移动开发平台

移动存储 (Storage)

产品文档

腾讯云

文档目录

移动存储 (Storage)
产品介绍

最近更新时间：2018-03-28 17:47:05

简介

Storage 是 MobileLine 提供的一项功能强大、操作简单且经济实惠的对象存储服务，可以快速轻松的帮助您进行专业的数据存储和处理，如将用户的照片和视频存储到网络上，同时，Storage 具有高扩展性、低成本、可靠和安全等特点，可以满足您的各种需求。

功能

功能	描述
稳健的操作性	Storage 支持文件上传和下载操作，当请求中断时，用户可以从之前停止的地方重新开始，从而为您的用户节省时间和带宽。
可靠的安全性	Storage 所有的文件操作都必须经过签名，只有文件的拥有者或者已授权对象才能进行访问。
高可扩展性	Storage 非常适合大量数据的存储，当您的应用大受欢迎、数据呈爆发式增长的情况，可提供 EB 级的容量规模。

优势

优势	描述
稳定持久	提供数据跨多架构、多设备冗余存储，为用户数据提供异地容灾和资源隔离功能，实现高达 99.999999999% 的数据持久性。
安全可靠	结合腾讯的攻击防御系统，能够有效抵御 DDoS 攻击、CC 攻击，保障您的业务正常运行。
成本最优	Storage 支持按需按量使用，您无需预先支付任何预留存储空间的费用，同时您无需任何运维和托管成本。

客户案例

bilibili、知乎、猎豹。

Android 文档
Android 快速入门

最近更新时间：2018-08-16 16:27:49

准备工作

- 首先您需要一个 Android 工程，这个工程可以是您现有的工程，也可以是您新建的工程。
- 其次您需要一个在后台搭建一个授权服务器，为 SDK 提供临时密钥，请参考 用户访问控制。

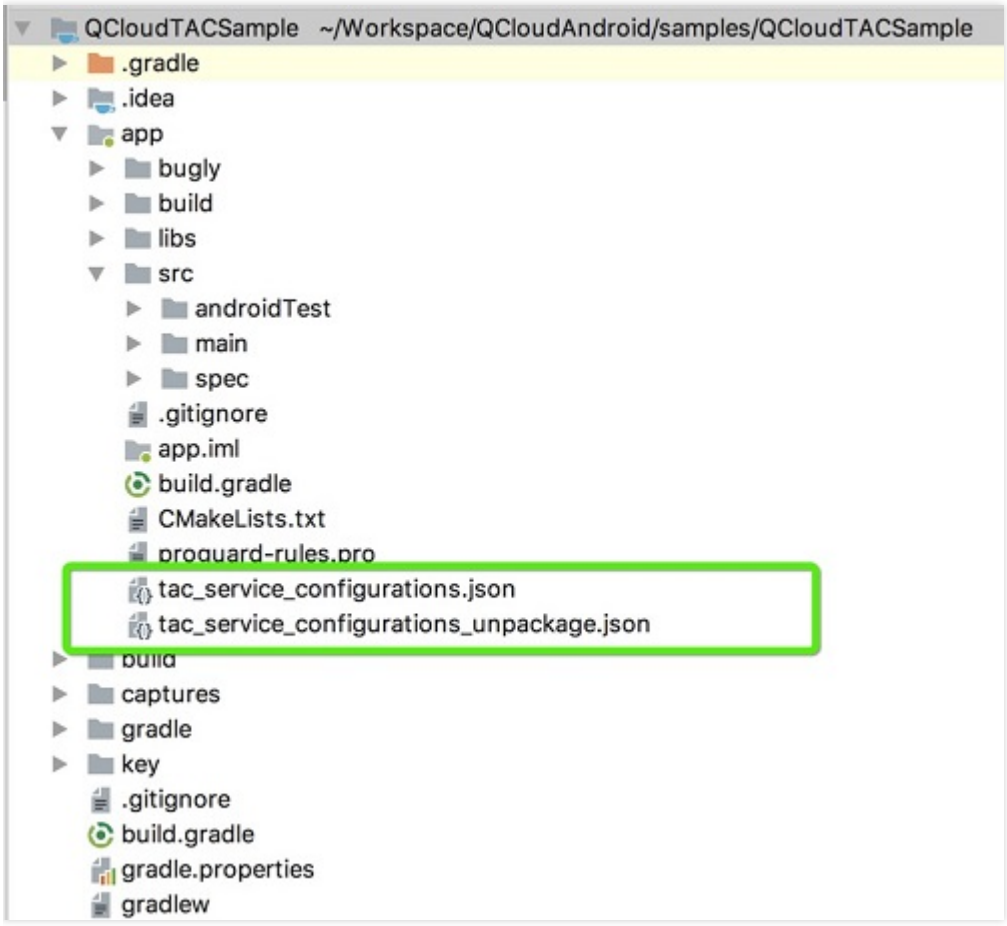
第一步：创建项目和应用（已完成请跳过）

在使用我们的服务前，您必须先在 MobileLine 控制台上 创建项目和应用。

第二步：添加配置文件（已完成请跳过）

在您创建好的应用上单击【下载配置】按钮来下载该应用的配置文件的压缩包：

解压该压缩包，您会得到 tac_service_configurations.json 和 tac_service_configurations_unpackage.json 两个文件，请您如图所示添加到您自己的工程中去。



注意：
请您按照图示来添加配置文件，`tac_service_configurations_unpackage.json` 文件中包含了敏感信息，请不要打包到 apk 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

您需要在工程级 build.gradle 文件中添加 SDK 插件的依赖：

```
buildscript {
...
dependencies {
classpath 'com.android.tools.build:gradle:3.0.1'
// 添加这行
classpath 'com.tencent.tac:tac-services-plugin:1.3.+'
}
}
```

在您应用级 build.gradle 文件（通常是 app/build.gradle）中添加 Storage 服务依赖，并使用插件：

```
dependencies {
// 增加这行
compile 'com.tencent.tac:tac-core:1.3.+'
compile 'com.tencent.tac:tac-storage:1.3.+'
}
...

// 在文件最后使用插件
apply plugin: 'com.tencent.tac.services'
```

配置使用权限

Storage SDK 需要一个后台授权服务器提供临时密钥，才能正常工作。关于如何在 SDK 里配置服务器接口，请参见 Android 配置授权服务器。

> **注意：**
> 请先完成配置，再调用 Storage 服务的任何功能。否则，我们的服务无法识别您的身份。

到此您已经成功接入了 MobileLine 移动存储服务。

Proguard 配置

如果您的代码开启了混淆，为了 SDK 可以正常工作，请在 `proguard-rules.pro` 文件中添加如下配置：

```
# MobileLine Core

-keep class com.tencent.qcloud.core.** { *;}
-keep class bolts.** { *;}
-keep class com.tencent.tac.** { *;}
-keep class com.tencent.stat.*{*;}
-keep class com.tencent.mid.*{*;}
-dontwarn okhttp3.**
-dontwarn okio.**
-dontwarn javax.annotation.**
-dontwarn org.conscrypt.**
```

后续步骤

您可以通过策略精确控制您数据的访问权限，可以参考 数据安全性最佳实践。

Android 手动集成

最近更新时间：2018-07-10 11:59:45

如果您无法通过 gradle 远程依赖的方式来集成 SDK，我们提供了手动的方式来集成服务：

1. 下载服务资源压缩包。

- 下载 移动开发平台（MobileLine）核心框架资源包，并解压。
- 下载 移动开发平台（MobileLine） Storage 资源包，并解压。

2. 修改您工程的 AndroidManifest.xml 文件

如果您不是通过 aar 集成，请按照下面的示例代码修改您工程下的 AndroidManifest.xml 文件。

```
<!-- 添加 Analytics 需要的权限 -->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>

<application>

...

<!-- 添加 Analytics 的 provider -->
<provider
android:name="com.tencent.mid.api.MidProvider"
android:authorities="你的包名.TENCENT.MID.V3"
android:exported="true" >
</provider>

</application>
```

Android 配置授权服务器

最近更新时间：2018-06-20 14:35:37

使用 Storage 服务时，后台需要对您的身份进行校验，校验过程是通过调用接口时携带签名实现的。因此，Storage SDK 需要提前设置临时密钥才能正常的访问数据。临时密钥有一定的有效期，过期后自动失效。由于临时密钥需要永久密钥生成，而永久密钥放在客户端中有极大的泄露风险，因此建议通过后台生成临时密钥，并下发到客户端中。

假设您已经按照 快速搭建后台授权服务 搭好了授权服务器，服务器的请求地址如下：

```
GET https://<SERVER_HOST><PATH>?<name>=<value>
Header: <header1>=<value1>
```

我们的 SDK 会在本地临时密钥不存在或者过期的时候，自动请求您设置的服务器接口，拿到新的临时密钥。请通过 `TACStorageOptions` 设置服务器接口地址。

SDK 设置授权服务器地址

设置方式取决于服务器返回的 JSON 数据的格式。

标准格式

JSON 是标准的临时密钥格式，即

```
{
"code":0,"message":"","codeDesc":"Success",
"data":
{
"credentials":
{
"sessionToken":"42f8151428b3960b1226f421b8f271c6242ad02c3",
"tmpSecretId":"AKIDtd9QSGWBIDuMaYFp57tSmrhJgohLtvpT",
"tmpSecretKey":"ZfV5PVLvFLCvPefPt76qKYXIo56tSmrg"
},
"expiredTime":1508400619
}
}
```

那么，您可以以下方式设置，不需要手动解析 JSON：

```
TACApplicationOptions applicationOptions = TACApplication.options();
TACStorageOptions storageOptions = applicationOptions.sub("storage");

// 配置授权服务器接口
storageOptions.setCredentialProvider(new HttpRequest.Builder<String>()
.scheme("https")
.host("<SERVER_HOST>")
.path("<PATH>")
.method("GET")
.query("<name>", "<value>")
.addHeader("<header1>", "<value1>")
.build());
```

自定义格式

JSON 是自定义格式，您可以通过如下代码配置服务器接口和响应处理类，其中处理类必须继承于 `SessionCredentialProvider`：

```
TACApplicationOptions applicationOptions = TACApplication.options();
TACStorageOptions storageOptions = applicationOptions.sub("storage");

// 配置自定义的响应处理类，MySessionCredentialProvider 继承于 SessionCredentialProvider
// 在 MySessionCredentialProvider 中设置密钥获取接口
storageOptions.setCredentialProvider(new MySessionCredentialProvider(new HttpRequest.Builder<St
ring>()
.scheme("https")
.host("<SERVER_HOST>")
.path("<PATH>")
.method("GET")
.query("<name>", "<value>")
.addHeader("<header1>", "<value1>")
.build()));
```

在响应处理类中，实现 `onRemoteCredentialReceived` 方法，返回一个 `SessionQCloudCredentials` 实例：

```
public class MySessionCredentialProvider extends SessionCredentialProvider {
public MySessionCredentialProvider(HttpRequest<String> httpRequest) {
super(httpRequest);
}

@Override
protected QCloudLifecycleCredentials onRemoteCredentialReceived(String jsonContent) throws QC
loudClientException {
// 在这里处理 jsonContent，并返回一个有效的密钥实例
```

```
...
return new SessionQCloudCredentials(secretId, secretKey, sessionToken, expiredTime);
}
}
```

实例化 `SessionQCloudCredentials` 需要的几个参数说明如下：

- secretId: 临时密钥 secret id
- secretKey: 临时密钥 secret key
- sessionToken: 临时密钥 token
- expiredTime: 密钥过期时间

上传文件

最近更新时间：2018-05-16 17:37:44

创建引用

要将文件上传到 Storage，首先要创建对文件的完整路径（包括文件名）的引用：

```
TACStorageService storage = TACStorageService.getInstance();
TACStorageReference reference = storage.referenceWithPath('images/imageA.jpg');
```

上传文件

您可以将内存中的数据，本地文件路径，或者是数据流传输到远程 Storage 中。

```
StorageReference storageRef = storage.referenceWithPath('images/imageA.jpg');

// 通过内存中的数据上传文件
byte[] tmpData = new byte[200];
reference.putData(tmpData, null));

// 上传本地文件
File localFile = new File("path/to/file");
reference.putFile(Uri.fromFile(localFile), null);

// 上传数据流
InputStream stream = new FileInputStream(new File("path/to/file"));
reference.putStream(stream, null);
```

添加文件元数据

在上传文件时，您还可以带上元数据。此元数据包含常见的文件元数据属性，如 name、size 和 contentType（通常称为 MIME 类型）。

```
StorageReference storageRef = storage.referenceWithPath('images/imageA.jpg');
TACStorageMetadata metadata = new TACStorageMetadata.Builder().contentLanguage("CN").build
();

// 通过内存中的数据上传文件，并带上metadata
reference.putData(tmpData, metadata));

// 上传本地文件，并带上metadata
reference.putFile(Uri.fromFile(localFile), metadata);
```

```
// 上传数据流，并带上metadata
reference.putStream(stream, metadata);
```

管理上传

如果您想要管理上传的行为，可以调用 pause 和 resume。

> **注意：**
> pause 和 resume 只针对大文件的上传有效。

```
TACStorageUploadTask uploadTask = reference.putData(tmpData, TACMetadata));

// 暂停任务
uploadTask.pause();

// 继续任务
uploadTask.resume();
```

通过重新启动进程继续上传

对于本地的大文件上传，我们支持断点续传，您可以在本地记录上传的 uploadId，在下次 App 启动的时候从上次停止的地方上传，而不会重头开始，节省您的带宽和时间。

```
TACStorageReference reference = tacStorageService.referenceWithPath("/tac_test/multipart");
File uploadFile = createFile(10 * 1024 * 1024);
TACStorageUploadTask uploadTask = reference.putFile(Uri.fromFile(uploadFile), null)
.addProgressListener(new StorageProgressListener<TACStorageTaskSnapshot>() {
@Override
public void onProgress(TACStorageTaskSnapshot snapshot) {
uploadId = snapshot.getUploadId();
}
});
```

您可以在 uploadId 存放在您本地的 sharedPreference，下次启动后，您可以继续上传：

```
// 传入上次的 uploadId，将会从之前断开的地方继续上传
TACStorageReference reference = tacStorageService.referenceWithPath("/tac_test/multipart");
reference.putFile(Uri.fromFile(uploadFile), null, uploadId);
```

添加任务结果监听

您可以调用 TACStorageTask 的 addResultListener 方法来监听任务结果：

```
TACStorageUploadTask uploadTask = reference.putData(tmpData, TACMetadata));
uploadTask.addResultListener(new StorageResultListener<TACStorageTaskSnapshot>() {
@Override
public void onSuccess(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传成功
}
});
}

@Override
public void onFailure(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传失败
Exception exception = snapshot.getError();
}
});
}
```

添加上传任务进度监听

您可以使用 addProgressListener 方法可以监听数据的进度：

```
TACStorageUploadTask uploadTask = reference.putData(tmpData, TACMetadata));
uploadTask.addProgressListener(new StorageProgressListener<TACStorageTaskSnapshot>() {
@Override
public void onProgress(TACStorageTaskSnapshot snapshot) {
Log.i("QCloudStorage", "progress = " + snapshot.getBytesTransferred() + "," +
snapshot.getTotalByteCount());
}
});
```

取消任务

您可以调用 cancel 方法取消任务。

注意:
根据任务当时的运行进度，**取消指令不一定能成功。**

```
TACStorageReference reference = tacStorageService.referenceWithPath("/tac_test/multipart3");
File uploadFile = createFile(10 * 1024 * 1024);
TACStorageUploadTask uploadTask = reference.putFile(Uri.fromFile(uploadFile), null);

uploadTask.cancel();
```

下载文件

最近更新时间：2018-05-16 17:39:15

创建引用

要将 Storage 上的文件下载到本地，首先要创建对文件的完整路径（包括文件名）的引用：

```
TACStorageService storage = TACStorageService.getInstance();
TACStorageReference reference = storage.referenceWithPath('images/imageA.jpg');
```

下载文件

您可以调用 downloadToFile 方法，将引用指向的文件下载到本地：

```
TACStorageReference reference = storage.referenceWithPath('images/imageA.jpg');

Uri fileUri = Uri.fromFile(new File(getExternalCacheDir() + File.separator + "local_tmp"));
reference.downloadToFile(fileUri);
```

添加任务结果监听

您可以调用 TACStorageDownloadTask 的 addResultListener 方法来监听任务结果：

```
TACStorageDownloadTask downloadTask = reference.downloadToFile(fileUri);
downloadTask.addResultListener(new StorageResultListener<TACStorageTaskSnapshot>() {
@Override
public void onSuccess(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传成功
}
});
}

@Override
public void onFailure(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传失败
```

```
Exception exception = snapshot.getError();
}
});
}
```

添加任务进度监听

您可以使用 addProgressListener 方法可以监听数据的进度：

```
TACStorageDownloadTask downloadTask = reference.downloadToFile(fileUri);
downloadTask.addProgressListener(new StorageProgressListener<TACStorageTaskSnapshot>() {
@Override
public void onProgress(TACStorageTaskSnapshot snapshot) {
Log.i("QCloudStorage", "progress = " + snapshot.getBytesTransferred() + "," +
snapshot.getTotalByteCount());
}
});
```

删除文件

最近更新时间：2018-05-16 17:40:17

创建引用

要删除 Storage 上的文件，首先要创建对文件的完整路径（包括文件名）的引用：

```
TACStorageService storage = TACStorageService.getInstance();
TACStorageReference reference = storage.referenceWithPath('images/imageA.jpg');
```

删除文件

您可以调用 delete 方法，删除一个远程文件：

```
TACStorageReference reference = storage.referenceWithPath('images/imageA.jpg');

reference.delete();
```

添加任务结果监听

您可以调用 TACStorageDeleteTask 的 addResultListener 方法来监听任务结果：

```
TACStorageDeleteTask deleteTask = reference.delete();
deleteTask.addResultListener(new StorageResultListener<TACStorageTaskSnapshot>() {
@Override
public void onSuccess(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传成功
}
});
}

@Override
public void onFailure(final TACStorageTaskSnapshot snapshot) {
showMessage(new Runnable() {
@Override
public void run() {
// 上传失败
Exception exception = snapshot.getError();
}
```

```
});
}
```

文件引用

最近更新时间：2018-05-16 17:59:15

您的文件存储在 腾讯云 COS 存储分区中。此存储分区中的文件以分层结构存储，就像本地硬盘中的文件系统一样。通过创建对文件的引用，您的应用可以获得对相应文件的访问权限。然后，借助所创建的这些引用，您可以上传或下载数据、获取或更新元数据，也可以删除文件。引用可以指向特定的文件，也可以指向层次结构中更高层级的节点。

创建引用

引用可以看作是指向云端文件的指针：要上传、下载和删除的文件或要获取和更新文件的元数据，请创建引用。

```
TACStorageService storage = TACStorageService.getInstance();

TACStorageReference reference = storage.rootReference();
```

上面的代码是获取整个 存储桶 的根目录的引用，如果想要创建某个子文件的引用，可以使用：

```
TACStorageReference reference = storage.referenceWithPath('images');
```

引用导航

引用可以向上或者向下导航：

```
// 获取根节点的引用
TACStorageReference rootRef = reference.root();

// 获得父节点的引用
TACStorageReference parentRef = reference.parent();

// 获取当前引用下某个子节点的引用
TACStorageReference childRef = reference.child('imageA.jpg');
```

引用属性

您可以使用 getPath()、getName() 和 getBucket() 方法检查引用，以便更好地了解它们指向的文件。

```
// 获取文件路径
reference.getPath();

// 获取文件名
reference.getName();
```

```
// 获取文件所在bucket
reference.getBucket();

// 获取bucket所在区域
reference.getRegion();
```

iOS 文档
iOS 快速入门

最近更新时间：2019-03-04 16:07:26

移动开发平台（MobileLine）使用起来非常容易，只需要简单的 4 步，您便可快速接入移动存储服务。接入后，您即可获得我们提供的各项能力，减少您在开发应用时的重复工作，提升开发效率。

准备工作

- 您首先需要一个 iOS 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。
- 其次您需要一个在后台搭建一个授权服务器，为SDK提供临时密钥，请参考 用户访问控制。

第一步：创建项目和应用

在使用我们的服务前，您必须先在 MobileLine 控制台上 创建项目和应用。

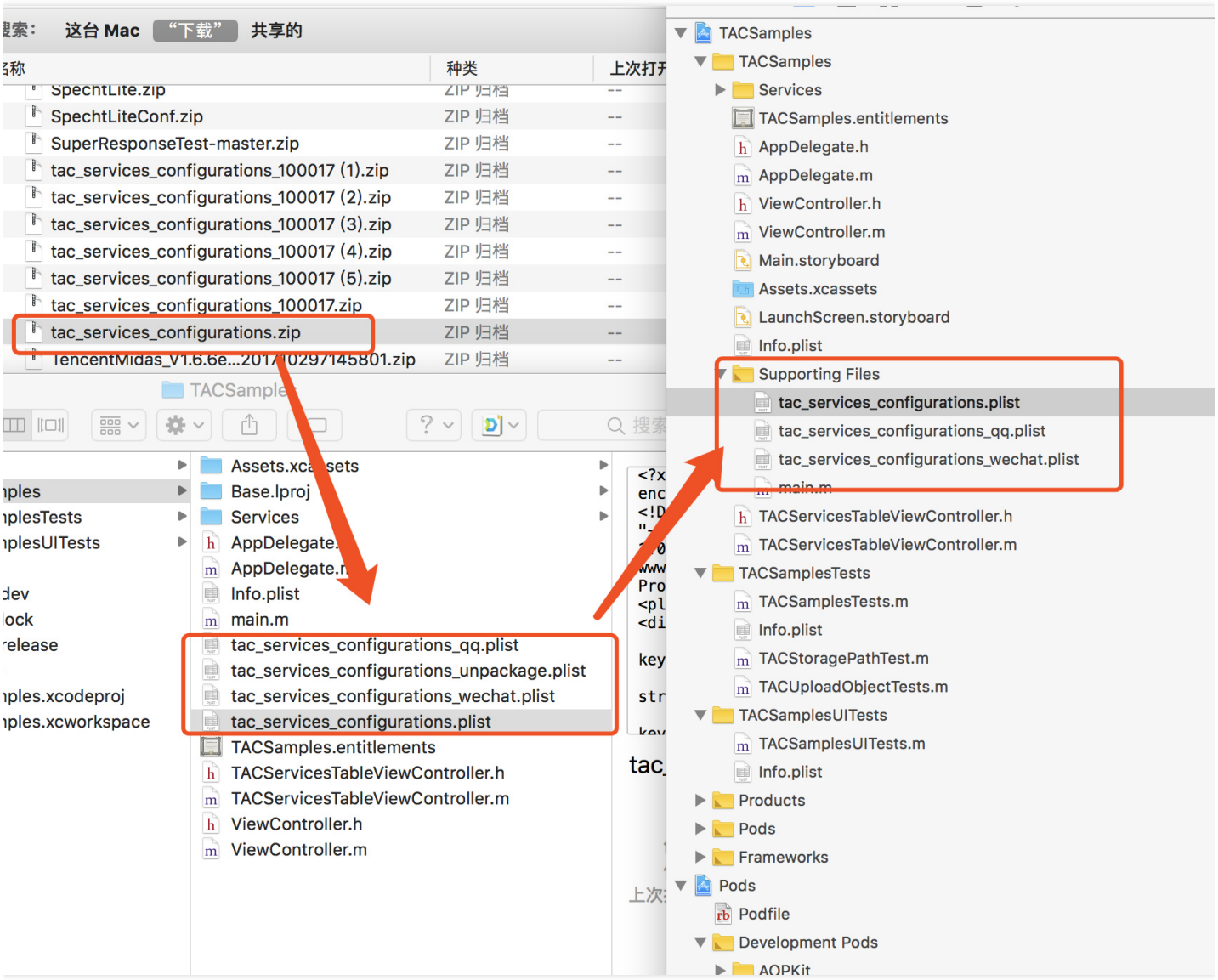
> 如果您已经在 MobileLine 控制台上创建过了项目和应用，请跳过此步。

第二步：添加配置文件

> **注意：**
> 如果您已经添加过配置文件，请跳过此步。

创建好应用后，您可以单击红框中的【下载配置】来下载该应用的配置文件的压缩包：

解压后将 tac_services_configurations.plist 文件集成进项目中。其中有一个 tac_services_configurations_unpackage.plist 文件，请将该文件放到您工程的根目录下面(**切记不要将改文件添加进工程中**)。 添加好配置文件后，继续单击【下一步】。



注意：

请您按照图示来添加配置文件， `tac_service_configurations_unpackage.plist` 文件中包含了敏感信息，请不要打包到 apk 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

如果还没有 Podfile，请创建一个。

```
$ cd your-project directory
$ pod init
```

并在您的 Podfile 文件中添加移动开发平台（MobileLine）的私有源：

```
source "https://git.cloud.tencent.com/qcloud_u/cocopoads-repo"
source "https://github.com/CocoaPods/Specs"
```

在 Podfile 中添加依赖：

```
pod 'TACStorage'
```

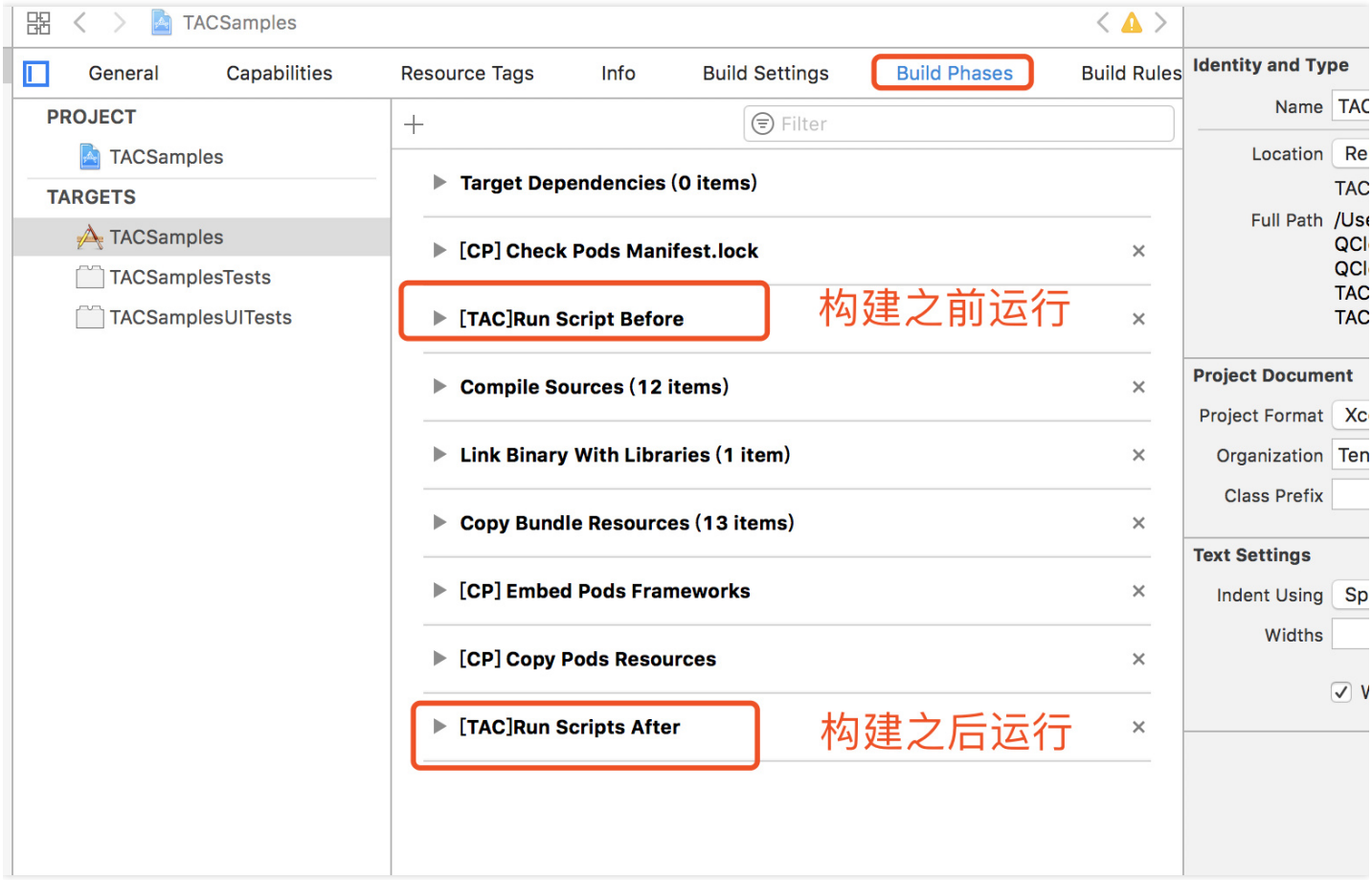
配置程序需要脚本

> 如果您在其他模块中完成了此步骤，请不要重复执行。

为了简化 SDK 的接入流程，我们使用 shell 脚本，帮助您自动化的去执行一些繁琐的操作，比如 crash 自动上报，在 Info.plist 里面注册各种第三方 SDK 的回调 scheme。因而，需要您添加以下脚本来使用我们自动化的加入流程。

脚本主要包括两个：

- 在构建之前运行的脚本，该类型的脚本会修改一些程序的配置信息，比如在 Info.plist 里面增加 qqwallet 的 scheme 回调。
- 在构建之后运行的脚本，该类型的脚本在执行结束后做一些动作，比如 Crash 符号表上报。



请按照以下步骤来添加脚本：

添加构建之前运行的脚本

1. 在导航栏中打开您的工程。
2. 打开 Tab `Build Phases`。
3. 单击 `Add a new build phase`，并选择 `New Run Script Phase`，您可以将改脚本命名 TAC Run Before

> **注意：**
> 请确保该脚本在 `Build Phases` 中排序为第二。

4. 根据自己集成的模块和集成方式将代码粘贴入 `Type a script...` 文本框:。

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.before.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.before.sh
```

其中 THIRD_FRAMEWORK_PATH 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 ${PODS_ROOT}/TACCore ，您需要黏贴的代码实例如下：

  ```
  ${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.before.sh
  ```

- 如果您使用手工集成的方式则为 您存储 TACCore 库的地址 ，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

  ```
  export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
  [您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.before.sh
  ```

添加构建之后运行的脚本

1. 在导航栏中打开您的工程。
2. 打开 Tab Build Phases 。
3. 单击 Add a new build phase ，并选择 New Run Script Phase ，您可以将改脚本命名 TAC Run Before。

> **注意：**
> 请确保该脚本在 Build Phases 中排序需要放到最后。

4. 根据自己集成的模块和集成方式将代码粘贴入 Type a script... 文本框:。

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.after.sh
```

其中 THIRD_FRAMEWORK_PATH 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 ${PODS_ROOT}/TACCore ，您需要黏贴的代码实例如下：

  ```
  ${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.after.sh
  ```

- 如果您使用手工集成的方式则为 [您存储 TACCore 库的地址] ，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
[您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.after.sh
```

第四步：初始化

集成好我们提供的 SDK 后，您需要在您自己的工程中添加初始化代码，从而让 MobileLine 服务在您的应用中进行自动配置。整个初始化的过程很简单。

步骤 1 导入 MobileLine 模块

在 UIApplicationDelegate 子类中导入移动开发平台（MobileLine）模块。

Objective-C 代码示例：

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

Swift 代码示例：

```
import TACCore
```

步骤 2 配置 TACApplication 共享实例

配置一个 TACApplication 共享实例，通常是在应用的 application:didFinishLaunchingWithOptions: 方法中配置。

使用默认配置

通常对于移动开发平台（MobileLine）的项目他的配置信息都是通过读取 tac_services_configuration.plist 文件来获取的。

Objective-C 代码示例：

```
[TACApplication configurate];
```

Swift 代码示例：

```
TACApplication.configurate();
```

步骤三 配置 TACStorage 的使用权限

Storage SDK 需要一个后台授权服务器提供临时密钥，才能正常工作。关于如何在 SDK 里配置服务器接口，请参见 iOS 配置授权服务器。

启动服务

移动存储服务无需启动，到此您已经成功接入了 MobileLine 移动存储服务。

后续步骤

您可以通过策略精确控制您数据的访问权限，可以参考 数据安全性最佳实践。

了解 MobileLine：

- 查看 MoblieLine 应用示例

向您的应用添加 MobileLine 功能：

- 借助 Analytics 深入分析用户行为。
- 借助 messaging 向用户发送通知。
- 借助 crash 确定应用崩溃的时间和原因。
- 借助 storage 存储和访问用户生成的内容（如照片或视频）。
- 借助 authorization 来进行用户身份验证。
- 借助 payment 获取微信和手 Q 支付能力

iOS 配置授权服务器

最近更新时间：2018-06-20 14:38:19

使用 Storage 服务时，后台需要对您的身份进行校验，校验过程是通过调用接口时携带签名实现的。因此，Storage SDK 需要提前设置临时密钥才能正常的访问数据。临时密钥有一定的有效期，过期后自动失效。由于临时密钥需要永久密钥生成，而永久密钥放在客户端中有极大的泄露风险，因此建议通过后台生成临时密钥，并下发到客户端中。

在调用 Storage 任何功能接口与前，都需要设置 [TACStorageService defaultStorage].credentailFenceQueue.delegate ，并且实现 `QCloudCredentailFenceQueueDelegate` 协议来提供相关的权限信息。

> 可以在 对象存储控制台 上面获取密钥，也就是 SecretID 与 SecretKey。

在签名接口的回调里，其实需要做的就是使用 SecretID，SecretKey（使用临时密钥服务的情况下，还有 token 和 ExpirationDate）传给 QCloudCredential 实例，然后通过该实例来生成一个签名 Creator，再调用 continueBlock 把签名 Creator 传进去即可。请参考下面的例子。

临时密钥使用指南

假设您已经按照 快速搭建后台授权服务 搭好了授权服务器，并且服务器直接将 CAM 返回的 JSON 数据透传给客户端。（如果是其它格式的数据，那么需要自定义解析过程）。

这里假设请求临时密钥的接口是：

```
GET https://<SERVER_HOST><PATH>?<name>=<value>
Header: <header1>=<value1>
```

然后，返回的数据是一个标准的 JSON 格式：

```
{
"code":0,"message":"","codeDesc":"Success",
"data":
{
"credentials":
{
"sessionToken":"42f8151428b3960b1226f421b8f271c6242ad02c3",
"tmpSecretId":"AKIDtd9QSGWBIDuMaYFp57tSmrhJgohLtvpT",
```

```
"tmpSecretKey":"ZfV5PVLvFLCvPefPt76qKYXIo56tSmrg"
},
"expiredTime":1508400619
}
}
```

那么我们需要做的是构造一个 HTTP 请求，去获取临时密钥，并对返回的数据进行解析，请参考下面的代码：

```
- (void) fenceQueue:(QCloudCredentailFenceQueue *)queue requestCreatorWithContinue:(QCloudCredentailFenceQueueContinue)continueBlock {

// 开始构造一个 NSURLHTTPRequest 进行网络请求
//请求的 URL
NSURL* URL = [NSURL URLWithString:@"https://<SERVER_HOST><PATH>?<name>=<value>"];
NSMutableURLRequest* urlrequest = [[NSMutableURLRequest alloc] initWithURL:URL];
//请求的方法，前面我们假设了是 GET
[urlrequest setHTTPMethod:@"GET"];
//如果需要设置头部 Header
[urlrequest setValue:@"<value1>" forHTTPHeaderField:@"<header1>"];
[[[NSURLSession sharedSession] dataTaskWithRequest:urlrequest completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
//请求返回 200，成功
if (((NSHTTPURLResponse*)response).statusCode == 200) {
//将 JSON 转为 NSDictionary* 类型的对象
NSDictionary* result = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableLeaves error:nil];

//构造生成签名的 Creator
QCloudCredential* crendential = [[QCloudCredential alloc] init];
//如果是前面假设的返回的JSON格式，那么可以按照这么解析，否则需要自行从字典中找到对应的参数。
crendential.secretID = [result valueForKeyPath:@"data.credentials.tmpSecretId"];
crendential.secretKey = [result valueForKeyPath:@"data.credentials.tmpSecretKey"];
crendential.experationDate = [result valueForKeyPath:@"data.expiredTime"];
crendential.token = [result valueForKeyPath:@"data.credentials.sessionToken"];
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc] initWithCredential:crendential];
continueBlock(creator, nil);
}
}] resume];
}
```

使用永久密钥进行上传（强烈不建议在线上环境使用）

在调试时可以使用永久密钥进行上传，使用 SecretID, SecretKey生成 QCloudCredential 实例后，再生成一个 QCloudAuthentationV5Creator 实例进行调用 continueBlock 即可。建议仅仅在调试阶段使用， **不建议线上环境使用** 。

Objective-C 代码示例：

```
@interface TACStorageDemoViewController () <QCloudCredentailFenceQueueDelegate>
@end

@implementation TACStorageDemoViewController

- (void)viewDidLoad {
[super viewDidLoad];
//设置提供签名的对象,该对象需要实现QCloudCredentailFenceQueueDelegate 协议
[TACStorageService defaultStorage].credentailFenceQueue.delegate = self;
}

- (void) fenceQueue:(QCloudCredentailFenceQueue *)queue requestCreatorWithContinue:(QCloudCredentailFenceQueueContinue)continueBlock
{
QCloudCredential* crendential = [[QCloudCredential alloc] init];

// 在调试阶段您可以通过直接设置secretID和secretKey来测试服务，但是强烈不建议在线上环境使用该方式！！！
QCloudCredential* crendential = [[QCloudCredential alloc] init];
crendential.secretID = <#secretID#>;
crendential.secretKey = <#secretKey#>;
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc] initWithCredential:crendential];
continueBlock(creator, nil);
}
@end
```

Swift 代码示例：

```
class TACStorageDemoViewController: UIViewController ,QCloudCredentailFenceQueueDelegate{
func fenceQueue(_ queue: QCloudCredentailFenceQueue!, requestCreatorWithContinue continueBlock: QCloudCredentailFenceQueueContinue!) {
// 在调试阶段您可以通过直接设置secretID和secretKey来测试服务，但是强烈不建议在线上环境使用该方式！！！
let crendential = QCloudCredential.init()
crendential.secretID = <#secretID#>
crendential.secretKey = <#secretKey#>
let creator = QCloudAuthentationV5Creator.init(credential: crendential)
continueBlock(creator,nil)
```

```
}
```

监听任务状态

最近更新时间：2018-05-16 19:06:20

不管是上传还是下载任务，都可以通过下面的方法来监听任务的状态。其中 status 传入希望监听的状态，当对应的状态变更时会调用 handler 进行回调。例如如果希望监听任务的进度，那么 status 可以传入 TACStorageTaskStatusProgress 。

```
- (TACStorageHandler) observeStatus:(TACStorageTaskStatus)status handler:(void (^)(TACStorageTaskSnapshot * ))handler
```

示例（监听一个下载任务的进度）：

Objective-C 代码示例：

```
TACStorageReference* ref = [[TACStorageService defaultStorage] referenceWithPath:@"hello"];
NSURL* url = @"file-local-url";
TACStorageDownloadTask* download = [ref downloadToFile:url completion:^(NSURL * _Nullable URL, NSError * _Nullable error) {
// finish callback
}];
[download observeStatus:TACStorageTaskStatusProgress handler:^(TACStorageTaskSnapshot *snapshot) {
NSLog(@"下载任务进度：%f"snapshot.progress.fractionCompleted > 0.3)
];
[download enqueue];
```

Swift 代码示例：

```
let ref = TACStorageService.defaultStorage().rootReference()
let url = URL.init(string: "file-local-url")
let downLoad = ref?.download(toFile: url! as URL, completion: { (url:URL?, error:Error?) in
// finish callback
})
downLoad?.observeStatus(TACStorageTaskStatus.progress, handler: { (snapshot:TACStorageTaskSnapshot?) in
print("下载任务进度: ",snapshot?.progress.fractionCompleted as Any)
})
downLoad?.enqueue()
```

添加文件元数据

最近更新时间：2018-05-16 19:47:07

添加文件元数据

在文件上传时可以传入元数据，当需要附带一些特殊的信息（例如 Content-Disposition 头部，或者加入自定义的头部）时，可以通过附带在元数据里，随着文件一起上传。例如上传时希望加入一个值为 x-cos-meta-test 的头部时，可以参考下面的例子：

Objective-C 代码示例：

```
TACStorageReference* ref = [[TACStorageService defaultStorage] rootReference];
ref = [ref child:@"object-test"];
NSString* content = @"file-content";
__block TACStorageMetadata* metaData = [[TACStorageMetadata alloc] init];
metaData.customMetadata = @{@"x-cos-meta-test":@"testCustomHeader"};//加入自定义头部
TACStorageUploadTask* uploadTask = [ref putData:[content dataUsingEncoding:NSUTF8StringEncod
ing] metaData:metaData completion:^(TACStorageMetadata * _Nullable metadata, NSError * _Nullabl
e error) {
//result
}];
[uploadTask enqueue];
```

Swift 代码示例：

```
var ref = TACStorageService.defaultStorage().rootReference()
ref = ref?.child("object-test")
let content = "file-content"
let metaData = TACStorageMetadata.init()
metaData.customMetadata = ["x-cos-meta-test":"testCustomHeader"]//加入自定义头部
let uploadTask = ref?.put(content.data(using: String.Encoding.utf8)!, metaData: metaData, completio
n:{ (metadata:TACStorageMetadata?, error:Error?) in
//result
})
uploadTask?.enqueue()
```

管理上传

最近更新时间：2018-05-16 19:07:24

对于较大的文件（1 MB 以上）会采取分块上传的形式，上传时会将文件切分成 1 MB 大小的数个块，然后并行进行上传。对于这类型的上传可以实现暂停和续传。

Objective-C 代码示例：

```
NSString* mb4bfile = @"file-local-path";
TACStorageReference* ref = [[TACStorageService defaultStorage] referenceWithPath:@"hello"];
__block TACStorageMetadata* result;
TACStorageUploadTask* task = [ref putFile:[NSURL fileURLWithPath:mb4bfile] metaData:nil completi
on:^(TACStorageMetadata * _Nullable metadata, NSError * _Nullable error) {
//完成回调
}];
[task enqueue];
//上传还没完成时
//暂停上传
[task pause];
//暂停下载
[task resume];
//取消上传
[task cancel];
```

Swift 代码示例：

```
let mb4bfile = "file-local-path"
let ref = TACStorageService.defaultStorage().reference(withPath: "hello")
var result:TACStorageMetadata?
let task = ref?.putFile(URL.init(fileURLWithPath: mb4bfile), metaData: nil, completion: { (metadata:TA
CStorageMetadata?, error:Error?) in
//完成回调
})
task?.enqueue()
//上传还没完成时
//暂停上传
task?.pause()
//暂停下载
task?.resume()
//取消上传
task?.cancel()
```

上传文件

最近更新时间：2018-05-16 19:46:56

有了引用之后，您可以通过两种方式将文件上传到 COS：

- 从内存中的数据上传。
- 从代表设备上某个文件的路径上传。

从内存中的数据上传

```
/**
上传一段内存数据（NSData类型）到当前的COS路径上去

@param data 需要上传的NSData数据
@param meteData 当前数据的元信息
@param completion 结果回调
@return TACStorageUploadTask对象
*/
- (TACStorageUploadTask*) putData:(NSData*)data
metaData:(nullable TACStorageMetadata*)meteData
completion:(nullable void (^)(TACStorageMetadata *_Nullable metadata,
NSError *_Nullable error))completion;
```

Objective-C 代码示例：

```
NSData *data = [NSData dataWithContentsOfFile:@"rivers.jpg"];
// Create a reference to the file you want to upload
TACStorageReference *riversRef = [storageRef child:@"images/rivers.jpg"];
// Upload the file to the path "images/rivers.jpg"
TACStorageUploadTask *uploadTask = [riversRef putData:data
metadata:nil
completion:^(TACStorageMetadata *metadata,
NSError *error) {
if (error != nil) {
// Uh-oh, an error occurred!
} else {
// Metadata contains file metadata such as size, content-type, and download URL.
NSURL *downloadURL = metadata.downloadURL;
}
}];
[TACStorageUploadTask enqueue];
```

Swift 代码示例：

```
let data = NSData.init(contentsOfFile: "rivers.jpg")
// Upload the file to the path "images/rivers.jpg"
let riversRef = storageRef?.child("images/rivers.jpg")
let uploadTask = riversRef?.put(data! as Data, metaData: nil, completion: { (metadata:TACStorageMetadata?, error:Error?) in
if error != nil{
// Uh-oh, an error occurred!
}else{
// Metadata contains file metadata such as size, content-type, and download URL.
let downloadURL = metadata?.downloadURL
}
})
uploadTask?.enqueue()
```

从文件中上传

```
/**
上传一个文件到当前的COS路径上去

@param file 需要上传的文件
@param meteData 当前数据的元信息
@param completion 结果回调
@return TACStorageUploadTask对象
*/
- (TACStorageUploadTask*) putFile:(NSURL*)file
metaData:(nullable TACStorageMetadata*)meteData
completion:(nullable void (^)(TACStorageMetadata *_Nullable metadata,
NSError *_Nullable error))completion;
```

Objective-C 代码示例：

```
// File located on disk
NSURL *localFile = [NSURL URLWithString:@"path/to/image"];
// Create a reference to the file you want to upload
TACStorageReference *riversRef = [storageRef child:@"images/rivers.jpg"];
// Upload the file to the path "images/rivers.jpg"
TACStorageReference *uploadTask = [riversRef putFile:localFile metadata:nil completion:^(TACStorageMetadata *metadata, NSError *error) {
if (error != nil) {
// Uh-oh, an error occurred!
```

```
} else {
// Metadata contains file metadata such as size, content-type, and download URL.
NSURL *downloadURL = metadata.downloadURL;
}
}];
[uploadTask enqueue];
```

Swift 代码示例：

```
// File located on disk
let localFile = NSURL.init(string: "path/to/image")
// Create a reference to the file you want to upload
let riversRef = storageRef?.child("images/rivers.jpg")
// Upload the file to the path "images/rivers.jpg"
let uploadTask = riversRef?.putFile(localFile! as URL, metaData: nil, completion: { (metadata:TACStora
geMetadata?, error:Error?) in
if error != nil{
// Uh-oh, an error occurred!
}else{
// Metadata contains file metadata such as size, content-type, and download URL.
let downloadURL = metadata?.downloadURL
}
})
uploadTask?.enqueue()
```

下载文件

最近更新时间：2018-05-16 19:46:02

可以参考以下示例代码，将文件下载至本地路径中：

```
TACStorageReference* ref = [[TACStorageService defaultStorage] referenceWithPath:@"hello"];
NSURL* url = <# Local path which the file will be saved into #>
TACStorageDownloadTask* download = [ref downloadToFile:url completion:^(NSURL * _Nullable URL, NSError * _Nullable error) {
// Download finish call back
}];
[download observeStatus:TACStorageTaskStatusProgress handler:^(TACStorageTaskSnapshot *snapshot) {
double progress = snapshot.progress.fractionCompleted
//progress call back
}];
[download enqueue];
```

删除文件

最近更新时间：2018-05-16 19:02:02

删除文件时，只需要对其引用直接调用 deleteWithCompletion: 方法即可，可参考下面的示例：

Objective-C 代码示例：

```
TACStorageReference* ref = [[TACStorageService defaultStorage] referenceWithPath:@"test"];
[ref deleteWithCompletion:^(NSError * _Nullable error) {
if (nil == error) {
//删除成功
} ;
}];
```

Swift 代码示例：

```
let ref = TACStorageService.defaultStorage().reference(withPath: "test")
ref?.delete(completion: { (error:Error?) in
if nil == error{
//删除成功
}
})
```

文件引用

最近更新时间：2018-05-16 19:03:05

使用 Storage 进行文件操作

您的文件存储在 COS 存储桶中，此存储桶中的文件以分层结构存储，就像本地硬盘中的文件系统一样，通过创建对文件的引用，您的应用可以获得对相应文件的访问权限。之后，借助所创建的这些引用，您可以上传或下载数据、获取或更新元数据，也可以删除文件。引用可以指向特定的文件，也可以指向层次结构中更高层级的节点。

创建引用

引用可以看作是指向云端文件的指针：要上传、下载或删除文件或要获取或更新文件的元数据，请创建引用。由于引用属于轻型项目，因此您可以根据需要创建任意多个引用，引用还可以在多个操作中复用。

引用是使用 TACStorageService 服务并调用其 referenceWithPath 或者 rootReference 方法创建的：(其中reference是TACStorageReference类的一个实例）

Objective-C 代码示例：

```
[TACStorageService defaultStorage].credentailFenceQueue.delegate = self;
self.reference = [[TACStorageService defaultStorage] referenceWithPath:@"test-file/test.png"];
```

Swift 代码示例：

```
TACStorageService.defaultStorage().credentailFenceQueue.delegate = self
self.reference = TACStorageService.defaultStorage().reference(withPath: "test-file/test.png")
```

引用导航

您还可以使用 parent 和 root 方法在我们的文件层次结构中向上导航，parent 可向上导航一级，而 root 可直接导航到根目录：

Objective-C 代码示例：

```
imagesRef = [spaceRef parent];
// Root allows us to move all the way back to the top of our bucket
// rootRef now points to the root
TACStorageReference *rootRef = [spaceRef root];
```

Swift 代码示例：

```
let imageRef = spaceRef?.parent()
// Root allows us to move all the way back to the top of our bucket
// rootRef now points to the root
let rootRef = spaceRef?.root()
```

child、parent 和 root 可以多次链接到一起，每次都会返回一个引用，root 的 parent 是例外，它是 nil：

Objective-C 代码示例：

```
TACStorageReference *earthRef = [[spaceRef parent] child:@"earth.jpg"];
// nilRef is nil, since the parent of root is nil
TACStorageReference *nilRef = [[spaceRef root] parent];
```

Swift 代码示例：

```
let earthRef = spaceRef?.parent().child("earth.jpg")
// nilRef is nil, since the parent of root is nil
let nilRef = spaceRef?.root().parent()
```

服务器 SDK

服务器 SDK

最近更新时间：2018-06-15 18:11:04

服务器 SDK

除了从 Android 和 iOS 端访问 Storage 服务，您可以从后台访问。我们提供了多个语言的 SDK，请参考：

服务器 SDK

安全和访问控制
用户访问控制

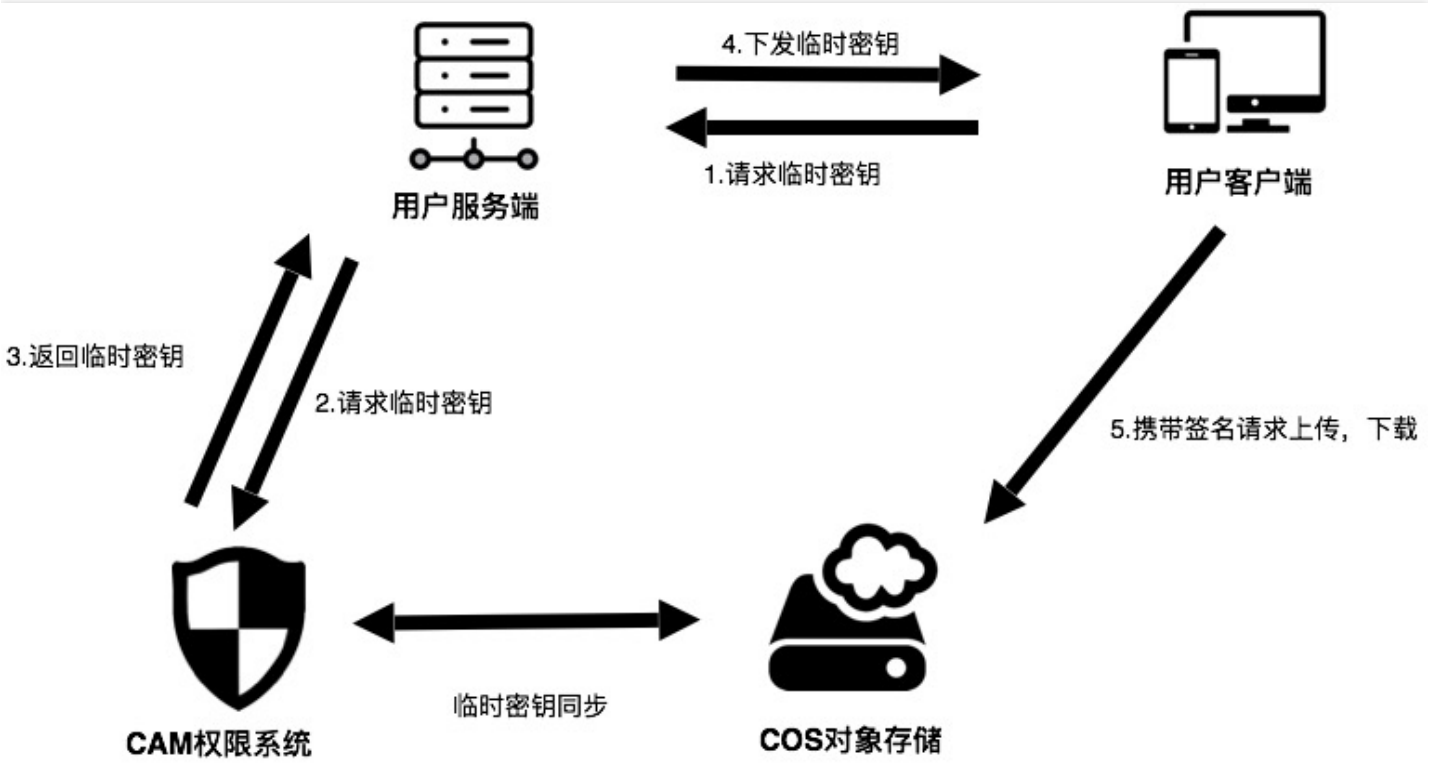
最近更新时间：2018-06-20 14:44:50

Storage 服务依赖于腾讯云 COS 服务，在使用服务时需要对请求进行鉴权。通过在请求时携带临时密钥，您可以临时授权您的 App 访问您的存储资源，而不会泄露您的永久密钥。密钥的有效期由您指定，过期后自动失效。**通常，我们都不建议您把永久密钥放到客户端代码中。**

本文主要介绍如何在后台服务器快速搭建一个授权服务，客户端通过拿到的临时密钥来对上传或者下载请求进行签名，从而保证您数据的安全性。

架构

整体架构图如下所示：



其中：

- 应用 APP：即用户手机上的 App。

- COS：腾讯云对象存储，负责存储 App 上传的数据。
- CAM：腾讯云访问管理，用于生成 COS 的临时密钥。
- 应用服务器：用户自己的后台服务器，这里用于获取临时密钥，并返回给应用 App。

获取永久密钥

临时密钥需要通过永久密钥才能生成。请登录 API密钥管理 获取，包含：

- SecretId
- SecretKey
- AppID

如何快速搭建后台授权服务

（建议）集成 SDK

如果您已经有后台服务器，我们建议您直接集成我们提供的 SDK 到现在的后台服务中。

第一步：集成 STS SDK

我们提供了以下语言的 SDK，帮忙您快速在后台集成请求密钥的功能，您可以根据自己后台的技术栈自行选择：

- Python SDK
- Java SDK
- Nodejs SDK

如果您使用其他语言开发，您也可以通过 REST API 直接访问我们的 STS 服务，请查看接口文档说明： 获取联合身份临时访问凭证。

第二步：发布 Web API

将您的服务以 Web API 的方式发布出去，这样您授权的客户端 APP 都可以通过标准的 HTTP 协议请求到临时密钥，访问 Storage 服务。

我们建议您直接把从 SDK 中得到的 JSON 数据作为您 API 的响应体，这样我们的 Storage Android 和 iOS SDK 会自动完成 JSON 格式的解析，您不需要手动处理。

使用 CVM 搭建授权服务

如果您目前还没有后台服务，那么您需要一台主机或者虚拟主机，来搭建一个独立的授权服务。下面以腾讯云 CVM Ubuntu 实例为例，详细介绍如何搭建一个简单的服务。

第一步：配置 Python 开发环境

购买 CVM 实例后，可直接用浏览器在控制台登录，或者使用终端直接 SSH 登录。登录成功后，首先需要配置 Python 开发环境：

```
sudo apt-get update
sudo apt-get install python-dev python-pip python-virtualenv
```

第二步：获取示例代码

下载示例代码

```
wget http://rickenwang-1253653367.cosgz.myqcloud.com/resources/signer-release.zip
unzip signer-release.zip
```

修改配置文件

下载并解压好的文件夹根目录下有 config.py 目录，可以来配置以下参数：

```
class Config(object):

def __init__(self):
pass

# 用户昵称，非必选
NAME = "WANG"
# 策略
POLICY = COMMON_POLICY
# 临时证书有效期
DURATION_SECOND = 1800
# 用户的secret id
SECRET_ID = "您的 secret id"
# 用户的secret key
SECRET_KEY = "您的 secret key"
```

其中 secret_key 和 secret_id 需要您修改为自己的密钥信息，具体值可以在 云 API 密钥控制台 上查询。COMMON_POLICY 提供的是一个拥有账户下所有资源读写权限的策略，如果您想配置更细化的策略，请参考 CAM COS 相关案例。

第三步：启动服务

激活 virtualenv 环境并进行配置

- 激活 virtualenv 环境

```
cd 到flask项目目录
virtualenv venv
source venv/bin/activate
```

- 配置环境

```
pip install gunicorn
pip install flask
pip install flask-script
```

使用 supervisor 管理服务

- 安装 supervisor 并打开配置文件

```
pip install supervisor
echo_supervisord_conf > supervisor.conf
vi supervisor.conf
```

- 在配置文件 supervisor.conf 的底部添加以下内容：

```
[program:manage]
command=./venv/bin/gunicorn -w4 -b0.0.0.0:5000 manage:app
directory=./
startsecs=0
stopwaitsecs=0
autostart=false
autorestart=false
stdout_logfile=./log/gunicorn.log
stderr_logfile=./log/gunicorn.err
```

启动服务

使用如下命令启动服务：

```
supervisord -c supervisor.conf
supervisorctl -c supervisor.conf start manage
```

第四步：验证服务

应用服务器启动后，直接用浏览器访问 `http://your.host.ip:5000/sign` 地址，若返回如下格式临时密钥 JSON 字符串，则说明服务正常运行。


```
{
"code":0,"message":"","codeDesc":"Success",
"data":
{
"credentials":
{
"sessionToken":"42f8151428b3960b1226f421b8f271c6242ad02c3",
"tmpSecretId":"AKIDtd9QSGWBIDuMaYFp57tSmrhJgohLtvpT",
"tmpSecretKey":"ZfV5PVLvFLCvPefPt76qKYXIo56tSmrg"
},
"expiredTime":1508400619
}
}
```


浏览器访问时 `your.host.ip` 请修改为您服务器的 IP 或者域名，默认的端口是 5000，您可自行在 `supervisor.conf` 文件中修改。

后续

如果您想获取更多关于数据安全的内容，可以参考 数据安全性最佳实践。

数据安全性最佳实践

最近更新时间：2019-03-06 21:02:37

安全性通常是应用开发过程中最复杂的部分之一。在大多数应用中，建立授权机制本身就十分困难，要确保正确则更加困难，而这直接关系到您的产品能否取得成功。

Storage 构建于腾讯云的安全体系中，为您的数据访问提供健壮的安全保障。我们提供基于存储桶的权限设置和基于资源的临时密钥机制，您可以按照需求，从不同粒度保障数据的安全访问。

前提

如果您还不了解我们整个鉴权体系的流程，以及如何搭建一个后台授权服务器，请查看 用户访问控制。

公开访问

公开访问是指您的存储资源可以直接访问，不需要授权。通常是您应用的一些静态资源。

访问分为读操作和写操作，您可以通过控制台设置存储桶的权限为:

- 公有读私有写：数据读取是公开的，但是写入需要授权。
- 公有读写：数据是完全公开读写的。请注意，这表示您的数据完全没有访问控制，容易导致数据泄露。

私有访问

私有访问是指您的存储资源只有持有合法的密钥才能访问。我们的应用数据可以允许 APP 内的用户访问，而不会暴露给其他的访问来源。

访问分为读操作和写操作，您可以通过控制台设置存储桶的权限为：

- 公有读私有写：数据读取是公开的，但是写入需要授权。
- 私有读写：数据的读写都需要授权。

当数据是私有访问时，每次请求都需要携带从 STS（临时密钥服务）获取的临时密钥，才能正常访问。注意在此之前，您最好在业务侧对访问身份先进行鉴定。

请求临时密钥时，您需要指定一个访问策略（Policy）。下面展示的是一个通用策略，它表示对所有资源都有读写权限：

```
{
"version": "2.0",
"statement": [{
"action": [
"name/cos:*"
],
"effect": "allow ",
"principal": {
"qcs": ["*"]
},
"resource": "*"
}]
}
```

如果您想了解策略的语法，请参考 策略语法说明。

基于用户的数据安全

基于用户的数据安全策略是指，数据允许自己的创建用户访问，而其他用户无法访问。这个适用于很多 APP 的场景，例如一个相册应用允许用户管理自己上传的图片，而其他用户没有访问权限。

您可以通过以下设置，达到用户级别的数据安全。

将数据设置为私有写

存储桶的权限应该是私有写的，根据应用场景，您可以选择是 `私有读写` 还是 `公有读私有写` 。

数据按照用户存储

Storage 的每个数据都有唯一的路径，在上传时，将用户唯一标识作为文件路径前缀的一部分。

举例：假如用户A唯一标识是 `2222` ，那么他上传的文件 `file1` 存放在 `/2222/file1` 中。

按照用户设置policy路径

使用用户唯一标识来设置 Policy 的资源路径前缀。

举例：假如您的存储桶地域是上海 `ap-shanghai` ，您的腾讯云 APPID 是 `12345678` ，用户A唯一标识是 `2222` ，当他访问时，Policy 设置为：

```
{
"version": "2.0",
"statement": [{
"action": [
"name/cos:*"
],
"effect": "allow",
"principal": {
"qcs": ["*"]
},
"resource": "qcs::cos:ap-shanghai:uid/12345678:prefix//12345678/2222/*"
}]
}
```

用这个 Policy 请求得到的临时密钥，只有对 `/2222/*` 目录下的数据有读写权限，而对其他路径的数据都没有访问权限。

您可以在 可用地域 查到不同地域的代码。
您可以在 API 密钥管理 查到您的腾讯云 APPID。

Java SDK使用说明

最近更新时间：2018-06-15 18:09:41

获取 SDK

Java SDK 下载>>

查看示例

请查看 `src/test` 下的 java 文件，里面描述了如何调用 SDK。

使用方法

1. 在 java 工程的 pom.xml 文件中集成依赖：

```
<dependency>
<groupId>com.qcloud</groupId>
<artifactId>qcloud-java-sdk</artifactId>
<version>2.0.6</version>
<scope>compile</scope>
</dependency>
```

2. 拷贝 `src/main` 下的 `StorageSts.java` 到您的工程中，调用代码如下：

```
TreeMap<String, Object> config = new TreeMap<String, Object>();

// 您的 SecretID
config.put("SecretId", "xxx");
// 您的 SecretKey
config.put("SecretKey", "xxx");
// 临时密钥有效时长，单位是秒，如果没有设置，默认是30分钟
config.put("durationInSeconds", 1800);

JSONObject credential = StorageSts.getCredential(config);
```

返回结果

成功的话，可以拿到包含密钥的 JSON 文本：

{"code":0,"message":"","codeDesc":"Success","data":{"credentials":{"sessionToken":"2a0c0ead3e6b8eed9608899eb74f2458812208ab30001","tmpSecretId":"AKIDBSrMaeFD0ZAECKuBzohnjAhJ53XNCE2F","tmpSecretKey":"UC7YjMrIlcuFgoWGwnrHwsMBrQrpUwYI"},"expiredTime":1526288317}}

自定义策略

默认情况下返回的密钥可以访问所有 cos 下的资源。如果您希望精确控制密钥的访问级别，您可以通过以下方式设置 policy：

```
config.put("policy", "your-policy");
```

关于策略描述和数据安全的最佳实践，请参见 数据安全性最佳实践。

Python SDK使用说明

最近更新时间：2018-06-15 18:10:17

获取 SDK

Python SDK 下载>>

查看示例

请查看 `sts_demo.py` 文件，里面描述了如何调用 SDK。

使用方法

拷贝 `sts.py` 文件到您的 Python 工程中，调用代码如下：

```
from sts import Sts

config = {
# 临时密钥有效时长，单位是秒，如果没有设置，默认是30分钟
'duration_in_seconds': 1800,
# 您的secret id
'secret_id': 'xxx',
# 您的secret key
'secret_key': 'xxx',
}

sts = Sts(config)
response = sts.get_credential()
json_content = response.content
```

返回结果

成功的话，可以拿到包含密钥的 JSON 文本：

```
{"code":0,"message":"","codeDesc":"Success","data":{"credentials":{"sessionToken":"2a0c0ead3e6b8eed9608899eb74f2458812208ab30001","tmpSecretId":"AKIDBSrMaeFD0ZAECKuBzohnjAhJ53XNCE2F","t
```

mpSecretKey":"UC7YjMrIlcuFgoWGwnrHwsMBrQrpUwYI"},"expiredTime":1526288317}}

自定义策略

默认情况下返回的密钥可以访问所有 cos 下的资源，如果你希望精确控制密钥的访问级别，例如访问某个路径下的资源，您可以通过以下方式设置 policy：

```
config = {
...,
policy: 'your-policy'
};
```

关于策略描述和数据安全的最佳实践，请参见 数据安全性最佳实践。

Nodejs SDK使用说明

最近更新时间：2018-06-15 18:10:33

获取 SDK

Nodejs SDK 下载>>

查看示例

请查看 `test` 下的 `sts_demo` ，里面描述了如何调用 SDK。

使用方法

1. npm install:

```
npm i qcloud-cos-sts --save
```

2. 调用代码如下：

```
var sts = require('tac-storage-sts');

var options = {
// 您的 secretId
secretId: 'xxx',
// 您的 secretKey
secretKey: 'xxx',
// 临时密钥有效时长，单位是秒
durationInSeconds: 1800
};


sts.getCredential(options, function(data) {
console.info(data)
});
```

返回结果

成功的话，可以拿到包含密钥的 JSON 文本：

{"code":0,"message":"","codeDesc":"Success","data":{"credentials":{"sessionToken":"2a0c0ead3e6b8eed9608899eb74f2458812208ab30001","tmpSecretId":"AKIDBSrMaeFD0ZAECKuBzohnjAhJ53XNCE2F","tmpSecretKey":"UC7YjMrIlcuFgoWGwnrHwsMBrQrpUwYI"},"expiredTime":1526288317}}

自定义策略

默认情况下返回的密钥可以访问所有 cos 下的资源，如果你希望精确控制密钥的访问级别，例如访问某个路径下的资源，您可以通过以下方式设置 policy：

```
var options = {
...,
policy: 'your-policy'
};
```

关于策略描述和数据安全的最佳实践，请参见 数据安全性最佳实践。

常见问题

最近更新时间：2018-03-28 17:49:32

Storage 可以存储哪种类型的数据？

Storage 可以存储任何格式的任何类型数据。

Storage 可以存储多少数据？

Storage 下的对象和目录无数量限制。