音视频终端引擎

短视频

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2022 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。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默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

短视频
短视频录制
拍照和录制
iOS

最近更新时间：2021-08-12 19:55:21

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。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

使用类介绍

腾讯云视立方短视频 UGSV 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

版权所有：腾讯云计算（北京）有限责任公司

使用流程

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。

示例代码

```
@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

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];
```

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];
```

版权所有：腾讯云计算（北京）有限责任公司

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。

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.width )
[recorder setWaterMark:waterMarkImage normalizationFrame:frame)
```

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];
```

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

Android

最近更新时间：2021-08-23 17:39:48

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。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

相关类介绍

类	功能
TXUGCRecord	实现视频的录制功能
TXUGCPartsManager	视频片段管理类，用于视频的多段录制，回删等
ITXVideoRecordListener	录制回调
TXRecordCommon	基本参数定义，包括了视频录制回调及发布回调接口

使用说明

使用流程

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。

版权所有：腾讯云计算（北京）有限责任公司

代码示例

```
// 创建一个用户相机预览的 TXCloudVideoView
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view);
// 1、配置录制参数，已推荐配置 TXUGCSimpleConfig 为例
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM;
// 2、启动画面预览
mTXCameraRecord.startCameraSimplePreview(param, mVideoView);
// 3、设置录制效果，这里以添加水印为例
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXCameraRecord.setWatermark(BitmapFactory.decodeResource(getResources(), R.drawable.
watermark), rect);
// 4、开始录制
int result = mTXCameraRecord.startRecord();
if (result != TXRecordCommon.START_RECORD_OK) {
if (result == -4) {画面还没出来}
else if (result == -3) {//版本太低}
else if (result == -5) {// licence 验证失败] }
}
else{// 启动成功}
// 结束录制
mTXCameraRecord.stopRecord();
// 录制完成回调
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode >= 0 ) {
// 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 TXRecordCommon 中“录制结果回调错误码定义”
}
}
```

画面预览

TXUGCRecord（位于 TXUGCRecord.java）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 mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext());
mTXCameraRecord.setVideoRecordListener(this); // 设置录制回调
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view); // 准备一个预览摄像头画面的
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p
param.isFront = true; // 是否使用前置摄像头
param.minDuration = 5000; // 视频录制的最小时长 ms
param.maxDuration = 60000; // 视频录制的最大时长 ms
param.touchFocus = false; // false 为自动聚焦；true 为手动聚焦
mTXCameraRecord.startCameraSimplePreview(param,mVideoView);

// 结束画面预览
mTXCameraRecord.stopCameraPreview();
```

2. 调整预览参数

在相机启动后，可以通过以下方法调整预览参数：

```
// 切换视频录制分辨率到540p
mTXCameraRecord.setVideoResolution(TXRecordCommon.VIDEO_RESOLUTION_540_960);

// 切换视频录制码率到6500Kbps
mTXCameraRecord.setVideoBitrate(6500);

// 获取摄像头支持的最大焦距
mTXCameraRecord.getMaxZoom();

// 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）
mTXCameraRecord.setZoom(3);
```

```
// 切换到后置摄像头 true 切换到前置摄像头；false 切换到后置摄像头
mTXCameraRecord.switchCamera(false);

// 打开闪光灯 true 为打开， false 为关闭.
mTXCameraRecord.toggleTorch(false);

// param.touchFocus 为 true 时为手动聚焦，可以通过下面接口设置聚焦位置
mTXCameraRecord.setFocusPosition(eventX, eventY);

// 设置自定义图像处理回调
mTXCameraRecord.setVideoProcessListener(this);
```

拍照

在相机开启预览后，即可使用拍照的功能。

```
// 截图/拍照，startCameraSimplePreview 或者 startCameraCustomPreview 之后调用有效
mTXCameraRecord.snapshot(new TXRecordCommon.ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
// 保存或者显示截图
}
});
```

录制过程控制

录制的开始、暂停与恢复。

```
// 开始录制
mTXCameraRecord.startRecord();

// 开始录制，可以指定输出视频文件地址和封面地址
mTXCameraRecord.startRecord(videoFilePath, 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、封面地址
mTXCameraRecord.startRecord(videoFilePath, videoPartFolder, coverPath);
```

```
// 暂停录制
mTXCameraRecord.pauseRecord();

// 继续录制
mTXCameraRecord.resumeRecord();

// 结束录制
mTXCameraRecord.stopRecord();
```

录制的过程和结果是通过 TXRecordCommon.ITXVideoRecordListener（位于 TXRecordCommon.java 中定义）接口反馈：

- 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(long 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(TXRecordResult result);
```

- onRecordEvent 录制事件回调，包含事件id和事件相关的参数（key,value）格式。

```
@optional
void onRecordEvent(final int event, final Bundle param);
```

录制属性设置

画面设置

```
// 设置横竖屏录制
mTXCameraRecord.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
```

```
// 设置视频预览方向
// TXLiveConstants.RENDER_ROTATION_0(常规竖屏)
// TXLiveConstants.RENDER_ROTATION_90(左旋90度)
// TXLiveConstants.RENDER_ROTATION_180(左旋180度)
// TXLiveConstants.RENDER_ROTATION_270(左旋270度)
// 注意：需要在 startRecord 之前设置，录制过程中设置无效
mTXCameraRecord.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);

// 设置录制的宽高比
// TXRecordCommon.VIDEO_ASPECT_RATIO_9_16 宽高比为9:16
// TXRecordCommon.VIDEO_ASPECT_RATIO_3_4 宽高比为3:4
// TXRecordCommon.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
mTXCameraRecord.setAspectRatio(TXRecordCommon.VIDEO_ASPECT_RATIO_9_16);
```

速度设置

```
// 设置视频录制速率
// TXRecordCommon.RECORD_SPEED_SLOWEST(极慢速)
// TXRecordCommon.RECORD_SPEED_SLOW(慢速)
// TXRecordCommon.RECORD_SPEED_NORMAL(标准)
// TXRecordCommon.RECORD_SPEED_FAST(快速)
// TXRecordCommon.RECORD_SPEED_FASTEST(极快速)
mTXCameraRecord.setRecordSpeed(TXRecordCommon.VIDEO_RECORD_SPEED_NORMAL);
```

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.
mTXCameraRecord.setMicVolume(volume);
// 设置录制是否静音 参数 isMute 代表是否静音，默认不静音
mTXCameraRecord.setMute(isMute);
```

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。

水印

```
// 设置全局水印
// TXRect-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）TXRect 三个参数设置为0.1，0.1，0.1
// 水印的实际像素坐标为（540 * 0.1，960 * 0.1，540 * 0.1 ，
// 540 * 0.1 * watermarkBitmap.height / watermarkBitmap.width ）
mTXCameraRecord.setWatermark(watermarkBitmap, txRect)
```

滤镜

```
// 设置颜色滤镜：浪漫、清新、唯美、粉嫩、怀旧...
// filterBitmap：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 RTMPAndroidDemo/app/src/main/res/drawable-xxhdpi/ 目录下。
mTXCameraRecord.setFilter(filterBitmap);

// 设置组合滤镜特效
// mLeftBitmap 左侧滤镜
// leftIntensity 左侧滤镜程度
// mRightBitmap 右侧滤镜
// rightIntensity 右侧滤镜程度
// 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。
mTXCameraRecord.setFilter(mLeftBitmap, leftIntensity, mRightBitmap, rightIntensity, leftRatio);

// 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
mTXCameraRecord.setSpecialRatio(0.5);
```

美颜

```
// 设置美颜类型
mTXCameraRecord.setBeautyStyle(style);

// 设置大眼效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setEyeScaleLevel(eyeScaleLevel);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 设置瘦脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceScaleLevel(faceScaleLevel);

// 设置V脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceVLevel(level);

// 设置下巴拉伸或收缩效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setChinLevel(scale);

// 设置缩脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceShortLevel(level);

// 设置小鼻效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setNoseSlimLevel(scale);

// 设置绿幕文件:目前图片支持 jpg/png，视频支持 mp4/3gp 等 Android 系统支持的格式并支持循环播放
mTXCameraRecord.setGreenScreenFile(path, isLoop);

// 设置动效贴纸 motionTmplPath 动效文件路径： 空 String "" 则取消动效
mTXCameraRecord.setMotionTmp(motionTmplPath);

// 设置动效贴纸是否静音: true: 动效贴纸静音；false：动效贴纸不静音
mTXCameraRecord.setMotionMute(true);
```

获取 License 信息

腾讯云视立方短视频 UGSV SDK 具备了短视频 License 的校验，如果校验没通过，您可以通过该接口来查询 License 中具体信息：

```
TXUGCBase.getInstance().getLicenceInfo(Context context);
```

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：2021-08-12 19:55:32

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

使用流程

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。

示例代码

```
//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];
```

Android

最近更新时间：2021-08-12 19:55:41

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	-	✓	-	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

使用流程

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. 继续录制。
7. 继续播放 BGM。
8. 停止录制。
9. 停止 BGM。

示例代码

```
// 开始录制
mTXCameraRecord.startRecord();

// pauseRecord 后会生成一段视频，视频可以在 TXUGCPartsManager 里面获取
mTXCameraRecord.pauseRecord();
mTXCameraRecord.pauseBGM();

// 继续录制视频
```

```
mTXCameraRecord.resumeRecord();
mTXCameraRecord.resumeBGM();

// 停止录制，将多段视频合成为一个视频输出
mTXCameraRecord.stopBGM();
mTXCameraRecord.stopRecord();

// 获取片段管理对象
mTXCameraRecord.getPartsManager();

// 获取当前所有视频片段的总时长
mTXUGCPartsManager.getDuration();

// 获取所有视频片段路径
mTXUGCPartsManager.getPartsPathList();

// 删除最后一段视频
mTXUGCPartsManager.deleteLastPart();

// 删除指定片段视频
mTXUGCPartsManager.deletePart(index);

// 删除所有片段视频
mTXUGCPartsManager.deleteAllParts();

// 您可以添加当前录制视频之外的视频
mTXUGCPartsManager.insertPart(videoPath, index) ;
```

录制草稿箱

iOS

最近更新时间：2021-08-12 19:55:46

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。

示例代码

```
//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：**
>
> 具体实现方法请参见 小视频源码 中的 UGCKitRecordViewController 类。

Android

最近更新时间：2021-12-20 16:21:13

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	-	✓	-	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。

示例代码

```
// 获取第一次视频录制对象
mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext());

// 开始录制
mTXCameraRecord.startRecord();

// 暂停录制
mTXCameraRecord.pauseRecord();
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 获取缓存的录制分片，记录到本地
List<String> pathList = mTXCameraRecord.getPartsManager().getPartsPathList(); // pathList 写本地

// 第二次打开 app，获取录制对象
mTXCameraRecord2 = TXUGCRecord.getInstance(this.getApplicationContext());

// 预加载本地缓存片段
mTXCameraRecord2.getPartsManager().insertPart(videoPath, 0);

// 开始录制
mTXCameraRecord2.startRecord();

// 结束录制,SDK 会把缓存视频片段和当前录制视频片段合成
mTXCameraRecord2.stopRecord();
```

说明：

具体实现方法请参见 小视频源码 中录制中的 RecordDraftManager 类的使用。

短视频编辑
添加背景音乐
iOS

最近更新时间：2021-08-12 19:55:54

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。

Android

最近更新时间：2021-08-12 19:55:57

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制添加 BGM

```
// 设置 BGM 路径
mTXCameraRecord.setBGM(path);

// 设置 BGM 播放回调 TXRecordCommon.ITXBGMNotify
mTXCameraRecord.setBGMNofify(notify);

// 播放 BGM
mTXCameraRecord.playBGMFromTime(startTime, endTime)

// 停止播放 BGM
mTXCameraRecord.stopBGM();

// 暂停播放 BGM
mTXCameraRecord.pauseBGM();

// 继续播放 BGM
mTXCameraRecord.resumeBGM();

// 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
// 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
mTXCameraRecord.setBGMVolume(x);
```

```
// 设置背景音乐播放的开始位置和结束位置
mTXCameraRecord.seekBGM(startTime, endTime);
```

编辑添加 BGM

```
// 设置 BGM 路径，返回值为0表示设置成功； 其他表示失败，如：不支持的音频格式。
public int setBGM(String path);

// 设置 BGM 开始和结束时间，单位毫秒
public void setBGMStartTime(long startTime, long endTime);

// 设置背景音乐是否循环播放：true：循环播放，false：不循环播放
public void setBGMLoop(boolean looping);

// 设置 BGM 在视频添加的起始位置
public void setBGMAtVideoTime(long videoStartTime);

// 设置视频声音大小， volume 表示声音的大小， 取值范围0 - 1 ， 0 表示静音， 1 表示原声大小。
public void setVideoVolume(float volume);

// 设置BGM声音大小，volume 表示声音的大小， 取值范围0 - 1 ， 0 表示静音， 1 表示原声大小。
public void setBGMVolume(float volume);
```

说明：

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：2021-08-12 19:56:01

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	✓	✓	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 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。

Android

最近更新时间：2021-08-12 19:56:04

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	✓	✓	✓	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制变声混响

```
// 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
mTXCameraRecord.setReverb(TXRecordCommon.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
```

版权所有：腾讯云计算（北京）有限责任公司

```
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
mTXCameraRecord.setVoiceChangerType(TXRecordCommon.VIDOE_VOICECHANGER_TYPE_
1);
```

说明：

变声混响只针对录制人声有效，针对 BGM 无效。

预览裁剪编辑

iOS

最近更新时间：2021-08-12 19:56:08

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。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。

代码示例

```
// 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。

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。

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：

```
- (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:@"bundl
e"];
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

6. 设置水印

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：

```
- (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.hei
ght / image.size.width)];//水印大小占视频宽度的30%，高度根据宽度自适应
```

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：

```
- (void) setTailWaterMark:(UIImage *)tailWaterMark normalizationFrame:(CGRect)normalizationF
rame
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
- 图片转场特效

Android

最近更新时间：2021-08-12 19:56:14

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。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

相关类介绍

类名	功能
TXVideoInfoReader	媒体信息获取
TXVideoEditer	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. 视频导入。
3. 添加效果。
4. 生成视频到指定文件。
5. 监听生成事件。
6. 资源释放。

视频信息获取

TXVideoInfoReader 的 getVideoFileInfo 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：

```
/**
* 获取视频信息
* @param videoPath 视频文件路径
* @return
*/
public TXVideoEditConstants.TXVideoInfo getVideoFileInfo(String videoPath);
```

返回的 TXVideoInfo 定义如下：

```
public final static class TXVideoInfo {
public Bitmap coverImage; // 视频首帧图片
public long duration; // 视频时长(ms)
public long fileSize; // 视频大小(byte)
public float fps; // 视频 fps
public int bitrate; // 视频码率 (kbps)
public int width; // 视频宽度
public int height; // 视频高度
public int audioSampleRate; // 音频码率
}
```

完整示例如下：

```
//sourcePath 为视频源路径
String sourcePath = Environment.getExternalStorageDirectory() + File.separator + "temp.mp4"
;
TXVideoEditConstants.TXVideoInfo info = TXVideoInfoReader.getInstance().getVideoFileInfo(sou
rcePath);
```

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。

按个数平分时间获取缩略图

快速导入生成精准缩略图

调用接口：

```
/**
* 获取缩略图列表
* @param count 缩略图张数
* @param width 缩略图宽度
* @param height 缩略图高度
* @param fast 缩略图是否关键帧的图片
* @param listener 缩略图的回调函数
*/
public void getThumbnail(int count, int width, int height, boolean fast, TXThumbnailListener list
ener)
```

参数 @param fast 可以使用两种模式：

- 快速出图：输出的缩略图速度比较快，但是与视频对应不精准，传入参数 true。
- 精准出图：输出的缩略图与视频时间点精准对应，但是在高分辨率上速度慢一些，传入参数 false。

完整示例：

```
mTXVideoEditer.getThumbnail(TCVideoEditerWrapper.mThumbnailCount, 100, 100, false, mThu
mbnailListener);
private TXVideoEditer.TXThumbnailListener mThumbnailListener = new TXVideoEditer.TXThum
bnailListener() {
@Override
public void onThumbnail(int index, long timeMs, final Bitmap bitmap) {
Log.i(TAG, "onThumbnail: index = " + index + ",timeMs:" + timeMs);
//将缩略图放入图片控件上
}
};
```

全功能导入获取缩略图

请参见 视频导入。

根据时间列表获取缩略图

```
List<Long> list = new ArrayList<>();
list.add(10000L);
```

```
list.add(12000L);
list.add(13000L);
list.add(14000L);
list.add(15000L);
TXVideoEditer txVideoEditer = new TXVideoEditer(TCVideoPreviewActivity.this);
txVideoEditer.setVideoPath(mVideoPath);
txVideoEditer.setThumbnailListener(new TXVideoEditer.TXThumbnailListener() {
@Override
public void onThumbnail(int index, long timeMs, Bitmap bitmap) {
Log.i(TAG, "bitmap:" + bitmap + ",timeMs:" + timeMs);
saveBitmap(bitmap, timeMs);
}
});
txVideoEditer.getThumbnailList(list, 200, 200);
```

⚠ 注意：

- List 中时间点不能超出视频总时长，对于超出总时长的返回最后一张图片。
- 设置的时间点单位是毫秒（ms）。

视频导入

快速导入

快速导入视频，可以直接观看到视频编辑的预览效果，支持视频裁剪、时间特效（慢动作）、滤镜特效、滤镜风格、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能，不支持的功能有时间特效（重复、倒放）。

全功能导入

全功能导入，支持所有的功能，包括时间特效（重复、倒放）。需要为视频先预处理操作。
经过全功能导入后的视频可以精确的 seek 到每个时间点，看到对应的画面，预处理操作同时还可以精确的生成当前时间点视频缩略图。

全功能导入步骤及调用接口如下：

1. 设置精确输出缩略图。

```
/**
* 设置预处理输出的缩略图
```

```
*/
public void setThumbnail(TXVideoEditConstants.TXThumbnail thumbnail)
```

2. 设置输出缩略图的回调。

```
/**
* 设置预处理输出缩略图回调
* @param listener
*/
public void setThumbnailListener(TXThumbnailListener listener)
```

> ⚠ 注意：
> 缩略图的宽高最好不要设置视频宽高，SDK 内部缩放效率更高。

3. 设置视频预处理回调接口。

```
/**
* 设置视频预处理回调
* @param listener
*/
public void setVideoProcessListener(TXVideoProcessListener listener)
```

4. 进行视频预处理。

```
public void processVideo();
```

完整示例如下：

```
int thumbnailCount = 10; //可以根据视频时长生成缩略图个数
TXVideoEditConstants.TXThumbnail thumbnail = new TXVideoEditConstants.TXThumbnail();
thumbnail.count = thumbnailCount;
thumbnail.width = 100; // 输出缩略图宽
thumbnail.height = 100; // 输出缩略图高
mTXVideoEditer.setThumbnail(thumbnail); // 设置预处理生成的缩略图
mTXVideoEditer.setThumbnailListener(mThumbnailListener); // 设置缩略图回调
```

```
mTXVideoEditer.setVideoProcessListener(this); // 视频预处理进度回调
mTXVideoEditer.processVideo(); // 进行预处理
```

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. 设置预览播放的 Layout

```
public void initWithPreview(TXVideoEditConstants.TXPreviewParam param)
```

设置预览 Layout 时，可以设置两种视频画面渲染模式，在 TXVideoEditConstants 常量中定义了这两种渲染模式

```
public final static int PREVIEW_RENDER_MODE_FILL_SCREEN = 1; // 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
public final static int PREVIEW_RENDER_MODE_FILL_EDGE = 2; // 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

经过 全功能导入 的视频可以精确预览到某一个时间点的视频画面。

```
public void previewAtTime(long timeMs);
```

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，从 startTime 到 endTime 的视频片段
public void startPlayFromTime(long startTime, long endTime);
```

4. 预览的暂停与恢复

```
// 暂停播放视频
public void pausePlay();
```

```
// 继续播放视频
public void resumePlay();

// 停止播放视频
public 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16种滤镜选择，同时也可以设置自定义的滤镜。

设置滤镜的方法：

```
void setFilter(Bitmap bmp)
```

其中 Bitmap 为滤镜映射图，bmp 设置为 null，会清除滤镜效果。

```
void setSpecialRatio(float specialRatio)
```

该接口可以调整滤镜程度值，一般为0.0 - 1.0。

```
void setFilter(Bitmap leftBitmap, float leftIntensity, Bitmap rightBitmap, float rightIntensity, floa
t leftRatio)
```

该接口能够实现组合滤镜，即左右可以添加不同的滤镜。leftBitmap 为左侧滤镜、leftIntensity 为左侧滤镜程度值；rightBitmap 为右侧滤镜、rightIntensity 为右侧滤镜程度值；leftRatio 为左侧滤镜所占的比例，一般为0.0 - 1.0。当 leftBitmap 或 rightBitmap 为 null，则该侧清除滤镜效果。

6. 水印

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。

设置水印方法：

```
public void setWaterMark(Bitmap waterMark, TXVideoEditConstants.TXRect rect);
```

其中 waterMark 表示水印图片，rect 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：

```
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXVideoEditer.setWaterMark(mWaterMarkLogo, rect);
```

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：

```
setTailWaterMark(Bitmap tailWaterMark, TXVideoEditConstants.TXRect txRect, int duration);
```

其中 tailWaterMark 表示片尾水印图片，txRect 是相对于视频图像的归一化 txRect，txRect 的 x、y、width 取值范围都为0 – 1，duration 为水印的持续时长，单位：秒。

Demo 实例：

设置水印在片尾中间，持续3秒。

```
Bitmap tailWaterMarkBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.tclo
ud_logo);
TXVideoEditConstants.TXRect txRect = new TXVideoEditConstants.TXRect();
txRect.x = (mTXVideoInfo.width - tailWaterMarkBitmap.getWidth()) / (2f * mTXVideoInfo.width);
txRect.y = (mTXVideoInfo.height - tailWaterMarkBitmap.getHeight()) / (2f * mTXVideoInfo.heigh
t);
txRect.width = tailWaterMarkBitmap.getWidth() / (float) mTXVideoInfo.width;
mTXVideoEditer.setTailWaterMark(tailWaterMarkBitmap, txRect, 3);
```

压缩裁剪

视频码率设置

目前支持自定义视频的码率，这里建议设置的范围600 – 12000kbps，如果设置了这个码率，SDK 最终压缩视频时会优先选取这个码率，注意码率不要太大或太小，码率太大，视频的体积会很大，码率太小，视频会模糊不清。

```
public void setVideoBitrate(int videoBitrate);
```

视频裁剪

设置视频裁剪的开始时间和结束时间。

```
/**
* 设置视频剪切范围
* @param startTime 视频剪切的开始时间(ms)
* @param endTime 视频剪切的结束时间(ms)
*/
public void setCutFromTime(long startTime, long endTime)

// ...
// 生成最终的视频文件
public void generateVideo(int videoCompressed, String videoOutputPath)
```

参数 videoCompressed 在 TXVideoEditConstants 中可选常量。

```
VIDEO_COMPRESSED_360P ——压缩至360P分辨率（360*640）
VIDEO_COMPRESSED_480P ——压缩至480P分辨率（640*480）
VIDEO_COMPRESSED_540P ——压缩至540P分辨率 (960*540)
VIDEO_COMPRESSED_720P ——压缩至720P分辨率 (1280*720)
```

如果源视频的分辨率小于设置的常量对应的分辨率，按照原视频的分辨率。
如果源视频的分辨率大于设置的常量对象的分辨率，进行视频压缩至相应分辨率。

资源释放

当您不再使用 mTXVideoEditer 对象时，一定要记得调用 **release()** 释放它。

高级功能

- 类抖音特效

- 设置背景音乐
- 贴纸字幕
- 图片转场特效

视频拼接
iOS

最近更新时间：2021-12-20 16:24:29

版本支持

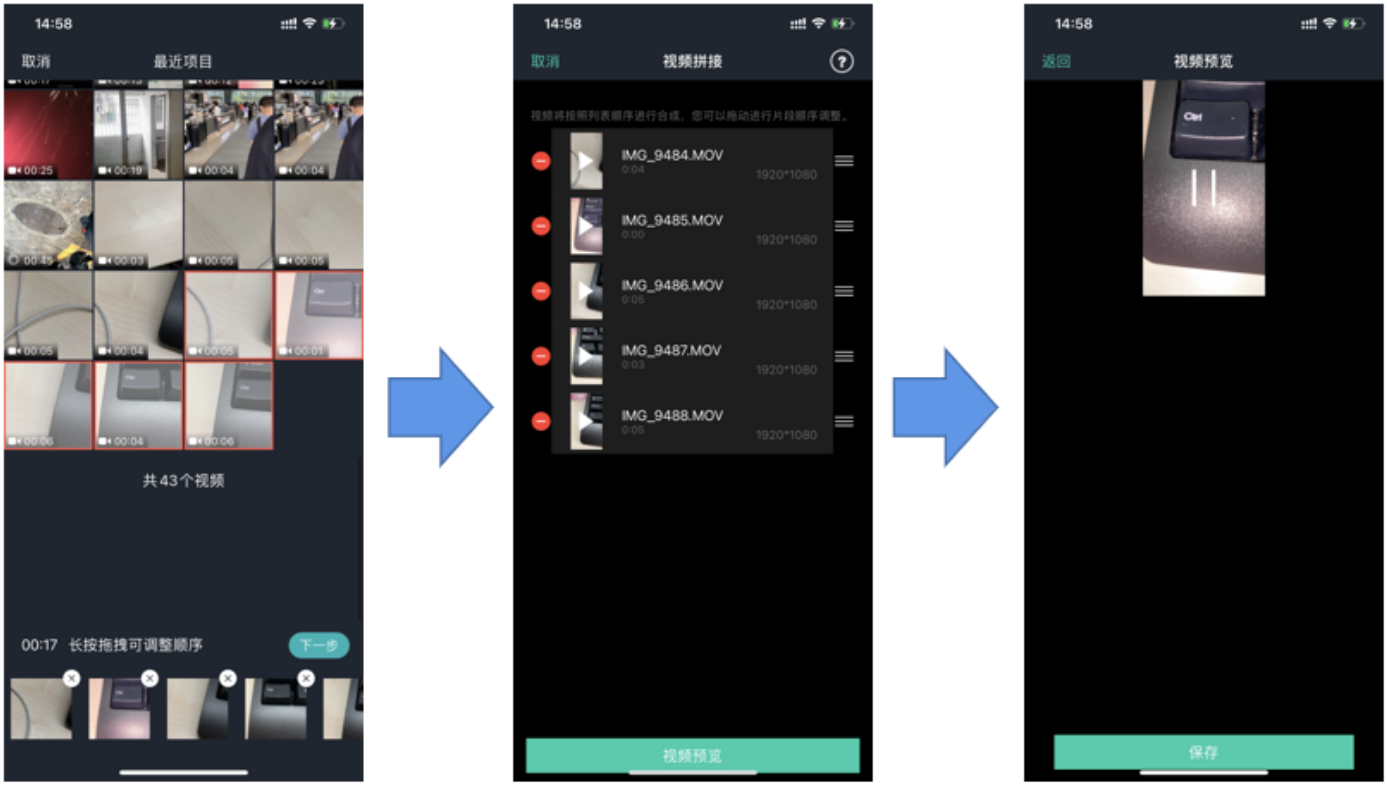
本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。

Android

最近更新时间：2021-08-12 19:56:28

版本支持

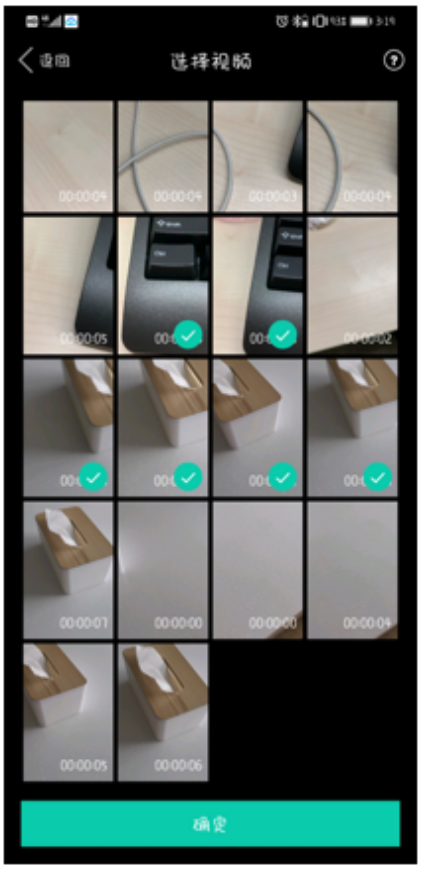
本页文档所描述功能，在腾讯云视立方中支持情况如下：

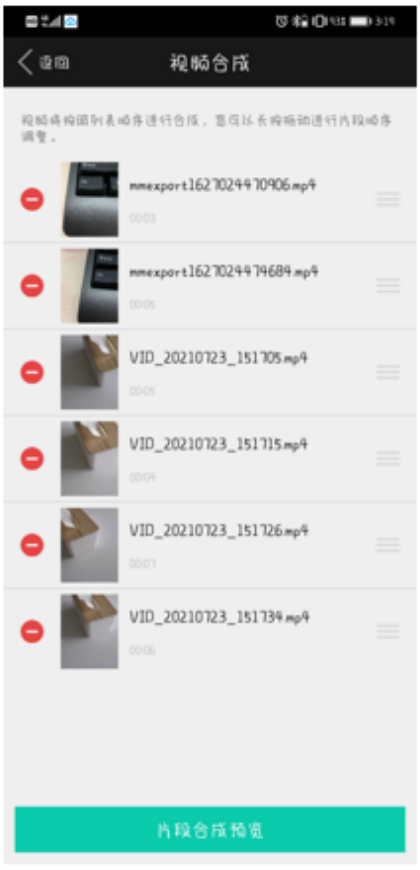
版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

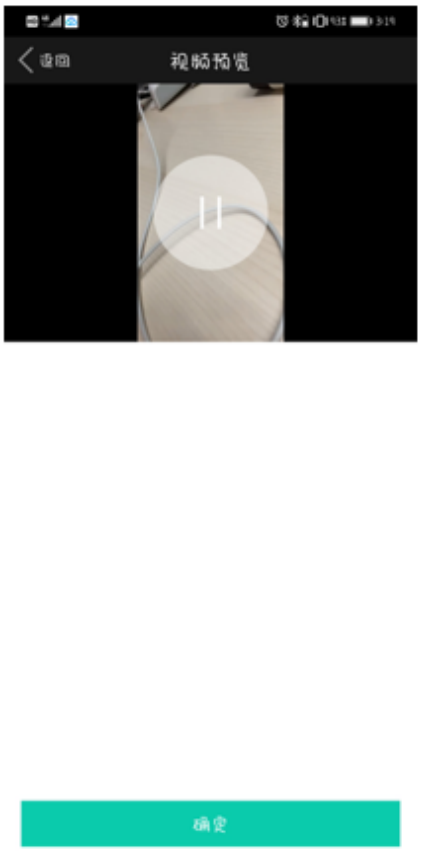
不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

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。







- TCVideoJoinerActivity 用于实现上图中的视频拼接列表，支持上下拖拽调整顺序。
- TCVideoJoinerPreviewActivity 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，决心自己实现 UI 部分，则可以参考如下的攻略进行对接：

1. 选择视频文件

自己实现多选文件功能。

2. 设置预览 View

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

```
//准备预览 View
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreviewPara
m();
param.videoView = mVideoView;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;

// 创建 TXUGCJoiner 对象并设置预览 view
TXVideoJoiner mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setTXVideoPreviewListener(this);
mTXVideoJoiner.initWithPreview(param);
// 设置待拼接的视频文件组 mVideoSourceList，也就是第一步中选择的若干个文件
mTXVideoJoiner.setVideoPathList(mVideoSourceList);
```

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：

```
mTXVideoJoiner.setVideoJoinerListener(this);
mTXVideoJoiner.joinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mVideoOutputPath);
```

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 TXVideoJoiner.TXVideoJoinerListener **以回调的形式通知用户。**

短视频上传和播放
签名派发

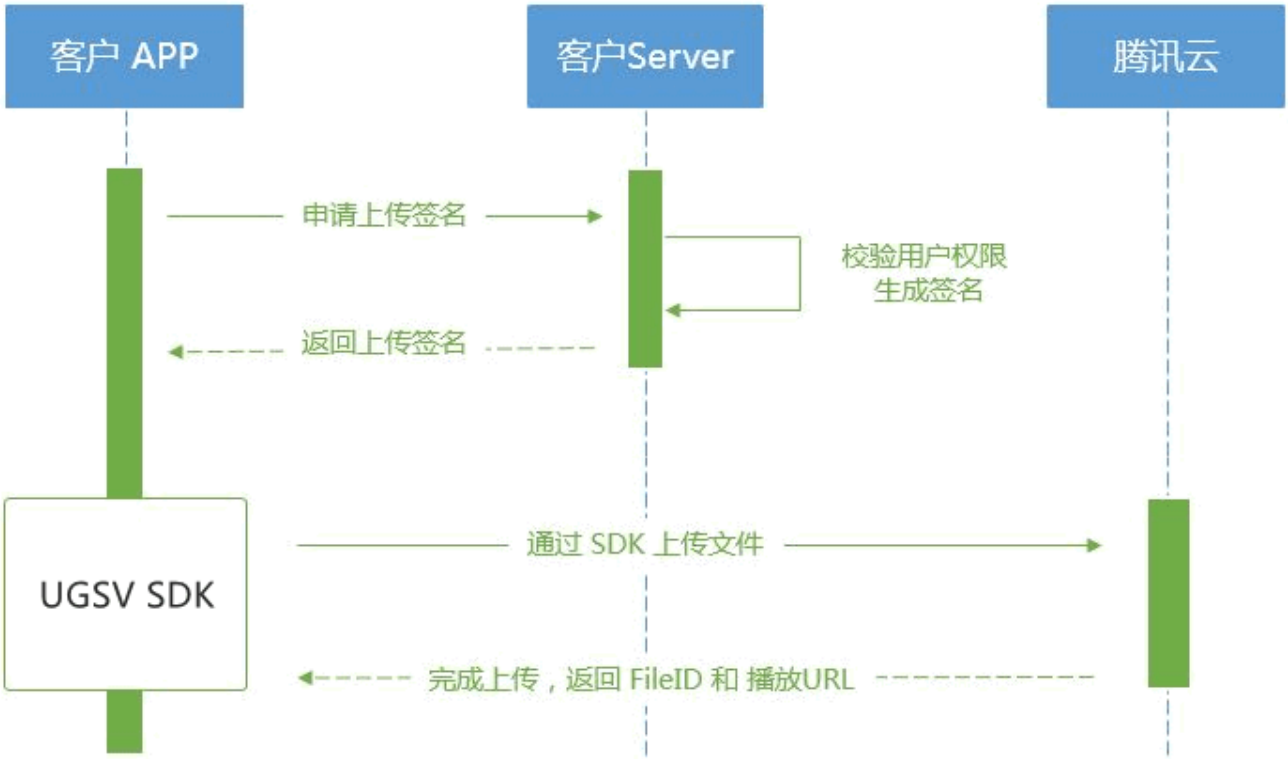
最近更新时间：2021-08-12 19:56:34

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腾讯云视立方短视频 UGSV 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({})
}
});
}
```

iOS

最近更新时间：2021-08-12 19:56:38

版本支持

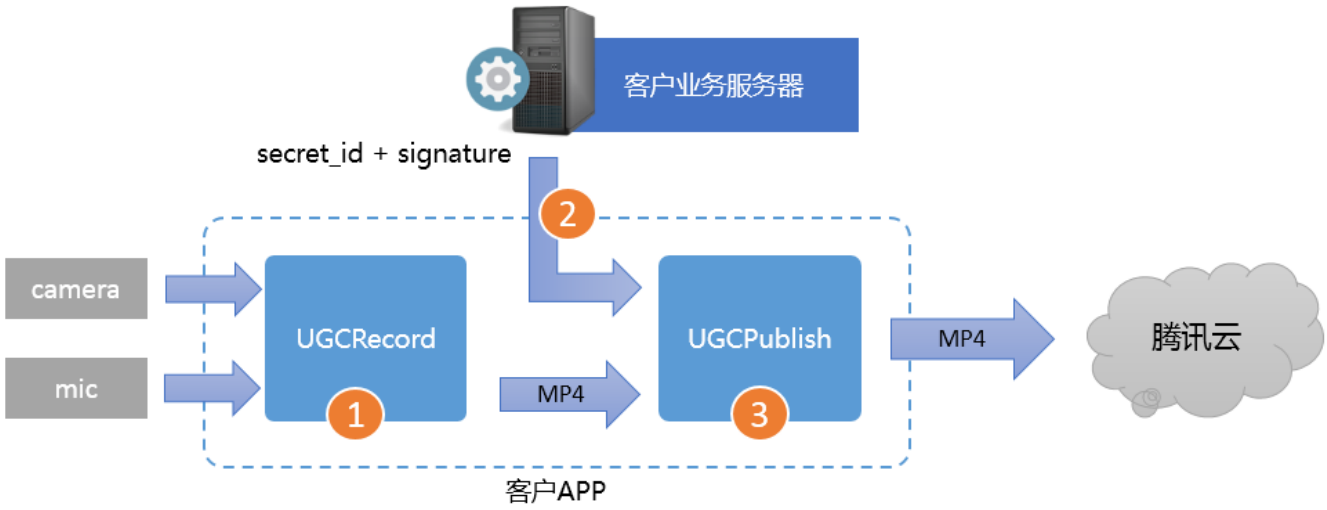
本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

对接流程

短视频发布是将 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。



1. 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。
2. 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。
3. 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。

注意事项

- 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。

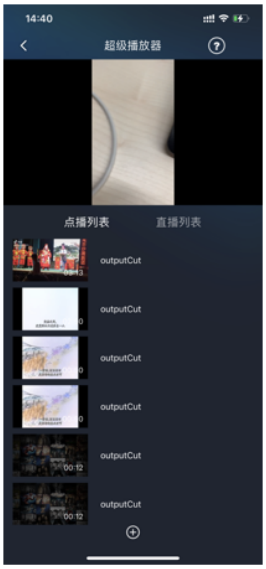
对接攻略

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，详情请参件 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，您可以通过此 ID 调用点播 服务端 API 接口。

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

- 通过 编辑错误码表 和 录制错误码表 确认短视频的发布结果。

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。

Android

最近更新时间：2021-08-12 19:56:41

版本支持

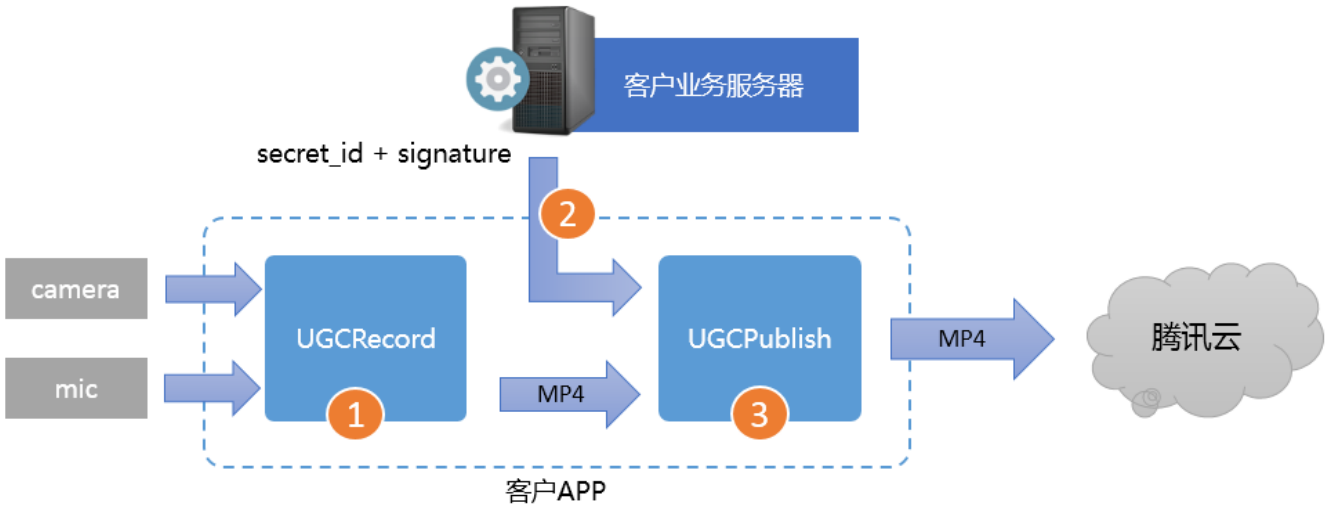
本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

对接流程

短视频发布是将 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。



1. 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。
2. 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。
3. 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。

注意事项

- 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。

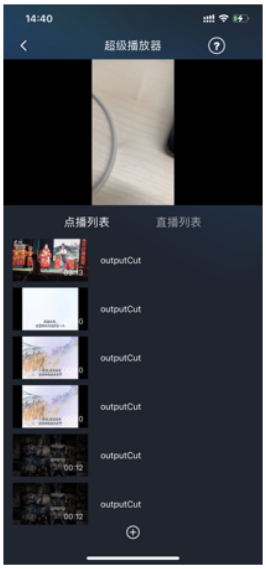
对接攻略

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，详情请参件 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，您可以通过此 ID 调用点播 服务端 API 接口。

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

- 通过 编辑错误码表 和 录制错误码表 确认短视频的发布结果。

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。

短视频特效
类抖音特效
iOS

最近更新时间：2021-08-12 19:56:46

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置方法

```
- (void) startEffect:(TXEffectType)type startTime:(float)startTime;
- (void) stopEffect:(TXEffectType)type endTime:(float)endTime;
//特效的类型（type 参数），在常量 TXEffectType 中有定义：
typedef NS_ENUM(NSInteger,TXEffectType)
{
TXEffectType_ROCK_LIGHT, //动感光波
TXEffectType_DARK_DRAEM, //暗黑幻境
TXEffectType_SOUL_OUT, //灵魂出窍
TXEffectType_SCREEN_SPLIT,//视频分裂
TXEffectType_WIN_SHADOW, //百叶窗
TXEffectType_GHOST_SHADOW,//鬼影
TXEffectType_PHANTOM, //幻影
```

```
TXEffectType_GHOST, //幽灵
TXEffectType_LIGHTNING, //闪电
TXEffectType_MIRROR, //镜像
TXEffectType_ILLUSION, //幻觉
};
- (void) deleteLastEffect;
- (void) deleteAllEffect;
```

- 调用 deleteLastEffect() 删除最后一次设置的滤镜特效。
- 调用 deleteAllEffect() 删除所有设置的滤镜特效。

Demo 示例

在1s – 2s之间应用第一种滤镜特效，在3s – 4s之间应用第2种滤镜特效，删除3s – 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
[_ugcEdit startEffect:TXEffectType_SOUL_OUT startTime:1.0];
[_ugcEdit stopEffect:TXEffectType_SOUL_OUT startTime:2.0)];
//在3-4s之间应用第2种滤镜特效
[_ugcEdit startEffect:TXEffectType_SPLIT_SCREEN startTime:3.0];
[_ugcEdit stopEffect:TXEffectType_SPLIT_SCREEN startTime:4.0];
//删除3-4s设置的滤镜特效
[_ugcEdit deleteLastEffect];
```

慢/快动作

您可以进行多段视频的慢速/快速播放。

设置方法

```
- (void) setSpeedList:(NSArray *)speedList;

//TXSpeed 的参数如下：
@interface TXSpeed: NSObject
@property (nonatomic, assign) CGFloat startTime; //加速播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //加速播放结束时间(s)
@property (nonatomic, assign) TXSpeedLevel speedLevel; //加速级别
@end
```

```
// 目前支持变速速度的几种级别，在常量 TXSpeedLevel 中有定义：
typedef NS_ENUM(NSInteger, TXSpeedLevel) {
SPEED_LEVEL_SLOWEST, //极慢速
SPEED_LEVEL_SLOW, //慢速
SPEED_LEVEL_NOMAL, //正常速
SPEED_LEVEL_FAST, //快速
SPEED_LEVEL_FASTEST, //极快速
};
```

Demo 示例

```
// SDK 拥有支持多段变速的功能。 此 Demo 仅展示一段慢速播放
TXSpeed *speed =[[TXSpeed alloc] init];
speed.startTime = 1.0;
speed.endTime = 3.0;
speed.speedLevel = SPEED_LEVEL_SLOW;
[_ugcEdit setSpeedList:@[speed]];
```

倒放

您可以将视频画面倒序播放。

设置方法

```
- (void) setReverse:(BOOL)isReverse;
```

Demo 示例

```
[_ugcEdit setReverse:YES];
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。

设置方法

```
- (void) setRepeatPlay:(NSArray *)repeatList;

//TXRepeat 的参数如下：
@interface TXRepeat: NSObject
@property (nonatomic, assign) CGFloat startTime; //重复播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //重复播放结束时间(s)
@property (nonatomic, assign) int repeatTimes; //重复播放次数
@end
```

Demo 示例

```
TXRepeat *repeat = [[TXRepeat alloc] init];
repeat.startTime = 1.0;
repeat.endTime = 3.0;
repeat.repeatTimes = 3; //重复次数
[_ugcEdit setRepeatPlay:@[repeat]];
```

Android

最近更新时间：2021-08-12 19:56:50

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	-	✓	-	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置方法

```
/**
* 设置滤镜特效开始时间
* @param type 滤镜特效类型
* @param startTime 滤镜特效开始时间（ms）
*/
public void startEffect(int type, long startTime);
/**
* 设置滤镜特效结束时间
* @param type 滤镜特效类型
* @param endTime 滤镜特效结束时间（ms）
*/
public void stopEffect(int type, long endTime);
```

参数说明：@param type 是滤镜特效的类型，在常量 TXVideoEditConstants 中有定义：

```
public static final int TXEffectType_SOUL_OUT = 0; //滤镜效果1
public static final int TXEffectType_SPLIT_SCREEN = 1; //滤镜效果2
public static final int TXEffectType_DARK_DRAEM = 2; //滤镜效果3
public static final int TXEffectType_ROCK_LIGHT = 3; //滤镜效果4
public static final int TXEffectType_WIN_SHADDOW = 4; //滤镜效果5
public static final int TXEffectType_GHOST_SHADDOW = 5; //滤镜效果6
public static final int TXEffectType_PHANTOM_SHADDOW = 6; //滤镜效果7
public static final int TXEffectType_GHOST = 7; //滤镜效果8
public static final int TXEffectType_LIGHTNING = 8; //滤镜效果9
public static final int TXEffectType_MIRROR = 9; //滤镜效果10
public static final int TXEffectType_ILLUSION = 10; //滤镜效果11
```

删除最后一个设置的滤镜特效：

```
public void deleteLastEffect();
```

删除所有设置的滤镜特效：

```
public void deleteAllEffect();
```

Demo 示例

在1s – 2s之间应用第一种滤镜特效，在3s – 4s之间应用第2种滤镜特效，删除3s – 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 1000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 2000);
//在3-4s之间应用第2种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 3000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 4000);
//删除3-4s设置的滤镜特效
mTXVideoEditer.deleteLastEffect();
```

慢/快动作

您可以进行多段视频的慢速/快速播放。

设置方法

```
public void setSpeedList(List speedList);

//TXSpeed 的参数如下：
public final static class TXSpeed {
public int speedLevel; // 变速级别
public long startTime; // 开始时间
public long endTime; // 结束时间
}

// 目前支持变速速度的几种级别，在常量 TXVideoEditConstants 中有定义：
SPEED_LEVEL_SLOWEST - 极慢
SPEED_LEVEL_SLOW - 慢
SPEED_LEVEL_NORMAL - 正常
SPEED_LEVEL_FAST - 快
SPEED_LEVEL_FASTEST - 极快
```

Demo 示例

```
List<TXVideoEditConstants.TXSpeed> list = new ArrayList<>();
TXVideoEditConstants.TXSpeed speed1 = new TXVideoEditConstants.TXSpeed();
speed1.startTime = 0;
speed1.endTime = 1000;
speed1.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; // 慢速
list.add(speed1);

TXVideoEditConstants.TXSpeed speed2 = new TXVideoEditConstants.TXSpeed();
speed2.startTime = 1000;
speed2.endTime = 2000;
speed2.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOWEST; // 极慢速
list.add(speed2);

TXVideoEditConstants.TXSpeed speed3 = new TXVideoEditConstants.TXSpeed();
speed3.startTime = 2000;
speed3.endTime = 3000;
speed3.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; //慢速
list.add(speed3);
```

版权所有：腾讯云计算（北京）有限责任公司

```
mTXVideoEditer.setSpeedList(list);
```

倒放

您可以将视频画面倒序播放。通过调用 **setReverse(true)** 开启倒序播放，调用 **setReverse(false)** 停止倒序播放。

> ⚠ **注意：**
> **setTXVideoReverseListener()** 老版本（SDK 4.5以前）首次监听需要手动调用，新版本不需要调用即可生效。

Demo 示例

```
mTXVideoEditer.setTXVideoReverseListener(mTxVideoReverseListener);
mTXVideoEditer.setReverse(true);
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。目前 Android 只支持设置一段画面重复，重复三次。如需取消之前设置的重复片段，调用 **setRepeatPlay(null)** 即可。

设置方法

```
public void setRepeatPlay(List repeatList);

//TXRepeat 的参数如下：
public final static class TXRepeat {
public long startTime; //重复播放起始时间(ms)
public long endTime; //重复播放结束时间(ms)
public int repeatTimes; //重复播放次数
}
```

Demo 示例

```
long currentPts = mVideoProgressController.getCurrentTimeMs();
```

```
List repeatList = new ArrayList<>();
TXVideoEditConstants.TXRepeat repeat = new TXVideoEditConstants.TXRepeat();
repeat.startTime = currentPts;
repeat.endTime = currentPts + DEAULT_DURATION_MS;
repeat.repeatTimes = 3; //目前只支持重复三次
repeatList.add(repeat); //目前只支持重复一段时间
mTXVideoEditer.setRepeatPlay(repeatList);
```

贴纸和字幕
iOS

最近更新时间：2021-08-12 19:56:56

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

静态贴纸

```
- (void) setPasterList:(NSArray *)pasterList;

// TXPaster 的参数如下：
@interface TXPaster: NSObject
@property (nonatomic, strong) UIImage* pasterImage; //贴纸图片
@property (nonatomic, assign) CGRect frame; //贴纸 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //贴纸起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //贴纸结束时间(s)
@end
```

动态贴纸

设置方法

```
- (void) setAnimatedPasterList:(NSArray *)animatedPasterList;

// TXAnimatedPaster 的参数如下：
```

```
@interface TXAnimatedPaster: NSObject
@property (nonatomic, strong) NSString* animatedPasterpath; //动图文件路径
@property (nonatomic, assign) CGRect frame; //动图的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat rotateAngle; //动图旋转角度 (0 ~ 360)
@property (nonatomic, assign) CGFloat startTime; //动图起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //动图结束时间(s)
@end
```

Demo 示例

```
- (void)setVideoPasters:(NSArray*)videoPasterInfos
{
NSMutableArray* animatePasters = [NSMutableArray new];
NSMutableArray* staticPasters = [NSMutableArray new];
for (VideoPasterInfo* pasterInfo in videoPasterInfos) {
if (pasterInfo.pasterInfoType == PasterInfoType_Animate) {
TXAnimatedPaster* paster = [TXAnimatedPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.rotateAngle = pasterInfo.pasterView.rotateAngle * 180 / M_PI;
paster.animatedPasterpath = pasterInfo.path;
[animatePasters addObject:paster];
}
else if (pasterInfo.pasterInfoType == PasterInfoType_static){
TXPaster *paster = [TXPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.pasterImage = pasterInfo.pasterView.staticImage;
[staticPasters addObject:paster];
}
}
[_ugcEditer setAnimatedPasterList:animatePasters];
[_ugcEditer setPasterList:staticPasters];
}
```

自定义动态贴纸

动态贴纸的本质是：将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

自定义贴纸

以工具包 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK 内部将获取到该动态贴纸对应的 config.json ，并且按照 JSON 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

气泡字幕

您可以为视频添加字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置方法

```
- (void) setSubtitleList:(NSArray *)subtitleList;

TXSubtitle 的参数如下：
@interface TXSubtitle: NSObject
@property (nonatomic, strong) UIImage* titleImage; //字幕图片 （这里需要客户把承载文字的控件转成 image 图片）
@property (nonatomic, assign) CGRect frame; //字幕的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //字幕起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //字幕结束时间(s)
@end
```

参数	说明
titleImage	表示字幕图片，如果上层使用的是 UILabel 之类的控件，请先把控件转成 UIImage，具体方法可以参照 Demo 的示例代码。
frame	表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 Demo 的示例代码。
startTime	字幕作用的起始时间。
endTime	字幕作用的结束时间。

因为字幕这一块的 UI 逻辑比较复杂，我们已经在 Demo 层有一整套的实现方法，推荐客户直接参考 Demo 实现， 可以大大降低您的接入成本。

Demo 示例

```
@interface VideoTextInfo : NSObject
@property (nonatomic, strong) VideoTextFiled* textField;
@property (nonatomic, assign) CGFloat startTime; //in seconds
@property (nonatomic, assign) CGFloat endTime;
@end

videoTextInfos = @[VideoTextInfo1, VideoTextInfo2 ...];

for (VideoTextInfo* textInfo in videoTextInfos) {
TXSubtitle* subtitle = [TXSubtitle new];
subtitle.titleImage = textInfo.textField.textImage; //UILabel（UIView） -> UIImage
subtitle.frame = [textInfo.textField textFrameOnView:_videoPreview]; //计算相对于渲染 view 的
```

```
坐标
subtitle.startTime = textInfo.startTime; //字幕起始时间
subtitle.endTime = textInfo.endTime; //字幕结束时间
[subtitles addObject:subtitle]; //添加字幕列表
}

[_ugcEditer setSubtitleList:subtitles]; //设置字幕列表
```

自定义气泡字幕

气泡字幕所需要参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，建议您在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 JSON 格式封装：

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

> 说明：
> 该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

我们的 Demo 提供了一个自动排版的控件，在当前字体大小下，且字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。使用该控件即可**实现字幕若输入过长情况下，通过排版使字幕与气泡美观地合并。**
您也可以修改相关控件源代码，来满足自身的业务要求。

Android

最近更新时间：2021-08-12 19:56:59

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

静态贴纸

设置方法

```
public void setPasterList(List pasterList);

// TXPaster 的参数如下：
public final static class TXPaster {
public Bitmap pasterImage; // 贴纸图片
public TXRect frame; // 贴纸的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
public long startTime; // 贴纸起始时间(ms)
public long endTime; // 贴纸结束时间(ms)
}
```

动态贴纸

设置方法

```
public void setAnimatedPasterList(List animatedPasterList);

// TXAnimatedPaster 的参数如下：
public final static class TXAnimatedPaster {
public String animatedPasterPathFolder; // 动态贴纸图片地址
```

```
public TXRect frame; // 动态贴纸 frame (注意这里的 frame 坐标是相对于渲染 view 的坐标 )
public long startTime; // 动态贴纸起始时间(ms)
public long endTime; // 动态贴纸结束时间(ms)
public float rotation;
}
```

Demo 示例

```
List animatedPasterList = new ArrayList<>();
List pasterList = new ArrayList<>();
for (int i = 0; i < mTCLayerViewGroup.getChildCount(); i++) {
PasterOperationView view = (PasterOperationView) mTCLayerViewGroup.getOperationView
(i);
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX();
rect.y = view.getImageY();
rect.width = view.getImageWidth();
TXCLog.i(TAG, "addPasterListVideo, adjustPasterRect, paster x y = " + rect.x + "," + rect.y);

int childType = view.getChildType();
if (childType == PasterOperationView.TYPE_CHILD_VIEW_ANIMATED_PASTER) {
TXVideoEditConstants.TXAnimatedPaster txAnimatedPaster = new TXVideoEditConstants.TX
AnimatedPaster();

txAnimatedPaster.animatedPasterPathFolder = mAnimatedPasterSDcardFolder + view.getPa
sterName() + File.separator;
txAnimatedPaster.startTime = view.getStartTime();
txAnimatedPaster.endTime = view.getEndTime();
txAnimatedPaster.frame = rect;
txAnimatedPaster.rotation = view.getImageRotate();

animatedPasterList.add(txAnimatedPaster);
TXCLog.i(TAG, "addPasterListVideo, txAnimatedPaster startTimeMs, endTime is : " + txAnima
tedPaster.startTime + ", " + txAnimatedPaster.endTime);
} else if (childType == PasterOperationView.TYPE_CHILD_VIEW_PASTER) {
TXVideoEditConstants.TXPaster txPaster = new TXVideoEditConstants.TXPaster();

txPaster.pasterImage = view.getRotateBitmap();
txPaster.startTime = view.getStartTime();
```

```
txPaster.endTime = view.getEndTime();
txPaster.frame = rect;

pasterList.add(txPaster);
TXCLog.i(TAG, "addPasterListVideo, txPaster startTimeMs, endTime is : " + txPaster.startTime
+ ", " + txPaster.endTime);
}
}

mTXVideoEditer.setAnimatedPasterList(animatedPasterList); //设置动态贴纸
mTXVideoEditer.setPasterList(pasterList); //设置静态贴纸
```

自定义动态贴纸

动态贴纸的本质是将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

封装格式

以 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK 内部将获取到该动态贴纸对应的 config.json ，并且按照 JSON 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

气泡字幕

您可以为视频设置气泡字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置方法

```
public void setSubtitleList(List subtitleList);

//TXSubtitle 的参数如下：
public final static class TXSubtitle {
public Bitmap titleImage; // 字幕图片
public TXRect frame; // 字幕的 frame
public long startTime; // 字幕起始时间(ms)
public long endTime; // 字幕结束时间(ms)
}

public final static class TXRect {
public float x;
public float y;
public float width;
}
```

参数	说明
titleImage	表示字幕图片，如果上层使用的是 TextView 之类的控件，请先把控件转成 Bitmap，具体方法可以参照 Demo 的示例代码。
frame	表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 Demo 的示例代码。
startTime	字幕作用的起始时间。
endTime	字幕作用的结束时间。

因为字幕的 UI 逻辑比较复杂，我们已经在 Demo 层有一整套的实现方法，推荐客户直接参考 Demo 实现， 可以大大降低您的接入成本。

Demo 示例

```
mSubtitleList.clear();
for (int i = 0; i < mWordInfoList.size(); i++) {
TCWordOperationView view = mOperationViewGroup.getOperationView(i);
TXVideoEditConstants.TXSubtitle subTitle = new TXVideoEditConstants.TXSubtitle();
subTitle.titleImage = view.getRotateBitmap(); //获取Bitmap
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX(); // 获取相对 parent view 的 x 坐标
rect.y = view.getImageY(); // 获取相对 parent view 的 y 坐标
rect.width = view.getImageWidth(); // 图片宽度
subTitle.frame = rect;
subTitle.startTime = mWordInfoList.get(i).getStartTime(); // 设置开始时间
subTitle.endTime = mWordInfoList.get(i).getEndTime(); // 设置结束时间
mSubtitleList.add(subTitle);
}
mTXVideoEditer.setSubtitleList(mSubtitleList); // 设置字幕列表
```

自定义气泡字幕

气泡字幕所需要参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，建议您在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 JSON 格式封装。

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
```

```
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

> 说明：
> 该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

我们的 Demo 提供了一个自动排版的控件，在当前字体大小下，且字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。使用该控件即可**实现字幕若输入过长情况下，通过排版使字幕与气泡美观地合并。**
您也可以修改相关控件源代码，来满足自身的业务要求。

版权所有：腾讯云计算（北京）有限责任公司

视频合唱
iOS

最近更新时间：2021-08-12 19:57:02

本篇教程向大家介绍如何从零开始完成合唱的基础功能。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

过程简介

1. 在界面上放两个 View， 一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

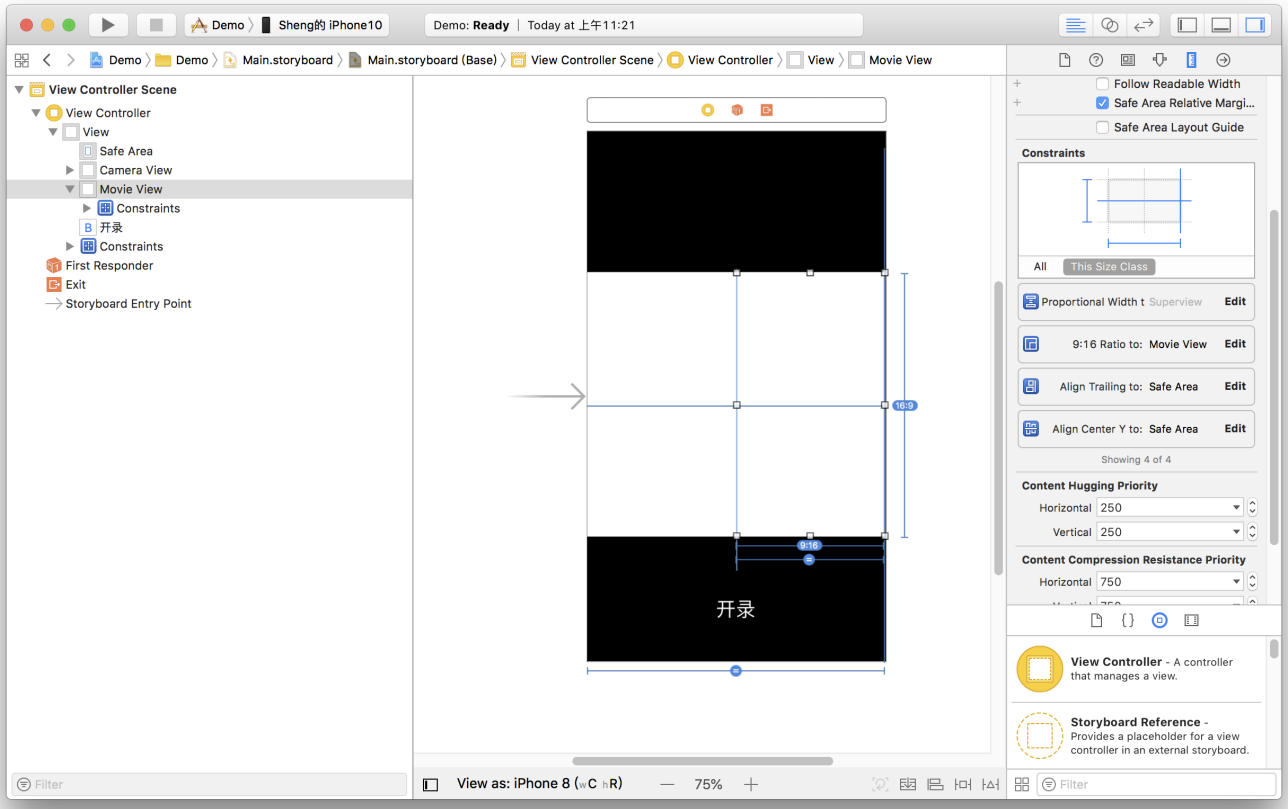
界面搭建

1. **开始工程的创建**：打开 Xcode，【File】>【New】>【Project】，然后起好工程名创建工程（后续称为 Demo）。因为要录像，所以需要相机和麦克风的权限，在 Info 中配置一下增加以下两项：

```
Privacy - Microphone Usage Description
Privacy - Camera Usage Description
```

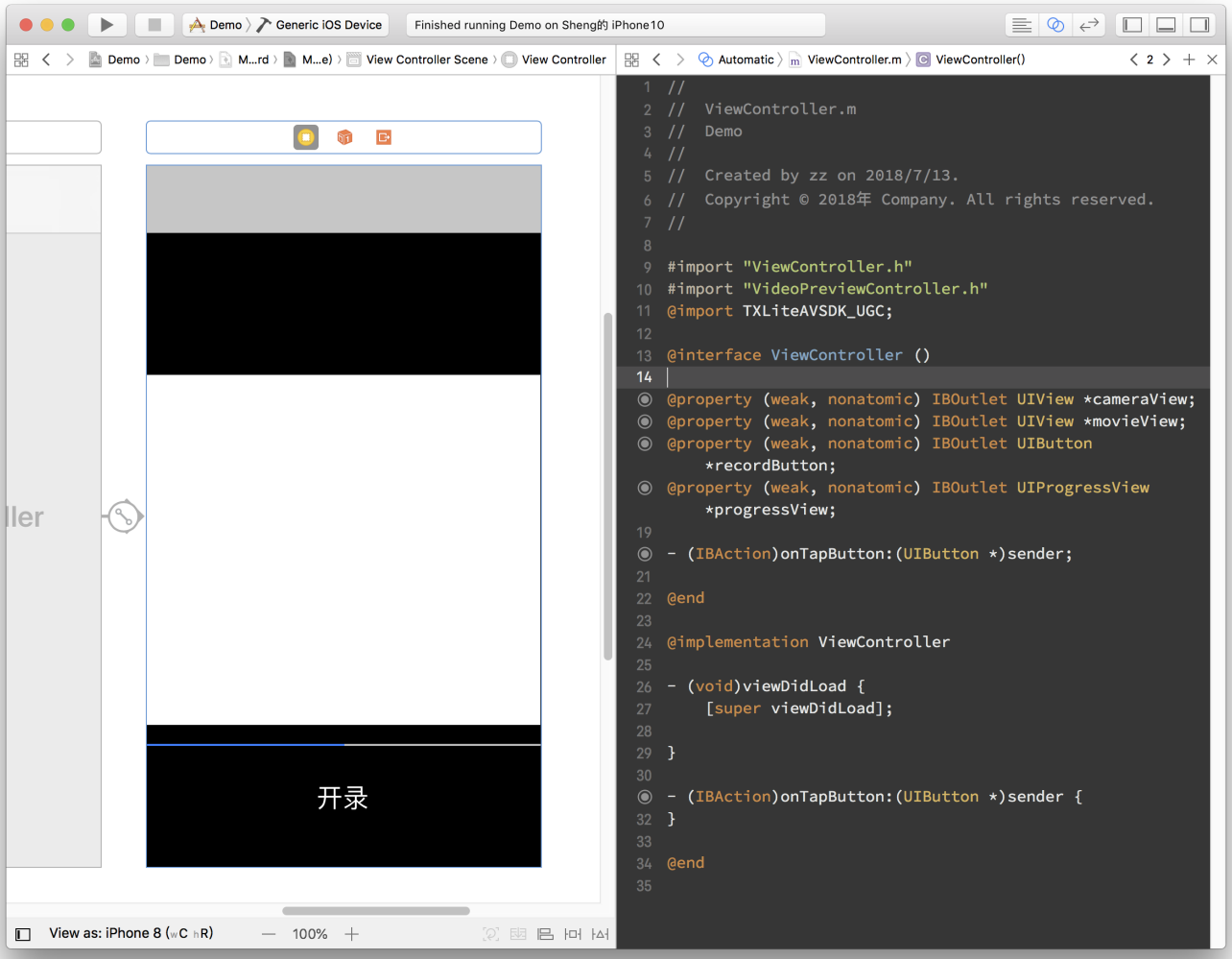
> **说明：**
> 该配置项可自定义，例如"录制视频"。

2. **配置一个简单的录制界面：**打开 Main.storyboard，拖进去两个 UIView，配置宽度为 superview 的0.5倍，长宽比16:9。



3. **加上进度条：**在 ViewController.m 中设置 IBOutlet 绑定界面，并设置好按钮的 IBAction。因为录制好后还要跳转到预览界面，还需要一个导航。单击黄色 VC 图标，在菜单栏依次进入【Editor】>【Embeded In】，单击【Navigation Controller】给 ViewController 套一层 Navigation Controller。完成基本 UI 的搭

建。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为： TXVideoEditer、TXUGCRecord、TXVideoJoiner。

1. 配置 License

在使用前要配置腾讯云视立方短视频 UGSV SDK 的 Licence，打开 AppDelegate.m 在里面添加以下代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
[TXUGCBase setLicenceURL:@"<Licence的URL>" key:@"<Licence的Key>"];
return YES;
}
```

> 说明：
> Licence 参数需要到 视立方控制台 去申请。申请审核通过后，页面上即可出现相关的 Licence 信息。

2. 声明与初始化

1. 打开 ViewContorller.m，引用 SDK 并声明上述三个类的实例。另外这里播放、录制和合成视频都是异步操作，需要监听他们的事件，所以要加上实现 TXVideoJoinerListener、TXUGCRecordListener、TXVideoPreviewListener 这三个协议的声明。加好后如下所示：

```
#import "ViewController.h"
@import TXLiteAVSDK_UGC;
@interface ViewController () <TXVideoJoinerListener, TXUGCRecordListener, TXVideoPrevi
ewListener>
{
TXVideoEditer *_editor;
TXUGCRecord *_recorder;
TXVideoJoiner *_joiner;
TXVideoInfo *_videoInfo;

NSString *_recordPath;
NSString *_resultPath;
}
@property (weak, nonatomic) IBOutlet UIView *cameraView;
@property (weak, nonatomic) IBOutlet UIView *movieView;
@property (weak, nonatomic) IBOutlet UIButton *recordButton;
@property (weak, nonatomic) IBOutlet UIProgressView *progressView;
- (IBAction)onTapButton:(UIButton *)sender;
@end
```

2. 准备好成员变量和接口实现声明后，我们在 viewDidLoad 中对上面的成员变量进行初始化。

```
- (void)viewDidLoad {
[super viewDidLoad];
// 这里找一段 mp4 视频放到了工程里，或者用手机录制的 mov 格式视频也可以
NSString *mp4Path = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"mp
4"];
_videoInfo = [TXVideoInfoReader getVideoInfo:mp4Path];
TXAudioSampleRate audioSampleRate = AUDIO_SAMPLERATE_48000;
if (_videoInfo.audioSampleRate == 8000) {
```

```
audioSampleRate = AUDIO_SAMPLERATE_8000;
}else if (_videoInfo.audioSampleRate == 16000){
audioSampleRate = AUDIO_SAMPLERATE_16000;
}else if (_videoInfo.audioSampleRate == 32000){
audioSampleRate = AUDIO_SAMPLERATE_32000;
}else if (_videoInfo.audioSampleRate == 44100){
audioSampleRate = AUDIO_SAMPLERATE_44100;
}else if (_videoInfo.audioSampleRate == 48000){
audioSampleRate = AUDIO_SAMPLERATE_48000;
}

// 设置录像的保存路径
_recordPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"record.mp
4"];
_resultPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"result.mp
4"];

// 播放器初始化
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = self.movieView;
param.renderMode = RENDER_MODE_FILL_EDGE;
_editor = [[TXVideoEditer alloc] initWithPreview:param];
[_editor setVideoPath:mp4Path];
_editor.previewDelegate = self;

// 录像参数初始化
_recorder = [TXUGCRecord shareInstance];
TXUGCCustomConfig *recordConfig = [[TXUGCCustomConfig alloc] init];
recordConfig.videoResolution = VIDEO_RESOLUTION_720_1280;
//这里保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象
//注意：这里获取的合唱视频的帧率是平均帧率，有可能为小数，做一下四舍五入操作
recordConfig.videoFPS = (int)(_videoInfo.fps + 0.5);
//这里保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象
recordConfig.audioSampleRate = audioSampleRate;
recordConfig.videoBitratePIN = 9600;
recordConfig.maxDuration = _videoInfo.duration;
_recorder.recordDelegate = self;

// 启动相机预览
[_recorder startCameraCustom:recordConfig preview:self.cameraView];
```

```
// 视频拼接
_joiner = [[TXVideoJoiner alloc] initWithPreview:nil];
_joiner.joinerDelegate = self;
[_joiner setVideoPathList:@[_recordPath, mp4Path]];
}
```

3. 录制

录制部分，只要响应用户单击按钮调用 SDK 方法就可以了，为了方便起见，这里复用了这个按钮来显示当前状态。另外加上在进度条上显示进度的逻辑。

```
- (IBAction)onTapButton:(UIButton *)sender {
[_editor startPlayFromTime:0 toTime:_videoInfo.duration];
if ([_recorder startRecord:_recordPath coverPath:[_recordPath stringByAppendingString:@".p
ng"]] != 0) {
NSLog(@"相机启动失败");
}
[sender setTitle:@"录像中" forState:UIControlStateNormal];
sender.enabled = NO;
}
#pragma mark TXVideoPreviewListener
-(void) onPreviewProgress:(CGFloat)time
{
self.progressView.progress = time / _videoInfo.duration;
}
```

4. 拼接

录制好后开始完成拼接部分，此处需要指定两个视频在结果中的位置，这里设置一左一右。

```
-(void)onRecordComplete:(TXUGCRecordResult*)result;
{
NSLog(@"录制完成，开始合成");
[self.recordButton setTitle:@"合成中..." forState:UIControlStateNormal];

//获取录制视频的宽高
TXVideoInfo *videoInfo = [TXVideoInfoReader getVideoInfo:_recordPath];
CGFloat width = videoInfo.width;
```

```
CGFloat height = videoInfo.height;

//录制视频和原视频左右排列
CGRect recordScreen = CGRectMake(0, 0, width, height);
CGRect playScreen = CGRectMake(width, 0, width, height);
[_joiner setSplitScreenList:@[[NSValue valueWithCGRect:recordScreen],[NSValue valueWithC
GRect:playScreen]] canvasWidth:width * 2 canvasHeight:height];
[_joiner splitJoinVideo:VIDEO_COMPRESSED_720P videoOutputPath:_resultPath];
}
```

5. 显示进度条

实现合成进度的委托方法，在进度条中显示进度。

```
-(void) onJoinProgress:(float)progress
{
NSLog(@"视频合成中%d%%",(int)(progress * 100));
self.progressView.progress = progress;
}
```

6. 预览

实现合成完成的委托方法，并切换到预览界面。

```
#pragma mark TXVideoJoinerListener
-(void) onJoinComplete:(TXJoinerResult *)result
{
NSLog(@"视频合成完毕");
VideoPreviewController *controller = [[VideoPreviewController alloc] initWithVideoPath:_resul
tPath];
[self.navigationController pushViewController:controller animated:YES];
}
```

此就制作完成了，上面提到了一个视频预览的 VideoPreviewController **代码如下：**

- **VideoPreviewController.h：**

```
#import <UIKit/UIKit.h>
@interface VideoPreviewController : UIViewController
```

```
- (instancetype)initWithVideoPath:(NSString *)path;
@end
```

- **VideoPreviewController.m：**

```
@import TXLiteAVSDK_UGC;
@interface VideoPreviewController () <TXVideoPreviewListener>
{
TXVideoEditer *_editor;
}
@property (strong, nonatomic) NSString *videoPath;
@end
@implementation VideoPreviewController
- (instancetype)initWithVideoPath:(NSString *)path {
if (self = [super initWithNibName:nil bundle:nil]) {
self.videoPath = path;
}
return self;
}

- (void)viewDidLoad {
[super viewDidLoad];
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = self.view;
param.renderMode = RENDER_MODE_FILL_EDGE;

_editor = [[TXVideoEditer alloc] initWithPreview:param];
_editor.previewDelegate = self;
[_editor setVideoPath:self.videoPath];
[_editor startPlayFromTime:0 toTime:[TXVideoInfoReader getVideoInfo:self.videoPath].dura
tion];
}
-(void) onPreviewFinished
{
[_editor startPlayFromTime:0 toTime:[TXVideoInfoReader getVideoInfo:self.videoPath].dura
tion];
}
@end
```

至此就完成了全部合唱的基础功能，功能更加丰富的示例请参见 小视频源码。

Android

最近更新时间：2021-08-12 19:57:06

本篇教程向大家介绍如何完成合唱的基础功能。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

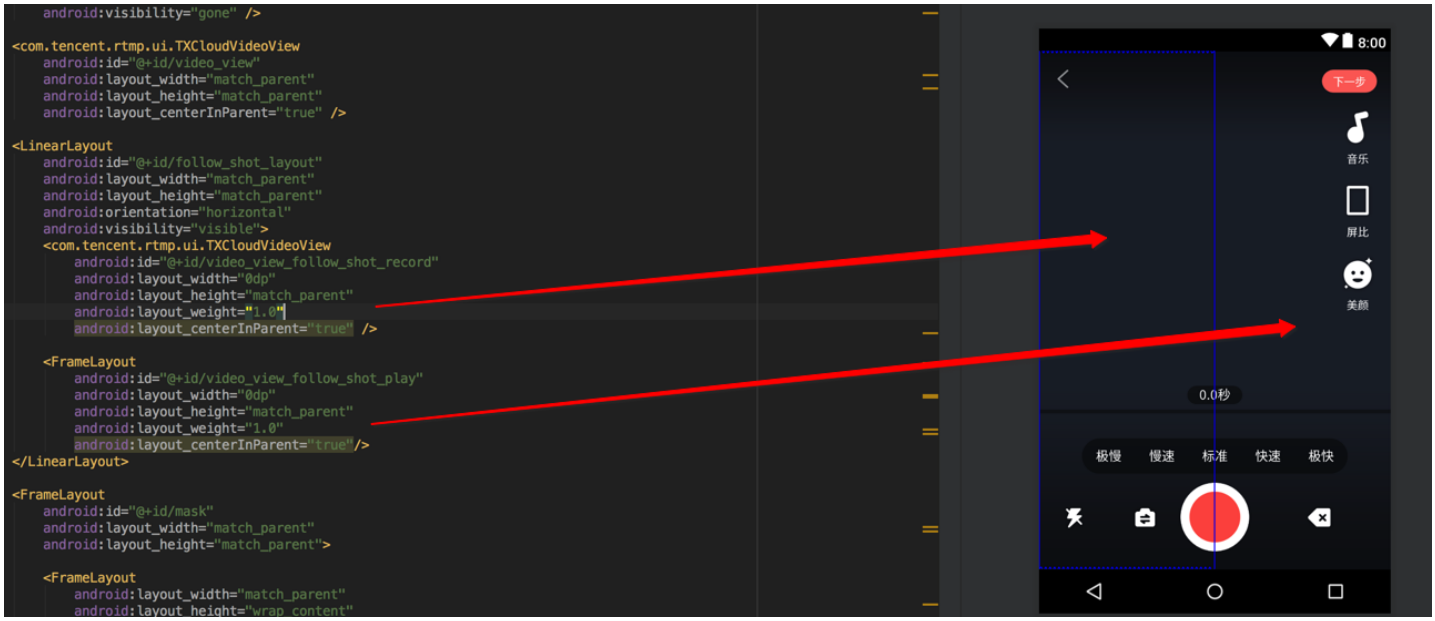
不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

过程简介

1. 在界面上放两个 View，一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

界面搭建

在录制界面 TCVideoRecordActivity 的 activity_video_record.xml 中创建两个 view，左半边是录制界面，右半边是播放界面。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为： TXVideoEditer、TXUGCRecord、TXVideoJoiner，其中播放也可以换成 TXVodPlayer 去播放。

1. 从小视频主页的视频列表中，选择一个视频进入播放界面 TCVodPlayerActivity，单击右下角的“合拍”按钮。
   首先会下载该视频到本地 sdcard 中，并获取该视频的音频采样率以及 FPS 等信息后进入录制界面。
2. 进入录制界面 TCVideoRecordActivity 进行合唱。需要注意以下几点：
   - 录制进度条以跟拍视频的进度为最大长度。
   - 保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象。
   - 保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象。
   - 录制设置渲染模式为自适应模式，在9:16的宽高比时能等比例缩放。
   - Android 的录制需要设置静音，否则会造成与跟拍视频的“二重唱”。

```
// 录制的界面
mVideoView = mVideoViewFollowShotRecord;
// 播放的视频
mFollowShotVideoPath = intent.getStringExtra(TCConstants.VIDEO_EDITER_PATH);
mFollowShotVideoDuration = (int)(intent.getFloatExtra(TCConstants.VIDEO_RECORD_D
URATION, 0) * 1000);
initPlayer();
// 录制进度条以跟拍视频的进度为最大长度，fps 以跟拍视频的 fps 为准
mMaxDuration = (int)mFollowShotVideoDuration;
mFollowShotVideoFps = intent.getIntExtra(TCConstants.RECORD_CONFIG_FPS, 20);
mFollowShotAudioSampleRateType = intent.getIntExtra(TCConstants.VIDEO_RECORD_A
```

```
UDIO_SAMPLE_RATE_TYPE, TXRecordCommon.AUDIO_SAMPLERATE_48000);
// 初始化合拍的接口
mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setVideoJoinerListener(this);
```

```
// 播放器初始化，这里使用 TXVideoEditer，也可以使用 TXVodPlayer
mTXVideoEditer = new TXVideoEditer(this);
mTXVideoEditer.setVideoPath(mFollowShotVideoPath);
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreview
Param();
param.videoView = mVideoViewPlay;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoEditer.initWithPreview(param);
```

```
customConfig.videoFps = mFollowShotVideoFps;
customConfig.audioSampleRate = mFollowShotAudioSampleRateType; // 录制的视频的音
频采样率必须与跟拍的音频采样率相同
customConfig.needEdit = false;
mTXCameraRecord.setVideoRenderMode(TXRecordCommon.VIDEO_RENDER_MODE_ADJ
UST_RESOLUTION); // 设置渲染模式为自适应模式
mTXCameraRecord.setMute(true); // 跟拍不从喇叭录制声音，因为跟拍的视频声音也会从喇叭
发出来被麦克风录制进去，造成跟原视频声音的"二重唱"。
```

3. 接下来就可以开始录制了，在录制到最大长度后，会回调 onRecordComplete，继续完成拼接部分，这里需要指定两个视频在结果中的位置。

```
private void prepareToJoiner(){
List<String> videoSourceList = new ArrayList<>();
videoSourceList.add(mRecordVideoPath);
videoSourceList.add(mFollowShotVideoPath);
mTXVideoJoiner.setVideoPathList(videoSourceList);
mFollowShotVideoOutputPath = getCustomVideoOutputPath("Follow_Shot_");
// 以左边录制的视频宽高为基准，右边视频等比例缩放
int followVideoWidth;
int followVideoHeight;
if ((float) followVideoInfo.width / followVideoInfo.height >= (float)recordVideoInfo.width / r
ecordVideoInfo.height) {
```

```
followVideoWidth = recordVideoInfo.width;
followVideoHeight = (int) ((float)recordVideoInfo.width * followVideoInfo.height / followVid
eoInfo.width);
} else {
followVideoWidth = (int) ((float)recordVideoInfo.height * followVideoInfo.width / followVide
oInfo.height);
followVideoHeight = recordVideoInfo.height;
}

TXVideoEditConstants.TXAbsoluteRect rect1 = new TXVideoEditConstants.TXAbsoluteRect
();
rect1.x = 0; //第一个视频的左上角位置
rect1.y = 0;
rect1.width = recordVideoInfo.width; //第一个视频的宽高
rect1.height = recordVideoInfo.height;

TXVideoEditConstants.TXAbsoluteRect rect2 = new TXVideoEditConstants.TXAbsoluteRect
();
rect2.x = rect1.x + rect1.width; //第2个视频的左上角位置
rect2.y = (recordVideoInfo.height - followVideoHeight) / 2;
rect2.width = followVideoWidth; //第2个视频的宽高
rect2.height = followVideoHeight;

List<TXVideoEditConstants.TXAbsoluteRect> list = new ArrayList<>();
list.add(rect1);
list.add(rect2);
mTXVideoJoiner.setSplitScreenList(list, recordVideoInfo.width + followVideoWidth, recordVi
deoInfo.height); //第2、3个 param：两个视频合成画布的宽高
mTXVideoJoiner.splitJoinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mFollow
ShotVideoOutputPath);
}
```

4. 监听合成的回调，在 onJoinComplete 后跳转到预览界面播放。

```
@Override
public void onJoinComplete(TXVideoEditConstants.TXJoinerResult result) {
mCompleteProgressDialog.dismiss();
if(result.retCode == TXVideoEditConstants.JOIN_RESULT_OK){
runOnUiThread(new Runnable() {
```

```
@Override
public void run() {
isReadyJoin = true;
startEditerPreview(mFollowShotVideoOutputPath);
if(mTXVideoEditer != null){
mTXVideoEditer.release();
mTXVideoEditer = null;
}
}
});
}else{
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(TCVideoRecordActivity.this, "合成失败", Toast.LENGTH_SHORT).show();
}
});
}
}
```

至此就完成了全部合唱的基础功能，完整代码请参见 小视频源码。

图片转场特效
iOS

最近更新时间：2021-08-12 19:57:10

腾讯云视立方短视频 UGSV SDK 在4.7版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

接口函数

```
/*
*pitureList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
*fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在，图片数量是否大于等于3张，fps 是否正常；
*/
- (int)setPictureList:(NSArray<UIImage *> *)pitureList fps:(int)fps;

/*
*transitionType：转场类型，详情见 TXTransitionType
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
```

```
- (void)setPictureTransition:(TXTransitionType)transitionType duration:(void(^)(CGFloat))dura
tion;
```

- setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过 duration 获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速、片尾水印。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

Android

最近更新时间：2021-08-12 19:57:13

腾讯云视立方短视频 UGSV SDK 在4.9版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

接口函数

```
/*

* bitmapList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
* fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在
*/
public int setPictureList(List<Bitmap> bitmapList, int fps);

/*
* type：转场类型，详情见 TXVideoEditConstants
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
public long setPictureTransition(int type)
```

- 其中，setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过返回值获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

版权所有：腾讯云计算（北京）有限责任公司

进阶功能
定制视频数据
iOS

最近更新时间：2021-08-12 19:57:17

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制预处理回调

```
/**

* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param texture 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理
* 说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEX
TURE_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height;

/**
* 五官位置回调（企业版接口）
* @prama points 五官坐标
* 说明：使用了挂件的相关功能如动效贴纸、大眼或者瘦脸等。此回调在 onPreProcessTexture:width:h
eight: 之前会被调用
```

```
*/
- (void)onDetectFacePoints:(NSArray *)points;

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

编辑预处理回调

```
/**
在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
@param texture 纹理 ID
@param width 纹理的宽度
@param height 纹理的高度
@param timestamp 纹理 timestamp 单位 ms
@return 返回给 SDK 的纹理
说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEXT
URE_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
timestamp 为当前视频帧的 pts ，单位是 ms ，客户可以根据自己的需求自定义滤镜特效
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height ti
mestamp:(UInt64)timestamp;

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

Android

最近更新时间：2021-08-12 19:57:20

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制预处理回调

```
public interface VideoCustomProcessListener {
/**

* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param textureId 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
* 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须
是 GLES20.GL_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height);

/**
* 增值版回调人脸坐标
* @param points 归一化人脸坐标，每两个值表示某点 P 的 X、Y 值。值域[0.f,1.f]
*/
void onDetectFacePoints(float[] points);

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
```

```
*/
void onTextureDestroyed();
}
```

编辑预处理回调

```
public interface TXVideoCustomProcessListener {
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
*
* @param textureId 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
* <p>
* 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GLES20.GL_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height, long timestamp);

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
void onTextureDestroyed();
}
```

视频鉴黄

最近更新时间：2021-08-12 19:57:25

在个人直播录制、短视频 UGSV 等场景中，视频内容是不可预知的。为了避免一些违规内容出现在点播平台上，开发者会要求先对上传的视频做审核，确认合规后再进行转码和分发。腾讯云视立方短视频 UGSV 解决方案支持对视频进行 AI 鉴黄，自动识别视频是否涉及色情内容。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	✓	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

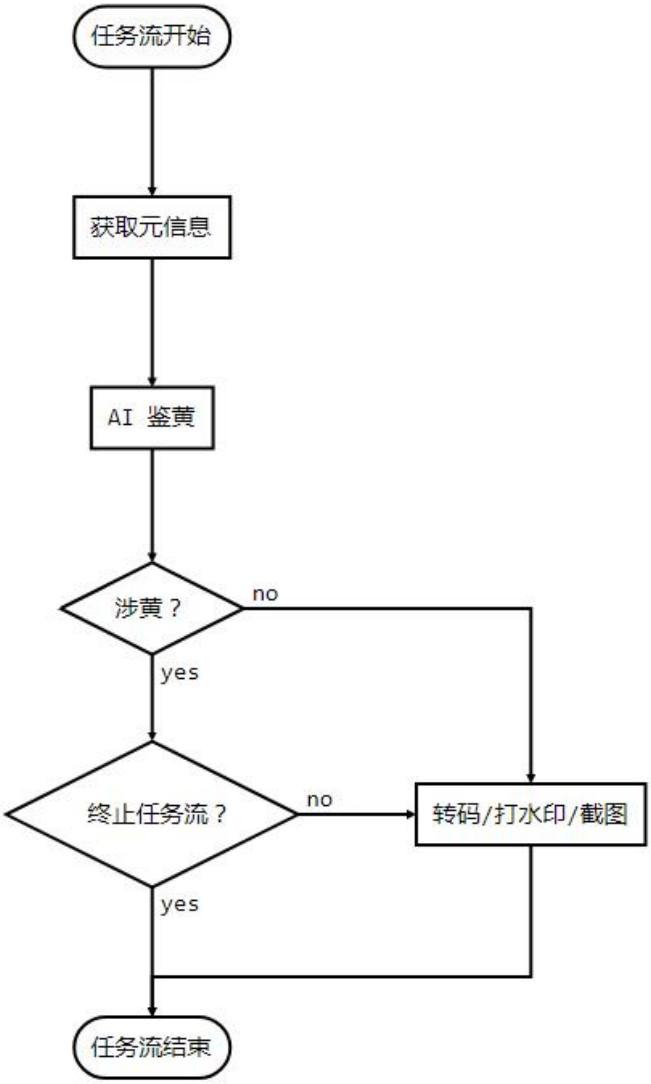
不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

使用 AI 鉴黄

AI 鉴黄功能需要集成在视频处理任务流中使用，它依赖于云点播后台的 视频 AI – 内容审核，审核结果通过事件通知的方式来通知 App 后台服务。云点播内置了一个任务流 QCVB_ProcessUGCFile 用于 UGSV 短视频鉴黄场景，当用户使用该任务流并指定进行 AI 鉴黄时，鉴黄操作将优先执行，并根据鉴黄结果来决定是否进行后续处理

版权所有：腾讯云计算（北京）有限责任公司

（转码、打水印和截图等）。流程图如下：



AI 模版介绍

模版 ID	涉黄处理	采样频率
10	终止任务流（转码、打水印、截图等均不会执行）	时长小于500秒的视频，每1秒采样1次；时长大于或等于500秒的视频，每1%时长采样1次
20	继续执行任务流	无

AI 鉴定接入示例

1. 在生成上传签名时提交 AI 鉴黄任务

```
/**
* 响应签名请求并上传带有 AI 鉴黄任务
*/
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
//上传时指明模版参数与任务流
signature: gVodHelper.createFileUploadSignature({ procedure: 'QCVB_SimpleProcessFile({1,1,1,
1})' })
}
});
}
```

2. 在获事件通知时获取 AI 鉴黄结果

```
/**
* 获取事件的 AI 鉴黄结果
*/
function getAiReviewResult(event){
let data = event.eventContent.data;
if(data.aiReview){
return data.aiReview;
}
return {};
}
```