# 移动直播 SDK

# 基础功能

# 产品文档

腾讯云

# 文档目录

# 基础功能
# SDK 集成
# iOS

最近更新时间：2021-07-16 14:50:22

本文主要介绍如何快速地将腾讯云移动直播 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 移动直播专业版 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
```

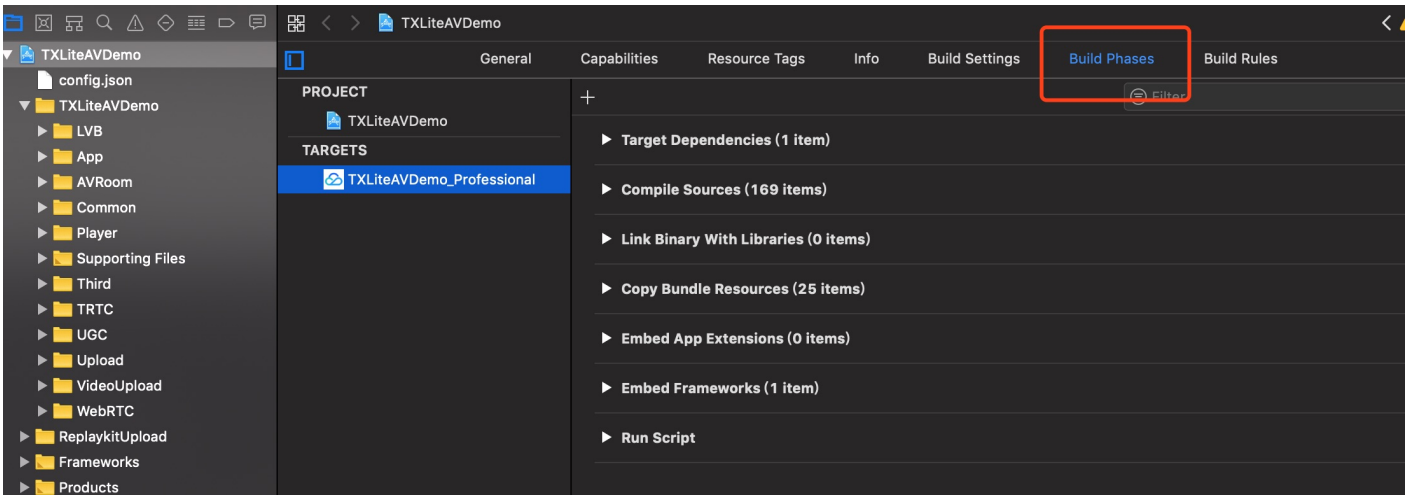
或使用以下命令更新本地库版本：

```
pod update
```

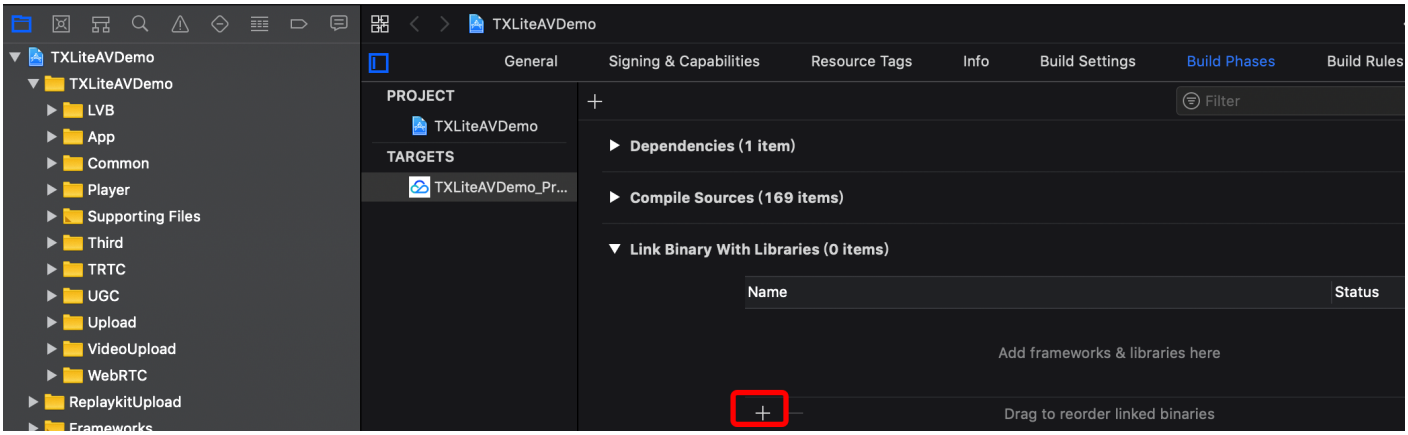
pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

### 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。

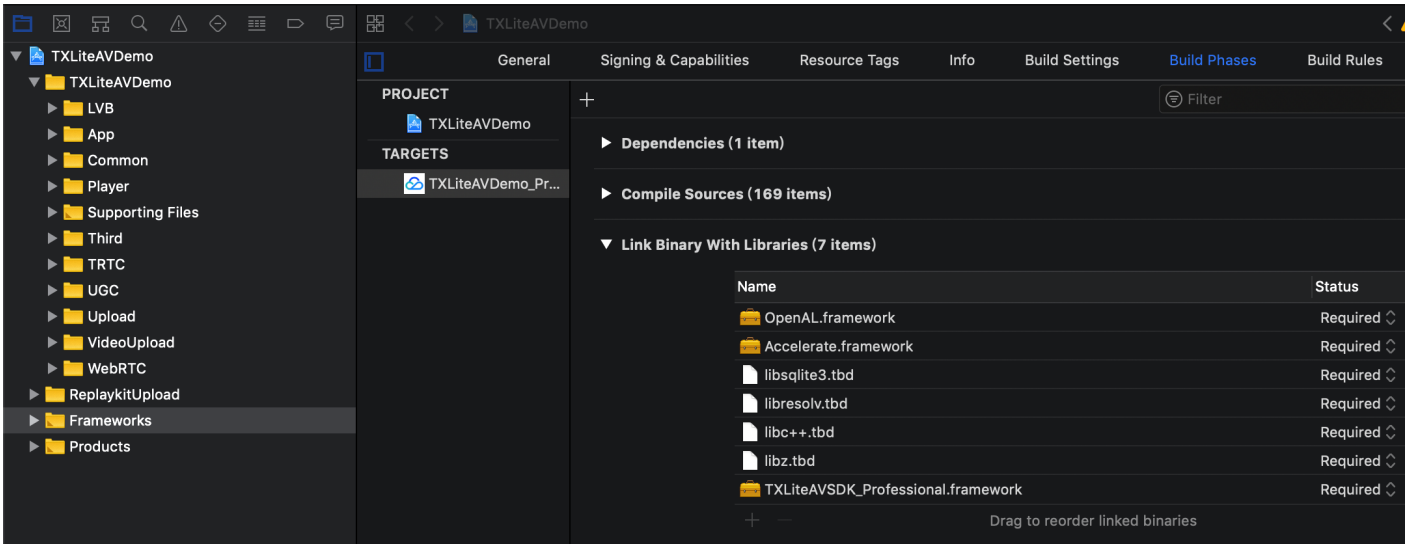



3. 单击 **Link Binary with Libraries** 项展开，单击底下的【+】添加依赖库。

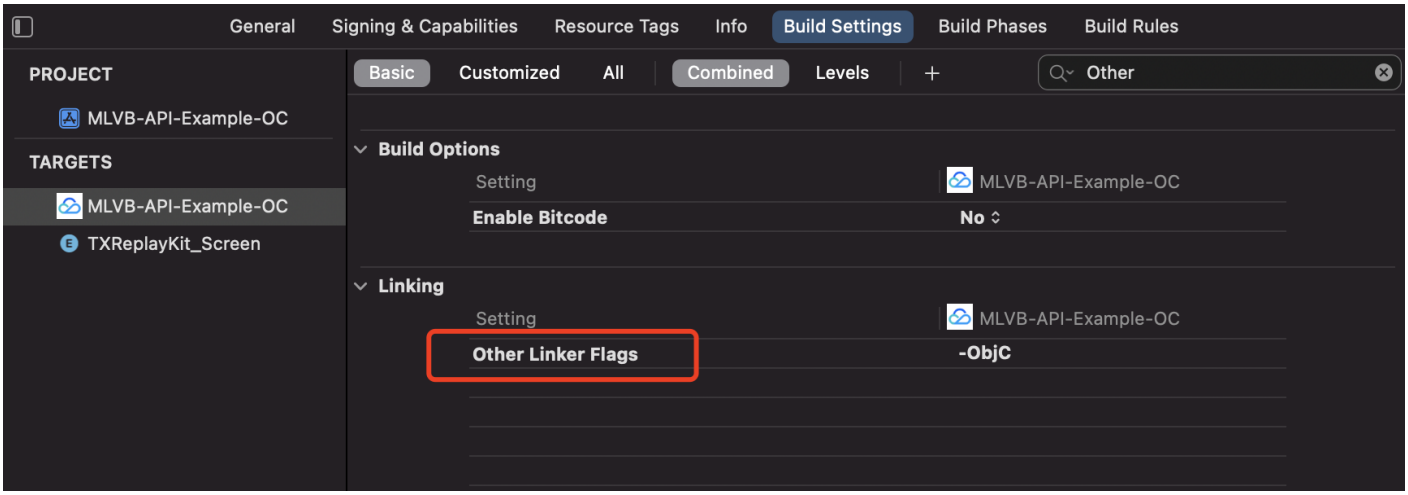



4. 依次添加所下载的 `TXLiteAVSDK_Professional.framework` 及其所需依赖库：

```
libz.tbd
libc++.tbd
libresolv.tbd
libsqlite3.tbd
Accelerate.framework
OpenAL.framework
```



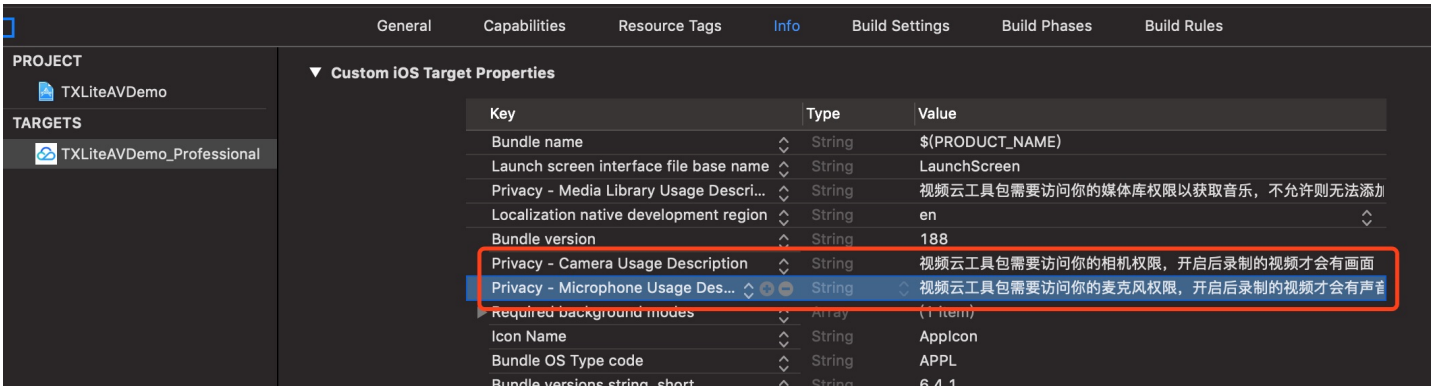

5. 选中 Build Settings 项，搜索 `Other Linker Flags`。添加 `-ObjC`。




## 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。

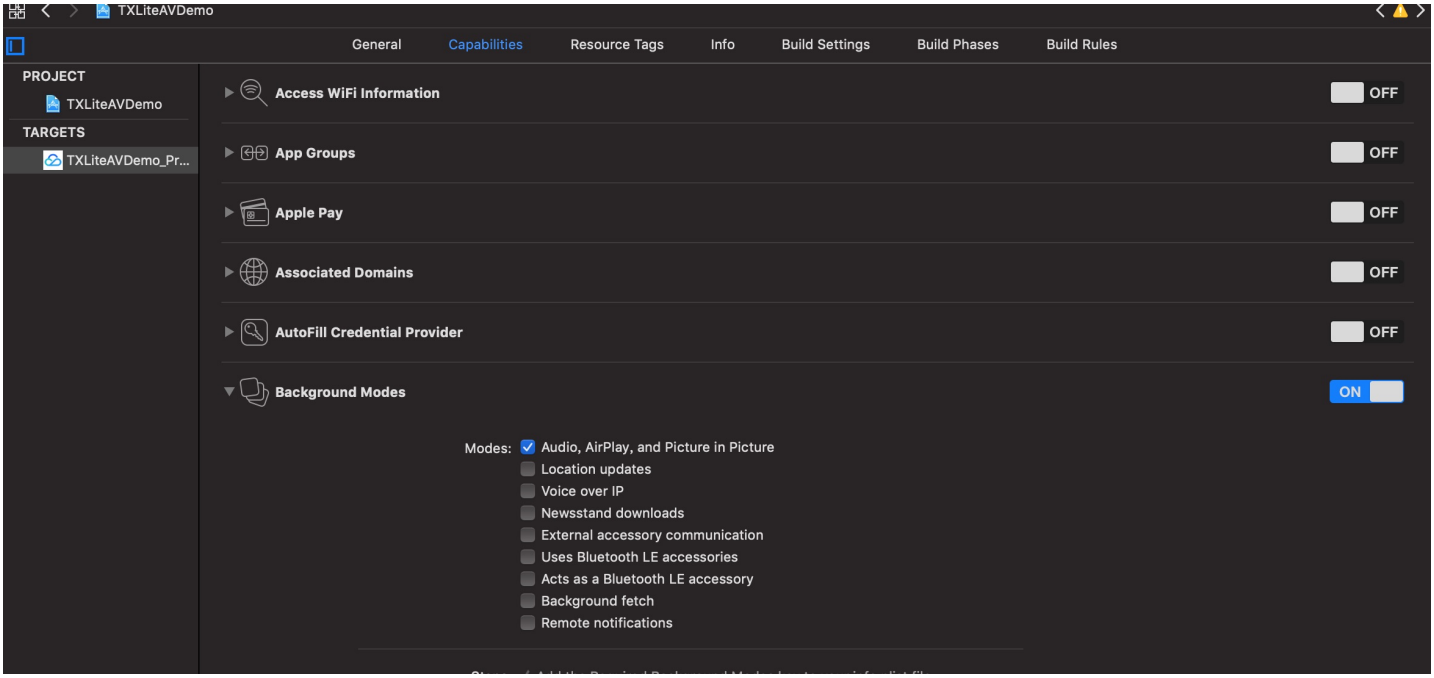
在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOption
s {
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：


TXLiteAVDemo
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
TXLiteAVDemo
TARGETS
TXLiteAVDemo_Pr...
Access WiFi Information
OFF
App Groups
OFF
Apple Pay
OFF
Associated Domains
OFF
AutoFill Credential Provider
OFF
Background Modes
ON
Modes:
Audio, AirPlay, and Picture in Picture
Location updates
Voice over IP
Newsstand downloads
External accessory communication
Uses Bluetooth LE accessories
Acts as a Bluetooth LE accessory
Background fetch
Remote notifications

# Android

最近更新时间：2021-07-23 18:43:27

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

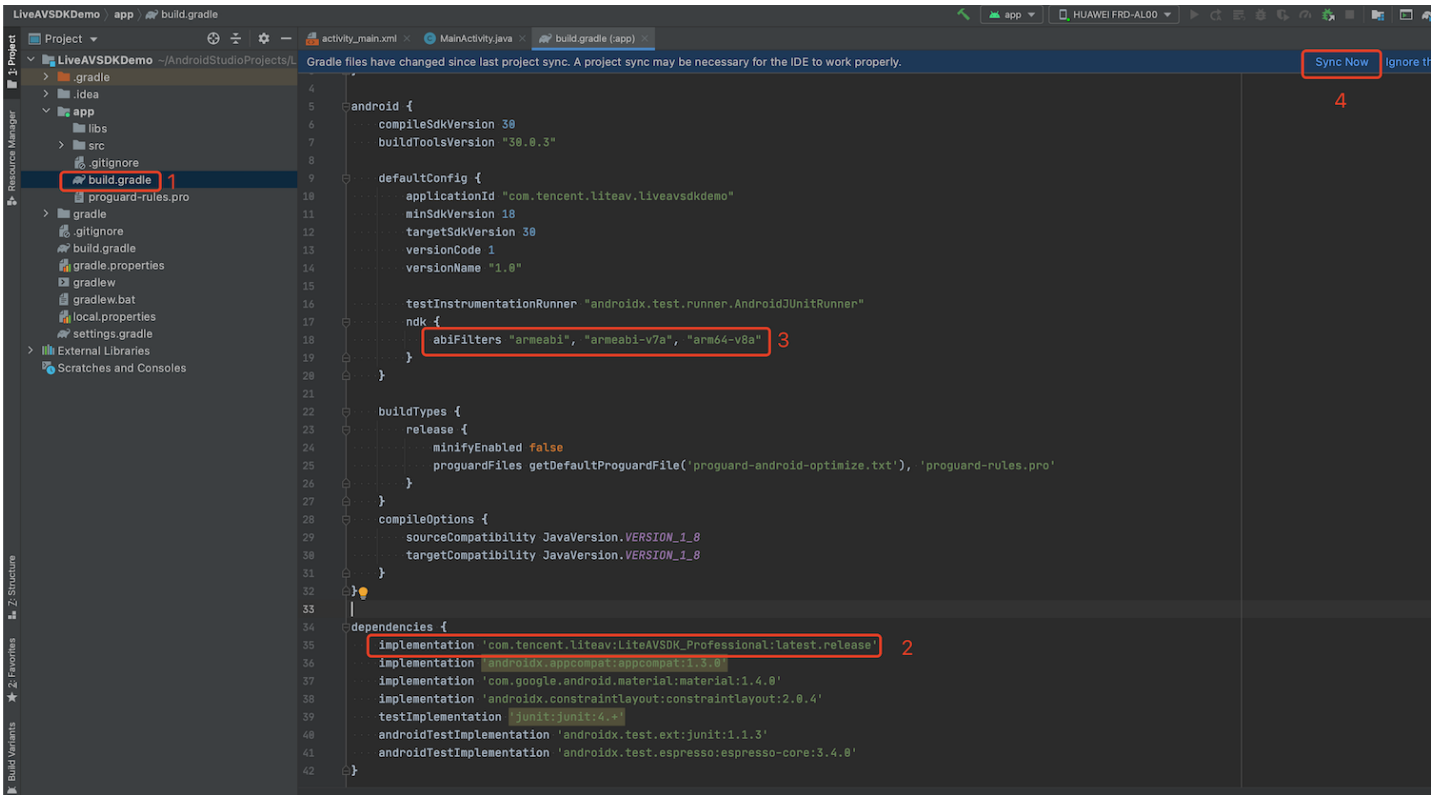
- Android Studio 2.0+。
- Android 4.1（SDK API 16）及以上系统。

## 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）

因为 jcenter 已经下线，您可以通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK。
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 `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_Professional:latest.release'
}
```

或

```
dependencies {
implementation 'com.tencent.liteav:LiteAVSDK_Professional: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）。

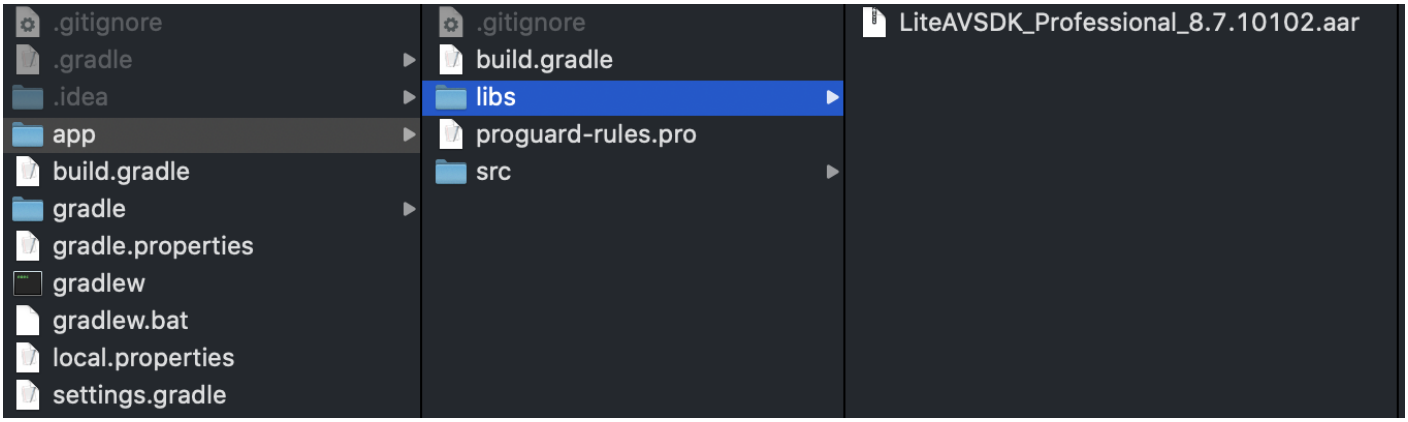
```
defaultConfig {
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
```

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

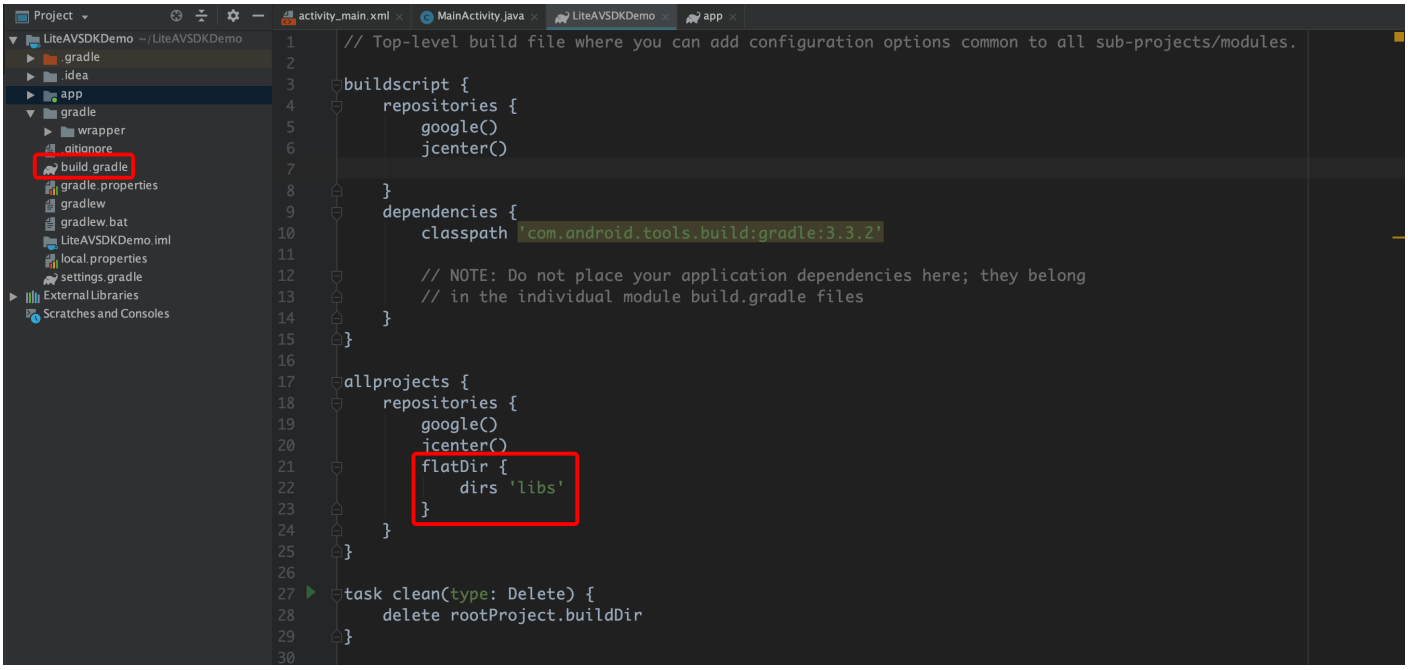
### 方法二：手动下载（aar）

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

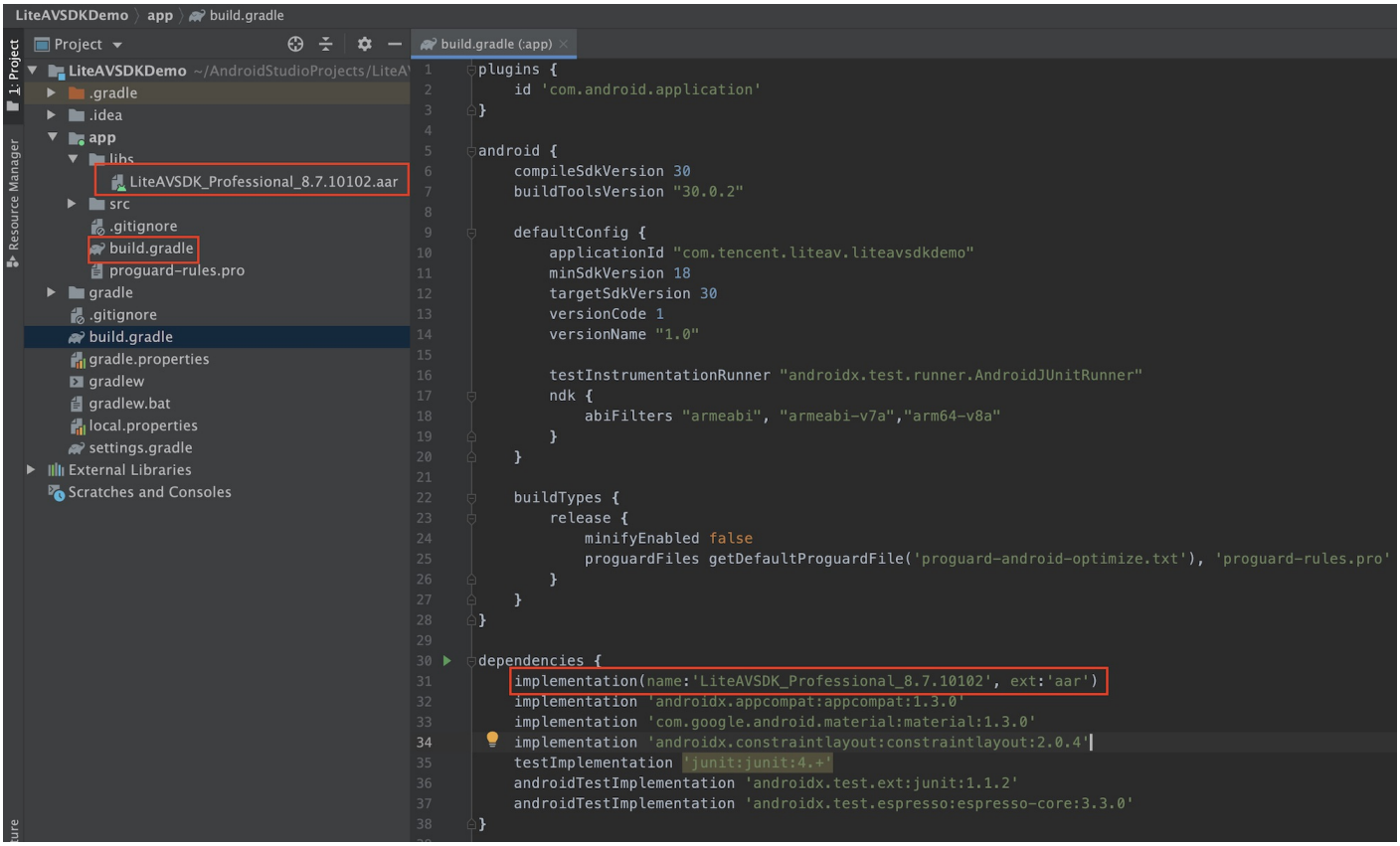
1. 下载 LiveAVSDK ，下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：



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_Professional_8.7.10102', 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 {
```

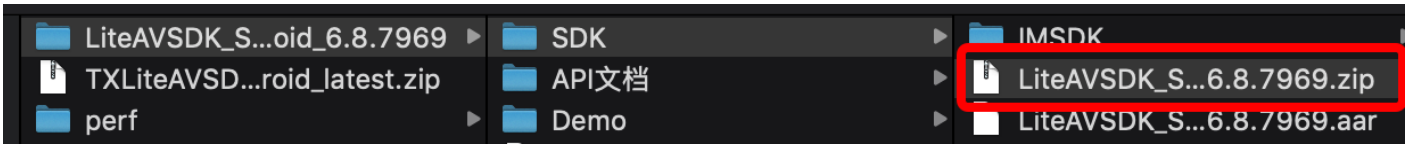
```
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
```

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

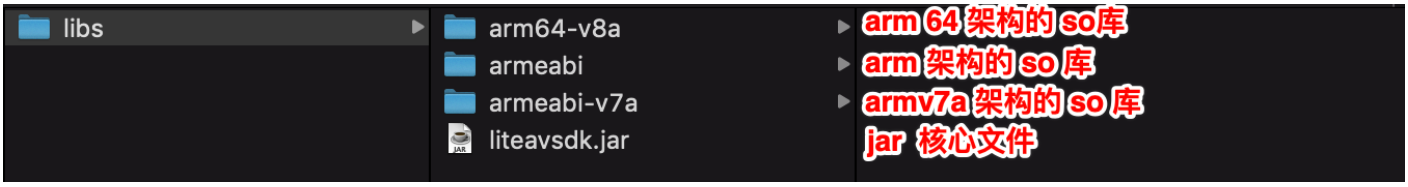
## 集成 SDK（jar）

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：

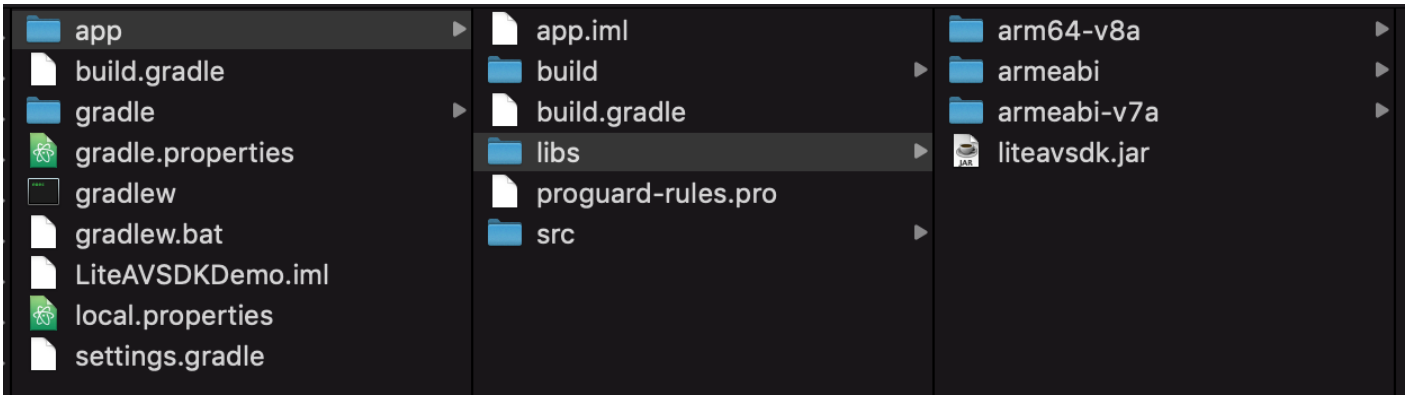
1. 下载 LiveAVSDK ，下载完成后进行解压。在 SDK 目录下找到 `LiteAVSDK_Professional_xxx.zip`（其中 xxx 为 LiteAVSDK 的版本号）：



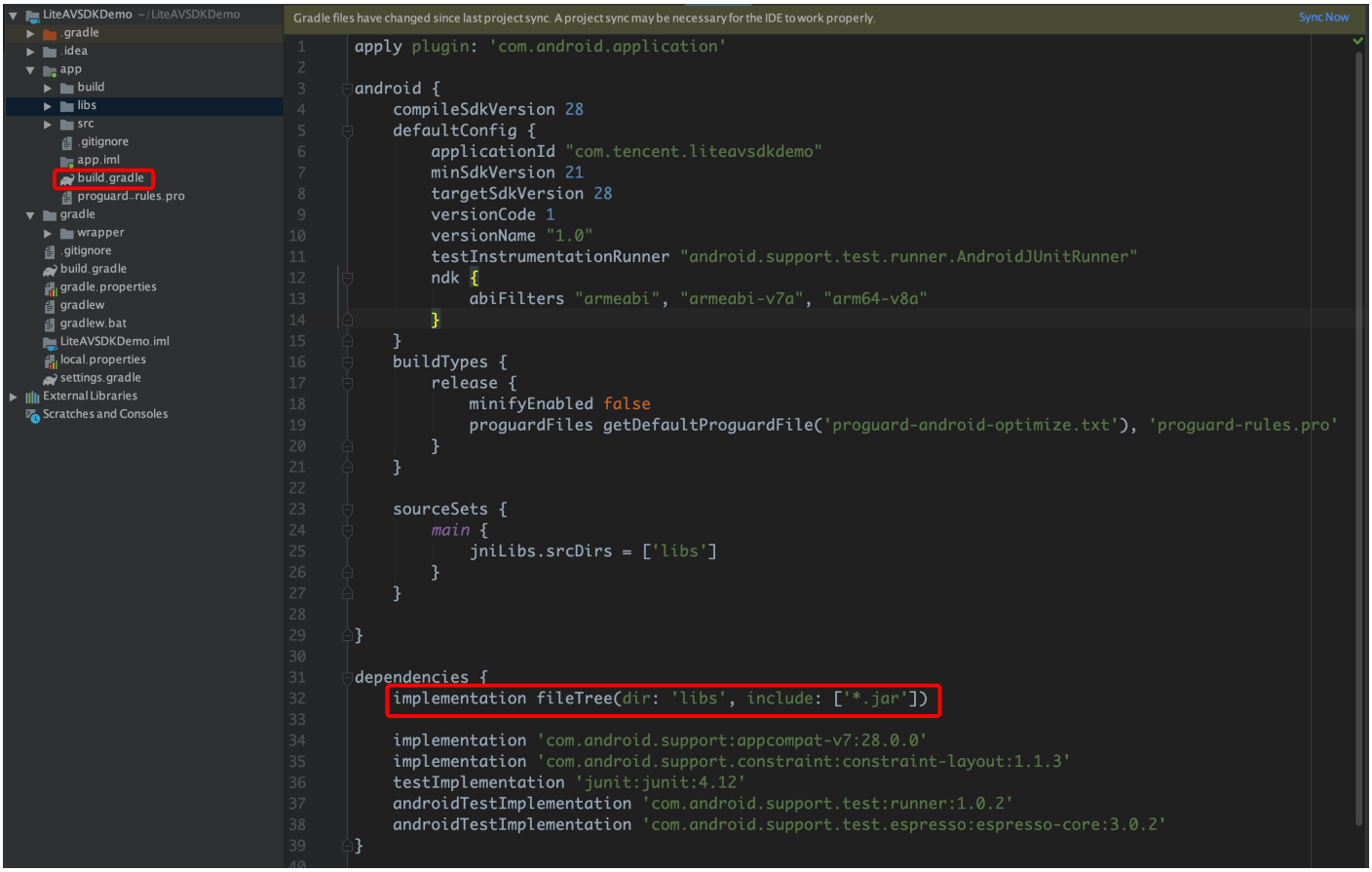

解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：




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。

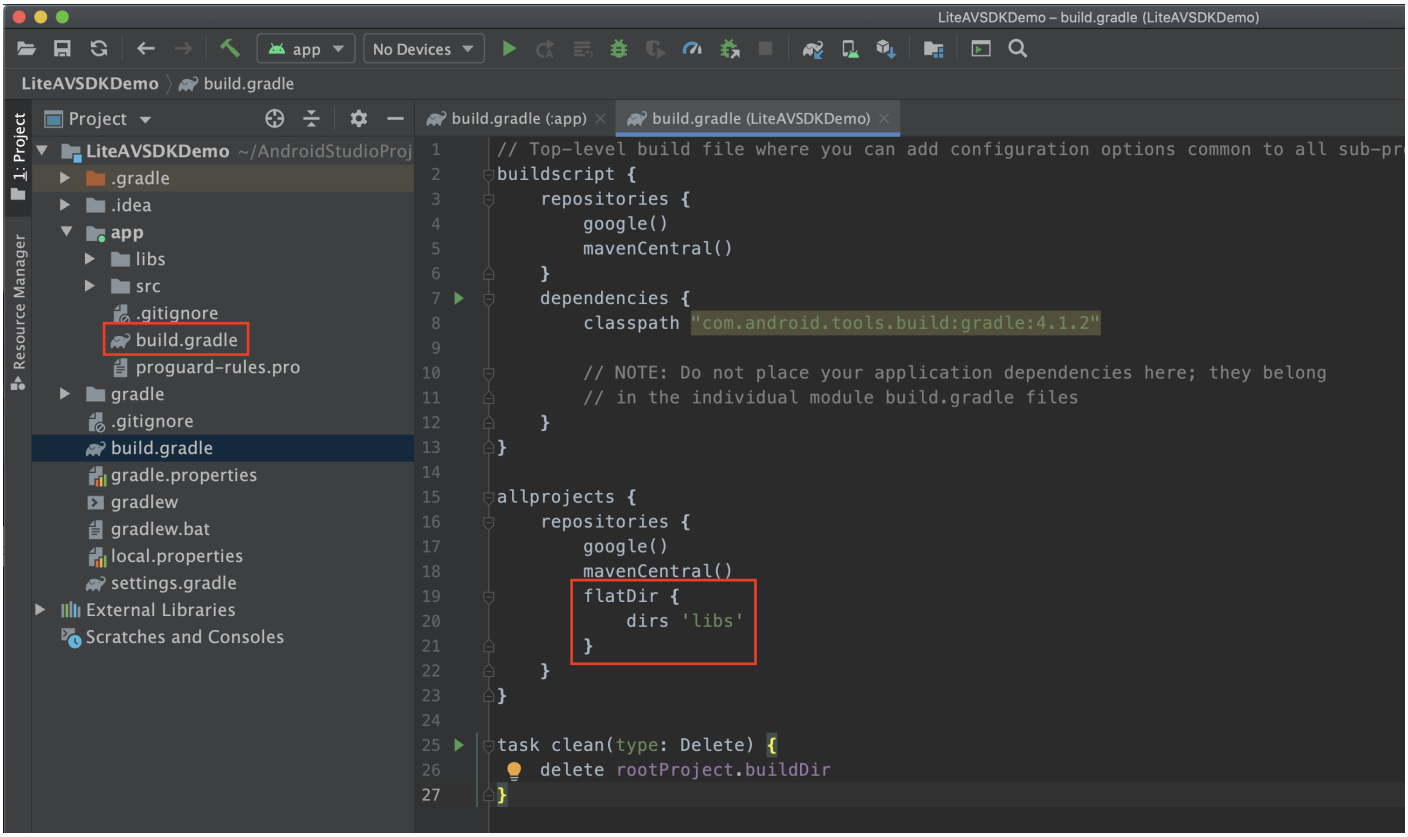


3. 在 app/build.gradle 中，添加引用 jar 库的代码。

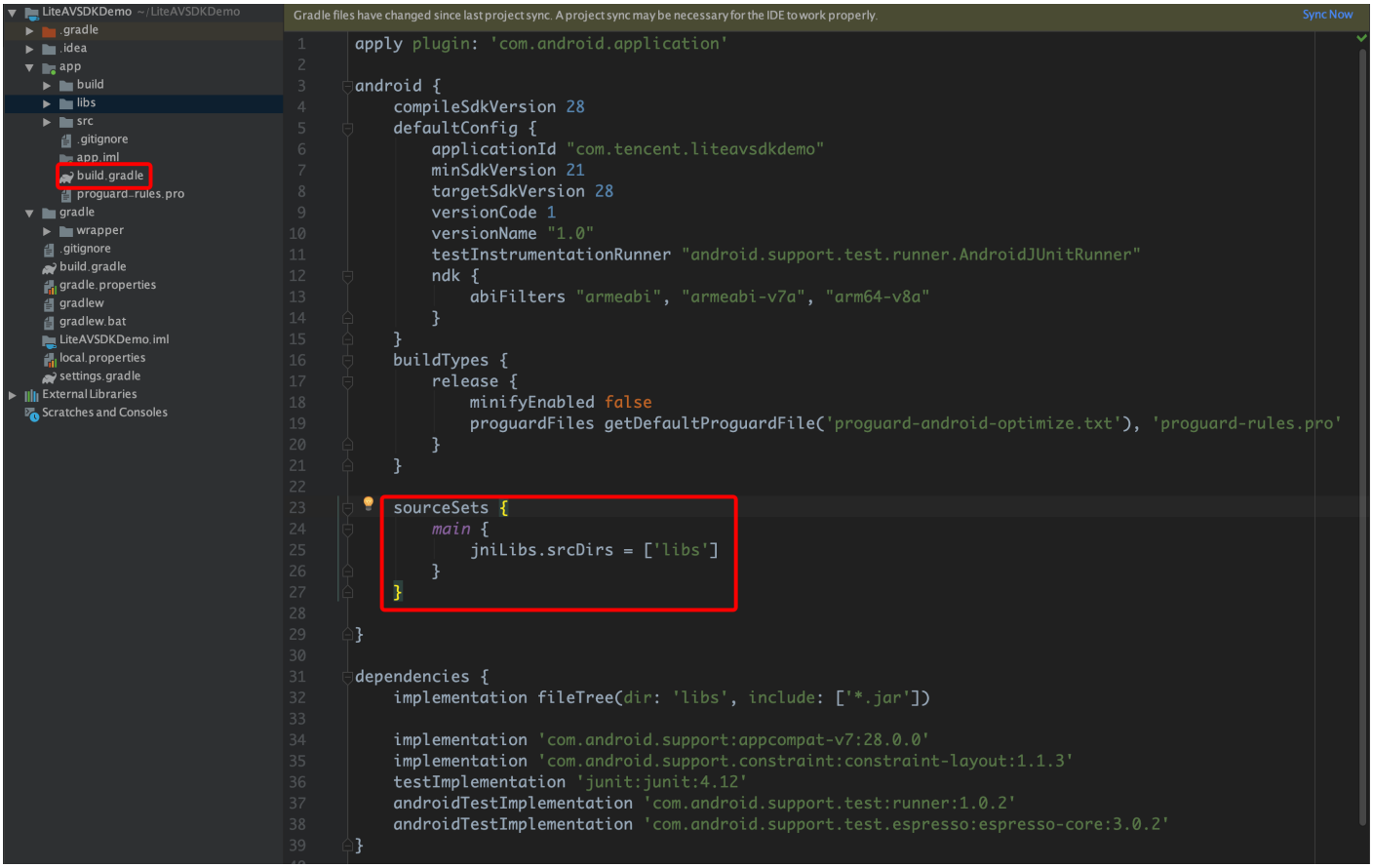



```
dependencies{
implementation fileTree(dir:'libs',include:['*.jar'])
}
```

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。

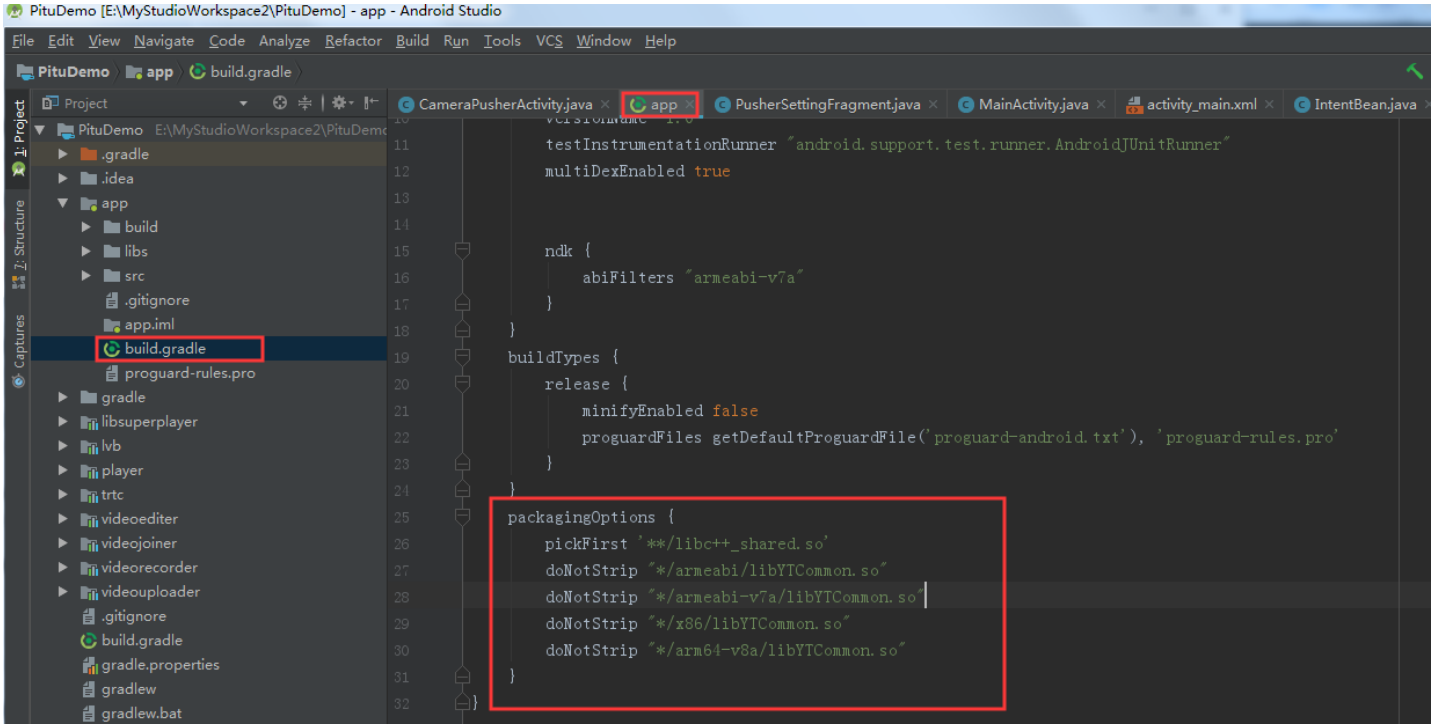



6. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构(目前 LiteAVSDK 支持 armeabi 、 armeabi-v7a 和 arm64-v8a) 。

```
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);
}
}
```

## 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.** { *; }
```

# 微信小程序

最近更新时间：2021-07-26 11:42:56

以下视频将为您讲解移动直播 SDK 如何快速集成到微信小程序：

点击查看视频

## 一、注册小程序

注册小程序请单击 微信公众平台 ，完成注册后，在小程序管理页面的【开发】>【基本配置】中记录下小程序 AppID 供后面使用。

> ⚠ 注意：
> 必须以**非个人主体**类型进行注册，否则无法开通 <live-pusher> 和 <live-player> 这两个标签。

## 二、开通标签使用权限

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 目前支持这两个标签的类目如下表格所示（只有非个人主体才有以下类目）：

| 一级类目/主体类型 | 二级类目 | 资质要求 | 类目适用范围 | 小程序直播内容场景 |
| --- | --- | --- | --- | --- |
| 社交 | 直播 | （3选1）：<br>1.《信息网络传播视听节目许可证》<br>2.《网络文化经营许可证》(经营范围含网络表演）<br>3.《统一社会信用代码》及《情况说明函件》（适用于政府主体） | 适用于提供在线直播等服务<br>注：<br>1.如提供时政信息服务，需补充：时政信息类目<br>2.选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 | 涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 |
| 教育 | 在线视频课程 | （5选1）：<br>1.《事业单位法人证书》（适用公立学校）<br>2. 区、县级教育部门颁发的《民办学校办学许可证》（适用培训机构）<br>3.《信息网络传播视听节目许可证》<br>4. 全国校外线上培训管理服务平台备案<br>5. 教育部门的批准文件 | 适用于教育行业提供，网课、在线培训、讲座等教育类视频/直播等服务 | 网课、在线培训、讲座等教育类直播 |
| 医疗 | 互联网医院 | （2选1）：<br>1. 卫生健康部门的《设置医疗机构批准书》；<br>2. 合作医院的《医疗机构执业许可证》与执业登记机关的审核合格文件 | 适用于互联网医院主体/医疗服务平台提供在线看诊、疾病咨询等线上医疗服务 | 问诊、大型健康讲座等直播 |
| | 公立医疗机构 | 《医疗机构执业许可证》与《事业单位法人证书》 | 适用于公立医疗机构提供的就医、健康咨询/问诊、医疗保健信息等服务 | |
| 金融 | 银行 | （2选1）：<br>1.《金融许可证》<br>2.《金融机构许可证》 | 适用于提供银行业务在线服务或交易等服务 | 金融产品视频客服理赔、金融产品推广直播等 |
| | 信托 | （2选1）：<br>1.《金融许可证》 | 适用于提供信托理财业务在线服务或交易等服务 | |

| | | | | |
|---|---|---|---|---|
| | | 2.《金融机构许可证》 | | |
| | 公募基金 | （3选1）：<br>1.《经营证券期货业务许可证》且业务范围必须包含“基金”<br>2.《基金托管业务许可证》<br>3.《基金销售业务资格证书》 | 适用于基金管理公司从事股票、债券等金融工具的投资服务 | |
| | 私募基金 | （2选1）：<br>1.《私募基金备案证明》<br>2.《私募投资基金管理人登记证书》 | 仅适用于私募基金展示、介绍、咨询等服务<br>注：暂不支持涉及私募产品公开募集或在线交易等服务 | |
| | 证券/期货 | 《经营证券期货业务许可证》 | 适用于提供证券资讯、证券咨询、证券期货经营等的在线服务 | |
| | 证券、期货投资咨询 | （2选1）：<br>1.《证券投资咨询业务资格证书》<br>2.《经营证券期货业务许可证》 | 适用于提供证券、期货投资等在线咨询服务 | |
| | 保险 | （8选1）：<br>1.《保险公司法人许可证》<br>2.《经营保险业务许可证》<br>3.《保险营销服务许可证》<br>4.《经营保险代理业务许可证》<br>5.《经营保险经纪业务许可证》<br>6.《经营保险公估业务许可证》<br>7.《经营保险资产管理业务许可证》<br>8.《保险兼业代理业务许可证》 | 适用于提供保险业务在线服务或交易等服务 | |
| | 征信业务 | （2选1）：<br>1.经营个人征信业务：《个人征信业务经营许可证》、《营业执照》<br>2.经营企业征信业务：经所在地的中国人民银行及其派出机构备案的《企业征信业务经营备案证》、《营业执照》 | 适用于银行或征信机构提供征信业务服务，包括：信贷记录、逾期记录、失信人查询等 | |
| | 新三板信息服务平台 | 全国中小企业股份转让系统有限责任公司的书面许可与《非经营性互联网信息服务备案核准》 | 适用于提供新三板信息行情资讯等服务 | |
| | 股票信息服务平台（港股/美股） | 《非经营性互联网信息服务备案核准》 | 适用于提供港股、美股行情资讯、行情分析等服务<br>注：如提供股票交易服务，需补充：金融业-证券/期货类目 | |
| | 消费金融 | 银监会核准开业的审批文件与《金融许可证》与《营业执照》 | 适用于提供消费金融线上服务或交易等服务 | |
| 汽车 | 汽车预售服务 | （3选1）：<br>1.汽车厂商：《营业执照》与《工信部道路机动车辆生产企业准入许可》<br>2.汽车经销商/4s店：《营业执照》与《厂商授权销售文件》与《工信部道路机动车辆生产企业准入许可》<br>3.下属子/分公司：《营业执照》与《工信部道路机动车辆生产企业准入许可》与《股权关系证明函》（含双方盖章） | 适用于提供汽车在线预付款等服务<br>注：平台暂不支持在线整车销售，如涉及整车销售服务，建议改为价格指导或移除相关功能 | 汽车预售、推广直播 |
| 政府主体 | - | - | - | 政府相关工作推广直 |

| | | | | |
|---|---|---|---|---|
| 帐号 | | | | 播、领导讲话直播等 |
| 工具 | 视频客服 | - | 适用于提供企业售后客服一对一视频等服务 | 不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等 |

> 说明：
> 可申请直播标签的小程序类目以 微信文档 说明为主，小程序类目的资质要求详见 非个人主体类目申请。

- 符合类目要求的小程序，需要在小程序管理后台的【开发】>【接口设置】中自助开通推拉流标签的使用权限，如下图所示：


> 说明：
> 如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

## 三、开通云直播服务

**1. 申请开通视频直播服务**

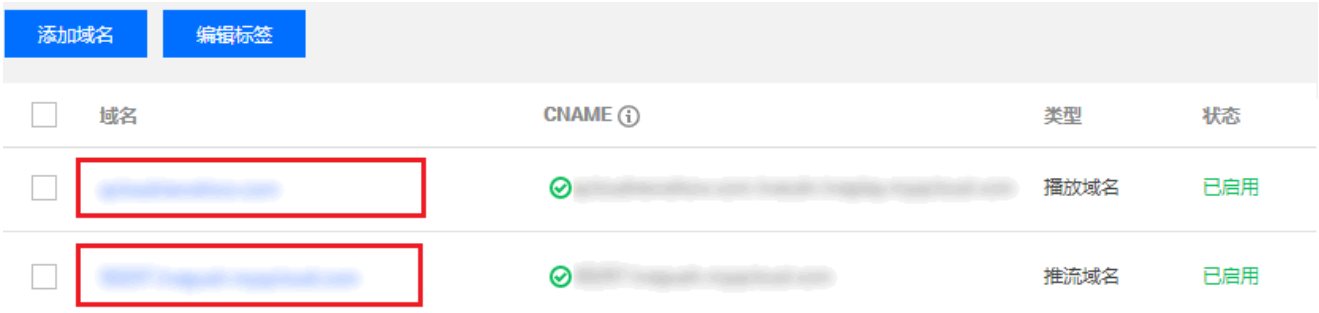
进入 云直播管理控制台 开通云直播服务。

**2. 添加自有域名**

- 进入【域名注册】进行 域名购买。您也可以通过其他域名服务商购买域名。
- 使用腾讯云的 域名备案 对已有域名进行备案。您也可以在其他域名服务商那进行备案。

> 注意：
> 根据国家工信部规定，域名必须进行备案，且备案时长需几个工作日，建议您提前进行备案，更多网站备案信息请参见 网站备案。新备案成功的域名需要1天左右的时间同步到腾讯云服务器，添加该类域名时可能会显示域名未备案。

- 选择云直播控制台的【域名管理】>【添加域名】添加您已备案后的推流域名和播放域名。




> 说明：
>
> - 添加成功后，云直播会生成一个对应的 CNAME 域名，您可通过域名管理的域名列表进行查看。
> - 域名列表里面有一个xxxx.livepush.myqcloud.com的推流域名，这个是我们为您提供的测试域名，可以通过这个域名进行推流测试，但强烈不建议您在正式的业务中使用这个域名作为推流域名。

### 3. 域名 CNAME

在您添加域名成功后，您的域名需要指向腾讯云直播的云服务集群。根据域名列表中的提示，您需要在您注册的域名服务商处将域名解析地址 CNAME 到云直播控制台的域名列表中对应域名的 CNAME 地址，CNAME 添加方法请参见 CNAME 配置。

> 注意：
>
> - CNAME 成功后通常需要一定时间生效，CNAME 不成功是无法使用腾讯云直播服务的。
> - 域名 CNAME 成功后，在云直播控制台的 域名管理 列表中可见域名 CNAME 地址状态符号会由 变成 。
> - 如果 CNAME 操作后，检测始终不成功，建议您向您的域名注册服务商咨询。

## 四、开通即时通信 IM 服务

进入 即时通信 IM 管理控制台，如果还没有开通服务，单击【立即开始】即可。如果是新认证的腾讯云账号，则即时通信 IM 的应用列表是空的。更多详情请参见 一分钟跑通 Demo。


## 五、开通房间管理服务

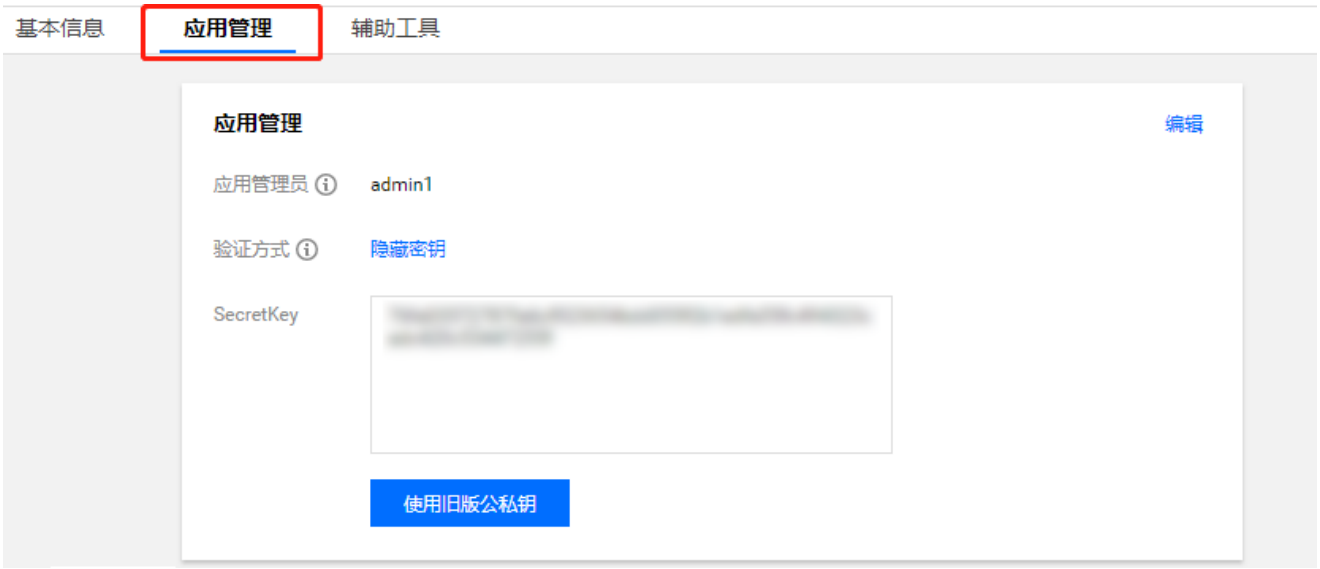
### 1. 创建应用

进入云直播控制台的【直播SDK】>【应用管理】页面，单击【创建应用】填写应用信息。完成创建后，您将会在应用列表中看到您创建的应用，记录其 SDKAppID 信息，如下图所示：


> **说明：**
> 该操作的目的是创建一个即时通信 IM 应用，并将当前云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。

**2. 查看并拷贝加密密钥**

如果在 创建应用 中已经成功创建了应用，单击操作栏的【管理】，选择【应用管理】，并单击【查看密钥】，即可获取加密密钥。




**3. 购买相关资源包**

小程序源码中的 <mlvb-live-room> 组件 为开发者提供了腾讯云直播连麦的能力，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**、**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。

## 六、下载 Demo 源码

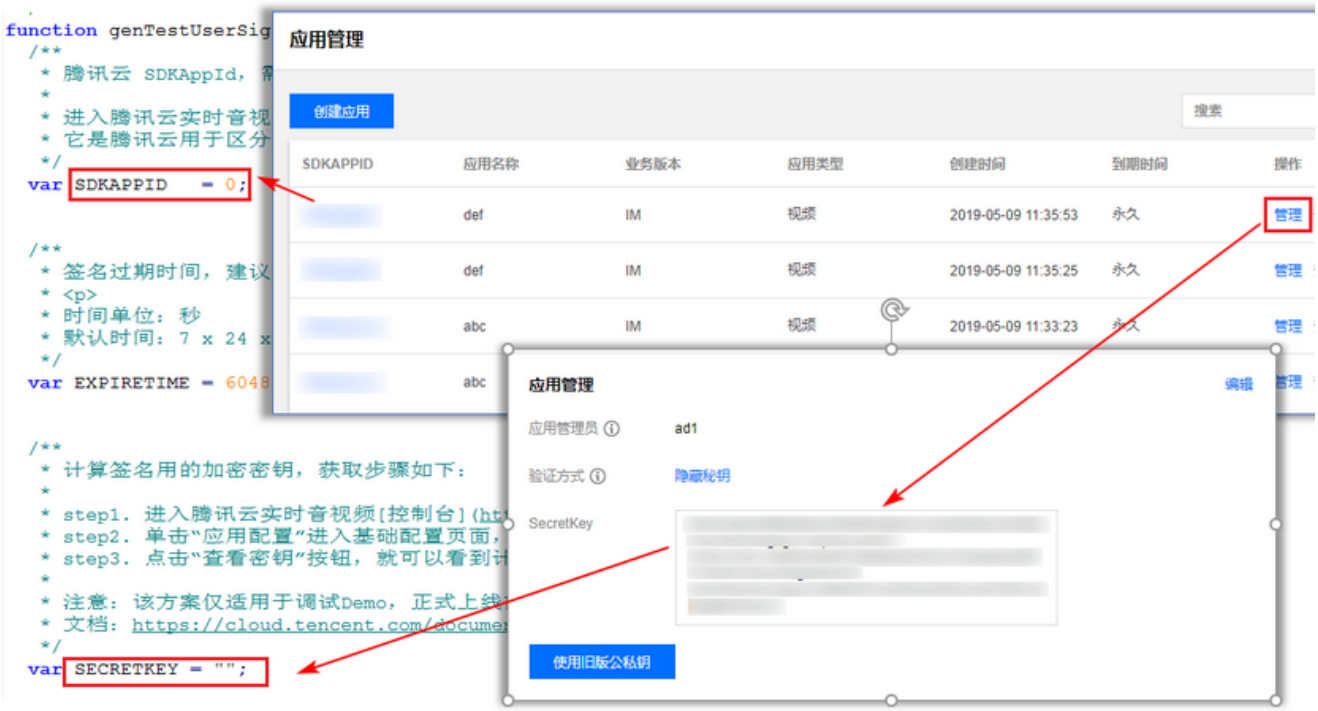
访问 Github，获取小程序 Demo 源码。

## 七、粘贴密钥到 Demo 工程的指定文件中

我们在各个平台的 Demo 的源码工程中都提供了一个叫做 GenerateTestUserSig 的文件，它可以通过 HMAC-SHA256 算法本地计算出 UserSig，用于快速跑通 Demo。

| 语言版本 | 适用平台 | GenerateTestUserSig 的源码位置 |
| --- | --- | --- |
| Objective-C | iOS | Github |
| Java | Android | Github |
| Javascript | 小程序 | Github |

您只需要将 步骤5 中获得的 SDKAppID 和加密密钥拷贝到文件中的指定位置即可，如下所示：




> ⚠ 注意：
>
> **安全警告**：本地计算 UserSig 的做法虽然能够工作，但仅适合于调试 Demo 的场景，不适用于线上产品。
>
> 这是因为客户端代码中的 SECRETKEY 很容易被反编译逆向破解，尤其是 Web 端的代码被破解的难度几乎为零。一旦您的密钥泄露，攻击者就可以计算出正确的 UserSig 来盗用您的腾讯云流量。
>
> **安全方案**：将 UserSig 的计算代码和加密密钥放在您的业务服务器上，然后由 App 按需向您的服务器获取实时算出的 UserSig。由于攻破服务器的成本要远高于破解客户端 App，所以服务器计算的方案能够更好地保护您的加密密钥。

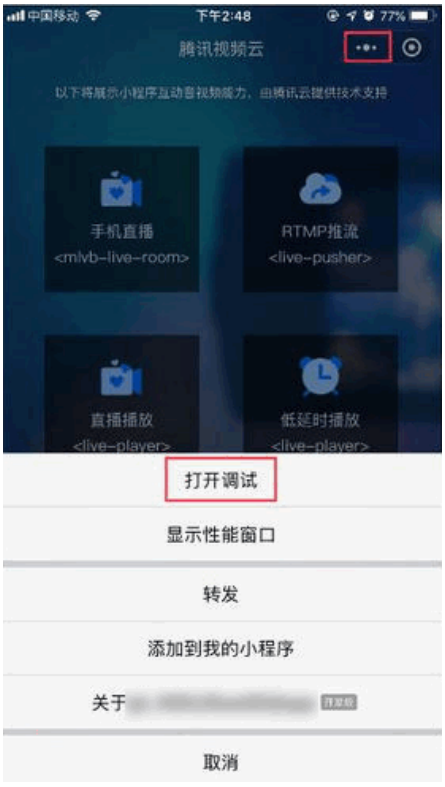
## 八、编译运行

1. 打开 微信开发者工具 后，选择【小程序】菜单栏，然后单击新建图标，选择【导入项目】。
2. 输入小程序 AppID（注意：不是上面的 SDKAppID），单击【导入】。

3. 单击【预览】，生成二维码，通过手机微信扫码二维码即可进入小程序。


**注意：**

- 小程序 <live-player> 和 <live-pusher> 标签需要在手机微信上才能使用，微信开发者工具上无法使用。
- 为了小程序能够使用腾讯云房间管理服务，您需要在手机微信上开启调试功能：手机微信扫码二维码后，单击右上角【...】>【打开调试】。

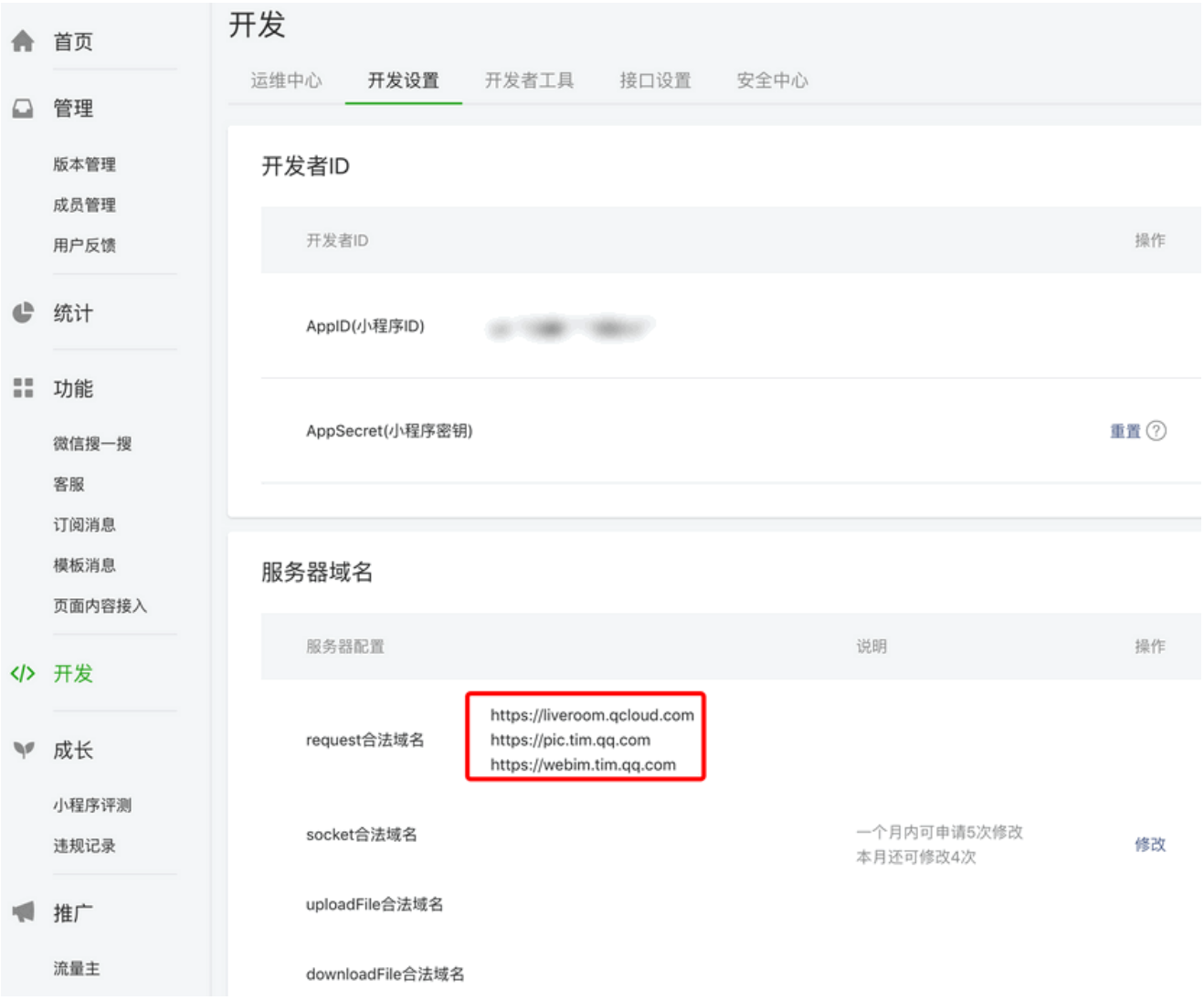



## 九、发布上线

关于小程序的发布流程，请参见 发布上线。

在小程序发布上线前，请务必要在微信小程序控制台的【开发】>【开发设置】>【服务器域名】中配置 “request 合法域名”，否则将无法使用腾讯云的房间管理服务。需要配置的域名包括：

| 域名 | 说明 |
| --- | --- |
| https://liveroom.qcloud.com | MLVB 移动直播房间管理服务域名 |
| https://webim.tim.qq.com | IM 域名 |
| https://pic.tim.qq.com | IM 图片上传域名 |




## 常见问题

### 1. 开发和运行环境要求

- 微信 App iOS 最低版本要求：6.5.21。
- 微信 App Android 最低版本要求：6.5.19。
- 小程序基础库最低版本要求：1.7.0。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。

### 2. 调试时为什么要开启调试模式?

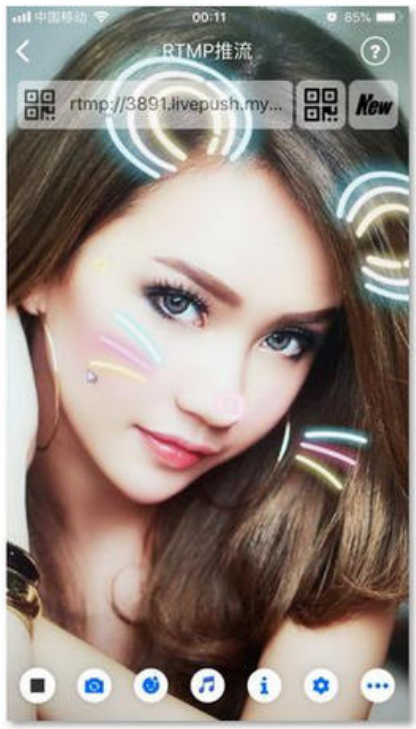
开启调试可以跳过把这些域名加入小程序白名单的工作，否则可能会遇到登录失败，通话无法连接的问题。

# 摄像头推流
# iOS

最近更新时间：2021-08-04 09:56:27

## 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 V2TXLivePusher 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

**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 | CameraPushMainActivity.java |

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

## 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

单击 License 申请 获取测试用的 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 *)launchOption
s {
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
```

### 3. 初始化 V2TXLivePusher 组件

创建一个 V2TXLivePusher 对象，需要指定对应的 V2TXLiveMode 。

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP]; // 指定对应的直播协议为
RTMP
```

### 4. 开启摄像头预览

想要开启摄像头预览，首先需要调用 V2TXLivePusher 中的 setRenderView 接口，改接口需要设置一个用于显示视频画面的 view 对象，然后调用 startCamera 接口开启当前手机摄像头的预览。

```
//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 setRenderView:_localView];
[_pusher startCamera:YES];
```

> ⚠ 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 startCamera 接口启动了摄像头预览，就可以调用 V2TXLivePusher 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 V2TXLivePusher 中的 stopPush 接口结束推流。

```
//结束推流
[_pusher stopPush];
```

> ⚠ 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。

- **如何获取可用的推流 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 V2TXLIVE_ERROR_INVALID_LICENSE 的原因**
  如果 startPush 接口返回 V2TXLIVE_ERROR_INVALID_LICENSE，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，或者在调用 startPush 之前不调用 startCamera 接口即可。

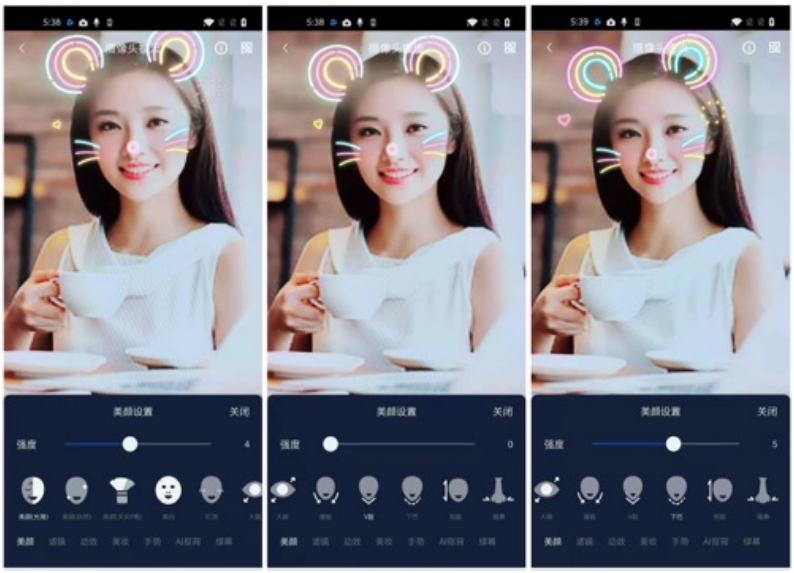
```
V2TXLivePusher *_pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
NSString* rtmpUrl = @"rtmp://test.com/live/xxxxxx";
[_pusher startMicrophone];
[_pusher startPush:rtmpUrl];
```

> 说明：
> 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 V2TXLivePusher 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 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美颜效果。

#### 美颜风格

SDK 内置三种不同的磨皮算法，每种磨皮算法即对应一种美颜风格，您可以选择最适合您产品定位的方案。详情请参见 TXBeautyManager.h 文件：

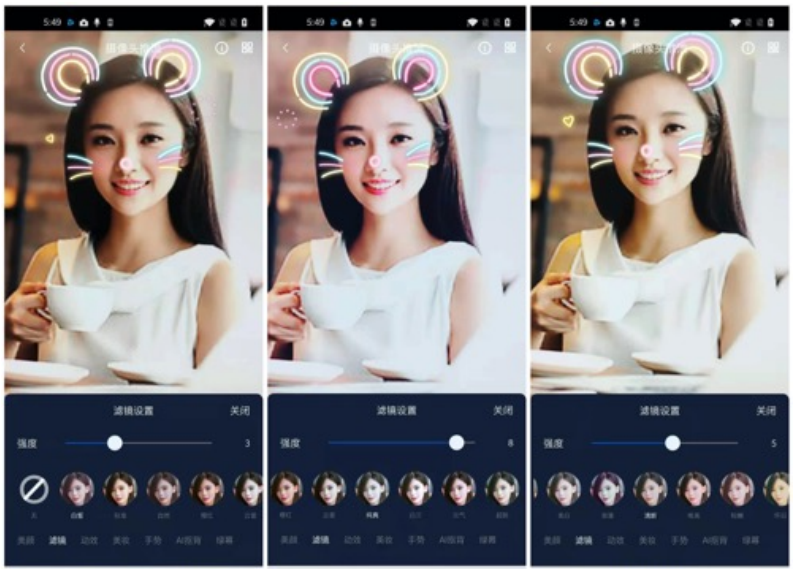
| 美颜风格 | 效果说明 |
| --- | --- |
| TXBeautyStyleSmooth | 光滑，适用于美女秀场，效果比较明显 |
| TXBeautyStyleNature | 自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然 |
| TXBeautyStylePitu | 由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高 |

美颜风格可以通过 TXBeautyManager 的 setBeautyStyle 接口设置：

| 美颜风格 | 设置方式 | 接口说明 |
| --- | --- | --- |

| | | |
|---|---|---|
| 美颜级别 | 通过 TXBeautyManager 的 setBeautyLevel 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |
| 美白级别 | 通过 TXBeautyManager 的 setWhitenessLevel 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |
| 红润级别 | 通过 TXBeautyManager 的 setRuddyLevel 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |

### 9. 色彩滤镜效果



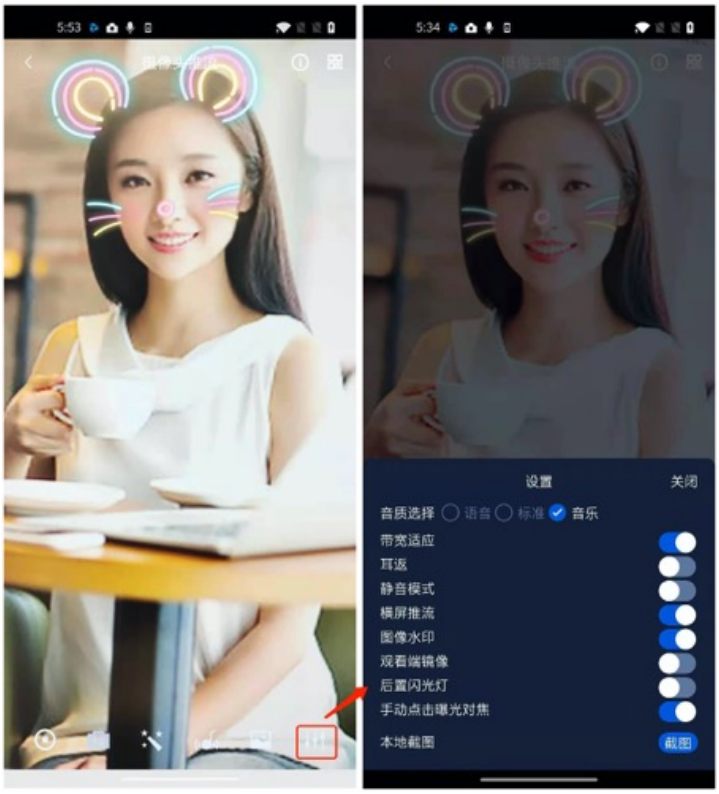
- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 setFilter 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
- 调用 TXBeautyManager 的 setFilterStrength 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

从手机 QQ 和 Now 直播的经验来看，单纯通过 TXBeautyManager 的 setBeautyStyle 调整美颜风格是不够的，只有将美颜风格和 setFilter 配合使用才能达到更加丰富的美颜效果。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。

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:lookupFileName];
UIImage *image = [UIImage imageWithContentsOfFile:path];
[[_pusher getBeautyManager] setFilter:image];
[[_pusher getBeautyManager] setFilterStrength:0.5f];
```

### 10. 设备管理

V2TXLivePusher 提供了一组 API 用户控制设备的行为，您可通过 getDeviceManager 获取 TXDeviceManager 实例进一步进行设备管理，详细用法请参见 TXDeviceManager API。




### 11. 观众端的镜像效果

通过调用 V2TXLivePusher 的 setRenderMirror 可以改变摄像头的镜像方式，继而影响观众端观看到的镜像效果。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，默认情况下自己看到的画面会被 SDK 反转。


### 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：

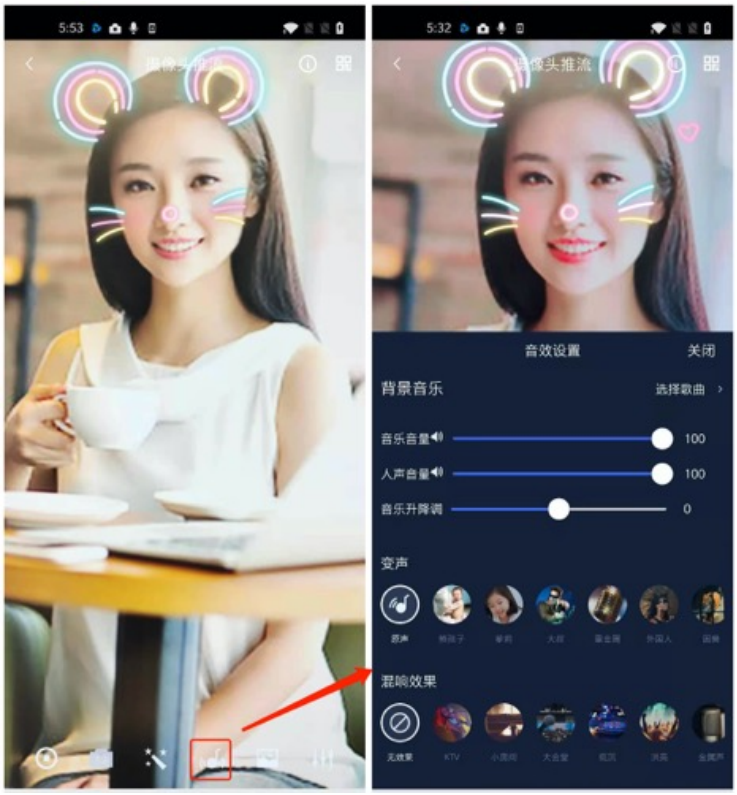

V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
[_pusher setVideoQuality:videoQuality
resolutionMode:isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideoResolutionModePortrait];
```

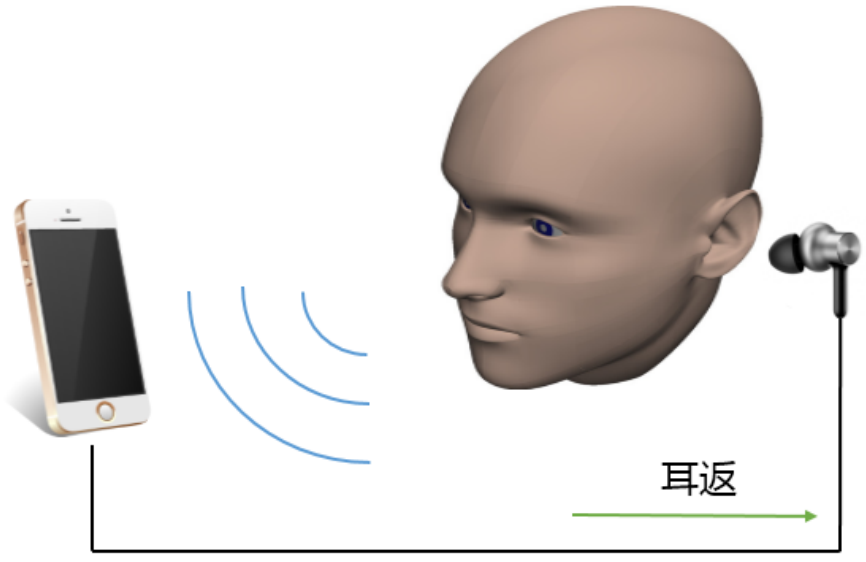
### 13. 音效设置

调用 V2TXLivePusher 中的 getAudioEffectManager 获取 TXAudioEffectManager 实例可以实现背景混音、耳返、混响等音效功能。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”。




- 调用 TXAudioEffectManager 中的 `enableVoiceEarMonitor` 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 TXAudioEffectManager 中的 `setVoiceReverbType` 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 TXAudioEffectManager 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

> 说明：
> 详细用法请参见 TXAudioEffectManager API。

### 14. 设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数（x，y，scale）所决定。

- 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 图片修改后缀名是不起作用的）。
- （x，y，scale）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 × 960，该字段设置为：（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
[_pusher setWatermark:[UIImage imageNamed:@"watermark"] x:0.03 y:0.015 scale:1];
```

### 15. 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 V2TXLivePusherObserver 里的 onWarning 可以捕获 V2TXLIVE_WARNING_NETWORK_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)onWarning:(V2TXLiveCode)code
message:(NSString *)msg
extraInfo:(NSDictionary *)extraInfo {
dispatch_async(dispatch_get_main_queue(), ^{
if (code == V2TXLIVE_WARNING_NETWORK_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];
}
});
}
```

## 事件处理

### 事件监听

SDK 通过 V2TXLivePusherObserver 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 。

### 错误通知

SDK 发现部分严重问题，推流无法继续。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_ERROR_FAILED | -1 | 暂未归类的通用错误 |
| V2TXLIVE_ERROR_INVALID_PARAMETER | -2 | 调用 API 时，传入的参数不合法 |
| V2TXLIVE_ERROR_REFUSED | -3 | API 调用被拒绝 |
| V2TXLIVE_ERROR_NOT_SUPPORTED | -4 | 当前 API 不支持调用 |
| V2TXLIVE_ERROR_INVALID_LICENSE | -5 | license 不合法，调用失败 |
| V2TXLIVE_ERROR_REQUEST_TIMEOUT | -6 | 请求服务器超时 |
| V2TXLIVE_ERROR_SERVER_PROCESS_FAILED | -7 | 服务器无法处理您的请求 |

### 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。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_NETWORK_BUSY | 1101 | 网络状况不佳：上行带宽太小，上传数据受阻 |
| V2TXLIVE_WARNING_VIDEO_BLOCK | 2105 | 视频回放期间出现滞后 |
| V2TXLIVE_WARNING_CAMERA_START_FAILED | -1301 | 摄像头打开失败 |
| V2TXLIVE_WARNING_CAMERA_OCCUPIED | -1316 | 摄像头正在被占用中，可尝试打开其他摄像头 |
| V2TXLIVE_WARNING_CAMERA_NO_PERMISSION | -1314 | 摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_MICROPHONE_START_FAILED | -1302 | 麦克风打开失败 |

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_MICROPHONE_OCCUPIED | -1319 | 麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败 |
| V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION | -1317 | 麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED | -1309 | 当前系统不支持屏幕分享 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED | -1308 | 开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED | -7001 | 录屏被系统中断 |

# Android

最近更新时间：2021-08-04 09:56:30

## 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 V2TXLivePusher 接口提供摄像头推流能力，如下是 LiteAVSDK 腾讯云工具包 App 中演示摄像头推流的相关操作界面：




## 特别说明

**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 |

> **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

## 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

单击 License 申请 获取测试用 License，您会获得两个字符串：其中一个字符串是 licenceURL ，另一个字符串是解密 licenceKey。
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);
}
}
```

### 3. 初始化 V2TXLivePusher 组件

创建一个 V2TXLivePusher 对象，该对象负责完成推流的主要工作。

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP); //指定对应的直播协议为 RTMP
```

### 4. 开启摄像头预览

想要开启摄像头的预览画面，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：

1. 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" />
```

2. 通过调用 V2TXLivePusher 中的 startCamera 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.setRenderView(mPusherView);
mLivePusher.startCamera(true);
```

### 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 V2TXLivePusher 中的 startPush 接口开始推流。

```
//启动推流
String rtmpURL = "rtmp://test.com/live/xxxxxx"; //此处填写您的 rtmp 推流地址
int ret = mLivePusher.startPush(rtmpURL.trim());
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

推流结束后，可以调用 V2TXLivePusher 中的 stopPush 接口结束推流。

```
//结束推流
mLivePusher.stopPush();
```

> ⚠ 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。

- **如何获取可用的推流 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 V2TXLIVE_ERROR_INVALID_LICENSE 的原因?**
  如果 startPush 接口返回 V2TXLIVE_ERROR_INVALID_LICENSE，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，或者在调用 startPush 之前调用 stopCamera 接口即可。

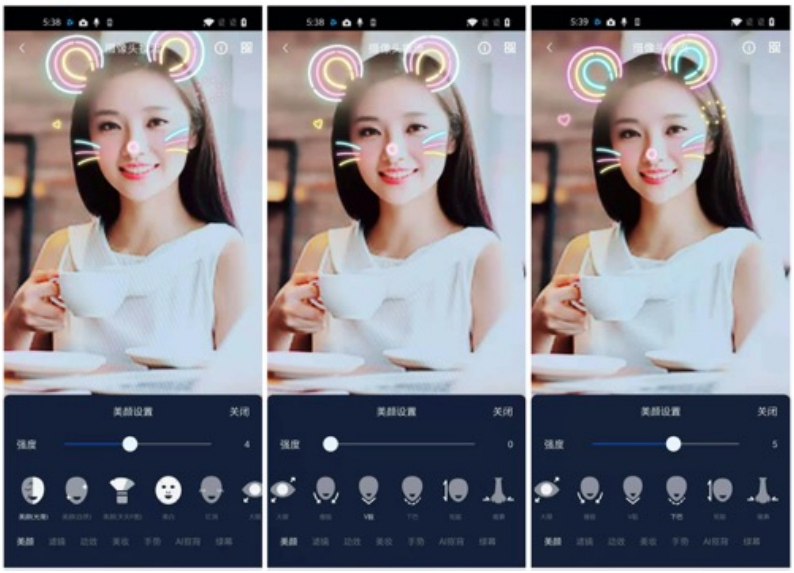
```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP); //指定对应的直播协议为 RTMP
mLivePusher.startMicrophone();
String rtmpURL = "rtmp://test.com/live/xxxxxx"; //此处填写您的 rtmp 推流地址
int ret = mLivePusher.startPush(rtmpURL.trim());
```

> 说明：
> 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 V2TXLivePusher 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 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美颜效果。

#### 美颜风格

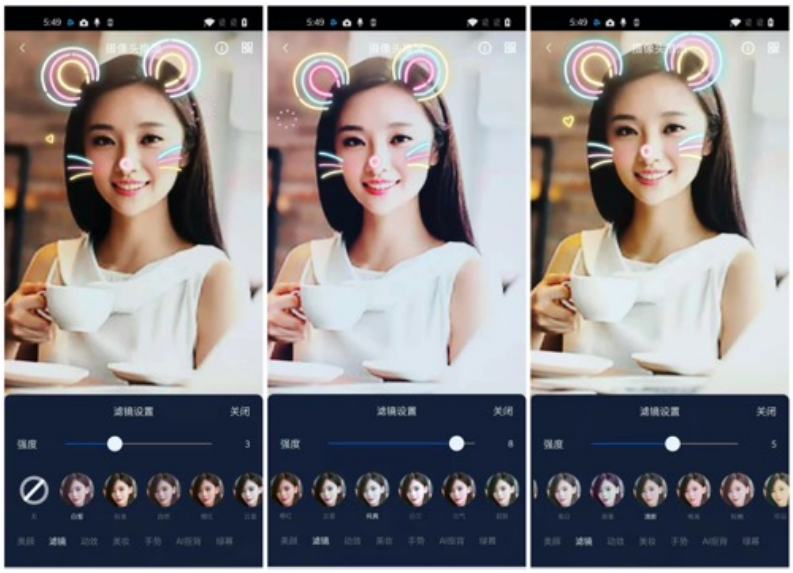
SDK 内置三种不同的磨皮算法，每种磨皮算法即对应一种美颜风格，您可以选择最适合您产品定位的方案。定义见 `TXLiveConstants.java` 文件：

| 美颜风格 | 效果说明 |
| --- | --- |
| BEAUTY_STYLE_SMOOTH | 光滑，适用于美女秀场，效果比较明显 |
| BEAUTY_STYLE_NATURE | 自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然 |
| BEAUTY_STYLE_PITU | 由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高 |

美颜风格可以通过 TXBeautyManager 的 `setBeautyStyle` 接口设置：

| 美颜风格 | 设置方式 | 接口说明 |
| --- | --- | --- |
| 美颜级别 | 通过 TXBeautyManager 的 `setBeautyLevel` 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |
| 美白级别 | 通过 TXBeautyManager 的 `setWhitenessLevel` 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |
| 红润级别 | 通过 TXBeautyManager 的 `setRuddyLevel` 设置 | 取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显 |

### 9. 色彩滤镜效果



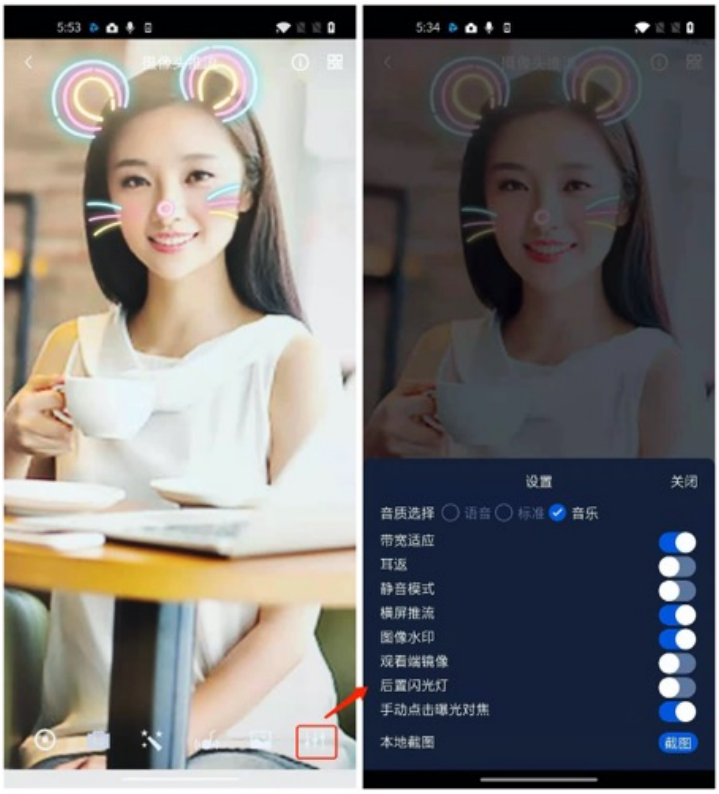

- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。
- 调用 TXBeautyManager 的 `setFilterStrength` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

从手机 QQ 和 Now 直播的经验来看，单纯通过 TXBeautyManager 的 `setBeautyStyle` 调整美颜风格是不够的，只有将美颜风格和 `setFilter` 配合使用才能达到更加丰富的美颜效果。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。

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

### 10. 设备管理

V2TXLivePusher 提供了一组 API 用户控制设备的行为。您通过 `getDeviceManager` 获取 TXDeviceManager 实例进一步进行设备管理，详细用法请参见 TXDeviceManager API。




### 11. 观众端的镜像效果

通过调用 V2TXLivePusher 的 setRenderMirror 可以改变摄像头的镜像方式，继而影响观众端观看到的镜像效果。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，默认情况下自己看到的画面会被 SDK 反转，这时主播就像照镜子一样，观众看到的效果和主播看到的是一致的。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：

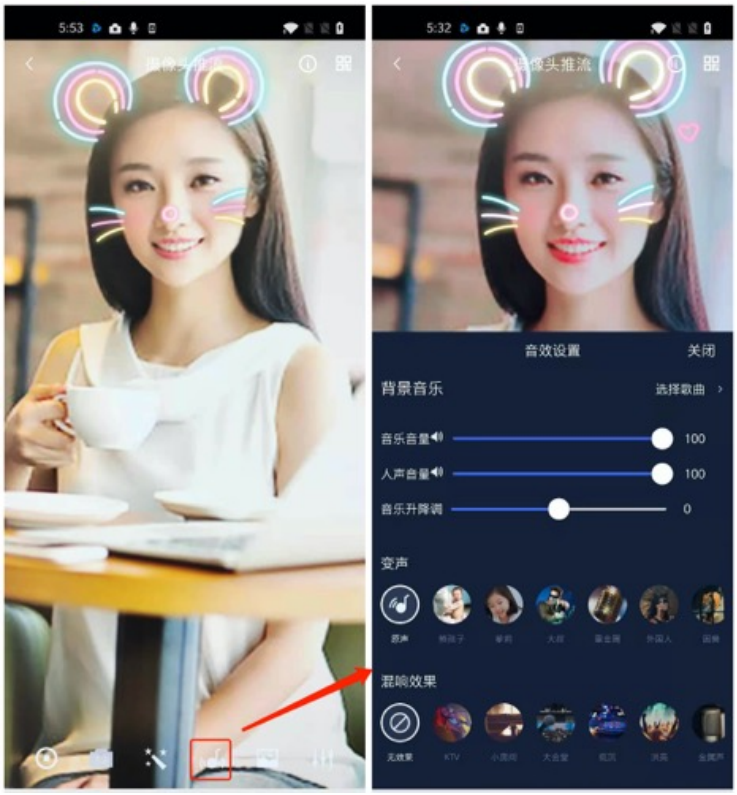

V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 `setVideoQuality` 接口的参数来设定观众端的画面横竖屏模式。

```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideoResolutionModePortrait);
```

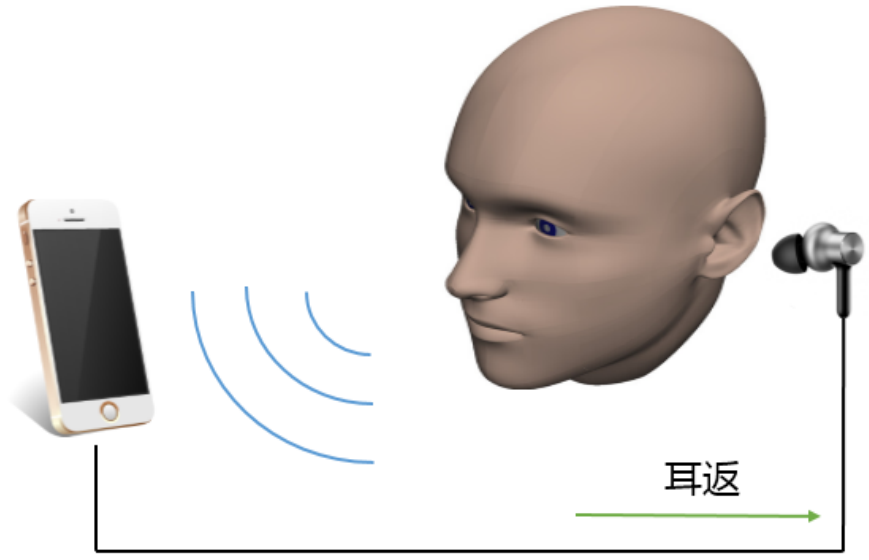
### 13. 音效设置

调用 V2TXLivePusher 中的 `getAudioEffectManager` 获取 TXAudioEffectManager 实例可以实现背景混音、耳返、混响等音效功能。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”。




- 调用 TXAudioEffectManager 中的 `enableVoiceEarMonitor` 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 TXAudioEffectManager 中的 `setVoiceReverbType` 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 TXAudioEffectManager 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

> 说明：
> 详细用法请参见 TXAudioEffectManager API。

### 14. 设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数（x，y，scale）所决定。

- SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 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 图片修改后缀名是不起作用的）。
- （x，y，scale）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 × 960，该字段设置为：（0.1，0.1，0.1），则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
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 0.03f, 0.015f,
1f);
```

### 15. 主播端弱网提醒

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，提示主播应当检查网络。
通过 V2TXLivePusherObserver 里的 onWarning 可以捕获 V2TXLIVE_WARNING_NETWORK_BUSY 事件，它代表当前主播的网络已经非常糟糕，出现此事件即代表观众端会出现卡顿。此时可以在 UI 上弹出一个“弱网提示”来强提醒主播检查网络。

```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
showNetBusyTips(); // 显示网络繁忙的提示
}
}
```

## 事件处理

### 事件监听

SDK 通过 V2TXLivePusherObserver 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。

### 错误通知

SDK 发现部分严重问题，推流无法继续。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_ERROR_FAILED | -1 | 暂未归类的通用错误 |
| V2TXLIVE_ERROR_INVALID_PARAMETER | -2 | 调用 API 时，传入的参数不合法 |
| V2TXLIVE_ERROR_REFUSED | -3 | API 调用被拒绝 |
| V2TXLIVE_ERROR_NOT_SUPPORTED | -4 | 当前 API 不支持调用 |
| V2TXLIVE_ERROR_INVALID_LICENSE | -5 | license 不合法，调用失败 |
| V2TXLIVE_ERROR_REQUEST_TIMEOUT | -6 | 请求服务器超时 |
| V2TXLIVE_ERROR_SERVER_PROCESS_FAILED | -7 | 服务器无法处理您的请求 |

### 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。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_NETWORK_BUSY | 1101 | 网络状况不佳：上行带宽太小，上传数据受阻 |
| V2TXLIVE_WARNING_VIDEO_BLOCK | 2105 | 当前视频播放出现卡顿 |
| V2TXLIVE_WARNING_CAMERA_START_FAILED | -1301 | 摄像头打开失败 |
| V2TXLIVE_WARNING_CAMERA_OCCUPIED | -1316 | 摄像头正在被占用中，可尝试打开其他摄像头 |
| V2TXLIVE_WARNING_CAMERA_NO_PERMISSION | -1314 | 摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_MICROPHONE_START_FAILED | -1302 | 麦克风打开失败 |
| V2TXLIVE_WARNING_MICROPHONE_OCCUPIED | -1319 | 麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败 |
| V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION | -1317 | 麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED | -1309 | 当前系统不支持屏幕分享 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED | -1308 | 开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED | -7001 | 录屏被系统中断 |

# 微信小程序

最近更新时间：2021-07-19 15:45:08

**<live-pusher> 是小程序内部用于支持音视频上行能力的功能标签，本文主要介绍该标签的使用方法。**

## 版本支持

- 微信 App iOS 最低版本要求：6.5.21。
- 微信 App Android 最低版本要求：6.5.19。
- 小程序基础库最低版本要求：1.7.0。

> 说明：
> 通过 wx.getSystemInfo 可以获取当前基础库版本信息。

## 使用限制

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 个人账号和企业账号的小程序暂时只开放如下表格中的类目：

| 一级类目/主体类型 | 二级类目 | 资质要求 | 类目适用范围 | 小程序直播内容场景 |
| --- | --- | --- | --- | --- |
| 社交 | 直播 | （3选1）：<br>1.《信息网络传播视听节目许可证》<br>2.《网络文化经营许可证》(经营范围含网络表演）<br>3.《统一社会信用代码》及《情况说明函件》（适用于政府主体） | 适用于提供在线直播等服务<br>注：<br>1.如提供时政信息服务，需补充：时政信息类目<br>2.选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 | 涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 |
| 教育 | 在线视频课程 | （5选1）：<br>1.《事业单位法人证书》（适用公立学校）<br>2. 区、县级教育部门颁发的《民办学校办学许可证》（适用培训机构）<br>3.《信息网络传播视听节目许可证》<br>4. 全国校外线上培训管理服务平台备案<br>5. 教育部门的批准文件 | 适用于教育行业提供，网课、在线培训、讲座等教育类视频/直播等服务 | 网课、在线培训、讲座等教育类直播 |
| 医疗 | 互联网医院 | （2选1）：<br>1. 卫生健康部门的《设置医疗机构批准书》；<br>2. 合作医院的《医疗机构执业许可证》与执业登记机关的审核合格文件 | 适用于互联网医院主体/医疗服务平台提供在线看诊、疾病咨询等线上医疗服务 | 问诊、大型健康讲座等直播 |
| | 公立医疗机构 | 《医疗机构执业许可证》与《事业单位法人证书》 | 适用于公立医疗机构提供的就医、健康咨询/问诊、医疗保健信息等服务 | |
| 金融 | 银行 | （2选1）：<br>1.《金融许可证》<br>2.《金融机构许可证》 | 适用于提供银行业务在线服务或交易等服务 | 金融产品视频客服理赔、金融产品推广直播等 |

| | | | | |
|---|---|---|---|---|
| | 信托 | （2选1）：<br>1.《金融许可证》<br>2.《金融机构许可证》 | 适用于提供信托理财业务在线服务或交易等服务 | |
| | 公募基金 | （3选1）：<br>1.《经营证券期货业务许可证》且业务范围必须包含“基金”<br>2.《基金托管业务许可证》<br>3.《基金销售业务资格证书》 | 适用于基金管理公司从事股票、债券等金融工具的投资服务 | |
| | 私募基金 | （2选1）：<br>1.《私募基金备案证明》<br>2.《私募投资基金管理人登记证书》 | 仅适用于私募基金展示、介绍、咨询等服务<br>注：暂不支持涉及私募产品公开募集或在线交易等服务 | |
| | 证券/期货 | 《经营证券期货业务许可证》 | 适用于提供证券资讯、证券咨询、证券期货经营等的在线服务 | |
| | 证券、期货投资咨询 | （2选1）：<br>1.《证券投资咨询业务资格证书》<br>2.《经营证券期货业务许可证》 | 适用于提供证券、期货投资等在线咨询服务 | |
| | 保险 | （8选1）：<br>1.《保险公司法人许可证》<br>2.《经营保险业务许可证》<br>3.《保险营销服务许可证》<br>4.《经营保险代理业务许可证》<br>5.《经营保险经纪业务许可证》<br>6.《经营保险公估业务许可证》<br>7.《经营保险资产管理业务许可证》<br>8.《保险兼业代理业务许可证》 | 适用于提供保险业务在线服务或交易等服务 | |
| | 征信业务 | （2选1）：<br>1.经营个人征信业务：《个人征信业务经营许可证》、《营业执照》<br>2.经营企业征信业务：经所在地的中国人民银行及其派出机构备案的《企业征信业务经营备案证》、《营业执照》 | 适用于银行或征信机构提供征信业务服务，包括：信贷记录、逾期记录、失信人查询等 | |
| | 新三板信息服务平台 | 全国中小企业股份转让系统有限责任公司的书面许可与《非经营性互联网信息服务备案核准》 | 适用于提供新三板信息行情资讯等服务 | |
| | 股票信息服务平台（港股/美股） | 《非经营性互联网信息服务备案核准》 | 适用于提供港股、美股行情资讯、行情分析等服务<br>注：如提供股票交易服务，需补充：金融业-证券/期货类目 | |
| | 消费金融 | 银监会核准开业的审批文件与《金融许可证》与《营业执照》 | 适用于提供消费金融线上服务或交易等服务 | |
| 汽车 | 汽车预售服务 | （3选1）：<br>1.汽车厂商：《营业执照》与《工信部道路机动车辆生产企业准入许可》<br>2.汽车经销商/4s店：《营业执照》与《厂商授权销售文件》与《工信部道路机动车辆生产企业准入许可》<br>3.下属子/分公司：《营业执照》与《工信部 | 适用于提供汽车在线预付款等服务<br>注：平台暂不支持在线整车销售，如涉及整车销售服务，建议改为价格指导或移除相关功能 | 汽车预售、推广直播 |

| | | | | |
|---|---|---|---|---|
| | | 道路机动车辆生产企业准入许可》与《股权关系证明函》（含双方盖章） | | |
| 政府主体帐号 | – | – | – | 政府相关工作推广直播、领导讲话直播等 |
| 工具 | 视频客服 | – | 适用于提供企业售后客服一对一视频等服务 | 不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等 |

**说明：**

**可申请直播标签的小程序类目以 微信文档 说明为主，小程序类目的资质要求详见 非个人主体类目申请。**

- **符合类目要求的小程序，需要在小程序管理后台的【 开发 】>【 接口设置 】中自助开通该组件权限，如下图所示：**


**注意：**

**如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。**

## 属性定义

| 属性名 | 类型 | 默认值 | 说明 |
|---|---|---|---|
| url | String | – | 用于音视频上行的推流 URL |
| mode | String | RTC | SD，HD，FHD，RTC |
| autopush | Boolean | false | 是否自动启动推流 |

| 属性名 | 类型 | 默认值 | 说明 |
| --- | --- | --- | --- |
| muted | Boolean | false | 是否静音 |
| enable-camera | Boolean | true | 开启\关闭摄像头 |
| auto-focus | Boolean | true | 手动\自动对焦 |
| orientation | String | vertical | vertical，horizontal |
| beauty | Number | 0 | 美颜指数，取值 0 - 9，数值越大效果越明显 |
| whiteness | Number | 0 | 美白指数，取值 0 - 9，数值越大效果越明显 |
| aspect | String | 9：16 | 3：4，9：16 |
| zoom | Boolean | false | 是否正常焦距，true 表示将摄像头放大 |
| device-position | String | front | front 前置摄像头，back 后置摄像头 |
| min-bitrate | Number | 200 | 最小码率，该数值决定了画面最差的清晰度表现 |
| max-bitrate | Number | 1000 | 最大码率，该数值决定了画面最好的清晰度表现 |
| audio-quality | String | low | low 适合语音通话，high 代表高音质 |
| waiting-image | String | - | 当微信切到后台时的垫片图片 |
| waiting-image-hash | String | - | 当微信切到后台时的垫片图片的校验值 |
| background-mute | Boolean | false | 当微信切到后台时是否禁用声音采集 |
| bindstatechange | String | - | 用于指定一个 javascript 函数来接收音视频事件 |
| debug | Boolean | false | 是否开启调试模式 |

## 示例代码

```
<view id='video-box'>
<live-pusher
id="pusher"
mode="RTC"
url="{{pusher.push_url}}"
autopush='true'
bindstatechange="onPush">
</live-pusher>
</view>
```

## 属性详解

- **url**
  用于音视频上行的推流 URL，以 rtmp:// 协议前缀开头，腾讯云推流 URL 的获取方法见 快速获取 URL 文档。

> 说明：

> 小程序内部使用的 RTMP 协议是支持 UDP 加速的版本，在同样网络条件下，UDP 版本的 RTMP 会比开源版本的有更好的上行速度和抗抖动能力。

- **mode**
  SD、HD 和 FHD 主要用于直播类场景，例如赛事直播、在线教育、远程培训等等。SD、HD 和 FHD 分别对应三种默认的清晰度。该模式下，小程序会更加注重清晰度和观看的流畅性，不会过分强调低延迟，也不会为了延迟牺牲画质和流畅性。

  RTC 则主要用于双向视频通话或多人视频通话场景，例如金融开会、在线客服、车险定损、培训会议等。该模式下，小程序会更加注重降低点到点的时延，也会优先保证声音的质量，在必要的时候会对画面清晰度和画面的流畅性进行一定的缩水。

- **orientation 和 aspect**
  横屏（horizontal）模式还是竖屏（vertical）模式，默认是竖屏模式，即 home 键朝下。这时，小程序推出的画面的宽高比是3：4或者9：16这两种竖屏宽高比的画面，也就是宽 < 高。如果改成横屏模式，小程序推出的画面宽高比即变为4：3或者16：9这种横屏宽高比的画面，也就是宽 > 高。

  具体的宽高比是由 aspect 决定的 ，默认是9：16，也可以支持3：4。这是在 orientation 的属性值为 vertical 的情况下。如果 orientation 的属性值为 horizontal，那么3：4的效果等价于4：3，9：16的效果等价于16：9。

- **min-bitrate 和 max-bitrate**
  这里首先要科普一个概念 —— 视频码率，指视频编码器每秒钟输出的视频数据的多少。在视频分辨率确定的情况下，视频码率越高，即每秒钟输出的数据越多，相应的画质也就越好。

  所以 min-bitrate 和 max-bitrate 这两个属性，分别用于决定输出画面的最低清晰度和最高清晰度。这两个数值并非越大越好，因为用户的网络上行不是无限好的。但也不是越小越好，因为实际应用场景中，清晰与否是用户衡量产品体验的一个重要指标。具体的数值设定我们会在 参数设置 部分详细介绍。

  小程序内部会自动处理好分辨率和码率的关系，例如2Mbps的码率，小程序会选择720p的分辨率进行匹配，而300kbps的码率下，小程序则会选择较低的分辨率来提高编码效率。所以您只需要关注 min-bitrate 和 max-bitrate 这一对参数就可以掌控画质。

- **waiting-image 和 waiting-image-hash**
  出于用户隐私的考虑，在微信切到后台以后，小程序希望停止摄像头的画面采集。但是对于另一端的用户而言，画面会变成黑屏或者冻屏（停留在最后一帧），这种体验是非常差的。为了解决这个问题，我们引入了 waiting-image 属性，您可以设置一张有 “稍候” 含义的图片（waiting-image 是该图片的 URL，waiting-image-hash 则是该图片对应的 md5 校验值）。当微信切到后台以后，小程序会使用该图片作为摄像头画面的替代，以极低的流量占用维持视频流3分钟时间。

- **debug**
  调试音视频相关功能，如果没有很好的工具会是一个噩梦，所以小程序为 live-pusher 标签支持了 debug 模式，开始 debug 模式之后，原本用于渲染视频画面的窗口上，会显示一个半透明的 log 窗口，用于展示各项音视频指标和事件，降低您调试相关功能的难度，具体使用方法我们在 FAQ 中有详细说明。

## 参数设置

这么多参数，具体要怎样设置才比较合适，我们给出如下建议值：

| 场景 | mode | min-bitrate | max-bitrate | audio-quality | 说明 |
| --- | --- | --- | --- | --- | --- |
| 标清直播 | SD | 300kbps | 800kbps | high | 窄带场景，例如户外或者网络不稳定的情况下适用 |
| 高清直播 | HD | 600kbps | 1200kbps | high | 目前主流的 App 所采用的参数设定，普通直播场景推荐使用这一档 |

| 场景 | mode | min-bitrate | max-bitrate | audio-quality | 说明 |
| --- | --- | --- | --- | --- | --- |
| 超清直播 | FHD | 600kbps | 1800kbps | high | 对清晰度要求比较苛刻的场景，普通手机观看使用 HD 即可 |
| 视频客服（用户） | RTC | 200kbps | 500kbps | high | 这是一种声音为主，画面为辅的场景，所以画质不要设置的太高 |
| 车险定损（车主） | RTC | 200kbps | 1200kbps | high | 由于可能要看车况详情，画质上限会设置的高一些 |
| 多人会议（主讲） | RTC | 200kbps | 1000kbps | high | 主讲人画质可以适当高一些，参与的质量可以设置的低一些 |
| 多人会议（参与） | RTC | 150kbps | 300kbps | low | 作为会议参与者，不需要太高的画质和音质 |

说明：
如果不是对带宽特别没有信心的应用场景，audio-quality 选项请不要选择 low，其音质和延迟感都会比 high 模式差很多。

## 对象操作

- **wx.createLivePusherContext()**
  通过 wx.createLivePusherContext() 可以将 <live-pusher> 标签和 javascript 对象关联起来，之后即可操作该对象。
- **start**
  开始推流，如果 <live-pusher> 的 autopush 属性设置为 false（默认值），那么就可以使用 start 来手动开始推流。
- **stop**
  停止推流。
- **pause**
  暂停推流。
- **resume**
  恢复推流，请与 pause 操作配对使用。
- **switchCamera**
  切换前后摄像头。
- **snapshot**
  推流截图，截图大小跟组件的大小一致。截图成功图片的临时路径为 ret.tempImagePath。

```
var pusher = wx.createLivePusherContext('pusher');
pusher.start({
success: function(ret){
console.log('start push success!')
}
fail: function(){
console.log('start push failed!')
}
```

```
complete: function(){
console.log('start push complete!')
}
});
```

## 内部事件

通过 <live-pusher> 标签的 bindstatechange 属性可以绑定一个事件处理函数，该函数可以监听推流模块的内部事件和异常通知。

### 1. 常规事件

| 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 |

### 2. 严重错误

| 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，且经三次重连无效，可以放弃，更多重试请自行重启推流 |

### 3. 警告事件

内部警告并非不可恢复的错误，小程序内部的音视频 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，例如：

- PUSH_WARNING_NET_BUSY
  上行网速不给力，建议提示用户改善当前的网络环境，例如让用户离家里的路由器近一点，或者切到 Wi-Fi 环境下再使用。
- PUSH_WARNING_SERVER_DISCONNECT
  请求被后台拒绝，出现这个问题一般是由于 URL 里的 txSecret 计算错，或者是 URL 被其他人占用（跟播放不同，一个推流 URL 同时只能有一个用户使用）。
- PUSH_WARNING_HANDUP_STOP
  当用户单击小程序右上角的圆圈或者返回按钮时，微信会将小程序挂起，此时 <live-pusher> 会抛出5000这个事件。

| 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） |

| code | 事件定义 | 含义说明 |
| --- | --- | --- |
| 1103 | PUSH_WARNING_HW_ACCELERATION_FAIL | 硬编码启动失败，自动切换到软编码 |
| 1107 | PUSH_WARNING_SWITCH_SWENC | 由于机器性能问题，自动切换到硬件编码 |
| 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 |
| 5000 | PUSH_WARNING_HANDUP_STOP | 小程序被用户挂起，停止推流 |

**4. 示例代码**

```
Page({
onPush: function(ret) {
if(ret.detail.code == 1002) {
console.log('推流成功了',ret);
}
},

/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
//...
}
})
```

# Web 推流

最近更新时间：2021-08-09 17:37:07

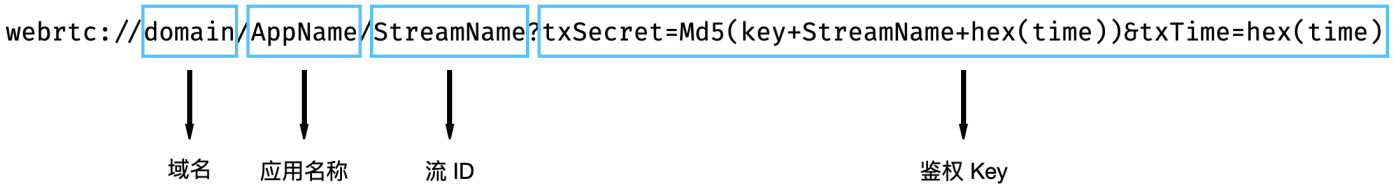
TXLivePusher 推流 SDK 主要用于视频云的快直播（超低延迟直播）推流，负责将浏览器采集的音视频画面通过 WebRTC 推送到直播服务器。目前支持摄像头推流、屏幕录制推流和本地媒体文件推流。

## 基础知识

对接前需要了解以下基础知识：

### 推流地址的拼装

使用腾讯云直播服务时，推流地址需要满足腾讯云标准直播推流 URL 的格式 ，如下所示，它由四个部分组成：




其中鉴权 Key 部分非必需，如果需要防盗链，请开启推流鉴权，具体使用说明请参见 自主拼装直播 URL 。

### 浏览器支持

快直播推流基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持。

除此以外，浏览器采集音视频画面的功能在移动端支持较差，例如移动端浏览器不支持屏幕录制，iOS 14.3及以上版本才支持获取用户摄像头设备。因此推流 SDK 主要适用于桌面端浏览器，目前最新版本的 chrome、Firefox 和 Safari 浏览器都是支持快直播推流的。

移动端建议使用 移动直播 SDK 进行推流。

## 对接攻略

### 步骤1：页面准备工作

在需要直播推流的页面（桌面端）中引入初始化脚本。

```
<script src="https://imgcache.qq.com/open/qcloud/live/webrtc/js/TXLivePusher-1.0.1.min.js" charset="utf-8">
</script>
```

如果在域名限制区域，可以引入以下链接：

```
<script src="https://cloudcache.tencent-cloud.com/open/qcloud/live/webrtc/js/TXLivePusher-1.0.1.min.js" cha
rset="utf-8"></script>
```

### 步骤2：在 HTML 中放置容器

在需要展示本地音视频画面的页面位置加入播放器容器，即放一个 div 并命名，例如 id_local_video，本地视频画面都会在容器里渲染。对于容器的大小控制，您可以使用 div 的 css 样式进行控制，示例代码如下：

```
<div id="id_local_video" style="width:100%;height:500px;display:flex;align-items:center;justify-content:cen
ter;"></div>
```

### 步骤3：直播推流

1. **生成推流 SDK 实例：**
   通过全局对象 `TXLivePusher` 生成 SDK 实例，后续操作都是通过实例完成。

```
var livePusher = new TXLivePusher();
```

2. **指定本地视频播放器容器：**
   指定本地视频播放器容器 div，浏览器采集到的音视频画面会渲染到这个 div 当中。

```
livePusher.setRenderView('id_local_video');
```

> 说明：
> 调用 `setRenderView` 生成的 video 元素默认有声音，如果需要静音的话，可以直接获取 video 元素进行操作。
>
> ```
> document.getElementById('id_local_video').getElementsByTagName('video')[0].muted = true;
> ```

3. **设置音视频质量：**
   采集音视频流之前，先进行音视频质量设置，如果预设的质量参数不满足需求，可以单独进行自定义设置。

```
// 设置视频质量
livePusher.setVideoQuality('720p');
// 设置音频质量
livePusher.setAudioQuality('standard');
// 自定义设置帧率
livePusher.setProperty('setVideoFPS', 25);
```

4. **开始采集流：**
   目前支持采集摄像头设备、麦克风设备、屏幕录制和本地媒体文件的流。当音视频流采集成功时，播放器容器中开始播放本地采集到的音视频画面。

```
// 打开摄像头
livePusher.startCamera();
// 打开麦克风
livePusher.startMicrophone();
```

5. **开始推流：**
   传入腾讯云快直播推流地址，开始推流。推流地址的格式参考 腾讯云标准直播 URL ，只需要将 RTMP 推流地址前面的 `rtmp://` 替换成 `webrtc://` 即可。

```
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
```

> 说明：

推流之前要保证已经采集到了音视频流，否则推流接口会调用失败，如果要实现采集到音视频流之后自动推流，可以通过回调事件通知，当收到采集首帧成功的通知后，再进行推流。如果同时采集了视频流和音频流，需要在视频首帧和音频首帧的采集成功回调通知都收到后再发起推流。

```
var hasVideo = false;
var hasAudio = false;
var isPush = false;
livePusher.setObserver({
onCaptureFirstAudioFrame: function() {
hasAudio = true;
if (hasVideo && !isPush) {
isPush = true;
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
}
},
onCaptureFirstVideoFrame: function() {
hasVideo = true;
if (hasAudio && !isPush) {
isPush = true;
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
}
}
});
```

6. **停止快直播推流：**

```
livePusher.stopPush();
```

7. **停止采集音视频流：**

```
// 关闭摄像头
livePusher.stopCamera();
// 关闭麦克风
livePusher.stopMicrophone();
```

## 进阶攻略

### 兼容性

SDK 提供静态方法用于检测浏览器对于 WebRTC 的兼容性。

```
TXLivePusher.checkSupport().then(function(data) {
// 是否支持WebRTC
if (data.isWebRTCSupported) {
console.log('WebRTC Support');
} else {
console.log('WebRTC Not Support');
}
```

```
// 是否支持H264编码
if (data.isH264EncodeSupported) {
console.log('H264 Encode Support');
} else {
console.log('H264 Encode Not Support');
}
});
```

### 回调事件通知

SDK 目前提供了回调事件通知，可以通过设置 Observer 来了解 SDK 内部的状态信息和 WebRTC 相关的数据统计。具体内容请参见 TXLivePusherObserver。

```
livePusher.setObserver({
// 推流警告信息
onWarning: function(code, msg) {
console.log(code, msg);
},
// 推流连接状态
onPushStatusUpdate: function(status, msg) {
console.log(status, msg);
},
// 推流统计数据
onStatisticsUpdate: function(data) {
console.log('video fps is ' + data.video.framesPerSecond);
}
});
```

### 设备管理

SDK 提供了设备管理实例帮助用户进行获取设备列表、切换设备等操作。

```
var deviceManager = livePusher.getDeviceManager();
// 获取设备列表
deviceManager.getDevicesList().then(function(data) {
data.forEach(function(device) {
console.log(device.deviceId, device.deviceName);
});
});
// 切换摄像头设备
deviceManager.switchCamera('camera_device_id');
```

# 录屏推流
# iOS

最近更新时间：2021-07-16 16:31:07

## 概述

录屏功能是 iOS 10 新推出的特性，苹果在 iOS 9 的 ReplayKit 保存录屏视频的基础上，增加了视频流实时直播功能，官方介绍见 Go Live with ReplayKit。iOS 11 增强为 ReplayKit2，进一步提升了 Replaykit 的易用性和通用性，并且可以对整个手机实现屏幕录制，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。

## 示例代码

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。

| 所属平台 | GitHub 地址 |
| --- | --- |
| iOS | Github |
| Android | Github |

**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。

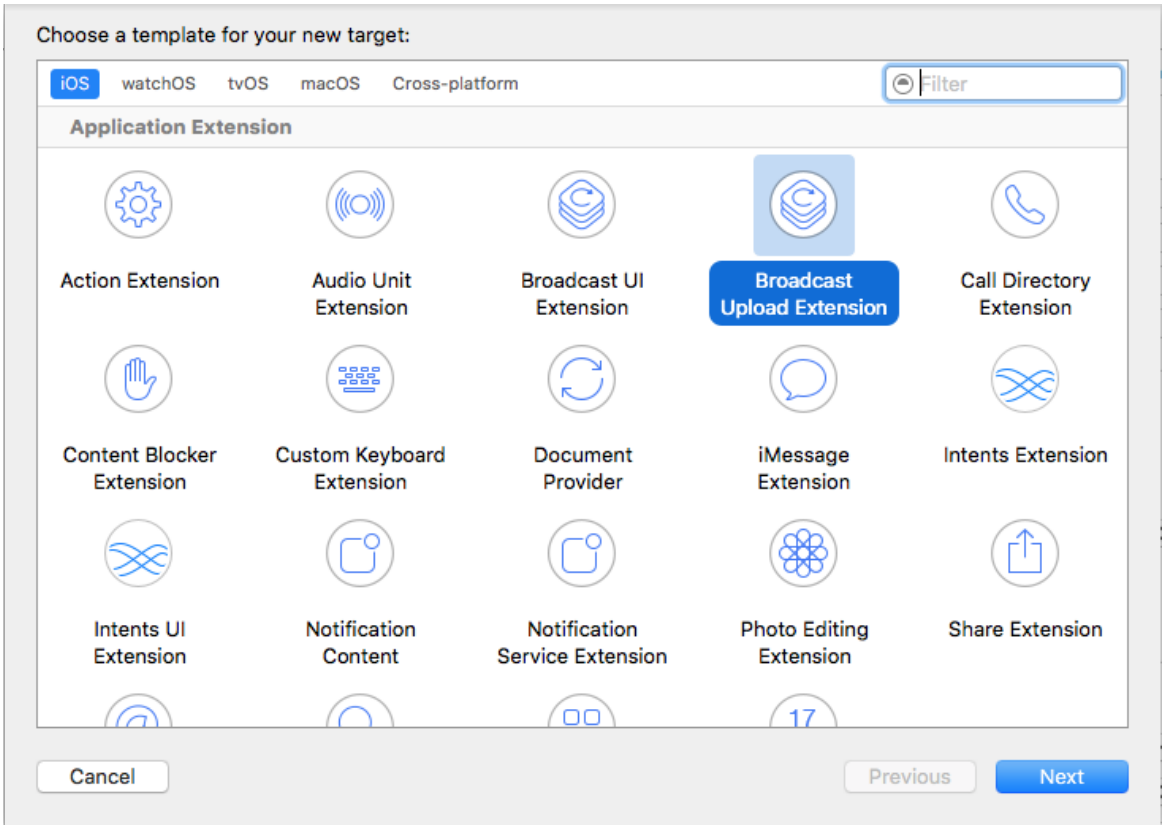
## 开发环境准备

### 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。具体请参见腾讯云官网 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 "V2TXLivePusher.h"

static V2TXLivePusher *s_txLivePublisher;
static NSString *s_rtmpUrl;
```

```
- (void)initPublisher {
if (s_txLivePublisher) {
[s_txLivePublisher stopPush];
}
s_txLivePublisher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
[s_txLivePublisher setObserver:self];
[s_txLivePublisher startPush:s_rtmpUrl];
}
```

- s_txLivePublisher 是我们用于推流的对象，因为系统录屏回调的 sampleHandler 实例有可能不只一个，因此对变量采用静态声明，确保录屏推流过程中使用的是同一个推流器。
- 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 扩展与宿主 App 之间的通信与数据传递方式 。

### 步骤2：横屏推流与分辨率设置

手机录屏直播提供了多个级别的分辨率可供选择。setVideoQuality 方法用来设置分辨率及横竖屏推流，以下录屏推流分辨率与横屏推流设置示例：

```
static BOOL s_landScape; //YES:横屏，  NO:竖屏
[s_txLivePublisher setVideoQuality:V2TXLiveVideoResolution960x540
resolutionMode:s_landScape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideoResolutionModePortrait];
[s_txLivePublisher startPush:s_rtmpUrl];
```

### 步骤3：发送视频

Replaykit 会将视频以回调的方式传给 -[SampleHandler processSampleBuffer:withType]。

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleBufferType {
switch (sampleBufferType) {
case RPSampleBufferTypeVideo:
// Handle video 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);
V2TXLiveVideoFrame *videoFrame = [V2TXLiveVideoFrame new];
videoFrame.bufferType = V2TXLiveBufferTypePixelBuffer;
videoFrame.pixelFormat = V2TXLivePixelFormatNV12;
videoFrame.pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
videoFrame.rotation = V2TXLiveRotation0;
[s_txLivePublisher sendCustomVideoFrame:videoFrame];
}
}
```

视频 sampleBuffer 只需要调用 -[V2TXLivePusher sendCustomVideoFrame:] 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。

> ⚠ 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：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
[s_txLivePublisher setWatermark:image x:0 y:0 scale:1];
```

### 步骤5：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;
}
}
```

因为用于推流的 V2TXLivePusher 对象同一时刻只能有一个在运行，所以结束推流时要做好清理工作。

## 事件处理

### 1. 事件监听

SDK 通过 V2TXLivePusherObserver 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。

### 2. 错误通知

SDK 发现部分严重问题，推流无法继续

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_ERROR_FAILED | -1 | 暂未归类的通用错误。 |
| V2TXLIVE_ERROR_INVALID_PARAMETER | -2 | 调用 API 时，传入的参数不合法。 |
| V2TXLIVE_ERROR_REFUSED | -3 | API 调用被拒绝。 |
| V2TXLIVE_ERROR_NOT_SUPPORTED | -4 | 当前 API 不支持调用。 |
| V2TXLIVE_ERROR_INVALID_LICENSE | -5 | license 不合法，调用失败。 |
| V2TXLIVE_ERROR_REQUEST_TIMEOUT | -6 | 请求服务器超时。 |
| V2TXLIVE_ERROR_SERVER_PROCESS_FAILED | -7 | 服务器无法处理您的请求。 |

### 3. 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。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_NETWORK_BUSY | 1101 | 网络状况不佳：上行带宽太小，上传数据受阻。 |
| V2TXLIVE_WARNING_VIDEO_BLOCK | 2105 | 当前视频播放出现卡顿 |
| V2TXLIVE_WARNING_CAMERA_START_FAILED | -1301 | 摄像头打开失败。 |
| V2TXLIVE_WARNING_CAMERA_OCCUPIED | -1316 | 摄像头正在被占用中，可尝试打开其他摄像头。 |

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_CAMERA_NO_PERMISSION | -1314 | 摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。 |
| V2TXLIVE_WARNING_MICROPHONE_START_FAILED | -1302 | 麦克风打开失败。 |
| V2TXLIVE_WARNING_MICROPHONE_OCCUPIED | -1319 | 麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败。 |
| V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION | -1317 | 麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED | -1309 | 当前系统不支持屏幕分享。 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED | -1308 | 开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了。 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED | -7001 | 录屏被系统中断。 |

### 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(),
kDarvinNotificationNamePushStart,
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(handleReplayKit2PushStartNotifi
cation:) 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 屏幕录制在 iOS 11 新推出功能，相关的官方文档比较少，且存在着一些问题，使得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. **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。

2. **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。

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

# Android

最近更新时间：2021-07-16 14:50:37

## 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。腾讯云 LiteAVSDK 通过 V2TXLivePusher 接口提供录屏推流能力，如下是 LiteAVSDK 腾讯云工具包 App 中演示摄像头推流的相关操作界面：

> 说明：
> 直播中叠加摄像头预览，即通过在手机上添加浮框，显示摄像头预览画面。录屏的时候会把浮框预览画面一并录制下来，达到叠加摄像头预览的效果。


## 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。

## 示例代码

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。

| 所属平台 | GitHub 地址 |
| --- | --- |
| iOS | Github |
| Android | Github |

## 对接攻略

### 步骤1：创建 Pusher 对象

创建一个 V2TXLivePusher 对象，我们后面主要用它来完成推流工作。

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(context, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP);
```

### 步骤2：启动推流

经过 步骤1 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.startMicrophone();
mLivePusher.startScreenCapture();
mLivePusher.startPush(rtmpUrl);
```

- **startScreenCapture** 的作用是启动屏幕录制，由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出提示，允许即可。
- **startPush** 的作用是告诉 LiteAV SDK 音视频流要推到哪个推流 URL 上去。

### 步骤3：设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数（x，y，scale）所决定。

- 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 图片修改后缀名是不起作用的）。
- （x，y，scale）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 × 960，该字段设置为：（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
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.watermark), 0.03f, 0.015f,
1f);
```

### 步骤4：推荐的清晰度

调用 V2TXLivePusher 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。

### 步骤5：提醒主播“网络不好”

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 V2TXLivePusherObserver 里的 onWarning 可以捕获 V2TXLIVE_WARNING_NETWORK_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 onWarning(int code, String msg, Bundle extraInfo) {
if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
showNetBusyTips(); // 显示网络繁忙的提示
}
}
```

### 步骤6：横竖屏适配

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：


V2TXLivePusher 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，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2TXLive
VideoResolutionModePortrait);
```

### 步骤7：结束推流

因为用于推流的 V2TXLivePusher 对象同一时刻只能有一个在运行，所以结束推流时要做好清理工作。

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

## 事件处理

### 事件监听

SDK 通过 V2TXLivePusherObserver 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。

### 错误通知

SDK 发现部分严重问题，推流无法继续。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_ERROR_FAILED | -1 | 暂未归类的通用错误 |
| V2TXLIVE_ERROR_INVALID_PARAMETER | -2 | 调用 API 时，传入的参数不合法 |
| V2TXLIVE_ERROR_REFUSED | -3 | API 调用被拒绝 |
| V2TXLIVE_ERROR_NOT_SUPPORTED | -4 | 当前 API 不支持调用 |
| V2TXLIVE_ERROR_INVALID_LICENSE | -5 | license 不合法，调用失败 |
| V2TXLIVE_ERROR_REQUEST_TIMEOUT | -6 | 请求服务器超时 |
| V2TXLIVE_ERROR_SERVER_PROCESS_FAILED | -7 | 服务器无法处理您的请求 |

### 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。

| 事件 ID | 数值 | 含义说明 |
| --- | --- | --- |
| V2TXLIVE_WARNING_NETWORK_BUSY | 1101 | 网络状况不佳：上行带宽太小，上传数据受阻 |
| V2TXLIVE_WARNING_VIDEO_BLOCK | 2105 | 当前视频播放出现卡顿 |
| V2TXLIVE_WARNING_CAMERA_START_FAILED | -1301 | 摄像头打开失败 |
| V2TXLIVE_WARNING_CAMERA_OCCUPIED | -1316 | 摄像头正在被占用中，可尝试打开其他摄像头 |
| V2TXLIVE_WARNING_CAMERA_NO_PERMISSION | -1314 | 摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_MICROPHONE_START_FAILED | -1302 | 麦克风打开失败 |
| V2TXLIVE_WARNING_MICROPHONE_OCCUPIED | -1319 | 麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败 |
| V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION | -1317 | 麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED | -1309 | 当前系统不支持屏幕分享 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED | -1308 | 开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了 |
| V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED | -7001 | 录屏被系统中断 |

# 标准直播拉流
# iOS

最近更新时间：2021-07-16 14:50:50

## 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，标准直播推荐使用 FLV 协议的直播地址（以 `http` 开头，以 `.flv` 结尾），快直播使用 WebRTC 协议，更多信息请参见 快直播拉流：

| 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 | 延迟较低 | 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 |
| WebRTC | 延迟最低 | 需集成 SDK 才能播放 | < 1s |

> 说明：
> 标准直播与快直播计费价格不同，更多计费详情请参见 标准直播计费 和 快直播计费。

## 特别说明

视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP、HLS（m3u8）和 WebRTC 四种格式的直播地址，以及 MP4、 HLS（m3u8）和 FLV 三种格式的点播地址。

## 示例代码

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。

| 所属平台 | GitHub 地址 |
| --- | --- |
| iOS | Github |
| Android | Github |

## 对接攻略

### 步骤1：创建 Player

视频云 SDK 中的 V2TXLivePlayer 模块负责实现直播播放功能。

```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
```

### 步骤2：渲染 View

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

```
//用 setRenderView 给播放器绑定决定渲染区域的view.
[_txLivePlayer setRenderView:_myView];
```

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
}];
```

### 步骤3：启动播放

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

### 步骤4：画面调整

- **setRenderFillMode：铺满 or 适应**

| 可选值 | 含义 |
| --- | --- |
| V2TXLiveFillModeFill | 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 |
| V2TXLiveFillModeFit | 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：视频画面顺时针旋转角度**

| 可选值 | 含义 |
| --- | --- |
| V2TXLiveRotation0 | 不旋转 |
| V2TXLiveRotation90 | 顺时针旋转90度 |
| V2TXLiveRotation180 | 顺时针旋转180度 |
| V2TXLiveRotation270 | 顺时针旋转270度 |


最长边填充


完全填充


横屏模式

### 步骤5：暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这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 pauseAudio];
[_txLivePlayer pauseVideo];
// 恢复
```

```
[_txLivePlayer resumeAudio];
[_txLivePlayer resumeVideo];
```

### 步骤6：结束播放

```
// 停止播放
[_txLivePlayer stopPlay];
```

### 步骤7：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通过 V2TXLivePlayerObserver 的 onSnapshotComplete 回调截屏图片，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。


```
...
[_txLivePlayer setObserver:self];
[_txLivePlayer snapshot];
...
- (void)onSnapshotComplete:(id<V2TXLivePlayer>)player image:(TXImage *)image {
if (image != nil) {
dispatch_async(dispatch_get_main_queue(), ^{
[self handle:image];
});
}
}
```

## 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**

```
//自动模式
[_txLivePlayer setCacheParams:1 maxTime:5];
//极速模式
[_txLivePlayer setCacheParams:1 maxTime:1];
//流畅模式
[_txLivePlayer setCacheParams:5 maxTime:5];
//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?

## SDK 事件监听

您可以为 V2TXLivePlayer 对象绑定一个 V2TXLivePlayerObserver，之后 SDK 的内部状态信息例如播放器状态、播放音量回调、音视频首帧回调、统计数据、警告和错误信息等会通过对应的回调通知给您。

### 定时触发的状态通知

- onStatisticsUpdate 通知每2秒都会被触发一次，目的是实时反馈当前的播放器状态，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 |
|---|---|
| appCpu | 当前 App 的 CPU 使用率（%） |
| systemCpu | 当前系统的 CPU 使用率（%） |
| width | 视频宽度 |
| height | 视频高度 |
| fps | 帧率（fps） |
| audioBitrate | 音频码率（Kbps） |
| videoBitrate | 视频码率（Kbps） |

- onPlayoutVolumeUpdate 播放器音量大小回调。这个回调仅当您调用 enableVolumeEvaluation 开启播放音量大小提示之后才会工作。回调的时间间隔也会与您在设置 `enableVolumeEvaluation` 的参数 `intervalMs` 保持一致。

### 非定时触发的状态通知

其余的回调仅在事件发生时才会抛出来。

# Android

最近更新时间：2021-07-16 14:50:55

## 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，标准直播推荐使用 FLV 协议的直播地址（以 `http` 开头，以 `.flv` 结尾），快直播使用 WebRTC协议，更多信息请参见 快直播拉流：

| 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 | 延迟较低 | 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 |
| WebRTC | 延迟最低 | 需集成 SDK 才能播放 | < 1s |

> 说明：
> 标准直播与快直播计费价格不同，更多计费详情请参见 标准直播计费 和 快直播计费。

## 特别说明

视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP、HLS（m3u8）和 WebRTC 四种格式的直播地址，以及 MP4、 HLS（m3u8）和 FLV 三种格式的点播地址。

## 示例代码

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。

| 所属平台 | GitHub 地址 |
| --- | --- |
| iOS | Github |
| Android | Github |

## 对接攻略

### 步骤1：添加渲染 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 文件里加入渲染 View：

```
<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"/>
```

### 步骤2：创建 Player

视频云 SDK 中的 **V2TXLivePlayer** 模块负责实现直播播放功能，并使用 **setRenderView** 接口将它与我们刚刚添加到界面上的 **video_view** 渲染控件进行关联。

```
//mPlayerView 即 step1 中添加的界面 view
TXCloudVideoView mView = (TXCloudVideoView) view.findViewById(R.id.video_view);
//创建 player 对象
V2TXLivePlayer mLivePlayer = new V2TXLivePlayerImpl(mContext);
//关键 player 对象与界面 view
mLivePlayer.setRenderView(mView);
```

### 步骤3：启动播放

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startPlay(flvUrl);
```

### 步骤4：画面调整

- **view：大小和位置**
  如需修改画面的大小及位置，直接调整 step1 中添加的 video_view 控件的大小和位置即可。
- **setRenderFillMode：铺满 or 适应**

| 可选值 | 含义 |
| --- | --- |
| V2TXLiveFillModeFill | 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 |
| V2TXLiveFillModeFit | 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：视频画面顺时针旋转角度**

| 可选值 | 含义 |
| --- | --- |
| V2TXLiveRotation0 | 不旋转 |
| V2TXLiveRotation90 | 顺时针旋转90度 |
| V2TXLiveRotation180 | 顺时针旋转180度 |
| V2TXLiveRotation270 | 顺时针旋转270度 |

```
// 设置填充模式
mLivePlayer.setRenderFillMode(V2TXLiveFillModeFit);
```

```
// 设置画面渲染方向
mLivePlayer.setRenderRotation(V2TXLiveRotation0);
```


最长边填充


完全填充


横屏模式

### 步骤5：暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这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.pauseAudio();
mLivePlayer.pauseVideo();
// 继续
mLivePlayer.resumeAudio();
mLivePlayer.resumeVideo();
```

### 步骤6：结束播放

结束播放非常简单，直接调用 `stopPlay` 即可。

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

### 步骤7：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.setObserver(new MyPlayerObserver());
mLivePlayer.snapshot();
// 在MyPlayerObserver的回调接口onSnapshotComplete中获取屏幕截图
private class MyPlayerObserver extends V2TXLivePlayerObserver {
...
@Override
public void onSnapshotComplete(V2TXLivePlayer v2TXLivePlayer, Bitmap bitmap) {
}
...
}
```

## 延时调节

**腾讯云 SDK 的云直播播放功能，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**

```
//自动模式
mLivePlayer.setCacheParams(1.0f, 5.0f);
//极速模式
mLivePlayer.setCacheParams(1.0f, 1.0f);
//流畅模式
mLivePlayer.setCacheParams(5.0f, 5.0f);
//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?

## SDK 事件监听

您可以为 V2TXLivePlayer 对象绑定一个 V2TXLivePlayerObserver，之后 SDK 的内部状态信息例如播放器状态、播放音量回调、音视频首帧回调、统计数据、警告和错误信息等会通过对应的回调通知给您。

### 定时触发的状态通知

- onStatisticsUpdate 通知每2秒都会被触发一次，目的是实时反馈当前的播放器状态，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 |
| --- | --- |
| appCpu | 当前 App 的 CPU 使用率（%） |
| systemCpu | 当前系统的 CPU 使用率（%） |
| width | 视频宽度 |
| height | 视频高度 |
| fps | 帧率（fps） |
| audioBitrate | 音频码率（Kbps） |
| videoBitrate | 视频码率（Kbps） |

- onPlayoutVolumeUpdate 播放器音量大小回调。这个回调仅当您调用 enableVolumeEvaluation 开启播放音量大小提示之后才会工作。回调的时间间隔也会与您在设置 `enableVolumeEvaluation` 的参数 `intervalMs` 保持一致。

### 非定时触发的状态通知

其余的回调仅在事件发生时才会抛出来。

# 微信小程序

最近更新时间：2021-06-24 11:23:27

**<live-player> 是小程序内部用于支持音视频下行（播放）能力的功能标签，本文主要介绍该标签的使用方法。**

## 版本支持

- 微信 App iOS 最低版本要求：6.5.21。
- 微信 App Android 最低版本要求：6.5.19。
- 小程序基础库最低版本要求：1.7.0。

> 说明：
> 通过 wx.getSystemInfo 可以获取当前基础库版本信息。

## 使用限制

出于政策和合规的考虑，微信暂时没有放开所有小程序对 <live-pusher> 和 <live-player> 标签的支持：

- 个人账号和企业账号的小程序暂时只开放如下表格中的类目：

| 一级类目/主体类型 | 二级类目 | 资质要求 | 类目适用范围 | 小程序直播内容场景 |
| --- | --- | --- | --- | --- |
| 社交 | 直播 | （3选1）：<br>1.《信息网络传播视听节目许可证》<br>2.《网络文化经营许可证》(经营范围含网络表演）<br>3.《统一社会信用代码》及《情况说明函件》（适用于政府主体） | 适用于提供在线直播等服务<br>注：<br>1.如提供时政信息服务，需补充：时政信息类目<br>2.选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 | 涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右 |
| 教育 | 在线视频课程 | （5选1）：<br>1.《事业单位法人证书》（适用公立学校）<br>2. 区、县级教育部门颁发的《民办学校办学许可证》（适用培训机构）<br>3.《信息网络传播视听节目许可证》<br>4. 全国校外线上培训管理服务平台备案<br>5. 教育部门的批准文件 | 适用于教育行业提供，网课、在线培训、讲座等教育类视频/直播等服务 | 网课、在线培训、讲座等教育类直播 |
| 医疗 | 互联网医院 | （2选1）：<br>1. 卫生健康部门的《设置医疗机构批准书》；<br>2. 合作医院的《医疗机构执业许可证》与执业登记机关的审核合格文件 | 适用于互联网医院主体/医疗服务平台提供在线看诊、疾病咨询等线上医疗服务 | 问诊、大型健康讲座等直播 |
| | 公立医疗机构 | 《医疗机构执业许可证》与《事业单位法人证书》 | 适用于公立医疗机构提供的就医、健康咨询/问诊、医疗保健信息等服务 | |
| 金融 | 银行 | （2选1）：<br>1.《金融许可证》<br>2.《金融机构许可证》 | 适用于提供银行业务在线服务或交易等服务 | 金融产品视频客服理赔、金融产品推广直播等 |
| | 信托 | （2选1）： | 适用于提供信托理财业务在线服务或 | |

| | | | | |
|---|---|---|---|---|
| | | 1.《金融许可证》<br>2.《金融机构许可证》 | 交易等服务 | |
| | 公募基金 | （3选1）：<br>1.《经营证券期货业务许可证》且业务范围必须包含“基金”<br>2.《基金托管业务许可证》<br>3.《基金销售业务资格证书》 | 适用于基金管理公司从事股票、债券等金融工具的投资服务 | |
| | 私募基金 | （2选1）：<br>1.《私募基金备案证明》<br>2.《私募投资基金管理人登记证书》 | 仅适用于私募基金展示、介绍、咨询等服务<br>注：暂不支持涉及私募产品公开募集或在线交易等服务 | |
| | 证券/期货 | 《经营证券期货业务许可证》 | 适用于提供证券资讯、证券咨询、证券期货经营等的在线服务 | |
| | 证券、期货投资咨询 | （2选1）：<br>1.《证券投资咨询业务资格证书》<br>2.《经营证券期货业务许可证》 | 适用于提供证券、期货投资等在线咨询服务 | |
| | 保险 | （8选1）：<br>1.《保险公司法人许可证》<br>2.《经营保险业务许可证》<br>3.《保险营销服务许可证》<br>4.《经营保险代理业务许可证》<br>5.《经营保险经纪业务许可证》<br>6.《经营保险公估业务许可证》<br>7.《经营保险资产管理业务许可证》<br>8.《保险兼业代理业务许可证》 | 适用于提供保险业务在线服务或交易等服务 | |
| | 征信业务 | （2选1）：<br>1.经营个人征信业务：《个人征信业务经营许可证》、《营业执照》<br>2.经营企业征信业务：经所在地的中国人民银行及其派出机构备案的《企业征信业务经营备案证》、《营业执照》 | 适用于银行或征信机构提供征信业务服务，包括：信贷记录、逾期记录、失信人查询等 | |
| | 新三板信息服务平台 | 全国中小企业股份转让系统有限责任公司的书面许可与《非经营性互联网信息服务备案核准》 | 适用于提供新三板信息行情资讯等服务 | |
| | 股票信息服务平台（港股/美股） | 《非经营性互联网信息服务备案核准》 | 适用于提供港股、美股行情资讯、行情分析等服务<br>注：如提供股票交易服务，需补充：金融业-证券/期货类目 | |
| | 消费金融 | 银监会核准开业的审批文件与《金融许可证》与《营业执照》 | 适用于提供消费金融线上服务或交易等服务 | |
| 汽车 | 汽车预售服务 | （3选1）：<br>1.汽车厂商：《营业执照》与《工信部道路机动车辆生产企业准入许可》<br>2.汽车经销商/4s店：《营业执照》与《厂商授权销售文件》与《工信部道路机动车辆生产企业准入许可》<br>3.下属子/分公司：《营业执照》与《工信部道路机动车辆生产企业准入许可》与《股权关系证明函》（含双方盖章） | 适用于提供汽车在线预付款等服务<br>注：平台暂不支持在线整车销售，如涉及整车销售服务，建议改为价格指导或移除相关功能 | 汽车预售、推广直播 |

| | | | | |
|---|---|---|---|---|
| 政府主体帐号 | – | – | – | 政府相关工作推广直播、领导讲话直播等 |
| 工具 | 视频客服 | – | 适用于提供企业售后客服一对一视频等服务 | 不涉及以上几类内容的一对一视频客服服务，如企业售后一对一视频服务等 |

> **说明：**
> 可申请直播标签的小程序类目以 微信文档 说明为主，小程序类目的资质要求详见 非个人主体类目申请。

- 符合类目要求的小程序，需要在小程序管理后台的 “【开发】>【接口设置】” 中自助开通该组件权限，如下图所示：


> **注意：**
> 如果以上设置都正确，但小程序依然不能正常工作，可能是微信内部的缓存没更新，请删除小程序并重启微信后，再进行尝试。

## 属性定义

| 属性名 | 类型 | 默认值 | 说明 |
|---|---|---|---|
| src | String | – | 用于音视频下行的播放 URL，支持 RTMP、FLV 等协议 |
| mode | String | live | live，RTC |
| autoplay | Boolean | false | 是否自动播放 |
| muted | Boolean | false | 是否静音 |
| orientation | String | vertical | vertical，horizontal |

| 属性名 | 类型 | 默认值 | 说明 |
| --- | --- | --- | --- |
| object-fit | String | contain | contain，fillCrop |
| background-mute | Boolean | false | 当微信切到后台时，是否关闭播放声音 |
| min-cache | Number | 1 | 最小缓冲延迟，单位：秒 |
| max-cache | Number | 3 | 最大缓冲延迟，单位：秒 |
| bindstatechange | EventHandler | - | 用于指定一个 javascript 函数来接受播放器事件 |
| bindfullscreenchange | EventHandler | - | 用于指定一个 javascript 函数来接受全屏事件 |
| debug | Boolean | false | 是否开启调试模式 |

## 示例代码

```
<view id='video-box'>
<live-player
wx:for="{{player}}"
id="player_{{index}}"
mode="RTC"
object-fit="fillCrop"
src="{{item.playUrl}}"
autoplay='true'
bindstatechange="onPlay">
</live-player>
</view>
```

## 超低时延

<live-player> 的 RTC 模式支持500ms以内的超低时延链路，可以应用在视频通话和远程遥控等场景中，要使用超低时延播放，需要注意如下几点：

- 推流端如果是微信小程序，请使用 <live-pusher> 的 RTC 模式。
- 推流端如果是 iOS 或者 Android SDK，请使用 setVideoQuality 的 MAIN_PUBLISHER 模式。
- <live-player> 的 `min-cache` 和 `max-cache` 请不要自行设置，使用默认值。
- 播放地址请使用超低延时播放地址，也就是带了防盗链签名的 `rtmp://` 地址，如下：

| 对比项目 | 示例 | 时延 |
| --- | --- | --- |
| 普通直播 URL | rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc | > 2s |
| 超低延时 URL | rtmp://3891.liveplay.myqcloud.com/live/3891_test_clock_for_rtmpacc?bizid=bizid&txTime=5FD4431C&txSerect=20e6d865f462dff61ada209d53c71cf9 | < 500ms |

## 属性详解

- **src**
  用于音视频下行的播放 URL，支持 RTMP 协议（URL 以 “rtmp://” 打头）和 FLV 协议（URL 以 “http://” 打头且以 “.flv” 结尾），腾讯云推流 URL 的获取方法见 DOC。

  > 说明：
  > <live-player> 标签是不支持 HLS(m3u8) 协议的，因为 <video> 已经支持 HLS(m3u8) 播放协议了。但直播观看不推荐使用 HLS(m3u8) 协议，延迟要比 RTMP 和 FLV 协议高一个数量级。

- **mode**
  live 模式主要用于直播类场景，例如赛事直播、在线教育和远程培训等。该模式下，小程序内部的模块会优先保证观看体验的流畅，通过调整 `min-cache` 和 `max-cache` 属性，您可以调节观众（播放）端所感受到的时间延迟的大小，文档下面会详细介绍这两个参数。

  RTC 则主要用于双向视频通话或多人视频通话场景，例如金融开会、在线客服、车险定损和培训会议等。在此模式下，对 `min-cache` 和 `max-cache` 的设置不会起作用，因为小程序内部会自动将延迟控制在一个很低的水平（500ms左右）。

- **min-cache 和 max-cache**
  这两个参数分别用于指定观看端的最小缓冲时间和最大缓冲时间。所谓**缓冲时间**，是指播放器为了缓解网络波动对观看流畅度的影响而引入的一个“蓄水池”，当来自网络的数据包出现卡顿甚至停滞的时候，“蓄水池”里的紧急用水可以让播放器还能坚持一小段时间，只要在这个短暂的时间内网速恢复正常，播放器就可以源源不断地渲染出流畅而平滑的视频画面。

  “蓄水池”里的水越多，抗网络波动的能力就越强，但代价就是观众端的延迟就越大，所以要在不同的场景下，使用不同的配置来达到体验上的平衡：

  - 码率比较低（1Mbps左右，画面以人物为主）的直播流，`min-cache = 1`，`max-cache = 3`较合适。
  - 码率比较高（2Mbps - 3Mbps的高清游戏画面为主）的直播流，`min-cache = 3`，`max-cache = 5`较合适。

  RTC 模式下这两个参数是无效的。

- **orientation**
  画面渲染角度，horizontal 代表是原始画面方向，vertical 代表向右旋转90度。

- **object-fit**
  画面填充模式，contain 代表把画面显示完成，但如果视频画面的宽高比和 <live-player> 标签的宽高比不一致，那么您将看到黑边。
  fillCrop 代表把屏幕全部撑满，但如果视频画面的宽高比和 <live-player> 标签的宽高比不一致，那么画面中多余的部分会被裁剪掉。

- **background-mute**
  微信切到后台以后是否继续播放声音，用于避免锁屏对于当前小程序正在播放的视频内容的影响。

- **sound-mode**
  设置播放模式，可设值为： ear 与 speaker，ear 代表使用听筒播放， speaker 代表使用扬声器，默认为扬声器

- **debug**
  为了很好地调试音视频的相关功能，小程序为 live-pusher 标签提供了 debug 模式，开始 debug 模式之后，原本用于渲染视频画面的窗口上，会显示一个半透明的 log 窗口，用于展示各项音视频指标和事件，降低您调试相关功能的难度，具体使用方法我们在 FAQ 中有详细说明。

## 对象操作

- **wx.createLivePlayerContext()**
  通过 wx.createLivePlayerContext() 可以将 <live-player> 标签和 javascript 对象关联起来，之后即可操作该对象。

- **play**
  开始播放，如果 <live-player> 的 autoplay 属性设置为 false（默认值），那么就可以使用 play 来手动启动播放。
- **stop**
  停止播放。
- **pause**
  暂停播放，停留在最后画面。
- **resume**
  继续播放，与 pause 成对使用。
- **mute**
  设置静音。
- **requestFullScreen**
  进入全屏幕。
- **exitFullScreen**
  退出全屏幕。

```
var player = wx.createLivePlayerContext('pusher');
player.requestFullScreen({
success: function(){
console.log('enter full screen mode success!')
}
fail: function(){
console.log('enter full screen mode failed!')
}
complete: function(){
console.log('enter full screen mode complete!')
}
});
```

## 内部事件

通过 **<live-player>** 标签的 **bindstatechange** 属性可以绑定一个事件处理函数，该函数可以监听推流模块的内部事件和异常通知。

**1. 关键事件**

| code | 事件定义 | 含义说明 |
| --- | --- | --- |
| 2001 | PLAY_EVT_CONNECT_SUCC | 已经连接到云端服务器 |
| 2002 | PLAY_EVT_RTMP_STREAM_BEGIN | 服务器开始传输音视频数据 |
| 2003 | PLAY_EVT_RCV_FIRST_I_FRAME | 网络接收到首段音视频数据 |
| 2004 | PLAY_EVT_PLAY_BEGIN | 视频播放开始，可以在收到此事件之前先用默认图片代表等待状态 |
| 2006 | PLAY_EVT_PLAY_END | 视频播放结束 |
| 2007 | PLAY_EVT_PLAY_LOADING | 进入缓冲中状态，此时播放器在等待或积攒来自服务器的数据 |

| code | 事件定义 | 含义说明 |
| --- | --- | --- |
| -2301 | PLAY_ERR_NET_DISCONNECT | 网络连接断开，且重新连接亦不能恢复，播放器已停止播放 |

> 说明：
> 播放 HTTP:// 打头的 FLV 协议地址时，如果观众遇到播放中直播流断开的情况，小程序是不会抛出 PLAY_EVT_PLAY_END 事件的，这是因为 FLV 协议中没有定义停止事件，所以只能通过监听 PLAY_ERR_NET_DISCONNECT 来替代之。

**2. 警告事件**

内部警告并非不可恢复的错误，小程序内部的音视频 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，例如：

| 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。 |
| 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）。 |

**3. 示例代码**

```
Page({
onPlay: function(ret) {
if(ret.detail.code == 2004) {
console.log('视频播放开始',ret);
}
},

/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
//...
}
})
```

## 特别说明

1. <live-player> 组件是由客户端创建的原生组件，它的层级是最高的，可以使用 <cover-view> 和 <cover-image> 覆盖在上面。
2. 请勿在 <scroll-view> 中使用 <live-player> 组件。
3. <live-player>组件的 RTC 模式有并发播放限制，目前最多同时10路并发播放。

> 说明：
> 设置该限制原因并非技术能力限制，而是希望您只考虑在互动场景中使用（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）。

# Web（H5）播放器

最近更新时间：2021-06-28 16:55:16

## 功能介绍

腾讯云 Web 超级播放器 TCPlayerLite 是为了解决在手机浏览器和 PC 浏览器上播放音视频流的问题，它使您的视频内容可以不依赖用户安装 App，就能在朋友圈和微博等社交平台进行传播。本文档适合有一定 Javascript 语言基础的开发人员阅读。
以下视频将为您讲解腾讯云播放器 SDK 的 Web 播放器的功能特性以及对接攻略：

点击查看视频

## 基础知识

对接前需要了解如下基础知识：

- **直播和点播**
  直播视频源是实时的，一旦主播停播，直播地址就失去意义，而且由于是实时直播，所以播放器在播直播视频的时候是没有进度条的。
  点播视频源是某个服务器上的文件，只要文件没有被提供方删除，就可以随时播放， 而且由于整个视频都在服务器上，所以播放器在播点播视频的时候是有进度条的。

- **协议支持**
  TCPlayerLite 的视频播放能力本身不是网页代码实现的，而是靠浏览器支持，所以其兼容性不像我们想象的那么好，因此，**不是所有的手机浏览器都能有符合预期的表现**。一般用于网页直播的视频源地址是以 M3U8 结尾的地址，我们称其为 HLS (HTTP Live Streaming)，这是苹果推出的标准，目前各种手机浏览器产品对这种格式的兼容性也最好，但它有个问题：延迟比较大，一般是20s – 30s左右的延迟。

对于 PC 浏览器，因为其目前还没有抛弃 Flash 控件，而 Flash 控件支持的视频源格式较多，并且浏览器上的 Flash 控件都是 Adobe 自己研发，所以兼容性很好。

| 视频协议 | 用途 | URL 地址格式 | PC 浏览器 | 移动浏览器 |
| --- | --- | --- | --- | --- |
| WebRTC | 只适用直播 | webrtc://xxx.liveplay.myqcloud.com/live/xxx | 支持 | 支持 |
| HLS（M3U8） | 可用于直播 | http://xxx.liveplay.myqcloud.com/xxx.m3u8 | 支持 | 支持 |
| HLS（M3U8） | 可用于点播 | http://xxx.vod.myqcloud.com/xxx.m3u8 | 支持 | 支持 |
| FLV | 可用于直播 | http://xxx.liveplay.myqcloud.com/xxx.flv | 支持 | 不支持 |
| FLV | 可用于点播 | http://xxx.vod.myqcloud.com/xxx.flv | 支持 | 不支持 |
| RTMP | 只适用直播 | rtmp://xxx.liveplay.myqcloud.com/live/xxx | 支持 | 不支持 |
| MP4 | 只适用点播 | http://xxx.vod.myqcloud.com/xxx.mp4 | 支持 | 支持 |

⚠ 注意：

- 播放 RTMP 格式的视频必须启用 Flash，目前浏览器默认禁用 Flash，需用户手动开启。
- 在不支持 WebRTC 的浏览器环境，传入播放器的 WebRTC 地址会自动进行协议转换来更好的支持媒体播放，默认在移动端转换为 HLS，PC 端转换为 FLV。

**功能支持**

| 功能\浏览器 | Chrome | Firefox | Edge | QQ 浏览器 | Mac Safari | iOS Safari | iOS 微信 QQ | Android Chrome | Android 微信、QQ | 手机 QQ 浏览器 | IE 8,9,10,11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 设置封面 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 多清晰度支持 | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | × | ✓ |
| 定制错误提示语 | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | ✓ |
| 快直播 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |

## 对接攻略

### Step1. 页面准备工作

在需要播放视频的页面（PC 或 H5）中引入初始化脚本。

```
<script src="https://web.sdk.qcloud.com/player/tcplayerlite/release/v2.4.1/TcPlayer-2.4.1.js" charset="utf-8"></script>;
```

建议在使用播放器 SDK 的时候自行部署资源，点击下载播放器资源。

如果您部署的地址为 `aaa.xxx.ccc`，在合适的地方引入播放器脚本文件：

```
<script src="aaa.xxx.ccc/TcPlayer-2.4.1.js"></script>
```

> ⚠ 注意：
> 直接用本地网页无法调试，Web 播放器无法处理该情况下的跨域问题。

### Step2. 在 HTML 中放置容器

在需要展示播放器的页面位置加入播放器容器，即放一个 div 并命名，例如 `id_test_video`，视频画面都会在容器里渲染。对于容器的大小控制，您可以使用 div 的属性进行控制，示例代码如下：

```
<div id="id_test_video" style="width:100%; height:auto;"></div>
```

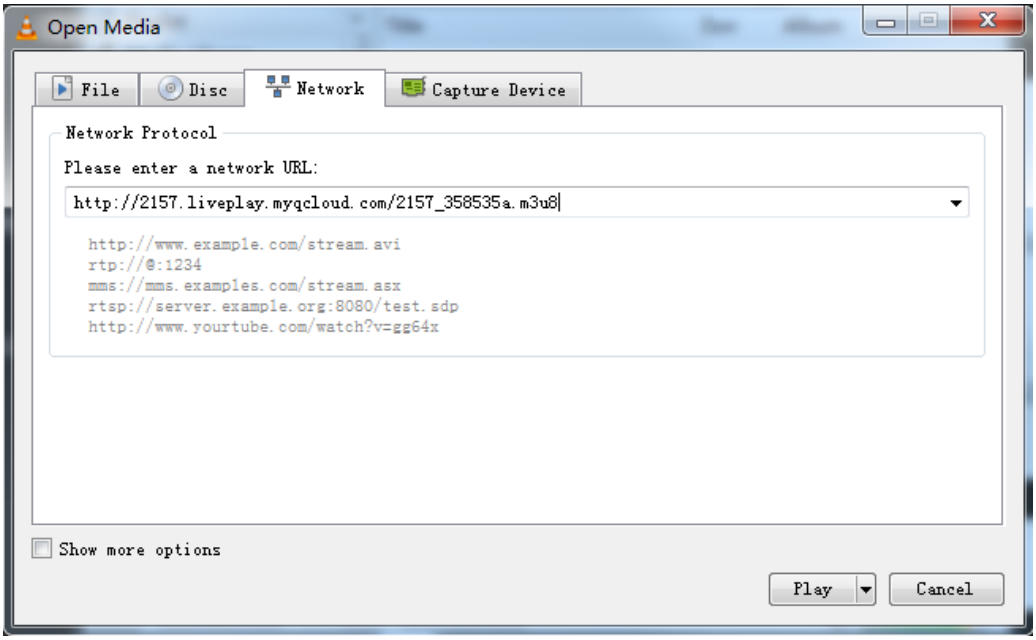
### Step3. 对接视频播放

编写 Javascript 代码，作用是去指定的 URL 地址拉取音视频流，并将视频画面呈现到添加的容器内。

#### 3.1 简单播放

如下是一个 直播格式的 URL 地址，使用 HLS（M3U8）协议，如果主播在直播中，则用 VLC 等播放器是可以直接打开该 URL 进行观看的：

```
http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 // m3u8 播放地址
```




如果要在手机浏览器上播放该 URL 的视频，则 Javascript 代码如下：

```
var player = new TcPlayer('id_test_video', {
"m3u8": "http://2157.liveplay.myqcloud.com/2157_358535a.m3u8", //请替换成实际可用的播放地址
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"width" : '480',//视频的显示宽度，请尽量使用视频分辨率宽度
"height" : '320'//视频的显示高度，请尽量使用视频分辨率高度
});
```

这段代码可以支持在 PC 及手机浏览器上播放 HLS（M3U8）协议的直播视频，虽然 HLS（M3U8）协议的视频兼容性不错，但部分 Android 手机依然不支持，其延迟较高，大约20秒以上的延迟。

### 3.2 实现更低延迟

PC 浏览器支持 Flash，其 Javascript 代码如下：

```
var player = new TcPlayer('id_test_video', {
"m3u8": "http://2157.liveplay.myqcloud.com/2157_358535a.m3u8",
"flv": "http://2157.liveplay.myqcloud.com/live/2157_358535a.flv", //增加了一个 flv 的播放地址，用于PC平台的播放 请替换成实际可用的播放地址
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"width" : '480',//视频的显示宽度，请尽量使用视频分辨率宽度
"height" : '320'//视频的显示高度，请尽量使用视频分辨率高度
});
```

这段代码中增加了 FLV 的播放地址，Web 播放器如果发现当前的浏览器是 PC 浏览器，会主动选择 FLV 链路，从而实现更低的延迟。如果对延迟有更高的要求，可以使用 WebRTC 拉流地址，基于 WebRTC 的播放系统可以实现超低延迟（500ms），前提条件是拉流地址都是可以

出流的，如果您使用腾讯云的视频直播服务，则无需考虑，因为腾讯云的直播频道默认支持 WebRTC、FLV、RTMP 和 HLS（M3U8）播放协议。

**无法播放怎么办?**

如果您发现视频无法播放，可能存在如下原因：

- **原因一：视频源有问题**
  如果是直播 URL，则需要检查主播是否已经停止推流，可以用浮窗提示观众：“主播已经离开”。请参见 直播推流。
  如果是点播 URL，则需要检查要播放的文件是否还存在于服务器上（如播放地址是否已经从点播系统移除）。

- **原因二：本地网页调试**
  目前 TCPlayerLite 不支持本地网页调试（即通过 `file://` 协议打开视频播放的网页），因为浏览器有跨域安全限制，所以在 Windows 系统上放置一个 test.html 文件来进行测试是无法播放的，需要将其上传到服务器上进行测试。而前端工程师可以通过反向代理的方式，对线上页面进行本地代理以实现本地调试，这是主流的本地调试方法。

- **原因三：手机兼容问题**
  普通的手机浏览器只支持 HLS（M3U8）协议的播放，不支持 FLV 和 RTMP 协议，最新版本的 QQ 浏览器支持 FLV 协议的播放。

- **原因四：跨域安全问题**
  PC 浏览器的视频播放基于 Flash 控件实现，但 **Flash 控件会做跨域访问检查**，如果播放视频所存放的服务器没有部署跨域策略，则会出现问题。
  解决方法：在视频存储服务器根域名下添加跨域配置文件 `crossdomain.xml`，并配置 Flash swf 所在域名，以允许 Flash 和 JavaScript 跨域播放视频。
  播放器的 Flash swf 文件默认存放在 `imgcache.qq.com` 域名下，如需部署到自己的服务器上，可自行下载并部署：swf 文件地址。
  如果是在域名限制区域，需要的播放器的 Flash swf 文件默认存放在 `cloudcache.tencent-cloud.com` 域名下，并且在播放器初始化的时候传递 flashUrl。

```
<cross-domain-policy>
<allow-access-from domain="*.*.com" secure="false"/>
</cross-domain-policy>
```

## Step4. 给播放器设置封面

设置封面涉及到 poster 属性，下面将详细介绍 poster 属性的使用方法。

> ⚠ 注意：
> 封面功能在部分移动端播放环境下可能失效，通常是由于移动端 webview 劫持视频播放造成的，需要 webview 支持 video 叠加元素或者放开劫持视频播放。相关详细说明请参见 常见问题。

### 4.1 简单设置封面

poster 支持传入图片地址作为播放器的封面，在播放器区域内居中，并且以图片的实际分辨率进行显示。

```
"poster" : "http://www.test.com/myimage.jpg"
```

### 4.2 设置封面样式

poster 支持传入一个对象，在对象中可以对封面的展现样式（style）和图片地址（src）进行设置。

style 支持的样式如下：

- default：居中并且以图片的实际分辨率进行显示。

- stretch：拉伸铺满播放器区域，图片可能会变形。
- cover：优先横向等比拉伸铺满播放器区域，图片某些部分可能无法显示在区域内。

```
"poster" : {"style":"stretch", "src":"http://www.test.com/myimage.jpg"}
```

#### 4.3 实现用例

使用 cover 方式显示封面。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：
视频封面

> ⚠ 注意：
>
> - 在某些移动端设置封面会无效，具体说明请参见 常见问题。
> - 以上示例链接仅用于文档演示，请勿用于生产环境。

### Step5. 多清晰度支持

#### 5.1 原理介绍

同腾讯视频，Web 播放器支持多清晰度，如下图所示：


**播放器本身是没有能力去改变视频清晰度的**，视频源只有一种清晰度，称之为原画，而原画视频的编码格式和封装格式多种，Web 端无法支持播放所有的视频格式，如点播支持以 H.264 为视频编码，MP4 和 FLV 为封装格式的视频。

**多清晰度的实现依赖于视频云：**

- 对于直播，来自主播端的原始视频会在腾讯云进行实时转码，分出多路转码后的视频，每一路视频都有其对应的地址，例如“高清-HD”和“标清-SD”，地址格式如下：

```
http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 // 原画
http://2157.liveplay.myqcloud.com/2157_358535a_900.m3u8 // 高清
http://2157.liveplay.myqcloud.com/2157_358535a_550.m3u8 // 标清
```

- 对于点播，一个视频文件上传到腾讯云后，您可以对该视频文件进行转码，产生其它几种清晰度的视频，例如“高清-HD”和“标清-SD”，地址格式如下：

```
http://200002949.vod.myqcloud.com/200002949_b6ffc.f240.m3u8 // 原画，用转码后的超清替换
http://200002949.vod.myqcloud.com/200002949_b6ffc.f230.av.m3u8 // 高清
http://200002949.vod.myqcloud.com/200002949_b6ffc.f220.av.m3u8 // 标清
```

> ⚠ 注意：
> 上传后的原始视频是未经过腾讯云转码的，不能直接用于播放。

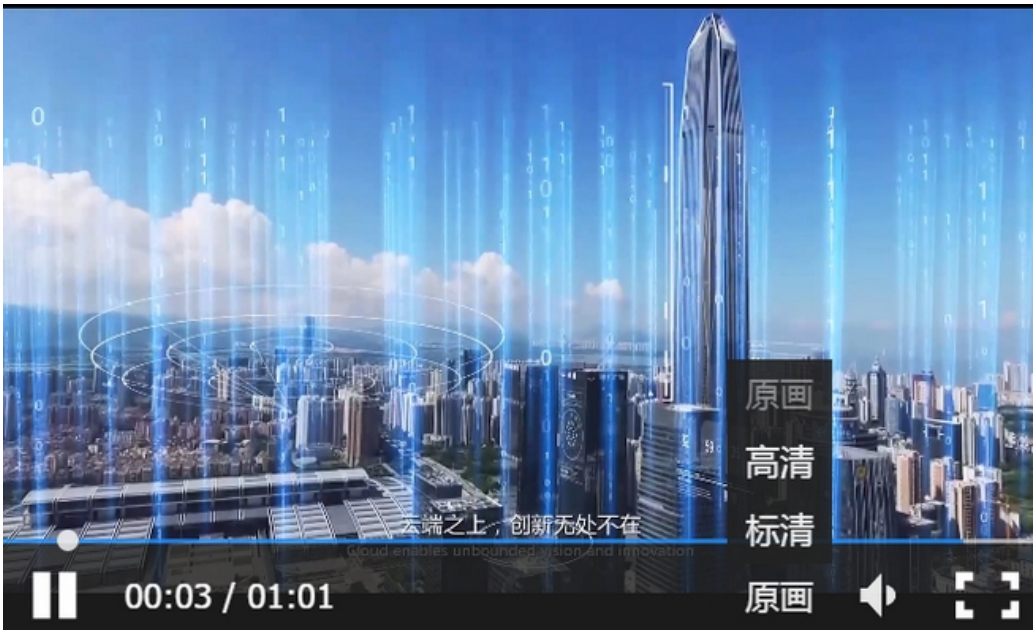
#### 5.2 代码实现

多清晰度支持的代码实现如下所示：

```
var player = new TcPlayer('id_test_video', {
"m3u8" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f240.m3u8",//请替换成实际可用的播放地址
"m3u8_hd" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f230.av.m3u8",
"m3u8_sd" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f220.av.m3u8",
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
});
```

#### 5.3 实现用例

使用多种分辨率的设置及切换功能。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：分辨率切换
正常情况将看到如下效果：




> ⚠ 注意：
> - PC 端现已支持多种清晰度播放及切换的功能，移动端尚未支持。
> - 以上示例链接仅用于文档演示，请勿用于生产环境

### Step6. 定制错误提示语

Web 播放器支持提示语定制。

#### 6.1 代码实现

如下是让播放器支持自定义提示语的核心代码，主要在 wording 属性上设置提示语。

```
var player = new TcPlayer('id_test_video', {
"m3u8" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.m3u8",//请替换成实际可用的播放地址
```

```
"autoplay" : true, //iOS 下 safari 浏览器是不开放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"wording": {
2032: "请求视频失败，请检查网络",
2048: "请求m3u8文件失败，可能是网络错误或者跨域问题"
}
});
```

### 6.2 实现用例

视频播放失败，同时使用自定义提示文案的功能。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：

```
https://web.sdk.qcloud.com/player/tcplayerlite/tcplayer-error.html
```

> ⚠ 注意：
> 以上示例链接仅用于文档演示，请勿用于生产环境。

### 6.3 错误码表

| Code | 提示语 | 说明 |
| --- | --- | --- |
| 1 | 网络错误，请检查网络配置或者播放链接是否正确。 | H5 提示的错误。 |
| 2 | 网络错误，请检查网络配置或者播放链接是否正确。 | 视频格式 Web 播放器无法解码。<br>H5 提示的错误。 |
| 3 | 视频解码错误。 | H5 提示的错误。 |
| 4 | 当前系统环境不支持播放该视频格式。 | H5 提示的错误。 |
| 5 | 当前系统环境不支持播放该视频格式。 | 播放器判断当前浏览器环境不支持播放传入的视频，可能是当前浏览器不支持 MSE 或者 Flash 插件未启用。 |
| 10 | 请勿在 file 协议下使用播放器，可能会导致视频无法播放。 | - |
| 11 | 使用参数有误，请检查播放器调用代码。 | - |
| 12 | 请填写视频播放地址。 | - |
| 13 | 直播已结束，请稍后再来。 | RTMP 正常播放过程中触发事件（NetConnection.Connect.Closed）。<br>Flash 提示的错误。 |
| 1001 | 网络错误，请检查网络配置或者播放链接是否正确。 | 网络已断开（NetConnection.Connect.Closed）。<br>Flash 提示的错误。 |
| 1002 | 获取视频失败，请检查播放链接是否有效。 | 拉取播放文件失败（NetStream.Play.StreamNotFound），可能是服务器错误或者视频文件不存在。<br>Flash 提示的错误。 |
| 2001 | 调用 WebRTC 接口失败 | 播放 WebRTC 时设置 sdp 失败提示的错误 |
| 2002 | 调用拉流接口失败 | 播放 WebRTC 时调用拉流接口失败提示的错误 |

| Code | 提示语 | 说明 |
| --- | --- | --- |
| 2003 | 连接服务器失败，并且连接重试次数已超过设定值 | 播放 WebRTC 时提示的错误，可用于确定是否为停止推流状态 |
| 2032 | 获取视频失败，请检查播放链接是否有效。 | Flash 提示的错误。 |
| 2048 | 无法加载视频文件，跨域访问被拒绝。 | 请求 M3U8 文件失败，可能是网络错误或者跨域问题。<br>Flash 提示的错误。 |

说明：

- Code 1 – 4 对应的是 H5 原生事件。
- 由于 Flash 的黑盒特性以及 H5 视频播放标准的不确定性，错误提示语会不定期更新。

## 源码参考

如下是一个线上示例代码，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：播放示例

注意：

以上示例链接仅用于文档演示，请勿用于生产环境。

## 参数列表

播放器支持的所有参数，如下所示：

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| webrtc | String | 无 | 原画 WebRTC 播放 URL。<br>示例：webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1 |
| webrtc_hd | String | 无 | 高清 WebRTC 播放 URL。<br>示例：<br>webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1_hd |
| webrtc_sd | String | 无 | 标清 WebRTC 播放 URL。<br>示例：<br>webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1_sd |
| m3u8 | String | 无 | 原画 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 |
| m3u8_hd | String | 无 | 高清 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535ahd.m3u8 |
| m3u8_sd | String | 无 | 标清 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535asd.m3u8 |
| flv | String | 无 | 原画 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535a.flv |
| flv_hd | String | 无 | 高清 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535ahd.flv |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| flv_sd | String | 无 | 标清 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535asd.flv |
| mp4 | String | 无 | 原画 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.mp4 |
| mp4_hd | String | 无 | 高清 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f40.mp4 |
| mp4_sd | String | 无 | 标清 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f20.mp4 |
| rtmp | String | 无 | 原画 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88 |
| rtmp_hd | String | 无 | 高清 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88hd |
| rtmp_sd | String | 无 | 标清 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88sd |
| width | Number | 无 | **必选**，设置播放器宽度，单位为像素。<br>示例：640 |
| height | Number | 无 | **必选**，设置播放器高度，单位为像素。<br>示例：480 |
| volume | Number | 0.5 | 设置初始音量，范围：0到1 [v2.2.0+]。<br>示例：0.6 |
| live | Boolean | false | **必选**，设置视频是否为直播类型，将决定是否渲染时间轴等控件，以及区分点直播的处理逻辑。<br>示例：true |
| autoplay | Boolean | false | 是否自动播放。<br>（**备注：该选项只对大部分 PC 平台生效**）<br>示例：true |
| poster | String / Object | 无 | 预览封面，可以传入一个图片地址或者一个包含图片地址 src 和显示样式 style 的对象。<br>style 可选属性：<br>• default 居中1：1显示。<br>• stretch 拉伸铺满播放器区域，图片可能会变形。<br>• cover 优先横向等比拉伸铺满播放器区域，图片某些部分可能无法显示在区域内。<br>示例： "http://www.test.com/myimage.jpg" 或者<br>{"style": "cover", "src": http://www.test.com/myimage.jpg}<br>[v2.3.0+] |
| controls | String | "default" | default 显示默认控件，none 不显示控件，system 移动端显示系统控件。<br>（备注：如果需要在移动端使用系统全屏，就需要设置为 system。默认全屏方案是使用 Fullscreen API + 伪全屏的方式，在线示例 ）<br>示例："system" |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| systemFullscreen | Boolean | false | 开启后，在不支持 Fullscreen API 的浏览器环境下，尝试使用浏览器提供的 webkitEnterFullScreen 方法进行全屏，如果支持，将进入系统全屏，控件为系统控件。<br>示例：true |
| flash | Boolean | true | 是否优先使用 Flash 播放视频。<br>（**备注：该选项只对 PC 平台生效**[v2.2.0+]）<br>示例：true |
| flashUrl | String | 无 | 可以设置 flash swf url。<br>（**备注：该选项只对 PC 平台生效** [v2.2.1+]） |
| h5_flv | Boolean | false | 是否启用 flv.js 的播放 flv。启用时播放器将在支持 MSE 的浏览器下，采用 flv.js 播放 flv，然而并不是所有支持 MSE 的浏览器都可以使用 flv.js，所以播放器不会默认开启这个属性，[v2.2.0+]。<br>示例：true |
| x5_player | Boolean | false | 是否启用 TBS 的播放 flv 或 hls 。启用时播放器将在 TBS 模式下(例如 Android 的微信、QQ 浏览器），将 flv 或 hls 播放地址直接赋给 <video> 播放。TBS 视频能力 [v2.2.0+]。<br>示例： true |
| x5_type | String | 无 | 通过 video 属性 “x5-video-player-type” 声明启用同层 H5 播放器，支持的值：h5-page (该属性为 TBS 内核实验性属性，非 TBS 内核不支持)，TBS H5 同层播放器接入规范。<br>示例："h5-page" |
| x5_fullscreen | String | 无 | 通过 video 属性 “x5-video-player-fullscreen” 声明视频播放时是否进入到 TBS 的全屏模式，支持的值：true (该属性为 TBS 内核实验性属性，非 TBS 内核不支持) 。<br>示例："true" |
| x5_orientation | Number | 无 | 通过 video 属性 “x5-video-orientation” 声明 TBS 播放器支持的方向，可选值：0（landscape 横屏），1：（portraint竖屏），2：（landscape |
| wording | Object | 无 | 自定义文案。<br>示例：{ 2032: '请求视频失败，请检查网络'} |
| clarity | String | 'od' | 默认播放清晰度 [v2.2.1+]。<br>示例：clarity: 'od' |
| clarityLabel | Object | {od: '超清', hd: '高清', sd: '标清'} | 自定义清晰度文案 [v2.2.1+]。<br>示例：clarityLabel: {od: '蓝光', hd: '高清', sd: '标清'}。 |
| listener | Function | 无 | 事件监听回调函数，回调函数将传入一个 JSON 格式的对象。<br>示例：function(msg){<br>//进行事件处理<br>} |
| pausePosterEnabled | Boolean | true | 暂停时显示封面 [v2.3.0+]。 |
| preload | String | 'auto' | 配置 video 标签的 preload 属性，只有部分浏览器生效[v2.3.0+]。 |
| hlsConfig | Object | 无 | hls.js 初始化配置项 [v2.3.0+]。 |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| flvConfig | Object | 无 | flv.js 初始化配置项 [v2.3.1+]。 |
| webrtcConfig | Object | 无 | webrtc 初始化配置项 [v2.4.1+]。<br>支持通过 streamType 指定拉流类型，默认拉取音视频，可选单独拉取视频或单独拉取音频，streamType 可选属性：<br>• auto：拉取视频流和音频流<br>• video：仅拉取视频流<br>• audio：仅拉取音频流<br>示例：`webrtcConfig: { streamType: 'video' }` |

⚠ 注意：

- WebRTC 快直播播放地址支持两种格式，除 `webrtc://domain/AppName/StreamName?txSecret=XXX&txTime=XXX` 以外，还支持 `http://domain/AppName/StreamName.sdp?txSecret=XXX&txTime=XXX` 格式的播放地址，但是需要配置播放域名 CNAME 到 `overseas-webrtc.liveplay.myqcloud.com` 。
- 由于 Web 浏览器目前不支持标准 WebRTC 协议携带 B 帧播放，如果原始流存在 B 帧，则后台会自动进行转码去掉 B 帧，这样会引入额外的转码延迟，并产生转码费用。建议尽量不推包含 B 帧的流，移动直播 SDK，IOS 不支持引入 B 帧，安卓如果不开启 B 帧设置，默认是没有带 B 帧。如果使用 OBS 推流，可以通过设置，关闭 B 帧。

## 实例方法列表

播放器实例支持的方法，如下所示：

| 方法 | 参数 | 返回值 | 说明 | 示例 |
| --- | --- | --- | --- | --- |
| play() | 无 | 无 | 开始播放视频。 | player.play() |
| pause() | 无 | 无 | 暂停播放视频。 | player.pause() |
| togglePlay() | 无 | 无 | 切换视频播放状态 。 | player.togglePlay() |
| mute(muted) | {Boolean} [可选] | true,false {Boolean} | 切换静音状态，不传参则返回当前是否静音。 | player.mute(true) |
| volume(val) | {int} 范围：0到1 [可选] | 范围：0到1 | 设置音量，不传参则返回当前音量 。 | player.volume(0.3) |
| playing() | 无 | true,false {Boolean} | 返回是否在播放中 。 | player.playing() |
| duration() | 无 | {int} | 获取视频时长 。<br>（备注：只适用于点播，需要在触发 loadedmetadata 事件后才可获取视频时长） | player.duration() |
| currentTime(time) | {int} [可选] | {int} | 设置视频播放时间点，不传参则返回当前播放时间点 。<br>（备注：只适用于点播 ） | player.currentTime() |

| 方法 | 参数 | 返回值 | 说明 | 示例 |
| --- | --- | --- | --- | --- |
| fullscreen(enter) | {Boolean} [可选] | true,false {Boolean} | 调用全屏接口(Fullscreen API)，不支持全屏接口时使用伪全屏模式，不传参则返回值当前是否是全屏。**（备注：移动端系统全屏没有提供 API，也无法获取系统全屏状态 ）** | player.fullscreen(true) |
| buffered() | 无 | 0到1 | 获取视频缓冲数据百分比。**（备注：只适用于点播）** | player.buffered() |
| destroy() | 无 | 无 | 销毁播放器实例[v2.2.1+]。 | player.destroy() |
| switchClarity() | {String} [必选] | 无 | 切换清晰度，传值 "od"、"hd"、"sd" [v2.2.1+]。 | player.switchClarity('od') |
| load(url) | {String} [必选] | 无 | 通过视频地址加载视频。**（备注：该方法只能加载对应播放模式下支持的视频格式，Flash 模式支持切换 RTMP、FLV、HLS 和 MP4 ，H5 模式支持 MP4、HLS 和 FLV（HLS、FLV 取决于浏览器是否支持）** [v2.2.2+]） | player.load(http://xxx.mp4) |

> **注意：**
> 以上方法必须是 TcPlayer 的实例化对象，且需要初始化完毕才可以调用（即 load 事件触发后）。

## 进阶攻略

下面介绍播放器 SDK 的进阶使用方法。

### ES Module

TCPlayerLite 提供了 ES Module 版本，module name 为 TcPlayer，下载地址：

```
https://web.sdk.qcloud.com/player/tcplayerlite/release/v2.4.1/TcPlayer-module-2.4.1.js
```

#### 开启优先 H5 播放模式

TCPlayerLite 采用 H5 <video> 和 Flash 相结合的方式来进行视频播放，根据不同的播放环境，播放器会选择默认最合适的播放方案。

虽然浏览器厂商已经开始逐步放弃对 Flash 插件的支持，但是在国内仍有大量的浏览器不支持 MSE，在播放 FLV 和 HLS（M3U8）时无法切换到 H5 <video> 模式，而播放 RTMP 必须使用 Flash。
因此，TCPlayerLite 默认优先启用 Flash 播放模式，如果在检测到 Flash 插件不可用的情况下，将采用 H5 <video> 进行播放。

> **说明：**
> 默认 Flash 模式的原因是 Flash 支持的视频格式最广，而 H5 <video> 默认只支持 MP4（H.264），在特定条件下才支持 HLS（M3U8）和 FLV。

**从2.2.0版本开始，提供了可以设置播放模式优先级的属性**，如果想优先采用 H5 <video> 播放模式，则需要把 Flash 属性设置为 False；如果 H5 <video> 不可用，则采用 Flash 播放；如果没有检测到 Flash 插件，则会提示“当前系统环境不支持播放该视频格式”。

#### 监听事件

TCPlayerLite 是采用 H5 <video> 和 Flash 相结合的方式来进行视频播放，由于两种方式播放视频时触发的事件不尽相同，所以我们以 H5 <video> 的规范，对 Flash 的播放事件做了一定程度的转换，以实现播放事件命名的统一， TcPlayer 对这两种播放方式所触发的原生事件进行了捕获和透传。

- H5 事件参考列表
- Flash 事件参考列表
- 统一后的事件列表

```
error
timeupdate
load
loadedmetadata
loadeddata
progress
fullscreen
play
playing
pause
ended
seeking
seeked
resize
volumechange
webrtcstatupdate
webrtcwaitstart
webrtcwaitend
webrtcstop
```

> **注意：**
>
> - 如果通过系统控制栏进行全屏，将无法监听到 fullscreen 事件。
> - Web 播放器的事件，依赖浏览器内置的解码器和 Flash 插件触发，Web 播放器仅透传事件。
> - Web 播放器监听不到直播停止推流的事件，需要通过额外的接口来确认推流状态，请参见 查询流状态。

- Flash 模式下特有的事件：netStatus。

> **说明：**
>
> 由于 Flash 的黑盒特性以及 H5 视频播放标准在各个平台终端的实现不一致性，事件的触发方式和结果会有差异。

在非自动播放的条件下，加载视频至待播放状态，移动端和 PC Flash 触发的事件区别。
**移动端：**

```
Object {type: "load", src: H5Video, ts: 0, detail: Object}
Object {type: "resize", src: H5Video, ts: 1150.5800000000002}
Object {type: "loadedmetadata", src: H5Video, ts: 1150.5850000000003}
Object {type: "volumechange", src: H5Video, ts: 1156.19}
Object {type: "seeking", src: H5Video, ts: 1168.665}
Object {type: "timeupdate", src: H5Video, ts: 1256.8400000000001}
Object {type: "seeked", src: H5Video, ts: 1256.85}
Object {type: "loadeddata", src: H5Video, ts: 1256.865}
Object {type: "timeupdate", src: H5Video, ts: 1256.9}
Object {type: "progress", src: H5Video, ts: 1408.7800000000002}
```

**PC Flash：**

```
Object {type: "load", src: FlashVideo, ts: 27, detail: Object}
Object {type: "loadedmetadata", src: FlashVideo, ts: 166, detail: Object}
Object {type: "volumechange", src: FlashVideo, ts: 184}
Object {type: "progress", src: FlashVideo, ts: 1741}
```

> **说明：**
> 以上是两种平台的差异，然而在移动端的各种设备和 App 之间同样存在差异。

事件监听函数返回的 msg 对象介绍：

| 名称 | 说明 |
| --- | --- |
| type | 事件类型。 |
| src | 事件源对象，即播放器实例，HTML5 或者 Flash。 |
| ts | 事件触发时的 UTC 时间戳。 |
| timeStamp | Event 实例的时间戳。 |

应用案例：通过事件监听，可以进行播放失败重连，单击访问 在线案例。

## 案例展示

结合了 TcPlayer 和即时通信 IM 的腾讯云 Web 直播互动组件：体验地址。

## 更新日志

TCPlayerLite 在不断更新及完善中，下面是 TCPlayerLite 发布的主版本介绍。

| 日期 | 版本 | 更新内容 |
| --- | --- | --- |
| 2020.06.25 | 2.4.1 | • 新增支持 v1 信令的 WebRTC 的流地址。<br>• 增加 webrtcConfig 参数。<br>• 增加 WebRTC 卡顿、卡顿结束、推流结束事件。 |
| 2021.06.03 | 2.4.0 | • 增加对快直播功能的支持。<br>• 修复其他已知问题。 |
| | | |

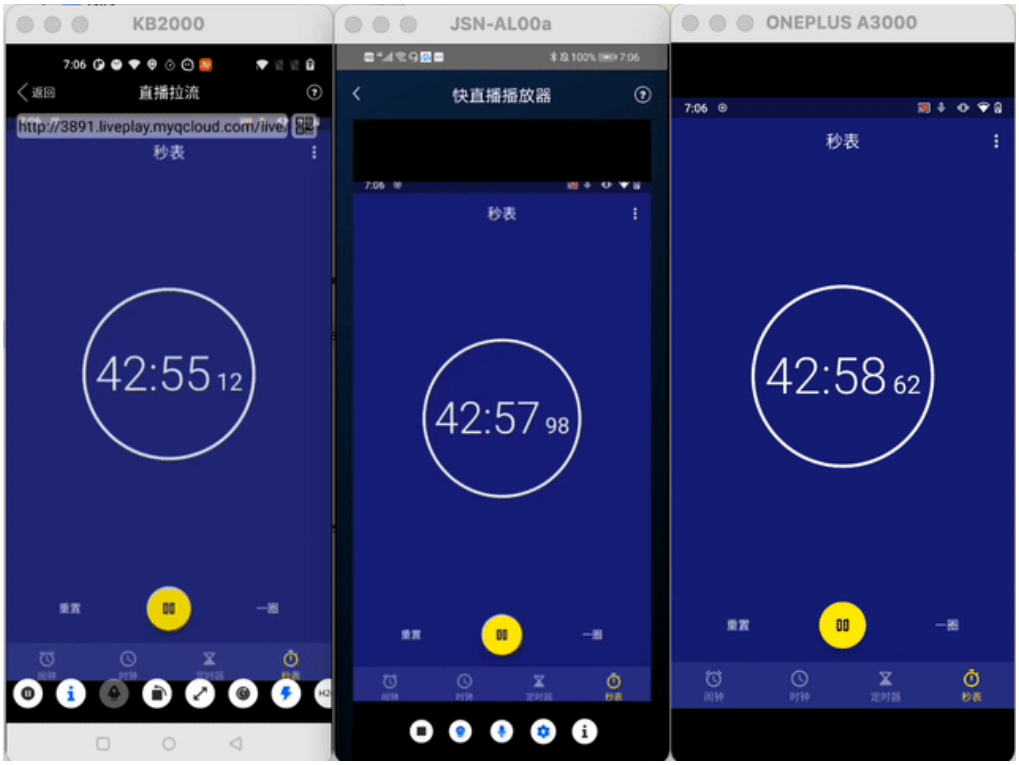
| | | |
|---|---|---|
| 2020.07.01 | 2.3.3 | • 修复 X5 环境下切换全屏时，事件派发异常的问题。<br>• 规避 hls 切换源时，相关事件触发时机很慢，导致封面显示异常的问题。 |
| 2019.08.20 | 2.3.2 | • 修改默认 hls 版本为0.12.4。<br>• 修复其他已知问题。 |
| 2019.04.26 | 2.3.1 | • 增加 fivConfig 参数。<br>• 默认加载 flv.1.5.js。<br>• 修复其他已知问题。 |
| 2019.04.19 | 2.3.0 | • 增加部分功能参数选项。<br>• 参数 coverpic 改为 poster。<br>• destroy 销毁 flv.js 实例。<br>• 修复其他已知问题。 |
| 2018.12.17 | 2.2.3 | • 优化播放逻辑。<br>• 解决 iOS 微信没有播放事件触发的情况下，出现 loading 动画的问题。<br>• 修复其他已知问题。 |
| 2018.05.03 | 2.2.2 | • 优化 loading 组件。<br>• 优化 Flash destroy 方法。<br>• 默认使用 H5 播放。<br>• 修复已知问题。 |
| 2017.12.20 | 2.2.1 | • 增加可配置清晰度文案功能。<br>• 设置默认清晰度。<br>• 支持切换清晰度方法。 |
| 2017.12.07 | 2.2.1 | • 增加 systemFullscreen 参数。<br>• 增加 flashUrl 参数。<br>• 修复音量 Max 后进行静音切换的 UI 问题。<br>• 修复 iOS 11 微信下需要单击两次才能播放的问题。<br>• 修复 safari 11 系统样式被遮挡的问题。<br>• 适配在 x5 内核会触发 seeking，但不会触发 seeked 的情况。<br>• 修复进度条拖拽到起始位置，设置 currentTime 失败的问题。<br>• 切换清晰度保持音量不变。<br>• 修复页面宽度为0，播放器宽度判断失败问题。<br>• destroy 方法增加完全销毁播放器节点。 |
| 2017.06.30 | 2.2.0 | • 增加控制播放环境判断的参数： Flash、h5_flv、x5_player。<br>• 调整播放器初始化逻辑，优化错误提示效果。<br>• 增加 flv.js 支持，在符合条件的情况下可以采用 flv.js 播放 FLV。<br>• 支持 x5-video-orientation 属性。<br>• 增加播放环境判断逻辑，可通过参数调整 H5 与 Flash 的优先级，以及是否启用 TBS 播放。<br>• 启用版本号发布方式，避免影响旧版本的使用者。<br>• 优化事件触发的时间戳，统一为标准时间。<br>• Bug 修复。 |
| 2017.03.04 | 2.1.0 | 至2017.06.30，经历数次的迭代开发，逐步趋于稳定，目前文档的功能描述中，如果没有特殊说明，皆基于此版本。 |
| 2016.12.28 | 2.0.0 | 首个版本。 |

# 快直播拉流
# iOS+Android

最近更新时间：2021-08-12 17:22:32

## 快直播概述

快直播（Live Event Broadcasting，LEB）是标准直播在超低延迟播放场景下的延伸，比传统直播协议延迟更低，为观众提供毫秒级的极致直播观看体验。 能够满足一些对延迟性能要求更高的特定场景需求，例如在线教育、体育赛事直播、在线答题等。




说明：

- 上图为快直播和标准的 CDN 直播的真实对比视频（使用 scrcpy 工具 配合录制），从左至右分别为：标准的 CDN 直播、**快直播**、推流端。
- 标准直播与快直播计费价格不同，更多计费详情请参见 标准直播计费 和 快直播计费。

### 方案优势

| 优势 | 说明 |
| --- | --- |
| 毫秒级超低延迟播放 | 采用 UDP 协议将传统直播中3秒 – 5秒延迟降低至1秒以内，同时兼顾秒开、卡顿率等核心指标，给用户带来极致的超低延迟直播体验。 |
| 功能完善，平滑兼容 | 兼容了标准直播包括推流、转码、录制、截图、鉴黄、播放等全功能，支持客户从现有的标准直播业务平滑迁移。 |
| 简单易用，安全可靠 | 采用标准协议，对接简单，在 Chrome 和 Safari 浏览器中无需任何插件即可进行播放。播放协议默认加密，更加安全可靠。 |

### 适用场景

| 场景 | 说明 |
| --- | --- |
| 体育赛事 | 快直播为体育赛事提供超低延迟的直播能力加持，使比赛赛事结果快速通过直播触达用户，让观众享受实时了解赛事动态的乐趣。 |
| 电商直播 | 电商直播中，商品拍卖、促销抢购等交易反馈对直播实时性要求很高，快直播的超低延迟能力，能让主播和观众能够及时得到交易反馈，提升边看边买的体验。 |
| 在线课堂 | 师生通过直播完成在线的课堂教学，得力于快直播的超低延迟能力，使课堂互动能力得到提升，让在线课堂也能像线下授课一样自然。 |
| 在线答题 | 传统的在线答题由于存在延迟，观众端有时需要进行补帧才能让观众主持两端同时显示。快直播的超低延迟能够完美解决这个问题，让双方实时看到答题画面，降低了实现难度，也让体验更加流畅。 |
| 秀场互动 | 快直播适用于秀场直播场景，极大优化了在观众送礼等对画面实时性要求高的直播互动场景中的观众互动体验。 |

## 体验快直播

视频云工具包是腾讯云开源的一套完整的音视频服务解决方案，包含实时音视频（TRTC）、移动直播（MLVB）、短视频（UGC）等多个 SDK 的能力展示，其中包含快直播相关体验 UI — **快直播播放** 。

> 说明：
> Demo 演示和体验步骤以 Android 为例，iOS Demo 界面略有不同。

### 源码及示例

| 源码下载 | 体验安装 | 推流演示（Android） | 播放演示（Android） |
| --- | --- | --- | --- |
| Android | |  |  |
| iOS | | | |



> 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

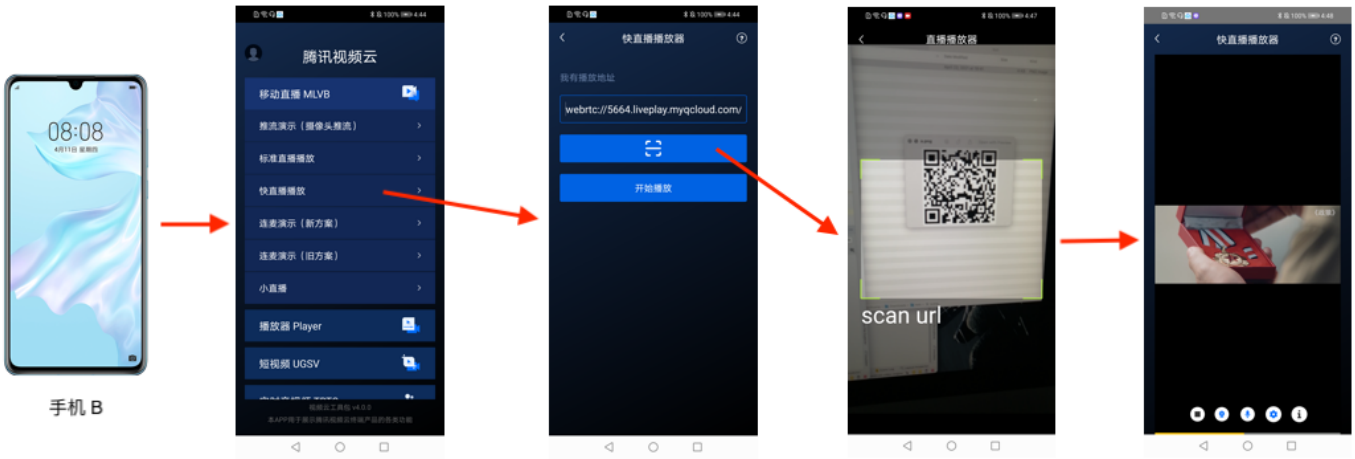
### 推流体验

快直播兼容了标准直播，因此可以使用普通推流端进行推流，然后使用快直播进行拉流。

1. 下载 视频云工具包，安装登录后，进入【推流演示（摄像头推流）】中。
2. 允许相关权限申请，单击【自动生成推流地址】，即开始推流了。
3. 成功推流之后，点击右上角的二维码图标，可以获取播放地址，其中【快直播】就是用于快直播的播放地址。
4. 成功开始推流后，可单击右下侧的菜单按钮，进行美颜、BGM、切换摄像头等设置操作。


### 播放体验

1. 下载 视频云工具包，安装登录后，进入【快直播播放】中。
2. 允许相关权限申请，单击二维码扫描按钮，扫描【推流体验】中得到的快直播播放地址。
3. 扫描完成后即开始播放，播放成功后，可单击右下侧的菜单按钮，进行静音、设置等操作。




## 接入工程

新版本的移动直播 SDK，可以使用 V2TXLivePlayer 来播放快直播的流，同时也提供了 `V2TXLivePusher` 来推流。快直播直播协议支持 WebRTC 标准协议，使用标准的扩展方式，其 URL 均以 `webrtc://` 字符开始。

### 步骤1：下载 SDK

可以在 SDK 下载 页面选择专业版或者企业版下载。

### 步骤2：获取播放 URL

在直播场景中，不论是推流还是拉流都离不开对应的 URL。请参见 快直播快速入门 获取快直播的播放 URL。
快直播 URL 均以 webrtc:// 字符开始，类似于这样：

```
webrtc://{Domain}/{AppName}/{StreamName}
```

在上述的 URL 中，存在一些关键字段，关于其中关键字段的含义信息，详见下表：

| 字段名称 | 字段含义 |
| --- | --- |
| webrtc:// | 快直播 URL 的前缀字段 |
| Domain | 快直播播放域名 |
| AppName | 应用名称，指的是直播流媒体文件存放路径，默认云直播会分配一个路径：live |
| StreamName | 流名称，指每路直播流唯一的标识符 |

> 说明：
> 如果需要推流，具体操作请参见 摄像头推流 或者 录屏推流。

### 步骤3：实现快直播播放

使用 V2TXLivePlayer 对象可以使用快直播进行拉流，具体做法如下（传入正确的 URL 是关键）：

**示例代码**

**Android**

```
// 创建⊠个 V2TXLivePlayer 对象；
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传⊠低延时协议播放地址，即可开始播放；
player.startPlay("webrtc://{Domain}/{AppName}/{StreamName}");
```

**iOS**

```
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
[player setObserver:self];
[player setRenderView:videoView];
[player startPlay:@"webrtc://{Domain}/{AppName}/{StreamName}"];
```

# Web（H5）播放器

最近更新时间：2021-06-30 10:33:20

## 功能介绍

腾讯云 Web 超级播放器 TCPlayerLite 是为了解决在手机浏览器和 PC 浏览器上播放音视频流的问题，它使您的视频内容可以不依赖用户安装 App，就能在朋友圈和微博等社交平台进行传播。本文档适合有一定 Javascript 语言基础的开发人员阅读。
以下视频将为您讲解腾讯云播放器 SDK 的 Web 播放器的功能特性以及对接攻略：

点击查看视频

## 基础知识

对接前需要了解如下基础知识：

- **直播和点播**
  直播视频源是实时的，一旦主播停播，直播地址就失去意义，而且由于是实时直播，所以播放器在播直播视频的时候是没有进度条的。
  点播视频源是某个服务器上的文件，只要文件没有被提供方删除，就可以随时播放， 而且由于整个视频都在服务器上，所以播放器在播点播视频的时候是有进度条的。

- **协议支持**
  TCPlayerLite 的视频播放能力本身不是网页代码实现的，而是靠浏览器支持，所以其兼容性不像我们想象的那么好，因此，**不是所有的手机浏览器都能有符合预期的表现**。一般用于网页直播的视频源地址是以 M3U8 结尾的地址，我们称其为 HLS (HTTP Live Streaming)，这是苹果推出的标准，目前各种手机浏览器产品对这种格式的兼容性也最好，但它有个问题：延迟比较大，一般是20s - 30s左右的延迟。

对于 PC 浏览器，因为其目前还没有抛弃 Flash 控件，而 Flash 控件支持的视频源格式较多，并且浏览器上的 Flash 控件都是 Adobe 自己研发，所以兼容性很好。

| 视频协议 | 用途 | URL 地址格式 | PC 浏览器 | 移动浏览器 |
| --- | --- | --- | --- | --- |
| WebRTC | 只适用直播 | webrtc://xxx.liveplay.myqcloud.com/live/xxx | 支持 | 支持 |
| HLS（M3U8） | 可用于直播 | http://xxx.liveplay.myqcloud.com/xxx.m3u8 | 支持 | 支持 |
| HLS（M3U8） | 可用于点播 | http://xxx.vod.myqcloud.com/xxx.m3u8 | 支持 | 支持 |
| FLV | 可用于直播 | http://xxx.liveplay.myqcloud.com/xxx.flv | 支持 | 不支持 |
| FLV | 可用于点播 | http://xxx.vod.myqcloud.com/xxx.flv | 支持 | 不支持 |
| RTMP | 只适用直播 | rtmp://xxx.liveplay.myqcloud.com/live/xxx | 支持 | 不支持 |
| MP4 | 只适用点播 | http://xxx.vod.myqcloud.com/xxx.mp4 | 支持 | 支持 |

⚠ 注意：

- 播放 RTMP 格式的视频必须启用 Flash，目前浏览器默认禁用 Flash，需用户手动开启。
- 在不支持 WebRTC 的浏览器环境，传入播放器的 WebRTC 地址会自动进行协议转换来更好的支持媒体播放，默认在移动端转换为 HLS，PC 端转换为 FLV。

**功能支持**

| 功能\浏览器 | Chrome | Firefox | Edge | QQ 浏览器 | Mac Safari | iOS Safari | iOS 微信 QQ | Android Chrome | Android 微信、QQ | 手机 QQ 浏览器 | IE 8,9,10,11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 设置封面 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 多清晰度支持 | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | × | ✓ |
| 定制错误提示语 | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | ✓ |
| 快直播 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |

## 对接攻略

### Step1. 页面准备工作

在需要播放视频的页面（PC 或 H5）中引入初始化脚本。

```
<script src="https://web.sdk.qcloud.com/player/tcplayerlite/release/v2.4.1/TcPlayer-2.4.1.js" charset="utf-8"></script>;
```

建议在使用播放器 SDK 的时候自行部署资源，点击下载播放器资源。

如果您部署的地址为 `aaa.xxx.ccc`，在合适的地方引入播放器脚本文件：

```
<script src="aaa.xxx.ccc/TcPlayer-2.4.1.js"></script>
```

> ⚠ 注意：
> 直接用本地网页无法调试，Web 播放器无法处理该情况下的跨域问题。

### Step2. 在 HTML 中放置容器

在需要展示播放器的页面位置加入播放器容器，即放一个 div 并命名，例如 `id_test_video`，视频画面都会在容器里渲染。对于容器的大小控制，您可以使用 div 的属性进行控制，示例代码如下：

```
<div id="id_test_video" style="width:100%; height:auto;"></div>
```

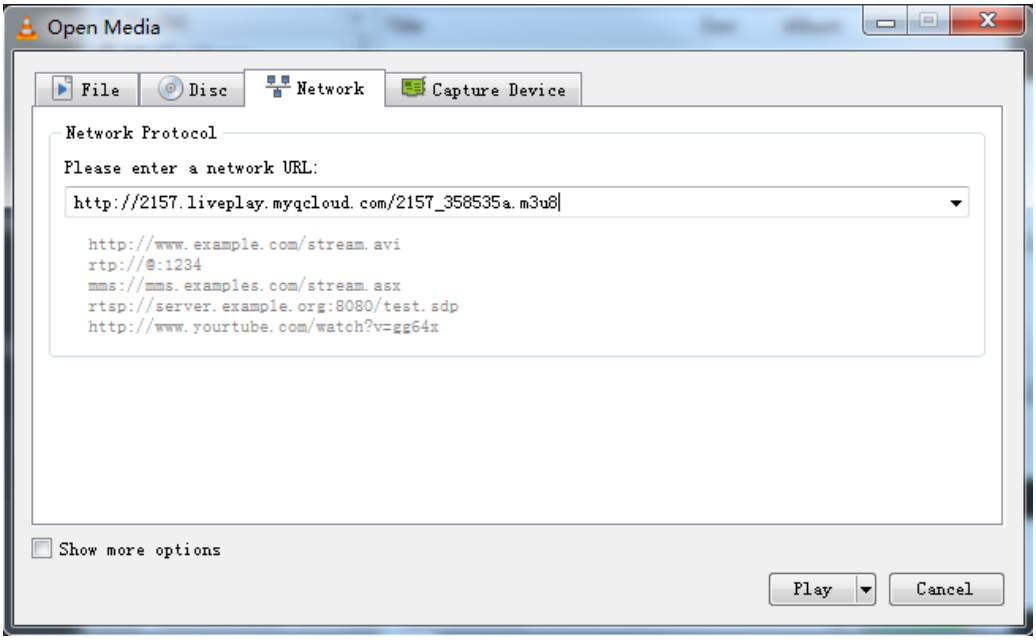
### Step3. 对接视频播放

编写 Javascript 代码，作用是去指定的 URL 地址拉取音视频流，并将视频画面呈现到添加的容器内。

#### 3.1 简单播放

如下是一个 直播格式的 URL 地址，使用 HLS（M3U8）协议，如果主播在直播中，则用 VLC 等播放器是可以直接打开该 URL 进行观看的：

```
http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 // m3u8 播放地址
```




如果要在手机浏览器上播放该 URL 的视频，则 Javascript 代码如下：

```
var player = new TcPlayer('id_test_video', {
"m3u8": "http://2157.liveplay.myqcloud.com/2157_358535a.m3u8", //请替换成实际可用的播放地址
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"width" : '480',//视频的显示宽度，请尽量使用视频分辨率宽度
"height" : '320'//视频的显示高度，请尽量使用视频分辨率高度
});
```

这段代码可以支持在 PC 及手机浏览器上播放 HLS（M3U8）协议的直播视频，虽然 HLS（M3U8）协议的视频兼容性不错，但部分 Android 手机依然不支持，其延迟较高，大约20秒以上的延迟。

### 3.2 实现更低延迟

PC 浏览器支持 Flash，其 Javascript 代码如下：

```
var player = new TcPlayer('id_test_video', {
"m3u8": "http://2157.liveplay.myqcloud.com/2157_358535a.m3u8",
"flv": "http://2157.liveplay.myqcloud.com/live/2157_358535a.flv", //增加了一个 flv 的播放地址，用于PC平台的播放 请替换成实际可用的播放地址
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"width" : '480',//视频的显示宽度，请尽量使用视频分辨率宽度
"height" : '320'//视频的显示高度，请尽量使用视频分辨率高度
});
```

这段代码中增加了 FLV 的播放地址，Web 播放器如果发现当前的浏览器是 PC 浏览器，会主动选择 FLV 链路，从而实现更低的延迟。如果对延迟有更高的要求，可以使用 WebRTC 拉流地址，基于 WebRTC 的播放系统可以实现超低延迟（500ms），前提条件是拉流地址都是可以

出流的，如果您使用腾讯云的视频直播服务，则无需考虑，因为腾讯云的直播频道默认支持 WebRTC、FLV、RTMP 和 HLS（M3U8）播放协议。

**无法播放怎么办?**

如果您发现视频无法播放，可能存在如下原因：

- **原因一：视频源有问题**
  如果是直播 URL，则需要检查主播是否已经停止推流，可以用浮窗提示观众：“主播已经离开”。请参见 直播推流。
  如果是点播 URL，则需要检查要播放的文件是否还存在于服务器上（如播放地址是否已经从点播系统移除）。

- **原因二：本地网页调试**
  目前 TCPlayerLite 不支持本地网页调试（即通过 `file://` 协议打开视频播放的网页），因为浏览器有跨域安全限制，所以在 Windows 系统上放置一个 test.html 文件来进行测试是无法播放的，需要将其上传到服务器上进行测试。而前端工程师可以通过反向代理的方式，对线上页面进行本地代理以实现本地调试，这是主流的本地调试方法。

- **原因三：手机兼容问题**
  普通的手机浏览器只支持 HLS（M3U8）协议的播放，不支持 FLV 和 RTMP 协议，最新版本的 QQ 浏览器支持 FLV 协议的播放。

- **原因四：跨域安全问题**
  PC 浏览器的视频播放基于 Flash 控件实现，但 **Flash 控件会做跨域访问检查**，如果播放视频所存放的服务器没有部署跨域策略，则会出现问题。
  解决方法：在视频存储服务器根域名下添加跨域配置文件 `crossdomain.xml`，并配置 Flash swf 所在域名，以允许 Flash 和 JavaScript 跨域播放视频。
  播放器的 Flash swf 文件默认存放在 `imgcache.qq.com` 域名下，如需部署到自己的服务器上，可自行下载并部署：swf 文件地址。
  如果是在域名限制区域，需要的播放器的 Flash swf 文件默认存放在 `cloudcache.tencent-cloud.com` 域名下，并且在播放器初始化的时候传递 flashUrl。

```
<cross-domain-policy>
<allow-access-from domain="*.*.com" secure="false"/>
</cross-domain-policy>
```

## Step4. 给播放器设置封面

设置封面涉及到 poster 属性，下面将详细介绍 poster 属性的使用方法。

> ⚠ 注意：
> 封面功能在部分移动端播放环境下可能失效，通常是由于移动端 webview 劫持视频播放造成的，需要 webview 支持 video 叠加元素或者放开劫持视频播放。相关详细说明请参见 常见问题。

### 4.1 简单设置封面

poster 支持传入图片地址作为播放器的封面，在播放器区域内居中，并且以图片的实际分辨率进行显示。

```
"poster" : "http://www.test.com/myimage.jpg"
```

### 4.2 设置封面样式

poster 支持传入一个对象，在对象中可以对封面的展现样式（style）和图片地址（src）进行设置。

style 支持的样式如下：

- default：居中并且以图片的实际分辨率进行显示。

- stretch：拉伸铺满播放器区域，图片可能会变形。
- cover：优先横向等比拉伸铺满播放器区域，图片某些部分可能无法显示在区域内。

```
"poster" : {"style":"stretch", "src":"http://www.test.com/myimage.jpg"}
```

#### 4.3 实现用例

使用 cover 方式显示封面。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：
视频封面

> ⚠ 注意：
>
> - 在某些移动端设置封面会无效，具体说明请参见 常见问题。
> - 以上示例链接仅用于文档演示，请勿用于生产环境。

### Step5. 多清晰度支持

#### 5.1 原理介绍

同腾讯视频，Web 播放器支持多清晰度，如下图所示：


**播放器本身是没有能力去改变视频清晰度的**，视频源只有一种清晰度，称之为原画，而原画视频的编码格式和封装格式多种，Web 端无法支持播放所有的视频格式，如点播支持以 H.264 为视频编码，MP4 和 FLV 为封装格式的视频。

**多清晰度的实现依赖于视频云：**

- 对于直播，来自主播端的原始视频会在腾讯云进行实时转码，分出多路转码后的视频，每一路视频都有其对应的地址，例如“高清-HD”和“标清-SD”，地址格式如下：

```
http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 // 原画
http://2157.liveplay.myqcloud.com/2157_358535a_900.m3u8 // 高清
http://2157.liveplay.myqcloud.com/2157_358535a_550.m3u8 // 标清
```

- 对于点播，一个视频文件上传到腾讯云后，您可以对该视频文件进行转码，产生其它几种清晰度的视频，例如“高清-HD”和“标清-SD”，地址格式如下：

```
http://200002949.vod.myqcloud.com/200002949_b6ffc.f240.m3u8 // 原画，用转码后的超清替换
http://200002949.vod.myqcloud.com/200002949_b6ffc.f230.av.m3u8 // 高清
http://200002949.vod.myqcloud.com/200002949_b6ffc.f220.av.m3u8 // 标清
```

> ⚠ 注意：
> 上传后的原始视频是未经过腾讯云转码的，不能直接用于播放。

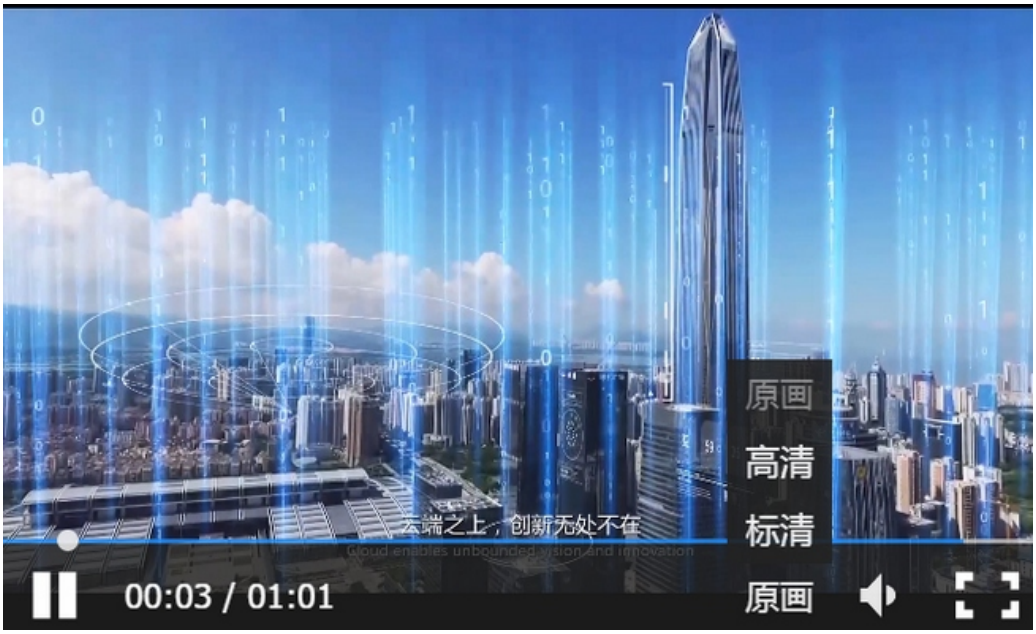
### 5.2 代码实现

多清晰度支持的代码实现如下所示：

```
var player = new TcPlayer('id_test_video', {
"m3u8" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f240.m3u8",//请替换成实际可用的播放地址
"m3u8_hd" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f230.av.m3u8",
"m3u8_sd" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f220.av.m3u8",
"autoplay" : true, //iOS 下 safari 浏览器，以及大部分移动端浏览器是不开放视频自动播放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
});
```

### 5.3 实现用例

使用多种分辨率的设置及切换功能。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：分辨率切换
正常情况将看到如下效果：




> ⚠ 注意：
> - PC 端现已支持多种清晰度播放及切换的功能，移动端尚未支持。
> - 以上示例链接仅用于文档演示，请勿用于生产环境

## Step6. 定制错误提示语

Web 播放器支持提示语定制。

### 6.1 代码实现

如下是让播放器支持自定义提示语的核心代码，主要在 wording 属性上设置提示语。

```
var player = new TcPlayer('id_test_video', {
"m3u8" : "http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.m3u8",//请替换成实际可用的播放地址
```

```
"autoplay" : true, //iOS 下 safari 浏览器是不开放这个能力的
"poster" : "http://www.test.com/myimage.jpg",
"wording": {
2032: "请求视频失败，请检查网络",
2048: "请求m3u8文件失败，可能是网络错误或者跨域问题"
}
});
```

### 6.2 实现用例

视频播放失败，同时使用自定义提示文案的功能。线上示例如下，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：

```
https://web.sdk.qcloud.com/player/tcplayerlite/tcplayer-error.html
```

> ⚠ 注意：
> 以上示例链接仅用于文档演示，请勿用于生产环境。

### 6.3 错误码表

| Code | 提示语 | 说明 |
| --- | --- | --- |
| 1 | 网络错误，请检查网络配置或者播放链接是否正确。 | H5 提示的错误。 |
| 2 | 网络错误，请检查网络配置或者播放链接是否正确。 | 视频格式 Web 播放器无法解码。<br>H5 提示的错误。 |
| 3 | 视频解码错误。 | H5 提示的错误。 |
| 4 | 当前系统环境不支持播放该视频格式。 | H5 提示的错误。 |
| 5 | 当前系统环境不支持播放该视频格式。 | 播放器判断当前浏览器环境不支持播放传入的视频，可能是当前浏览器不支持 MSE 或者 Flash 插件未启用。 |
| 10 | 请勿在 file 协议下使用播放器，可能会导致视频无法播放。 | - |
| 11 | 使用参数有误，请检查播放器调用代码。 | - |
| 12 | 请填写视频播放地址。 | - |
| 13 | 直播已结束，请稍后再来。 | RTMP 正常播放过程中触发事件（NetConnection.Connect.Closed）。<br>Flash 提示的错误。 |
| 1001 | 网络错误，请检查网络配置或者播放链接是否正确。 | 网络已断开（NetConnection.Connect.Closed）。<br>Flash 提示的错误。 |
| 1002 | 获取视频失败，请检查播放链接是否有效。 | 拉取播放文件失败（NetStream.Play.StreamNotFound），可能是服务器错误或者视频文件不存在。<br>Flash 提示的错误。 |
| 2001 | 调用 WebRTC 接口失败 | 播放 WebRTC 时设置 sdp 失败提示的错误 |
| 2002 | 调用拉流接口失败 | 播放 WebRTC 时调用拉流接口失败提示的错误 |

| Code | 提示语 | 说明 |
| --- | --- | --- |
| 2003 | 连接服务器失败，并且连接重试次数已超过设定值 | 播放 WebRTC 时提示的错误，可用于确定是否为停止推流状态 |
| 2032 | 获取视频失败，请检查播放链接是否有效。 | Flash 提示的错误。 |
| 2048 | 无法加载视频文件，跨域访问被拒绝。 | 请求 M3U8 文件失败，可能是网络错误或者跨域问题。<br>Flash 提示的错误。 |

**说明：**

- Code 1 – 4 对应的是 H5 原生事件。
- 由于 Flash 的黑盒特性以及 H5 视频播放标准的不确定性，错误提示语会不定期更新。

## 源码参考

如下是一个线上示例代码，在 PC 浏览器中右键单击【查看页面源码】即可查看页面的代码实现：播放示例

**注意：**

以上示例链接仅用于文档演示，请勿用于生产环境。

## 参数列表

播放器支持的所有参数，如下所示：

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| webrtc | String | 无 | 原画 WebRTC 播放 URL。<br>示例：webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1 |
| webrtc_hd | String | 无 | 高清 WebRTC 播放 URL。<br>示例：<br>webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1_hd |
| webrtc_sd | String | 无 | 标清 WebRTC 播放 URL。<br>示例：<br>webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1_sd |
| m3u8 | String | 无 | 原画 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535a.m3u8 |
| m3u8_hd | String | 无 | 高清 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535ahd.m3u8 |
| m3u8_sd | String | 无 | 标清 M3U8 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535asd.m3u8 |
| flv | String | 无 | 原画 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535a.flv |
| flv_hd | String | 无 | 高清 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535ahd.flv |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| flv_sd | String | 无 | 标清 FLV 播放 URL。<br>示例：http://2157.liveplay.myqcloud.com/2157_358535asd.flv |
| mp4 | String | 无 | 原画 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f0.mp4 |
| mp4_hd | String | 无 | 高清 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f40.mp4 |
| mp4_sd | String | 无 | 标清 MP4 播放 URL。<br>示例：<br>http://200002949.vod.myqcloud.com/200002949_b6ffc.f20.mp4 |
| rtmp | String | 无 | 原画 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88 |
| rtmp_hd | String | 无 | 高清 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88hd |
| rtmp_sd | String | 无 | 标清 RTMP 播放 URL。<br>示例：rtmp://2157.liveplay.myqcloud.com/live/2157_280d88sd |
| width | Number | 无 | **必选**，设置播放器宽度，单位为像素。<br>示例：640 |
| height | Number | 无 | **必选**，设置播放器高度，单位为像素。<br>示例：480 |
| volume | Number | 0.5 | 设置初始音量，范围：0到1 [v2.2.0+]。<br>示例：0.6 |
| live | Boolean | false | **必选**，设置视频是否为直播类型，将决定是否渲染时间轴等控件，以及区分点直播的处理逻辑。<br>示例：true |
| autoplay | Boolean | false | 是否自动播放。<br>（**备注：该选项只对大部分 PC 平台生效**）<br>示例：true |
| poster | String / Object | 无 | 预览封面，可以传入一个图片地址或者一个包含图片地址 src 和显示样式 style 的对象。<br>style 可选属性：<br>• default 居中1：1显示。<br>• stretch 拉伸铺满播放器区域，图片可能会变形。<br>• cover 优先横向等比拉伸铺满播放器区域，图片某些部分可能无法显示在区域内。<br>示例： "http://www.test.com/myimage.jpg" 或者<br>{"style": "cover", "src": http://www.test.com/myimage.jpg}<br>[v2.3.0+] |
| controls | String | "default" | default 显示默认控件，none 不显示控件，system 移动端显示系统控件。<br>（备注：如果需要在移动端使用系统全屏，就需要设置为 system。默认全屏方案是使用 Fullscreen API + 伪全屏的方式，在线示例 ）<br>示例："system" |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| systemFullscreen | Boolean | false | 开启后，在不支持 Fullscreen API 的浏览器环境下，尝试使用浏览器提供的 webkitEnterFullScreen 方法进行全屏，如果支持，将进入系统全屏，控件为系统控件。<br>示例：true |
| flash | Boolean | true | 是否优先使用 Flash 播放视频。<br>（**备注：该选项只对 PC 平台生效**[v2.2.0+]）<br>示例：true |
| flashUrl | String | 无 | 可以设置 flash swf url。<br>（**备注：该选项只对 PC 平台生效** [v2.2.1+]） |
| h5_flv | Boolean | false | 是否启用 flv.js 的播放 flv。启用时播放器将在支持 MSE 的浏览器下，采用 flv.js 播放 flv，然而并不是所有支持 MSE 的浏览器都可以使用 flv.js，所以播放器不会默认开启这个属性，[v2.2.0+]。<br>示例：true |
| x5_player | Boolean | false | 是否启用 TBS 的播放 flv 或 hls 。启用时播放器将在 TBS 模式下(例如 Android 的微信、QQ 浏览器），将 flv 或 hls 播放地址直接赋给 <video> 播放。TBS 视频能力 [v2.2.0+]。<br>示例： true |
| x5_type | String | 无 | 通过 video 属性 “x5-video-player-type” 声明启用同层 H5 播放器，支持的值：h5-page (该属性为 TBS 内核实验性属性，非 TBS 内核不支持)，TBS H5 同层播放器接入规范。<br>示例："h5-page" |
| x5_fullscreen | String | 无 | 通过 video 属性 “x5-video-player-fullscreen” 声明视频播放时是否进入到 TBS 的全屏模式，支持的值：true (该属性为 TBS 内核实验性属性，非 TBS 内核不支持) 。<br>示例："true" |
| x5_orientation | Number | 无 | 通过 video 属性 “x5-video-orientation” 声明 TBS 播放器支持的方向，可选值：0（landscape 横屏），1：（portraint竖屏），2：（landscape |
| wording | Object | 无 | 自定义文案。<br>示例：{ 2032: '请求视频失败，请检查网络'} |
| clarity | String | 'od' | 默认播放清晰度 [v2.2.1+]。<br>示例：clarity: 'od' |
| clarityLabel | Object | {od: '超清', hd: '高清', sd: '标清'} | 自定义清晰度文案 [v2.2.1+]。<br>示例：clarityLabel: {od: '蓝光', hd: '高清', sd: '标清'}。 |
| listener | Function | 无 | 事件监听回调函数，回调函数将传入一个 JSON 格式的对象。<br>示例：function(msg){<br>//进行事件处理<br>} |
| pausePosterEnabled | Boolean | true | 暂停时显示封面 [v2.3.0+]。 |
| preload | String | 'auto' | 配置 video 标签的 preload 属性，只有部分浏览器生效[v2.3.0+]。 |
| hlsConfig | Object | 无 | hls.js 初始化配置项 [v2.3.0+]。 |

| 参数 | 类型 | 默认值 | 参数说明 |
| --- | --- | --- | --- |
| flvConfig | Object | 无 | flv.js 初始化配置项 [v2.3.1+]。 |
| webrtcConfig | Object | 无 | webrtc 初始化配置项 [v2.4.1+]。<br>支持通过 streamType 指定拉流类型，默认拉取音视频，可选单独拉取视频或单独拉取音频，streamType 可选属性：<br>• auto：拉取视频流和音频流<br>• video：仅拉取视频流<br>• audio：仅拉取音频流<br>示例：`webrtcConfig: { streamType: 'video' }` |

⚠ 注意：

- WebRTC 快直播播放地址支持两种格式，除 `webrtc://domain/AppName/StreamName?txSecret=XXX&txTime=XXX` 以外，还支持 `http://domain/AppName/StreamName.sdp?txSecret=XXX&txTime=XXX` 格式的播放地址，但是需要配置播放域名 CNAME 到 `overseas-webrtc.liveplay.myqcloud.com` 。
- 由于 Web 浏览器目前不支持标准 WebRTC 协议携带 B 帧播放，如果原始流存在 B 帧，则后台会自动进行转码去掉 B 帧，这样会引入额外的转码延迟，并产生转码费用。建议尽量不推包含 B 帧的流，移动直播 SDK，IOS 不支持引入 B 帧，安卓如果不开启 B 帧设置，默认是没有带 B 帧。如果使用 OBS 推流，可以通过设置，关闭 B 帧。

## 实例方法列表

播放器实例支持的方法，如下所示：

| 方法 | 参数 | 返回值 | 说明 | 示例 |
| --- | --- | --- | --- | --- |
| play() | 无 | 无 | 开始播放视频。 | player.play() |
| pause() | 无 | 无 | 暂停播放视频。 | player.pause() |
| togglePlay() | 无 | 无 | 切换视频播放状态 。 | player.togglePlay() |
| mute(muted) | {Boolean} [可选] | true,false {Boolean} | 切换静音状态，不传参则返回当前是否静音。 | player.mute(true) |
| volume(val) | {int} 范围：0到1 [可选] | 范围：0到1 | 设置音量，不传参则返回当前音量 。 | player.volume(0.3) |
| playing() | 无 | true,false {Boolean} | 返回是否在播放中 。 | player.playing() |
| duration() | 无 | {int} | 获取视频时长 。<br>（备注：只适用于点播，需要在触发 loadedmetadata 事件后才可获取视频时长） | player.duration() |
| currentTime(time) | {int} [可选] | {int} | 设置视频播放时间点，不传参则返回当前播放时间点 。<br>（备注：只适用于点播 ） | player.currentTime() |

| 方法 | 参数 | 返回值 | 说明 | 示例 |
| --- | --- | --- | --- | --- |
| fullscreen(enter) | {Boolean} [可选] | true,false {Boolean} | 调用全屏接口(Fullscreen API)，不支持全屏接口时使用伪全屏模式，不传参则返回值当前是否是全屏。**（备注：移动端系统全屏没有提供 API，也无法获取系统全屏状态 ）** | player.fullscreen(true) |
| buffered() | 无 | 0到1 | 获取视频缓冲数据百分比。**（备注：只适用于点播）** | player.buffered() |
| destroy() | 无 | 无 | 销毁播放器实例[v2.2.1+]。 | player.destroy() |
| switchClarity() | {String} [必选] | 无 | 切换清晰度，传值 "od"、"hd"、"sd" [v2.2.1+]。 | player.switchClarity('od') |
| load(url) | {String} [必选] | 无 | 通过视频地址加载视频。**（备注：该方法只能加载对应播放模式下支持的视频格式，Flash 模式支持切换 RTMP、FLV、HLS 和 MP4 ，H5 模式支持 MP4、HLS 和 FLV（HLS、FLV 取决于浏览器是否支持）** [v2.2.2+]） | player.load(http://xxx.mp4) |

> **注意：**
> 以上方法必须是 TcPlayer 的实例化对象，且需要初始化完毕才可以调用（即 load 事件触发后）。

## 进阶攻略

下面介绍播放器 SDK 的进阶使用方法。

### ES Module

TCPlayerLite 提供了 ES Module 版本，module name 为 TcPlayer，下载地址：

```
https://web.sdk.qcloud.com/player/tcplayerlite/release/v2.4.1/TcPlayer-module-2.4.1.js
```

#### 开启优先 H5 播放模式

TCPlayerLite 采用 H5 <video> 和 Flash 相结合的方式来进行视频播放，根据不同的播放环境，播放器会选择默认最合适的播放方案。

虽然浏览器厂商已经开始逐步放弃对 Flash 插件的支持，但是在国内仍有大量的浏览器不支持 MSE，在播放 FLV 和 HLS（M3U8）时无法切换到 H5 <video> 模式，而播放 RTMP 必须使用 Flash。
因此，TCPlayerLite 默认优先启用 Flash 播放模式，如果在检测到 Flash 插件不可用的情况下，将采用 H5 <video> 进行播放。

> **说明：**
> 默认 Flash 模式的原因是 Flash 支持的视频格式最广，而 H5 <video> 默认只支持 MP4（H.264），在特定条件下才支持 HLS（M3U8）和 FLV。

**从2.2.0版本开始，提供了可以设置播放模式优先级的属性**，如果想优先采用 H5 <video> 播放模式，则需要把 Flash 属性设置为 False；如果 H5 <video> 不可用，则采用 Flash 播放；如果没有检测到 Flash 插件，则会提示“当前系统环境不支持播放该视频格式”。

#### 监听事件

TCPlayerLite 是采用 H5 <video> 和 Flash 相结合的方式来进行视频播放，由于两种方式播放视频时触发的事件不尽相同，所以我们以 H5 <video> 的规范，对 Flash 的播放事件做了一定程度的转换，以实现播放事件命名的统一， TcPlayer 对这两种播放方式所触发的原生事件进行了捕获和透传。

- H5 事件参考列表
- Flash 事件参考列表
- 统一后的事件列表

```
error
timeupdate
load
loadedmetadata
loadeddata
progress
fullscreen
play
playing
pause
ended
seeking
seeked
resize
volumechange
webrtcstatupdate
webrtcwaitstart
webrtcwaitend
webrtcstop
```

> **注意：**
>
> - 如果通过系统控制栏进行全屏，将无法监听到 fullscreen 事件。
> - Web 播放器的事件，依赖浏览器内置的解码器和 Flash 插件触发，Web 播放器仅透传事件。
> - Web 播放器监听不到直播停止推流的事件，需要通过额外的接口来确认推流状态，请参见 查询流状态。

- Flash 模式下特有的事件：netStatus。

> **说明：**
>
> 由于 Flash 的黑盒特性以及 H5 视频播放标准在各个平台终端的实现不一致性，事件的触发方式和结果会有差异。

在非自动播放的条件下，加载视频至待播放状态，移动端和 PC Flash 触发的事件区别。
**移动端：**

```
Object {type: "load", src: H5Video, ts: 0, detail: Object}
Object {type: "resize", src: H5Video, ts: 1150.5800000000002}
Object {type: "loadedmetadata", src: H5Video, ts: 1150.5850000000003}
Object {type: "volumechange", src: H5Video, ts: 1156.19}
Object {type: "seeking", src: H5Video, ts: 1168.665}
Object {type: "timeupdate", src: H5Video, ts: 1256.8400000000001}
Object {type: "seeked", src: H5Video, ts: 1256.85}
Object {type: "loadeddata", src: H5Video, ts: 1256.865}
Object {type: "timeupdate", src: H5Video, ts: 1256.9}
Object {type: "progress", src: H5Video, ts: 1408.7800000000002}
```

**PC Flash：**

```
Object {type: "load", src: FlashVideo, ts: 27, detail: Object}
Object {type: "loadedmetadata", src: FlashVideo, ts: 166, detail: Object}
Object {type: "volumechange", src: FlashVideo, ts: 184}
Object {type: "progress", src: FlashVideo, ts: 1741}
```

> **说明：**
> 以上是两种平台的差异，然而在移动端的各种设备和 App 之间同样存在差异。

事件监听函数返回的 msg 对象介绍：

| 名称 | 说明 |
| --- | --- |
| type | 事件类型。 |
| src | 事件源对象，即播放器实例，HTML5 或者 Flash。 |
| ts | 事件触发时的 UTC 时间戳。 |
| timeStamp | Event 实例的时间戳。 |

应用案例：通过事件监听，可以进行播放失败重连，单击访问 在线案例。

## 案例展示

结合了 TcPlayer 和即时通信 IM 的腾讯云 Web 直播互动组件：体验地址。

## 更新日志

TCPlayerLite 在不断更新及完善中，下面是 TCPlayerLite 发布的主版本介绍。

| 日期 | 版本 | 更新内容 |
| --- | --- | --- |
| 2021.06.25 | 2.4.1 | • 新增支持 v1 信令的 WebRTC 的流地址。<br>• 增加 webrtcConfig 参数。<br>• 增加 WebRTC 卡顿、卡顿结束、推流结束事件。 |
| 2021.06.03 | 2.4.0 | • 增加对快直播功能的支持。<br>• 修复其他已知问题。 |
| | | |

| | | |
|---|---|---|
| 2020.07.01 | 2.3.3 | • 修复 X5 环境下切换全屏时，事件派发异常的问题。<br>• 规避 hls 切换源时，相关事件触发时机很慢，导致封面显示异常的问题。 |
| 2019.08.20 | 2.3.2 | • 修改默认 hls 版本为0.12.4。<br>• 修复其他已知问题。 |
| 2019.04.26 | 2.3.1 | • 增加 fivConfig 参数。<br>• 默认加载 flv.1.5.js。<br>• 修复其他已知问题。 |
| 2019.04.19 | 2.3.0 | • 增加部分功能参数选项。<br>• 参数 coverpic 改为 poster。<br>• destroy 销毁 flv.js 实例。<br>• 修复其他已知问题。 |
| 2018.12.17 | 2.2.3 | • 优化播放逻辑。<br>• 解决 iOS 微信没有播放事件触发的情况下，出现 loading 动画的问题。<br>• 修复其他已知问题。 |
| 2018.05.03 | 2.2.2 | • 优化 loading 组件。<br>• 优化 Flash destroy 方法。<br>• 默认使用 H5 播放。<br>• 修复已知问题。 |
| 2017.12.20 | 2.2.1 | • 增加可配置清晰度文案功能。<br>• 设置默认清晰度。<br>• 支持切换清晰度方法。 |
| 2017.12.07 | 2.2.1 | • 增加 systemFullscreen 参数。<br>• 增加 flashUrl 参数。<br>• 修复音量 Max 后进行静音切换的 UI 问题。<br>• 修复 iOS 11 微信下需要单击两次才能播放的问题。<br>• 修复 safari 11 系统样式被遮挡的问题。<br>• 适配在 x5 内核会触发 seeking，但不会触发 seeked 的情况。<br>• 修复进度条拖拽到起始位置，设置 currentTime 失败的问题。<br>• 切换清晰度保持音量不变。<br>• 修复页面宽度为0，播放器宽度判断失败问题。<br>• destroy 方法增加完全销毁播放器节点。 |
| 2017.06.30 | 2.2.0 | • 增加控制播放环境判断的参数： Flash、h5_flv、x5_player。<br>• 调整播放器初始化逻辑，优化错误提示效果。<br>• 增加 flv.js 支持，在符合条件的情况下可以采用 flv.js 播放 FLV。<br>• 支持 x5-video-orientation 属性。<br>• 增加播放环境判断逻辑，可通过参数调整 H5 与 Flash 的优先级，以及是否启用 TBS 播放。<br>• 启用版本号发布方式，避免影响旧版本的使用者。<br>• 优化事件触发的时间戳，统一为标准时间。<br>• Bug 修复。 |
| 2017.03.04 | 2.1.0 | 至2017.06.30，经历数次的迭代开发，逐步趋于稳定，目前文档的功能描述中，如果没有特殊说明，皆基于此版本。 |
| 2016.12.28 | 2.0.0 | 首个版本。 |

# 连麦互动
# iOS+Android

最近更新时间：2021-08-03 17:22:48

## 基于 RTC 协议的连麦方案

目前，在 **连麦互动 – RTMP方案** 中，腾讯云移动直播 SDK 提供连麦互动组件 `MLVBLiveRoom` 用来帮助开发者快速实现连麦需求，为了更好的满足开发者针对连麦功能的需求，腾讯云新增了基于 RTC 协议的连麦方案，同时提供了更加简单灵活的 V2 接口。

移动直播 V2 接口同时支持通过 RTMP 协议及 RTC 协议进行推流/连麦，开发者可根据自身需求选择适合的方案，对比如下：

| 对比项 | RTMP 方案 | RTC 方案 |
| --- | --- | --- |
| 协议 | RTMP 基于 TCP 协议 | RTC 基于 UDP 协议（更适合流媒体传输） |
| QoS | 弱网抗性能力弱 | 50%丢包率可正常视频观看，70%丢包率可正常语音连麦 |
| 支持区域 | 仅支持中国内地（大陆）地区 | 全球覆盖 |
| 使用产品 | 需开通移动直播、云直播服务 | 需开通移动直播、云直播、实时音视频服务 |
| 价格 | 0.016元/分钟 | 阶梯价格，详情请参见 RTC 连麦方案怎么计算费用 |

## RTC 连麦方案如何接入

移动直播 SDK 提供了新的 V2 接口： `V2TXLivePusher` （推流）、 `V2TXLivePlayer` （拉流），用来帮助客户实现更加灵活、更低延时、更多人数的直播互动场景。开播端可以利用 V2 提供的 RTC 推流能力，默认情况下，观众端观看则可使用 CDN 方式进行拉流。 CDN 观看费用较低。如果观众端有连麦需求，连麦观众上麦后，可以从 CDN 切换到 RTC 进行观看，这样延时更低，互动效果更好。以下是相关示例代码和具体的步骤说明，方便您快速接入：

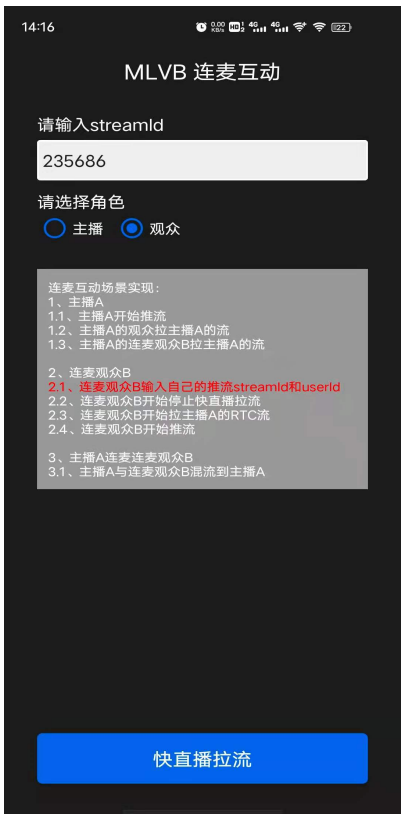
### 示例工程

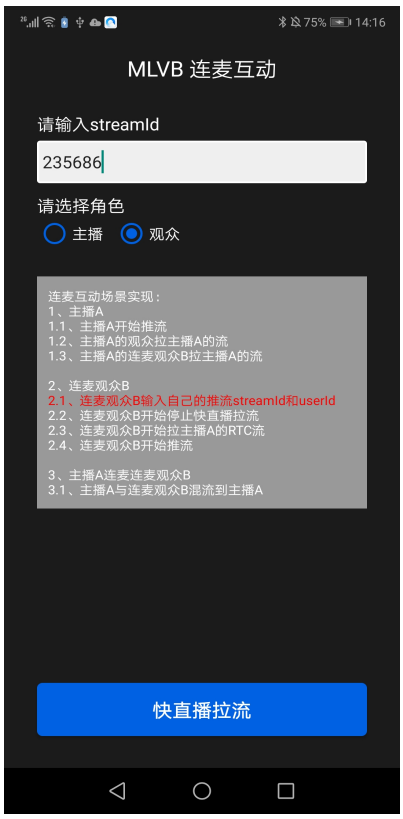
| 平台 | 源码地址 | 目标文件夹 |
| --- | --- | --- |
| Android | Github | LiveLink |
| iOS | Github | LiveLink |

### 演示图示

**直播前**

| 主播端（手机 A） | 连麦观众（手机 B） | 普通观众（手机 C） |
| --- | --- | --- |
| | | |


**连麦中**

| 主播端（手机 A） | 连麦观众（手机 B） | 普通观众（手机 C） |
| --- | --- | --- |
|  |  |  |







**步骤1：服务开通**

RTC 连麦互动直播需要在开始接入前，先开通腾讯云 **实时音视频** 服务，具体步骤如下：

1. 您需要 **注册腾讯云** 账号，并完成 **实名认证**。
2. 登录实时音视频控制台，选择【 **应用管理** 】。
3. 单击【 创建应用 】，输入应用名称，例如 V2Demo ，单击【 确定 】。


4. 创建成功后，单击右侧【 应用信息 】，查看应用对应的 SDKAppID 信息。
5. 单击【 快速上手 】，加载完成后，记录出现的 **UserSig 的密钥**。

> **注意：**
>
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 UserSig，该UserSig 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此 **该方法仅适合本地跑通 Demo 和功能调试** 。
> - 正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 **服务端生成 UserSig**。

6. 在播放端，推荐使用CDN播放，所以需要在 **实时音视频控制台** 开启**旁路推流**功能。


> **说明：**
>
> 在服务开通后，建议先可以编译和体验一下上述示例代码章节中腾讯云提供的移动直播的 API-Example 工程，配合上下文可以快速的了解相关 API 的使用。

## 步骤2：V2TXLivePusher RTC 推流

### URL 拼接

具体的推流 URL字符串，需要开发者按照下方协议解析中的规则，在工程代码中自行拼接。URL 的示例如下：

**推流 URL**

```
trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx
```

在上述的 URL 中，存在一些关键字段，关于其中关键字段的含义信息，详见下表：

| 字段名称 | 字段含义 |
| --- | --- |
| trtc:// | 互动直播推流 URL 的前缀字段 |
| cloud.tencent.com | 互动直播特定域名，**请勿修改** |
| push | 标识位，表示推流 |
| streamid | 流 ID，需要由开发者自定义 |
| sdkappid | 对应 服务开通 一节中生成的 SDKAppID |
| userId | 主播 ID，需要由开发者自定义 |
| usersig | 对应 服务开通 中获取的 UserSig 密钥 |

**示例代码**

```
// 创建⊠个 V2TXLivePusher 对象，并指定模式为 TXLiveMode_RTC；
V2TXLivePusher pusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
pusher.setObserver(new MyPusherObserver());
pusher.setRenderView(mSurfaceView);
pusher.startCamera(true);
pusher.startMicrophone();
// 传⊠互动直播RTC推流协议地址，即可开始推流，其中streamid设置为自定义值，比如12345687；；
pusher.startPush("trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=finnguan&usersig=xxxxx"
);
```

### 步骤3：V2TXLivePlayer CDN 播放

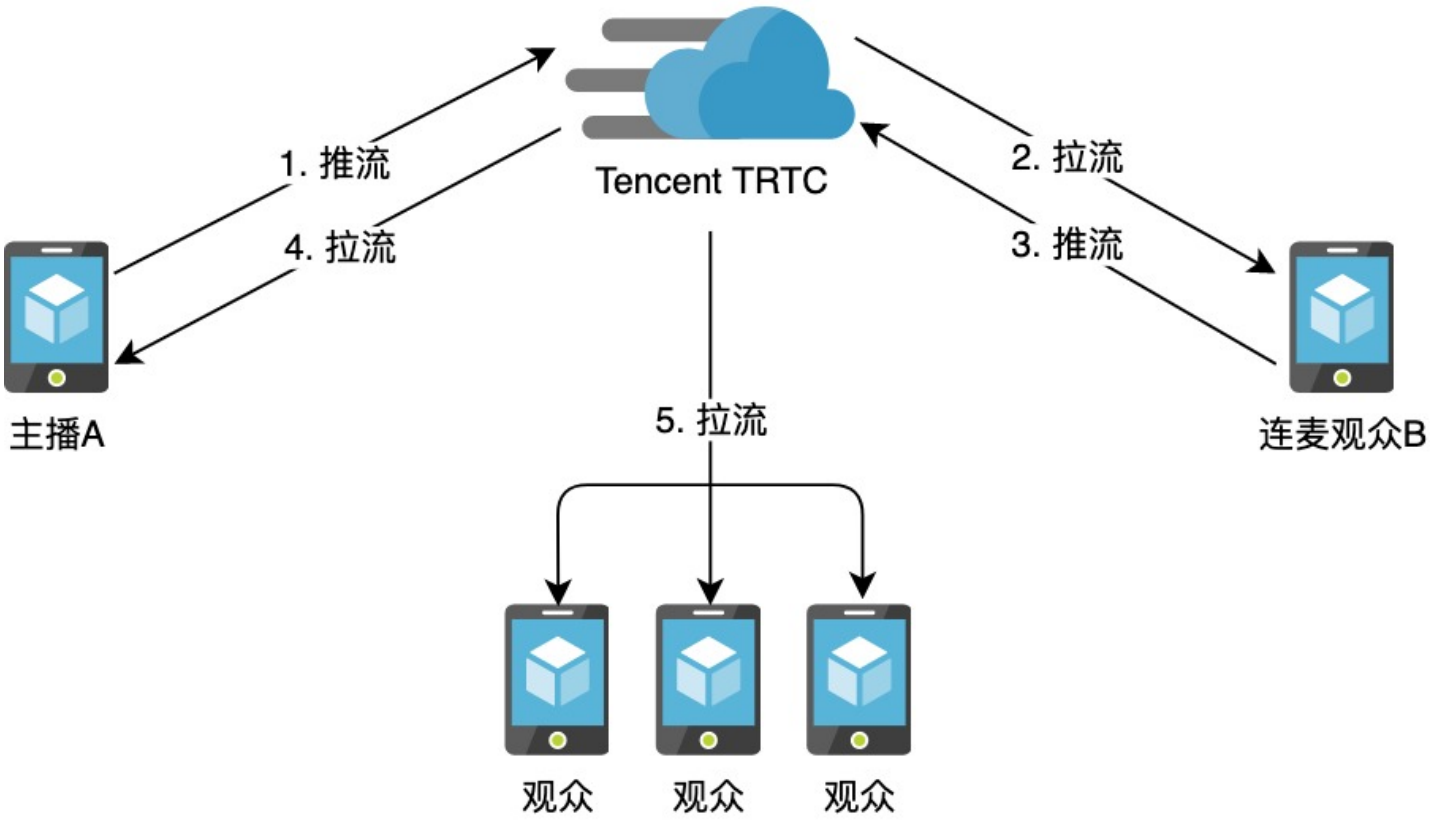
#### URL 拼接

具体的播放 URL 字符串，需要开发者按照 域名 + streamID，在工程代码中自行拼接。

**示例代码**

```
// 创建⊠个 V2TXLivePlayer 对象；
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传⊠互动直播播放协议地址，即可开始播放，streamid对应推流时设置的streamid，比如12345687；
player.startPlay("https://3891.liveplay.myqcloud.com/live/streamid.flv");
```

### 步骤4：实现观众连麦




**1. 主播 RTC 推流**

**主播 A 开始推流，调用 V2TXLivePusher 组件开始主播 A 的推流。**

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
/**
* pushURLA= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
pusherA.startPush(pushURLA);
```

**2. 观众 CDN 拉流**

**所有观众观看主播 A 直播，调用 V2TXLivePlayer 开始播放主播 A 的流。**

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
* 这里使用CDN拉流，支持flv，hls，webrtc协议，任选一种协议。flv，hls等标准协议价格更合理，webrtc快直播能够提供更低延迟的互动体验。
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidA.flv";
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidA.hls";
* playURLA= "webrtc://3891.liveplay.myqcloud.com/live/streamidA"
*/
playerA.startPlay(playURLA);
```

**3. 观众发起连麦**

**其中观众 B 调用 V2TXLivePusher 发起推流（后续会称呼为连麦观众B）。**

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this,V2TXLiveMode.TXLiveMode_RTC);
...
/*
* pushURLB= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
pusherB.startPush(pushURLB);
```

**4. 进入连麦状态**

**主播 A 调用 V2TXLivePlayer 使用RTC协议拉取放连麦观众 B 的流。**

```
V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);
...
/**
* playURLB= "trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
playerB.startPlay(playURLB);
```

**同时，连麦观众 B 调用 V2TXLivePlayer 切换至 RTC 协议，开始播放主播 A 的流。**

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
*playURLA= "trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
playerA.startPlay(playURLA);
```

**此时主播 A 和连麦观众 B 即可进入超低延时的实时互动场景中。**

**5. 连麦成功后，进行混流**

**为了保证观众可以看到连麦观众B的画面，这里主播 A 需要发起一次混流操作。也就是将主播A自己和连麦观众 B，混合成一路流。观众可以在一路流上看到主播和连麦观众进行互动。 A 调用 setMixTranscodingConfig 接口启动云端混流，调用时需要设置音频相关的参数，例如 音频采样率 audioSampleRate 、音频码率 audioBitrate 和 声道数 audioChannels 等。**
**如果您的业务场景中也包含视频，需同时设置视频相关的参数，例如 视频宽度 videoWidth 、视频高度 videoHeight 、视频码率 videoBitrate 、视频帧率 videoFramerate 等。**

**示例代码：**

```
V2TXLiveDef.V2TXLiveTranscodingConfig config = new V2TXLiveDef.V2TXLiveTranscodingConfig();
// 设置分辨率为 720 × 1280，码率为 1500kbps，帧率为 20FPS
config.videoWidth = 720;
config.videoHeight = 1280;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
```

```
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;
config.mixStreams = new ArrayList<>();
// 主播摄像头的画面位置
V2TXLiveDef.V2TXLiveMixStream local = new V2TXLiveDef.V2TXLiveMixStream();
local.userId = "localUserId";
local.streamId = null; // 本地画面不用填写 streamID，远程需要
local.x = 0;
local.y = 0;
local.width = videoWidth;
local.height = videoHeight;
local.zOrder = 0; // zOrder 为 0 代表主播画面位于最底层
config.mixStreams.add(local);
// 连麦者的画面位置
V2TXLiveDef.V2TXLiveMixStream remoteA = new V2TXLiveDef.V2TXLiveMixStream();
remoteA.userId = "remoteUserIdA";
remoteA.streamId = "remoteStreamIdA"; // 本地画面不用填写 streamID，远程需要
remoteA.x = 400; //仅供参考
remoteA.y = 800; //仅供参考
remoteA.width = 180; //仅供参考
remoteA.height = 240; //仅供参考
remoteA.zOrder = 1;
config.mixStreams.add(remoteA);
// 连麦者的画面位置
V2TXLiveDef.V2TXLiveMixStream remoteB = new V2TXLiveDef.V2TXLiveMixStream();
remoteB.userId = "remoteUserIdB";
remoteB.streamId = "remoteStreamIdB"; // 本地画面不用填写 streamID，远程需要
remoteB.x = 400; //仅供参考
remoteB.y = 800; //仅供参考
remoteB.width = 180; //仅供参考
remoteB.height = 240; //仅供参考
remoteB.zOrder = 1;
config.mixStreams.add(remoteB);
// 发起云端混流
pusher.setMixTranscodingConfig(config);
```

> ⚠ **注意：**
> 发起云端混流后，默认混流 ID，是发起混流者的 ID，如果需要指定流 ID，需要进行传入。

这样其他其他观众在观看时，就可以看到 A，B 两个主播的连麦互动。

### 步骤5：实现主播 PK

1. 主播 A 开始推流，调用 V2TXLivePusher 组件开始主播 A 的推流。

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
```

```
/**
* pushURLA= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
pusherA.startPush(pushURLA);
```

2. 主播 B 开始推流，调用 V2TXLivePusher组件开始主播 B 的推流。

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
/**
* pushURLB "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
pusherB.startPush(pushURLB);
```

3. 开始 PK，主播 A 和主播 B 分别调用 V2TXLivePlayer 开始播放对方的流，此时主播 A 和主播 B 即进入 RTC 连麦互动直播场景中。

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
* 这里使用CDN拉流，支持flv，hls，webrtc协议，任选一种协议。flv，hls等标准协议价格更合理，webrtc快直播能够提供更低延迟的互动体验。
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidB.flv";
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidB.hls";
* playURLA= "webrtc://3891.liveplay.myqcloud.com/live/streamidB"
*/
playerA.startPlay(playURLA);

V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);
...
/**
* 这里使用CDN拉流，支持flv，hls，webrtc协议，任选一种协议。flv，hls等标准协议价格更合理，webrtc快直播能够提供更低延迟的互动体验。
* playURLB= "http://3891.liveplay.myqcloud.com/live/streamidA.flv";
* playURLB= "http://3891.liveplay.myqcloud.com/live/streamidA.hls";
* playURLB= "webrtc://3891.liveplay.myqcloud.com/live/streamidA"
*/
playerB.startPlay(playURLB);
```

4. PK 成功后，主播 A 和主播 B 的观众各自调用 V2TXLivePlayer 开始播放另外一名主播的推流内容。

> 说明：
> 此处开发者可能会有疑问：貌似新的 RTC 连麦方案还需要我们自己维护一套房间和用户状态，这样不是更麻烦吗？是的，**没有更好的方案，只有更适合自己的方案**，我们也有考虑到这样的场景：
>
> - 如果对时延和并发要求并不高的场景，可以继续使用连麦互动的旧方案。
> - 如果既想用到 V2 相关的接口，但是又不想维护一套单独的房间状态，可以尝试搭配 腾讯云 IM SDK，快速实现相关逻辑。

## RTC 连麦方案怎么计算费用

RTC 连麦互动直服务费用按所有参与连麦的用户产生的 视频时长 和 语音时长 来统计连麦服务产生的用量。

> ⚠ 注意：
> 时长统计精度为秒，以当月累计秒数转换成分钟数后进行计费，不足一分钟按一分钟计。

### 视频时长

视频时长是指所有用户观看其他参与连麦的用户视频流的累计时间。RTC 连麦会根据用户**实际接收到的视频分辨率**划分视频档位，然后分别对不同档位的视频时长进行计费。

**视频档位与分辨率的对应关系如下表所示：**

| 视频档位 | 实际接收分辨率 |
| --- | --- |
| 标清 SD | 不高于640 × 480（含） |
| 高清 HD | 640 × 480 – 1280 × 720（含） |
| 超清 HD+ | 高于1280 × 720 |

- 用户观看视频时，不管该视频里面有没有包含音频，都只统计一次视频时长，不会重复计算语音时长。
- 单个用户同时观看多路视频流时，其观看的每一路视频时长将分别统计后叠加计算。

### 语音时长

语音时长指所有用户收听其他参与连麦的用户音频流的时间。

- 只有当用户没有观看视频流时，才会统计语音时长。
- 当用户同时收听多个用户音频流时，统计语音时长时会去除掉重叠部分的时长。

### 服务定价

视频互动直播服务的刊例价如下表所示：

| 计费项 | 单价（元/千分钟） |
| --- | --- |
| 语音 | 7.00 |
| 标清 SD | 14.00 |
| 高清 HD | 28.00 |
| 超清 HD+ | 105.00 |

### 计费方式

即支付方式，RTC 连麦互动直播支持**预付费套餐包**和**后付费**，默认采用预付费套餐包，后付费只能通过购买的套餐包消耗完或过期后自动开通，无法直接开通。

#### 预付费套餐包

RTC连麦互动直服务为您提供音视频通用套餐包，可按照**1:2:4:15**分别抵扣语音、标清 SD、高清 HD 和超清 HD+ 时长，例如1分钟高清视频时长扣除4分钟通用套餐包时长。
通用套餐包定价如下表所示：

| 套餐包类型 | 套餐包时长（千分钟） | 刊例价（元/千分钟） | 套餐内单价（元/千分钟） | 套餐包价格（元） | 折扣 |
| --- | --- | --- | --- | --- | --- |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 固定套餐包 | 25 | 7.00 | 6.720 | 168.00 | 96% |
| | 250 | 7.00 | 6.352 | 1588.00 | 91% |
| | 1000 | 7.00 | 5.968 | 5968.00 | 85% |
| | 3000 | 7.00 | 5.630 | 16888.00 | 80% |
| 自定义套餐包 | 0 < X < 25 | 7.00 | 7.000 | 套餐内单价乘以套餐包时长 X | 100% |
| | 25 ≤ X < 250 | 7.00 | 6.720 | | 96% |
| | 250 ≤ X < 1000 | 7.00 | 6.352 | | 91% |
| | 1000 ≤ X < 3000 | 7.00 | 5.968 | | 85% |
| | X ≥ 3000 | 7.00 | 5.630 | | 80% |

> 说明：
> 表格中套餐内单价按每千分钟单价向上取整精确到小数点后3位，实际计费按每分钟单价精确到小数点后8位。

通用套餐包说明：

- 通用套餐包的有效期为购买当日 – 次年当月最后一天。
  例如，2020年05月01日购买的套餐包，其有效时间为2020年05月01日 – 2021年05月31日。
- 通用套餐包可以叠加购买，根据各类型用量实际产生的时间实时从通用套餐包中扣除相应分钟数，优先使用先过期的套餐包进行抵扣。
- 新购套餐包支付成功后5分钟左右生效，**新购套餐包生效后会立即扣除购买新套餐包当日0点起产生的未被其他套餐包抵扣过的用量**。
- 为了不影响您线上业务的正常运行，**通用套餐包用完或过期后不会自动停服**，超出套餐包的用量将采用 后付费 的计费方式。
- 所有通用套餐包到期后未消耗的分钟数将自动清零且无法恢复。

**后付费**

当服务用量无套餐包可抵扣或超出套餐包余量时，将采用后付费的方式，按 刊例价 计费。
RTC 连麦互动直播服务后付费有 日结 和 月结 两种结算周期。

- **日结后付费：**
  2020年09月01日起首次在 实时音视频 控制台创建 应用 的用户，后付费生效后默认采用**日结**方式结算。按日计费，每天上午10点扣除前一天产生的费用。

- **月结后付费：**
  2020年08月31日及之前首次在实时音视频控制台创建 应用 的用户，后付费生效后默认采用**月结**方式结算。按月计费，每月01日 – 05日从您的账户余额中扣除前一月产生的费用，详情以 计费账单 为准。

> 注意：
> - 后付费只能通过先购买套餐包，待购买的所有套餐包都消耗完或过期后自动开通，无法直接开通。
> - 结算周期无法自主变更，若您希望将月结变更为日结，可以 提交工单 寻求帮助。
> - 若您的账户因余额不足而无法抵扣账单费用时，您使用的其他腾讯云服务也可能会因为账户欠费而自动停服。例如，云端录制依赖**云直播**和**云点播**，如果腾讯云账户欠费，将导致云端录制失败。

**计费示例**

> 注意：

> 本文计费示例采用刊例价计算，您可以通过 购买套餐包 的方式节省费用。

**纯语音连麦示例**

假设用户 A、B、C 三人使用RTC连麦互动直播服务连麦30分钟。A、B、C 三人始终没有接收视频画面。
则产生的语音时长**总费用为** 语音时长单价 × 所有用户语音时长之和 = 7.00元/千分钟 ×（30分钟 + 30分钟 + 30分钟）/ 1000 = 0.63 元 。

**纯视频连麦示例**

假设用户 A、B 使用RTC连麦互动直播服务连麦45分钟。A、B 都始终观看对方的视频流。A 、B 实际观看到的视频分辨率如下表所示：

| 时间 | A 接收 B 的分辨率 | B 接收 A 的分辨率 |
| --- | --- | --- |
| 前30分钟 | 1280 × 720（高清） | 1920 × 1080（超清） |
| 后15分钟 | 640 × 360（标清） | 640 × 360（标清） |

**分析：**

- A 产生的用量及费用：
  - A 观看 B 的分辨率前30分钟位于高清档，后15分钟位于标清档。
  - A 产生的费用为 高清视频时长单价 × 高清视频时长 + 标清视频时长单价 × 标清视频时长 = 28元/千分钟 ×（30分钟 / 1000）+ 14 元/千分钟 ×（15分钟 / 1000）= 1.05元。
- B 产生的用量及费用：
  - B 观看 A 的分辨率前30分钟位于超清档，后15分钟位于标清档。
  - B 产生的费用为 超清视频时长单价 × 超清视频时长 + 标清视频时长单价 × 标清视频时长 = 105元/千分钟 ×（30分钟 / 1000）+ 14 元/千分钟 ×（15分钟 / 1000）= 3.36元。

则产生的**总费用为** 用户 A 产生的费用 + 用户 B 产生的费用 = 4.41元 。

**语音时长和视频时长混合示例**

假设用户 A、B 使用RTC连麦互动直播服务连麦45分钟。A 始终观看 B 的视频流；B 前30分钟观看了 A 的视频流，后15分钟没有观看 A 的视频流 仅收听 A 的音频流。A 、B 实际接收到的视频分辨率如下表所示：

| 时间 | A 接收 B 的分辨率 | B 接收 A 的分辨率 |
| --- | --- | --- |
| 前30分钟 | 1280 × 720（高清） | 1920 × 1080（超清） |
| 后15分钟 | 640 × 360（标清） | 语音（无视频） |

**分析：**

- A 产生的用量及费用：
  - A 接收 B 的分辨率前30分钟位于高清档，后15分钟位于标清档。
  - A 产生的费用为 高清视频时长单价 × 高清视频时长 + 标清视频时长单价 × 标清视频时长 = 28元/千分钟 ×（30分钟 / 1000） + 14 元/千分钟 ×（15分钟 / 1000）= 1.05元。
- B 产生的用量及费用：
  - B 接收 A 的分辨率前30分钟位于超清档，后15分钟没有接收 A 的视频流。
  - B 产生的费用为 超清视频时长单价 × 超清视频时长 + 语音时长单价 × 语音时长 = 105元/千分钟 ×（30分钟 / 1000）+ 7元/千分钟 ×（15分钟 / 1000）= 3.255元。

则产生的**总费用为** 用户 A 产生的费用 + 用户 B 产生的费用 = 4.305元 。

## 常见问题

**1. 为什么使用 V2TXLivePusher&V2TXLivePlayer 接口时，同一台设备不支持使用相同 streamid 同时推流和拉流，而 TXLivePusher&TXLivePlayer 可以支持?**

是的，目前 V2TXLivePusher&V2TXLivePlayer 是 腾讯云 TRTC 协议实现，其基于 UDP 的超低延时的私有协议暂时还不支持**同一台设备，使用相同的 streamid，一边推超低延时流，一边拉超低延时的流**，同时考虑到用户的使用场景，所以暂时并未支持，后续会酌情考虑此问题的优化。

**2. 服务开通 章节中生成参数都是什么意思呢?**

SDKAppID 用于标识您的应用，UserID 用于标识您的用户，而 UserSig 则是基于前两者计算出的安全签名，它由 HMAC SHA256 加密算法计算得出。只要攻击者不能伪造 UserSig，就无法盗用您的云服务流量。UserSig 的计算原理如下图所示，其本质就是对 SDKAppID、UserID、ExpireTime 等关键信息进行了一次哈希加密：

```
//UserSig 计算公式，其中 secretkey 为计算 usersig 用的加密密钥
usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
base64(userid + sdkappid + currtime + expire)))
```

**3. V2TXLivePusher&V2TXLivePlayer 如何设置音质或者画质呢?**

我们有提供对应的音质和画质的设置接口，详情见 API 文件：设置推流音频质量 和 设置推流视频参数。

**4. 收到一个错误码：-5，代表什么意思?**

-5表示由于许可证无效，因此无法调用API，对应的枚举值为：V2TXLIVE_ERROR_INVALID_LICENSE，更多错误码请参见 API 状态码。

**5. RTC连麦方案的时延性有可以参考的数据吗?**

新的 RTC 连麦方案中，主播连麦的延时 < 200ms，主播和观众的延时在 100ms - 1000ms。

**6. RTC 推流成功后，使用 CDN 拉流一直提示404?**

检查一下是否有开启实时音视频服务的旁路直播功能，基本原理是 RTC 协议推流后，如果需要使用 CDN 播放，RTC 会在后台服务中旁路流信息到 CDN 上。

# 旧版文档
# SDK 集成
# iOS

最近更新时间：2021-02-19 10:43:08

本文主要介绍如何快速地将腾讯云移动直播 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 移动直播专业版 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 => 'http://pod-1252463788.cosgz.myqcloud.com/liteavsdkspec/TXL
iteAVSDK_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
```

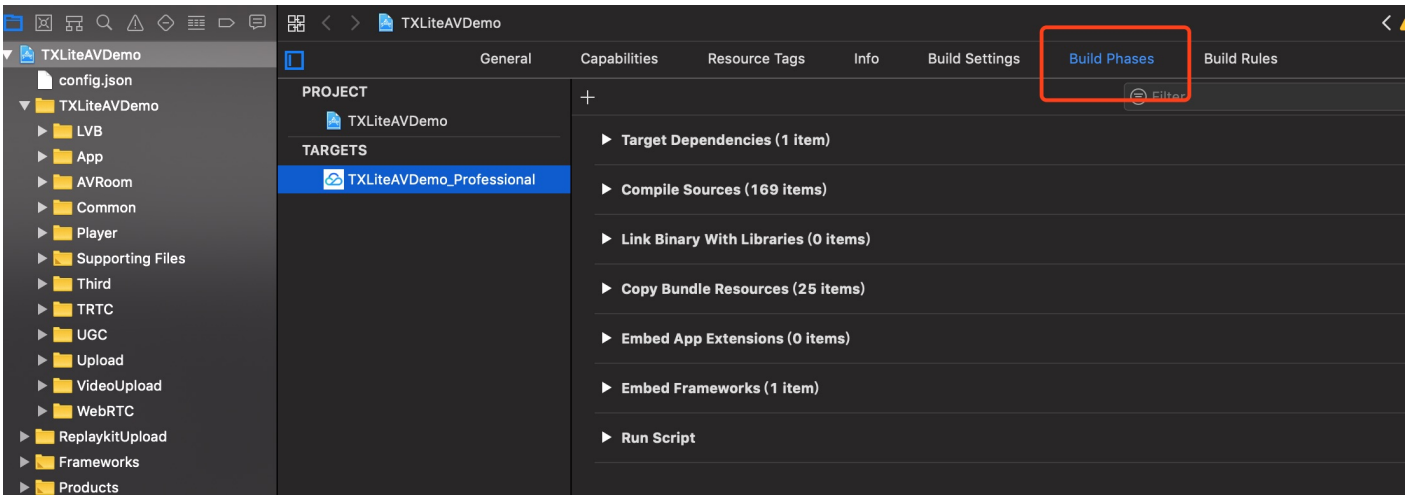
或使用以下命令更新本地库版本：

```
pod update
```

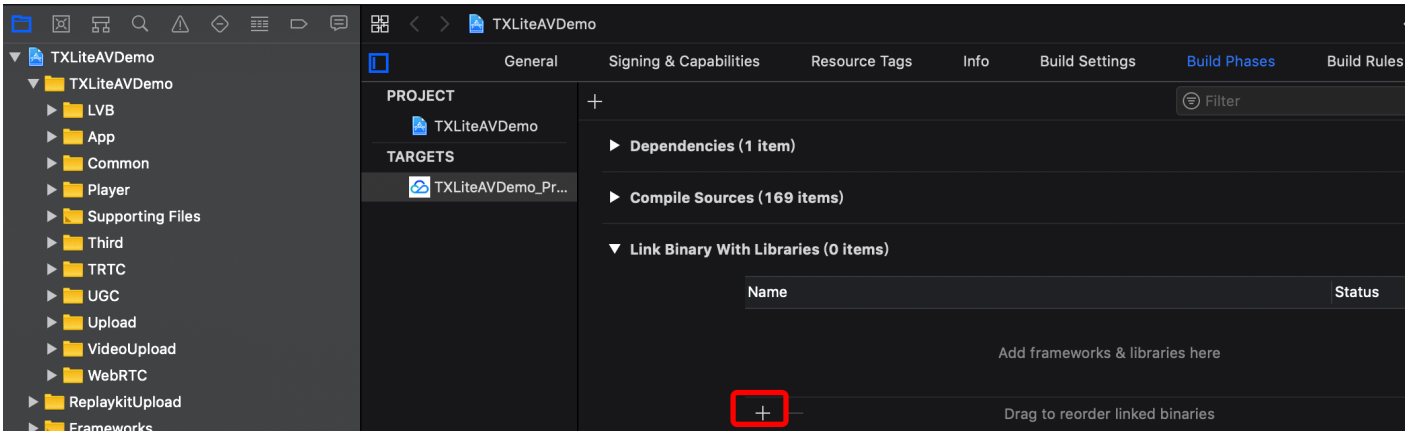
pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

### 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。

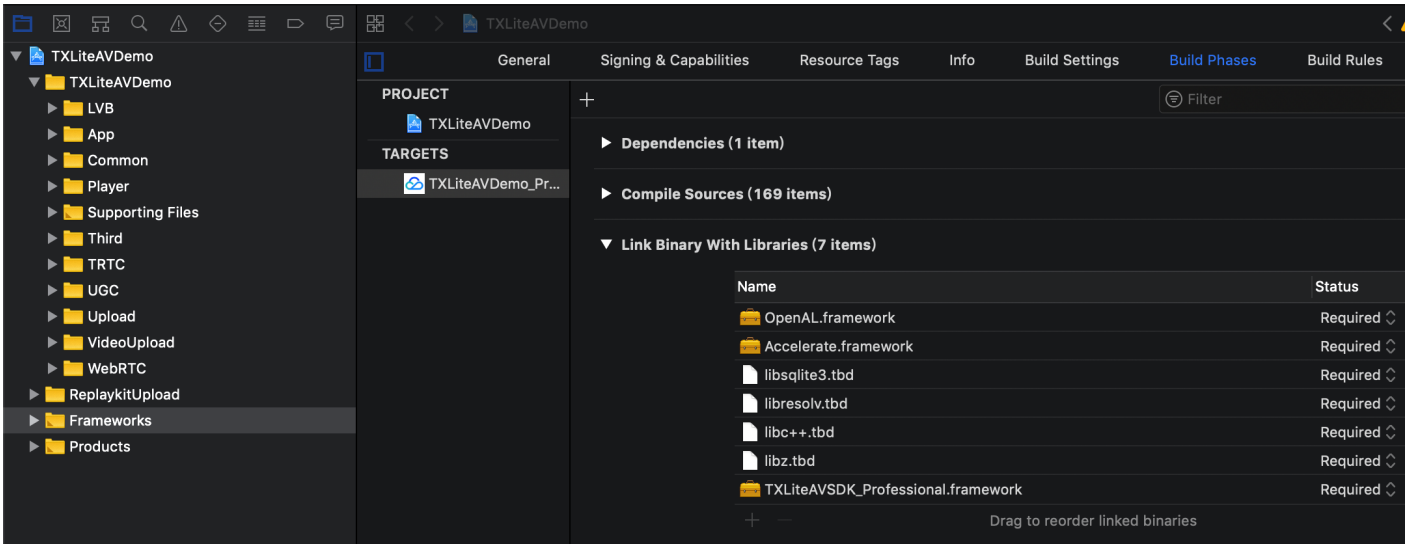



3. 单击 **Link Binary with Libraries** 项展开，单击底下的“+”添加依赖库。




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

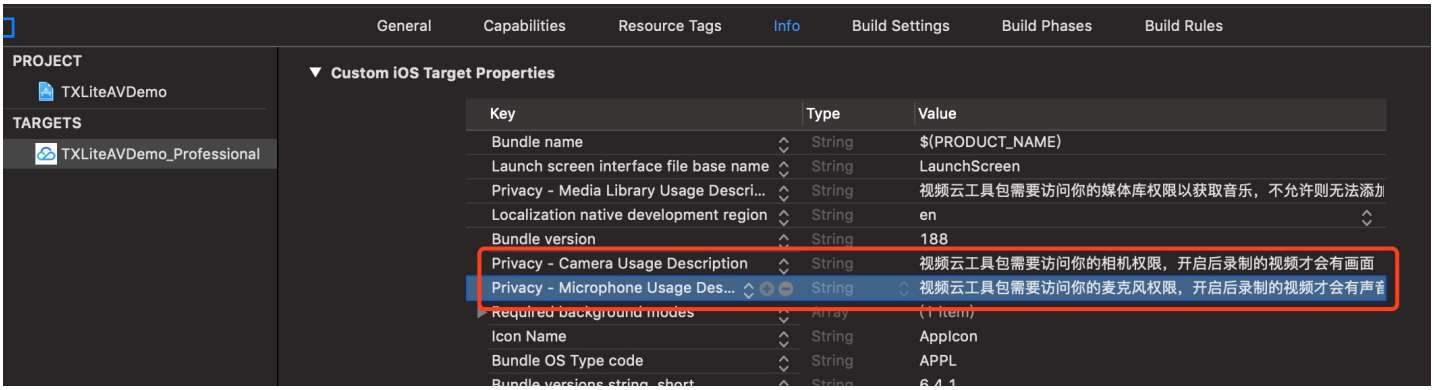
```
libz.tbd
libc++.tbd
libresolv.tbd
libsqlite3.tbd
Accelerate.framework
OpenAL.framework
```




## 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。

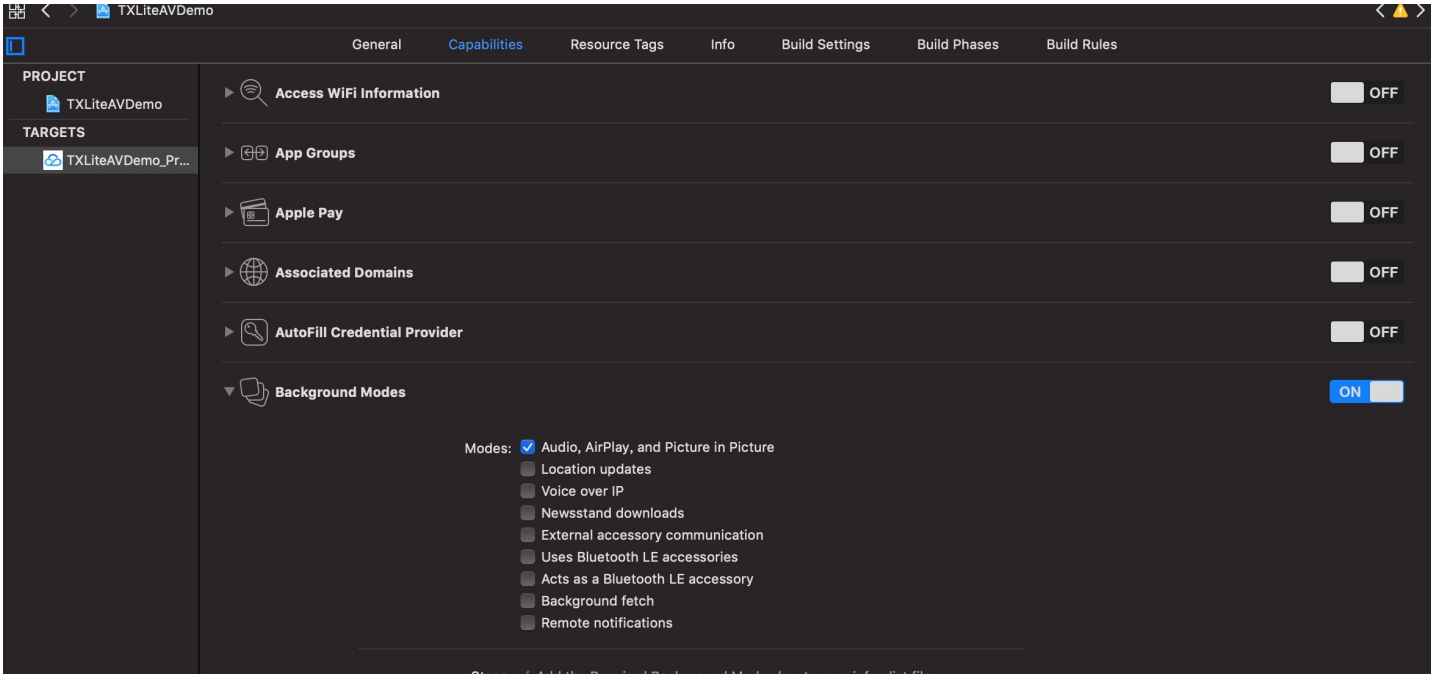
在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 - [AppDelegate application:didFinishLaunchingWithOptions:] 中）进行如下设置：

```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOption
s {
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：2021-07-23 18:43:33

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

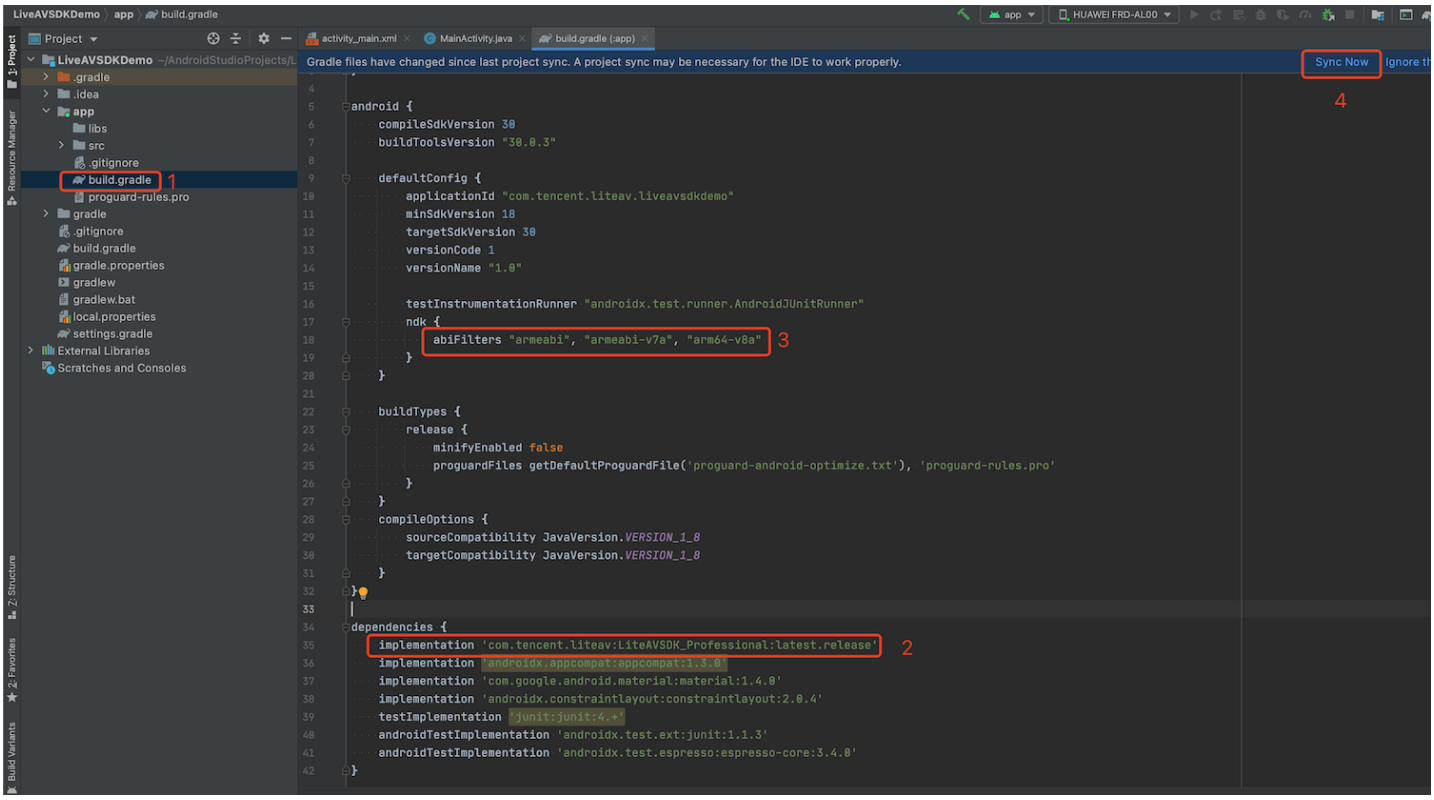
- Android Studio 2.0+。
- Android 4.1（SDK API 16）及以上系统。

## 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）。

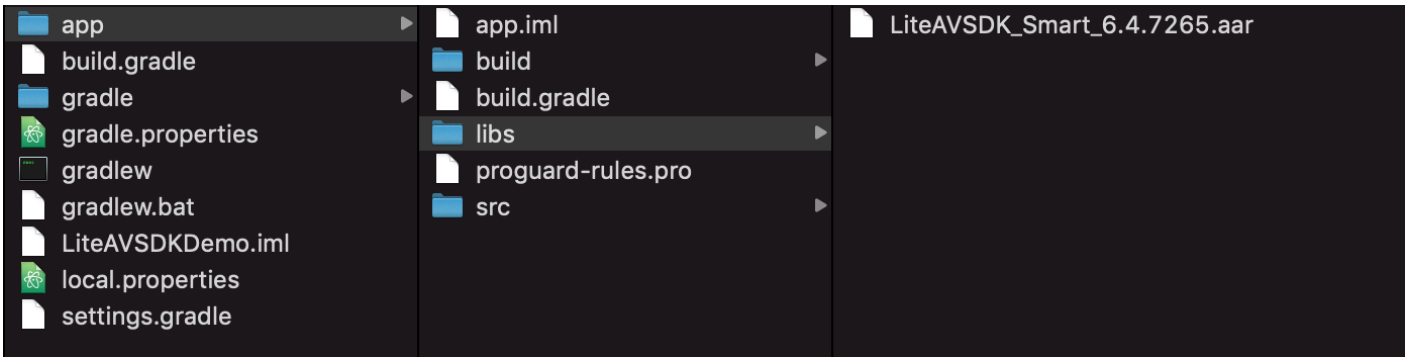
```
defaultConfig {
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
```

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。

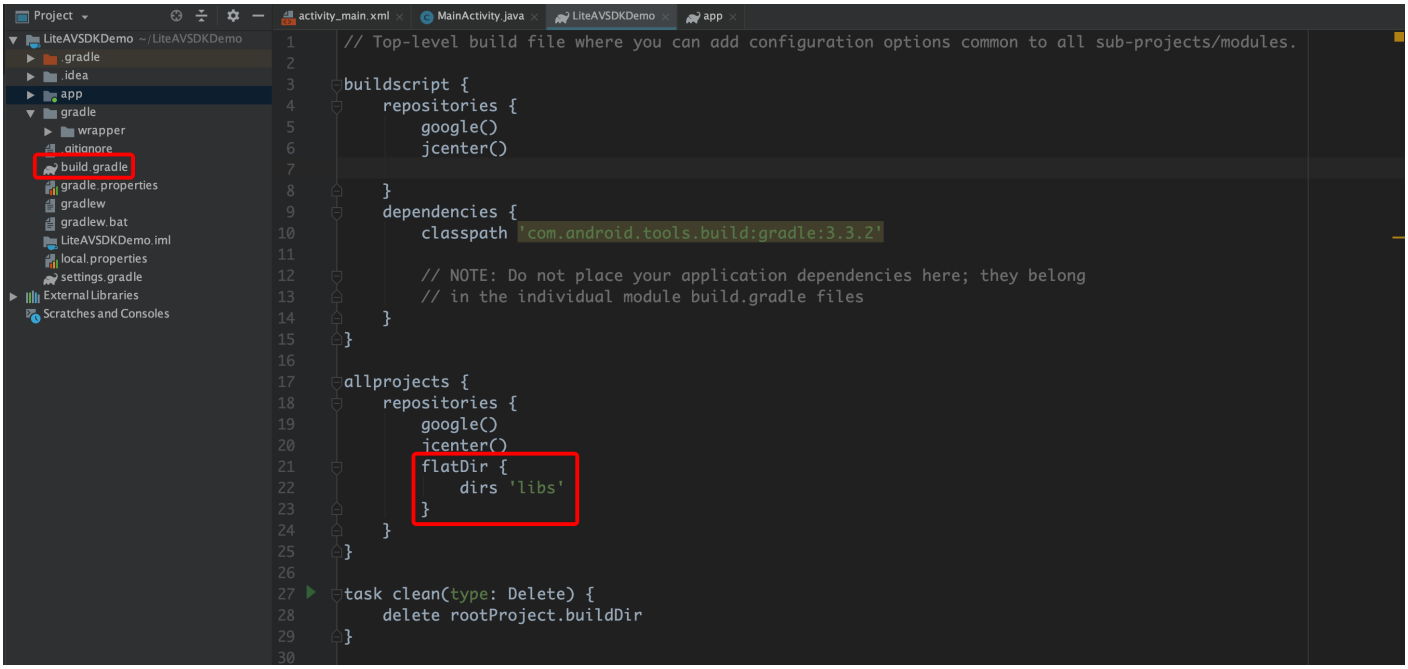
### 方法二：手动下载（aar）

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：

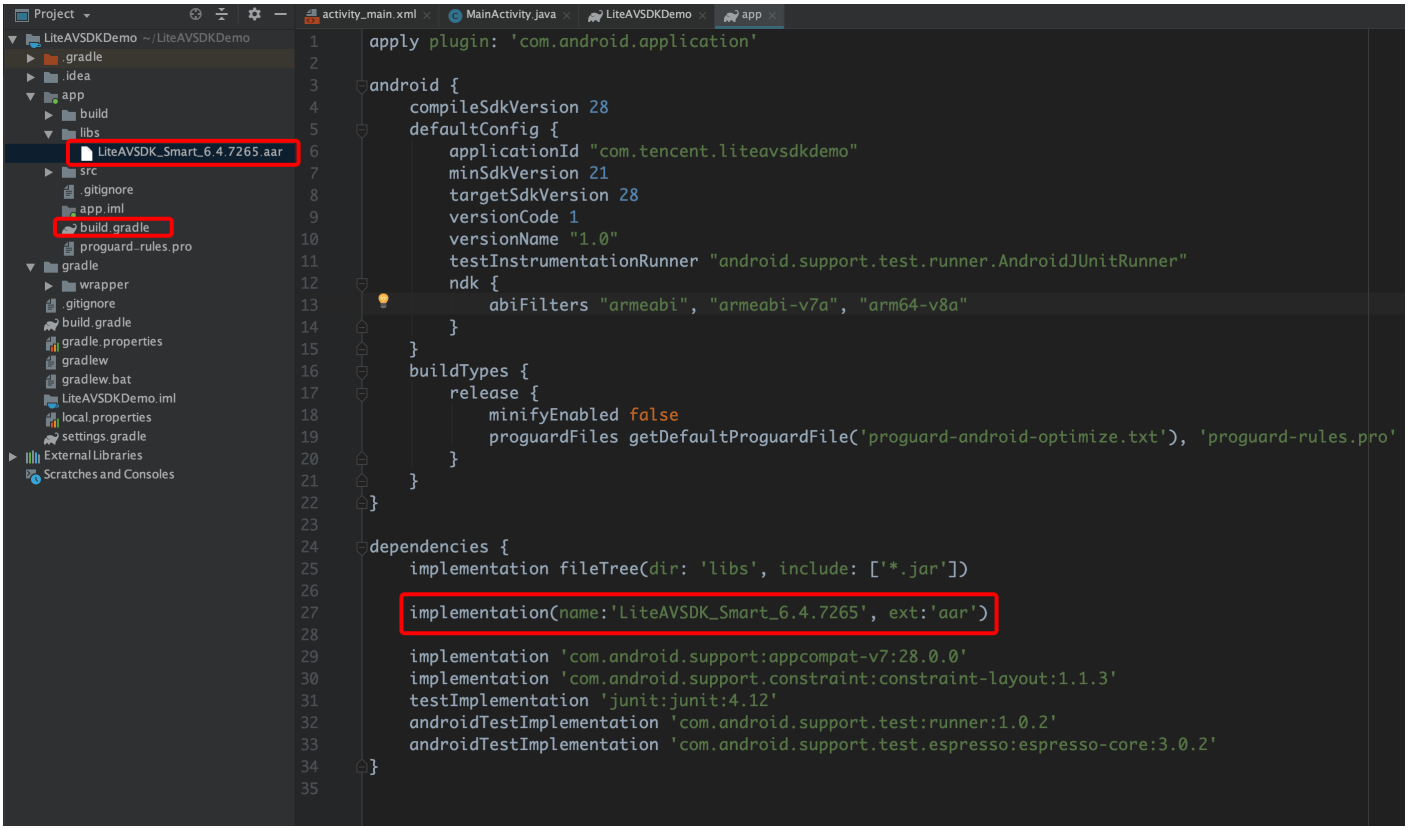
1. 下载 LiveAVSDK ，下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：




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）。

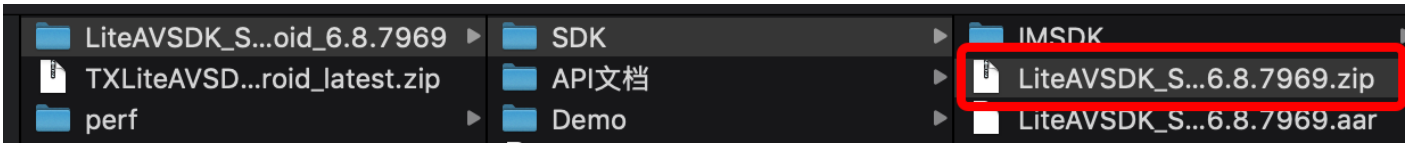
```
defaultConfig {
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
```

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

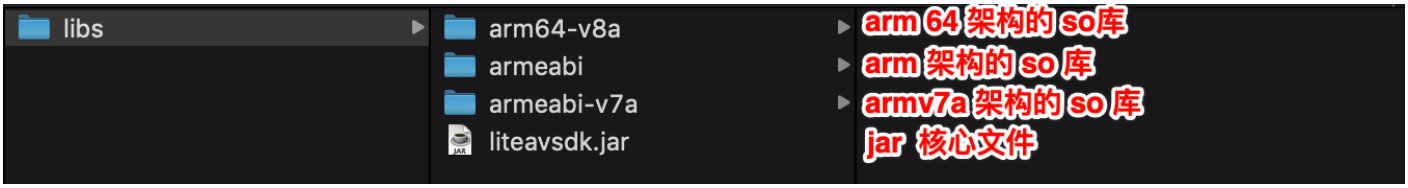
## 集成 SDK（jar）

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：

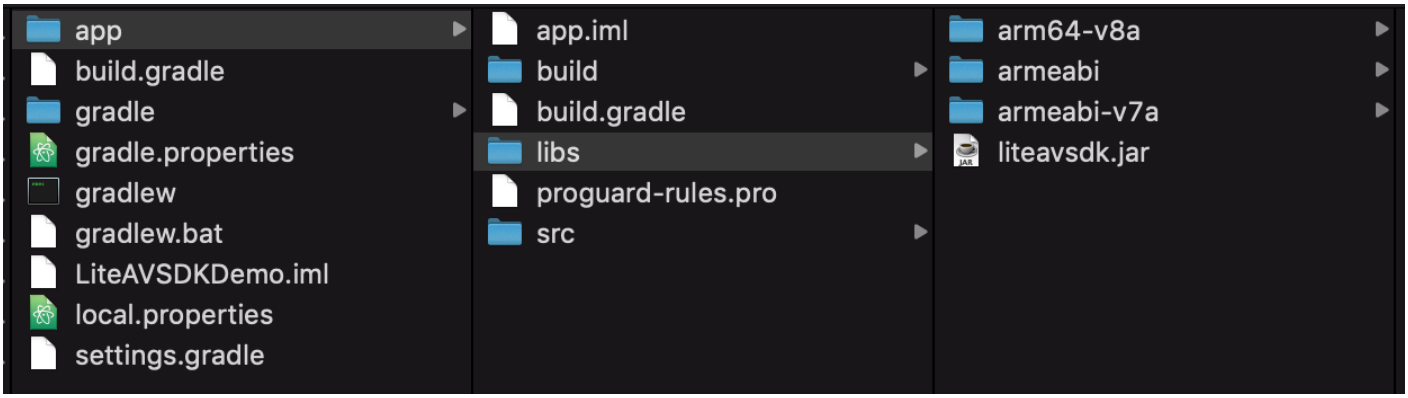
1. 下载 LiveAVSDK ，下载完成后进行解压。在 SDK 目录下找到 LiteAVSDK_Smart_xxx.zip（其中 xxx 为 LiteAVSDK 的版本号）：



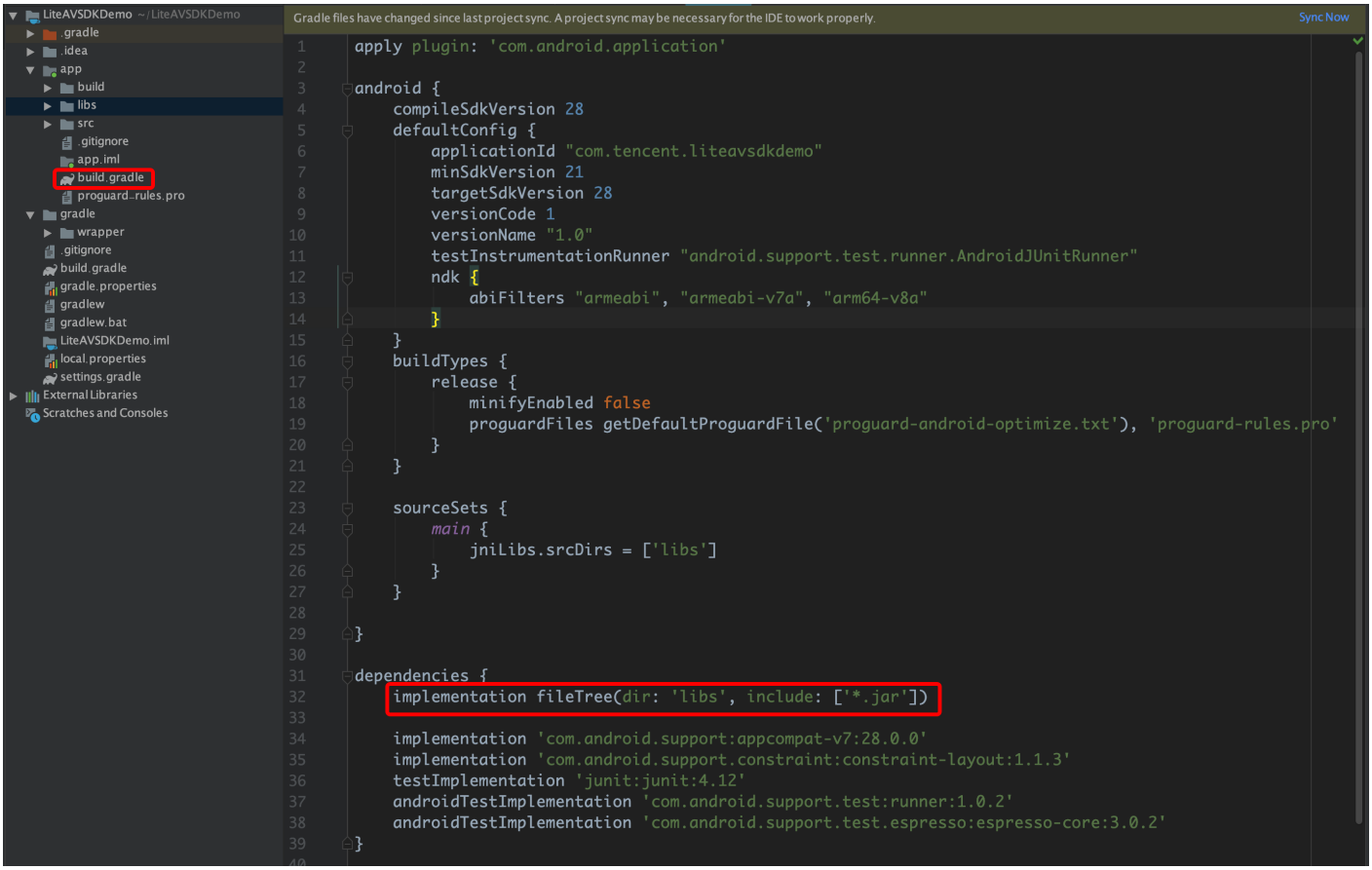
解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：




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。

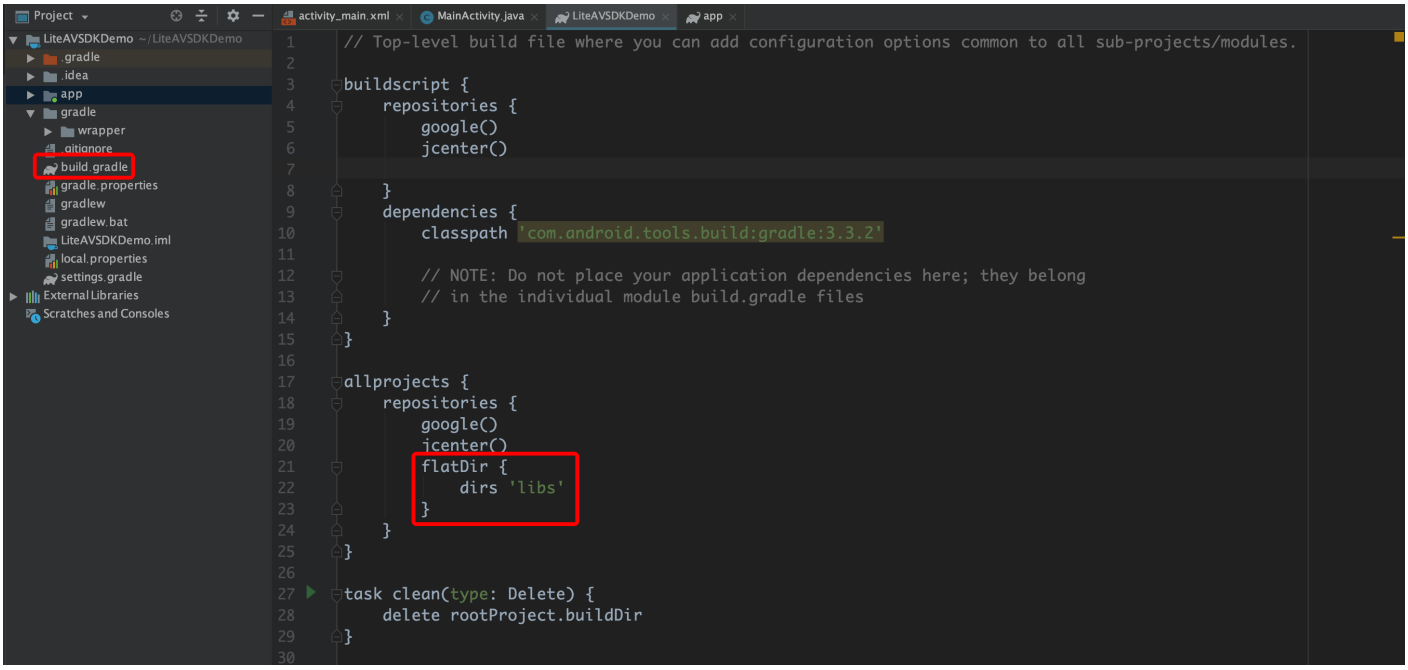



3. 在 app/build.gradle 中，添加引用 jar 库的代码。



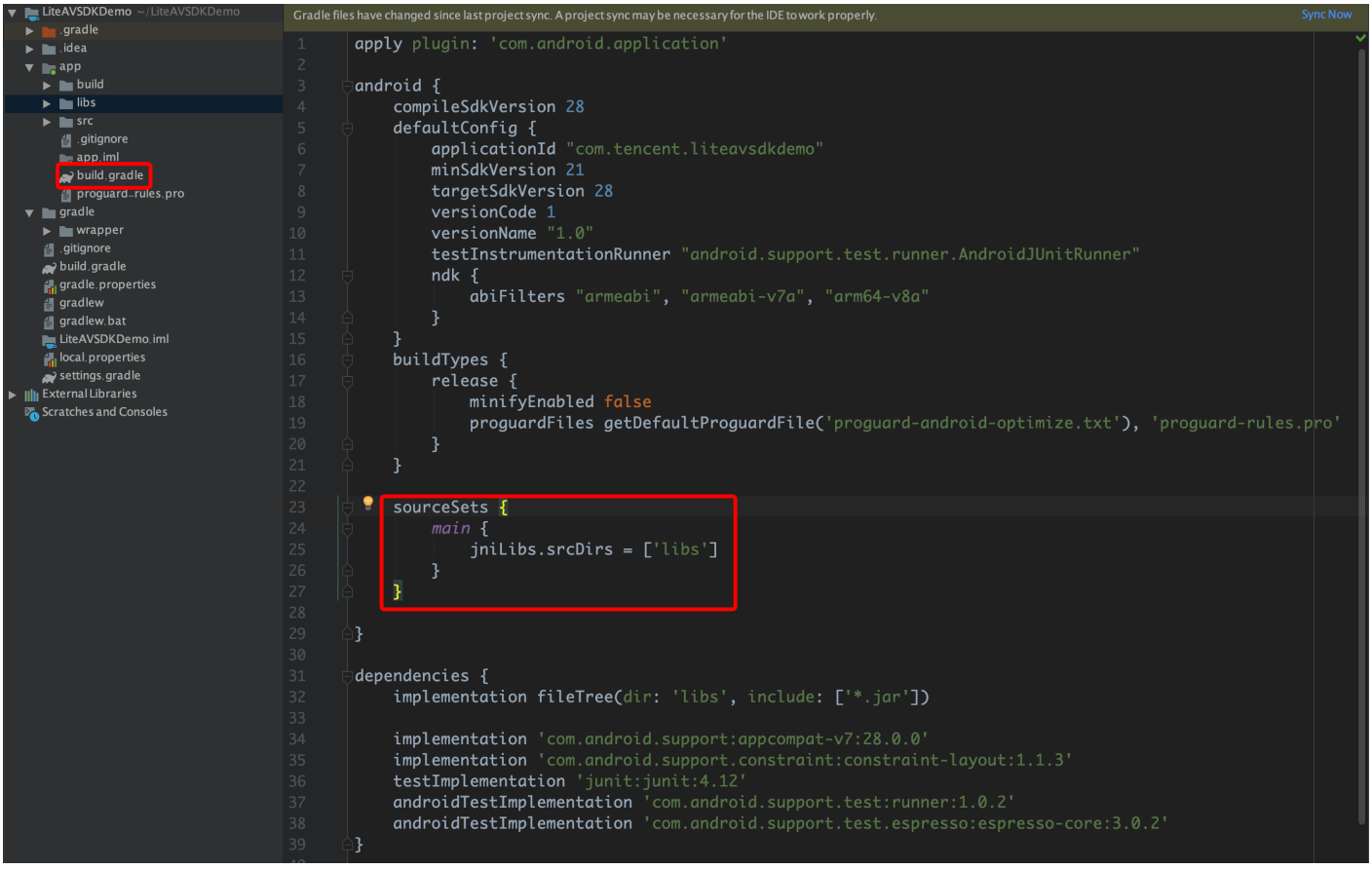

```
dependencies{
implementation fileTree(dir:'libs',include:['*.jar'])
}
```

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



```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.2'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

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

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

5. 在 app/build.gradle 中，添加引用 so 库的代码。



```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.tencent.liteavsdkdemo"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        ndk {
            abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
        }
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }

    sourceSets {
        main {
            jniLibs.srcDirs = ['libs']
        }
    }

}

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

    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

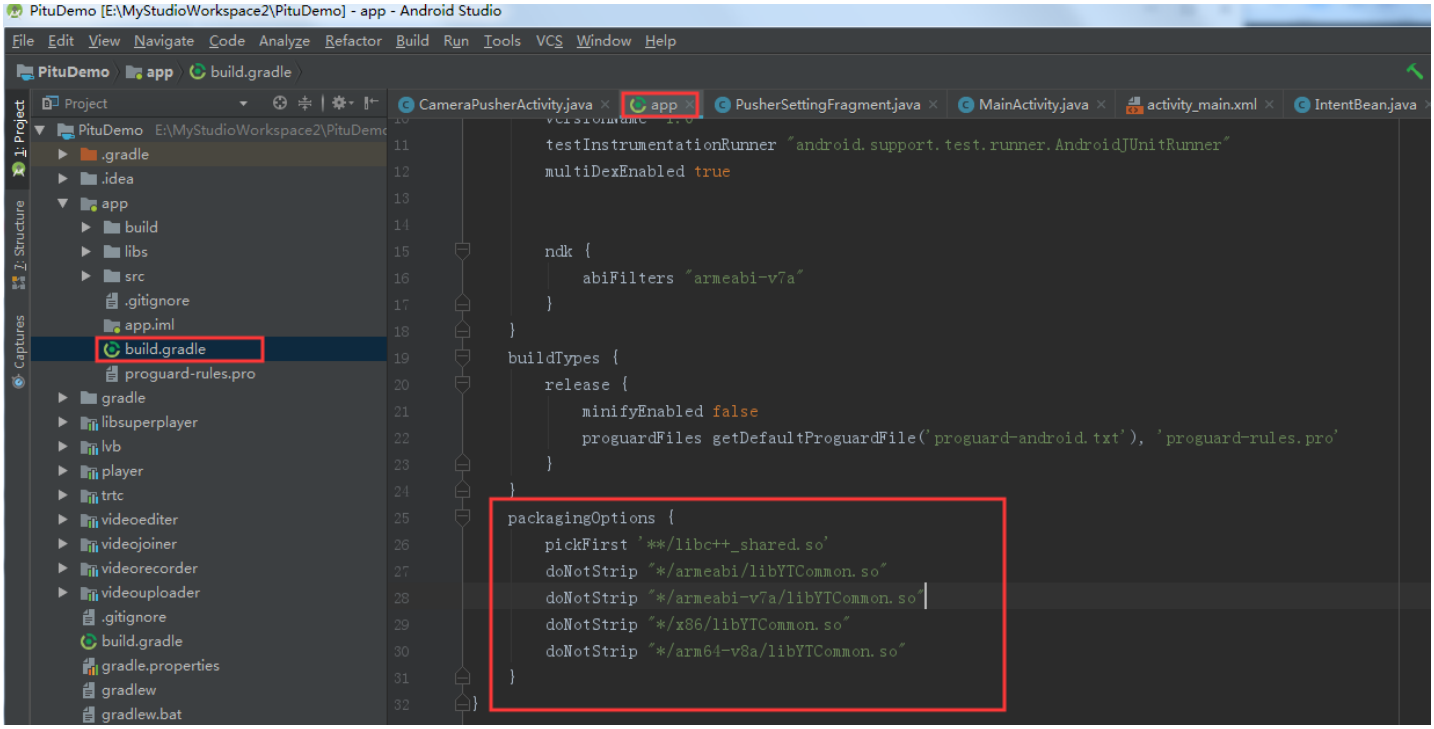
6. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构(目前 LiteAVSDK 支持 armeabi 、 armeabi-v7a 和 arm64-v8a) 。

```
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);
}
}
```

## 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：

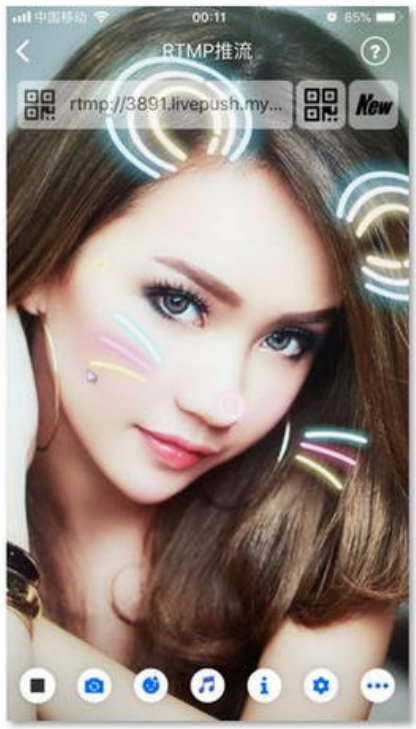
```
-keep class com.tencent.** { *; }
```

# 摄像头推流
# iOS

最近更新时间：2021-05-24 16:54:25

## 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

单击 License 申请 获取测试用的 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 *)launchOption
s {
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
```

### 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];
```

- **如何获取可用的推流 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

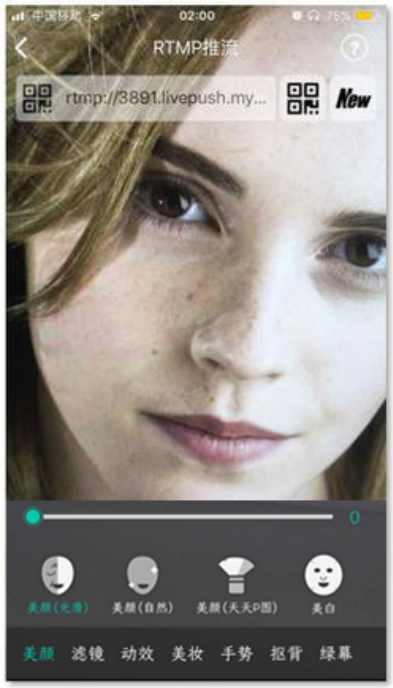
**TXLivePusher # setVideoQuality ( quality, adjustBitrate, *adjustResolution* );**

| 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 |

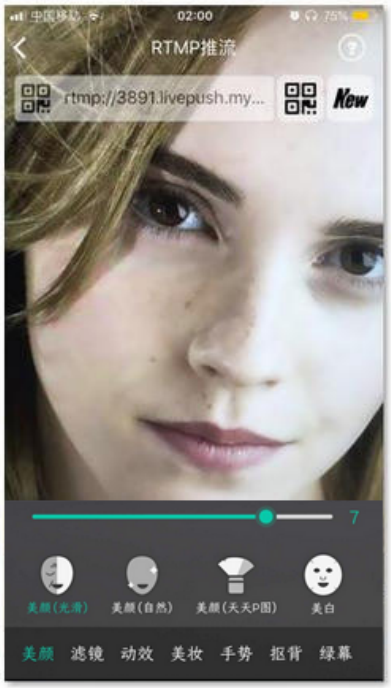
### 8. 美颜美白和红润特效

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）：

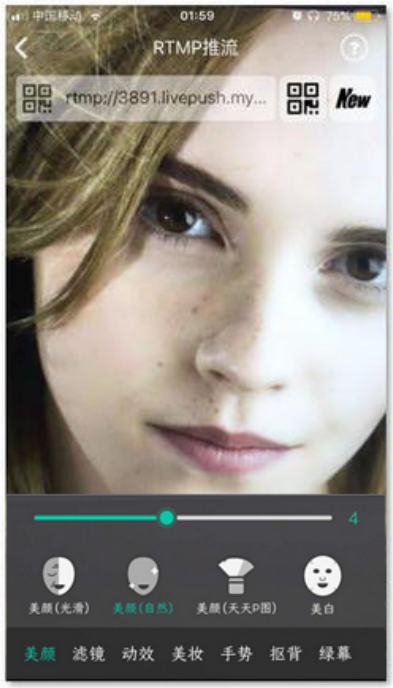
| 美颜风格 | 效果说明 |
| --- | --- |
| BEAUTY_STYLE_SMOOTH | 光滑风格，算法更加注重皮肤的光滑程度，适合秀场直播类场景下使用。 |
| BEAUTY_STYLE_NATURE | 自然风格，算法更加注重保留皮肤细节，适合对真实性要求更高的主播。 |




美颜关闭




美颜风格：光滑




美颜风格：自然

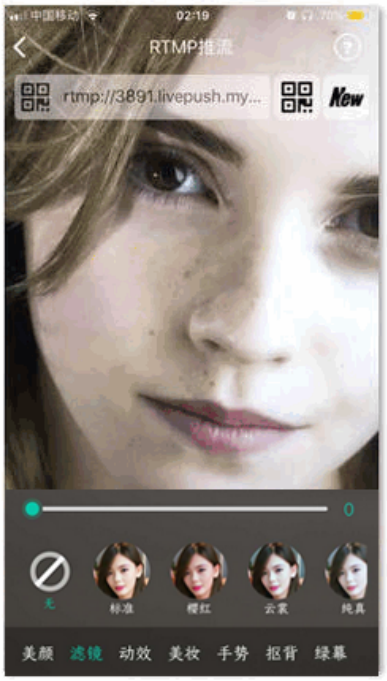
```
// beautyStyle : 美颜算法，目前支持 自然 和 光滑 两种。
// beautyLevel : 磨皮级别，取值范围 0 - 9； 0 表示关闭 1 - 9值越大 效果越明显。
// whitenessLevel : 美白级别，取值范围 0 - 9； 0 表示关闭 1 - 9值越大 效果越明显。
// ruddinessLevel : 红润级别，取值范围 0 - 9； 0 表示关闭 1 - 9值越大 效果越明显。
(void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel
whitenessLevel:(float)whitenessLevel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

### 9. 色彩滤镜效果

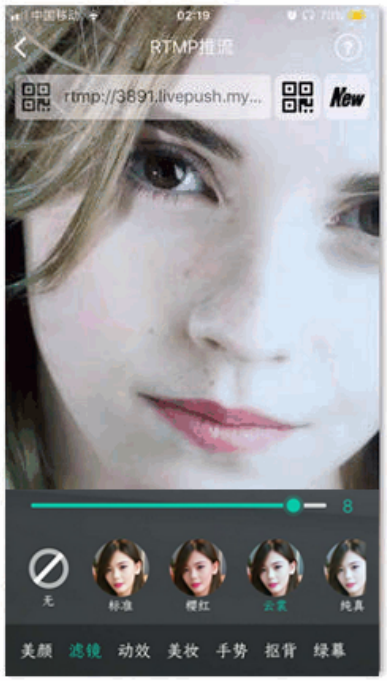
调用 TXLivePush 中的 setFilter 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。

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。

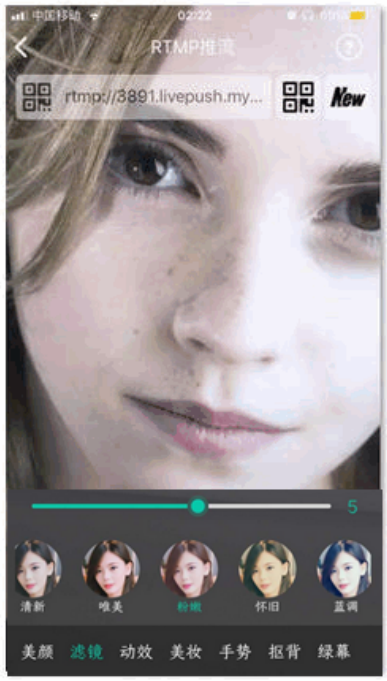
从手机 QQ 和 Now 直播的经验来看，单纯通过 setBeautyStyle 调整磨皮效果是不够的，只有将美颜效果和 setFilter 配合使用才能达到更加多变的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜，并将其默认打包在 Demo 中供您使用。




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

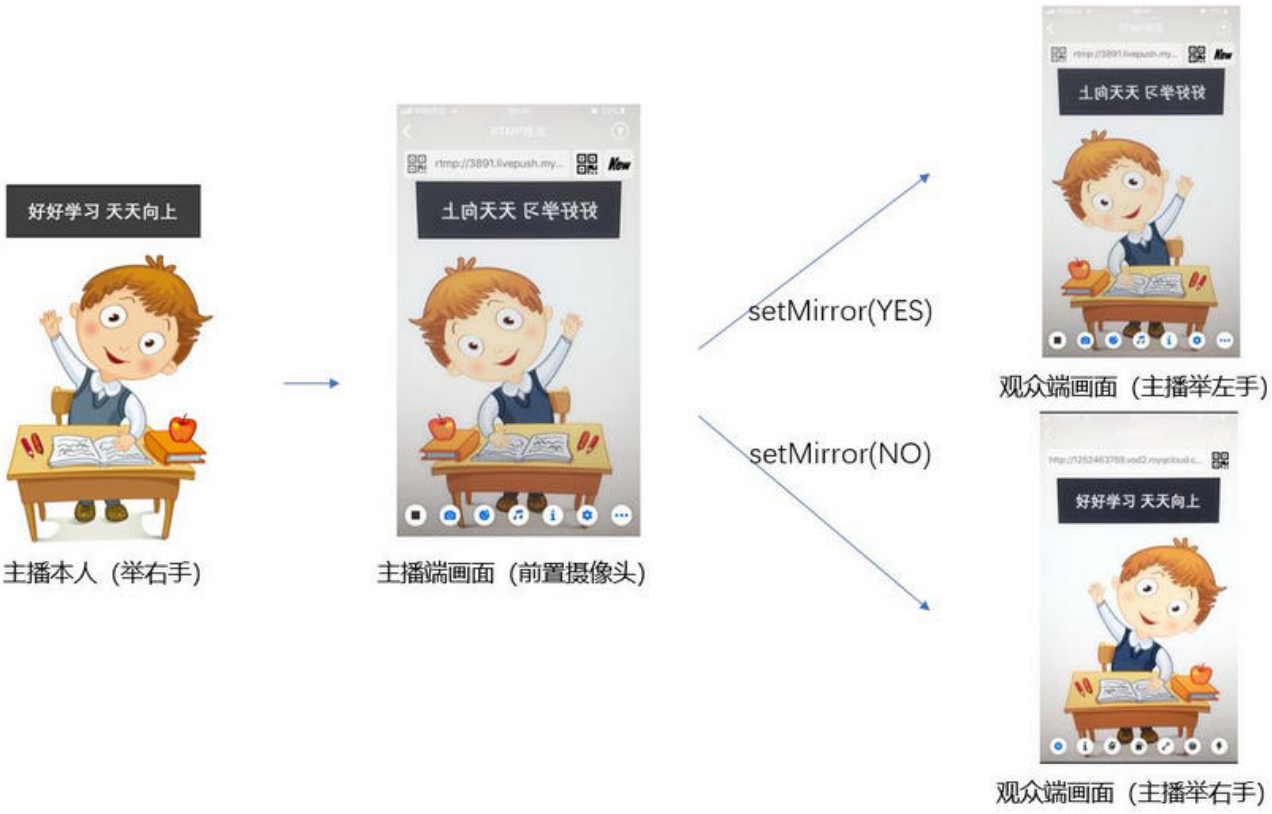
TXLivePush 提供了一组 API 用户控制摄像头的行为：

| 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。

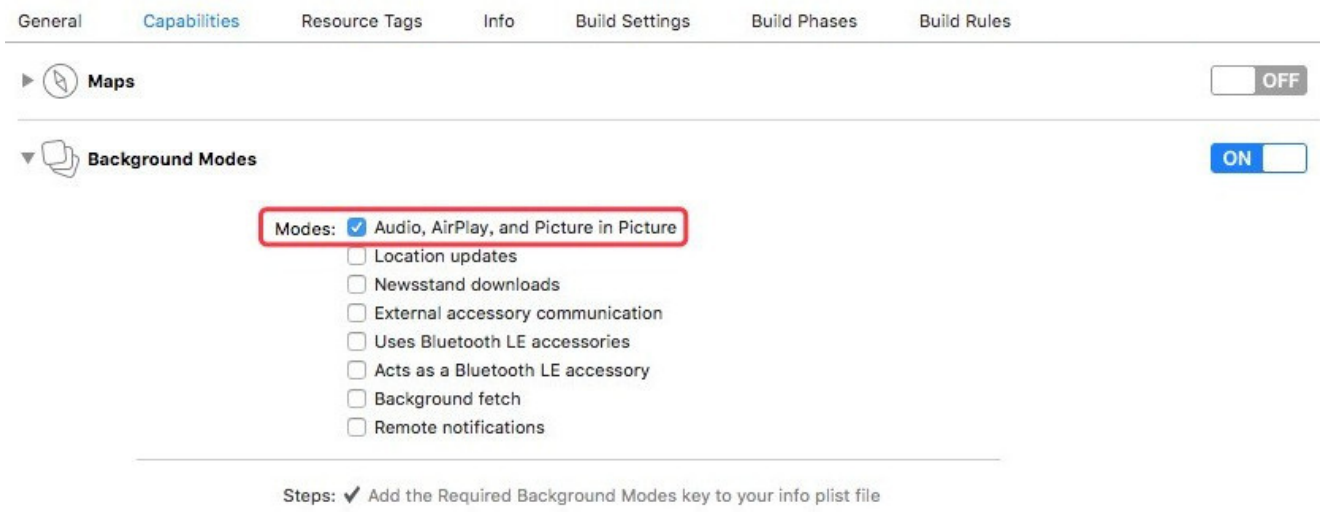
```
// 如果希望竖屏推流（HOME 键在下），这是 SDK 的默认行为
_config.homeOrientation = HOME_ORIENTATION_DOWN;
[_pusher setConfig:_config];
[_pusher setRenderRotation:0];

// 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:UIApplicationWillResignActiveNotifi
cation object:nil];
[center addObserver:self selector:@selector(didBecomeActive:) name:UIApplicationDidBecomeActiveNotifica
tion 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”。

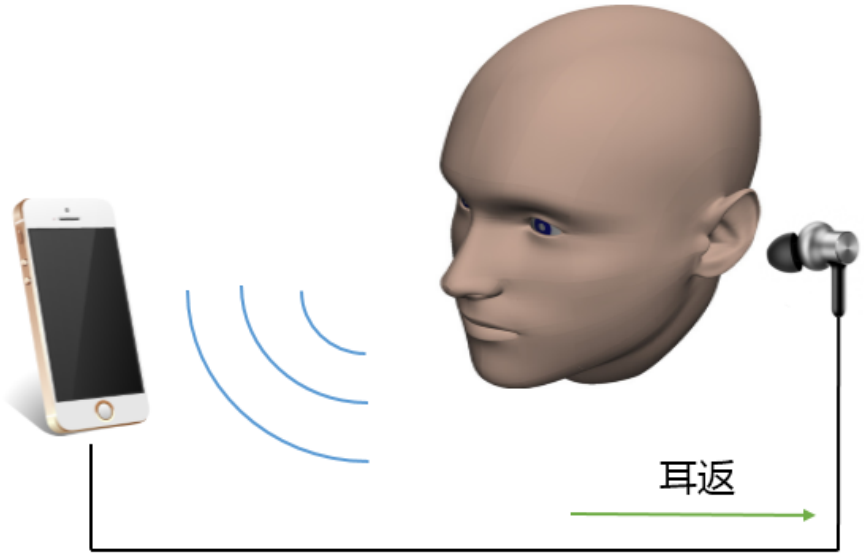

精简版 Demo 中的混音功能


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，水印位置位是由 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。
- 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。
- 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”完美同步。
2. 秀场直播：推流端将**歌词**下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。
3. 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈划线。

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：2021-05-24 16:46:50

## 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 | TCScreenAnchorActivity.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

单击 License 申请 获取测试用 License，您会获得两个字符串：其中一个字符串是 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);
}
}
```

### 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。
```

- **如何获取可用的推流 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。

| 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

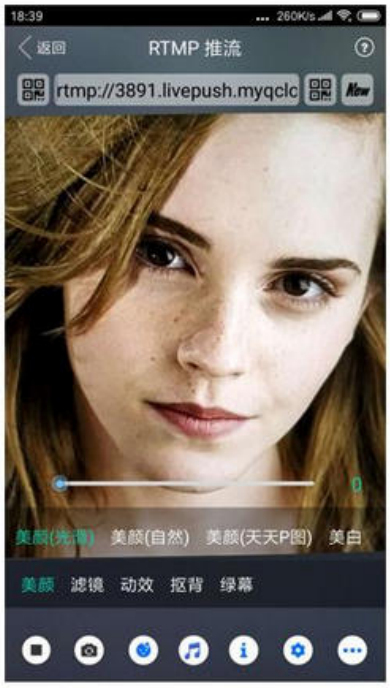
```
TXLivePusher # setVideoQuality ( quality, adjustBitrate, adjustResolution );
```

| 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

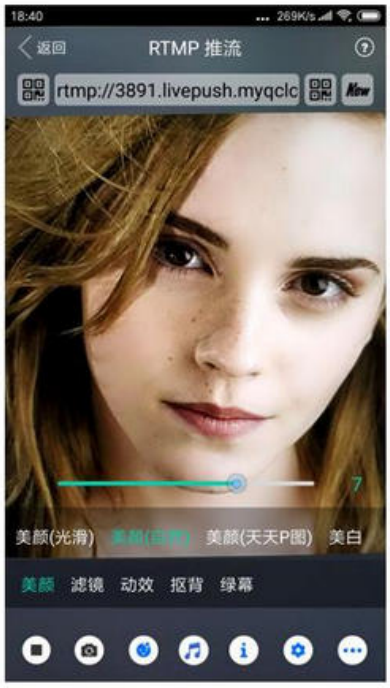
调用 TXLivePush 中的 setBeautyFilter 接口可以设置美颜效果，SDK 中提供了三种磨皮算法（style），定义见 TXLiveConstants.java 文件：

| 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

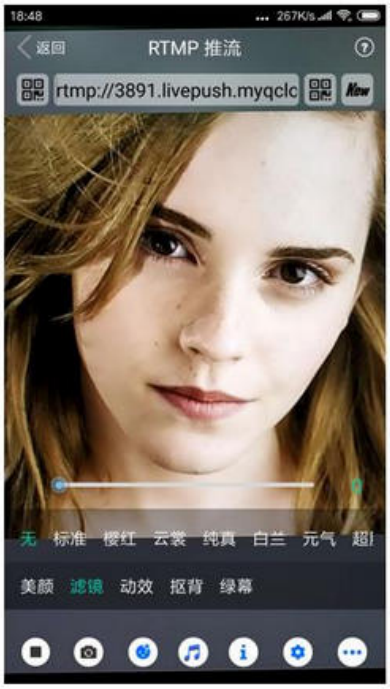
```
//style 美颜算法： 0：光滑 1：自然 2：朦胧
//beautyLevel 磨皮等级： 取值为 0-9.取值为 0 时代表关闭美颜效果.默认值: 0,即关闭美颜效果.
//whiteningLevel 美白等级： 取值为 0-9.取值为 0 时代表关闭美白效果.默认值: 0,即关闭美白效果.
//ruddyLevel 红润等级： 取值为 0-9.取值为 0 时代表关闭美白效果.默认值: 0,即关闭美白效果.
//
public boolean setBeautyFilter(int style, int beautyLevel, int whiteningLevel, int ruddyLevel);
```

### 9. 色彩滤镜效果

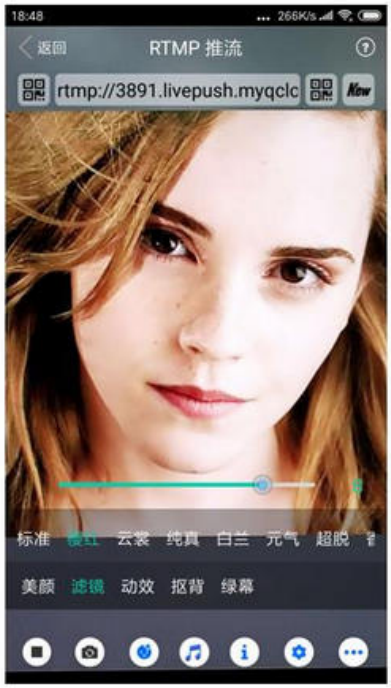
调用 TXLivePusher 中的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。

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。

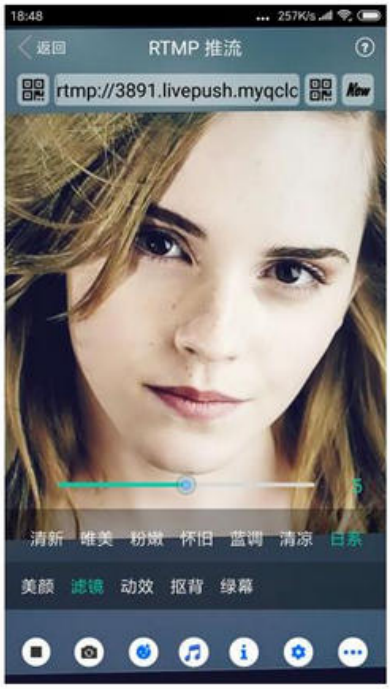
从手机 QQ 和 Now 直播的经验来看，单纯通过 `setBeautyStyle` 调整磨皮效果是不够的，只有将美颜效果和 `setFilter` 配合使用才能达到更加多遍的美颜效果。所以，我们的设计师团队提供了17种默认的色彩滤镜，并将其默认打包在了 Demo 中供您使用。




关闭滤镜




开启滤镜：樱红




开启滤镜：日系

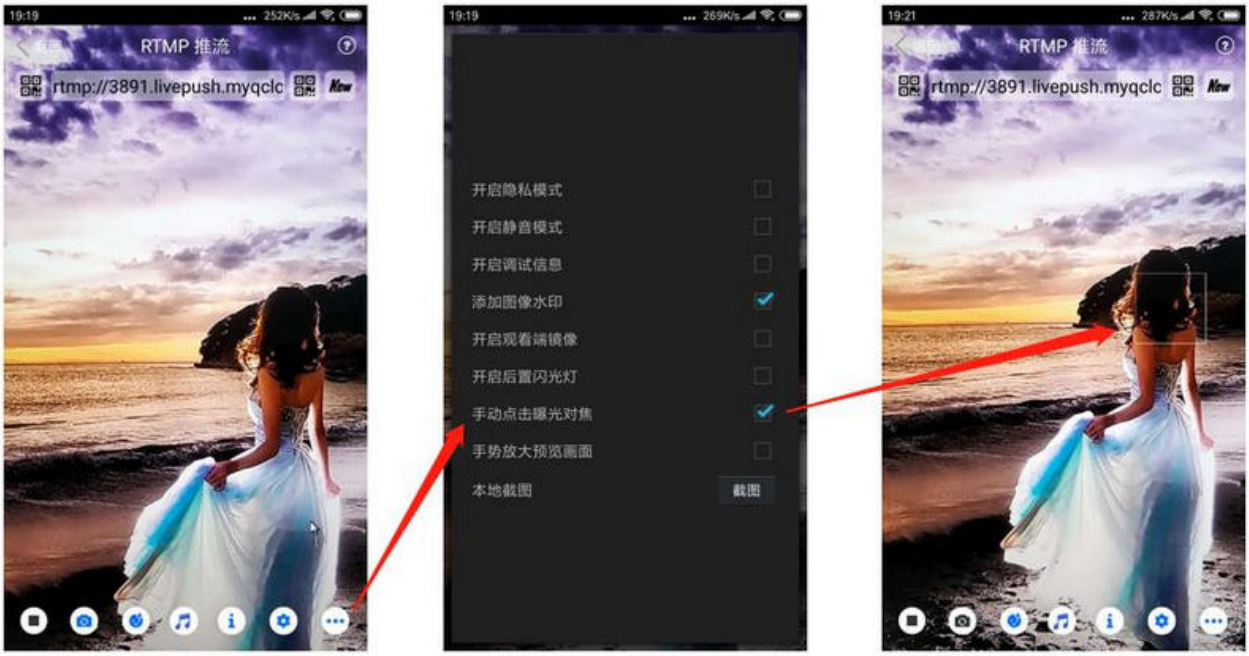
```
//选择期望的色彩滤镜文件
Bitmap filterBmp = decodeResource(getResources(), R.drawable.filter_biaozhun);
mLivePusher.setFilter(filterBmp);
mLivePusher.setSpecialRatio(0.5f);
```

### 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。



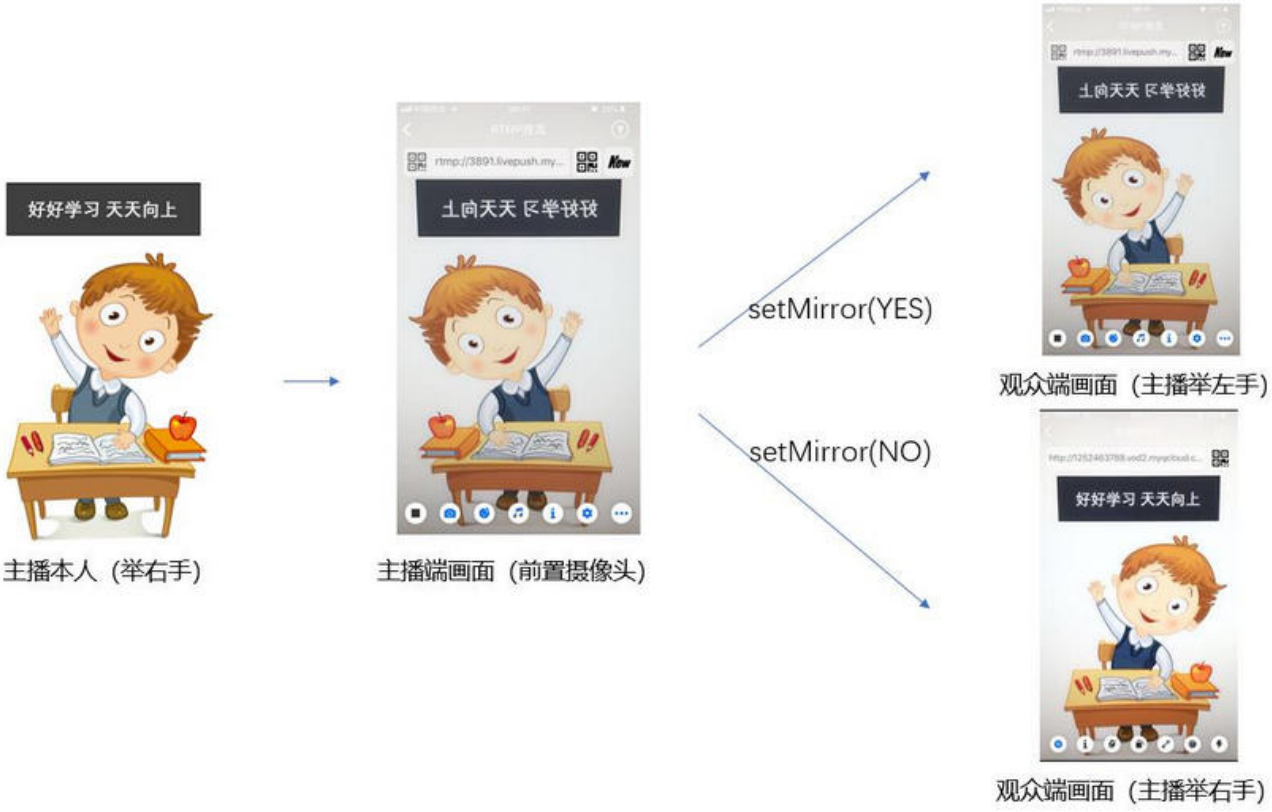

关闭手动曝光　　开启手动曝光　　点击屏幕触发

```
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。**

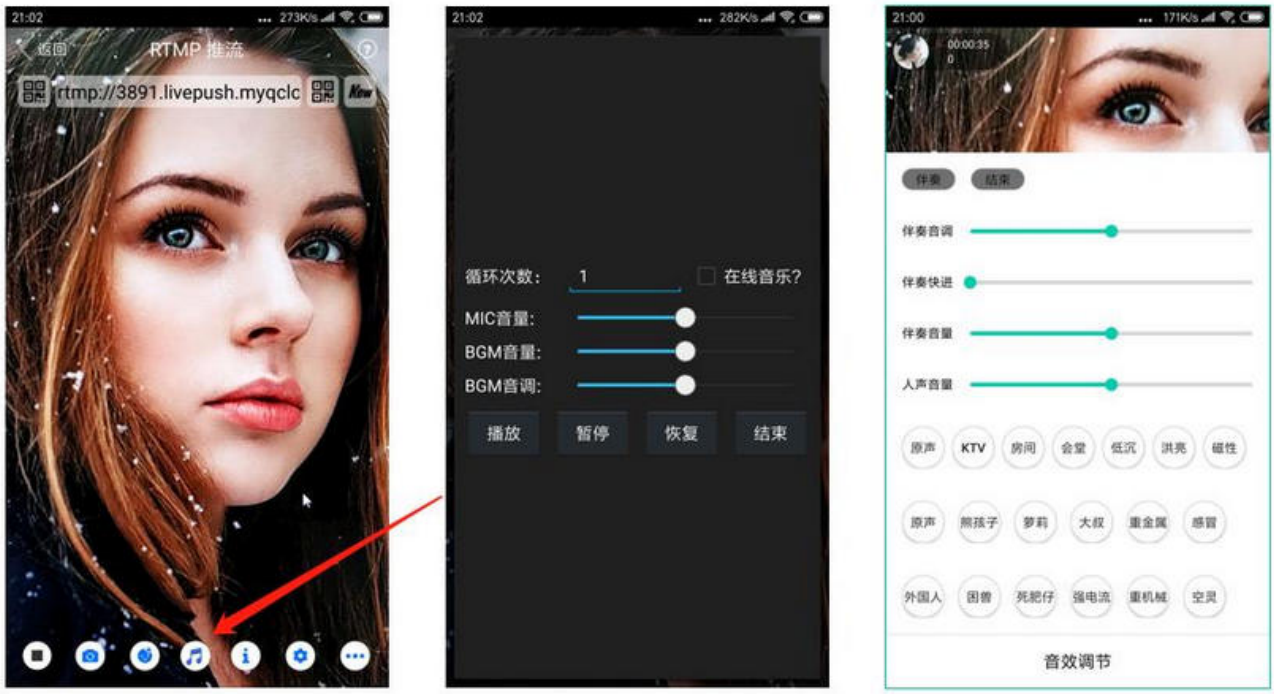
```
TXLivePushConfig mLivePushConfig = new TXLivePushConfig();
TXLivePusher mLivePusher = new TXLivePusher(this);

// 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，水印的位置位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编，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”完美同步。
2. 秀场直播：推流端将**歌词**下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。

3. 在线教育：推流端将**激光笔**和**涂鸦**操作下发到观众端，可以在播放端实时地划圈划线。

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 所播放的视频流中由 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：2021-07-16 16:45:27

## 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。

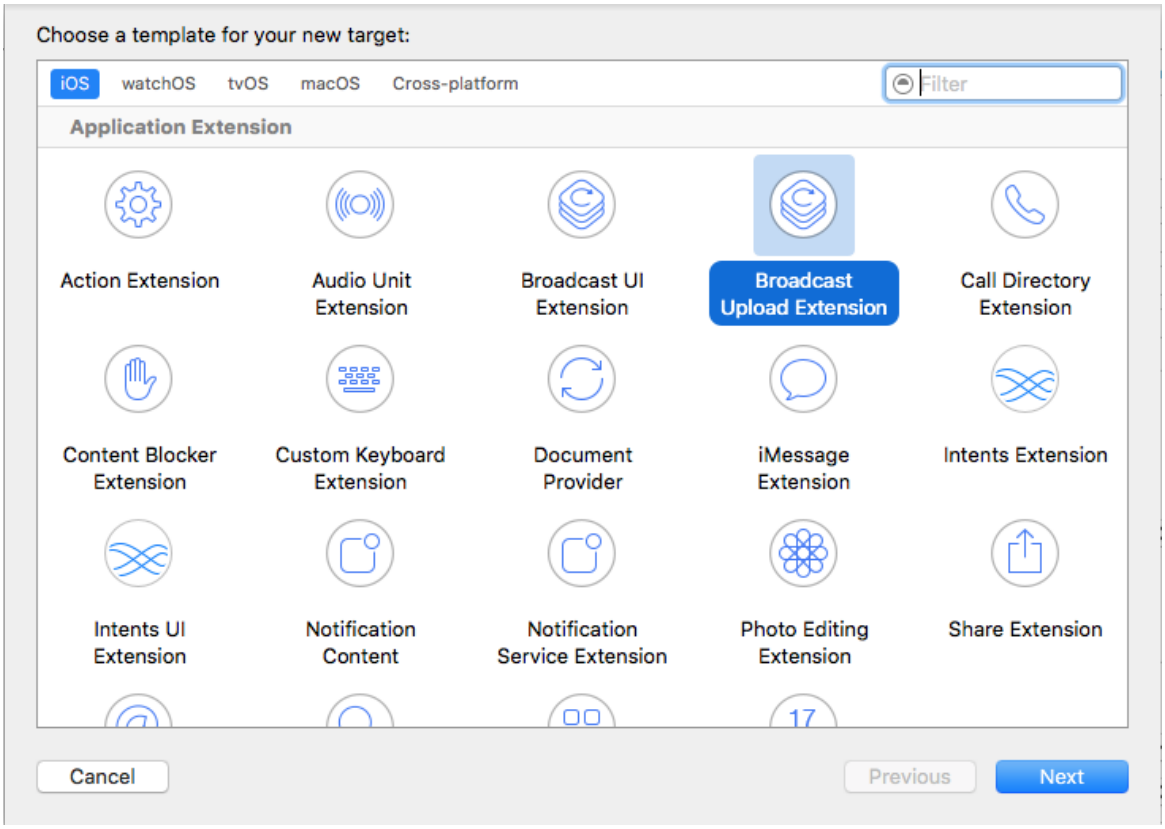
## 开发环境准备

### 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 实例有可能不只一个，因此对变量采用静态声明，确保录屏推流过程中使用的是同一个推流器。
- s_txLivePublisher 的 config 默认的配置为摄像头推流配置，因此需要额外配置为自定义采集视频和音频模式，视频开启 autoSampleBufferSize，SDK 会自动根据输入的分辨率设置编码器，您不需要关心推流的分辨率；如果您关闭此选项，那么代表您需要自

定义分辨率。

- 因为系统录制对不同机型屏幕所得到的分辨率不一致，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72
0P
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：
>
> 1. 一般手机上为9：16，而在 iPhoneX 上画面比例为1125：2436，因此此处使用屏幕比例进行计算分辨率。
> 2. 在 ReplayKit2 上采集的都是竖屏的分辨率，如果需要推送横屏分辨率，除了设置横屏分辨率外还需同时指定 homeOrientation 为横屏推流，否则会引起画面变形。

### 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:
// 发送来着 Mic 的音频数据
```

```
[s_txLivePublisher sendAudioSampleBuffer:sampleBuffer withType:sampleBufferType];
break;
```

SDK 支持同时发送两路数据，内部会对两路数据进行混音处理。

### 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(),kDarvinNotificationNamePushStart,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 屏幕录制在 iOS 11 新推出功能，比较少官方文档，并且存在着一些问题每个版本的系统都在不断修复完善中。以下是一些使用中的常见现象或问题：

**1. 系统有声音在播放但观众端无法听到声音？**

系统在做屏幕音频采集时，在从 home 界面切到有声音播放的 App 时才会采集声音，从有声音播放的 App 切换到无声音播放的 App 时，即

使原 App 还在播放声音系统也不会进行音频采集，此时需要从 home 界面重新进入到有声音播放的 App 时系统才会重新采集。

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

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

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

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

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

# Android

最近更新时间：2021-07-26 14:02:36

## 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。

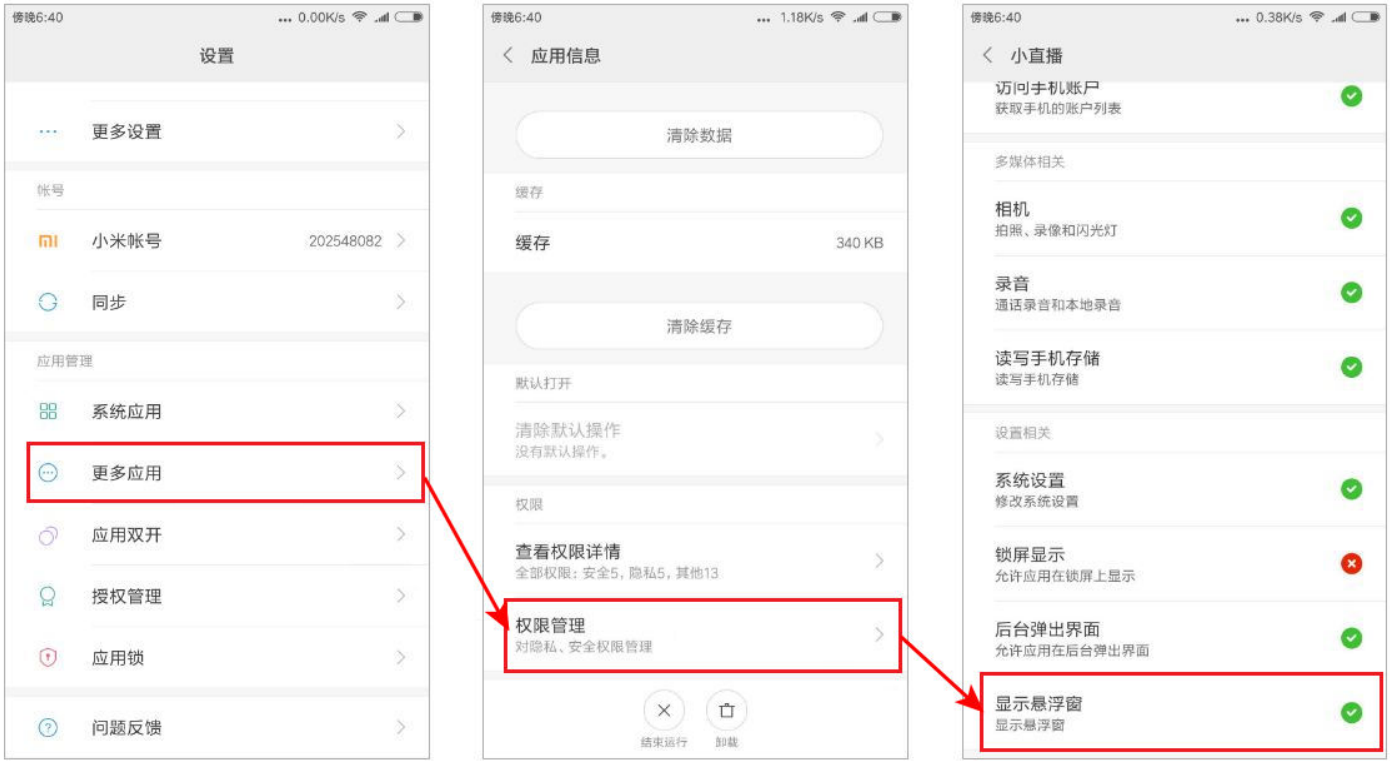
> 说明：
> 直播中叠加摄像头预览，即通过在手机上添加浮框，显示摄像头预览画面。录屏的时候会把浮框预览画面一并录制下来，达到叠加摄像头预览的效果。具体实现方法可参见 小直播源码。

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

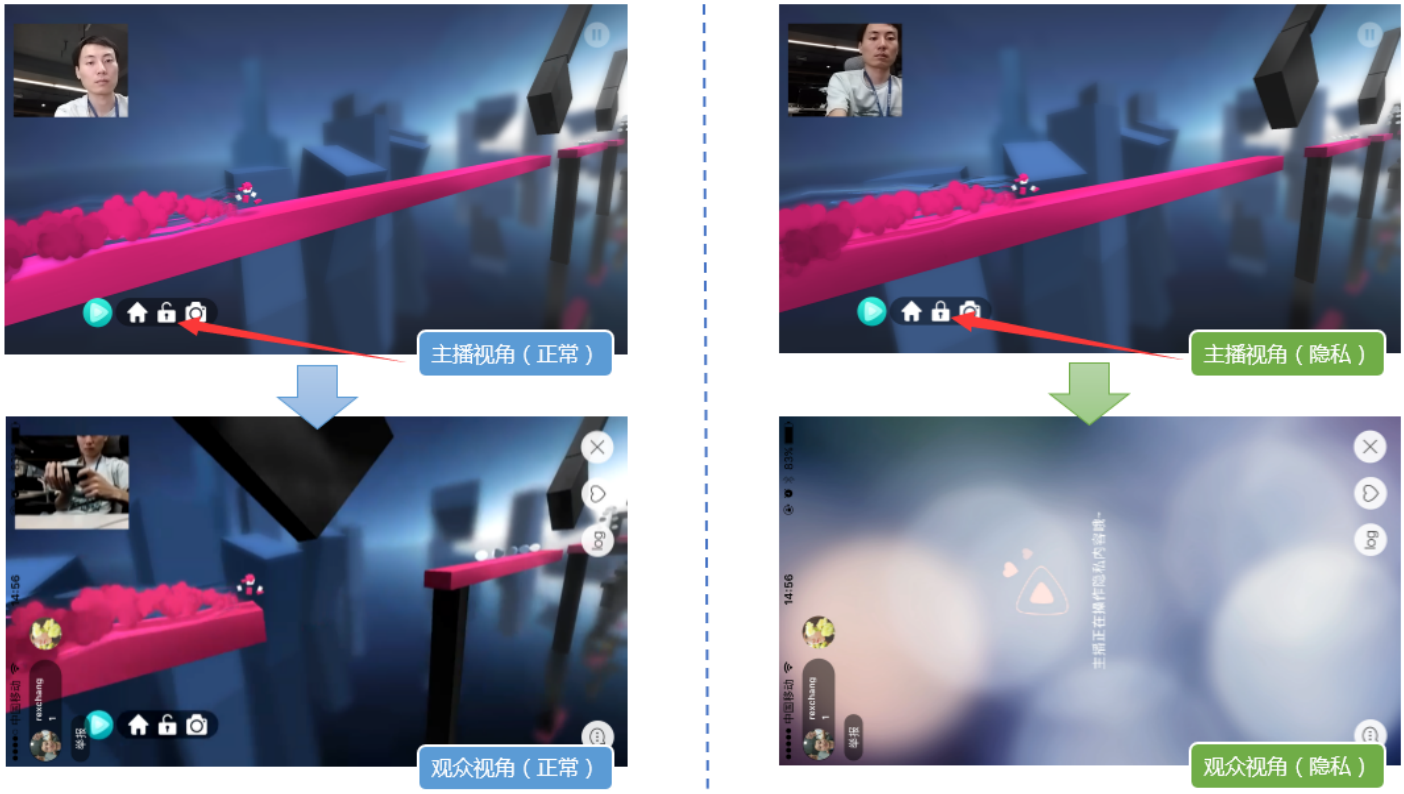
经过 步骤 1 和 步骤 2 的准备之后，用下面这段代码就可以启动推流了：

```
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，处于安全考虑，Android 系统会在开始录屏前弹出一个提示，旨在告诫用户：“有 App 要截取您屏幕上的所有内容”。

### 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：

- **4.1) 设置 pauseImg**
  在开始推流前，使用 TXLivePushConfig 的 setPauseImg 接口设置一张等待图片，例如“主播把画面切走一会儿...”。
- **4.2) 隐私模式开关**
  在用于工具条的悬浮窗口上增加一个用于开关隐私模式的按钮，打开隐私模式的响应逻辑为对 TXLivePusher##pausePush 接口函数的调用，关闭隐私模式的响应逻辑为对 TXLivePusher##resumePush 接口函数的调用。

```
public void triggerPrivateMode() {
if (mInPrivacy) {
Toast.makeText(getApplicationContext(), “隐私模式已开启”, Toast.LENGTH_SHORT).show();
mTVPrivateMode.setText(getString(R.string.private_mode_off));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOn,null,null,null);
mPrivateBtn.setImageResource(R.mipmap.lock_off);
mTXLivePusher.pausePusher();
```

```
} else {
Toast.makeText(getApplicationContext(), “隐私模式已关闭”, Toast.LENGTH_SHORT).show();
mTXLivePusher.resumePusher();
mPrivateBtn.setImageResource(R.mipmap.lock_on);
mTVPrivateMode.setText(getString(R.string.private_mode_on));
mTVPrivateMode.setCompoundDrawables(mDrawableLockOff,null,null,null);
}
mInPrivacy = !mInPrivacy;
}
```

### 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 |

| 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） |
| --- | --- | --- | --- | --- |
| 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，于此同时，您也需要指定声音采样率等和声道数等关键信息。

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

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

### 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：2021-07-12 10:19:08

## 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： 创建 Player

视频云 SDK 中的 TXLivePlayer 模块负责实现直播播放功能。

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

### step 2：渲染 View

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

```
//用 setupVideoWidget 给播放器绑定决定渲染区域的view，其首个参数 frame 在 1.5.2 版本后已经被废弃
[_txLivePlayer setupVideoWidget:CGRectMake(0, 0, 0, 0) 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 3：启动播放

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

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

**关于 HLS(m3u8)**

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

### step 4：画面调整

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

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

- **setRenderRotation：画面旋转**

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


最长边填充


完全填充


横屏模式

### step 5：暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这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 6：结束播放

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

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

### step 7：消息接收

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

- 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”完美同步。
- 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）**

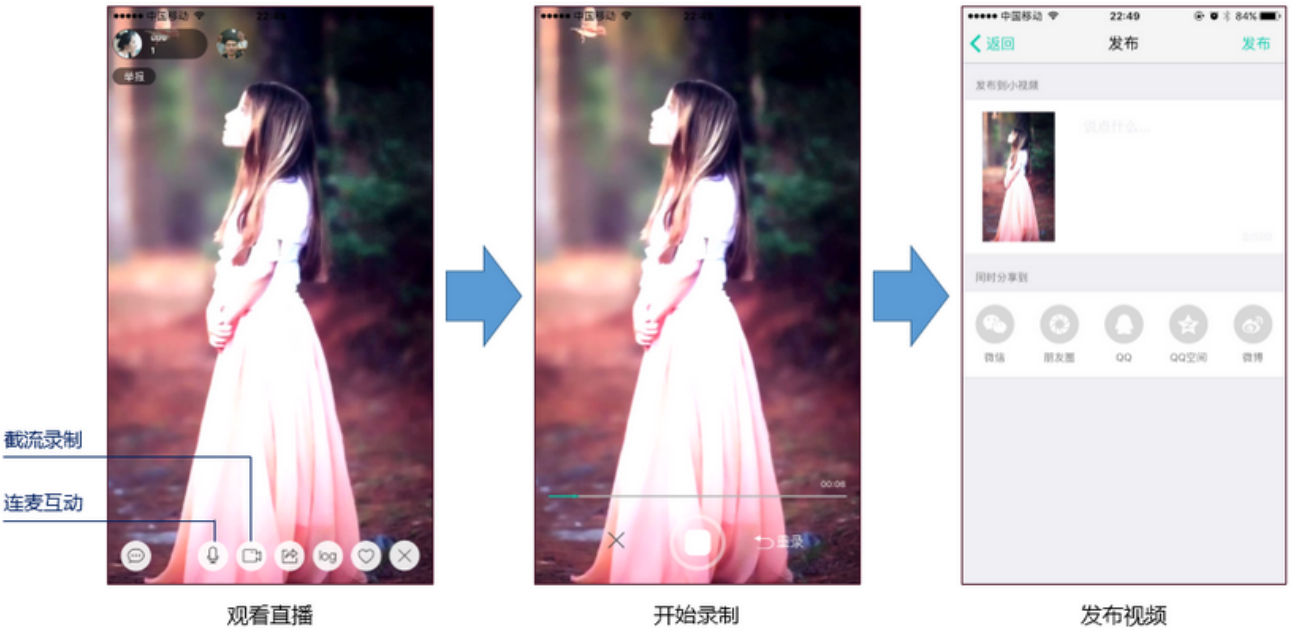
```
-(void) onPlayEvent:(int)EvtID withParam:(NSDictionary *)param {
[self asyncRun:^{
if (EvtID == PLAY_EVT_GET_MESSAGE) {
dispatch_async(dispatch_get_main_queue(), ^{
if ([_delegate respondsToSelector:@selector(onPlayerMessage:)]) {
[_delegate onPlayerMessage:param[@"EVT_GET_MSG"]];
}
});
}
}];
}
```

### step 8： 屏幕截图

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


### step 9： 截流录制

**截流录制是直播播放场景下的一种扩展功能：观众在观看直播时，可以通过单击录制按钮把一段直播的内容录制下来，并通过视频分发平台（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，具体使用方法可以参考 短视频-文件发布。

### step 10： 清晰度无缝切换

日常使用中，网络情况在不断发生变化。在网络较差的情况下，最好适度降低画质，以减少卡顿；反之，网速比较好，可以提高观看画质。
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 11： 直播回看

时移功能是腾讯云推出的特色能力，可以在直播过程中，随时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。

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

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

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

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* _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 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，具体价格请参考 **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_END | 2006 | 播放结束，HTTP-FLV 的直播流是不抛这个事件的 |
| 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_VOD_PLAY_PREPARED | 2013 | 如果您在直播中收到此消息，可以忽略 |
| PLAY_EVT_VOD_LOADING_END | 2014 | 如果您在直播中收到此消息，可以忽略 |
| PLAY_EVT_STREAM_SWITCH_SUCC | 2015 | 直播流切换完成，请参考 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_EVT_PLAY_END | 2006 | 视频播放结束 |
| PLAY_ERR_NET_DISCONNECT | -2301 | 网络断连，且经多次重连亦不能恢复，更多重试请自行重启播放 |

**如何判断直播已结束?**

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。所以 2006 和 -2301 都要监听，用来作为直播结束的判定事件。

#### 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：2021-07-12 10:18:18

## 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: 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 2: 创建 Player

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

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

### step 3: 启动播放

```
String flvUrl = "http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
mLivePlayer.startPlay(flvUrl, TXLivePlayer.PLAY_TYPE_LIVE_FLV); //推荐 FLV
```

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

**关于 HLS(m3u8)**

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

### step 4: 画面调整

- **view：大小和位置**
  如需修改画面的大小及位置，直接调整 step1 中添加的 video_view 控件的大小和位置即可。
- **setRenderMode：铺满or适应**

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

- **setRenderRotation：画面旋转**

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

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


最长边填充


完全填充


横屏模式

### step 5: 暂停播放

对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这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 6: 结束播放

结束播放时**记得销毁 view 控件**，尤其是在下次 startPlay 之前，否则会产生大量的内存泄露以及闪屏问题。
同时，在退出播放界面时，记得一定要调用渲染 View 的 `onDestroy()` 函数，否则可能会产生内存泄露和“Receiver not registered”报警。

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

stopPlay 的布尔型参数含义为——“是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。
如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

### step 7: 消息接收

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

- 冲顶大会：推流端将**题目**下发到观众端，可以做到“音-画-题”完美同步。
- 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) {
}
});
```

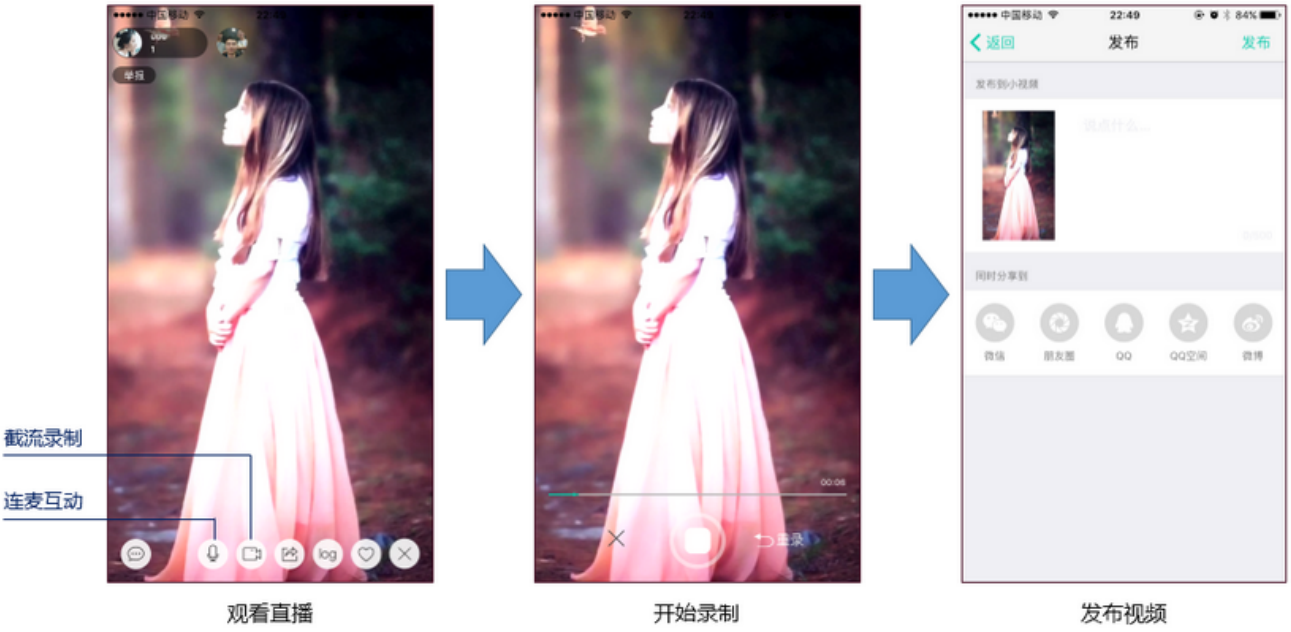
### step 8: 屏幕截图

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


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

### step 9: 截流录制

**截流录制是直播播放场景下的一种扩展功能：观众在观看直播时，可以通过单击录制按钮把一段直播的内容录制下来，并通过视频分发平台（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，具体使用方法可以参考 视频上传（Android）。

### step 10: 清晰度无缝切换

日常使用中，网络情况在不断发生变化。在网络较差的情况下，最好适度降低画质，以减少卡顿；反之，网速比较好，可以提高观看画质。
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_62fe94d692ab11e791eae435c87f075e_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 11: 直播回看

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

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

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

```
mLivePlayer.seek(600); // 从第10分钟开始播放
```

> 说明：
> time 为视频流时间点，当前取值为600，单位为秒。

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，具体价格请参考 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，说明错用成了 TXVodPlayer |
| PLAY_EVT_PLAY_END | 2006 | 播放结束，HTTP-FLV 的直播流是不抛这个事件的 |
| 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 |

| 事件ID | 数值 | 含义说明 |
| --- | --- | --- |
| 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，说明错用成了 TXVodPlayer |
| PLAY_EVT_CHANGE_ROTATION | 2011 | 如果您在直播中收到此消息，说明错用成了 TXVodPlayer |
| 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_VOD_PLAY_PREPARED | 2013 | 如果您在直播中收到此消息，说明错用成了 TXVodPlayer |
| PLAY_EVT_VOD_LOADING_END | 2014 | 如果您在直播中收到此消息，说明错用成了 TXVodPlayer |
| PLAY_EVT_STREAM_SWITCH_SUCC | 2015 | 直播流切换完成，请参考 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_EVT_PLAY_END | 2006 | 视频播放结束 |
| PLAY_ERR_NET_DISCONNECT | -2301 | 网络断连，且经多次重连亦不能恢复，更多重试请自行重启播放 |

**如何判断直播已结束?**

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。

所以 2006 和 -2301 都要监听，用来作为直播结束的判定事件。

### 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） |

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

## 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 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）
# iOS + Android

最近更新时间：2021-07-16 14:51:43

## 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

移动直播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。

- 购买连麦预付费套餐包
- 移动直播连麦计费说明

> 说明：
>
> - 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 | 当前用户在您的帐号系统中的 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 |
| --- | --- | --- |

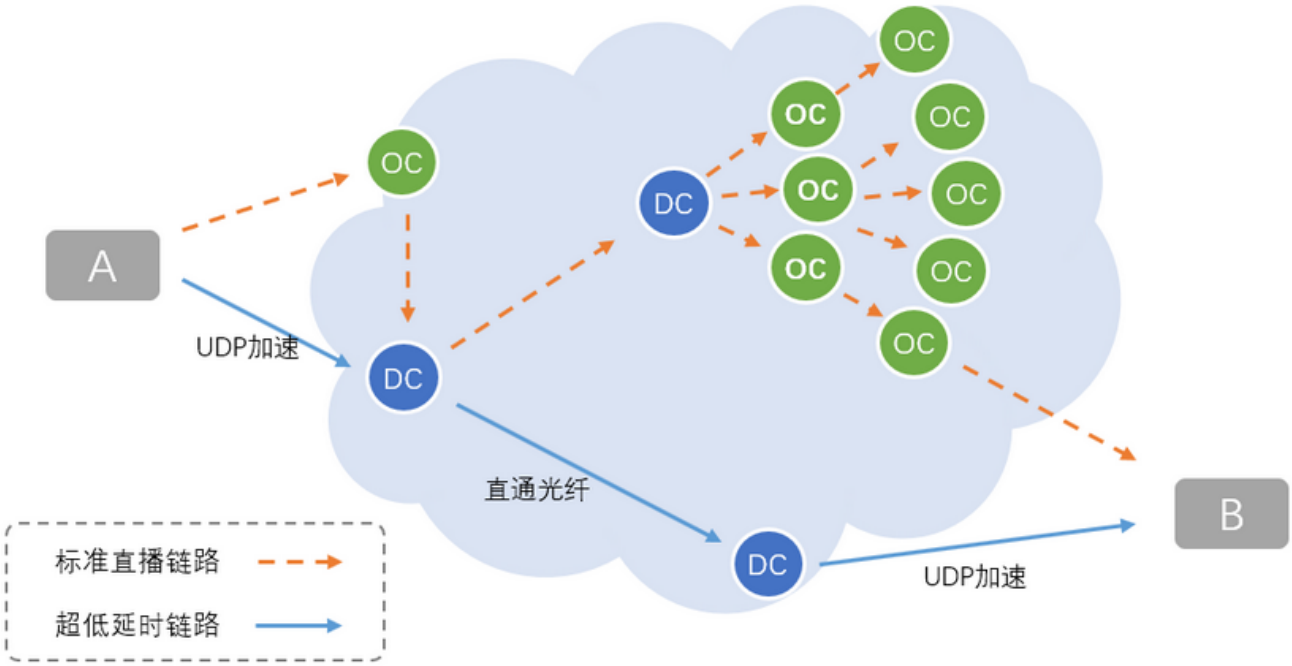
| 步骤 | 角色 | 详情 |
| --- | --- | --- |
| 第一步 | 主播 A | 主播 A 调用`requestRoomPK()`向主播 B 发起连麦请求。 |
| 第二步 | 主播 B | 主播 B 会收到`MLVBLiveRoomDelegate#onRequestRoomPK(AnchorInfo)`回调通知。 |
| 第三步 | 主播 B | 主播 B 调用`responseRoomPK()`确定是否接受主播 A 的 PK 请求。如果采用服务器配对的 PK 方案，此处可以默认接受，不需要由主播 B 来决策。 |
| 第四步 | 主播 B | 主播 B 在接受主播 A 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

**移动直播是不是使用 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 |

| 通道 | 直播通道 | 连麦通道 |
| --- | --- | --- |
| 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：2021-07-12 10:05:15

## 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、移动直播 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。

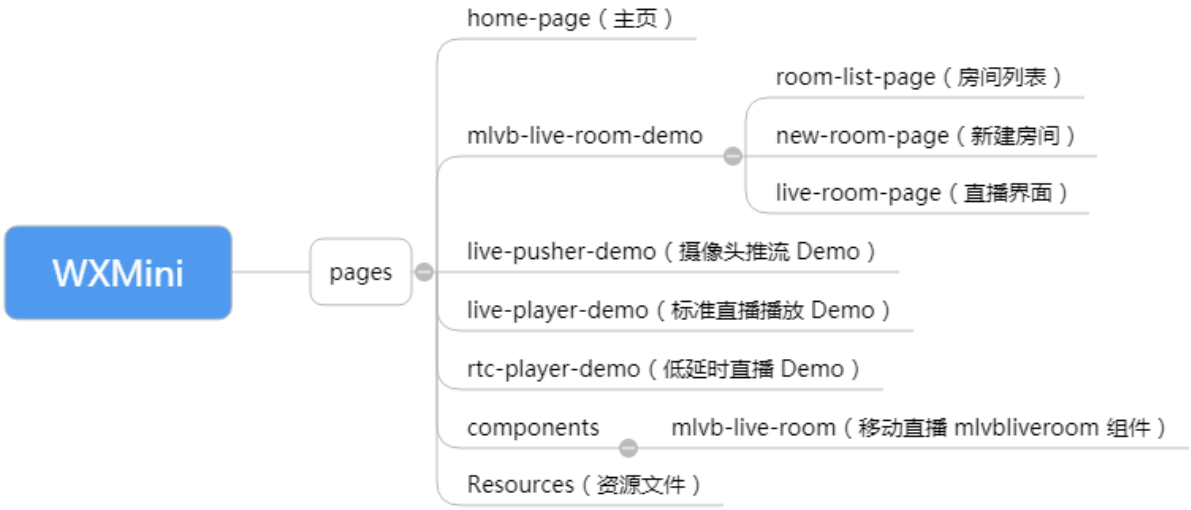
## 示例代码

- 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。

- 购买连麦预付费套餐包
- 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：
> 参考代码： 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：
> 参考代码： 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, aduience);
```

> 说明：
> 参考代码： 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“界面模版”规范进行定制：

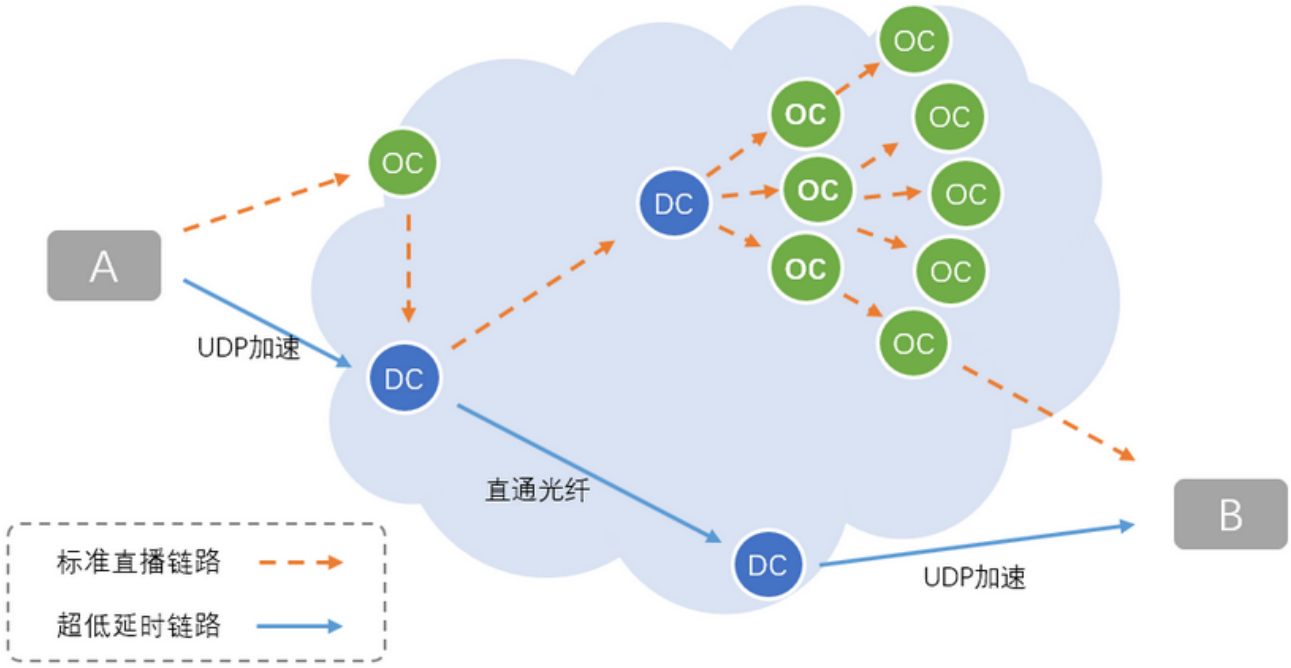
- 您可以直接修改 floattemplate.xml 模版文件和 floattemplate.wxss 样式文件。
- 您也可以增加新的模版 wxml 和样式文件 wxss，但新加的文件需要在 mlvbliveroomview.wxml 和 mlvbliveroomview.wxss 两个文件中增加相关的配置。

## 常见问题

### 1. <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 |