# S3 C++ sdk 文档

# 1. 前言

## 1.1. 简介

对象存储 S3 接口 C++ SDK 采用了开源的 aws-sdk-cpp。
本文档主要介绍 SDK 的安装、使用与注意事项。
假设您已经开通了对象存储服务， 并创建了 ak 与 sk。

# 2. 安装

## 2.1. 在Linux中安装aws-sdk-cpp

环境依赖：
Debian/Ubuntu-based systems : sudo apt-get install libcurl4-openssl-dev libssl-dev uuid-dev zlib1g-dev libpulse-dev cmake
Redhat/Fedora-based systems : sudo dnf install libcurl-devel openssl-devel libuuid-devel pulseaudio-devel cmake

获取SDK源码：
git clone --recurse-submodules https://github.com/aws/aws-sdk-cpp.git
PS：如果在Linux上使用 git 无法克隆下来（网络原因很容易失败），可以在windows上安装git，然后挂上vpn进行clone，再拷贝至Linux环境中。

编译：
进到aws-sdk-cpp源码的同级目录。

```
mkdir sdk_build
cd sdk_build
cmake ../aws-sdk-cpp -DCMAKE_BUILD_TYPE=Debug -DCMAKE_PREFIX_PATH=/usr/local/ -DCMAKE_INSTALL_PREFIX=/usr/local/ -DBUILD_SHARED_LIBS=on -DBUILD_ONLY="s3" -DENABLE_TESTING=OFF
make
make install
```

遇到问题：

1. 编写完测试程序进行编译时会提示“undefined reference to xxx”和“error: ld returned 1 exit status”，原因是编译时需要链接aws库，如“g++ main.cpp -o main -laws-cpp-sdk-s3 -laws-cpp-sdk-core”。
2. 运行测试程序会提示 aws-cpp-sdk-s3 库找不到，原因是库安装至 /usr/local/ 目录，系统默认不包含该目录，通过以下方式解决：

```
# echo "/usr/local/lib" >> /etc/ld.so.conf
# ldconfig
```

# 3. 初始化

请先阅读理解对象存储中的AccessKey， SecretKey， Endpoint相关的概念。

## 3.1. 确定EndPoint

EndPoint 是各个区域的地址，目前支持以区域域名地址的形式。进入控制台，在桶的属性中可以查找到当前桶所在的区域及域名，桶的域名的后缀部分为该桶的公网域名，例如:http://liulongzhen.61e78a07740e4eadb86e8dbe04e9ed06.oss-cnbj01.cdsgss.com中的oss-cnbj01.cdsgss.com 为该桶的公网EndPoint。

## 3.2. 配置密钥

要接入对象存储服务，您需要一对有效的AccessKey（包括AccessKeyId与AccessKeySecret）来进行签名验证。在获取到AccessKeyId与AccessKeySecret之后，可以按照以下的步骤进行初始化。

## 3.3. 新建S3Client

要使用 SDK 的功能，需要先初始化，代码如下所示：

```
    #include <aws/core/Aws.h>
    #include <aws/core/utils/logging/LogLevel.h>
    #include <aws/s3/S3Client.h>
    #include <aws/core/auth/AWSCredentialsProviderChain.h>
    #include <iostream>

    Aws::SDKOptions options;
    options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options);
    {
      Aws::Client::ClientConfiguration clientConfig;
      clientConfig.verifySSL = false;
      clientConfig.scheme = Aws::Http::Scheme::HTTP;
      clientConfig.endpointOverride = Aws::String("oss-cnbj01.cdsgss.com");

      Aws::String ak = "2d4345e5ffa4582eb981530527869d45";
      Aws::String sk = "84d8fd7ed4945464b4c966426ba0b3cb";

      Aws::S3::S3Client s3_client(Aws::Auth::AWSCredentials(ak, sk), clientConfig,
    Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never, false);

      ……
    }
```

## 3.4. 关闭S3Client

```
Aws::ShutdownAPI(options);
```

# 4. 管理 bucket

## 4.1. 创建 bucket

您可以使用接口创建 Bucket。如下代码展示如何新建一个 Bucket：

```
void createBucket(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName) {
    Aws::S3::Model::CreateBucketRequest request;
    request.SetBucket(bucketName);

    auto outcome = s3Client.CreateBucket(request);
    if (outcome.IsSuccess()) {
        std::cout << "createBucket success " << std::endl;
    }
    else {
        std::cout << "createBucket failed with error: " << outcome.GetError() <<
std::endl;
    }
}
```

注：
Bucket 的命名需要遵循对象存储中容器的命名规范。
Bucket 的名字是全局唯一的，所以您需要保证 Bucket 名称不与别人重复。

## 4.2. 列举 bucket

您可以使用接口列举指定用户下的所有 Bucket：

```
void listBuckets(const Aws::S3::S3Client& s3Client) {
    auto outcome = s3Client.ListBuckets();
    if (outcome.IsSuccess()) {
        std::cout << "Found " << outcome.GetResult().GetBuckets().size() << "
buckets\n";
        for (auto&& b : outcome.GetResult().GetBuckets()) {
            std::cout << b.GetName() << std::endl;
        }
    }
    else {
        std::cout << "Failed with error: " << outcome.GetError() << std::endl;
    }
}
```

## 4.3. 删除 Bucket

您可以使用接口删除 Bucket：

```
void deleteBucket(const Aws::S3::S3Client& s3Client, const Aws::String& bucketName) {
    Aws::S3::Model::DeleteBucketRequest request;
    request.SetBucket(bucketName);

    auto outcome = s3Client.DeleteBucket(request);
    if (outcome.IsSuccess()) {
        std::cout << "deleteBucket success " << std::endl;
    }
    else {
        std::cout << "deleteBucket failed with error: " << outcome.GetError() <<
std::endl;
    }
}
```

注：
如果容器不为空，则无法被删除，服务端会返回错误；
必选先删除容器中的所有对象后，容器才能成功删除。

# 5. 上传下载

## 5.1. 上传对象

在对象存储中用户的基本操作单元是对象，亦可以理解为文件， CPP SDK提供了丰富的上传接口，可以通过以下的方式上传文件:
流式上传(包含文件上传)；
分块上传。
流式上传、单块上传最大为100M；分块上传除最后一块外，分块不能小于16k，每一个分块不能大于100M，最多能上传10000个分块，即分块上传的文件最大支持1T。

### 5.1.1. 流式上传

流式上传使用S3Client.PutObject上传一个一个Stream中的内容，数据可以是二进制流或者本地文件。以下是上传文件的实例代码

```
void putObjectFromContent(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName,
    const Aws::String& objectName, const Aws::String& fileName){
    Aws::S3::Model::PutObjectRequest object_request;
    object_request.WithBucket(bucketName).WithKey(objectName);

    auto input_data = Aws::MakeShared<Aws::FStream>(objectName.c_str(),
fileName.c_str(), std::ios_base::in | std::ios_base::binary);

    object_request.SetBody(input_data);

    auto put_object_outcome = s3Client.PutObject(object_request);

    if (put_object_outcome.IsSuccess())
    {
        std::cout << "Upload Done!" << std::endl;
    }
    else
    {
        std::cout << "PutObject error: " <<
        put_object_outcome.GetError().GetExceptionName() << " " <<
        put_object_outcome.GetError().GetMessage() << std::endl;
    }
}
```

如果是直接上传字符串内容，只需要把上面的 input_data 改成如下方式初始化即可：
auto input_content = Aws::MakeShared<Aws::StringStream>("test","hello world");

### 5.1.2. 分片上传

除了通过putObject接口上传文件之外，还提供了另外一种上传模式-分块上传,用户可以在如下应用场景内（但不限于此），使用分块上传模式，如：
需支持断点上传；
上传超过100M的文件；
网络条件较差，经常和服务器断开连接；
上传文件之前无法确定文件的大小；

分块上传一般流程如下所示:
初始化一个分块上传任务(createMultipartRequest)；
上传分块(UploadPart)；
完成分块上传(CompleteMultipartUpload)或者取消分块上传(AbortMultipartUpload)；

```
Aws::String initMultipart(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName,
    const Aws::String& objectName) {
    Aws::S3::Model::CreateMultipartUploadRequest createMultipartRequest;
    createMultipartRequest.WithBucket(bucketName).WithKey(objectName);


    //1. init
    auto createMultiPartResult =
s3Client.CreateMultipartUpload(createMultipartRequest);
    if(createMultiPartResult.IsSuccess()){
        std::cout << "createMultipartUpload ok, uploadId = " <<
createMultiPartResult.GetResult().GetUploadId() << std::endl;
        return createMultiPartResult.GetResult().GetUploadId();
    } else {
        std::cout << "Create MultipartUpload failed" <<
createMultiPartResult.GetError().GetMessage() <<  std::endl;
        return "";
    }
}
```

```
Aws::Vector<Aws::S3::Model::CompletedPart> uploadParts(const Aws::S3::S3Client&
s3Client, const Aws::String& bucketName,
    const Aws::String& objectName, const Aws::String& uploadId, const Aws::String&
uploadFilePath){


    long partSize =  5 * 1024 * 1024;
    std::fstream file(uploadFilePath.c_str(),std::ios::in | std::ios::binary);
    file.seekg(0,std::ios::end);
    long   fileSize = file.tellg();
    std::cout << file.tellg() << std::endl;
    file.seekg(0, std::ios::beg);



    char* buffer = new char[partSize];

    long filePosition = 0;
    Aws::Vector<Aws::S3::Model::CompletedPart> completeParts;

    int partNumber = 1;
    while(filePosition < fileSize){
        partSize = std::min(partSize,(fileSize - filePosition));
        file.read(buffer,partSize);
        //std::cout << "readSize : " << partSize << std::endl;
        Aws::S3::Model::UploadPartRequest uploadPartRequest;

uploadPartRequest.WithBucket(bucketName).WithKey(objectName).WithUploadId(uploadId).W
ithPartNumber(partNumber).WithContentLength(partSize);


        Aws::String str(buffer,partSize);
        auto input_data = Aws::MakeShared<Aws::StringStream>("UploadPartStream",str);
        uploadPartRequest.SetBody(input_data);
        filePosition += partSize;


        auto uploadPartResult = s3Client.UploadPart(uploadPartRequest);
        //std::cout << uploadPartResult.GetResult().GetETag() << std::endl;

completeParts.push_back(Aws::S3::Model::CompletedPart().WithETag(uploadPartResult.Get
Result().GetETag()).WithPartNumber(partNumber));
        memset(buffer, 0, partSize);
        ++partNumber;
    }
    return completeParts;
}
```

```
void completeMultipart(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName,
    const Aws::String& objectName, const Aws::String& uploadId, const
Aws::Vector<Aws::S3::Model::CompletedPart>& completedParts){
    Aws::S3::Model::CompleteMultipartUploadRequest complete;

    complete.WithBucket(bucketName).WithKey(objectName).
    WithUploadId(uploadId).

WithMultipartUpload(Aws::S3::Model::CompletedMultipartUpload().WithParts(completedPar
ts));
    auto completeMultipartUploadResult = s3Client.CompleteMultipartUpload(complete);
    if( completeMultipartUploadResult.IsSuccess()){
        std::cout << "multipartUpload done" << std::endl;
    } else {
        Aws::S3::Model::AbortMultipartUploadRequest abortRequest;

abortRequest.WithBucket(bucketName).WithKey(objectName).WithUploadId(uploadId);
        s3Client.AbortMultipartUpload(abortRequest);
        std::cout << "multipartUpload failed" << std::endl;
    }
}
```

```
void multipartUpload(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName,
    const Aws::String& objectName, const Aws::String& uploadFilePath){
    //1. init
    const Aws::String uploadId = initMultipart(s3Client, bucketName, objectName);

    //2. uploadPart
    Aws::Vector<Aws::S3::Model::CompletedPart> completeParts = uploadParts(s3Client,
bucketName, objectName, uploadId, uploadFilePath);

    //3. complete multipart upload

    completeMultipart(s3Client,bucketName,objectName,uploadId,completeParts);
}
```

注意：
上面程序一共分为三个步骤：1. initiate 2. uploadPart 3. complete
UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于或等于16K。但是Upload Part接口并不会立即校验上传Part的大小（因为不知道是否为最后一块）；只有当Complete Multipart Upload的时候才会校验。
Part号码的范围是1~10000。如果超出这个范围，将返回InvalidArgument的错误码。
分块上传除最后一块外，分块不能小于16k，每一个分块不能大于100M。
每次上传Part时都要把流定位到此次上传块开头所对应的位置。
分片上传任务初始化或上传部分分片后，可以使用abortMultipartUpload接口中止分片上传事件。当分片上传事件被中止后，就不能再使用这个Upload ID做任何操作，已经上传的分片数据也会被删除。
每次上传Part之后，返回结果会包含一个 PartETag 对象，它是上传块的ETag与块编号（PartNumber）的组合。在后续完成分片上传的步骤中会用到它，因此我们需要将其保存起

来，然后在第三步complete的时候使用，具体操作参考上面代码。

### 5.1.3. 取消分片上传

您可以使用S3Client.AbortMultipartUpload取消上传事件，具体实现如下:

```
void abortMultipartUpload(const Aws::S3::S3Client& s3Client,const Aws::String&
bucketName,const Aws::String& objectName,const Aws::String& uploadId){
    Aws::S3::Model::AbortMultipartUploadRequest abortRequest;
    abortRequest.WithBucket(bucketName).WithKey(objectName).WithUploadId(uploadId);
    auto result = s3Client.AbortMultipartUpload(abortRequest);
    if (result.IsSuccess()) {
        std::cout << "abort succ" << std::endl;
    } else {
        std::cout << "abort failed" << std::endl;
    }
}
```

### 5.1.4. 查看已经上传的分片

查看已经上传的分片可以罗列出指定Upload ID(InitiateMultipartUpload时获取)所属的所有已经上传成功的分片，您可以通过S3Client.ListParts接口获取已经上传的分片，可以参考以下代码:

```
void ListParts(const Aws::S3::S3Client& s3Client,const Aws::String& bucketName,const Aws::String& objectName,const Aws::String& uploadId){
    Aws::S3::Model::ListPartsRequest listPartRequest;

listPartRequest.WithBucket(bucketName).WithKey(objectName).WithUploadId(uploadId).WithMaxParts(10).WithPartNumberMarker(10);
    auto result = s3Client.ListParts(listPartRequest);
    if (result.IsSuccess()) {
        for(auto part : result.GetResult().GetParts()){
            std::cout << part.GetPartNumber() << "-->" << part.GetETag() << std::endl;
        }
    } else {
        std::cout << "list part request faield" << std::endl;
    }
}
```

### 5.1.5. 查看当前正在进行的分片上传任务

查看正在进行的分片上传任务可以罗列出正在进行，还未完成的分片上传任务，您可以通过S3Client.ListMultipartUploads接口当前的上传任务，可以参考以下代码:

```
void listMultipartUpload(const Aws::S3::S3Client& s3Client,const Aws::String& bucketName){
    Aws::S3::Model::ListMultipartUploadsRequest listMultipartUploadRequest;
    listMultipartUploadRequest.WithBucket(bucketName);
    auto result = s3Client.ListMultipartUploads(listMultipartUploadRequest);
    if (result.IsSuccess()) {
        for(auto upload : result.GetResult().GetUploads()){
            std::cout << upload.GetKey() << std::endl;
        }
    } else {
        std::cout << "failed" << std::endl;
    }
}
```

## 5.2. 下载对象

SDK提供了丰富的文件下载接口，用户可以通过以下方式获取文件:
下载文件到内存
下载到本地文件
分段下载
条件下载

### 5.2.1. 下载文件到本地文件

以下源代码实现将文件下载到本地文件:

```
void downloadObject(const Aws::S3::S3Client& s3Client, const Aws::String& bucketName,
    const Aws::String& objectName, const Aws::String& saveName){
    Aws::S3::Model::GetObjectRequest object_request;
    object_request.WithBucket(bucketName).WithKey(objectName);

    auto get_object_outcome = s3Client.GetObject(object_request);

    if (get_object_outcome.IsSuccess())
    {
        Aws::OFStream local_file;
        local_file.open(saveName.c_str(), std::ios::out | std::ios::binary);
        local_file << get_object_outcome.GetResult().GetBody().rdbuf();
        std::cout << "Download Done!" << std::endl;
    }
    else
    {
        std::cout << "GetObject error: " <<
        get_object_outcome.GetError().GetExceptionName() << " " <<
        get_object_outcome.GetError().GetMessage() << std::endl;
    }

}
```

# 6. 管理 object

用户可以通过一系列的接口管理桶(Bucket)中的文件(Object)，比如ListObjects，DeleteObject，CopyObject，DoesObjectExist等。

## 6.1. 判断对象是否存在

```
bool doesObjectExits(const Aws::S3::S3Client& s3Client, const Aws::String&
bucketName, const Aws::String& objectName){

    Aws::S3::Model::HeadObjectRequest headObjectRequest;

    headObjectRequest.WithBucket(bucketName).WithKey(objectName);

    auto result = s3Client.HeadObject(headObjectRequest);

    if (result.IsSuccess()) {

        std::cout << "Object exits" << std::endl;

    } else {

        std::cout << "Object doesn't exits" << std::endl;

    }

    return result.IsSuccess();

}
```

## 6.2. 列出 bucket 中对象

您可以使用S3Client.ListObjects列出bucket中的文件，以下代码列出了bucket中的文件以及每个文件的大小：

```
void listObjectsWithPrintKeySize(const Aws::S3::S3Client& s3Client, const
Aws::String& bucketName) {
    Aws::S3::Model::ListObjectsRequest listObjectRequest;
    listObjectRequest.WithBucket(bucketName);
    auto result = s3Client.ListObjects(listObjectRequest);
    if (result.IsSuccess()) {
        for (auto content : result.GetResult().GetContents()) {
            std::cout << content.GetKey() << " --> " << content.GetSize() <<
std::endl;
        }
    }
    else {
        std::cout << "list Objects failed" << std::endl;
    }
}
```

上述代码中的ListObjectsRequest中的可选参数如下所示：

| 参数 | 含义 |
| --- | --- |

| | |
|---|---|
| Delimiter | 用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现delimiter字符之间的object作为一组元素。限制长度为10。 |
| Prefix | 限定返回的object key必须以prefix作为前缀。注意使用prefix查询时，返回的key中仍会包含prefix |
| MaxKeys | 限定此次返回object的最大数，如果不设定，默认为100 |
| Marker | 设定结果从marker之后按字母排序的第一个开始返回 |

## 6.3. 删除单个对象

您可以使用S3Client.DeleteObject删除单个需要删除的文件:

```
void deleteObject(const Aws::S3::S3Client& s3Client, const Aws::String& bucketName,
const Aws::String& objectName){


    Aws::S3::Model::DeleteObjectRequest object_request;
    object_request.WithBucket(bucketName).WithKey(objectName);


    auto delete_object_outcome = s3Client.DeleteObject(object_request);


    if (delete_object_outcome.IsSuccess())
    {
        std::cout << "Delete Done!" << std::endl;
    }
    else
    {
        std::cout << "DeleteObject error: " <<
        delete_object_outcome.GetError().GetExceptionName() << " " <<
        delete_object_outcome.GetError().GetMessage() << std::endl;
    }
}
```

## 6.4. 删除多个对象

您可以使用S3Client.DeleteObjects批量删除文件:

```
void deleteObjects(const Aws::S3::S3Client& s3Client,const Aws::String&
bucketName,const Aws::Vector<Aws::S3::Model::ObjectIdentifier>& objectNames){
    Aws::S3::Model::DeleteObjectsRequest deleteObjectsRequest;
    Aws::S3::Model::Delete deleteObjects;
    deleteObjects.SetObjects(objectNames);
    deleteObjects.SetQuiet(false);
    deleteObjectsRequest.WithBucket(bucketName).WithDelete(deleteObjects);
    auto result = s3Client.DeleteObjects(deleteObjectsRequest);
    if (result.IsSuccess()) {
        for(auto err : result.GetResult().GetErrors()){
            std::cout << "delete err : " << err.GetKey() << " --> " <<
err.GetMessage() << std::endl;
        }
        for (auto del : result.GetResult().GetDeleted()) {
            std::cout << "delete ok : " << del.GetKey()  << std::endl;
        }
        std::cout << "ok" << std::endl;
    } else {
        std::cout << result.GetError().GetMessage() << std::endl;
        std::cout << "failed" << std::endl;
    }
}
```

## 6.5. 拷贝文件

您可以使用S3Client.GetObjectMetadata获取对象的元数据信息:

```
void copyObject(const Aws::S3::S3Client& s3Client, const Aws::String& srcBucket,
    const Aws::String& srcObject, const Aws::String& dstBucket, const Aws::String&
dstObject){
    Aws::S3::Model::CopyObjectRequest copyObjectRequest;
    copyObjectRequest.WithBucket(dstBucket).WithKey(dstObject).WithCopySource("/" +
srcBucket + "/" + srcObject);
    auto result = s3Client.CopyObject(copyObjectRequest);
    if (result.IsSuccess()) {
        std::cout << "copy done" << std::endl;
    } else {
        std::cout << "copy failed" << std::endl;
    }
}
```

注意：支持跨桶的文件copy

## 6.6. 获取文件的文件元信息

您可以使用S3Client.GetObjectMetadata获取对象的元数据信息:

```
void getObjectMeta(const Aws::S3::S3Client& s3Client, const Aws::String& bucketName,
const Aws::String& objectName){
    Aws::S3::Model::HeadObjectRequest headObjectRequest;
    headObjectRequest.WithBucket(bucketName).WithKey(objectName);
    auto result = s3Client.HeadObject(headObjectRequest);
    if (result.IsSuccess()) {
        std::cout << "getObjectMeta test" << std::endl;
        for(auto entry : result.GetResult().GetMetadata()){
            std::cout << entry.first << " -- > " << entry.second << std::endl;
        }
    } else {
        std::cout << "failed" << std::endl;
    }
}
```

## 6.7. 生成共享外链

您可以使用S3Client.GetPreSignedURL获取对象的下载URL:,使用浏览器访问该链接即可下载该对象。您可以将该链接共享给其他人，从而达到共享该文件的目的。

```
void getObjectDonwloadUrl(Aws::S3::S3Client& s3Client, const Aws::String& bucketName,
const Aws::String& objectName){
    Aws::String url = s3Client.GeneratePresignedUrl(bucketName, objectName,
Aws::Http::HttpMethod::HTTP_GET, 20);
    std::cout << "url : " << url << std::endl;
}
```

注意：
带有预签名的 URL 链接，都带有过期时间。
如果您希望链接不过期，可以将过期时间设置的很大，或将对象设置成公共可访问（访问时不需要签名）。