



天翼云媒体存储

C++ SDK 使用指导书

天翼云科技有限公司

2026年3月27日

天翼云媒体存储C++ SDK使用指导书

SDK安装

开发环境

- GNU编译器集合 (GCC) 4.9或更高版本
- Clang 3.3或更高版本
- Microsoft Visual Studio (Windows) 或C++ 11编译器 (Linux / macOS)
- CMake 3.2或更高版本

安装依赖库

您必须安装如下的依赖库 libcurl, libopenssl, libuuid, zlib 和 libpulse

在基于 *Debian / Ubuntu* 的系统上安装软件包

```
sudo apt-get install libcurl4-openssl-dev libssl-dev uuid-dev zlib1g-dev libpulse-dev
```

在基于 *Redhat / Fedora / CentOS* 的系统上安装软件包

```
sudo yum install libcurl-devel openssl-devel libuuid-devel pulseaudio-libs-devel
```

使用源代码构建SDK

1 下载源代码

下载xos-cpp-sdk, 下载链接为 [xos-cpp-sdk.zip](#)

2 解压源代码并进入源代码目录

```
unzip xos-cpp-sdk.zip
cd xos-cpp-sdk
```

3 创建build目录, 在build目录中运行 cmake 生成Makefile, 可以指定Debug版本和 Release版本

```
mkdir <install/prefix/path> #创建目录存放编译好的sdk头文件和库文件
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=<install/prefix/path>
```

4 编译源码: 仅使用s3 sdk的情况下只需要编译aws-cpp-sdk-core和aws-cpp-sdk-s3

```
make -C aws-cpp-sdk-core
make -C aws-cpp-sdk-s3
make -C aws-cpp-sdk-sts #使用sts服务需要构建
make -C aws-cpp-sdk-transfer #封装分片上传接口
```

5 执行安装操作，sdk头文件和库文件将被生成到步骤3创建的目录中

```
make install -C aws-cpp-sdk-core
make install -C aws-cpp-sdk-s3
make install -C aws-cpp-sdk-sts
make install -C aws-cpp-sdk-transfer
```

初始化

使用说明

媒体存储的 C++ 开发工具包必须通过调用 `Aws::InitAPI` 进行初始化操作。在应用程序终止之前，必须通过调用 `Aws::ShutdownAPI` 关闭 SDK。每个方法都接受 `Aws::SDKOptions` 的参数。可以在这两个方法调用之间执行对 SDK 的所有其他调用。所有的接口调用应该在 `Aws::InitAPI` 和 `Aws::ShutdownAPI` 之间被执行。使用媒体存储的 C++ 开发工具包的所有应用程序都必须包含该文件 `aws/core/Aws.h`。

代码示例

```
#include <aws/core/Aws.h>
int main(int argc, char** argv)
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        // make your SDK calls here.
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

初始化 `Aws::S3::S3Client`，`S3Client` 是 SDK 的入口。

```
// S3Demo.h
#pragma once

#include <memory>

namespace Aws {
    namespace S3 {
        class S3Client;
    }
}

class S3Demo
{
public:
    S3Demo();
    ~S3Demo();

private:
    void init();

private:
```

```
std::shared_ptr<Aws::S3::S3Client> s3_client;
};
```

```
// S3Demo.cpp
#include <aws/core/auth/AWSCredentials.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>

#define OSS_ACCESS_KEY "<your-access-key>"
#define OSS_SECRET_KEY "<your-secret-key>"
#define OSS_ENDPOINT "<your-endpoint>" // e.g.
http://xxoss.xstore.ctyun.cn
#define OSS_BUCKET_NAME "<your-bucket-name>"

S3Demo::S3Demo()
{
    init();
}

S3Demo::~S3Demo()
{
}

void S3Demo::init()
{
    Aws::String ak = OSS_ACCESS_KEY;
    Aws::String sk = OSS_SECRET_KEY;
    Aws::String endPoint = OSS_ENDPOINT;

    Aws::Auth::AWSCredentials cred(ak, sk);
    Aws::Client::ClientConfiguration cfg;
    cfg.endpointOverride = endPoint;
    cfg.scheme = Aws::Http::Scheme::HTTP;
    cfg.verifySSL = false;
    s3_client = std::make_shared<Aws::S3::S3Client>(cred, cfg,
        Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never, false);
}
```

桶相关接口

创建桶

功能说明

您可以使用CreateBucket方法创建存储桶。

代码示例

```
bool S3Demo::CreateBucket()
{
    const Aws::String bucket_name = "<your-bucket-name>";
```

```

    Aws::S3::Model::CreateBucketRequest request;
    request.SetBucket(bucket_name);
    Aws::S3::Model::CreateBucketOutcome outcome = s3_client-
>CreateBucket(request);
    if (outcome.IsSuccess()) {
        std::cout << "CreateBucket " << bucket_name << " success";
        return true;
    }
    else {
        std::cout << "Error: CreateBucket: " << outcome.GetError().GetMessage() <<
std::endl;
        return false;
    }
}

```

请求参数

| 参数名 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| bucket | string | 桶名称 | 是 |

获取桶列表

功能说明

您可以使用ListBuckets接口获取桶列表。

代码示例

```

bool S3Demo::ListBuckets()
{
    Aws::S3::Model::ListBucketsOutcome outcome = s3_client->ListBuckets();
    if (outcome.IsSuccess()) {
        std::cout << "Bucket names:" << std::endl << std::endl;

        Aws::Vector<Aws::S3::Model::Bucket> buckets =
            outcome.GetResult().GetBuckets();

        for (Aws::S3::Model::Bucket& bucket : buckets) {
            std::cout << bucket.GetName() << std::endl;
        }
        return true;
    }
    else {
        std::cout << "Error: ListBuckets: " << outcome.GetError().GetMessage() <<
std::endl;
        return false;
    }
}

```

请求参数

无

返回结果

| 参数 | 类型 | 说明 |
|---------|----------|-----|
| Buckets | Bucket数组 | 桶列表 |

判断桶是否存在

功能说明

您可以使用HeadBucket接口判断桶是否存在。

代码示例

```
bool S3Demo::HeadBucket()
{
    Aws::S3::Model::HeadBucketRequest request;
    request.SetBucket("<your-bucket-name>");
    Aws::S3::Model::HeadBucketOutcome outcome = s3_client->HeadBucket(request);
    if (outcome.IsSuccess()) {
        std::cout << "HeadBucket success";
        return true;
    } else {
        std::cout << "Error: HeadBucket: " << outcome.GetError().GetMessage() <<
        std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

删除桶

功能说明

您可以使用DeleteBucket删除存储桶。

代码示例

```
bool S3Demo::DeleteBucket()
{
    const Aws::String bucket_name = "<your-bucket-name>";

    Aws::S3::Model::DeleteBucketRequest request;
    request.SetBucket(bucket_name);
    Aws::S3::Model::DeleteBucketOutcome outcome = s3_client-
    >DeleteBucket(request);
    if (outcome.IsSuccess()) {
```

```

std::cout << "DeleteBucket " << bucket_name << " success";
return true;
}
else {
std::cout << "Error: DeleteBucket: " << outcome.GetError().GetMessage() <<
std::endl;
return false;
}
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

注意：在删除桶前，必须先确保桶为空，否则会出现如下错误：

BucketNotEmpty - Unable to parse ExceptionName: BucketNotEmpty Message

设置桶访问权限

功能说明

媒体存储支持一组预先定义的授权，称为Canned ACL。每个Canned ACL都有一组预定义的被授权者和权限，下表列出了相关的预定义授权含义。

| ACL | 权限 | 描述 |
|--------------------|--------|-----------------------------|
| private | 私有读写 | 存储桶拥有者有读写权限，其他用户没有访问权限。 |
| public-read | 公共读私有写 | 存储桶拥有者有读写权限，其他用户只有该存储桶的读权限。 |
| public-read-write | 公共读写 | 所有用户都有该存储桶的读写权限。 |
| authenticated-read | 注册用户可读 | 存储桶拥有者有读写权限，注册用户具有该存储桶的读权限。 |

您可以通过PutBucketAcl接口设置一个存储桶的访问权限。用户在设置存储桶的ACL之前需要具备WRITE_ACP 权限。

代码示例

```

bool S3Demo::PutBucketAcl()
{
    Aws::S3::Model::PutBucketAclRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetACL(Aws::S3::Model::BucketCannedACL::private_);

    Aws::S3::Model::PutBucketAclOutcome outcome = s3_client-
>PutBucketAcl(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutBucketAcl success";
        return true;
    }
}

```

```

    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutBucketAcl: " << (int)err.GetResponseCode() << ",
Message:" <<
            err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|-----------------|------|------|
| Bucket | string | 桶名称 | 是 |
| ACL | BucketCannedACL | acl值 | 是 |

获取桶访问权限

功能说明

您可以通过GetBucketAcl接口获取存储桶的access control list (ACL) 信息。存储桶的ACL可以在创建的时候设置并且通过API查看，用户需要具有READ_ACP（读取存储桶 ACL信息）权限才可以查询存储桶的ACL信息。

代码示例

```

bool S3Demo::GetBucketAcl()
{
    Aws::S3::Model::GetBucketAclRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::GetBucketAclOutcome outcome = s3_client-
>GetBucketAcl(request);
    if (outcome.IsSuccess()) {

        Aws::Vector<Aws::S3::Model::Grant> grants = outcome.GetResult().GetGrants();
        for (Aws::S3::Model::Grant& grant : grants)
        {
            std::cout << "Grant:" << grant.GetGrantee().GetDisplayName() << ",
permission:" << (int)grant.GetPermission() << std::endl;
        }
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: GetBucketAcl: " << (int)err.GetResponseCode() << ",
Message:" <<
            err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|--------|--------|---------------|
| Owner | Owner | 所有者信息 |
| Grants | Grants | 每种类型用户的详细权限信息 |

设置桶策略

功能说明

存储桶授权策略 (bucket policy) 可以灵活地配置用户各种操作和访问资源的权限。访问控制列表 (access control lists, ACL) 只能对单一对象设置权限, 而存储桶授权策略可以基于各种条件对一个桶内的全部或者一组对象配置权限。桶的拥有者拥有PutBucketPolicy操作的权限, 如果桶已经被设置了policy, 则新的policy会覆盖原有的policy。您可以通过PutBucketPolicy接口设置桶策略, 描述桶策略的信息以JSON格式的字符串形式通过Policy参数传入。一个policy的示例如下:

```
{
  "Id": "<your-policy-id>",
  "Version": "2012-10-17",
  "Statement" : [{
    "Sid": "<your-statement-id>",
    "Principal": {
      "AWS": ["arn:aws:iam::user/<your-user-name>"]
    },
    "Effect": "Allow",
    "Action": [
      "s3:ListBucket",
      "s3:CreateBucket"
    ],
    "Resource": [
      "arn:aws:iam::<your-bucket-name>/*"
    ],
    "Condition": "<some-conditions>"
  }]
}
```

Statement的内容说明如下:

| 元素 | 描述 | 是否必要 |
|--------------|---|-------------------------------|
| Sid | statement Id, 可选关键字, 描述statement的字符串 | 否 |
| Principal | 可选关键字, 被授权人, 指定本条statement权限针对的Domain以及User, 支持通配符"*", 表示所有用户(匿名用户)。当对Domain下所有用户授权时, Principal格式为arn:aws:iam:::user/*。当对某个User进行授权时, Principal格式为arn:aws:iam:::user/<your-user-name> | 可选, Principal与NotPrincipal选其一 |
| NotPrincipal | 可选关键字, 不被授权人, statement匹配除此之外的其他人。取值同Principal | 可选, NotPrincipal与Principal选其一 |
| Action | 可选关键字, 指定本条statement作用的操作, Action字段为媒体存储支持的所有操作集合, 以字符串形式表示, 不区分大小写。支持通配符"*, 表示该资源能进行的所有操作。例如: "Action":["s3:List*", "s3:Get*"] | 可选, Action与NotAction选其一 |
| NotAction | 可选关键字, 指定一组操作, statement匹配除该组操作之外的其他操作。取值同Action | 可选, NotAction与Action选其一 |
| Effect | 必选关键字, 指定本条statement的权限是允许还是拒绝, Effect的值必须为Allow或者Deny | 必选 |
| Resource | 可选关键字, 指定statement起作用的一组资源, 支持通配符"*, 表示所有资源 | 可选, Resource与NotResource选其一 |
| NotResource | 可选关键字, 指定一组资源, statement匹配除该组资源之外的其他资源。取值同Resource | 可选, NotResource与Resource选其一 |
| Condition | 可选关键字, 本条statement生效的条件 | 可选 |

代码示例

```
bool S3Demo::PutBucketPolicy()
{
    Aws::String policyBody = "{\"Version\":\"2012-10-17\",\"Statement\":\
[{\\"Sid\\":\\"1\\",\\"Effect\\":\\"Allow\\",\\"Principal\\":{\\"AWS\\":\\"*\\"},\\"Action\\":\
[\\"s3:GetObject\\"],\\"Resource\\":[\\"arn:aws:s3:::<bucket-name>/*\"]}]}";
    std::shared_ptr<Aws::StringStream> request_body =
        Aws::MakeShared<Aws::StringStream>("");
```

```

*request_body << policyBody;

Aws::S3::Model::PutBucketPolicyRequest request;
request.SetBucket("<your-bucket-name>");
request.SetBody(request_body);

Aws::S3::Model::PutBucketPolicyOutcome outcome = s3_client-
>PutBucketPolicy(request);
if (outcome.IsSuccess()) {
    std::cout << "PutBucketPolicy success";
    return true;
} else
{
    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: PutBucketPolicy: " << (int)err.GetResponseCode() << ",
Message:" <<
    err.GetMessage() << std::endl;

    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|------------|--------|---------------|------|
| Bucket | string | 桶名称 | 是 |
| policyBody | string | 策略内容, json字符串 | 是 |

获取桶策略

功能说明

您可以通过GetBucketPolicy接口获取指定存储桶的授权策略。如果您使用的身份不是该存储桶的拥有者，则调用身份必须对指定存储桶具有GetBucketPolicy权限，且属于该存储桶所有者的账户。如果您没有GetBucketPolicy权限，方法将返回403 Access Denied错误。如果您具有正确的权限，但您没有使用属于存储桶所有者账户的身份，则返回405 Method Not Allowed错误。

代码示例

```

bool S3Demo::GetBucketPolicy()
{
    Aws::S3::Model::GetBucketPolicyRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::GetBucketPolicyOutcome outcome = s3_client-
>GetBucketPolicy(request);
    if (outcome.IsSuccess()) {
        Aws::String line;
        outcome.GetResult().GetPolicy() >> line;
        std::cout << "GetBucketPolicy success " << line;
        return true;
    } else
    {

```

```

    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: GetBucketPolicy: " << (int)err.GetResponseCode() << ",
Message:" <<
    err.GetMessage() << std::endl;

    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|--------|----------|---------------|
| Policy | IOStream | 策略内容, json字符串 |

删除桶策略

功能说明

您可以通过DeleteBucketPolicy接口删除指定存储桶的授权策略。如果您使用的身份不是该存储桶的拥有者，则调用身份必须对指定存储桶具有DeleteBucketPolicy权限，且属于该存储桶所有者的帐户。

如果您没有DeleteBucketPolicy权限，方法将返回403 Access Denied错误。如果您具有正确的权限，但您没有使用属于存储桶所有者账户的身份，则返回405 Method Not Allowed错误。

代码示例

```

bool S3Demo::DeleteBucketPolicy()
{
    Aws::S3::Model::DeleteBucketPolicyRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::DeleteBucketPolicyOutcome outcome = s3_client-
>DeleteBucketPolicy(request);
    if (outcome.IsSuccess()) {
        std::cout << "DeleteBucketPolicy success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteBucketPolicy: " << (int)err.GetResponseCode() <<
", Message:" <<
        err.GetMessage() << std::endl;

        return false;
    }
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

设置桶生命周期配置

功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。您可以使用 PutBucketLifecycleConfiguration 接口设置桶的生命周期配置，配置规则可以通过匹配对象key前缀、标签的方法设置当前版本或者历史版本对象的过期时间，对象过期后会被自动删除。

代码示例

```
bool S3Demo::PutBucketLifecycleConfiguration()
{
    Aws::S3::Model::PutBucketLifecycleConfigurationRequest request;
    request.SetBucket("<your-bucket-name>");
    Aws::S3::Model::BucketLifecycleConfiguration config;
    Aws::S3::Model::LifecycleRule rule;
    rule.SetExpiration(Aws::S3::Model::LifecycleExpiration().WithDays(100));
    rule.SetID("123");
    rule.SetStatus(Aws::S3::Model::ExpirationStatus::Enabled);
    rule.SetFilter(Aws::S3::Model::LifecycleRuleFilter().WithPrefix(""));
    config.AddRules(rule);
    request.SetLifecycleConfiguration(config);

    Aws::S3::Model::PutBucketLifecycleConfigurationOutcome outcome = s3_client-
>PutBucketLifecycleConfiguration(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutBucketLifecycleConfiguration success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutBucketLifecycleConfiguration: " <<
(int)err.GetResponseCode() << ", Message:" <<
        err.GetMessage() << std::endl;

        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

| | | | |
|--------------------------------|--------------------------------|--------------------------------|-------------|
| LifecycleConfiguration | BucketLifecycleConfiguration | 封装了生命周期规则的数组，最多包含1000条规则 | 是否必要 |
| 参数 关于生命周期规则Rule一些说明 | 类型 | 说明 | |
| 参数 | 类型 | 说明 | 是否必要 |
| ID | string | 规则ID | 否 |
| Status | ExpirationStatus | 是否启用规则 (Enabled Disabled) | 是 |
| Expiration | LifecycleExpiration | 文件过期时间 | 否 |
| AbortIncompleteMultipartUpload | AbortIncompleteMultipartUpload | 未完成上传的分片过期时间 | 否 |
| Transitions | Vector<Transition> | 文件转换到低频存储规则（距离修改时间） | 否 |
| Filter | LifecycleRuleFilter | 应用范围，可以指定前缀或对象标签 | 否 |

关于Expiration的说明：

| 参数 | 类型 | 说明 |
|------|-----|------|
| Days | int | 过期天数 |

关于AbortIncompleteMultipartUpload的说明：

| 参数 | 类型 | 说明 |
|---------------------|-----|------|
| DaysAfterInitiation | int | 过期天数 |

关于Transition的说明：

| 参数 | 类型 | 说明 |
|--------------|------------------------|----------|
| Days | int | 转换过期天数 |
| StorageClass | TransitionStorageClass | 要转换的存储类型 |

关于Filter的说明：

| 参数 | 类型 | 说明 |
|--------|--------|---------|
| Prefix | string | 需要过滤的前缀 |

获取桶生命周期配置

功能说明

您可以使用GetBucketLifecycleConfiguration接口获取桶的生命周期配置。

代码示例

```

bool S3Demo::GetBucketLifecycleConfiguration()
{
    Aws::S3::Model::GetBucketLifecycleConfigurationRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::GetBucketLifecycleConfigurationOutcome outcome = s3_client-
>GetBucketLifecycleConfiguration(request);
    if (outcome.IsSuccess()) {
        Aws::Vector<Aws::S3::Model::LifecycleRule> rules =
outcome.GetResult().GetRules();
        for (Aws::S3::Model::LifecycleRule& rule : rules) {
            std::cout << rule.GetExpiration().GetDays() << std::endl;
        }
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: GetBucketLifecycleConfiguration: " <<
(int)err.GetResponseCode() << ", Message:" <<
err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

返回参数

| 参数 | 类型 | 说明 |
|-------|-----------------------|--|
| Rules | Vector<LifecycleRule> | 一个描述生命周期管理的规则数组，一条规则包含了规则ID、匹配的对象key前缀、匹配的对象标签信息、当前版本对象过期时间、历史版本对象过期时间和是否生效标识等信息 |

关于生命周期规则Rule一些说明

| 参数 | 类型 | 说明 |
|--------------------------------|--------------------------------|--------------------------------|
| ID | string | 规则ID |
| Status | ExpirationStatus | 是否启用规则 (Enabled Disabled) |
| Expiration | LifecycleExpiration | 文件过期时间 |
| AbortIncompleteMultipartUpload | AbortIncompleteMultipartUpload | 未完成上传的分片过期时间 |
| Transitions | Vector<Transition> | 文件转换到低频存储规则（距离修改时间） |
| Filter | LifecycleRuleFilter | 应用范围，可以指定前缀或对象标签 |

关于Expiration的说明：

| 参数 | 类型 | 说明 |
|------|-----|------|
| Days | int | 过期天数 |

关于AbortIncompleteMultipartUpload的说明：

| 参数 | 类型 | 说明 |
|---------------------|-----|------|
| DaysAfterInitiation | int | 过期天数 |

关于Transition的说明：

| 参数 | 类型 | 说明 |
|--------------|------------------------|----------|
| Days | int | 转换过期天数 |
| StorageClass | TransitionStorageClass | 要转换的存储类型 |

关于Filter的说明：

| 参数 | 类型 | 说明 |
|--------|--------|---------|
| Prefix | string | 需要过滤的前缀 |

删除桶生命周期配置

功能说明

您可以使用DeleteBucketLifecycle接口删除桶的生命周期配置。

代码示例

```
bool S3Demo::DeleteBucketLifecycle()
{
    Aws::S3::Model::DeleteBucketLifecycleRequest request;
    request.SetBucket("<your-bucket-name>");
}
```

```

    Aws::S3::Model::DeleteBucketLifecycleOutcome outcome = s3_client-
>DeleteBucketLifecycle(request);
    if (outcome.IsSuccess()) {
        std::cout << "DeleteBucketLifecycle success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteBucketLifecycle: " << (int)err.GetResponseCode()
<< ", Message:" <<
            err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

设置桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过PutBucketCors接口设置桶的跨域访问配置。

代码示例

```

bool S3Demo::PutBucketCors()
{
    Aws::S3::Model::PutBucketCorsRequest request;
    request.SetBucket("<your-bucket-name>");
    Aws::S3::Model::CORSConfiguration config;
    Aws::S3::Model::CORSRule rule;
    rule.AddAllowedMethods("PUT");
    rule.AddAllowedMethods("GET");
    rule.AddAllowedMethods("HEAD");
    rule.AddAllowedMethods("POST");
    rule.AddAllowedMethods("DELETE");
    rule.AddAllowedHeaders("*");
    rule.AddAllowedOrigins("*"); // 可以使用http://domain:port
    rule.AddExposeHeaders("ETag");
    rule.SetMaxAgeSeconds(3600);
    config.AddCORSRules(rule);
    request.SetCORSConfiguration(config);

    Aws::S3::Model::PutBucketCorsOutcome outcome = s3_client-
>PutBucketCors(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutBucketCors success";
        return true;
    }
}

```

```

    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutBucketCors: " << (int)err.GetResponseCode() << ",
Message:" <<
            err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-------------------|-------------------|----------|------|
| Bucket | string | 桶名称 | 是 |
| CORSConfiguration | CORSConfiguration | 跨域访问规则数组 | 是 |

关于CORSRule一些说明

| 参数 | 说明 |
|----------------|----------------------|
| AllowedMethods | 允许的请求方法 |
| AllowedOrigins | 允许的请求源 |
| AllowedHeaders | 允许的请求头 |
| ExposedHeaders | 允许返回的Response Header |
| MaxAgeSeconds | 跨域请求结果的缓存时间 |

获取桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过GetBucketCors接口获取桶跨域访问配置。

代码示例

```

bool S3Demo::GetBucketCors()
{
    Aws::S3::Model::GetBucketCorsRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::GetBucketCorsOutcome outcome = s3_client-
>GetBucketCors(request);
    if (outcome.IsSuccess()) {
        Aws::Vector<Aws::S3::Model::CORSRule> rules =
outcome.GetResult().GetCORSRules();
        for (Aws::S3::Model::CORSRule& rule : rules) {
            std::cout << rule.GetMaxAgeSeconds() << std::endl;
        }
    }
}

```

```

    }
    return true;
} else
{
    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: GetBucketCors: " << (int)err.GetResponseCode() << ",
Message:" <<
        err.GetMessage() << std::endl;

    return false;
}
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|-----------|------------------|----------|
| CORSRules | Vector<CORSRule> | 跨域访问规则数组 |

关于CORSRule一些说明

| 参数 | 说明 |
|----------------|----------------------|
| AllowedMethods | 允许的请求方法 |
| AllowedOrigins | 允许的请求源 |
| AllowedHeaders | 允许的请求头 |
| ExposedHeaders | 允许返回的Response Header |
| MaxAgeSeconds | 跨域请求结果的缓存时间 |

删除桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过DeleteBucketCors接口删除桶跨域访问配置。

代码示例

```

bool S3Demo::DeleteBucketCors()
{
    Aws::S3::Model::DeleteBucketCorsRequest request;
    request.SetBucket("<your-bucket-name>");
}

```

```

    Aws::S3::Model::DeleteBucketCorsOutcome outcome = s3_client-
>DeleteBucketCors(request);
    if (outcome.IsSuccess()) {
        std::cout << "DeleteBucketCors success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteBucketCors: " << (int)err.GetResponseCode() << ",
Message:" <<
        err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

设置桶版本控制状态

功能说明

通过媒体存储提供的版本控制，您可以在一个桶中保留多个对象版本。例如，image.jpg(版本1)和image.jpg(版本2)。如果您希望防止自己意外覆盖和删除版本，或存档对象，以便您可以检索早期版本的对象，您可以开启版本控制功能。

您必须在您的存储桶上显式启用版本控制。默认情况下，版本控制处于禁用状态。无论您是否已启用版本控制，您的存储桶中的每个对象都具有版本 ID。如果未启用版本控制，则版本 ID 值被设置为空。如果已启用版本控制，则对象会被指定唯一版本 ID 值。在存储桶上启用版本控制时，该存储桶中的现有对象（如果有）不会发生更改：版本 ID（空）、内容和权限保持为相同。

在开启版本控制功能后，上传同名对象将不再删除旧对象，而是添加一个新的对象。普通的删除操作也不会将对象彻底删除，而是添加一个 Delete Marker 作为标识。容器开启版本控制功能之后，无法再关闭该功能，只能暂停。您可以使用 PutBucketVersioning 接口开启和暂停版本控制。

代码示例

以下代码展示如何开启版本控制

```

bool S3Demo::PutBucketVersioning()
{
    Aws::S3::Model::VersioningConfiguration config;
    config.SetStatus(Aws::S3::Model::BucketVersioningStatus::Enabled);
    Aws::S3::Model::PutBucketVersioningRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetVersioningConfiguration(config);

    Aws::S3::Model::PutBucketVersioningOutcome outcome = s3_client-
>PutBucketVersioning(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutBucketVersioning success";
    }
}

```

```

    return true;
} else
{
    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: PutBPutBucketVersioningucketAcl: " <<
(int)err.GetResponseCode() << ", Message:" <<
    err.GetMessage() << std::endl;

    return false;
}
}

```

以下代码展示如何暂停版本控制

```

bool S3Demo::PutBucketVersioning()
{
    Aws::S3::Model::VersioningConfiguration config;
    config.SetStatus(Aws::S3::Model::BucketVersioningStatus::Suspended);
    Aws::S3::Model::PutBucketVersioningRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetVersioningConfiguration(config);

    Aws::S3::Model::PutBucketVersioningOutcome outcome = s3_client-
>PutBucketVersioning(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutBucketVersioning success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutBPutBucketVersioningucketAcl: " <<
(int)err.GetResponseCode() << ", Message:" <<
        err.GetMessage() << std::endl;

        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-------------------------|-------------------------|----------------|------|
| Bucket | string | 存储桶的名称 | 是 |
| VersioningConfiguration | VersioningConfiguration | 封装了设置版本控制状态的参数 | 是 |

VersioningConfiguration说明

| 参数 | 类型 | 说明 |
|--------|------------------------|-------------------------------|
| Status | BucketVersioningStatus | Enabled Suspended, 版本控制开关状态 |

获取桶版本控制状态

功能说明

您可以使用 GetBucketVersioning 接口获取版本控制状态。

代码示例

```
bool S3Demo::GetBucketVersioning()
{
    Aws::S3::Model::GetBucketVersioningRequest request;
    request.SetBucket("<your-bucket-name>");

    Aws::S3::Model::GetBucketVersioningOutcome outcome = s3_client-
>GetBucketVersioning(request);
    if (outcome.IsSuccess()) {

        auto status = outcome.GetResult().GetStatus();
        std::cout << "status:" << (int)status << std::endl;
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: GetBucketVersioning: " << (int)err.GetResponseCode() <<
", Message:" <<
        err.GetMessage() << std::endl;

        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名称 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|--------|------------------------|-------------------------------|
| Status | BucketVersioningStatus | Enabled Suspended, 版本控制开关状态 |

对象相关接口

获取对象列表

功能说明

您可以使用 ListObjects 接口获取某一个桶中的所有对象。

代码示例

```

bool S3Demo::ListObjects()
{
    Aws::S3::Model::ListObjectsRequest request;
    request.WithBucket("<your-bucket-name>");

    auto outcome = s3_client->ListObjects(request);
    if (outcome.IsSuccess()) {
        std::cout << "Objects in bucket:" << std::endl;

        Aws::Vector<Aws::S3::Model::Object> objects =
            outcome.GetResult().GetContents();

        for (Aws::S3::Model::Object& object : objects) {
            std::cout << object.GetKey() << std::endl;
        }

        return true;
    }
    else {
        std::cout << "Error: ListObjects: " << outcome.GetError().GetMessage() <<
            std::endl;
        return false;
    }
}

```

如果 list 大于1000，则返回的结果中 isTruncated 为true，通过返回的 nextMarker 标记可以作为下次读取的起点。列举所有对象示例代码如下：

```

bool S3Demo::ListObjects2()
{
    Aws::S3::Model::ListObjectsRequest request;
    request.WithBucket("<your-bucket-name>");

    Aws::String nextMarker = "";
    bool isTruncated = false;
    do {
        request.withMarker(nextMarker);
        auto outcome = s3_client->ListObjects(request);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: ListObjects: " << outcome.GetError().GetMessage()
                << std::endl;
            return false;
        }
        Aws::Vector<Aws::S3::Model::Object> objects =
            outcome.GetResult().GetContents();
        for (Aws::S3::Model::Object &object : objects) {
            std::cout << object.GetKey() << std::endl;
        }
        nextMarker = outcome.GetResult().GetNextMarker();
        isTruncated = outcome.GetResult().GetIsTruncated();
    } while (isTruncated);
    return true;
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|---------|--------|---------------------------------|------|
| Bucket | string | 桶名称 | 是 |
| MaxKeys | int | 设置响应中返回的最大键数。默认值和可设置最大值均为1000 | 否 |
| Prefix | string | 指定列出对象的键名需要包含的前缀 | 否 |
| Marker | string | 用于在某一个具体的键名后列出对象，可指定存储桶中的任意一个键名 | 否 |

返回结果

| 参数 | 类型 | 说明 |
|----------|----------|------|
| Contents | Object数组 | 对象列表 |

上传对象

功能说明

您可以使用PutObject接口上传对象。

代码示例

```
bool S3Demo::PutObject()
{
    const Aws::String file_name = "<file-path>";
    const Aws::String object_name = "<your-object-key>";
    const Aws::String bucket_name = "<your-bucket-name>";
    // Verify that the file exists.
    struct stat buffer;
    if (stat(file_name.c_str(), &buffer) == -1)
    {
        std::cout << "Error: PutObject: File '" << object_name << "' does not exist." << std::endl;
        return false;
    }

    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(bucket_name);
    request.SetKey(object_name);
    request.SetACL(Aws::S3::Model::ObjectCannedACL::public_read); // 设置ACL

    std::shared_ptr<Aws::IOStream> input_data =
        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
            file_name.c_str(),
            std::ios_base::in | std::ios_base::binary);

    request.SetBody(input_data);

    Aws::S3::Model::PutObjectOutcome outcome = s3_client->PutObject(request);
    if (outcome.IsSuccess()) {
```

```

std::cout << "Added object '" << object_name << "' to bucket '"
  << bucket_name << "'.";
return true;
}
else
{
std::cout << "Error: PutObject: " << outcome.GetError().GetMessage() <<
std::endl;
return false;
}
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|----------|-----------------|---|------|
| Bucket | string | 桶名 | 是 |
| Key | string | 对象名 | 是 |
| Body | IOStream | 要上传的数据流对象 | 是 |
| ACL | ObjectCannedACL | 对象访问权限, 取值private public-read public-read-write | 否 |
| Metadata | Map | 自定义元数据 | 否 |

返回结果

| 参数 | 类型 | 说明 |
|------|--------|---------|
| ETag | string | 对象的唯一标签 |

注意: PutObject对文件大小有限制, 最大能上传5GB大小的文件, 超过5GB需要使用分片上传。

下载对象

功能说明

您可以使用GetObject接口获取指定桶中的指定对象的内容。

代码示例

```

bool S3Demo::GetObject()
{
    const Aws::String object_name = "<your-object-key>";
    Aws::S3::Model::GetObjectRequest object_request;
    object_request.SetBucket("<your-bucket-name>");
    object_request.SetKey(object_name);

    Aws::S3::Model::GetObjectOutcome get_object_outcome = s3_client-
>GetObject(object_request);
    if (get_object_outcome.IsSuccess()) {

```

```

    auto& retrieved_file =
get_object_outcome.GetResultWithOwnership().GetBody();

    // Print a beginning portion of the text file.
    std::cout << "Beginning of file contents:\n";
    char file_data[255] = { 0 };
    retrieved_file.getline(file_data, 254);
    std::cout << file_data << std::endl;
    return true;
}
else {
    auto err = get_object_outcome.GetError();
    std::cout << "Error: GetObject: " <<
        err.GetExceptionName() << ": " << err.GetMessage() << std::endl;
    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必须 |
|--------|--------|-----|------|
| Bucket | string | 桶名 | 是 |
| Key | string | 对象名 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|----------|----------|--------|
| Body | IOStream | 对象数据内容 |
| Metadata | Map | 自定义元数据 |

复制对象

功能说明

您可以使用CopyObject接口拷贝某一个桶中的对象到另外一个指定的桶中。

代码示例

```

bool S3Demo::CopyObject()
{
    const Aws::String bucket_source = "<source-bucket-name>";
    const Aws::String bucket_dest = "<dst-bucket-name>";
    const Aws::String object_source = "<source-object-key>";
    const Aws::String object_dest = "<dst-object-key>";

    Aws::S3::Model::CopyObjectRequest request;
    request.SetBucket(bucket_dest);
    request.SetKey(object_dest);
    request.SetCopySource(bucket_source + "/" + object_source); // 注意这个参数
}

```

```

Aws::S3::Model::CopyObjectOutcome outcome = s3_client->CopyObject(request);
if (outcome.IsSuccess()) {
    std::cout << "CopyObject " << object_source << " to " << object_dest <<
std::endl;
    return true;
}
else {
    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: CopyObject: " << (int)err.GetResponseCode() << ",
Message:" <<
    err.GetMessage() << std::endl;
    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|------------|--------|---|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 目的对象key | 是 |
| CopySource | string | URL格式的拷贝对象数据来源，包含了桶名称和对象key的信息，二者之间使用正斜杠 (/) 分割 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|------|--------|---------|
| ETag | string | 对象的唯一标签 |

删除对象

功能说明

您可以使用DeleteObject接口删除某一个桶中的对象。

代码示例

```

bool S3Demo::DeleteObject()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::DeleteObjectRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::DeleteObjectOutcome outcome = s3_client-
>DeleteObject(request);
    if (outcome.IsSuccess()) {

```

```

        std::cout << "DeleteObject " << object_name << " success";
        return true;
    }
    else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteObject: " << (int)err.GetResponseCode() << ",
Message:" <<
            err.GetMessage() << std::endl;
        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-----|------|
| Bucket | string | 桶名 | 是 |
| Key | string | 对象名 | 是 |

批量删除对象

功能说明

您可以使用DeleteObjects接口批量删除多个对象，可以减少发起多个请求去删除大量对象的花销。DeleteObjects操作发起一个包含了最多1000个key的删除请求，媒体存储服务会对相应的对象逐个进行删除，并且将删除成功或者失败的结果通过response返回。如果请求删除的对象不存在，会返回已删除的结果。

DeleteObjects操作返回包含verbose 和quiet两种response模式。verbose response是默认返回模式，该模式的返回结果包含了每个key的删除结果。quiet response返回模式返回的结果仅包含了删除失败的key，对于一个完全成功的删除操作，该返回模式不在相应消息体中返回任何信息。

代码示例

```

bool S3Demo::DeleteObjects()
{
    Aws::S3::Model::DeleteObjectsRequest request;
    request.SetBucket("<your-bucket-name>");
    Aws::S3::Model::Delete deleteObject;
    deleteObject.AddObjects(Aws::S3::Model::ObjectIdentifier().WithKey("ExampleObject.txt"));
    deleteObject.AddObjects(Aws::S3::Model::ObjectIdentifier().WithKey("ExampleObject1.txt"));
    request.SetDelete(deleteObject);

    Aws::S3::Model::DeleteObjectsOutcome outcome = s3_client->DeleteObjects(request);
    if (outcome.IsSuccess()) {
        std::cout << "DeleteObjects success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteObjects: " << (int)err.GetResponseCode() << ",
Message:" <<

```

```

    err.GetMessage() << std::endl;
    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-------------|------|
| Bucket | string | 桶名称 | 是 |
| Delete | Delete | 要删除的对象key列表 | 是 |

获取对象元数据

功能说明

您可以使用HeadObject接口获取对象元数据信息。

代码示例

```

bool S3Demo::HeadObject()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::HeadObjectRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::HeadObjectOutcome outcome = s3_client->HeadObject(request);
    if (outcome.IsSuccess()) {
        std::cout << "HeadObject " << object_name << " success";
        return true;
    }
    else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: HeadObject: " << (int)err.GetResponseCode() << ",
Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |

设置对象访问权限

功能说明

媒体存储支持一组预先定义的授权，称为Canned ACL。每个Canned ACL都有一组预定义的被授权者和权限，下表列出了相关的预定义授权含义。

| ACL | 权限 | 描述 |
|--------------------|--------|---------------------------|
| private | 私有读写 | 对象拥有者有读写权限，其他用户没有访问权限。 |
| public-read | 公共读私有写 | 对象拥有者有读写权限，其他用户只有该对象的读权限。 |
| public-read-write | 公共读写 | 所有用户都有该对象的读写权限。 |
| authenticated-read | 注册用户可读 | 对象拥有者有读写权限，注册用户具有该对象的读限。 |

PutObjectAcl操作可以为媒体存储服务中的对象设置ACL。对一个对象执行该操作需要具有WRITE_ACP权限。

代码示例

```
bool S3Demo::PutObjectAcl()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::PutObjectAclRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);
    request.SetACL(Aws::S3::Model::ObjectCannedACL::public_read);

    Aws::S3::Model::PutObjectAclOutcome outcome = s3_client->PutObjectAcl(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutObjectAcl " << object_name << " success";
        return true;
    } else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutObjectAcl: " << (int)err.GetResponseCode() << ",
        Message:" <<
            err.GetMessage() << std::endl;

        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|-----------------|-------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| ACL | ObjectCannedACL | acl值 | 是 |

获取对象访问权限

功能说明

您可以使用 `GetObjectAcl` 操作获取对象的 access control list (ACL) 信息。

代码示例

```
bool S3Demo::GetObjectAcl()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::GetObjectAclRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::GetObjectAclOutcome outcome = s3_client->GetObjectAcl(request);
    if (outcome.IsSuccess()) {

        Aws::Vector<Aws::S3::Model::Grant> grants = outcome.GetResult().GetGrants();
        for (Aws::S3::Model::Grant& grant : grants)
        {
            std::cout << "Grant:" << grant.GetGrantee().GetDisplayName() << ",
permission:" << (int)grant.GetPermission() << std::endl;
        }
        return true;
    }
    else
    {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: GetObjectAcl: " << (int)err.GetResponseCode() << ",
Message:" <<
err.GetMessage() << std::endl;

        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |

返回参数

| 参数 | 类型 | 说明 |
|--------|---------|---------------|
| Owner | Owner | 所有者信息 |
| Grants | Grant数组 | 每种类型用户的详细权限信息 |

获取对象标签

功能说明

您可以使用GetObjectTagging接口获取对象标签。

代码示例

```
bool S3Demo::GetObjectTagging()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::GetObjectTaggingRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::GetObjectTaggingOutcome outcome = s3_client-
>GetObjectTagging(request);
    if (outcome.IsSuccess()) {
        std::cout << "GetObjectTagging " << object_name << " success";
        Aws::Vector<Aws::S3::Model::Tag> tags = outcome.GetResult().GetTagSet();
        for (Aws::S3::Model::Tag& tag: tags)
        {
            std::cout << tag.GetKey() << ":" << tag.GetValue() << std::endl;
        }
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: GetObjectTagging: " << (int)err.GetResponseCode() << ",
Message:" <<
            err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-----------|--------|----------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| VersionId | string | 设置标签信息的对象的版本Id | 否 |

返回结果

| 参数 | 类型 | 说明 |
|--------|-----------|---|
| TagSet | Tag 数组 | 设置的标签信息，包含了一个Tag结构体的数组，每个Tag以Key-Value的形式说明了标签的内容 |

删除对象标签

功能说明

您可以使用DeleteObjectTagging接口删除对象标签。

代码示例

```
bool S3Demo::DeleteObjectTagging()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::DeleteObjectTaggingRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::DeleteObjectTaggingOutcome outcome = s3_client-
>DeleteObjectTagging(request);
    if (outcome.IsSuccess()) {
        std::cout << "DeleteObjectTagging " << object_name << " success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: DeleteObjectTagging: " << (int)err.GetResponseCode() <<
", Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-----------|--------|----------------|------|
| Bucket | string | 执行本操作的桶名称 | 是 |
| Key | string | 设置标签信息的对象key | 是 |
| VersionId | string | 设置标签信息的对象的版本Id | 否 |

设置对象标签

功能说明

您可以使用PutObjectTagging接口为对象设置标签。bucket的拥有者默认拥有给bucket中的对象设置标签的权限，并且可以将权限授予其他用户。每次执行PutObjectTagging操作会覆盖对象已有的标签信息。标签是一个键值对，每个对象最多可以设置10个标签，标签Key和Value区分大小写，并且Key不可重复。每个标签的Key长度不超过128字节，Value长度不超过256字节。SDK通过HTTP header的方式设置标签且标签中包含任意字符时，需要对标签的Key和Value做URL编码。设置对象标签信息不会更新对象的最新更改时间。

代码示例

```
bool S3Demo::PutObjectTagging()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::PutObjectTaggingRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);
    Aws::S3::Model::Tagging tagging;
    tagging.AddTagSet(Aws::S3::Model::Tag().WithKey("key1").WithValue("value1"));
    tagging.AddTagSet(Aws::S3::Model::Tag().WithKey("key2").WithValue("value2"));
    request.SetTagging(tagging);

    Aws::S3::Model::PutObjectTaggingOutcome outcome = s3_client-
>PutObjectTagging(request);
    if (outcome.IsSuccess()) {
        std::cout << "PutObjectTagging " << object_name << " success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: PutObjectTagging: " << (int)err.GetResponseCode() << ",
Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-----------|---------|---|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| Tagging | Tagging | 设置的标签信息，包含了一个Tag结构体的数组，每个Tag以Key-Value的形式说明了标签的内容 | 是 |
| VersionId | string | 设置标签信息的对象的版本Id | 否 |

生成下载预签名URL

功能说明

您可以使用GeneratePresignedUrl接口为一个指定对象生成一个预签名的下载链接，访问该链接可以直接下载该对象。

代码示例

生成下载预签名URL：

```
bool S3Demo::GeneratePresignUrl()
{
    const Aws::String object_name = "<your-object-key>";
    long expirationInSeconds = 900;

    Aws::String path = s3_client->GeneratePresignedUrl("<your-bucket-name>",
    object_name, Aws::Http::HttpMethod::HTTP_GET, expirationInSeconds);

    std::cout << "GeneratePresignUrl: " << path << std::endl;
    return true;
}
```

生成下载对象的预签名URL后，可以通过该URL下载文件：

```
void S3Demo::GetObjusingPresignedUrl(const std::string& url, const std::string&
file_path) {
    try {
        std::ofstream file(file_path, std::ios::binary);
        if (!file.is_open()) {
            std::cerr << "Failed to open file for writing: " << file_path <<
std::endl;
            return;
        }

        curlpp::Easy request;
        request.setOpt(curlpp::options::Url(url));
        request.setOpt(curlpp::options::WriteStream(&file));
        request.perform();

        std::cout << "Download successful: " << file_path << std::endl;
    } catch (curlpp::RuntimeError& e) {
        std::cerr << "Runtime Error: " << e.what() << std::endl;
    } catch (curlpp::LogicError& e) {
        std::cerr << "Logic Error: " << e.what() << std::endl;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|------------|--------------------------------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| method | HttpMethod | http请求方法， HTTP_GET表示下载， HTTP_PUT表示上传 | 是 |

| | | | |
|---------------------------|-----------------|----------------|---------------------|
| expirationInSeconds 参数 | long long 类型 | 超时时间 (秒) 说明 | 默认 否 是否 必要 |
|---------------------------|-----------------|----------------|---------------------|

返回结果

生成对应的预签名下载 URL，该链接允许用户在指定的时间内直接从媒体存储下载对象。

生成上传预签名URL

功能说明

您可以使用GeneratePresignedUrl接口为一个指定对象生成一个预签名的上传链接，访问该链接可以直接上传该对象。

代码示例

生成上传预签名URL：

```
bool S3Demo::GeneratePutPresignedUrl() {
    const Aws::String object_name = "ExampleObject.txt";
    long expirationInSeconds = 900;

    Aws::String path = s3_client->GeneratePresignedUrl(s3_bucket_name,
        object_name, Aws::Http::HttpMethod::HTTP_PUT, expirationInSeconds);

    std::cout << "GeneratePutPresignedUrl: " << path << std::endl;
    return true;
}
```

通过该预签名URL，可以直接将文件上传到指定的桶：

```
void S3Demo::PutObjUsingPresignedUrl(const std::string& url, const std::string&
file_path) {
    try {
        std::ifstream file(file_path, std::ios::binary);
        if (!file.is_open()) {
            std::cerr << "Failed to open file for reading: " << file_path <<
std::endl;
            return;
        }

        // 获取文件大小
        file.seekg(0, std::ios::end);
        std::streamsize file_size = file.tellg();
        file.seekg(0, std::ios::beg);

        // 使用 curlpp 进行 PUT 上传
        curlpp::Easy request;
        request.setOpt(curlpp::options::Url(url));
        request.setOpt(curlpp::options::Upload(true));
        request.setOpt(curlpp::options::InfileSize(file_size));
        request.setOpt(curlpp::options::ReadStream(&file));
        request.perform();

        std::cout << "Upload successful: " << file_path << std::endl;
    }
```

```

    } catch (curlpp::RuntimeError& e) {
        std::cerr << "Runtime Error: " << e.what() << std::endl;
    } catch (curlpp::LogicError& e) {
        std::cerr << "Logic Error: " << e.what() << std::endl;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|---------------------|------------|--------------------------------------|---------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| method | HttpMethod | http请求方法, HTTP_GET表示下载, HTTP_PUT表示上传 | 是 |
| expirationInSeconds | long long | 超时时间 (秒) | 否, 默认7天 |

返回结果

生成对应的预签名上传 URL, 该链接允许用户在指定的时间内直接将对象上传到媒体存储存储桶。

追加写

功能说明

AppendObject可以对桶中的一个对象进行追加写操作, 如果该对象已经存在, 执行该操作则向文件末尾追加内容, 否则将创建对象。

通过Append操作创建的Object类型为Appendable, 而通过PutObject操作上传的Object的类型是Normal。对Appendable类型的对象进行普通上传操作之后会覆盖原有对象的内容并且将其类型设置为Normal。

Append操作仅可以在未开启版本控制的桶中执行, 如果桶的版本控制状态为启用 (Enabled) 或者暂停 (Suspended) 状态将不支持Append操作。

代码示例

```

bool S3Demo::AppendObject()
{
    const Aws::String file_name = "<file-path>";
    const Aws::String object_name = "<your-object-key>";
    const Aws::String bucket_name = "<your-bucket-name>";
    // Verify that the file exists.
    struct stat buffer;
    if (stat(file_name.c_str(), &buffer) == -1) {
        std::cout << "Error: PutObject: File '" << object_name << "' does not exist." << std::endl;
        return false;
    }
    auto appPos = 0;
}

```

```

Aws::S3::Model::PutObjectRequest request;
request.SetBucket(bucket_name);
request.SetKey(object_name);
request.SetACL(Aws::S3::Model::ObjectCannedACL::public_read_write); // 设置ACL
request.SetAppend(true); // 设置追加模式上传Object
request.SetAppendPosition(appPos); // 指定追加位置

std::shared_ptr<Aws::IOStream> input_data =
    Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
        file_name.c_str(),
        std::ios_base::in | std::ios_base::binary);
std::shared_ptr<Aws::IOStream> append_data =
    Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
        file_name.c_str(),
        std::ios_base::in | std::ios_base::binary);

std::cout<<"首次写入Pos: " << appPos << std::endl;

request.SetBody(input_data);
Aws::S3::Model::PutObjectOutcome outcome = s3_client->PutObject(request);

if (!outcome.IsSuccess())
{
    std::cout << "ERROR: PutObject: " << "Error Msg: " <<
outcome.GetError().GetMessage() << std::endl;
    return false;
}
else
{
    appPos = outcome.GetResult().GetAppendPosition(); // 追加模式下，获取下一次追加的起始位置
    std::cout<<"追加写入Pos: " << appPos << std::endl;
    // 追加写对象
    request.SetAppend(true);
    request.SetAppendPosition(appPos);
    request.SetBody(append_data);

    outcome = s3_client->PutObject(request);
    result = outcome.GetResult();

    std::cout << "Etag:" << outcome.GetResult().GetETag() << std::endl;
    return true;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|----------|-----------|------|
| Bucket | string | 桶名 | 是 |
| Key | string | 对象名 | 是 |
| Body | IOStream | 要上传的数据流对象 | 是 |

| | | | |
|----------------|-----------------|---|---------------|
| Append 参数 | bool 类型 | 是否为Appendable类型 说明 | 是否 必要 是 |
| AppendPosition | int | 追加前对象大小 | |
| ACL | ObjectCannedACL | 对象访问权限，取值private public-read public-read-write | 否 |
| Metadata | Map | 自定义元数据 | 否 |

返回结果

| 参数 | 类型 | 说明 |
|------|--------|---------|
| ETag | string | 对象的唯一标签 |

注意：对Appendable类型的对象进行普通上传操作会将其类型设置为Normal，但无法对Normal类型的对象进行追加上传操作将其类型设置为Appendable类型。

获取多版本对象列表

功能说明

如果桶开启了版本控制，您可以使用 ListObjectVersions接口列举对象的版本，每次list操作最多返回1000个对象。

代码示例

```
bool S3Demo::ListObjectVersions()
{
    Aws::S3::Model::ListObjectVersionsRequest request;
    request.WithBucket("<your-bucket-name>");

    Aws::S3::Model::ListObjectVersionsOutcome outcome = s3_client-
>ListObjectVersions(request);
    if (outcome.IsSuccess()) {
        std::cout << "Objects in bucket:" << std::endl;

        Aws::Vector<Aws::S3::Model::ObjectVersion> objects =
            outcome.GetResult().GetVersions();

        for (Aws::S3::Model::ObjectVersion& object : objects) {
            std::cout << object.GetKey() << std::endl;
        }

        return true;
    } else {
        std::cout << "Error: ListObjectVersions: " <<
outcome.GetError().GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|---------|--------|---------------------------------|------|
| Bucket | string | 桶名称 | 是 |
| MaxKeys | int | 设置响应中返回的最大键数。默认值和可设置最大值均为1000 | 否 |
| Prefix | string | 指定列出对象的键名需要包含的前缀 | 否 |
| Marker | string | 用于在某一个具体的键名后列出对象，可指定存储桶中的任意一个键名 | 否 |

返回结果

| 参数 | 类型 | 说明 |
|----------|-----------------|------|
| Versions | ObjectVersion数组 | 对象列表 |

分片上传接口

融合接口

功能说明

分片上传步骤较多，包括初始化、文件切片、各个分片上传、完成上传。为了简化分片上传，可以使用TransferManager类的UploadFile接口进行分片上传。

代码示例

```
bool S3Demo::TransferUpload()
{
    const Aws::String object_name = "<your-object-key>";
    const Aws::String local_path = "<file-path>";

    std::shared_ptr<Aws::Utils::Threading::PooledThreadExecutor> executor =
        Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>("executor",
25);
    Aws::Transfer::TransferManagerConfiguration transferConfig(executor.get());
    transferConfig.s3Client = s3_client;
    // 默认分片大小5MB
    // transferConfig.bufferSize = Aws::Transfer::MB5;
    // 设置公共读
    //
    transferConfig.putObjectTemplate.SetACL(Aws::S3::Model::ObjectCannedACL::public_read);
    //
    transferConfig.createMultipartUploadTemplate.SetACL(Aws::S3::Model::ObjectCanned
ACL::public_read);

    std::shared_ptr<Aws::Transfer::TransferHandle> requestPtr(nullptr);

    transferConfig.downloadProgressCallback =
```

```

    [](const Aws::Transfer::TransferManager*, const std::shared_ptr<const
Aws::Transfer::TransferHandle>& handle)
    {
        std::cout << "\r" << "<AWS UPLOAD> Upload Progress: " <<
            static_cast<int>(handle->GetBytesTransferred() * 100.0 / handle-
>GetBytesTotalSize()) << " Percent " <<
            handle->GetBytesTransferred() << " bytes\n";
    };

    std::shared_ptr<Aws::Transfer::TransferManager> transferManager =
        Aws::Transfer::TransferManager::Create(transferConfig);

    Aws::String contentType = "binary/octet-stream";
    Aws::Map<Aws::String, Aws::String> metadata;
    requestPtr = transferManager->UploadFile(local_path, "<your-bucket-name>",
        object_name, contentType, metadata);
    requestPtr->waitUntilFinished();

    // Check status
    if (requestPtr->GetStatus() == Aws::Transfer::TransferStatus::COMPLETED) {
        if (requestPtr->GetBytesTotalSize() == requestPtr->GetBytesTransferred()) {
            std::cout << "success" << std::endl;
        }
        else {
            std::cout << "failed" << std::endl;
        }
    }
    else {
        std::cout << "failed" << std::endl;
    }
    return true;
}

```

请求参数

| 参数 | 意义 | 类型 | 是否必要 |
|-------------|-------------------|---------------------|---------|
| local_path | 要上传的本地文件 | string | 是 |
| bucket_name | 桶名 | string | 是 |
| object_name | 对象名 | string | 是 |
| contentType | http contentType头 | string | 是 |
| metadata | 对象自定义元数据 | map<string, string> | 是, 可以为空 |

注意: acl在TransferManagerConfiguration中设置

初始化分片上传任务

功能说明

您可以使用CreateMultipartUpload接口创建上传任务。

代码示例

```

bool S3Demo::CreateMultipartUpload()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::CreateMultipartUploadRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);

    Aws::S3::Model::CreateMultipartUploadOutcome outcome = s3_client-
>CreateMultipartUpload(request);
    if (outcome.IsSuccess()) {
        Aws::String uploadId = outcome.GetResult().GetUploadId();
        std::cout << "CreateMultipartUpload " << object_name << ":" << uploadId << "
success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: CreateMultipartUpload: " << (int)err.GetResponseCode()
<< ", Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|--------|--------|-------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |

返回结果

| 参数 | 类型 | 说明 |
|----------|--------|-----------|
| UploadId | string | 分片上传任务的id |

上传分片

功能说明

初始化分片上传任务后，指定分片上传任务的id可以上传分片数据，可以将大文件分割成分片后上传，除了最后一个分片，每个分片的数据大小为5MB~5GB，每个分片上传任务最多上传10000个分片。您可以使用UploadPart上传分片。

代码示例

```

bool S3Demo::UploadPart()
{
    const Aws::String object_name = "<your-object-key>";
    Aws::String upload_id = "<upload-id>";

```

```

std::shared_ptr<Aws::IOStream> input_data = Aws::MakeShared<Aws::StringStream>
("MyStream");
*input_data << "<upload part content>";
int part_num = 1;

Aws::S3::Model::UploadPartRequest request;
request.SetBucket("<your-bucket-name>");
request.SetKey(object_name);
request.SetBody(input_data);
request.SetUploadId(upload_id);
request.SetPartNumber(part_num);

Aws::S3::Model::UploadPartOutcome outcome = s3_client->UploadPart(request);
if (outcome.IsSuccess()) {
    std::cout << "UploadPart " << object_name << ":" << part_num << ":" <<
outcome.GetResult().GetETag() << " success";
    return true;
} else {
    Aws::S3::S3Error err = outcome.GetError();
    std::cout << "Error: UploadPart: " << (int)err.GetResponseCode() << ",
Message:" <<
    err.GetMessage() << std::endl;
    return false;
}
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|------------|----------|--------------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| Body | IOStream | 对象的数据 | 是 |
| PartNumber | int | 当前分片号码 | 是 |
| UploadId | string | 通过创建上传任务接口获取到的任务Id | 是 |

返回结果

| 参数 | 类型 | 说明 |
|------|--------|---------------------|
| ETag | string | 本次上传分片对应的Entity Tag |

合并分片

功能说明

合并指定分片上传任务id对应任务中已上传的对象分片，使之成为一个完整的文件。您可以使用 CompleteMultipartUpload接口合并分片。

代码示例

```

bool S3Demo::CompleteMultipartUpload()
{
    const Aws::String object_name = "<your-object-key>";
    Aws::String upload_id = "<upload-id>";
    Aws::String eTag = "<part-etag>";
    int part_num = 1;

    Aws::S3::Model::CompletedMultipartUpload multiupload;
    Aws::S3::Model::CompletedPart part;
    part.SetETag(eTag);
    part.SetPartNumber(part_num);
    multiupload.AddParts(part);

    Aws::S3::Model::CompleteMultipartUploadRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);
    request.SetUploadId(upload_id);
    request.SetMultipartUpload(multiupload);

    Aws::S3::Model::CompleteMultipartUploadOutcome outcome = s3_client-
>CompleteMultipartUpload(request);
    if (outcome.IsSuccess()) {
        std::cout << "CompleteMultipartUpload " << object_name << " success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: CompleteMultipartUpload: " <<
(int)err.GetResponseCode() << ", Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}

```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|-----------------|-----------------|--------------------------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| MultipartUpload | MultipartUpload | 包含了每个已上传的分片的ETag和PartNumber等信息 | 是 |
| UploadId | string | 通过创建上传任务接口获取到的任务Id | 是 |

返回结果

| 参数 | 类型 | 说明 |
|------|--------|----------------------|
| ETag | string | 本次上传对象后对应的Entity Tag |

列举分片上传任务

功能说明

您可以使用ListMultipartUploads获取未完成的上传任务。

代码示例

```
bool S3Demo::ListMultipartUploads()
{
    const Aws::String object_name = "<your-object-key>";

    Aws::S3::Model::ListMultipartUploadsRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetMaxUploads(50);

    Aws::S3::Model::ListMultipartUploadsOutcome outcome = s3_client-
>ListMultipartUploads(request);
    if (outcome.IsSuccess()) {
        std::cout << "ListMultipartUploads " << object_name << " success" <<
std::endl;
        Aws::Vector<Aws::S3::Model::MultipartUpload> uploads =
outcome.GetResult().GetUploads();
        for (Aws::S3::Model::MultipartUpload& upload : uploads) {
            std::cout << upload.GetKey() << ":" << upload.GetUploadId() << std::endl;
        }
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: ListMultipartUploads: " << (int)err.GetResponseCode()
<< ", Message:" <<
err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|------------|--------|------------------------|------|
| Bucket | string | 桶名称 | 是 |
| MaxUploads | int | 用于指定获取任务的最大数量 (1-1000) | 否 |

返回结果

| 参数 | 类型 | 说明 |
|---------|---------|---|
| Uploads | Uploads | 包含了零个或多个已初始化的上传分片信息的数组。数组中的每一项包含了分片初始化时间、分片上传操作发起者、对象key、对象拥有者、存储类型和UploadId等信息 |

列举已上传的分片

功能说明

您可以使用ListParts获取一个未完成的上传任务中已完成上传的分片信息。

代码示例

```
bool S3Demo::ListParts()
{
    const Aws::String object_name = "<your-object-key>";
    Aws::String upload_id = "<upload-id>";

    Aws::S3::Model::ListPartsRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);
    request.SetUploadId(upload_id);

    Aws::S3::Model::ListPartsOutcome outcome = s3_client->ListParts(request);
    if (outcome.IsSuccess()) {
        std::cout << "ListParts " << object_name << " success" << std::endl;
        Aws::Vector<Aws::S3::Model::Part> parts = outcome.GetResult().GetParts();
        for (Aws::S3::Model::Part& part : parts) {
            std::cout << part.GetPartNumber() << ":" << part.GetETag() << std::endl;
        }
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: ListParts: " << (int)err.GetResponseCode() << ",
        Message:" <<
            err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|----------|--------|-----------------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| UploadId | string | 指定返回该任务id所属的分片上传的分片信息 | 是 |

返回结果

| 参数 | 类型 | 说明 |
|-------|-------|--|
| Parts | Parts | 包含了已上传分片信息的数组，数组中的每一项包含了该分片的Entity tag、最后修改时间、PartNumber和大小等信息 |

复制分片

功能说明

复制分片操作可以从一个已存在的对象中拷贝指定分片的数据，当拷贝的对象大小超过5GB，必须使用复制分片操作完成对象的复制。除了最后一个分片外，每个拷贝分片的大小范围是[5MB, 5GB]。在拷贝一个大对象之前，需要使用初始化分片上传操作获取一个upload id，在完成拷贝操作之后，需要使用合并分片操作组装已拷贝的分片成为一个对象。您可以使用UploadPartCopy复制一个分片。

代码示例

```
bool S3Demo::UploadPartCopy()
{
    const Aws::String bucket_source = "<source-bucket-name>";
    const Aws::String bucket_dest = "<dst-bucket-name>";
    const Aws::String object_source = "<source-object-key>";
    const Aws::String object_dest = "<dst-object-key>";

    Aws::String copy_source = bucket_source + "/" + object_source;
    Aws::String upload_id = "<upload-id>";
    Aws::String copy_source_range = "bytes=0-5242879";
    int part_num = 1;

    Aws::S3::Model::UploadPartCopyRequest request;
    request.SetBucket(bucket_dest);
    request.SetKey(object_dest);
    request.SetCopySource(copy_source);
    request.SetUploadId(upload_id);
    request.SetPartNumber(part_num);
    request.SetCopySourceRange(copy_source_range);

    Aws::S3::Model::UploadPartCopyOutcome outcome = s3_client->UploadPartCopy(request);
    if (outcome.IsSuccess()) {
        std::cout << "UploadPartCopy " << object_dest << ":" << part_num << ":" <<
outcome.GetResult().GetCopyPartResult().GetETag() << " success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: UploadPartCopy: " << (int)err.GetResponseCode() << ",
Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|------------|--------|--------------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| PartNumber | int | 当前分片号码 | 是 |
| UploadId | string | 通过创建上传任务接口获取到的任务Id | 是 |

| | | | |
|------------------|--------------|---|---------------|
| CopySource 参数 | string 类型 | URL格式的拷贝对象数据来源，包含了桶名称和对象key的信息，二者之间使用正斜杠 (/) 分割 说明 | 是否 必要 否 |
| CopySourceRange | string | 指定本次分片拷贝的数据范围，必须是"bytes=first-last"的格式，例如"bytes=0-9"表示拷贝原对象中前10字节的数据，只有当拷贝的分片大小大于5MB的时候有效 | |

返回结果

| 参数 | 类型 | 说明 |
|----------------|----------------|-----------------------------|
| CopyPartResult | CopyPartResult | 包含拷贝分片的Entity Tag和最后修改时间等信息 |

取消分片上传任务

功能说明

您可以使用AbortMultipartUpload终止一个分片上传任务。

代码示例

```
bool S3Demo::AbortMultipartUpload()
{
    const Aws::String object_name = "<your-object-key>";
    Aws::String upload_id = "<upload-id>";

    Aws::S3::Model::AbortMultipartUploadRequest request;
    request.SetBucket("<your-bucket-name>");
    request.SetKey(object_name);
    request.SetUploadId(upload_id);

    Aws::S3::Model::AbortMultipartUploadOutcome outcome = s3_client-
>AbortMultipartUpload(request);
    if (outcome.IsSuccess()) {
        std::cout << "AbortMultipartUpload " << object_name << " success";
        return true;
    } else {
        Aws::S3::S3Error err = outcome.GetError();
        std::cout << "Error: AbortMultipartUpload: " << (int)err.GetResponseCode()
<< ", Message:" <<
        err.GetMessage() << std::endl;
        return false;
    }
}
```

请求参数

| 参数 | 类型 | 说明 | 是否必要 |
|----------|--------|-------------|------|
| Bucket | string | 桶名称 | 是 |
| Key | string | 对象key | 是 |
| UploadId | string | 需要终止的上传任务id | 是 |

安全凭证服务(STS)

STS即Secure Token Service 是一种安全凭证服务，可以使用STS来完成对于临时用户的访问授权。对于跨用户短期访问对象存储资源时，可以使用STS服务。这样就不需要透露主账号AK/SK，只需要生成一个短期访问凭证给需要的用户使用即可，避免主账号AK/SK泄露带来的安全风险。

初始化STS服务

```
Aws::String ak = "<your-access-key>";
Aws::String sk = "<your-secret-access-key>";
Aws::String endPoint = "<your-endpoint>";

Aws::Auth::AWSCredentials cred(ak, sk);
Aws::Client::ClientConfiguration cfg;
cfg.endpointOverride = endPoint;
cfg.scheme = Aws::Http::Scheme::HTTP;
cfg.verifySSL = false;
sts_client = new Aws::STS::STSClient(cred, cfg);
```

获取临时token

```
const Aws::String roleArn = "arn:aws:iam::role/xxxxxx";
const Aws::String roleSessionName = "<your-session-name>";
const Aws::String bucket_name = "<your-bucket-name>";
const Aws::String policy = "{\"version\":\"2012-10-17\","
    "\"Statement\":["
    "{\"Effect\":\"Allow\","
    "\"Action\":[\"s3:*\"],\" // 允许进行 s3 的所有操作。如果仅需要上传，这里可以设置为
PutObject
    \"Resource\":[\"arn:aws:s3::\" + bucket_name + "\",\"arn:aws:s3::\" +
bucket_name + \"/*\"]\"// 允许操作默认桶中的所有文件，可以修改此处来保证操作的文件
    ]}";
Aws::STS::Model::AssumeRoleRequest request;
request.SetPolicy(policy);
request.SetRoleArn(roleArn);
request.SetRoleSessionName(roleSessionName);
std::cout << "policy:" << policy << std::endl;
Aws::STS::Model::AssumeRoleOutcome outcome = sts_client->AssumeRole(request);
if (outcome.IsSuccess())
{
    auto& cred = outcome.GetResult().GetCredentials();
    std::cout << "ak:" << cred.GetAccessKeyId() << std::endl;
    std::cout << "sk:" << cred.GetSecretAccessKey() << std::endl;
    std::cout << "token:" << cred.GetSessionToken() << std::endl;
    return true;
}
```

```

}
else
{
    auto err = outcome.GetError();
    std::cout << "Error: AssumeRole: " <<
        err.GetExceptionName() << ", " << err.GetMessage() << std::endl;
    return false;
}

```

| 参数 | 类型 | 描述 | 是否必要 |
|-----------------|---------|----------------------|------|
| RoleArn | String | 角色的ARN，在控制台创建角色后可以查看 | 是 |
| Policy | String | 角色的policy，需要是json格式 | 是 |
| RoleSessionName | String | 角色会话名称 | 是 |
| DurationSeconds | Integer | 会话有效期时间，默认为3600s | 否 |

常见问题

关于cpp程序的编译指令

编译需要指定头文件目录和库文件目录，添加动态链接库aws-cpp-sdk-core和aws-cpp-sdk-s3。

```

g++ -I<install/prefix/path>/include -L<install/prefix/path>/lib64 -laws-cpp-sdk-core -laws-cpp-sdk-s3 xxx.cpp -o xxx

```

如果需要使用分片上传融合接口，则需要增加aws-cpp-sdk-transfer依赖库；如果需要使用sts功能接口，则需要增加aws-cpp-sdk-sts依赖库。

```

g++ -I<install/prefix/path>/include -L<install/prefix/path>/lib64 -laws-cpp-sdk-core -laws-cpp-sdk-s3 -laws-cpp-sdk-transfer -laws-cpp-sdk-sts xxx.cpp -o xxx

```

运行程序需要链接动态库时，提示找不到相关的.so库的报错

报错描述如下：

```

error while loading shared libraries: libaws-cpp-sdk-core.so: cannot open shared object file: No such file or dictionary

```

解决方法：可以使用 -Wl,-rpath 命令在编译的时候指定运行时库文件查找路径。

```

g++ -I<install/prefix/path>/include -L<install/prefix/path>/lib64 -wl,-rpath=<install/prefix/path>/lib64 -laws-cpp-sdk-core -laws-cpp-sdk-s3 xxx.cpp -o xxx

```

找不到公共库

报错描述如下：

```
warning: libaws-c-event-stream.so.0unstable, not found (try using -rpath or -rpath-link)  
warning: libaws-c-common.so.0unstable, not found (try using -rpath or -rpath-link)  
warning: libaws-checksums.so, not found (try using -rpath or -rpath-link)
```

使用cmake的时候会生成公共库的so文件，并安装到 `<install/prefix/path>/lib64` 目录中，如果在目录中没有找到，可以到 `xos-cpp-sdk/build/.dep/install` 目录下寻找，并复制到 `<install/prefix/path>/lib64` 目录中。