点播

视频上传

产品文档

腾讯云

文档目录

视频上传
视频上传综述

最近更新时间：2018-05-28 14:24:22

视频上传方式综述

所谓视频上传，是指开发者或其用户将视频文件上传到点播的视频存储中，以便进行视频处理、分发等。腾讯云点播支持如下几种视频上传方式：

- 控制台上传：在点播控制台上进行操作，将本地视频上传到云点播，适用于直接管理少量视频的场景，具有方便快捷、无技术门槛的优点；
- 服务端上传：开发者将存储在其后台服务器中的视频上传到云点播，适用于自动化、系统化的运营场景；
- 客户端上传：终端用户将客户端本地视频上传到云点播，适用于 UGC、PGC 等场景，支持如下三端：
  - Android 端上传 SDK；
  - iOS 端上传 SDK；
  - Web 端上传 SDK。
- 离线拉取上传：将其他站点的视频拉取到云点播中，适用于小批量视频迁移场景；
- 源站同步：将保存在其他存储系统的视频迁移到点播系统中，适用于大规模视频迁移场景。

视频上传能力综述

文件类型

支持的视频文件格式如下：

- AVI 格式：文件后缀为 AVI；
- QuickTime 格式：文件后缀为 MOV；
- MPEG 格式：文件后缀为 MPG, MPEG, MPE, DAT, VOB, ASF, 3GP, MP4 等；
- WMV 格式：文件后缀为 WMV, ASF；
- Real 格式：文件后缀为 RM, RMVB；
- Flash 格式：文件后缀为 FLV, F4V；
- Matroska 格式：文件后缀为 MKV。

支持的音频文件格式如下：

- MP3；
- AAC；

- OGG；
- FLAC。

上传时附带封面

在发起视频上传时，可以额外指定一张本地图片，来作为视频的封面。在进行视频播放时，该图片可以作为视频的内容预览图片。参见：

- 客户端上传指引；
- 服务端上传指引。

上传时指定指定视频处理方式

部分场景下，开发者需要在视频上传完成之后自动进行转码等视频处理操作；app 可以在上传视频的同时，指定视频处理方式；当视频上传完成之后，点播后台将依照 app 指定的方式来处理视频。参见：

- 客户端上传指引；
- 服务端上传指引。

客户端断点续传

客户端视频上传的过程中，如果出现异常（例如程序崩溃），则在尝试重新上传视频的过程中，已经完成上传的部分不需要再次上传。参见：

- Android 端上传 SDK；
- iOS 端上传 SDK；
- Web 端上传 SDK。

暂停/恢复上传

客户端在视频上传的过程中，可以暂停上传、恢复上传。参见：

- Android 端上传 SDK；
- iOS 端上传 SDK；
- Web 端上传 SDK。

取消上传

客户端在视频上传的过程中，可以取消上传。参见：

- Android 端上传 SDK；
- iOS 端上传 SDK；
- Web 端上传 SDK。

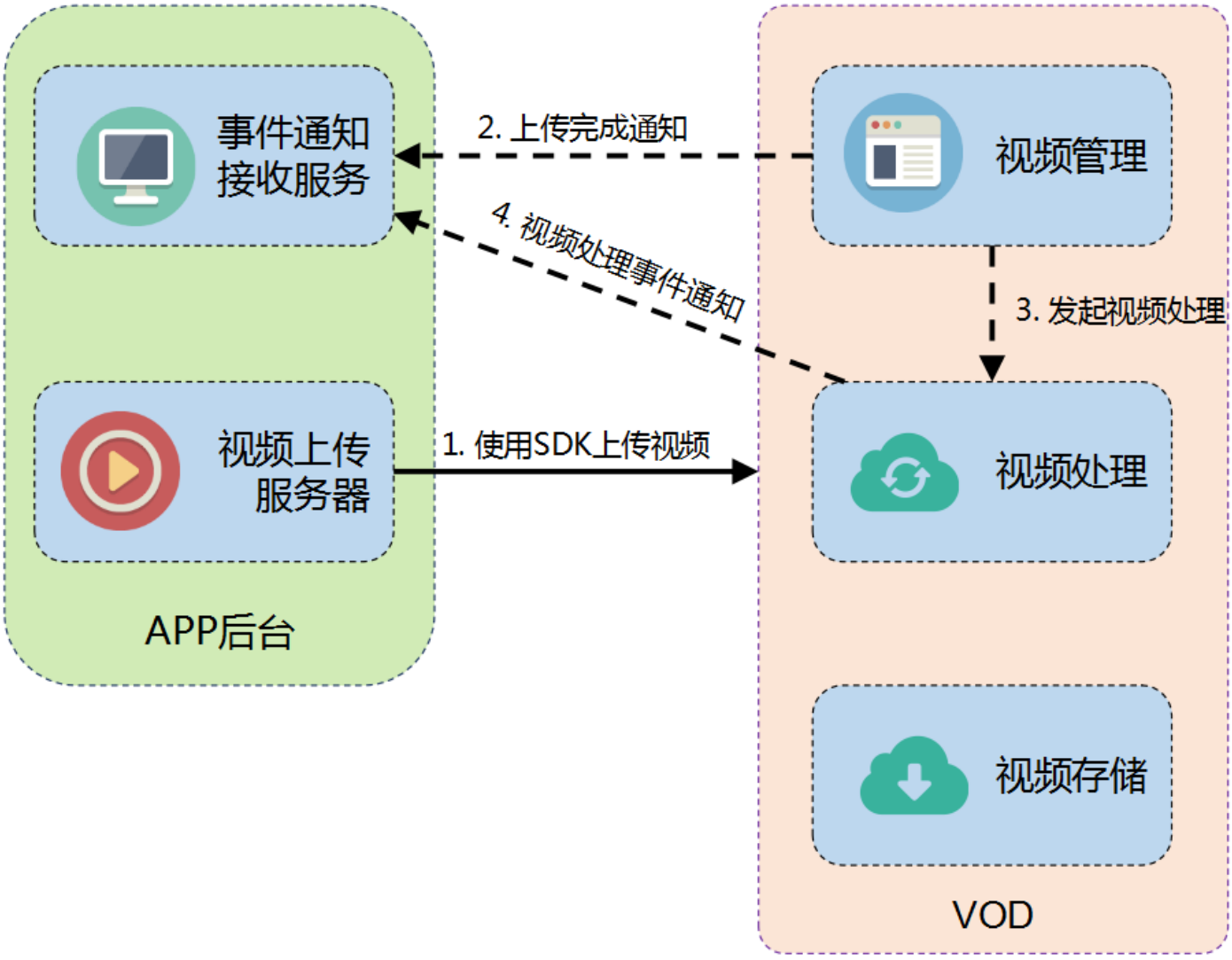
视频上传事件通知

在视频上传过程中，可以经过配置，将视频上传这一事件通知 app 服务器。参见：
事件通知-视频上传完成

服务端上传
服务端上传指引

最近更新时间：2018-05-28 14:35:51

所谓服务端视频上传，是指 App **后台**将视频上传到点播平台。服务端上传的整体流程如下图所示。



准备工作

开通服务

如果您尚未开通点播服务，则需要先开通服务，参见购买流程。

获取云API密钥

服务端上传涉及多个服务端 API，故而需要首先获取调用服务端 API 所需的安全凭证，即 SecretId 和 SecretKey。其具体步骤如下：

Step1：登录腾讯云管理中心控制台。

Step2：单击【云产品】，选择【监控与管理】栏下的【云 API 密钥】，进入云 API 密钥管理页面，如下图所示：

Step3：获取云 API 密钥，如下图所示。如果您尚未创建密钥，则单击【新建】即可创建一对 SecretId/SecretKey。

发起上传

视频上传分为申请上传、上传文件、确认上传三步，分别对应上传流程图中的第 1、2、3 步。

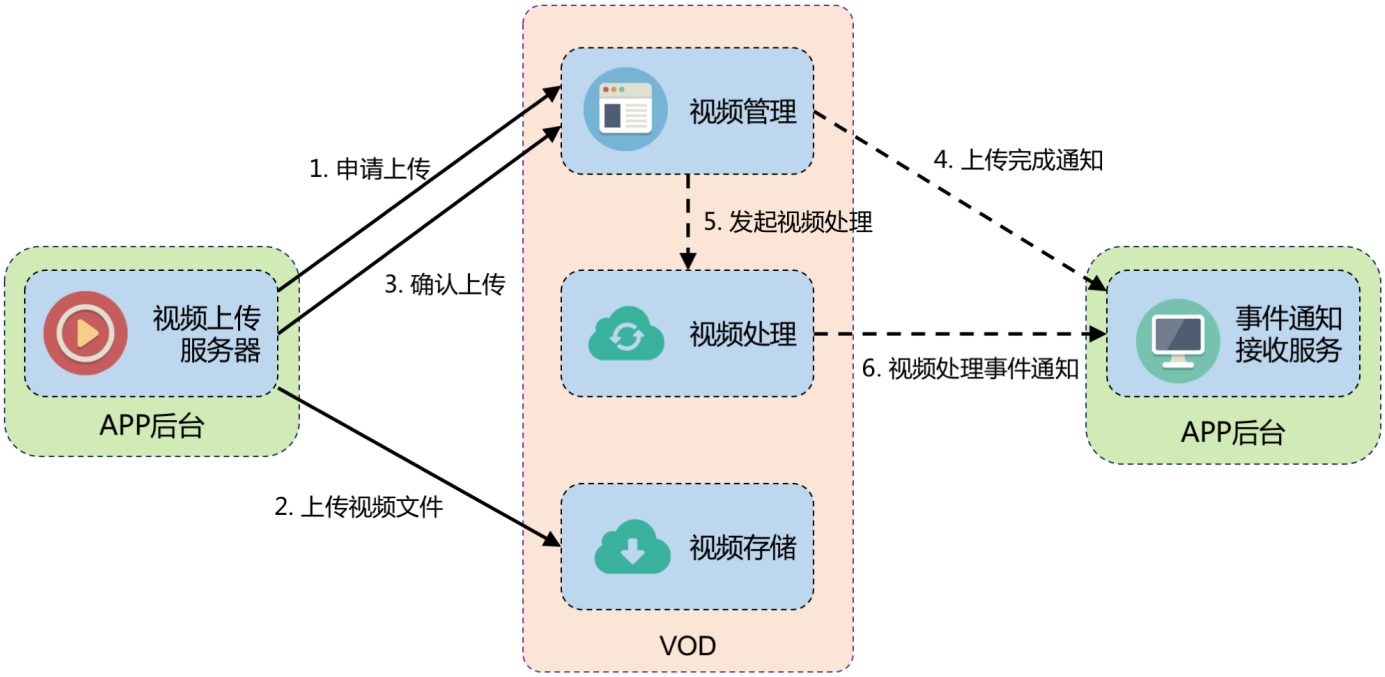
通过点播服务端上传 SDK 发起上传

为了方便用户开发端上传功能，云点播提供了基于多语言平台 SDK，每种语言的 SDK 均包含相应 DEMO。参见：

- PHP SDK；
- Java SDK。

通过服务端 API 发起上传

如果点播提供的上传 SDK 并未涵盖 App 后台所使用的语言，则 App 后台需要自行调用点播的服务端 API 进行视频上传。这种方式流程较为复杂，不推荐作为首选。基于 API 上传的完整业务流程图如下：



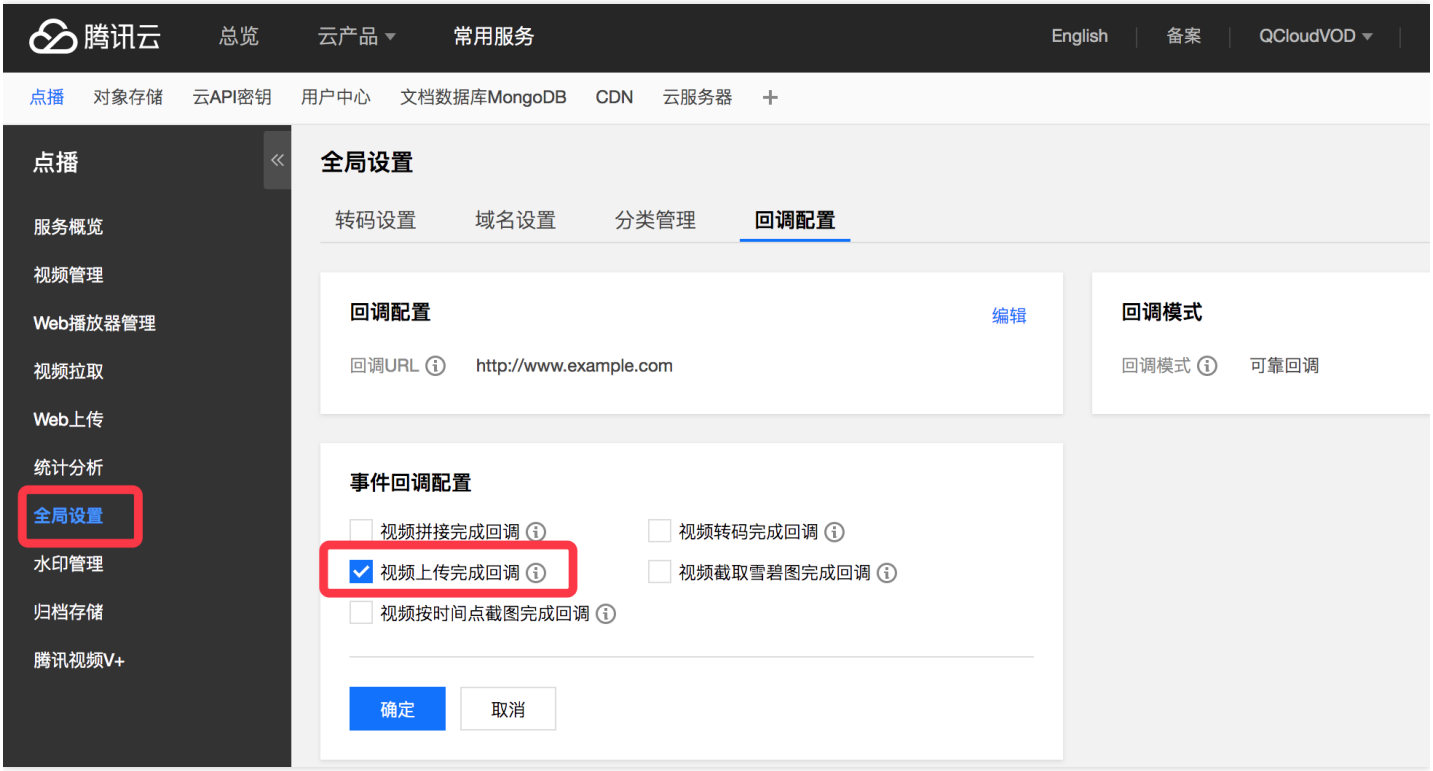
可以看到比起基于 SDK 方式的上传，基于 API 的服务端视频上传需要自行实现「申请上传」、「上传视频文件」等步骤。而「上传视频文件」也没有基于 SDK 的方式上传方便，对于大文件需要自己做分片上传等逻辑。具体参见：

- 服务端 API：申请上传；
- 服务端 API：上传文件；
- 服务端 API：确认上传。

事件通知

视频上传完成之后，腾讯云点播会给 App 后台发起事件通知-视频上传完成，App 后台可以据此感知到视频上传行为。

要接收事件通知，App 首先需要到点播控制台开通事件通知，如下图所示：



事件通知-视频上传完成主要包含如下几项信息：

- 新视频文件的 FileId 和 URL；
- 点播支持在视频上传时指定透传字段，事件完成将透传字段通知到 App 后台。在事件通知中，有 2 个字段：`SourceType`（该字段被腾讯云固定成 `ServerUpload`，表示上传来源为服务端上传）和用户自定义透传字段 `SourceCopntext`，`SourceContext` 是 App 后台在派发签名时指定的透传内容（对应签名中的 `sourceContext` 参数）；
- 点播支持在视频上传完成时自动发起视频处理，如果上传时指定了视频处理任务流，则在事件通知内容中也会携带任务 ID（事件通知中的 `data.procedureTaskId` 字段）。

更多参见：

- 任务管理与事件通知；
- 事件通知-视频上传完成。

高级功能

上传时指定任务流

如果开发者需要在视频上传完成后自动发起视频处理任务流（例如转码、截图等），可以在调用服务端 API 申请上传时通过 `procedure` 参数来实现。

示例1：以下设置表示在视频上传完成之后，按照控制台【全局设置】-【转码设置】中的配置进行转码，并在转码时打上【水印管理】中的默认水印：

```
procedure=QCVB_SimpleProcessFile(1,1)
```

示例2：以下设置表示在视频上传完成之后，使用转码模板 10，20 进行转码；转码过程使用水印模板 150 打水印；使用指定时间点截图模板 10 截取首帧设置封面；使用采样截图模板 10 进行采样截图：

```
procedure=QCVB_SimpleProcessFile({10,20},150,10,10)
```

`procedure` 参数亦可填写为其他点播内置任务流，或者开发者自定义的任务流（目前自定义功能未开放，需提工单）。

上传时指定存储区域

目前服务端上传默认是将存储到广州，但点播也支持存储到其他区域。如果需要存储到其他区域，可以提工单申请其他存储区域。之后上传时就可以指定区域，参见：

- PHP SDK；
- Java SDK。

更多参见：

- 参数模板与任务流；
- 点播内置任务流 QCVB_SimpleProcessFile。

Java SDK

最近更新时间：2018-05-28 14:23:59

对于在服务端上传视频的场景，腾讯云点播提供了 Java SDK 来实现。上传的流程可以参见 服务端上传指引。

集成方式

Maven依赖引入

在项目的pom.xml文件添加点播SDK依赖即可：

```
<dependency>
<groupId>com.qcloud</groupId>
<artifactId>vod_api</artifactId>
<version>1.1.1</version>
</dependency>
```

jar包导入

如果项目没有采用Maven的方式进行依赖管理，可采用下述方式，下载各个所需的jar包，导入项目即可：

jar文件	说明
vod_api-1.1.0.jar	点播SDK
jackson-annotations-2.8.0.jar,jackson-core-2.8.5.jar,jackson-databind-2.8.5.jar	开源的JSON相关库
cos_api-5.1.9.jar	腾讯云对象存储服务（COS）SDK
qcloud-java-sdk-2.0.1.jar	腾讯云API SDK
commons-codec-1.10.jar,commons-logging-1.2.jar,log4j-1.2.17.jar,slf4j-api-1.7.21.jar,slf4j-log4j12-1.7.21.jar	开源日志相关库
httpclient-4.5.3.jar,httpcore-4.4.6.jar,httpmime-4.5.2.jar	开源的http处理库
joda-time-2.9.6.jar	开源时间处理库

为方便客户使用，可单击下载下面提供的链接，将下载的jar包导入项目中，即可使用：

点播Java SDK关联jar包

简单视频上传

初始化一个上传对象

使用云API密钥初始化VodApi实例

```
VodApi vodApi = new VodApi("your secretId", "your secretKey");
```

调用上传

传入视频地址进行上传

```
VodUploadCommitResponse response = vodApi.upload("/data/videos/Wildlife.wmv");
System.out.println(response.getFileId());
```

高级功能

携带封面

调用upload的重载方法，传入封面地址

```
VodApi vodApi = new VodApi("your secretId", "your secretKey");
VodUploadCommitResponse response = vodApi.upload("/data/videos/Wildlife.wmv", "/data/videos/
Wildlife.jpg");
System.out.println(response.getFileId());
```

指定任务流

调用upload的重载方法，传入任务流参数，具体的任务流介绍参考任务流综述，视频上传成功后会自动执行任务流

```
VodApi vodApi = new VodApi("your secretId", "your secretKey");
VodUploadCommitResponse response = vodApi.upload("/data/videos/Wildlife.wmv", "/data/videos/
Wildlife.jpg", "QCVB_SimpleProcessFile(1, 1)");
System.out.println(response.getFileId());
```

指定上传区域

调用upload的重载方法，传入指定的地域标识，即可将视频上传指定的区域，详见服务端上传指引。

```
VodApi vodApi = new VodApi("your secretId", "your secretKey");
Map<String, Object> extraParams = new HashMap<String, Object>();
```

```
extraParams.put("storageRegion", "tj");
VodUploadCommitResponse response = vodApi.upload("/data/videos/Wildlife.wmv", "/data/videos/Wildlife.jpg", extraParams);
System.out.println(response.getFileId());
```

接口描述

初始化上传对象 `VodApi`

参数名称	参数描述	类型	必填
secretId	云API密钥ID	String	是
secretKey	云API密钥Key	String	是

上传方法 `VodApi.upload(String videoPath)`

参数名称	参数描述	类型	必填
videoPath	视频路径	String	是

上传重载方法 `VodApi.upload(String videoPath, String coverPath)`

参数名称	参数描述	类型	必填
videoPath	视频路径	String	是
coverPath	封面路径	String	是

上传重载方法 `VodApi.upload(String videoPath, String coverPath, String procedure)`

参数名称	参数描述	类型	必填
videoPath	视频路径	String	是
coverPath	封面路径	String	是
procedure	任务流	String	是

上传重载方法 `VodApi.upload((String videoPath, String coverPath, Map<String, Object> extraParams)`

参数名称	参数描述	类型	必填
videoPath	视频路径	String	是
coverPath	封面路径	String	是
extraParams	任务流	Map	是

extraParams支持下述参数

参数名称	参数描述	类型
sourceContext	用户自定义上下文	String
storageRegion	指定存储地区	String
procedure	任务流	String

上传结果 `VodUploadCommitResponse`

成员变量名称	变量说明	类型
code	结果码	int
message	上传失败的错误描述	String
fileId	点播视频文件Id	String
video	视频存储信息	Object
video.url	视频存储地址	String
video.verify_content	视频存储校验信息	String
cover	封面存储信息	Object
cover.url	封面存储地址	String
cover.verify_content	封面存储校验信息	String

错误码列表

调用SDK上传后，可以利用 `VodUploadCommitResponse` 中的 code 来确认视频上传的情况

状态码	含义
0	上传成功
31001	用户请求session_key错误
31002	用户请求中的VOD签名重复
31003	上传文件不存在
32001	服务错误

PHP SDK

最近更新时间：2018-05-28 14:23:10

对于在服务端上传视频的场景，腾讯云点播提供了 PHP SDK 来实现。上传的流程可以参见 服务端上传指引。

集成方式

使用 composer 引入

```
{
"require": {
"qcloud/vod-sdk-v5": "v1.2.1"
}
}
```

源文件导入

如果项目当中没有使用 composer 工具进行依赖管理的，可以直接下载源码导入项目中使用：

- 从 Github 访问 >>
- 单击下载 PHP SDK >>

复制 src 文件下的源码和 test/non-composer 文件的 cos-sdk-v5、qcloudapi-sdk-php 到项目同级目录即可

简单视频上传

初始化上传对象

使用云API密钥初始化VodApi

对于使用 composer 导入的

```
<?php
require 'vendor/autoload.php';

use Vod\VodApi;

VodApi::initConf("your secretId", "your secretKey");
```

对于使用源码导入的

```php
<?php
require './cos-sdk-v5/cos-autoloader.php';
require './qcloudapi-sdk-php/src/QcloudApi/QcloudApi.php';
require './src/Vod/VodApi.php';
require './src/Vod/Conf.php';

use Vod\VodApi;

VodApi::initConf("your secretId", "your secretKey");
```

调用上传

传入视频地址进行上传

```php
$result = VodApi::upload(
array (
'videoPath' => '/data/videos/Wildlife.wmv',
)
);
echo "upload to vod result: " . json_encode($result) . "\n";
```

高级功能

携带封面

同时传入视频地址和封面地址

```php
$result = VodApi::upload(
array (
'videoPath' => '/data/videos/Wildlife.wmv',
'coverPath' => '/data/videos/Wildlife.jpg',
)
);
echo "upload to vod result: " . json_encode($result) . "\n";
```

指定任务流

传入任务流参数，具体的任务流介绍参考任务流综述，视频上传成功后会自动执行任务流

```php
$result = VodApi::upload(
array (
```

```
'videoPath' => '/data/videos/Wildlife.wmv',
'coverPath' => '/data/videos/Wildlife.jpg',
),
array (
'procedure' => 'QCVB_SimpleProcessFile(1, 1)'
)
);
echo "upload to vod result: " . json_encode($result) . "\n";
```

指定上传区域

传入指定的地域标识，即可将视频上传指定的区域，详见服务端上传指引。

```
$result = VodApi::upload(
array (
'videoPath' => '/data/videos/Wildlife.wmv',
'coverPath' => '/data/videos/Wildlife.jpg',
),
array (
'storageRegion' => 'tj'
)
);
echo "upload to vod result: " . json_encode($result) . "\n";
```

接口描述

初始化上传对象 `VodApi::initConf(secretId, secretKey)`

参数名称	参数描述	类型	必填
secretId	云API密钥ID	String	是
secretKey	云API密钥Key	String	是

上传方法 `VodApi.upload(src, parameter)`

src 参数

参数名称	参数描述	类型	必填
videoPath	视频路径	String	是
coverPath	封面路径	String	否

参数名称	参数描述	类型	必填

parameter 参数

参数名称	参数描述	类型	必填
videoName	视频名称	String	否
sourceContext	用户自定义上下文	String	否
storageRegion	指定存储地区	String	否
procedure	任务流	String	否

上传结果

成员变量名称	变量说明	类型
code	结果码	int
message	提示信息	String
data	返回数据	Object
data.fileId	点播视频文件Id	String
data.video.url	视频存储地址	String
data.cover.url	封面存储地址	String

错误码列表

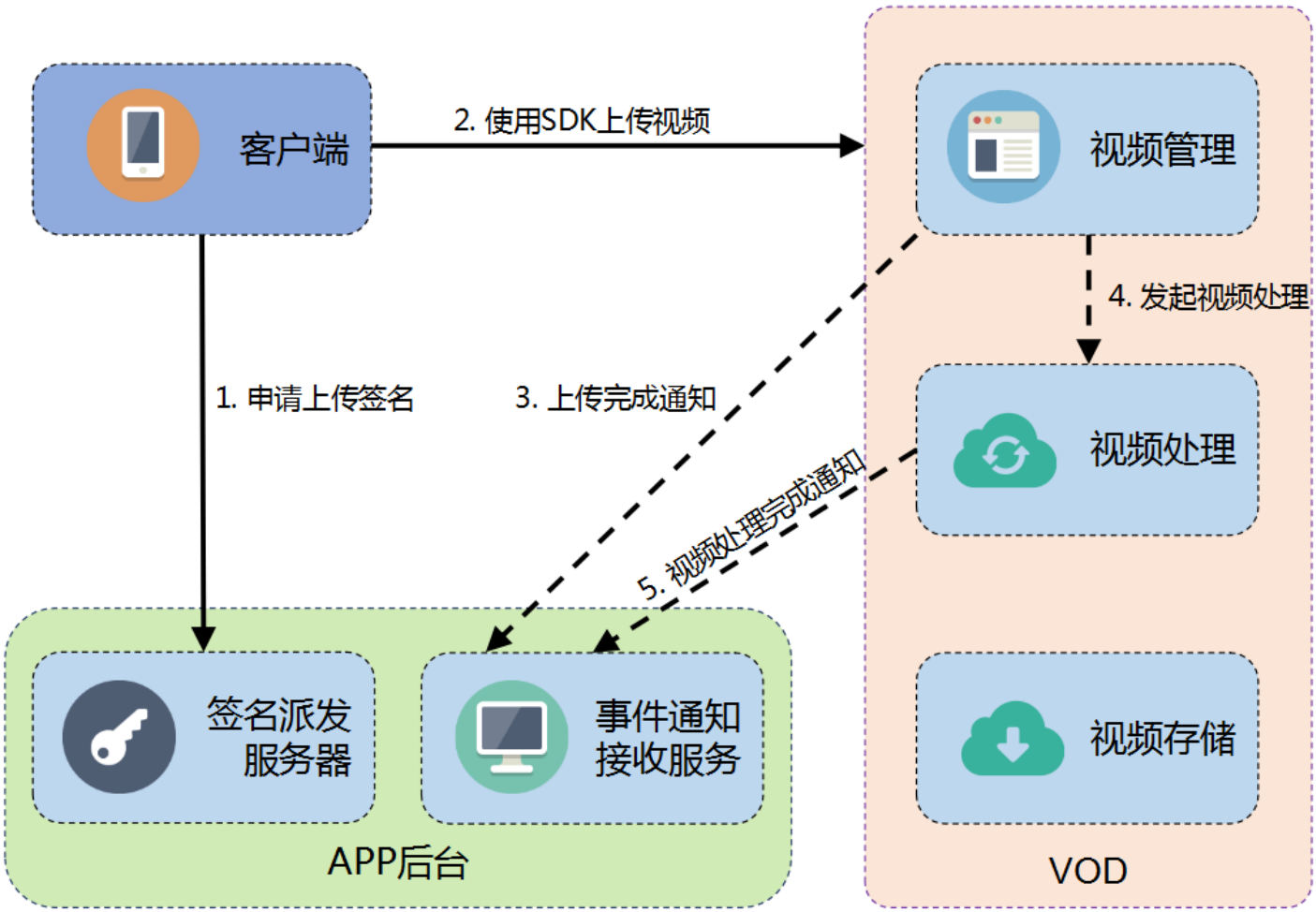
调用SDK上传后， 可以根据结果中的 code 来确认视频上传的情况

状态码	含义
0	上传成功
31001	用户请求session_key错误
31002	用户请求中的VOD签名重复
31003	上传文件不存在
32001	服务错误

客户端上传
客户端上传指引

最近更新时间：2018-05-28 14:28:13

所谓客户端视频上传，是指 App 的**最终用户**将本地视频上传到点播平台。客户端上传的整体流程如下图所示。



准备工作

开通服务

如果您尚未开通点播服务，则需要先开通服务，参见 购买流程。

获取云 API 密钥

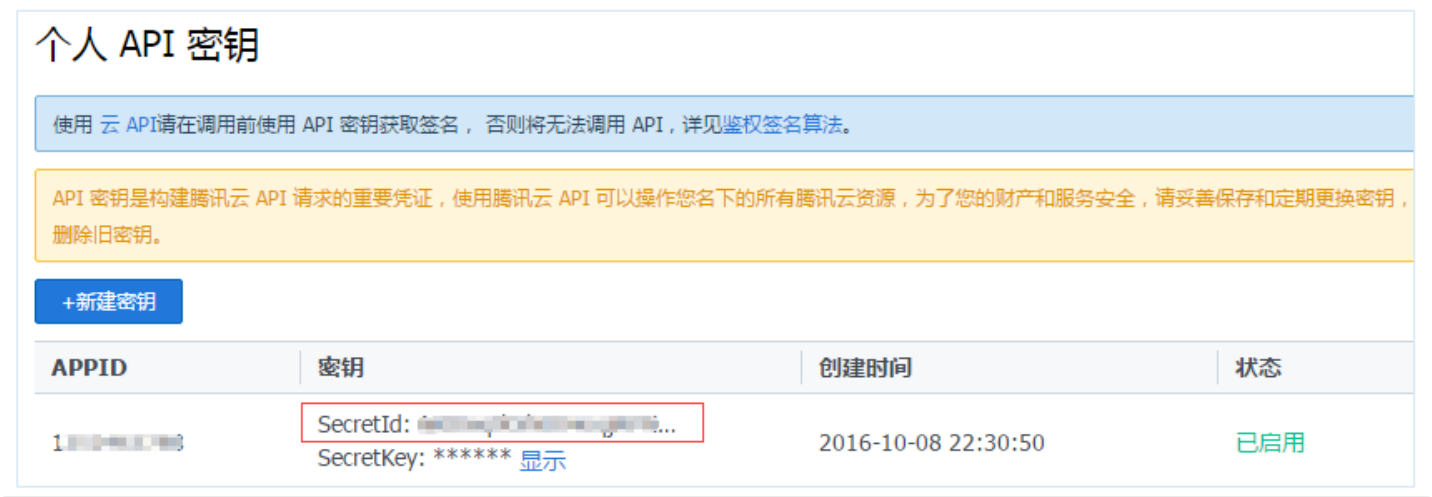
客户端上传在申请上传签名的时候需要使用腾讯云密钥，即 SecretId 和 SecretKey。

其具体步骤如下：

Step1：登录 腾讯云管理中心控制台。

Step2：单击【云产品】，选择【监控与管理】栏下的【云 API 密钥】，进入云 API 密钥管理页面，如下图所示：

Step3：获取云 API 密钥，如下图所示。如果您尚未创建密钥，则单击【新建】即可创建一对 SecretId/SecretKey。



搭建签名派发服务

在客户端上传场景下，客户端是直接将视频文件上传到腾讯云点播，不需要由 App 服务端进行中转。因此，腾讯云点播必须对发起请求的客户端进行鉴权。但由于 SecretKey 的权限过大，App 不应该将此信息泄露到客户端，否则将会造成严重的安全问题。

因此，客户端在发起上传之前，必须要到 App 的签名派发服务申请上传签名，即流程图中的第 1 步。该步骤可以走 App 的客户端和服务器之间的任何安全通道。

App 服务器生成上传签名的算法参见：客户端上传签名。

多语言签名生成示例代码参见：

- PHP 签名生成示例；
- Java 签名生成示例；
- Node.js 签名生成示例；
- C# 签名生成示例。

服务搭建完毕之后，开发者可以通过腾讯云点播提供的工具来校验签名的正确性：

- 签名生成工具：根据参数和密钥，快速生成签名；
- 签名校验工具：对签名进行解析，得到生成签名时所使用的各项参数。

客户端集成

点播上传提供 Android、iOS、Web 三大平台的 SDK 方便客户接入。参见：

- Android 上传 SDK；
- iOS 上传 SDK；
- Web 上传 SDK。

事件通知

视频上传完成之后，腾讯云点播会给 App 后台发起 事件通知-视频上传完成，即业务流程图中的第 3 步，App 台可以据此感知到视频上传行为。
要接收事件通知，App 首先需要到点播控制台开通事件通知，参见 回调配置
事件通知-视频上传完成 主要包含如下几项信息：

- 新视频文件的 FileId 和 URL；
- 点播支持在视频上传时指定自定义透传信息，上传完成时点播会将透传信息一并通知到 App 后台。具体的，在事件通知中有 2 个字段： `SourceType` 和 `SourceCopntext`。 `SourceType` 被腾讯云固定成 `ClientUpload`，表示上传来源为客户端上传； `SourceContext` 是用户自定义透传字段，App 后台可以在派发客户端上传签名时指定它的内容（对应 客户端上传签名 中的 `sourceContext` 参数），在上传事件回调时会透传回来。
- 点播支持在视频上传完成时自动发起视频处理，如果上传时指定了 视频处理任务流，则在事件通知内容中也会携带任务 ID（事件通知中的 `data.procedureTaskId` 字段）。

更多参见：

- 任务管理与事件通知；
- 事件通知-视频上传完成。

高级功能

上传时指定任务流

上文提到点播的视频处理主要基于 视频处理任务流。如果开发者需要在视频上传完成后自动发起任务流，可以在生成上传签名时通过 `procedure` 参数来实现。

示例1：以下设置表示在视频上传完成之后，按照控制台【全局设置】-【转码设置】中的配置进行转码，并在转码时打上【水印管理】中的默认水印：

```
procedure=QCVB_SimpleProcessFile(1,1)
```

示例2：以下设置表示在视频上传完成之后，使用转码模板 10，20 进行转码；转码过程使用水印模板 150 打水印；使用指定时间点截图模板 10 截取首帧设置封面；使用采样截图模板 10 进行采样截图：

```
procedure=QCVB_SimpleProcessFile({10,20},150,10,10)
```

`procedure` 参数亦可填写为其他点播内置任务流，或者开发者自定义的任务流（目前自定义功能未开放，需提工单）。

上传时附带封面

在视频上传过程中，点播允许携带封面上传。具体的就是在上传 SDK 接口中填写相关的封面路径，参见：

- Android 上传 SDK；
- iOS 上传 SDK；
- Web 上传 SDK。

单次有效签名

在视频上传过程中，App 后台派发的签名默认是在有效期内是可以多次使用的。如果 App 对视频上传安全性要求很高，可以指定签名单次有效。单次有效签名意味着这个签名有且只能被使用一次。值得注意的是这种签名方式虽然更加安全，但是需要 App 做额外的异常处理，比如上传出错时，不能简单地只是重试流程图步骤 2 中的“使用 SDK 上传视频”，还需要重新执行步骤 1，即 App 后台服务器需要重新派发签名。使用单次有效签名的方式是：在 App 后台派发签名时，指定参数 `oneTimeValid` 为 1 即可，参见：

- 客户端上传签名

断点续传

在视频上传过程中，点播支持断点续传，即当上传意外终止时，用户再次上传该文件，可以从中断处继续上传，减少重复上传时间。断点续传的有效时间是 1 天，即同一个视频上传被中断，那么 1 天内再次上传可以直接从断点处上传，超过 1 天则默认会重新上传完整视频。App 需要断点续传的功能时，只需要在上传时指定启用该功能即可，具体如下：

- Android 上传 SDK，上传时设置 `enableResume` 字段为 True 即可；
- iOS 上传 SDK，上传时设置 `enableResume` 字段为 True 即可；
- Web 上传 SDK，内置断点续传，无需用户操作。

暂停/恢复/取消上传

在视频上传过程中，点播 SDK 允许暂停、恢复、取消上传。参见：

- Android 上传 SDK；
- iOS 上传 SDK；
- Web 上传 SDK。

FAQ

如何在视频上传完成之后自动发起转码？

可以通过客户端上传签名中的 `procedure` 参数指定视频上传完成之后的处理方式，参见上传时指定任务流。

App 后台收到视频上传完成通知，如何识别是哪个客户上传的？

在客户端上传签名中增加 `sourceContext` 参数，通过该参数来携带用户身份信息。上传完成通知会将该参数原样携带给 App 后台。参见 事件通知。

客户端上传签名

最近更新时间：2018-07-10 09:31:13

简介

客户端在发起上传前，需要向 App 服务器请求上传签名（背景请参考 客户端上传指引）。如果 App 服务器允许客户端上传，则应按照本文介绍的签名规则为客户端生成一个上传签名。客户端执行上传操作时，必须携带该签名，让腾讯云点播验证客户端的上传是否被授权。

签名参数

参数名称	必选	类型	说明
secretId	是	String	云 API 密钥中的 SecretId，获取方式参考 客户端上传指引-获取云 API 密钥。
currentTimeStamp	是	Integer	当前 Unix 时间戳
expireTime	是	Integer	签名到期 Unix 时间戳。 `expireTime = currentTimeStamp + 签名有效时长` 签名有效时长最大取值为 7776000 ，即 90 天。
random	是	Integer	构造签名明文串的参数，无符号 32 位随机数
classId	否	Integer	视频文件分类，默认为 0
procedure	否	String	视频后续任务操作，详见 任务流综述
taskPriority	否	Integer	视频后续任务优先级（仅当指定了 procedure 时才有效），取值范围为 [-10, 10]，默认为0
taskNotifyMode	否	String	任务流状态变更通知模式（仅当指定了 procedure 时才有效）。 • Finish：只有当任务流全部执行完毕时，才发起一次事件通知； • Change：只要任务流中每个子任务的状态发生变化，都进行事件通知； • None：不接受该任务流回调。 默认为Finish。

参数名称	必选	类型	说明
sourceContext	否	String	客户端上传附带信息，在 事件通知-上传完成通知 中可以根据该字段识别一次上传行为，参见 客户端上传指引-事件通知
oneTimeValid	否	Integer	签名是否单次有效，参见 客户端上传指引-单次有效签名，默认为 0 表示不启用；1 表示签名单次有效，相关错误码详见下文的单次有效签名相关部分。

签名生成步骤

客户端上传签名的生成包括以下三步：

1. 获取 API 密钥；
2. 拼接明文串；
3. 将明文串转为最终签名。

> **注意：**
> 生成客户端签名代码较为复杂，点播提供了多种语言的签名生成示例代码，参见多语言签名生成示例。

第一步：获取 API 密钥

参考 客户端上传指引-获取云 API 密钥 获取或者创建一个 SecretId，并拿到其对应的 SecretKey。

第二步：拼接明文串

按照 URL QueryString 的格式要求生成签名明文串 Original[注 1]，格式如下：

```
secretId=[secretId]&currentTimeStamp=[currentTimeStamp]&expireTime=[expireTime]&random=[random]
```

第三步：将明文串转为最终签名

生成签名明文串 Original 后，用已获取的 SecretKey 对明文串进行 HMAC-SHA1加密，得到 SignatureTmp[注 2]：

```
SignatureTmp = HMAC-SHA1(secretKey, Original)
```

将密文串 SignatureTmp 放在明文串 Original 前面，拼接后进行 Base64 编码，得到最终的签名 Signature：

```
Signature = Base64(append(SignatureTmp, Original))
```

注意事项

注 1

签名明文串 Original 需要满足：

- 至少包含 secretId, currentTimeStamp, expireTime 和 random 四个必选参数，可包含任意多个选填参数；
- 参数值***必须***经过 UrlEncode，否则可能导致 QueryString 解析失败。

> ***建议***：直接操作字符串的方式生成 Original 很容易出错，大多数编程语言均提供了相关类库帮助开发者完成 QueryString 的拼接和编码。因此，建议开发者尽量使用标准的类库来构造 Original。

注 2

签名密文串 SignatureTmp 的输出结果是 20 字节的二进制串。

多语言签名生成示例

此处提供多种语言平台的客户端签名生成示例，App 可以根据开发语言的偏好参考对应的示例：

- PHP 示例
- Node.js 示例
- Java 示例
- C# 示例

签名生成和校验工具

此处提供了一组签名工具，帮助用户验证自己生成的签名是否正确：
点播客户端上传-签名生成工具：在页面上填写签名所需要的参数和密钥，即可生成一个合法的签名。
点播客户端上传-签名校验工具：对一个合法的签名进行解析，获得生成签名时所使用的参数。

单次有效签名相关

- 使用单次有效签名之后，签名服务器需要保证每次派发给用户的签名不相同（例如保证同一个时间点派发的签名的 random 不重复），否则会导致重复签名的错误。
- 由签名错误导致的上传失败，如果需要重试，需要获取新的签名，否则签名重复将导致重试失败（Android 和 Java SDK 签名错误引起的错误状态码是 1001）。

客户端上传签名示例

最近更新时间：2018-02-26 11:12:40

PHP 签名示例

```php
<?php
// 确定 App 的云 API 密钥
$secret_id = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
$secret_key = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";

// 确定签名的当前时间和失效时间
$current = time();
$expired = $current + 86400; // 签名有效期：1天

// 向参数列表填入参数
$arg_list = array(
"secretId" => $secret_id,
"currentTimeStamp" => $current,
"expireTime" => $expired,
"random" => rand());

// 计算签名
$orignal = http_build_query($arg_list);
$signature = base64_encode(hash_hmac('SHA1', $orignal, $secret_key, true).$orignal);

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

Java 签名示例

```java
import java.util.Random;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import sun.misc.BASE64Encoder;

class Signature {
private String secretId;
```

```
private String secretKey;
private long currentTime;
private int random;
private int signValidDuration;

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

public static byte[] byteMerger(byte[] byte1, byte[] byte2) {
byte[] byte3 = new byte[byte1.length + byte2.length];
System.arraycopy(byte1, 0, byte3, 0, byte1.length);
System.arraycopy(byte2, 0, byte3, byte1.length, byte2.length);
return byte3;
}

public String getUploadSignature() throws Exception {
String strSign = "";
String contextStr = "";

long endTime = (currentTime + signValidDuration);
contextStr += "secretId=" + java.net.URLEncoder.encode(secretId, "utf8");
contextStr += "&currentTimeStamp=" + currentTime;
contextStr += "&expireTime=" + endTime;
contextStr += "&random=" + random;

try {
Mac mac = Mac.getInstance(HMAC_ALGORITHM);
SecretKeySpec secretKey = new SecretKeySpec(this.secretKey.getBytes(CONTENT_CHARSET), mac.get
Algorithm());
mac.init(secretKey);

byte[] hash = mac.doFinal(contextStr.getBytes(CONTENT_CHARSET));
byte[] sigBuf = byteMerger(hash, contextStr.getBytes("utf8"));
strSign = new String(new BASE64Encoder().encode(sigBuf).getBytes());
strSign = strSign.replace(" ", "").replace("\n", "").replace("\r", "");
} catch (Exception e) {
throw e;
}
return strSign;
}

public void setSecretId(String secretId) {
this.secretId = secretId;
}
```

```
public void setSecretKey(String secretKey) {
this.secretKey = secretKey;
}

public void setCurrentTime(long currentTime) {
this.currentTime = currentTime;
}

public void setRandom(int random) {
this.random = random;
}

public void setSignValidDuration(int signValidDuration) {
this.signValidDuration = signValidDuration;
}
}

public class Test {
public static void main(String[] args) {
Signature sign = new Signature();
sign.setSecretId("个人API密钥中的Secret Id");
sign.setSecretKey("个人API密钥中的Secret Key");
sign.setCurrentTime(System.currentTimeMillis() / 1000);
sign.setRandom(new Random().nextInt(java.lang.Integer.MAX_VALUE));
sign.setSignValidDuration(3600 * 24 * 2);

try {
String signature = sign.getUploadSignature();
System.out.println("signature : " + signature);
} catch (Exception e) {
System.out.print("获取签名失败");
e.printStackTrace();
}
}
}
```

注意

- 需要导入第三方包 javax-crpyto.jar 和 sun.misc.BASE64Encoder.jar。
- 如果导入第三方包 sun.misc.BASE64Encoder.jar 出现 Access restriction 的错误，可以通过调整错误级别解决：

> Windows -> Preferences -> Java -> Compiler -> Errors/Warnings -> Deprecated **and** trstricted API -> Forbidden reference (access rules): -> change to warning

Node.js 签名示例

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

// 确定 app 的云 API 密钥
var secret_id = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
var secret_key = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";

// 确定签名的当前时间和失效时间
var current = parseInt((new Date()).getTime() / 1000)
var expired = current + 86400; // 签名有效期：1天

// 向参数列表填入参数
var arg_list = {
secretId : secret_id,
currentTimeStamp : current,
expireTime : expired,
random : Math.round(Math.random() * Math.pow(2, 32))
}

// 计算签名
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);
```

C# 签名示例

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

class Signature
{
```

```
public string m_strSecId;
public string m_strSecKey;
public int m_iRandom;
public long m_qwNowTime;
public int m_iSignValidDuration;
public static long GetIntTimeStamp()
{
TimeSpan ts = DateTime.UtcNow - new DateTime(1970, 1, 1);
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 GetUploadSignature()
{
string strContent = "";
strContent += ("secretId=" + Uri.EscapeDataString((m_strSecId)));
strContent += ("&currentTimeStamp=" + m_qwNowTime);
strContent += ("&expireTime=" + (m_qwNowTime + m_iSignValidDuration));
strContent += ("&random=" + m_iRandom);

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)
{
Signature sign = new Signature();
sign.m_strSecId = "个人 API 密钥中的Secret Id";
sign.m_strSecKey = "个人 AP I密钥中的Secret Key";
sign.m_qwNowTime = Signature.GetIntTimeStamp();
sign.m_iRandom = new Random().Next(0, 1000000);
sign.m_iSignValidDuration = 3600 * 24 * 2;

Console.WriteLine(sign.GetUploadSignature());
```

```
}
}
```

Web 端上传 SDK

最近更新时间：2018-05-15 15:59:11

简介

对于浏览器上传音视频的场景，腾讯云点播提供了 Web 上传 SDK 来实现。上传的流程可以参见客户端上传指引。

Demo 下载

http://video.qcloud.com/sdk/ugcuploader.html

简单视频上传

引入 SDK 到页面中

```
<script src="//imgcache.qq.com/open/qcloud/js/vod/sdk/ugcUploader.js"></script>
```

SDK 依赖 jQuery 来发送请求，所以请把 jQuery 也引入到页面中。

定义获取上传签名的函数

```
var getSignature = function(callback){
$.ajax({
url: 'yourinterface', //获取客户端上传签名的 URL
type: 'POST',
dataType: 'json',
success: function(result){//result 是派发签名服务器的回包
//假设回包为 { "code": 0, "signature": "xxxx" }
//将签名传入 callback，SDK 则能获取这个上传签名，用于后续的上传视频步骤。
callback(result.signature);
}
});
};
```

`url` 是您派发签名服务的 URL，参见客户端上传指引。
`signature` 计算规则可参考客户端上传签名。

上传视频

上传视频是通过调用 `qcVideo.ugcUploader.start` 来实现的。实例如下：

```
qcVideo.ugcUploader.start({
videoFile: videoFile,//视频，类型为 File
getSignature: getSignature,//前文中所述的获取上传签名的函数
error: function(result){//上传失败时的回调函数
//...
console.log('上传失败的原因：' + result.msg);
},
finish: function(result){//上传成功时的回调函数
console.log('上传结果的fileId：' + result.fileId);
console.log('上传结果的视频名称：' + result.videoName);
console.log('上传结果的视频地址：' + result.videoUrl);
}
});
```

高级功能

同时上传视频和封面

```
qcVideo.ugcUploader.start({
videoFile: videoFile,
coverFile: coverFile,//封面, 类型为 File
getSignature: getSignature,
error: function(result){
//...
console.log('上传失败的原因：' + result.msg);
},
finish: function(result){
console.log('上传结果的fileId：' + result.fileId);
console.log('上传结果的视频名称：' + result.videoName);
console.log('上传结果的视频地址：' + result.videoUrl);
console.log('上传结果的封面名称：' + result.coverName);
console.log('上传结果的封面地址：' + result.coverUrl);
}
});
```

获取上传进度

SDK 支持以回调的形式展示当前的上传进度，如下：

```
qcVideo.ugcUploader.start({
videoFile: videoFile,
coverFile: coverFile,//封面，类型为 File
getSignature: getSignature,
error: function(result){
console.log('上传失败的文件类型：' + result.type);
console.log('上传失败的原因：' + result.msg);
},
progress: function(result){
console.log('上传进度：' + result.curr);
},
finish: function(result){
console.log('上传结果的 fileId：' + result.fileId);
console.log('上传结果的视频名称：' + result.videoName);
console.log('上传结果的视频地址：' + result.videoUrl);
console.log('上传结果的封面名称：' + result.coverName);
console.log('上传结果的封面地址：' + result.coverUrl);
}
});
```

取消上传

SDK 支持取消正在上传的视频或封面，可以通过调用 `qcVideo.ugcUploader.cancel` 实现。

```
//用于实现取消上传的两个对象。
var uploadCos;//需要在 progress 回调中赋值。
var uploadTaskId;//需要在 progress 回调中赋值。

qcVideo.ugcUploader.start({
videoFile: videoFile,//这个是视频，类型为 File
getSignature: getSignature,//这个是第二步定义的函数
error: function(result){
console.log('上传失败的原因：' + result.msg);
},
progress: function(result){
console.log('上传进度的文件类型：' + result.type);
console.log('上传进度的文件名称：' + result.name);
console.log('上传进度：' + result.curr);
```

```
uploadCos = result.cos;
uploadTaskId = result.taskId;
},
finish: function(result){
console.log('上传结果的 fileId：' + result.fileId);
console.log('上传结果的视频名称：' + result.videoName);
console.log('上传结果的视频地址：' + result.videoUrl);
}
});

// ...

// 取消上传
qcVideo.ugcUploader.cancel({
cos: uploadCos,
taskId: uploadTaskId
});
```

断点续传

SDK 支持自动断点续传功能，无需做额外操作。当上传意外终止时，用户再次上传该文件，可以从中断处继续上传，减少重复上传时间。断点续传的有效时间是 1 天，即同一个视频上传被中断，那么 1 天内再次上传可以直接从断点处上传，超过 1 天则默认会重新上传完整视频。

接口描述

上传视频 `qcVideo.ugcUploader.start` 中的参数：

参数名称	必填	类型	参数描述
videoFile	是	File	要上传的视频文件
coverFile	否	File	要上传的封面文件

回调函数说明

函数名	含义	参数类型	补充说明
getSignature	获取签名回调	Function	回调参数 callback，实现时需要把获取到的签名传给 callback，用于后续上传视频步骤的鉴权

函数名	含义	参数类型	补充说明
error	上传失败回调	Object	回调的 result 对象中： type 字段表示上传失败的文件种类，'video' 表示视频, 'cover' 表示封面 msg 字段表示上传失败的原因。
progress	上传进度回调	Object	回调的 result 对象中： type 字段表示上传失败的文件种类； video 字段表示视频或者 cover 字段表示封面 name 字段表示上传中的文件名 curr 字段表示文件上传进度
finish	上传结果回调	Object	回调的 result 对象中： fileId 字段表示视频文件 ID videoName 字段表示视频名称 videoUrl 字段表示视频播放地址 coverName 字段表示封面名称 coverUrl 字段表示封面展示地址

其他

支持上传的文件类型

类型	支持格式
视频	WMV、WM、ASF、ASX RM、RMVB、RA、RAM MPG、MPEG、MPE、VOB、DAT MOV、3GP、MP4、MP4V、M4V、MKV、AVI、FLV、F4V
图片	JPG、JPEG、JPE PSD SVG、SVGZ TIFF、TIF BMP、GIF、PNG
音频	MP3、WAV

音频的上传方式跟视频类似，在 `qcVideo.ugcUploader.start` 中，将 `videoFile` 设置为音频文件，将 `allowAudio` 设置为 1。

常见问题

1. `File` 对象怎么获取？
   使用 `input` 标签， `type` 为 `file` 类型，即可拿到 `File` 对象

2. 上传的文件是否有大小限制?
   最大支持 60GB。

3. SDK 支持的浏览器版本有哪些？
   chrome、firefox等支持 `HTML 5` 的主流浏览器，IE 方面支持的最低版本是 IE10。

小程序端上传 SDK

最近更新时间：2018-09-20 10:11:36

在小程序端上传视频的 Web SDK。

Demo 体验

请打开微信，扫一扫下方二维码体验 Demo：

> 如果您需要 Demo 代码，可单击 Demo下载代码。

上传视频步骤

1. 引入SDK

```
const VodUploader = require('../../lib/vod-web-sdk-v5');
```

2. 定义获取上传签名的函数

```
getSignature: function(callback) {
wx.request({
url: 'https://xzb.qcloud.com/get_vod_sign',
method: 'POST',
data: {
Action: 'GetVodSignatureV2'
},
dataType: 'json',
success: function(res) {
if (res.data && res.data.data.signature) {
callback(res.data.data.signature);
} else {
return '获取签名失败';
}
}
});
}
```

url 是您派发签名服务的 URL，参见 客户端上传指引。
signature 计算规则可参考 客户端上传签名。

3. 上传视频

上传视频是通过调用 VodUploader.start 来实现的。实例如下：

```
VodUploader.start({
videoFile: file, //必填，把chooseVideo回调的参数(file)传进来
fileName: fileName, //选填，视频名称，强烈推荐填写(如果不填，则默认为“来自微信小程序”)
getSignature: getSignature, //必填，获取签名的函数
success: function(result) {
console.log('success');
console.log(result);
},
error: function(result) {
console.log('error');
console.log(result);
wx.showModal({
title: '上传失败',
content: JSON.stringify(result),
showCancel: false
});
},
```

```
progress: function(result) {
console.log('progress');
console.log(result);
},
finish: function(result) {
console.log('finish');
console.log(result);
wx.showModal({
title: '上传成功',
content: 'fileId:' + result.fileId + '\nvideoName:' + result.videoName,
showCancel: false
});
}
});
```

详细实例，可参考 Demo 的代码。

其他

1. 因为小程序没有获取真实文件名的 API，所以需要在上传视频之前，输入视频名称。如果不输入，SDK 会设置视频名称为“来自小程序”。
2. 只支持上传视频。
3. 不支持断点续传和分片上传。
4. 小程序的 chooseVideo API，目前支持上传文件为 25MB。
5. request 和 uploadFile 合法域名，请提工单 反馈给我们，.我们查询后给到您。

Android 上传 SDK

最近更新时间：2018-09-20 17:42:58

对于在 Android 平台上传视频的场景，腾讯云点播提供了 Android 上传 DEMO 来实现。上传的流程可以参见 客户端上传指引。

源代码下载

您可以在腾讯云官网更新 Android 上传 demo + 源代码。
下载完的 zip 包解压后可以看到 Demo 目录，上传相关源代码在 Demo/app/src/main/java/com/tencent/ugcupload/demo/videoupload 目录下。

集成上传库和源代码

1.拷贝上传源代码目录 Demo/app/src/main/java/com/tencent/ugcupload/demo/videoupload 到您的工程目录中，需要手动修改一下 package 名。
2.将 Demo/app/libs/upload 目录下的所有 jar 包集成到您的项目中，建议您保留 upload 目录结构，方便以后对库进行更新。

依赖库说明：

jar 文件	说明
cosxml-5.4.10.jar	腾讯云对象存储服务（COS）的文件上传包， 此组件用于视频上传（TXUGCPublish）功能
qcloud-foundation-1.5.1.jar	腾讯云对象存储服务（COS）的文件上传包， 此组件用于视频上传（TXUGCPublish）功能
okhttp-3.8.1.jar	开源 HTTP 组件
okio-1.13.0.jar	开源网络 I/O 组件
xstream-1.4.7.jar	开源序列化组件
bolts-tasks-1.4.0.jar	开源 多线程 组件

3.使用视频上传需要网络、存储等相关的一些访问权限，可在 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"/>


<receiver android:name=".videoupload.impl.TVCNetWorkStateReceiver">
<intent-filter>
//检测网络变化的acton
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</receiver>
```

简单视频上传

初始化一个上传对象

```
TXUGCPublish mVideoPublish = new TXUGCPublish(this.getApplicationContext(), "independence_android")
```

设置上传对象的回调

```
mVideoPublish.setListener(new TXUGCPublishTypeDef.ITXVideoPublishListener() {
@Override
public void onPublishProgress(long uploadBytes, long totalBytes) {
mProgress.setProgress((int) (100*uploadBytes/totalBytes));
}

@Override
public void onPublishComplete(TXUGCPublishTypeDef.TXPublishResult result) {
mResultMsg.setText(result.retCode + " Msg:" + (result.retCode == 0 ? result.videoURL : result.descMsg));
}
});
```

构造上传参数

```
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam();

param.signature = "xxx";
param.videoPath = "xxx";
```

signature 计算规则可参考 客户端上传签名。

调用上传

```
int publishCode = mVideoPublish.publishVideo(param);
```

高级功能

携带封面

在上传参数中带上封面路径即可。

```
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam();

param.signature = "xxx";
param.videoPath = "xxx";
param.coverPath = "xxx";
```

signature 计算规则可参考 客户端上传签名。

取消、恢复上传

取消上传，调用 `TXUGCPublish` 的 `canclePublish()` 。

```
mVideoPublish.canclePublish();
```

恢复上传，用相同的上传参数（视频路径和封面路径不变）再调用一次 `TXUGCPublish` 的 `publishVideo` 。

断点续传

在视频上传过程中，点播支持断点续传，即当上传意外终止时，用户再次上传该文件，可以从中断处继续上传，减少重复上传时间。断点续传的有效时间是 1 天，即同一个视频上传被中断，那么 1 天内再次上传可以直接从断点处

上传，超过 1 天则默认会重新上传完整视频。
上传参数中的 `enableResume` 为断点续传开关，默认是开启的。

预上传

经统计，在实际上传过程中很大部分的错误都是由于网络连接失败或者超时导致的，为优化此类问题增加了预上传优化逻辑。
预上传包含：httpdns解析、获取建议上传园区、探测最优上传园区。
建议您在app启动的时候调用 `TXUGCPublishOptCenter.getInstance().prepareUpload(signature)` ，预上传模块会把<域名, ip>映射表和最优上传园区缓存在本地，监听到网络切换的时候清空缓存并自动刷新。

注意：一定要在 AndroidManifest.xml 中注册网络监听模块

```
<receiver android:name=".videoupload.impl.TVCNetWorkStateReceiver">
<intent-filter>
//检测网络变化的acton
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</receiver>
```

> 参数 `signature` 计算规则可参考 客户端上传签名。

接口描述

初始化上传对象 `TXUGCPublish`

参数名称	参数描述	类型	必填
context	application 上下文	Context	是
customKey	用于区分不同的用户，建议使用app的账号id，方便后续定位问题	String	否

设置点播appId `TXUGCPublish.setAppId`

参数名称	参数描述	类型	必填
appId	点播appId	int	是

上传 TXUGCPublish.publishVideo

参数名称	参数描述	类型	必填
param	上传参数	TXUGCPublishTypeDef.TXPublishParam	是

上传参数 TXUGCPublishTypeDef.TXPublishParam

参数名称	参数描述	类型	必填
signature	客户端上传签名	String	是
videoPath	本地视频文件路径	String	是
coverPath	本地封面文件路径，默认不带封面文件	String	否
enableResume	是否启动断点续传，默认开启	boolean	否
enableHttps	是否启动 HTTPS，默认关闭	boolean	否
fileName	上传到点播系统的视频文件名称，不填默认用本地文件名	String	否

设置上传回调 TXUGCPublish.setListener

参数名称	参数描述	类型	必填
listener	上传进度和结果回调监听	TXUGCPublishTypeDef.ITXVideoPublishListener	是

进度回调 TXUGCPublishTypeDef.ITXVideoPublishListener.onPublishProgress

变量名称	变量描述	类型
uploadBytes	已经上传的字节数	long
totalBytes	总字节数	long

结果回调 TXUGCPublishTypeDef.ITXVideoPublishListener.onPublishComplete

变量名称	变量描述	类型
result	上传结果	TXUGCPublishTypeDef.TXPublishResult

上传结果 TXUGCPublishTypeDef.TXPublishResult

成员变量名称	变量说明	类型
retCode	结果码	int
descMsg	上传失败的错误描述	String
videoId	点播视频文件Id	String
videoURL	视频存储地址	String
coverURL	封面存储地址	String

预上传 `TXUGCPublishOptCenter.prepareUpload`

参数名称	参数描述	类型	必填
signature	客户端上传签名	String	是

错误码

SDK 通过 `TXUGCPublishTypeDef.TXVideoPublishListener` 接口来监听视频上传相关的状态。因此，可以利用 `TXUGCPublishTypeDef.TXPublishResult` 中的 `retCode` 来确认视频上传的情况。

状态码	在 TVCConstants 中所对应的常量	含义
0	NO_ERROR	上传成功
1001	ERR_UGC_REQUEST_FAILED	请求上传失败，通常是客户端签名过期或者非法，需要 App 重新申请签名
1002	ERR_UGC_PARSE_FAILED	请求信息解析失败
1003	ERR_UPLOAD_VIDEO_FAILED	上传视频失败
1004	ERR_UPLOAD_COVER_FAILED	上传封面失败
1005	ERR_UGC_FINISH_REQUEST_FAILED	结束上传请求失败
1006	ERR_UGC_FINISH_RESPONSE_FAILED	结束上传响应错误
1007	ERR_CLIENT_BUSY	客户端正忙(对象无法处理更多请求)
1008	ERR_FILE_NOEXIT	上传文件不存在
1009	ERR_UGC_PUBLISHING	视频正在上传中

状态码	在 TVCConstants 中所对应的常量	含义
1010	ERR_UGC_INVALID_PARAM	上传参数为空
1012	ERR_UGC_INVALID_SIGNATURE	视频上传 signature 为空
1013	ERR_UGC_INVALID_VIDOPATH	视频文件的路径为空
1014	ERR_UGC_INVALID_VIDEO_FILE	当前路径下视频文件不存在
1015	ERR_UGC_FILE_NAME	视频上传文件名太长（超过 40）或含有特殊字符
1016	ERR_UGC_INVALID_COVER_PATH	视频文件封面路径不对，文件不存在
1017	ERR_USER_CANCEL	用户取消上传
1018	ERR_UPLOAD_VOD	小于5m的文件直接上传到点播失败

iOS上传SDK

最近更新时间：2018-06-07 18:04:57

对于在 iOS 平台上传视频的场景，腾讯云点播提供了 iOS 上传 DEMO 来实现。上传的流程可以参见客户端上传指引。

源代码下载

您可以在腾讯云官网更新 iOS 上传 demo + 源代码。
下载完的 zip 包解压后可以看到 TXUGCUploadDemo 目录，发布相关源代码在 TXUGCUploadDemo/upload 目录下。

集成上传库和源代码

1. 拷贝上传源代码目录 TXUGCUploadDemo/upload 到您的工程中。

2. 导入动态库QCloudCore.framework、QCloudCOSXML.framework（TXUGCUploadDemo目录下）到您的工程中。并添加以下依赖库：

   1、**CoreTelephony**
   2、**Foundation**
   3、**SystemConfiguration**
   4、**libstdc**++.tbd

3. 在 Build Settings 中设置 Other Linker Flags，加入参数***-ObjC***

简单视频上传

初始化一个上传对象

```
TXUGCPublish *_videoPublish = [[TXUGCPublish alloc] initWithUserID:@"carol_ios"];
```

设置上传对象的回调

```
_videoPublish.delegate = self;
#pragma mark - TXVideoPublishListener
-(void) onPublishProgress:(NSInteger)uploadBytes totalBytes: (NSInteger)totalBytes
{
self.progressView.progress = (float)uploadBytes/totalBytes;
NSLog(@"onPublishProgress [%lld/%lld]", uploadBytes, totalBytes);
}

-(void) onPublishComplete:(TXPublishResult*)result
{
NSString *string = [NSString stringWithFormat:@"上传完成，错误码[%d]，信息[%@]", result.retCode, result.retCode == 0? result.videoURL: result.descMsg];
[self showErrorMessage:string];
NSLog(@"onPublishComplete [%d/%@]", result.retCode, result.retCode == 0? result.videoURL: result.descMsg);
}
```

构造上传参数

```
TXPublishParam *videoPublishParams = [[TXPublishParam alloc] init];

videoPublishParams.signature = @"xxx";
videoPublishParams.videoPath = self.uploadTempFilePath;
```

> signature 计算规则可参考客户端上传签名。

调用上传

```
[_videoPublish publishVideo:videoPublishParams];
```

高级功能

携带封面

在上传参数中带上封面图片即可。

```
TXPublishParam *videoPublishParams = [[TXPublishParam alloc] init];
videoPublishParams.signature = @"xxx";
```

```
videoPublishParams.coverPath = @"xxx";
videoPublishParams.videoPath = self.uploadTempFilePath;
```

取消、恢复上传

取消上传，调用 TXUGCPublish 的 anclePublish() 。

```
[_videoPublish canclePublish];
```

恢复上传，用相同的上传参数（视频路径和封面路径不变）再调用一次 TXUGCPublish 的 publishVideo 。

断点续传

在视频上传过程中，点播支持断点续传，即当上传意外终止时，用户再次上传该文件，可以从中断处继续上传，减少重复上传时间。断点续传的有效时间是 1 天，即同一个视频上传被中断，那么 1 天内再次上传可以直接从断点处上传，超过 1 天则默认会重新上传完整视频。
上传参数中的 enableResume 为断点续传开关，默认是开启的。

接口描述

初始化上传对象 TXUGCPublish::initWithUserID

参数名称	参数描述	类型	必填
userID	用户 userID，用于区分不同的用户	NSString*	否

上传 TXUGCPublish.publishVideo

参数名称	参数描述	类型	必填
param	发布参数	TXPublishParam*	是

上传参数 TXPublishParam

参数名称	参数描述	类型	必填
signature	客户端上传签名	NSString*	是
videoPath	本地视频文件路径	NSString*	是
coverPath	封面图片本地路径，可不设置。	NSString*	否
fileName	上传到点播系统的视频文件名称，不填默认用本地文件名	NSString*	否

参数名称	参数描述	类型	必填
enableResume	是否启动断点续传，默认开启	BOOL	否
enableHttps	是否启动 HTTPS，默认关闭	BOOL	否

设置上传回调 `TXUGCPublish.delegate`

成员变量名称	变量描述	类型	必填
delegate	上传进度和结果回调监听	TXVideoPublishListener	是

进度回调 `TXVideoPublishListener.onPublishProgress`

变量名称	变量描述	类型
uploadBytes	已经上传的字节数	NSInteger
totalBytes	总字节数	NSInteger

结果回调 `TXVideoPublishListener.onPublishComplete`

变量名称	变量描述	类型
result	上传结果	TXPublishResult*

上传结果 `TXPublishResult`

成员变量名称	变量说明	类型
retCode	结果码	int
descMsg	上传失败的错误描述	NSString*
videoId	点播视频文件Id	NSString*
videoURL	视频存储地址	NSString*
coverURL	封面存储地址	NSString*

错误码

SDK 通过 `TXVideoPublishListener` 接口来监听视频上传相关的状态。因此，可以利用 `TXPublishResult` 中的 `retCode` 来确认视频发布的情况。

状态码	在 TVCCommon 中所对应的常量	含义
0	TVC_OK	上传成功
1001	TVC_ERR_UGC_REQUEST_FAILED	请求上传失败，通常是客户端签名过期或者非法，需要 App 重新申请签名
1002	TVC_ERR_UGC_PARSE_FAILED	请求信息解析失败
1003	TVC_ERR_VIDEO_UPLOAD_FAILED	上传视频失败
1004	TVC_ERR_COVER_UPLOAD_FAILED	上传封面失败
1005	TVC_ERR_UGC_FINISH_REQ_FAILED	结束上传请求失败
1006	TVC_ERR_UGC_FINISH_RSP_FAILED	结束上传响应错误
1008	TVC_ERR_FILE_NOT_EXIST	上传文件不存在
1012	TVC_ERR_INVALID_SIGNATURE	视频上传 signature 为空
1013	TVC_ERR_INVALID_VIDEOPATH	视频文件的路径为空
1017	TVC_ERR_USER_CANCLE	用户调用取消上传