移动直播

Android端集成

产品文档

腾讯云

文档目录

Android端集成
API 接口

最近更新时间：2018-09-12 18:25:37

下面是腾讯视频云Android SDK的主要接口列表，分为TXLivePusher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。

TXLivePusher

API列表

名称	描述
setConfig(TXLivePushConfig config)	设置推流配置信息
getConfig()	获取推流配置信息
setPushListener(ITXLivePushListener listener)	设置推流事件状态回调
setVideoProcessListener(VideoCustomProcessListener listener)	设置自定义视频处理回调
startCameraPreview(TXCloudVideoView view)	启动摄像头预览
stopCameraPreview(isNeedClearLastImg)	关闭摄像头预览
startPusher(String url)	启动推流
stopPusher()	停止推流
pausePusher()	暂停推流
resumePusher()	恢复推流
startScreenCapture()	启动录屏
stopScreenCapture()	结束录屏
setVideoQuality(quality, adjustBitrate, adjustResolution)	设置推流视频质量，是否开启 Qos 流量控制， 是否允许动态分辨率
setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel)	设置美颜风格、磨皮程度、美白级别和红润级别

名称	描述
setExposureCompensation()	调整曝光
setFilter()	设置指定素材滤镜特效
setChinLevel()	设置下巴拉伸或收缩，企业版本有效
setEyeScaleLevel()	设置大眼级别，企业版本有效
setFaceShortLevel()	设置设置短脸，企业版本有效
setFaceVLevel()	设置V脸，企业版本有效
setMotionTmpl()	设置动效，企业版本有效
setGreenScreenFile()	设置绿幕文件，企业版本有效
switchCamera()	切换前后置摄像头，支持在推流中动态切换
setMute(mute)	静音接口
setRenderRotation(rotation)	设置本地预览图像的顺时针旋转角度
setMirror(enable)	设置播放端水平镜像
playBGM(path)	播放背景音乐，用于混音处理
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
getMusicDuration(path)	获取音乐文件时长
setMicVolume()	设置混音时麦克风的音量大小
setBGMVolume()	设置混音时背景音乐的音量大小
setVoiceChangerType()	设置变声类型
setReverb()	设置推流端混响效果
startRecord(videoFilePath)	开始视频录制

名称	描述
stopRecord()	停止视频录制
setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)	设置视频录制回调
snapshot(ITXSnapshotListener)	截取视频画面
setAudioProcessListener(listener)	设置自定义音频处理回调
sendCustomVideoData(buffer, bufferType, w, h)	推送自定义视频数据
sendCustomVideoTexture(textureID, w, h)	发送客户自定义的视频纹理
sendCustomPCMData(pcmBuffer)	推送自定义音频数据
sendMesageEx(byte[] msg)	向播放端发送消息 (消息大小不允许超过2K)

TXLivePushConfig

API列表

名称	描述
enableAEC(enable)	开启回声消除(连麦专用)
enableNearestIP(enable)	开启就近选路
enablePureAudioPush(enable)	开启纯音频推流
enableScreenCaptureAutoRotate(enable)	设置录屏是否自适应旋转
setConnectRetryCount(count)	设置推流端重连次数
setConnectRetryInterval(interval)	设置推流端重连间隔
setEyeScaleLevel(level)	设置大眼效果
setFaceSlimLevel(level)	设置瘦脸效果
setFrontCamera(front)	设置是否使用前置摄像头
setHomeOrientation(homeOrientation)	设置采集的视频的旋转角度
setPauseFlag(flag)	设置后台推流,后台推流的选项

名称	描述
setPauseImg(img)	设置后台暂停,后台播放的暂停图片
setPauseImg(time, fps)	设置推流暂停时,后台播放暂停图片的方式
setVideoEncodeGop(gop)	设置视频编码GOP
setVideoResolution(resolution)	设置视频帧率
setWatermark(watermark, x, y, width)	设置水印图片
setWatermark(watermark, x,y)	设置水印图片

TXLivePlayer

API列表

名称	描述
setConfig(TXLivePlayConfig config)	设置播放配置信息
getConfig()	获取播放配置信息
setPlayerView(TXCloudVideoView view)	把渲染 view 绑定到 player
setPlayListener(ITXLivePlayListener listener)	设置TXLivePlayer 的回调
startPlay(url, type)	开始播放
stopPlay(isNeedClearLastImg)	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
setAudioRoute(audioRoute)	设置声音播放模式(切换扬声器，听筒)
switchStream(playUrl)	flv直播无缝切换
enableHardwareDecode(enable)	启用或禁用视频硬解码.
isPlaying()	是否正在播放

名称	描述
setRenderMode(mode)	设置图像的渲染（填充）模式
setRenderRotation(rotation)	设置图像的顺时针旋转角度
startRecord(recordType)	开启截流录制
stopRecord()	停止截流录制
setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)	设置视频录制回调
setMute(mute)	设置静音
snapshot(TXLivePlayer.ITXSnapshotListener listener)	截取视频图像
setSurface(Surface surface)	设置渲染surface
addVideoRawData(byte[] yuvBuffer)	设置播放yuv数据接收buffer
setVideoRawDataListener(ITXVideoRawDataListener listener)	设置视频yuv数据回调
setAudioRawDataListener(ITXAudioRawDataListener listener)	设置音频pcm数据回调

TXLivePlayConfig

API列表

名称	描述
enableAEC(enable)	开启回声消除(连麦专用)
setConnectRetryCount(count)	设置播放器重连次数
setConnectRetryInterval(interval)	设置播放器重连间隔
setEnableMessage(enable)	设置消息通道是否打开
setAutoAdjustCacheTime(bAuto)	设置是否根据网络状况自动调整播放器缓存时间
setCacheTime(time)	设置播放器缓存时间
setMaxAutoAdjustCacheTime(time)	设置自动调整时播放器最大缓存时间
setMinAutoAdjustCacheTime(time)	设置自动调整时播放器最小缓存时间

TXLivePusher接口详情

1.setConfig(TXLivePushConfig config)

接口详情：void setConfig(TXLivePushConfig config)

设置推流器配置信息.

- **参数说明**

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

- **示例代码**：

```
TXLivePushConfig mLivePushConfig = new TXLivePushConfig();
mLivePushConfig.setBeautyFilter(...);
.....
mLivePusher.setConfig(mLivePushConfig);
```

2.getConfig()

接口详情：TXLivePushConfig getConfig()

获取推流配置信息

- **示例代码**：

```
mLivePusher.getConfig()
```

3.setPushListener(ITXLivePushListener listener)

接口详情：void setPushListener(ITXLivePushListener listener)

设置推流事件状态回调 ，具体的事件及状态参看推流事件说明

- **参数说明**

参数	类型	说明
listener	ITXLivePushListener	推流事件回调接口

- **示例代码**：

```
mLivePusher.setPushListener(new ITXLivePushListener() {
@Override
public void onPushEvent(int i, Bundle bundle) {
// ...
}

@Override
public void onNetStatus(Bundle bundle) {
// ...
}
});
```

4.setVideoProcessListener(VideoCustomProcessListener listener)

接口详情：void setVideoProcessListener(VideoCustomProcessListener listener)

设置自定义视频处理回调，在主播预览及编码前回调出来，用户可以用来做自定义美颜或者增加视频特效

- **参数说明**

参数	类型	说明
listener	VideoCustomProcessListener	自定义视频处理回调

- **示例代码**：

```
public interface VideoCustomProcessListener {

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

/**
* 在OpenGL线程中回调，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GLES20.GL_TEXTURE_2D，接口返回给SDK的纹理类型也必须是GLES20.G
```

```
L_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height);

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

5. startCameraPreview(TXCloudVideoView view)

接口详情：void startCameraPreview(TXCloudVideoView view)

启动摄像头预览，默认是使用前置摄像头，可以通过TXLivePushConfig.setFrontCamera()设置。接口支持推流过程中切换不同的 TXCloudVideoView 。要先调用 stopCameraPreview()，再调用 startCameraPreview()

- **参数说明**

参数	类型	说明
view	TXCloudVideoView	预览视频的渲染view

- **示例代码**：

```
TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mLivePusher.startCameraPreview(mCaptureView);
```

6. stopCameraPreview(isNeedClearLastImg)

接口详情：void stopCameraPreview(boolean isNeedClearLastImg)

关闭摄像头预览。可以保留或者清除最后一帧画面，如果停止预览后还留在推流页面建议保留，否则清除。

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
isNeedClearLastImg	boolean	是否需要清除最后一帧画面；true：清除最后一帧画面， false：保留最后一帧画面.

- **示例代码**：

```
mLivePusher.stopCameraPreview(true);
```

7.startPusher(url)

接口详情：void startPusher(String url)

启动推流 (在 startPush 之前需要先 startCameraPreview 启动摄像头预览，否则数据无法推流上去)

注意推流 url 有排他性，也就是一个推流 Url 同时只能有一个推流端在推流

- **参数说明**

参数	类型	说明
url	const String	一个合法的推流地址，支持 rtmp 协议（URL 以 “rtmp://” 打头 ，腾讯云推流 URL 的获取方法见 DOC ）

- **示例代码**：

```
mLivePusher.startPusher(rtmpUrl.trim());
```

8.stopPusher()

接口详情：void stopPusher()

停止推流

- **示例代码**：

```
mLivePusher.stopPusher();
```

9.pausePusher()

接口详情：void pausePusher()

暂停摄像头推流。在正常推流中，如果App被切到后台，可以调用 pausePusher，这样SDK会停止采集摄像头的画面，同时会推送一个垫片和静音数据。该垫片可以通过 TXLivePushConfig.setPauseImg() 接口设置，如果没有设置则垫片是一个纯黑色画面。

- **示例代码**：

```
mLivePusher.pausePusher();
```

10.resumePusher()

接口详情：void resumePusher()

恢复摄像头推流。等App切回前台之后，调用 resumePusher，SDK 会停止推送垫片，继续采集摄像头的画面和麦克风声音，进行推流

- **示例代码**：

```
mLivePusher.resumePusher();
```

11.startScreenCapture()

接口详情：void startScreenCapture()

启动屏幕录制。由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出一个提示，旨在告诫用户："有 App 要截取您屏幕上的所有内容"。

注意：该接口在Android API 21接口才生效。录屏接口和摄像头预览（startCameraPreview）互斥，同时只能由一个生效

- **示例代码**：

```
mLivePusher.startScreenCapture();
```

12.stopScreenCapture()

接口详情：void stopScreenCapture()

停止屏幕录制。

- **示例代码**：

```
mLivePusher.stopScreenCapture();
```

13.setVideoQuality(quality, adjustBitrate, adjustResolution)

接口详情：void setVideoQuality(int quality, boolean adjustBitrate, boolean adjustResolution)

设定推流的画面清晰度。SDK 中提供了六种基础档位，根据主播的场景模式来选择相对应的模式。不同模式对应不同的分辨率和码率。根据我们服务大多数客户的经验进行积累和配置，我们找到分辨率和码率的最优的搭配，从而

实现很好的画质效果。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。

具体值如下：

- **参数说明**

参数	类型	说明
quality	int	画质档次
adjustBitrate	boolean	动态码率开关
adjustResolution	boolean	动态切分辨率开关

- **画质档位说明**

档位	常量值	分辨率	视频码率
STANDARD	TXLiveConstants.VIDEO_QUALITY_STANDARD_DEFINITION	360*640	300~800
HIGH	TXLiveConstants.VIDEO_QUALITY_HIGH_DEFINITION	540*960	600~1500
SUPER	TXLiveConstants.VIDEO_QUALITY_SUPER_DEFINITION	720*1280	600~1800

- **示例代码**：

```
mLivePusher.setVideoQuality(TXLiveConstants.VIDEO_QUALITY_HIGH_DEFINITION, true, true);;
```

14. setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel)

接口详情：boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel)

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
style	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
whiteningLevel	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
ruddyLevel	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**：

```
mLivePusher.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

15. setExposureCompensation(float value)

接口详情：void setExposureCompensation(float value)

调整曝光

- **参数说明**

参数	类型	说明
value	float	曝光比例，表示该手机支持最大曝光调整值的比例，取值范围从-1到1。负数表示调低曝光，正数表示调高曝光， 0 表示不调整曝光

- **示例代码**：

```
mLivePusher.setExposureCompensation(0.5f);
```

16. setFilter()

接口详情：setFilter(Bitmap bmp)

设置指定素材滤镜特效

- **参数说明**

参数	类型	说明
bmp	Bitmap	指定素材，即颜色查找表图片。图片要求png 格式

- **示例代码**：

```
mLivePusher.setFilter(bitmap);
```

17. setChinLevel()

接口详情：setChinLevel(int chinLevel)

设置下巴拉伸或收缩（企业版本有效，普通版本设置此参数无效）

- **参数说明**

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

- **示例代码**：

```
mLivePusher.setChinLevel(0);
```

18. setEyeScaleLevel()

接口详情：setEyeScaleLevel(int eyeScaleLevel)

设置大眼级别（企业版本有效，普通版本设置此参数无效）

- **参数说明**

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

- **示例代码**：

```
mLivePusher.eyeScaleLevel(1);
```

19. setFaceShortLevel()

接口详情：setFaceShortLevel(int faceShortlevel)

设置短脸级别（企业版本有效，普通版本设置此参数无效）

- **参数说明**

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

- **示例代码**：

```
mLivePusher.setFaceShortLevel(1);
```

20. setFaceVLevel()

接口详情：setFaceVLevel(int faceVLevel)

设置V脸级别（企业版本有效，普通版本设置此参数无效）

- **参数说明**

参数	类型	说明
faceVLevel	int	V脸级别取值范围 0 ~ 9； 0 表示关闭，值越大 效果越明显

- **示例代码**：

```
mLivePusher.setFaceVLevel(1);
```

21. setMotionTmpl()

接口详情：setMotionTmpl(java.lang.String motionPath)

设置 P 图动效（企业版本有效，普通版本设置此参数无效）

- **参数说明**

参数	类型	说明
motionPath	String	动效完整路径

- **示例代码**：

```
mLivePusher.setMotionTmpl(motionPath);
```

22. setGreenScreenFile()

接口详情：setGreenScreenFile(java.lang.String file)

设置绿幕文件（企业版本有效，普通版本设置此参数无效）

- **参数说明**

参数	类型	说明
file	String	绿幕文件路径，支持 mp4 文件; null 表示关闭绿幕

- **示例代码**：

```
mLivePusher.setGreenScreenFile(file);
```

23.switchCamera()

接口详情：void switchCamera()

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(TXCloudVideoView) 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。

- **示例代码**：

```
mLivePusher.switchCamera();
```

24.setMute(mute)

接口详情：void setMute(mute)

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
mute	boolean	是否静音

- **示例代码**：

```
mLivePusher.setMute(true);
```

25.setRenderRotation(rotation)

接口详情：void setRenderRotation(int rotation)

设置本地预览图像的顺时针旋转角度。一般运用于横屏推流场景，结合 LivePushConfig.setHomeOrientation() 一起使用

- **参数说明**

参数	类型	说明
rotation	int	值一般为 0 和 90

- **示例代码**：

```
// 竖屏状态，进行竖屏推流，本地渲染相对正方向的角度为 0
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_DOWN);
mLivePusher.setRenderRotation(0);

// 横屏状态，进行横屏推流，本地渲染相对正方向的角度为 90
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
mLivePusher.setRenderRotation(90);
```

26.setMirror(enable)

接口详情：void setMirror(boolean enable)

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固定的，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
enable	boolean	true 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
mLivePusher.setMirror(true);
```

27.playBGM(path)

接口详情：boolean playBGM(String path)

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	String	背景音乐文件位于手机中的绝对路径

- **示例代码**：

```
mLivePusher.playBGM(musicFilePath);
```

28.stopBGM()

接口详情：boolean stopBGM()

停止播放背景音乐。返回结果中，true 表示停止播放成功，false 表示停止播放失败。

- **示例代码**：

```
mLivePusher.stopBGM();
```

29.pauseBGM()

接口详情：boolean pauseBGM()

暂停播放背景音乐。返回结果中，true 表示暂停播放成功，false 表示暂停播放失败。

- **示例代码**：

```
mLivePusher.pauseBGM();
```

30.resumeBGM()

接口详情：boolean resumeBGM()

恢复播放背景音乐。返回结果中，true 表示恢复播放成功，false 表示恢复播放失败。

- **示例代码**：

```
mLivePusher.resumeBGM();
```

31.getMusicDuration(path)

接口详情：int getMusicDuration(java.lang.String path)

获取音乐文件时长，单位ms。

- **参数说明**

参数	类型	说明
path	音乐文件路径	如果 path == null ，获取当前播放歌曲时长。如果 path != null ，则获取path路径歌曲时长。

- **示例代码**：

```
mLivePusher.getMusicDuration(path);
```

32.setMicVolume()

接口详情：boolean setMicVolume(float x)

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
x	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

- **示例代码**：

```
mLivePusher.setMicVolume(2f);
```

33.setBGMVolume()

接口详情：boolean setBGMVolume(float x)

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
x	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

- **示例代码**：

```
mLivePusher.setBGMVolume(2f);
```

34.setVoiceChangerType()

接口详情：void setVoiceChangerType(int voiceChangerType)

设置设置变声类型。

- **参数说明**

参数	类型	说明
voiceChangerType	int	变声类型

- **示例代码**：

```
mLivePusher.setVoiceChangerType(1);
```

35.setReverb()

接口详情：void setReverb(int reverbType)

设置推流端混响效果

- **参数说明**

参数	类型	说明
setReverb	int	混响类型，设置推流端混响效果

- **示例代码**：

```
mLivePusher.setReverb(TXLiveConstants.REVERB_TYPE_1);
```

36.startRecord(videoFilePath)

接口详情： int startRecord(final String videoFilePath)

开始录制视频。该接口用于主播端将推流预览实时保存到本地文件。

注意：该接口需要在startpusher后调用，另外生成的视频文件由应用负责管理，SDK不做清理

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 表示还未开始推流，这次启动录制失败

- **参数说明**

参数	类型	说明
videoFilePath	String	录制的视频文件位于手机中的绝对路径，调用者保证应用拥有该路径权限

- **示例代码**：

```
String videoFile = Environment.getExternalStorageDirectory() + File.separator + "TXUGC/test.mp4";
mLivePusher.startRecord(videoFile);
```

37.stopRecord()

接口详情： void stopRecord()

停止录制视频。录制结果会通过录制回调异步通知出来

- **示例代码**：

```
mLivePusher.stopRecord();
```

38.setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)

接口详情：void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)

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
listener	TXRecordCommon.ITXVideoRecordListener	视频录制回调

- **示例代码**：

```
mLivePusher.setVideoRecordListener(new TXRecordCommon.ITXVideoRecordListener(){
@Override
public void onRecordEvent(int event, Bundle param) {
}

@Override
public void onRecordProgress(long milliSecond) {
Log.d(TAG, "record progress:" + milliSecond);
}

@Override
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode == TXRecordCommon.RECORD_RESULT_OK) {
String videoFile = result.videoPath;
String videoCoverPic = result.coverPath;
} else {
Log.d(TAG, "record error:" + result.retCode + ", error msg:" + result.descMsg);
}
}
});
```

39.snapshot(ITXSnapshotListener)

接口详情： void snapshot(final ITXSnapshotListener listener)

截取视频图像。该接口用于主播端实时截取一帧视频。

注意：截图结果会通过异步回调通知，回调线程是主线程

- **参数说明**

参数	类型	说明
listener	ITXSnapshotListener	视频画面回调接口

参数	类型	说明

- **示例代码**：

```
mLivePusher.snapshot(new TXLivePusher.ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
//bmp 一帧视频画面
}
});
```

40.setAudioProcessListener(listener)

接口详情：void setAudioProcessListener(TXLivePusher.AudioCustomProcessListener listener)

该接口设置自定义音频处理回调。数据回调时机是在音频数据送到编码器编码前。

- **示例代码**：

```
mLivePusher.setAudioProcessListener(new TXLivePusher.AudioCustomProcessListener() {
@Override
public void onRecordPcmData(byte[] data, long ts, int sampleRate, int channels, int bits) {
// data - pcm 数据
// ts - pcm 对应时间戳
// sampleRate - 音频采样率
// channels - 音频通道
// bits - 音频 bits
}
});
```

41.sendCustomVideoData(buffer, bufferType, w, h)

接口详情： int sendCustomVideoData(byte[] buffer, int bufferType, int w, int h)

该接口是向 SDK 塞入您自定义采集和处理后的视频数据（美颜、滤镜等），目前支持 i420 格式。该接口适用场景是只想使用我们 SDK 来 来编码和推流。调用该接口前提是，不再调用 TXLivePusher 的 startCameraPreview 接口。

返回结果的说明：

结果	说明
>0	发送成功,但帧率过高,超过了TXLivePushConfig中设置的帧率,帧率过高会导致视频编码器输出的码率超过TXLivePushConfig中设置的码率,返回值表示当前YUV视频帧提前的毫秒数
0	发送成功

结果	说明
-1	视频分辨率非法
-2	YUV数据长度与设置的视频分辨率所要求的长度不一致
-3	视频格式非法
-4	视频图像长宽不符合要求,画面比要求的小了
-1000	SDK内部错误

- **参数说明**

参数	类型	说明
buffer	byte[]	视频数据
bufferType	int	视频格式.目前只支持 TXLivePusher.YUV_420P和TXLivePusher.RGB_RGBA
w	int	视频图像的宽度
h	int	视频图像的高度

- **示例代码**：

```
//以下是简单的实例，获取摄像机预览回调的视频数据并推流
@Override
public void onPreviewFrame(byte[] data, Camera camera) {
// 假设摄像机获取的视频格式是 NV21, 预览画面大小为 1280X720
if (!isPush) {
} else {
// 开始自定义推流
// 需要将视频格式转码为 I420
byte[] buffer = new byte[data.length];
buffer = nv21ToI420(data, mPreviewWidth, mPreviewHeight);

int customModeType = 0;
customModeType |= TXLiveConstants.CUSTOM_MODE_VIDEO_CAPTURE;
// 只能分辨率的宽和高小于或者等于预览画面的宽和高的分辨率
// 还能选择 480x640 等，但不能选择 540x960。因指定分辨率的高(960) > 预览画面的高(720)，编码器无法裁剪画面。
mLivePushConfig.setVideoResolution(TXLiveConstants.VIDEO_RESOLUTION_TYPE_1280_720);
mLivePushConfig.setAutoAdjustBitrate(false);
```

```
mLivePushConfig.setVideoBitrate(1200);
mLivePushConfig.setVideoEncodeGop(3);
mLivePushConfig.setVideoFPS(15);
mLivePushConfig.setCustomModeType(customModeType);
mLivePusher.setConfig(mLivePushConfig);

int result= mLivePusher.sendCustomVideoData(buffer, TXLivePusher.YUV_420P, mPreviewWidth, mPr
eviewHeight);
}
}

/**
* nv21转I420
* @param data
* @param width
* @param height
* @return
*/
public static byte[] nv21ToI420(byte[] data, int width, int height) {
byte[] ret = new byte[data.length];
int total = width * height;

ByteBuffer bufferY = ByteBuffer.wrap(ret, 0, total);
ByteBuffer bufferU = ByteBuffer.wrap(ret, total, total / 4);
ByteBuffer bufferV = ByteBuffer.wrap(ret, total + total / 4, total / 4);

bufferY.put(data, 0, total);
for (int i=total; i<data.length; i+=2) {
bufferV.put(data[i]);
bufferU.put(data[i+1]);
}
return ret;
}
```

42.sendCustomVideoTexture(buffer, bufferType, w, h)

接口详情： int sendCustomVideoTexture(int textureID, int w, int h)

发送客户自定义的视频纹理。注意，1)该接口需要在OpenGL线程调用 2)必须使用硬件加速。

返回结果的说明：

结果	说明
0	发送成功

结果	说明
-1	视频分辨率非法
-3	视频格式非法
-4	视频图像长宽不符合要求,画面比要求的小了
-1000	SDK内部错误

- **参数说明**

参数	类型	说明
textureID	int	视频纹理ID
w	int	视频图像的宽度
h	int	视频图像的高度

- **示例代码**：

```
```

43. sendCustomPCMData(pcmBuffer)

接口详情：void sendCustomPCMData(byte[] pcmBuffer)

该接口是向 SDK 塞入您自定义采集和处理后的音频数据，请使用单声道或双声道、16位宽、48000Hz 的 PCM 声音数据。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。该接口一般结合 sendCustomVideoData(buffer, bufferType, w, h) 一起使用

- **参数说明**

参数	类型	说明
pcmBuffer	byte[]	pcm 音频数据

- **示例代码**：

```
//以下是简单的实例，获取麦克风采集音频数据。
AudioRecord mAudioRecord = null;
int mMinBufferSize = 0; //最小缓冲区大小

private static final int DEFAULT_SOURCE = MediaRecorder.AudioSource.MIC;
// 设置采样率为 48000
private static final int DEFAULT_SAMPLE_RATE = 48000;
// 支持单声道(CHANNEL_IN_MONO) 和 双声道(CHANNEL_IN_STEREO)
private static final int DEFAULT_CHANNEL_CONFIG = AudioFormat.CHANNEL_IN_STEREO;
// 量化位数
private static final int DEFAULT_AUDIO_FORMAT = AudioFormat.ENCODING_PCM_16BIT;

private boolean mIsCaptureStarted = false;
private volatile boolean mIsLoopExit = true;

private Thread mCaptureThread;
private OnAudioFrameCapturedListener mAudioFrameCapturedListener;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

// 启动音频采集
startCapture();
}

public interface OnAudioFrameCapturedListener {
public void onAudioFrameCaptured(byte[] audioData);
}

public boolean isCaptureStarted() {
return mIsCaptureStarted;
}

public void setOnAudioFrameCapturedListener(OnAudioFrameCapturedListener listener) {
mAudioFrameCapturedListener = listener;
}

public boolean startCapture() {
return startCapture(DEFAULT_SOURCE, DEFAULT_SAMPLE_RATE, DEFAULT_CHANNEL_CONFIG,
DEFAULT_AUDIO_FORMAT);
}

public boolean startCapture(int audioSource, int sampleRateInHz, int channelConfig, int audioForm
at) {
```

```
if (mIsCaptureStarted) {
Log.e(TAG, "audio Capture already started !");
return false;
}

// SDK 要求双声道要 4096, 单声道 2048
mMinBufferSize = 4096;
if (mMinBufferSize == AudioRecord.ERROR_BAD_VALUE) {
Log.e(TAG, "Invalid AudioRecord parameter !");
return false;
}
Log.d(TAG , "getMinBufferSize = "+mMinBufferSize+" bytes !");

mAudioRecord = new AudioRecord(audioSource,sampleRateInHz,channelConfig,audioFormat,mMin
BufferSize);
if (mAudioRecord.getState() == AudioRecord.STATE_UNINITIALIZED) {
Log.e(TAG, "AudioRecord initialize fail !");
return false;
}

mAudioRecord.startRecording();

mIsLoopExit = false;
mCaptureThread = new Thread(new AudioCaptureRunnable());
mCaptureThread.start();

mIsCaptureStarted = true;
Log.d(TAG, "Start audio capture success !");

return true;
}

public void stopCapture() {
if (!mIsCaptureStarted) {
return;
}

mIsLoopExit = true;
try {
mCaptureThread.interrupt();
mCaptureThread.join(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
```

```
if (mAudioRecord.getRecordingState() == AudioRecord.RECORDSTATE_RECORDING) {
mAudioRecord.stop();
}

mAudioRecord.release();

mIsCaptureStarted = false;
mAudioFrameCapturedListener = null;

Log.d(TAG, "Stop audio capture success !");
}

private class AudioCaptureRunnable implements Runnable {
@Override
public void run() {
while (!mIsLoopExit) {

byte[] buffer = new byte[mMinBufferSize];

int ret = mAudioRecord.read(buffer, 0, mMinBufferSize);
if (ret == AudioRecord.ERROR_INVALID_OPERATION) {
Log.e(TAG , "AudioRecord Error ERROR_INVALID_OPERATION");
} else if (ret == AudioRecord.ERROR_BAD_VALUE) {
Log.e(TAG , "AudioRecord Error ERROR_BAD_VALUE");
} else {
if (mAudioFrameCapturedListener != null) {
mAudioFrameCapturedListener.onAudioFrameCaptured(buffer);
}
if (isPush) {
mLivePusher.sendCustomPCMData(buffer);
}
}
SystemClock.sleep(10);
}
}
}
```

44. sendMessageEx(byte[] msg)

接口详情：void sendMessageEx(byte[] msg)

该接口用于向音视频流中塞入自定义的音视频数据，数据被伪装在 SEI 解码器信息中，几乎所有的播放器都不会主动解析 SEI 信息，所以这种在音视频流塞“私货”的方案是非常安全的，但是需要TXLivePlayer才能解读这些信息，具体方法请参考DOC

- **参数说明**

参数	类型	说明
msg	byte[]	在音视频流中塞入自定义数据

TXLivePushConfig接口详情

1. enableAEC(enable)

接口详情：void enableAEC(boolean enable)

开启回声消除。连麦时必须开启，非连麦正常推流时不要开启。默认关闭

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启，false 表示关闭，默认值为 false。

- **示例代码**：

```
mLivePushConfig.enableAEC(true);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

2. enableNearestIP(enable)

接口详情：void enableNearestIP(boolean enable)

开启就近选路。我们 SDK 不绑定腾讯云，如果要推流到非腾讯云地址或者海外推流，请在推流前设置 TXLivePushConfig 中的 enableNearestIP 设置为 false。但如果您要推流的地址为腾讯云地址，可以不用设置。

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启，false 表示关闭，默认值为 true。

- **示例代码**：

```
mLivePushConfig.enableNearestIP(true);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

3. enablePureAudioPush(enable)

接口详情：void enablePureAudioPush(boolean enable)

开启纯音频推流。该接口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。

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启，false 表示关闭。默认值为 false。

- **示例代码**：

```
mLivePushConfig.enablePureAudioPush(true);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

4. enableScreenCaptureAutoRotate(enable)

接口详情：void enableScreenCaptureAutoRotate(boolean enable)

设置录屏是否自适应旋转。该接口只对录屏生效

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启，视频内容为根据屏幕旋转后最大化显示；false 表示关闭，视频内容为屏幕内容缩放居中显示。默认值为 true。

- **示例代码**：

```
mLivePushConfig.enableScreenCaptureAutoRotate(true);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

5. setConnectRetryCount(count)

接口详情：void setConnectRetryCount(int count)

设置推流端重连次数。当SDK与服务器异常断开连接时,SDK会尝试与服务器重连，通过此函数设置SDK重连次数。一般结合 setConnectRetryInterval 一起使用

- **参数说明**

参数	类型	说明
count	int	SDK重连次数，最小值为 1， 最大值为 10，默认值为 3

- **示例代码**：

```
mLivePushConfig.setConnectRetryCount(5);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

6. setConnectRetryInterval(interval)

接口详情：void setConnectRetryInterval(int interval)

设置推流端重连间隔。当SDK与服务器异常断开连接时,SDK会尝试与服务器重连。通过此函数来设置两次重连间隔时间。一般结合 setConnectRetryCount 一起使用

- **参数说明**

参数	类型	说明
interval	int	SDK重连间隔，单位秒。最小值为 3，最大值为 30， 默认值为 3

- **示例代码**：

```
mLivePushConfig.setConnectRetryInterval(10);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

7. setEyeScaleLevel(level)

接口详情：void setEyeScaleLevel(int level)

设置大眼效果。商用企业版 调用该接口才能生效。

- **参数说明**

参数	类型	说明

参数	类型	说明
level	int	大眼等级取值为0-9。取值为 0 表示关闭大眼效果。默认值:0

- **示例代码**：

```
mLivePushConfig.setEyeScaleLevel(1);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

8. setFaceSlimLevel(level)

接口详情：void setFaceSlimLevel(int level)

设置瘦脸效果。商用企业版 调用该接口才能生效。

- **参数说明**

参数	类型	说明
level	int	瘦脸等级取值为0-9。取值为0时代表关闭瘦脸效果。默认值：0

- **示例代码**：

```
mLivePushConfig.setFaceSlimLevel(1);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

9. setFrontCamera(front)

接口详情：void setFrontCamera(boolean front)

设置是否使用前置摄像头。默认使用前置摄像头

- **参数说明**

参数	类型	说明
front	boolean	true 表示使用前置摄像头， false表示使用后置摄像头，默认值:false

- **示例代码**：

```
mLivePushConfig.setFrontCamera(true);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

10. setHomeOrientation(homeOrientation)

接口详情：void setHomeOrientation(int homeOrientation)

设置采集的视频的旋转角度。该接口主要运用在横屏推流场景，一般结合 TXLivePusher.setRenderRotation(rotation) 一起使用。

- **参数说明**

参数	类型	说明
homeOrientation	int	采集的视频的旋转角度，具体值参考 TXLiveConstants 中 **视频旋转角度** 的定义

- **示例代码**：

```
// 竖屏状态, 手机 Home 键在正下方。 旋转 0 度
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_DOWN);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
// 竖屏状态，本地渲染相对正方向的角度为0。
mLivePusher.setRenderRotation(0);


// 横屏状态，手机 Home 键在右手方。 旋转 270 度
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
// 横屏状态，本地渲染相对正方向的角度为90。
mLivePusher.setRenderRotation(90);
```

11. setPauseFlag(flag)

接口详情：void setPauseFlag(int flag)

设置推流暂停时，设置是否停止视频采集或者停止音频采集。一般运用在后台推流场景。建议结合 setPauseImg() 一起使用

- **参数说明**

参数	类型	说明

参数	类型	说明
flag	int	暂停推流时，按照flag选项推流，具体值参考 TXLiveConstants 中 **后台推流选项** 的定义

- **示例代码**：

```
mLivePushConfig.setPauseFlag(TXLiveConstants.PAUSE_FLAG_PAUSE_VIDEO | TXLiveConstants.PAUSE_FLAG_PAUSE_AUDIO);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

12. setPauseImg(img)

接口详情：void setPauseImg(Bitmap img)

设置后台暂停时，推送的垫片。一般运用在后台推流场景。建议结合 setPauseFlag() 一起使用

- **参数说明**

参数	类型	说明
img	Bitmap	后台播放的暂停图片，图片最大尺寸不能超过1920*1920

- **示例代码**：

```
Bitmap bitmap = decodeResource(getResources(),R.drawable.pause_publish);
mLivePushConfig.setPauseImg(bitmap);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

13.setPauseImg(time, fps)

接口详情：void setPauseImg(int time, int fps)

设置推流暂停时，后台垫片的持续时间及帧率 。一般运用在后台推流场景。建议结合 setPauseFlag() 一起使用

- **参数说明**

参数	类型	说明
time	int	后台垫片的最长持续时间，单位是秒，默认值是300
fps	int	后台垫片的帧率，最小值为3，最大值为8，默认是5

- **示例代码**：

```
mLivePushConfig.setPauseImg(300,5);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

14.setVideoEncodeGop(gop)

接口详情：void setVideoEncodeGop(int gop)

设置视频编码GOP。

注意：gop过大会增加延时，无特殊要求建议使用TXLivePusher.setVideoQuality，由SDK内部根据档位调整到合适的值

- **参数说明**

参数	类型	说明
gop	int	视频编码GOP，单位 秒，默认值为3

- **示例代码**：

```
mLivePushConfig.setVideoEncodeGop(1);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

15.setVideoResolution(resolution)

接口详情：void setVideoResolution(int resolution)

设置视频编码分辨率，可以设置的值参考**推流视频分辨率。**

注意：无特殊要求建议使用TXLivePusher.setVideoQuality，由SDK内部根据档位调整到合适的值

- **参数说明**

参数	类型	说明
resolution	int	视频编码分辨率，默认值VIDEO_RESOLUTION_TYPE_540_960

- **示例代码**：

```
mLivePushConfig.setVideoResolution(TXLiveConstants.VIDEO_RESOLUTION_TYPE_540_960);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

16.setWatermark(watermark, x, y,)

接口详情：void setWatermark(Bitmap watermark, int x, int y)

设置水印图片。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。如果您需要对水印图片的位置做机型适配，建议使用 setWatermark(watermark, x, y, width) 接口。

- **参数说明**

参数	类型	说明
watermark	Bitmap	水印图片
x	int	水印位置的 X 轴坐标
y	int	水印位置的 Y 轴坐标

- **示例代码**：

```
//设置视频水印
mLivePushConfig.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 10, 10);
//后面两个参数分别是水印位置的 X 轴坐标和 Y 轴坐标
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

17.setWatermark(watermark, x, y, width)

接口详情：void setWatermark(Bitmap watermark, float x, float y, float width)

设置水印图片。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。
如果您需要对水印图片的位置做机型适配，建议使用该接口。

- **参数说明**

参数	类型	说明
watermark	Bitmap	水印图片
x	int	归一化水印位置的X轴坐标，取值[0,1]

参数	类型	说明
y	int	归一化水印位置的Y轴坐标，取值[0,1]
y	int	归一化水印宽度，取值[0,1]

- **示例代码**：

```
//设置视频水印
//参数分别是水印图片的 Bitmap、水印位置的 X 轴坐标，水印位置的 Y 轴坐标，水印宽度。后面三个参数取值范围是[0, 1]
//后面两个参数分别是水印位置的X轴坐标和 Y 轴坐标
mLivePushConfig.setWatermark(mBitmap, 0.02f, 0.05f, 0.2f);
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config
```

TXLivePlayer接口详情

1.setConfig(TXLivePlayConfig config)

接口详情：void setConfig(TXLivePlayConfig config)

设置获取播放器器配置信息。

注意：请在startPlay前设置，不支持播放过程中设置config。

- **参数说明**

参数	类型	说明
config	TXLivePlayConfig	播放器器配置信息

- **示例代码**：

```
TXLivePlayConfig mPlayConfig = new TXLivePlayConfig();
mPlayConfig.setConnectRetryCount(3);
......
mLivePlayer.setConfig(mPlayConfig);
......
mLivePlayer.startPlay(...);
```

2. setPlayerView(TXCloudVideoView view)

接口详情：void setPlayerView(TXCloudVideoView view)

把渲染 view 绑定到 TXLivePlayer。支持在播放过程中， 不停止播放情况下，将同一个 player绑定到不同的 TXCloudVideoView。

- **参数说明**

参数	类型	说明
view	TXCloudVideoView	播放画面渲染的 View

- **示例代码**：

```
// 初始化
TXCloudVideoView playerView = (TXCloudVideoView) view.findViewById(R.id.video_view);
//关联 player 对象与界面 view
mLivePlayer.setPlayerView(playerView);
```

3. setPlayListener(ITXLivePlayListener listener)

接口详情：void setPlayListener(ITXLivePlayListener listener)

设置TXLivePlayer 的回调，用于接收播放事件及播放器状态。

```
// 初始化
mLivePlayer.setPlayListener(new ITXLivePlayListener() {
@Override
public void onPlayEvent(int i, Bundle bundle) {
// 收到播放器事件，具体事件定义请参看TXLiveConstants.PLAY_EVT_xxx
}

@Override
public void onNetStatus(Bundle bundle) {
// 收到播放器当前状态，2s一次刷新，具体状态请参看TXLiveConstants.NET_STATUS_xxx
}
});
```

4.startPlay(url, type)

接口详情：int startPlay(String url，int type)

启动播放。返回结果中，0 表示启动播放成功；-1 表示启动播放失败，因 playUrl 为空； -2 表示启动播放失败，因 playUrl 非法； -3 表示启动播放失败，因 playType 非法

- **参数说明**

参数	类型	说明
url	String	一个合法的拉流地址，视频播放 URL（URL 以 “rtmp://” 打头 ，腾讯云拉流 URL 的获取方法见 DOC ），我们建议使用 FLV 格式
type	int	播放类型，参考 TXLivePlayer 中定义的播放类型枚举值

- **示例代码** :

```
int result = mLivePlayer.startPlay(playUrl,TXLivePlayer.PLAY_TYPE_LIVE_FLV);
```

5.stopPlay(isNeedClearLastImg)

接口详情：int stopPlay(boolean isNeedClearLastImg)

停止播放。返回结果中，0 表示停止播放成功，非0 表示停止播放失败。

- **参数说明**

参数	类型	说明
isNeedClearLastImg	int	是否需要清除最后一帧画面。true 表示清除最后一帧画面，正常停止播放时，推荐清除。 false 表示保留最后一帧画面，异常停止播放(如网络异常,导致播放被迫停止)，而SDK使用者希望重连服务器，继续播放时，推荐保留

- **示例代码** :

```
mLivePlayer.stopPlay(true);
```

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。

- **示例代码** :

```
mLivePlayer.pause();
```

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。

- **示例代码**：

```
mLivePlayer.resume();
```

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。不支持非腾讯云地址。

- **示例代码**：

```
mLivePlayer.prepareLiveSeek();
```

9.setAudioRoute()

接口详情：void setAudioRoute(int audioRoute)

设置声音播放模式(切换扬声器，听筒)

- **示例代码**：

```
// 设置扬声器
mLivePlayer.setAudioRoute(TXLiveConstants.AUDIO_ROUTE_SPEAKER);
// 设置听筒
mLivePlayer.setAudioRoute(TXLiveConstants.AUDIO_ROUTE_RECEIVER);
```

10.switchStream(playUrl)

接口详情：void switchStream(java.lang.String playUrl)

flv直播无缝切换

- **参数说明**

参数	类型	说明
playUrl	String	播放的流地址. playUrl必须是当前播放直播流的不同清晰度，切换到无关流地址可能会失败

- **示例代码**：

```
mLivePlayer.switchStream(playUrl);
```

11.enableHardwareDecode(enable)

接口详情：void enableHardwareDecode(enable)

启用或禁用视频硬解码，SDK默认是视频软解码。

注意：硬件解码只在 Android 系统 4.2 以上（API 级别 16）的手机支持。硬件解码失败后，SDK内部会自动切换成软解。该接口设置只在startPlay前设置生效。

- **示例代码**：

```
mLivePlayer.enableHardwareDecode(true);
.....
mLivePlayer.startPlay(...)
```

12.isPlaying()

接口详情：boolean isPlaying()

是否正在播放

返回true表示正在播放，false表示尚未播放。

- **示例代码**：

```
boolean isPlaying = mLivePlayer.isPlaying();
```

13.setRenderMode(mode)

接口详情： void setRenderMode(int mode)

设置图像的渲染（填充）模式。

注意：该接口目前只支持如下两种模式。

TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN（视频画面全屏铺满）：将图像等比例铺满整个屏幕,多余部分裁剪掉，此模式下画面不留黑边，但视频长宽比例和view不对的情况下会显示不全。

TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION（视频画面自适应屏幕）：将图像等比例缩放,缩放后的宽和高都不会超过显示区域,居中显示,可能会留有黑边

- **参数说明**

参数	类型	说明
mode	int	TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN或者 TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION。默认是FULL_FILL_SCREEN

- **示例代码**：

```
// 设置填充模式为自适应
mLivePlayer.setRenderMode(TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION);
// 设置填充模式为全屏
mLivePlayer.setRenderMode(TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN);
```

14.setRenderRotation(rotation)

接口详情：void setRenderRotation(int rotation)

设置图像的顺时针旋转角度，即设置图像呈现效果为竖屏或者横屏。

可设置的值为TXLiveConstants.RENDER_ROTATION_PORTRAIT（竖屏），
TXLiveConstants.RENDER_ROTATION_LANDSCAPE（横屏）。

- **参数说明**

参数	类型	说明
rotation	int	竖屏或者横屏渲染，默认是竖屏

- **示例代码**：

```
// 设置画面渲染方向为横屏
mLivePlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_LANDSCAPE);
// 设置画面渲染方向为竖屏
mLivePlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
```

15.startRecord()

接口详情：int startRecord(int recordType)

启动截流录制。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，可以通过点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。

注意：该接口正在Android 4.4，API 18以上接口使用，并且开始播放后才能调用。

- **参数说明**

参数	类型	说明
recordType	int	目前该参数废弃，只支持纯视频录制

- **示例代码**：

```
int recordType = TXRecordCommon.RECORD_TYPE_STREAM_SOURCE;
mLivePlayer.startRecord(recordType);
```

16.stopRecord()()

接口详情：int stopRecord()

停止截流录制。返回结果中，0 表示成功，非0表示失败。录制结果会通过录制回调通知出来

- **示例代码**：

```
mLivePlayer.stopRecord();
```

17.setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)

接口详情：void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)

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
listener	TXRecordCommon.ITXVideoRecordListener	视频录制回调

- **示例代码**：

```
mLivePusher.setVideoRecordListener(new TXRecordCommon.ITXVideoRecordListener(){
@Override
public void onRecordEvent(int event, Bundle param) {
}

@Override
public void onRecordProgress(long milliSecond) {
Log.d(TAG, "record progress:" + milliSecond);
}

@Override
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode == TXRecordCommon.RECORD_RESULT_OK) {
String videoFile = result.videoPath;
String videoCoverPic = result.coverPath;
} else {
Log.d(TAG, "record error:" + result.retCode + ", error msg:" + result.descMsg);
}
}
});
```

18.setMute(mute)

接口详情：void setMute(boolean mute)

设置是否静音播放。该接口在播放前和播放过程中设置都是生效的。

- **参数说明**

参数	类型	说明
mute	boolean	是否静音，默认非静音

- **示例代码**：

```
mLivePlayer.setMute(true);
```

19. snapshot(TXLivePlayer.ITXSnapshotListener listener)

接口详情：void snapshot(TXLivePlayer.ITXSnapshotListener listener)

截取视频截图。该接口截取截取当前直播流的中一帧视频画面。如果您需要截取当前的整个 UI 界面，请调用系统 API 来实现。截图结果回调接口会异步在主线程回调。

- **示例代码**：

```
mLivePlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
//获取到截图 bitmap
}
}
});
```

20. setSurface(Surface surface)

接口详情：void setSurface(Surface surface)

设置渲染surface。可以用来接管SDK的渲染。
注意：这个接口只适用于硬件解码，并且不能设置setPlayViiew。

- **示例代码**：

```
TextureView videoView = (TextureView) findViewById(R.id.video_view);
videoView.setSurfaceTextureListener(new TextureView.SurfaceTextureListener() {
@Override
public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int height) {
mLivePlayer.setSurface(new Surface(surface));
}

@Override
public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width, int height) {

}

@Override
public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
if (mLivePlayer != null) {
mLivePlayer.setSurface(null);
}
return false;
}

@Override
```

```
public void onSurfaceTextureUpdated(SurfaceTexture surface) {

}
});
```

21. addVideoRawData(byte[] yuvBuffer)

接口详情：void addVideoRawData(byte[] yuvBuffer)

设置播放yuv数据接收buffer。和setVideoRawDataListener接口配合，可以用来接管SDK的渲染。
注意：这个接口只适用于软件解码，并且不能设置setPlayViiew。buffer的大小请用视频实际分辨率计算。

22. setVideoRawDataListener(ITXVideoRawDataListener listener)

接口详情：void setVideoRawDataListener(ITXVideoRawDataListener listener)

设置视频yuv数据回调。和addVideoRawData接口配合，可以用来接管SDK的渲染。
注意：这个接口只适用于软件解码，并且不能设置setPlayViiew。

示例代码：

```
public void onPlayEvent(int event, Bundle param) {
.....
if (event == TXLiveConstants.PLAY_EVT_CHANGE_RESOLUTION) {
videoWidth = param.getInt(TXLiveConstants.EVT_PARAM1,0);
videoHeight = param.getInt(TXLiveConstants.EVT_PARAM2,0);
mLivePlayer.addVideoRawData(new byte[videoWidth*videoHeight*3/2]);
}
.....
}


mLivePlayer.setVideoRawDataListener(new TXLivePlayer.ITXVideoRawDataListener() {
@Override
public void onVideoRawDataAvailable(byte[] buf, int width, int height, int timestamp) {
mLivePlayer.addVideoRawData(new byte[videoWidth*videoHeight*3/2]);
}
});
```

23. setAudioRawDataListener(ITXAudioRawDataListener listener)

接口详情：void setAudioRawDataListener(ITXAudioRawDataListener listener)

设置音频pcm数据回调。

示例代码：

```
mLivePlayer.setAudioRawDataListener(new TXLivePlayer.ITXAudioRawDataListener() {
@Override
public void onPcmDataAvailable(byte[] buf, long timestamp) {
//音频pcm数据
}

@Override
public void onAudioInfoChanged(int sampleRate, int channels, int bits) {
//音频采样率，通道数
}
});
```

TXLivePlayConfig接口详情

1. enableAEC(enable)

接口详情：void enableAEC(boolean enable)

开启回声消除。连麦时必须开启，非连麦时不要开启。

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启，false 表示关闭。默认值为 false。

- **示例代码**：

```
mPlayConfig.enableAEC(true);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

2. setConnectRetryCount(count)

接口详情：void setConnectRetryCount(int count)

设置播放器重连次数。当SDK与服务器异常断开连接时,SDK会尝试与服务器重连，通过此函数设置SDK重连次数。一般结合 setConnectRetryInterval 一起使用

- **参数说明**

参数	类型	说明

参数	类型	说明
count	int	SDK重连次数，最小值为 1， 最大值为 10，默认值为 3

- **示例代码**：

```
mPlayConfig.setConnectRetryCount(5);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

3. setConnectRetryInterval(interval)

接口详情：void setConnectRetryInterval(int interval)

设置播放器重连间隔。当SDK与服务器异常断开连接时,SDK会尝试与服务器重连。通过此函数来设置两次重连间隔时间。一般结合 setConnectRetryCount 一起使用

- **参数说明**

参数	类型	说明
interval	int	SDK重连间隔，单位秒。最小值为 3，最大值为 30， 默认值为 3

- **示例代码**：

```
mPlayConfig.setConnectRetryInterval(10);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

4. setEnableMessage(enable)

接口详情：void setEnableMessage(boolean enable)

开启消息通道。

- **参数说明**

参数	类型	说明
enable	boolean	true 表示开启消息通道， false 表示关闭消息通道。默认值是 false。

- **示例代码**：

```
mPlayConfig.setEnableMessage(true);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

5. setAutoAdjustCacheTime(bAuto)

接口详情：void setAutoAdjustCacheTime(boolean bAuto)

设置是否根据网络状况自动调整播放器缓存时间。一般结合 setMinAutoAdjustCacheTime(time 和 setMaxAutoAdjustCacheTime(time) 一起使用。

启用自动调整时,SDK将根据网络状况在一个范围内调整缓存时间，自动调整的范围可以通过修改MaxAutoAdjustCacheTime和修改MinAutoAdjustCacheTime来调整。

关闭自动调整时，SDK将使用固定的缓存时间，固定的缓存时间可以通过修改cacheTime来调整。

- **参数说明**

参数	类型	说明
bAuto	boolean	true 表示启用自动调整，false 表示关闭自动调整。默认值是true。

- **示例代码**：

```
//自动模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(5);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

6. setCacheTime(time)

接口详情：void setCacheTime(float time)

设置播放器缓存时间。播放器固定缓冲时间，需要设置setAutoAdjustCacheTime(false)。

- **参数说明**

参数	类型	说明
time	float	播放器缓存时间，单位秒，默认值为 5，取值需要大于0

- **示例代码**：

```
//流畅模式
mPlayConfig.setAutoAdjustCacheTime(false);
mPlayConfig.setCacheTime(5);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

7. setMaxAutoAdjustCacheTime(time)

接口详情：void setMinAutoAdjustCacheTime(float time)

设置自动调整时播放器最大缓存时间。结合 setAutoAdjustCacheTime(bAuto) 和 setMinAutoAdjustCacheTime(time) 一起使用。

- **参数说明**

参数	类型	说明
time	float	播放器最大缓存时间，单位秒，默认值为 5，取值需要大于0

- **示例代码**：

```
//极速模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(1);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

8. setMinAutoAdjustCacheTime(time)

接口详情：void setMinAutoAdjustCacheTime(float time)

设置自动调整时播放器最小缓存时间。结合 setAutoAdjustCacheTime(bAuto) 和 setMaxAutoAdjustCacheTime(time) 一起使用。

- **参数说明**

参数	类型	说明
time	float	播放器最小缓存时间，单位秒，默认值为 1，取值需要大于0

- **示例代码**：

```
//极速模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(1);
mLivePlayer.setConfig(mPlayConfig); // 重新设置 config
```

枚举类型定义

1. 推流视频分辨率

如果使用摄像头推流一般不使用以下参数。如果使用录屏推流，有横竖屏切换的场景，需要使用以下参数来改变推流的分辨率。

```
VIDEO_RESOLUTION_TYPE_360_640 = 0; // 360 * 640 的分辨率
VIDEO_RESOLUTION_TYPE_540_960 = 1; // 540 * 640 的分辨率
VIDEO_RESOLUTION_TYPE_720_1280 = 2; // 720 * 1280 的分辨率
VIDEO_RESOLUTION_TYPE_640_360 = 3; // 640 * 360 的分辨率
VIDEO_RESOLUTION_TYPE_960_540 = 4; // 960 * 440 的分辨率
VIDEO_RESOLUTION_TYPE_1280_720 = 5; // 1280 * 720 的分辨率
VIDEO_RESOLUTION_TYPE_320_480 = 6; // 320 * 480 的分辨率
VIDEO_RESOLUTION_TYPE_180_320 = 7; // 180 * 320 的分辨率
VIDEO_RESOLUTION_TYPE_270_480 = 8; // 270 * 480 的分辨率
VIDEO_RESOLUTION_TYPE_320_180 = 9; // 320 * 180 的分辨率
VIDEO_RESOLUTION_TYPE_480_270 = 10; // 480 * 270 的分辨率
```

2. 软硬编选项

推流编码的类型。如果你不清楚要何时开启硬件加速, 建议设置为 ENCODE_VIDEO_AUTO。默认是启用软件编码, 但手机 CPU 使用率超过 80% 或者帧率 <= 10, SDK 内部会自动切换为硬件编码

```
ENCODE_VIDEO_SOFTWARE = 0; // 软编
ENCODE_VIDEO_HARDWARE = 1; // 硬编
ENCODE_VIDEO_AUTO = 2; // 自动决定软硬编
```

3. 图像平铺模式

播放器设置画面渲染的填充模式，铺满或者适应

```
RENDER_MODE_FULL_FILL_SCREEN = 0; //视频画面全屏铺满:将图像等比例铺满整个屏幕,多余部分裁剪掉,此模式下画面不留黑边
```

```
RENDER_MODE_ADJUST_RESOLUTION = 1; //视频画面自适应屏幕:将图像等比例缩放,缩放后的宽和高都不会超过显示区域,居中显示,可能会留有黑边
```

4. 图像渲染角度

播放器设置画面渲染的旋转的角度，竖屏或者横屏

```
RENDER_ROTATION_PORTRAIT = 0; // 常规竖屏
RENDER_ROTATION_LANDSCAPE = 270; // 右旋90度，即横屏
```

5. 视频旋转角度

在横屏推流场景中，推流端设置观众端观看画面旋转的角度。

```
VIDEO_ANGLE_HOME_RIGHT = 0; // home在右边
VIDEO_ANGLE_HOME_DOWN = 1; // home在下面
VIDEO_ANGLE_HOME_LEFT = 2; // home在左边
VIDEO_ANGLE_HOME_UP = 3; // home在上面
```

6. 后台推流选项

setPauseFlag 的选项

```
PAUSE_FLAG_PAUSE_VIDEO = 1; // pausePusher时，设置此标志位暂停原采集(camera, 录屏)的画面，采用pauseImg作为推流的画面，不设置此标志位，不会停止原采集(camera, 录屏)，继续推流
PAUSE_FLAG_PAUSE_AUDIO = 2; // pausePusher时，设置此标志位暂停音频采集，推送静音数据，不设置此标志位，不会暂停音频采集，继续推送麦克风采集到的音频
```

7. 播放类型

播放器支持播放类型。在 APP 中播放视频，直播我们建议使用FLV 格式。

```
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
```

ITXLivePushListener事件回调

推流事件接口

接口定义	功能说明
onPushEvent(int event, Bundle param)	推流事件通知接口
onNetStatus(Bundle param)	推流状态通知接口

推流事件列表

```
PUSH_EVT_CONNECT_SUCC = 1001, // 已经连接推流服务器
PUSH_EVT_PUSH_BEGIN = 1002, // 已经与服务器握手完毕,开始推流
PUSH_EVT_OPEN_CAMERA_SUCC = 1003, // 打开摄像头成功
PUSH_EVT_SCREEN_CAPTURE_SUCC = 1004; // 录屏启动成功
PUSH_EVT_CHANGE_RESOLUTION = 1005, // 推流动态调整分辨率
PUSH_EVT_CHANGE_BITRATE = 1006, // 推流动态调整码率
PUSH_EVT_FIRST_FRAME_AVAILABLE = 1007, // 首帧画面采集完成
PUSH_EVT_START_VIDEO_ENCODER = 1008, // 编码器启动

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,且经多次重连抢救无效,可以放弃治疗,更多重试请自行重启推流
PUSH_ERR_SCREEN_CAPTURE_START_FAILED = -1308; // 开始录屏失败,可能是被用户拒绝了
PUSH_ERR_SCREEN_CAPTURE_UNSURPORT = -1309; // 录屏失败,不支持的Android系统版本,需要5.0以上的系统
PUSH_ERR_SCREEN_CAPTURE_DISTURBED = -1310; // 录屏被其他应用打断了，先开MediaProjection的应用会被后开MediaProjection的应用停掉，SDK抛出此error
PUSH_ERR_MIC_RECORD_FAIL = -1311; // Android Mic打开成功，但是连续6次录不到音频数据 (Audio录制线程会退出)
PUSH_ERR_SCREEN_CAPTURE_SWITCH_DISPLAY_FAILED = -1312; // 录屏动态切横竖屏失败

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_VIDEO_ENCODE_FAIL = 1104, // 视频编码失败,非致命错,内部会重启编码器
PUSH_WARNING_BEAUTYSURFACE_VIEW_INIT_FAIL = 1105, // 视频编码码率异常，警告
PUSH_WARNING_VIDEO_ENCODE_BITRATE_OVERFLOW = 1106, // 视频编码码率异常，警告
PUSH_WARNING_DNS_FAIL = 3001, // RTMP -DNS解析失败
PUSH_WARNING_SEVER_CONN_FAIL = 3002, // RTMP服务器连接失败
PUSH_WARNING_SHAKE_FAIL = 3003, // RTMP服务器握手失败
PUSH_WARNING_SERVER_DISCONNECT = 3004, // RTMP服务器主动断开，请检查推流地址的合法性或防
```

盗链有效期
PUSH_WARNING_SERVER_NO_DATA = 3005, // 超过30s没有数据发送，主动断开连接。

推拉流状态信息

SDK 指标监控，主要回调推流或拉流的状态数据。

推流状态	类型	含义说明
NET_STATUS_CPU_USAGE	String	当前进程的CPU使用率和本机总体的CPU使用率
NET_STATUS_VIDEO_WIDTH	int	**当前视频的宽度（单位：像素值）**
NET_STATUS_VIDEO_HEIGHT	int	**当前视频的高度（单位：像素值）**
NET_STATUS_NET_SPEED	int	当前的发送速度（单位：kbps）
NET_STATUS_VIDEO_BITRATE	int	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	int	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
NET_STATUS_VIDEO_FPS	int	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	int	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	int	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	String	连接的推流服务器的IP

ITXLivePlayListener事件回调

播放事件接口

接口定义	功能说明
onPlayEvent(int event, Bundle param)	TXLivePlayer 的播放事件通知
onNetStatus(Bundle param)	TXLivePlayer的播放状态通知

播放事件列表

```
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
PLAY_EVT_RCV_FIRST_I_FRAME = 2003, // 渲染首个视频数据包(IDR)
PLAY_EVT_PLAY_BEGIN = 2004, // 视频播放开始
PLAY_EVT_PLAY_PROGRESS = 2005, // 视频播放进度
PLAY_EVT_PLAY_END = 2006, // 视频播放结束
PLAY_EVT_PLAY_LOADING = 2007, // 视频播放loading
PLAY_EVT_START_VIDEO_DECODER = 2008, // 解码器启动
PLAY_EVT_CHANGE_RESOLUTION = 2009, // 视频分辨率改变

PLAY_ERR_NET_DISCONNECT = -2301, // 网络断连,且经多次重连抢救无效,可以放弃治疗,更多重试请自行重启播放
PLAY_ERR_GET_RTMP_ACC_URL_FAIL = -2302, // 获取加速拉流地址失败

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 (自动重连连续失败超过三次会放弃)
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（用户直观感受）
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_FIRST_IDR_HW_DECODE_FAIL = 2108, // 当前流硬解第一个I帧失败，SDK自动切软解
PLAY_WARNING_DNS_FAIL = 3001, // RTMP -DNS解析失败
PLAY_WARNING_SEVER_CONN_FAIL=: 3002, // RTMP服务器连接失败
PLAY_WARNING_SHAKE_FAIL = 3003, // RTMP服务器握手失败
PLAY_WARNING_SERVER_DISCONNECT = 3004, // RTMP服务器主动断开
```

标准直播
集成(Android Studio)

最近更新时间：2018-08-22 15:16:14

1 SDK 信息

您可以在腾讯云官网更新 小直播 SDK，目前小直播 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：

文件名	说明
LiteAVSDK_Professional_3.0.1185.aar	aar 封装方式的 SDK，适用于 Android Studio 用户
LiteAVSDK_Professional_3.0.1185.zip	jar + so 封装方式的 SDK，适用于 Eclipse 用户，如果您觉得 SDK 全量打包进 apk 会增大安装包体积，可以将 zip 包中的 so 文件上传到服务器，通过使用时再下载 so 文件的方式减少 apk 的体积，具体使用方法见 如何减少 apk 体积？

文件名	说明
Demo	基于 aar 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Android Studio可以快速导入并体验。
API文档	点击文件夹里的 index.html 文件可以查看本 SDK 的所有接口描述

2 系统要求

SDK 支持 在 Android 4.0.3（API 15）及以上系统上运行，但只有（Android 4.3) API 18 以上的系统才能开启硬件编码。

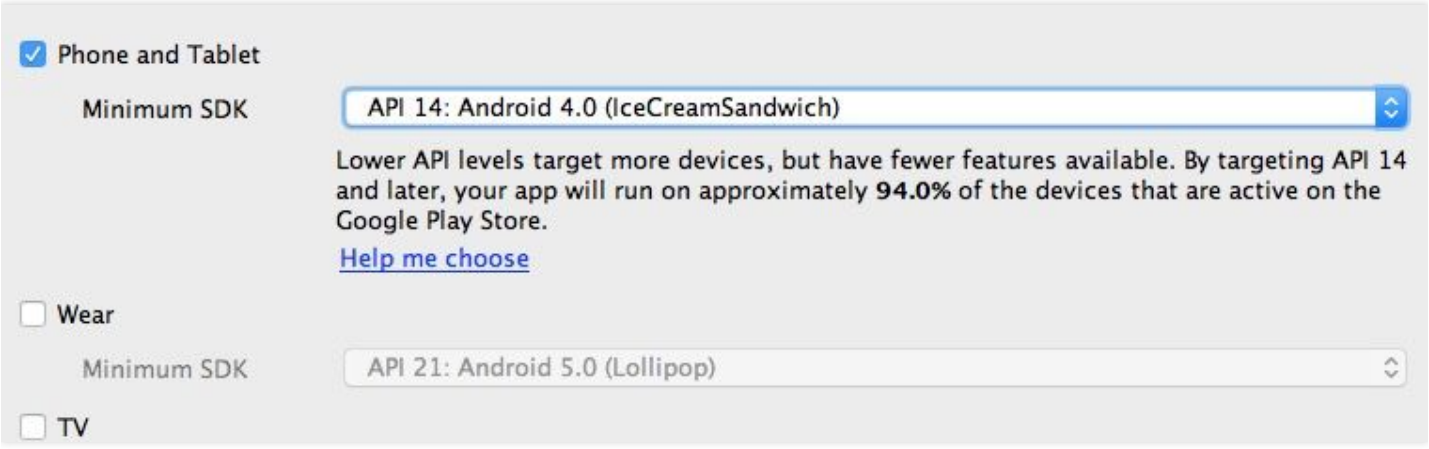
3 开发环境

以下是 SDK 的开发环境，APP 开发环境不需要与 SDK 一致，但要保证兼容：

- Android NDK: android-ndk-r12b
- Android SDK Tools: android-sdk_25.0.2
  - minSdkVersion: 15
  - targetSdkVersion: 21
- Android Studio（推荐您也使用 Android Studio，当然您也可以使用 Eclipse+ADT）

4 集成攻略（aar）

4.1 新建工程



4.2 拷贝文件

将 aar 包放在工程 libs 目录下即可

4.3 工程配置

- 在工程 app 目录下的 build.gradle 中，添加引用 aar 包的代码：

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
// 导入腾讯云直播 SDK aar
compile(name: 'LiteAVSDK_Professional_3.0.1185', ext: 'aar')
}
```

- 在工程目录下的 build.gradle 中，添加 flatDir，指定本地仓库：

```
allprojects {
repositories {
jcenter()
flatDir {
dirs 'libs'
}
}
}
```

- 在工程目录下的 build.gradle 的 defaultConfig 里面，指定 ndk 兼容的架构：

```
defaultConfig {
applicationId "com.tencent.liteav.demo"
minSdkVersion rootProject.ext.minSdkVersion
targetSdkVersion rootProject.ext.targetSdkVersion
versionCode 1
versionName "2.0"

ndk {
abiFilters "armeabi", "armeabi-v7a"
// 如果您使用的是商业版，只能使用 armeabi 架构，即：
// abiFilters "armeabi",
}
}
```

- 最后编译一下工程 Rebuild Project。

5 集成攻略（jar）

5.1 库说明

解压 LiteAVSDK_Professional_3.0.1185.zip 压缩包后得到 libs 目录，里面主要包含 so 文件和 jar 文件，文件清单如下：

so文件	说明
liteavsdk.jar	小直播 SDK android 核心库
libliteavsdk.so	小直播 SDK 核心组件
libsaturn.so	小直播 SDK 核心组件
libtraeimp-rtmp-armeabi.so	连麦功能所使用的声学组件库
libstlport_shared.so	C++ stl 基础库（请不要随意替换，版本不匹配亦产生崩溃）
libijkffmpeg.so	ffmpeg 基础库（ijk 版本），用于点播播放功能，解决一些奇葩视频格式的兼容问题
libijkplayer.so	ijkplayer 开源库，用于点播播放功能，解决一些奇葩视频格式的兼容问题
libijksdl.so	ijkplayer 开源库，用于点播播放功能，解决一些奇葩视频格式的兼容问题

5.2 拷贝文件

如果您的工程之前没有指定过 jni 的加载路径，推荐您将刚才解压的 jar 包和 so 库在 /src/main/jniLibs 目录下，这是 android studio 默认的 jni 加载目录。

5.3 工程配置

在工程 app 目录下的 build.gradle 中，添加引用 jar 包和 so 库的代码。

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
// 导入腾讯云直播 SDK jar
compile fileTree(dir: 'src/main/jniLibs', includes: ['*.jar'])
}
```

6 配置 APP 权限

在 AndroidManifest.xml 中配置 APP 的权限，音视频类 APP 一般需要以下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.Camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

7 验证

在工程中调用 SDK 接口，获取 SDK 版本信息，以验证工程设备是否正确。

7.1 引用 SDK

在 MainActivity.java 中引用 SDK 的 class：

```
import com.tencent.rtmp.TXLiveBase;
```

7.2 调用接口

在 onCreate 中调用 getSDKVersioin 接口获取版本号：

```
String sdkver = TXLiveBase.getSDKVersionStr();
Log.d("liteavsdk", "liteav sdk version is : " + sdkver);
```

7.3 编译运行

如果前面各步骤都操作正确，demo 工程将顺利编译通过，运行之后将在 logcat 中看到如下 log 信息：

```
08-10 19:30:36.547 19577-19577/ D/liteavsdk: liteav sdk version is : 3.0.1185
```

8 减少 APK 体积

整个 SDK 的体积主要来自于 so 文件，这些 so 文件是 SDK 正常运行所依赖的音视频编解码库、图像处理库 以及声学处理组件，如果小直播 SDK 的功能不是 App 的核心功能，您可以考虑采用在线加载的方式减少最终 apk 安装包的大小。

8.1 上传 SO 文件

将 SDK 压缩包中的 so 文件上传到 CDN，并记录下载地址，比如 `http://xxx.com/so_files.zip` 。

8.2 启动准备

在用户启动 SDK 相关功能前，比如开始播放视频之前，先用 loading 动画提示用户“正在加载相关的功能模块”。

8.3 下载 SO 文件

在用户等待过程中，APP 就可以到 `http://xxx.com/so_files.zip` 下载 so 文件，并存入应用目录下（比如应用根目录下的 files 文件夹），为了确保这个过程不受运营商 DNS 拦截的影响，请在文件下载完成后校验 so 文件的完整性。

8.4 加载 SO 文件

等待所有 so 文件就位以后，调用 TXLiveBase 的 setLibraryPath 将下载的目标 path 设置给 SDK， 然后再调用 SDK 的相关功能。之后，SDK 会到这些路径下加载需要的 so 文件并启动相关功能。

9 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 Android Studio 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 sdcard 上的 **log / tencent / liteav** 文件夹下。
  如果您需要我们的技术支持，建议将次开关打开，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(true);
TXLiveBase.setLogLevel(TXLiveConstants.LOG_LEVEL_DEBUG);
```

常见问题排查

如果您将 SDK 导入到您的工程，编译运行出现类似以下错误：

```
Caused by: android.view.InflateException:
Binary XML file #14:Error inflating class com.tencent.rtmp.ui.TXCloudVideoView
```

可以按照以下流程来排查问题：

- 确认是否已经将 SDK 中的 jar 包和 so 库放在 jniLibs 目录下。
- 如果您使用 aar 集成方式的完整版本，在工程目录下的 build.gradle 的 defaultConfig 里面确认下是否将 x64 架构的 so 库过滤掉。因为完整版本中连麦功能所使用的声学组件库暂时不支持 x64 架构的手机。

```
defaultConfig {
applicationId "com.tencent.liteav.demo"
minSdkVersion rootProject.ext.minSdkVersion
targetSdkVersion rootProject.ext.targetSdkVersion
versionCode 1
versionName "2.0"

ndk {
abiFilters "armeabi", "armeabi-v7a"
}
}
```

- 检查下混淆规则，确认已将 SDK 的相关包名加入了不混淆名单。

```
-keep class com.tencent.** { *; }
```

推流(LivePusher)

最近更新时间：2018-08-10 16:21:46

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 false。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

true，否则推流质量可能会因为运营商 DNS 不准确而受到影响。

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。

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、个人直播以及活动直播等场景。

step 1: 添加界面元素

为了能够展示摄像头预览的影像，您需要在您的布局 xml 文件里加入如下一段代码，他会在您的UI上安插一个 TXCloudVideoView 控件，这是我们用来显示摄像头影像的专用控件：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/video_view"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_centerInParent="true"
android:visibility="gone"/>
```

step 2: 创建 Pusher 对象

创建一个 **TXLivePusher** 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 new 出来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，简单地塞给 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。

```
TXLivePusher mLivePusher = new TXLivePusher(getActivity());
mLivePushConfig = new TXLivePushConfig();
mLivePusher.setConfig(mLivePushConfig);
```

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：

```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(rtmpUrl);

TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mLivePusher.startCameraPreview(mCaptureView);
```

- **startPusher** 的作用是告诉 SDK 音视频流要推到哪个推流URL上去。
- **startCameraPreview** 则是将界面元素和 Pusher 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。

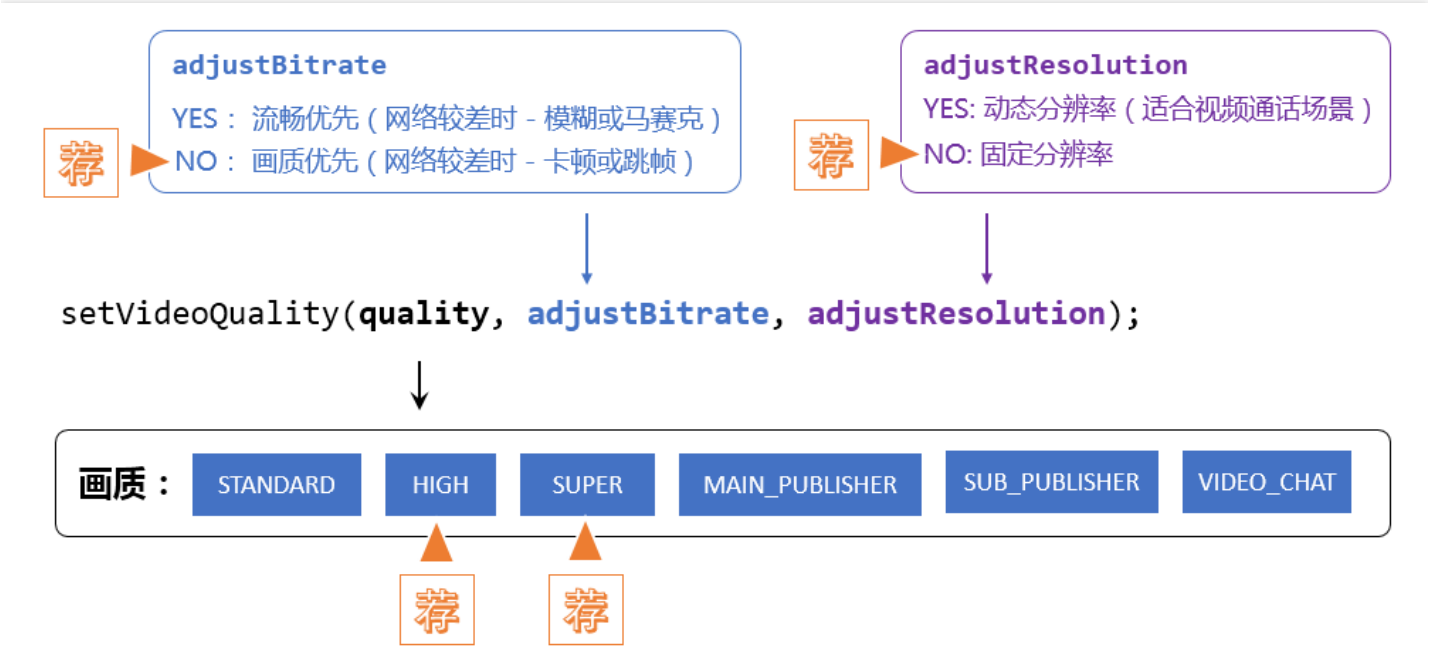
```
// 只有在推流启动前设置启动纯音频推流才会生效，推流过程中设置不会生效。
mLivePushConfig.enablePureAudioPush(true); // true 为启动纯音频推流，而默认值是 false；
mLivePusher.setConfig(mLivePushConfig); // 重新设置 config

String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(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。请提工单联系我们。

step 4: 设定清晰度

使用 setVideoQuality 接口的可以设定推流的画面清晰度：



- **推荐参数设置**

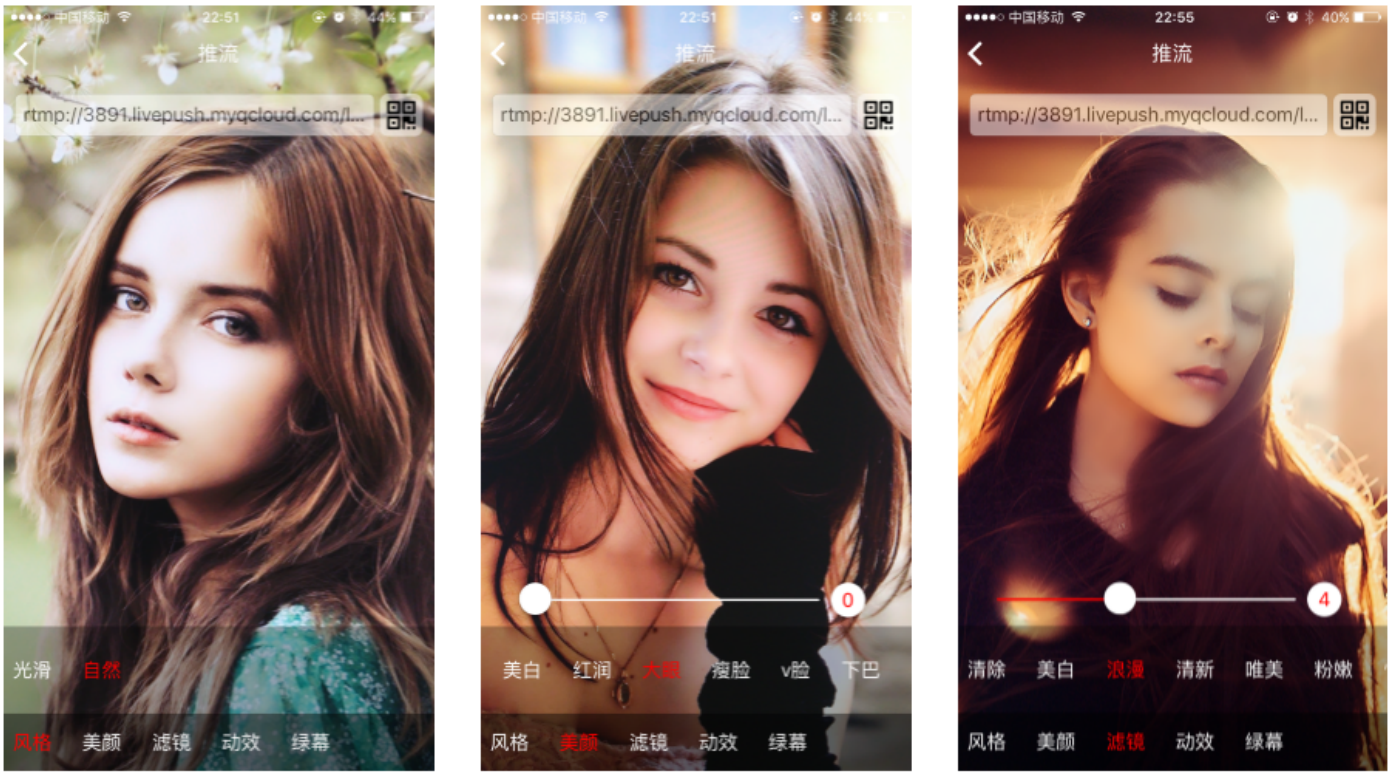
应用场景	quality	adjustBitrate	adjustResolution
秀场直播	VIDEO_QUALITY_HIGH_DEFINITION 或 VIDEO_QUALITY_SUPER_DEFINITION	false	false
手游直播	VIDEO_QUALITY_SUPER_DEFINITION	true	true
连麦（主画面）	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	true	true
连麦（小画面）	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	false	false
视频通话	VIDEO_QUALITY_REALTIEM_VIDEOCHAT	true	true

- **内部数据指标**

quality	adjustBitrate	adjustResolution	码率范围	分辨率范围
STANDARD	true	true	300~800kbps	270x480 ~ 360x640
STANDARD	true	false	300~800kbps	360x640
STANDARD	true	false	800kbps	360x640
HIGH	true	true	600~1500kbps	360x640~540x960

quality	adjustBitrate	adjustResolution	码率范围	分辨率范围
HIGH	true	false	600~1500kbps	540x960
HIGH	false	false	1200kbps	540x960
SUPER	true	true	600~1800kbps	360x640~720x1280
SUPER	true	false	600~1800kbps	720x1280
SUPER	false	false	1800kbps	720x1280
MAIN_PUBLISHER	true	true	600~1500kbps	360x640~540x960
SUB_PUBLISHER	false	false	350kbps	320x480
VIDEOCHAT	true	true	200~800kbps	190x320~360x640

step 5: 美颜滤镜



- **美颜**
  setBeautyFilter 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：

```
//style 磨皮风格： 0：光滑 1：自然 2：朦胧
//beautyLevel 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.
//whiteningLevel 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.
//ruddyLevel 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.
//
public boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel);
```

- **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。

```
Bitmap bmp = null;
bmp = decodeResource(getResources(), R.drawable.langman);
if (mLivePusher != null) {
mLivePusher.setFilter(bmp);
}
```

  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...

- **曝光**

  setExposureCompensation 可以调节曝光值，这个调整项在 iOS 端是没有的（我们使用了系统的自动曝光）。但是 Android 机型差异太大，很多千元机的自动曝光效果实在一般，所以我们推荐在您的 UI 界面上提供一个自动曝光的操作滑竿，让主播可以自己调节曝光值大小。

  setExposureCompensation 的参数为 -1 到 1 的浮点数： 0 表示不调整， -1 是将曝光降到最低， 1 表示是将曝光加强到最高。

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 TXLivePushConfig 的配置函数 setFrontCamera 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 TXLivePushConfig 和 TXLivePusher 对象都已经初始化。

```
// 默认是前置摄像头
mLivePusher.switchCamera();
```

- **打开或关闭闪光灯**

  只有后置摄像头才可以打开闪光灯，另外该接口需要在启动预览之后调用

```
//mFlashTurnOn为true表示打开，否则表示关闭
if (!mLivePusher.turnOnFlashLight(mFlashTurnOn)) {
Toast.makeText(getActivity().getApplicationContext(),
"打开闪光灯失败:绝大部分手机不支持前置闪光灯!", Toast.LENGTH_SHORT).show();
}
```

- **摄像头自动或手动对焦**

  大部分后置摄像头才支持对焦，SDK 支持两种对焦模式：**手动对焦**和**自动对焦**。

  自动对焦是系统提供的能力，但有些机型并不支持自动对焦。手动对焦和自动对焦是互斥的，开启自动对焦后，手动对焦将不生效。

  SDK 默认配置是手动对焦，您可以通过 TXLivePushConfig 的配置函数 setTouchFocus 接口进行切换：

```
mLivePushConfig.setTouchFocus(mTouchFocus);
mLivePusher.setConfig(mLivePushConfig);
```

- **设置镜像效果**

  主播一般是采用前置摄像头进行直播。所以对比主播端预览画面和观众端的画面，会发现左右颠倒。这个跟我们照镜子的原理是一样的。如果你想要让画面保持一致，需要设置播放端水平镜像。

```
// 需在 mLivePusher.setConfig(mLivePushConfig); 之后调用
mLivePusher.setMirror(true);
```

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
mLivePushConfig.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 10, 10);
//后面两个参数分别是水印位置的 X 轴坐标和 Y 轴坐标
mLivePusher.setConfig(mLivePushConfig);
```

如果您需要对水印图片的位置做机型适配，那么您需要调用水印归一化接口。

```
//设置视频水印
mLivePushConfig.setWatermark(mBitmap, 0.02f, 0.05f, 0.2f);
//参数分别是水印图片的 Bitmap、水印位置的 X 轴坐标，水印位置的 Y 轴坐标，水印宽度。后面三个参数取值范围是[0, 1]
//后面两个参数分别是水印位置的X轴坐标和 Y 轴坐标
mLivePusher.setConfig(mLivePushConfig);
```

step 8: 硬件编码

TXLivePushConfig 中的 **setHardwareAcceleration** 设置接口可以开启或关闭硬件编码。

```
if (mHWVideoEncode){
if (mLivePushConfig != null) {
if(Build.VERSION.SDK_INT < 18){
Toast.makeText(getApplicationContext(), "硬件加速失败，当前手机 API 级别过低（最低 18）",
Toast.LENGTH_SHORT).show();
mHWVideoEncode = false;
}
}
}

mLivePushConfig.setHardwareAcceleration(mHWVideoEncode ?
TXLiveConstants.ENCODE_VIDEO_HARDWARE : TXLiveConstants.ENCODE_VIDEO_SOFTWARE);
mLivePusher.setConfig(mLivePushConfig);

// 如果你不清楚要何时开启硬件加速, 建议设置为 ENCODE_VIDEO_AUTO
// 默认是启用软件编码, 但手机 CPU 使用率超过 80% 或者帧率 <= 10, SDK 内部会自动切换为硬件编码
```

mHWVideoEncode 有以下选项。

硬件加速选项	含义

硬件加速选项	含义
ENCODE_VIDEO_HARDWARE	开启硬件加速
ENCODE_VIDEO_SOFTWARE	禁用硬件加速，默认禁用硬件加速
ENCODE_VIDEO_AUTO	自动选择是否启用硬件加速

- **兼容性评估**
  Android 手机目前对硬件加速的支持已较前两年有明显的进步，目前支持度还是不错的，但仍有个别机型有兼容性问题，目前 RTMP SDK 通过一个内部的黑名单进行控制，避免在部分兼容性差的机型上出现问题。如果您使用硬件编码失败，SDK 内部会自动切换为软件编码。

- **效果差异**
  开启硬件加速后手机耗电量会有明显降低，机身温度也会比较理想，但画面大幅运动时马赛克感会比软编码要明显很多，而且越是早起的低端机，马赛克越是严重。所以如果您是对画质要求很高的客户，不推荐开启硬件加速。

- **推荐的设计**
  我们在 setVideoQuality 的高清档（推荐档位）和标清档均推荐使用软件编码，如果您担心 CPU 和发热问题，可以做一个简单的保护逻辑：

  > 如果发现推流的 **FPS**（通过 TXLivePushListener 的 NET_STATUS_VIDEO_FPS 事件可以获知）持续过低，比如半分钟内持续低于 10 帧/秒，表示 CPU 负载过重，则切换为硬编码。

step 9: 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 videoFilePath 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。
- 通过 setVideoRecordListener 接口，可以设置 TXRecordCommon.ITXVideoRecordListener 监听器给 TXLivePusher ，从而获取录制相关的事件通知。

```
// 启动录制：返回 0 开始录制成功；-1 表示正在录制，这次启动录制忽略；-2 表示还未开始推流，这次启动录制失败
public int startRecord(final String videoFilePath)
// 结束录制
```

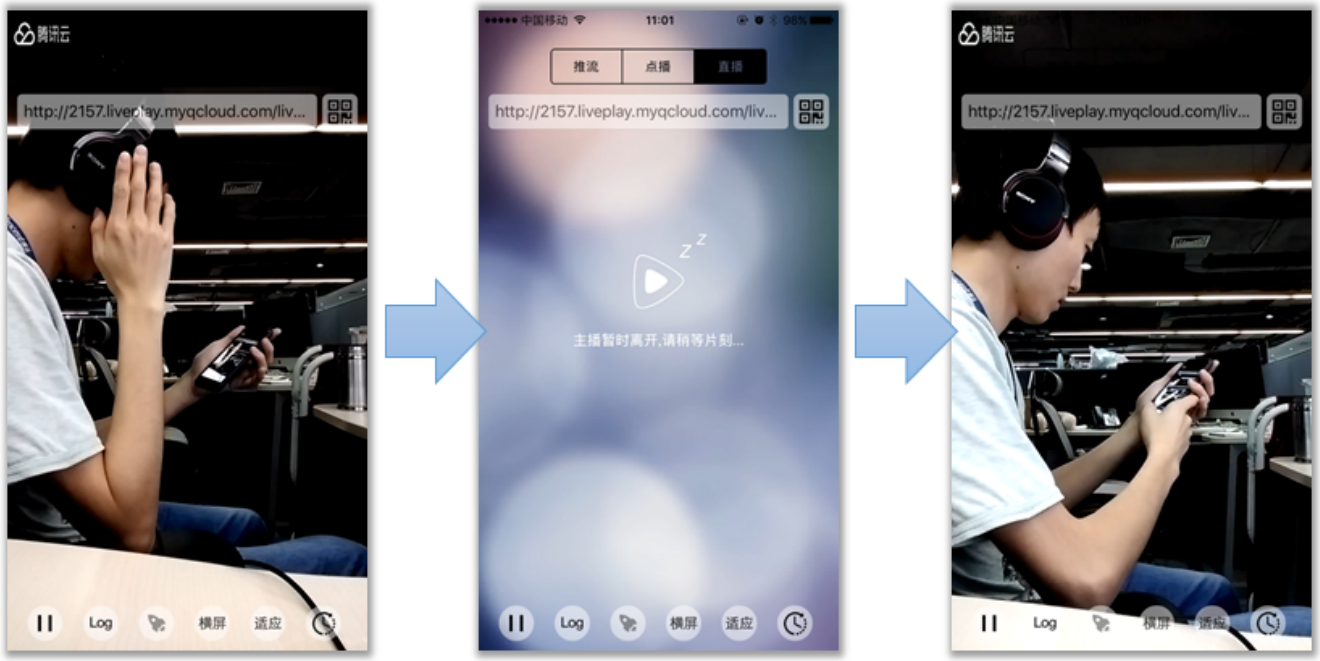
```
public void stopRecord()
//
// 视频录制回调
public interface ITXVideoRecordListener {
void onRecordEvent(final int event, final Bundle param);
void onRecordProgress(long milliSecond);
void onRecordComplete(TXRecordResult result);
}
```

step 10: 后台推流

常规模式下，App 一旦切到后台，摄像头的采集能力就被 Android 系统停掉了，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退出，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，但各云商的安全保护措施会让主播的直播被迫提前结束。

我们可以采用如下方案规避：



- **10.1、设置 pauseImg**
  在开始推流前，使用 TXLivePushConfig 的 setPauseImg 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”。

```
mLivePushConfig.setPauseImg(300,5);
// 300 为后台播放暂停图片的最长持续时间,单位是秒
// 10 为后台播放暂停图片的帧率,最小值为 5,最大值为 20
Bitmap bitmap = decodeResource(getResources(), R.drawable.pause_publish);
mLivePushConfig.setPauseImg(bitmap);
// 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.
mLivePusher.setConfig(mLivePushConfig);
```

- **10.2、设置setPauseFlag**

在开始推流前，使用 TXLivePushConfig 的 setPauseFlag 接口设置切后台 pause 推流时需要停止哪些采集，停止视频采集则会推送 pauseImg 设置的默认图，停止音频采集则会推送静音数据。

```
setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO|PAUSE_FLAG_PAUSE_AUDIO);//表示同时停止视频和音频采集，并且推送填充用的音视频流；

setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO);//表示停止摄像头采集视频画面，但保持麦克风继续采集声音，用于主播更衣等场景；
```

- **10.3、切后台处理**

推流中，如果App被切了后台，调用 TXLivePusher 中的 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。

```
// activity 的 onStop 生命周期函数
@Override
public void onStop(){
super.onStop();
mCaptureView.onPause(); // mCaptureView 是摄像头的图像渲染view
mLivePusher.pausePusher(); // 通知 SDK 进入“后台推流模式”了
}
```

- **10.4、切前台处理**

等待App切回前台之后，调用 TXLivePusher 的 resumePush 接口函数，之后，SDK 会继续采集摄像头的画面进行推流。

```
// activity 的 onStop 生命周期函数
@Override
public void onResume() {
super.onResume();
mCaptureView.onResume(); // mCaptureView 是摄像头的图像渲染view
```

```
mLivePusher.resumePusher(); // 通知 SDK 重回前台推流
}
```

step 10+: 后台推摄像头采集数据

如果希望主播在切后台或者跳转其他界面还能看到摄像头采集的画面， 按照以下配置即可。

1、step 10.1 和 step 10.2 无需设置。

2、在 step 10.3 中，注释 mLivePusher.pausePusher() 该方法。

3、在 step 10.4 中，注释 mLivePusher.resumePusher() 该方法。

注意：使用该功能注意保护主播隐私。

step 11: 网络质量提示

- 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 12: 横屏推流

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：

- **调整观众端表现**

  通过对 LivePushConfig 中的 **setHomeOrientation** 设置项进行配置，此接口提供了**手机旋转了 0，90，180，270 度** 四个参数供设置旋转角度。调整后的结果可以用播放器 Demo 查看以确认是否符合预期。

```
// 竖屏状态, 手机 Home 键在正下方。 旋转 0 度
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_DOWN);
// 横屏状态，手机 Home 键在右手方。 旋转 270 度
mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
```

- **调整主播端表现**

  观众端的画面表现符合预期以后，剩下要做的就是调整主播端的预览画面，这时可以通过 TXLivePusher 中的 setRenderRotation 接口，来旋转主播端看到的画面旋转方向，它决定主播端看到的视频宽高比是 **16:9** 还是 **6:19**

```
// 竖屏状态，本地渲染相对正方向的角度为0。
mLivePusher.setRenderRotation(0);
```

```
// 横屏状态，本地渲染相对正方向的角度为90。
mLivePusher.setRenderRotation(90);
```

- **Activity自动旋转**
  Android 系统的 Activity 本身支持跟随手机的重力感应进行旋转（设置 android:configChanges），CODE演示了如何做到下面这种重力感应效果：

step 13: 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 TXLivePusher 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 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 TXLivePusher 和用于显示影像的 TXCloudVideoView 都是不能多实例并行运转的，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
public void stopRtmpPublish() {
mLivePusher.stopCameraPreview(true); //停止摄像头预览
mLivePusher.stopPusher(); //停止推流
mLivePusher.setPushListener(null); //解绑 listener
}
```

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。

```
//Android 示例代码
mTXLivePusher.sendMessage(questionInfo.getBytes("UTF-8"));
```

播放端使用 TXLivePlayer 的 onPlayEvent（PLAY_EVT_GET_MESSAGE）可以用来接收消息。具体参考播放文档

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

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）
PUSH_EVT_CHANGE_RESOLUTION	1005	推流动态调整分辨率
PUSH_EVT_CHANGE_BITRATE	1006	推流动态调整码率

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
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）
PUSH_WARNING_SERVER_DISCONNECT	3004	RTMP 服务器主动断开连接（会触发重试流程）

全部事件定义请参阅头文件“**TXLiveConstants.java**”

直播(LivePlayer)

最近更新时间：2018-09-26 10:26:14

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: 添加 View

为了能够展示播放器的视频画面，我们第一步要做的就是在布局xml文件里加入如下一段代码：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/video_view"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_centerInParent="true"
android:visibility="gone"/>
```

step 2: 创建 Player

视频云 SDK 中的 **TXLivePlayer** 模块负责实现直播播放功能，并使用 **setPlayerView** 接口将这它与我们刚刚添加到界面上的 **video_view** 控件进行关联。

```
//mPlayerView 即 step1 中添加的界面 view
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);

//创建 player 对象
TXLivePlayer mLivePlayer = new TXLivePlayer(getActivity());

//关键 player 对象与界面 view
mLivePlayer.setPlayerView(mView);
```

step 3: 启动播放

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startPlay(flvUrl, TXLivePlayer.PLAY_TYPE_LIVE_FLV); //推荐 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 step1 中添加的 “video_view” 控件的大小和位置即可。

- **setRenderMode：铺满or适应**

可选值	含义
RENDER_MODE_FULL_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_ADJUST_RESOLUTION	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）

```
// 设置填充模式
mLivePlayer.setRenderMode(TXLiveConstants.RENDER_MODE_ADJUST_RESOLUTION);
// 设置画面渲染方向
mLivePlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_LANDSCAPE);
```

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
mLivePlayer.pause();
// 继续
mLivePlayer.resume();
```

step 6: 结束播放

结束播放时 **记得销毁view控件** ，尤其是在下次 startPlay 之前，否则会产生大量的内存泄露以及闪屏问题。

同时，在退出播放界面时，记得一定要调用渲染View的 `onDestroy()` 函数，否则可能会产生内存泄露和 “Receiver not registered” 报警。

```
@Override
public void onDestroy() {
super.onDestroy();
mLivePlayer.stopPlay(true); // true 代表清除最后一帧画面
```

```
mView.onDestroy();
}
```

stopPlay 的布尔型参数含义为—— “是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。

如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

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 **setEnableMessage** 开关置为 **true**。
- TXLivePlayer 通过 **TXLivePlayListener** 监听消息，消息编号：**PLAY_EVT_GET_MESSAGE（2012）**

```
//Android 示例代码
mTXLivePlayer.setPlayListener(new ITXLivePlayListener() {
@Override
public void onPlayEvent(int event, Bundle param) {
if (event == TXLiveConstants.PLAY_ERR_NET_DISCONNECT) {
roomListenerCallback.onDebugLog("[AnswerRoom] 拉流失败：网络断开");
roomListenerCallback.onError(-1, "网络断开，拉流失败");
}
else if (event == TXLiveConstants.PLAY_EVT_GET_MESSAGE) {
String msg = null;
try {
msg = new String(param.getByteArray(TXLiveConstants.EVT_GET_MSG), "UTF-8");
roomListenerCallback.onRecvAnswerMsg(msg);
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}
}
@Override
public void onNetStatus(Bundle status) {
}
});
```

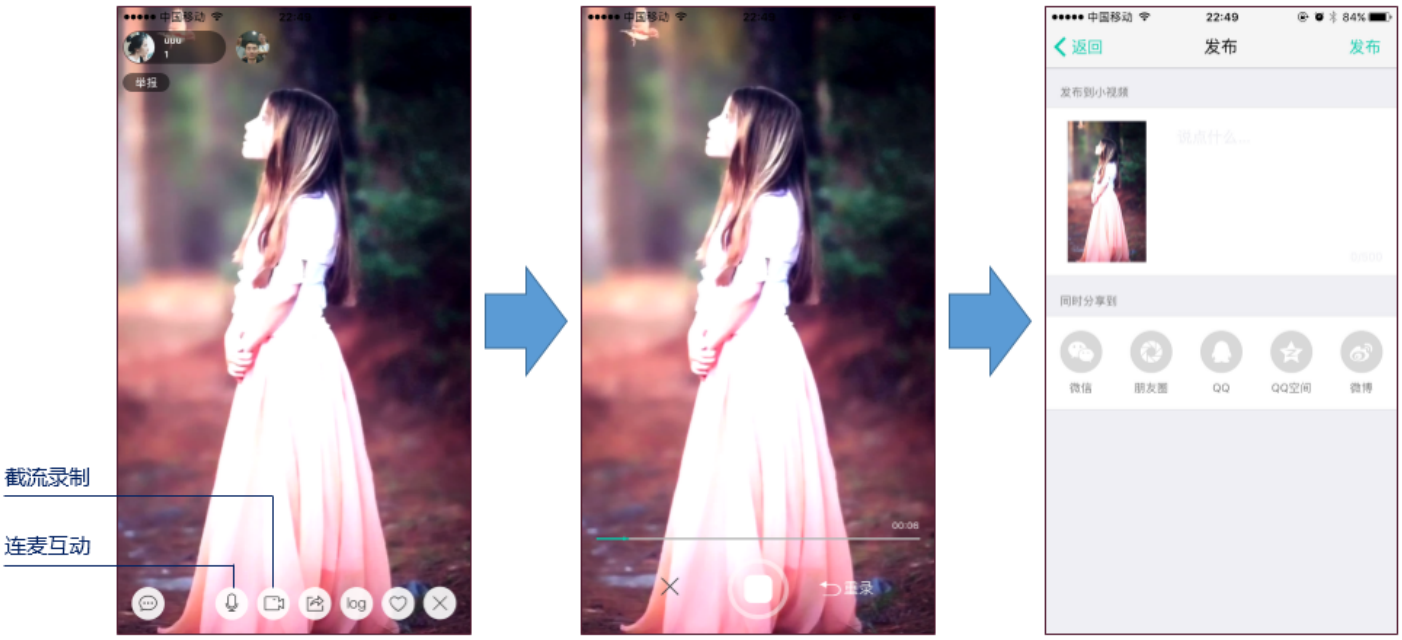
step 8: 屏幕截图

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 Android 的系统 API 来实现。


```
mLivePlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
//获取到截图 bitmap
}
}
});
```

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。




```
//指定一个 ITXVideoRecordListener 用于同步录制的进度和结果
mLivePlayer.setVideoRecordListener(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
mLivePlayer.startRecord(int recordType);
// ...
// ...
//结束录制，可放于结束按钮的响应函数里
mLivePlayer.stopRecord();
```

- 录制的进度以时间为单位，由 ITXVideoRecordListener 的 onRecordProgress 通知出来。
- 录制好的文件以 MP4 文件的形式，由 ITXVideoRecordListener 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
mLivePlayer.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
mLivePlayer.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

```
mLivePlayer.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

控制模式	卡顿率	平均延迟	适用场景	原理简述
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 mPlayConfig = new TXLivePlayConfig();
//
//自动模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(5);
//
//极速模式
mPlayConfig.setAutoAdjustCacheTime(true);
mPlayConfig.setMinAutoAdjustCacheTime(1);
mPlayConfig.setMaxAutoAdjustCacheTime(1);
//
//流畅模式
mPlayConfig.setAutoAdjustCacheTime(false);
mPlayConfig.setMinAutoAdjustCacheTime(5);
mPlayConfig.setMaxAutoAdjustCacheTime(5);
//
mLivePlayer.setConfig(mPlayConfig);
//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时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是技术能力限制，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），避免因为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）

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）

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
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 16:21:54

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: 添加View

为了能够展示播放器的视频画面，我们第一步要做的就是在布局xml文件里加入如下一段代码：

```
<com.tencent.rtmp.ui.TXCloudVideoView
android:id="@+id/video_view"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_centerInParent="true"
android:visibility="gone"/>
```

step 2: 创建Player

接下来创建一个**TXVodPlayer**的对象，并使用 setPlayerView 接口将这它与我们刚刚添加到界面上的**video_view**控件进行关联。

```
//mPlayerView即step1中添加的界面view
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);
//创建player对象
TXVodPlayer mVodPlayer = new mVodPlayer(getActivity());
//关键player对象与界面view
mVodPlayer.setPlayerView(mView);
```

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

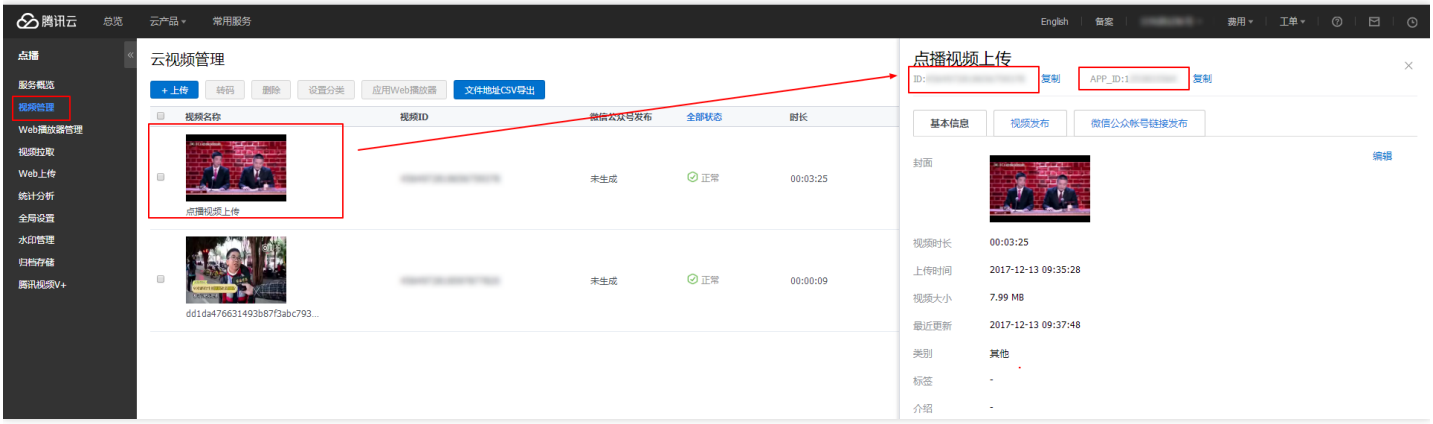
   TXVodPlayer 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startPlay 函数即可。

   ```
   String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
   mVodPlayer.startPlay(url);
   ```

2. 通过fileId方式

   ```
   TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
   authBuilder.setAppId(1252463788);
   authBuilder.setFileId("4564972819220421305");
   mVodPlayer.startPlay(authBuilder);
   ```

   在点播视频管理 找到对应的文件。点开后在右侧视频详情中，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 TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL 事件，反之收到 TXLiveConstants.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 step1中添加的 “video_view” 控件的大小和位置即可。

- **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_ADJUST_RESOLUTION	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: 播放控制

```
// 调整进度
mVodPlayer.seek(seekBar.getProgress());
// 暂停播放
mVodPlayer.pause();
// 恢复播放
mVodPlayer.resume();
```

step 6: 结束播放

结束播放时 **记得销毁view控件** ，尤其是在下次startPlay之前，否则会产生大量的内存泄露以及闪屏问题。

同时，在退出播放界面时，记得一定要调用渲染View的 `onDestroy()` 函数，否则可能会产生内存泄露和 “Receiver not registered” 报警。

```
@Override
public void onDestroy() {
super.onDestroy();
mVodPlayer.stopPlay(true); // true代表清除最后一帧画面
mView.onDestroy();
}
```

stopPlay 的布尔型参数含义为—— “是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。

如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

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 Android 的系统 API 来实现。


```
mVodPlayer.snapshot(new ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
if (null != bmp) {
//获取到截图bitmap
}
}
});
```

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

```
//如下代码用于展示点播倍速播放
//设置1.2倍速播放
mVodPlayer.setRate(1.2);
// ...
//开始播放
mVodPlayer.startPlay(playUrl,_playType);
```

step 9: 本地缓存

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。

```
//指定一个本地mp4缓存目录
TXVodPlayConfig mConfig = new TXVodPlayConfig();
mConfig.setCacheFolderPath(
Environment.getExternalStorageDirectory().getPath(); +"/txcache");
```

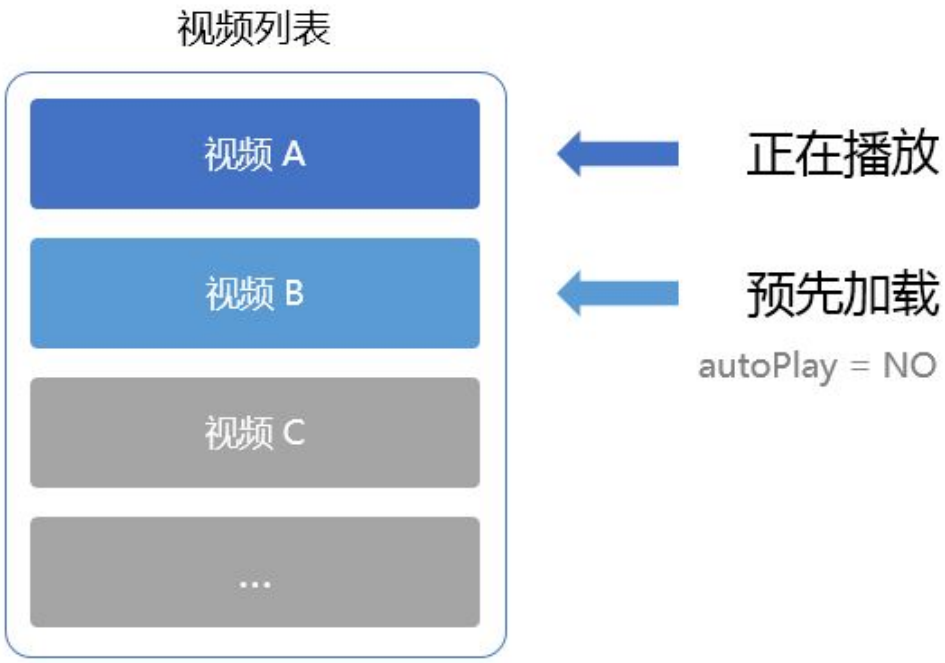
```
//指定本地最多缓存多少文件，避免缓存太多数据
mConfig.setMaxCacheItems(10);
mVodPlayer.setConfig(mConfig);
// ...
//开始播放
mVodPlayer.startPlay(playUrl);
```

缓存的文件可能会被系统图库扫描到，如果您不希望缓存的文件出现在系统图库中，您可以在缓存目录下新建一个名为".nomedia"的空文件。系统图库发现该文件存在后，会跳过扫描此目录。

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 setAutoPlay 开关来实现这个功能，具体做法如下：




```
// 播放视频A: 如果将 autoPlay 设置为 true， 那么 startPlay 调用会立刻开始视频的加载和播放
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
playerA.setAutoPlay(true);
```

```
playerA.startPlay(urlA);

// 在播放视频 A 的同时，预加载视频 B，做法是将 true 设置为 false
String urlB = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
playerB.setAutoPlay(false);
playerB.startPlay(urlB); // 不会立刻开始播放，而只会开始加载视频
```

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。

```
public void onPlayEvent(TXVodPlayer player, int event, Bundle 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 (event == PLAY_EVT_PLAY_END) {
playerA.stop();
playerB.setPlayerView(mPlayerView);
playerB.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 false 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

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

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。

```
mVodPlayer.stopPlay(true);
mVodPlayer.enableHardwareDecode(true);
mVodPlayer.startPlay(flvUrl, type);
```

step 15: 多码率文件

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

```
ArrayList<TXBitrateItem> bitrates = mVodPlayer.getSupportedBitrates(); //获取多码率数组
```

在播放过程中，可以随时通过 mVodPlayer.setBitrateIndex(int) 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 **TXVodPlayerListener** 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。

```
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {

if (event == PLAY_EVT_PLAY_PROGRESS) {
// 加载进度, 单位是秒
int duration = param.getInt(TXLiveConstants.EVT_PLAYABLE_DURATION);
mLoadBar.setProgress(duration);

// 播放进度, 单位是秒
int progress = param.getInt(TXLiveConstants.EVT_PLAY_PROGRESS);
mSeekBar.setProgress(progress);

// 视频总长, 单位是秒
int duration = param.getInt(TXLiveConstants.EVT_PLAY_DURATION);
// 可以用于设置时长显示等等
}
}
```

如果点播播放场景需要获取到毫秒级别的时间戳来加载字幕，您需要用到以下回调。

```
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {

if (event == PLAY_EVT_PLAY_PROGRESS) {
// 加载进度, 单位是毫秒
int duration_ms = param.getInt(TXLiveConstants.EVT_PLAYABLE_DURATION_MS);
mLoadBar.setProgress(duration_ms);

// 播放进度, 单位是毫秒
int progress_ms = param.getInt(TXLiveConstants.EVT_PLAY_PROGRESS_MS);
mSeekBar.setProgress(progress_ms);

// 视频总长, 单位是毫秒
int duration_ms = param.getInt(TXLiveConstants.EVT_PLAY_DURATION_MS);
// 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）
PLAY_WARNING_SEVER_CONN_FAIL	3002	RTMP服务器连接失败（仅播放RTMP地址时会抛送）

事件ID	数值	含义说明
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_ROATION	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
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽

评估参数	含义说明
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.getWidth()` 和 `TXVodPlayer.getHeight()` 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 `TXLiveConstants.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.getInstance();
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/26f327f9vodgzp1253131631/f4bdff799031868222924043041/playlist.m3u8");
```

fileid下载至少需要传入appId和fileId

```
TXPlayerAuthBuilder auth = new TXPlayerAuthBuilder();
auth.setAppId(1252463788);
auth.setFileId("4564972819220421305");
TXVodDownloadDataSource source = new TXVodDownloadDataSource(auth, QUALITY_OD);
downloader.startDownload(source);
```

> fileid的获取方式可参考 https://cloud.tencent.com/document/product/454/12148#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listener

```
downloader.setListener(this);
```

可能收到的任务回调有：

1. void onDownloadStart(TXVodDownloadMediaInfo mediaInfo)
   任务开始，表示SDK已经开始下载。
2. void onDownloadProgress(TXVodDownloadMediaInfo mediaInfo)
   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. void onDownloadStop(TXVodDownloadMediaInfo mediaInfo)
   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. void onDownloadFinish(TXVodDownloadMediaInfo mediaInfo)
   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. void onDownloadError(TXVodDownloadMediaInfo mediaInfo, int error, String reason)
   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。错误码位

于 TXVodDownloadManager 中。

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 downloader.stopDownload() 方法，参数为 downloader.startDownload() 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 downloader.deleteDownloadFile() 方法删除文件，以释放存储空间。

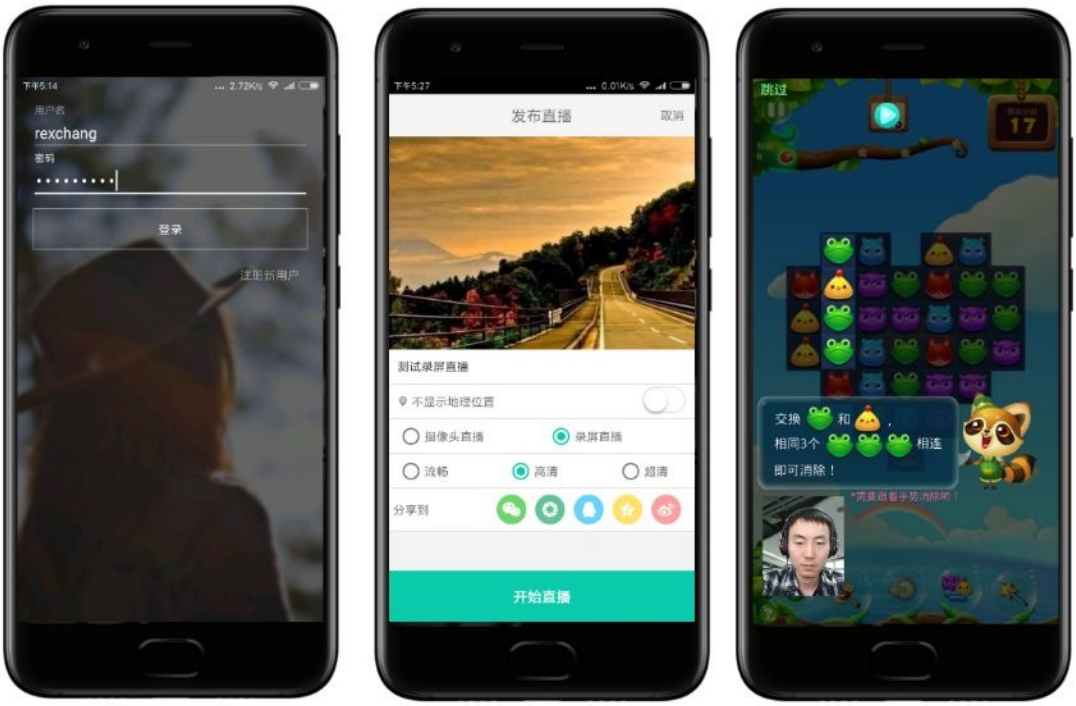
录屏(ScreenCapturer)

最近更新时间：2018-08-10 16:21:58

功能介绍

手机录屏直播，即可以直接把主播的手机画面作为直播源，同时可以叠加摄像头预览，应用于游戏直播、移动端APP演示等需要手机屏幕画面的场景。

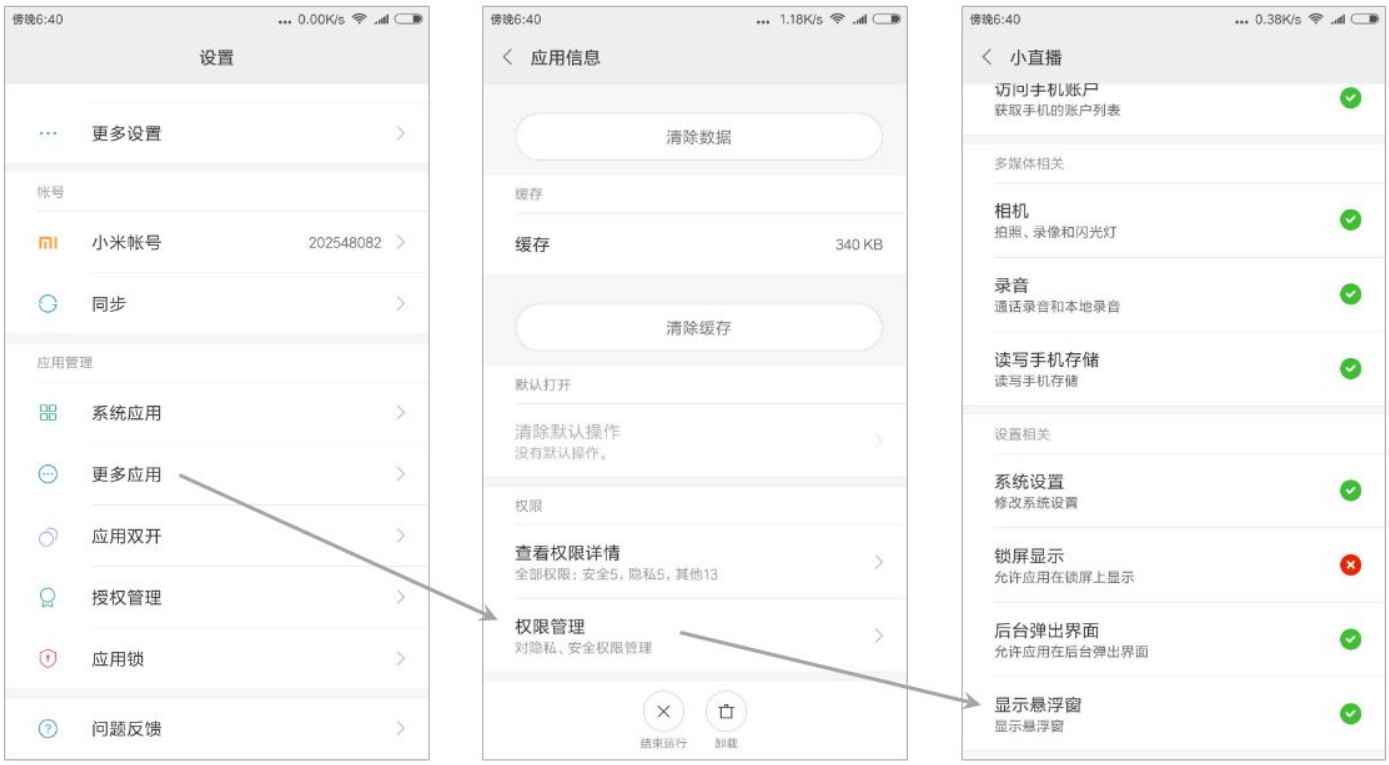
扫码安装小直播



录屏功能在 iOS 和 Android 下有两套截然不同的实现方案，本文档

限制说明

- Android 5.0 系统以后开始支持录屏功能。
- 悬浮窗在部分手机和系统上需要通过手动设置打开。
- 录屏直播时，请先关闭小米的**神隐模式**。



对接攻略

step 1: 添加Activity

在manifest文件中黏贴进如下一个activity

```
<activity
android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
android:theme="@android:style/Theme.Translucent"/>
```

step 2: 创建Pusher对象

创建一个**TXLivePusher**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new出来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，简单地塞给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。

```
TXLivePusher mLivePusher = new TXLivePusher(getActivity());
mLivePushConfig = new TXLivePushConfig();
mLivePusher.setConfig(mLivePushConfig);
```

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：

```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(rtmpUrl);
mLivePusher.startScreenCapture();
```

- **startPusher** 的作用是告诉 RTMP SDK 音视频流要推到哪个推流URL上去。
- **startScreenCapture** 的作用是启动屏幕录制，由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出一个提示，旨在告诫用户："有 App 要截取您屏幕上的所有内容"。

step 4: 隐私模式

隐私模式是录屏直播的一项基础功能：主播在录屏直播过程中，如果有些操作不希望观众看到（比如输入游戏的账号密码等等），那么此时TA可以启动**隐私模式**，隐私模式下，主播的推流还是持续的，观众也一直能看到的画面，只是看到的画面是一张提示"主播正在忙..."的等待中画面。

要实现这样功能，您可以按如下步骤进行对接：

- **4.1) 设置pauseImg**
  在开始推流前，使用 TXLivePushConfig 的 setPauseImg 接口设置一张等待图片，比如"主播把画面切走一会儿..."。

- **4.2) 隐私模式开关**
  在用于工具条的悬浮窗口上增加一个用于开关隐私模式的按钮，打开隐私模式的响应逻辑为对 TXLivePusher##pausePush 接口函数的调用，关闭隐私模式的响应逻辑为对 TXLivePusher##resumePush 接口函数的调用。

```
public void triggerPrivateMode() {
if (mInPrivacy) {
Toast.makeText(getApplicationContext(), "隐私模式已开启"，Toast.LENGTH_SHORT).show();
mTVPrivateMode.setText(getString(R.string.private_mode_off));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOn,null,null,null);
mPrivateBtn.setImageResource(R.mipmap.lock_off);
mTXLivePusher.pausePusher();
} else {
Toast.makeText(getApplicationContext(), "隐私模式已关闭"，Toast.LENGTH_SHORT).show();
mTXLivePusher.resumePusher();
mPrivateBtn.setImageResource(R.mipmap.lock_on);
mTVPrivateMode.setText(getString(R.string.private_mode_on));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOff,null,null,null);
}
mInPrivacy = !mInPrivacy;
}
```

step 5: 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
mLivePushConfig.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 10, 10);
mLivePusher.setConfig(mLivePushConfig);
```

step 6: 推荐的清晰度

影响画质的主要因素是三个：**分辨率**、**帧率**和**码率**。

- **分辨率**
  手机录屏直播提供了三个级别的分辨率可供选择：360*640，540*960，720*1280，设置接口为 TXLivePushConfig 中的 setVideoResolution。
- **帧率**
  FPS <=10 会明显感觉到卡顿，手机录屏直播推荐设置 20 - 25 FPS 的帧率，设置接口为TXLivePushConfig 中的 setVideoFPS。
- **码率**
  编码器每秒编出的数据大小，单位是kbps，比如800kbps代表编码器每秒产生800kb（或100KB）的数据。设置接口为TXLivePushConfig 中的 setVideoBitrate。

相比于摄像头直播，录屏直播的不确定性会大很多，其中一个最大的不确定性因素就是录屏的场景。
（1）一种极端就是手机屏幕停在一个界面保持不动，比如桌面，这个时候编码器可以用很小的码率输出就能完成任务。

（2）另一种极端情况就是手机屏幕每时每刻都在发生剧烈的变化，比如主播在玩《神庙逃跑》，这个时候即使 540 * 960 的普通分辨率也至少需要 2Mbps 的码率才能保证没有马赛克。

档位	分辨率	FPS	码率-游戏录屏（扑鱼达人）	码率-游戏录屏（神庙逃跑）
标清	VIDEO_RESOLUTION_TYPE_360_640	20	800kbps	1200kbps
高清	VIDEO_RESOLUTION_TYPE_540_960	20	1200kbps	2000kbps
超清	VIDEO_RESOLUTION_TYPE_720_1280	20	1600kbps	3000kbps

step 7: 提醒主播“网络不好”

step 9 中会介绍 RTMP SDK 的推流事件处理，其中 **PUSH_WARNING_NET_BUSY** 这个很有用，它的含义是：**当前主播的上行网络质量很差，观众端已经出现了卡顿。**

当收到此WARNING时，您可以通过UI提醒主播换一下网络出口，或者离WiFi近一点，或者让他吼一嗓子：“领导，我在直播呢，别上淘宝了行不！什么？没上淘宝？那韩剧也是一样的啊。”

step 8: 横竖屏适配

腾讯云 RTMP SDK 中内部已经实现了动态横竖屏切换视频逻辑，所以在使用录屏直播时无需关注这个问题，主播的手机在横竖屏切换的时候，观众端看到的画面会同主播的视角保持一致。

step 9: 向SDK填充自定义Audio数据

如果您希望把音频的采集替换成自己的逻辑，需要为 CustomMode 设置项追加 CUSTOM_MODE_AUDIO_CAPTURE，于此同时，您也需要指定声音采样率等和声道数等关键信息。

```
// (1)将 CustomMode 设置为：自己采集音频数据，SDK只负责编码&发送
_config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE;
//
// (2)设置音频编码参数：音频采样率和声道数
_config.audioSampleRate = 44100;
_config.audioChannels = 1;
```

之后，调用**sendCustomPCMData**向SDK塞入您自己的PCM数据即可。

step 10: 事件处理

事件监听

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

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）

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
PUSH_ERR_OPEN_CAMERA_FAIL	-1301	打开摄像头失败

事件ID	数值	含义说明
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

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。

- PUSH_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）。

- PUSH_WARNING_SERVER_DISCONNECT
  推流请求被后台拒绝了，会触发有限次数的重试逻辑，有可能可以在某一次重试中推流成功。但实话实说，大部分场景中都是推流地址里的txSecret计算错了，或者被其他人占用了测试地址，所以这个 warning 对您的调试帮助意义更大。

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）

全部事件定义请参阅头文件“**TXLiveConstants.java**”

step 11: 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 TXLivePusher 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。

```
//结束录屏直播，注意做好清理工作
public void stopPublish() {
mTXLivePusher.stopScreenCapture();
mTXLivePusher.setPushListener(null);
mTXLivePusher.stopPusher();
}
```

特效(AI Effects)

最近更新时间：2018-08-10 16:22:02

特效功能（大眼、瘦脸、动效、绿幕等）

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

申请步骤如下：

1. 提工单或客服电话（400-9100-100）联系我们商务同学。

2. 下载示例表格，按照表格填好信息后，邮件发送到 jerryqian@tencent.com 并抄送给您联系的商务同学（重要）。

3. 敦促商务同学回复邮件确认，未经腾讯云商务同学确认的邮件，我们可能会视为骚扰邮件不予处理。

4. 确认后，我们会第一时间替您向优图实验室申请试用 Licence，并同压缩包解压密码一起发给您。

   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 页面下方下载商用版本 SDK 压缩包，压缩包有加密（解压密码 & licence文件 可以跟我们的商务同学获取），成功解压后得到一个 `LiteAVSDK_Enterprise_3.9.2749.aar` 和 `LiteAVSDK_Enterprise_3.9.2749.zip` ，分别对应两种集成方式。

工程设置

参考 工程配置

添加SDK

使用aar方式集成

直接把LiteAVSDK_Enterprise_3.9.2749.aar替换你工程中的非商业版的aar，并在app目录下的build.gradle中修改对应的名称即可，相对简单

使用jar包方式集成

1. 需要解压LiteAVSDK_Enterprise_3.9.2749.zip，把libs下的jar包和so拷贝到你的jni加载路径下。其中跟动效有关的jar包和so如下：

jar		
filterengine.bundle.jar	ptu_algo_cb6bc16f389.jar	segmenter-lib.jar
video_module.jar	YTCommon.jar	

so		
libalgo_rithm_jni.so	libalgo_youtu_jni.so	libformat_convert.so
libGestureDetectJni.so	libimage_filter_common.so	libimage_filter_gpu.so
libnnpack.so	libParticleSystem.so	libpitu_tools.so
libsegmentern.so	libsegmentero.so	libYTCommon.so
libYTFaceTrackPro.so	libYTHandDetector.so	libYTIllumination.so

1. 把解压后的assets文件夹下的资源拷贝到你的工程的assets目录下，包括asset根目录下的文件和camera文件夹下的文件

导入licence文件

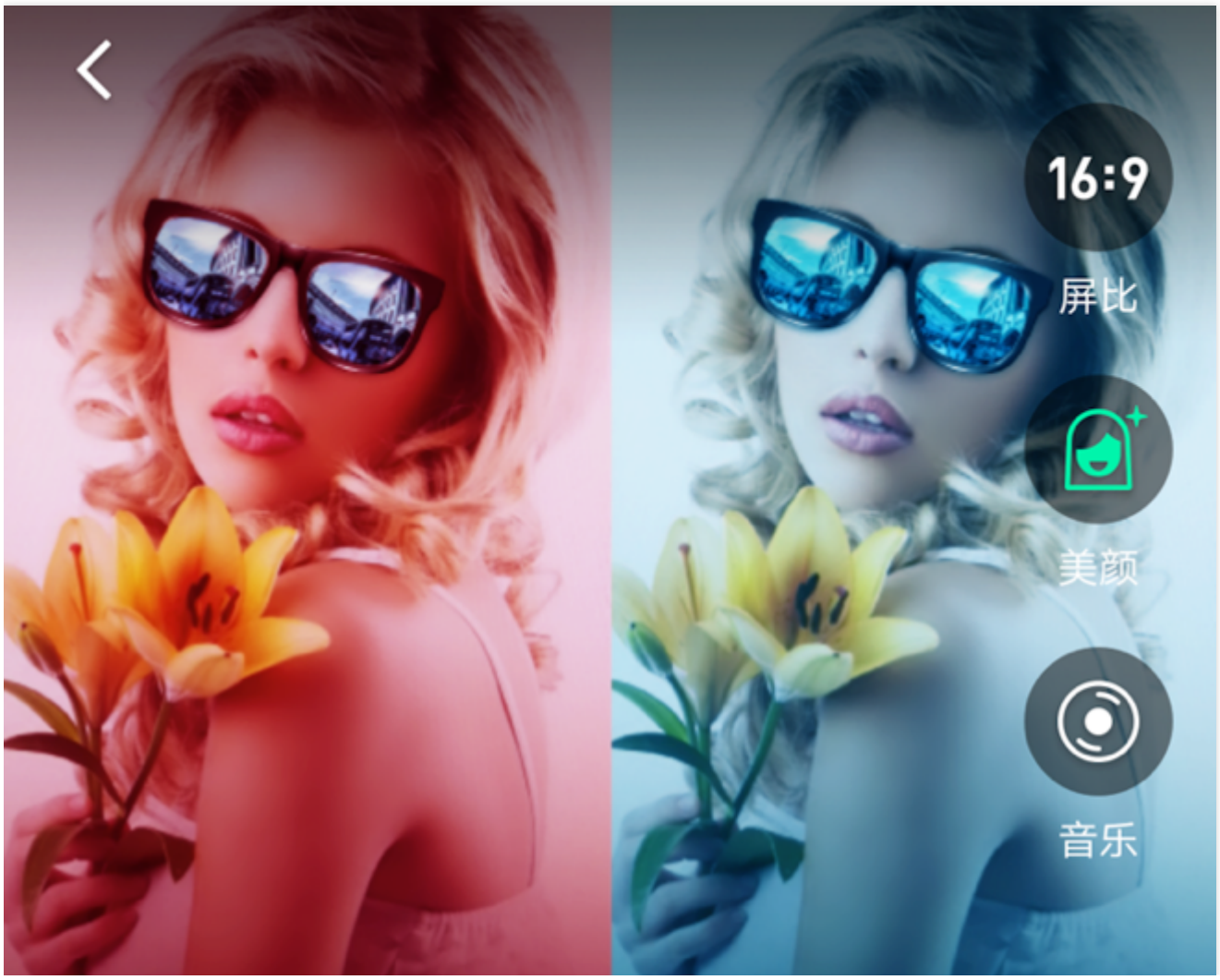
商用版需要 licence 验证通过后，相应功能才能生效。您可以向我们的商务同学申请一个免费 30 天的调试用 licence。
得到 licence 后，您需要将其命名为**YTFaceSDK.licence**，放到工程的assets目录下。

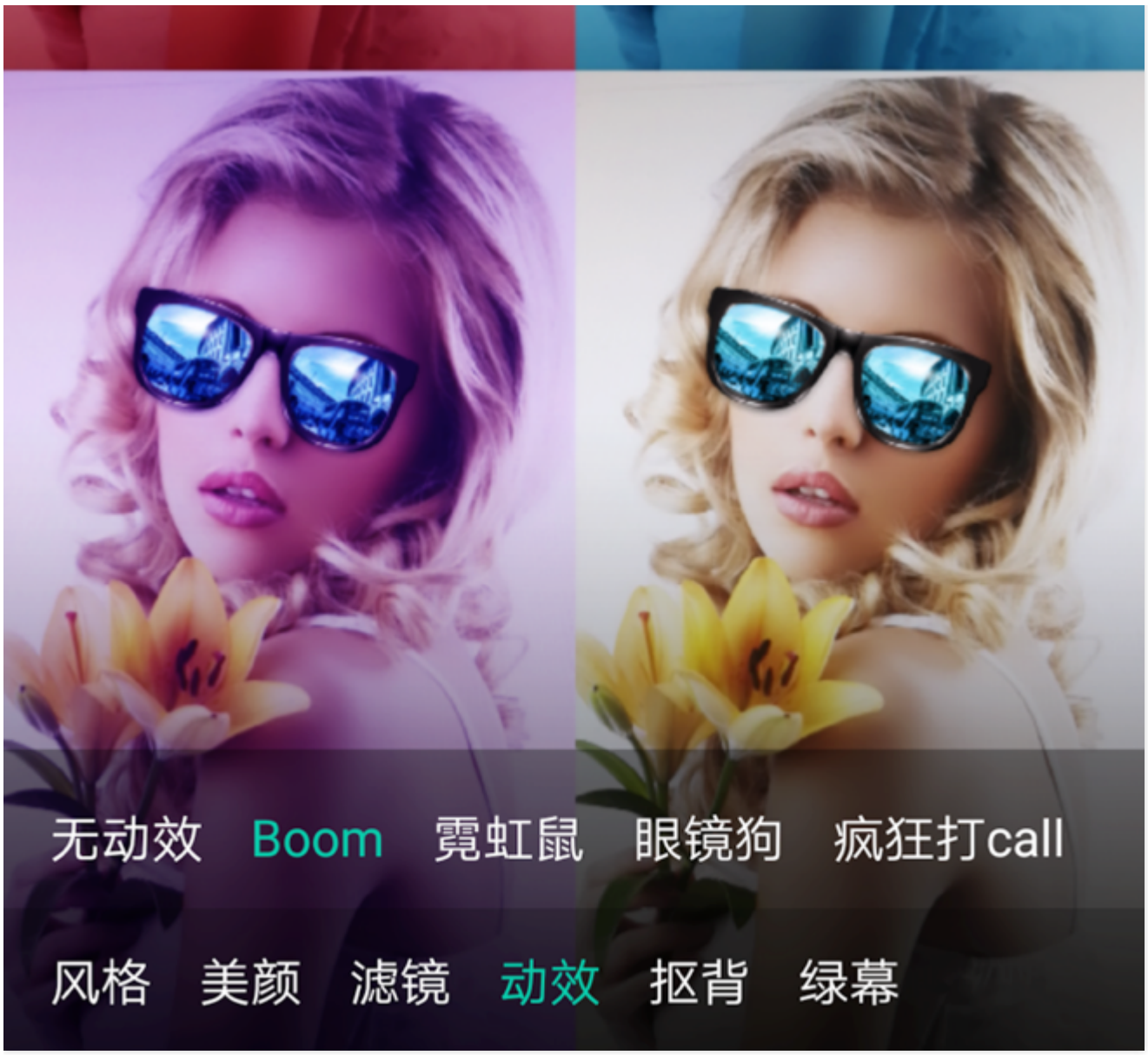
> 每个licence都有绑定具体的package name，修改app的package name会导致验证失败。
>
> YTFaceSDK.licence的文件名固定，不可修改、且必须放在assets目录下。
>
> iOS 和 Android 不需要重复申请 licence，一个 licence 可以同时授权一个 iOS 的 bundleid 和一个 Android 的packageName。

功能调用

1.动效功能

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。

DEMO中的示例代码是从后台下载动效资源，再统一解压到sdcard。您可以在DEMO代码中找到动效资源的下载地址，格式如下

```
http://dldir1.qq.com/hudongzhibo/AISpecial/Android/156/(动效name).zip
```

强烈建议客户将动效资源放在自己的服务器上，以防DEMO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

```
/**
* setMotionTmpl 设置动效贴图文件位置
* @param tmplPath
*/
public void setMotionTmpl(String tmplPath);
```

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

```
/**
* setMotionTmpl 设置动效贴图文件位置
* @param tmplPath
*/
public void setMotionTmpl(String tmplPath);
```

3. 美妆美容

```
// 大眼效果 0~9
mTXCameraRecord.setEyeScaleLevel(eyeScaleLevel);
// 瘦脸效果 0~9
mTXCameraRecord.setFaceScaleLevel(faceScaleLevel);
// V脸效果 0~9
mTXCameraRecord.setFaceVLevel(level)
// 下巴拉伸或收缩效果 0~9
mTXCameraRecord.setChinLevel(scale)
// 缩脸效果 0~9
mTXCameraRecord.setFaceShortLevel(level)
// 瘦鼻效果 0~9
mTXCameraRecord.setNoseSlimLevel(scale)
```

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:目前图片支持jpg/png，视频支持mp4/3gp等Android系统支持的格式
* API要求18
* @param path ：绿幕文件位置，支持两种方式：
* 1.资源文件放在assets目录，path直接取文件名
* 2.path取文件绝对路径
*/
@TargetApi(18)
public void setGreenScreenFile(String path);
```

问题排查

1. 工程运行过程中crash？

1. 检查build.gradle设置，abiFilters是否设置为armeabi，目前只支持armeabi架构
2. 如果是jar集成方式，检查动效对应的so是否都考到工程jniLibs目录下

2. 工程特效不生效？

1. 检查YTFaceSDK.licence 命名是否正确，YTFaceSDK.licence必须放在assets根目录下
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. 如果是jar集成方式，检查pitu资源是否添加正确（sdk解压出来的assets目录内容都要拷贝到工程的assets目录下）
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：2018-07-23 10:53:27

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

名称	描述
setLiveRoomListener(ILiveRoomListener listener)	设置liveroom回调
login(serverDomain, loginInfo, loginCallback)	登录到RoomService后台
logout()	从RoomService后台登出
getRoomList(int index, int count, GetRoomListCallback callback)	获取房间列表（非必须，可选择使用您自己的房间列表）
getAudienceList(String roomID, final GetAudienceListCallback callback)	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）
createRoom(String roomID, String roomInfo, CreateRoomCallback cb)	主播：创建房间 （roomID 可不填）
enterRoom(String roomID, TXCloudVideoView videoView, EnterRoomCallback cb)	观众：进入房间
exitRoom(ExitRoomCallback callback)	主播 OR 观众：退出房间
startLocalPreview(TXCloudVideoView videoView)	主播 OR 连麦观众：开启摄像头预览
stopLocalPreview()	停止摄像头预览
requestJoinPusher(int timeout, RequestJoinPusherCallback callback)	观众：发起连麦请求
joinPusher(final JoinPusherCallback cb)	观众：进入连麦状态
quitPusher(final QuitPusherCallback cb)	观众：退出连麦状态
acceptJoinPusher(String userID)	主播：接受来自观众的连麦请求
rejectJoinPusher(String userID, String reason)	主播：拒绝来自观众的连麦请求
kickoutSubPusher(String userID)	主播：踢掉连麦中的某个观众
getOnlinePusherList(final GetOnlinePusherListCallback callback)	主播PK：获取在线的主播列表

``

名称	描述
startPlayPKStream(final String playUrl, TXCloudVideoView videoView, final PKStreamPlayCallback callback)	主播PK：开始播放对方的视频流
stopPlayPKStream()	主播PK：结束播放对方的视频流
sendPKRequest(String userID, int timeout, final RequestPKCallback callback)	主播PK：发送PK请求
sendPKFinishRequest(String userID)	主播PK：发送结束PK的请求
acceptPKRequest(String userID)	主播PK：授受PK请求
rejectPKRequest(String userID, String reason)	主播PK：拒绝PK请求
addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)	主播：播放连麦观众的远程视频画面
deleteRemoteView(PusherInfo pusherInfo)	主播：移除连麦观众的远程视频画面
sendRoomTextMsg(String message, SendTextMessageCallback callback)	发送文本（弹幕）消息
sendRoomCustomMsg(String cmd, String message, SendCustomMessageCallback callback)	发送自定义格式的消息（点赞，送花）
startScreenCapture()	开始屏幕录制 （仅 Android）
stopScreenCapture()	停止屏幕录制 （仅 Android）
switchToBackground()	App 从前台切换到后台
switchToForeground()	App 从后台切换到前台
setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel)	设置美颜
switchCamera()	切换前后置摄像头，支持在推流中动态切换
setMute(mute)	静音

名称	描述
setMirror(enable)	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）
playBGM(String path)	开始播放背景音乐（path 指定音乐文件路径）
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(x)	设置混音时麦克风的音量大小
setBGMVolume(x)	设置混音时背景音乐的音量大小
getMusicDuration(fileName)	获取背景音乐时长
startRecord(recordType)	开始视频录制
stopRecord()	停止视频录制
setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)	设置视频录制回调
incCustomInfo(fieldName, count)	增加房间自定义数值 fieldName
decCustomInfo(fieldName, count)	减少房间自定义数值 fieldName
updateSelfUserInfo(userName, userAvatar)	更新liveroom的用户信息
setPauseImage(bitmap)	设置后台时推送的图片

ILiveRoomListener

名称	描述

名称	描述
onGetPusherList(pusherList)	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）
onPusherJoin(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）
onPusherQuit(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）
onRecvJoinPusherRequest(userID, userName, userAvatar)	通知：主播收到观众的连麦请求
onKickOut()	观众：收到被大主播踢开的消息
onRecvPKRequest(String userID, String userName, String userAvatar, String streamUrl)	收到PK请求
onRecvPKFinishRequest(String userID)	收到结束PK的请求
onRecvRoomTextMsg(roomID, userID, userName, userAvatar, message)	聊天室：收到文本消息
onRecvRoomCustomMsg(roomID, userID, userName, userAvatar, cmd, message)	聊天室：收到自定义消息
onRoomClosed(roomID)	通知：房间解散通知
onDebugLog(log)	LOG：日志回调
onError(errorCode, errorMessage)	ERROR：错误回调

LiveRoom 接口详情

1.setLiveRoomListener

- 接口定义：void setLiveRoomListener(ILiveRoomListener listener)
- 接口说明：设置 ILiveRoomListener 回调 ，具体回调函数请参考 ILiveRoomListener 的接口说明
- 参数说明：

参数	类型	说明
listener	ILiveRoomListener	liveroom 回调接口

- 示例代码：

```
mLiveRoom.setLiveRoomListener(new ILiveRoomListener() {
@Override
void onPusherJoin(PusherInfo pusherInfo) {
// ...
}

@Override
void onPusherQuit(PusherInfo pusherInfo) {
// ...
}

.....
});
```

2.login

- 接口定义：void login(String serverDomain, final LoginInfo loginInfo, final LoginCallback callback)
- 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。
callback	LoginCallback	登录成功与否的回调

- 示例代码：

```
final String DOMAIN = "https://room.qcloud.com/weapp/live_room ";
LoginInfo loginInfo = new LoginInfo();
loginInfo.sdkAppID = sdkAppID;
loginInfo.userID = userID;
loginInfo.userSig = userSig;
loginInfo.accType = accType;
loginInfo.userName = userName;
loginInfo.userAvatar = userAvatar;
mLiveRoom.login(DOMAIN, loginInfo, new LiveRoom.LoginCallback() {
```

```
@Override
public void onError(int errCode, String errInfo) {
//
}

@Override
public void onSuccess(String userId) {
//登录成功,返回userId
}
});
```

3.logout

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

```
mLiveRoom.logout();
```

4.getRoomList

- 接口定义：void getRoomList(int index, int count, GetRoomListCallback callback)
- 接口说明：拉取房间列表，index 和 count 两个参数用于做分页处理，表示：从序号 index 的房间开始拉取 count 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
count	int	希望 RoomService 返回的房间个数
callback	GetRoomListCallback	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
mLiveRoom.getRoomList(0, 20, new LiveRoom.GetRoomListCallback() {
@Override
```

```
public void onSuccess(ArrayList<RoomInfo> data) {
//每个房间的信息请参考RoomInfo定义
}

@Override
public void onError(int errCode, String e) {
}
});
```

5.getAudienceList

- 接口定义：void getAudienceList(String roomID, final GetAudienceListCallback callback)
- 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。

6. createRoom

- 接口定义：void createRoom(final String roomID, final String roomInfo, final CreateRoomCallback cb)
- 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 CreateRoomCallback 返回给你您。
roomInfo	String	由创建者自定义。在getRoomList中返回该信息
cb	CreateRoomCallback	创建房间结果回调

- 示例代码：

```
String roomInfo = mTitle;
try {
roomInfo = new JSONObject()
.put("title", mTitle)
.put("frontcover", mCoverPicUrl)
.put("location", mLocation)
.toString();
} catch (JSONException e) {
roomInfo = mTitle;
}
mLiveRoom.createRoom("", roomInfo, new LiveRoom.CreateRoomCallback() {
```

```
@Override
public void onSuccess(String roomId) {
Log.w(TAG,String.format("创建直播间%s成功", roomId));
}

@Override
public void onError(int errCode, String e) {
Log.w(TAG,String.format("创建直播间错误, code=%s,error=%s", errCode, e));
}
});
```

7. enterRoom

- 接口定义：void enterRoom(String roomID, TXCloudVideoView videoView, EnterRoomCallback cb)
- 接口说明：（观众）进入直播间。
- 示例代码：

```
mLiveRoom.enterRoom(mGroupId, mTXCloudVideoView, new LiveRoom.EnterRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "enter room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "enter room success ");
}
});
```

8. exitRoom

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

```
mLiveRoom.exitRoom(new LiveRoom.ExitRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "exit room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "exit room success ");
}
});
```

9. startLocalPreview

- 接口定义：void startLocalPreview(TXCloudVideoView 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：

```
TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mLiveRoom.startLocalPreview(mCaptureView);
```

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：

```
mLiveRoom.stopLocalPreview();
```

11.requestJoinPusher

- 接口定义：void requestJoinPusher(int timeout, RequestJoinPusherCallback callback)
- 接口说明：（观众）请求和主播连麦时调用该接口。
- 示例代码：

```
mLiveRoom.requestJoinPusher(10, new LiveRoom.RequestJoinPusherCallback() {
@Override
public void onAccept() {
mLiveRoom.startLocalPreview(videoView);
```

```
mLiveRoom.setPauseImage(BitmapFactory.decodeResource(getResources(), R.drawable.pause_publis
h));
mLiveRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
mLiveRoom.joinPusher(new LiveRoom.JoinPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {
mLiveRoom.startLocalPreview(videoView);
Toast.makeText(LivePlayActivity.this, "连麦失败：" + errInfo, Toast.LENGTH_SHORT).show();
}

@Override
public void onSuccess() {

}
});
}

@Override
public void onReject(String reason) {
Toast.makeText(LivePlayActivity.this, "主播拒绝你的连麦请求", Toast.LENGTH_SHORT).show();
}

@Override
public void onTimeOut() {
Toast.makeText(LivePlayActivity.this, "连麦请求超时，主播没有做出回应", Toast.LENGTH_SHORT).show();
}

@Override
public void onError(int code, String errInfo) {
Toast.makeText(LivePlayActivity.this, "连麦请求失败：" + errInfo, Toast.LENGTH_SHORT).show();
}
});
```

12.joinPusher

- 接口定义：void joinPusher(final JoinPusherCallback cb)
- 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 JoinPusherCallback 的回调结果。
- 示例代码：

```
mLiveRoom.joinPusher(new LiveRoom.JoinPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {
mLiveRoom.startLocalPreview(videoView);
Toast.makeText(LivePlayActivity.this, "连麦失败：" + errInfo, Toast.LENGTH_SHORT).show();
```

```
}

@Override
public void onSuccess() {

}
});
```

13.quitPusher

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

```
mLiveRoom.quitPusher(new LiveRoom.QuitPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess() {

}
});
```

14.acceptJoinPusher

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

```
mLiveRoom.acceptJoinPusher(userId);
```

15.rejectJoinPusher

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

```
mLiveRoom.rejectJoinPusher(userId, "");
```

16.kickoutSubPusher

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

```
mLiveRoom.kickoutSubPusher(userId);
```

17.getOnlinePusherList

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

```
mLiveRoom.getOnlinePusherList(new LiveRoom.GetOnlinePusherListCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess(final ArrayList<PusherInfo> pusherList) {

}
});
```

18.startPlayPKStream

- 接口定义：void startPlayPKStream(final String playUrl, TXCloudVideoView videoView, final PKStreamPlayCallback callback)
- 接口说明：主播PK：开始播放对方的流
- 示例代码：

```
mLiveRoom.startPlayPKStream(streamUrl, videoView, new LiveRoom.PKStreamPlayCallback() {
@Override
public void onPlayBegin() {

}

@Override
```

```
public void onPlayError() {

}
});
```

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：

```
mLiveRoom.stopPlayPKStream()
```

20.sendPKRequest

- 接口定义：void sendPKRequest(String userID, int timeout, final RequestPKCallback callback)
- 接口说明：主播PK：发送PK请求
- 示例代码：

```
mLiveRoom.sendPKRequest(userID, 10, new LiveRoom.RequestPKCallback() {
@Override
public void onAccept(String streamUrl) {

}

@Override
public void onReject(String reason) {

}

@Override
public void onTimeOut() {

}

@Override
public void onError(int code, String errInfo) {

}
});
```

21.sendPKFinishRequest

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

```
mLiveRoom.sendPKFinishRequest(userID)
```

22.acceptPKRequest

- 接口定义：void acceptPKRequest(String userID)
- 接口说明：主播PK：授受PK请求
- 示例代码：

```
mLiveRoom.acceptPKRequest(userID);
```

23.rejectPKRequest

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

```
mLiveRoom.rejectPKRequest(userID, "");
```

24.addRemoteView

- 接口定义：void addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)
- 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：

```
public void onPusherJoin(PusherInfo pusherInfo) {
......
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
}

@Override
```

```
public void onPlayError() {
}
});
.....
}
```

25.deleteRemoteView

- 接口定义：void deleteRemoteView(final PusherInfo pusherInfo)
- 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：

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mLiveRoom.deleteRemoteView(pusherInfo);
.....
}
```

26.sendRoomTextMsg

- 接口定义：void sendRoomTextMsg(@NonNull String message, final SendTextMessageCallback callback)
- 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：

```
mLiveRoom.sendRoomTextMsg("hello", new LiveRoom.SendTextMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.d(TAG, "sendRoomTextMsg error:");
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomTextMsg success:");
}
});
```

27.sendRoomCustomMsg

- 接口定义：void sendRoomCustomMsg(@NonNull String cmd, @NonNull String message, final SendCustomMessageCallback callback)

- 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：

```
mLiveRoom.sendRoomCustomMsg(String.valueOf(TCConstants.IMCMD_DANMU),
 “hello ” , new LiveRoom.SendCustomMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.w(TAG, "sendRoomDanmuMsg error: "+errInfo);
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomDanmuMsg success");
}
});
```

28.startScreenCapture

- 接口定义：void startScreenCapture()
- 接口说明：启动屏幕录制。由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出一个提示，旨在告诫用户：“有 App 要截取您屏幕上的所有内容”。

> 注意：该接口在Android API 21接口才生效。录屏接口和摄像头预览（startCameraPreview）互斥，同时只能由一个生效

29.stopScreenCapture

- 接口定义：void stopScreenCapture()
- 接口说明：停止屏幕录制。

30.switchToBackground

- 接口定义：void switchToBackground()
- 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

31.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。

32.setBeautyFilter

- 接口定义：boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel)
- 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
style	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
whiteningLevel	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
ruddyLevel	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：

```
mLiveRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

33.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(TXCloudVideoView) 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。

34.setMute

- 接口定义：void setMute(mute)
- 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
mute	boolean	是否静音

35.setMirror

- 接口定义：void setMirror(boolean enable)
- 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
enable	boolean	true 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
mLiveRoom.setMirror(true);
```

36.playBGM

- 接口定义：boolean playBGM(String 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	String	背景音乐文件位于手机中的绝对路径

37.stopBGM

- 接口定义：boolean stopBGM()
- 接口说明：停止播放背景音乐。返回结果中，true 表示停止播放成功，false 表示停止播放失败。

38.pauseBGM

- 接口定义：boolean pauseBGM()
- 接口说明：暂停播放背景音乐。返回结果中，true 表示暂停播放成功，false 表示暂停播放失败。

39.resumeBGM

- 接口定义：boolean resumeBGM()
- 接口说明：恢复播放背景音乐。返回结果中，true 表示恢复播放成功，false 表示恢复播放失败。

40.setMicVolume

- 接口定义：boolean setMicVolume(float x)
- 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
x	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

41.setBGMVolume

- 接口定义：boolean setBGMVolume(float x)
- 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
x	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

42.getMusicDuration

- 接口定义：int getMusicDuration(String path)
- 接口说明：获取背景音乐时长。返回结果单位为毫秒。
- 参数说明：

参数	类型	说明
path	String	path == null 获取当前播放歌曲时长；path != null 获取path路径歌曲时长

43.startRecord

- 接口定义： int startRecord(int recordType)
- 接口说明：开始录制视频。该接口用于观众端将观看的视频实时保存到本地文件。
- 特别注意：该接口需要在 enterRoom 成功后调用，另外生成的视频文件由您的应用层代码负责管理，SDK不做清理。
- 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 表示还未开始推流，这次启动录制失败。
- 参数说明：

参数	类型	说明
recordType	int	录制类型，目前只支持纯视频录制 TXRecordCommon.RECORD_TYPE_STREAM_SOURCE

- 示例代码：

```
mLiveRoom.startRecord(TXRecordCommon.RECORD_TYPE_STREAM_SOURCE);
```

44.stopRecord

- 接口定义： int 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。

45.setVideoRecordListener

- 接口定义： void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)
- 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
listener	TXRecordCommon.ITXVideoRecordListener	视频录制回调

- 示例代码：

```
mLiveRoom.setVideoRecordListener(new TXRecordCommon.ITXVideoRecordListener(){
@Override
public void onRecordEvent(int event, Bundle param) {
}

@Override
public void onRecordProgress(long milliSecond) {
Log.d(TAG, "record progress:" + milliSecond);
}

@Override
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode == TXRecordCommon.RECORD_RESULT_OK) {
String videoFile = result.videoPath;
String videoCoverPic = result.coverPath;
} else {
Log.d(TAG, "record error:" + result.retCode + ", error msg:" + result.descMsg);
}
}
});
```

46.incCustomInfo

- 接口定义： void incCustomInfo(String fieldName, int count)
- 接口说明：增加自定义fieldName统计值。该接口用于统计房间的点赞，礼物等总数。最终的累计值可以通过房间信息roominfo的custom字段获得
- 参数说明：

参数	类型	说明
fieldName	String	需要统计的字段名称
count	int	一次统计增加的值，一般一次增加1

- 示例代码：

```
mLiveRoom.incCustomInfo(“praise”,1);
```

47.decCustomInfo

- 接口定义： void decCustomInfo(String fieldName, int count)
- 接口说明：减少自定义fieldName统计值。该接口用于统计房间的点赞，礼物等总数。最终的累计值可以通过房间信息roominfo的custom字段获得
- 参数说明：

参数	类型	说明
fieldName	String	需要统计的字段名称
count	int	一次统计减少的值，一般一次减少1

48. updateSelfUserInfo

- 接口定义： void updateSelfUserInfo(String userName, String userAvatar)
- 接口说明：更新用户的昵称和头像信息。该接口主要用于用户修改昵称或者头像后实时更新liveroom的信息

49.setPauseImage

- 接口定义：void setPauseImage(Bitmap bitmap)
- 接口说明：设置从前台切换到后台时，推送的图片。
- 参数说明：

参数	类型	说明
bitmap	Bitmap	背景图片bitmap

ILiveRoomListener 接口详情

1. onGetPusherList

- 接口定义：void onGetPusherList(List pusherList)
- 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：

```
public void onGetPusherList(List<PusherInfo> pusherList) {
.....
for (PusherInfo pusherInfo : pusherInfoList) {
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
}
.....
}
```

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：

```
public void onPusherJoin(final PusherInfo pusherInfo) {
.....
mLiveRoom.addRemoteView(videoView, pusherInfo, new LiveRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
.....
}
```

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：

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mLiveRoom.deleteRemoteView(pusherInfo);
.....
}
```

4. onRecvJoinPusherRequest

- 接口定义：void onRecvJoinPusherRequest(String userID, String userName, String userAvatar)
- 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：

```
public void onRecvJoinPusherRequest(final String userId, String userName, String userAvatar) {
final AlertDialog.Builder builder = new AlertDialog.Builder(mActivity)
.setCancelable(true)
.setTitle("提示")
.setMessage(userName + "向您发起连麦请求")
```

```
.setPositiveButton("接受", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.acceptJoinPusher(userId);
dialog.dismiss();
}
})
.setNegativeButton("拒绝", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.rejectJoinPusher(userId, "主播拒绝了您的连麦请求");
dialog.dismiss();
}
});
}
```

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：

```
public void onKickOut() {
......
mLiveRoom.stopLocalPreview();
mLiveRoom.quitPusher(new LiveRoom.QuitPusherCallback() {
@Override
public void onError(int errCode, String errInfo) {

}

@Override
public void onSuccess() {

}
});
......
}
```

6. onRecvPKRequest

- 接口定义：void onRecvPKRequest(String userID, String userName, String userAvatar, String 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：

```
@Override
public void onRecvPKRequest(final String userID, final String userName, final String userAvatar, fina
l String streamUrl){
final AlertDialog.Builder builder = new AlertDialog.Builder(mActivity)
.setCancelable(true)
.setTitle("提示")
.setMessage(userName + "向您发起PK请求")
.setPositiveButton("接受", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.acceptPKRequest(userID);
mLiveRoom.startPlayPKStream(streamUrl, videoView, new LiveRoom.PKStreamPlayCallback() {
@Override
public void onPlayBegin() {

}

@Override
public void onPlayError() {

}
});
dialog.dismiss();
}
})
.setNegativeButton("拒绝", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
mLiveRoom.rejectPKRequest(userID, "主播拒绝了您的PK请求");
dialog.dismiss();
}
});

final AlertDialog alertDialog = builder.create();
alertDialog.setCancelable(false);
alertDialog.setCanceledOnTouchOutside(false);
alertDialog.show();
}
```

7. onRecvPKFinishRequest

- 接口定义：void onRecvPKFinishRequest(String 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：

```
@Override
public void onRecvPKFinishRequest(final String userID){
mLiveRoom.stopPlayPKStream();
}
```

8. onRecvRoomTextMsg

- 接口定义：void onRecvRoomTextMsg(String roomID, String userID, String userName, String userAvatar, String message)
- 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：

```
public void onRecvRoomTextMsg(String roomid, String userid, String userName, String userAvatar, String message) {
//do nothing
}
```

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。

10. onRoomClosed

- 接口定义：void onRoomClosed(String 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：

```
public void onRoomClosed(String roomId) {
.....
mLiveRoom.exitRoom(new LiveRoom.ExitRoomCallback() {
@Override
public void onSuccess() {
```

```
Log.i(TAG, "exitRoom Success");
}

@Override
public void onError(int errCode, String e) {
Log.e(TAG, "exitRoom failed, errorCode = " + errCode + " errMessage = " + e);
}
});
.....
}
```

11. onDebugLog

- 接口定义：void onDebugLog(String log)
- 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：

```
public void onDebugLog(String line) {
Log.i(TAG,line);
}
```

12. onError

- 接口定义：void onError(int errorCode, String errorMessage)
- 接口说明：直播间错误回调
- 示例代码：

```
public void onError(final int errorCode, final String errorMessage) {
mLiveRoom.exitRoom(null);
new AlertDialog.Builder(mActivity)
.setTitle("直播间错误")
.setMessage(errorMessage + "[" + errorCode + "]")
.setNegativeButton("确定", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

}
}).show();
}
```

视频通话（RTCRoom）

最近更新时间：2018-07-23 10:57:25

RTCRoom

名称	描述
setRTCRoomListener(IRTCRoomListener listener)	设置rtcroom回调
login(serverDomain, loginInfo, loginCallback)	登录到RoomService后台
logout()	从RoomService后台登出
getRoomList(int index, int count, GetRoomListCallback callback)	获取房间列表（非必须，可选择使用您自己的房间列表）
createRoom(String roomID, String roomInfo, CreateRoomCallback cb)	会议创建者：创建房间（roomID 可不填）
enterRoom(String roomID, EnterRoomCallback cb)	会议参与者：进入房间
exitRoom(ExitRoomCallback callback)	会议创建者 OR 会议参与者：退出房间
startLocalPreview(TXCloudVideoView videoView)	会议创建者 OR 会议参与者：开启摄像头预览
stopLocalPreview()	停止摄像头预览
addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)	播放会议参与者的远程视频画面
deleteRemoteView(PusherInfo pusherInfo)	停止播放会议参与者的远程视频画面
sendRoomTextMsg(String message, SendTextMessageCallback callback)	发送文本（弹幕）消息
sendRoomCustomMsg(String cmd, String message, SendCustomMessageCallback callback)	发送自定义格式的消息（点赞，送花）
switchToBackground()	App 从前台切换到后台
switchToForeground()	App 从后台切换到前台

名称	描述
setBeautyFilter(style, beautyLevel, whiteningLevel, ruddyLevel)	设置美颜
switchCamera()	切换前后置摄像头，支持在推流中动态切换
setMute(mute)	静音
setMirror(enable)	画面镜像（此接口仅影响观众端效果，推流端一直保持镜像效果）
playBGM(String path)	开始播放背景音乐（path 指定音乐文件路径）
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(x)	设置混音时麦克风的音量大小
setBGMVolume(x)	设置混音时背景音乐的音量大小
getMusicDuration(fileName)	获取背景音乐时长
setBitrateRange(minBitrate, maxBitrate)	设置视频码率范围
setPauseImage(bitmap)	设置后台时推送的图片

IRTCRoomListener

名称	描述
onGetPusherList(pusherList)	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）
onPusherJoin(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）
onPusherQuit(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）

名称	描述
onRecvRoomTextMsg(roomID, userID, userName, userAvatar, message)	聊天室：收到文本消息
onRecvRoomCustomMsg(roomID, userID, userName, userAvatar, cmd, message)	聊天室：收到自定义消息
onRoomClosed(roomID)	通知：房间解散通知
onDebugLog(log)	LOG：日志回调
onError(errorCode, errorMessage)	ERROR：错误回调

RTCRoom 接口详情

1.setRTCRoomListener

- 接口定义：void setRTCRoomListener(IRTCRoomListener listener)
- 接口说明：设置 IRTCRoomListener 回调 ，具体回调函数请参考 IRTCRoomListener 的接口说明
- 参数说明：

参数	类型	说明
listener	IRTCRoomListener	rtcroom 回调接口

- 示例代码：

```
mRTCRoom.setRTCRoomListener(new IRTCRoomListener() {
@Override
void onPusherJoin(PusherInfo pusherInfo) {
// ...
}

@Override
void onPusherQuit(PusherInfo pusherInfo) {
// ...
}

.....
});
```

2.login

- 接口定义：void login(String serverDomain, final LoginInfo loginInfo, final LoginCallback callback)
- 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。
callback	LoginCallback	登录成功与否的回调

- 示例代码：

```
final String DOMAIN = "https://room.qcloud.com/weapp/rtc_room ";
LoginInfo loginInfo = new LoginInfo();
loginInfo.sdkAppID = sdkAppID;
loginInfo.userID = userID;
loginInfo.userSig = userSig;
loginInfo.accType = accType;
loginInfo.userName = userName;
loginInfo.userAvatar = userAvatar;
mRTCRoom.login(DOMAIN, loginInfo, new RTCRoom.LoginCallback() {
@Override
public void onError(int errCode, String errInfo) {
//
}

@Override
public void onSuccess(String userId) {
//登录成功,返回userId
}
});
```

3.logout

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

```
mRTCRoom.logout();
```

4.getRoomList

- 接口定义：void getRoomList(int index, int count, GetRoomListCallback callback)
- 接口说明：拉取房间列表，index 和 count 两个参数用于做分页处理，表示：从序号 index 的房间开始拉取 count 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
count	int	希望 RoomService 返回的房间个数
callback	GetRoomListCallback	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
mRTCRoom.getRoomList(0, 20, new RTCRoom.GetRoomListCallback() {
@Override
public void onSuccess(ArrayList<RoomInfo> data) {
//每个房间的信息请参考RoomInfo定义
}

@Override
public void onError(int errCode, String e) {
}
});
```

5. createRoom

- 接口定义：void createRoom(final String roomID, final String roomInfo, final CreateRoomCallback cb)
- 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

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 CreateRoomCallback 返回给你您。
roomInfo	String	由创建者自定义。在getRoomList中返回该信息
cb	CreateRoomCallback	创建房间结果回调

- 示例代码：

```
String roomInfo = mTitle;
try {
roomInfo = new JSONObject()
.put("title", mTitle)
.put("frontcover", mCoverPicUrl)
.put("location", mLocation)
.toString();
} catch (JSONException e) {
roomInfo = mTitle;
}
mRTCRoom.createRoom("", roomInfo, new RTCRoom.CreateRoomCallback() {
@Override
public void onSuccess(String roomId) {
Log.w(TAG,String.format("创建直播间%s成功", roomId));
}

@Override
public void onError(int errCode, String e) {
Log.w(TAG,String.format("创建直播间错误, code=%s,error=%s", errCode, e));
}
});
```

6. enterRoom

- 接口定义：void enterRoom(String roomID, EnterRoomCallback cb)
- 接口说明：（会议参与者）进入直播间。
- 示例代码：

```
mRTCRoom.enterRoom(mRoomId, new RTCRoom.EnterRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "enter room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "enter room success ");
}
});
```

7. exitRoom

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

```
mRTCRoom.exitRoom(new RTCRoom.ExitRoomCallback() {
@Override
public void onError(int errCode, String errInfo) {
TXLog.w(TAG, "exit room error : "+errInfo);
}

@Override
public void onSuccess() {
TXCLog.d(TAG, "exit room success ");
}
});
```

8. startLocalPreview

- 接口定义：void startLocalPreview(TXCloudVideoView 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：

```
TXCloudVideoView mCaptureView = (TXCloudVideoView) view.findViewById(R.id.video_view);
mRTCRoom.startLocalPreview(mCaptureView);
```

9. stopLocalPreview

- 接口定义：void stopLocalPreview(boolean isNeedClearLastImg)
- 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：

```
mRTCRoom.stopLocalPreview();
```

10.addRemoteView

- 接口定义：void addRemoteView(TXCloudVideoView videoView, PusherInfo pusherInfo, RemoteViewPlayCallback callback)
- 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：

```
public void onPusherJoin(PusherInfo pusherInfo) {
......
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
}

@Override
public void onPlayError() {
}
});
......
}
```

11.deleteRemoteView

- 接口定义：void deleteRemoteView(final PusherInfo pusherInfo)
- 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：

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mRTCRoom.deleteRemoteView(pusherInfo);
.....
}
```

12.sendRoomTextMsg

- 接口定义：void sendRoomTextMsg(@NonNull String message, final SendTextMessageCallback callback)
- 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：

```
mRTCRoom.sendRoomTextMsg("hello", new RTCRoom.SendTextMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.d(TAG, "sendRoomTextMsg error:");
}

@Override
public void onSuccess() {
Log.d(TAG, "sendRoomTextMsg success:");
}
});
```

13.sendRoomCustomMsg

- 接口定义：void sendRoomCustomMsg(@NonNull String cmd, @NonNull String message, final SendCustomMessageCallback callback)
- 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：

```
mRTCRoom.sendRoomCustomMsg(String.valueOf(TCConstants.IMCMD_DANMU),
 “hello ” , new RTCRoom.SendCustomMessageCallback() {
@Override
public void onError(int errCode, String errInfo) {
Log.w(TAG, "sendRoomDanmuMsg error: "+errInfo);
}

@Override
public void onSuccess() {
```

```
Log.d(TAG, "sendRoomDanmuMsg success");
}
});
```

14.switchToBackground

- 接口定义：void switchToBackground()
- 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

15.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。

16.setBeautyFilter

- 接口定义：boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel)
- 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
style	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
whiteningLevel	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
ruddyLevel	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：

```
mRTCRoom.setBeautyFilter(mBeautyStyle, mBeautyLevel, mWhiteningLevel, mRuddyLevel);
```

17.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(TXCloudVideoView) 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。

18.setMute

- 接口定义：void setMute(mute)
- 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
mute	boolean	是否静音

19.setMirror

- 接口定义：void setMirror(boolean enable)
- 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，不影响推流推流端。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
enable	boolean	true 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
mRTCRoom.setMirror(true);
```

20.playBGM

- 接口定义：boolean playBGM(String 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	String	背景音乐文件位于手机中的绝对路径

21.stopBGM

- 接口定义：boolean stopBGM()
- 接口说明：停止播放背景音乐。返回结果中，true 表示停止播放成功，false 表示停止播放失败。

22.pauseBGM

- 接口定义：boolean pauseBGM()
- 接口说明：暂停播放背景音乐。返回结果中，true 表示暂停播放成功，false 表示暂停播放失败。

23.resumeBGM

- 接口定义：boolean resumeBGM()
- 接口说明：恢复播放背景音乐。返回结果中，true 表示恢复播放成功，false 表示恢复播放失败。

24.setMicVolume

- 接口定义：boolean setMicVolume(float x)
- 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
x	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

25.setBGMVolume

- 接口定义：boolean setBGMVolume(float x)
- 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
x	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

26.getMusicDuration

- 接口定义：int getMusicDuration(String path)
- 接口说明：获取背景音乐时长。返回结果单位为毫秒。
- 参数说明：

参数	类型	说明

参数	类型	说明
path	String	path == null 获取当前播放歌曲时长；path != null 获取path路径歌曲时长

27.setBitrateRange

- 接口定义：void setBitrateRange(int minBitrate, int maxBitrate)
- 接口说明：设置视频的码率区间。双人一般设为400到800；多人一般设为200到400
- 参数说明：

参数	类型	说明
minBitrate	int	最小码率
maxBitrate	int	最大码率

28.setPauseImage

- 接口定义：void setPauseImage(Bitmap bitmap)
- 接口说明：设置从前台切换到后台时，推送的图片。
- 参数说明：

参数	类型	说明
bitmap	Bitmap	背景图片bitmap

IRTCRoomListener 接口详情

1. onGetPusherList

- 接口定义：void onGetPusherList(List\ pusherList)

- 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：

```
public void onGetPusherList(List<PusherInfo> pusherInfoList) {
for (PusherInfo pusherInfo : pusherInfoList) {
.....
```

```
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
}
}
```

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：

```
public void onPusherJoin(final PusherInfo pusherInfo) {
......
mRTCRoom.addRemoteView(videoView, pusherInfo, new RTCRoom.RemoteViewPlayCallback() {
@Override
public void onPlayBegin() {
//
}

@Override
public void onPlayError() {

}
});
......
}
```

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：

```
public void onPusherQuit(PusherInfo pusherInfo) {
.....
mRTCRoom.deleteRemoteView(pusherInfo);
.....
}
```

4. onRecvRoomTextMsg

- 接口定义：void onRecvRoomTextMsg(String roomID, String userID, String userName, String userAvatar, String message)
- 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：

```
public void onRecvRoomTextMsg(String roomid, String userid, String userName, String userAvatar, String message) {
//do nothing
}
```

5. onRecvRoomCustomMsg

- 接口定义：void onRecvRoomCustomMsg(String roomID, String userID, String userName, String userAvatar, String cmd, String message)
- 接口说明：当会议参与者调用sendRoomCustomMsg时，房间内的其他会议参与者都会收到该通知。

6. onRoomClosed

- 接口定义：void onRoomClosed(String 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：

```
public void onRoomClosed(String roomId) {
.....
mRTCRoom.exitRoom(new RTCRoom.ExitRoomCallback() {
@Override
public void onSuccess() {
Log.i(TAG, "exitRoom Success");
```

```
}

@Override
public void onError(int errCode, String e) {
Log.e(TAG, "exitRoom failed, errorCode = " + errCode + " errMessage = " + e);
}
});
.....
}
```

7. onDebugLog

- 接口定义：void onDebugLog(String log)
- 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：

```
public void onDebugLog(String line) {
Log.i(TAG,line);
}
```

8. onError

- 接口定义：void onError(int errorCode, String errorMessage)
- 接口说明：直播间错误回调
- 示例代码：

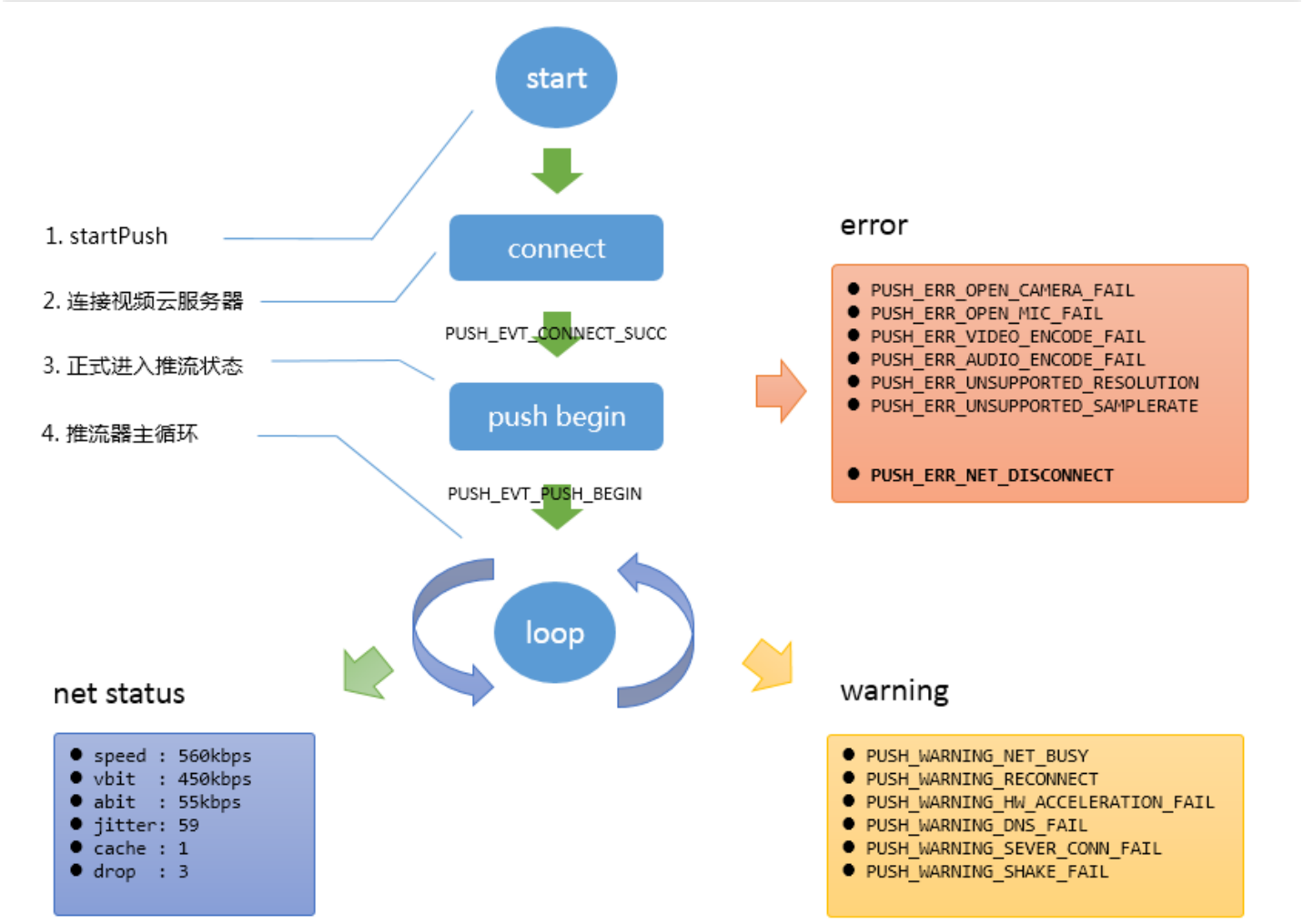
```
public void onError(final int errorCode, final String errorMessage) {
mRTCRoom.exitRoom(null);
new AlertDialog.Builder(mActivity)
.setTitle("直播间错误")
.setMessage(errorMessage + "[" + errorCode + "]")
.setNegativeButton("确定", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

}
}).show();
}
```

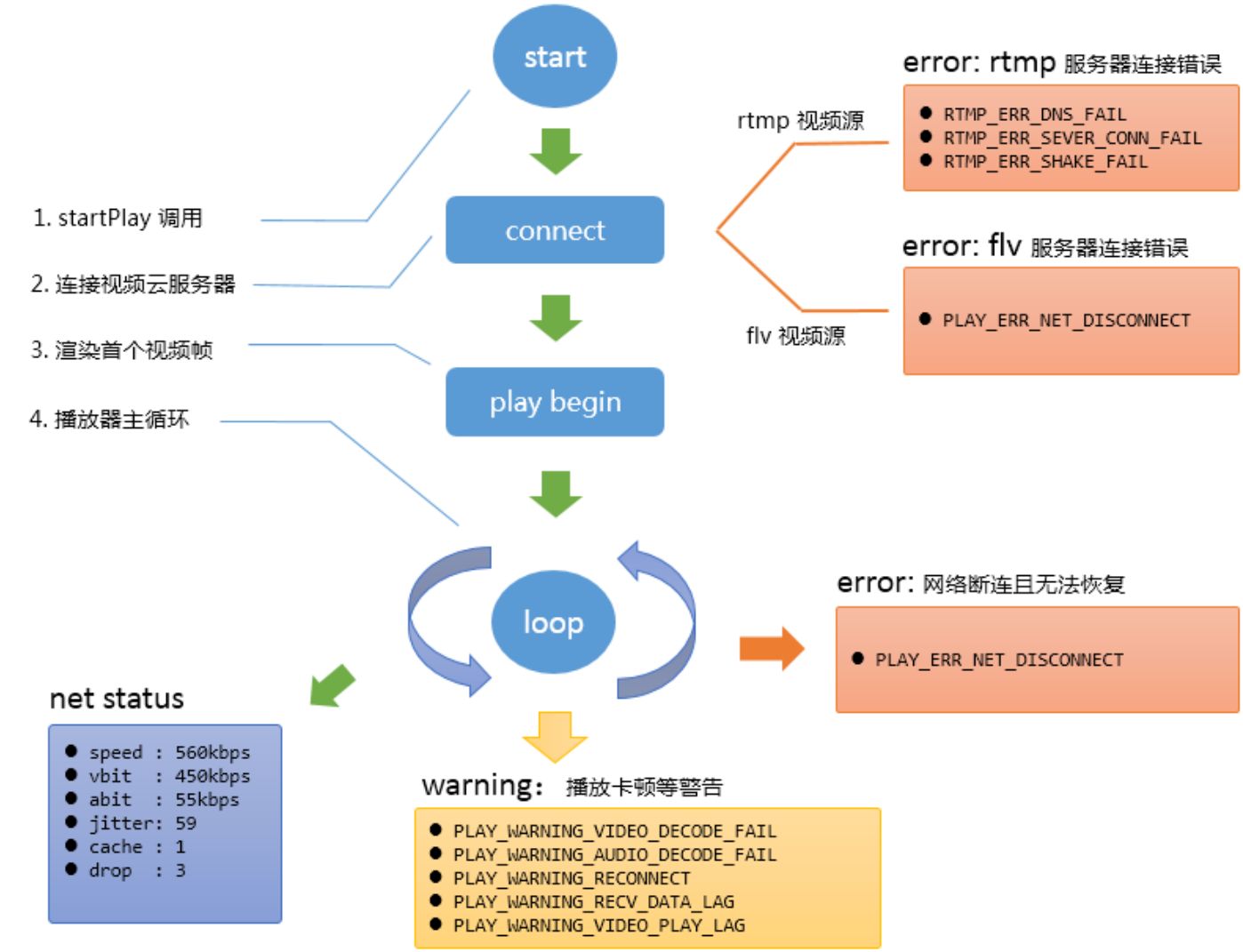
进阶功能
SDK内部原理

最近更新时间：2017-06-24 19:55:13

TXLivePusher



TXLivePlayer


start
connect
play begin
loop
1. startPlay 调用
2. 连接视频云服务器
3. 渲染首个视频帧
4. 播放器主循环
rtmp 视频源
flv 视频源
error: rtmp 服务器连接错误
RTMP_ERR_DNS_FAIL
RTMP_ERR_SEVER_CONN_FAIL
RTMP_ERR_SHAKE_FAIL
error: flv 服务器连接错误
PLAY_ERR_NET_DISCONNECT
error: 网络断连且无法恢复
PLAY_ERR_NET_DISCONNECT
net status
speed : 560kbps
vbit : 450kbps
abit : 55kbps
jitter: 59
cache : 1
drop : 3
warning: 播放卡顿等警告
PLAY_WARNING_VIDEO_DECODE_FAIL
PLAY_WARNING_AUDIO_DECODE_FAIL
PLAY_WARNING_RECONNECT
PLAY_WARNING_RECV_DATA_LAG
PLAY_WARNING_VIDEO_PLAY_LAG

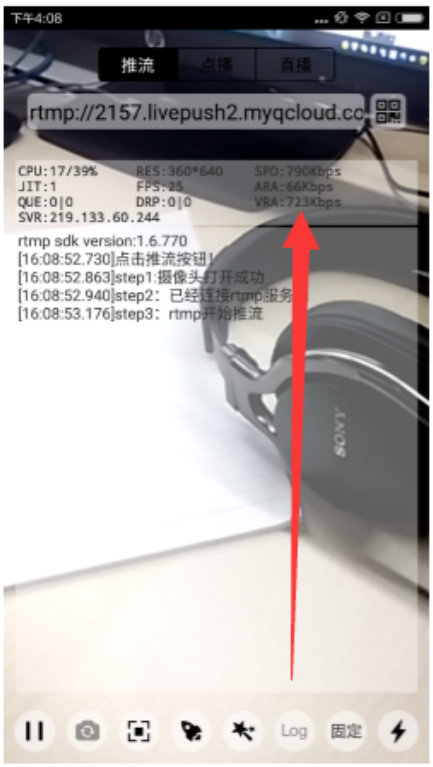
SDK指标监控

最近更新时间：2017-06-24 19:56:04

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	CPU使用率	17% /39%	App: 17% Sys: 39%
RES	推流分辨率	360 * 640	分辨率 360 * 640
SPD	网络上传速度	790kbps	每秒上传790kb数据
FPS	视频帧率	25帧/秒	每秒25帧画面
ARA	音频码率	66kbps	每秒编码出66kb音频数据
VRA	视频码率	723kpbs	每秒编码出723kb音频数据
QUE	缓冲积压	0 | 0	主播端视频|音频积压情况
DRP	主动丢包	0 | 0	主播端视频|音频丢包情况

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）

推流状态	含义说明
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
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。

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。

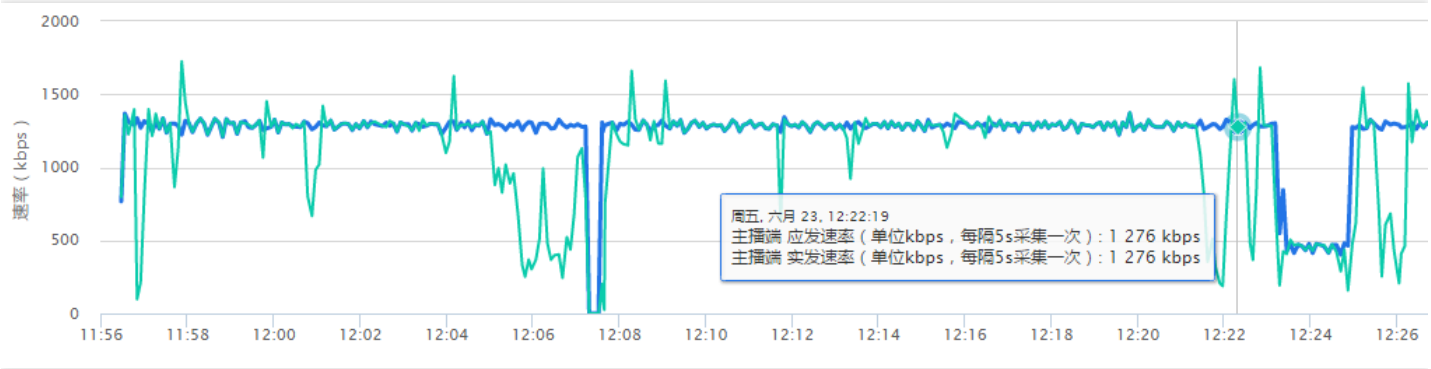
SERVER_IP

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。

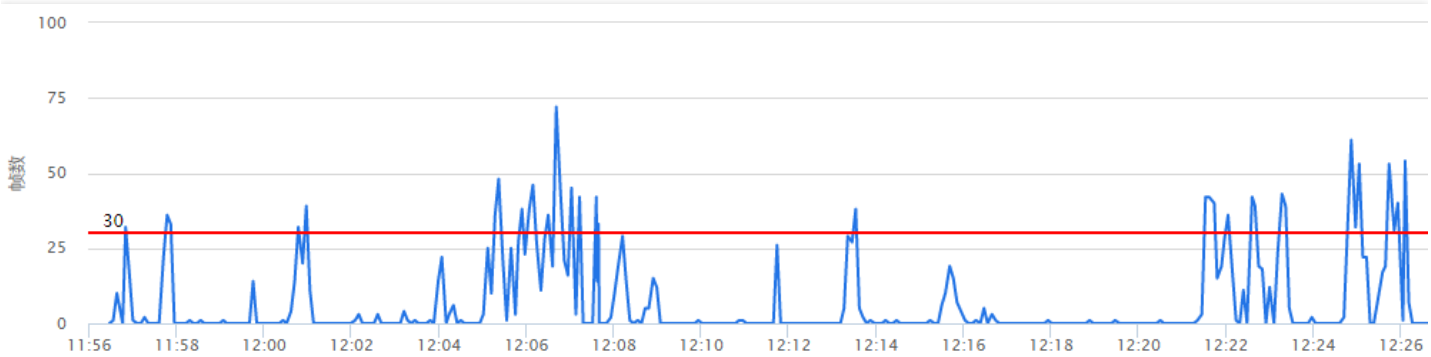
3. 如何看懂腾讯云推流图表？

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：

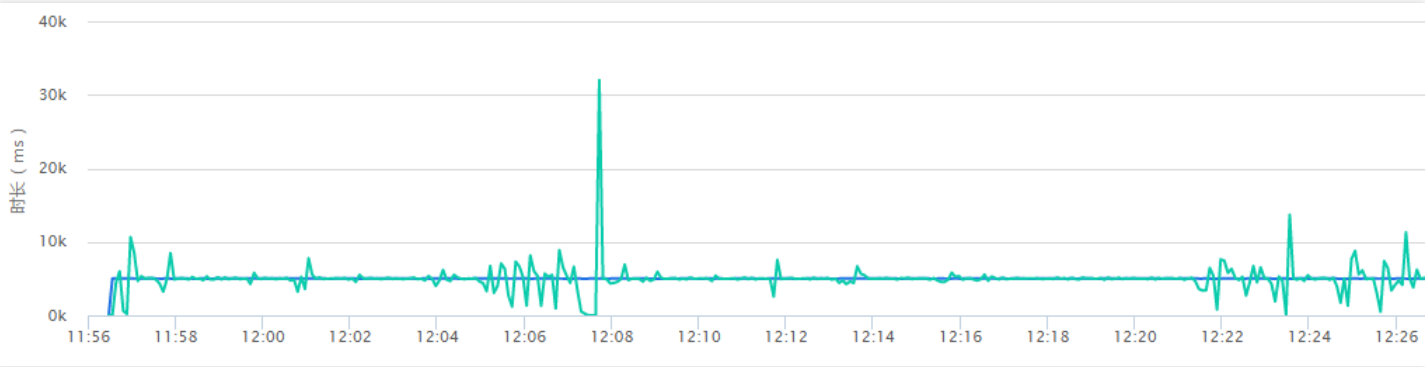
- **主播端-应发速率-实发速率曲线图**
  蓝色曲线代表 BITRATE 的统计曲线，即 SDK 产生的音视频数据，即绿色曲线发表实际网络发出去多少。两条线重合度越高表示推流质量越好。



- **主播端-音视频数据堆积情况**
  - 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。
  - 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；
  - 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（数据源来自 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

推流状态	参数名称	示例指标	指标含义说明
CPU	CPU使用率	7.8% /16%	App: 7.8% Sys: 16%
RES	推流分辨率	960*540	分辨率 960* 540
SPD	网络下载速度	1073kbps	每秒下载1073kb数据
FPS	视频帧率	30帧/秒	每秒30帧画面
ARA	音频码率	72kbps	每秒需要解码多少音频数据
VRA	视频码率	993kpbs	每秒需要解码多少视频数据
QUE	播放器缓冲区	213 | 35	213ms音频缓冲，6帧视频缓冲
SVR	服务器IP	183.3.255.175	当前播放器连接的服务器 IP

播放状态	含义说明
NET_STATUS_CPU_USAGE	当前瞬时CPU使用率

播放状态	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：2017-06-24 20:16:41

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。

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	● 270*480 ● 360*640	360*640
HIGH	300kbps – 1500kbps	1200kbps	● 270*480 ● 360*640 ● 540*960	540*960
SUPER	300kbps – 1800kbps	1800kbps	● 270*480 ● 360*640 ● 540*960 ● 720*1280	720*1280
VIDEOCHAT	200kbps – 800kbps	不支持关闭	● 192*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：2017-12-19 18:07:36

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, 该设置项才能启作用	1200
videoBitrateMin	最小输出码率：只有开启自适应码率, 该设置项才能启作用	800
videoEncodeGop	关键帧间隔（单位：秒）即多少秒出一个I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 mLivePushConfig 和 mLivePusher
....
//初始化 mLivePushConfig
mLivePushConfig = new TXLivePushConfig();

// 修改参数设置为声音44100采样率，视频800固定码率
mLivePushConfig.setAudioSampleRate(44100);
mLivePushConfig.setAutoAdjustBitrate(false);
mLivePushConfig.setVideoBitrate(800);

//初始化 mLivePusher
mLivePusher.setConfig(mLivePushConfig);
```

定制视频数据

最近更新时间：2018-04-20 17:52:53

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**
  通过 **TXLivePusher** 的 **setVideoProcessListener** 接口设置自定义视频处理回调

```
public interface VideoCustomProcessListener {

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

/**
* 在OpenGL线程中回调，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GLES20.GL_TEXTURE_2D，接口返回给SDK的纹理类型也必须是GLES20.GL_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height);

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

- **在回调函数中对视频数据进行加工**
  实现 VideoCustomProcessListener 的 onTextureCustomProcess() 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 onTextureCustomProcess 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 TXLivePusher 的 startCameraPreview 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. 使用 sendCustomVideoData 向SDK填充Video数据**
  sendCustomVideoData 的作用是向 SDK 塞入您采集和处理后的视频数据，目前支持 i420 格式。
- **Step3. 使用 sendCustomPCMData 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。

定制播放数据

方案一：修改OpenGL纹理（仅适用硬解码播放）

- **step 1: 添加一个 TextureView**
  为了能够展示播放器的视频画面，您需要在布局xml文件中使用 `TextureView` 替代 `TXCloudVideoView`

```
<TextureView
android:id="@+id/video_view"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

- **step 2: 在代码中获取 TextureView 对象**

```
mTextureView = (TextureView) findViewById(R.id.video_view);
```

- **step 3: 通过 setSurface 接口绑定**
  通过 TXLivePlayer 的 setSurface(mSurface) 接口将视频数据渲染的 mTextureView 绑定到 TXLivePlayer

```
mTextureView.setSurfaceTextureListener(new TextureView.SurfaceTextureListener() {

@Override
public void onSurfaceTextureAvailable(SurfaceTexture texture, int width, int height) {
//创建 Surface
mSurface = new Surface(texture);
//设置 Surface，mSurface 和 mTextureView 通过 texture 关联在了一起，
//所以调用该接口后，解码数据自动渲染到 mTextureView 上
mLivePlayer.setSurface(mSurface);
}

.....
});
```

- **OpenGL 相关参考代码**
  如果您在调整画面尺寸遇到问题，可以参考下完整的示例代码 - 调整画面尺寸
  如果您在使用 OpenGL ES 对视频数据进行二次处理没有思路，可以参考下示例代码 - OpenGL ES 处理数据

方案二：获取 YUV 数据（仅适用软解码播放）

如果您想通过获取 SDK 解码之后的 YUV 类型的视频数据，您可以按如下步骤实现。

- 监听 PLAY_EVT_CHANGE_RESOLUTION 事件

```
public void onPlayEvent(int event, Bundle param) {
//...
if (event == TXLiveConstants.PLAY_EVT_CHANGE_RESOLUTION) {
//获取视频的宽高
int width = param.getInt(TXLiveConstants.EVT_PARAM1, 0);
int height = param.getInt(TXLiveConstants.EVT_PARAM2, 0);
if (width != 0 && height != 0 && !mHWDecode) {
//创建存储 yuv 数据的 buffer，目前输出的 yuv 格式为 I420
byte[] buf = new byte[width * height * 3 / 2];
//将 buffer 设置进 mLivePlayer
mLivePlayer.addVideoRawData(buf);
}
}
}
```

- 通过 VideoRawDataListener 获取 YUV 裸数据

```
TXVideoRawDataObserver.ITXVideoRawDataListener rawDataListener = new
TXVideoRawDataObserver.ITXVideoRawDataListener() {
@Override
public void onVideoRawDataAvailable(byte[] buf, int width, int height, int timestamp) {
//解码一帧后的数据回调，buf 中存放了 yuv 数据，格式为 I420
if (!mHWDecode) {
//如果需要继续获取yuv数据，需要重新调用addVideoRawData方法
mLivePlayer.addVideoRawData(buf);
}
}
};

mLivePlayer.setVideoRawDataListener(rawDataListener);
```