短视频 SDK

单功能集成(iOS)

产品文档

文档目录

单功能集成(iOS)
SDK 集成(XCode)

最近更新时间：2022-04-07 18:19:14

支持平台

SDK 支持 iOS 8.0 以上系统。

开发环境

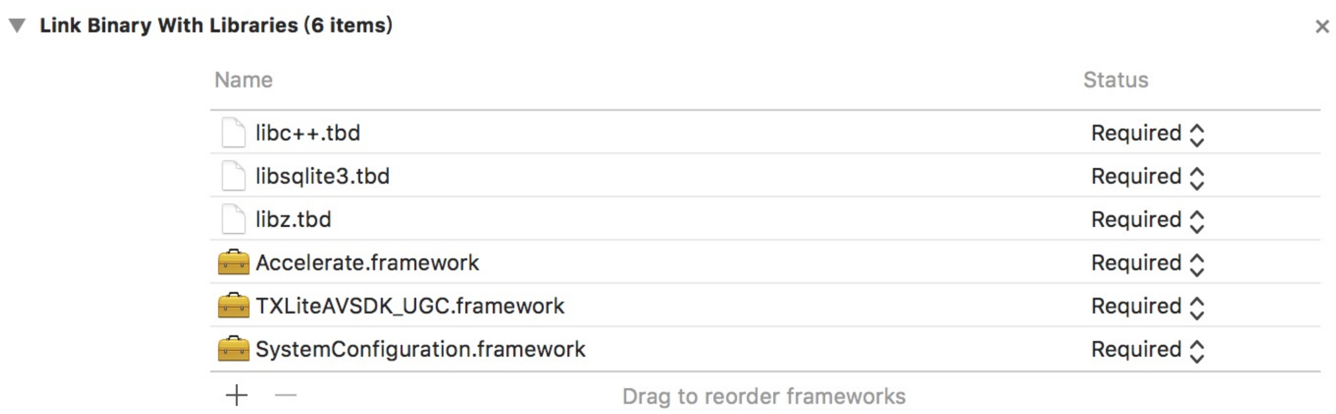
- Xcode 9 或更高版本。
- OS X 10.10 或更高版本。

设置步骤

步骤1：链接 SDK 及系统库

1. 将下载的 SDK 资源包解压，并将 SDK 文件夹中 TXLiteAVSDK_ 开头的 framework（如 TXLiteAVSDK_UGC.framework）复制到工程所在文件夹，并拖动到工程当中。
2. 选中工程的 Target，添加以下系统库：
    - Accelerate.framework
    - SystemConfiguration.framework
    - libc++.tbd
    - libsqlite3.tbd

    添加完毕后，工程库依赖如下图所示：



3. 选中工程的 Target，在 Build Settings 中搜索 bitcode，将 Enable Bitcode 设置为 NO。

步骤2：配置 App 权限

应用会需要相册及相册的访问权限，需要在 Info.plist 中添加对应项，可以通过在 Info.plist 中右键选 Open as / Source Code 粘贴并修改以下内容进行配置。

```
<key>NSAppleMusicUsageDescription</key>
<string>视频云工具包需要访问您的媒体库权限以获取音乐，不允许则无法添加音乐</string>
<key>NSCameraUsageDescription</key>
<string>视频云工具包需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>视频云工具包需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
<key>NSPhotoLibraryAddUsageDescription</key>
<string>视频云工具包需要访问您的相册权限，开启后才能保存编辑的文件</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>视频云工具包需要访问您的相册权限，开启后才能编辑视频文件</string>
```

步骤3：SDK License 设置与基本信息获取

通过 License 申请 的指引申请 License 后，从 控制台 复制 key 和 url，见下图。

短视频License参数 短视频SDK接入指引

App Name	test12
Package Name	test12
Bundle Id	test12
key	[illegible]
licenseUrl	http://license.vod2.myqcloud.com/license/v1[illegible]/TXUgcSDK.licence
公司名称	222
申请人姓名	222
手机号码	222
开始日期	2018-06-11
结束日期	2018-07-11

下载License

在您的应用中使用短视频功能之前，建议在 - [AppDelegate application:didFinishLaunchingWithOptions:] 中进行如下设置：

```
@import TXLiteAVSDK_UGC;
@implementation AppDelegate
- (BOOL)application:(UIApplication*)applicationdidFinishLaunchingWithOptions:(NSDictinoary*)options {
NSString * const licenceURL = @"<获取到的licnseUrl>";
NSString * const licenceKey = @"<获取到的key>";
[TXUGCBase setLicenceURL:licenceURL key:licenceKey];
NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

说明：

- **对于使用4.7版本 License 的用户，如果您升级了 SDK 到4.9版本，您可以登录控制台，单击下图的【切换到新版License】生成对应的 key 和 url，切换后的 License 必须使用4.9及更高的版本，切换后按照上述操作集成即可。**

- **企业版请参考 动效变脸。**

步骤4：Log 配置

在 TXLiveBase 中可以设置 log 是否在控制台打印以及 log 的级别，相关接口如下：

- **setConsoleEnabled**
  设置是否在 xcode 的控制台打印 SDK 的相关输出。
- **setLogLevel**
  设置是否允许 SDK 打印本地 log，SDK 默认会将 log 写到当前 App 的 **Documents/logs** 文件夹下。
  如果您需要我们的技术支持，建议将此开关打开，在重现问题后提供 log 文件，非常感谢您的支持。
- **Log 文件的查看**
  小直播 SDK 为了减少 log 的存储体积，对本地存储的 log 文件做了加密，并且限制了 log 数量的大小，所以要查看 log 的文本内容，需要使用 log 解压缩工具。

```
[TXLiveBase setConsoleEnabled:YES];
[TXLiveBase setLogLevel:LOGLEVEL_DEBUG];
```

步骤5：编译运行

如果前面各步骤都操作正确的话，HelloSDK 工程就可以顺利编译通过。在 Debug 模式下运行 App，Xcode 的 Console 窗格会打印出 SDK 的版本信息：

```
2017-09-26 16:16:15.767 HelloSDK[17929:7488566] SDK Version = 5.2.5541
```

快速接入功能模块

为了方便您快速集成 SDK 各项功能，我们提供了 UGCKit。UGCKit 是在短视频 SDK 基础上构建的一套 UI 组件库。

您可以通过 GitHub 或 资源下载 中提供的 SDK 压缩包获取 UGCKit。UGCKit 位于压缩包 Demo/TXLiteAVDemo/UGC/UGCKit 目录下。

UGCKit 开发环境要求

- Xcode 10 及以上。
- iOS 9.0 及以上。

步骤1：集成 UGCKit

1. **项目配置**：
   i. 项目中使用 cocoapods，根据实际情况选择其中一种操作：
      - 在项目根目录，执行 pod init && pod install，可得到 Podfile 文件。
      - 把 **BeautySettingKit** 和 **UGCKit** 文件夹拷贝到项目根目录下（Podfile 同级目录）。
   ii. 打开 Podfile 文件，增加：

```
pod 'BeautySettingKit', :path => 'BeautySettingKit/BeautySettingKit.podspec'
pod 'UGCKit', :path => 'UGCKit/UGCKit.podspec', :subspecs => ["UGC"] #subspecs 根据SDK来选择
```

iii. 执行 **pod install**，并打开 项目名.xcworkspace，可以看到在 Pods/Development Pods 目录下已有 UGCKit BeautySettingKit。

2. **导入企业版资源（仅用于企业版）：**
将企业版 SDK ZIP 包中 EnterprisePITU（在 App/AppCommon 目录下）文件夹拖动到工程中，选择 **Create groups** 并勾选您的 Target，单击 **Finish**。

步骤2：使用 UGCKit

1. **录制**
UGCKitRecordViewController提供了完整的录制功能，您只需实例化这个控制器后展现在界面中即可。

```
UGCKitRecordViewController *recordViewController = [[UGCKitRecordViewController alloc]
initWithConfig:nil theme:nil];
[self.navigationController pushViewController:recordViewController]
录制后的结果将通过 completion block 回调，示例如下：
recordViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// 录制出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出录制界面
[self.navigationController popViewControllerAnimated:YES];
} else {
// 录制成功，用结果进行下一步处理
[self processRecordedVideo:result.media];
}
}
};
```

2. **编辑**
UGCKitEditViewController提供了完整的图片转场和视频编辑功能，实例化时需要传入待编辑的媒体对象，以处理录制结果为例，示例如下：

```
- (void)processRecordedVideo:(UGCKitMedia *)media {
// 实例化编辑控制器
UGCKitEditViewController *editViewController = [[UKEditViewController alloc] initWithMedi
a:media conifg:nil theme:nil];
// 展示编辑控制器
[self.navigationController pushViewController:editViewController animated:YES];
编辑后的结果将通过 completion block 回调，示例如下：
editViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// 出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出编辑界面
[self.navigationController popViewControllerAnimated:YES];
} else {
// 编辑保存成功，用结果进行下一步处理
[self processEditedVideo:result.path];
}
}
}
```

3. **从相册中选择视频或图片**

UGCKitMediaPickerViewController用来处理媒体的选择与合并，当选择多个视频时，将会返回拼接后的视频。示例如下：

```
// 初始化配置
UGCKitMediaPickerConfig *config = [[UGCKitMediaPickerConfig alloc] init];
config.mediaType = UGCKitMediaTypeVideo;//选择视频
config.maxItemCount = 5; //最多选5个
// 实例化媒体选择器
UGCKitMediaPickerViewController *mediaPickerViewController = [[UGCKitMediaPickerView
Controller alloc] initWithConfig:config theme:nil];
// 展示媒体选择器
[self presentViewController:mediaPickerViewController animated:YES completion:nil];
选择的结果将通过 completion block 回调，示例如下：
mediaPickerViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
```

```
// 出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出选择器界面
[self dismissViewControllerAnimated:YES completion:nil];
} else {
// 编辑保存成功，用结果进行下一步处理
[self processEditedVideo:result.media];
}
}
}
```

4. **裁剪**
   UGCKitCutViewController提供视频的裁剪功能，与编辑接口相同，在实例化时传入媒体对象，在 completion 中处理剪辑结果即可。示例如下：

```
UGCKitMedia *media = [UGCKitMedia mediaWithVideoPath:@"<#视频路径#>"];
UGCKitCutViewController *cutViewController = [[UGCKitCutViewController alloc] initWithMedia:media theme:nil];
cutViewController.completion = ^(UGCKitResult *result) {
if (!result.cancelled && !result.error) {
[self editVideo:result.media];
} else {
[self.navigationController popViewControllerAnimated:YES];
}
}
[self.navigationController pushViewController: cutViewController]
```

详细介绍

以下为 SDK 各模块的详细说明：

- 视频录制
- 视频编辑
- 视频拼接
- 视频上传
- 视频播放
- 动效变脸（企业版）

拍照和录制
拍照和录制(iOS)

最近更新时间：2021-01-27 16:07:53

功能概览

视频录制包括视频变速录制、美颜、滤镜、声音特效、背景音乐设置等功能。

使用类介绍

腾讯云 UGC SDK 提供了相关接口用来实现短视频的录制，其详细定义如下：

接口文件	功能
TXUGCRecord.h	小视频录制功能
TXUGCRecordListener.h	小视频录制回调
TXUGCRecordEventDef.h	小视频录制事件回调
TXUGCRecordTypeDef.h	基本参数定义
TXUGCPartsManager.h	视频片段管理类，用于视频的多段录制，回删等

使用说明

视频录制的基本使用流程如下：

1. 配置录制参数。
2. 启动画面预览。
3. 设置录制效果。
4. 完成录制。

示例

```
@interface VideoRecordViewController <txugcrecordlistener> {
UIView *_videoRecordView;
}
```

```
@implementation VideoRecordViewController
- (void)viewDidLoad {
[super viewDidLoad];

// 创建一个视图用于显示相机预览图片
_videoRecordView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view addSubview:_videoRecordView];

// 1. 配置录制参数
TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
param.videoQuality = VIDEO_QUALITY_MEDIUM;

// 2. 启动预览, 设置参数与在哪个View上进行预览
[[TXUGCRecord shareInstance] startCameraSimple:param preview:_videoRecordView];

// 3. 设置录制效果，这里以添加水印为例
UIImage *watermarke = [UIImage imageNamed:@"watermarke"];
[[TXUGCRecord shareInstance] setWaterMark:watermarke normalizationFrame:CGRectMake(
0.01, 0.01, 0.1, 0)];
}

// 4. 开始录制
- (IBAction)onStartRecord:(id)sender {
[TXUGCRecord shareInstance].recordDelegate = self;
int result = [[TXUGCRecord shareInstance] startRecord];
if(0 != result) {
if(-3 == result) [self alert:@"启动录制失败" msg:@"请检查摄像头权限是否打开"];
else if(-4 == result) [self alert:@"启动录制失败" msg:@"请检查麦克风权限是否打开"];
else if(-5 == result) [self alert:@"启动录制失败" msg:@"licence 验证失败"];
} else {
// 启动成功
}
}

// 结束录制
- (IBAction)onStopRecord:(id)sender {
[[TXUGCRecord shareInstance] stopRecord];
}

// 录制完成回调
```

```
-(void) onRecordComplete:(TXUGCRecordResult*)result
{
if (result.retCode == UGC_RECORD_RESULT_OK) {
// 录制成功， 视频文件在result.videoPath中
} else {
// 错误处理，错误码定义请参见 TXUGCRecordTypeDef.h 中 TXUGCRecordResultCode 的定义
}
}

-(void)alert:(NSString *)title msg:(NSString *)msg
{
UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title message:msg delegate:self cancel
ButtonTitle:@"确定" otherButtonTitles:nil, nil];
[alert show];
}
@end
```

画面预览

TXUGCRecord（位于 TXUGCRecord.h）负责小视频的录制功能，我们的第一个工作是先把预览功能实现。startCameraSimplePreview 函数用于启动预览。由于启动预览要打开摄像头和麦克风，所以这里可能会有权限申请的提示窗。

1. 启动预览

```
TXUGCRecord *record = [TXUGCRecord sharedInstance];
record.recordDelegate = self; //设置录制回调，回调方法见 TXUGCRecordListener

//配置相机及启动预览
TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p
param.frontCamera = YES; //使用前置摄像头
param.minDuration = 5; //视频录制的最小时长5s
param.maxDuration = 60; //视频录制的最大时长60s
param.enableBFrame = YES; // 开启B帧，相同码率下能获得更好的画面质量

//在self.previewView中显示照相机预览画面
```

```
[recorder startCameraSimple:param preview:self.previewView];

//结束画面预览
[[TXUGCRecord shareInstance] stopCameraPreview];
```

2. 调整预览参数

如果在相机启动后，可以通过以下方法修改：

```
// 切换视频录制分辨率到540p
[recorder setVideoResolution: VIDEO_RESOLUTION_540_960];

// 切换视频录制码率到6500Kbps
[recorder setVideoBitrate: 6500];

// 设置焦距为3, 当为1的时候为最远视角（正常镜头），当为5的时候为最近视角（放大镜头）
[recorder setZoom: 3];

// 切换到后置摄像头 YES 切换到前置摄像头 NO 切换到后置摄像头
[recorder switchCamera: NO];

// 打开闪光灯 YES为打开， NO为关闭.
[recorder toggleTorch: YES];

// 设置自定义图像处理回调
recorder.videoProcessDelegate = delegate;
```

录制过程控制

录制的开始、暂停与恢复

```
// 开始录制
[recorder startRecord];

// 开始录制，可以指定输出视频文件地址和封面地址
[recorder startRecord:videoFilePath coverPath:coverPath];

// 开始录制,可以指定输出视频文件地址、视频分片存储地址和封面地址
[recorder startRecord:videoFilePath videoPartsFolder:videoPartFolder coverPath:coverPath];
```

```
// 暂停录制
[recorder pauseRecord];

// 继续录制
[recorder resumeRecord];

// 结束录制
[recorder stopRecord];
```

录制的过程和结果是通过 TXUGCRecordListener（位于 TXUGCRecordListener.h 中定义）协议进行回调：

- onRecordProgress 用于反馈录制的进度，参数 millisecond 表示录制时长，单位毫秒。

```
@optional
(void)onRecordProgress:(NSInteger)milliSecond;
```

- onRecordComplete 反馈录制的结果，TXRecordResult 的 retCode 和 descMsg 字段分别表示错误码和错误描述信息，videoPath 表示录制完成的小视频文件路径，coverImage 为自动截取的小视频第一帧画面，便于在视频发布阶段使用。

```
@optional
(void)onRecordComplete:(TXUGCRecordResult*)result;
```

- onRecordEvent 录制事件回调预留的接口，暂未使用。

```
@optional
(void)onRecordEvent:(NSDictionary*)evt;
```

录制属性设置

1. 画面设置

```
// 设置横竖屏录制
[recorder setHomeOrientation:VIDOE_HOME_ORIENTATION_RIGHT];
// 设置视频预览方向
// rotation : 取值为 0 , 90, 180, 270（其他值无效） 表示视频预览向右旋转的角度
```



```
// 注意：需要在startRecord 之前设置，录制过程中设置无效
[recorder setRenderRotation:rotation];
// 设置录制的宽高比
// VIDEO_ASPECT_RATIO_9_16 宽高比为9:16
// VIDEO_ASPECT_RATIO_3_4 宽高比为3:4
// VIDEO_ASPECT_RATIO_1_1 宽高比为1:1
// 注意：需要在startRecord 之前设置，录制过程中设置无效
[recorder setAspectRatio:VIDEO_ASPECT_RATIO_9_16];
```

2. 速度设置

```
// 设置视频录制速率
// VIDEO_RECORD_SPEED_SLOWEST, 极慢速
// VIDEO_RECORD_SPEED_SLOW, 慢速
// VIDEO_RECORD_SPEED_NOMAL, 正常速
// VIDEO_RECORD_SPEED_FAST, 快速
// VIDEO_RECORD_SPEED_FASTEST, 极快速
[recorder setRecordSpeed:VIDEO_RECORD_SPEED_NOMAL];
```

3. 声音设置

```
// 设置麦克风的音量大小，播放背景音混音时使用，用来控制麦克风音量大小
// 音量大小,1为正常音量,建议值为0-2,如果需要调大音量可以设置更大的值.
[recorder setMicVolume:volume];

// 设置录制是否静音 参数 isMute 代表是否静音，默认不静音
[recorder setMute:isMute];
```

拍照

```
// 截图/拍照，startCameraSimplePreview 或者 startCameraCustomPreview 之后调用有效
[recorder snapshot:^(UIImage *image) {
// image 为截图结果
}];
```

设置效果

在视频录制的过程中，您可以给录制视频的画面设置各种特效。

1. 水印效果

```
// 设置全局水印
// normalizationFrame : 水印相对于视频图像的归一化值，sdk 内部会根据水印宽高比自动计算 height
// 例如视频图像大小为（540，960） frame 设置为（0.1，0.1，0.1，0）
// 水印的实际像素坐标为
// (540*0.1, 960*0.1, 540*0.1, 540*0.1*waterMarkImage.size.height / waterMarkImage.size.wid
th）
[recorder setWaterMark:waterMarkImage normalizationFrame:frame)
```

2. 滤镜效果

```
//设置风格滤镜
// 设置颜色滤镜：浪漫、清新、唯美、粉嫩、怀旧...
// filterImage : 指定滤镜用的颜色查找表。注意：一定要用 png 格式
// demo 用到的滤镜查找表图片位于 FilterResource.bundle 中
[recorder setFilter:filterImage];

// 用于设置滤镜的效果程度，从0到1，越大滤镜效果越明显，默认取值0.5
[recorder setSpecialRatio:ratio];

// 设置组合滤镜特效
// mLeftBitmap 左侧滤镜
// leftIntensity 左侧滤镜强度
// mRightBitmap 右侧滤镜
// rightIntensity 右侧滤镜强度
// leftRadio 左侧图片占的比例大小
// 可以此接口实现滑动切换滤镜的效果，详见 demo。
[recorder setFilter:leftFilterImgage leftIntensity:leftIntensity rightFilter:rightFilterImgage right
Intensity:rightIntensity leftRatio:leftRatio];
```

3. 美颜效果

```
// 设置美颜风格、级别、美白及红润的级别
// beautyStyle的定义如下:
```

```
// typedef NS_ENUM(NSInteger, TXVideoBeautyStyle) {
// VIDOE_BEAUTY_STYLE_SMOOTH = 0, // 光滑
// VIDOE_BEAUTY_STYLE_NATURE = 1, // 自然
// VIDOE_BEAUTY_STYLE_PITU = 2, // pitu 美颜, 需要购买企业版
// };
// 级别的范围为0-9 0为关闭， 1-9值越大，效果越明显
[recorder setBeautyStyle:beautyStyle beautyLevel:beautyLevel whitenessLevel:whitenessLevel
ruddinessLevel:ruddinessLevel];
```

高级功能

[多段录制]
[录制草稿箱]
[添加背景音乐]
[变声和混响]
[定制视频数据]

多段录制(iOS)

最近更新时间：2021-01-27 15:49:17

视频多段录制基本使用流程如下：

1. 启动画面预览。
2. 开始录制。
3. 开始播放 BGM。
4. 暂停录制。
5. 暂停播放 BGM。
6. 继续播放 BGM。
7. 继续录制。
8. 停止录制。
9. 停止播放 BGM。

```
//开启画面预览
recorder = [TXUGCRecord shareInstance];
[recorder startCameraCustom:param preview:preview];

// 开始录制
[recorder startRecord];

//设置BGM
[recorder setBGM:BGMPath];

//开始播放BGM
[recorder playBGMFromTime:beginTime toTime:_BGMDuration withBeginNotify:^(NSInteger
errCode) {
//开始播放
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
//播放进度
} andCompleteNotify:^(NSInteger errCode) {
//播放结束
}];

// 调用 pauseRecord 后会生成一段视频，视频可以在 TXUGCPartsManager 里面获取管理
[recorder pauseRecord];
```

```
//暂停播放BGM
[recorder pauseBGM];

//继续播放BGM
[recorder resumeBGM];

// 继续录制视频
[recorder resumeRecord];

// 停止录制，将多段视频合成为一个视频输出
[recorder stopRecord];

// 停止播放BGM
[recorder stopBGM];

//获取视频分片管理对象
TXUGCPartsManager *partsManager = recorder.partsManager;

//获取当前所有视频片段的总时长
[partsManager getDuration];

//获取所有视频片段路径
[partsManager getVideoPathList];

// 删除最后一段视频
[partsManager deleteLastPart];

// 删除指定片段视频
[partsManager deletePart:1];

// 删除所有片段视频
[partsManager deleteAllParts];

//您可以添加当前录制视频之外的视频
[partsManager insertPart:videoPath atIndex:0];

//合成所有片段视频
[partsManager joinAllParts: videoOutputPath complete:complete];
```

录制草稿箱(iOS)

最近更新时间：2021-07-26 16:49:12

草稿箱实现步骤：

第一次录制

1. 开始录制。
2. 暂停/结束第一次录制。
3. 缓存视频分片到本地（草稿箱）。

第二次录制

1. 预加载本地缓存视频分片。
2. 继续录制。
3. 结束录制。

```
//获取第一次视频录制对象
record = [TXUGCRecord shareInstance];

//开始录制
[record startRecord];

//暂停录制，缓存视频分片
[record pauseRecord:^{
NSArray *videoPathList = record.partsManager.getVideoPathList;
//videoPathList 写本地
}];

//获取第二次视频录制对象
record2 = [TXUGCRecord shareInstance];

//预加载本地缓存分片
[record2.partsManager insertPart:videoPath atIndex:0];

//开始录制
[record2 startRecord];
```

```
//结束录制,SDK会合成缓存视频片段和当前录制视频片段
[record2 stopRecord];
```

> **注意：**
> 具体实现方法请参考 小视频源码 中的 UGCKitRecordViewController 类。

添加背景音乐(iOS)

最近更新时间：2021-01-27 15:52:49

录制添加 BGM

```
//获取 recorder 对象
TXUGCRecord *recorder = [TXUGCRecord shareInstance];

// 设置 BGM 文件路径
[recorder setBGMAsset:path];

// 设置 BGM，从系统媒体库 loading 出来的音乐，可以直接传入对应的 AVAsset
[recorder setBGMAsset:asset];

// 播放 BGM
[recorder playBGMFromTime:beginTime
toTime:endTime
withBeginNotify:^(NSInteger errCode) {
// 播放开始回调, errCode 0为成功其它为失败
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
// progressMS: 已经播放的时长， durationMS: 总时长
} andCompleteNotify:^(NSInteger errCode) {
// 播放结束回调, errCode 0为成功其它为失败
}];

// 停止播放 BGM
[recorder stopBGM];

// 暂停播放 BGM
[recorder pauseBGM];

// 继续播放 BGM
[recorder resumeBGM];

// 设置麦克风的音量大小，播放背景音乐混音时使用，用来控制麦克风音量大小
// volume: 音量大小，1为正常音量，建议值为0-2，如果需要调大音量可以设置更大的值
[recorder setMicVolume:1.0];
```



```
// setBGMVolume 设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景音音量大小
// volume: 音量大小，1为正常音量，建议值为0-2，如果需要调大背景音量可以设置更大的值
[recorder setBGMVolume:1.0];
```

编辑添加 BGM

```
//初始化编辑器
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = videoView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;
ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];

//设置 BGM 路径
[ugcEdit setBGMAsset:fileAsset result:^(int result) {
}];

//设置 BGM 开始和结束时间
[ugcEdit setBGMStartTime:0 endTime:5];

//设置 BGM 是否循环
[ugcEdit setBGMLoop:YES];

//设置 BGM 在视频添加的起始位置
[ugcEdit setBGMAtVideoTime:0];

//设置视频声音大小
[ugcEdit setVideoVolume:1.0];

//设置 BGM 声音大小
[ugcEdit setBGMVolume:1.0];
```

BGM 设置完之后，当启动编辑器预览，BGM 就会根据设置的参数播放，当启动编辑器生成，BGM 也会按照设置的参数合成到生成的视频中。

变声和混响(iOS)

最近更新时间：2021-01-27 15:47:51

录制变声混响：

```
//获取 recorder 对象
recorder = [TXUGCRecord shareInstance];

// 设置混响
// TXRecordCommon.VIDOE_REVERB_TYPE_0 关闭混响
// TXRecordCommon.VIDOE_REVERB_TYPE_1 KTV
// TXRecordCommon.VIDOE_REVERB_TYPE_2 小房间
// TXRecordCommon.VIDOE_REVERB_TYPE_3 大会堂
// TXRecordCommon.VIDOE_REVERB_TYPE_4 低沉
// TXRecordCommon.VIDOE_REVERB_TYPE_5 洪亮
// TXRecordCommon.VIDOE_REVERB_TYPE_6 金属声
// TXRecordCommon.VIDOE_REVERB_TYPE_7 磁性
[recorder setReverbType:VIDOE_REVERB_TYPE_1];

// 设置变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0 关闭变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1 熊孩子
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2 萝莉
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3 大叔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4 重金属
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6 外国人
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7 困兽
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8 死肥仔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9 强电流
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
[record setVoiceChangerType:VIDOE_VOICECHANGER_TYPE_1];
```

> **说明：**
> 变声混响只针对录制人声有效，针对 BGM 无效。

预览裁剪和拼接
视频编辑(iOS)

最近更新时间：2021-07-27 17:30:40

功能概览

视频编辑包括视频裁剪、时间特效（慢动作、倒放、重复）、滤镜特效（动感光波、暗黑幻影、灵魂出窍、画面分裂）、滤镜风格（唯美、粉嫩、蓝调等）、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能。

相关类介绍

类名	功能
TXVideoInfoReader.h	媒体信息获取
TXVideoEditer.h	视频编辑

使用说明

视频编辑的基本使用流程如下：

1. 设置视频路径。
2. 添加效果。
3. 生成视频到指定文件。
4. 监听生成事件。

示例

```
// 这以使用了 Demo 中的 Common/UGC/VideoPreview 来做预览的视图
#import "VideoPreview.h"

@implementation EditViewController
{
TXVideoEditer *editor;
VideoPreview *_videoPreview;
}

- (void)viewDidLoad {
```

```
[super viewDidLoad];
_videoPreview = [[VideoPreview alloc] initWithFrame:self.view.bounds];
[self.view addSubview:_videoPreview];
// 编辑预览参数
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;

// 1. 初始化编辑器, 如无需预览，可以传 nil 或直接调用 init 方法
TXVideoEditer *editor = [[TXVideoEditer alloc] initWithPreview:param];

// 设置源视频路径
NSString *path = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"mp4"]
[editor setVideoPath: path];

// 配置代理
editor.generateDelegate = self; // 设置生成事件的回调委托对象，可以获取生成进度与结果

// 2. 对视频进行处理，这里以添加水印为例
[editor setWaterMark:[UIImage imageNamed:@"water_mark"]
normalizationFrame:CGRectMake(0,0,0.1,0)];
}

// 3. 生成视频, 以响应用户点击为例
- (IBAction)onGenerate:(id)sender {
NSString *output = [NSTemporaryDirectory() stringByAppendingPathComponent:@"temp.mp
4"];
[editor generateVideo:VIDEO_COMPRESSED_720P videoOutputPath:output];
}

// 4. 获取生成进度
-(void) onGenerateProgress:(float)progress
{
}

// 获取生成结果
-(void) onGenerateComplete:(TXGenerateResult *)result
{
if (result.retCode == 0) {
// 生成成功
```

```
} else {
// 生成失败，原因可以查看 result.descMsg
}
}
@end
```

视频信息获取

TXVideoInfoReader 的 getVideoInfo 方法可以获取指定视频文件的一些基本信息，相关接口如下：

```
// 获取视频文件的信息
+ (TXVideoInfo *)getVideoInfo:(NSString *)videoPath;

/** 获取视频文件信息
* @param videoAsset 视频文件属性
* @return 视频信息
*/
+ (TXVideoInfo *)getVideoInfoWithAsset:(AVAsset *)videoAsset;
```

返回的 TXVideoInfo 定义如下：

```
/// 视频信息
@interface TXVideoInfo : NSObject
/// 视频首帧图片
@property (nonatomic, strong) UIImage* coverImage;
/// 视频时长(s)
@property (nonatomic, assign) CGFloat duration;
/// 视频大小(byte)
@property (nonatomic, assign) unsigned long long fileSize;
/// 视频fps
@property (nonatomic, assign) float fps;
/// 视频码率 (kbps)
@property (nonatomic, assign) int bitrate;
/// 音频采样率
@property (nonatomic, assign) int audioSampleRate;
/// 视频宽度
@property (nonatomic, assign) int width;
/// 视频高度
```

```
@property (nonatomic, assign) int height;
/// 视频旋转角度
@property (nonatomic, assign) int angle;
@end
```

缩略图获取

缩略图的接口主要用于生成视频编辑界面的预览缩略图，或获取视频封面等。

1. 按个数平分时间获取缩略图

TXVideoInfoReader 的 getSampleImages 可以获取按指定数量，时间间隔相同的预览图：

```
/** 获取视频的采样图列表
* @param count 获取的采样图数量（均匀采样）
* @param maxSize 缩略图的最大大小，生成的缩略图大小不会超出这个宽高
* @param videoAsset 视频文件属性
* @param sampleProcess 采样进度
*/
+ (void)getSampleImages:(int)count
maxSize:(CGSize)maxSize
videoAsset:(AVAsset *)videoAsset
progress:(sampleProcess)sampleProcess;
```

开发包中的 VideoRangeSlider 即使用了 getSampleImages 获取了10张缩略图来构建一个由视频预览图组成的进度条。

2. 根据时间列表获取缩略图

```
/**
* 根据时间列表获取缩略图列表
* @param asset 视频文件对象
* @param times 获取的时间列表
* @param maxSize 缩略图大小
*/
+ (UIImage *)getSampleImagesFromAsset:(AVAsset *)asset
times:(NSArray<nsnumber*> *)times
maxSize:(CGSize)maxSize
progress:(sampleProcess)sampleProcess;
```

编辑预览

视频编辑提供了**定点预览**（将视频画面定格在某一时间点）与**区间预览**（循环播放某一时间段 A<=>B 内的视频片段）两种效果预览方式，使用时需要给 SDK 绑定一个 UIView 用于显示视频画面。

1. 绑定 UIView

TXVideoEditer 的 initWithPreview 函数用于绑定一个 UIView 给 SDK 来渲染视频画面，通过控制 TXPreviewParam 的 renderMode 来设置**自适应与填充**两种模式。

```
PREVIEW_RENDER_MODE_FILL_SCREEN - 填充模式，尽可能充满屏幕不留黑边，所以可能会裁剪掉一部分画面。
PREVIEW_RENDER_MODE_FILL_EDGE - 适应模式，尽可能保持画面完整，但当宽高比不合适时会有黑边出现。
```

2. 定点预览

TXVideoEditer 的 previewAtTime 函数用于定格显示某一个时间点的视频画面。

```
/** 渲染某一时刻的视频画面
* @param time 预览帧时间(s)
*/
- (void)previewAtTime:(CGFloat)time;
```

3. 区间预览

TXVideoEditer 的 startPlayFromTime 函数用于循环播放某一时间段 A<=>B 内的视频片段。

```
/** 播放某一时间段的视频
* @param startTime 播放起始时间(s)
* @param endTime 播放结束时间(s)
*/
- (void)startPlayFromTime:(CGFloat)startTime
toTime:(CGFloat)endTime;
```

4. 预览的暂停与恢复

```
/// 暂停播放
- (void)pausePlay;
```

```
/// 继续播放
- (void)resumePlay;
/// 停止播放
- (void)stopPlay;
```

5. 美颜滤镜

您可以给视频添加滤镜效果，例如美白、浪漫、清新等滤镜，demo 提供了多种滤镜选择，对应的滤镜资源在 Common/Resource/Filter/FilterResource.bundle 中，同时也可以设置自定义的滤镜。
设置滤镜的方法为：

```
- (void) setFilter:(UIImage *)image;
```

其中 image 为滤镜映射图，image 设置为 nil，会清除滤镜效果。

Demo 示例：

```
TXVideoEditer *_ugcEdit;
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

6. 设置水印

1. 设置全局水印

您可以为视频设置水印图片，并且可以指定图片的位置。
设置水印的方法为：

```
- (void) setWaterMark:(UIImage *)waterMark normalizationFrame:(CGRect)normalizationFrame;
```

其中 waterMark 表示水印图片，normalizationFrame 是相对于视频图像的归一化 frame，frame 的 x、y、width、height 的取值范围都为0 – 1。

Demo 示例：

```
UIImage *image = [UIImage imageNamed:@"watermark"];
[_ugcEdit setWaterMark:image normalizationFrame:CGRectMake(0, 0, 0.3 , 0.3 * image.size.height / image.size.width)];//水印大小占视频宽度的30%，高度根据宽度自适应
```

2. 设置片尾水印

您可以为视频设置片尾水印，并且可以指定片尾水印的位置。
设置片尾水印的方法为：

```
- (void) setTailWaterMark:(UIImage *)tailWaterMark normalizationFrame:(CGRect)normalizationFrame
duration:(CGFloat)duration;
```

其中 tailWaterMark 表示片尾水印图片，normalizationFrame 是相对于视频图像的归一化 frame，frame 的 x、y、width、height 的取值范围都为0 – 1，duration 为水印的持续时长。
Demo 示例：设置水印在片尾中间，持续时间1s。

```
UIImage *tailWaterimage = [UIImage imageNamed:@"tcloud_logo"];
float w = 0.15;
float x = (1.0 - w) / 2.0;
float width = w * videoMsg.width;
float height = width * tailWaterimage.size.height / tailWaterimage.size.width;
float y = (videoMsg.height - height) / 2 / videoMsg.height;
[_ugcEdit setTailWaterMark:tailWaterimage normalizationFrame:CGRectMake(x,y,w,0) duration:1];
```

压缩裁剪

视频码率设置

```
/**
* 设置视频码率
* @param bitrate 视频码率 单位:kbps
* 如果设置了码率，SDK 生成视频会优先使用这个码率，注意码率不要太大或则太小，码率太小视频会模糊不清，码率太大，生成视频体积会很大
* 这里建议设置范围为：600-12000，如果没有调用这个接口，SDK内部会根据压缩质量自动计算码率
```

```
*/
- (void) setVideoBitrate:(int)bitrate;
```

视频裁剪

视频编辑类操作都符合同一个操作原则：即先设定操作指定，最后用 generateVideo 将所有指令顺序执行，这种方式可以避免多次重复压缩视频引入的不必要的质量损失。

```
TXVideoEditer* _ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
// 设置裁剪的起始时间和结束时间
[_ugcEdit setCutFromTime:_videoRangeSlider.leftPos toTime:_videoRangeSlider.rightPos];
// ...
// 生成最终的视频文件
_ugcEdit.generateDelegate = self;
[_ugcEdit generateVideo:VIDEO_COMPRESSED_540P videoOutputPath:_videoOutputPath];
```

输出时指定文件压缩质量和输出路径，输出的进度和结果会通过 generateDelegate 以回调的形式通知用户。

高级功能

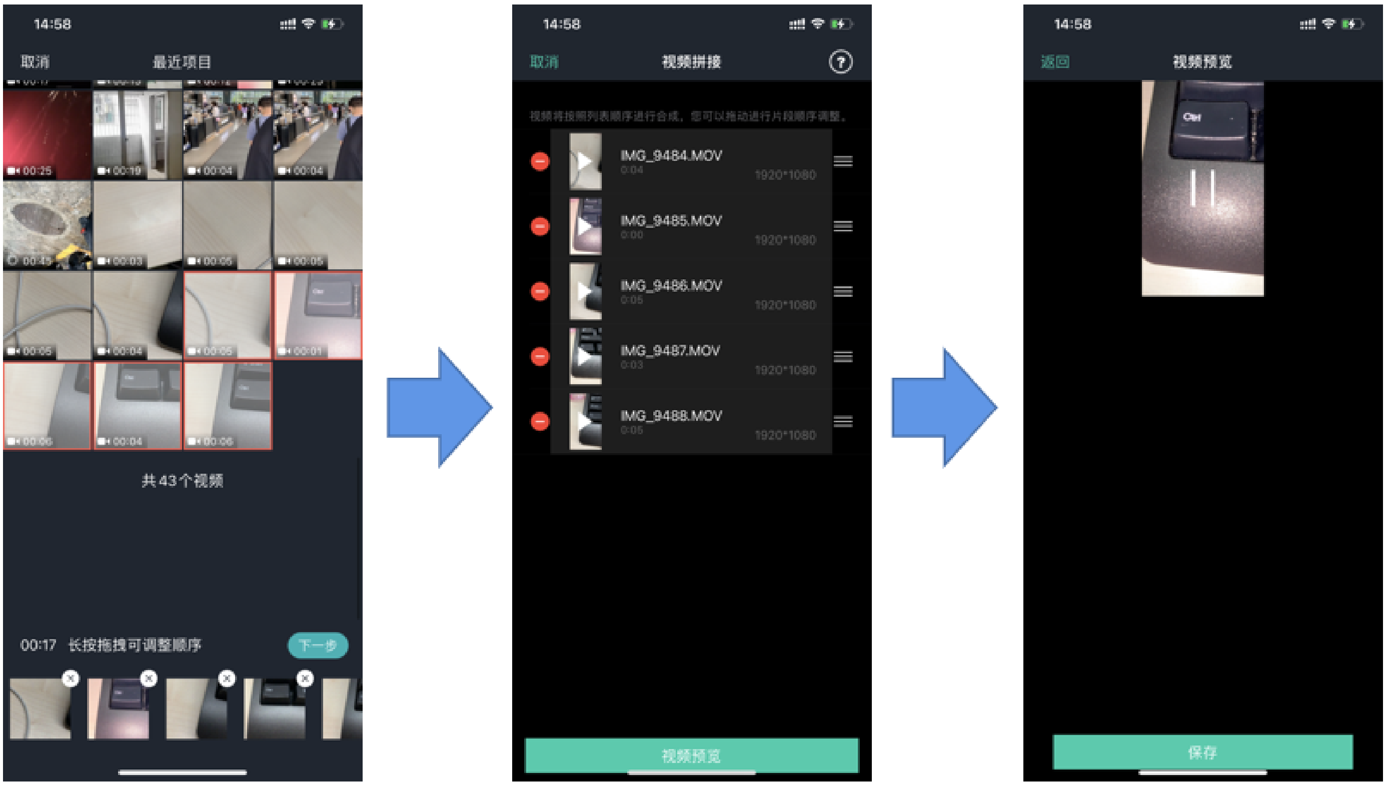
- 类抖音特效
- 设置背景音乐
- 贴纸字幕
- 图片编辑

视频拼接(iOS)

最近更新时间：2021-12-20 15:49:17

复用现有 UI

视频拼接器具有比较复杂的交互逻辑，这也决定了其 UI 复杂度很高，所以我们比较推荐复用 SDK 开发包中的 UI 源码。VideoJoiner 目录包含短视频拼接器的 UI 源码。



视频选取　　顺序调整　　效果预览

- **VideoJoinerController**：用于实现上图中的视频拼接列表，支持上下拖拽调整顺序。
- **VideoJoinerCell**：用于实现拼接列表中的每一个视频片段。
- **VideoEditPrevController**：用于预览拼接后的视频观看效果。

自己实现 UI

如果您不考虑复用我们开发包中的 UI 代码，想自己实现 UI 部分，则可以参考如下的攻略进行对接。

1. 选择视频文件

Demo 中使用了 QBImagePicker 这样一个开源库实现了多个文件的选择功能，相关代码在 Demo 的 MainViewController 里有所体现。

2. 设置预览 View

视频合成需要创建 TXVideoJoiner 对象，同 TXUGCEditer 类似，预览功能也需要上层提供预览 UIView：

```
//准备预览 View
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;

// 创建 TXVideoJoiner 对象并设置预览 view
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc] initWithPreview:param];
_videoJoin.previewDelegate = _videoPreview;

// 设置待拼接的视频文件组 _composeArray，也就是第一步中选择的若干个文件
[_videoJoin setVideoPathList:_composeArray];
```

设置好预览 view 同时传入待合成的视频文件数组后，可以开始播放预览，合成模块提供了一组接口来做视频的播放预览：

- startPlay：表示视频播放开始。
- pausePlay：表示视频播放暂停。
- resumePlay：表示视频播放恢复。

3. 生成最终文件

预览效果满意后调用生成接口即可生成合成后的文件：

```
_videoJoin.joinerDelegate = self;
[_videoJoin joinVideo:VIDEO_COMPRESSED_540P videoOutputPath:_outFilePath];
```

合成时指定文件压缩质量和输出路径，输出的进度和结果会通过 joinerDelegate 以回调的形式通知用户。

上传和播放

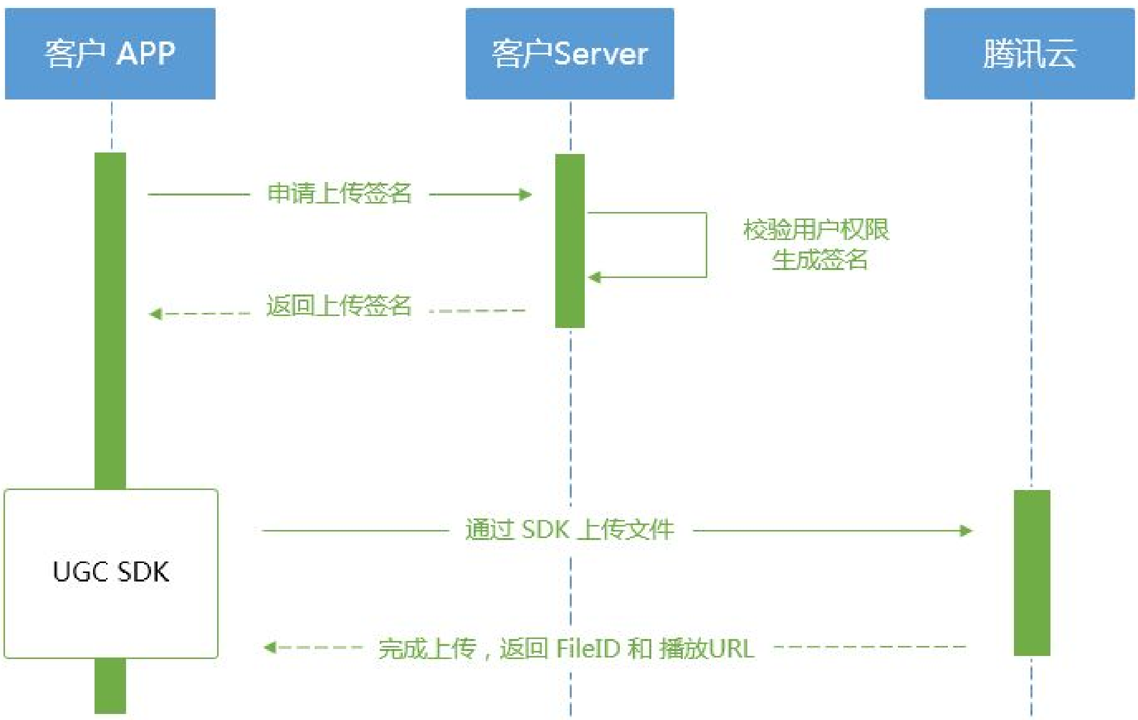
签名派发

最近更新时间：2021-07-26 16:21:03

客户端视频上传，是指 App 的最终用户将本地视频直接上传到腾讯云点播。客户端上传的详细介绍请参考点播 客户端上传指引，本文将以最简洁的方式介绍客户端上传的签名生成方法。

总体介绍

客户端上传的整体流程如下图所示：



为了支持客户端上传，开发者需要搭建两个后台服务：签名派发服务和事件通知接收服务。

- 客户端首先向签名派发服务请求上传签名。
- 签名派发服务校验该用户是否有上传权限，若校验通过，则生成签名并下发；否则返回错误码，上传流程结束。
- 客户端拿到签名后使用短视频 SDK 中集成的上传功能来上传视频。
- 上传完成后，点播后台会发送 上传完成事件通知 给开发者的事件通知接收服务。
- 如果签名派发服务在签名中指定了视频处理 任务流，点播服务会在视频上传完成后根据指定流程自动进行视频处理。短视频场景下的视频处理一般为 AI 鉴黄。
- 视频处理完成之后，点播后台会发送 任务流状态变更事件通知 给开发者的事件通知接收服务。

至此整个视频上传 - 处理流程结束。

签名生成

有关客户端上传签名的详细介绍请参考点播 客户端上传签名。

签名派发服务实现示例

```
/**
* 计算签名
*/
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId = 0, oneTime
Valid = 0, sourceContext = '' }) {
// 确定签名的当前时间和失效时间
let current = parseInt((new Date()).getTime() / 1000)
let expired = current + timeStamp; // 签名有效期：1天
// 向参数列表填入参数
let arg_list = {
//required
secretId: this.conf.SecretId,
currentTimeStamp: current,
expireTime: expired,
random: Math.round(Math.random() * Math.pow(2, 32)),
//opts
procedure,
classId,
oneTimeValid,
sourceContext
}
// 计算签名
let orignal = querystring.stringify(arg_list);
let orignal_buffer = new Buffer(orignal, "utf8");
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(orignal_buffer).digest();
let signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");
return signature;
}
/**
* 响应签名请求
*/
```



```
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
signature: gVodHelper.createFileUploadSignature({})
}
});
}
```

视频上传（iOS）

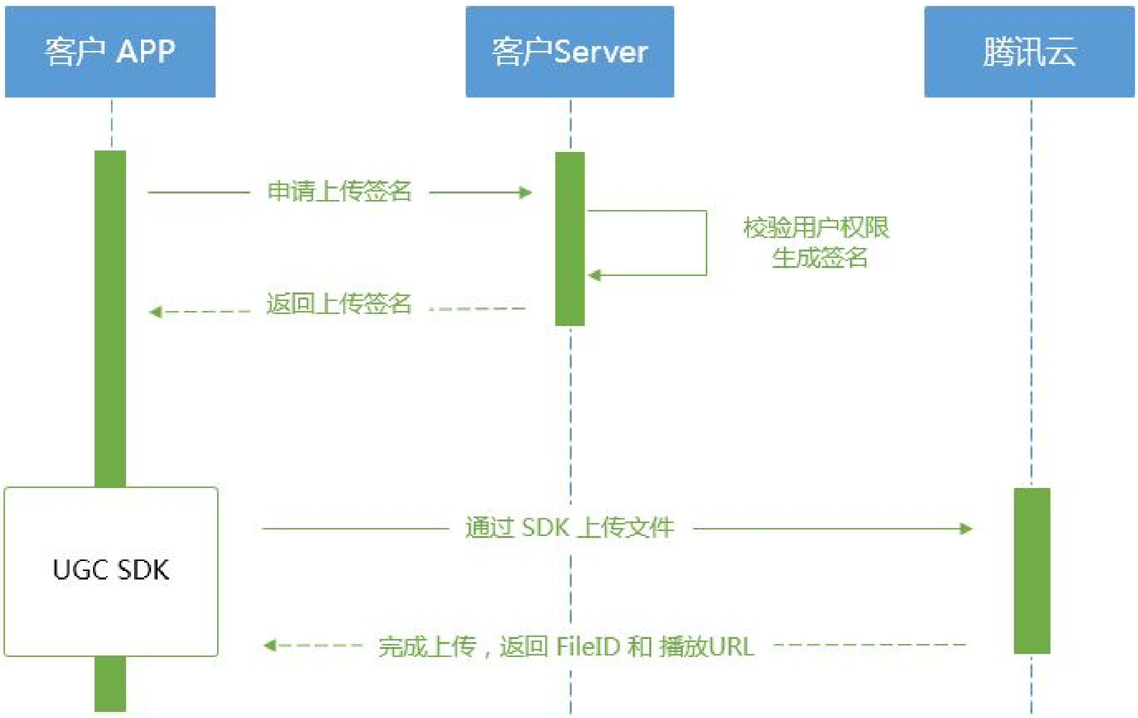
最近更新时间：2021-07-26 09:48:57

计算上传签名

客户端视频上传，是指 App 的最终用户将本地视频直接上传到腾讯云点播。客户端上传的详细介绍请参见点播 客户端上传指引，本文将以最简洁的方式介绍客户端上传的签名生成方法。

总体介绍

客户端上传的整体流程如下图所示：



为了支持客户端上传，开发者需要搭建两个后台服务：签名派发服务和事件通知接收服务。

- 客户端首先向签名派发服务请求上传签名。
- 签名派发服务校验该用户是否有上传权限，若校验通过，则生成签名并下发；否则返回错误码，上传流程结束。
- 客户端拿到签名后使用短视频 SDK 中集成的上传功能来上传视频。
- 上传完成后，点播后台会发送 上传完成事件通知 给开发者的事件通知接收服务。
- 如果签名派发服务在签名中指定了视频处理 任务流，点播服务会在视频上传完成后根据指定流程自动进行视频处理。短视频场景下的视频处理一般为 AI 鉴黄。
- 视频处理完成之后，点播后台会发送 任务流状态变更事件通知 给开发者的事件通知接收服务。

至此整个视频上传-处理流程结束。

签名生成

有关客户端上传签名的详细介绍请参见点播 客户端上传签名。

签名派发服务实现示例

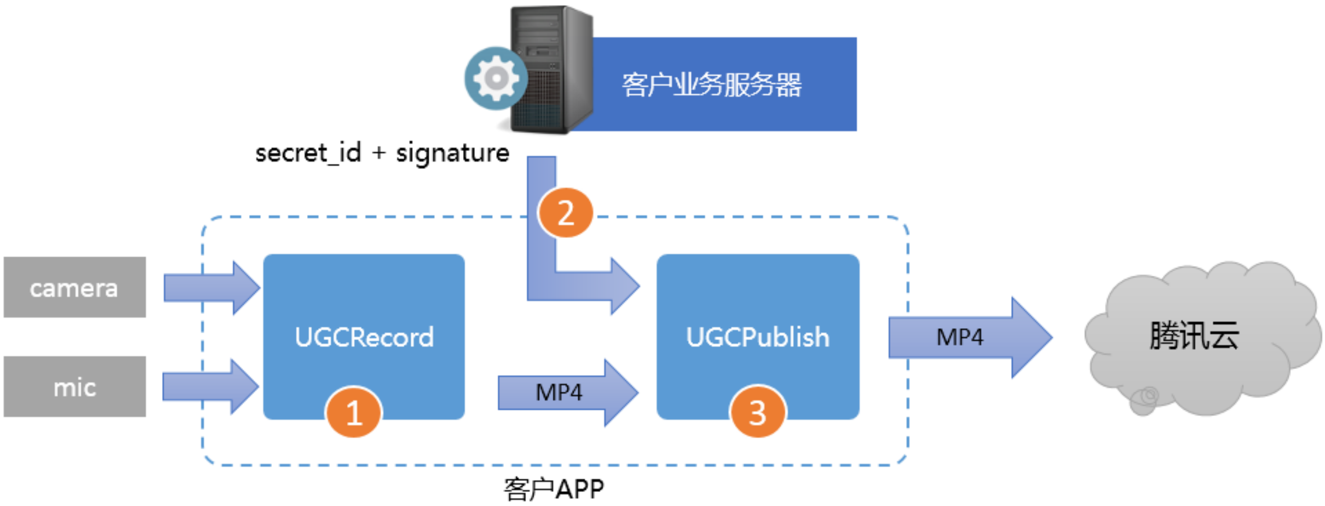
```
/**
* 计算签名
*/
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId = 0, oneTime
Valid = 0, sourceContext = '' }) {
// 确定签名的当前时间和失效时间
let current = parseInt((new Date()).getTime() / 1000)
let expired = current + timeStamp; // 签名有效期：1天
// 向参数列表填入参数
let arg_list = {
//required
secretId: this.conf.SecretId,
currentTimeStamp: current,
expireTime: expired,
random: Math.round(Math.random() * Math.pow(2, 32)),
//opts
procedure,
classId,
oneTimeValid,
sourceContext
}
// 计算签名
let orignal = querystring.stringify(arg_list);
let orignal_buffer = new Buffer(orignal, "utf8");
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(orignal_buffer).digest();
let signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");
return signature;
}
/**
* 响应签名请求
*/
function getUploadSignature(req, res) {
```

```
res.json({
code: 0,
message: 'ok',
data: {
signature: gVodHelper.createFileUploadSignature({})
}
});
}
```

对接流程

短视频发布

将 MP4 文件上传到腾讯视频云，并获得在线观看 URL， 腾讯视频云支持视频观看的就近调度、秒开播放、动态加速 以及海外接入等要求，从而确保优质的观看体验。



- 第一步：使用 TXUGCRecord 接口录制一段小视频，录制结束后会生成一个小视频文件（MP4）回调给客户。
- 第二步：您的 App 向您的业务服务器申请上传签名。上传签名是 App 将 MP4 文件上传到腾讯云视频分发平台的“许可证”，为了确保安全性，这些上传签名都要求由您的业务 Server 进行签发，而不能由终端 App 生成。
- 第三步：使用 TXUGCPublish 接口发布视频，发布成功后 SDK 会将观看地址的 URL 回调给您。

特别注意

- App 千万不要把计算上传签名的 SecretID 和 SecretKey 写在客户端的代码里，这两个关键信息泄露将导致安全隐患，如恶意攻击者一旦破解 App 获取该信息，就可以免费使用您的流量和存储服务。
- 正确的做法是在您的服务器上用 SecretID 和 SecretKey 生成一次性的上传签名然后将签名交给 App。因为服务器一般很难被攻陷，所以安全性是可以保证的。

- 发布短视频时，请务必保证正确传递 Signature 字段，否则会发布失败。

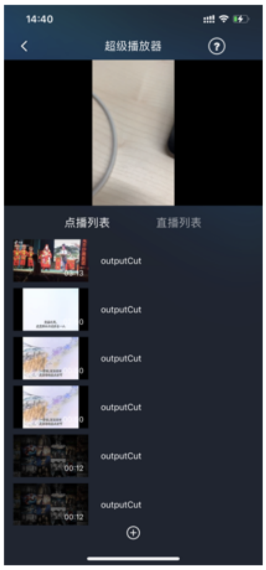
对接攻略

上传入口

选择视频

压缩视频

预览发布



视频点播

1. 选择视频

可以接着上篇文档中的录制或者编辑，把生成的视频进行上传，或者可以选择手机本地的视频进行上传。

2. 压缩视频

对选择的视频进行压缩，使用 TXVideoEditer.generateVideo(int videoCompressed, String videoOutputPath) 接口，支持4种分辨率的压缩，后续会增加自定义码率的压缩。

3. 发布视频

把刚才生成的 MP4 文件发布到腾讯云上，App 需要拿到上传文件用的短期有效上传签名，这部分有独立的文档介绍，详情请参考 签名派发。

TXUGCPublish（位于 TXUGCPublish.h）负责将 MP4 文件发布到腾讯云视频分发平台上，以确保视频观看的就近调度、秒开播放、动态加速以及海外接入等需求。

```
TXPublishParam * param = [[TXPublishParam alloc] init];
param.signature = _signature; // 需要填写第四步中计算的上传签名
// 录制生成的视频文件路径 TXVideoRecordListener 的 onRecordComplete 回调中可以获取
param.videoPath = _videoPath;
// 录制生成的视频首帧预览图路径。值为通过调用 startRecord 指定的封面路径，或者指定一个路径，然后将TXVideoRecordListener 的 onRecordComplete 回调中获取到的 UIImage 保存到指定路径下，可以置为 nil。
param.coverPath = _coverPath;
```

```
TXUGCPublish *_ugcPublish = [[TXUGCPublish alloc] init];
// 文件发布默认是采用断点续传
_ugcPublish.delegate = self; // 设置 TXVideoPublishListener 回调
[_ugcPublish publishVideo:param];
```

发布的过程和结果是通过 TXVideoPublishListener（位于 TXUGCPublishListener.h 头文件中定义）接口反馈出来的：

- onPublishProgress 用于反馈文件发布的进度，参数 uploadBytes 表示已经上传的字节数，参数 totalBytes 表示需要上传的总字节数。

```
@optional
-(void) onPublishProgress:(NSInteger)uploadBytes totalBytes: (NSInteger)totalBytes;
```

- onPublishComplete 用于反馈发布结果，TXPublishResult 的字段 errCode 和 descMsg 分别表示错误码和错误描述信息，videoURL 表示短视频的点播地址，coverURL 表示视频封面的云存储地址，videoId 表示视频文件云存储 Id，您可以通过这个 Id 调用点播 服务端API接口。

```
@optional
-(void) onPublishComplete:(TXPublishResult*)result;
```

- 发布结果
  通过 错误码表 来确认短视频发布的结果。

4. 播放视频

第3步上传成功后，会返回视频的 fileId，播放地址 URL，封面 URL。用 点播播放器 可以直接传入 fileId 播放，或者 URL 播放。

iOS 播放器 SDK

最近更新时间：2021-07-28 15:55:17

简介

超级播放器 SDK 是腾讯云开源的一款播放器组件，简单几行代码即可拥有类似腾讯视频强大的播放功能，包括横竖屏切换、清晰度选择、手势和小窗等基础功能，还支持视频缓存，软硬解切换和倍速播放等特殊功能，相比系统播放器，支持格式更多，兼容性更好，功能更强大，同时还具备首屏秒开、低延迟的优点，以及视频缩略图等高级能力。

SDK 下载

云点播 iOS 超级播放器的项目地址是 SuperPlayer_iOS。

阅读对象

本文档部分内容为腾讯云专属能力，使用前请开通 腾讯云 相关服务，未注册用户可 注册账号。

快速集成

本项目支持 cocoapods 安装，只需要将如下代码添加到 Podfile 中：

```
pod 'SuperPlayer'
```

执行 pod install 或 pod update。

使用播放器

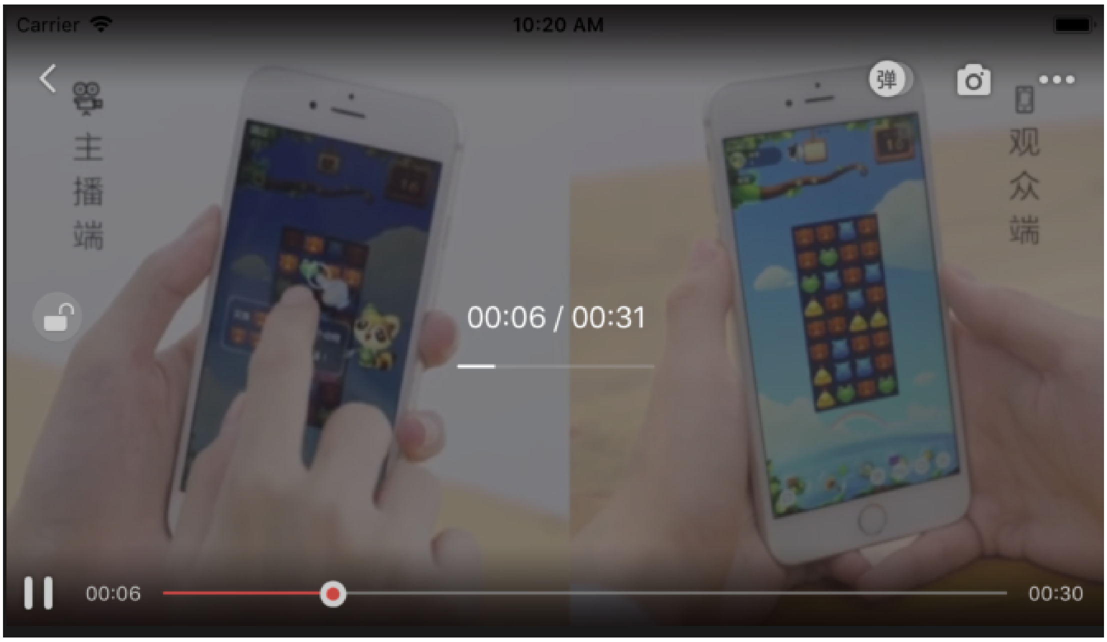
播放器主类为 SuperPlayerView，创建后即可播放视频。

```
// 引入头文件
#import <SuperPlayer/SuperPlayer.h>
// 创建播放器
_playerView = [[SuperPlayerView alloc] init];
// 设置代理，用于接受事件
_playerView.delegate = self;
// 设置父 View，_playerView 会被自动添加到 holderView 下面
_playerView.fatherView = self.holderView;
```

```
//不开防盗链
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329073;// 配置 AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = "5285890799710670616"; // 配置 FileId
[_playerView playWithModel:model];
```

```
//开启防盗链需填写 psign，psign 即超级播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// 配置 AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = "5285890799710173650"; // 配置 FileId
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
[_playerView playWithModel:model];
```

运行代码，可以看到视频在手机上播放，并且界面上大部分功能都处于可用状态。



选择 FileId

视频 FileId 在一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 FileId 到客户端。
2. 服务端视频上传时，在 确认上传 的通知中包含对应的 FileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件，查看 FileId。如下图所示，ID 即表示 FileId：

视频信息	视频状态	视频分类	视频来源	上传时间	操作
00:01:01 ID:	正常	其他	上传	2019-02-01 15:00:33	管理 删除
00:01:01 ID:	正常	其他	上传	2019-02-01 12:04:50	管理 删除
00:01:01 ID:	正常	其他	上传	2018-05-24 10:12:37	管理 删除

打点功能

在播放长视频时，打点信息有助于观众找到感兴趣的点。使用 修改媒体文件属性 API，通过 AddKeyFrameDescs.N 参数可以为视频设置打点信息。

调用后，播放器的界面会增加新的元素。

小窗播放

小窗播是指在 App 内，悬浮在主 window 上的播放器。使用小窗播放非常简单，只需要在适当位置调用下面代码即可：

```
[SuperPlayerWindow sharedInstance].superPlayer = _playerView; // 设置小窗显示的播放器
[SuperPlayerWindow sharedInstance].backController = self; // 设置返回的 view controller
[[SuperPlayerWindow sharedInstance] show]; // 悬浮显示
```

退出播放

当不需要播放器时，调用 resetPlayer 清理播放器内部状态，释放内存。

```
[_playerView resetPlayer];
```

更多功能

完整功能可扫码下载视频云工具包体验，或直接运行工程 Demo。