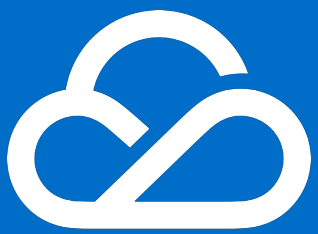
腾讯云Video on Demand

客户端视频上传(UGC)

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

UGC视频上传综述

功能介绍

随着终端用户个性化的需求愈加丰富，简单的文字交互和图片上传已经不能满足展示与分享信息的诉求，UGC(User Generated Content)也就应运而生。腾讯视频云点播业务的UGC功能，即客户端上传视频功能，支持终端用户将一段短小的视频快速地上传到云端。另外，值得一提的是，依托于腾讯云的支持，后续也能做到播放流畅。

目前腾讯云点播提供给如下平台的UGC上传SDK：

1. iOS UGC SDK；
2. Android UGC SDK；
3. Web UGC SDK。

如果您有其他平台的UGC上传需求，可以基于点播UGC视频上传接口进行开发，包括如下三个接口：

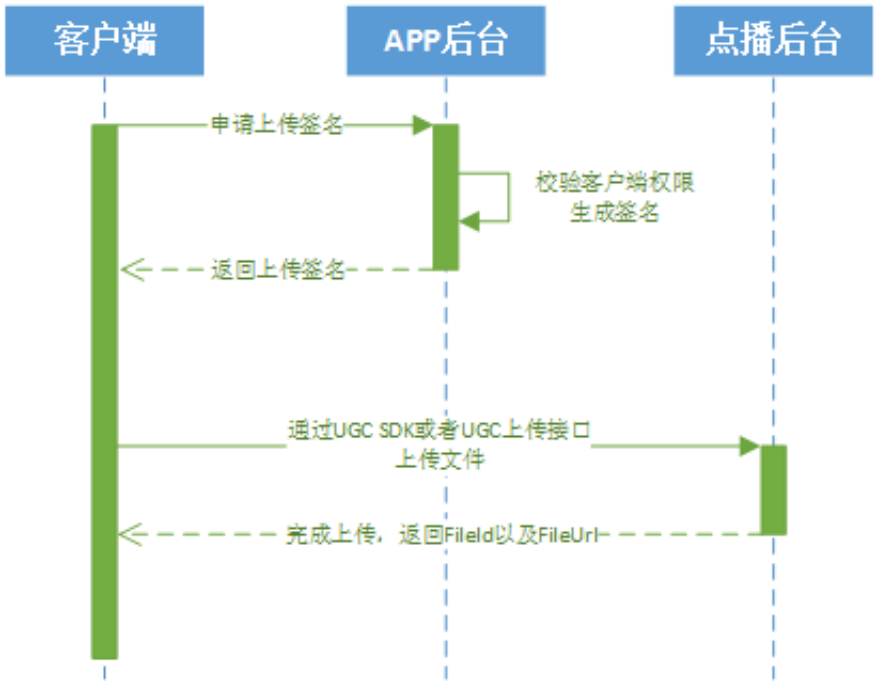
1. 初始化上传(UGC)；
2. 分片上传(UGC)；
3. 结束上传(UGC)。

业务流程

UGC视频上传的整体流程分为如下两步：

1. 客户端向服务端申请上传签名；
2. 调用腾讯云点播的UGC SDK（或者上传接口），进行视频上传。

如下图所示：



注意：

1. APP千万不要
   把自己的SecretID和SecretKey暴露给客户端，这两个关键信息泄露将导致很严重的安全隐患；
2. 为了确保APP在点播的存储空间不被恶意使用，点播要求UGC上传必须携带APP后台派发的上传签名；
3. 视频上传之后是否进一步进行处理（例如转码、鉴黄），是由APP服务端通过签名进行控制的。

UGC视频上传签名生成

点播UGC视频上传签名是一段经过Base64编码的二进制串，其中包含的主要信息如下：

1. 上传参数信息，包括：
   1. APP的SecretID；
   2. 视频的基本信息，例如视频名称、标签；
   3. 视频上传到点播之后的处理方式，例如是否进行转码、是否进行鉴黄等；
2. 用SecretKey生成的HMAC-SHA1签名，点播后台据此来校验UGC上传签名的合法性。

上传参数信息

字段	类型	必填	说明
s	String	是	云api管理页中的Secret ID

字段	类型	必填	说明
f	String	是	视频文件本地名称，长度在40个字节以内，不得包含 / : * ? " < > 等字符
t	Integer	是	当前时间戳，是一个符合 Unix Epoch 时间戳规范的数值，单位为秒
e	Integer	是	签名失效时刻，是一个符合 Unix Epoch时间戳规范的数值，单位为秒。e 的计算方式为 e = t + 签名有效时长。签名有效时长最大取值为7776000（90天）
r	Integer	是	随机串，无符号10进制整数，用户需自行生成，最长10位
fs	String	是	文件的SHA-1签名，由客户端计算并提交到APP后台
ft	String	是	文件类型，例如mp4,flv,avi等，注意不需要"."
uid	String	是	用户在APP内的唯一标识，建议取md5计算结果，例如对qq，号码12345就是uid=md5(12345)
tc	Integer	否	是否转码，需要转码时，tc=1
ss	Integer	否	转码时是否截图，需要截图时，ss=1。本参数仅tc=1时有效
wm	Integer	否	转码时是否加水印，需要转码加水印时，wm=1。本参数仅tc=1时有效
cid	Integer	否	文件分类id，需要指定分类

字段	类型	必填	说明
			id时填入
tag.n	String	否	文件标签，可指定最多10个标签，使用方式：例如三个标签就是tag.1=a&tag.2=b&tag.3=c

UGC视频上传签名生成

计算UGC视频上传签名，主要分为如下三个步骤：

1. 获取签名计算所需信息；
2. 拼接明文字符串；
3. 依照文字符串构造签名。

云点播提供了多种不同语言的签名上传示例代码，参见UGC上传签名生成示例代码。

获取签名计算所需信息

在生成签名所需信息中，Secret ID和Secret Key需要在腾讯云管理中心控制台页面的【云产品】---【监控与管理】下的【云 API 密钥】页面查看。

其他信息，除了文件的SHA信息必须依赖客户端提交，其他参数都可以由APP后台控制。

拼接签名明文字符串

拼接签名明文字符串，形式为HTTP QueryString。格式如下：

s=[SecretID]&f=[FileName]&fs=[FileSha]&t=[currentTime]&e=[expiredTime]&r=[rand]&uid=[uid]

我们设拼接好的明文字符串为Original。

注意：

1. 与服务端API的签名生成方式不同，这里不需要对所有参数进行排序；
2. 建议APP使用编程语言已有的QueryString相关类库来生成签名，而不是手工拼装字符串。

将明文字符串转化为签名

拼接好签名的明文字符串Original
后，用已经获取的SecretKey对明文串进行HMAC-SHA1加密，得到SignTmp：

SignTmp = HMAC-SHA1(SecretKey, Original)

将密文串SignTmp放在明文串Origin前面，拼接后进行Base64编码，得到最终的签名Sign：

Sign = Base64(append(SignTmp, Original))

UGC上传签名生成示例代码

- PHP示例
- Node.js示例

UGC视频上传SDK

iOS UGC SDK

SDK集成

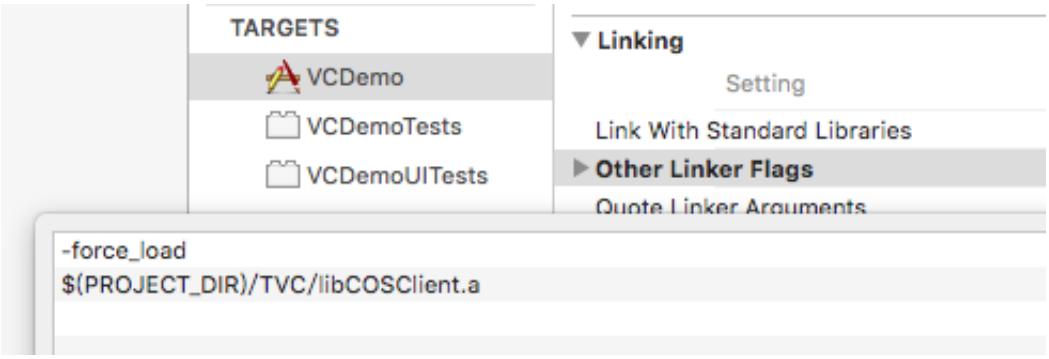
下载SDK

下载TVCClientSDK.framework。

引入依赖包

配置项目

Build Settings -> Other Linker Flags



注意事项：

- libCOSClient.a有模拟器和真机版本，请在不同的开发环境使用对应的版本
- 务必强制加载libCOSClient.a静态库，否则会导致crash

本地视频上传

Step1：生成上传对象

```
TVCConfig *config = [[TVCConfig alloc] init];
config.signature = signature;
config.secretId = secretId;
config.forceHttps = NO;
self.client = [[TVCClient alloc] initWithConfig:config];
```

Step2：生成上传参数

```
TVCUploadParam *param = [[TVCUploadParam alloc] init];
param.videoPath = videoPath;
param.coverPath = coverPath;
```

Step3：视频上传

```
[ws.client uploadVideo:param result:^(TVCUploadResponse *resp) {
  NSLog(@"result : %d-%@-%@-%@-%@",
  resp.retCode,
  resp.descMsg,
  resp.videoId,resp.videoURL,
  resp.coverURL);
} progress:^(NSInteger bytesUpload, NSInteger bytesTotal) {
  NSLog(@"progress : %ld-%ld",
  (long)bytesUpload,
  (long)bytesTotal);
}];
```

注意事项：

- TVCConfig参数字段不能为空；
- TVCUploadParam参数字段video路径不能为空，cover字段为空表示不上传封面预览图；
- 苹果规定2017年后所有上架App强制使用ATS，目前Http或https服务均可配置使用，苹果强制使用ATS后可在TVCConfig中配置使用Https协议。

Android UGC SDK

SDK集成

点击下载Android UGC SDK。解压zip包，配置工程导入其中的jar包:

- tvcsdk.jar
- okio-1.6.0.jar
- okhttp-3.2.0.jar
- cos-sdk-android-1.4.2.jar

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

```
<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"/>
```

本地视频上传

Step 1：创建上传对象

```
TVCClient client = new TVCClient(getApplicationContext(), SecretId, Signature);
```

参数名称	参数类型	参数说明

context	Context	上下文
scretId	String	密钥
signature	String	从服务端获取的上传签名

Step 2：创建上传配置

```
TVCUploadInfo info = new TVCUploadInfo(fileType, videoPath, coverType, coverPath);
```

参数名称	参数类型	参数说明
fileType	String	视频文件类型，支持mp4, flv
filePath	String	视频文件路径
coverType	String	封面图片类型，必须为jpg，如果不上传则填空字符串
coverPath	String	封面图片路径，如果不上传则填空字符串

Step3：视频上传

```
client.uploadVideo(info, new TVCUploadListener() {
  @Override
  public void onSucess(String fileId, String playUrl, String coverUrl) {
  Log.v(TAG, "uploadVideo->fileId:"+fileId);
  Log.v(TAG, "uploadVideo->playUrl:"+playUrl);
  Log.v(TAG, "uploadVideo->coverUrl:"+coverUrl);
  }

  @Override
  public void onFailed(int errCode, String errMsg) {
  Toast.makeText(MainActivity.this, "err " + errCode + ""
  + errMsg, Toast.LENGTH_SHORT).show();
  }

  @Override
  public void onProgress(long currentSize, long totalSize) {
```

```
double percent = (double) currentSize / (double) totalSize;
NumberFormat nt = NumberFormat.getPercentInstance();
nt.setMinimumFractionDigits(2);
Log.i("onProgess", "onProgress: " + nt.format(percent));
}
});
```

Web UGC SDK

SDK集成

1. 页面中引入js脚本
   https://qzonestyle.gtimg.cn/open/qcloud/js/vod/sdk/uploaderh5V3.js

2. 服务器端部署 calculator_worker_sha1.js, 该文件用于计算待上传文件的sha1，一般情况把直接该js文件下载到上传服务同域名根目录下即可。之所以有同域的限制，是因为SDK使用了html5 worker异步计算SHA1，优化上传体验，而同域是html5 worker本身的限制。

本地视频上传

上传组件初始化

```
var ErrorCode = qcVideo.get('ErrorCode');
ErrorCode.UN_SUPPORT_BROWSE !== qcVideo.uploader.initUGC(
 //1: 上传基础条件
 {
 upBtnId: upBtnId, //上传按钮ID（任意页面元素ID）

 /*
 @desc 从服务端获取签名的函数。该函数包含两个参数：
 argObj: 待上传文件的信息，关键信息包括：
 f: 视频文件名(可从getSignature的argObj中获取)，
 ft: 视频文件的类型(可从getSignature的argObj中获取)，
 fs: 视频文件的sha1值(必须从getSignature的argObj中获取)
 callback：客户端从自己的服务端得到签名之后，调用该函数将签名传递给SDK
 */
 getSignature: function(argObj, callback){
 // 调用APP后台服务器，返回签名
 var sigUrl = 'http://yourdomain?'
 + 'f=' + encodeURIComponent(argObj.f)
```

```
  + '&ft=' + encodeURIComponent(argObj.ft)
  + '&fs=' + encodeURIComponent(argObj.fs);

  $.get(sigUrl).done(function(ret) {
  callback(ret.signature);
  })
  }
  ,after_sha_start_upload: false
//上传分为两个阶段，sha计算和文件网络传输；这个选项设置是否在sha计算完成后，立即进行网络传输上传(默认非立即上传)
  ,sha1js_path: 'http://您的域名/您的设置目录/计算sha1脚本.js' //计算sha1的位置 ，默认为'http://你的域名/calculator_worker_sha1.js'
  }
  //2: 回调函数
  , {
  /**
  * 更新文件状态和进度
  * @param args { id: 文件ID, size: 文件大小, name: 文件名称, status: 状态, percent: 进度,speed: 速度, errorCode: 错误码 }
  */
  onFileUpdate: function (args) {
  console.log(args);
  },
  /**
  * 文件状态发生变化
  * @param info { done: 完成数量 , fail: 失败数量 , sha: 计算SHA或者等待计算SHA中的数量 , wait: 等待上传数量 , uploading: 上传中的数量 }
  */
  onFileStatus: function (info) {
  console.log('各状态总数' , info);
  },
  /**
  * 上传时错误文件过滤提示
  * @param args {code:{-1: 文件类型异常,-2: 文件名异常} , message: 错误原因 ， solution: 解决方法 }
```

```
*/
onFilterError: function (args) {
console.log('message:' + args.message + (args.solution ? (';solution==' + args.solution) : ''));
}
}
));
```

API

qcVideo.uploader.startUpload()

功能： 启动上传

参数： 无;

返回： 无;

qcVideo.uploader.stopUpload()

功能： 停止上传

参数： 无;

返回： 无;

qcVideo.uploader.reUpload()

功能： 恢复上传（错误文件重新上传）

参数： 无;

返回： 无;

qcVideo.uploader.deleteFile(fid)

功能： 删除本地上传任务

参数： fid 文件id;

返回： 无;

qcVideo.uploader.getOriginalFile(fid)

功能： 获取本地文件对象
参数： 无;
返回： FILE对象;

兼容性

该SDK当前仅支持HTML5上传，主流浏览器比如chrome, safari等经测试可正常使用，受限于各厂商对HTML5实现不一致，个别机型如无法正常使用，需要具体情况具体分析。

UGC视频上传接口

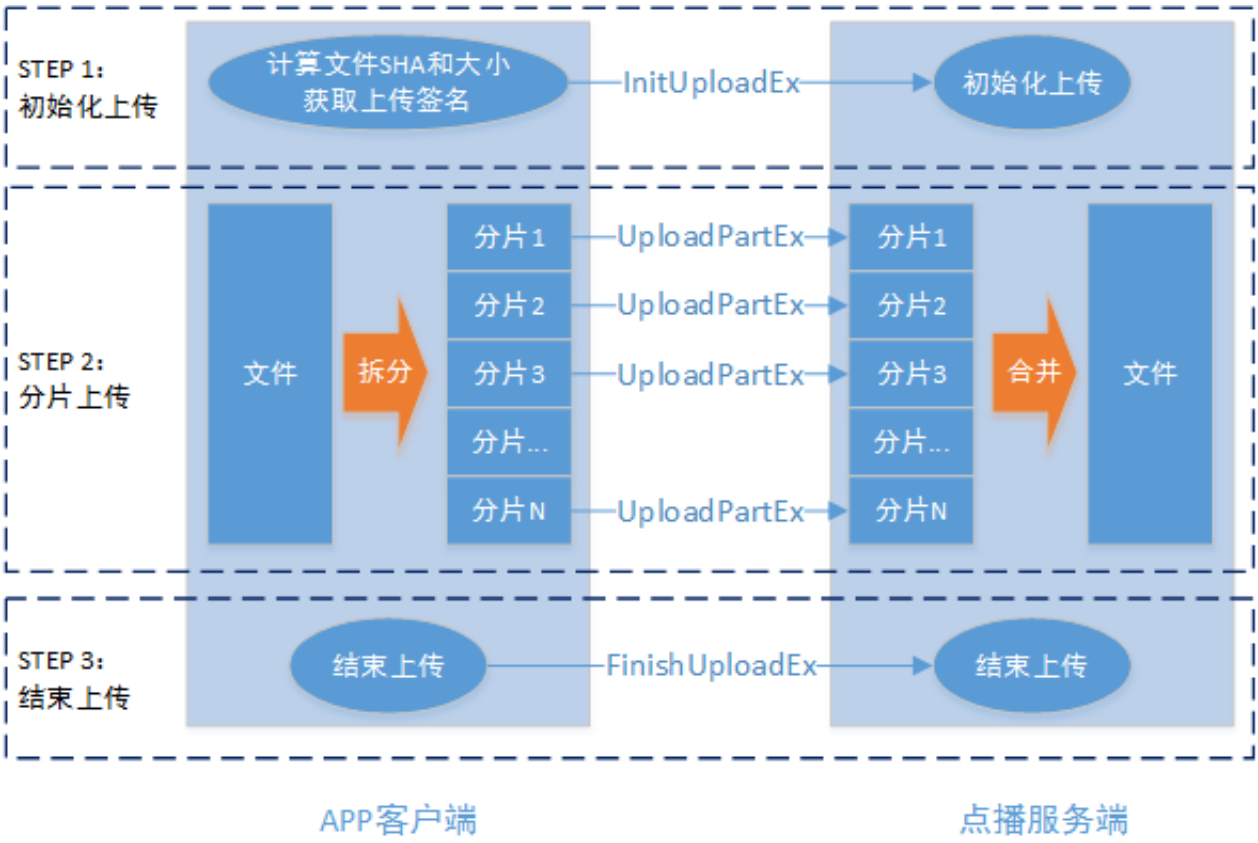
初始化上传(UGC)

接口名称

InitUploadEx

功能说明

1. 该接口适用于APP将自己客户端上的视频上传到点播服务端；
2. 该接口亦可用于APP将自己服务端视频上传到点播服务端的场景，但我们建议使用服务端上传SDK或者服务端上传接口（InitUpload/UploadPart/FinishUpload）来进行上传；
3. 由于视频文件通常较大，故而一般需要采用分片上传的方式；整个上传过程涉及初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)三步，具体流程参见下图；
4. 支持秒传、断点续传；
5. 接口本身逻辑较为复杂，点播封装了多种语言的UGC上传SDK来简化开发者的调用，详见UGC视频上传综述。



请求方式

请求域名

vod2.qcloud.com

注意：

- UGC视频上传、服务端视频上传的域名与其他服务端API不同，不是vod.api.qcloud.com；
- UGC视频上传的签名(signature)生成方式与服务端API不同，方法参见UGC视频上传签名生成；
- 单个视频文件上传所调用的所有接口，包括初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)，均使用相同的signature；
- 该接口仅支持GET方法，不支持POST方法。

参数说明

参数名称	必填	类型	说明
fileSha	是	String	视频文件的SHA值，计算方法见下文
fileSize	是	Integer	视频文件的总大小，单位字节Byte
dataSize	是	Integer	调用分片上传接口(UploadPartEx)时，每个分片的大小；可选值：524288(512KB)、1048576(1MB)
signature	是	String	UGC上传签名，参见UGC视频上传签名生成

注意：

- 文件名称、文件分类、是否转码等控制项，不是由客户端指定的，而是由服务端在signature中传递。

fileSha的计算方法：

1. 采用SHA-1散列算法计算整个视频文件的哈希值，得到160位(20字节)的二进制串；
2. 对得到的二进制串进行十六进制编码，编码算法必须使用小写字母，即a-f，而不是A-F。

请求示例

https://vod2.qcloud.com/v2/index.php?Action=InitUploadEx
&fileSha=b4a5c70c76e79e01ab3a5c306de3d9eedeadeca9
&fileSize=20350000
&dataSize=1048576
&signature=IEmbRAPy5IgIAFnt7XPAToaY3RRzPUFLSURVZ

接口应答

参数说明

参数名称	类型	说明
code	Integer	小于0：失败；0：初始化成功，可以进行分片上传；1：文件部分存在，可以进行断点续传；2：文件已经存在，无需再次上传(秒传)
message	String	返回信息说明
listParts	Array	当可以进行断点续传时(即code为1)，该数组列出了服务端目前已经存在的分片，这些分片无需再次进行上传
listParts.offset	Integer	已经存在的分片在文件中的相对偏移
listParts.dataSize	Integer	已经存在的分片的大小
listParts.dataMd5	String	已经存在的分片的md5
codeDesc	String	后台记录的错误信息
dataSize	Integer	code为1时返回，表示该文件可以进行断点续传，但之前的上传已指定了分片大小，本次上传过程中，后续的分片上传(UploadPart)的dataSize必须需以此值为准
fileId	String	code为2时返回，表示文件已经存在，该值表示视频文件的fileId
url	String	code为2时返回，表示文件已经存在，该值表示视频文件的url
canRetry	Integer	当code小于0时，说明上传过程遇到错误；此时如果该值为1表示该错误可以通过重试来解决，否则表示该错误无法通过重试解决，必须进行排查

错误码说明

错误码	含义说明
-10001	检查公共参数出错
-10002	检查签名出错
-10003	检查协议参数出错

错误码	含义说明
-10004	写缓存信息失败
-10005	获取缓存信息失败
-10006	包体非法

其他错误码可能为网络原因导致的偶发失败。所有小于0的错误码，当canRetry为1时，均可重试解决。

应答示例

正常上传：


```
{
  "code": 0,
  "message": "",
  "codeDesc": "",
  "canRetry": 0
}
```


断点续传：

如下应答包体的含义是：视频文件的第一个1MB的分片，以及第二个1MB的分片已经存在，后续的分片上传(UploadPart)无需再上传这两个分片，只需补齐其他分片即可。


```
{
  "code": 1,
  "message": "",
  "codeDesc": "",
  "canRetry": 0,
  "listParts": [
  {
  "offset": 0,
  "dataSize": 1048576,
  "dataMd5": "0bee89b07a248e27c83fc3d5951213c1"
```

},
{
"offset": 1048576,
"dataSize": 1048576,
"dataSha": "f5ac8127b3b6b85cdc13f237c6005d80"
}
]
}

秒传：

{
"code": 2,
"message": "",
"fileId": "16092504232103571364",
"url": "http://10013.vod2.myqcloud.com/vod10013/16092504232103571364/f0.mp4"
}

上传失败：

{
"code": -10001,
"message": "invalid arg",
"canRetry" : 0
}

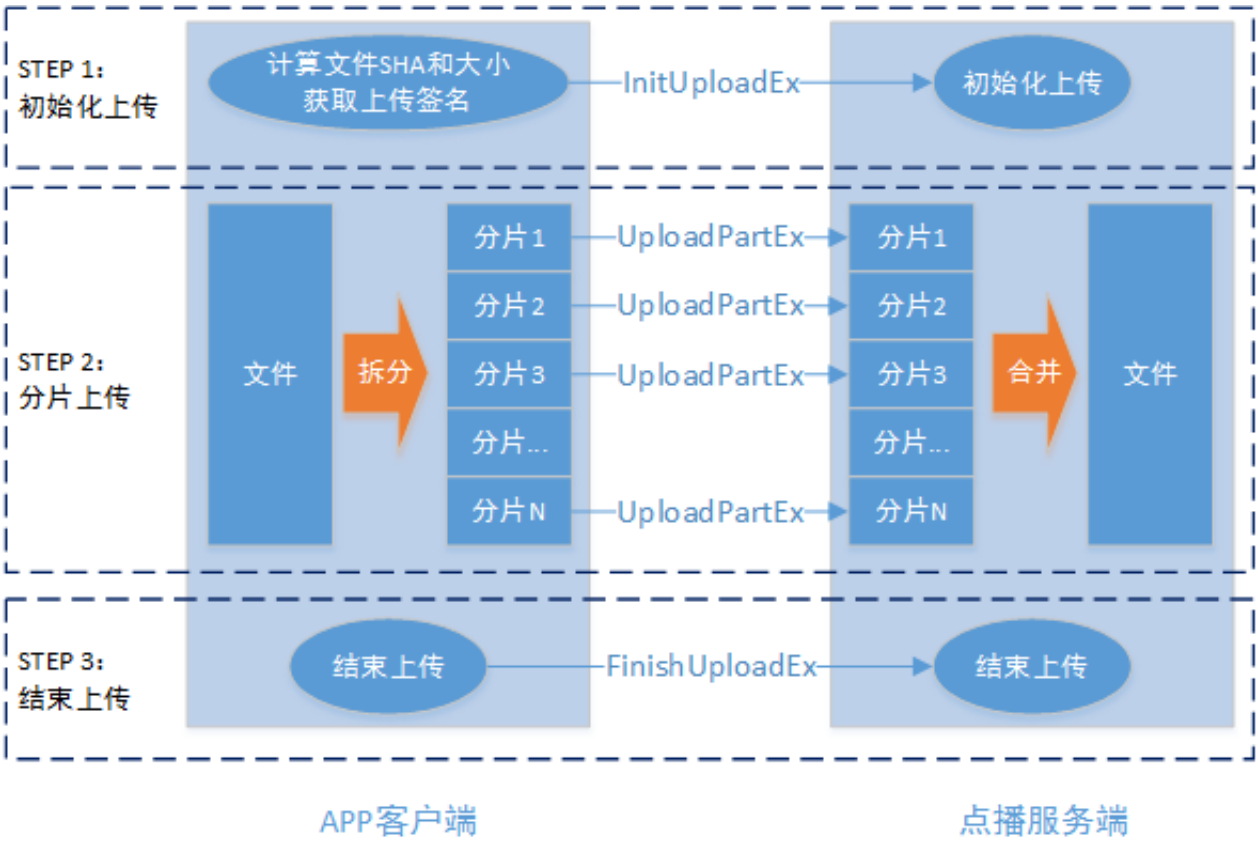
分片上传(UGC)

接口名称

UploadPartEx

功能说明

1. 该接口适用于APP将自己客户端上的视频上传到点播服务端；
2. 该接口亦可用于APP将自己服务端视频上传到点播服务端的场景，但我们建议使用服务端上传SDK或者服务端上传接口（InitUpload/UploadPart/FinishUpload）来进行上传；
3. 由于视频文件通常较大，故而一般需要采用分片上传的方式；整个上传过程涉及初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)三步，具体流程参见下图；
4. 支持秒传、断点续传；
5. 接口本身逻辑较为复杂，点播封装了多种语言的UGC上传SDK来简化开发者的调用，详见UGC视频上传综述。



请求方式

请求域名

vod2.qcloud.com

注意：

- UGC视频上传、服务端视频上传的域名与其他服务端API不同，不是vod.api.qcloud.com；
- UGC视频上传的签名(signature)生成方式与服务端API不同，方法参见UGC视频上传签名生成；
- 单个视频文件上传所调用的所有接口，包括初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)，均使用相同的signature；
- 该接口仅支持POST方法，不支持GET方法。

参数说明

参数名称	必填	类型	说明
fileSha	是	String	文件SHA（注意不是当前分片的SHA），必须与初始化上传(InitUpload)中所填写的fileSha一致
offset	是	Integer	分片在文件中的相对偏移，注意该值必须为dataSize的整数倍
dataSize	是	Integer	分片大小，详见下文介绍
dataMd5	是	String	该分片所上传数据的md5，计算方法详见
signature	是	String	UGC上传签名，参见UGC视频上传签名生成

dataMd5的计算方法：

1. 采用MD5散列算法计算整个视频文件的哈希值，得到128位(16字节)的二进制串；
2. 对得到的二进制串进行十六进制编码，编码算法必须使用小写字母，即a-f，而不是A-F。

关于dataSize：

1. 除了最后一个分片之外，其他分片dataSize的值必须与调用初始化上传(InitUpload)时传入的值保持一致；
2. 最后一个分片的dataSize必须小于等于初始化上传(InitUpload)时传入的dataSize值。

举例：

1. 假定需要上传一个6000000字节（大约5.72MB）的视频文件，初始化上传时设置的dataSize为1048576(1MB)；
2. 整个视频会被分为6个分片，对于前五个分片，其dataSize为1048576(1MB)，offset分别为0, 1048576, 2097152，3145728，4194304；
3. 对于最后一个分片，其dataSize为757120(计算方法：6000000 - 5 * 1048576)，offset为5242880。

请求示例

如下为请求URL，真正的视频数据需要在POST请求的body中提交。

```
https://vod2.qcloud.com/v2/index.php?Action=UploadPartEx
&fileSha=b4a5c70c76e79e01ab3a5c306de3d9eedeadeca9
&offset=0
&dataSize=1048576
&dataMd5=0bee89b07a248e27c83fc3d5951213c1
&signature=IEmbRAPy5IgIAFnt7XPAToaY3RRzPUFLSURVZ
```

接口应答

参数说明

参数名称	类型	说明
code	Integer	小于0：失败；0：成功
message	String	返回信息说明
codeDesc	String	后台记录的错误信息，腾讯云后台定位用
canRetry	Integer	当code小于0时，说明上传过程遇到错误；此时如果该值为1表示该错误可以通过重试来解决，否则表示该错误无法通过重试解决，必须进行排查

错误码说明

错误码	含义说明
4000-7000	参见公共错误码
-10001	检查公共参数出错
-10002	检查签名出错
-10003	检查协议参数出错
-10004	写缓存信息失败
-10005	获取缓存信息失败
-10006	包体非法

应答示例

```
{
  "code": 0,
  "message": "",
  "codeDesc": "",
  "canRetry": 0
}
```

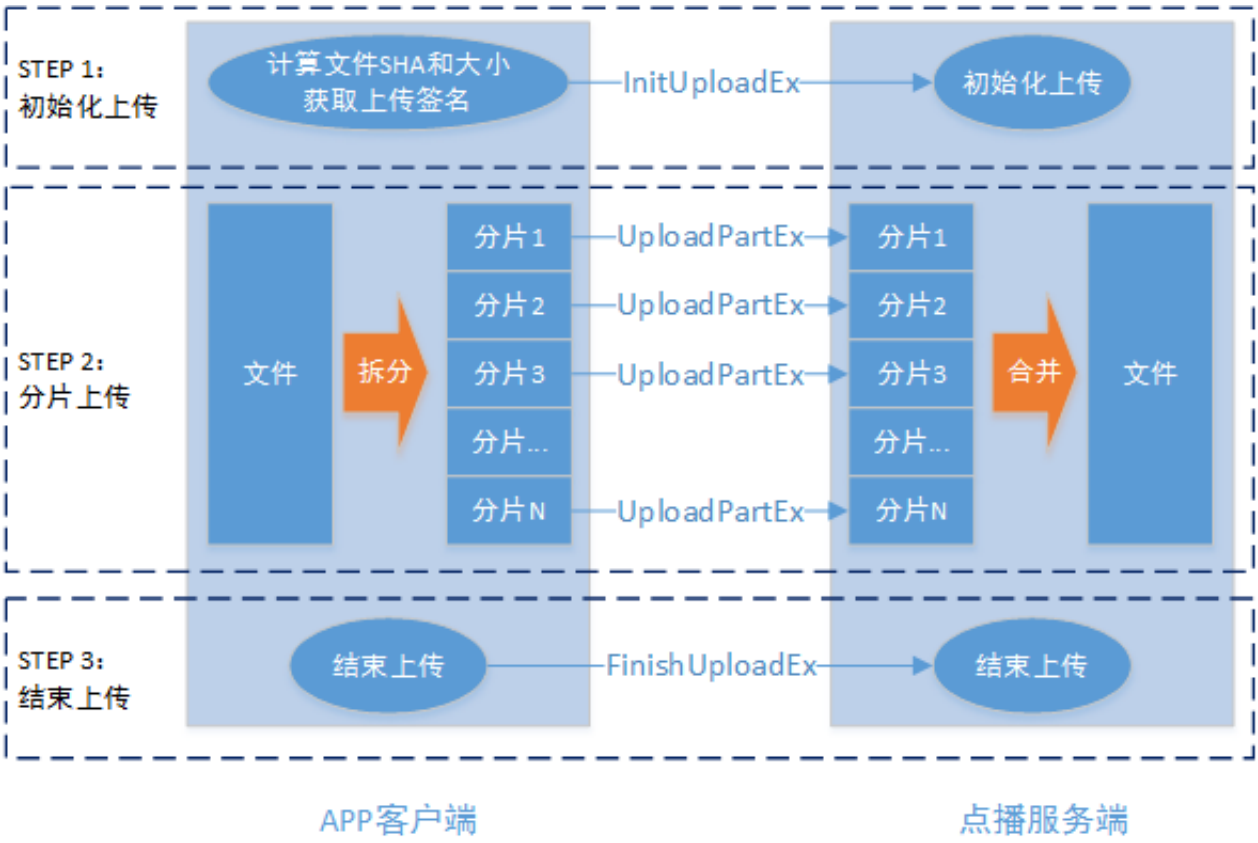
结束上传(UGC)

接口名称

FinishUploadEx

功能说明

1. 该接口适用于APP将自己客户端上的视频上传到点播服务端；
2. 该接口亦可用于APP将自己服务端视频上传到点播服务端的场景，但我们建议使用服务端上传SDK或者服务端上传接口（InitUpload/UploadPart/FinishUpload）来进行上传；
3. 由于视频文件通常较大，故而一般需要采用分片上传的方式；整个上传过程涉及初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)三步，具体流程参见下图；
4. 支持秒传、断点续传；
5. 若客户端上传成功后将fileID上报到APP服务器，支持APP服务器通过该接口提供的校验信息字段对fileID的合法性进行校验，防止客户端篡改上传文件的信息；
6. 接口本身逻辑较为复杂，点播封装了多种语言的UGC上传SDK来简化开发者的调用，详见UGC视频上传综述。



请求方式

请求域名

vod2.qcloud.com

注意：

- UGC视频上传、服务端视频上传的域名与其他服务端API不同，不是vod.api.qcloud.com；
- UGC视频上传的签名(signature)生成方式与服务端API不同，方法参见UGC视频上传签名生成；
- 单个视频文件上传所调用的所有接口，包括初始化上传(UGC)、分片上传(UGC)、结束上传(UGC)，均使用相同的signature；
- 该接口仅支持GET方法，不支持POST方法。

参数说明

参数名称	必填	类型	说明
fileSha	是	String	文件sha
signature	是	String	UGC上传签名，参见UGC视频上传签名生成

请求示例

https://vod2.qcloud.com/v2/index.php?Action=FinishUploadEx
&fileSha=b4a5c70c76e79e01ab3a5c306de3d9eedeadeca9
&signature=IEmbRAPy5IgIAFnt7XPAToaY3RRzPUFLSURVZ

接口应答

参数说明

参数名称	类型	说明
code	Integer	小于0：失败；0：成功
message	String	返回信息说明
codeDesc	String	后台记录的错误信息，腾讯云后台定位用
canRetry	Integer	当code小于0时，说明上传过程遇到错误；此时如果该值为1表示该错误可以通过重试来解决，否则表示该错误无法通过重试解决，必须进行排查
fileId	String	文件id
url	String	文件url
verify_content	String	信息合法性校验内容，当客户端向业务Server上报上传成功的fileId信息时，可以使用该字段校验上报的fileID是否合法（使用方式请见“合法性校验方式说明”），如果无需校验可

参数名称	类型	说明
		以忽略这个字段

错误码说明

错误码	含义说明
-10001	检查公共参数出错
-10002	检查签名出错
-10003	检查协议参数出错
-10004	写缓存信息失败
-10005	获取缓存信息失败
-10006	包体非法

其他错误码可能为网络原因导致的偶发失败。所有小于0的错误码，当canRetry=1时，均可重试解决。

应答示例

{
"code": 0,
"message": "",
"fileId": "7031868222808505913",
"verify_content":
"MzMyOTY0NGIwNTk4YTc2YzZjNDljNTk3YTJhNzNkOGE1ZjA3YWJlOUV4cFRpbWU9MTQ4ODE2MDI2NCZGaWxlSWQ9NzAzMTg2ODIyMjgwODUwNTkxMw=="
}

合法性校验方式说明

客户端结束上传后，将获得上传文件的fileId。一般的，APP服务器会要求客户端将上传的fileId上报给服务器记录下来，用于上传视频的管理。但是，如果用户单独上传fileId，APP服务器无法保证fileId的合法性，即用户有可能随意篡改上报的fileId。

为了校验用户上传fileID的合法性，用户可以要求客户端将结束上传得到的响应包体里的verify_content字段连同fileId一起上报到APP服务器，APP服务器可以用点播派发的verify_key来校验fileId是否合法。

####例子

如果用户有校验fileId合法的需求，可以向点播客户索要校验用的verify_key，这个例子中verify_key为字符串

"6367c48dd193d56ea7b0baad25b19455e529f5ee"

。

当客户端结束上传后，将获得fileId和verify_content，这里假设fileId和verify_content分别为字符串

"7031868222808505913"

和字符串

"MzMyOTY0NGIwNTk4YTc2YzZjNDljNTk3YTJhNzNkOGE1ZjA3YWJlOUV4cFRpbWU9MTQ4ODE2MDI2NCZGaWxlSWQ9NzAzMTg2ODIyMjgwODUwNTkxMw=="

。

客户端需要将fileId和verify_content一同上报给APP服务器。

APP服务器接收到verify_content后进行BASE64解码，得到一段二进制内容，其中前20字节为HashedContent，后面剩下的内容为PlainText。

PlainText为字符形式，例子中PlainText为字符串

"ExpTime=1488160264&FileId=7031868222808505913"

。APP服务器可以校验PlainText的内容是否已经过期（ExpTime超过当前服务器时间）以及上报内容是否匹配（FileId是否匹配）。如果内容已经过期或者不匹配，表示验证失败。

如果PlainText校验通过，则对PlainText使用verify_key作为秘钥进行HAMC-SHA1加密并得出结果（例子中的加密结果使用16进制表示为

3329644b0598a76c6c49c597a2a73d8a5f07abe9

），如果该结果能够与前面提到的HashedContent匹配，则验证通过，否则验证失败。

UGC视频上传签名生成

PHP示例

```
<?php
// Step 1：获取签名所需信息获取得到的签名所需信息，如下
$secret_id = "AKIDUfLUEUigQiXqm7CVSspKJnuaiIKtxqAv";
$secret_key = "bLcPnl88WU30VY57ipRhSePfPdOfSruK";

// Step 2：设置签名有效时间
$current = time();
$expired = $current + 86400; // 签名有效期：1天

// Step 3：根据客户端所提交的文件信息，拼装参数列表
$file_name = "tencent_test.mp4";
$file_sha = "a9993e364706816aba3e25717850c26c9cd0d89d";
$uid="1234";

$arg_list = array(
  "s" => $secret_id,
  "t" => $current,
  "e" => $expired,
  "f" => $file_name,
  "fs" => $file_sha,
  "ft" => $file_type,
  "uid" => $uid,
  "r" => rand());

// Step 4：生成签名
$orignal = http_build_query($arg_list);
$signature = base64_encode(hash_hmac('SHA1', $orignal, $secret_key, true).$orignal);

echo $signature;
echo "\n";
?>
```

Java示例

```
import java.util.Random;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import sun.misc.BASE64Encoder;

public class UgcSign {
  public long m_qwNowTime;

  public String m_strFileName;
  public String m_strFileSha;

  public String m_strFileType;
  public int m_iRandom;

  public String m_strSecId = "AKIDR20GpXsc4fixxxxxxxbuWQCeTpw9ljzt";
  public String m_strSecKey = "wGxKo4cu6WFBWxxxxxxxbH7BTTiUn4bV";

  public int m_isTrans = 0;
  public int m_isScreenShot = 0;
  public int m_isWaterMark = 0;

  public int m_iClassId = 0;
  public int m_iSignValidDuration = 3600 * 24 * 2;

  public String m_strUid;

  private static final String HMAC_ALGORITHM = "HmacSHA1";
  private static final String CONTENT_CHARSET = "UTF-8";

  public static byte[] byteMerger(byte[] byte_1, byte[] byte_2) {
  byte[] byte_3 = new byte[byte_1.length + byte_2.length];
```

```
System.arraycopy(byte_1, 0, byte_3, 0, byte_1.length);
System.arraycopy(byte_2, 0, byte_3, byte_1.length, byte_2.length);
return byte_3;
}

String GetUgcExSign() {
String strSign = "";
String contextStr = "";
long endTime = (m_qwNowTime + m_iSignValidDuration);
try {
contextStr += "f=" + java.net.URLEncoder.encode(m_strFileName, "utf8");
contextStr += "&fs=" + this.m_strFileSha;
contextStr += "&ft=" + this.m_strFileType;

contextStr += "&t=" + this.m_qwNowTime;
contextStr += "&e=" + endTime;
contextStr += "&r=" + this.m_iRandom;
contextStr += "&s=" + java.net.URLEncoder.encode(this.m_strSecId, "utf8");
contextStr += "&uid=" + m_strUid;
contextStr += "&tc=" + this.m_isTrans;
contextStr += "&ss=" + this.m_isScreenShot;
contextStr += "&wm=" + this.m_isWaterMark;

if (this.m_iClassId != 0) {
contextStr += "&cid=" + this.m_iClassId;
}

String s = contextStr;
String sig = null;
Mac mac = Mac.getInstance(HMAC_ALGORITHM);
SecretKeySpec secretKey = new SecretKeySpec(m_strSecKey.getBytes(CONTENT_CHARSET),
mac.getAlgorithm());
mac.init(secretKey);
byte[] hash = mac.doFinal(contextStr.getBytes(CONTENT_CHARSET));
```

```
  byte[] sigBuf = byteMerger(hash, contextStr.getBytes("utf8"));
  // base64
  strSign = new String(new BASE64Encoder().encode(sigBuf).getBytes());
  // Echo(strSign);
  } catch (Exception e) {
  System.out.printf(e.toString());
  return "";
  }
  return strSign;
  }
}

class Test {
  public static void main(String[] args) {
  UgcSign sign = new UgcSign();
  //从https://console.qcloud.com/capi获取，分别对应SecretId和SecretKey
  sign.m_strSecId = "AKIDR20GpXsc4fixxxxxxxbuWQCeTpw9ljzt";
  sign.m_strSecKey = "wGxKo4cu6WFBWxxxxxxxbH7BTTiUn4bV";

  //当前时间
  sign.m_qwNowTime = System.currentTimeMillis() / 1000;
  sign.m_iRandom = new Random().nextInt(java.lang.Integer.MAX_VALUE);

  //上传者在app内的唯一标识，如果不需要，可以保持不变
  sign.m_strUid = "123";

  //文件名
  sign.m_strFileName = "test";
  //文件sha。由客户端算好带到后台
  sign.m_strFileSha = "534928ad7165be24caaaaaf28568be23497f1467";
  //文件类型
  sign.m_strFileType = "mp4";
  //分类id
  sign.m_iClassId = 0;
```

```
  //签名有效期
  sign.m_iSignValidDuration = 3600 * 24 * 2;
  //转码/截图/水印开关
  sign.m_isTrans = 1;
  sign.m_isScreenShot = 1;
  sign.m_isWaterMark = 1;

  System.out.printf(sign.GetUgcExSign());
  }
}
```

C#示例

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

namespace GetUgcSign
{
  class UgcSign
  {
  public string m_strSecId;
  public string m_strSecKey;

  public string m_strFileName;
  public string m_strFileSha;

  public string m_strFileType;

  public string m_strUid;
  public int m_iRandom;

  public int m_isTrans = 0;
  public int m_isScreenShot = 0;
  public int m_isWaterMark = 0;

  public int m_iClassId;

  public long m_qwNowTime;
  public int m_iSignValidDuration;
  public static long GetIntTimeStamp()
   {
  TimeSpan ts = DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, 0);
  return Convert.ToInt64(ts.TotalSeconds);
  }
```

```
private byte[] hash_hmac_byte(string signatureString, string secretKey)
 {
var enc = Encoding.UTF8;
HMACSHA1 hmac = new HMACSHA1(enc.GetBytes(secretKey));
hmac.Initialize();

byte[] buffer = enc.GetBytes(signatureString);
return hmac.ComputeHash(buffer);
}
public string GetSign()
 {
string strContent = "";
strContent += ("s=" + Uri.EscapeDataString((m_strSecId)));
strContent += ("&f=" + Uri.EscapeDataString(m_strFileName));
strContent += ("&t=" + m_qwNowTime);
strContent += ("&ft=" + m_strFileType);
strContent += ("&e=" + m_qwNowTime + m_iSignValidDuration);
strContent += ("&fs=" + m_strFileSha);
strContent += ("&uid=" + m_strUid);
strContent += ("&r=" + m_iRandom);

strContent += "&tc=" + m_isTrans;
strContent += "&ss=" + m_isScreenShot;
strContent += "&wm=" + m_isWaterMark;

byte[] bytesSign = hash_hmac_byte(strContent, m_strSecKey);
byte[] byteContent = System.Text.Encoding.Default.GetBytes(strContent);
byte[] nCon = new byte[bytesSign.Length + byteContent.Length];
bytesSign.CopyTo(nCon, 0);
byteContent.CopyTo(nCon, bytesSign.Length);
return Convert.ToBase64String(nCon);
}
}
class Program
```

```
{
static void Main(string[] args)
 {
UgcSign sign = new UgcSign();
//从https://console.qcloud.com/capi获取，分别对应SecretId和SecretKey
sign.m_strSecId = "AKIDR20GpXsc4fixxxxxxxbuWQCeTpw9ljzt";
sign.m_strSecKey = "wGxKo4cu6WFBWxxxxxxxbH7BTTiUn4bV";

//当前时间
sign.m_qwNowTime = UgcSign.GetIntTimeStamp();
sign.m_iRandom = new Random().Next(0, 1000000);

//上传者在app内的唯一标识，如果不需要，可以保持不变
sign.m_strUid = "123";

//文件名
sign.m_strFileName = "test";
//文件sha。由客户端算好带到后台
sign.m_strFileSha = "534928ad7165be24caaaaaf28568be23497f1467";
//文件类型
sign.m_strFileType = "mp4";
//分类id
sign.m_iClassId = 0;
//签名有效期
sign.m_iSignValidDuration = 3600 * 24 * 2;
//转码/截图/水印开关
sign.m_isTrans = 1;
sign.m_isScreenShot = 1;
sign.m_isWaterMark = 1;

string strSign = sign.GetSign();
Console.WriteLine(strSign);
//Thread.Sleep(1000000);
}
```

```
  }
}
```

Node.js示例

```
var querystring = require("querystring");
var crypto = require('crypto');

// Step 1：获取签名所需信息获取得到的签名所需信息，如下
var secret_id = "AKIDUfLUEUigQiXqm7CVSspKJnuaiIKtxqAv";
var secret_key = "bLcPnl88WU30VY57ipRhSePfPdOfSruK";

// Step 2：设置签名有效时间
var current = parseInt((new Date()).getTime() / 1000)
var expired = current + 86400; // 签名有效期：1天

// Step 3：根据客户端所提交的文件信息，拼装参数列表
var file_name = "tencent_test.mp4";
var file_sha = "a9993e364706816aba3e25717850c26c9cd0d89d";
var uid="1234";
var ft="AVI";

var arg_list = {
  s : secret_id,
  t : current,
  e : expired,
  f : file_name,
  fs : file_sha,
  ft : file_type,
  r : Math.round(Math.random() * Math.pow(10, 10)),
  uid : uid
}

// Step 4：生成签名
var orignal = querystring.stringify(arg_list);
var orignal_buffer = new Buffer(orignal, "utf8");

var hmac = crypto.createHmac("sha1", secret_key);
```

```
var hmac_buffer = hmac.update(orignal_buffer).digest();

var signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");

console.log(signature);
```