sid\": \"id-1\","
                "\"Effect\": \"Allow\","
                "\"Principal\":{\"AWS\":\"arn:aws:iam::*\"},"
                "\"Action\": [\"s3:PutObject\",\"s3:PutObjectAcl\"],"
                "\"Resource\": [\"arn:aws:s3:::bucket-tt-test/*\"]" /* 这里需要设置您需要访问的桶 */
            "}"
        "]"
    "}";
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
```

```
    };
    S3_set_bucket_policy(&bucketContext, policy, 0, &responseHandler, 0);
    s3_status = get_statusG();
    S3_deinitialize();
    return s3_status;
}
```

获取桶策略

GetBucketPolicy

GetBucketPolicy 操作用于获取桶的桶策略信息
api接口

```
void S3_get_bucket_policy(const S3BucketContext *bucketContext,
                          char *policy_data, int policy_data_len,
                          S3RequestContext *requestContext,
                          const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
policy_data	const char *	返回的Policy数据	是
policy_data_len	int	policy_data的长度	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在或者桶上不存在桶策略

示例

```c
int get_bucket_policy_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char bucket_policy[64 * 1024];
    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_bucket_policy(&bucketContext, bucket_policy,
        sizeof(bucket_policy), 0, &responseHandler, 0);
    s3_status = get_statusG();
    if (s3_status != S3StatusOK) {
       // printError();
    }
```

```
    S3_deinitialize();
    return s3_status;
}
```

删除桶策略

DeleteBucketPolicy

DeleteBucketPolicy 操作用于删除桶的桶策略信息, 桶的创建者默认具有删除桶策略的权限

api接口

```
void S3_delete_bucket_policy(const S3BucketContext *bucketContext,
                             S3RequestContext *requestContext,
                             const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

相应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在或者桶上不存在桶策略

示例

```
int delete_bucket_policy_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_delete_bucket_policy(&bucketContext, 0, &responseHandler, 0);
        s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
      //  printError();
    }
    S3_deinitialize();
```

```
    return s3_status;
}
```

设置版本控制

PutBucketVersioning

PutBucketVersioning 操作用于设置桶的版本控制状态. 共有以下两种状态:

- Enabled: 开启桶的版本控制, 之后每个添加到桶中的对象都会被设置一个唯一的 version ID
- Suspended: 关闭桶的版本控制, 之后每个添加到桶中的对象的vesrion ID都会被设置为null

api接口

```
void S3_set_bucket_version(const S3BucketContext *bucketContext,
                           const char *version, S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
version	const char *	需要设置的桶的版本开关值，只允许取值为 Suspended 和Enabled	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	ErrorInvalidArgument	用户设置的桶version参数信息有误
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	需要操作的桶不存在

示例

```c
int set_bucket_version_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* 桶版本控制信息 Suspended 关闭 Enabled 开启 */
    const char *version = NULL;
    if (argc < 2) {
        printf("please inpur your bucket && version \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    version = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_set_bucket_version(&bucketContext, version, 0, &responseHandler, 0);
    s3_status = get_statusG();
    S3_deinitialize();
    return s3_status;
```

```
}
```

获取版本控制信息

GetBucketVersioning

GetBucketVersioning 操作用于获取桶的版本控制状态信息, 只有桶的拥有者才能获取到桶的版本控制信息
桶的版本控制有三种状态: 未开启/开启(Enabled)/暂停(Suspended). 桶在被设置版本状态前默认为未开启状态, 执行 GetBucketVersioning 将看不到bucket的版本信息，即返回值中没有Status这一项

api接口

```
void S3_get_bucket_version(const S3BucketContext *bucketContext,
                           char *data, int data_len,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData);
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
data	char *	获取桶版本信息的返回信息	是
data_len	int	data的长度	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	需要操作的桶不存在

示例

```c
int get_bucket_version_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char bucket_policy[64 * 1024];
    if (argc < 1) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_bucket_version(&bucketContext, bucket_policy,
        sizeof(bucket_policy), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
```

```
    }
    S3_deinitialize();
    return s3_status;
}
```

设置桶生命周期规则

PutBucketLifecycleConfiguration

生命周期规则可以通过设置规则实现自动清理过期的对象, 优化存储空间

PutBucketLifecycleConfiguration 操作用于设置桶的生命周期规则. 规则可以通过匹配对象key前缀或标签等方式来设置其当前版本或历史版本对象的过期时间, 对象过期后会被自动删除. 历史版本对象过期时间的配置只在桶的版本控制状态处于Enabled或Suspended时才能生效. 每次执行PutBucketLifecycleConfiguration 操作都会覆盖桶中已存在的生命周期规则.

api接口

```
void S3_set_lifecycle(const S3BucketContext *bucketContext,
                      const char *json_data, S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
json_data	const char*	传入的生命周期规则信息，json格式	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

其中json_data这一项，需要您自己将生命周期规则转换为json格式的生命周期规则。可参照示例中的格式，需要注意的是，json格式需要保证无误，否则转换为xml的时候可能会失败，导致设置生命周期规则失败。

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int set_bucket_life_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* 生命周期json 数据，请按照c语言格式给出json数据，否则可能无法正确解析，按照以下示例赋值 */
    const char json[] = "{"
       "'Rule': ["
    "{"
                "'Status': 'Enabled',"
                "'Expiration': {"
                   "'Days': 365"
                  "},"
                "'ID': 'Lifecycle Test',"
                "'Filter': {"
                    "'And': {"
                        "'Prefix': 'test',"
                        "'Tag': ["
                            "{"
                               "'Key': 'tag-1',"
                                "'Value': 'val-1'"
                            "}"
                        "]"
                    "}"
                "}"
            "}"
        "]"
    "}";
    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
```

```
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_set_lifecycle(&bucketContext, json, 0, &responseHandler, 0);
    s3_status = get_statusG();
    S3_deinitialize();
    return s3_status;
}
```

查看桶生命周期规则

GetBucketLifecycleConfiguration

GetBucketLifecycleConfiguration 操作用于查看桶当前的生命周期规则

api接口

```
void S3_get_lifecycle(const S3BucketContext *bucketContext,
                      char *lifecycleXmlDocumentReturn, int lifecycleXmlDocumentBufferSize,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
lifecycleXmlDocumentReturn	char *	服务端返回的生命周期规则信息,	是
lifecycleXmlDocumentBufferSize	lifecycleXmlDocumentReturn的大小	请求参数	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_life_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char lifecycleBuffer[64 * 1024];
    if (argc < 1) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_lifecycle(&bucketContext, lifecycleBuffer,
            sizeof(lifecycleBuffer), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
    //    printError();
```

```
    }
    S3_deinitialize();
    return s3_status;
}
```

删除生命周期规则

DeleteBucketLifecycle

DeleteBucketLifecycle 操作用于删除桶的全部生命周期规则
api接口

```
void S3_delete_lifecycle(const S3BucketContext *bucketContext,
                         S3RequestContext *requestContext,
                         const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int delete_bucket_life_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    if (argc < 1) {
        printf("please inpur your bucket \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_delete_lifecycle(&bucketContext, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }
    S3_deinitialize();
```

```
    return s3_status;
}
```

设置桶tag

PutBukcetTag

为指定的 Bucket 设置标签。一个 Bucket 最多设置 50 个标签。该操作需要 s3:PutBucketTagging 权限，桶的所有者默认拥有该权限。该操作会覆盖原有标签。
api接口

```
void S3_set_bucket_tag(const S3BucketContext *bucketContext,
                       const char *json_data, const char *version,
                       S3RequestContext *requestContext,
                       const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	请求的tagging信息	是
version	const char *	bucket相关的一般填写为NULL	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int set_bucket_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag = "{"
        "'TagSet': ["
           "{"
             "'Key': 'key-1',"
             "'Value': 'val-1'"
           "},"
           "{"
             "'Key': 'key-2',"
             "'Value': 'val-2'"
           "},"
           "{"
             "'Key': 'key-3',"
             "'Value': 'val-3'"
           "}"
        "]"
    "}";

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
```

```
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_set_bucket_tag(&bucketContext, tag, version,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

获取桶tag信息

GetBukcetTag

获取指定 BUCKET 的标签。该操作需要 s3:GetBucketTagging 权限，桶的拥有者默认具有该权限。
api接口

```
void S3_get_bucket_tag(const S3BucketContext *bucketContext,
                       const char *version, char *data, int data_len,
                       S3RequestContext *requestContext,
                       const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
version	const char *	bucket相关的一般填写为NULL	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_bucket_tag(&bucketContext, version, data,
        sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

删除桶tag信息

DeleteBukcetTag

删除 Bucket 上的标签。该操作需要 s3:PutBucketTagging 权限，桶的拥有者默认具有该权限。

api接口

```
void S3_delete_bucket_tag(const S3BucketContext *bucketContext,
                          const char *version,
                          S3RequestContext *requestContext,
                          const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
version	const char *	bucket相关的一般填写为NULL	否

参数	类型	说明	是否必要
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int delete_bucket_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_delete_bucket_tag(&bucketContext, version, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
      //  printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置桶cors

PutBucketCors

Put Bucket CORS 接口用来请求设置 Bucket 的跨域资源共享权限。

api接口

```
void S3_set_bucket_cors(const S3BucketContext *bucketContext,
                        const char *json_data, const char *version,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	cors规则信息，以json格式给出	是
version	const char *	bucket相关的一般填写为NULL	否
handler	const	请求的回调函数	是

参数	类型	说明	是否必要
	S3ResponseHandler *		
callbackData	void *	请求回调中的数据	否

json设置cors的示例为

```
const char *tag = "{"
        "'CORSRule': ["
            "{"
                "'AllowedHeader': ["
                    "'*'"
                "],"
                "'AllowedMethod': ["
                    "'PUT'"
                "],"
                "'AllowedMethod': ["
                    "'GET'"
                "],"
                "'AllowedMethod': ["
                    "'POST'"
                "],"
                "'AllowedOrigin': ["
                    "'http://127.0.0.1:5500'"
                "],"
                "'ExposeHeader': ["
                    "'ETag'"
                "],"
                "'MaxAgeSeconds': '30'"
            "}"
        "]"
 "}";
```

各个参数名称说明：

参数	说明	是否必要
CORSRules	为指定 bucket 配置的所有跨域规则	是

参数	说明	是否必要
	的集合，允许配置100 条规则	
ID	跨域规则的 ID,最大长度 255	否
AllowedHeaders	允许浏览器发送 CORS 请求时携带的自定义 HTTP 请求头部，不区分英文大小写，单条 CORSRule 可以配置多个 AllowedHeader。	否
AllowedMethods	允许该源执行的 HTTP 方法列表，包括 GET , PUT , HEAD , POST , and DELETE。单挑规则可以配置多个方法。	是
AllowedOrigins	允许能够访问该 bucket 的一个或多个源,支持 * 通配符，表示所有域名都允许访问，不推荐。一条 CORSRule 可以配置多个 allowedorigins	是
ExposeHeaders	允许浏览器获取的 CORS 请求响应中的头部，不区分英文大小写,单条 CORSRule 可以配置多个ExposeHeader。	否
MaxAgeSeconds	跨域资源共享配置的有效时间，单位为秒，对应 CORS请求响应中的 Access-Control-Max-Age 头部，单条CORSRule 只能配置一个 MaxAgeSeconds	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```
int set_bucket_cors_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag = "{"
            "'CORSRule': ["
                "{"
                    "'AllowedHeader': ["
                        "'*'"
                    "],"
                    "'AllowedMethod': ["
                        "'PUT'"
                    "],"
                    "'AllowedMethod': ["
                        "'GET'"
                    "],"
                    "'AllowedMethod': ["
                        "'POST'"
                    "],"
                    "'AllowedOrigin': ["
                        "'http://127.0.0.1:5500'"
                    "],"
                    "'ExposeHeader': ["
                        "'ETag'"
                    "],"
                    "'MaxAgeSeconds': '30'"
                "}"
            "]"
    "}";

    if (argc < 1) {
        printf("please inpur your bucket \n");
        return -1;
    }
```

```
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_set_bucket_cors(&bucketContext, tag, version,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

获取桶cors

GetBucketCors

Get Bucket CORS 接口用来请求获取 Bucket 的跨域资源共享权限配置。

api接口

```
void S3_get_bucket_cors(const S3BucketContext *bucketContext,
                        const char *version, char *data, int data_len,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
version	const char *	bucket相关的一般填写为NULL	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作

HTTP状态	响应码	描述
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_cors_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_bucket_cors(&bucketContext, version, data,
        sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

删除桶cors

deleteBucketCors

deleteBucketCors 接口用来请求删除 Bucket 的跨域资源共享权限配置。

api接口

```
void S3_delete_bucket_cors(const S3BucketContext *bucketContext,
                           const char *version,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
version	const char *	bucket相关的一般填写为NULL	否
handler	const	请求的回调函数	是

参数	类型	说明	是否必要
	S3ResponseHandler *		
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int delete_bucket_cors_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_delete_bucket_cors(&bucketContext, version, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
      //  printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置桶object lock

setBucketObjectLock

setBucketObjectLock 请求在指定的存储桶上增加对象锁定配置。默认规则将会应用到每一个新放入桶中的对象。

api接口

```
void S3_set_object_lock(const S3BucketContext *bucketContext,
                        const char *json_data, const char *version,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	object lock配置json文件	是
version	const char *	bucket相关的一般填写为NULL	否

参数	类型	说明	是否必要
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

object lock配置json文件示例如下:

```
    const char *tag =  "{"
          "'ObjectLockEnabled': 'Enabled',"
          "'Rule': {"
            "'DefaultRetention': {"
                "'Mode': 'GOVERNANCE',"
                "'Days': 1"
            "}"
          "}"
        "}";
```

参数说明

参数	说明	是否必要
ObjectLockEnabled	表示指定桶的对象锁定功能是否生效。当对存储桶设置了对象锁定配置，即为生效	是
Rule	指定对象的对象锁定规则，配置需要指定模式和时间。年或日只能指定一个，不能在配置中既指定年又指定日	否
DefaultRetention	对象锁定规则中指定的默认模式和时间。存储桶配置同时需要模式和时间，年和日不能同时指定，只能指定一个	否
Mode	存储桶默认的对象锁定保留期限模式	否
Days	保留期限日期，单位天。与年数设置只能二选其一	否
Years	保留期限日期，单位年。与天数设置只能二选其一	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```
int set_object_lock_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag =  "{"
          "'ObjectLockEnabled': 'Enabled',"
          "'Rule': {"
            "'DefaultRetention': {"
                "'Mode': 'GOVERNANCE',"
                "'Days': 1"
            "}"
          "}"
        "}";
    if (argc < 1) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
```

```
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_set_object_lock(&bucketContext, tag, version,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

获取桶object lock

getBucketObjectLock

get bucket object lock 请求获取存储桶的对象锁定配置。默认的对象锁定功能将会应用到每一个新放入到存储桶中的对象。

api接口

```
void S3_get_object_lock(const S3BucketContext *bucketContext,
                        const char *version, char *data, int data_len,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
version	const char *	bucket相关的一般填写为NULL	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_object_lock_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_object_lock(&bucketContext, version, data,
        sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置桶website

setBucketwebsite

调用 setBucketwebsite 接口将存储空间（Bucket）设置成静态网站托管模式
并设置跳转规则（RoutingRule）

api接口

```
void S3_set_bucket_website(const S3BucketContext *bucketContext,
                           const char *json_data,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	setBuckewebsite配置json文件	是
handler	const	请求的回调函数	是

参数	类型	说明	是否必要
	S3ResponseHandler *		
callbackData	void *	请求回调中的数据	否

json_data的各个参数设置格式为

```
WebsiteConfiguration={
'ErrorDocument': {
    'Key': 'string'
},
'IndexDocument': {
    'Suffix': 'string'
},
'RedirectAllRequestsTo': {
    'HostName': 'string',
    'Protocol': 'http'|'https'
},
'RoutingRules': [
{
    'Condition': {
        'HttpErrorCodeReturnedEquals': 'string',
        'KeyPrefixEquals': 'string'
},
'Redirect': {
    'HostName': 'string',
    'HttpRedirectCode': 'string',
    'Protocol': 'http'|'https',
    'ReplaceKeyPrefixWith': 'string',
    'ReplaceKeyWith': 'string'
}
}
};
```

参数说明

参数	说明	是否必要
ErrorDocument	错误文档配置	否
Key	指定通用错误文档的对象键， 当发生错误且未命中重 定向规则中的错误码重定向时，将返回该对象键的内容	是
IndexDocument	索引文档配置	是
Suffix	指定索引文档的对象键后缀。 例如指定为index.html， 那么当访问到存储桶的根目录时， 会自动返回 index.html 的 内容，或者当访问到article/目录时， 会自动返回 article/index.html 的内容	是
RedirectAll RequestsTo	重定向所有请求配置，该规则与其 他规则互斥，也就是说使用了重定 向规则就不能配置其他规则。	否
HostName	要重定向的主机名	是
Protocol	重定向时使用的协议， 默认使用原请求的协议	否
RoutingRules	重定向规则配置	否
Condition	重定向规则的条件配置	否
HttpErrorCode ReturnedEquals	指定重定向规则的错误码匹配条件， 只支持配置4XX 返回码， 例如 403 或 404	当 condition 配置后， HttpErrorCode ReturnedEquals 和KeyPrefixEquals 两者只能配置一个。
KeyPrefixEquals	指定重定向规则的对象键前缀匹配条件	当 condition 配置后， HttpErrorCode

参数	说明	是否必要
		ReturnedEquals和KeyPrefixEquals两者只能配置一个。
Redirect	重定向格则容器，可以配置规则重定向其他主机、页面或其他协议，当发生错误时，也可以配置错误码。	RoutingRules中的必要配置。
HostName	重定向机器名	否
HttpRedirectCode	重定向请求要用的协议，默认使用原请求所是应用的协议。	否
ReplaceKeyPrefixWith	指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分，仅可在 Condition 为 KeyPrefixEquals 时设置	ReplaceKeyWith与ReplaceKeyPrefixWith必选其一
ReplaceKeyWith	指定重定向规则的具体重定向目标的对象键，替换方式为替换整个原始请求的对象键	ReplaceKeyWith 与ReplaceKeyPrefixWith必选其一

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int put_bucket_website_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* website 数据，请按照c语言格式给出json数据，否则可能无法正确解析，按照以下示例赋值 */
    const char data[] =  "{"
        "'ErrorDocument': {"
            "'Key': 'error.html'"
        "},"
        "'IndexDocument': {"
            "'Suffix': 'index.html'"
        "}"
    "}";

    if (argc < 1) {
        printf("please input your bucket name\n");
        return -1;
    }

    bucket = argv[1];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
```

```
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };


    S3_set_bucket_website(&bucketContext, data, 0, &responseHandler, 0);
    s3_status = get_statusG();

    S3_deinitialize();

    return s3_status;
}
```

获取桶website

getBucketwebsite

调用 getBucketwebsite 接口请求用于查询与存储桶关联的静态网站配置信息。

api接口

```
void S3_get_bucket_website(const S3BucketContext *bucketContext,
                           char *data, int data_len,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否

参数	类型	说明	是否必要
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_website_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char bucket_website[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_bucket_website(&bucketContext, bucket_website,
        sizeof(bucket_website), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

删除桶website

DeleteBucketwebsite

调用 DeleteBucketwebsite 接口请求用于删除存储桶中的静态网站配置。

api接口

```
void S3_delete_bucket_website(const S3BucketContext *bucketContext,
                              S3RequestContext *requestContext,
                              const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int delete_bucket_website_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;

    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_delete_bucket_website(&bucketContext, 0, &responseHandler, 0);
```

```
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
        // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置桶Logging

setBuckeLogging

put bucket logging 请求设置日志转存参数。所有的日志将会保留到和源存储桶属于同一拥有者的目标存储桶中。桶的拥有者可以设置桶的日志状态。桶的拥有者对所有的日志具有 FULL_CONTROL 权限，可以通过 Grantee 授权其他用户，其中 Permissions 参数指定了用户对日志的访问权限。

api接口

```
void S3_set_bucket_logging(const S3BucketContext *bucketContext, const char *json_data,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	setBuckeLogging配置json文件	是
handler	const	请求的回调函数	是

参数	类型	说明	是否必要
	S3ResponseHandler *		
callbackData	void *	请求回调中的数据	否

setBuckeLoggingd的配置文件组装格式为

```
BucketLoggingStatus={
'LoggingEnabled': {
'TargetBucket': 'string',
'TargetGrants': [
{
    'Grantee': {
    'DisplayName': 'string',
    'EmailAddress': 'string',
    'ID': 'string',
    'Type':
    'CanonicalUser'|'AmazonCustomerByEmail'|'Group',
    'URI': 'string'
},
'Permission': 'FULL_CONTROL'|'READ'|'WRITE'
},
],
'TargetPrefix': 'string'
}
}
```

参数说明

参数	说明	是否必要
LoggingEnabled	描述日志存储位置和日志对象前缀	否
TargetBucket	日志存储位置。可以将日志存放到任意用户拥有的桶中，包含源存储桶。用户可以配置多个源桶的日志均投放到同一个目标存储桶中，在这种情况下，用户可以使用 TargetPrefix 区分日志来自哪个源存储桶。	是

参数	说明	是否必要
TargetGrants	授权信息	否
Grantee	授权许可	否
DisplayName	展示名字	否
EmailAddress	邮件地址	否
ID	授权用户ID	否
Type	授权类型	是
URI	授权组 URI	否
Permission	日志访问许可	否
TargetPrefix	日志对象前缀。若多个源存储桶的日志均写到同一个目标存储桶中，则可以通过目标前缀来区分日志来自哪一个源存储桶	是

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int put_bucket_logging_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* logging 数据，请按照c语言格式给出json数据，否则可能无法正确解析，按照以下示例赋值 */
    const char data[] =  "{"
    "'LoggingEnabled': {"
        "'TargetBucket': 'bucket-ac',"
        "'TargetGrants': ["
            "{"
            "'Grantee': {"
                "'ID': '21f1c2f7-ee85-5ba7-9355-dfc3c008eb21',"
                "'Type': 'CanonicalUser'"
            "},"
            "'Permission': 'READ'"
            "}"
        "],"
        "'TargetPrefix': 'log/'"
    "}"
    "}";

    if (argc < 1) {
        printf("please input your bucket name\n");
        return -1;
    }

    bucket = argv[1];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
```

```
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };


    S3_set_bucket_logging(&bucketContext, data, 0, &responseHandler, 0);
    s3_status = get_statusG();

    S3_deinitialize();

    return s3_status;
}
```

获取桶logging

getBuckelogging

调用 getBuckelogging 请求获取存储桶的日志转存配置
api接口

```
void S3_get_bucket_logging(const S3BucketContext *bucketContext,
                           char *data, int data_len,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_logging_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char bucket_logging[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_bucket_logging(&bucketContext, bucket_logging,
        sizeof(bucket_logging), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置桶encryption

setBucketencryption

setBucketencryption 请求可以启用存储桶默认加密功能

api接口

```
void S3_set_bucket_encryption(const S3BucketContext *bucketContext,
                              const char *json_data,
                              S3RequestContext *requestContext,
                              const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	setBucketencryption配置json文件	是
handler	const S3ResponseHandler	请求的回调函数	是

参数	类型	说明	是否必要
	*		
callbackData	void *	请求回调中的数据	否

json文件的组装格式为

```
ServerSideEncryptionConfiguration={
'Rules': [
{
    'ApplyServerSideEncryptionByDefault': {
    'SSEAlgorithm': 'AES256'|'aws:kms',
    'KMSMasterKeyID': 'string'
},
},
]
}
```

参数说明

参数	说明	是否必要
Rules	一个特殊的服务端加密配置规则信息	是
ApplyServerSide EncryptionByDefault	指定默认的服务端加密会应用于 新对象上传至存储桶时。若是在上传对象时请求中未指定任何加密信息，则存储桶默认加密将会应用	否
SSEAlgorithm	服务端加密算法	是
KMSMasterKeyID	若加密算法选用的是 aws:kms，则此项必填， 按照cmkuuid:keyspec:userid 模式配置，其中 cmkuuid 是CMKID， keyspec 是指定生成的数据密钥长度，userid 是用户 id； 若是 AES256 算法， 则此项可不填，若填，则字符长度需为 32	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```
int put_bucket_encryption_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* website 数据，请按照c语言格式给出json数据，否则可能无法正确解析，按照以下示例赋值 */
    const char data[] =  "{"
    "'Rule': ["
        "{"
            "'ApplyServerSideEncryptionByDefault': {"
            "'SSEAlgorithm': 'AES256'"
            "}"
        "}]"
    "}";

    if (argc < 1) {
        printf("please input your bucket name\n");
        return -1;
    }

    bucket = argv[1];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
```

```
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };


    S3_set_bucket_encryption(&bucketContext, data, 0, &responseHandler, 0);
    s3_status = get_statusG();

    S3_deinitialize();

    return s3_status;
}
```

获取桶encryption

getBucketEncryption

调用 getBucketEncryption 接口可以返回存储桶默认加密配置。若是存储桶不存 在默认加密配置，则返回 NoSuchEncryptionSetError 错误。

api接口

```
void S3_get_bucket_encryption(const S3BucketContext *bucketContext,
                        char *data, int data_len,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是

参数	类型	说明	是否必要
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int get_bucket_encryption_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
```

```
    S3_get_bucket_encryption(&bucketContext, data,
        sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

删除桶Encrytion

DeleteBucketEncrytion

调用 DeleteBucketEncrytion 接口请求删除存储桶默认加密配置

api接口

```
void S3_delete_bucket_encryption(const S3BucketContext *bucketContext,
                                 S3RequestContext *requestContext,
                                 const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

```c
int delete_bucket_encryption_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;

    if (argc < 2) {
        printf("please inpur your bucket \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_delete_bucket_encryption(&bucketContext, 0, &responseHandler, 0);
```

```
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

对象操作

上传对象

PutObject

PutObject 操作用于上传对象. 单次上传最大不能超过5GB, 超过5GB的文件需要使用分段上传操作。对象权限默认为私有权限。

api接口展示

```
void S3_put_object(const S3BucketContext *bucketContext, const char *key,
                   uint64_t contentLength,
                   const S3PutProperties *putProperties,
                   S3RequestContext *requestContext,
                   const S3PutObjectHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
key	const char *	将上传对象的名	是
contentLength	uint64_t	将上传对象的大小（byte）	是

参数	类型	说明	是否必要
putProperties	const S3PutProperties *	请求消息头	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

在putProperties 请求参数中， 您可以设置md 字段来保证数据被正确上传, 若上传的数据不能通过MD5校验则将返回错误提示. 用户也可以自行对比上传后得到的ETag和原文件的MD5值来确认是否正确上传。

你也可以在putProperties结构体中设置HTTP协议规定请求头: Cache-Control , Expires ,Content-Encoding , Content-Disposition , Content-Type, ACL等 . 若上传对象的请求设置了这些请求头, 服务端会记录他们, 当对象被下载时这些值会被设置到对应的HTTP头域中返回客户端. 这些值可以通过修改对象元数据操作进行修改。如果您在putProperties中没有设置ACL信息，则对象的ACL默认为私有权限。

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例代码

```
int put_object(int argc, char **argv)
{
    const char *bucket = NULL; /* 要上传对象的桶 */
    const char *key = NULL; /* 上传到桶内的对象的key */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *filename = NULL;
    int s3_status = 0;
    uint64_t contentLength = 0;/* 要上传的对象长度 */
    uint64_t expires = -1;
    S3Status status;
    S3Protocol protocolG = S3ProtocolHTTP;
    const char *cacheControl = 0, *contentType = 0, *md5 = "";
    const char *contentDispositionFilename = 0, *contentEncoding = 0;
    S3CannedAcl cannedAcl = S3CannedAclPrivate;
    int metaPropertiesCount = 0;
    S3NameValue metaProperties[S3_MAX_METADATA_COUNT];
    put_object_callback_data data;
    /* 获取要上传的文件长度 */
    if (argc < 3) {
        printf("please input bucket/key/filePath\n");
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    filename = argv[3];
    data.gb = 0;
    data.infile = 0;
    data.noStatus = 1;
    if (filename) {
        if (!contentLength) { /* 文件长度没有定义 */
            struct stat statbuf;
            if (stat(filename, &statbuf) == -1) {
                printf("\n Error: filed to open file %s \n", filename);
                return -1;
            }
            contentLength = statbuf.st_size;
        }
```

```
        /* 打开需要上传的文件 */
        if (!(data.infile = fopen(filename, "r"))) {
            fprintf(stderr, "\n ERROR: Failed to open input file %s: ",
                    filename);
            return -1;
        }
    } else {
        printf("please input your please input which object you want put\n");
        return -1;
    }
   // printf("\n we'll put file   %s     to bueckt    %s    \n", filename, bucket);
    data.contentLength = data.originalContentLength = contentLength;
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        fclose(data.infile);
        exit(-1);
    }
     /* 2. s3基本信息设置 */
    S3BucketContext bucketContext =
    {
        hostname,
        bucket,
        protocolG,
        S3UriStylePath,
        access_key,
        secret_key
    };
    /* 3. put object基本信息设置 */
    S3PutProperties putProperties =
    {
        contentType,
        md5,
        cacheControl,
        contentDispositionFilename,
        contentEncoding,
        expires,
        cannedAcl,
        metaPropertiesCount,
```

```
        metaProperties
    };
    /* 4. 回调信息设置 */
    S3PutObjectHandler putObjectHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &putObjectDataCallback
    };
    /* 5. 调用接口 */
    S3_put_object(&bucketContext, key, contentLength, &putProperties,
        0, &putObjectHandler, &data);
    s3_status = get_statusG();
    if (data.infile) {
        fclose(data.infile);
    }
    if (s3_status != S3StatusOK) {
       // printError();
    } else if (data.contentLength) {
        fprintf(stderr, "\nERROR: Failed to read remaining %llu bytes from "
                "input\n", (unsigned long long) data.contentLength);
    } else {
        printf("\n put file   %s     to bucket    %s     successfully \n", filename, bucket);
    }
    S3_deinitialize();
    return s3_status;
}
```

获取对象

GetObject

GetObject 操作用于获取对象数据. 执行 GetObject 操作必须对目标对象具有 READ 权限

api接口示例

```
void S3_get_object(const S3BucketContext *bucketContext, const char *key,
                   const S3GetConditions *getConditions,
                   uint64_t startByte, uint64_t byteCount,
                   S3RequestContext *requestContext,
                   const S3GetObjectHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
key	const char *	将要获取对象的名	是
getConditions	const S3GetConditions *	成功返回条件	是
startByte	uint64_t	开始位置	是
byteCount	uint64_t	总长度	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	是

在getConditions参数中，您可以设置获取对象的条件，参数有以下几个

参数	说明
ifModifiedSince	只有当对象自指定时间以来已被修改时才返回对象, 否则返回304
ifNotModifiedSince	只有当对象自指定时间以来未被修改时才返回对象, 否则返回412
ifMatchETag	只有当对象的 ETag 与该值相同时才返回对象, 否则返回412错误
ifNotMatchETag	只有当对象的ETag与该值相同时才返回对象, 否则返回304错误

startByte和byteCount都设置为0即代表获取整个对象
响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
304	NotModified	文件未在指定参数后修改
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	操作指定的桶不存在
404	NoSuchKey	操作指定的对象不存在
412	PreconditionFailed	IfMatch 或者 IfUnmodifiedSince 前置条件不符合
416	InvalidRange	Range 指定的范围不合法

示例

```
int get_object(int argc, char **argv)
{
    const char *bucket = NULL; /* 要获取对象的桶 */
    const char *key = NULL; /* 桶内的对象的key */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *filename = NULL;
    int64_t ifModifiedSince = -1, ifNotModifiedSince = -1;
    const char *ifMatch = NULL, *ifNotMatch = NULL;
    unsigned long long startByte = 0;
    unsigned long long byteCount = 0; /* 设置为0即代码获取整个对象 */
    FILE *outfile = 0;
    int s3_status = 0;
    S3Status status;

    printf("%llu\n", byteCount);

    if (argc < 3) {
        printf("please input bucket/key/filename\n");
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    filename = argv[3];

    /* 先测试给出的文件是否可以打开，如文件不存在，则直接创建一个 */
    if (filename) {
        struct stat buf;
        if (stat(filename, &buf) == -1) {
            outfile = fopen(filename, "w");
        }
        else {
            outfile = fopen(filename, "r+");
        }

        if (!outfile) {
            fprintf(stderr, "\nERROR: Failed to open output file %s: ",
                    filename);
```

```
        return -1;
    }
} else {
    printf("filename is null. please input object out file!!\n");
    return -1;
}

/* 1. 初始化libs3 */
if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
    fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
    fclose(outfile);
    return -1;
}

 /* 2. s3基本信息设置 */
S3BucketContext bucketContext =
{
    hostname,
    bucket,
    S3ProtocolHTTP,
    S3UriStylePath,
    access_key,
    secret_key
};

/* 3. 获取对象的条件 */
S3GetConditions getConditions =
{
    ifModifiedSince,
    ifNotModifiedSince,
    ifMatch,
    ifNotMatch
};

/* 4. 回调函数 */
S3GetObjectHandler getObjectHandler =
{
    { &responsePropertiesCallback, &responseCompleteCallback },
    &getObjectDataCallback
```

```
    };

    /* 调用获取对象接口 */
    S3_get_object(&bucketContext, key, &getConditions, startByte,
        byteCount, 0, &getObjectHandler, outfile);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    } else {
        printf("\n get bucket/obj %s / %s successfully, path %s  \n", bucket, key, filename);
    }

    fclose(outfile);

    S3_deinitialize();

    return s3_status;
}
```

获取对象列表

ListObjects

ListObjects 操作用于列出桶中的全部对象,该操作最多返回1000个对象信息, 可以通过设置过滤条件来列出桶中符合特定条件的对象

api接口

```
void S3_list_bucket(const S3BucketContext *bucketContext, const char *prefix,
                    const char *marker, const char *delimiter, int maxkeys,
                    S3RequestContext *requestContext,
                    const S3ListBucketHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
prefix	const char *	返回以指定前缀开头的key	否
marker	const char *	指定开始列出对象的分隔key (按字典序)	否
delimiter	const char *	一个分隔符, 用于分组键	否
maxkeys	int	返回的对象最大个数, 默认以及最大值均为1000	否
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	是

响应结果:

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例：

```c
int list_object(int argc, char **argv)
{
    const char *bucket = ""; /* 要获取对象的桶 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *prefix = "", *marker = 0, *delimiter = 0; /* 对应sdk文档中介绍的几个参数 */
    int maxkeys = 100, allDetails = 1;  /* max key,  allDetails表示是否打印对象的详细信息*/
    int s3_status = 0;
    S3Status status;
    if (argc < 2) {
        printf("please input your bucket name\n");
        return -1;
    }
    bucket = argv[1];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    /* 2. S3请求基础信息 */
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTPS,
        S3UriStylePath,
        access_key,
        secret_key
    };
    /* 3. 回调函数 */
    S3ListBucketHandler listBucketHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &listBucketCallback
    };
    /* 4. callback data结构定义 */
    list_bucket_callback_data data;
    snprintf(data.nextMarker, sizeof(data.nextMarker), "%s", marker);
```

```
    data.keyCount = 0;
    data.allDetails = allDetails;
    /* 调用接口请求list object */
    do {
        data.isTruncated = 0;
        do {
            S3_list_bucket(&bucketContext, prefix, data.nextMarker,
                           delimiter, maxkeys, 0, &listBucketHandler, &data);
            s3_status = get_statusG();
        } while (S3_status_is_retryable(s3_status) && should_retry());
        if (s3_status != S3StatusOK) {
            break;
        }
    } while (data.isTruncated && (!maxkeys || (data.keyCount < maxkeys)));
    if (s3_status == S3StatusOK) {
        if (!data.keyCount) {
            printListBucketHeader(allDetails);
        }
    }
    else {
      //  printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

获取桶中所有对象以及对象的versions

Listversions

该接口可获取一个桶中所有对象以及对象的所有version

api接口

```
void S3_list_versions(const S3BucketContext *bucketContext,
                      char *data, int data_len,  list_versions_t *params,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
params	list_versions_t *	list versions时筛选的条件	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int list_versions_test(int argc, char **argv)
{
    S3Status status;
    int s3_status = 0;
    char data[2 * 1024 * 1024];
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */

    if (argc < 1) {
        printf("please input your bucket name\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    list_versions_t params = {
        NULL,/* Delimiter */
        NULL,/* EncodingType */
        NULL,/* KeyMarker */
        NULL,/* MaxKeys */
        NULL, /* Prefix */
        NULL,/* VersionIdMarker */
        NULL,/* ExpectedBucketOwner */
    };

    bucket = argv[1];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
```

```
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_list_versions(&bucketContext, data, sizeof(data), &params, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

删除对象

DeleteObject

DeleteObject 操作用于删除一个对象

api接口

```
void S3_delete_object(const S3BucketContext *bucketContext, const char *key,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
key	const char *	需要删除的对象名称	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
204	NoContent	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

注：即使删除不存在的对象也会返回204 成功错误码。

示例

```c
int delete_object_test(int argc, char **argv)
{
    const char *bucket = ""; /* 要删除的对象的桶 */
    const char *key = "";
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    S3Status status;
    if (argc < 2) {
        printf("please input your bucket name && object key\n");
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    /* 2. S3请求信息 */
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTPS,
        S3UriStylePath,
        access_key,
        secret_key
    };
    /* 3. 回调信息 */
    S3ResponseHandler responseHandler =
    {
        0,
        &responseCompleteCallback
    };
    /* 4. 调用S3删除对象接口 */
    S3_delete_object(&bucketContext, key, 0, &responseHandler, 0);
    s3_status = get_statusG();
```

```
    return s3_status;
}
```

批量删除对象

DeleteObjects

DeleteObject 操作用于删除一批对象

api接口

```
void S3_delete_objcets(const S3BucketContext *bucketContext,
                       const char *objects,
                       S3RequestContext *requestContext,
                       const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
objects	const char *	需要删除的对象名称组合，以json格式给出	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
204	NoContent	操作成功

HTTP状态	响应码	描述
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```
int delete_objects_test(int argc, char **argv)
{
    S3Status status;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    const char *bucket = NULL;
    const char *objects = "{"
          "'Object': {'Key': 'object12'},"
          "'Object': {'Key': 'object1'},"
          "'Object': {'Key': 'test_pu10'}"
        "}";

    if (argc < 1) {\
       printf("please input your bucket name\n");
       return -1;
    }

    bucket = argv[1];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
```

```
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_delete_objcets(&bucketContext, objects, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

copy对象

CopyObject

CopyObject 操作用于将已有的对象拷贝到其他位置. 可以拷贝的单个对象最大为5GB, 执行 CopyObject 操作必须具有对被拷贝对象的READ权限和对目标桶的WRITE权限。
拷贝生成的对象默认保留原对象的元数据信息, 也可以在操作时指定新的元数据. 拷贝生成的对象并不会保留原本的ACL信息, 该对象权限默认为操作者私有。
若源桶开启了版本控制, 操作默认拷贝原对象当前版本的数据, 将生成一个与原对象不同的唯一的版本id并返回. 如需要特定版本的数据, 可以通过版本id参数来指定. 若原对象的版本已被删除标记, 则不会进行拷贝.

api接口

```
void S3_copy_object(const S3BucketContext *bucketContext, const char *key,
                    const char *destinationBucket, const char *destinationKey,
                    const S3PutProperties *putProperties,
                    int64_t *lastModifiedReturn, int eTagReturnSize,
                    char *eTagReturn, S3RequestContext *requestContext,
                    const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	将拷贝的对象名	是
destinationBucket	const char *	拷贝的目的bucket	是
destinationKey	const char *	目的对象名	是
putProperties	const S3PutProperties *	请求消息头	否
lastModifiedReturn	int64_t *	对象上次修改时间	是
eTagReturnSize	int	eTag缓存区大小	是
eTagReturn	char *	eTag缓存区	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

您可以在putProperties 结构体中设置目标对象的元数据信息。如没有设置，则沿用源对象的元数据信息。目前大文件尚未支持copy object。

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在

示例

```c
int copy_object_test(int argc, char **argv)
{
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    /* 需要拷贝的源桶，需要拷贝对象的目标桶，拷贝对象的源key，拷贝对象的目标key */
    const char *src_bucket = NULL, *dst_bucket = NULL, *src_key = NULL,  *dst_key = NULL;
    int modify_meta = 0, s3_status = 0;
    S3Status status;
    int64_t lastModified;
    char eTag[256];
    if (argc < 4) {
        printf("please input your src_bucket / dst bucket / src_key / dst key info \n");
        return -1;
    }
    src_bucket = argv[1];
    dst_bucket = argv[2];
    src_key = argv[3];
    dst_key = argv[4];
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    /* S3请求基本信息 */
    S3BucketContext bucketContext =
    {
        0,
        src_bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    /* 3. 定义拷贝目标对象的信息，如果您需要目标对象的信息有所改变，则将modify_meta赋值为1 */
    S3PutProperties putProperties =
    {
        0,
        0,
```

```
        0,
        0,
        0,
        0,
        S3CannedAclPublicReadWrite,
        0,
        0
    };
    lastModified = 0;
    /* 4. 回调信息 */
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    /* 5. 调用S3接口拷贝对象 */
    S3_copy_object(&bucketContext, src_key, dst_bucket,
        dst_key, modify_meta ? 0 : &putProperties,
        &lastModified, sizeof(eTag), eTag, 0,
        &responseHandler, 0);
    s3_status = get_statusG();
    /* 6. 打印目标对象的信息 */
    if (s3_status == S3StatusOK) {
        if (lastModified >= 0) {
            char timebuf[256];
            time_t t = (time_t) lastModified;
            strftime(timebuf, sizeof(timebuf), "%Y-%m-%dT%H:%M:%SZ",
                     gmtime(&t));
            printf("Last-Modified: %s\n", timebuf);
        }
        if (eTag[0]) {
            printf("ETag: %s\n", eTag);
        }
    }
    else {
       // printError();
    }
    S3_deinitialize();
    return s3_status;
```

```
}
```

获取对象元数据

HeadObject

HeadObject 操作用于获取对象的元数据信息. 执行 HeadObject 操作必须具有对该对象的 READ 权限. 可以将 HeadObject 当作 GetObject 的弱化版

api接口

```
void S3_head_object(const S3BucketContext *bucketContext, const char *key,
                    S3RequestContext *requestContext,
                    const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	需要head的对象名称	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```
int head_object_test(int argc, char **argv)
{
    const char *bucket = NULL, *key = NULL; /* 要访问的bucket以及Key name */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    if (argc < 2) {
        printf("please inpur your bucket / key, so we can get it \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_head_object(&bucketContext, key, 0, &responseHandler, 0);
    s3_status = get_statusG();
    if ((s3_status != S3StatusOK) &&
        (s3_status != S3StatusErrorPreconditionFailed)) {
     //   printError();
    }
```

```
    S3_deinitialize();
    return s3_status;
}
```

设置对象标签

PutObjectTagging

PutObjectTagging 操作用于为对象设置标签. 标签的格式为键值对, 每个对象最多有10个标签. 桶的拥有者默认拥有给桶中对象设置标签的权限, 可以将该权限授予其他用户
每次 PutObjectTagging 都会覆盖对象原有的标签信息, 设置标签并不会导致对象的最后更改时间发生变化
api接口

```
void S3_set_object_tag(const S3BucketContext *bucketContext,
                       const char *json_data, const char *key, const char *version,
                       S3RequestContext *requestContext,
                       const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
json_data	const * char	传入的对象标签信息，json格式	是
key	const char *	需要操作的对象名称	是
version	const char *	如果对象开启了多版本，需要传入对象的版本id	否
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler	回调函数	是

参数	类型	说明	是否必要
	*		
callbackData	void *	回调数据	否

用户需要传入的标签键值对信息可参考天翼云官网说明
响应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidBucketName	请求中设置的桶的名称不合法
400	InvalidObjectName	请求中设置的对象的名称不合法
400	InvalidTag	设置的标签数量太多或者标签内容长度超限
400	MalformedXML	设置的标签内容非法或者存在key重复的情况 或者xml格式有问题
403	AccessDenied	用户没有权限执行操作
404	NoSuchKey	操作指定的对象不存在
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int set_object_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag = "{"
        "'TagSet': ["
           "{"
             "'Key': 'key-1',"
             "'Value': 'val-1'"
           "},"
           "{"
             "'Key': 'key-2',"
             "'Value': 'val-2'"
           "},"
           "{"
             "'Key': 'key-3',"
             "'Value': 'val-3'"
           "}"
        "]"
    "}";
    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
```

```
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_set_object_tag(&bucketContext, tag, key, version,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

获取对象标签

GetObjectTagging

GetObjectTagging 操作用于查询对象的标签信息
api接口

```
void S3_get_object_tag(const S3BucketContext *bucketContext, const char *key,
                       char *data, int data_len,
                       S3RequestContext *requestContext,
                       const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	需要操作的对象名称	是
version	const char *	如果对象开启了多版本， 需要传入对象的版本id	否
data	char *	请求回调的数据，不必赋值， 需要给出结构体	是
data_len	int	data的长度	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidBucketName	请求中设置的桶的名称不合法
400	InvalidObjectName	请求中设置的对象的名称不合法
403	AccessDenied	用户没有权限执行操作
404	NoSuchKey	操作指定的对象不存在
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int get_object_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];
    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_object_tag(&bucketContext, key, version, data,
        sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();
```

```
    if (s3_status != S3StatusOK) {
       // printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

删除对象标签

DeleteObjectTagging

DeleteObjectTagging 操作用于删除一个对象的全部标签信息. 删除时可以指定版本id参数来删除指定版本的对象的标签信息, 不指定时默认操作于当前版本

```
void S3_delete_object_tag(const S3BucketContext *bucketContext,
                          const char *key, const char *version,
                          S3RequestContext *requestContext,
                          const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	需要操作的对象名称	是
version	const char *	如果对象开启了多版本, 需要传入对象的版本id	否
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应码

HTTP状态	响应码	描述
204	Success	操作成功
400	InvalidBucketName	请求中设置的桶的名称不合法
400	InvalidObjectName	请求中设置的对象的名称不合法
403	AccessDenied	用户没有权限执行操作
404	NoSuchKey	操作指定的对象不存在
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int delete_object_tag_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_delete_object_tag(&bucketContext, key, version, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
```

```
      //  printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

设置对象访问权限

PutObjectAcl

PutObjectAcl 操作用于为对象设置访问权限, 对一个对象执行 PutObjectAcl 操作需要具有 WRITE_ACP 权限

对象的权限说明

权限类型	说明
READ	可以读取对象的数据和元数据信息
READ_ACP	可以读取对象的ACL信息. 对象的拥有者默认具有对象的 READ_ACP 权限
WRITE	可以修改原有对象数据和删除对象
WRITE_ACP	可以修改对象的ACL信息, 授予该权限相当于授予 FULL_CONTROL 权限, 因为具有 WRITE_ACP 权限的用户可以配置对象的任意权限. 对象的拥有者默认具有对象的 WRITE_ACP 权限
FULL_CONTROL	同时授予 READ , READ_ACP , WRITE 和 WRITE_ACP 权限

api接口

```
void S3_set_acl(const S3BucketContext *bucketContext, const char *key,
                const char *ownerId, const char *ownerDisplayName,
                int aclGrantCount, const S3AclGrant *aclGrants,
                S3RequestContext *requestContext,
                const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	需要设置的对象名称	是
ownerId	char *	设置acl所需要的owner id	是
ownerDisplayName	char *	设置acl所需要的owner displayName	是
aclGrantCountReturn	int *	设置acl权限个数	是
aclGrants	S3AclGrant *	设置的acl权限信息	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
400	InvalidBucketName	请求中设置的桶的名称不合法
400	InvalidObjectName	请求中设置的对象的名称不合法
400	InvalidArgument	设置的ACL参数无效
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	操作指定的桶不存在
404	NoSuchKey	操作指定的对象不存在或版本id不存在

示例

```c
int set_object_acl_test(int argc, char **argv)
{
    const char *bucket = NULL, *key = NULL; /* 要访问的bucket以及Key name */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    int aclGrantCount = 1;/* 赋予的acl cnt */
    S3AclGrant aclGrants[S3_MAX_ACL_GRANT_COUNT]; /* acl信息 */
    const char *displayName = "胡豪";
    const char *id = "21f1c2f7-ee85-5ba7-9355-dfc3c008eb21";
    const char *email = "nocompatible@outlook.com";
    if (argc < 2) {
        printf("please inpur your bucket / key, so we can get it \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    aclGrants[0].granteeType = S3GranteeTypeAmazonCustomerByEmail; /* acl设置的账户认证类型 */
```

```
  //  copy_string_test(displayName, aclGrants[0].grantee.canonicalUser.displayName);
   // copy_string_test(id, aclGrants[0].grantee.canonicalUser.id);
    copy_string_test(email, aclGrants[0].grantee.amazonCustomerByEmail.emailAddress);
    aclGrants[0].permission = S3PermissionFullControl; /* acl设置对象的权限 */
    S3_set_acl(&bucketContext, key, id, displayName,
        aclGrantCount, aclGrants, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

获取对象访问权限

GetObjectAcl

GetObjectAcl 操作用于获取对象的访问权限信息, 对一个对象执行 GetObjectAcl 操作需要具有 READ_ACP 权限

api接口

```
void S3_get_acl(const S3BucketContext *bucketContext, const char *key,
                char *ownerId, char *ownerDisplayName,
                int *aclGrantCountReturn, S3AclGrant *aclGrants,
                S3RequestContext *requestContext,
                const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是

参数	类型	说明	是否必要
key	const char *	需要获取的对象名称	是
ownerId	char *	返回结果的owner id, 需要传入结构体，不必赋值	是
ownerDisplayName	char *	返回结果的owner displayname, 需要传入结构体，不必赋值	是
aclGrantCountReturn	int *	返回的acl信息个数, 需要传入变量，不必赋值	是
aclGrants	S3AclGrant *	返回的acl权限信息, 需要传入结构体，不必赋值	是
requestContext	S3RequestContext *	请求参数	否
handler	const S3ResponseHandler *	回调函数	是
callbackData	void *	回调数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```
int get_object_acl_test(int argc, char **argv)
{
    const char *bucket = NULL, *key = NULL; /* 要访问的bucket以及Key name */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    /* acl相关信息变量 */
    int aclGrantCount;
    S3AclGrant aclGrants[S3_MAX_ACL_GRANT_COUNT];
    char ownerId[S3_MAX_GRANTEE_USER_ID_SIZE];
    char ownerDisplayName[S3_MAX_GRANTEE_DISPLAY_NAME_SIZE];
    if (argc < 2) {
        printf("please inpur your bucket / key,so we can get it \n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };
    S3_get_acl(&bucketContext, key, ownerId, ownerDisplayName,
```

```
                &aclGrantCount, aclGrants, 0, &responseHandler, 0);
    s3_status = get_https_status();
    if (s3_status >= 200 && s3_status< 300) {
        printf( "OwnerID %s %s\n\n", ownerId, ownerDisplayName);
        printf("%-6s  %-90s  %-12s\n", " Type",
                "                                       User Identifier",
                " Permission");
        printf("------  "
                "------------------------------------------------------------"
                "------------------------------  ------------\n");
        int i;
        for (i = 0; i < aclGrantCount; i++) {
            S3AclGrant *grant = &(aclGrants[i]);
            const char *type;
            char composedId[S3_MAX_GRANTEE_USER_ID_SIZE +
                            S3_MAX_GRANTEE_DISPLAY_NAME_SIZE + 16];
            const char *id;
            switch (grant->granteeType) {
            case S3GranteeTypeAmazonCustomerByEmail:
                type = "Email";
                id = grant->grantee.amazonCustomerByEmail.emailAddress;
                break;
            case S3GranteeTypeCanonicalUser:
                type = "UserID";
                snprintf(composedId, sizeof(composedId),
                         "%s (%s)", grant->grantee.canonicalUser.id,
                         grant->grantee.canonicalUser.displayName);
                id = composedId;
                break;
            case S3GranteeTypeAllAwsUsers:
                type = "Group";
                id = "Authenticated AWS Users";
                break;
            case S3GranteeTypeAllUsers:
                type = "Group";
                id = "All Users";
                break;
            default:
                type = "Group";
```

```
                id = "Log Delivery";
                break;
            }
            const char *perm;
            switch (grant->permission) {
            case S3PermissionRead:
                perm = "READ";
                break;
            case S3PermissionWrite:
                perm = "WRITE";
                break;
            case S3PermissionReadACP:
                perm = "READ_ACP";
                break;
            case S3PermissionWriteACP:
                perm = "WRITE_ACP";
                break;
            default:
                perm = "FULL_CONTROL";
                break;
            }
            printf("%-6s  %-90s  %-12s\n", type, id, perm);
        }
    }
    else {
    //    printError();
    }
    S3_deinitialize();
    return s3_status;
}
```

生成预签名下载链接

GeneratePresignedUrl

GeneratePresignedUrl 操作用于生成预签名的下载链接, 打开该链接可直接下载对象. 该操作通过GetObjectRequest 操作和 Presign 组合实现, 故执行该操作需要具有对象的 READ 权限。该接口使用v2算法计算预签名。使用预签名上传的对象，权限默认为私有。

api接口

```
S3Status S3_generate_authenticated_query_string
    (char *buffer, const S3BucketContext *bucketContext,
     const char *key, int64_t expires, const char *resource, const char *http)
```

参数说明

参数	类型	说明	是否必要
buffer	char *	生成的url信息	是
bucketContext	const S3BucketContext *	获取桶的信息	是
key	char *	需要获取的对象名	是
expires	int64_t	url到期时间, 自UnixEpoch以来已过去的秒数, 在大于这个直接以后的请求将不再有效； 如果该值为负值，则为最大	是
resource	const char *	子资源信息	否
http	const char *	http请求方法，GET or PUT	是

在示例中，笔者给出的expires是通过Iso8601T格式进行转化的，您也可以自己生成url的过期时间，如觉得麻烦，可以仿造示例中的方式。

示例

```c
int generate_presigned_url_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char *http_method = NULL;/* HTTP方法 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *expires_Iso8601 = NULL; /* 输入的日期需要以iso8601格式给出，例如2024-03-29T20:36:14 *
    const char *resource = NULL; /* 子资源信息，如果预签名中想要设置获取的子资源信息，在这里设置 */
    int64_t expires = -1;
    S3Status status = 0;

    if (argc < 3) {
        printf("please input bucket/key/expires/http_method\n");
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    /* 转换时间 */
    expires_Iso8601 = argv[3];
    expires = parseIso8601TimeNew(expires_Iso8601);
    if (expires < 0) {
        printf("\nERROR: Invalid expires time value; ISO 8601 time format required\n");
        return -1;
    }

    http_method = argv[4];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
```

```
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    char buffer[S3_MAX_AUTHENTICATED_QUERY_STRING_SIZE];

    status = S3_generate_authenticated_query_string(buffer,
        &bucketContext, key, expires, resource, http_method);

    if (status != S3StatusOK) {
        printf("Failed to generate authenticated query string: %s\n",
               S3_get_status_name(status));
    }
    else {
        printf("%s\n", buffer);
    }

    S3_deinitialize();

    return status;
}
```

GeneratePresignedUrlV4

GeneratePresignedUrl 操作用于生成预签名的下载链接, 打开该链接可直接下载对象. 该操作通过GetObjectRequest 操作和 Presign 组合实现, 故执行该操作需要具有对象的 READ 权限。该接口使用v4算法计算预签名。使用预签名上传的对象，权限默认为私有。
api接口

```
S3Status S3_generate_authenticated_query_string_v4
    (char *buffer, const S3BucketContext *bucketContext,
     const char *key, int64_t expires, const char *resource, const char *httpMethod)
```

参数说明

v4的预签名接口除了expires和v2的预签名有区别外，其他的都是一样的。V4算法中传入的expires代表预签名请求超时时间，单位为秒，最大为7天,这里和v2的预签名有区别。

示例

```c
int generate_presigned_url_v4_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *key = NULL; /* 要访问的对象名称 */
    const char *http_method = NULL;/* HTTP方法 */
    const char *char_expires = NULL; /* 预签名请求超时时间，单位为秒，最大为7天,这里和v2的预签名有区别
    const char *resource = NULL; /* 暂不支持子资源信息设置 */
    long long expires = -1;
    S3Status status = 0;

    if (argc < 3) {
        printf("please input bucket/key/expires/http_method\n");
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    /* 转换时间 */
    char_expires = argv[3];
    sscanf(char_expires, "%lld", &expires);

    http_method = argv[4];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
```

```
        access_key,
        secret_key
    };

    char buffer[S3_MAX_AUTHENTICATED_QUERY_STRING_SIZE];

    status = S3_generate_authenticated_query_string_v4(buffer,
        &bucketContext, key, expires, resource, http_method);

    if (status != S3StatusOK) {
        printf("Failed to generate authenticated query string: %s\n",
               S3_get_status_name(status));
    }
    else {
        printf("%s\n", buffer);
    }

    S3_deinitialize();

    return status;
}
```

设置对象LegalHold

SetObjectLegalHold

设置对象LegalHold,在指定对象上使用依法保留配置。（前提：需要对象已开启obeckt-lock才可以设置成功）

api接口

```
void S3_set_objcet_legal(const S3BucketContext *bucketContext,
                         const char *json_data, const char *version, const char *key,
                         S3RequestContext *requestContext,
                         const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	LegalHold配置json文件	是
version	const char *	如对象存在版本号，需要指定版本信息	否
key	const char *	需要设置的key	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int set_object_legal_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag =  "{"
          "'Status': 'ON'" /* ON/OFF,对应开启和关闭 */
        "}";

    if (argc < 2) {
        printf("please inpur your bucket && key\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
```

```
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_set_objcet_legal(&bucketContext, tag, version, key,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

获取对象LegalHold

GetObjectLegalHold

请求获取指定对象的当前依法保留状态

api接口

```
void S3_get_objcet_legal(const S3BucketContext *bucketContext,
                         const char *version, char *data, int data_len,const char *key,
                         S3RequestContext *requestContext,
                         const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是

参数	类型	说明	是否必要
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
key	const char *	需要访问的key	是
version	const char *	如对象存在版本号，需要指定版本信息	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在获取改对象不存在

```c
int get_object_legal_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
```

```
        &responseCompleteCallback
    };

    S3_get_objcet_legal(&bucketContext, version, data,
        sizeof(data), key, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

设置对象Retention

SetObjectRetention

设置对象保留期限配置。

api接口

```
void S3_set_objcet_retention(const S3BucketContext *bucketContext,
                         const char *json_data, const char *version, const char *key,
                         S3RequestContext *requestContext,
                         const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	Retention配置json文件	是

参数	类型	说明	是否必要
version	const char *	如对象存在版本号，需要指定版本信息	否
key	const char *	需要设置的key	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int set_object_retention_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *tag = "{"
          "'Mode': 'GOVERNANCE',"
          /* 超时格式按示例格式给出 xxxx-xx-xxTxx:xx:xxZ */
          "'RetainUntilDate': '2024-04-30T20:36:14Z'"
        "}";

    if (argc < 2) {
        printf("please inpur your bucket && key\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
```

```
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_set_objcet_retention(&bucketContext, tag, version, key,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

获取对象Retention

GetObjectRetention

获取对象的保留期限设置

api接口

```
void S3_get_objcet_retention(const S3BucketContext *bucketContext,
                             const char *version, char *data, int data_len,const char *key,
                             S3RequestContext *requestContext,
                             const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const	bucket及请求参数	是

参数	类型	说明	是否必要
	S3BucketContext *		
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
key	const char *	需要访问的key	是
version	const char *	如对象存在版本号，需要指定版本信息	否
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在获取改对象不存在

示例

```c
int get_object_retention_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
```

```
        &responseCompleteCallback
    };

    S3_get_objcet_retention(&bucketContext, version, data,
        sizeof(data), key, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

对象文件在线解压

PostObjectUnzip

PostObjectUnzip 接口可将您桶中的ZIP压缩文件进行在线解压，并保存到指定的桶的路径中。

api接口

```
void S3_post_object_unzip(const S3BucketContext *bucketContext,
                    const char *key, char *json,
                    S3RequestContext *requestContext,
                    const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
json_data	const char *	在线解压的配置json文件	是

参数	类型	说明	是否必要
key	const char *	需要访问的key	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

在线解压的json 组装格式为

```
    char *json = "{"
        "'DestBucket': 'zip',"
        "'DestPath': 'zip_test2',"
        "'KeepZipFileName': 'false | true',"
        "'Overwrite': '2 | 1 | 0'"
    "}";
```

参数	说明
DestBucket	解压后的文件保存的桶,解压的key源的桶不要和目标桶都属于一个桶
DestPath	解压后的文件保存的路径
KeepZipFileName	是否需要保存zip的文件名称
Overwrite	如果存在同名的文件，执行哪种操作？ 0: 覆盖 1: 跳过 2: 重命名

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```
int post_object_unzip_test(int argc, char **argv)
{
    S3Status status;
    const char *bucket = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *key = NULL;
    int s3_status = 0;
    /* 解压配置文件 */
    char *json = "{"
        "'DestBucket': 'zip',"
        "'DestPath': 'zip_test2',"
        "'KeepZipFileName': 'false',"
        "'Overwrite': '2'"
    "}";

    if (argc < 2) {
       printf("please input src bucket && src key\n");
       return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
```

```
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_post_object_unzip(&bucketContext, key, json, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

在线解压查询接口

getObjectUnzip

getObjectUnzip 可查询你之前下发的在线解压任务的执行进度。

api接口

```
void S3_get_unzip(const S3BucketContext *bucketContext,
                        const char *key, char *data, int data_len, char *persistent,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const	bucket及请求参数	是

参数	类型	说明	是否必要
	S3BucketContext *		
requestContext	S3RequestContext *	请求参数	否
key	const char *	需要访问的key	是
data	char *data	get的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
persistent	chat *	持久化ID，在您下发在线解压任务后，服务端会给您返回	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int get_object_unzip_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *key = NULL;
    char *persistent = NULL;
    S3Status status;
    int s3_status = 0;
    char data[64 * 1024];

    if (argc < 3) {
        printf("please inpur your bucket && key && persistent id\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];
    persistent = argv[3];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
```

```
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_get_unzip(&bucketContext, key, data, sizeof(data), persistent, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

在线压缩功能

PostPackage

多文件压缩，为用户提供批量文件的打包压缩存储功能。用户通过指定存储桶中的一系列对象，即可将ZOS中存储的资源文件，在ZOS服务端打包压缩后存储。用户需要指定打包压缩后的文件的存储位置以及文件名称.

api接口

```
void S3_post_package(const S3BucketContext *bucketContext,
                     const char *key, char *json,
                     S3RequestContext *requestContext,
                     const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否

参数	类型	说明	是否必要
json_data	const char *	压缩配置json文件	是
key	const char *	需要设置的key	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

其中json的组装格式为

```
SourceConfigPolicy={
    'Mode': 2,  // mode 为2，指的是用下面的SourceRules 执行文件进行压缩;
                // mode为4，指的是用SourceIndex 执行文件压缩
    'SourceRules': [
        {
            'Resource': 'bucket_12/index.html', // 需要压缩的文件
            'renamePath': 'dir1_rename/file1.txt' // 压缩以后，这个文件在压缩包中的路径
        },
        {
            'Resource': 'bucket_ver/messages',
            'renamePath': 'dir1_rename/file2.txt'
        }
        #{
        #    'Resource': 'bucket_dns/3G'
        #}
    ],
    'SourceIndex': 'bucket_web/mkzip_index.json'
}
```

参数说明

参数	说明	是否必要
Mode	mode 为2，指的是用的SourceRules 执行文件进行压缩; mode 为4，指的是用SourceIndex 执行文件压缩	是
SourceRules	压缩文件的配置信息	如mode为2，则必须

参数	说明	是否必要
Resource	需要压缩的文件	是
renamePath	压缩以后，这个文件在压缩包中的路径	否
SourceIndex	所有压缩文件配置的json文件路径，仅在mode为2时剩下	如mode为4，则必须

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

```c
int post_package_test(int argc, char **argv)
{
    S3Status status;
    const char *bucket = NULL;
    const char *key = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    /* 压缩配置json文件 */
    char *json = "{"
        "'Mode': '2',"
        "'SourceList': ["
            "{"
            "'SourceRule': ["
                    "{"
                        "'Resource': 'bucket-5f80/1.sh',"
                        "'renamePath': 'dir1_rename/file1.txt'"
                    "},"
                    "{"
                        "'Resource': 'bucket-5f80/2.sh',"
                        "'renamePath': 'dir1_rename/file2.txt'"
                    "}"
                "],"
            "'SourceIndex': 'bucket_web/mkzip_index.json'"
            "}"
        "]"
    "}";


    if (argc < 2) {
       printf("please input src bucket && src key\n");
       return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
```

```
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_post_package(&bucketContext, key, json, 0, &responseHandler, 0);
    s3_status = get_statusG();

    S3_deinitialize();

    return s3_status;
}
```

获取用户unzip任务

GetUserUnzipTask

接口用于获取用户所有的unzip任务

api接口

```
void S3_get_unzip_usertask(const S3BucketContext *bucketContext,
                           char *data, int data_len,
                           S3RequestContext *requestContext,
                           const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
data	char *	get到的数据存放的字符串数组	是
data_len	int	字符串数组长度	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在获取改对象不存在

示例

```c
int get_unzip_usertask_test(int argc, char **argv)
{
    S3Status status;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int s3_status = 0;
    char data[64 * 1024];

    if (argc || argv) {
        /* do nothing */
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        0,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_get_unzip_usertask(&bucketContext, data, sizeof(data), 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
```

```
        // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

取消unzip任务

CancelUnzipTask

接口用于取消用户已经下发的unzip任务

api接口

```
void S3_cancel_objects_unzip(const S3BucketContext *bucketContext,
                      const char *key, char *persistent,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
key	const char *	需要访问的key	是
persistent	char *	需要取消的Unzip任务的ID	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在获取改对象不存在

示例

```c
int cancel_object_unzip_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    char *persistent = NULL;
    S3Status status;
    int s3_status = 0;

    if (argc < 3) {
        printf("please inpur your bucket && key && persistent id\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];
    persistent = argv[3];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
```

```
        &responseCompleteCallback
    };

    S3_cancel_objects_unzip(&bucketContext, key,  persistent, 0, &responseHandler, 0);

    S3_deinitialize();

    return s3_status;
}
```

Restore object

RestoreObject

用于解冻归档对象数据。

api接口

```
void S3_restore_obj(const S3BucketContext *bucketContext,
                        const char *json_data, const char *key, const char *version,
                        S3RequestContext *requestContext,
                        const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
json_data	const char *	解冻参数信息	是
key	const char *	需要访问的key	是
version	const char *	如需要指定解冻对象的version，则指定	否
requestContext	S3RequestContext *	请求上下文	否
handler	const	请求的回调函数	是

参数	类型	说明	是否必要
	S3ResponseHandler *		
callbackData	void *	请求回调中的数据	否

其中解冻参数的格式如下：

```
const char *json = "{"
    "'GlacierJobParameters': ["
       "{"
         "'Tier': 'Standard'" /* 解冻模式，目前只支持Standard */
        "}"
    "],"
    "'Days':'4'"  /* 解冻之后副本的保留时间，目前支持范围是1-31 */
"}";
```

响应结果

HTTP状态	响应码	描述
202	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶不存在获取改对象不存在

示例

```c
int restore_object_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL;
    const char *version = NULL;/* 如果您开启了多版本，需要在这里指定版本信息 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    const char *json = "{"
        "'GlacierJobParameters': ["
           "{"
             "'Tier': 'Standard'"
            "}"
        "],"
        "'Days':'4'" /* 文件解冻时间 */
    "}";

    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    key = argv[2];

    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
```

```
        access_key,
        secret_key
    };

    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_restore_obj(&bucketContext, json, key, version,
        0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

文件分段上传

标准接口

分段上传操作可以将超过5GB的大文件分段后上传, 一共包括三个步骤:

1. 通过 CreateMultipartUpload 发起分段上传请求, 获取 UploadId
2. 将大文件分段上传, 每个片段数据大小为5MB~5GB, 均附带 UploadId
3. 完成分段上传, 发送一个附带 UploadId 的请求

CreateMultipartUpload

Create Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。如您没有指定对象权限，则权限默认为私有。

api接口

```
void S3_initiate_multipart(S3BucketContext *bucketContext, const char *key,
                           S3PutProperties *putProperties,
                           S3MultipartInitialHandler *handler,
                           S3RequestContext *requestContext,
                           void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	对象名称	否
putProperties	S3PutProperties *	post请求的参数	否
requestContext	S3RequestContext *	请求参数	否
handler	S3MultipartInitialHandler *	回调函数	是
callbackData	void *	回调数据	否

可在putProperties中设置本次需要上传对象的属性，包括ACL，ContentType，Metadata等信息。

返回值

名称	解释
bucket	本次操作的bucket id
key	本次操作的对象key id
uploadid	如果您init分段任务成功，则会返回upload id，后续您的分段任务依靠upload来进行标识

示例

```
int init_multpart_put_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    S3Status status;
    int s3_status = 0;
    UploadManager manager;
    manager.upload_id = 0;
    manager.gb = 0;
    if (argc < 2) {
        printf("please inpur your bucket && key \n");
        return -1;
    }
    manager.upload_id = (char *)malloc(sizeof(char) * 128);
    if (manager.upload_id == NULL) {
        printf("mem char *128 failed\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        free(manager.upload_id);
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3PutProperties putProperties =
```

```
    {
        0,
        0,
        0,
        0,
        0,
        0,
        S3CannedAclPublicReadWrite,
        0,
        0
    };
    S3MultipartInitialHandler handler = {
        {
            &responsePropertiesCallback,
            &responseCompleteCallback
        },
        &initial_multipart_callback
    };
    do {
        S3_initiate_multipart(&bucketContext, key, &putProperties,
            &handler, 0, &manager);
        s3_status = get_statusG();
    } while (S3_status_is_retryable(s3_status) && should_retry());
    if (s3_status != S3StatusOK) {
       // printError();
    }
    S3_deinitialize();
    free(manager.upload_id);
    return s3_status;
}
```

UploadPartData

UploadPartData请求实现在初始化以后的分块上传，支持的块的数量为 1 到 10000，除了最后一块，其他每块大小都必须大于或等于5M。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号。

api接口

```
void S3_upload_part(S3BucketContext *bucketContext, const char *key,
                    S3PutProperties *putProperties,
                    S3PutObjectHandler *handler, int part_num,
                    const char *upload_id, int partContentLength,
                    S3RequestContext *requestContext,
                    void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	对象名称	是
putProperties	S3PutProperties *	put请求的参数，对象的部分请求信息已经在init时初始化了，您可以在这里设置MD5信息	否
part_num	int	本次分段上传的part_num，取值1-1000	是
upload_id	const char *	在init阶段获取到uploadid	是
partContentLength	int	本次上传的数据长度	是
handler	S3PutObjectHandler *	回调函数	是
requestContext	S3PutObjectHandler *	请求参数	否
callbackData	void *	回调数据	否

返回信息如下

名称	解释
eTag	本次分段请求返回的eTag信息

需要注意的是，您本次上传需要记录您上传的part_num和eTag信息，这两个是一一对应的关系，后续分段上传完毕后需要用这两个信息来complete分段上传请求

示例

```c
int upload_multpart_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *upload_id = NULL; /* init时获取到的upload id */
    const char *filename = NULL; /* 需要上传的文件路径 */
    int part_num = 0, file_len = 0;/* 您本次上传的part_num, 文件长度 */
    S3Status status;
    int s3_status = 0;
    put_object_callback_data data;
    if (argc < 5) {
        printf("please inpur your bucket && key && uploadid && part_num && filename\n");
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    upload_id = argv[3];
    part_num = atoi(argv[4]);
    filename = argv[5];
    memset(&data, 0, sizeof(put_object_callback_data));
    printf("args bucket:%s key: %s uploadid: %s part_num: %d filedname: %s\n",
        bucket, key, upload_id, part_num, filename);
    struct stat statbuf;
    if (stat(filename, &statbuf) == -1) {
        printf("\n Error: filed to open file %s \n", filename);
        return -1;
    }
    file_len = statbuf.st_size;
    /* 打开需要上传的文件 */
    if (!(data.infile = fopen(filename, "r"))) {
        fprintf(stderr, "\n ERROR: Failed to open input file %s: ",
                filename);
        return -1;
    }
    data.contentLength = data.originalContentLength = file_len;
    /* 1. 初始化libs3 */
```

```
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        fclose(data.infile);
        exit(-1);
    }
     /* 2. s3基本信息设置 */
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3PutObjectHandler putObjectHandler =
    {
        { &MultipartResponseProperiesCallback, &responseCompleteCallback },
        &putObjectDataCallback
    };
    S3_upload_part(&bucketContext, key, 0,
        &putObjectHandler, part_num, upload_id,
        file_len, 0, &data);
    if (data.infile) {
        fclose(data.infile);
    }
    S3_deinitialize();
    return s3_status;
}
```

CompleteMultipartUpload

Complete Multipart Upload 用来实现完成整个分块上传。当您已经使用 Upload Parts 上传所有块以后，你可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
api接口

```
void S3_complete_multipart_upload(S3BucketContext *bucketContext,
                                  const char *key, S3MultipartCommitHandler *handler,
                                  const char *upload_id, int contentLength,
                                  const char *etags_json, S3RequestContext *requestContext,
                                  void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	对象名称	是
handler	S3MultipartCommitHandler *	回调函数	是
upload_id	const char *	在init阶段获取到uploadid	是
contentLength	int	callbackData的长度	否
etags_json	const char *	已经完成分段上传的part_num和eTag的集合，以json格式给出	是
requestContext	S3PutObjectHandler *	请求参数	否
callbackData	void *	回调数据	否

您需要在etags_json中组装分段上传的part_num和eTag的集合，这两个信息在upload步骤中获取。

示例

```c
int commit_multpart_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *upload_id = NULL;
    const char *etags =
    "{"
        "'Parts':"
        "["
            "{"
                "'PartNumber': '1',"
                "'ETag': 'c1175e5948750bc088bd3858e6f59006'"
            "},"
            "{"
                "'PartNumber': '2',"
                "'ETag': 'c1175e5948750bc088bd3858e6f59006'"
            "},"
            "{"
                "'PartNumber': '3',"
                "'ETag': 'c1175e5948750bc088bd3858e6f59006'"
            "},"
            "{"
                "'PartNumber': '4',"
                "'ETag': 'c1175e5948750bc088bd3858e6f59006'"
            "}"
        "]"
    "}";
    S3Status status;
    int s3_status = 0;
    if (argc < 3) {
        printf("please inpur your bucket && key && upload id\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
```

```
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    upload_id = argv[3];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3MultipartCommitHandler commit_handler = {
        {
            &responsePropertiesCallback,&responseCompleteCallback
        },
        &multipartPutXmlCallback,
        0
    };
    do {
        S3_complete_multipart_upload(&bucketContext, key, &commit_handler, upload_id,
            0, etags, 0, 0);
        s3_status = get_statusG();
    } while (S3_status_is_retryable(s3_status) && should_retry());
    S3_deinitialize();
    return s3_status;
}
```

标准扩展接口

终止分段上传

AbortMultipartUpload 操作用于终止一个分段上传. 当分段上传被中止后不会再有数据通过与之相应的 UploadId 上传, 同时已经被上传的片段所占用的空间将被释放. 由于执行 AbortMultipartUpload 操作后正在上传的片段可能上传成功也可能中止, 故在必要的情况下需要多次执行以保证所有片段都被释放. 可以通过执行 ListParts 操作来确认终止后所有上传片段是

否被释放

api接口

```
void S3_abort_multipart_upload(S3BucketContext *bucketContext, const char *key,
                               const char *uploadId,
                               S3AbortMultipartUploadHandler *handler)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	对象名称	是
upload_id	const char *	在init阶段获取到uploadid	是
handle	S3AbortMultipartUploadHandler *	回调函数	是

相应结果

http状态	响应码	描述
204	NoContent	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchUpload	操作指定的桶或对象或 UploadId 不存在

示例

```c
int abort_multpart_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *upload_id = NULL;
    S3Status status;
    int s3_status = 0;
    if (argc < 3) {
        printf("please inpur your bucket && key && upload id\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    upload_id = argv[3];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3AbortMultipartUploadHandler handle =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
    };
    S3_abort_multipart_upload(&bucketContext, key, upload_id, &handle);
    s3_status = get_statusG();
    S3_deinitialize();
    return s3_status;
```

```
}
```

列出分段上传任务

ListMultipartUploads 操作用于列出一个桶中正在进行的分段上传, 这些分段上传的请求已经发起但还未完成或被中止

api接口

```
void S3_list_multipart_uploads(S3BucketContext *bucketContext,
                               const char *prefix, const char *keymarker,
                               const char *uploadidmarker,
                               const char *encodingtype, const char *delimiter,
                               int maxuploads, S3RequestContext *requestContext,
                               const S3ListMultipartUploadsHandler *handler,
                               void *callbackData)
```

参数	类型	说明	是否必要
bucket Context	const S3Bucket Context *	源bucket及请求参数	是
prefix	const char *	与 delimiter 参数一起用于对对象进行分组的字符. 所有key包含指定的 Prefix 且第一次出现 Delimiter 字符之间的对象作为一组	否
keymar ker	const char *	和 UploadIdMarker 参数一起用于指定返回哪部分分段上传的信息. 若未设置 UploadIdMarker 参数, 则只返回对象key按字典序排于 KeyMarker 标识符后的片段信息. 若设置了 UploadIdMarker 参数, 则在前者基础上还会返回对象key等于 KeyMarker 且 UploadId 大于UploadIdMarker 的片段信息	否

参数	类型	说明	是否必要
uploadidmarker	const char *	和 KeyMarker 参数一起用于指定返回哪部分分段上传的信息. 仅当设置了 KeyMarker 参数的时候有效. 设置后在KeyMarker 的基础上将对象key等于 KeyMarker 且UploadId 大于 UploadIdMarker 的片段信息也纳入返回范围	否
encodingtype	const char *	用于设置响应中对象key的字符编码类型	否
delimiter	const char *	若设置了 Prefix , 所有包含指定 Prefix 且第一次出现Delimiter 字符的对象key作为一组. 若未指定 Prefix 参数, 按 Delimiter 对所有对象key进行分割	否
maxuploads	int	指定响应消息体中正在进行的分段上传 消息的最大数量, 默认和最大都是1000	否
requestContext	S3RequestContext *	请求上下文	否
handler	const S3ListMultipartUploadsHandler *	回调函数	否
callbackData	void *	回调数据	否

返回信息以桶中每一个在上传的分段对象为一个整体的数据，每个对象的数据包括：

变量	描述
Key	分段上传中的对象名

变量	描述
Initiated	初始化时间
UploadId	upload id
Initiator ID	初始化时的用户ID
Initiator Display Name	初始化时的用户的displayname
Owner ID	owner ID
Owner Display Name	owner display name
StorageClass	存储类别

示例

```c
int list_multpart_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    // list multpart 参数
    const char *prefix = NULL;
    const char *keymarker = NULL;
    const char *uploadidmarker = NULL;
    const char *encodingtype = NULL;
    const char *delimiter = NULL;
    int maxuploads = 1000;
    // end -/*list multpart 参数*/
    S3Status status;
    int s3_status = 0;
    if (argc < 2) {
        printf("please input your bucket !\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ListMultipartUploadsHandler listMultipartUploadsHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &listMultipartCallback
```

```
    };
    list_multiparts_callback_data data;
    memset(&data, 0, sizeof(list_multiparts_callback_data));
    if (keymarker != 0) {
        snprintf(data.nextKeyMarker, sizeof(data.nextKeyMarker), "%s",
                 keymarker);
    }
    if (uploadidmarker != 0) {
        snprintf(data.nextUploadIdMarker, sizeof(data.nextUploadIdMarker),
                 "%s", uploadidmarker);
    }
    data.uploadCount = 0;
    data.allDetails = 0;
    do {
        data.isTruncated = 0;
        S3_list_multipart_uploads(&bucketContext, prefix,
            data.nextKeyMarker,
            data.nextUploadIdMarker, encodingtype,
            delimiter, maxuploads, 0,
            &listMultipartUploadsHandler, &data);
        s3_status = get_statusG();
        if (s3_status != S3StatusOK) {
            break;
        }
    } while (data.isTruncated &&
             (!maxuploads || (data.uploadCount < maxuploads)));
    S3_deinitialize();
    return s3_status;
}
```

列出已上传的片段

ListParts 操作用于列出一个分段上传操作中已经上传完毕但还没合并的片段信息. 执行该操作需要提供桶名, 对象的key和 UploadId
api接口

```
void S3_list_parts(S3BucketContext *bucketContext, const char *key,
                   int partnumbermarker, const char *uploadid,
                   const char *encodingtype, int maxparts,
                   S3RequestContext *requestContext,
                   const S3ListPartsHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	源bucket及请求参数	是
key	const char *	对象名称	是
partnumbermarker	int	用于指定返回PartNumber大于PartNumberMarker的片段信息	否
upload_id	const char *	在init阶段获取到uploadid	是
encodingtype	const char *	编码方式	否
maxparts	int	指定返回片段信息的数量,默认和最大值均为1000	否
handle	S3AbortMultipartUploadHandler *	回调函数	是
requestContext	S3RequestContext *	请求上下文	否
callbackData	void *	回调数据	否

返回信息

参数	说明
LastModified	分段最后修改时间
PartNum	分段的PartNumber

参数	说明
eTag	分段的eTag信息
SIZE	分段大小

示例

```c
int list_part_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *upload_id = NULL;
    const char *encoding = NULL;
    int partnumbermarker = 0; /* list的分段编号的起始编号，只有分段编号大于这个数字的分段信息才会返回 */
    int maxpart = 3; /* 最多返回的分段数目 */
    S3Status status;
    int s3_status = 0;
    if (argc < 3) {
        printf("please inpur your bucket && key && upload id\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    upload_id = argv[3];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    S3ListPartsHandler listPartsHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
        &listPartsCallback
    };
```

```
    list_parts_callback_data data;
    memset(&data, 0, sizeof(list_parts_callback_data));
    if (partnumbermarker != 0) {
        snprintf(data.nextPartNumberMarker,
                 sizeof(data.nextPartNumberMarker), "%d", partnumbermarker);
    }
    data.partsCount = 0;
    data.allDetails = 0;
    data.noPrint = 0;
    do {
        data.isTruncated = 0;
        do {
            S3_list_parts(&bucketContext, key, partnumbermarker,
                upload_id, encoding, maxpart, 0, &listPartsHandler, &data);
            s3_status = get_statusG();
        } while (S3_status_is_retryable(s3_status) && should_retry());
        if (s3_status != S3StatusOK) {
            break;
        }
    } while (data.isTruncated && (!maxpart || (data.partsCount < maxpart)));
    S3_deinitialize();
    return s3_status;
}
```

分段任务碎片清理

当我们的桶中有很多分段上传任务时，如您想清理当前所有未完成的分段上传任务，可参照本章节的示例。

流程

1. list本bucket 上所有的分段上传任务
2. 逐个将本Bucket上的所有分段任务给abort

示例如下

```c
int mult_upload_clean_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    // list multpart 参数
    const char *prefix = NULL;
    const char *keymarker = NULL;
    const char *uploadidmarker = NULL;
    const char *encodingtype = NULL;
    const char *delimiter = NULL;
    int maxuploads = 5;
    // end -/*list multpart 参数*/
    S3Status status;
    int s3_status = 0, i = 0;

    if (argc < 2) {
        printf("please input your bucket !\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    bucket = argv[1];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
```

```
S3ListMultipartUploadsHandler listMultipartUploadsHandler =
{
    { &responsePropertiesCallback, &responseCompleteCallback },
    &listMultipartCallback
};

S3AbortMultipartUploadHandler handle =
{
    { &responsePropertiesCallback, &responseCompleteCallback },
};

list_multiparts_callback_data data;
memset(&data, 0, sizeof(list_multiparts_callback_data));
data.save_info = 1;
data.part_num = 0;
data.part_info = (mult_part_info_t *)malloc(sizeof(mult_part_info_t) * maxuploads);
if (!data.part_info) {
    printf("alloc key or uploads failed\n");
    goto l_end;
}

if (keymarker != 0) {
    snprintf(data.nextKeyMarker, sizeof(data.nextKeyMarker), "%s",
             keymarker);
}
if (uploadidmarker != 0) {
    snprintf(data.nextUploadIdMarker, sizeof(data.nextUploadIdMarker),
             "%s", uploadidmarker);
}

data.uploadCount = 0;
data.allDetails = 0;
do {
    data.isTruncated = 0;
    /* 先获取桶上的所有分段请求 */
    S3_list_multipart_uploads(&bucketContext, prefix,
        data.nextKeyMarker,
        data.nextUploadIdMarker, encodingtype,
        delimiter, maxuploads, 0,
```

```
            &listMultipartUploadsHandler, &data);
        s3_status = get_statusG();
        if (s3_status != S3StatusOK) {
            break;
        }

        /* 分批进行abort */
        for (i = 0; i < data.part_num; i++) {
            S3_abort_multipart_upload(&bucketContext, (const char *)data.part_info[i].keys,
                (const char *)data.part_info[i].uploads, &handle);
            memset(data.part_info[i].keys, 0, MAX_KEY_STRING_LEN);
            memset(data.part_info[i].uploads, 0, MAX_KEY_STRING_LEN);
        }
        data.part_num = 0;
    } while (data.isTruncated);

l_end:
    /* 释放内存 */
    if (!data.part_info) {
        free(data.part_info);
    }

    S3_deinitialize();

    return s3_status;
}
```

融合接口

SDK中也有封装好的融合接口，可直接一步上传大于5G的对象，方便用户实现分段上传功能api接口

```
int upload_big_file(S3BucketContext *bucketContext, const char *key,
    S3PutProperties *putProperties, const char *file, const int chunk_size_mb)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3Bucket Context *	源bucket及请求参数	是
key	const char *	对象名称	是
putProperties	S3PutProperties *	post请求的参数	否
file	const char * file	需要上传的文件路径	是
chunk_size_mb	int	如果您需要指定分段的大小, 可以通过该值进行设置, 取值5-5120, 单位为mb,默认为768mb	否

可在putProperties中设置本次需要上传对象的属性,
包括ACL，ContentType，Metadata等信息。

示例

```c
int put_big_file_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要访问的bucket */
    const char *key = NULL; /* 要访问的对象名称 */
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *file = NULL;
    int chunk_size_mb = 6;/* 如果您想要指定每个分段的大小，修改这个值，单位为Mb,取值范围为 5 - 5120;
    如果不需要，请赋值为-1 */
    S3Status status;
    int s3_status = 0;
    if (argc < 3) {
        printf("please inpur your bucket && key && file id\n");
        return -1;
    }
    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    file = argv[3];
    S3BucketContext bucketContext =
    {
        0,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
    s3_status = upload_big_file(&bucketContext, key, 0, file, chunk_size_mb);
    S3_deinitialize();
    return s3_status;
}
```

CopyPart

CopyPart 接口用户可您在初始化分段任务后，将你zos上存在的对象copy到本次分段任务中，作为分段上传任务的一个part

api接口

```
void S3_copy_part_object(const S3BucketContext *bucketContext,
                         const char *destinationKey,
                         const S3PutProperties *putProperties,
                         copy_part_param_t *copy_param,
                         int64_t *lastModifiedReturn, int eTagReturnSize,
                         char *eTagReturn, S3RequestContext *requestContext,
                         const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
destinationKey	const char *	拷贝的目标key	是
putProperties	const S3PutProperties *	上传文件的配置信息	是
copy_param	copy_part_param_t *	copy的配置信息	是
lastModifiedReturn	int64_t *lastModifiedReturn	对象返回时的lastModifiedReturn	是
eTagReturnSize	int	eTagReturn 数组的长度	是
eTagReturn	char *	对象copy成功后，返回的etag信息存放数组	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

其中copy_part_param_t结构体的说明如下:

```
typedef struct copy_part_param {
    /* 只有当源对象的etag等于该值时才copy，否则返回412错误，非必须填写值 */
    char *CopySourceIfMatch;
     /* 只有当对象自指定时间以来已被修改时才copy,非必须填写值 */
    char *CopySourceIfModifiedSince;
    /* 只有当源对象的etag等于该值时才copy，否则返回304错误，非必须填写值 */
    char *CopySourceIfNoneMatch;
    /* 只有当对象自指定时间以来未被修改时才copy,非必须填写值 */
    char *CopySourceIfUnmodifiedSince;
    /* copy源对象的范围，必须填写 */
    char *CopySourceRange;
    /* copy源对象的路径，bucket/key 格式，必须填写 */
    char *copySource;
    /* copy到目标分段任务对象的upload id，  必须填写 */
    char *upload_id;
     /* copy到目标对象时的part_num，必须填写 */
    char *part_num;
} copy_part_param_t;
```

示例

```c
int copy_part_upload_test(int argc, char **argv)
{
    const char *dst_bucket = NULL, *dst_key = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    char *part_num = NULL, *upload_id = NULL;
    int s3_status = 0;
    S3Status status;
    int64_t lastModified = 0;
    char eTag[256];

    if (argc < 4) {
        printf("please input your dst_bucket / dst_key / part_num / upload_id\n");
        return -1;
    }

    dst_bucket = argv[1];
    dst_key = argv[2];
    part_num = argv[3];
    upload_id = argv[4];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    /* S3请求基本信息 */
    S3BucketContext bucketContext =
    {
        0,
        dst_bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
```

```
copy_part_param_t copy_part_param =
{
    NULL,                       // CopySourceIfMatch;
    NULL,                       // CopySourceIfModifiedSince;
    NULL,                       // CopySourceIfNoneMatch;
    NULL,                       // CopySourceIfUnmodifiedSince;
    "bytes=0-1024",              // CopySourceRange;
    "bucket-5f80/copy_python",  // copySource;  bucket/key 在这里输入copy 源bucket 和 key
    upload_id,                  // upload_id;
    part_num                    // part_num;
};

extra_info_t extra_info =
{
    NULL,
    NULL,
    &copy_part_param
};

S3PutProperties putProperties =
{
    0,0,0,0,0,-1,0,0,0,0,0,
    &extra_info
};

/* 回调信息 */
S3ResponseHandler responseHandler =
{
    &responsePropertiesCallback,
    &responseCompleteCallback
};

/* 调用S3接口拷贝对象 */
S3_copy_part_object(&bucketContext, dst_key, &putProperties, &copy_part_param,
    &lastModified, sizeof(eTag), eTag, 0, &responseHandler, 0);
s3_status = get_statusG();

/* 打印目标对象的信息 */
if (s3_status == S3StatusOK) {
```

```
        if (lastModified >= 0) {
            char timebuf[256];
            time_t t = (time_t) lastModified;
            strftime(timebuf, sizeof(timebuf), "%Y-%m-%dT%H:%M:%SZ",
                     gmtime(&t));
            printf("Last-Modified: %s\n", timebuf);
        }
        if (eTag[0]) {
            printf("ETag: %s\n", eTag);
        }
    }
    else {
       // printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

图片处理

图片处理方法

ZosProcess 的定义图片处理的方法

图片缩放

(e.g. image/resize,w_300,h_200,m_fixed)

参数名称	参数用途	取值	是否必要
w	指定目标缩放图宽度	[1,4096]	使用按百分比缩放可不指定宽高
h	指定目标缩放图高度	[1,4096]	使用按百分比缩放可不

参数名称	参数用途	取值	是否必要
			指定宽高
m	指定缩放模式	lfit（默认值）：等比缩放，目标缩放图为指定 w 和 h 矩形框内的最大图形。 mfit：等比缩放，目标缩放图为延伸出指定 w 和 h 矩形框外的最小图形。 fill：将原图等比缩放为延伸出指定 w 与 h 的矩形框外的最小图片，之后将超出的部分进行居中裁剪。 pad: 将原图等比缩放为指定 w 和h 矩形框内最大的图形，然后使用color 指定的颜色将矩形框内剩余部分进行填充。 fixed: 固定宽高，强制缩放	否
color	缩放模式为 pad 时， 指定填充颜色	RGB 颜色值， 默认 FFFFFF(白色)	否（仅当 m 为 pad 模式时使用）
p	按百分比进行缩放	[1,1000]小于 100 缩小，大于 100 放大	否
limit	指定目标缩放图大于原图时是否缩放	1(默认)：目标缩放图大于原图时返回原图 0：按指定参数缩放	否

图片裁剪

(e.g. image/crop,w_100,h_100,x_10,y_10,g_se)

参数名称	参数用途	取值	是否必要
w	指定裁剪宽度。	[0,图片宽度] 默认为最大值。	否
h	指定裁剪高度。	[0,图片高度] 默认为最大值。	否

参数名称	参数用途	取值	是否必要
x	指定裁剪起点横坐标（默认左上角为原点）。	[0,图片边界]	否
y	指定裁剪起点纵坐标（默认左上角为原点）。	[0,图片边界]	否
g	设置裁剪的原点位置。原点按照九宫格的形式分布，一共有九个位置可以设置，为每个九宫格的左上角顶点。	nw：左上(默认) north：中上 ne：右上 west：左中 center：中部 east：右中 sw：左下 south：中 se：右下	否

图片旋转

(e.g. image/rotate,45)

参数名称	参数用途	取值	是否必要
[value]	图片按顺时针旋转的角度。	[0,360]默认值：0，表示不旋转。	是

水印

图片水印(e.g. image/watermark,image_aHVkaWUuanBnP3gtem9zLXByb2Nlc3M9aW1hZ2UvcmVzaXplLHBfMzAvcm90YXRlLDE4MA==,g_north,t_40)

文字水印(e.g. image/watermark,text_Q2hpbmF0ZWxlY29t,type_heiti,color_FF0000,size_40,g_se,t_80)

参数名称	参数用途	取值	是否必要
t	图片水印或文字水印的透明	[0, 100]	否

参数名称	参数用途	取值	是否必要
	度		
x	文字水印距 离图片边界的水 平距离	[0, 4096] 默认值：10	否
y	文字水印距 离图片边界的垂 直距离	[0, 4096] 默认值：10	否
text	指定文字 水印内容	Base64 编码后的字符串， 编码结果字符串中 ‘/’要替换为‘_’	否
color	指定文字 水印的颜色	RGB 颜色值。 默认：FFFFFF（白色）	否
size	指定文字水 印的字体大小	默认值：40	否
type	指定文字水 印的字体类型	如 Airal, Helvetica, 支持中文字体包括 yahei(微软雅黑)，heiti(黑体)， kaishu（楷书）， youyuan(幼圆)	否
rotate	指定文字 水印顺时针旋转角度	[0, 360]默认值：0	否
image	指定图片水 印名称，水印图片需要 和原图存放在相同 存储空间	水印图片可以 进行预处理（e.g. 水印图片缩放为 30%并旋转 180 度， x-zos-process= image/resize, p_30/rotate,180 ），需要转换成	否

参数名称	参数用途	取值	是否必要
		base64 编码， 编码结果字符串 中‘/’要替换为‘_’	
g	设置裁剪的原点 位置。原点按照九宫格的 形式分布，一共有九 个位置可以设置， 为每个九宫格的左上角 顶点。	nw：左上(默认) north：中上 ne：右上 west：左中 center：中部 east：右中 sw：左下 south：中 se：右下	否

格式转化

(e.g. image/format,png)

参数名称	参数用途	取值	是否必要
[value]	将原图转换成指定格式	Jpg、png、webp、bmp、tiff	是

获取图片信息

(e.g. image/info)

Post请求处理图片接口

该功能是对存储桶中的图片进行处理，并将处理后的图片持久化到指定的存储桶中，支持处理图片格式 JPG、PNG、WEBP、BMP、TIFF。

api接口

```
void S3_process_image(const S3BucketContext *bucketContext,
                      const char *process, const char *version, const char *key,
                      const char *dst_bucket, const char *dst_key,
                      S3RequestContext *requestContext,
                      const S3ResponseHandler *handler, void *callbackData)
```

参数说明

参数	类型	说明	是否必要
bucketContext	const S3BucketContext *	bucket及请求参数	是
requestContext	S3RequestContext *	请求参数	否
process	const char *	图片处理操作	是
version	const char *	如对象存在版本号，需要指定版本信息	否
key	const char *	需要设置的key	是
dst_bucket	const char *	图片处理完成后存放的桶	是
dst_key	const char *	图片处理完成后存放key名称	是
handler	const S3ResponseHandler *	请求的回调函数	是
callbackData	void *	请求回调中的数据	否

其中process 的请求格式示例为

```
const char *process = "image/rotate,45";
```

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int post_image_test(int argc, char **argv)
{
    S3Status status;
    int s3_status = 0;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    const char *src_bucket;
    const char *src_key;
    const char *dst_bucket;
    const char *dst_key;
    const char *process = "image/rotate,45";
    const char *version = NULL;

    if (argc < 4) {
        printf("please input your src bucket/key, dst bucket/key\n");
        return -1;
    }

    src_bucket = argv[1];
    src_key = argv[2];
    dst_bucket = argv[3];
    dst_key = argv[4];

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        return -1;
    }

    S3BucketContext bucketContext =
    {
        0,
        dst_bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };
```

```
    S3ResponseHandler responseHandler =
    {
        &responsePropertiesCallback,
        &responseCompleteCallback
    };

    S3_process_image(&bucketContext, process,
        version, dst_key, src_bucket, src_key, 0, &responseHandler, 0);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
     //   printError();
    }

    S3_deinitialize();

    return s3_status;
}
```

get请求处理图片接口

图片处理是在 get_object 基础进行的扩展，用于对存储桶中的图片文件在线处理，支持处理图片格式 JPG、PNG、WEBP、BMP、TIFF。

api接口

```
void S3_get_object(const S3BucketContext *bucketContext, const char *key,
                   const S3GetConditions *getConditions,
                   uint64_t startByte, uint64_t byteCount,
                   S3RequestContext *requestContext,
                   const S3GetObjectHandler *handler, void *callbackData)
```

图片处理的接口沿用get_object接口的参数，唯一有变化的是需要在getConditions中增加图片处理操作的指令信息

响应结果

HTTP状态	响应码	描述
200	Success	操作成功
403	AccessDenied	用户没有权限执行操作
404	NoSuchBucket	该桶或者该对象不存在

示例

```c
int get_object_image_test(int argc, char **argv)
{
    const char *bucket = NULL; /* 要获取对象的桶 */
    const char *key = NULL; /* 桶内的对象的key */
    const char *filename = NULL;
    const char hostname[] = ""; /* your host */
    const char access_key[] = ""; /* your ak */
    const char secret_key[] = ""; /* your sk */
    int64_t ifModifiedSince = -1, ifNotModifiedSince = -1;
    const char *ifMatch = NULL, *ifNotMatch = NULL;
    unsigned long long startByte = 0;
    unsigned long long byteCount = 0; /* 设置为0即代码获取整个对象 */
    FILE *outfile = 0;
    int s3_status = 0;
    S3Status status;

    printf("%llu\n", byteCount);

    if (argc < 3) {
        printf("please input bucket/key/filename\n");
        return -1;
    }
    bucket = argv[1];
    key = argv[2];
    filename = argv[3];

    /* 先测试给出的文件是否可以打开，如文件不存在，则直接创建一个 */
    if (filename) {
        struct stat buf;
        if (stat(filename, &buf) == -1) {
            outfile = fopen(filename, "w");
        }
        else {
            outfile = fopen(filename, "r+");
        }

        if (!outfile) {
            fprintf(stderr, "\nERROR: Failed to open output file %s: ",
                    filename);
```

```
            return -1;
        }
    } else {
        printf("filename is null. please input object out file!!\n");
        return -1;
    }

    /* 1. 初始化libs3 */
    if ((status = S3_initialize("s3", S3_INIT_ALL, hostname)) != S3StatusOK) {
        fprintf(stderr, "Failed to initialize libs3: %s\n", S3_get_status_name(status));
        fclose(outfile);
        return -1;
    }

     /* 2. s3基本信息设置 */
    S3BucketContext bucketContext =
    {
        hostname,
        bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
    };

    /* 3. 获取对象的条件 */
    S3GetConditions getConditions =
    {
        ifModifiedSince,
        ifNotModifiedSince,
        ifMatch,
        ifNotMatch,
        "image/rotate,45", /* 这里如果设值了，get Object即为图片GET接口 */
    };

    /* 4. 回调函数 */
    S3GetObjectHandler getObjectHandler =
    {
        { &responsePropertiesCallback, &responseCompleteCallback },
```

```
        &getObjectDataCallback
    };

    /* 调用获取对象接口 */
    S3_get_object(&bucketContext, key, &getConditions, startByte,
        byteCount, 0, &getObjectHandler, outfile);
    s3_status = get_statusG();

    if (s3_status != S3StatusOK) {
       // printError();
    } else {
        printf("\n get bucket/obj %s / %s successfully, path %s  \n", bucket, key, filename);
    }

    fclose(outfile);

    S3_deinitialize();

    return s3_status;
}
```