1. 概述

PLDroidRTCStreaming 是基于 PLDroidMediaStreaming 开发的一套具有连麦功能的 SDK，既可以在不影响原有推流的基础上，使用简单易用的连麦 API，快速实现直播连麦功能。也可以抛开推流模块，使用更加灵活的连麦 API，快速实现纯视频连麦互动的功能。

1.1 最新版内容提要

[Android] 连麦 SDK v2.0.0 发布

PLDroidRTCStreaming 是基于 PLDroidMediaStreaming 开发的一套具有连麦功能的 SDK，既可以在不影响原有推流的基础上，使用简单易用的连麦 API，快速实现直播连麦功能。也可以抛开推流模块，使用更加灵活的连麦 API，快速实现纯视频连麦互动的功能。

版本

- 发布 pldroid-rtc-streaming-2.0.0.jar
- 发布 libpldroid_rtc_streaming.so
- 发布 libpldroid_mmprocessing.so
- 发布 libpldroid_streaming_aac_encoder.so
- 发布 libpldroid_streaming_amix.so
- 发布 libpldroid_streaming_h264_encoder.so
- 发布 libpldroid_streaming_core.so

2. 功能列表

- 基本的推流和连麦对讲功能
- 基本的视频合流和音频混音功能
- 支持内置音视频采集，带美颜、水印、闪光灯、摄像头切换、聚焦等常见功能
- 支持外部采集视频数据，支持的格式为：NV21 和 I420
- 支持外部采集音频数据，支持的格式为：PCM，单通道，16bit 位宽
- 支持外部美颜
- 支持踢人功能
- 支持静音功能
- 支持连麦的帧率配置
- 支持连麦的视频码率的配置
- 支持连麦的视频尺寸配置
- 支持丰富的连麦消息回调
- 支持纯音频连麦
- 支持连麦大小窗口切换
- 支持推流的软硬编配置
- 支持连麦的软硬编配置
- 支持连麦画面的截帧
- 支持动态镜像功能

- 支持获取远端麦克风音量大小
- 支持推流码率的自动调节/手动调节
- 支持获取连麦房间统计信息（帧率、码率等）

3. 阅读对象

本文档为技术文档，需要阅读者：

- 具有基本的 Android 开发能力
- 准备接入七牛云直播

4 开发准备

4.1 设备以及系统要求

- 系统要求：Android 4.1 (API 16) 及其以上

4.2 混淆

```
-keep class com.youme.** {*;}
-keep class org.webrtc.** {*;}
-keep class com.qiniu.android.dns.** {*;}
-keep class com.qiniu.pili.droid.streaming.** {*;}
```

5 快速开始

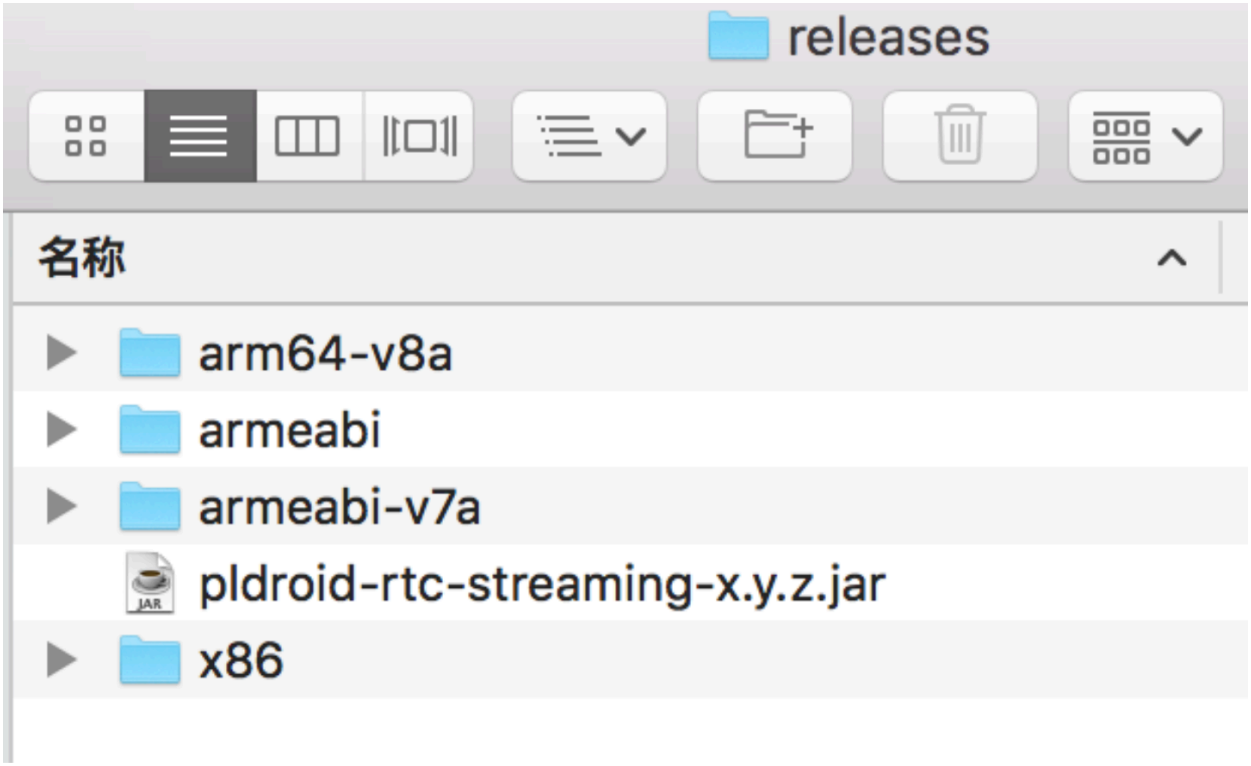
5.1 开发环境

- Android Studio 开发工具，官方下载地址
- Android 官方开发 SDK，官方下载地址。

5.2 下载和导入连麦 SDK

SDK 主要包含 demo 代码、sdk jar 包，以及 sdk 依赖的动态库文件。

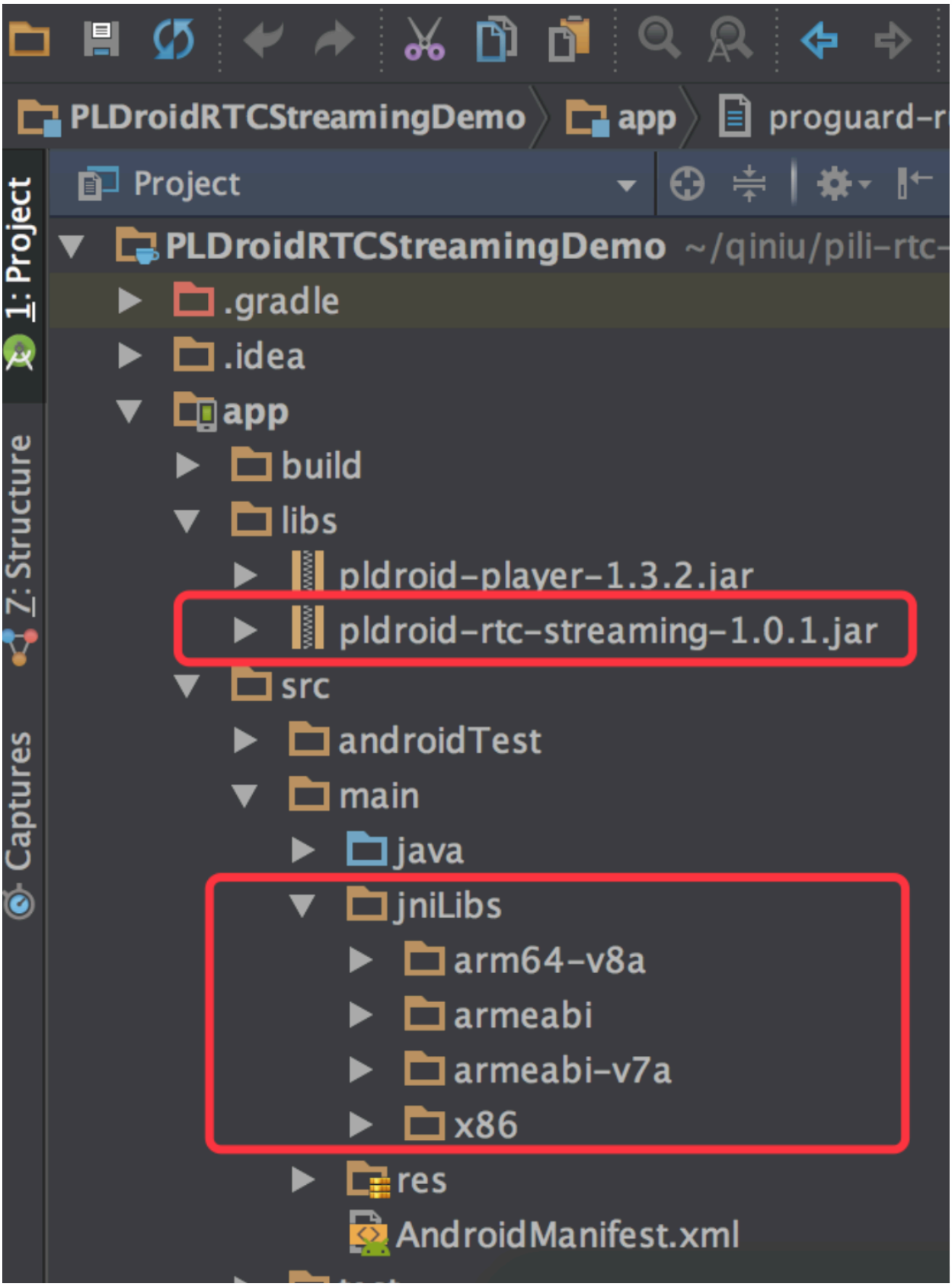
其中，release 目录下是需要拷贝到您的 Android 工程的所有文件，列表如下：



- 将 pldroid-rtc-x.y.z.jar 包拷贝到您的工程的 libs 目录下
- 将动态库拷贝到您的工程对应的目录下，例如：armeabi-v7a 目录下的 so 则拷贝到工程的 jniLibs/armeabi-v7a 目录下

- 将 pldroid-rtc-x.y.z.jar 包拷贝到您的工程的 libs 目录下
- 将动态库拷贝到您的工程对应的目录下，例如：armeabi-v7a 目录下的 so 则拷贝到工程的 jniLibs/armeabi-v7a 目录下

具体可以参考 SDK 包含的 demo 工程，集成后的工程示例如下：



5.3 修改 build.gradle

双击打开您的工程目录下的 `build.gradle`，确保已经添加了如下依赖，如下所示：

```
dependencies {
  compile 'com.qiniu:happy-dns:0.2.x'
  compile files('libs/pldroid-rtc-streaming-x.y.z.jar')
}
```

5.4 添加相关权限

在 app/src/main 目录中的 AndroidManifest.xml 中增加如下 `uses-permission` 声明

```
<!-- 访问网络状态 -->
<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.CHANGE_WIFI_STATE" />

<!-- 系统相关 -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

5.5 开发步骤

5.5.1 初始化

首先，在应用启动后（或者在 Application 里），完成 SDK 的初始化操作：

```
RTCMediaStreamingManager.init(getApplicationContext(),
RTCServerRegion.RTC_CN_SERVER);
```

5.5.2 XML 文件中添加本地预览窗口

在 XML 文件中，配置本地 Camera 预览的视图布局，如下所示：

```
<com.qiniu.pili.droid.streaming.widget.AspectFrameLayout
    android:id="@+id/AspectLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <android.opengl.GLSurfaceView
        android:id="@+id/LocalPreivew"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center" />
</com.qiniu.pili.droid.streaming.widget.AspectFrameLayout>
```

5.5.3 XML 文件中添加远端画面显示窗口

在 XML 文件中，配置远端画面的视图布局，如下所示：

```
<FrameLayout
    android:id="@+id/RemoteWindowA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="invisible"
    android:clickable="true"
    android:onClick="onClickRemoteWindowA" >
    <com.qiniu.pili.droid.rtcstreaming.RTCSurfaceView
        android:id="@+id/RemoteGLSurfaceViewA"
        android:layout_width="120dp"
        android:layout_height="160dp"
        android:visibility="invisible"/>
</FrameLayout>
```

如果是多人连麦，可以添加多个窗口。

5.5.4 创建推流&连麦核心对象

本操作推荐在 `Activity.onCreate()` 函数中完成。

```
AspectFrameLayout afl = (AspectFrameLayout) findViewById(R.id.AspectLayout);
afl.setShowMode(AspectFrameLayout.SHOW_MODE.FULL);

GLSurfaceView localCameraView = (GLSurfaceView) findViewById(R.id.LocalPreivew);
mMediaStreamingManager = new RTCMediaStreamingManager(getApplicationContext(), afl,
localCameraView, AVCodecType.SW_VIDEO_WITH_SW_AUDIO_CODEC);
```

5.5.5 创建本地 Camera 的配置

本操作推荐在 `Activity.onCreate()` 函数中完成。

```
CameraStreamingSetting cameraStreamingSetting = new CameraStreamingSetting();
cameraStreamingSetting.setCameraFacingId(mCameraFacingId)
    .setContinuousFocusModeEnabled(true)
    .setRecordingHint(false)
    .setResetTouchFocusDelayInMs(3000)
    .setFocusMode(CameraStreamingSetting.FOCUS_MODE_CONTINUOUS_PICTURE)
    .setCameraPrvSizeLevel(CameraStreamingSetting.PREVIEW_SIZE_LEVEL.MEDIUM)
    .setCameraPrvSizeRatio(CameraStreamingSetting.PREVIEW_SIZE_RATIO.RATIO_16_9)
    .setBuiltInFaceBeautyEnabled(true) // Using sdk built in face beauty algorithm
    .setFaceBeautySetting(new CameraStreamingSetting.FaceBeautySetting(0.8f, 0.8f,
0.6f)) // sdk built in face beauty settings
    .setVideoFilter(CameraStreamingSetting.VIDEO_FILTER_TYPE.VIDEO_FILTER_BEAUTY);
// set the beauty on/off
```

`CameraStreamingSetting` 的具体含义，请参考推流 SDK 的文档：摄像头参数配置

5.5.6 配置连麦参数

本操作推荐在 `Activity.onCreate()` 函数中完成。

```
RTCConferenceOptions options = new RTCConferenceOptions();
options.setVideoEncodingSizeRatio(RTCConferenceOptions.VIDEO_ENCODING_SIZE_RATIO.RA
TIO_16_9);
options.setVideoEncodingSizeLevel(RTCConferenceOptions.VIDEO_ENCODING_SIZE_HEIGHT_4
80);
options.setVideoEncodingOrientation(RTCConferenceOptions.VideoEncodingOrientation.P
ORT);
// 主播 / 副主播可以配置不同的连麦码率
options.setVideoBitrateRange(300 * 1024, 800 * 1024);
// 配置房间状态回调的触发间隔，0 即为关闭（默认关闭）
options.setStreamStatsInterval(500);
// 配置是否开启硬编
options.setHWCodecEnabled(!isSwCodec);
mMediaStreamingManager.setConferenceOptions(options);
```

5.5.7 添加远端画面的窗口

本操作推荐在 `Activity.onCreate()` 函数中完成。

- 创建远端窗口对象：

```
RTCVideoWindow windowA = new RTCVideoWindow(findViewById(R.id.RemoteWindowA),
(RTCSurfaceView)findViewById(R.id.RemoteGLSurfaceViewA));
RTCVideoWindow windowB = new RTCVideoWindow(findViewById(R.id.RemoteWindowB),
(RTCSurfaceView)findViewById(R.id.RemoteGLSurfaceViewB));
```

- 如果是**主播**，则需要配置连麦合流的参数，如果是连麦观众，则跳过下面的步骤：

```
windowA.setAbsoluteMixOverlayRect(options.getVideoEncodingHeight() - 240, 100, 240,
320);
windowB.setAbsoluteMixOverlayRect(options.getVideoEncodingHeight() - 240, 420, 240,
320);
```

- 完成窗口的添加：

```
mMediaStreamingManager.addRemoteWindow(windowA);
mMediaStreamingManager.addRemoteWindow(windowB);
```

5.5.8 配置推流参数

本操作推荐在 `Activity.onCreate()` 函数中完成。

如果是**主播**，则需要配置推流参数，如果是连麦观众，则跳过下面的步骤：

```
// 配置推流参数
StreamingProfile mStreamingProfile = new StreamingProfile();
mStreamingProfile.setVideoQuality(StreamingProfile.VIDEO_QUALITY_MEDIUM2)
    .setAudioQuality(StreamingProfile.AUDIO_QUALITY_MEDIUM1)
    .setEncoderRCMode(StreamingProfile.EncoderRCModes.QUALITY_PRIORITY)
    .setEncodingOrientation(StreamingProfile.ENCODING_ORIENTATION.PORT)
    .setPreferredVideoEncodingSize(options.getVideoEncodingWidth(),
options.getVideoEncodingHeight()); // 配置推流的尺寸，建议与连麦尺寸一致

// 配置水印参数
WatermarkSetting watermarksetting = new WatermarkSetting(this);
watermarksetting.setResourceId(R.drawable.qiniu_logo)
                .setSize(WatermarkSetting.WATERMARK_SIZE.MEDIUM)
                .setAlpha(100)
                .setCustomPosition(0.5f, 0.5f);
```

注意： 通过 setPreferredVideoEncodingSize 方法设置推流尺寸的时候，要根据推流的尺寸进行相应的配置。比如：横屏推流，则传入（width, height）；竖屏推流，则需传入（height, width）。

推流参数的配置，可以参考我们推流 SDK 的官方文档：推流参数设置

5.5.9 调用 `prepare()` 完成配置

本操作推荐在 `Activity.onCreate()` 函数中完成。

- 如果是主播，则调用下面的函数，配置 Camera 参数的同时，配置水印和推流的参数

```
mMediaStreamingManager.prepare(cameraStreamingSetting, null,  watermarksetting,
mStreamingProfile);
```

- 如果是连麦观众，则调用下面的函数，配置 Camera 参数即可

```
mMediaStreamingManager.prepare(mCameraStreamingSetting, null);
```

5.5.10 开始/停止数据采集

无论是推流还是连麦，都需要数据的采集，因此，建议在 `Activity.onResume()` 和 `Activity.onPause()` 的时候，开启和关闭数据的采集。

```
@Override
protected void onResume() {
     super.onResume();
     mMediaStreamingManager.startCapture();
}

@Override
protected void onPause() {
    super.onPause();
    mMediaStreamingManager.stopCapture();
}
```

5.5.11 开始/停止连麦

上面一系列的配置结束后，就可以在合适的时机开始/停止连麦了，相应的函数原型如下：

```
/**
 * 开始连麦
 *
 * 调用成功后，SDK 首先会加入“连麦房间“，然后开始推送本地音视频流到“连麦服务器”，同时拉取房间中的音视频流
 * 可以通过 `ConferenceStateChangedListener.onConferenceStateChanged(RTCConferenceState state, int extra)` 接收相关状态信息的回调，相关状态包括：
 * 1. RTCConferenceState.VIDEO_PUBLISH_FAILED
 * 2. RTCConferenceState.AUDIO_PUBLISH_FAILED
 * 3. RTCConferenceState.VIDEO_PUBLISH_SUCCESS
 * 4. RTCConferenceState.AUDIO_PUBLISH_SUCCESS
 *
 * 调用本方法之前，需从先业务服务器获取主播的"连麦房间号"和"连麦房间的密钥"
 * 必须在 `RTCConferenceState.READY` 状态之后，才允许调用本方法
 *
 * @param userId    加入连麦的用户ID，必须唯一，同一个ID加入会议会将已加入的用户踢出
 * @param roomName  主播的连麦房间名称
 * @param roomToken 主播的连麦房间的Token，由App服务器动态生成
 * @param callback  调用结果的回调
 */
public void startConference(String userId, String roomName, String roomToken, RTCStartConferenceCallback callback);

/**
 * 停止连麦
 *
 * 调用成功后，SDK 会停止推送本地音视频流到“连麦服务器”，同时停止拉取房间中的音视频流，然后退出“连麦房间”
 */
public void stopConference();
```

5.5.12 开始/停止推流

主播 在配置完成后，可以在合适的时候开启推流，不过要记得先配置推流地址：

```
mStreamingProfile.setPublishUrl(publishAddr);
mMediaStreamingManager.setStreamingProfile(mStreamingProfile);
```

配置好了推流地址以后，就可以开始/停止推流了：

```
/**
 * 开始推流
 * 调用成功后，SDK 开始推送本地音视频流到"流媒体服务器"
 * 必须在 `StreamingStateChangedListener.READY` 状态之后，才允许调用本方法
 *
 * @return 本方法的调用结果
 */
public boolean startStreaming();

/**
 * 停止推流
 * 调用成功后，SDK 会停止推送本地音视频流到"流媒体服务器"
 *
 * @return 本方法的调用结果
 */
 public boolean stopStreaming();
```

5.5.13 退出整个页面

无论是推流还是连麦，在整个 `Activity` 销毁的时候，需要调用如下函数，释放资源：

```
/**
 * 销毁对象，释放资源
 *
 * @return 本方法的调用结果
 */
public void destroy();

/**
 * 销毁引擎
 * 本方法调用后不能再使用该类的其它接口
 */
public static void deinit();
```

5.5.14 后台推流和连麦

由于我们不支持后台推流和连麦，因此，当应用进入后台后，请停止采集、连麦和推流。

6. SDK 接口的设计

6.1 连麦 SDK 接口概述

连麦 SDK 的核心接口类有 3 个，分别是 `RTCMediaStreamingManager`， `RTCStreamingManager` 和 `RTCConferenceManager`， 其中， `RTCMediaStreamingManager` 提供了内置音视频数据的采集和美颜等功能， `RTCStreamingManager` 提供了使用外部数据源做连麦和推流的接口， `RTCConferenceManager` 提供了更加灵活的音视频操作接口，但是该接口类不包含推流功能，仅

适用于纯连麦场景。

6.2 RTCMediaStreamingManager

`RTCMediaStreamingManager` 是负责管理整个推流和连麦生命周期的核心类之一，该类提供了内置音视频采集功能，支持美颜和水印，其核心的接口描述如下：

6.2.1 引擎的初始化/销毁

初始化和反初始化建议在整个 app 开始运行和最终销毁时调用，调用一次即可，初始化后不会在后台占用 CPU 资源

```
/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域，详细区域请参考 RTCServerRegion
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId);

/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域
  * @param logPath 日志保存路径，默认为 null
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId, String logPath);

/**
 * 销毁引擎
 * 本方法调用后不能再使用该类的其它接口
 */
public static void deinit();
```

6.2.2 构造函数

```
/**
  * 构造 RTCMediaStreamingManager 对象
  * 用于音频+视频直播和连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  * @param localView 本地 Camera 预览窗口
  */
```

```
public RTCMediaStreamingManager(Context ctx, GLSurfaceView localView);

/**
  * 构造 RTCMediaStreamingManager 对象
  * 用于音频+视频直播和连麦
  *
  * @param ctx Android 的上下文句柄
  * @param localView 本地 Camera 预览窗口
  * @param encodingType 可选参数：HW_VIDEO_WITH_HW_AUDIO_CODEC 或者
SW_VIDEO_WITH_SW_AUDIO_CODEC
  */
public RTCMediaStreamingManager(Context ctx, GLSurfaceView localView, AVCodecType
encodingType);

 /**
  * 构造 RTCMediaStreamingManager 对象
  * 用于音频+视频直播和连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  * @param layout SDK 提供的用于尺寸变换的视图类
  * @param localView 本地 Camera 预览窗口
  */
public RTCMediaStreamingManager(Context ctx, AspectFrameLayout layout,
GLSurfaceView localView);

 /**
  * 构造 RTCMediaStreamingManager 对象
  * 用于音频+视频直播和连麦
  *
  * @param ctx Android 的上下文句柄
  * @param layout SDK 提供的用于尺寸变换的视图类
  * @param localView 本地 Camera 预览窗口
  * @param encodingType 可选参数：HW_VIDEO_WITH_HW_AUDIO_CODEC 或者
SW_VIDEO_WITH_SW_AUDIO_CODEC
  */
public RTCMediaStreamingManager(Context ctx, AspectFrameLayout layout,
GLSurfaceView localView, AVCodecType encodingType);

 /**
  * 构造 RTCMediaStreamingManager 对象
  * 用于纯音频直播和连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  */
public RTCMediaStreamingManager(Context ctx);

 /**
  * 构造 RTCMediaStreamingManager 对象
  * 用于纯音频直播和连麦
```

```
     *
     * @param ctx Android 的上下文句柄
     * @param encodingType 可选参数：HW_AUDIO_CODEC 或者 SW_AUDIO_CODEC
     */
    public RTCMediaStreamingManager(Context ctx, AVCodecType encodingType);
```

6.2.3 添加连麦窗口

```
/**
 * 添加一个连麦画面的显示窗口
 * 本方纯音频连麦的场景下，也可以添加窗口，用于显示对方的画面
 *
 * @param window 显示窗口
 */
public void addRemoteWindow(RTCVideoWindow window);
```

6.2.4 初始化配置

```
/**
 * 初始化“摄像头“和“麦克风”配置
 * 本方法在不需要推流的场景下使用，即："非主播端"
 * 建议在 Activity.onCreate() 中调用
 *
 * @param camSetting 摄像头的配置参数
 * @param microphoneSetting 麦克风的配置参数
 * @return 本方法的调用结果
 */
public boolean prepare(CameraStreamingSetting camSetting,
MicrophoneStreamingSetting microphoneSetting);

/**
 * 初始化“摄像头“、“麦克风”以及“推流“配置
 * 本方法在需要“推流＋连麦”的场景中调用，即："主播端"
 * 建议在 Activity.onCreate() 中调用
 *
 * @param camSetting 摄像头的配置参数
 * @param microphoneSetting 麦克风的配置参数
 * @param profile 推流的配置参数
 * @return 本方法的调用结果
 */
public boolean prepare(CameraStreamingSetting camSetting,
MicrophoneStreamingSetting microphoneSetting, StreamingProfile profile);

/**
 * 初始化“摄像头“、“麦克风”以及“推流“配置，同时配置水印参数
 * 本方法在需要“推流＋连麦”的场景中调用，即：“主播端“
 * 建议在 Activity.onCreate() 中调用
```

```
     *
     * @param camSetting 摄像头的配置参数
     * @param microphoneSetting 麦克风的配置参数
     * @param wmSetting 水印的配置参数
     * @param profile 推流的配置参数
     * @return 本方法的调用结果
     */
    public boolean prepare(CameraStreamingSetting camSetting,
    MicrophoneStreamingSetting microphoneSetting, WatermarkSetting wmSetting,
    StreamingProfile profile);

    /**
     * 初始化“麦克风”配置
     * 本方法在不需要推流的场景下使用，即："非主播端"，用于纯音频连麦
     * 建议在 Activity.onCreate() 中调用
     *
     * @param microphoneSetting 麦克风的配置参数
     * @return 本方法的调用结果
     */
    public boolean prepare(MicrophoneStreamingSetting microphoneSetting);

    /**
     * 初始化“麦克风”配置
     * 本方法在需要“推流＋连麦”的场景中调用，即：“主播端“，用于纯音频推流＋连麦
     * 建议在 Activity.onCreate() 中调用
     *
     * @param microphoneSetting 麦克风的配置参数
     * @param profile 推流的配置参数
     * @return 本方法的调用结果
     */
    public boolean prepare(MicrophoneStreamingSetting microphoneSetting,
    StreamingProfile profile);
```

6.2.5 开始音视频采集

```
    /**
     * 打开摄像头和麦克风采集
     * 建议在 Activity.onResume() 中调用
     * 打开成功后，会回调`READY`状态
     *
     * @return 本方法的调用结果
     */
     public boolean startCapture();
```

6.2.6 停止音视频采集

```
/**
 * 关闭摄像头和麦克风采集
 * 建议在 Activity.onPause() 中调用
 *
 * @return 本方法的调用结果
 */
 public void stopCapture();
```

6.2.7 释放资源

```
/**
 * 销毁整个对象，释放相关资源
 * 建议在 Activity.onDestroy() 中调用
 *
 * @return 本方法的调用结果
 */
 public void destroy();
```

6.2.8 开始连麦

```
/**
 * 开始连麦
 *
 * 调用成功后，SDK 首先会加入“连麦房间“，然后开始推送本地音视频流到“连麦服务器”，同时拉取房间中的音视频流
 * 可以通过 `ConferenceStateChangedListener.onConferenceStateChanged(RTCConferenceState state, int extra)` 接收相关状态信息的回调，相关状态包括：
 * 1. RTCConferenceState.VIDEO_PUBLISH_FAILED
 * 2. RTCConferenceState.AUDIO_PUBLISH_FAILED
 * 3. RTCConferenceState.VIDEO_PUBLISH_SUCCESS
 * 4. RTCConferenceState.AUDIO_PUBLISH_SUCCESS
 *
 * 调用本方法之前，需从先业务服务器获取主播的"连麦房间号"和"连麦房间的密钥"
 * 必须在 `RTCConferenceState.READY` 状态之后，才允许调用本方法
 *
 * @param userId     加入连麦的用户ID，必须唯一，同一个ID加入会议会将已加入的用户踢出
 * @param roomName   主播的连麦房间名称
 * @param roomToken 主播的连麦房间的Token，由App服务器动态生成
 * @param callback   调用结果的回调
 */
public void startConference(String userId, String roomName, String roomToken, RTCStartConferenceCallback callback);
```

6.2.9 停止连麦

```
/**
 * 停止连麦
 *
 * 调用成功后，SDK 会停止推送本地音视频流到“连麦服务器”，同时停止拉取房间中的音视频流，然后退出“连麦房间”
 */
public void stopConference();
```

6.2.10 开始推流

```
/**
 * 开始推流
 * 调用成功后，SDK 开始推送本地音视频流到"流媒体服务器"
 *
 * @return 本方法的调用结果
 */
public boolean startStreaming();
```

6.2.11 停止推流

```
/**
 * 停止推流
 * 调用成功后，SDK 会停止推送本地音视频流到"流媒体服务器"
 *
 * @return 本方法的调用结果
 */
public boolean stopStreaming();
```

6.2.12 切换摄像头

```
/**
 * 切换摄像头
 * @param facingId 支持的枚举值包括：CAMERA_FACING_BACK, CAMERA_FACING_FRONT, CAMERA_FACING_3RD;
 *
 * @return 本方法的调用结果
 */
public boolean switchCamera(CameraStreamingSetting.CAMERA_FACING_ID facingId);
```

6.2.13 静音功能

```
/**
 * 静音
 *
 * @param audioSource 静音的目标设备（麦克风、扬声器、合流数据）
 */
public void mute(RTCAudioSource audioSource);

/**
 * 取消静音
 *
 * @param audioSource 取消静音的目标设备（麦克风、扬声器、合流数据）
 */
public void unMute(RTCAudioSource audioSource);
```

RTCAudioSource 的具体内容如下：

```
public enum RTCAudioSource {
    // 麦克风
    MIC,
    // 扬声器
    SPEAKER,
    // 合流数据
    MIXAUDIO
}
```

6.2.14 踢人功能

```
/**
 * 根据 userId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param userId 被踢的用户ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(String userId);
```

```
/**
 * 根据 glSurfaceViewId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param glSurfaceViewId 显示被踢用户的窗口ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(int glSurfaceViewId);
```

6.2.15 设置连麦参数

```
/**
 * 设置连麦参数
 * 配置连麦的相关参数，包括：视频的码率、帧率等等
 */
public void setConferenceOptions(RTCConferenceOptions options);
```

6.2.16 更新推流参数

```
/**
 * 更新推流参数
 * 可以在推流之前，更新推流参数，比如配置推流地址等
 */
public void setStreamingProfile(StreamingProfile profile);
```

6.2.17 设置日志级别

```
/**
 * 设置日志级别
 *
 * @param enabled 是否打开调试日志
 */
public void setDebugLoggingEnabled(boolean enabled);
```

6.2.18 获取参与连麦的人数

```
/**
 * 获取参与连麦的人数，不包括自己
 */
public int getParticipantsCount();
```

6.2.19 获取参与连麦的用户ID列表

```
/**
 * 获取参与连麦的用户ID列表，不包括自己
 */
public List<String> getParticipants();
```

6.2.20 大小窗口切换

```
/**
 * 大小窗口切换
 *
 * @param viewToBottom 希望切换到底层的 View
 * @param viewToUpper 希望切换到上层的 View
 */
public void switchRenderView(SurfaceView viewToBottom, SurfaceView viewToUpper);
```

6.2.21 截帧功能

```
/**
 * 截帧功能
 * 只有主播在连麦推流的时候，可以截取合流画面，其他情况下，本功能只支持截取自己预览的画面
 *
 * @param callback 截帧的回调
 */
public void captureFrame(RTCFrameCapturedCallback callback)
```

6.2.22 镜像功能

```
/**
 * 设置 Camera 预览是否镜像
 *
 * @param mirror 是否镜像
 */
public boolean setPreviewMirror(boolean mirror);

/**
 * 设置连麦或者推流出去的画面是否镜像
 *
 * @param mirror 是否镜像
 */
public boolean setEncodingMirror(boolean mirror);
```

6.2.23 设置房间内音频流的音量回调

```
/**
 * 注册音频音量回调
 * @param Callback 对象
*/
public void setAudioLevelCallback(RTCAudioLevelCallback callback);
```

该回调需要在连麦成功后调用 `setAudioLevelMonitorEnabled(true)` 之后才会被触发；当希望停止监听音频信息时，需要调用 `setAudioLevelMonitorEnabled(false)` 来关闭回调。

6.2.24 自定义视频窗口位置

设置合流后的视频尺寸，以及主播画面在合流图像中的位置和大小。此方法必须在 `RTCMediaStreamingManager.prepare` 之前调用，否则将不会生效。

如果不调用此方法，SDK 在合流时，默认会将主播的视频窗口置于最下层并将其大小设置为最终的推流大小，副主播的窗口置于主播视频之上。

```
/**
 * Set local window position
 *
 * @param x the x pos in mixed video
 * @param y the y pos in mixed video
 * @param width the width in mixed video
 * @param height the height in mixed video
 */
public void setLocalWindowPosition(int x, int y, int width, int height);
```

6.2.25 处理合流数据回调

```
public void setMixedFrameCallback(RTCFrameMixedCallback callback);

public void setMixedFrameCallbackEnabled(boolean enable);
```

通过 `setMixedFrameCallback` 注册过 `RTCFrameMixedCallback` 对象后，可以调用 `setMixedFrameCallbackEnabled` 动态开启或关闭合流数据回调。

6.2.26 获取房间统计信息

```
/**
 * 开启数据统计功能，每隔 interval ms 会回调一次数据，数据为该时间段内的统计结果
 *
 * @param interval 统计时间间隔（单位：毫秒）默认为0，为0时不统计
 */
public RTCConferenceOptions setStreamStatsInterval(int interval);

/**
 * 注册数据统计的监听回调，该接口用于 RTCMediaStreamingManager 和 RTCConferenceManager
 */
public final void setRTCStreamStatsCallback(RTCStreamStatsCallback callback);
```

数据统计回调的接口和回调数据类型如下：

```
public interface RTCStreamStatsCallback {
    // 音频码率
    int RTC_STATS_AUDIO_BITRATE = 1;
    // 视频码率
    int RTC_STATS_VIDEO_BITRATE = 2;
    // 视频帧率
    int RTC_STATS_VIDEO_FPS = 3;
    // 音频丢包率，千分比
    int RTC_STATS_AUDIO_PACKET_LOSS_RATE = 4;
    // 视频丢包率，千分比
    int RTC_STATS_VIDEO_PACKET_LOSS_RATE = 5;

    void onStreamStatsChanged(String userId, int statsType, int value);
}
```

6.2.27 图片推流功能

```
/**
 * 在 StreamingProfile 中调用此方法设置推流图片的资源 ID
 *
 * @param resId 推流图片的资源 ID
 */
public void setPictureStreamingResourceId(int resId);

/**
  * 在 StreamingProfile 中调用此方法设置推流图片的文件路径
  *
  * @param filePath 推流图片的文件路径
  */
public void setPictureStreamingFilePath(String filePath);

/**
 * 切换图片推流模式
 */
public boolean togglePictureStreaming();
```

6.2.28 返听功能

```
/**
 * 开启返听功能
 *
 * @return 开启成功与否
 */
public boolean startPlayback();
/**
 * 关闭返听功能
 */
public void stopPlayback()
```

6.3 RTCStreamingManager

`RTCStreamingManager` 是负责管理整个推流和连麦生命周期的核心类之一，该类并不提供视频数据的采集功能，推流和连麦的音视频数据需要由外部导入，其接口描述如下：

6.3.1 引擎的初始化/销毁

初始化和反初始化建议在整个 app 开始运行和最终销毁时调用，调用一次即可，初始化后不会在后台占用 CPU 资源

```
/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域，详细区域请参考 RTCServerRegion
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId);

/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域
  * @param logPath 日志保存路径，默认为 null
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId, String logPath);

/**
 * 销毁引擎
 * 本方法调用后不能再使用该类的其它接口
 */
public static void deinit();
```

6.3.2 构造函数

```
 /**
  * 构造 RTCStreamingManager 对象
  *
  * @param ctx Android 的上下文句柄
  */
public RTCStreamingManager(Context ctx);

 /**
  * 构造 RTCStreamingManager 对象
  *
  * @param ctx Android 的上下文句柄
  * @param encodingType 推流的编码类型
  */
public RTCStreamingManager(Context ctx, AVCodecType encodingType);
```

6.3.3 添加连麦窗口

```
/**
 * 添加一个连麦画面的显示窗口
 *
 * @param window 显示窗口
 */
public void addRemoteWindow(RTCVideoWindow window);
```

6.3.4 初始化推流配置

```
/**
 * 初始化"推流"配置
 * 本方法在需要“推流＋连麦”的场景中调用，比如"主播端"
 * 建议在 Activity.onCreate() 中调用
 *
 * @param profile 推流参数配置对象
 * @return 本方法的调用结果
 */
public boolean prepare(StreamingProfile profile);
```

6.3.5 开始外部采集

```
/**
 * 开始接受外部数据源输入
 * 建议在 Activity.onResume() 中调用
 *
 * @return 本方法的调用结果
 */
 public boolean resume();
```

6.3.6 停止外部采集

```
/**
 * 停止接受外部数据源输入
 * 建议在 Activity.onPause() 中调用
 *
 * @return 本方法的调用结果
 */
 public void pause();
```

6.3.7 释放资源

```
/**
 * 销毁整个对象，释放相关资源
 * 建议在 Activity.onDestroy() 中调用
 * @return 本方法的调用结果
 */
 public void destroy();
```

6.3.8 开始连麦

```
/**
 * 开始连麦
 *
 * 调用成功后，SDK 首先会加入“连麦房间“，然后开始推送本地音视频流到“连麦服务器”，同时拉取房间中的音视频流
 * 可以通过 `ConferenceStateChangedListener.onConferenceStateChanged(RTCConferenceState state, int extra)` 接收相关状态信息的回调，相关状态包括：
 * 1. RTCConferenceState.VIDEO_PUBLISH_FAILED
 * 2. RTCConferenceState.AUDIO_PUBLISH_FAILED
 * 3. RTCConferenceState.VIDEO_PUBLISH_SUCCESS
 * 4. RTCConferenceState.AUDIO_PUBLISH_SUCCESS
 *
 * 调用本方法之前，需从先业务服务器获取主播的"连麦房间号"和"连麦房间的密钥"
 * 必须在 `RTCConferenceState.READY` 状态之后，才允许调用本方法
 *
 * @param userId    加入连麦的用户ID，必须唯一，同一个ID加入会议会将已加入的用户踢出
 * @param roomName  主播的连麦房间名称
 * @param roomToken 主播的连麦房间的 Token，由 App 服务器动态生成
 * @param callback  调用结果的回调
 */
public void startConference(String userId, String roomName, String roomToken, RTCStartConferenceCallback callback);
```

6.3.9 停止连麦

```
/**
 * 停止连麦
 *
 * 调用成功后，SDK 会停止推送本地音视频流到“连麦服务器”，同时停止拉取房间中的音视频流，然后退出“连麦房间”
 */
public void stopConference();
```

6.3.10 开始推流

```
/**
 * 开始推流
 * 调用成功后，SDK 开始连接到"流媒体服务器"
 *
 * @return 本方法的调用结果
 */
public boolean startStreaming();
```

6.3.11 停止推流

```
/**
 * 开始推流
 * 调用成功后，SDK 断开与"流媒体服务器"的连接
 *
 * @return 本方法的调用结果
 */
public boolean stopStreaming();
```

6.3.12 静音功能

```
/**
 * 静音
 *
 * @param audioSource 静音的目标设备（麦克风、扬声器、合流数据）
 */
public void mute(RTCAudioSource audioSource);

/**
 * 取消静音
 *
 * @param audioSource 取消静音的目标设备（麦克风、扬声器、合流数据）
 */
public void unMute(RTCAudioSource audioSource);
```

RTCAudioSource 的具体内容如下：

```
public enum RTCAudioSource {
    // 麦克风
    MIC,
    // 扬声器
    SPEAKER,
    // 合流数据
    MIXAUDIO
}
```

6.3.13 踢人功能

```
/**
 * 根据 userId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param userId 被踢的用户ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(String userId);
```

```
/**
 * 根据 glSurfaceViewId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param glSurfaceViewId 显示被踢用户的窗口ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(int glSurfaceViewId);
```

6.3.14 设置连麦参数

```
/**
 * 设置连麦参数
 * 配置连麦的相关参数，包括：视频的码率、帧率等等
 *
 * @return 无
 */
public void setConferenceOptions(RTCConferenceOptions options);
```

6.3.15 更新推流参数

```
/**
 * 更新推流参数
 * 可以在推流之前，更新推流参数，配置推流地址等
 *
 * @return 无
 */
public void setStreamingProfile(StreamingProfile profile);
```

6.3.16 设置日志级别

```
/**
 * 设置日志级别
 *
 * @param enabled 是否打开调试日志
 *
 * @return 无
 */
public void setDebugLoggingEnabled(boolean enabled);
```

6.3.17 送入外部视频数据

```
/**
 * 送入外部视频数据
 *
 * @param data 存放视频帧的数组
 * @param width 外部视频的宽
 * @param height 外部视频的高
 * @param rotation 外部视频的角度，0，90，180，270度
 * @param mirror 是否镜像
 * @param fmt 视频格式，只支持：PLFourCC.FOURCC_NV21，PLFourCC.FOURCC_I420
 * @param ts 视频帧的时间戳，单位：ns
 *
 * @return 本方法的调用结果
 */
public boolean inputVideoFrame(byte[] data, int width, int height, int rotation,
boolean mirror, int fmt, long ts)
```

6.3.18 送入外部音频数据

```
/**
 * 送入外部音频数据
 *
 * @param data 存放音频帧的数组
 * @param ts 音频帧的时间戳，单位：ns
 *
 * @return 本方法的调用结果
 */
public boolean inputAudioFrame(byte[] data, long ts)
```

6.3.19 获取参与连麦的人数

```
/**
 * 获取参与连麦的人数，不包括自己
 */
public int getParticipantsCount();
```

6.3.20 获取参与连麦的用户ID列表

```
/**
 * 获取参与连麦的用户ID列表，不包括自己
 */
public List<String> getParticipants();
```

6.3.21 设置房间内音频流的音量回调

```
/**
 * 注册音频音量回调
 * @param Callback 对象
*/
public void setAudioLevelCallback(RTCAudioLevelCallback callback);
```

该回调需要在连麦成功后调用 `setAudioLevelMonitorEnabled(true)` 之后才会被触发；当希望停止监听音频信息时，需要调用 `setAudioLevelMonitorEnabled(false)` 来关闭回调。

6.3.22 处理合流数据回调

```
public void setMixedFrameCallback(RTCFrameMixedCallback callback);

public void setMixedFrameCallbackEnabled(boolean enable);
```

通过 `setMixedFrameCallback` 注册过 `RTCFrameMixedCallback` 对象后，可以调用 `setMixedFrameCallbackEnabled` 动态开启或关闭合流数据回调。

6.3.23 获取房间统计信息

```
/**
 * 开启数据统计功能，每隔 interval ms 会回调一次数据，数据为该时间段内的统计结果
 *
 * @param interval 统计时间间隔（单位：毫秒）默认为0，为0时不统计
 */
public RTCConferenceOptions setStreamStatsInterval(int interval);

/**
 * 注册数据统计的监听回调，该接口用于 RTCMediaStreamingManager 和 RTCConferenceManager
 */
public final void setRTCStreamStatsCallback(RTCStreamStatsCallback callback);
```

数据统计回调的接口和回调数据类型如下：

```
public interface RTCStreamStatsCallback {
    // 音频码率
    int RTC_STATS_AUDIO_BITRATE = 1;
    // 视频码率
    int RTC_STATS_VIDEO_BITRATE = 2;
    // 视频帧率
    int RTC_STATS_VIDEO_FPS = 3;
    // 音频丢包率，千分比
    int RTC_STATS_AUDIO_PACKET_LOSS_RATE = 4;
    // 视频丢包率，千分比
    int RTC_STATS_VIDEO_PACKET_LOSS_RATE = 5;

    void onStreamStatsChanged(String userId, int statsType, int value);
}
```

6.4 RTCConferenceManager

`RTCConferenceManager` 是负责管理纯连麦场景下连麦生命周期的核心类，该类不包含推流功能，但是提供了更加灵活可控的音视频操作的接口，其核心的接口描述如下：

6.4.1 引擎的初始化/销毁

初始化和反初始化建议在整个 app 开始运行和最终销毁时调用，调用一次即可，初始化后不会在后台占用 CPU 资源

```
/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域，详细区域请参考 RTCServerRegion
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId);

/**
  * 初始化引擎
  * 该类的所有其他接口必须在本方法调用成功后才能使用
  *
  * @param context the context Android 的上下文句柄
  * @param serverRegionId 连麦服务器的区域
  * @param logPath 日志保存路径，默认为 null
  * @return 错误码，0 代表成功，其他数值为初始化失败
  */
public static int init(Context context, int serverRegionId, String logPath);
/**
 * 销毁引擎
 * 本方法调用后不能再使用该类的其它接口
 */
public static void deinit();
```

6.4.2 构造函数

```
/**
  * 构造 RTCConferenceManager 对象
  * 用于纯音频连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  */
public RTCConferenceManager(Context ctx);
/**
  * 构造 RTCConferenceManager 对象
  * 用于音频+视频连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  * @param localView 本地 Camera 预览窗口
  */
public RTCConferenceManager(Context ctx, GLSurfaceView localView);
/**
  * 构造 RTCConferenceManager 对象
  * 用于音频+视频连麦，默认采用软编推流
  *
  * @param ctx Android 的上下文句柄
  * @param layout SDK 提供的用于尺寸变换的视图类
  * @param localView 本地 Camera 预览窗口
  */
public RTCConferenceManager(Context ctx, AspectFrameLayout layout, GLSurfaceView
localView);
/**
  * 构造 RTCConferenceManager 对象
  * 用于音频+视频连麦
  *
  * @param ctx Android 的上下文句柄
  * @param layout SDK 提供的用于尺寸变换的视图类
  * @param localView 本地 Camera 预览窗口
  * @param encodingType 编码方式
  */
public RTCConferenceManager(Context ctx, AspectFrameLayout layout, GLSurfaceView
localView, AVCodecType encodingType);
```

6.4.3 添加连麦窗口

```
/**
 * 添加一个连麦画面的显示窗口
 * 本方纯音频连麦的场景下，也可以添加窗口，用于显示对方的画面
 *
 * @param window 显示窗口
 */
public void addRemoteWindow(RTCVideoWindow window);
```

6.4.4 初始化配置

```
/**
 * 初始化“麦克风”配置
 * 本方法用于纯音频连麦
 * 建议在 Activity.onCreate() 中调用
 *
 * @param microphoneSetting 麦克风的配置参数
 * @return 本方法的调用结果
 */
public boolean prepare(MicrophoneStreamingSetting microphoneSetting);
/**
 * 初始化“摄像头“和“麦克风”配置
 * 建议在 Activity.onCreate() 中调用
 *
 * @param camSetting 摄像头的配置参数
 * @param microphoneSetting 麦克风的配置参数
 * @return 本方法的调用结果
 */
public boolean prepare(CameraStreamingSetting camSetting,
MicrophoneStreamingSetting microphoneSetting);
/**
 * 初始化“摄像头“、“麦克风”配置，同时配置水印参数
 * 建议在 Activity.onCreate() 中调用
 *
 * @param camSetting 摄像头的配置参数
 * @param microphoneSetting 麦克风的配置参数
 * @param wmSetting 水印的配置参数
 * @return 本方法的调用结果
 */
public boolean prepare(CameraStreamingSetting camSetting,
MicrophoneStreamingSetting microphoneSetting, WatermarkSetting wmSetting);
```

6.4.5 开始/关闭视频数据采集

开启和关闭视频数据的采集，建议在 `Activity.onResume()` 和 `Activity.onPause()` 的时候调用。

```
@Override
protected void onResume() {
     super.onResume();
     mRTCStreamingManager.startVideoCapture();
}

@Override
protected void onPause() {
    super.onPause();
    mRTCStreamingManager.stopVideoCapture();
}
```

6.4.6 开始/关闭音频数据采集

开启和关闭音频数据的采集，建议在 `Activity.onResume()` 和 `Activity.onPause()` 的时候调用。

```
@Override
protected void onResume() {
     super.onResume();
     mRTCStreamingManager.startAudioCapture();

@Override
protected void onPause() {
    super.onPause();
    mRTCStreamingManager.stopAudioCapture();
}
```

6.4.7 开始/关闭视频数据的发布

开启和关闭视频数据的发布。

```
mRTCStreamingManager.publishLocalVideo();
mRTCStreamingManager.unpublishLocalVideo();
```

6.4.8 开始/关闭音频数据的发布

开启和关闭音频数据的发布。

```
mRTCStreamingManager.publishLocalAudio();
mRTCStreamingManager.unpublishLocalAudio();
```

6.4.9 释放资源

```
/**
 * 销毁整个对象，释放相关资源
 * 建议在 Activity.onDestroy() 中调用
 *
 * @return 本方法的调用结果
 */
 public void destroy();
```

6.4.10 开始连麦

```
/**
 * 开始连麦
 *
 * 调用成功后，SDK 首先会加入“连麦房间“，然后开始推送本地音视频流到“连麦服务器”，同时拉取房间中的音视频流
 * 可以通过 `ConferenceStateChangedListener.onConferenceStateChanged(RTCConferenceState state, int extra)` 接收相关状态信息的回调，相关状态包括：
 * 1. RTCConferenceState.VIDEO_PUBLISH_FAILED
 * 2. RTCConferenceState.AUDIO_PUBLISH_FAILED
 * 3. RTCConferenceState.VIDEO_PUBLISH_SUCCESS
 * 4. RTCConferenceState.AUDIO_PUBLISH_SUCCESS
 *
 * 调用本方法之前，需从先业务服务器获取主播的"连麦房间号"和"连麦房间的密钥"
 * 必须在 `RTCConferenceState.READY` 状态之后，才允许调用本方法
 *
 * @param userId     加入连麦的用户ID，必须唯一，同一个ID加入会议会将已加入的用户踢出
 * @param roomName   主播的连麦房间名称
 * @param roomToken 主播的连麦房间的Token，由App服务器动态生成
 * @param callback   调用结果的回调
 */
public void startConference(String userId, String roomName, String roomToken, RTCStartConferenceCallback callback);
```

6.4.11 停止连麦

```
/**
 * 停止连麦
 *
 * 调用成功后，SDK 会停止推送本地音视频流到“连麦服务器”，同时停止拉取房间中的音视频流，然后退出“连麦房间”
 */
public void stopConference();
```

6.4.12 切换摄像头

```
/**
 * 切换摄像头
 * @param facingId 支持的枚举值包括：CAMERA_FACING_BACK, CAMERA_FACING_FRONT,
CAMERA_FACING_3RD;
 *
 * @return 本方法的调用结果
 */
public boolean switchCamera(CameraStreamingSetting.CAMERA_FACING_ID facingId);
```

6.4.13 静音功能

```
/**
 * 静音
 *
 * @param audioSource 静音的目标设备（麦克风、扬声器、合流数据）
 */
public void mute(RTCAudioSource audioSource);

/**
 * 取消静音
 *
 * @param audioSource 取消静音的目标设备（麦克风、扬声器、合流数据）
 */
public void unMute(RTCAudioSource audioSource);
```

RTCAudioSource 的具体内容如下：

```
public enum RTCAudioSource {
    // 麦克风
    MIC,
    // 扬声器
    SPEAKER,
    // 合流数据
    MIXAUDIO
}
```

6.4.14 踢人功能

```
/**
 * 根据 userId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param userId 被踢的用户ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(String userId);
```

```
/**
 * 根据 glSurfaceViewId 踢人
 * 被踢的用户会收到 "USER_KICKOUT_BY_HOST" 的消息回调
 *
 * @param glSurfaceViewId 显示被踢用户的窗口ID
 *
 * @return 本方法的调用结果
 */
public boolean kickoutUser(int glSurfaceViewId);
```

6.4.15 设置连麦参数

```
/**
 * 设置连麦参数
 * 配置连麦的相关参数，包括：视频的码率、帧率等等
 */
public void setConferenceOptions(RTCConferenceOptions options);
```

6.4.16 设置日志级别

```
/**
 * 设置日志级别
 *
 * @param enabled 是否打开调试日志
 */
public void setDebugLoggingEnabled(boolean enabled);
```

6.4.17 获取参与连麦的人数

```
/**
 * 获取参与连麦的人数，不包括自己
 */
public int getParticipantsCount();
```

6.4.18 获取参与连麦的用户ID列表

```
/**
 * 获取参与连麦的用户ID列表，不包括自己
 */
public List<String> getParticipants();
```

6.4.19 大小窗口切换

```
/**
 * 大小窗口切换
 *
 * @param viewToBottom 希望切换到底层的 View
 * @param viewToUpper 希望切换到上层的 View
 */
public void switchRenderView(SurfaceView viewToBottom, SurfaceView viewToUpper);
```

6.4.20 截帧功能

```
/**
 * 截帧功能
 * 只有主播在连麦推流的时候，可以截取合流画面，其他情况下，本功能只支持截取自己预览的画面
 *
 * @param callback 截帧的回调
 */
public void captureFrame(RTCFrameCapturedCallback callback)
```

6.4.21 镜像功能

```
/**
 * 设置 Camera 预览是否镜像
 *
 * @param mirror 是否镜像
 */
public boolean setPreviewMirror(boolean mirror);

/**
 * 设置连麦或者推流出去的画面是否镜像
 *
 * @param mirror 是否镜像
 */
public boolean setEncodingMirror(boolean mirror);
```

6.4.22 设置房间内音频流的音量回调

```
/**
 * 注册音频音量回调
 * @param Callback 对象
*/
public void setAudioLevelCallback(RTCAudioLevelCallback callback);
```

该回调需要在连麦成功后调用 `setAudioLevelMonitorEnabled(true)` 之后才会被触发；当希望停止监听音频信息时，需要调用 `setAudioLevelMonitorEnabled(false)` 来关闭回调。

6.4.23 获取房间统计信息

```
/**
 * 开启数据统计功能，每隔 interval ms 会回调一次数据，数据为该时间段内的统计结果
 *
 * @param interval 统计时间间隔（单位：毫秒）默认为0，为0时不统计
 */
public RTCConferenceOptions setStreamStatsInterval(int interval);

/**
 * 注册数据统计的监听回调，该接口用于 RTCMediaStreamingManager 和 RTCConferenceManager
 */
public final void setRTCStreamStatsCallback(RTCStreamStatsCallback callback);
```

数据统计回调的接口和回调数据类型如下：

```
public interface RTCStreamStatsCallback {
    // 音频码率
    int RTC_STATS_AUDIO_BITRATE = 1;
    // 视频码率
    int RTC_STATS_VIDEO_BITRATE = 2;
    // 视频帧率
    int RTC_STATS_VIDEO_FPS = 3;
    // 音频丢包率，千分比
    int RTC_STATS_AUDIO_PACKET_LOSS_RATE = 4;
    // 视频丢包率，千分比
    int RTC_STATS_VIDEO_PACKET_LOSS_RATE = 5;

    void onStreamStatsChanged(String userId, int statsType, int value);
}
```

6.5 参数配置/状态获取

6.5.1 CameraStreamingSetting

`CameraStreamingSetting` 用于配置内置的 Camera 参数，包括：分辨率、对焦模式、美颜等等，详细用法可以参考我们推流SDK的官方文档：[摄像头参数配置]

6.5.2 StreamingProfile

`StreamingProfile` 用于配置推流参数，包括推流的地址、编码参数等等，详细用法可以参考我们推流SDK的官方文档：[推流参数设置]

6.5.3 RTCConferenceOptions

`RTCConferenceOptions` 用于配置连麦会话的参数，包括连麦的码率、帧率、软编硬编等，其定义如下：

6.5.3.1 设置连麦编码器的软编硬编

```
/**
  * 设置连麦编码器的软编硬编
  * 默认情况下优先使用硬编
  *
  * @param enabled 是否启用硬编
  * @return 本对象的指针
  */
public RTCConferenceOptions setHWCodecEnabled(boolean enabled)
```

6.5.3.2 设置连麦编码器输出的码率

```
/**
 * 设置连麦编码器输出的码率
 * 连麦的码率会根据当前的网络状况动态的调整
 *
 * @param minBitrateInBps 码率浮动的下限，默认值：100 * 1024，单位：bps
 * @param maxBitrateInBps 码率浮动的上限，默认值：300 * 1024，单位：bps
 * @note: 如果 minBitrateInBps == maxBitrateInBps，则采用固定码率
 * @return 本对象的指针
 */
public RTCConferenceOptions setVideoBitrateRange(int minBitrateInBps, int
maxBitrateInBps);
```

6.5.3.3 设置连麦编码器输出的帧率

```
/**
 * 设置连麦编码器输出的帧率
 *
 * @param fps 视频帧率
 * @return 本对象的指针
 */
public RTCConferenceOptions setVideoEncodingFps(int fps);
```

6.5.3.4 设置连麦编码器输出的画面比例

```
/**
 * 设置连麦编码器输出的画面比例
 *
 * @param ratio 比例，可选参数：RATIO_4_3，RATIO_16_9
 * @return 本对象的指针
 */
public RTCConferenceOptions setVideoEncodingSizeRatio(VIDEO_ENCODING_SIZE_RATIO
ratio);
```

6.5.3.5 设置连麦编码器输出的画面尺寸级别

```
/**
 * 设置连麦编码器输出的画面尺寸级别
 *
 * @param ratio 比例，可选参数：
 * <li> VIDEO_ENCODING_SIZE_DEFAULT </li>
 * <li> VIDEO_ENCODING_SIZE_HEIGHT_240 </li>
 * <li> VIDEO_ENCODING_SIZE_HEIGHT_480 </li>
 * <li> VIDEO_ENCODING_SIZE_HEIGHT_544 </li>
 * <li> VIDEO_ENCODING_SIZE_HEIGHT_720 </li>
 * <li> VIDEO_ENCODING_SIZE_HEIGHT_1088 </li>
 *
 * @return 本对象的指针
 */
public RTCConferenceOptions setVideoEncodingSizeLevel(int level);
```

6.5.3.8 获取连麦编码器输出的画面尺寸

画面比例和尺寸级别最终对应的图像宽高值的关系如下表所示：

Level	Resolution(16:9)	Resolution(4:3)
VIDEO_ENCODING_SIZE_DEFAULT	原始输入尺寸	原始输入尺寸
VIDEO_ENCODING_HEIGHT_240	424 x 240	320 x 240
VIDEO_ENCODING_HEIGHT_480	848 x 480	640 x 480
VIDEO_ENCODING_HEIGHT_544	960 x 544	720 x 544
VIDEO_ENCODING_HEIGHT_720	1280 x 720	960 x 720
VIDEO_ENCODING_HEIGHT_1088	1920 x 1088	1440 x 1088

可以通过 `Get` 方法获取最终的尺寸配置结果，如下所示：

```
/**
 * 获取编码器输出的画面的宽
 *
 * @return 宽，单位：px
 */
public int getVideoEncodingWidth();

/**
 * 获取编码器输出的画面的高
 *
 * @return 高，单位：px
 */
public int getVideoEncodingHeight();
```

6.5.4 RTCVideoWindow

`RTCVideoWindow` 是连麦画面的窗口对象，用来管理窗口对象以及配置合流参数，注：只有主播需要配置合流画面的位置和大小。

6.5.4.1 构造函数

```
/**
 * 构造函数
 *
 * @param surfaceView 用于渲染连麦画面的 View
 */
public RTCVideoWindow(RTCSurfaceView surfaceView);

/**
 * 构造函数
 *
 * @param parentView 伴随连麦窗口（glSurfaceView）一起显示/隐藏的父窗口
 * @param surfaceView 用于渲染连麦画面的 View
 */
public RTCVideoWindow(View parentView, RTCSurfaceView surfaceView);
```

6.5.4.2 配置合流参数(方法一)

```
/**
 * 使用相对值来配置该窗口在合流画面中的位置和大小
 * 主窗口的原点坐标在左上角
 *
 * @param x 相对于合流主窗口的 x 方向的位置比例，取值范围：0.0 ~ 1.0
 * @param y 相对于合流主窗口的 y 方向的位置比例，取值范围：0.0 ~ 1.0
 * @param w 相对于合流主窗口的宽的比例，取值范围：0.0 ~ 1.0
 * @param h 相对于合流主窗口的高的比例，取值范围：0.0 ~ 1.0
 */
public void setRelativeMixOverlayRect(float x, float y, float w, float h);
```

假设主窗口是 640 x 480 的画面，连麦窗口的配置是：[0.5, 0.3, 0.5, 0.5]，那么，实际的效果是：

连麦窗口的大小是：320 x 240
连麦窗口的位置是：x 坐标位于主窗口从左到右的 1/2 处，y 坐标位于主窗口从上往下的 3/10 处。

6.5.4.3 配置合流参数(方法二)

```
/**
 * 使用绝对值来配置该窗口在合流画面中的位置和大小
 * 主窗口的原点坐标在左上角
 *
 * @param x 连麦窗口的 x 坐标
 * @param y 连麦窗口的 y 坐标
 * @param w 连麦窗口的宽，单位：像素
 * @param h 连麦窗口的高，单位：像素
 */
public void setAbsolutetMixOverlayRect(int x, int y, int w, int h);
```

6.5.4.4 配置窗口的上下层级

```
/**
 * 设置窗口的 z order
 */
public void setZOrderMediaOverlay(boolean isMediaOverlay);
public void setZOrderOnTop(boolean onTop);
```

6.5.4.5 优化合流参数的配置

合理的配置连麦画面的尺寸和合流参数的大小，可以显著降低主播端的 CPU、内存、带宽和功耗。

- 减小连麦者的画面尺寸

由于连麦对象的画面是小窗口，因此，对于连麦者，可以考虑使用比主播小一些的画面尺寸，比如：`VIDEO_ENCODING_HEIGHT_240`，这样可以显著降低主播端对连麦画面的剪裁压力。

- 主播端合流参数尽量匹配连麦画面

主播端配置连麦合流窗口的尺寸尽量接近连麦者的画面尺寸，比如：如果连麦者输出的画面尺寸是 320 x 240 ，那么主播端配置该对象合流的尺寸就用 320 x 240 或者小于该尺寸的值，这样可以避免或者减少主播端对连麦画面进行拉伸，显著降低主播端的 CPU 和功耗。

- 主播端预览尺寸尽量匹配推流尺寸

主播端配置的预览尺寸尽量要匹配推流的尺寸，比如：如果推流的编码尺寸配置的是 848 x 480，那么预览时摄像头采集的尺寸也要尽量和 848 x 480 相匹配，这样可以避免或者减少主播端合流时对连麦画面进行拉伸，显著降低主播端的 CPU 和功耗，同时会提高连麦推流的流畅性。

6.5.5 RTCConferenceStateChangedListener

`ConferenceStateChangedListener` 用于监听连麦过程的相关回调消息，其定义如下：

```
public interface ConferenceStateChangedListener {
    /**
     * @param state 状态枚举
     * @param extra 附加参数，比如错误码
     */
    void onConferenceStateChanged(RTCConferenceState state, int extra);
}
```

目前定义的 `RTCConferenceState` 主要包括：

```
public enum RTCStreamingState {
    /**
     * 准备就绪
     * 当摄像头和麦克风都成功打开后，则进入就绪状态
     * 必须进入就绪状态后，才能开始连麦
     */
    READY

    /**
     * 正在连接服务器
     * 当网络断开后，会回调该状态，自动重连服务器
     */
    RECONNECTING,

    /**
     * 已经连接上服务器
     */
    RECONNECTED,

    /**
     * 多次尝试连接服务器均失败后，会回调该状态
     */
    RECONNECT_FAIL,

    /**
```

```
     * 无法发布视频到连麦房间
     */
    VIDEO_PUBLISH_FAILED,

    /**
     * 无法发布音频到连麦房间
     */
    AUDIO_PUBLISH_FAILED,

    /**
     * 成功发布视频到连麦房间
     */
    VIDEO_PUBLISH_SUCCESS,

    /**
     * 成功发布音频到连麦房间
     */
    AUDIO_PUBLISH_SUCCESS,

    /**
     * 用户在其它地方登录
     */
    USER_JOINED_AGAIN,

    /**
     * 用户被管理员踢出房间
     */
    USER_KICKOUT_BY_HOST,

    /**
     * 打开摄像头失败
     */
    OPEN_CAMERA_FAIL,

    /**
     * 打开麦克风失败
     */
    AUDIO_RECORDING_FAIL,
}
```

6.5.6 RTCUserEventListener

`RTCUserEventListener` 用于监听连麦事件，包括远端用户加入连麦、远端用户退出连麦等。

```
/**
 * 远端用户成功加入连麦
 * @param userId 远端用户的 userId
 */
void onUserJoinConference(String userId);

/**
 * 远端用户退出连麦
 * @param userId 远端用户的 userId
 */
void onUserLeaveConference(String userId);
```

6.5.7 RTCRemoteWindowEventListener

`RTCRemoteWindowEventListener` 用于监听远端窗口显示/隐藏事件。

```
/**
 * 远端连麦视频窗口显示事件
 * @param window 显示远端用户的窗口对象
 * @param remoteUserId 远端用户的 userId
 */
void onRemoteWindowAttached(RTCVideoWindow window, String remoteUserId);

/**
 * 远端连麦视频窗口隐藏事件
 * @param window 显示远端用户的窗口对象
 * @param remoteUserId 远端用户的 userId
 */
void onRemoteWindowDetached(RTCVideoWindow window, String remoteUserId);

/**
 * 第一帧视频画面渲染时间
 *
 * @param remoteUserId 远端用户的 userId
 */
void onFirstRemoteFrameArrived(String remoteUserId);
```

6.5.8 StreamStatusCallback

`StreamStatusCallback` 用于监听推流统计信息回调，其定义如下：

```
public interface StreamStatusCallback() {
    void notifyStreamStatusChanged(final StreamingProfile.StreamStatus status);
}
```

目前定义的推流统计信息主要包括：

```
public static class StreamStatus {
    /**
     * 音频的实时帧率
     * */
    public int audioFps;

    /**
     * 视频的实时帧率
     * */
    public int videoFps;

    /**
     * 音视频的平均码率
     * */
    public int totalAVBitrate;  // bps

    /**
     * 音频的实时码率
     * */
    public int audioBitrate;  // bps

    /**
     * 视频的实时码率
     * */
    public int videoBitrate;  // bps
}
```

6.5.9 StreamingStateChangedListener

`StreamingStateChangedListener` 用于监听推流的状态信息回调，其定义如下：

```
public interface StreamingStateChangedListener() {
     void onStateChanged(StreamingState status, Object extra);
}
```

目前定义的推流状态信息主要包括：

```
public enum StreamingState {
    /**
     * 未知状态
     *
     * */
    UNKNOWN,

    /**
     * 正在初始化
     *
     * */
```

```
    PREPARING,

    /**
     * 准备就绪
     * */
    READY,

    /**
     * 正在连接流媒体服务器
     *
     * */
    CONNECTING,

    /**
     * 开始推流
     *
     * */
    STREAMING,

    /**
     * 停止推流
     *
     * */
    SHUTDOWN,

    /**
     * 无法连接流媒体服务器
     * */
    IOERROR,

    /**
     * 发送队列数据已空
     *
     * */
    SENDING_BUFFER_EMPTY,

    /**
     * 发送队列数据已满
     *
     * */
    SENDING_BUFFER_FULL,

    /**
     * 与流媒体服务器连接断开
     *
     * */
    DISCONNECTED,

    /**
```

```
     * 打开摄像头失败
     */
    OPEN_CAMERA_FAIL,

    /**
     * 打开麦克风失败
     */
    AUDIO_RECORDING_FAIL,

    /**
     * 无效的推流地址
     *
     * */
    INVALID_STREAMING_URL,

    /**
     * 流媒体服务器连接成功
     *
     * */
    CONNECTED,
}
```

6.5.10 RTCAudioLevelCallback

`RTCAudioLevelCallback` 用于监听当前连麦房间里所有人的音量，其定义如下：

```
public interface RTCAudioLevelCallback {
    /**
     * Callback only when audio level monitor enabled.
     *
     * @param userId the userId
     * @param level the audio level
     */
    void onAudioLevelChanged(String userId, int level);
}
```

6.5.11 RTCFrameMixedCallback

当有新的视频 / 音频帧合流完成时，`RTCFrameMixedCallback` 会被触发。其定义如下：

```
public interface RTCFrameMixedCallback {
    void onVideoFrameMixed(byte[] data, int width, int height, int fmt, long
timestamp);
    void onAudioFrameMixed(byte[] pcm, long timestamp);
}
```

目前 `RTCFrameMixedCallback` 有如下方法：

```
    /**
     * 当视频帧合流完成时触发
     * @param data 视频数据
     * @param width 宽
     * @param height 高
     * @param fmt 视频数据的格式，0 for I420，1 for NV21
     * @param timestamp 视频时间戳
     */
    void onVideoFrameMixed(byte[] data, int width, int height, int fmt, long
timestamp);

    /**
     * 当音频帧合流完成时触发
     * @param pcm 音频的 PCM 数据
     * @param timestamp 音频时间戳
     */
    void onAudioFrameMixed(byte[] pcm, long timestamp);
```

6.5.12 RTCRemoteAudioEventListener

`RTCRemoteAudioEventListener` 用于监听远端音频的发布/取消发布的事件。因为直播连麦场景下默认会发布音频，故该 listener 仅用于纯连麦互动场景，不应用于直播连麦场景

```
/**
 * 远端连麦音频发布事件
 * @param userId 远端用户的 userId
 */
void onRemoteAudioPublished(String userId);

/**
 * 远端连麦视频取消发布事件
 * @param userId 远端用户的 userId
 */
void onRemoteAudioUnpublished(String userId);
```

6.5.13 RTCServerRegion

`RTCServerRegion` 用于设置首选连接服务器的区域码，具体内容如下：

常量	值	含义
RTC_CN_SERVER	0	中国
RTC_HK_SERVER	1	香港
RTC_US_SERVER	2	美国东部
RTC_SG_SERVER	3	新加坡
RTC_KR_SERVER	4	韩国
RTC_AU_SERVER	5	澳洲
RTC_DE_SERVER	6	德国
RTC_BR_SERVER	7	巴西
RTC_IN_SERVER	8	印度
RTC_JP_SERVER	9	日本
RTC_IE_SERVER	10	爱尔兰
RTC_USW_SERVER	11	美国西部
RTC_USM_SERVER	12	美国中部
RTC_CA_SERVER	13	加拿大
RTC_LON_SERVER	14	伦敦
RTC_FRA_SERVER	15	法兰克福
RTC_DXB_SERVER	16	迪拜
RTC_EXT_SERVER	10000	使用扩展服务器
RTC_DEFAULT_SERVER	10001	缺省服务器

7. 常见问题

7.1 连麦 SDK 需要替换以前的推流 SDK 吗？

是的，连麦 SDK 是在推流 SDK 基础上增加了连麦功能，替换掉推流 SDK 后，原有的推流 SDK 的 API 均可照常使用。

7.2 连麦窗口是否可以移动？

连麦的窗口是在本地的 XML 中配置的，可以随意配置位置和拖动，但是观众端从 CDN 观看的画面是由主播合流后的画面，不能移动。

7.3 推流是否也必须加入 "连麦房间" ？

推流依然沿用以前的流程，不用加入 “连麦房间”，调用连麦的时候，会在 SDK 内部 “自动” 加入房间。

7.4 如何优化带宽占用

连麦互动相比于原有的单向推流＋播放，增加了更多的带宽占用，因此需要合理的分配推流&连麦的参数，以达到最好的效果。

- 对于主播而言，同时需要 “推流”＋“连麦”，因此，主播端的上行带宽主要由两部分决定：“推流的码率” ＋ “连麦码率”
- 由于主播上行带宽有限，因此，连麦码率可以配置得相对低一点，从而优先保证推流的质量
- 连麦观众的码率可以配置得相对高一点，这样可以保证在主播端合流后的画面质量
- 推流的码率是在 `StreamingProfile` 中配置的，具体参考：《推流参数设置》
- 连麦的码率是在 `RTCConferenceOptions` 中配置的，具体参考：《3.4.3 RTCConferenceOptions》
- 推荐阅读：《码率、fps、分辨率对清晰度及流畅度的影响》

7.5 如何优化功耗

对于主播而言，同时需要 “推流”＋“连麦”，因此对手机的性能提出了更高的要求，我们也可以通过合理的参数配置，来适当减轻主播端的功耗压力。

- 合理配置连麦者的画面尺寸

  由于连麦对象的画面是小窗口，因此，对于连麦者，可以考虑使用比主播小一些的画面尺寸，这样可以显著降低主播端对连麦画面的剪裁压力。

  注：连麦画面尺寸的配置参考：《3.4.3.7 设置连麦编码器输出的画面尺寸级别》

- 合理配置主播端的合流参数

  主播端配置连麦合流窗口的尺寸尽量接近连麦者的画面尺寸，比如：如果连麦者输出的画面尺寸是 320 x 240 ，那么主播端配置该对象合流的尺寸就用 320 x 240 或者小于该尺寸的值，这样可以避免或者减少主播端对连麦画面进行拉伸，画面拉伸需要进行插值，非常消耗 CPU 。

  注：合流参数的配置参考：《3.4.4 RTCVideoWindow》

- 合理配置主播端预览尺寸

  主播端配置的预览尺寸尽量要匹配推流的尺寸，比如：如果推流的编码尺寸配置的是 848 x 480，那么预览时摄像头采集的尺寸也要尽量和 848 x 480 相匹配，这样可以避免或者减少主播端合流时对连麦画面进行拉伸，显著降低主播端的 CPU 和功耗，同时会提高连麦推流的流畅性。

- 适当降低连麦帧率

  连麦对讲的时候，画面很少剧烈运动，一般 15～20 帧的帧率足够了，降低帧率，可以显著降低 CPU 的处理压力，从而优化功耗。

  注：连麦帧率的配置参考：《3.4.3.5 设置连麦编码器输出的帧率》

7.6 如何重连

重连分为推流断线重连和连麦断线重连，下面分别描述：

7.6.1 推流断线如何重连

推流的状态消息会通过 `StreamingStateChangedListener` 进行回调， `startStreaming()` 如果无法连接上服务器，则会回调 `IOERROR` ，如果 `startStreaming()` 成功，则会进入 `STREAMING` 状态，此状态表示正在推流，如果发生断线，则会首先回调 `DISCONNECTED` 消息，再进入 `onRestartStreamingHandled` 回调。因此，关于重连的处理，我们需要关注 `IOERROR` 和 `DISCONNECTED` 两个消息。

收到 `DISCONNECTED` 消息之后，可以提示用户 "连接断开"，如果需要重连，则需要在 `StreamingSessionListener.onRestartStreamingHandled()` 函数中进行，重连的示例代码如下：

```
@Override
public boolean onRestartStreamingHandled(int code) {
    return mRTCStreamingManager.startStreaming();
}
```

当执行重连后，SDK 内部会进行 3 次连接尝试，如果 3 次尝试均失败，则会回调 `IOERROR` 状态。

收到 `IOERROR` 状态后，表明无法连接流媒体服务器，建议调用 `stopStreaming()` 停止推流，当然，你依然可以考虑继续尝试调用 `startStreaming()` 重连，不过建议如果在此状态下重连之前，检测一下网络联通性以及做一些 delay 操作。

7.6.2 连麦断线如何重连

连麦断线后，SDK 内部会自动重连，断线和重连的状态消息会通过 `RTCConferenceStateChangedListener` 进行回调， `RECONNECTING` 表示与服务器断线了，正在重连； `RECONNECTED` 表示与服务器连接成功；目前的机制是重连中没有主动调用 LeaveRoom 的话，会重试最多500次

7.7 如何后台推流&连麦

目前我们不支持在后台进行推流和连麦，这一点要注意，因此，在 `Activity` 进入 `pause` 状态后，需要关闭音视频采集、关闭推流、关闭连麦，恢复到前台后，再重新开始，进入后台的处理示例代码如下：

```
@Override
protected void onPause() {
    super.onPause();
    mRTCStreamingManager.stopCapture();
    mRTCStreamingManager.stopConference();
    mRTCStreamingManager.stopStreaming();
}
```

7.8 如何美颜

7.8.1 使用内部美颜

SDK 内部集成有美颜功能，通过如下方式开启：

```
CameraStreamingSetting cameraStreamingSetting = new CameraStreamingSetting();
// 使用内部美颜功能，自定义美颜的话记得关闭掉
cameraStreamingSetting.setBuiltInFaceBeautyEnabled(true);
// 设置美颜参数，三个参数分别代表：磨皮、美白、红润
cameraStreamingSetting.setFaceBeautySetting(new
cameraStreamingSetting.FaceBeautySetting(0.8f, 0.8f, 0.6f));
// 开启/关闭美颜效果
cameraStreamingSetting.setVideoFilter(CameraStreamingSetting.VIDEO_FILTER_TYPE.VIDE
O_FILTER_BEAUTY);
```

7.8.2 自定义美颜

自定义美颜需要调用 `setBuiltInFaceBeautyEnabled(false)` 先关闭 SDK 内部的美颜，然后分别处理预览显示的 filter 效果和编码的 filter 效果：预览显示通过实现 `SurfaceTextureCallback` 接口；编码通过实现 `StreamingPreviewCallback` 接口。两者分别实现，互不影响。

- 编码部分

```
public interface StreamingPreviewCallback {
    /**
     * 视频采集数据源回调
     *
     * @param data 采集到的 YUV 数据，格式由 fmt 决定
     * @param width 图像的宽
     * @param height 图像的高
     * @param rotation 图像的旋转角度
     * @param tsInNanoTime 时间戳
     * @param fmt 图像的格式，参考 {@link
com.qiniu.pili.droid.streaming.av.common.PLFourCC}
     * */
    boolean onPreviewFrame(byte[] data, int width, int height, int rotation, int
fmt, long tsInNanoTime);
}
```

`onPreviewFrame` 会回调 NV21 格式的 YUV 数据，进行 filter 算法处理后的结果仍然保存在 bytes 数组中，SDK 会将 bytes 中的数据当作数据源进行后续的编码和封包等操作；`onPreviewFrame` 运行在名称为 CameraManagerHt 的子线程中；onPreviewFrame 仅在 STATE.STREAMING 状态下被回调。

- 预览显示部分

```
public interface SurfaceTextureCallback {
    void onSurfaceCreated();
    void onSurfaceChanged(int width, int height);
    void onSurfaceDestroyed();
    int onDrawFrame(int texId, int width, int height);
}
```

四个回调均执行在 GL rendering thread；如果 onDrawFrame 直接返回 texId，代表放弃 filter 处理，否则 onDrawFrame 应该返回一个 filter 算法处理过的纹理 id； 自定义的 Texture id，即 onDrawFrame 返回的纹理 id， 必须是 GLES20.GL_TEXTURE_2D 类型；SDK 回调的纹理 id，即 onDrawFrame 的参数 texId 类型为 GLES11Ext.GL_TEXTURE_EXTERNAL_OES。

8. 常见的错误码

连麦常见的错误码对照表如下：

错误码	描述
0	SUCCESS
-2	ERROR_INVALID_ALG
-3	ERROR_ALREADY_INITIALIZED
-4	ERROR_NOT_INITIALIZED
-7	ERROR_WRONG_STATE
-11	ERROR_ROOM_TOKEN
-100	ERROR_OUT_OF_MEMORY
-101	ERROR_ENGINE_START_FAILED
-102	ERROR_ENGINE_STOP_FAILED
-103	ERROR_ILLEGAL_SDK
-104	ERROR_SERVER_INVALID
-105	ERROR_NETWORK_ERROR
-106	ERROR_SERVER_INNER_ERROR
-1000	ERROR_UNEXPECTED
2001	ERROR_UNKNOWN
2002	ERROR_NOT_JOIN_ROOM
2003	ERROR_CAMERA_NOT_READY
2004	ERROR_AUTH_DNSLOOKUP_FAILED
2005	ERROR_AUTH_CONNECT_FAILED
2006	ERROR_AUTH_HTTP_BAD_REQUEST
2007	ERROR_AUTH_HTTP_UNAUTHORIZED
2008	ERROR_AUTH_HTTP_NOT_FOUND
2009	ERROR_MALFORMEDURL_EXCEPTION
2010	ERROR_JSON_EXCEPTION
2011	ERROR_IO_EXCEPTION
2012	ERROR_SERVER_TIMEOUT

9. 历史记录

- **2.0.0**
  - 发布 pldroid-rtc-streaming-2.0.0.jar
  - 发布 libpldroid_rtc_streaming.so
  - 发布 libpldroid_mmprocessing.so
  - 发布 libpldroid_streaming_aac_encoder.so
  - 发布 libpldroid_streaming_amix.so
  - 发布 libpldroid_streaming_h264_encoder.so
  - 发布 libpldroid_streaming_core.so
  - 基本的推流和连麦对讲功能
  - 基本的视频合流和音频混音功能
  - 支持内置音视频采集，带美颜、水印、闪光灯、摄像头切换、聚焦等常见功能
  - 支持外部采集视频数据，支持的格式为：NV21 和 I420
  - 支持外部采集音频数据，支持的格式为：PCM，单通道，16bit 位宽
  - 支持外部美颜
  - 支持踢人功能
  - 支持静音功能
  - 支持连麦的帧率配置
  - 支持连麦的视频码率的配置
  - 支持连麦的视频尺寸配置
  - 支持丰富的连麦消息回调
  - 支持纯音频连麦
  - 支持连麦大小窗口切换
  - 支持推流的软硬编配置
  - 支持连麦的软硬编配置
  - 支持连麦画面的截帧
  - 支持动态镜像功能
  - 支持获取远端麦克风音量大小
  - 支持推流码率的自动调节/手动调节
  - 支持获取连麦房间统计信息（帧率、码率等）