直播 SDK

无 UI 集成方案

腾讯云

文档目录

无 UI 集成方案
0. 跑通 Demo
iOS

最近更新时间：2024-12-11 16:20:42

说明：
- 为了帮助您更好地上手移动端直播 APP 的搭建工作，我们推出了腾讯云 MLVB-API-Example Demo，您可以根据下列操作指引快速跑通 Demo。
- 本文主要介绍如何快速运行腾讯云 MLVB-API-Example（iOS）。

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。

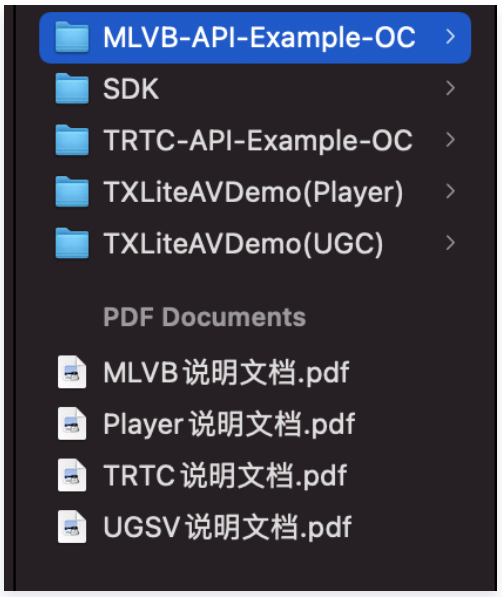
前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤一：下载 SDK 和 MLVB-API-Example 源码

1. 根据实际业务需求 下载 相应的压缩包，这里以 Professional 为例。
2. 下载完成后，解压。



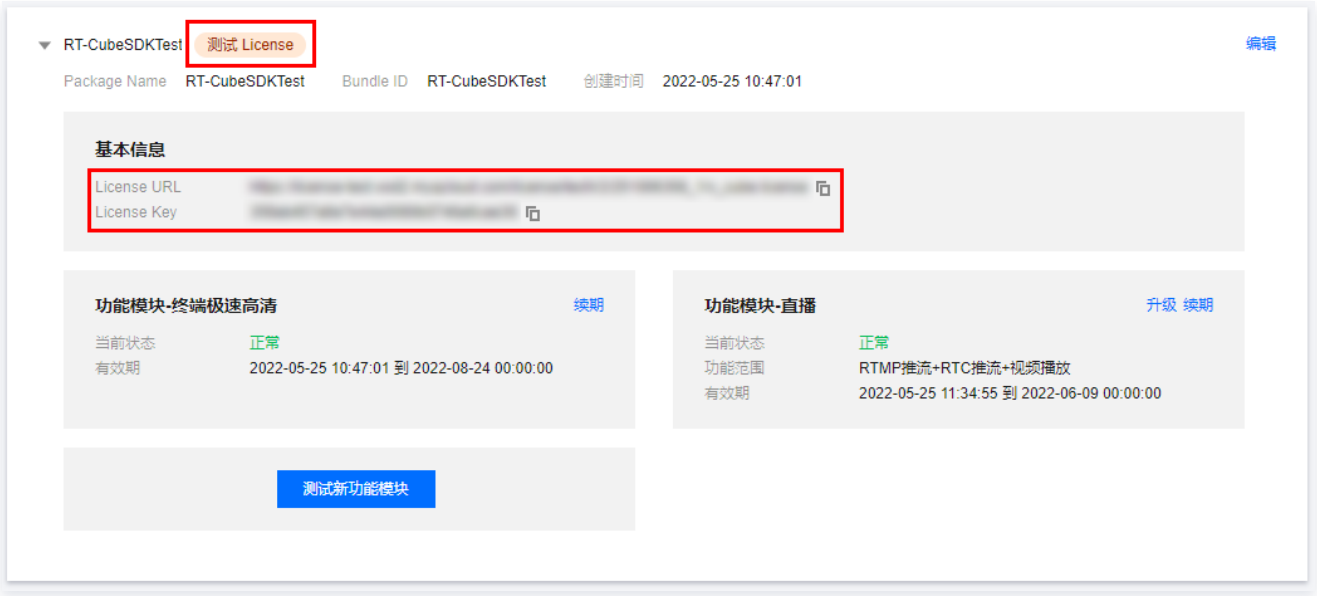
注意：
源码也可以从 Github 获得。

步骤二：配置 License

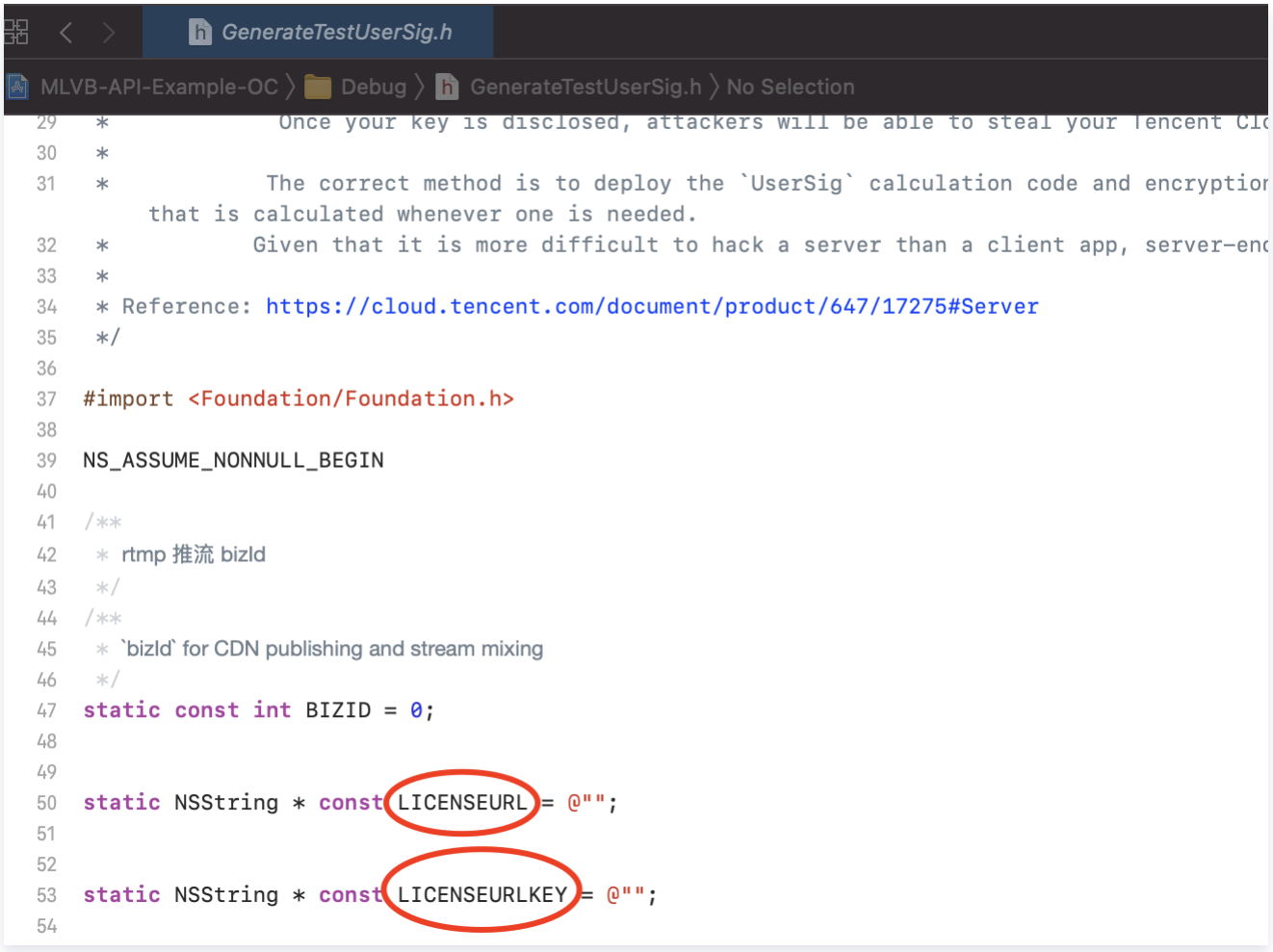
1. 登录 云直播控制台，在左侧菜单中选择 **直播 SDK > License管理**，单击 **新建测试 License**。

2. 根据实际需求填写 `App Name`、`Package Name` 和 `Bundle ID`，勾选功能模块 **直播**（直播推流 + 视频播放），单击**确定**。
   - **Package Name**：请在App目录下的 **build.gradle** 文件查看 **applicationId** 。
   - **Bundle ID**：请在 **xcode** 中查看项目的 **Bundle Identifier** 。

3. 测试版 License 成功创建后，页面会显示生成的 License 信息。**在 SDK 初始化配置时需要传入 Key 和 License URL 两个参数，请妥善保存以下信息。**



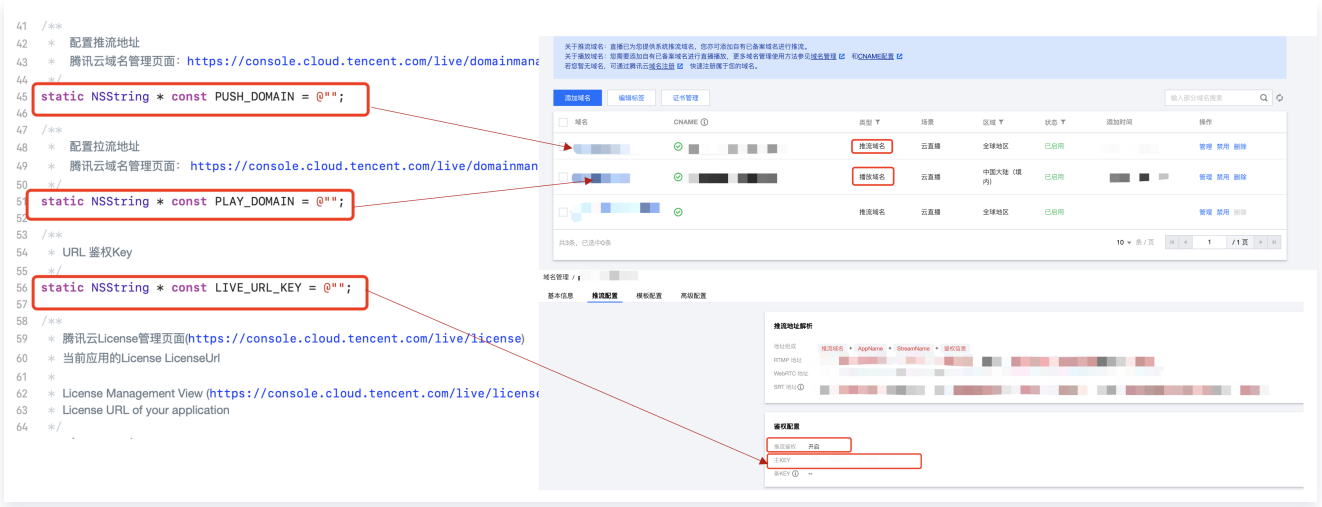
4. 打开 `LiteAVSDK_Professional_iOS_版本号/MLVB-API-Example-OC/Debug/GenerateTestUserSig.h` 文件。
设置 `GenerateTestUserSig.h` 文件中的相关参数：
   - LICENSEURL：默认为空，请设置为您的下载 Licence url。
   - LICENSEURLKEY：默认为空，请设置为您的下载 Licence key。



步骤三：配置推流/播放能力

1. 已在 域名注册 申请域名，并备案成功。
2. 已在 **云直播控制台** > 域名管理 中添加推流/播放域名，具体操作请参见 添加自有域名 。
3. 成功 配置域名 CNAME 。
4. 配置好推流/播放域名后，在推流/播放域名的 **基本信息** 页面可以获得 `CNAME` 信息。

5. 打开 `LiteAVSDK_Professional_iOS_版本号/MLVB-API-Example-OC/Debug/GenerateTestUserSig.h` 文件。
   设置 `GenerateTestUserSig.h` 文件中的相关参数：
   - **PUSH_DOMAIN**：请设置为您的 推流域名 。
   - **PLAY_DOMAIN**：请设置为您的 播放域名 。
   - **LIVE_URL_KEY**：非必需，用于生成 txSecret 等鉴权信息，具体计算请参见 推拉流 URL ，查询步骤参见 域名页面 > **管理** > **推流配置** > **鉴权配置**。

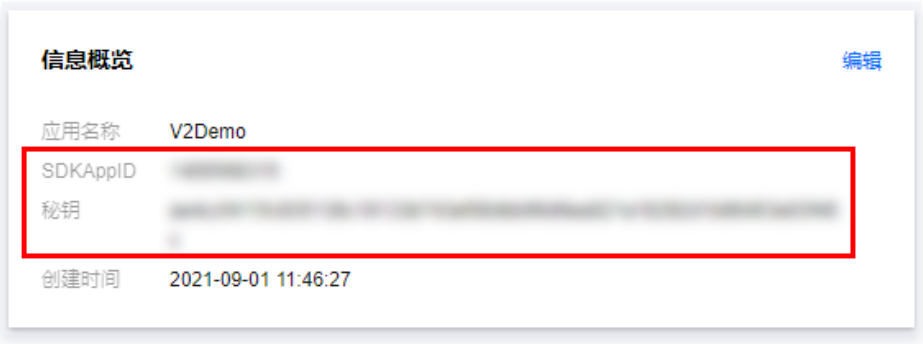


步骤四：配置连麦或 PK 能力/服务开通

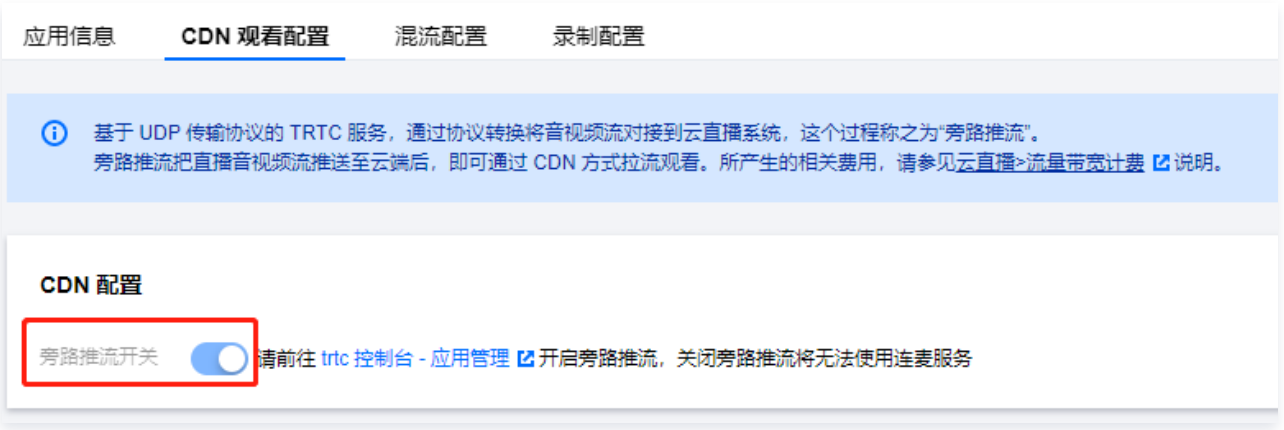
> ⚠ **注意：**

- 直播连麦现已全面升级，新版连麦方案 **基于实时音视频 TRTC 能力实现**，跑通 Demo 时会统计所有参与连麦/PK 的用户产生的视频时长和语音时长用量，**收取实时音视频 TRTC 音视频时长费用**。计费说明请参见 新版连麦方案计费 。
- **测试连麦及直播 CDN 观看功能时请注意测试时长，避免因为长时间测试消耗引起账号欠费**。若您是实时音视频 TRTC 新账号，首次可前往 试用中心 免费领取10000分钟音视频时长用于体验测试连麦功能。

1. 登录**云直播控制台**，选择 **连麦管理** > **连麦应用** 。
2. 单击 **新建连麦应用** ，输入应用名称，例如 `V2Demo` ，单击 **确定** 。
3. 创建成功后，单击应用列表中 **应用名称** 为 `V2Demo` 这行右侧的 **管理** ，查看应用对应的 `SDKAppID` 和 **秘钥** 信息。



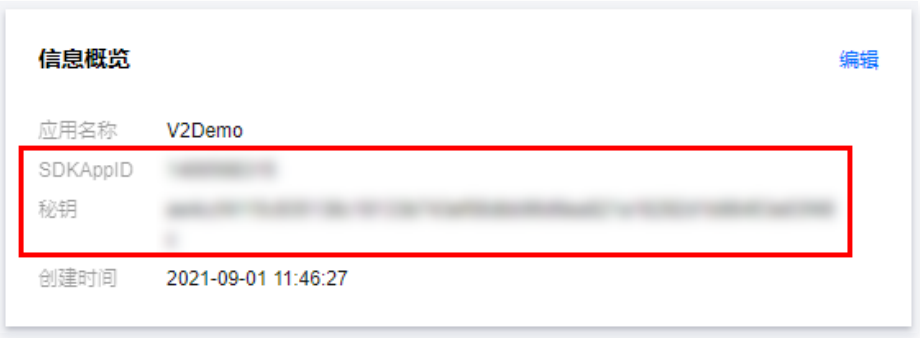
4. 若您的播放端需要进行 CDN 播放，则需要在 **连麦管理** > 连麦应用 中选择 `V2Demo` 行右侧的 **管理**，选择 **CDN 观看配置** 页，开启 **旁路推流** 功能。



> **说明**
> 旁路推流的方式默认选择 `指定流旁路` 即可，对于 V2TXLivePusher 两种方式没有区别。

配置推流参数

1. 找到并打开 `LiteAVSDK_Professional_iOS_版本号/MLVB-API-Example-OC/Debug/GenerateTestUserSig.h` 文件。
2. 根据上面 服务开通 设置 GenerateTestUserSig.h 文件中的相关参数：
   - SDKAppID：默认为 0 ，请设置为实际的 SDKAppID。
   - SECRETKEY：默认为空 ，请设置为实际的密钥信息。

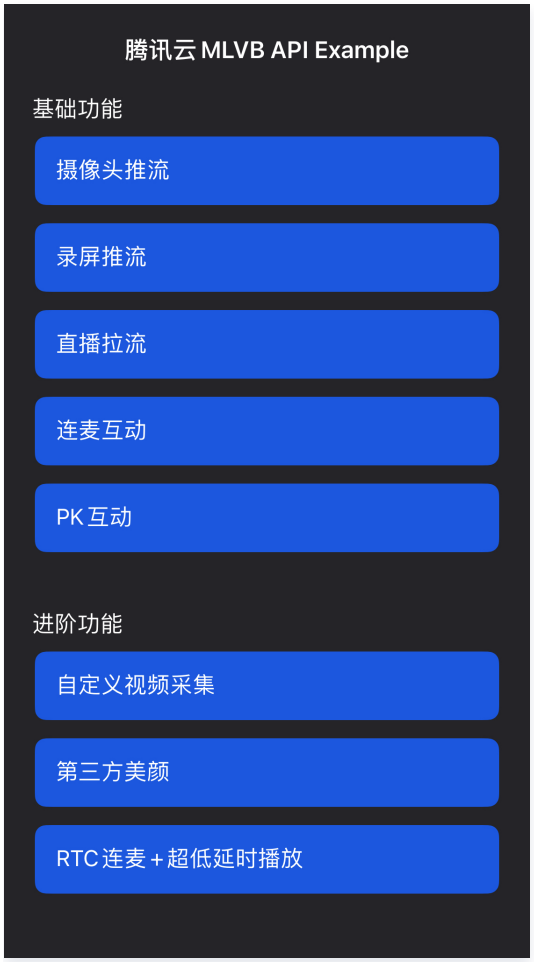


推流 URL 字段说明

具体的推拉流 URL 字符串，需要开发者按照对应的协议自行拼接，拼装方案请参见 推拉流 URL 。Demo 中已经拼接好，运行后即可播放。

步骤五：编译运行

使用 Xcode（9.0及以上的版本）打开源码工程 `MLVB-API-Example-OC`，**单击运行即可**。



Android

最近更新时间：2025-06-17 18:11:22

说明：

- 为了帮助您更好地上手移动端直播 APP 的搭建工作，我们推出了腾讯云 MLVB-API-Example Demo，您可以根据下列操作指引快速跑通 Demo。
- 本文主要介绍如何快速运行腾讯云 MLVB-API-Example（Android）。

环境要求

- 最低兼容 Android 4.1（SDK API Level 16），建议使用 Android 5.0（SDK API Level 21）及以上版本。
- Android Studio 3.5 及以上版本。
- App 要求 Android 4.1 及以上设备。

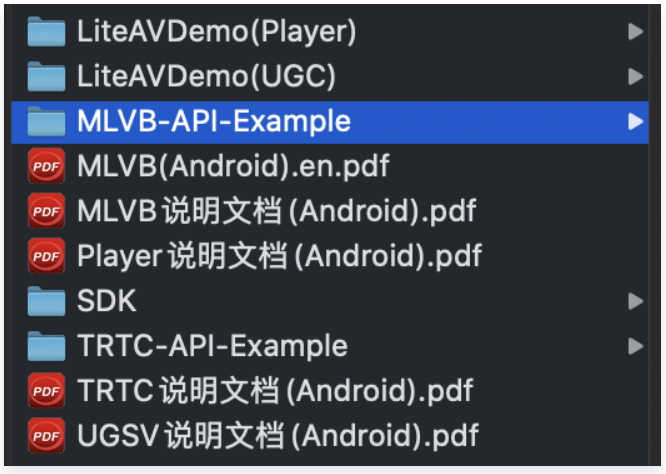
前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤一：下载 SDK 和 MLVB-API-Example 源码

1. 根据实际业务需求 下载 相应的压缩包，这里以 Professional 为例。
2. 下载完成后，解压。



注意：

源码也可以从 Github 获得。

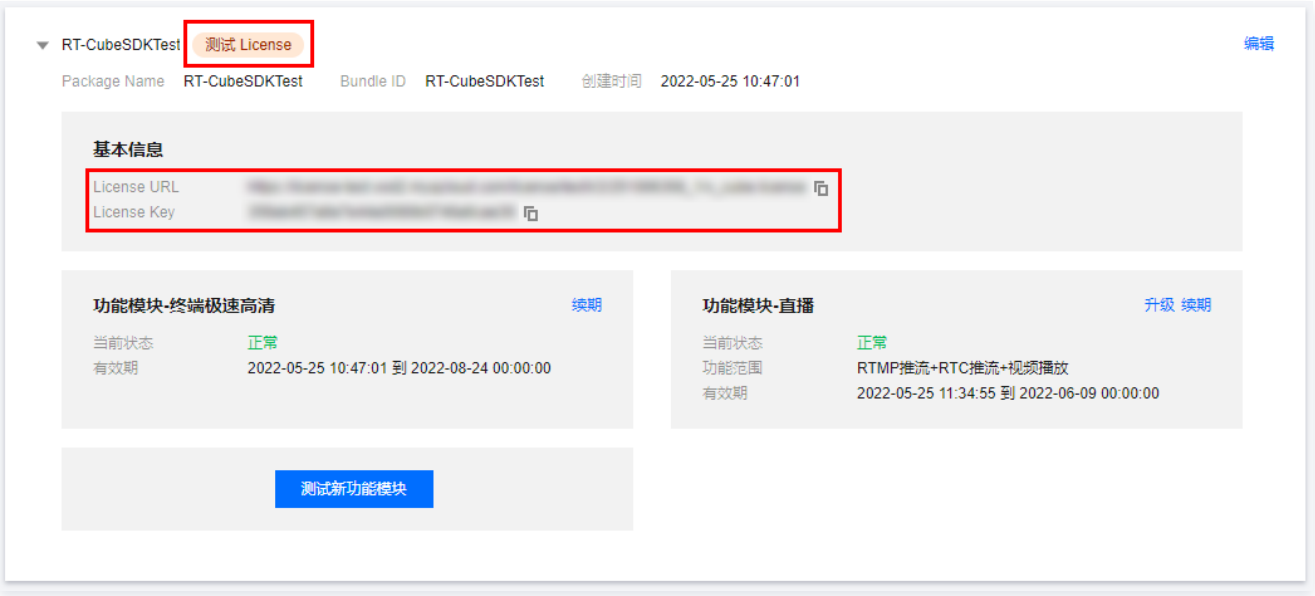
步骤二：配置 License

1. 登录 云直播控制台，在左侧菜单中选择**直播 SDK** > **License 管理**，单击**新建测试 License**。

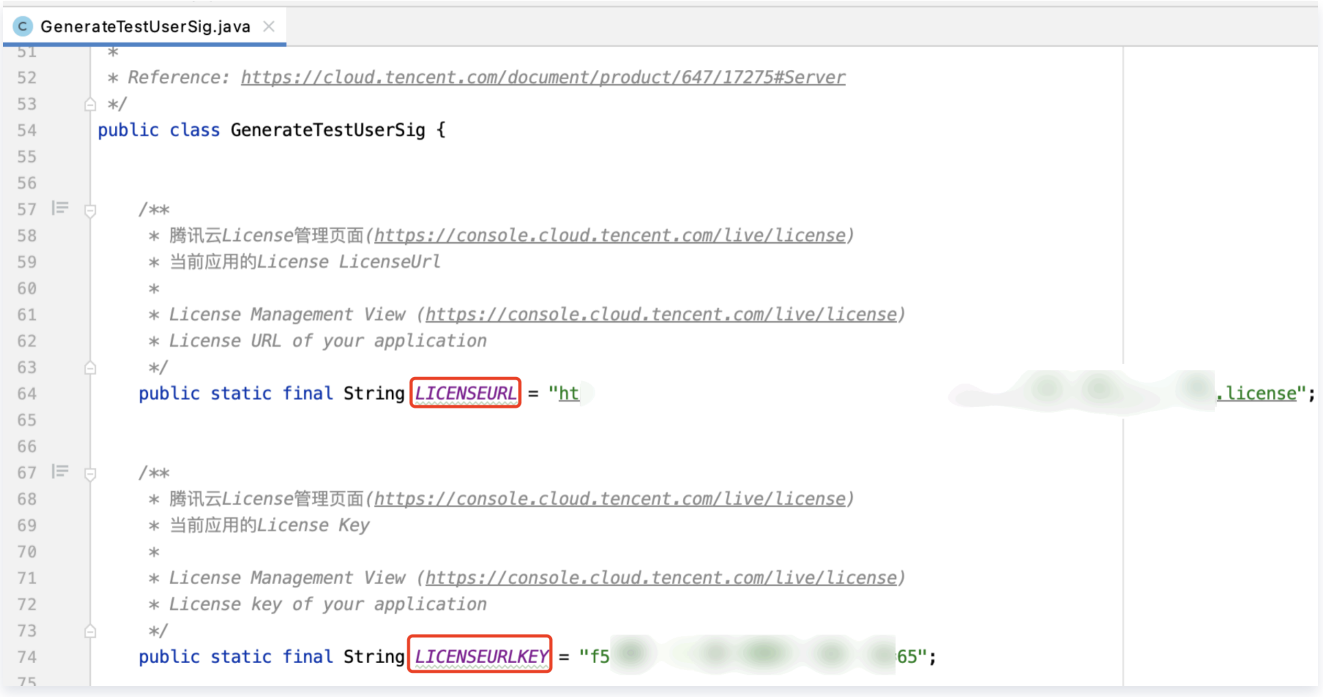
2. 根据实际需求填写 `App Name`、`Package Name` 和 `Bundle ID`，勾选功能模块 **直播**（直播推流 + 视频播放），单击**确定**。

- **Package Name**：请在 App 目录下的 **build.gradle** 文件查看 **applicationId** 。
- **Bundle ID**：请在 **xcode** 中查看项目的 **Bundle Identifier** 。

3. 测试版 License 成功创建后，页面会显示生成的 License 信息。**在 SDK 初始化配置时需要传入 Key 和 License URL 两个参数，请妥善保存以下信息。**



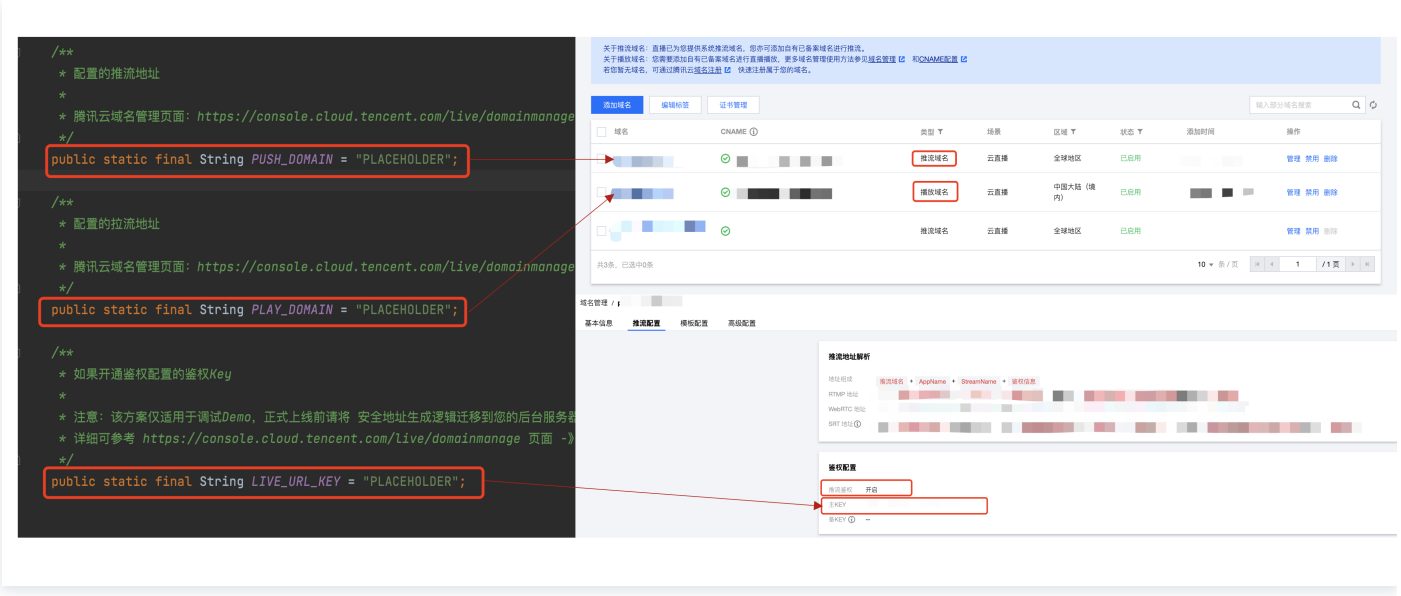
4. 打开 **LiteAVSDK_Professional_Android版本号/MLVB-API-Example/Debug/src/main/java/com/tencent/mlvb/debug/GenerateTestUserSig.java** 文件，设置 `GenerateTestUserSig.java` 文件中的相关参数：
   - LICENSEURL：默认为 PLACEHOLDER，请设置为您的下载 Licence url。
   - LICENSEURLKEY：默认为 PLACEHOLDER，请设置为您的下载 Licence key。



步骤三：配置推流/播放能力

1. 在 域名注册 申请域名，并备案成功。
2. 在**云直播控制台** > 域名管理 中添加推流/播放域名，具体操作请参见 添加自有域名 。
3. 成功 配置域名 CNAME 。
4. 配置好推流/播放域名后，在推流/播放域名的 **基本信息** 页面可以获得 `CNAME` 信息。

5. 打开 LiteAVSDK_Professional_Android版本号/MLVB-API-Example/Debug/src/main/java/com/tencent/mlvb/debug/GenerateTestUserSig.java 文件。设置 `GenerateTestUserSig.java` 文件中的相关参数：
   - **PUSH_DOMAIN**：请设置为您的 **推流域名**。
   - **PLAY_DOMAIN**：请设置为您的 **播放域名**。
   - **LIVE_URL_KEY**：非必需，用于生成 txSecret 等鉴权信息，具体计算请参见 **推拉流 URL**，查询步骤参见 **域名页面** > **管理** > **推流配置** > **鉴权配置**。

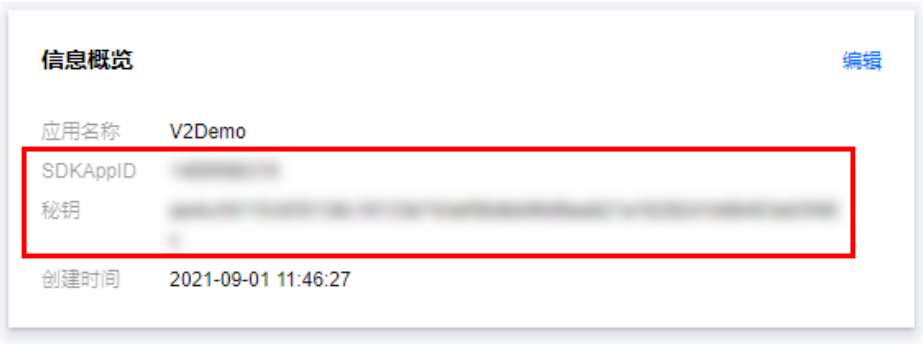


步骤四：配置连麦或 PK 能力/服务开通

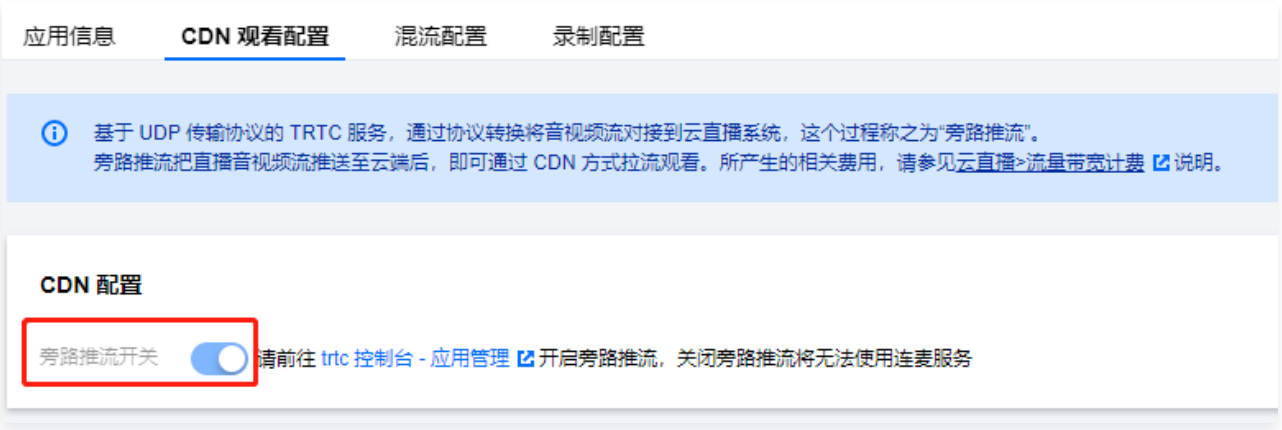
注意：

- 直播连麦现已全面升级，新版连麦方案 **基于实时音视频 TRTC 能力实现**，跑通 Demo 时会统计所有参与连麦/PK 的用户产生的视频时长和语音时长用量，**收取实时音视频 TRTC 音视频时长费用**。计费说明请参见 新版连麦方案计费 。
- **测试连麦及直播 CDN 观看功能时请注意测试时长，避免因为长时间测试消耗引起账号欠费**。若您是实时音视频 TRTC 新账号，首次可前往 试用中心 免费领取10000分钟音视频时长用于体验测试连麦功能。

1. 登录**云直播控制台**，选择**连麦管理** > **连麦应用** 。
2. 单击**新建连麦应用** ，输入应用名称，例如 `V2Demo` ，单击 **确定** 。
3. 创建成功后，单击应用列表中 **应用名称** 为 `V2Demo` 这行右侧的 **管理** ，查看应用对应的 `SDKAppID` 和密钥信息。



4. 若您的播放端需要进行 CDN 播放，则需要在**连麦管理** > 连麦应用 中选择 `V2Demo` 行右侧的**管理**，选择 **CDN 观看配置** 页，开启**旁路推流**功能。



> **说明**
> 旁路推流的方式默认选择 `指定流旁路` 即可，对于 V2TXLivePusher 两种方式没有区别。

配置推流参数

1. 找到并打开 **LiteAVSDK_Professional_Android版本号/MLVB-API-Example/Debug/src/main/java/com/tencent/mlvb/debug/GenerateTestUserSig.java** 文件。
2. 根据上面 服务开通 设置 GenerateTestUserSig.java 文件中的相关参数：
   - SDKAPPID：默认为 PLACEHOLDER，请设置为实际的 SDKAppID。
   - SECRETKEY：默认为 PLACEHOLDER，请设置为实际的密钥信息。

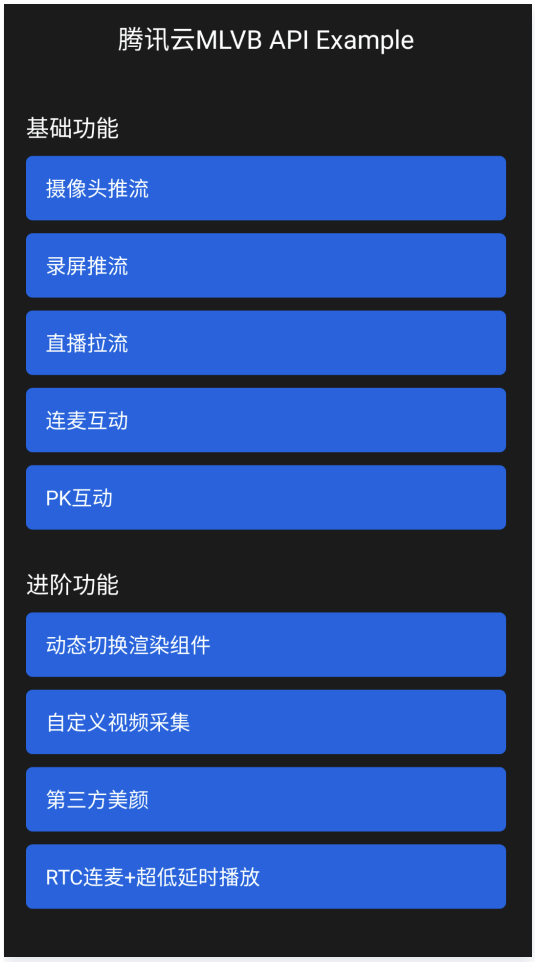
```
/**
 * Tencent Cloud `SDKAppID`. Set it to the `SDKAppID` of your account.
 *
 * You can view your `SDKAppID` after creating an application in the [TRTC console](https://console.cloud.tencent.com/rav).
 * `SDKAppID` uniquely identifies a Tencent Cloud account.
 */
public static final int SDKAPPID = 1      4;

/**
 * 签名过期时间，建议不要设置的过短
 * <p>
 * 时间单位：秒
 * 默认时间：7 x 24 x 60 x 60 = 604800 = 7 天
 */

/**
 * Signature validity period, which should not be set too short
 * <p>
 * Unit: second
 * Default value: 604800 (7 days)
 */
private static final int EXPIRETIME = 604800;


/**
 * 计算签名用的加密密钥，获取步骤如下：
 *
 * step1. 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav )，如果还没有应用就创建一个，
 * step2. 单击应用信息，并进一步找到“快速上手”部分。
 * step3. 点击“复制密钥”按钮，复制密钥，请将其拷贝并复制到如下的变量中
 *
 * 注意：该方案仅适用于调试Demo，正式上线前请将 UserSig 计算代码和密钥迁移到您的后台服务器上，以避免加密密钥泄露导致的流量盗用。
 * 文档：https://cloud.tencent.com/document/product/647/17275#Server
 */

/**
 * Follow the steps below to obtain the key required for UserSig calculation.
 *
 * Step 1. Log in to the [TRTC console](https://console.cloud.tencent.com/rav), and create an application if you don’t have one.
 * Step 2. Find your application, click “Application Info”, and click the “Quick Start” tab.
 * Step 3. Copy and paste the key to the code, as shown below.
 *
 * Note: this method is for testing only. Before commercial launch, please migrate the UserSig calculation code and key to your ba
 * Reference: https://cloud.tencent.com/document/product/647/17275#Server
 */
public static final String SECRETKEY = "3465                                  c8d082db20de2ffed36069ab72e";

/**
 * 计算 UserSig 签名
```

推流 URL 字段说明

具体的推拉流 URL 字符串，需要开发者按照对应的协议自行拼接，拼装方案请参见 推拉流 URL 。Demo 中已经拼接好，运行后即可播放。

步骤五：编译运行

使用 Android Studio（3.5及以上的版本）打开源码工程 `MLVB-API-Example`，单击 **运行** 即可。



1. SDK 集成
iOS

最近更新时间：2024-12-25 15:23:02

本文主要介绍如何快速地将腾讯云视立方 · 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 全功能版 SDK 为例：

开发环境要求

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

集成 LiteAVSDK

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

CocoaPods

1. **安装 CocoaPods**
   在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：

```
sudo gem install cocoapods
```

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

```
pod init
```

3. **编辑 Podfile 文件**
   编辑 Podfile 文件，有两种设置方式：
   - 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
```

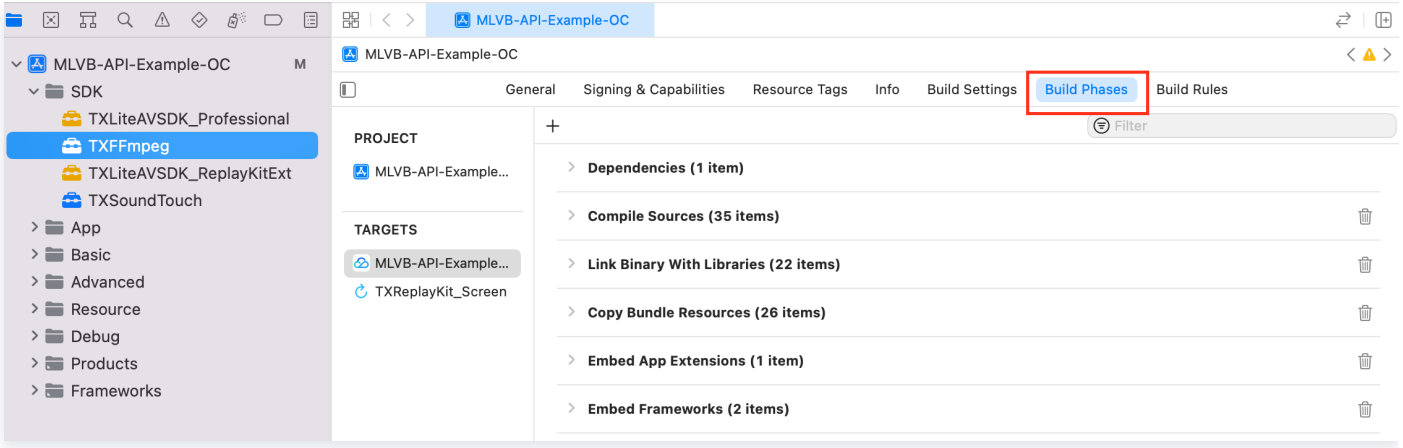
   - 或使用以下命令更新本地库版本：

```
pod update
```

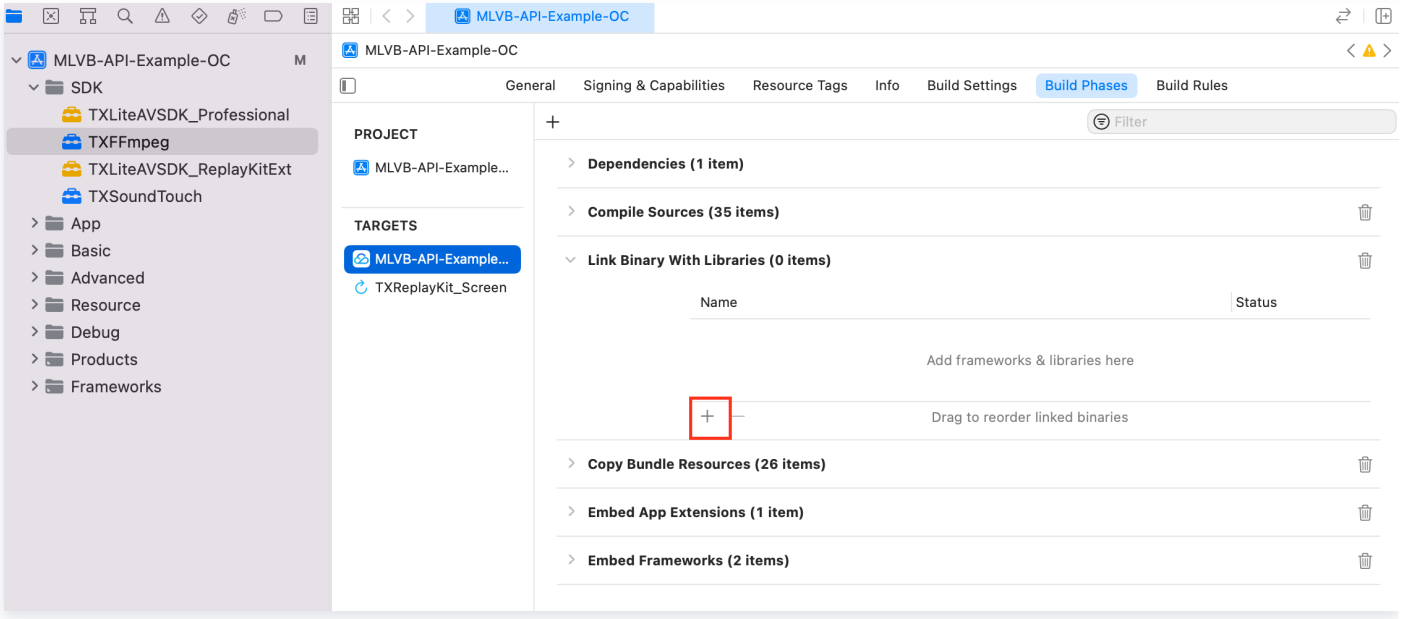
   pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

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。



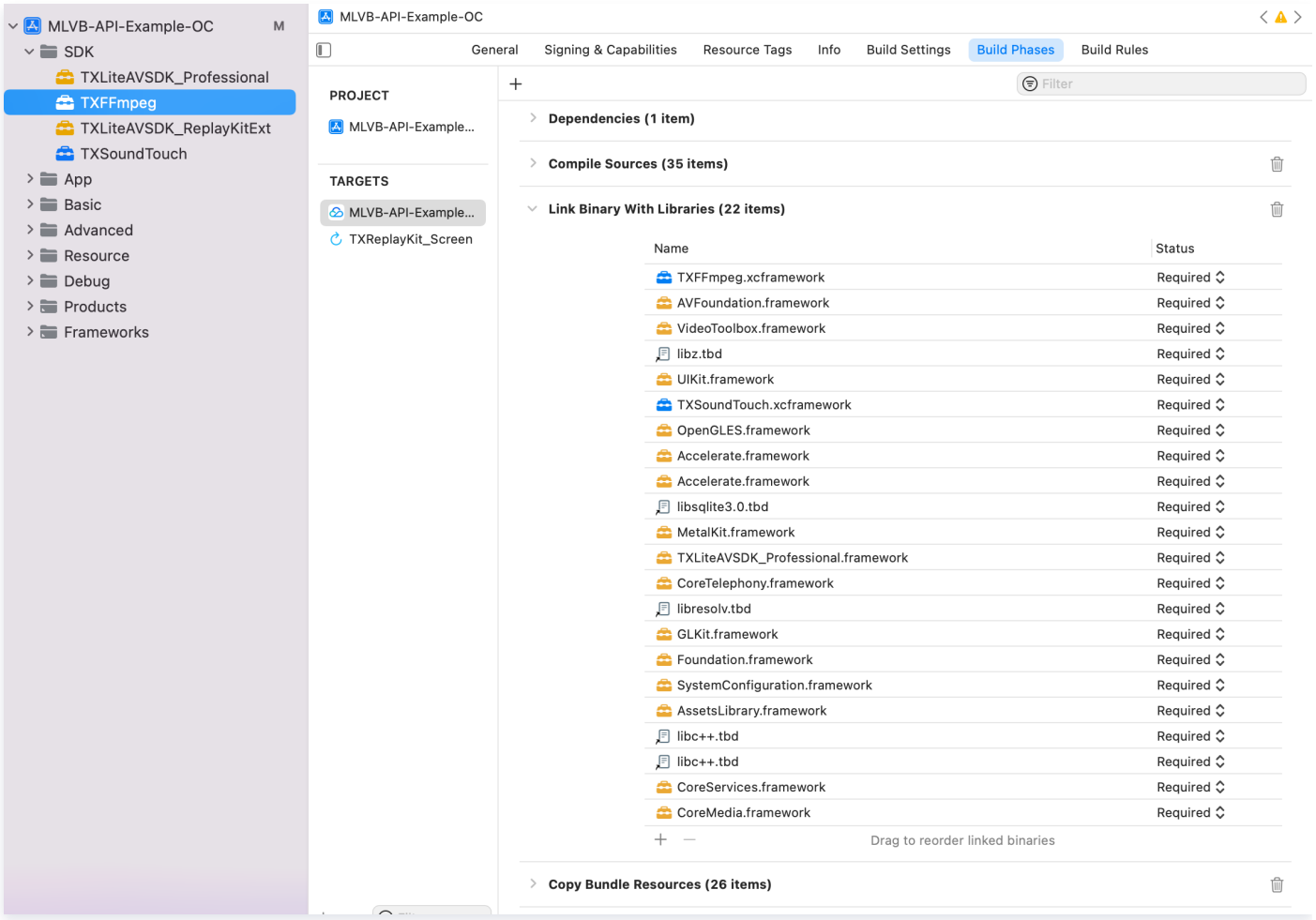
3. 单击 **Link Binary with Libraries** 项展开，单击底下的 + 添加依赖库。



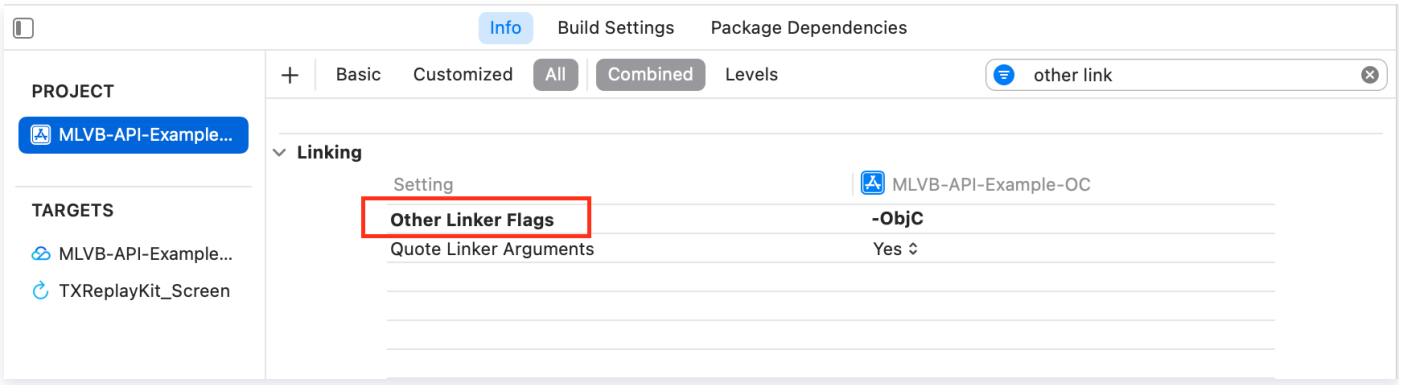
4. 依次添加所下载的 `TXLiteAVSDK_Professional.xcframework` 、 `TXFFmpeg.xcframework` 、 `TXSoundTouch.xcframework` 、 `TXLiteAVSDK_ReplayKitExt.xcframework` 及其所需依赖库：

```
AVFoundation.framework
```

```
VideoToolbox.framework
libz.tbd
OpenGLES.framework
Accelerate.framework
libsqlite3.0.tbd
MetalKit.framework
CoreTelephony.framework
libresolv.tbd
GLKit.framework
Foundation.framework
SystemConfiguration.framework
AssetsLibrary.framework
libc++.tbd
CoreServices.framework
CoreMedia.framework
AVKit.framework
MetalPerformanceShaders.framework
```



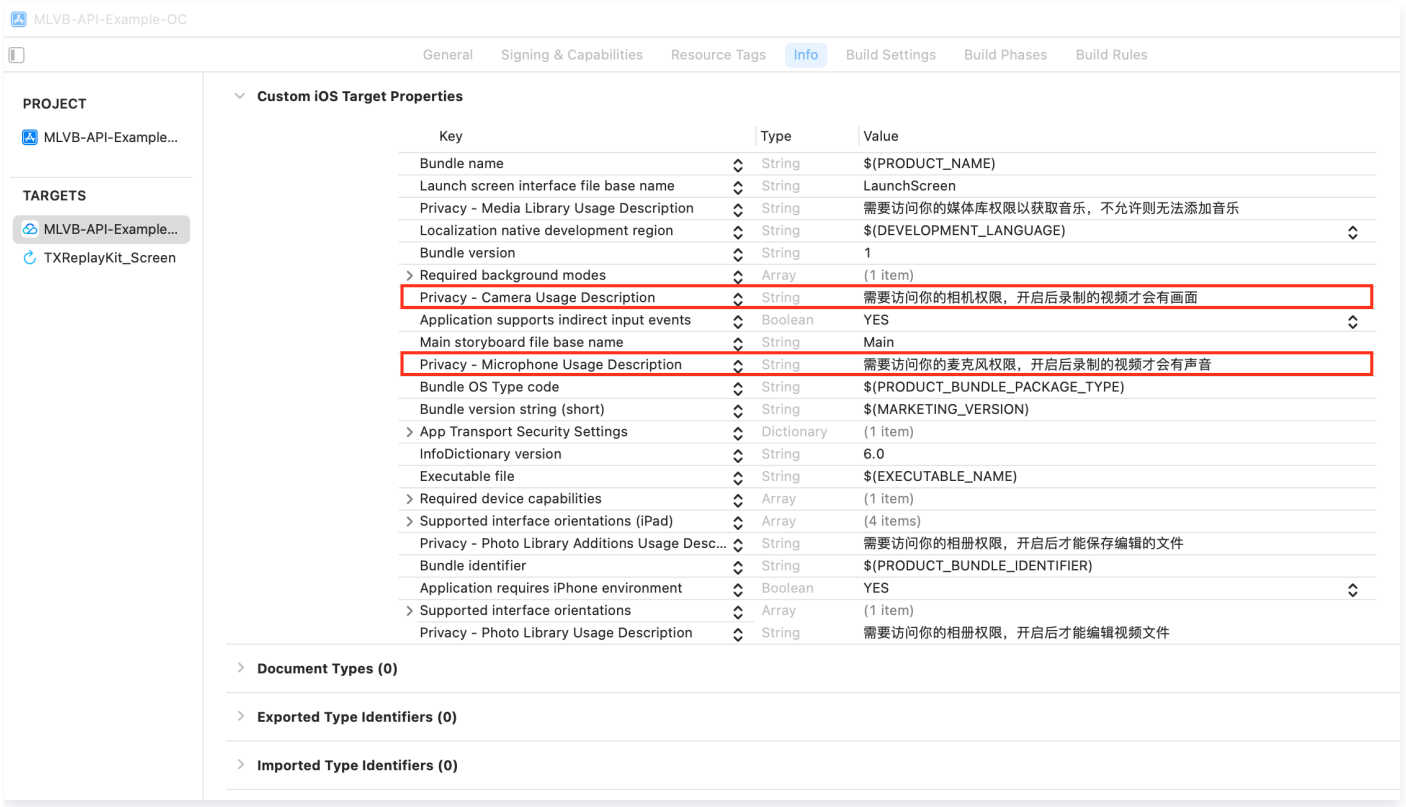
5. 选中 Build Settings 项，搜索 `Other Linker Flags` 。添加 `-ObjC` 。



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

1. 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。
2. 在您的 App 调用 LiteAVSDK 的相关功能之前（建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：

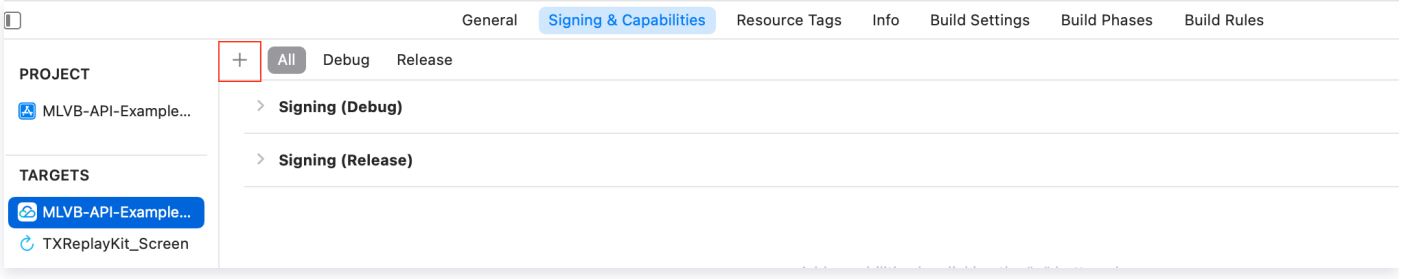
```
@import TXLiteAVSDK_Professional;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

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

常见问题

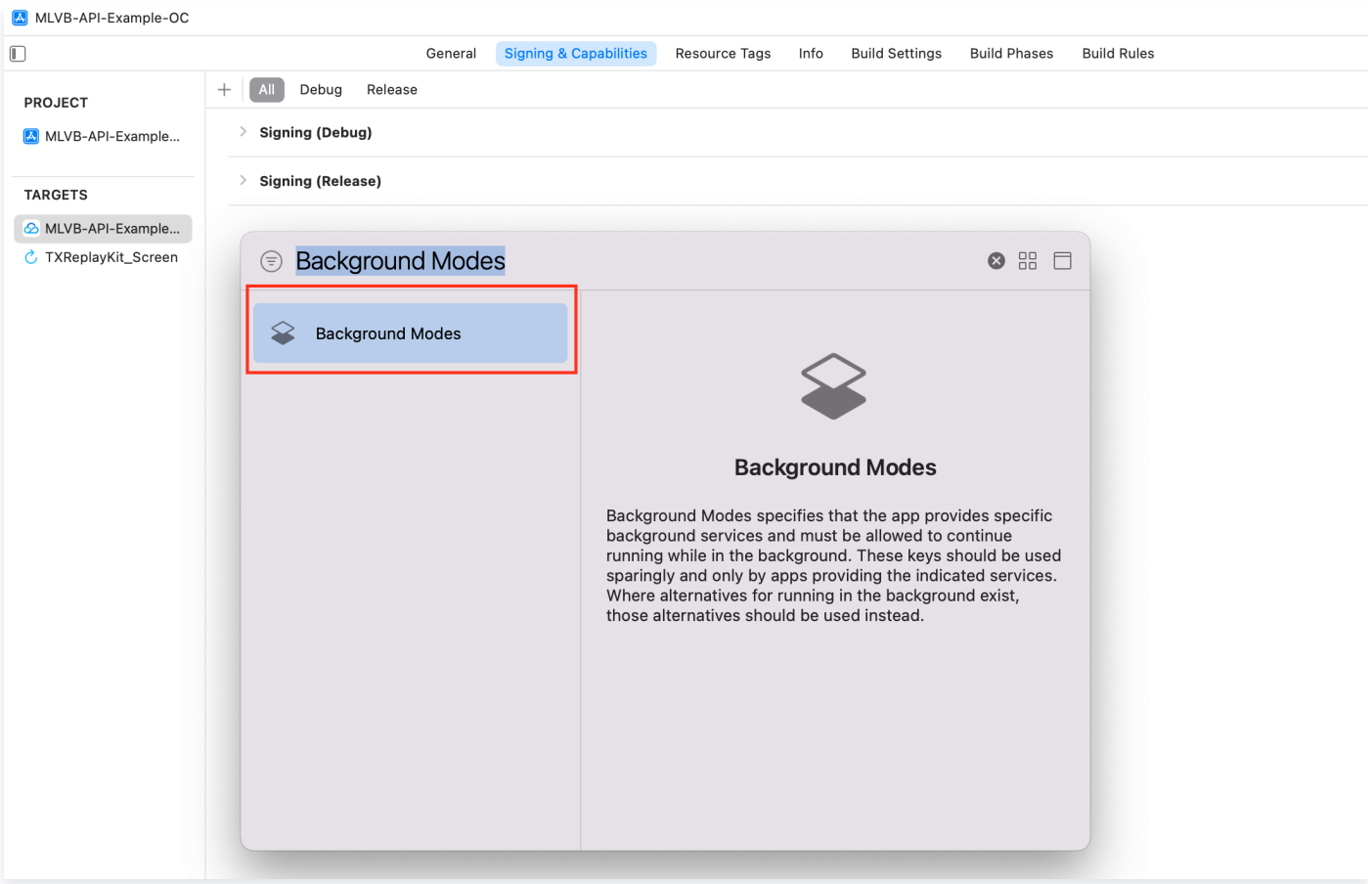
1. LiteAVSDK 是否支持后台运行?

支持，如需要进入后台仍然运行相关功能，操作如下：

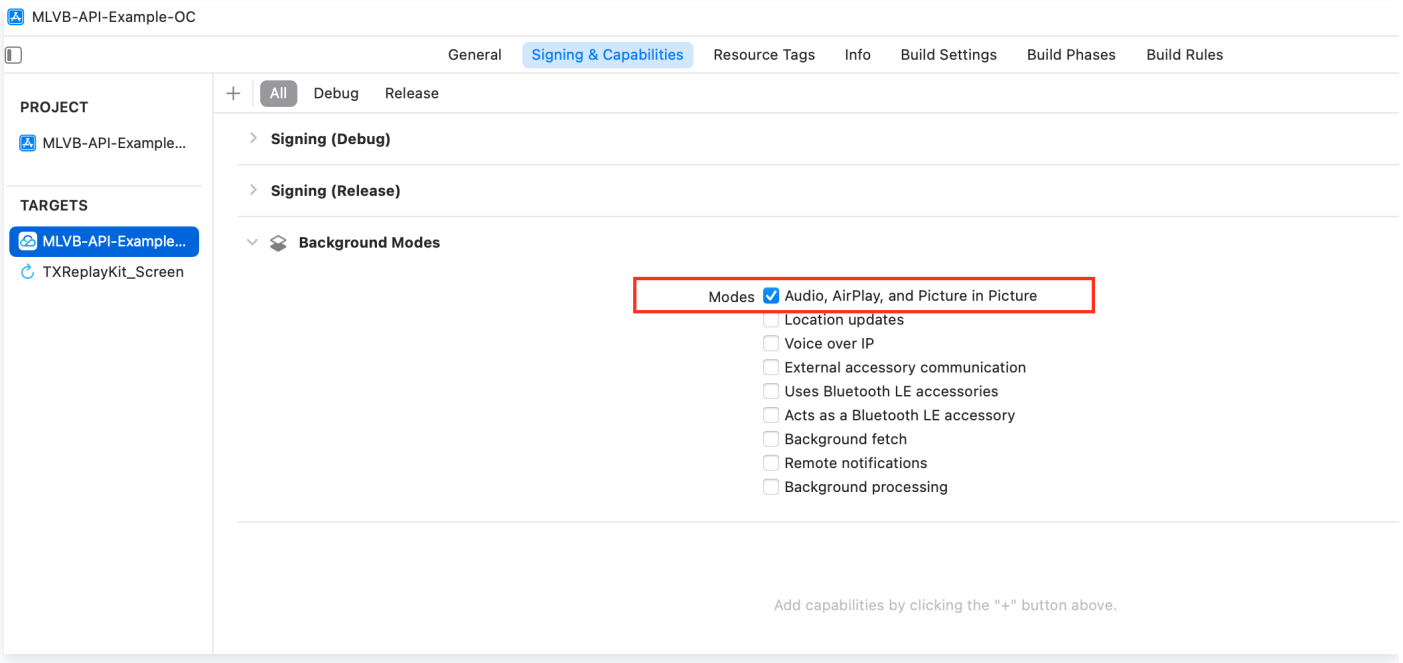
1. 选中当前工程项目，选择 **Signing&Capabilities**，单击左上角“+”，如图所示：



2. 选择 Background Modes。



3. 在 Background Modes中勾选 Audio，AirPlay and Picture in Picture ，如下图所示：



2. 项目里面同时集成了直播 SDK/实时音视频/播放器等 LiteAVSDK 系列的多个 SDK 报符号冲突问题怎么解决?

如果集成了2个或以上产品（直播、播放器、TRTC、短视频）的 LiteAVSDK 版本，编译时会出现库冲突问题，因为有些 SDK 底层库有相同符号文件，这里建议只集成一个全功能版 SDK 可以解决，直播、播放器、TRTC、短视频这些都包含在一个 SDK 里面。具体请参见

SDK 下载。

Android

最近更新时间：2025-06-04 17:30:31

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。
下面以全功能的 全功能版 SDK 为例：

开发环境要求

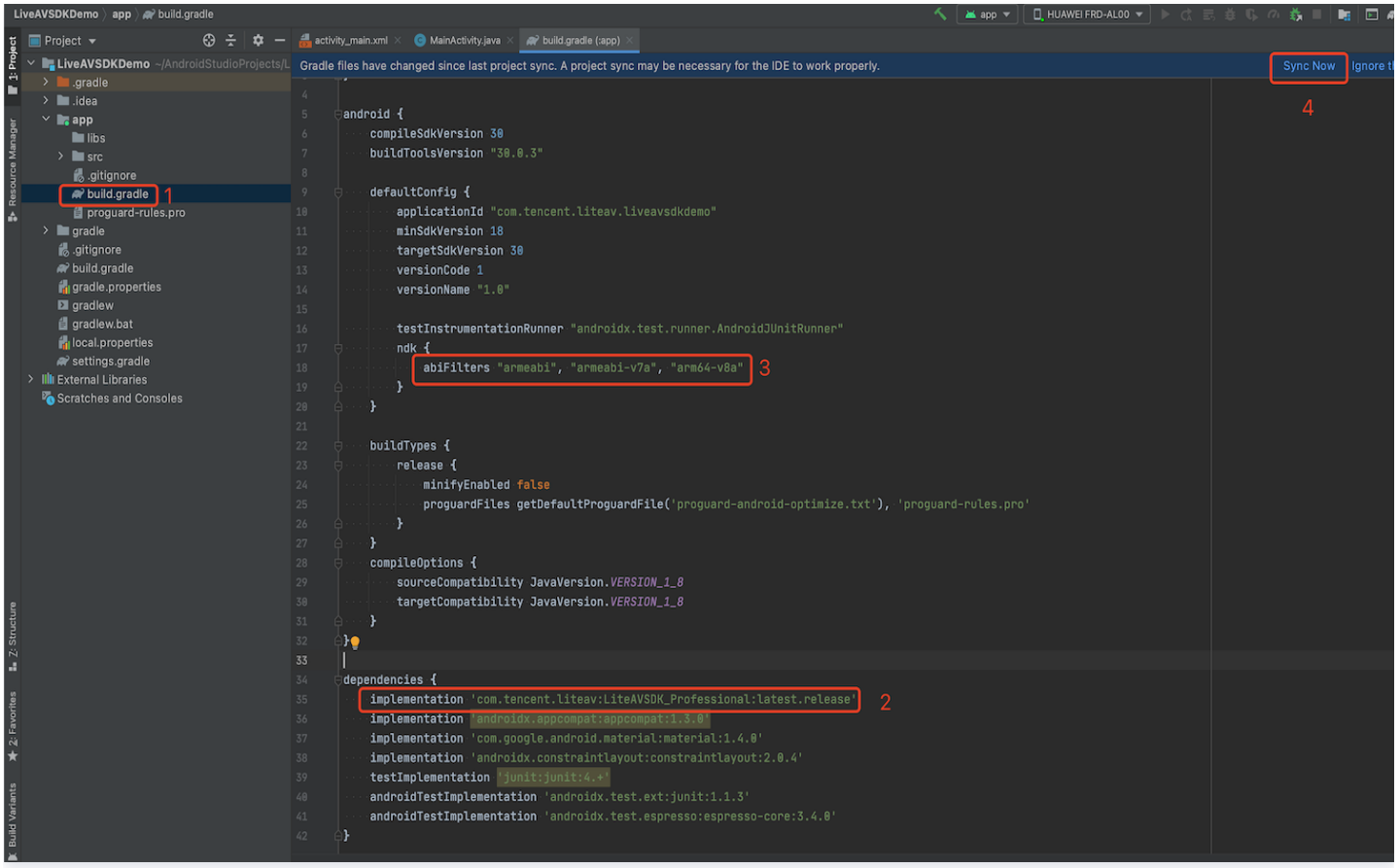
- Android Studio 2.0+。
- Android 4.1（SDK API 16）及以上系统。

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）。

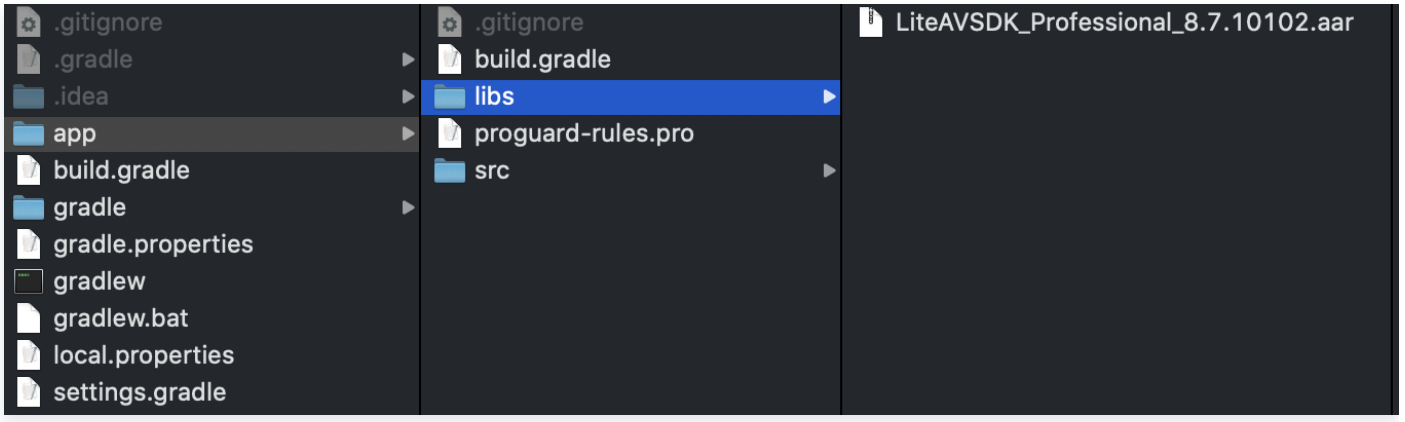
```
defaultConfig {
  ndk {
    abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
  }
}
```

4. 单击 Sync Now 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。

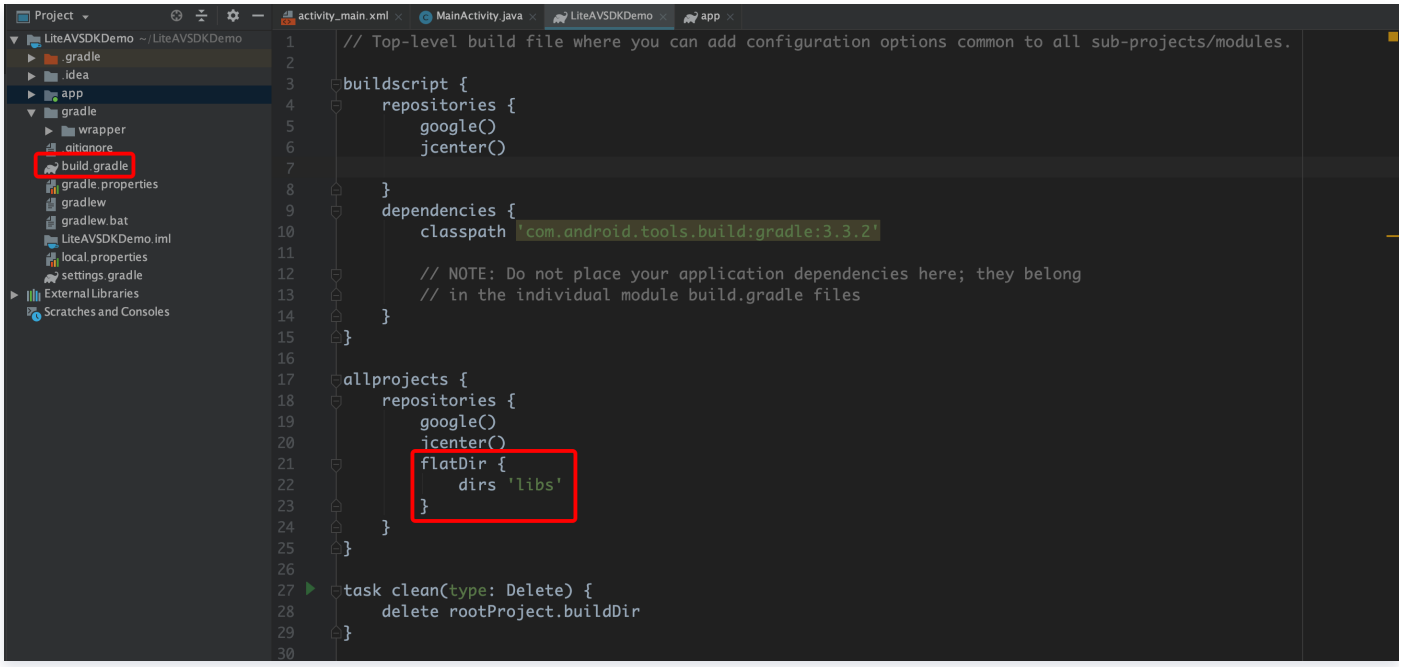
方法二：手动下载（aar）

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：

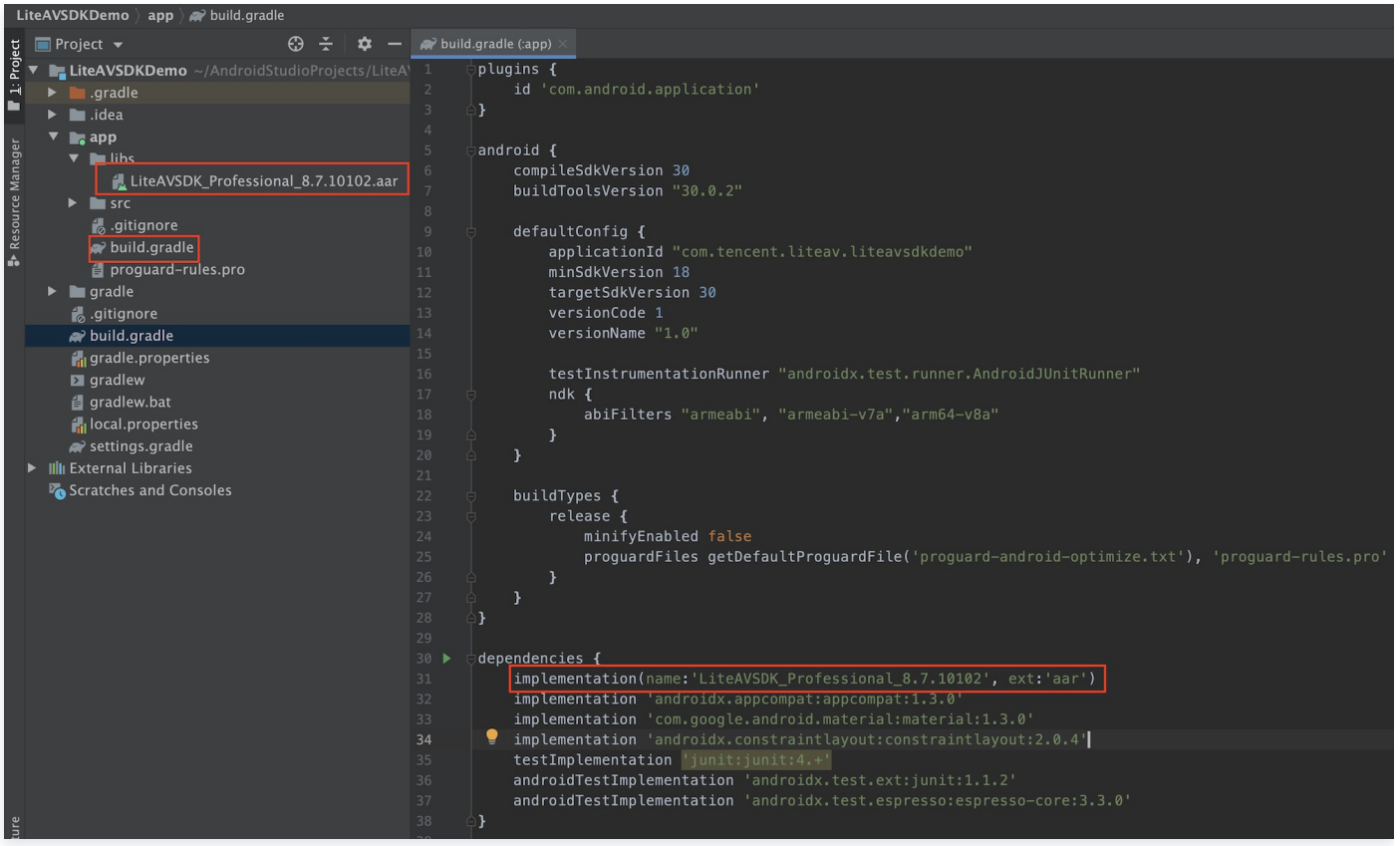
1. 下载 LiveAVSDK ，下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：



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）。

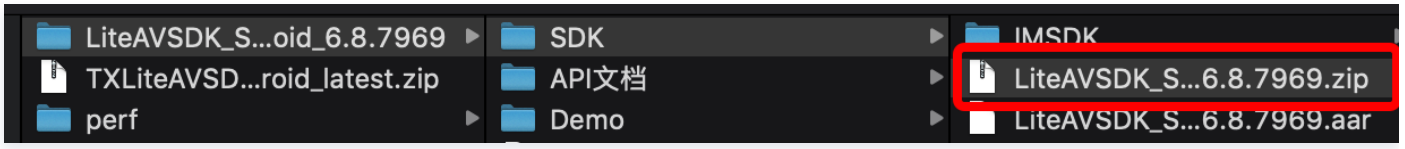
```
defaultConfig {
      ndk {
          abiFilters "armeabi-v7a", "arm64-v8a"
      }
}
```

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

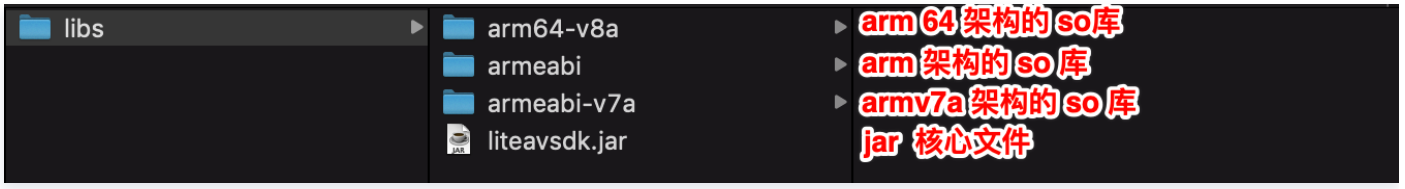
集成 SDK（jar）

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：

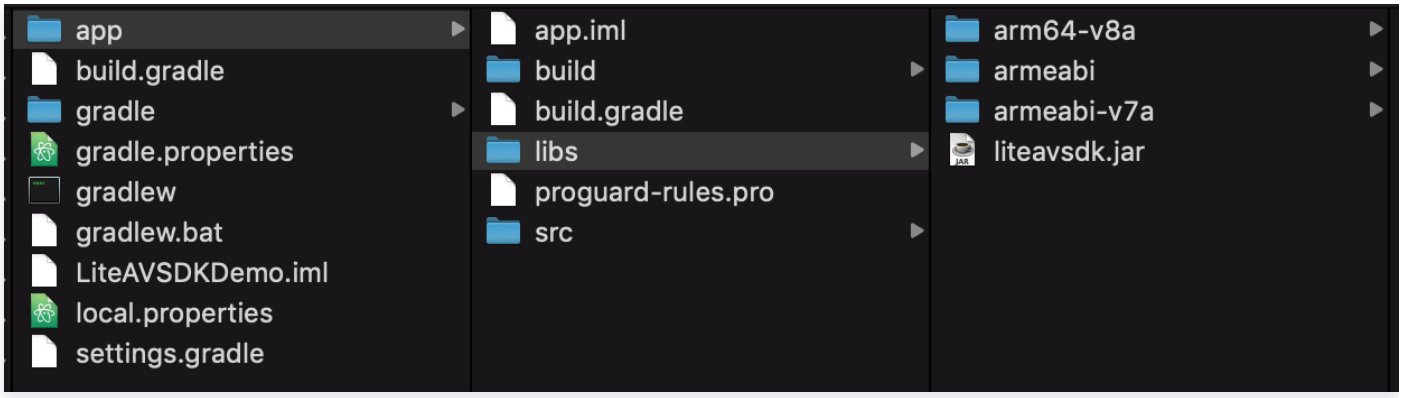
1. 下载 LiveAVSDK ，下载完成后进行解压。在 SDK 目录下找到 `LiteAVSDK_Professional_xxx.zip`（其中 `xxx` 为 LiteAVSDK 的版本号）：



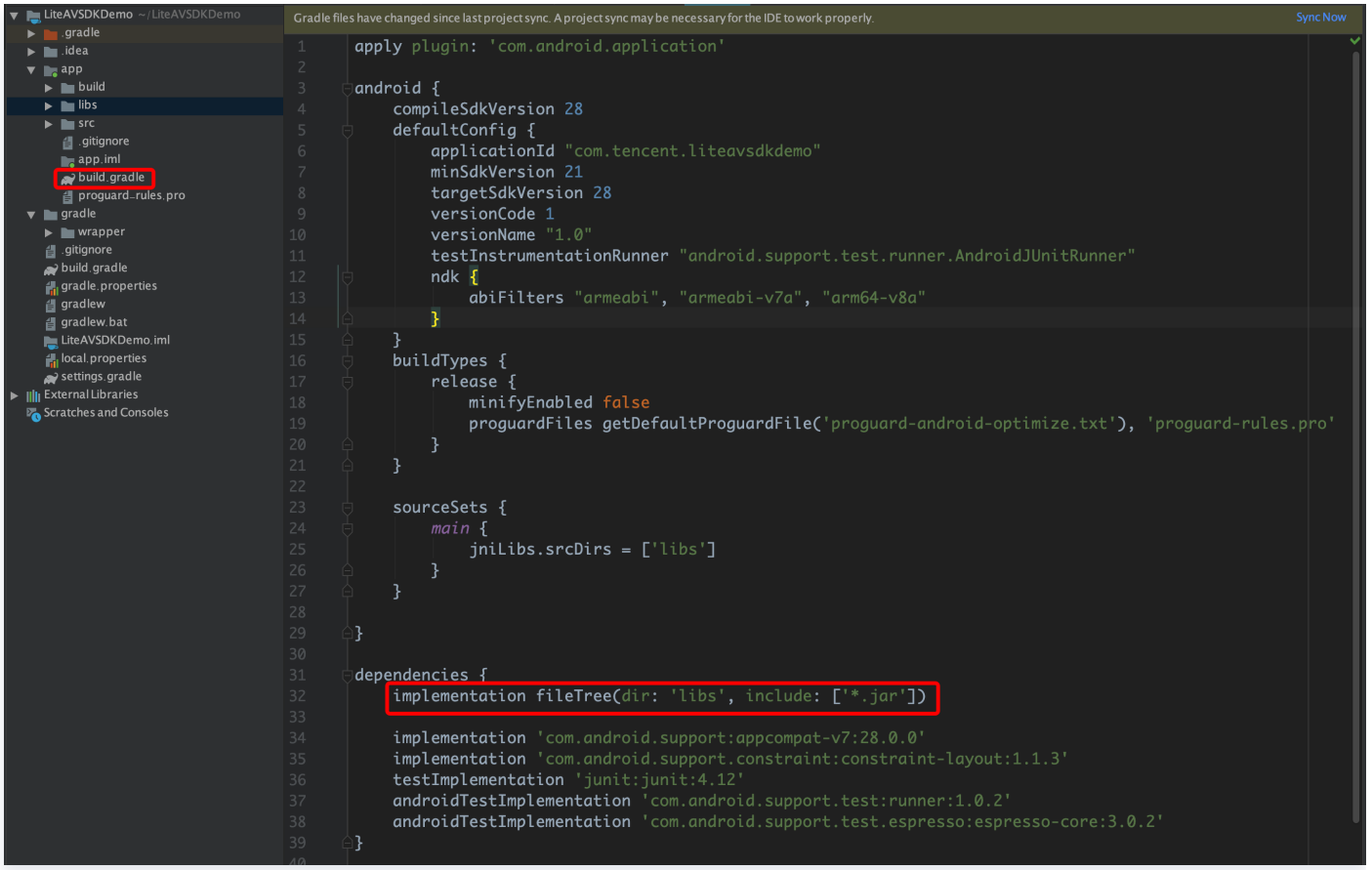
解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：



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。

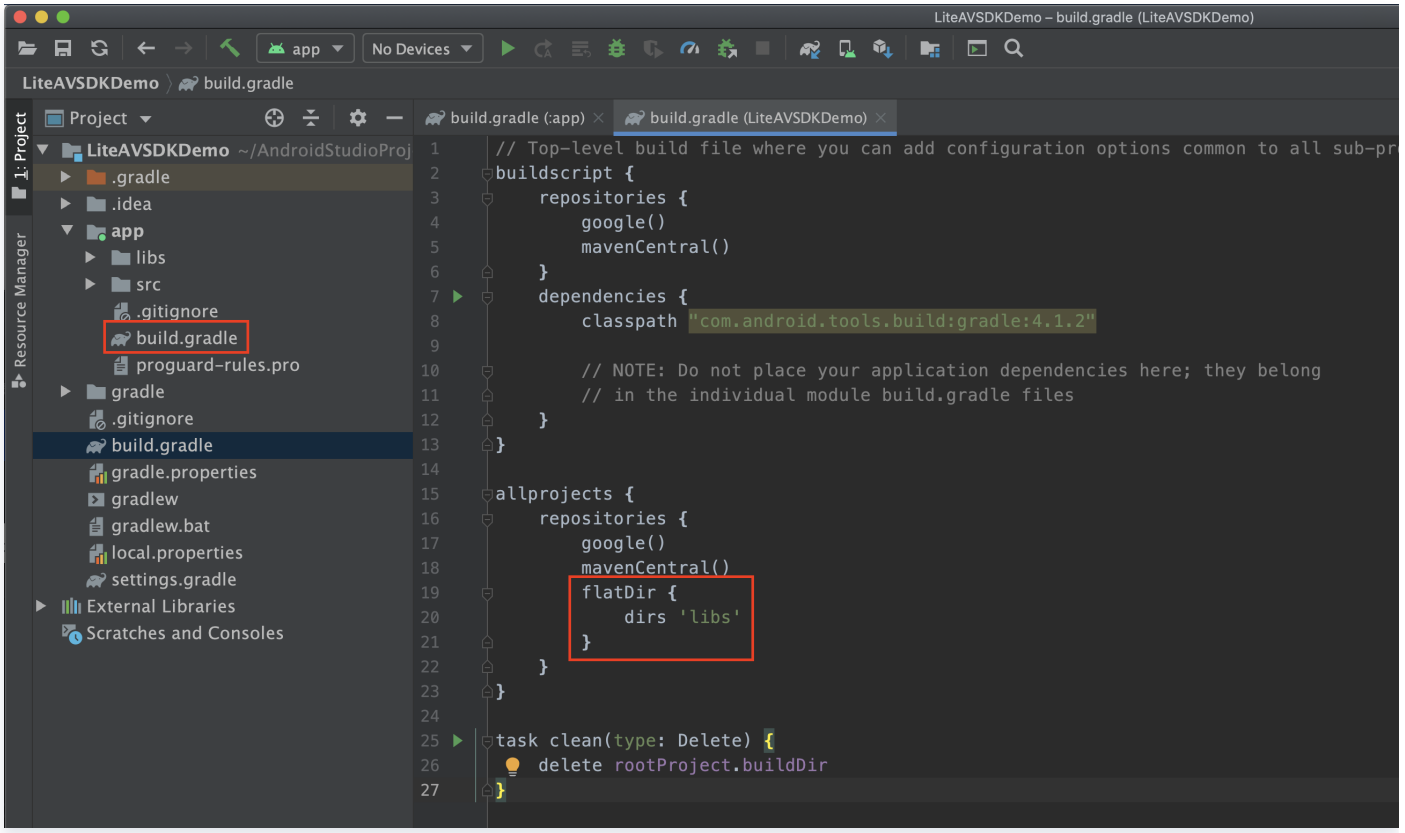


3. 在 `app/build.gradle` 中，添加引用 jar 库的代码。

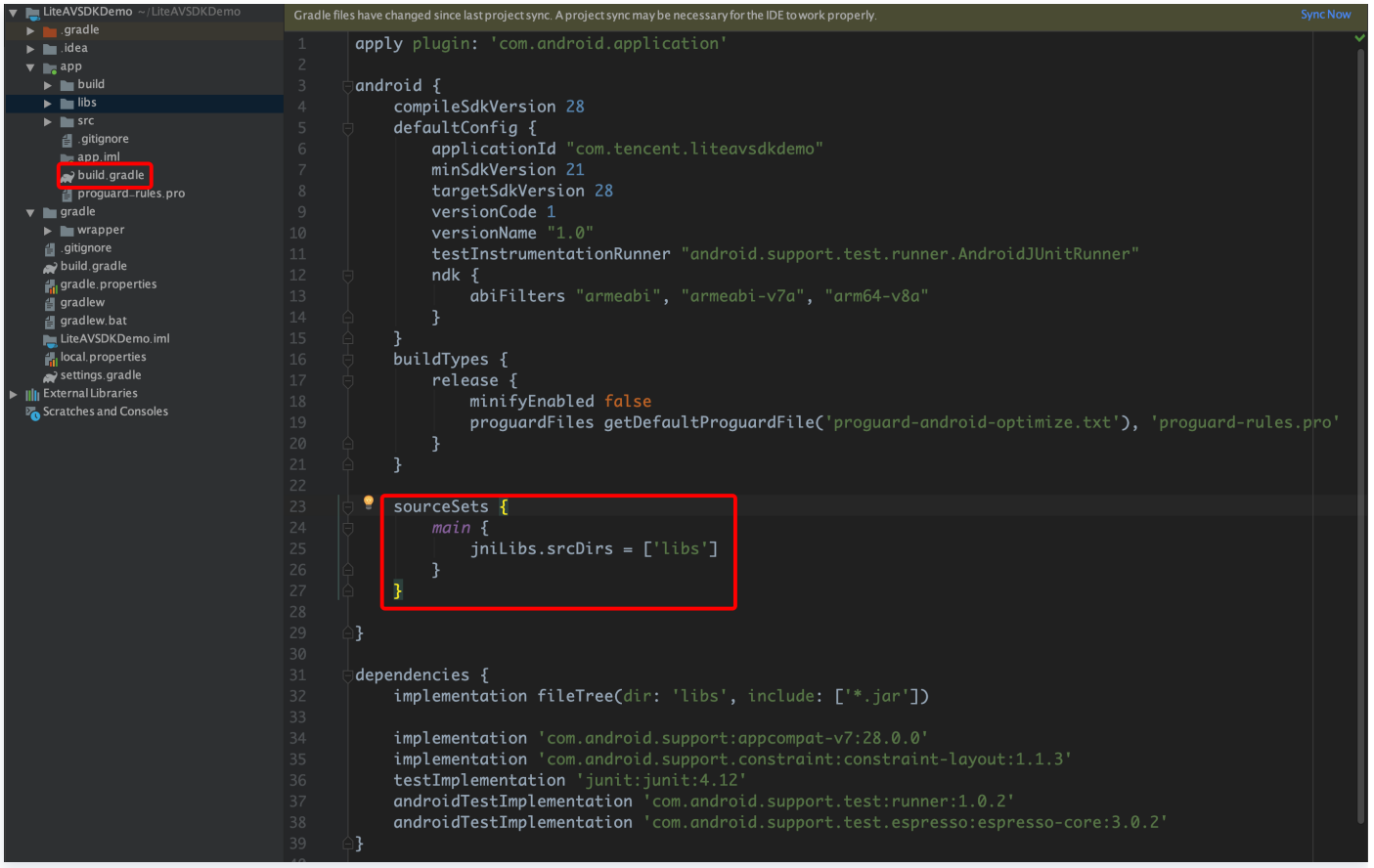



```
dependencies{
  implementation fileTree(dir:'libs',include:['*.jar'])
}
```

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 同步 SDK，完成 LiteAVSDK 的集成工作。

配置 App 打包参数


```
    packagingOptions {
        pickFirst '**/libc++_shared.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.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 调用全功能版 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.** { *; }
```

常见问题

1. Android 端使用 LiteAVSDK 录屏/屏幕共享功能必现 crash 问题怎么解决?

请您先检查下项目里面 targetSdkVersion 设置，如果设置的29那么运行 Android 10 设备使用录屏共享会触发闪退问题。原因是安卓隐私策略有更改，解决办法需要启动前台 service，并指定 type 为 mediaProjection 即可，不需要在 Service 里面调用 startScreenCapture。

2. 项目里面同时集成了直播 SDK/实时音视频/播放器等 LiteAVSDK 系列的多个 SDK 报符号冲突问题怎么解决?

如果集成了2个或以上产品（直播、播放器、TRTC、短视频）的 LiteAVSDK 版本，编译时会出现库冲突问题，因为有些 SDK 底层库有相同符号文件，这里建议只集成一个全功能版 SDK 可以解决，直播、播放器、TRTC、短视频这些都包含在一个 SDK 里面。具体请参见 SDK 下载 。

微信小程序

最近更新时间：2024-07-15 17:56:21

以下视频将为您讲解直播 SDK 如何快速集成到微信小程序：
观看视频

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** > **开发管理** > **开发设置**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）： 1.《信息网络传播视听节目许可证》 2.《网络文化经营许可证》（经营范围含网络表演）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 **注意：** 1. 如提供时政信息服务，需补充：时政信息类目 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）： 1.《事业单位法人证书》（适用公立学校） 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） 3.《信息网络传播视听节目许可证》 4. 全国校外线上培训管理服务平台备案 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）： 1. 卫生健康部门的《设置医疗机构批准书》； 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	

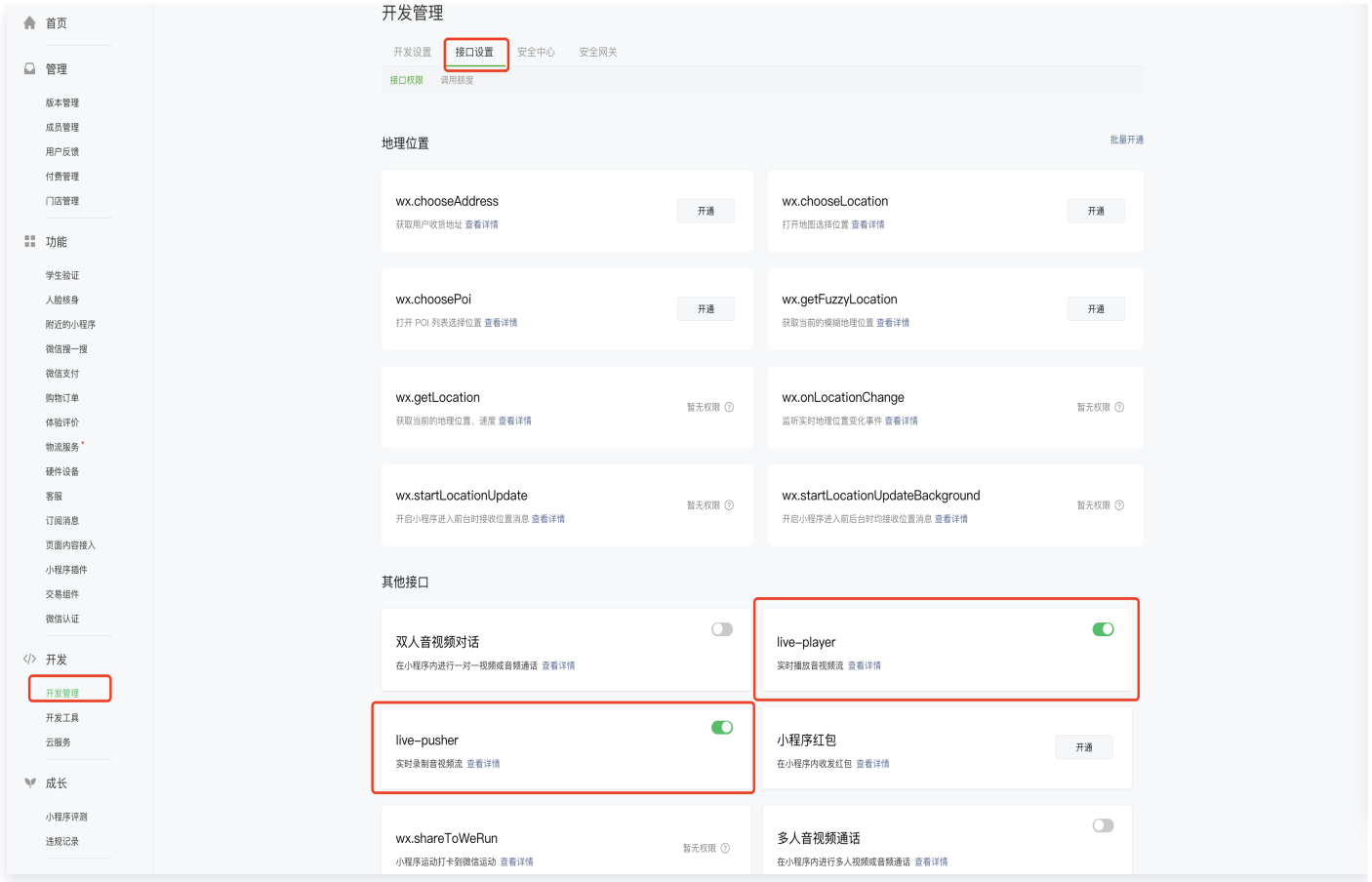
金融	银行	（2选1）： 1.《金融许可证》 2.《金融机构许可证》	适用于提供银行业务在线服务或交易等服务	金融产品视频客服理赔、金融产品推广直播等
	信托	（2选1）： 1.《金融许可证》 2.《金融机构许可证》	适用于提供信托理财业务在线服务或交易等服务	
	公募基金	（3选1）： 1.《经营证券期货业务许可证》且业务范围必须包含“基金” 2.《基金托管业务许可证》 3.《基金销售业务资格证书》	适用于基金管理公司从事股票、债券等金融工具的投资服务	
	私募基金	（2选1）： 1.《私募基金备案证明》 2.《私募投资基金管理人登记证书》	仅适用于私募基金展示、介绍、咨询等服务 注：暂不支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）： 1.《证券投资咨询业务资格证书》 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）： 1.《保险公司法人许可证》 2.《经营保险业务许可证》 3.《保险营销服务许可证》 4.《经营保险代理业务许可证》 5.《经营保险经纪业务许可证》 6.《经营保险公估业务许可证》 7.《经营保险资产管理业务许可证》 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）： 1. 经营个人征信业务：《个人征信业务经营许可证》、《营业执照》 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 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）： 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》 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》 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 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
政府主体账号	–	–	–	政府相关工作推广直播、领导讲话直播等
IT 科技	音视频设备	《中国国家强制性产品认证证书》	为多方提供电话会议/视频会议等服务；智能家居场景下控制摄像头	适用于提供音视频设备、信息技术设备、电信终端设备等线下硬件在线销售及服务。
	多方通信	《增值电信业务经营许可证》（业务范围需含"国内多方通信服务业务"）		仅适用于为多方提供电话会议/视频会议等服务
房地产服务	房地产营销	（3选1）： 1.《商品房预售许可证》 2.《商品房销售许可证》 3.《商品房现售备案证明》	房地产营销直播服务、在线音视频带看等	适用于房地产开发商提供购房意向款、优惠券、权益券等营销活动

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。

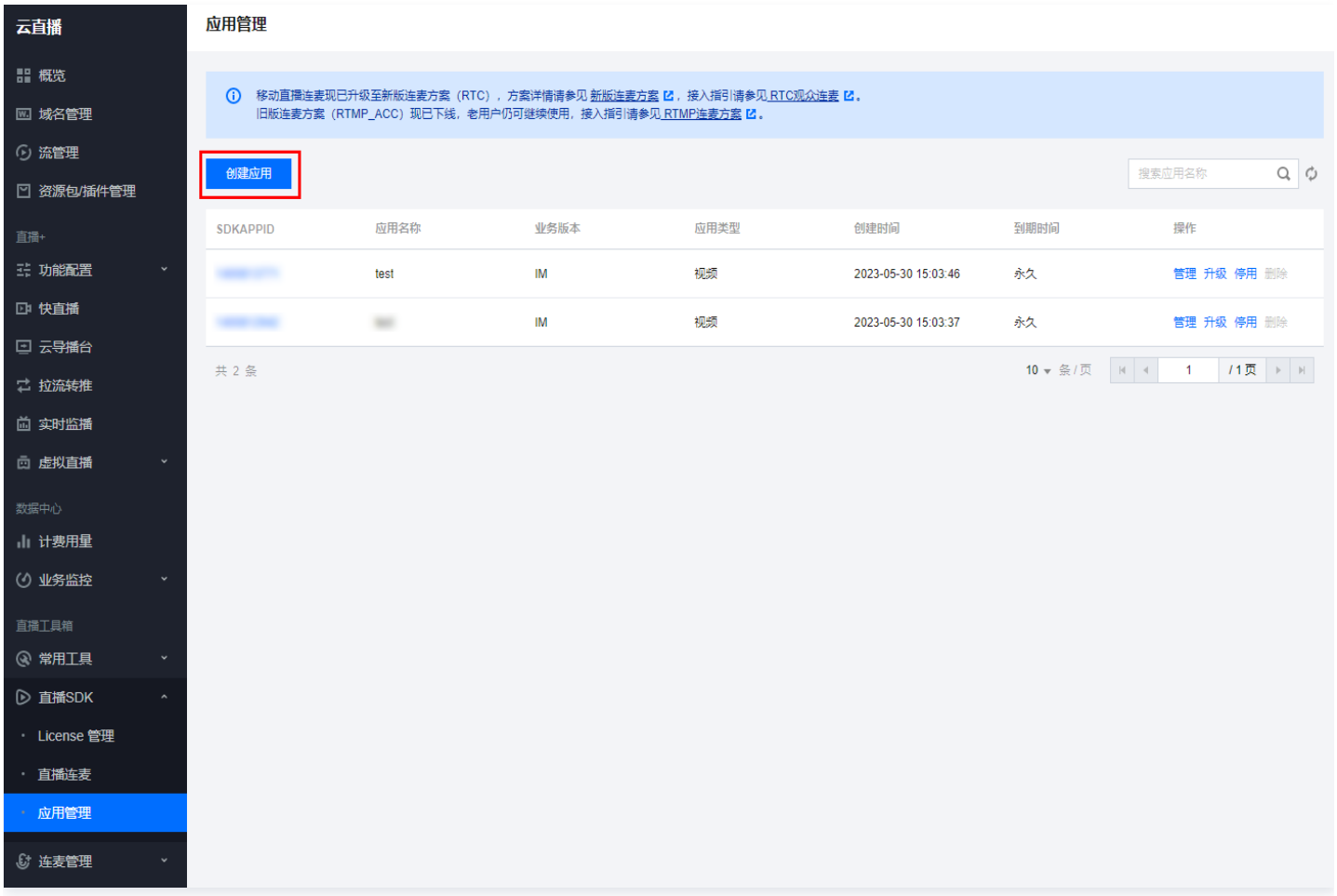
四、开通即时通信 IM 服务

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

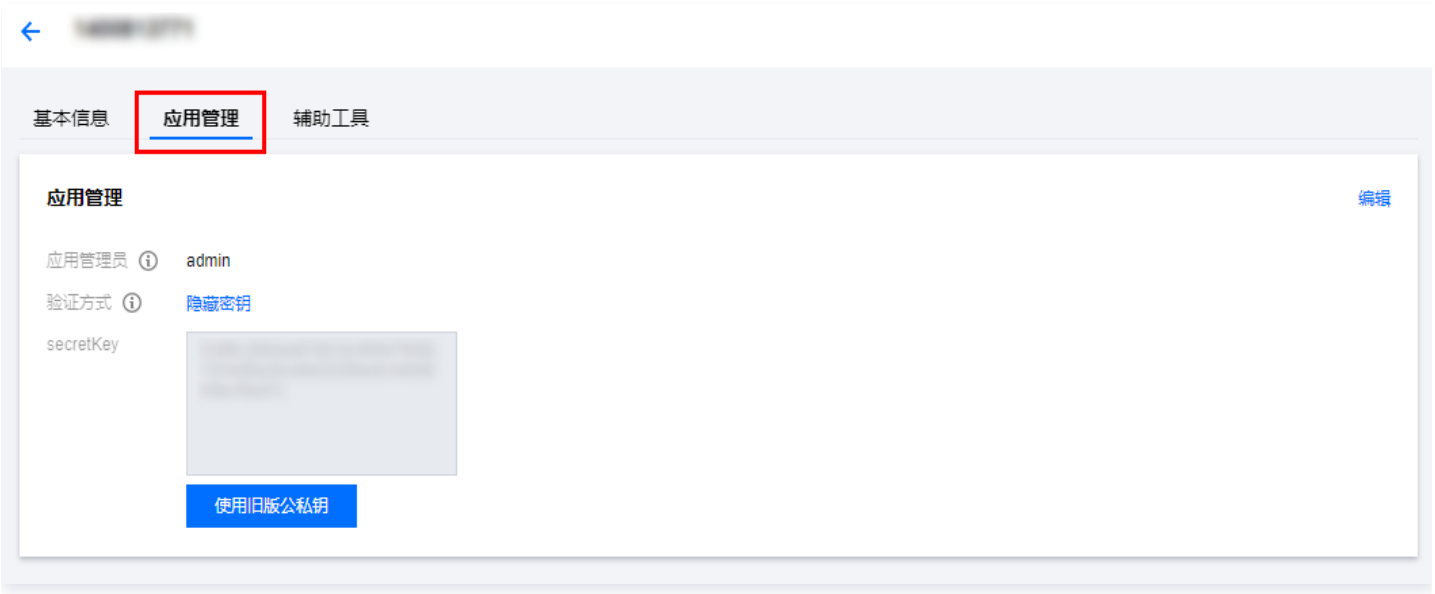
进入 **云直播控制台** > **直播 SDK** > **应用管理** 页面，单击**创建应用**填写应用信息。完成创建后，您将会在应用列表中看到您创建的应用，记录其 SDKAppID 信息，如下图所示：



> **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

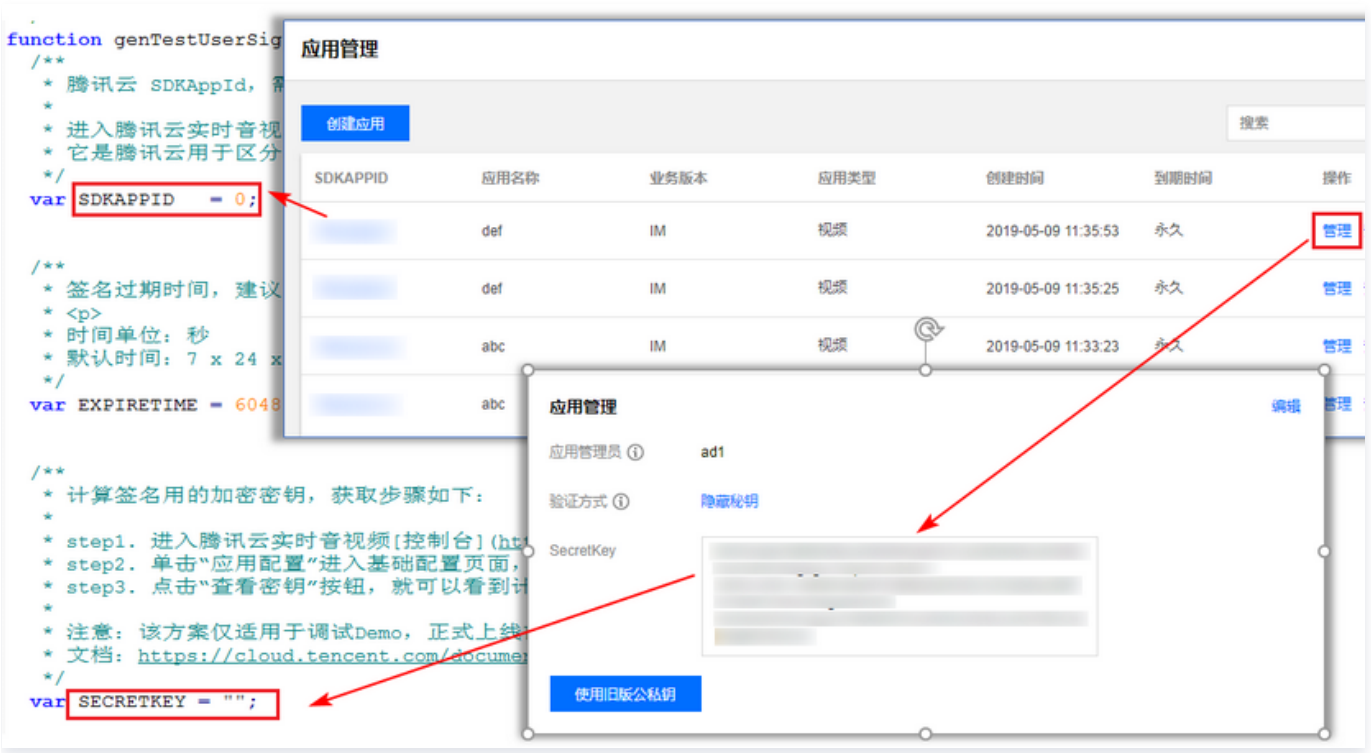
访问 Github，获取小程序 Demo 源码。

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，创建一个小程序项目，选择**不使用模板**。
2. 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

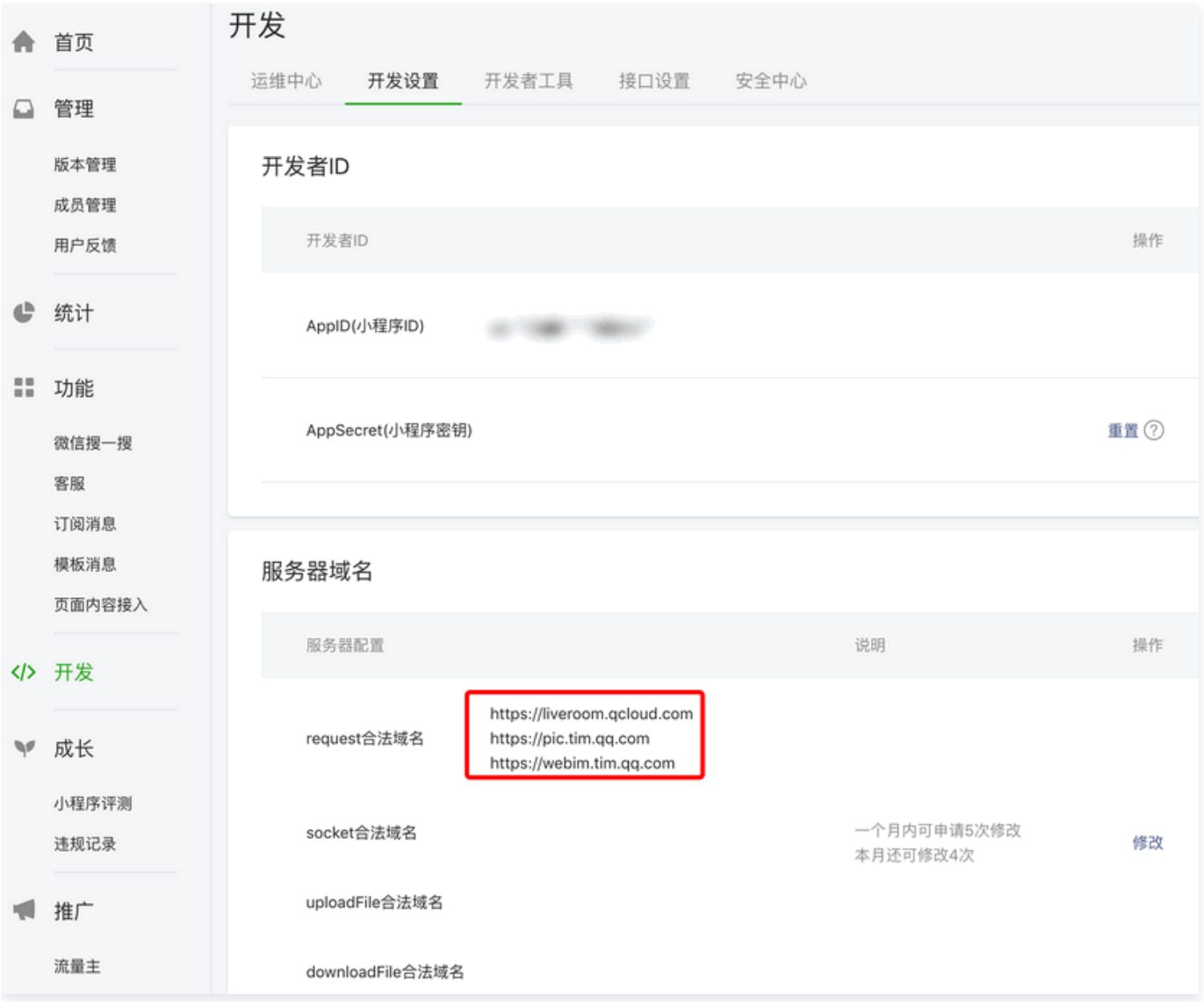
- 小程序 <live-player> 和 <live-pusher> 标签需要在手机微信上才能使用，微信开发者工具上无法使用。
- 为了小程序能够使用腾讯云房间管理服务，您需要在手机微信上开启调试功能：手机微信扫码二维码后，单击右上角"..." > **开发调试**。

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`	直播 SDK 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。

Flutter

最近更新时间：2023-10-11 16:20:22

本文主要介绍如何快速地将 live_flutter_plugin Flutter 插件 集成到您的项目中，按照如下步骤进行配置，就可以完成 live_flutter_plugin 的集成工作。示例工程请参见 live_flutter_plugin_example 。

环境准备

- Flutter 2.0 及以上版本。
- **Android 端开发：**
  - Android Studio 3.5及以上版本。
  - App 要求 Android 4.1及以上版本设备。
- **iOS & macOS 端开发：**
  - Xcode 11.0及以上版本。
  - osx 系统版本要求 10.11 及以上版本
  - 请确保您的项目已设置有效的开发者签名。

快速集成 SDK

Flutter SDK 已经发布到 pub 库，您可以通过配置 pubspec.yaml 自动下载更新。

1. 在项目的 pubspec.yaml 中写如下依赖：

```
dependencies:
  live_flutter_plugin: latest version number
```

2. 开通摄像头和麦克风的权限，即可开启语音通话功能。

iOS

1. 需要在 `Info.plist` 中加入对相机和麦克风的权限申请：

```
<key>NSCameraUsageDescription</key>
<string>授权摄像头权限才能正常视频通话</string>
<key>NSMicrophoneUsageDescription</key>
<string>授权麦克风权限才能正常语音通话</string>
```

2. 开通摄像头和麦克风的权限，即可开启语音通话功能。

Android

1. 打开 `/android/app/src/main/AndroidManifest.xml` 文件。
2. 将 `xmlns:tools="http://schemas.android.com/tools"` 加入到 manifest 中。
3. 将 `tools:replace="android:label"` 加入到 application 中。

> **说明**

> 若不执行此步，会出现 Android Manifest merge failed 编译失败问题。

android > app > src > main > AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mlp">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        tools:replace="android:label"
        android:name="io.flutter.app.FlutterApplication"
        android:label="mlp"
        android:icon="@mipmap/ic_launcher">
```

快速开始

1. 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。
2. 在您的 App 调用 live_flutter_plugin 的相关功能之前进行如下设置：

```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.cloud.tencent.com/live/license)
setupLicense() {
  // 当前应用的License LicenseUrl
  var LICENSEURL = "";
  // 当前应用的License Key
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

常见问题

更多常见问题请参见 Flutter 相关问题 。

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 。

iOS 无法显示视频（Android 正常）?

请确认在您的 info.plist 中 `io.flutter.embedded_views_preview` 是否为 `YES` 。

Android Manifest merge failed 编译失败?

1. 请打开 `/example/android/app/src/main/AndroidManifest.xml` 文件。
2. 将 `xmlns:tools="http://schemas.android.com/tools"` 加入到 manifest 中。
3. 将 `tools:replace="android:label"` 加入到 application中。

android > app > src > main > AndroidManifest.xml

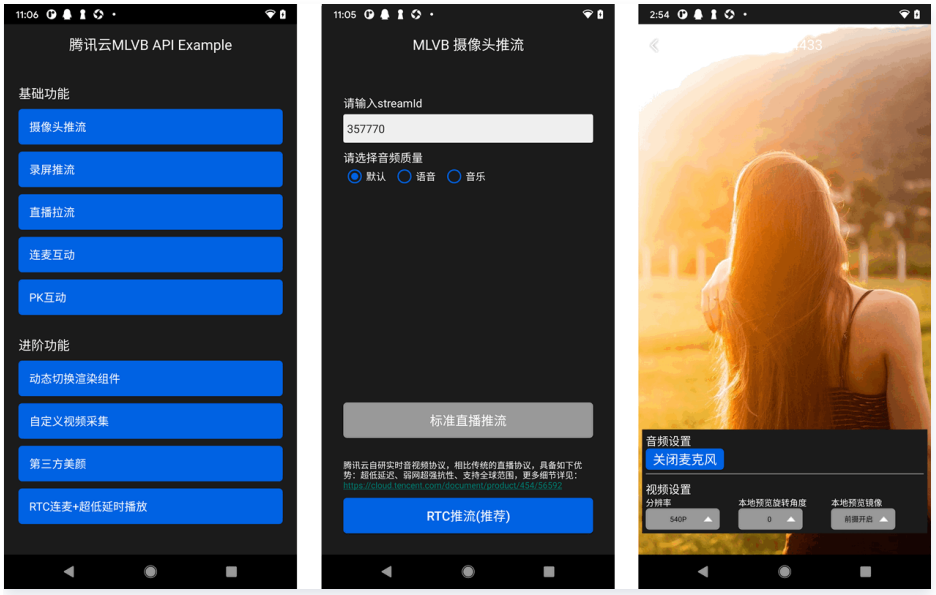
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mlp">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        tools:replace="android:label"
        android:name="io.flutter.app.FlutterApplication"
        android:label="mlp"
        android:icon="@mipmap/ic_launcher">
```

2. 直播推流
iOS
摄像头推流

最近更新时间：2024-11-28 17:05:52

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
Flutter	Github	live_camera_push.dart

功能对接

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

1. 获取 License 授权：

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

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

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

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

3. 初始化 V2TXLivePusher 组件

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

```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
// 指定对应的直播协议为 RTC，该协议支持连麦。如果在直播过程中有连麦需求，需要选择该协议
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
```

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，该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];
[_pusher startMicrophone];
```

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。

说明：

- 如果在 第3步 中选择 `RTMP` 协议推流，推流地址的生成请参见 RTMP 地址。
- 如果在 第3步 中选择 `RTC` 协议推流，推流地址的生成请参见 RTC 地址 。

```
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";     //不支持
连麦，直接推流到直播 CDN
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
NSString* url = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxxxx";  //支持连麦
[_pusher startPush:url];
```

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

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 中回调 `onLicenceLoaded` 的错误码和错误描述。

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 中打开摄像头的操作，仅仅打开麦克风即可。

```
[_pusher startMicrophone];
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";     //不支持连麦，直接推流到直播 CDN
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
NSString* url = @"trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx";  //支持连麦
[_pusher startPush:url];
```

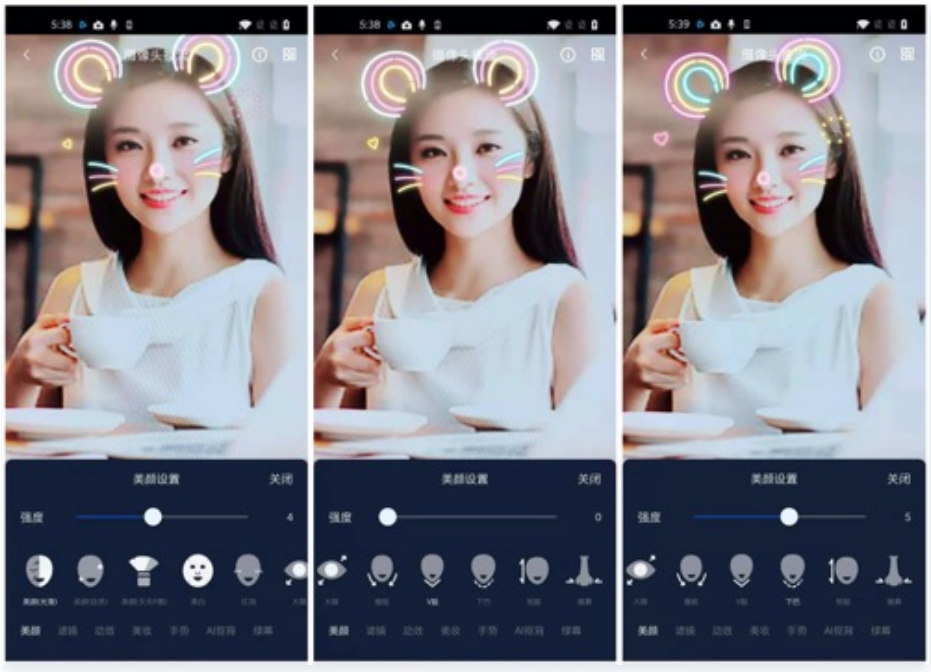
说明

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：

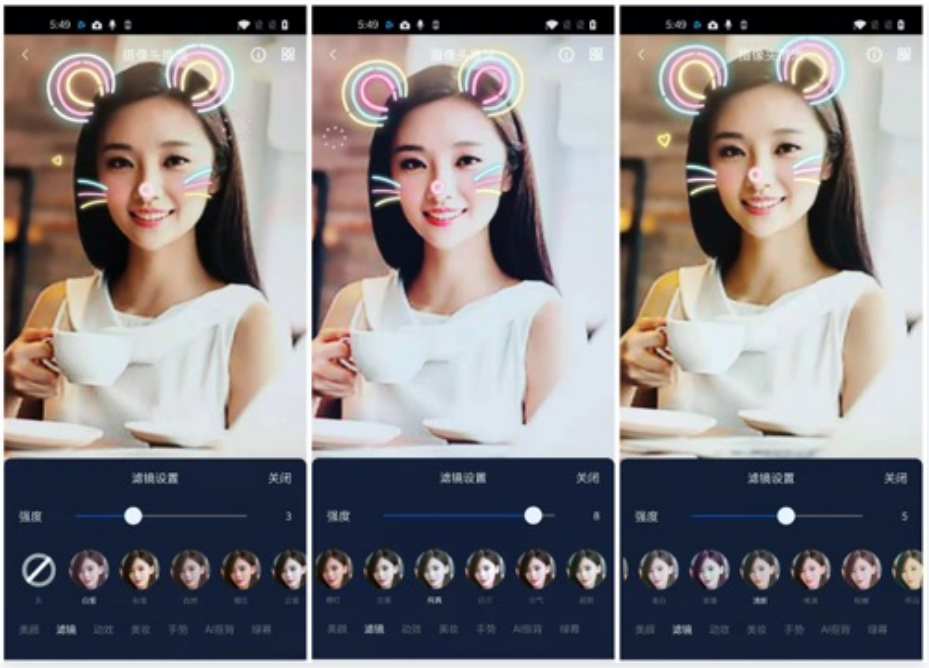
美颜风格	效果说明
TXBeautyStyleSmooth	光滑，适用于美女秀场，效果比较明显

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

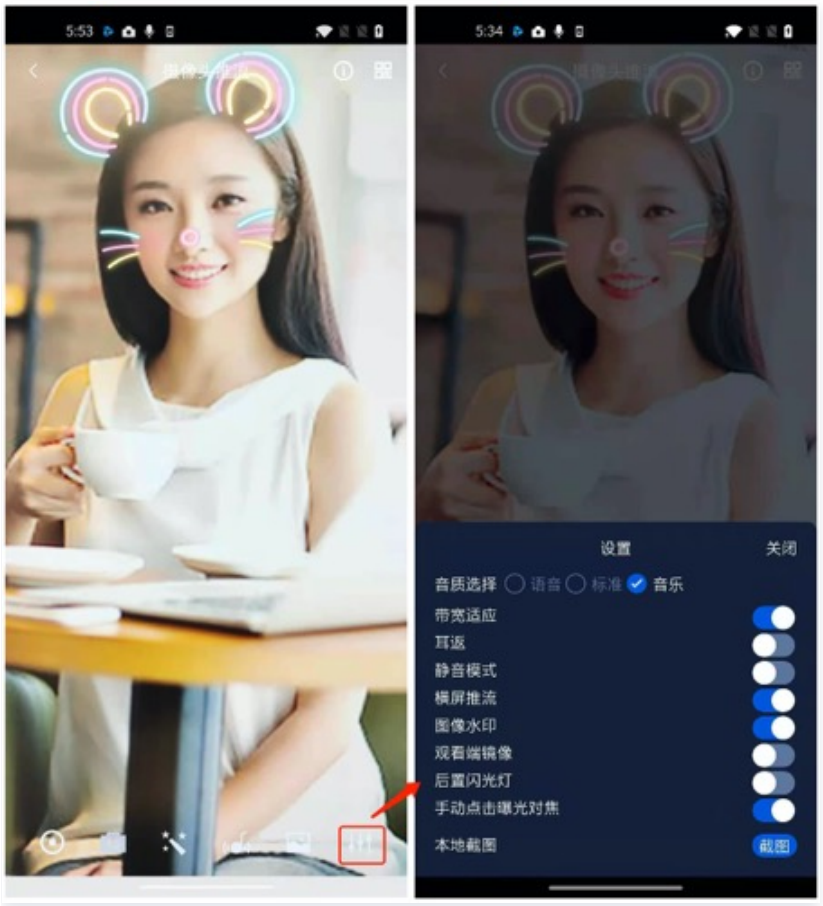


- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。 我们的设计师团队提供了17种默认的 色彩滤镜 供您使用。
- 调用 TXBeautyManager 的 `setFilterStrength` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

```
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 setEncoderMirror 可以改变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

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：

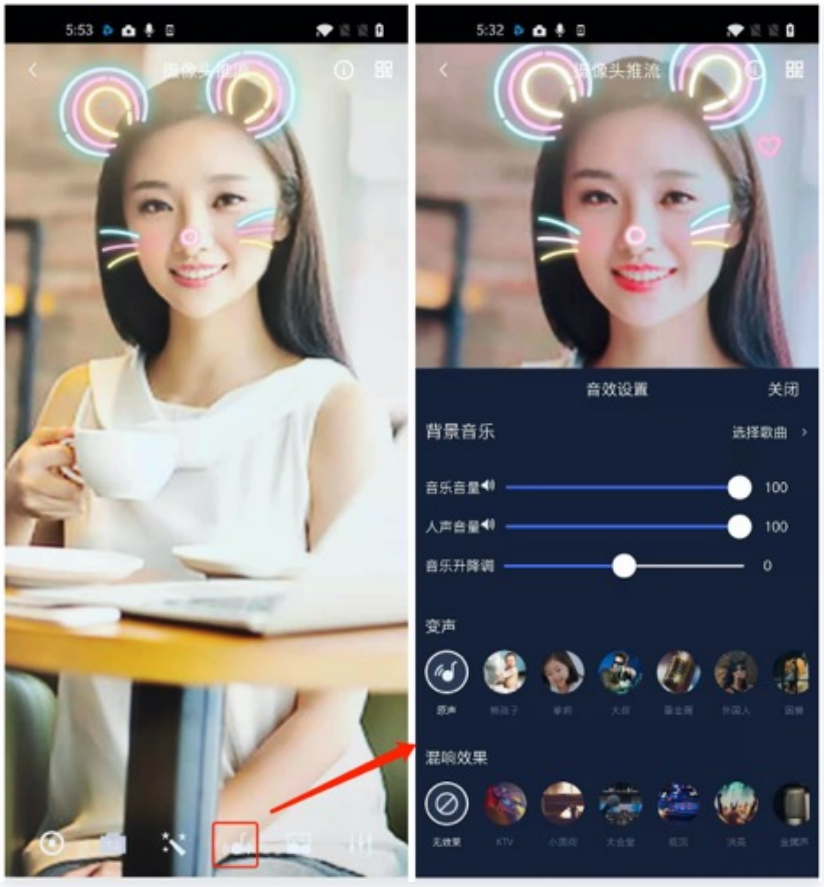
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
V2TXLiveVideoEncoderParam *videoEncoderParam = [[V2TXLiveVideoEncoderParam alloc]
initWith:V2TXLiveVideoResolution960x540];
videoEncoderParam.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionModeLandscape
: V2TXLiveVideoResolutionModePortrait;
[_pusher setVideoQuality:videoEncoderParam];
```

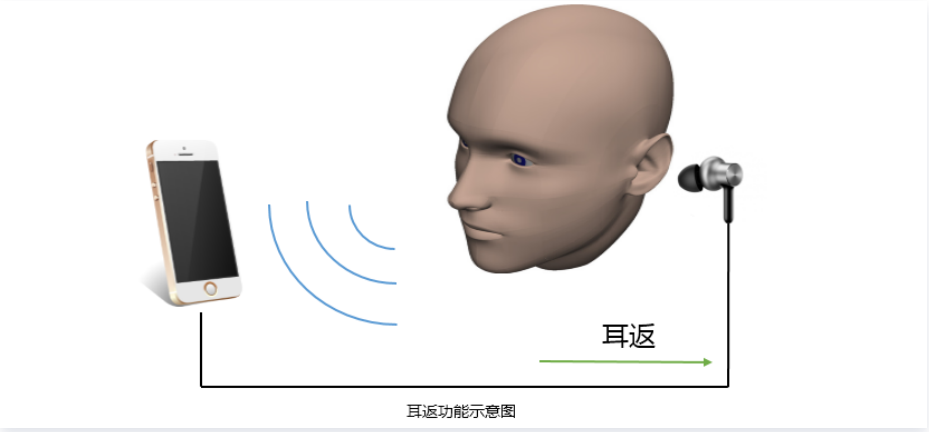
13. 音效设置

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。



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

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];
        }
    });
}
```

16. 发送 SEI 消息

调用 V2TXLivePusher 中的 sendSeiMessage 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：

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

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。

```
int payloadType = 5;
NSString* msg = @"test";
[_pusher sendSeiMessage:payloadType data:[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 V2TXLivePlayer 才能解析这些消息：

1. 设置：

```
int payloadType = 5;
[_player enableReceiveSeiMessage:YES payloadType:payloadType];
```

2. 当 V2TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 V2TXLivePlayerObserver 中的 onReceiveSeiMessage 回调来接收该消息。

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
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

录屏推流

最近更新时间：2025-01-21 10:42:32

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 SDK API-Example 工程中演示录屏推流的相关操作界面：

限制说明

- 录屏推流功能仅11.0以上系统可体验，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。更详细的使用说明可以参考 Demo 里 TXReplayKit_Screen 文件夹示例代码。
- 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。

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。

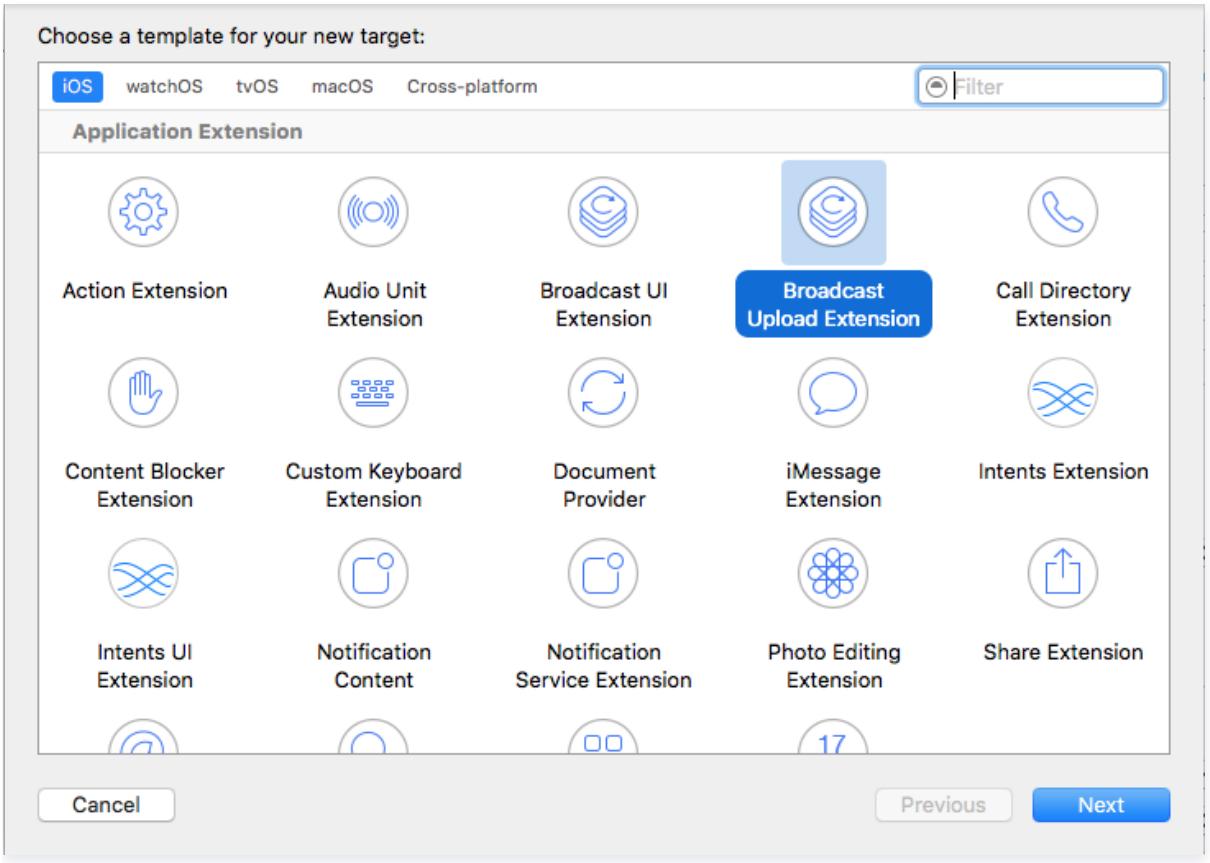
所属平台	GitHub 地址
iOS	LivePushScreenViewController.m
Android	LivePushScreenActivity.java
Flutter	live_screen_push.dart

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。

对接攻略

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

1. 获取 License 授权：

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

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

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

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
```

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

3. 初始化 V2TXLivePusher 组件

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

```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP];
// 指定对应的直播协议为 RTC，该协议支持连麦。如果在直播过程中有连麦需求，需要选择该协议
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
```

> **说明**

如果您有连麦需求，需要创建 `V2TXLivePusher` 对象时选择 `RTC` 协议。`RTC` 协议与 `RTMP` 协议在 API 使用上没有区别。

4. 配置 RPBroadcastSampleHandler

录屏需要使用系统 API RPBroadcastSampleHandler 的子类获取屏幕音视频数据，所以这里我们通过自定义子类 SampleHandler.m 中添加下面代码实现录屏推流：

```
#import "SampleHandler.h"
@import TXLiteAVSDK_ReplayKitExt;

@implementation SampleHandler
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupInfo {
    // App group ID，此处 ID 与 V2TXLivePusher startScreenCapture 对应，可以指定为 nil，但按照文档设置会使功能更加可靠。
    [[TXReplayKitExt sharedInstance] setupWithAppGroup:APPGROUP delegate:self];
}

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

- (void)broadcastResumed {
    // User has requested to resume the broadcast. Samples delivery will resume.
    [[TXReplayKitExt sharedInstance] broadcastResumed];
}

- (void)broadcastFinished {
    // User has requested to finish the broadcast.
    [[TXReplayKitExt sharedInstance] broadcastFinished];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleBufferType {
    [[TXReplayKitExt sharedInstance] sendSampleBuffer:sampleBuffer withType:sampleBufferType];
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)reason
{
    NSString *tip = @"";
    switch (reason) {
        case TXReplayKitExtReasonRequestedByMain:
            tip = NSLocalizedString(@"MLVB-API-Example.liveStop", "");
            break;
        case TXReplayKitExtReasonDisconnected:
            tip = NSLocalizedString(@"MLVB-API-Example.appReset", "");
            break;
        case TXReplayKitExtReasonVersionMismatch:
            tip = NSLocalizedString(@"MLVB-API-Example.sdkError", "");
            break;
```

```
    }

    NSError *error = [NSError errorWithDomain:NSStringFromClass(self.class)
                                         code:0
                                     userInfo:@{
                                         NSLocalizedFailureReasonErrorKey:tip
                                     }];
    [self finishBroadcastWithError:error];
}

@end
```

- RPBroadcastSampleHandler 的实现类中需要调用对应 TXReplayKitExt 的方法来设置录屏信息和处理录屏事件。
- **broadcastFinished** 回调可以获取录屏结束的原因，用以提示客户进一步处理操作。
- 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 。

5. 开启屏幕录制和推流

调用 startScreenCapture 启动屏幕录制，然后调用 V2TXLivePusher 中的 startPush 接口开始推流。

> **说明**
> 如果在 第3步 中选择 `RTMP` 协议推流，推流地址的生成请参见 RTMP 地址 。
> 如果在 第3步 中选择 `RTC` 协议推流，推流地址的生成请参见 RTC 地址 。

```
// appGroup 主 App 与 Broadcast 共享的 Application Group Identifier，可以指定为 nil，但按照文档设置会使功能更加可靠。
[livePusher startScreenCapture:@"group.com.xxx"];
[livePusher startMicrophone];

// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
NSString * const url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
NSString * const url = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxxxx";
V2TXLiveCode code = [livePusher startPush:url];
if (code != V2TXLIVE_OK) {
    // 请检查错误码
}
```

> **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 中回调 `onLicenceLoaded` 的错误码和错误描述。
> - iOS 11 的系统上仅能通过下拉通知栏，长按录屏按钮的方式开启屏幕录制。
> - iOS 12 及以上的系统可以通过 RPSystemBroadcastPickerView 来弹出录屏选择界面，具体用法请参见 TRTCBroadcastExtensionLauncher.m 。

6. 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：

```
BOOL landScape; //YES:横屏， NO:竖屏
V2TXLiveVideoEncoderParam *videoParam = [[V2TXLiveVideoEncoderParam alloc] init];
videoParam.videoResolution = V2TXLiveVideoResolution960x540;
videoParam.videoResolutionMode = landScape ? V2TXLiveVideoResolutionModeLandscape :
V2TXLiveVideoResolutionModePortrait;
[livePusher setVideoQuality:videoParam];
```

7. 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
[livePusher setWatermark:image x:0 y:0 scale:1];
```

8. 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。

```
// 停止推流
[livePusher stopScreenCapture];
[livePusher stopPush];
```

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。
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(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，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：

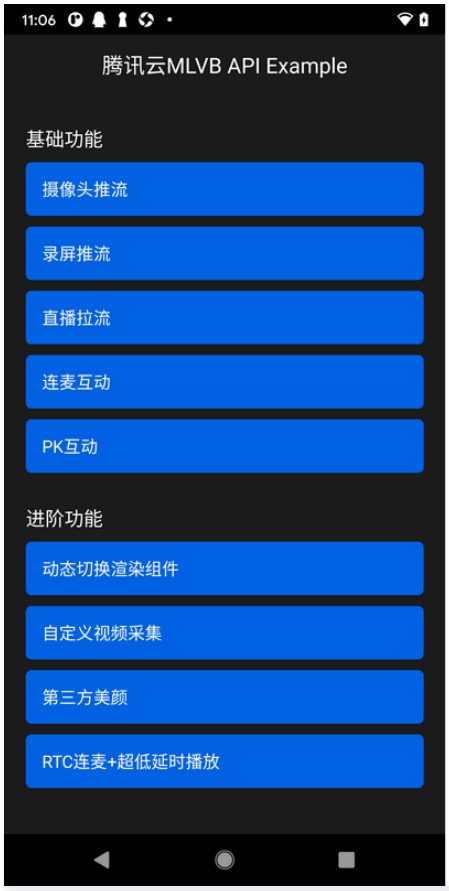
1. **屏幕录制何时会自动停止?**
   系统在锁屏或有电话打入时，会自动停止屏幕录制，此时 SampleHandler 里的 broadcastFinished 函数会被调用，可在此函数发通知提示用户。
2. **采集推流过程中有时屏幕录制会自动停止问题?**
   通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2 的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此 ReplayKit2 上建议推流分辨率不高于720P。
3. **iPhoneX 手机的兼容性与画面变形问题?**
   iPhoneX 手机因为有刘海，屏幕采集的画面分辨率不是 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。

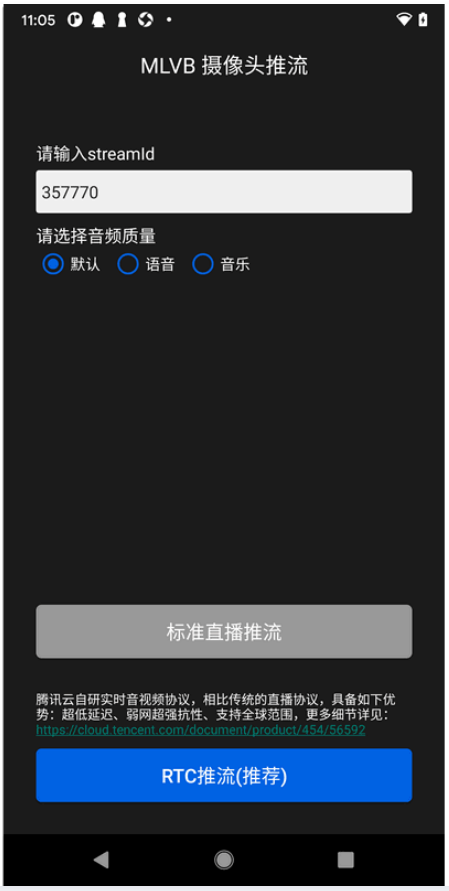
Android
摄像头推流

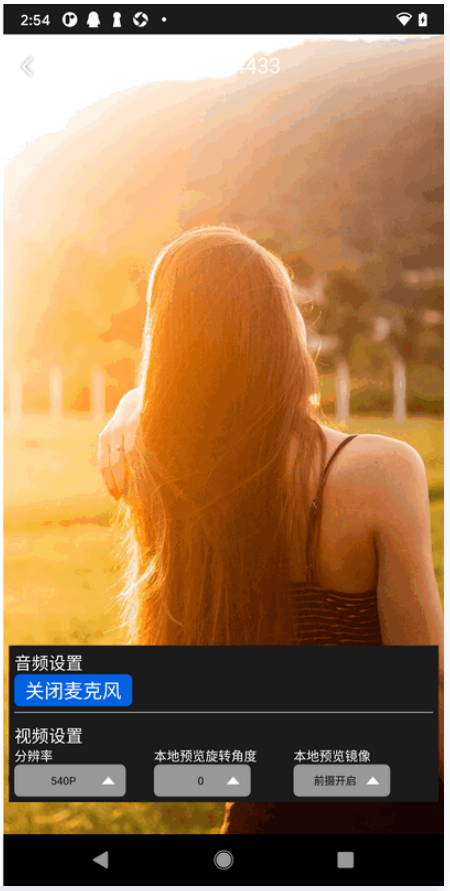
最近更新时间：2025-01-21 10:42:32

功能概述

摄像头推流，是指采集手机摄像头的画面以及麦克风的声音，进行编码之后再推送到直播云平台上。腾讯云 LiteAVSDK 通过 `V2TXLivePusher` 接口提供摄像头推流能力，如下是 SDK API-Example 工程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	LivePushCameraViewController.m
Android	Github	LivePushCameraActivity.java
Flutter	Github	live_camera_push.dart

功能对接

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

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

   - 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。
2. 在您的 App 调用 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
 V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
 V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
         @Override
         public void onLicenceLoaded(int result, String reason) {
             Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
         }
     });
}
```

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

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。

```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
```

```
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this,
V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP);
// 指定对应的直播协议为 RTC，该协议支持连麦。如果在直播过程中有连麦需求，需要选择该协议
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this,
V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
```

说明

如果您有连麦需求，需要创建 `V2TXLivePusher` 对象时选择 `RTC` 协议。 `RTC` 协议与 `RTMP` 协议在 API 使用上没有区别。

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);
mLivePusher.startMicrophone();
```

5. 启动和结束推流

1. 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。

说明

如果在 第3步 中选择 `RTMP` 协议推流，推流地址的生成请参见 RTMP 地址 。
如果在 第3步 中选择 `RTC` 协议推流，推流地址的生成请参见 RTC 地址 。

```
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
String url = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
String url = "trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxxxx";
int ret = mLivePusher.startPush(url);
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
 Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

注意

> **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 中设置的 url 和 key。

2. 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();
```

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 中打开摄像头的操作，仅仅打开麦克风即可。

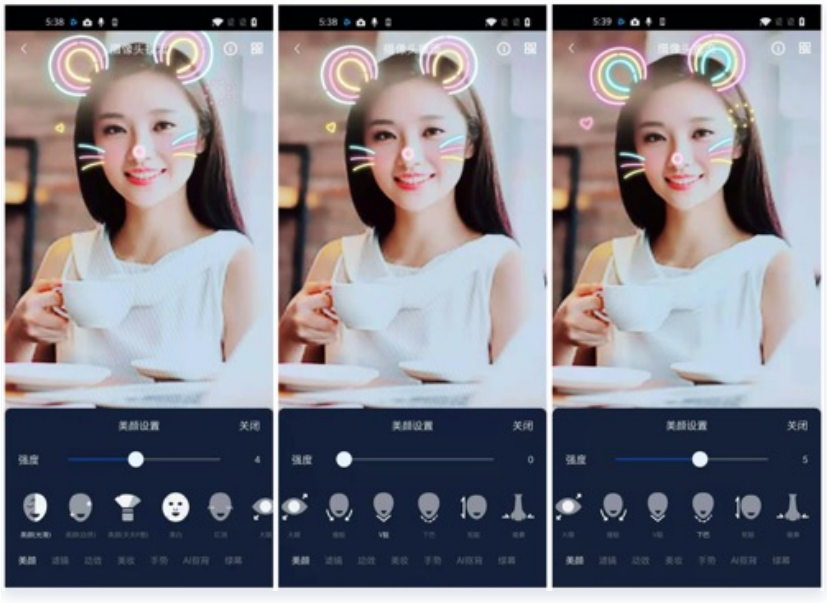
```
mLivePusher.startMicrophone();
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
String url = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
String url = "trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxxxx";
int ret = mLivePusher.startPush(url);
```

> **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

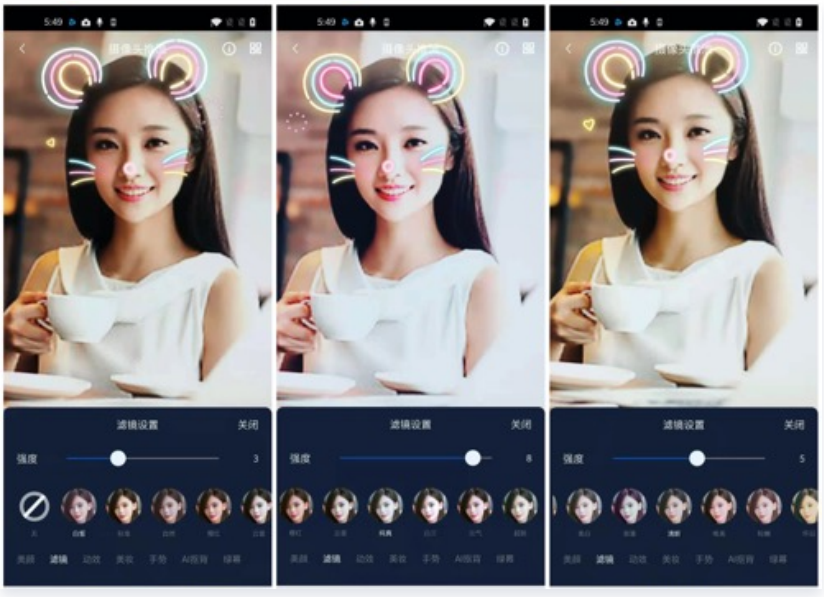
SDK 内置三种不同的磨皮算法，每种磨皮算法即对应一种美颜风格，您可以选择最适合您产品定位的方案。定义见 `TXLiveConstants.java` 文件：

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

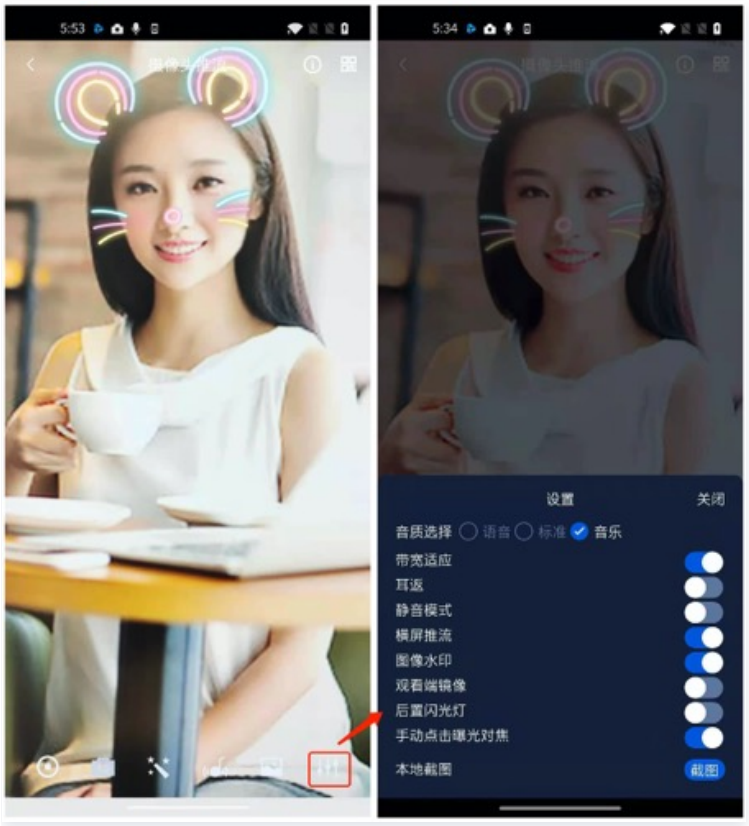


- 调用 V2TXLivePusher 中的 getBeautyManager 接口可以获取 TXBeautyManager 实例进一步设置美色彩滤镜效果。
- 调用 TXBeautyManager 的 `setFilter` 接口可以设置色彩滤镜效果。所谓色彩滤镜，是指一种将整个画面色调进行区域性调整的技术，例如将画面中的淡黄色区域淡化实现肤色亮白的效果，或者将整个画面的色彩调暖让视频的效果更加清新和温和。我们的设计师团队提供了17种默认的 色彩滤镜 供您使用。
- 调用 TXBeautyManager 的 `setFilterStrength` 接口可以设定滤镜的浓度，设置的浓度越高，滤镜效果也就越明显。

```
// 选择期望的色彩滤镜文件，滤镜文件可以在小直播 App 的资源文件中获取（以 tuibeauty_filter_ 开头的 .png 文件）。
Bitmap filterBmp = decodeResource(getResources(), R.drawable.tuibeauty_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 setEncoderMirror 可以改变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：

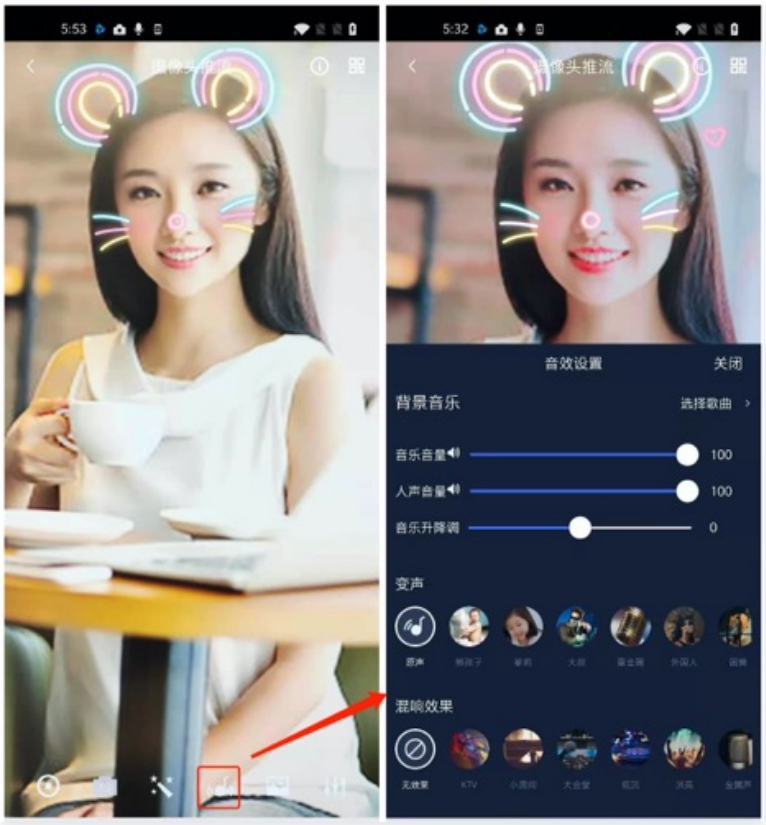
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 `setVideoQuality` 接口的参数来设定观众端的画面横竖屏模式。

```
V2TXLiveDef.V2TXLiveVideoEncoderParam param = new
V2TXLiveDef.V2TXLiveVideoEncoderParam(V2TXLiveDef.V2TXLiveVideoResolution.V2TXLiveVideoReso
```

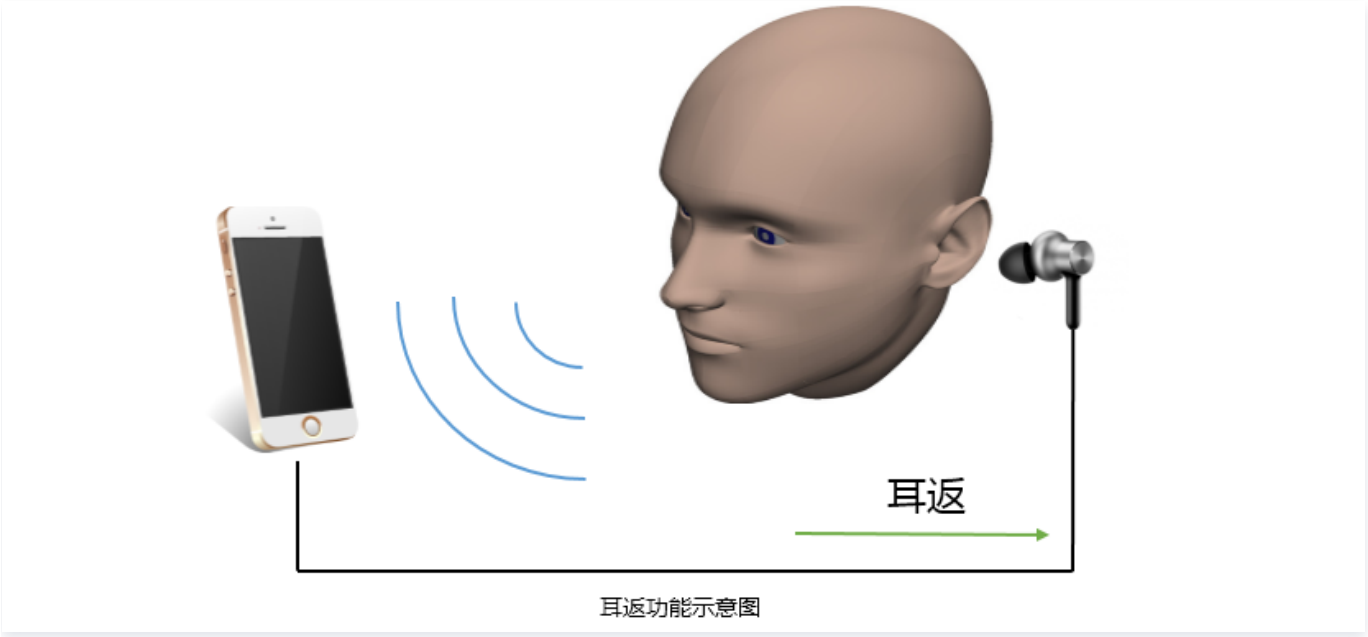
```
lution960x540);
param.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionModeLandscape :
V2TXLiveVideoResolutionModePortrait;
mLivePusher.setVideoQuality(param);
```

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”指的是当主播戴上耳机来唱歌时，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
    }
}
```

16. 发送 SEI 消息

调用 V2TXLivePusher 中的 sendSeiMessage 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：

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

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
int payloadType = 5;
String msg = "test";
mTXLivePusher.sendSeiMessage(payloadType, 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 V2TXLivePlayer 才能解析这些消息：

1. 设置：

```
int payloadType = 5;
mTXLivePlayer.enableReceiveSeiMessage(true, payloadType)
```

2. 当 V2TXLivePlayer 所播放的视频流中有 SEI 消息时，会通过 V2TXLivePlayerObserver 中的 onReceiveSeiMessage 回调来接收该消息。

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_PERMISSIO	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限

N		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

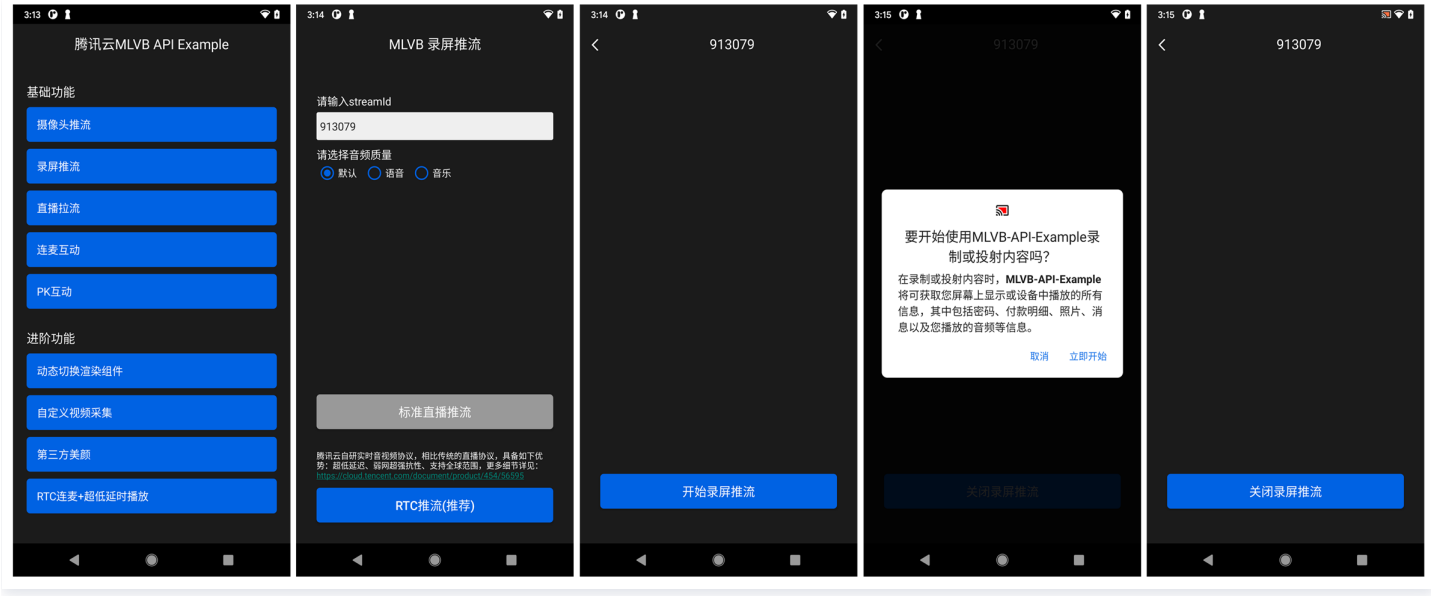
录屏推流

最近更新时间：2025-01-21 10:42:32

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 SDK API-Example 工程中演示录屏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

所属平台	GitHub 地址
iOS	LivePushScreenViewController.m
Android	LivePushScreenActivity.java
Flutter	live_screen_push.dart

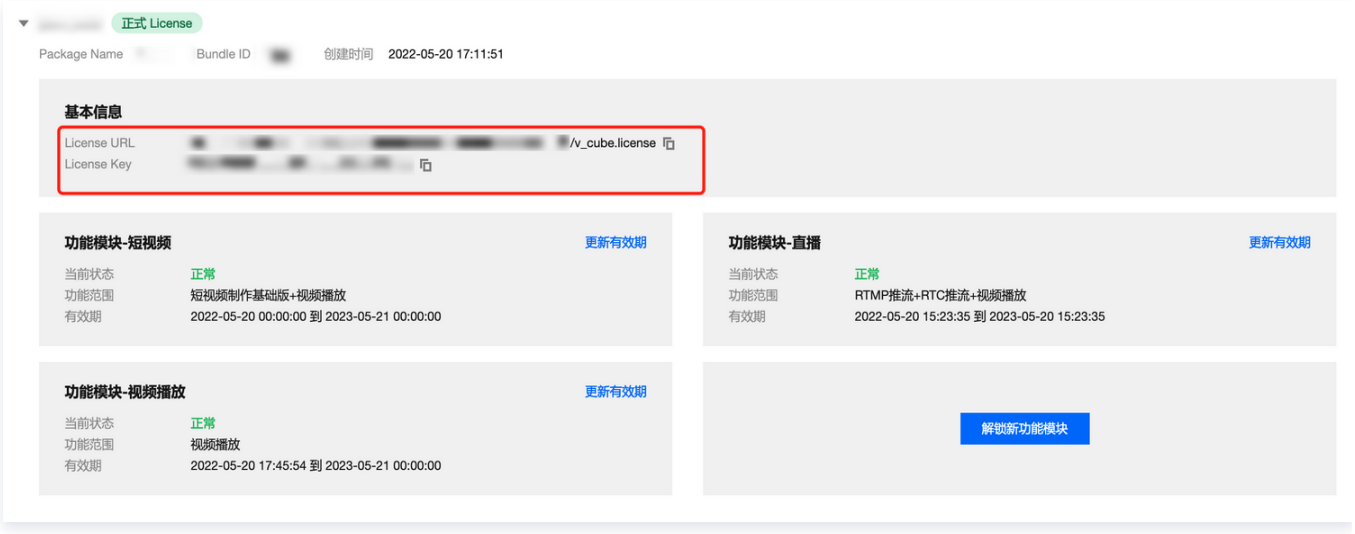
对接攻略

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

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



   - 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。
2. 在您的 App 调用 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
V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
        @Override
        public void onLicenceLoaded(int result, String reason) {
            Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
        }
    });
}
```

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

3. 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"/>
```

4. 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。

```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this,
V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTMP);
// 指定对应的直播协议为 RTC，该协议支持连麦。如果在直播过程中有连麦需求，需要选择该协议
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this,
V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
```

说明

如果您有连麦需求，需要创建 `V2TXLivePusher` 对象时选择 `RTC` 协议。 `RTC` 协议与 `RTMP` 协议在 API 使用上没有区别。

5. 启动推流

调用 startScreenCapture 启动屏幕录制，然后调用 V2TXLivePusher 中的 startPush 接口开始推流：

说明

如果在 第4步 中选择 `RTMP` 协议推流，推流地址的生成请参见 RTMP 地址 。
如果在 第4步 中选择 `RTC` 协议推流，推流地址的生成请参见 RTC 地址 。

```
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
String url = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
// 根据推流协议传入对应的 URL 即可启动推流， RTC 协议以 trtc:// 开头，该协议支持连麦
String url = "trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxxxx";
mLivePusher.startMicrophone();
mLivePusher.startScreenCapture();
int ret = mLivePusher.startPush(url);
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
    Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

注意

- **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 中设置的 url 和 key。
- **startScreenCapture** 的作用是启动屏幕录制，由于录屏是基于 Android 系统的原生能力实现的，出于安全考虑，Android 系统会在开始录屏前弹出提示，允许即可。

6. 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);
```

7. 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 。

8. 提醒主播“网络不好”

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
    }
}
```

9. 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 : V2TXLiveVideoResolutionModePortrait);
```

10. 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
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_STAR	-1308	开始录屏失败，如果在移动设备出现，可能是权限被

T_FAILED		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：2024-11-21 10:44:02

<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）： 1.《信息网络传播视听节目许可证》 2.《网络文化经营许可证》（经营范围含网络表演）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 **注意：** 1. 如提供时政信息服务，需补充：时政信息类目 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）： 1.《事业单位法人证书》（适用公立学校） 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） 3.《信息网络传播视听节目许可证》 4. 全国校外线上培训管理服务平台备案 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）： 1. 卫生健康部门的《设置医疗机构批准书》； 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）： 1.《金融许可证》 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）： 1.《金融许可证》 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）： 1.《经营证券期货业务许可证》且业务范围必须包含“基金” 2.《基金托管业务许可证》 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）： 1.《私募基金备案证明》 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 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）： 1.《证券投资咨询业务资格证书》 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）： 1.《保险公司法人许可证》 2.《经营保险业务许可证》 3.《保险营销服务许可证》 4.《经营保险代理业务许可证》 5.《经营保险经纪业务许可证》 6.《经营保险公估业务许可证》 7.《经营保险资产管理业务许可证》 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）： 1. 经营个人征信业务：《个人征信业务经营许可证》、《营业执照》 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 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）： 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》 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》 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 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
政府主体账号	–	–	–	政府相关工作推广直播、领导讲话直播等
IT 科技	音视频设备	《中国国家强制性产品认证证书》	为多方提供电话会议/视频会议等服务；智能家居场景下控制摄像头	适用于提供音视频设备、信息技术设备、电信终端设备等线下硬件在线销售及服务。
	多方通信	《增值电信业务经营许可证》（业务范围需含"国内多方通信服务业务"）		仅适用于为多方提供电话会议/视频会议等服务
房地产服务	房地产营销	（3选1）： 1.《商品房预售许可证》 2.《商品房销售许可证》 3.《商品房现售备案证明》	房地产营销直播服务、在线音视频带看等	适用于房地产开发商提供购房意向款、优惠券、权益券等营销活动

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**开发管理>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
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
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

参数	说明
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!')
        }
});
```

美颜特效

<live-pusher> 标签从微信8.0.31版本开始支持接入 Web 美颜特效，详情请参考 结合云直播的小程序推流美颜 。

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。

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

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

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）
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

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：2024-11-29 10:06:53

云直播提供了推流 SDK `TXLivePusher` 用于 Web 推流，负责将浏览器采集的音视频画面通过 WebRTC 协议推送到直播服务器。目前支持摄像头采集、麦克风采集、屏幕分享采集、本地媒体文件采集和用户自定义采集等采集方式，支持对采集到的内容进行本地混流处理，然后推送到后端服务器。
您可以在 Web 端进行 WebRTC 协议推流，在 PC 端，您还可以使用 OBS 工具进行 WebRTC 推流，具体操作方法请参考 OBS WebRTC 推流 相关内容。
使用 Web 进行 WebRTC 推流的优点是无需安装额外的软件，只需在浏览器中操作即可。本文将介绍采用 **Web** 进行推流的操作方式。

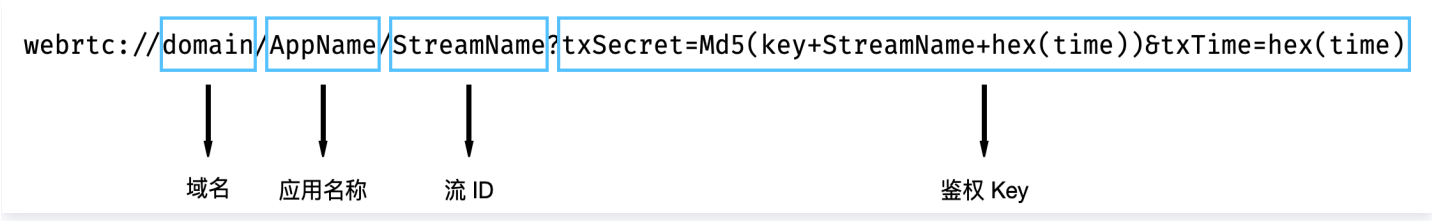
> ⚠ **注意：**
> 使用 WebRTC 协议推流，每个推流域名默认限制**1000路并发**推流数，如您需要超过此推流限制，可通过 提交工单 的方式联系我们进行申请。

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

Web 推流基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持，目前版本的 Chrome、Edge、Firefox 和 Safari 浏览器都支持 Web 推流。

> ⚠ **注意：**
> 浏览器采集音视频画面的部分功能在移动端 H5 受到限制，例如移动端浏览器不支持屏幕分享，iOS 14.3 及以上版本才支持获取用户摄像头设备。

对接攻略

步骤1：页面准备工作

在需要直播推流的页面中引入初始化脚本。

```
<script src="https://video.sdk.qcloudecdn.com/web/TXLivePusher-2.1.1.min.js" charset="utf-8"></script>
```

> **说明：**
> 需要在 HTML 的 body 部分引入脚本，如果在 head 部分引入会报错。

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 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="local_video" style="width:100%;height:500px;display:flex;align-items:center;justify-content:center;"></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。

   ```
   const 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('local_video');
   ```

   > **说明：**
   > 调用 `setRenderView` 生成的 video 元素默认有声音，如果播放从麦克风采集的声音，可能会产生回声现象。可以将 video 元素进行静音，避免回声现象的出现。
   >
   > ```
   > livePusher.videoView.muted = true;
   > ```

3. **设置音视频采集质量：**
   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、屏幕分享、本地媒体文件和自定义的流。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. **开始直播推流：**

传入云直播推流地址，开始推流。推流地址的格式参考 拼装推流 URL 。

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

> **说明：**
> 推流之前要确保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，可以等待视频流和音频流采集成功之后，再进行推流。

```
// 采集完摄像头画面后自动推流
livePusher.startCamera()
.then(function () {
 livePusher.startPush('webrtc://domain/AppName/StreamName?
txSecret=xxx&txTime=xxx');
})
.catch(function (error) {
 console.log('打开摄像头失败: '+ error.toString());
});

// 采集完摄像头和麦克风之后自动推流
Promise.all([livePusher.startCamera(), livePusher.startMicrophone()])
.then(function() {
 livePusher.startPush('webrtc://domain/AppName/StreamName?
txSecret=xxx&txTime=xxx');
});
```

6. **停止直播推流：**

```
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

检测浏览器兼容性

SDK 提供静态方法 checkSupport() 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，可以通过设置回调事件来了解 SDK 内部的状态信息和 WebRTC 相关的数据统计。具体内容请参考接口 setObserver() 。

```
livePusher.setObserver({
  // 推流连接状态
  onPushStatusUpdate: function(status, message) {
    console.log(status, message);
  },
  // 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 TXDeviceManager 帮助用户进行获取设备列表、切换设备等操作。

```
const deviceManager = livePusher.getDeviceManager();
let cameraDeviceId = null;

// 获取设备列表
deviceManager.getDevicesList().then(function(data) {
  data.forEach(function(device) {
    console.log(device.type, device.deviceId, device.deviceName);

    if (device.type === 'video') {
      cameraDeviceId = device.deviceId;
    }
  });

  // 切换摄像头设备
  if (cameraDeviceId) {
    deviceManager.switchCamera(cameraDeviceId);
  }
});
```

调整音量

SDK 提供了音频效果管理实例 TXAudioEffectManager 用来调整音频流的音量。

```
const audioEffectManager = livePusher.getAudioEffectManager();

audioEffectManager.setVolume(50);
```

视频效果

SDK 提供了视频效果管理实例 TXVideoEffectManager 用来设置视频流画中画、镜像、滤镜、水印、文本等效果。具体用法请参考 Web 端本地混流 。

API 接口协议

SDK 相关接口说明，请参考 Web 推流 SDK API 。

Flutter
摄像头推流

最近更新时间：2024-12-11 16:20:42

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 live_flutter_plugin 通过 `v2_tx_live_pusher` 接口提供摄像头推流能力。

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
Flutter	Github	live_camera_push.dart

> **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
> - Flutter：Live-API-Example

快速开始

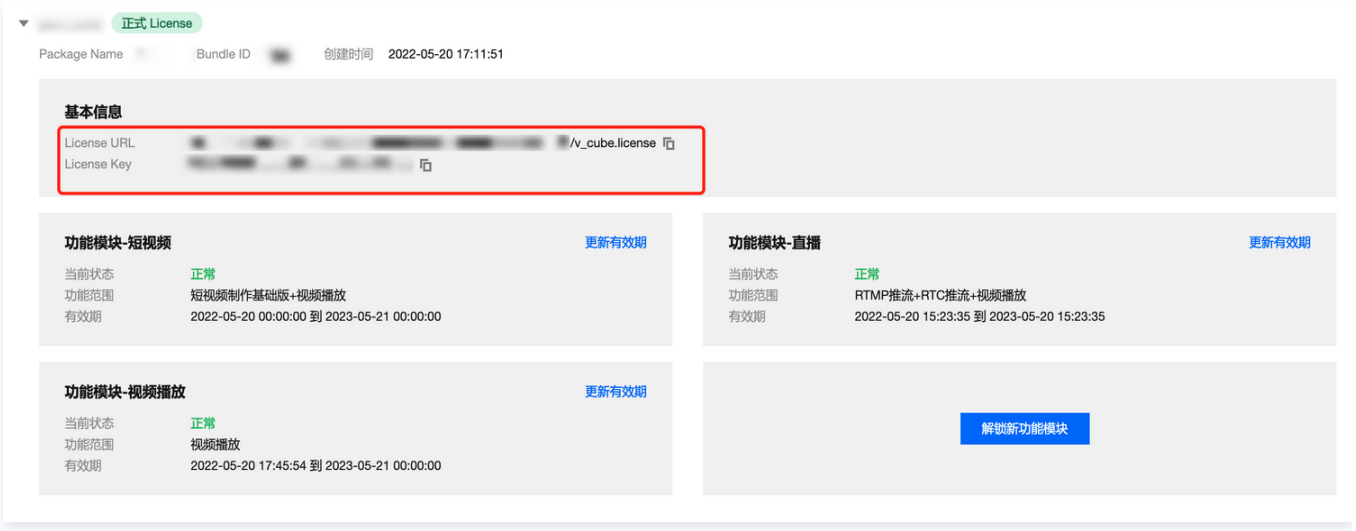
1. 设置依赖

按照 SDK 集成指引 将 live_flutter_plugin 嵌入您的 App 工程中。

```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

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



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

2. 在您的 App 调用 live_flutter_plugin 的相关功能之前进行如下设置：

```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.cloud.tencent.com/live/license)
setupLicense() {
  // 当前应用的License LicenseUrl
  var LICENSEURL = "";
  // 当前应用的License Key
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

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

3. 初始化 V2TXLivePusher 组件

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

```
import 'package:live_flutter_plugin/v2_tx_live_pusher.dart';

/// V2TXLivePusher 初始化
initPusher() {
  _livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTMP);
}
```

4. 设置视频渲染 RenderView

```
import 'package:live_flutter_plugin/widget/v2_tx_live_video_widget.dart';

/// 视频渲染View Widget
```

```
Widget renderView() {
  return V2TXLiveVideoWidget(
    onViewCreated: (viewId) async {
      /// 设置视频渲染View
      _livePusher.setRenderViewID(_renderViewId);
      /// 开启摄像头预览
      _livePusher.startCamera(true);
    },
  );
}
```

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。推流地址可以使用 TRTC 地址 ，或者使用 RTMP 地址 ，前者使用 UDP 协议，推流质量更高，并支持连麦互动。

```
/// 开始推流
startPush() async {
  // 生成推流地址 RTMP/TRTC
  var url = "";
  // 开始推流
  await _livePusher.startPush(url);
    // 打开麦克风
  await _livePusher.startMicrophone();
}
```

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

```
/// 停止推流
stopPush() async {
  // 关闭摄像头
  await _livePusher.stopCamera();
  // 关闭麦克风
  await _livePusher.stopMicrophone();
  // 停止推流
  await _livePusher.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。

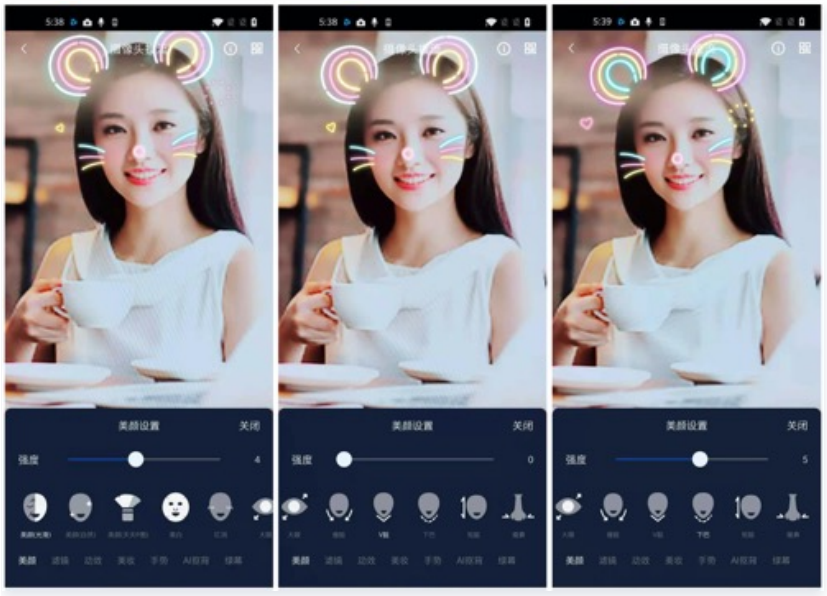
```
/// 开始推流
startPush() async {
  // 初始化V2TXLivePusher
  _livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTMP);
  // 生成推流地址 RTMP/TRTC
  var url = "";
  // 开始推流
  await _livePusher.startPush(url);
    // 打开麦克风
  await _livePusher.startMicrophone();
}
```

> **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：

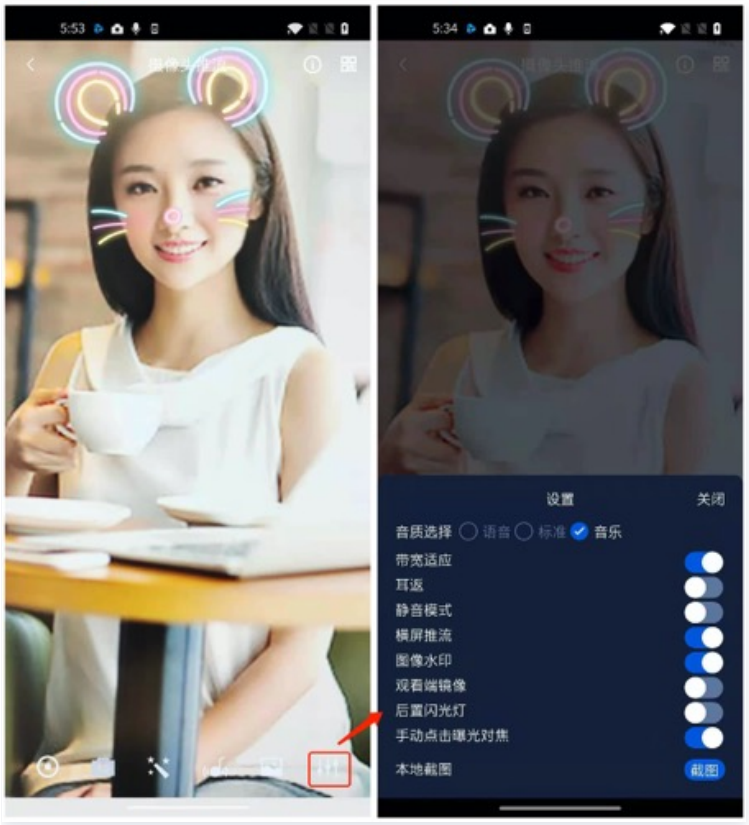
美颜风格	效果说明
TXBeautyStyleSmooth	光滑，适用于美女秀场，效果比较明显
TXBeautyStyleNature	自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然
TXBeautyStylePitu	由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高

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. 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 。



10. 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。

11. 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：

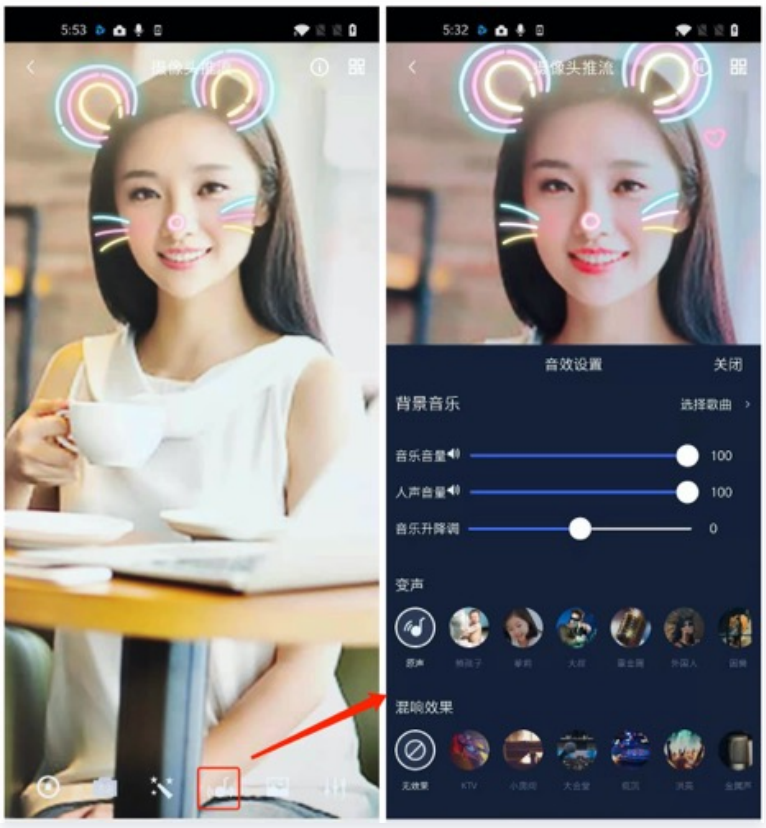
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。

```
// 视频编码参数
var param = V2TXLiveVideoEncoderParam();
param.videoResolutionMode = isLandscape ?
V2TXLiveVideoResolutionMode.v2TXLiveVideoResolutionModeLandscape :
V2TXLiveVideoResolutionMode.v2TXLiveVideoResolutionModePortrait;
_livePusher.setVideoQuality(param);
```

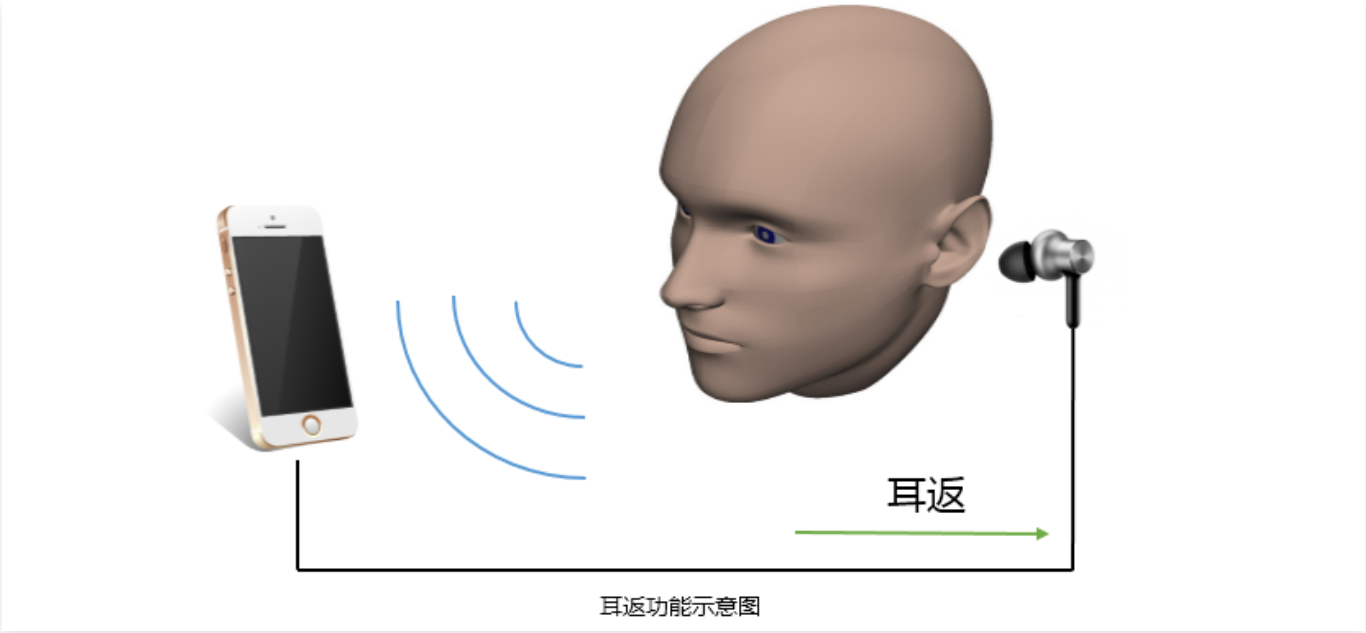
12. 音效设置

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 。

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
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_STA	-1308	开始录屏失败，如果在移动设备出现，可能是权限被

RT_FAILED		用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断

录屏推流

最近更新时间：2023-09-19 18:43:52

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 live_flutter_plugin 通过 `V2TXLivePusher` 接口提供摄像头推流能力。

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
Flutter	Github

开发环境准备

Android

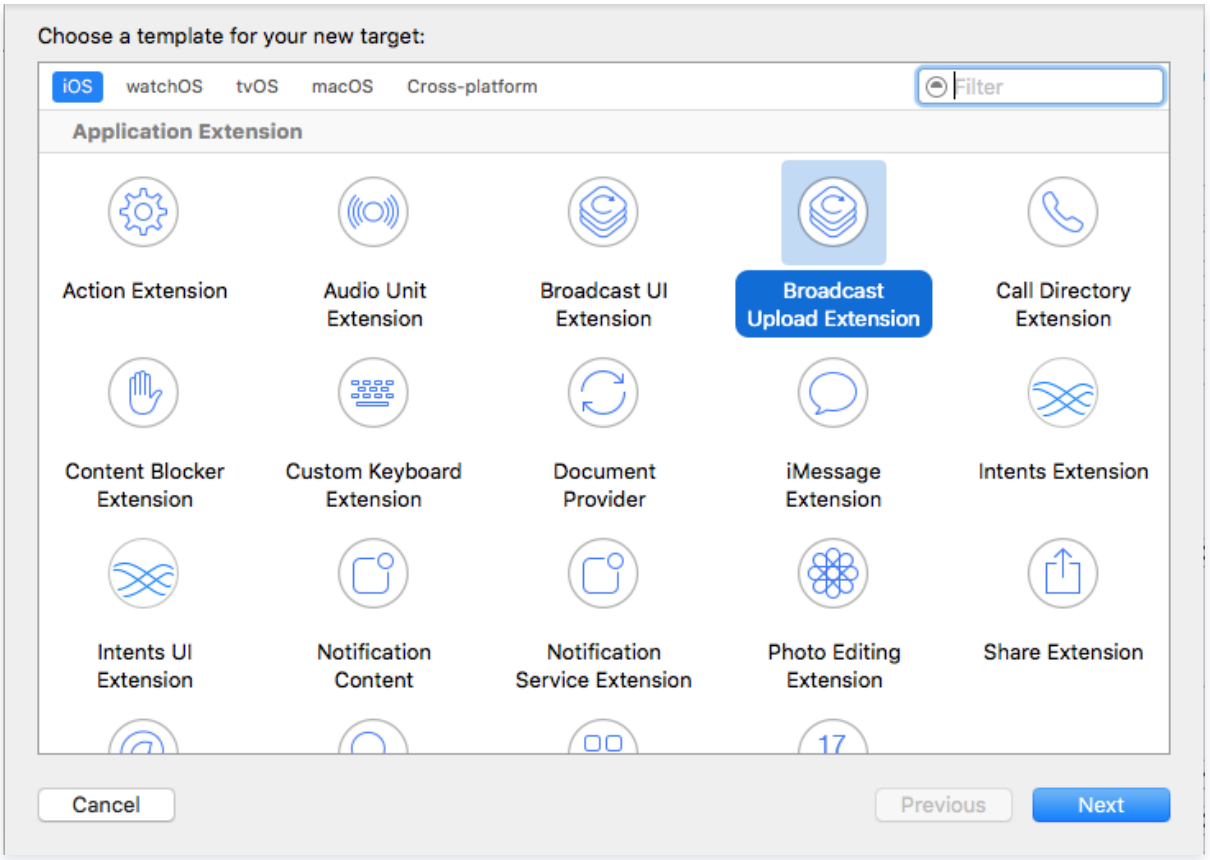
- Android 5.0 系统以后开始支持录屏功能。
- 悬浮窗在部分手机和系统上需要通过手动设置打开。

iOS

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。更详细的使用说明可以参考 Demo 里 Live Demo Screen 文件夹示例代码。

1. **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。

> ⚠ **注意**
> 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。

2. **导入TXLiteAVSDK_ReplayKitExt.framework**
直播扩展需要导入 TXLiteAVSDK_ReplayKitExt.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）。

快速开始

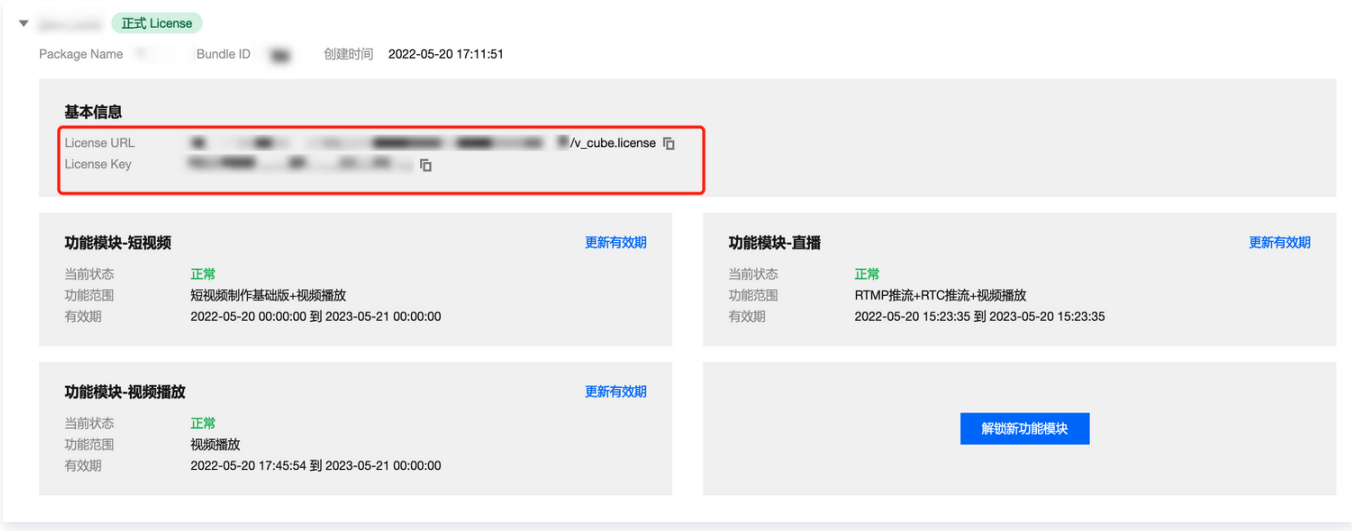
1. 设置依赖

按照 SDK 集成指引 将 live_flutter_plugin 嵌入您的 App 工程中。

```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

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



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

2. 在您的 App 调用 live_flutter_plugin 的相关功能之前进行如下设置：

```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.cloud.tencent.com/live/license)
setupLicense() {
  // 当前应用的License LicenseUrl
  var LICENSEURL = "";
  // 当前应用的License Key
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

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

3. 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 livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTMP);
```

4. 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";
livePusher.startMicrophone();
livePusher.startScreenCapture();
livePusher.startPush(rtmpUrl);
```

- **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。

5. 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
void stopPush() {
    livePusher.stopMicrophone();
    livePusher.stopScreenCapture();
    livePusher.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
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

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. **屏幕录制何时会自动停止?**
   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。

3. 直播播放
iOS
标准直播拉流

最近更新时间：2024-12-02 18:02:32

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
Flutter	Github

对接攻略

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

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

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

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

3. 创建 Player

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

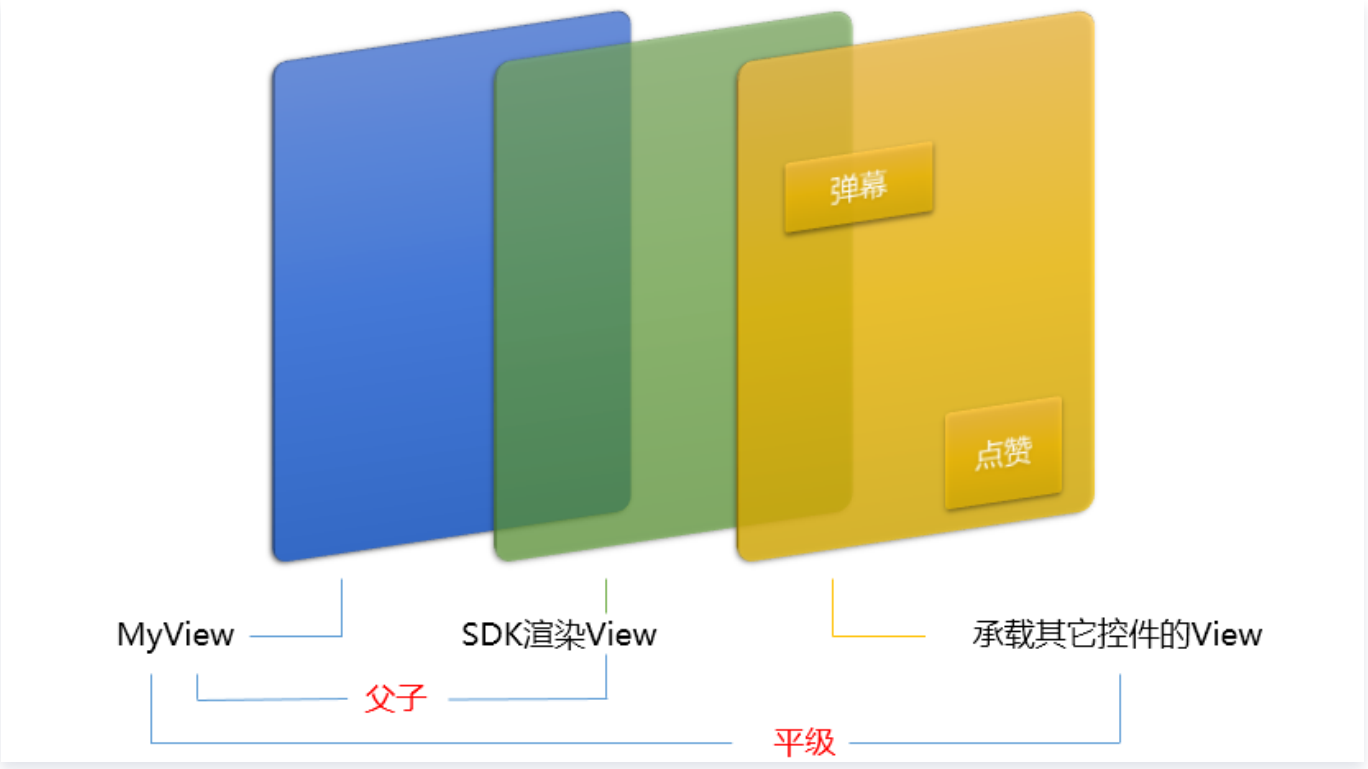
```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
```

4. 渲染 View

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

```
//用 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
}];
```

5. 启动播放

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

6. 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

7. 暂停播放

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

```
// 暂停
[_txLivePlayer pauseAudio];
[_txLivePlayer pauseVideo];
// 恢复
[_txLivePlayer resumeAudio];
[_txLivePlayer resumeVideo];
```

8. 结束播放

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

9. 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

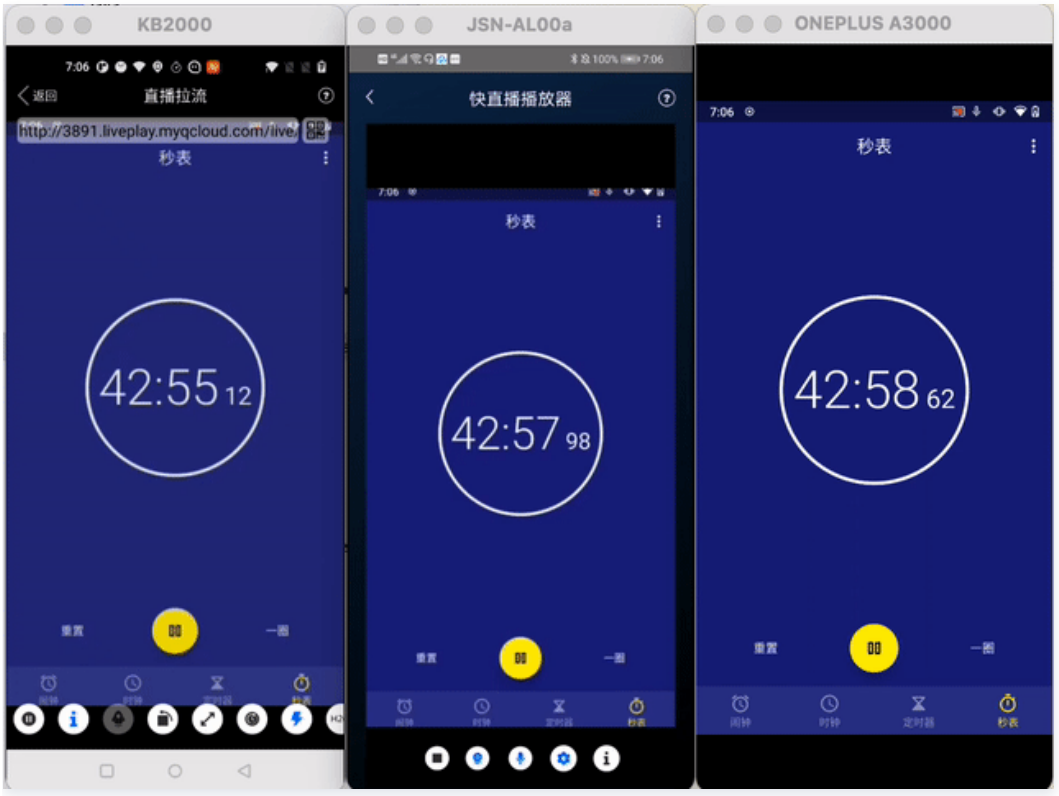
其余的回调仅在事件发生时才会抛出来。

快直播拉流

最近更新时间：2024-11-28 17:05:52

快直播概述

快直播（Live Event Broadcasting，LEB）是标准直播在超低延时播放场景下的延伸，比传统直播协议延时更低，为观众提供毫秒级的流畅直播观看体验。 能够满足一些对延时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
毫秒级超低延时播放	采用 UDP 协议将传统直播中3秒 – 5秒延时降低至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时直播体验。
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时的直播能力加持，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，快直播的超低延时能力，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时能力，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时，观众端有时需要进行补帧才能让观众主持两端同时显示。快直播的超低延时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）、直播 SDK、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	升级维护中		

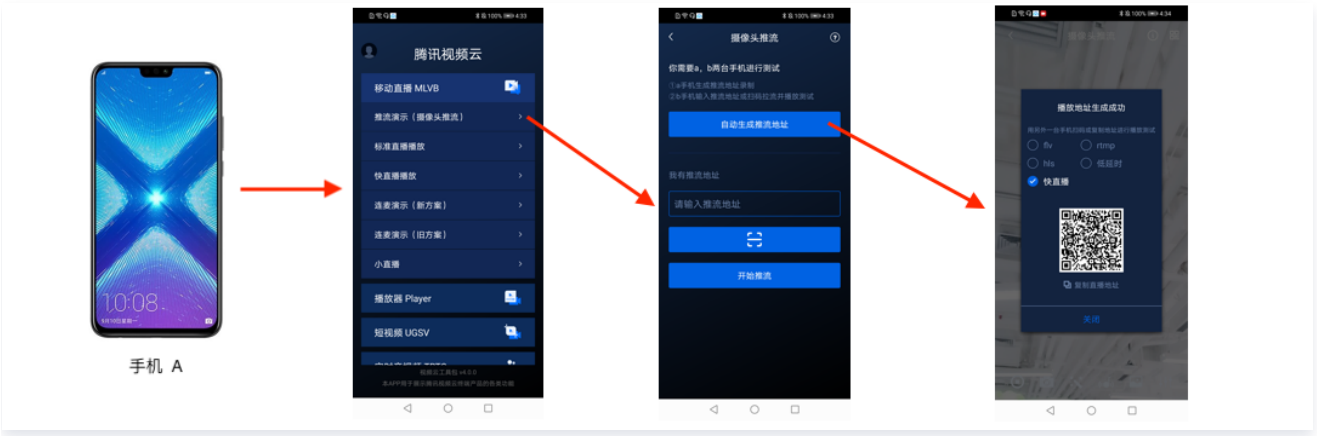
说明：
除上述示例外，针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

- iOS：MLVB-API-Example
- Android：MLVB-API-Example
- Flutter：Live-API-Example

推流体验

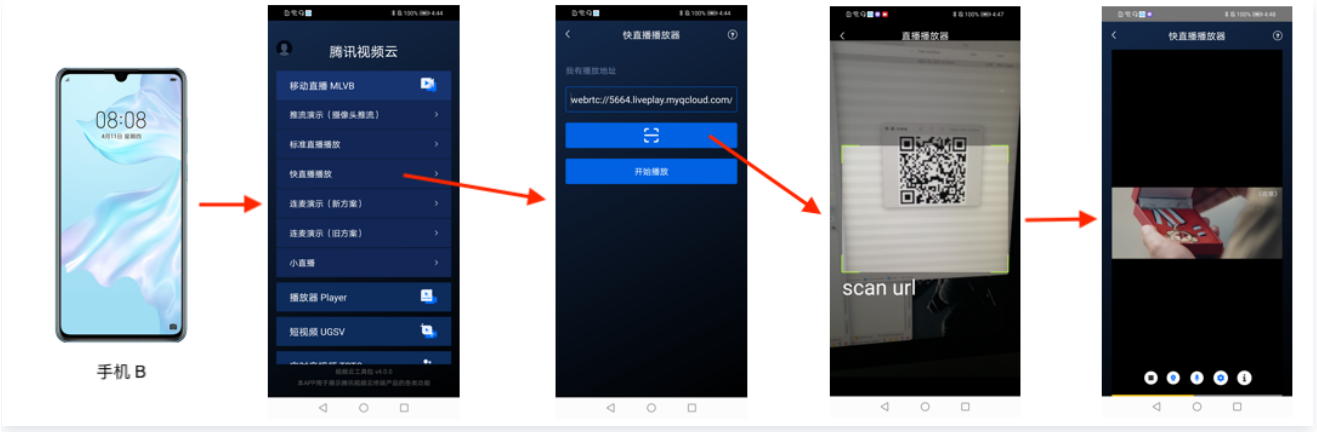
快直播兼容了标准直播，因此可以使用普通推流端进行推流，然后使用快直播进行拉流。

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。

1. 下载 SDK

可以在 SDK 下载 页面选择全功能版下载。

2. 给 SDK 配置 License 授权

1. 获取 License 授权：

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

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

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

Android

```
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
    V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
            }
        });
}
```

iOS

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
```

```
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

3. 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 。

4. 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.startLivePlay("webrtc://{Domain}/{AppName}/{StreamName}");
```

iOS

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

Android
标准直播拉流

最近更新时间：2023-09-19 18:43:53

基础知识

本文主要介绍视频云 SDK 的直播播放功能。

直播和点播

- **直播（LIVE）**的视频源是主播实时推送的。因此，主播停止推送后，播放端的画面也会随即停止，而且由于是实时直播，所以播放器在播直播 URL 的时候是没有进度条的。
- **点播（VOD）**的视频源是云端的一个视频文件，只要未被从云端移除，视频就可以随时播放， 播放中您可以通过进度条控制播放位置，腾讯视频和优酷、土豆等视频网站上的视频观看就是典型的点播场景。

协议的支持

通常使用的直播协议如下，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
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 和 WebRTC 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。

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
Flutter	Github

对接攻略

1. 下载 SDK 开发包

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

2. 给 SDK 配置 License 授权

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

   - 若您暂未获得 License 授权，需先参考 新增与续期License 进行申请。
2. 在您的 App 调用 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
 V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
 V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
         @Override
         public void onLicenceLoaded(int result, String reason) {
             Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
         }
     });
}
```

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

3. 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"/>
```

4. 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 step3 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);
```

5. 启动播放

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

6. 画面调整

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

7. 暂停播放

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

```
// 暂停
mLivePlayer.pauseAudio();
mLivePlayer.pauseVideo();
// 继续
mLivePlayer.resumeAudio();
mLivePlayer.resumeVideo();
```

8. 结束播放

结束播放非常简单，直接调用 `stopPlay` 即可。

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

9. 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

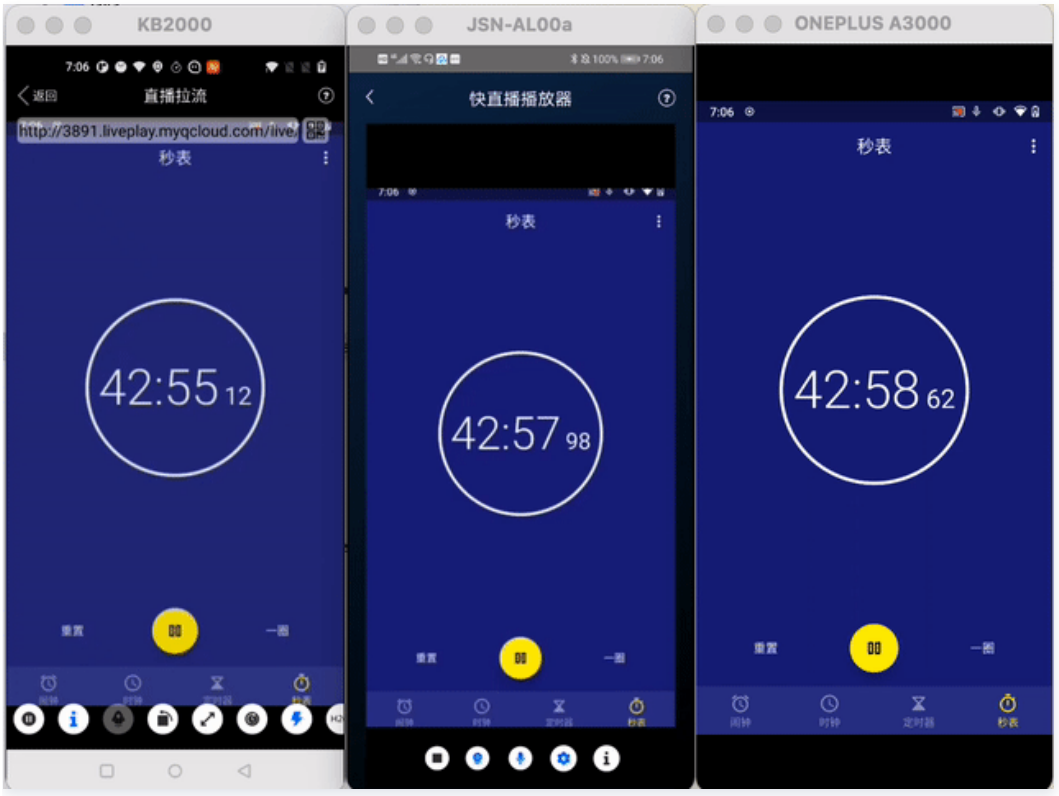
其余的回调仅在事件发生时才会抛出来。

快直播拉流

最近更新时间：2024-11-28 17:05:52

快直播概述

快直播（Live Event Broadcasting，LEB）是标准直播在超低延时播放场景下的延伸，比传统直播协议延时更低，为观众提供毫秒级的高品质直播观看体验。能够满足一些对延时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
毫秒级超低延时播放	采用 UDP 协议将传统直播中3秒 – 5秒延时降低至1秒以内，同时兼顾秒开、卡顿率等核心指标，给用户带来高品质的超低延时直播体验。
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时的直播能力加持，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，快直播的超低延时能力，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时能力，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时，观众端有时需要进行补帧才能让观众主持两端同时显示。快直播的超低延时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）、直播 SDK、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	升级维护中		

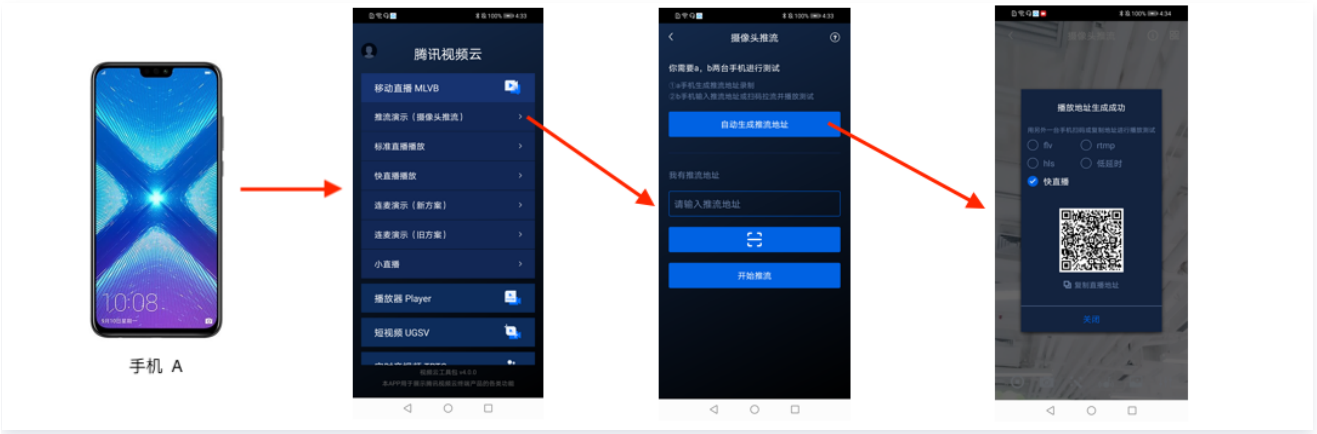
说明：
除上述示例外，针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

- iOS：MLVB-API-Example
- Android：MLVB-API-Example
- Flutter：Live-API-Example

推流体验

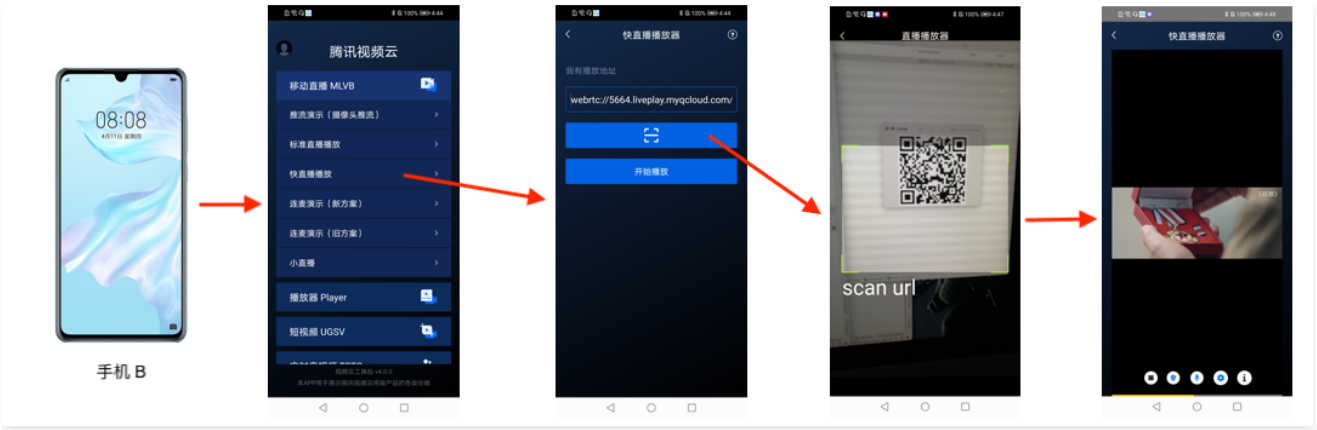
快直播兼容了标准直播，因此可以使用普通推流端进行推流，然后使用快直播进行拉流。

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。

1. 下载 SDK

可以在 SDK 下载 页面选择全功能版下载。

2. 给 SDK 配置 License 授权

1. 获取 License 授权：

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

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

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

Android

```
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
    V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
            }
        });
}
```

iOS

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

    // V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
```

```
}

#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

3. 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 。

4. 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.startLivePlay("webrtc://{Domain}/{AppName}/{StreamName}");
```

iOS

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

微信小程序

最近更新时间：2024-11-21 10:44:02

<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）： 1.《信息网络传播视听节目许可证》 2.《网络文化经营许可证》（经营范围含网络表演）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 **注意：** 1. 如提供时政信息服务，需补充：时政信息类目 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）： 1.《事业单位法人证书》（适用公立学校） 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） 3.《信息网络传播视听节目许可证》 4. 全国校外线上培训管理服务平台备案 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）： 1. 卫生健康部门的《设置医疗机构批准书》； 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）： 1.《金融许可证》 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）： 1.《金融许可证》 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）： 1.《经营证券期货业务许可证》且业务范围必须包含“基金” 2.《基金托管业务许可证》 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）： 1.《私募基金备案证明》 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 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）： 1.《证券投资咨询业务资格证书》 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）： 1.《保险公司法人许可证》 2.《经营保险业务许可证》 3.《保险营销服务许可证》 4.《经营保险代理业务许可证》 5.《经营保险经纪业务许可证》 6.《经营保险公估业务许可证》 7.《经营保险资产管理业务许可证》 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）： 1. 经营个人征信业务：《个人征信业务经营许可证》、《营业执照》 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 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）： 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》 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》 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 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
政府主体账号	–	–	–	政府相关工作推广直播、领导讲话直播等
IT 科技	音视频设备	《中国国家强制性产品认证证书》	为多方提供电话会议/视频会议等服务；智能家居场景下控制摄像头	适用于提供音视频设备、信息技术设备、电信终端设备等线下硬件在线销售及服务。
	多方通信	《增值电信业务经营许可证》（业务范围需含"国内多方通信服务业务"）		仅适用于为多方提供电话会议/视频会议等服务
房地产服务	房地产营销	（3选1）： 1.《商品房预售许可证》 2.《商品房销售许可证》 3.《商品房现售备案证明》	房地产营销直播服务、在线音视频带看等	适用于房地产开发商提供购房意向款、优惠券、权益券等营销活动

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
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&txSecret=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

参数	说明
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。

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
-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。

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）。

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

- <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。
- 请勿在 <scroll-view> 中使用 <live-player> 组件。
- <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
Web（TCPlayer）

最近更新时间：2024-03-04 10:00:31

本文档将介绍适用于点播播放和直播播放的 Web 播放器 SDK（TCPlayer），它可快速与自有 Web 应用集成，实现视频播放功能。Web 播放器 SDK（TCPlayer）内默认包含部分 UI，您可按需取用。

概述

Web 播放器是通过 HTML5 的 `<video>` 标签以及 Flash 实现视频播放。在浏览器不支持视频播放的情况下，实现了视频播放效果的多平台统一体验，并结合腾讯云点播视频服务，提供防盗链和播放 HLS 普通加密视频等功能。

协议支持

音视频协议	用途	URL 地址格式	PC 浏览器	移动浏览器
MP3	音频	http://xxx.vod.myqcloud.com/xxx.mp3	支持	支持
MP4	点播	http://xxx.vod.myqcloud.com/xxx.mp4	支持	支持
HLS（M3U8）	直播	http://xxx.liveplay.myqcloud.com/xxx.m3u8	支持	支持
	点播	http://xxx.vod.myqcloud.com/xxx.m3u8	支持	支持
FLV	直播	http://xxx.liveplay.myqcloud.com/xxx.flv	支持	部分支持
	点播	http://xxx.vod.myqcloud.com/xxx.flv	支持	部分支持
WebRTC	直播	webrtc://xxx.liveplay.myqcloud.com/live/xxx	支持	支持

说明：

- 视频编码格式仅支持 H.264 编码。
- 播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。
- HLS、FLV 视频在部分浏览器环境播放需要依赖 Media Source Extensions。
- 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。

功能支持

功能\浏览器	Chrome	Firefox	Edge	QQ 浏览器	Mac Safari	iOS Safari	微信	Android Chrome	IE 11
播放器尺寸设置	✓	✓	✓	✓	✓	✓	✓	✓	✓
续播功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
倍速播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
缩略图预览	✓	✓	✓	✓	–	–	–	–	✓

切换 fileID 播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
镜像功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
进度条标记	✓	✓	✓	✓	✓	–	–	–	✓
HLS 自适应码率	✓	✓	✓	✓	✓	✓	✓	✓	✓
Referer 防盗链	✓	✓	✓	✓	✓	✓	✓	–	✓
清晰度切换提示	✓	✓	✓	✓	–	–	–	✓	✓
试看功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 标准加密播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 私有加密播放	✓	✓	✓	–	–	–	• Android：✓ • iOS: –	✓	✓
视频统计信息	✓	✓	✓	✓	–	–	–	–	–
视频数据监控	✓	✓	✓	✓	–	–	–	–	–
自定义提示文案	✓	✓	✓	✓	✓	✓	✓	✓	✓
自定义UI	✓	✓	✓	✓	✓	✓	✓	✓	✓
弹幕	✓	✓	✓	✓	✓	✓	✓	✓	✓
水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
幽灵水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
视频列表	✓	✓	✓	✓	✓	✓	✓	✓	✓
弱网追帧	✓	✓	✓	✓	✓	✓	✓	✓	✓

说明：

- 视频编码格式仅支持 H.264 编码。
- Chrome、Firefox 包括 Windows、macOS 平台。
- Chrome、Firefox、Edge 及 QQ 浏览器播放 HLS 需要加载 `hls.js` 。
- Referer 防盗链功能是基于 HTTP 请求头的 Referer 字段实现的，部分 Android 浏览器发起的 HTTP 请求不会携带 Referer 字段。

播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。例如：在 Chrome 等现代浏览器中优先使用 HTML5 技术实现视频播放，而手机浏览器上会使用 HTML5 技术或者浏览器内核能力实现视频播放。

准备工作

播放器 SDK Web端（TCPlayer）自 5.0.0 版本起需获取 License 授权后方可使用。若您无需使用新增的高级功能，可直接申请基础版 License 以**继续免费使用 TCPlayer**；若您需要使用新增的高级功能则需购买高级版 License。具体信息如下：

TCPlayer 功能	功能范围	所需 License	定价	授权单位
基础版功能	包含 5.0.0 以前版本提供的全部功能，详情见 产品功能	播放器 Web 端基础版 License	0元 免费申请	精准域名（1个 License 最多可授权10个精准域名）
高级版功能	基础版功能、VR 播放、安全检查	播放器 Web 端高级版 License	399元/月 立即购买	泛域名（1个 License 最多可授权1个泛域名）

说明：

1. 播放器 Web 端基础版 License 可免费申请，申请后有效期默认1年；在有效期剩余时间小于30天时，可免费续期。
2. 为方便本地开发，播放器不会校验 localhost 或者 127.0.0.1，因此申请 License 时不需要申请这类本地服务域名。

集成指引

通过以下步骤，您就可以在网页上添加一个视频播放器。

步骤1：在页面中引入文件

播放器 SDK 支持 cdn 和 npm 两种集成方式：

1. 通过 cdn 集成

在本地的项目工程内新建 index.html 文件，html 页面内引入播放器样式文件与脚本文件：

```
 <link href="https://web.sdk.qcloud.com/player/tcplayer/release/v5.1.0/tcplayer.min.css"
rel="stylesheet"/>
 <!--播放器脚本文件-->
 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v5.1.0/tcplayer.v5.1.0.min.js">
</script>
```

建议在使用播放器 SDK 的时候自行部署资源，单击下载播放器资源 。部署解压后的文件夹，不能调整文件夹里面的目录，避免资源互相引用异常。
如果您部署的地址为 `aaa.xxx.ccc` ，在合适的地方引入播放器样式文件与脚本文件，自行部署情况下，需要手动引用资源包文件夹 libs 下面的依赖文件，否则会默认请求腾讯云 cdn 文件。

```
 <link href="aaa.xxx.ccc/tcplayer.min.css" rel="stylesheet"/>
 <!--如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在
tcplayer.vx.x.x.min.js 之前引入 hls.min.x.xx.m.js。-->
 <script src="aaa.xxx.ccc/libs/hls.min.x.xx.m.js"></script>
 <!--播放器脚本文件-->
 <script src="aaa.xxx.ccc/tcplayer.vx.x.x.min.js"></script>
```

2. 通过 npm 集成

首先安装 tcplayer 的 npm 包：

```
npm install tcplayer.js
```

导入 SDK 和样式文件：

```
import TCPlayer from 'tcplayer.js';
import 'tcplayer.js/dist/tcplayer.min.css';
```

步骤2：放置播放器容器

在需要展示播放器的页面位置加入播放器容器。例如，在 index.html 中加入如下代码（容器 ID 以及宽高都可以自定义）。

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline webkit-playsinline>
</video>
```

说明：

- 播放器容器必须为 `<video>` 标签。
- 示例中的 `player-container-id` 为播放器容器的 ID，可自行设置。
- 播放器容器区域的尺寸，建议通过 CSS 进行设置，通过 CSS 设置比属性设置更灵活，可以实现例如铺满全屏、容器自适应等效果。
- 示例中的 `preload` 属性规定是否在页面加载后载入视频，通常为了更快的播放视频，会设置为 `auto`，其他可选值：`meta`（当页面加载后只载入元数据），`none`（当页面加载后不载入视频），移动端由于系统限制不会自动加载视频。
- `playsinline` 和 `webkit-playsinline` 这几个属性是为了在标准移动端浏览器不劫持视频播放的情况下实现行内播放，此处仅作示例，请按需使用。
- 设置 `x5-playsinline` 属性在 TBS 内核会使用 X5 UI 的播放器。

步骤3：播放器初始化

页面初始化后，即可播放视频资源。播放器同时支持点播和直播两种播放场景，具体播放方式如下：

- 点播播放：播放器可以通过 FileID 播放腾讯云点播媒体资源，云点播具体流程请参见 使用播放器播放 文档。
- 直播播放：播放器通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播 URL 生成方式可参见 自主拼装直播 URL 。

通过 URL 播放（直播、点播）

在页面初始化之后，调用播放器实例上的方法，将 URL 地址传入方法。

```
// player-container-id 为播放器容器 ID，必须与 html 中一致
var player = TCPlayer('player-container-id', {
    sources: [{
      src: 'path/to/video', // 播放地址
    }],
    licenseUrl: 'license/url', // license 地址，参考准备工作部分，在视立方控制台申请 license 后可获得 licenseUrl
});
```

```
// player.src(url); // url 播放地址
```

通过 FileID 播放（点播）

在 index.html 页面初始化的代码中加入以下初始化脚本，传入在准备工作中获取到的 fileID（媒资管理）中的视频 ID 与 appID（在账号信息>基本信息中查看）。

```
var player = TCPlayer('player-container-id', { // player-container-id 为播放器容器 ID，必须与 html 中一致
    fileID: '3701925921299637010', // 请传入需要播放的视频 fileID（必须）
    appID: '1500005696', // 请传入点播账号的 appID（必须）
    //私有加密播放需填写 psign， psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436

psign:'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwNTY5NiwiZmlsZUlkIjoiMzcwMTkyNTkyMTI5OTYzNzAxMCIsImN1cnJlbnRUaW1lU3RhbXAiOjE2MjY4NjAxNzYsImV4cGlyZVRpbWVTdGFtcCI6MjYyNjg1OTE3OSwicGNmZyI6InByaXZhdGUiLCJ1cmxBY2Nlc3NJbmZvIjp7InQiOiI5YzkyYjBhYiJ9LCJkcm1MaWNlbnNlSW5mbyI6eyJleHBpcmVUaW1lU3RhbXAiOjI2MjY4NTkxNzksInN0cmljdE1vZGUiOjJ9fQ.Bo5K5ThInc4n8AlzIZQ-CP9a49M2mEr9-zQLH9ocQgI',
    licenseUrl: 'license/url', // 参考准备工作部分，在视立方控制台申请 license 后可获得 licenseUrl
});
```

注意：
要播放的视频建议使用腾讯云转码，原始视频无法保证在浏览器中正常播放。

步骤4: 更多功能

播放器可以结合云点播的服务端能力实现高级功能，比如自动切换自适应码流、预览视频缩略图、添加视频打点信息等。这些功能在 播放长视频方案 中有详细的说明，可以参考文档实现。
此外，播放器还提供更多其他功能，功能列表和使用方法请参见 功能展示 页面。

TCPlayerLite 升级指引

最近更新时间：2023-09-19 18:43:53

TCPlayerLite 为旧版播放器，仅包含基础直播场景的播放功能，而 TCPlayer 是兼顾直播和点播场景的完整版播放器，包含 TCPlayerLite 全部能力，同时拥有更多更强大的播放以及数据统计等功能。
若您正在使用 TCPlayerLite，建议您升级到 TCPlayer 以享受更多更全面的功能及服务，本文将为您介绍如何从 TCPlayerLite 升级为 TCPlayer。

> **说明：**
> TCPlayerLite 仍将持续维护现有能力，您可继续使用，但后续 Web 端播放器功能迭代将主要在 TCPlayer 内进行，TCPlayerLite不再主动做功能迭代。

参见文档

- TCPlayer
- TCPlayerLite

功能展示

更直观的体验 TCPlayer，可以参见 Web 端播放器体验，可体验 TCPlayer 的各项功能并查看相关代码示例。

操作步骤

1. 替换 SDK 文件

页面引用的样式和 js 文件如下，可以参见 TCPlayer 接入文档，引用最新版本的播放器 sdk 及依赖，或从文档中下载所需文件，自行部署使用。

```
<!-- 样式文件 -->
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.1/tcplayer.min.css"
rel="stylesheet"/>

<!-- 依赖文件，按需使用-->
<!--如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 Webrtc 视频，需要在
tcplayer.vx.x.x.min.js 之前引入 TXLivePlayer-x.x.x.min.js。-->
<!--有些浏览器环境不支持 Webrtc，播放器会将 Webrtc 流地址自动转换为 HLS 格式地址，因此快直播场景同样需要
引入hls.min.x.xx.xm.js。-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.1/libs/TXLivePlayer-
1.2.0.min.js"></script>
<!--如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在
tcplayer.vx.x.x.min.js 之前引入 hls.min.x.xx.xm.js。-->
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.1/libs/hls.min.0.13.2m.js">
</script>
<!--如果需要在现代浏览器中播放 FLV 格式的视频，需要在 tcplayer.vx.x.x.min.js 之前引入
flv.min.x.x.js。-->
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.1/libs/flv.min.1.5.js">
</script>

<!--播放器脚本文件-->
```

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.1/tcplayer.v4.5.1.min.js">
</script>
```

2. 初始化播放器

2.1 在 TCPlayer 中初始化播放器时，可以通过 URL 形式播放，也可以通过 FileID 形式播放。这里对播放 URL 进行举例说明：
2.2 初始化播放器时，可以通过 sources 字段指定所要播放的 URL，或者在初始化播放器之后，调用播放器实例上的 src 方法进行播放。

```
// 1. 通过 sources 字段播放
var player = TCPlayer('player-container-id',{
  sources: [{
    // 快直播地址
    src: 'webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=6403f7bb'
  }, {
    // HLS直播地址
    src: 'https://5664.liveplay.myqcloud.com/live/5664_harchar1.m3u8?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=6403f7bb'
  }],
  // 可配置参数说明
https://cloud.tencent.com/document/product/881/30820#options-.E5.8F.82.E6.95.B0.E5.88.97
.E8.A1.A8
});

// 2. 通过 src 方法播放
player.src(url); // url 播放地址
```

2.3 如果需要在直播场景设置多清晰度播放，可以参见如下方式：

```
var player = TCPlayer('player-container-id',{
  multiResolution:{
    sources:{
      'SD':[
        {
          src: 'webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=6403f7bb',
        }
      ],
      'HD':[
        {
          src: 'webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=6403f7bb',
        }
      ],
      'FHD':[
        {
          src: 'webrtc://5664.liveplay.myqcloud.com/live/5664_harchar1?
txSecret=f22a813b284137ed10d3259a7b5c224b&txTime=6403f7bb',
        }
      ]
    },
```

```
    // labels:{
    //    'SD':'标清','HD':'高清','FHD':'超清'
    // },
    showOrder:['SD','HD','FHD'],
    defaultRes: 'SD'
  },
});
```

3. 事件监听方式

在 TCPlayer 中，监听事件的方式有所区别，所有事件参见 API 文档。

```
var player = TCPlayer('player-container-id', options);
// player.on(type, function);
player.on('error', function(error) {
  // 做一些处理
});
```

Flutter
标准直播拉流

最近更新时间：2025-01-21 10:42:32

基础知识

本文主要介绍视频云 Live Flutter Plugin 的直播播放功能。

直播简介

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。

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
Flutter	Github

快速开始

1. 设置依赖

按照 SDK 集成指引 将 live_flutter_plugin 嵌入您的 App 工程中。

```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

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

   - 若您暂未获得 License 授权，需先参考 新增与续期License 进行申请。
2. 在您的 App 调用 live_flutter_plugin 的相关功能之前进行如下设置：

```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.cloud.tencent.com/live/license)
setupLicense() {
  // 当前应用的License LicenseUrl
  var LICENSEURL = "";
  // 当前应用的License Key
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

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

3. 创建 Player

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

```
import 'package:live_flutter_plugin/v2_tx_live_player.dart';
/// 初始化V2TXLivePlayer
```

```
initPlayer() {
  _livePlayer = V2TXLivePlayer();
  _livePlayer.addListener(onPlayerObserver);
}
```

4. 渲染 View

接下来我们要给播放器的视频画面找个地方来显示，Flutter 需要依赖 `v2_tx_live_video_widget` 创建视频渲染 View。

```
import 'package:live_flutter_plugin/widget/v2_tx_live_video_widget.dart';

/// 视频渲染View Widget
Widget renderView() {
  return V2TXLiveVideoWidget(
    onViewCreated: (viewId) async {
      // 设置视频渲染View
      _livePlayer.setRenderViewID(_renderViewId);
    },
  );
}
```

5. 启动播放

```
/// 开始拉流
startLivePlay() async {
  // 生成拉流url RTMP/TRTC/Led
  var url = ""
  // 开始拉流
  await _livePlayer?.startLivePlay(url);
}
```

- **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 > 常用工具 > 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. 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

7. 暂停播放

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

```
// 暂停
_livePlayer.pauseAudio();
_livePlayer.pauseVideo();
// 恢复
_livePlayer.resumeAudio();
_livePlayer.resumeVideo();
```

8. 结束播放

```
// 停止播放
_livePlayer.stopPlay();
```

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, 5);
//极速模式
_txLivePlayer.setCacheParams(1, 1);
//流畅模式
_txLivePlayer.setCacheParams(5, 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。

4. 直播互动

概述

最近更新时间：2024-12-11 16:20:42

方案介绍

直播 SDK 基于实时音视频 TRTC 来实现更灵活、更低延时、更多人数的直播互动能力。支持 iOS、Android、小程序和 Flutter 端。一般流程如下：

- **观众连麦流程：**
  - 连麦前：主播使用 RTC 地址推流，观众使用 CDN 地址进行拉流观看即可。
  - 连麦时：观众切换为 RTC 地址推送自己的音视频，使用 RTC 地址播放主播音视频流，此时主播使用 RTC 地址播放观众的音视频流。
  - 连麦后：观众切换为 CDN 地址拉流观看，主播停止播放观众音视频流。
- **主播 PK 流程：**
  - PK 前：主播们各自用 RTC 地址推流。
  - PK 时：主播们之间相互播放对方的 RTC 流地址。
  - PK 后：主播们停止播放对方的 RTC 地址。

方案优势

为了提高在弱网下的推流效果，直播 SDK 在传统的 RTMP 协议推流的基础上增加了 RTC 协议的推流方式。详情请参见 RTC 推流优势 。

连麦费用

连麦费用采用阶梯价格，详情请参见 RTC 连麦方案怎么计算费用 。

控制台介绍

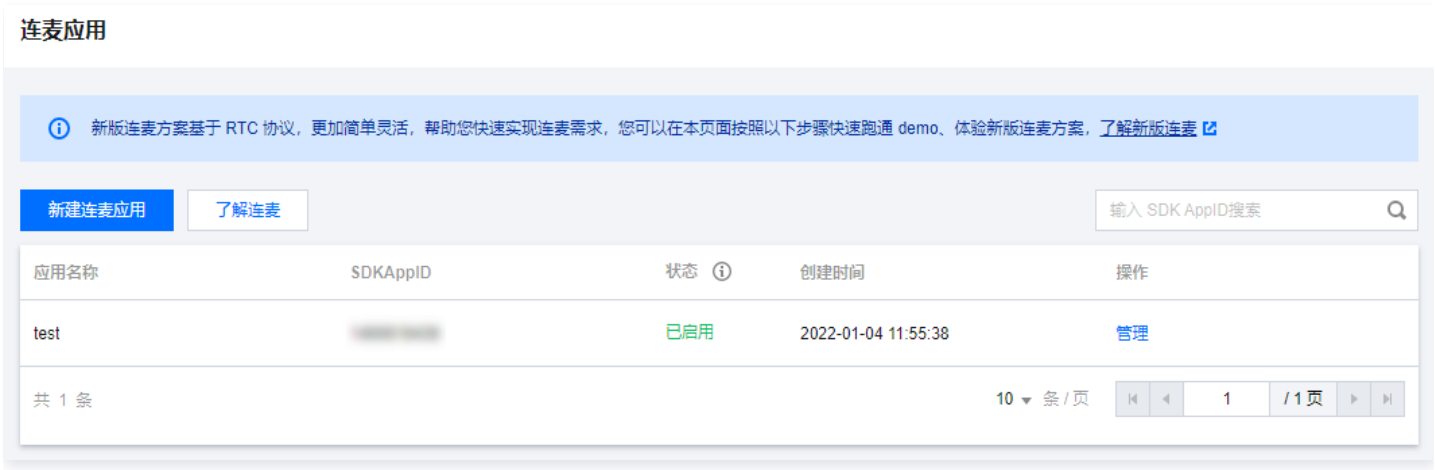
为了方便用户快速接入以及体验，我们在云直播控制台上线了连麦管理功能，包括 快速上手 、连麦应用 、用量统计 和 地址生成器 四个功能页面。

快速上手

我们提供了 MLVB-API-Example Demo 来演示如何使用 SDK 的 `V2TXLivePusher` 和 `V2TXLivePlayer` API 实现观众连麦和主播 PK 能力。您可以通过下载 SDK 源码，简单配置 License、连麦应用以及域名后，即可快速跑通 Demo。详细操作指引请参见 快速上手 。

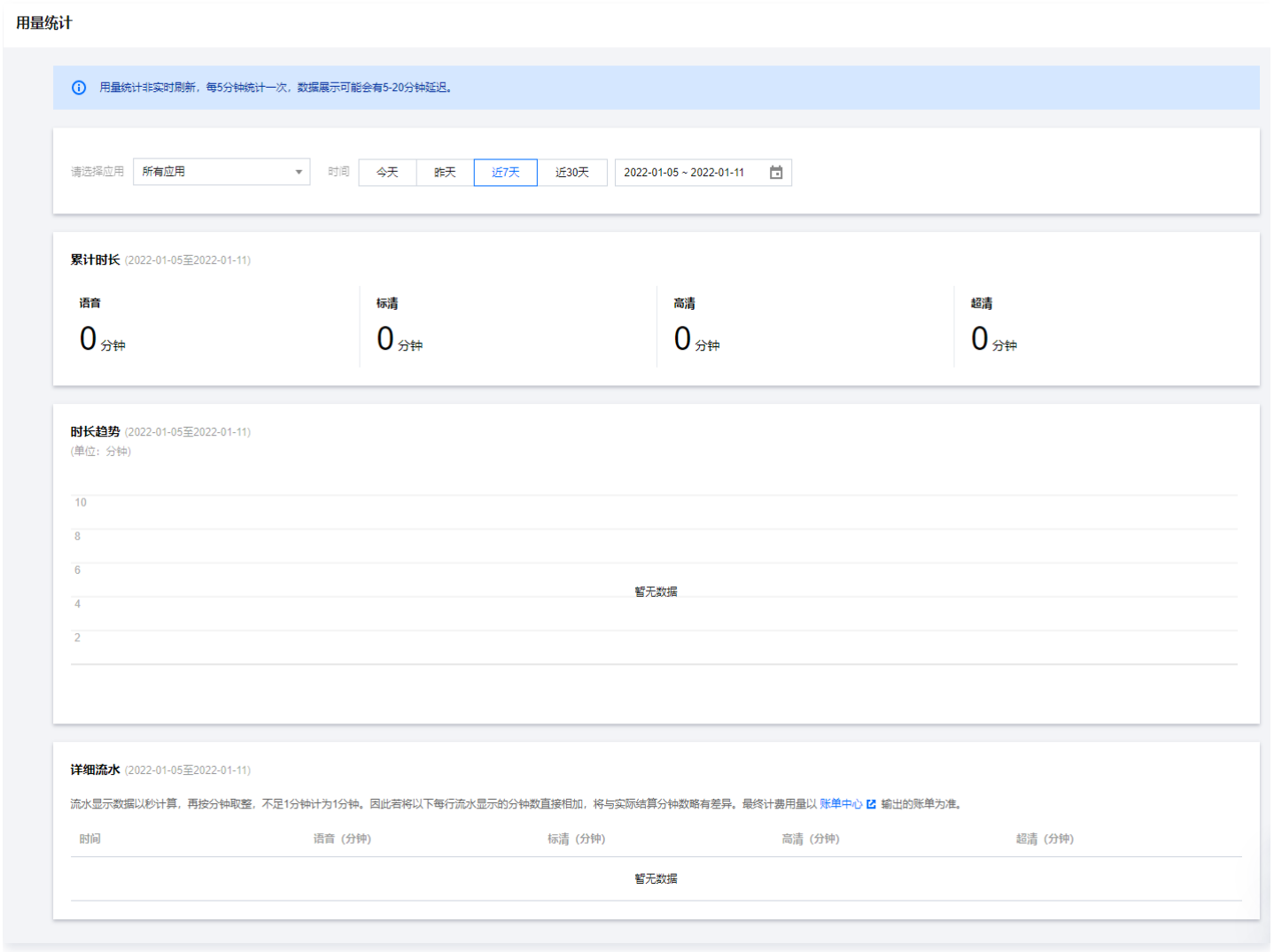
连麦应用

您可以新建连麦应用并对连麦应用进行查看和管理，详细操作指引请参见 连麦应用 。



用量统计

腾讯云视立方提供连麦用量统计查询功能，便于您在本页面查看连麦应用的相关用量及详细流水信息。详细操作指引请参见 用量统计 。



地址生成器

为了便于您快速使用新版连麦方案，我们提供快速生成可用的 RTC 推/拉流地址和 CDN 播放地址的工具。与云直播控制台的地址生成器不同，您可根据业务需求，在连麦管理的地址生成器中选择观众连麦 或者 主播 PK 场景，填写必要信息后快速一键生成该场景的 RTC 推拉流地址和各协议的 CDN 播放地址。同时提供配套的场景解析图供您了解连麦原理并按步骤完成 SDK 接入。详细操作指引请参见 地址生成器 。

地址生成器

您可在本地址生成器中，快速获取用于观众连麦/主播 PK 连麦场景的 trtc 推/拉流地址和 CDN 播放地址。

观众连麦 | 主播PK

选择连麦应用 * test

主播 Stream Id * 请设置自定义流ID
仅支持英文字母、数字和符号

主播 User Id * 请设置自定义UserId
仅支持英文字母、数字和符号

连麦观众 Stream Id * 请设置自定义流ID
仅支持英文字母、数字和符号

连麦观众 User Id * 请设置自定义UserId
仅支持英文字母、数字和符号

选择域名 * 请选择

App Name live
默认为 live，支持自定义设置，可输入英文字母、数字和符号

有效时间 * 2022-01-12 09:20

生成地址 拼接地址说明

观众连麦

最近更新时间：2024-10-29 11:32:41

连麦方案介绍

连麦方案演示

一般情况观众连麦流程如下：

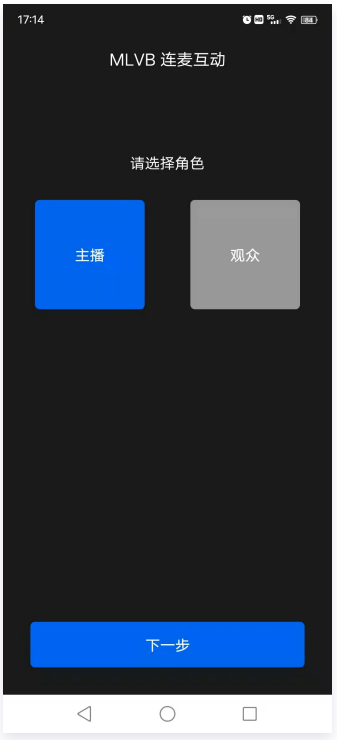
- 连麦前：主播使用 RTC 地址推流，观众使用 CDN 地址进行拉流观看即可。
- 连麦时：观众切换为 RTC 地址推送自己的音视频，使用 RTC 地址播放主播音视频流，此时主播使用 RTC 地址播放观众的音视频流。
- 连麦后：观众切换为 CDN 地址拉流观看，主播停止播放观众音视频流。

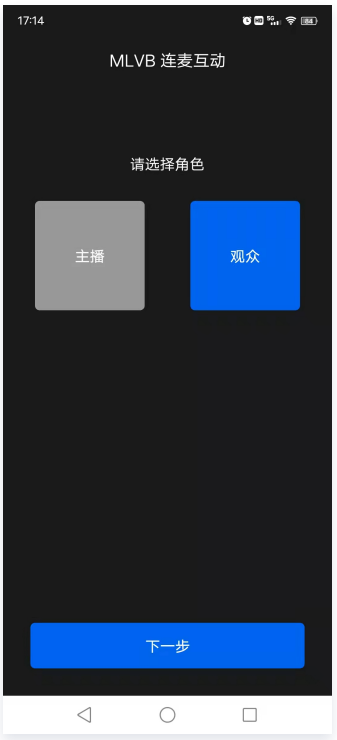
下面是 MLVB-API-Example Demo 的演示效果。

演示图示

直播前

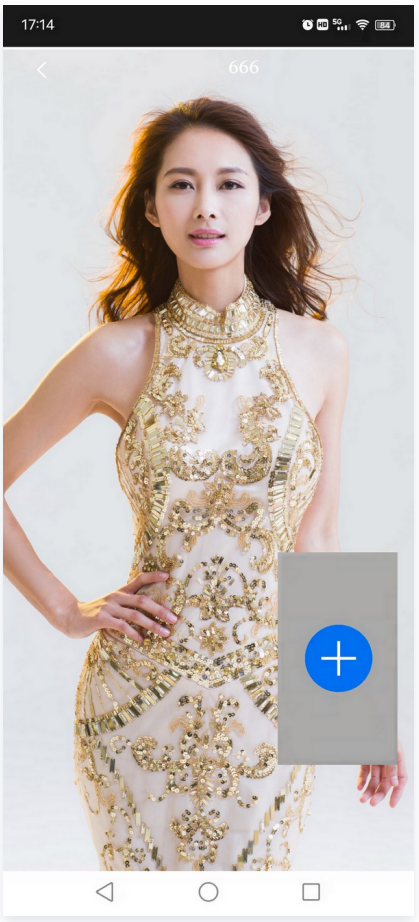
主播

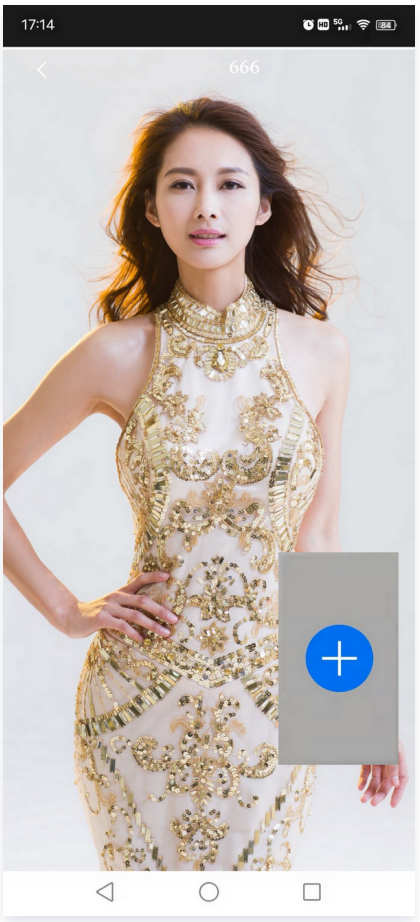


连麦观众和观众			

连麦操作

主播	连麦观众


17:14
666


17:14
666


17:14
666
请输入观众的用户名
确定

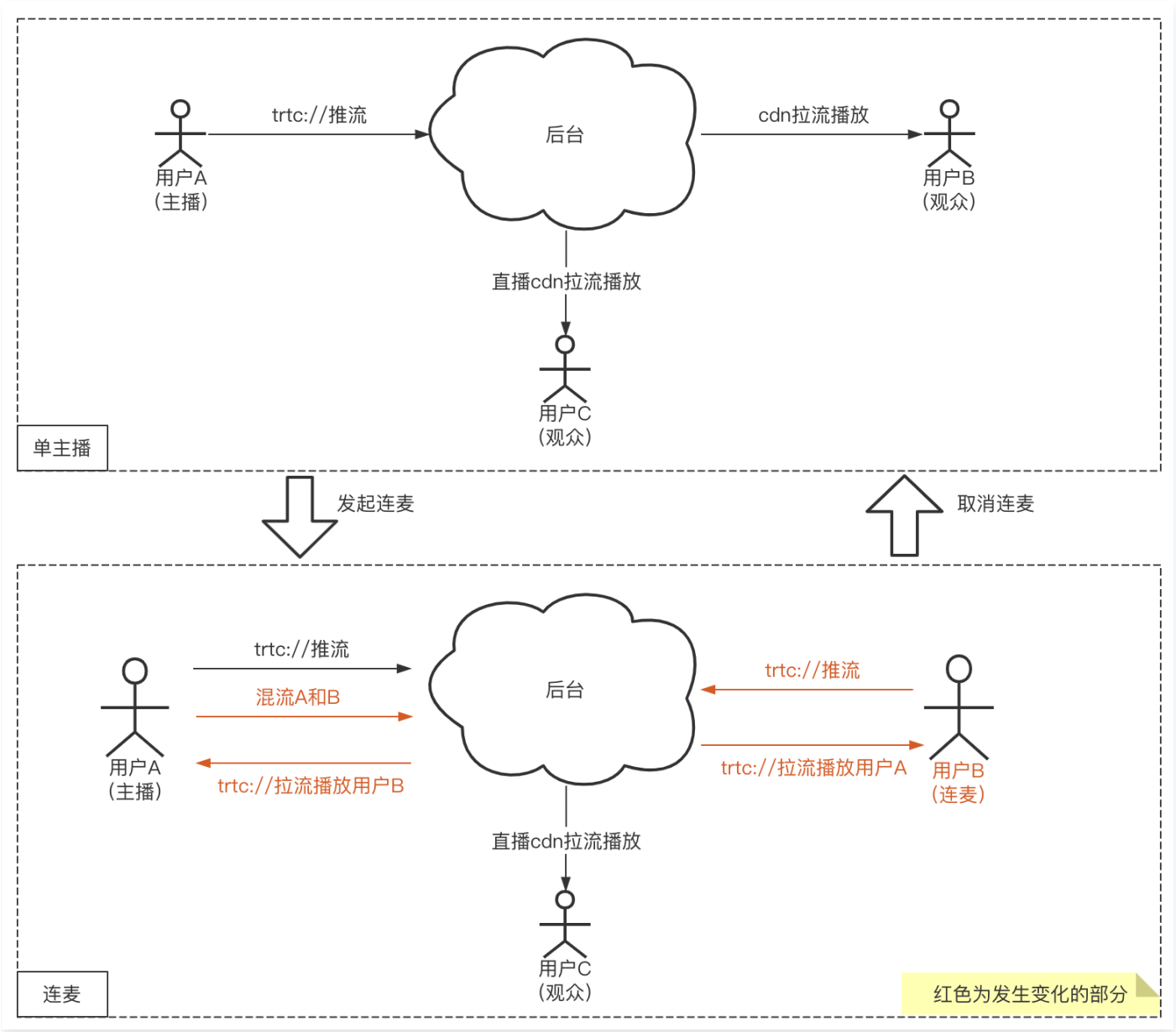
连麦中

主播端（手机 A）	连麦观众（手机 B）	普通观众（手机 C）
		

连麦功能实现

如下示意图，用户 A 是主播，B 和 C 都是观众，如果观众 B 要与 主播 A 连麦，需要做的事情非常简单：

- 观众 B：启动 `trtc://` 协议推流，播放流也从 CDN 切到超低延迟的 `trtc://` 协议。
- 主播 A：开始播放观众 B 的流，同时发起混流指令，把 A 和 B 的内容合成一路。
- 观众 C：无需变化，继续 CDN 播放即可，只不过会看到混流后的连麦画面。



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。URL 拼装方案请参见 如何拼装 URL 。

java

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

Objective-C

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
NSString *pushURLA = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxx";
```

```
[pusher startPush:pushURLA];
```

小程序

```
// wxml
<live-pusher url="{{pusher.url}}" />
// js
import TRTC from 'trtc-wx-sdk'
this.TRTC = new TRTC(this)
this.TRTC.createPusher({})
data: { pusher: {} }
this.setData({
    pusher: this.TRTC.enterRoom({
          userID: 'Anchor',
          sdkAppID: 000000,
          userSig: 'xxxxxx',
          roomID: 123456
        }),
  }, () => {
    trtc.getPusherInstance().start() // 开始推流
  })
}
```

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。URL 拼装方案请参见 如何拼装 URL 。

java

```
V2TXLivePlayer player = 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"
*/
player.startLivePlay(playURLA);
```

Objective-C

```
V2TXLivePlayer * player = [[V2TXLivePlayer alloc] init];
/**
* 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。
* NSString *playURLA= @"http://3891.liveplay.myqcloud.com/live/streamidA.flv";
* NSString *playURLA= @"http://3891.liveplay.myqcloud.com/live/streamidA.hls";
* NSString *playURLA= @"webrtc://3891.liveplay.myqcloud.com/live/streamidA"
*/
```

```
[player setRenderView:view];
[player startLivePlay:playURLA];
```

小程序

```
// wxml
<live-player src="{{ url }}" />
/**
* 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"
*/
this.setData({ ulr: 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）。

java

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

Objective-C

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
NSString *pushURLB = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=B&usersig=xxx";
[pusher startPush:pushURLB];
```

小程序

```
// wxml
<live-pusher url="{{pusher.url}}" />
// js
import TRTC from 'trtc-wx-sdk'
this.TRTC = new TRTC(this)
this.TRTC.createPusher({})
data: { pusher: {} }
this.setData({
    pusher: this.TRTC.enterRoom({
          userID: 'audience',
          sdkAppID: 000000,
          userSig: 'xxxxxx',
          roomID: 123456
```

```
        }),
  }, () => {
    trtc.getPusherInstance().start() // 开始推流
  })
}
```

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。

java

```
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
playURLB= "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=A&usersig=xxx&appscene=live";
player.startLivePlay(playURLB);
```

Objective-C

```
V2TXLivePlayer * player = [[V2TXLivePlayer alloc] init];
NSString* playURLB = @"trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=A&usersig=xxx&appscene=live";
[player setRenderView:view];
[player startLivePlay:playURLB];
```

小程序

```
// wxml
<live-player src="{{player.src}} mute-audio={{ false }}" mute-vedio="{{ false }}" />
// js
import TRTC from 'trtc-wx-sdk'
this.TRTC = new TRTC(this)
data: {
    player: {}
}
// 远端用户推送视频
this.TRTC.on(TRTC_EVENT.REMOTE_VIDEO_ADD, (event) => {
    const { player } = event.data
    this.setData({ player: player })
})
// 远端用户推送音频
this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_ADD, (event) => {
    const { player } = event.data
    this.setData({ player: player })
})
```

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。

java

```
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
playURLA= "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=B&usersig=xxx&appscene=live";
player.startLivePlay(playURLA);
```

Objective-C

```
V2TXLivePlayer * player = [[V2TXLivePlayer alloc] init];
NSString* playURLA = @"trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=B&usersig=xxx&appscene=live";
[player setRenderView:view];
[player startLivePlay:playURLA];
```

小程序

```
// wxml
<live-player src="{{player.src}} mute-audio={{ false }}" mute-vedio="{{ false }}" />
// js
import TRTC from 'trtc-wx-sdk'
this.TRTC = new TRTC(this)
data: {
    player: {}
}
// 远端用户推送视频
this.TRTC.on(TRTC_EVENT.REMOTE_VIDEO_ADD, (event) => {
    const { player } = event.data
    this.setData({ player: player })
})
// 远端用户推送音频
this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_ADD, (event) => {
    const { player } = event.data
    this.setData({ player: player })
})
```

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：

java

```
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.outputStreamId  = null;
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);
```

Objective-C

```
V2TXLiveTranscodingConfig *config = [[V2TXLiveTranscodingConfig alloc] init];
// 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.outputStreamId   = nil;

// 主播摄像头的画面位置
V2TXLiveMixStream *local = [[V2TXLiveMixStream alloc] init];
local.userId   = @"localUserId";
local.streamId = nil; // 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

// 连麦者的画面位置
V2TXLiveMixStream *remoteA = [[V2TXLiveMixStream alloc] init];
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;

// 连麦者的画面位置
V2TXLiveMixStream *remoteB = [[V2TXLiveMixStream alloc] init];
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;

//设置混流 streams
config.mixStreams = @[local,remoteA,remoteB];

// 发起云端混流
pusher.setMixTranscodingConfig(config);
```

小程序

参考文档：服务端 REST API 混流方案 https://cloud.tencent.com/document/product/647/16827

⚠ 注意：

- 发起云端混流后
  - 不设置 outputStreamId，SDK 会执行默认逻辑，即房间里的多路流会混合到该接口调用者的视频流上，也就是 A + B => A
  - 设置 outputStreamId，SDK 会将房间里的多路流混合到您指定的直播流 ID 上，也就是 A + B => C。
- **推荐取值**默认值，不进行设置，即房间里的多路流会混合到该接口调用者的视频流上。

6. 结束连麦

> ⚠ **注意：**
> RTC 连麦基于实时音视频 TRTC 实现，因此连麦结束后，需停止所有连麦参与者（主播和观众）的 RTC 推流和拉流，否则会产生额外的 TRTC 在房时长费用，具体实现请参考下方代码。

主播 A 停止连麦，先调用 `V2TXLivePusher` 组件的 setMixTranscodingConfig 并设置为 null 从而停止混流。然后调用 `V2TXLivePlayer` 的 `stopPlay` 停止播放连麦观众 B 的流。

java

```
// 停止混流
pusher.setMixTranscodingConfig(null);
// 停止播放连麦观众 B 的流
player.stopPlay();
```

Objective-C

```
// 停止混流
[pusher setMixTranscodingConfig: nil];
// 停止播放连麦观众 B 的流
[player stopPlay];
```

小程序

```
// wxml
<live-player src="{{ player.url }}" />
// 停止播放连麦观众 B 的流
this.setData({ player: {} })
```

连麦观众 B 停止连麦，先调用 `V2TXLivePusher` 组件的 stopPush 停止自身推流。然后停止播放主播 A 的 RTC 流。如需继续以观众身份留在房中观看，请按照步骤2，进行 CDN 拉流。

java

```
// 停止推流
pusher.stopPush();
// 停止播放主播 A 的 RTC 流
player.stopPlay();
```

Objective-C

```
// 停止推流
[pusher stopPush];
// 停止播放主播 A 的 RTC 流
[player stopPlay];
```

小程序

```
// wxml
<live-pusher url="{{ pusher.url }}" />
<live-player src="{{ player.src }}" />
// 停止播放主播 A 的 RTC 流
this.setData({ pusher: {}, player: {} })
```

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?

当前 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，考虑到用户的具体使用场景，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。

2. 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，具体可以参考 设置画面质量 。

3. V2TXLivePusher#startPush 或 V2TXLivePlayer#startLivePlay 收到错误码： -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 。

4. RTC连麦方案的时延性有可以参考的数据吗?

主播连麦的延时 < 200ms，主播和观众的延时在 100ms - 1000ms。

5. 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。

6. 如何避免额外计费?

及时主动停止推流（ V2TXLivePusher 的 stopPush ）和拉流（ V2TXLivePlayer 的 stopPlay ）。只要推流或拉流在进行中，就会正常计费。

7.观众连麦支持哪些平台?

观众连麦支持 iOS、Android、小程序和 Flutter 端。

小程序跨房连麦

最近更新时间：2025-01-21 10:42:32

> **说明：**
> 新版直播连麦基于 TRTC 能力实现，小程序端连麦需要订阅实时音视频 TRTC 全新上线的包月套餐「尊享版」及以上版本，来解锁实时音视频 TRTC 小程序端服务，包月套餐中赠送的时长补充包可用于抵扣连麦时长消耗，计费详情请查看 新版直播连麦费用，实时音视频 TRTC 包月套餐详情请参见 包月套餐计费说明。

跨房连麦流程介绍

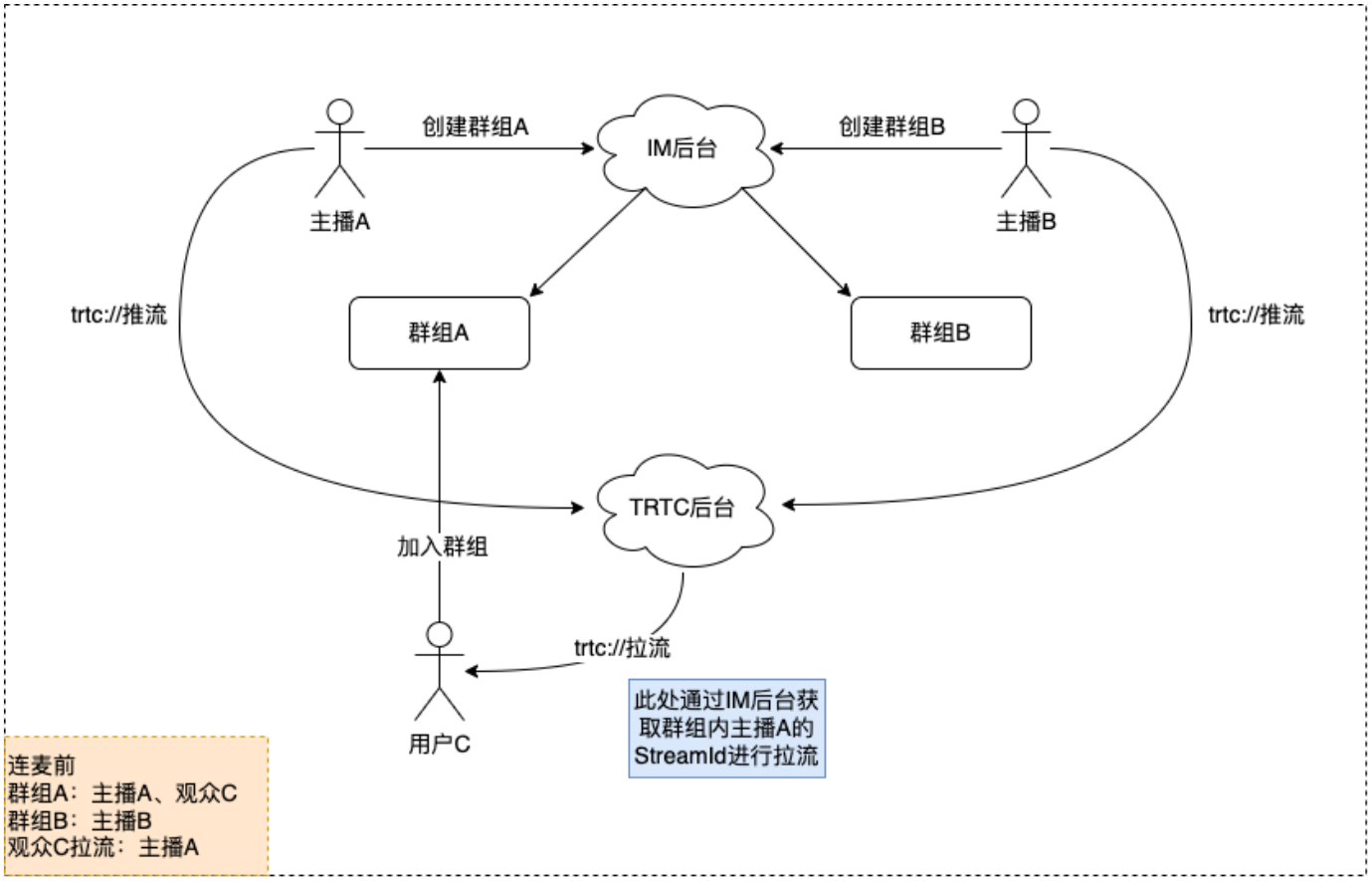
假设场景：主播 A、主播 B、观众 C

- 连麦前：主播 A、B 分别使用 RTC 地址推流，此时观众可通过 RTC 地址拉取主播 A 或 B 的流
- 连麦时：主播 A、B 分别拉取对方的音视频流
- 连麦后：观众同时拉取主播 A、B 的音视频流

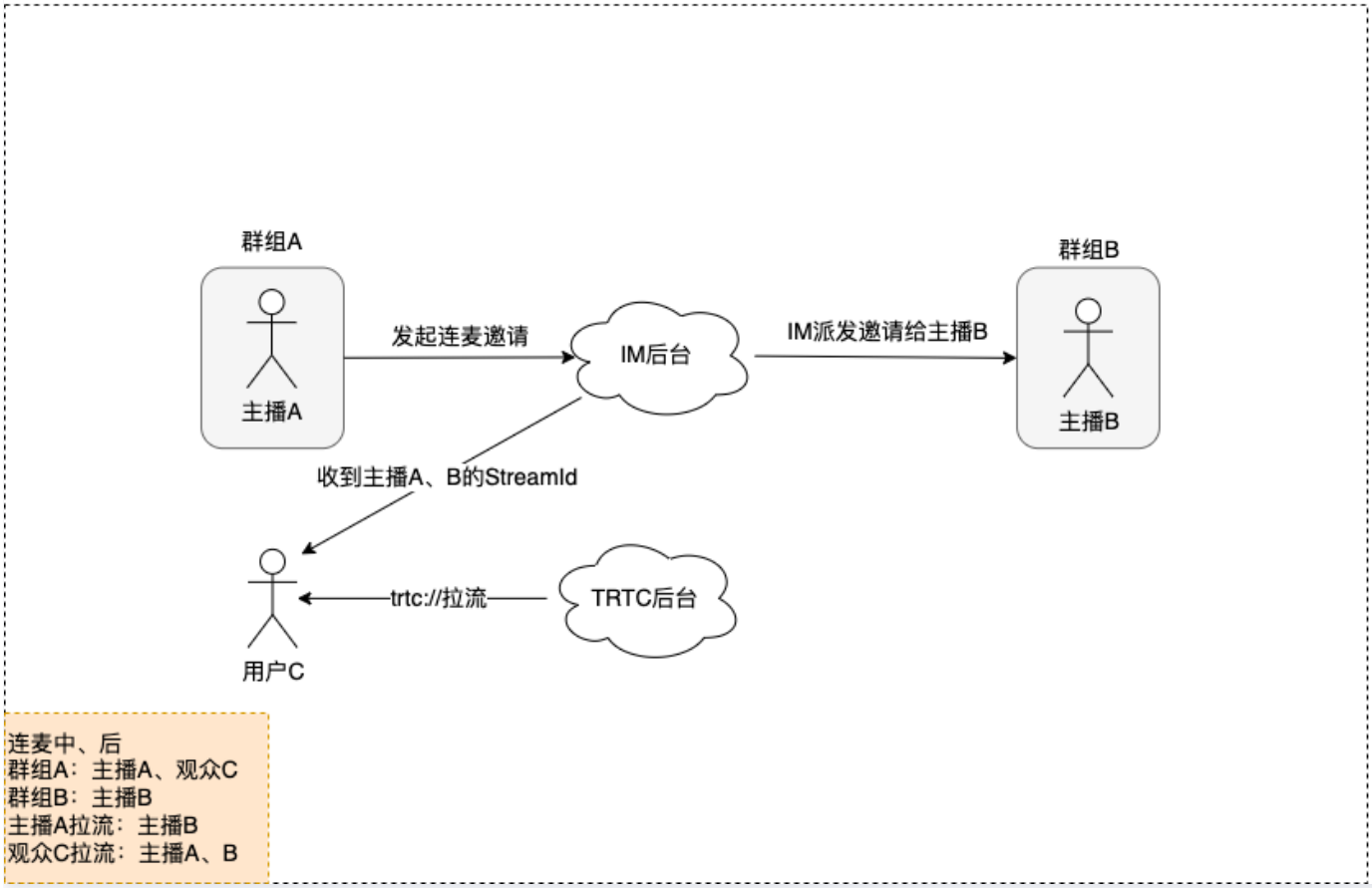
跨房连麦功能实现

- **连麦前：**
  - 主播 A、B 分别通过 IM 创建各自的群组 A、B
  - 观众 C 加入群组 A，拉取主播 A 的音视频流
- **连麦中：**
  - 主播 A 发送连麦请求给主播 B
  - 主播 A 收到主播 B 同意连麦请求的消息
- **连麦后：**
  - 主播 A 推流的同时拉取主播 B 的流
  - 观众 C 同时拉取主播 A、B 的流

如下示意图：



连麦前
群组A：主播A、观众C
群组B：主播B
观众C拉流：主播A



连麦中、后
群组A：主播A、观众C
群组B：主播B
主播A拉流：主播B
观众C拉流：主播A、B

代码片段

1. 公共库

```
 qsStringify(obj) {
   if (typeof obj !== 'object' || obj === null) {
     return ''
   }
   const objKeys = Object.keys(obj)
   const keys = []
   for (let i = 0; i < objKeys.length; ++i) {
     const key = objKeys[i]
     if (obj[key] === null) {
       continue
      }
      keys[i] = `${key}=${obj[key]}`
    }
     return keys.join('&')
  },
```

2. 创建推流 URL

```
// streamId为用户自定义变量
// appscene: 视频通话/videocall、在线直播/live、语音通话/audiocall、语音聊天室/voicechatroom
// 生成url后将其赋值给live-pusher的url即可进行推流
const pusherUrl = `trtc://cloud.tencent.com/push/${pusherConfig.streamId}?${
              this.qsStringify({
                 sdkappid: pusherConfig.sdkAppID,
                 userId: pusherConfig.userId,
                 usersig: pusherConfig.userSig,
                 streamId: pusherConfig.streamId,
                 appscene: pusherConfig.appscene,
              })
           }`
```

3. 创建拉流 URL

```
// 生成url后将其赋值给live-player的src即可进行拉流
const playurl: `trtc://cloud.tencent.com/play/${this.data.streamId}?${
        this.qsStringify({
        streamId: this.data.streamId,
        userId,
        sdkappid: sdkAppID,
        usersig: userSig,
        })
      }`
```

常见问题

1. 如何控制推拉流（暂停、截图等操作）

推拉流可通过 live-pusher/player 自身标签赋值或者 wx 提供的方法进行控制，参考文档：

- live-pusher.html
- live-player.html
- livePusherContext.html
- livePlayerContext.html

2. 跨房连麦可以使用trtc-wx.js么?

不可以，trtc-wx.js 里集成的是 room 协议，跨房连麦不能使用 room 协议，因此房间、成员列表等概念需要通过后台实现，可直接使用腾讯云IM进行集成。

3. 如何发送连麦申请?

此处以腾讯云 IM 举例，主播A可以通过 IM 发送自定义消息给主播 B，主播 B 解析消息内容，发现为连麦邀请，主播 B 再向 A 发送自定义消息，A 收到消息解析，发现为同意连麦邀请，这时 A 拉取 B 的音视频流（此处 streamId 可通过 B 传递过来，或者使用 userId 作为 streamId），同时主播 A 发送给主播B连麦成功的消息，B 收到消息拉取主播 A 的音视频流。
简述流程为：A 发起邀请 => B 收到邀请，B 同意 => A 收到同意，A 发送连麦成功 => B 收到连麦成功

4. 如何进行多端互通?

如果您集成了腾讯云的 IM 和 TRTC，它们本身就是不区分平台的，用连麦场景举例，小程序端发起的连麦邀请在 native 端也可收到，您可以通过自定义消息设定特殊字段进行多端统一的内容解析，而trtc本身更不需要做多余处理，生成 URL 进行推拉流即可。

5. 小程序端如何混流实现?

小程序目前只能通过调用服务端 REST API 实现混流，参考文档：

- 云端混流转码
- 混流接口介绍

混流方案的实现与上述流程大致一致，区别在于连麦时主播 A、B 分别去混对方的流，用户 C 只需要拉主播 A 的流即可看到主播 B 的画面。

主播 PK

最近更新时间：2024-12-11 16:20:42

主播 PK 方案介绍

主播 PK 流程

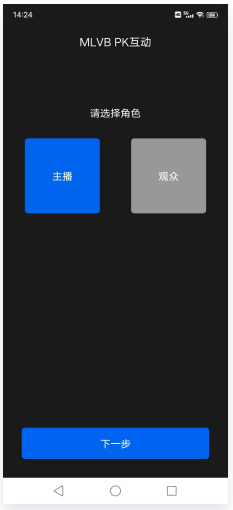
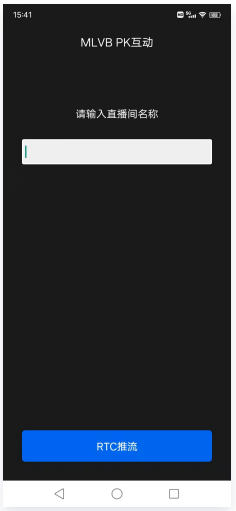
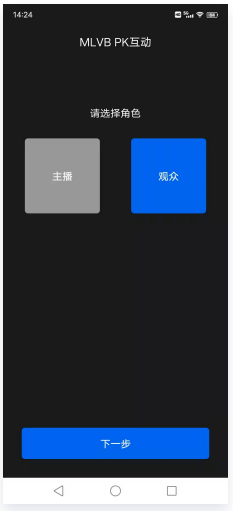
一般情况主播 PK 流程如下：

- PK 前：主播们各自用 RTC 地址推流。
- PK 时：主播们之间相互播放对方的 RTC 流地址。
- PK 后：主播们停止播放对方的 RTC 地址。

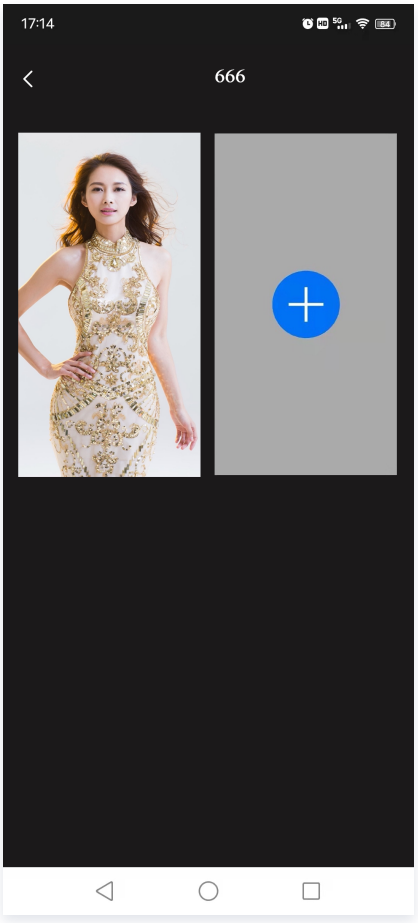
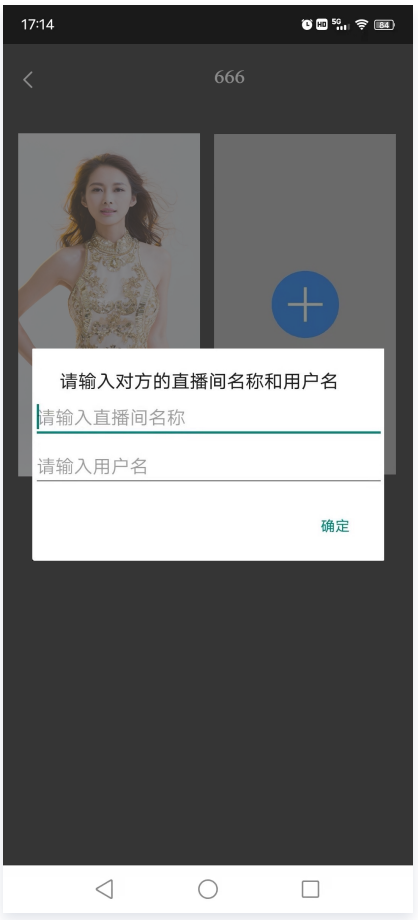
方案演示

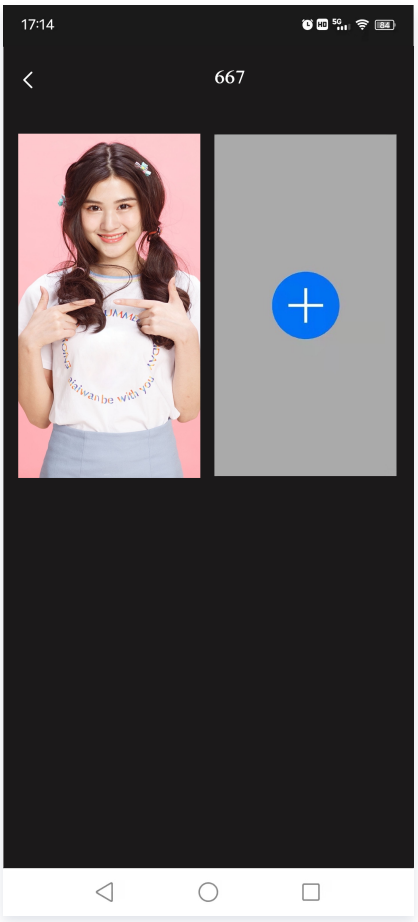
下面是 MLVB-API-Example Demo 的演示效果。

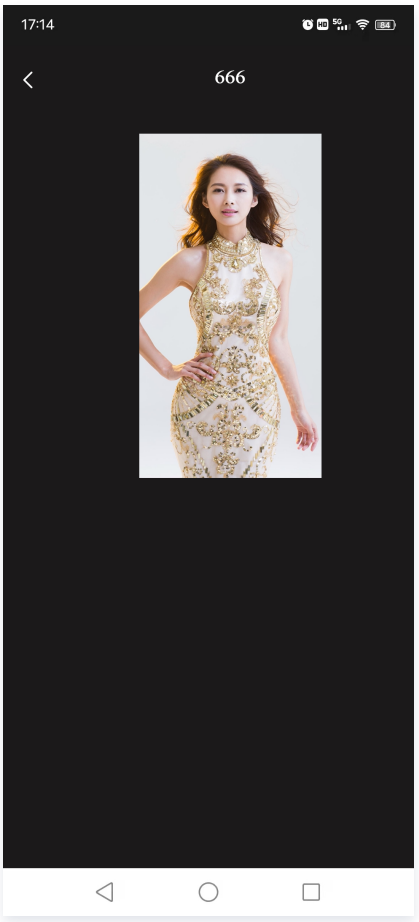
直播前

主播 A 和主播 B			
观众			

PK 操作

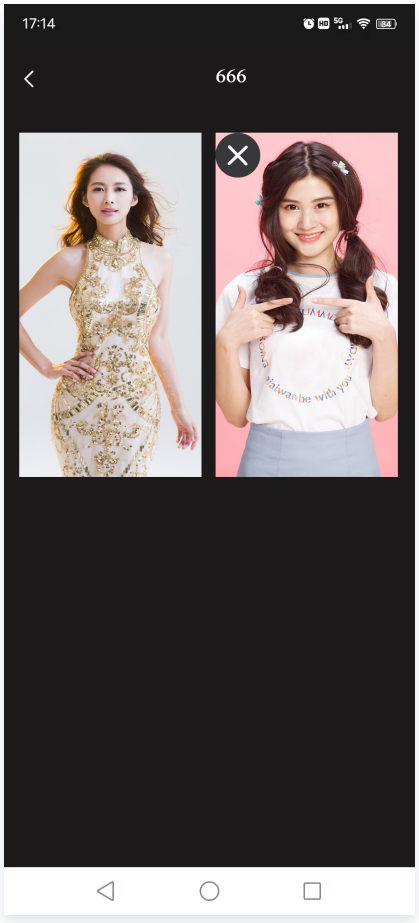
主播 A	 17:14 666	 17:14 666 请输入对方的直播间名称和用户名 请输入直播间名称 请输入用户名 确定

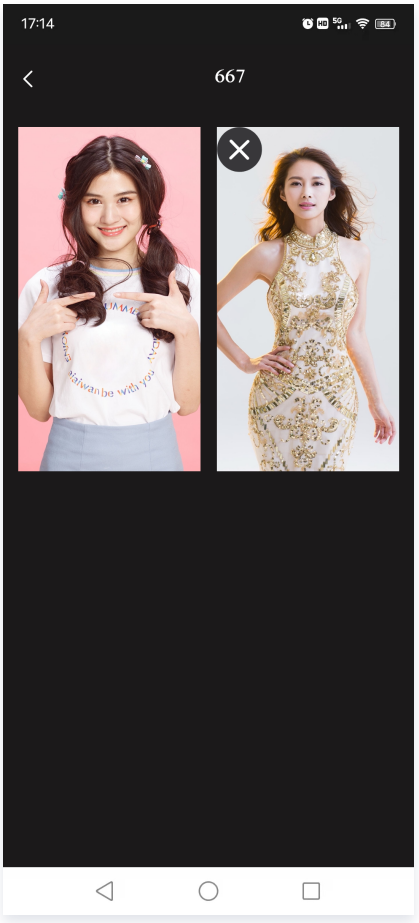
主播 B	 17:14 667	 17:14 667 请输入对方的直播间名称和用户名 请输入直播间名称 请输入用户名 确定

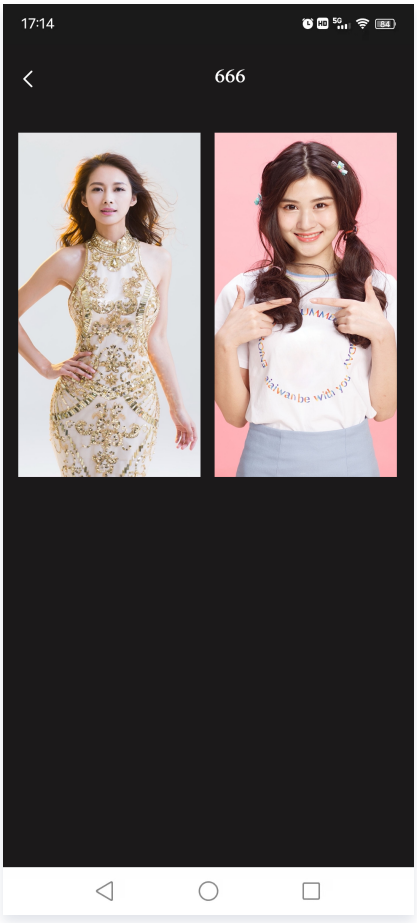
主播 A 的观众	

PK 中

主播 A（手机 A）	主播 B（手机 B）	主播 A 的观众（手机 C）



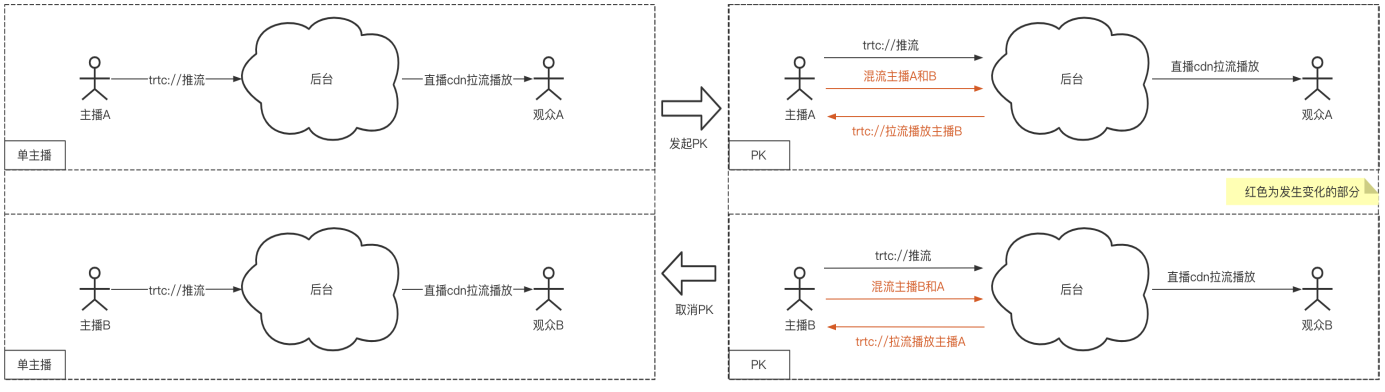




RTC PK 功能实现

如下示意图，主播 A 有观众 A，主播 B 有观众 B，如果主播 A 和 B 进行 PK，需要做的事情非常简单：

- 主播 A：开始播放主播 B 的流，同时发起混流指令，把 A 和 B 的内容合成一路，供主播 A 的观众观看。
- 主播 B：开始播放主播 A 的流，同时发起混流指令，把 B 和 A 的内容合成一路，供主播 B 的观众观看。
- 观众 A 和 B：无需变化，继续 CDN 播放即可，只不过会看到各自主播混流后的 PK 画面。



1. 主播 A 开始推流

调用 `V2TXLivePusher` 组件开始主播 A 的推流。URL 拼装方案请参见 如何拼装 URL 。

java

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

Objective-C

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
NSString *pushURLA = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=A&usersig=xxx";
[pusher startPush:pushURLA];
```

2. 主播 B 开始推流

调用 `V2TXLivePusher` 组件开始主播 B 的推流。URL 拼装方案请参见 如何拼装 URL 。

java

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

Objective-C

```
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTC];
NSString *pushURLB = @"trtc://cloud.tencent.com/push/streamid?
sdkappid=1400188888&userId=B&;usersig=xxx";
[pusher 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 PK 互动直播场景中。URL 拼装方案请参见 如何拼装 URL 。

java

```
// 主播A
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
playURLB = "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=A&usersig=xxx&appscene=live"
player.startLivePlay(playURLB);
```

```
...

// 主播B
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
playURLA= "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=B&usersig=xxx&appscene=live"
player.startLivePlay(playURLA);
```

Objective-C

```
// 主播A
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
NSString *playURLB = "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=A&usersig=xxx&appscene=live"
[player setRenderView:view];
[player startLivePlay:playURLB];
...

// 主播B
V2TXLivePlayer *player = [[V2TXLivePlayer alloc] init];
 NSString *playURLA = "trtc://cloud.tencent.com/play/streamid?
sdkappid=1400188888&userId=B&usersig=xxx&appscene=live"
[player setRenderView:view];
[player startLivePlay:playURLA];
```

4. PK 成功，观看 PK 内容

PK 成功后，观众有两种方式可以观看 PK 内容。

1. 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。
2. 主播 A 和主播 B 进行混流，观众端播放 URL 保持不变。
   主播 A 和主播 B 发起一次混流操作，也就是主播将自己和对方主播混合成一路流，自己直播间的观众就可以在看到自己和对方主播进行互动。主播 A 和 B 各自调用 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。

示例代码：

java

```
// 主播 A
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 A 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);

// PK主播 B 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);

//主播 B
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 B 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);

// PK主播 A 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);

// 发起云端混流
pusher.setMixTranscodingConfig(config);
```

Objective-C

```
// 主播 A
V2TXLiveTranscodingConfig *config = [[V2TXLiveTranscodingConfig alloc] init];

// 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;

// 主播 A 摄像头的画面位置
V2TXLiveMixStream *local = [[V2TXLiveMixStream alloc] init];
local.userId    = @"localUserId";
local.streamId = nil; // 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

// PK主播 B 的画面位置
V2TXLiveMixStream *remoteB = [[V2TXLiveMixStream alloc] init];
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;

//设置混流 streams
config.mixStreams = @[local,remoteB];

// 发起云端混流
pusher.setMixTranscodingConfig(config);
```

```
// 主播 B
V2TXLiveTranscodingConfig *config = [[V2TXLiveTranscodingConfig alloc] init];

// 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;

// 主播 A 摄像头的画面位置
V2TXLiveMixStream *local = [[V2TXLiveMixStream alloc] init];
local.userId    = @"localUserId";
local.streamId = nil; // 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

// PK主播 A 的画面位置
V2TXLiveMixStream *remoteA = [[V2TXLiveMixStream alloc] init];
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;

//设置混流 streams
config.mixStreams = @[local,remoteA];

// 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。

5. 结束 PK

⚠ 注意：
RTC PK 基于实时音视频 TRTC 实现，因此 PK 结束后，需停止所有 PK 参与者（多个主播）的 RTC 推流和拉流，否则会产生额外的 TRTC 在房时长费用，具体实现请参考下方代码。

主播 A 和主播 B 分别调用 `V2TXLivePlayer` 停止播放对方的流。然后根据观众观看的两种方式分别处理。
1. 主播端未开启混流，而是观众端自己去播放对方主播的推流内容，此时观众端停止播放即可。

2. 主播端开启了混流，那么主播端需要停止混流，观众端不影响。

java

```
// 主播 A 停止播放主播 B 的流
player.stopPlay();
// 主播 B 停止播放主播 A 的流
player.stopPlay();

// 方式1：观众各自调用 V2TXLivePlayer 停止播放对方主播的推流内容
player.stopPlay();
// 方式2：主播端停止混流（主播 A 和主播 B 都要执行）
pusher.setMixTranscodingConfig(null);
```

Objective-C

```
// 主播 A 停止播放主播 B 的流
[player stopPlay];
// 主播 B 停止播放主播 A 的流
[player stopPlay];

// 方式1：观众各自调用 V2TXLivePlayer 停止播放对方主播的推流内容
[player stopPlay];
// 方式2：主播端停止混流（主播 A 和主播 B 都要执行）
[pusher setMixTranscodingConfig: nil];
```

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?

当前 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，考虑到用户的具体使用场景，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。

2. 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，具体请参见 设置画面质量 。

3. V2TXLivePusher#startPush 或 V2TXLivePlayer#startLivePlay 收到错误码：-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 。

4. RTC连麦方案的时延性有可以参考的数据吗?

主播连麦的延时 < 200ms，主播和观众的延时在 100ms - 1000ms。

5. 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。

6. 如何避免额外计费?

及时主动停止推流（ `V2TXLivePusher` 的 `stopPush` ）和拉流（ `V2TXLivePlayer` 的 `stopPlay` ）。只要推流或拉流在进行中，就会正常计费。

7.主播 PK 支持哪些平台?

主播 PK 支持 iOS、Android、小程序和 Flutter 端。