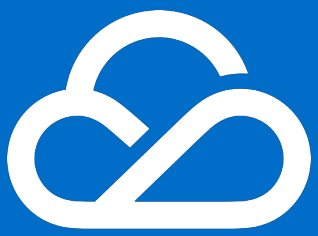
腾讯云对象存储

SDK 文档

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

概览

SDK 概览

除了直接使用 API 接口外，COS 提供了丰富多样的 SDK 供开发者使用，SDK由JSON API和XML API封装组成。

> 注意
> 目前 COS 的可用地域（Region）根据 XML API和 JSON API 有不同的取值，在使用不同的 API 和对应的 SDK 时要求使用对应的地域字段，详细信息请参见 可用地域 文档。

基于 JSON API 封装的 SDK

SDK	接入文档
PHP SDK	PHP SDK 接入说明
Python SDK	Python SDK 接入说明
Java SDK	Java SDK 接入说明
C++ SDK	C++ SDK 接入说明
C sharp SDK	C sharp SDK 接入说明
Android SDK	Android SDK 接入说明
iOS SDK	iOS SDK 接入说明
JavaScript SDK	JavaScript SDK 接入说明

基于 XML API 封装的 SDK

SDK	接入文档
Node.js SDK	Node.js SDK 接入说明

基于 JSON API 的 SDK

Android SDK

开发准备

SDK 获取

对象存储服务的 Android SDK 的下载github地址:Android SDK。
对象存储服务的 Android SDK 本地下载。

更多示例可参考Demo:Android SDK Demo。
（本版本SDK基于JSON API封装组成）

开发准备

1. SDK 支持 Android 2.2 及以上版本的手机系统；
2. 手机必须要有网络（GPRS、3G或 WIFI 网络等）；
3. 手机可以没有存储空间，但会使部分功能无法正常工作；
4. 从控制台获取APP ID、SecretID、SecretKey，详情参考权限控制。

SDK 配置

配置工程导入下列 jar 包：

- cos-android-sdk1.4.2.jar
- okhttp-3.2.0.jar
- okio-1.6.0.jar

SDK 需要网络访问相关的一些权限，需要在 AndroidManifest.xml 中增加如下权限声明：

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

初始化

进行操作之前需要实例化COSClient 和 COSClientConfig；

实例化 COSClientConfig

调用 COSClientConfig() 方法实例化 COSClientConfig 对象。

```
COSClientConfig config = new COSClientConfig();
```

配置 COSClientConfig

方法	方法描述
setEndPoint(String endPoint)	设置园区：华南 "gz"， 华北 "tj"， 华东"sh"，新加坡"sgp";sdk中默认为华南地区
setConnectionTimeout(int connectionTimeout)	连接超时设置
setSocketTimeout(int socketTimeout)	读取超时设置
setMaxConnectionsCount(int maxConnectionsCount)	并发数大小设置
setMaxRetryCount(int maxRetryCount)	失败请求重试次数
setHttpProtocol(String httpProtocol)	设置请求协议类型：默认为http请求，即"http://"; 若为https请求,则为 "https://"

实例化 COSClient

调用 COSClient（Context context, String appid, COSClientConfig config, String peristenceId)
方法实例化 COSClient 对象。

参数说明

参数名称	类型	是否必填	参数描述
context	Context	是	上下文
appid	String	是	腾讯云注册的APPID
config	COSClientConfig	否	配置设置
persistenceId	String	否	持久化 ID，每个 COSClient 需设置一个唯一的 ID 用于持久化保存未完成任务列表，以便应用退出重进后能够继续进行上传；传入为 Null，则不会进行持久化保存

示例

```
//创建COSClientConfig对象，根据需要修改默认的配置参数
COSClientConfig config = new COSClientConfig();
//设置园区
config.setEndPoint(COSEndPoint.COS_GZ);

Context context = getApplicationContext()；
String appid = "腾讯云注册的appid";
String peristenceId = "持久化Id";

//创建COSlient对象，实现对象存储的操作
COSClient cos = new COSClient(context,appid,config,peristenceId);
```

快速入门

初始化 COSClient

```
String appid = "腾讯云注册的appid";
```

```
Context context = getApplicationContext() ;
String peristenceId = "持久化Id";

//创建COSClientConfig对象，根据需要修改默认的配置参数
COSClientConfig config = new COSClientConfig();
//如设置园区
config.setEndPoint(COSEndPoint.COS_GZ);

COSClient cos = new COSClient(context,appid,config,peristenceId);
```

上传文件

```
String bucket = "cos空间名称";
String cosPath = "远端路径，即存储到cos上的路径";
String srcPath = "本地文件的绝对路径";
String sign = "签名，此处使用多次签名";

PutObjectRequest putObjectRequest = new PutObjectRequest();
putObjectRequest.setBucket(bucket);
putObjectRequest.setCosPath(cosPath);
putObjectRequest.setSrcPath(srcPath);
putObjectRequest.setSign(sign);
putObjectRequest.setListener(new IUploadTaskListener(){
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {

  PutObjectResult result = (PutObjectResult) cosResult;
  if(result != null){
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append(" 上传结果： ret=" + result.code + "; msg =" +result.msg + "\n");
  stringBuilder.append(" access_url= " + result.access_url == null ? "null" :result.access_url + "\n");
  stringBuilder.append(" resource_path= " + result.resource_path == null ? "null"
:result.resource_path + "\n");
```

```
 stringBuilder.append(" url= " + result.url == null ? "null" :result.url);
 Log.w("TEST",stringBuilder.toString());
 }
 }

 @Override
 public void onFailed(COSRequest COSRequest, final COSResult cosResult) {
 Log.w("TEST","上传出错： ret =" +cosResult.code + "; msg =" + cosResult.msg);
 }

 @Override
 public void onProgress(COSRequest cosRequest, final long currentSize, final long totalSize) {
 float progress = (float)currentSize/totalSize;
 progress = progress *100;
 Log.w("TEST","进度： " + (int)progress + "%");
 }
});
PutObjectResult result = cos.PutObject(putObjectRequest);
```

下载文件

```
String downloadURl = "下载文件的URL";
String savePath = "本地保存文件的路径";
String sign = "开启token防盗链了，则需要签名；否则，不需要";

GetObjectRequest getObjectRequest = new GetObjectRequest(downloadURl,savePath);
getObjectRequest.setSign(null);
getObjectRequest.setListener(new IDownloadTaskListener() {
 @Override
 public void onProgress(COSRequest cosRequest, final long currentSize, final long totalSize) {
 float progress = currentSize / (float)totalSize;
 progress = progress * 100;
 progressText.setText("progress =" + (int) (progress) + "%");
```

```
Log.w("TEST", "progress =" + (int) (progress) + "%");
}

@Override
public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
Log.w("TEST","code =" + cosResult.code + "; msg =" + cosResult.msg);
}

@Override
public void onFailed(COSRequest COSRequest, COSResult cosResult) {
Log.w("TEST","code =" + cosResult.code + "; msg =" + cosResult.msg);
}
});

GetObjectResult getObjectResult = cos.getObject(getObjectRequest);
```

生成签名

签名类型：

类型	含义
多次有效	有效时间内多次始终有效
单次有效	与资源URL绑定，一次有效

签名获取：

SDK 中用到的 SIGN，推荐使用 服务器端SDK，并由移动端向业务服务器请求。SIGN 的具体生成和使用请参照 访问权限。

目录操作

创建目录

方法原型

调用此接口可在指定的 bucket 下创建目录，具体步骤如下：

1. 调用 CreateDirRequest() 实例化 CreateDirRequest 对象；
2. 调用 COSClient 的 createDir 方法，传入 CreateDirRequest，返回 CreateDirResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	需要创建目录的路径
biz_attr	String	否	目录绑定的属性信息，由用户维护
sign	String	是	签名信息，此处使用多次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过CreateDirResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息
ctime	long(Unix时间戳)	创建时间

示例

```
CreateDirRequest createDirRequest = new CreateDirRequest();
createDirRequest.setBucket(bucket);
createDirRequest.setCosPath(cosPath);
createDirRequest.setBiz_attr(biz_attr);
createDirRequest.setSign(sign);
createDirRequest.setListener(new ICmdTaskListener() {
```

```
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  final CreateDirResult createDirResult = (CreateDirResult) cosResult;
  Log.w("TEST","目录创建成功： ret=" + createDirResult.code + "; msg=" + createDirResult.msg
  +"ctime = " + createDirResult.ctime));
  }

  @Override
  public void onFailed(COSRequest COSRequest, final COSResult cosResult) {
  Log.w("TEST","目录创建失败： ret=" + cosResult.code + "; msg=" + cosResult.msg);
  }
});

CreateDirResult result = cos.createDir(createDirRequest);
```

目录列表查询

方法原型

调用此接口可查询指定目录下的列表信息，具体步骤如下：

1. 调用 ListDirRequest() 实例化 ListDirRequest 对象；
2. 调用 COSClient 的 listDir 方法，传入 ListDirRequest，返回 ListDirResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
num	int	否	返回的数目，默认为1000，最大1000
content	String	否	透传字段，首次拉取必须清空。拉取下一页，需要将前一页返回值中的context透

参数名称	类型	是否必填	参数描述
			传到参数中
prefix	String	否	前缀查询的字符串
sign	String	是	签名信息，此处使用多次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过ListDirResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息
content	String	透传字段
listover	boolean	标识是否还有数据， true：列举结束；false：还有数据
infos	List	列举目录列表中文件或文件夹的属性

示例

```
ListDirRequest listDirRequest = new ListDirRequest();
listDirRequest.setBucket(bucket);
listDirRequest.setCosPath(cosPath);
listDirRequest.setNum(100);
listDirRequest.setContent("");
listDirRequest.setSign(sign);
listDirRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  //查询成功
  ListDirResult listObjectResult = (ListDirResult) cosResult;
  if(listObjectResult.infos != null && listObjectResult.infos.size() > 0) {
  for(int i = 0; i < length; i++){
  String str = listObjectResult.infos.get(i);
```

```
 try {
 JSONObject jsonObject = new JSONObject(str);
 if(jsonObject.has("sha")){
 //当前搜索的结果是文件
 }else{
 //当前搜索的结果是文件夹
 }
 } catch (JSONException e) {
 e.printStackTrace();
 }
 }
 }

 if (!listover) {
 // 存在下一页保存当前页的状态信息
 String content = result.content;
 }
 }

 @Override
 public void onFailed(COSRequest COSRequest, final COSResult cosResult) {
 Log.w("TEST",目录查询失败：ret=" + cosResult.code + "; msg =" + cosResult.msg);
 }
});


// 支持前缀查询
listDirRequest.setPrefix(prefix);
ListDirResult result=cos.listDir(listDirRequest);
```

目录更新

方法原型

调用此接口可更新指定目录的信息，具体步骤如下：

1. 调用 UpdateObjectRequest() 实例化 UpdateObjectRequest 对象；
2. 调用 COSClient 的 updateObject 方法，传入 UpdateObjectRequest，返回 UpdateObjectResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处使用单次签名
bizAttr	String	否	目录绑定的属性信息
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过UpdateObjectResult对象的成员变量成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息

示例

```
UpdateObjectRequest updateObjectRequest = new UpdateObjectRequest();
updateObjectRequest.setBucket(bucket);
updateObjectRequest.setCosPath(cosPath);
updateObjectRequest.setBizAttr(biz_attr);
updateObjectRequest.setSign(onceSign);
updateObjectRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  //更新成功
```

```
Log.w("TEST", cosResult.code+" : "+cosResult.msg);
}

@Override
public void onFailed(COSRequest COSRequest, COSResult cosResult) {
//更新失败
Log.w("TEST", cosResult.code+" : "+cosResult.msg);
}
});

UpdateObjectResult result = cos.updateObject(updateObjectRequest);
```

目录查询

方法原型

调用此接口可查询指定目录的信息，具体步骤如下：

1. 调用 GetObjectMetadataRequest() 方法实例化 GetObjectMetadataRequest 对象；
2. 调用 COSClient 的 getObjectMetadata方法，传入 GetObjectMetadataRequest，返回 GetObjectMetadataResult对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处使用多次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过GetObjectMetadataResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息
biz_attr	String	目录绑定的属性信息
ctime	long(Unix时间戳)	创建时间
mtime	long(Unix时间戳)	最后一次修改时间

示例

```
GetObjectMetadataRequest getObjectMetadataRequest = new GetObjectMetadataRequest();
getObjectMetadataRequest.setBucket(bucket);
getObjectMetadataRequest.setCosPath(cosPath);
getObjectMetadataRequest.setSign(sign);
getObjectMetadataRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  GetObjectMetadataResult result = (GetObjectMetadataResult) cosResult;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append("code=" + result.code + "; msg=" +result.msg + "\n");
  stringBuilder.append("ctime =" +result.ctime + "; mtime=" +result.mtime + "\n" );
  stringBuilder.append("biz_attr=" + result.biz_attr == null ? "" : result.biz_attr );
  Log.w("TEST",stringBuilder.toString());
  }

  @Override
  public void onFailed(COSRequest cosRequest, final COSResult cosResult) {
  Log.w("TEST", cosResult.code+" : "+cosResult.msg);
  }

});

GetObjectMetadataRequest result = cos.getObjectMetadata(getObjectMetadataRequest);
```

目录删除

方法原型

调用此接口可删除指定目录，具体步骤如下，注意只能删除空目录：

1. 调用 RemoveEmptyDirRequest() 方法实例化 RemoveEmptyDirRequest 对象；
2. 调用 COSClient 的 removeEmptyDir 方法，传入 RemoveEmptyDirRequest，返回 RemoveEmptyDirResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处使用单次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过RemoveEmptyDirResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息

示例

```
RemoveEmptyDirRequest removeEmptyDirRequest = new RemoveEmptyDirRequest();
removeEmptyDirRequest.setBucket(bucket);
removeEmptyDirRequest.setCosPath(cosPath);
removeEmptyDirRequest.setSign(onceSign);
removeEmptyDirRequest.setListener(new ICmdTaskListener() {
  @Override
```

```
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  final DeleteObjectResult removeEmptyDirResult = (DeleteObjectResult) cosResult;
  Log.w("TEST","removeDir 结果： ret=" + removeEmptyDirResult.code + "; msg=" +
removeEmptyDirResult.msg );
  }

  @Override
  public void onFailed(COSRequest COSRequest, final COSResult cosResult) {
  Log.w("TEST","目录删除失败：ret=" + cosResult.code + "; msg =" + cosResult.msg);
  }
});

RemoveEmptyDirResult result = cos.removeEmptyDir(removeEmptyDirRequest);
```

文件操作

文件上传

方法原型

调用此接口上传指定的文件，具体步骤如下：

1. 调用 PutObjectRequest() 方法实例化 PutObjectRequest对象；
2. 调用 COSClient 的 putObject 方法，传入 PutObjectRequest，返回 PutObjectResult对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
srcPath	String	是	本地绝对路径
insertOnly	String	否	是否覆盖相同文件名："0"，允许覆盖；"1",不允许覆

参数名称	类型	是否必填	参数描述
			盖；默认为："1"．
slice_size	int	否	分片上传时，设置的分片大小，默认为:1M
sign	String	是	签名信息，此处使用多次签名
listener	IUploadTaskListener	否	结果回调

返回结果说明

通过PutObjectResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息
access_url	String	访问文件的URL
url	String	操作文件的url

示例

```
PutObjectRequest putObjectRequest = new PutObjectRequest();
putObjectRequest.setBucket(bucket);
putObjectRequest.setCosPath(cosPath);
putObjectRequest.setSrcPath(srcPath);
putObjectRequest.setSign(sign);

 /* 设置是否允许覆盖同名文件： "0"，允许覆盖；"1",不允许覆盖；
putObjectRequest.setInsertOnly("1");

//设置是否开启分片上传
putObjectRequest.setSliceFlag(true);//设置是否分片上传：true，分片上传;false,简单文件上传
putObjectRequest.setSlice_size(1024*1024);//分片上传时，分片的大小
 */

putObjectRequest.setListener(new IUploadTaskListener(){
```

```
@Override
public void onSuccess(COSRequest cosRequest, COSResult cosResult) {

PutObjectResult result = (PutObjectResult) cosResult;
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append(" 上传结果： ret=" + result.code + "; msg =" +result.msg + "\n");
stringBuilder.append(" access_url= " + result.access_url + "\n");
stringBuilder.append(" resource_path= " + result.resource_path + "\n");
stringBuilder.append(" url= " result.url);
Log.w("TEST",stringBuilder.toString();
}

@Override
public void onFailed(COSRequest COSRequest, final COSResult cosResult) {
Log.w("TEST","上传出错： ret =" +cosResult.code + "; msg =" + cosResult.msg);
}

@Override
public void onProgress(COSRequest cosRequest, final long currentSize, final long totalSize) {
float progress = (float)currentSize/totalSize;
progress = progress *100;
Log.w("TEST","进度： " + (int)progress + "%");
}
});

PutObjectResult result = cos.putObject(putObjectRequest);
```

文件更新

方法原型

调用此接口可更新指定文件信息，具体步骤如下：

1. 调用 UpdateObjectRequest() 方法实例化 UpdateObjectRequest 对象；
2. 调用 COSClient 的 updateObject 方法，传入 UpdateObjectRequest，返回 UpdateObjectResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处使用单次签名
bizAttr	String	否	目录绑定的属性信息
authority	String	否	文件操作权限，与bucket拥有相同的权限(COSAuthority.EINVALID),私有读写权限（COSAuthority.EWRPRIVATE），私有写公有读权限（COSAuthority.EWPRIVATERPUBLIC）
custom_headers	Map	否	文件自定义的header: 如Cache-Control, Content-Disposition,Content-Language,x-cos-meta-。
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过UpdateObjectResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息

示例

```
UpdateObjectRequest updateObjectRequest = new UpdateObjectRequest();
updateObjectRequest.setBucket(bucket);
updateObjectRequest.setCosPath(cosPath);
updateObjectRequest.setBizAttr(biz_attr);
updateObjectRequest.setAuthority(authority);
updateObjectRequest.setCustomHeader(customer);
updateObjectRequest.setSign(onceSign);
updateObjectRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  Log.w("TEST", cosResult.code+" : "+cosResult.msg);
  }

  @Override
  public void onFailed(COSRequest COSRequest, COSResult cosResult) {
  Log.w("TEST", cosResult.code+" : "+cosResult.msg);
  }
});

UpdateObjectResult result= cos.updateObject(updateObjectRequest);
```

文件查询

方法原型

调用此接口可查询指定文件信息，具体步骤如下：

1. 调用 GetObjectMetadataRequest() 方法实例化 GetObjectMetadataRequest 对象；
2. 调用 COSClien t的 getObjectMetadata 方法，传入 GetObjectMetadataRequest，返回 GetObjectMetadataResult 对象；

参数说明

参数名称	类型	是否必填	参数描述

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处使用多次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过GetObjectMetadataResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息
biz_attr	String	目录绑定的属性信息
ctime	long(Unix时间戳)	创建时间
mtime	long(Unix时间戳)	最后一次修改时间
sha	String	文件的sha值
customs_headers	Map	文件的头部属性
filelen	int	文件的长度大小

示例

```
GetObjectMetadataRequest getObjectMetadataRequest = new GetObjectMetadataRequest();
getObjectMetadataRequest.setBucket(bucket);
getObjectMetadataRequest.setCosPath(cosPath);
getObjectMetadataRequest.setSign(sign);
getObjectMetadataRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  GetObjectMetadataResult result = (GetObjectMetadataResult) cosResult;
  StringBuilder stringBuilder = new StringBuilder();
  stringBuilder.append("code=" + result.code + "; msg=" +result.msg + "\n");
  stringBuilder.append("ctime =" +result.ctime + "; mtime=" +result.mtime + "\n" );
```

```
stringBuilder.append("biz_attr=" + result.biz_attr == null ? "" : result.biz_attr );
stringBuilder.append("sha=" + result.sha);
Log.w("TEST",stringBuilder.toString());
}

@Override
public void onFailed(COSRequest cosRequest, final COSResult cosResult) {
Log.w("TEST", cosResult.code+" : "+cosResult.msg);
}

});

GetObjectMetadataRequest result=cos.getObjectMetadata(getObjectMetadataRequest);
```

文件删除

方法原型

调用此接口可删除指定文件，具体步骤如下：

1. 调用 DeleteObjectRequest() 方法实例化 DeleteObjectRequest 对象；
2. 调用 COSClient 的 deleteObject 方法，传入 DeleteObjectRequest，回 DeleteObjectResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	腾讯云APP ID
bucket	String	是	目录所属bucket 名称
cosPath	String	是	远程相对路径
sign	String	是	签名信息，此处单次签名
listener	ICmdTaskListener	否	结果回调

返回结果说明

通过DeleteObjectResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息

示例

```
DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest();
deleteObjectRequest.setBucket(bucket);
deleteObjectRequest.setCosPath(cosPath);
deleteObjectRequest.setSign(onceSign);
deleteObjectRequest.setListener(new ICmdTaskListener() {
  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  Log.w("TEST", cosResult.code+" : "+cosResult.msg);
  }

  @Override
  public void onFailed(COSRequest COSRequest, COSResult cosResult) {
  Log.w("TEST", cosResult.code+" : "+cosResult.msg);
  }
});

DeleteObjectResult result = cos.deleteObject(deleteObjectRequest);
```

文件下载

方法原型

调用此接口可下载指定文件，具体步骤如下：

1. 调用 GetObjectRequest(String downloadURl,String savePath) 方法实例化 GetObjectRequest对象；

2. 调用 COSClient 的 getObject 方法，传入 GetObjectRequest,返回 GetObjectResult 对象；

参数说明

参数名称	类型	是否必填	参数描述
downloadUrl	String	是	下载文件的URL
localPath	String	是	本地保存文件的绝对地址
sign	String	否	签名信息：开启了防盗链则需要，否则不需要
listener	IDownloadTaskListener	否	结果回调

返回结果说明

通过GetObjectResult对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
code	String	结果码
msg	String	详细结果信息

示例

```
GetObjectRequest getObjectRequest = new GetObjectRequest(downloadURl,savePath);
getObjectRequest.setSign(null);
getObjectRequest.setListener(new IDownloadTaskListener() {
  @Override
  public void onProgress(COSRequest cosRequest, final long currentSize, final long totalSize) {
  float progress = currentSize / (float)totalSize;
  progress = progress * 100;
  progressText.setText("progress =" + (int) (progress) + "%");
  Log.w("TEST", "progress =" + (int) (progress) + "%");

  }

  @Override
  public void onSuccess(COSRequest cosRequest, COSResult cosResult) {
  Log.w("TEST","code =" + cosResult.code + "; msg =" + cosResult.msg);
```

```
  }

  @Override
  public void onFailed(COSRequest COSRequest, COSResult cosResult) {
  Log.w("TEST","code =" + cosResult.code + "; msg =" + cosResult.msg);
  }
});


GetObjectResul result = cos.getObject(getObjectRequest);
```

C sharp SDK

开发准备

相关资源

C sharp SDK github项目下载地址

开发准备

1. sdk依赖C# 4.0版本及以上， 推荐使用相同的版本。
2. 从控制台获取APP ID、SecretID、SecretKey。
3. 修改园区，CosCloud.cs文件内的URL定义，例如华东：http://sh.file.myqcloud.com/files/v2/
   华北：http://tj.file.myqcloud.com/files/v2/ 华南：http://gz.file.myqcloud.com/files/v2/
   （本版本SDK基于JSON API封装组成）

SDK 配置

直接下载github上提供的源代码，集成到开发环境。

若需 HTTPS 支持，则将 cos_dotnet_sdk/Api/CosCloud.cs 文件中变量 COSAPI_CGI_URL 中的 http 修改为 https 即可。

生成签名

多次有效签名

方法原型

public static string Signature(int appId, string secretId, string secretKey, long expired, string bucketName)

参数说明

参数名	类型	必填	参数描述
appId	int	是	AppId
secretId	string	是	Secret Id
secretKey	string	是	Secret Key，以上三项从控制台获取。
expired	long	是	过期时间，Unix时间戳
bucketName	string	是	bucket名称，bucket创建参见创建Bucket

示例

```
var sign = Sign.Signature(appId, secretId, secretKey, expired, bucketName);
```

单次有效签名

方法原型

```
public static string SignatureOnce(int appId, string secretId, string secretKey, string remotePath, string bucketName)
```

参数说明

参数名	类型	必填	参数描述
appId	int	是	AppId
secretId	string	是	Secret Id
secretKey	string	是	Secret Key，以上三项从控制台获取。
bucketName	string	是	bucket名称，bucket创建参见创建Bucket
remotePath	string	是	文件唯一的标识，格式/appid/bucketname/filepath/filename，其中/filepath/

参数名	类型	必填	参数描述
			filename为文件在此bucketname下的全路径，

示例

```
var sign = Sign.SignatureOnce(appId, secretId, secretKey,remotePath, bucketName);
```

更多签名详细说明，请参考权限控制 。

目录操作

创建目录

接口说明：用于目录的创建，可以通过此接口在指定bucket下创建目录。

方法原型

```
public string CreateFolder(string bucketName, string remotePath, Dictionary<string, string>
parameterDic = null)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	需要创建目录的全路径
parameterDic	string	否	属性字典

属性字典说明

参数名	类型	必填	参数描述
biz_attr	string	否	目录属性

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	否	返回数据，请参考《Restful API 创建目录》

示例

```
var createFolderParasDic = new Dictionary<string, string>();
  createFolderParasDic.Add(CosParameters.PARA_BIZ_ATTR,"new attribute");
var result = cos.CreateFolder(bucketName, folder, createFolderParasDic);
```

目录更新

接口说明：用于目录业务自定义属性的更新，可以通过此接口更新业务的自定义属性字段。

方法原型

```
public string UpdateFolder(string bucketName, string remotePath, Dictionary<string, string>
parameterDic = null)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	需要创建目录的全路径
parameterDic	string	是	目录更新参数字典

目录更新参数字典

参数名	类型	必填	参数描述
biz_attr	string	否	目录属性

返回结果说明

参数名	类型	必选	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
var updateParasDic = new Dictionary<string, string>();
  updateParasDic.Add(CosParameters.PARA_BIZ_ATTR,"new attribute");
var result = cos.UpdateFolder(bucketName, folder, updateParasDic);
```

目录查询

接口说明：用于目录属性的查询，可以通过此接口查询目录的属性。

方法原型

```
public string GetFolderStat(string bucketName, string remotePath)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	目录的全路径

返回结果说明

参数名	类型	必选	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	否	目录属性数据，请参考《Restful API 目录查询》

示例

```
var result = cos.GetFolderStat(bucketName, folder);
```

目录删除

接口说明：用于目录的删除，可以通过此接口删除空目录，如果目录中存在有效文件或目录，将不能删除。

方法原型

public string DeleteFolder(string bucketName, string remotePath)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	目录的全路径

返回结果说明

参数名	类型	必选	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
var result = cos.DeleteFolder(bucketName, folder);
```

列举目录下文件&目录

接口说明：用于列举目录下文件和目录，可以通过此接口查询目录下的文件和目录属性。

方法原型

public string GetFolderList(string bucketName, string remotePath, Dictionary<string, string> parameterDic = null)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	目录的全路径
parameterDic	Dictionary	是	目录列表参数字典

目录列表参数字典

参数名	类型	必填	参数描述
num	string	是	要查询的目录/文件数量，最大1000
listFlag	string	否	listFlag:大于0返回全部数据，否则返回部分数据
context	String	否	透传字段,用于翻页,前端不需理解,需要往后翻页则透传回来
prefix	string	否	搜索前缀

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	API 错误码，成功时为0
message	String	是	错误信息
data	Array	是	返回数据，请参考《Restful API 目录列表》

示例

```
var folderlistParasDic = new Dictionary<string, string>();
  folderlistParasDic.Add(CosParameters.PARA_NUM,"100");
var result = cos.GetFolderList(bucketName, folder, folderlistParasDic);
```

文件操作

文件上传

接口说明：文件上传的统一接口，对于大于8m的文件，内部会通过多次分片的方式进行文件上传。没有断点续传功能，需要自己用分片上传接口实现。

方法原型

public string UploadFile(string bucketName, string remotePath, string localPath, Dictionary<string, string> parameterDic = null)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称，bucket创建参见创建Bucket
remotePath	string	是	文件在服务端的全路径
localPath	string	是	文件本地路径
parameterDic	Dictionary	否	文件上传参数字典

文件上传参数字典

参数名	类型	必填	参数描述
biz_attr	string	否	文件属性
insertOnly	string	否	同名文件是否覆盖；0：覆盖，1：不覆盖（默认）
slice_size	string	否	可选取值为:641024 5121024，11024 1024（ 默认），21024 1024，310241024

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0

参数名	类型	必带	参数描述
message	String	是	错误信息
data	Array	是	返回数据，请参考《Restful API 创建文件》

示例

```
var uploadParasDic = new Dictionary<string, string>();
uploadParasDic.Add(CosParameters.PARA_BIZ_ATTR,"file attribute");
uploadParasDic.Add(CosParameters.PARA_INSERT_ONLY,"0");
uploadParasDic.Add(CosParameters.PARA_SLICE_SIZE,SLICE_SIZE.SLIZE_SIZE_3M.ToString());
result = cos.UploadFile(bucketName, remotePath, localPath, uploadParasDic);
```

文件属性更新

接口说明：用于目录业务自定义属性的更新，可以通过此接口更新业务的自定义属性字段。

方法原型

```
public string UpdateFile(string bucketName, string remotePath, Dictionary<string, string>
parameterDic = null)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在Cos的全路径
parameterDic	string	否	文件更新参数字典

文件更新参数字典

参数名	类型	必填	参数描述
biz_attr	string	否	待更新的文件属性信息
authority	string	否	eInvalid(继承Bucket的读

参数名	类型	必填	参数描述
			写权限)；eWRPrivate(私有读写)；eWPrivateRPublic(公有读私有写)
Cache-Control	string	否	指定请求和响应遵循的缓存机制，如：no-cache；max-age=200
Content-Type	string	否	返回内容的MIME类型，如：text/html
Content-Disposition	string	否	控制用户请求所得的内容存为一个文件的时候提供一个默认的文件名，如：attachment; filename="fname.ext"
Content-Language	string	否	使用的语言，如：zh-CN
x-cos-meta-自定义内容	string	否	表示以“x-cos-meta-”名字开头的参数，用户按照自身业务场景，设置需要在 Header 中传输什么参数

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
var optionParasDic = new Dictionary<string, string>();
  optionParasDic.Add(CosParameters.PARA_BIZ_ATTR,"new attribute");
  optionParasDic.Add(CosParameters.PARA_AUTHORITY,AUTHORITY.AUTHORITY_PRIVATEPUBLIC);
  optionParasDic.Add(CosParameters.PARA_CACHE_CONTROL,"no");
  optionParasDic.Add(CosParameters.PARA_CONTENT_TYPE,"application/text");
  optionParasDic.Add(CosParameters.PARA_CONTENT_DISPOSITION,"filename=\"test.pdf\"");
  optionParasDic.Add(CosParameters.PARA_CONTENT_LANGUAGE,"en");
  optionParasDic.Add("x-cos-meta-test","test");
```

result = cos.UpdateFile(bucketName, remotePath, optionParasDic);

文件查询

接口说明：用于文件的查询，可以通过此接口查询文件的各项属性信息。

方法原型

public string GetFileStat(string bucketName, string remotePath)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在Cos的全路径

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	是	文件属性数据，请参考《Restful API 文件查询》

示例

var result = cos.GetFileStat(bucketName, remotePath);

文件删除

接口说明：用于文件的删除，可以通过此接口删除已经上传的文件。

方法原型

publicstring DeleteFile(string bucketName, string remotePath)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在服务端的全路径

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
var result = cos.DeleteFile(bucketName, remotePath);
```

分片上传list

接口说明：查询分片上传的情况

方法原型

public string UploadSliceList(string bucketName, string remotePath)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在服务端的全路径

返回结果说明

参数名	类型	必带	参数描述

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	是	分片上传信息

上传完成data的数据说明

参数名	类型	必带	参数描述
url	String	是	操作文件的url
access_url	String	是	生成的文件下载url
source_url	String	是	源地址
resource_path	String	是	资源路径. 格式:/appid/bucket/xxx
preview_url	String	否	预览地址

上传未完成data的数据说明

参数名	类型	必带	参数描述
session	String	是	init返回的标识
filesize	Int	是	文件大小
slice_size	Int	是	分片大小（64K-3M） 大于1M 必须为1M 整数倍
sha	String	否	文件的全文sha值，init时 若已带sha值，则返回该值
listparts	Json Array	是	已上传完成的分片，形如： [{ “offset” :0, “datalen” :1024}, {}, {}].

示例

```
var fileSha = SHA1.GetFileSHA1(localPath);
var result = SliceUploadInit(bucketName, remotePath, localPath, fileSha, bizAttribute, sliceSize,
insertOnly);
```

分片上传init

接口说明：分片上传的第一次握手,如果上一次分片上传未完成，会返回{"code":-4019,"message":"_ERROR_FILE_NOT_FINISH_UPLOAD"}init

接口说明：分片上传的第一次握手

方法原型

public string SliceUploadInit(string bucketName, string remotePath, string localPath, string fileSha,string bizAttribute = "", int sliceSize = SLICE_SIZE.SLIZE_SIZE_1M, int insertOnly = 1)

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在服务端的全路径
localPath	string	是	本地文件路径
fileSha	string	是	文件的Sha值

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	是	分片上传信息

data的数据说明

参数名	类型	必带	参数描述
session	string	否	(非秒传的大部分情况会有)唯一标识此文件传输过程的id
slice_size	Int	否	(非秒传大部分情况下会有)分片大小

参数名	类型	必带	参数描述
serial_upload	int	否	(非秒传大部分情况下会有) 1：只支持串行分片上传其它：支持并行分片

示例

```
var fileSha = SHA1.GetFileSHA1(localPath);
var result = SliceUploadInit(bucketName, remotePath, localPath, fileSha, bizAttribute, sliceSize, insertOnly);
```

分片上传

接口说明：分片上传数据

方法原型

```
public string SliceUploadData(string bucketName, string remotePath, string localPath, string fileSha, string session, long offset,int sliceSise,string sign)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在服务端的全路径
localPath	string	是	本地文件路径
fileSha	string	是	文件的Sha值
session	string	是	唯一标识此文件传输过程的id
offset	int	是	分片的偏移量
sliceSize	int	是	分片的大小
sign	string	是	签名（多次）

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	是	分片上传信息

data的数据说明

参数名	类型	必带	参数描述
session	string	是	(非秒传的大部分情况会有)唯一标识此文件传输过程的id
offset	Int	是	当前分片的offset
datalen	int	是	分片长度slice_size,返回的datalen就是当前分片的大小
serial_upload	int	否	(非秒传大部分情况下会有)1：只支持串行分片上传其它：支持并行分片

示例

```
var fileSha = SHA1.GetFileSHA1(localPath);
var result = SliceUploadDataInit(bucketName, remotePath, localPath, fileSha, session, offset, sliceSize,sign);
```

分片上传 finish

接口说明：告诉COS所有分片上传成功

方法原型

```
public string SliceUploadFinish(string bucketName, string remotePath, string localPath, string fileSha, string session)
```

参数说明

参数名	类型	必填	参数描述
bucketName	string	是	bucket名称
remotePath	string	是	文件在服务端的全路径
localPath	string	是	本地文件路径
fileSha	string	是	文件的Sha值
session	string	是	唯一标识此文件传输过程的id

返回结果说明

参数名	类型	必带	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Array	是	分片上传信息

data的数据说明

参数名	类型	必带	参数描述
session	string	是	(非秒传的大部分情况会有)唯一标识此文件传输过程的id
offset	Int	是	当前分片的offset
datalen	int	是	分片文件长度

示例

result = SliceUploadFinish(bucketName,remotePath,localPath,fileSha, sessionbizAttribute, sliceSize, insertOnly);

C++ SDK

开发准备

相关资源

github项目地址

C++ SDK 本地下载

开发环境

1. 安装openssl的库和头文件 http://www.openssl.org/source/
2. 安装curl的库和头文件 http://curl.haxx.se/download/curl-7.43.0.tar.gz
3. 安装jsoncpp的库和头文件 https://github.com/open-source-parsers/jsoncpp
4. 安装boost的库和头文件 http://www.boost.org/
5. 安装cmake工具 http://www.cmake.org/download/
6. 从控制台获取APP ID、SecretID、SecretKey。

注意：

1. 本 SDK 仅适用于 Linux 系统和编译环境，暂不支持 Windows 环境。
2. sdk中提供了curl和jsoncpp的库以及头文件，以上库编译好后替换掉sdk中相应的库和头文件即可，如果以上库已经安装到系统里，也可删除sdk中相应的库和头文件。
3. curl默认不支持多线程环境，如果项目使用多线程，在编译curl执行 configure 时需指定 --enable-ares 参数来开启异步DNS解析，依赖 c-ares库，如果系统没有，可到http://c-ares.haxx.se/ 下载安装。
4. jsoncpp的1.y.x版本需要c++11的支持，如果编译器不支持，可以换成 0.y.x版本。（本版本SDK基于JSON API封装组成）

SDK 配置

直接下载github上提供的源代码，集成到您的开发环境。

执行下面的命令 ：

```
cd ${cos-cpp-sdk}
mkdir -p build
cd build
cmake ..
make
```

cos_demo.cpp里面有常见API的例子。生成的cos_demo可以直接运行，生成的静态库名称为：libcossdk.a。生成的 libcossdk.a 放到你自己的工程里lib路径下，include 目录拷贝到自己的工程的include路径下。

生成签名

多次有效签名

方法原型

```
static string AppSignMuti(const uint64_t appId,
  const string &secretId,
  const string &secretKey,
  const uint64_t expired_time,
  const string &bucketName);
```

参数说明

参数名	类型	是否必填	默认值	参数描述
appId	uint64_t	是	无	项目APP ID
secretId	String	是	无	用户 Secret ID
secretKey	String	是	无	用户 SecretKey
expired_time	uint64_t	否	无	过期时间，Unix时间戳

参数名	类型	是否必填	默认值	参数描述
bucketName	String	否	无	bucket名称

返回结果说明

返回值：签名字符串

示例

```
uint64_t expired = time(NULL) + 60;
string sign = Auth::AppSignMuti(10000000,"SecretId","SecretKey",expired,"bucketName");
```

单次有效签名

方法原型

```
static string AppSignOnce(const uint64_t appId,
  const string &secretId,
  const string &secretKey,
  const string &path,
  const string &bucketName);
```

参数说明

参数名	类型	必须	默认值	参数描述
appId	uint64_t	是	无	项目 APP ID
secretId	String	是	无	项目 SecretID
secretKey	String	是	无	项目 SecretKey
bucketName	String	否	无	bucket名称
path	String	是	无	文件路径，以斜杠开头，例如/filepath/filename，为文件在此bucketname下的全路径

返回值说明

返回值：签名字符串

示例

```
string path= "/myFloder/myFile.rar";
sign = Auth::AppSignOnce(10000000, "SecretId", "SecretKey", path, bucketName);
```

初始化操作

初始化

接口说明：在使用COS操作之前，需要首先进行COS系统参数的设置，然后分别创建CosConfig以及CosAPI对象，COS的操作都是基于CosAPI对象进行的。

配置文件

```
"AppID":********,
"SecretID":"**********************************",
"SecretKey":"**********************************",
"Region":"sh", //COS区域, 华东园区：sh ；华南园区：gz ；华北园区：tj
；上传和下载域名均是跟此有关系，因此一定要保证正确
"SignExpiredTime":360, //签名超时时间
"CurlConnectTimeoutInms":180, //http超时时间
"CurlGlobalConnectTimeoutInms":360, //
"UploadSliceSize":1048576, //文件分片大小，可选值有524288、1048576、2097152、3145728
"IsUploadTakeSha":0, //文件上传时是否携带文件sha值，0：不携带，1：携带
"DownloadDomainType":2, //下载域名类型,1:cdn,2:cos,3:innercos,4:自定义域名
"SelfDomain":"",//自定义域名
"UploadThreadPoolSize":5, //单文件分片上传线程池大小
"AsynThreadPoolSize":2, //异步上传下载线程池大小
"LogoutType":1 //日志输出类型,0:不输出,1:输出到屏幕,2输出到syslog
```

COS API对象构造原型

CosConfig(const string& config_file);
CosAPI(CosConfig& config);

目录操作

创建目录

接口说明：用于目录的创建，调用者可以通过此接口在指定bucket下创建目录。

方法原型

string CosAPI::FolderCreate(FolderCreateReq& request);

参数说明

参数名	类型	默认值	参数描述
request	FolderCreateReq	无	目录创建请求类型

request成员	类型	默认值	设置方法	描述
bucket	string	无	构造函数	bucket名称
cosPath	string	无	构造函数	上传的目标路径
bizAttr	string	空字符串	构造函数	文件夹属性

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	参数描述
code	Int	返回码，成功时为0
message	String	描述信息

参数名	类型	参数描述
data	Array	返回数据，请参考《Restful API 创建目录》

示例

```
FolderCreateReq folderCreateReq(bucket,folder,folder_biz_attr);
string rsp = cos.FolderCreate(folderCreateReq);
```

目录更新

接口说明：用于目录业务自定义属性的更新，可以通过此接口更新业务的自定义属性字段。

方法原型

```
string FolderUpdate(FolderUpdateReq& request);
```

参数说明

参数名	类型	默认值	参数描述
request	FolderUpdateReq	无	目录更新请求类型

参数名	类型	默认值	设置方法	参数描述
bucket	String	无	构造函数	bucket名称
cosPath	String	无	构造函数	目录的全路径
bizAttr	string	空	构造函数	文件夹属性

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	参数描述
code	Int	返回码，成功时为0
message	String	描述信息
data	Array	返回数据，请参考《Restful API

参数名	类型	参数描述
		目录更新》

示例

```
FolderUpdateReq folderUpdateReq(bucket,folder, folder_biz_attr);
string rsp = cos.FolderUpdate(folderUpdateReq);
```

目录查询

接口说明：用于目录属性的查询，可以通过此接口查询目录的属性。

方法原型

```
string CosAPI::FolderStat(FolderStatReq& request);
```

参数说明

参数名	类型	默认值	参数描述
request	FolderStatReq	无	目录查询请求类型

参数名	类型	默认值	设置方法	参数描述
bucket	String	无	构造函数	bucket名称
cosPath	String	无	构造函数	目录的全路径

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	参数描述
code	Int	返回码，成功时为0
message	String	描述信息

示例

```
FolderStatReq folderStatReq(bucket,folder);
```

```
string rsp = cos.FolderStat(folderStatReq);
```

目录删除

接口说明：用于目录的删除，可以通过此接口删除空目录，如果目录中存在有效文件或目录，将不能删除。

方法原型

```
string CosAPI::FolderDelete(FolderDeleteReq& request);
```

参数说明

参数名	类型	默认值	参数描述
request	FolderDeleteReq	无	目录删除请求类型

参数名	类型	默认值	设置方法	参数描述
bucket	String	无	构造函数	bucket名称
cosPath	String	无	构造函数	目录的全路径

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	参数描述
code	Int	返回码，成功时为0
message	String	描述信息

示例

```
FolderDeleteReq folderDeleteReq(bucket,folder);
string rsp = cos.FolderDelete(folderDeleteReq);
```

目录列表

接口说明：用于列举目录下文件和目录，可以通过此接口查询目录下的文件和目录属性。

方法原型

string CosAPI::FolderList(FolderListReq& request);

参数说明

参数名	类型	默认值	参数描述
request	FolderListReq	无	目录列表请求类型

参数名	类型	默认值	设置方法	参数描述
bucket	String	无	构造函数	bucket名称
cosPath	String	无	构造函数	目录的全路径
listNum	int	1000	构造函数	查询数目，最大为1000
context	string	空字符串	构造函数	查询上下文。查看第一页，则传空字符串；若需翻页，需要将前一页查询响应中的context设置到参数中

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	是否必然返回	参数描述
code	Int	是	返回码，成功时为0
message	String	是	描述信息
data	Array	是	返回数据，请参考《Restful API 目录列表》

示例

```
FolderListReq folderListReq(bucket,folder);
string rsp = cos.FolderList(folderListReq);
```

文件操作

文件上传

接口说明：文件上传，包括单文件上传和分片上传（大于8M的文件需要通过将文件内容进行分片上传）。

方法原型

string CosAPI::FileUpload(FileUploadReq& request);

参数说明

参数名	类型	默认值	参数描述
request	FileUploadReq	无	文件上传请求类型

参数名	类型	默认值	设置方法	参数描述
bucket	String	无	构造函数	bucket名字
srcPath	String	无	构造函数	待上传的本地文件路径
cosPath	String	无	构造函数	文件的全路径
bizAttr	string	无	setBizAttr()	文件属性
insertOnly	int	1	构造函数及setInsertOnly()	同名文件是否覆盖。0：表示覆盖同名文件，1：表示不覆盖，当文件存在返回错误
sliceSize	int	1048576	setSliceSize()	分片大小，可选值512K/1M/2M/3M；默认1M，均需转换为字节数值

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	是否必然返回	参数描述
code	Int	是	返回码，成功时为0
message	String	是	描述信息
data	Array	否	返回数据，请参考《Restful API 文件上传》

示例

```
FileUploadReq fileUploadReq(bucket, srcPath, dstPath);
string rsp = cos.FileUpload(fileUploadReq);
```

文件更新

接口说明：用于目录业务自定义属性的更新，可以通过此接口更新业务的自定义属性字段。

方法原型

```
string CosAPI::FileUpdate(FileUpdateReq& request);
```

参数说明

参数名	类型	默认值	参数描述
request	FileUpdateReq	无	文件更新请求类型

参数名	类型	是否必填	设置方法	参数描述
bucket	String	是	构造函数	bucket名称
cosPath	String	是	构造函数	文件在对象存储服务端的路径
bizAttr	string	否	setBizAttr()	文件属性
authority	string	否	setAuthority()	文件权限，默认是继承bucket的权限合法取值: eInvalid(继承bucket), eWRPrivate(私有读写), eWPrivateRPublic(私有写,

参数名	类型	是否必填	设置方法	参数描述
				公有读)
forbid	int	否	setForbid()	文件封禁标志
custom_header	map	否	setCustomHeader()	用户自定义属性

custom_header说明

参数名	类型	是否必填	参数描述
Cache-Control	string	否	参见HTTP的Cache-Control
Content-Type	string	否	参见HTTP的Content-Type
Content-Disposition	string	否	参见HTTP的Content-Language
Content-Language	string	否	参见HTTP的Content-Disposition
Content-Encoding	string	否	参见HTTP的Content-Encoding
x-cos-meta-	string	否	自定义HTTP 头，参数必须以x-cos-meta-开头，值由用户定义，可设置多个

tips: custom_header是整体覆盖式更新，即更新属性可以选择其中的某几个，如果本次只更新其中的某几个，其他的都会被抹掉。

返回结果说明

通过函数返回值返回请求结果的json字符串：

参数名	类型	是否必然返回	参数描述
code	Int	是	返回码，成功时为0
message	String	是	描述信息

示例

FileUpdateReq fileUpdateReq(bucket,dstpath);

```
fileUpdateReq.setBizAttr(file_biz_attr);
fileUpdateReq.setAuthority(auth);
fileUpdateReq.setForbid(0);
fileUpdateReq.setCustomHeader(custom_header);
string rsp = cos.FileUpdate(fileUpdateReq);
```

文件查询

接口说明：用于文件的查询，可以通过此接口查询文件的各项属性信息。

方法原型

```
string CosAPI::FileStat(FileStatReq& request)
```

参数说明

参数名	类型	默认值	参数描述
request	FileStatReq	无	文件查询请求类型

FileStatReq

参数名	类型	是否必填	默认值	参数描述
bucket	String	是	无	bucket名称
cosPath	String	是	无	文件在对象存储服务端的全路径

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

参数名	类型	必然返回	参数描述
data	Array	是	返回数据，请参考《Restful API 文件查询》

示例

```
FileStatReq fileStatReq(bucket,dstpath);
string rsp = cos.FileStat(fileStatReq);
```

文件删除

接口说明：用于文件的删除，可以通过此接口删除已经上传的文件。

方法原型

```
FileDeleteReq fileDeleteReq(bucket,dstpath);
string rsp = cos.FileDelete(fileDeleteReq);
```

参数说明

参数名	类型	默认值	参数描述
request	FileDeleteReq	无	文件删除请求类型

FolderDeleteReq

参数名	类型	是否必填	默认值	参数描述
bucket	String	是	无	bucket名称
cosPath	String	是	无	文件在对象存储服务端的全路径

返回结果说明

通过函数返回值返回请求结果的json字符串

参数名	类型	是否必然返回	参数描述
code	Int	是	返回码，成功时为0
message	String	是	描述信息

示例

```
FileDeleteReq fileDeleteReq(bucket,dstpath);
string rsp = cos.FileDelete(fileDeleteReq);
```

Java SDK

开发准备

相关资源

cos java sdk v4 github项目

Java SDK本地下载

环境依赖

JDK 1.7
（本版本SDK基于JSON API封装组成）

安装SDK

- maven安装

pom.xml 添加依赖

```
<dependency>
  <groupId>com.qcloud</groupId>
  <artifactId>cos_api</artifactId>
  <version>4.4</version>
</dependency>
```

- 源码安装

从cos java sdk v4 github下载源码

卸载SDK

删除pom依赖或源码

历史版本

4.2版本是针对COS 4.X系统，接口与3.x的基本一致, 如果需要使用历史版本, 请参见cos java sdk v3 github

生成客户端对象

初始化秘钥信息

```
long appId = 1000000;
 String secretId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
 String secretKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
 // 设置要操作的bucket
 String bucketName = "xxxxxxxxxx";
 // 初始化秘钥信息
 Credentials cred = new Credentials(appId, secretId, secretKey);
```

初始化客户端配置(如设置园区)

```
// 初始化客户端配置
 ClientConfig clientConfig = new ClientConfig();
 // 设置bucket所在的区域，比如华南园区：gz； 华北园区：tj；华东园区：sh ；
 clientConfig.setRegion("gz");
```

生成客户端

```
// 初始化cosClient
 COSClient cosClient = new COSClient(clientConfig, cred);
```

文件操作

上传文件

方法原型

String uploadFile(UploadFileRequest request);

参数说明

参数名	类型	默认值	参数描述
request	UploadFileRequest	无	上传文件类型请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt
localPath	String	无	构造函数或set方法	通过磁盘文件上传的本地绝对路径
contentBufer	byte[]	无	构造函数或set方法	通过内存上传的buffer内容
bizAttr	String	空	构造函数或set方法	文件的备注，主要用于对该文件用途的描述
insertOnly	InsertOnly (枚举)	NO_OVER_WRITE (不覆盖)	set方法	是否直插入不覆盖已存在的文件。 NO_OVER_WRITE表示只直插入不覆盖，当文件存在时会返回错误；OVER_WRITE表示允许覆盖，当文件存在时覆盖原有文件，覆盖不会产生错误。
enableSavePoint	boolean	true	set方法	是否开启断点文件，

request成员	类型	默认值	设置方法	描述
				开启断点文件后，会在本地记录一个断点，如果上传失败，则会跳过已经上传过的片。开启断点文件会牺牲一部分上传速度。
enableShaDigest	boolean	false	set方法	是否计算sha摘要，如果开启sha，并且bucket下有相同内容文件，则会触发秒传。sha计算会耗费一定的CPU和时间，建议大文件不开启。
taskNum	int	16	set方法	文件上传的并发数

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
UploadFileRequest uploadFileRequest = new UploadFileRequest(bucketName, "/sample_file.txt",
"local_file_1.txt");
String uploadFileRet = cosClient.uploadFile(uploadFileRequest);
```

下载文件

方法原型

```
String getFileLocal(GetFileLocalRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	GetFileLocalRequest	无	下载文件请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt
localPath	String	无	构造函数或set方法	要下载到的本地路径
useCDN	boolean	true	set方法	是否通过CDN进行下载
referer	String	空串	set方法	设置Referer(针对开启了refer防盗链的bucket)
rangeStart	long	0	set方法	要下载的字节起始，参见Http Range
rangeEnd	long	Long.MAX_VALUE	set方法	下载的字节结束, 参见Http Range

示例

```
String localPathDown = "src/test/resources/local_file_down.txt";
GetFileLocalRequest getFileLocalRequest =
  new GetFileLocalRequest(bucketName, cosFilePath, localPathDown);
getFileLocalRequest.setUseCDN(false);
getFileLocalRequest.setReferer("*.myweb.cn");
String getFileResult = cosClient.getFileLocal(getFileLocalRequest);
```

移动文件

方法原型

String moveFile(MoveFileRequest request);

参数说明

参数名	参数类型	默认值	参数描述
request	MoveFileRequest	无	移动文件请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如/mytest/demo.txt
dstCosPath	String	无	构造函数或set方法	移动文件的目标地址，必须从bucket下的根/开始，文件路径不能以/结尾，例如/mytest/demo.txt.move
overWrite	枚举类型	NO_OVER_WRITE(不覆盖)	set方法setOverWrite	在移动的目标文件存在时，选择不覆盖还是覆盖，默认不覆盖

示例

```
String cosFilePath = "/sample_file.txt";
String dstCosFilePath = "/sample_file.txt.bak";
MoveFileRequest moveRequest =
  new MoveFileRequest(bucketName, cosFilePath, dstCosFilePath);
String moveFileResult = cosClient.moveFile(moveRequest);
```

获取文件属性

方法原型

String statFile(StatFileRequest request);

参数说明

参数名	参数类型	默认值	参数描述
request	StatFileRequest	无	获取文件属性请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
StatFileRequest statFileRequest = new StatFileRequest(bucketName, "/sample_file.txt");
String statFileRet = cosClient.statFile(statFileRequest);
```

更新文件属性

方法原型

String updateFile(UpdateFileRequest request);

参数说明

参数名	参数类型	默认值	参数描述
request	UpdateFileRequest	无	更新文件属性请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt
bizAttr	String	无	set方法	文件的备注，主要用于对改文件用途的描述
authority	String (枚举)	无	set方法	文件权限，默认是继承bucket的权限合法取值: eInvalid(继承bucket), eWRPrivate(私有读写), eWPrivateRPublic(私有写, 公有读)
cacheControl	String	无	set方法	参见HTTP的Cache-Control
contentType	String	无	set方法	参见HTTP的Content-Type
contentLanguage	String	无	set方法	参见HTTP的Content-Language
contentDisposition	String	无	set方法	参见HTTP的Content-Disposition
x-cos-meta-	String	无	set方法	自定义HTTP 头，参数必须以x-cos-meta-开头，值由用户定义，可设置多个

tips: 更新属性可以选择其中的某几个，对于HTTP头部cache_control，content_type, content_disposition和x-cos-meta-,

如果本次只更新其中的某几个，其他的都会被抹掉，即这4个属性是整体更新。

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
UpdateFileRequest updateFileRequest = new UpdateFileRequest(bucketName, "/sample_file.txt");

updateFileRequest.setBizAttr("测试目录");
updateFileRequest.setAuthority(FileAuthority.WPRIVATE);
updateFileRequest.setCacheControl("no cache");
updateFileRequest.setContentDisposition("cos_sample.txt");
updateFileRequest.setContentLanguage("english");
updateFileRequest.setContentType("application/json");
updateFileRequest.setXCosMeta("x-cos-meta-xxx", "xxx");
updateFileRequest.setXCosMeta("x-cos-meta-yyy", "yyy");

String updateFileRet = cosClient.updateFile(updateFileRequest);
```

删除文件

方法原型

```
String delFile(DelFileRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	DelFileRequest	无	删除文件请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
DelFileRequest delFileRequest = new DelFileRequest(bucketName, "/sample_file_move.txt");
String delFileRet = cosClient.delFile(delFileRequest);
```

目录操作

创建目录

方法原型

```
String createFolder(CreateFolderRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	CreateFolderRequest	无	创建目录请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从buck

request成员	类型	默认值	设置方法	描述
				et下的根/开始，目录路径必须以/结尾，例如 /mytest/dir/
bizAttr	String	空	set方法	目录的备注，主要用于对目录用途的描述

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
CreateFolderRequest createFolderRequest = new CreateFolderRequest(bucketName,
"/sample_folder/");
String createFolderRet = cosClient.createFolder(createFolderRequest);
```

获取目录属性

方法原型

```
String statFolder(StatFolderRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	StatFolderRequest	无	获取目录属性请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾,

request成员	类型	默认值	设置方法	描述
				例如 /mytest/dir/

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
StatFolderRequest statFolderRequest = new StatFolderRequest(bucketName, "/sample_folder/");
String statFolderRet = cosClient.statFolder(statFolderRequest);
```

更新目录属性

方法原型

```
String updateFolder(UpdateFolderRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	UpdateFolderRequest	无	更新目录属性请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/
bizAttr	String	空	set方法	目录的备注，主要用于对目录用途的描述

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
UpdateFolderRequest updateFolderRequest = new UpdateFolderRequest(bucketName,
"/sample_folder/");
updateFolderRequest.setBizAttr("这是一个测试目录");
String updateFolderRet = cosClient.updateFolder(updateFolderRequest);
```

获取目录列表

方法原型

```
String listFolder(ListFolderRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	ListFolderRequest	无	获取目录成员请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/
num	int	199	构造函数或set方法	获取列表成员的数量，最大为199
prefix	String	空	构造函数或set方法	搜索成员的前缀, 例如prefix为test表示只搜索以test开头的文件或目录

request成员	类型	默认值	设置方法	描述
context	String	空	构造函数或set方法	搜索上下文, 由上一次list的结果返回，作为这一次搜索的起点，用于循环获取一个目录下的所有成员

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含成员列表, 详情请参见返回值模块

示例

```
ListFolderRequest listFolderRequest = new ListFolderRequest(bucketName, "/sample_folder/");
String listFolderRet = cosClient.listFolder(listFolderRequest);
```

删除目录

方法原型

```
String delFolder(DelFolderRequest request);
```

参数说明

参数名	参数类型	默认值	参数描述
request	DelFolderRequest	无	删除目录请求

request成员	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	bucket名称
cosPath	String	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/

返回值

返回值类型	返回值描述
String	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
DelFolderRequest delFolderRequest = new DelFolderRequest(bucketName, "/sample_folder/");
String delFolderRet = cosClient.delFolder(delFolderRequest);
```

签名管理

签名模块提供了生成多次签名、单次签名和下载签名的接口，其中多次签名和单次签名在文件和目录操作的api内部使用，用户不用关心，下载签名用于方便用户生成下载私有bucket的文件签名。

多次签名

String getPeriodEffectiveSign(String bucketName, String cosPath, Credentials cred, long expired)

使用场景

上传文件, 重命名文件, 创建目录, 获取文件目录属性, 拉取目录列表

参数说明

参数名	参数类型	默认值	参数描述
bucket	String	无	bucket名称
cos_path	String	无	要签名的cos路径
cred	Credentials	无	用户身份信息, 包括appid, secretId, secretkey
expired	long	无	签名过期时间, UNIX时间戳

返回值

base64编码的字符串

示例

```
Credentials cred = new Credentials(appId, secretId, secretKey);
long expired = System.currentTimeMillis() / 1000 + 600;
String signStr = Sign.getPeriodEffectiveSign(bucketName, "/pic/test.jpg", cred, expired);
```

单次签名

String getOneEffectiveSign(String bucketName, String cosPath, Credentials cred)

使用场景

删除和更新文件目录

参数说明

参数名	参数类型	默认值	参数描述
bucket	unicode	无	bucket名称
cos_path	unicode	无	要签名的cos路径
cred	Credentials	无	用户身份信息, 包括appid, secretId, secretkey

返回值

base64编码的字符串

示例

```
Credentials cred = new Credentials(appId, secretId, secretKey);
```

String signStr = Sign.getOneEffectiveSign(bucketName, "/pic/test.jpg", cred);

下载签名

String getDownLoadSign(String bucketName, String cosPath, Credentials cred, long expired)

使用场景

生成文件的下载签名, 用于下载私有bucket的文件

参数说明

参数名	参数类型	默认值	参数描述
bucket	unicode	无	bucket名称
cos_path	unicode	无	要签名的cos路径
cred	Credentials	无	用户身份信息, 包括appid, secretId, secretkey
expired	long	无	签名过期时间, UNIX时间戳

返回值

base64编码的字符串

示例

```
Credentials cred = new Credentials(appId, secretId, secretKey);
long expired = System.currentTimeMillis() / 1000 + 600;
String signStr = Sign.getDownLoadSign(bucketName, "/pic/test.jpg", cred, expired);
```

返回值

code	含义

code	含义
0	操作成功
-1	输入参数错误, 例如输入的本地文件路径不存在, cos文件路径不符合规范
-2	网络错误, 如404等
-3	连接cos时发生异常，如连接超时

PHP SDK

开发准备

相关资源

cos php sdk v4 github项目（本版本SDK基于JSON API封装组成）

PHP SDK 本地下载

开发环境

依赖环境：PHP 5.3.0 版本及以上

SDK 配置

下载 SDK 后，在使用 SDK 时，加载 cos-php-sdk-v4/include.php 并设置全局的超时时间及 COS 所在的区域即可。

```
require('cos-php-sdk-v4/include.php');
use Qcloud\Cos\Api;

$config = array(
  'app_id' => '',
  'secret_id' => '',
  'secret_key' => '',
  'region' => 'gz',
  'timeout' => 60
);

$cosApi = new Api($config);
```

生成签名

多次有效签名

方法原型

public function createReusableSignature($expiration, $bucket, $filepath);

参数说明

参数名	参数描述	类型	必填
expiration	过期时间，Unix 时间戳	long	是
bucket	Bucket 名称	String	是
filepath	文件路径	String	否

示例

```
$auth = new Auth($appId = '',$secretId ='',$secretKey = '');
$expiration = time() + 60;
$bucket = 'testbucket';
$filepath = "/myFloder/myFile.rar";
$sign = $auth->createReusableSignature($expiration, $bucket, $filepath);
```

单次有效签名

方法原型

public function createNonreusableSignature($bucket, $filepath);

参数说明

参数名	参数描述	类型	必填
bucket	Bucket 名称	String	是
filepath	文件路径，以斜杠开头。 例如	String	是

参数名	参数描述	类型	必填
	/filepath/filename 为文件在此 bucketname 下的全路径		

示例

```
$auth = new Auth($appId = '',$secretId ='',$secretKey = '');
$bucket = 'testbucket';
$filepath = "/myFloder/myFile.rar";
$sign = $auth->createNonreusableSignature($bucket, $filepath);
```

目录操作

创建目录

接口说明：用于目录的创建，可通过此接口在指定 Bucket 下创建目录。

方法原型

```
public function createFolder($bucketName, $path, $bizAttr = null);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	目录全路径	String	是
bizAttr	目录属性信息，业务自行维护	String	否

返回值说明(json)

参数名	参数描述	类型	必带

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是
data	返回数据，请参考《Restful API 创建目录》	Array	否

示例

```
$bizAttr = "attr_folder";
$result = $cosApi->createFolder($bucketName, $path,$bizAttr)
```

目录更新

接口说明：用于目录业务自定义属性的更新，调用者可以通过此接口更新业务的自定义属性字段。

方法原型

```
public function updateFolder($bucketName, $path, $bizAttr = null);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	目录路径	String	是
bizAttr	目录属性信息	String	否

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是

示例

```
$bizAttr = "folder new attribute";
$result = $cosApi->updateFolder($bucketName, $path, $bizAttr)
```

目录查询

接口说明：用于目录属性的查询，调用者可以通过此接口查询目录的属性。

原型方法

```
public function statFolder($bucketName, $path);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	目录路径	String	是

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是
data	目录属性数据，请参考《Restful API 目录查询》	Array	否

示例

```
$result = $cosApi->statFolder($bucketName, $path);
```

删除目录

接口说明：用于目录的删除，调用者可以通过此接口删除空目录，如果目录中存在有效文件或目录，将不能删除。

方法原型

public function delFolder($bucketName, $path);

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	目录全路径	String	是

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是

示例

```
$result = $cosApi->delFolder($bucketName, $path);
```

列举目录中的文件和目录

接口说明：用于列举目录下文件和目录，可以通过此接口查询目录下的文件和目录属性。

方法原型

public function listFolder($bucketName, $path, $num = 20, $pattern = 'eListBoth', $context = null);

参数说明

参数名	参数描述	类型	必填

参数名	参数描述	类型	必填
bucketName	Bucket名称	String	是
path	目录的全路径	String	是
num	要查询的目录/文件数量	int	否
context	透传字段，查看第一页，则传空字符串。 若需要翻页，需要将前一页返回值中的 context 透传到参数中	String	否
pattern	eListBoth：列举文件和目录；eListDirOnly：仅列举目录；eListFileOnly：仅列举文件	String	否

返回值说明(json)

参数名	参数描述	类型	必带
code	API 错误码，成功时为 0	Int	是
message	错误信息	String	是
data	返回数据，请参考《Restful API 目录列表》	Array	是

示例

```
$result = $cosApi->listFolder($bucketName, $path, 20, 'eListBoth',0);
```

列举目录下指定前缀文件&目录

接口说明：用于列举目录下指定前缀的文件和目录，可以通过此接口查询目录下的指定前缀的文件和目录信息。

原型方法

```
public function prefixSearch($bucketName, $prefix, $num = 20, $pattern = 'eListBoth',
$context = null);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称，Bucket 创建参见 创建Bucket	String	是
prefix	列出含此前缀的所有文件(带全路径)	String	是
num	要查询的目录/文件数量	int	否
context	透传字段，查看第一页，则传空字符串。 若需要翻页，需要将前一页返回值中的context透传到参数中	String	否
pattern	eListBoth：列举文件和目录；eListDirOnly：仅列举目录；eListFileOnly：仅列举文件	String	否

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	API 错误信息	String	是
data	返回数据，请参考《Restful API 目录列表》	Array	是

示例

```
$prefix= "/myFolder/2015-";
$result = $cosApi->prefixSearch($bucketName, $prefix, 20, 'eListBoth',0);
```

文件操作

文件上传

接口说明：文件上传的统一接口，对于大于 20M 的文件，内部会通过多次分片的方式进行文件上传。

原型方法

```
public function upload($bucketName, $srcPath, $dstPath,
  $bizAttr = null, $slicesize = null, $insertOnly = null);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称，Bucket 创建参见 创建 Bucket	String	是
srcPath	本地要上传文件的全路径	String	是
dstPath	文件在 COS 服务端的全路径，不包括 /appid/bucketname	String	是
bizAttr	文件属性，业务端维护	String	否
slicesize	文件分片大小，当文件大于 20M 时，SDK 内部会通过多次分片的方式进行上传；默认分片大小为 1M，支持的最大分片大小为 3M	int	否
insertOnly	同名文件是否进行覆盖。0：覆盖；1：不覆盖	int	否

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是
data	返回数据，请参考《Restful API 创建文件》	Array	是

示例

```
$dstPath = "/myFolder/test.mp4";
$bizAttr = "";
$insertOnly = 0;
$sliceSize = 3 * 1024 * 1024;
$result = $cosApi->upload($srcPath,$bucketName,dstPath ,"biz_attr");
```

文件属性更新

接口说明：用于目录业务自定义属性的更新，可以通过此接口更新业务的自定义属性字段。

原型方法

```
public function update($bucketName, $path,
  $bizAttr = null, $authority=null,$customer_headers_array=null);
```

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	文件在文件服务端的全路径，不包括 /appid/bucketname	String	是
bizAttr	待更新的文件属性信息	String	否
authority	eInvalid(继承Bucket的读写权限)；eWRPrivate(私有读写)；eWPrivateRPublic(公有读私有写)	String	否
customer_headers_array	用户自定义头域。可携带参数名分别为：	String	否

参数名	参数描述	类型	必填
	Cache-Control 、 Content-Type 、 Content-Disposition 、 Content-Language 、以及以 x-cos-meta- 为前缀的参数名称		

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是

示例

```
$bizAttr = "";
$authority = "eWPrivateRPublic";
$customer_headers_array = array(
  'Cache-Control' => "no",
  'Content-Language' => "ch",
);
$result = $cosApi->update($bucketName, $dstPath, $bizAttr,$authority, $customer_headers_array);
```

文件查询

接口说明：用于文件的查询，调用者可以通过此接口查询文件的各项属性信息。

原型方法

public function stat($bucketName, $path);

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	文件在文件服务端的全路径	String	是

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是
data	文件属性数据，请参考《Restful API 文件查询》	Array	是

示例

$result = $cosApi->stat($bucketName, $path);

文件删除

接口说明：用于文件的删除，调用者可以通过此接口删除已经上传的文件。

原型方法

public function delFile($bucketName, $path);

参数说明

参数名	参数描述	类型	必填
bucketName	Bucket 名称	String	是
path	文件的全路径	String	是

返回值说明(json)

参数名	参数描述	类型	必带
code	错误码，成功时为 0	Int	是
message	错误信息	String	是

示例

```
$result = $cosApi->delFile($bucketName, $path);
```

iOS SDK

开发准备

SDK 获取

对象存储服务的 iOS SDK 的下载地址：iOS SDK

iOS SDK 本地下载

更多示例可参考Demo：iOS Demo
（本版本SDK基于JSON API封装组成）

开发准备

- iOS 8.0+；
- 手机必须要有网络（GPRS、3G或Wifi网络等）；
- 从控制台获取APP ID、SecretID、SecretKey。

SDK 配置

SDK 导入

使用Cocoapods导入

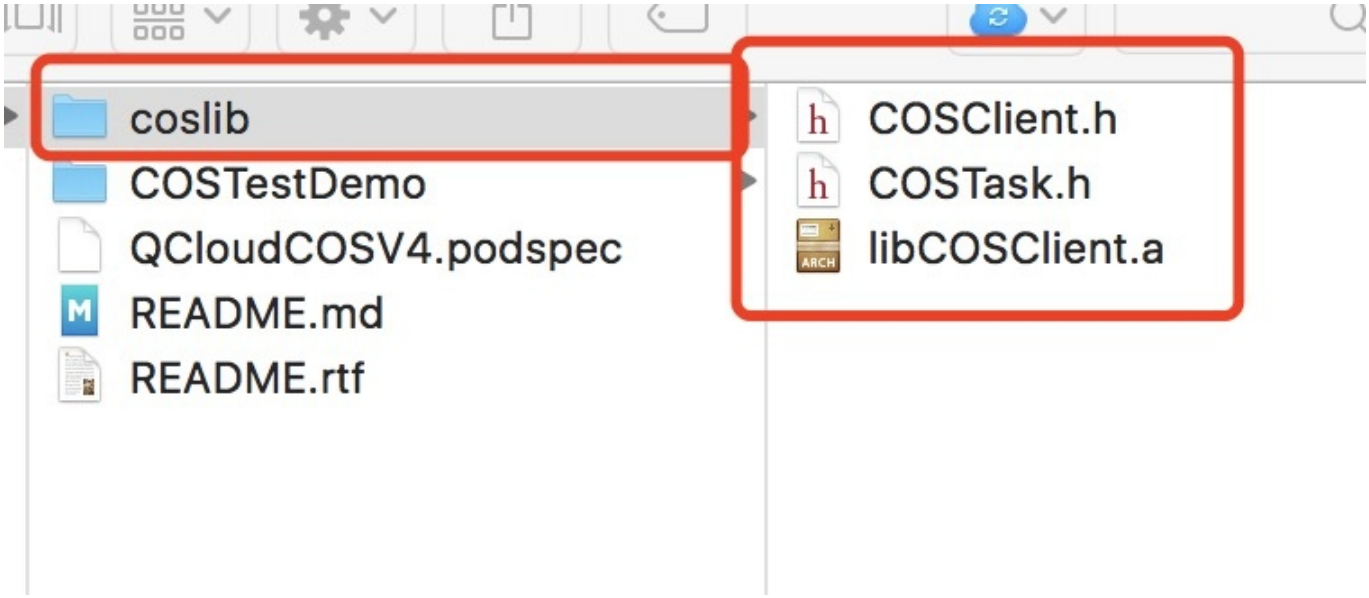
在Podfile文件中使用：

pod "QCloudCOSV4"

使用静态库导入

git clone https://github.com/tencentyun/COS_iOS_SDK.git

将目录coslib下面的文件拖入到工程中即可：

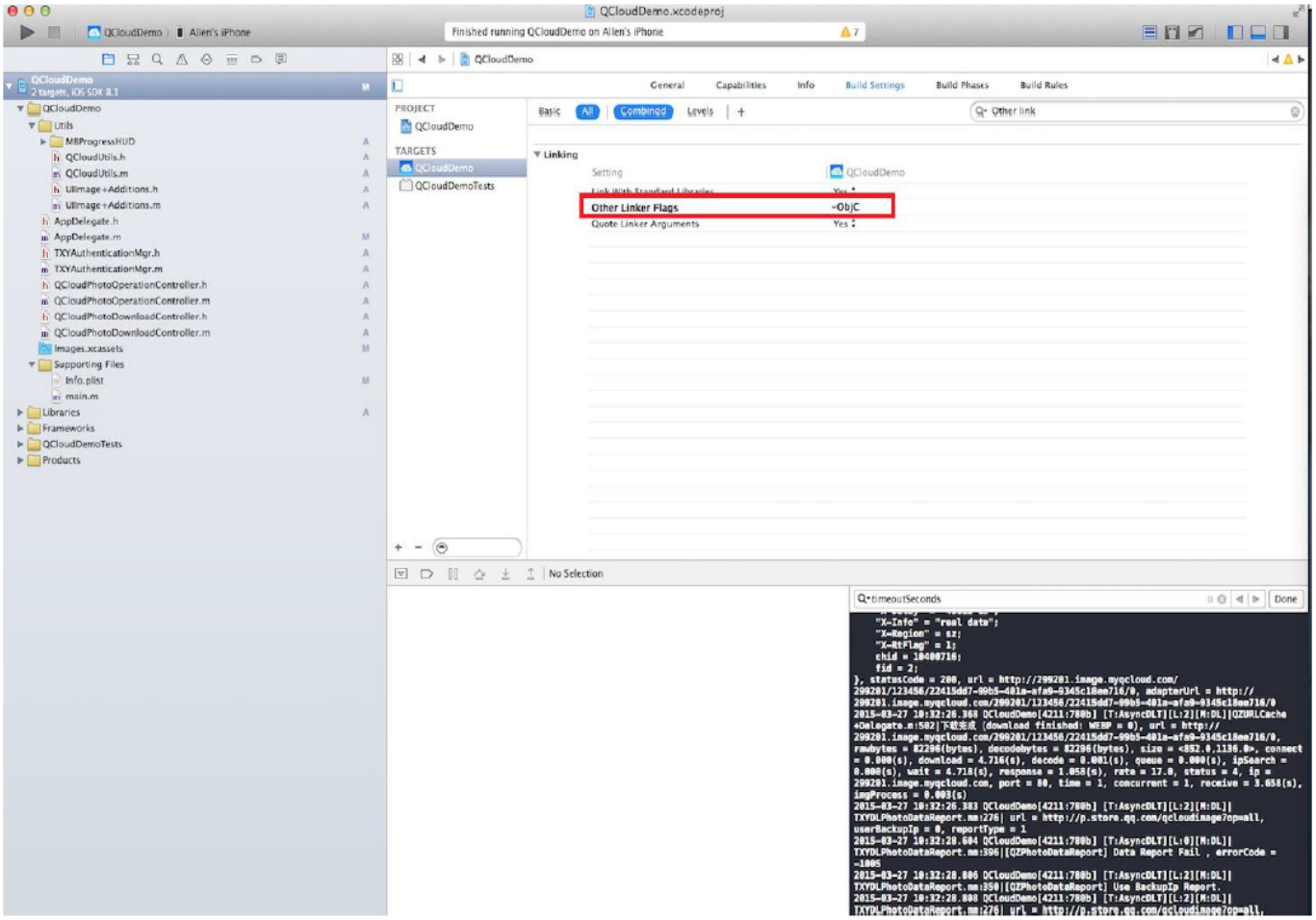


将目录coslib下面的文件拖入到工程拖入工程目录，Xcode 会自动将其加入链接库列表中。

□□

工程配置

在 Build Settings 中设置 Other Linker Flags，加入参数 -ObjC。



在工程info.plist文件中添加App Transport Security Settings 类型，然后在App Transport Security Settings下添加Allow Arbitrary Loads 类型Boolean,值设为YES。

如果想要在程序中使用HTTPS协议，请做如下设置：

```
COSClient *client= [[COSClient alloc] initWithAppId:appId
withRegion:@"sh";//@ “sh” bucket所在机房
[client openHttpsRequest:YES];
```

初始化

引入上传 SDK 的头文件 COSClient .h，使用目录操作时，需要先实例化 COSClient 对象。

方法原型

```
- (instancetype)initWithAppId:(NSString*)appId withRegion:(NSString *)region;
```

参数说明

参数名称	类型	是否必填	说明
appId	NSString *	是	项目ID，即APP ID。
region	NSString *	是	bucket被创建的时候机房区域，比如华东园区：“sh”，华南园区："gz"，华北园区："tj"

示例

```
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:@“sh”];
```

快速入门

这里演示的上传和下载的基本流程，更多细节可以参考demo；在进行这一步之前必须在腾讯云控制台上申请COS业务的appid；

STEP - 1 初始化COSClient

示例

```
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
```

STEP - 2 上传文件

在这里我们假设您已经申请了自己业务bucket。SDK所有的任务对应了相应的task，只要把相应的task参数传递给client，就可以完成相应的动作；

示例

```
COSObjectPutTask *task = [COSObjectPutTask new];
```

```
task.filePath = path;
task.fileName = fileName;
task.bucket = bucket;
task.attrs = @"customAttribute";
task.directory = dir;
task.insertOnly = YES;
task.sign = _sign;
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
client.progressHandler = ^(NSInteger bytesWritten,NSInteger totalBytesWritten,NSInteger
totalBytesExpectedToWrite){
//progress
};
[client putObject:task];
```

STEP - 3 下载文件

示例

```
COSObjectGetTask *cm = [[COSObjectGetTask alloc] initWithUrl:imgUrl.text];
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
//
};
client.downloadProgressHandler = ^(int64_t receiveLength,int64_t contentLength){
};
[client getObjectRequest:cm];
```

生成签名

签名类型：

类型	含义
多次有效	有效时间内多次始终有效
单次有效	与资源URL绑定，一次有效

签名获取：

移动端 SDK 中用到的签名，推荐使用 服务器端SDK，并由移动端向业务服务器请求。

目录操作

目录创建

方法原型

通过此接口在指定的 bucket 下创建目录，具体步骤如下：

1. 实例化 COSCreateDirCommand 对象；
2. 调用 COSClient 的 createDirRequest 方法，将 COSCreateDirCommand 对象传入。
3. 通过COSCreatDirTaskRsp 的对象返回结果

参数说明

参数名称	类型	是否必填	说明
dir	NSString *	是	目录路径（相对于bucket的路径）
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名
attrs	NSString *	否	用户自定义属性

返回结果说明

通过COSCreatDirTaskRsp 的对象返回结果

属性名称	类型	说明
retCode	int	任务描述代码
descMsg	NSString *	任务描述信息

示例

```
COSCreateDirCommand *cm = [COSCreateDirCommand new];
cm.directory = dir;
cm.bucket = bucket;
cm.sign = _sign;
cm.attrs = @"dirTest";
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{
//fail
}
};
[client createDir:cm];
```

目录属性更新

方法原型

通过调用此接口更新目录的自定义属性，具体步骤如下：

1. 实例化 COSUpdateDirCommand 对象；
2. 调用 COSClient 的 updateDirRequest 方法，将 COSUpdateDirCommand 对象传入；
3. 通过COSUpdateDirTaskRsp 对象返回结果

参数说明

参数名称	类型	是否必填	说明
dir	NSString *	是	目录路径（相对于bucket的路径）
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名
attrs	NSString *	否	用户自定义属性

返回结果说明

通过COSUpdateDirTaskRsp 的对象返回结果

属性名称	类型	说明
retCode	int	任务描述代码
descMsg	NSString *	任务描述信息

示例

```
COSUpdateDirCommand *cm = [COSUpdateDirCommand new];
cm.directory = dir;
cm.bucket = bucket;
cm.sign = _sign;//本业务使用一次性签名
cm.attrs = @"dirTest";
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
[client updateDir:cm];
```

目录属性查询

方法原型

通过调用此接口来查询目录的详细属性，具体步骤如下：

1. 实例化 COSDirmMetaCommand 对象；
2. 调用 COSClient 的 getDirMetaData 方法，将 COSDirmMetaCommand 对象传入；
3. 通过COSDirMetaTaskRsp的对象返回结果信息；

参数说明

参数名称	类型	是否必填	说明
dir	NSString *	是	目录路径（相对于bucket的路径）
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名

返回结果说明

通过COSDirmMetaCommand的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码
descMsg	NSString *	任务描述信息
data	NSDictionary *	任务描述信息

示例

```
COSDirmMetaCommand *cm = [COSDirmMetaCommand new];
cm.directory = dir;
cm.bucket = bucket;
cm.sign = sign;
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
[client getDirMetaData:cm];
```

目录删除

方法原型

调用此接口，进行指定 bucket
下目录的删除，如果目录中存在有效文件或目录，将不能删除。具体步骤如下：

1. 实例化 COSDeleteDirCommand 对象；
2. 调用 COSClient 的 deleteDirRequest 命令，传入 COSDeleteDirCommand 对象；
3. 通过COSdeleteDirTaskRsp 的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
dir	NSString *	是	目录路径（相对于bucket的路径）
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名

返回结果说明

通过COSdeleteDirTaskRsp的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码
descMsg	NSString *	任务描述信息

示例

```
COSDeleteDirCommand *cm = [COSDeleteDirCommand new];
cm.directory = dir;
cm.bucket = bucket;
cm.sign = _oneSign;//删除使用一次性签名
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
```

```
//sucess;
}else{}
};
[client deleteDir:cm];
```

目录列表

方法原型

调用此接口可以列出 bucket 中，指定目录下的文件、目录信息，具体步骤如下：

1. 实例化 COSListDirCommand 对象；
2. 调用 COSClient 对象的 listDirRequest 方法，将 COSListDirCommand 对象传入;
3. 通过COSDirListTaskRsp 的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
path	NSString *	是	目录路径（相对于bucket的路径）
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名
num	NSUInteger	是	一次拉取数目设定
pageContext	NSString *	是	透传字段，查看第一页，则传空字符串。若需要翻页，需要将前一页返回值中的context透传到参数中
prefix	NSString *	是	前缀查询

返回结果说明

通过TXYListDirCommandRsp的对象返回结果信息

属性名称	类型	说明

属性名称	类型	说明
context	NSString *	目录个数
listover	NSString *	文件个数
infos	NSArray *	文件目录属性列表
retCode	int	任务描述代码
descMsg	NSString *	任务描述信息

示例

```
COSListDirCommand *cm = [COSListDirCommand new] ;
cm.directory = dir;
cm.bucket = bucket;
cm.sign = _sign;
cm.number = 100;
cm.pageContext = @"";
cm.prefix = @"xx";
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
[client listDir:cm];
```

文件操作

初始化

方法原型

与目录操作相同，在进行文件操作之前，需引入上传 SDK 的头文件 COSClient .h，实例化 COSClient 对象。

参数说明

参数名称	类型	是否必填	说明
appId	NSString *	是	项目ID，即APP ID。
region	NSString *	是	bucket被创建的时候机房区域，比如上海： “sh” 广州："gz"

示例

```
- (instancetype)initWithAppId:(NSString*)appId withRegion:(NSString *)region;
```

文件上传

方法原型

调用此接口者可进行本地文件上传操作，具体步骤如下：

1. 实例化 COSObjectPutTask ；
2. 调用 COSClient 对象的 putObject 方法，将 COSObjectPutTask 对象传入；
3. 通过COSObjectUploadTaskRsp的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
filePath	NSString *	是	文件路径
sign	NSString *	是	签名
bucket	NSString *	是	目标 Bucket 名称
fileName	NSString *	是	目标 文件上传cos后显示的名称
attrs	NSString *	否	文件自定义属性
directory	NSString *	是	文件上传目录，相对路径,举例 “/path”
insertOnly	BOOL	是	上传文件的动作是插入覆盖，举例 “YES” 文件不会覆盖之前的上传的文件

返回结果说明

通过COSObjectUploadTaskRsp的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息
sourceURL	NSString *	成功后，后台返回文件的 CDN url
sourceURL	NSString *	成功后，后台返回文件的 源站 url

示例

```
COSObjectPutTask *task = [COSObjectPutTask new];
task.filePath = path;
task.fileName = fileName;
task.bucket = bucket;
task.attrs = @"customAttribute";
task.directory = dir;
task.insertOnly = YES;
task.sign = _sign;
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
client.progressHandler = ^(NSInteger bytesWritten,NSInteger totalBytesWritten,NSInteger
totalBytesExpectedToWrite){
//progress
};
[client putObject:task];
```

文件属性更新

方法原型

调用此接口更新文件的自定义属性，具体步骤如下：

1. 实例化 COSObjectUpdateCommand 对象；
2. 调用 COSClient 的 updateObject 命令，传入 COSObjectUpdateCommand 对象；
3. 通过COSObjectUpdateTaskRsp的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
fileName	NSString *	是	
bucket	NSString *	是	目录所属 bucket 名称
sign	NSString *	是	签名
attrs	NSString *	否	用户自定义属性

返回结果说明

通过TXYUpdateCommandRsp的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息

示例

```
COSObjectUpdateCommand *cm = [COSObjectUpdateCommand new]
  cm.fileName = file;
  cm.bucket = bucket;
  cm.sign = _oneSign;//单次签名
  COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
  client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
  if (resp.retCode == 0) {
  //sucess
  }else{
```

```
};
[client updateObject:cm];
```

文件属性查询

方法原型

调用此接口可查询文件的属性信息，具体步骤如下：

1. 实例化 COSObjectMetaCommand 对象；
2. 调用 COSClient 的 getObjectInfo 命令，传入 COSObjectMetaCommand 对象；
3. 通过COSObjectMetaTaskRsp 类返回结果信息

参数说明

参数名称	类型	是否必填	说明
filename	NSString *	是	
bucket	NSString *	是	文件所属 bucket 名称
directory	NSString *	是	目录路径（相对于bucket的路径）
sign	NSString *	是	签名

返回结果说明

通过TXYStatCommandRsp类返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息
data	NSDictionary *	成功时，文件基本信息

示例

```
COSObjectMetaCommand *cm = [COSObjectMetaCommand new] ;
```

```
cm.fileName = file;
cm.bucket = bucket;
cm.directory = dir;
cm.sign = _oneSign;//单次签名
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
[client getObjectMetaData:cm];
```

文件删除

方法原型

调用此接口进行文件的删除操作，具体步骤如下：

1. 实例化 COSObjectDeleteCommand 对象；
2. 调用 COSClient 的 deleteObject 命令，传入 COSObjectDeleteCommand 对象。
3. 通过COSObjectDeleteTaskRsp的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
filename	NSString *	是	
bucket	NSString *	是	文件所属 Bucket 名称
directory	NSString *	是	目录路径（相对于bucket的路径）
sign	NSString *	是	签名
objectType	TXYObjectType	是	业务类型，文件删除时设置为：TXYObjectFile

返回结果说明

通过COSObjectDeleteTaskRsp的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息

示例

```
COSObjectDeleteCommand *cm = [COSObjectDeleteCommand new]；
cm.fileName = file;
cm.bucket = bucket;
cm.directory = dir;
cm.sign = _oneSign;//单次签名
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];;
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{ }
};
[client deleteObject:cm];
```

文件下载

方法原型

调用此接口进行文件的下载操作，具体步骤如下：

1. 实例化 COSObjectGetTask 对象；
2. 调用 COSClient 的 getObjectRequest 命令，传入 COSObjectGetTask 对象。
3. 通过COSGetObjectTaskRsp 的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明

参数名称	类型	是否必填	说明
filePath	NSString *	是	文件下载地址

返回结果说明

通过 通过COSGetObjectTaskRsp 的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息
object	NSMutableData *	下载文件

示例

```
COSObjectGetTask *cm = [[COSObjectGetTask alloc] initWithUrl:imgUrl.text];
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
//
};
client.downloadProgressHandler = ^(int64_t receiveLength,int64_t contentLength){
};
[client getObject:cm];
```

文件分片上传

方法原型

调用此接口进行文件的下载操作，具体步骤如下：

1. 实例化 COSObjectPutTask 对象；
2. 调用 COSClient 的 putObject 命令，传入 COSObjectPutTask 对象。
3. 通过COSObjectUploadTaskRsp 的对象返回结果信息
4. 当multipartUpload 参数设置为YES 的时候上传文件上传的方式为分片上传，该参数默认为NO；

参数说明

参数名称	类型	是否必填	说明
filePath	NSString *	是	文件路径
multipartUpload	BOOL	否	文件上传是否使用分片上传
sign	NSString *	是	签名
bucket	NSString *	是	目标 Bucket 名称
fileName	NSString *	是	目标 文件上传cos后显示的名称
attrs	NSString *	否	文件自定义属性
directory	NSString *	是	文件上传目录，相对路径,举例“/path”
insertOnly	BOOL	是	上传文件的动作是插入覆盖，举例“YES” 文件不会覆盖之前的上传的文件

返回结果说明

通过COSObjectUploadTaskRsp 的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息

示例

```
COSObjectPutTask *task = [[COSObjectPutTask alloc] init];
task.multipartUpload = YES;//分片上传设置参数
task.filePath = path;
task.fileName = fileName;
task.bucket = bucket;
task.attrs = @"customAttribute";
task.directory = dir;
task.insertOnly = YES;
task.sign = _sign;
```

```
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){

if (rsp.retCode == 0) {
//sucess
}else{ }
};
client.progressHandler = ^(NSInteger bytesWritten,NSInteger totalBytesWritten,NSInteger
totalBytesExpectedToWrite){
//进度
};
[client putObject:task];
```

文件断点续传

方法原型

调用此接口进行文件的下载操作，具体步骤如下：

1. 实例化 COSObjectPutTask ；
2. 调用 COSClient 对象的 putObject 方法，将 之前上传过的COSObjectPutTask 对象传入；
3. 通过COSObjectUploadTaskRsp的对象返回结果信息

参数说明

参数名称	类型	是否必填	说明
filePath	NSString *	是	文件路径
sign	NSString *	是	签名
bucket	NSString *	是	目标 Bucket 名称
fileName	NSString *	是	目标 文件上传cos后显示的名称
attrs	NSString *	否	文件自定义属性
directory	NSString *	是	文件上传目录，相对路径,举例“/path”

参数名称	类型	是否必填	说明
insertOnly	BOOL	是	上传文件的动作是插入覆盖，举例“YES” 文件不会覆盖之前的上传的文件

返回结果说明

通过COSObjectUploadTaskRsp的对象返回结果信息

属性名称	类型	说明
retCode	int	任务描述代码，为retCode >= 0时标示成功，为负数表示为失败
descMsg	NSString *	任务描述信息
sourceURL	NSString *	成功后，后台返回文件的 CDN url
sourceURL	NSString *	成功后，后台返回文件的 源站 url

示例

```
COSObjectPutTask *task = [COSObjectPutTask new];
task.filePath = path;
task.fileName = fileName;
task.bucket = bucket;
task.attrs = @"customAttribute";
task.directory = dir;
task.insertOnly = YES;
task.sign = _sign;
COSClient *client= [[COSClient alloc] initWithAppId:appId withRegion:[Congfig instance].region];
client.completionHandler = ^(COSTaskRsp *resp, NSDictionary *context){
if (resp.retCode == 0) {
//sucess
}else{}
};
client.progressHandler = ^(NSInteger bytesWritten,NSInteger totalBytesWritten,NSInteger
totalBytesExpectedToWrite){
//progress
};
```

```
[client putObject:task];
```

Python SDK

开发准备

相关资源

- GitHub地址 GitHub项目地址，欢迎贡献代码以及反馈问题。
- Python SDK本地下载。
- PyPi PyPi项目地址 。
  （本版本SDK基于JSON API封装组成）

环境依赖

python 2.7

获取python版本的方法:

- Linux Shell

```
$ python -V
Python 2.7.11
```

- Windows cmd

```
D:\>python -V
Python 2.7.11
```

如果提示不是内部或外部命令，请现在windows环境变量PATH里添加上python的绝对路径

安装SDK

- pip安装

pip install qcloud_cos_v4

- 源码安装

github上下载SDK, 解压后如下执行(如果提示permission deny, 需要有管理员权限)

python setup.py install

卸载SDK

pip uninstall qcloud_cos_v4

历史版本

3.3版本对接口等进行了重构，和之前的历史版本诸多不同，同时修复了一些bug, 和历史版本不兼容,
如果需要使用历史版本, 请参见v3版的python sdk

生成客户端对象

初始化客户端

```
appid = 100000 # 替换为用户的appid
  secret_id = u'xxxxxxxx' # 替换为用户的secret_id
  secret_key = u'xxxxxxx' # 替换为用户的secret_key
  region_info = "sh" # 替换为用户的region，例如 sh 表示华东园区, gz 表示华南园区, tj 表示华北园区
  cos_client = CosClient(appid, secret_id, secret_key, region=region_info)
```

自定义接入点

如果需要使用自定义的接入域名，可以通过下面的方式设置

```
conf = CosConfig(hostname='', download_hostname='')
 cos_client.set_config(conf)
```

文件操作

上传文件

方法原型

```
def upload_file(self, request)
```

参数说明

参数名	类型	默认值	参数描述
request	UploadFileRequest	无	上传文件类型请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如/mytest/demo.txt
local_path	unicode	无	构造函数或set方法	要上传的本地文件的绝对路径
biz_attr	unicode	空	构造函数或set方法	文件的备注，主要用于对该文件用途的描述
insert_only	int (枚举)	1	构造函数或set方法	是否直插入不覆盖已存在的文件, 1表示只

request成员	类型	默认值	设置方法	描述
				直插入不覆盖, 当文件存在返回错误 0 表示允许覆盖

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
request = UploadFileRequest(bucket, u'/sample_file.txt', u'local_file_1.txt')
  upload_file_ret = cos_client.upload_file(request)
```

获取文件属性

方法原型

```
def stat_file(self, request)
```

参数说明

参数名	参数类型	默认值	参数描述
request	StatFileRequest	无	获取文件属性请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
request = StatFileRequest(bucket, u'/sample_file.txt')
  stat_file_ret = cos_client.stat_file(request)
```

更新文件属性

方法原型

```
def update_file(self, request)
```

参数说明

参数名	参数类型	默认值	参数描述
request	UpdateFileRequest	无	更新文件属性请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt
biz_attr	unicode	无	set方法	文件的备注，主要用于对改文件用途的描述
authority	unicode (枚举)	无	set方法	文件权限，默认是继承bucket的权限合法

request成员	类型	默认值	设置方法	描述
				取值: eInvalid(继承bucket), eWRPrivate(私有读写), eWPrivateRPublic(私有写,公有读)
cache_control	unicode	无	set方法	参见HTTP的Cache-Control
content_type	unicode	无	set方法	参见HTTP的Content-Type
content_language	unicode	无	set方法	参见HTTP的Content-Language
content_disposition	unicode	无	set方法	参见HTTP的Content-Disposition
x-cos-meta-	unicode	无	set方法	自定义HTTP 头，参数必须以x-cos-meta-开头，值由用户定义，可设置多个

tips:

用户可以在以上这些属性中选择几个进行更新。如果本次只更新HTTP头部cache_control，content_type, content_disposition和x-cos-meta-这四个中的某几个，其他的几个没有更新和设置，那么其他没有被设置的头部会被清除删除掉，不会出现，即这4个属性会出现整体一起更新变动。

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
request = UpdateFileRequest(bucket, u'/sample_file.txt')

 request.set_biz_attr(u'这是个demo文件') # 设置文件biz_attr属性
```

```
request.set_authority(u'eWRPrivate') # 设置文件的权限
request.set_cache_control(u'cache_xxx') # 设置Cache-Control
request.set_content_type(u'application/text') # 设置Content-Type
request.set_content_disposition(u'ccccxxx.txt') # 设置Content-Disposition
request.set_content_language(u'english') # 设置Content-Language
request.set_x_cos_meta(u'x-cos-meta-xxx', u'xxx') # 设置自定义的x-cos-meta-属性
request.set_x_cos_meta(u'x-cos-meta-yyy', u'yyy') # 设置自定义的x-cos-meta-属性

update_file_ret = cos_client.update_file(request)
```

下载文件

方法原型

def download_file(self, request)

参数说明

参数名	参数类型	默认值	参数描述
request	DownloadFileRequest	无	下载文件请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数	bucket名称
cos_path	unicode	无	构造函数	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt
local_filename	unicode	无	构造函数	本地文件路径

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因,

返回值类型	返回值描述
	详情请参见返回值模块

示例

```
request = DownloadFileRequest(bucket, u'/sample_file_move.txt', u'/tmp/a.txt')
 download_ret = cos_client.download_file(request)
```

删除文件

方法原型

```
def del_file(self, request)
```

参数说明

参数名	参数类型	默认值	参数描述
request	DelFileRequest	无	删除文件请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，文件路径不能以/结尾, 例如 /mytest/demo.txt

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
request = DelFileRequest(bucket, u'/sample_file_move.txt')
```

```
del_ret = cos_client.del_file(request)
```

目录操作

创建目录

方法原型

```
def create_folder(self, request)
```

参数说明

参数名	参数类型	默认值	参数描述
request	CreateFolderRequest	无	创建目录请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/
biz_attr	unicode	空	set方法	目录的备注，主要用于对目录用途的描述

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
request = CreateFolderRequest(bucket, u'/sample_folder/')
create_folder_ret = cos_client.create_folder(request)
```

获取目录属性

方法原型

def stat_folder(self, request)

参数说明

参数名	参数类型	默认值	参数描述
request	StatFolderRequest	无	获取目录属性请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含相关的属性, 详情请参见返回值模块

示例

```
request = StatFolderRequest(bucket, u'/sample_folder/')
  stat_folder_ret = cos_client.stat_folder(request)
```

更新目录属性

方法原型

def update_folder(self, request)

参数说明

参数名	参数类型	默认值	参数描述
request	UpdateFolderRequest	无	更新目录属性请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/
biz_attr	unicode	空	set方法	目录的备注，主要用于对目录用途的描述

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
request = UpdateFolderRequest(bucket, u'/sample_folder/')
  request.set_biz_attr(u'这是一个测试目录')
  update_folder_ret = cos_client.update_folder(request)
```

获取目录列表

方法原型

```
def list_folder(self, request)
```

参数说明

参数名	参数类型	默认值	参数描述

参数名	参数类型	默认值	参数描述
request	ListFolderRequest	无	获取目录成员请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/
num	int	199	构造函数或set方法	获取列表成员的数量，最大为199
prefix	unicode	空	构造函数或set方法	搜索成员的前缀, 例如prefix为test表示只搜索以test开头的文件或目录
context	unicode	空	构造函数或set方法	透传字段，从响应的返回内容中得到。若查看第一页，则将空字符串作为 context 传入。若需要翻页，需要将前一页返回内容中的 context 透传到参数中。

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess, 'data':\$data}, code为0表示成功, message为SUCCESS或者失败原因, data中包含成员列表, 详情请参见返回值模块

示例

```
request = ListFolderRequest(bucket, u'/sample_folder/')
  list_folder_ret = cos_client.list_folder(request)
```

删除目录

方法原型

def del_folder(self, request)

参数说明

参数名	参数类型	默认值	参数描述
request	DelFolderRequest	无	删除目录请求

request成员	类型	默认值	设置方法	描述
bucket_name	unicode	无	构造函数或set方法	bucket名称
cos_path	unicode	无	构造函数或set方法	cos路径, 必须从bucket下的根/开始，目录路径必须以/结尾, 例如 /mytest/dir/

返回值

返回值类型	返回值描述
dict	{'code':\$code, 'message':$mess}, code为0表示成功, message为SUCCESS或者失败原因, 详情请参见返回值模块

示例

```
request = DelFolderRequest(bucket, u'/sample_folder/')
  delete_folder_ret = cos_client.del_folder(request)
```

签名管理

签名模块提供了生成多次签名、单次签名和下载签名的接口，其中多次签名和单次签名在文件和目录操作的api内部使用，用户不用关心，下载签名用于方便用户生成下载私有bucket的文件签名。

多次签名

def sign_more(self, bucket, cos_path, expired)

使用场景

上传文件, 重命名文件, 创建目录, 获取文件目录属性, 拉取目录列表

参数说明

参数名	参数类型	默认值	参数描述
bucket	unicode	无	bucket名称
cos_path	unicode	无	要签名的cos路径
expired	int	无	签名过期时间, UNIX时间戳

返回值

base64编码的字符串

示例

```
cred = CredInfo(100000, u'xxxxxxx', u'xxxxxxxx') # appid, secret_id, secret_key
auth_obj = Auth(cred)
sign_str = auth_obj.sign_more(u'mybucket', u'/pic/1.jpg', int(time.time()) + 600)
```

单次签名

def sign_once(self, bucket, cos_path)

使用场景

删除和更新文件目录

参数说明

参数名	参数类型	默认值	参数描述
bucket	unicode	无	bucket名称
cos_path	unicode	无	要签名的cos路径

返回值

base64编码的字符串

示例

```
cred = CredInfo(100000, u'xxxxxxx', u'xxxxxxxx') # appid, secret_id, secret_key
auth_obj = Auth(cred)
sign_str = auth_obj.sign_once(u'mybucket', u'/pic/1.jpg')
```

下载签名

def sign_download(self, bucket, cos_path, expired)

使用场景

生成文件的下载签名, 用于下载私有bucket的文件

参数说明

参数名	参数类型	默认值	参数描述
bucket	unicode	无	bucket名称
cos_path	unicode	无	要签名的cos路径
expired	int	无	签名过期时间, UNIX时间戳

返回值

base64编码的字符串

示例

```
cred = CredInfo(100000, u'xxxxxxx', u'xxxxxxxx') # appid, secret_id, secret_key
auth_obj = Auth(cred)
sign_str = auth_obj.sign_download(u'mybucket', u'/pic/1.jpg', int(time.time()) + 600)
```

返回码

code	含义
0	操作成功
-1	输入参数错误, 例如输入的本地文件路径不存在, cos文件路径不符合规范
-2	网络错误, 如404等
-3	连接cos时发生异常，如连接超时
-71	操作频率过快，触发cos的攻击

JavaScript SDK

开发准备

SDK 获取

COS服务的js sdk v4版本的GitHub下载地址： https://github.com/tencentyun/cos-js-sdk-v4.git

开发环境

1. 使用SDK需要浏览器支持HTML 5
2. 请您到https://console.qcloud.com/cos
   获取您的项目ID(appid)，bucket，secret_id和secret_key。
3. 请您到https://console.qcloud.com/cos 针对您要操作的bucket进行跨域（CORS）设置
   （本版本SDK基于JSON API封装组成）

SDK 配置

直接下载github上提供的源代码，使用SDK之前，加载dist目录里的cos-js-sdk-v4.js文件即可。

```
<script type="text/javascript" src="cos-js-sdk-v4.js"></script>
```

###初始化

```
//初始化逻辑
//特别注意: JS-SDK使用之前请先到console.qcloud.com/cos 对相应的Bucket进行跨域设置
var cos = new CosCloud({
appid: appid,// APPID 必填参数
bucket: bucket,//bucketName 必填参数
region: 'sh',//地域信息 必填参数 华南地区填gz 华东填sh 华北填tj
getAppSign: function (callback) {//获取签名 必填参数

//下面简单讲一下获取签名的几种办法
```

```
//1.搭建一个鉴权服务器，自己构造请求参数获取签名，推荐实际线上业务使用，优点是安全性好，不会暴露自己的私钥
 //拿到签名之后记得调用callback
 /**
 $.ajax('SIGN_URL').done(function (data) {
 var sig = data.sign;
 callback(sig);
 });
 **/

 //2.直接在浏览器前端计算签名，需要获取自己的accessKey和secretKey, 一般在调试阶段使用
 //拿到签名之后记得调用callback
 //var res = getAuth(); //这个函数自己根据签名算法实现
 //callback(res);


 //3.直接复用别人算好的签名字符串, 一般在调试阶段使用
 //拿到签名之后记得调用callback
 //callback('YOUR_SIGN_STR')
 //

 },
 getAppSignOnce: function (callback) {//单次签名，必填参数，参考上面的注释即可
 //填上获取单次签名的逻辑
 }
 });
```

初始化参数说明

参数名	类型	是否必填	默认值	参数描述
appid	int	是	无	appid
bucket	String	是	无	bucket名称，bucket

参数名	类型	是否必填	默认值	参数描述
				创建参见创建Bucket
region	String	是	'gz'	地域信息，必填参数 华南地区填gz 华东填sh 华北填tj
getAppSign	Function	是	无	获取多次签名的函数，建议从服务器端获取签名字符串
getAppSignOnce	Function	是	无	获取单次签名的函数，建议从服务器端获取签名字符串

返回结果说明

返回值：cos object对象，初始化之后可以用这个cos object对象进行内置接口调用例如uploadFile,deleteFile等

文件操作

普通文件上传

接口说明：通常用于较小文件(一般小于20MB)的上传，可以通过此接口上传较小的文件并获得文件的url，如果文件大于20M则本接口内部会去调用分片上传接口。

方法原型

CosCloud.prototype.uploadFile(successCallBack, errorCallBack, progressCallBack, bucket, path, file, insertOnly);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	上传成功的回调
errorCallBack	Function	是	无	上传失败的回调

参数名	类型	是否必填	默认值	参数描述
progressCallBack	Function	是	无	上传过程进度的回调，比如文件1M已经上传了100K则会回调进度0.1
bucket	String	是	无	bucket名称
path	String	是	无	文件在COS服务端的路径
file	File	是	无	本地要上传文件的文件对象（二进制数据）
insertOnly	Int	否	无	insertOnly==0 表示允许覆盖文件 1表示不允许 其他值忽略

返回结果说明(json字符串)

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	提示信息
data	Object	是	返回数据
data.access_url	String	是	生成的文件CDN下载url
data.source_url	String	是	生成的文件COS源站url
data.url	String	是	操作文件的url
data.resource_path	String	是	资源路径. 格式:/appid/bucket/xxx

示例

```
var myFolder = '/111/';//需要操作的目录
var successCallBack = function (result) {
$("#result").val(JSON.stringify(result));
};

var errorCallBack = function (result) {
```

```
result = result || {};
$("#result").val(result.responseText || 'error');
};

var progressCallBack = function(curr){
$("#result").val('uploading... curr progress is '+curr);
};

$('#js-file').off('change').on('change', function (e) {
var file = e.target.files[0];
cos.uploadFile(successCallBack, errorCallBack, progressCallBack, bucket, myFolder+file.name, file, 0);
return false;
});
```

大文件分片上传

接口说明：通常用于较大文件(一般大于20MB)的上传，可以通过此接口大的文件并获得文件的url。

方法原型

CosCloud.prototype.sliceUploadFile(successCallBack, errorCallBack, progressCallBack, bucket, path, file, insertOnly);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	上传成功的回调
errorCallBack	Function	是	无	上传失败的回调
progressCallBack	Function	是	无	上传过程进度的回调，比如文件1M已经上传了100K则会回调进度0.1
bucket	String	是	无	bucket名称

参数名	类型	是否必填	默认值	参数描述
path	String	是	无	文件在COS服务端的路径
file	File	是	无	本地要上传文件的文件对象（二进制数据）
insertOnly	Int	否	无	insertOnly==0 表示允许覆盖文件 1表示不允许 其他值忽略

返回结果说明(json字符串)

需要注意的是大文件上传会经多个接口处理，以下是最后一片上传成功才会触发的回调

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	提示信息
data	Object	是	返回数据
data.access_url	String	是	生成的文件CDN下载url
data.source_url	String	是	生成的文件COS源站url
data.url	String	是	操作文件的url
data.resource_path	String	是	资源路径. 格式:/appid/bucket/xxx

示例

```
var myFolder = '/111/';//需要操作的目录
var successCallBack = function (result) {
$("#result").val(JSON.stringify(result));
};

var errorCallBack = function (result) {
result = result || {};
$("#result").val(result.responseText || 'error');
};
```

```
var progressCallBack = function(curr){
$("#result").val('uploading... curr progress is '+curr);
};

$('#js-file').off('change').on('change', function (e) {
var file = e.target.files[0];
//大文件也可以直接调用uploadFile
cos.uploadFile(successCallBack, errorCallBack, progressCallBack, bucket, myFolder+file.name, file, 0);
//也可以用sliceUploadFile，选一个即可
//cos.sliceUploadFile(successCallBack, errorCallBack, progressCallBack, bucket, myFolder+file.name,
file, 0);
return false;
});
```

删除文件

接口说明：删除文件

方法原型

CosCloud.prototype.deleteFile(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	文件在COS服务端的路径

返回结果说明(json字符串)

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	提示信息

示例

```
//删除文件
$('#deleteFile').on('click', function () {
var myFile = myFolder+'2.txt';//填你自己实际存在的文件
cos.deleteFile(successCallBack, errorCallBack, bucket, myFile);
});
```

获取文件属性

接口说明：通过此接口查询文件的各项属性信息。

方法原型

CosCloud.prototype.getFileStat(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	文件在COS服务端的路径

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Object	是	文件属性数据

参数名	类型	是否必然返回	参数描述
data.name	String	是	文件或目录名
data.biz_attr	String	是	文件属性，业务端维护
data.ctime	String	是	文件的创建时间，unix时间戳
data.mtime	String	是	文件的修改时间，unix时间戳
data.filesize	Int	是	文件大小
data.filelen	Int	是	文件已传输大小
data.sha	String	是	文件文件sha
data.access_url	String	是	生成的文件下载url
data.authority	String	否	eInvalid,eWRPrivate,eWPrivateRPublic,文件可以与bucket拥有不同的权限类型，已经设置过权限的文件如果想要撤销，直接赋值为eInvalid，则会采用bucket的权限
data.custom_headers	String	否	自定义header对象

示例

```
//获取文件属性
$('#getFileStat').on('click', function () {
var myFile = myFolder+'2.txt';//填你自己实际存在的文件
cos.getFileStat(successCallBack, errorCallBack, bucket, myFile);
});
```

更新文件属性

接口说明：通过此接口更新文件的属性信息。

方法原型

CosCloud.prototype.updateFile(successCallBack, errorCallBack, bucket, path, bizAttr);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	文件在COS服务端的路径
bizAttr	String	是	无	文件的自定义属性

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
//更新文件属性
$('#updateFile').on('click', function () {
var myFile = myFolder+'2.txt';//填你自己实际存在的文件
cos.updateFile(successCallBack, errorCallBack, bucket, myFile, 'my new file attr');
});
```

拷贝文件

接口说明：通过此接口把文件拷贝（即复制）到另一个路径。

方法原型

CosCloud.prototype.copyFile(successCallBack, errorCallBack, bucket, path, destPath, overWrite);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要复制的文件在COS服务端的路径
destPath	String	是	无	复制的目标的路径
overWrite	Int	是	无	是否覆盖同名文件，0表示不覆盖，1表示覆盖

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
//拷贝文件，从源文件地址复制一份到新地址
$('#copyFile').on('click', function () {

var myFile = '111/2.txt';//填你自己实际存在的文件

//注意一下目标的路径，这里如果填333/2.txt 则表示文件复制到111/333/2.txt
//如果填/333/2.txt 则表示文件复制到bucket根目录下的333/2.txt
var newFile = '/333/2.txt';
var overWrite = 1;//0 表示不覆盖 1表示覆盖
cos.copyFile(successCallBack, errorCallBack, bucket, myFile, newFile, overWrite);
});
```

移动文件

接口说明：通过此接口把文件移动（剪切）到另一个路径，如果是移动到相同路径的话，可以达到修改文件名的效果。

方法原型

CosCloud.prototype.moveFile(successCallBack, errorCallBack, bucket, path, destPath, overWrite);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要移动的文件在COS服务端的路径
destPath	String	是	无	移动的目标的路径
overWrite	Int	是	无	是否覆盖同名文件，0表示不覆盖，1表示覆盖

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
//移动文件，把源文件移动到新地址，如果是同一个目录移动且文件名不同的话，相当于改了一个文件名
//如果是移动到新目录，相当于剪切当前的文件，粘贴到了新目录
$('#moveFile').on('click', function () {

var myFile = '/111/2.txt';//填你自己实际存在的文件

//注意一下目标的路径，这里如果填333/2.txt 则表示文件移动到111/333/2.txt
//如果填/333/2.txt 则表示文件移动到bucket根目录下的333/2.txt
```

```
//如果填/111/3.txt 则相当于把2.txt改名成3.txt
var newFile = '/333/2.txt';
var overWrite = 1;//0 表示不覆盖 1表示覆盖
cos.moveFile(successCallBack, errorCallBack, bucket, myFile, newFile, overWrite);
});
```

文件夹（目录）操作

新增文件夹

接口说明：通过此接口增加一个指定的文件夹。

方法原型

CosCloud.prototype.createFolder(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要操作的文件夹在COS服务端的路径

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
$('#createFolder').on('click', function () {
```

```
var newFolder = '/333/';//填你需要创建的文件夹，记得用斜杠包一下
cos.createFolder(successCallBack, errorCallBack, bucket, newFolder);
});
```

删除文件夹

接口说明：通过此接口删除指定的文件夹。

方法原型

CosCloud.prototype.deleteFolder(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要操作的文件夹在COS服务端的路径

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
//删除文件夹
$('#deleteFolder').on('click', function () {
var newFolder = '/333/';//填你需要删除的文件夹，记得用斜杠包一下
cos.deleteFolder(successCallBack, errorCallBack, bucket, newFolder);
});
```

获取文件夹属性

接口说明：通过此接口获取指定的文件夹属性。

方法原型

CosCloud.prototype.getFolderStat(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要操作的文件夹在COS服务端的路径

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息
data	Object	是	文件夹属性信息对象
data.biz_attr	String	是	文件夹属性信息字符串

示例

```
//获取文件夹属性
$('#getFolderStat').on('click', function () {
cos.getFolderStat(successCallBack, errorCallBack, bucket, '/333/');
});
```

更新文件夹属性

接口说明：通过此接口更新指定的文件夹属性。

方法原型

CosCloud.prototype.updateFolder(successCallBack, errorCallBack, bucket, path, bizAttr);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要操作的文件夹在COS服务端的路径
bizAttr	String	是	无	新的属性信息

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	错误码，成功时为0
message	String	是	错误信息

示例

```
//更新文件夹属性
$('#updateFolder').on('click', function () {
cos.updateFolder(successCallBack, errorCallBack, bucket, '/333/', 'new attr');
});
```

获取文件夹内列表

接口说明：通过此接口获取指定的文件夹内的文件列表。

方法原型

CosCloud.prototype.getFolderList(successCallBack, errorCallBack, bucket, path);

参数说明

参数名	类型	是否必填	默认值	参数描述
successCallBack	Function	是	无	操作成功的回调
errorCallBack	Function	是	无	操作失败的回调
bucket	String	是	无	bucket名称
path	String	是	无	需要操作的文件夹在COS服务端的路径

返回结果说明(json字符串)：

参数名	类型	是否必然返回	参数描述
code	Int	是	API 错误码，成功时为0
message	String	是	错误信息
data	Array	是	返回数据
data.listover	Bool	是	是否有内容可以继续往前/往后翻页
data.context	String	是	透传字段，查看第一页，则传空字符串。若需要翻页，需要将前一页返回值中的context透传到参数中。order用于指定翻页顺序。若order填0，则从当前页正序/往下翻页；若order填1，则从当前页倒序/往上翻页
data.infos	Array	是	文件、目录集合，可以为空
data.infos.name	String	是	文件或目录名
data.infos.biz_attr	String	是	目录或文件属性，业务端维护
data.infos.ctime	String	是	目录或文件的创建时间，unix时间戳
data.infos.mtime	String	是	目录或文件的修改时间，unix时间戳

参数名	类型	是否必然返回	参数描述
data.infos.filesize	Int	否(当类型为文件时返回)	文件大小
data.infos.filelen	Int	否(当类型为文件时返回)	文件已传输大小(通过与filesize对比可知文件传输进度)
data.infos.sha	String	否(当类型为文件时返回)	文件sha
data.infos.access_url	String	否(当类型为文件时返回)	生成的文件下载url
data.infos.authority	String	否	eInvalid,eWRPrivate,eWPrivateRPublic,文件可以与bucket拥有不同的权限类型，已经设置过权限的文件如果想要撤销，直接赋值为eInvalid，则会采用bucket的权限

示例

```
//获取指定文件夹内的列表,默认每次返回20条
$('#getFolderList').on('click', function () {
cos.getFolderList(successCallBack, errorCallBack, bucket, myFolder);
});
```

基于 XML API 的 SDK

Node.js SDK

开发准备

相关资源

COS服务的Node.js SDK v5版本的GitHub下载地址： https://github.com/tencentyun/cos-nodejs-sdk-v5.git
（本版本SDK基于XML API封装组成）

大部分接口的使用 demo 在这里： demo

npm 引入

```
npm i cos-nodejs-sdk-v5 --save
```

开发环境

1. 使用SDK需要您的运行环境包含nodejs 以及npm，nodejs版本建议7.0版本以上
2. 安装好 npm 之后记得在sdk的解压目录npm install 一次（安装依赖包）；
3. 去您的控制台获取 AppId, SecretId, SecretKey, 地址在 https://console.qcloud.com/capi

SDK配置

```
var COS = require('cos-nodejs-sdk-v5');

var params = {
  AppId: 'STRING_VALUE', /* 必须 */
  SecretId: 'STRING_VALUE', /* 必须 */
  SecretKey: 'STRING_VALUE', /* 必须 */
  FileParallelLimit: 'NUMBER', /* 非必须 */
  ChunkParallelLimit: 'NUMBER', /* 非必须 */
```

```
  ChunkSize: 'NUMBER', /* 非必须 */
  ProgressInterval: 'NUMBER', /* 非必须 */
  Domain: 'STRING_VALUE', /* 非必须 */
};

var cos = new COS(params);
```

操作参数说明

- params (Object) ： 参数列表
  - AppId —— (String) ： 用户的 AppId
  - SecretId —— (String) ： 用户的 SecretId
  - SecretKey —— (String) ： 用户的 SecretKey
  - FileParallelLimit —— (Number) ： 控制文件上传并发数，默认为 3
  - ChunkParallelLimit —— (Number) ： 控制单个文件下分片上传并发数，默认为 3
  - ChunkSize —— (Number) ： 控制分片大小，单位 Byte，默认为 1 1024 1024 ，即 1 MB
  - ProgressInterval —— (Number) ： 控制 onProgress 回调的间隔，单位为 ms，用于防止 onProgress 频繁触发造成的抖动，默认为 1000
  - Domain —— (String) ： 用户的自定义域名，如果设置了自定义域名，则所有对 Bucket 和 Object 的操作请求将发送到自定义域名。默认对 Bucket 的操作域名为 {{Bucket}}-{{AppId}}.cos.{{Region}}.myqcloud.com，例如 annexwu-123456789.cos.ap-beijing-1.myqcloud.com

鉴权操作

Get Auth

功能说明

Get Auth 用于计算鉴权信息（Authorization），用以验证请求合法性的签名信息。

操作方法原型

- 调用 Get Auth 操作

```
var params = {
  Method: 'STRING_VALUE', /* 必须 */
  Key: 'STRING_VALUE', /* 非必须 */
  SecretId: 'STRING_VALUE', /* 非必须 */
  SecretKey: 'STRING_VALUE', /* 非必须 */
};

var Authorization = cos.getAuth(params);
```

操作参数说明

- params (Object) ： 参数列表
  - Method —— (String) ： 操作方法，如 get, post, delete, head 等 HTTP 方法
  - Key —— (String) ： 操作的 object
    名称，如果请求操作是对文件的，则为文件名，且为必须参数。如果操作是对于
    Bucket，则为空
  - SecretId —— (String) ： 用户的 SecretId，如果 SecretId 和 COS
    实例创建时相同，则可以不填
  - SecretKey —— (String) ： 用户的 SecretKey，如果 SecretKey 和 COS
    实例创建时相同，则可以不填

返回值说明

- Authorization —— (String) ： 操作的鉴权签名

Service操作

Get Service

功能说明

Get Service 接口是用来获取请求者名下的所有存储空间列表（Bucket list）。该 API 接口不支持匿名请求，您需要使用带 Authorization 签名认证的请求才能获取 Bucket 列表，且只能获取签名中 AccessID 所属账户的 Bucket 列表。

操作方法原型

- 调用 Get Service 操作

```
cos.getService(function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- 无特殊参数

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Buckets —— (Array) ：说明本次返回的Bucket列表的所有信息
  - Name —— (String) ：Bucket 的名称
  - CreateDate —— (String) ：Bucket 创建时间。ISO8601 格式，例如 2016-11-09T08:46:32.000Z

- Location —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Bucket操作

Head Bucket

功能说明

Head Bucket 请求可以确认该 Bucket 是否存在，是否有权限访问。Head 的权限与 Read 一致。当该 Bucket 存在时，返回 HTTP 状态码 200；当该 Bucket 无访问权限时，返回 HTTP 状态码 403；当该 Bucket 不存在时，返回 HTTP 状态码 404。

操作方法原型

- 调用 Head Bucket 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.headBucket(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 的名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - BucketExist —— (Boolean) ： Bucket 是否存在
  - BucketAuth —— (Boolean) ： 是否拥有该 Bucket 的权限
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Get Bucket

功能说明

Get Bucket 请求等同于 List Object 请求，可以列出该 Bucket 下的部分或者全部 Object。此 API 调用者需要对 Bucket 有 Read 权限。

操作方法原型

- 调用 Get Bucket 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Prefix : 'STRING_VALUE', /* 非必须 */
  Delimiter : 'STRING_VALUE', /* 非必须 */
```

```
  Marker : 'STRING_VALUE', /* 非必须 */
  MaxKeys : 'STRING_VALUE', /* 非必须 */
  EncodingType : 'STRING_VALUE', /* 非必须 */
};

cos.getBucket(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 的名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Prefix —— (String) : 前缀匹配，用来规定返回的文件前缀地址
  - Delimiter —— (String) ：定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
  - Marker —— (String) ：默认以 UTF-8 二进制顺序列出条目，所有列出条目从marker开始
  - MaxKeys —— (String) ：单次返回最大的条目数量，默认1000
  - EncodingType —— (String) ：规定返回值的编码方式，可选值：url

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：

请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。

- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - CommonPrefixes —— (Array) ： 将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix
  - Prefix —— (String) ：单条 Common 的前缀
  - Name —— (String) ： 说明 Bucket 的信息
  - Prefix —— (String) ： 前缀匹配，用来规定返回的文件前缀地址
  - Marker —— (String) ： 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
  - MaxKeys —— (String) ： 单次响应请求内返回结果的最大的条目数量
  - IsTruncated —— (String) ： 响应请求条目是否被截断，字符串，'true' 或者 'false'
  - NextMarker —— (String) ：
    假如返回条目被截断，则返回NextMarker就是下一个条目的起点
  - Encoding-Type —— (String) ：
    返回值的编码方式，作用于Delimiter，Marker，Prefix，NextMarker，Key
  - Contents —— (Array) ： 元数据信息
  - ETag —— (String) ： 文件的 MD-5 算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

    ，注意前后携带双引号
  - Size —— (String) ： 说明文件大小，单位是 Byte
  - Key —— (String) ： Object名称
  - LastModified —— (String) ： 说明 Object 最后被修改时间，如
    2017-06-23T12:33:27.000Z
  - Owner —— (Object) ： Bucket 持有者信息
    - ID —— (String) ： Bucket 的 AppID
  - StorageClass —— (String) ： Object
    的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Put Bucket

功能说明

Put Bucket 接口请求可以在指定账号下创建一个 Bucket。该 API 接口不支持匿名请求，您需要使用带 Authorization 签名认证的请求才能创建新的 Bucket 。创建 Bucket 的用户默认成为 Bucket 的持有者。

操作方法原型

- 调用 Put Bucket 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
  GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putBucket(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
  - GrantRead —— (String) ： 赋予被授权者读的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="q

cs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'

- GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Location —— (String) ： 创建成功后，Bucket 的操作地址
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404 等

Delete Bucket

功能说明

Delete Bucket 接口请求可以在指定账号下删除 Bucket，删除之前要求 Bucket 内的内容为空，只有删除了 Bucket 内的信息，才能删除 Bucket 本身。注意，如果删除成功，则返回的 HTTP 状态码为 200 或 204 。

操作方法原型

- 调用 Delete Bucket 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.deleteBucket(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，204，403，404等

Get Bucket ACL

功能说明

Get Bucket ACL 接口用来获取 Bucket 的 ACL(access control list)，
即用户空间（Bucket）的访问权限控制列表。 此 API 接口只有 Bucket 的持有者有权限操作。

操作方法原型

- 调用 Get Bucket ACL 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketAcl(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Owner —— (Object) ： Bucket 持有者信息
  - DisplayName —— (String) ： Bucket 持有者的名称
  - ID —— (String) ： Bucket 持有者 ID，格式：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
    如果是根帐号，<OwnerUin> 和 <SubUin> 是同一个值
  - Grants —— (Array) ： 被授权者信息与权限信息列表
  - Permission —— (String) ：
    指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL
  - Grantee —— (Object) ： 说明被授权者的信息。type 类型可以为 RootAccount，
    Subaccount；当 type 类型为 RootAccount 时，ID 中指定的是根帐号;当 type 类型为
    Subaccount 时，ID 中指定的是子帐号
    - DisplayName —— (String) ： 用户的名称
    - ID —— (String) ： 用户的
      ID，如果是根帐号，格式为：qcs::cam::uin/<OwnerUin>:uin/<OwnerUin> 或
      qcs::cam::anyone:anyone（指代所有用户）如果是子帐号，格式为：
      qcs::cam::uin/<OwnerUin>:uin/<SubUin>
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Put Bucket ACL

功能说明

Put Bucket ACL 接口用来写入 Bucket 的 ACL 表，您可以通过 Header："x-cos-acl"，"x-cos-grant-read"，"x-cos-grant-write"，"x-cos-grant-full-control" 传入 ACL 信息，或者通过 Body 以 XML 格式传入 ACL 信息。

操作方法原型

- 调用 Put Bucket ACL 操作

```
var params = {
```

```
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
  GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putBucketAcl(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： 地域名称
  - ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
  - GrantRead —— (String) ： 赋予被授权者读的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，

id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ： 请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Get Bucket CORS

功能说明

Get Bucket CORS 接口实现 Bucket 持有者在 Bucket 上进行跨域资源共享的信息配置。（CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin resource sharing））。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

操作方法原型

- 调用 Get Bucket CORS 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketCors(params, function(err, data) {
  if(err) {
  console.log(err);
```

```
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - CORSRules —— (Array) ： 说明跨域资源共享配置的所有信息列表
  - AllowedMethods —— (Array) ： 允许的 HTTP
    操作，枚举值：GET，PUT，HEAD，POST，DELETE
  - AllowedOrigins —— (Array) ： 允许的访问来源，支持通配符 *
    格式为：协议://域名[:端口]如：http://www.qq.com
  - AllowedHeaders —— (Array) ： 在发送 OPTIONS
    请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *
  - ExposeHeaders —— (Array) ： 设置浏览器可以接收到的来自服务器端的自定义头部信息
  - MaxAgeSeconds —— (String) ： 设置 OPTIONS 请求得到结果的有效期
  - ID —— (String) ： 配置规则的 ID

Put Bucket CORS

功能说明

Put Bucket CORS 接口用来请求设置 Bucket 的跨域资源共享权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为64 KB。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

操作方法原型

- 调用 Put Bucket CORS 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  CORSRules : [
  {
  ID : 'STRING_VALUE', /* 非必须 */
  AllowedMethods: [ /* 必须 */
  'STRING_VALUE',
  ...
  ],
  AllowedOrigins: [ /* 必须 */
  'STRING_VALUE',
  ...
  ],
  AllowedHeaders: [ /* 非必须 */
  'STRING_VALUE',
  ...
  ],
  ExposeHeaders: [ /* 非必须 */
  'STRING_VALUE',
  ...
  ],
  MaxAgeSeconds: 'STRING_VALUE' /* 非必须 */
  },
```

```
  ....
  ]
};

cos.putBucketCors(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - CORSRules —— (Array) ：说明跨域资源共享配置的所有信息列表
  - ID —— (String) ： 配置规则的 ID，可选填
  - AllowedMethods —— (Array) ： 允许的 HTTP
    操作，枚举值：GET，PUT，HEAD，POST，DELETE
  - AllowedOrigins —— (Array) ： 允许的访问来源，支持通配符 *
    格式为：协议://域名[:端口]如：http://www.qq.com
  - AllowedHeaders —— (Array) ： 在发送 OPTIONS
    请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，暂不支持通配符 "*"
  - ExposeHeaders —— (Array) ： 设置浏览器可以接收到的来自服务器端的自定义头部信息
  - MaxAgeSeconds —— (String) ： 设置 OPTIONS 请求得到结果的有效期

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Delete Bucket CORS

功能说明

Delete Bucket CORS 接口请求实现删除跨域访问配置信息。

操作方法原型

- 调用 Delete Bucket CORS 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.deleteBucketCors(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，204，403，404等

Get Bucket Location

功能说明

Get Bucket Location 接口用于获取 Bucket 所在的地域信息，该 GET 操作使用 location 子资源返回 Bucket 所在的区域，只有 Bucket 持有者才有该 API 接口的操作权限。

操作方法原型

- 调用 Get Bucket Location 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketLocation(params, function(err, data) {
  if(err) {
```

```
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - LocationConstraint —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Object操作

Head Object

功能说明

Head Object 接口请求可以获取对应 Object 的 meta 信息数据，Head 的权限与 Get 的权限一致

操作方法原型

- 调用 Head Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  IfModifiedSince : 'STRING_VALUE' /* 非必须 */
};

cos.headObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - IfModifiedSince —— (String) ： 当 Object 在指定时间后被修改，则返回对应 Object 的 meta 信息，否则返回 304

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - x-cos-object-type —— (String) ： 用来表示 Object 是否可以被追加上传，枚举值：normal 或者 appendable
  - x-cos-storage-class —— (String) ：Object 的存储级别，枚举值：STANDARD, STANDARD_IA, NEARLINE
  - x-cos-meta-* —— (String) ：用户自定义的 meta
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，304 等，如果在指定时间后未被修改，则返回 304
  - NotModified —— (Boolean) ：Object 是否在指定时间后未被修改

Get Object

功能说明

Get Object 接口请求可以在 COS 的 Bucket 中将一个文件（Object）下载至本地。该操作需要请求者对目标 Object 具有读权限或目标 Object 对所有人都开放了读权限（公有读）。

操作方法原型

- 调用 Get Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  ResponseContentType : 'STRING_VALUE', /* 非必须 */
  ResponseContentLanguage : 'STRING_VALUE', /* 非必须 */
  ResponseExpires : 'STRING_VALUE', /* 非必须 */
  ResponseCacheControl : 'STRING_VALUE', /* 非必须 */
  ResponseContentDisposition : 'STRING_VALUE', /* 非必须 */
  ResponseContentEncoding : 'STRING_VALUE', /* 非必须 */
```

```
  Range : 'STRING_VALUE', /* 非必须 */
  IfModifiedSince : 'STRING_VALUE', /* 非必须 */
  IfUnmodifiedSince : 'STRING_VALUE', /* 非必须 */
  IfMatch : 'STRING_VALUE', /* 非必须 */
  IfNoneMatch : 'STRING_VALUE', /* 非必须 */
  Output : 'WRITE_STREAM', /* 必须 */
  onProgress : function(progressData) { /* 非必须 */
  console.log(JSON.stringify(progressData));
  }
};

cos.getObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - ResponseContentType —— (String) ： 设置响应头部中的 Content-Type 参数
  - ResponseContentLanguage —— (String) ： 设置返回头部中的 Content-Language 参数
  - ResponseExpires —— (String) ： 设置返回头部中的 Content-Expires 参数
  - ResponseCacheControl —— (String) ： 设置返回头部中的 Cache-Control 参数
  - ResponseContentDisposition —— (String) ： 设置返回头部中的 Content-Disposition 参数
  - ResponseContentEncoding —— (String) ： 设置返回头部中的 Content-Encoding 参数
  - Range —— (String) ： RFC 2616 中定义的指定文件下载范围，以字节（bytes）为单位

- IfModifiedSince —— (String) ： 当Object在指定时间后被修改，则返回对应Object meta 信息，否则返回304
- IfUnmodifiedSince —— (String) ：
  如果文件修改时间早于或等于指定时间，才返回文件内容。否则返回 412 (precondition failed)
- IfMatch —— (String) ： 当 ETag 与指定的内容一致，才返回文件。否则返回 412 (precondition failed)
- IfNoneMatch —— (String) ： 当 ETag 与指定的内容不一致，才返回文件。否则返回304 (not modified)
- Output —— (WriteStream) ： 需要输出文件的写流
- onProgress —— (Function) ：
  进度的回调函数，进度回调响应对象（progressData）属性如下
- progressData.loaded —— (Number) ： 已经下载的文件部分大小，以字节（bytes）为单位
- progressData.total —— (Number) ： 整个文件的大小，以字节（bytes）为单位
- progressData.speed —— (Number) ： 文件的下载速度，以字节/秒（bytes/s）为单位
- progressData.percent —— (Number) ： 文件下载的百分比，以小数形式呈现，例如：下载 50% 即为 0.5

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - x-cos-object-type —— (String) ： 用来表示 object 是否可以被追加上传，枚举值：normal 或者 appendable
  - x-cos-storage-class —— (String) ：Object
    的存储级别，枚举值：S
    TANDARD，STANDARD_IA，NEARLINE，
    注意：如果没有返回该头部，则说明文件存储级别为 STANDARD （标准存储）

- x-cos-meta-* —— (String) ：用户自定义的元数据
- NotModified —— (Boolean) ：如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，304，403，404等

Put Object

功能说明

Put Object 接口请求可以将本地的文件（Object）上传至指定 Bucket 中。该操作需要请求者对 Bucket 有 WRITE 权限。

注意，Key(文件名) 不能以

/

结尾，否则会被识别为文件夹

单个 Bucket 下 acl 策略限制 1000 条，因此在单个 bucket 下，最多允许对 999 个文件设置 acl 权限

操作方法原型

- 调用 Put Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  ContentLength : 'STRING_VALUE', /* 必须 */
  CacheControl : 'STRING_VALUE', /* 非必须 */
  ContentDisposition : 'STRING_VALUE', /* 非必须 */
  ContentEncoding : 'STRING_VALUE', /* 非必须 */
  ContentType : 'STRING_VALUE', /* 非必须 */
  Expect : 'STRING_VALUE', /* 非必须 */
```

```
  Expires : 'STRING_VALUE', /* 非必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
  GrantFullControl : 'STRING_VALUE', /* 非必须 */
  StorageClass : 'STRING_VALUE', /* 非必须 */
  'x-cos-meta-*' : 'STRING_VALUE', /* 非必须 */
  Body: 'Buffer || ReadStream || File || Blob', /* 必须 */
  onProgress : function(progressData) { /* 非必须 */
  console.log(JSON.stringify(progressData));
  }
};
cos.putObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - CacheControl —— (String) ： RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
  - ContentDisposition —— (String) ： RFC 2616 中定义的文件名称，将作为 Object 元数据保存
  - ContentEncoding —— (String) ： RFC 2616 中定义的编码格式，将作为 Object 元数据保存
  - ContentLength —— (String) ： RFC 2616 中定义的 HTTP 请求内容长度（字节）
  - ContentType —— (String) ： RFC 2616 中定义的内容类型（MIME），将作为 Object

元数据保存

- Expect —— (String) ： 当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容
- Expires —— (String) ： RFC 2616 中定义的过期时间，将作为 Object 元数据保存
- ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
- GrantRead —— (String) ： 赋予被授权者读的权限。格式：id="",id="" ；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id="",id="" ；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id="",id="" ；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- StorageClass —— (String) ： 设置 Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE，默认值：STANDARD
- x-cos-meta-* —— (String) ： 允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制 2K
- Body —— (Buffer || ReadStream || File || Blob) ： 上传文件的内容或者流
- onProgress —— (Function) ： 进度的回调函数，进度回调响应对象（progressData）属性如下
- progressData.loaded —— (Number) ： 已经下载的文件部分大小，以字节（bytes）为单位
- progressData.total —— (Number) ： 整个文件的大小，以字节（bytes）为单位
- progressData.speed —— (Number) ： 文件的下载速度，以字节/秒（bytes/s）为单位
- progressData.percent —— (Number) ： 文件下载的百分比，以小数形式呈现，例如：下载 50% 即为 0.5

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ： 请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等
  - ETag —— (String) ： 返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 在上传过程中是否有损坏，注意：这里的 ETag 值字符串前后带有双引号，例如

    "09cba091df696af91549de27b8e7d0f6"

Delete Object

功能说明

Delete Object 接口请求可以在 COS 的 Bucket 中将一个文件（Object）删除。该操作需要请求者对 Bucket 有 WRITE 权限。

操作方法原型

- 调用 Delete Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE' /* 必须 */
};

cos.deleteObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
```

```
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如
    200，204，403，404等，如果删除成功或者文件不存在则返回 204 或
    200，如果找不到指定的 Bucket，则返回 404
  - BucketNotFound —— (Boolean) ：如果找不到指定的 Bucket，则为 true

Options Object

功能说明

Options Object 接口实现 Object 跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 HEADER 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回 403 Forbidden。
可以通过 Put Bucket CORS 接口来开启 Bucket 的 CORS 支持

操作方法原型

- 调用 Options Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  Origin : 'STRING_VALUE', /* 必须 */
  AccessControlRequestMethod : 'STRING_VALUE', /* 必须 */
  AccessControlRequestHeaders : 'STRING_VALUE' /* 非必须 */
};

cos.optionsObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - Origin —— (String) ： 模拟跨域访问的请求来源域名
  - AccessControlRequestMethod —— (String) ： 模拟跨域访问的请求 HTTP 方法
  - AccessControlRequestHeaders —— (String) ： 模拟跨域访问的请求头部

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - AccessControlAllowOrigin —— (String) ： 模拟跨域访问的请求来源域名，中间用逗号间隔，当来源不允许的时候，此Header不返回。例如：*
  - AccessControlAllowMethods —— (String) ：模拟跨域访问的请求HTTP方法，中间用逗号间隔，当请求方法不允许的时候，此Header不返回。例如：PUT,GET,POST,DELETE,HEAD
  - AccessControlAllowHeaders —— (String) ：模拟跨域访问的请求头部，中间用逗号间隔，当模拟任何请求头部不允许的时候，此Header不返回该请求头部。例如：accept,content-type,origin,authorization
  - AccessControlExposeHeaders —— (String)
    ：跨域支持返回头部，中间用逗号间隔。例如：ETag
  - AccessControlMaxAge —— (String) ：设置 OPTIONS
    请求得到结果的有效期。例如：3600
  - OptionsForbidden —— (Boolean) ：OPTIONS 请求是否被禁止，如果返回的 HTTP 状态码为 403，则为 true
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Get Object ACL

功能说明

Get Object ACL 接口用来获取某个 Bucket 下的某个 Object 的访问权限。只有 Bucket 的持有者才有权限操作。

操作方法原型

- 调用 Get Object ACL 操作

```
varparams = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE' /* 必须 */
};

cos.getObjectAcl(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Owner —— (Object) ： 标识资源的所有者
  - ID —— (String) ： Object 持有者 ID，格式：qcs::cam::uin/:uin/
    如果是根帐号，<OwnerUin> 和<SubUin> 是同一个值

- DisplayName —— (String) ： Object 持有者的名称
- Grants —— (Array) ： 被授权者信息与权限信息列表
- Permission —— (String) ：
  指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL
- Grantee —— (Object) ： 说明被授权者的信息。type 类型可以为 RootAccount，Subaccount；当 type 类型为 RootAccount 时，ID 中指定的是根帐号;当 type 类型为 Subaccount 时，ID 中指定的是子帐号
  - DisplayName —— (String) ： 用户的名称
  - ID —— (String) ： 用户的
    ID，如果是根帐号，格式为：qcs::cam::uin/<OwnerUin>:uin/<OwnerUin> 或 qcs::cam::anyone:anyone （指代所有用户）如果是子帐号，格式为：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Put Object ACL

功能说明

Put Object ACL 接口用来对某个 Bucket 中的某个的 Object 进行 ACL 表的配置

单个 Bucket 下 acl 策略限制 1000 条，因此在单个 bucket 下，最多允许对 999 个文件设置 acl 权限

操作方法原型

- 调用 Put Object ACL 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
```

```
  GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putObjectAcl(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： 地域名称
  - Key —— (String) ： 文件名称
  - ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
  - GrantRead —— (String) ： 赋予被授权者读的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Delete Multiple Object

功能说明

Delete Multiple Object 接口请求实现在指定 Bucket 中批量删除 Object，单次请求最大支持批量删除 1000 个 Object。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式：Verbose 模式将返回每个 Object 的删除结果；Quiet 模式只返回报错的 Object 信息。

操作方法原型

- 调用 Delete Multiple Object 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Quiet : 'BOOLEAN_VALUE', /* 非必须 */
  Objects : [ /* 必须 */
  {
  Key : 'STRING_VALUE' /* 必须 */
  },
  ...
  ]
};
```

```
cos.deleteMultipleObject(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Quiet —— (Boolean) ： 布尔值，这个值决定了是否启动 Quiet 模式。值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false
  - Objects —— (Array) ： 要删除的文件列表
  - Key —— (String) ： 要删除的文件名

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Deleted —— (Array) ： 说明本次删除的成功Object信息
  - Key —— (String) ： Object的名称
  - Error —— (Array) ： 说明本次删除的失败Object信息
  - Key —— (String) ： Object的名称
  - Code —— (String) ： 删除失败的错误码

- Message —— (String) ： 删除错误信息
- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Put Object Copy

功能说明

Put Object Copy 请求实现将一个文件从源路径复制到目标路径。建议文件大小 1M 到 5G，超过 5G 的文件请使用分块上传 Upload - Copy。在拷贝的过程中，文件元属性和 ACL 可以被修改。用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

操作方法原型

- 调用 Put Object Copy 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  CopySource : 'STRING_VALUE', /* 必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
  GrantFullControl : 'STRING_VALUE', /* 非必须 */
  MetadataDirective : 'STRING_VALUE', /* 非必须 */
  CopySourceIfModifiedSince : 'STRING_VALUE', /* 非必须 */
  CopySourceIfUnmodifiedSince : 'STRING_VALUE', /* 非必须 */
  CopySourceIfMatch : 'STRING_VALUE', /* 非必须 */
  CopySourceIfNoneMatch : 'STRING_VALUE', /* 非必须 */
  StorageClass : 'STRING_VALUE', /* 非必须 */
  CacheControl : 'STRING_VALUE', /* 非必须 */
  ContentDisposition : 'STRING_VALUE', /* 非必须 */
  ContentEncoding : 'STRING_VALUE', /* 非必须 */
```

```
  ContentType : 'STRING_VALUE', /* 非必须 */
  Expect : 'STRING_VALUE', /* 非必须 */
  Expires : 'STRING_VALUE', /* 非必须 */
  'x-cos-meta-*' : 'STRING_VALUE', /* 非必须 */
};

cos.putObjectCopy(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - CopySource —— (String) ： 源文件 URL 路径，可以通过 versionid 子资源指定历史版本
  - ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
  - GrantRead —— (String) ： 赋予被授权者读的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
  - GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，

id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'

- MetadataDirective —— (String) ： 是否拷贝元数据，枚举值：Copy, Replaced，默认值 Copy。假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；假如标记为 Replaced，按 Header 信息修改元数据。当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
- CopySourceIfModifiedSince —— (String) ： 当 Object 在指定时间后被修改，则执行操作，否则返回 412。可与 CopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突
- CopySourceIfUnmodifiedSince —— (String) ： 当 Object 在指定时间后未被修改，则执行操作，否则返回 412。可与 CopySourceIfMatch 一起使用，与其他条件联合使用返回冲突
- CopySourceIfMatch —— (String) ： 当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412。可与CopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突
- CopySourceIfNoneMatch —— (String) ： 当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412。可与 CopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突
- StorageClass —— (String) ： 存储级别，枚举值：存储级别，枚举值：Standard, Standard_IA，Nearline；默认值：Standard
- x-cos-meta-* —— (String) ： 其他自定义的文件头部
- CacheControl —— (String) ： 指定所有缓存机制在整个请求/响应链中必须服从的指令
- ContentDisposition —— (String) ： MIME 协议的扩展，MIME 协议指示 MIME 用户代理如何显示附加的文件
- ContentEncoding —— (String) ： HTTP 中用来对「采用何种编码格式传输正文」进行协定的一对头部字段
- ContentType —— (String) ： RFC 2616 中定义的 HTTP 请求内容类型（ MIME ），例如text/plain
- Expect —— (String) ： 请求的特定的服务器行为
- Expires —— (String) ： 响应过期的日期和时间

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - ETag —— (String) ： 文件的 MD-5 算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

    ，注意前后携带双引号
  - LastModified —— (String) ： 说明 Object 最后被修改时间，如
    2017-06-23T12:33:27.000Z
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

分块上传操作

Initiate Multipart Upload

功能说明

Initiate Multipart Upload请求实现初始化分片上传，成功执行此请求以后会返回Upload ID用于后续的Upload Part请求

操作方法原型

- 调用 Initiate Multipart Upload 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  CacheControl : 'STRING_VALUE', /* 非必须 */
  ContentDisposition : 'STRING_VALUE', /* 非必须 */
```

```
  ContentEncoding : 'STRING_VALUE', /* 非必须 */
  ContentType : 'STRING_VALUE', /* 非必须 */
  Expires : 'STRING_VALUE', /* 非必须 */
  ACL : 'STRING_VALUE', /* 非必须 */
  GrantRead : 'STRING_VALUE', /* 非必须 */
  GrantWrite : 'STRING_VALUE', /* 非必须 */
  GrantFullControl : 'STRING_VALUE', /* 非必须 */
  StorageClass : 'STRING_VALUE', /* 非必须 */
  'x-cos-meta-*' : 'STRING_VALUE' /* 非必须 */
};

cos.multipartInit(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - CacheControl —— (String) ： RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
  - ContentDisposition —— (String) ： RFC 2616 中定义的文件名称，将作为 Object 元数据保存
  - ContentEncoding —— (String) ： RFC 2616 中定义的编码格式，将作为 Object 元数据保存
  - ContentType —— (String) ： RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存
  - Expires —— (String) ： RFC 2616 中定义的过期时间，将作为 Object 元数据保存

- ACL —— (String) ： 定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
- GrantRead —— (String) ： 赋予被授权者读的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- GrantWrite —— (String) ： 赋予被授权者写的权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- GrantFullControl —— (String) ： 赋予被授权者读写权限。格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"，当需要给根账户授权时，id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"，例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'
- StorageClass —— (String) ： 设置Object的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE，默认值：STANDARD
- x-cos-meta-* —— (String) ： 允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制2K

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Bucket —— (String) ： 分片上传的目标 Bucket
  - Key —— (String) ： Object 的名称
  - UploadId —— (String) ： 在后续上传中使用的 ID

Upload Part

功能说明

Upload Part 接口请求实现在初始化以后的分块上传，支持的块的数量为1到10000，块的大小为1 MB 到5 GB。
使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。在每次请求 Upload Part 时候，需要携带 partNumber 和 uploadId，partNumber为块的编号，支持乱序上传。
当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回 404 错误，NoSuchUpload.

操作方法原型

- 调用 Upload Part 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  ContentLength : 'STRING_VALUE', /* 必须 */
  PartNumber : 'STRING_VALUE', /* 必须 */
  UploadId : 'STRING_VALUE', /* 必须 */
  Expect : 'STRING_VALUE', /* 非必须 */
  ContentMD5 : 'STRING_VALUE', /* 非必须 */
};

cos.multipartUpload(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - ContentLength —— (String) ： RFC 2616 中定义的 HTTP 请求内容长度（字节）
  - PartNumber —— (String) ： 分块的编号
  - UploadId —— (String) ： 上传任务编号
  - Expect —— (String) ： 当使用 Expect: 100-continue
    时，在收到服务端确认后，才会发送请求内容
  - ContentMD5 —— (String) ： RFC 1864 中定义的经过Base64编码的128-bit 内容 MD5
    校验值。此头部用来校验文件内容是否发生变化

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - ETag —— (String) ： 文件的 MD-5 算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

    ，注意前后携带双引号
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Complete Multipart Upload

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。
由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回 200 的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS会在 Body 中返回合并后块的内容。
当上传块小于 1 MB 的时候，在调用该 API 时，会返回 400 EntityTooSmall；
当上传块编号不连续的时候，在调用该 API 时，会返回 400 InvalidPart；
当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回 400 InvalidPartOrder；
当 UploadId 不存在的时候，在调用该 API 时，会返回 404 NoSuchUpload。

操作方法原型

- 调用 Complete Multipart Upload 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  UploadId : 'STRING_VALUE', /* 必须 */
  Parts : [
  {
  PartNumber : 'STRING_VALUE', /* 必须 */
  ETag : 'STRING_VALUE' /* 必须 */
  },
  ...
  ]
};

cos.multipartComplete(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
```

```
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - UploadId —— (String) ： 上传任务编号
  - Parts —— (Array) ： 用来说明本次分块上传中块的信息列表
  - PartNumber —— (String) ： 分块的编号
  - ETag —— (String) ： 每个块文件的 MD5 算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

    ，注意前后携带双引号

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Location —— (String) ： 创建的Object的外网访问域名
  - Bucket —— (String) ： 分块上传的目标Bucket
  - Key —— (String) ： Object的名称
  - ETag —— (String) ： 合并后文件的 MD5算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

，注意前后携带双引号

- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

List Parts

功能说明

List Parts 用来查询特定分块上传中的已上传的块，即罗列出指定 UploadId 所属的所有已上传成功的分块。

操作方法原型

- 调用 List Parts 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  UploadId : 'STRING_VALUE', /* 必须 */
  EncodingType : 'STRING_VALUE', /* 非必须 */
  MaxParts : 'STRING_VALUE', /* 非必须 */
  PartNumberMarker : 'STRING_VALUE' /* 非必须 */
};

cos.multipartListPart(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - UploadId —— (String) ： 标识本次分块上传的 ID。使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID
    不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。
  - EncodingType —— (String) ： 规定返回值的编码方式
  - MaxParts —— (String) ： 单次返回最大的条目数量，默认 1000
  - PartNumberMarker —— (String) ： 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Bucket —— (String) ： 分块上传的目标 Bucket
  - Encoding-type —— (String) ： 规定返回值的编码方式
  - Key —— (String) ： Object 的名称
  - UploadId —— (String) ： 标识本次分块上传的 ID
  - Initiator —— (Object) ： 用来表示本次上传发起者的信息
  - DisplayName —— (String) ： 上传发起者的名称
  - ID —— (String) ： 上传发起者 ID，格式：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
    如果是根帐号，<OwnerUin> 和 <SubUin> 是同一个值
  - Owner —— (Object) ： 用来表示这些分块所有者的信息
  - DisplayName —— (String) ： Bucket 持有者的名称
  - ID —— (String) ： Bucket 持有者 ID，一般为用户的 UIN
  - StorageClass —— (String) ：
    用来表示这些分块的存储级别，枚举值：Standard，Standard_IA，Nearline

- PartNumberMarker —— (String) ： 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
- NextPartNumberMarker —— (String) ： 假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点
- MaxParts —— (String) ： 单次返回最大的条目数量
- IsTruncated —— (String) ： 返回条目是否被截断，'true' 或者 'false'
- Part —— (Array) ： 分块信息列表
- PartNumber —— (String) ： 块的编号
- LastModified —— (String) ： 块最后修改时间
- ETag —— (String) ： 块的MD5算法校验值
- Size —— (String) ： 块大小，单位 Byte
- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Abort Multipart Upload

功能说明

Abort Multipart Upload 用来实现舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回 404 NoSuchUpload。

建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

操作方法原型

- 调用 Abort Multipart Upload 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Key : 'STRING_VALUE', /* 必须 */
  UploadId : 'STRING_VALUE' /* 必须 */
};
```

```
cos.multipartAbort(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： 文件名称
  - UploadId —— (String) ： 标识本次分块上传的 ID。使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID
    不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

List Multipart Uploads

功能说明

List Multiparts Uploads用来查询正在进行中的分块上传。单次最多列出1000个正在进行中的分块上传。

操作方法原型

- 调用 List Multipart Uploads 操作

```
var params = {
  Bucket : 'STRING_VALUE', /* 必须 */
  Region : 'STRING_VALUE', /* 必须 */
  Delimiter : 'STRING_VALUE', /* 非必须 */
  EncodingType : 'STRING_VALUE', /* 非必须 */
  Prefix : 'STRING_VALUE', /* 非必须 */
  MaxUploads : 'STRING_VALUE', /* 非必须 */
  KeyMarker : 'STRING_VALUE', /* 非必须 */
  UploadIdMarker : 'STRING_VALUE' /* 非必须 */
};

cos.multipartList(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Delimiter —— (String) ： 定界符为一个符号，对 Object 名字包含指定前缀且第一次出现

delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始

- EncodingType —— (String) ： 规定返回值的编码格式，合法值：url
- Prefix —— (String) ： 限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix
- MaxUploads —— (String) ：设置最大返回的 multipart 数量，合法取值从1到1000，默认1000
- KeyMarker —— (String) ： 与 upload-id-marker 一起使用，当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出，当upload-id-marker被指定时，ObjectName 字母顺序大于key-marker的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。当upload-id-marker未被指定时，ObjectName字母顺序大于key-marker的条目将被列出，当upload-id-marker被指定时，ObjectName字母顺序大于key-marker的条目被列出，ObjectName字母顺序等于key-marker同时UploadID大于upload-id-marker的条目将被列出
- UploadIdMarker —— (String) ： 与 key-marker 一起使用，当 key-marker 未被指定时，upload-id-marker 将被忽略，当 key-marker 被指定时，ObjectName字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。当key-marker未被指定时，upload-id-marker将被忽略，当key-marker被指定时，ObjectName字母顺序大于key-marker的条目被列出，ObjectName字母顺序等于key-marker同时UploadID大于upload-id-marker的条目将被列出

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Bucket —— (String) ： 分块上传的目标 Bucket
  - Encoding-Type —— (String) ： 规定返回值的编码格式，合法值：url
  - KeyMarker —— (String) ： 列出条目从该 key 值开始

- UploadIdMarker —— (String) ： 列出条目从该 UploadId 值开始
- NextKeyMarker —— (String) ： 假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点
- NextUploadIdMarker —— (String) ： 假如返回条目被截断，则返回 UploadId 就是下一个条目的起点
- MaxUploads —— (String) ： 设置最大返回的 multipart 数量，合法取值从 1 到 1000
- IsTruncated —— (String) ： 返回条目是否被截断，'true' 或者 'false'
- Delimiter —— (String) ： 定界符为一个符号，对 object 名字包含指定前缀且第一次出现 delimiter 字符之间的object作为一组元素：common prefix。如果没有 prefix，则从路径起点开始
- Prefix —— (String) ： 限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix
- CommonPrefixs —— (Array) ： 将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix
- Prefix —— (String) ： 显示具体的CommonPrefixs
- Upload —— (Array) ： Upload的信息集合
- Key —— (String) ： Object的名称
- UploadId —— (String) ： 标示本次分块上传的 ID
- StorageClass —— (String) ： 用来表示分块的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE
- Initiator —— (Object) ： 用来表示本次上传发起者的信息
  - DisplayName —— (String) ： 上传发起者的名称
  - ID —— (String) ： 上传发起者 ID，格式：qcs::cam::uin/<OwnerUin>:uin/<SubUin> 如果是根帐号，<OwnerUin> 和 <SubUin> 是同一个值
- Owner —— (Object) ： 用来表示这些分块所有者的信息
  - DisplayName —— (String) ： Bucket 持有者的名称
  - ID —— (String) ： Bucket 持有者 ID，格式：qcs::cam::uin/<OwnerUin>:uin/<SubUin> 如果是根帐号，<OwnerUin> 和 <SubUin> 是同一个值
- Initiated —— (String) ： 分块上传的起始时间
- headers —— (Object)： 请求返回的头部信息
- statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

分块上传任务操作

该类方法是对上面原生方法的封装，实现了分块上传的全过程，支持并发分块上传，支持断点续传，支持上传任务的取消，暂停和重新开始等。

Slice Upload File

功能说明

Slice Upload File 可用于实现文件的分块上传。

操作方法原型

- 调用 Slice Upload File 操作

```
var params = {
  Bucket: 'STRING_VALUE', /* 必须 */
  Region: 'STRING_VALUE', /* 必须 */
  Key: 'STRING_VALUE', /* 必须 */
  FilePath: 'STRING_VALUE', /* 必须 */
  SliceSize: 'STRING_VALUE', /* 非必须 */
  StorageClass: 'STRING_VALUE', /* 非必须 */
  AsyncLimit: 'NUMBER', /* 非必须 */
  TaskReady: function(taskId) { /* 非必须 */
  console.log(taskId);
  },
  onHashProgress: function (progressData) { /* 非必须 */
  console.log(JSON.stringify(progressData));
  },
  onProgress: function (progressData) { /* 非必须 */
  console.log(JSON.stringify(progressData));
  }
};
```

```
cos.sliceUploadFile(params, function(err, data) {
  if(err) {
  console.log(err);
  } else {
  console.log(data);
  }
});
```

操作参数说明

- params (Object) ： 参数列表
  - Bucket —— (String) ： Bucket 名称
  - Region —— (String) ： Bucket 所在区域。枚举值请见：Bucket 地域信息
  - Key —— (String) ： Object名称
  - FilePath —— (String) ： 本地文件路径
  - SliceSize —— (String) ： 分块大小
  - AsyncLimit —— (String) ： 分块的并发量
  - StorageClass —— (String) ： Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE
  - TaskReady —— (Function) ： 上传任务创建时的回调函数，返回一个taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）
  - taskId —— (String) ： 上传任务的编号
  - onHashProgress —— (Function) ： 计算文件MD5值的进度回调函数，回调参数为进度对象 progressData
  - progressData.loaded —— (Number) ： 已经校验的文件部分大小，以字节（bytes）为单位
  - progressData.total —— (Number) ： 整个文件的大小，以字节（bytes）为单位
  - progressData.speed —— (Number) ： 文件的校验速度，以字节/秒（bytes/s）为单位
  - progressData.percent —— (Number) ： 文件的校验百分比，以小数形式呈现，例如：下载 50% 即为 0.5
  - onProgress —— (Function) ： 上传文件的进度回调函数，回调参数为进度对象 progressData
  - progressData.loaded —— (Number) ： 已经上传的文件部分大小，以字节（bytes）为单位

- progressData.total —— (Number) ： 整个文件的大小，以字节（bytes）为单位
- progressData.speed —— (Number) ： 文件的上传速度，以字节/秒（bytes/s）为单位
- progressData.percent —— (Number) ： 文件的上传百分比，以小数形式呈现，例如：下载50% 即为 0.5

回调函数说明

function(err, data) { ... }

回调参数说明

- err —— (Object) ：
  请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空。
- data —— (Object)： 请求成功时返回的对象，如果请求发生错误，则为空。
  - Location —— (String) ： 创建的Object的外网访问域名
  - Bucket —— (String) ： 分块上传的目标Bucket
  - Key —— (String) ： Object的名称
  - ETag —— (String) ： 合并后文件的 MD5算法校验值，如

    "22ca88419e2ed4721c23807c678adbe4c08a7880"

    ，注意前后携带双引号
  - headers —— (Object)： 请求返回的头部信息
  - statusCode —— (Number)： 请求返回的 HTTP 状态码，如 200，403，404等

Cancel Task

功能说明

根据 taskId 取消分块上传任务

操作方法原型

- 调用 Cancel Task 操作

```
var taskId = 'xxxxx'; /* 必须 */

cos.cancelTask(taskId);
```

操作参数说明

- taskId (String) ：文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId

Pause Task

功能说明

根据 taskId 暂停分块上传任务

操作方法原型

- 调用 Pause Task 操作

```
var taskId = 'xxxxx'; /* 必须 */

cos.pauseTask(taskId);
```

操作参数说明

- taskId (String) ：文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId

Restart Task

功能说明

根据 taskId 重新开始上传任务，可以用于开启用户手动停止的（调用 pauseTask

停止）或者因为上传错误而停止的上传任务。

操作方法原型

- 调用 Restart Task 操作

```
var taskId = 'xxxxx'; /* 必须 */

cos.restartTask(taskId);
```

操作参数说明

- taskId (String) ：文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId

Java SDK

开发准备

相关资源

v5 版本 COS Java SDK 相关资源地址：github项目

环境依赖

v5 版本 COS Java SDK 适用于 JDK 1.7 及以上版本。

安装SDK

安装 SDK 的方式有两种：maven 安装和源码安装。

- maven 安装
  在 maven 工程中使用 pom.xml 添加相关依赖，内容如下：

```
<dependency>
  <groupId>com.qcloud</groupId>
  <artifactId>cos_api</artifactId>
  <version>5.1.9</version>
</dependency>
```

- 源码安装
  从 github 项目资源中下载源码, 通过 maven 导入项目中。比如 eclipse，选择 File->Import->maven->Existing Maven Projects，确认即可。

卸载SDK

卸载 SDK的方式即删除 pom 依赖或源码。

快速入门

```
// 1 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("10000", "secret_id", secret_key);
// 2 设置 Bucket 的区域, XML的区域详细信息请参见
可用地域(https://www.qcloud.com/document/product/436/6224) 文档
ClientConfig clientConfig = new ClientConfig(new Region("cn-north"));
// 3 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
// 操作 API， 如下文所述.或者参照 Demo(https://github.com/tencentyun/cos-java-sdk-
v5/blob/master/src/main/java/com/qcloud/cos/demo/Demo.java)
```

基本 API 描述

COS XML API JAVA SDK 操作成功会返回每种 API 对应的返回类型，失败会报出异常(CosClientException 和 CosServiception)。异常类型请见后文异常描述。

创建 Bucket

新创建一个当前账户下的 COS 中不存在的 Bucket (创建一个新 Bucket)。Bucket 是有限的资源，Bucket 不等同于目录, 且 Bucket 下的文件（ Object ）数量无限，建议不要创建大量的 Bucket。

方法原型

```
public Bucket createBucket(CreateBucketRequest createBucketRequest)
  throws CosClientException, CosServiceException;
```

参数说明

参数名	类型	默认值	参数描述
createBucketRequest	CreateBucketRequest	无	创建 Bucket 请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket名称，由数字和小写字母构成
cannedAcl	CannedAccessControlList	无	set 方法	批量的权限设置，如私有读写, 私有写公有读,公有读写
accessControlList	AccessControlList	无	set 方法	权限设置

成功返回值

返回值类型	返回值描述
Bucket	有关 Bucket 的描述(如 Bucket 的名称, owner 和创建日期)

示例

```
String bucketName = "movie";
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucketName);
createBucketRequest.setCannedAcl(CannedAccessControlList.PublicReadWrite);
Bucket bucket = cosClient.createBucket(createBucketRequest);
```

删除Bucket

在当前账户下的 COS 中删除一个 Bucket，该 Bucket 需为空，即 Bucket 下没有任何的文件（Object）。

方法原型

```
public void deleteBucket(DeleteBucketRequest deleteBucketRequest) throws CosClientException,
CosServiceException;
```

参数说明

参数名	类型	默认值	参数描述
deleteBucketRequest	DeleteBucketRequest	无	删除 Bucket 请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称,由数字和小写字母构成

成功返回值

该方法无返回值。

示例

```
String bucketName = "movie";
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest(bucketName);
cosClient.deleteBucket(deleteBucketRequest);
```

简单文件上传

将本地文件或者已知长度的输入流内容上传到 COS 的 Bucket 中。推荐用于图片类小文件上传(20MB以下),最大支持文件长度 5GB， 5GB 以上的文件上传请使用分块文件上传方式上传。

方法原型

```
public PutObjectResult putObject(PutObjectRequest putObjectRequest)
  throws CosClientException, CosServiceException;
public PutObjectResult putObject(String bucketName, String key, File file)
  throws CosClientException, CosServiceException;
public PutObjectResult putObject(String bucketName, String key, InputStream input,
  ObjectMetadata metadata) throws CosClientException, CosServiceException;
```

参数说明

参数名	参数类型	默认值	参数描述
putObjectRequest	PutObjectRequest	无	上传文件请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或set方法	Bucket 名称
key	String	无	构造函数或set方法	COS 的文件路径, 即从 Bucket 开始
file	File	无	构造函数或set方法	本地文件
input	InputStream	无	构造函数或set方法	输入流
metadata	ObjectMetadata	无	构造函数或set方法	文件的元信息

成功返回值

返回值类型	返回值描述
PutObjectResult	文件上传成功后属性信息, 如 ETag

示例

```
// 本地文件上传
File localFile = new File("/data/test.txt");
String key = "/aaa.txt";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);

// 从输入流上传(需提前告知输入流的长度, 否则可能导致oom)
FileInputStream fileInputStream = new FileInputStream(localFile);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentLength(500); // 设置长度为500
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, fileInputStream,
objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
// 关闭输入流...
// ....
```

分块文件上传

分块文件上传是通过将文件拆分成多个小块进行上传，多个小块可以并发上传, 最大支持 40TB。

分块文件上传的步骤为:

1. 初始化分块上传，获取 uploadId 。(initiateMultipartUpload)
2. 分块数据上传（可并发）。 (uploadPart)
3. 完成分块上传。 (completeMultipartUpload)

另外在分块文件上传过程中还包含获取已上传分块(listParts)和终止分块上传(abortMultipartUpload)。分块上传的步骤较多，建议使用后文的高级 API 上传方式上传大型文件。

方法原型

```
// 初始化分块上传
public InitiateMultipartUploadResult initiateMultipartUpload(
  InitiateMultipartUploadRequest request) throws CosClientException, CosServiceException;
// 上传数据分块
public UploadPartResult uploadPart(UploadPartRequest uploadPartRequest)
  throws CosClientException, CosServiceException;
// 完成分块上传
public CompleteMultipartUploadResult completeMultipartUpload(
  CompleteMultipartUploadRequest request) throws CosClientException, CosServiceException;
// 罗列已上传分块
public PartListing listParts(ListPartsRequest request)
  throws CosClientException, CosServiceException;
// 终止分块上传
public void abortMultipartUpload(AbortMultipartUploadRequest request)
  throws CosClientException, CosServiceException;
```

示例

```
// 初始化分块
InitiateMultipartUploadRequest initRequest =
  new InitiateMultipartUploadRequest(bucketName, key);
InitiateMultipartUploadResult initResponse = cosClient.initiateMultipartUpload(initRequest);
String uploadId = initResponse.getUploadId()
```

```
// 上传分块, 最多1000个分块, 分块大小支持为1M * 5G.
// 分块大小设置为4M. 如果总计n个分块, 则1~n-1的分块大小一致, 最后一块小于等于前面的分块大小
List<PartETag> partETags = new ArrayList<PartETag>();
int partNumber = 1;
int partSize = 4 * 1024 * 1024;
// partStream代表part数据的输入流, 流长度为partSize
UploadPartRequest uploadRequest = new UploadPartRequest().withBucketName(bucketName).
 withUploadId(uploadId).withKey(key).withPartNumber(partNumber).
  withInputStream(partStream).withPartSize(partSize);
UploadPartResult uploadPartResult = cosClient.uploadPart(uploadRequest);
String eTag = uploadPartResult.getETag(); // 获取part的Etag
partETags.add(new PartETag(partNumber, eTag)); // partETags记录所有已上传的part的Etag信息
// ... 上传partNumber第2个到第n个分块

// complete 完成分块上传.
CompleteMultipartUploadRequest compRequest = new
CompleteMultipartUploadRequest(bucketName, key,
  uploadId, partETags);
CompleteMultipartUploadResult result = cosClient.completeMultipartUpload(compRequest);

// ListPart用于在完成分块上传前或者终止分块上传前获取uploadId对应的已上传的分块信息,
可以用来构造partEtags
ListPartsRequest listPartsRequest = new ListPartsRequest(bucket, key, uploadId);
do {
  partListing = cosclient.listParts(listPartsRequest);
  for (PartSummary partSummary : partListing.getParts()) {
  partETags.add(new PartETag(partSummary.getPartNumber(), partSummary.getETag()));
  }
  listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());

// abortMultipartUpload 用于终止一个分块上传
AbortMultipartUploadRequest abortMultipartUploadRequest =
  new AbortMultipartUploadRequest(bucket, key, uploadId);
```

cosclient.abortMultipartUpload(abortMultipartUploadRequest);

下载文件

将指定 Bucket 中的文件（Object）下载到本地或者获取下载文件下载输入流。

方法原型

```
// 下载文件，并获取输入流
public COSObject getObject(GetObjectRequest getObjectRequest)
  throws CosClientException, CosServiceException;
// 下载文件到本地.
public ObjectMetadata getObject(GetObjectRequest getObjectRequest, File destinationFile)
  throws CosClientException, CosServiceException;
```

参数说明

参数名	参数类型	默认值	参数描述
getObjectRequest	GetObjectRequest	无	下载文件请求
destinationFile	File	无	本地的保存文件

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
key	String	无	构造函数或 set 方法	COS 的文件路径，即从 Bucket 开始
range	long[]	无	set 方法	下载的 range 范围

成功返回值

返回值类型	返回值描述
COSObject	文件的输入流以及元信息
ObjectMetadata	文件相关的属性信息

示例

```
// 下载文件到本地
File downFile = new File("src/test/resources/len5M_down.txt");
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
ObjectMetadata downObjectMeta = cosClient.getObject(getObjectRequest, downFile);

// 下载文件(获取输入流)
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
COSObject cosObject = cosClient.getObject(getObjectRequest);
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();
```

删除文件

删除当前账户下 COS 中 Bucket 上的文件（Object）。

方法原型

```
// 删除文件
public void deleteObject(DeleteObjectRequest deleteObjectRequest)
  throws CosClientException, CosServiceException;
```

参数说明

参数名	参数类型	默认值	参数描述
deleteObjectRequest	DeleteObjectRequest	无	删除文件请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
key	String	无	构造函数或 set 方法	COS 的文件路径, 从 Bucket 开始

成功返回值

该方法无返回值。

示例

```
// 删除当前账户下 COS 中 Bucket 文件
DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest(bucketName, key);
cosClient.deleteObject(deleteObjectRequest);
```

获取文件属性

查询获取当前账户下的 COS 中 Bucket 上的文件（Object）属性。

方法原型

```
// 获取文件属性
public ObjectMetadata getObjectMetadata(GetObjectMetadataRequest getObjectMetadataRequest)
  throws CosClientException, CosServiceException;
```

参数说明

参数名	参数类型	默认值	参数描述
getObjectMetadataRequest	GetObjectMetadataRequest	无	获取文件属性请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
key	String	无	构造函数或 set 方法	COS 的文件路径，从 Bucket 开始

成功返回值

返回值类型	返回值描述
ObjectMetadata	文件相关的属性信息

示例

```
// 获取文件属性
GetObjectMetadataRequest getObjectMetadataRequest =
  new GetObjectMetadataRequest(bucketName, key);
ObjectMetadata statObjectMeta = cosClient.getObjectMetadata(getObjectMetadataRequest);
```

获取文件列表

查询获取当前账户下的 COS 中 Bucket 上的文件（Object）成员列表。

方法原型

```
// 获取文件列表
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest)
  throws CosClientException, CosServiceException;
```

参数说明

参数名	参数类型	默认值	参数描述
listObjectsRequest	ListObjectsRequest	无	获取文件列表请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
prefix	String	" "	构造函数或 set 方法	标记 list 以 prefix 为前缀的成员，默认不进行限制，即 Bucket 下所有的成员
marker	String	无	构造函数或 set 方法	标记 list 的起点位置，第一次为空，后续以上一次 list 的返回值中的 marker
delimiter	String	无	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter

Request 对象成员变量	类型	默认值	设置方法	描述
				一次出现的结束的路径
maxKeys	Integer	1000	构造函数或 set 方法	最大返回的成员个数(不得超过 1000)

成功返回值

返回值类型	返回值描述
ObjectListing	包含 list 的子成员

示例

```
// 获取 Bucket 下成员(设置 delimiter)
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
listObjectsRequest.setBucketName(bucketName);
// 设置 delimiter为 /, 即获取的是直接成员，不包含目录下的递归子成员
listObjectsRequest.setDelimiter("/");
ObjectListing objectListing = cosClient.listObjects(listObjectsRequest);
List<COSObjectSummary> objectListSummary = objectListing.getObjectSummaries();
```

生成签名

COSSigner 提供构造 COS 签名的方法，用于给移动端分发签名，构造下载链接等。

方法原型

```
// 构造签名
public String buildAuthorizationStr(HttpMethodName methodName, String resourcePath,
  Map<String, String> headerMap, Map<String, String> paramMap, COSCredentials cred);
```

参数说明

参数名	参数类型	默认值	参数描述

参数名	参数类型	默认值	参数描述
methodName	HttpMethodName	无	http 请求方法(PUT, GET, DELETE, HEAD, POST)
resourcePath	String	无	路径地址
headerMap	Map<String, String>	无	要签发的 HTTP头部
paramMap	Map<String, String>	无	URL 路径中的参数 KV 对
cred	COSCredentials	无	身份信息

成功返回值

返回值类型	返回值描述
String	签名

示例

```
// 以下用于生成带下载签名的链接
COSSigner signer = new COSSigner();
signer.setSignExpiredTime(1800); // 设置签名有效时间为1800秒，默认是3600秒
// 生成签名
String origin_auth_str = signer.buildAuthorizationStr(HttpMethodName.GET, key, cred);
// 对签名进行URL ENCODE
String auth_str = UrlEncoderUtils.encode(origin_auth_str);
StringBuilder strBuilder = new StringBuilder();
strBuilder.append("http://").append(bucketName).append("-").append(appid).append(".")
  .append("cn-north").append(".myqcloud.com")
  .append(UrlEncoderUtils.encodeEscapeDelimiter(key)).append("?sign=")
```

高级 API 文件上传(推荐)

高级 API 由类 TransferManger 通过封装上传以及下载接口，内部有一个线程池，接受用户的上传和下载请求，因此用户可选择异步的提交任务。用户上传或下载文件（Object）推荐使用高级 API 。

```
// 生成 TransferManager
TransferManager transferManager = new TransferManager(cosClient);
// .....(提交上传下载请求, 如下文所述)
```

```
// 关闭 TransferManger
transferManager.shutdownNow();
```

上传文件

上传接口根据用户文件的长度自动选择简单上传或分块上传的方式将文件上传到 Bucket 中，降低用户的使用门槛。并且使用分块上传时用户不用关心每一个步骤情况，十分便捷。

方法原型

```
// 上传文件
public Upload upload(final PutObjectRequest putObjectRequest)
  throws CosServiceException, CosClientException;
```

参数说明

参数名	参数类型	默认值	参数描述
putObjectRequest	PutObjectRequest	无	上传文件请求

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
key	String	无	构造函数或 set 方法	COS 的文件路径，即从 Bucket 开始
file	File	无	构造函数或 set 方法	本地文件
input	InputStream	无	构造函数或 set 方法	输入流
metadata	ObjectMetadata	无	构造函数或 set 方法	文件的元信息

成功返回值

返回值类型	返回值描述
Upload	用于查询上传结果，获取返回值。

示例

```
// 本地文件上传
```

```
Upload upload = transferManager.upload(putObjectRequest, localFile);
 // 等待传输结束(如果想同步的等待上传结束，则调用waitForCompletion)
UploadResult uploadResult = upload.waitForUploadResult();
```

下载文件

将指定 Bucket 中的文件（Object）下载到指定的本地文件中。

方法原型

```
// 下载文件
public Download download(final GetObjectRequest GetObjectRequest, final File file);
```

参数说明

参数名	参数类型	默认值	参数描述
getObjectRequest	GetObjectRequest	无	下载文件请求
file	File	无	下载目的地

Request 对象成员变量	类型	默认值	设置方法	描述
bucketName	String	无	构造函数或 set 方法	Bucket 名称
key	String	无	构造函数或 set 方法	COS 的文件路径, 即从 Bucket 开始
file	File	无	构造函数或 set 方法	本地文件
input	InputStream	无	构造函数或 set 方法	输入流
metadata	ObjectMetadata	无	构造函数或 set 方法	文件的元信息

成功返回值

返回值	返回值描述
Download	用于获取上传结果，获取返回值

示例

```
// 下载文件
```

```
Download download = transferManager.download(GetObjectRequest, localFile);
 // 等待传输结束(如果想同步的等待上传结束，则调用 waitForCompletion)
download.waitForCompletion();
```

Android SDK

开发准备

SDK 获取

对象存储服务的 XML Android SDK 的下载github地址：XML Android SDK.

更多示例可参考Demo：XML Android SDK Demo.

开发准备

1. SDK 支持 Android 2.2 及以上版本的手机系统；
2. 手机必须要有网络（GPRS、3G或 WIFI 网络等）；
3. 手机可以没有存储空间，但会使部分功能无法正常工作；
4. 从控制台获取 APPID、SecretID、SecretKey。

SDK 配置

配置工程导入下列 jar 包：

- cos-xml-android-sdk-1.0.jar
- qcloud-network-android-sdk-1.0.jar
- okhttp-3.8.1.jar
- okio-1.13.0.jar
- slf4j-android-1.6.1-RC1.jar
- xstream-1.4.7.jar
- fastjson-1.1.60.android.jar

SDK 需要网络访问相关的一些权限，需要在 AndroidManifest.xml 中增加如下权限声明：

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

快速入门

初始化 CosXmlService 和 CosXmlServiceConfig

```
String appid = "对象存储 的服务APPID";
String region = "存储桶 所在的地域";

String secretId = "云 API 密钥 secretId";
String secretKey ="云 API 密钥 secretKey";
long keyDuration = 600; //secretKey的有效时间,单位秒

//创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig(appid,region);

//创建获取签名类
CosXmlCredentialProvider cosXmlCredentialProvider = new CosXmlLocalCredentialProvider(secretId,
secretKey, keyDuration);

//创建 CosXmlService 对象，实现对象存储服务各项操作.
CosXmlService cosXmlService = new CosXmlService(context,cosXmlServiceConfig,
cosXmlCredentialProvider);
```

简单上传文件

```
String bucket = "存储桶名称";
String cosPath = "远端路径，即存储到cos上的绝对路径"; //格式如 cosPath = "/test.txt";
String srcPath = "本地文件的绝对路径"; // 如 srcPath =
Environment.getExternalStorageDirectory().getPath() + "/test.txt";
long signDuration = 600; //签名的有效期，单位为秒
```

```
PutObjectRequest putObjectRequest = new PutObjectRequest();
putObjectRequest.setBucket(bucket);
putObjectRequest.setCosPath(cosPath);
putObjectRequest.setSrcPath(srcPath);
putObjectRequest.setSign(signDuration,null,null);

//设置进度显示
putObjectRequest.setProgressListener(new QCloudProgressListener() {
  @Override
  public void onProgress(long progress, long max) {
  float result = (float) (progress * 100.0/max);
  Log.w("TEST","progress =" + (long)result + "%");
  }
});

//使用同步方法上传
try {
  PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);

  //上传失败， 返回httpCode 不在【200，300）之内；
  if(putObjectResult.getHttpCode() >= 300 || putObjectResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(putObjectResult.error.code)
  .append(putObjectResult.error.message)
  .append(putObjectResult.error.resource)
  .append(putObjectResult.error.requestId)
  .append(putObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //上传成功
  if(putObjectResult.getHttpCode() >= 200 && putObjectResult.getHttpCode() < 300){
  //上传成功后，则可以拼接访问此文件的地址，格式为：bucket-appid.region.myqcloud.com.cosPath;
```

```
  Log.w("TEST","accessUrl =" + putObjectResult.accessUrl);
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}

//使用异步回调上传：sdk为对象存储各项服务提供异步回调操作方法
/**

cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {
  Log.w("TEST","accessUrl =" + result.accessUrl);
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(putObjectResult.error.code)
  .append(putObjectResult.error.message)
  .append(putObjectResult.error.resource)
  .append(putObjectResult.error.requestId)
  .append(putObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

分片上传文件

```
//分片上传一般需要经历：初始化分片上传->分块上传->完成等3个阶段.

//第一步，初始化分片上传，获取 uploadId
InitMultipartUploadRequest initMultipartUploadRequest = new InitMultipartUploadRequest();
initMultipartUploadRequest.setBucket(bucket);
initMultipartUploadRequest.setCosPath(cosPath);
initMultipartUploadRequest.setSign(600,null,null);
try {
  InitMultipartUploadResult initMultipartUploadResult =
  cosXmlService.initMultipartUpload(initMultipartUploadRequest);

  //若初始化成功，则获取uploadId;
  String uploadId;
  if(initMultipartUploadResult.getHttpCode()< 200 || initMultipartUploadResult.getHttpCode() >=
300){
  Log.w("TEST","初始化失败");
  }else{
  Log.w("TEST","初始化成功");
  uploadId = initMultipartUploadResult.initMultipartUpload.uploadId;
  }
}catch (QCloudException e) {
  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}

//第二步，分片上传，需要参数 uploadId 和分片号 partNumber; # 此处只演示只有一个分片的文件例子 #.
int partNumber = 1;
UploadPartRequest uploadPartRequest = new UploadPartRequest();
uploadPartRequest.setBucket(bucket);
uploadPartRequest.setCosPath(cosPath);
uploadPartRequest.setUploadId(uploadId);
uploadPartRequest.setPartNumber(partNumber); //上传分片编码，从1开始； 此处演示上传第一个分片
```

```
uploadPartRequest.setSign(600,null,null);
try {
  uploadPartRequest.setProgressListener(new QCloudProgressListener() {
  @Override
  public void onProgress(long progress, long max) {
  float result = (float) (progress * 100.0/max);
  Log.w("TEST","progress =" + (long)result + "%");
  }
  });
  UploadPartResult uploadPartResult = cosXmlService.uploadPart(uploadPartRequest);
  if(putObjectResult.getHttpCode() >= 300 || putObjectResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(putObjectResult.error.code)
  .append(putObjectResult.error.message)
  .append(putObjectResult.error.resource)
  .append(putObjectResult.error.requestId)
  .append(putObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }else{
  //上传成功，继续上传剩下的分片
  Log.w("TEST","上传成功，使用相同的方法，继续上传剩下的分片");

  String eTag = uploadPartResult.getETag(); // 获取分片文件的 md5
  }
}catch (QCloudException e) {
  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}

//第三步，当确定所有分片全部上传完成之后，调用CompleteMultiUploadRequest完成分片上传结束.
//需要参数 uploadId， partNumber和对应每块分片文件的md5值
int partNumber = 1;
String eTag = "分片文件的MD5";
CompleteMultiUploadRequest completeMultiUploadRequest = new CompleteMultiUploadRequest();
```

```
completeMultiUploadRequest.setBucket(bucket);
completeMultiUploadRequest.setCosPath(cosPath);
completeMultiUploadRequest.setUploadId(uploadId);
completeMultiUploadRequest.setParNumberAndSha1(partNumber, eTag);
//此处只演示一个分片的例子
completeMultiUploadRequest.setSign(600,null,null);
try {
  CompleteMultiUploadResult completeMultiUploadResult =
  cosXmlService.completeMultiUpload(completeMultiUploadRequest);
  if(completeMultiUploadResult.getHttpCode() >= 300 || completeMultiUploadResult.getHttpCode()
< 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(putObjectResult.error.code)
  .append(putObjectResult.error.message)
  .append(putObjectResult.error.resource)
  .append(putObjectResult.error.requestId)
  .append(putObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }else{
  //上传成功

  Log.w("TEST","accessUrl =" + completeMultiUploadResult.accessUrl );
  }
catch (QCloudException e) {
  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}
```

初始化

进行操作之前需要实例化 CosXmlService 和 CosXmlServiceConfig.

实例化 CosXmlServiceConfig

调用

CosXmlServiceConfig(String appid, String region)

构造方法，实例化 CosXmlServiceConfig 对象.

参数说明

参数名称	类型	是否必填	参数描述
appid	String	是	对象存储 的服务APPID
region	String	是	存储桶 所在的地域

其它配置设置方法

方法	方法描述
setHttpProtocol(boolean)	true: https请求； false: http请求； 默认http请求
setConnectionTimeout(int)	连接超时设置
setSocketTimeout(int)	读写超时设置
setMaxRetryCount(int)	失败请求重试次数

示例

```
String appid = "对象存储 的服务APPID";
String region = "存储桶 所在的地域"; //所属地域：在创建好存储桶后，可通过对象存储控制台查看
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig(appid,region);
```

实例化 CosXmlService

调用

CosXmlService(Context context, CosXmlServiceConfig serviceConfig, CosXmlCredentialProvider cloudCredentialProvider)

构造方法，实例化 CosXmlService 对象.

参数说明

参数名称	类型	是否必填	参数描述
context	Context	是	上下文
serviceConfig	CosXmlServiceConfig	是	SDK 的配置设置类
cloudCredentialProvider	CosXmlCredentialProvider	是	服务请求的签名获取类

示例

```
String appid = "对象存储 的服务APPID";
String region = "存储桶 所在的地域";

//创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig(appid,region);

/**
*
* 创建 CosXmlCredentialProvider 签名获取类对象，用于使用对象存储服务时计算签名.
* 参考SDK提供签名格式，可实现自己的签名方法.
* 此处使用SDK提供的默认签名计算方法.
*
*/
String secretId = "云 API 密钥 secretId";
String secretKey ="云 API 密钥 secretKey";
long keyDuration = 600; //secretKey的有效时间,单位秒
CosXmlCredentialProvider cosXmlCredentialProvider = new CosXmlLocalCredentialProvider(secretId,
secretKey, keyDuration);

//创建 CosXmlService 对象，实现对象存储服务各项操作.
Context context = getApplicationContext()； //应用的上下文
CosXmlService cosXmlService = new CosXmlService(context,cosXmlServiceConfig,
cosXmlCredentialProvider);
```

生成签名

签名具体的生成和使用请参照签名流程.

SDK 中已提供了签名获取类，用户只需要继承 CosXmlCredentialProvider 类，并重写

signaturePair()

方法.

示例

```
public class CosXmlLocalCredentialProvider extends CosXmlCredentialProvider{
  private String secretKey;
  private long duration;

  public CosXmlLocalCredentialProvider(String secretId, String secretKey, long keyDuration) {
  super(secretId);
  this.secretKey = secretKey;
  this.duration = keyDuration;
  }

  @Override
  public CosXmlSignaturePair signaturePair() throws QCloudException {
  long current = System.currentTimeMillis() / 1000;
  long expired = current + duration;
  String keyTime = current+";"+expired;
  return new CosXmlSignaturePair(secretKeyToSignKey(secretKey, keyTime), keyTime);
  }

  private String secretKeyToSignKey(String secretKey, String keyTime) {
  String signKey = null;
  try {
  if (secretKey == null) {
```

```
throw new IllegalArgumentException("secretKey is null");
}
if (keyTime == null) {
throw new IllegalArgumentException("qKeyTime is null");
}
} catch (IllegalArgumentException e) {
e.printStackTrace();
}
try {
byte[] byteKey = secretKey.getBytes("utf-8");
SecretKey hmacKey = new SecretKeySpec(byteKey, "HmacSHA1");
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(hmacKey);
signKey = StringUtils.toHexString(mac.doFinal(keyTime.getBytes("utf-8")));
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
} catch (InvalidKeyException e) {
e.printStackTrace();
}
return signKey;
}
}
```

简单上传文件

调用此接口可以将本地的文件上传至指定 Bucket 中.具体步骤如下：

1. 调用

   PutObjectRequest()

构造方法，实例化 PutObjectRequest 对象.

2. 调用 CosXmlService 的 getService 方法，传入 PutObjectRequest，返回 PutObjectResult 对象.（或者 调用 putObjectAsync 方法，传入 PutObjectRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
srcPath	String	是	本地文件的绝对路径
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
qCloudProgressListener	QCloudProgressListener	否	上传进度回调
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 PutObjectResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
accessUrl	String	请求成功时，返回访问文件的地址

示例

```
PutObjectRequest putObjectRequest = new PutObjectRequest();
putObjectRequest.setBucket(bucket);
putObjectRequest.setCosPath(cosPath);
putObjectRequest.setSrcPath(srcPath);
putObjectRequest.setSign(signDuration,null,null);
putObjectRequest.setProgressListener(new QCloudProgressListener() {
  @Override
```

```
  public void onProgress(long progress, long max) {
  float result = (float) (progress * 100.0/max);
  Log.w("TEST","progress =" + (long)result + "%");
  }
});

//使用同步方法上传
try {
  PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);

  //上传失败， 返回httpCode 不在【200，300）之内；
  if(putObjectResult.getHttpCode() >= 300 || putObjectResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(putObjectResult.error.code)
  .append(putObjectResult.error.message)
  .append(putObjectResult.error.resource)
  .append(putObjectResult.error.requestId)
  .append(putObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //上传成功
  if(putObjectResult.getHttpCode() >= 200 && putObjectResult.getHttpCode() < 300){
  //上传成功后，则可以拼接访问此文件的地址，格式为：bucket-appid.region.myqcloud.com.cosPath;

  Log.w("TEST","accessUrl =" + putObjectResult.accessUrl);
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}
```

```
//**使用异步回调上传**
/**

cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {
  Log.w("TEST","accessUrl =" + result.accessUrl);
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

分片上传

初始化分片

调用此接口实现初始化分片上传，成功执行此请求以后会返回 UploadId 用于后续的 Upload Part 请求.具体步骤如下：

1. 调用

InitMultipartUploadRequest()

构造方法，实例化 InitMultipartUploadRequest 对象.

2. 调用 CosXmlService 的 initMultipartUpload 方法，传入 InitMultipartUploadRequest，返回 InitMultipartUploadResult 对象.
（或者 调用 initMultipartUploadAsync 方法，传入 InitMultipartUploadRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 InitMultipartUploadResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
initMultipartUpload	InitMultipartUpload	请求成功的返回结果

示例

```
InitMultipartUploadRequest initMultipartUploadRequest = new InitMultipartUploadRequest();
initMultipartUploadRequest.setBucket(bucket);
initMultipartUploadRequest.setCosPath(cosPath);
initMultipartUploadRequest.setSign(signDuration,null,null);

String uploadId = null;
```

```
//使用同步方法请求
try {
 InitMultipartUploadResult initMultipartUploadResult =
cosXmlService.initMultipartUpload(initMultipartUploadRequest);

 //请求失败，返回httpCode 不在【200，300）之内；
 if(initMultipartUploadResult.getHttpCode() >= 300 || initMultipartUploadResult.getHttpCode() <
200){
 StringBuilder stringBuilder = new StringBuilder("Error\n");
 stringBuilder.append(initMultipartUploadResult.error.code)
 .append(initMultipartUploadResult.error.message)
 .append(initMultipartUploadResult.error.resource)
 .append(initMultipartUploadResult.error.requestId)
 .append(initMultipartUploadResult.error.traceId);
 Log.w("TEST",stringBuilder.toString());
 }

 //请求成功
 if(initMultipartUploadResult.getHttpCode() >= 200 && initMultipartUploadResult.getHttpCode() <
300){

 uploadId =initMultipartUploadResult.initMultipartUpload.uploadId;
 Log.w("TEST","请求成功");
 }


 } catch (QCloudException e) {

 //抛出异常
 Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}
```

```
//**使用异步回调请求**
/**

cosXmlService.initMultipartUploadAsync(initMultipartUploadRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  uploadId =initMultipartUploadResult.initMultipartUpload.uploadId;
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

上传分片

调用此接口实现分块上传，支持的块的数量为1到10000，块的大小为1 MB 到5 GB.具体步骤如下：

1. 调用

   UploadPartRequest()

   构造方法，实例化 UploadPartRequest 对象.

2. 调用 CosXmlService 的 uploadPart 方法，传入 UploadPartRequest，返回 UploadPartResult 对象.
   （或者 调用 uploadPartAsync 方法，传入 UploadPartRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
uploadId	String	是	初始化分片上传，返回的uploadId
partNumber	int	是	分片块的编号，从 1 开始起
srcPath	String	是	本地文件的绝对路径
fileOffset	long	是	该分片在文件的中起始位置
contentLength	long	否	该分片的内容大小
signDuration	long	否	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
qCloudProgressListener	QCloudProgressListener	否	上传进度回调
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 UploadPartResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
eTag	String	请求成功,返回分片文件的MD5值，用于最后完成分片

示例

```
UploadPartRequest uploadPartRequest = new UploadPartRequest();
uploadPartRequest.setBucket(bucket);
uploadPartRequest.setCosPath(cosPath);
uploadPartRequest.setUploadId(uploadId);
uploadPartRequest.setPartNumber(partNumber); //此次上传分片的编号，从1开始
uploadPartRequest.setSrcPath(srcPath, fileOffset, contentLength); //fileOffset
该分片的起始位置，contentLength该分片的大小
uploadPartRequest.setSign(signDuration,null,null);
uploadPartRequest.setProgressListener(new QCloudProgressListener() {
  @Override
  public void onProgress(long progress, long max) {
  float result = (float) (progress * 100.0/max);
  Log.w("TEST","progress =" + (long)result + "%");
  }
});

String eTag = null;

//使用同步方法上传
try {
  UploadPartResult uploadPartResult = cosXmlService.uploadPart(uploadPartRequest);

  //上传失败， 返回httpCode 不在【200，300）之内；
  if(uploadPartResult.getHttpCode() >= 300 || uploadPartResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(uploadPartResult.error.code)
  .append(uploadPartResult.error.message)
  .append(uploadPartResult.error.resource)
  .append(uploadPartResult.error.requestId)
  .append(uploadPartResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //上传成功
```

```
  if(uploadPartResult.getHttpCode() >= 200 && uploadPartResult.getHttpCode() < 300){

  eTag =uploadPartResult.getETag();
  Log.w("TEST","请求成功");
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**

cosXmlService.uploadPartAsync(uploadPartRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  eTag =result.getETag();
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
```

});

*/

完成整个分片上传

当上传完所有分块以后，必须调用此接口用来实现完成整个分块上传.具体步骤如下：

1. 调用

   CompleteMultiUploadRequest()

   构造方法，实例化 CompleteMultiUploadRequest 对象.
2. 调用 CosXmlService 的 completeMultiUpload 方法，传入 CompleteMultiUploadRequest，返回 CompleteMultiUploadResult 对象.
   （或者 调用 completeMultiUploadAsync 方法，传入 CompleteMultiUploadRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
uploadId	String	是	初始化分片上传，返回的uploadId
partNumberAndETag	Map	是	分片编号 和对应的分片MD5值
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 CompleteMultiUploadResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
completeMultipartUpload	CompleteMultipartResult	请求成功的返回结果
accessUrl	String	请求成功时，返回访问文件的地址

示例

```
CompleteMultiUploadRequest completeMultiUploadRequest = new CompleteMultiUploadRequest();
completeMultiUploadRequest.setBucket(bucket);
completeMultiUploadRequest.setCosPath(cosPath);
completeMultiUploadRequest.setUploadId(uploadId);
completeMultiUploadRequest.setPartNumberAndETag(partNumberAndETag);
completeMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
  CompleteMultiUploadResult completeMultiUploadResult =
cosXmlService.completeMultiUpload(completeMultiUploadRequest);

  //请求失败， 返回httpCode 不在【200，300）之内；
  if(completeMultiUploadRequest.getHttpCode() >= 300 ||
completeMultiUploadRequest.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(completeMultiUploadRequest.error.code)
  .append(completeMultiUploadRequest.error.message)
  .append(completeMultiUploadRequest.error.resource)
  .append(completeMultiUploadRequest.error.requestId)
  .append(completeMultiUploadRequest.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
```

```
  //请求成功
  if(completeMultiUploadRequest.getHttpCode() >= 200 &&
completeMultiUploadRequest.getHttpCode() < 300){

  String accessUrl =completeMultiUploadResult.accessUrl;
  Log.w("TEST","请求成功 " + completeMultiUploadResult.completeMultipartUpload.toString());
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**

cosXmlService.completeMultiUploadAsync(completeMultiUploadRequest, new
CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {
  String accessUrl =completeMultiUploadResult.accessUrl;
  Log.w("TEST","请求成功 " + completeMultiUploadResult.completeMultipartUpload.toString());
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
```

```
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

列举已上传的分片

调用此接口用来查询特定分块上传中的已上传的块，即罗列出指定 UploadId 所属的所有已上传成功的分块.

1. 调用

   ListPartsRequest()

   构造方法，实例化 ListPartsRequest 对象.
2. 调用 CosXmlService 的 listParts 方法，传入 ListPartsRequest，返回 ListPartsResult 对象.
   （或者 调用 listPartsAsync 方法，传入 ListPartsRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
uploadId	String	是	初始化分片上传，返回的uploadId
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 ListPartsResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
listParts	ListParts	请求成功返回的结果

示例

```
ListPartsRequest listPartsRequest = new ListPartsRequest();
listPartsRequest.setBucket(bucket);
listPartsRequest.setCosPath(cosPath);
listPartsRequest.setUploadId(uploadId);
listPartsRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
  ListPartsResult listPartsResult = cosXmlService.listParts(listPartsRequest);

  //请求失败， 返回httpCode 不在【200，300）之内；
  if(listPartsResult.getHttpCode() >= 300 || listPartsResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(listPartsResult.error.code)
  .append(listPartsResult.error.message)
  .append(listPartsResult.error.resource)
  .append(listPartsResult.error.requestId)
  .append(listPartsResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //请求成功
  if(listPartsResult.getHttpCode() >= 200 && listPartsResult.getHttpCode() < 300){

  Log.w("TEST","请求成功 " + listPartsResult.listParts.toString());
```

```
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**

cosXmlService.listPartsAsync(listPartsRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  Log.w("TEST","请求成功 " + listPartsResult.listParts.toString());
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

舍弃并删除已上传的分片

调用此接口用来用来实现舍弃一个分块上传并删除已上传的块.

1. 调用

   AbortMultiUploadRequest()

   构造方法，实例化 AbortMultiUploadRequest 对象.
2. 调用 CosXmlService 的 abortMultiUpload 方法，传入 AbortMultiUploadRequest，返回 AbortMultiUploadResult 对象.
   （或者 调用 abortMultiUploadAsync 方法，传入 AbortMultiUploadRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
uploadId	String	是	初始化分片上传，返回的uploadId
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 AbortMultiUploadResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
httpCode	int	请求成功返回的结果

示例

```
AbortMultiUploadRequest abortMultiUploadRequest = new AbortMultiUploadRequest();
abortMultiUploadRequest.setBucket(bucket);
abortMultiUploadRequest.setCosPath(cosPath);
abortMultiUploadRequest.setUploadId(uploadId);
abortMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
  AbortMultiUploadResult abortMultiUploadResult =
cosXmlService.abortMultiUpload(abortMultiUploadRequest);

  //请求失败， 返回httpCode 不在【200，300）之内；
  if(abortMultiUploadResult.getHttpCode() >= 300 || abortMultiUploadResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(abortMultiUploadResult.error.code)
  .append(abortMultiUploadResult.error.message)
  .append(abortMultiUploadResult.error.resource)
  .append(abortMultiUploadResult.error.requestId)
  .append(abortMultiUploadResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //请求成功
  if(abortMultiUploadResult.getHttpCode() >= 200 && abortMultiUploadResult.getHttpCode() <
300){

  Log.w("TEST","请求成功 ");
  }


  } catch (QCloudException e) {
```

```
  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}



//**使用异步回调请求**
/**

cosXmlService.abortMultiUploadAsync(abortMultiUploadRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  Log.w("TEST","请求成功 ");
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

删除文件

删除单个文件

调用此接口可以在指定的 Bucket 中将一个文件删除.具体步骤如下：

1. 调用

   DeleteObjectRequest()

   构造方法，实例化 DeleteObjectRequest 对象.
2. 调用 CosXmlService 的 completeMultiUpload 方法，传入 DeleteObjectRequest，返回 DeleteObjectResult 对象.
   （或者 调用 deleteObjectAsync 方法，传入 DeleteObjectRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 DeleteObjectResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
httpCode	int	请求成功返回的结果

示例

```
DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest();
deleteObjectRequest.setBucket(bucket);
```

```
deleteObjectRequest.setCosPath(cosPath);
completeMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法删除
try {
  DeleteObjectResult deleteObjectResult = cosXmlService.deleteObject(deleteObjectRequest);

  //删除失败， 返回httpCode 不在【200，300）之内；
  if(deleteObjectResult.getHttpCode() >= 300 || deleteObjectResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(deleteObjectResult.error.code)
  .append(deleteObjectResult.error.message)
  .append(deleteObjectResult.error.resource)
  .append(deleteObjectResult.error.requestId)
  .append(deleteObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //删除成功
  if(deleteObjectResult.getHttpCode() >= 200 && deleteObjectResult.getHttpCode() < 300){

  Log.w("TEST","请求成功 ");
  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**
```

```
cosXmlService.deleteObjectAsync(deleteObjectRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  Log.w("TEST","请求成功 ");
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```

删除多个文件

调用此接口可以在指定 Bucket 中批量删除文件，单次请求最大支持批量删除 1000 个 文件.具体步骤如下：

1. 调用

   DeleteMultiObjectRequest()

   构造方法，实例化 DeleteMultiObjectRequest 对象.
2. 调用 CosXmlService 的 deleteMultiObject 方法，传入 DeleteMultiObjectRequest，返回 DeleteMultiObjectResult 对象.

（或者 调用 deleteMultiObjectAsync 方法，传入 DeleteMultiObjectRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
quiet	boolean	是	true: 只返回删除报错的文件信息; false: 返回每个文件的删除结果
objectList	List	是	需要删除的文件路径列表
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 DeleteMultiObjectResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
httpCode	int	请求成功返回的结果

示例

```
DeleteMultiObjectRequest deleteMultiObjectRequest = new DeleteMultiObjectRequest();
deleteMultiObjectRequest.setBucket(bucket);
deleteMultiObjectRequest.setQuiet(quiet);
deleteMultiObjectRequest.setObjectList(objectList); // 每个文件均是绝对路径，如 /2/test.txt;
completeMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法删除
try {
  DeleteMultiObjectResult deleteMultiObjectResult
```

```
=cosXmlService.deleteMultiObject(DeleteMultiObjectRequest);

 //删除失败， 返回httpCode 不在【200，300）之内；
 if(deleteMultiObjectResult.getHttpCode() >= 300 || deleteMultiObjectResult.getHttpCode() < 200){
 StringBuilder stringBuilder = new StringBuilder("Error\n");
 stringBuilder.append(deleteMultiObjectResult.error.code)
 .append(deleteMultiObjectResult.error.message)
 .append(deleteMultiObjectResult.error.resource)
 .append(deleteMultiObjectResult.error.requestId)
 .append(deleteMultiObjectResult.error.traceId);
 Log.w("TEST",stringBuilder.toString());
 }

 //删除成功
 if(deleteMultiObjectResult.getHttpCode() >= 200 && deleteMultiObjectResult.getHttpCode() <
300){

 Log.w("TEST","删除成功： " + deleteMultiObjectResult.deleteResult.toString());

 }


 } catch (QCloudException e) {

 //抛出异常
 Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**

cosXmlService.deleteMultiObjectAsync(DeleteMultiObjectResult, new CosXmlResultListener() {
 @Override
```

```
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","删除成功： " + deleteMultiObjectResult.deleteResult.toString());
}

@Override
public void onFail(CosXmlRequest request, CosXmlResult result) {
StringBuilder stringBuilder = new StringBuilder("Error\n");
stringBuilder.append(result.error.code)
.append(result.error.message)
.append(result.error.resource)
.append(result.error.requestId)
.append(result.error.traceId);
Log.w("TEST",stringBuilder.toString());
}
});

*/
```

下载文件

调用此接口将指定Bucket 中的一个文件下载至本地.具体步骤如下：

1. 调用

   GetObjectRequest(String)

   构造方法，实例化 GetObjectRequest 对象.
2. 调用 CosXmlService 的 getObject 方法，传入 GetObjectRequest，返回 GetObjectResult 对象.（或者 调用 getObjectAsync 方法，传入 GetObjectRequest 和 CosXmlResultListener 进行异步回调操作）.

参数说明

参数名称	类型	是否必填	参数描述
bucket	String	是	存储桶名称
cosPath	String	是	远端路径，即存储到cos上的绝对路径
savaPath	String	否	文件下载到本地文件夹的绝对路径
start	long	否	请求文件的开始位置
end	long	否	请求文件的结束位置
signDuration	long	是	签名的有效期，单位为秒
checkHeaderListForSign	Set	否	签名中需要验证的请求头
checkParameterListForSing	Set	否	签名中需要验证的请求参数
qCloudProgressListener	QCloudProgressListener	否	下载进度回调
cosXmlResultListener	CosXmlResultListener	否	上传结果回调

返回结果说明

通过 GetObjectResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
error	COSXMLError	请求失败的返回结果
httpCode	int	请求成功返回的结果

示例

```
GetObjectRequest getObjectRequest = GetObjectRequest(**savePath**);
getObjectRequest.setBucket(bucket);
getObjectRequest.setCosPath(cosPath);
getObjectRequest.setSign(signDuration,null,null);
getObjectRequest.setProgressListener(new QCloudProgressListener() {
  @Override
  public void onProgress(long progress, long max) {
  float result = (float) (progress * 100.0/max);
  Log.w("TEST","progress =" + (long)result + "%");
  }
});
```

```
//使用同步方法下载
try {
  GetObjectResult getObjectResult =cosXmlService.getObject(getObjectRequest);

  //下载失败， 返回httpCode 不在【200，300）之内；
  if(getObjectResult.getHttpCode() >= 300 || getObjectResult.getHttpCode() < 200){
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(getObjectResult.error.code)
  .append(getObjectResult.error.message)
  .append(getObjectResult.error.resource)
  .append(getObjectResult.error.requestId)
  .append(getObjectResult.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }

  //下载成功
  if(getObjectResult.getHttpCode() >= 200 && getObjectResult.getHttpCode() < 300){

  Log.w("TEST","下载成功： " + getObjectResult.xCOSStorageClass);

  }


  } catch (QCloudException e) {

  //抛出异常
  Log.w("TEST","exception =" + e.getExceptionType() + "; " + e.getDetailMessage());
}


//**使用异步回调请求**
/**
```

```
cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
  @Override
  public void onSuccess(CosXmlRequest request, CosXmlResult result) {

  Log.w("TEST","下载成功： " + getObjectResult.xCOSStorageClass);
  }

  @Override
  public void onFail(CosXmlRequest request, CosXmlResult result) {
  StringBuilder stringBuilder = new StringBuilder("Error\n");
  stringBuilder.append(result.error.code)
  .append(result.error.message)
  .append(result.error.resource)
  .append(result.error.requestId)
  .append(result.error.traceId);
  Log.w("TEST",stringBuilder.toString());
  }
});

*/
```