直播 SDK

旧版文档

腾讯云

文档目录

旧版文档
SDK 集成
iOS

最近更新时间：2024-12-02 16:19:12

本文主要介绍如何快速地将 LiteAVSDK（iOS）集成到您的项目中，按照如下步骤进行配置，就可以完成 SDK 的集成工作。下面以全功能的 全功能版 SDK 为例：

开发环境要求

- Xcode 9.0+。
- iOS 9.0 以上的 iPhone 或者 iPad 真机。
- 项目已配置有效的开发者签名。

集成 LiteAVSDK

您可以选择使用 CocoaPods 自动加载的方式，或者先下载 SDK，再将其导入到您当前的工程项目中。

CocoaPods

1. 安装 CocoaPods

在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：

```
sudo gem install cocoapods
```

2. 创建 Podfile 文件

进入项目所在路径，输入以下命令行之后项目路径下会出现一个 Podfile 文件。

```
pod init
```

3. 编辑 Podfile 文件

编辑 Podfile 文件，有如下有两种设置方式：

- 方式一：使用腾讯云 LiteAVSDK 的 podspec 文件路径。

```
  platform :ios, '9.0'

  target 'App' do
  pod 'TXLiteAVSDK_Professional', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_Professional.podspec'
  end
```

- 方式二：使用 CocoaPod 官方源，支持选择版本号。

```
   platform :ios, '9.0'
   source 'https://github.com/CocoaPods/Specs.git'

   target 'App' do
   pod 'TXLiteAVSDK_Professional'
```

```
    end
```

4. 更新并安装 SDK

在终端窗口中输入如下命令以更新本地库文件，并安装 LiteAVSDK：

```
pod install
```

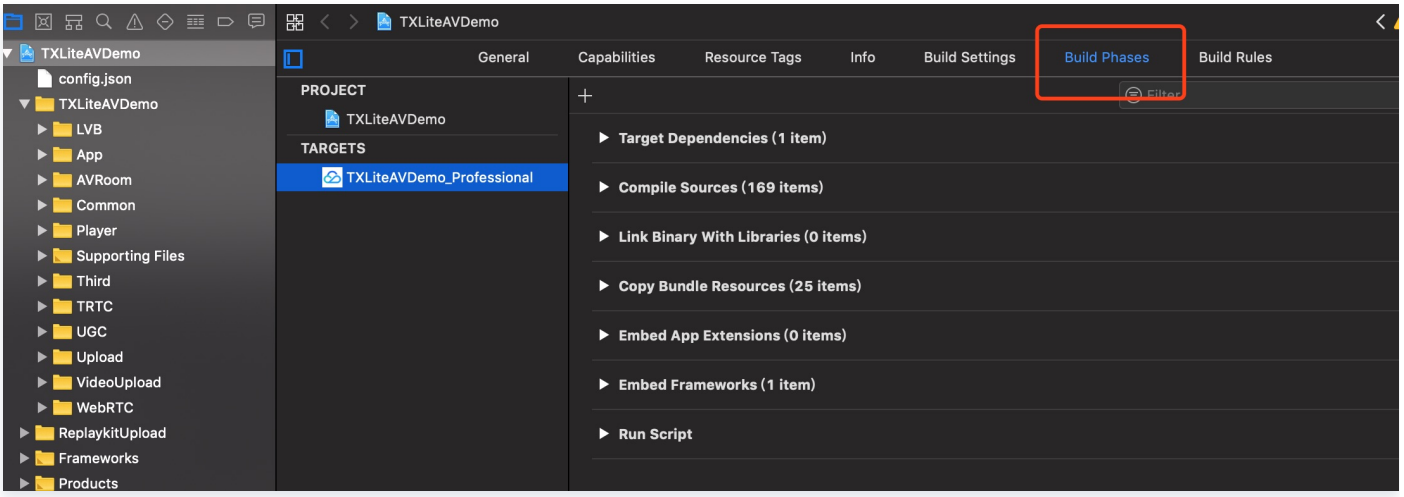
或使用以下命令更新本地库版本：

```
pod update
```

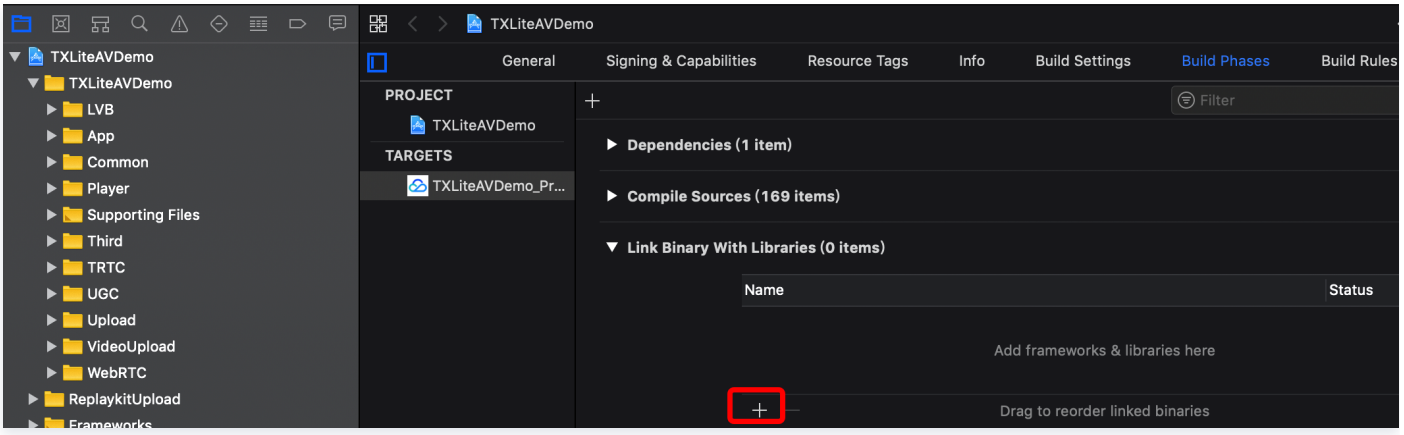
pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

手动集成

1. 下载 LiveAVSDK ，下载完成后进行解压。
2. 打开您的 Xcode 工程项目，选择要运行的 target , 选中 **Build Phases** 项。



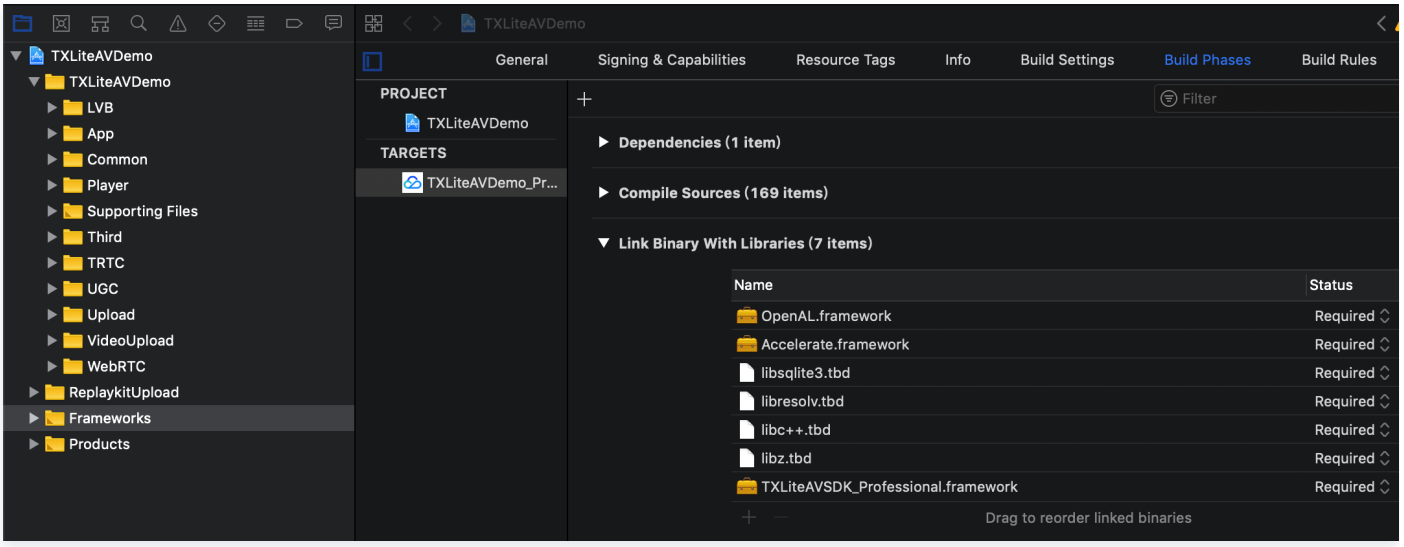
3. 单击 **Link Binary with Libraries** 项展开，单击底下的“+”添加依赖库。



4. 依次添加所下载的 `TXLiteAVSDK_Professional.framework` 及其所需依赖库：

```
libz.tbd
libc++.tbd
libresolv.tbd
libsqlite3.tbd
```

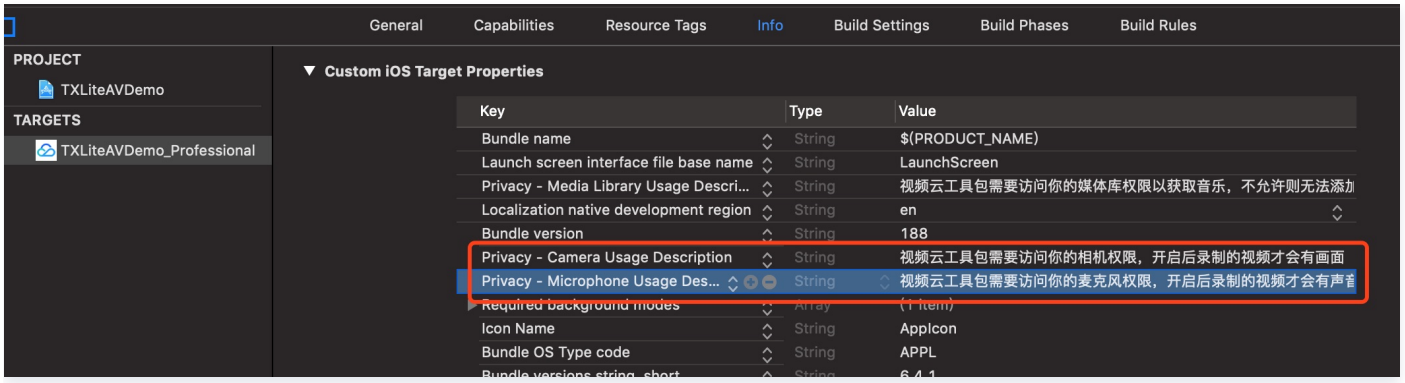
```
Accelerate.framework
OpenAL.framework
```



授权摄像头和麦克风使用权限

使用 SDK 的音视频功能，需要授权麦克风和摄像头的使用权限。在 App 的 Info.plist 中添加以下两项，分别对应麦克风和摄像头在系统弹出授权对话框时的提示信息。

- **Privacy – Microphone Usage Description**，并填入麦克风使用目的提示语。
- **Privacy – Camera Usage Description**，并填入摄像头使用目的提示语。



在工程中引入 SDK

项目代码中使用 SDK 有两种方式：

- **方式一：** 在项目需要使用 SDK API 的文件里，添加模块引用。

```
@import TXLiteAVSDK_Professional;
```

- **方式二：** 在项目需要使用 SDK API 的文件里，引入具体的头文件。

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

给 SDK 配置 License 授权

单击 License 申请 获取测试用 License，您会获得两个字符串：一个字符串是 licenseURL，另一个字符串是解密 key。

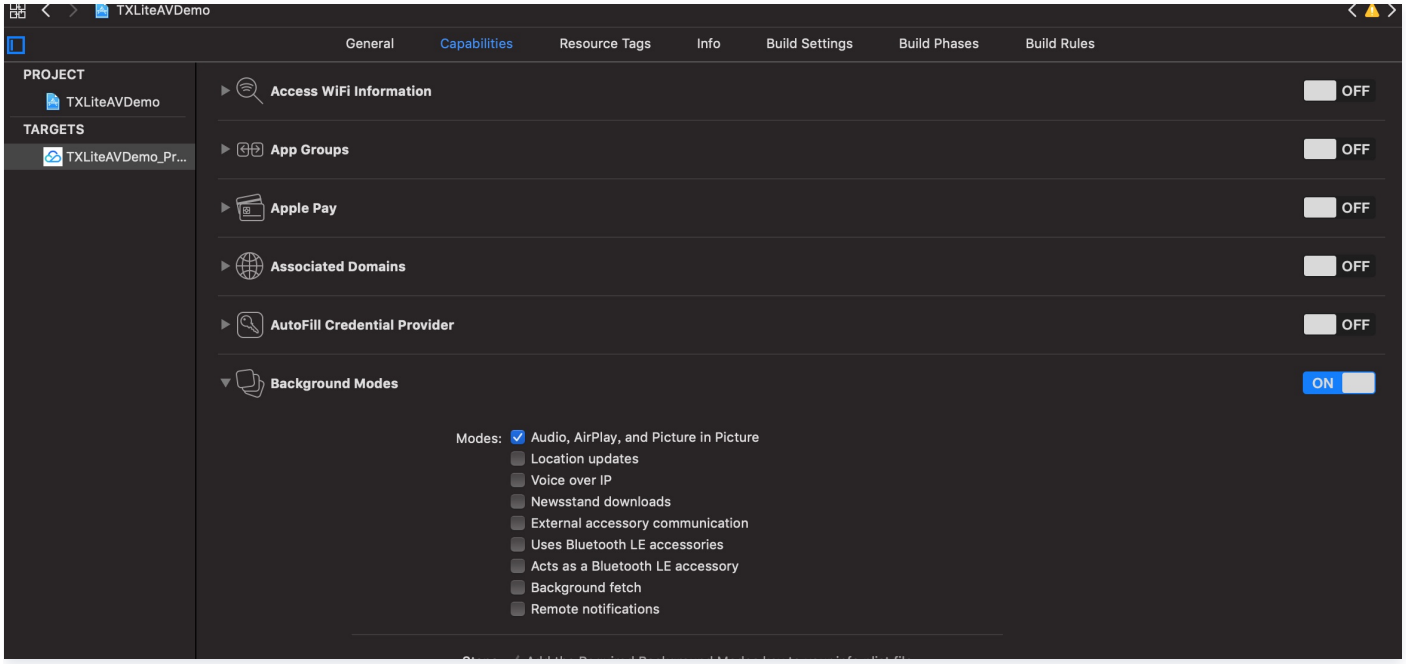
在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

    //TXLiveBase 位于 "TXLiveBase.h" 头文件中
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

常见问题

LiteAVSDK 是否支持后台运行?

支持，如需要进入后台仍然运行相关功能，可选中当前工程项目，在 Capabilities 下设置 Background Modes 为 ON，并勾选 Audio，AirPlay and Picture in Picture，如下图所示：



Android

最近更新时间：2025-06-17 18:11:22

本文主要介绍如何快速地将腾讯云 LiteAVSDK（Android）集成到您的项目中，按照如下步骤进行配置，就可以完成 SDK 的集成工作。

开发环境要求

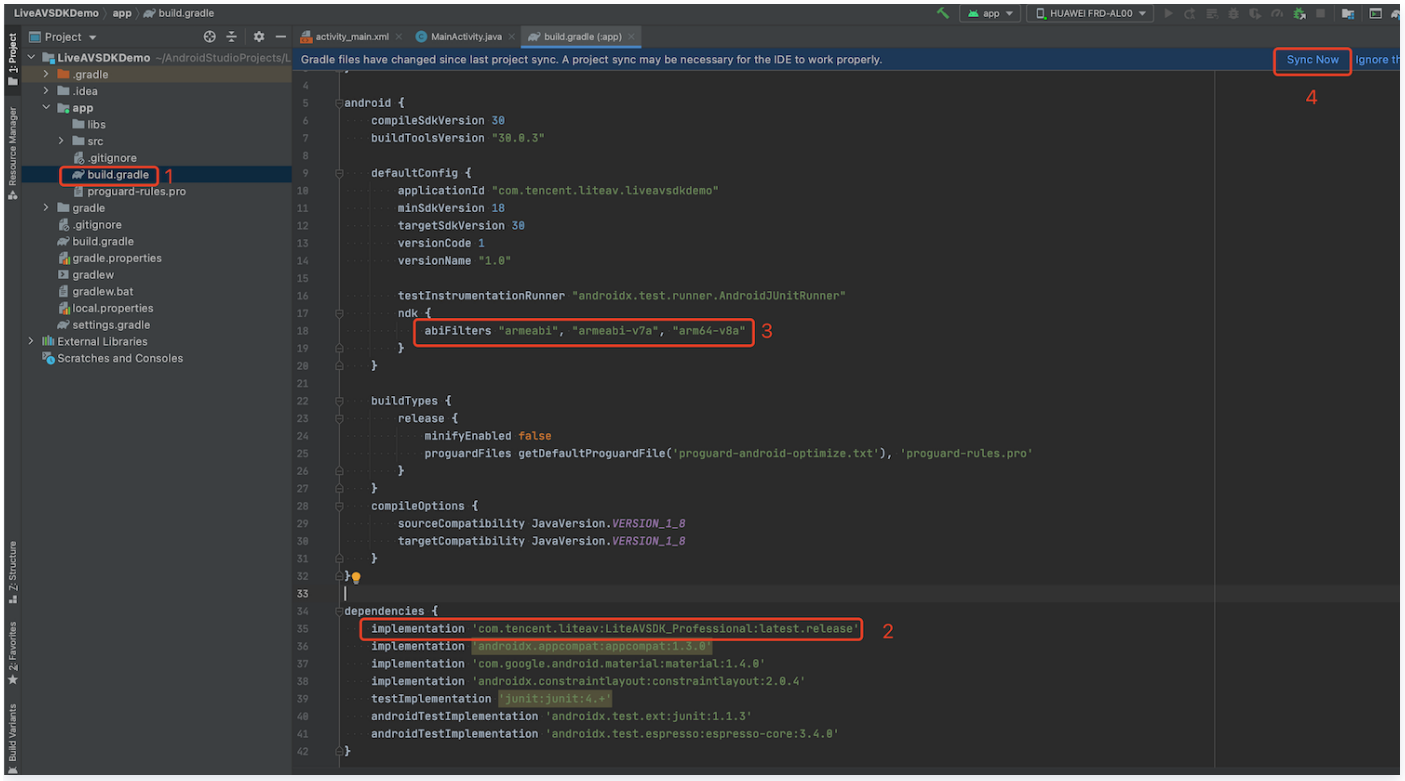
- Android Studio 2.0+。
- Android 4.1（SDK API 16）及以上系统。

集成 SDK（aar）

您可以选择使用 Gradle 自动加载的方式，或者手动下载 aar 再将其导入到您当前的工程项目中。

方法一：自动加载（aar）

LiteAVSDK 已经发布到 jcenter 库，您可以通过配置 gradle 自动下载更新。
只需要用 Android Studio 打开需要集成 SDK 的工程，然后通过简单的四个步骤修改 `app/build.gradle` 文件，就可以完成 SDK 集成：



1. 打开 app 下的 build.gradle。
2. 在 dependencies 中添加 LiteAVSDK 的依赖。

```
dependencies {
        implementation 'com.tencent.liteav:LiteAVSDK_Smart:latest.release'
}
```

或

```
dependencies {
        implementation 'com.tencent.liteav:LiteAVSDK_Smart:latest.release@aar'
}
```

3. 在 defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、 armeabi-v7a 和 arm64-v8a）。

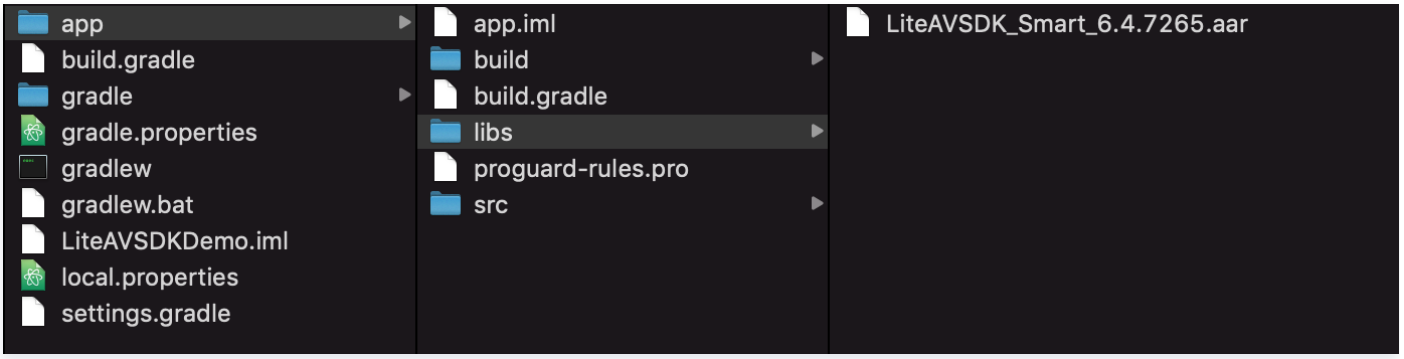
```
defaultConfig {
      ndk {
          abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
      }
}
```

4. 单击 Sync Now 按钮同步 SDK，如果您的网络连接 jcenter 没有问题，很快 SDK 就会自动下载集成到工程里。

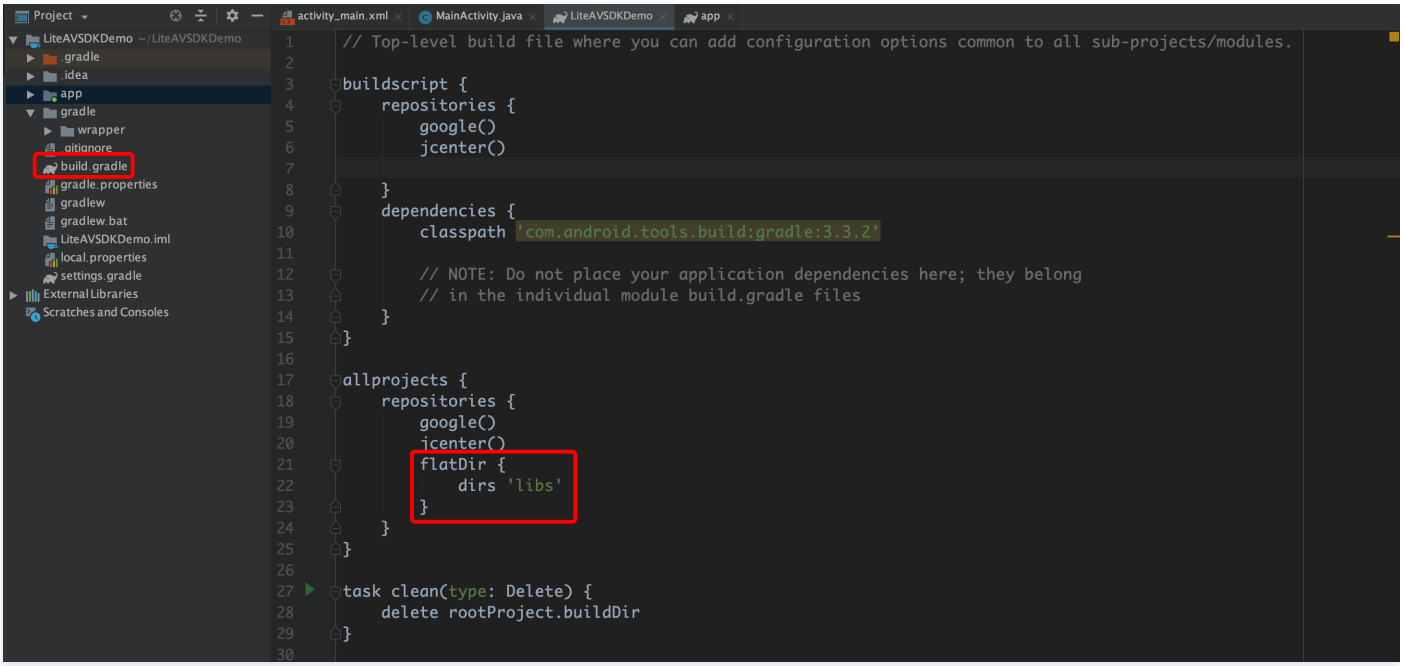
方法二：手动下载（aar）

如果您的网络连接 jcenter 有问题，也可以手动下载 SDK 集成到工程里：

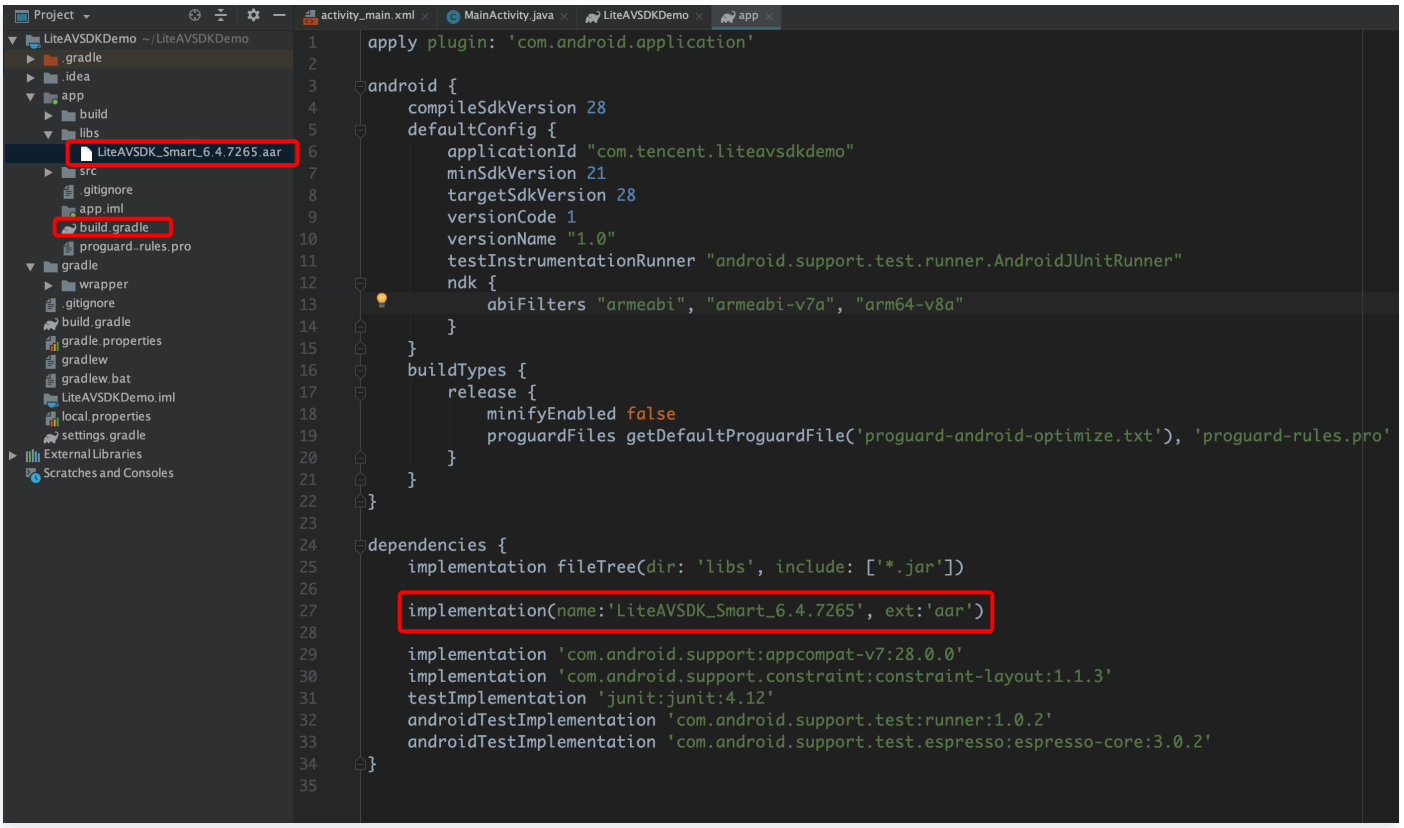
1. 下载 LiveAVSDK ，下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：



3. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



4. 添加 LiteAVSDK 依赖，在 app/build.gradle 中，添加引用 aar 包的代码。




```
implementation(name:'LiteAVSDK_Smart_6.4.7265', ext:'aar')
```

5. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、armeabi-v7a 和 arm64-v8a）。

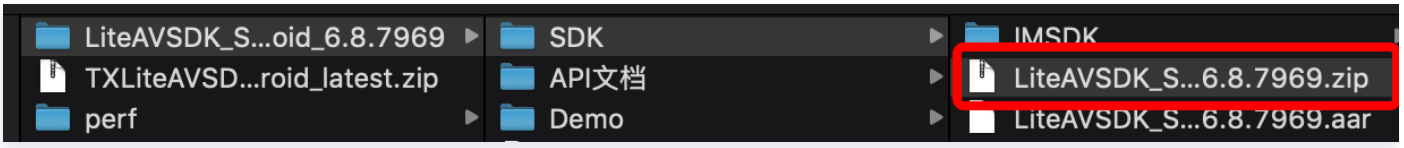
```
defaultConfig {
      ndk {
          abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
      }
  }
```

6. 单击 Sync Now 按钮同步 SDK，完成 LiteAVSDK 的集成工作。

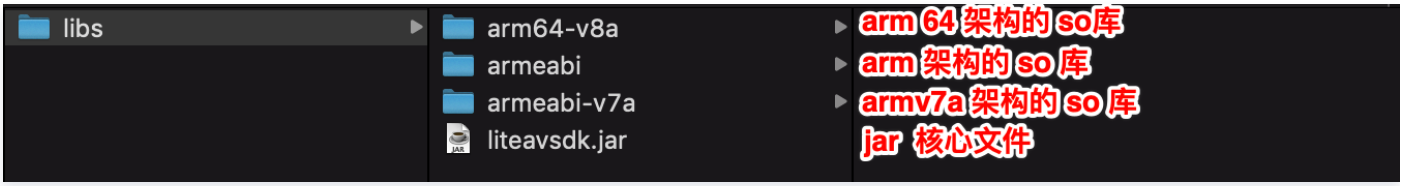
集成 SDK（jar）

如果您不想集成 aar 库，也可以通过导入 jar 和 so 库的方式集成 LiteAVSDK：

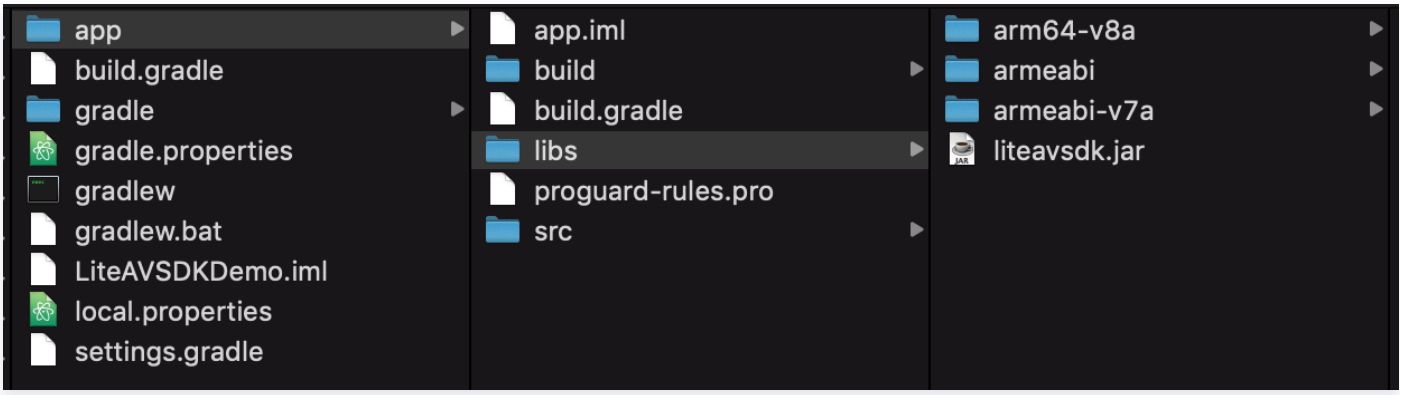
1. 下载 LiveAVSDK ，下载完成后进行解压。在 SDK 目录下找到 `LiteAVSDK_Smart_xxx.zip`（其中 xxx 为 LiteAVSDK 的版本号）：



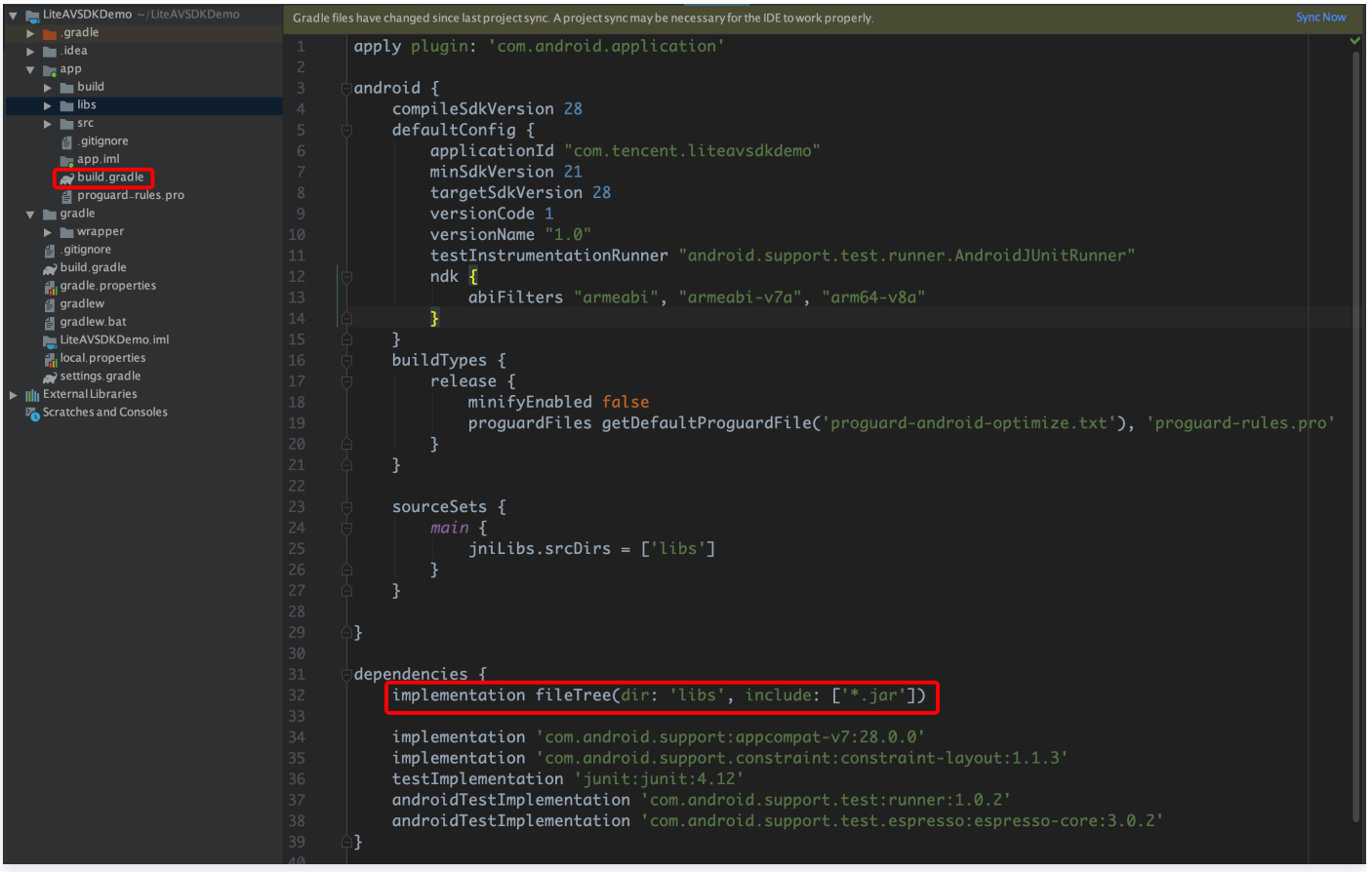
解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：



2. 将解压得到的 jar文件和 armeabi、armeabi-v7a、arm64-v8a 文件夹拷贝到 app/libs 目录下。

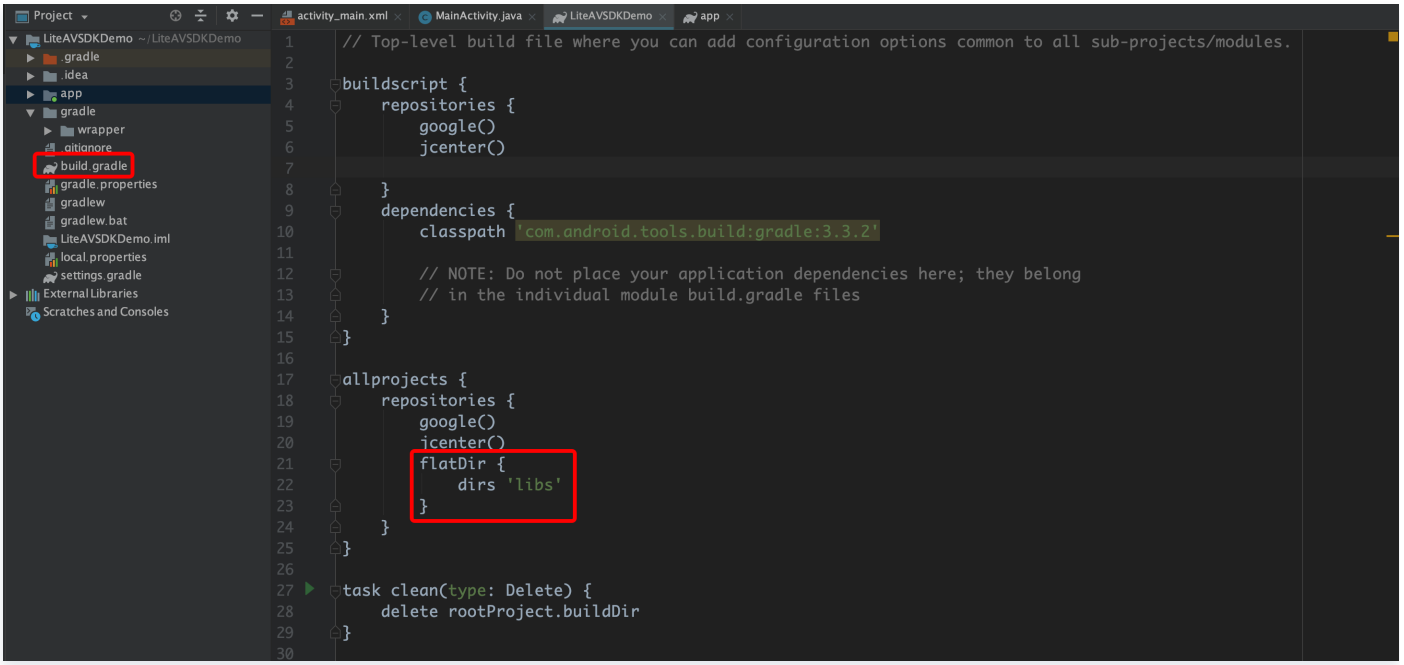


3. 在 app/build.gradle 中，添加引用 jar 库的代码。

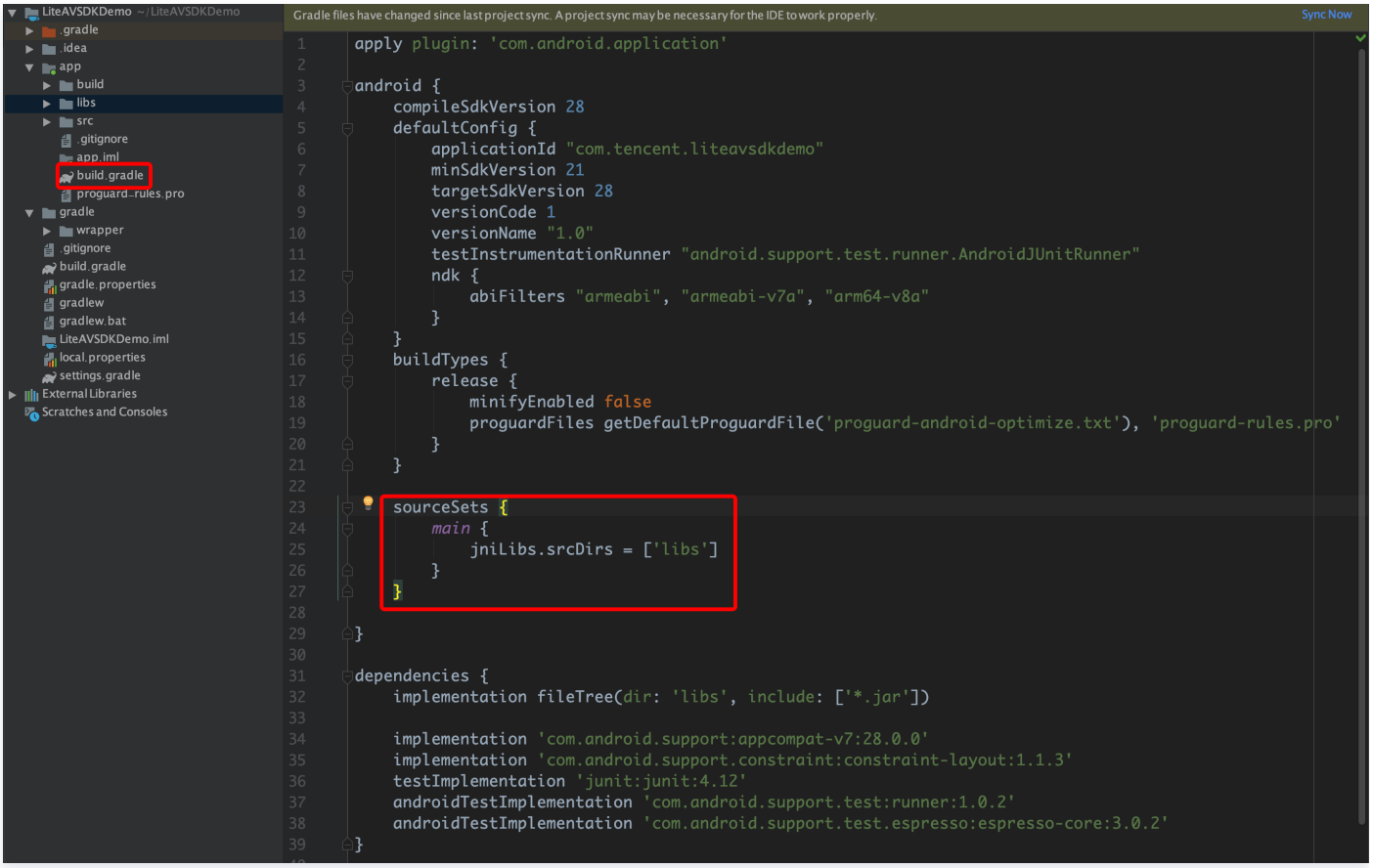



```
dependencies{
    implementation fileTree(dir:'libs',include:['*.jar'])
}
```

4. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



5. 在 app/build.gradle 中，添加引用 so 库的代码。



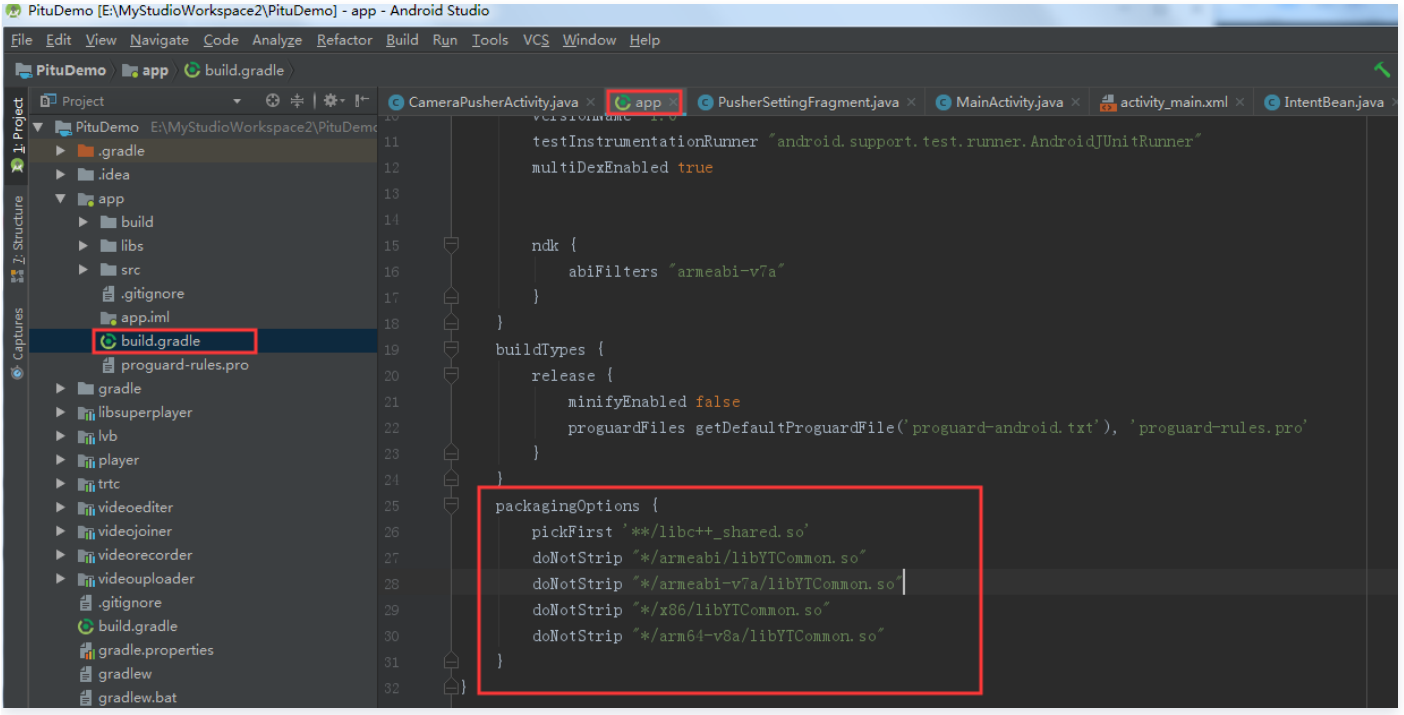
6. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构(目前 LiteAVSDK 支持 armeabi 、 armeabi-v7a 和 arm64-v8a) 。

```
defaultConfig {
```

```
    ndk {
        abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
    }
}
```

7. 单击 Sync Now 按钮同步 SDK，完成 LiteAVSDK 的集成工作。

配置 App 打包参数




```
packagingOptions {
    pickFirst '**/libc++_shared.so'
    doNotStrip "*/armeabi/libYTCommon.so"
    doNotStrip "*/armeabi-v7a/libYTCommon.so"
    doNotStrip "*/x86/libYTCommon.so"
    doNotStrip "*/arm64-v8a/libYTCommon.so"
}
```

配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，LiteAVSDK 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.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-feature android:name="android.hardware.Camera"/>
```

```
<uses-feature android:name="android.hardware.camera.autofocus" />
```

配置 License 授权

单击 License 申请 获取测试用 License，具体操作请参见 测试版 License 。您会获得两个字符串：一个字符串是 licenseURL，另一个字符串是解密 key。
在您的 App 调用企业版 SDK 相关功能之前（建议在 Application类中）进行如下设置：

```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // 获取到的 licence url
        String licenceKey = ""; // 获取到的 licence key
        TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
    }
}
```

> **注意**
> 企业版已不对外提供，美颜相关功能可参见 腾讯特效 SDK（美颜 SDK）。

设置混淆规则

在 proguard-rules.pro 文件中，将 LiteAVSDK 相关类加入不混淆名单：

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

功能说明

最近更新时间：2024-12-10 16:08:53

移动直播 SDK 提供4个版本，分别是**基础直播 Smart**、**互动直播 Live**、**全功能版 ALL** 和**企业版 Enterprise**，可通过移动直播 License 使用相应的直播功能。

SDK 版本对应关系

- **基础直播 Smart**：通过直播推流 License（原移动直播的基础版 License），可使用直播推流、直播播放和基础美颜（美白、磨皮等）功能。
- **互动直播 Live**：在基础直播（Smart）的功能之上增加了连麦相关的功能支持，满足互动连麦直播场景需求。
- **全功能版 ALL**：多个音视频产品的合集，除了增加了对快直播的支持，其他功能上和互动直播（Live）的没有区别。主要是为了解决同时集成多个音视频产品时的符号冲突问题，通过直播推流 License（原移动直播的基础版 License）可使用直播推流相关功能，其他产品功能需要开通对应服务。
- **企业版 Enterprise**：在全功能版 SDK 上增加了高级美颜特效能力，需要购买移动直播企业版 License 来使用高级美颜特效等功能。

> **注意：**
> 移动直播企业版 License 可额外购买美妆特效或者手势动效素材，购买后可使用美妆特效或者手势动效的功能，详情请参见 价格总览 。

SDK 与授权 License 对应关系

- 基础直播 Smart、互动直播 Live 和全功能版 ALL 可 申请测试的直播推流 License（原移动直播的基础版 License），或 购买正式的直播推流 License（原移动直播的基础版 License） 进行使用。
- 企业版 Enterprise 使用请先 申请企业版 License，或者 联系腾讯云商务 进行申请。

> **注意：**
> - 若在企业版 Enterprise 上使用基础版 License，则只能使用移动直播基础版部分的功能。
> - 若在基础直播 Smart 上使用企业版 License，则无法使用高级美颜特效等功能。

SDK 功能列表

功能模块	功能项	功能简介	基础直播 Smart	互动直播 Live	全功能版 All	企业版 Enterprise

界面	自定义 UI	开发者自定义 UI。小直播 App 提供了一套完整的 UI 交互源码，可复用或自定义。	✓	✓	✓	✓
直播推流	RTMP 推流	用于实现主播端的手机推流功能（秀场直播）。	✓	✓	✓	✓
	录屏推流	用于实现主播端的屏幕推流功能（游戏直播）。	✓	✓	✓	✓
直播播放	RTMP 播放	用于实现 RTMP 协议的直播播放功能。	✓	✓	✓	✓
	FLV 播放	用于实现 HTTP + FLV 协议的直播播放功能。	✓	✓	✓	✓
	HLS 播放	用于实现 HLS（m3u8）协议的直播播放功能。	✓	✓	✓	✓
	WebRTC 播放	用于实现快直播（LEB）的直播播放功能。	×	✓	✓	✓
点播播放	点播播放	用于实现视频点播回放功能。	✓（不支持 FLV 格式）	×	✓	✓
直播连麦	连麦互动	用于实现主播与观众之间的1vn视频连麦互动。	×	✓	✓	✓
	主播 PK	用于实现主播与主播之间的1v1视频 PK。	×	✓	✓	✓
采集拍摄	屏比	支持16:9，4:3，1:1多种屏比拍摄。	✓	✓	✓	✓
	清晰度	支持标清、高清及超清拍摄，支持自定义码率、帧率及 gop。	✓	✓	✓	✓
	拍摄控制	拍摄前后摄像头切换和灯光的控制。	✓	✓	✓	✓
	水印	拍摄支持添加水印。	✓	✓	✓	✓
	焦距	拍摄支持调节焦距。	✓	✓	✓	✓
	对焦模式	支持手动对焦和自动对焦。	✓	✓	✓	✓
	拍照	支持拍摄照片。	✓	✓	✓	✓
	背景音乐	拍摄前可以选择本地的 MP3 作为背景音。	✓	✓	✓	✓
	变声和混响	拍摄前对录制的声音变声（如萝莉、大叔）和混响效果（如 KTV、会堂）。	×	×	✓	✓
	滤镜	支持自定义滤镜及设置滤镜程度。	✓	✓	✓	✓
	基础美颜	拍摄设置面部磨皮、美白、红润并调节强度。	✓	✓	✓	✓
	高级美颜	拍摄设置大眼、瘦脸、V 脸、下巴调整、短脸、小鼻效果，并支持调节强度。	×	×	×	✓
	动效贴纸	定位五官位置，然后添加变形、覆盖贴纸挂件等效果。（通过素材提供）	×	×	×	✓
	AI 抠图	识别出背景轮廓，把背景抠除，替换成素材视频的元素。（通过素材提供）	×	×	×	✓

	绿幕抠像	将画面中的绿色元素（如纯绿背景）抠除，替换成其他的元素。	×	×	×	✓

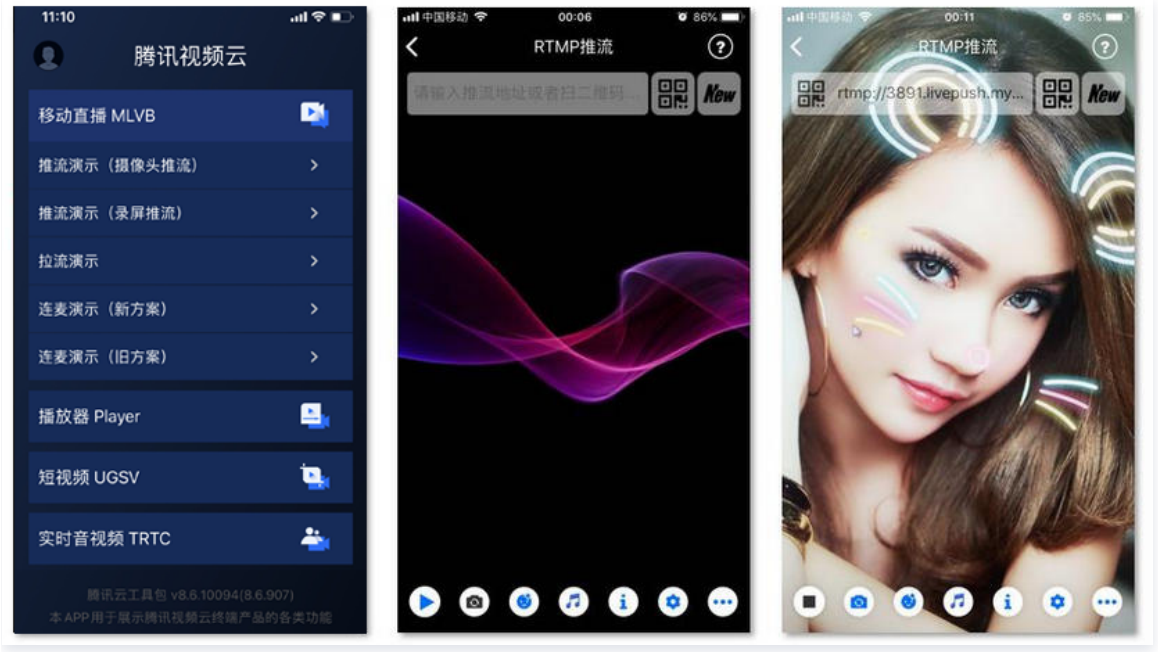
注意：
若需要使用美妆特效或手势动效功能，需在购买移动直播企业版 License 基础上，额外购买美妆特效者手势动效素材，相应价格请参见 价格总览 。

基础功能
摄像头推流
iOS

最近更新时间：2025-01-21 10:42:32

功能概述

摄像头推流，是指采集手机摄像头的画面以及麦克风的声音，进行编码之后再推送到直播云平台上。腾讯云 LiteAVSDK 通过 TXLivePusher 接口提供摄像头推流能力，如下是 LiteAVSDK 的简单版 Demo 中演示摄像头推流的相关操作界面：



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` 为 false。但当您要推流的地址为腾讯云地址时，请务必在推流前将其设置为 YES，否则 SDK 针对腾讯云的协议优化将不能发挥作用。
- **x86 模拟器调试**
  由于 SDK 大量使用 iOS 系统的音视频接口，这些接口在 Mac 上自带的 x86 仿真模拟器下往往不能工作。所以，如果条件允许，推荐您尽量使用真机调试。

示例代码

所属平台	GitHub 地址	关键类
iOS	Github	CameraPushViewController.m
Android	Github	CameraPushImpl.java

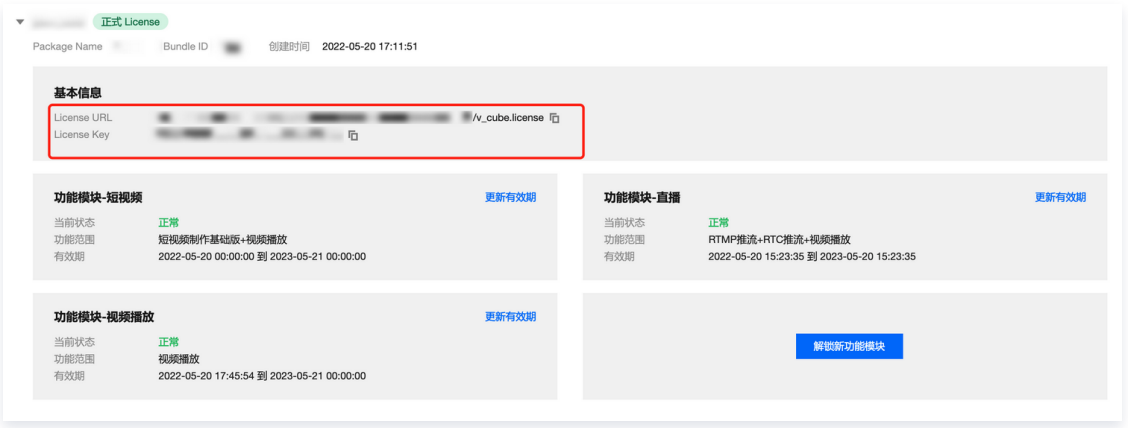
功能对接

1. 下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

2. 给 SDK 配置 License 授权

1. 获取 License 授权：
   - 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。



   - 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。
2. 在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

    //TXLiveBase 位于 "TXLiveBase.h" 头文件中
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

> ⚠ **注意：**
> License 中配置的 BundleId 必须和应用本身一致，否则会推流失败。

3. 初始化 TXLivePush 组件

首先创建一个 `TXLivePushConfig` 对象。该对象可以指定一些高级配置参数，但一般情况下我们不建议您操作该对象，因为我们已经在其内部配置好了所有需要校调的参数。之后再创建一个 `TXLivePush` 对象，该对象负责完成推流的主要工作。

```
TXLivePushConfig *_config = [[TXLivePushConfig alloc] init];  // 一般情况下不需要修改默认 config

TXLivePush *_pusher = [[TXLivePush alloc] initWithConfig: _config]; // config 参数不能为空
```

4. 开启摄像头预览

调用 TXLivePush 中的 `startPreview` 接口可以开启当前手机的摄像头预览。您需要为 `startPreview` 接口提供一个用于显示视频画面的 view 对象。

```
//创建一个 view 对象，并将其嵌入到当前界面中
UIView *_localView = [[UIView alloc] initWithFrame:self.view.bounds];
```

```
[self.view insertSubview:_localView atIndex:0];
_localView.center = self.view.center;

//启动本地摄像头预览
[_pusher startPreview:_localView];
```

> ⚠ **注意：**
> 如果要给 view 增加动画效果，需要修改 view 的 transform 属性而不是 frame 属性。

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

5. 启动和结束推流

如果已经通过 `startPreview` 接口启动了摄像头预览，就可以调用 TXLivePush 中的 `startPush` 接口开始推流。

```
//启动推流
NSString* rtmpUrl = @"rtmp://test.com/live/xxxxxx";     //此处填写您的 rtmp 推流地址
[_pusher startPush:rtmpUrl];
```

推流结束后，可以调用 TXLivePush 中的 `stopPush` 接口结束推流。请注意，如果已经启动了摄像头预览，请在结束推流时将其关闭，否则会导致 SDK 的表现异常。

```
//结束推流
[_pusher stopPreview]; //如果已经启动了摄像头预览，请在结束推流时将其关闭。
[_pusher stopPush];
```

- **获取可用的推流 URL**
  开通直播服务后，可以使用**直播控制台 > 辅助工具 > 地址生成器** 生成推流地址，详细信息请参见 推拉流 URL 。

- **返回 -5 的原因**
  如果 `startPush` 接口返回 -5，则代表您的 License 校验失败了，请检查 第2步“给 SDK 配置 License 授权” 中的工作是否有问题。

6. 纯音频推流

如果您的直播场景是纯音频直播，不需要视频画面，那么您可以不执行 第4步 中的操作，取而代之的是开启 TXLivePushConfig 中的 `enablePureAudioPush` 配置。

```
//通过修改 enablePureAudioPush 开关，开启纯音频推流
TXLivePushConfig *_config = [[TXLivePushConfig alloc] init];
_config.enablePureAudioPush = YES;    // YES 为启动纯音频推流，只有在调用 startPush 前设置才会生效。

TXLivePush *_pusher = [[TXLivePush alloc] initWithConfig: _config];
NSString* rtmpUrl = @"rtmp://test.com/live/xxxxxx";
[_pusher startPush:rtmpUrl];
```

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

7. 设定画面清晰度

调用 TXLivePush 中的 `setVideoQuality` 接口，可以设定观众端的画面清晰度。之所以说是观众端的画面清晰度，是因为主播看到的视频画面是未经编码压缩过的高清原画，不受设置的影响。而 `setVideoQuality` 通过设定视频编码器的编码质量，使观众端感受到画质的差异。详情请参考 设定画面质量 。

取值	含义
YES	弱网状态时降低画质，画面较流畅
NO	弱网状态时保持画质，画面较卡顿

待废弃，推荐设置为 NO

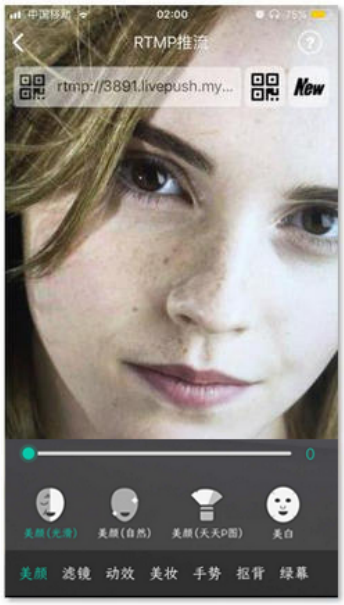
TXLivePusher # setVideoQuality (quality, adjustBitrate, *adjustResolution*);

画质	定义	说明
标清	VIDEO_QUALITY_STANDARD_DEFINITION	采用 360x640 的分辨率
高清	VIDEO_QUALITY_HIGH_DEFINITION	采用 540x960 的分辨率
超清	VIDEO_QUALITY_SUPER_DEFINITION	采用 720x1280 的分辨率
连麦（主）	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	进入连麦状态后，大主播使用
连麦（副）	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	进入连麦状态后，副主播使用
视频通话	VIDEO_QUALITY_REALTIME_VIDEOCHAT	已经废弃

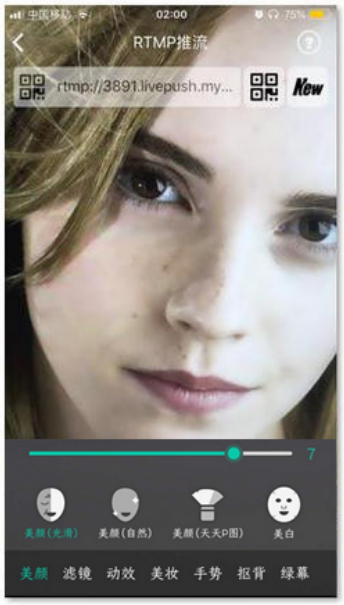
8. 美颜美白和红润特效

调用 TXLivePush 中的 `setBeautyStyle` 接口可以设置美颜效果，SDK 中提供了两种磨皮算法（beautyStyle）：

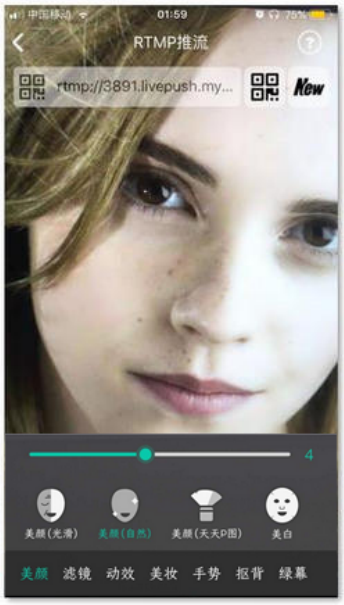
美颜风格	效果说明
BEAUTY_STYLE_SMOOTH	光滑风格，算法更加注重皮肤的光滑程度，适合秀场直播类场景下使用。
BEAUTY_STYLE_NATURE	自然风格，算法更加注重保留皮肤细节，适合对真实性要求更高的主播。



美颜关闭



美颜风格：光滑

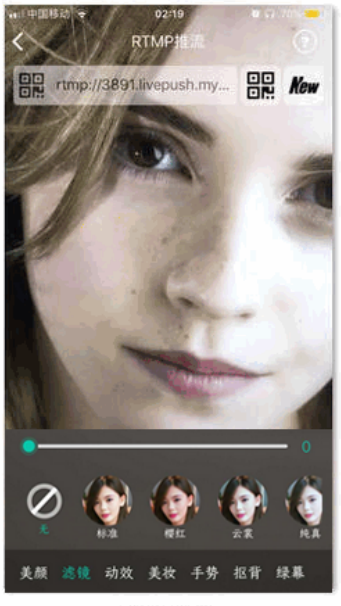


美颜风格：自然

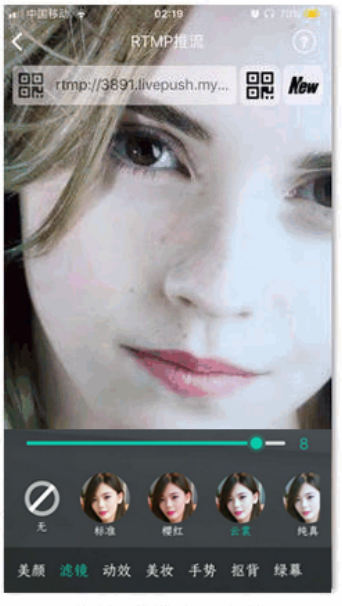
```
//      beautyStyle     : 美颜算法，目前支持 自然 和 光滑 两种。
//      beautyLevel     : 磨皮级别，取值范围 0 - 9；  0 表示关闭 1 - 9值越大 效果越明显。
//      whitenessLevel  : 美白级别，取值范围 0 - 9；  0 表示关闭 1 - 9值越大 效果越明显。
//      ruddinessLevel  : 红润级别，取值范围 0 - 9；  0 表示关闭 1 - 9值越大 效果越明显。
(void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel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

9. 色彩滤镜效果

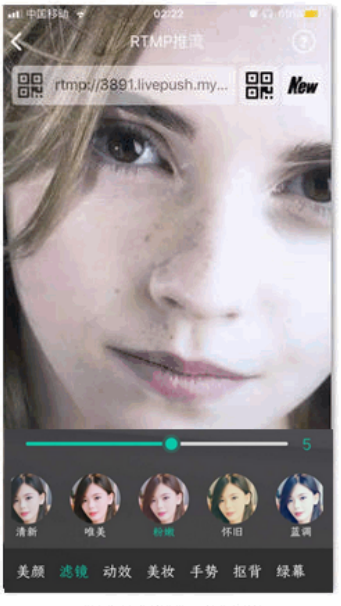
调用 TXLivePush 中的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
调用 TXLivePush 中的 `setSpecialRatio` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。
从手机 QQ 和 Now 直播的经验来看，单纯通过 `setBeautyStyle` 调整磨皮效果是不够的，只有将美颜效果和 `setFilter` 配合使用才能达到更加多变的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜，并将其默认打包在 Demo 中供您使用。



关闭滤镜



开启滤镜：云裳



开启滤镜：粉嫩

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:lookupFileName];

UIImage *image = [UIImage imageWithContentsOfFile:path];
[_pusher setFilter:image];
[_pusher setSpecialRatio:0.5f];
```

10. 控制摄像头

TXLivePush 提供了一组 API 用户控制摄像头的行为：

API 函数	功能说明	备注说明
switchCamera	切换前后摄像头	Mac 平台对应的函数为 `selectCamera`。
toggleTorch	打开或关闭闪光灯	仅在当前是后置摄像头时有效。
setZoom	调整摄像头的焦距	焦距大小，取值范围：1 – 5，默认值建议设置为1即可。
setFocusPosition	设置手动对焦位置	需要将 TXLivePushConfig 中的 `touchFocus` 选项设置为 YES 后才能使用。

11. 观众端的镜像效果

调用 TXLivePush 中的 `setMirror` 接口可以设置观众端的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，自己看到的画面会被 SDK 默认反转，这时主播的体验跟自己照镜子时的体验是保持一致的。`setMirror` 所影响的是观众端观看到的画面情况，如下图所示：

12. 横屏推流

大多数情况下，主播习惯以“竖屏持握”手机进行直播拍摄，观众端看到的也是竖屏分辨率的画面（例如 540 × 960 这样的分辨率）；有时主播也会“横屏持握”手机，这时观众端期望能看到是横屏分辨率的画面（例如 960 × 540 这样的分辨率），如下图所示：

TXLivePush 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，需要：

1. 设置 TXLivePushConfig 中的 `homeOrientation` 接口可以改变观众端看到的视频画面宽高比方向。
2. 调用 TXLivePush 中的 `setRenderRotation` 接口可以改变主播端的视频画面的渲染方向。

```
// 如果希望竖屏推流（HOME 键在下），这是 SDK 的默认行为
_config.homeOrientation = HOME_ORIENTATION_DOWN;
[_pusher setConfig:_config];
[_pusher setRenderRotation:0];

// 如果希望横屏推流（HOME 键在右）
_config.homeOrientation = HOME_ORIENTATION_RIGHT;
[_pusher setConfig:_config];
[_pusher setRenderRotation:90];
```

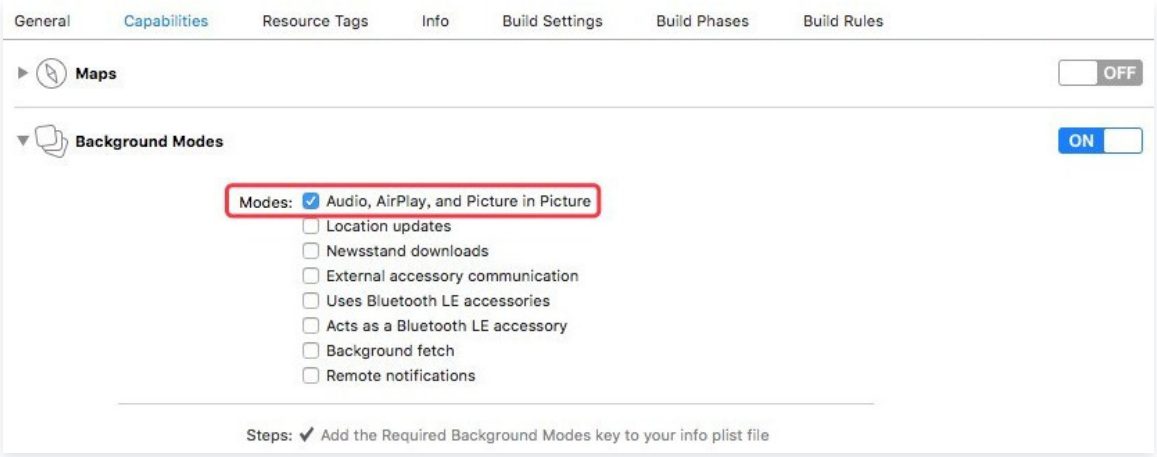
13. 隐私模式（垫片推流）

有时候主播的一些动作不希望被观众看到，但直播过程中又不能下播，那就可以考虑进入隐私模式。在隐私模式下，SDK 可以暂停采集主播手机的摄像头画面以及麦克风声音，并使用一张默认图片作为替代图像进行推流，也就是所谓的“垫片”。

该功能也常用于 App 被切到后台时：在 iOS 系统中，当 App 切到后台以后，操作系统不再允许该 App 继续采集摄像头画面。此时就可以通过调用 `pausePush` 进入垫片状态。因为对于大多数直播 CDN 而言，如果超过一定时间（腾讯云目前为70s）不推视频数据，服务器就会断开当前的推流

链接，所以在 App 切到后台后进入垫片模式是很有必要的。

- **step1: 开启 XCode 中的 Background 模式**



- **step2: 设置 TXLivePushConfig 中的相关参数**

  在开始推流前，使用 LivePushConfig 的 `pauseImg` 、 `pauseFps` 和 `pauseTime` 接口可以设置垫片推流的详细参数：

```
TXLivePushConfig *_config = [[TXLivePushConfig alloc] init];
// 设置后台推流持续时长，单位秒，默认300秒。
_config.pauseTime = 300;
// 设置后台推流帧率，最小值为5，最大值为20，默认为10。
_config.pauseFps = 10;
// 设置后台推流的默认图片，默认为黑色背景，图片最大尺寸不能超过1920*1920。
_config.pauseImg = [UIImage imageNamed:@"pause_publish.jpg"];

TXLivePush *_pusher = [[TXLivePush alloc] initWithConfig: _config];
```

- **step3: 监听 App 的前后台切换事件**

  如果 App 在切到后台后就被 iOS 系统彻底休眠掉，SDK 将无法继续推流，观众端就会看到主播画面进入黑屏或者冻屏状态。您可以使用下面的代码让 App 在切到后台后还可再跑几分钟。

```
// 注册应用监听事件
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
```

```
[center addObserver:self selector:@selector(willResignActive:)
name:UIApplicationWillResignActiveNotification object:nil];
[center addObserver:self selector:@selector(didBecomeActive:)
name:UIApplicationDidBecomeActiveNotification object:nil];


// 具体实现，_livePuser 为当前TXLivePush实例对象
#pragma mark - 前后台切换
- (void)willResignActive:(NSNotification *)notification {
    [_livePusher pausePush];
    _inBackground = YES;
}

- (void)didBecomeActive:(NSNotification *)notification {
    [_livePusher resumePush];
    _inBackground = NO;
    // 其他唤醒业务逻辑
}
```

注意：
请注意调用顺序：startPush => (pausePush => resumePush) => stopPush，错误的调用顺序会导致 SDK 表现异常，因此使用成员变量对执行顺序进行保护是很有必要的。

14. 背景混音

调用 TXLivePush 中的 BGM 相关接口可以实现背景混音功能。背景混音是指主播在直播时可以选取一首歌曲伴唱，歌曲会在主播的手机端播放出来，同时也会被混合到音视频流中被观众端听到，所以被称为“混音”。

精简版 Demo 中的混音功能

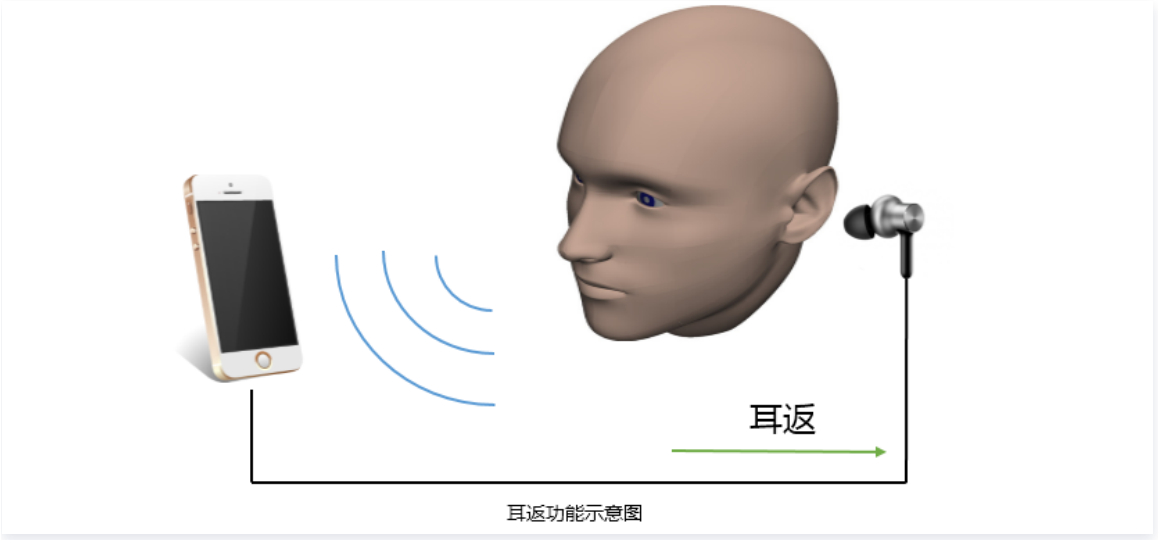
小直播开源 App 中的混音功能

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。
setBgmPitch	调整背景音乐的音调高低。

15. 耳返和混响

调用 TXLivePushConfig 中的 `enableAudioPreview` 选项可以开启耳返功能，“耳返”指的是当主播带上耳机来唱歌时，耳机中要能实时反馈主播的声音。
调用 TXLivePush 中的 `setReverbType` 接口可以设置混响效果，例如 KTV、会堂、磁性、金属等，这些效果也会作用到观众端。
调用 TXLivePush 中的 `setVoiceChangerType` 接口可以设置变调效果，例如“萝莉音”，“大叔音”等，用来增加直播和观众互动的趣味性，这些效果也会作用到观众端。



耳返功能示意图

16. 设置 Logo 水印

设置 TXLivePushConfig 中的 `watermark` 可以让 SDK 在推出的视频流中增加一个水印，水印位置是由 `watermarkNormalization` 选项决定。

- SDK 所要求的水印图片格式为 png 而不是 jpg，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（将 jpg 图片在 Windows 下修改后缀名是不起作用的）。
- `watermarkNormalization` 设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：（0.1，0.1，0.1，0.0），那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

```
//设置视频水印
_config.watermark = [UIImage imageNamed:@"watermark.png"];
_config.watermarkNormalization = CGRectMake(0.1f, 0.1f, 0.1f, 0.0f);
```

17. 本地录制

调用 TXLivePush 中的 `startRecord` 接口可以启动本地录制，录制格式为 MP4，通过参数 `videoPath` 可以指定 MP4文件的存放路径。
调用 TXLivePush 中的 `stopRecord` 接口可以结束录制。如果您已经通过 `recordDelegate` 接口注册了监听器给 TXLivePusher，那么一旦录制结束，录制出来的文件会通过 `TXLiveRecordListener` （详见 TXLiveRecordTypeDef.h 头文件）回调通知出来。

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

注意：

- 只有启动推流后才能开始录制，非推流状态下启动录制无效。
- 出于安装包体积的考虑，仅专业版和企业版两个版本的 LiteAVSDK 支持该功能，直播精简版仅定义了接口但并未实现。
- 企业版已不对外提供，美颜相关功能可参见 腾讯特效 SDK（美颜 SDK）。
- 录制过程中请勿动态切换分辨率和软/硬剪辑，会有很大概率导致生成的视频异常。
- 使用 TXLivePusher 录制视频会一定程度地降低推流性能，云直播服务也提供了云端录制功能，具体使用方法请参考 直播录制 。

18. 主播端弱网提醒

手机连接 Wi-Fi 网络不一定就非常好，如果 Wi-Fi 信号差或者出口带宽很有限，可能网速不如4G，如果主播在推流时遇到网络很差的情况，需要有一个友好的提示，提示主播应当切换网络。

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。此时就可以像上图一样在 UI 上弹出一个“弱网提示”。

```
- (void)onPushEvent:(int)evtID withParam:(NSDictionary *)param {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (evtID == PUSH_ERR_NET_DISCONNECT || evtID == PUSH_ERR_INVALID_ADDRESS) {
            //...
        } else if (evtID == PUSH_WARNING_NET_BUSY) {
            [_notification displayNotificationWithMessage:
                @"您当前的网络环境不佳，请尽快更换网络保证正常直播" forDuration:5];
        }
        //...
    });
```

```
}
```

19. 发送 SEI 消息

调用 TXLivePush 中的 `sendMessageEx` 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

1. 答题直播：推流端将**题目**下发到观众端，可以做到“音-画-题”高度协调同步。
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。

由于自定义消息是直接被塞入视频数据中的，所以不能太大（几个字节比较合适），一般常用于塞入自定义的时间戳等信息。

```
NSString* msg = @"test";
[_pusher sendMessageEx:[msg dataUsingEncoding:NSUTF8StringEncoding]];
```

常规开源播放器或者网页播放器是不能解析 SEI 消息的，必须使用 LiteAVSDK 中自带的 TXLivePlayer 才能解析这些消息：

1. 设置 TXLivePlayConfig 中的 `enableMessage` 选项为 YES。
2. 当 TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 TXLivePlayListener 中的 onPlayEvent(PLAY_EVT_GET_MESSAGE) 通知给您。

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通知和错误通知，详细的事件表和错误码表请参见 错误码表 。需要注意的是：**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有（例如，收到1003就意味着摄像头的画面开始渲染）：

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（部分 Android 手机在此过程需要耗时1s - 2s）。

3. 错误通知

SDK 发现部分严重问题，推流无法继续（例如，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，且经三次重连无效，更多重试请自行重启推流。

4. 警告事件

SDK 发现部分警告问题，但 WARNING 级别的事件都会触发一些尝试性的保护逻辑或者恢复逻辑，而且有很大概率能够恢复。

- **WARNING_NET_BUSY**
  主播网络差，如果您需要 UI 提示，这个 WARNING 相对比较有用。
- **WARNING_SERVER_DISCONNECT**
  推流请求被后台拒绝，一般是由于推流地址里的 txSecret 计算错误，或者是推流地址被其他人占用（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）。

Android

最近更新时间：2024-11-28 17:05:53

功能概述

摄像头推流，是指采集手机摄像头的画面以及麦克风的声音，进行编码之后再推送到直播云平台上。腾讯云 LiteAVSDK 通过 TXLivePusher 接口提供摄像头推流能力，如下是 LiteAVSDK 的简单版 Demo 中演示摄像头推流的相关操作界面：

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` 为 false。但当您要推流的地址为腾讯云地址时，请务必在推流前将其设置为 YES，否则 SDK 针对腾讯云的协议优化将不能发挥作用。
- **真机调试**
  由于 SDK 大量使用 Android 系统的音视频接口，这些接口在仿真模拟器下往往不能工作，推荐您尽量使用真机调试。

示例代码

所属平台	GitHub 地址	关键类
iOS	Github	CameraPushViewController.m
Android	Github	CameraPushMainActivity.java

功能对接

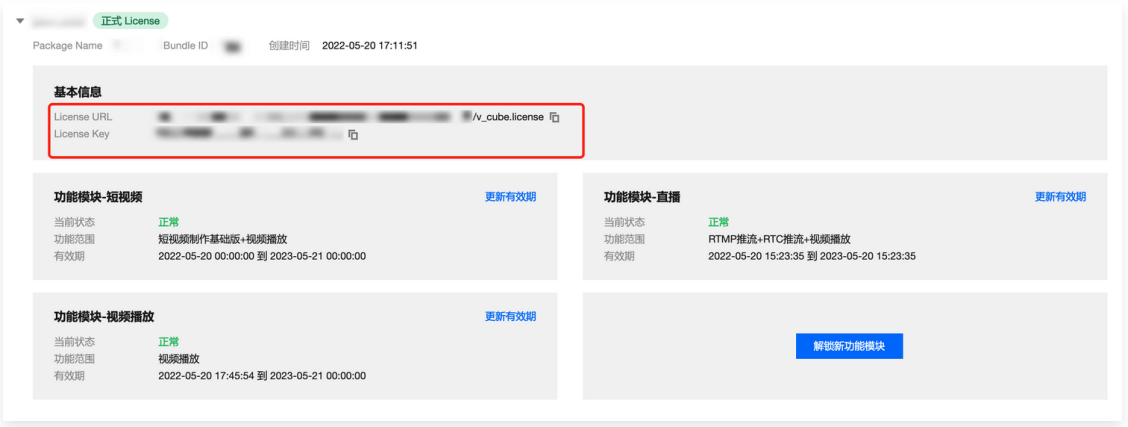
1. 下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

2. 给 SDK 配置 License 授权

1. 获取 License 授权：

- 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。



- 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。

2. 在您的 App 调用企业版 SDK 相关功能之前（建议在 Application类中）进行如下设置：

```
public class MApplication extends Application {

 @Override
 public void onCreate() {
     super.onCreate();
     String licenceURL = ""; // 获取到的 licence url
     String licenceKey = ""; // 获取到的 licence key
     TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
     TXLiveBase.setListener(new TXLiveBaseListener() {
         @Override
         public void onLicenceLoaded(int result, String reason) {
             Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
         }
     });
 }
}
```

> **注意**
> - 企业版已不对外提供，美颜相关功能可参见 腾讯特效 SDK（美颜 SDK）。
> - License 中配置的 packageName 必须和应用本身一致，否则会推流失败。

3. 初始化 TXLivePusher 组件

首先创建一个 `TXLivePushConfig` 对象。该对象可以指定一些高级配置参数，但一般情况下我们不建议您操作该对象，因为我们已经在其内部配置好了所有需要校调的参数。之后再创建一个 `TXLivePusher` 对象，该对象负责完成推流的主要工作。

```
TXLivePushConfig mLivePushConfig  = new TXLivePushConfig();
TXLivePusher mLivePusher = new TXLivePusher(this);

// 一般情况下不需要修改 config 的默认配置
mLivePusher.setConfig(mLivePushConfig);
```

4. 开启摄像头预览

欲展示摄像头的预览画面，您需要先给 SDK 提供一个用于显示视频画面的 `TXCloudVideoView` 对象，由于 `TXCloudVideoView` 是继承自 Android 中的 `FrameLayout`，所以您可以直接在 xml 文件中添加一个视频渲染控件：

```
<com.tencent.rtmp.ui.TXCloudVideoView
        android:id="@+id/pusher_tx_cloud_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

之后，就可以通过调用 TXLivePusher 中的 `startCameraPreview` 接口开启当前手机摄像头的预览画面。

```
//启动本地摄像头预览
TXCloudVideoView mPusherView = (TXCloudVideoView) findViewById(R.id.pusher_tx_cloud_view);
mLivePusher.startCameraPreview(mPusherView);
```

5. 启动和结束推流

如果已经启动了摄像头预览，就可以调用 TXLivePusher 中的 `startPusher` 接口开始推流。

```
String rtmpURL = "rtmp://test.com/live/xxxxxx"; //此处填写您的 rtmp 推流地址
int ret = mLivePusher.startPusher(rtmpURL.trim());
if (ret == -5) {
    Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

推流结束后，可以调用 TXLivePusher 中的 `stopPusher` 接口结束推流。请注意，如果已经启动了摄像头预览，请在结束推流时将其关闭，否则会导致 SDK 的表现异常。

```
mLivePusher.stopPusher();
mLivePusher.stopCameraPreview(true); //如果已经启动了摄像头预览，请在结束推流时将其关闭。
```

- **获取可用的推流 URL**
  开通直播服务后，可以使用**直播控制台 > 直播工具箱 > 地址生成器** 生成推流地址，详细信息请参见 推拉流 URL 。

- **返回 -5 的原因**如果 `startPusher` 接口返回 -5，则代表您的 License 校验失败了，请检查第2步“给 SDK 配置 License 授权”中的工作是否有问题。

6. 纯音频推流

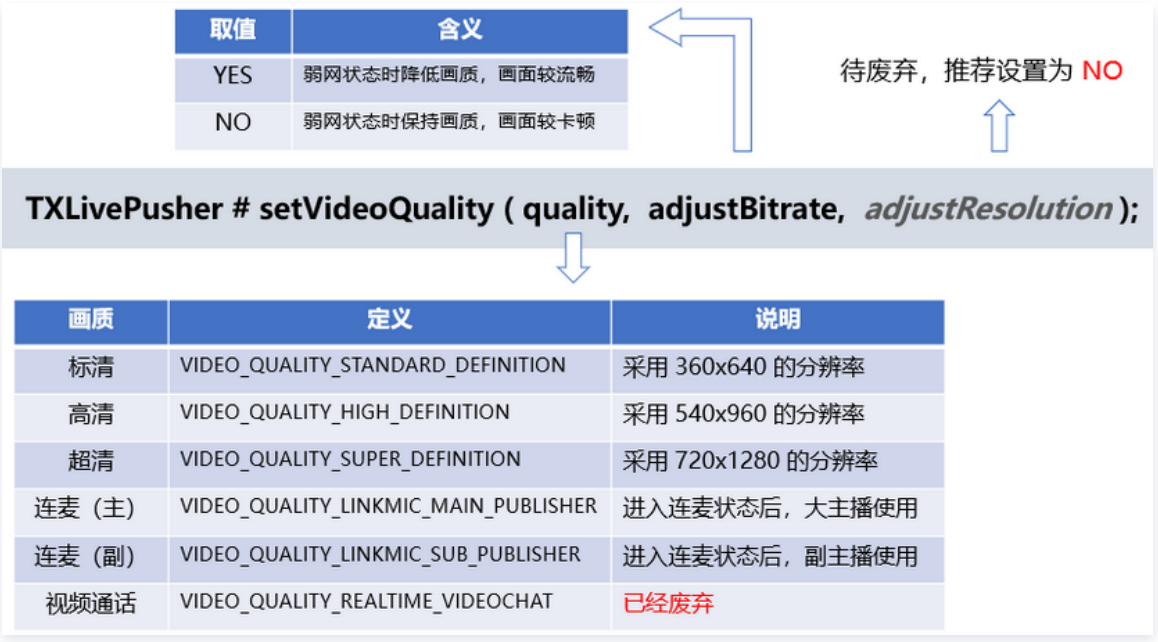
如果您的直播场景是纯音频直播，不需要视频画面，那么您可以不执行 第 4 步 中的操作，取而代之的是开启 TXLivePushConfig 中的 `enablePureAudioPush` 配置。

```
TXLivePushConfig mLivePushConfig  = new TXLivePushConfig();
TXLivePusher mLivePusher = new TXLivePusher(this);

//开启纯音频推流，只有在调用 startPusher 前设置才会生效。
mLivePushConfig.enablePureAudioPush(true);
mLivePusher.setConfig(mLivePushConfig);
String rtmpURL = "rtmp://test.com/live/xxxxxx";
mLivePusher.startPusher(rtmpURL.trim());
```

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

7. 设定画面清晰度

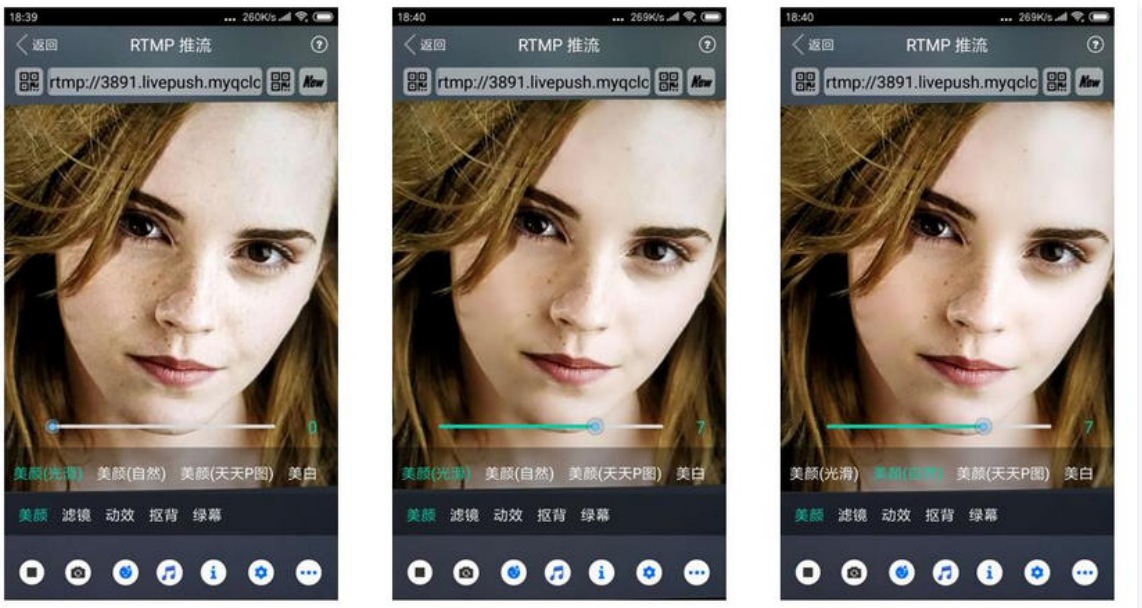
调用 TXLivePusher 中的 `setVideoQuality` 接口，可以设定观众端的画面清晰度。之所以说是观众端的画面清晰度，是因为主播看到的视频画面是未经编码压缩过的高清原画，不受设置的影响。而 `setVideoQuality` 设定的视频编码器的编码质量，观众端可以感受到画质的差异。详情请参见 设定画面质量。



8. 美颜美白和红润特效

调用 TXLivePush 中的 `setBeautyFilter` 接口可以设置美颜效果，SDK 中提供了三种磨皮算法（style），定义见 `TXLiveConstants.java` 文件：

美颜风格	效果说明
TXLiveConstants.BEAUTY_STYLE_SMOOTH	光滑风格，算法更加注重皮肤的光滑程度，适合秀场直播类场景下使用。
TXLiveConstants.BEAUTY_STYLE_NATURE	自然风格，算法更加注重保留皮肤细节，适合对真实性要求更高的主播。
TXLiveConstants.BEAUTY_STYLE_HAZY	朦胧风格，算法会更加侧重画面去噪，使整体画面风格偏柔和。

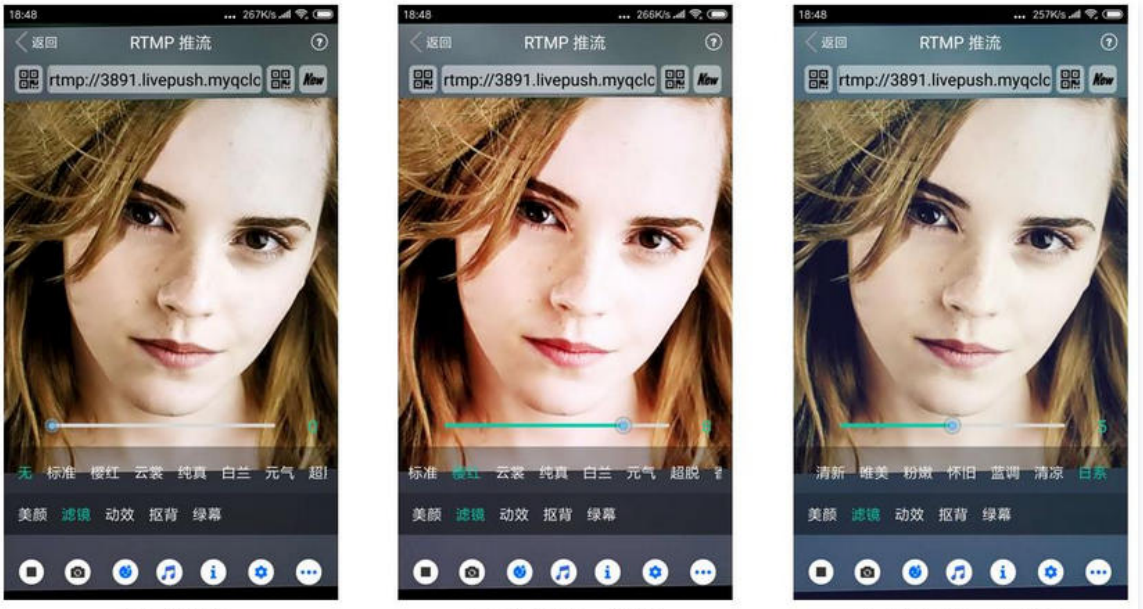


关闭美颜　美颜算法：光滑　美颜算法：自然

```
//style                 美颜算法：  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);
```

9. 色彩滤镜效果

调用 TXLivePusher 中的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
调用 TXLivePush 中的 `setSpecialRatio` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。
从手机 QQ 和 Now 直播的经验来看，单纯通过 `setBeautyStyle` 调整磨皮效果是不够的，只有将美颜效果和 `setFilter` 配合使用才能达到更加多变的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜，并将其默认打包在了 Demo 中供您使用。



关闭滤镜　开启滤镜：樱红　开启滤镜：日系

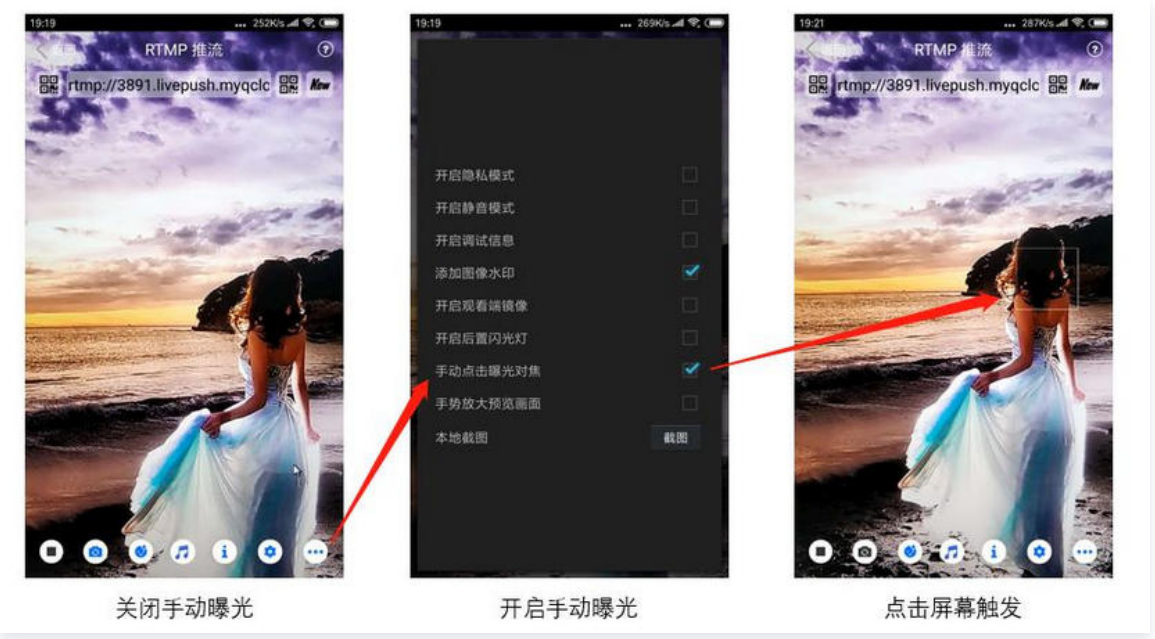
```
//选择期望的色彩滤镜文件
Bitmap filterBmp = decodeResource(getResources(), R.drawable.filter_biaozhun);
mLivePusher.setFilter(filterBmp);
mLivePusher.setSpecialRatio(0.5f);
```

10. 控制摄像头

TXLivePusher 提供了一组 API 用户控制摄像头的行为：

API 函数	功能说明	备注说明
switchCamera	切换前后摄像头	-
turnOnFlashLight	打开或关闭闪光灯	仅在当前是后置摄像头时有效。
setZoom	调整摄像头的焦距	可以通过 TXLivePusher 的 `getMaxZoom()` 函数获取最大焦距， `setZoom` 的取值范围即为 1 - 最大焦距。
setExposureCompensation	设置曝光比例，取值范围从-1到1	负数表示调低曝光，-1是最小值，对应 `getMinExposureCompensation` 。正数表示调高曝光，1是最大值，对 `getMaxExposureCompensation` 。0表示不调整曝光，默认值为0。

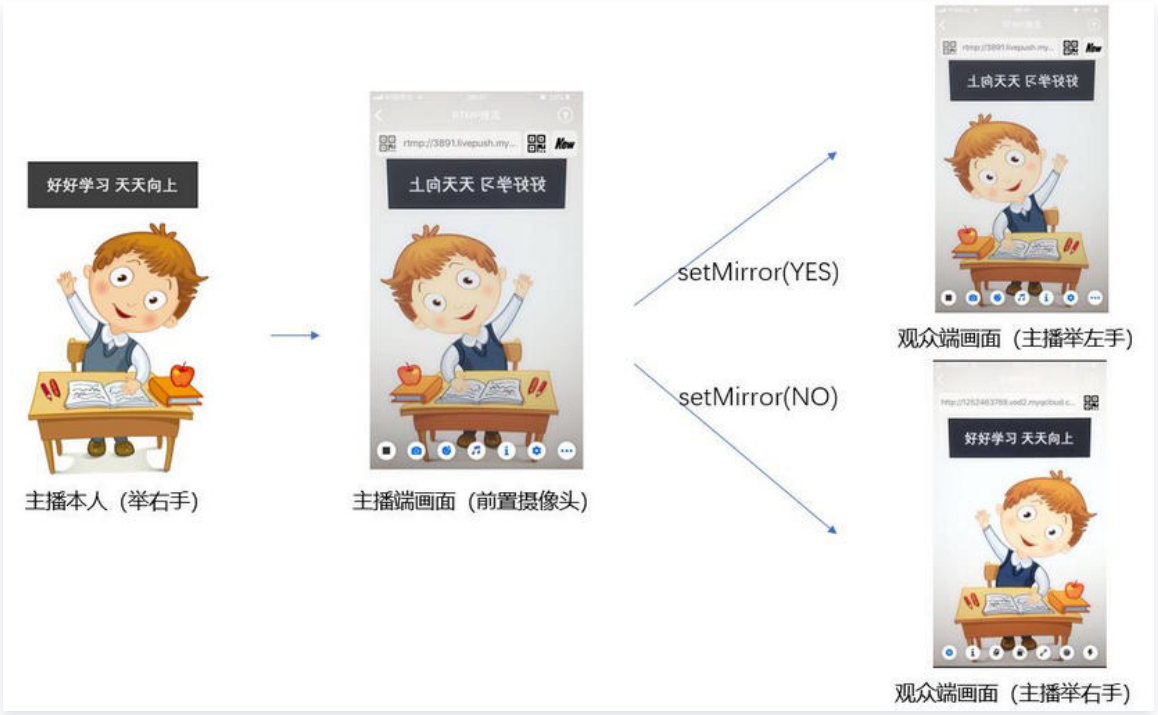
除了 TXLivePusher，TXLivePushConfig 中也提供了一个叫做 `setTouchFocus` 的设置项，用于控制是手动对焦还是自动对焦。




```
TXLivePushConfig mLivePushConfig  = new TXLivePushConfig();
TXLivePusher mLivePusher = new TXLivePusher(this);

//开启手动曝光（需要 API 14 以上的 Android 系统才能支持）
mLivePushConfig.setTouchFocus(true);
mLivePusher.setConfig(mLivePushConfig);
```

11. 观众端的镜像效果

调用 TXLivePusher 中的 `setMirror` 接口可以设置观众端的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，自己看到的画面会被 SDK 默认反转，这时主播的体验跟自己照镜子时的体验是保持一致的。 `setMirror` 所影响的是观众端观看到的画面情况，如



下图所示：

12. 横屏推流

大多数情况下，主播习惯以“竖屏持握”手机进行直播拍摄，观众端看到的也是竖屏分辨率的画面（例如 540 × 960 这样的分辨率）；有时主播也会“横屏持握”手机，这时观众端期望能看到是横屏分辨率的画面（例如 960 × 540 这样的分辨率），如下图所示：

TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，需要：

1. 设置 TXLivePushConfig 中的 `setHomeOrientation` 改变观众端看到的视频画面的宽高比方向。
2. 调用 TXLivePusher 中的 `setRenderRotation` 接口改变主播端的视频画面的渲染方向。

```
public void onOrientationChange(boolean isPortrait) {
    if (isPortrait) {
        mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_DOWN);
```

```
        mLivePusher.setConfig(mLivePushConfig);
        mLivePusher.setRenderRotation(0);
    } else {
        mLivePushConfig.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
        mLivePusher.setConfig(mLivePushConfig);
        // 因为采集旋转了，为了保证本地渲染是正的，则设置渲染角度为90度。
        mLivePusher.setRenderRotation(90);
    }
}
```

注意

Android 中的 Activity 支持跟随手机的重力感应自动渲染，如果您开启了 Activity 的自动重力感应旋转，请参见 CameraPushMainActivity.java 中的 `setRotationForActivity` 以及相关示例代码。

13. 隐私模式（垫片推流）

有时候主播的一些动作不希望被观众看到，但直播过程中又不能下播，那就可以考虑进入隐私模式。在隐私模式下，SDK 可以暂停采集主播手机的摄像头画面以及麦克风声音，并使用一张默认图片作为替代图像进行推流，也就是所谓的“垫片”。

通过 TXLivePushConfig 中的 `setPauseImg` 接口可以设置垫片用的背景图片、垫片的最大时长以及视频帧率。通过 TXLivePushConfig 中的 `setPauseFlag` 接口可以设置是暂停视频采集、还是暂停声音采集，还是两者都暂停。

```
TXLivePushConfig mLivePushConfig  = new TXLivePushConfig();
TXLivePusher mLivePusher = new TXLivePusher(this);
```

```
// bitmap: 用于指定垫片图片，最大尺寸不能超过 1920*1920
// time：垫片最长持续时间，单位是秒，300即代表最长持续300秒
// fps：垫片帧率，最小值为 5fps，最大值为 20fps。
Bitmap bitmap = decodeResource(getResources(), R.drawable.pause_publish);
mLivePushConfig.setPauseImg(bitmap);
mLivePushConfig.setPauseImg(300, 5);
//表示仅暂停视频采集，不暂停音频采集
//mLivePushConfig.setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO);
//表示同时暂停视频和音频采集
mLivePushConfig.setPauseFlag(PAUSE_FLAG_PAUSE_VIDEO|PAUSE_FLAG_PAUSE_AUDIO);
mLivePusher.setConfig(mLivePushConfig);
```

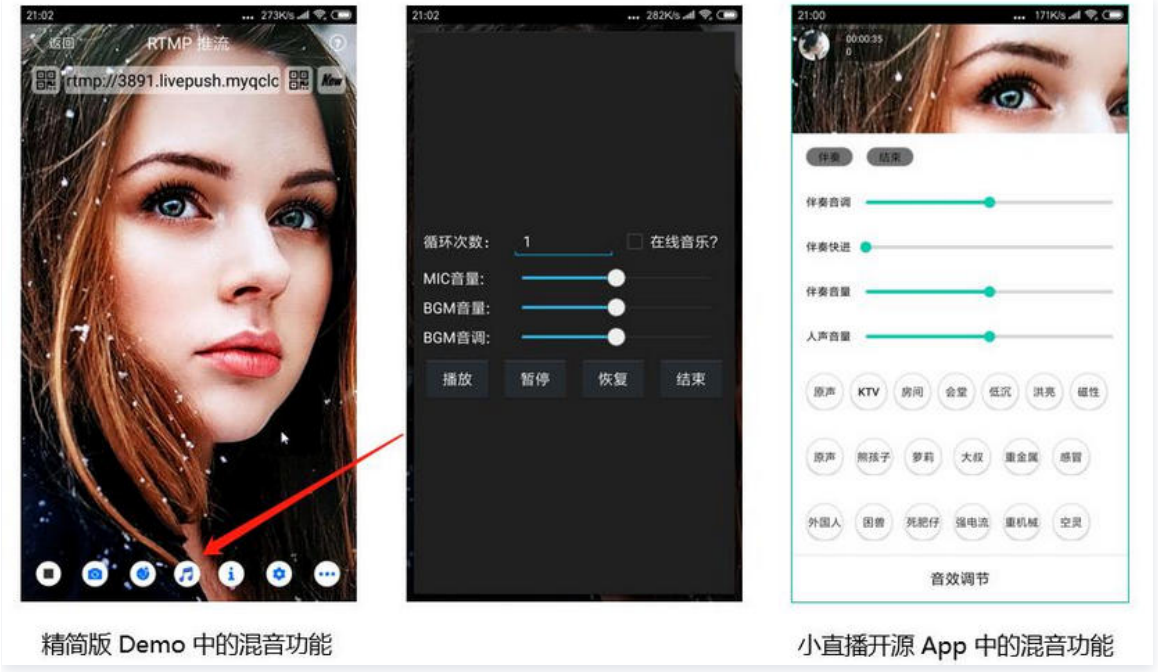
设置完成之后，就可以调用 TXLivePusher 中的 `pausePusher` 进入隐私模式，也可以调用 `resumePusher` 退出隐私模式，但请注意保持正确的调用顺序：startPush=> (pausePush => resumePush) => stopPush，错误的调用顺序会导致 SDK 表现异常，因此使用成员变量对执行顺序进行保护是很有必要的。

```
// 进入隐私模式
mLivePusher.pausePusher();
...
// 退出隐私模式
mLivePusher.resumePusher();
```

注意

- SDK 支持在 App 切后台之后继续采集画面和声音，您只需不调用 `pausePusher()` 就可以达到这个效果，但这不利于保护主播的隐私。
- 隐私模式可以通过在 App 界面上加一个切换按钮来让主播自主进入，也可以编写代码，让 App 在切后台时自动进入，目前 Demo 就采用了 App 切后台时自动进入隐私模式的交互方案，注释掉源码中对 `pausePusher()` 和 `resumePusher()` 的调用就可以关闭这个特性。

14. 背景混音和混响



精简版 Demo 中的混音功能　　小直播开源 App 中的混音功能

调用 TXLivePush 中的 BGM 相关接口可以实现背景混音功能。背景混音是指主播在直播时可以选取一首歌曲伴唱，歌曲会在主播的手机端播放出来，同时也会被混合到音视频流中被观众端听到，所以被称为“混音”。

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。
setMicVolumeOnMixing	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。
setBgmPitch	调整背景音乐的音调高低。

调用 TXLivePush 中的 `setReverbType` 接口可以设置混响效果，例如 KTV、会堂、磁性、金属等，这些效果也会作用到观众端。调用 TXLivePush 中的 `setVoiceChangerType` 接口可以设置变调效果，例如“萝莉音”，“大叔音”等，用来增加直播和观众互动的趣味性，这些效果也会作用到观众端。

15. 设置 Logo 水印

通过 TXLivePushConfig 中的 `setWatermark` 接口可以让 SDK 在推出的视频流中增加一个水印，水印的位置由该接口函数的后三个参数决定。

- SDK 所要求的水印图片格式为 png 而不是 jpg，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（将 jpg 图片在 Windows 下修改后缀名是不起作用的）。
- `setWatermark` 中后三个参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 x 960，后三个参数 x、y 和 width 被分别设置为：（0.1，0.1，0.1），那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

```
//设置视频水印
TXLivePushConfig mLivePushConfig  = new TXLivePushConfig();
//四个参数依次是水印图片的 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]
Bitmap waterBmp = decodeResource(getResources(), R.drawable.filter_water);
mLivePushConfig.setWatermark(waterBmp, 0.1f, 0.1f, 0.1f);
mLivePusher.setConfig(mLivePushConfig);
```

16. 本地录制

调用 TXLivePusher 中的 `startRecord` 接口可以启动本地录制，录制格式为 MP4，通过参数 `videoPath` 可以指定 MP4 文件的存放路径。调用 TXLivePusher 中的 `stopRecord` 接口可以结束录制。如果您已经通过 `setVideoRecordListener` 接口注册监听器给 TXLivePusher，那么一旦录制结束，录制出来的文件会通过 `TXRecordCommon.ITXVideoRecordListener` 回调通知出来。

```
// 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);
}
```

> ⚠ 注意
>
> 1. 只有启动推流后才能开始录制，非推流状态下启动录制无效。
> 2. 出于安装包体积的考虑，仅专业版和企业版两个版本的 LiteAVSDK 支持该功能，直播精简版仅定义了接口但并未实现。
> 3. 录制过程中请勿动态切换分辨率和软/硬剪辑，会有很大概率导致生成的视频异常。
> 4. 使用 TXLivePusher 录制视频会一定程度地降低推流性能，云直播服务也提供了云端录制功能，具体使用方法请参考 直播录制 。

17. 主播端弱网提醒

手机连接 Wi-Fi 网络不一定就非常好，如果 Wi-Fi 信号差或者出口带宽很有限，可能网速不如4G，如果主播在推流时遇到网络很差的情况，需要有一个友好的提示，提示主播应当切换网络。

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。此时就可以像上图一样在 UI 上弹出一个“弱网提示”。

```
@Override
    public void onPushEvent(int event, Bundle param) {
        if (event == TXLiveConstants.PUSH_ERR_NET_DISCONNECT
                || event == TXLiveConstants.PUSH_ERR_INVALID_ADDRESS
                || event == TXLiveConstants.PUSH_ERR_OPEN_CAMERA_FAIL
                || event == TXLiveConstants.PUSH_ERR_OPEN_MIC_FAIL) {
            // 遇到以上错误，则停止推流
            //...
        } else if (event == TXLiveConstants.PUSH_WARNING_NET_BUSY) {
            //您当前的网络环境不佳，请尽快更换网络保证正常直播
            showNetBusyTips();
        }
    }
```

18. 发送 SEI 消息

调用 TXLivePush 中的 `sendMessageEx` 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

1. 答题直播：推流端将**题目**下发到观众端，可以做到“音-画-题”高度协调同步。
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。

由于自定义消息是直接被塞入视频数据中的，所以不能太大（几个字节比较合适），一般常用于塞入自定义的时间戳等信息。

```
//Android 示例代码
String msg = "test";
mTXLivePusher.sendMessageEx(msg.getBytes("UTF-8"));
```

常规开源播放器或者网页播放器是不能解析 SEI 消息的，必须使用 LiteAVSDK 中自带的 TXLivePlayer 才能解析这些消息：

1. 设置 TXLivePlayConfig 中的 `enableMessage` 选项为 YES。
2. 当 TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 TXLivePlayListener 中的 `onPlayEvent(PLAY_EVT_GET_MESSAGE)` 通知给您。

事件处理

1. 事件监听

SDK 通过 ITXLivePushListener 代理来监听推流相关的事件通知和错误通知，详细的事件表和错误码表请参见 错误码表 ，也可以查阅 `TXLiveConstants.java` 代码文件。需要注意的是：**ITXLivePushListener 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有（例如，收到1003就意味着摄像头的画面开始渲染）：

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（部分 Android 手机这个过程需要1秒 - 2秒）。

3. 错误通知

SDK 发现部分严重问题，推流无法继续（例如，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，且经三次重连无效，更多重试请自行重启推流。

4. 警告事件

SDK 发现了一些问题，但这并不意味着推流无法继续，SDK 会在警告事件发生后，尽可能地启动一些重试性的保护逻辑或者恢复措施，例如：

- **WARNING_NET_BUSY** 主播网络差，如果您需要 UI 提示，这个 WARNING 相对比较有用。
- **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）。

录屏推流
iOS

最近更新时间：2024-11-28 17:05:53

概述

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

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

功能体验

体验 iOS 录屏可以单击安装 视频云工具包 或通过扫码进行安装。

> **注意：**
> 录屏推流功能仅11.0以上系统可体验。

使用步骤：

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

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

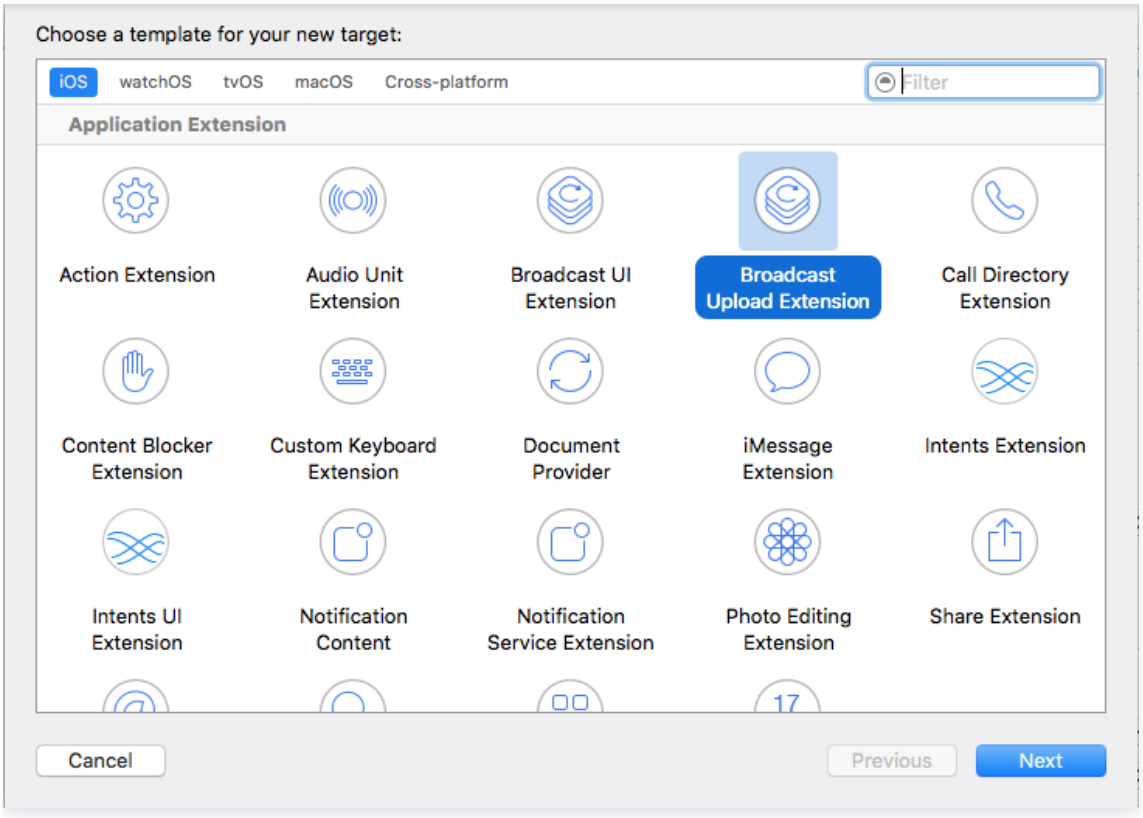
开发环境准备

Xcode 准备

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

创建直播扩展

在现有工程选择 **New > Target…**，选择 **Broadcast Upload Extension**，如图所示。



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

导入 LiteAV SDK

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

对接流程

步骤 1：创建推流对象

在 SampleHandler.m 中添加下面代码

```
#import "SampleHandler.h"
#import "TXRTMPSDK/TXLiveSDKTypeDef.h"
#import "TXRTMPSDK/TXLivePush.h"
#import "TXRTMPSDK/TXLiveBase.h"

static TXLivePush *s_txLivePublisher;
static NSString *s_rtmpUrl;
```

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

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

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

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

- s_txLivePublisher 是我们用于推流的对象，因为系统录屏回调的 sampleHandler 实例有可能不止一个，因此对变量采用静态声明，确保录屏推流过程中使用的是同一个推流器。
- s_txLivePublisher 的 config 默认的配置为摄像头推流配置，因此需要额外配置为自定义采集视频和音频模式，视频开启 autoSampleBufferSize，SDK 会自动根据输入的分辨率设置编码器，您不需要关心推流的分辨率；如果您关闭此选项，那么代表您需要自定义分辨率。
- 因为系统录制对不同机型屏幕所得到的分辨率不一致，所以录屏推流不建议您开启 autoSampleBufferSize，使用自定义分辨率设置。
- 实例化 s_txLivePublisher 的最佳位置是在 `-[SampleHandler broadcastStartedWithSetupInfo:]` 方法中，直播扩展启动后会回调这个函数，就可以进行推流器初始化开始推流。但在 ReplayKit2 的屏幕录制扩展启动时，回调给 s_txLivePublisher 的 setupInfo 为 nil，无法获取启动推流所需要的推流地址等信息，因此通常回调此函数时发通知给主 App，在主 App 中设置好推流地址，横竖屏清晰度等信息后再传递给扩展并通知扩展启动推流。
- 扩展与主 App 间的通信请参见后面所附的 扩展与宿主 App 之间的通信与数据传递方式 。

步骤 2：横屏推流与自定义分辨率

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

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

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

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

    }
    else {  //高清
        config.sampleBufferSize = CGSizeMake(544, (uint)(540 * screenSize.height /
screenSize.width));
        config.videoBitrateMin = 1000;
        config.videoBitratePIN = 1400;
        config.videoBitrateMax = 1800;
        config.videoFPS = 24;
    }

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

注意：

- 一般手机上为9：16，而在 iPhoneX 上画面比例为1125：2436，因此此处使用屏幕比例进行计算分辨率。
- 在 ReplayKit2 上采集的都是竖屏的分辨率，如果需要推送横屏分辨率，除了设置横屏分辨率外还需同时指定 homeOrientation 为横屏推流，否则会引起画面变形。

步骤 3：发送视频

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

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:
(RPSampleBufferType)sampleBufferType {
```

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

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

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

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

步骤 4：发送音频

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

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

SDK 支持同时发送两路数据，内部会对两路数据进行混音处理。

步骤 5：暂停与恢复

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

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

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

步骤 6：SDK 事件处理

事件监听

SDK 事件监听需要设置 `TXLivePush` 的 delegate 属性，该 delegate 遵循 `TXLivePushListener` 协议。底层的事件会通过 `-(void) onPushEvent:(int)EvtID withParam:(NSDictionary*)param` 接口回调过来。录屏推流过程中，一般会收到以下事件：

常规事件

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

可在 PUSH_EVT_PUSH_BEGIN 事件时通知用户推流成功。

错误事件

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

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

警告事件

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

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

> **说明：**

> 全部事件定义请参阅头文件 “TXLiveSDKEventDef.h”。

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

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

步骤 7：结束推流

结束推流 ReplayKit 会调用 `-[SampleHandler broadcastFinished]`，示例代码：

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

结束推流后，直播扩展进程可能会被系统回收，所以需要在此处做好清理工作。

附: 扩展与宿主 App 之间的通信与数据传递方式参考

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

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

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

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

    UNMutableNotificationContent* content = [[UNMutableNotificationContent alloc] init];
    content.title = [NSString localizedUserNotificationStringForKey:title arguments:nil];
    content.body = [NSString localizedUserNotificationStringForKey:msg  arguments:nil];
```

```
    content.sound = [UNNotificationSound defaultSound];
    content.userInfo = userInfo;

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

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

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

    }];
}
```

通过此通知可以提示用户回到主 App 设置推流信息、启动推流等。

2. **进程间的通知 CFNotificationCenter**

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

```
CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter(),kDarvinNotifi
cationNamePushStart,NULL,nil,YES);
```

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

```
    CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
                                    (__bridge const void *)(self),
                                    onDarwinReplayKit2PushStart,
                                    kDarvinNotificationNamePushStart,
                                    NULL,
                                    CFNotificationSuspensionBehaviorDeliverImmediately);

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


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

```
}

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

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

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

常见问题

ReplayKit2 屏幕录制是 iOS 11 新推出功能，官方文档较少，且存在着一些问题，这些问题在每个版本的系统都在不断修复完善中。以下是一些使用中的常见现象或问题：

1. **系统有声音在播放但观众端无法听到声音?**
   系统在做屏幕音频采集时，在从 home 界面切到有声音播放的 App 时才会采集声音，从有声音播放的 App 切换到无声音播放的 App 时，即使原 App 还在播放声音系统也不会进行音频采集，此时需要从 home 界面重新进入到有声音播放的 App 时系统才会重新采集。
2. **收到推送信息观众端有时听不到声音?**
   这个是 ReplayKit2 在早期系统中存在的问题，收到推送消息后会停止屏幕录制的声音采集或采集到的是静音数据，需要重新从 home 界面切回到有时间的 App 才能恢复音频采集。在11.3之后的版本系统修复了这个问题。
3. **打开麦克风录制时系统播放声音会变小?**
   这个是属于系统机制：打开麦克风采集时系统音频处于录制模式，会自动将其它的 App 播放的声音变为听筒模式，中途关闭麦克风采集也不会恢复，只有关闭或重新启动无麦克风录制时才会恢复为扬声器的播放。这个机制不影响 App 那路声音的录制，即观众端声音听到的声音大小不受影响。
4. **屏幕录制何时自动会停止?**
   系统在锁屏或有电话打入时，会自动停止屏幕录制，此时 SampleHandler 里的 broadcastFinished 函数会被调用，可在此函数发通知提示用户。
5. **采集推流过程中有时屏幕录制会自动停止问题?**
   通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2 的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此 ReplayKit2 上建议推流分辨率不高于720P。另外不建议使用 autoSampleBufferSize 时做横竖屏切换，因为 Plus 的手机的分辨率可达1080 * 1920，容易触发系统内存限制而被强制停止。
6. **iPhoneX 手机的兼容性与画面变形问题?**
   iPhoneX 由于拥有刘海设计，导致其屏幕采集的画面分辨率并非标准的9:16。若将推流输出分辨率设定为9:16比例，例如高清分辨率960*540，由于源分辨率与此不符，推出的画面可能会略显变形。建议根据屏幕实际分辨率比例来设置推流分辨率。在拉流端，使用 AspectFit 显示模式时，iPhoneX 屏幕采集的推流出现黑边是正常现象；而使用 AspectFill 模式时，画面可能会显示不完整。

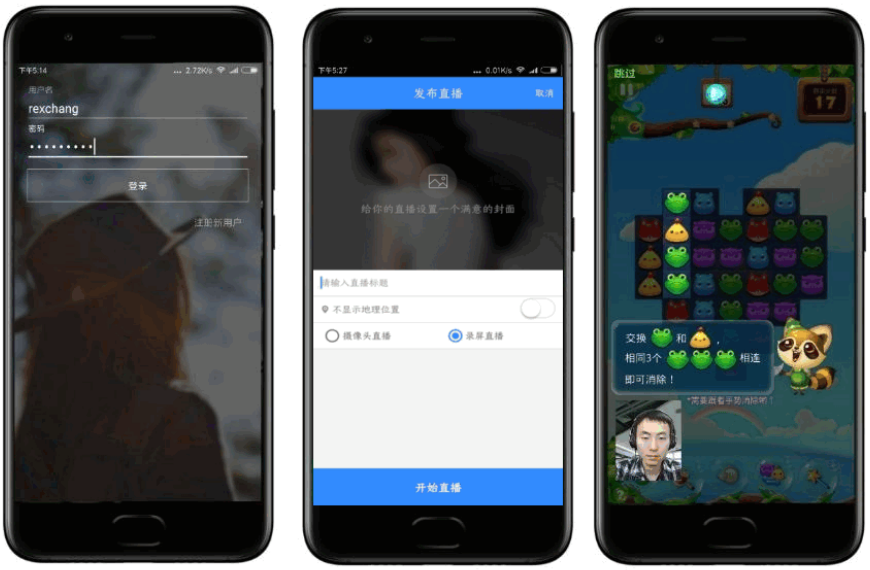
Android

最近更新时间：2023-07-12 15:40:22

功能介绍

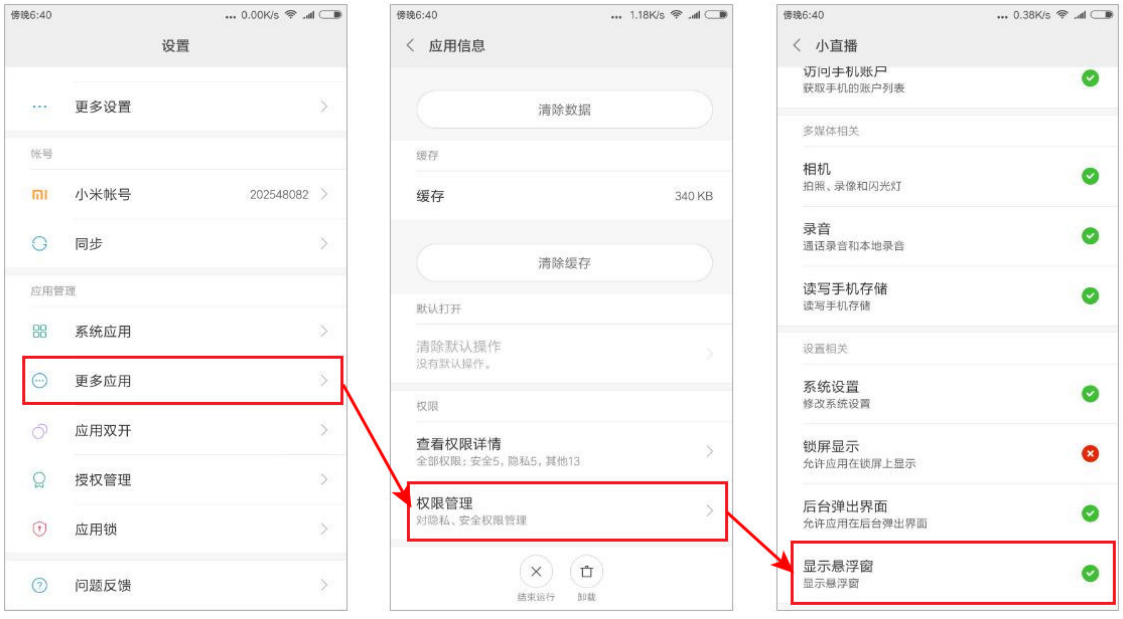
手机录屏直播，即可以直接把主播的手机画面作为直播源，同时可以叠加摄像头预览，应用于游戏直播、移动端 App 演示等需要手机屏幕画面的场景。

扫码安装小直播



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

步骤 1：添加 Activity

在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。

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

步骤 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，例如推流用多大的分辨率、每秒钟要多少帧画面（FPS）以及多少秒一个I帧（ Gop）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);
```

步骤 3：启动推流

经过 步骤 1 和 步骤 2 的准备之后，用下面这段代码就可以启动推流了：

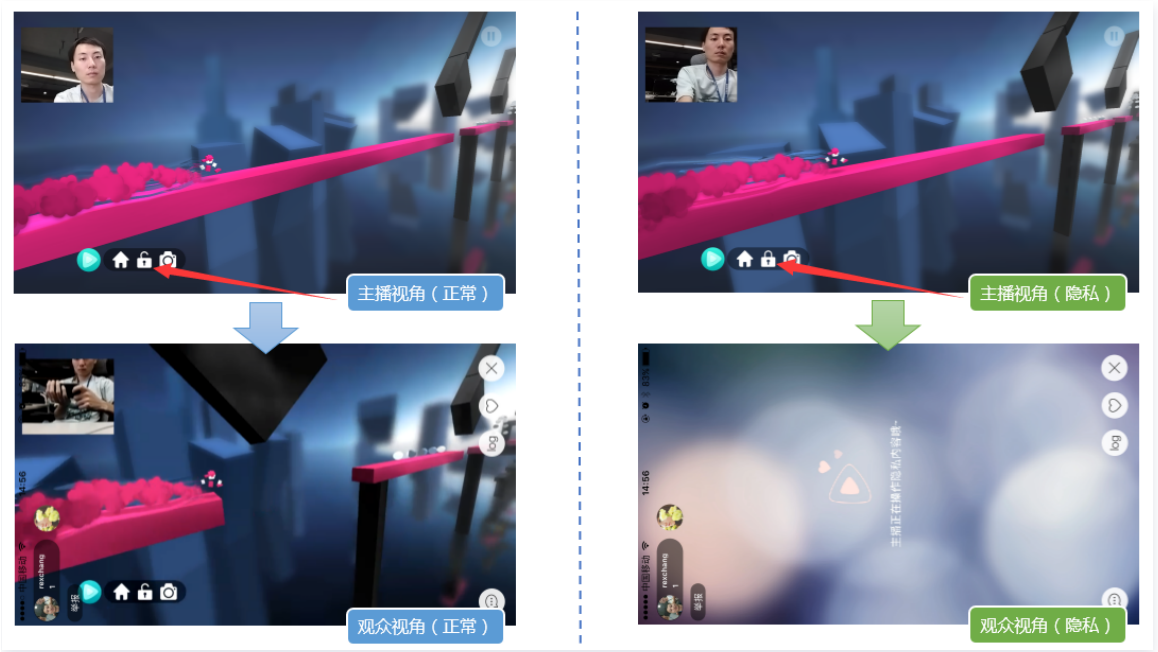
```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
mLivePusher.startPusher(rtmpUrl);
mLivePusher.startScreenCapture();
```

- **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，出于安全考虑，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”。

步骤 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（例如输入游戏的账号密码等等），那么此时主播可以启动**隐私模式**。在隐私模式下，主播的推流还是持续的，观众也一直能看到画面，只是看到的画面是一张提示“主播正在操作隐私内容哦~”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：

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，例如“主播把画面切走一会儿...”。
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;
  }
```

步骤 5：设置 Logo 水印

据相关政策规定，直播视频必须加上水印。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：

- 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。
- 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。
- 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（建议您不要在 Windows 下将 jpg 格式的图片修改后缀名就直接使用，因为专业的 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);
```

步骤 6：推荐的清晰度

影响画质的主要因素有：**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20FPS – 25FPS 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，例如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。

- 一种极端就是手机屏幕停在一个界面保持不动，例如桌面，这个时候编码器可以用很小的码率输出就能完成任务。
- 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，例如主播在玩《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（捕鱼达人）	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

步骤 7：提醒主播“网络不好”

步骤 10 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，或者建议主播离 Wi-Fi 近一点，或者让他提醒一声：“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。”

步骤 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。

步骤 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，与此同时，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。

步骤 10：事件处理

事件监听

RTMP SDK 通过 ITXLivePushListener 代理来监听推流相关的事件，注意 ITXLivePushListener 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，例如收到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秒 - 2秒）

错误通知

SDK 发现了一些严重问题，推流无法继续了，例如，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，且经三次重连无效，可以放弃，更多重试请自行重启推流

警告事件

SDK 发现了一些问题，但这并不意味着无法解决，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”。

- **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（步骤10）。
- **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。但实际上，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）

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

步骤 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();
}
```

标准直播拉流
iOS

最近更新时间：2024-10-29 11:32:41

基础知识

本文主要介绍视频云 SDK 的直播播放功能。

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”开头，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。

对接攻略

step 1：下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

step 2：给 SDK 配置 License 授权

1. 获取 License 授权：
   - 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。

   - 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。

2. 在您的 App 调用 SDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

    //TXLiveBase 位于 "TXLiveBase.h" 头文件中
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

> ⚠ **注意：**
> License 中配置的 BundleId 必须和应用本身一致，否则会播放失败。

step 3：创建 Player

视频云 SDK 中的 TXLivePlayer 模块负责实现直播播放功能。

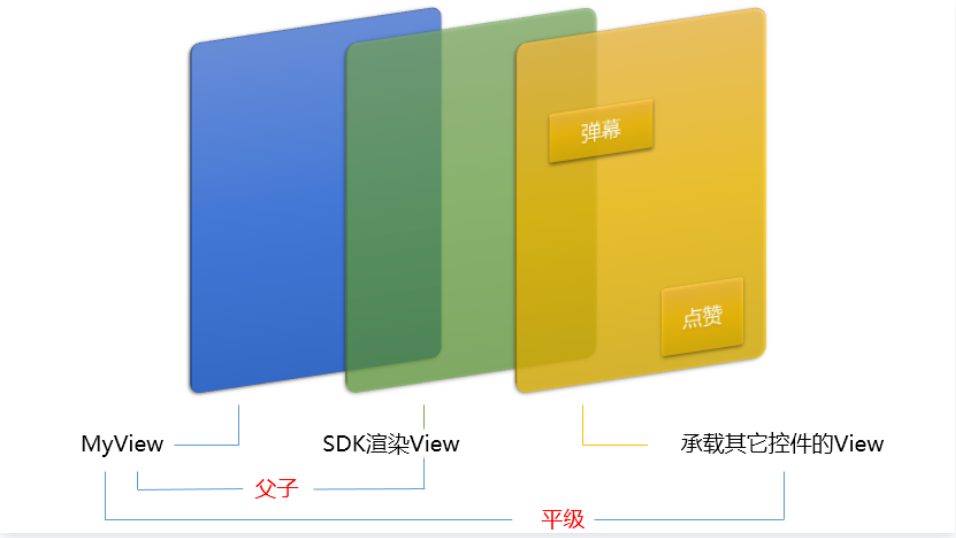
```
TXLivePlayer *_txLivePlayer = [[TXLivePlayer alloc] init];
```

step 4：渲染 View

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

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

内部原理上，播放器并不是直接把画面渲染到您提供的 view（示例代码中的 _myView）上，而是在这个 view 之上创建一个用于 OpenGL 渲染的子视图（subView）。
如果您要调整渲染画面的大小，只需要调整您所常见的 view 的大小和位置即可，SDK 会让视频画面跟着您的 view 的大小和位置进行实时的调整。



如何做动画?

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

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

step 5：启动播放

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

可选值	枚举值	含义
PLAY_TYPE_LIVE_RTMP	0	传入的 URL 为 RTMP 直播地址
PLAY_TYPE_LIVE_FLV	1	传入的 URL 为 FLV 直播地址
PLAY_TYPE_LIVE_RTMP_ACC	5	低延迟链路地址（仅适合于连麦场景）
PLAY_TYPE_LIVE_HLS	7	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（虽然它很适合用于点播），因为延迟太高，在 App 上推荐使用 LIVE_FLV 或者 LIVE_RTMP 播放协议。

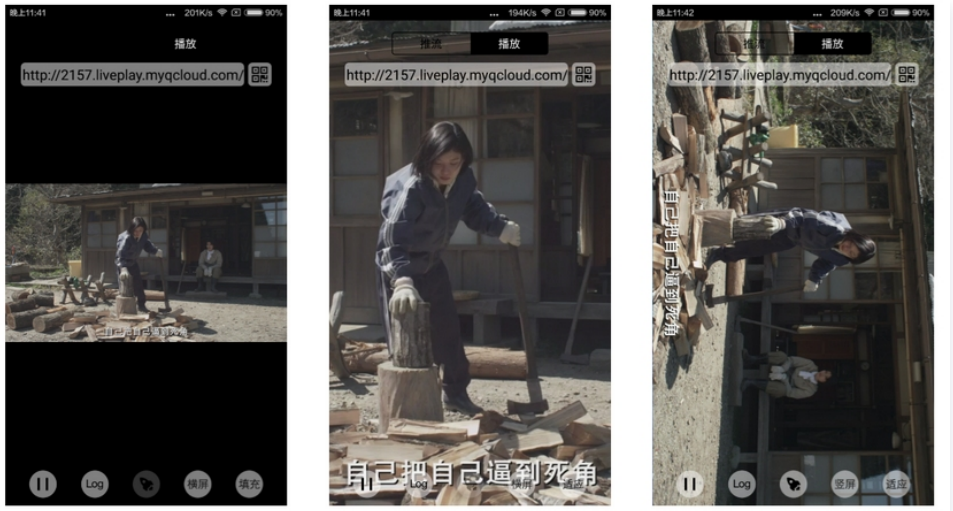
step 6：画面调整

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

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

- **setRenderRotation：画面旋转**

可选值	含义
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）



step 7：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，这是和点播对比的最大不同点（点播播放器的暂停和继续与播放本地视频文件时的表现相同）。

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

step 8：结束播放

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

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

step 9：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：

- 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”紧密同步。
- 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。
- 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈、划线。

通过如下方案可以使用此功能：

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

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

```
        }
    }];
}
```

step 10：屏幕截图

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

step 11：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（例如云点播系统）发布出去，这样就可以在微信朋友圈等社交平台上以 UGC 消息的形式进行传播。


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

- 录制的进度以时间为单位，由 TXVideoRecordListener 的 `onRecordProgress` 通知出来。
- 录制好的文件以 MP4 文件的形式，由 TXVideoRecordListener 的 `onRecordComplete` 通知出来。
- 视频的上传和发布由 TXUGCPublish 负责，具体使用方法请参见 短视频-文件发布 。

step 12：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，可以提高观看画质。
传统切流方式一般是重新播放，会导致切换前后画面衔接不上、黑屏、卡顿等问题。使用无缝切换方案，在不中断直播的情况下，能直接切到另条流上。
清晰度切换在直播开始后，任意时间都可以调用。调用方式如下：

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

> **说明**
> 清晰度无缝切换功能需要在后台配置 PTS 对齐，如您需要可 提交工单 申请使用。

step 13：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，随时回退到任意直播历史时间点观看，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。

首先在官网了解时移基本概念，并开通时移功能。

```
// 按照文档拼接时移的播放地址
NSString *timeShiftUrl = @"http://[Domain]/timeshift/[AppName]/[StreamName]/timeshift.m3u8?
delay=xxx";
[_txLivePlayer stopPlay]; // 停止正在播放的直播流
[_txVodPlayer startVodPlay:timeShiftUrl]; // 播放时移流

// 返回直播
[_txVodPlayer stopPlay]; // 停止正在播放的时移流
[_txLivePlayer startLivePlay:liveUrl type:PLAY_TYPE_LIVE_FLV]; // 播放直播流
```

接入时移功能时需要在后台打开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。

> **说明**
> 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（例如绝地求生）非常适合，卡顿率最低

自动模式	网络自适应	2s-8s	混合场景	观众端的网络越好，延迟就越低；观众端网络越差，延迟就越高

- **三种模式的对接代码**

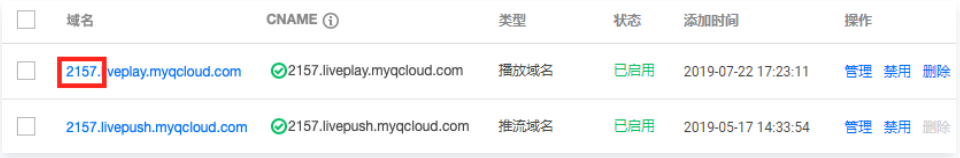
```
TXLivePlayConfig*  _config = [[TXLivePlayConfig alloc] init];
//自动模式
_config.bAutoAdjustCacheTime   = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 5;
//极速模式
_config.bAutoAdjustCacheTime   = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 1;
//流畅模式
_config.bAutoAdjustCacheTime   = NO;
_config.minAutoAdjustCacheTime = 5;
_config.maxAutoAdjustCacheTime = 5;

[_txLivePlayer setConfig:_config];

//设置完成之后再启动播放
```

> **说明**
> 更多关于卡顿和延迟优化的技术知识，请参见 如何优化视频卡顿 。

超低延时播放

支持400ms左右的超低延迟播放是云直播播放器的一个特点，它可以用于一些对延时要求极为苛刻的场景，例如**远程夹娃娃**或者**主播连麦**等，关于这个特性，您需要知道：

- **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和 bizid 参数，防盗链签名的计算方法请参见 防盗链计算 。
  bizid 的获取需要进入 域名管理 页面，在默认域名中出现的第一个数字即为 bizid，如图所示：



  如果您的防盗链地址为：
  `rtmp://domain/live/test?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1bd34` 。
  则加速流地址为：
  `rtmp://domain/live/test?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1bd34&bizid=2157` 。

> **说明**
> 防盗链计算默认使用推流防盗链 Key。

- **播放类型需要指定 ACC**
  在调用 startLivePlay 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（例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高于 CDN 线路的价格）。

- **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。
- **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，具体价格请参见 价格总览 。

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。

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	收到首帧数据，越快收到此消息说明链路质量越好
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始，如果您自己做 loading，会需要它
PLAY_EVT_PLAY_PROGRESS	2005	播放进度，如果您在直播中收到此消息，可以忽略
PLAY_EVT_PLAY_LOADING	2007	视频播放进入缓冲状态，缓冲结束之后会有 PLAY_BEGIN 事件
PLAY_EVT_START_VIDEO_DECODER	2008	视频解码器开始启动（2.0 版本以后新增）
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率发生变化（分辨率在 EVT_PARAM 参数中）
PLAY_EVT_GET_PLAYINFO_SUCC	2010	如果您在直播中收到此消息，可以忽略
PLAY_EVT_CHANGE_ROTATION	2011	如果您在直播中收到此消息，可以忽略
PLAY_EVT_GET_MESSAGE	2012	获取夹在视频流中的自定义 SEI 消息，消息的发送需使用 TXLivePusher
PLAY_EVT_STREAM_SWITCH_SUCC	2015	直播流切换完成，请参见 清晰度无缝切换

不要在收到 PLAY_LOADING 后隐藏播放画面
因为 `PLAY_LOADING -> PLAY_BEGIN` 的等待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背景透明的 loading 动画。

结束事件

事件ID	数值	含义说明
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?
RTMP 协议中规定了直播结束事件，但是 HTTP-FLV 则没有，如果您在播放 FLV 的地址时直播结束了，可预期的 SDK 的表现是：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。

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）
PLAY_WARNING_RECV_DATA_LAG	2104	网络来包不稳：可能是下行带宽不足，或由于主播端出流不均匀
PLAY_WARNING_VIDEO_PLAY_LAG	2105	当前视频播放出现卡顿
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	当前视频帧不连续，可能丢帧
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS 解析失败（仅播放 RTMP 地址时会抛送）
PLAY_WARNING_SEVER_CONN_FAIL	3002	RTMP 服务器连接失败（仅播放 RTMP 地址时会抛送）
PLAY_WARNING_SHAKE_FAIL	3003	RTMP 服务器握手失败（仅播放 RTMP 地址时会抛送）

获取视频分辨率

通过 onPlayEvent 通知的 **PLAY_EVT_CHANGE_RESOLUTION** 事件可以获取视频流当前的宽高比，这是获取视频流分辨率的最快速办法，大约会在启动播放后的100ms - 200ms左右就能得到。
视频的宽高信息位于 TXLivePlayListener 的 `onPlayEvent:(int)EvtID withParam:(NSDictionary*)param;` 通知中的 **param** 参数中。

参数	含义	数值
EVT_PARMA1	视频宽度	分辨率数值，如1920
EVT_PARMA2	视频高度	分辨率数值，如1080

定时触发的状态通知

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

Android

最近更新时间：2024-11-28 17:05:53

基础知识

本文主要介绍视频云 SDK 的直播播放功能。

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”开头，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 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：下载 SDK 开发包

下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

step 2：给 SDK 配置 License 授权

1. 获取 License 授权：
   - 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。

- 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。

2. 在您的 App 调用 SDK 相关功能之前（建议在 Application 类中）进行如下设置：

```
public class MApplication extends Application {

@Override
public void onCreate() {
 super.onCreate();
 String licenceURL = ""; // 获取到的 licence url
 String licenceKey = ""; // 获取到的 licence key
 TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
 TXLiveBase.setListener(new TXLiveBaseListener() {
         @Override
         public void onLicenceLoaded(int result, String reason) {
             Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
         }
     });
}
```

> ⚠ **注意：**
> License 中配置的 packageName 必须和应用本身一致，否则会播放失败。

step 3：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="visible"/>
```

step 4：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 5：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_LIVE_HLS	7	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 6：画面调整

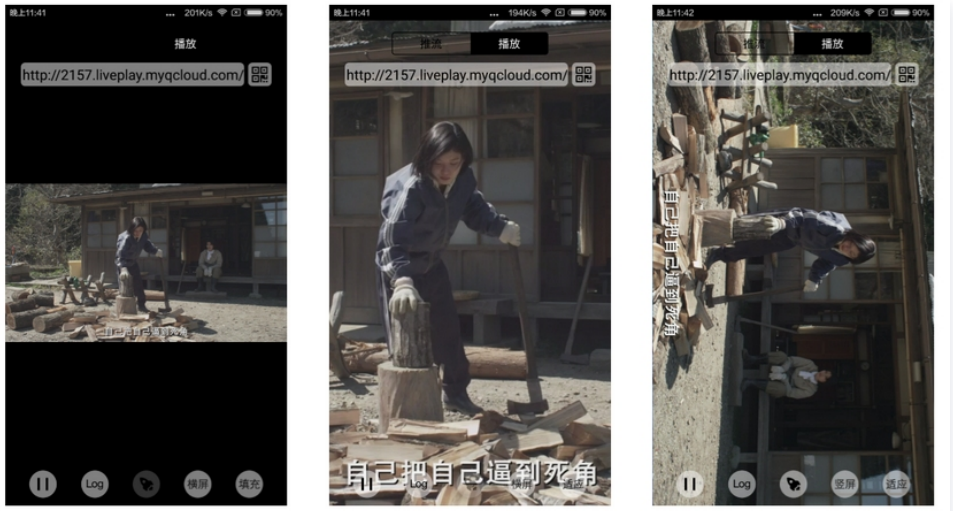
- **view：大小和位置**
  如需修改画面的大小及位置，直接调整 step3 中添加的 `video_view` 控件的大小和位置即可。
- **setRenderMode：铺满or适应**

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 7：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，这是和点播对比的最大不同点（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 8：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 9：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：

- 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”高度协调同步。
- 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。
- 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 10：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 11：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（例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，具体使用方法请参见 视频上传（Android）。

step 12：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，可以提高观看画质。
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_62fe94d692ab11e791eae435c87
f075e_900.flv");
```

当 switchStream() 方法没有回调时，则需要检查返回值，如果 URL 相同或上一个切换没完成，则切换时会返回错误。错误码解析请参见 错误码表。

> **说明：**
> 清晰度无缝切换功能需要在后台配置 PTS 对齐，如您需要可 提交工单 申请使用。

step 13：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。

首先在官网了解时移基本概念，并开通时移功能。

```
// 按照文档拼接时移的播放地址
String timeShiftUrl = "http://[Domain]/timeshift/[AppName]/[StreamName]/timeshift.m3u8?delay=xxx";
mLivePlayer.stopPlay(); // 停止正在播放的直播流
mVodPlayer.startVodPlay(timeShiftUrl); // 播放时移流

// 返回直播
mVodPlayer.stopPlay(); // 停止正在播放的时移流
mLivePlayer.startLivePlay(liveUrl, TXLivePlayer.PLAY_TYPE_LIVE_FLV); // 播放直播流
```

接入时移功能需要在后台打开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。

> **说明：**
> 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（例如绝地求生）非常适合，卡顿率最低
自动模式	网络自适应	2s – 8s	混合场景	观众端的网络越好，延迟就越低；观众端网络越差，延迟就越高

- **三种模式的对接代码**

```
TXLivePlayConfig 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);
```

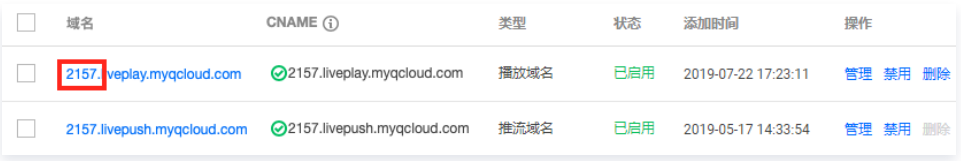
```
//设置完成之后再启动播放
```

> **说明：**
> 更多关于卡顿和延迟优化的技术知识，可以阅读 如何优化视频卡顿 。

超低延时播放

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

- **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和 bizid 参数，防盗链签名的计算方法请参见 防盗链计算 。
  bizid 的获取需要进入 域名管理 页面，在默认域名中出现的第一个数字即为 bizid，如图所示：



  如果您的防盗链地址为：
  `rtmp://domain/live/test?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1bd34`
  则加速流地址为：
  `rtmp://domain/live/test?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1bd34&bizid=2157`

> **说明：**
> 防盗链计算默认使用推流防盗链 Key。

- **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（例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。
- **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，具体价格请参见 价格总览 。

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。

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	收到首帧数据，越快收到此消息说明链路质量越好
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始，如果您自己做 loading，会需要它

PLAY_EVT_PLAY_LOADING	2007	视频播放进入缓冲状态，缓冲结束之后会有 PLAY_BEGIN 事件
PLAY_EVT_START_VIDEO_DECODER	2008	视频解码器开始启动（2.0 版本以后新增）
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率发生变化（分辨率在 EVT_PARAM 参数中）
PLAY_EVT_GET_MESSAGE	2012	获取夹在视频流中的自定义 SEI 消息，消息的发送需使用 TXLivePusher
PLAY_EVT_STREAM_SWITCH_SUCC	2015	直播流切换完成，请参见 清晰度无缝切换

不要在收到 PLAY_LOADING 后隐藏播放画面

因为 `PLAY_LOADING -> PLAY_BEGIN` 的等待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背景透明的 loading 动画。

结束事件

事件ID	数值	含义说明
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?

RTMP 协议中规定了直播结束事件，但是 HTTP-FLV 则没有，如果您在播放 FLV 的地址时直播结束了，可预期的 SDK 的表现是：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。

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）
PLAY_WARNING_RECV_DATA_LAG	2104	网络来包不稳：可能是下行带宽不足，或由于主播端出流不均匀
PLAY_WARNING_VIDEO_PLAY_LAG	2105	当前视频播放出现卡顿
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	当前视频帧不连续，可能丢帧
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS 解析失败（仅播放 RTMP 地址时会抛送）
PLAY_WARNING_SEVER_CONN_FAIL	3002	RTMP 服务器连接失败（仅播放 RTMP 地址时会抛送）
PLAY_WARNING_SHAKE_FAIL	3003	RTMP 服务器握手失败（仅播放 RTMP 地址时会抛送）

获取视频分辨率

通过 onPlayEvent 通知的 **PLAY_EVT_CHANGE_RESOLUTION** 事件可以获取视频流当前的宽高比，这是获取视频流分辨率的快速办法，大约会在启动播放后的100ms - 200ms左右就能得到。
视频的宽高信息位于 TXLivePlayListener 的 `onPlayEvent(int event, Bundle param)` 通知中的 **param** 参数中。

参数	含义	数值
EVT_PARAM1	视频宽度	分辨率数值，如1920
EVT_PARAM2	视频高度	分辨率数值，如1080

定时触发的状态通知

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

连麦互动（RTMP 方案）
费用说明

最近更新时间：2025-01-22 11:06:02

本方案为通过 RTMP_ACC 实现的旧版移动直播连麦方案，推荐您优先使用 RTC 连麦新方案 。如果您的直播业务中有使用到互动连麦功能和低延时播放（RTMP_ACC）功能 ，涉及终端包括微信小程序端、iOS 端以及 Android 端，那么需要按照移动直播连麦服务计费。

注意：
- 移动直播中的 CDN 线路依然按照云直播带宽或者流量计费，连麦服务仅对连麦通话和低延时播放链路进行计费。
- 连麦互动功能支持中国内地（大陆）地区使用，**暂不支持中国港澳台/境外地区**。

预付费连麦资源包

移动直播连麦资源包规格	资源包规格（分钟数）	有效期（月）	原价（元）	折扣	价格（元）	技术支持方式
体验版（每个用户仅限购买一次）	3000	12	48	20%	9.8	文档&工单
入门版	50000	12	800	95%	758	文档&工单
标准版	250000	12	4000	90%	3598	文档&工单
企业版	1000000	12	16000	85%	13598	文档&工单
尊享版	4000000	12	64000	80%	51198	单独拉群专人支持

计费说明

- 仅限老用户前往 **云直播控制台** > **直播 SDK** > **直播连麦**，单击**购买连麦包**。
- 连麦资源包在您支付购买后立即生效，有效期为**1年**，到期资源包若有余量则一次性扣除。
- 每天出日结连麦账单（具体以账单时间为准）扣除前一天连麦费用时，优先抵扣测试/正式资源包余量，超出部分按后付费日结使用时长出账。资源包购买当日不支持抵扣前一日的计费用量。
- 支持购买多个资源包，可叠加使用，有效期不叠加，实际抵扣按最早过期时间依次抵扣。
- 每个客户限制只能购买**1次**体验版连麦资源包，体验包用尽次日出账后自动暂停连麦服务。
- 购买正式版连麦资源包开通连麦服务，正式包用尽后继续使用连麦服务则按日结后付费价格结算，不自动暂停连麦服务。
- 当且仅当移动直播连麦计费方式为日结计费时，可使用连麦资源包进行时长抵扣。
- 当计费方式为月结计费时，资源包余量冻结，不参与抵扣，冻结期间不延长有效期。联系商务经理切换为 连麦日结后付费 后，资源包可继续使用。
- 资源包仅限中国内地（大陆）使用，**连麦功能不支持中国港澳台/境外地区**。
- 以上套餐价格涵盖范围为视频分辨率小于等于720P的场景，高于720P的需求需单独联系客服评估。
- 直播连麦分钟数按照每个终端的观看时长计算，当多人连麦时，最终连麦计费时长为每个人观看其他人直播分钟数之和。

计费示例

A、B 及 C 三个人进行视频连麦，其中 A 和 B 两人连麦10分钟，C 从第5分钟进入连麦房间，并且同时观看 A 和 B 的视频直播画面，具体计费规则为：

- A 观看 B 时间为10分钟，观看 C 时间为5分钟，A 的费用为（10分钟 + 5分钟）× 0.016元/分钟 = 0.24元。
- B 观看 A 时间为10分钟，观看 C 时间为5分钟，B 的费用为（10分钟 + 5分钟）× 0.016元/分钟 = 0.24元。
- C 观看 A 时间为5分钟，观看 B 时间为5分钟，C 的费用为（5分钟 + 5分钟）× 0.016元/分钟 = 0.16元。

- 故本次连麦总费用为 0.24 + 0.24 + 0.16 = 0.64元。

后付费模式

计费项	单价	单位	付费方式	备注
视频通话720P及以下（含纯语音）	0.016	元/分钟	后付费日结	该价格涵盖范围为视频分辨率小于等于720P的场景，高于720P的需求需单独联系客服评估报价。

计费说明

- 计费项：移动直播连麦时长。
- 计费方式：后付费计费，购买正式连麦套餐包开通服务后，套餐包用尽自动转日结后付费。
- 计费周期：按日计费，每天出日结连麦账单（具体以账单时间为准），扣除前一天的连麦费用。
- 计费规则：直播连麦分钟数按照每个终端的观看时长计算，当多人连麦时，最终连麦计费时长为每个人观看其他人直播分钟数之和。

计费示例

A、B 及 C 三个人进行视频连麦，其中 A 和 B 两人连麦10分钟，C 从第5分钟进入连麦房间，并且同时观看 A 和 B 的视频直播画面，具体计费规则为：

- A 观看 B 时间为10分钟，观看 C 时间为5分钟，A 的费用为（10分钟 + 5分钟）× 0.016元/分钟 = 0.24元。
- B 观看 A 时间为10分钟，观看 C 时间为5分钟，B 的费用为（10分钟 + 5分钟）× 0.016元/分钟 = 0.24元。
- C 观看 A 时间为5分钟，观看 B 时间为5分钟，C 的费用为（5分钟 + 5分钟）× 0.016元/分钟 = 0.16元。
- 故本次连麦总费用为 0.24 + 0.24 + 0.16 = 0.64元。

iOS + Android

最近更新时间：2024-11-21 10:44:02

功能介绍

TXLivePusher 和 TXLivePlayer 这两个基础组件可以比较容易的实现推流和拉流功能，但如果想要实现复杂的直播连麦功能，就需要借助我们提供给您的 MLVBLiveRoom 组件，该组件基于云直播和即时通信（IM）两个 PAAS 服务组合而成，支持：

- 主播创建新的直播间开播，观众进入直播间观看。
- 主播和观众进行视频连麦互动。
- 两个不同房间的主播 PK 互动。
- 每一个直播间都有一个不限制房间人数的聊天室，支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。

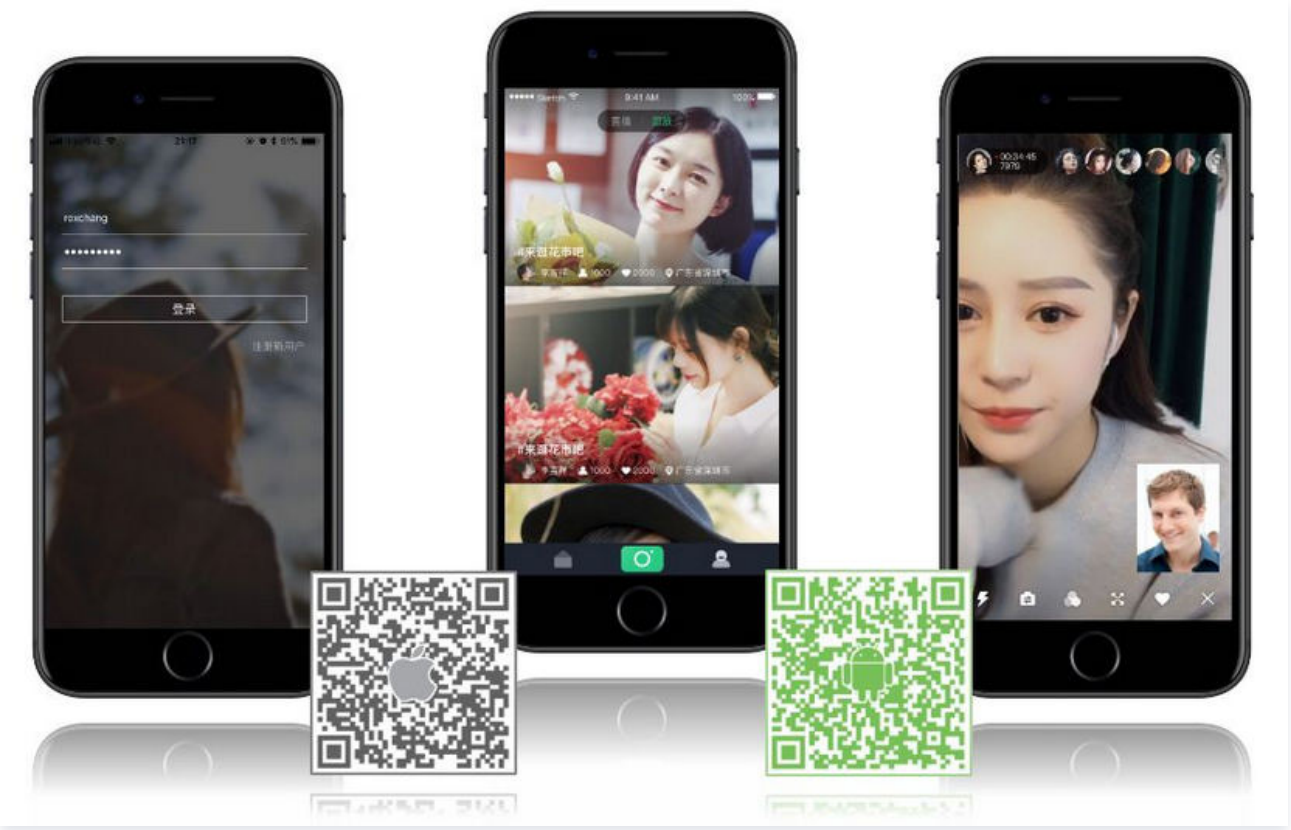
> **说明：**
> 连麦互动功能支持中国内地（大陆）地区使用，暂不支持中国港澳台/境外地区。

功能体验

我们提供了 iOS、Android 以及微信小程序三个平台上的直播连麦体验，它们均是使用 MLVBLiveRoom 组件实现的直播加连麦功能：

- **iOS**
  进入 App Store 安装应用“小直播”，注册一个账号即可开始体验。

- **Android**
  下载 apk 安装包，安装“小直播”，注册一个账号即可开始体验。
- **微信小程序**
  打开微信，选择**发现** > **小程序**，搜索“腾讯视频云”，单击“移动直播”功能即可体验。



代码对接

Step1. 下载 LiteAVSDK 和 MLVBLiveRoom 组件

直播 SDK 提供的连麦能力需要依赖三个组件：

- LiteAVSDK：闭源，负责直播推流，直播拉流，以及连麦视频通话功能。
- TIMSDK：闭源，负责构建直播聊天室，以及聊天室中用户之间的消息传输功能。
- MLVBLiveRoom：开源，基于 LiteAVSDK 和 TIMSDK 搭建一个支持连麦互动和消息互动的“直播间”。

我们已将上述组件均托管在 Github 上，clone 下来便可使用，关键组件的具体获取路径如下表所示：

所属平台	LiteAVSDK	TIMSDK	MLVBLiveRoom 组件	示例代码
iOS	MLVBSDK	TIMSDK	MLVBLiveRoom	SimpleCode
Android	MLVBSDK	TIMSDK	MLVBLiveRoom	SimpleCode

> **说明：**
> 除上述示例外，针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。
> - **IOS：** MLVB-API-Example
> - **Android：** MLVB-API-Example

Step2. 申请 License

下载 LiteAVSDK 后需要 License 授权才能使用，请阅读 License 申请 了解 License 的申请方法和使用方法。

- iOS
  建议在 `[AppDelegate application:didFinishLaunchingWithOptions:]` 中添加：

```
[TXLiveBase setLicenceURL:LicenceUrl key:Key];
```

- Android
  建议在 application 中添加：

```
TXLiveBase.getInstance().setLicence(context, LicenceUrl, Key);
```

Step3. 购买连麦套餐包

由于连麦功能会使用到高速专线来降低音视频传输延迟，这部分功能需要额外购买套餐包才能开通，否则直播 SDK 的各端 SDK 只能使用云直播的普通服务（推流和拉流），并不能开启连麦功能。

- 仅限老用户前往 **云直播控制台** > **直播 SDK** > **直播连麦**，单击**购买连麦包**。
- 移动直播连麦计费说明

> **注意：**
> - **旧版移动直播连麦资源包已下线，仅限连麦老用户购买。**
> - 默认开通连麦服务需要先购买正式的连麦套餐包（非体验包），开通后可选择继续购买套餐包进行消费抵扣，也可以按照后付费日结计费进行结算。
> - 若您购买的套餐包为体验包，用尽后次日会自动停止连麦服务，超出部分仍会按照后付费计费。
> - 使用连麦的**双方**是**按连麦时长计费**，普通观众观看会先通过直播混流，将连麦画面混合后通过 CDN 播放来降低播放成本。

- 观众观看混流后产生的直播费用，按照直播流量/带宽计费。
- 直播混流功能会产生标准转码费用，按照输出的分辨率和时长计费。

Step4. 在应用管理中添加一个新的应用

进入 **云直播控制台** > **直播SDK** > **应用管理**，单击**创建应用**。待应用创建完成后，记录其 SDKAPPID 信息。

> **说明**
> 该操作的目的是创建一个即时通信 IM 应用，并将当前直播账号和该即时通信 IM 应用绑定起来。即时通信 IM 应用能为小直播 App 提供聊天室和连麦互动的能力。

Step5. 登录房间服务

MLVBLiveRoom 单靠一个终端的组件无法独自运行，它依赖一个后台服务为其实现房间管理和状态协调，这个后台服务我们称之为**房间服务**（RoomService）。而要使用这个房间服务，MLVBLiveRoom 就需要先进行**登录**（login）。
MLVBLiveRoom 的 login 函数需要指定相关参数：

参数	类型	填写方案
sdkAppID	数字	当前应用的 AppID，在 Step4 中可以获取到。
userID	字符串	当前用户在您的账号系统中的 ID。
userName	字符串	用户名（昵称）。
userAvatar	字符串	用户头像的 URL 地址。
userSig	字符串	登录签名，计算方法请参见 计算 UserSig 。

> **说明：**
> - 由于 login 是一个需要跟后台服务器通讯的过程，建议等待 login 函数的异步回调后再调用其他函数。
> - 后台接口限制并发为每秒100次请求，若您有高并发请求请提前 联系我们 处理，避免影响服务调用。

Step6. 获取房间列表（非必需）

> **说明：**
> 如果您希望使用自己的房间列表，该步骤可跳过，但需要您在 Step7 中自行指定 roomID。为避免房间号重复，建议使用主播的 userID 作为 roomID。

不管是主播还是观众都需要有一个房间列表，调用 MLVBLiveRoom 的 **getRoomList** 接口可以获得一个简单的房间列表：
- 当主播通过 `createRoom` 创建一个新房间时，房间列表中会相应地增加一条新的房间信息。
- 当主播通过 `exitRoom` 退出房间时，房间列表中会移除该房间。

列表中每个房间都有对应的 roomInfo，由主播通过 `createRoom` 创建房间时传入，为提高扩展性，建议将 roomInfo 定义为 JSON 格式。

Step7. 主播开播

主播开播前，需要先调用 MLVBLiveRoom 中的 **startLocalPreview** 接口开启本地摄像头预览，该函数需要传入 view 对象用于显示摄像头的视频影像，这期间 MLVBLiveRoom 会申请摄像头使用权限。同时，主播也可以对着摄像头调整美颜和美白的具体效果。
然后调用 **createRoom** 接口，MLVBLiveRoom 会在后台的房间列表中新建一个直播间，同时主播端会进入直播状态。

> **说明：**
> 为避免房间号重复，建议使用主播的 userID 作为 roomID。如果您不手动设置 roomID，后台将会自动为您分配一个 roomID。
> 如果您想要管理房间列表，可以先由您的服务器确定 roomID，再通过 createRoom、enterRoom 和 exitRoom 接口使用 MLVBLiveRoom 的连麦能力。

Step8. 观看直播

观众通过 MLVBLiveRoom 中的 **enterRoom** 接口可以进入直播间观看视频直播，enterRoom 函数需要传入 view 对象用于显示直播流的视频影像。
进入房间后，通过调用 **getAudienceList** 接口可以获取观众列表。如果少于30名观众，列表会展示全部观众信息。如果多于30名观众，列表仅展示新进入房间的30名观众的信息。

Step9. 弹幕消息

MLVBLiveRoom 包装了 TIMSDK 的消息发送接口，您可以通过 **sendRoomTextMsg** 函数发送普通的文本消息（用于弹幕），也可以通过 **sendRoomCustomMsg** 发送自定义消息（用于点赞，送花等等）。

> **注意：**
> 腾讯云 IM 的直播聊天室，每秒钟最多可以收取40条的消息。如果您以高于40条/次的速度刷新 UI 上的弹幕界面，很容易导致 CPU 100%，请注意控制刷新频率，避免高频刷新。

Step10. 观众与主播连麦

步骤	角色	详情
第一步	观众	观众调用 `requestJoinAnchor()` 向主播发起连麦请求。
第二步	主播	主播会收到 `MLVBLiveRoomDelegate#onRequestJoinAnchor(AnchorInfo, String)` 通知，之后可以展示一个 UI 提示，询问主播要不要接受连麦。
第三步	主播	主播调用 `responseJoinAnchor()` 确定是否接受观众的连麦请求。

第四步	观众	观众会收到 `MLVBLiveRoomDelegate.RequestJoinAnchorCallback` 回调通知，得知请求是否被同意。
第五步	观众	观众如果请求被同意，则调用 startLocalPreview() 开启本地摄像头，如果 App 还没有取得摄像头和麦克风权限，会触发 UI 提示用户授权摄像头和麦克风的使用权限。
第六步	观众	观众调用 `joinAnchor()` 正式进入连麦状态。
第七步	主播	当观众进入连麦状态后，主播就会收到 `MLVBLiveRoomDelegate#onAnchorEnter(AnchorInfo)` 通知。
第八步	主播	主播调用 `startRemoteView()` 就可以看到连麦观众的视频画面。
第九步	观众	如果直播间里已经有其他观众正在跟主播进行连麦，那么新加入的这位连麦观众也会收到 `onAnchorJoin()` 通知，用于展示（ `startRemoteView()` ）其他连麦者的视频画面。
第九步	主播或观众	主播或观众随时都可以通过 `quitJoinAnchor()` 接口退出连麦状态，同时，主播还可以通过 `kickoutJoinAnchor()` 接口移除连麦观众。

说明：
MLVBLiveRoom 在设计上最多支持10人同时连麦，但出于兼容低端 Android 终端和实际体验效果的考虑，建议将同时连麦人数控制在6人以下。

Step11. 主播间跨房间 PK

主播间跨房 PK 常被用于活跃直播平台的氛围，提升打赏频率，对平台的主播人数有一定要求。目前常见的主播 PK 方式是将所有愿意 PK 的主播“圈”在一起，再后台进行随机配对，每次 PK 都有一定时间要求，例如5分钟，超过后即结束 PK 状态。
由于我们暂时未在 MLVBLiveRoom 的房间服务里加入配对逻辑，因此目前仅提供了基于客户端 API 接口的简单 PK 流程，您可以通过即时通信 IM 服务的消息下发 REST API 接口，由您的配对服务器，将配对开始、配对结束等指令发送给指定的主播，从而实现服务器控制的目的。如果采用此种控制方式，下述步骤中的第三步实现为默认接受即可。

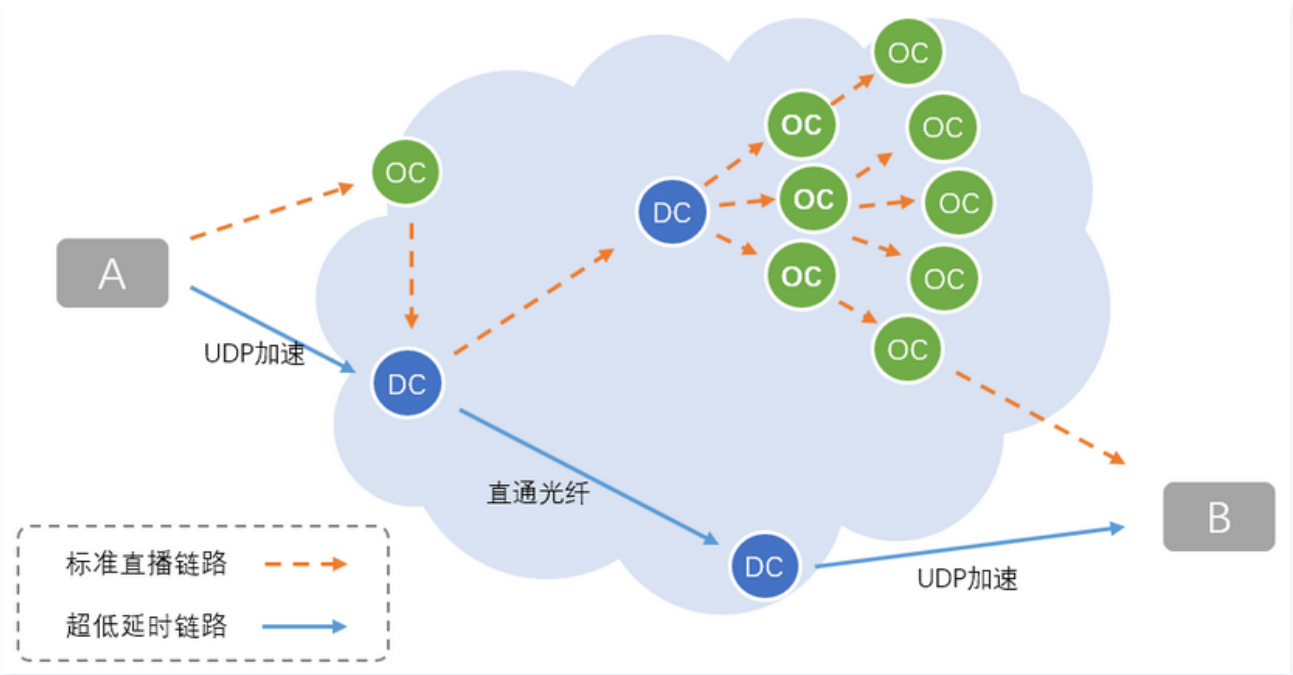
步骤	角色	详情
第一步	主播 A	主播 A 调用 `requestRoomPK()` 向主播 B 发起连麦请求。
第二步	主播 B	主播 B 会收到 `MLVBLiveRoomDelegate#onRequestRoomPK(AnchorInfo)` 回调通知。
第三步	主播 B	主播 B 调用 `responseRoomPK()` 确定是否接受主播 A 的 PK 请求。如果采用服务器配对的 PK 方案，此处可以默认接受，不需要由主播 B 来决策。
第四步	主播 B	主播 B 在接受主播 A 的请求后，即可调用 `startRemoteView()` 来显示主播 A 的视频画面。
第五步	主播 A	主播 A 会收到 `MLVBLiveRoomDelegate.RequestRoomPKCallback` 回调通知，可以得知请求是否被同意，如果请求被同意，则可以调用 `startRemoteView()` 显示主播 B 的视频画面。
第六步	主播 A 或 B	主播 A 或 B 均可以通过调用 `quitRoomPK()` 接口结束 PK 状态。

常见问题

直播 SDK 是不是使用 RTMP 协议进行连麦?

不是。腾讯云采用了两种传输通道才实现了直播 + 连麦功能：

- 直播采用标准的 HTTP-FLV 协议，使用标准 CDN 线路，没有并发观看人数的限制，且带宽成本很低，但延迟一般在3s以上。
- 连麦采用 UDP 协议，使用专用加速线路，延迟一般在500ms以内，但由于线路成本较高，因此采用连麦时长进行计费。



通道	直播通道	连麦通道
通讯延迟	≥ 3s	≤ 500ms
底层协议	HTTP-FLV 协议	UDP 协议
价格/费用	按带宽计费	按时长计费
最高并发	无上限	≤ 10人
TXLivePusher	setVideoQuality 为 SD、HD、FHD	setVideoQuality 为 MAIN_PUBLISHER、SUB_PUBLISHER
TXLivePlayer	PLAY_TYPE_LIVE_FLV	PLAY_TYPE_LIVE_RTMP_ACC
播放URL	普通的 FLV 地址	带防盗链签名的 RTMP-ACC 地址

小程序

最近更新时间：2025-01-22 11:06:02

功能介绍

<live-pusher> 和 <live-player> 这两个基础标签可以比较方便的实现推流和拉流功能，但如果想要实现复杂的直播连麦功能，就需要借助我们提供给您的 **<mlvb-live-room>** 小程序组件，该组件基于云直播和即时通信（IM）两款 PAAS 产品组合而成，支持：

- 主播创建新的直播间开播，观众进入直播间观看。
- 主播和观众进行视频连麦互动。
- 每一个直播间都有一个不限制房间人数的聊天室，支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。

说明：
连麦互动功能支持中国内地（大陆）地区使用，暂不支持中国港澳台/境外地区。

功能体验

我们提供了 iOS、Android 以及微信小程序三个平台上的直播连麦体验，它们都是使用 MLVBLiveRoom 组件实现的直播加连麦功能：

- **微信小程序**
  打开微信，选择**发现** > **小程序**，搜索“腾讯视频云”，单击“移动直播”功能即可体验。
- **iOS**
  进入 App Store 安装应用“小直播”，注册一个账号即可开始体验。

- Android
  下载 Apk 安装包，安装应用“小直播”，注册一个账号即可开始体验。

计费说明

小程序 <mlvb-live-room> 组件为开发者提供了云直播连麦的能力，具有低卡顿、低延时和易接入等特点。如果您希望用它来快速实现直播连麦应用，那么您需要购买直播流量资源包、直播 SDK License、移动直播连麦资源包和即时通信 IM 套餐包。详细计费说明请查看 云直播计费说明 和 即时通信 IM 定价 。

> ⚠ **注意：**
> 旧版移动直播连麦资源包已下线，仅限连麦老用户购买。

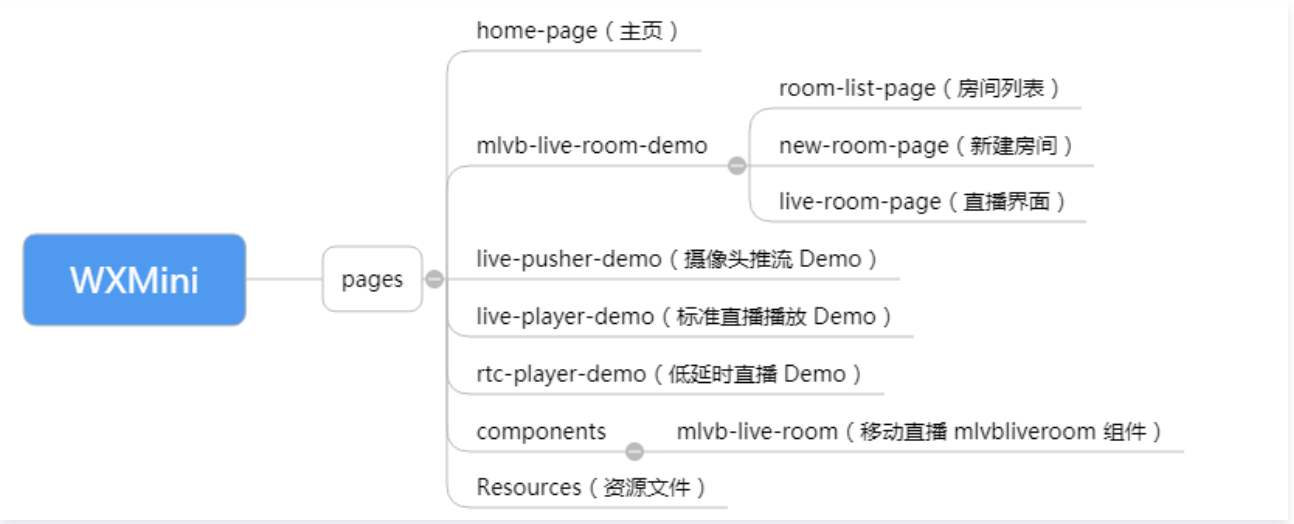
示例代码

- 示例代码：Github
- 组件 API：<mlvb-live-room>

接入指引

Step1. 下载 <mlvb-live-room> 小程序组件

单击 Github 下载 <mlvb-live-room> 组件代码，其目录结构如下：



Step2. 在工程中引入组件

在当前页面的 JSON 配置文件里必须引用 <mlvb-live-room> 组件，因为 <mlvb-live-room> 并非微信小程序的原生标签，需要用 `usingComponents` 指定加载路径。

```
"usingComponents": {
      "mlvb-live-room": "/pages/components/mlvb-live-room/mlvbliveroomview"
 }
```

Step3. 购买连麦套餐包

由于连麦功能会使用到高速专线来降低音视频传输延迟，这部分功能需要额外购买套餐包才能开通，否则移动直播的各端 SDK 只能使用云直播的普通服务（推流和拉流），并不能开启连麦功能。

- 仅限老用户前往 **云直播控制台** > **直播 SDK** > **直播连麦**，单击**购买连麦包**
- **移动直播连麦计费说明**

Step4. 在应用管理中添加一个新的应用

选择 **云直播控制台** > **直播SDK** > **应用管理**，单击**创建应用**开始创建一个新的应用。

应用创建完成后要记录好 SDKAppID，这个 ID 会在 Step5 中使用到。

> **说明：**
> 这一步的目的是创建一个 TIM 即时通信 IM 应用，并将当前直播账号和即时通信 IM 应用绑定起来，即时通信 IM 应用主要提供聊天室和连麦互动的能力。

Step5. 登录房间服务（必需）

- 在使用 <mlvb-live-room> 标签前需要先调用 mlvbliveroomcore.js 中的 `login()` 方法进行登录。
- <mlvb-live-room> 单靠 Github 上的这些 javascript 代码是无法独自运行的，它依赖一个后台服务为其实现房间管理和主播间的状态同步，这个后台服务我们称之为**房间服务**（RoomService）。而要使用这个房间服务，<mlvb-live-room> 就需要先进行登录（login）。
- <mlvb-live-room> 的 login 函数需要指定一些参数，这些参数的填写方式如下：

参数	类型	填写方案
sdkAppID	数字	当前应用的 AppID，在 Step4 中可以获取到。
userID	字符串	当前用户在您的账号系统中的 ID。
userName	字符串	用户名（昵称）。
userAvatar	字符串	用户头像的 URL 地址。
userSig	字符串	登录签名，计算方法请参见 计算 UserSig。

```
//如下示例代码中的路径地址可能与您当前工程配置不符，请根据当前工程的具体情况进行相应地修改
var liveroom = require('components/mlvb-live-room/mlvbliveroomcore.js');
...
var loginInfo = {
        sdkAppID: this.sdkAppID,
        userID: this.userID,
        userSig: this.userSig,
        userName: this.userName,
        userAvatar: this.userAvatar
}
liveroom.login({
        data: loginInfo,
        success: options.success,
        fail: options.fail
    });
```

> ⚠ **注意：**
> 后台接口限制并发为每秒100次请求，若您有高并发请求请提前 联系我们 处理，避免影响服务调用。

Step6. 获取房间列表（非必需）

您可以通过调用 mlvbliveroomcore.js 中的 `getRoomList` 方法来获取房间列表。

```
var liveroom = require('components/mlvb-live-room/mlvbliveroomcore.js');
...
liveroom.getRoomList({
    data: {
        index: 0,   //列表索引号
        cnt: 20   //要拉取的列表个数
    },
    success: function (ret) {
        console.log('获取房间列表:', ret.rooms);
    },
    fail: function (ret) {
        console.error('获取房间列表失败:', ret);
    }
});
```

> **说明：**
> 如果您希望使用自己的房间列表，这一步可以省略，因为您可以在 Step7 中自行指定 roomID，直播间列表的管理可以自行处理。

Step7. 在 UI 层添加视频标签

您需要在 page 目录下的 wxml 文件中添加 <mlvb-live-room> 标签。

```
<mlvb-live-room id="id_liveroom" wx:if="{{showLiveRoom}}"
                roomid="{{roomID}}" role="{{role}}" roomname="{{roomName}}"
                beauty="{{beauty}}" template="float"
                bindRoomEvent="onRoomEvent">  // 绑定事件回调函数

// 此处可以添加<cover-view>
</mlvb-live-room>
```

> **说明：**
> 示例代码请参见 room.wxml，Demo 中的直播主界面，主播和观众都共用这一个 xml 界面配置。

Step8. 主播开播和观众观看

Step7 仅仅是实现了 UI 布局相关工作，不管是主播开播，还是观众端开始观看，都需要调用 mlvbliveroomview.js 中的 `start()` 方法，并且在此之前还需要设置 <mlvb-live-room> 标签中的若干属性。

```
 self.setData({
       roomID: options.roomID,
       roomName: options.roomName,
       userName: options.userName,
       pureAudio: JSON.parse(options.pureAudio),
       role: role,
       showLiveRoom: true
  }, function () {
        self.start();        // 启动组件进行开播或者播放
})
```

> **说明：**
> 示例代码请参见 `room.js`，demo 中的主要逻辑代码，包含主播开播、观众观看以及主播和观众连麦的相关逻辑。

Step9. 连麦互动

观众可以通过调用 mlvbliveroomview.js 中的 `requestJoinAnchor()` 方法向主播发起连麦请求。

```
// 观众：可以向主播发起连麦请求
var liveroom = this.selectComponent("#id_liveroom");
liveroom.requestJoinAnchor();
```

此时主播可以通过在 Step7 中绑定的 `onRoomEvent` 函数接收到这一请求，并可以调用 mlvbliveroomview.js 中的 `respondJoinAnchor()` 函数进行“接受”或者“拒绝”的回应。

```
// 主播：可以接受或者拒绝连麦请求
var liveroom = this.selectComponent("#id_liveroom");
liveroom.respondJoinAnchor(true, audience);
```

> **说明：**
> 示例代码请参见 room.js，Demo 中的主要逻辑代码，包含主播开播、观众观看以及主播和观众连麦的相关逻辑。

Step10. 定制 UI 界面

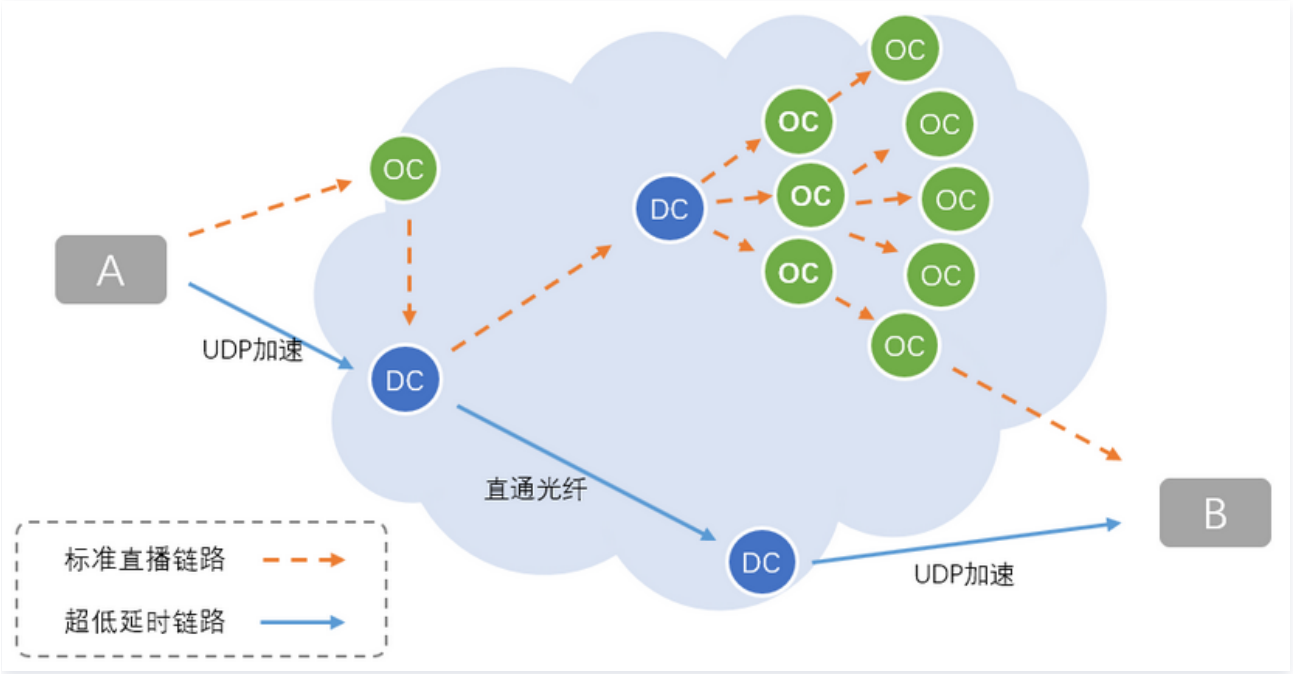
如果我们默认实现的界面布局不符合您的要求，您可以根据微信小程序的“界面模板”规范进行定制：

- 您可以直接修改 floattemplate.wxml 模板文件和 floattemplate.wxss 样式文件。
- 您也可以增加新的模板 wxml 和样式文件 wxss，但新加的文件需要在 `mlvbliveroomview.wxml` 和 `mlvbliveroomview.wxss` 两个文件中增加相关的配置。

常见问题

<mlvb-live-room> 是不是使用 RTMP 协议进行连麦?

不是的。
腾讯云采用了两种传输通道才实现了直播 + 连麦功能，其中直播采用标准的 HTTP-FLV 协议，走标准 CDN 线路，没有并发观看人数的限制，且带宽成本很低，但延迟一般在3s以上。
连麦则采用了 UDP 协议，走专用加速线路，延迟一般在500ms以内，但由于线路成本较高，因此采用连麦时长进行计费。



通道	直播通道	连麦通道
通讯延迟	⩾ 3s	⩽ 500ms
底层协议	HTTP-FLV 协议	UDP 协议
价格/费用	按带宽计费	按时长计费
最高并发	无上限	⩽ 10人
<live-pusher> 的 mode 参数	HD	RTC
<live-player> 的 mode 参数	LIVE	RTC
播放 URL	普通的 FLV 地址	带防盗链签名的 RTMP-ACC 地址

错误码

最近更新时间：2023-06-16 11:33:23

errorCode	errorMsg
200100	请求包错误，HTTP 方法错误或参数错误
200101	JSON 请求体无法解析
201001	login 操作中无 sdk_appid 或 user_sig
201002	缺少 user_sig
201003	URL 中的 user_id 与 body 中的 user_id 不一致
201004	操作缺少 room_id 参数
201005	获取 room 列表中，count 参数未设置或设置为0
201006	云端混流参数缺少 interface 等必要参数
201007	设置自定义字段时，自定义字段名称为空或者自定义操作（set/dec/inc）为空
201008	设置自定义字段时，对自定义字段的操作不是（set/dec/inc）操作
201009	set 操作，但是 value 为空
202001	Token 鉴权失败
202002	连接 IM 鉴权 server 失败
202003	IM 鉴权服务器的响应不合法
202004	登出失败，可能是后台删除记录失败
202005	sdkappid 没有对应的 appid
202006	请求太频繁导致被限频
203002	创建房间失败
203003	销毁房间失败
203004	获取房间列表失败
203005	已经在房间，但是更新房间信息失败
203006	进房失败，可能原因： • 未开通直播 • 未自定义域名 • 开通直播，并自定义了域名，可能因为缓存需要等一段时间
203007	房间内的主播个数太多
203009	房间名长度过长
203010	成员不在房间内
204001	获取推流 URL 失败，可能原因： • 未开通直播 • 未自定义域名

	• 开通直播，并自定义了域名，可能因为缓存需要等一段时间
204002	获取主播列表失败
204003	删除主播信息失败，可能原因:房间不存在
204004	获取加速流播放地址失败，可能原因: • 未开通直播 • 未自定义域名 • 开通直播，并自定义了域名，可能因为缓存需要等一段时间
205001	观众数目到达上限
205002	新增观众记录时处理失败
205003	删除观众记录失败
205004	获取观众列表失败
206001	设置心跳记录报错
207001	获取 redis 的地址出错
207002	混流失败
207003	设置自定义字段出错
207004	获取自定义字段出错
207005	不支持的用户操作
207006	不支持的房间类型

高级功能
设定画面质量

最近更新时间：2022-10-08 16:33:50

功能介绍

LiteAVSDK 提供了两种方式来设定画面质量：

- 使用 TXLivePusher 提供的 setVideoQuality 接口。
- 使用 TXLivePushConfig 中的音视频编码参数。

我们推荐使用前者，而且不建议您直接修改 TXLivePushConfig 中的音视频编码参数。

设定建议

推荐直接使用 TXLivePusher 提供的 `setVideoQuality` 接口设定画面质量：

取值	含义
YES	弱网状态时降低画质，画面较流畅
NO	弱网状态时保持画质，画面较卡顿

待废弃，推荐设置为 NO

TXLivePusher # setVideoQuality (quality, adjustBitrate, *adjustResolution*);

画质	定义	说明
标清	VIDEO_QUALITY_STANDARD_DEFINITION	采用360x640的分辨率
高清	VIDEO_QUALITY_HIGH_DEFINITION	采用540x960的分辨率
超清	VIDEO_QUALITY_SUPER_DEFINITION	采用720x1280的分辨率
蓝光	VIDEO_QUALITY_ULTRA_DEFINITION	采用1080x1920的分辨率
连麦（主）	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	进入连麦状态后，大主播使用
连麦（副）	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	进入连麦状态后，副主播使用
视频通话	VIDEO_QUALITY_REALTIEM_VIDEOCHAT	已经废弃

参数设定建议

应用场景	quality	adjustBitrate	adjustResolution
秀场直播	VIDEO_QUALITY_HIGH_DEFINITION 或 VIDEO_QUALITY_SUPER_DEFINITION	NO	NO
手游直播	VIDEO_QUALITY_SUPER_DEFINITION	YES	NO
连麦（主画面）	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	YES	NO
连麦（小画面）	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	NO	NO
蓝光直播	VIDEO_QUALITY_ULTRA_DEFINITION	NO	NO

内部指标

quality	adjustBitrate	分辨率	码率范围
VIDEO_QUALITY_STANDARD_DEFINITION	YES	360 × 640	500kbps - 900kbps
VIDEO_QUALITY_STANDARD_DEFINITION	NO	360 × 640	800kbps
VIDEO_QUALITY_HIGH_DEFINITION	YES	540 × 960	800kbps - 1500kbps
VIDEO_QUALITY_HIGH_DEFINITION	NO	540 × 960	1200kbps
VIDEO_QUALITY_SUPER_DEFINITION	YES	720 × 1280	1000kbps - 1800kbps
VIDEO_QUALITY_SUPER_DEFINITION	NO	720 × 1280	1800kbps
VIDEO_QUALITY_ULTRA_DEFINITION	YES	1080 × 1920	2500kbps - 3000kbps
VIDEO_QUALITY_ULTRA_DEFINITION	NO	1080 × 1920	3000kbps
VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	不支持设置	540 × 960	800kbps - 1500kbps
VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	不支持设置	320 × 480	350kbps

注意事项

1. 由于标准 CDN 对可变分辨率的直播流的兼容性较差，setVideoQuality 的第三个参数 adjustResolution 即将被废弃，请统一设置为 NO。
2. quality 的 `VIDEO_QUALITY_REALTIME_VIDEOCHAT` 已经废弃，如果您希望实现纯视频通话而非直播功能，推荐使用腾讯云 实时音视频 TRTC 服务。
3. 进入连麦状态后，请调用 setVideoQuality() 将 quality 挡位设置为 `VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER`（主播）或 `VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER`（连麦观众），结束连麦状态后，请再次调用 setVideoQuality() 将 quality 挡位设置为连麦前的值。
4. 当 quality 选定为 `VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER` 时，不支持设置 `adjustBitrate` 参数，内部设置为 true。
5. 当 quality 选定为 `VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER` 时，不支持设置 `adjustBitrate` 参数，内部设置为 false。

常见问题

1. 为什么观众端看到的画面没有主播端清晰?

主播端看到的画面，是从摄像头采集的原始画面，经过前处理（美颜、镜像、裁剪等操作）后直接渲染给主播观看，所以清晰度是最高的。而观众端看到的是经过编码器压缩再解码的画面，由于编码本身会降低压缩质量（视频码率设置的越低，压缩程度越严重），所以观众端看到的画面会比主播端清晰度低。

2. 为什么 TXLivePusher 推出来的流会有368 × 640或者544 × 960这样的分辨率?

在开启硬件加速后，您可能会发现诸如368 × 640或者544 × 960此类“不完美”分辨率，这是由于部分硬编码器要求像素能被16整除所致，属于正常现象，您可以通过播放端的填充模式解决“小黑边”问题。

自定义采集和渲染

最近更新时间：2022-10-08 16:33:50

定制推流画面

iOS 平台

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（例如堆加字幕），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。

1. **设置视频处理回调。**
   设置 **TXLivePush** 的 videoProcessDelegate 代理点，即可实现对视频画面的定制。

```
@protocol TXVideoCustomProcessDelegate

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

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

@end
```

2. **在回调函数中对视频数据进行加工。**
   实现 TXVideoCustomProcessDelegate 的 onPreProcessTexture 函数，以实现对视频画面的自定义处理。textureId 指定的纹理是一块类型为 GLES20.GL_TEXTURE_2D 的纹理。
   对于 texture 数据的操作，需要一定的 OpenGL 基础知识，另外计算量不宜太大，因为 onPreProcessTexture 的调用频率跟 FPS 相同，过于繁重的处理很容易造成 GPU 过热。

方案二：自己采集数据

如果您只希望使用 SDK 来编码和推流（例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：

1. **不再调用 TXLivePush 的 startPreview 接口。**
   这样 SDK 本身就不会再采集视频数据和音频数据，而只是启动预处理、编码、流控、发送等跟推流相关的工作。
2. **通过 TXLivePushConfig 设置 customModeType。**

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

```
//如果声音和视频都要自己采集，可以设置 customModeType 为 3
@interface TXLivePushConfig : NSObject
@property(nonatomic, assign) int customModeType;
@end
```

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

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


//可以塞入自己采集和处理的 RGBA 或者 NV12 数据
[pusher sendVideoSampleBuffer:sampleBuffer];
```

4. **使用 sendAudioSampleBuffer 向 SDK 填充 Audio 数据。**
   sendAudioSampleBuffer 的作用是向 SDK 塞入您采集和处理后的音频数据，请使用单声道、16位宽、48000Hz 的 PCM 声音数据。
   该函数支持两种 RPSampleBufferType， RPSampleBufferTypeAudioApp 和 RPSampleBufferTypeAudioMic，前者用于 replaykit，后者用于常规的麦克风采集（也用于 replaykit）。

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


//可以塞入自己采集和处理的 声音数据
[pusher sendAudioSampleBuffer:sampleBuffer withType:RPSampleBufferTypeAudioMic];
```

Android 平台

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（例如堆加字幕），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。

1. **设置视频处理回调。**
   设置 **TXLivePush** 的 setVideoProcessListener 自定义视频回调，即可实现对视频画面的定制。

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

```
int onTextureCustomProcess(int textureId, int width, int height);

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

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

2. **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（例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：

1. **不再调用 TXLivePush 的 startPreview 接口。**
   这样 SDK 本身就不会再采集视频数据和音频数据，而只是启动预处理、编码、流控、发送等跟推流相关的工作。
2. **通过 TXLivePushConfig 设置 customModeType**

```
public static final int CUSTOM_MODE_AUDIO_CAPTURE = 0X001;    //客户自定义音频采集
public static final int CUSTOM_MODE_VIDEO_CAPTURE = 0X002;    //客户自定义视频采集

//如果声音和视频都要自己采集，可以设置 customModeType 为 3
public class TXLivePushConfig {
    public void setCustomModeType(int modeType) {}
}
```

3. **使用 sendCustomVideoTexture 或者 sendCustomVideoData 向 SDK 填充 Video 数据。**
   sendCustomVideoTexture 的作用是向 SDK 塞入您采集和处理后的 texture 视频数据，目前仅支持普通纹理，暂不支持外部纹理。
   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 YUV_420P 和 RGB_RGBA 两种格式。

```
TXLivePushConfig config = new TXLivePushConfig();
config.setCustomModeType(CUSTOM_MODE_VIDEO_CAPTURE); // 自定义采集数据
config.setVideoResolution(VIDEO_RESOLUTION_TYPE_640_360); // 设置为您发送 YUV 数据的 width、
height
TXLivePusher pusher = new TXLivePusher(context);
pusher.setConfig(config);

//可以塞入自己采集和处理的 texture 数据
pusher.sendCustomVideoTexture(textureID, width, height);

//可以塞入自己采集和处理的 RGBA 或者 NV12 数据
```

```
pusher.sendCustomVideoData(buffer, bufferType, width, height);
```

4. **使用 sendCustomPCMData 向 SDK 填充 Audio 数据。**
   sendCustomPCMData 的作用是向 SDK 发送自己采集的音频 PCM 数据。
   SDK 对每次传入的 PCM buffer 大小有严格要求，每一个采样点要求是16位宽。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。

```
TXLivePushConfig config = new TXLivePushConfig();
config.setCustomModeType(CUSTOM_MODE_AUDIO_CAPTURE); // 自定义采集数据
config.setAudioSampleRate(48000); // 采样率：48000Hz
config.setAudioChannels(1);       // 默认值：1（单声道）
TXLivePusher pusher = new TXLivePusher(context);
pusher.setConfig(config);

//可以塞入自己采集和处理的 声音数据
pusher.sendCustomPCMData(pcmBuffer);
```

定制播放数据

iOS 平台

1. **设置 TXLivePlayer 的 TXVideoCustomProcessDelegate 属性。**

```
@interface TXLivePlayer : NSObject
// 设置之后，Player 的每帧画面都会经过 onPlayerPixelBuffer
@property(nonatomic, weak) id videoProcessDelegate;
```

2. **通过 onPlayerPixelBuffer 回调捕获 Player 的图像数据。**
   如果当前 Player 是硬解码模式，pixelBuffer 的图像格式是 **NV12**，如果当前 Player 是软解码模式，pixelBuffer 的图像格式是 **i420** 。
   onPlayerPixelBuffer 返回值为 YES 时， SDK 不会继续做图像渲染，这种方式可以解决 OpenGL 线程冲突的问题。

```
@protocol TXVideoCustomProcessDelegate
@optional

/**
* 视频渲染对象回调
* @prarm pixelBuffer 渲染图像
* @return 返回YES则SDK不再显示；返回NO则SDK渲染模块继续渲染
* 说明：渲染图像的数据类型为config中设置的renderPixelFormatType
*/
-(BOOL)onPlayerPixelBuffer:(CVPixelBufferRef)pixelBuffer;
@end
```

Android 平台

1. **设置 TXLivePlayer 的 ITXLivePlayVideoRenderListener 接口。**

```
public class TXLivePlayer {
    /**
     * 设置视频渲染纹理回调
     *
     * @param listener 自定义渲染纹理回调
```

```
     * @param glContext OpenGL 上下文，如果您的渲染模块使用 (javax.microedition.khronos.egl.*) 定义
的 OpenGL 接口，请填 javax.microedition.khronos.egl.EGLContext 类型的对象
     *                  如果您的渲染模块使用 (android.opengl.*) 定义的 OpenGL 接口，请填
android.opengl.EGLContext 类型的对象。
     *                  如果填 null 表示使用 SDK 内部默认的
javax.microedition.khronos.egl.EGLContext 环境
     * @return 0：成功；-1：glContext 类型非法。
     */
    public int setVideoRenderListener(final ITXLivePlayVideoRenderListener listener, Object
glContext) {}
}
```

2. **通过 onRenderVideoFrame 回调捕获 Player 的图像数据。**

```
static public class TXLiteAVTexture
{
    public int      textureId;      //  视频纹理 ID
    public int      width;          //  视频宽度
    public int      height;         //  视频高度
    public Object   eglContext;     //  javax.microedition.khronos.egl.EGLContext 或
android.opengl.EGLContext 定义的 OpenGL Context
}

public interface ITXLivePlayVideoRenderListener {
    /**
        * 解码器解出一帧数据回调一次
        *
        * @param texture 视频纹理
        */
    void onRenderVideoFrame(TXLiteAVTexture texture);
}
```

SDK 指标监控

最近更新时间：2023-06-16 11:33:24

简介

SDK 的各项监控指标可以从 TXLivePushListener 和 TXLivePlayListener 的回调中获取。

TXLivePushListener

1. 如何获取推流的状态数据?

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

TXLivePushListener - onNetStatus

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

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

NET_STATUS_VIDEO_DROP	视频全局丢帧次数，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帧，丢帧次数越多，说明网络问题越严重。
NET_STATUS_AUDIO_DROP	音频全局丢包次数，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。
NET_STATUS_VIDEO_GOP	当前视频 GOP，也就是每两个关键帧(I帧)间隔时长，单位 s。

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 。
- **NET_STATUS_VIDEO_CACHE、NET_STATUS_AUDIO_CACHE、NET_STATUS_VIDEO_DROP、NET_STATUS_AUDIO_DROP**
  如果主播当前网络的上传速度不佳，就很容易出现 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。如果积压的数据超过警戒阈值，SDK 会主动丢弃一些音视频数据。
  - NET_STATUS_VIDEO_CACHE 表示视频积压的帧数。
  - NET_STATUS_AUDIO_CACHE 表示音频积压的包数。
  - NET_STATUS_VIDEO_DROP 表示累计丢弃的视频帧数。
  - NET_STATUS_AUDIO_DROP 表示累计丢弃的音频包数。

- **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。

> **说明**

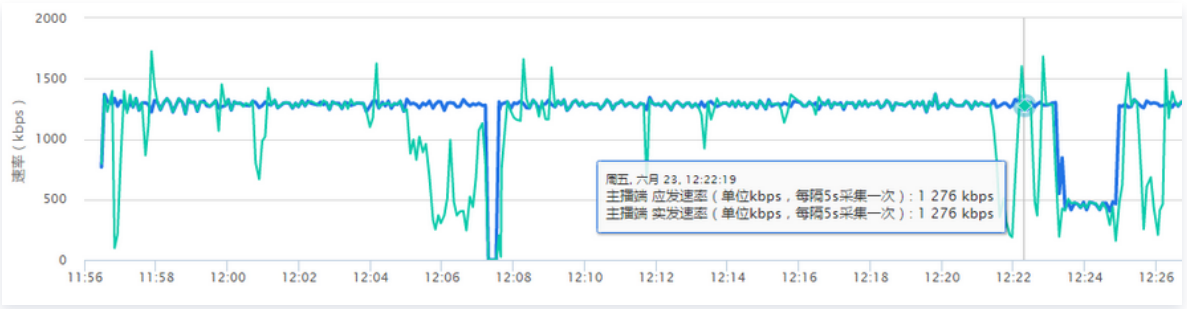
> 很多客户会遇到的一个问题： App 在线下测试时性能表现极佳，但在 App 外发上线后，前排房间里的互动消息的滚屏和刷新会产生极大的 CPU 消耗导致直播画面卡顿严重。

- **SERVER_IP**
  如果主播到 SERVER_IP 给出的 IP 地址的 ping 值很高（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。

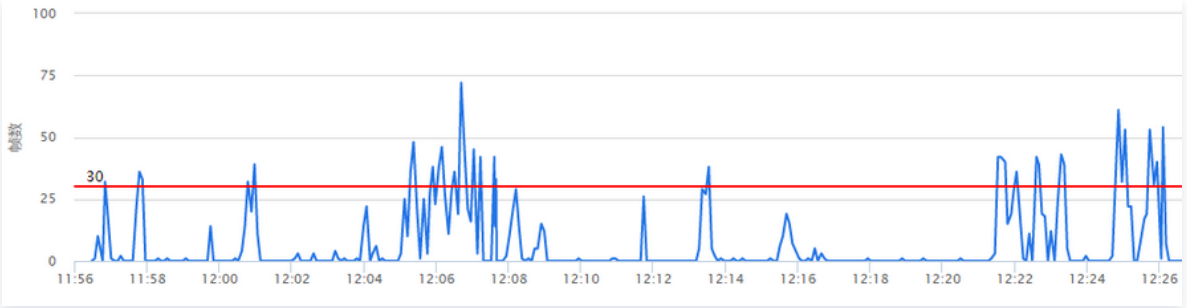
3. 如何看懂腾讯云推流图表?

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：

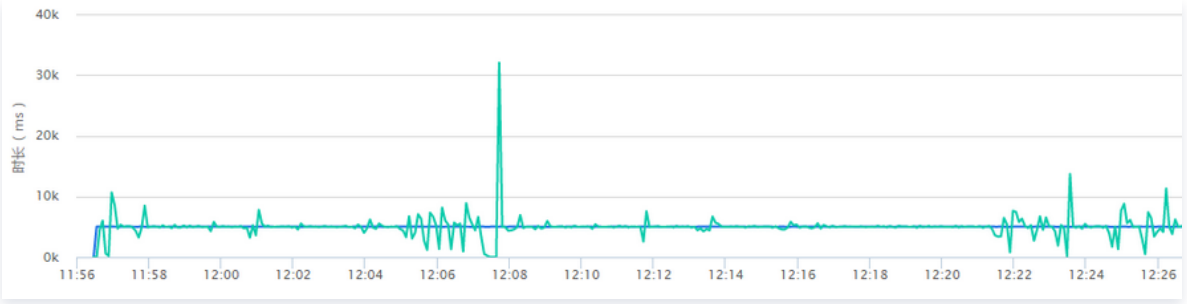
- **主播端–应发速率–实发速率曲线图**
  蓝色曲线代表 BITRATE 的统计曲线，即 SDK 产生的音视频数据，绿色曲线代表实际网络发出去多少。两条线重合度越高表示推流质量越好。



- **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。
  - 如果出现大于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秒 – 2秒会将 SDK 内部的状态指标同步出来，其中如下指标比较有意义：

TXLivePlayListener - onNetStatus

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

播放状态	含义说明
NET_STATUS_CPU_USAGE	当前瞬时 CPU 使用率。
NET_STATUS_VIDEO_WIDTH	视频分辨率 – 宽。
NET_STATUS_VIDEO_HEIGHT	视频分辨率 – 高。
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_VIDEO_CACHE	视频播放缓冲区（jitterbuffer）大小，缓冲区越小越难以抵抗卡顿。
NET_STATUS_AUDIO_CACHE	音频播放缓冲区（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_VIDEO_CACHE 、NET_STATUS_AUDIO_CACHE**
  这两个指标用于衡量播放缓冲区（jitterbuffer）的大小：
- 缓冲区越大，播放的延迟越高，但网络有波动时越不容易卡顿。
- 缓冲区越小，播放的延迟越低，但下载速度稍有波动就可能引发卡顿。

TXLivePlayer 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 – 直播拉流 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，延迟越低；网络越差，延迟越高。

API 文档
iOS
概览

最近更新时间：2025-03-27 15:54:02

TXLivePlayer

视频播放器

请参见 TXLivePlayer 。
主要负责将直播流的音视频画面进行解码和本地渲染，包含如下技术特点：

- 针对腾讯云的拉流地址，可使用低延时拉流，实现直播连麦等相关场景。
- 针对腾讯云的拉流地址，可使用直播时移功能，能够实现直播观看与时移观看的无缝切换。
- 支持自定义的音视频数据处理，让您可以根据项目需要处理直播流中的音视频数据后，进行渲染以及播放。

SDK 基础函数

API	描述
delegate	设置播放回调，见 `TXLivePlayListener.h` 文件中的详细定义。
videoProcessDelegate	设置视频处理回调，见 `TXVideoCustomProcessDelegate.h` 文件中的详细定义。
audioRawDataDelegate	设置音频处理回调，见 `TXAudioRawDataDelegate.h` 文件中的详细定义。
enableHWAcceleration	是否开启硬件加速，默认值：NO。
config	设置 TXLivePlayConfig 播放配置项，见 `TXLivePlayConfig.h` 文件中的详细定义。
recordDelegate	设置短视频录制回调，见 `TXLiveRecordListener.h` 文件中的详细定义。
isAutoPlay	startLivePlay 后是否立即播放，默认 YES，只有点播有效。

播放基础接口

API	描述
setupVideoWidget	创建 Video 渲染 View，该控件承载着视频内容的展示。
removeVideoWidget	移除 Video 渲染 Widget。
startLivePlay	启动从指定 URL 播放 RTMP 音视频流。10.7 版本开始，`startPlay` 变更为 `startLivePlay` ，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用，若您暂未获取上述 Licence，可 购买 Licence 。
stopPlay	停止播放音视频流。
isPlaying	是否正在播放。
pause	暂停播放。
resume	继续播放，适用于点播，直播。

视频相关接口

API	描述
setRenderRotation	设置画面的方向。
setRenderMode	设置画面的裁剪模式。
snapshot	截屏。

音频相关接口

API	描述
setMute	设置静音。
setVolume	设置音量。
setAudioRoute	设置声音播放模式（切换扬声器，听筒）。
setAudioVolumeEvaluationListener	设置音量大小回调接口。

直播时移相关接口

API	描述
prepareLiveSeek	直播时移准备，拉取该直播流的起始播放时间。
resumeLive	停止时移播放，返回直播。
seek	-

视频录制相关接口

API	描述
startRecord	开始录制短视频。
stopRecord	结束录制短视频。
setRate	设置播放速率。

更多实用接口

API	描述
setLogViewMargin	设置状态浮层 view 在渲染 view 上的边距。
showVideoDebugLog	是否显示播放状态统计及事件消息浮层 view。
switchStream	FLV 直播无缝切换。
callExperimentalAPI	调用实验性 API 接口。

枚举值

枚举	描述
TX_Enum_PlayType	支持的直播和点播类型。

TXLivePlayConfig

腾讯云直播播放器的参数配置模块

请参见 TXLivePlayConfig 。
主要负责 TXLivePlayer 对应的参数设置，其中绝大多数设置项在播放开始之后再设置是无效的。

TXLivePlayListener

腾讯云直播播放的回调通知

请参见 TXLivePlayListener 。

API	描述
onPlayEvent	直播事件通知。
onNetStatus	网络状态通知。

TXLivePush

直播推流类

请参见 TXLivePush 。
主要负责将本地的音视频画面进行编码和 RTMP 推送，包含如下技术特点：

- 针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。
- 内嵌套的 Qos 流量控制技术具备上行网络自适应能力，可以根据主播端网络的具体情况实时调节音视频数据量。
- 内嵌多套美颜磨皮算法（自然和光滑）和多款色彩空间滤镜（支持自定义滤镜），可以根据需要自行选择。
- 商业版包含了基于优图 AI 识别技术的大眼、瘦脸、隆鼻以及动效挂架，只需要购买**优图 License** 就可以零成本集成。
- 支持自定义的音视频采集和渲染，让您可以根据项目需要选择自己的音视频数据源。

SDK 基础函数

API	描述
config	设置 TXLivePushConfig 推流配置项，见 `TXLivePushConfig.h` 文件中的详细定义。
delegate	设置推流回调接口，见 `TXLivePushListener.h` 文件中的详细定义。
initWithConfig	创建 TXLivePusher 示例。

推流基础接口

API	描述
rtmpURL	获取当前推流的 RTMP 地址。
startPreview	启动摄像头预览。
stopPreview	停止摄像头预览。
startPush	启动 RTMP 推流。
stopPush	停止 RTMP 推流。
pausePush	暂停摄像头采集并进入垫片推流状态。
resumePush	恢复摄像头采集并结束垫片推流状态。
isPublishing	查询是否正在推流。

视频相关接口

API	描述
frontCamera	查询当前是否为前置摄像头。
setVideoQuality	设置视频编码质量。
switchCamera	切换前后摄像头（iOS）。
selectCamera	选择摄像头（macOS）。
setMirror	设置视频镜像效果。
setRenderRotation	设置本地摄像头预览画面的旋转方向。
toggleTorch	打开后置摄像头旁边的闪光灯。
setZoom	调整摄像头的焦距。
setFocusPosition	设置手动对焦区域。

美颜相关接口

API	描述
getBeautyManager	获取美颜管理对象 TXBeautyManager，美颜的设置通过 TXBeautyManager 来设置。

音频相关接口

API	描述
setMute	开启静音。
playBGM	播放背景音乐。
playBGM	播放背景音乐（高级版本）。
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	获取背景音乐文件的总时长，单位：毫秒。
setBGMVolume	设置混音时背景音乐的音量大小，仅在播放背景音乐混音时使用。
setMicVolume	设置混音时麦克风音量大小，仅在播放背景音乐混音时使用。
setBgmPitch	调整背景音乐的音调高低。
setReverbType	设置混响效果。
setVoiceChangerType	设置变声类型。

本地录制接口

API	描述
recordDelegate	录制回调接口，详见 `TXLiveRecordTypeDef.h` 中的 TXLiveRecordListener 定义。

startRecord	开始录制短视频。
stopRecord	-
snapshot	推流过程中本地截图。

自定义采集和处理

API	描述
videoProcessDelegate	自定义视频处理回调。
audioProcessDelegate	自定义音频处理回调。
sendVideoSampleBuffer	自定义视频采集，向 SDK 发送自己采集的视频数据。
sendCustomPCMData	自定义音频采集，向 SDK 发送自己采集的音频 PCM 数据。
sendAudioSampleBuffer	自定义音频采集，向 SDK 发送自己采集的音频数据。
setSendAudioSampleBufferMuted	要求 SDK 发送静音数据。

更多实用接口

API	描述
sendMessageEx	发送 SEI 消息，播放端（TXLivePlayer）通过 onPlayEvent（EVT_PLAY_GET_MESSAGE）来接收该消息。
sendMessage	-
showVideoDebugLog	打开包含视频状态信息的调试浮层，该浮层一般用于 SDK 调试期间，外发版本请不要打开。
setLogViewMargin	设置调试浮层在视频 view 上的位置。
setEnableClockOverlay	设置推流是否覆盖时钟。
enableClockOverlay	获取当前推流画面是否有覆盖时钟。

TXLivePushConfig

腾讯云直播推流用 RTMP SDK 的参数配置模块

请参见 TXLivePushConfig 。
主要负责 TXLivePusher 对应的参数设置，其中绝大多数设置项在推流开始之后再设置是无效的。

TXLivePushListener

腾讯云直播推流的回调通知

请参见 TXLivePushListener 。

API	描述
onPushEvent	事件通知。
onNetStatus	状态通知。

推流
TXLivePush

最近更新时间：2024-11-21 10:44:02

功能

直播推流类。

介绍

主要负责将本地的音视频画面进行编码和 RTMP 推送，包含如下技术特点：

- 针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。
- 内嵌套的 Qos 流量控制技术具备上行网络自适应能力，可以根据主播端网络的具体情况实时调节音视频数据量。
- 内嵌多套美颜磨皮算法（自然&光滑）和多款色彩空间滤镜（支持自定义滤镜），可以根据需要自行选择。
- 企业版 SDK 包含了大眼、瘦脸以及隆鼻等功能，配合高级美颜动效素材，可快速地完成功能集成。
- 支持自定义的音视频采集和渲染，让您可以根据项目需要选择自己的音视频数据源。

SDK 基础函数

config

设置 TXLivePushConfig 推流配置项，见 `TXLivePushConfig.h` 文件中的详细定义。

```
@property (nonatomic, copy) TXLivePushConfig * config
```

delegate

设置推流回调接口，见 `TXLivePushListener.h` 文件中的详细定义。

```
@property (nonatomic, weak) id< TXLivePushListener > delegate
```

initWithConfig

创建 TXLivePusher 示例。

```
- (id)initWithConfig:(TXLivePushConfig *)config
```

参数

参数	类型	含义
config	TXLivePushConfig *	TXLivePushConfig 推流配置项，见 `TXLivePushConfig.h` 文件中的详细定义。

推流基础接口

rtmpURL

获取当前推流的 RTMP 地址。

```
@property (nonatomic, readonly, assign) NSString * rtmpURL
```

startPreview

启动摄像头预览。

```
- (int)startPreview:(TXView *)view
```

参数

参数	类型	含义
view	TXView *	承载视频画面的控件。

介绍

启动预览后并不会立刻开始 RTMP 推流，需要调用 startPush 才能真正开始推流。

stopPreview

停止摄像头预览。

```
- (void)stopPreview
```

startPush

启动 RTMP 推流。

```
- (int)startPush:(NSString *)rtmpURL
```

参数

参数	类型	含义
rtmpURL	NSString *	推流地址，请参见 获取推流地址 。

返回

0：启动成功；-1：启动失败；-5：license 校验失败。

介绍

针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。

> **说明**
> -5返回码代表 license 校验失败，TXLivePusher 需要 License 校验通过才能工作。

stopPush

停止 RTMP 推流。

```
- (void)stopPush
```

pausePush

暂停摄像头采集并进入垫片推流状态。

```
- (void)pausePush
```

介绍

SDK 会暂时停止摄像头采集，并使用 TXLivePushConfig.pauseImg 中指定的图片作为替代图像进行推流，也就是所谓的“垫片”。这项功能常用于 App 被切到后台运行的场景，尤其是在 iOS 系统中，当 App 切到后台以后，操作系统不会再允许该 App 继续使用摄像头。此时就可以通过调用 pausePush 进入垫片状态。
对于绝大多数推流服务器而言，如果超过一定时间不推视频数据，服务器会断开当前的推流链接。
在 TXLivePushConfig 您可以指定：

- pauseImg 设置后台推流的默认图片，默认为黑色背景。
- pauseFps 设置后台推流帧率，最小值为5，最大值为20，默认为10。
- pauseTime 设置后台推流持续时长，单位秒，默认300秒。

> **说明**
> 请注意调用顺序：startPush => (pausePush => resumePush) => stopPush，错误的调用顺序会导致 SDK 表现异常。

resumePush

恢复摄像头采集并结束垫片推流状态。

```
- (void)resumePush
```

isPublishing

查询是否正在推流。

```
- (bool)isPublishing
```

返回
YES：推流中；NO：没有在推流。

视频相关接口

frontCamera

查询当前是否为前置摄像头。

```
@property (nonatomic, readonly, assign) BOOL frontCamera
```

setVideoQuality

设置视频编码质量。

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

参数

参数	类型	含义
quality	TX_Enum_Type_VideoQuality	画质类型（标清，高清，超高清）。
adjustBitrate	BOOL	动态码率开关。
adjustResolution	BOOL	动态切分辨率开关。

介绍
推荐设置：秀场直播 quality：HIGH_DEFINITION；adjustBitrate：NO；adjustResolution：NO。请参见 设定清晰度 。

> **说明**
> adjustResolution 早期被引入是为了让 TXLivePusher 能够满足视频通话这一封闭场景下的一些需求，现已不推荐使用。 如果您有视频通话的需求，可以使用我们专门为视频通话打造的 TRTC 服务。 由于目前很多 H5 播放器不支持分辨率动态变化，所以开启分辨率自适应以后，会导致 H5 播放端和录制文件的很多兼容问题。

switchCamera

切换前后摄像头（iOS）。

```
- (int)switchCamera
```

selectCamera

选择摄像头（macOS）。

```
- (void)selectCamera:(AVCaptureDevice *)camera
```

setMirror

设置视频镜像效果。

```
- (void)setMirror:(BOOL)isMirror
```

参数

参数	类型	含义
isMirror	BOOL	YES：播放端看到的是镜像画面；NO：播放端看到的是非镜像画面。

介绍

由于前置摄像头采集的画面是取自手机的观察视角，如果将采集到的画面直接展示给观众，是完全没有问题的。 但如果将采集到的画面也直接显示给主播，则会跟主播照镜子时的体验完全相反，会让主播感觉到很奇怪。 因此，SDK 会默认开启本地摄像头预览画面的镜像效果，让主播直播时跟照镜子时保持一个体验效果。
setMirror 所影响的则是观众端看到的视频效果，如果想要保持观众端看到的效果跟主播端保持一致，需要开启镜像； 如果想要让观众端看到正常的未经处理过的画面（如主播弹吉他的时候有类似需求），则可以关闭镜像。

setRenderRotation

设置本地摄像头预览画面的旋转方向。

```
- (void)setRenderRotation:(int)rotation
```

参数

参数	类型	含义
rotation	int	取值为0、90、180和270（其他值无效），表示主播端摄像头预览视频的顺时针旋转角度。

介绍

该接口仅能够改变主播本地预览画面的方向，而不会改变观众端的画面效果。 如果希望改变观众端看到的视频画面的方向，例如原来是540 × 960，希望变成960 × 540，则可以通过设置 TXLivePushConfig 中的 homeOrientation 来实现。

```
// 竖屏推流（HOME 键在下）
```

```
_config.homeOrientation = HOME_ORIENTATION_DOWN;
[_txLivePublisher setConfig:_config];
[_txLivePublisher setRenderRotation:0];
// 横屏推流（HOME 键在右）
_config.homeOrientation = HOME_ORIENTATION_RIGHT;
[_txLivePublisher setConfig:_config];
[_txLivePublisher setRenderRotation:90];
```

toggleTorch

打开后置摄像头旁边的闪光灯。

```
- (BOOL)toggleTorch:(BOOL)bEnable
```

参数

参数	类型	含义
bEnable	BOOL	YES：打开；NO：关闭。

返回

YES：打开成功；NO：打开失败。

介绍

此操作对于前置摄像头是无效的，因为绝大多数手机都没有给前置摄像头配置闪光灯。

setZoom

调整摄像头的焦距。

```
- (void)setZoom:(CGFloat)distance
```

参数

参数	类型	含义
distance	CGFloat	焦距大小，取值范围：1 – 5，默认值建议设置为1即可。

> **说明**
> 当 distance 为1的时候为最远视角（正常镜头），当为5的时候为最近视角（放大镜头），最大值不要超过5，超过5后画面会模糊不清。

setFocusPosition

设置手动对焦区域。

```
- (void)setFocusPosition:(CGPoint)touchPoint
```

介绍

SDK 默认使用摄像头自动对焦功能，您也可以通过 TXLivePushConfig 中的 touchFocus 选项关闭自动对焦，改用手动对焦。改用手动对焦之后，需要由主播自己单击摄像头预览画面上的某个区域，来手动指导摄像头对焦。

> **说明**
> 早期 SDK 版本仅仅提供了手动和自动对焦的选择开关，并不支持设置对焦位置，3.0版本以后，手动对焦的接口才开放出来。

美颜相关接口

getBeautyManager

获取美颜管理对象 TXBeautyManager 。

```
- (TXBeautyManager *)getBeautyManager
```

通过美颜管理，您可以使用以下功能：

- 设置“美颜风格”、“美白”、“红润”、“大眼”、“瘦脸”、“V脸”、“下巴”、“短脸”、“小鼻”、“亮眼”、“白牙”、“祛眼袋”、“祛皱纹”、“祛法令纹”等美容效果。
- 调整“发际线”、“眼间距”、“眼角”、“嘴形”、“鼻翼”、“鼻子位置”、“嘴唇厚度”、“脸型”。
- 设置人脸挂件（素材）等动态效果。
- 添加美妆。
- 进行手势识别。

音频相关接口

setMute

开启静音。

```
- (void)setMute:(BOOL)bEnable
```

参数

参数	类型	含义
bEnable	BOOL	是否开启静音。

介绍

开启静音后，SDK 并不会继续采集麦克风的声音，但是会用非常低（5kbps左右）的码率推送伪静音数据， 这样做的目的是为了兼容 H5 上的 video 标签，并让录制出来的 MP4 文件有更好的兼容性。

playBGM

播放背景音乐。

```
- (BOOL)playBGM:(NSString *)path
```

参数

参数	类型	含义
path	NSString *	本地音乐文件路径。

返回

YES：成功；NO：失败。

介绍

SDK 会将背景音乐和麦克风采集的声音进行混合并一起推送到云端。

playBGM

播放背景音乐（高级版本）。

```
- (BOOL)playBGM:(NSString *)path withBeginNotify:(void(^)(NSInteger errCode))beginNotify
withProgressNotify:(void(^)(NSInteger progressMS, NSInteger durationMS))progressNotify
andCompleteNotify:(void(^)(NSInteger errCode))completeNotify
```

参数

参数	类型	含义
path	NSString *	本地音乐文件路径。
beginNotify	void(^)(NSInteger errCode)	播放开始的回调。
progressNotify	void(^)(NSInteger progressMS, NSInteger durationMS)	播放进度回调。
completeNotify	void(^)(NSInteger errCode)	播放完毕回调。

返回

YES：成功；NO：失败。

stopBGM

停止播放背景音乐。

```
- (BOOL)stopBGM
```

pauseBGM

暂停播放背景音乐。

```
- (BOOL)pauseBGM
```

resumeBGM

继续播放背景音乐。

```
- (BOOL)resumeBGM
```

getMusicDuration

获取背景音乐文件的总时长，单位是毫秒。

```
- (int)getMusicDuration:(NSString *)path
```

参数

参数	类型	含义
path	NSString *	音乐文件路径，如果 path 为空，那么返回当前正在播放的背景音乐的时长。

setBGMVolume

设置混音时背景音乐的音量大小，仅在播放背景音乐混音时使用。

```
- (BOOL)setBGMVolume:(float)volume
```

参数

参数	类型	含义
volume	float	音量大小，1为正常音量，范围是0 – 1之间的浮点数。

返回

YES：成功；NO：失败。

setMicVolume

设置混音时麦克风音量大小，仅在播放背景音乐混音时使用。

```
- (BOOL)setMicVolume:(float)volume
```

参数

参数	类型	含义
volume	float	音量大小，1为正常音量，范围是0 – 1之间的浮点数。

返回

YES：成功；NO：失败。

setBgmPitch

调整背景音乐的音调高低。

```
- (BOOL)setBgmPitch:(float)pitch
```

参数

参数	类型	含义
pitch	float	音调，默认值是0.0f，范围是-1 – 1之间的浮点数。

返回

YES：成功；NO：失败。

setReverbType

设置混响效果。

```
- (BOOL)setReverbType:(TXReverbType)reverbType
```

参数

参数	类型	含义
reverbType	TXReverbType	混响类型，详见 `TXLiveSDKTypeDef.h` 中的 TXReverbType 定义。

返回

YES：成功；NO：失败。

setVoiceChangerType

设置变声类型。

```
- (BOOL)setVoiceChangerType:(TXVoiceChangerType)voiceChangerType
```

参数

参数	类型	含义
voiceChangerType	TXVoiceChangerType	混响类型，详见 `TXLiveSDKTypeDef.h` 中的 voiceChangerType 定义。

返回

YES：成功；NO：失败。

本地录制接口

recordDelegate

录制回调接口，详见 `TXLiveRecordTypeDef.h` 中的 TXLiveRecordListener 定义。

```
@property (nonatomic, weak) id< TXLiveRecordListener > recordDelegate
```

startRecord

开始录制短视频。

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

参数

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

返回

0：成功；-1：videoPath 为空；-2：上次录制尚未结束，请先调用 stopRecord；-3：推流尚未开始。

> **说明**
>
> - 只有启动推流后才能开始录制，非推流状态下启动录制无效。
> - 出于安装包体积的考虑，仅专业版和商业版两个版本的 LiteAVSDK 支持该功能，直播精简版仅定义了接口但并未实现。
> - 录制过程中请勿动态切换分辨率和软/硬剪辑，会有很大概率导致生成的视频异常。

stopRecord

```
- (int)stopRecord
```

返回

0：成功；-1：不存在录制任务。

snapshot

推流过程中本地截图。

```
- (void)snapshot:(void(^)(TXImage *))snapshotCompletionBlock
```

参数

参数	类型	含义
snapshotCompletionBlock	void(^)(TXImage *)	截图完成的回调函数。

自定义采集和处理

videoProcessDelegate

自定义视频处理回调。

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

介绍

自定义视频采集和自定义视频处理不能同时开启，与自定义视频采集不同，自定义视频处理依然是由 SDK 采集摄像头的画面， 但 SDK 会通过 TXVideoCustomProcessDelegate（见 `TXVideoCustomProcessDelegate.h` ）回调将数据回调给您的 App 进行二次加工。
如果要开启自定义视频处理，需要给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_VIDEO_PREPROCESS 选项。

> **说明**
> 出于性能和稳定性考虑，一般不建议开启此特性。

audioProcessDelegate

自定义音频处理回调。

```
@property (nonatomic, weak) id< TXAudioCustomProcessDelegate > audioProcessDelegate
```

介绍

自定义音频采集和自定义音频处理不能同时开启，与自定义音频采集不同，自定义音频处理依然是由 SDK 采集麦克风的声音， 但 SDK 会通过 TXAudioCustomProcessDelegate（见 `TXAudioCustomProcessDelegate.h` ）回调将数据回调给您的 App 进行二次加工。
如果要开启自定义音频处理，需要给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_AUDIO_PREPROCESS 选项。

> **说明**
> 出于性能和稳定性考虑，一般不建议开启此特性。

sendVideoSampleBuffer

自定义视频采集，向 SDK 发送自己采集的视频数据。

```
- (void)sendVideoSampleBuffer:(CMSampleBufferRef)sampleBuffer
```

参数

参数	类型	含义
sampleBuffer	CMSampleBufferRef	向 SDK 发送的 SampleBuffer。

介绍

在自定义视频采集模式下，SDK 不再继续从摄像头采集图像，只保留编码和发送能力，您需要定时地发送自己采集的 SampleBuffer。要开启自定义视频采集，需要完成如下两个步骤：

1. 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_VIDEO_CAPTURE 选项，代表开启自定义视频采集。

2. 设定视频分辨率：将 TXLivePushConfig 中的 sampleBufferSize 属性设置为您期望的分辨率。如果期望编码分辨率跟采集分辨率一致，可以不设置 sampleBufferSize 属性，而是将 autoSampleBufferSize 设置为 YES。

> **说明**
> 1. 开启自定义视频采集后，即无需再调用 startPreview 来开启摄像头采集。
> 2. SDK 内部有简单的帧率控制，如果发送太快时 SDK 会自动丢弃多余的帧率；如果超时不发送，SDK 会不断地重复发送最后一帧。

sendCustomPCMData

自定义音频采集，向 SDK 发送自己采集的音频 PCM 数据。

```
- (void)sendCustomPCMData:(unsigned char *)data len:(unsigned int)len
```

参数

参数	类型	含义
data	unsigned char *	要发送的 PCM buffer。
len	unsigned int	数据长度。

介绍

在自定义音频采集模式下，SDK 不再继续从麦克风采集声音，只保留编码和发送能力，您需要定时地发送自己采集的声音数据（PCM 格式）要开启自定义音频采集，需要完成如下两个步骤：

1. 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_AUDIO_CAPTURE 选项，代表开启自定义音频采集。
2. 设定音频采样率：将 TXLivePushConfig 中的 audioSampleRate 属性设置为您期望的音频采样率，audioChannels 设置为期望的声道数，默认值：1（单声道）。

> **说明**
> SDK 对每次传入的 PCM buffer 大小有严格要求，每一个采样点要求是16位宽。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。

sendAudioSampleBuffer

自定义音频采集，向 SDK 发送自己采集的音频数据。

```
- (void)sendAudioSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:
(RPSampleBufferType)sampleBufferType
```

参数

参数	类型	含义
sampleBuffer	CMSampleBufferRef	采集到的声音 sampleBuffer。
sampleBufferType	RPSampleBufferType	• RPSampleBufferTypeAudioApp：ReplayKit 采集到的 App 声音。 • RPSampleBufferTypeAudioMic：ReplayKit 采集到的麦克风声音。

介绍

相比于 sendCustomPCMData，sendAudioSampleBuffer 主要用于 ReplayKit 录屏推流的场景。要开启自定义音频采集，需要完成如下两个步骤：

1. 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_AUDIO_CAPTURE 选项，代表开启自定义音频采集。
2. 设定音频采样率：将 TXLivePushConfig 中的 audioSampleRate 属性设置为您期望的音频采样率，audioChannels 设置为期望的声道数，默认值：1（单声道）。

当使用 ReplayKit 做录屏推流时，iOS 的 ReplayKit 接口会回调两种类型的声音数据：

- RPSampleBufferTypeAudioApp，也就是要录制的 App 的声音数据。
- RPSampleBufferTypeAudioMic，也就是要录制的麦克风的声音数据。

当您通过 sendAudioSampleBuffer 向 SDK 调用各种类型的声音数据时，SDK 内部会进行混流，否则只会发送一路的声音数据。

setSendAudioSampleBufferMuted

要求 SDK 发送静音数据。

```
- (void)setSendAudioSampleBufferMuted:(BOOL)muted
```

参数

参数	类型	含义
muted	BOOL	YES；静音；NO；关闭静音。

介绍

该函数配合 sendAudioSampleBuffer 使用，在 InApp 类型录制切后台场合时需要调用，系统屏幕录制不需要。

更多实用接口

sendMessageEx

发送 SEI 消息，播放端（TXLivePlayer）通过 onPlayEvent（EVT_PLAY_GET_MESSAGE）来接收该消息。

```
- (BOOL)sendMessageEx:(NSData *)data
```

介绍

本接口是将数据直接塞入视频数据头中，因此不能太大（几个字节比较合适），一般常用于塞入自定义时间戳等信息。

> **说明**
> - sendMessage 已经不推荐使用，会导致 H5 播放器产生兼容性问题，请使用 sendMessageEx。
> - 若您使用过 sendMessage，不推荐立刻升级到 sendMessageEx。
> - sendMessageEx 发送消息给旧版本5.0及以前的 SDK 版本时，消息会无法正确解析，但播放不受影响。

sendMessage

```
- (void)sendMessage:(NSData *)data
```

showVideoDebugLog

打开包含视频状态信息的调试浮层，该浮层一般用于 SDK 调试期间，外发版本请不要打开。

```
- (void)showVideoDebugLog:(BOOL)isShow
```

setLogViewMargin

设置调试浮层在视频 view 上的位置。

```
- (void)setLogViewMargin:(TXEdgeInsets)margin
```

setEnableClockOverlay

设置推流是否覆盖时钟。

```
- (void)setEnableClockOverlay:(BOOL)enabled
```

> **说明**
> 需要双方的时间相同才能获取准确的延迟时间。

enableClockOverlay

获取当前推流画面是否有覆盖时钟。

```
- (BOOL)enableClockOverlay
```

TXLivePushListener

最近更新时间：2022-10-08 16:33:50

功能

腾讯云直播推流的回调通知。

onPushEvent

事件通知。

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

参数

参数	类型	含义
EvtID	int	参见头文件 TXLiveSDKEventDef.h
param	NSDictionary *	参见头文件 TXLiveSDKTypeDef.h

onNetStatus

状态通知。

```
- (void)onNetStatus:(NSDictionary *)param
```

参数

参数	类型	含义
param	NSDictionary *	参见头文件 TXLiveSDKTypeDef.h

TXLivePushConfig

最近更新时间：2025-01-22 11:06:02

功能

腾讯云直播推流用 RTMP SDK 的参数配置模块。

介绍

主要负责 TXLivePusher 对应的参数设置，**其中绝大多数设置项在推流开始之后再设置是无效的。**

属性列表

属性	类型	字段含义	推荐取值	特别说明
homeOrientation	int	HOME 键所在方向，用来切换横竖屏推流，默认值：HOME_ORIENTATION_DOWN（竖屏推流）。	-	常用的还有 HOME_ORIENTATION_RIGHT 和 HOME_ORIENTATION_LEFT，也就是横屏推流。改变该字段的设置以后，本地摄像头的预览画面方向也会发生改变，请调用 TXLivePush 的 setRenderRotation 进行矫正。
touchFocus	BOOL	是否允许点击曝光聚焦，默认值：NO。	-	-
enableZoom	BOOL	是否允许双指手势放大预览画面，默认值：NO。	-	-
watermark	TXImage *	水印图片，设为 nil 等同于关闭水印。	-	-
watermarkPos	CGPoint	水印图片位置，水印大小为图片实际大小，待废弃，推荐使用 watermarkNormalization。	-	-
watermarkNormalization	CGRect	水印图片相对于推流分辨率的归一化坐标。	假设推流分辨率为：540 x 960，该字段设置为：（0.1，0.1，0.1，0.0），那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。	watermarkNormalization 的优先级高于 watermarkPos。
localVideoMirrorType	int	本地预览画面的镜像类型，默认值：LocalVideoMirrorType_Auto 即前置摄像头镜像，后置摄像头不镜像。	-	-
pauseTime	int	垫片推流的最大持续时间，单位秒，默认值：300s。	-	调用 TXLivePusher 的 pausePush 接口，会暂停摄像头采集并进入垫片推流状态，如果该状态一直保持， 可能会消耗主播过多的手机流量，本字段用于指定垫

				片推流的最大持续时间，超过后即断开与云服务器的连接。
pauseFps	int	垫片推流时的视频帧率，取值范围3 - 8，默认值：5FPS。	-	-
pauseImg	TXImage *	垫片推流时使用的图片素材，最大尺寸不能超过1920 × 1920。	-	-
videoResolution	int	视频分辨率，默认值：VIDEO_RESOLUTION_TYPE_360_640。	-	推荐直接使用 TXLivePusher 的 setVideoQuality 接口调整画面质量。
videoFPS	int	视频帧率，默认值：15FPS。	-	推荐直接使用 TXLivePusher 的 setVideoQuality 接口调整画面质量。
videoEncodeGop	int	视频编码 GOP，也就是常说的关键帧间隔，单位秒；默认值：3s。	-	推荐直接使用 TXLivePusher 的 setVideoQuality 接口调整画面质量。
videoBitratePIN	int	视频编码的平均码率，默认值：700kbps。	-	推荐直接使用 TXLivePusher 的 setVideoQuality 接口调整画面质量。
enableAutoBitrate	BOOL	码率自适应开关，开启后，SDK 会根据网络情况自动调节视频码率，调节范围在(videoBitrateMin - videoBitrateMax)。	NO	-
autoAdjustStrategy	int	码率自适应算法。	AUTO_ADJUST_BITRATE_STRATEGY_1	-
videoBitrateMax	int	码率自适应 - 最高码率，默认值：1000kbps。	-	-
videoBitrateMin	int	码率自适应 - 最低码率，默认值：400kbps。	不要设置太低的数值，过低的码率会导致运动画面出现大面积马赛克。	-
audioSampleRate	int	音频采样率，采样率越高音质越好，对于有音乐的场景请使用48000的采样率。	AUDIO_SAMPLE_RATE_48000	-
audioChannels	int	音频声道数，默认值：1（单声道）。	-	-
enableAudioPreview	BOOL	是否开启耳返特效。	NO	开启耳返会消耗更多的 CPU，只有在主播带耳机唱歌的时候才有必要开启此功能。
enablePureAudioPush	BOOL	是否为纯音频推流。	NO	如果希望实现纯音频推流的功能，需要在推流前就设置该参数，否则播放端会有兼容性问题。

connectRetryCount	int	推流遭遇网络连接断开时 SDK 默认重试的次数，取值范围1 – 10，默认值：3。	–	–
connectRetryInterval	int	网络重连的时间间隔，单位秒，取值范围3 – 30，默认值：3。	–	–
customModeType	int	自定义采集和自定义处理开关。	–	该字段需要使用与运算符进行级联操作（自定义采集和自定义处理不能同时开启）： • 开启自定义视频采集：_config.customModeType |= CUSTOM_MODE_VIDEO_CAPTURE； • 开启自定义音频采集：_config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE。
sampleBufferSize	CGSize	仅开启自定义采集时有效，用于设置编码分辨率。	–	此值设置需与调用 sendVideoSampleBuffer 时传入的 SampleBuffer 的宽高比一致，否则会引起画面变形。 如果指定 autoSampleBufferSize 为 YES，则不需要设置该字段。
autoSampleBufferSize	BOOL	仅开启自定义采集时有效，YES 代表编码分辨率等于输入的 SampleBuffer 的分辨率，默认值：NO。	–	–
enableNAS	BOOL	是否开启噪声抑制（注意：早期版本引入了拼写错误，考虑到接口兼容一直没有修正，正确拼写应该是 ANS）。	NO：ANS 对于直播环境中由其它设备外放的音乐是不友好的，通过 playBGM 设置的背景音不受影响。	如果直播场景只有主播在说话，ANS 有助于让主播的声音更清楚，但如果主播在吹拉弹唱，ANS 会损伤乐器的声音。
enableAEC	BOOL	是否开启回声抑制。	NO：回声抑制会启用通话模式音量，导致音质变差，非连麦场景下请不要开启。	只有在连麦模式下才需要开启 AEC，如果是普通的直播，将主播的手机和观众的手机放在一起所产生的啸叫是正常现象。
enableHWAcceleration	BOOL	开启视频硬件加速， 默认值：YES。	–	–
enableAudioAcceleration	BOOL	开启音频硬件加速， 默认值：YES。	–	–
enableAGC	BOOL	开启音频自动增益， 默认值：NO。	–	–
volumeType	TXSystemAudioVolumeType	系统音量类型， 默认值：SYSTEM_VOLUME_TYPE_AUTO。	–	–

frontCamera	BOOL	是否前置摄像头，待废弃，建议直接使用 TXLivePusher 的 frontCamera 属性和 switchCamera 函数。	-	-
beautyFilterDepth	float	美颜强度，待废弃，建议直接使用 TXLivePusher 的 setBeautyStyle 函数。	-	-
whiteningFilterDepth	float	美白强度，待废弃，建议直接使用 TXLivePusher 的 setBeautyStyle 函数。	-	-
enableNearestIP	BOOL	是否开启就近选路，待废弃，默认值：YES。	-	-
rtmpChannelType	int	RTMP 传输通道的类型，待废弃，默认值为：AUTO。	-	-

拉流
TXLivePlayer

最近更新时间：2024-12-10 16:08:53

功能

视频播放器。

介绍

主要负责将直播流的音视频画面进行解码和本地渲染，包含如下技术特点：

- 针对腾讯云的拉流地址，可使用低延时拉流，实现直播连麦等相关场景。
- 针对腾讯云的拉流地址，可使用直播时移功能，能够实现直播观看与时移观看的无缝切换。
- 支持自定义的音视频数据处理，让您可以根据项目需要处理直播流中的音视频数据后，进行渲染以及播放。

SDK 基础函数

delegate

设置播放回调，见 `TXLivePlayListener.h` 文件中的详细定义。

```
@property (nonatomic, weak) id< TXLivePlayListener > delegate
```

videoProcessDelegate

设置视频处理回调，见 `TXVideoCustomProcessDelegate.h` 文件中的详细定义。

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

audioRawDataDelegate

设置音频处理回调，见 `TXAudioRawDataDelegate.h` 文件中的详细定义。

```
@property (nonatomic, weak) id< TXAudioRawDataDelegate > audioRawDataDelegate
```

enableHWAcceleration

是否开启硬件加速，默认值：NO。

```
@property (nonatomic, assign) BOOL enableHWAcceleration
```

config

设置 TXLivePlayConfig 播放配置项，见 `TXLivePlayConfig.h` 文件中的详细定义。

```
@property (nonatomic, copy) TXLivePlayConfig * config
```

recordDelegate

设置短视频录制回调，见 `TXLiveRecordListener.h` 文件中的详细定义。

```
@property (nonatomic, weak) id< TXLiveRecordListener > recordDelegate
```

播放基础接口

setupVideoWidget

创建 Video 渲染 View，该控件承载着视频内容的展示。

```
- (void)setupVideoWidget:(CGRect)frame containView:(TXView *)view insertIndex:(unsigned int)idx
```

参数

参数	类型	含义
frame	CGRect	Widget 在父 view 中的 frame。
view	TXView *	父 view。
idx	unsigned int	Widget 在父 view 上的层级位置。

介绍

变更历史：1.5.2版本将参数 frame 废弃，设置此参数无效，控件大小与参数 view 的大小保持一致，如需修改控件的大小及位置，请调整父 view 的大小及位置。 请参见 绑定渲染界面 。

removeVideoWidget

移除 Video 渲染 Widget。

```
- (void)removeVideoWidget
```

startLivePlay

启动从指定 URL 播放 RTMP 音视频流。

```
- (int)startLivePlay:(NSString *)url type:(TX_Enum_PlayType)playType
```

参数

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

返回

0表示成功，其它为失败；

> **说明**
> 10.7 版本开始，`startPlay` 变更为 `startLivePlay`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用，若您暂未获取上述 Licence，可 购买 Licence 。

stopPlay

停止播放音视频流。

```
- (int)stopPlay
```

返回
0：成功；其它：失败。

isPlaying

是否正在播放。

```
- (BOOL)isPlaying
```

返回
YES 拉流中，NO 没有拉流。

pause

暂停播放。

```
- (void)pause
```

介绍
适用于点播，直播（此接口会暂停数据拉流，不会销毁播放器，暂停后，播放器会显示最后一帧数据图像）。

resume

继续播放，适用于点播，直播。

```
- (void)resume
```

视频相关接口

setRenderRotation

设置画面的方向。

```
- (void)setRenderRotation:(TX_Enum_Type_HomeOrientation)rotation
```

参数

参数	类型	含义
rotation	TX_Enum_Type_HomeOrientation	方向。

setRenderMode

设置画面的裁剪模式。

```
- (void)setRenderMode:(TX_Enum_Type_RenderMode)renderMode
```

参数

参数	类型	含义
renderMode	TX_Enum_Type_RenderMode	裁剪。

snapshot

截屏。

```
- (void)snapshot:(void(^)(TXImage *))snapshotCompletionBlock
```

参数

参数	类型	含义
snapshotCompletionBlock	void(^)(TXImage *)	通过回调返回当前图像。

音频相关接口

setMute

设置静音。

```
- (void)setMute:(BOOL)bEnable
```

setVolume

设置音量。

```
- (void)setVolume:(int)volume
```

参数

参数	类型	含义
volume	int	音量大小，取值范围 0 - 100。

setAudioRoute

设置声音播放模式（切换扬声器，听筒）。

```
+ (void)setAudioRoute:(TXAudioRouteType)audioRoute
```

参数

参数	类型	含义
audioRoute	TXAudioRouteType	声音播放模式。

setAudioVolumeEvaluationListener

设置音量大小回调接口。

```
- (void)setAudioVolumeEvaluationListener:(void(^)(int))volumeEvaluationListener
```

参数

参数	类型	含义
volumeEvaluationListener	(void(^)(int))	音量大小回调接口。

enableAudioVolumeEvaluation

启用音量大小评估。

```
void enableAudioVolumeEvaluation(int intervalMs)
```

参数

参数	类型	含义
intervalMs	int	intervalMs 决定了 volumeEvaluationListener 回调的触发间隔，单位为ms，最小间隔为 100ms，如果小于等于 0 则会关闭回调，建议设置为 300ms。

介绍

开启后会在 volumeEvaluationListener 中获取到 SDK 对音量大小值的评估。

视频录制相关接口

startRecord

开始录制短视频。

```
- (int)startRecord:(TXRecordType)recordType
```

参数

参数	类型	含义
recordType	TXRecordType	参见 TXRecordType 定义。

返回

0：成功；1：正在录制短视频；-2：videoRecorder 初始化失败。

stopRecord

结束录制短视频。

```
- (int)stopRecord
```

返回

0：成功；1：不存在录制任务；-2：videoRecorder 未初始化。

更多实用接口

setLogViewMargin

设置状态浮层 view 在渲染 view 上的边距。

```
- (void)setLogViewMargin:(TXEdgeInsets)margin
```

参数

参数	类型	含义
margin	TXEdgeInsets	边距。

showVideoDebugLog

是否显示播放状态统计及事件消息浮层 view。

```
- (void)showVideoDebugLog:(BOOL)isShow
```

参数

参数	类型	含义
isShow	BOOL	是否显示。

switchStream

FLV 直播无缝切换。

```
- (int)switchStream:(NSString *)playUrl
```

参数

参数	类型	含义
playUrl	NSString *	播放地址。

返回

0：成功；其它：失败。

> **注意：**
> 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。

callExperimentalAPI

调用实验性 API 接口。

```
- (void)callExperimentalAPI:(NSString*)jsonStr
```

参数

参数	类型	含义
jsonStr	NSString *	jsonStr 接口及参数描述的 JSON 字符串。

介绍

该接口用于调用一些实验性功能。

TX_Enum_PlayType

功能

支持的直播和点播类型。

> **说明：**
> 点播播放请使用 TXVodPlayer 播放器，具体请参见头文件 TXVodPlayer.h。

枚举	含义

PLAY_TYPE_LIVE_RTMP	RTMP 直播。
PLAY_TYPE_LIVE_FLV	FLV 直播。
PLAY_TYPE_LIVE_HLS	HLS 点播。
PLAY_TYPE_LIVE_RTMP_ACC	RTMP 直播加速播放。

TXLivePlayListener

最近更新时间：2022-10-08 16:33:51

功能

腾讯云直播播放的回调通知。

onPlayEvent

直播事件通知。

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

参数

参数	类型	含义
EvtID	int	参见头文件 TXLiveSDKEventDef.h
param	NSDictionary *	参见头文件 TXLiveSDKTypeDef.h

onNetStatus

网络状态通知。

```
- (void)onNetStatus:(NSDictionary *)param
```

参数

参数	类型	含义
param	NSDictionary *	参见头文件 TXLiveSDKTypeDef.h

TXLivePlayConfig

最近更新时间：2023-07-06 16:54:42

功能

腾讯云直播播放器的参数配置模块。

介绍

主要负责 TXLivePlayer 对应的参数设置，其中绝大多数设置项在播放开始之后再设置是无效的。

属性列表

属性	类型	字段含义	推荐取值	特别说明
cacheTime	float	播放器缓存时间，单位秒，取值需要大于0，默认值：5。	-	-
bAutoAdjustCacheTime	BOOL	是否自动调整播放器缓存时间，默认值：YES YES：启用自动调整，自动调整的最大值和最小值可以分别通过修改 maxCacheTime 和 minCacheTime 来设置 NO：关闭自动调整，采用默认的指定缓存时间（1s），可以通过修改 cacheTime 来调整缓存时间。	-	-
maxAutoAdjustCacheTime	float	播放器缓存自动调整的最大时间，单位秒，取值需要大于0，默认值：5。	-	-
minAutoAdjustCacheTime	float	播放器缓存自动调整的最小时间，单位秒，取值需要大于0，默认值为1。	-	-
videoBlockThreshold	int	播放器视频卡顿报警阈值，单位毫秒。	800	只有渲染间隔超过这个阈值的卡顿才会有 PLAY_WARNING_VIDEO_PLAY_LAG 通知。
connectRetryCount	int	播放器遭遇网络连接断开时 SDK 默认重试的次数，取值范围1 - 10，默认值：3。	-	-
connectRetryInterval	int	网络重连的时间间隔，单位秒，取值范围 3 - 30，默认值：3。	-	-
enableAEC	BOOL	是否开启回声消除， 默认值为 NO。	-	-
enableMessage	BOOL	是否开启消息通道， 默认值为 NO。	-	-
playerPixelFormatType	OSType	视频渲染对象回调的视频格式，默认值：kCVPixelFormatType_420YpCbCr8Planar。	-	支持：kCVPixelFormatType_420YpCbCr8Planar 和 kCVPixelFormatType_420YpCbCr8BiPlanarFullRange。
enableNearestIP	BOOL	是否开启就近选路，待废弃，默认值：YES。	-	-
rtmpChannelType	int	RTMP 传输通道的类型，待废弃，默认值为：	-	-

		RTMP_CHANNEL_TYPE_AUTO。		
headers	NSDictionary *	自定义 HTTP Headers。	-	-

Android
概览

最近更新时间：2024-12-13 14:25:12

TXLivePlayer

视频播放器

请参见 TXLivePlayer 。
主要负责将直播流的音视频画面进行解码和本地渲染，包含如下技术特点：

- 针对腾讯云的拉流地址，可使用低延时拉流，实现直播连麦等相关场景。
- 针对腾讯云的拉流地址，可使用直播时移功能，能够实现直播观看与时移观看的无缝切换。
- 支持自定义的音视频数据处理，让您可以根据项目需要处理直播流中的音视频数据后，进行渲染以及播放。

SDK 基础函数

API	描述
TXLivePlayer	创建 TXLivePlayer 实例。
setConfig	设置 TXLivePlayer 播放配置项。
setPlayListener	设置推流回调接口。

播放基础接口

API	描述
setPlayerView	设置播放器的视频渲染 View。
startLivePlay	播放器开始播放。10.7 版本开始，`startPlay` 变更为 `startLivePlay`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用，若您暂未获取上述 Licence，可 购买 Licence。
stopPlay	停止播放。
isPlaying	是否正在播放。
pause	暂停播放。
resume	恢复播放。
setSurface	使用 Surface 模式用于本地渲染。
setSurfaceSize	设置渲染 Surface 的大小。

播放配置接口

API	描述
setRenderMode	设置播放渲染模式。
setRenderRotation	设置图像渲染角度。
enableHardwareDecode	开启硬件加速。

setMute	设置是否静音播放。
setAudioRoute	设置声音播放模式。
setVolume	设置音量。
switchStream	多清晰度切换。
setAudioVolumeEvaluationListener	设置音量大小回调接口。

本地录制和截图

API	描述
setVideoRecordListener	设置录制回调接口。
startRecord	启动视频录制。
stopRecord	停止视频录制。
snapshot	播放过程中本地截图。

自定义数据处理

API	描述
addVideoRawData	设置软解码数据载体 Buffer。
setVideoRawDataListener	设置软解码视频数据回调。
setAudioRawDataListener	设置音频数据回调。

直播时移接口

API	描述
prepareLiveSeek	直播时移准备。
seek	直播时移跳转。
resumeLive	恢复直播播放。

截图回调接口类

请参见 ITXSnapshotListener 。

API	描述
onSnapshot	截图回调。

软解视频数据回调接口类

请参见 ITXVideoRawDataListener 。

API	描述
onVideoRawDataAvailable	软解码器解出一帧数据回调一次。

音频原始数据接口类

请参见 ITXAudioRawDataListener 。

API	描述
onPcmDataAvailable	音频播放数据回调，数据格式： PCM。
onAudioInfoChanged	音频播放信息回调。

播放器音量大小接口类

请参见 ITXAudioVolumeEvaluationListener 。

API	描述
onAudioVolumeEvaluationNotify	播放器音量大小回调, 取值范围 [0，100]。

TXLivePlayConfig

腾讯云直播播放器的参数配置模块

请参见 TXLivePlayConfig 。
主要负责 TXLivePlayer 对应的参数设置，其中绝大多数设置项在播放开始之后再设置是无效的。

常用设置项

API	描述
setAutoAdjustCacheTime	设置是否自动调整缓存时间。
setCacheTime	设置播放器缓存时间。
setMaxAutoAdjustCacheTime	设置最大的缓存时间。
setMinAutoAdjustCacheTime	设置最小的缓存时间。
setVideoBlockThreshold	设置播放器视频卡顿报警阈值。
setConnectRetryCount	设置播放器重连次数。
setConnectRetryInterval	设置播放器重连间隔。

专业设置项

API	描述
setEnableMessage	开启消息通道。
enableAEC	设置回声消除。

ITXLivePlayListener

腾讯云直播播放的回调通知

请参见 ITXLivePlayListener 。

API	描述
onPlayEvent	播放事件通知。
onNetStatus	网络状态通知。

TXLivePusher

直播推流类

请参见 TXLivePusher 。
主要负责将本地的音视频画面进行编码和 RTMP 推送，包含如下技术特点：

- 针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。
- 内嵌套的 Qos 流量控制技术具备上行网络自适应能力，可以根据主播端网络的具体情况实时调节音视频数据量。
- 内嵌多套美颜磨皮算法（自然&光滑）和多款色彩空间滤镜（支持自定义滤镜），可以根据需要自行选择。
- 商业版包含了基于优图 AI 识别技术的大眼、瘦脸、隆鼻以及动效挂架，只需要购买**优图 License** 就可以零成本集成。
- 支持自定义的音视频采集和渲染，让您可以根据项目需要选择自己的音视频数据源。

SDK 基础函数

API	描述
TXLivePusher	创建 TXLivePusher 实例。
setConfig	设置 TXLivePusher 推流配置项。
getConfig	获取推流器配置信息。
setPushListener	设置推流回调接口。

推流基础接口

API	描述
startCameraPreview	启动摄像头预览。
stopCameraPreview	停止摄像头预览。
startPusher	启动 RTMP 推流。
stopPusher	停止 RTMP 推流。
startScreenCapture	启动录屏推流（基于 MediaProjection 技术实现）。
stopScreenCapture	结束录屏推流。
pausePusher	暂停摄像头或屏幕采集并进入垫片推流状态。
resumePusher	恢复摄像头采集并结束垫片推流状态。
isPushing	查询是否正在推流。

视频相关接口

API	描述
setVideoQuality	设置视频编码质量。
switchCamera	切换前后摄像头。
setMirror	设置视频镜像效果。
setRenderRotation	设置本地摄像头预览画面的旋转方向。

turnOnFlashLight	打开后置摄像头旁边的闪光灯。
getMaxZoom	获取摄像头支持的焦距。
setZoom	调整摄像头的焦距。
setExposureCompensation	调整曝光比例。

美颜相关接口

API	描述
getBeautyManager	获取美颜管理对象 TXBeautyManager，美颜的设置通过 TXBeautyManager 来设置。

音频相关接口

API	描述
setMute	开启静音。
setBGMNofify	设置背景音乐的回调接口。
playBGM	播放背景音乐。
stopBGM	停止播放背景音乐。
pauseBGM	暂停播放背景音乐。
resumeBGM	继续播放背音乐。
getMusicDuration	获取背景音乐文件的总时长，单位是毫秒。
setBGMVolume	设置混音时背景音乐的音量大小，仅在播放背景音乐混音时使用。
setMicVolume	设置混音时麦克风音量大小，仅在播放背景音乐混音时使用。
setBgmPitch	调整背景音乐的音调高低。
setReverb	设置混响效果。
setVoiceChangerType	设置变声类型。

本地录制接口

API	描述
setVideoRecordListener	设置录制回调接口。
startRecord	开始录制短视频。
stopRecord	结束录制短视频，当停止推流后，如果视频还在录制中，SDK 内部会自动结束录制。
snapshot	推流过程中本地截图。

自定义采集和处理

API	描述
sendCustomVideoTexture	自定义视频采集，向 SDK 发送自己采集的 texture 视频数据。
sendCustomVideoData	自定义视频采集，向 SDK 发送自己采集的 YUV 视频数据。

sendCustomPCMData	自定义音频采集，向 SDK 发送自己采集的音频 PCM 数据。
setVideoProcessListener	自定义视频处理回调。
setAudioProcessListener	自定义音频处理回调。
setSurface	指定 SDK 渲染所使用的 Surface（仅供微信 App 使用）。
setSurfaceSize	设置渲染 Surface 的大小（仅供微信 App 使用）。
setFocusPosition	在 Surface 模式下，设置摄像机的对焦位置。

更多实用接口

API	描述
sendMessageEx	发送 SEI 消息，播放端 TXLivePlayer 通过 onPlayEvent(EVT_PLAY_GET_MESSAGE) 来接收该消息。
sendMessage	-
onLogRecord	输出自己的 log，保存到 SDK 内部的 xlog 文件中。

自定义视频处理回调类

请参见 VideoCustomProcessListener 。

API	描述
onTextureCustomProcess	在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理。
onDetectFacePoints	增值版回调人脸坐标。
onTextureDestoryed	在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源。

自定义音频处理回调类

请参见 AudioCustomProcessListener 。

API	描述
onRecordRawPcmData	回调未经过任何处理的 SDK 录制音频 PCM 数据。
onRecordPcmData	回调 SDK 录制音频 PCM 数据。

背景音乐回调类

请参见 OnBGMNotify 。

API	描述
onBGMStart	音乐播放开始的回调通知。
onBGMProgress	音乐播放进度的回调通知。
onBGMComplete	音乐播放结束的回调通知。

截图回调类

请参见 ITXSnapshotListener 。

API	描述

onSnapshot	-

TXLivePushConfig

腾讯云直播推流用 RTMP SDK 的参数配置模块

请参见 TXLivePushConfig 。
主要负责 TXLivePusher 对应的参数设置，其中绝大多数设置项在推流开始之后再设置是无效的。

常用设置项

API	描述
setHomeOrientation	设置采集的视频的旋转角度。
setTouchFocus	设置是否开启手动对焦。
setEnableZoom	设置是否允许双指手势放大预览画面。
setWatermark	设置水印图片及水印图片位置。
setWatermark	设置水印图片及水印图片位置。
setLocalVideoMirrorType	设置本地预览画面的镜像类型。
setVolumeType	设置系统音量类型。

垫片推流

API	描述
setPauseImg	设置垫片推流的图片素材。
setPauseImg	设置垫片的帧率与最长持续时间。
setPauseFlag	设置后台推流的选项。

音视频编码参数

API	描述
setVideoResolution	设置采集的视频的分辨率。
setVideoFPS	设置视频帧率。
setVideoEncodeGop	设置视频编码 GOP。
setVideoBitrate	设置视频编码码率。
setMaxVideoBitrate	设置最大视频码率。
setMinVideoBitrate	设置最小视频码率。
setAutoAdjustBitrate	设置是否开启码率自适应。
setAutoAdjustStrategy	设置动态调整码率的策略。
setAudioSampleRate	设置声音采样率。
setAudioChannels	设置声道数。

enablePureAudioPush	开启纯音频推流。
enableScreenCaptureAutoRotate	设置录屏推流时是否要根据情况自适应旋转（仅用于录屏推流）。
enableHighResolutionCaptureMode	是否固定摄像头的采集分辨率为720p。
setVideoEncoderXMirror	设置观众端水平镜像。

网络相关参数

API	描述
setConnectRetryCount	设置推流端重连次数。
setConnectRetryInterval	设置推流端重连间隔。

自定义采集和处理

API	描述
setCustomModeType	自定义采集和自定义处理开关。

专业设置项

API	描述
enableAEC	设置回声消除。
enableAGC	设置自动增益。
enableANS	设置噪声抑制。
setHardwareAcceleration	设置硬件加速选项。
enableVideoHardEncoderMainProfile	是否开启 MainProfile 硬编码模式。

ITXLivePushListener

腾讯云直播推流的回调通知

请参见 ITXLivePushListener 。

API	描述
onPushEvent	推流事件通知。
onNetStatus	网络状态通知。

MLVBLiveRoom

腾讯云直播 SDK - 连麦直播间

请参见 MLVBLiveRoom 。

基于腾讯云直播、云点播（VOD）和即时通信（IM）三大 PAAS 服务组合而成，支持：

- 主播创建新的直播间开播，观众进入直播间观看。
- 主播和观众进行视频连麦互动。
- 两个不同房间的主播 PK 互动。
- 一个直播间都有一个不限制房间人数的聊天室，支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。

连麦直播间（MLVBLiveRoom）是一个开源的 Class，依赖两个腾讯云的闭源 SDK：

- LiteAVSDK：使用了其中的 TXLivePusher 和 TXLivePlayer 两个组件，前者用于推流，后者用于拉流。
- IM SDK：使用 IM SDK 的 AVChatroom 用于实现直播聊天室的功能，同时，主播间的连麦流程也是依靠 IM 消息串联起来的。

请参见 直播连麦 。

SDK 基础函数

API	描述
sharedInstance	获取 MLVBLiveRoom 单例对象。
destroySharedInstance	销毁 MLVBLiveRoom 单例对象。
setListener	设置回调接口。
setListenerHandler	设置驱动回调的线程。
login	登录。
logout	退出登录。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
getRoomList	获取房间列表。
getAudienceList	获取观众列表。
createRoom	创建房间（主播调用）。
enterRoom	进入房间（观众调用）。
exitRoom	离开房间。
setCustomInfo	设置自定义信息。
getCustomInfo	获取自定义信息。

主播和观众连麦

API	描述
requestJoinAnchor	观众请求连麦。
responseJoinAnchor	主播处理连麦请求。
joinAnchor	进入连麦状态。
quitJoinAnchor	观众退出连麦。
kickoutJoinAnchor	主播踢除连麦观众。

主播跨房间 PK

API	描述
requestRoomPK	请求跨房 PK。

responseRoomPK	响应跨房 PK 请求。
quitRoomPK	退出跨房 PK。

视频相关接口函数

API	描述
startLocalPreview	开启本地视频的预览画面。
stopLocalPreview	停止本地视频采集及预览。
startRemoteView	启动渲染远端视频画面。
stopRemoteView	停止渲染远端视频画面。
startScreenCapture	启动录屏。
stopScreenCapture	结束录屏。

音频相关接口函数

API	描述
muteLocalAudio	是否屏蔽本地音频。
muteRemoteAudio	设置指定用户是否静音。
muteAllRemoteAudio	设置所有远端用户是否静音。

摄像头相关接口函数

API	描述
switchCamera	切换摄像头。
setZoom	设置摄像头缩放因子（焦距）。
enableTorch	开关闪光灯。
setCameraMuteImage	主播屏蔽摄像头期间需要显示的等待图片。
setCameraMuteImage	主播屏蔽摄像头期间需要显示的等待图片。

美颜滤镜相关接口函数

API	描述
getBeautyManager	获取美颜管理对象 TXBeautyManager，美颜的设置通过 TXBeautyManager 来设置。
setFilterConcentration	设置滤镜浓度。
setWatermark	添加水印，height 不用设置，SDK 内部会根据水印宽高比自动计算 height。
setExposureCompensation	调整曝光。

消息发送接口函数

API	描述

sendRoomTextMsg	发送文本消息。
sendRoomCustomMsg	发送自定义文本消息。

背景混音相关接口函数

API	描述
playBGM	播放背景音乐。
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。
getBGMDuration	获取音乐文件总时长。
setMicVolumeOnMixing	设置麦克风的音量大小，播放背景音乐混音时使用，用来控制麦克风音量大小。
setBGMVolume	设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景音音量大小。
setReverbType	设置混响效果。
setVoiceChangerType	设置变声类型。
setBgmPitch	设置背景音乐的音调。

IMLVBLiveRoomListener

MLVBLiveRoom 事件回调

请参见 IMLVBLiveRoomListener 。
包括房间关闭、Debug 事件信息及出错说明等。

通用事件回调

API	描述
onError	错误回调。
onWarning	警告回调。
onDebugLog	-

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onAnchorEnter	收到新主播进房通知。
onAnchorExit	收到主播退房通知。
onAudienceEnter	收到观众进房通知。
onAudienceExit	收到观众退房通知。
onRequestJoinAnchor	主播收到观众连麦请求时的回调。

onKickoutJoinAnchor	连麦观众收到被踢出连麦的通知。
onRequestRoomPK	收到请求跨房 PK 通知。
onQuitRoomPK	收到断开跨房 PK 通知。

消息事件回调

API	描述
onRecvRoomTextMsg	收到文本消息。
onRecvRoomCustomMsg	收到自定义消息。

登录结果回调接口

请参见 LoginCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

获取房间列表回调接口

请参见 GetRoomListCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

获取观众列表回调接口

请参见 GetAudienceListCallback 。
观众进房时，后台会将其信息加入观众列表中，观众列表最大保存30名观众信息。

API	描述
onError	错误回调。
onSuccess	成功回调。

创建房间的结果回调接口

请参见 CreateRoomCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

进入房间的结果回调接口

请参见 EnterRoomCallback 。

API	描述

onError	错误回调。
onSuccess	成功回调。

离开房间的结果回调接口

请参见 ExitRoomCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

观众请求连麦的结果回调接口

请参见 RequestJoinAnchorCallback 。

API	描述
onAccept	主播接受连麦。
onReject	主播拒绝连麦。
onTimeOut	请求超时。
onError	错误回调。

进入连麦的结果回调接口

请参见 JoinAnchorCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

退出连麦的结果调用接口

请参见 QuitAnchorCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

请求跨房 PK 的结果回调接口

请参见 RequestRoomPKCallback 。

API	描述
onAccept	主播接受连麦。
onReject	拒绝 PK。
onTimeOut	请求超时。
onError	错误回调。

退出跨房 PK 的结果回调接口

请参见 QuitRoomPKCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

播放器回调接口

请参见 PlayCallback 。

API	描述
onBegin	开始回调。
onError	错误回调。
onEvent	其他事件回调。

发送文本消息回调接口

请参见 SendRoomTextMsgCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

发送自定义消息回调接口

请参见 SendRoomCustomMsgCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

设置自定义信息回调接口

请参见 SetCustomInfoCallback 。

API	描述
onError	错误回调。
onSuccess	成功回调。

获取自定义信息回调接口

请参见 GetCustomInfoCallback 。

API	描述
onError	错误回调。
onGetCustomInfo	获取自定义信息的回调。

推流
TXLivePusher

最近更新时间：2025-06-30 19:49:52

功能
直播推流类。
介绍
主要负责将本地的音视频画面进行编码和 RTMP 推送，包含如下技术特点：

- 针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。
- 内嵌套的 Qos 流量控制技术具备上行网络自适应能力，可以根据主播端网络的具体情况实时调节音视频数据量。
- 内嵌多套美颜磨皮算法（自然&光滑）和多款色彩空间滤镜（支持自定义滤镜），可以根据需要自行选择。
- 企业版 SDK 包含了大眼、瘦脸以及隆鼻等功能，配合高级美颜动效素材，可快速地完成功能集成。
- 支持自定义的音视频采集和渲染，让您可以根据项目需要选择自己的音视频数据源。

SDK 基础函数

TXLivePusher

创建 TXLivePusher 实例。

```
TXLivePusher(Context context)
```

参数

参数	类型	含义
context	Context	上下文。

setConfig

设置 TXLivePusher 推流配置项。

```
void setConfig(TXLivePushConfig config)
```

参数

参数	类型	含义
config	TXLivePushConfig	推流配置项，请参见 TXLivePushConfig 。

getConfig

获取推流器配置信息。

```
TXLivePushConfig getConfig()
```

返回
推流器配置信息。

setPushListener

设置推流回调接口。

```
void setPushListener(ITXLivePushListener listener)
```

参数

参数	类型	含义
listener	ITXLivePushListener	播放器回调，请参考 ITXLivePushListener 。

推流基础接口

startCameraPreview

启动摄像头预览。

```
void startCameraPreview(TXCloudVideoView view)
```

参数

参数	类型	含义
view	TXCloudVideoView	承载视频画面的控件。

介绍

启动预览后并不会立即开始 RTMP 推流，需要调用 TXLivePusher#startPusher(String) 才能真正开始推流。

stopCameraPreview

停止摄像头预览。

```
void stopCameraPreview(boolean isNeedClearLastImg)
```

参数

参数	类型	含义
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。

返回

是否成功停止播放，0：成功；非0：失败。

startPusher

启动 RTMP 推流。

```
int startPusher(String rtmpURL)
```

参数

参数	类型	含义
rtmpURL	String	推流地址，请参见 获取推流地址 。

返回

0：成功； -1：启动失败。

介绍
针对腾讯云的推流地址，会采用 QUIC 协议进行加速，配合改进后的 BBR2 带宽测算方案，可以最大限度的利用主播的上行带宽，降低直播卡顿率。

stopPusher

停止 RTMP 推流。

```
void stopPusher()
```

startScreenCapture

启动录屏推流（基于 MediaProjection 技术实现）。

```
void startScreenCapture()
```

介绍
如果要开启“隐私模式”，请调用 pausePusher 接口推默认图及静音数据，取消隐私模式调用 resumePusher。

stopScreenCapture

结束录屏推流。

```
void stopScreenCapture()
```

pausePusher

暂停摄像头或屏幕采集并进入垫片推流状态。

```
void pausePusher()
```

介绍
SDK 会暂时停止摄像头或屏幕采集，并使用 TXLivePushConfig.pauseImg 中指定的图片作为替代图像进行推流，也就是所谓的“垫片”。这项功能常用于 App 被切到后台运行的场景，尤其是在 iOS 系统中，当 App 切到后台以后，操作系统不会再允许该 App 继续使用摄像头。此时就可以通过调用 pausePush() 进入垫片状态。
对于绝大多数推流服务器而言，如果超过一定时间不推视频数据，服务器会断开当前的推流链接。
在 TXLivePushConfig 您可以指定：

- pauseImg 设置后台推流的默认图片，不设置该参数，则默认为黑色背景。
- pauseFps 设置后台推流帧率，最小值为5，最大值为20，默认10。
- pauseTime 设置后台推流持续时长，单位秒，默认300秒。

> **说明**
> 请注意调用顺序：startPush => (pausePush => resumePush) => stopPush()，错误的调用顺序会导致 SDK 表现异常。

resumePusher

恢复摄像头采集并结束垫片推流状态。

```
void resumePusher()
```

isPushing

查询是否正在推流。

```
boolean isPushing()
```

返回
true：正在推流，false：未推流。

视频相关接口

setVideoQuality

设置视频编码质量。

```
void setVideoQuality(int quality, boolean adjustBitrate, boolean adjustResolution)
```

参数

参数	类型	含义
quality	int	画质类型（标清、高清、超高清）。
adjustBitrate	boolean	动态码率开关。
adjustResolution	boolean	动态切分辨率开关。

介绍
推荐设置：秀场直播 quality：HIGH_DEFINITION；adjustBitrate：false；adjustResolution：false。请参见 设定清晰度 。

> **说明**
> adjustResolution 早期被引入是为了让 TXLivePusher 能够满足视频通话这一封闭场景下的一些需求，现已不推荐使用。如果您有视频通话的需求，可以使用我们专门为视频通话打造的 TRTC 服务。由于目前很多 H5 播放器不支持分辨率动态变化，所以开启分辨率自适应以后，会导致 H5 播放端和录制文件的很多兼容问题。

switchCamera

切换前后摄像头。

```
void switchCamera()
```

> **说明**
> 默认使用前置摄像头，该接口在启动预览 startCameraPreview(TXCloudVideoView) 后调用才能生效，预览前调用无效。

setMirror

设置视频镜像效果。

```
boolean setMirror(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：播放端看到的是镜像画面；false：播放端看到的是非镜像画面。

介绍

由于前置摄像头采集的画面是取自手机的观察视角，如果将采集到的画面直接展示给观众，是完全没有问题的。但如果将采集到的画面也直接显示给主播，则会跟主播照镜子时的体验完全相反，会让主播感觉到很奇怪。因此，SDK 会默认开启本地摄像头预览画面的镜像效果，让主播直播时跟照镜子时保持一个体验效果。
setMirror 所影响的则是观众端看到的视频效果，如果想要保持观众端看到的效果跟主播端保持一致，需要开启镜像； 如果想要让观众端看到正常的未经处理过的画面（例如主播弹吉他的时候有类似需求），则可以关闭镜像。

setRenderRotation

设置本地摄像头预览画面的旋转方向。

```
void setRenderRotation(int rotation)
```

参数

参数	类型	含义
rotation	int	取值为0，90，180，270（其他值无效），表示主播端摄像头预览视频的顺时针旋转角度。

介绍
该接口仅能够改变主播本地预览画面的方向，而不会改变观众端的画面效果。如果希望改变观众端看到的视频画面的方向，例如原来是540x960，希望变成960x540，则可以通过设置 TXLivePushConfig 中的 homeOrientation 来实现。

```
// 竖屏推流（HOME 键在下）
_config.homeOrientation = TXLiveConstants.VIDEO_ANGLE_HOME_DOWN;
[_txLivePublisher setConfig:_config];
[_txLivePublisher setRenderRotation:0];</pre>

<pre>// 横屏推流（HOME 键在右）
_config.homeOrientation = TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT;
[_txLivePublisher setConfig:_config];
[_txLivePublisher setRenderRotation:90];
```

turnOnFlashLight

打开后置摄像头旁边的闪光灯。

```
boolean turnOnFlashLight(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：打开闪光灯； false：关闭闪光灯。

返回
true：打开成功；false：打开失败。
介绍
此操作对于前置摄像头是无效的，因为绝大多数手机都没有给前置摄像头配置闪光灯。

getMaxZoom

获取摄像头支持的焦距。

```
int getMaxZoom()
```

返回
0：不支持变焦；大于0：最大焦距。

setZoom

调整摄像头的焦距。

```
boolean setZoom(int value)
```

参数

参数	类型	含义
value	int	焦距大小，取值范围：1 – getMaxZoom()，默认值建议设置为1即可。

返回
true：成功； false：失败。

> **说明**
> 当 distance 为1的时候为最远视角（正常镜头），当为 getMaxZoom() 的时候为最近视角（放大镜头），最大值不要超过，超过后画面会模糊不清。

setExposureCompensation

调整曝光比例。

```
void setExposureCompensation(float value)
```

参数

参数	类型	含义
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，-1是最小值，对应getMinExposureCompensation。正数表示调高曝光，1是最大值，对应getMaxExposureCompensation。0表示不调整曝光，默认值为0。

美颜相关接口

getBeautyManager

获取美颜管理对象 TXBeautyManager。
通过美颜管理，您可以使用以下功能：

- 设置”美颜风格”、”美白”、“红润”、“大眼”、“瘦脸”、“V脸”、“下巴”、“短脸”、“小鼻”、“亮眼”、“白牙”、“祛眼袋”、“祛皱纹”、“祛法令纹”等美容效果。
- 调整“发际线”、“眼间距”、“眼角”、“嘴形”、“鼻翼”、“鼻子位置”、“嘴唇厚度”、“脸型”。
- 设置人脸挂件（素材）等动态效果。
- 添加美妆。
- 进行手势识别。

```
public TXBeautyManager getBeautyManager()
```

音频相关接口

setMute

开启静音。

```
void setMute(boolean mute)
```

参数

参数	类型	含义
mute	boolean	true：静音；false：不静音。

介绍

开启静音后，SDK 并不会继续采集麦克风的声音，但是会用非常低（5kbps左右）的码率推送伪静音数据， 这样做的目的是为了兼容 H5 上的 video 标签，并让录制出来的 MP4 文件有更好的兼容性。

setBGMNofify

设置背景音乐的回调接口。

```
void setBGMNofify(OnBGMNotify notify)
```

参数

参数	类型	含义
notify	OnBGMNotify	回调接口。

playBGM

播放背景音乐。

```
boolean playBGM(String path)
```

参数

参数	类型	含义
path	String	背景音乐文件路径。

返回

true：播放成功；false：播放失败。

介绍

SDK 会将背景音乐和麦克风采集的声音进行混合并一起推送到云端。

stopBGM

停止播放背景音乐。

```
boolean stopBGM()
```

返回

true：停止播放成功； false：停止播放失败。

pauseBGM

暂停播放背景音乐。

```
boolean pauseBGM()
```

返回
true：暂停播放成功； false：暂停播放失败。

resumeBGM

继续播放背音乐。

```
boolean resumeBGM()
```

返回
true：恢复播放成功； false：恢复播放失败。

getMusicDuration

获取背景音乐文件的总时长，单位是毫秒。

```
int getMusicDuration(String path)
```

参数

参数	类型	含义
path	String	音乐文件路径，如果 path 为空，那么返回当前正在播放的背景音乐的时长。

setBGMVolume

设置混音时背景音乐的音量大小，仅在播放背景音乐混音时使用。

```
boolean setBGMVolume(float x)
```

参数

参数	类型	含义
x	float	音量大小，1为正常音量，范围是：[0 ~ 1] 之间的浮点数。

返回
true：成功；false：失败。

setMicVolume

设置混音时麦克风音量大小，仅在播放背景音乐混音时使用。

```
boolean setMicVolume(float x)
```

参数

参数	类型	含义
x	float	音量大小，1为正常音量，范围是：[0 ~ 1] 之间的浮点数。

返回
true：成功；false：失败。

setBgmPitch

调整背景音乐的音调高低。

```
void setBgmPitch(float pitch)
```

参数

参数	类型	含义
pitch	float	音调，默认值是0.0f，范围是：-1 - 1之间的浮点数。

setReverb

设置混响效果。

```
void setReverb(int reverbType)
```

参数

参数	类型	含义
reverbType	int	混响类型，具体值请参见 TXLiveConstants 中的 REVERB_TYPE_X 定义。

setVoiceChangerType

设置变声类型。

```
void setVoiceChangerType(int voiceChangerType)
```

参数

参数	类型	含义
voiceChangerType	int	具体值请参见 TXLiveConstants 中的 VOICECHANGER_TYPE_X 定义。

本地录制接口

setVideoRecordListener

设置录制回调接口。

```
void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)
```

参数

参数	类型	含义
listener	TXRecordCommon.ITXVideoRecordListener	录制回调接口。

startRecord

开始录制短视频。

```
int startRecord(final String videoFilePath)
```

参数

参数	类型	含义
videoFilePath	final String	视频录制后存储路径。

返回

0：成功；-1：videoPath 为空；-2：上次录制尚未结束，请先调用 stopRecord；-3：推流尚未开始。

> **说明**
> - 只有启动推流后才能开始录制，非推流状态下启动录制无效。
> - 出于安装包体积的考虑，仅专业版和商业版两个版本的 LiteAVSDK 支持该功能，直播精简版仅定义了接口但并未实现。
> - 录制过程中请勿动态切换分辨率和软硬编，会有很大概率导致生成的视频异常。

stopRecord

结束录制短视频，当停止推流后，如果视频还在录制中，SDK 内部会自动结束录制。

```
void stopRecord()
```

返回

0：成功； -1：不存在录制任务。

snapshot

推流过程中本地截图。

```
void snapshot(final ITXSnapshotListener listener)
```

参数

参数	类型	含义
listener	final ITXSnapshotListener	截图完成回调。

自定义采集和处理

sendCustomVideoTexture

自定义视频采集，向 SDK 发送自己采集的 texture 视频数据。

```
int sendCustomVideoTexture(int textureID, int w, int h)
```

参数

参数	类型	含义
textureID	int	视频纹理 ID。
w	int	视频图像的宽度，不能大于 TXLivePushConfig 中的 videoResolution 属性设定的宽度，否则会失败，小于时 SDK 会自动裁剪。
h	int	视频图像的高度，不能大于 TXLivePushConfig 中的 videoResolution 属性设定的宽度，否则会失败，小于时 SDK 会自动裁剪。

返回

返回视频数据的发送结果：

- 0：发送成功；
- 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。

介绍

在自定义视频采集模式下，SDK 不再继续从摄像头采集图像，只保留编码和发送能力，您需要定时地发送自己采集的视频数据。 要开启自定义视频采集，需要完成如下两个步骤：

- 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_VIDEO_CAPTURE 选项，代表开启自定义视频采集。
- 设定视频分辨率：将 TXLivePushConfig 中的 videoResolution 属性设置为您发送 YUV 数据的 width、height。

> **说明**
>
> - 该接口目前仅支持在 OpenGL（EGL10）环境线程调用，SDK 内部会自动获取 EGL10 的上下文，然后维护一个 sharecontext。
> - 目前仅支持普通纹理，暂不支持外部纹理。
> - 开启自定义视频采集后，即无需再调用 startPreview 来开启摄像头采集。
> - SDK 内部不再做帧率控制，请务必保证调用该函数的频率和 TXLivePushConfig 中设置的帧率一致，否则编码器输出的码率会不受控制。

sendCustomVideoData

自定义视频采集，向 SDK 发送自己采集的 YUV 视频数据。

```
int sendCustomVideoData(byte [] buffer, int bufferType, int w, int h)
```

参数

参数	类型	含义
buffer	byte []	视频数据。
bufferType	int	视频格式 目前仅支持 YUV_420P 、RGB_RGBA 两种数据格式。
w	int	视频图像的宽度；不能大于 TXLivePushConfig 中的 videoResolution 属性设定的宽度，否则会失败，小于时 SDK 会自动裁剪。
h	int	视频图像的高度；不能大于 TXLivePushConfig 中的 videoResolution 属性设定的宽度，否则会失败，小于时 SDK 会自动裁剪。

返回

如果返回值大于0，代表发送成功，但发送的帧率过高，超过了 TXLivePushConfig 中设置的帧率， 帧率过高会导致最终编出的码率超过 TXLivePushConfig 中设置的码率，返回值表示当前视频帧比预期提前的毫秒数。

- 0：发送成功
- 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

介绍

在自定义视频采集模式下，SDK 不再继续从摄像头采集图像，只保留编码和发送能力，您需要定时地发送自己采集的视频数据。 要开启自定义视频采集，需要完成如下两个步骤：

- 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_VIDEO_CAPTURE 选项，代表开启自定义视频采集。
- 设定视频分辨率：将 TXLivePushConfig 中的 videoResolution 属性设置为您发送 YUV 数据的 width、height。

> **说明**
> - 开启自定义视频采集后，即无需再调用 startPreview 来开启摄像头采集。
> - buffer 方式的处理性能要比 texture 方式的处理性能差很多。
> - SDK 内部不再做帧率控制，请务必保证调用该函数的频率和 TXLivePushConfig 中设置的帧率一致，否则编码器输出的码率会不受控制。

sendCustomPCMData

自定义音频采集，向 SDK 发送自己采集的音频 PCM 数据。

```
void sendCustomPCMData(byte [] pcmBuffer)
```

参数

参数	类型	含义
pcmBuffer	byte []	pcm 音频数据。

介绍

在自定义音频采集模式下，SDK 不再继续从麦克风采集声音，只保留编码和发送能力，您需要定时地发送自己采集的声音数据（PCM 格式）
要开启自定义音频采集，需要完成如下两个步骤：

- 开启自定义采集：给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_AUDIO_CAPTURE 选项，代表开启自定义音频采集。
- 设定音频采样率：将 TXLivePushConfig 中的 audioSampleRate 属性设置为您期望的音频采样率，audioChannels 设置为期望的声道数，默认值：1（单声道）。

> **说明**
> SDK 对每次传入的 PCM buffer 大小有严格要求，每一个采样点要求是16位宽。 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。

setVideoProcessListener

自定义视频处理回调。

```
void setVideoProcessListener(VideoCustomProcessListener listener)
```

介绍

自定义视频采集和自定义视频处理不能同时开启，与自定义视频采集不同，自定义视频处理依然是由 SDK 采集摄像头的画面， 但 SDK 会通过 VideoCustomProcessListener 回调将数据回调给您的 App 进行二次加工。
如果要开启自定义视频处理，需要给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_VIDEO_PREPROCESS 选项。

> **说明**
> 出于性能和稳定性考虑，一般不建议开启此特性。

setAudioProcessListener

自定义音频处理回调。

```
void setAudioProcessListener(AudioCustomProcessListener listener)
```

介绍

自定义音频采集和自定义音频处理不能同时开启，与自定义音频采集不同，自定义音频处理依然是由 SDK 采集麦克风的声音， 但 SDK 会通过 AudioCustomProcessListener 回调将数据回调给您的 App 进行二次加工。
如果要开启自定义音频处理，需要给 TXLivePushConfig 中的 customModeType 属性增加 CUSTOM_MODE_AUDIO_PREPROCESS 选项。

> **说明**
> 出于性能和稳定性考虑，一般不建议开启此特性。

setSurface

指定 SDK 渲染所使用的 Surface（仅供微信 App 使用）。

```
void setSurface(Surface surface)
```

参数

参数	类型	含义
surface	Surface	渲染 surface，Nonnull：开始渲染；null：停止渲染。

介绍

该接口是为了支持微信小程序最新版本中的同层渲染能力而增加的，目的是让微信小程序通知设置渲染用的 Surface， 我们推荐您不要使用此接口，建议直接使用 TXCloudVideoView。

> **说明**
> - 使用该接口需要 startCameraPreview(TXCloudVideoView) 传入 null。
> - 此功能为高级特性，除非您非常明确需要使用该特性，否则建议您使用 startCameraPreview(TXCloudVideoView)。

setSurfaceSize

设置渲染 Surface 的大小（仅供微信 App 使用）。

```
void setSurfaceSize(int width, int height)
```

参数

参数	类型	含义
width	int	surface 宽度。
height	int	surface 高度。

介绍

该接口是为了支持微信小程序最新版本中的同层渲染能力而增加的，目的是让微信小程序通知设置渲染用的 Surface， 我们推荐您不要使用此接口，建议直接使用 TXCloudVideoView。

> **说明**
> - Surface 大小变化后，需要重新设定。
> - 此功能为高级特性，除非您非常明确需要使用该特性，否则建议您使用 startCameraPreview(TXCloudVideoView)。

setFocusPosition

在 Surface 模式下，设置摄像机的对焦位置。

```
void setFocusPosition(float x, float y)
```

参数

参数	类型	含义
x	float	聚焦点位置 x 值。
y	float	聚焦点位置 y 值。

> **说明**
> - Surface 模式下，需要您自行监听点击事件，通知 SDK 进行对焦；详细实现方式，可参考 Demo 实现。
> - 此功能为高级特性，除非您非常明确需要使用该特性，否则建议您使用 startCameraPreview(TXCloudVideoView)。

更多实用接口

sendMessageEx

发送 SEI 消息，播放端 TXLivePlayer 通过 onPlayEvent(PLAY_EVT_GET_MESSAGE) 来接收该消息。

```
boolean sendMessageEx(byte [] msg)
```

参数

参数	类型	含义
msg	byte []	-

返回
true：消息发送成功；false：消息发送失败。

> **说明**
> - sendMessage 已经不推荐使用，会导致 H5 播放器产生兼容性问题，请使用 sendMessageEx。
> - 若您使用过 sendMessage，不推荐立刻升级到 sendMessageEx。
> - sendMessageEx 发送消息给旧版本5.0及以前的 SDK 版本时，消息会无法正确解析，但播放不受影响。

sendMessage

```
void sendMessage(byte [] msg)
```

onLogRecord

输出自己的 log，保存到 SDK 内部的 xlog 文件中。

```
void onLogRecord(String str)
```

参数

参数	类型	含义
str	String	存入本地文件的 log。

> **说明**
> 此功能一般仅用在协助调试的情况下。

VideoCustomProcessListener

功能
自定义视频处理回调类。

onTextureCustomProcess

在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理。

```
int onTextureCustomProcess(int textureId, int width, int height)
```

参数

参数	类型	含义
textureId	int	纹理 ID。
width	int	纹理的宽度。
height	int	纹理的高度。

返回
返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可。

onDetectFacePoints

企业版回调人脸坐标。

```
void onDetectFacePoints(float [] points)
```

参数

参数	类型	含义
points	float []	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]。

onTextureDestoryed

在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源。

```
void onTextureDestoryed()
```

AudioCustomProcessListener

功能

自定义音频处理回调类。

onRecordRawPcmData

回调未经过任何处理的 SDK 录制音频 PCM 数据。

```
void onRecordRawPcmData(byte [] data, long ts, int sampleRate, int channels, int bits, boolean withBgm)
```

参数

参数	类型	含义
data	byte []	pcm 数据。
ts	long	pcm 对应时间戳。
sampleRate	int	音频采样率。
channels	int	音频通道。
bits	int	音频 bits。
withBgm	boolean	回调的数据是否包含 BGM，当不开启回声消除时，回调的 raw pcm 会包含 bgm。

onRecordPcmData

回调 SDK 录制音频 PCM 数据。

```
void onRecordPcmData(byte [] data, long ts, int sampleRate, int channels, int bits)
```

参数

参数	类型	含义
data	byte []	pcm 数据。
ts	long	pcm 对应时间戳。
sampleRate	int	音频采样率。
channels	int	音频通道。
bits	int	音频 bits。

OnBGMNotify

功能

背景音乐回调类。

onBGMStart

音乐播放开始的回调通知。

```
void onBGMStart()
```

onBGMProgress

音乐播放进度的回调通知。

```
void onBGMProgress(long progress, long duration)
```

参数

参数	类型	含义
progress	long	当前 BGM 已播放时间（ms）。
duration	long	当前 BGM 总时间（ms）。

onBGMComplete

音乐播放结束的回调通知。

```
void onBGMComplete(int err)
```

参数

参数	类型	含义
err	int	0：正常结束；-1：出错结束。

ITXSnapshotListener

功能
截图回调类。

onSnapshot

```
void onSnapshot(Bitmap bmp)
```

TXLivePushConfig

最近更新时间：2024-08-14 20:00:51

功能

腾讯云直播推流用 RTMP SDK 的参数配置模块。

介绍

主要负责 TXLivePusher 对应的参数设置，**其中绝大多数设置项在推流开始之后再设置是无效的。**

常用设置项

setHomeOrientation

设置采集的视频的旋转角度。

```
void setHomeOrientation(int homeOrientation)
```

参数

参数	类型	含义
homeOrientation	int	采集的视频的旋转角度。

介绍

接口说明：

- 默认值：TXLiveConstants.VIDEO_ANGLE_HOME_DOWN（竖屏推流）。
- 其他取值：TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT 和 TXLiveConstants.VIDEO_ANGLE_HOME_LEFT (横屏推流)。
- 改变该字段的设置以后，本地摄像头的预览画面方向也会发生改变，请调用 TXLivePusher 的 setRenderRotation 进行矫正。

setTouchFocus

设置是否开启手动对焦。

```
void setTouchFocus(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启手动对焦；false：不开启手动对焦。

介绍

接口说明：

- 默认值：true。
- 因为硬件的限制，API 14以上的版本以及后置摄像头才会支持。

setEnableZoom

设置是否允许双指手势放大预览画面。

```
void setEnableZoom(boolean enableZoom)
```

参数

参数	类型	含义
enableZoom	boolean	-

介绍

接口说明：

- 默认值：false。

setWatermark

设置水印图片及水印图片位置。

```
void setWatermark(Bitmap watermark, int x, int y)
```

参数

参数	类型	含义
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。

介绍

接口说明：

- 水印位置坐标系与系统保持一致。
- 设置为 null 关闭水印。

setWatermark

设置水印图片及水印图片位置。

```
void setWatermark(Bitmap watermark, float x, float y, float width)
```

参数

参数	类型	含义
watermark	Bitmap	水印图片。
x	float	归一化水印位置的 X 轴坐标，取值[0，1]。
y	float	归一化水印位置的 Y 轴坐标，取值[0，1]。
width	float	归一化水印宽度，取值[0，1]。

介绍

接口说明：

- 使用归一化坐标。
- 假设推流分辨率为：540 × 960，x，y，width 分别设置为：（0.1， 0.1， 0.1），那么水印的实际像素坐标为：（540 * 0.1， 960 * 0.1，水印宽度 * 0.1， 水印高度会被自动计算）。

setLocalVideoMirrorType

设置本地预览画面的镜像类型。

```
void setLocalVideoMirrorType(int mirrorType)
```

参数

参数	类型	含义
mirrorType	int	镜像类型。

介绍

接口说明：

- 默认值：TXLiveConstants#LOCAL_VIDEO_MIRROR_TYPE_AUTO。
- TXLiveConstants#LOCAL_VIDEO_MIRROR_TYPE_AUTO 表示由 SDK 决定镜像方式：前置摄像头镜像，后置摄像头不镜像。
- TXLiveConstants#LOCAL_VIDEO_MIRROR_TYPE_ENABLE 表示前置摄像头和后置摄像头都镜像。
- TXLiveConstants#LOCAL_VIDEO_MIRROR_TYPE_DISABLE 表示前置摄像头和后置摄像头都不镜像。

垫片推流

setPauseImg

设置垫片推流的图片素材。

```
void setPauseImg(Bitmap img)
```

参数

参数	类型	含义
img	Bitmap	bitmap 图片。

介绍

接口说明：

- 图片最大尺寸不能超过1920 × 1920。

setPauseImg

设置垫片的帧率与最长持续时间。

```
void setPauseImg(int time, int fps)
```

参数

参数	类型	含义
time	int	时间。
fps	int	帧率。

介绍

接口说明：

- 默认值：最大持续时间为300秒，帧率为10。
- 调用 TXLivePusher 的 pausePush() 接口，会暂停摄像头采集并进入垫片推流状态，如果该状态一直保持， 可能会消耗主播过多的手机流量，本字段用于指定垫片推流的最大持续时间，超过后不会断开云服务器的链接，但会进入纯音频推流。

> **说明**

> 若您的移动直播服务是在2018年09月之前开通的，指定垫片推流持续时间超过后即会断开云服务器的连接。

setPauseFlag

设置后台推流的选项。

```
void setPauseFlag(int flag)
```

参数

参数	类型	含义
flag	int	选项。

介绍

接口说明：

- 默认值：TXLiveConstants#PAUSE_FLAG_PAUSE_VIDEO。
- TXLiveConstants#PAUSE_FLAG_PAUSE_VIDEO 表示暂停推流时，采用 TXLivePushConfig#setPauseImg(Bitmap) 传入的图片作为画面推流，声音不做暂停，继续录制麦克风或 custom 音频发送。
- TXLiveConstants#PAUSE_FLAG_PAUSE_AUDIO 表示暂停推流时，推静音数据，画面数据不做暂停，继续发送摄像头、录屏或 custom 视频数据。
- TXLiveConstants#PAUSE_FLAG_PAUSE_VIDEO|TXLiveConstants#PAUSE_FLAG_PAUSE_AUDIO 表示暂停推流时，推送暂停图片和静音数据。

音视频编码参数

setVideoResolution

设置采集的视频的分辨率。

```
void setVideoResolution(int resolution)
```

参数

参数	类型	含义
resolution	int	分辨率。

介绍

接口说明：

- 默认值：TXLiveConstants#VIDEO_RESOLUTION_TYPE_540_960 。
- 其他值可参见 TXLiveConstants VIDEO_RESOLUTION_TYPE_XXX 。

setVideoFPS

设置视频帧率。

```
void setVideoFPS(int fps)
```

参数

参数	类型	含义
fps	int	帧率。

介绍

接口说明：

- 默认值：20。

setVideoEncodeGop

设置视频编码 GOP。

```
void setVideoEncodeGop(int gop)
```

参数

参数	类型	含义
gop	int	视频 GOP。

介绍

接口说明：

- 默认值：3，单位为秒。

setVideoBitrate

设置视频编码码率。

```
void setVideoBitrate(int bitrate)
```

参数

参数	类型	含义
bitrate	int	视频编码码率，单位：kbps。

介绍

接口说明：

- 默认值：1200。
- 不开启码率自适应时，视频以此码率编码。

setMaxVideoBitrate

设置最大视频码率。

```
void setMaxVideoBitrate(int maxBitrate)
```

参数

参数	类型	含义
maxBitrate	int	最大视频码率。

介绍

接口说明：

- 默认值：1000。
- 只有开启码率自适应，该设置项才能起作用：setAutoAdjustBitrate(boolean)。

setMinVideoBitrate

设置最小视频码率。

```
void setMinVideoBitrate(int minBitrate)
```

参数

参数	类型	含义
minBitrate	int	最小视频码率。

介绍

接口说明：

- 默认值：400。
- 只有开启码率自适应，该设置项才能起作用：setAutoAdjustBitrate(boolean)。

setAutoAdjustBitrate

设置是否开启码率自适应。

```
void setAutoAdjustBitrate(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启码率自适应，false：禁用码率自适应。

介绍

接口说明：

- 默认值：false。
- 开启后，SDK会根据网络情况自动调节视频码率，调节范围在（videoBitrateMin - videoBitrateMax）。

setAutoAdjustStrategy

设置动态调整码率的策略。

```
void setAutoAdjustStrategy(int strategy)
```

参数

参数	类型	含义
strategy	int	-

介绍

接口说明：

- 默认值： TXLiveConstants#AUTO_ADJUST_BITRATE_STRATEGY_1。
- 其他值： 请参见 TXLiveConstants 类中 AUTO_ADJUST_XXX 。

setAudioSampleRate

设置声音采样率。

```
void setAudioSampleRate(int sample)
```

参数

参数	类型	含义
sample	int	音频采样率。

介绍

接口说明：

- 默认值：48000。
- 其他值：8000、16000、32000、44100、48000。

setAudioChannels

设置声道数。

```
void setAudioChannels(int channels)
```

参数

参数	类型	含义
channels	int	声道数。

介绍

接口说明：

- 默认值：1。
- 其他值：1、2。

enablePureAudioPush

开启纯音频推流。

```
void enablePureAudioPush(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：启动纯音频推流；false：关闭纯音频推流。

介绍

接口说明：

- 默认值：false。
- 只有在推流启动前设置才会生效，推流过程中设置不会生效。

enableScreenCaptureAutoRotate

设置录屏推流时是否要根据情况自适应旋转（仅用于录屏推流）。

```
void enableScreenCaptureAutoRotate(boolean enable)
```

参数

参数	类型	含义

enable	boolean	true：视频内容为屏幕旋转后最大化显示；false：视频内容为屏幕内容缩放居中显示。

介绍

接口说明：

- 默认值：false。

enableHighResolutionCaptureMode

是否固定摄像头的采集分辨率为720p。

```
void enableHighResolutionCaptureMode(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启； false：关闭。

介绍

接口说明：

- 默认值：true，采用1280 × 720的采集分辨率。

setVideoEncoderXMirror

设置观众端水平镜像。

```
void setVideoEncoderXMirror(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：镜像；false：不镜像。

介绍

接口说明：

- 默认值：false。

网络相关参数

setConnectRetryCount

设置推流端重连次数。

```
void setConnectRetryCount(int count)
```

参数

参数	类型	含义
count	int	重连次数。

介绍

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。

接口说明：

- 默认值：3。

- 取值范围：1 – 10。

setConnectRetryInterval

设置推流端重连间隔。

```
void setConnectRetryInterval(int interval)
```

参数

参数	类型	含义
interval	int	SDK 重连间隔，单位秒。

介绍

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。

接口说明：

- 默认值：3秒。
- 取值范围：3秒 – 30秒。

自定义采集和处理

setCustomModeType

自定义采集和自定义处理开关。

```
void setCustomModeType(int modeType)
```

参数

参数	类型	含义
modeType	int	模式类型。

介绍

接口说明：

- 该字段需要使用与运算符进行级联操作（自定义采集和自定义处理不能同时开启）：
  - 开启自定义视频采集：_config.customModeType |= CUSTOM_MODE_VIDEO_CAPTURE。
  - 开启自定义音频采集：_config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE。
- 其他值：请参见 TXLiveConstants 中 CUSTOM_MODE_XXX 。

专业设置项

enableAEC

设置回声消除。

```
void enableAEC(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启回声消除； false：不开启。

介绍

接口说明：

- 默认值：false。
- 连麦时必须开启，非连麦时不要开启。

enableAGC

设置自动增益。

```
void enableAGC(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启自动增益； false：不开启。

介绍

接口说明：

- 默认值：false。

enableANS

设置噪声抑制。

```
void enableANS(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启噪声抑制； false：不开启。

介绍

接口说明：

- 默认值：false。

setVolumeType

设置系统音量类型。

```
void setVolumeType(int volumeType)
```

参数

参数	类型	含义
volumeType	int	系统音量类型

介绍

接口说明：

- 默认值：TXLiveConstants#AUDIO_VOLUME_TYPE_AUTO。
- 其他值
  - TXLiveConstants#AUDIO_VOLUME_TYPE_VOIP 表示通话音量类型。
  - TXLiveConstants#AUDIO_VOLUME_TYPE_MEDIA 表示媒体音量类型。

setHardwareAcceleration

设置硬件加速选项。

```
void setHardwareAcceleration(int encodeOpt)
```

参数

参数	类型	含义
encodeOpt	int	硬件加速选项。

介绍

接口说明：

- 默认值：TXLiveConstants#ENCODE_VIDEO_AUTO，自动选择是否启用硬件加速。
- 其他值：
  - TXLiveConstants#ENCODE_VIDEO_HARDWARE：开启硬件加速。
  - TXLiveConstants#ENCODE_VIDEO_SOFTWARE，禁用硬件加速，默认禁用硬件加速。

enableVideoHardEncoderMainProfile

是否开启 MainProfile 硬编码模式。

```
void enableVideoHardEncoderMainProfile(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启；false：关闭。

介绍

接口说明：

- 默认值：true，此参数仅在开启硬件编码加速时有效。

ITXLivePushListener

最近更新时间：2022-10-08 16:33:52

功能

腾讯云直播推流的回调通知。

onPushEvent

推流事件通知。

```
void onPushEvent(final int event, final Bundle param)
```

参数

参数	类型	含义
event	final int	事件 ID。ID 类型请参考头文件 TXLiveConstants.PUSH_EVT_CONNECT_SUCC 推流事件列表。
param	final Bundle	事件相关的参数。（key，value）格式，其中 key 请查看代码中的 TXLiveConstants.EVT_TIME 事件参数。

onNetStatus

网络状态通知。

```
void onNetStatus(final Bundle status)
```

参数

参数	类型	含义
status	final Bundle	通知的内容。（key，value）格式，其中 key 请查看代码中的 TXLiveConstants.NET_STATUS_VIDEO_BITRATE 网络状态通知。

拉流
TXLivePlayer

最近更新时间：2024-12-10 16:08:53

功能

视频播放器。

介绍

主要负责将直播流的音视频画面进行解码和本地渲染，包含如下技术特点：

- 针对腾讯云的拉流地址，可使用低延时拉流，实现直播连麦等相关场景。
- 针对腾讯云的拉流地址，可使用直播时移功能，能够实现直播观看与时移观看的无缝切换。
- 支持自定义的音视频数据处理，您可以根据项目需要处理直播流中的音视频数据，然后进行渲染以及播放。
- SDK 仅支持 Android 4.2 以上的版本。

SDK 基础函数

TXLivePlayer

创建 TXLivePlayer 实例。

```
TXLivePlayer(Context context)
```

参数

参数	类型	含义
context	Context	上下文。

setConfig

设置 TXLivePlayer 播放配置项。

```
void setConfig(TXLivePlayConfig config)
```

参数

参数	类型	含义
config	TXLivePlayConfig	播放器配置项，请参见 TXLivePlayConfig 。

setPlayListener

设置播放回调接口。

```
void setPlayListener(ITXLivePlayListener listener)
```

参数

参数	类型	含义
listener	ITXLivePlayListener	播放器回调，请参见 ITXLivePlayListener 。

播放基础接口

setPlayerView

设置播放器的视频渲染 View。

```
void setPlayerView(TXCloudVideoView glRootView)
```

参数

参数	类型	含义
glRootView	TXCloudVideoView	视频渲染 View。

startLivePlay

播放器开始播放。

```
int startLivePlay(String playUrl, int playType)
```

参数

参数	类型	含义
playUrl	String	播放的流地址。
playType	int	播放类型。

返回

是否成功启动播放， 0：成功；-1：失败，playUrl 为空；-2：失败，playUrl 非法；-3：失败，playType 非法；-5：licence 校验失败。

介绍

可播放的直播流连接：

- RTMP 直播流：PLAY_TYPE_LIVE_RTMP。
- FLV 直播流：PLAY_TYPE_LIVE_FLV。
- RTMP 加速流，用于连麦：PLAY_TYPE_LIVE_RTMP_ACC。

> **说明：**
> 在 `startLivePlay` 之前，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用，若您暂未获取上述 Licence，可 购买 Licence 。

stopPlay

停止播放。

```
int stopPlay(boolean isNeedClearLastImg)
```

参数

参数	类型	含义
isNeedClearLastImg	boolean	true：清除；false：不清除。

返回

0：成功；非0：失败。

介绍

isNeedClearLastImg 提供是否清除最后一帧画面的逻辑：

- 推荐在正常停止播放时，进行清除。
- 异常播放，如网络异常等，而您希望等待重连服务器，继续播放时，推荐保留。

isPlaying

是否正在播放。

```
boolean isPlaying()
```

返回

true：正在播放；false：未播放。

pause

暂停播放。

```
void pause()
```

介绍

停止获取流数据，保留最后一帧画面。

resume

恢复播放。

```
void resume()
```

介绍

重新获取数据，获取当前直播数据。

setSurface

使用 Surface 模式用于本地渲染。

```
void setSurface(Surface surface)
```

参数

参数	类型	含义
surface	Surface	视频渲染 surface。

说明

- 目前仅支持硬解。
- 使用该接口需要 setPlayerView(TXCloudVideoView) 传入 null。
- 此功能为高级特性，除非您需要使用该特性，否则建议您使用 setPlayerView(TXCloudVideoView) 。

setSurfaceSize

设置渲染 Surface 的大小。

```
void setSurfaceSize(int width, int height)
```

参数

参数	类型	含义
width	int	宽。
height	int	高。

说明

- Surface 大小变化后，需要重新设定。
- 此功能为高级特性，除非您需要使用该特性，否则建议您使用 setPlayerView(TXCloudVideoView) 。

播放配置接口

setRenderMode

设置播放渲染模式。

```
void setRenderMode(int mode)
```

参数

参数	类型	含义
mode	int	图像渲染模式，可以设置值为：TXLiveConstants#RENDER_MODE_FULL_FILL_SCREEN、TXLiveConstants#RENDER_MODE_ADJUST_RESOLUTION。

介绍

渲染模式有两种：

- 平铺模式：视频画面将会按照比例铺满屏幕，多余部分会被裁减掉，此模式下不会有黑边。
- 自适应模式：视频画面将等比例缩放，会居中显示，此模式可能会有黑边。

setRenderRotation

设置图像渲染角度。

```
void setRenderRotation(int rotation)
```

参数

参数	类型	含义
rotation	int	图像渲染角度，可设置值为：TXLiveConstants#RENDER_ROTATION_PORTRAIT、TXLiveConstants#RENDER_ROTATION_LANDSCAPE。

介绍

渲染角度有两种：

- 竖屏：播放是竖屏播放的时候使用。
- 横屏：播放是横屏播放的时候使用。

enableHardwareDecode

开启硬件加速。

```
boolean enableHardwareDecode(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：启用视频硬解码， false：禁用视频硬解码。

返回

true：关闭或开启硬件加速成功；false：关闭或开启硬件加速失败。

setMute

设置是否静音播放。

```
void setMute(boolean mute)
```

参数

参数	类型	含义
mute	boolean	true：静音播放；false：不静音播放。

setVolume

设置音量。

```
void setVolume(int volume)
```

参数

参数	类型	含义
volume	int	音量大小，取值范围 0 - 100。

setAudioRoute

设置声音播放模式。

```
void setAudioRoute(int audioRoute)
```

参数

参数	类型	含义
audioRoute	int	声音播放模式，可设置值：TXLiveConstants#AUDIO_ROUTE_RECEIVER、TXLiveConstants#AUDIO_ROUTE_SPEAKER。

介绍

播放模式有两种：

- 听筒：声音将从听筒播出。
- 扬声器：声音将从扬声器播出。

switchStream

FLV 多清晰度切换。

```
int switchStream(String playUrl)
```

参数

参数	类型	含义
playUrl	String	播放的流地址。

介绍

使用说明：

- 必须是腾讯云的直播地址。
- 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。

setAudioVolumeEvaluationListener

设置音量大小回调接口。

```
void setAudioVolumeEvaluationListener(ITXAudioVolumeEvaluationListener listener)
```

参数

参数	类型	含义
listener	ITXAudioVolumeEvaluationListener	音量大小回调接口。

enableAudioVolumeEvaluation

启用音量大小评估。

```
void enableAudioVolumeEvaluation(int intervalMs)
```

参数

参数	类型	含义
intervalMs	int	intervalMs 决定了 onAudioVolumeEvaluationNotify 回调的触发间隔，单位为ms，最小间隔为100ms，如果小于等于 0 则会关闭回调，建议设置为 300ms。

介绍

开启后会在 onAudioVolumeEvaluationNotify 中获取到 SDK 对音量大小值的评估。

callExperimentalAPI

调用实验性 API 接口。

```
void callExperimentalAPI(final String jsonStr)
```

参数

参数	类型	含义
jsonStr	String	jsonStr 接口及参数描述的 JSON 字符串。

介绍

该接口用于调用一些实验性功能。

本地录制和截图

setVideoRecordListener

设置录制回调接口。

```
void setVideoRecordListener(TXRecordCommon.ITXVideoRecordListener listener)
```

参数

参数	类型	含义
listener	TXRecordCommon.ITXVideoRecordListener	接口。

startRecord

启动视频录制。

```
int startRecord(int recordType)
```

参数

参数	类型	含义
recordType	int	TXRecordCommon#RECORD_TYPE_STREAM_SOURCE

返回
0表示成功，非0表示失败。
介绍
目前录制格式仅支持录制直播流，TXRecordCommon#RECORD_TYPE_STREAM_SOURCE。

stopRecord

停止视频录制。

```
int stopRecord()
```

返回
0表示成功，非0表示失败。

snapshot

播放过程中本地截图。

```
void snapshot(ITXSnapshotListener listener)
```

参数

参数	类型	含义
listener	ITXSnapshotListener	截图回调。

自定义数据处理

setVideoRawDataListener

设置软解码视频数据回调。

```
void setVideoRawDataListener(final ITXVideoRawDataListener listener)
```

说明

- 此功能会有一定的性能开销，特别是在高分辨率的情况下。
- 除非您有特殊的需求，否则不建议您开启。

setAudioRawDataListener

设置音频数据回调。

```
void setAudioRawDataListener(ITXAudioRawDataListener listener)
```

参数

参数	类型	含义
listener	ITXAudioRawDataListener	音频数据回调。

说明

- 音频播放器会在播放数据的前一刻，调用此函数，同步回调将要播放的数据。
- 请不要在函数内做耗时操作，否则会影响声音播放的流畅性。

ITXSnapshotListener

功能

截图回调接口类。

onSnapshot

截图回调。

```
void onSnapshot(Bitmap bmp)
```

参数

参数	类型	含义
bmp	Bitmap	当前视频图片。

ITXVideoRawDataListener

功能

软解视频数据回调接口类。

onVideoRawDataAvailable

软解码器解出一帧数据回调一次。

```
void onVideoRawDataAvailable(byte [] yuvBuffer, int width, int height, int timestamp)
```

参数

参数	类型	含义
yuvBuffer	byte []	I420 格式 YUV 数据。
width	int	视频宽度。
height	int	视频高度。
timestamp	int	视频 PTS。

ITXAudioRawDataListener

功能

音频原始数据接口类。

onPcmDataAvailable

音频播放数据回调，数据格式： PCM。

```
void onPcmDataAvailable(byte [] buf, long timestamp)
```

参数

参数	类型	含义
buf	byte []	pcm 数据。
timestamp	long	时间戳。

介绍

音频播放器会在播放数据的前一刻，调用此函数，同步回调将要播放的数据。因此在函数内部做耗时操作可能会影响播放。

onAudioInfoChanged

音频播放信息回调。

```
void onAudioInfoChanged(int sampleRate, int channels, int bits)
```

参数

参数	类型	含义
sampleRate	int	采样率。
channels	int	声道数。
bits	int	采样点大小。

ITXAudioVolumeEvaluationListener

功能

播放器音量大小回调。

onAudioVolumeEvaluationNotify

播放器音量大小回调。

```
void onAudioVolumeEvaluationNotify(int volume)
```

参数

参数	类型	含义
volume	int	音量大小, 取值范围 [0，100]。

TXLivePlayConfig

最近更新时间：2024-12-11 16:20:42

功能

腾讯云直播播放器的参数配置模块。

介绍

主要负责 V2TXLivePlayer 对应的参数设置，**其中绝大多数设置项在播放开始之后再设置是无效的。**

常用设置项

setAutoAdjustCacheTime

设置是否自动调整缓存时间。

```
void setAutoAdjustCacheTime(boolean bAuto)
```

参数

参数	类型	含义
bAuto	boolean	true：启用； false：关闭。

介绍

接口说明：

- 默认值：true。
- true：启用自动调整， SDK 将根据网络状况在一个范围内调整缓存时间；通过 setMaxAutoAdjustCacheTime 和 setMinAutoAdjustCacheTime 两个接口来进行设置。
- false：关闭自动调整， SDK 将使用固定缓存时长；通过 setCacheTime(float) 来进行设置。

setCacheTime

设置播放器缓存时间。

```
void setCacheTime(float time)
```

参数

参数	类型	含义
time	float	播放器缓存时长。

介绍

接口说明：

- 设置播放器缓存时间，单位为秒，默认值为5秒。
- 不建议设置过大，会影响秒开以及直播流播放的实时性。

setMaxAutoAdjustCacheTime

设置最大的缓存时间。

```
void setMaxAutoAdjustCacheTime(float time)
```

参数

参数	类型	含义
time	float	播放器最大缓存时间。

介绍

接口说明：

- 默认值：5，单位为秒。
- 仅在启用自动调用缓存时间接口时，有效。

setMinAutoAdjustCacheTime

设置最小的缓存时间。

```
void setMinAutoAdjustCacheTime(float time)
```

参数

参数	类型	含义
time	float	播放器最小缓存时间。

介绍

接口说明：

- 默认值：1，单位为秒。
- 仅在启用自动调用缓存时间接口时，有效。

setVideoBlockThreshold

设置播放器视频卡顿报警阈值。

```
void setVideoBlockThreshold(int threshold)
```

参数

参数	类型	含义
threshold	int	播放器视频卡顿报警阈值。

介绍

接口说明：

- 默认值：800，单位为毫秒。
- 当渲染间隔超过此阈值时候，表明产生了卡顿；播放器会通过 ITXLivePlayListener#onPlayEvent(int， Bundle) 回调 PLAY_WARNING_VIDEO_PLAY_LAG 事件通知。

setConnectRetryCount

设置播放器重连次数。

```
void setConnectRetryCount(int count)
```

参数

参数	类型	含义
count	int	SDK 重连次数。

介绍

接口说明：

- 默认值：3；取值范围：1 – 10。
- 当 SDK 与服务器异常断开连接时，SDK 会尝试与服务器重连；您可通过此接口设置重连次数。

setConnectRetryInterval

设置播放器重连间隔。

```
void setConnectRetryInterval(int interval)
```

参数

参数	类型	含义
interval	int	SDK 重连间隔。

介绍

接口说明：

- 默认值：3，单位为秒；取值范围：3 – 30。
- 当 SDK 与服务器异常断开连接时， SDK 会尝试与服务器重连；您可通过此接口设置连续两次重连的时间间隔。

专业设置项

setEnableMessage

开启消息通道。

```
void setEnableMessage(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启，false：关闭。

介绍

此接口在视频帧与消息需要高同步的情况使用，如：直播答题场景。

接口说明：

- 默认值：false。
- 此接口需要搭配 TXLivePusher#sendMessageEx(byte[]) 使用。
- 此接口存在一定的性能开销以及兼容性风险。

enableAEC

设置回声消除。

```
void enableAEC(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启； false：关闭。

介绍

接口说明：

- 默认值为：false。
- 连麦时，麦克风和播放有回音，所以必须开启回声消除。非连麦情况下，建议不开启。

待废弃设置项

setEnableNearestIP

设置就近选路。

```
void setEnableNearestIP(boolean enable)
```

参数

参数	类型	含义
enable	boolean	true：开启；false：关闭。

介绍

待废弃，默认值：true。

只对加速拉流生效，用于指定加速拉流是否开启就近选路。

setRtmpChannelType

设置 RTMP 传输通道的类型。

```
void setRtmpChannelType(int type)
```

参数

参数	类型	含义
type	int	通道类型。

介绍

待废弃，默认值：TXLiveConstants#RTMP_CHANNEL_TYPE_AUTO。

通道类型说明：

- TXLiveConstants#RTMP_CHANNEL_TYPE_AUTO：自动。
- TXLiveConstants#RTMP_CHANNEL_TYPE_STANDARD：标准的 RTMP 协议，网络层采用 TCP 协议。
- TXLiveConstants#RTMP_CHANNEL_TYPE_PRIVATE：标准的 RTMP 协议，网络层采用私有通道传输（在 UDP 上封装的一套可靠快速的传输通道），能够更好地抵抗网络抖动；对于播放来说，私有传输通道只有在拉取低时延加速流时才可以生效。

setHeaders

设置自定义 HTTP Headers。

```
void setHeaders(Map< String, String > headers)
```

参数

参数	类型	含义
headers	Map< String, String >	HTTP 头。

ITXLivePlayListener

最近更新时间：2022-10-08 16:33:52

功能

腾讯云直播播放的回调通知。

onPlayEvent

播放事件通知。

```
void onPlayEvent(final int event, final Bundle param)
```

参数

参数	类型	含义
event	final int	事件 ID，ID 类型请查看代码中的 com.tencent.rtmp.TXLiveConstants. PUSH_EVT_CONNECT_SUCC 播放事件列表。
param	final Bundle	事件相关的参数（key，value）格式，其中 key 请查看代码中的 com.tencent.rtmp.TXLiveConstants.EVT_TIME 事件参数。

onNetStatus

网络状态通知。

```
void onNetStatus(final Bundle status)
```

参数

参数	类型	含义
status	final Bundle	通知的内容（key，value）格式，其中 key 请查看代码中的 com.tencent.rtmp.TXLiveConstants.NET_STATUS_VIDEO_BITRATE 网络状态通知。

错误码表

最近更新时间：2022-10-08 16:33:52

PushEvent

常规事件

每一次成功的推流都会有通知事件，例如收到1003就意味着摄像头的画面开始渲染。

code	事件定义	含义说明
1001	PUSH_EVT_CONNECT_SUCC	已经成功连接到云端服务器
1002	PUSH_EVT_PUSH_BEGIN	与服务器握手完毕，一切正常，准备开始上行推流
1003	PUSH_EVT_OPEN_CAMERA_SUCC	已成功启动摄像头，摄像头被占用或者被限制权限的情况下无法打开
1004	PUSH_EVT_SCREEN_CAPTURE_SUCC	录屏启动成功（Android SDK 专用）
1005	PUSH_EVT_CHANGE_RESOLUTION	推流动态调整分辨率
1006	PUSH_EVT_CHANGE_BITRATE	推流动态调整码率
1007	PUSH_EVT_FIRST_FRAME_AVAILABLE	首帧画面采集完成
1008	PUSH_EVT_START_VIDEO_ENCODER	编码器启动
1009	PUSH_EVT_CAMERA_REMOVED	摄像头设备已被移出（Windows SDK 专用）
1010	PUSH_EVT_CAMERA_AVAILABLE	摄像头设备重新可用（Windows SDK 专用）
1011	PUSH_EVT_CAMERA_CLOSED	关闭摄像头完成（Windows SDK 专用）
1012	PUSH_EVT_SNAPSHOT_RESULT	截图快照返回码（Windows SDK 专用）
1018	PUSH_EVT_ROOM_IN	已经在 WebRTC 房间里面，进房成功后通知
1019	PUSH_EVT_ROOM_OUT	不在 WebRTC 房间里面，进房失败或者中途退出房间时通知
1020	PUSH_EVT_ROOM_USERLIST	下发 WebRTC 房间成员列表（不包括自己）
1021	PUSH_EVT_ROOM_NEED_REENTER	Wi-Fi 切换到 4G 会触发断线重连，此时需要重新进入 WebRTC 房间（拉取最优的服务器地址）
5000	PUSH_WARNING_HANDUP_STOP	小程序被用户挂起，停止推流

严重错误

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

> **注意**
> 视频编码失败并不会直接影响推流，SDK 会做自行处理以保证后面的视频编码成功。

code	事件定义	含义说明
-1301	PUSH_ERR_OPEN_CAMERA_FAIL	打开摄像头失败

-1302	PUSH_ERR_OPEN_MIC_FAIL	打开麦克风失败
-1303	PUSH_ERR_VIDEO_ENCODE_FAIL	视频编码失败
-1304	PUSH_ERR_AUDIO_ENCODE_FAIL	音频编码失败
-1305	PUSH_ERR_UNSUPPORTED_RESOLUTION	不支持的视频分辨率
-1306	PUSH_ERR_UNSUPPORTED_SAMPLERATE	不支持的音频采样率
-1307	PUSH_ERR_NET_DISCONNECT	网络断连，且连续三次无法重新连接，需要自行重启推流
-1308	• PUSH_ERR_CAMERA_OCCUPY • PUSH_ERR_AUDIO_SYSTEM_NOT_WORK • PUSH_ERR_SCREEN_CAPTURE_START_FAILED	• 摄像头正在被占用中，可尝试打开其他摄像头（Windows） • 系统异常，录音失败（iOS） • 开始录屏失败，可能是被用户拒绝了（Android）
-1309	PUSH_ERR_SCREEN_CAPTURE_UNSURPORT	录屏失败，不支持的 Android 系统版本，需要5.0以上的系统（Android SDK 专用）

警告事件

大部分的警告事件会触发一些重试性的保护逻辑或者恢复逻辑，很大概率能够由 SDK 自行恢复。部分警告事件需要由开发者处理。

- **WARNING_NET_BUSY：**主播网络不给力，可以通过 UI 向用户进行提示。
- **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）。

code	事件定义	含义说明
1101	PUSH_WARNING_NET_BUSY	上行网速不够用，建议提示用户改善当前的网络环境
1102	PUSH_WARNING_RECONNECT	网络断连，已启动重连流程（重试失败超过三次会放弃）
1103	PUSH_WARNING_HW_ACCELERATION_FAIL	硬编码启动失败，自动切换到软编码
1104	PUSH_WARNING_VIDEO_ENCODE_FAIL	视频编码失败，非致命错，内部会重启编码器
1107	PUSH_WARNING_VIDEO_ENCODE_SW_SWITCH_HW	由于机器性能问题，自动切换到硬件编码（Android SDK 专用）
3001	PUSH_WARNING_DNS_FAIL	DNS 解析失败，启动重试流程
3002	PUSH_WARNING_SEVER_CONN_FAIL	服务器连接失败，启动重试流程
3003	PUSH_WARNING_SHAKE_FAIL	服务器握手失败，启动重试流程
3004	PUSH_WARNING_SERVER_DISCONNECT	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
3005	PUSH_WARNING_READ_WRITE_FAIL	RTMP 读/写失败，将会断开连接

PlayEvent

关键事件

code	事件定义	含义说明
2001	PLAY_EVT_CONNECT_SUCC	已经连接服务器
2002	PLAY_EVT_RTMP_STREAM_BEGIN	已经连接服务器，开始拉流
2003	PLAY_EVT_RCV_FIRST_I_FRAME	渲染首个视频数据包（IDR）
2004	PLAY_EVT_PLAY_BEGIN	视频播放开始
2005	PLAY_EVT_PLAY_PROGRESS	视频播放进度（点播专用）
2006	PLAY_EVT_PLAY_END	视频播放结束
2007	PLAY_EVT_PLAY_LOADING	视频播放加载中
2008	PLAY_EVT_START_VIDEO_DECODER	解码器启动
2009	PLAY_EVT_CHANGE_RESOLUTION	视频分辨率改变
2010	• PLAY_EVT_GET_PLAYINFO_SUCC • PLAY_EVT_SNAPSHOT_RESULT	获取点播文件信息成功（Android iOS） 系截图快照返回码（Windows）
2011	PLAY_EVT_CHANGE_ROTATION	MP4 视频旋转角度（Android、iOS）
2012	PLAY_EVT_GET_MESSAGE	用于接收夹在音视频流中的消息，详情参考 iOS 消息接收 、Android 消息接收
2013	PLAY_EVT_PREPARED	视频加载完毕（Android、iOS）
2014	PLAY_EVT_VOD_LOADING_END	加载结束（Android、iOS）
-2301	PLAY_ERR_NET_DISCONNECT	网络断连，且连续三次无法重新连接，需要自行重启推流
-2302	PLAY_ERR_GET_RTMP_ACC_URL_FAIL	获取加速拉流失败，这是由于您传给 `liveplayer` 的加速流地址中没有携带 `txTime` 和 `txSecret` 签名，或者是签名计算的不对。出现这个错误时，`liveplayer` 会放弃拉取加速流转而拉取 CDN 上的视频流，从而导致延迟很大。 需要注意：只有 RTMP 协议的播放地址才支持低延时加速。
-2303	PLAY_ERR_FILE_NOT_FOUND	播放文件不存在（Android、iOS）
-2304	PLAY_ERR_HEVC_DECODE_FAIL	H265 解码失败（Android、iOS）
-2305	PLAY_ERR_HLS_KEY	HLS 解码 key 获取失败（Android、iOS）
-2306	PLAY_ERR_GET_PLAYINFO_FAIL	获取点播文件信息失败（Android、iOS）

注意

- **判断直播是否结束：**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都要判断，用来作为直播结束的判定事件。
- **不要在收到 PLAY_LOADING 后隐藏播放画面：**因为 PLAY_LOADING 到 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加载动画。

警告事件

内部警告并非不可恢复的错误，SDK 会启动相应的恢复措施，警告的目的主要用于提示开发者或者最终用户。

code	事件定义	含义说明
2101	PLAY_WARNING_VIDEO_DECODE_FAIL	当前视频帧解码失败
2102	PLAY_WARNING_AUDIO_DECODE_FAIL	当前音频帧解码失败
2103	PLAY_WARNING_RECONNECT	网络断连，已启动自动重连恢复（重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了）
2104	PLAY_WARNING_RECV_DATA_LAG	视频流不太稳定，可能是观看者当前网速不充裕
2105	PLAY_WARNING_VIDEO_PLAY_LAG	当前视频播放出现卡顿
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解
2107	PLAY_WARNING_VIDEO_DISCONTINUITY	当前视频帧不连续，视频源可能有丢帧，可能会导致画面花屏
2108	PLAY_WARNING_FIRST_IDR_HW_DECODE_FAIL	当前流硬解第一个 I 帧失败，SDK 自动切软解
3001	PLAY_WARNING_DNS_FAIL	DNS 解析失败（仅播放 RTMP:// 地址时会抛送）
3002	PLAY_WARNING_SEVER_CONN_FAIL	服务器连接失败（仅播放 RTMP:// 地址时会抛送）
3003	PLAY_WARNING_SHAKE_FAIL	服务器握手失败（仅播放 RTMP:// 地址时会抛送）
3004	PLAY_WARNING_SERVER_DISCONNECT	RTMP 服务器主动断开