天翼云媒体存储

IOS 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IOS SDK使用指导书

初始化SDK

接入方式

方式一：使用官网下载的SDK

在官网下载SDK，下载地址：IOS SDK

下载完成之后，解压到项目根路径下，配置Podfile

```
target 'CtyunS3Demo' do
  pod 'AWSS3', :path => './oss-ios-sdk'
  pod 'AWSCore', :path => './oss-ios-sdk'
end
```

方式二：直接使用aws的SDK

直接配置Podfile

```
target 'CtyunS3Demo' do
  pod 'AWSS3'
end
```

初始化SDK

注意：直接在客户端上使用主账号存在账号泄露的风险，在客户端上必须使用sts功能生成的临时账号，此初始化流程只能用于测试。如何使用sts初始化参考 本文的安全凭证服务(STS)章节。

引用sdk的头文件

```
#import <AWSS3/AWSS3.h>
```

通过sdk使用s3服务的时候主要需要设置3个配置参数，accessKey，secretKey和endpoint，使用以下方法进行设置，完成sdk的初始化。

```
-(id)init {
    if (self = [super init]) {
        AWSStaticCredentialsProvider *credentialsProvider =
[[AWSStaticCredentialsProvider alloc] initWithAccessKey:@"填入你的accesskey"
secretKey:@"填入你的secretkey"];
        AWSEndpoint *endPoint = [[AWSEndpoint alloc] initWithURLString:@"http://
填入S3的地址和端口"];

        AWSServiceConfiguration *configuration = [[AWSServiceConfiguration
alloc]
```

```
 initWithRegion:AWSRegionUSEast1
                                                    endpoint:endPoint

 credentialsProvider:credentialsProvider];
        [AWSServiceManager defaultServiceManager].defaultServiceConfiguration =
configuration;

        self.s3 = [AWSS3 defaultS3];
    }

    return self;
}
```

参数	说明
accessKey	用户账号 access key
secretKey	用户账号 secret key
endpoint	天翼云资源池的地址，必须指定http或https前缀

桶相关接口

创建桶

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本文介绍如何创建桶（Bucket）。

接口定义

```
- (void)listBuckets:(AWSRequest *)request
     completionHandler:(void (^)(AWSS3ListBucketsOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名称
createBucketConfiguration	AWSS3CreateBucketConfiguration	如果非NULL，则是用于授权签名的AWS区域
ACL	AWSS3BucketCannedACL	设定的权限

代码示例

```
- (void) createBucketWithName:(NSString*) bucketName {
    AWSS3CreateBucketRequest *request = [[AWSS3CreateBucketRequest alloc] init];
    request.bucket = bucketName;
    [self.s3 createBucket:request completionHandler:^(AWSS3CreateBucketOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

删除桶

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本文介绍如何删除桶（Bucket）。

注意：待删除的bucket必须是空的，否则会报错。

接口：

```
- (void)deleteBucket:(AWSS3DeleteBucketRequest *)request
     completionHandler:(void (^)(NSError *error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	要删除的bucket名

代码示例

```
- (void) deleteBucketWithName:(NSString*) bucketName {
    AWSS3DeleteBucketRequest *request = [[AWSS3DeleteBucketRequest alloc] init];
    request.bucket = bucketName;
    [self.s3 deleteBucket:request completionHandler:^(NSError * _Nullable error)
{
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

判断桶是否存在

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本文介绍如何判断桶（Bucket）是否存在。

接口定义

```
- (void)headBucket:(AWSS3HeadBucketRequest *)request
     completionHandler:(void (^)(NSError *error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名

代码示例

```
- (void) headBucketWithName:(NSString*) bucketName {
    AWSS3HeadBucketRequest *request = [[AWSS3HeadBucketRequest alloc] init];
    request.bucket = bucketName;
    [self.s3 headBucket:request completionHandler:^(NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

获取桶列表

Bucket是用于存储对象（Object）的容器，所有的对象都必须隶属于某个Bucket。本文介绍如何获取桶（Bucket）列表。

接口定义

```
- (void)listBuckets:(AWSRequest *)request
     completionHandler:(void (^)(AWSS3ListBucketsOutput *response, NSError
*error))completionHandler
```

代码示例

```
- (void) listBuckets {
    AWSRequest *request = [[AWSRequest alloc] init];
    [self.s3 listBuckets:request completionHandler:^(AWSS3ListBucketsOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

对象相关接口

上传对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何上传对象。

接口定义

```
- (void)putObject:(AWSS3PutObjectRequest *)request
     completionHandler:(void (^)(AWSS3PutObjectOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	将要上传的对象的文件名
body	id （NSString|NSData）	要上传的内容
contentLength	NSNumber	必填，上传内容的长度
ACL	AWSS3ObjectCannedACL	可选，文件控制权限，如AWSS3ObjectCannedACLPublicRead
contentType	NSString	可选，文件类型，如image/jpeg

代码示例

```
- (void) putObjectWithBucket:(NSString*) bucketName key:(NSString*)keyName {
    NSString * body = @"This is a test file";

    AWSS3PutObjectRequest *request = [[AWSS3PutObjectRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    request.body = body;
    request.contentLength = [NSNumber numberWithInteger:body.length];
    request.ACL = AWSS3ObjectCannedACLPublicRead;
    [self.s3 putObject:request completionHandler:^(AWSS3PutObjectOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
        NSLog(@"success: %@", response.ETag);
    }];
}
```

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

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何下载对象。

接口定义

```
- (void)getObject:(AWSS3GetObjectRequest *)request
     completionHandler:(void (^)(AWSS3GetObjectOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	对象名称
range	NSString	下载区间，参考 https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35

代码示例

```
- (void) getObjectWithBucket:(NSString*) bucketName
                         key:(NSString*)keyName {
    AWSS3GetObjectRequest *request = [[AWSS3GetObjectRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    [self.s3 getObject:request completionHandler:^(AWSS3GetObjectOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

删除对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何删除对象。

接口定义

```
- (void)deleteObject:(AWSS3DeleteObjectRequest *)request
     completionHandler:(void (^)(AWSS3DeleteObjectOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	包含bucket及相关的请求参数
key	NSString	要删除的对象名称

代码示例

```
 - (void) deleteObjectWithBucket:(NSString*) bucketName
                              key:(NSString*)keyName {
    AWSS3DeleteObjectRequest *request = [[AWSS3DeleteObjectRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    [self.s3 deleteObject:request completionHandler:^(AWSS3DeleteObjectOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

复制对象

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何复制对象。

接口定义

```
- (void)replicateObject:(AWSS3ReplicateObjectRequest *)request
     completionHandler:(void (^)(AWSS3ReplicateObjectOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	目的bucket
key	NSString	目的对象名
replicateSource	NSString	源bucket和对象名，使用/分割

代码示例

```
- (void) copyObjectWithBucket:(NSString*)bucketName
                          key:(NSString*)keyName
                 sourceBucket:(NSString*)sourceBucketName
                    sourceKey:(NSString*)sourceKey {
    AWSS3ReplicateObjectRequest *request = [[AWSS3ReplicateObjectRequest alloc]
init];
    request.bucket = bucketName;
    request.key = keyName;
    request.replicateSource = [NSString stringWithFormat:@"%@/%@",
sourceBucketName, sourceKey];
    [self.s3 replicateObject:request
completionHandler:^(AWSS3ReplicateObjectOutput * _Nullable response, NSError *
_Nullable error) {
        if (error != nil) {
```

```
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

获取对象元数据

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何获取对象元数据。

接口定义

```
- (void)headObject:(AWSS3HeadObjectRequest *)request
     completionHandler:(void (^)(AWSS3HeadObjectOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	对象名称

代码示例

```
- (void) headObjectWithBucket:(NSString*) bucketName
                          key:(NSString*)keyName {
    AWSS3HeadObjectRequest *request = [[AWSS3HeadObjectRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    [self.s3 headObject:request completionHandler:^(AWSS3HeadObjectOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

获取对象列表

对象是存储数据的基本单元。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成。对象由桶内部唯一的Key来标识。本文介绍如何获取对象列表。

接口定义

```
- (void)listObjects:(AWSS3ListObjectsRequest *)request
     completionHandler:(void (^)(AWSS3ListObjectsOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
prefix	NSString	如果非NULL，则仅列举以指定的prefix作为前缀的对象
marker	NSString	如果非NULL，指定一个标识符，在列举桶内对象列表时，返回的对象列表将仅是按照字典顺序排序后位于这个标识符之后的对象
delimiter	NSString	如果非NULL，则是用来对桶内对象进行分组的字符串。所有名称包含指定的前缀且第一次出现delimiter字符之间的对象将作为一组元素，在返回信息的CommonPrefixes节点显示
maxkeys	NSNumber	指定返回对象的最大数量，若为0则列举所有对象

代码示例

```
- (void) listObjectsWithBucket:(NSString*) bucketName {
    AWSS3ListObjectsRequest *request = [[AWSS3ListObjectsRequest alloc] init];
    request.bucket = bucketName;
    [self.s3 listObjects:request completionHandler:^(AWSS3ListObjectsOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
    }];
}
```

设置对象访问权限

对象默认的权限是拥有者私有权限，只能由拥有者进行访问；系统提供公共读和公共读写等权限供用户选择。用户可以在上传的时候设置对象的访问权限，也可以通过putObjectAcl接口修改对象的访问权限。

接口定义

```
- (void)putObjectAcl:(AWSS3PutObjectAclRequest *)request
    completionHandler:(void (^ _Nullable)(AWSS3PutObjectAclOutput * _Nullable
response, NSError * _Nullable error))completionHandler;
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	将要上传的对象的文件名
ACL	AWSS3ObjectCannedACL	文件控制权限，如AWSS3ObjectCannedACLPublicRead

代码示例

```
- (void) putObjectAcl:(NSString*) bucketName key:(NSString*)keyName {
    AWSS3PutObjectAclRequest *request = [[AWSS3PutObjectAclRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    request.ACL = AWSS3ObjectCannedACLPublicRead;
    [self.s3 putObjectAcl:request completionHandler:^(AWSS3PutObjectAclOutput *
_Nullable response, NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
        NSLog(@"success");
    }];
}
```

分片上传接口

融合接口

SDK提供封装好的融合接口，方便用户实现分片上传的功能。

接口定义

```
- (AWSTask<AWSS3TransferUtilityUploadTask *> *)uploadData:(NSData *)data
                                                      key:(NSString *)key
                                              contentType:(NSString
*)contentType
                                               expression:(nullable
AWSS3TransferUtilityUploadExpression *)expression
                                        completionHandler:(nullable
AWSS3TransferUtilityUploadCompletionHandlerBlock)completionHandler;

- (AWSTask<AWSS3TransferUtilityUploadTask *> *)uploadData:(NSData *)data
                                                   bucket:(NSString *)bucket
                                                      key:(NSString *)key
                                              contentType:(NSString
*)contentType
                                               expression:(nullable
AWSS3TransferUtilityUploadExpression *)expression
                                        completionHandler:(nullable
AWSS3TransferUtilityUploadCompletionHandlerBlock)completionHandler;
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	要上传的对象名称
data	NSData	文件内容
contentType	NSString	文件类型
expression	AWSS3TransferUtilityUploadExpression	其他参数，设置progress回调，设置acl
completionHandler	AWSS3TransferUtilityUploadCompletionHandlerBlock	上传文件结果回调

代码示例

```
- (IBAction)start:(id)sender {
    __weak UploadViewController *weakSelf = self;

    NSString *fileName = @"test40M.mp4";
    NSString *path = [[NSBundle mainBundle] pathForResource:fileName
ofType:nil];
    NSData *fileData = [NSData dataWithContentsOfFile:path
options:NSDataReadingMappedIfSafe error:nil];

    AWSS3TransferUtilityUploadExpression * expression =
[AWSS3TransferUtilityUploadExpression new];
    //设置公共读
    //[expression setValue:@"public-read" forRequestHeader:@"x-amz-acl"];
    expression.progressBlock = ^(AWSS3TransferUtilityTask * _Nonnull task,
NSProgress * _Nonnull progress) {
        dispatch_async(dispatch_get_main_queue(), ^{
            weakSelf.progressView.progress = progress.fractionCompleted;
        });
    };

    AWSS3TransferUtilityUploadCompletionHandlerBlock completionHandler =
^(AWSS3TransferUtilityUploadTask *task, NSError *error) {
        dispatch_async(dispatch_get_main_queue(), ^{
            if(error) {
                weakSelf.statusLabel.text = @"Failed to Upload";
            } else {
                weakSelf.statusLabel.text = @"Successfully Uploaded";
            }
        });
    };

    AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility
defaultS3TransferUtility];
    [[transferUtility uploadData:fileData
                          bucket:self.mBucketName
                             key:fileName
                     contentType:@"video/mp4"
                      expression:expression
                completionHandler:completionHandler]
continueWithBlock:^id(AWSTask *task) {
```

```
        if (task.error) {
            NSLog(@"Error: %@", task.error);
            return nil;
        }

        if (task.result) {
            NSLog(@"Upload Starting!");
        }
        return nil;
    }];
}
```

标准接口

标准接口由以下接口组合完成分片上传文件的功能，createMultipartUpload，uploadPart，completeMultipartUpload，abortMultipartUpload。

创建分片上传任务

创建分片上传任务，返回分片上传任务的ID。

接口定义

```
- (AWSTask<AWSS3CreateMultipartUploadOutput *> *)createMultipartUpload:
(AWSS3CreateMultipartUploadRequest *)request

- (void)createMultipartUpload:(AWSS3CreateMultipartUploadRequest *)request
     completionHandler:(void (^)(AWSS3CreateMultipartUploadOutput *response,
NSError *error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	要上传的对象名
ACL	AWSS3ObjectCannedACL	权限设置

代码示例

```
AWSS3CreateMultipartUploadRequest *createReq =
[AWSS3CreateMultipartUploadRequest new];
createReq.key = self.fileName;
createReq.bucket = self.bucketName;
__weak CTYunStorageTask *weakSelf = self;

[self.requestArray addObject:createReq];

[[[AWSS3 defaultS3] createMultipartUpload:createReq] continueWithBlock:^id
_Nullable(AWSTask<AWSS3CreateMultipartUploadOutput *> * _Nonnull t)
{
    dispatch_sync(weakSelf.queue, ^{
```

```
        LOGI(@"createMultipartUpload success");
        AWSS3CreateMultipartUploadOutput *outPut = t.result;
        weakSelf.uploadId = outPut.uploadId;
        [weakSelf.userDefault setObject:weakSelf.uploadId forKey:@"uploadId"];
        [weakSelf uploadPartData:progressBlock success:completion
failure:failure];
    });

    return nil;
}];
```

上传一个分片

获取到分片任务ID之后，通过ID来上传分片内容到S3服务器。

接口定义

```
- (AWSTask<AWSS3UploadPartOutput *> *)uploadPart:(AWSS3UploadPartRequest
*)request

- (void)uploadPart:(AWSS3UploadPartRequest *)request
     completionHandler:(void (^)(AWSS3UploadPartOutput *response, NSError
*error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	要上传的对象名称
body	id	上传的文件内容（NSData）
uploadId	NSString	上传任务ID
contentLength	NSNumber	上传内容长度
partNumber	NSNumber	分片ID（1-10000）
uploadProgress	AWSNetworkingUploadProgressBlock	上传进度回调接口

代码示例

```
AWSS3UploadPartRequest *uploadPartRequest = [AWSS3UploadPartRequest new];
uploadPartRequest.key = self.fileName;
uploadPartRequest.partNumber = @(i);
uploadPartRequest.body = partData;
uploadPartRequest.contentLength = @(dataLength);
uploadPartRequest.uploadId = self.uploadId;
uploadPartRequest.bucket = self.bucketName;
uploadPartRequest.uploadProgress = ^(int64_t bytesSent, int64_t totalBytesSent,
int64_t totalBytesExpectedToSend){
```

```
    dispatch_async(dispatch_get_main_queue(), ^{
        weakSelf.uploadByteSent += bytesSent;
        int64_t fileSize = [self.fileData length];
        if (weakSelf.uploadByteSent < fileSize){
            weakSelf.progress = weakSelf.uploadByteSent/(fileSize*1.0);
            progressBlock(weakSelf.progress);
        }
    });
};
[self.requestArray addObject:uploadPartRequest];

[[[AWSS3 defaultS3] uploadPart:uploadPartRequest] continueWithBlock:^id
_Nullable(AWSTask<AWSS3UploadPartOutput *> * _Nonnull t)
{
    LOGI(@"uploadPartData finished");
    if (t.error){
        weakSelf.isError = YES;
    } else {
        AWSS3UploadPartOutput *outputPart = t.result;
        if (outputPart){
            AWSS3CompletedPart *partObj = [AWSS3CompletedPart new];
            partObj.partNumber = @(i);
            partObj.ETag = outputPart.ETag;
            NSDictionary *cacheDic = @{@"part":@(i), @"ETag":outputPart.ETag,
@"size":@(dataLength)};
            [weakSelf.partETags setObject:cacheDic forKey:[NSString
stringWithFormat:@"%d", i]];
            [weakSelf.mutArray addObject:partObj];
            [weakSelf.userDefault setObject:weakSelf.partETags
forKey:@"partArray"];
            [weakSelf.userDefault synchronize];
        } else {
            LOGI(@"uploadPartData error, outputPart=nil");
        }
    }
    dispatch_group_leave(group);
    return nil;
}];
```

完成分片上传任务

完成所有分片的上传之后，调用完成接口，服务端会把所有分片合并成对象保存。

接口定义

```
- (AWSTask<AWSS3CompleteMultipartUploadOutput *> *)completeMultipartUpload:
(AWSS3CompleteMultipartUploadRequest *)request

- (void)completeMultipartUpload:(AWSS3CompleteMultipartUploadRequest *)request
     completionHandler:(void (^)(AWSS3CompleteMultipartUploadOutput *response,
NSError *error))completionHandler
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	要上传的对象名
uploadId	NSString	上传任务ID
multipartUpload	AWSS3CompletedMultipartUpload	上传的分片信息列表

代码示例

```
- (void)uploadComplete:(void (^)(float progress))progressBlock
               success:(void (^)(void))completion
               failure:(void (^)(NSError *))failure
{
    LOGI(@"uploadComplete send request");
    AWSS3 *transferManager = [AWSS3 defaultS3];
    AWSS3CompleteMultipartUploadRequest *completeUpload =
[AWSS3CompleteMultipartUploadRequest new];
    completeUpload.key = self.fileName;
    completeUpload.uploadId = self.uploadId;
    completeUpload.bucket = self.bucketName;
    completeUpload.multipartUpload = [AWSS3CompletedMultipartUpload new];
    completeUpload.multipartUpload.parts = [NSArray
arrayWithArray:self.mutArray] ;
    [self.requestArray addObject:completeUpload];

    [[transferManager completeMultipartUpload:completeUpload]
continueWithBlock:^id _Nullable(AWSTask<AWSS3CompleteMultipartUploadOutput *> *
_Nonnull t) {
        LOGI(@"uploadComplete recv response");
        if (t.error) {
            failure(t.error);
        } else {
            LOGI(@"uploadComplete success!");
            self.progress = 1;
            progressBlock(self.progress);
            completion();
        }
        return nil;
    }];
}
```

终止分片上传任务

上传失败的时候调用此接口，服务器会清除残留的分片数据。

接口定义

```
- (AWSTask<AWSS3AbortMultipartUploadOutput *> *)abortMultipartUpload:
(AWSS3AbortMultipartUploadRequest *)request;

- (void)abortMultipartUpload:(AWSS3AbortMultipartUploadRequest *)request
completionHandler:(void (^ _Nullable)(AWSS3AbortMultipartUploadOutput *
_Nullable response, NSError * _Nullable error))completionHandler;
```

参数说明

参数名	类型	说明
bucket	NSString	bucket名
key	NSString	要上传的对象名
uploadId	NSString	上传任务ID

代码示例

```
- (void) abortMultipartUpload:(NSString*) bucketName key:(NSString*)keyName
uploadId:(NSString*)uploadId {
    AWSS3AbortMultipartUploadRequest *request =
[[AWSS3AbortMultipartUploadRequest alloc] init];
    request.bucket = bucketName;
    request.key = keyName;
    request.uploadId = uploadId;
    [self.s3 abortMultipartUpload:request
completionHandler:^(AWSS3AbortMultipartUploadOutput * _Nullable response,
NSError * _Nullable error) {
        if (error != nil) {
            NSLog(@"error: %@", error);
            return;
        }
        NSLog(@"success");
    }];
}
```

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。

获取临时token

在服务端生成临时token，可参考java、python、nodejs、CPP、donet、go、php SDK说明，请从SDK概览 页面选择进入对应的开发指南查阅。

使用临时token

实现一个MyCredentialsProvider，支持更新ak/sk和token。

```
```

```
// .h
@interface MyCredentialsProvider: NSObject <AWSCredentialsProvider>

- (instancetype)initWithAccessKey:(NSString *)accessKey
                        secretKey:(NSString *)secretKey
                     sessionToken:(NSString *)sessionToken;

- (void)updateCredWithAccessKey:(NSString *)accessKey
                      secretKey:(NSString *)secretKey
                   sessionToken:(NSString *)sessionToken;
@end


// .m
@interface MyCredentialsProvider()
@property (atomic, strong) AWSCredentials *internalCredentials;
@end

@implementation MyCredentialsProvider
- (instancetype)initWithAccessKey:(NSString *)accessKey
                        secretKey:(NSString *)secretKey
                     sessionToken:(NSString *)sessionToken {
    if (self = [super init]) {
        _internalCredentials = [[AWSCredentials alloc]
initWithAccessKey:accessKey

secretKey:secretKey

 sessionKey:sessionToken
                                                              expiration:nil];
    }
    return self;
}

- (AWSTask<AWSCredentials *> *)credentials {
    return [AWSTask taskWithResult:self.internalCredentials];
}

- (void)invalidateCachedTemporaryCredentials {
}

- (void)updateCredWithAccessKey:(NSString *)accessKey
                      secretKey:(NSString *)secretKey
                   sessionToken:(NSString *)sessionToken {
    self.internalCredentials = [[AWSCredentials alloc]
initWithAccessKey:accessKey
                                                                secretKey:secretKey

 sessionKey:sessionToken
                                                             expiration:nil];
}
@end
```

使用临时token初始化sdk

```
#define ACCESS_KEY @"<your-access-key>"
#define SECRET_KEY @"<your-secret-key>"
```

```
#define ENDPOINT @"<your-endpoint>"
#define SESSION_TOKEN @"<your-session-token>"

-(id)initWithToken {
    if (self = [super init]) {
        self.credProvider = [[MyCredentialsProvider alloc]
initWithAccessKey:ACCESS_KEY secretKey:SECRET_KEY sessionToken:SESSION_TOKEN];

        AWSEndpoint *endPoint = [[AWSEndpoint alloc]
initWithURLString:ENDPOINT];

        AWSServiceConfiguration *configuration = [[AWSServiceConfiguration
alloc]

 initWithRegion:AWSRegionUSEast1
                                                    endpoint:endPoint

 credentialsProvider:self.credProvider];
        [AWSServiceManager defaultServiceManager].defaultServiceConfiguration =
configuration;

        self.s3 = [AWSS3 defaultS3];
    }

    return self;
}
```