对象存储服务

C SDK 开发指南

文档版本	03
发布日期	2024-04-29

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 SDK 下载

- OBS C SDK最新版本源码：**最新源码下载**

2 SDK 编译

获取到**SDK源码**后，根据使用平台来开展编译工作。

- linux下：

linux可以直接进入到**source/eSDK_OBS_API/eSDK_OBS_API_C++/下执行下面的脚本来编译(x86和arm略有不同请注意)：**

export SPDLOG_VERSION=spdlog-1.9.2

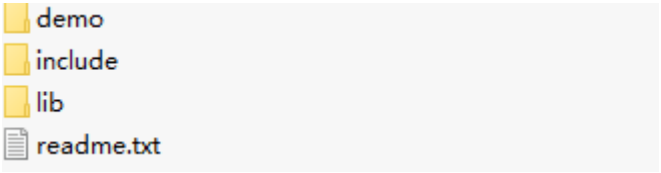
#x86执行这条命令

bash build.sh sdk

#arm执行这条命令

bash build_aarch.sh sdk

具体参数可见脚本内注释，生成产物为一个包含了demo代码，include，和lib的demo包。



- Windows下：

使用visual studio打开**source**/**eSDK_OBS_API**/**eSDK_OBS_API_C++**/**sln**/vc100/下的sln工程，生成obs项目，即可在输出目录（可在工程属性中查看）生成huaweisecurec.lib，huaweisecurec.dll,libeSDKOBS.lib和libeSDKOBS.dll。

- MAC下：

参照**compile_for_macos.txt**

常见编译问题见：**常见编译问题**

3 兼容性

- 3.*.* 相对于 3.0.0兼容
- 3.*.* 不兼容 2.*.*
- 3.*.* 不兼容 1.*.*
- arm编译环境：

```
NAME="EulerOS"
VERSION="2.0 (SP8)"
ID="euleros"
ID_LIKE="rhel fedora centos"
VERSION_ID="2.0"
PRETTY_NAME="EulerOS 2.0 (SP8)"
ANSI_COLOR="0;31"
```

内核版本：

```
4.19.36-vhulk1905.1.0.h276.eulerosv2r8.aarch64
```

gcc/g++版本：

```
gcc (GCC) 10.3.0/g++ (GCC) 10.3.0
```

- linux编译环境：

```
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

内核版本：

```
3.10.0-957.5.1.el7.x86_64
```

gcc/g++版本：

```
gcc (GCC) 10.3.0/g++ (GCC) 10.3.0
```

说明

SDK二进制包编译环境如上，其余内核和操作系统的兼容性不做保证。如果有其它操作系统或者内核版本的需求，请使用开源代码自行编译。

4 快速入门

4.1 使用前需知

须知

开发过程中，您有任何问题可以在github上**提交issue**。

- 请确认您已经熟悉OBS的基本概念，如**桶（Bucket）**、**对象（Object）**、**访问密钥（AK和SK）**等。
- 使用OBS客户端进行接口调用操作完成后，没有返回异常，则表明接口调用成功；如果返回异常，则说明操作失败，此时应从**SDK错误处理**中获取错误信息。
- 当前各区域特性开放不一致，部分特性只在部分区域开放，使用过程中如果接口HTTP状态码为405，请确认该区域是否支持该功能特性。

4.2 OBS 服务环境搭建

须知

开发过程中，您有任何问题可以在github上**提交issue**。

步骤1 注册云服务账号

使用OBS之前必须要有一个云服务账号。

1. 打开浏览器。
2. 登录公有云网站**www.huaweicloud.com/intl/zh-cn**。
3. 在页面右上角单击“注册”。
4. 按需填写注册信息并单击“同意协议并注册”。

步骤2 开通OBS服务

使用OBS服务之前必须先充值，才能正常使用OBS服务。

1. 登录OBS管理控制台。
2. 单击页面右上角的“费用”进入费用中心页面。
3. 单击“充值”，系统自动跳转到充值窗口。
4. 根据界面提示信息，对账户进行充值。
5. 充值成功后，关闭充值窗口，返回管理控制台首页。
6. 单击“对象存储服务”，开通并进入OBS管理控制台。

步骤3 创建访问密钥

OBS通过用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。以下是对AK和SK的解释说明：

- AK：Access Key ID，接入键标识，用户在对象存储服务系统中的接入键标识，一个接入键标识唯一对应一个用户，一个用户可以同时拥有多个接入键标识。对象存储服务系统通过接入键标识识别访问系统的用户。
- SK：Secret Access Key，安全接入键，用户在对象存储服务系统中的安全接入键，是用户访问对象存储服务系统的密钥，用户根据安全接入键和请求头域生成鉴权信息。安全接入键和接入键标识一一对应。

访问密钥分永久访问密钥（AK/SK）和临时访问密钥（AK/SK和SecurityToken）两种。每个用户最多可创建两个有效的永久访问密钥。临时访问密钥只在设置的有效期内能够访问OBS，过期后需要重新获取。出于安全性考虑，建议您使用临时访问密钥访问OBS，或使用永久访问密钥访问OBS时，定期更新您的访问密钥（AK/SK）。两种密钥的获取方式如下。

- 永久访问密钥：
  a. 登录OBS控制台。
  b. 单击页面右上角的用户名，并选择“我的凭证”。
  c. 在“我的凭证”页面，单击左侧导航栏的“访问密钥”。
  d. 在“访问密钥”页面，单击“新增访问密钥”。
  e. 在弹出的“新增访问密钥”对话框中，输入登录密码和对应验证码。

     说明

     - 用户如果未绑定邮箱和手机，则只需输入登录密码。
     - 用户如果同时绑定了邮箱和手机，可以选择其中一种方式进行验证。

  f. 单击“确定”。
  g. 在弹出的“下载确认”提示框中，单击“确定”后，密钥会直接保存到浏览器默认的下载文件夹中。
  h. 打开下载下来的“credentials.csv”文件既可获取到访问密钥（AK和SK）。

     说明

     - 每个用户最多可创建两个有效的访问密钥。
     - 为防止访问密钥泄露，建议您将其保存到安全的位置。如果用户在此提示框中单击“取消”，则不会下载密钥，后续也将无法重新下载。如果需要使用访问密钥，可以重新创建新的访问密钥。
- 临时访问密钥：

临时AK/SK和SecurityToken是系统颁发给用户的临时访问令牌，通过接口设置有效期，范围为15分钟至24小时，过期后需要重新获取。临时AK/SK和SecurityToken遵循权限最小化原则。使用临时AK/SK鉴权时，临时AK/SK和SecurityToken必须同时使用。

获取临时访问密钥的接口请参考**获取临时AK/SK和securitytoken**。

须知

OBS属于全局级服务，所以在获取临时访问密钥时，需要设置Token的使用范围取值为domain，表示获取的Token可以作用于全局服务，全局服务不区分项目或者区域。

----结束

4.3 服务地址

须知

开发过程中，您有任何问题可以在github上**提交issue**。

- 您可以从**这里**查看OBS当前开通的服务地址和区域信息。

4.4 初始化 SDK

详见**初始化SDK**

4.5 初始化 option

须知

开发过程中，您有任何问题可以在github上**提交issue**。

在调用C SDK的功能函数时，都要传入option参数，您可通过init_obs_options函数初始化option配置，通过option设置AK、SK、Endpoint、bucket、超时时间、临时鉴权：

- 永久访问密钥（AK/SK）创建并初始化option如下：

```
// 创建并初始化option
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
option.bucket_options.bucket_name = "<Your bucketname>";

// 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
// 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
```

```
intl/zh-cn/usermanual-ca/ca_01_0003.html
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```

- 临时访问密钥（AK/SK和SecurityToken）创建并初始化option如下：

```
// 创建并初始化option
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
option.bucket_options.bucket_name = "<Your bucketname>";

// 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
// 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
option.bucket_options.token = getenv("SecurityToken");
```

须知

OBS属于全局级服务，所以在获取临时访问密钥时，需要设置Token的使用范围取值为domain，表示获取的Token可以作用于全局服务，全局服务不区分项目或者区域。

4.6 创建桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。

桶是OBS全局命名空间，相当于数据的容器、文件系统的根目录，可以存储若干对象。以下代码展示如何新建一个桶：

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");


    // 设置obs.xxx.com响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };

    // 创建桶，<预定义访问策略>值参考5.5 管理桶访问权限
    create_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
```

```
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。桶命名规则如下：

- 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
- 禁止使用IP地址。
- 禁止以“-”或“.”开头及结尾。
- 禁止两个“.”相邻（如：“my..bucket”）。
- 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 同一用户多次创建同名桶不会报错，创建的桶属性以第一次请求为准。

本示例创建的桶的访问权限默认是私有读写，存储类型默认是标准类型，区域位置为全局域名所在的默认区域。

更多创建桶的信息，请参见**创建桶**。

须知

- 创建桶时，如果使用的终端节点归属于默认区域华北-北京一（cn-north-1），则可以不指定区域；如果使用的终端节点归属于其他区域，则必须指定区域，且指定的区域必须与终端节点归属的区域一致。当前有效的区域名称可从**这里**查询。比如初始化时使用的终端节点EndPoint是obs. ap-southeast-1.myhuaweicloud.com，那么在创建桶的时候必须指定Location： ap-southeast-1才会创建成功，否则会返回状态码400。

4.7 上传对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

数据流保存到callback_data（参见**流式上传**中的参数描述）中，使用结构体obs_put_object_handler中定义的回调函数put_object_data_callback把上传对象的内容拷贝到该回调函数的参数字符指针参数buffer中。以下代码展示了如何进行对象上传：

```
static void test_put_object_from_buffer()
{
    // 待上传buffer
    char *buffer = "abcdefg";
    // 待上传buffer的长度
    int buffer_size = strlen(buffer);
    // 上传的对象名
    char *key = "put_buffer_test";

    // 初始化option
```

```
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucket name>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //自定义存储上传数据的结构体,
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // 把buffer赋值到上传数据结构中
    data.put_buffer = buffer;
    // 设置buffersize
    data.buffer_size = buffer_size;
    // 设置回调函数,需要实现对应的回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    put_object(&option, key, buffer_size, &put_properties,0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from buffer successfully. \n");
    }
    else
    {
        printf("put object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

更多上传对象的信息，请参见**上传对象**。

4.8 下载对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

以下代码展示了如何进行对象下载：

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
```



```
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置下载的对象
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    //根据业务需要,自定义存放下载对象数据的结构
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // 定义范围下载参数
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    // 自定义下载的回调函数
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };
    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

说明

更多下载对象的信息，请参见**下载对象**。

4.9 列举对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

以下代码展示了如何列举对象：

```
static void test_list_bucket_objects()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_list_objects_handler list_bucket_objects_handler =
    {
        { &response_properties_callback, &list_objects_complete_callback },
        &list_objects_callback
```

```
    };

    // 用户自定义回调数据
    list_bucket_callback_data data;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    // 列举对象
    list_bucket_objects(&option, "<prefix>", "<marker>", "<delimiter>", "<maxkeys>",
&list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list bucket objects successfully. \n");
    }
    else
    {
        printf("list bucket objects failed(%s).\n",
            obs_get_status_name(data.ret_status));
    }
}
```

说明

- 更多列举对象的信息，请参见：**列举对象**

4.10 删除对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

以下代码展示了如何删除对象：

```
static void test_delete_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<Your Key>";
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存
放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境
变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 删除对象
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
    {
```

```
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

- 更多删除对象的信息，请参见：**删除对象**

5 初始化

5.1 配置密钥

须知

开发过程中，您有任何问题可以在github上**提交issue**。

要接入OBS服务，您需要拥有一组有效的访问密钥（AK和SK）用来进行签名认证。具体可参考**OBS服务环境搭建**。

获取AK和SK之后，您便可以按照以下步骤进行初始化。

5.2 初始化 SDK

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS客户端（ObsClient）是访问OBS服务的C客户端，它为调用者提供一系列与OBS服务进行交互的接口，用于管理、操作桶（Bucket）和对象（Object）等OBS服务上的资源。使用OBS C SDK发起OBS请求，您需要先调用初始化接口，在进程退出的时候调用取消初始化的接口，释放资源。

在使用C SDK前要先调用初始化接口obs_initialize，而且主进程中只能调用一次：

```
obs_status  ret_status = OBS_STATUS_BUTT;
ret_status = obs_initialize(OBS_INIT_ALL);
if (OBS_STATUS_OK != ret_status)
{
    printf("obs_initialize failed(%s).\n", obs_get_status_name(ret_status));
    return ret_status;
}
```

5.3 配置 option

须知

开发过程中，您有任何问题可以在github上**提交issue**。

在调用C SDK的功能函数时，都要传入obs_options参数，您可通过init_obs_options函数初始化obs_options配置，通过obs_options设置AK、SK、Endpoint、bucket、超时时间、临时鉴权。obs_options主要包括obs_bucket_context和obs_http_request_option两个结构，可以设置的参数见下表：

表 5-1 obs_options.obs_bucket_context 参数

参数	描述	默认值	建议值
host_name	请求使用的主机名，是指存放资源的服务器的域名，就是终端节点endpoint。	NULL	-
bucket_name	操作的桶名。	NULL	-
protocol	请求使用的协议类型: http、https。（出于安全性考虑，建议使用https协议）	HTTPS协议：OBS_PROTOCOL_HTTPS	OBS_PROTOCOL_HTTPS
access_key	连接对象存储服务的AK	NULL	-
secret_access_key	鉴权使用的SK，可用于字符串的签名。	NULL	-
obs_storage_class	在PUT，POST请求中，需要配置存储类型时设置此参数。	标准存储：OBS_STORAGE_CLASS_STANDARD	默认值
token	临时访问密钥的SecurityToken。	NULL	-
bucket_type	创桶时，指定是对象桶还是并行文件系统	对象桶：OBS_BUCKET_OBJECT	-
bucket_list_type	列举桶时，确定列举桶的类型：所有桶、对象桶、并行文件系统	所有桶：OBS_BUCKET_LIST_ALL	-

表 5-2 obs_options.obs_http_request_option 参数

参数	描述	默认值	建议值
connect_time	建立HTTP/HTTPS连接的超时时间（单位：毫秒）。默认为60000毫秒。	60000	[10000, 60000]
max_connected_time	请求超时时间（单位：秒）。0代表永远不会断开链接。	0	0
proxy_auth	代理认证信息，格式username:password	NULL	-
proxy_host	代理服务器	NULL	-

说明

如网络状况不佳，建议增大connect_time和max_connected_time的值。

5.4 配置 SDK 日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS C SDK的日志路径是通过OBS.ini中LogPath字段指定的，日志默认生成于与C SDK动态库lib目录同级的logs目录中，OBS.ini文件应与动态库（libeSDKLogAPI.so）同一目录。

OBS C SDK支持通过set_obs_log_path来指定日志路径，该方法存在两个参数，第一个参数为指定的路径，第二个参数为判断设定路径方式的信号，当该信号为True时，SDK将在第一个参数给定的路径下寻找OBS.ini进行日志配置，当该信号为Fasle时，SDK将在第一个参数给定的路径下生成OBS.ini和日志文件。

6 管理桶

6.1 创建桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过create_bucket或create_bucket_with_params创建一个对象桶，通过create_pfs_bucket创建一个并行文件系统的桶。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
canned_acl	obs_canned_acl，**管理桶访问权限**	必选	权限控制策略。
location_constraint	char *	可选	桶的区域位置；该参数定义了桶将会被创建在哪个区域，如果使用的终端节点是obs.myhuaweicloud.com，可以不携带此参数；如果使用的终端节点不是obs.myhuaweicloud.com，则必须携带此参数。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

字段名	类型	约束	说明
param	obs_create_bucket_params *	必选	仅限create_bucket_with_params接口使用。 结构体包含： ● obs_canned_acl canned_acl：权限控制策略 ● obs_az_redundancy az_redundancy：多AZ或单AZ – 单AZ: OBS_REDUNDANCY_1AZ – 3AZ: OBS_REDUNDANCY_3AZ ● const char *location_constraint：桶的区域位置，要创建3AZ的桶一定要指定支持3AZ的region

示例代码

以下代码展示如何新建一个对象桶：

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建桶，<预定义访问策略>值参考5.5管理桶访问权限
    create_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
```

```
    }
}
```

以下代码展示如何新建一个3AZ的对象桶：

```
static void test_create_3az_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。      // 您可以登录访问管理控制台获取访问密钥AK/SK
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建桶，<预定义访问策略>值参考5.5管理桶访问权限
    obs_create_bucket_params create_param;
    create_param.canned_acl = canned_acl;
    create_param.location_constraint = "Region name that support 3AZ";
    create_param.az_redundancy = OBS_REDUNDANCY_3AZ;
    create_bucket_with_params(&option, &create_param, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket with params successfully. \n");
    }
    else
    {
        printf("create bucket with params failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

以下代码展示如何新建一个并行文件系统的桶：

```
static void test_create_pfs_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint;
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。      // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建桶，<预定义访问策略>值参考5.5管理桶访问权限
    create_pfs_bucket(&option, "<bucket ACL>", NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
```



```
    }
}
```

说明

桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。桶命名规则如下：

- 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
- 禁止使用IP地址。
- 禁止以“-”或“.”开头及结尾。
- 禁止两个“.”相邻（如：“my..bucket”）。
- 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 同一用户在同一个区域多次创建同名桶不会报错，创建的桶属性以第一次请求为准。

本示例创建的桶的访问权限默认是私有读写，存储类型默认是标准类型，区域位置为全局域名所在的默认区域。

须知

- 创建桶时，如果使用的终端节点归属于默认区域华北-北京一（cn-north-1），则可以不指定区域；如果使用的终端节点归属于其他区域，则必须指定区域，且指定的区域必须与终端节点归属的区域一致。当前有效的区域名称可从**这里**查询。比如初始化时使用的终端节点EndPoint是obs. ap-southeast-1.myhuaweicloud.com，那么在创建桶的时候必须指定Location： ap-southeast-1才会创建成功，否则会返回状态码400。

带参数创建

创建桶时可以指定桶的访问权限、存储类型和区域位置。OBS支持的桶的存储类型有三类，参见**桶存储类型**。示例代码如下：

```
obs_status ret_status = OBS_STATUS_BUTT;

// 创建并初始化option
obs_options option;
init_obs_options(&option);
option.bucket_options.host_name = "<your-endpoint>";
option.bucket_options.bucket_name = "<Your bucketname>";

// 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
// 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

// 设置响应回调函数
obs_response_handler response_handler =
{
    0, &response_complete_callback
};
// 设置桶的存储类型
option.bucket_options.storage_class = "<Your bucket storage policy>";

// 创建桶，入参可设置桶预定义访问策略、桶区域位置
create_bucket(&option, "<bucket ACL>", "<Your bucket location>", &response_handler, &ret_status);
if (ret_status == OBS_STATUS_OK) {
    printf("create bucket successfully. \n");
}
```

```
else
{
    printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
}
```

6.2 列举桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过list_bucket_obs列举桶。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_list_service_obs_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_list_bucket_obs()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    list_service_data data;
    memset_s(&data, sizeof(list_service_data), 0, sizeof(list_service_data));
    // 自定义响应回调函数
    obs_list_service_obs_handler listHandler =
    {
        {NULL,
        &list_bucket_complete_callback },
        &listServiceObsCallback
    };
    // 列举桶
    list_bucket_obs(&option,&listHandler,&data);
    if (data.ret_status == OBS_STATUS_OK)
    {
        printf("list bucket successfully. \n");
    }
    else
    {
```

```
        printf("list bucket failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

获取到的桶列表将按照桶名字典顺序排列。

6.3 删除桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过delete_bucket删除桶。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_delete_bucket(char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        NULL,
        &response_complete_callback
    };

    delete_bucket(&option, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("delete bucket successfully. \n");
    }
```

```
    else
    {
        printf("delete bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

- 如果桶不为空（包含对象或分段上传碎片），则该桶无法删除。
- 删除桶非幂等操作，删除不存在的桶会报错。

6.4 判断桶是否存在

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过obs_head_bucket接口判断该桶是否已存在。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_head_bucket(char *bucket_name)
{
    // 创建并初始化option
    obs_status ret_status = OBS_STATUS_BUTT;
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 判断桶是否存在
    obs_head_bucket(&option, &response_handler, &ret_status);
```

```
    if (ret_status == OBS_STATUS_OK)
    {
        printf("head bucket successfully. \n");
    }
    else
    {
        printf("head bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

6.5 管理桶访问权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。

桶访问权限（**ACL**）可以通过三种方式设置：

1. 创建桶时指定预定义访问策略。
2. 调用set_bucket_acl_by_head指定预定义访问策略。
3. 调用set_bucket_acl直接设置。

OBS支持的桶或对象权限包含五类，见下表：

权限	描述	OBS C SDK对应值
读权限	如果有桶的读权限，则可以获取该桶内对象列表和桶的元数据。 如果有对象的读权限，则可以获取该对象内容和元数据。	obs_permission.OBS_PERMISSION_READ
写权限	如果有桶的写权限，则可以上传、覆盖和删除该桶内任何对象。 此权限在对象上不适用。	obs_permission.OBS_PERMISSION_WRITE
读ACP权限	如果有读ACP的权限，则可以获取对应的桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有读对应桶或对象ACP的权限。	obs_permission.OBS_PERMISSION_READ_ACP

权限	描述	OBS C SDK对应值
写ACP权限	如果有写ACP的权限，则可以更新对应桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有写对应桶或对象的ACP的权限。 拥有了写ACP的权限，由于可以更改权限控制策略，实际上意味着拥有了完全访问的权限。	obs_permission.OBS_PERMISSION_WRITE_ACP
完全控制权限	如果有桶的完全控制权限意味着拥有READ、WRITE、READ_ACP和WRITE_ACP的权限。 如果有对象的完全控制权限意味着拥有READ、READ_ACP和WRITE_ACP的权限。	obs_permission.OBS_PERMISSION_FULL_CONTROL

OBS预定义的访问策略包含五类，见下表：

表 6-1 预定义访问策略

权限	描述	OBS C SDK对应值
私有读写	桶或对象的所有者拥有完全控制的权限，其他任何人都没有访问权限。	obs_canned_acl.OBS_CANNED_ACL_PRIVATE
公共读私有写	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据、桶的多版本。 设在对象上，所有人可以获取该对象内容和元数据。	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ
公共读写	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据、上传对象删除对象、初始化段任务、上传段、合并段、拷贝段、取消多段上传任务。 设在对象上，所有人可以获取该对象内容和元数据。	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_WRITE

权限	描述	OBS C SDK对应值
桶公共读，桶内对象公共读	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据，可以获取该桶内对象的内容和元数据。 不能应用于对象。	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_DELIVERED
桶公共读写，桶内对象公共读写	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据、上传对象、删除对象、初始化段任务、上传段、合并段、拷贝段、取消多段上传任务，可以获取该桶内对象的内容和元数据。 不能应用于对象。	obs_canned_acl.OBS_CANNED_ACL_PUBLIC_READ_WRITE_DELIVERED

创桶时指定预定义访问策略

以下代码展示如何在创建桶时指定预定义访问策略：

```
static void test_create_bucket(obs_canned_acl canned_acl, char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建桶 canned_acl指定预定义访问策略
    create_bucket(&option, canned_acl, NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("create bucket successfully. \n");
    }
    else
    {
        printf("create bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

为桶设置预定义访问策略

以下代码展示如何为桶设置预定义访问策略参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
canned_acl	obs_canned_acl	必选	请参考**5.5-管理桶访问权限，表1 预定义访问策略**。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
void test_set_bucket_acl_byhead(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 设置桶预定义访问策略
    obs_canned_acl canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    set_bucket_acl_by_head(&option, canned_acl, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket acl by head successfully. \n");
    }
    else
    {
        printf("set bucket acl by head failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

直接设置桶访问权限

以下代码展示如何直接设置桶访问权限参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
aclinfo	manager_acl_info *	必选	管理ACL权限信息相关结构体。

字段名	类型	约束	说明
aclinfo->object_info	obs_object_info *	忽略	对象名和版本号，非多版本对象，version设置为0。
aclinfo->owner_id	char *	忽略	用户的DomainId。
aclinfo->acl_grant_count_return	int *	必选	指向返回aclinfo->acl_grants的个数的指针
aclinfo->acl_grants	obs_acl_grant *	必选	权限信息结构体指针，请查看下表，**表2 权限信息结构体obs_acl_grant描述**。
aclinfo->object_delivered	obs_object_delivered	可选	对象ACL是否继承桶的ACL，有效值OBJECT_DELIVERED_TRUE和OBJECT_DELIVERED_FALSE。默认OBJECT_DELIVERED_TRUE。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

表 6-2 权限信息结构体 obs_acl_grant 描述

字段名	类型	说明
grantee_type	obs_grantee_type	参看下表，**表3 授权者类型描述**。
grantee.canonical_user.id	char	CanonicalUser ID。
permission	obs_permission	参看5.5管理桶访问权限。
bucket_delivered	obs_bucket_delivered	桶的ACL是否向桶内对象传递，有效值BUCKET_DELIVERED_TRUE和BUCKET_DELIVERED_FALSE。默认BUCKET_DELIVERED_FALSE。

表 6-3 授权者类型描述

字段名	说明
obs_grantee_type.OBS_GRANTEE_TYPE_CANONICAL_USER	OBS用户，桶或对象的权限可以授予任何拥有OBS账户的用户，被授权后对应的 OBS 用户可以使用AK和SK访问OBS。

字段名	说明
obs_grantee_type.OBS_GRANTEE_TYPE_ALL_USERS	所有人，所有人都可以访问对应的桶或对象，包括匿名用户。

示例代码如下：

```
void test_set_bucket_acl(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建并初始化acl信息
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    // 设置桶的访问权限
    set_bucket_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket acl successfully. \n");
    }
    else
    {
        printf("set bucket acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 释放内存
    deinitialize_acl_info(&aclinfo);
}
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账号ID，可通过OBS控制台“我的凭证”页面查看。

获取桶访问权限

您可以通过get_bucket_acl获取桶的访问权限。以下代码展示如何获取桶访问权限：

```
void test_get_bucket_acl(char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存
```

```
放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0,&response_complete_callback
    };
    // 申请acl结构内存
    manager_acl_info *aclinfo = malloc_acl_info();
    // 调用获取权限接口
    get_bucket_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get bucket acl: -------------");
        printf("%s\n", aclinfo->owner_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
        printf("get bucket acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 释放内存
    free_acl_info(&aclinfo);
}
```

6.6 获取桶存量信息

须知

开发过程中，您有任何问题可以在github上**提交issue**。

桶存量信息包括桶已使用的空间大小以及桶包含的对象个数。您可以通过get_bucket_storage_info获取桶的存量信息。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
capacity_length	int	必选	使用容量的缓存大小。
capacity	char *	必选	使用容量。
object_number_length	int	必选	对象数目的缓存大小。
object_number	char *	必选	对象数目的缓存。
handler	obs_response_handler *	必选	回调函数。

字段名	类型	约束	说明
callback_data	void *	可选	回调数据。

示例代码

```
static void test_get_bucket_storage_info(char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    obs_status ret_status = OBS_STATUS_BUTT;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    //定义桶容量缓存及对象数缓存
    char capacity[OBS_COMMON_LEN_256] = {0};
    char obj_num[OBS_COMMON_LEN_256] = {0};

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        NULL,
        &response_complete_callback
    };

    // 获取桶存量信息
    get_bucket_storage_info(&option, OBS_COMMON_LEN_256, capacity, OBS_COMMON_LEN_256, obj_num,
        &response_handler, &ret_status);

    if (ret_status == OBS_STATUS_OK) {
         printf("get_bucket_storage_info success,bucket=%s objNum=%s capacity=%s\n",
            bucket_name, obj_num, capacity);
    }
    else
    {
       printf("head bucket failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

6.7 桶配额

须知

开发过程中，您有任何问题可以在github上**提交issue**。

设置桶配额

您可以通过函数set_bucket_quota设置桶配额，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
storage_quota	uint64_t	必选	配额大小，单位字节。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_set_bucket_quota( char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    uint64_t bucketquota = 104857600;

    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 设置桶配额
    set_bucket_quota(&option, bucketquota, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("set bucket quota successfully. \n");
    }
    else
    {
        printf("set bucket quota failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

桶配额值必须为非负整数，单位为字节，支持的最大值为2^{63} - 1。

获取桶配额

您可以通过函数get_bucket_quot获取桶配额，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。

字段名	类型	约束	说明
storagequota_return	uint64_t *	必选	获取到的配额大小，单位字节。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_get_bucket_quota( char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 获取桶配额
    uint64_t bucketquota = 0;
    get_bucket_quota(&option, &bucketquota, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("Bucket=%s  Quota=%lu \n get bucket quota successfully. \n ",
            bucket_name, bucketquota);
    }
    else
    {
        printf("get bucket quota failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

6.8 桶存储类型

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS允许您对桶配置不同的存储类型，桶中对象的存储类型默认将与桶的存储类型保持一致。不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。桶的存储类型分为三类，见下表：

类型	说明	OBS C SDK对应值
标准存储	标准存储拥有低访问时延和较高的吞吐量，适用于有大量热点对象（平均一个月多次）或小对象（<1MB），且需要频繁访问数据的业务场景。	OBS_STORAGE_CLASS_STANDARD
低频访问存储	低频访问存储适用于不频繁访问（平均一年少于12次）但在需要时也要求能够快速访问数据的业务场景。	OBS_STORAGE_CLASS_STANDARD_IA
归档存储	归档存储适用于很少访问（平均一年访问一次）数据的业务场景。	OBS_STORAGE_CLASS_GLACIER

更多关于桶存储类型的内容请参考**桶的存储类别**。

设置桶存储类型

您可以通过set_bucket_storage_class_policy设置桶存储类型，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
storage_class_policy	obs_storage_class	必选	桶存储类型。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_set_bucket_storage_class(char *bucket_name,
                obs_storage_class storage_class_policy)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```

```
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };

    set_bucket_storage_class_policy(&option, storage_class_policy,
            &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket storage class successfully. \n");
    }
    else
    {
        printf("set bucket storage class failed(%s).\n",
                        obs_get_status_name(ret_status));
    }
}
```

获取桶存储类型

您可以通过get_bucket_storage_class_policy获取桶存储类型，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_get_bucket_storage_class_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_get_bucket_storage_class(char *bucket_name)
{
    // 创建并初始化option
    obs_status ret_status = OBS_STATUS_BUTT;
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_get_bucket_storage_class_handler getBucketStorageResponse =
    {
        {0, &response_complete_callback},
        &get_bucket_storageclass_handler
    };
    //获取桶存储类型
    get_bucket_storage_class_policy(&option, &getBucketStorageResponse, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get bucket storage class successfully.\n");
    }
```

```
    else
    {
        printf("get bucket storage class failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


7 上传对象

7.1 流式上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。

流式上传需要将您的数据流传入回调数据callback_data中，然后使用结构体obs_put_object_handler中定义的回调函数put_object_data_callback来处理传入的回调数据。通过调用put_object实现流式上传。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
content_length	uint64_t	必选	对象内容长度。
put_properties	obs_put_properties*	必选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	上传对象加密设置。
handler	obs_put_object_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_put_object_from_buffer()
{
```

```
    // 待上传buffer
    char *buffer = "abcdefg";
    // 待上传buffer的长度
    int buffer_size = strlen(buffer);
    // 上传的对象名
    char *key = "put_buffer_test";

    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    //初始化存储上传数据的结构体
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // 把buffer赋值到上传数据结构中
    data.put_buffer = buffer;
    // 设置buffersize
    data.buffer_size = buffer_size;

    // 设置回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };

    put_object(&option, key, buffer_size, &put_properties,0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from buffer successfully. \n");
    }
    else
    {
        printf("put object from buffer failed(%s).\n",
                        obs_get_status_name(data.ret_status));
    }
}
```

说明

- 上传的内容大小不能超过5GB。
- put_object接口中第五个参数是提供服务端加密功能的参数，具体用法参考服务端加密章节。

7.2 文件上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。

文件上传使用本地文件作为对象的数据源，参数同**流式上传**。以下代码展示了如何进行文件上传：

```
static void test_put_object_from_file()
{
    // 上传对象名
    char *key = "put_file_test";
    // 上传的文件
    char file_name[256] = "./put_file_test.txt";
    uint64_t content_length = 0;
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    // 初始化存储上传数据的结构体
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 打开文件，并获取文件长度
    content_length = open_file_and_get_length(file_name, &data);
    // 设置回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };

    put_object(&option, key, content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

说明

- 上传的内容大小不能超过5GB。
- 上传文件流时，必须以二进制模式（“rb”模式或者“rb+”模式）打开文件。

7.3 创建文件夹

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。创建文件夹实际上是创建了一个大小为0且对象名以“/”结尾的对象，这类对象与其他对象无任何差异，可以进行下载、删除等操作，只是OBS控制台会将这类以“/”结尾的对象以文件夹的方式展示。

```
static void test_creat_folder()
{
    char *key = "YourFolder/";
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    option.bucket_options.uri_style = OBS_URI_STYLE_PATH;
    temp_auth_configure tempauth;
    tempAuthResult  ptrResult;
    memset(&ptrResult,0,sizeof(tempAuthResult));
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 初始化存储上传数据的结构体
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 设置回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };

    put_object(&option, key, 0, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status)
    {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
            obs_get_status_name(data.ret_status));
    }
}
```

说明

- 创建文件夹本质上来说是创建了一个大小为0且对象名以“/”结尾的对象。
- 多级文件夹创建最后一级即可，比如src1/src2/src3/，创建src1/src2/src3/即可，无需创建src1/、src1/src2/。

7.4 设置对象属性

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以在上传对象时设置对象属性。对象属性包含对象长度、对象MIME类型、对象MD5值（用于校验）、对象存储类型、对象自定义元数据。对象属性可以在多种上传方式下（流式上传、文件上传、分段上传、基于表单上传），或**复制对象**时进行设置。对象属性参数在结构体obs_put_properties中。

对象属性详细说明见下表：

名称	描述	默认值
对象长度（content_length）	上传对象的长度，超过流/文件的长度会截断。	流/文件实际长度
对象MIME类型（content_type）	对象的MIME类型，定义对象的类型及网页编码，决定浏览器将以什么形式、什么编码读取对象。	无
对象MD5值（md5）	对象数据的MD5值（经过Base64编码），提供给OBS服务端，校验数据完整性。	无
对象存储类型	对象的存储类型，不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。默认与桶的存储类型保持一致，可以设置为与桶的存储类型不同。	无
对象自定义元数据	用户对上传到桶中对象的自定义属性描述，以便对对象进行自定义管理。	无

设置对象 MIME 类型

您可以通过在结构体obs_put_properties中对元素content_type赋值来设置对象MIME类型。以文件上传为例展示如何设置对象MIME类型：

```
static void test_put_object_from_file2()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 初始化结构体put_properties，可以通过该结构体设置对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 设置MIME类型
    put_properties.content_type = "text/html";
    // 回调数据
```

```
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 将要上传的文件读到回调数据中
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // 回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
     };
    // 上传数据流
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

说明

如果不设置对象MIME类型，SDK会根据上传对象的后缀名自动判断对象MIME类型，如.xml判断为application/xml文件；.html判断为text/html文件。

设置对象存储类型

您可以通过obs_bucket_context中对元素storage_class赋值来设置对象存储类型。以下代码展示如何设置对象存储类型：

```
static void test_put_object_from_file3()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    //设置存储类型为归档存储
    option.bucket_options.storage_class = OBS_STORAGE_CLASS_GLACIER;
    // 初始化结构体put_properties，可以通过该结构体设置对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 回调数据
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 将要上传的文件读到回调数据中
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // 回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
    };
    // 上传数据流
```

```
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
            obs_get_status_name(data.ret_status));
    }

}
```

说明

- 如果不设置，对象的存储类型默认与桶的存储类型保持一致。
- 对象的存储类型分为三类，其含义与**桶存储类型**一致。
- 下载归档存储类型的对象前必须将其恢复。

设置对象自定义元数据

您可以通过obs_put_properties中对元素meta_data赋值来设置对象自定义元数据。以下代码展示如何设置对象自定义元数据：

```
static void test_put_object_from_file4()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 初始化结构体put_properties，可以通过该结构体设置对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //设置自定义元数据
    obs_name_value matadata;
    matadata.name ="property1";
    matadata.value ="property-value1";
    put_properties.meta_data = matadata;
    // 回调数据
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 将要上传的文件读到回调数据中
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // 回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_object_data_callback
     };
    // 上传数据流
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
```



```
            obs_get_status_name(data.ret_status));
    }
}
```

说明

- 在上面设置对象自定义元数据示例代码中，用户自定义了一个名称为“property1”，值为“property-value1”的元数据
- 一个对象可以有多个元数据，总大小不能超过8KB。
- 对象的自定义元数据可以通过get_object_metadata获取，参见**获取对象属性**。
- 使用get_object下载对象时，对象的自定义元数据也会同时下载。

7.5 分段上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。

对于较大文件上传，可以切分成段上传。用户可以在如下的应用场景内（但不仅限于此），使用分段上传的模式：

- 上传超过100MB大小的文件。
- 网络条件较差，和OBS服务端之间的链接经常断开。
- 上传前无法确定将要上传文件的大小。

分段上传分为如下3个步骤：

步骤1 初始化分段上传任务（initiate_multi_part_upload）。

步骤2 逐个或并行上传段（upload_part）。

步骤3 合并段（complete_multi_part_upload）或取消分段上传任务(abort_multi_part_upload)。

----结束

初始化分段上传任务

使用分段上传方式传输数据前，必须先通知OBS初始化一个分段上传任务。该操作会返回一个OBS服务端创建的全局唯一标识（upload_id），用于标识本次分段上传任务。您可以根据这个唯一标识来发起相关的操作，如取消分段上传任务、列举分段上传任务、列举已上传的段等。

请注意，多段上传的对象的相关属性（如acl、过期时间等等）是在初始化分段上传任务时设置，不能在上传段或合并段时设置，请注意设置相关属性的时机。

您可以通过initiate_multi_part_upload初始化一个分段上传任务，参数描述如下表所示：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。

字段名	类型	约束	说明
key	char *	必选	对象名。
upload_id_return_size	int	必选	多段上传id缓存大小。
upload_id_return	char *	必选	多段上传id缓存。
put_properties	obs_put_properties*	可选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	服务端加密设置。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

```
static void test_initiate_multi_part_upload()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 定义多段上传id缓存及大小
    char upload_id[OBS_COMMON_LEN_256] = {0};
    int upload_id_size = OBS_COMMON_LEN_256;
    // 设置响应回调函数
    obs_response_handler handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 初始化分段上传任务,这里的upload_id就是接口定义中的upload_id_return
    initiate_multi_part_upload(&option, "<object key>", upload_id_size, upload_id, NULL, NULL, &handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("test init upload part successfully. uploadId= %s\n", upload_id);
    }
    else
    {
        printf("test init upload part faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

- 在结构体obs_put_properties中，您可以设置对象MIME类型、对象自定义元数据。
- initiate_multi_part_upload返回分段上传任务的全局唯一标识（upload_id），在后面的操作中将用到它。

上传段

初始化一个分段上传任务之后，可以根据指定的对象名和uploadId来分段上传数据。每一个上传的段都有一个标识它的号码——分段号（Part Number，范围是1~10000）。对于同一个uploadId，该分段号不但唯一标识这一段数据，也标识了这段数据在整个对象内的相对位置。如果您用同一个分段号上传了新的数据，那么OBS上已有的这个段号的数据将被覆盖。除了最后一段以外，其他段的大小范围是100KB~5GB；最后段大小范围是0~5GB。每个段不需要按顺序上传，甚至可以在不同进程、不同机器上上传，OBS会按照分段号排序组成最终对象。

您可以通过upload_part上传段，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
upload_part_info	obs_upload_part_info *	必选	上传段的信息。
upload_part_info->part_number	unsigned int	必选	上传段的段号。取值为从1到10000的整数。
upload_part_info->upload_id	char *	必选	多段上传任务Id。
content_length	uint64_t	必选	上传内容长度。
put_properties	obs_put_properties*	可选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	服务端加密设置。
handler	obs_upload_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_upload_part()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 定义分片大小5M
    uint64_t uploadSliceSize =5L * 1024 * 1024;
```

```
    // 定义并初始化上传段大小
    uint64_t uploadSize = uploadSliceSize;
    // 定义并初始化上传文件长度变量
    uint64_t filesize = 0;
    //初始化put_properties
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //回调函数
    obs_upload_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &test_upload_file_data_callback
    };
    //回调数据初始化
    test_upload_file_callback_data data;
    memset(&data, 0, sizeof(test_upload_file_callback_data));
    filesize = get_file_info(filename,&data);
    data.noStatus = 1;
    data.part_size = uploadSize;
    data.part_num = (filesize % uploadSize == 0) ? (filesize / uploadSize) : (filesize / uploadSize +1);

    //上传第一段
    uploadPartInfo.part_number=1;
    uploadPartInfo.upload_id = "<upload id>";
    data.start_byte  = 0;
    upload_part(&option,key,&uploadPartInfo,uploadSize,
                                &put_properties,0,&Handler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 1 successfully. \n");
    }
    else
    {
        printf("test upload part 1 faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    //上传第二段
    uploadPartInfo.part_number=2;
    uploadPartInfo.upload_id = "<upload id>";
    filesize = get_file_info(filename,&data);
    uploadSize =filesize - uploadSize;
    data.part_size = uploadSize;
    data.start_byte = uploadSliceSize;
    fseek(data.infile, data.start_byte, SEEK_SET);
    upload_part(&option,key,&uploadPartInfo,uploadSize, &put_properties,0,&Handler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("test upload part 2 successfully. \n");
    }
    else
    {
        printf("test upload part 2 faied(%s).\n", obs_get_status_name(data.ret_status));
    }

}
```

说明

- 上传段接口要求除最后一段以外，其他的段大小都要大于100KB。但是上传段接口并不会立即校验上传段的大小（因为不知道是否为最后一块）；只有调用合并段接口时才会校验。
- OBS会将服务端收到段数据的ETag值（段数据的MD5值）返回给用户。
- 为了保证数据在网络传输过程中不出现错误，可以通过设置MD5值，并放到Content-MD5请求头中；OBS服务端会计算上传数据的MD5值与SDK计算的MD5值比较，保证数据完整性。
- 可以通过put_properties.md5直接设置上传数据的MD5值，提供给OBS服务端用于校验数据完整性。
- 分段号的范围是1~10000。如果超出这个范围，OBS将返回400 Bad Request错误。
- OBS 3.0的桶支持最小段的大小为100KB，OBS 2.0的桶支持最小段的大小为5MB。请在OBS 3.0的桶上执行分段上传操作。

合并段

所有分段上传完成后，需要调用合并段接口来在OBS服务端生成最终对象。在执行该操作时，需要提供所有有效的分段列表（包括分段号和分段ETag值）；OBS收到提交的分段列表后，会逐一验证每个段的有效性。当所有段验证通过后，OBS将把这些分段组合成最终的对象。

您可以通过complete_multi_part_upload合并段，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
upload_id	char *	必选	指明多段上传任务。
part_number	unsigned int	必选	段个数，complete_upload_Info数组长度
complete_upload_Info	obs_complete_upload_Info *	必选	段信息数组。
complete_upload_Info->part_number	unsigned int	必选	段号。
complete_upload_Info->etag	char *	必选	对应段的ETag值。
put_properties	obs_put_properties*	可选	上传对象属性。
handler	obs_complete_multi_part_upload_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_complete_upload(char *filename, char *key)
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //从环境变量读取ak/sk
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 初始化结构体put_properties
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
```

```
    // 设置分段信息
    char *uploadId = "<upload id>";
    obs_complete_upload_Info info[2];
    info[0].part_number="1";
    info[0].etag="65fe0e161b35c8deead213871033f7fa";
    info[1].part_number="2";
    info[1].etag="0433d5ffc28450be3b6cf25ab8955267";
    // 设置响应回调函数
    obs_complete_multi_part_upload_handler Handler =
    {
        {&response_properties_callback,
         &response_complete_callback},
        &CompleteMultipartUploadCallback
    };
    // 合并段
    complete_multi_part_upload(&option,key,uploadId,number,info,&putProperties,
              &Handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test complete upload successfully. \n");
    }
    else
    {
        printf("test complete upload faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

- 上面代码中的info结构体数组是进行上传段后保存的分段号和分段ETag值的列表。
- 分段可以是不连续的。

并发分段上传

分段上传的主要目的是解决大文件上传或网络条件较差的情况。下面的示例代码展示了如何使用分段上传并发上传大文件：

```
static void test_concurrent_upload_part(char *filename, char *key, uint64_t uploadSliceSize)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //从环境变量读取ak/sk
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    char *concurrent_upload_id;
    uint64_t uploadSize = uploadSliceSize;
    uint64_t filesize =0;
    //初始化结构体put_properties
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //大文件信息:文件指针，文件大小，按照分段大小的分段数
    test_upload_file_callback_data data;
    memset(&data, 0, sizeof(test_upload_file_callback_data));
    filesize = get_file_info(filename,&data);
    data.noStatus = 1;
    data.part_size = uploadSize;
    data.part_num = (filesize % uploadSize == 0) ? (filesize / uploadSize) : (filesize / uploadSize +1);
    // 初始化上传段回调函数
    obs_response_handler Handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // 合并段回调函数
    obs_complete_multi_part_upload_handler complete_multi_handler =
```

```
    {
        {&response_properties_callback,
         &response_complete_callback},
        &CompleteMultipartUploadCallback
    };
    //初始化上传段任务返回uploadId: uploadIdReturn
    char uploadIdReturn[256] = {0};
    int upload_id_return_size = 255;
    initiate_multi_part_upload(&option,key,upload_id_return_size,uploadIdReturn, &putProperties,
            0,&Handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test init upload part return uploadIdReturn(%s). \n", uploadIdReturn);
        strcpy(concurrent_upload_id,uploadIdReturn);
    }
    else
    {
        printf("test init upload part faied(%s).\n", obs_get_status_name(ret_status));
    }
    // 并发上传
    test_concurrent_upload_file_callback_data *concurrent_upload_file;
    concurrent_upload_file =(test_concurrent_upload_file_callback_data *)malloc(
                    sizeof(test_concurrent_upload_file_callback_data)*(data.part_num+1));
    if(concurrent_upload_file == NULL)
    {
         printf("malloc test_concurrent_upload_file_callback_data failed!!!");
         return ;
    }
    test_concurrent_upload_file_callback_data *concurrent_upload_file_complete =
                            concurrent_upload_file;
    start_upload_threads(data, concurrent_upload_id,filesize, key, option, concurrent_upload_file_complete);
     // 合并段
    obs_complete_upload_Info *upload_Info = (obs_complete_upload_Info *)malloc(
                sizeof(obs_complete_upload_Info)*data.part_num);
    for(i=0; i<data.part_num; i++)
    {
        upload_Info[i].part_number = concurrent_upload_file_complete[i].part_num;
        upload_Info[i].etag = concurrent_upload_file_complete[i].etag;
    }
    complete_multi_part_upload(&option, key, uploadIdReturn,
data.part_num,upload_Info,&putProperties,&complete_multi_handler,&ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("test complete upload successfully. \n");
    }
    else
    {
        printf("test complete upload faied(%s).\n", obs_get_status_name(ret_status));
    }
    if(concurrent_upload_file)
    {
        free(concurrent_upload_file);
        concurrent_upload_file = NULL;
    }
    if(upload_Info)
    {
        free(upload_Info);
        upload_Info = NULL;
    }
}
```

7.6 分段复制

须知

开发过程中，您有任何问题可以在github上**提交issue**。

分段复制是分段上传的一种特殊情况，即分段上传任务中的段通过复制OBS指定桶中现有对象（或对象的一部分）来实现。您可以通过copy_part来复制段，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
object_info	obs_copy_destination_object_info *	必选	指明多段上传任务。
object_info->destination_bucket	char *	必选	目标对象所在桶。
object_info->destination_key	char *	必选	目标对象名称。
object_info->last_modified_return	int64_t *	必选	对象上次修改的时间。
object_info->etag_return_size	int	必选	eTag缓存大小。
object_info->etag_return	char *	必选	eTag缓存。
copypart	obs_upload_part_info *	必选	上传段信息。
copypart->part_number	unsigned int	必选	上传段的段号。
copypart->upload_id	char *	必选	多段上传任务Id。
put_properties	obs_put_properties*	可选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	服务端加密设置。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_copy_part()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
```

```
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // SSE-KMS加密
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    // 复制段后返回的etag值
    char etagreturn[256] ={0};

    char *key= "<Source object key>"
    // 定义拷贝段信息
    obs_copy_destination_object_info object_info;
    memset(&object_info, 0, sizeof(obs_copy_destination_object_info));
    object_info.destination_bucket = "<Your destination bucketname>";
    object_info.destination_key = "<Your destination object key>";
    object_info.etag_return = etagreturn;
    object_info.etag_return_size = 256;
    obs_upload_part_info copypart;
    memset(&copypart, 0, sizeof(obs_upload_part_info));
    // 设置响应回调函数
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 拷贝第一段
    copypart.part_number = "1";
    copypart.upload_id = "<upload id>";
    copy_part(&option, key, &object_info, &copypart,
              &putProperties,&encryption_params,&responseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf(" copy part 1 successfully. \n");
    }
    else
    {
        printf("copy part 1 failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 拷贝第二段
    copypart.part_number = "2";
    copypart.upload_id = "<upload id>";
    copy_part(&option, key, &object_info, &copypart,
              &putProperties,&encryption_params,&responseHandler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf(" copy part 2 successfully. \n");
    }
    else
    {
        printf("copy part 2 failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

7.7 断点续传上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。

当上传大文件时，经常出现因网络不稳定或程序崩溃导致上传失败的情况。失败后再次重新上传不仅浪费资源，而且当网络不稳定时仍然有上传失败的风险。断点续传上传接口能有效地解决此类问题引起的上传失败，其原理是将待上传的文件分成若干个分段分别上传，并实时地将每段上传结果统一记录在checkpoint文件中，仅当所有分段都上传成功时返回上传成功的结果，否则抛出异常提醒用户再次调用接口进行重新上传（重新上传时因为有checkpoint文件记录当前的上传进度，避免重新上传所有分段，从而节省资源提高效率）。

您可以通过upload_file进行断点续传上传。该接口可设置的参数如下：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
upload_file_config	obs_upload_file_configuration *	必选	上传文件的配置说明，请参看下表。
encryption_params	server_side_encryption_params *	可选	上传对象加密设置。
handler	obs_upload_file_response_handler *	必选	回调结构体，所有成员都是回调函数的指针。
callback_data	void *	可选	回调数据。

上传文件结构obs_upload_file_configuration描述如下表：

成员名	类型	约束	说明
upload_file	char *	必选	待上传的本地文件。
part_size	uint64_t	必选	分段大小，单位字节，取值范围是100KB~5GB，默认为5MB。

成员名	类型	约束	说明
check_point_file	char *	必选	记录上传进度的文件，只在断点续传模式下有效。当该值为空时，默认与待上传的本地文件同目录。
enable_check_point	int	必选	是否开启断点续传模式，默认为0，表示不开启。
task_num	int	必选	分段上传时的最大并发数，默认为1。

示例代码

以下代码展示了如何使用断点续传上传接口上传文件：

```
void uploadFileResultCallback(obs_status status,
                              char *resultMsg,
                              int partCountReturn,
                              obs_upload_file_part_info * uploadInfoList,
                              void *callbackData);
//回调函数声明
static void test_upload_file()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

     // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    //初始化结构体put_properties
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    obs_upload_file_configuration uploadFileInfo;

memset_s(&uploadFileInfo,sizeof(obs_upload_file_configuration),0,sizeof(obs_upload_file_configuration));
    uploadFileInfo.check_point_file = 0;
    uploadFileInfo.enable_check_point = 1;
    uploadFileInfo.part_size = "<part size>";
    uploadFileInfo.task_num = "<task num>";
    uploadFileInfo.upload_file = "<upload filename>";
    uploadFileInfo.put_properties = &put_properties;

    //回调函数
    obs_upload_file_response_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback_for_multi_task},
        &uploadFileResultCallback
    };
    initialize_break_point_lock();
    upload_file(&option, "<Your Key>", 0, &uploadFileInfo, Null, &Handler, &ret_status);
    deinitialize_break_point_lock();
    if (OBS_STATUS_OK == ret_status) {
```

```
        printf("test upload file successfully. \n");
    }
    else
    {
        printf("test upload file faied(%s).\n", obs_get_status_name(ret_status));
    }
}
//uploadFileResultCallback示例，printf语句可以替换为自定义的日志打印语句
void uploadFileResultCallback(obs_status status,
                              char *resultMsg,
                              int partCountReturn,
                              obs_upload_file_part_info * uploadInfoList,
                              void *callbackData)
{
    int i=0;
    obs_upload_file_part_info * pstUploadInfoList = uploadInfoList;
    printf("status return is %d(%s)\n",status,obs_get_status_name(status));
    printf("%s",resultMsg);
    printf("partCount[%d]\n",partCountReturn);
    for(i=0;i<partCountReturn;i++)
    {
        printf("partNum[%d],startByte[%llu],partSize[%llu],status[%d]\n",
        pstUploadInfoList->part_num,
        pstUploadInfoList->start_byte,
        pstUploadInfoList->part_size,
        pstUploadInfoList->status_return);
        pstUploadInfoList++;
    }
    if (callbackData) {
        obs_status* retStatus = (obs_status*)callbackData;
        (*retStatus) = status;
    }
}
```

说明

- 断点续传上传接口是利用**分段上传**特性实现的，是对分段上传的封装和加强。
- 断点续传上传接口不仅能在失败重传时节省资源提高效率，还因其对分段进行并发上传的机制能加快上传速度，帮助用户快速完成上传业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发上传的实现等内部细节。
- enable_check_point参数：0代表不启用断点续传模式，此时断点续传上传接口退化成对分段上传的简单封装，不会产生checkpoint文件，1代表启用断点续传模式。

7.8 追加写

须知

开发过程中，您有任何问题可以在github上**提交issue**。

追加写接口append_object同put_object参数和使用方法一样，只不过是多了一个写对象的起始位置position。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。

字段名	类型	约束	说明
key	char *	必选	对象名。
content_length	uint64_t	必选	对象内容长度。
position	char *	必选	追加写的起始位置。
put_properties	obs_put_properties*	必选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	上传对象加密设置。
handler	obs_append_object_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_append_object_from_buffer()
{
    // 待上传buffer
    char *buffer = "abcdefg";
    // 待上传buffer的长度
    int buffer_size = strlen(buffer);
    // 上传的对象名
    char *key = "put_buffer_test";
    char * position = "0";
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //初始化存储上传数据的结构体
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // 把buffer赋值到上传数据结构中
    data.put_buffer = buffer;
    // 设置buffersize
    data.buffer_size = buffer_size;
    // 设置回调函数
    obs_append_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    append_object(&option, key, buffer_size, position, &put_properties,
    0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from buffer successfully. \n");
    }
```

```
    else
    {
        printf("put object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

- put_object上传的对象可覆盖append_object上传的对象，覆盖后对象变为普通对象，不可再进行追加上传。
- 第一次调用追加上传时，如果已存在同名的普通对象，则会报错（HTTP状态码为409）。
- 每次追加上传返回的ETag是当次追加数据内容的ETag，不是完整对象的ETag；
- 单次追加上传的内容不能超过5GB，且最多支持10000次追加上传。
- append_object接口中第六个参数是提供服务端加密功能的参数，具体用法参考服务端加密章节。

7.9 修改写

须知

开发过程中，您有任何问题可以在github上**提交issue**。

修改写接口modify_object只适用于并行文件系统的桶，对象桶该接口不支持。该接口同put_object参数和使用方法一样，只不过是多了一个写对象的起始位置position。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
content_length	uint64_t	必选	对象内容长度。
position	char *	必选	修改写的起始位置。
put_properties	obs_put_properties*	必选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	上传对象加密设置。
handler	obs_modify_object_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_modify_object_from_buffer()
{
    // 待上传buffer
```

```
    char *buffer = "abcdefg";
    // 待上传buffer的长度
    int buffer_size = strlen(buffer);
    // 上传的对象名
    char *key = "put_buffer_test";
    char * position = "0";
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //初始化存储上传数据的结构体
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    // 把buffer赋值到上传数据结构中
    data.put_buffer = buffer;
    // 设置buffersize
    data.buffer_size = buffer_size;
    // 设置回调函数
    obs_modify_object_handler modify0bjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    modify_object(&option, key, buffer_size, position, &put_properties,
    0, &modify0bjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("modify object from buffer successfully. \n");
    }
    else
    {
        printf("modify object from buffer failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

- modify_object只适用于并行文件系统的桶。
- 第一次调用modify_object时，如果修改的对象不存在，则会报错（HTTP状态码为404）。
- modify_object接口中第六个参数是提供服务端加密功能的参数，具体用法参考服务端加密章节。

8 下载对象

8.1 对象下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以调用get_object函数下载对象。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。
get_conditions	obs_get_conditions *	必选	对象筛选条件和读取范围设置。
encryption_params	server_side_encryption_params *	可选	获取对象的解密设置。
handler	obs_get_object_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
```

```
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置下载的对象
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    //根据业务需要设置存放下载对象数据的结构
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // 定义范围下载参数
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.start_byte = "<start byte>";
    // 下载长度，默认0，读到对象尾
    getcondition.byte_count = "<byte count>";

    // 定义下载的回调函数
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

说明

- 通过操作对象输入流可将对象的内容读取到本地文件或者内存中。
- getcondition可以传空，表示下载整个对象。

8.2 限定条件下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。

下载对象时，可以指定一个或多个限定条件，满足限定条件时则进行下载，否则抛出异常，下载对象失败。

您可以使用的限定条件如下：

参数	作用
obs_get_conditions.if_modified_since	如果对象在指定的时间后有修改，则返回对象内容，否则返回错误。
obs_get_conditions.if_not_modified_since	如果对象在指定的时间后没有修改，则返回对象内容，否则返回错误。
obs_get_conditions.if_match_etag	如果对象的ETag值与该参数值相同，则返回对象内容，否则抛出异常。
obs_get_conditions.if_not_match_etag	如果对象的ETag值与该参数值不相同，则返回对象内容，否则抛出异常。

说明

- 对象的ETag值是指对象数据的MD5校验值。
- 如果包含if_not_modified_since并且不符合或者包含If-Match并且不符合，抛出异常中HTTP状态码为：412 precondition failed。
- 如果包含if_modified_since并且不符合或者包含If-None-Match并且不符合，抛出异常中HTTP状态码为：304 Not Modified。

以下代码展示了如何进行限定条件下载：

```
static void test_get_object_by_range()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 范围下载参数
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.if_match_etag = "<object etag>";
    getcondition.if_modified_since = "<time object modified>";
    getcondition.if_not_match_etag = "<not matched etag>";
    getcondition.if_not_modified_since = "<time object modified>";
    // 下载对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // 下载后保存在本地文件信息
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
    // 设置响应回调函数
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
```



```
    &get_object_data_callback
};

//下载对象
get_object(&option,&object_info,&getcondition,0,&getobjectHandler,&data);
fclose(data.outfile);
if (OBS_STATUS_OK == data.ret_status) {
    printf("get object successfully. \n");
}
else
{
    printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
}
}
```

8.3 下载归档存储对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

如果要下载归档存储对象，需要先将归档存储对象恢复。恢复归档存储对象的恢复选项可支持二类，见下表：

选项	说明	OBS C SDK对应值
快速恢复	恢复耗时1~5分钟。	OBS_TIER_EXPEDITED
标准恢复	恢复耗时3~5小时。默认值。	OBS_TIER_STANDARD

⚠ 注意

重复恢复归档存储数据时在延长恢复有效期的同时，也将会对恢复时产生的恢复费用进行重复收取。产生的标准存储类别的对象副本有效期将会延长，并且收取延长时间段产生的标准存储副本费用。

您可以通过restore_object恢复归档存储对象。以下代码展示了如何下载归档存储对象：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。
days	char *	必选	恢复对象的保存时间。

字段名	类型	约束	说明
tier	obs_tier	可选	恢复选项，可以为：obs_tier.OBS_TIER_EXPEDITED，obs_tier.OBS_TIER_STANDARD。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

```
static void test_restore_object()
{
    // 定义对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler handler =
    {
      &response_properties_callback, &response_complete_callback
    };
    // 恢复对象
    obs_tier tier = OBS_TIER_EXPEDITED;
    restore_object(&option, &object_info, "<stored time>",tier,&handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("restore object successfully. \n");
    }
    else
    {
        printf("restore object faied(%s).\n", obs_get_status_name(ret_status));
        return;
    }
    // 下载对象回调数据
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
    // 设置响应回调函数
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };
    // 下载对象
    get_object(&option,&object_info, 0,0,&getobjectHandler,&data);
```

```
    fclose(data.outfile);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

- object_info.key中指定的对象必须是归档存储类型，否则调用该接口会报错。
- "<恢复对象的保存时间>"指定恢复对象保存的时间，取值范围是1~30。

8.4 断点续传下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。

当下载大对象到本地文件时，经常出现因网络不稳定或程序崩溃导致下载失败的情况。失败后再次重新下载不仅浪费资源，而且当网络不稳定时仍然有下载失败的风险。断点续传下载接口能有效地解决此类问题引起的下载失败，其原理是将待下载的对象分成若干个分段分别下载，并实时地将每段下载结果统一记录在checkpoint文件中，仅当所有分段都下载成功时返回下载成功的结果，否则抛出异常提醒用户再次调用接口进行重新下载（重新下载时因为有checkpoint文件记录当前的下载进度，避免重新下载所有分段，从而节省资源提高效率）。

您可以通过download_file进行断点续传下载。该接口可设置的参数描述如下：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
version_id	char *	必选	对象的版本号。
download_file_config	obs_download_file_configuration *	必选	下载文件的配置说明，请参考下表obs_download_file_configuration成员说明。
get_conditions	obs_get_conditions *	必选	对象筛选条件和读取范围设置，请参考**7.2限定条件**说明。
encryption_params	server_side_encryption_params *	可选	上传对象加密设置。
handler	obs_download_file_response_handler *	必选	回调函数。

字段名	类型	约束	说明
callback_data	void *	可选	回调数据。

下载文件结构obs_download_file_configuration描述如下表：

字段名	类型	约束	说明
downLoad_file	char *	必选	下载对象的本地文件路径。当该值为空时，默认为当前程序的运行目录。
part_size	uint64_t	必选	分段大小，单位字节，取值范围是5MB~5GB，默认为5MB。
task_num	int	必选	分段下载时的最大并发数，默认为1。
enable_check_point	int	必选	是否开启断点续传模式，默认为0，表示不开启。
check_point_file	char *	必选	记录下载进度的文件，只在断点续传模式下有效。当该值为空时，默认与下载对象的本地文件路径同目录。

以下代码展示了如何使用断点续传下载接口下载对象到本地文件：

```
void downloadFileResultCallback(obs_status status,
                                char *resultMsg,
                                int partCountReturn,
                                obs_download_file_part_info * downloadInfoList,
                                void *callbackData);
//回调函数声明
static void test_download_file(char *filename, char *key)
{
    obs_status ret_status = OBS_STATUS_BUTT;
    uint64_t uploadSliceSize =5L * 1024 * 1024;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 初始化getConditions
    obs_get_conditions getConditions;
    memset_s(&getConditions,sizeof(obs_get_conditions),0,sizeof(obs_get_conditions));
    init_get_properties(&getConditions);
    // 断点下载对象信息
    obs_download_file_configuration downloadFileConfig;
```



```
    memset_s(&downloadFileConfig,sizeof(obs_download_file_configuration),0,
                sizeof(obs_download_file_configuration));
    downloadFileConfig.check_point_file = NULL;
    downloadFileConfig.enable_check_point = 1;
    downloadFileConfig.part_size = uploadSliceSize;
    downloadFileConfig.task_num = 10;
    downloadFileConfig.downLoad_file= filename;
    // 设置响应回调函数
    obs_download_file_response_handler Handler =
    {
        {&response_properties_callback, &response_complete_callback_for_multi_task},
        &downloadFileResultCallback
    };
    initialize_break_point_lock();
    download_file(&option, key, 0,&getConditions,0,&downloadFileConfig,
          &Handler, &ret_status);
    deinitialize_break_point_lock();
    if (OBS_STATUS_OK == ret_status) {
        printf("test download file successfully. \n");
    }
    else
    {
        printf("test download file faied(%s).\n", obs_get_status_name(ret_status));
    }
}
//downloadFileResultCallback 示例，printf语句可以替换为自定义的日志打印语句
void downloadFileResultCallback(obs_status status,
                                char *resultMsg,
                                int partCountReturn,
                                obs_download_file_part_info * downloadInfoList,
                                void *callbackData)
{
    int i=0;
    obs_download_file_part_info * pstDownloadInfoList = downloadInfoList;
    printf("status return is %d(%s)\n", status, obs_get_status_name(status));
    printf("%s",resultMsg);
    printf("partCount[%d]\n",partCountReturn);
    for(i=0;i<partCountReturn;i++)
    {
        printf("partNum[%d],startByte[%llu],partSize[%llu],status[%d]\n",
        pstDownloadInfoList->part_num,
        pstDownloadInfoList->start_byte,
        pstDownloadInfoList->part_size,
        pstDownloadInfoList->status_return);
        pstDownloadInfoList++;
    }
    if (callbackData) {
        obs_status* retStatus = (obs_status*)callbackData;
        (*retStatus) = status;
    }
}
```

说明

- 断点续传下载接口是利用**分段上传**特性实现的，是对范围下载的封装和加强。
- 断点续传下载接口不仅能在失败重下时节省资源提高效率，还因其对分段进行并发下载的机制能加快下载速度，帮助用户快速完成下载业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发下载的实现等内部细节。
- enable_check_point参数默认是0，代表不启用断点续传模式，此时断点续传下载接口退化成对范围下载的简单封装，不会产生checkpoint文件。
- check_point_file参数仅在enable_check_point参数为1时有效。

8.5 图片处理

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS为用户提供了稳定、安全、高效、易用、低成本的图片处理服务。当要下载的对象是图片文件时，您可以通过传入图片处理参数对图片文件进行图片剪切、图片缩放、图片水印、格式转换等处理。

更多关于图片处理的内容，参见**图片处理特性指南**。

以下代码展示了如何使用下载对象接口实现图片处理：

```
static void test_get_object_image()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 图片转码配置
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.image_process_config.image_process_mode = obs_image_process_cmd;
    getcondition.image_process_config.cmds_stylename = "resize,m_fixed,w_100,h_100/rotate,90";
    //下载对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // 下载对象回调数据
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file("<file path>");
    // 设置响应回调函数
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback, &get_object_complete_callback},
        &get_object_data_callback
    };

    // 下载对象
    get_object(&option,&object_info,&getcondition,0,&getobjectHandler,&data);
    fclose(data.outfile);
     if (OBS_STATUS_OK == data.ret_status) {
            printf("get object successfully. \n");
        }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

- 使用getcondition.image_process_config指定图片处理参数。
- 图片处理参数当前仅支持命令方式，即image/commands格式。
- 图片处理参数支持级联处理，可对图片文件依次实施多条命令。

9 管理对象

9.1 获取对象属性

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_object_metadata来获取对象属性，包括对象长度，对象MIME类型，对象自定义元数据等信息。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。
encryption_params	server_side_encryption_params *	可选	服务端加密配置参数。
handler	obs_get_object_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

以下代码展示了如何获取对象属性：

```
static void test_get_object_metadata()
{
    obs_status ret_status = OBS_STATUS_BUTT;
```

```
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 对象信息
    obs_object_info object_info;
    memset(&object_info,0,sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // 获取对象属性
    get_object_metadata(&option,&object_info,0, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get object metadata successfully. \n");
    }
    else
    {
        printf("get object metadata failed.\n", obs_get_status_name(ret_status));
    }
}
```

9.2 管理对象访问权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。

对象访问权限与桶访问权限类似，也可支持预定义访问策略（参见**管理桶访问权限**）或直接设置。

对象访问权限（**ACL**）可以通过三种方式设置：

1. 上传对象时指定预定义访问策略。
2. 调用set_object_acl_by_head指定预定义访问策略。
3. 调用set_object_acl直接设置。

上传对象时指定预定义访问策略

以下代码展示如何在上传对象时指定预定义访问策略：

```
static void test_put_object_acl()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
```

```
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 初始化结构体put_properties，可以通过该结构体设置对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 指定预定义访问策略
    put_properties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    // 回调数据
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 将要上传的文件读到回调数据中
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // 回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_buffer_object_data_callback
     };
    // 上传数据流
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

为对象设置预定义访问策略

您可以通过set_object_acl_by_head来设置对象属性，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。
canned_acl	obs_canned_acl	必选	请参考**5.5-管理桶访问权限，表1 预定义访问策略**。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
void test_set_object_acl_byhead()
{
```

```
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    obs_canned_acl canned_acl = OBS_CANNED_ACL_PUBLIC_READ_WRITE;
    obs_object_info object_info;
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    // 设置对象预定义访问策略
    set_object_acl_by_head(&option, &object_info, canned_acl, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket acl by head successfully. \n");
    }
    else
    {
        printf("set bucket acl by head failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

直接设置对象访问权限

您可以通过set_object_acl来设置对象属性，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
aclinfo	manager_acl_info *	必选	管理ACL权限信息相关结构体。
aclinfo->object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。
aclinfo->owner_id	char *	可选	用户的DomainId。
aclinfo->acl_grant_count_return	int *	必选	指向返回aclinfo->acl_grants的个数的指针
aclinfo->acl_grants	obs_acl_grant *	必选	权限信息结构体指针，请查看**5.5-管理桶访问权限**，**表2 权限信息结构体obs_acl_grant描述**。

字段名	类型	约束	说明
aclinfo->object_delivered	obs_object_delivered	可选	对象ACL是否继承桶的ACL，有效值OBJECT_DELIVERED_TRUE和OBJECT_DELIVERED_FALSE。默认OBJECT_DELIVERED_TRUE。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
void test_set_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 定义对象访问权限信息
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    aclinfo.object_info.key = "<object key>";
    aclinfo.object_info.version_id = "<object version ID>";
    // 设置对象访问权限
    set_object_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set object acl successfully. \n");
    }
    else
    {
        printf("set object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 销毁内存
    deinitialize_acl_info(&aclinfo);
}
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账号ID，可通过OBS控制台“我的凭证”页面查看。

获取对象访问权限

您可以通过get_object_acl获取对象的访问权限。以下代码展示如何获取对象访问权限：

```
void test_get_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    manager_acl_info *aclinfo = malloc_acl_info();
    aclinfo->object_info.key = "<object key>";
    aclinfo->object_info.version_id = "<object version ID>";
    // 获取对象权限信息
    get_object_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get object acl: --------------");
        printf("%s %d %s %s\n", aclinfo->owner_id, aclinfo->object_delivered,
            aclinfo->object_info.key, aclinfo->object_info.version_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
        printf("get object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 销毁内存
    free_acl_info(&aclinfo);
}
```

9.3 列举对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数list_bucket_objects列举出桶里的对象。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
prefix	char *	可选	限定返回的对象名必须带有prefix前缀。
marker	char *	可选	列举对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。
delimiter	char *	可选	用于对对象名进行分组的字符。对于对象名中包含delimiter的对象，其对象名（如果请求中指定了prefix，则此处的对象名需要去掉prefix）中从首字符至第一个delimiter之间的字符串将作为一个分组并作为commonPrefix返回。 对于并行文件系统，不携带此参数时默认列举是递归列举此目录下所有内容，会列举子目录。在大数据场景下（目录层级深、目录下文件多）的列举，建议设置[delimiter="/"]，只列举当前目录下的内容，不列举子目录，提高列举效率。
maxkeys	int	必选	列举对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
handler	obs_list_objects_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_list_bucket_objects(char *bucket_name)
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境
```

```
变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_list_objects_handler list_bucket_objects_handler =
    {
        { &response_properties_callback, &listobjects_complete_callback },
        &list_objects_callback
    };

    // 用户自定义回调数据
    list_bucket_callback_data data;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    // 列举对象
    list_bucket_objects(&option, "<prefix>", "<marker>", "<delimiter>", "<maxkeys>",
&list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list bucket objects successfully. \n");
    }
    else
    {
        printf("list bucket objects failed(%s).\n",
        obs_get_status_name(data.ret_status));
    }
}
```

说明

- 每次至多返回1000个对象，如果指定桶包含的对象数量大于1000，则返回结果中list_objects_data.is_truncated为true表明本次没有返回全部对象，并可通过list_objects_data.next_marker获取下次列举的起始位置。
- 想获取所有对象，可以采用分页列举的方式。

9.4 删除对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

说明

请您谨慎使用删除操作，如果对象所在的桶未开启多版本控制功能，该对象一旦删除将无法恢复。

删除单个对象

您可以通过delete_object删除单个对象，参数描述如下：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	对象名和版本号，非多版本对象，version设置为0。

字段名	类型	约束	说明
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_delete_object(char *key, char *version_id, char *bucket_name)
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // 创建并初始化对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = key;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };

    // 删除对象
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
    {
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

批量删除对象

您可以通过batch_delete_objects批量删除对象。

每次最多删除1000个对象，并支持两种响应模式：详细（verbose）模式和简单（quiet）模式。

- 详细模式：返回的删除成功和删除失败的所有结果，默认模式。
- 简单模式：只返回的删除过程中出错的结果。

参数描述如下：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
object_info	obs_object_info *	必选	删除的对象名和版本号，非多版本对象，version设置为0。
delobj	obs_delete_object_info*	必选	删除对象个数和指定使用quiet模式。
put_properties	obs_put_properties *	可选	设置删除对象的校验属性。
handler	obs_delete_object_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_batch_delete_objects()
{
    obs_status ret_status = OBS_STATUS_BUTT;

    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);

    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 初始化待删对象信息
    obs_object_info objectinfo[100];
    objectinfo[0].key = "obj1";
    objectinfo[0].version_id = "versionid1";
    objectinfo[1].key = "obj2";
    objectinfo[1].version_id = "versionid2";

    obs_delete_object_info delobj;
    memset_s(&delobj,sizeof(obs_delete_object_info),0,sizeof(obs_delete_object_info));
    delobj.keys_number = 2;

    // 设置响应回调函数
    obs_delete_object_handler handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &delete_objects_data_callback
    };

    // 批量删除对象
    batch_delete_objects(&option, objectinfo, &delobj, 0, &handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test batch_delete_objects successfully. \n");
    }
```



```
    else
    {
        printf("test batch_delete_objects faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

9.5 复制对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

复制对象特性用来为OBS上已经存在的对象创建一个副本。

您可以通过copy_object来复制对象。复制对象时，可重新指定新对象的属性和设置对象权限，且支持条件复制。

说明

- 如果待复制的源对象是归档存储类型，则必须先恢复源对象才能进行复制。

简单复制

copy_object的参数描述如下：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
version_id	char *	可选	对象的版本号
object_info	obs_copy_destination_object_info *	必选	指明拷贝对象信息。
object_info->destination_bucket	char *	必选	目标对象所在桶。
object_info->destination_key	char *	必选	目标对象名称。
object_info->last_modified_return	int64_t *	必选	对象上次修改的时间。
object_info->etag_return_size	int	必选	eTag缓存大小。
object_info->etag_return	char *	必选	eTag缓存。

字段名	类型	约束	说明
is_copy	unsigned int	必选	用来指定新对象的元数据是从源对象中复制，还是用请求中的元数据替换。
put_properties	obs_put_properties*	可选	上传对象属性。
encryption_params	server_side_encryption_params *	可选	服务端加密设置。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码如下：

```
static void test_copy_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置目的对象信息
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // 设置响应回调函数
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 拷贝对象
    copy_object(&option, key, version_id, &objectinfo, 1, NULL, NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


限定条件复制

复制对象时，可以指定一个或多个限定条件，满足限定条件时则进行复制，否则抛出异常，复制对象失败。

您可以使用的限定条件如下：

参数	作用
if_modified_since	如果源对象在指定的时间后有修改，则进行复制，否则抛出异常。
if_not_modified_since	如如果源对象在指定的时间后没有修改，则进行复制，否则抛出异常。
if_match_etag	如果源对象的ETag值与该参数值相同，则进行复制，否则抛出异常。
if_not_match_etag	如果源对象的ETag值与该参数值不相同，则进行复制，否则抛出异常。

说明

- 源对象的ETag值是指源对象数据的MD5校验值。
- 如果包含if_not_modified_since并且不符合，或者包含if_match_etag并且不符合，或者包含if_modified_since并且不符合，或者包含if_not_match_etag并且不符合，则复制失败，抛出异常中HTTP状态码为：412 precondition failed。
- if_modified_since和if_not_match_etag可以一起使用；if_not_modified_since和if_match_etag可以一起使用。

以下代码展示了如何进行限定条件复制：

```
static void test_copy_object_condition()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置目的对象信息
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    //限定条件
    obs_put_properties putProperties={0};
```



```
    init_put_properties(&putProperties);
    putProperties.get_conditions.if_match_etag = "<etag>";
    putProperties.get_conditions.if_modified_since = "<time>";
    putProperties.get_conditions.if_not_match_etag = "<etag>";
    putProperties.get_conditions.if_not_modified_since = "<time>";
    // 设置响应回调函数
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 拷贝对象
    copy_object(&option, key, version_id, &objectinfo, 1, &putProperties,
NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

重写对象访问权限

以下代码展示了如何在复制对象时重写对象访问权限：

```
static void test_copy_object_acl()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置目的对象信息
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    //重写对象访问权限
    obs_put_properties putProperties={0};
    init_put_properties(&putProperties);
    putProperties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ;
    // 设置响应回调函数
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 拷贝对象
    copy_object(&option, key, version_id, &objectinfo, 1, &putProperties,
NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
```



```
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

9.6 重命名对象或目录

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过rename_object来修改对象名和目录名，rename_object接口只适用于并行文件系统，对象桶该接口不支持。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	需要重命名的对象名或目录名。
new_object_name	char *	必选	新名字。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

以下代码展示对象如何重命名：

```
static void test_rename_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 需要重命名的对象名或目录名
    char *key = "put_buffer_test";
    // 新名字
    char *new_key_name = "put_buffer_test_new";
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```

```
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // 重命名对象
    rename_object(&option, key, new_key_name, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("rename_object successfully. \n");
    }
    else
    {
        printf("rename_object failed.\n", obs_get_status_name(ret_status));
    }
}
```

说明

- rename_object只适用于并行文件系统的桶。
- rename时，如果原对象不存在，则会报错（HTTP状态码为404）。

9.7 截断对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过调用truncate_object来对对象进行截断操作，truncate_object接口只适用于并行文件系统的桶，对象桶该接口不支持。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
key	char *	必选	对象名。
object_length	uint64_t	必选	截断到的大小。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

以下代码展示如何截断对象：

```
static void test_truncate_object()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 对象名
    char *key = "put_buffer_test";
    // 对象截断成的大小
```

```
    uint64_t object_length = 10240;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // 截断对象
    truncate_object(&option, key, object_length, &response_handler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("truncate_object successfully. \n");
    }
    else
    {
        printf("truncate_object failed.\n", obs_get_status_name(ret_status));
    }
}
```

说明

- truncate_object只适用于并行文件系统的桶。
- truncate_object时，如果对象不存在，则会报错（HTTP状态码为404）。

10 临时授权访问

10.1 什么是临时授权访问

须知

开发过程中，您有任何问题可以在github上**提交issue**。

临时授权访问是指通过访问密钥、请求方法类型、请求参数等信息生成一个临时访问权限的URL，这个URL中会包含鉴权信息，您可以使用该URL进行访问OBS服务进行特定操作。在生成URL时，您需要指定URL的有效期。生成临时授权访问的URL是通过设置结构体temp_auth_configure来实现的。

temp_auth_configure结构体存在于obs_options结构体中。该方法适用于每个C SDK接口。

参数	作用	SDK中对应的结构体
expires	生成的临时URL的有效期	obs_options.temp_auth_configure
temp_auth_callback	回调函数用于返回生成的临时URL	
callback_data	回调数据	

⚠ 注意

如果遇到跨域报错、签名不匹配问题，请参考以下步骤排查问题：

1. 未配置跨域，需要在控制台配置CORS规则，请参考**配置桶允许跨域请求**。
2. 签名计算问题，请参考**URL中携带签名**排查签名参数是否正确；比如上传对象功能，后端将Content-Type参与计算签名生成授权URL，但是前端使用授权URL时没有设置Content-Type字段或者传入错误的值，此时会出现跨域错误。解决方案为：Content-Type字段前后端保持一致。

10.2 临时授权访问示例

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过SDK接口传入temp_auth_configure结构体参数生成临时授权访问的URL。以下代码展示了如何生成常用操作的URL，包括：创建桶、上传对象、下载对象、列举对象、删除对象。

说明

以下示例中生成的URL记录在tmpAuthUrl.txt文件中，你可以使用生成的URL执行对应的操作。

10.2.1 生成创建桶的 URL

须知

开发过程中，您有任何问题可以在github上**提交issue**。

```
static void test_create_bucket_auth()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    //回调数据
    tempauth.callback_data = (void *)(&ptrResult);
    // 有效时间
    tempauth.expires = 10;
    // 回调函数 返回生成的临时URL
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // 回调函数赋值
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 接口调用
    create_bucket(&option, canned_acl, NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("the temporary signature url of create bucket generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
```

```
    }
    else
    {
        printf(" the temporary signature url of create bucket generation failed(%s).\n",
obs_get_status_name(ret_status));
    }
}
```

10.2.2 生成上传对象的 URL

须知

开发过程中，您有任何问题可以在github上**提交issue**。

```
static void test_put_object_auth()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    //回调数据
    tempauth.callback_data = (void *)(&ptrResult);
    // 有效时间
    tempauth.expires = 10;
    // 回调函数 返回生成的临时URL
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // 初始化结构体put_properties
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 回调数据
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file(file_name, &data);
    // 设置响应回调函数
    obs_put_object_handler putobjectHandler =
    {
    { &response_properties_callback,
        &response_complete_callback },
      &put_buffer_object_data_callback
     };
    // 上传数据流
    put_object(&option,key,content_length,&put_properties,0,&putobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of put object from file generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of put object from file generation faied(%s).\n",
obs_get_status_name(data.ret_status));
```



```
    }
}
```

10.2.3 生成下载对象的 URL

须知

开发过程中，您有任何问题可以在github上**提交issue**。

```
static void test_get_object_auth()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    //回调数据
    tempauth.callback_data = (void *)(&ptrResult);
    // 有效时间
    tempauth.expires = 10;
    // 回调函数 返回生成的临时URL
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // 下载对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    // 下载后保存在本地文件信息
    char *file_name = local_file_name;
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // 回调函数
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };
    // 下载对象
    get_object(&option,&object_info,0,0,&getobjectHandler,&data);
    fclose(data.outfile);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of get object generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of get object generation faied(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

10.2.4 生成列举对象的 URL

须知

开发过程中，您有任何问题可以在github上**提交issue**。

```
static void test_list_object_auth()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    //回调数据
    tempauth.callback_data = (void *)(&ptrResult);
    // 有效时间
    tempauth.expires = 10;
    // 回调函数 返回生成的临时URL
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    //列举对象的最大数目
    int maxkeys  =  100;
    obs_list_objects_handler list_bucket_objects_handler =
    {
        { &response_properties_callback, &list_object_complete_callback },
        &list_objects_callback
     };
    list_object_callback_data data;
    memset(&data, 0, sizeof(list_object_callback_data));
    list_bucket_objects(&option, NULL, data.next_marker,NULL, maxkeys,
    &list_bucket_objects_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {printf("the temporary signature url of list bucket objects generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of list bucket objects generation failed(%s).\n",
                        obs_get_status_name(data.ret_status));
    }
}
```

10.2.5 生成删除对象的 URL

须知

开发过程中，您有任何问题可以在github上**提交issue**。

```
static void test_delete_object_auth()
{
```



```
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    temp_auth_configure tempauth;

    tempAuthResult  ptrResult;
    memset_s(&ptrResult,sizeof(tempAuthResult),0,sizeof(tempAuthResult));
    //回调数据
    tempauth.callback_data = (void *)(&ptrResult);
    // 有效时间
    tempauth.expires = 10;
    // 回调函数 返回生成的临时URL
    tempauth.temp_auth_callback = &tempAuthCallBack_getResult;
    option.temp_auth = &tempauth;
    // 对象的信息
    obs_object_info object_info;
    memset(&object_info,0,sizeof(obs_object_info));
    object_info.key = key;
    // 回调函数赋值
    obs_response_handler resqonseHandler =
    {
    &response_properties_callback,
        &response_complete_callback
    };
    // 删除对象
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {printf("the temporary signature url of delete object generated successfully. The result is recorded in the tmpAuthUrl.txt file. \n");
    }
    else
    {
        printf("the temporary signature url of delete object generation failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


11 自定义域名访问

11.1 自定义域名相关说明

使用自定义域名访问服务端之前，需要先在console界面配置自定义域名。

自定义域名访问介绍与配置

当以自定义域名访问OBS桶时，需要先将该自定义域名同对应OBS桶访问域名进行绑定，相关配置请参见**自定义域名绑定简介**，**自定义域名绑定配置**。

⚠ 注意

当在自定义域名上配置了CDN加速服务，即自定义域名为CDN服务的加速域名时，需要额外对CDN服务进行配置，以保证可以正常采用自定义域名访问OBS服务。

以华为云CDN服务为例，相关配置如下所示：

步骤1 登录华为云CDN服务，从CDN服务左侧列表中选择域名管理项，在该项中可以查看到所有配置的CDN服务域名信息。

步骤2 配置源站。单击要使用的自定义域名项，进入域名配置界面，编辑源站配置，选择主源站类型为源站域名类型，对应源站为要访问的OBS桶域名。

步骤3 配置回源HOST。回源HOST必须指定为加速域名即访问OBS服务时访问的自定义域名，否则可能会出现回源鉴权失败的问题。

----结束

11.2 c sdk 通过自定义域名访问 obs

通过sdk，使用自定义域名访问obs，可以完全复用对应接口的示例，只需要注意在设置option的时候，按如下方式设置：

```
option.bucket_options.useCname = true;
option.bucket_options.host_name = "yourCustomDomain";
```

sdk通过自定义域名上传对象：

```
static void test_put_object_from_file()
{
    // 上传对象名
    char *key = "put_file_test";
    // 上传的文件
    char file_name[256] = "./put_file_test.txt";
    uint64_t content_length = 0;
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
```



```
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.certificate_info = "<Your certificate>";

    // 设置自定义域名
    option.bucket_options.useCname = true;
    option.bucket_options.host_name = "yourCustomDomain";
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);

    // 初始化存储上传数据的结构体
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 打开文件，并获取文件长度
    content_length = open_file_and_get_length(file_name, &data);
    // 设置回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_file_complete_callback },
        &put_file_data_callback
    };

    put_object(&option, key, content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

sdk通过自定义域名下载对象：

```
static void test_get_object()
{
    char *file_name = "./test";
    obs_object_info object_info;
    // 初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置自定义域名
    option.bucket_options.useCname = true;
    option.bucket_options.host_name = "yourCustomDomain";

    // 设置下载的对象
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = "<object key>";
    object_info.version_id = "<object version ID>";
    //根据业务需要设置存放下载对象数据的结构
    get_object_callback_data data;
```



```
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);
    // 定义范围下载参数
    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);
    getcondition.start_byte = "<start byte>";
    // 下载长度，默认0，读到对象尾
    getcondition.byte_count = "<byte count>";

    // 定义下载的回调函数
    obs_get_object_handler get_object_handler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, 0, &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

12 多版本控制

12.1 多版本控制简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS支持保存一个对象的多个版本，使您更方便地检索和还原各个版本，在意外操作或应用程序故障时快速恢复数据。

更多关于多版本控制的内容请参见**多版本控制**。

12.2 设置桶多版本状态

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过set_bucket_version_configuration设置桶的多版本状态。OBS中的桶支持两种多版本状态：

多版本状态	说明	OBS C SDK对应值
启用状态	1. 上传对象时，系统为每一个对象创建一个唯一版本号，上传同名的对象将不再覆盖旧的对象，而是创建新的不同版本号的同名对象。 2. 可以指定版本号下载对象，不指定版本号默认下载最新对象。 3. 删除对象时可以指定版本号删除，不带版本号删除对象仅产生一个带唯一版本号的删除标记，并不删除对象。 4. 列出桶内对象列表（list_bucket_objects）时默认列出最新对象列表，可以指定列出桶内所有版本对象列表（list_versions）。 5. 除了删除标记外，每个版本的对象存储均需计费。	OBS_VERSION_STATUS_ENABLED
暂停状态	1. 旧的版本数据继续保留。 2. 上传对象时创建对象的版本号为null，上传同名的对象将覆盖原有同名的版本号为null的对象。 3. 可以指定版本号下载对象，不指定版本号默认下载最新对象。 4. 删除对象时可以指定版本号删除，不带版本号删除对象将产生一个版本号为null的删除标记，并删除版本号为null的对象。 5. 除了删除标记外，每个版本的对象存储均需计费。	OBS_VERSION_STATUS_SUSPENDED

以下代码展示了如何设置桶的多版本状态：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。

字段名	类型	约束	说明
version_status	char *	必选	表示桶的多版本状态：OBS_VERSION_STATUS_ENABLED，OBS_VERSION_STATUS_SUSPENDED。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

```
static void test_set_bucket_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 开启桶的多版本控制
    set_bucket_version_configuration(&option, OBS_VERSION_STATUS_ENABLED,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket version successfully. \n");
    }
    else
    {
        printf("set bucket version failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

12.3 查看桶多版本状态

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_bucket_version_configuration查看桶的多版本状态。以下代码展示了如何查看桶的多版本状态：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
status_return_size	int	必选	多版本状态缓存大小。
status_return	char *	必选	多版本状态缓存。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

```
static void test_get_bucket_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 获取桶的多版本状态
    char status[OBS_COMMON_LEN_256] = {0};
    get_bucket_version_configuration(&option, sizeof(status), status,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get bucket version successfully.\n policy=(%s)\n", status);
    }
    else
    {
        printf("get bucket version failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

12.4 获取多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_object接口传入版本号（obs_object_info .version_id）来获取多版本对象，示例代码如下：

```
static void test_get_object_version()
{
    // 创建并初始化对象，通过obs_object_info.key指定对象版本
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 创建并初始化回调数据
    char *file_name = "./test";
    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = NULL;
    data.outfile = write_to_file(file_name);
    // 设置响应回调函数
    obs_get_object_handler getobjectHandler =
    {
        { &response_properties_callback,
          &get_object_complete_callback},
        &get_object_data_callback
    };
    // 获取多版本对象
    get_object(&option,&object_info,0,0,&getobjectHandler,&data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object successfully. \n");
    }
    else
    {
        printf("get object faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

说明

如果版本号为空则默认下载最新版本的对象。

12.5 复制多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过copy_object接口传入版本号（version_id）来复制多版本对象，示例代码如下：

```
static void test_copy_object_version()
{
```

```
    obs_status ret_status = OBS_STATUS_BUTT;
    char eTag[OBS_COMMON_LEN_256] = {0};
    int64_t lastModified;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置目的对象信息
    obs_copy_destination_object_info objectinfo ={0};
    objectinfo.destination_bucket = target_bucket;
    objectinfo.destination_key = destinationKey;
    objectinfo.etag_return = eTag;
    objectinfo.etag_return_size = sizeof(eTag);
    objectinfo.last_modified_return = &lastModified;
    // 设置响应回调函数
    obs_response_handler responseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 拷贝对象
    copy_object(&option, key, version_id, &objectinfo, 1, NULL, NULL,&responseHandler,&ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_copy_object  successfully. \n");
    }
    else
    {
        printf("test_copy_object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

12.6 恢复多版本归档存储对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过restore_object接口传入版本号（obs_object_info .version_id）来恢复多版本归档存储对象，示例代码如下：

```
static void test_restore_object_version()
{
    // 创建并初始化对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;
    object_info.version_id = versionid;
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存
```

```
放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    obs_tier tier = OBS_TIER_EXPEDITED;
    // 设置响应回调函数
    obs_response_handler handler =
    {
      &response_properties_callback,
      &response_complete_callback
    };
    // 恢复多版本归档存储对象
    restore_object(&option, &object_info, days,tier,&handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("restore object successfully. \n");
    }
    else
    {
        printf("restore object faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

⚠ 注意

重复恢复归档存储数据时在延长恢复有效期的同时，也将会对恢复时产生的恢复费用进行重复收取。产生的标准存储类别的对象副本有效期将会延长，并且收取延长时间段产生的标准存储副本费用。

12.7 列举多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过list_versions列举多版本对象。

该接口可设置的参数如下：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
prefix	char *	可选	限定返回的对象名必须带有prefix前缀。
key_marker	char *	可选	列举多版本对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。

字段名	类型	约束	说明
delimiter	char *	可选	用于对对象名进行分组的字符。对于对象名中包含delimiter的对象，其对象名（如果请求中指定了prefix，则此处的对象名需要去掉prefix）中从首字符至第一个delimiter之间的字符串将作为一个分组并作为common_prefixes返回。
version_id_marker	char *	可选	与key_marker配合使用，返回的对象列表将是对象名和版本号按照字典序排序后该参数以后的所有对象。
maxkeys	int	必选	列举对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
handler	obs_list_versions_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

说明

- 如果version_id_marker不是key_marker的一个版本号，则该参数无效。
- list_versions返回结果包含多版本对象和对象删除标记。

以下代码展示如何列举多版本对象：

```
static void test_list_versions()
{
    char *prefix = "o";
    char *key_marker = "obj";
    char *delimiter = "/";
    int maxkeys  =  10;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_list_versions_handler list_versions_handler =
    {
        { &response_properties_callback, &list_versions_complete_callback },
        &listVersionsCallback
    };
```



```
    // 创建并初始化回调数据
    list_versions_callback_data data;
    char* version_id_marker = NULL;
    memset(&data, 0, sizeof(list_bucket_callback_data));
    data.ret_status = OBS_STATUS_BUTT;
    snprintf(data.next_key_marker, sizeof(data.next_key_marker), "%s", key_marker);
    if (version_id_marker)
    {
        snprintf(data.next_versionId_marker, sizeof(data.next_versionId_marker), "%s", version_id_marker);
    }
    data.keyCount = 0;
    data.allDetails = 1;
    data.is_truncated = 0;
    // 列举多版本对象，通过prefix指定对象前缀，maxkeys指定数目列举，可实现分页列举，delimiter指定分组列举的字符，按文件夹分组，则delimiter设置为”/”，通过key_marker指定多版本起始位置，version_id_marker指定版本号
    list_versions(&option, prefix, key_marker, delimiter, maxkeys, version_id_marker,
                  &list_versions_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("list versions successfully. \n");
    }
    else
    {
        printf("list versions failed(%s).\n", obs_get_status_name(data.ret_status));
    }
}
```

说明

- 每次至多返回1000个多版本对象，如果指定桶包含的对象数量大于1000，则返回结果中is_truncated为true表明本次没有返回全部对象，并可通过next_key_marker和next_versionId_marker获取下次列举的起始位置。
- 如果想获取指定桶包含的所有多版本对象，可以采用分页列举的方式。

12.8 多版本对象权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。

设置多版本对象访问权限

您可以通过set_object_acl接口传入版本号（version_id）设置多版本对象的访问权限，示例代码如下：

```
static void test_set_object_acl_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
```

```
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 初始化acl信息，指定对象名和版本号
    manager_acl_info aclinfo;
    init_acl_info(&aclinfo);
    aclinfo.object_info.key = key;
    aclinfo.object_info.version_id = version_id;
    // 设置对象访问权限
    set_object_acl(&option, &aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set object acl successfully. \n");
    }
    else
    {
        printf("set object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 释放内存
    deinitialize_acl_info(&aclinfo);
}
```

说明

ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账号ID，可通过OBS控制台“我的凭证”页面查看。

获取多版本对象访问权限

您可以通过get_object_acl接口传入版本号（version_id）获取多版本对象的访问权限，示例代码如下：

```
static void test_get_object_acl_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    //从环境变量读取ak/sk
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 创建acl结构，指定对象名和版本号
    manager_acl_info *aclinfo = malloc_acl_info();
    aclinfo->object_info.key = key;
    aclinfo->object_info.version_id = version_id;
    // 获取对象权限
    get_object_acl(&option, aclinfo, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("get object acl: ----------");
        printf("%s %s %s %s\n", aclinfo->owner_id, aclinfo->owner_display_name,
            aclinfo->object_info.key, aclinfo->object_info.version_id);
        if (aclinfo->acl_grant_count_return)
        {
            print_grant_info(*aclinfo->acl_grant_count_return, aclinfo->acl_grants);
        }
    }
    else
    {
```



```
        printf("get object acl failed(%s).\n", obs_get_status_name(ret_status));
    }
    // 释放内存
    free_acl_info(&aclinfo);
}
```

12.9 删除多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。

删除单个多版本对象

您可以通过delete_object接口传入版本号（version_id）删除多版本对象，示例代码如下：

```
static void test_delete_object_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化对象信息
    obs_object_info object_info;
    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key = key;
    object_info.version_id = version_id;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler resqonseHandler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 删除对象
    delete_object(&option,&object_info,&resqonseHandler, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        printf("delete object successfully. \n");
    }
    else
    {
        printf("delete object failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

批量删除多版本对象

您可以通过batch_delete_objects接口传入每个待删除对象的版本号（version_id）批量删除多版本对象，示例代码如下：

```
static void test_batch_delete_object_version()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 初始化待删对象信息
    obs_object_info objectinfo[100];
    objectinfo[0].key = "obj1";
    objectinfo[0].version_id = "versionid1";
    objectinfo[1].key = "obj2";
    objectinfo[1].version_id = "versionid2";
    obs_delete_object_info delobj;
    memset_s(&delobj,sizeof(obs_delete_object_info),0,sizeof(obs_delete_object_info));
    delobj.keys_number = 2;
    // 设置响应回调函数
    obs_delete_object_handler handler =
    {
        {&response_properties_callback, &response_complete_callback},
        &delete_objects_data_callback
    };
    // 批量删除对象
    batch_delete_objects(&option, objectinfo, &delobj, 0, &handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test batch_delete_objects successfully. \n");
    }
    else
    {
        printf("test batch_delete_objects faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```


13 生命周期管理

13.1 生命周期管理简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS允许您对桶设置生命周期规则，实现自动转换对象的存储类型、自动淘汰过期的对象，以有效利用存储特性，优化存储空间。针对不同前缀的对象，您可以同时设置多条规则。一条规则包含：

- 规则ID，用于标识一条规则，不能重复。
- 受影响的对象前缀，此规则只作用于符合前缀的对象。
- 最新版本对象的转换策略，指定方式为：
  a. 指定满足前缀的对象创建后第几天时转换为指定的存储类型。
  b. 直接指定满足前缀的对象转换为指定的存储类型的日期。
- 最新版本对象过期时间，指定方式为：
  a. 指定满足前缀的对象创建后第几天时过期。
  b. 直接指定满足前缀的对象过期日期。
- 历史版本对象转换策略，指定方式为：
  - 指定满足前缀的对象成为历史版本后第几天时转换为指定的存储类型。
- 历史版本对象过期时间，指定方式为：
  - 指定满足前缀的对象成为历史版本后第几天时过期。
- 是否生效标识。

更多关于生命周期的内容请参考**生命周期管理**。

说明

- 对象过期后会被OBS服务端自动删除。
- 对象转换策略中的时间必须早于对象过期时间；历史版本对象转换策略中的时间也必须早于历史版本对象的过期时间。
- 桶必须开启多版本状态，历史版本对象转换策略和历史版本对象过期时间配置才能生效。

13.2 设置生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数set_bucket_lifecycle_configuration设置桶的生命周期规则。

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
bucket_lifecycle_conf	obs_lifecycle_conf *	必选	桶生命周期配置说明，具体说明请参看下表。
blcc_number	unsigned int	必选	数组bucket_lifecycle_conf的数组成员个数。
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

桶生命周期配置结构obs_lifecycle_conf描述如下表：

字段名	类型	约束	说明
date	const char *	如果没有days元素，且没有transition, noncurrent_version_days, noncurrent_version_transition，则必选	表示针对最新版本的对象过期规则生效的时间。该值必须兼容ISO8601格式，而且必须是UTC午夜0点。

字段名	类型	约束	说明
days	const char *	如果没有date元素，且没有transition, noncurrent_version_days, noncurrent_version_transition，则必选	表示在对象创建时间后第几天时过期规则生效（仅针对对象的最新版本）。
id	const char *	可选	一条Rule的标识，由不超过255个字符的字符串组成。
prefix	const char *	必选	对象名前缀，用以标识哪些对象可以匹配到当前这条Rule。
status	const char *	必选	标识当前这条Rule是否启用。
noncurrent_version_days	const char *	可选	表示对象在成为历史版本之后第几天时过期规则生效（仅针对历史版本） 生命周期配置中表示历史版本过期时间的Container。您可以将该动作设置在已启用多版本（或暂停）的桶，来让系统删除对象的满足特定生命周期的历史版本（仅针对历史版本）。
transition_num	unsigned int	如果transition非空，则必选	数组transition的数组成员个数。
transition	obs_lifecycle_transtion *	如果没有date，days，noncurrent_version_transition或者noncurrent_version_days，则必选	生命周期配置中表示迁移时间和迁移后对象存储级别的元素（仅针对对象的最新版本）。
transition->date	const char *	如果transition存在且没有transition.days元素，则必选	表示针对最新版本的对象转换规则生效的时间。该值必须兼容ISO8601格式，而且必须是UTC午夜0点。
transition->days	const char *	如果transition存在且没有transition.date元素，则必选	表示在对象创建时间后第几天时转换规则生效（仅针对对象的最新版本）。
transition->storage_class	obs_storage_class	如果transition存在，则必须按	表示最新版本对象将被修改成存储级别。

字段名	类型	约束	说明
noncurrent_version_transition_num	unsigned int	如果obs_lifecycle_noncurrent_transtion非空，则必选	数组noncurrent_version_transition的数组成员个数
noncurrent_version_transition	obs_lifecycle_noncurrent_transtion *	如果没有date，days，transition或者noncurrent_version_days，则必选	生命周期配置中表示对象的历史版本迁移时间和迁移后对象存储级别的元素。
noncurrent_version_transition->noncurrent_version_days	const char *	如果noncurrent_version_transition存在，必选	表示对象在成为历史版本之后第几天时转换规则生效（仅针对历史版本）
noncurrent_version_transition->storage_class	obs_storage_class	如果noncurrent_version_transition存在，必选	表示历史版本对象将被修改成的存储级别。

设置对象转换策略

以下代码展示了如何设置最新版本对象和历史版本对象的转换策略：

```
static void test_set_bucket_lifecycle_configuration1()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置完成的回调函数
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };
    obs_lifecycle_conf bucket_lifecycle_conf;
    memset(&bucket_lifecycle_conf, 0, sizeof(obs_lifecycle_conf));
    //生命周期规则的id
    bucket_lifecycle_conf.id = "test3";
    // 指定前缀"test"
    bucket_lifecycle_conf.prefix = "bcd";
    // 该生命周期规则生效
    bucket_lifecycle_conf.status = "Enabled";
    // 指定满足前缀的对象创建10天后过期
```



```
    bucket_lifecycle_conf.days = "10";
    obs_lifecycle_transtion transition;
    memset(&transition, 0, sizeof(obs_lifecycle_transtion));
    // 指定满足前缀的对象创建30天后转换
    transition.days = "30";
    // 指定对象转换后的存储类型
    transition.storage_class = OBS_STORAGE_CLASS_STANDARD_IA;
    bucket_lifecycle_conf.transition = &transition;
    bucket_lifecycle_conf.transition_num = 1;
    obs_lifecycle_noncurrent_transtion noncurrent_transition;
    memset(&noncurrent_transition, 0, sizeof(obs_lifecycle_noncurrent_transtion));
    // 指定满足前缀的对象的历史版本30天后转换
    noncurrent_transition.noncurrent_version_days = "30";
    // 指定满足前缀的对象的历史版本转换后的存储类型
    noncurrent_transition.storage_class = OBS_STORAGE_CLASS_STANDARD_IA;
    bucket_lifecycle_conf. noncurrent_version_transition = &noncurrent_transition;
    bucket_lifecycle_conf.noncurrent_version_transition_num = 1;
    set_bucket_lifecycle_configuration(&option, &bucket_lifecycle_conf, 1,
                &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket lifecycle configuration success.\n");
    }
    else
    {
        printf("set bucket lifecycle configuration failed(%s).\n",
                obs_get_status_name(ret_status));
    }
}
```

设置对象过期时间

以下代码展示了如何设置最新版本对象和历史版本对象的过期时间：

```
static void test_set_bucket_lifecycle_configuration2()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 设置option
    init_obs_options(&option);
    option.bucket_options.host_name = HOST_NAME;
    option.bucket_options.bucket_name = bucket_name;

    //从环境变量读取ak/sk
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置完成的回调函数
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };
    obs_lifecycle_conf bucket_lifecycle_conf;
    memset(&bucket_lifecycle_conf, 0, sizeof(obs_lifecycle_conf));
    //生命周期规则的id
    bucket_lifecycle_conf.id = "test1";
    // 指定前缀"test"
    bucket_lifecycle_conf.prefix = "test";
    // 指定满足前缀的对象创建10天后过期
    bucket_lifecycle_conf.days = "10";
    // 指定满足前缀的对象的历史版本20天后过期
    bucket_lifecycle_conf.noncurrent_version_days = "20";
    // 该生命周期规则生效
    bucket_lifecycle_conf.status = "Enabled";
    set_bucket_lifecycle_configuration(&option, &bucket_lifecycle_conf, 1,
              &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
         printf("set bucket lifecycle configuration success.\n");
    }
    else
    {
```



```
        printf("set bucket lifecycle configuration failed(%s).\n",
               obs_get_status_name(ret_status));
    }
}
```

13.3 查看生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数get_bucket_lifecycle_configuration()查看桶的生命周期规则。以下代码展示了如何查看桶的生命周期规则：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_lifecycle_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_get_bucket_lifecycle_configuration()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置回调函数
    obs_lifecycle_handler lifeCycleHandlerEx =
    {
        {&response_properties_callback, &response_complete_callback},
        &getBucketLifecycleConfigurationCallbackEx
    };
    get_bucket_lifecycle_configuration(&option, &lifeCycleHandlerEx, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get_lifecycle_config success.\n");
    }
    else
    {
        printf("get_lifecycle_config faied(%s).\n", obs_get_status_name(ret_status));
```

```
    }
}
```

13.4 删除生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数delete_bucket_lifecycle_configuration()删除桶的生命周期规则。以下代码展示了如何删除桶的生命周期规则：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_delete_bucket_lifecycle_configuration()
{
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    delete_bucket_lifecycle_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("test_delete_lifecycle_config success.\n");
    }
    else
    {
        printf("test_delete_lifecycle_config faied(%s).\n",
            obs_get_status_name(ret_status));
    }
}
```

14 跨域资源共享

14.1 跨域资源共享简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

跨域资源共享（CORS）允许Web端的应用程序访问不属于本域的资源。OBS提供接口方便开发者控制跨域访问的权限。

更多关于跨域资源共享的内容请参考**跨域资源访问**。

14.2 设置跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数set_bucket_cors_configuration()设置桶的跨域规则，如果原规则存在则覆盖原规则。以下代码展示了如何设置跨域规则：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
obs_cors_conf_info	obs_bucket_cors_conf *	必选	CORS规则具体内容，具体说明请参看下表。

字段名	类型	约束	说明
conf_num	unsigned int	必选	数组obs_cors_conf_info的数组成员个数。
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

CORS规则结构obs_bucket_cors_conf描述如下表：

字段名	类型	约束	说明
id	const char *	可选	桶内对象名称。
allowed_method	const char **	必选	CORS规则允许的Method。
allowed_method_number	unsigned int	必选	allowed_method的个数。
allowed_origin	const char **	必选	CORS规则允许的Origin（表示域名的字符串），可以带一个匹配符“*”。每一个allowedOrigin可以带最多一个“*”通配符。
allowed_origin_number	unsigned int	必选	allowed_origin的个数。
allowed_header	const char **	可选	配置CORS请求中允许携带的“Access-Control-Request-Headers”头域。如果一个请求带了“Access-Control-Request-Headers”头域，则只有匹配上AllowedHeader中的配置才认为是一个合法的CORS请求。每一个allowed_header可以带最多一个“*”通配符，不可出现空格。
allowed_header_number	unsigned int	可选	allowed_header的个数
max_age_seconds	const char *	可选	客户端可以缓存的CORS响应时间，以秒为单位。每个CORS Rule可以包含至多一个max_age_seconds，可以设置为负值。
expose_header	const char **	可选	CORS响应中带的附加头域，给客户端提供额外的信息，不可出现空格。

字段名	类型	约束	说明
expose_header_number	unsigned int	可选	expose_header的个数。

示例代码

```
static void test_set_bucket_cors()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    obs_bucket_cors_conf bucketCorsConf;
    // 设置option
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置回调函数
    obs_response_handler response_handler =
    {
        NULL, &response_complete_callback
    };

    char *id_1= "1";
    // 指定浏览器对特定资源的预取(OPTIONS)请求返回结果的缓存时间,单位为秒
    char *max_age_seconds = "100";
    // 指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)
    const char* allowedMethod_1[5] = {"GET","PUT","HEAD","POST","DELETE"};
    // 指定允许跨域请求的来源
    const char* allowedOrigin_1[2] = {"obs.xxx.com", "www.example.com"};
    // 指定允许用户从应用程序中访问的header
    const char* allowedHeader_1[2] = {"header-1", "header-2"};
    // 响应中带的附加头域
    const char* exposeHeader_1[2]  = {"hello", "world"};

    memset(&bucketCorsConf, 0, sizeof(obs_bucket_cors_conf));
    bucketCorsConf.id = id_1;
    bucketCorsConf.max_age_seconds = max_age_seconds;
    bucketCorsConf.allowed_method = allowedMethod_1;
    bucketCorsConf.allowed_method_number = 5;
    bucketCorsConf.allowed_origin = allowedOrigin_1;
    bucketCorsConf.allowed_origin_number = 2;
    bucketCorsConf.allowed_header = allowedHeader_1;
    bucketCorsConf.allowed_header_number = 2;
    bucketCorsConf.expose_header = exposeHeader_1;
    bucketCorsConf.expose_header_number = 2;

    set_bucket_cors_configuration(&option, &bucketCorsConf, 1, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set_bucket_cors success.\n");
    }
    else {
        printf("set_bucket_cors faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

allowed_origin、allowed_method、allowed_header都能够最多支持一个通配符“*”。“*”表示对于所有的域来源、操作或者头域都满足。

14.3 查看跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数get_bucket_cors_configuration()查看桶的跨域规则。以下代码展示了如何查看跨域规则：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_cors_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_get_cors_config()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 设置option
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置回调函数
    obs_cors_handler cors_handler_info =
    {
        {&response_properties_callback, &response_complete_callback},
        &get_cors_info_callback
    };

    get_bucket_cors_configuration(&option, &cors_handler_info, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get_cors_config success.\n");
    }
    else {
        printf("get_cors_config faied(%s).\n", obs_get_status_name(ret_status));
```

```
    }
}
```

14.4 删除跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过函数delete_bucket_cors_configuration()删除桶的跨域规则。以下代码展示了如何删除跨域规则：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

```
static void test_delete_cors_config()
{
    obs_options option;
    obs_status  ret_status = OBS_STATUS_BUTT;
    // 设置option
    init_obs_options(&option);
    option->bucket_options.hostName = "<your-endpoint>";
    option->bucket_options.bucketName = "<Your bucketname>";


    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option->bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option->bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };

    delete_bucket_cors_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("delete_cors_config success.\n");
    }
    else
    {
        printf("delete_cors_config faied(%s).\n", obs_get_status_name(ret_status));
    }
}
```

15 设置访问日志

15.1 日志简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS允许您对桶设置访问日志记录，设置之后对于桶的访问会被记录成日志，日志存储在OBS上您指定的目标桶中。

更多关于访问日志的内容请参考**日志记录**。

15.2 开启桶日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过set_bucket_logging_configuration_obs开启桶日志功能。

须知

日志目标桶与源桶必须在同一个区域（region）。

说明

如果桶的存储类型为低频访问存储或归档存储，则不能作为日志目标桶。

以下代码展示了如何直接开启桶日志：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
target_bucket	char *	必选	在生成日志时，源桶的所有者可以指定一个目标桶，将生成的所有日志放到该桶中。 在OBS系统中，支持多个源桶生成的日志放在同一个目标桶中，如果这样做，就需要指定不同的target_prefix以达到为来自不同源桶的日志分类的目的。
target_prefix	char *	必选	通过该元素可以指定一个前缀给一类日志生成的对象。
agency	char *	设置logging时必选，关闭logging时勿选。	产生logging日志桶owner创建委托OBS上传logging日志的委托名。
acl_group	obs_acl_group *	可选	权限信息组结构体。
acl_group->acl_grant_count	int	可选	返回obs_acl_grant的个数。
acl_group->acl_grants	obs_acl_grant *	可选	权限信息结构体指针，请查看**5.5-管理桶访问权限**，**表2 权限信息结构体obs_acl_grant描述**。
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_set_bucket_logging_configuration()
{
    // 设置目标桶日志投递组用户写权限和读ACP权限
    set_log_delivery_acl(bucket_name_target);
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
```

```
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 设置桶日志配置
    set_bucket_logging_configuration_obs(&option, bucket_name_target, "prefix-log",  "Your agency",
        NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket(%s) logging successfully. \n",bucket_name_src);
    }
    else
    {
        printf("set bucket logging faied(%s).\n",obs_get_status_name(ret_status));
    }
}
```

为日志对象设置权限

以下代码展示了如何为日志对象设置权限：

```
static void test_set_bucket_logging_configuration_acl()
{
    // 设置目标桶日志投递组用户写权限和读ACP权限
    set_log_delivery_acl(bucket_name_target);
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
   //从环境变量读取ak/sk
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    int aclGrantCount = 2;
    obs_acl_grant acl_grants[2] = {0};
    // 为OBS授权用户设置对日志文件的全部权限
    acl_grants[0].grantee_type = OBS_GRANTEE_TYPE_CANONICAL_USER;
    strcpy(acl_grants[0].grantee.canonical_user.id, "userid1");
    strcpy(acl_grants[0].grantee.canonical_user.display_name, "dis1");
    acl_grants[0].permission = OBS_PERMISSION_READ ;
    // 为所有用户设置对日志对象的读权限
    acl_grants[1].grantee_type = OBS_GRANTEE_TYPE_ALL_OBS_USERS;
    acl_grants[1].permission = OBS_PERMISSION_READ ;
    obs_acl_group g;
    g.acl_grants = acl_grants;
    g.acl_grant_count = aclGrantCount;
    // 设置桶日志配置
    set_bucket_logging_configuration_obs(&option, bucket_name_target, "prefix-log",  "Your agency",
        &g, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("set bucket(%s) logging successfully. \n", bucket_name_src);
    }
    else
    {
```



```
        printf("set bucket logging faied(%s).\n",obs_get_status_name(ret_status));
    }
}
```

15.3 查看桶日志配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_bucket_logging_configuration查看桶日志配置。以下代码展示了如何查看桶日志配置：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
logging_message_data	bucket_logging_message *	必选	当前桶日志管理配置情况，bucket_logging_message说明请参看下表。
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

当前桶日志管理配置结构bucket_logging_message描述如下表：

字段名	类型	说明
target_bucket	char *	在生成日志时，源桶的owner可以指定一个目标桶，将生成的所有日志放到该桶中。在OBS系统中，支持多个源桶生成的日志放在同一个目标桶中，如果这样做，就需要指定不同的target_prefix以达到为来自不同源桶的日志分类的目的。
target_bucket_size	int	target_bucket总大小。
target_prefix	char *	通过该元素可以指定一个前缀给一类日志生成的对象。
target_prefix_size	int	target_prefix总大小。
acl_grants	obs_acl_grant *	权限信息结构体指针。
acl_grant_count	int *	返回的acl_grants个数的指针。

字段名	类型	说明
agency	char *	产生logging日志桶Owner创建委托OBS上传logging日志的委托名。
agency_size	int	agency总大小。

示例代码

```
static void test_get_bucket_logging_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback, &response_complete_callback
    };
    // 初始化日志配置结构体
    bucket_logging_message logging_message;
    init_bucket_get_logging_message(&logging_message);
    // 获取桶日志配置
    get_bucket_logging_configuration(&option, &response_handler, &logging_message, &ret_status);
    if (OBS_STATUS_OK == ret_status)
    {
        if (logging_message.target_bucket)
        {
            printf("Target_Bucket: %s\n", logging_message.target_bucket);
            if ( logging_message.target_prefix)
            {
                printf("Target_Prefix: %s\n",  logging_message.target_prefix);
            }
            if (logging_message.agency && logging_message.agency[0] != '\0')
            {
                printf(" Agency: %s\n",  logging_message.agency);
            }
            print_grant_info(*logging_message.acl_grant_count, logging_message.acl_grants);
        }
        else
        {
            printf("Service logging is not enabled for this bucket.\n");
        }
    }
    else
    {
        printf("get bucket logging faied(%s).\n", obs_get_status_name(ret_status));
    }
    // 销毁日志配置结构体
    destroy_logging_message(&logging_message);
}
```

15.4 关闭桶日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。

关闭桶日志功能实际上就是调用set_bucket_logging_configuration_ob将日志配置清空，以下代码展示了如何关闭桶日志：

```
static void test_close_bucket_logging_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 关闭桶日志
    set_bucket_logging_configuration_obs(&option, NULL, NULL, NULL,
        NULL, &response_handler, &ret_status);
    if (ret_status == OBS_STATUS_OK) {
        printf("close bucket(%s) logging successfully. \n",bucket_name_src);
    }
    else
    {
        printf("close bucket logging failed(%s).\n",obs_get_status_name(ret_status));
    }
}
```

16 静态网站托管

16.1 静态网站托管简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以将静态网站文件上传至OBS的桶中作为对象，并对这些对象赋予公共读权限，然后将该桶配置成静态网站托管模式，以实现在OBS上托管静态网站的目的。第三方用户在访问您网站的时候，实际上是在访问OBS的桶中的对象。在使用静态网站托管功能时，OBS还支持配置请求重定向，通过重定向配置您可以将特定的请求或所有请求实施重定向。

更多关于静态网站托管的内容请参考**静态网站托管**。

16.2 网站文件托管

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可通过以下步骤实现网站文件托管：

步骤1 将网站文件上传至OBS的桶中，并设置对象MIME类型。

步骤2 设置对象访问权限为公共读。

步骤3 通过浏览器访问对象。

----结束

以下代码展示了如何实现网站文件托管：

```
static void test_put_object()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 初始化结构体put_properties，可以通过该结构体设置对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    // 设置MIME类型
    put_properties.content_type = "text/html";
    // 设置对象访问权限为公共读
    put_properties.canned_acl = OBS_CANNED_ACL_PUBLIC_READ
    // 回调数据
    put_file_object_callback_data data;
    memset(&data, 0, sizeof(put_file_object_callback_data));
    // 将要上传的文件读到回调数据中
    data.infile = 0;
    data.noStatus = 1;
    content_length = read_bytes_from_file("<Uploaded filename>", &data);
    // 回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &response_complete_callback },
        &put_buffer_object_data_callback
     };
    // 上传数据流
    put_object(&option,"<object key>", content_length, &put_properties, 0, &putobjectHandler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object from file successfully. \n");
    }
    else
    {
        printf("put object failed(%s).\n",
               obs_get_status_name(data.ret_status));
    }
}
```

16.3 设置托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过set_bucket_website_configuration设置桶的托管配置。

配置默认主页错误页面和重定向规则

以下代码展示了如何配置默认主页、错误页面和重定向规则，参数描述如下表：

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
set_bucket_redirect_all	obs_set_bucket_redirect_all_conf *	必选	描述重定向的配置。
set_bucket_redirect_all->host_name	const char *	必选	描述重定向的站点名。
set_bucket_redirect_all->protocol	const char *	可选	描述重定向请求时使用的协议（http，https），默认使用http协议。
set_bucket_website_conf	obs_set_bucket_website_conf *	必选	描述重定向规则website元素的配置。
set_bucket_website_conf->suffix	const char *	必选	suffix元素被追加在对文件夹的请求的末尾（例如：suffix配置的是“index.html”，请求的是“samplebucket/images/”，返回的数据将是“samplebucket”桶内名为“images/index.html”的对象的内容）。suffix元素不能为空或者包含“/”字符。
set_bucket_website_conf->key	const char *	可选	当4XX错误出现时使用的对象的名称。这个元素指定了当错误出现时返回的页面。
set_bucket_website_conf->routingrule_info	bucket_website_routingrule *	可选	重定向规则的具体描述，请参看下表
set_bucket_website_conf->routingrule_count	int	可选	set_bucket_website_conf.routingrule_info的总大小
handler	obs_response_handler*	必选	回调函数。
callback_data	void *	可选	回调数据。

重定向规则结构bucket_website_routingrule描述如下表：

字段名	类型	约束	说明
key_prefix_equals	const char *	可选	描述当重定向生效时对象名的前缀。

字段名	类型	约束	说明
http_errorcode_returned_equals	const char *	可选	描述重定向生效时的HTTP错误码。当发生错误时，如果错误码等于这个值，那么重定向生效。
protocol	const char *	可选	描述重定向请求时使用的协议。
host_name	const char *	可选	描述重定向请求时使用的站点名。
replace_key_prefix_with	const char *	可选	描述重定向请求时使用的对象名前缀。
replace_key_with	const char *	可选	描述重定向请求时使用的对象名。
http_redirect_code	const char *	可选	描述响应中的HTTP状态码。

```
static void test_set_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    obs_set_bucket_website_conf set_bucket_website_conf;
    // 配置默认主页
    set_bucket_website_conf.suffix = "index.html";
    // 配置错误页面
    set_bucket_website_conf.key = "Error.html";
    // 定义重定向规则
    set_bucket_website_conf.routingrule_count = 2;
    bucket_website_routingrule temp[2];
    memset(&temp[0], 0, sizeof(bucket_website_routingrule));
    memset(&temp[1], 0, sizeof(bucket_website_routingrule));
    set_bucket_website_conf.routingrule_info = temp;
    temp[0].key_prefix_equals = "key_prefix1";
    temp[0].replace_key_prefix_with = "replace_key_prefix1";
    temp[0].http_errorcode_returned_equals="404";
    temp[0].http_redirect_code = NULL;
    temp[0].host_name = "www.example.com";
    temp[0].protocol = "http";
    temp[1].key_prefix_equals = "key_prefix2";
    temp[1].replace_key_prefix_with = "replace_key_prefix2";
    temp[1].http_errorcode_returned_equals="404";
    temp[1].http_redirect_code = NULL;
    temp[1].host_name = "www.example.com";
    temp[1].protocol = "http";
    // 设置响应回调函数
```

```
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    // 设置重定向规则
    set_bucket_website_configuration(&option, NULL, &set_bucket_website_conf,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket website conf successfully. \n");
    }
    else
    {
        printf("set bucket website conf failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

配置所有请求重定向

以下代码展示了如何配置所有请求重定向：

```
static void test_set_bucket_website_all()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0, &response_complete_callback
    };
    // 设置所有请求重定向
    obs_set_bucket_redirect_all_conf set_bucket_redirect_all;
    set_bucket_redirect_all.host_name = "www.example.com";
    set_bucket_redirect_all.protocol = "https";
    set_bucket_website_configuration(&option, &set_bucket_redirect_all, NULL,
        &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket website all successfully. \n");
    }
    else
    {
        printf("set bucket website all failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

16.4 查看托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_bucket_website_configuration查看桶的托管配置。以下代码展示了如何查看托管配置：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_get_bucket_websiteconf_handler *	必选	回调函数。
handler->response_handler	obs_response_handler	必选	回调函数。
handler->get_bucket_website_conf_callback	obs_get_bucket_websiteconf_callback *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_get_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_get_bucket_websiteconf_handler response_handler =
    {
        {&response_properties_callback, &response_complete_callback},
          &get_bucket_websiteconf_callback
    };
    // 获取桶的托管配置
    get_bucket_website_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("get bucket website successfully.\n");
    }
    else
    {
        printf("get bucket website failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

16.5 清除托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过delete_bucket_website_configuration清除桶的托管配置。以下代码展示了如何清除托管配置：

参数描述

字段名	类型	约束	说明
option	请求桶的上下文，**配置option**	必选	桶参数。
handler	obs_response_handler *	必选	回调函数。
callback_data	void *	可选	回调数据。

示例代码

```
static void test_delete_bucket_website_configuration()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
    };
    // 删除桶的托管配置
    delete_bucket_website_configuration(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("delete bucket website successfully.\n");
    }
    else
    {
        printf("delete bucket website failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

17 标签管理

17.1 标签简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

标签用于标识OBS中的桶，以此来达到对OBS中的桶进行分类的目的。

17.2 设置桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过set_bucket_tagging设置桶标签。

参数描述

字段名	类型	约束	说明
options	请求桶的上下文，**配置option**	必选	桶参数
tagging_list	obs_name_value *	必选	标签列表
number	unsigned int	必选	标签个数
handler	obs_response_handler *	必选	回调函数
callback_data	void *	可选	回调数据

标签列表结构obs_name_value描述如下表：

字段名	类型	说明
name	char *	标签键
value	char *	标签值

示例代码

```
static void test_set_bucket_tagging()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        0,
        &response_complete_callback
    };
    // 定义桶标签
    char tagKey[][OBS_COMMON_LEN_256] = {{"k1"},{"k2"},{"k3"},{"k4"},{"k5"},{"k6"},{"k7"},{"k8"},{"k9"},{"k10"}};
    char tagValue[][OBS_COMMON_LEN_256] = {{"v1"},{"v2"},{"v3"},{"v4"},{"v5"},{"v6"},{"v7"},{"v8"},{"v9"},{"v10"}};
    obs_name_value tagginglist[10] = {0};
    int i=0;
    for(;i<10;i++)
    {
        tagginglist[i].name = tagKey[i];
        tagginglist[i].value  = tagValue[i];
    }
    // 设置桶标签
    set_bucket_tagging(&option, tagginglist, 8, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("set bucket tagging successfully. \n");
    }
    else
    {
        printf("set bucket tagging failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```

说明

- 每个桶支持最多10个标签。
- 标签的Key和Value支持Unicode。

17.3 查看桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过get_bucket_tagging查看桶标签。

参数描述

字段名	类型	约束	说明
options	请求桶的上下文，**配置option**	必选	桶参数
handler	obs_get_bucket_tagging_handler *	必选	回调函数
callback_data	void *	可选	回调数据

回调函数类型obs_get_bucket_tagging_handler描述如下表：

字段名	类型	说明
response_handler	obs_response_handler	响应回调函数句柄
get_bucket_tagging_callback	obs_get_bucket_tagging_callback *	获取桶标签回调函数

示例代码

```
static void test_get_bucket_tagging()
{
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");

    // 设置响应回调函数
    obs_get_bucket_tagging_handler response_handler =
    {
        {&response_properties_callback, &get_bucket_tagging_complete_callback},
          &get_bucket_tagging_callback
    };
```

```
    // 创建回调数据
    TaggingInfo tagging_info;
    memset(&tagging_info, 0, sizeof(TaggingInfo));
    tagging_info.ret_status = OBS_STATUS_BUTT;
    // 获取桶标签
    get_bucket_tagging(&option, &response_handler, &tagging_info);
    if (OBS_STATUS_OK == tagging_info.ret_status) {
        printf("get bucket tagging successfully.\n");
    }
    else
    {
        printf("get bucket tagging failed(%s).\n", obs_get_status_name(tagging_info.ret_status));
    }
}
```

17.4 删除桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。

您可以通过delete_bucket_tagging删除桶标签。

参数描述

字段名	类型	约束	说明
options	请求桶的上下文，**配置option**	必选	桶参数
handler	obs_response_handler *	必选	回调函数
callback_data	void *	可选	回调数据

示例代码

```
static void test_delete_bucket_tagging()
{
    obs_status ret_status = OBS_STATUS_BUTT;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    // 设置响应回调函数
    obs_response_handler response_handler =
    {
        &response_properties_callback,
        &response_complete_callback
```

```
    };
    // 删除桶标签
    delete_bucket_tagging(&option, &response_handler, &ret_status);
    if (OBS_STATUS_OK == ret_status) {
        printf("delete bucket tagging successfully.\n");
    }
    else
    {
        printf("delete bucket tagging failed(%s).\n", obs_get_status_name(ret_status));
    }
}
```


18 服务端加密

18.1 服务端加密简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS支持服务端加密功能，使对象加密的行为在OBS服务端进行。

更多关于服务端加密的内容请参考**服务端加密**。

18.2 加密说明

须知

开发过程中，您有任何问题可以在github上**提交issue**。

OBS C SDK支持服务端加密的接口见下表：

OBS C SDK接口方法	描述	支持加密类型	传输协议
put_object	上传对象时设置加密算法、密钥，对对象启用服务端加密。	SSE-KMS SSE-C	HTTPS
get_object	下载对象时设置解密算法、密钥，用于解密对象。	SSE-KMS SSE-C	HTTPS

OBS C SDK接口方法	描述	支持加密类型	传输协议
copy_object	1. 复制对象时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制对象时设置目标对象的加密算法、密钥，对目标对象启用加密算法。	SSE-KMS SSE-C	当目标对象是非加密对象时：HTTP or HTTPS 其他情况都是HTTPS
get_object_metadata	获取对象元数据时设置解密算法、密钥，用于解密对象。	SSE-KMS SSE-C	当加密类型是SSE-KMS：HTTP or HTTPS 其余情况是HTTPS
initiate_multi_part_upload	初始化分段上传任务时设置加密算法、密钥，对分段上传任务最终生成的对象启用服务端加密。	SSE-KMS SSE-C	HTTPS
upload_part	上传段时设置加密算法、密钥，对分段数据启用服务端加密。	SSE-KMS SSE-C	HTTPS
complete_multi_part_upload	合并段时设置加密算法、密钥，对合并段后的数据启用服务端加密。	SSE-KMS SSE-C	HTTP or HTTPS
copy_part	1. 复制段时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制段时设置目标段的加密算法、密钥，对目标段启用加密算法。	SSE-KMS SSE-C	HTTPS

参数说明

加解密参数server_side_encryption_params说明如下表所示：

字段名	类型	说明
encryption_type	obs_encryption_type	服务端加密方式 OBS_ENCRYPTION_KMS：使用SE-KMS加密方式 OBS_ENCRYPTION_SSEC：SSE-C加密使用
kms_server_side_encryption	char *	使用该参数表示服务端加密是SSE-KMS方式。目标对象使用SSE-KMS方式加密。
kms_key_id	char *	SSE-KMS方式下使用该参数，表示加密目标对象使用的主密钥，如果用户没有提供该头域，那么默认的主密钥将会被使用。
ssec_customer_algorithm	char *	SSE-C方式下使用该参数，表示加密目标对象使用的算法。
ssec_customer_key	char *	SSE-C方式下使用该参数，表示加密目标对象使用的密钥。
des_ssec_customer_algorithm	char *	SSE-C方式下使用该参数，表示解密源对象使用的算法。
des_ssec_customer_key	char *	SSE-C方式下使用该参数，表示解密源对象使用的密钥。用于解密源对象。

18.3 加密示例

须知

开发过程中，您有任何问题可以在github上**提交issue**。

上传对象加密

以下代码展示了在上传对象时使用服务端加密功能：

```
static void test_put_object_by_aes_encrypt()
{
    // 待上传的buffer
    char *buffer = "11111111";
    // 待上传的buffer的长度
```

```
    int buffer_size = strlen(buffer);
    // 上传的对象名
    char *key = "put_buffer_aes";
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.protocol = OBS_PROTOCOL_HTTPS;
    // 初始化上传对象属性
    obs_put_properties put_properties;
    init_put_properties(&put_properties);
    //初始化存储上传数据的结构体
    put_buffer_object_callback_data data;
    memset(&data, 0, sizeof(put_buffer_object_callback_data));
    data.put_buffer = buffer;
    data.buffer_size = buffer_size;
    //服务端加密 SSE加密
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    encryption_params.ssec_customer_algorithm = "AES256";
    encryption_params.ssec_customer_key =
            "K7QkYpBkM5+hcs27fsNkUnNVaobncnLht/rCB2o/9Cw=";
    // 设置回调函数
    obs_put_object_handler putobjectHandler =
    {
        { &response_properties_callback, &put_buffer_complete_callback },
          &put_buffer_data_callback
    };
    put_object(&option, key, buffer_size, &put_properties,
               &encryption_params,&putobjectHandler,&data);

    if (OBS_STATUS_OK == data.ret_status) {
        printf("put object by_aes_encrypt successfully. \n");
    }
    else
    {
        printf("put object by_aes_encrypt encryption failed(%s).\n",
                obs_get_status_name(data.ret_status));
    }
}
```

下载对象解密

以下代码展示了在下载对象时使用服务端解密功能：

```
static void test_get_object_by_aes_encrypt()
{
    char *file_name = "./test_by_aes";
    char *key = "put_buffer_aes";
    obs_object_info object_info;
    // 创建并初始化option
    obs_options option;
    init_obs_options(&option);
    option.bucket_options.host_name = "<your-endpoint>";
    option.bucket_options.bucket_name = "<Your bucketname>";

    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ACCESS_KEY_ID和SECRET_ACCESS_KEY。
    // 您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
```

```
intl/zh-cn/usermanual-ca/ca_01_0003.html
    option.bucket_options.access_key = getenv("ACCESS_KEY_ID");
    option.bucket_options.secret_access_key = getenv("SECRET_ACCESS_KEY");
    option.bucket_options.protocol = OBS_PROTOCOL_HTTPS;

    // SSE加密的对象下载，需要传入SSE的密钥
    server_side_encryption_params encryption_params;
    memset(&encryption_params, 0, sizeof(server_side_encryption_params));
    encryption_params.use_ssec = '1';
    encryption_params.ssec_customer_algorithm = "AES256";
    encryption_params.ssec_customer_key = "K7QkYpBkM5+hcs27fsNkUnNVaobncnLht/rCB2o/9Cw=";

    memset(&object_info, 0, sizeof(obs_object_info));
    object_info.key =key;

    get_object_callback_data data;
    data.ret_status = OBS_STATUS_BUTT;
    data.outfile = write_to_file(file_name);

    obs_get_conditions getcondition;
    memset(&getcondition, 0, sizeof(obs_get_conditions));
    init_get_properties(&getcondition);

    obs_get_object_handler get_object_handler =
    {
        { NULL, &get_object_complete_callback},
        &get_object_data_callback
    };

    get_object(&option, &object_info, &getcondition, &encryption_params,
               &get_object_handler, &data);
    if (OBS_STATUS_OK == data.ret_status) {
        printf("get object by_aes successfully . \n");
    }
    else
    {
        printf("get object by_aes faied(%s).\n", obs_get_status_name(data.ret_status));
    }
    fclose(data.outfile);
}
```

19 异常处理

19.1 OBS 服务端错误码

在向OBS服务端发出请求后，如果遇到错误，会在响应中包含响应的错误码并描述错误信息。详细的错误码及其对应的描述和HTTP状态码**参见OBS API错误码**。。

19.2 SDK 错误处理

SDK错误返回包含： SDK检查函数参数返回的错误和OBS服务端返回的错误。

SDK错误处理信息：

- obs_status：错误码。
- obs_get_status_name()：获取错误描述。
- obs_status_is_retryable()：确认错误码是否需要业务重试。

19.3 日志分析

日志路径

OBS C SDK的日志路径是通过OBS.ini中LogPath字段指定的，默认存放于与C SDK动态库lib目录同级的logs目录中。定位问题只需要查看同级logs目录下运行日志eSDK-OBS-API-*-C.run.log或者obs-sdk-c.run.log。

OBS.ini文件应与动态库（libeSDKLogAPI.so）同一目录。

日志内容格式

SDK日志格式为：日志时间|日志级别|线程号|日志内容。示例如下：

```
运行日志：
2018-05-15 22:22:54 803| INFO|[140677572568864]|request_perform start
```


日志级别

当系统出现问题需要定位且当前的日志无法满足要求时，可以通过修改日志的级别来获取更多的信息。其中DEBUG(0)日志信息最丰富，ERROR(3)日志信息最少。

具体说明如下：

- DEBUG(0)：调试级别，如果设置为这个级别，除了打印INFO级别的信息外，还将打印其它帮助调试的信息等。
- INFO(1)：信息级别，如果设置为这个级别，除了打印WARN级别的信息外，还将打印OBS接口的调用过程和关键信息等。
- WARN(2)：告警级别，如果设置为这个级别，除了打印ERROR级别的信息外，还将打印一些关键事件的信息，如curl_global_init初始化失败等。
- ERROR(3)：错误级别，如果设置为这个级别，仅打印发生异常时的错误信息。

设置方式

修改lib目录下的OBS.ini，配置日志的大小、个数以及级别（其中*_Run参数的配置为最常用的配置项）：

```
;Every line must be less than 1024
[LogConfig]
;Log Size: unit=KB, 10MB = 10KB * 1024 = 10240KB
LogSize_Interface=10240
LogSize_Operation=10240
LogSize_Run=10240
;Log Num
LogNum_Interface=10
LogNum_Operation=10
LogNum_Run=10
;Log level: debug = 0,info = 1,warn = 2,error = 3
LogLevel_Interface=0
LogLevel_Operation=0
LogLevel_Run=0
;LogFilePermission
LogFilePermission=0600
[ProductConfig]
;Product Name
sdkname=eSDK-OBS-API-Linux-C
[LogPath]
;Log Path is relative to the path of configuration file
LogPath=../logs
```

其他日志相关配置

Windows下，OBS.ini中LogPath字段可以以wchar_t格式读取，需要通过设置文件路径编码来实现，案例如下：

set_file_path_code(UNICODE_CODE);//默认是ANSI_CODE

设置后，以下方法的本地文件路径也需要是wchar_t类型（参数是以char*格式传入，内部处理逻辑是wchar_t）

表 19-1

影响函数	说明
download_file	参数download_file_config的downLoad_file、check_point_file成员需要是wchar_t类型，以char*格式传入

影响函数	说明
upload_file	参数upload_file_config的upload_file、check_point_file成员需要是wchar_t类型，以char*格式传入
set_obs_log_path	参数log_path需要是wchar_t类型，以char*格式传入

19.4 签名不匹配异常

如果从请求响应中获取到HTTP状态码为403，OBS服务端错误码为OBS_STATUS_SignatureDoesNotMatch，请检查AK/SK是否有误。

19.5 HTTP 状态码报 405

调用接口失败，HTTP状态码为405时，应先确认该区域是否支持所调用的接口功能。

20 常见问题

20.1 常见编译问题

本章节收录了编译sdk或者sdk demo可能遇到的一些编译问题，对应版本的问题可能在其他旧版本中也出现，都可以参考

v3.23.3版本反映的常见问题：

问题1：

编译sdk的vs工程报错：

报错文件：目录${yourSDKPath}\platform\eSDK_LogAPI_V2.1.10\C\include下面的eSDKLogAPI.h、eSDKLogDataType.h这两个文件

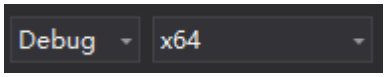
vs编译器错误码：c2018

解决方案：

修改上述目录下的两个文件（eSDKLogAPI.h、eSDKLogDataType.h）的行尾序列为CRLF

问题2：sdk的vs工程编译成功后还需要手动拷贝lib、dll到demo工程目录下才能运行demo程序，不够方便

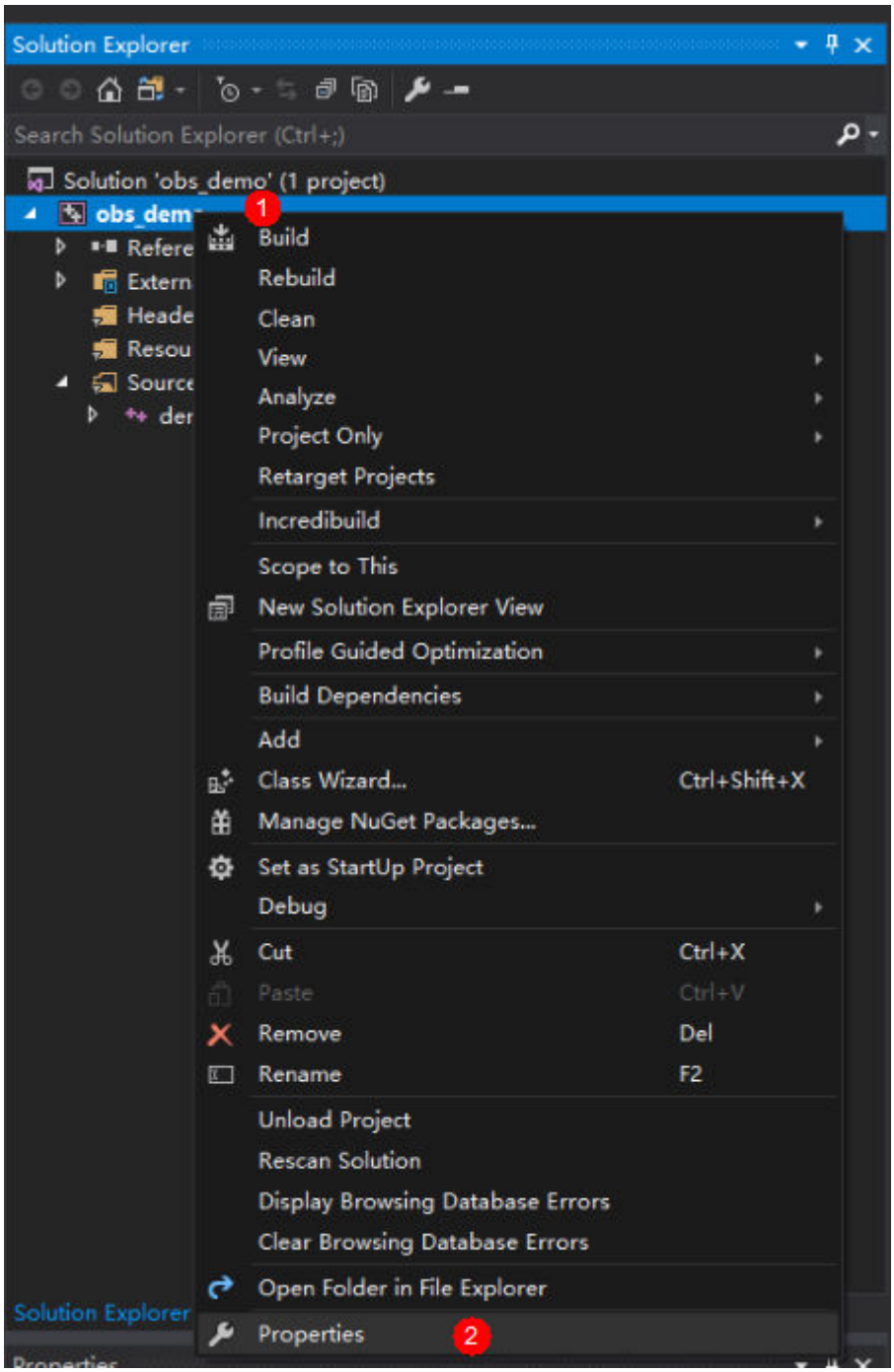
解决方案（以Debug，x64为例）：

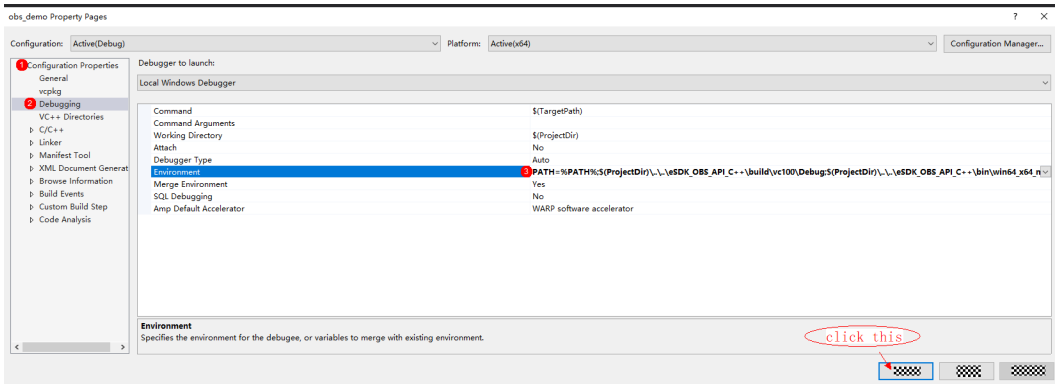


1、编译Debug，x64的obs.sln解决方案

2、打开obs_demo.sln解决方案，调整属性为Debug，x64，按图操作后，填入值：

PATH=%PATH%;$(ProjectDir)..\..\eSDK_OBS_API_C++\build\vc100\Debug;$(ProjectDir)..\..\eSDK_OBS_API_C++\bin\win64_x64_msvc\release;$(ProjectDir)..\..\..\..\platform\eSDK_LogAPI_V2.1.10\C\release_x64;$(LocalDebuggerEnvironment)





v3.22.7版本反映的Sdk本体常见编译问题

问题1：

```
-- Configuring incomplete, errors occurred!
make: *** No targets specified and no makefile found.  Stop.
cp: cannot stat 'cmake-build/cmake/lib/*.so': No such file or directory
cp: cannot stat './../../../platform/huaweisecurec/lib/linux/libsecurec.so': No such file or directory
demo/
```

解决方案：通过下面的脚本去编译securec组件，并将产物拷贝到对应目录

export Your_SDK_path='Your_SDK_path'

cd ${Your_SDK_path}/platform/huaweisecurec/src/

make

mkdir ${Your_SDK_path}/platform/huaweisecurec/lib/linux

cp ${Your_SDK_path}/platform/huaweisecurec/lib/libsecurec.so ${Your_SDK_path}/platform/securec/lib/linux

cd ${Your_SDK_path}/source/eSDK_OBS_API/eSDK_OBS_API_C++

#需要先cd到目录下，因为脚本按相对路径查找依赖项

sh build.sh

问题2：

```
efinition of `g_mutexThreadCheckpoint_download'
ned here
le_is_valid':
ommon.c:67: multiple definition of `g_mutexThreadCheckpoint'
ang/huaweicloud-sdk-c-obs-3.22.7/source/eSDK_OBS_API/eSDK_OBS_API_C+
```

解决方案：

在source\eSDK_OBS_API\eSDK_OBS_API_C++\src\object\download_file.c

中删除26、27行（如下）

pthread_mutex_t g_mutexThreadCheckpoint;

pthread_mutex_t g_mutexThreadCheckpoint_download;

在source\eSDK_OBS_API\eSDK_OBS_API_C++\src\object\object_common.c

中删除31、32行（如下）

pthread_mutex_t g_mutexThreadCheckpoint;

pthread_mutex_t g_mutexThreadCheckpoint_download;

之后编译即可

问题3：

```
cp: cannot stat 'inc/eSDKOBS.h': No such file or directory
cp: cannot stat '../../../../platform/huaweisecurec/include/*': No such file or directory
cp: cannot stat '../../../../platform/huaweisecurec/lib/linux/libsecurec.so': No such file or directory
cp: cannot stat '../../../../platform/eSDK_LogAPI_V2.1.10/C/linux_64/libeSDKLogAPI.so': No such file or directory
cp: cannot stat '../../../../platform/eSDK_LogAPI_V2.1.10/C/linux_64/liblog4cpp*': No such file or directory
cp: cannot stat '../../../../build/script/Provider/build/linux/curl-7.78.0/lib/*': No such file or directory
cp: cannot stat '../../../../build/script/Provider/build/linux/libxml2-2.9.9/lib/*': No such file or directory
cp: cannot stat '../../../../build/script/Provider/build/linux/openssl-1.1.1k/lib/*': No such file or directory
cp: cannot stat '../../../../build/script/Provider/build/linux/pcre-8.45/lib/*': No such file or directory
cp: cannot stat '../../../../build/script/Provider/build/linux/iconv-1.15/lib/*': No such file or directory
cp: cannot stat 'Makefile_obs': No such file or directory
cp: cannot stat 'OBS.ini': No such file or directory
cp: cannot stat '../../../../source/eSDK_OBS_API/eSDK_OBS_API_C++_Demo/object_test.c': No such file or directory
cp: cannot stat '../../../../source/eSDK_OBS_API/eSDK_OBS_API_C++_Demo/demo.c': No such file or directory
cp: cannot stat '../../../../source/eSDK_OBS_API/eSDK_OBS_API_C++_Demo/demo_common.c': No such file or directory
cp: cannot stat '../../../../source/eSDK_OBS_API/eSDK_OBS_API_C++_Demo/demo_common.h': No such file or directory
cp: cannot stat 'cert/client.pem': No such file or directory
cp: cannot stat 'cert/client.pem': No such file or directory
```

这个问题一般是在其他路径中执行build.sh导致的，需要先cd到对应目录下再执行，因为脚本按相对路径查找依赖项，执行如下脚本编译

export Your_SDK_path='Your_SDK_path'

cd ${Your_SDK_path}/source/eSDK_OBS_API/eSDK_OBS_API_C++

sh build.sh

v3.22.7版本反映的常见Sdk Demo编译问题

问题1：

```
object_test.c:35:10: fatal error: cJSON.h: No such file or directory
   35 | #include "cJSON.h"
      |          ^~~~~~~~~
compilation terminated.
make: *** [object_test] Error 1
make: *** Waiting for unfinished jobs....
demo.c:35:10: fatal error: cJSON.h: No such file or directory
   35 | #include "cJSON.h"
      |          ^~~~~~~~~
```

需要拷贝cjson头文件以及相关库(如果是arm需要把路径中的linux替换为arm)

同时需要改Makefile

export Your_SDK_path='Your_SDK_path'

export Your_DEMO_path='Your_DEMO_path'

cp ${Your_SDK_path}/build/script/Provider/build/linux/cjson-1.7.15/include/cJSON.h ${Your_DEMO_path}/include

cp ${Your_SDK_path}/build/script/Provider/build/linux/cjson-1.7.15/lib/libcjson.so* ${Your_DEMO_path}/lib

打开${Your_DEMO_path}/demo/Makefile

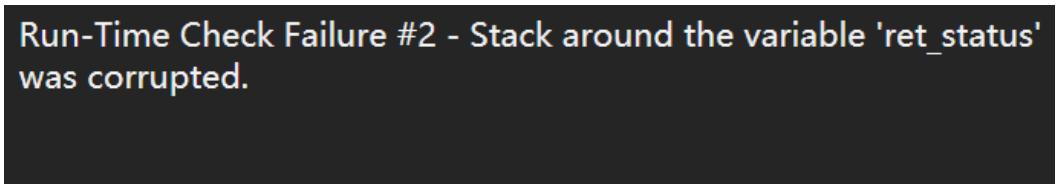
将16行（LIB=-leSDKOBS -lsecurec）替换为LIB=-lcjson -leSDKOBS -lsecurec

cd ${Your_DEMO_path}/demo

make

20.2 代理设置失效

1、sdk Windows端 demo中设置代理时出现如下问题，程序报错且代理设置失败



问题根因：某些sdk版本demo头文件eSDKOBS.h与sdk的eSDKOBS.h未同步更新，导致option中设置的代理失效

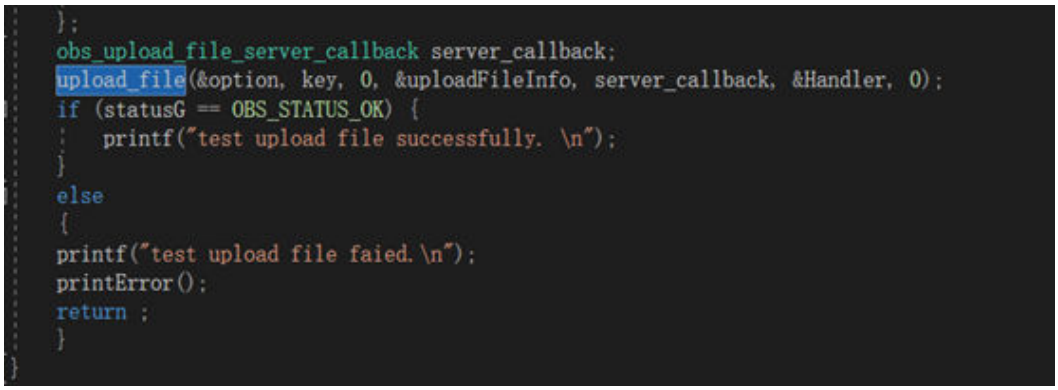
解决方法：将yourSDKpath\source\eSDK_OBS_API\eSDK_OBS_API_C++\inc\eSDKOBS.h

替换掉yourSDKpath\source\eSDK_OBS_API\eSDK_OBS_API_C++\build\obs\demo\eSDKOBS.h

同时demo做如下改动来适配eSDKOBS.h的更改（适配过程以3.22.7版本为例，其他版本可能略有不同）

在文件yourSDKpath\source\eSDK_OBS_API\eSDK_OBS_API_C++\build\obs\demo\demo_windows.cpp中4749行新增obs_upload_file_server_callback server_callback;

同时4750行中，函数upload_file第四个参数后增加, server_callback，如下图



2、设置了代理还是连接失败，可能是因为sdk的 request.c的 get_api_version函数中未设置代理

可以参考sdk的 request.c的 setup_curl函数中设置代理的方式在get_api_version函数中添加向curl中设置代理（CURLOPT_PROXY项与CURLOPT_PROXYUSERPWD项）的逻辑进行修复

A 修订记录

发布日期	修订记录
2024-04-29	第三次正式发布。 新增章节： • **自定义域名相关说明** • **c sdk通过自定义域名访问obs**
2023-11-01	第二次正式发布。 刷新章节： • **OBS服务环境搭建** • **配置SDK日志** • **断点续传下载** • **日志分析**
2020-03-31	第一次正式发布。