

移动直播

iOS端集成

产品文档



腾讯云

【版权声明】

©2013-2018 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

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。

【服务声明】

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

文档目录

iOS端集成

API 接口

标准直播

- 集成(XCode)

- 推流(LivePusher)

- 直播(LivePlayer)

- 点播(VodPlayer)

- 录屏(ReplayKit)

- 特效(AI Effects)

实时连麦

- 直播连麦 (LiveRoom)

- 视频通话 (RTCRoom)

进阶功能

- SDK内部原理

- SDK指标监控

- Qos流量控制

- 编码参数调整

- 定制视频数据

iOS端集成

API 接口

最近更新时间：2018-09-29 09:59:58

下面是腾讯视频云iOS SDK的主要接口列表，分为TXLivePush和TXLivePlayer两个类及相应的回调接口，详细接口请查看[API 接口文档](#)。

接口概览

TXLivePush

名称	描述
(id)initWithConfig:(TXLivePushConfig *)config;	设置推流配置信息
(int)startPush:(NSString *)rtmpURL;	启动到指定URL推流
(void)stopPush;	停止推流
(void)pausePush;	暂停推流
(void)resumePush;	恢复推流
(bool)isPublishing;	是否正在推流中
(int) startRecord:(NSString *)videoPath;	开始录制短视频
(int) stopRecord	结束录制短视频
(int)startPreview:(UIView *)view;	开始推流画面的预览
(void)stopPreview;	停止预览
(int)switchCamera;	切换前后摄像头
(void)setMirror:(BOOL)isMirror	设置是否为镜像画面
(void)setBeautyStyle	设置美颜 和 美白 效果级别
(void)setEyeScaleLevel:(float)eyeScaleLevel;	设置大眼级别
(void)setFaceScaleLevel:(float)faceScaleLevel;	设置瘦脸级别

名称	描述
(void)setFilter:(UIImage *)image;	设置滤镜
(void)setSpecialRatio:(float)specialValue	设置滤镜效果程度
(void)setFaceVLevel:(float)faceVLevel;	设置V脸
(void)setChinLevel:(float)chinLevel;	设置下巴拉伸或收缩
(void)setFaceShortLevel:(float)faceShortlevel;	设置瘦脸
(void)setNoseSlimLevel:(float)noseSlimLevel;	设置瘦鼻
(BOOL)toggleTorch:(BOOL)bEnable;	打开或关闭闪光灯
(void)setRenderRotation:(int)rotation;	设置本地视频方向
(void)setMute:(BOOL)bEnable;	设置静音
(void)sendCustomPCMDData	发送客户自定义的音频PCM数据
(void)sendVideoSampleBuffer	发送自定义的SampleBuffer
(void)sendAudioSampleBuffer	发送自定义音频包
(void)setSendAudioSampleBufferMuted:(BOOL)muted;	Replaykit发送静音包
(void)setFocusPosition:(CGPoint)touchPoint;	调用手动对焦功能
(void)setZoom:(CGFloat)distance;	推送自定义视频数据
(BOOL)playBGM:(NSString *)path;	播放背景音乐
(BOOL)stopBGM;	停止播放背景音乐
(BOOL)resumeBGM;	继续播放背景音乐
(int)getMusicDuration:(NSString *)path;	获取音乐文件总时长，单位毫秒
(BOOL)setMicVolume:(float)volume;	设置麦克风的音量大小
(BOOL)setBGMVolume:(float)volume;	设置背景音乐的音量大小
(BOOL)setBgmPitch:(float)pitch;	设置背景音的变声类型
(BOOL)setReverbType:(TXReverbType)reverbType;	设置混响效果, 详见 TXReverbType
(BOOL)setVoiceChangerType	设置变声类型

名称	描述
(void)setGreenScreenFile:(NSURL *)file;	设置绿幕文件。仅增值版有效
(void)selectMotionTmpl	选择动效
(void)setLogViewMargin	设置状态浮层view在渲染view上的边距
(void)showVideoDebugLog	是否显示播放状态统计及事件消息浮层view
(void)snapshot	推流截图.仅对硬编起效
(BOOL)sendMessageEx	发送消息

TXLivePlayer

名称	描述
setupVideoWidget	创建Video渲染Widget
removeVideoWidget	移除Video渲染Widget
setPlayListener	设置TXLivePlayer 的回调
startPlay	开始播放
stopPlay	停止播放
pause	暂停播放
resume	恢复播放
prepareLiveSeek	直播时移准备，拉取该直播流的起始播放时间
resumeLive	是否正在播放
seek	跳转到音视频流某个时间
width	视频宽度
height	视频高度
setRenderRotation	设置画面的方向
setRenderMode	设置画面的裁剪模式
setMute	设置静音

名称	描述
startRecord	开始录制短视频
stopRecord	结束录制短视频
snapshot	通过回调返回当前图像
setRate	设置播放速率。点播有效
setLogViewMargin	设置状态浮层view在渲染view上的边距
showVideoDebugLog	是否显示播放状态统计及事件消息浮层view
setAudioRoute	设置声音播放模式(切换扬声器, 听筒)
switchStream	flv直播无缝切换

TXLivePushConfig

属性名	类型	说明
customModeType	int	客户自定义模式
beautyFilterDepth	float	美颜强度 0 ~ 9, 默认值为0
whiteningFilterDepth	float	播美白强度:0 ~ 9, 默认值为0
enableHWAceleration	BOOL	开启硬件加速, iOS系统版本>8.0 默认开启
homeOrientation	int	home键所在方向, 用来切换横竖屏推流 (tips : 此参数的设置可能会改变推流端本地视频流方向, 此参数设置后, 请调用TXLivePush里的setRenderRotation 来修正推流端本地视频流方向, 具体请参考demo设置) ,默认值为HOME_ORIENTATION_DOWN
videoFPS	int	视频采集帧率, 默认值为 15
enableAEC	BOOL	是否开启回声消除, 默认值为 NO
videoResolution	int	视频分辨率, 默认值为 VIDEO_RESOLUTION_TYPE_360_640
videoBitratePIN	int	视频固定码率, 默认值为700
videoEncodeGop	int	视频编码 GOP, 单位 second 秒, 默认值为3

属性名	类型	说明
audioSampleRate	int	音频采样率, 取值设置为 枚举值 TX_Enum_Type_AudioSampleRate, 也可直接设置为对应的采样率, 比如 audioSampleRate = AUDIO_SAMPLE_RATE_48000 或 audioSampleRate = 48000, 默认值为 AUDIO_SAMPLE_RATE_48000
audioChannels	int	音频声道数, 默认值为1
enableAutoBitrate	BOOL	码率自适应: SDK会根据网络情况自动调节视频码率, 调节范围在 (videoBitrateMin - videoBitrateMax), 默认值为NO
autoAdjustStrategy	int	码率自适应: SDK会根据网络情况自动调节视频码率, 同时自动调整分辨率, 默认值为 AUTO_ADJUST_BITRATE_STRATEGY_1
videoBitrateMax	int	视频最大码率, 仅当enableAutoBitrate = YES 时有效, 默认值为 1000
videoBitrateMin	int	视频最小码率, 仅当enableAutoBitrate = YES 时有效, 默认值为 400
enableNAS	BOOL	噪音抑制, 默认值为 YES
frontCamera	BOOL	是否前置 camera, 默认值为 YES
touchFocus	BOOL	是否允许点击曝光聚焦, 默认为 YES
enableZoom	BOOL	是否允许双指手势放大预览画面, 默认为 NO
connectRetryCount	int	推流器连接重试次数: 最小值为 1, 最大值为 10, 默认值为 3
connectRetryInterval	int	推流器连接重试间隔: 单位秒, 最小值为 3, 最大值为 30, 默认值为 3
watermark	UIImage *	设置水印图片. 设为nil等同于关闭水印
watermarkPos	CGPoint	设置水印图片位置, 水印大小为图片实际大小

属性名	类型	说明
watermarkNormalization	CGRect	水印相对于推流分辨率的归一化坐标, x,y,width,height 取值范围 0~1; height不用设置, sdk内部会根据水印宽高比自动计算 height
pVideoFuncPtr	PVideoProcessHookFunc	视频预处理Hook
pAudioFuncPtr	PAudioProcessHookFunc	音频预处理Hook
sampleBufferSize	CGSize	发送自定义 CMSampleBuffer 的输出分辨率。当设置此属性时, videoResolution 自动失效。此值设置需与源 SampleBuffer 的画面比例一致, 否则会引起画面变形。调用 sendVideoSampleBuffer 必须设置此值, 或者设置autoSampleBufferSize=YES
autoSampleBufferSize	BOOL	设置 YES 时, 调用 sendVideoSampleBuffer 输出分辨率等于输入分辨率, 默认值为 NO
enableAudioAcceleration	BOOL	开启音频硬件加速, 默认值为 YES
pauseTime	int	后台推流时长, 单位秒, 默认 300 秒
pauseFps	UIImage *	后台推流图片, 图片最大尺寸不能超过 1920*1920
enableAEC	BOOL	是否开启回声消除, 默认值为 YES
enableAudioPreview	BOOL	是否开启耳返, 默认值为 NO
enablePureAudioPush	BOOL	是否纯音频推流, 默认值为 NO
enableNearestIP	BOOL	是否就近选路, 默认值为 YES
rtmpChannelType	int	RTMP 传输通道的类型, 取值为枚举值: TX_Enum_Type_RTMPChannel, 默认值为 RTMP_CHANNEL_TYPE_AUTO

TXLivePlayConfig

属性名	类型	说明
cacheTime	float	播放器缓存时间: 单位秒, 取值需要大于0, 默认值为5
bAutoAdjustCacheTime	BOOL	播放器缓存时间: 单位秒, 取值需要大于0, 默认值为5

属性名	类型	说明
maxAutoAdjustCacheTime	float	播放器缓存自动调整的最大时间：单位秒，取值需要大于0, 默认值为5
minAutoAdjustCacheTime	float	播放器缓存自动调整的最小时间：单位秒，取值需要大于0, 默认值为5
videoBlockThreshold	int	播放器视频卡顿报警阈值，只有渲染间隔超过这个阈值的卡顿才会有PLAY_WARNING_VIDEO_PLAY_LAG通知
connectRetryCount	int	播放器连接重试次数：最小值为 1，最大值为 10, 默认值为 3
enableAEC	BOOL	是否开启回声消除，默认值为 NO
enableMessage	BOOL	是否开启消息通道，默认值为 NO
playerPixelFormatType	OSType	视频渲染对象回调的视频格式. 仅支持 kCVPixelFormatType_420YpCbCr8Planar和 kCVPixelFormatType_420YpCbCr8BiPlanarFullRange, 默认值为kCVPixelFormatType_420YpCbCr8Planar
enableNearestIP	BOOL	只对加速拉流生效，用于指定加速拉流是否开启就近选路(当前版本不启用)
rtmpChannelType	int	RTMP传输通道的类型，取值为枚举值： TX_Enum_Type_RTMPChannel, 默认值为 RTMP_CHANNEL_TYPE_AUTO

接口详情

TXLivePush

1.initWithConfig

接口详情：`(id)initWithConfig:(TXLivePushConfig *)config`

init 时候初始化 config.

• 参数说明

参数	类型	说明
config	TXLivePushConfig *	推流器配置信息

2.startPush

接口详情：`(int)startPush:(NSString *)rtmpURL`

启动到指定URL推流（rtmpURL 腾讯云的推流地址）

• 参数说明

参数	类型	说明
url	NSString *	RTMP完整的URL

• 返回值

0 为 OK

3.stopPush

接口详情：`(void)stopPush`

停止推流

4.pausePush

接口详情：`(void)pausePush`

暂停推流，后台视频发送 TXLivePushConfig 里面设置的图像，音频会继续录制声音发送, 如果不需要录制声音，需要再调下 `setMute` 接口。

- 当从前台切到后台的时候，调用 `pausePush` 会推配置里设置的图片(TXLivePushConfig.pauseImg)
- 当从后台回到前台的时候，调用 `resumePush` 恢复推送 camera 采集的数据
- 相关属性设置请参考 TXLivePushConfig：
- `pauseImg` 设置后台推流的默认图片，不设置为默认黑色背景
- `pauseFps` 设置后台推流帧率，最小值为 5，最大值为 20，默认 10
- `pauseTime` 设置后台推流持续时长，单位秒，默认 300 秒

5.isPublishing

接口详情：`(bool)isPublishing`

是否正在推流中。YES 推流中，NO 没有推流

6.startRecord

接口详情：`(int)startRecord:(NSString *)videoPath`

开始录制短视频，开始推流后才能启动录制

注意：

1. 录制过程中请勿动态切换分辨率和软硬编，可能导致生成的视频异常
2. 目前仅支持 Enterprise 和 Professional SDK版本，其他版本调用无效

注意：关闭摄像头是同步操作，正常会耗时100~200ms，在某些手机上耗时会多一些，如果觉得耗时有影响可以抛到异步线程调用该接口

• 参数说明

参数	类型	说明
videoPath	NSString *	视频录制后存储路径

• 返回值

- 0 成功；
- -1 videoPath 为nil；
- -2 上次录制未结束，请先stopRecord
- -3 推流未开始

7. stopRecord

接口详情：`(int) stopRecord`

结束录制短视频，停止推流后，如果视频还在录制中，SDK内部会自动结束录制

• 返回值

- 0 成功；
- -1 不存在录制任务；

8.stopPreview

接口详情：`(void)stopPreview`

停止预览

9.switchCamera

接口详情：`(int)switchCamera`

切换前后摄像头

10.setMirror

接口详情：`(void)setMirror:(BOOL)isMirror`

isMirror YES：播放端看到的是镜像画面 NO：播放端看到的是非镜像画面

tips：推流端前置摄像头默认看到的是镜像画面，后置摄像头默认看到的是非镜像画面

11.setBeautyStyle

接口详情：`(void)setBeautyStyle:(TX_Enum_Type_BeautyStyle)beautyStyle beautyLevel:(float)beautyLevel whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel`

设置美颜和美白效果级别。

参数	类型	说明
beautyStyle	TX_Enum_Type_BeautyStyle	
beautyLevel	float	美颜级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。
whitenessLevel	float	美白级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。
ruddinessLevel	float	红润级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。

其中，TX_Enum_Type_BeautyStyle 如下

名称	值	说明
BEAUTY_STYLE_SMOOTH	0	光滑
BEAUTY_STYLE_NATURE	1	自然
BEAUTY_STYLE_PITU	2	P 图美颜

12.setEyeScaleLevel

接口详情：`(void)setEyeScaleLevel:(float)eyeScaleLevel`

设置大眼级别（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
eyeScaleLevel	float	大眼级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。

13.setFaceScaleLevel

接口详情：`(void)setFaceScaleLevel:(float)eyeScaleLevel`

设置瘦脸级别（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
faceScaleLevel	float	瘦脸级别取值范围 0 ~ 9 ; 0 表示关闭 1 ~ 9值越大 效果越明显。

14. setFilter

接口详情：`(void)setFilter:(UIImage *)image`

设置指定素材滤镜特效。

参数	类型	说明
image	UIImage *	指定素材，即颜色查找表图片。注意：一定要用png格式

15.setSpecialRatio

接口详情：`(void)setSpecialRatio:(float)specialValue`

设置滤镜效果程度

参数	类型	说明
specialValue	float	从 0 到 1，越大滤镜效果越明显，默认取值 0.5

16.setFaceVLevel

接口详情：`(void)setFaceVLevel:(float)faceVLevel`

设置V脸（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
faceVLevel	float	取值范围 0 ~ 9 ; 0 表示关闭 1 ~ 9值越大 效果越明显

17.setFaceShortLevel

接口详情：`(void)setChinLevel:(float)chinLevel`

设置下巴拉伸或收缩（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
chinLevel	float	下巴拉伸或收缩级别取值范围 -9 ~ 9 ; 0 表示关闭 -9收缩 ~ 9拉伸

18.setFaceShortLevel

接口详情：`(void)setFaceShortLevel:(float)faceShortlevel`

设置短脸（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
faceShortlevel	float	短脸级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。

19.setNoseSlimLevel

接口详情：`(void)setNoseSlimLevel:(float)noseSlimLevel`

设置瘦鼻（特权版本有效，普通版本设置此参数无效）

参数	类型	说明
noseSlimLevel	float	瘦鼻级别取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。

20.toggleTorch

接口详情：`(BOOL)toggleTorch:(BOOL)bEnable`

打开闪光灯

参数	类型	说明
bEnable	BOOL	YES 打开，NO 关闭

返回值

- YES，打开成功。
- NO，打开失败。

21.setRenderRotation

接口详情：`(void)setRenderRotation:(int)rotation`

设置本地视频方向。

参数	类型	说明
rotation	int	取值为 0, 90, 180, 270（其他值无效）表示推流端本地视频向右旋转的角度

22.setMute

接口详情：`(void)setMute:(BOOL)bEnable`

设置静音。

23.sendCustomPCMDData

接口详情：`(void)sendCustomPCMDData:(unsigned char *)data len:(unsigned int)len`

发送客户自定义的音频PCM数据。

说明：目前SDK只支持16位采样的PCM编码；如果是单声道，请保证传入的PCM长度为2048；如果是双声道，请保证传入的PCM长度为4096。

参数	类型	说明
data	unsigned char *	PCM 数据
len	unsigned int	数据的长度

24.sendVideoSampleBuffer

接口详情：`(void)sendVideoSampleBuffer:(CMSampleBufferRef)sampleBuffer`

发送自定义的SampleBuffer，代替sendCustomVideoData。内部有简单的帧率控制，发太快会自动丢帧；超时则会重发最后一帧。

相关属性设置请参考TXLivePushConfig，autoSampleBufferSize优先级高于sampleBufferSize。

25.sendAudioSampleBuffer

接口详情：`(void)sendAudioSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleBufferType`

Replaykit发送自定义音频包。

- 参数说明

参数	类型	说明
sampleBuffer	CMSampleBufferRef	声音
sampleBufferType	RPSampleBufferType	RPSampleBufferTypeAudioApp or RPSampleBufferTypeAudioMic, 当两种声音都发送时，内部做混音；否则只发送一路声音

26.setSendAudioSampleBufferMuted

接口详情：`(void)setSendAudioSampleBufferMuted:(BOOL)muted`

Replaykit发送静音包。

27.setFocusPosition

接口详情：`(void)setFocusPosition:(CGPoint)touchPoint`

调用手动对焦功能。

说明: 早期SDK版本手动对焦功能是由SDK内部触发, 现在把手动对焦的接口开放出来, 客户可以根据自己需求触发, 如果客户调用这个接口, SDK内部触发对焦的逻辑将会停止, 避免重复触发对焦逻辑。

参数	类型	说明
touchPoint	CGPoint	传入的对焦点位置

28.setZoom

接口详情：`(void)setZoom:(CGFloat)distance`

调整焦距。

• 参数说明

参数	类型	说明
distance	CGFloat	distance取值范围 1~5 , 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) , 这里最大值推荐为5, 超过5后视频数据会变得模糊不清

29.playBGM

接口详情：`(BOOL)playBGM:(NSString *)path`

以下接口用于混音处理, 背景音与 Mic 采集到的人声混合。

播放背景音乐。

• 参数说明

参数	类型	说明
path	NSString *	音乐文件路径, 一定要是 app 对应的 document 目录下面的路径, 否则文件会读取失败

30. playBGM

接口详情：

```
(BOOL) playBGM:(NSString *)path  
withBeginNotify:(void (^)(NSInteger errorCode))beginNotify
```

```
withProgressNotify:(void (^)(NSInteger progressMS, NSInteger durationMS))progressNotify
andCompleteNotify:(void (^)(NSInteger errorCode))completeNotify;
```

播放背景音乐。

参数	类型	说明
path	NSString *	音乐文件路径，一定要是 app 对应的 document 目录下面的路径，否则文件会读取失败
beginNotify	void (^)(NSInteger errorCode))beginNotify	音乐播放开始的回调通知
withProgressNotify	void (^)(NSInteger progressMS, NSInteger durationMS))progressNotify	音乐播放的进度通知，单位毫秒
completeNotify	(void (^)(NSInteger errorCode))completeNotify	音乐播放结束的回调通知

31.stopBGM

接口详情：`(BOOL)stopBGM`

停止播放背景音乐。

32.pauseBGM

接口详情：`(BOOL)pauseBGM`

暂停播放背景音乐。

33.resumeBGM

接口详情：`(BOOL)resumeBGM`

继续播放背景音乐。

34.getMusicDuration

接口详情：`(int)getMusicDuration:(NSString *)path`

获取音乐文件总时长，单位毫秒。

参数	类型	说明
path	NSString *	音乐文件路径，如果path为空，那么返回当前正在播放的music时长

返回值：音乐文件总时长，单位为毫秒。

35.setMicVolume

接口详情：`(BOOL)setMicVolume:(float)volume`

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。

参数	类型	说明
volume	float	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

36.setBGMVolume

接口详情：`(BOOL)setBGMVolume:(float)volume`

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

参数	类型	说明
volume	float	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

37.setBGMPitch

接口详情：`(BOOL)setBgmPitch:(float)pitch`

设置背景音的变声类型

参数	类型	说明
pitch	float	默认值是0.f;范围是 [-1,1]

38.setVideoQuality

接口详情：

```
(void)setVideoQuality:(TX_Enum_Type_VideoQuality)quality
adjustBitrate:(BOOL) adjustBitrate
adjustResolution:(BOOL) adjustResolution;
```

设置视频质量。

参数	类型	说明
quality	TX_Enum_Type_VideoQuality	画质类型(标清，高清，超高清)
adjustBitrate	BOOL	动态码率开关

参数	类型	说明
adjustResolution	BOOL	动态切分辨率开关

39.setReverbType

接口详情：(BOOL)setReverbType:(TXReverbType)reverbType

设置混响效果。

参数	类型	说明
reverbType	TXReverbType	混响类型，详见 TXReverbType

TXRevbType:

```
typedef NS_ENUM(NSInteger, TXReverbType) {
    REVERB_TYPE_0 = 0, //关闭混响
    REVERB_TYPE_1 = 1, //KTV
    REVERB_TYPE_2 = 2, //小房间
    REVERB_TYPE_3 = 3, //大会堂
    REVERB_TYPE_4 = 4, //低沉
    REVERB_TYPE_5 = 5, //洪亮
    REVERB_TYPE_6 = 6, //金属声
    REVERB_TYPE_7 = 7, //磁性
};
```

40.setVoiceChangerType

接口详情：(BOOL)setVoiceChangerType:(TXVoiceChangerType)voiceChangerType

设置变声类型。

参数	类型	说明
voiceChangerType	TXVoiceChangerType	变声类型, 详见 TXVoiceChangerType

其中，TXVoiceChangerType定义如下：

```
typedef NS_ENUM(NSInteger, TXVoiceChangerType) {
    VOICECHANGER_TYPE_0 = 0, //关闭变声
    VOICECHANGER_TYPE_1 = 1, //熊孩子
    VOICECHANGER_TYPE_2 = 2, //萝莉
    VOICECHANGER_TYPE_3 = 3, //大叔
};
```

```

VOICECHANGER_TYPE_4 = 4, //重金属
VOICECHANGER_TYPE_5 = 5, //感冒
VOICECHANGER_TYPE_6 = 6, //外国人
VOICECHANGER_TYPE_7 = 7, //困兽
VOICECHANGER_TYPE_8 = 8, //死肥仔
VOICECHANGER_TYPE_9 = 9, //强电流
VOICECHANGER_TYPE_10 = 10, //重机械
VOICECHANGER_TYPE_11 = 11, //空灵
};
    
```

41.setGreenScreenFile

接口详情：`(void)setGreenScreenFile:(NSURL *)file`

设置绿幕文件。仅增值版有效

参数	类型	说明
file	NSURL *	绿幕文件路径。支持mp4; nil 关闭绿幕

42.selectMotionTmpl

接口详情：`(void)selectMotionTmpl:(NSString *)tmplName inDir:(NSString *)tmplDir`

选择动效。仅增值版有效

参数	类型	说明
tmplName	NSString *	动效名称
tmplDir	NSString *	动效目录

43.setLogViewMargin

接口详情：`(void)setLogViewMargin:(UIEdgeInsets)margin`

设置状态浮层view在渲染view上的边距

参数	类型	说明
margin	UIEdgeInsets	状态浮层view在渲染view上的边距

44.showVideoDebugLog

接口详情：`(void)showVideoDebugLog:(BOOL)isShow`

是否显示播放状态统计及事件消息浮层view。

45.snapshot

接口详情：`(void)snapshot:(void (^)(UIImage *))snapshotCompletionBlock`

推流截图，仅对硬编起效。

参数	类型	说明
snapshotCompletionBlock	void (^)(UIImage *)	截图完成回调

46.sendMessageEx

接口详情：`(void)sendMessageEx:(NSData *) data`

发送消息(消息大小不允许超过2K)，播放端通过 `onPlayEvent(PLAY_EVT_GET_MESSAGE)`接收。

该接口发送消息，能够解决旧的sendMessage接口会导致在iOS上无法播放对应的HLS流的问题。

参数	类型	说明
data	NSData *	要发送的内容

TXLivePlayer

1. setupVideoWidget

接口详情：`(void)setupVideoWidget:(CGRect)frame containerView:(UIView *)view insertIndex:(unsigned int)idx`

创建Video渲染Widget,该控件承载着视频内容的展示。

• 参数说明

参数	类型	说明
frame	CGRect	Widget在父view中的rc
view	UIView *	父view
idx	unsigned int	Widget在父view上的层级位置

2. removeVideoWidget

接口详情：`(void)removeVideoWidget`

移除Video渲染Widget。

3. startPlay

接口详情：`(int)startPlay:(NSString *)url type:(TX_Enum_PlayType)playType`

启动从指定 URL 播放 RTMP 音视频流。

参数	类型	说明
url	NSString *	完整的URL(如果播放的是本地视频文件，这里传本地视频文件的完整路径)
playType	TX_Enum_PlayType	播放类型

返回值：0 = OK

4.stopPlay

接口详情：`(int)stopPlay`

停止播放音视频流。

返回：0 = OK

5.isPlaying

接口详情：`(bool)isPlaying`

是否正在播放。

返回值：YES 为正在播放。

6.pause

接口详情：`(void)pause`

点播场景是暂停播放。直播场景是没有暂停播放这说法，调用该方法意味这停止拉流。

7.resume

接口详情：`(void)resume`

点播场景是从 pause 位置恢复播放。直播场景则是重新拉流。

8.prepareLiveSeek

接口详情：`(int)prepareLiveSeek`

直播时移准备，拉取该直播流的起始播放时间。

使用时移功能需在播放开始后调用此方法，否者时移失败。时移的使用请参考文档

<https://cloud.tencent.com/document/product/266/9237>

非腾讯云直播地址不能时移。

- 返回值

返回值	含义
0	ok
-1	appId 未配置
-2	流 ID 格式错误

9.resumeLive

接口详情：`(int)resumeLive`

停止时移播放，返回直播

返回 0 代表 ok。

10.seek

接口详情：`(int)seek:(float)time`

点播流播放跳转到音视频流某个时间；直播流则时移到该位置。

- 参数说明

参数	类型	说明
time	float	时间，单位为秒

11.setRenderRotation

接口详情：`(void)setRenderRotation:(TX_Enum_Type_HomeOrientation)rotation`

设置画面的方向。

其中，TX_Enum_Type_HomeOrientation 的定义如下：

```
typedef NS_ENUM(NSInteger, TX_Enum_Type_HomeOrientation) {  
    HOME_ORIENTATION_RIGHT = 0, // home在右边  
    HOME_ORIENTATION_DOWN, // home在下面  
    HOME_ORIENTATION_LEFT, // home在左边  
    HOME_ORIENTATION_UP, // home在上面  
};
```

12.setRenderMode

接口详情：(void)setRenderMode:(TX_Enum_Type_RenderMode)renderMode

- 参数说明

参数	类型	说明
renderMode	TX_Enum_Type_RenderMode	详见 TX_Enum_Type_RenderMode 的定义

```
typedef NS_ENUM(NSUInteger, TX_Enum_Type_RenderMode) {
    RENDER_MODE_FILL_SCREEN = 0, // 图像铺满屏幕
    RENDER_MODE_FILL_EDGE // 图像长边填满屏幕
};
```

13.setMute

接口详情：(void)setMute:(BOOL)bEnable

设置静音

14.startRecord

接口详情：(int) startRecord:(TXRecordType)recordType

开始录制短视频。

- 参数说明

参数	类型	说明
recordType	TXRecordType	参见TXRecordType定义

TXRecordType 定义如下

```
typedef NS_ENUM(NSUInteger, TXRecordType)
{
    RECORD_TYPE_STREAM_SOURCE = 1, //视频源为正在播放的视频流
};
```

- 返回值

返回值	含义
0	ok

返回值	含义
-1	正在录制短视频
-2	videoRecorder初始化失败

15.stopRecord

接口详情：`(int) stopRecord`

结束录制短视频。

返回值	含义
0	ok
-1	不存在录制任务
-2	videoRecorder 没有初始化

16. snapshot

接口详情：`(void) snapshot:(void (^)(UIImage *)) snapshotCompletionBlock`

截屏。snapshotCompletionBlock 通过回调返回当前图像。

17. setRate

接口详情：`(void) setRate:(float) rate`

设置播放速率。点播有效。

参数	类型	说明
rate	float	1.0

18. setLogViewMargin

接口详情：`(void) setLogViewMargin:(UIEdgeInsets) margin`

设置状态浮层view在渲染view上的边距。

19. showVideoDebugLog

接口详情：`(void) showVideoDebugLog:(BOOL) isShow`

是否显示播放状态统计及事件消息浮层view。

20. setAudioRoute

接口详情：(void)setAudioRoute:(TXAudioRouteType)audioRoute

设置声音播放模式(切换扬声器，听筒)。

其中，TXAudioRouteType 定义如下：

```
typedef NS_ENUM(NSUInteger, TXAudioRouteType) {  
    AUDIO_ROUTE_SPEAKER = 0, //扬声器  
    AUDIO_ROUTE_RECEIVER = 1, //听筒  
};
```

21. switchStream

接口详情：(int)switchStream:(NSString *)playUrl

flv 直播无缝切换。

参数	类型	说明
playUrl	NSString *	播放地址

返回值：0 为 ok

标准直播 集成(XCode)

最近更新时间：2018-09-21 19:17:25

SDK信息

您可以在腾讯云官网更新 [直播SDK](#)，目前直播SDK有如下几下版本：

版本类型	功能
直播精简版	支持推流、直播、点播
独立播放器版	支持直播、点播
短视频功能版	支持短视频和点播
全功能专业版	支持推流、直播、点播、连麦、短视频
商用企业版	在全功能专业版基础上增加动效贴纸、美瞳瘦脸、绿幕抠图功能

以专业版为例，下载完的SDK解压后有以下几个部分：



名称	修改日期	类型	大小
Demo	2017/8/11 21:09	文件夹	
SDK	2017/8/11 21:09	文件夹	
iOS 开发包使用指引.pdf	2017/8/11 20:15	Adobe Acrobat ...	77 KB

文件名	说明
SDK	包含 framework 的SDK目录
Demo	基于 framework 方式的简化 Demo，包含简单的 UI 界面和 SDK 的主要功能演示，使用 xcode可以快速导入并体验。
iOS 开发包使用指引.pdf	介绍SDK的基本功能

Xcode工程设置

一、支持平台

- SDK支持iOS 8.0以上系统

二、开发环境

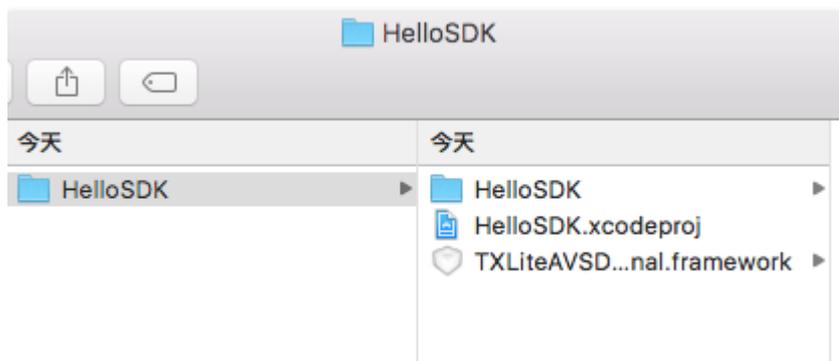
- Xcode 9或更高版本
- OS X 10.10或更高版本

三、Xcode工程设置

下面通过一个简单的iOS Application工程，说明如和在Xcode工程中配置SDK。

1、拷贝SDK文件

在本例中，新建一个名字叫做HelloSDK的iOS工程，将下载下来的 `TXLiteAVSDK_Professional.framework` 拷贝至工程目录。目录结构如下图所示：



2、添加Framework

在工程中添加 `TXLiteAVSDK_Professional.framework`，同时还要添加以下系统依赖库

1. libz.tbd
2. libc++.tbd
3. libresolv.tbd
4. libsqlite3.tbd
5. Accelerate.framework
6. GPUImage.framework(企业版需要)

3、添加头文件

在Build Settings->Search Paths->User Header Search Paths中添加头文件搜索路径。注意此项不是必须的，如果您没有添加TXLiteAVSDK_Professional的头文件搜索路径，则在引用SDK的相关头文件时，需要在头文件前增加"TXLiteAVSDK_Professional/"，如下所示：

```
#import "TXLiteAVSDK_Professional/TXLivePush.h"
```

四、验证

下面在HelloSDK的代码中，调用SDK的接口，获取SDK版本信息，以验证工程设置是否正确。

1、引用头文件

在ViewController.m开头引用SDK的头文件：

```
#import "TXLiteAVSDK_Professional/TXLiveBase.h"
```

2、添加调用代码

在viewDidLoad方法中添加代码：

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // 打印SDK的版本信息  
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);  
}
```

3、编译运行

如果前面各个步骤都操作正确的话，HelloSDK工程应该可以顺利编译通过。在Debug模式下运行APP，Xcode的Console窗格会打印出SDK的版本信息。

```
2017-08-11 16:16:15.767 HelloSDK[17929:7488566] SDK Version = 3.0.1185
```

至此，工程配置完成。

LOG打印

在TXLiveBase中可以设置log是否在控制台打印以及log的级别，具体代码如下：

- **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/log** 文件夹下。

如果您需要我们的技术支持，建议将开关打开，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];
```

- **SDK日志回调** 如果您需要将日志打印到自己的日志文件上，可以通过实现 `TXLiveBaseDelegate` 回调中的方法:

```
-(void) onLog:(NSString*)log LogLevel:(int)level WhichModule:(NSString*)module
```

推流(LivePusher)

最近更新时间：2018-08-10 16:20:46

基础知识

推流是指将音视频数据采集编码之后，推送到您指定的视频云平台上，这里涉及大量的音视频基础知识，而且需要长时间的打磨和优化才能达到符合预期的效果。

腾讯视频云 SDK 主要帮您解决在智能手机上的推流问题，它的接口非常简单易用，只需要一个推流URL就能驱动：



特别说明

- 不绑定腾讯云**

SDK 不绑定腾讯云，如果要推流到非腾讯云地址，请在推流前设置 TXLivePushConfig 中的 enableNearestIP 设置为 NO。但如果您要推流的地址为腾讯云地址，请务必在推流前将其设置为 YES，否则推流质量可能会因为运营商 DNS 不准确而受到影响。

- **x86 模拟器调试**

由于 SDK 大量使用iOS系统的高级特性，我们不能保证所有特性在x86环境的模拟器下都能正常运行，而且音视频是性能敏感的功能，模拟器下的表现跟真机会有很大的不同。所以，如果条件允许，推荐您尽量使用真机调试。

准备工作

- **获取开发包**

[下载](#) SDK 开发包，并按照[工程配置](#)指引将 SDK 嵌入您的 APP 开发工程。

- **获取测试URL**

开通直播服务后，可以使用 [直播控制台](#)>>[直播码接入](#)>>[推流生成器](#) 生成推流地址，详细信息可以参考 [获得推流播放URL](#)。

过期时间:	2018-01-12 23:59:59	直播码:	3891_ test	生成推流地址
推流地址 :	rtmp://3891.livepush.myqcloud.com/live/3891_test?bizid=3891&txSecret=1d2f4f66920c707aa01e5d327c725555&txTime=5A58DB7F			
播放地址 (RTMP) :	rtmp://3891.liveplay.myqcloud.com/live/3891_test			
播放地址 (FLV) :	http://3891.liveplay.myqcloud.com/live/3891_test.flv			
播放地址 (HLS) :	http://3891.liveplay.myqcloud.com/live/3891_test.m3u8			

代码对接

本篇攻略主要是面向[摄像头直播](#)的解决方案，该方案主要用于美女秀场直播、活动直播等场景，游戏直播请参考本目录下的同级文档。

step 1: 创建Pusher对象

先创建一个 **LivePush** 对象，我们后面主要用它来完成推流工作。

不过在创建 LivePush 对象之前，还需要您指定一个**LivePushConfig**对象，该对象的用途是决定 LivePush 推流时各个环节的配置参数，比如推流用多大的分辨率、每秒钟要多少帧画面（FPS）以及Gop（表示多少秒一个I帧）等等。

LivePushConfig 在alloc之后便已经装配了一些我们反复调过的参数，如果您不需要自己定制这些配置，简单的 alloc出来并塞给LivePush对象就可以了。如果您有相关领域的经验基础，需要对这些默认配置进行调整，可以阅读**进阶篇**中的内容。

```
// 创建 LivePushConfig 对象，该对象默认初始化为基础配置
TXLivePushConfig* _config = [[TXLivePushConfig alloc] init];
//在 _config中您可以对推流的参数（如：美白，硬件加速，前后置摄像头等）做一些初始化操作，需要注意 _config不能为nil
_txLivePush = [[TXLivePush alloc] initWithConfig: _config];
```

step 2: 渲染View

接下来我们要给摄像头的影像画面找个地方来显示，iOS系统中使用view作为基本的界面渲染单位，所以您只需要准备一个view传给 LivePush 对象的 **startPreview** 接口函数就可以了。

- **推荐的布局！**

实际上，SDK 的内部并不是直接把画面渲染在您提供的view上，而是在这个view之上创建一个用于 OpenGL渲染的子视图（ subView ），但是，这个渲染用的subView的大小会跟随您提供的view大小变化

而自动调整。



不过即如此，如果您想要在渲染画面之上实现弹幕、献花之类的UI控件，我们也推荐您“另起炉灶”（再创建一个平级的view），这样可以避免很多前后画面覆盖的问题。

• 如何做动画？

针对view做动画是比较自由的，不过请注意此处动画所修改的目标属性应该是transform属性而不是frame属性。

```
[UIView animateWithDuration:0.5 animations:^(
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
)];
```

step 3: 启动推流

经过 step1 和 step2 的准备之后，用下面这段代码就可以启动推流了：

```
NSString* rtmpUrl = @"rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
[_txLivePush startPreview:_myView]; // _myView 就是step2中需要您指定的view
[_txLivePush startPush:rtmpUrl];
```

- **startPush** 的作用是告诉 SDK 音视频流要推到哪个推流URL上去。
- **startPreview** 的参数就是step2中需要您指定的view，startPreview 的作用就是将界面view控件和LivePush对象关联起来，从而能够将手机摄像头采集到的画面渲染到屏幕上。

step 3+: 纯音频推流

如果你的直播场景是声音直播，那么需要更新下推流的配置信息。前面 step1 和 step2 准备步骤不变，使用以下代码设置纯音频推流并启动推流。

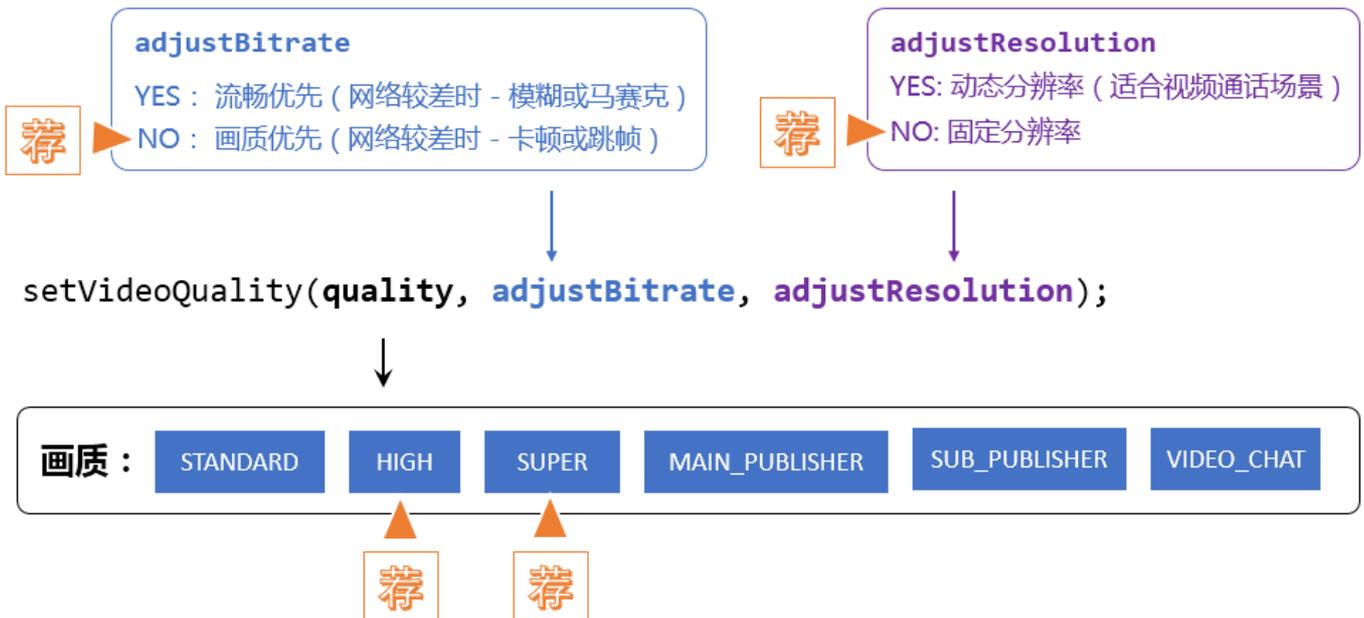
```
// 只有在推流启动前设置启动纯音频推流才会生效，推流过程中设置不会生效。
txLivePush.config.enablePureAudioPush = YES; // true 为启动纯音频推流，而默认值是 false ;
[_txLivePublisher setConfig:_config]; // 重新设置 config

NSString* rtmpUrl = @"rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
[_txLivePush startPush:rtmpUrl];
```

如果你启动纯音频推流，但是 rtmp、flv、hls 格式的播放地址拉不到流，那是因为线路配置问题，请提工单联系我们帮忙修改配置。

step 4: 设定清晰度

如果您是第一次接触音视频，**不推荐** 您自行设置分辨率、码率等视频参数，参数配置不当可能影响最终的画质表现，推荐您使用 TXLivePusher::setVideoQuality 接口的可以设定推流的画面清晰度：



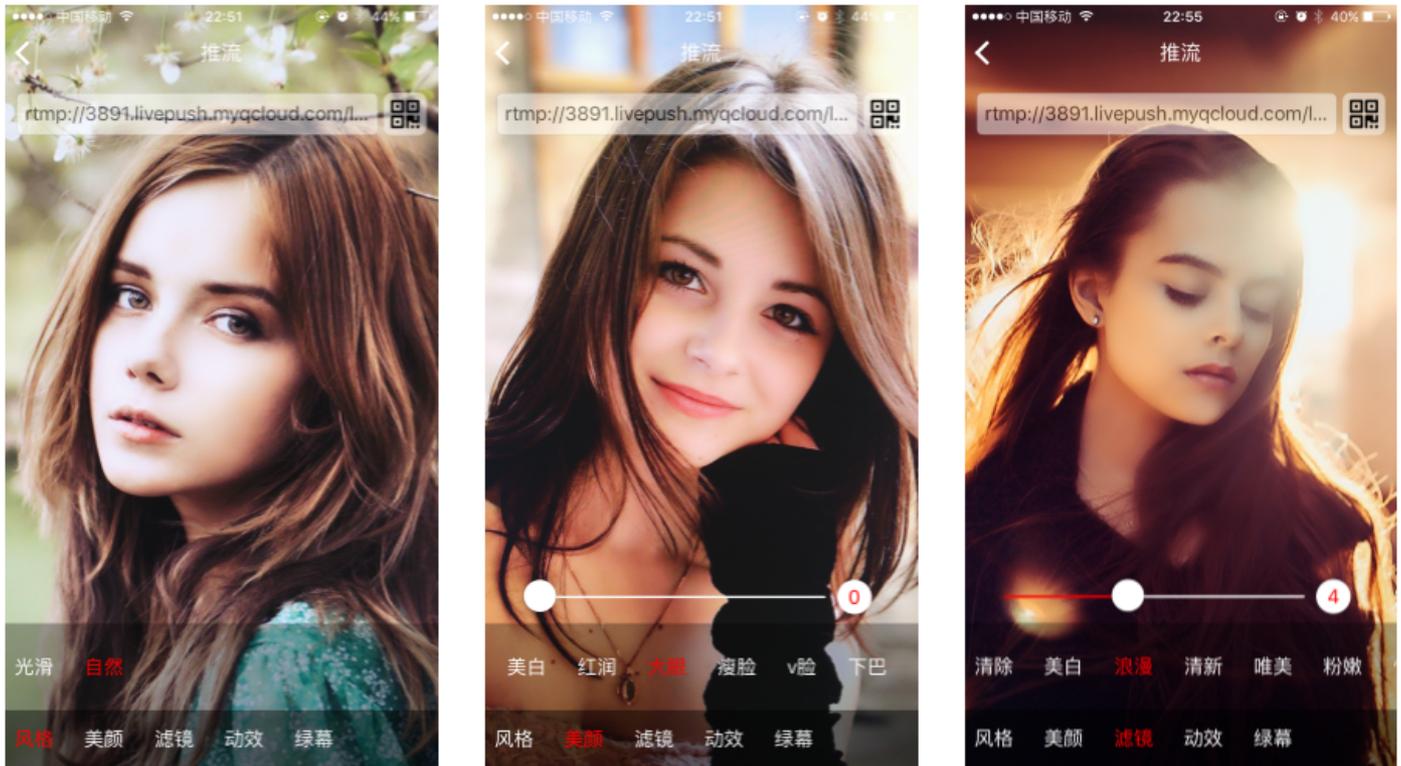
• 推荐参数设置

应用场景	quality	adjustBitrate	adjustResolution
秀场直播	VIDEO_QUALITY_HIGH_DEFINITION 或 VIDEO_QUALITY_SUPER_DEFINITION	NO	NO
手游直播	VIDEO_QUALITY_SUPER_DEFINITION	YES	YES
连麦（主画面）	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	YES	YES
连麦（小画面）	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	NO	NO
视频通话	VIDEO_QUALITY_REALTIEM_VIDEOCHAT	YES	YES

• 内部数据指标

quality	adjustBitrate	adjustResolution	码率范围	分辨率范围
STANDARD	YES	YES	300~800kbps	270x480 ~ 360x640
STANDARD	YES	NO	300~800kbps	360x640
STANDARD	NO	NO	800kbps	360x640
HIGH	YES	YES	600~1500kbps	360x640~540x960
HIGH	YES	NO	600~1500kbps	540x960
HIGH	NO	NO	1200kbps	540x960
SUPER	YES	YES	600~1800kbps	360x640~720x1280
SUPER	YES	NO	600~1800kbps	720x1280
SUPER	NO	NO	1800kbps	720x1280
MAIN_PUBLISHER	YES	YES	600~1500kbps	360x640~540x960
SUB_PUBLISHER	NO	NO	350kbps	320x480
VIDEOCHAT	YES	YES	200~800kbps	190x320~360x640

step 5: 美颜滤镜



• 美颜

setBeautyStyle 接口可以设置美颜风格、磨皮程度、美白级别和红润级别，配合 540 * 960 分辨率（setVideoQuality - VIDEO_QUALITY_HIGH_DEFINITION），可以达到最佳的画质效果：

```
// beautyStyle : 磨皮风格，目前支持 自然 和 光滑 两种。
// beautyLevel : 磨皮级别，取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。
// whitenessLevel : 美白级别，取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。
// ruddinessLevel : 红润级别，取值范围 0 ~ 9；0 表示关闭 1 ~ 9值越大 效果越明显。
(void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel
whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel;
```

• 滤镜

setFilter 接口可以设置滤镜效果，滤镜本身是一张直方图文件，我们设计师团队提供了八种素材，默认打包在了 Demo 中，您可以随意使用，不用担心版权问题。

setSpecialRatio 接口则可以设置滤镜的程度，从 0 到 1，越大滤镜效果越明显，默认取值 0.5。

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
if (path != nil && index != FilterType_None && _txLivePublisher != nil) {
```

```
path = [path stringByAppendingPathComponent:lookupFileName];
UIImage *image = [UIImage imageWithContentsOfFile:path];
[_txLivePublisher setFilter:image];
}
```

如果要自定义滤镜，一定要用 PNG 格式的图片，**不要用 JPG，不要用 JPG，不要用 JPG...**

step 6: 控制摄像头

• 切换前置或后置摄像头

默认是使用**前置**摄像头（可以通过修改 LivePushConfig 的配置项 frontCamera 来修改这个默认值），调用一次 switchCamera 切换一次，注意切换摄像头前必须保证 LivePushConfig 和 LivePush 对象都已经初始化。

```
// 默认是前置摄像头，可以通过修改 LivePushConfig 的配置项 frontCamera 来修改这个默认值
[_txLivePush switchCamera];
```

• 打开或关闭闪光灯

只有后置摄像头才可以打开闪光灯（您可以通过"TXLivePush.h"里面的frontCamera成员来确认当前摄像头是前置还是后置）

```
if(!frontCamera) {
    BOOL bEnable = YES;
    //bEnable为YES，打开闪光灯; bEnable为NO，关闭闪光灯
    BOOL result = [_txLivePush toggleTorch: bEnable];
    //result为YES，打开成功;result为NO，打开失败
}
```

• 自定义手动对焦

SDK 的iOS版本内部有默认的手动对焦逻辑，虽然功能没什么问题，但是经常由于屏幕的触控事件被抢占而无法发挥作用。同时，界面排布的自由我们原则上绝不能干预。

我们在新版本的 TXLivePush 里增加了一个 setFocusPosition 函数接口，您可以自己根据手指触控位置进行手动对焦。

```
// 如果客户调用这个接口，SDK内部触发对焦的逻辑将会停止，避免重复触发对焦逻辑
- (void)setFocusPosition:(CGPoint)touchPoint;
```

step 7: 设置Logo水印

最近相关政策规定，直播的视频必须要打上水印，所以这个之前看起来并不是特别起眼的功能现在要重点说一下：腾讯视频云目前支持两种水印设置方式：一种是在推流SDK进行设置，原理是在SDK内部进行视频编码前就给画面打上水印。另一种方式是在云端打水印，也就是云端对视频进行解析并添加水印Logo。

这里我们特别建议您使用**SDK添加水印**，因为在云端打水印有三个明显的问题：

- (1) 这是一种很耗云端机器的服务，而且不是免费的，会拉高您的费用成本；
- (2) 在云端打水印对于推流期间切换分辨率等情况的兼容并不理想，会有很多花屏的问题发生。
- (3) 在云端打水印会引入额外的3s以上的视频延迟，这是转码服务所引入的。

SDK所要求的水印图片格式为png，因为png这种图片格式有透明度信息，因而能够更好地处理锯齿等问题。（您可千万别把jpg图片在windows下改个后缀名就塞进去了，专业的png图标都是需要由专业的美工设计师处理的）

```
//设置视频水印
_config.watermark = [UIImage imageNamed:@"watermark.png"];
_config.watermarkPos = (CGPoint){10, 10};
```

step 8: 本地录制

使用 startRecord 接口可以启动本地录制，录制格式为 MP4，通过 videoPath 可以指定 MP4 文件的存放路径。

- 录制过程中请勿动态切换分辨率和软硬编，可能导致生成的视频异常
- 如果是云端录制，只需要在推流 URL 后面拼接 &record=mp4 即可，详情请参考[云端录制](#)。
- stopRecord 调用之后，录制出来的文件会通过 TXLiveRecordListener 通知出来。

```
-(int) startRecord:(NSString *)videoPath;
-(int) stopRecord;
```

step 9: 后台推流

常规模式下，App一旦切换到后台，摄像头的采集能力就会被 iOS 暂时停止掉，这就意味着 SDK 不能再继续采集并编码出音视频数据。如果我们什么都不做，那么故事将按照如下的剧本发展下去：

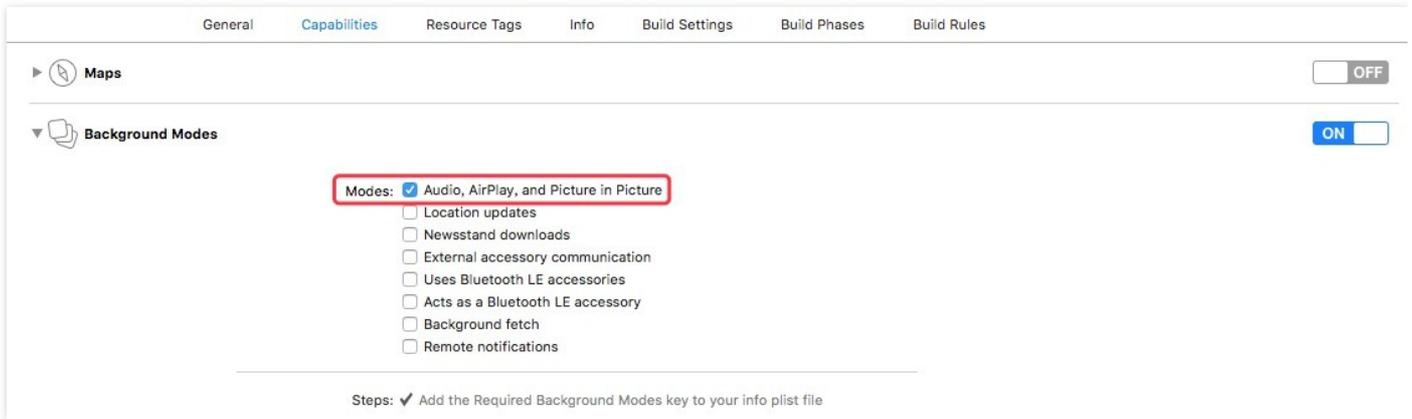
- 阶段一（切后台开始 -> 之后的10秒内）- CDN因为没有数据所以无法向观众提供视频流，观众看到画面卡主。
- 阶段二（10秒 -> 70秒内）- 观众端的播放器因为持续收不到直播流而直接推出，直播间已经人去楼空。
- 阶段三（70秒以后）- 推流的 RTMP 链路被服务器直接断掉，主播需要重新开启直播才能继续。

主播可能只是短暂接个紧急电话而已，但上述的交互体验显然会让观众全部离开直播间，怎么优化呢？

从 **SDK 1.6.1** 开始，我们引入了一种解决方案，如下是从观众端的视角看去，该方案可以达到的效果：



• 9.1) 调整XCode设置



• 9.2) 设置pauseImg

在开始推流前，使用 LivePushConfig 的 pauseImg 接口设置一张等待图片，图片含义推荐为“主播暂时离开一下下，稍后回来”。

```
// 300 为后台播放暂停图片的最长持续时间,单位是秒
_config.pauseTime = 300;
// 10 为后台播放暂停图片的帧率,最小值为5,最大值为20
_config.pauseFps = 10;
```

```
// 设置推流暂停时,后台播放的暂停图片, 图片最大尺寸不能超过1920*1920.
_config.pauseImg = [UIImage imageNamed:@"pause_publish.jpg"];
[_txLivePublisher setConfig:_config];
```

• 9.3) 设置App后台（短暂）运行

App 如果切后台后就彻底被休眠掉，那么 SDK 也就无法继续推流了，观众端就会看到直播间进入黑屏或者冻屏状态。我们可以使用下面的代码让 App 在切到后台后还可再跑几分钟，至少足够主播接听一个短暂的私人电话。

```
//先注册后退消息通知
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handleEnterBackground:)
name:UIApplicationDidEnterBackgroundNotification object:nil];

//收到通知后，调用beginBackgroundTaskWithExpirationHandler
-(void)handleEnterBackground:(NSNotification *)notification
{
[[UIApplication sharedApplication] beginBackgroundTaskWithExpirationHandler:^(
)];
}
```

• 9.4) 切后台处理

在上一步的 handleEnterBackground 里，调用 TXLivePush 的 pausePush 接口函数，之后 SDK 虽然采集不到摄像头的画面了，但可以用您刚才设置的 pauseImg 持续推流。

```
//切后台处理：在上一步的基础上补一句
-(void)handleEnterBackground:(NSNotification *)notification
{
[[UIApplication sharedApplication] beginBackgroundTaskWithExpirationHandler:^(
)];
[_txLivePush pausePush];
}
```

• 9.5) 切前台处理

等待 App 切回前台之后（在 handleEnterForeground 里），调用 TXLivePush 的 resumePush 接口函数，SDK 会恢复对摄像头画面的采集。要注意的是：由于 pausePush 和 resumePush 跟 SDK 内部状态息息相关，必须要 **配对使用**，否则会引入很多不必要的 BUG。

```
//切前台处理
-(void)handleEnterForeground:(NSNotification *)notification
{
[_txLivePush resumePush];
}
```

step 10: 卡顿预警提示

- 如果主播网络质量不好，我们应该怎么做？
- 主动降低清晰度来确保流畅性？这样观众端的感受就是模糊和马赛克。
- 主动丢掉一部分视频帧，以确保画面还能持续有一定的清晰度？这样观众端的感受就是持续卡顿。
- 以上都是我们不想要的？那怎么办？
- “既然马儿跑得快，又让马儿不吃草。”我们都知道，这是不可能的事情。

通过 TXLivePushListener 里的 onPlayEvent 可以捕获 **PUSH_WARNING_NET_BUSY** 事件，它代表当前主播的网络已经非常糟糕，出现此事件即代表观众端会出现卡顿。

此时可以提示主播“您当前的网络状况不佳，推荐您离 WiFi 近一点，尽量不要让 WiFi 穿墙”。

step 11: 横屏推流

大多数情况下，用户习惯以“竖屏持握”进行直播拍摄，观看端看到的也是竖屏样式；有时候用户在直播的时候需要更广的视角，则拍摄的时候需要“横屏持握”，这个时候其实是期望观看端能看到横屏画面，就需要做横屏推流，下面两幅示意图分别描述了横竖屏持握进行横竖屏推流在观众端看到的效果。



注意： 横屏推流和竖屏推流，观众端看到的图像的宽高比是不同的，竖屏9:16，横屏16：9。

要实现横屏推流，需要在两处进行设置：

- **调整观众端表现**

通过对 LivePushConfig 中的 **homeOrientation** 设置项进行配置，它控制的是观众端看到的视频宽高比是 **16:9** 还是 **6:19**，调整后的结果可以用播放器查看以确认是否符合预期。

- **调整主播端表现**

接下来就看主播本地渲染是否正常，这里可以通过 TXLivePush 中的 setRenderRotation 接口来设置主播看到的画面的旋转方向。此接口提供了 **0, 90, 180, 270** 四个参数供设置旋转角度。

step 12: 背景混音

SDK 1.6.1 开始支持背景混音，支持主播带耳机和不带耳机两种场景，您可以通过 TXLivePush 中的如下这组接口实现背景混音功能：

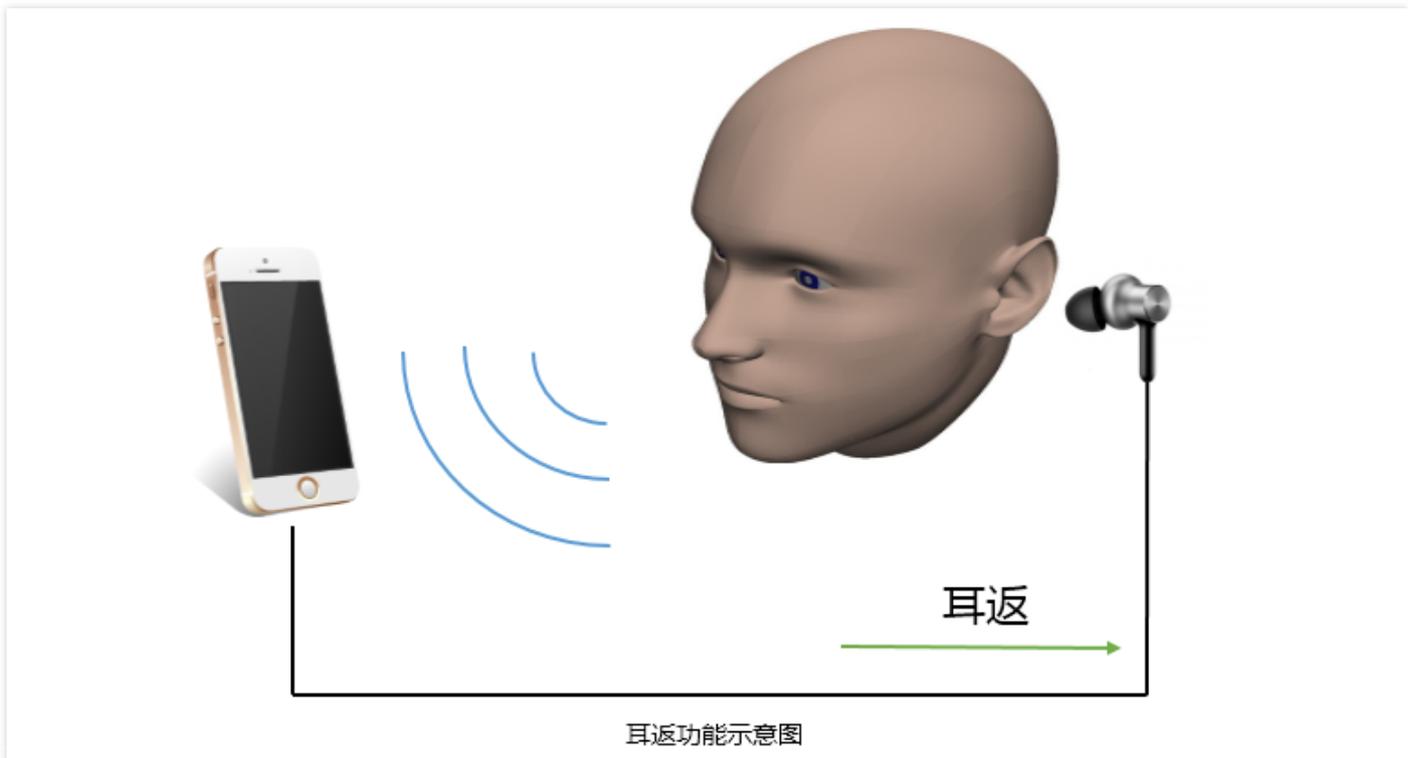
接口	说明
playBGM	通过path传入一首歌曲， 小直播Demo 中我们是从iOS的本地媒体库中获取音乐文件
stopBGM	停止播放背景音乐
pauseBGM	暂停播放背景音乐
resumeBGM	继续播放背景音乐
setMicVolume	设置混音时麦克风的音量大小，推荐在UI上实现相应的一个滑动条，由主播自己设置
setBGMVolume	设置混音时背景音乐的音量大小，推荐在UI上实现相应的一个滑动条，由主播自己设置

step 13: 耳返&混响

- **耳返**

指的是当主播带上耳机来唱歌时，耳机中要能实时反馈主播的声音，这是由于主播自己的声音是通过头部骨骼（固体）传入耳朵，而观众听到声音最终是通过空气介质传入耳朵，这两种方式听的声音是有很大差异的，因此

主播有需求直接听到观众端的效果。



在 `TXLivePushConfig` 中 `enableAudioPreview` 接口可以打开耳返（如果是连麦，推荐大主播开，小主播就不要开了，实时音视频通话时开耳返会很奇怪。）

• 混响

指的是耳返的时候，声音要加入一些特效，比如KTV、会堂、磁性、金属等等，这个效果最终也会作用到观众端，让主播的歌声更加有味道。通过 `TXLivePush` 的成员函数 `setReverbType` (1.9.2 开始支持) 可以设置混响特效。

目前可以支持的混响特效有：KTV、小房间、大会堂、低沉、洪亮、金属声 以及 磁性 等 6 种。

step 14: 结束推流

结束推流很简单，不过要做好清理工作，因为用于推流的 `TXLivePush` 对象同一时刻只能有一个在运行，所以清理工作不当会导致下次直播遭受不良的影响。

```
//结束推流，注意做好清理工作
- (void)stopRtmpPublish {
    [_txLivePush stopPreview];
    [_txLivePush stopPush];
    _txLivePush.delegate = nil;
}
```

发送消息

此功能可以在推流端将一些自定义 message 随着音视频线路直接下发到观众端，适用场景例如：

- (1) 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”完美同步。
- (2) 秀场直播：推流端将**歌词**下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
- (3) 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈划线。

```
[_answerPusher sendMessage:[mesg dataUsingEncoding:NSUTF8StringEncoding]];
```

使用 TXLivePlayer 的 onPlayEvent (PLAY_EVT_GET_MESSAGE) 可以用来接收消息。

事件处理

1. 事件监听

SDK 通过 TXLivePushListener 代理来监听推流相关的事件，注意 TXLivePushListener 只能监听到得到 PUSH_ 前缀的推流事件。

2. 常规事件

一次成功的推流都会通知的事件，比如收到1003就意味着摄像头的画面会开始渲染了

事件ID	数值	含义说明
PUSH_EVT_CONNECT_SUCC	1001	已经成功连接到腾讯云推流服务器
PUSH_EVT_PUSH_BEGIN	1002	与服务器握手完毕,一切正常,准备开始推流
PUSH_EVT_OPEN_CAMERA_SUCC	1003	推流器已成功打开摄像头 (Android部分手机这个过程需要1-2秒)

3. 错误通知

SDK发现了一些严重问题，推流无法继续了，比如用户禁用了APP的Camera权限导致摄像头打不开。

事件ID	数值	含义说明
------	----	------

事件ID	数值	含义说明
PUSH_ERR_OPEN_CAMERA_FAIL	-1301	打开摄像头失败
PUSH_ERR_OPEN_MIC_FAIL	-1302	打开麦克风失败
PUSH_ERR_VIDEO_ENCODE_FAIL	-1303	视频编码失败
PUSH_ERR_AUDIO_ENCODE_FAIL	-1304	音频编码失败
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	不支持的视频分辨率
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	不支持的音频采样率
PUSH_ERR_NET_DISCONNECT	-1307	网络断连,且经三次抢救无效,可以放弃治疗,更多重试请自行重启推流

4. 警告事件

SDK发现了一些问题，但这并不意味着无可救药，很多 WARNING 都会触发一些重试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复，所以，千万不要“小题大做”哦。

- **WARNING_NET_BUSY**

主播网络不给力，如果您需要UI提示，这个 warning 相对比较有用（step10）。

- **WARNING_SERVER_DISCONNECT**

推流请求被后台拒绝了，出现这个问题一般是由于推流地址里的 txSecret 计算错了，或者是推流地址被其他人占用了（一个推流URL同时只能有一个端推流）。

事件ID	数值	含义说明
PUSH_WARNING_NET_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻
PUSH_WARNING_RECONNECT	1102	网络断连,已启动自动重连(自动重连连续失败超过三次会放弃)
PUSH_WARNING_HW_ACCELERATION_FAIL	1103	硬编码启动失败，采用软编码
PUSH_WARNING_DNS_FAIL	3001	RTMP -DNS解析失败（会触发重试流程）
PUSH_WARNING_SEVER_CONN_FAIL	3002	RTMP服务器连接失败（会触发重试流程）
PUSH_WARNING_SHAKE_FAIL	3003	RTMP服务器握手失败（会触发重试流程）

事件ID	数值	含义说明
PUSH_WARNING_SERVER_DISCONNECT	3004	RTMP服务器主动断开连接（会触发重试流程）

直播(LivePlayer)

最近更新时间：2018-09-26 10:20:50

基础知识

本文主要介绍视频云 SDK 的直播播放功能，在此之前，先了解如下一些基本知识会大有裨益：

• 直播和点播

直播 (LIVE) 的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。

点播 (VOD) 的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放，播放中您可以通过进度条控制播放位置，腾讯视频和优酷土豆等视频网站上的视频观看就是典型的点播场景。

• 协议的支持

通常使用的直播协议如下，App端推荐使用 FLV 协议的直播地址(以“http”打头，以“.flv”结尾)：

直播协议	优点	缺点	播放延迟
FLV	成熟度高、高并发无压力	需集成SDK才能播放	2s - 3s
RTMP	优质线路下理论延迟最低	高并发情况下表现不佳	1s - 3s
HLS(m3u8)	手机浏览器支持度高	延迟非常高	10s - 30s

特别说明

• 是否有限制？

视频云 SDK **不会对** 播放地址的来源做限制，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV、RTMP 和 HLS (m3u8) 三种格式的直播地址，以及 MP4、HLS (m3u8) 和 FLV 三种格式的点播地址。

• 历史因素

SDK 早期版本只有 TXLivePlayer 一个 Class 承载直播和点播功能，但是由于点播功能越做越多，我们最终在 SDK 3.5 版本开始，将点播功能单独分离出来，交由 TXVodPlayer 来负责。但是为了保证编译通过，您在 TXLivePlayer 中依然可以看到类似 seek 等点播才具备的功能。

对接攻略

step 1: 创建Player

视频云 SDK 中的 TXLivePlayer 模块负责实现直播播放功能。

```
TXLivePlayer _txLivePlayer = [[TXLivePlayer alloc] init];
```

step 2: 渲染View

接下来我们要给播放器的视频画面找个地方来显示，iOS系统中使用 view 作为基本的界面渲染单位，所以您只需要准备一个 view 并调整好布局就可以了。

```
//用 setupVideoWidget 给播放器绑定决定渲染区域的view，其首个参数 frame 在 1.5.2 版本后已经被废弃
[_txLivePlayer setupVideoWidget:CGRectMake(0, 0, 0, 0) containerView:_myView insertIndex:0];
```

内部原理上讲，播放器并不是直接把画面渲染到您提供的 view（示例代码中的 _myView）上，而是在这个view之上创建一个用于OpenGL渲染的子视图（subView）。

如果您要调整渲染画面的大小，只需要调整您所常见的 view 的大小和位置即可，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。



如何做动画？

针对view做动画是比较自由的，不过请注意此处动画所修改的目标属性应该是 **transform** 属性而不是 frame 属性。

```
[UIView animateWithDuration:0.5 animations:^(
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
)];
```

step 3: 启动播放

```
NSString* flvUrl = @"http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
[_txLivePlayer startPlay:flvUrl type:PLAY_TYPE_LIVE_FLV];
```

可选值	枚举值	含义
PLAY_TYPE_LIVE_RTMP	0	传入的URL为RTMP直播地址
PLAY_TYPE_LIVE_FLV	1	传入的URL为FLV直播地址
PLAY_TYPE_LIVE_RTMP_ACC	5	低延迟链路地址（仅适合于连麦场景）
PLAY_TYPE_VOD_HLS	3	传入的URL为HLS(m3u8)播放地址

关于HLS(m3u8)

在 APP 上我们不推荐使用 HLS 这种播放协议播放直播视频源（虽然它很适合用来做点播），因为延迟太高，在 APP 上推荐使用 LIVE_FLV 或者 LIVE_RTMP 播放协议。

step 4: 画面调整

- **view : 大小和位置**

如需修改画面的大小及位置，直接调整 setupVideoWidget 的参数 view 的大小和位置，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。

- **setRenderMode : 铺满or适应**

可选值	含义
-----	----

可选值	含义
RENDER_MODE_FILL_SCREEN	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全。
RENDER_MODE_FILL_EDGE	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边。

• setRenderRotation : 画面旋转

可选值	含义
RENDER_ROTATION_PORTRAIT	正常播放（Home键在画面正下方）
RENDER_ROTATION_LANDSCAPE	画面顺时针旋转270度（Home键在画面正左方）



最长边填充



完全填充



横屏模式

step 5: 暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这跟点播是有很大的不同的（点播播放器的暂停和继续与播放本地视频文件时的表现相同）。

```
// 暂停
[_txLivePlayer pause];
// 恢复
[_txLivePlayer resume];
```

step 6: 结束播放

结束播放时，如果要推出当前的UI界面，要记得用 **removeVideoWidget** 销毁view控件，否则会产生内存泄露或闪屏问题。

```
// 停止播放
[_txLivePlayer stopPlay];
[_txLivePlayer removeVideoWidget]; // 记得销毁view控件
```

step 7: 消息接收

此功能可以在推流端将一些自定义 message 随着音视频线路直接下发到观众端，适用场景例如：

- (1) 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”完美同步。
- (2) 秀场直播：推流端将**歌词**下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
- (3) 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈划线。

通过如下方案可以使用此功能：

- TXLivePlayConfig 中的 **enableMessage** 开关置为 **YES**。
- TXLivePlayer 通过 **TXLivePlayListener** 监听消息，消息编号：**PLAY_EVT_GET_MESSAGE (2012)**

```
-(void) onPlayEvent:(int)EvtID withParam:(NSDictionary *)param {
    [self asyncRun:^(
        if (EvtID == PLAY_EVT_GET_MESSAGE) {
            dispatch_async(dispatch_get_main_queue(), ^{
                if ([_delegate respondsToSelector:@selector(onPlayerMessage:)]) {
                    [_delegate onPlayerMessage:param[@"EVT_GET_MSG"]];
                }
            });
        }
    )];
}
```

step 8: 屏幕截图

通过调用 **snapshot** 您可以截取当前直播画面为一帧屏幕，此功能只会截取当前直播流的视频画面，如果您需要截取当前的整个 UI 界面，请调用 iOS 的系统 API 来实现。



屏幕截图

仅截取视频画面

step 9: 截流录制

截流录制是直播播放场景下的一种扩展功能：观众在观看直播时，可以通过单击录制按钮把一段直播的内容录制下来，并通过视频分发平台（比如腾讯云的点播系统）发布出去，这样就可以在微信朋友圈等社交平台上以 UGC 消息的形式进行传播。



```
//如下代码用于展示直播播放场景下的录制功能
//
//指定一个 TXVideoRecordListener 用于同步录制的进度和结果
_txLivePlayer.recordDelegate = recordListener;
//启动录制，可放于录制按钮的响应函数里，目前只支持录制视频源，弹幕消息等等目前还不支持
[_txLivePlayer startRecord: RECORD_TYPE_STREAM_SOURCE];
// ...
// ...
//结束录制，可放于结束按钮的响应函数里
[_txLivePlayer stopRecord];
```

- 录制的进度以时间为单位，由 TXVideoRecordListener 的 onRecordProgress 通知出来。
- 录制好的文件以 MP4 文件的形式，由 TXVideoRecordListener 的 onRecordComplete 通知出来。
- 视频的上传和发布由 TXUGCPublish 负责，具体使用方法可以参考 [短视频-文件发布](#)。

step 10: 清晰度无缝切换

日常使用中，网络情况在不断发生变化。在网络较差的情况下，最好适度降低画质，以减少卡顿；反之，网速比较好，可以观看更高画质。

传统切流方式一般是重新播放，会导致切换前后画面衔接不上、黑屏、卡顿等问题。使用无缝切换方案，在不中断直播的情况下，能直接切到另条流上。

清晰度切换在直播开始后，任意时间都可以调用。调用方式如下

```
// 正在播放的是流http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab11e791eae435c87f075e.flv,
// 现切换到码率为900kbps的新流上
[_txLivePlayer switchStream:@"http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab11e791eae435c87f075e_900.flv"];
```

step 11: 直播回看

时移功能是腾讯云推出的特色能力，可以在直播过程中，随时观看回退到任意直播历史时间点，并能在此时间点一直观看直播。非常适合游戏、球赛等互动性不高，但观看连续性较强的场景。

```
// 设置直播回看前，先调用startPlay
// 开始播放 ...
[TXLiveBase setAppID:@"1253131631"]; // 配置appId
[_txLivePlayer prepareLiveSeek]; // 后台请求直播起始时间
```

配置正确后，在PLAY_EVT_PLAY_PROGRESS事件里，当前进度就不是从0开始，而是根据实际开播时间计算而来。调用seek方法，就能从历史事件点重新直播

```
[_txLivePlayer seek:600]; // 从第10分钟开始播放
```

接入时移需要在后台打开2处配置：

1. 录制：配置时移时长、时移储存时长
2. 播放：时移获取元数据

时移功能处于公测申请阶段，如您需要可提交工单申请使用。

延时调节

腾讯云 SDK 的直播播放（LVB）功能，并非基于 ffmpeg 做二次开发，而是采用了自研的播放引擎，所以相比于开源播放器，在直播的延迟控制方面有更好的表现，我们提供了三种延迟调节模式，分别适用于：秀场，游戏以及混合场景。

• 三种模式的特性对比

控制模式	卡顿率	平均延迟	适用场景	原理简述
极速模式	较流畅 偏高	2s- 3s	美女秀场（冲顶大会）	在延迟控制上有优势，适用于对延迟大小比较敏感的场景
流畅模式	卡顿率 最低	>= 5s	游戏直播（企鹅电竞）	对于超大码率的游戏直播（比如吃鸡）非常适合，卡顿率最低
自动模式	网络自 适应	2s-8s	混合场景	观众端的网络越好，延迟就越低；观众端网络越差，延迟就越高

• 三种模式的对接代码

```
TXLivePlayConfig* _config = [[TXLivePlayConfig alloc] init];
//自动模式
_config.bAutoAdjustCacheTime = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 5;
//极速模式
_config.bAutoAdjustCacheTime = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 1;
//流畅模式
```

```
_config.bAutoAdjustCacheTime = NO;
_config.minAutoAdjustCacheTime = 5;
_config.maxAutoAdjustCacheTime = 5;
```

```
[_txLivePlayer setConfig:_config];
```

```
//设置完成之后再启动播放
```

更多关于卡顿和延迟优化的技术知识，可以阅读[视频卡顿怎么办？](#)

超低延时播放

支持 **400ms** 左右的超低延迟播放是腾讯云直播播放器的一个特点，它可以用于一些对时延要求极为苛刻的场景，比如[远程夹娃娃](#)或者[主播连麦](#)，等等，关于这个特性，您需要知道：

- **该功能是不需要开通的**

该功能并不需要提前开通，但是要求直播流必须位于腾讯云，跨云商实现低延时链路的难度不仅仅是技术层面的。

- **播放地址需要带防盗链**

播放URL 不能用普通的 CDN URL，必须要带防盗链签名，防盗链签名的计算方法见 [txTime&txSecret](#)。

- **播放类型需要指定ACC**

在调用 startPlay 函数时，需要指定 type 为 **PLAY_TYPE_LIVE_RTMP_ACC**，SDK 会使用 RTMP-UDP 协议拉取直播流。

- **该功能有并发播放限制**

目前最多同时 **10 路** 并发播放，设置这个限制的原因并非技术能力限制，而是希望您只考虑在互动场景中使用（比如连麦时只给主播使用，或者夹娃娃直播中只给操控娃娃机的玩家使用），避免因盲目追求低延时而产生不必要的费用损失（低延迟线路的价格要贵于CDN线路）。

- **Obs的延时是不达标的**

推流端如果是 [TXLivePusher](#)，请使用 [setVideoQuality](#) 将 quality 设置为 MAIN_PUBLISHER 或者 VIDEO_CHAT。如果是 Windows 端，请使用我们的 [Windows SDK](#)，Obs 的推流端积压比较严重，是无法达到低延时效果的。

- **该功能按播放时长收费**

本功能按照播放时长收费，费用跟拉流的路数有关系，跟音视频流的码率无关，具体价格请参考[价格指引](#)。

SDK事件监听

您可以为 TXLivePlayer 对象绑定一个 **TXLivePlayListener**，之后SDK 的内部状态信息均会通过 onPlayEvent（事件通知）和 onNetStatus（状态反馈）通知给您。

1. 播放事件

事件ID	数值	含义说明
PLAY_EVT_CONNECT_SUCC	2001	已经连接服务器
PLAY_EVT_RTMP_STREAM_BEGIN	2002	已经连接服务器，开始拉流（仅播放RTMP地址时会抛送）
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包(IDR)
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始，如果有转菊花什么的这个时候该停了
PLAY_EVT_PLAY_LOADING	2007	视频播放loading，如果能够恢复，之后会有BEGIN事件
PLAY_EVT_GET_MESSAGE	2012	用于接收夹在音视频流中的消息，详情参考 消息接受

- **不要在收到 PLAY_LOADING 后隐藏播放画面**

因为PLAY_LOADING -> PLAY_BEGIN 的时间长短是不确定的，可能是 5s 也可能是 5ms，有些客户考虑在LOADING 时隐藏画面，BEGIN 时显示画面，会造成严重的画面闪烁（尤其是直播场景下）。推荐的做法是在视频播放画面上叠加一个半透明的 loading 动画。

2. 结束事件

事件ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放

- **如何判断直播已结束？**

基于各种标准的实现原理不同，很多直播流通常没有结束事件（2006）抛出，此时可预期的表现是：主播结束推流后，SDK 会很快发现数据流拉取失败（WARNING_RECONNECT），然后开始重试，直至三次重试失败后抛出PLAY_ERR_NET_DISCONNECT 事件。

所以 2006 和 -2301 都要监听，用来作为直播结束的判定事件。

3. 警告事件

如下的这些事件您可以不用关心，我们只是基于白盒化的SDK设计理念，将事件信息同步出来

事件ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_RECV_DATA_LAG	2104	网络来包不稳: 可能是下行带宽不足, 或由于主播端出流不均匀
PLAY_WARNING_VIDEO_PLAY_LAG	2105	当前视频播放出现卡顿
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败, 采用软解
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	当前视频帧不连续, 可能丢帧
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS解析失败 (仅播放RTMP地址时会抛送)
PLAY_WARNING_SEVER_CONN_FAIL	3002	RTMP服务器连接失败 (仅播放RTMP地址时会抛送)
PLAY_WARNING_SHAKE_FAIL	3003	RTMP服务器握手失败 (仅播放RTMP地址时会抛送)

视频宽高

视频的宽高 (分辨率) 是多少?

站在 SDK 的角度, 如果只是拿到一个 URL 字符串, 它是回答不出这个问题的。要知道视频画面的宽和高各是多少个 pixel, SDK 需要先访问云端服务器, 直到加载到足够能够分析出视频画面大小的信息才行, 所以对于视频信息而言, SDK 也只能以通知的方式告知您的应用程序。

onNetStatus 通知每秒都会被触发一次, 目的是实时反馈当前的推流器状态, 它就像汽车的仪表盘, 可以告知您目前SDK内部的一些具体情况, 以便您能对当前网络状况和视频信息等有所了解。

评估参数	含义说明
------	------

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时CPU使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 - 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度
NET_STATUS_NET_JITTER	网络抖动情况，抖动越大，网络越不稳定
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_CACHE_SIZE	缓冲区 (jitterbuffer) 大小，缓冲区当前长度为 0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器IP

点播(VodPlayer)

最近更新时间：2018-08-10 16:20:59

基础知识

本文主要介绍视频云 SDK 的点播播放功能，在此之前，先了解如下一些基本知识会大有裨益：

• 直播和点播

直播 (LIVE) 的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。

点播 (VOD) 的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放，播放中您可以通过进度条控制播放位置，腾讯视频和优酷土豆等视频网站上的视频观看就是典型的点播场景。

• 协议的支持

通常使用的点播协议如下，现在比较流行的是HLS(以“http”打头，以“.m3u8”结尾)的点播地址：

点播协议	优点	缺点
HLS(m3u8)	手机浏览器支持度高	大量小分片的文件组织形式，错误率和维护成本均高于单一文件
MP4	手机浏览器支持度高	格式过于复杂和娇贵，容错性很差，对播放器的要求很高
FLV	格式简单问题少，适合直播转录制场景	手机浏览器支持差，需集成SDK才能播放

特别说明

视频云 SDK **不会对** 播放地址的来源做限制，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV、RTMP 和 HLS (m3u8) 三种格式的直播地址，以及 MP4、HLS (m3u8) 和 FLV 三种格式的点播地址。

对接攻略

step 1: 创建Player

视频云 SDK 中的 TXVodPlayer 模块负责实现点播播放功能。

```
TXVodPlayer *_txVodPlayer = [[TXVodPlayer alloc] init];
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

step 2: 渲染View

接下来我们要给播放器的视频画面找个地方来显示，iOS系统中使用 view 作为基本的界面渲染单位，所以您只需要准备一个 view 并调整好布局就可以了。

```
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

内部原理上讲，播放器并不是直接把画面渲染到您提供的 view（示例代码中的 _myView）上，而是在这个view之上创建一个用于OpenGL渲染的子视图（subView）。

如果您要调整渲染画面的大小，只需要调整你所常见的 view 的大小和位置即可，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。



如何做动画？

针对view做动画是比较自由的，不过请注意此处动画所修改的目标属性应该是 **transform** 属性而不是 frame 属性。

```
[UIView animateWithDuration:0.5 animations:^(
    _myView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
)];
```

step 3: 启动播放

TXVodPlayer支持两种播放模式，您可以根据需要自行选择

1. 通过url方式

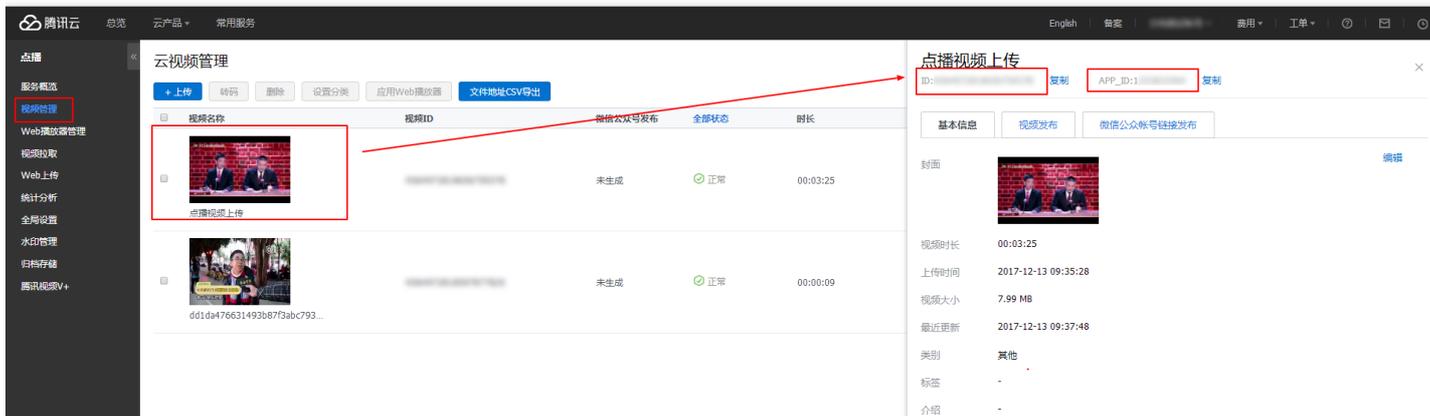
TXVodPlayer 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startPlay 函数即可。

```
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startPlay:url];
```

2. 通过fileId方式

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788;
p.fileId = @"4564972819220421305";
[_txVodPlayer startPlayWithParams:p];
```

在[点播视频管理](#)找到对应的文件。点开后可在右侧视频详情中，可以看到appId和fileId。



通过fileId方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或fileId不存在，则会收到 `PLAY_ERR_GET_PLAYINFO_FAIL` 事件，反之收到 `PLAY_EVT_GET_PLAYINFO_SUCC` 表示请求成功。

step 4: 画面调整

• view : 大小和位置

如需修改画面的大小及位置，直接调整 `setupVideoWidget` 的参数 `view` 的大小和位置，SDK 会让视频画面跟着

您的 view 的大小和位置进行实时的调整。

- **setRenderMode** : 铺满or适应

可选值	含义
RENDER_MODE_FILL_SCREEN	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全。
RENDER_MODE_FILL_EDGE	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边。

- **setRenderRotation** : 画面旋转

可选值	含义
HOME_ORIENTATION_RIGHT	home在右边
HOME_ORIENTATION_DOWN	home在下面
HOME_ORIENTATION_LEFT	home在左边
HOME_ORIENTATION_UP	home在上面



最长边填充



完全填充



横屏模式

step 5: 播放控制

```
// 调整进度
[_txVodPlayer seek:slider.value];
// 暂停播放
[_txVodPlayer pause];
// 恢复播放
[_txVodPlayer resume];
```

step 6: 结束播放

结束播放时，如果要退出当前的UI界面，要记得用 **removeVideoWidget** 销毁view控件，否则会产生内存泄露或闪屏问题。

```
// 停止播放
[_txVodPlayer stopPlay];
[_txVodPlayer removeVideoWidget]; // 记得销毁view控件
```

step 7: 屏幕截图

通过调用 **snapshot** 您可以截取当前视频为一帧画面，此功能只会截取当前直播流的视频画面，如果您需要截取当前的整个 UI 界面，请调用 iOS 的系统 API 来实现。



→ 屏幕截图
仅截取视频画面

step 8: 变速播放

点播播放器支持变速播放，通过接口 `setRate` 设置点播播放速率来完成，支持快速与慢速播放，如0.5X、1.0X、1.2X、2X等。



→ 变速播放
支持加速和慢播

```
//设置1.2倍速播放
[ _txVodPlayer setRate:1.2];
// ...
```

```
//开始播放  
[_txVodPlayer startPlay:url];
```

step 9: 本地缓存 [UGC版本暂不支持]

在短视频播放场景中，视频文件的本地缓存是很刚需的一个特性，对于普通用户而言，一个已经看过的视频再次观看时，不应该再消耗一次流量。

• 格式支持

SDK 支持 HLS(m3u8) 和 MP4 两种常见点播格式的缓存功能。

• 何时开启?

SDK 并不默认开启缓存功能，对于用户回看率不高的场景，也并不推荐您开启此功能。

• 如何开启？

开启此功能需要配置两个参数：本地缓存目录及需要缓存的视频个数。

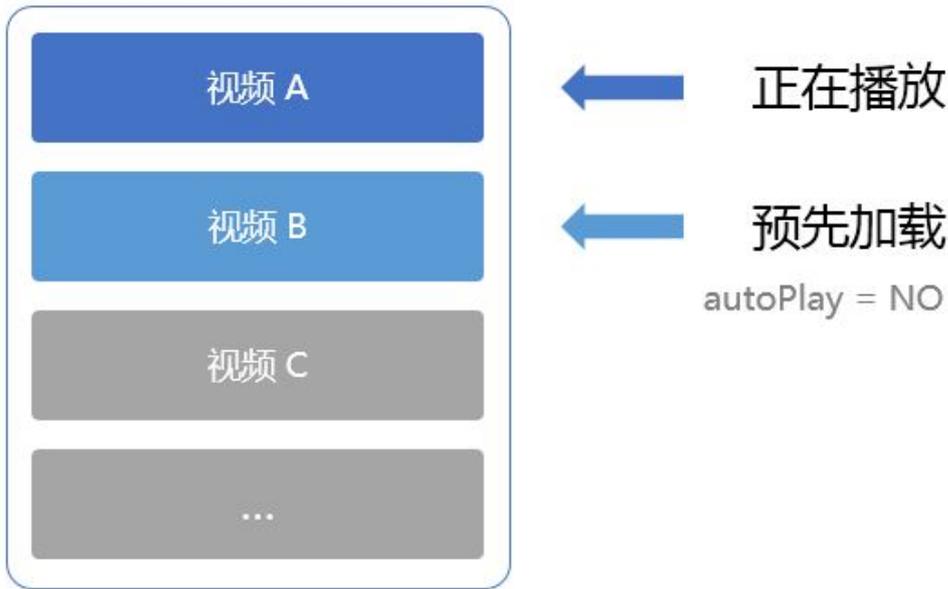
```
TXVodPlayConfig _config = [[TXVodPlayConfig alloc] init];  
  
// 设置缓存路径  
_config.cacheFolderPath =  
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];  
  
// 设置最多缓存多少个文件，避免缓存太多数据  
_config.maxCacheItems = 10;  
  
[_txVodPlayer setConfig: _config];  
// ...  
//开始播放  
[_txVodPlayer startPlay:playUrl];
```

step 10: 预加载

在短视频播放场景中，预加载功能对于流畅的观看体验很有帮助：在观看当前视频的同时，在后台加载即将要播放的下一个视频URL，这样一来，当用户真正切换到下一个视频时，已经不需要从头开始加载了，而是可以做到立刻播放。

这就是视频播放中无缝切换的背后技术支撑，您可以使用 TXVodPlayer 中的 isAutoPlay 开关来实现这个功能，具体做法如下：

视频列表



// 播放视频A: 如果将 isAutoPlay 设置为 YES , 那么 startPlay 调用会立刻开始视频的加载和播放

```
NSString* url_A = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
_player_A.isAutoPlay = YES;
[_player_A startPlay:url_A];
```

// 在播放视频 A 的同时, 预加载视频 B , 做法是将 isAutoPlay 设置为 NO

```
NSString* url_B = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
_player_B.isAutoPlay = NO;
[_player_B startPlay:url_B];
```

等到视频 A 播放结束, 自动 (或者用户手动切换到) 视频B时, 调用 resume 函数即可实现立刻播放。

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*)param
{
    // 在视频 A 播放结束的时候, 直接启动视频 B 的播放, 可以做到无缝切换
    if (EvtID == PLAY_EVT_PLAY_END) {
        [_player_A stopPlay];
        [_player_B setupVideoWidget:mVideoContainer atIndex:0];
        [_player_B resume];
    }
}
```

step 11: 贴片广告

autoPlay 还可以用来做贴片广告功能，由于设置了 autoPlay 为 NO 之后，播放器会立刻加载但又不会立刻播放，因此可以在此时展示贴片广告，等广告播放结束，在使用 resume 函数立即开始视频的播放。

step 12: 加密播放 [UGC版本暂不支持]

视频加密方案主要用于在线教育等需要对视频版权进行保护的场景。如果要对您的视频资源进行加密保护，就不仅仅需要在播放器上做改造，还需要对视频源本身进行加密转码，亦需要您的后台和终端研发工程师都参与其中。在[视频加密解决方案](#)中您会了解到全部细节内容。

目前 TXVodPlayer 也是支持加密播放的，您可以使用通过 URL 携带身份认证信息的方案，该种方案下 SDK 的调用方式跟普通情况没有什么区别。您也可以使用 Cookie 携带身份认证信息的方案，该种方案下，需要您通过 TXVodPlayConfig 中的 headers 字段设置 cookie 信息于 http 请求头中。

step 13: HTTP-REF [UGC版本暂不支持]

TXVodPlayConfig 中的 headers 可以用来设置 http 请求头，比如常用的防止 URL 被到处拷贝的 Referer 字段（腾讯云可以提供更加安全的签名防盗链方案），以及用于验证客户端身份信息的 Cookie 字段。

step 14: 硬件加速

对于蓝光级别（1080p）的画质，简单采用软件解码的方式很难获得较为流畅的播放体验，所以如果您的场景是以游戏直播为主，一般都推荐开启硬件加速。

软解和硬解的切换需要在切换之前先stopPlay，切换之后再startPlay，否则会产生比较严重的花屏问题。

```
[_txVodPlayer stopPlay];
_txVodPlayer.enableHWAceleration = YES;
[_txVodPlayer startPlay:_flvUrl type:_type];
```

step 15: 多码率文件 [UGC版本暂不支持]

SDK支持hls的多码率格式，方便用户切换不同码率的播放流。在收到PLAY_EVT_PLAY_BEGIN事件后，可以通过下面方法获取多码率数组

```
NSArray *bitrates = [_txVodPlayer supportedBitrates]; //获取多码率数组
```

在播放过程中，可以随时通过 `-[TXVodPlayer setBitrateIndex:]` 切换码率。切换过程中，会重新拉取另一条流的数据，因此会有稍许卡顿。SDK针对腾讯云的多码率文件做过优化，可以做到切换无卡顿。

进度展示

点播进度分为两个指标：**加载进度** 和 **播放进度**，SDK 目前是以事件通知的方式将这两个进度实时通知出来的。

您可以为 TXVodPlayer 对象绑定一个 **TXVodPlayListener** 监听器，进度通知会通过 **PLAY_EVT_PLAY_PROGRESS** 事件回调到您的应用程序，该事件的附加信息中即包含上述两个进度指标。



```

-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*)param {
if (EvtID == PLAY_EVT_PLAY_PROGRESS) {
// 加载进度, 单位是秒, 小数部分为毫秒
float playable = [param[EVT_PLAYABLE_DURATION] floatValue];
[_loadProgressBar setValue:playable];

// 播放进度, 单位是秒, 小数部分为毫秒
float progress = [param[EVT_PLAY_PROGRESS] floatValue];
[_seekProgressBar setValue:progress];

// 视频总长, 单位是秒, 小数部分为毫秒
float duration = [param[EVT_PLAY_DURATION] floatValue];
// 可以用于设置时长显示等等
}
}

```

事件监听

除了 PROGRESS 进度信息，SDK 还会通过 onPlayEvent（事件通知）和 onNetStatus（状态反馈）同步给您的应用程序很多其它的信息：

1. 播放事件

事件ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始，如果有转菊花什么的这个时候该停了
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度 和总体时长
PLAY_EVT_PLAY_LOADING	2007	视频播放loading，如果能够恢复，之后会有BEGIN事件

2. 结束事件

事件ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放
PLAY_ERR_HLS_KEY	-2305	HLS解密key获取失败

3. 警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连,已启动自动重连(重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_RECV_DATA_LAG	2104	网络来包不稳:可能是下行带宽不足,或由于主播端出流不均匀
PLAY_WARNING_VIDEO_PLAY_LAG	2105	当前视频播放出现卡顿
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败,采用软解
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	当前视频帧不连续,可能丢帧
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS解析失败(仅播放RTMP地址时会抛送)

事件ID	数值	含义说明
PLAY_WARNING_SEVER_CONN_FAIL	3002	RTMP服务器连接失败（仅播放RTMP地址时会抛送）
PLAY_WARNING_SHAKE_FAIL	3003	RTMP服务器握手失败（仅播放RTMP地址时会抛送）

4. 连接事件

此外还有几个连接服务器的事件，主要用于测定和统计服务器连接时间，您也无需关心：

事件ID	数值	含义说明
PLAY_EVT_CONNECT_SUCC	2001	已经连接服务器
PLAY_EVT_RTMP_STREAM_BEGIN	2002	已经连接服务器，开始拉流（仅播放RTMP地址时会抛送）
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包(IDR)

5. 分辨率事件

以下事件用于获取画面变化信息，您也无需关心：

事件ID	数值	含义说明
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率改变
PLAY_EVT_CHANGE_ROTATION	2011	MP4视频旋转角度

视频宽高

视频的宽高（分辨率）是多少？

站在 SDK 的角度，如果只是拿到一个 URL 字符串，它是回答不出这个问题的。要知道视频画面的宽和高各是多少个 pixel, SDK 需要先访问云端服务器，直到加载到足够能够分析出视频画面大小的信息才行，所以对于视频信息而言，SDK 也只能以通知的方式告知您的应用程序。

onNetStatus 通知每秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前SDK内部的一些具体情况，以便您能对当前网络状况和视频信息等有所了解。

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时CPU使用率

评估参数	含义说明
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 - 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_CACHE_SIZE	缓冲区 (jitterbuffer) 大小，缓冲区当前长度为 0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器IP

您也可直接调用 `-[TXVodPlayer width]` 和 `-[TXVodPlayer height]` 直接获取当前宽高.

视频信息

如果通过fileId方式播放且请求成功，SDK会将一些请求信息通知到上层。您需要在收到 `PLAY_EVT_GET_PLAYINFO_SUCC` 事件后，解析param中的信息。

视频信息	含义说明
EVT_PLAY_COVER_URL	视频封面地址
EVT_PLAY_URL	视频播放地址
EVT_PLAY_DURATION	视频时长

离线下载

点播离线播放是一个非常普遍的需求，用户可以在有网络的地方先下载好视频，等到了无网络的环境可以再次观看。SDK提供了播放本地文件的能力，但仅限于mp4和flv这种单一文件格式，HLS流媒体因为无法保存到本地，所以不能本地播放。现在，您可以通过 `TXVodDownloadManager` 将HLS下载到本地，以实现离线播放HLS的能力。

step1：准备工作

`TXVodDownloadManager` 被设计为单例，因此您不能创建多个下载对象。用法如下

```
TXVodDownloadManager *downloader = [TXVodDownloadManager sharedInstance];  
[downloader setDownloadPath:"<指定您的下载目录>"];
```

step2: 开始下载

开始下载有两种方式：url和fileid。url方式非常简单，只需要传入下载地址即可

```
[downloader startDownloadUrl:@"http://1253131631.vod2.myqcloud.com/26f327f9vodgzp125313163  
1/f4bdf799031868222924043041/playlist.m3u8"]
```

fileid下载至少需要传入appId和fileId

```
TXPlayerAuthParams *auth = [TXPlayerAuthParams new];  
auth.appId = 1252463788;  
auth.fileId = @"4564972819220421305";  
TXVodDownloadDataSource *dataSource = [TXVodDownloadDataSource new];  
dataSource.auth = auth;  
[downloader startDownload:dataSource];
```

fileid的获取方式可参考 <https://cloud.tencent.com/document/product/454/12147#step-3.3A-.E5.90.AF.E5.8A.A8.E6.92.AD.E6.94.BE>

step3 : 任务信息

在接收任务信息前，需要先设置回调delegate

```
downloader.delegate = self;
```

可能收到的任务回调有：

1. -[TXVodDownloadDelegate onDownloadStart:]
任务开始，表示SDK已经开始下载。
2. -[TXVodDownloadDelegate onDownloadProgress:]
任务进度，下载过程中，SDK会频繁回调此接口，你可以在这里更新进度显示
3. -[TXVodDownloadDelegate onDownloadStop:]
任务停止，当您调用是 `stopDownload` 停止下载，收到此消息表示停止成功
4. -[TXVodDownloadDelegate onDownloadFinish:]
下载完成，收到此回调表示已全部下载。此时下载文件可以给TXVodPlayer播放
5. -[TXVodDownloadDelegate onDownloadError:errorMsg:]
下载错误，下载过程中遇到网络断开会回调此接口，同时下载任务停止。所有错误码请参

考 `TXDownloadError`。

由于downloader可以同时下载多个任务，所以回调接口里带上了 `TXVodDownloadMediaInfo` 对象，您可以访问 `url`或`dataSource`判断下载源，同时还可以获取到下载进度、文件大小等信息。

step4 : 中断下载

停止下载请调用 `-[TXVodDownloadManager stopDownload:]` 方法，参数为 `-[TXVodDownloadManager sartDownloadUrl:]` 返回的对象。**SDK支持断点续传**，当下载目录没有发生改变时，下次下载同一个文件时会从上次停止的地方重新开始。

如果您不需要重新下载，请调用 `-[TXVodDownloadManager deleteDownloadFile:]` 方法删除文件，以释放存储空间。

录屏(ReplayKit)

最近更新时间：2018-08-10 16:21:04

概述

录屏功能是 iOS 10 新推出的特性，苹果在 iOS 9 的 ReplayKit 保存录屏视频的基础上，增加了视频流实时直播功能，官方介绍见 [Go Live with ReplayKit](#)。iOS 11 增强为 [ReplayKit2](#)，进一步提升了 Replaykit 的易用性和通用性，并且可以对整个手机实现屏幕录制，而非某些做了支持ReplayKit功能的App，因此录屏推流建议直接使用 iOS11的ReplayKit2屏幕录制方式。系统录屏采用的是扩展方式，扩展程序有单独的进程，iOS 系统为了保证系统流畅，给扩展程序的资源相对较少，扩展程序内存占用过大也会被 Kill 掉。腾讯云LiteAV SDK 在原有直播的高质量、低延迟的基础上，进一步降低系统消耗，保证了扩展程序稳定。

注：本文主要介绍 iOS 11 上ReplayKit2录屏使用 SDK 推流的方法，涉及 SDK 的使用介绍同样适用于其它方式的自定义推流。更详细的使用可参考 Demo里 ReplaykitUpload 文件夹的示例代码。

功能体验

体验iOS录屏可以单击安装 [视频云工具包](#)
或 扫码安装



注：录屏推流功能仅11.0以上系统可体验

使用步骤：

1. 打开控制中心，长按屏幕录制按钮，选择【视频云工具包】。
2. 打开【视频云工具包】->【录屏幕推流】，输入推流地址或单击【New】自动获取推流地址，单击【开始推流】。



推流设置成功后，顶部通知栏会提示推流开始，此时您可以在其它设备上看到该手机的屏幕画面。单击手机状态栏的红条，即可停止推流。

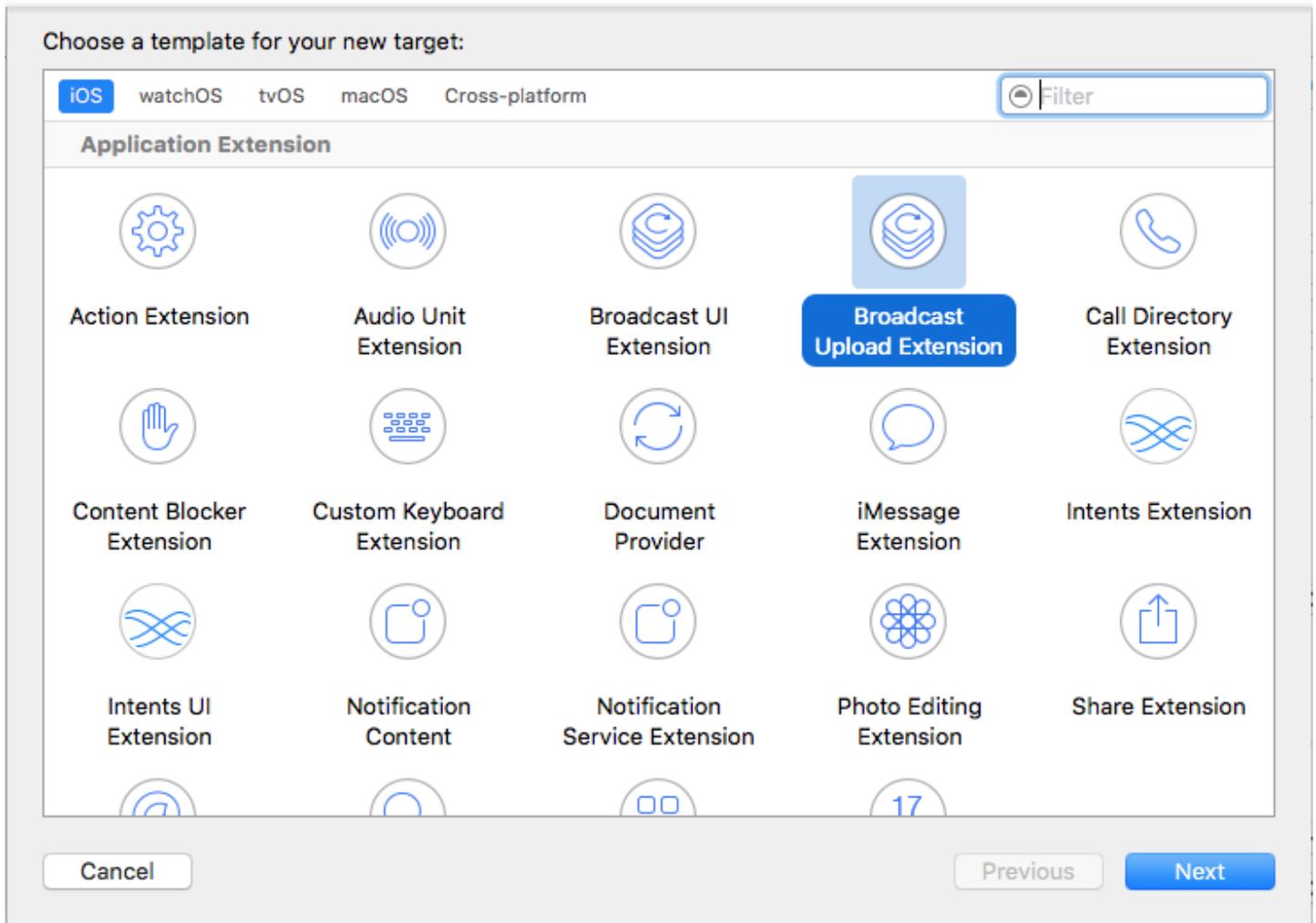
开发环境准备

Xcode 准备

Xcode 9 及以上的版本，手机也必须升级至 iOS 11 以上，模拟器无法使用录屏特性。

创建直播扩展

在现有工程选择【New】->【Target...】，选择【Broadcast Upload Extension】，如图所示。



配置好Product Name。点【Finish】后可以看到，工程多了所输Product Name的目录，目录下有个系统自动生成的SampleHandler类，这个类负责录屏的相关处理。

导入 LiteAV SDK

直播扩展需要导入 TXLiteAVSDK.framework。扩展导入 framework 的方式和主 App 导入方式相同，SDK 的系统依赖库也没有区别。具体可参考腾讯云官网 [工程配置\(iOS\)](#)。

对接流程

Step 1: 创建推流对象

在 SampleHandler.m 中添加下面代码

```
#import "SampleHandler.h"
#import "TXRTMP SDK/TXLiveSDKTypeDef.h"
#import "TXRTMP SDK/TXLivePush.h"
```

```
#import "TXRTMPSDK/TXLiveBase.h"

static TXLivePush *s_txLivePublisher;
static NSString *s_rtmpUrl;

- (void)initPublisher {
    if (s_txLivePublisher) {
        [s_txLivePublisher stopPush];
    }
    TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
    config.customModeType |= CUSTOM_MODE_VIDEO_CAPTURE; //自定义视频模式
    config.enableAutoBitrate = YES;
    config.enableHWAacceleration = YES;

    config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE; //自定义音频模式
    config.audioSampleRate = AUDIO_SAMPLE_RATE_44100; //系统录屏的音频采样率为44100
    config.audioChannels = 1;

    //config.autoSampleBufferSize = YES;
    config.autoSampleBufferSize = NO;
    config.sampleBufferSize = CGSizeMake(544, 960);
    config.homeOrientation = HOME_ORIENTATION_DOWN;

    s_txLivePublisher = [[TXLivePush alloc] initWithConfig:config];
    s_txLivePublisher.delegate = self;
    //[[s_txLivePublisher startPush:s_rtmpUrl];
}
```

s_txLivePublisher 是我们用于推流的对象，因为系统录屏回调的sampleHandler实例有可能不只一个，因此对变量采用静态声明，确保录屏推流过程中使用的是同一个推流器。s_txLivePublisher 的 config 默认的配置为摄像头推流配置，因此需要额外配置为自定义采集视频和音频模式，视频开启autoSampleBufferSize，SDK 会自动根据输入的分辨率设置编码器，您不需要关心推流的分辨率；如果您关闭此选项，那么代表您需要自定义分辨率。因为系统录制对不同机型屏幕所得到的分辨率不一致，因此录屏推流不建议开启autoSampleBufferSize，使用自定义分辨率设置。

实例化 s_txLivePublisher 的最佳位置是在 `-[SampleHandler broadcastStartedWithSetupInfo:]` 方法中，直播扩展启动后会回调这个函数，就可以进行推流器初始化开始推流。但在ReplayKit2的屏幕录制扩展启动时，回调给这里的 setupInfo 为nil，无法获取启动推流所需要的推流地址等信息，因此通常回调此函数时发通知给主App，在主App中设置好推流地址，横竖屏清晰度等信息后再传递给扩展并通知扩展启动推流。扩展与主App间的通信请参见后面所附的 [扩展与宿主App之间的通信与数据传递方式](#) 一节

Step 2: 横屏推流与自定义分辨率

您可以指定任意一个分辨率，SDK 内部将根据您指定的分辨率进行缩放，但设置的分辨率比例应与源画面分辨率比例一致，否则会引起画面变形。homeOrientation属性用来设置横竖屏推流，分辨率需要同时设置为对应的横竖屏比例。以下录屏推流常用的三种清晰度与横屏推流设置示例：

```
static NSString* s_resolution; //SD(标清), HD(高清), FHD(超清)
static BOOL s_landScape; //YES:横屏, NO:竖屏

CGSize screenSize = [[UIScreen mainScreen] currentMode].size;
config.autoSampleBufferSize = NO;
config.homeOrientation = HOME_ORIENTATION_DOWN;

if ([s_resolution isEqualToString:@"SD"]) { //标清
config.sampleBufferSize = CGSizeMake(368, (uint)(360 * screenSize.height / screenSize.width));
config.videoBitrateMin = 400;
config.videoBitratePIN = 800;
config.videoBitrateMax = 1200;
config.videoFPS = 20;
}
else if ([s_resolution isEqualToString:@"FHD"]) { //超清
config.sampleBufferSize = CGSizeMake(720, (uint)(720 * screenSize.height / screenSize.width)); //建议
不超过720P
config.videoBitrateMin = 1200;
config.videoBitratePIN = 1800;
config.videoBitrateMax = 2400;
config.videoFPS = 30;
}
else { //高清
config.sampleBufferSize = CGSizeMake(544, (uint)(540 * screenSize.height / screenSize.width));
config.videoBitrateMin = 1000;
config.videoBitratePIN = 1400;
config.videoBitrateMax = 1800;
config.videoFPS = 24;
}

if (s_landScape) { //横屏推流
config.sampleBufferSize = CGSizeMake(config.sampleBufferSize.height, config.sampleBufferSize.widht
h);
config.homeOrientation = HOME_ORIENTATION_RIGHT;
}
[s_txLivePublisher setConfig:config];
```

注1：一般手机上为9:16，而在iPhoneX上画面比例为1125:2436，因此此处使用屏幕比例进行计算分辨率。

注2：在ReplayKit2上采集的都是竖屏的分辨率，如果需要推送横屏分辨率，除了设置横屏分辨率外还需同时指定homeOrientation为横屏推流，否则会引起画面变形。

Step 3: 发送视频

Replaykit 会将音频和视频都以回调的方式传给 `-[SampleHandler processSampleBuffer:withType]`

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleBufferType {
    switch (sampleBufferType) {
        case RPSampleBufferTypeVideo:
            // Handle audio sample buffer
            {
                if (!CMSampleBufferIsValid(sampleBuffer))
                    return;
                //保存一帧在startPush时发送,防止推流启动后或切换横竖屏因无画面数据而推流不成功
                if (s_lastSampleBuffer) {
                    CFRelease(s_lastSampleBuffer);
                    s_lastSampleBuffer = NULL;
                }
                s_lastSampleBuffer = sampleBuffer;
                CFRetain(s_lastSampleBuffer);

                [s_txLivePublisher sendVideoSampleBuffer:sampleBuffer];
            }
    }
}
```

视频 `sampleBuffer` 只需要调用 `-[TXLivePush sendVideoSampleBuffer:]` 发送即可。

系统分发视频 `sampleBuffer` 的频率并不固定，如果画面静止，可能很长时间才会有一帧数据过来。SDK 考虑到这种情况，内部会做补帧逻辑，使其达到 `config` 所设置的帧率（默认为 20fps）。

注：建议保存一帧给推流启动时使用，防止推流启动或切换横竖屏时因无新的画面数据采集发送，因为画面没有变化时系统可能会很长时间才采集一帧画面。

Step 4: 发送音频

音频也是通过 `-[SampleHandler processSampleBuffer:withType]` 给到直播扩展，区别在于音频有两路数据：一路来自 App 内部，一路来自麦克风。

```

case RPSampleBufferTypeAudioApp:
    // 来自 App 内部的音频
    [s_txLivePublisher sendAudioSampleBuffer:sampleBuffer withType:sampleBufferType];
    break;
case RPSampleBufferTypeAudioMic:
    // 发送来着 Mic 的音频数据
    [s_txLivePublisher sendAudioSampleBuffer:sampleBuffer withType:sampleBufferType];
    break;
    
```

SDK 支持同时发送两路数据，内部会对两路数据进行混音处理。

Step 5: 暂停与恢复

SDK 内部对视频有补帧逻辑，没有视频时会重发最后一帧数据。音频暂停需要调用 `-[TXLivePush setSendAudioSampleBufferMuted:]`，此时 SDK 自动发送静音数据。

```

- (void)broadcastPaused {
    // User has requested to pause the broadcast. Samples will stop being delivered.
    [s_txLivePublisher setSendAudioSampleBufferMuted:YES];
}

- (void)broadcastResumed {
    // User has requested to resume the broadcast. Samples delivery will resume.
    [s_txLivePublisher setSendAudioSampleBufferMuted:NO];
}
    
```

Step 6: SDK 事件处理

事件监听

SDK 事件监听需要设置 `TXLivePush` 的 `delegate` 属性，该 `delegate` 遵循 `TXLivePushListener` 协议。底层的事件会通过

```
-(void) onPushEvent:(int)EvtID withParam:(NSDictionary*)param
```

接口回调过来。录屏推流过程中，一般会收到以下事件：

常规事件

事件 ID	数值	含义说明
PUSH_EVT_CONNECT_SUCC	1001	已经成功连接到腾讯云推流服务器
PUSH_EVT_PUSH_BEGIN	1002	与服务器握手完毕,一切正常,准备开始推流

可在PUSH_EVT_PUSH_BEGIN事件时通知用户推流成功。

错误事件

事件 ID	数值	含义说明
PUSH_ERR_VIDEO_ENCODE_FAIL	-1303	视频编码失败
PUSH_ERR_AUDIO_ENCODE_FAIL	-1304	音频编码失败
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	不支持的视频分辨率
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	不支持的音频采样率
PUSH_ERR_NET_DISCONNECT	-1307	网络断连,且经三次抢救无效,可以放弃治疗,更多重试请自行重启推流

可在PUSH_ERR_NET_DISCONNECT事件时通知用户推流失败。 视频编码失败并不会直接影响推流，SDK 会做处理以保证后面的视频编码成功。

警告事件

事件 ID	数值	含义说明
PUSH_WARNING_NET_BUSY	1101	网络状况不佳：上行带宽太小，上传数据受阻
PUSH_WARNING_RECONNECT	1102	网络断连,已启动自动重连(自动重连连续失败超过三次会放弃)
PUSH_WARNING_HW_ACCELERATION_FAIL	1103	硬编码启动失败，采用软编码
PUSH_WARNING_DNS_FAIL	3001	RTMP -DNS 解析失败（会触发重试流程）
PUSH_WARNING_SEVER_CONN_FAIL	3002	RTMP 服务器连接失败（会触发重试流程）
PUSH_WARNING_SHAKE_FAIL	3003	RTMP 服务器握手失败（会触发重试流程）
PUSH_WARNING_SERVER_DISCONNECT	3004	RTMP 服务器主动断开连接（会触发重试流程）

警告事件表示内部遇到了一些问题，但并不影响推流。建议在PUSH_WARNING_NET_BUSY事件时通知用户网络状态不佳。

全部事件定义请参阅头文件“TXLiveSDKEventDef.h”

直播扩展由于系统限制，不能触发界面动作，但可以通过发本地通知的方式告知用户推流异常。

事件处理示例：

```
-(void) onPushEvent:(int)EvtID withParam:(NSDictionary*)param {
    NSLog(@"onPushEvent %d", EvtID);
    if (EvtID == PUSH_ERR_NET_DISCONNECT) {
        [self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"推流失败!请重新启动推流" userInfo:nil];
    } else if (EvtID == PUSH_EVT_PUSH_BEGIN) {
        [self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"连接成功!开始推流" userInfo:nil];
    } else if (EvtID == PUSH_WARNING_NET_BUSY) {
        [self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"网络上行带宽不足" userInfo:nil];
    }
}
```

Step 7: 结束推流

结束推流 Replay Kit 会调用 `-[SampleHandler broadcastFinished]`，示例代码。

```
-(void)broadcastFinished {
    // User has requested to finish the broadcast.
    if (s_txLivePublisher) {
        [s_txLivePublisher stopPush];
        s_txLivePublisher = nil;
    }
}
```

结束推流后，直播扩展进程可能会被系统回收，所以需要在此处做好清理工作。

附: 扩展与宿主App之间的通信与数据传递方式参考

ReplayKit2录屏只唤起upload直播扩展，直播扩展不能进行UI操作，也不适于做复杂的业务逻辑，因此通常宿主App负责鉴权及其它业务逻辑，直播扩展只负责进行屏幕的音画采集与推流发送，扩展就经常需要与宿主App之间进行数据传递与通信。

1. 发本地通知

扩展的状态需要反馈给用户，有时宿主App并未启动，此时可通过发送本地通知的方式进行状态反馈给用户与激活宿主App进行逻辑交互，如在直播扩展启动时通知宿主App：

```
-(void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *, NSObject *> *)setupInfo {
    [self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"录屏已开始, 请从这里单击回到Demo->录屏幕推流->设置推流URL与横竖屏和清晰度" userInfo:@{kReplayKit2UploadingKey: kReplayKit2Uploading}];
}
```

```
}

- (void)sendLocalNotificationToHostAppWithTitle:(NSString*)title msg:(NSString*)msg userInfo:(NSDictionary*)userInfo
{
    UNNotificationCenter* center = [UNNotificationCenter currentNotificationCenter];

    UNMutableNotificationContent* content = [[UNMutableNotificationContent alloc] init];
    content.title = [NSString localizedUserNotificationStringForKey:title arguments:nil];
    content.body = [NSString localizedUserNotificationStringForKey:msg arguments:nil];
    content.sound = [UNNotificationSound defaultSound];
    content.userInfo = userInfo;

    // 在设定时间后推送本地推送
    UNTimeIntervalNotificationTrigger* trigger = [UNTimeIntervalNotificationTrigger
        triggerWithTimeInterval:0.1f repeats:NO];

    UNNotificationRequest* request = [UNNotificationRequest requestWithIdentifier:@"ReplayKit2Demo"
        content:content trigger:trigger];

    //添加推送成功后的处理！
    [center addNotificationRequest:request withCompletionHandler:^(NSError * _Nullable error) {

    }];
}
```

通过此通知可以提示用户回到主App设置推流信息、启动推流等。

2.进程间的通知CFNotificationCenter

扩展与宿主App之间还经常需要实时的交互处理，本地通知需要用户点知横幅才能触发代码处理，因此不能通过本地通知的方式。而NSNotificationCenter不能跨进程，因此可以利用CFNotificationCenter在宿主App与扩展之前通知发送，但此通知不能通过其中的userInfo字段进行数据传递，需要通过配置App Group方式使用NSUserDefaults进行数据传递(也可以使用剪贴板，但剪贴板有时不能实时在进程间获取数据，需要加些延迟规避)，如主App在获取好推流URL等后，通知扩展可以进行推流时，可通过CFNotificationCenter进行通知发送直播扩展开始推流：

```
CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter(),kDarwinNotificationNamePushStart,NULL,nil,YES);
```

扩展中可通过监听此开始推流通知，由于此通知是在CF层，需要通过NSNotificationCenter发送到Cocoa类层方便处理：

```
CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
    (__bridge const void*)(self),
```

```
onDarwinReplayKit2PushStart,
kDarwinNotificationNamePushStart,
NULL,
CFNotificationSuspensionBehaviorDeliverImmediately);

[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handleReplayKit2PushStartNotification:) name:@"Cocoa_ReplayKit2_Push_Start" object:nil];

static void onDarwinReplayKit2PushStart(CFNotificationCenterRef center,
void *observer, CFStringRef name,
const void *object, CFDictionaryRef
userInfo)
{
//转到cocoa层框架处理
[[NSNotificationCenter defaultCenter] postNotificationName:@"Cocoa_ReplayKit2_Push_Start" object:
nil];
}

- (void)handleReplayKit2PushStartNotification:(NSNotification*)noti
{
//通过NSUserDefaults或剪贴板拿到宿主主要传递的数据
// NSUserDefaults *defaults = [[NSUserDefaults alloc] initWithSuiteName:kReplayKit2AppGroupId];

UIPasteboard* pb = [UIPasteboard generalPasteboard];
NSDictionary* defaults = [self jsonData2Dictionary:pb.string];

s_rtmpUrl = [defaults objectForKey:kReplayKit2PushUrlKey];
s_resolution = [defaults objectForKey:kReplayKit2ResolutionKey];
if (s_resolution.length < 1) {
s_resolution = kResolutionHD;
}
NSString* rotate = [defaults objectForKey:kReplayKit2RotateKey];
if ([rotate isEqualToString:kReplayKit2Portrait]) {
s_landScape = NO;
}
else {
s_landScape = YES;
}
[self start];
}
```

常见问题

ReplayKit2屏幕录制在iOS11新推出功能，比较少官方文档并且存在着一些问题每个版本的系统都在不断修复完善中。以下是一些使用中的常见现象或问题：

1. 系统有声音在播放但观众端无法听到声音

系统在做屏幕音频采集时，在从home界面切到有声音播放的App时才会采集声音，从有声音播放的App切换到无声音播放的App时，即使原App还在播放声音系统也不会进行音频采集，此时需要从home界面重新进入到有声音播放的App时系统才会重新采集。

2. 收到推送信息观众端有时听不到声音

这个是ReplayKit2在早期系统中存在的问题，收到推送消息后会停止屏幕录制的声音采集或采集到的是静音数据，需要重新从home界面切回到有时间的App才能恢复音频采集。在11.3之后的版本系统修复了这个问题。

3. 打开麦克风录制时系统播放声音会变小

这个是属于系统机制：打开麦克风采集时系统音频处于录制模式，会自动将其它的App播放的声音变为听筒模式，中途关闭麦克风采集也不会恢复，只有关闭或重新启动无麦克风录制时才会恢复为扬声器的播放。这个机制不影响App那路声音的录制，即观众端声音听到的声音大小不受影响。

4. 屏幕录制何时会自动会停止

系统在锁屏或有电话打入时，会自动停止屏幕录制，此时SampleHandler里的broadcastFinished函数会被调用，可在此函数发通知提示用户。

5. 采集推流过程中有时屏幕录制会自动停止问题

通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此ReplayKit2上建议推流分辨率不高于720P。另外不建议使用autoSampleBufferSize时做横竖屏切换，因为Plus的手机的分辨率可达1080*1920,容易触发系统内存限制而被强制停止

6. iphoneX手机的兼容性与画面变形问题

iphoneX手机因为有刘海，屏幕采集的画面分辨率不是9:16，如果设了推流输出分辨率为9:16的比例如高清里是为960*540的分辨率，这时因为源分辨率不是9:16的，推出去的画面就会稍有变形。建议设置分辨率时根据屏幕分辨率比例来设置，拉流端用AspectFit显示模式iPhoneX的屏幕采集推流会有黑边是正常现象，AspectFill看画面会不全。

特效(AI Effects)

最近更新时间：2018-08-10 16:21:08

特效功能（大眼、瘦脸、动效、绿幕）

功能说明

大眼、瘦脸、动效贴纸、绿幕等特效功能，是基于优图实验室的人脸识别技术和天天P图的美妆技术为基础开发的特权功能，腾讯云小直播团队通过跟优图和P图团队合作，将这些特效深度整合到 RTMP SDK 的图像处理流程中，以实现更好的视频特效。

接入流程

登录 [腾讯云点播控制台](#) 在线申请为期两周的商业版试用LICENSE，可免费申请两次。Licence有两种：

- 试用Licence：**有效期为一个月**，用于调试和测试动效SDK，如果您用试用Licence发布了您的应用，会导致有效期过后动效的功能不可用。
- 正式Licence：有效期根据最终的合同而定，一般为一年。

版本下载

可以到 [RTMP SDK 开发包](#) 页面下方下载特权版 SDK 压缩包，压缩包有加密（解压密码 & licence 的获取参考接入流程），成功解压后得到一个 Demo 和 SDK 文件，特效资源存放在SDK/Resource下。

区分特权版与非特权版，可以查看SDK的bundler id。bundler id为 com.tencent.TXRTMPSDK 表示非特权版，com.tencent.TXRTMPSDK.pitu 表示特权版。

也可以通过体积直观判断，特权版SDK的体积也比非特权版大很多。

Xcode工程设置

参考 [工程配置](#)

1. 添加Framework

特权版需要额外链接一些系统framework

1. AssetsLibrary.framework
2. CoreMedia.framework
3. Accelerate.framework
4. Metal.framework

2. 添加链接参数

在工程 Build Setting -> Other Link Flags 里，增加 `-ObjC` 选项。

3. 添加动效资源

将SDK/Resource下列文件添加到工程中

1. 3DFace
2. detector.bundle
3. FilterEngine.bundle
4. model
5. PE.dat
6. poseest.bundle
7. RPNSegmenter.bundle
8. ufa.bundle

将Demo/TXLiteAVDemo/Resource/Beauty/pitu/data/ 下的SegmentationShader.metal文件添加到工程中

1. SegmentationShader.metal

4. 添加动效资源示例

将zip包中Resource里面的资源以groups refrence形式添加到工程中，这里需要注意的是 handdetect,handtrack,res18_3M三个文件要以folder refrence形式添加，SegmentationShader.metal 文件在 Demo/TXLiteAVDemo/Resource/Beauty/pitu/data/ 下，您可以找到直接添加，具体操作如图所示：

Choose options for adding these files:

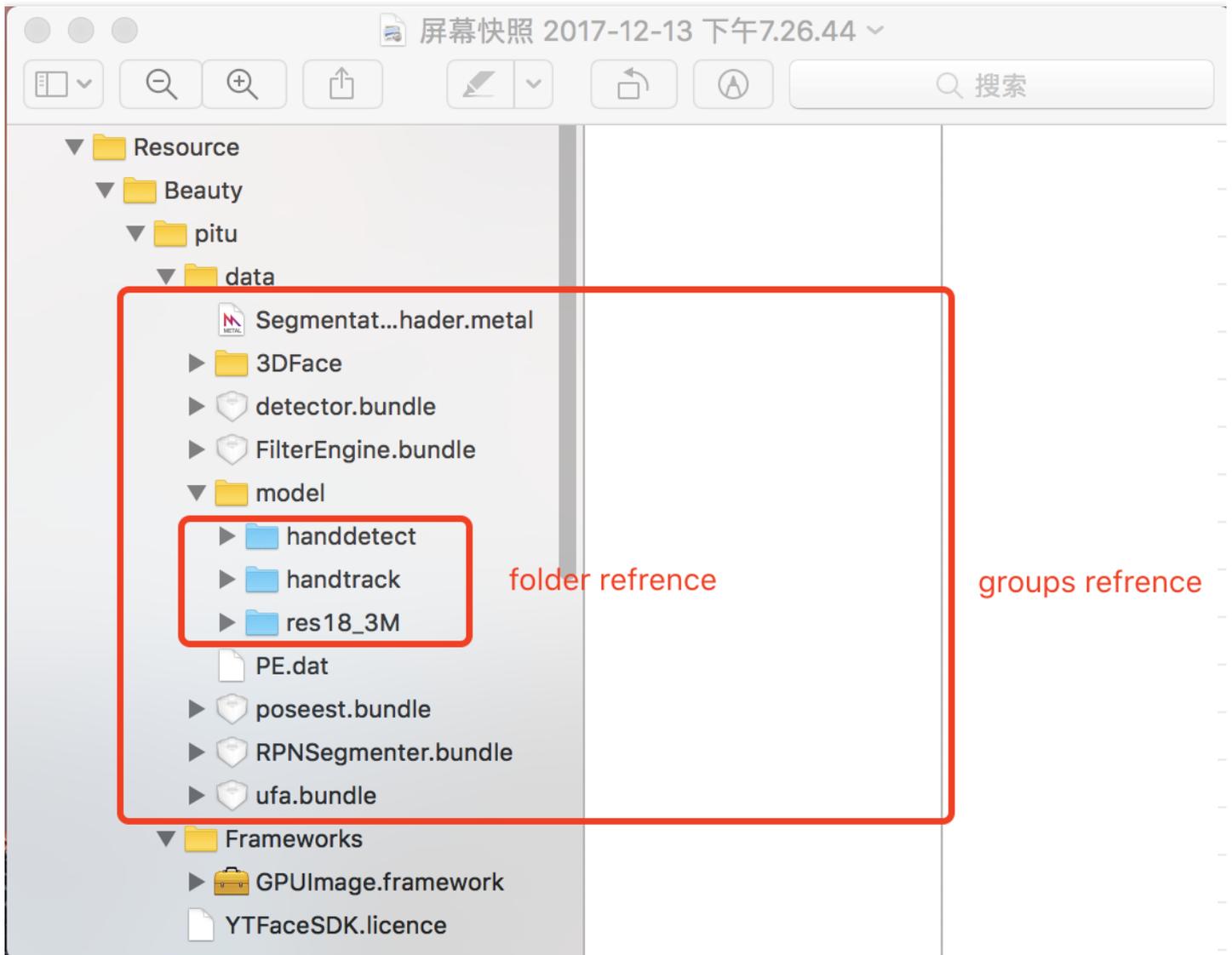
Destination: Copy items if needed

Added folders: Create groups
 Create folder references

Add to targets:  TXLiteAVDemo_Enterprise

Cancel

Finish



这些资源非常重要，否则切换到换脸类素材时会发生crash。

5. 导入licence文件

特权版需要 licence 验证通过后，相应功能才能生效。您可以向我们的商务同学申请一个免费 30 天的调试用 licence。

得到 licence 后，您需要将其命名为**YTFaceSDK.licence**，然后如上图所示添加到工程。

每个licence都有绑定具体的Bundle Identifier，修改app的Bundle Identifier会导致验证失败。

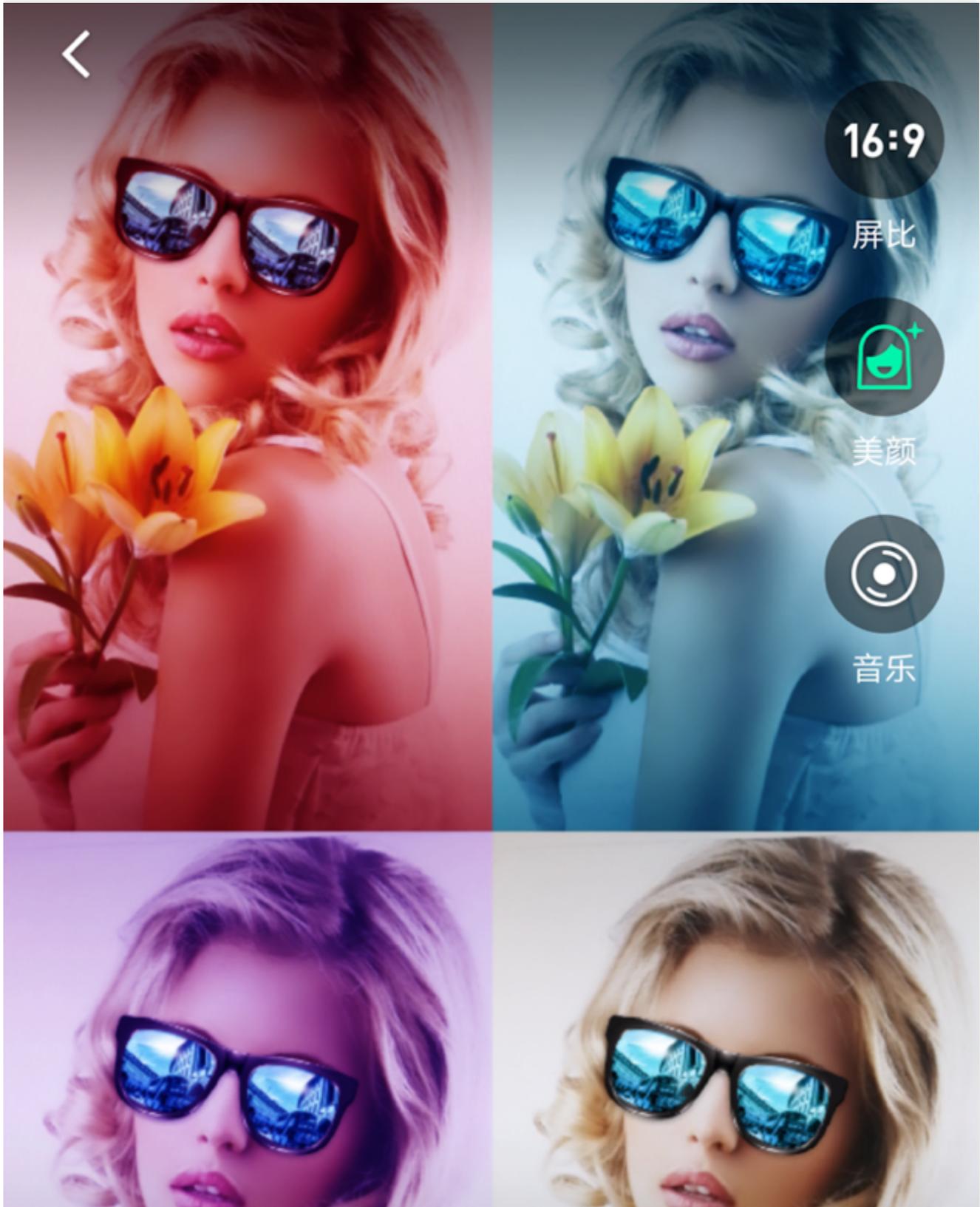
YTFaceSDK.licence的文件名固定，不可修改。

iOS 和 Android 不需要重复申请 licence，一个 licence 可以同时授权一个 iOS 的 bundleid 和一个 Android 的 packageName。

功能调用

1. 动效贴纸

示例：





一个动效模版是一个目录，里面包含很多资源文件。每个动效因为复杂度不同，目录个数以和文件大小也不尽相同。

小直播中的示例代码是从后台下载动效资源，再统一解压到Resource目录。您可以在小直播代码中找到动效资源和动效缩略图的下载地址，格式如下

```
https://st1.xiangji.qq.com/yunmaterials/{动效名}.zip
```

```
https://st1.xiangji.qq.com/yunmaterials/{动效名}.png
```

强烈建议客户将动效资源放在自己的服务器上，以防小直播变动造成不必要的影响。

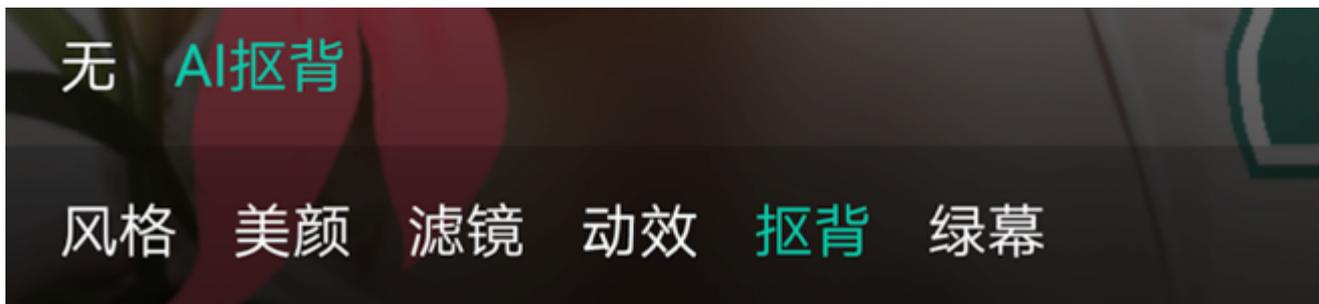
当解压完成后，即可通过以下接口开启动效效果

```
/**
 * 选择动效
 *
 * @param tmplName: 动效名称
 * @param tmplDir: 动效所在目录
 */
- (void)selectMotionTpl:(NSString *)tmplName inDir:(NSString *)tmplDir;
```

2. AI抠背

示例：





需要下载AI抠背的资源，接口跟动效接口相同

```
/**
 * 选择扣背动效
 *
 * @param tmplName: 动效名称
 * @param tmplDir: 动效所在目录
 */
- (void)selectMotionTpl:(NSString *)tmplName inDir:(NSString *)tmplDir;
```

3. 美妆美容

```
/* setEyeScaleLevel 设置大眼级别 (增值版本有效, 普通版本设置此参数无效)
 * 参数:
 * eyeScaleLevel: 大眼级别取值范围 0 ~ 9; 0 表示关闭 1 ~ 9值越大 效果越明显。
 */
- (void) setEyeScaleLevel:(float)eyeScaleLevel;

/* setFaceScaleLevel 设置瘦脸级别 (增值版本有效, 普通版本设置此参数无效)
 * 参数:
 * faceScaleLevel: 瘦脸级别取值范围 0 ~ 9; 0 表示关闭 1 ~ 9值越大 效果越明显。
 */
- (void) setFaceScaleLevel:(float)faceScaleLevel;

/* setFaceVLevel 设置V脸 (增值版本有效, 普通版本设置此参数无效)
 * 参数:
 * faceVLevel: V脸级别取值范围 0 ~ 9; 0 表示关闭 1 ~ 9值越大 效果越明显。
 */
- (void) setFaceVLevel:(float)faceVLevel;

/* setChinLevel 设置下巴拉伸或收缩 (增值版本有效, 普通版本设置此参数无效)
 * 参数:
 * chinLevel: 下巴拉伸或收缩取值范围 -9 ~ 9; 0 表示关闭 -9收缩 ~ 9拉伸。
 */
- (void) setChinLevel:(float)chinLevel;
```

```
/* setFaceShortLevel 设置短脸 ( 增值版本有效 , 普通版本设置此参数无效 )
* 参数 :
* faceShortlevel : 短脸级别取值范围 0 ~ 9 ; 0 表示关闭 1 ~ 9值越大 效果越明显。
*/
- (void) setFaceShortLevel:(float)faceShortlevel;

/* setNoseSlimLevel 设置瘦鼻 ( 增值版本有效 , 普通版本设置此参数无效 )
* 参数 :
* noseSlimLevel : 瘦鼻级别取值范围 0 ~ 9 ; 0 表示关闭 1 ~ 9值越大 效果越明显。
*/
- (void) setNoseSlimLevel:(float)noseSlimLevel;
```

4. 绿幕功能

使用绿幕需要先准备一个用于播放的mp4文件，通过调用以下接口即可开启绿幕效果

```
/**
* 设置绿幕文件
*
* @param file: 绿幕文件路径。支持mp4; nil 关闭绿幕
*/
-(void)setGreenScreenFile:(NSURL *)file;
```

问题排查

1. 工程编译不过？

1. 检查AssetsLibrary.framework、CoreMedia.framework、Accelerate.framework、Metal.framework 依赖库是否已经添加

2. 工程运行过程中crash？

1. 检查工程是否配置了 -ObjC
2. 检查 Metal API Validation 是否设置成了Disabled

3. 工程特效不生效？

1. 检查YTFaceSDK.licence 命名是否正确
2. 检查licence是否过期（下载[查询工具](#)或则联系我们的开发同学）
3. 检查pitu资源是否添加正确，尤其要注意 handdetect,handtrack,res18_3M三个文件要以folder refrence形式添加，最简单的方法就是比对自己工程添加的动效文件是否和我们demo添加的完全一致
4. 如果客户更新了licence，请确保使用的是最新的licence，如果不确定，可以查下licence的有效期（下载[查询工具](#)或则联系我们开发同学），另外如果工程更换了licence，请先clean工程，删除本地安装包，重新编译

[查询工具](#)是一个xcode工程，目前仅支持在mac上使用，后续会开放其他查询方式

实时连麦

直播连麦 (LiveRoom)

最近更新时间：2018-05-17 21:11:33

功能介绍

直播+连麦 是在 **秀场直播** 和 **在线教育** 场景中经常使用的直播模式，它既能支持高并发和低成本的在线直播，又能通过连麦实现主播和观众之间的视频通话互动，具有极强的场景适用性。



腾讯云基于 **LiveRoom** 组件实现“直播 + 连麦”功能，它分成 Client 和 Server 两个部分（都是开源的），对接攻略请参考 [DOC](#)，本文档主要是详细列出了 Client 端的 API 列表：

LiveRoom

名称	描述
LiveRoomListener	liveroom 回调

名称	描述
- (void)login:(NSString)serverDomain loginInfo:(LoginInfo)loginInfo withCompletion:(ILoginCompletionHandler)completion;	登录到RoomService后台
-(void)logout:(ILogoutCompletionHandler)completion	从RoomService后台登出
- (void)getRoomList:(int)index cnt:(int)cnt withCompletion:(IGetRoomListCompletionHandler)completion	获取房间列表（非必须，可选择使用您自己的房间列表）
- (void)getAudienceList:(NSString *)roomId withCompletion:(IGetAudienceListCompletionHandler)completion	获取某个房间里的观众列表（最多返回最近加入的 30 个观众）
- (void)createRoom:(NSString)roomId roomInfo: (NSString)roomInfo withCompletion:(ICreateRoomCompletionHandler)completion	主播：创建房间（roomId 可不填）
- (void)enterRoom:(NSString)roomId withView:(UIView)view withCompletion:(IEnterRoomCompletionHandler)completion	观众：进入房间
- (void)exitRoom:(IExitRoomCompletionHandler)completion	主播 OR 观众：退出房间
- (void)startLocalPreview:(UIView *)view	主播 OR 连麦观众：开启摄像头预览
- (void)stopLocalPreview	停止摄像头预览
- (void)requestJoinPusher:(NSInteger)timeout withCompletion:(IRequestJoinPusherCompletionHandler)completion	观众：发起连麦请求
- (void)joinPusher:(IJoinPusherCompletionHandler)completion	观众：进入连麦状态
- (void)quitPusher:(IQuitPusherCompletionHandler)completion	观众：退出连麦状态
- (void)acceptJoinPusher:(NSString *)userID	主播：接受来自观众的连麦请求
- (void)rejectJoinPusher:(NSString)userID reason:(NSString)reason	主播：拒绝来自观众的连麦请求
- (void)kickoutSubPusher:(NSString *)userID	主播：踢掉连麦中的某个观众
- (void)getOnlinePusherList:(IGetOnlinePusherListCompletionHandler)completion	主播PK：获取在线的主播列表
- (void)startPlayPKStream:(NSString)playUrl view:(UIView)view playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError	主播PK：开始播放对方的视频流

名称	描述
- (void)stopPlayPKStream	主播PK：结束播放对方的视频流
- (void)sendPKReques:(NSString *)userID timeout:(NSInteger)timeout withCompletion:(IRequestPKCompletionHandler)completion	主播PK：发送PK请求
- (void)sendPKFinishRequest:(NSString *)userID	主播PK：发送结束PK的请求
- (void)acceptPKRequest:(NSString *)userID	主播PK：授受PK请求
- (void)rejectPKRequest:(NSString *)userID reason:(NSString *)reason	主播PK：拒绝PK请求
- (void)addRemoteView:(UIView *)view withUserID:(NSString *)userID playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError	主播：播放连麦观众的远程视频画面
- (void)deleteRemoteView:(NSString *)userID	主播：移除连麦观众的远程视频画面
- (void)sendRoomTextMsg:(NSString *)textMsg	发送文本（弹幕）消息
- (void)sendRoomCustomMsg:(NSString *)cmd msg:(NSString *)msg	发送自定义格式的消息（点赞，送花）
- (void)switchToBackground:(UIImage *)pauseImage	App 从前台切换到后台
- (void)switchToForeground	App 从后台切换到前台
- (void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel	设置美颜
- (void)switchCamera	切换前后置摄像头，支持在推流中动态切换
- (void)setMute:(BOOL)isMute	静音
- (void)setMirror:(BOOL)isMirror	画面镜像（此接口仅影响观众端效果，主播一直保持镜像效果）
- (BOOL)playBGM:(NSString *)path	开始播放背景音乐（path 一定要是app对应的document目录下面的路径）
- (BOOL)stopBGM	停止播放背景音乐

名称	描述
- (BOOL)pauseBGM	暂停播放背景音乐
- (BOOL)resumeBGM	继续播放背景音乐
- (BOOL)setMicVolume:(float)volume	设置混音时麦克风的音量大小
- (BOOL)setBGMVolume:(float)volume	设置混音时背景音乐的音量大小
- (void)startRecord	开始视频录制
- (void)stopRecord	停止视频录制
TXLiveRecordListener	视频录制回调

LiveRoomListener

名称	描述
- (void)onGetPusherList:(NSArray *)pusherInfoArray	通知：房间里已有的推流者列表（也就是当前有几路远程画面）
- (void)onPusherJoin:(PusherInfo *)pusherInfo	通知：新的推流者加入进来（也就是通知您多了一路远程画面）
- (void)onPusherQuit:(PusherInfo *)pusherInfo	通知：有推流者离开房间（也就是通知您少了一路远程画面）
- (void)onRecvJoinPusherRequest:(NSString)userID nickName:(NSString)nickName headPic:(NSString *)headPic	通知：主播收到观众的连麦请求
- (void)onKickout	观众：收到被大主播踢开的消息
- (void)onRecvPKRequest:(NSString)userID userName:(NSString)userName userAvatar:(NSString)userAvatar streamUrl:(NSString)streamUrl	收到PK请求
- (void)onRecvPKFinishRequest:(NSString *)userID	收到结束PK的请求

名称	描述
- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID nickName:(NSString *)nickName headPic:(NSString *)headPic textMsg:(NSString *)textMsg	聊天室：收到文本消息
- (void)onRecvRoomCustomMsg:(NSString *)roomID userID:(NSString *)userID nickName:(NSString *)nickName headPic:(NSString *)headPic cmd:(NSString *)cmd msg:(NSString *)msg	聊天室：收到自定义消息
- (void)onRoomClose:(NSString *)roomID	通知：房间解散通知
- (void)onDebugMsg:(NSString *)msg	LOG：日志回调
- (void)onError:(int)errCode errMsg:(NSString *)errMsg	ERROR：错误回调

LiveRoom 接口详情

1.setLiveRoomListener

- 接口定义：@property (nonatomic, weak) id < LiveRoomListener > delegate;
- 接口说明：设置 LiveRoomListener 代理回调，具体回调函数请参考 LiveRoomListener 的接口说明
- 参数说明：

参数	类型	说明
delegate	LiveRoomListener	liveroom 回调接口

- 示例代码：

```
self.liveRoom.delegate = self(回调监听者)
```

2.login

- 接口定义：- (void)login:(NSString *)serverDomain loginInfo:(LoginInfo *)loginInfo withCompletion:(ILoginCompletionHandler)completion
- 接口说明：登录到 RoomService 后台，通过参数 serverDomain 可以指定是使用腾讯云的 RoomService 还是使用自建的 RoomService。
- 参数说明：

参数	类型	说明
serverDomain	String	RoomService 的服务器地址，这部分可以参考 DOC 。
loginInfo	LoginInfo	登录参数，这部分可以参考 DOC 。
completion	ILoginCompletionHandler	登录成功与否的回调

- 示例代码：

```

LoginInfo* initInfo = [LoginInfo new];
initInfo.sdkAppID = sdkAppID;
initInfo.userID = username;
initInfo.headPicUrl = @"";
initInfo.userSig = sign;
initInfo.accType = accountType;
[self.liveRoom login:kHttpServerAddrDomain loginInfo:initInfo withCompletion:^(int errorCode, NSString *errMsg) {
    NSLog(@"errorCode:%d, errMsg:%@", errorCode, errMsg);
    if (errorCode == ROOM_SUCCESS) {
        succ(username, hashPwd);
    }
    else {
        fail(errorCode, errMsg);
    }
}];

```

3.logout

- 接口定义：-(void)logout:(ILogoutCompletionHandler)completion
- 接口说明：从 RoomService 后台注销
- 示例代码：

```

[self.liveRoom logout:^(int errorCode, NSString *errMsg) {
    // to do
}];

```

4.getRoomList

- 接口定义：-(void)getRoomList:(int)index cnt:(int)cnt withCompletion:(IGetRoomListCompletionHandler)completion

- 接口说明：拉取房间列表，index 和 cnt 两个参数用于做分页处理，表示：从序号 index 的房间开始拉取 cnt 个房间。这并非一个必须调用的 API，如果您已经有自己的房间列表服务模块，可以继续使用。
- 参数说明：

参数	类型	说明
index	int	从第几个房间开始拉取
cnt	int	希望 RoomService 返回的房间个数
completion	IGetRoomListCompletionHandler	拉取房间列表的回调

- 示例代码：

```

//拉取序号0开始，拉取20个房间
[self.liveRoom getRoomList:0 cnt:20 withCompletion:^(int errCode, NSString *errMsg,
NSArray<RoomInfo *> *roomInfoArray) {
if (errCode == 0) {
//拉取成功
} else {
//拉取失败
}
}];
    
```

5.getAudienceList

- 接口定义：- (void)getAudienceList:(NSString *)roomID withCompletion:(IGetAudienceListCompletionHandler)completion
- 接口说明：获取某个房间里的观众列表，只返回最近进入房间的 30 位观众。
- 参数说明：

参数	类型	说明
roomID	NSString	房间roomID
completion	IGetAudienceListCompletionHandler	拉取房间观众列表的回调

- 示例代码：

```
//获取房间的观众列表
[_liveRoom getAudienceList:roomId withCompletion:^(int errorCode, NSString *errMsg,
NSArray<AudienceInfo *> *audienceInfoArray) {
//to do
}];
```

其中 audienceInfoArray 是一个观众信息的数组，其结构定义为：

```
// 普通观众信息
@interface AudienceInfo : NSObject
@property (nonatomic, copy) NSString* userID;
@property (nonatomic, copy) NSString* userInfo;
@end
```

6.createRoom

- 接口定义：- (void)createRoom:(NSString *)roomId roomInfo:(NSString *)roomInfo withCompletion:(ICreateRoomCompletionHandler)completion
- 接口说明：在 RoomService 后台创建一个直播房间。
- 参数说明：

参数	类型	说明
roomId	String	您可以通过 roomId 参数指定新房间的 ID，也可以不指定。如果您不指定房间 ID，RoomService 会自动生成一个新的 roomId 并通过 ICreateRoomCompletionHandler 返回给您。
roomInfo	String	由创建者自定义。在 getRoomList 中返回该信息
completion	ICreateRoomCompletionHandler	创建房间结果回调

- 示例代码：

```
[self.liveRoom createRoom:@" " roomInfo:@"创建房间者相关信息" withCompletion:^(int errorCode, NSString *errMsg) {
NSLog(@"createRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);
}];
```

7.enterRoom

- 接口定义：- (void)enterRoom:(NSString *)roomId withView:(UIView *)view withCompletion:(IEnterRoomCompletionHandler)completion
- 接口说明：（观众）进入直播间。
- 示例代码：

```
[_liveRoom enterRoom:groupid withView:videoParentView withCompletion:^(int errorCode, NSString *errMsg) {  
    NSLog(@"enterRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);  
}];
```

8.exitRoom

- 接口定义：- (void)exitRoom:(IExitRoomCompletionHandler)completion
- 接口说明：（主播 OR 观众）退出房间。
- 示例代码：

```
[self.liveRoom exitRoom:^(int errorCode, NSString *errMsg) {  
    NSLog(@"exitRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);  
}];
```

9.startLocalPreview

- 接口定义：- (void)startLocalPreview:(UIView *)view
- 接口说明：（主播 OR 连麦观众）启动摄像头预览，默认是使用前置摄像头，可以通过switchCamera切换前后摄像头
- 示例代码：

```
[self.liveRoom startLocalPreview:videoParentView];
```

10.stopLocalPreview

- 接口定义：- (void)stopLocalPreview
- 接口说明：（主播 OR 连麦观众）关闭摄像头预览。
- 示例代码：

```
[self.liveRoom stopLocalPreview];
```

11.requestJoinPusher

- 接口定义：- (void)requestJoinPusher:(NSInteger)timeout withCompletion:(IRequestJoinPusherCompletionHandler)completion
- 接口说明：（观众）请求和主播连麦时调用该接口。
- 示例代码：

```
[self.liveRoom requestJoinPusher:20 withCompletion:^(int errorCode, NSString *errMsg) {  
    if (errorCode == 0) {  
        [TCUtil toastTip:@"主播接受了您的连麦请求，开始连麦" parentView:self.view];  
    }  
    else {  
        [TCUtil toastTip:errMsg parentView:self.view];  
    }  
}];
```

12.joinPusher

- 接口定义：- (void)joinPusher:(IJoinPusherCompletionHandler)completion
- 接口说明：（观众）开始推流，并进入连麦状态，是否最终连麦成功，要看 IJoinPusherCompletionHandler 的回调结果。
- 示例代码：

```
[self.liveRoom joinPusher:^(int errorCode, NSString *errMsg) {  
    //连麦结果回调  
}];
```

13.quitPusher

- 接口定义：- (void)quitPusher:(IQuitPusherCompletionHandler)completion
- 接口说明：（观众）主动退出连麦状态。
- 示例代码：

```
[self.liveRoom quitPusher:^(int errorCode, NSString *errMsg) {  
    //退出连麦状态结果回调  
}];
```

14.acceptJoinPusher

- 接口定义：- (void)acceptJoinPusher:(NSString *)userID
- 接口说明：（主播）同意观众的连麦请求。
- 示例代码：

```
[self.liveRoom acceptJoinPusher:userID];
```

15.rejectJoinPusher

- 接口定义：- (void)rejectJoinPusher:(NSString *)userID reason:(NSString *)reason
- 接口说明：（主播）拒绝观众的连麦请求。
- 示例代码：

```
[self.liveRoom rejectJoinPusher:userID reason:@"主播端连麦人数超过最大限制"];
```

16.kickoutSubPusher

- 接口定义：- (void)kickoutSubPusher:(NSString *)userID
- 接口说明：主播：踢掉连麦中的某个观众。
- 示例代码：

```
[self.liveRoom kickoutSubPusher:userID];
```

17.getOnlinePusherList

- 接口定义：- (void)getOnlinePusherList:(IGetOnlinePusherListCompletionHandler)completion
- 接口说明：主播PK：获取在线的主播列表
- 示例代码：

```
[_liveRoom getOnlinePusherList:^(NSArray<PusherInfo *> *pusherInfoArray) {  
  
}];
```

18.startPlayPKStream

- 接口定义：- (void)startPlayPKStream:(NSString *)playUrl view:(UIView *)view playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError
- 接口说明：主播PK：开始播放对方的流

- 示例代码：

```
[_liveRoom startPlayPKStream:playUrl view:playerView playBegin:^(  
    } playError:^(int errorCode, NSString *errMsg) {  
  
    }];
```

19.stopPlayPKStream

- 接口定义：- (void)stopPlayPKStream
- 接口说明：主播PK：结束播放对方的流
- 示例代码：

```
[_liveRoom stopPlayPKStream];
```

20.sendPKRequest

- 接口定义：- (void)sendPKReques:(NSString *)userID timeout:(NSInteger)timeout withCompletion:(IRequestPKCompletionHandler)completion
- 接口说明：主播PK：发送PK请求
- 示例代码：

```
[_liveRoom sendPKReques:userID timeout:10 withCompletion:^(int errorCode, NSString *errMsg, NSString *streamUrl) {  
  
    }];
```

21.sendPKFinishRequest

- 接口定义：- (void)sendPKFinishRequest:(NSString *)userID
- 接口说明：主播PK：发送结束PK的请求
- 示例代码：

```
[_liveRoom sendPKFinishRequest:userID];
```

22.acceptPKRequest

- 接口定义：- (void)acceptPKRequest:(NSString *)userID

- 接口说明：主播PK：授受PK请求
- 示例代码：

```
[_liveRoom acceptPKRequest:userID];
```

23.rejectPKRequest

- 接口定义：- (void)rejectPKRequest:(NSString *)userID reason:(NSString *)reason
- 接口说明：主播PK：拒绝PK请求
- 示例代码：

```
[_liveRoom rejectPKRequest];
```

24.addRemoteView

- 接口定义：- (void)addRemoteView:(UIView *)view withUserID:(NSString *)userID playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError
- 接口说明：（主播）播放连麦观众的远程视频画面，一般在收到 onPusherJoin（新进连麦通知）时调用。
- 示例代码：

```
- (void)onPusherJoin:(PusherInfo *)pusherInfo
{
    NSString* userID = pusherInfo.userID;
    NSString* strPlayUrl = pusherInfo.playUrl;
    if (userID == nil || strPlayUrl == nil) {
        return;
    }
    __weak typeof(self) weakSelf = self;
    [self.liveRoom addRemoteView:videoView withUserID:userID playBegin:^(
        //播放成功
    ) playError:^(int errCode, NSString *errMsg) {
        //播放失败
    }];
}
```

25.deleteRemoteView

- 接口定义：- (void)deleteRemoteView:(NSString *)userID
- 接口说明：停止播放某个连麦主播视频，一般在收到 onPusherQuit（连麦者离开）时调用。
- 示例代码：

```
[self.liveRoom deleteRemoteView:userID];
```

26.sendRoomTextMsg

- 接口定义：- (void)sendRoomTextMsg:(NSString *)textMsg
- 接口说明：发送文本消息，直播间里的其他人会收到 onRecvRoomTextMsg 通知。
- 示例代码：

```
[[TCLiveRoomMgr getSharedLiveRoom] sendRoomTextMsg:textMsg];
```

27.sendRoomCustomMsg

- 接口定义：- (void)sendRoomCustomMsg:(NSString)cmd msg:(NSString)msg;
- 接口说明：发送自定义消息。直播间其他人会收到 onRecvRoomCustomMsg 通知。
- 示例代码：

```
[[TCLiveRoomMgr getSharedLiveRoom] sendRoomCustomMsg:[@(TCMsgModelType_DanmaMsg) stringValue] msg:textMsg];
```

28.switchToBackground

- 接口定义：- (void)switchToBackground:(UIImage *)pauseImage
- 接口说明：从前台切换到后台，关闭采集摄像头数据，推送默认图片

29.switchToForeground

- 接口定义：- (void)switchToForeground
- 接口说明：由后台切换到前台，开启摄像头数据采集。

30.setBeautyStyle

- 接口定义：- (void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel
- 接口说明：设置美颜风格、磨皮程度、美白级别和红润级别。
- 参数说明：

参数	类型	说明
beautyStyle	int	磨皮风格：0：光滑 1：自然 2：朦胧

参数	类型	说明
beautyLevel	int	磨皮等级：取值为 0-9.取值为 0 时代表关闭美颜效果.默认值: 0,即关闭美颜效果
whitenessLevel	int	美白等级：取值为 0-9.取值为 0 时代表关闭美白效果.默认值: 0,即关闭美白效果
ruddinessLevel	int	红润等级：取值为 0-9.取值为 0 时代表关闭美白效果.默认值: 0,即关闭美白效果

- 示例代码：

```
[self.liveRoom setBeautyStyle:0 beautyLevel:_beauty_level whitenessLevel:_whitening_level ruddinessLevel:0];
```

31.switchCamera

- 接口定义：- (void)switchCamera
- 接口说明：切换摄像头，前置摄像头时调用后变成后置，后置摄像头时调用后变成前置。该接口在启动预览 startLocalPreview 后调用才能生效，预览前调用无效。SDK启动预览默认使用前置摄像头。

32.setMute

- 接口定义：- (void)setMute:(BOOL)isMute
- 接口说明：设置静音接口。设置为静音后SDK不再推送麦克风采集的声音，而是推送静音。
- 参数说明：

参数	类型	说明
isMute	BOOL	是否静音

33.setMirror

- 接口定义：- (void)setMirror:(BOOL)isMirror
- 接口说明：设置播放端水平镜像。注意这个只影响播放端效果，不影响推流主播端。推流端看到的镜像效果是始终存在的，使用前置摄像头时推流端看到的是镜像画面，使用后置摄像头时推流端看到的是非镜像。
- 参数说明：

参数	类型	说明
isMirror	BOOL	YES 表示播放端看到的是镜像画面，false表示播放端看到的是非镜像画面

- 示例代码：

```
//观众端播放看到的是镜像画面
[self.liveRoom setMirror:YES];
```

34.playBGM

- 接口定义：- (BOOL)playBGM:(NSString *)path
- 接口说明：播放背景音乐。该接口用于混音处理，比如将背景音乐与麦克风采集到的声音混合后播放。返回结果中，true 表示播放成功，false 表示播放失败。
- 参数说明：

参数	类型	说明
path	NSString	背景音在app对应的document目录下面的路径

35.stopBGM

- 接口定义：- (BOOL)stopBGM
- 接口说明：停止播放背景音乐。返回结果中，YES 表示停止播放成功，NO 表示停止播放失败。

36.pauseBGM

- 接口定义：- (BOOL)pauseBGM
- 接口说明：暂停播放背景音乐。返回结果中，YES 表示暂停播放成功，NO 表示暂停播放失败。

37.resumeBGM

- 接口定义：- (BOOL)resumeBGM
- 接口说明：恢复播放背景音乐。返回结果中，YES 表示恢复播放成功，NO 表示恢复播放失败。

38.setMicVolume

- 接口定义：- (BOOL)setMicVolume:(float)volume
- 接口说明：设置混音时麦克风的音量大小。返回结果中，true 表示设置麦克风的音量成功，false 表示设置麦克风的音量失败。
- 参数说明：

参数	类型	说明
volume	float	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。推荐在UI上实现相应的一个滑动条，由主播自己设置

39.setBGMVolume

- 接口定义：- (BOOL)setBGMVolume:(float)volume
- 接口说明：设置混音时背景音乐的音量大小。返回结果中，true 表示设置背景音的音量成功，false 表示设置背景音的音量失败。
- 参数说明：

参数	类型	说明
volume	float	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。推荐在UI上实现相应的一个滑动条，由主播自己设置

40.startRecord

- 接口定义：- (void)startRecord
- 接口说明：开始录制视频。该接口用于播放端将播放的实时视频保存到本地文件。
- 特别注意：该接口需要在 enterRoom 成功后调用
- 返回结果：接口返回 0 启动录制成功；-1 表示正在录制，忽略这次录制启动；-2 videoRecorder初始化失败
- 示例代码：

```
[self.liveRoom startRecord];
```

41.stopRecord

- 接口定义：- (void)stopRecord
- 接口说明：停止录制视频。录制结果会通过录制回调异步通知出来。

42.TXLiveRecordListener

- 接口定义：@property (nonatomic, weak) id < TXLiveRecordListener > recordDelegate
- 接口说明：设置视频录制回调，用于接收视频录制进度及录制结果。
- 参数说明：

参数	类型	说明
recordDelegate	TXLiveRecordListener	视频录制回调

- 示例代码：

```
self.liveRoom.recordDelegate = self;
-(void) onRecordProgress:(NSInteger)milliSecond
{
if (!_videoRecordView.hidden) {
float progress = (milliSecond/1000)/kMaxRecordDuration;
[_videoRecordView setVideoRecordProgress:progress];
}
}

-(void) onRecordComplete:(TXRecordResult*)result
{
if (_isResetVideoRecord) return;

if (result.retCode == RECORD_RESULT_FAILED || result.retCode == RECORD_RESULT_OK_INTERRUPT)
{
[TCUtil toastTip:result.descMsg parentView:self.view];
} else {
TCVideoPublishController *vc = [[TCVideoPublishController alloc] init:[TXLivePlayer new]
recordType:kRecordType_Play RecordResult:result TCLiveInfo:self.liveInfo];
[self.navigationController pushViewController:vc animated:true];
}
}
```

LiveRoomListener 接口详情

1. onGetPusherList

- 接口定义：- (void)onGetPusherList:(NSArray *)pusherInfoArray
- 接口说明：房间里已有的推流者列表（也就是当前有几路远程画面）。当新的连麦者加入房间时，新的连麦者会收到该通知。回调中您可以调用 addRemoteView 播放房间里已有连麦者的视频。
- 示例代码：

```
- (void)onGetPusherList:(NSArray<PusherInfo *> *)pusherInfoArray {
for (PusherInfo *pusherInfo in pusherInfoArray) {
[_liveRoom addRemoteView:playerView withUserID:pusherInfo.userID playBegin:^(
} playError:^(int errCode, NSString *errMsg) {

}];
};
```

```
}  
}
```

2. onPusherJoin

- 接口定义：- (void)onPusherJoin:(PusherInfo *)pusherInfo
- 接口说明：当新的连麦者加入房间时，大主播和其他的连麦者都会收到该通知。回调中您可以调用 `addRemoteView` 播放这个新来的连麦者的视频。
- 示例代码：

```
- (void)onPusherJoin:(PusherInfo *)pusherInfo  
{  
    NSString* userID = pusherInfo.userID;  
    NSString* strPlayUrl = pusherInfo.playUrl;  
    if (userID == nil || strPlayUrl == nil) {  
        return;  
    }  
    __weak typeof(self) weakSelf = self;  
    [weakSelf.liveRoom addRemoteView:item.videoView withUserID:userID playBegin:^(  
        //to do  
    ) playError:^(int errCode, NSString *errMsg) {  
        //to do  
    }];  
}
```

3. onPusherQuit

- 接口定义：- (void)onPusherQuit:(PusherInfo *)pusherInfo
- 接口说明：当连麦者离开房间时，大主播和其他的连麦者都会收到该通知。回调中您可以调用 `deleteRemoteView` 停止播放这个连麦者的视频。
- 示例代码：

```
- (void)onPusherQuit:(PusherInfo *)pusherInfo  
{  
    NSString* userID = pusherInfo.userID;  
    //混流：减少一路  
    [weakSelf.liveRoom deleteRemoteView:userID];  
}
```

4. onRecvJoinPusherRequest

- 接口定义：- (void)onRecvJoinPusherRequest:(NSString *)userID nickName:(NSString *)nickName headPic:(NSString *)headPic
- 接口说明：当观众向主播申请连麦时，主播收到该通知。主播可以在回调中接受（acceptJoinPusher）或者拒绝（rejectJoinPusher）申请。
- 示例代码：

```
- (void)onRecvJoinPusherRequest:(NSString *)userID nickName:(NSString *)nickName headPic:(NSString *)headPic
{
    if ([_setLinkMemeber count] >= MAX_LINKMIC_MEMBER_SUPPORT) {
        [TCUtil toastTip:@"主播端连麦人数超过最大限制" parentView:self.view];
        [self.liveRoom rejectJoinPusher:userID reason:@"主播端连麦人数超过最大限制"];
    }
    else if (_userIdRequest && _userIdRequest.length > 0) {
        [TCUtil toastTip:@"请稍后，主播正在处理其它人的连麦请求" parentView:self.view];
        [self.liveRoom rejectJoinPusher:userID reason:@"请稍后，主播正在处理其它人的连麦请求"];
    }
    else {
        //接受连麦
        _userIdRequest = userID;
        [self.liveRoom acceptJoinPusher:_userIdRequest];
    }
}
```

5. onKickOut

- 接口定义：- (void)onKickout
- 接口说明：当主播把一个连麦者踢出连麦状态后，对应的连麦者会收到该通知。在回调中您可以停止本地预览以及退出直播。
- 示例代码：

```
- (void)onKickout
{
    [TCUtil toastTip:@"不好意思，您被主播踢开" parentView:self.view];
    [self.liveRoom stopLocalPreview];
}
```

6. onRecvPKRequest

- 接口定义：- (void)onRecvPKRequest:(NSString *)userID userName:(NSString *)userName userAvatar:(NSString *)userAvatar streamUrl:(NSString *)streamUrl
- 接口说明：当一个主播调用sendPKRequest向另一个主播发起PK请求的时候；另一个主播会收到该回调通知。在该回调中，您可以展示一个收到PK请求的提示框，询问用户是接受还是拒绝。
- 示例代码：

```
- (void)onRecvPKRequest:(NSString *)userID userName:(NSString *)userName userAvatar:(NSString *)userAvatar streamUrl:(NSString *)streamUrl {
    NSString *msg = [NSString stringWithFormat:@"[%@]请求PK", userName];
    UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"提示" message:msg preferredStyle:UIAlertControllerStyleAlert];
    [alertController addAction:[UIAlertAction actionWithTitle:@"拒绝" style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
        [_liveRoom rejectPKRequest:userID reason:@"主播不同意您的PK"];
    }]];
    [alertController addAction:[UIAlertAction actionWithTitle:@"接受" style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
        [_liveRoom acceptPKRequest:userID];
        [_liveRoom startPlayPKStream:playUrl view:playerView playBegin:^(
        } playError:^(int errorCode, NSString *errMsg) {

        }];
    }]];

    [self.navigationController presentViewController:alertController animated:YES completion:nil];
}
```

7. onRecvPKFinishRequest

- 接口定义：- (void)onRecvPKFinishRequest:(NSString *)userID
- 接口说明：当一个主播调用sendPKFinishRequest通知另一个主播结束PK的时候；另一个主播会收到该回调通知。在该回调中，您需要调用stopPlayPKStream结束PK，并做好相关清理工作。
- 示例代码：

```
- (void)onRecvPKFinishRequest:(NSString *)userID {
    [_liveRoom stopPlayPKStream];
}
```

8. onRecvRoomTextMsg

- 接口定义：- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID nickName:(NSString *)nickName headPic:(NSString *)headPic textMsg:(NSString *)textMsg
- 接口说明：当主播或者观众端调用sendRoomTextMsg时，房间内的主播或者观众都会收到该通知。
- 示例代码：

```
- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID nickName:(NSString *)nickName
headPic:(NSString *)headPic textMsg:(NSString *)textMsg
{
    IMUserAble* info = [IMUserAble new];
    info.imUserId = userID;
    info.imUserName = nickName.length > 0? nickName : userID;
    info.imUserIconUrl = headPic;
    info.cmdType = TCMsgModelType_NormalMsg;
    [_logicView handleIMMessage:info msgText:textMsg];
}
```

9. onRecvRoomCustomMsg

- 接口定义：void onRecvRoomCustomMsg(String roomId, String userID, String userName, String userAvatar, String cmd, String message)
- 接口说明：当主播或者观众端调用sendRoomCustomMsg时，房间内的主播或者观众都会收到该通知。
- 示例代码：

```
- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID nickName:(NSString *)nickName
headPic:(NSString *)headPic textMsg:(NSString *)textMsg
{
    IMUserAble* info = [IMUserAble new];
    info.imUserId = userID;
    info.imUserName = nickName.length > 0? nickName : userID;
    info.imUserIconUrl = headPic;
    info.cmdType = TCMsgModelType_NormalMsg;
    [_logicView handleIMMessage:info msgText:textMsg];
}
```

10. onRoomClosed

- 接口定义：- (void)onRoomClose:(NSString *)roomId
- 接口说明：当房间销毁时，观众会收到该通知。需要在回调中退出房间。
- 示例代码：

```
- (void)onRoomClose:(NSString *)roomId
{
    NSLog(@"onRoomClose, roomId:%@", roomId);
}
```

11. onDebugMsg

- 接口定义：- (void)onDebugMsg:(NSString *)msg
- 接口说明：直播间日志回调。可以在回调中将日志保存到文件中，方便问题分析。
- 示例代码：

```
- (void)onDebugMsg:(NSString *)msg
{
    NSLog(@"onDebugMsg:%@", msg);
}
```

12. onError

- 接口定义：- (void)onError:(int)errCode errMsg:(NSString *)errMsg
- 接口说明：直播间错误回调
- 示例代码：

```
- (void)onError:(int)errCode errMsg:(NSString *)errMsg;
{
    NSLog(@"onError:%d, %@", errCode, errMsg);
}
```

视频通话 (RTCRoom)

最近更新时间：2018-03-27 11:08:56

RTCRoom

名称	描述
@property (nonatomic, weak) id < RTCRoomListener > delegate;	设置rtcroom回调
- (void)login:(NSString)serverDomain loginInfo:(LoginInfo)loginInfo withCompletion:(ILoginCompletionHandler)completion	登录到RoomService后台
-(void)logout:(ILogoutCompletionHandler)completion	从RoomService后台登出
- (void)getRoomList:(int)index cnt:(int)cnt withCompletion:(IGetRoomListCompletionHandler)completion	获取房间列表（非必须，可选择使用您自己的房间列表）
- (void)createRoom:(NSString)roomId roomInfo:(NSString)roomInfo withCompletion:(ICreateRoomCompletionHandler)completion	会议创建者：创建房间（roomId可不填）
- (void)enterRoom:(NSString*)roomId withCompletion:(IEnterRoomCompletionHandler)completion	会议参与者：进入房间
- (void)exitRoom:(IExitRoomCompletionHandler)completion	会议创建者 OR 会议参与者：退出房间
- (void)startLocalPreview:(UIView*)view	会议创建者 OR 会议参与者：开启摄像头预览
- (void)stopLocalPreview	停止摄像头预览
- (void)addRemoteView:(UIView)view withUserID:(NSString)userID playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError	播放会议参与者的远程视频画面
- (void)deleteRemoteView:(NSString*)userID	停止播放会议参与者的远程视频画面
- (void)sendRoomTextMsg:(NSString*)textMsg	发送文本（弹幕）消息
- (void)switchToBackground:(UIImage*)pauseImage	App 从前台切换到后台
- (void)switchToForeground	App 从后台切换到前台

名称	描述
- (void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel whitenessLevel:(float)whitenessLevel ruddinessLevel: (float)ruddinessLevel	设置美颜
- (void)switchCamera	切换前后置摄像头，支持在推流中动态切换
- (void)setMute:(BOOL)isMute	静音

RTCRoomListener

名称	描述
- (void)onGetPusherList:(NSArray *)pusherInfoArray	通知：房间里已有的推流者列表（也就是当前有几路远程画面）
- (void)onPusherJoin:(PusherInfo *)pusherInfo;	通知：新的推流者加入进来（也就是通知您多了一路远程画面）
- (void)onPusherQuit:(PusherInfo *)pusherInfo	通知：有推流者离开房间（也就是通知您少了一路远程画面）
- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID userName:(NSString *)userName userAvatar:(NSString *)userAvatar textMsg:(NSString *)textMsg	聊天室：收到文本消息
- (void)onRoomClose:(NSString *)roomID	通知：房间解散通知
- (void)onDebugMsg:(NSString *)msg	LOG：日志回调
- (void)onError:(int)errCode errMsg:(NSString *)errMsg	ERROR：错误回调

RTCRoom 接口详情

1.setDelegate

- 接口定义：@property (nonatomic, weak) id < RTCRoomListener> delegate
- 接口说明：设置 RTCRoomListener 回调，具体回调函数请参考 RTCRoomListener 的接口说明

- 参数说明：

参数	类型	说明
delegate	RTCRoomListener	rtcroom 回调接口

- 示例代码：

```
[self.rtcRoom setDelegate:self];
```

2.login

- 接口定义：- (void)login:(NSString)serverDomain loginInfo:(LoginInfo)loginInfo withCompletion:(ILoginCompletionHandler)completion
- 接口说明：登录到 RoomService 后台，通过参数 serverDomain 可以指定是使用腾讯云的 RoomService 还是使用自建的 RoomService。
- 参数说明：

参数	类型	说明
serverDomain	NSString	RoomService 的服务器地址，这部分可以参考 DOC 。
loginInfo	LoginInfo	登录参数，这部分可以参考 DOC 。
completion	ILoginCompletionHandler	登录成功与否的回调

- 示例代码：

```

LoginInfo *loginInfo = [LoginInfo new];
loginInfo.sdkAppID = sdkAppID;
loginInfo.userID = userID;
loginInfo.userName = userName;
loginInfo.userAvatar = userAvatar;
loginInfo.userSig = userSig;
loginInfo.accType = accType;

__weak __typeof(self) weakSelf = self;
[weakSelf.rtcRoom login:kHttpServerAddrDomain loginInfo:loginInfo withCompletion:^(int errorCode,
NSString *errMsg) {
    NSLog(@"init RTCRoom errorCode[%d] errMsg[%@]", errorCode, errMsg);
}];
    
```

```

if (errCode == 0) {
    //success
} else {
    [weakSelf alertTips:@"rtcRoom init失败" msg:errMsg];
}
}];

```

3.logout

- 接口定义：-(void)logout:(ILogoutCompletionHandler)completion
- 接口说明：从 RoomService 后台注销
- 示例代码：

```
[self.rtcRoom logout:nil];
```

4.getRoomList

- 接口定义：-(void)getRoomList:(int)index cnt:(int)cnt withCompletion:(IGetRoomListCompletionHandler)completion
- 接口说明：拉取房间列表，index 和 cnt 两个参数用于做分页处理，表示：从序号 index 的房间开始拉取 cnt 个房间。这并非一个必须调用的 API，如果您已经有自己的房间列表服务模块，可以继续使用。
- 参数说明：

参数	类型	说明
index	int	从第几个房间开始拉取
cnt	int	希望 RoomService 返回的房间个数
completion	IGetRoomListCompletionHandler	拉取房间列表的回调

- 示例代码：

```

//拉取序号0开始，拉取20个房间
[self.rtcRoom getRoomList:0 cnt:20 withCompletion:^(int errCode, NSString *errMsg, NSArray<RoomInfo *> *roomInfoArray) {
    NSLog(@"getRoomList errCode[%d] errMsg[%@]", errCode, errMsg);
}];

```

5. createRoom

- 接口定义：- (void)createRoom:(NSString *)roomID roomInfo:(NSString *)roomInfo withCompletion:(ICreateRoomCompletionHandler)completion
- 接口说明：在 RoomService 后台创建一个直播房间。
- 参数说明：

参数	类型	说明
roomID	NSString	您可以通过 roomID 参数指定新房间的 ID，也可以不指定。如果您不指定房间 ID，RoomService 会自动生成一个新的 roomID 并通过 CreateRoomCallback 返回给您。
roomInfo	NSString	由创建者自定义。在 getRoomList 中返回该信息
completion	ICreateRoomCompletionHandler	创建房间结果回调

- 示例代码：

```
[self.rtcRoom createRoom:@" " roomInfo:@"roomName" withCompletion:^(int errorCode, NSString *errMsg) {
    NSLog(@"createRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);
}];
```

6. enterRoom

- 接口定义：- (void)enterRoom:(NSString *)roomID withCompletion:(IEnterRoomCompletionHandler)completion
- 接口说明：（会议参与者）进入直播间。
- 示例代码：

```
[self.rtcRoom enterRoom:roomID withCompletion:^(int errorCode, NSString *errMsg) {
    NSLog(@"enterRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);
}];
```

7. exitRoom

- 接口定义：- (void)exitRoom:(IExitRoomCompletionHandler)completion
- 接口说明：（会议创建者 OR 会议参与者）退出房间。
- 示例代码：

```
[self.rtcRoom exitRoom:^(int errorCode, NSString *errMsg) {  
    NSLog(@"exitRoom: errorCode[%d] errMsg[%@]", errorCode, errMsg);  
}];
```

8. startLocalPreview

- 接口定义：- (void)startLocalPreview:(UIView *)view
- 接口说明：（会议创建者 OR 会议参与者）启动摄像头预览，默认是使用前置摄像头，可以通过switchCamera切换前后摄像头
- 示例代码：

```
[self.rtcRoom startLocalPreview:pusherView];
```

9. stopLocalPreview

- 接口定义：- (void)stopLocalPreview;
- 接口说明：（会议创建者 OR 会议参与者）关闭摄像头预览。
- 示例代码：

```
[self.rtcRoom stopLocalPreview];
```

10.addRemoteView

- 接口定义：- (void)addRemoteView:(UIView *)view withUserID:(NSString *)userID playBegin:(IPlayBegin)playBegin playError:(IPlayError)playError
- 接口说明：（会议创建者 OR 会议参与者）播放会议参与者的远程视频画面，一般在收到 onPusherJoin（新会议参与者进入通知）时调用。
- 示例代码：

```
[self.rtcRoom addRemoteView:playerView withUserID:userID playBegin:nil playError:^(int errorCode, NSString *errMsg) {
```

```
//to do
};
```

11.deleteRemoteView

- 接口定义：- (void)deleteRemoteView:(NSString *)userID
- 接口说明：停止播放某个会议参与者视频，一般在收到 onPusherQuit（会议参与者离开）时调用。
- 示例代码：

```
[self.rtcRoom deleteRemoteView:userID];
```

12.sendRoomTextMsg

- 接口定义：- (void)sendRoomTextMsg:(NSString *)textMsg
- 接口说明：发送文本消息，房间里的其他人会收到 onRecvRoomTextMsg 通知。
- 示例代码：

```
[self.rtcRoom sendRoomTextMsg:textMsg];
```

13.switchToBackground

- 接口定义：- (void)switchToBackground:(UIImage *)pauselImage
- 接口说明：从前台切换到后台，关闭采集摄像头数据，推送默认 pauselImage 图片

14.switchToForeground

- 接口定义：- (void)switchToForeground
- 接口说明：由后台切换到前台，开启摄像头数据采集。

15.setBeautyStyle

- 接口定义：- (void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel
- 接口说明：设置美颜风格、磨皮程度、美白级别和红润级别。
- 参数说明：

参数	类型	说明
beautyStyle	int	磨皮风格：0：光滑 1：自然 2：朦胧
beautyLevel	int	磨皮等级：取值为 0-9.取值为 0 时代表关闭美颜效果.默认值: 0,即关闭美颜效果

参数	类型	说明
whitenessLevel	int	美白等级：取值为 0-9。取值为 0 时代表关闭美白效果。默认值: 0,即关闭美白效果
ruddinessLevel	int	红润等级：取值为 0-9。取值为 0 时代表关闭美白效果。默认值: 0,即关闭美白效果

- 示例代码：

```
[self.rtcRoom setBeautyStyle: beautyStyle beautyLevel: beautyLevel whitenessLevel: whitenessLevel ruddinessLevel: ruddinessLevel];
```

16.switchCamera

- 接口定义：- (void)switchCamera
- 接口说明：切换摄像头，前置摄像头时调用后变成后置，后置摄像头时调用后变成前置。该接口在启动预览 startCameraPreview 后调用才能生效，预览前调用无效。SDK启动预览默认使用前置摄像头。

17.setMute

- 接口定义：- (void)setMute:(BOOL)isMute
- 接口说明：设置静音接口。设置为静音后SDK不再推送麦克风采集的声音，而是推送静音。
- 参数说明：

参数	类型	说明
isMute	BOOL	是否静音

RTCRoomListener 接口详情

1. onGetPusherList

- 接口定义：- (void)onGetPusherList:(NSArray *)pusherInfoArray
- 接口说明：当新会议参与者加入房间时，会收到房间已存在的会议者列表。回调中您可以调用 addRemoteView 播放其他会议人员的视频。
- 示例代码：

```
- (void)onGetPusherList:(NSArray<PusherInfo *> *)pusherInfoArray {
dispatch_async(dispatch_get_main_queue(), ^{
// 播放其他人的画面
UIView *playerView = [[UIView alloc] init];
[playerView setBackgroundColor:UIColorFromRGB(0x262626)];
[self.view addSubview:playerView];

for (PusherInfo *pusherInfo in pusherInfoArray) {
[self.rtcRoom addRemoteView:playerView withUserID:pusherInfo.userID playBegin:nil playError:^(int
errCode, NSString *errMsg) {
//to do
}];
}
});
}
```

2. onPusherJoin

- 接口定义：- (void)onPusherJoin:(PusherInfo *)pusherInfo
- 接口说明：当新的会议参与者加入房间时，房间中其他的会议参与者都会收到该通知。回调中您可以调用 addRemoteView 播放这个新来的会议参与者的视频。
- 示例代码：

```
- (void)onPusherJoin:(PusherInfo *)pusherInfo {
dispatch_async(dispatch_get_main_queue(), ^{
UIView *playerView = [[UIView alloc] init];
[playerView setBackgroundColor:UIColorFromRGB(0x262626)];
[self.view addSubview:playerView];

[self.rtcRoom addRemoteView:playerView withUserID:pusherInfo.userID playBegin:nil playError:^(int
errCode, NSString *errMsg) {
//to do
}];
});
}
```

3. onPusherQuit

- 接口定义：- (void)onPusherQuit:(PusherInfo *)pusherInfo

- 接口说明：当会议参与者离开房间时，房间的其他会议参与者都会收到该通知。回调中您可以调用 `deleteRemoteView` 停止播放这个会议参与者的视频。
- 示例代码：

```
- (void)onPusherQuit:(PusherInfo *)pusherInfo {
    dispatch_async(dispatch_get_main_queue(), ^{
        UIView *playerView = [_playerViewDic objectForKey:pusherInfo.userID];
        [playerView removeFromSuperview];
    });
}
```

4. onRecvRoomTextMsg

- 接口定义：- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID userName:(NSString *)userName userAvatar:(NSString *)userAvatar textMsg:(NSString *)textMsg
- 接口说明：当会议参与者调用 `sendRoomTextMsg` 时，房间内的其他会议参与者都会收到该通知。
- 示例代码：

```
- (void)onRecvRoomTextMsg:(NSString *)roomID userID:(NSString *)userID userName:(NSString *)userName userAvatar:(NSString *)userAvatar textMsg:(NSString *)textMsg {
    //to do
}
```

5. onRoomClosed

- 接口定义：- (void)onRoomClose:(NSString *)roomID
- 接口说明：当房间销毁时，会议参与者会收到该通知。需要在回调中退出房间。
- 示例代码：

```
- (void)onRoomClose:(NSString *)roomID {
    [self alertTips:@"提示" msg:@"会话已被解散" completion:^(
    [self.navigationController popViewControllerAnimated:YES];
    });
}
```

6. onDebugMsg

- 接口定义：- (void)onDebugMsg:(NSString *)msg
- 接口说明：直播间日志回调。可以在回调中将日志保存到文件中，方便问题分析。
- 示例代码：

```
- (void)onDebugMsg:(NSString *)msg {  
    NSLog(@"%@",msg);  
}
```

7. onError

- 接口定义 : - (void)onError:(int)errCode errMsg:(NSString *)errMsg
- 接口说明 : 直播间错误回调
- 示例代码 :

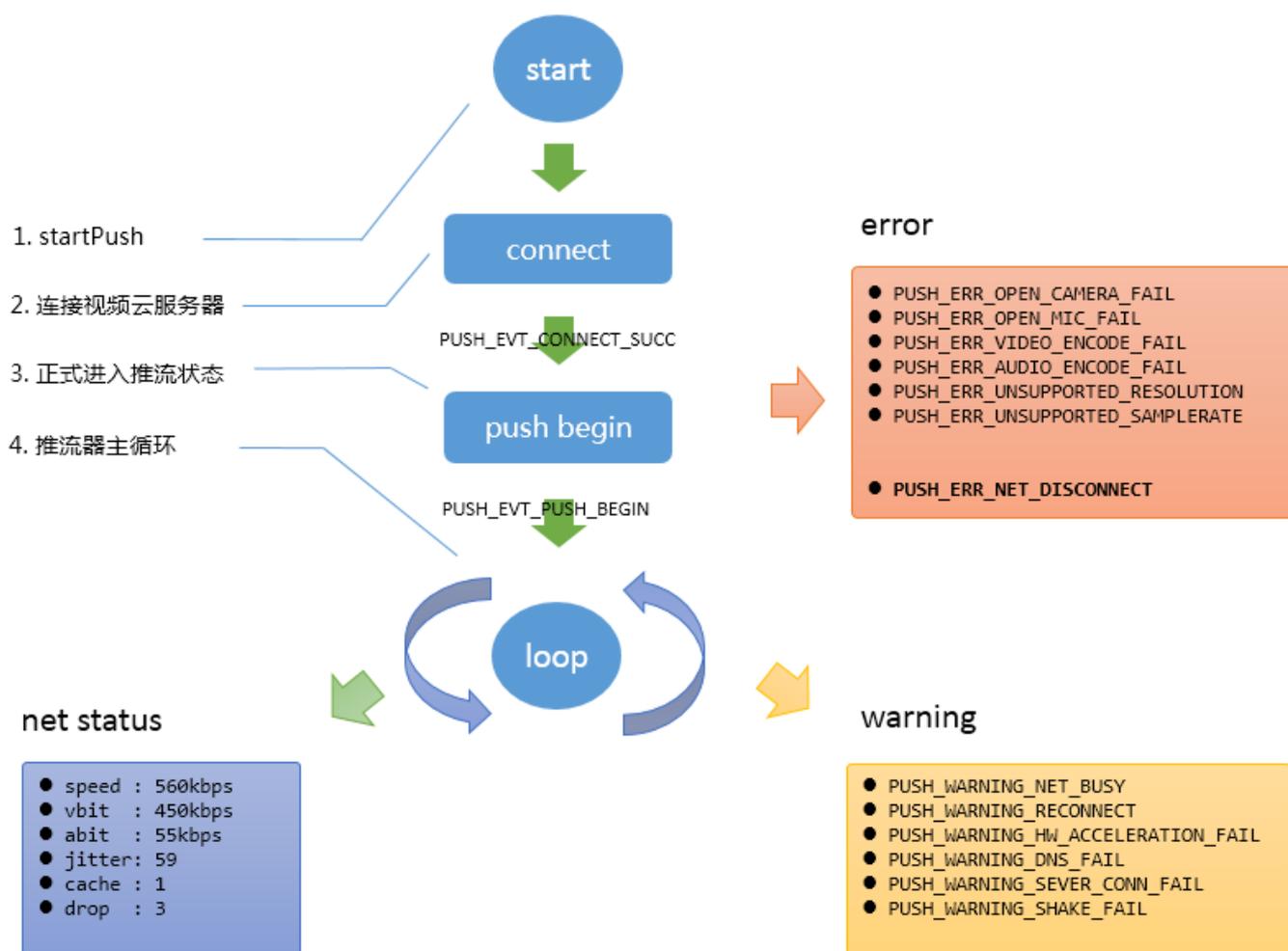
```
- (void)onError:(int)errCode errMsg:(NSString *)errMsg {  
    [self alertTips:@"提示" msg:errMsg completion:^(  
    [self.navigationController popViewControllerAnimated:YES];  
    });  
}
```

进阶功能

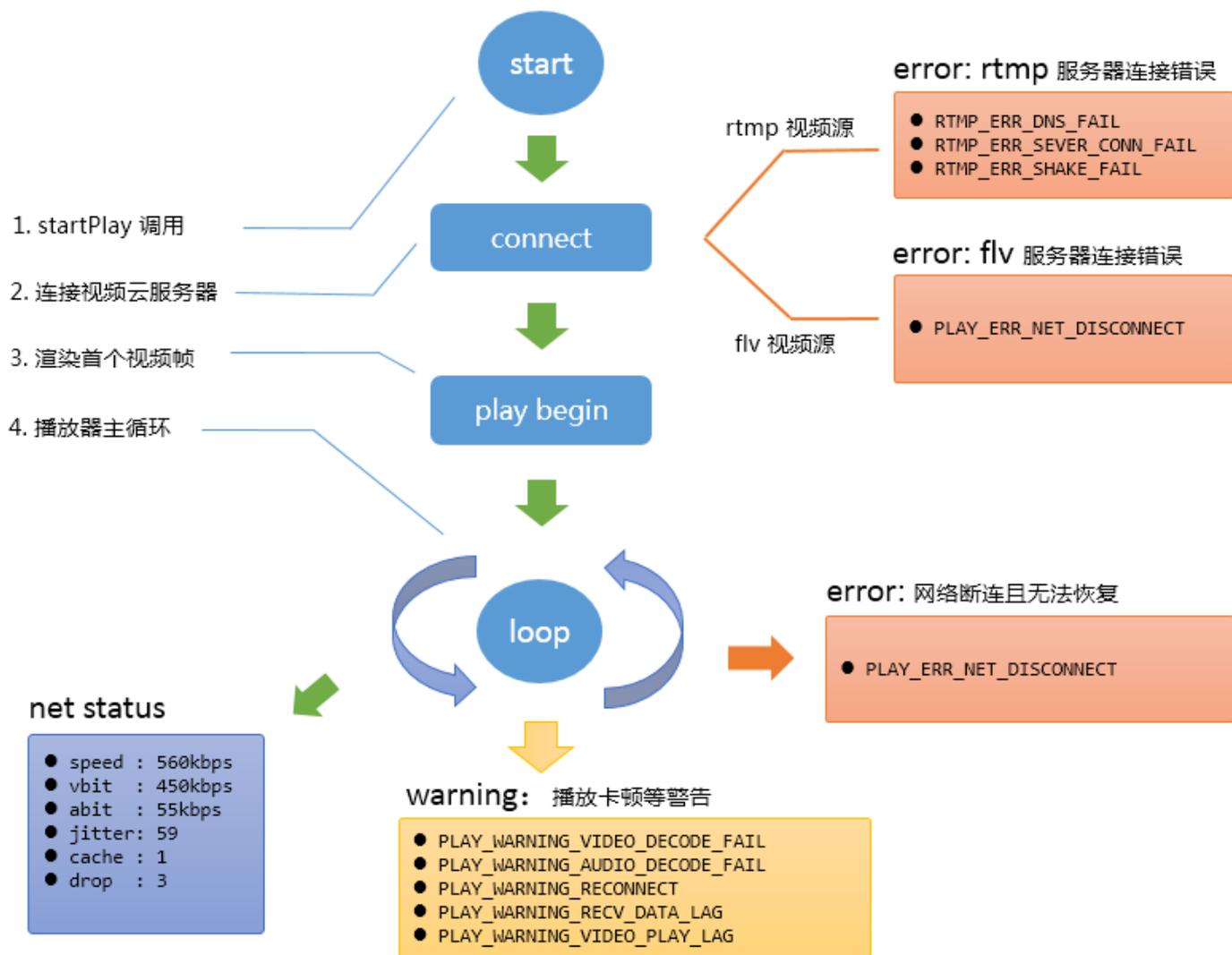
SDK内部原理

最近更新时间：2018-07-23 15:22:47

TXLivePusher



TXLivePlayer



SDK指标监控

最近更新时间：2018-07-23 15:47:21

TXLivePushListener

1. 如何获取推流的状态数据？

TXLivePushListener 的 onNetStatus 回调，会每隔 1-2 秒会将 SDK 内部的状态指标同步出来，其中如下指标比较有意义：



TXLivePushListener - onNetStatus

推流状态	参数名称	示例指标	指标含义说明
CPU	App CPU使用率	24.5% 71.3%	App: 24.5%
	系统CPU使用率		Sys: 71.3%
RES	推流分辨率	544*960	分辨率: 544*960
SPD	网络上传速度	1219kb/s	每秒上传1219kb数据
FPS	视频帧率	14	每秒14帧画面
GOP	关键帧间隔	5s	每隔5秒编一个I帧
ARA	音频码率	64kb/s	每秒编码出64kb音频数据
VRA	视频码率	1155kb/s	每秒编码出1155kb视频数据
SVR	推流服务器IP: 端口	125.94.63.141: 443	IP地址: 125.94.63.141 端口号: 443
AUD 0	当前aec类型	1 48000, 1	当前aec类型 (0: 没有开启aec; 1: 使用系统aec; 2: 使用trae)
	音频信息		音频采样率: 48000, 声道数: 1

推流状态	含义说明
NET_STATUS_CPU_USAGE	当前进程的CPU使用率和本机总体的CPU使用率
NET_STATUS_VIDEO_WIDTH	当前视频的宽度 (单位: 像素值)
NET_STATUS_VIDEO_HEIGHT	当前视频的高度 (单位: 像素值)
NET_STATUS_NET_SPEED	当前的发送速度 (单位: kbps)
NET_STATUS_VIDEO_BITRATE	当前视频编码器输出的比特率, 也就是编码器每秒生产了多少视频数据, 单位: kbps
NET_STATUS_AUDIO_BITRATE	当前音频编码器输出的比特率, 也就是编码器每秒生产了多少音频数据, 单位: kbps

推流状态	含义说明
NET_STATUS_VIDEO_FPS	当前视频帧率，也就是视频编码器每条生产了多少帧画面
NET_STATUS_CACHE_SIZE	音视频数据堆积情况，这个数字超过个位数，即说明当前上行带宽不足以消费掉已经生产的音视频数据
NET_STATUS_CODEC_DROP_CNT	全局丢包次数，为了避免延迟持续恶性堆积，SDK在数据积压超过警戒线以后会主动丢包，丢包次数越多，说明网络问题越严重。
NET_STATUS_SERVER_IP	连接的推流服务器的IP

2. 哪些状态指标有参考价值？

BITRATE vs NET_SPEED

BITRATE(= VIDEO_BITRATE + AUDIO_BITRATE) 指的是编码器每秒产生了多少音视频数据要推出去，NET_SPEED 指的是每秒钟实际推出了多少数据。

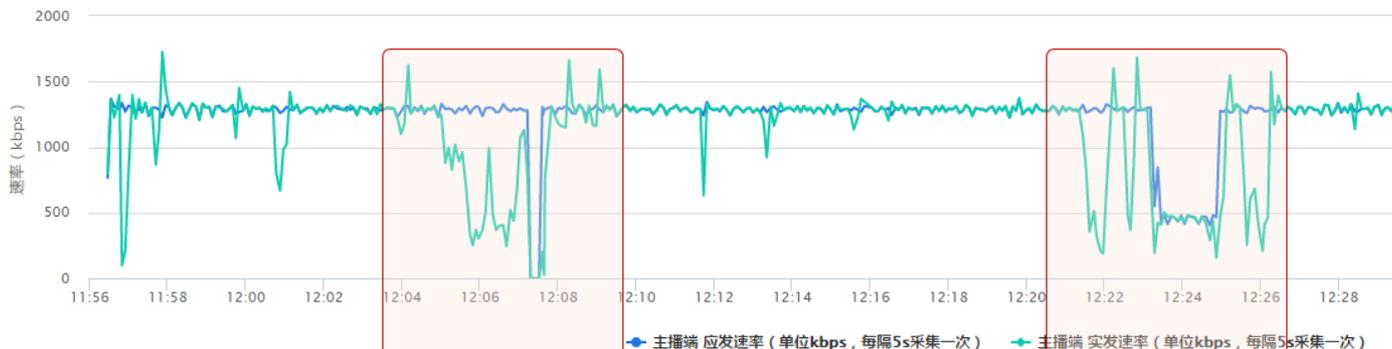
- 如果 BITRATE == NET_SPEED 的情况是常态，则推流质量会非常良好；
- 如果 BITRATE >= NET_SPEED ，且这种情况的持续时间比较长，音视频数据会堆积在主播的手机上撑大 CACHE_SIZE 并最终被 SDK 丢弃形成 DROP_CNT 。

CACHE_SIZE & DROP_CNT

如果主播当前网络的上传速度不OK，那么很容易出现 BITRATE >= NET_SPEED 的情况，此时音视频数据会在主播的手机上积压起来，积压的严重程度以 CACHE_SIZE 进行评估，如果 CACHE_SIZE 超过警戒阈值，SDK 会主动丢弃一些音视频数据，进而触发 DROP_CNT 的累加。

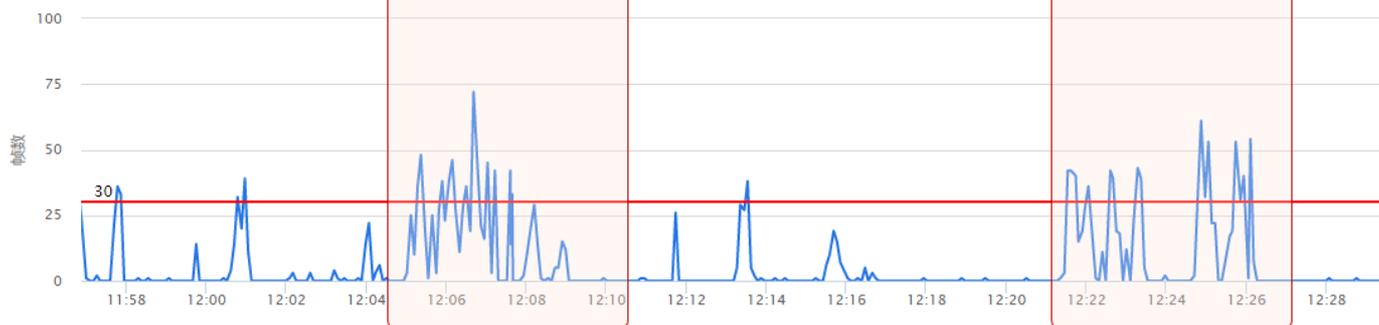
主播端 应发速率-实发速率曲线图

图例说明：视频生成码率曲线和推流速率曲线越重合，说明主播端推送越流畅，反之主播推送不流畅，易引起播放端观看卡顿。（在图上按下鼠标左键拖动可以查看细节）



主播端 音视频数据堆积情况

图例说明：主播端不能及时发送到云端的音视频数据会被堆积起来，数据堆积超过三秒（下图红色警示线以上部分）会被主动丢弃，因此当主播的上行网络不理想时，观看端会出现



CPU_USAGE

- 如果 **系统CPU使用率** 超过 80%，音视频编码的稳定性会受到影响，可能导致画面和声音的随机卡顿。
- 如果 **系统CPU使用率** 经常 100%，会导致视频编码帧率不足，音频编码跟不上，必然导致画面和声音的严重卡顿。

很多客户会遇到的一个问题：App 在线下测试时性能表现极佳，但在 App 外发上线后，前排房间里的互动消息的滚屏和刷新会产生极大的 CPU 消耗导致直播画面卡顿严重。

SERVER_IP

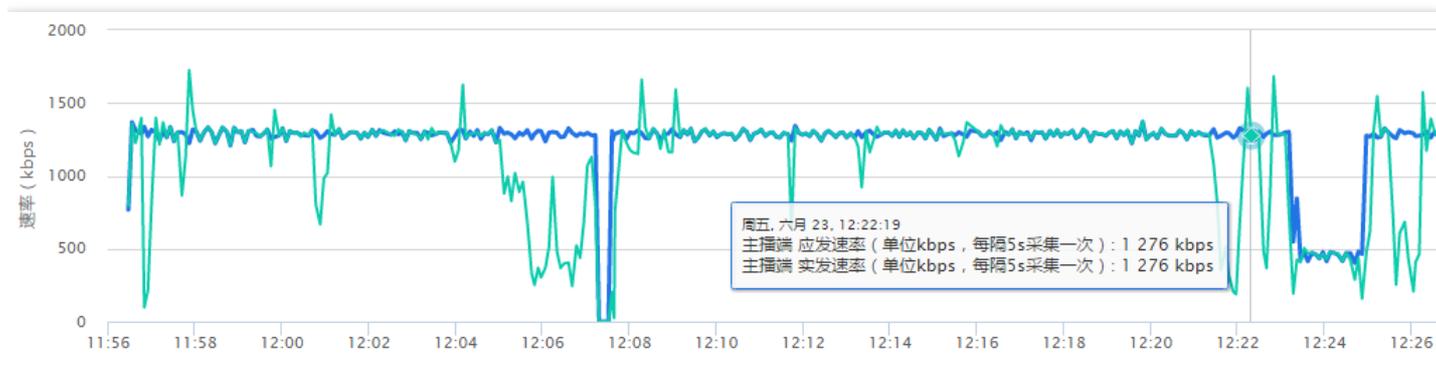
如果主播到 SERVER_IP 给出的 ip 地址的 ping 值很高（比如超过 500ms），那么推流质量一定无法保障。**就近接入**是我们腾讯云应该做好的事情，如您发现有这样的案例，请反馈给我们，我们的运维团队会持续调整和优化之。

3. 如何看懂腾讯云推流图表？

在 [直播控制台-质量监控](#) 您可以看到您所属账户里的直播间情况，以及每个直播间的推流质量数据：

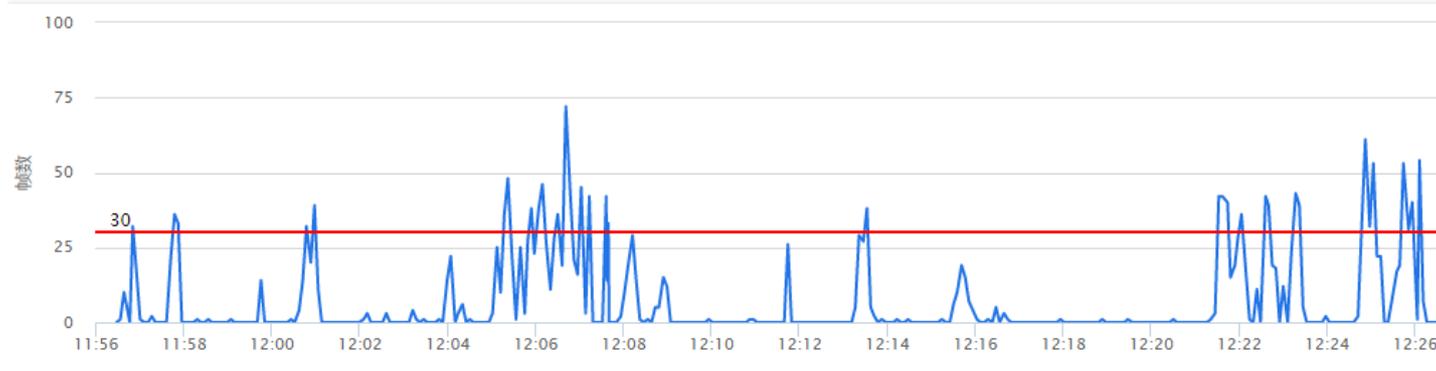
主播端-应发速率-实发速率曲线图

蓝色曲线代表 BITRATE 的统计曲线，即 SDK 产生的音视频数据，即绿色曲线发表实际网络发出去多少。两条线重合度越高表示推流质量越好。



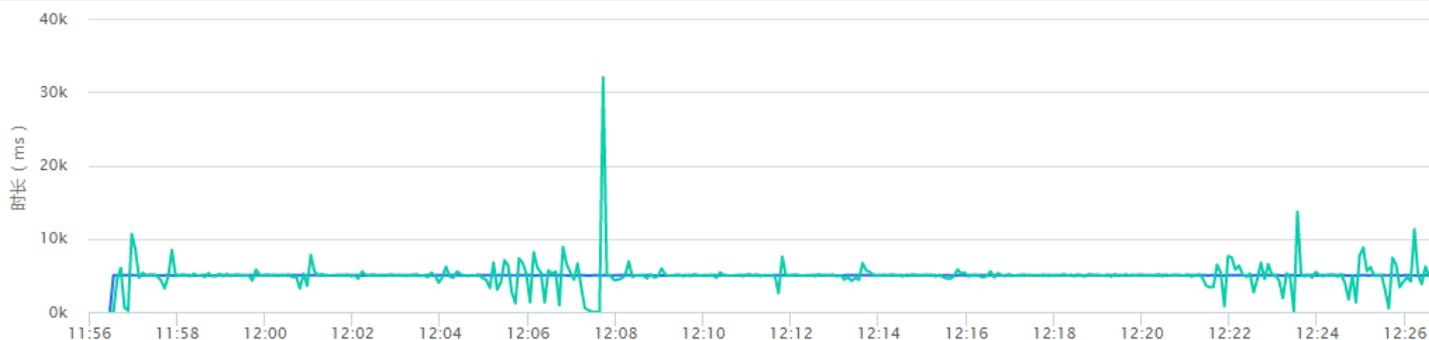
主播端-音视频数据堆积情况

- o 如果曲线始终贴着 0 刻度线走，说明整个推流过程非常顺畅，一点都没有堆积。
- o 如果出现 > 0 的部分，说明当时有网络波动导致数据积压，有可能在播放端产生轻微卡顿和音画不同步的现象；
- o 如果堆积超出红色警戒线，说明已经产生了丢包，必然会在播放端产生卡顿和音画不同步的现象。



云端-应收视频时长-实收视频时长曲线

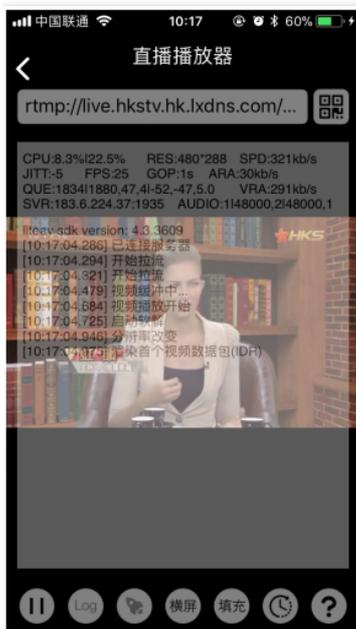
这里是腾讯云服务端的统计图表，如果您不是使用腾讯云 SDK 推流，那么您将只能看到这个图表，前面两个（数据来自 SDK）是看不到的。蓝绿两条线重合度越高，说明推流质量越好。



TXLivePlayListener

1. 如何获取播放的状态数据？

TXLivePlayListener 的 onNetStatus 回调，会每隔 1-2 秒会将 SDK 内部的状态指标同步出来，其中如下指标比较有意义：



TXLivePlayListener - onNetStatus

拉流状态	参数名称	示例指标	指标含义说明
CPU	App CPU使用率	8.3% 22.5%	App: 8.3%
	系统CPU使用率		Sys: 22.5%
RES	拉流分辨率	480*288	分辨率: 480*288
SPD	网络下载速度	321kb/s	每秒下载321kb数据
FPS	视频帧率	25	每秒25帧画面
GOP	关键帧间隔	1s	每隔1秒有一个I帧
ARA	音频码率	30kb/s	每秒需解码30kb音频数据
QUE	音频缓冲时长	1834 1880, 47, 4 -52, -47, 5.0	音频缓冲时长: 1834毫秒
	视频缓冲时长		视频缓冲时长: 1880毫秒
	视频缓冲总帧数		视频缓冲总帧数: 47帧
	视频解码器缓冲帧数		视频解码器缓冲帧数: 4帧
	音视频网络收帧时间差		音频与视频网络收帧时间差: -52毫秒
	音视频当前帧渲染时间差		音频与视频当前帧渲染时间差: -47毫秒
VRA	平衡点		平衡点
VRA	视频码率	291kb/s	每秒需解码291kb视频数据
SVR	拉流服务器IP: 端口	183.6.224.37:1935	IP地址: 183.6.224.37 端口号: 1935
AUD 0	当前aec类型	1 48000, 2 48000, 1	当前aec类型 (0: 没有开启aec; 1: 使用系统aec; 2: 使用trae)
	原始音频信息		原始音频采样率: 48000, 原始声道数: 2
	播放音频信息		播放音频采样率: 48000, 播放声道数: 1

播放状态	含义说明
NET_STATUS_CPU_USAGE	当前瞬时CPU使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 - 高

播放状态	含义说明
NET_STATUS_NET_SPEED	当前的网络的下载速度
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_CACHE_SIZE	播放缓冲区 (jitterbuffer) 大小，缓冲区越小越难以抵抗卡顿
NET_STATUS_SERVER_IP	当前连接的服务器IP

2. 哪些状态指标有参考价值？

NET_STATUS_CACHE_SIZE

这个指标用于衡量播放缓冲区 (jitterbuffer) 的大小：

- CACHE_SIZE 越大，播放的延迟越高，但网络有波动时越不容易卡顿。
- CACHE_SIZE 越小，播放的延迟越低，但下载速度稍有风吹草动就可能引发卡顿。

TXLivePlayer 的有三种模式用于控制播放缓冲区的大小，设置方法可以参考【基础功能-播放功能】的对接文档：

播放模式	卡顿率	延迟	使用场景	原理简述
极速模式	较高	2s - 3s	美女秀场	尽可能保证较低的播放缓冲区，进而减少延迟，适合 1Mbps 左右的低码率场景。
流畅模式	较低	>= 5s	游戏直播	确保随时都有较大的播放缓冲区，通过牺牲延迟来应对大码率（2M左右）下的网络波动的影响。
自动模式	自适应	2s - 8s	泛场景	缓冲区大小自动调节： 网络越好，延迟月底；网络越差，延迟越高。

Qos流量控制

最近更新时间：2018-07-23 12:03:25

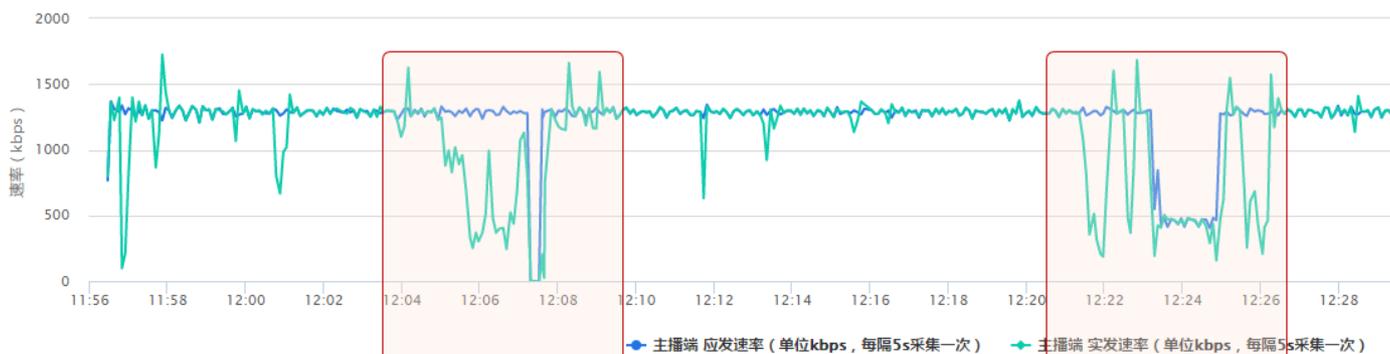
背景说明

RTMP 推流质量对于观看体验非常关键，因为如果主播的推流质量不佳，那么所有观众看到的视频画面都是卡顿的，据统计，视频云客户群 80% 以上的直播间卡顿问题均是由于 RTMP 推流质量不佳导致的。

在众多的推流质量问题中，主播的上行网络不给力引发的问题又是最主要的，上行带宽不足会导致音视频数据在主播端堆积并丢弃，从而使观众端看到的视频画面出现卡顿甚至长时间卡死。

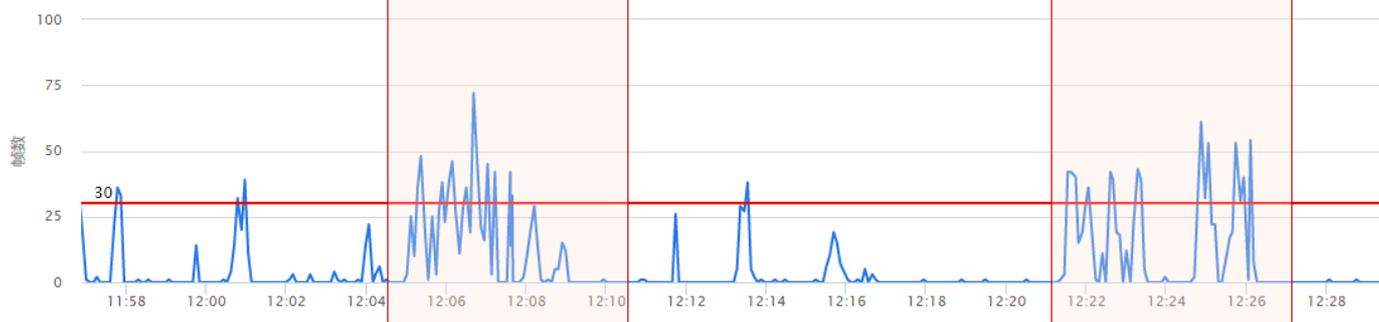
主播端 应发速率-实发速率曲线图

图例说明：视频生成码率曲线和推流速率曲线越重合，说明主播端推送越流畅，反之主播推送不流畅，易引起播放端观看卡顿。（在图上按下鼠标左键拖动可以查看细节）



主播端 音视频数据堆积情况

图例说明：主播端不能及时发送到云端的音视频数据会被堆积起来，数据堆积超过三秒（下图红色警示线以上部分）会被主动丢弃，因此当主播的上行网络不理想时，观看端会出现



所以，优化主播上行卡顿问题能够有效地提升推流质量，进而提升观看端的质量，尤其在国内运营商普遍限制上行带宽的情况下。

但是网络问题不随人的意志为转移，主播家里买的 4Mbps 宽带套餐，不可能因为主播装一个 App 就让其变成 8Mbps，因此，我们只能采用顺势而为的做法 —— **主动适应上行网络的情况。**

快速对接

使用 TXLivePusher 的 setVideoQuality 接口的 参数即可开启 Qos 流控，开启 Qos 流控以后 SDK 会根据主播上行网络的好坏决定视频的清晰度。



- **quality**

SDK 提供了六种基础档位，根据我们服务大多数客户的经验进行积累和配置。其中 STANDARD、HIGH、SUPER 适用于直播模式，MAIN_PUBLISHER 和 SUB_PUBLISHER 适用于连麦直播中的大小画面，VIDEOCHAT 用于实时音视频。

- **adjustBitrate**

是否开启 Qos 流量控制，开启后 SDK 会根据主播上行网络的好坏自动调整视频码率。相应的代价就是，主播如果网络不好，画面会很模糊且有很多马赛克。

- **adjustResolution**

是否允许动态分辨率，开启后 SDK 会根据当前的视频码率选择相匹配的分辨率，这样能获得更好的清晰度。相应的代价就是，动态分辨率的直播流所录制下来的文件，在很多播放器上会有兼容性问题。

quality	Qos (开启)	Qos (关闭)	动态分辨率 (启用)	动态分辨率 (关闭)
STANDARD	300kbps – 800kbps	800kbps	<ul style="list-style-type: none"> ● 270*480 ● 360*640 	360*640
HIGH	600kbps – 1200kbps	1200kbps	<ul style="list-style-type: none"> ● 360*640 ● 540*960 	540*960
SUPER	600kbps – 1800kbps	1800kbps	<ul style="list-style-type: none"> ● 360*640 ● 540*960 ● 720*1280 	720*1280
VIDEOCHAT	200kbps – 800kbps	不支持关闭	<ul style="list-style-type: none"> ● 190*320 ● 270*480 ● 360*640 	不支持关闭

精细校调

如果您觉得 `setVideoQuality` 里的默认参数无法满足您的需求，您可以通过 `TXLivePushConfig` 进行定制：

- **enableAutoBitrate**

是否开启码率自适应，也就是 Qos 流控，如果开启，开启后 SDK 会根据主播上行网络的好坏决定视频的码率。

- **autoAdjustStrategy**

在开启码率自适应以后才能设置，否则设置是无效的，`autoAdjustStrategy`支持如下四种策略：

策略名称	策略说明
AUTO_ADJUST_BITRATE_STRATEGY_1	整个直播过程中不断地探测网速并相应地进行调整，适合秀场直播等场景。
AUTO_ADJUST_BITRATE_RESOLUTION_STRATEGY_1	在调整码率的同时相应的调整分辨率，以保持码率和分辨率之间的平衡
AUTO_ADJUST_BITRATE_STRATEGY_2	开播的最初半分钟里快速探测网速，之后会尽量少地进行调整，适合手游场景。
AUTO_ADJUST_BITRATE_RESOLUTION_STRATEGY_2	在调整码率的同时相应的调整分辨率，以保持码率和分辨率之间的平衡

- **videoBitrateMin**：最低码率，在开启码率自适应以后才能设置，否则设置是无效。
- **videoBitrateMax**：最高码率，在开启码率自适应以后才能设置，否则设置是无效。
- **videoBitratePIN**：初始码率，`videoBitrateMin <= videoBitratePIN <= videoBitrateMax`。

编码参数调整

最近更新时间：2018-07-23 16:04:08

参数定制

如果您希望定制视频编码参数，音频编码参数等等，您可以通过设置Config对象实现您的自定义需求，目前我们支持的setting接口如下：

参数名	含义	默认值
audioSampleRate	音频采样率：录音设备在一秒钟内对声音信号的采集次数	44100
enableNAS	噪声抑制：开启后可以滤掉背景杂音（32000以下采样率有效）	关闭
enableHWAcceleration	视频硬编码：开启后最高可支持720p，30fps视频采集	开启
videoFPS	视频帧率：即视频编码器每秒生产出多少帧画面，注意由于大部分机器性能不足以支持30FPS以上的编码，推荐您设置FPS为20	20
videoResolution	视频分辨率：目前提供四种16：9分辨率可供您选择	640 * 360
videoBitratePIN	视频比特率：即视频编码器每秒生产出多少数据，单位 kbps	800
enableAutoBitrate	带宽自适应：该功能会根据当前网络情况，自动调整视频比特率	关闭
videoBitrateMax	最大输出码率：只有开启自适应码率, 该设置项才能起作用	1200
videoBitrateMin	最小输出码率：只有开启自适应码率, 该设置项才能起作用	800
videoEncodeGop	关键帧间隔（单位：秒）即多少秒出一个帧	3s
homeOrientation	设置视频图像旋转角度，比如是否要横屏推流	home在右边（0）home在下面（1）home在左面（2）home在上面（3）

参数名	含义	默认值
beautyFilterDepth	美颜级别：支持1~9 共9个级别，级别越高，效果越明显。0表示关闭	关闭
frontCamera	默认是前置还是后置摄像头	前置
watermark	水印图片（UIImage对象）	腾讯云Logo（demo）
watermarkPos	水印图片相对左上角坐标的位置	（0,0）

设置方法

这些参数的设置推荐您在启动推流之前就指定，因为大部分设置项是在重新推流之后才能生效的。参考代码如下：

```
//成员变量中声明_config和_pusher
....
//初始化_config
_config = [[TXLivePushConfig alloc] init];

//修改参数设置为声音44100采样率，视频800固定码率
_config.audioSampleRate = 44100;
_config.enableAutoBitrate = NO;
_config.videoBitratePIN = 800;

//初始化_pusher
_pusher = [[TXLivePush alloc] initWithConfig:_config];
```

定制视频数据

最近更新时间：2018-07-23 16:01:48

定制推流画面

方案一：修改OpenGL纹理

研发实力不俗的客户，会有自定义图像处理的需求（比如堆加字幕），同时又希望复用rtmp sdk的整体流程，如果是这样，您可以按照如下攻略进行定制。

- 设置视频处理回调

设置 `TXLivePush` 的 `videoProcessDelegate` 代理点，即可实现对视频画面的定制。

@protocol TXVideoCustomProcessDelegate <NSObject>

```
/**
 * 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
 * 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_TEXTURE_2
 * D
 */
-(GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height;

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

@end
```

- 在回调函数中对视频数据进行加工

实现 `TXVideoCustomProcessDelegate` 的 `onPreProcessTexture` 函数，以实现对视频画面的自定义处理。
`textureId` 指定的纹理是一块类型为 `GLES20.GL_TEXTURE_2D` 的纹理。

对于 `texture` 数据的操作，需要一定的 `OpenGL` 基础知识，另外计算量不宜太大，因为 `onPreProcessTexture` 的调用频率跟 `FPS` 相同，过于繁重的处理很容易造成 `GPU` 过热。

方案二：自己采集数据

如果您只希望使用 SDK 来编码和推流（比如已经对接了商汤等产品），音视频采集和预处理（即美颜、滤镜这些）全部由自己的代码来控制，可以按如下步骤实现：

- **Step1. 不再调用 TXLivePush 的 startPreview 接口**

这样 SDK 本身就不会再采集视频数据和音频数据，而只是启动预处理、编码、流控、发送 等跟推流相关的工作。

- **Step2. 通过 TXLivePushConfig 设置 customModeType**

```
#define CUSTOM_MODE_AUDIO_CAPTURE 0X001 //客户自己采集声音
#define CUSTOM_MODE_VIDEO_CAPTURE 0X002 //客户自己采集视频
```

//如果声音和视频都要自己采集，可以设置 customModeType 为 3

```
@interface TXLivePushConfig : NSObject
@property(nonatomic, assign) int customModeType;
@end
```

- **Step3. 使用 sendVideoSampleBuffer 向SDK填充Video数据**

sendVideoSampleBuffer 的作用是向 SDK 塞入您采集和处理后的视频数据，目前支持 RGBA 和 NV12 两种格式。

```
TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
config.customModeType = CUSTOM_MODE_VIDEO_CAPTURE; // 自定义采集数据
TXLivePush pusher = [[TXLivePush alloc] initWithConfig:config];
```

//可以塞入自己采集和处理的 RGBA 或者 NV12 数据

```
[pusher sendVideoSampleBuffer:sampleBuffer];
```

- **Step4. 使用 sendAudioSampleBuffer 向SDK填充Audio数据**

sendAudioSampleBuffer 的作用是向 SDK 塞入您采集和处理后的音频数据，请使用单声道、16位宽、48000Hz 的 PCM 声音数据。

该函数支持两种 RPSampleBufferType，RPSampleBufferTypeAudioApp 和 RPSampleBufferTypeAudioMic，前者用于 replaykit，后者用于常规的麦克风采集（也用于 replaykit）。

```
TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
config.customModeType = CUSTOM_MODE_AUDIO_CAPTURE; // 自定义采集数据
```

```
TXLivePush pusher = [[TXLivePush alloc] initWithConfig:config];
```

```
//可以塞入自己采集和处理的 声音数据
```

```
[s_txLivePublisher sendAudioSampleBuffer:sampleBuffer withType:RPSampleBufferTypeAudioMic];
```

定制播放数据

- 设置 TXLivePlayConfig 的 TXVideoCustomProcessDelegate 属性

```
@interface TXLivePlayer : NSObject
```

```
// 设置之后, Player 的每帧画面都会经过 onPlayerPixelBuffer
```

```
@property(nonatomic, weak) id <TXVideoCustomProcessDelegate> videoProcessDelegate;
```

- 通过 onPlayerPixelBuffer 回调捕获 Player 的图像数据。

如果当前 Player 是硬解码模式, pixelBuffer 的图像格式是 **NV12**, 如果当前 Player 是软解码模式, pixelBuffer 的图像格式是 **i420**。

onPlayerPixelBuffer 返回值为 YES 时, SDK 不会继续做图像渲染, 这种方式可以解决 OpenGL 线程冲突的问题。

```
@protocol TXVideoCustomProcessDelegate <NSObject>
```

```
@optional
```

```
/**
```

```
* 视频渲染对象回调
```

```
* @param pixelBuffer 渲染图像
```

```
* @return 返回YES则SDK不再显示; 返回NO则SDK渲染模块继续渲染
```

```
* 说明: 渲染图像的数据类型为config中设置的renderPixelFormatType
```

```
*/
```

```
-(BOOL)onPlayerPixelBuffer:(CVPixelBufferRef)pixelBuffer;
```

```
@end
```