直播 SDK

无 UI 集成方案

产品文档

腾讯云

文档目录

无 UI 集成方案
跑通 Demo
iOS

最近更新时间：2024-01-13 15:53:49

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。

操作步骤

步骤一：下载 SDK 和 MLVB-API-Example 源码

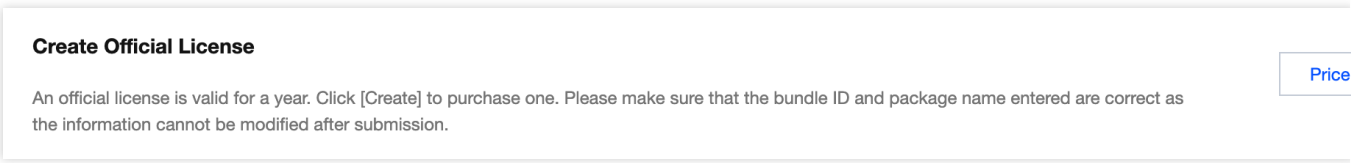
1. 根据实际业务需求 下载 相应的压缩包，这里以 Live版本 为例。
2. 下载完成后，解压。

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**。

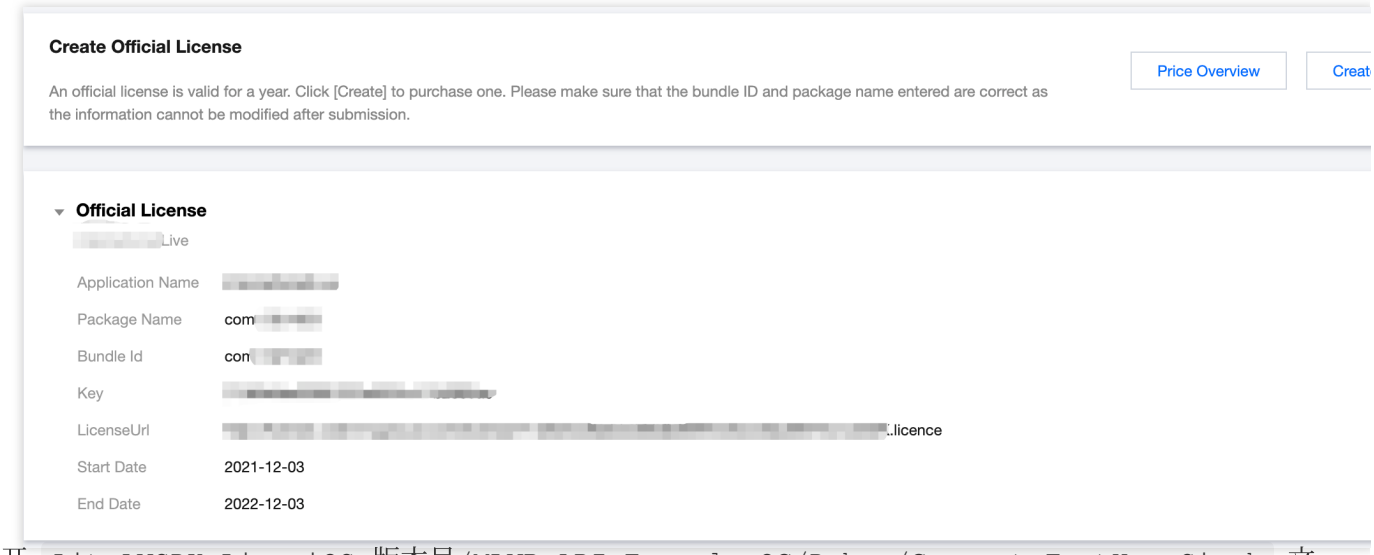


2. 根据实际需求填写 `App Name` 、 `Package Name` 和 `Bundle ID` ，勾选功能模块**直播**（直播推流 + 视频播放），单击**确定**。

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。**

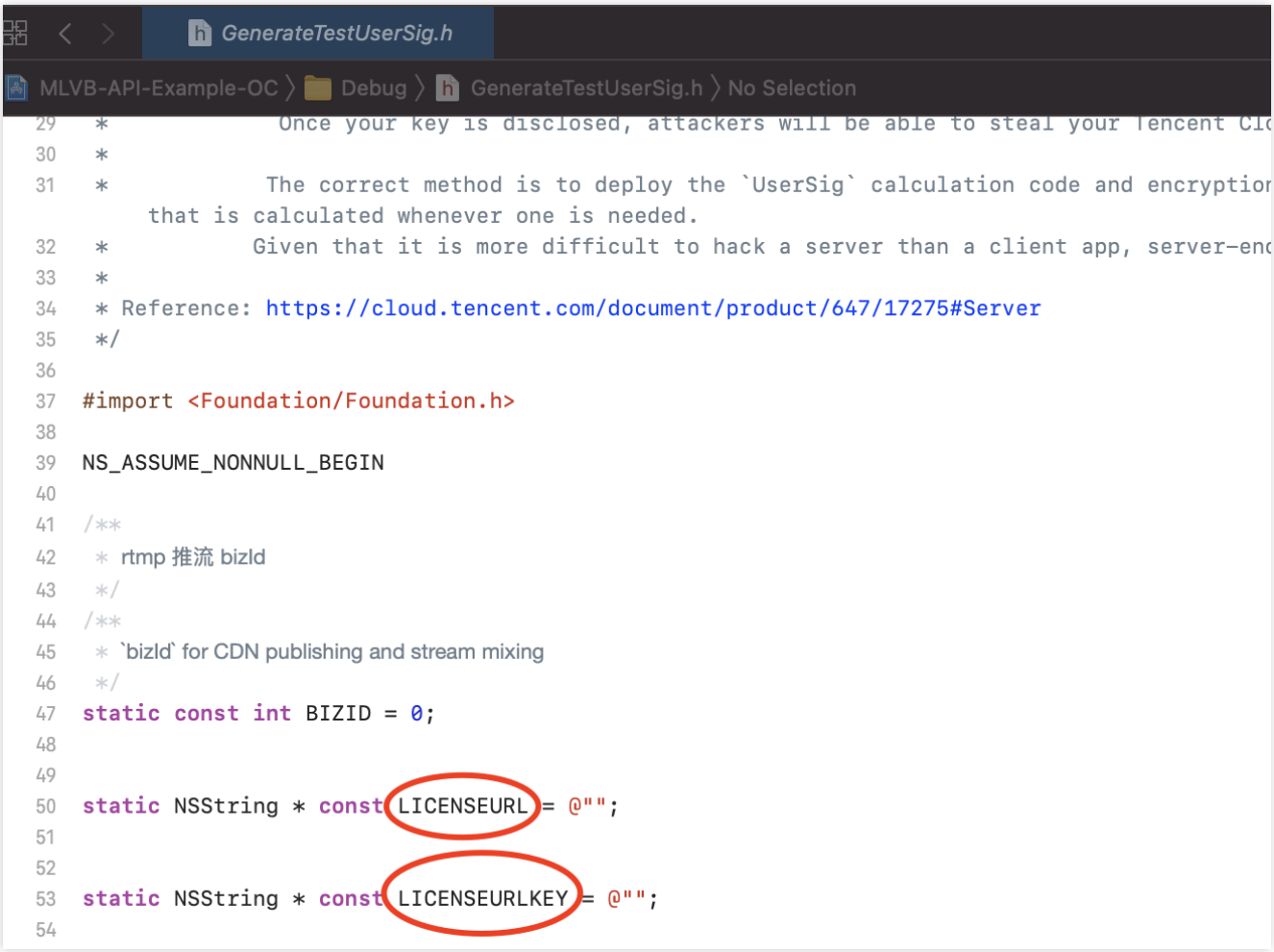


4. 打开 `LiteAVSDK_Live_iOS_版本号/MLVB-API-Example-OC/Debug/GenerateTestUserSig.h` 文件。

设置 `GenerateTestUserSig.h` 文件中的相关参数：

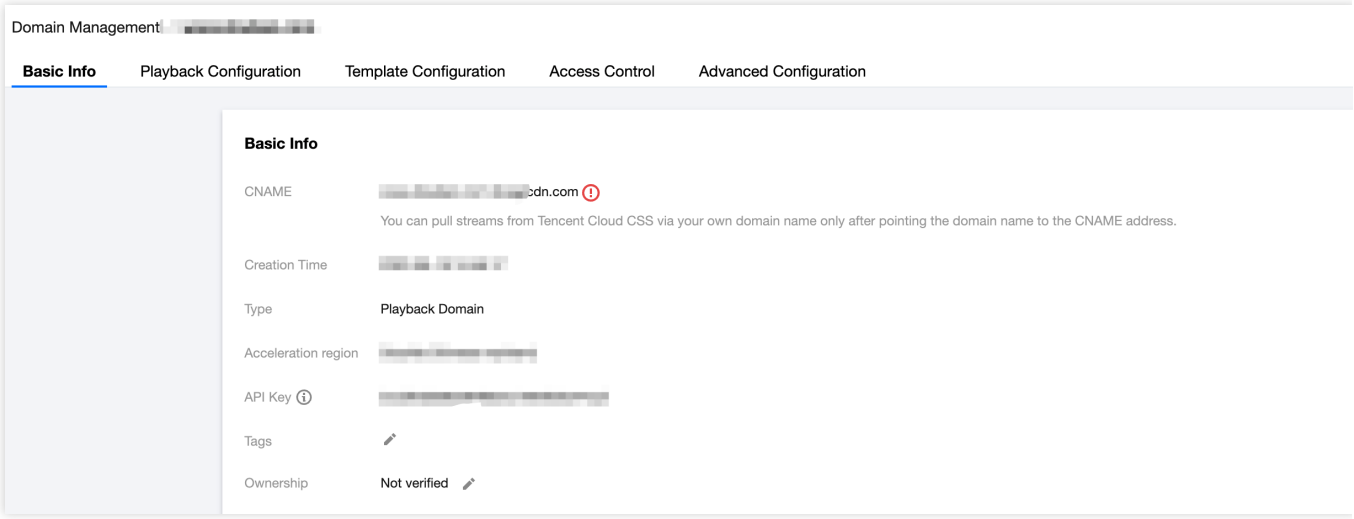
LICENSEURL：默认为空，请设置为您的下载 Licence url。

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_Live_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**。

配置推流参数

1. 找到并打开 `LiteAVSDK_Live_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。

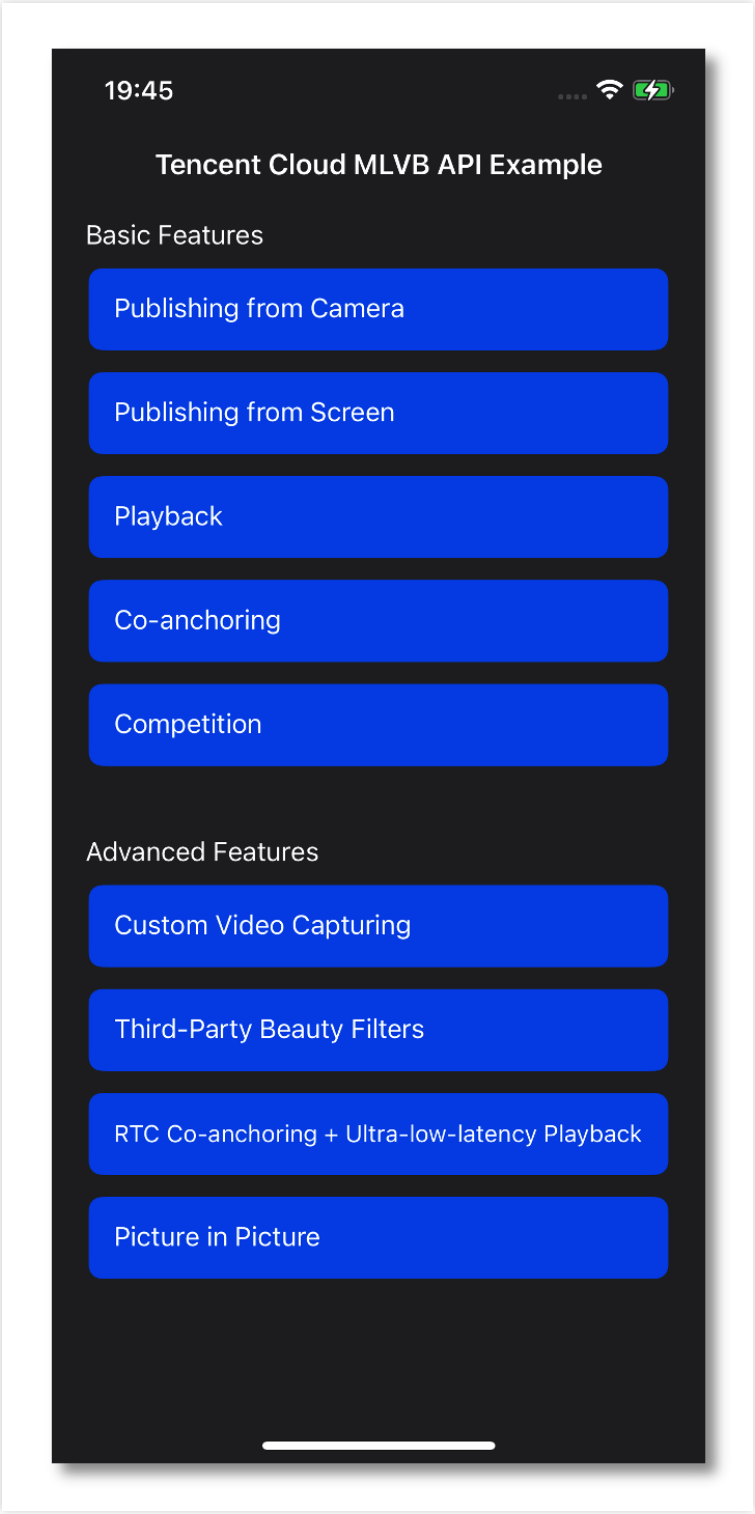
SECRETKEY：默认为空，请设置为实际的密钥信息。

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。


19:45
Tencent Cloud MLVB API Example
Basic Features
Publishing from Camera
Publishing from Screen
Playback
Co-anchoring
Competition
Advanced Features
Custom Video Capturing
Third-Party Beauty Filters
RTC Co-anchoring + Ultra-low-latency Playback
Picture in Picture

Android

最近更新时间：2024-01-13 15:53:49

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。

操作步骤

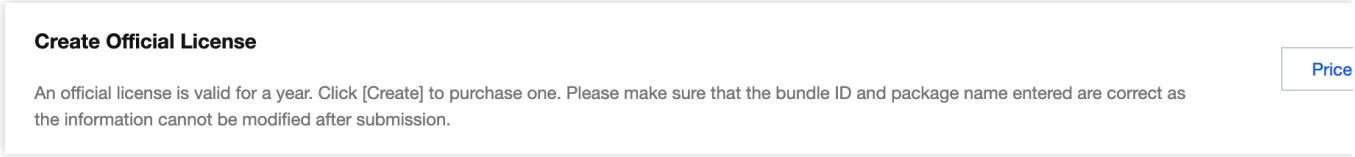
步骤一：下载 SDK 和 MLVB-API-Example 源码

1. 根据实际业务需求 下载 相应的压缩包，这里以 Live版本 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**创建License**。

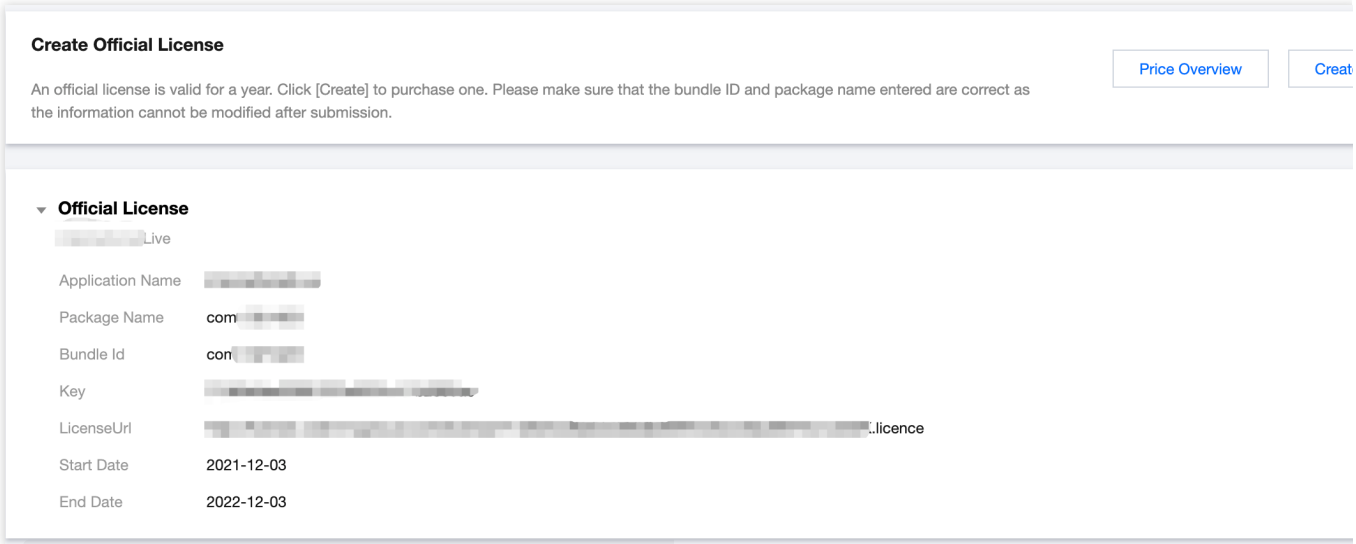


2. 根据实际需求填写 `App Name` 、 `Package Name` 和 `Bundle ID` ，勾选功能模块**直播**（直播推流 + 视频播放），单击**确定**。

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. 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。**



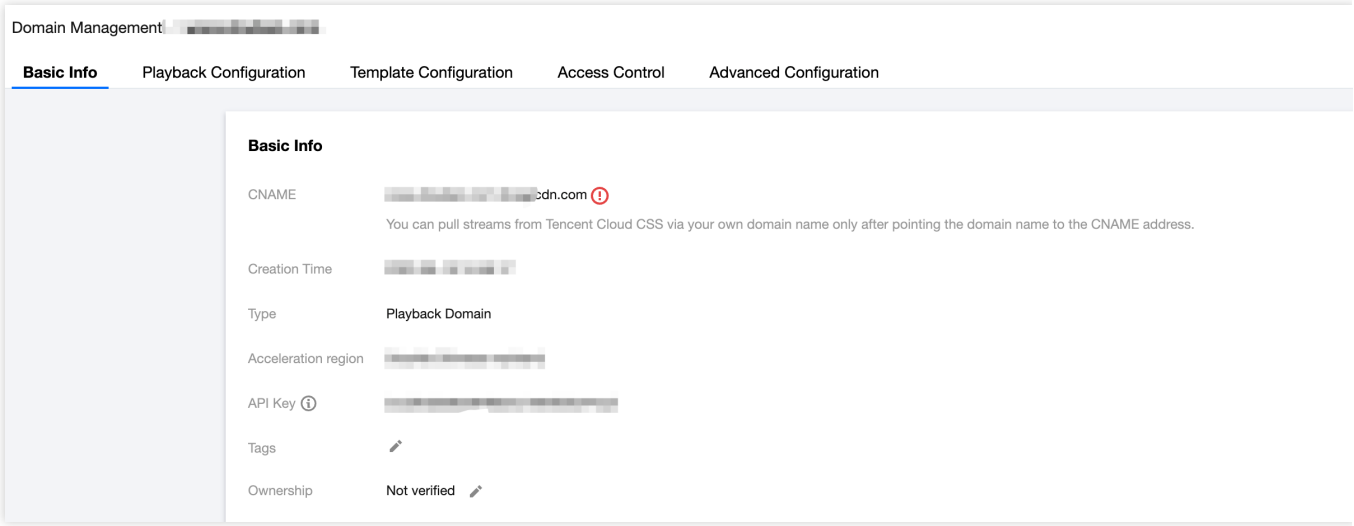
4. 打开 `LiteAVSDK_Live_Android_版本号/MLVB-API-Example/Debug/src/main/java/com/tencent/mlvb/debug/GenerateTestUserSig.java` 文件，设置 `GenerateTestUserSig.java` 文件中的相关参数：

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_Live_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**。

配置推流参数

1. 找到并打开 `LiteAVSDK_Live_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。

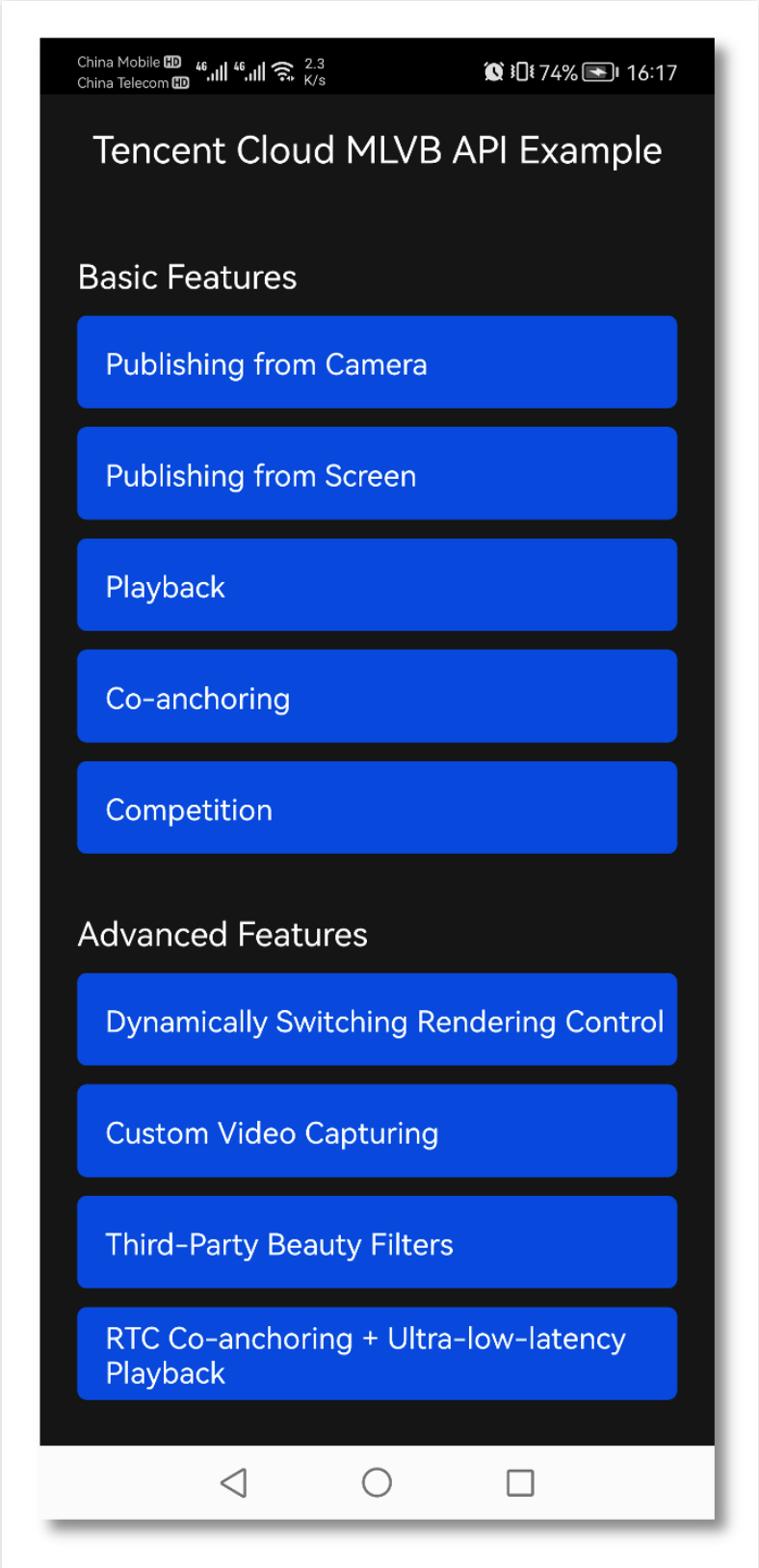
SECRETKEY：默认为 PLACEHOLDER，请设置为实际的密钥信息。

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。


China Mobile HD
China Telecom HD
2.3 K/s
74% 16:17
Tencent Cloud MLVB API Example
Basic Features
Publishing from Camera
Publishing from Screen
Playback
Co-anchoring
Competition
Advanced Features
Dynamically Switching Rendering Control
Custom Video Capturing
Third-Party Beauty Filters
RTC Co-anchoring + Ultra-low-latency Playback

SDK 集成
iOS

最近更新时间：2024-01-13 15:53:49

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 Live版 SDK 为例：

开发环境要求

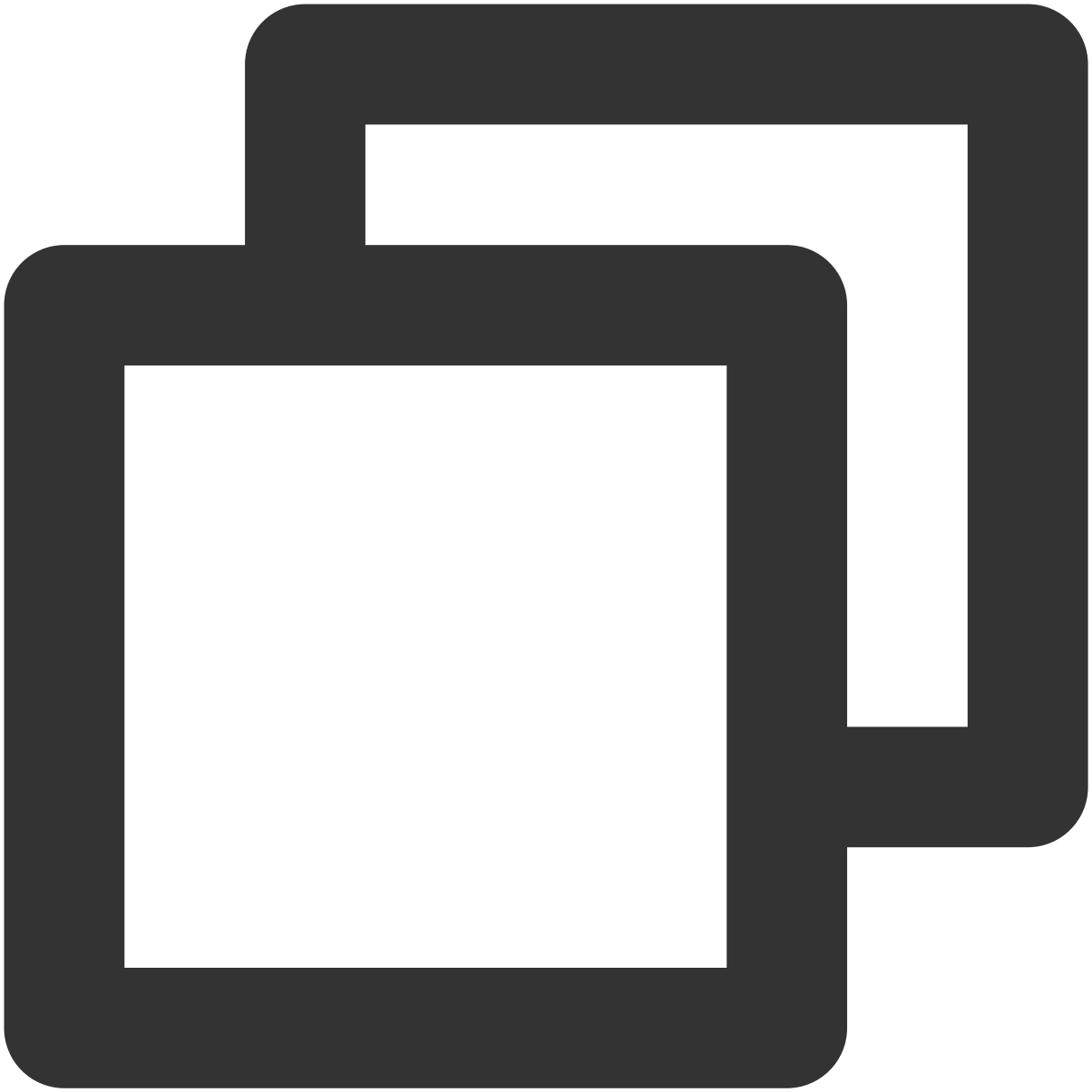
Xcode 9.0+。
iOS 9.0 以上的 iPhone 或者 iPad 真机。
项目已配置有效的开发者签名。

集成 LiteAVSDK

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

CocoaPods

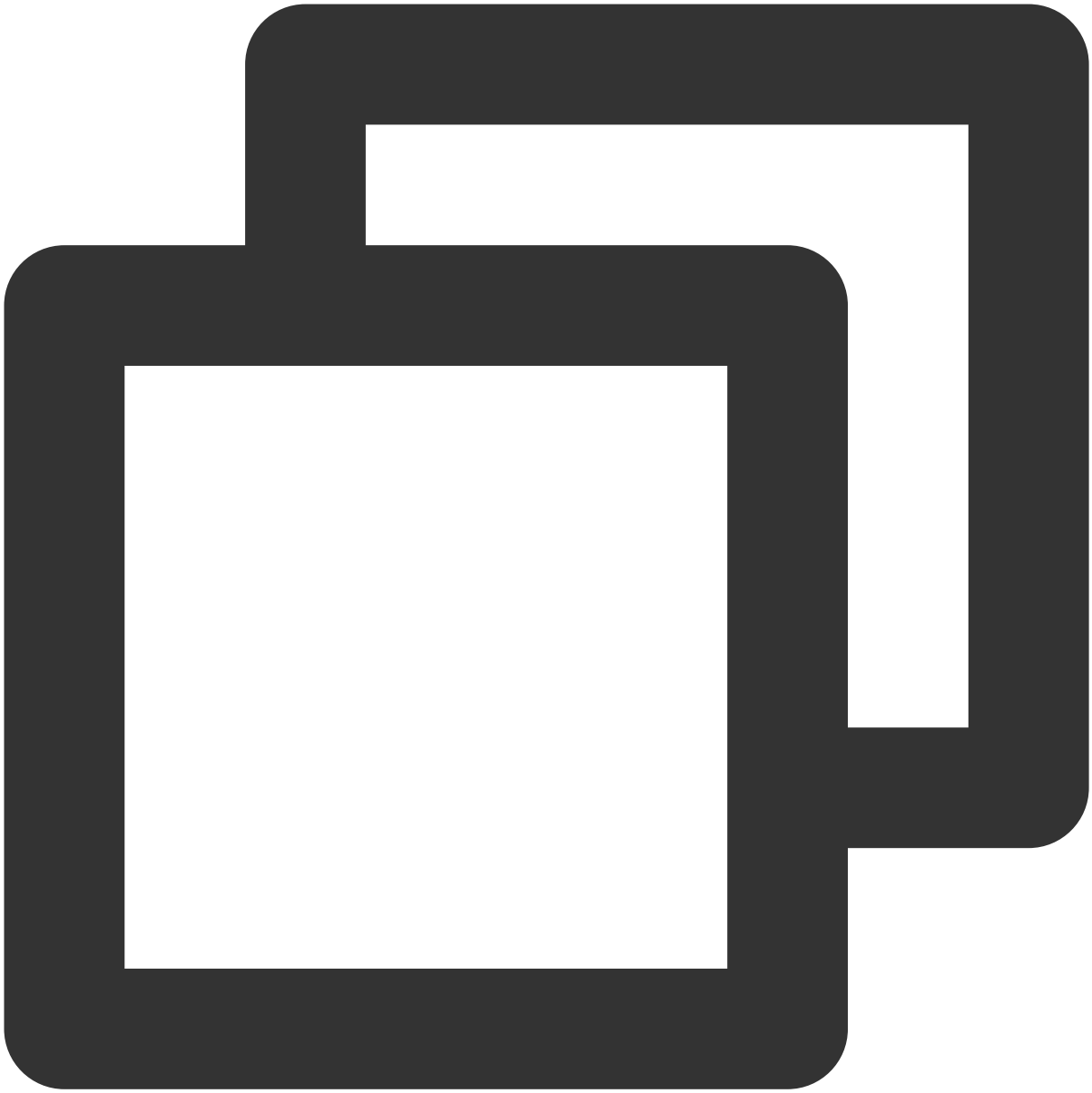
1. **安装 CocoaPods**
在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：



```
sudo gem install cocoapods
```

2. 创建 Podfile 文件

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

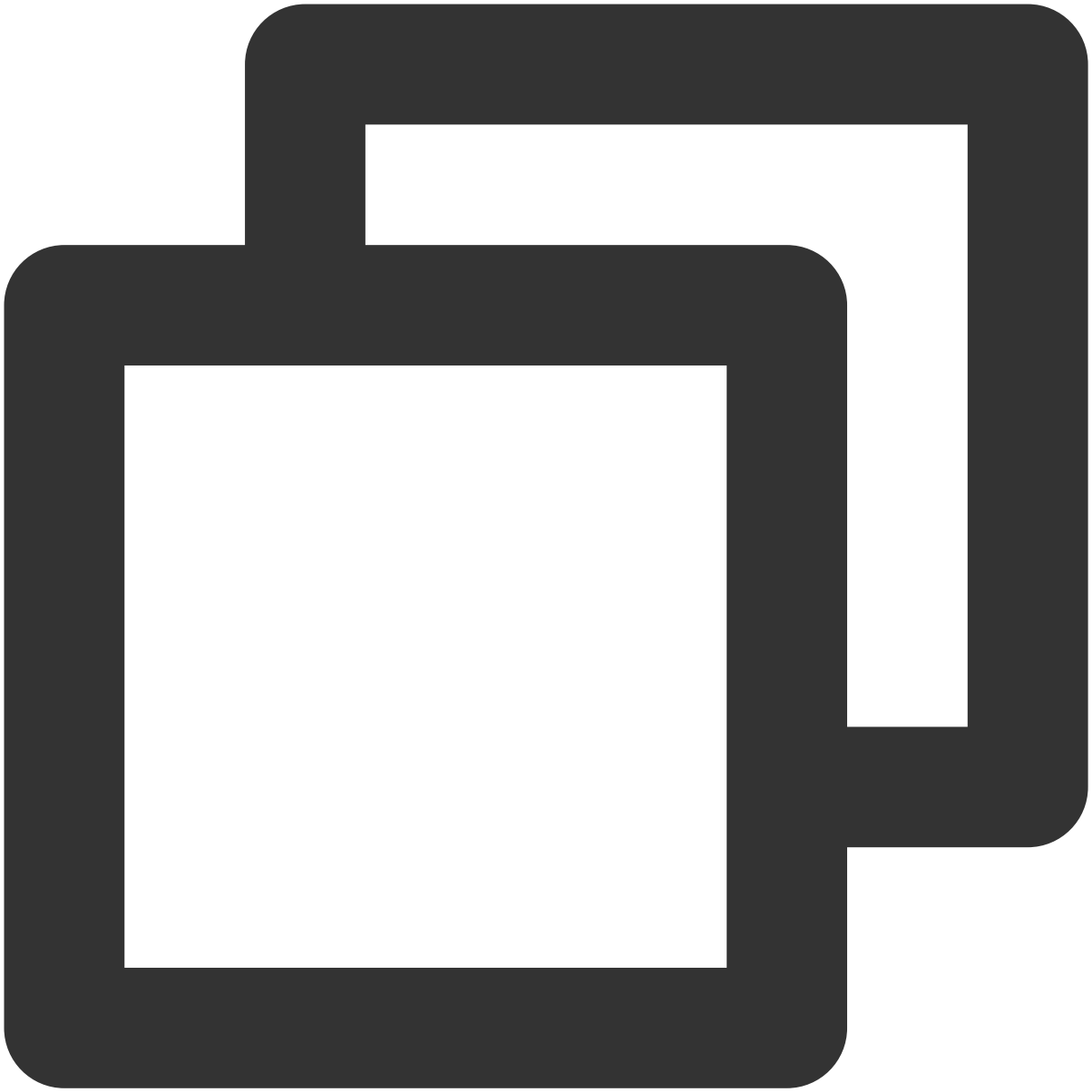


```
pod init
```

3. **编辑 Podfile 文件**

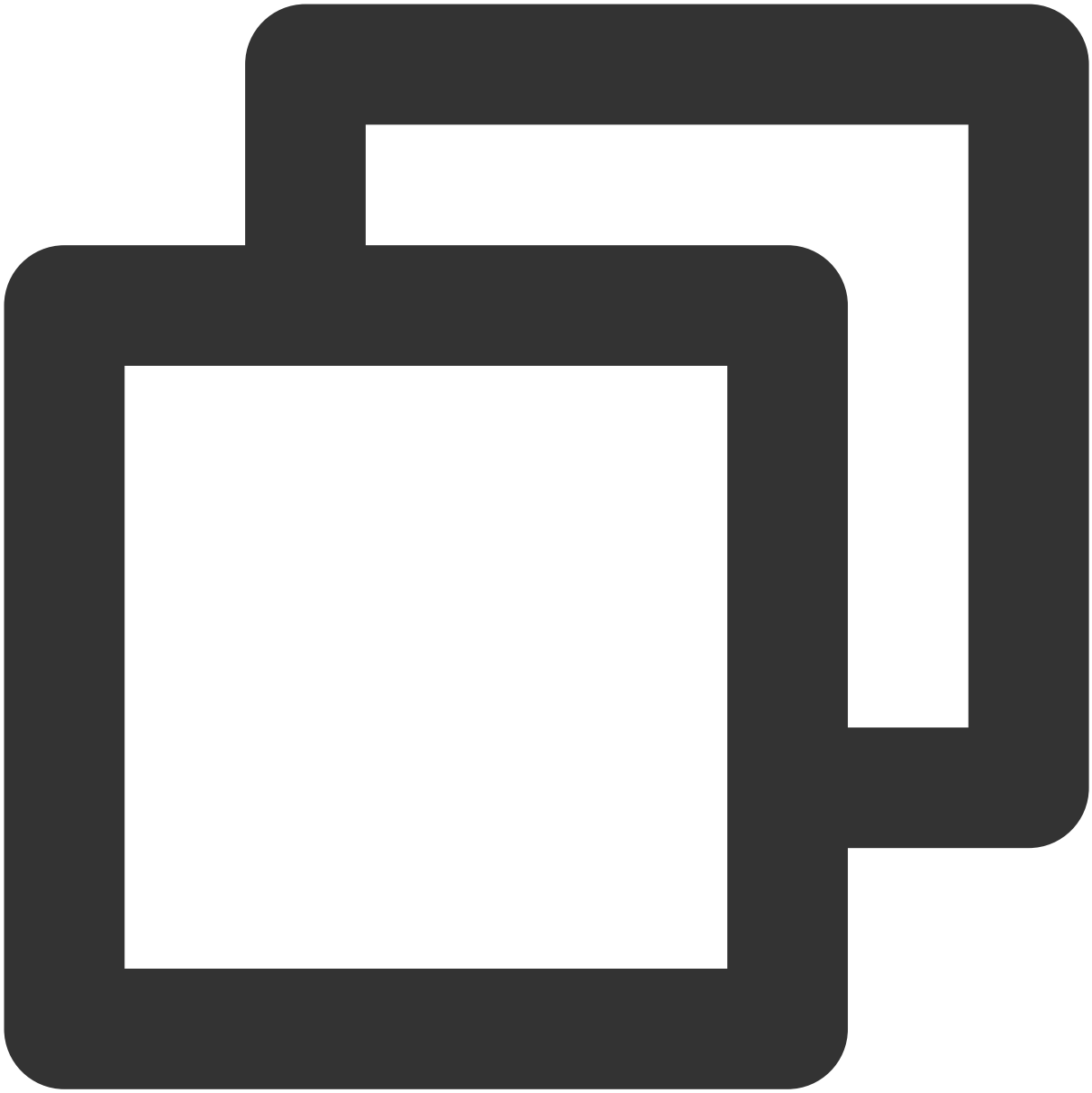
编辑 Podfile 文件，有如下有两种设置方式：

方式一：使用腾讯云 LiteAVSDK 的 podspec 文件路径。



```
platform :ios, '9.0'

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

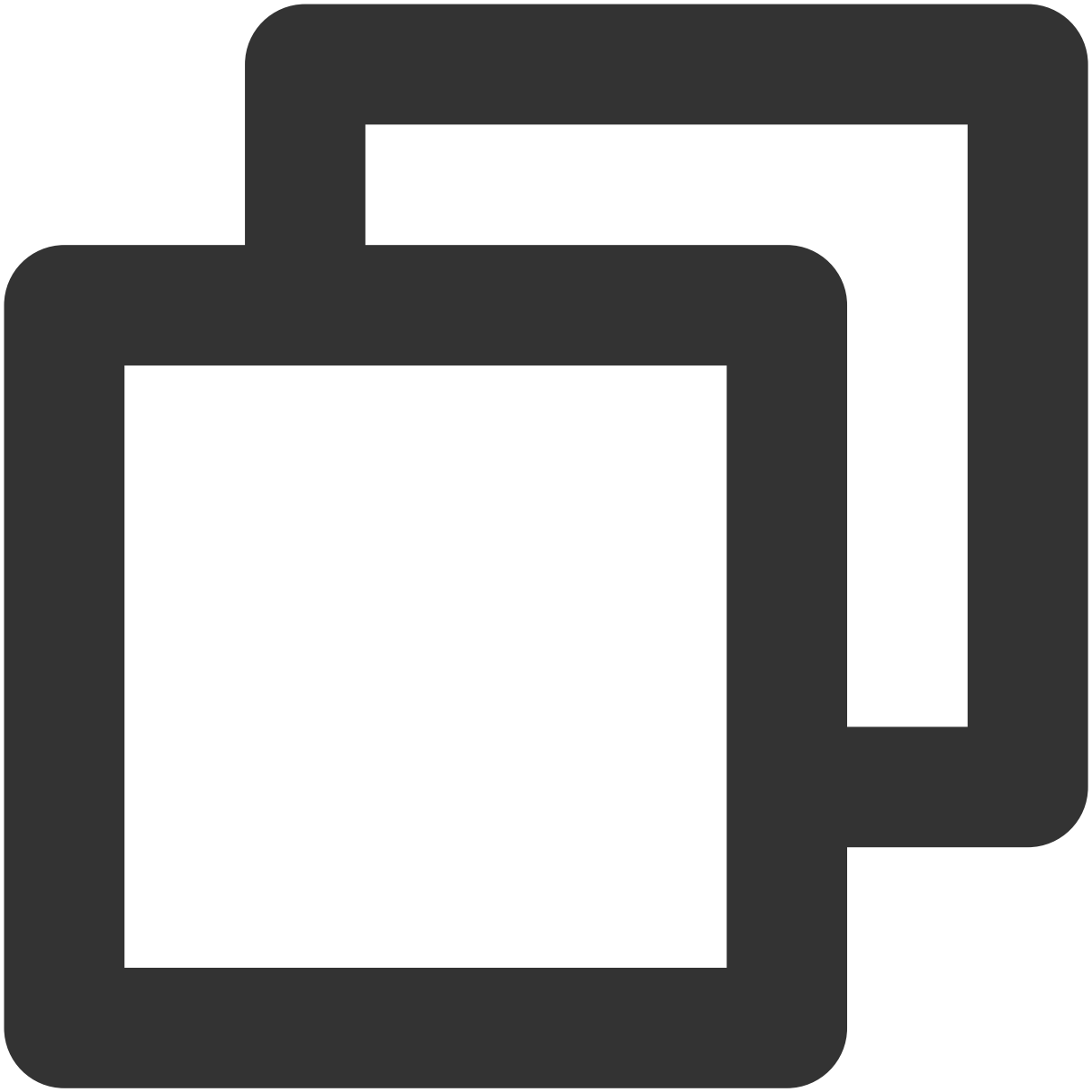
方式二：使用 CocoaPod 官方源，支持选择版本号。



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

target 'App' do
pod 'TXLiteAVSDK_Live'
end
```

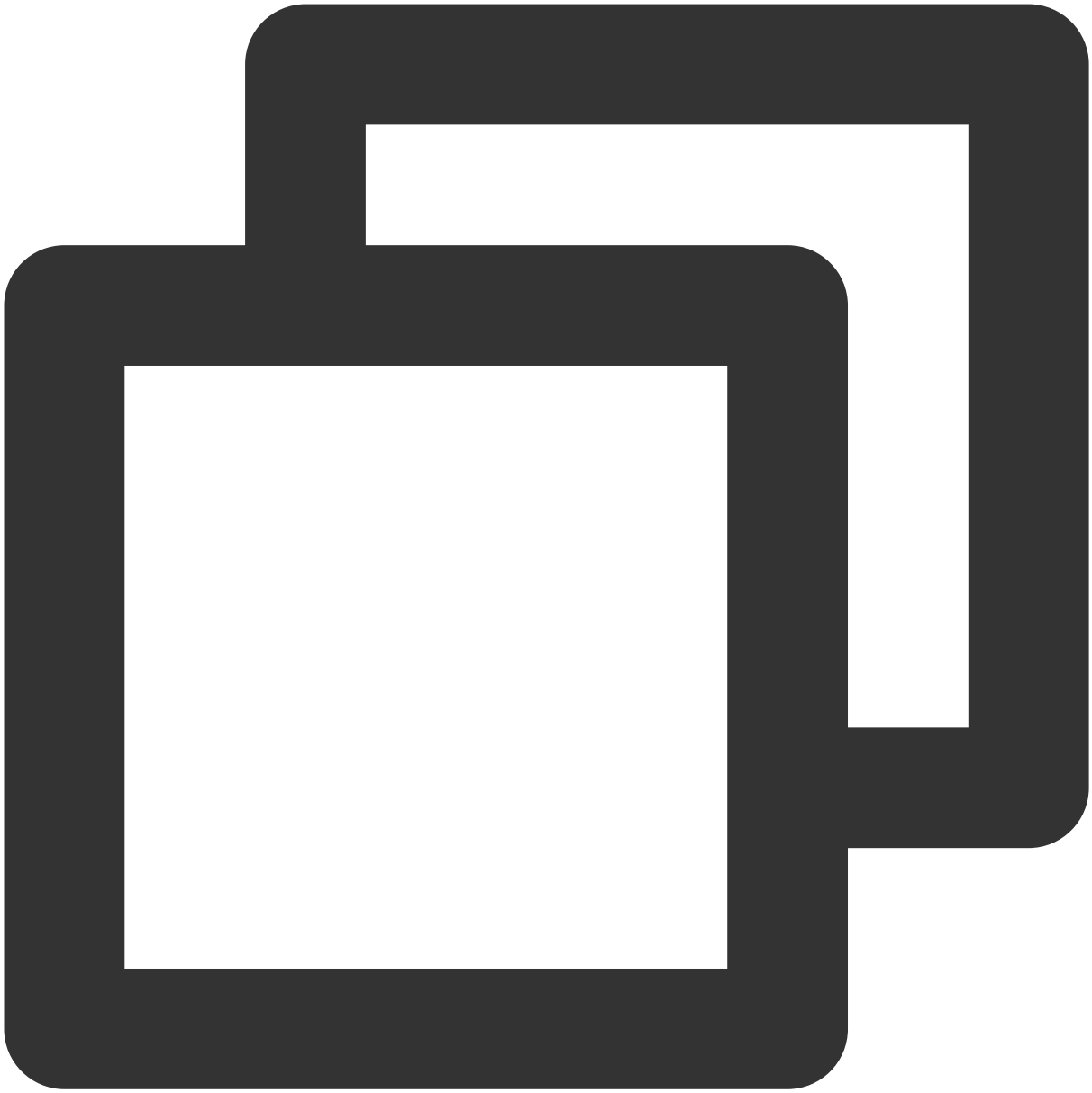
4. 更新并安装 SDK

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：

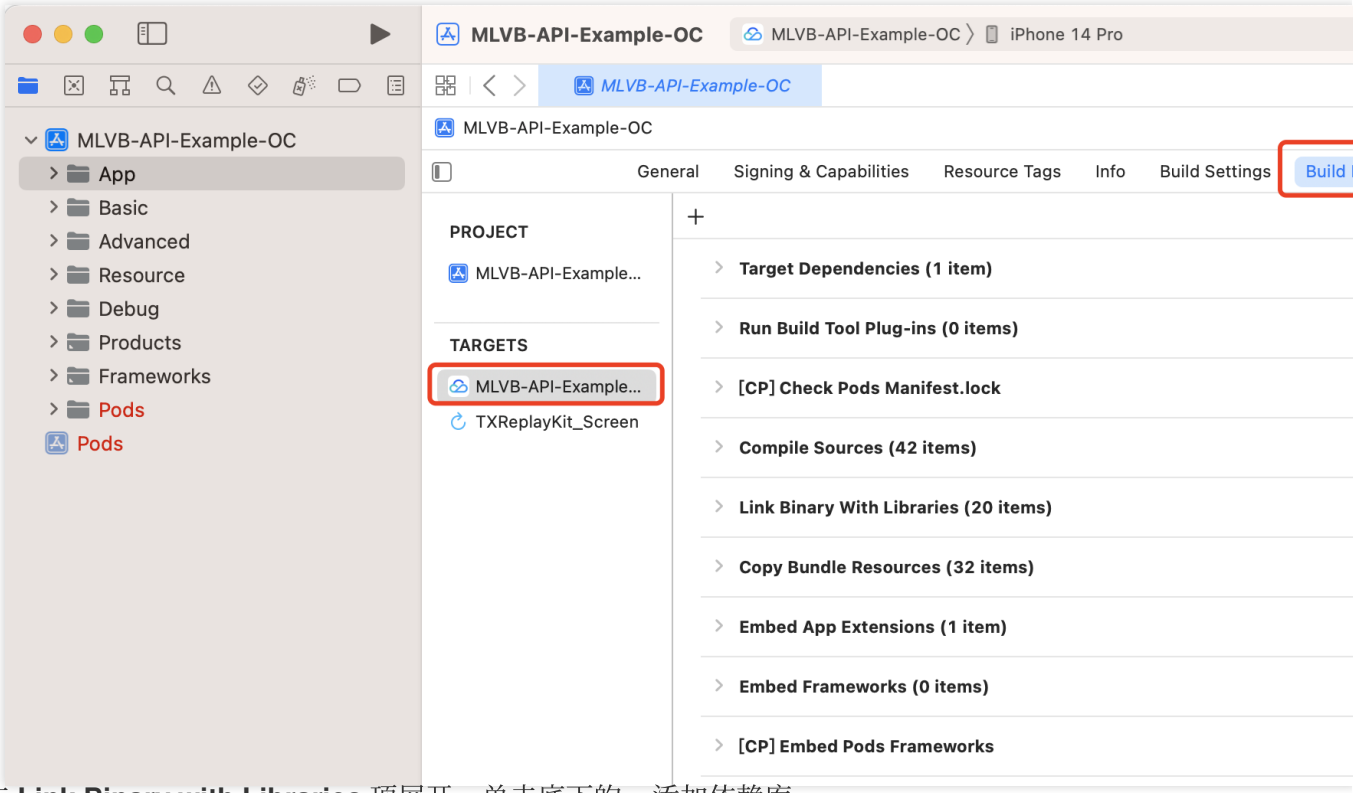


```
pod update
```

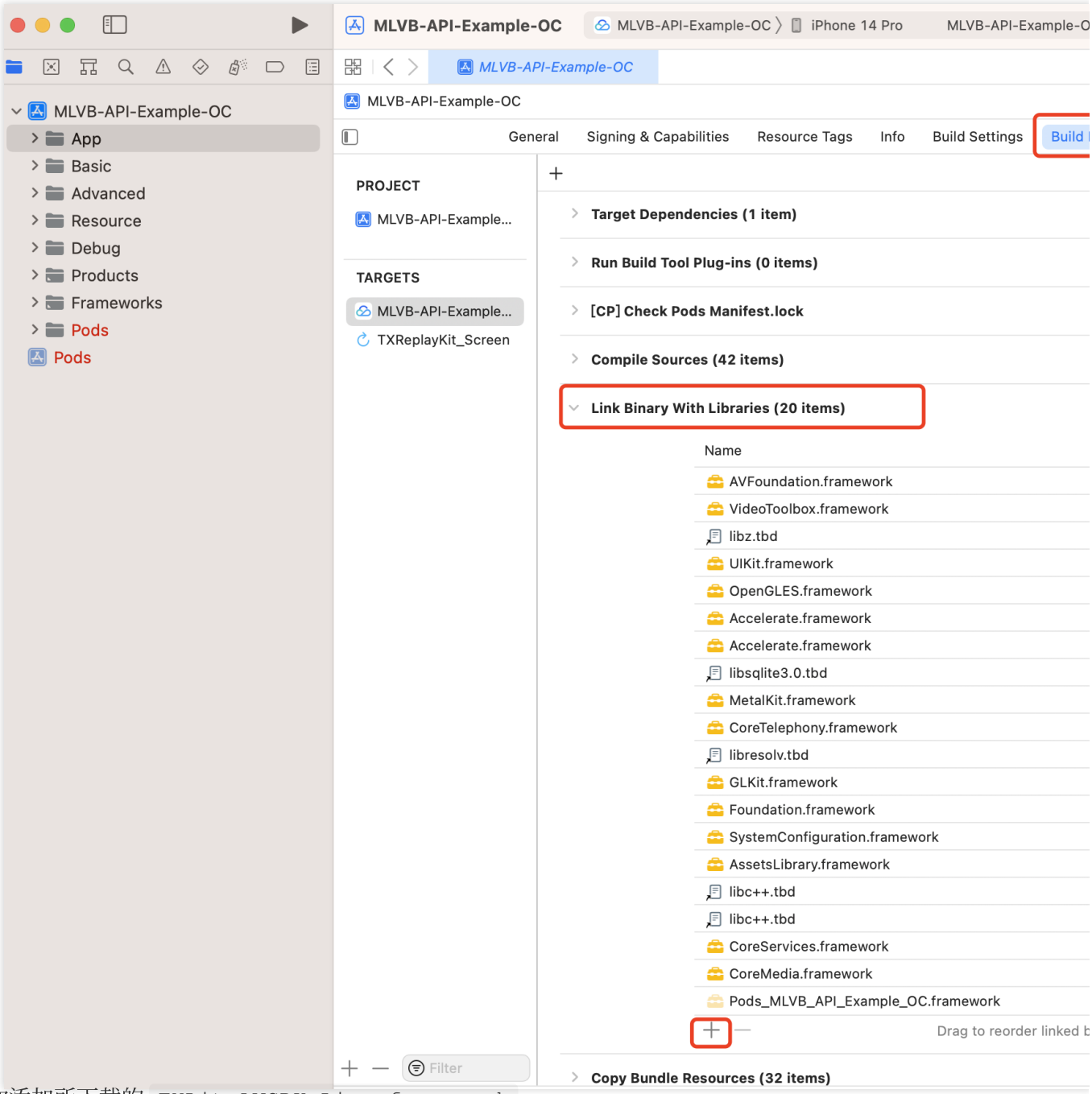
pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

手动集成

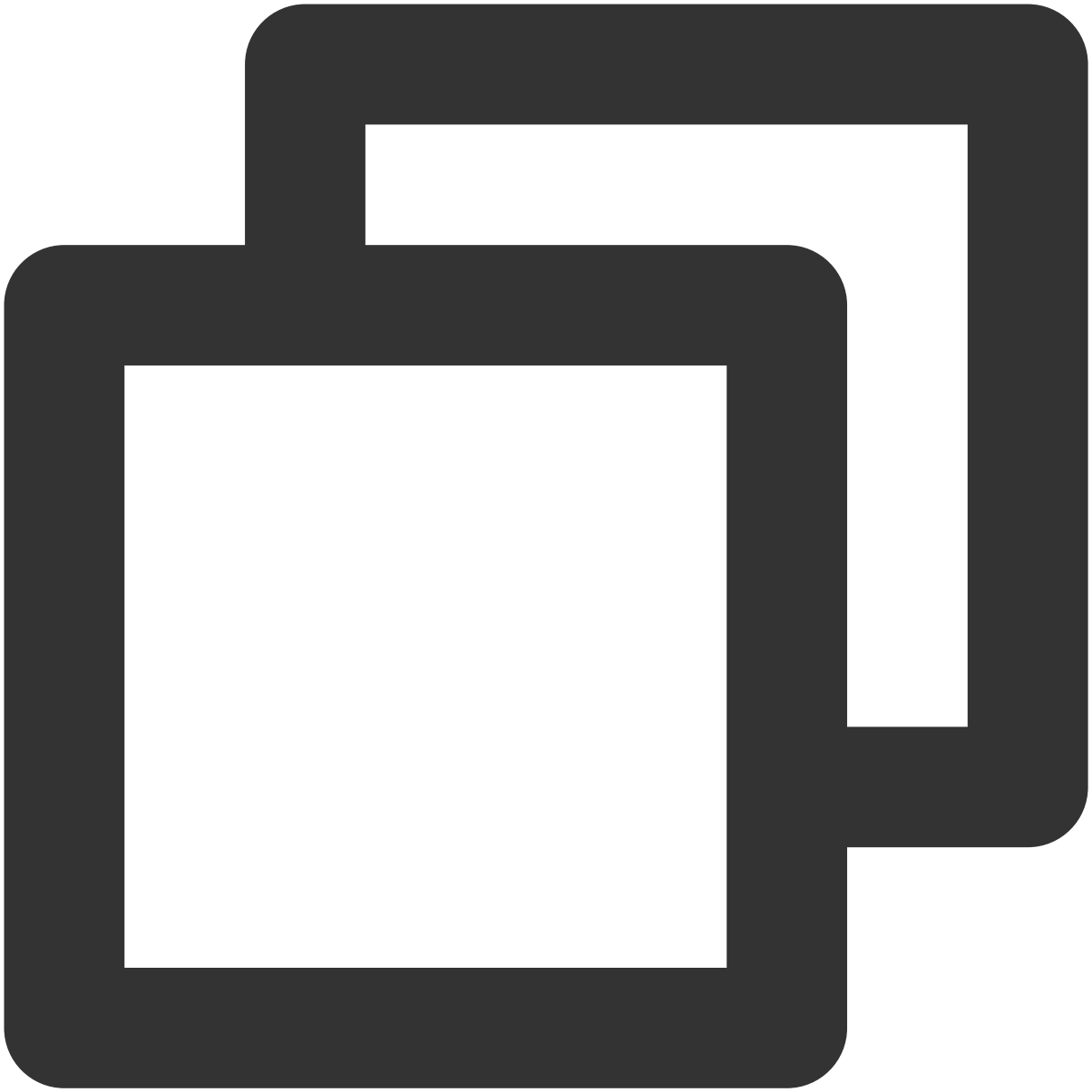
1. 下载 LiveAVSDK ， 下载完成后进行解压。
2. 打开您的 Xcode 工程项目，选择要运行的 target , 选中**Build Phases**项。



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

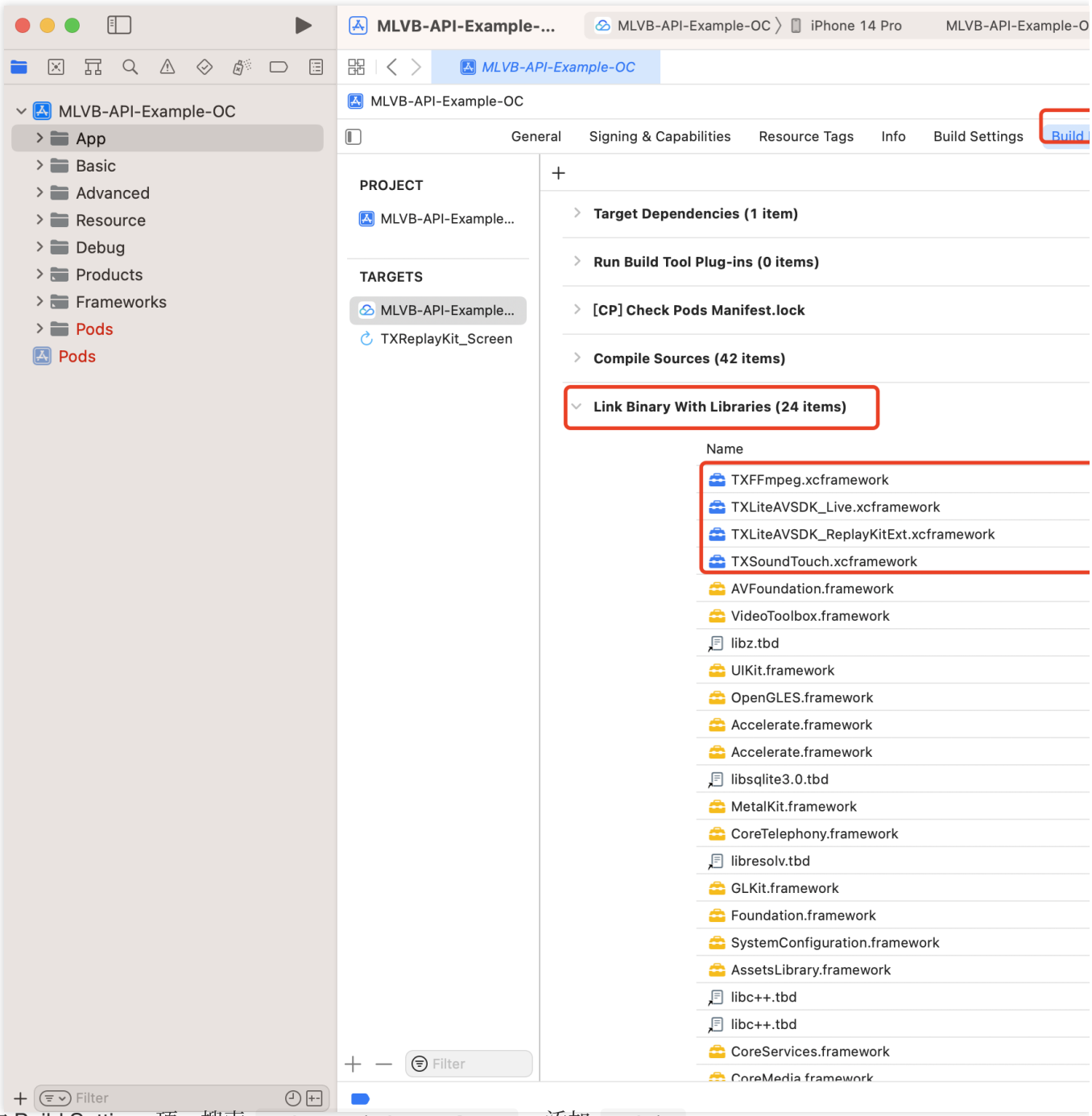


4. 依次添加所下载的 `TXLiteAVSDK_Live.framework` 、 `TXFFmpeg.xcframework` 、 `TXSoundTouch.xcframework` 及其所需依赖库：

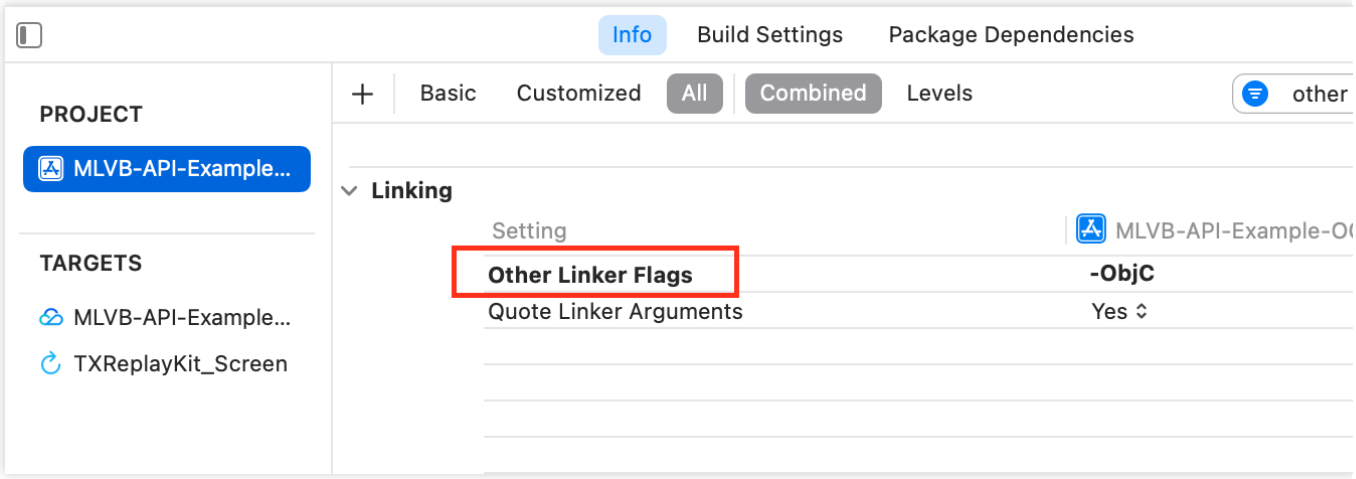


```
AVFoundation.framework
VideoToolbox.framework
libz.tbd
OpenGLES.framework
Accelerate.framework
libsqlite3.0.tbd
MetalKit.framework
CoreTelephony.framework
libresolv.tbd
GLKit.framework
Foundation.framework
```

```
SystemConfiguration.framework
AssetsLibrary.framework
libc++.tbd
CoreServices.framework
CoreMedia.framework
```



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

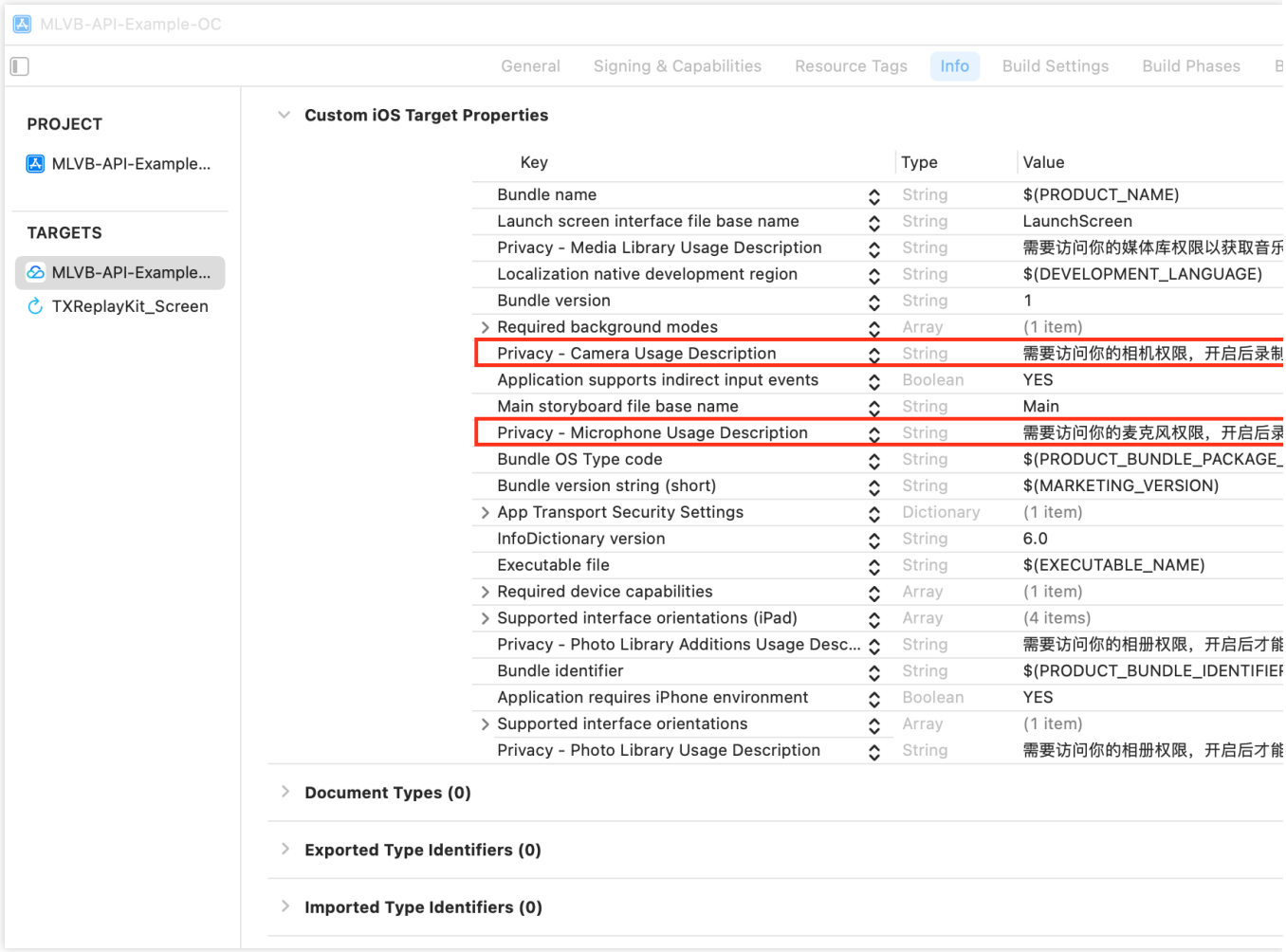


授权摄像头和麦克风使用权限

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。

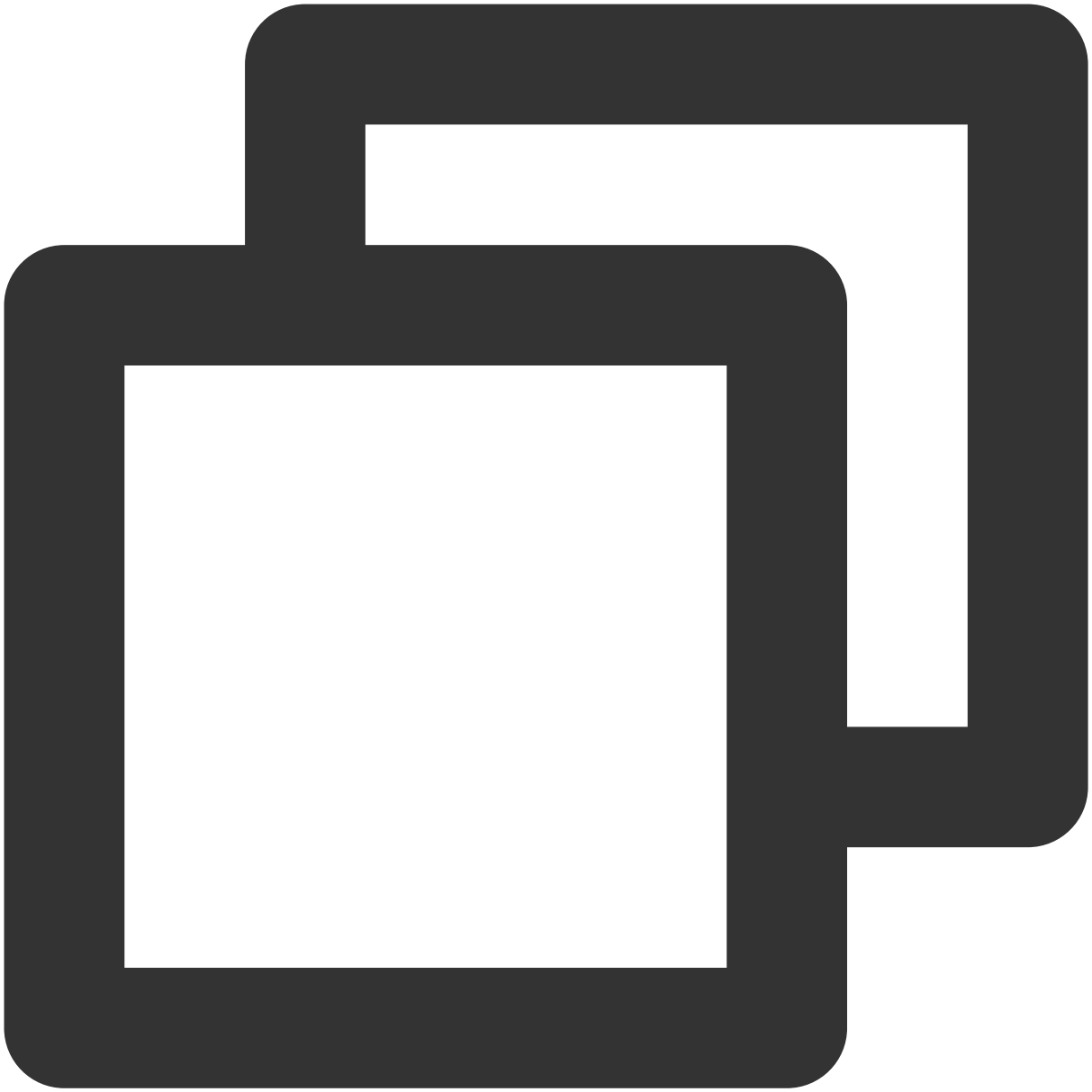
Privacy - Camera Usage Description，并填入摄像头使用目的提示语。



在工程中引入 SDK

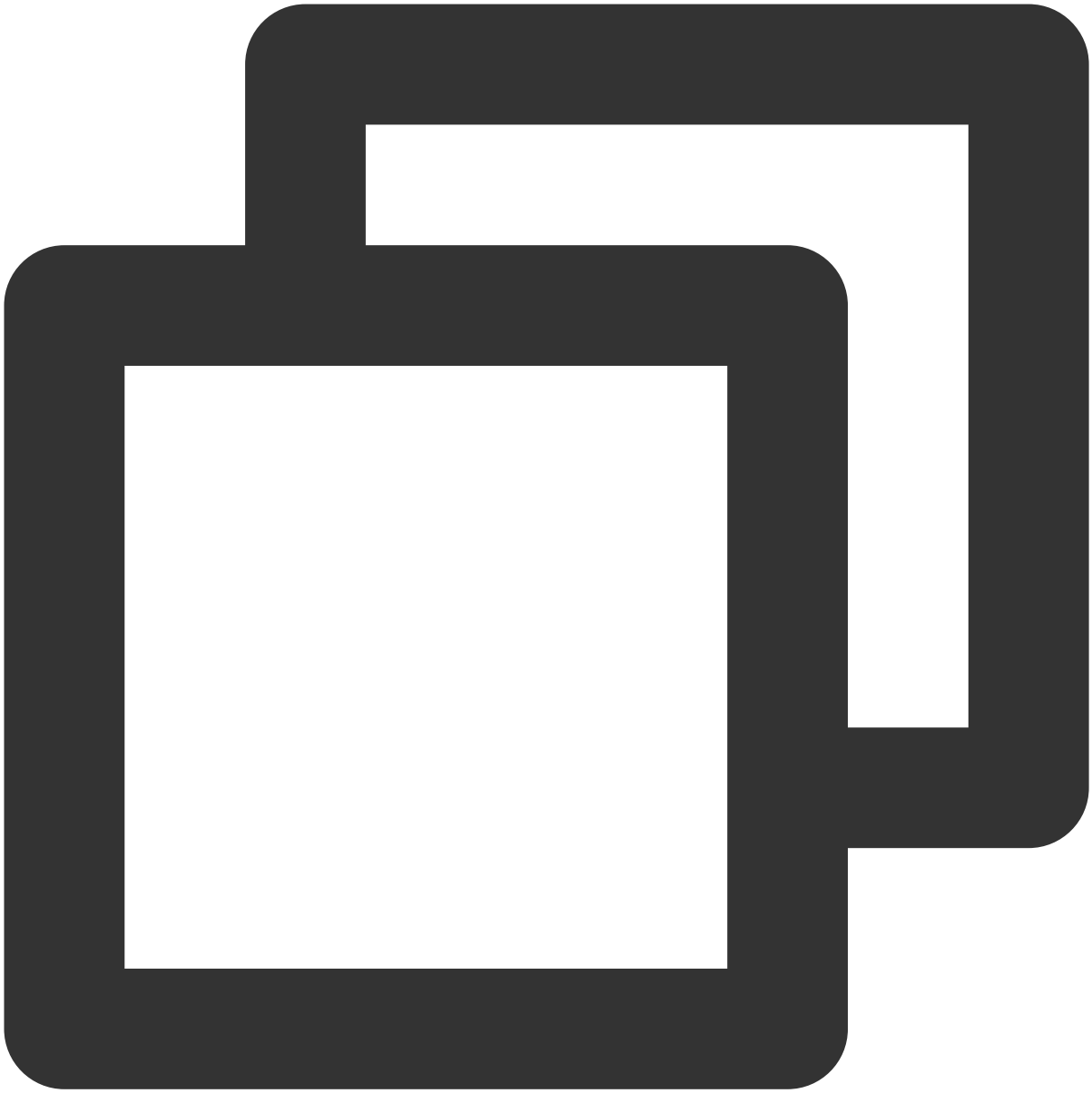
项目代码中使用 SDK 有两种方式：

方式一： 在项目需要使用 SDK API 的文件里，添加模块引用。



```
@import TXLiteAVSDK_Live;
```

方式二：在项目需要使用 SDK API 的文件里，引入具体的头文件。

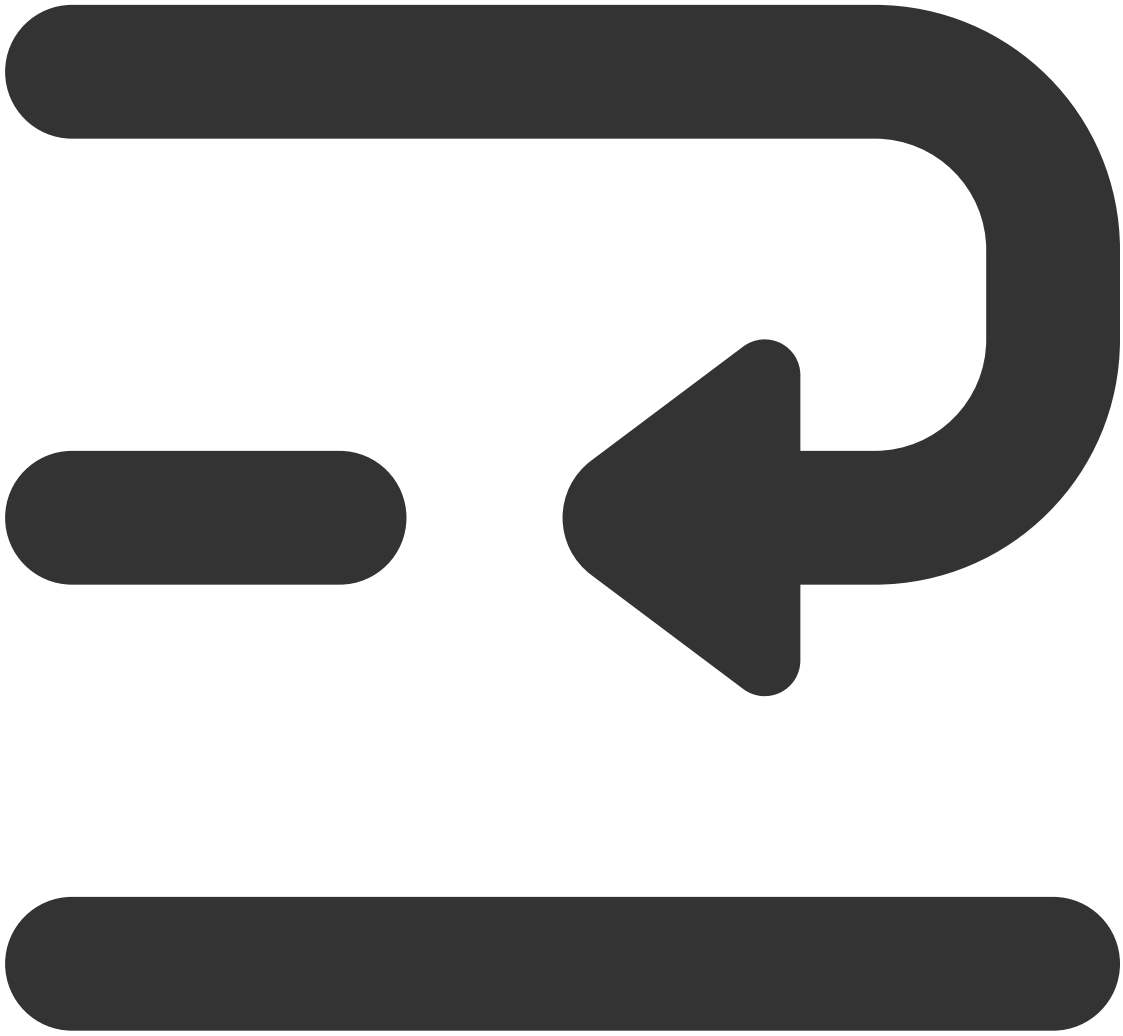


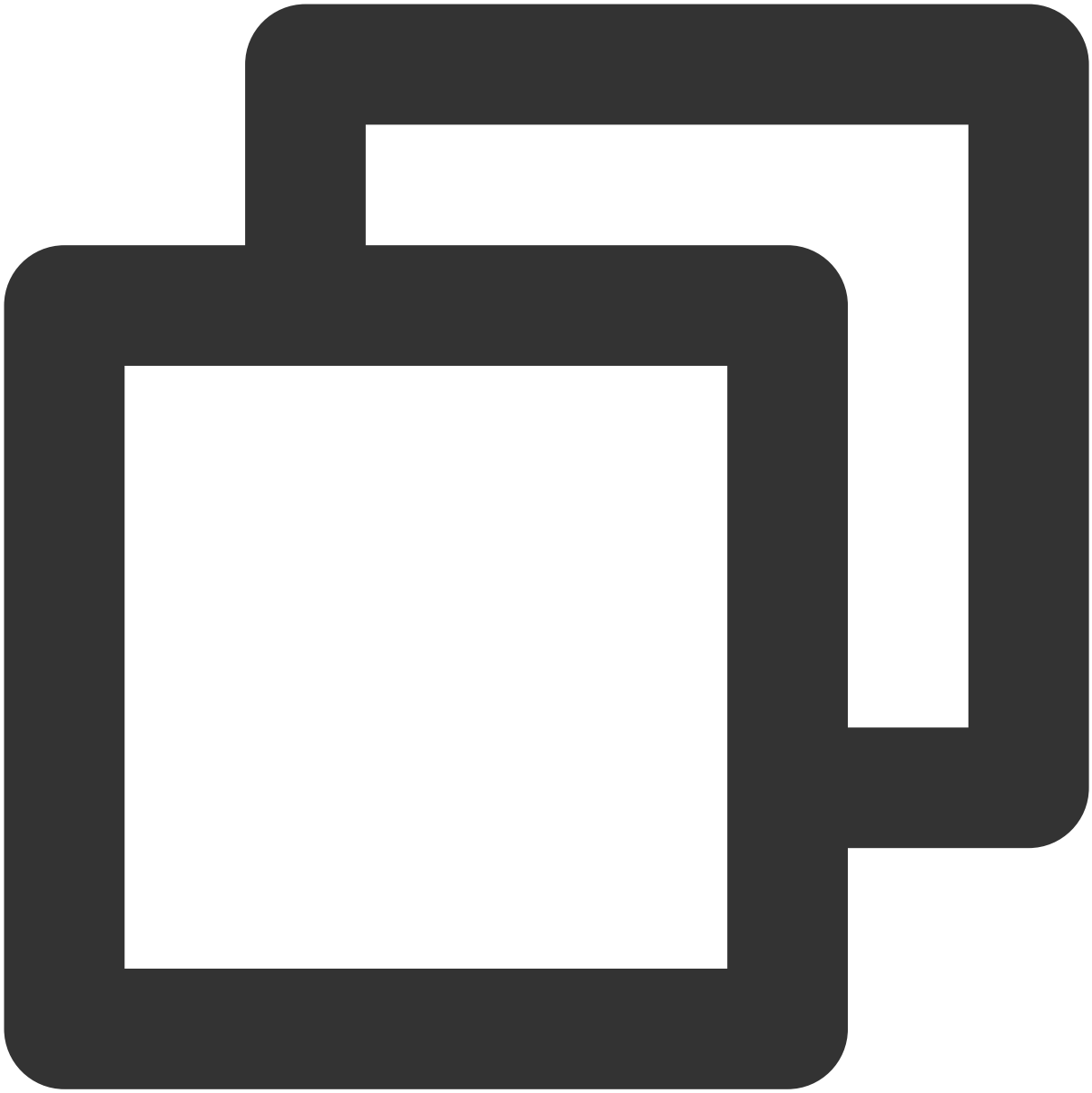
```
#import "TXLiteAVSDK_Live/TXLiteAVSDK.h"
```

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_Live;
@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
  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 setEnvironment:@"GDPR"]; // 设置环境
  [V2TXLivePremier setLicence:licenceURL key:licenceKey];
  [V2TXLivePremier setObserver:self];
  NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
```

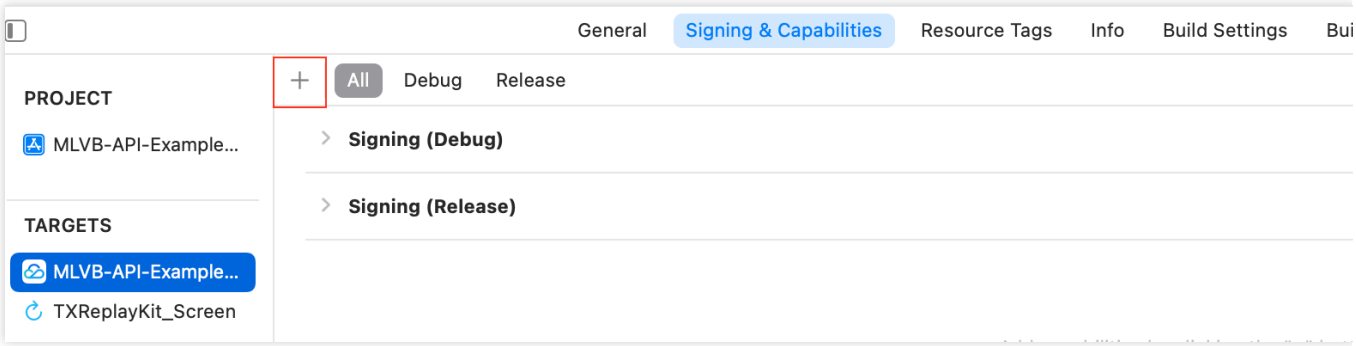
```
}
@end
```

常见问题

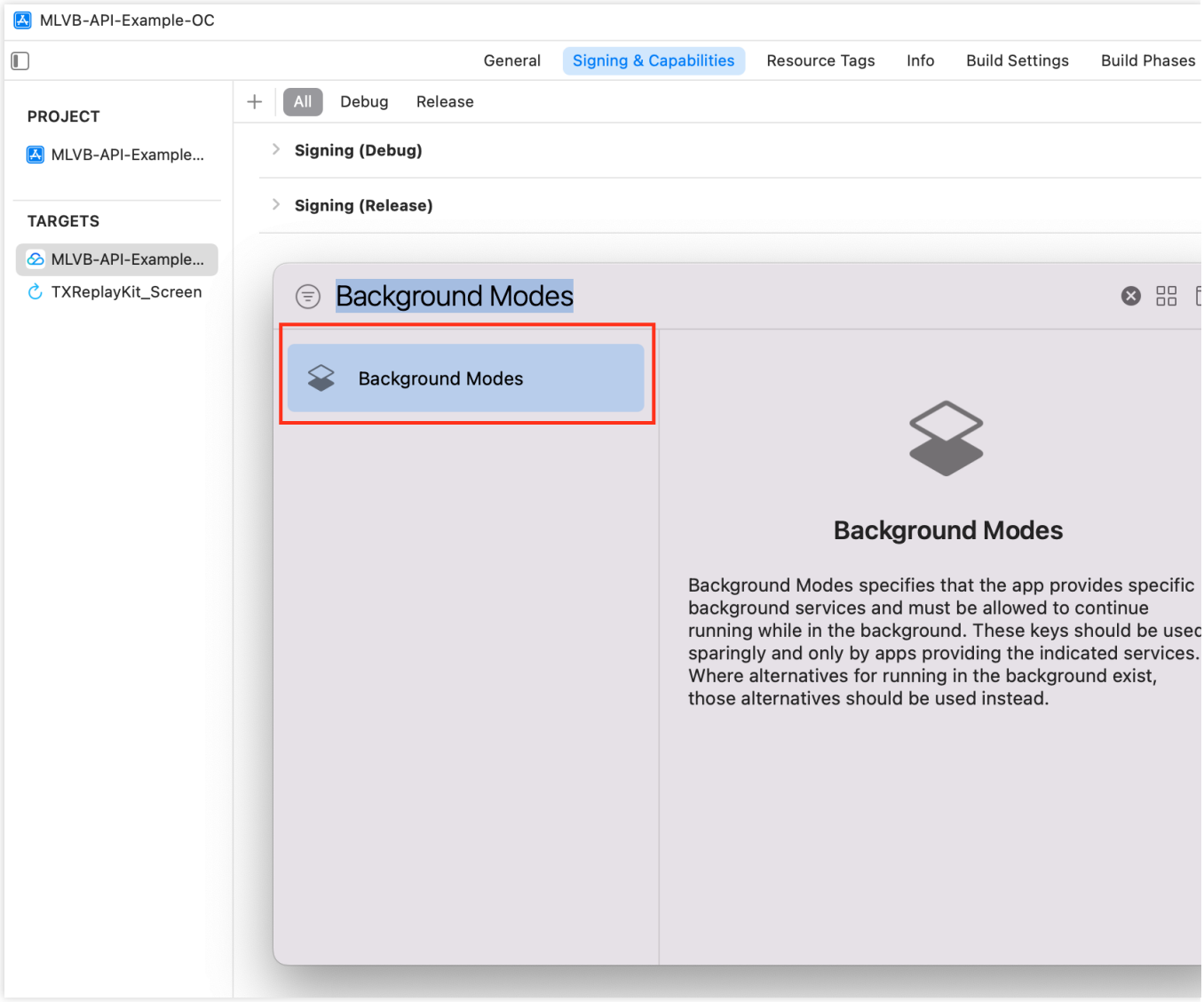
1. LiteAVSDK 是否支持后台运行？

支持，如需要进入后台仍然运行相关功能，操作如下：

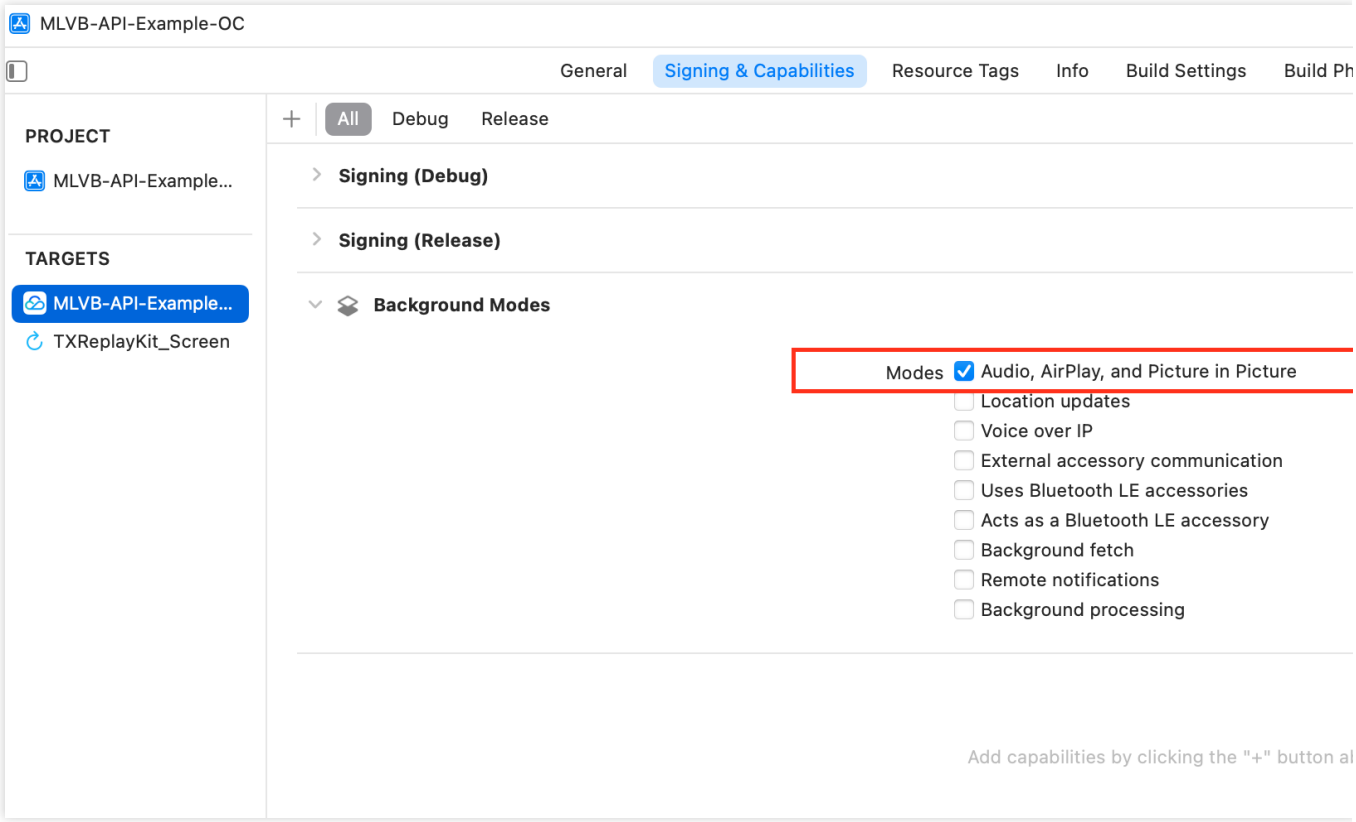
1. 选中当前工程项目，选择 **Signing&Capabilities**，单击左上角**+**，如图所示：



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：2024-01-13 15:53:49

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 Live版 SDK 为例：

开发环境要求

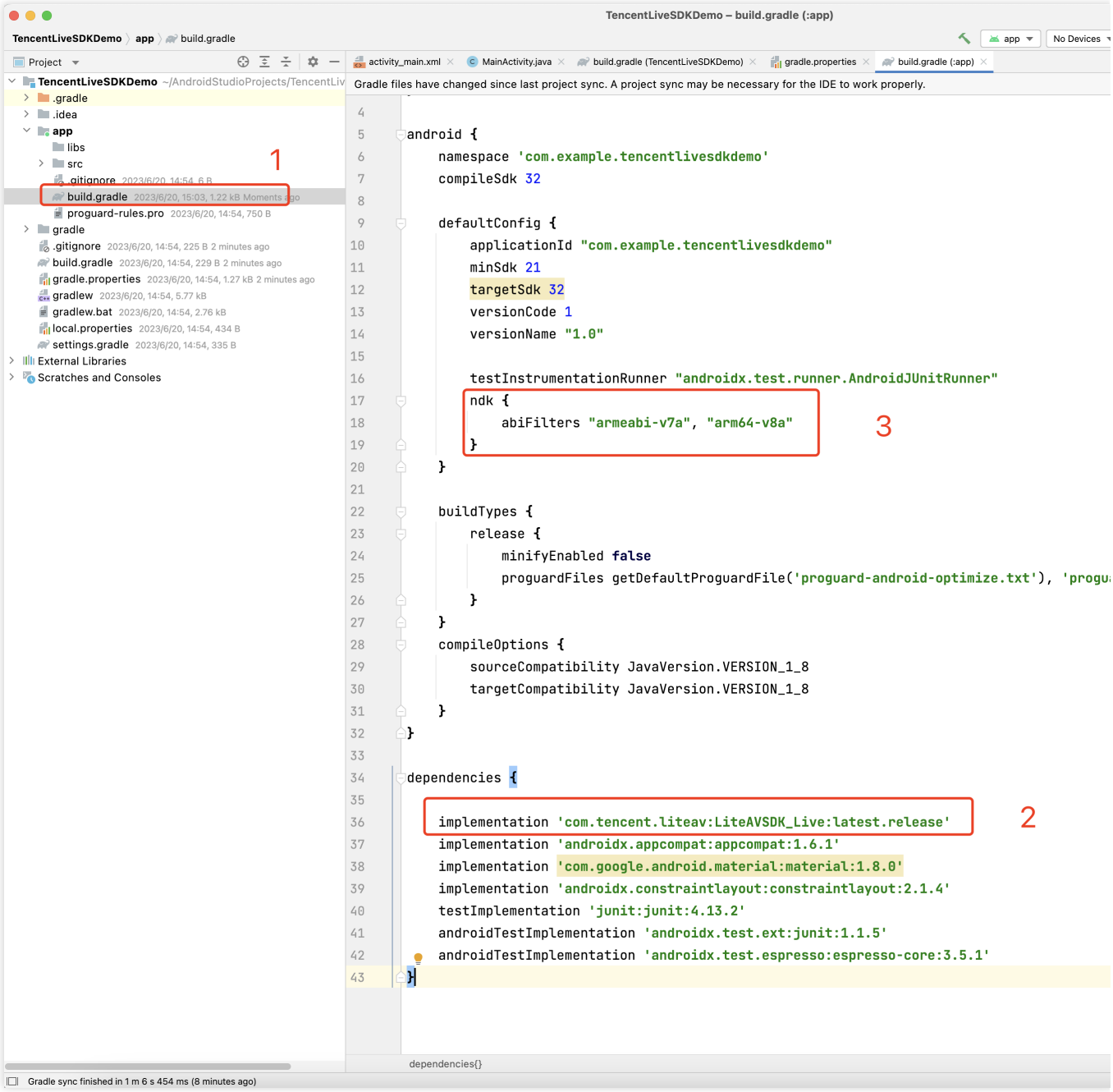
Android Studio 3.5+。
Android 4.1（SDK API 16）及以上系统。

集成 SDK（aar）

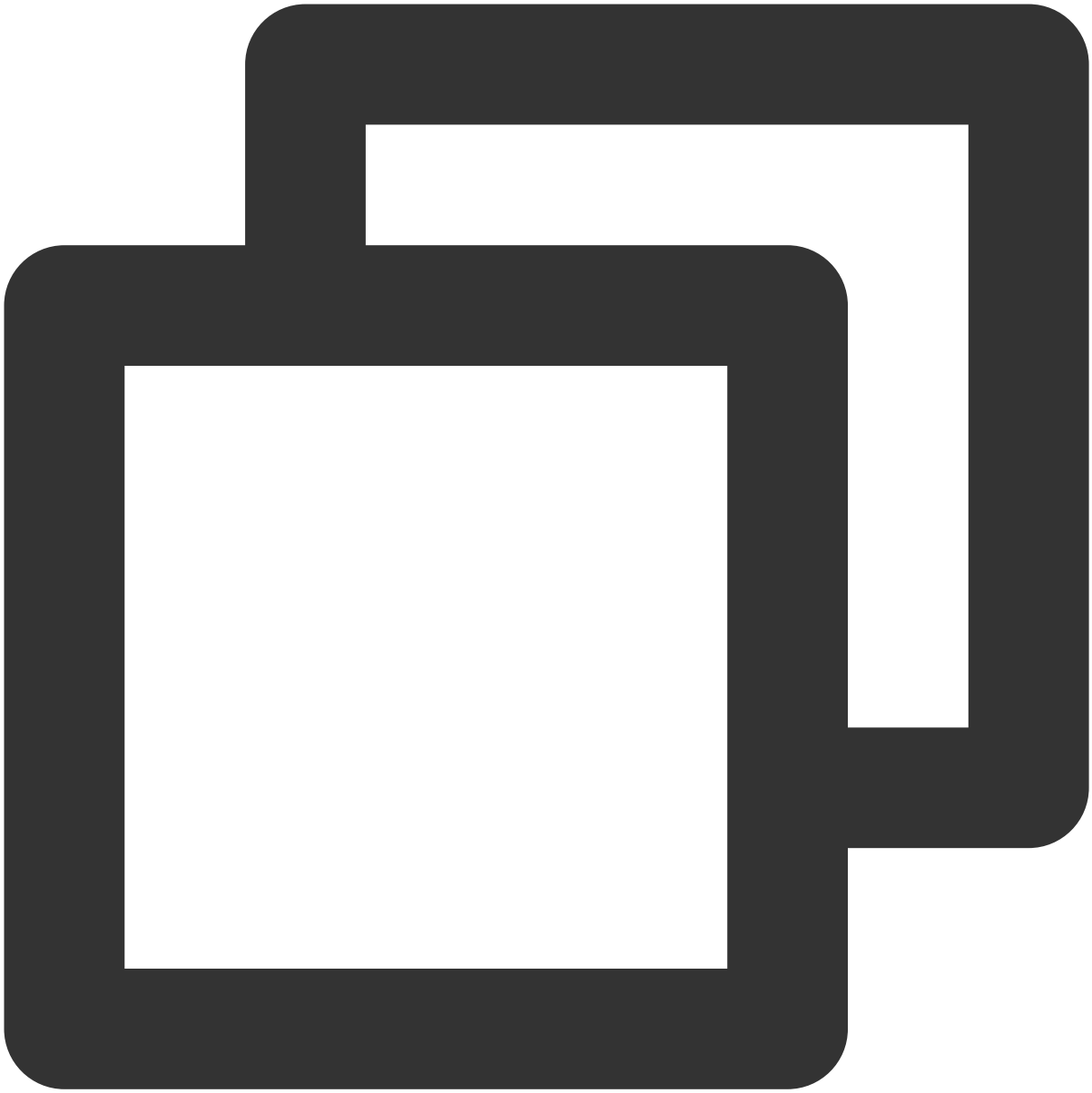
您可以选择使用 Gradle 自动加载的方式，或者手动下载 aar 再将其导入到您当前的工程项目中。

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：

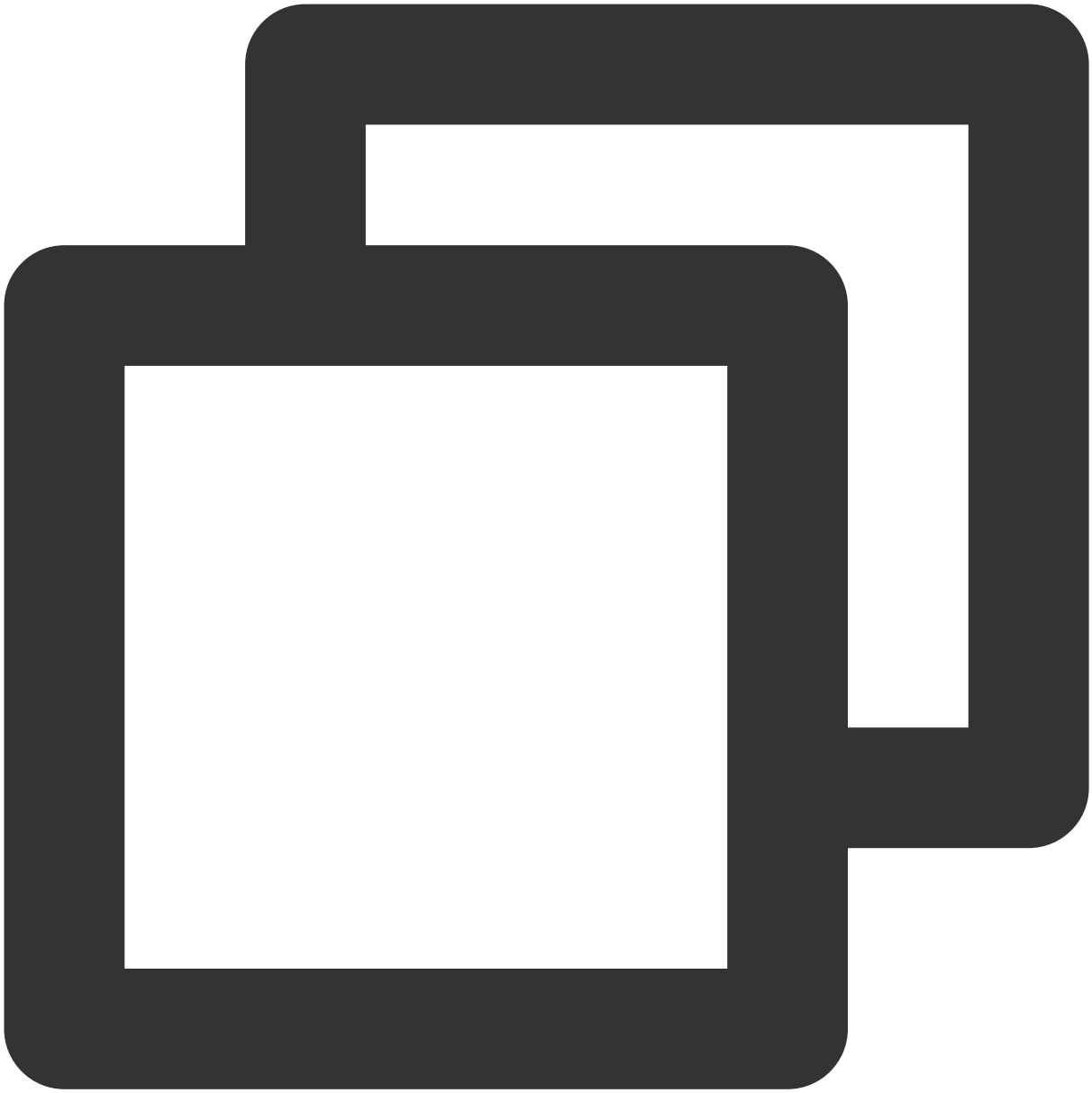


1. 打开 app 下的 build.gradle。
2. 在 dependencies 中添加 LiteAVSDK 的依赖。



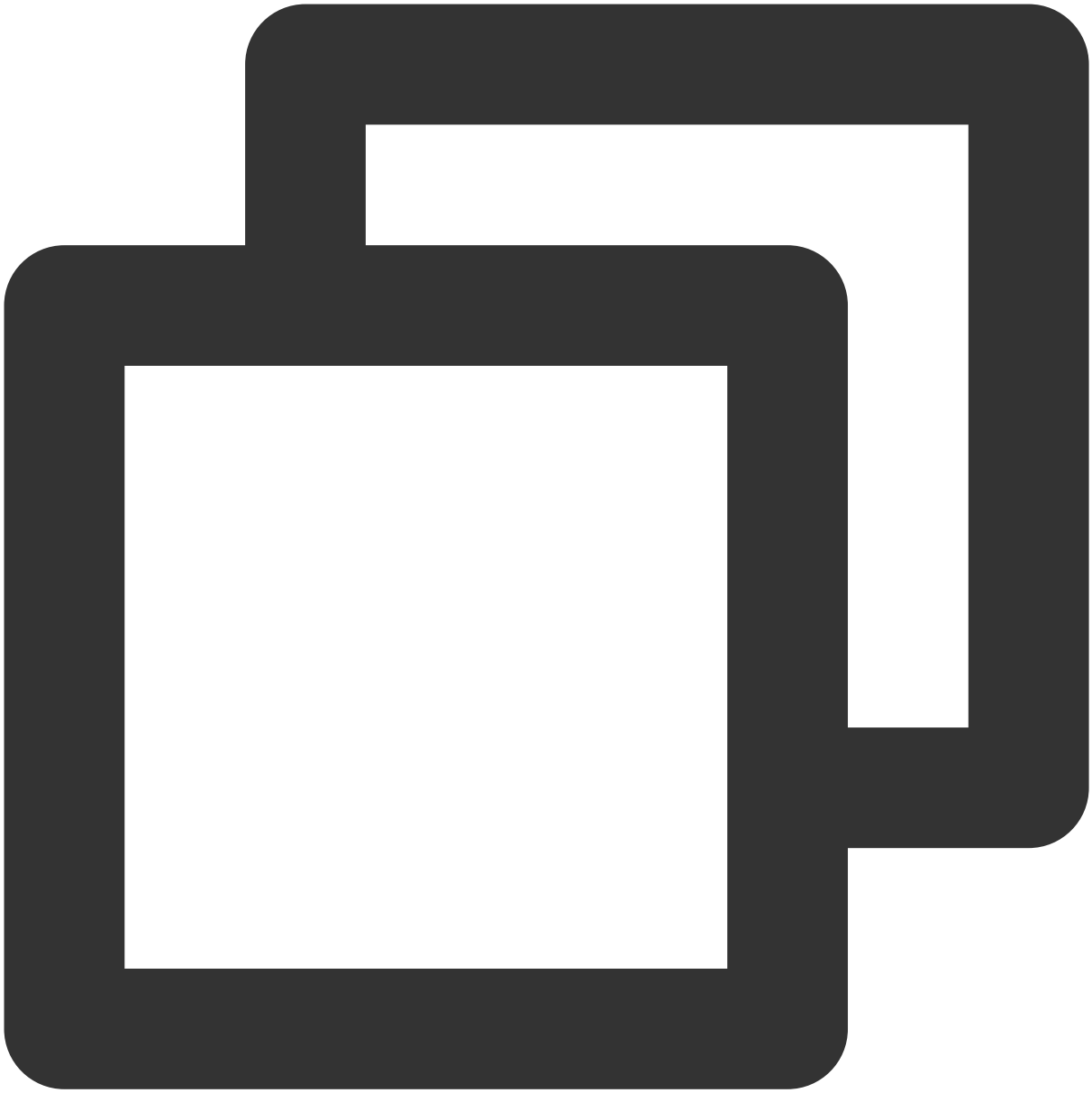
```
dependencies {
    implementation 'com.tencent.liteav:LiteAVSDK_Live:latest.release'
}
```

或



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

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



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

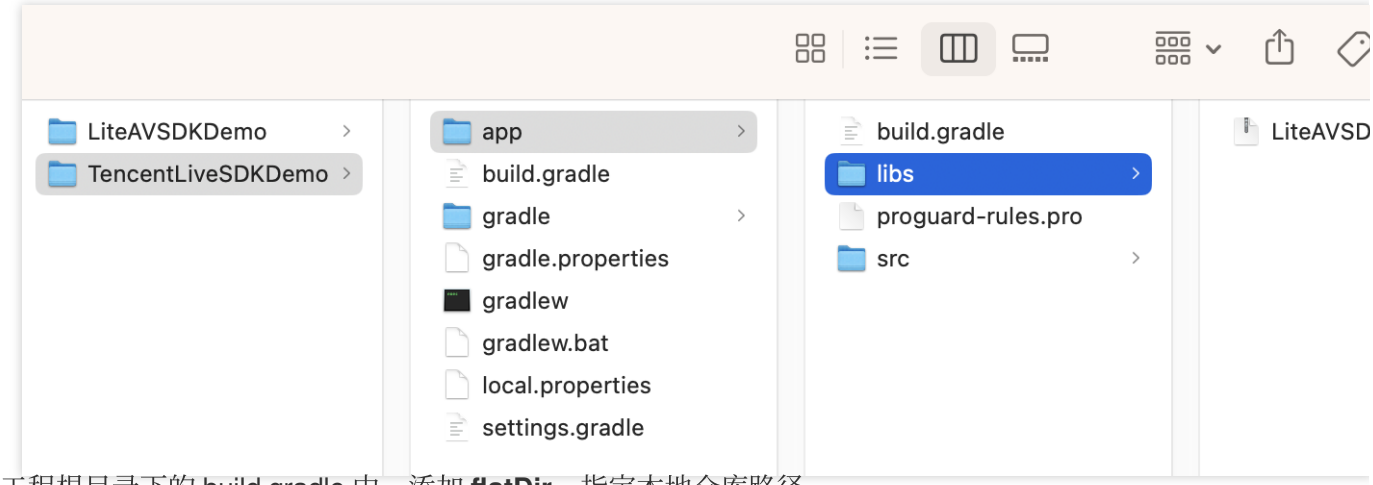
4. 单击

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。

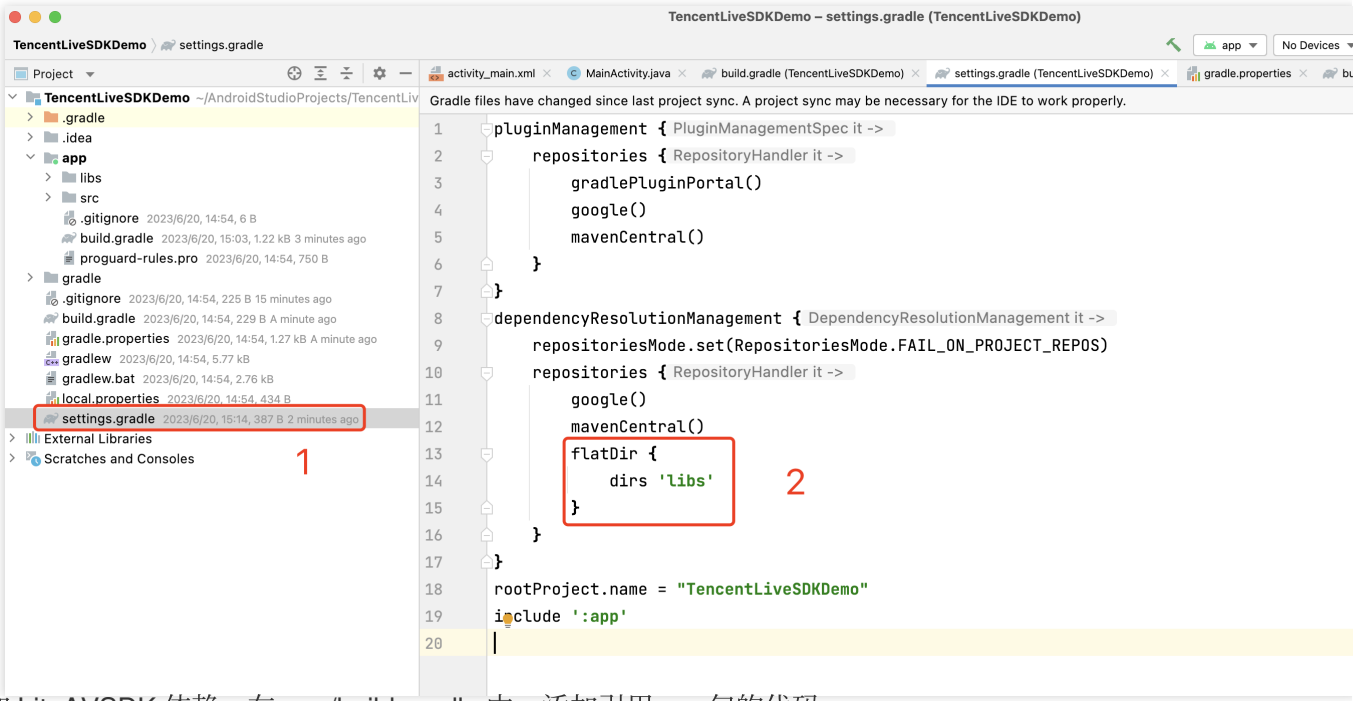
方法二：手动下载（aar）

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：

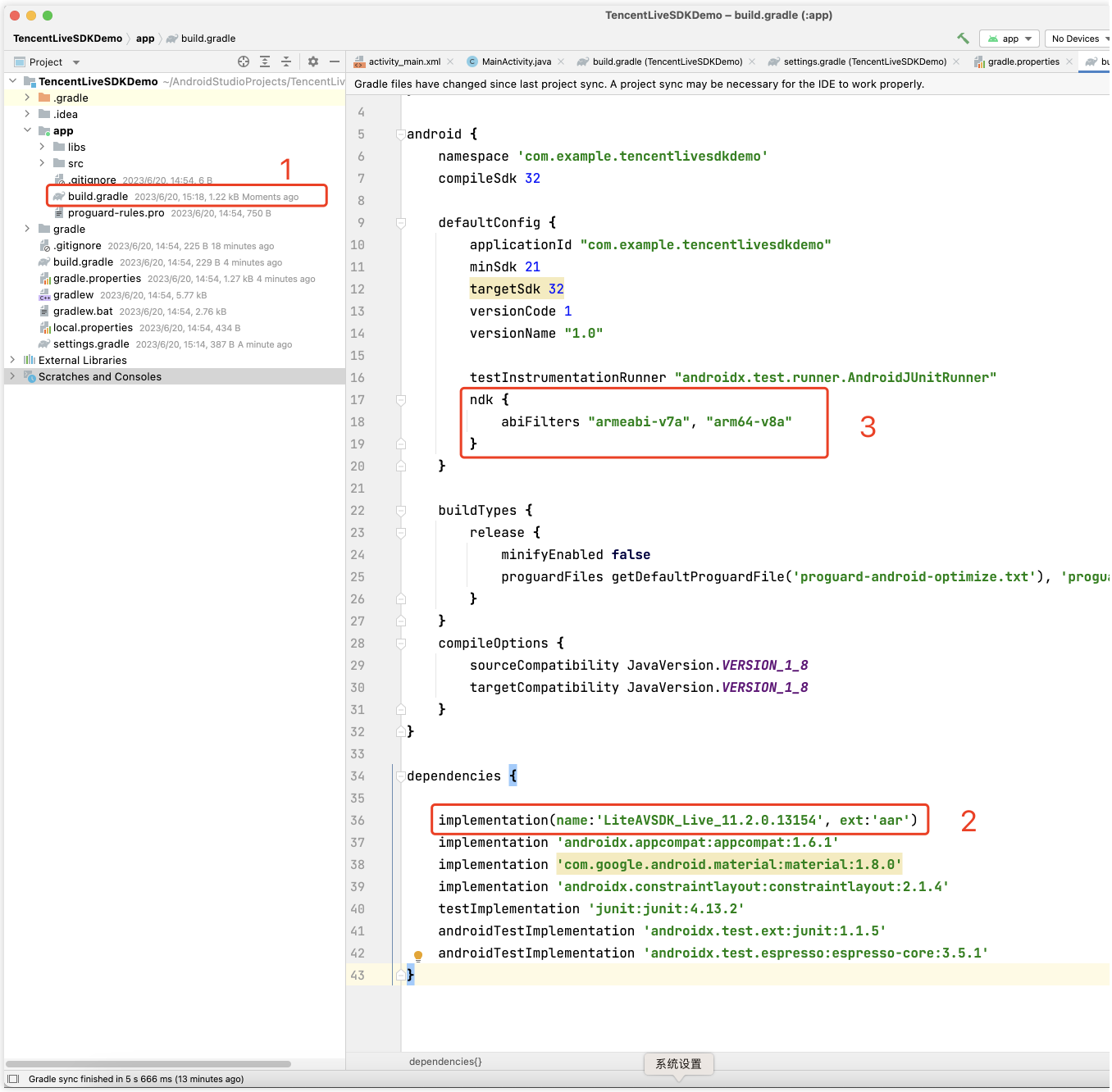
1. 下载 LiveAVSDK ， 下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：

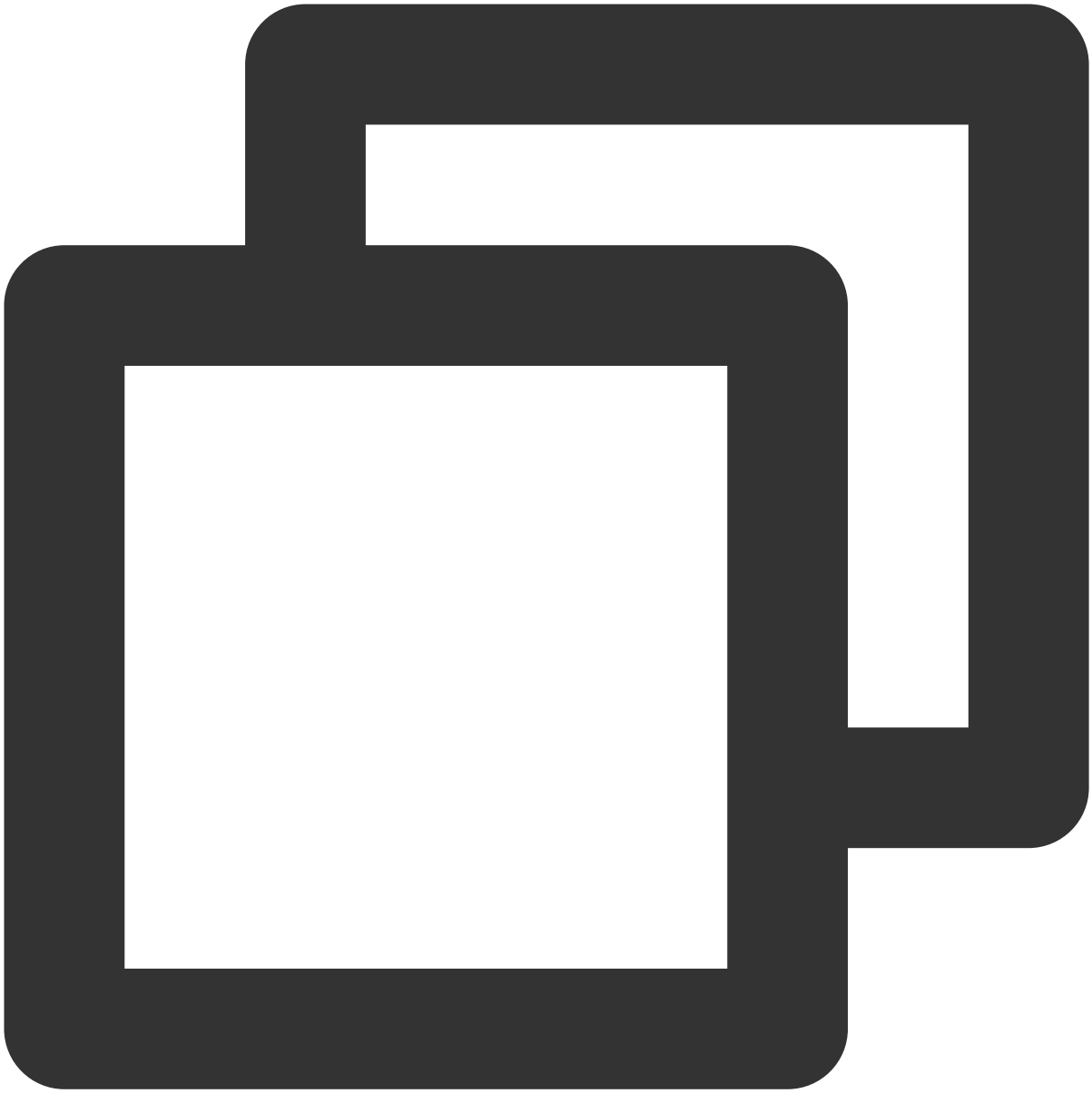


3. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



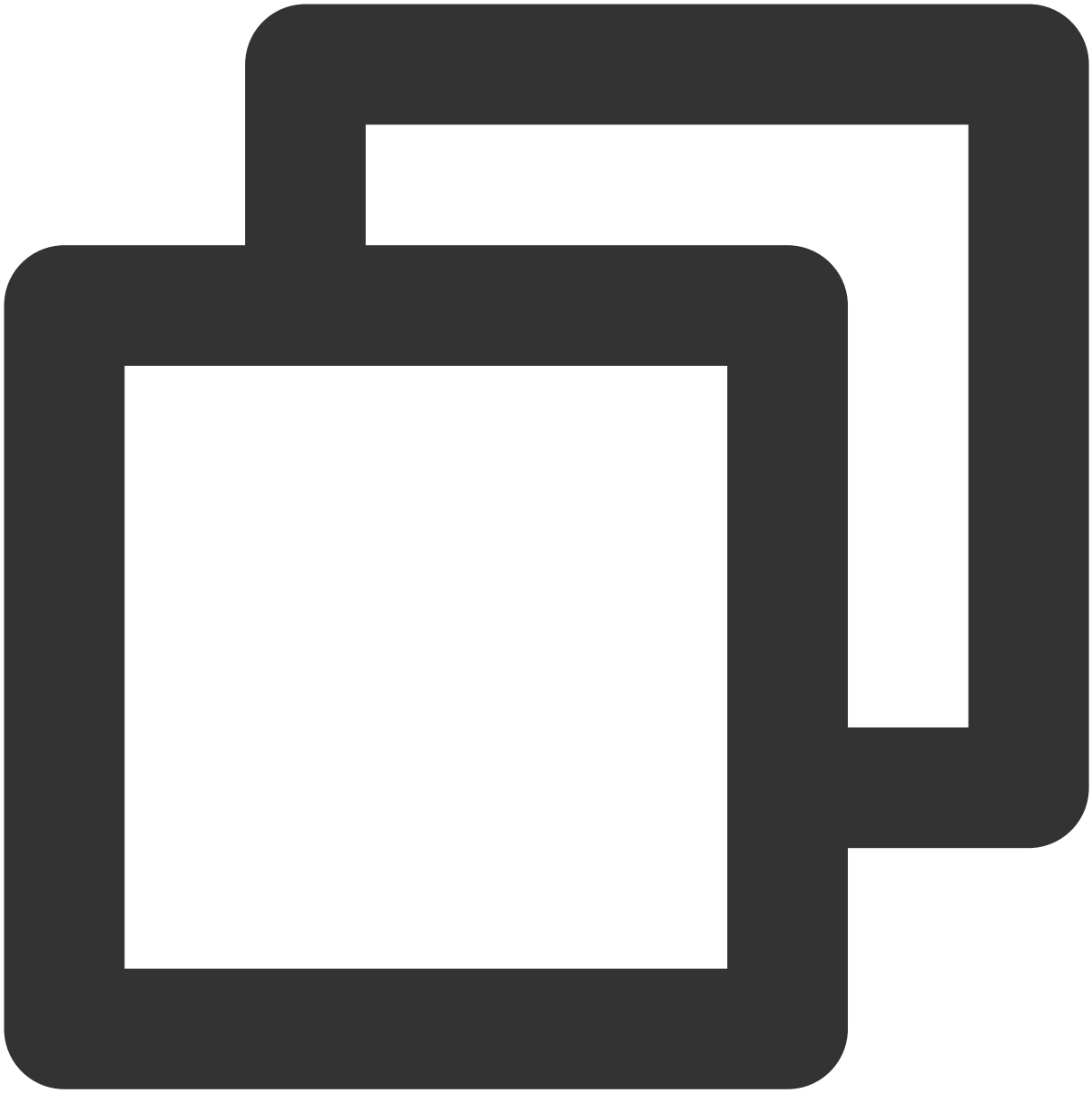
4. 添加 LiteAVSDK 依赖，在 app/build.gradle 中，添加引用 aar 包的代码。


TencentLiveSDKDemo – build.gradle (:app)
TencentLiveSDKDemo 〉 app 〉 build.gradle
Project
TencentLiveSDKDemo ~/AndroidStudioProjects/TencentLiv
.gradle
.idea
app
libs
src
.gitignore 2023/6/20, 14:54, 6 B
build.gradle 2023/6/20, 15:18, 1.22 kB Moments ago
proguard-rules.pro 2023/6/20, 14:54, 750 B
gradle
.gitignore 2023/6/20, 14:54, 225 B 18 minutes ago
build.gradle 2023/6/20, 14:54, 229 B 4 minutes ago
gradle.properties 2023/6/20, 14:54, 1.27 kB 4 minutes ago
gradlew 2023/6/20, 14:54, 5.77 kB
gradlew.bat 2023/6/20, 14:54, 2.76 kB
local.properties 2023/6/20, 14:54, 434 B
settings.gradle 2023/6/20, 15:14, 387 B A minute ago
External Libraries
Scratches and Consoles
1
activity_main.xml
MainActivity.java
build.gradle (TencentLiveSDKDemo)
settings.gradle (TencentLiveSDKDemo)
gradle.properties
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
android {
namespace 'com.example.tencentlivesdkdemo'
compileSdk 32
defaultConfig {
applicationId "com.example.tencentlivesdkdemo"
minSdk 21
targetSdk 32
versionCode 1
versionName "1.0"
testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
ndk {
abiFilters "armeabi-v7a", "arm64-v8a"
}
3
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'progu
}
}
compileOptions {
sourceCompatibility JavaVersion.VERSION_1_8
targetCompatibility JavaVersion.VERSION_1_8
}
}
dependencies {
implementation(name:'LiteAVSDK_Live_11.2.0.13154', ext:'aar')
2
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'com.google.android.material:material:1.8.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.5'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
dependencies{}
系统设置
Gradle sync finished in 5 s 666 ms (13 minutes ago)



```
implementation(name:'LiteAVSDK_Live_11.2.0.13154', 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-v7a 和 arm64-v8a）。



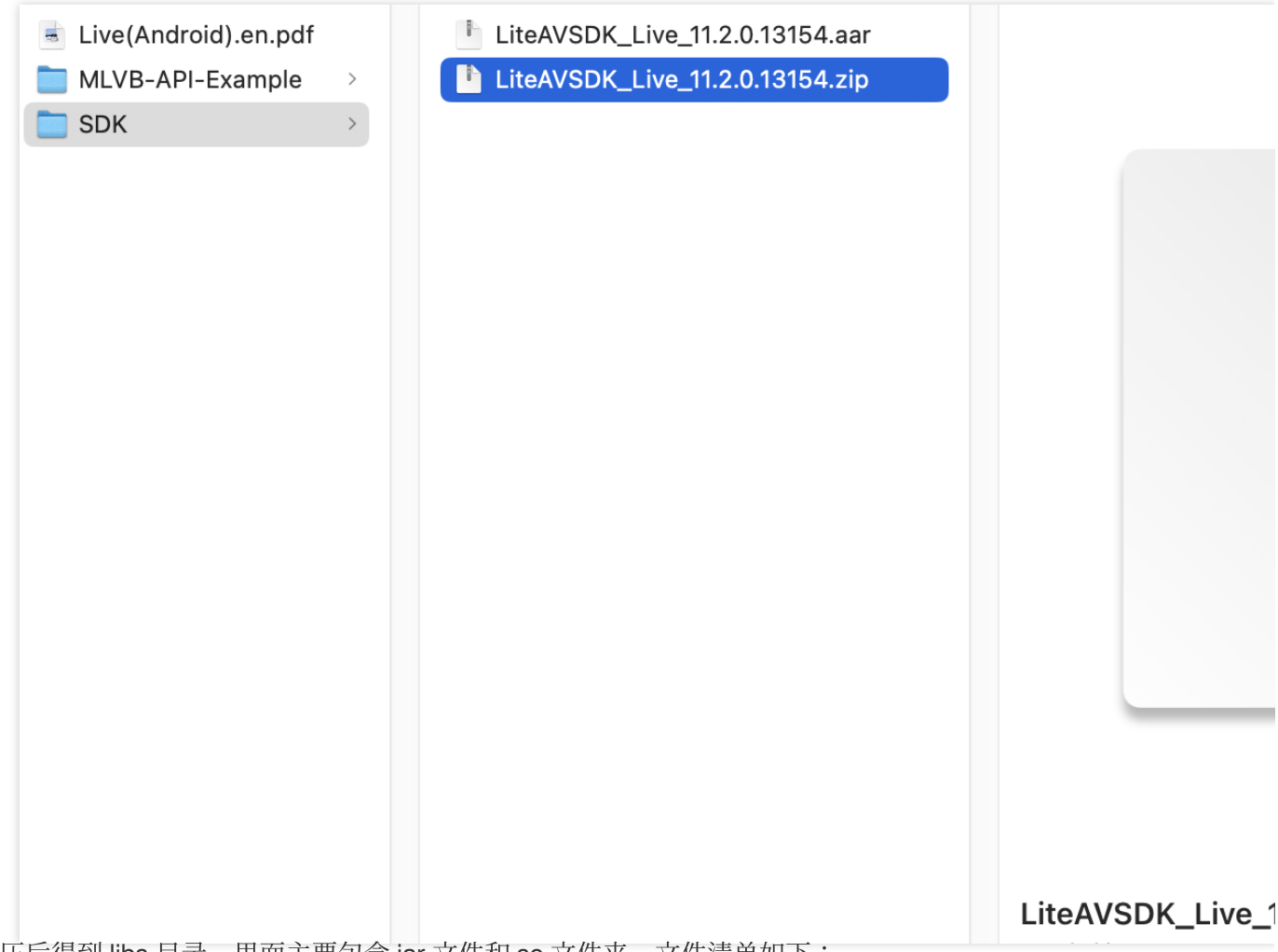
```
defaultConfig {
      ndk {
          abiFilters "armeabi-v7a", "arm64-v8a"
      }
}
```

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

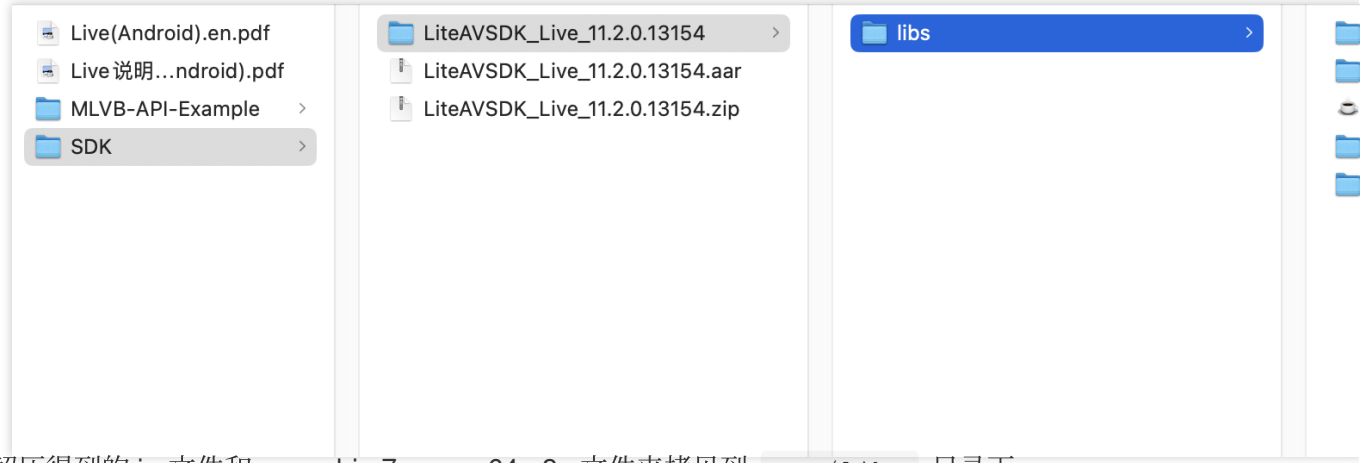
集成 SDK（jar）

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：

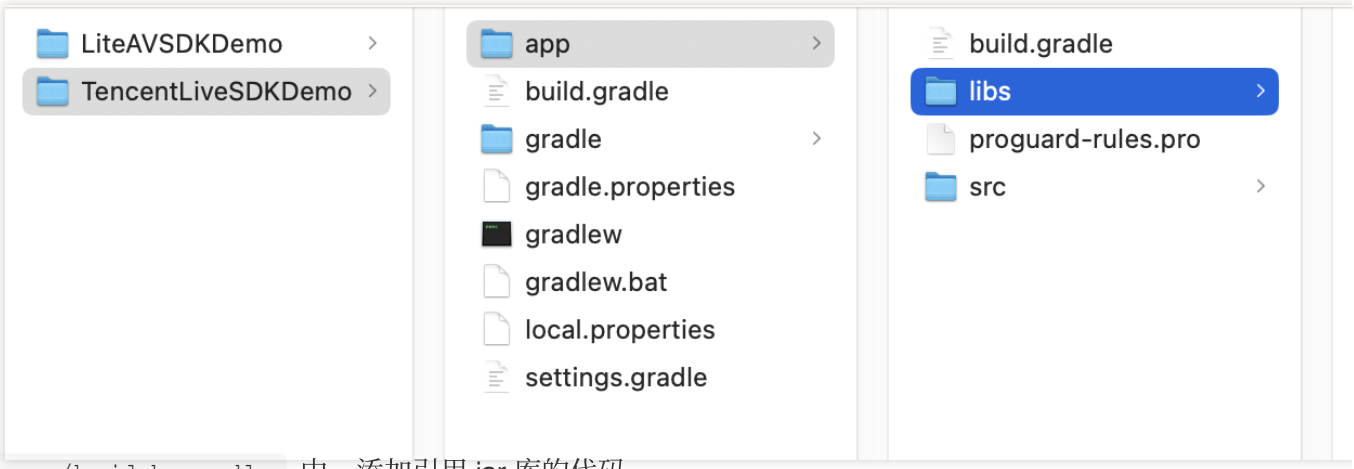
1. 下载 LiveAVSDK，下载完成后进行解压。在 SDK 目录下找到 `LiteAVSDK_Live_xxx.zip`（其中 `xxx` 为 LiteAVSDK 的版本号）：



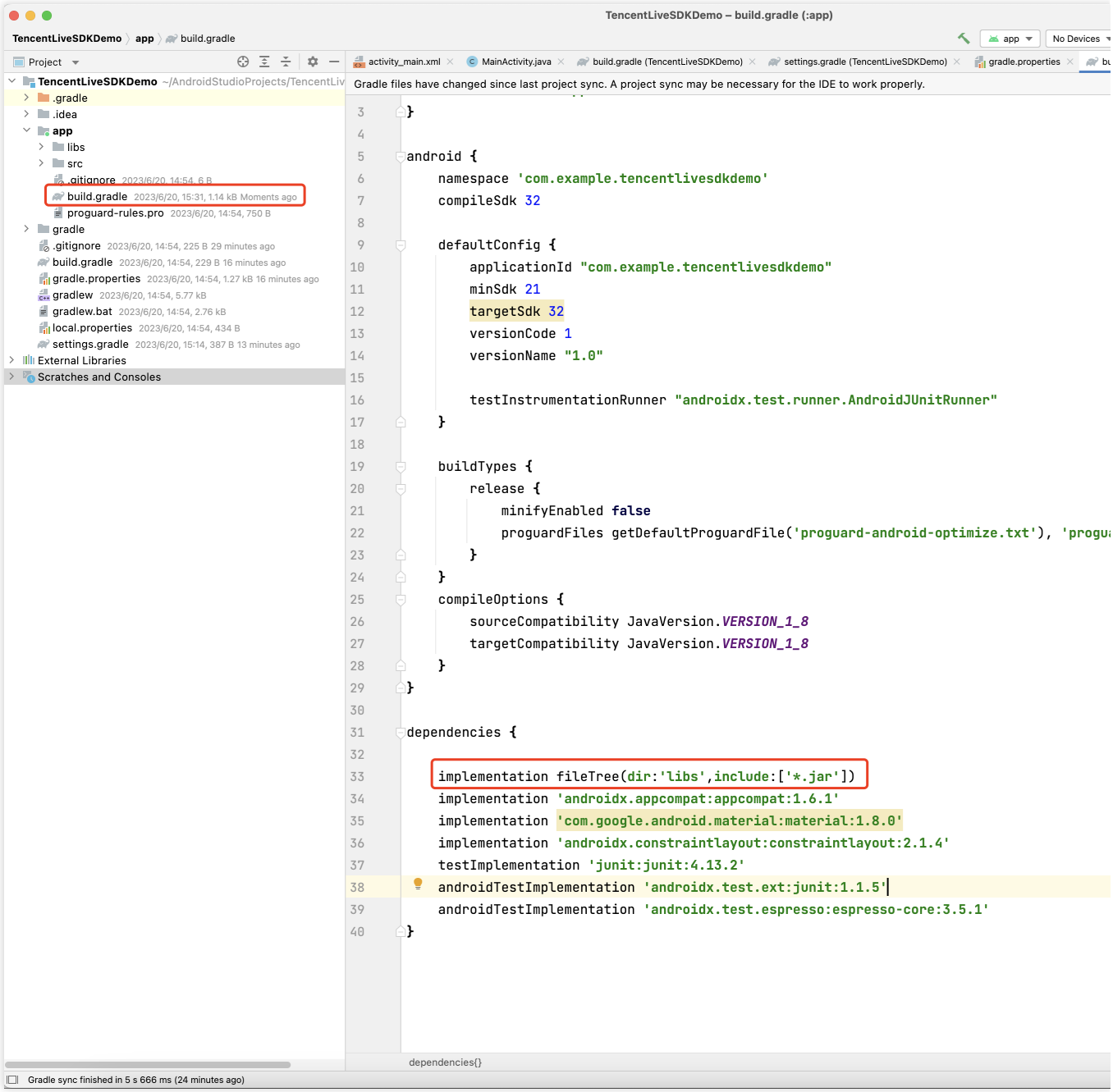
2. 解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：

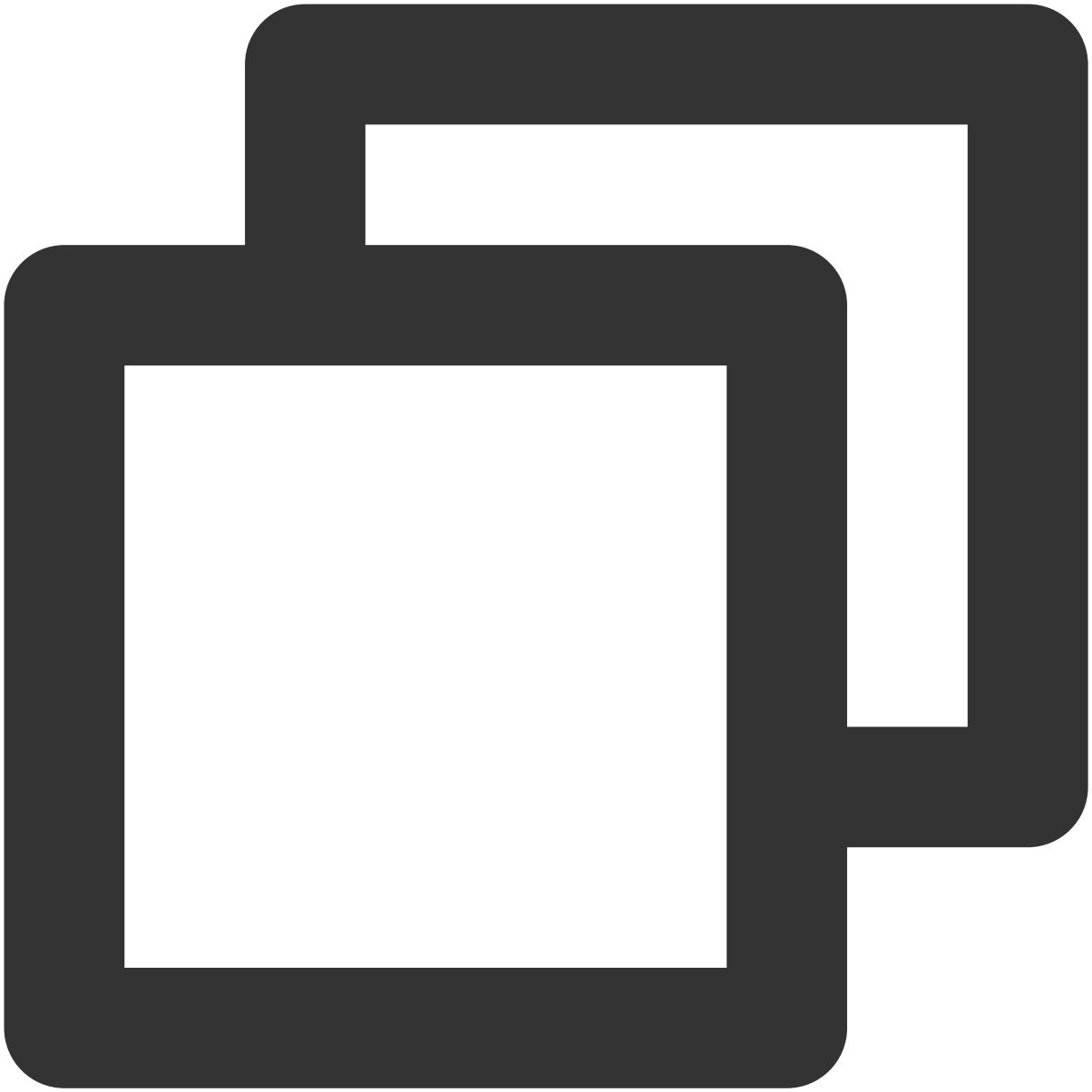


3. 将解压得到的 jar文件和 armeabi-v7a、arm64-v8a 文件夹拷贝到 `app/libs` 目录下。



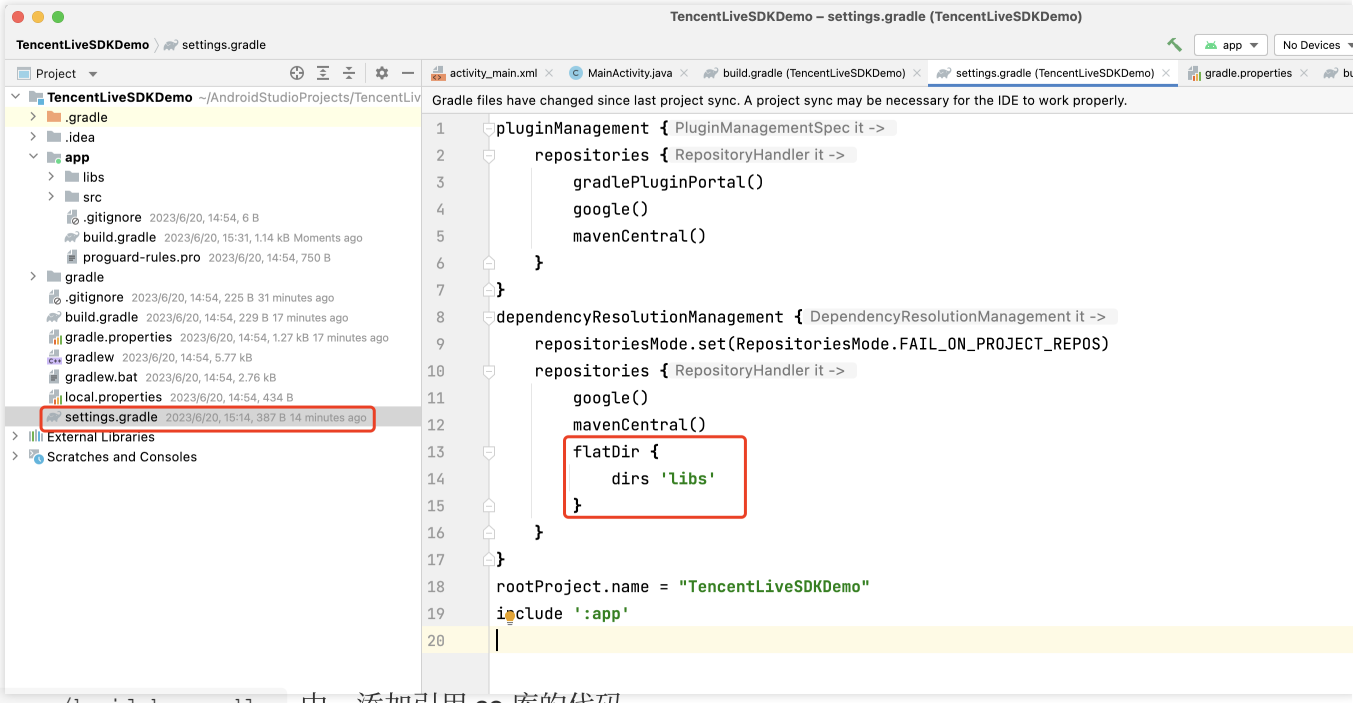
4. 在 `app/build.gradle` 中，添加引用 jar 库的代码。


TencentLiveSDKDemo – build.gradle (:app)
TencentLiveSDKDemo 〉 app 〉 build.gradle
app
No Devices
Project
activity_main.xml
MainActivity.java
build.gradle (TencentLiveSDKDemo)
settings.gradle (TencentLiveSDKDemo)
gradle.properties
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
TencentLiveSDKDemo ~/AndroidStudioProjects/TencentLiv
.gradle
.idea
app
libs
src
.gitignore 2023/6/20, 14:54, 6 B
build.gradle 2023/6/20, 15:31, 1.14 kB Moments ago
proguard-rules.pro 2023/6/20, 14:54, 750 B
gradle
.gitignore 2023/6/20, 14:54, 225 B 29 minutes ago
build.gradle 2023/6/20, 14:54, 229 B 16 minutes ago
gradle.properties 2023/6/20, 14:54, 1.27 kB 16 minutes ago
gradlew 2023/6/20, 14:54, 5.77 kB
gradlew.bat 2023/6/20, 14:54, 2.76 kB
local.properties 2023/6/20, 14:54, 434 B
settings.gradle 2023/6/20, 15:14, 387 B 13 minutes ago
External Libraries
Scratches and Consoles
}
android {
namespace 'com.example.tencentlivesdkdemo'
compileSdk 32
defaultConfig {
applicationId "com.example.tencentlivesdkdemo"
minSdk 21
targetSdk 32
versionCode 1
versionName "1.0"
testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'progu
}
}
compileOptions {
sourceCompatibility JavaVersion.VERSION_1_8
targetCompatibility JavaVersion.VERSION_1_8
}
}
dependencies {
implementation fileTree(dir:'libs',include:['*.jar'])
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'com.google.android.material:material:1.8.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.5'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
dependencies{}
Gradle sync finished in 5 s 666 ms (24 minutes ago)

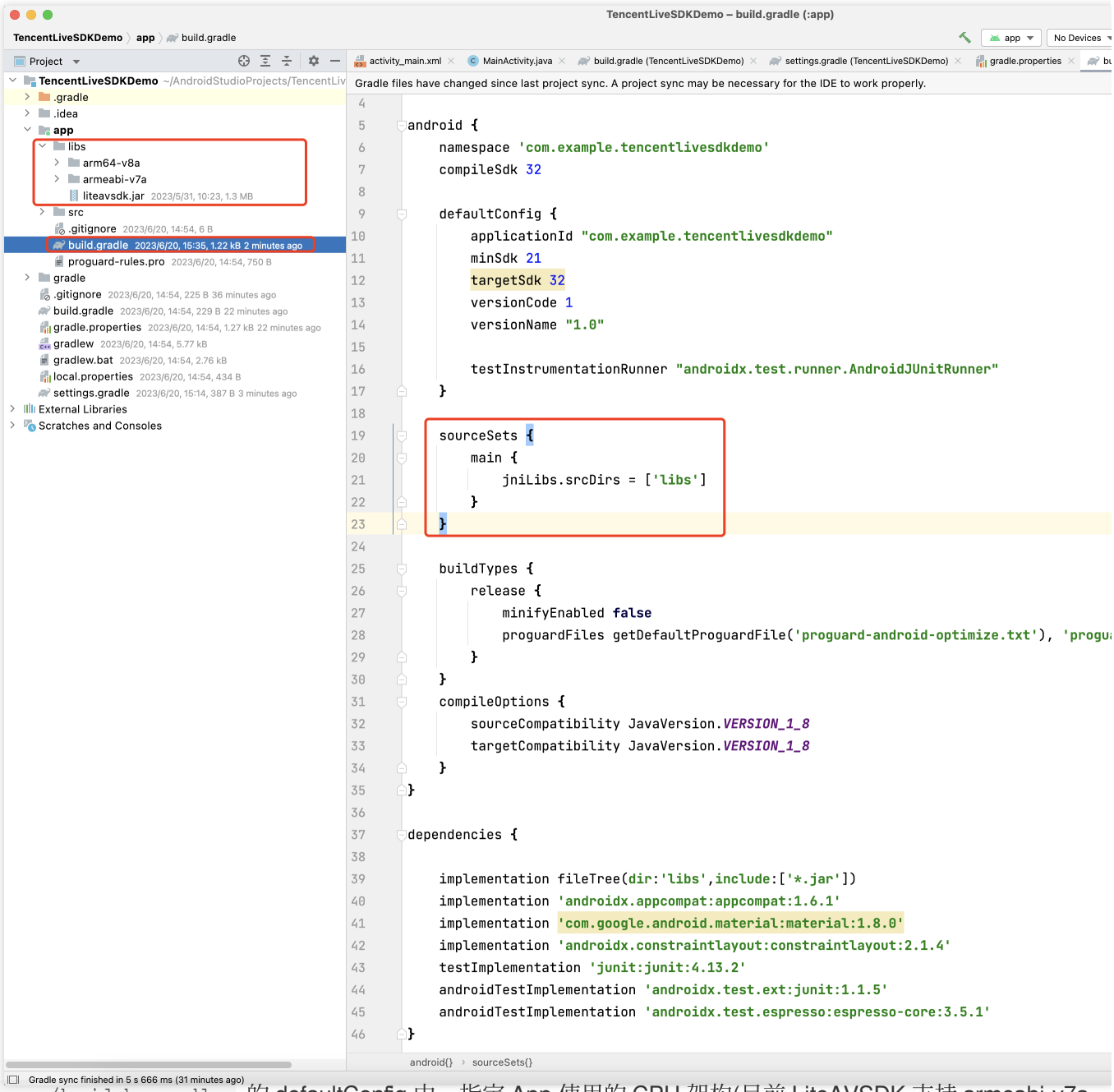


```
dependencies{
    implementation fileTree(dir:'libs',include:['*.jar'])
}
```

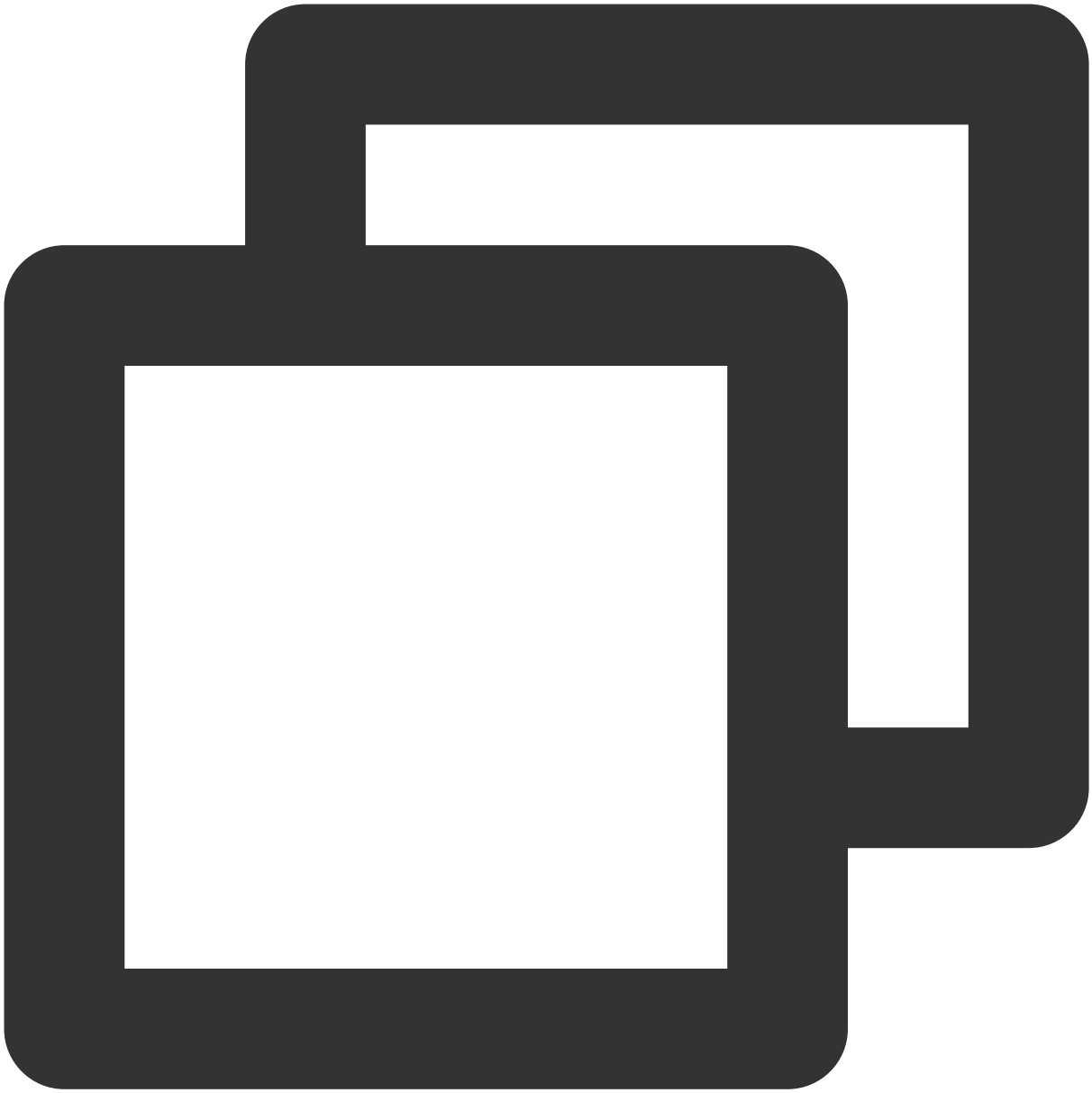
5. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



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



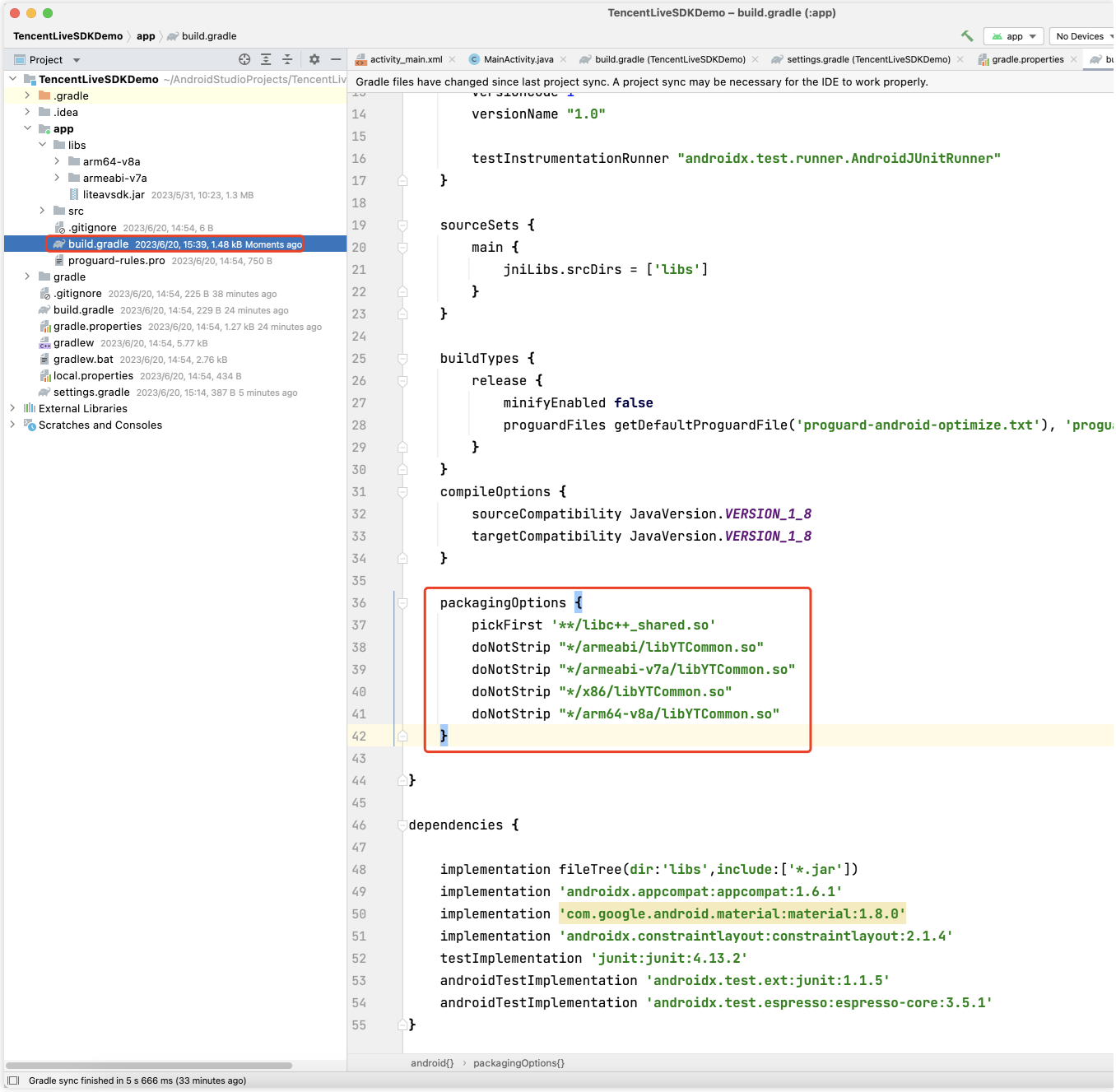
7. 在 `app/build.gradle` 的 defaultConfig 中，指定 App 使用的 CPU 架构(目前 LiteAVSDK 支持 armeabi-v7a 和 arm64-v8a) 。

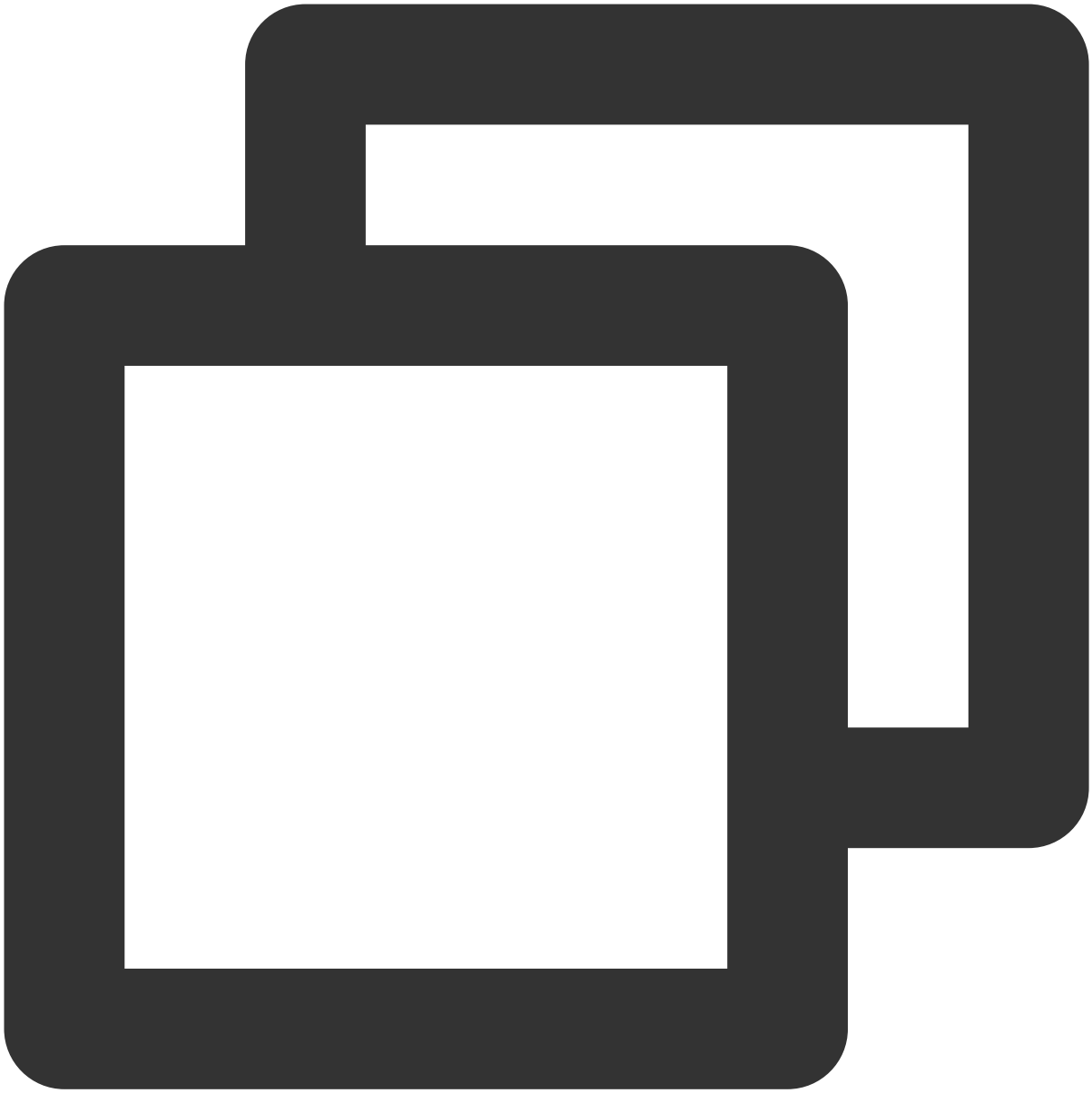


```
defaultConfig {
    ndk {
        abiFilters "armeabi-v7a", "arm64-v8a"
    }
}
```

8. 单击 Sync Now 同步 SDK，完成 LiteAVSDK 的集成工作。

配置 App 打包参数

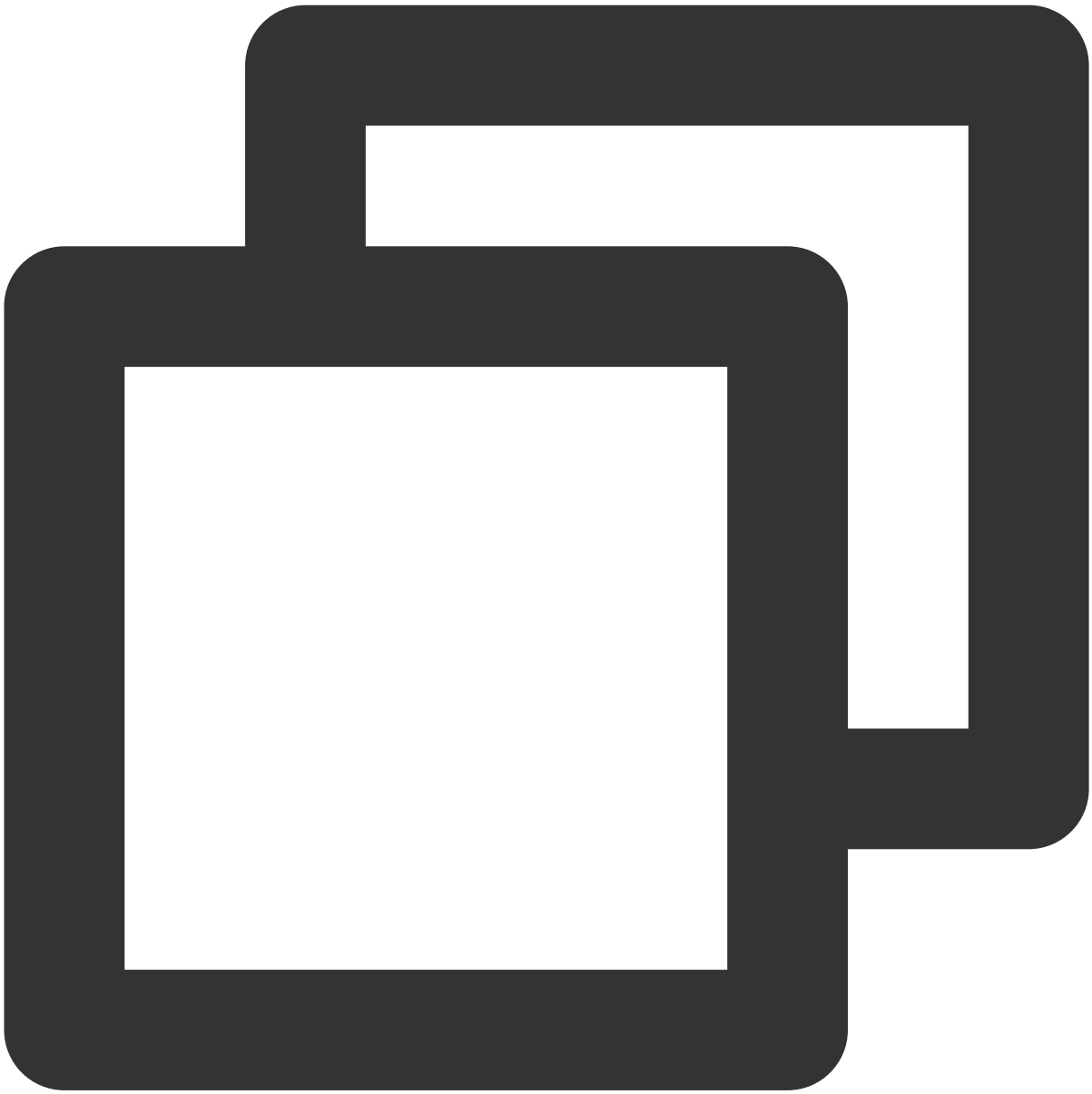




```
packagingOptions {
    pickFirst '**/libc++_shared.so'
    doNotStrip "*/armeabi/libYTCommon.so"
    doNotStrip "*/armeabi-v7a/libYTCommon.so"
    doNotStrip "*/x86/libYTCommon.so"
    doNotStrip "*/arm64-v8a/libYTCommon.so"
}
```

配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，LiteAVSDK 需要以下权限：

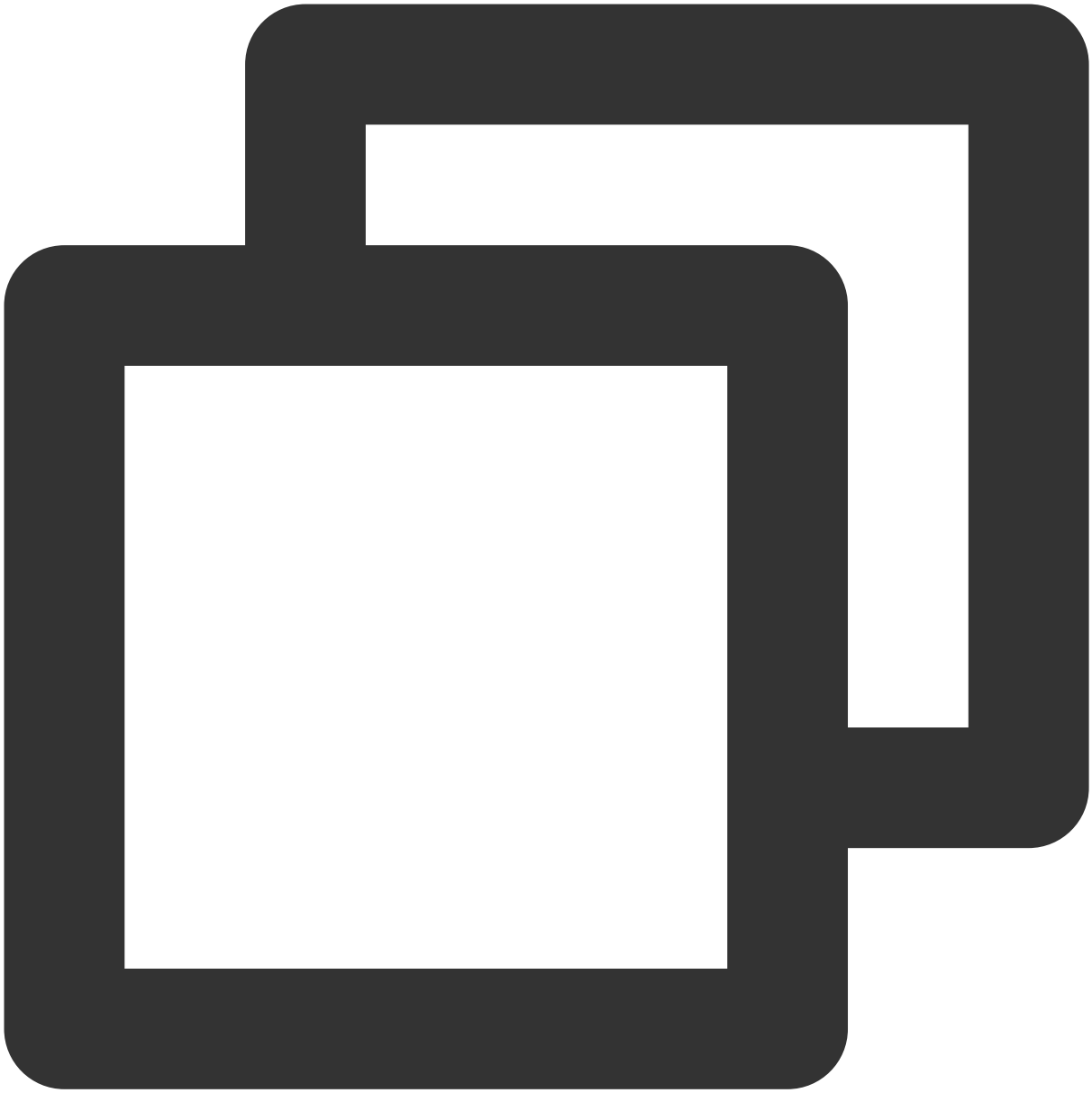


```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

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

配置 License 授权

单击 License 申请 获取 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：

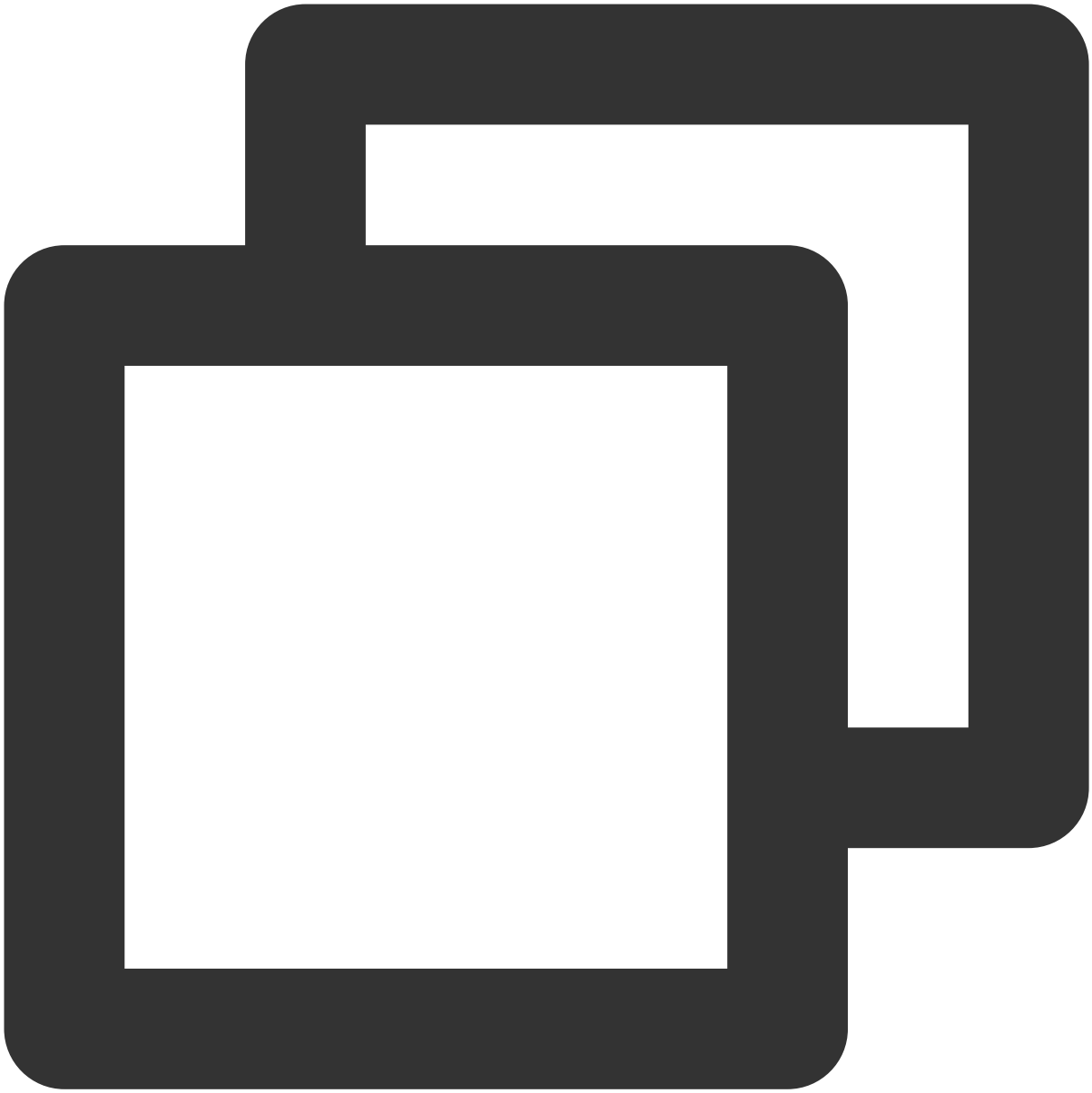


```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // 获取到的 licence url
        String licenceKey = ""; // 获取到的 licence key
        V2TXLivePremier.setEnvironment("GDPR"); // 设置环境
        V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
        V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
```

```
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reas
            }
        });
    }
}
```

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。

Flutter

最近更新时间：2024-01-13 15:53:49

本文主要介绍如何快速地将 腾讯云视立方·移动直播 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
   - Android
     i. 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>
```

ii. 开通摄像头和麦克风的权限，即可开启语音通话功能。

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.tencentcloud.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。

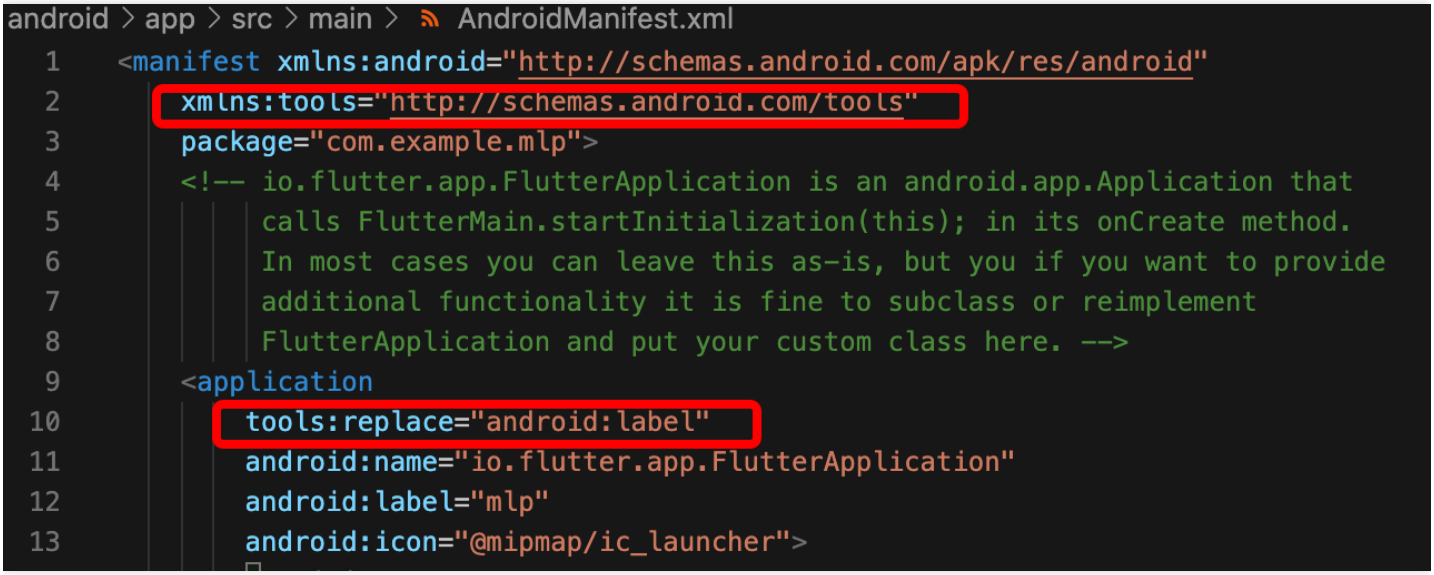
iOS 无法显示视频（Android 正常）？

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。

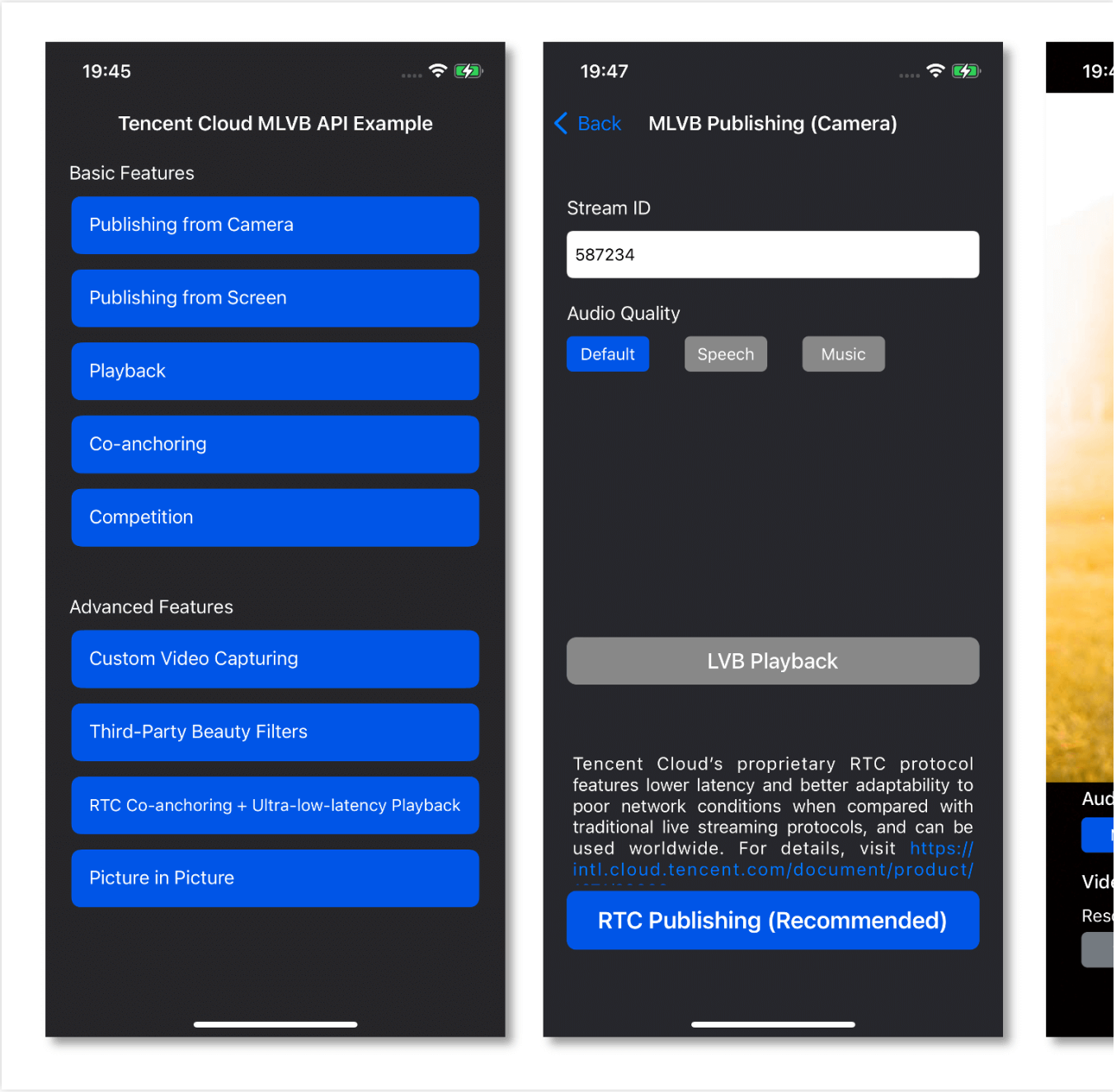


直播推流
iOS
摄像头推流

最近更新时间：2024-01-13 15:53:49

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

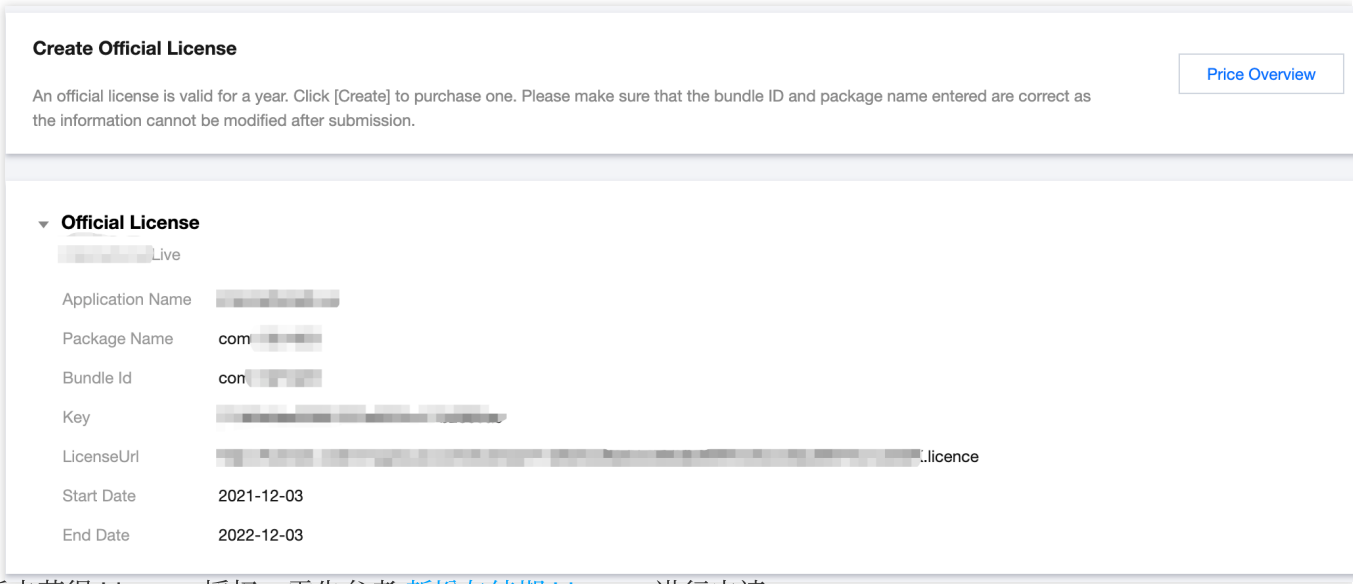
功能对接

1. 下载 SDK 开发包

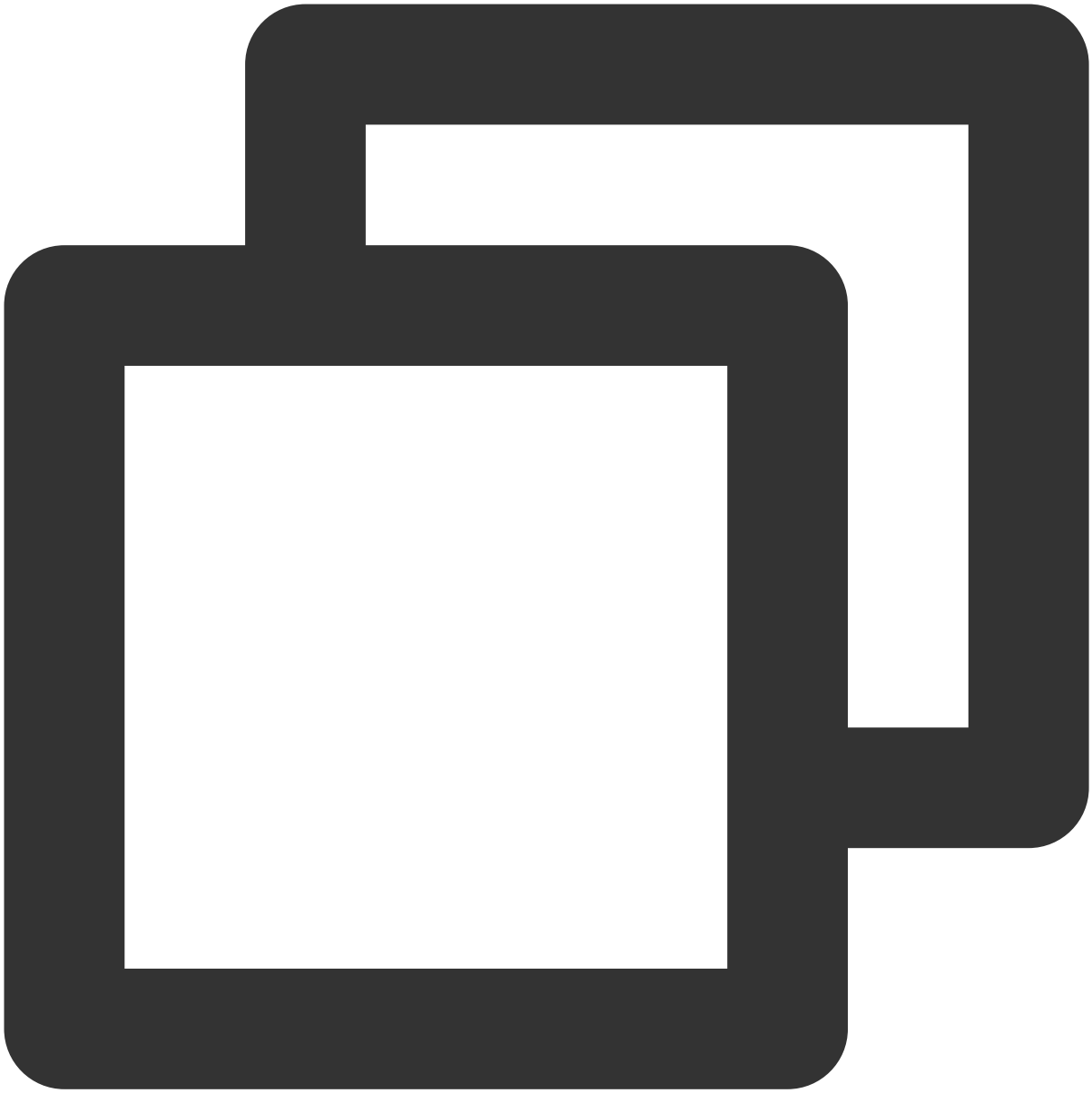
下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

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:(NSD
    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 setEnvironment:@"GDPR"]; // 设置环境
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];
    [V2TXLivePremier setObserver:self];
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
    return YES;
}
```

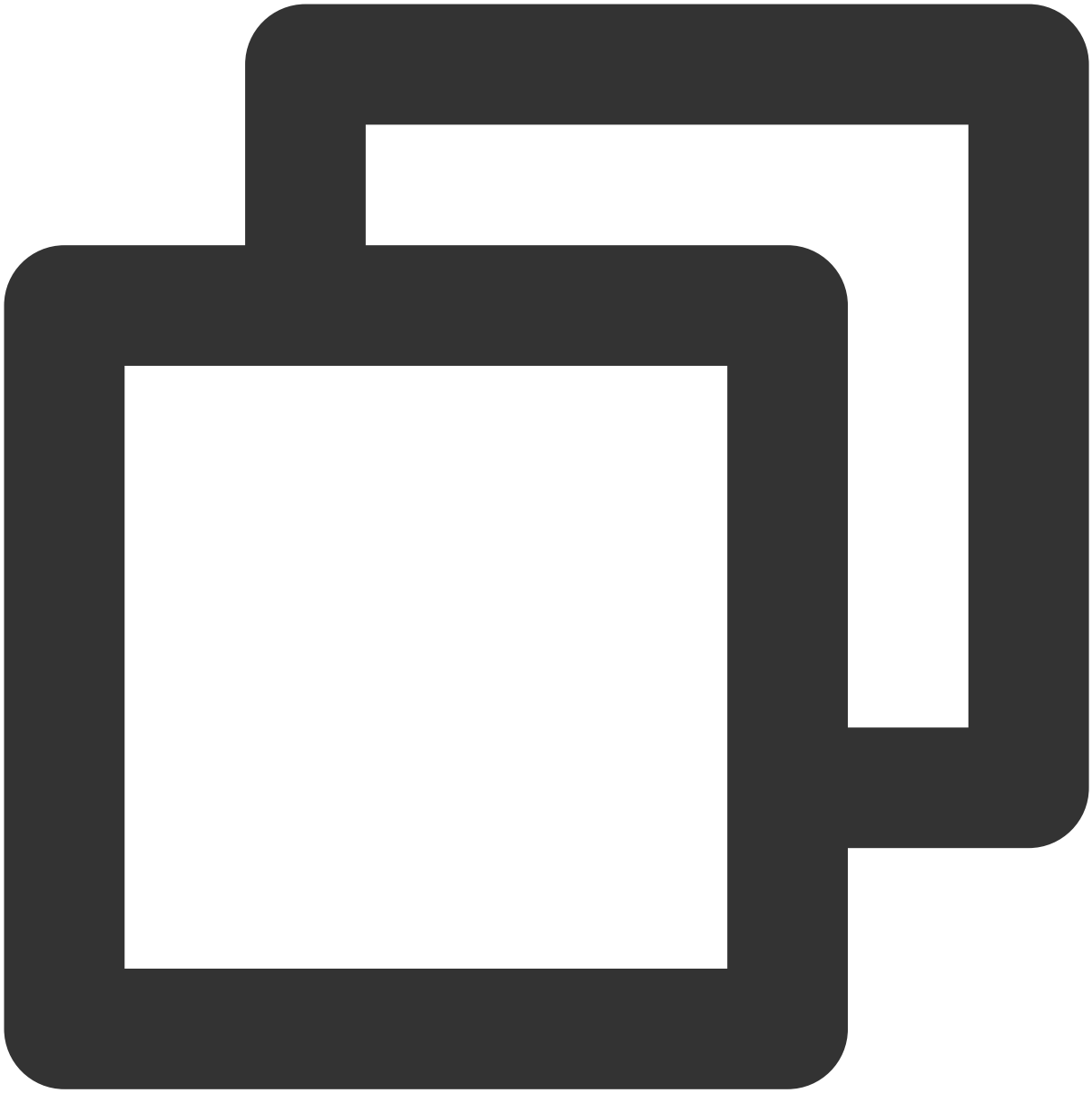
```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

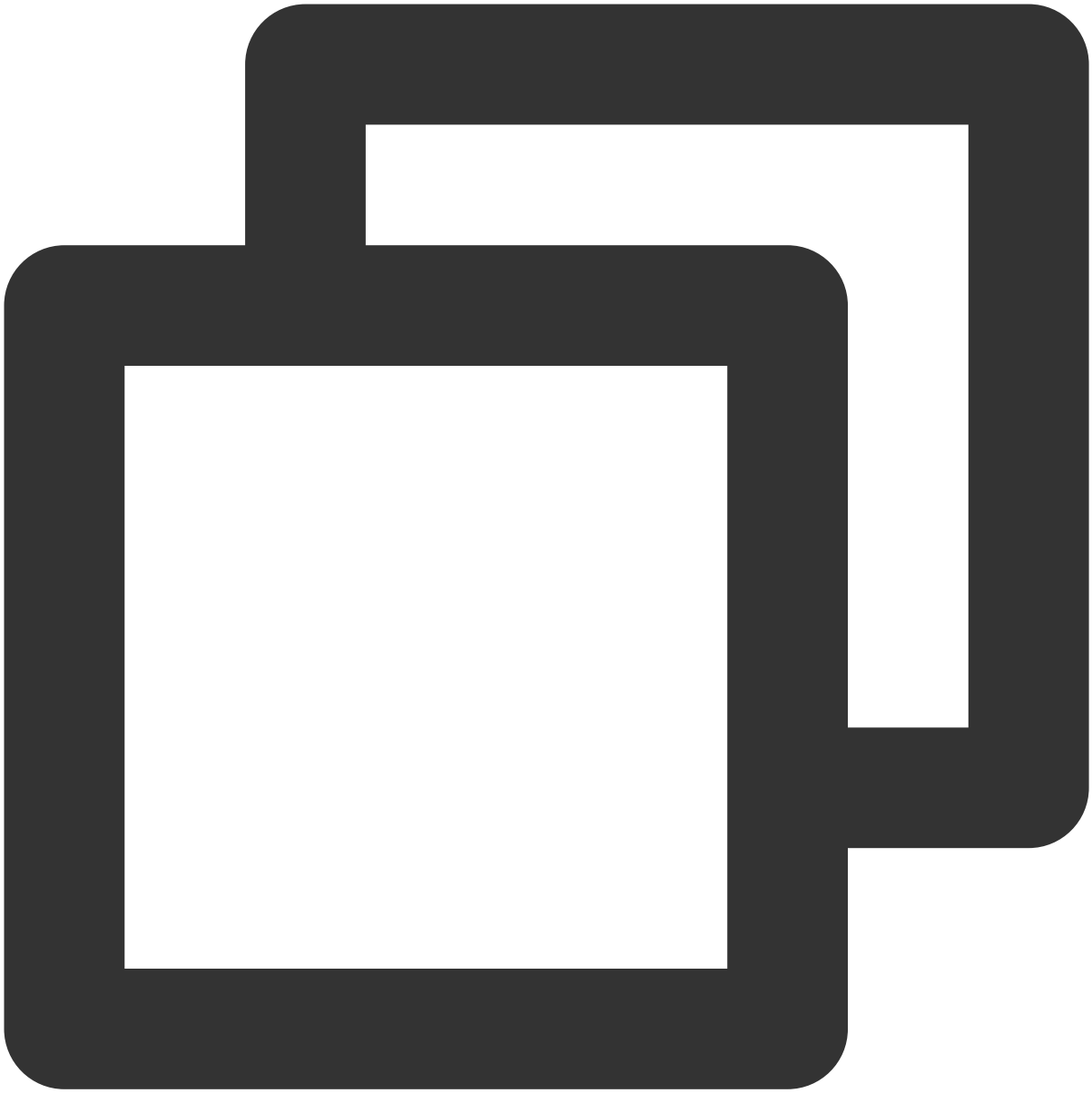
创建一个 `V2TXLivePusher` 对象，需要指定对应的 `V2TXLiveMode` 。



```
// 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
```

4. 开启摄像头预览

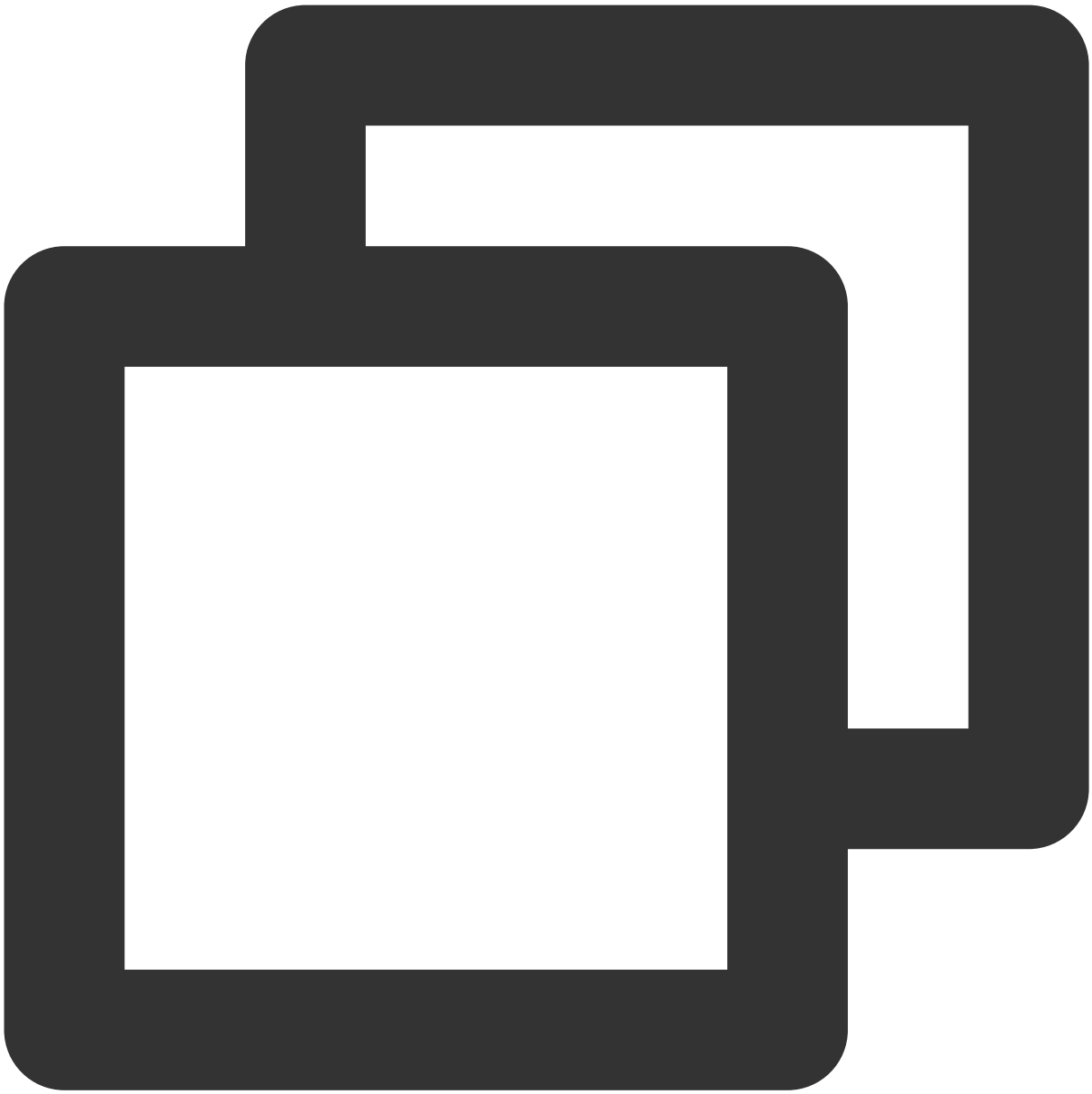
想要开启摄像头预览，首先需要调用 V2TXLivePusher 中的 `setRenderView` 接口，改接口需要设置一个用于显示视频画面的 view 对象，然后调用 `startCamera` 接口开启当前手机摄像头的预览。



```
//创建一个 view 对象，并将其嵌入到当前界面中
UIView *_localView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view insertSubview:_localView atIndex:0];
_localView.center = self.view.center;

//启动本地摄像头预览
[_pusher setRenderView:_localView];
[_pusher startCamera:YES];
[_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。



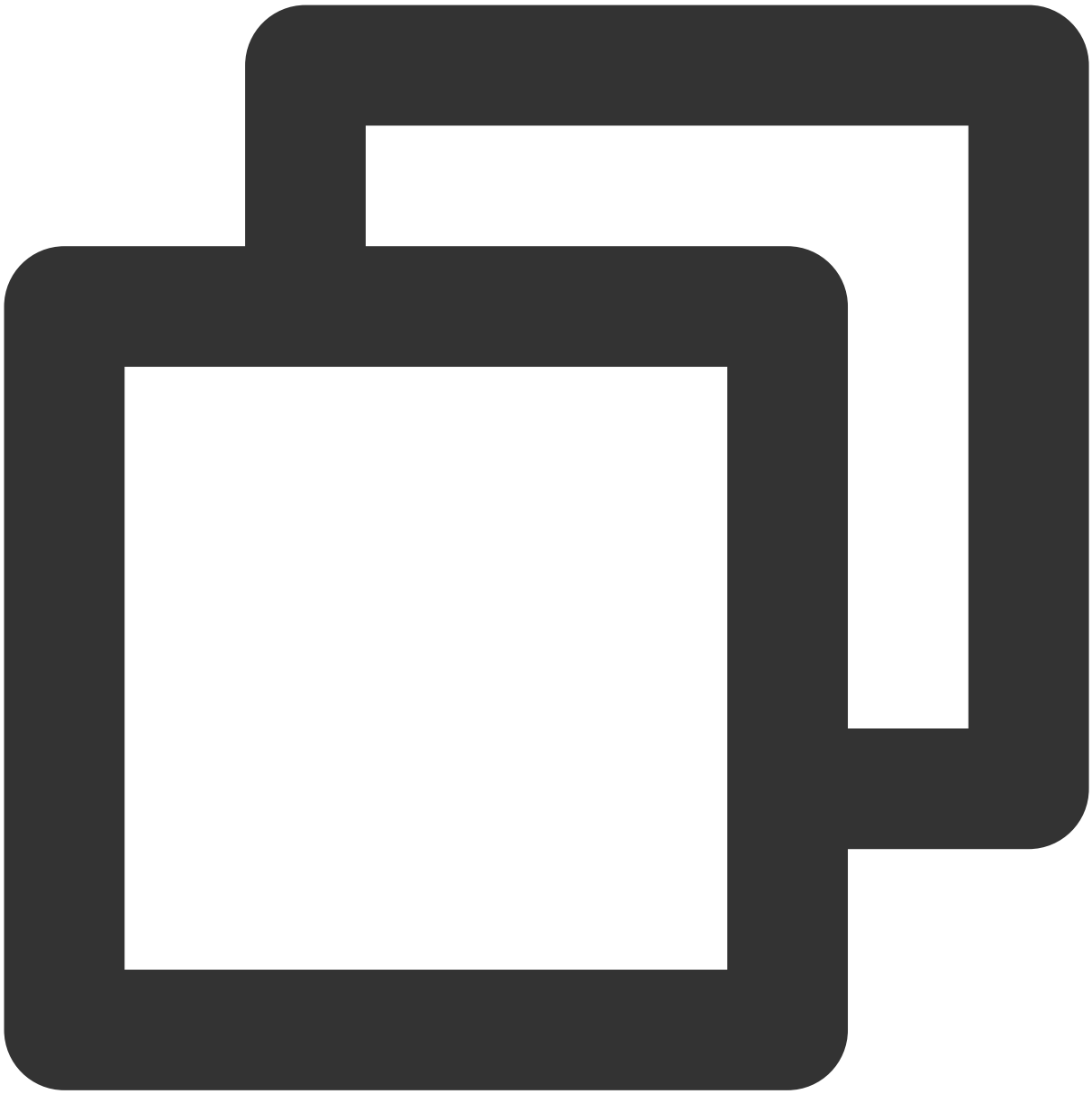
```
[UIView animateWithDuration:0.5 animations:^{
      _localView.transform = CGAffineTransformMakeScale(0.3, 0.3); //缩小1/3
}];
```

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。

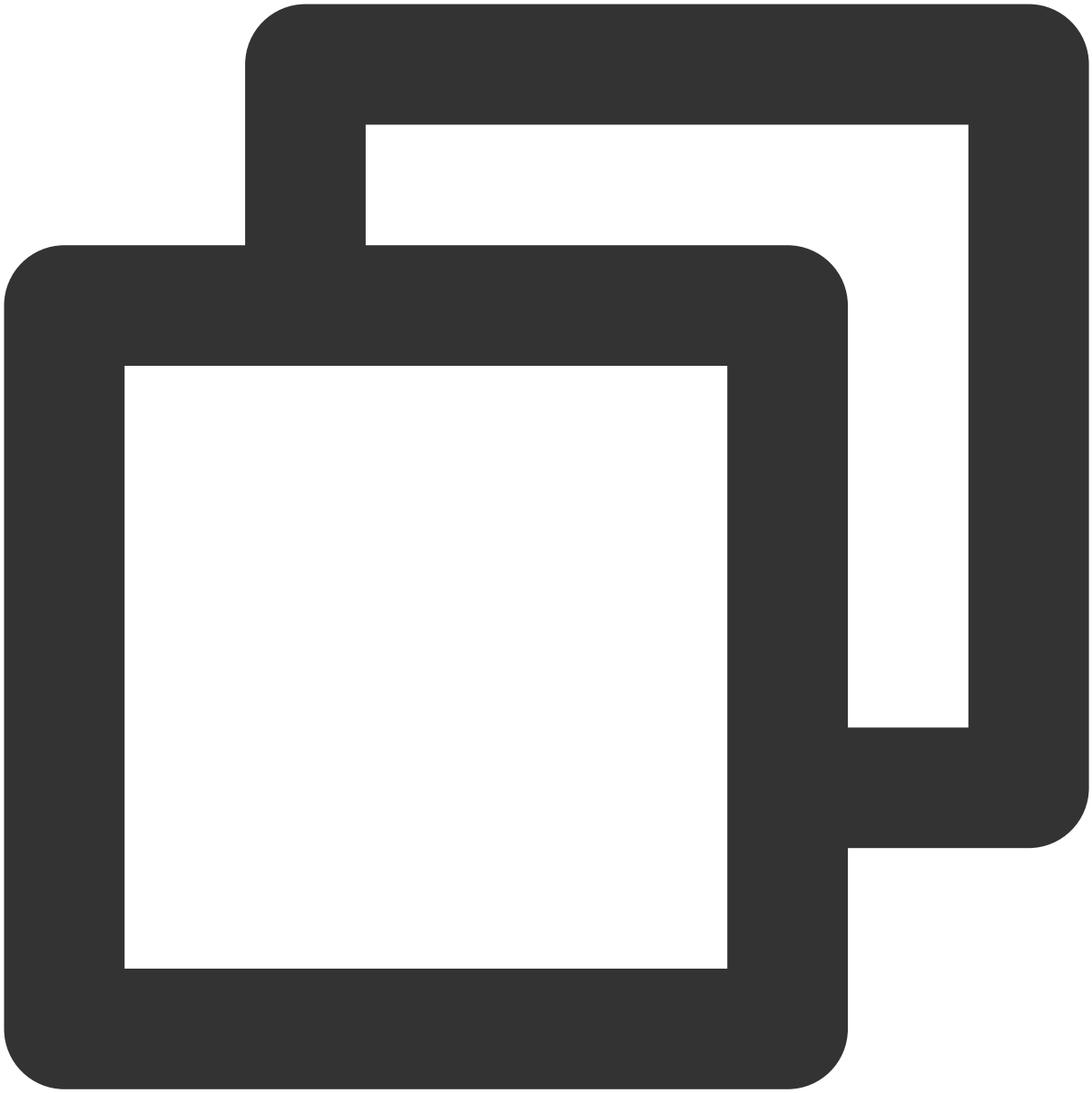
说明：

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。



```
// 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";
[_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];
```

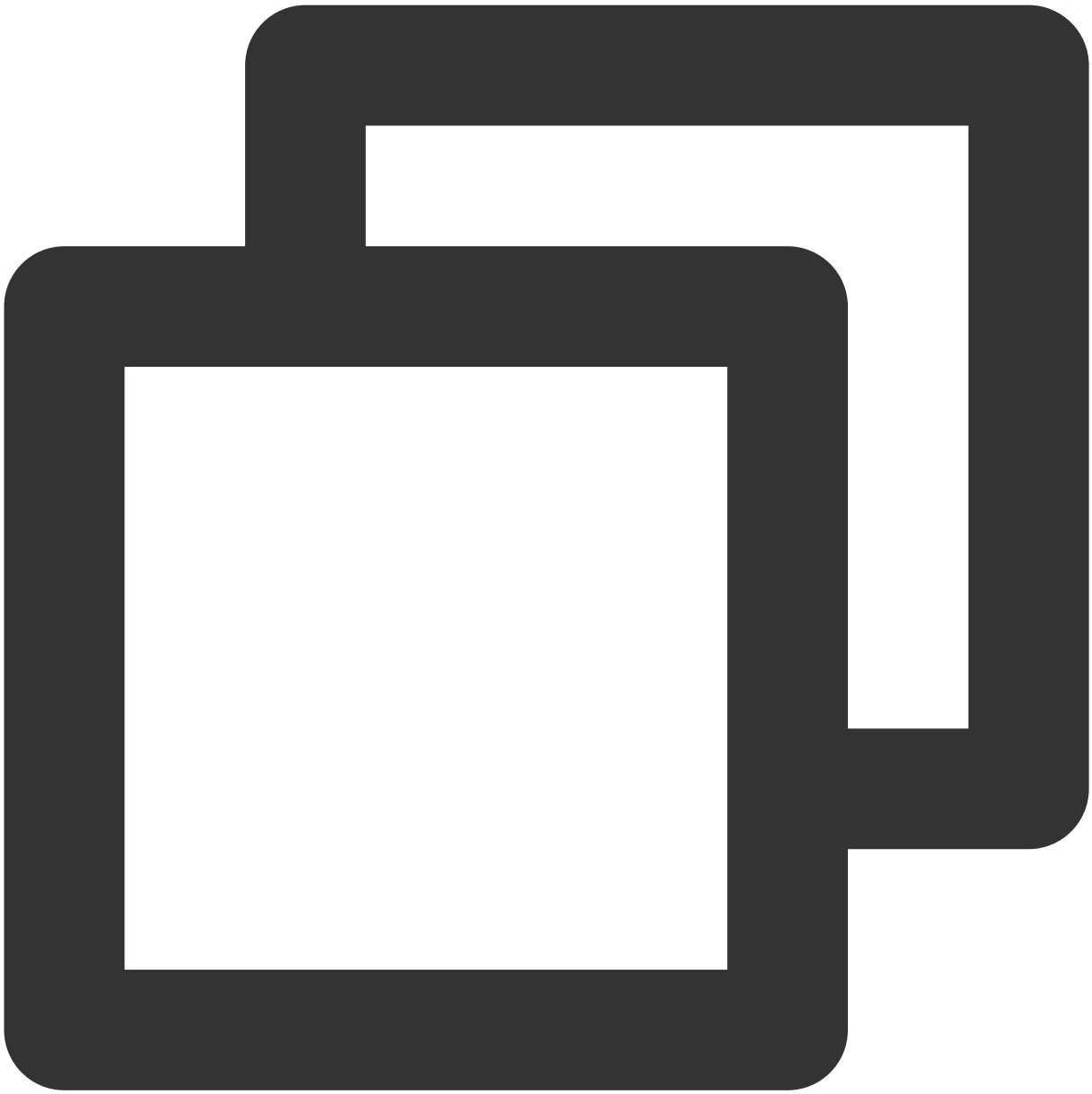
注意：

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";
[_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：

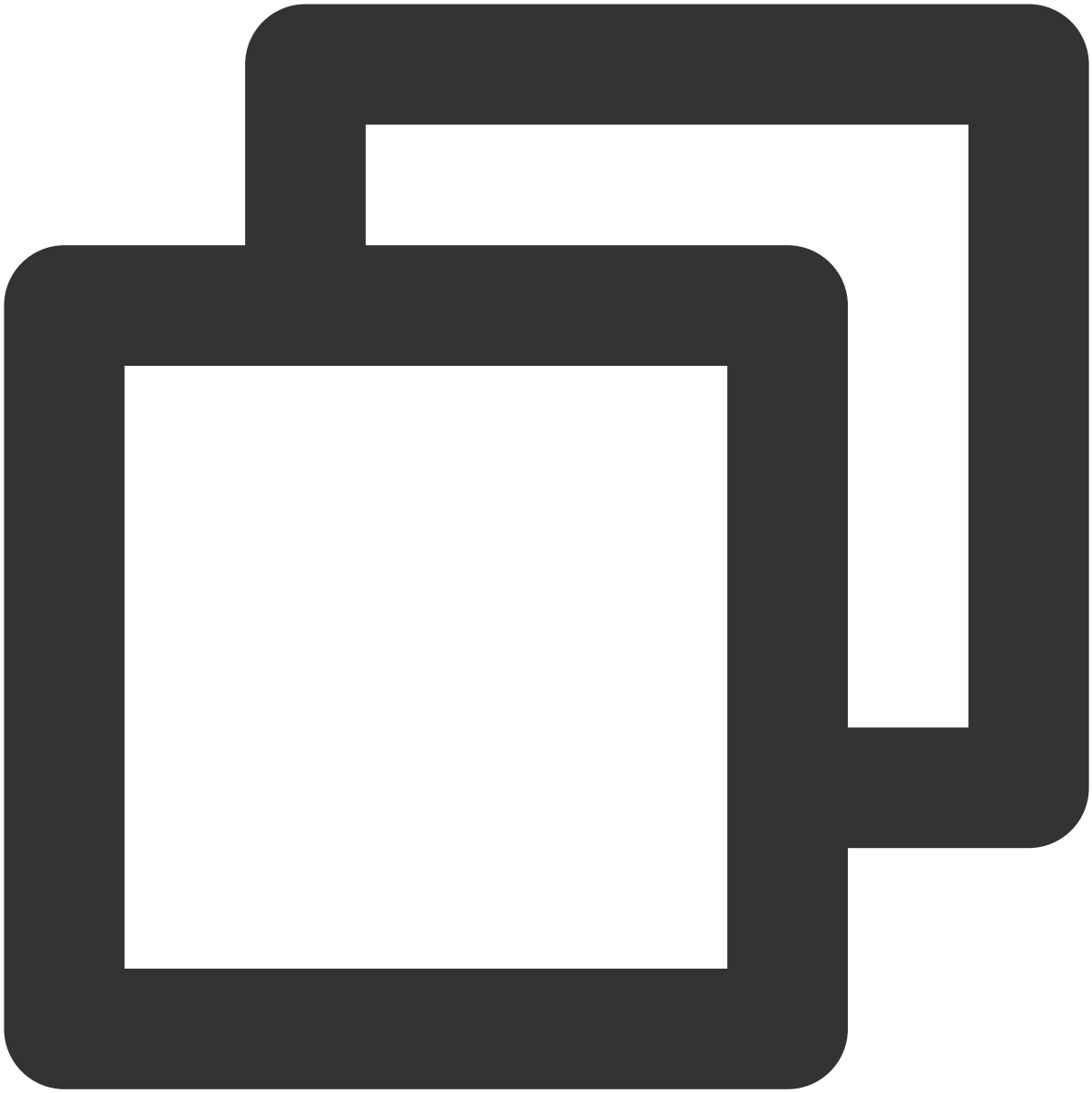
美颜风格	效果说明
TXBeautyStyleSmooth	光滑，适用于美女秀场，效果比较明显
TXBeautyStyleNature	自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然
TXBeautyStylePitu	由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高

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。



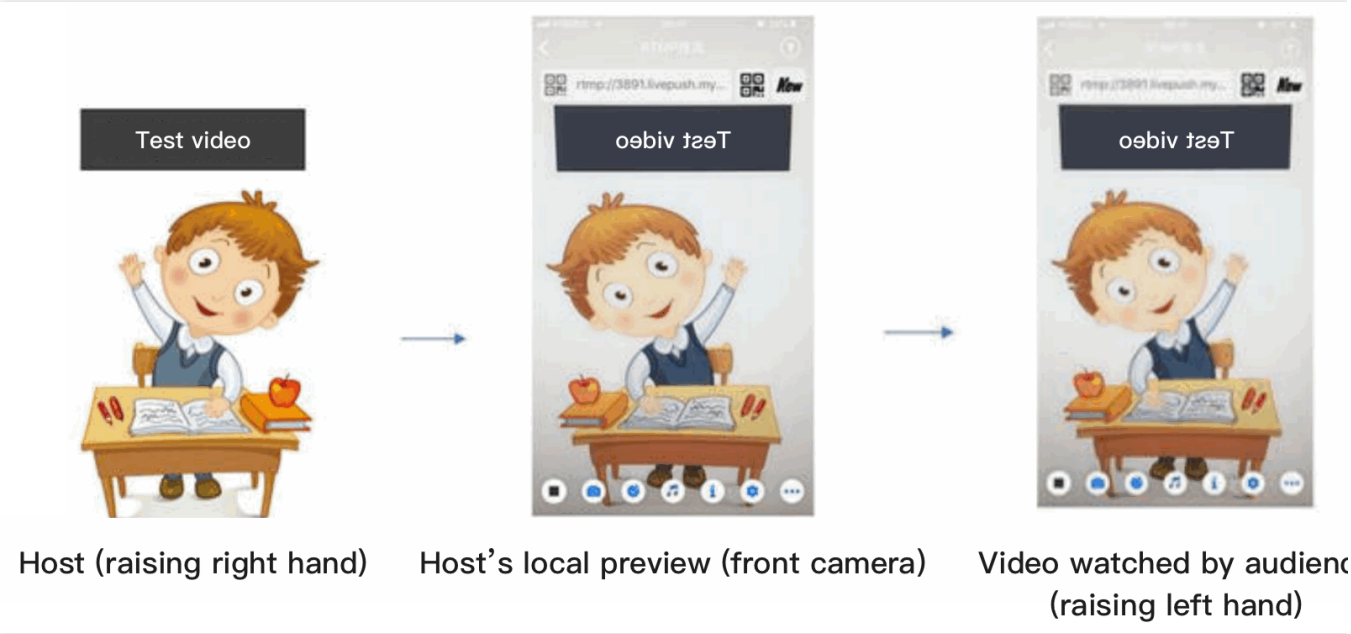
```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@
path = [path stringByAppendingPathComponent:lookupFileName];
UIImage *image = [UIImage imageWithContentsOfFile:path];
[[_pusher getBeautyManager] setFilter:image];
[[_pusher getBeautyManager] setFilterStrength:0.5f];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

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

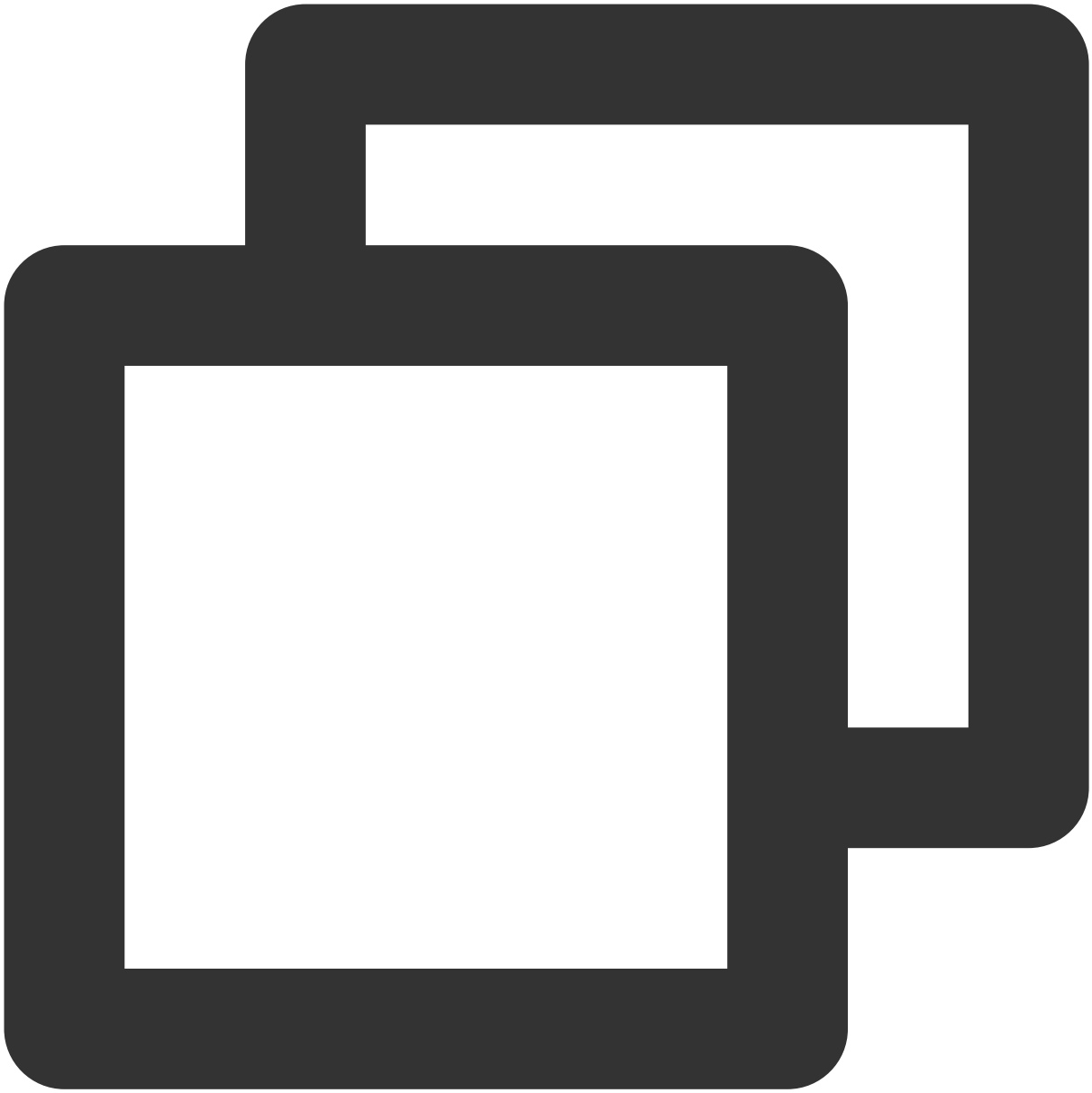
通过调用 V2TXLivePusher 的 setEncoderMirror 可以改变观众端观看到的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，默认情况下自己看到的画面会被 SDK 反转，这时主播就像照镜子一样，观众看到的效果和主播看到的是一致的。如下



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）。

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。

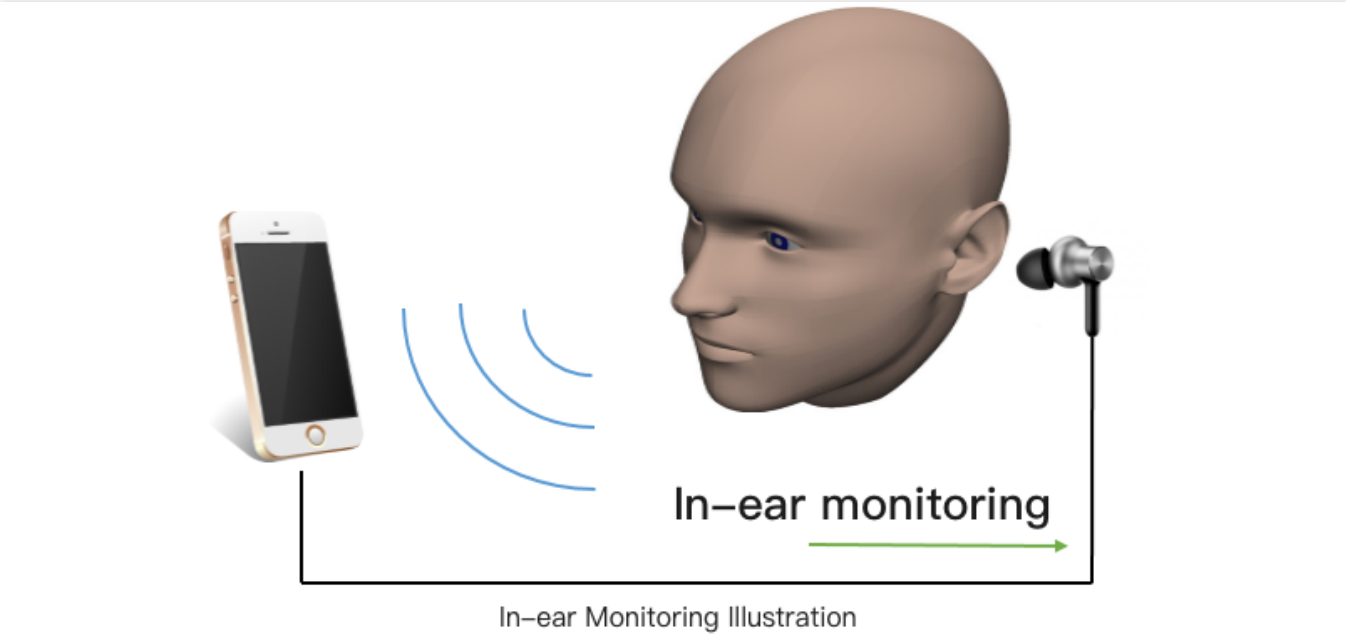


```
V2TXLiveVideoEncoderParam *videoEncoderParam = [[V2TXLiveVideoEncoderParam alloc] i
videoEncoderParam.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionModeLa
[_pusher setVideoQuality:videoEncoderParam];
```

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。

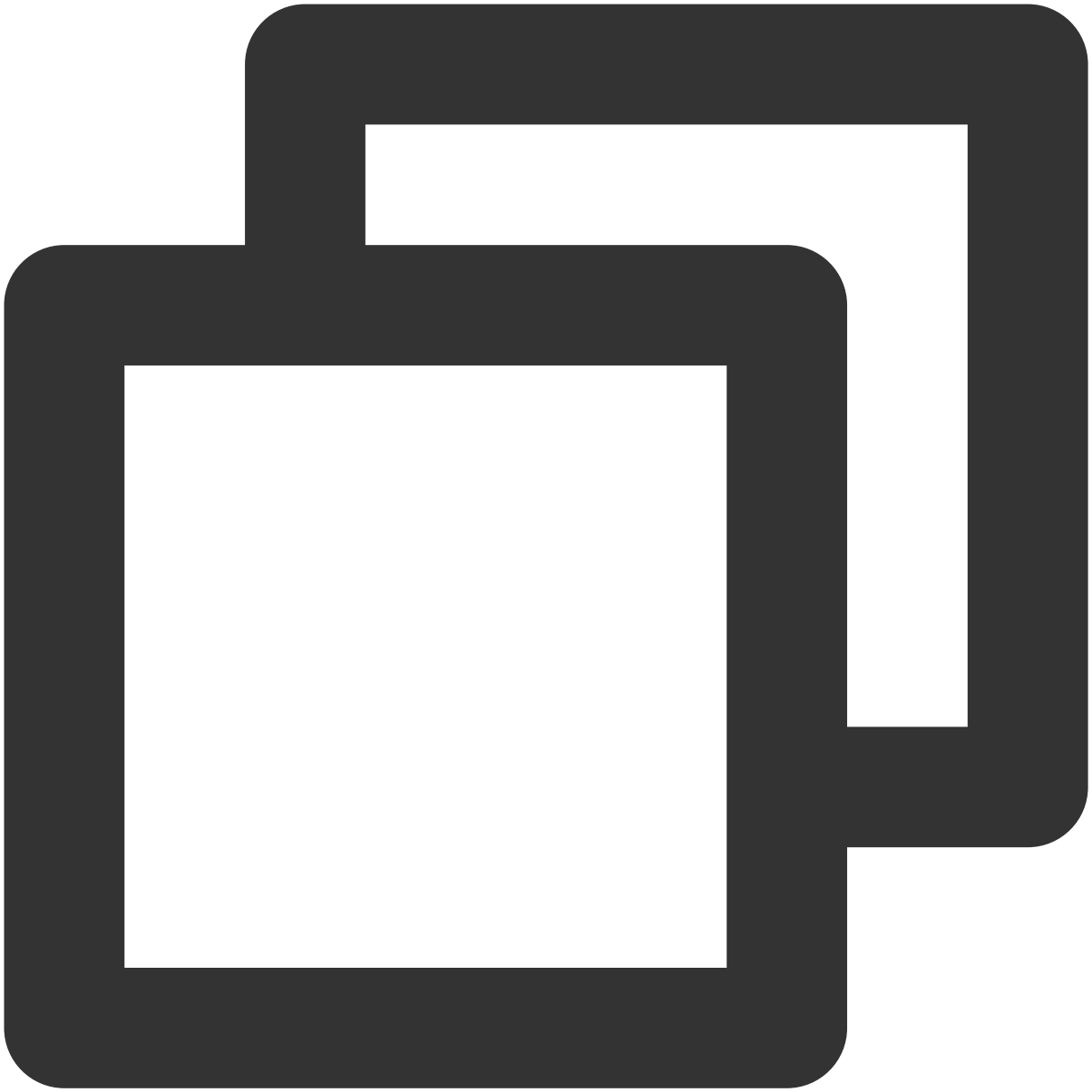


In-ear Monitoring Illustration

说明：
详细用法请参见 TXAudioEffectManager API。

14. 设置 Logo 水印

设置 V2TXLivePusher 中的 setWatermark 可以让 SDK 在推出的视频流中增加一个水印，水印位置位是由传入参数 `(x, y, scale)` 所决定。
SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。
`(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`(0.1, 0.1, 0.1)`，那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。

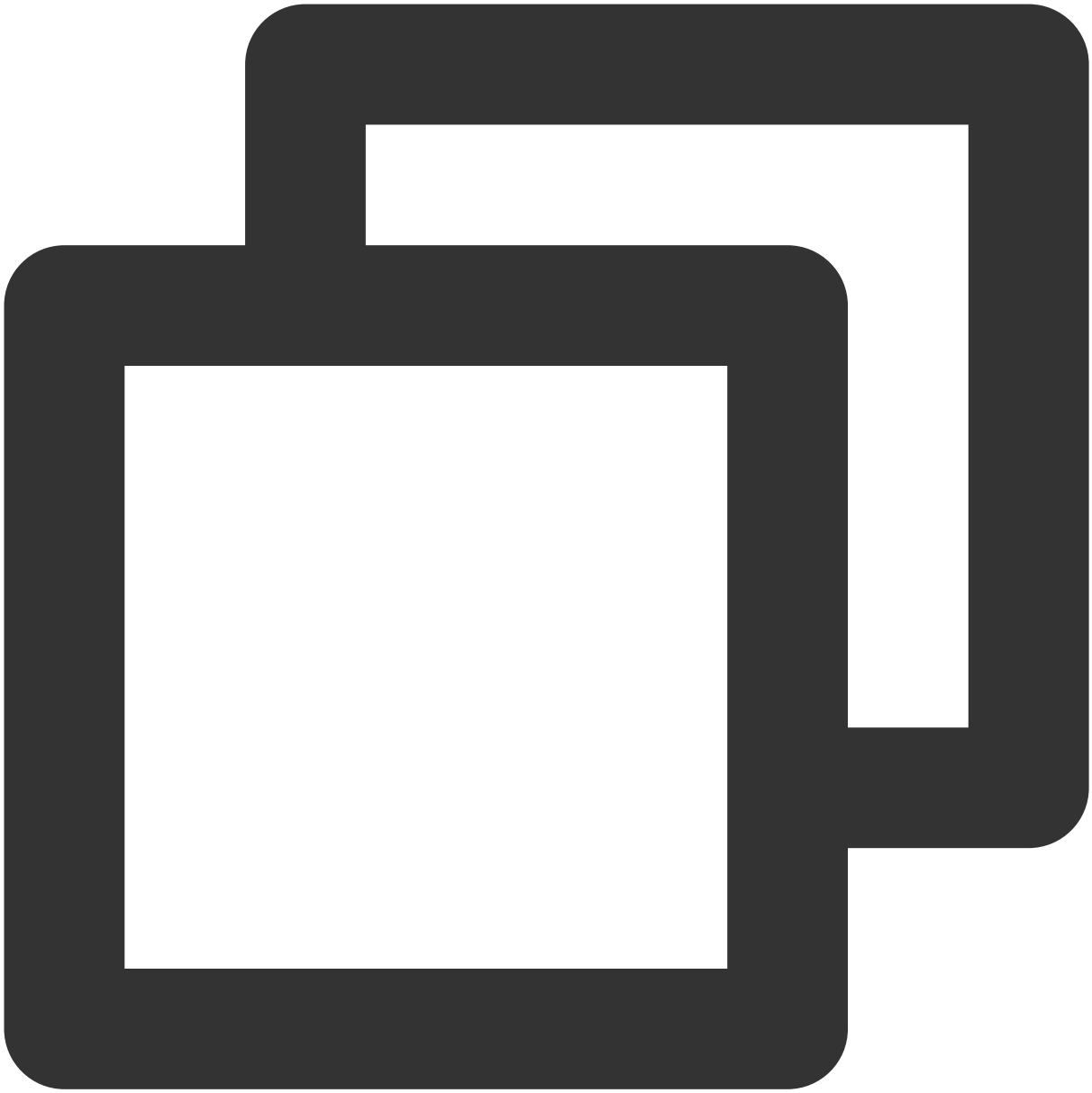


```
// 设置视频水印
[_pusher setWatermark:[UIImage imageNamed:@"watermark"] x:0.03 y:0.015 scale:1];
```

15. 主播端弱网提醒

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

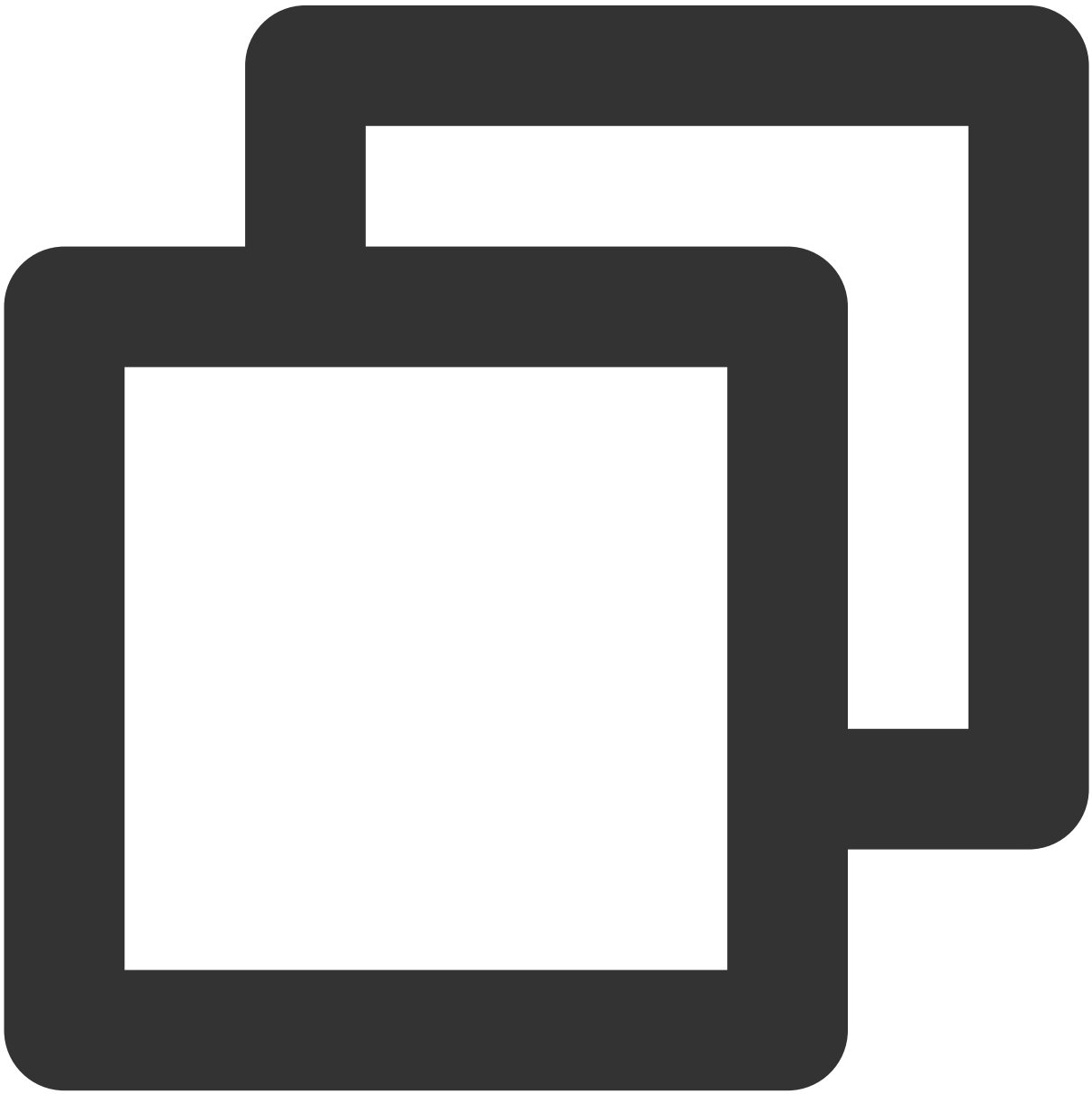
调用 V2TXLivePusher 中的 sendSeiMessage 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

答题直播：推流端将题目下发到观众端，可以做到“音-画-题”完美同步。

秀场直播：推流端将歌词下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。

在线教育：推流端将激光笔和涂鸦操作下发到观众端，可以在播放端实时地划圈划线。

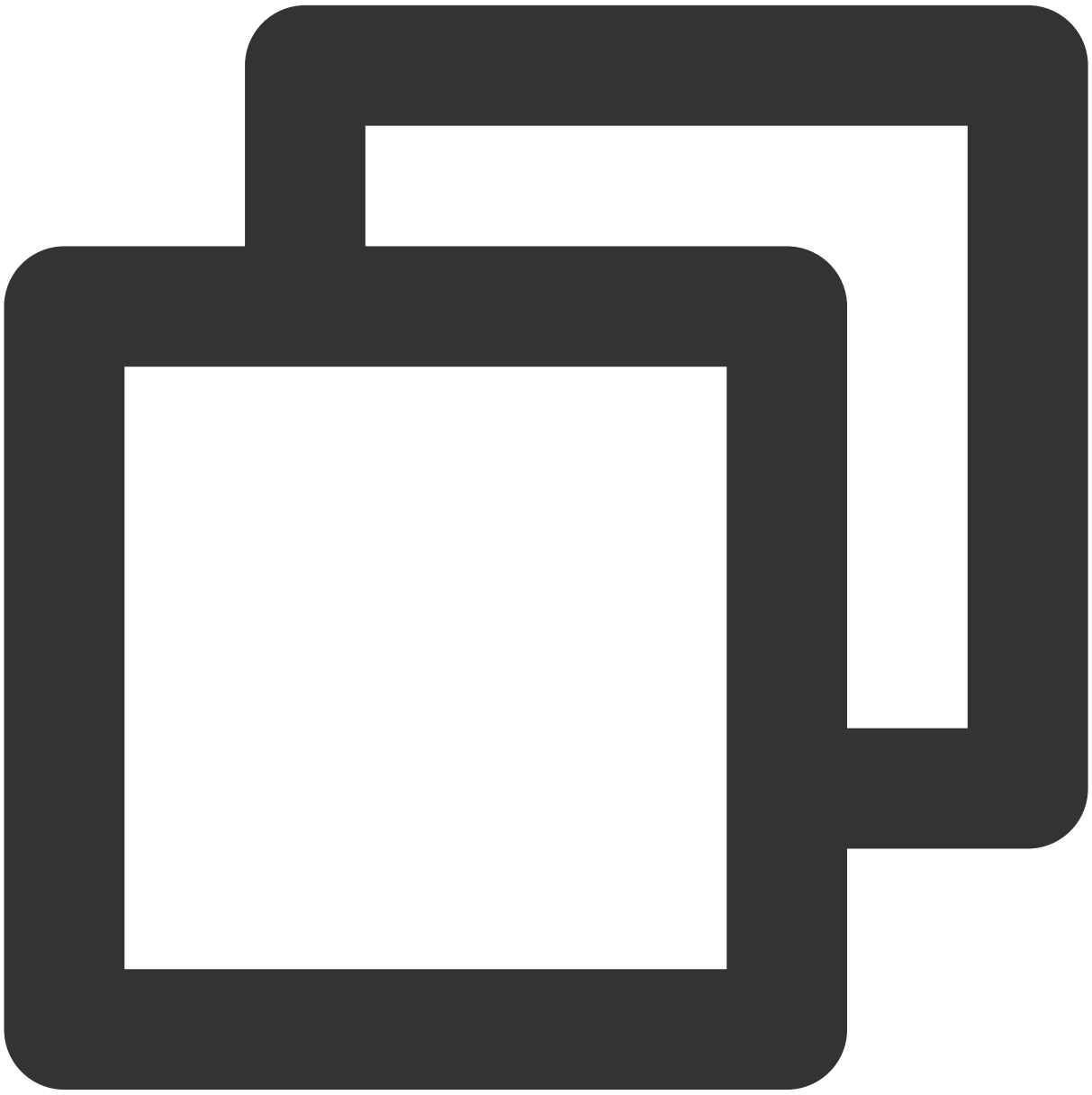
由于自定义消息是直接被塞入视频数据中的，所以不能太大（几个字节比较合适），一般常用于塞入自定义的时间戳等信息。



```
int payloadType = 5;
NSString* msg = @"test";
[_pusher sendSeiMessage:payloadType data:[msg dataUsingEncoding:NSUTF8StringEncodin
```

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

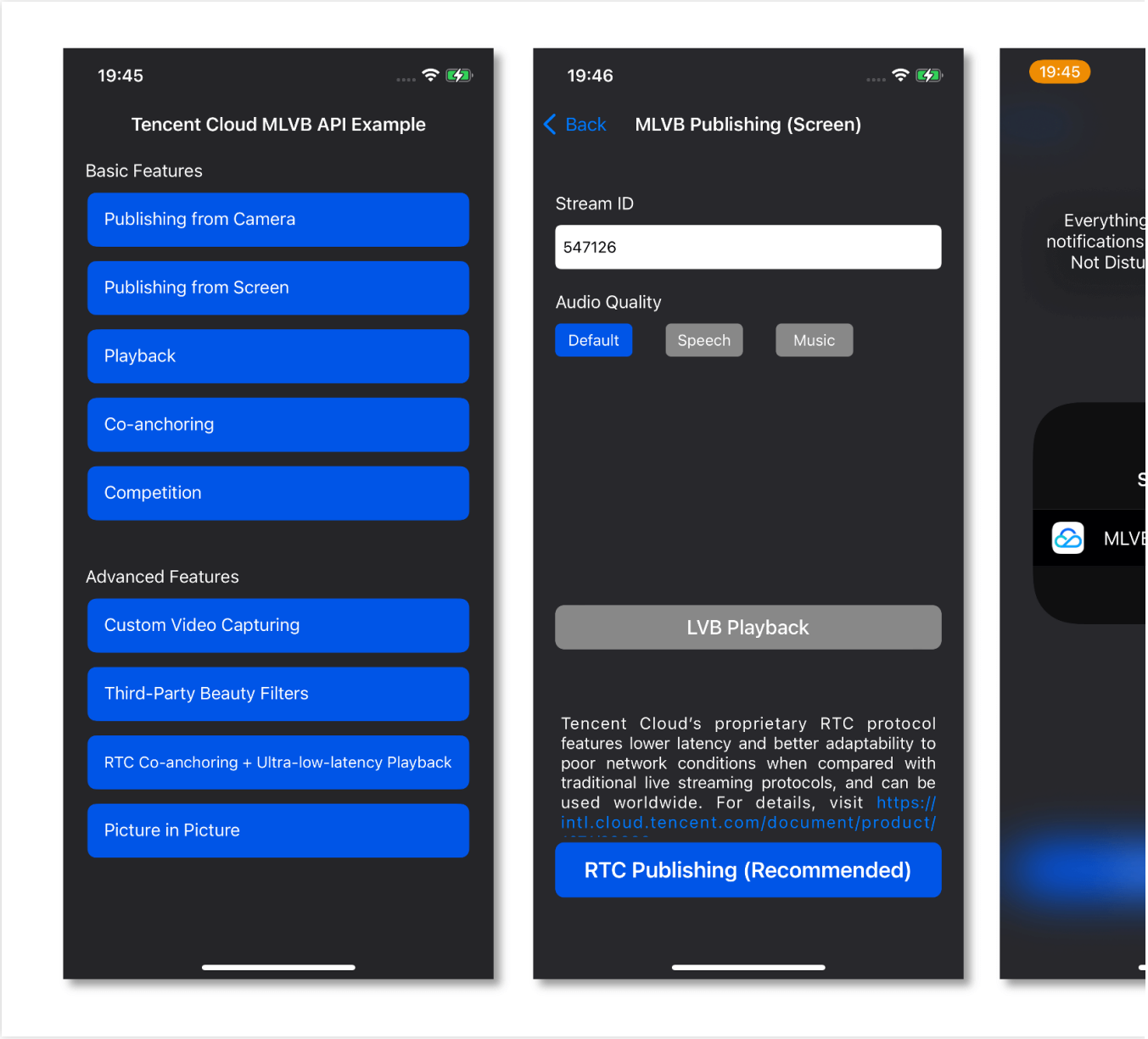
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	麦克风打开失败
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-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：2024-01-13 15:53:49

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。

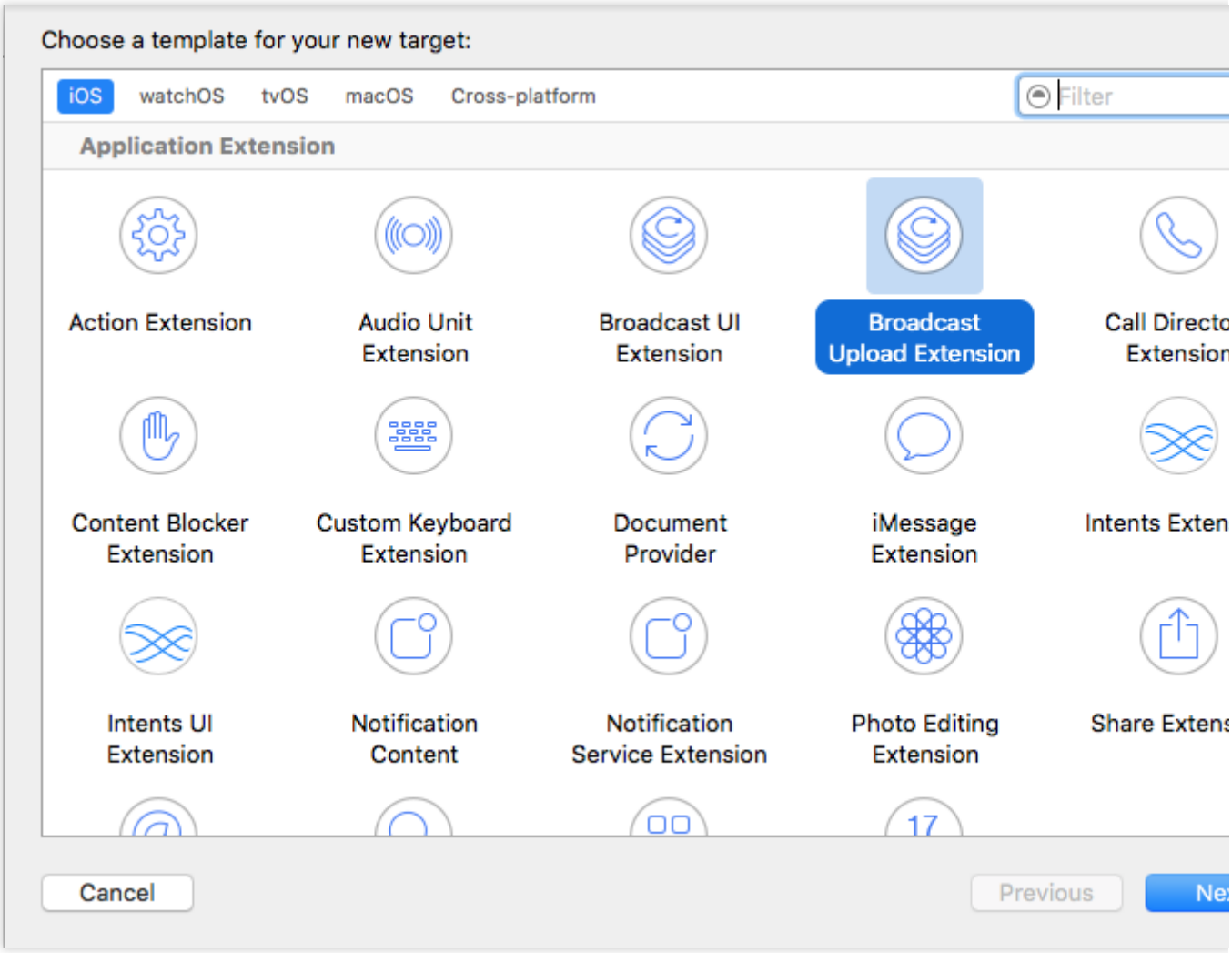
所属平台	GitHub 地址
iOS	LivePushScreenViewController.m
Android	LivePushScreenActivity.java
Flutter	live_screen_push.dart

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。

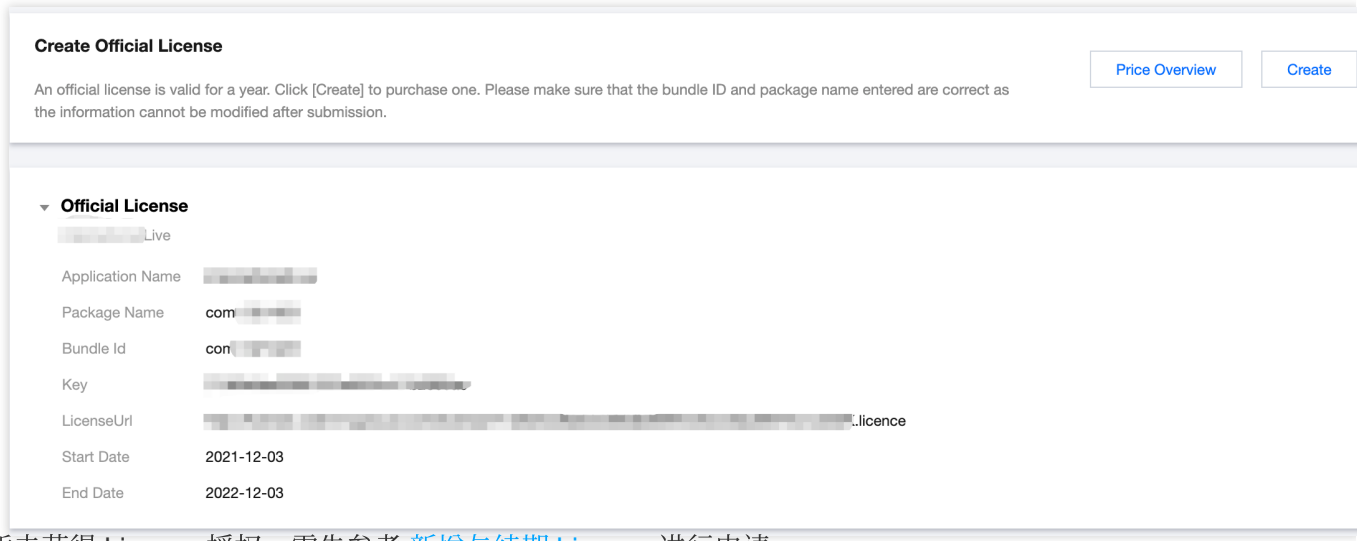
对接攻略

1. 下载 SDK 开发包

下载 SDK 开发包，并按照 [SDK 集成指引](https://www.tencentcloud.com/document/product/1071/38155 将 SDK 嵌入您的 App 工程中。

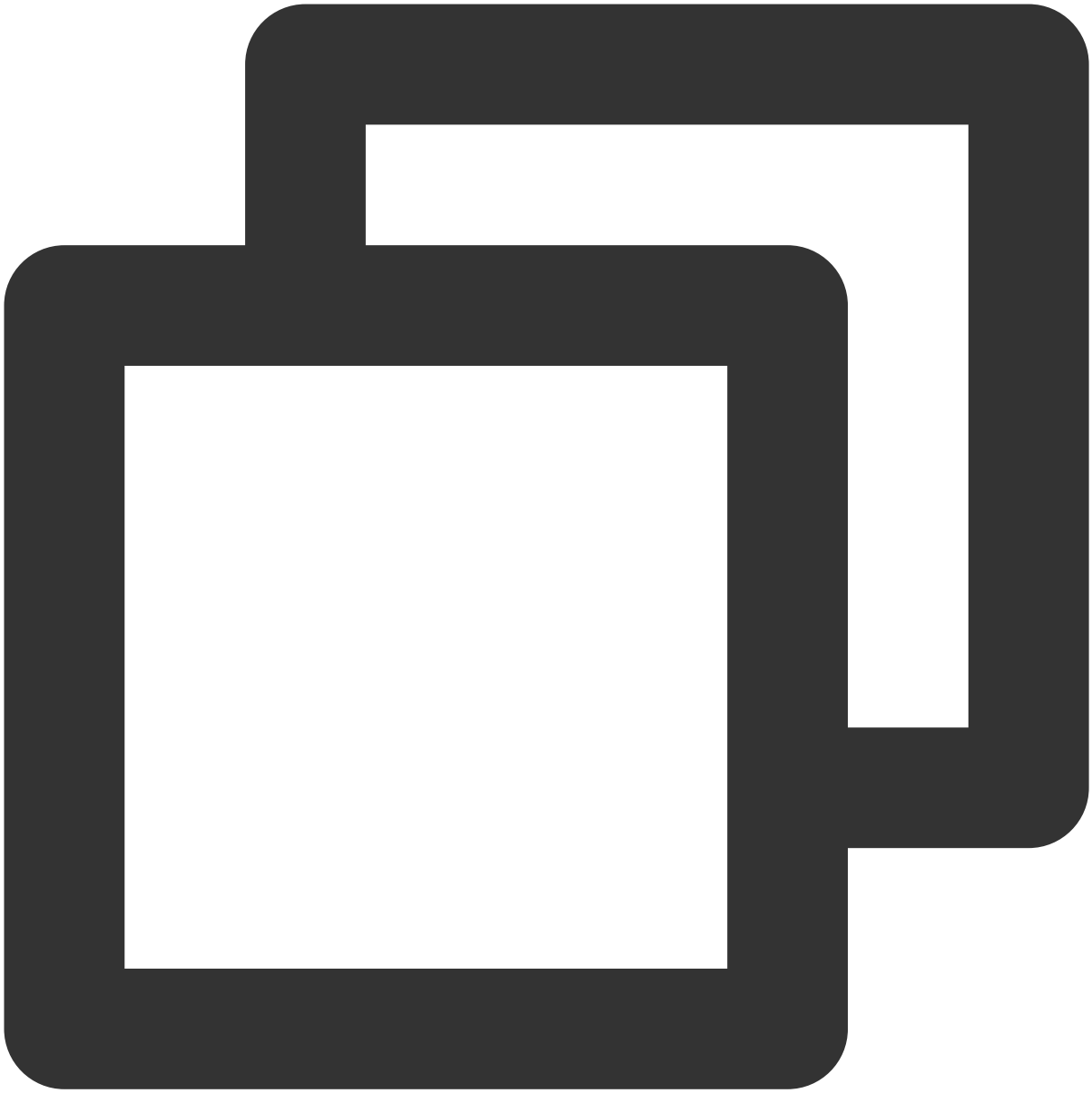
2. 给 SDK 配置 License 授权

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:(NSD
    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;
}
```

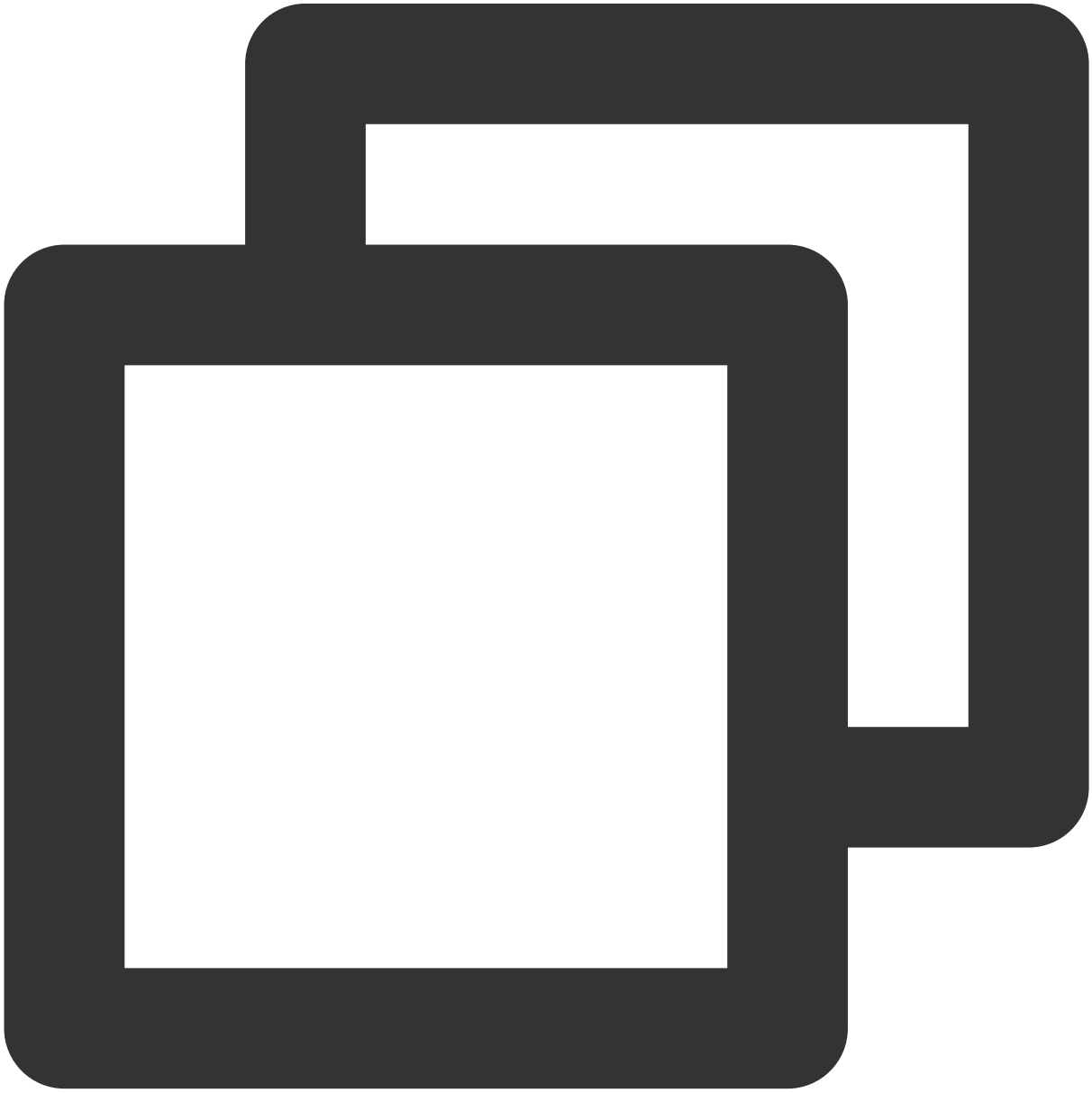
```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
```

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

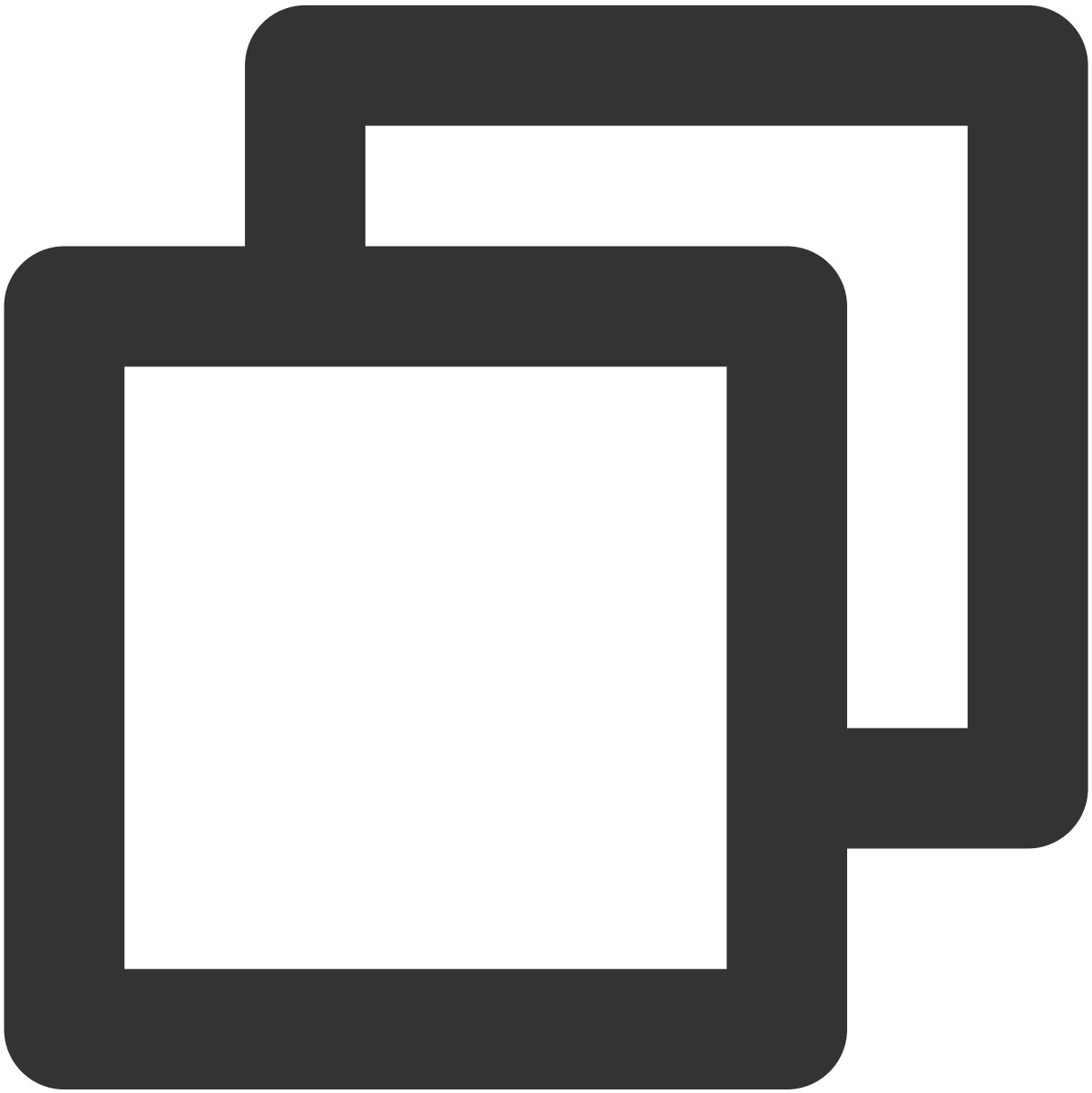
创建一个 `V2TXLivePusher` 对象，需要指定对应的 `V2TXLiveMode` 。



```
// 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
```

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 *> *)setupI
    // App group ID, 此处 ID 与 V2TXLivePusher startScreenCapture 对应, 可以指定为 nil,
    [[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:(RPSampleBuffe
    [[TXReplayKitExt sharedInstance] sendSampleBuffer:sampleBuffer withType:sampleB
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)
{
    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 的实现类中的需要调用对应 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。

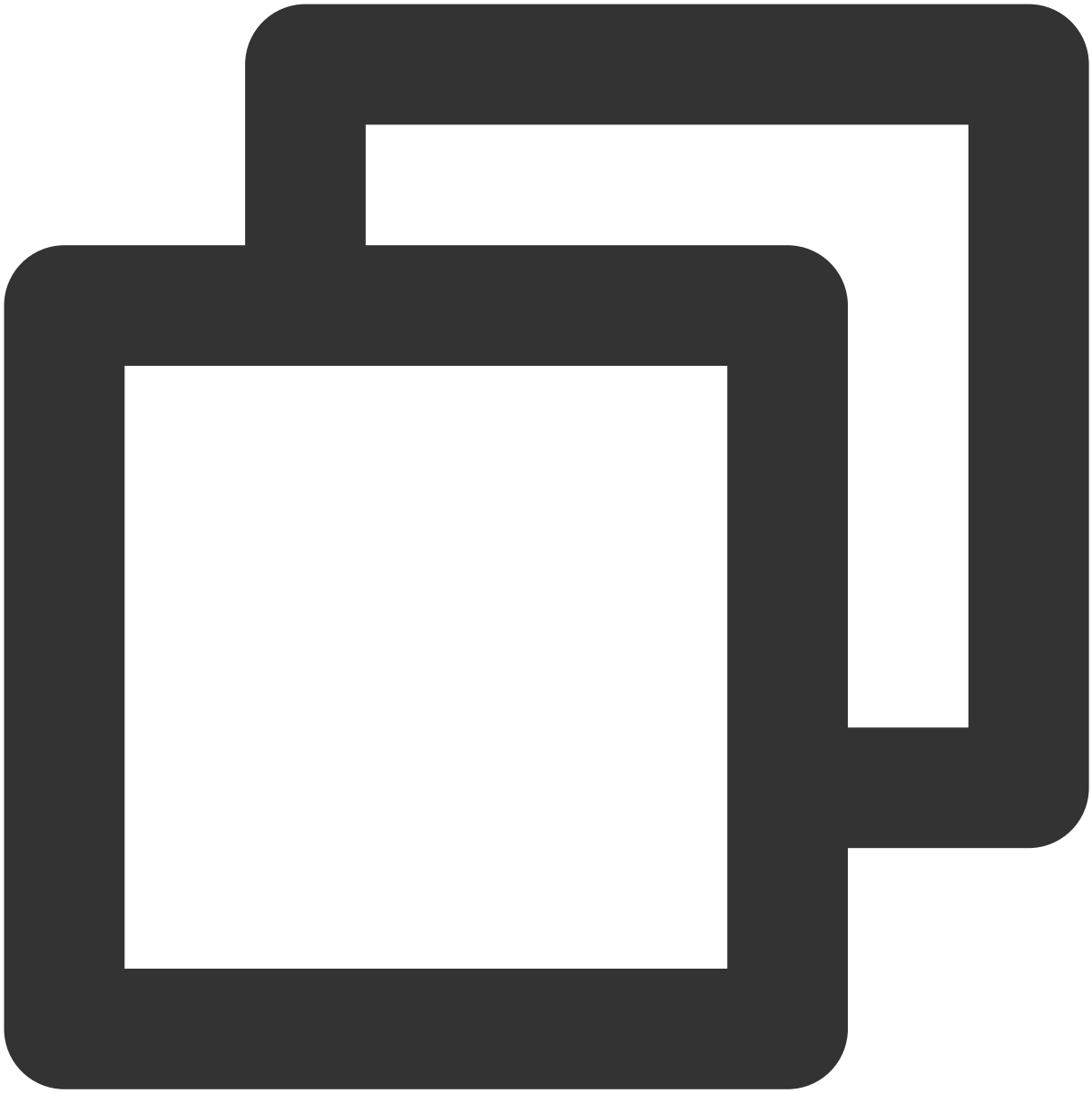
扩展与主 App 间的通信请参见 扩展与宿主 App 之间的通信与数据传递方式 。

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。



```
// appGroup 主 App 与 Broadcast 共享的 Application Group Identifier，可以指定为 nil，但
[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=xxxxxx
V2TXLiveCode code = [livePusher startPush:url];
if (code != V2TXLIVE_OK) {
    // 请检查错误码
}
```

返回 V2TXLIVE_ERROR_INVALID_LICENSE 的原因：

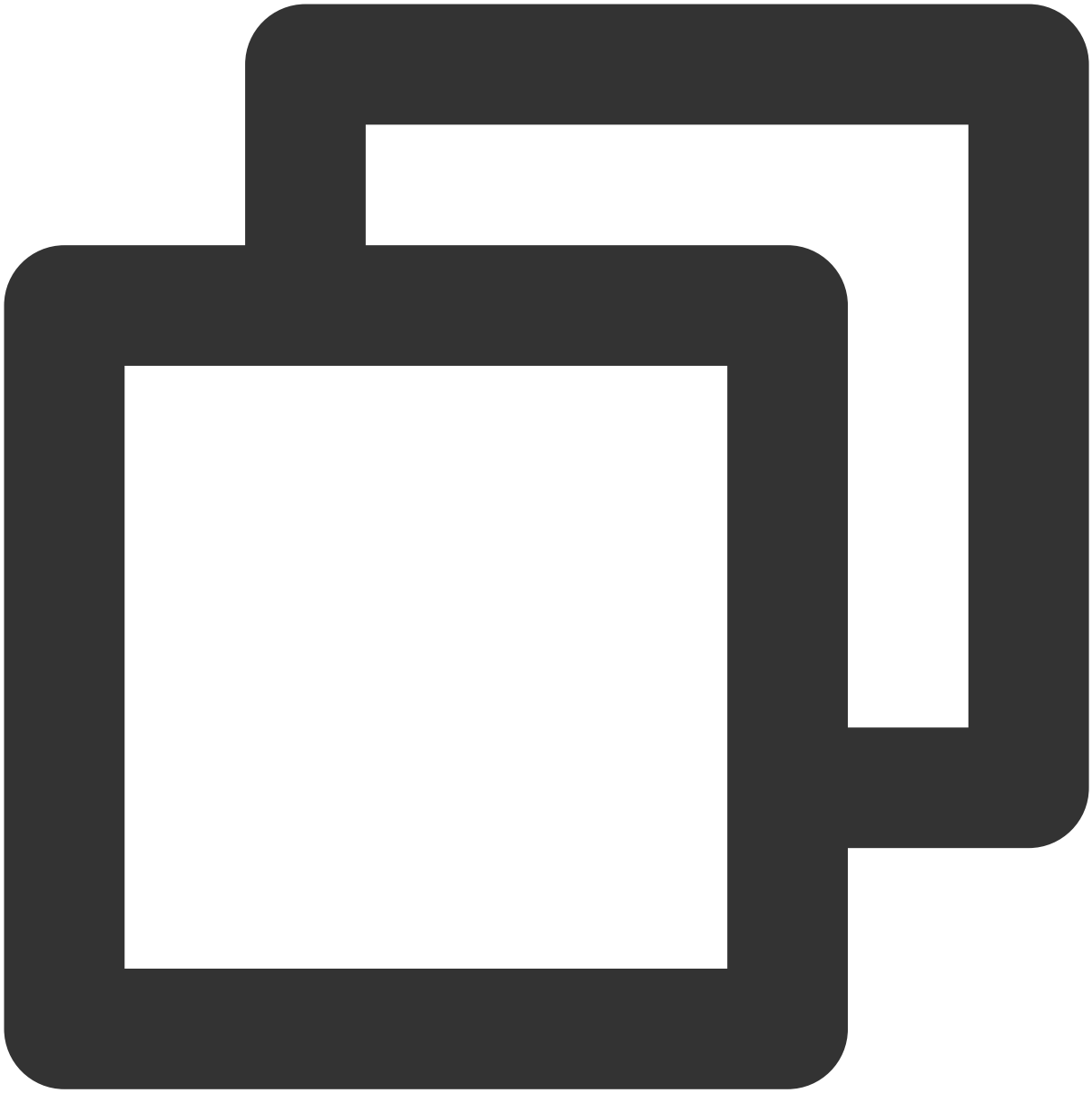
如果 `startPush` 接口返回 `V2TXLIVE_ERROR_INVALID_LICENSE`，则代表您的 License 校验失败了，请检查第2步：给 SDK 配置 License 授权 中回调 `onLicenceLoaded` 的错误码和错误描述。

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：



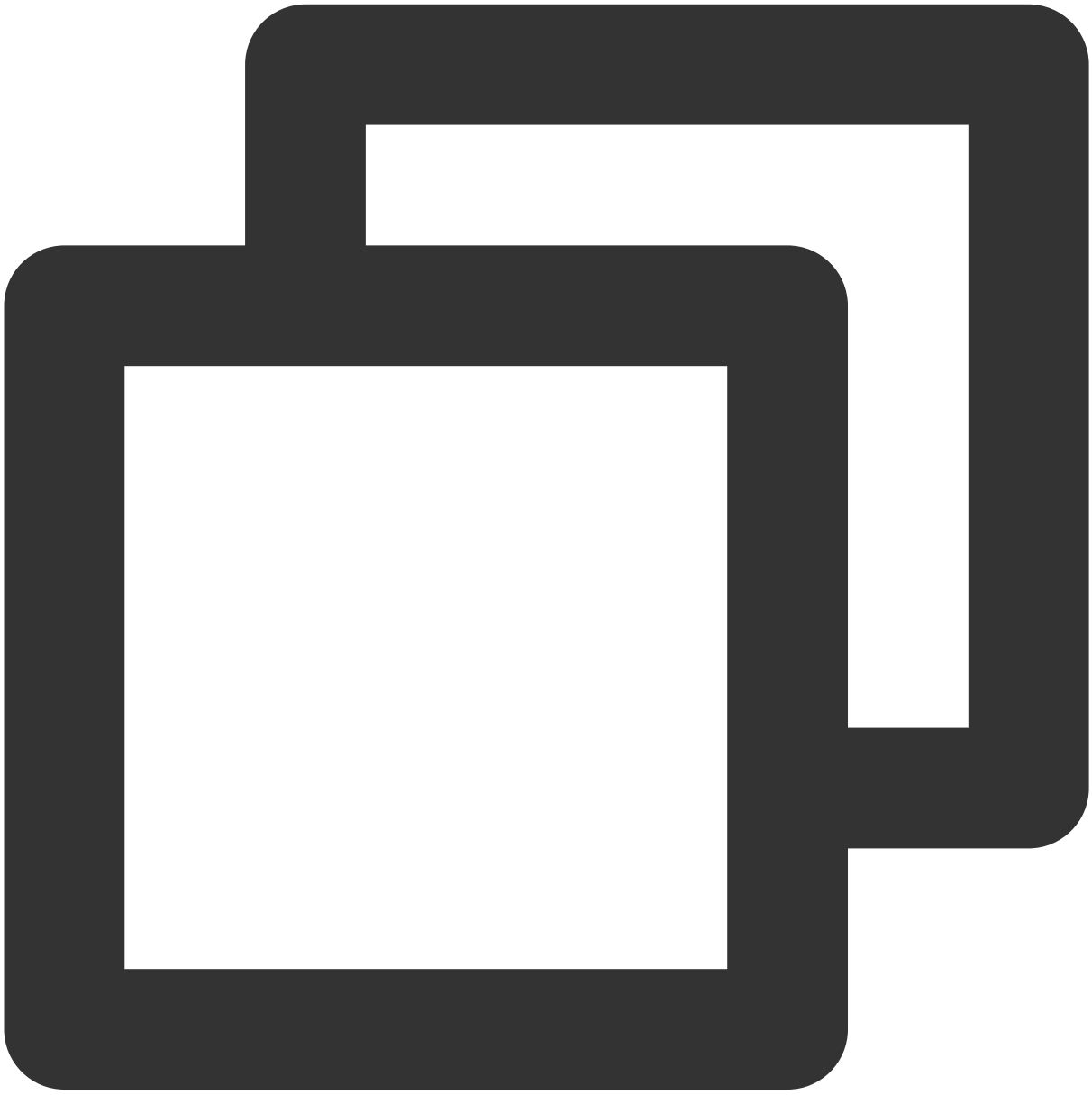
```
BOOL landScape; //YES:横屏,  NO:竖屏
V2TXLiveVideoEncoderParam *videoParam = [[V2TXLiveVideoEncoderParam alloc] init
videoParam.videoResolution = V2TXLiveVideoResolution960x540;
videoParam.videoResolutionMode = landScape ? V2TXLiveVideoResolutionModeLandsca
[livePusher setVideoQuality:videoParam];
```

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，水印位置位是由传入参数 `(x, y, scale)` 所决定。

SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。

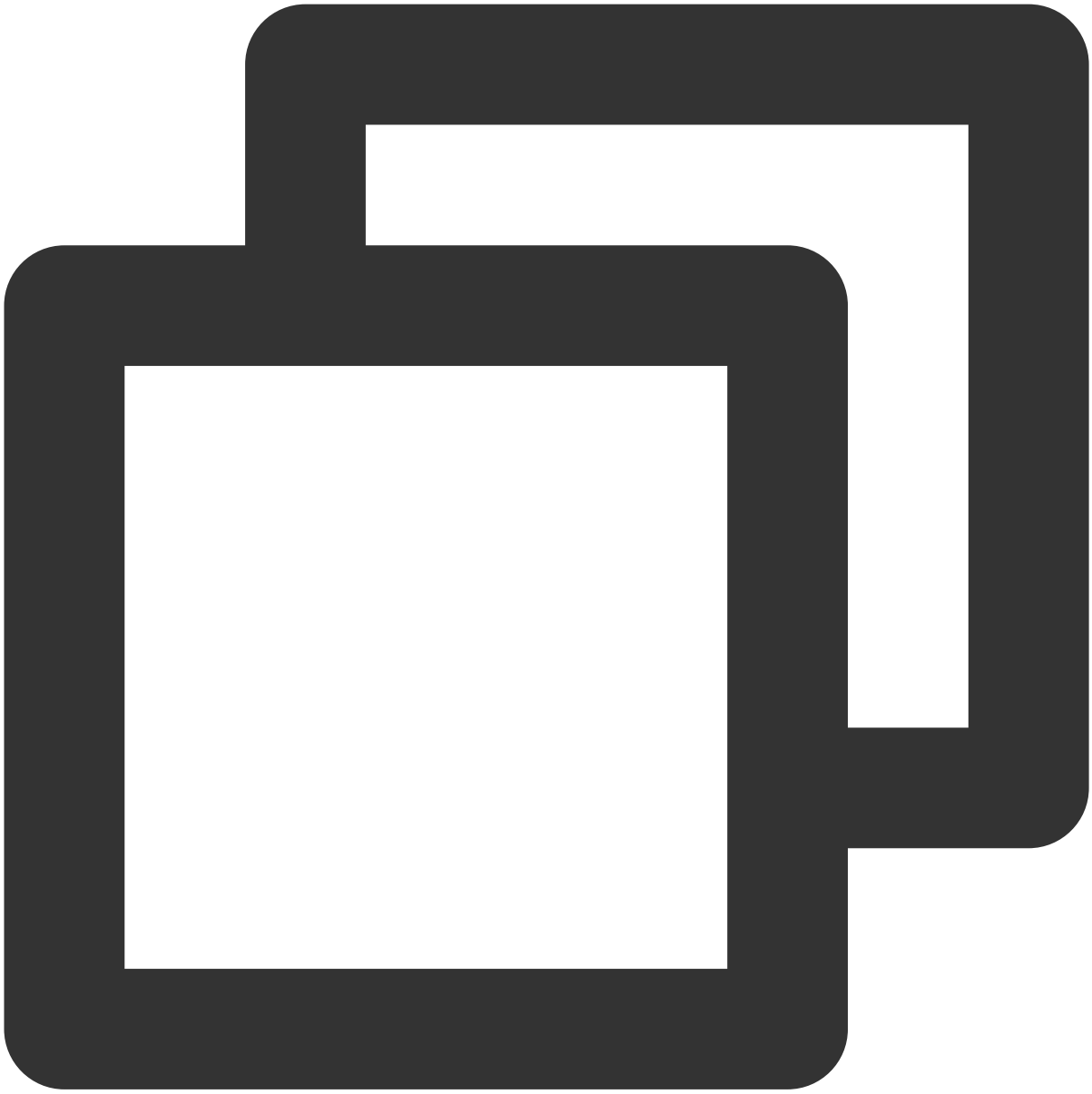
`(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`（0.1，0.1，0.1）`，那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。



```
// 设置视频水印
[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

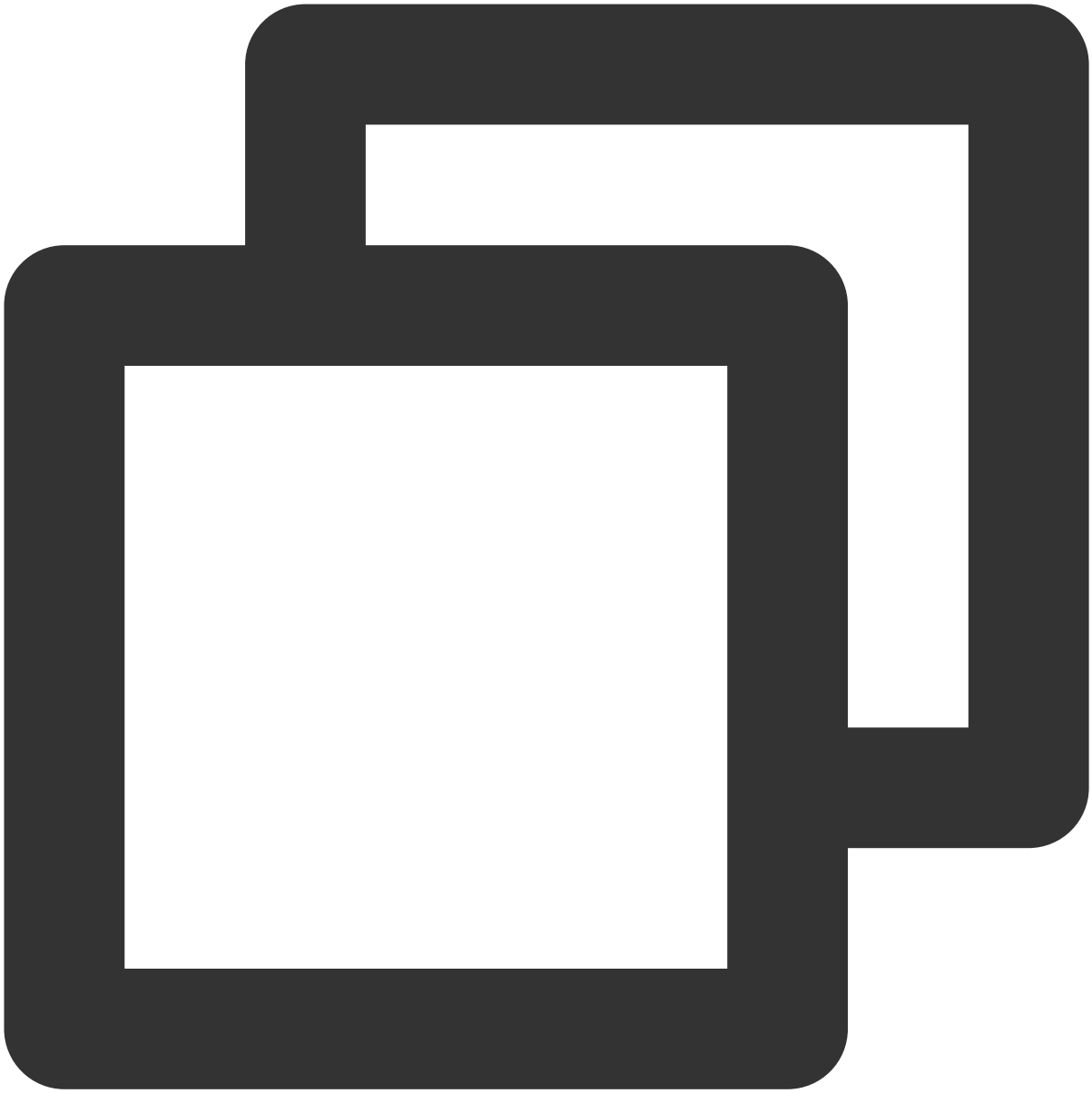
		移动设备正在通话时，打开麦克风会失败。
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	当前系统不支持屏幕分享。
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	开始录屏失败，如果在移动设备出现，可能是权限被用户拒绝了。
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	录屏被系统中断。

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：



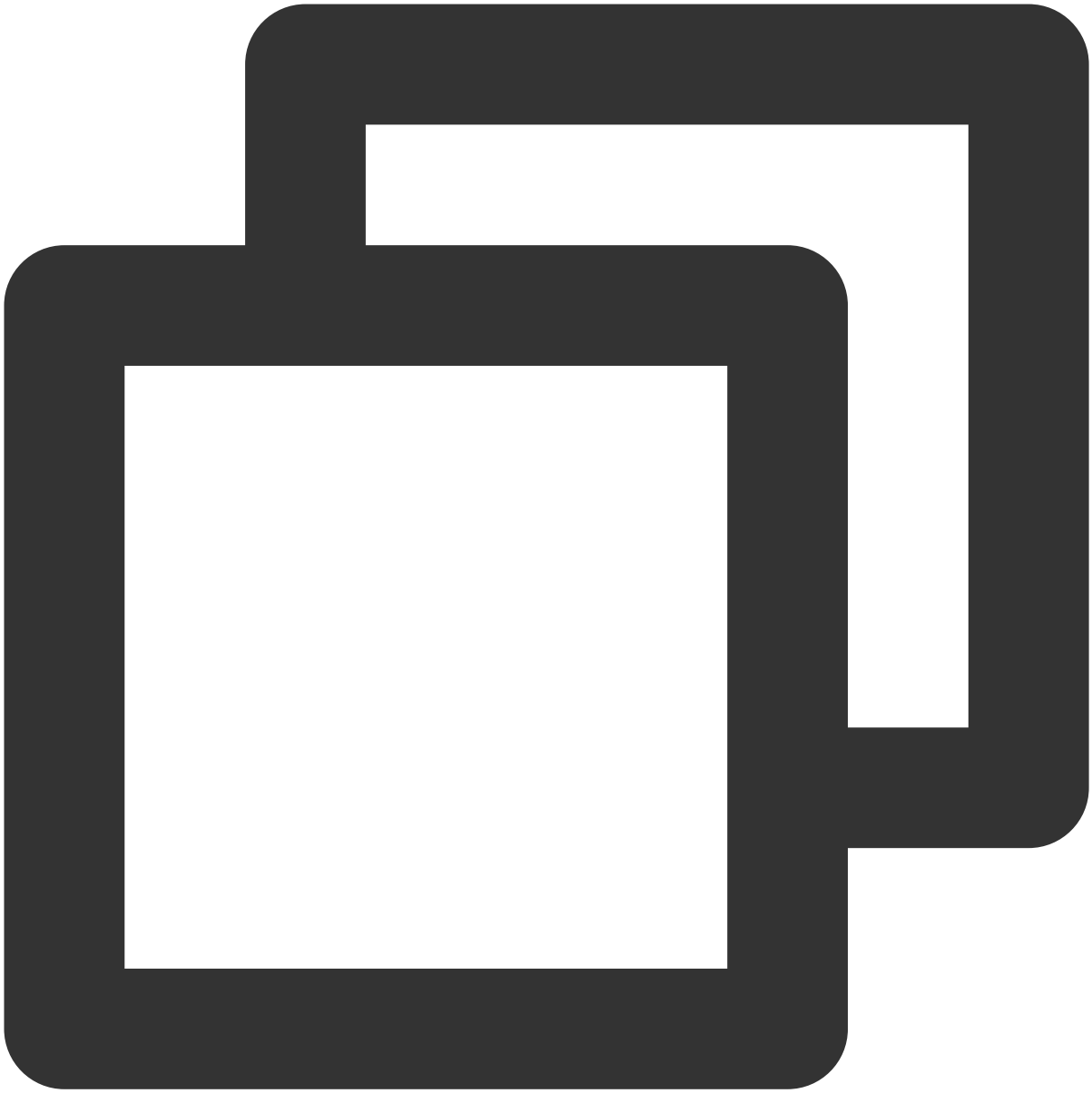
```
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupI
    [self sendLocalNotificationToHostAppWithTitle:@"腾讯云录屏推流" msg:@"录屏已开始，请
}

- (void)sendLocalNotificationToHostAppWithTitle:(NSString*)title msg:(NSString*)msg
{
    UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificatio

    UNMutableNotificationContent* content = [[UNMutableNotificationContent alloc] i
    content.title = [NSString localizedUserNotificationStringForKey:title arguments
    content.body = [NSString localizedUserNotificationStringForKey:msg  arguments:n
```

```
    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 repe

    UNNotificationRequest* request = [UNNotificationRequest requestWithIdentifier:@
                                                                content:c

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

    }];
}
```

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。

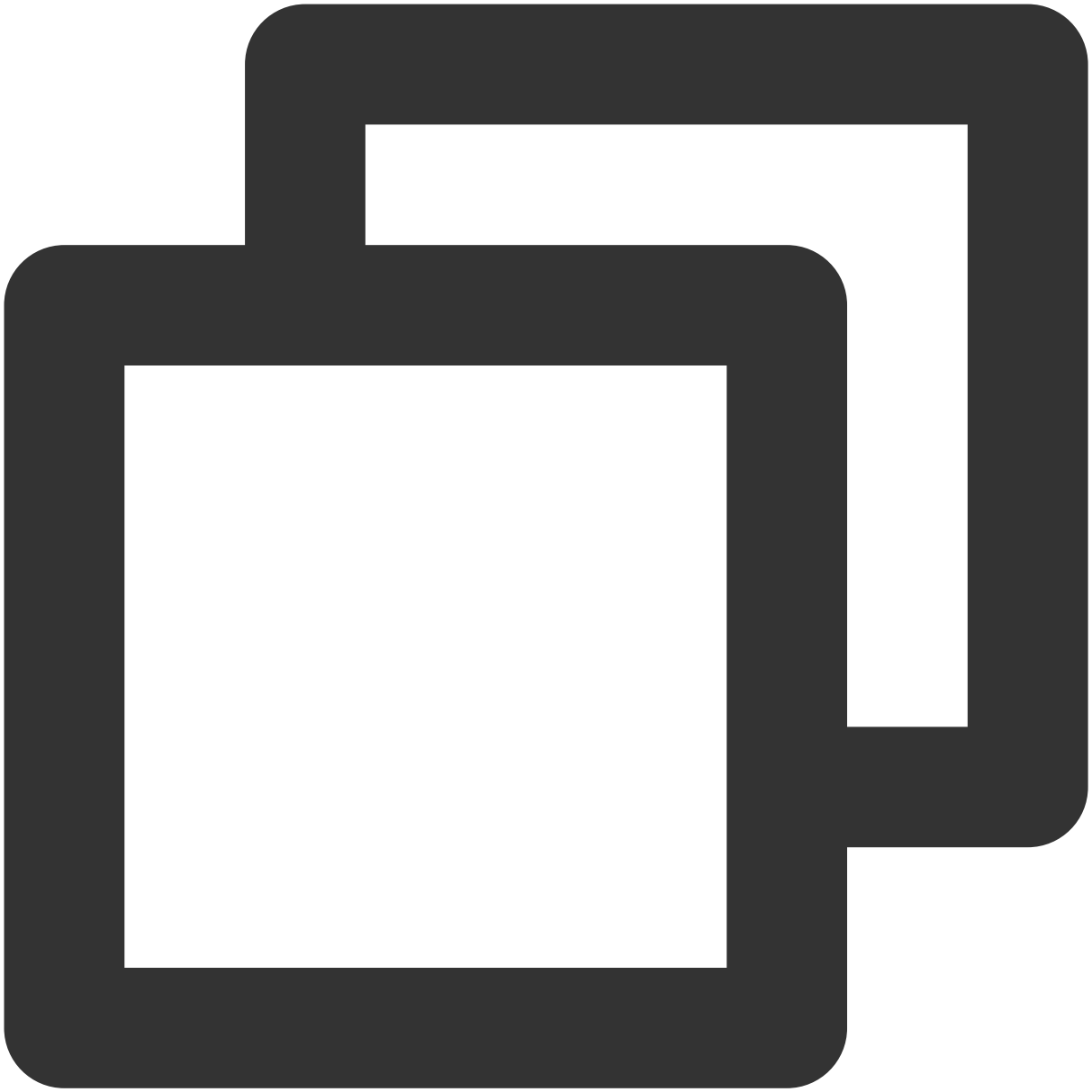
2. 进程间的通知 CFNotificationCenter

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,
                                    CFNotificationSuspensionBehaviorDeliverImmediat

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handl


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_P
}

- (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:kReplayK

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

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

常见问题

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

1. 屏幕录制何时自动会停止？

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

2. 采集推流过程中有时屏幕录制会自动停止问题？

通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2 的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此 ReplayKit2 上建议推流分辨率不高于720P。

3. **iPhoneX 手机的兼容性与画面变形问题？**

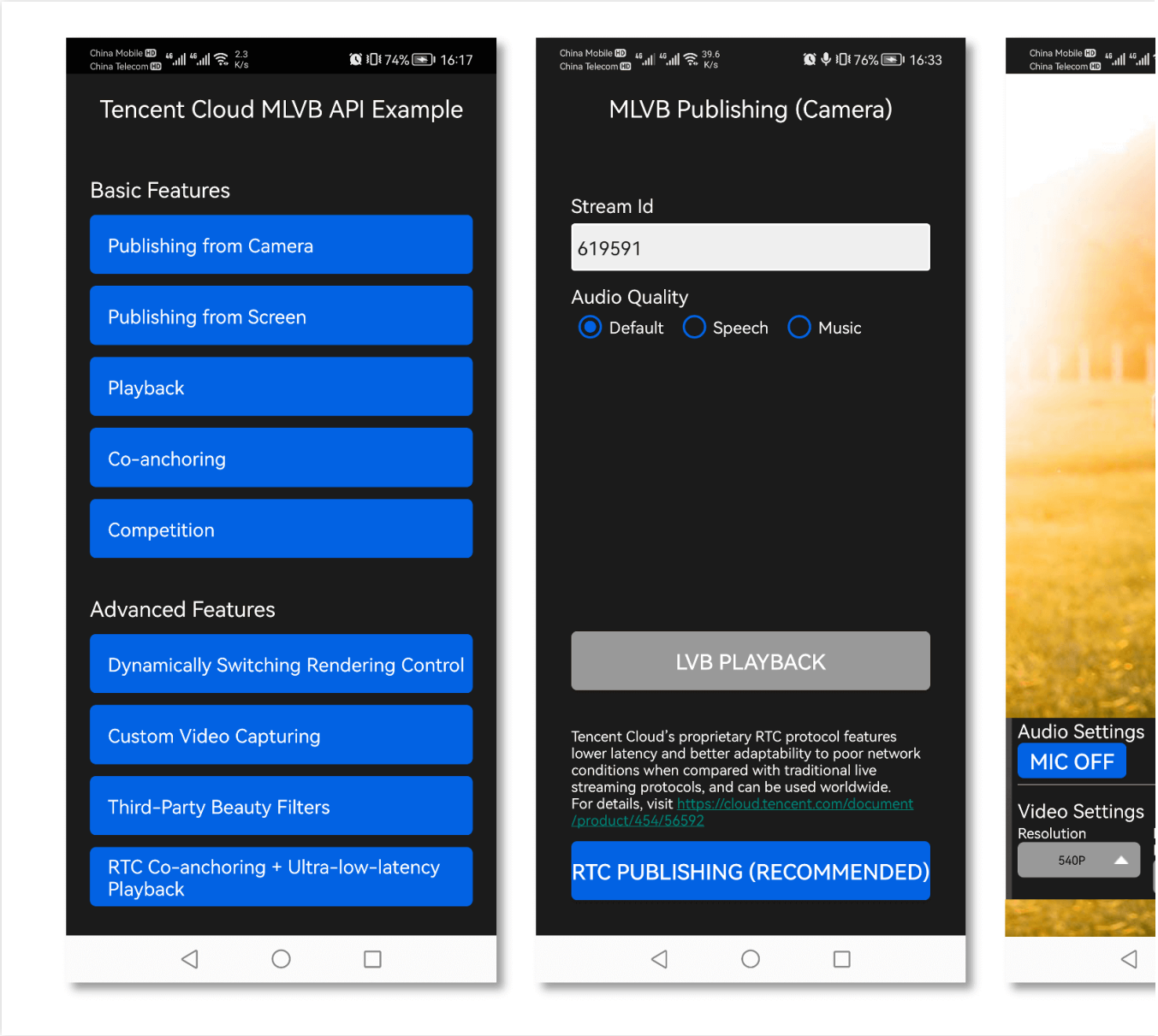
iPhoneX 手机因为有刘海，屏幕采集的画面分辨率不是 9:16。如果设了推流输出分辨率为 9:16 的比例，如高清里是为 960 × 540 的分辨率，这时因为源分辨率不是 9:16 的，推出去的画面就会稍有变形。建议设置分辨率时根据屏幕分辨率比例来设置，拉流端用 AspectFit 显示模式 iPhoneX 的屏幕采集推流会有黑边是正常现象，AspectFill 看画面会不全。

Android
摄像头推流

最近更新时间：2024-01-13 15:53:49

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

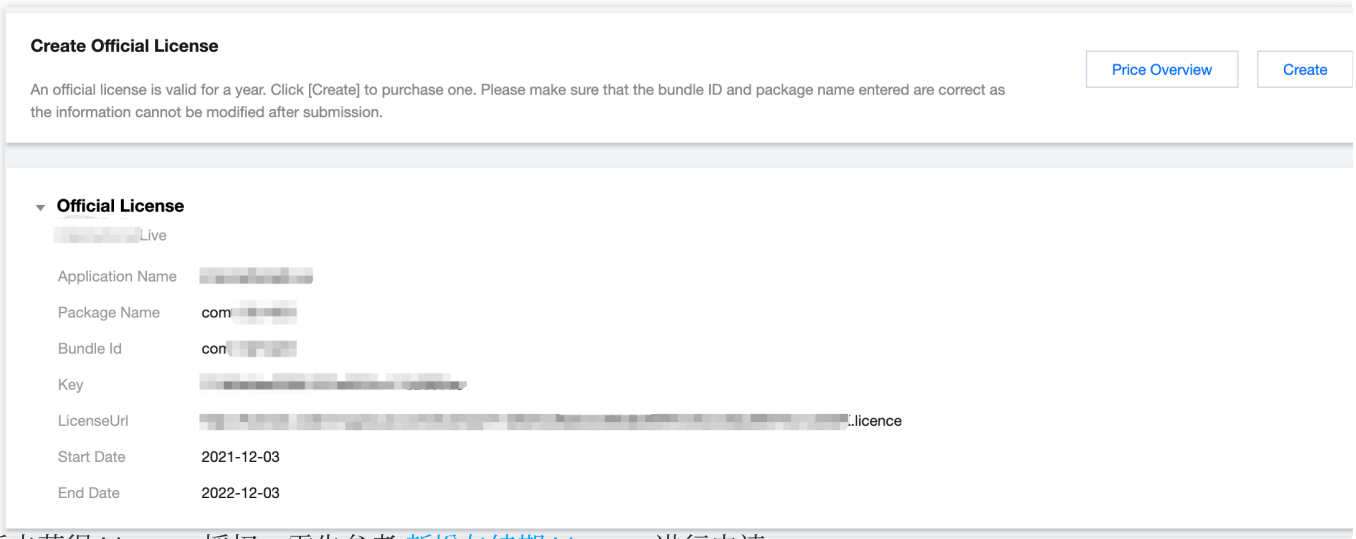
功能对接

1. 下载 SDK 开发包

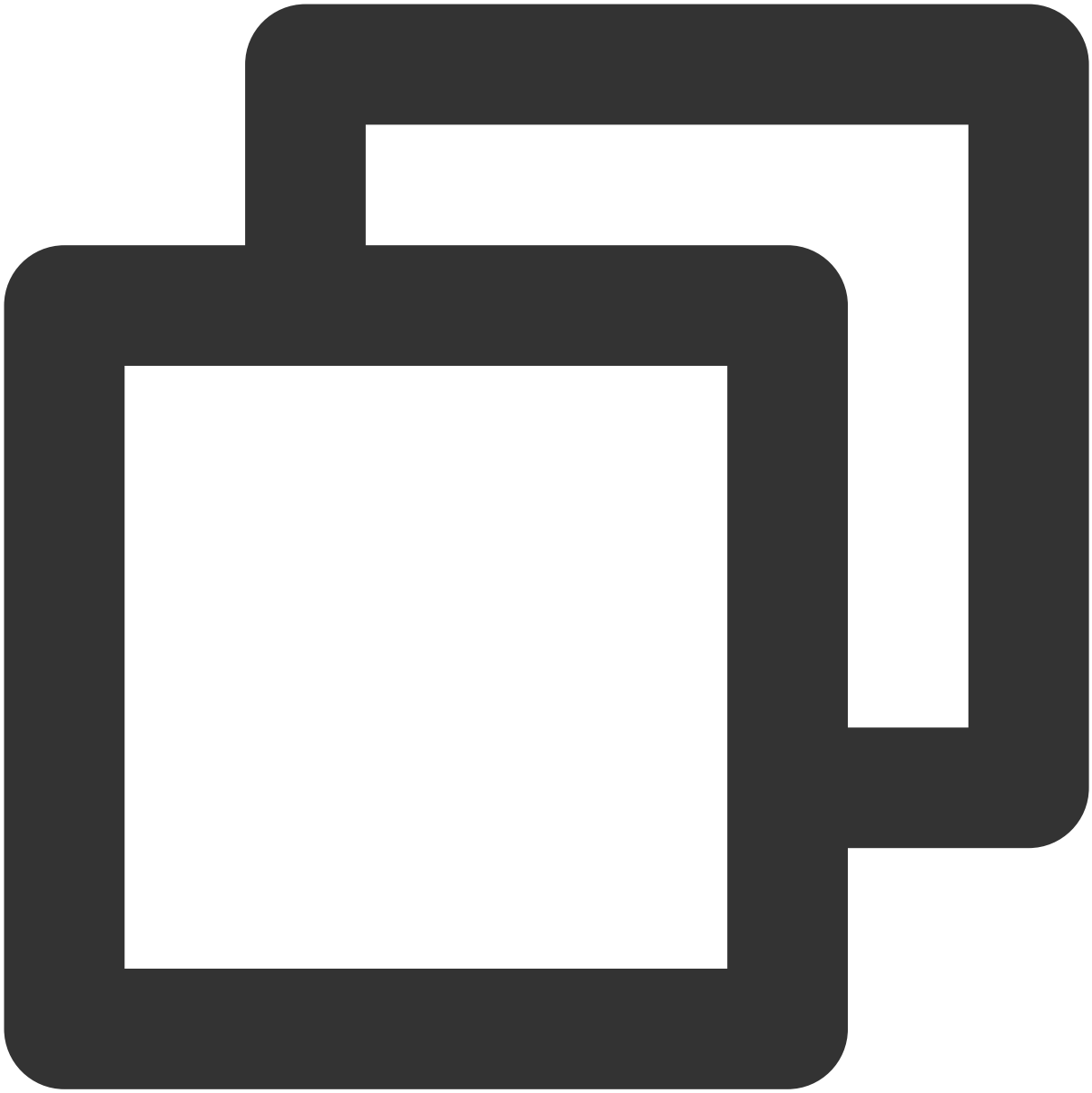
下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

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.setEnvironment("GDPR");  // 设置环境
 V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
 V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
        @Override
```

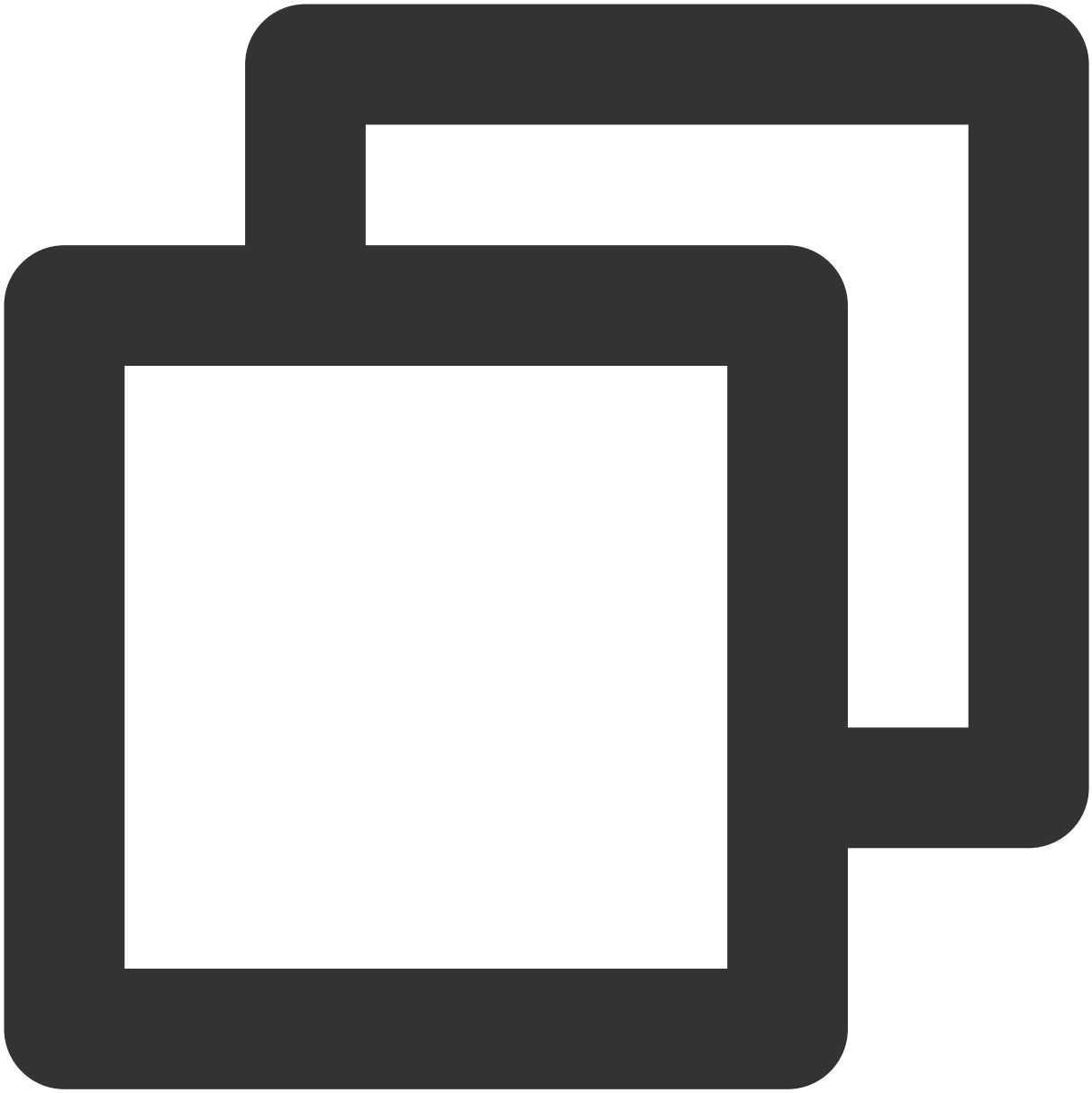
```
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason)
            }
        });
    }
```

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。

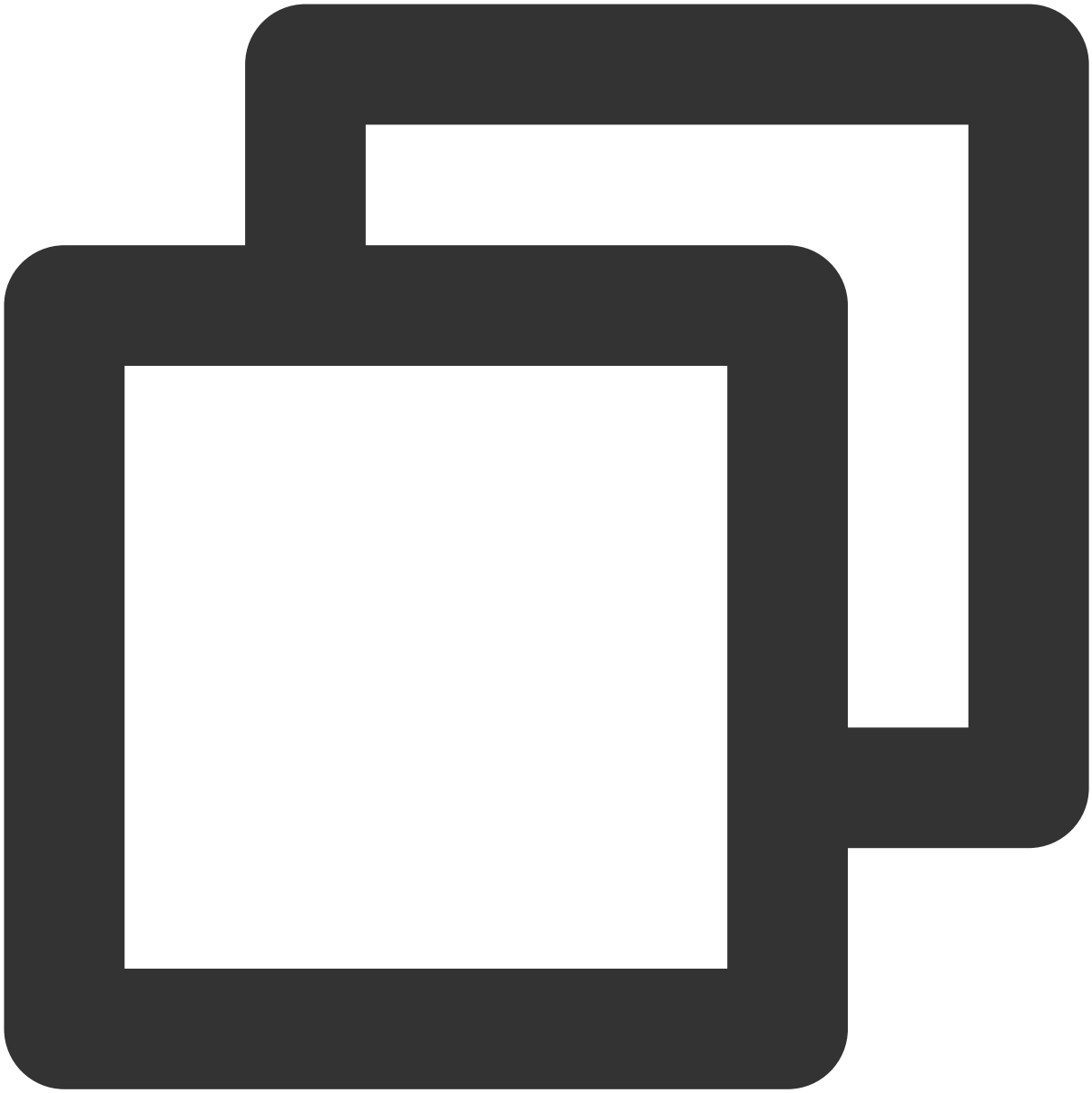


```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.
```

4. 开启摄像头预览

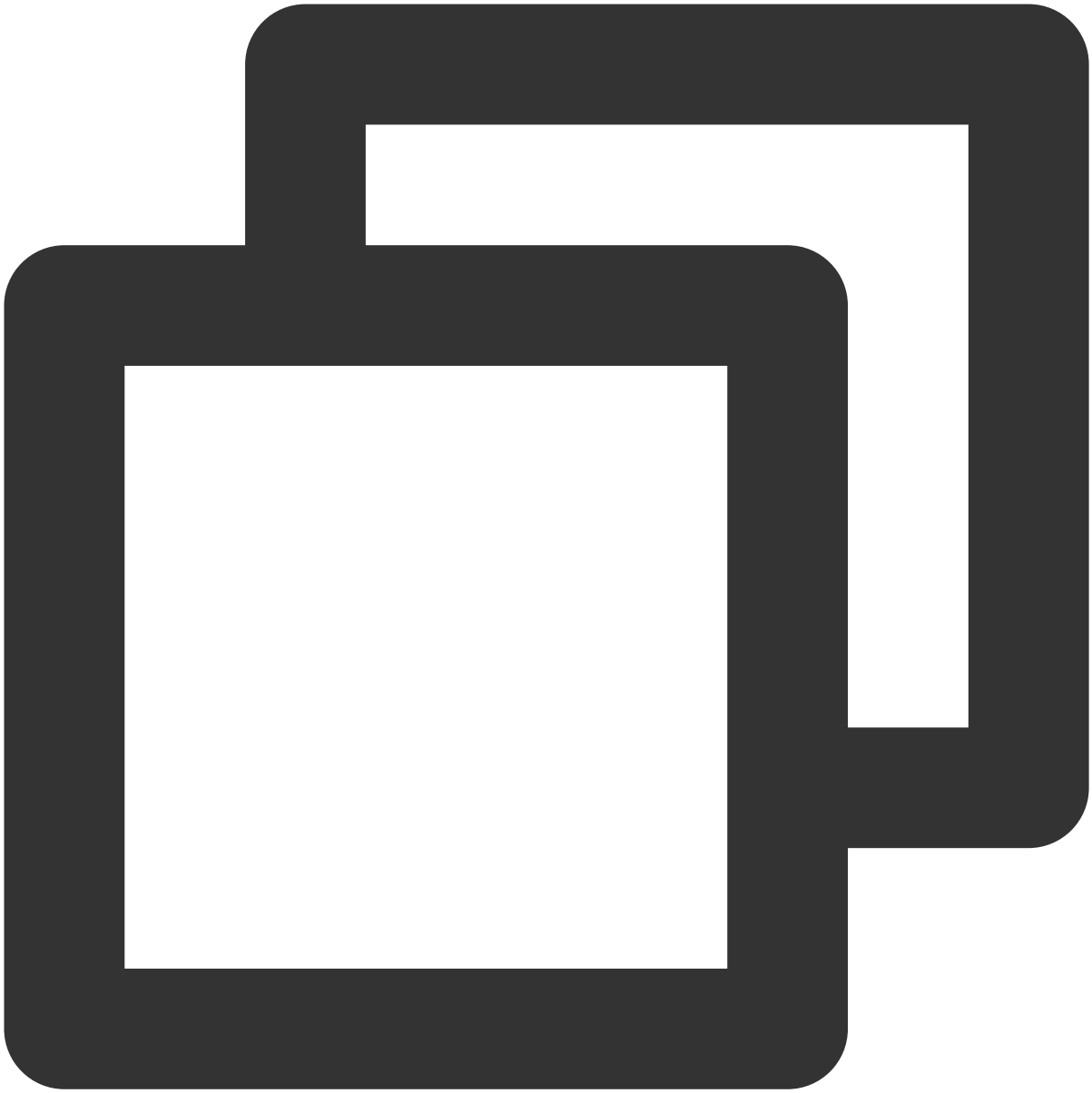
想要开启摄像头的预览画面，您需要先给 SDK 提供一个用于显示视频画面的 `TXCloudVideoView` 对象，由于 `TXCloudVideoView` 是继承自 Android 中的 `FrameLayout`，所以您可以：

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。



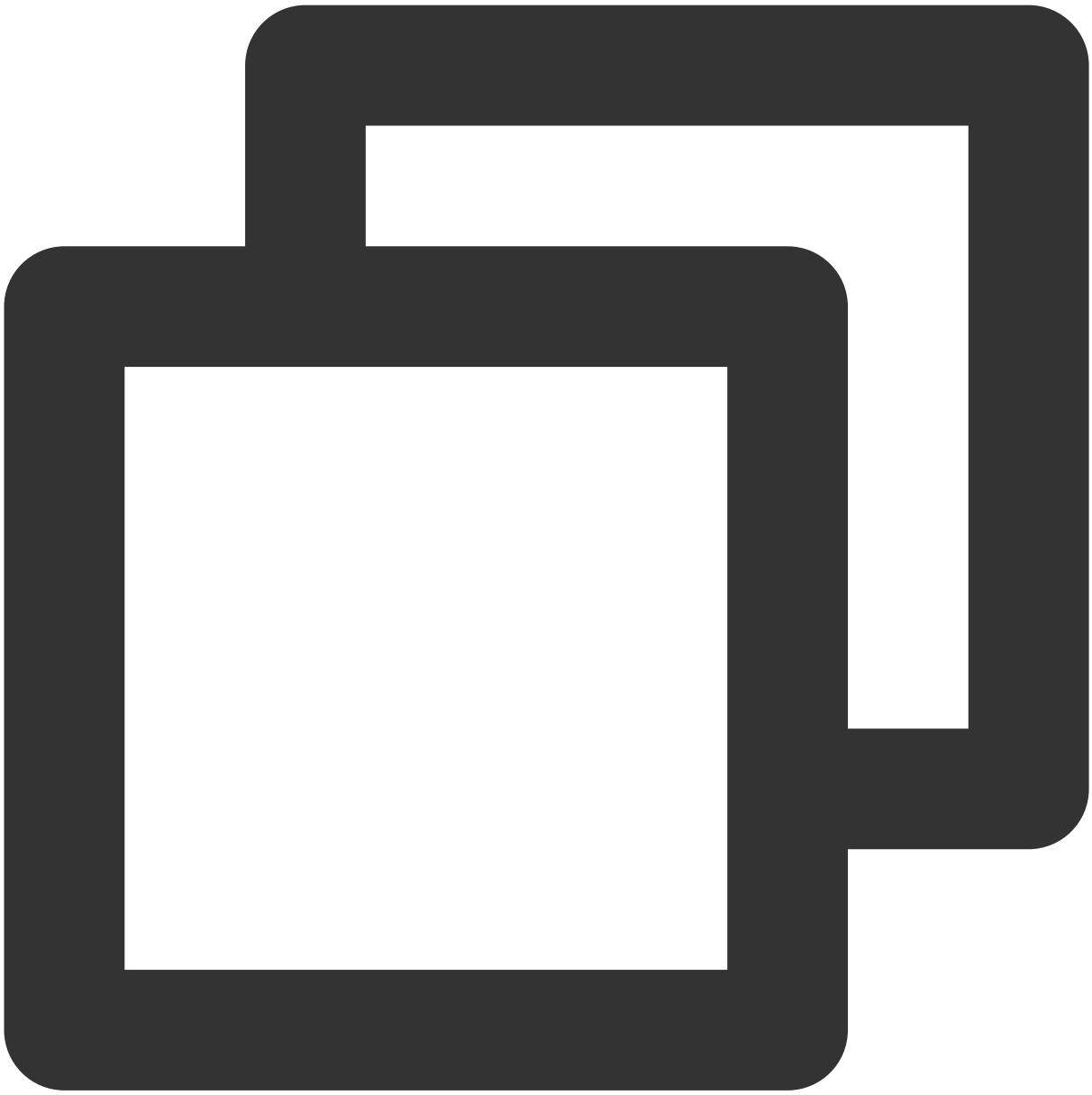
```
// 启动本地摄像头预览
TXCloudVideoView mPusherView = (TXCloudVideoView) findViewById(R.id.pusher_tx_cloud
mLivePusher.setRenderView(mPusherView);
mLivePusher.startCamera(true);
mLivePusher.startMicrophone();
```

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。



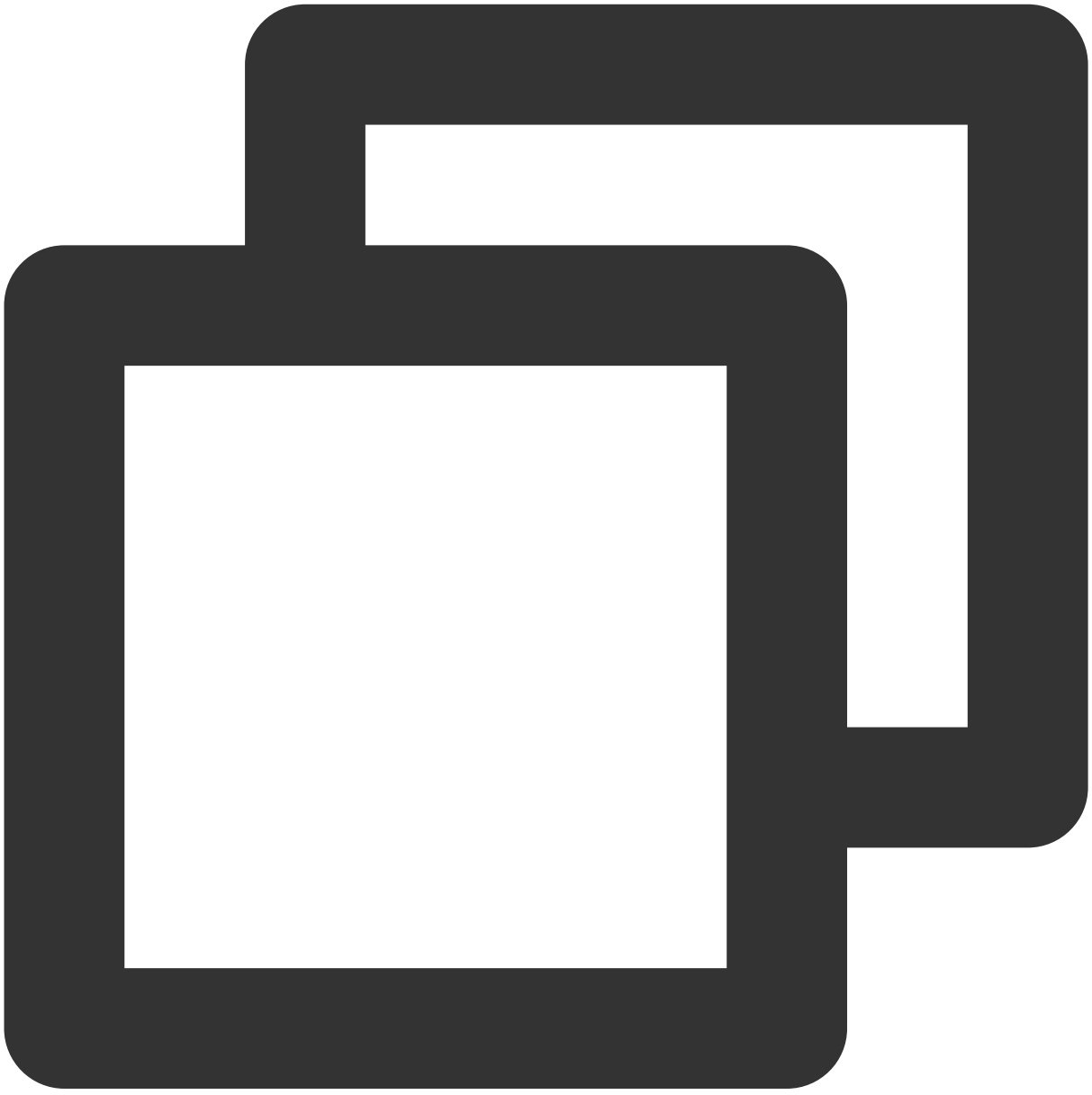
```
// 根据推流协议传入对应的 URL 即可启动推流， RTMP 协议以 rtmp:// 开头，该协议不支持连麦
String url = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
int ret = mLivePusher.startPush(url);
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
 Log.i(TAG, "startRTMPPush: license 校验失败");
}
```

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。

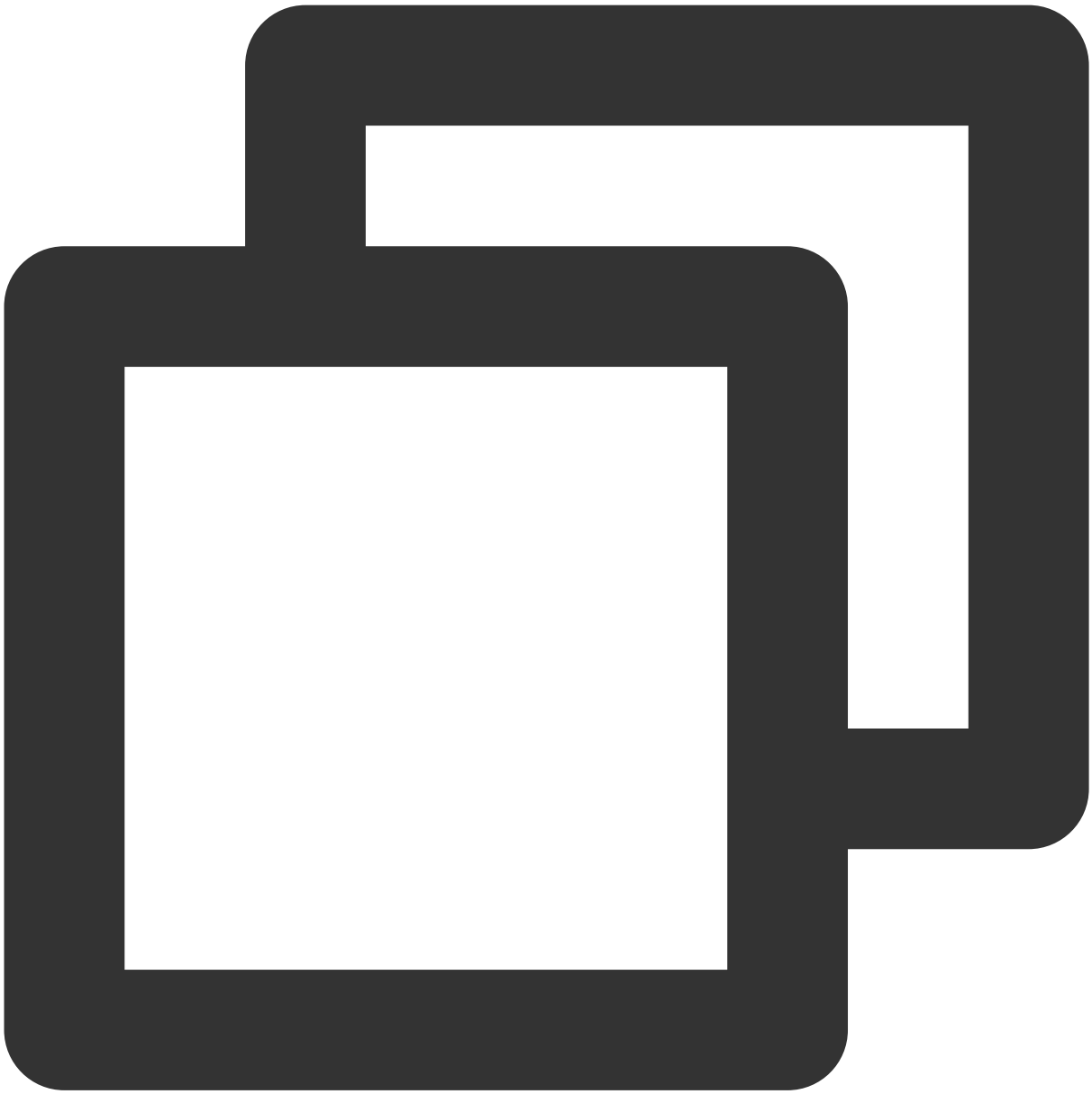
2. 推流结束后，可以调用 V2TXLivePusher 中的 stopPush 接口结束推流。



```
// 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";
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：

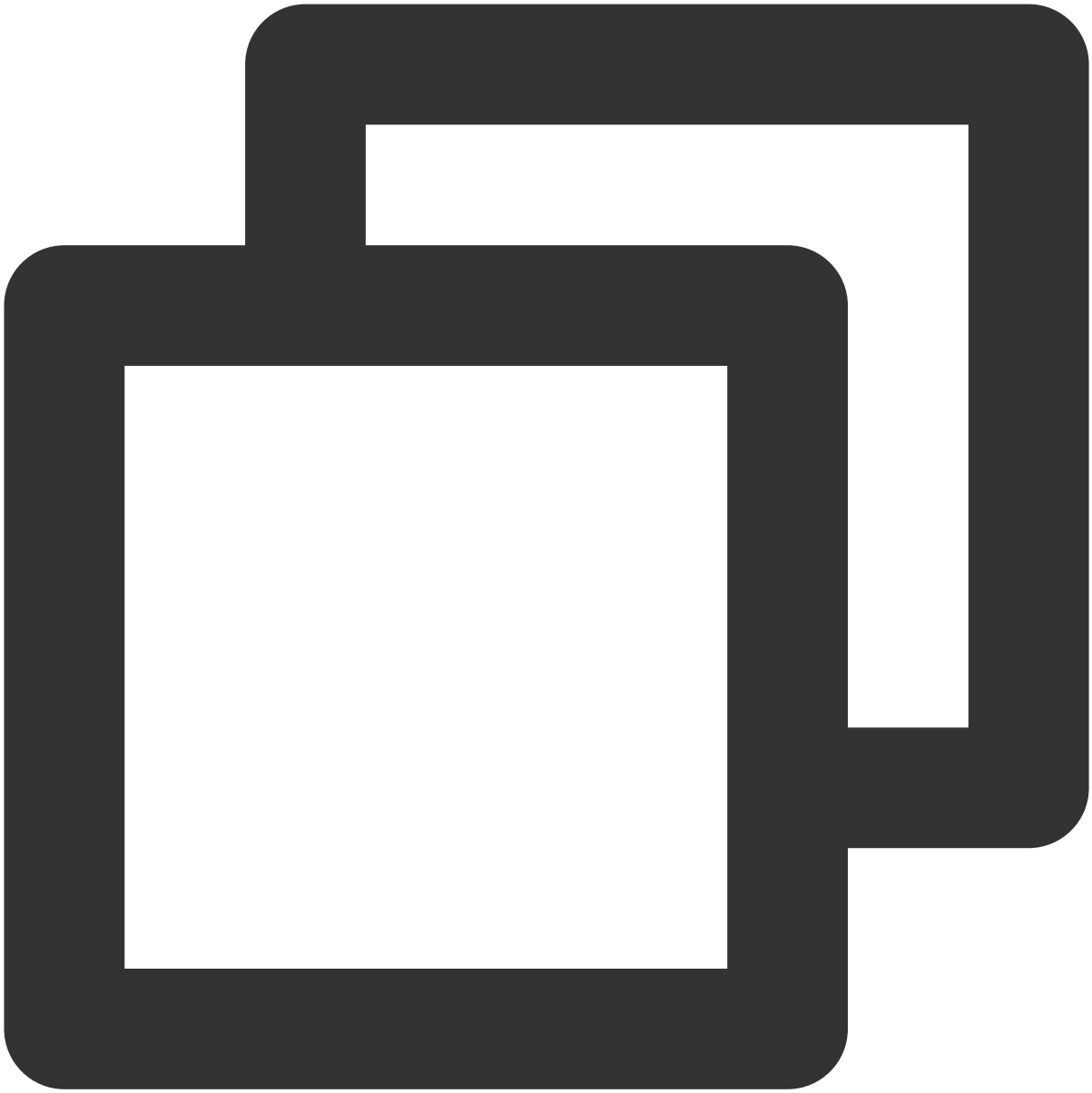
美颜风格	效果说明
BEAUTY_STYLE_SMOOTH	光滑，适用于美女秀场，效果比较明显
BEAUTY_STYLE_NATURE	自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然
BEAUTY_STYLE_PITU	由上海优图实验室提供的美颜算法，磨皮效果介于光滑和自然之间，比光滑保留更多皮肤细节，比自然磨皮程度更高

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。



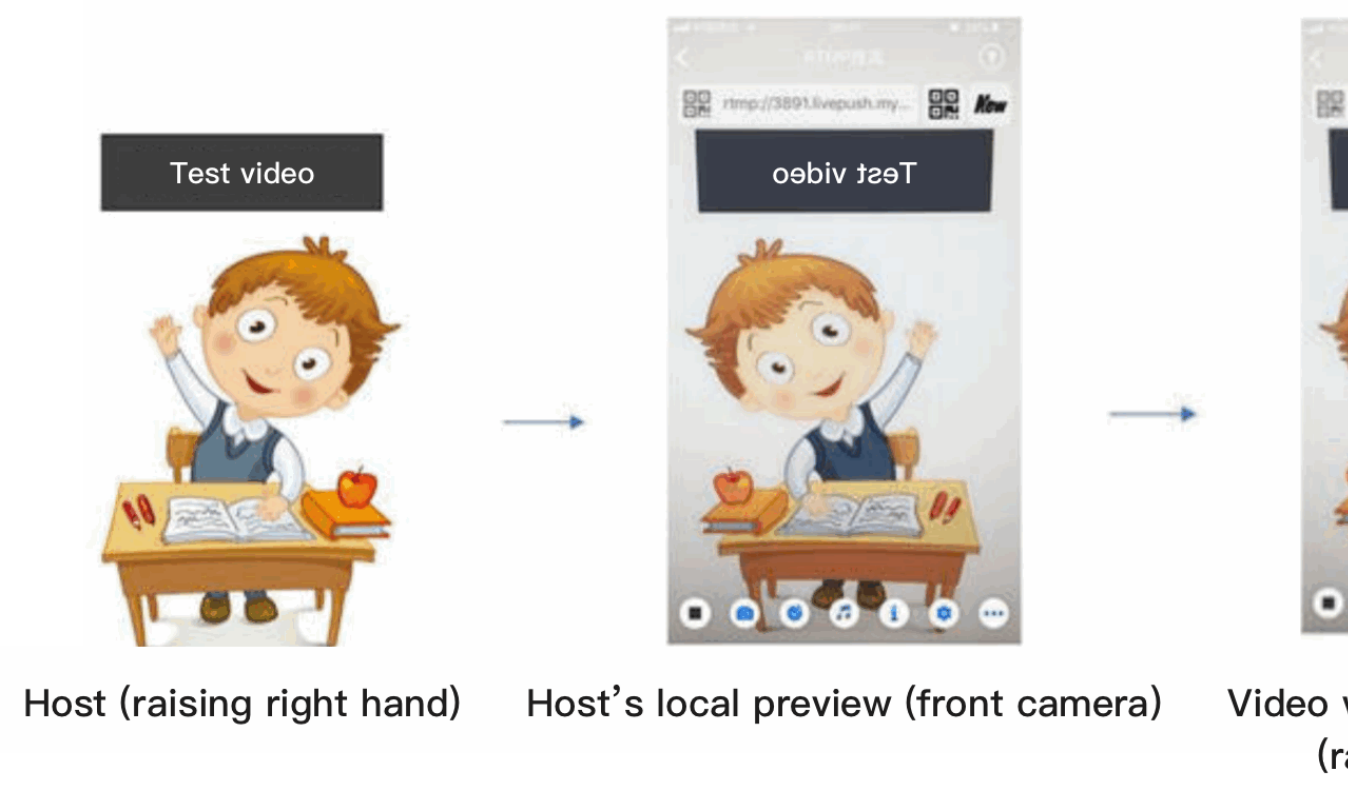
```
// 选择期望的色彩滤镜文件，滤镜文件可以在小直播 App 的资源文件中获取（以 tuibeauty_filter_ 开头
Bitmap filterBmp = decodeResource(getResources(), R.drawable.tuibeauty_filter_biaoz
mLivePusher.getBeautyManager().setFilter(filterBmp);
mLivePusher.getBeautyManager().setFilterStrength(0.5f);
```

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

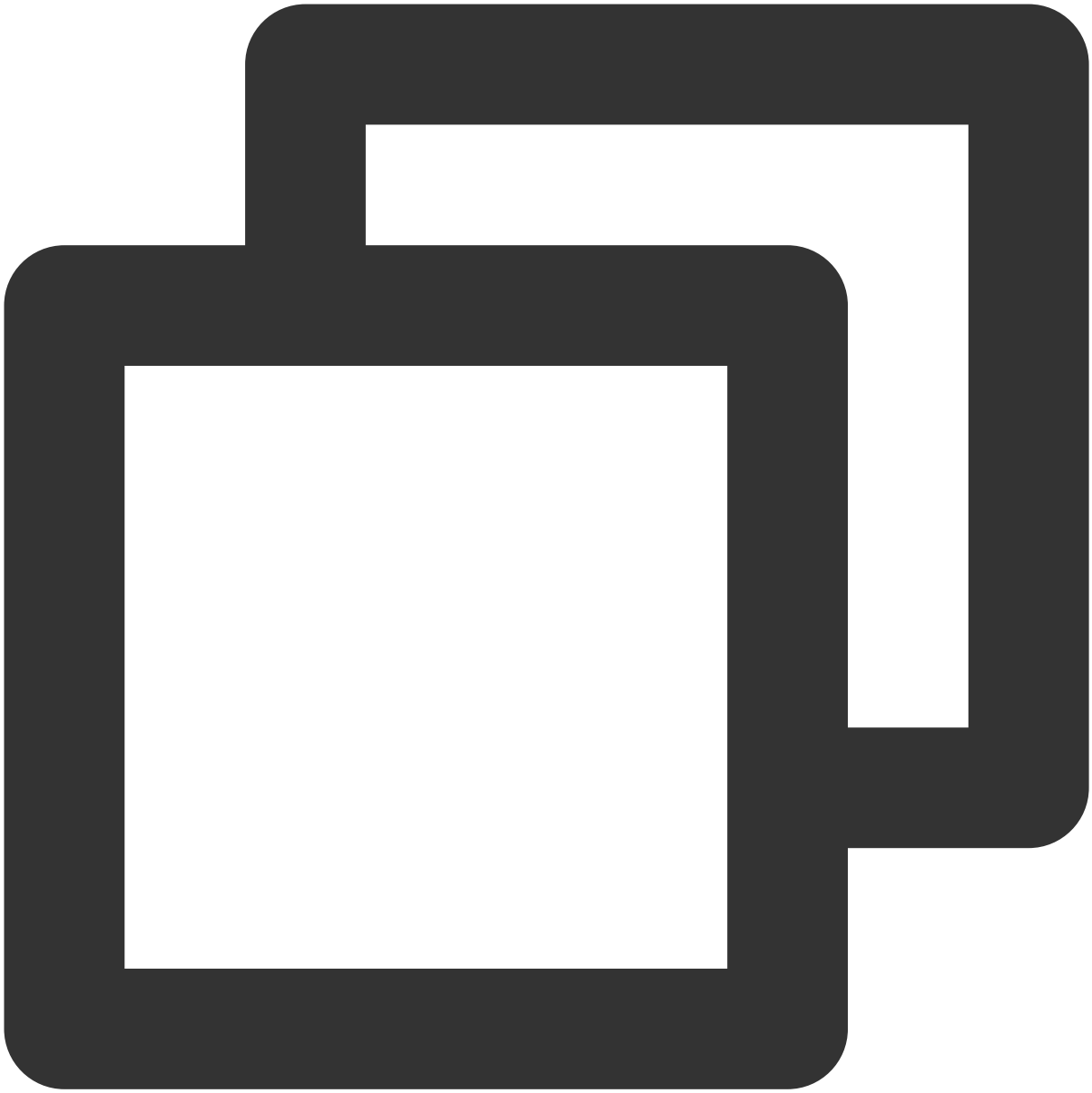
通过调用 V2TXLivePusher 的 setEncoderMirror 可以改变观众端观看到的镜像效果。之所以说是观众端的镜像效果，是因为当主播在使用前置摄像头直播时，默认情况下自己看到的画面会被 SDK 反转，这时主播就像照镜子一样，观众看到的效果和主播看到的是一致的。如下图所示：



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）。

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。

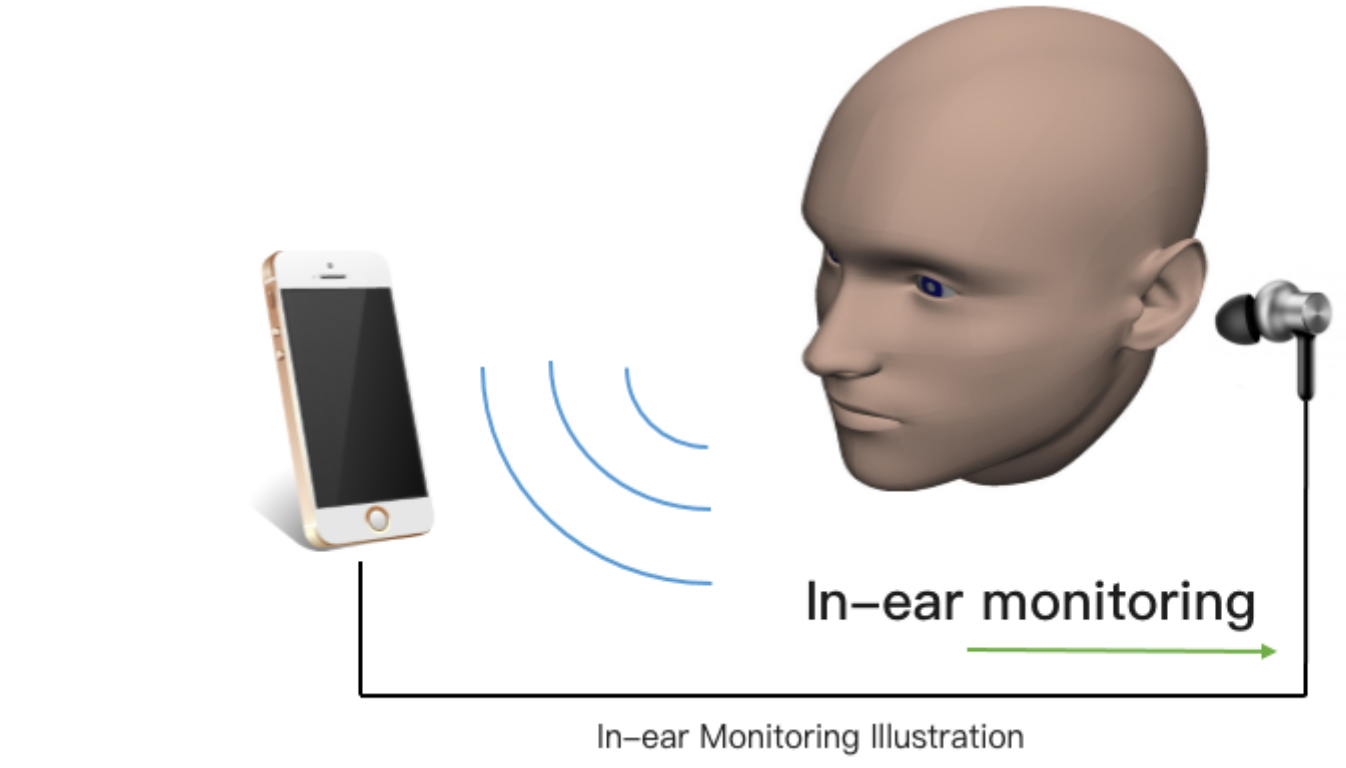


```
V2TXLiveDef.V2TXLiveVideoEncoderParam param = new V2TXLiveDef.V2TXLiveVideoEncoderP
param.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2
mLivePusher.setVideoQuality(param);
```

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。

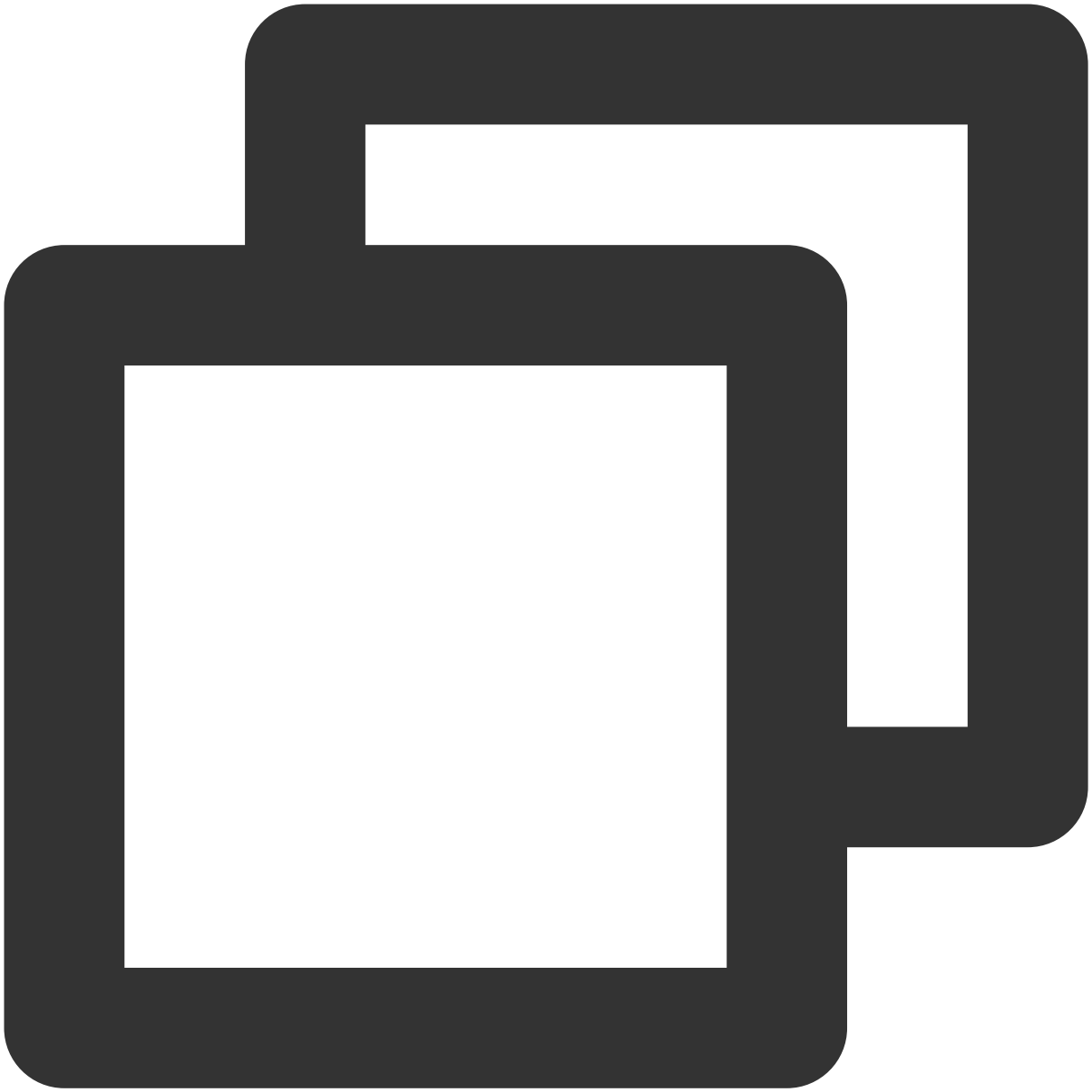


In-ear Monitoring Illustration

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）。

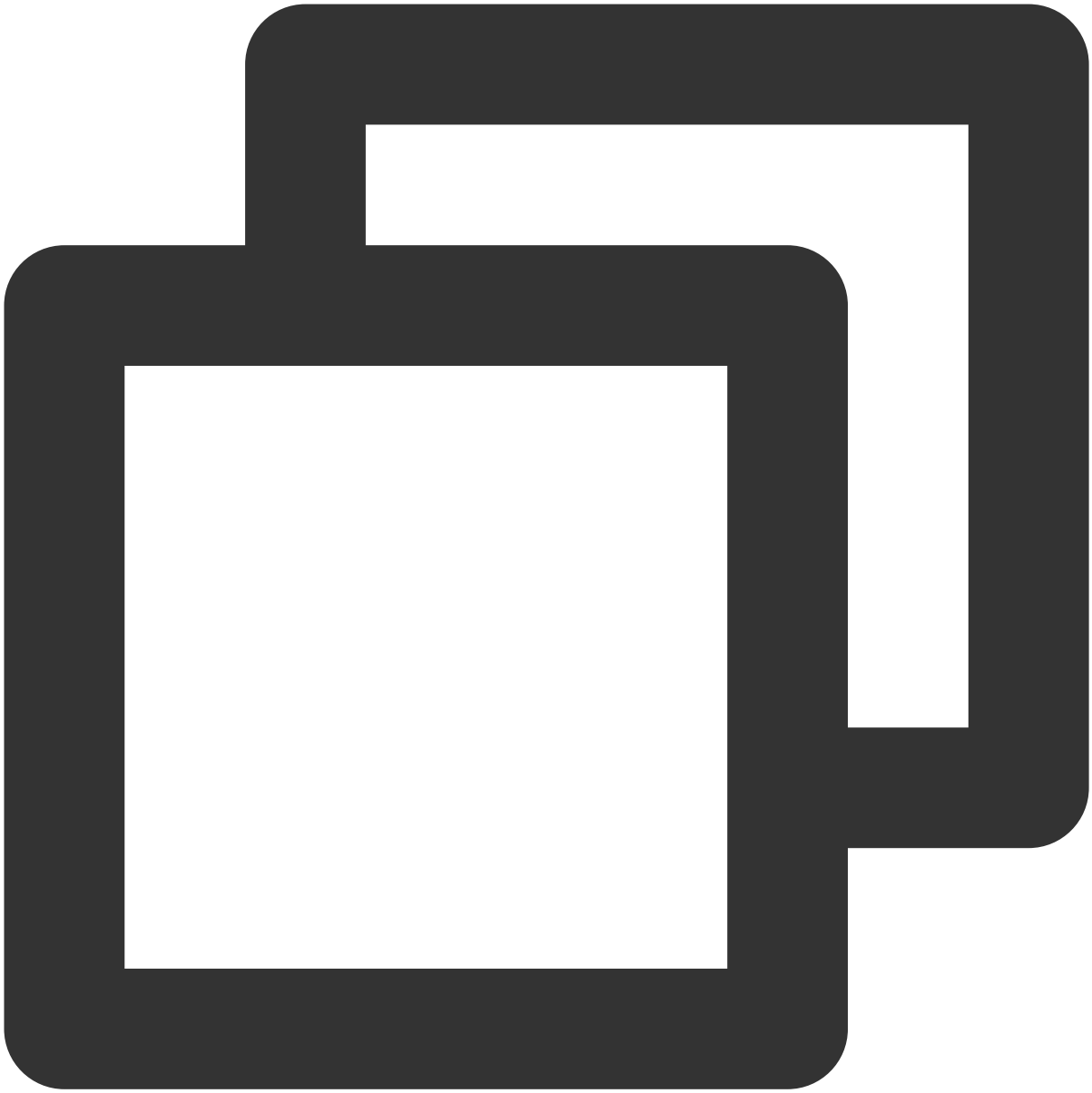


```
// 设置视频水印
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.wat
```

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

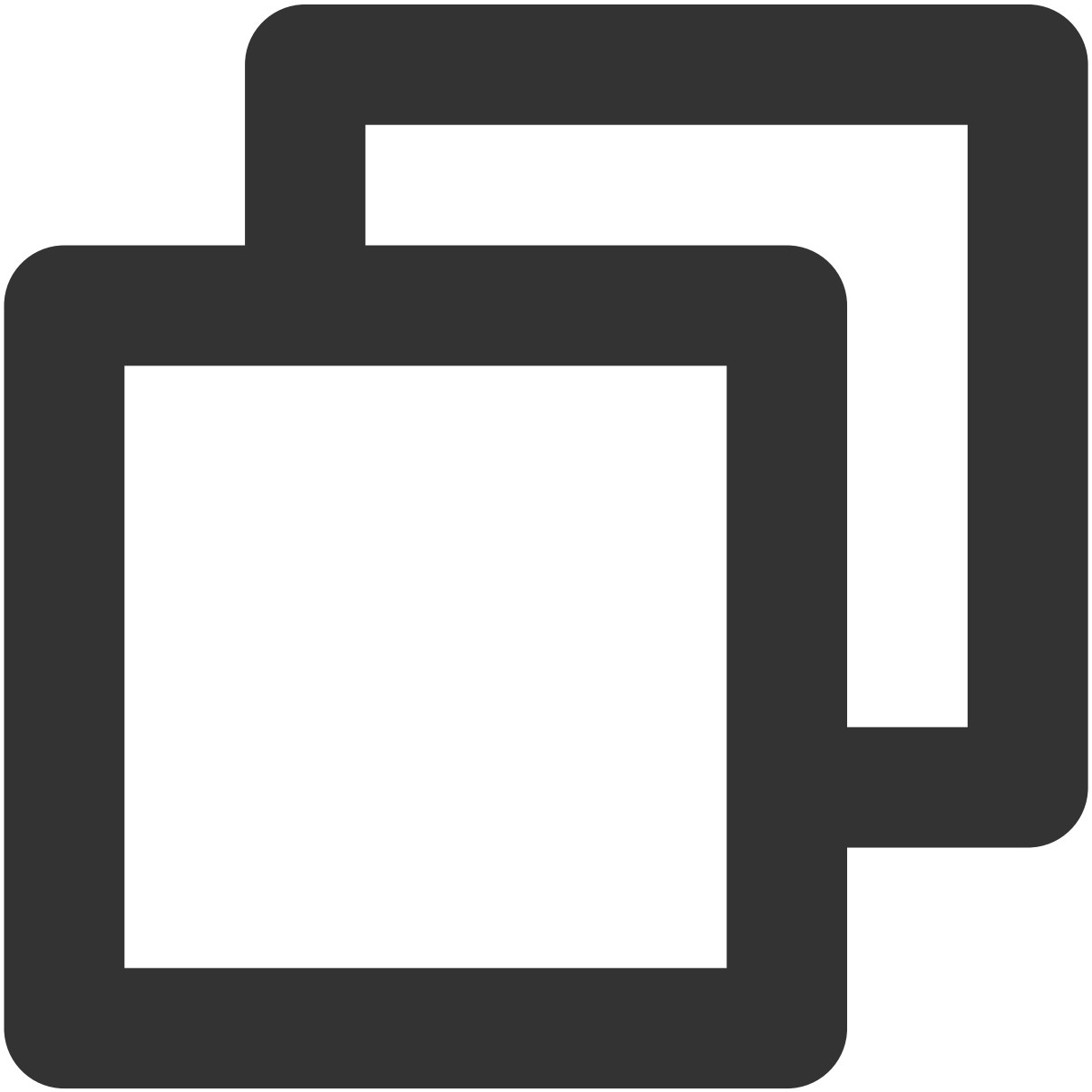
调用 V2TXLivePusher 中的 sendSeiMessage 接口可以发送 SEI 消息。所谓 SEI，是视频编码数据中规定的一种附加增强信息，平时一般不被使用，但我们可以在其中加入一些自定义消息，这些消息会被直播 CDN 转发到观众端。使用场景有：

答题直播：推流端将题目下发到观众端，可以做到“音-画-题”完美同步。

秀场直播：推流端将歌词下发到观众端，可以在播放端实时绘制出歌词特效，因而不受视频编码的降质影响。

在线教育：推流端将激光笔和涂鸦操作下发到观众端，可以在播放端实时地划圈划线。

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。

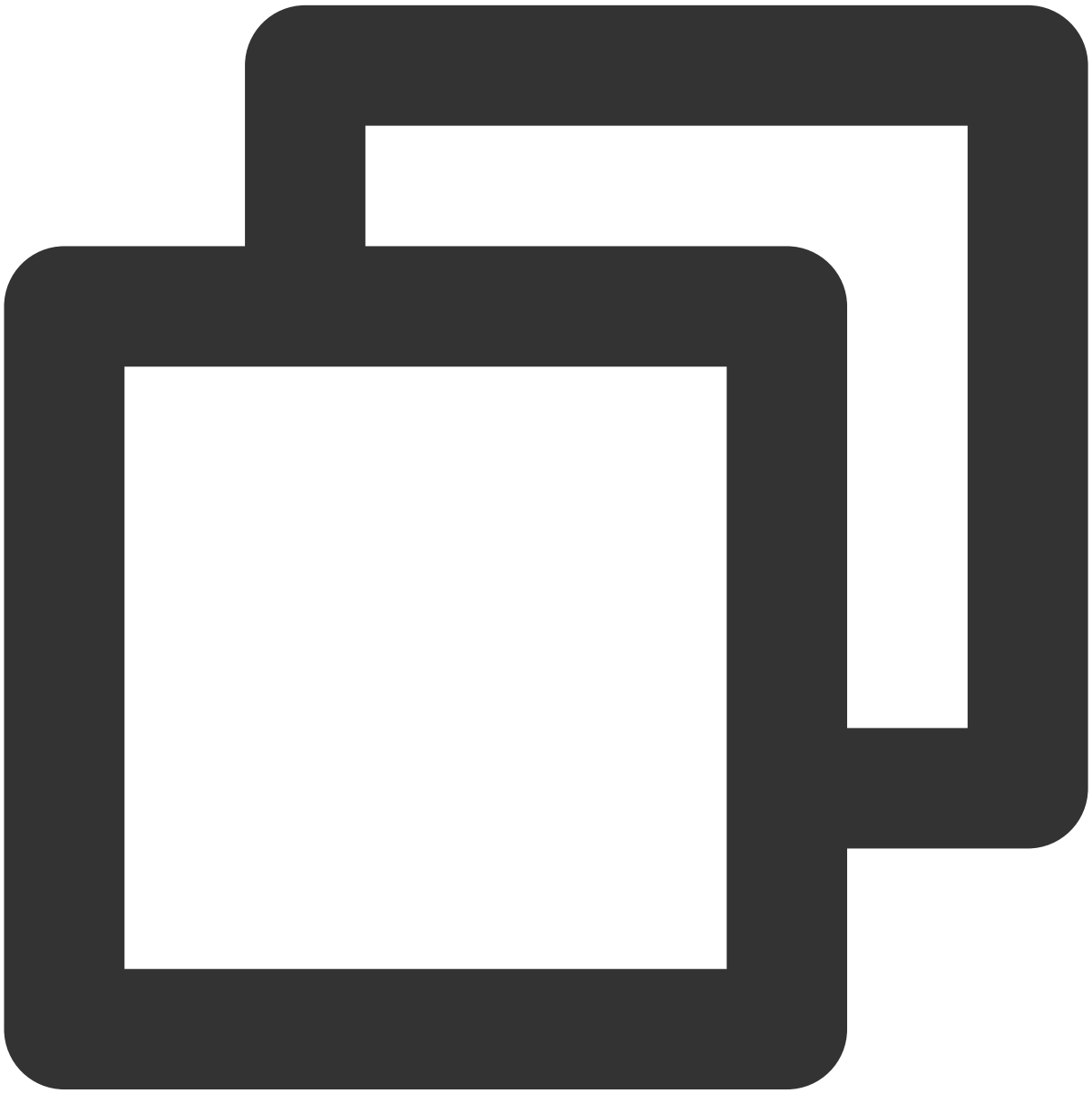


```
//Android 示例代码
```

```
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_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

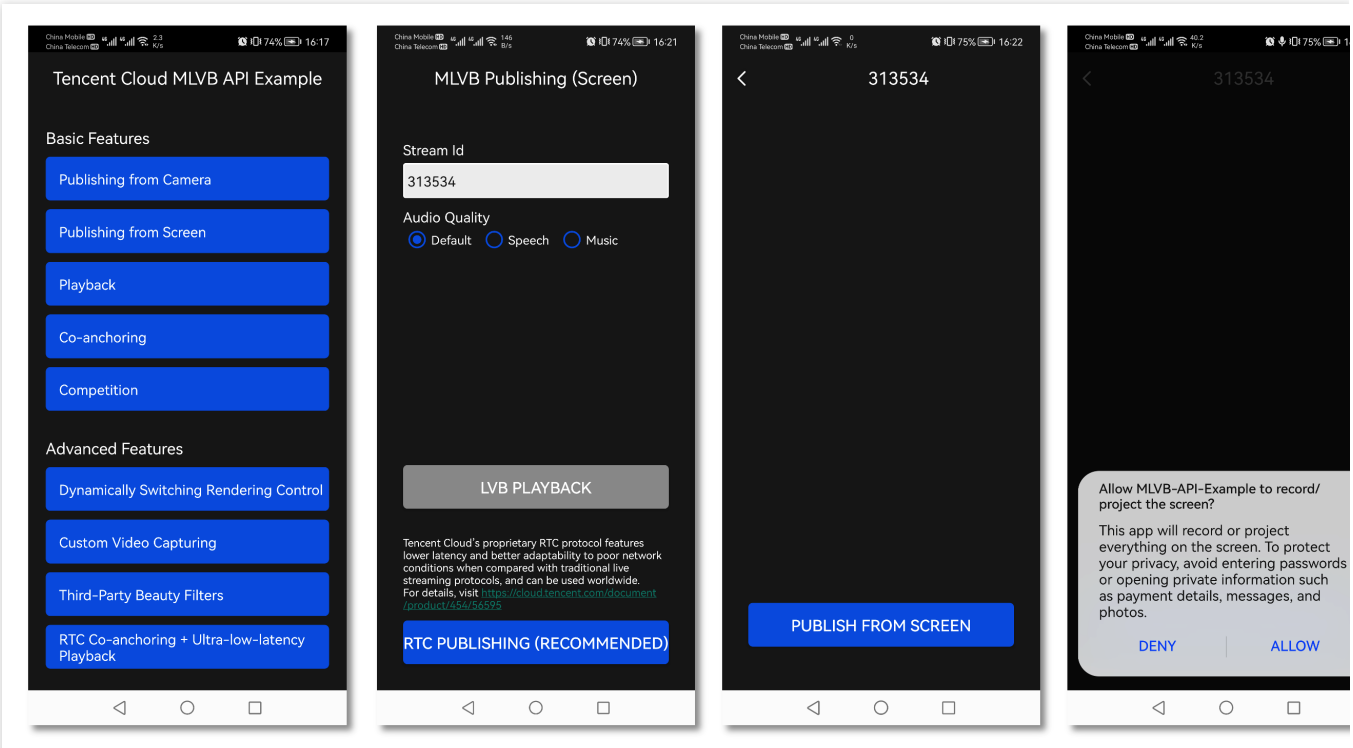
最近更新时间：2024-01-13 15:53:49

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

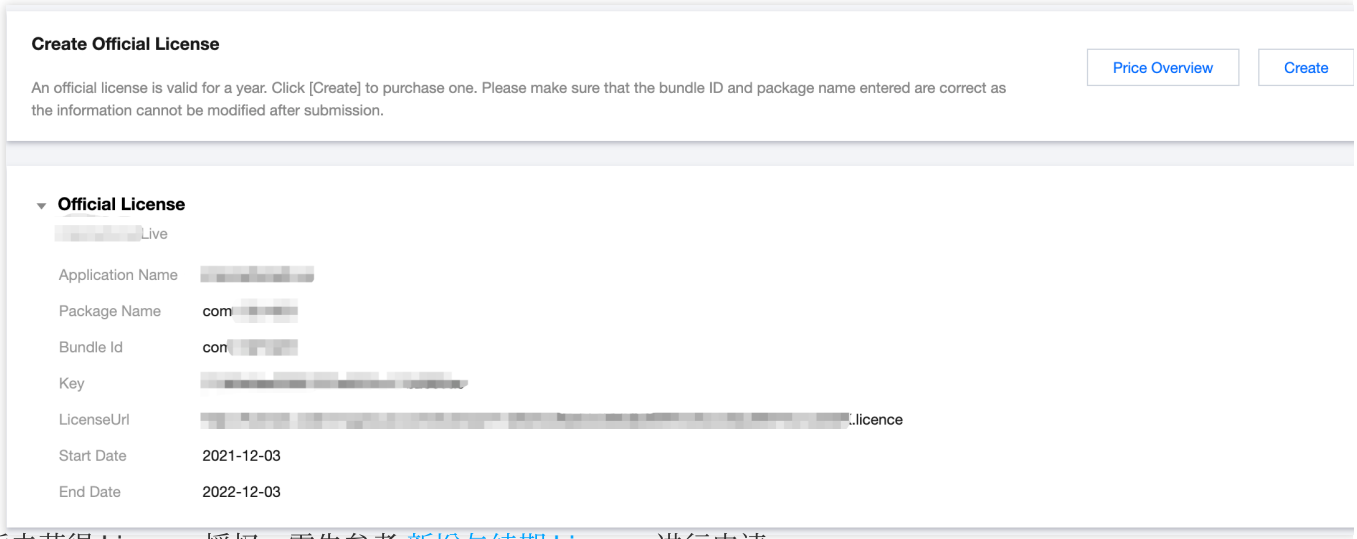
对接攻略

1. 下载 SDK 开发包

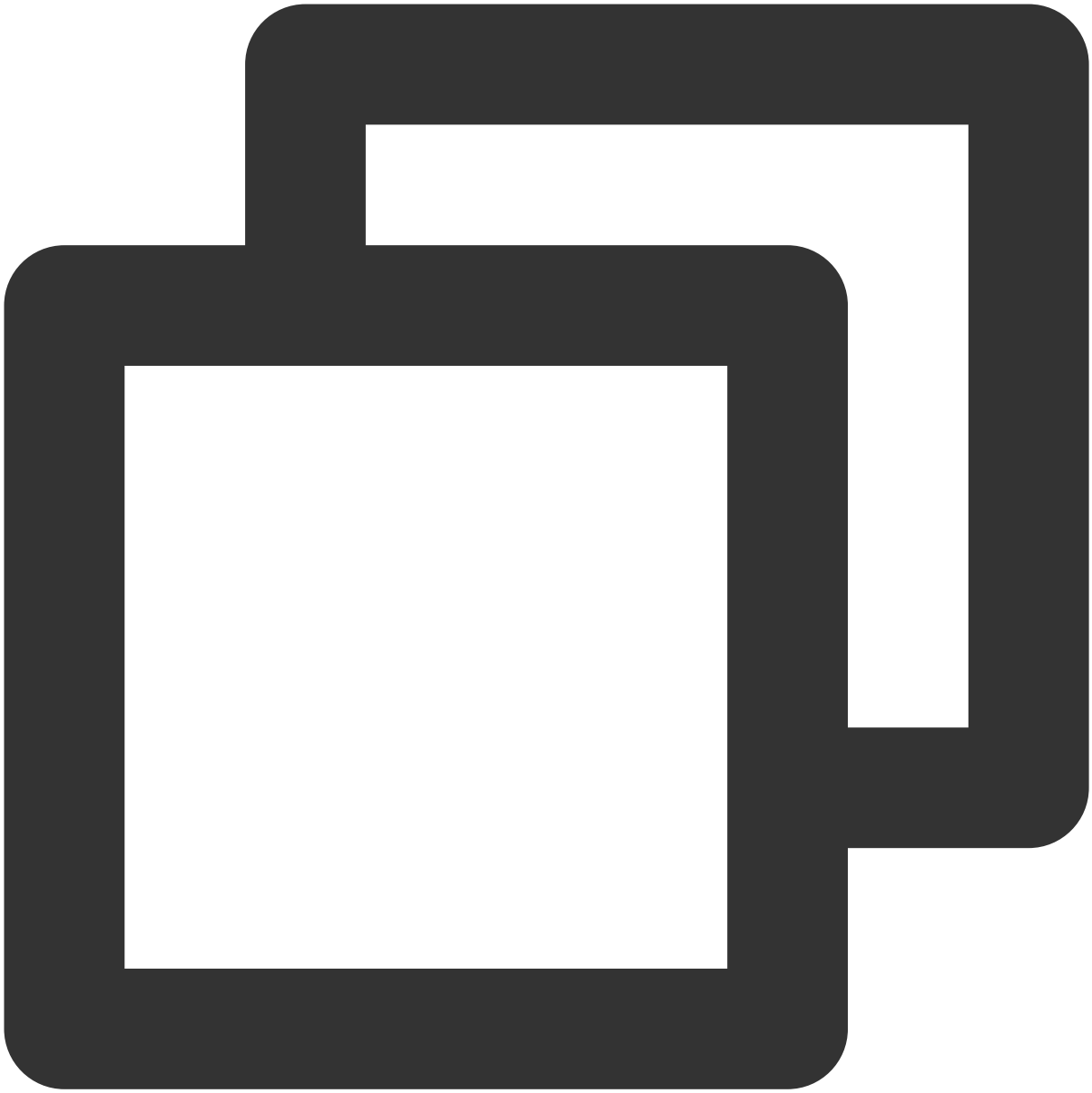
下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

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.setEnvironment("GDPR");  // 设置环境
V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
        @Override
```

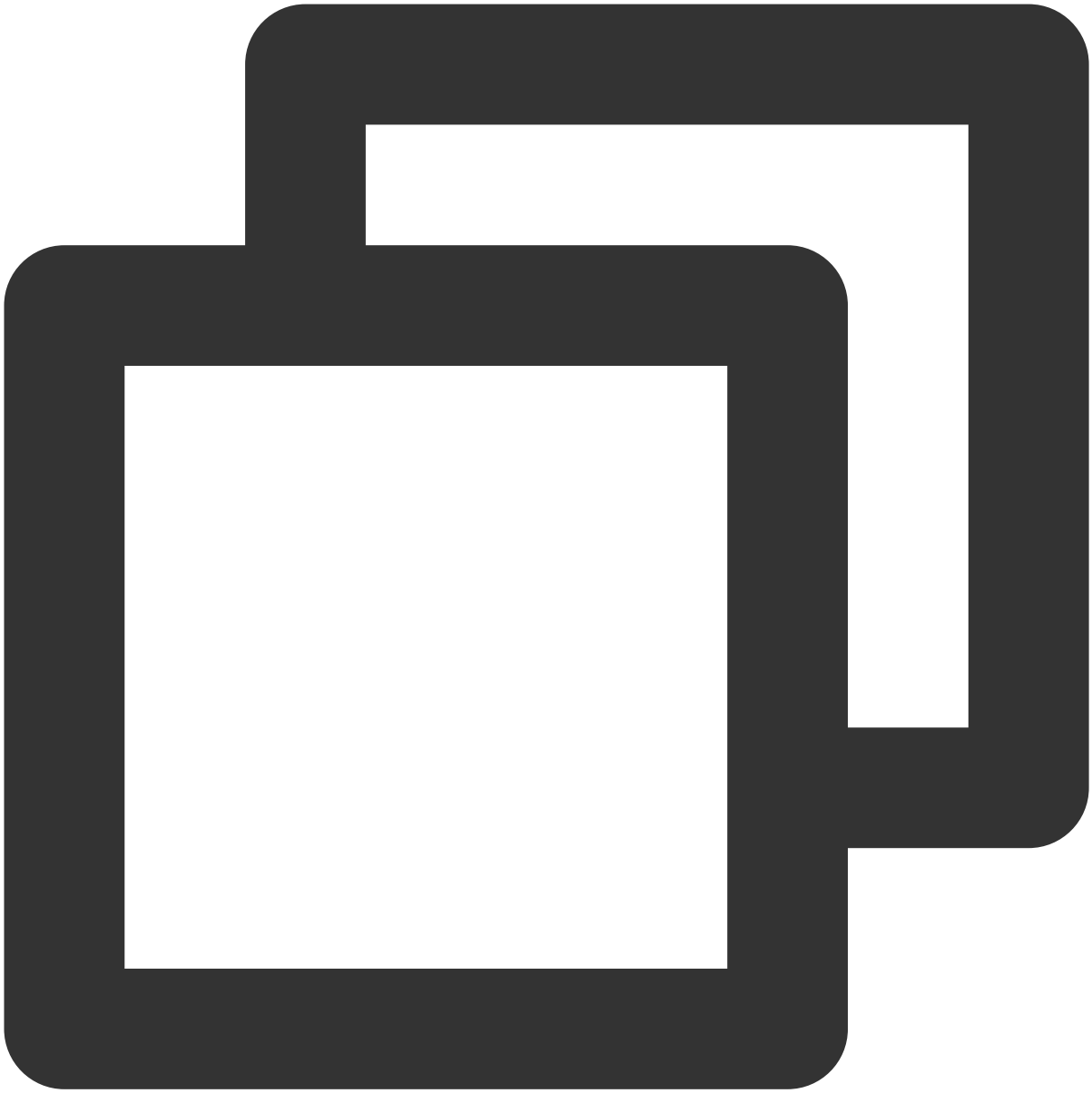
```
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
            }
        });
    }
```

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

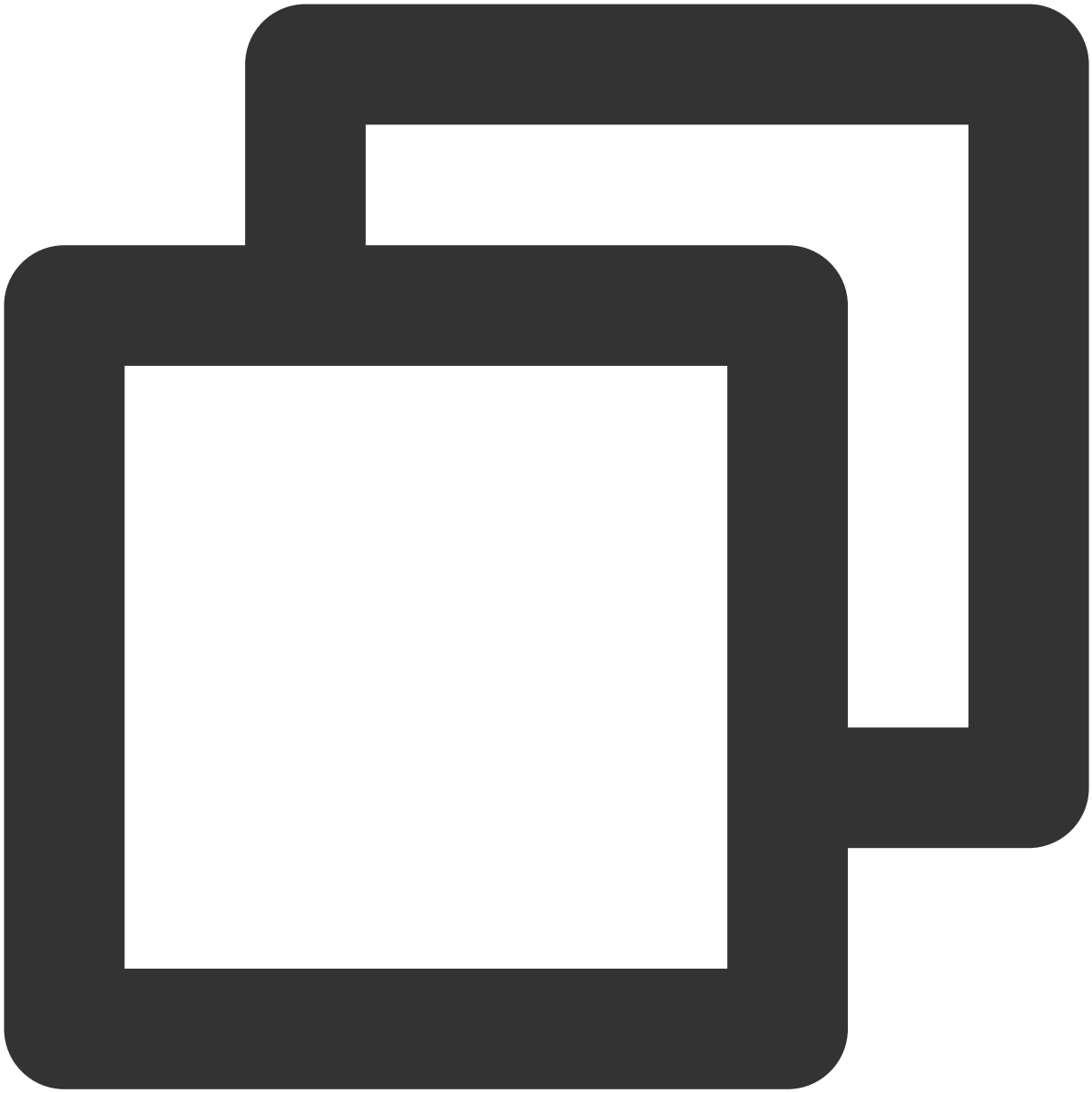
在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。



```
<activity
    android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantAc
    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。



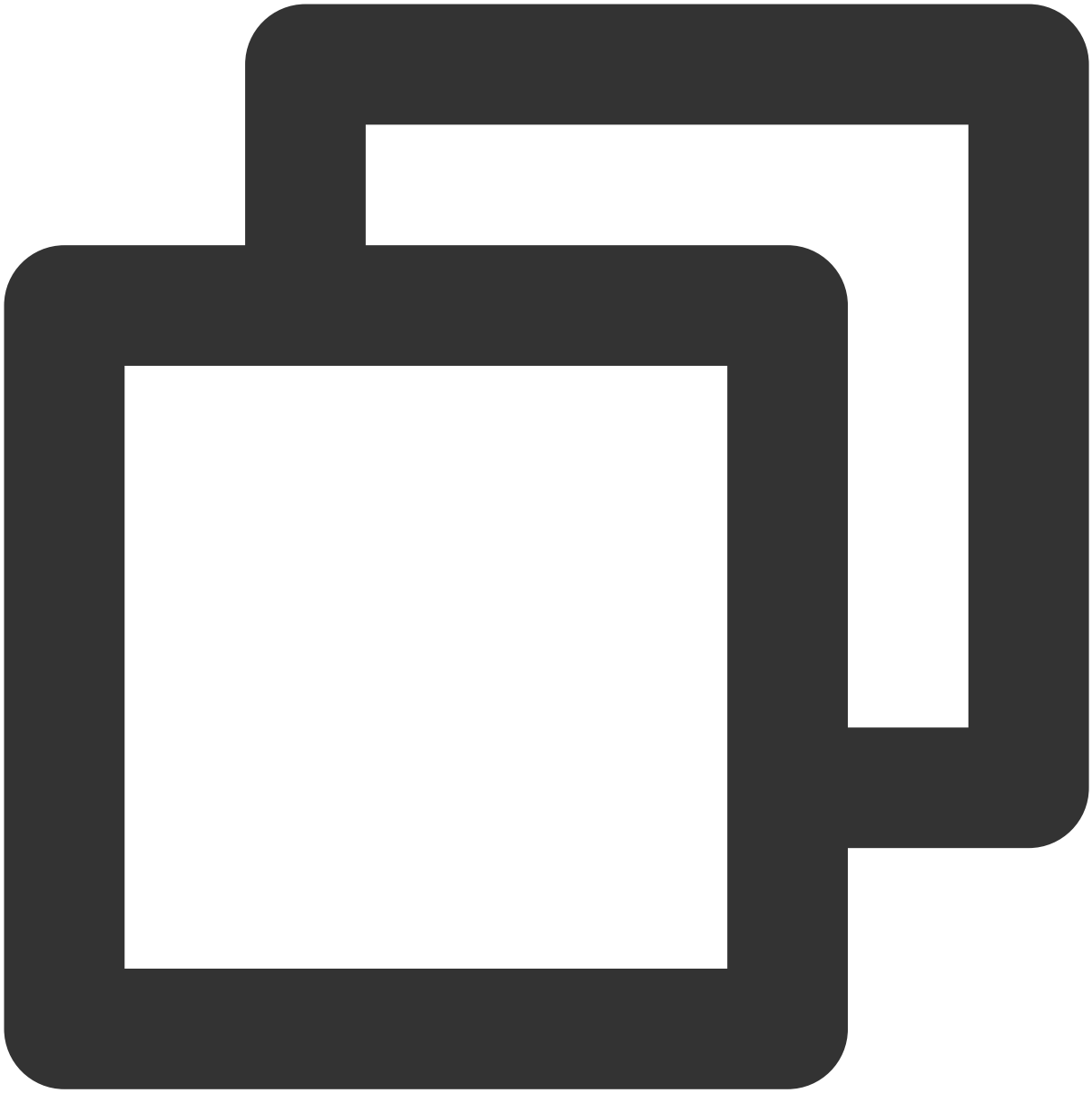
```
// 指定对应的直播协议为 RTMP，该协议不支持连麦
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.
```

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。



```
// 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";
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。

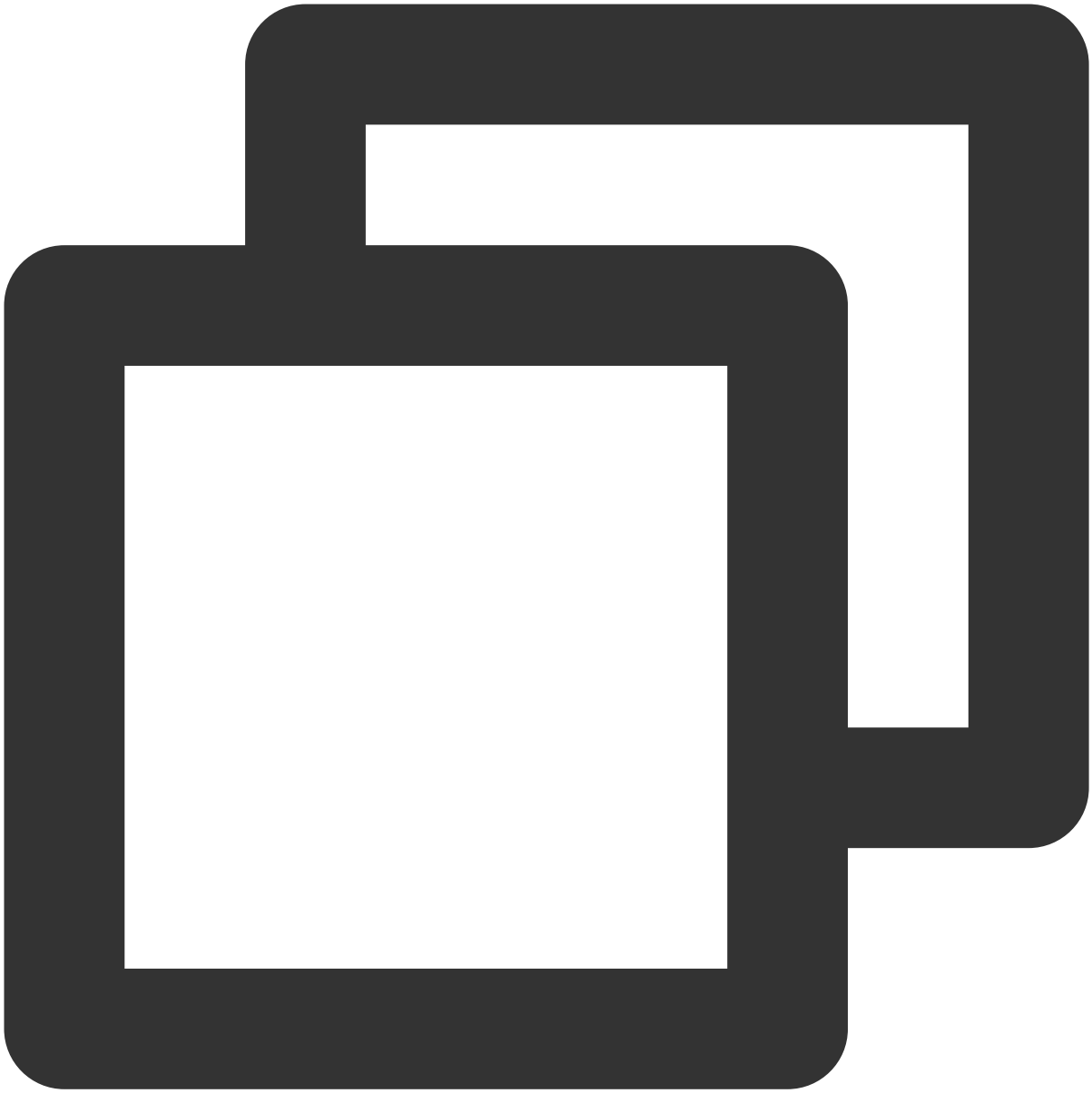
startScreenCapture 的作用是启动屏幕录制，由于录屏是基于 Android 系统的原生能力实现的，处于安全考虑，Android 系统会在开始录屏前弹出提示，允许即可。

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，水印位置位是由传入参数 `(x, y, scale)` 所决定。

SDK 所要求的水印图片格式为 PNG 而不是 JPG，因为 PNG 这种图片格式有透明度信息，因而能够更好地处理锯齿等问题（将 JPG 图片修改后缀名是不起作用的）。

`(x, y, scale)` 参数设置的是水印图片相对于推流分辨率的归一化坐标。假设推流分辨率为：540 × 960，该字段设置为：`（0.1, 0.1, 0.1）`，那么水印的实际像素坐标为：（540 × 0.1，960 × 0.1，水印宽度 × 0.1，水印高度会被自动计算）。



```
//设置视频水印
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(),R.drawable.wat
```

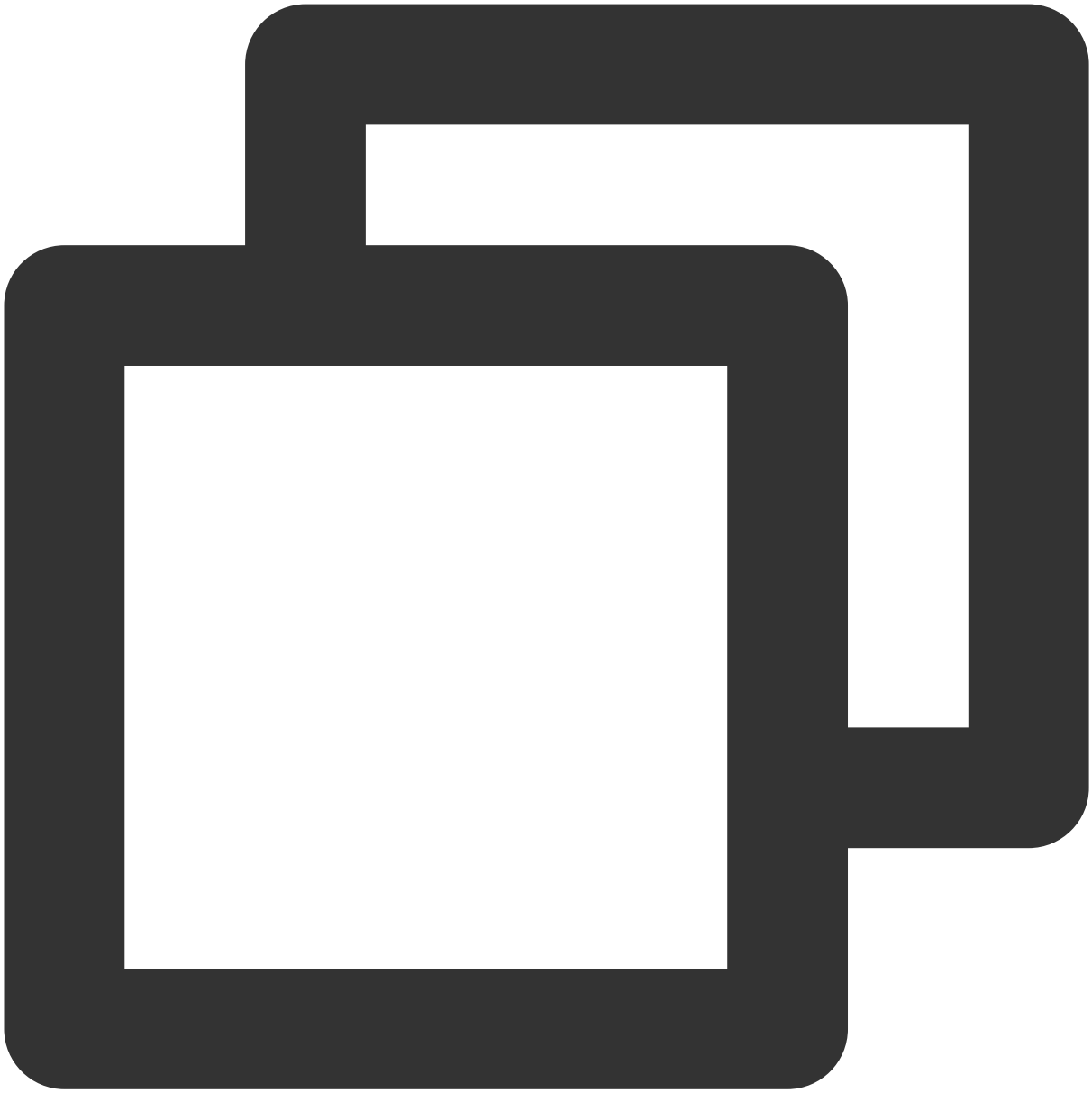
7. 推荐的清晰度

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”。

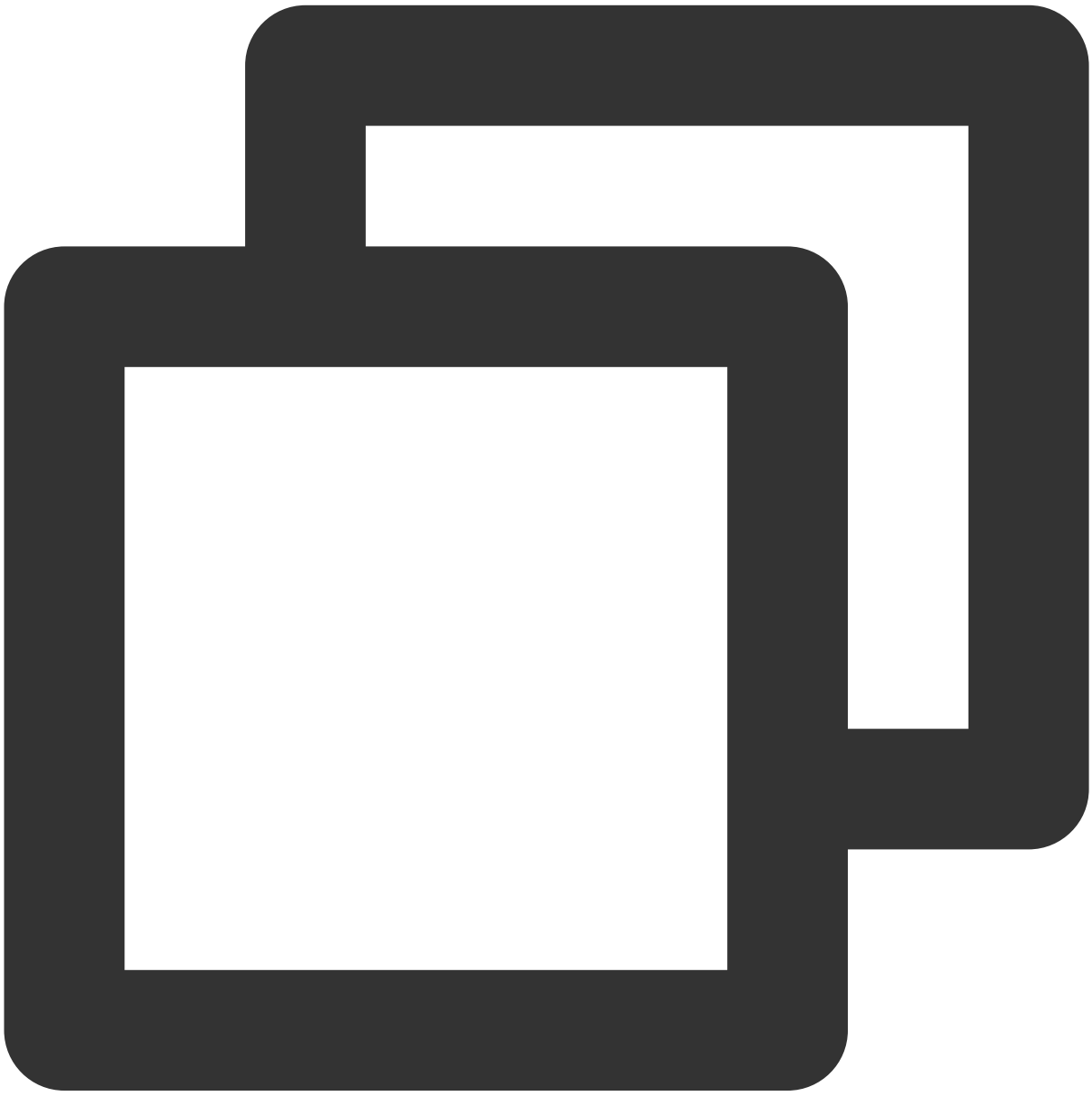


```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
    if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
        showNetBusyTips(); // 显示网络繁忙的提示
    }
}
```

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）。

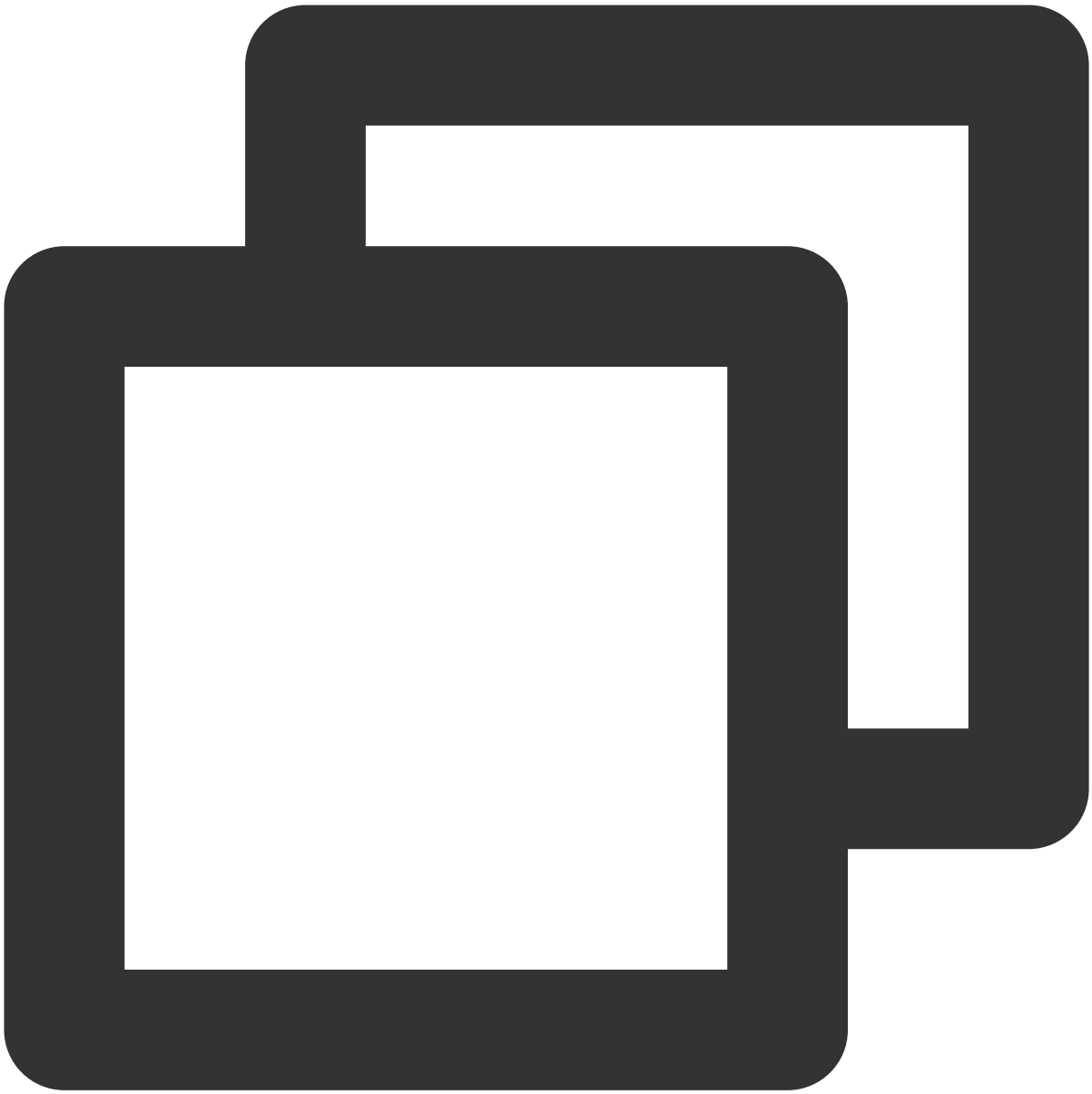
V2TXLivePusher 默认推出的是竖屏分辨率的视频画面，如果希望推出横屏分辨率的画面，可以修改 setVideoQuality 接口的参数来设定观众端的画面横竖屏模式。



```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResolution
```

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_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

Web 推流

最近更新时间：2024-01-13 15:53:49

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

注意

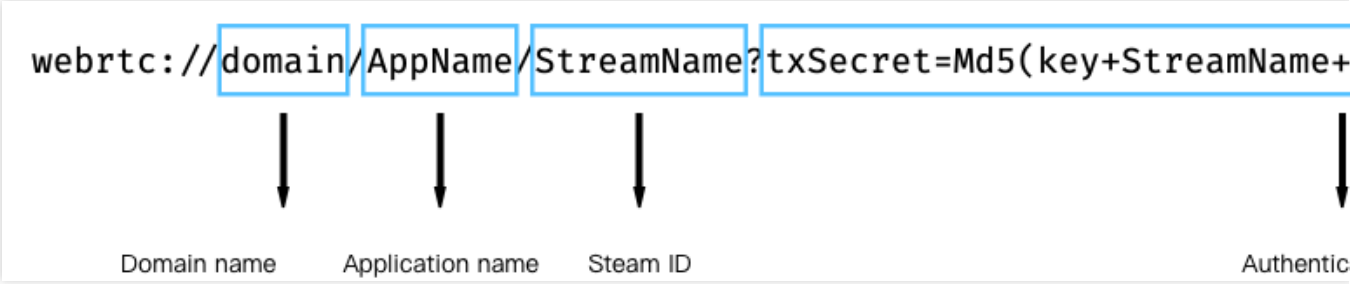
使用 WebRTC 协议推流，每个推流域名默认限制**100路并发**推流数，如您需要超过此推流限制，可通过 提交工单 的方式联系我们进行申请。

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

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

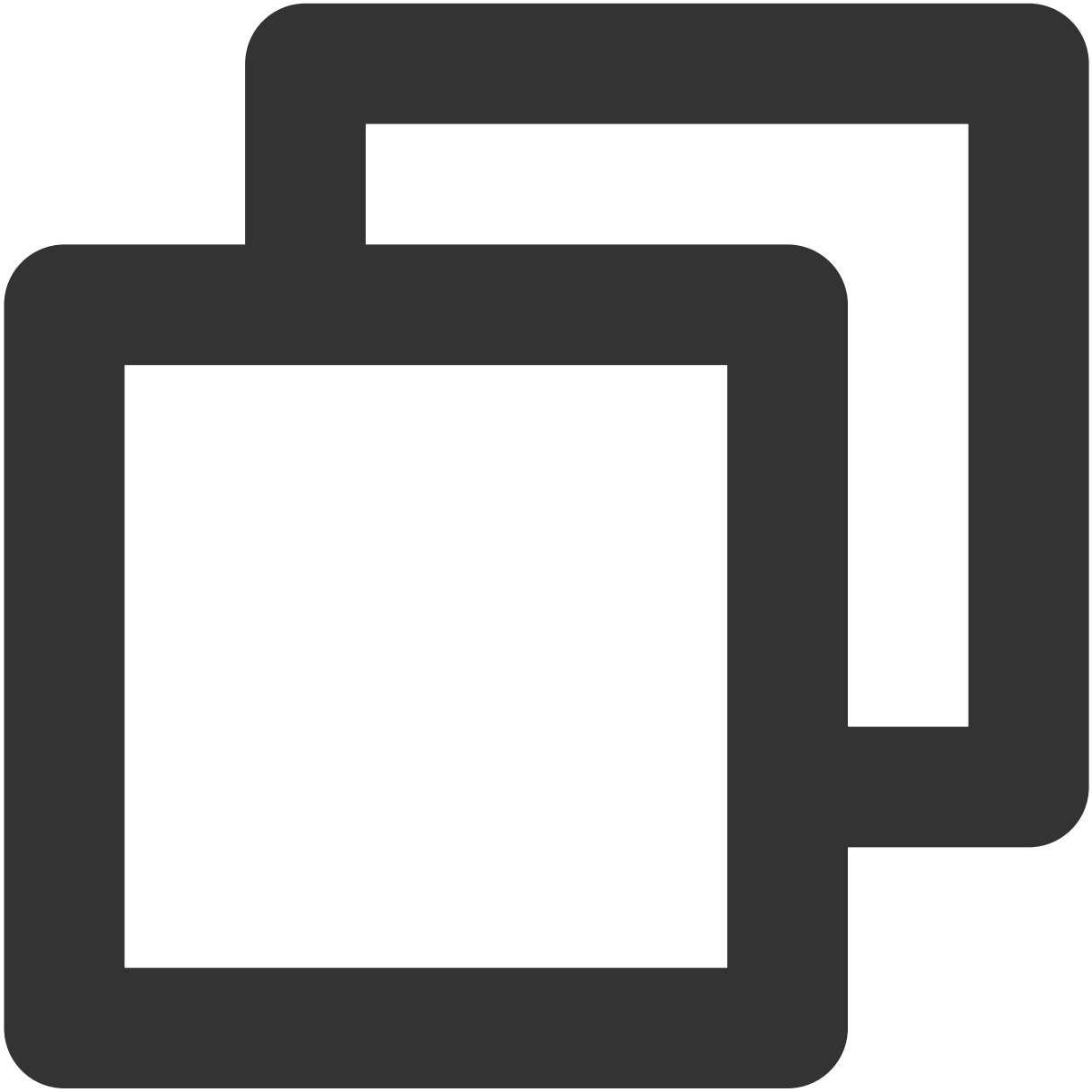
除此以外，浏览器采集音视频画面的功能在移动端支持较差，例如移动端浏览器不支持屏幕录制，iOS 14.3及以上版本才支持获取用户摄像头设备。因此推流 SDK 主要适用于桌面端浏览器，目前最新版本的 chrome、Firefox 和 Safari 浏览器都是支持快直播推流的。

移动端建议使用 直播 SDK 进行推流。

对接攻略

步骤1：页面准备工作

在需要直播推流的页面（桌面端）中引入初始化脚本。



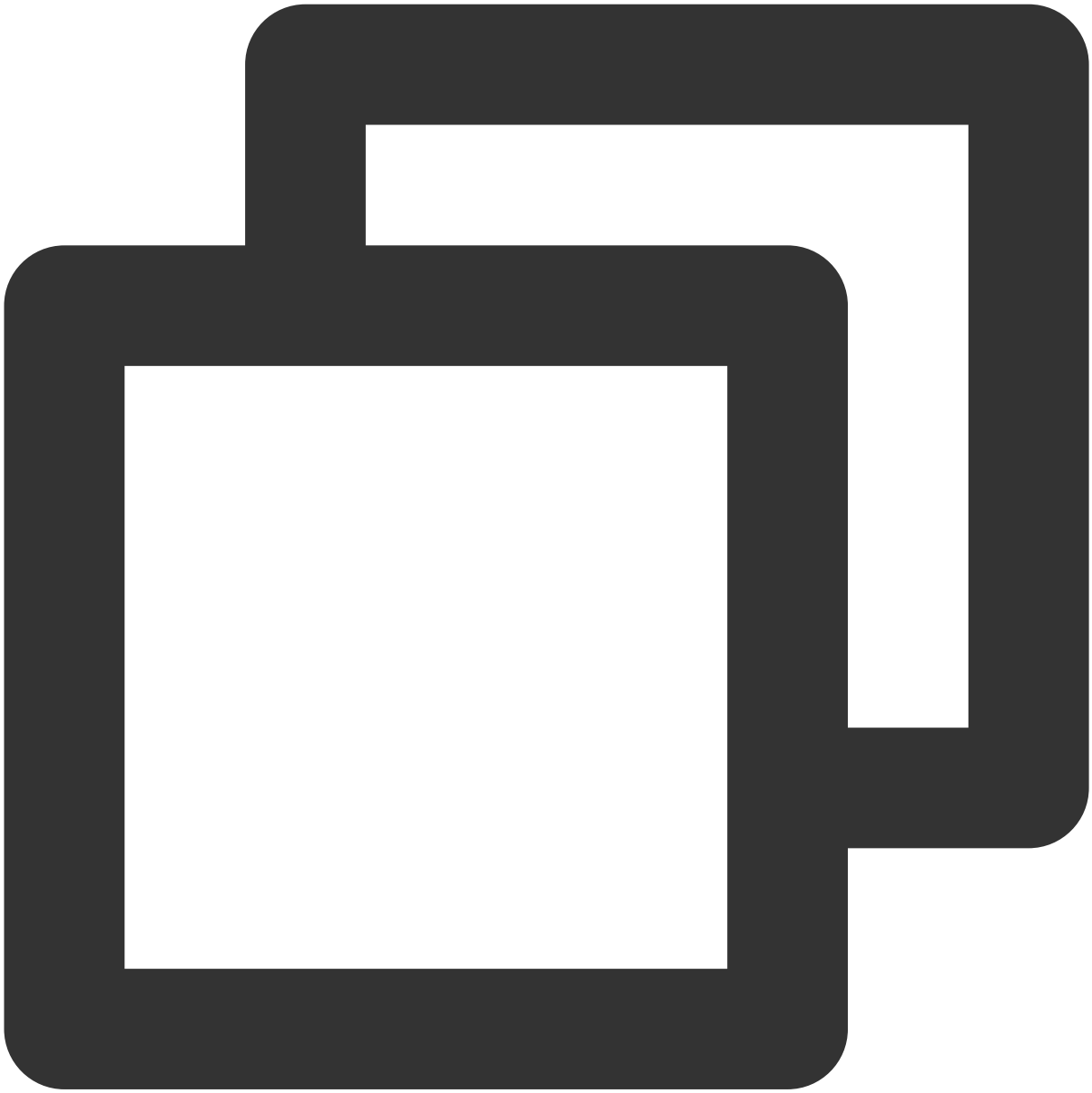
```
<script src="https://video.sdk.qcloudecdn.com/web/TXLivePusher-2.1.0.min.js" charse
```

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 id_local_video，本地视频画面都会在容器里渲染。对于容器的大小控制，您可以使用 div 的 css 样式进行控制，示例代码如下：

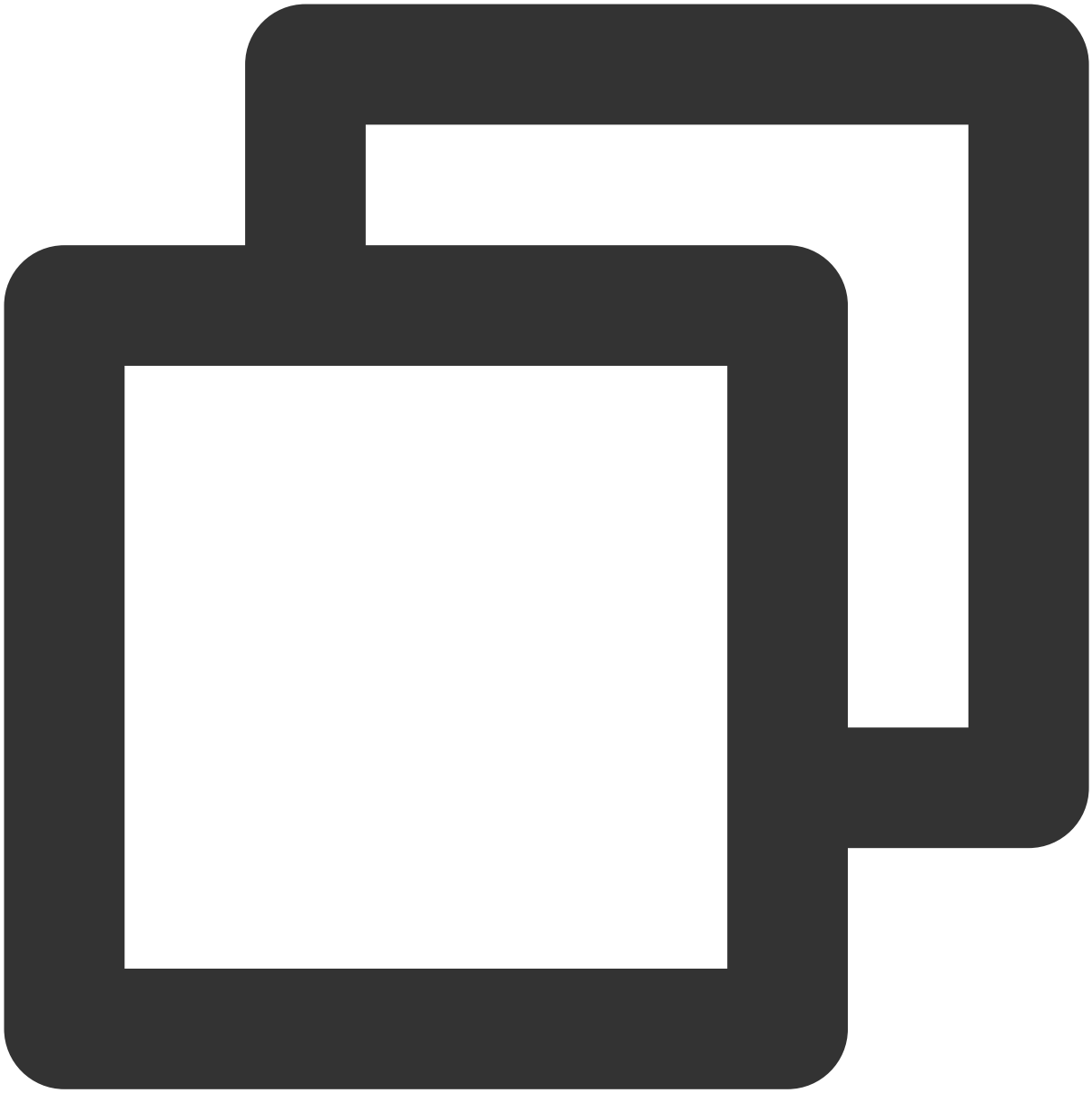


```
<div id="id_local_video" style="width:100%;height:500px;display:flex;align-items:ce
```

步骤3：直播推流

1. 生成推流 SDK 实例：

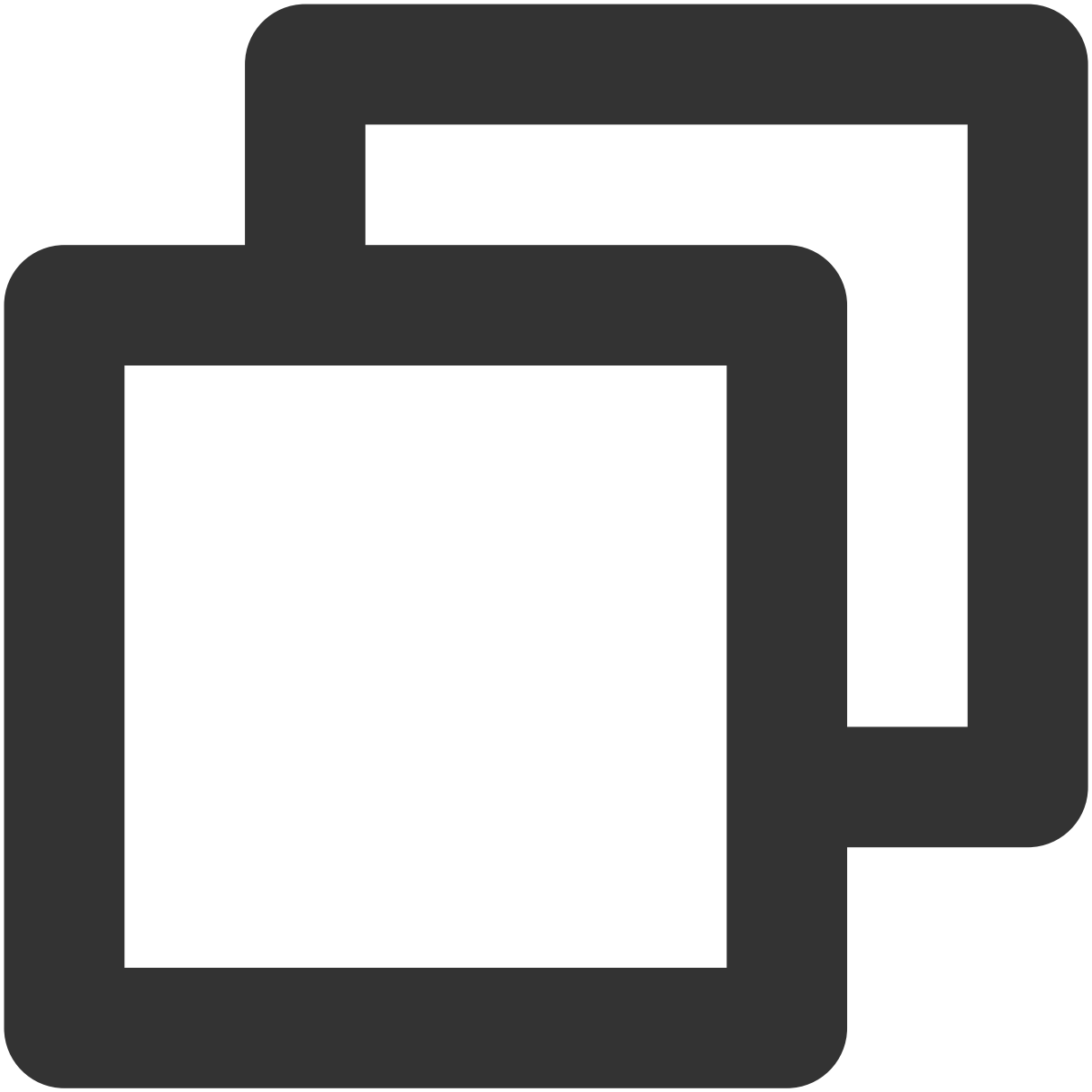
通过全局对象 `TXLivePusher` 生成 SDK 实例，后续操作都是通过实例完成。



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

2. **指定本地视频播放器容器：**

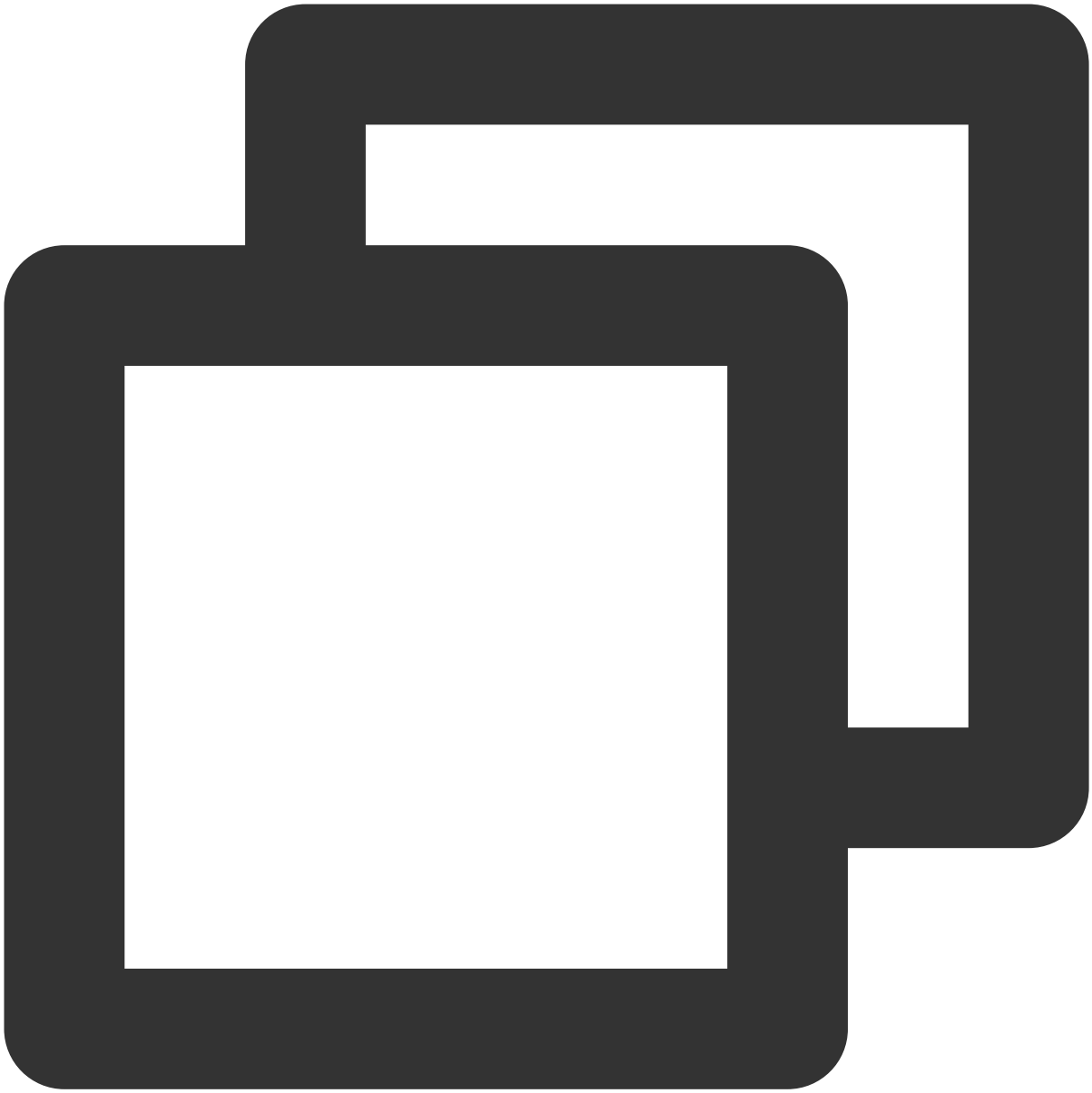
指定本地视频播放器容器 div，浏览器采集到的音视频画面会渲染到这个 div 当中。



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

说明

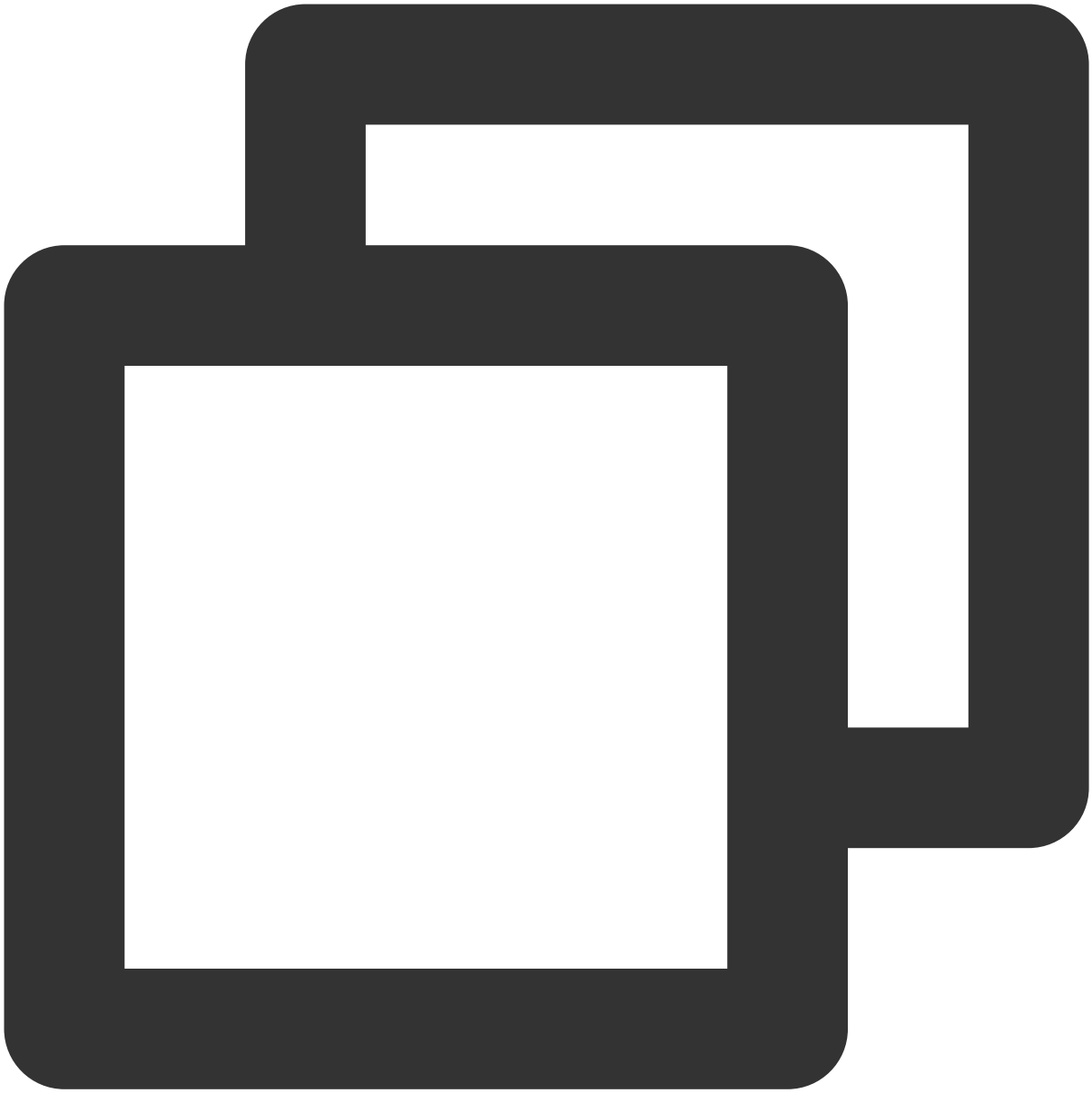
调用 `setRenderView` 生成的 video 元素默认有声音，如果需要静音的话，可以直接获取 video 元素进行操作。



```
document.getElementById('id_local_video').getElementsByTagName('video')[0].muted =
```

3. 设置音视频质量：

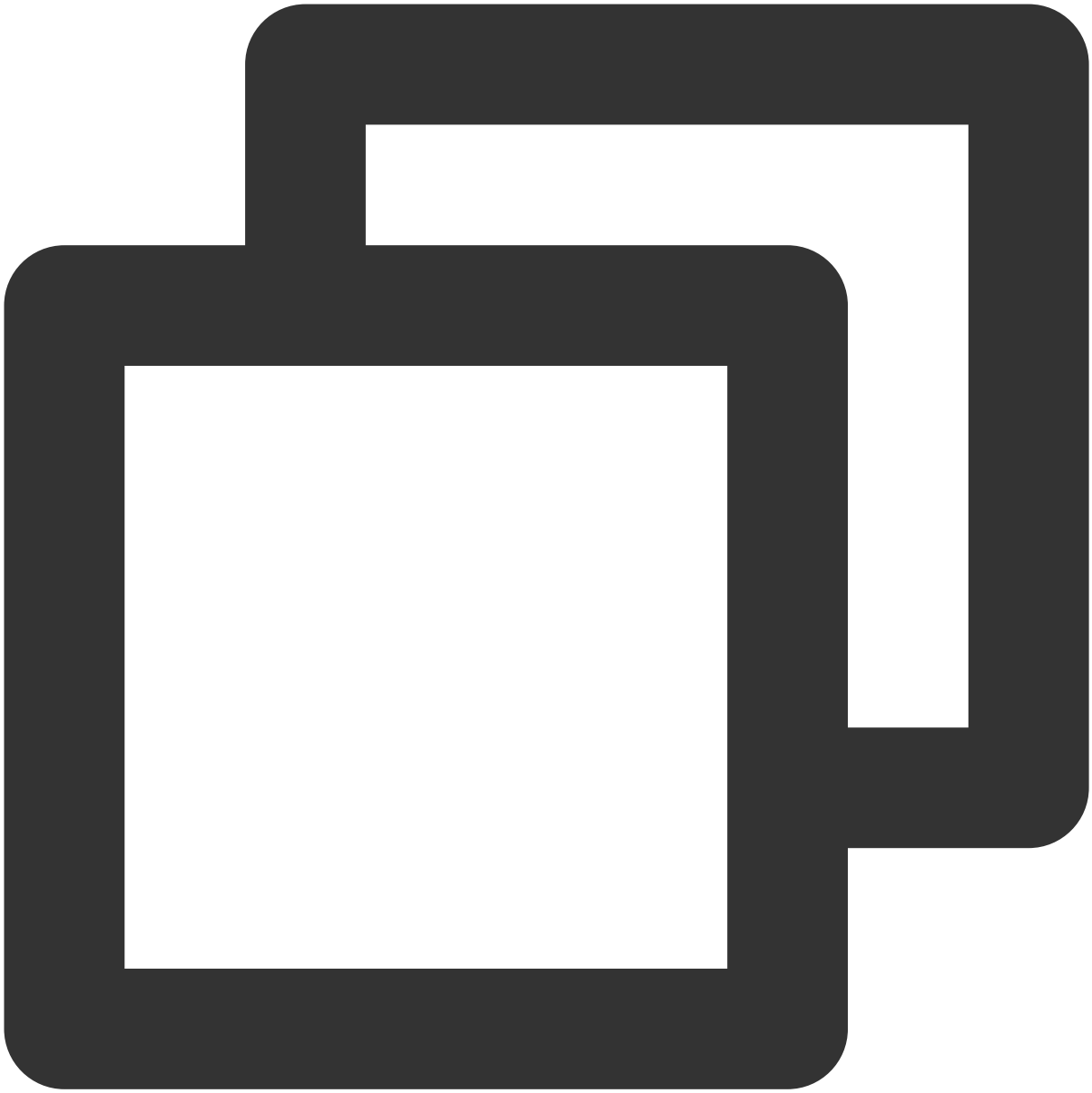
采集音视频流之前，先进行音视频质量设置，如果预设的质量参数不满足需求，可以单独进行自定义设置。



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

4. 开始采集流：

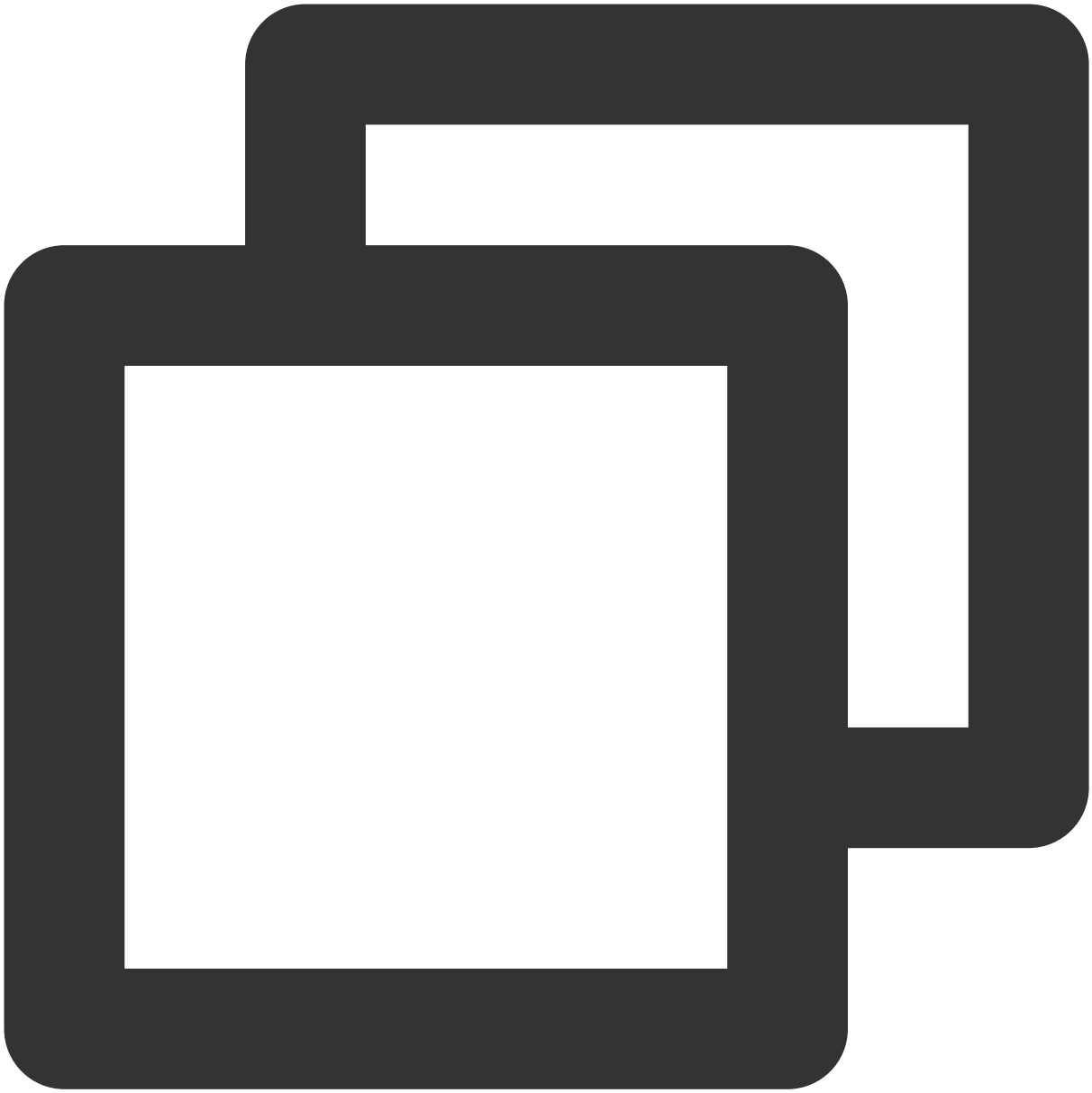
目前支持采集摄像头设备、麦克风设备、屏幕录制和本地媒体文件的流。当音视频流采集成功时，播放器容器中开始播放本地采集到的音视频画面。



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

5. 开始推流：

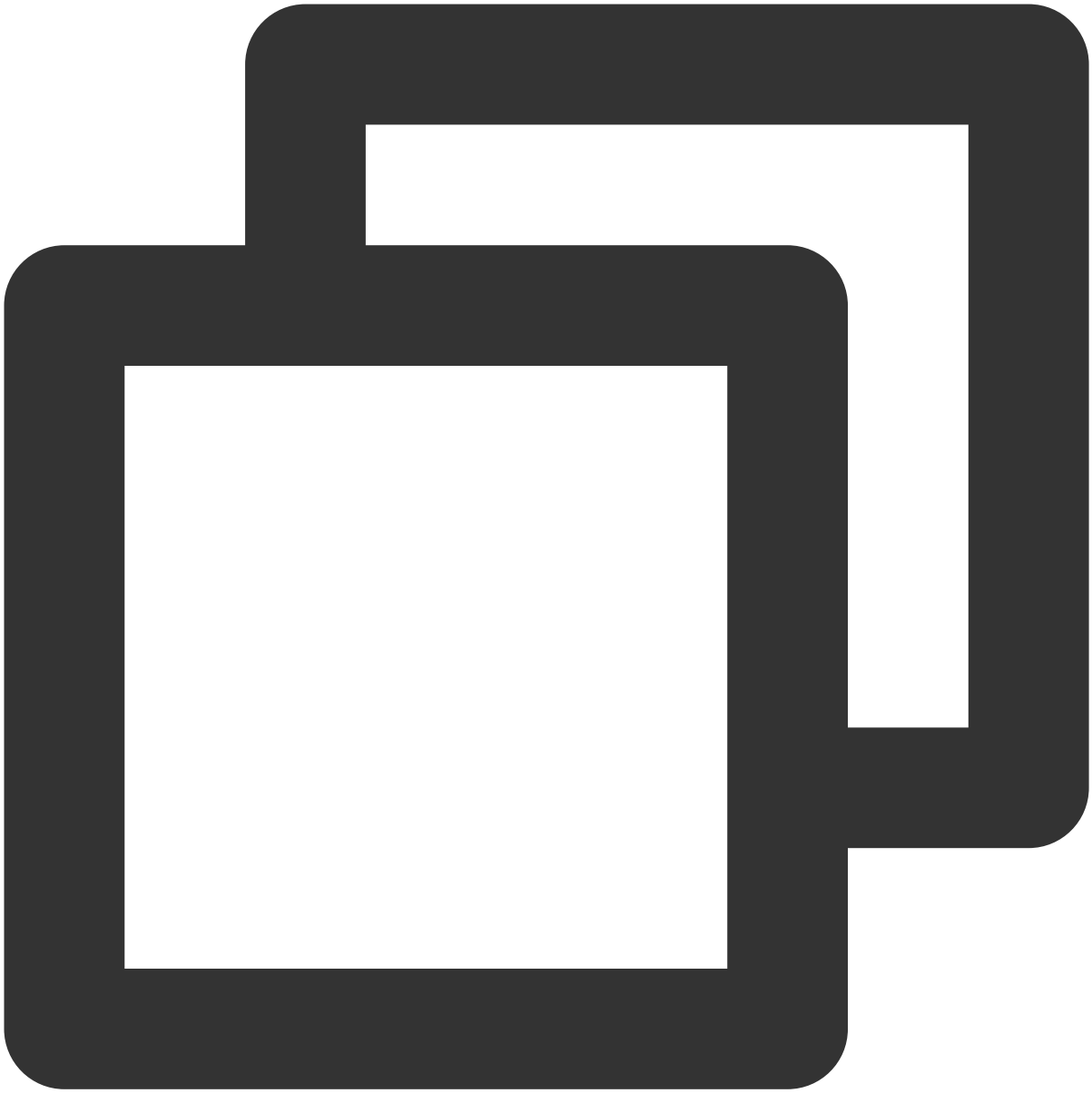
传入腾讯云快直播推流地址，开始推流。推流地址的格式参考 腾讯云标准直播 URL ，只需要将 RTMP 推流地址前面的 `rtmp://` 替换成 `webrtc://` 即可。



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

说明

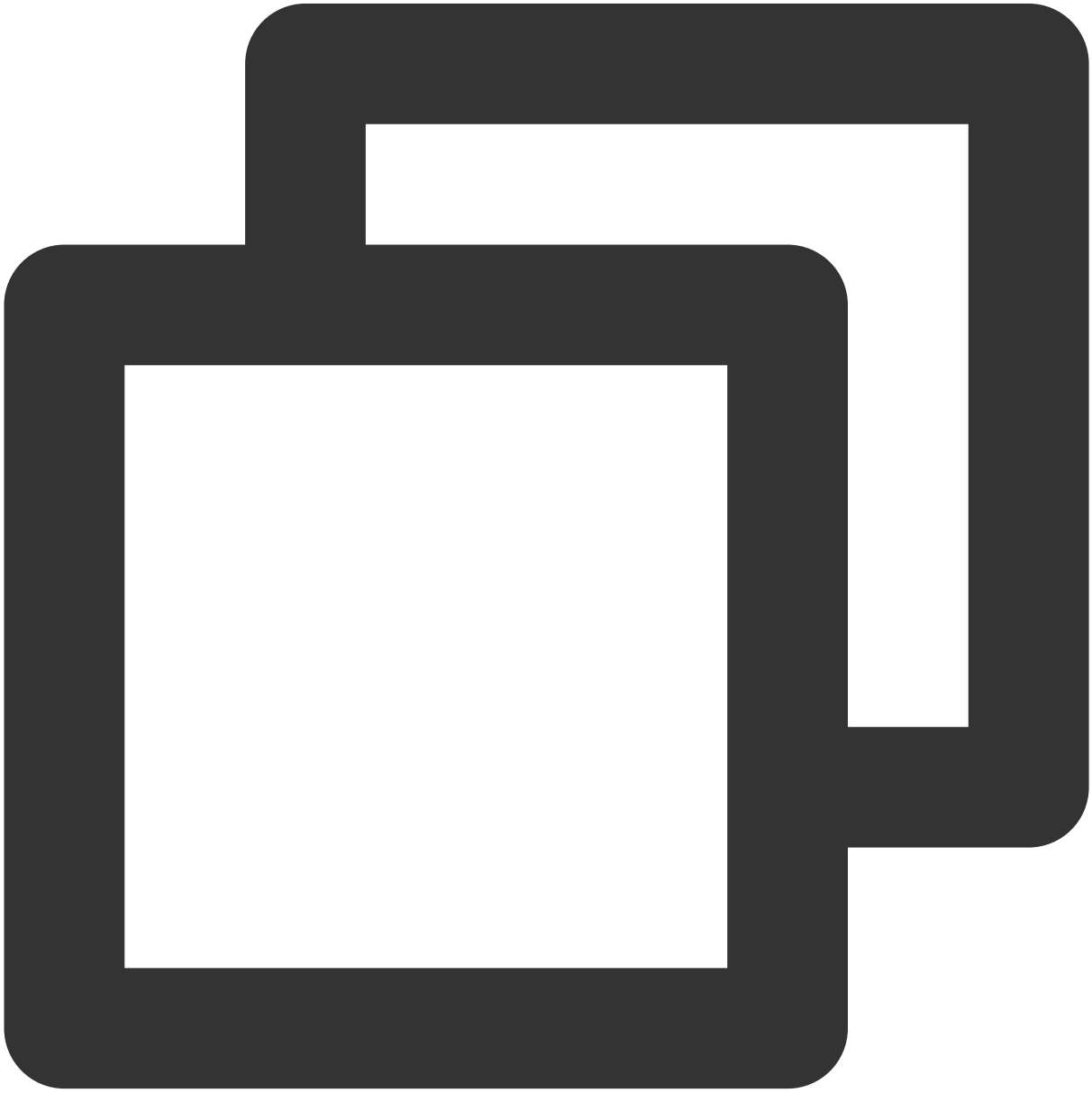
推流之前要保证已经采集到了音视频流，否则推流接口会调用失败，如果要实现采集到音视频流之后自动推流，可以通过回调事件通知，当收到采集首帧成功的通知后，再进行推流。如果同时采集了视频流和音频流，需要在视频首帧和音频首帧的采集成功回调通知都收到后再发起推流。



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

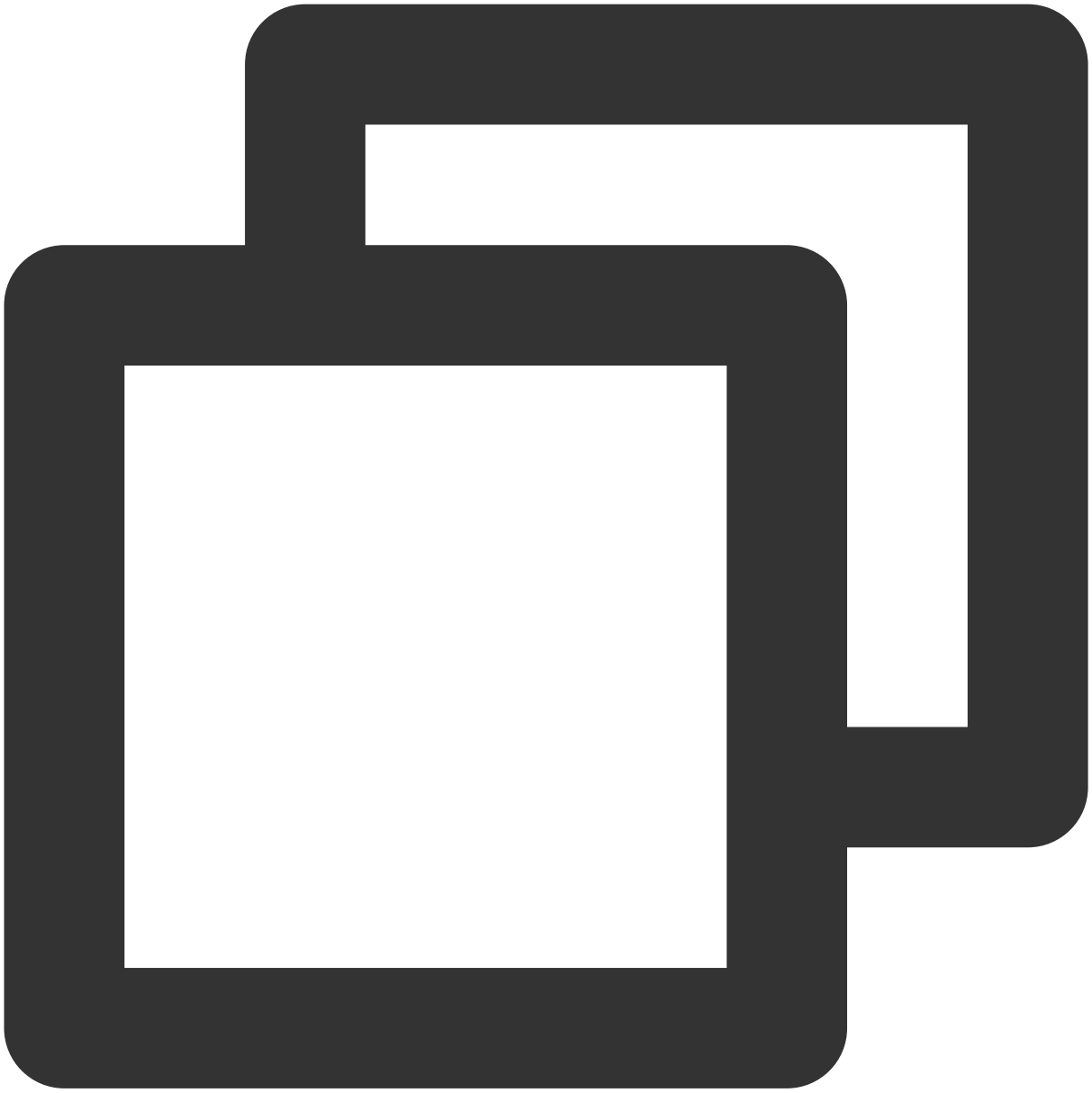
```
  onCaptureFirstVideoFrame: function() {
    hasVideo = true;
    if (hasAudio && !isPush) {
      isPush = true;
      livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=x
    }
  }
});
```

6. **停止快直播推流：**



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

7. **停止采集音视频流：**

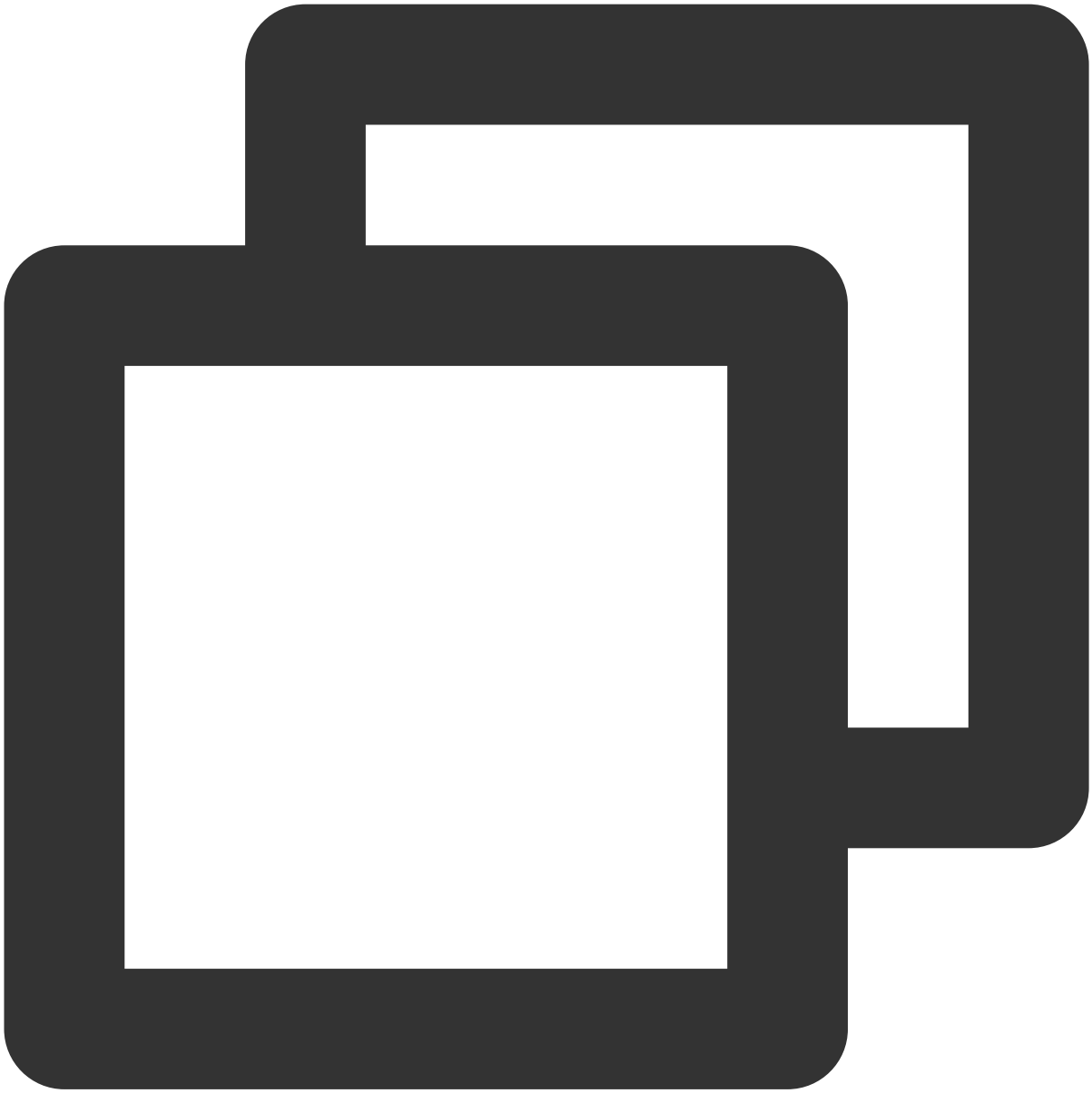


```
// 关闭摄像头
livePusher.stopCamera();
// 关闭麦克风
livePusher.stopMicrophone();
```

进阶攻略

兼容性

SDK 提供静态方法用于检测浏览器对于 WebRTC 的兼容性。

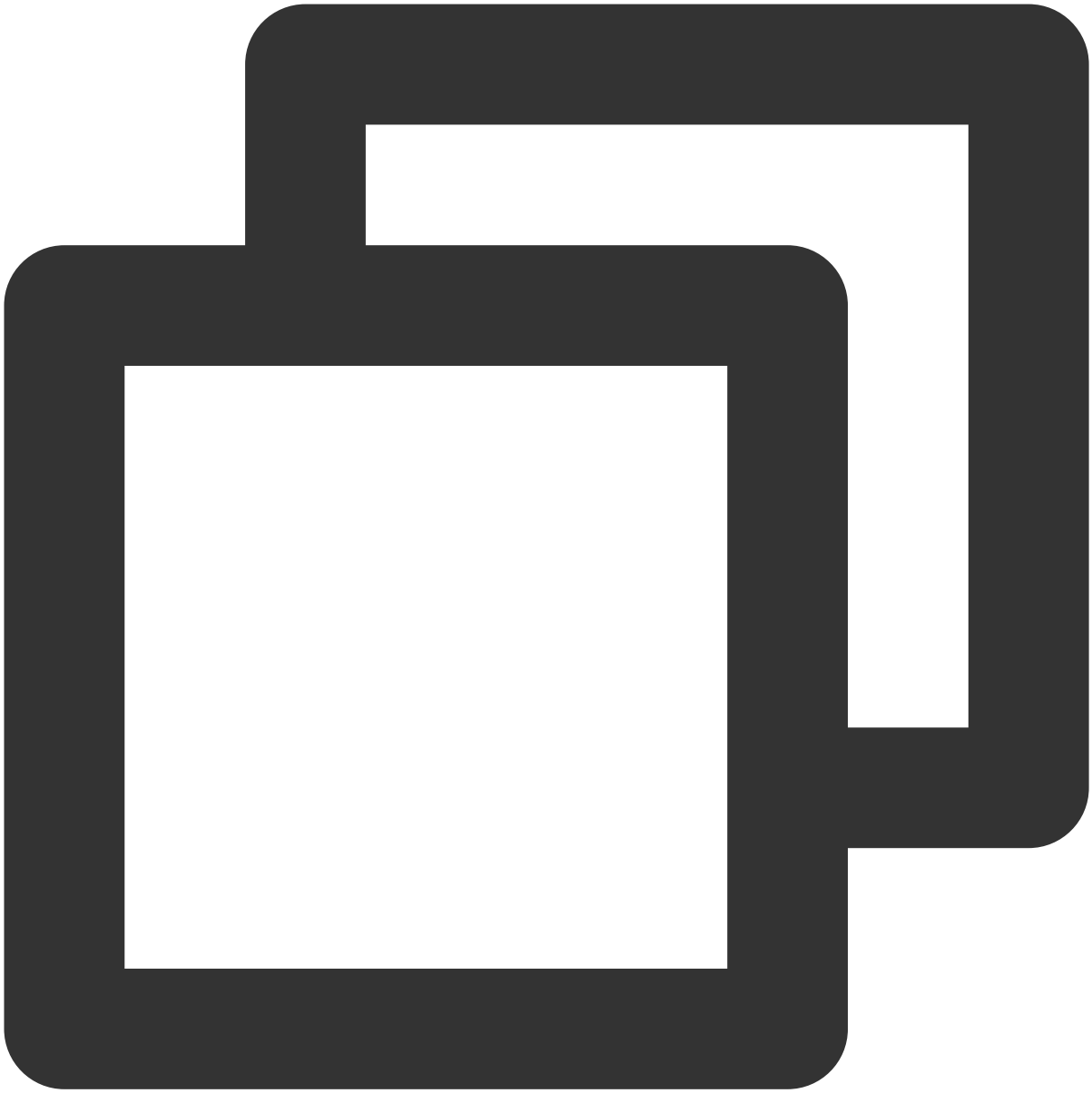


```
TXLivePusher.checkSupport().then(function(data) {
  // 是否支持WebRTC
  if (data.isWebRTCSupported) {
    console.log('WebRTC Support');
  } else {
    console.log('WebRTC Not Support');
  }
  // 是否支持H264编码
```

```
  if (data.isH264EncodeSupported) {
    console.log('H264 Encode Support');
  } else {
    console.log('H264 Encode Not Support');
  }
});
```

回调事件通知

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

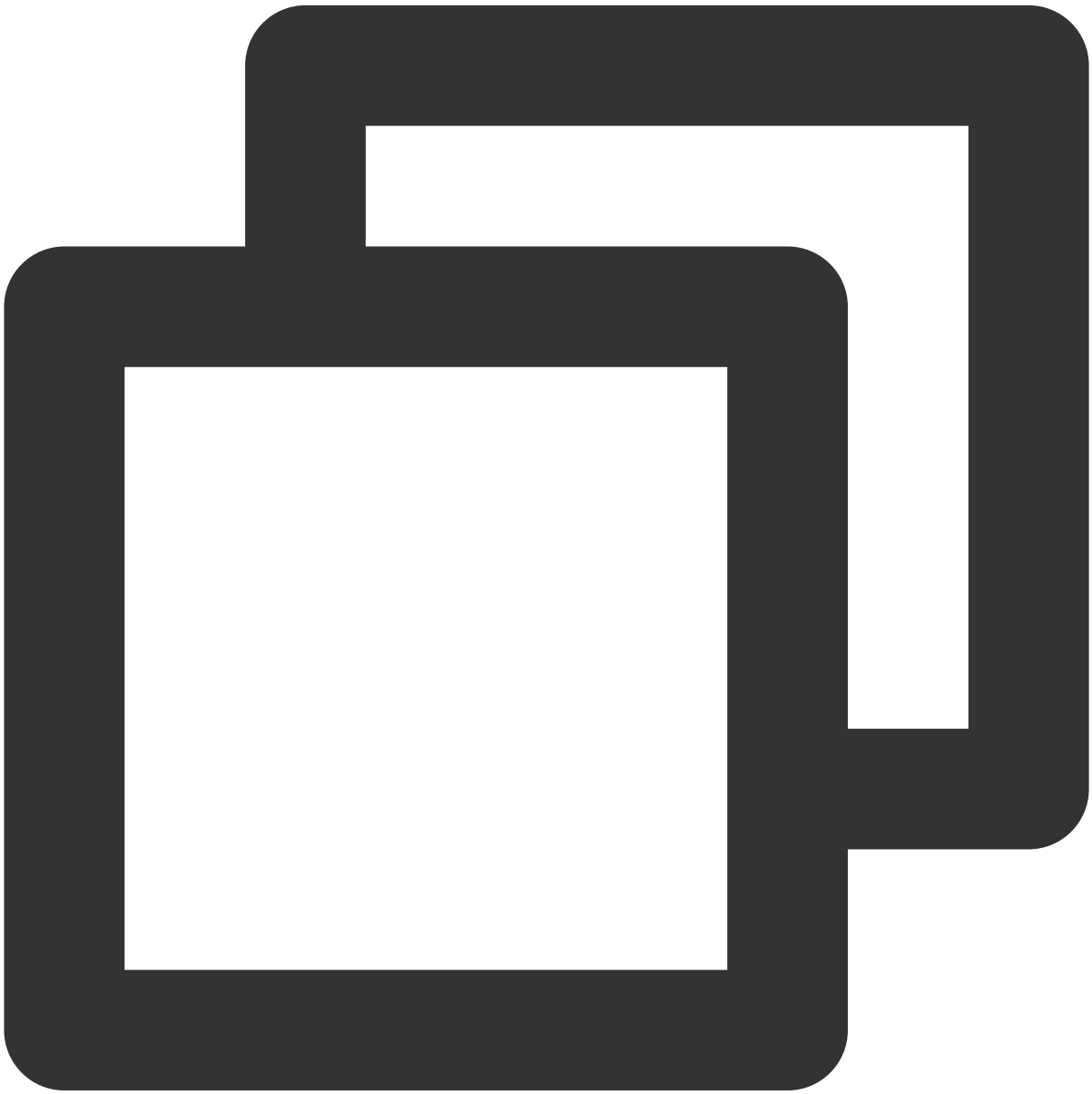


```
livePusher.setObserver({
  // 推流警告信息
  onWarning: function(code, msg) {
    console.log(code, msg);
  },
  // 推流连接状态
  onPushStatusUpdate: function(status, msg) {
    console.log(status, msg);
  },
  // 推流统计数据
  onStatisticsUpdate: function(data) {
```

```
    console.log('video fps is ' + data.video.framesPerSecond);
  }
});
```

设备管理

SDK 提供了设备管理实例帮助用户进行获取设备列表、切换设备等操作。



```
var deviceManager = livePusher.getDeviceManager();
// 获取设备列表
```

```
deviceManager.getDevicesList().then(function(data) {
  data.forEach(function(device) {
    console.log(device.deviceId, device.deviceName);
  });
});
// 切换摄像头设备
deviceManager.switchCamera('camera_device_id');
```

Flutter
摄像头推流

最近更新时间：2024-01-13 15:53:49

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

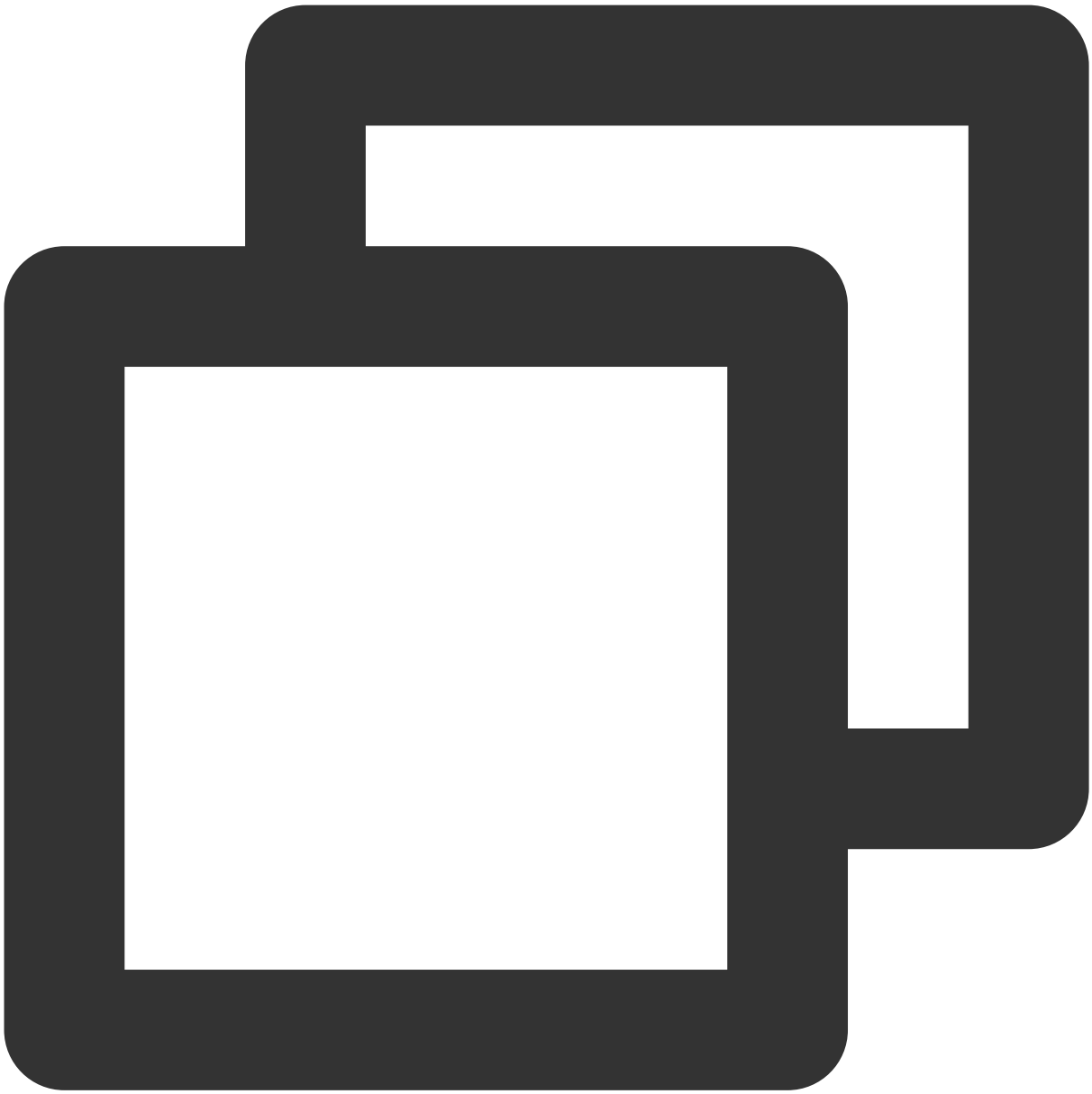
所属平台	GitHub 地址	关键类
iOS	Github	CameraPushViewController.m
Android	Github	CameraPushMainActivity.java
Flutter	Github	live_camera_push.dart

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。

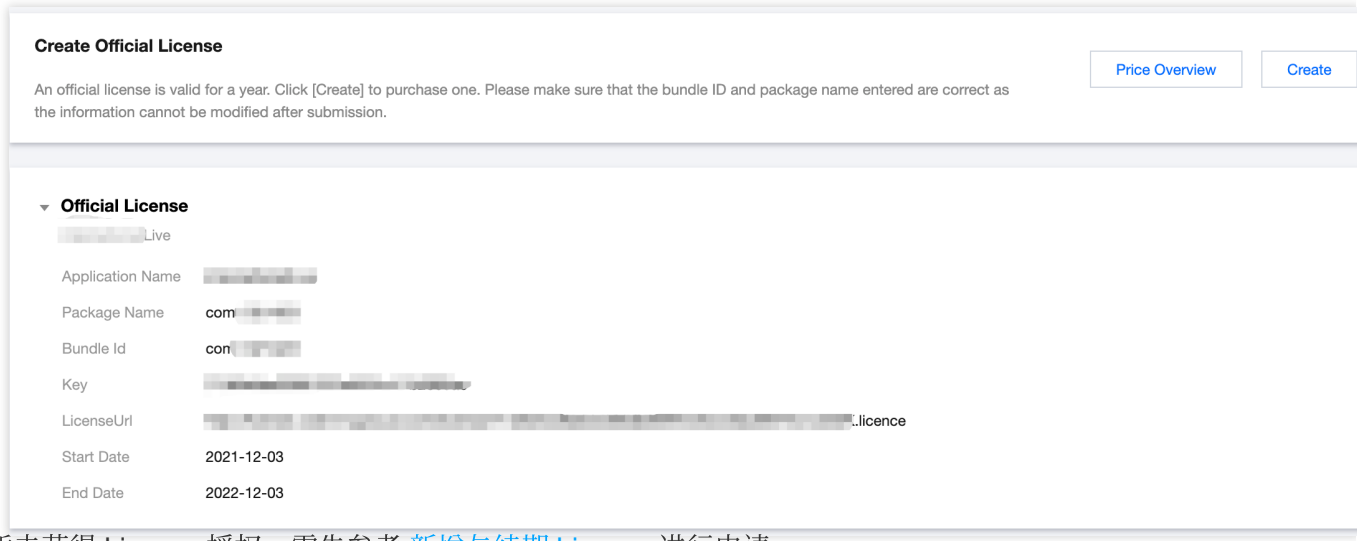


```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

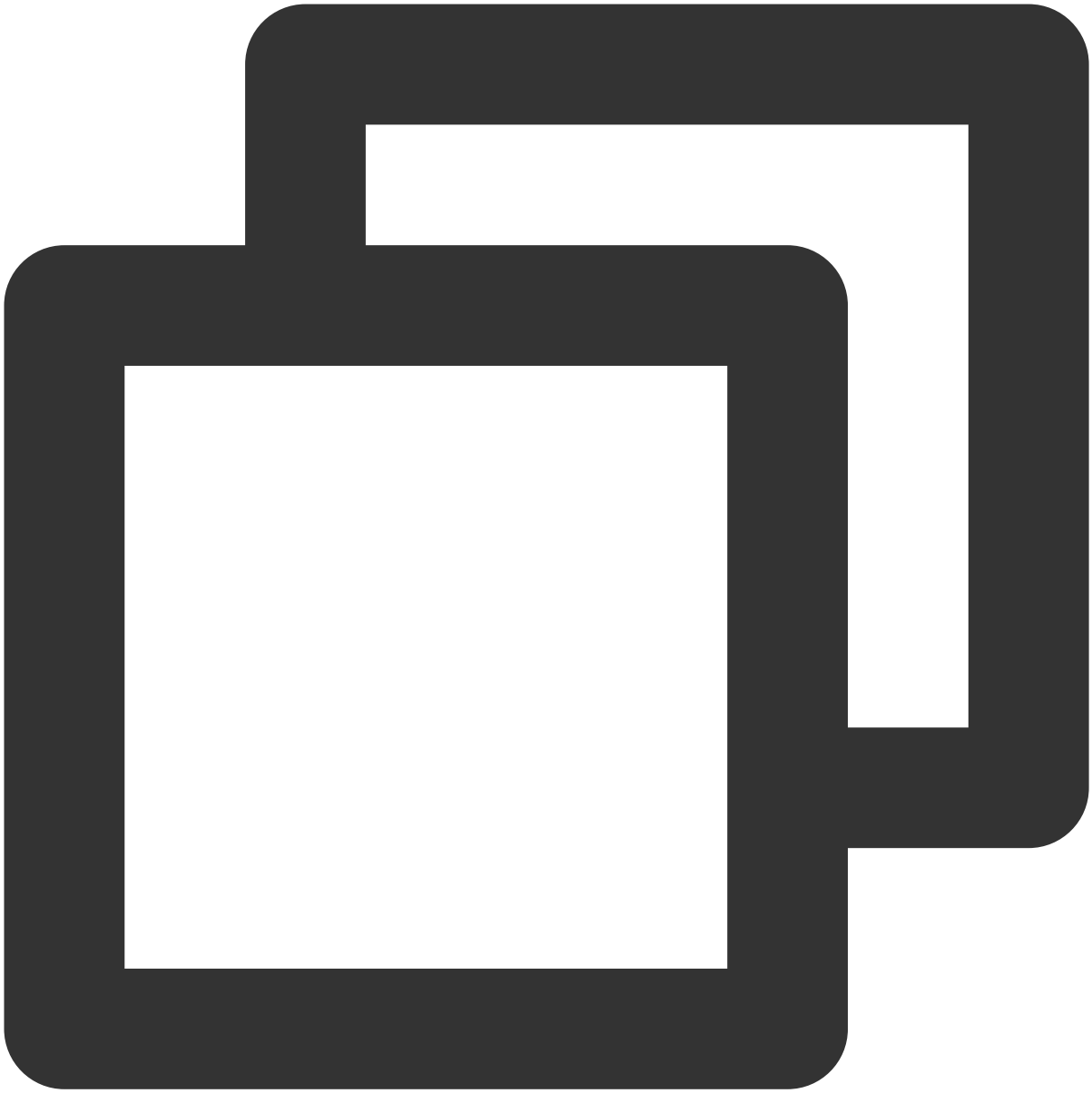
1. 获取 License 授权：

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：



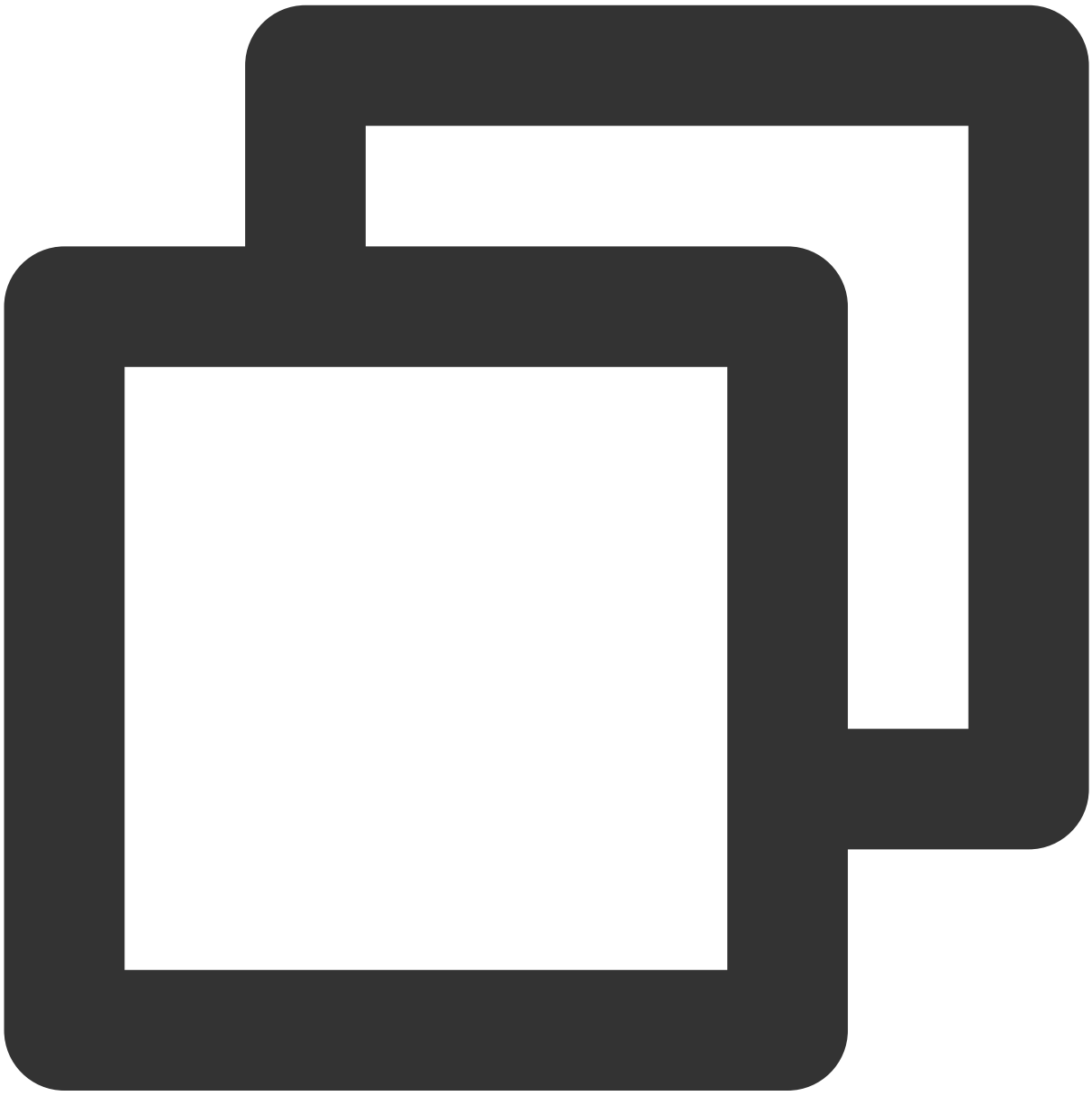
```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.tencentcloud.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

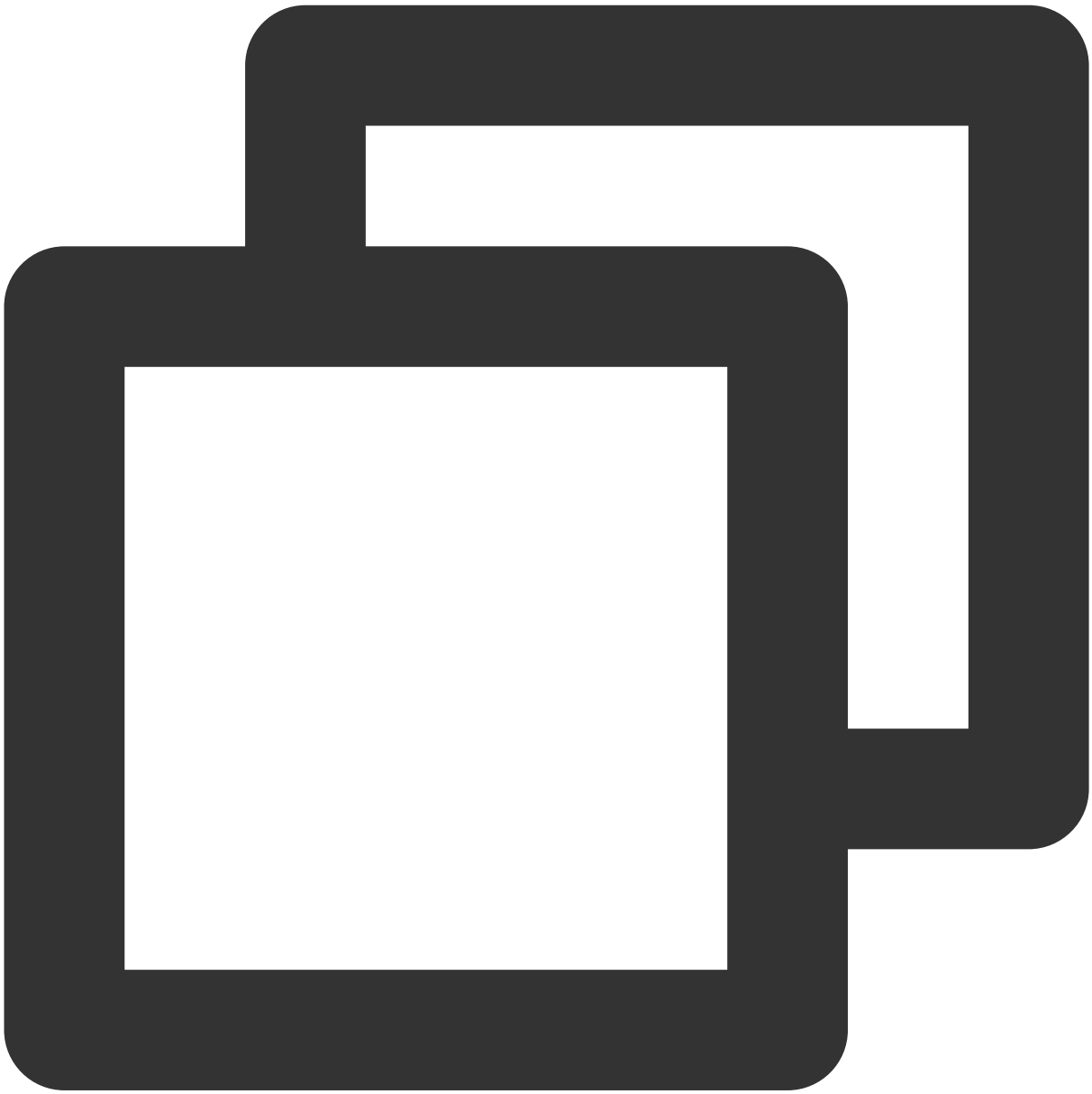
创建一个 `V2TXLivePusher` 对象，需要指定对应的 `V2TXLiveMode` 。



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

/// V2TXLivePusher 初始化
initPusher() {
```

```
  _livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTC);
}
```

4. 设置视频渲染 RenderView

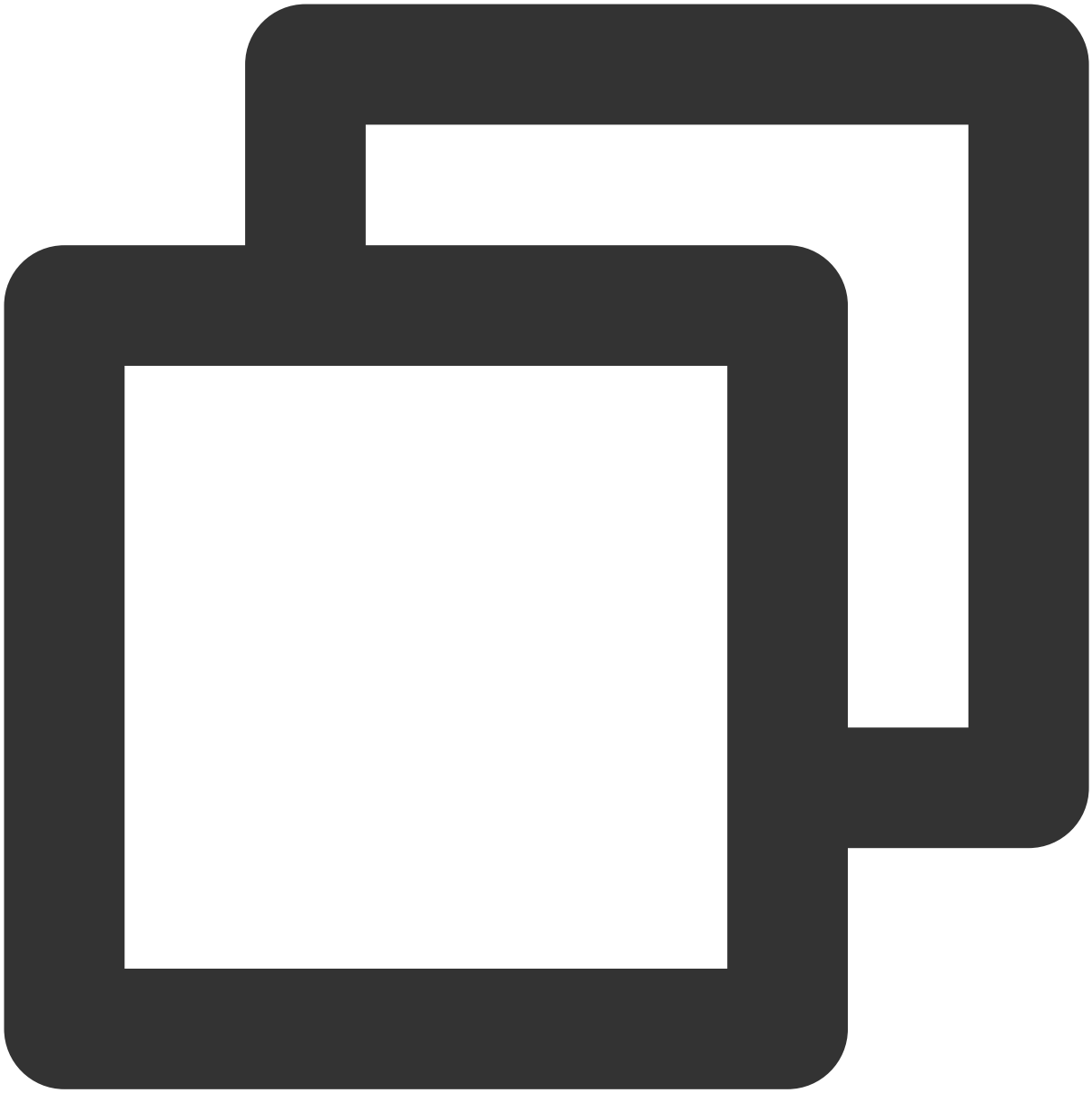


```
import 'package:live_flutter_plugin/widget/v2_tx_live_video_widget.dart';

/// 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);
      },
    );
  }
```

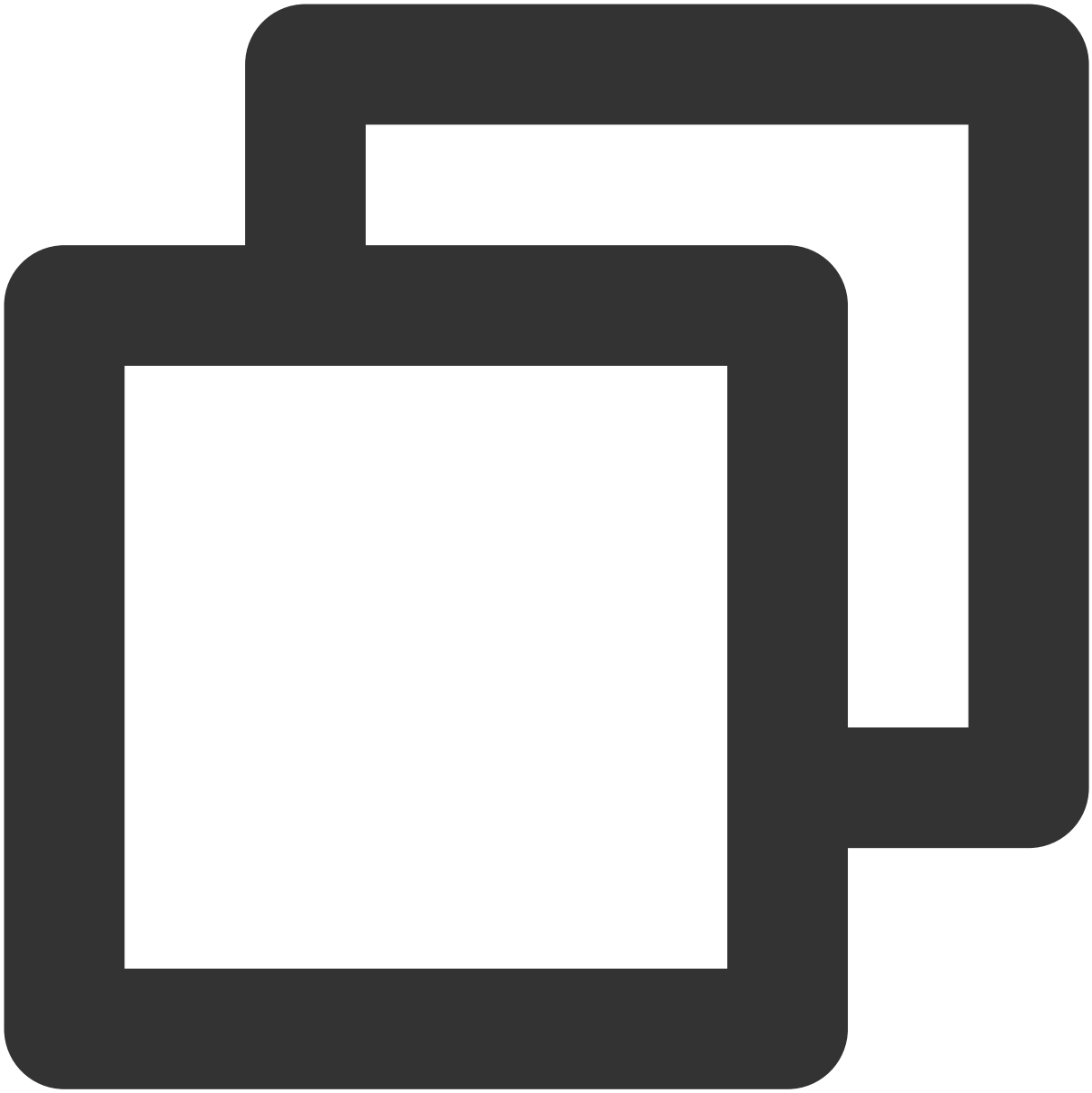
5. 启动和结束推流

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。



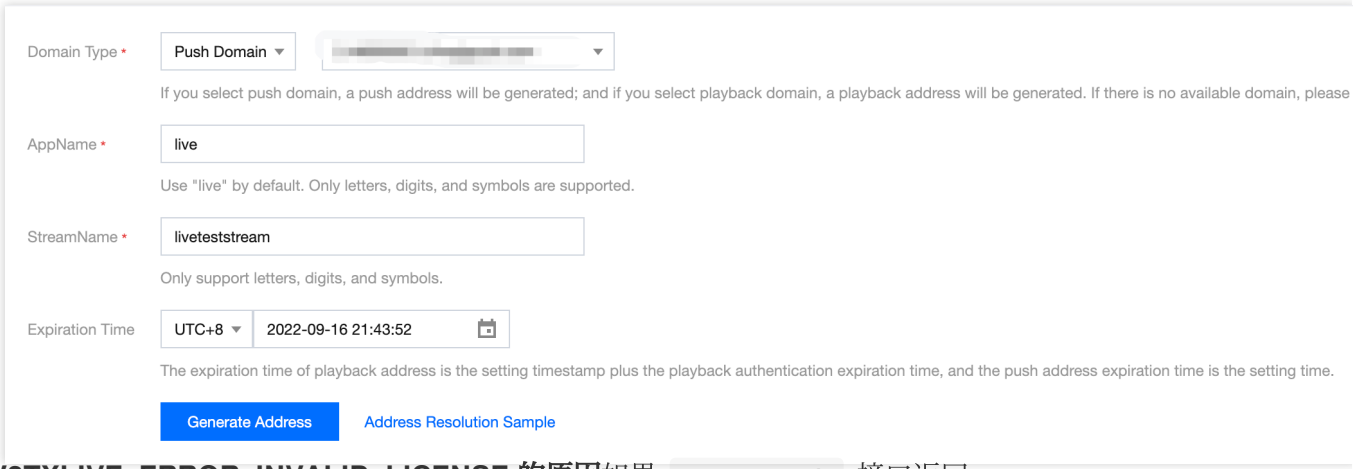
```
/// 停止推流
stopPush() async {
  // 关闭摄像头
  await _livePusher.stopCamera();
  // 关闭麦克风
  await _livePusher.stopMicrophone();
  // 停止推流
  await _livePusher.stopPush();
}
```

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

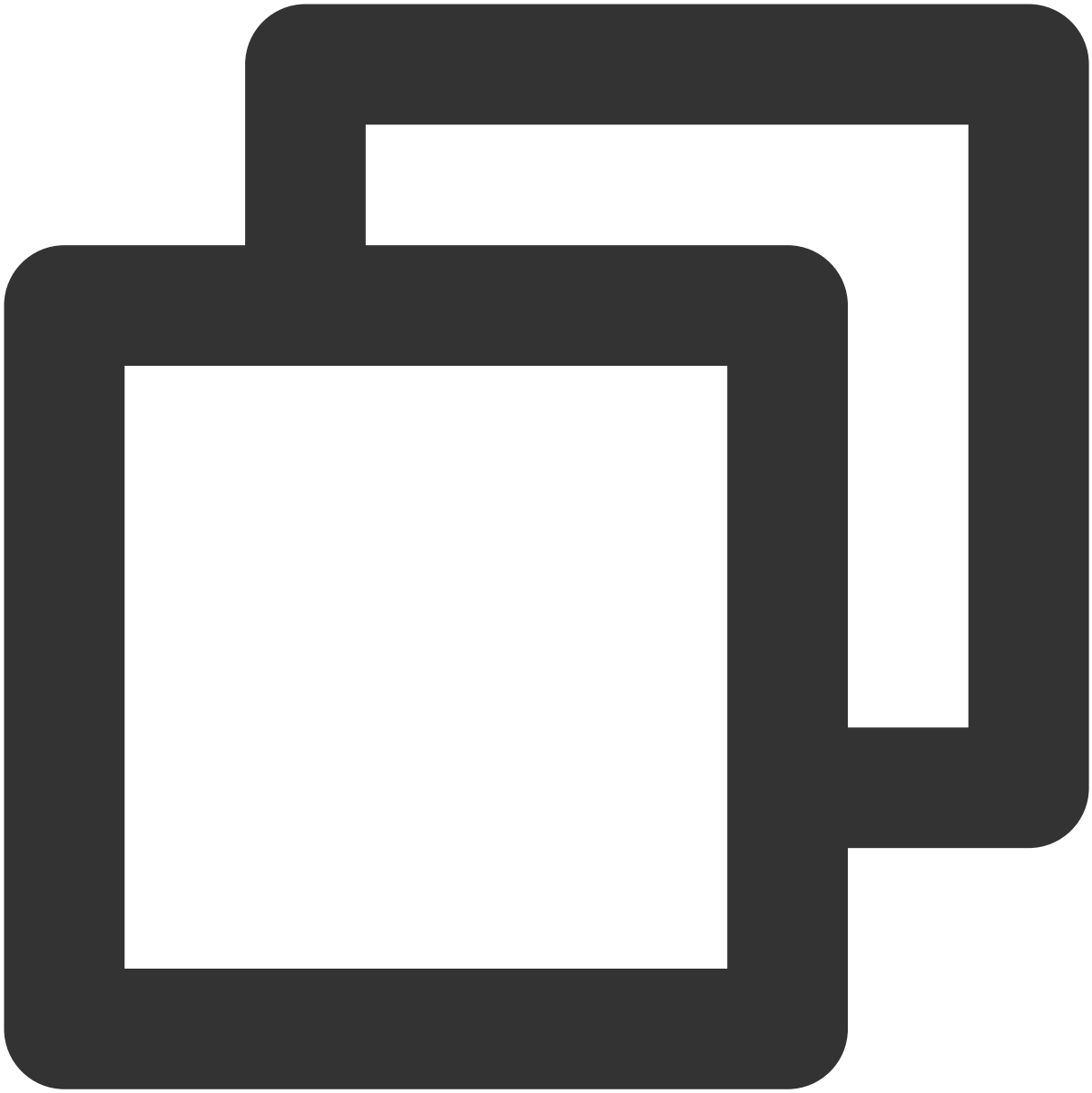
开通直播服务后，可以使用**直播控制台 > 辅助工具 > 地址生成器** 生成推流地址，详细信息请参见 推拉流 URL。



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.v2TXLiveModeRTC);
  // 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：

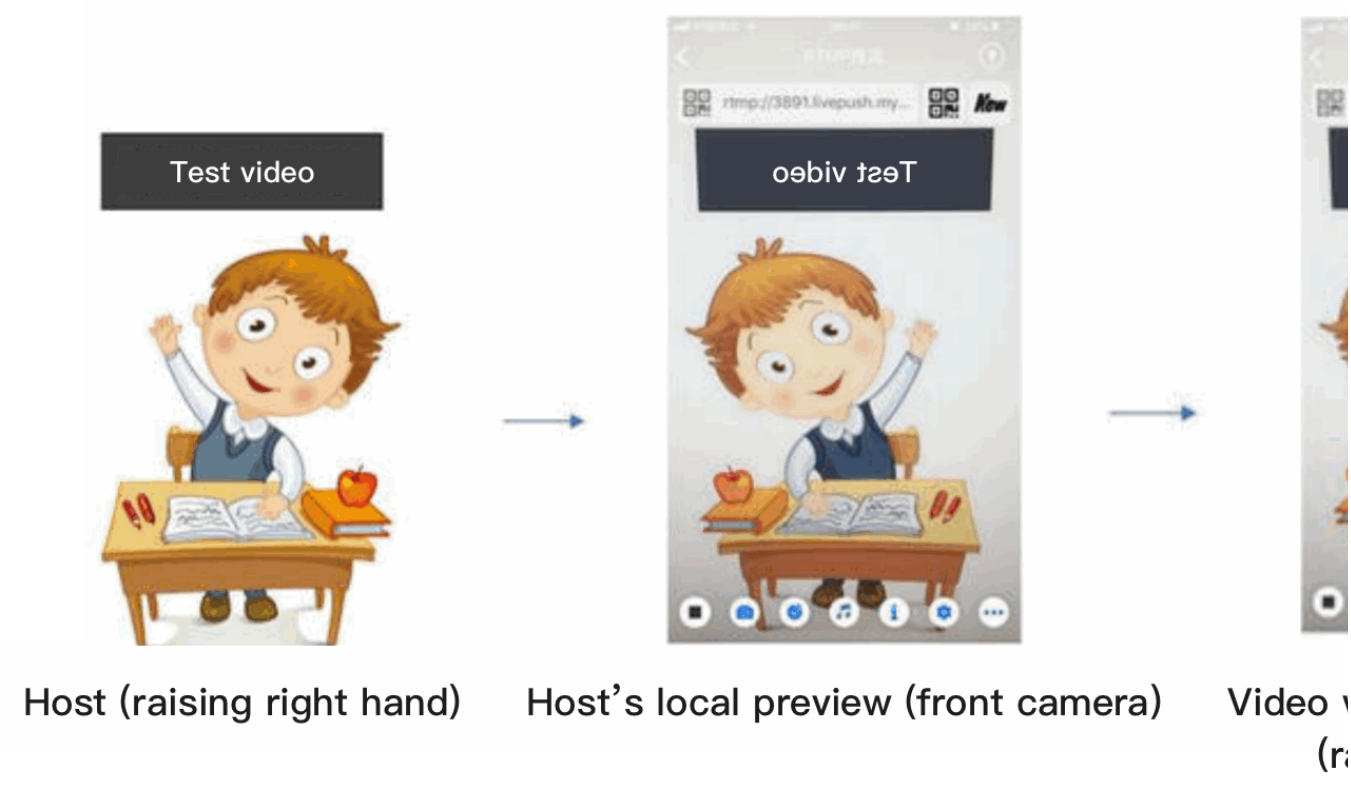
美颜风格	设置方式	接口说明
美颜级别	通过 TXBeautyManager 的 setBeautyLevel 设置	取值范围0 - 9； 0表示关闭，1 - 9 值越大，效果越明显
美白级别	通过 TXBeautyManager 的 setWhitenessLevel 设置	取值范围0 - 9； 0表示关闭，1 - 9 值越大，效果越明显
红润级别	通过 TXBeautyManager 的 setRuddyLevel 设置	取值范围0 - 9； 0表示关闭，1 - 9 值越大，效果越明显

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。

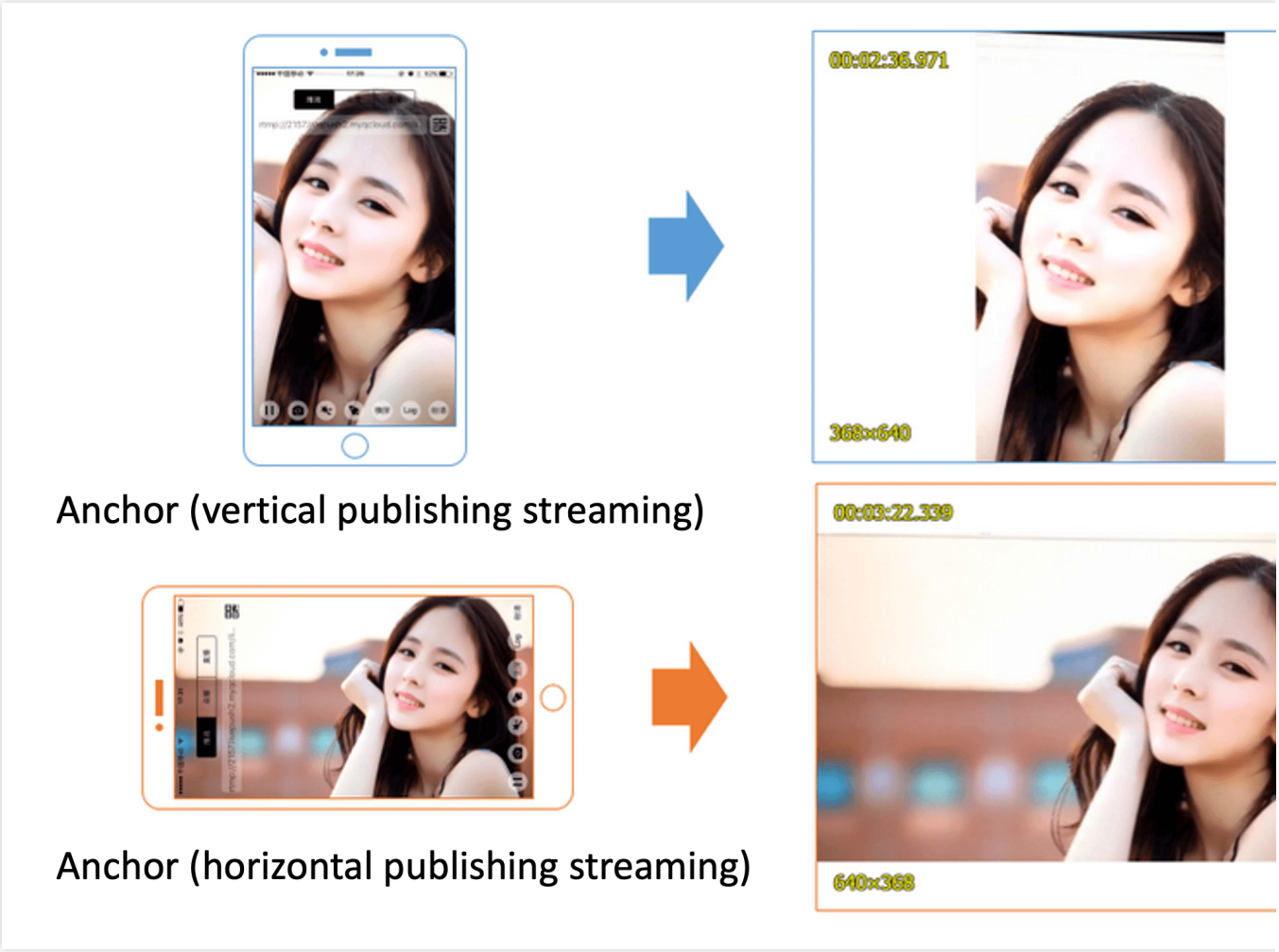
10. 观众端的镜像效果

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。

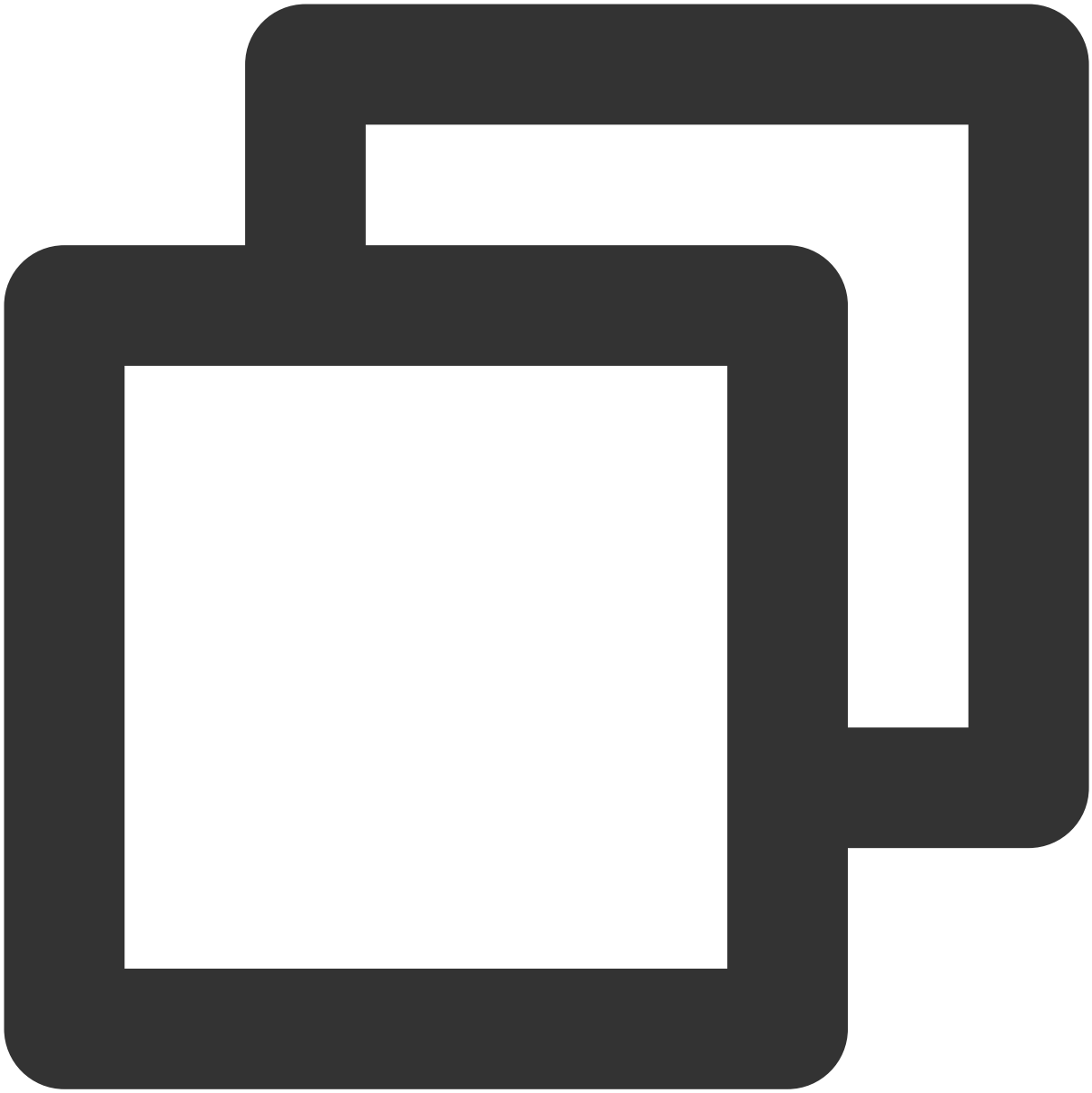


11. 横屏推流

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）。



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。

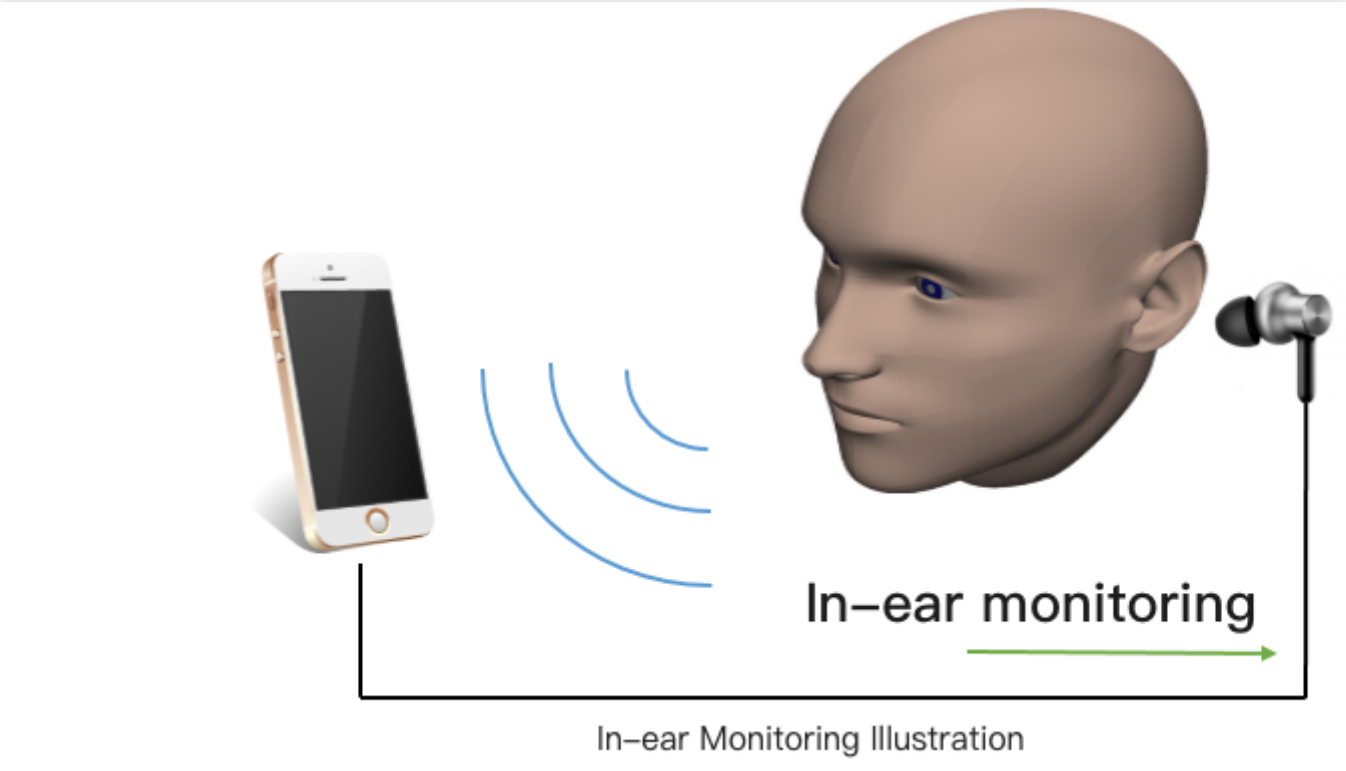


```
// 视频编码参数
var param = V2TXLiveVideoEncoderParam();
param.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionMode.v2TXLiveVideo
_livePusher.setVideoQuality(param);
```

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。



In-ear Monitoring Illustration

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_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：2024-01-13 15:53:49

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。

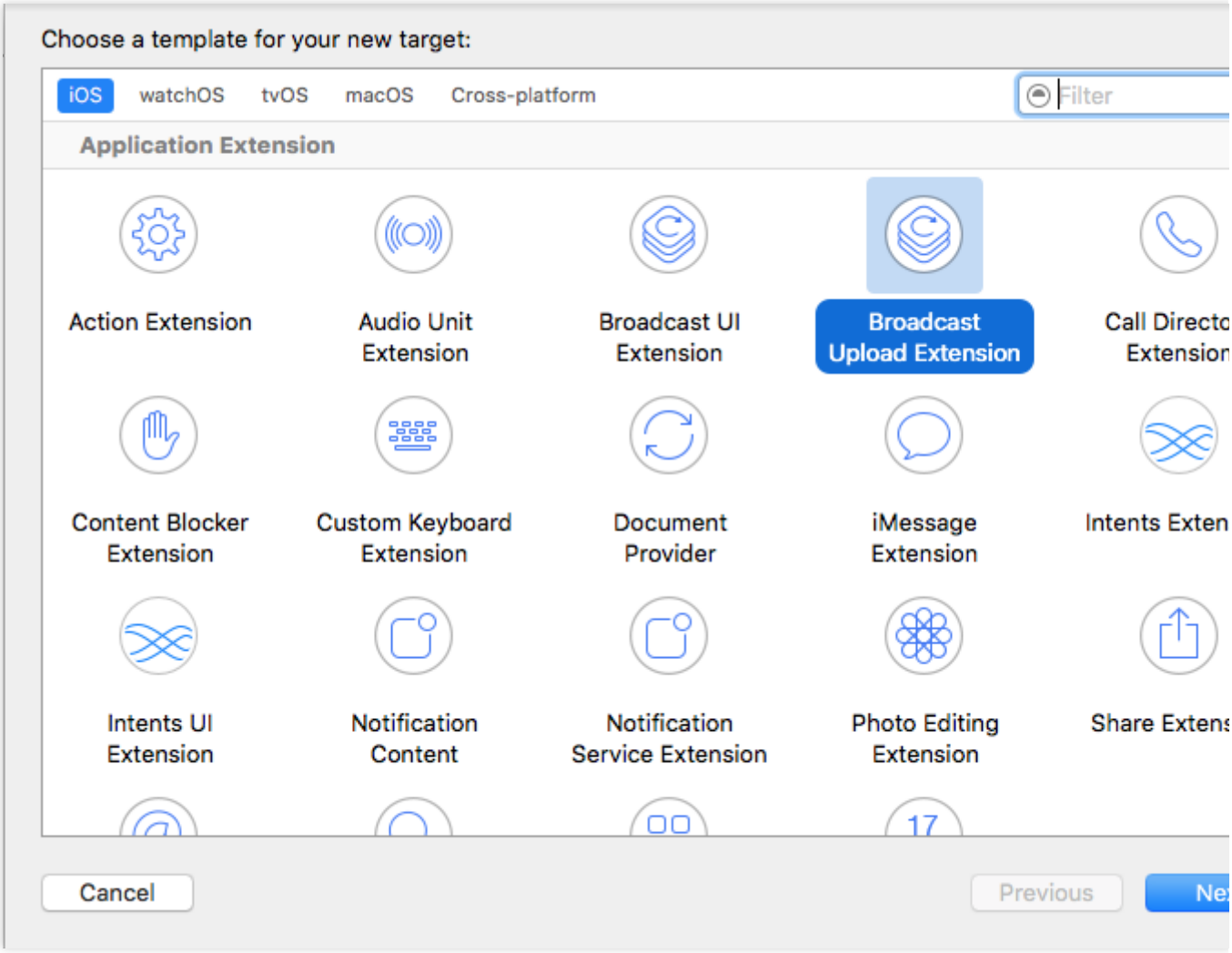
iOS

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。

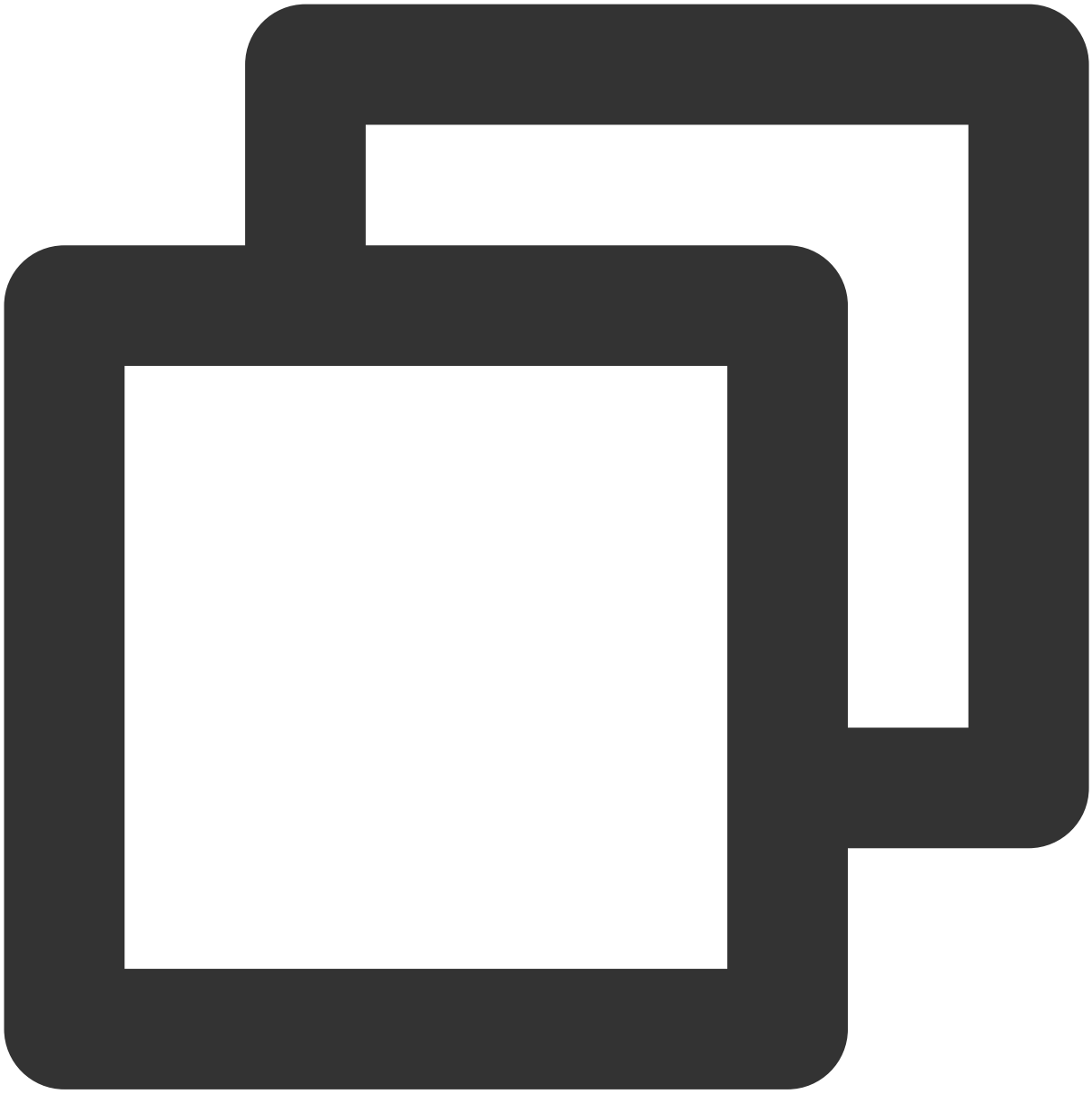
2. **导入TXLiteAVSDK_ReplayKitExt.framework**

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。

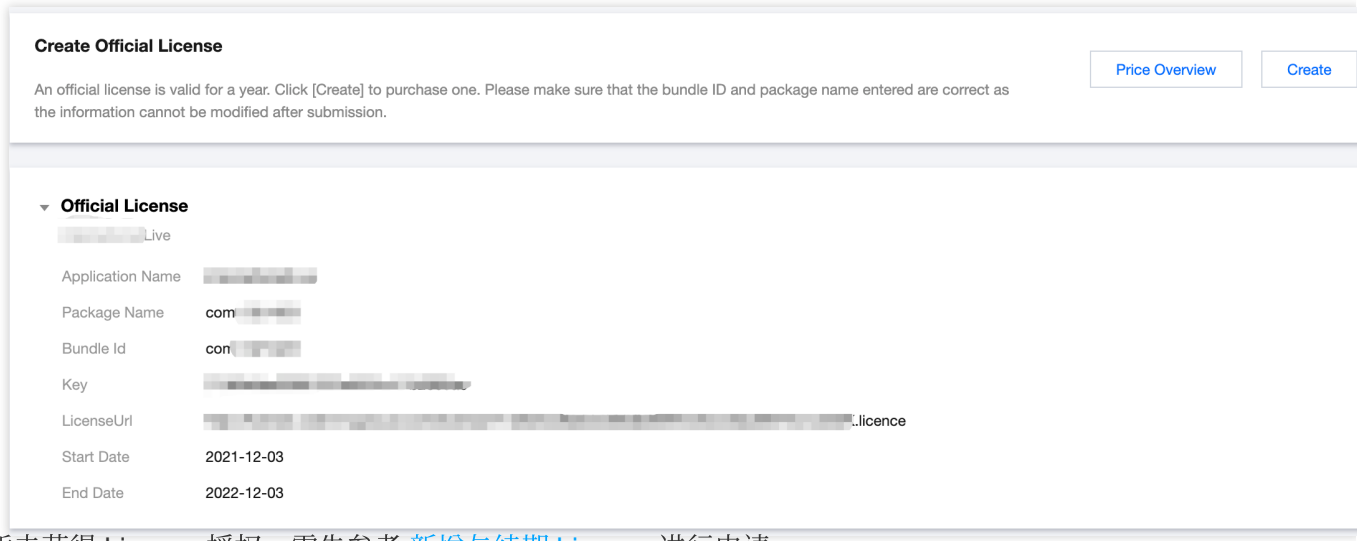


```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

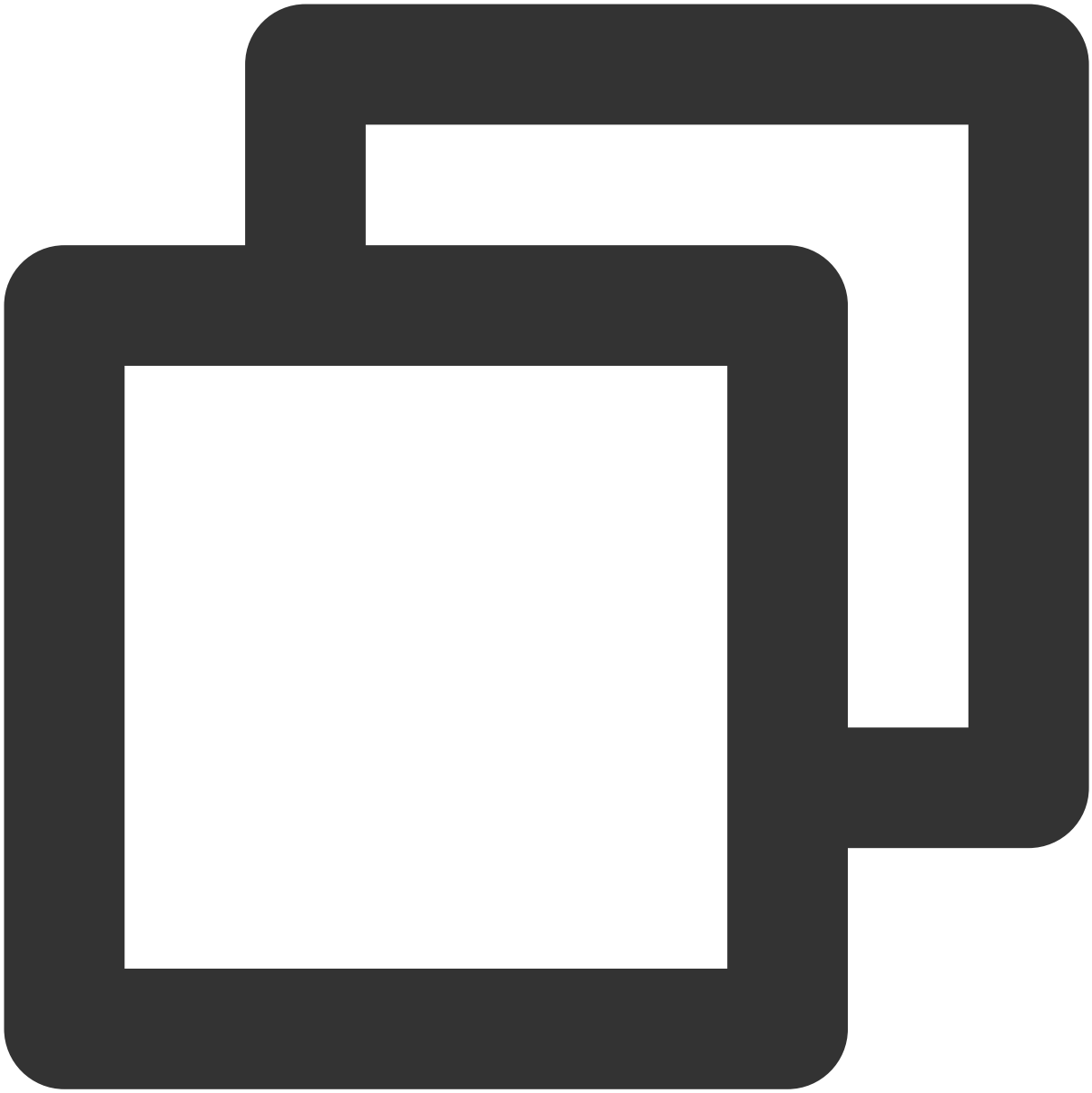
1. 获取 License 授权：

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：



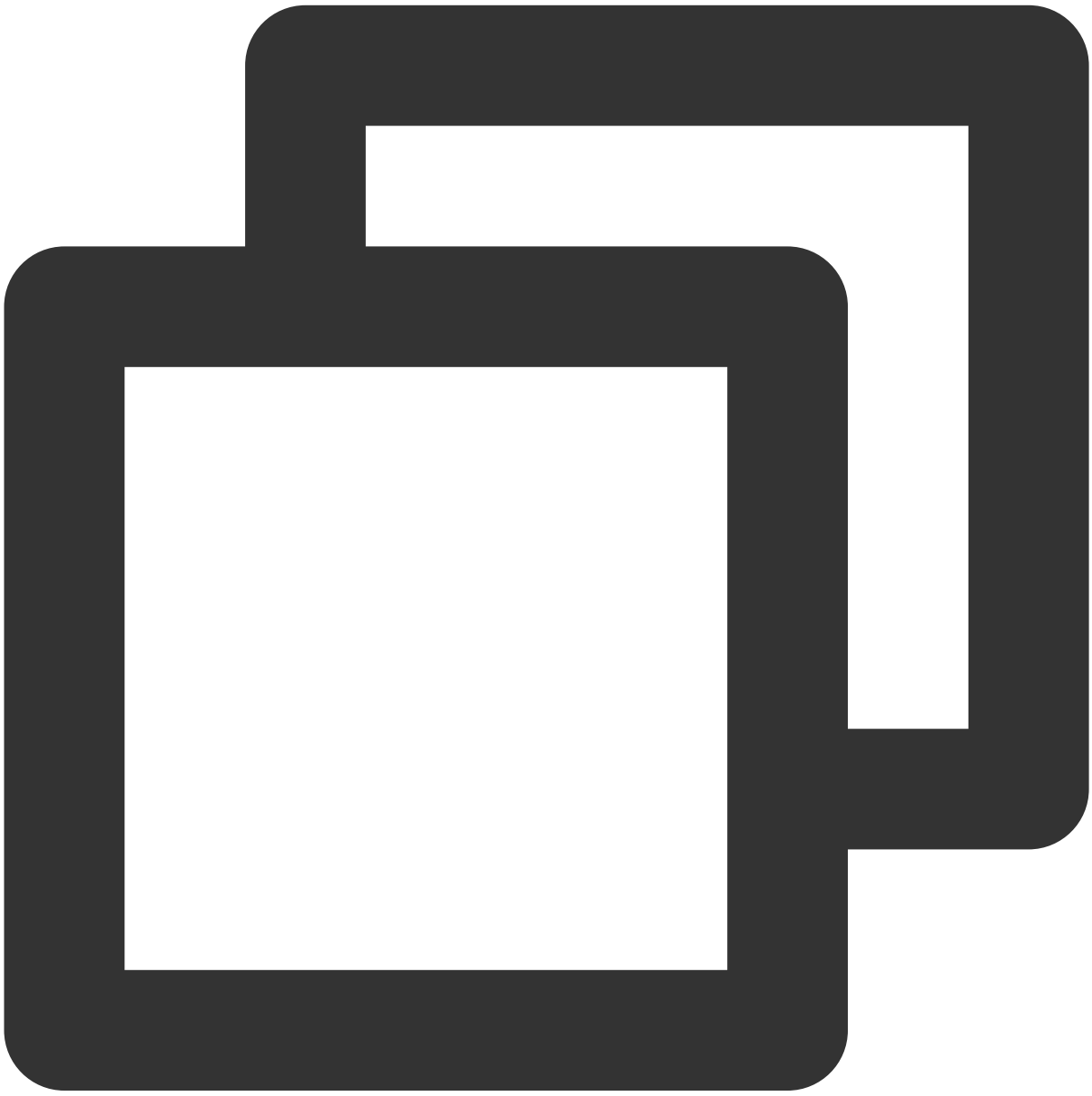
```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.tencentcloud.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

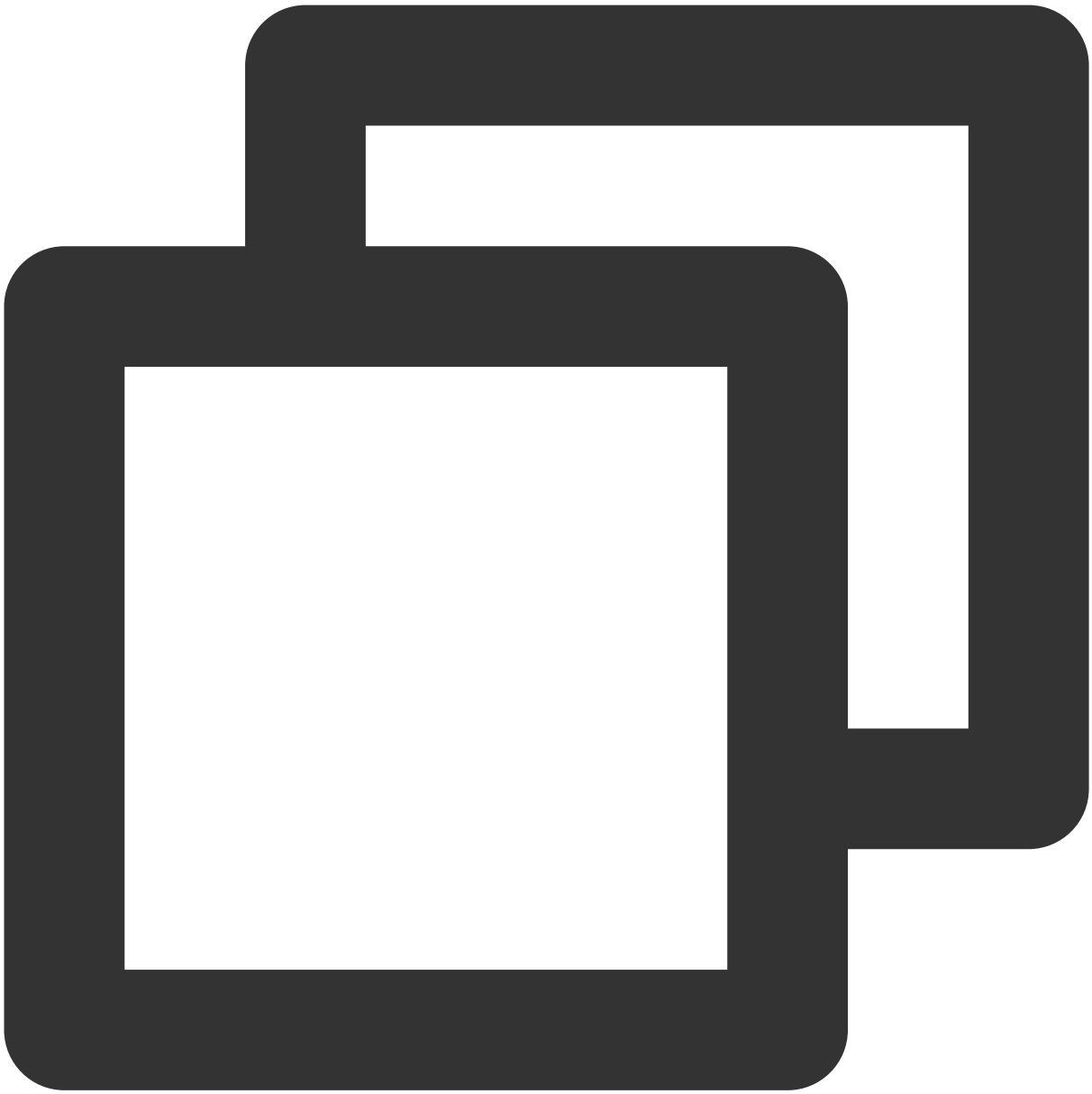
创建一个 **V2TXLivePusher** 对象，我们后面主要用它来完成推流工作。



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

4. 启动推流

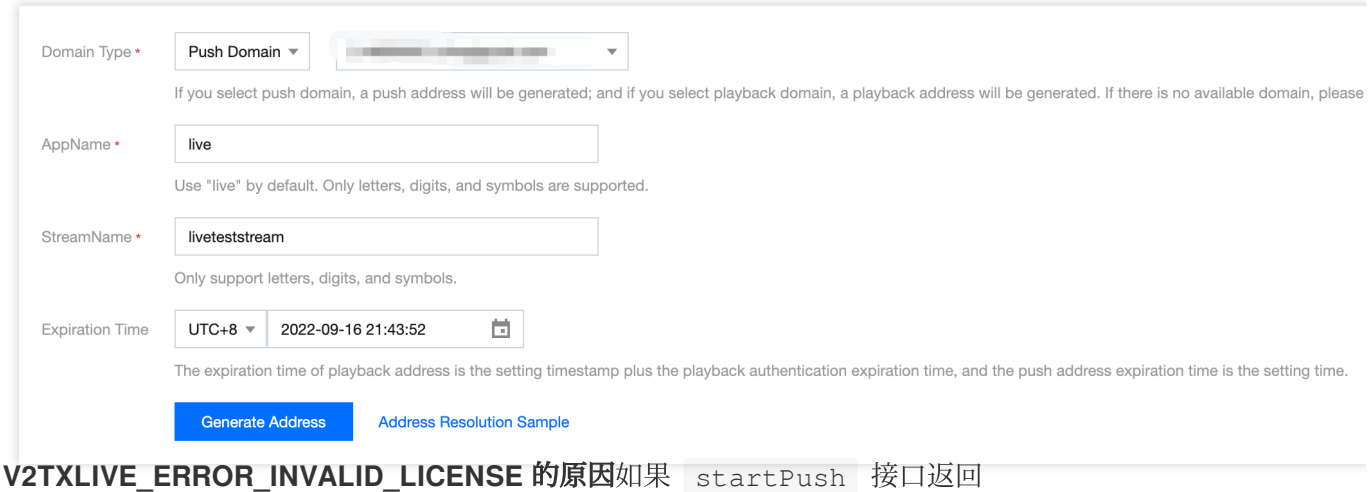
经过 步骤1 的准备之后，用下面这段代码就可以启动推流了：



```
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？

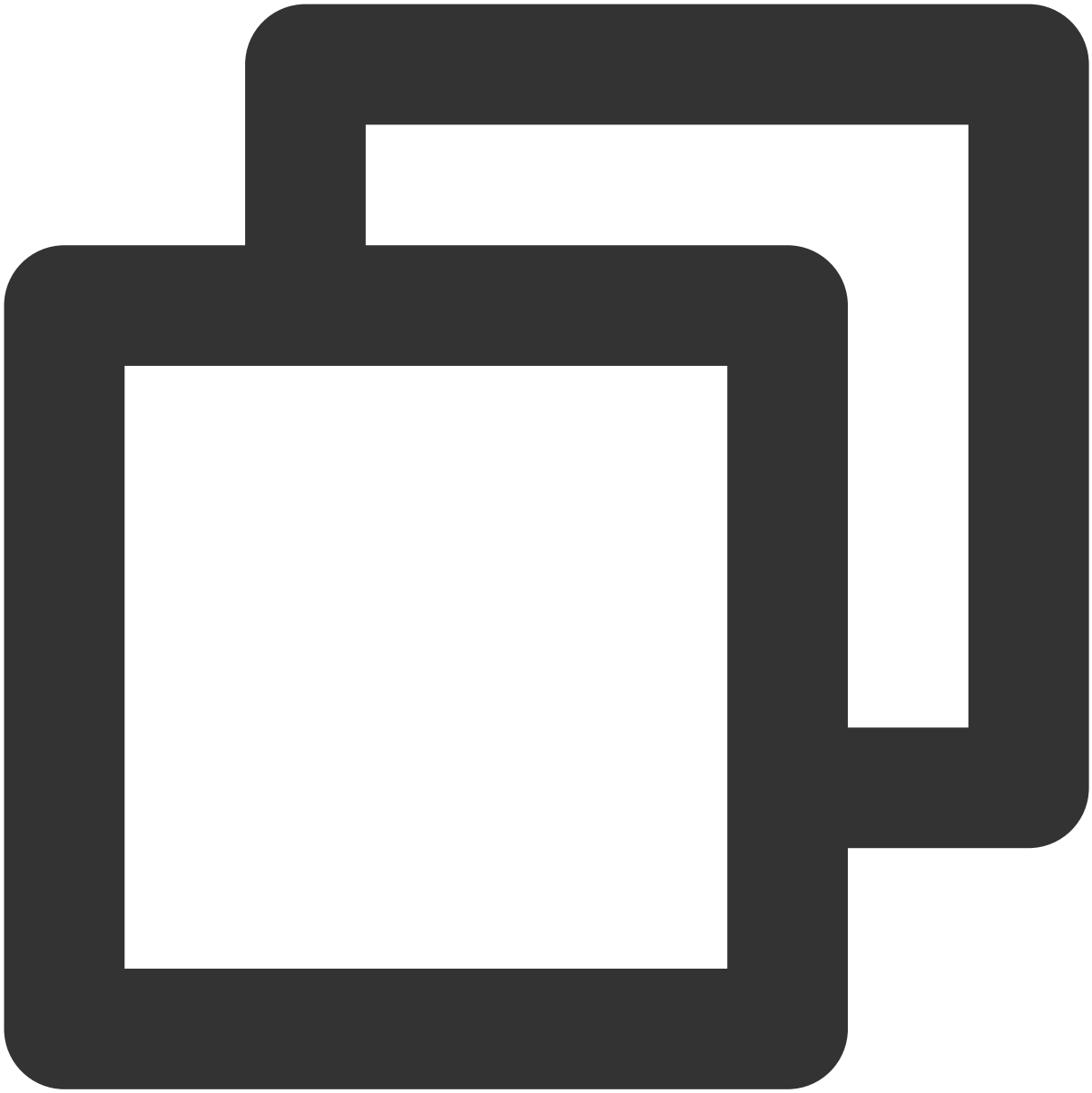
开通直播服务后，可以使用**直播控制台** > **辅助工具** > **地址生成器** 生成推流地址，详细信息请参见 推拉流 URL。



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. **屏幕录制何时自动会停止？**

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

2. **采集推流过程中有时屏幕录制会自动停止问题？**

通常是因为设置的推流分辨率过高时在做横竖屏切换过程中容易出现。ReplayKit2 的直播扩展目前是有50M的内存使用限制，超过此限制系统会直接杀死扩展进程，因此 ReplayKit2 上建议推流分辨率不高于720P。

3. **iPhoneX 手机的兼容性与画面变形问题？**

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

直播播放
iOS
标准直播拉流

最近更新时间：2024-01-13 15:53:49

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
HLS	成熟度高、高并发无压力	需集成 SDK 才能播放	3s - 5s
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、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

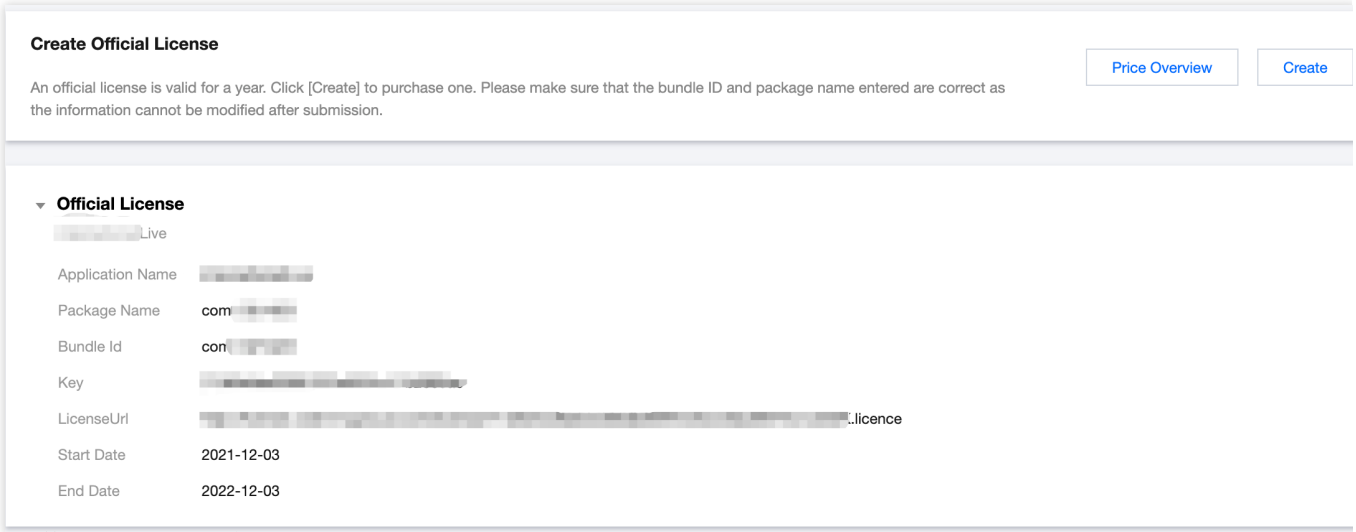
对接攻略

1. 下载 SDK 开发包

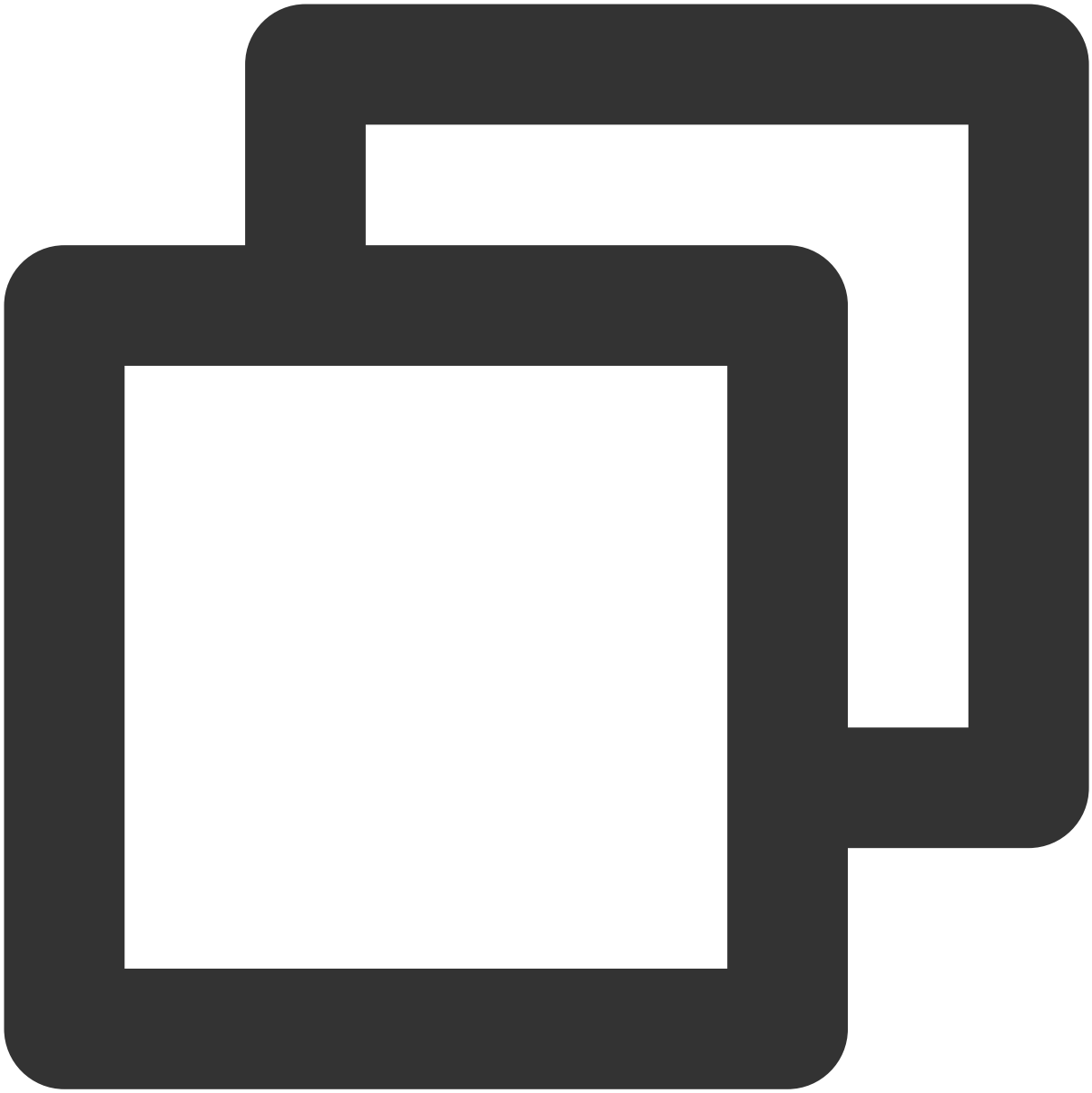
下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

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:(NSD
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

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

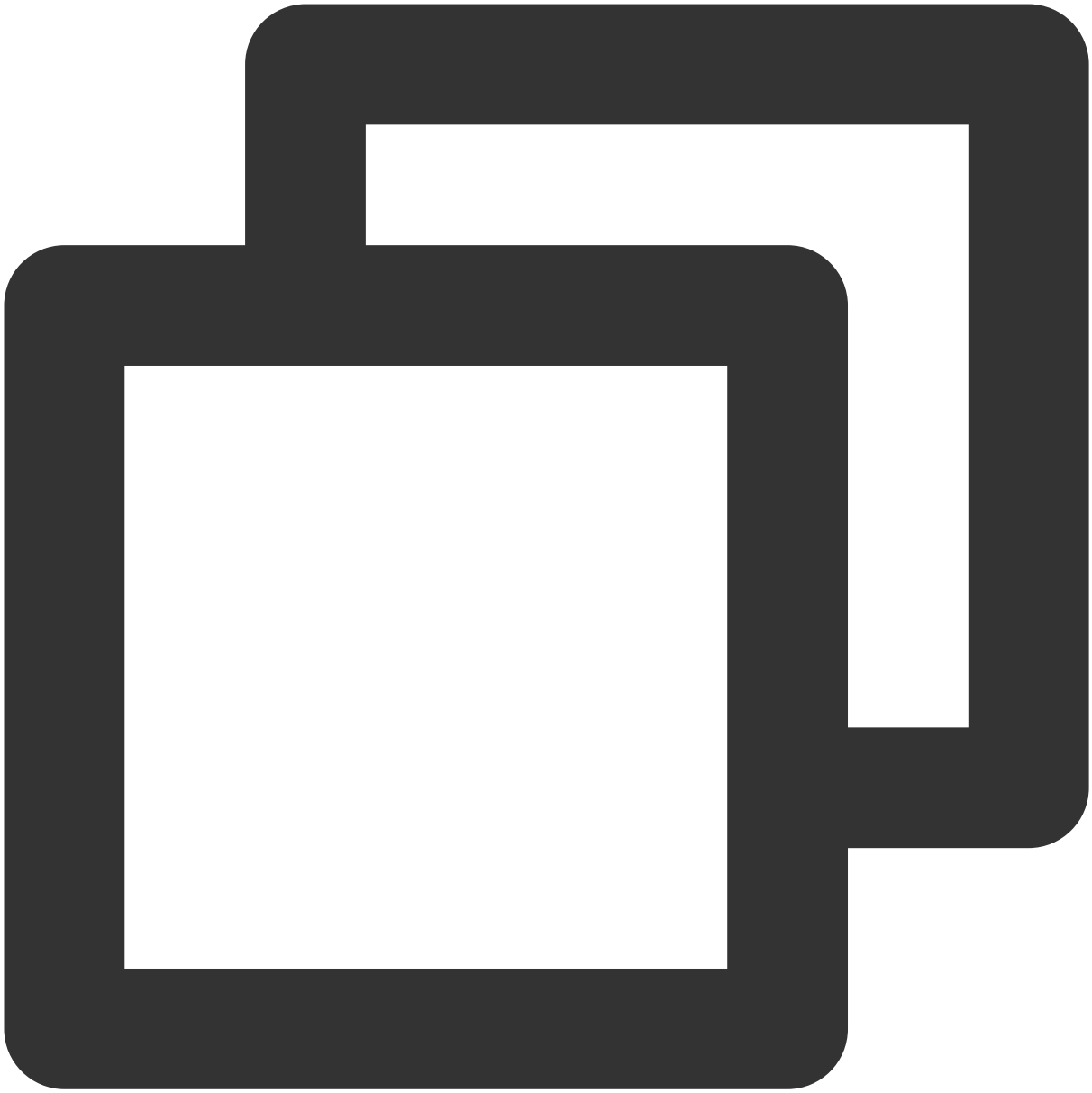
```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

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

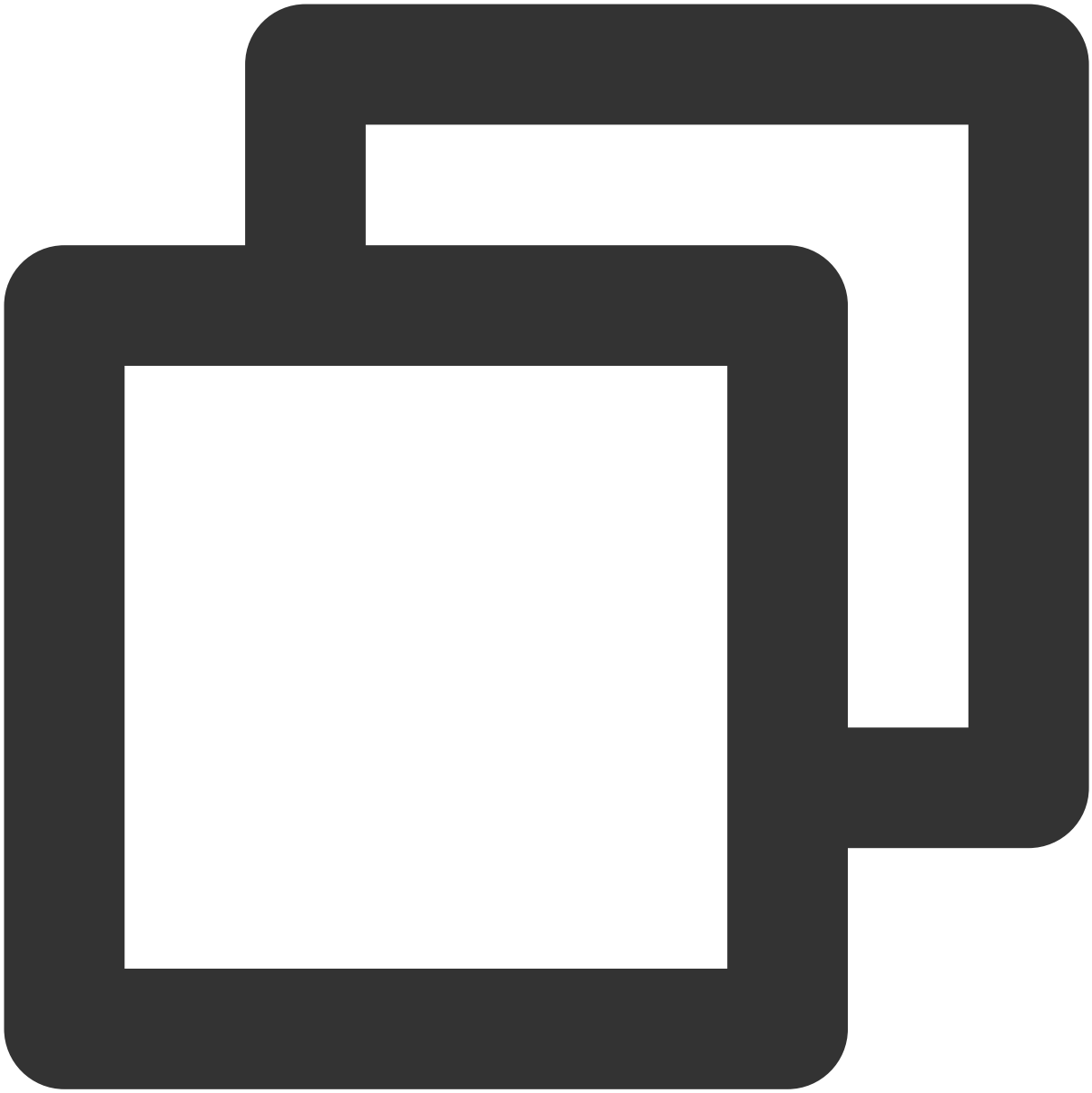
视频云 SDK 中的 V2TXLivePlayer 模块负责实现直播播放功能。



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

4. 渲染 View

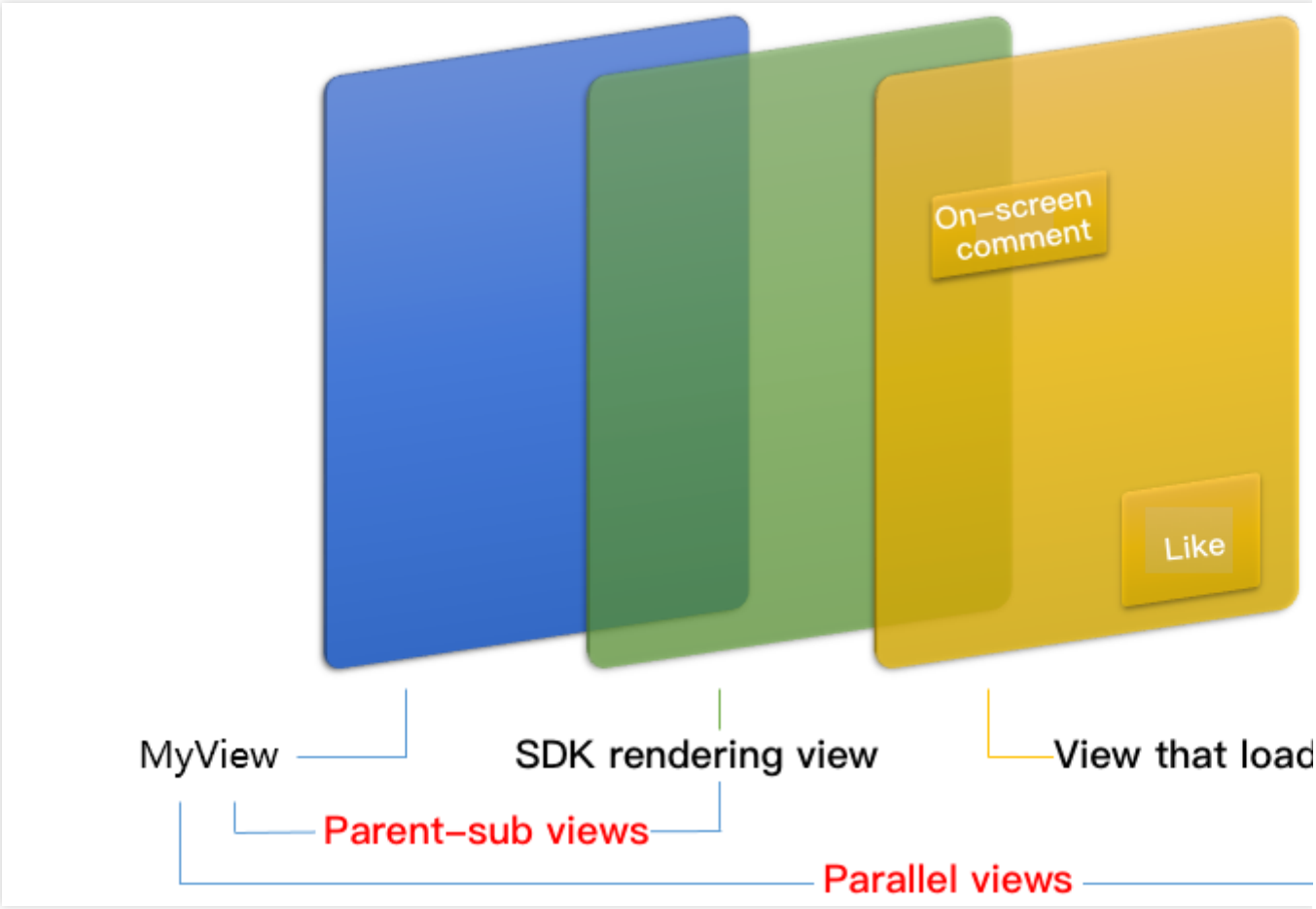
接下来我们要给播放器的视频画面找个地方来显示，iOS 系统中使用 view 作为基本的界面渲染单位，所以您只需要准备一个 view 并调整好布局就可以了。



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

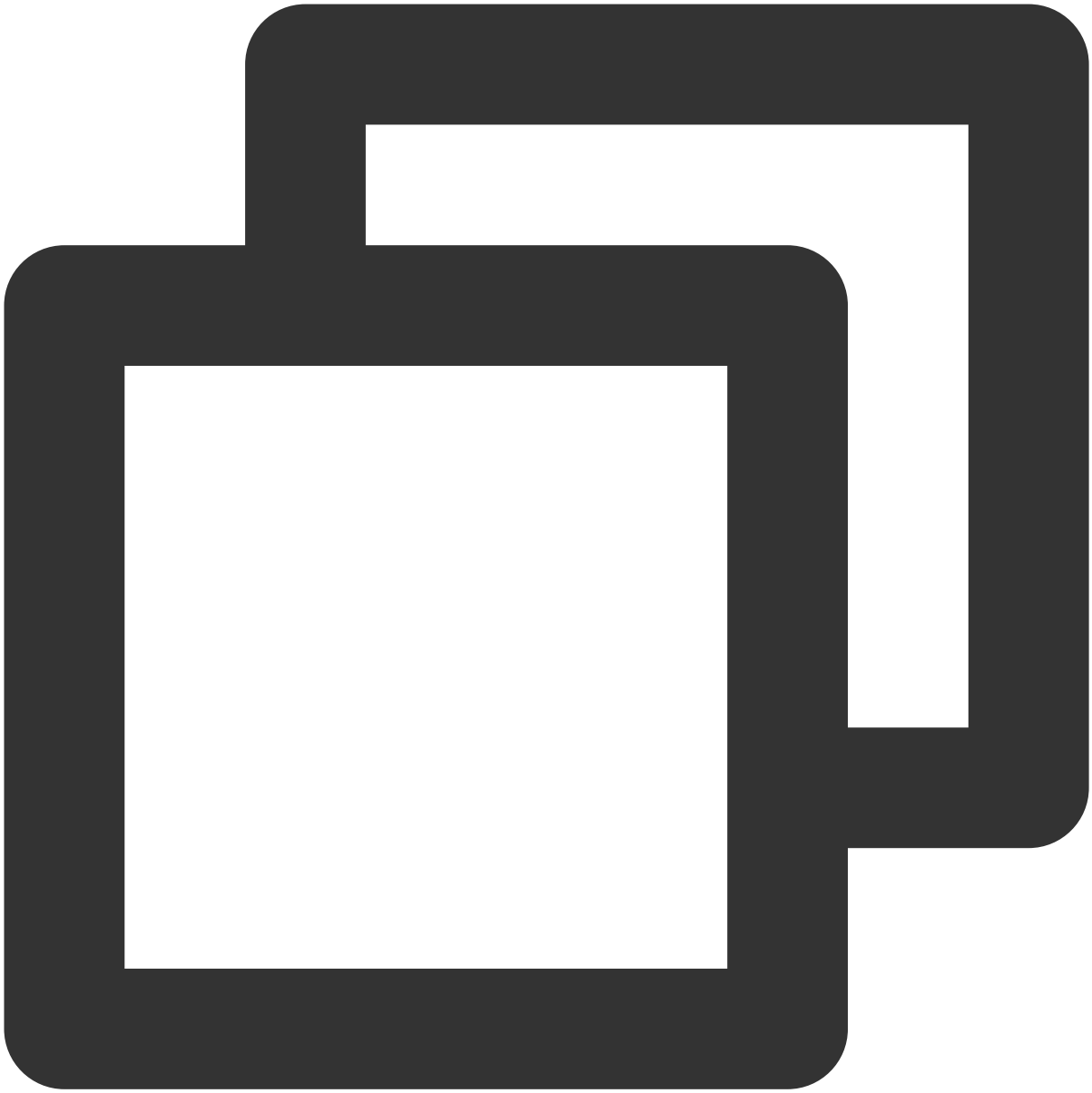
内部原理上，播放器并不是直接把画面渲染到您提供的 view （示例代码中的 `_myView` ）上，而是在这个 view 之上创建一个用于 OpenGL 渲染的子视图（subView）。

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。



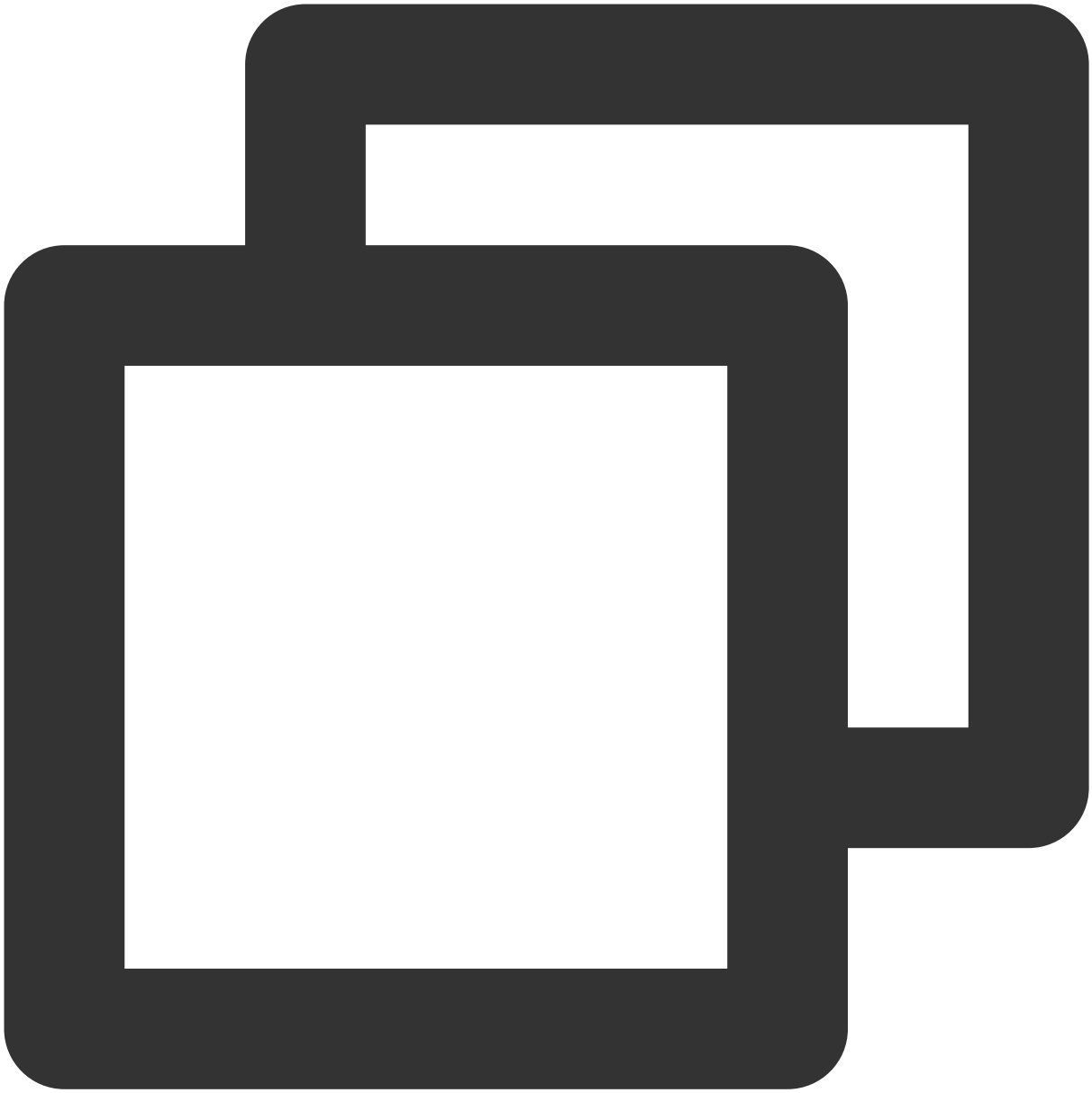
如何做动画？

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 startPlay:url];
```

6. 画面调整

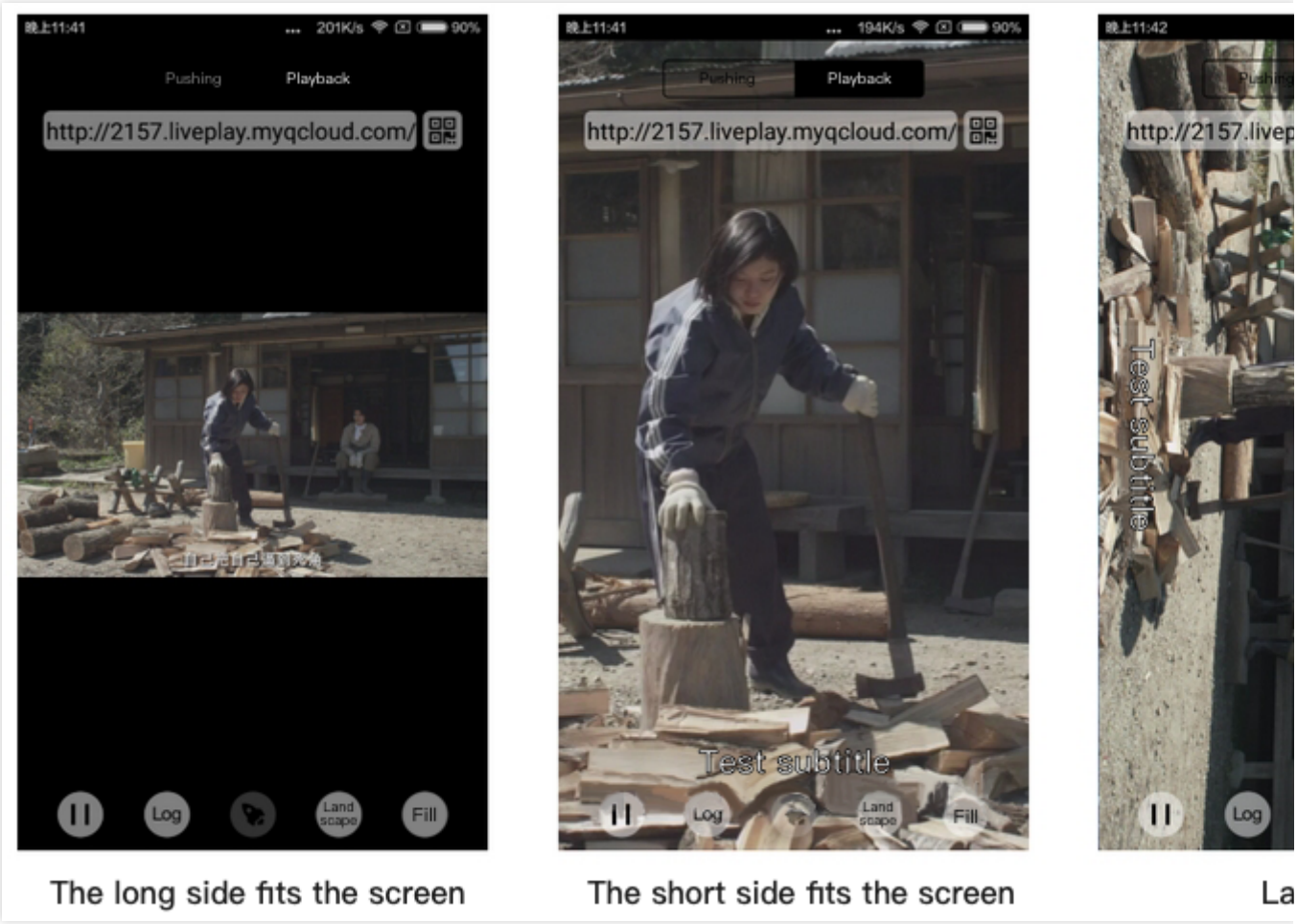
setRenderFillMode：铺满 or 适应

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

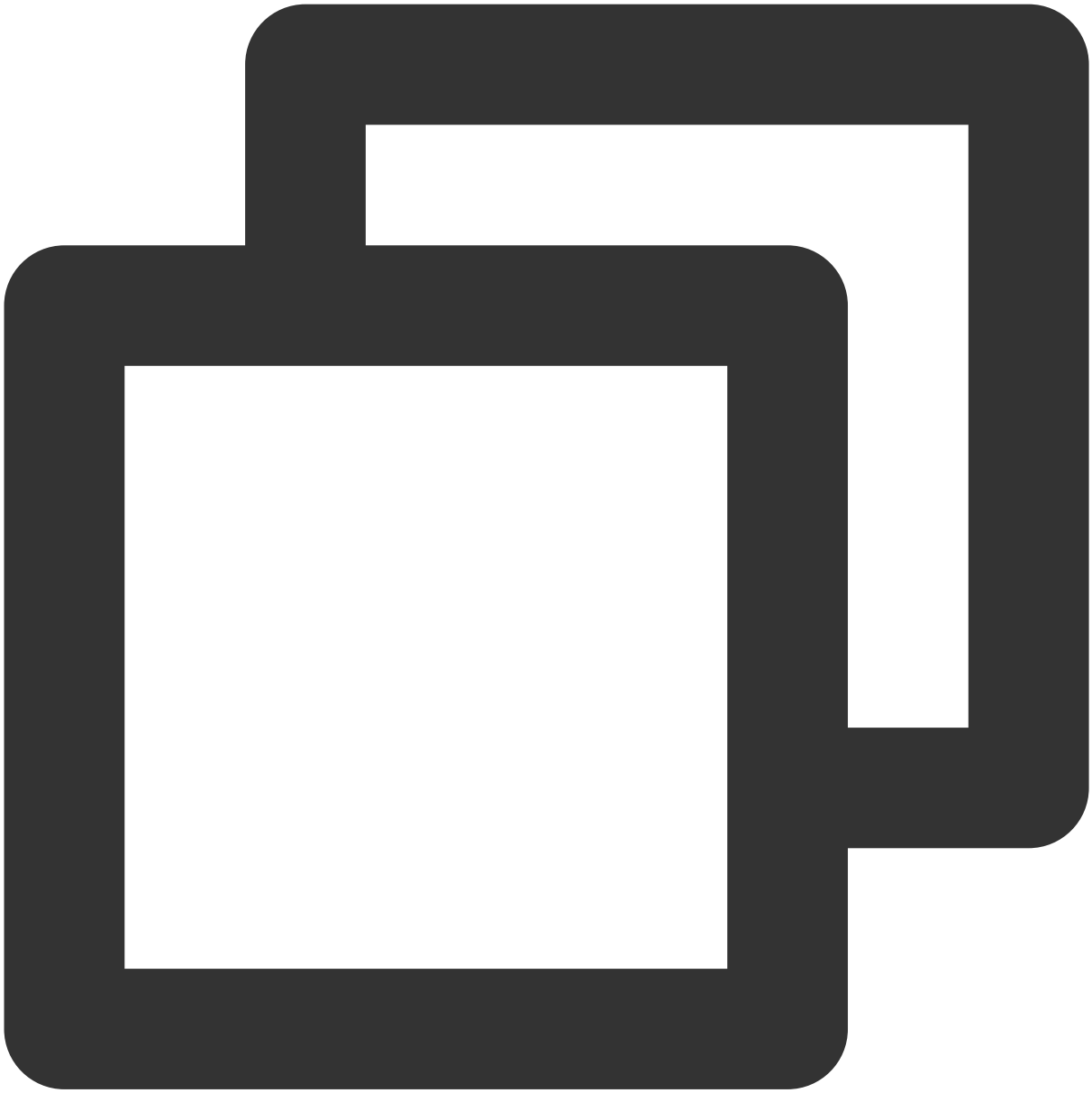
setRenderRotation：视频画面顺时针旋转角度

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



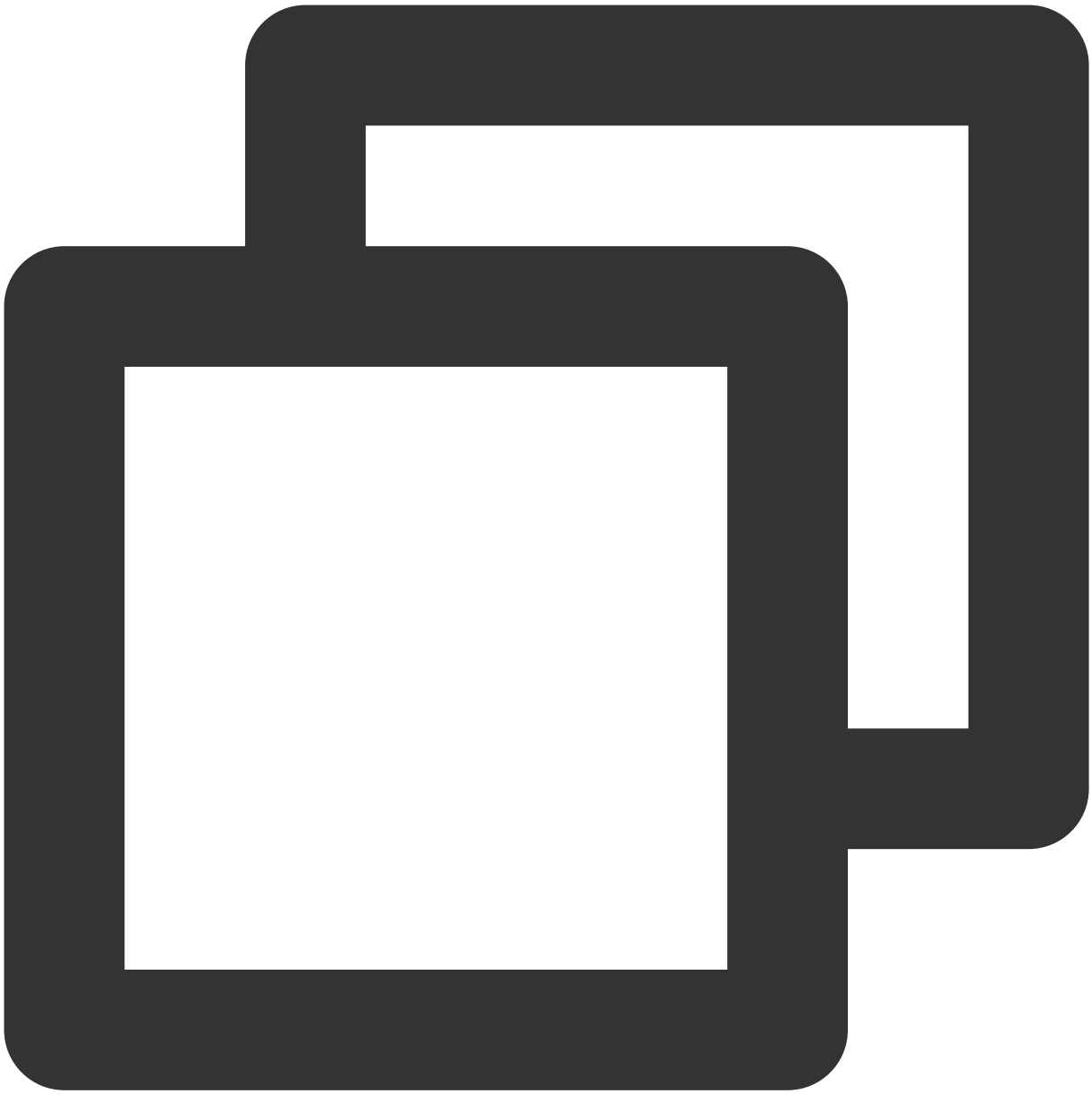
7. 暂停播放

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];
```

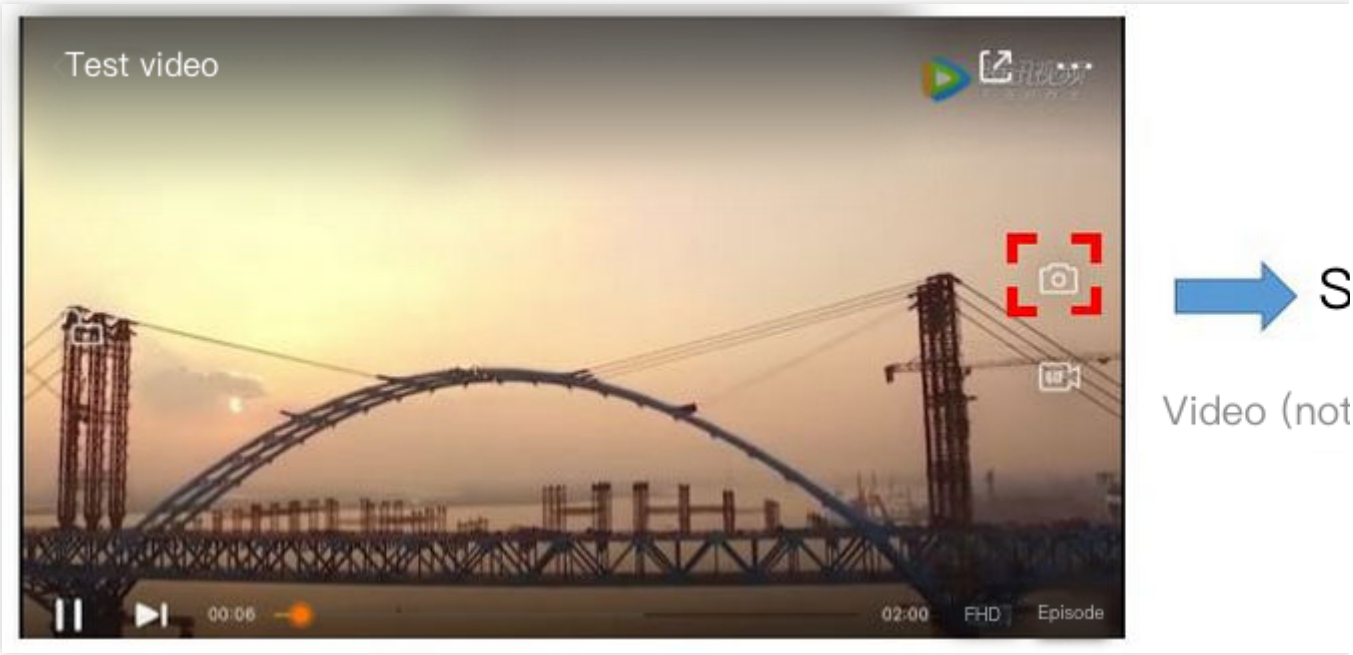
8. 结束播放

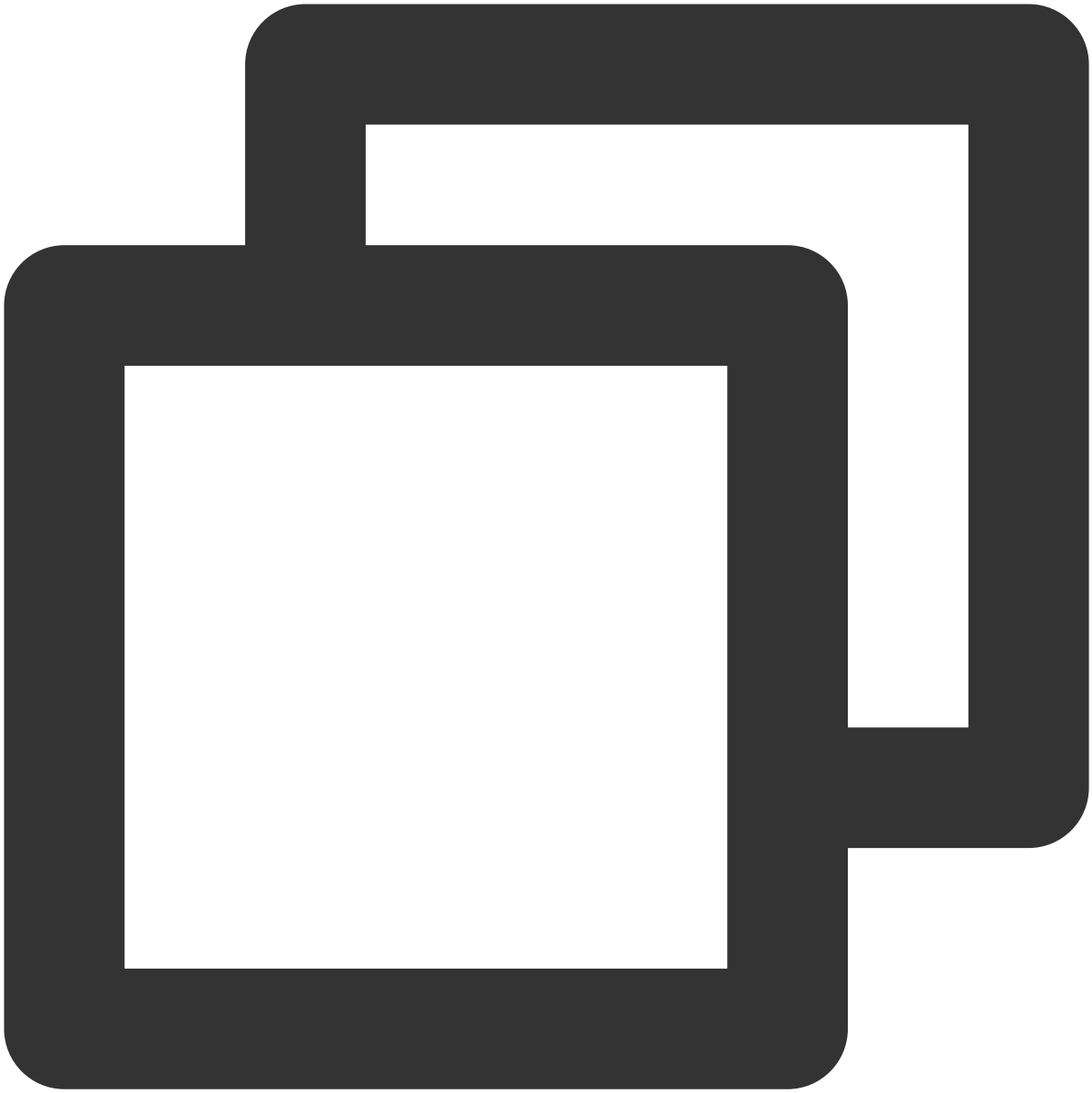


```
// 停止播放
[_txLivePlayer stopPlay];
```

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。


Test video
00:06
02:00
FHD
Episode
S
Video (not



```
...
[_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];
        });
    }
```

```
}
```

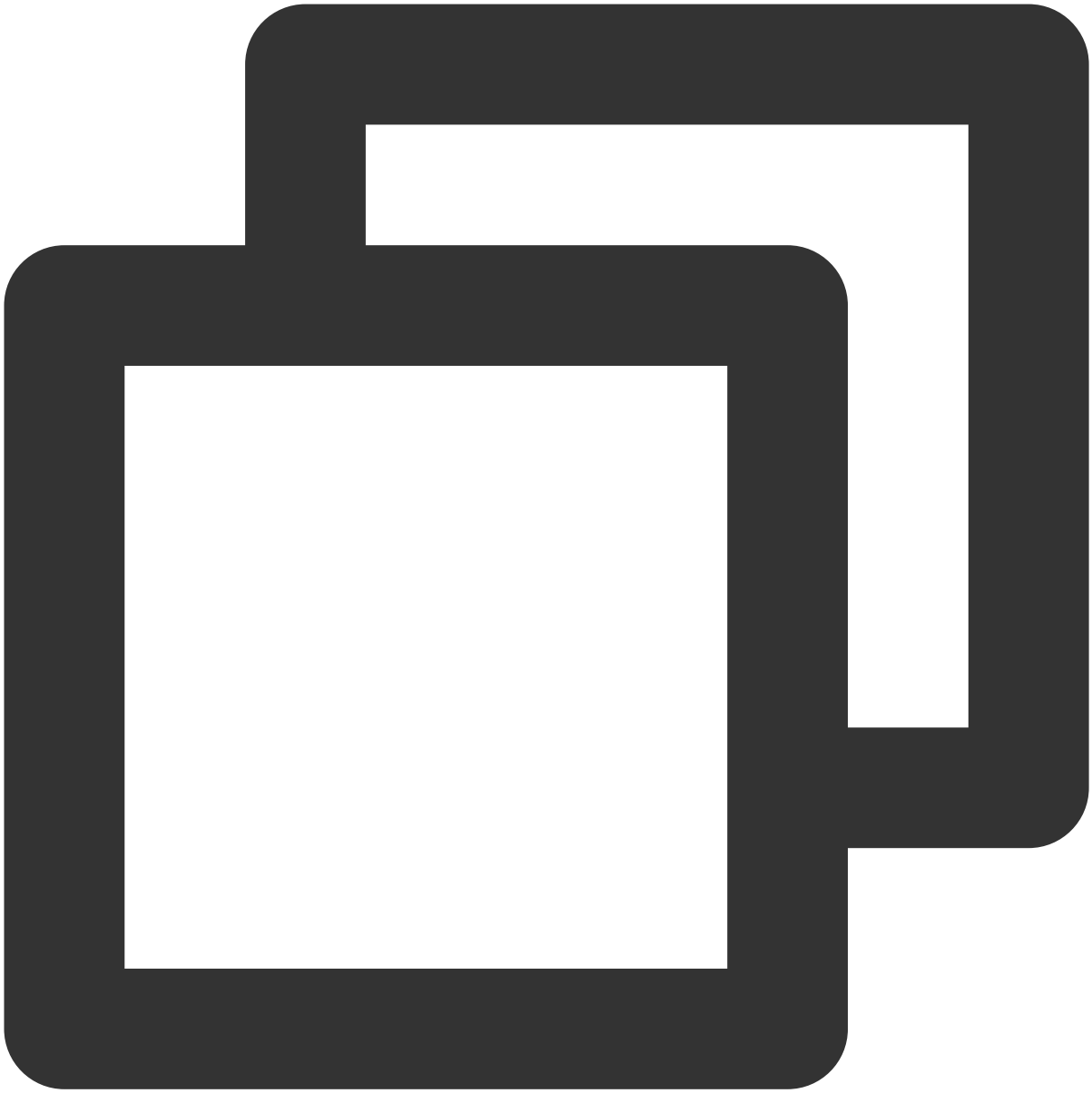
延时调节

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：2024-01-13 15:53:49

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极致直播观看体验。 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。

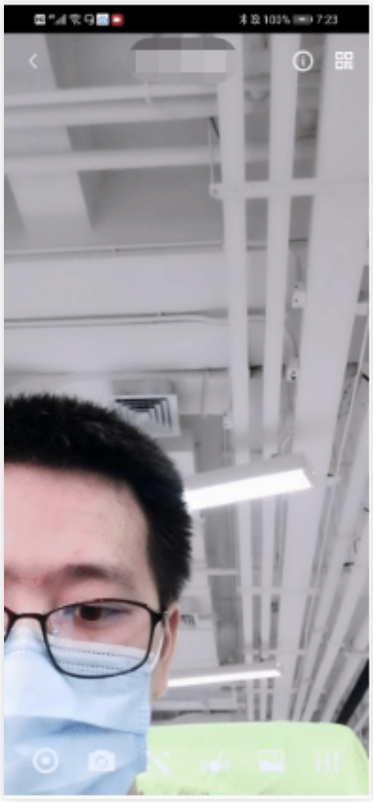
体验快直播

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
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。

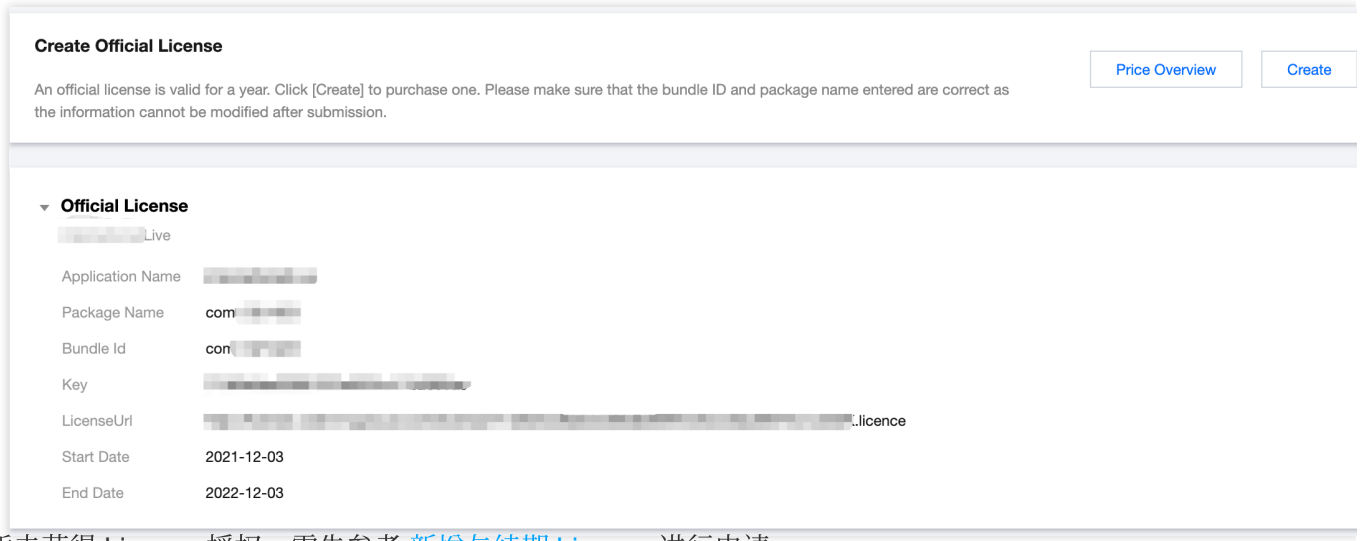
1. 下载 SDK

可以在 SDK 下载 页面选择全功能版下载。

2. 给 SDK 配置 License 授权

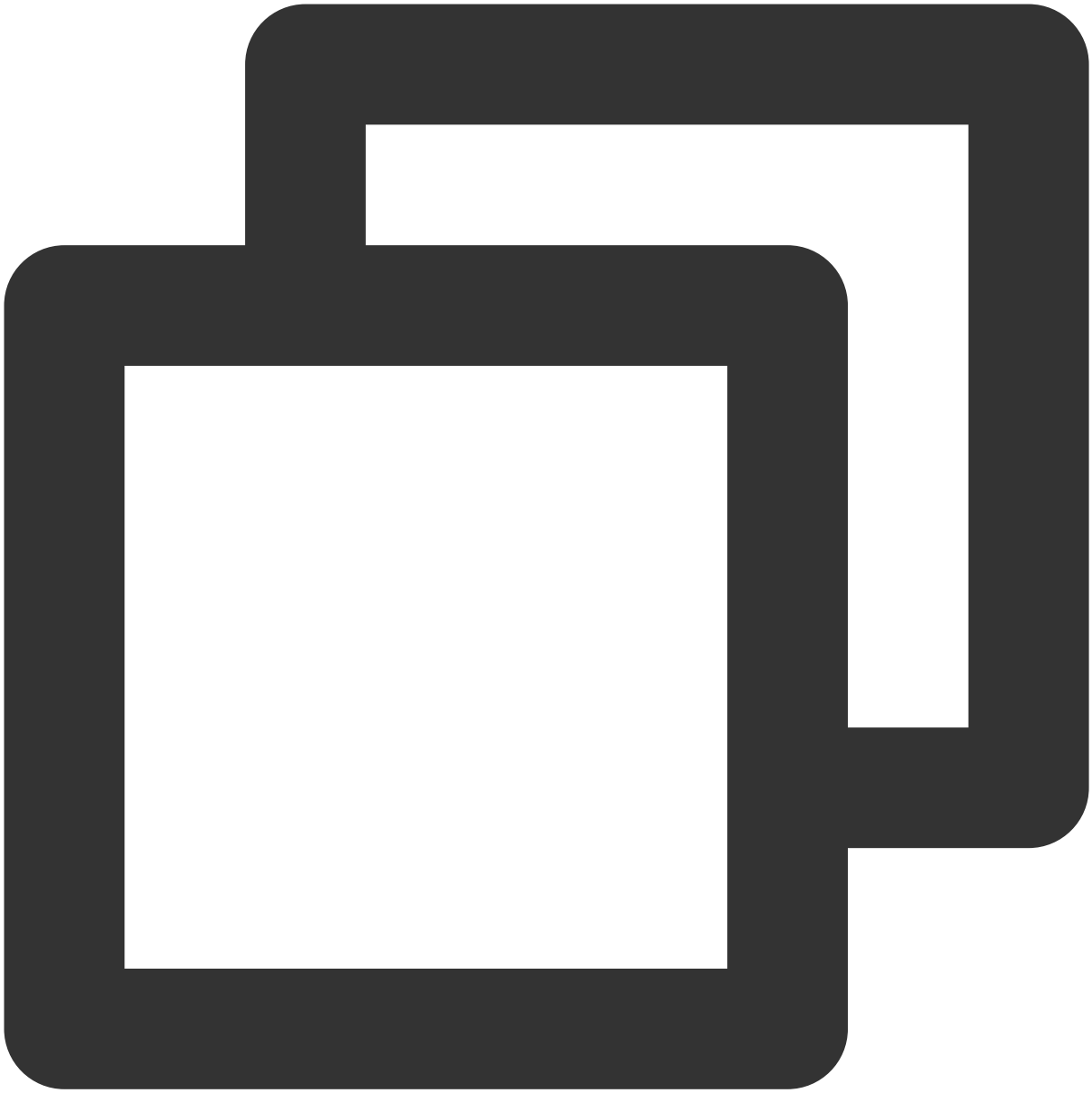
1. 获取 License 授权：

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 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中）进行如下设置：



```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    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 setEnvironment:@"GDPR"];// 设置环境
    [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
```

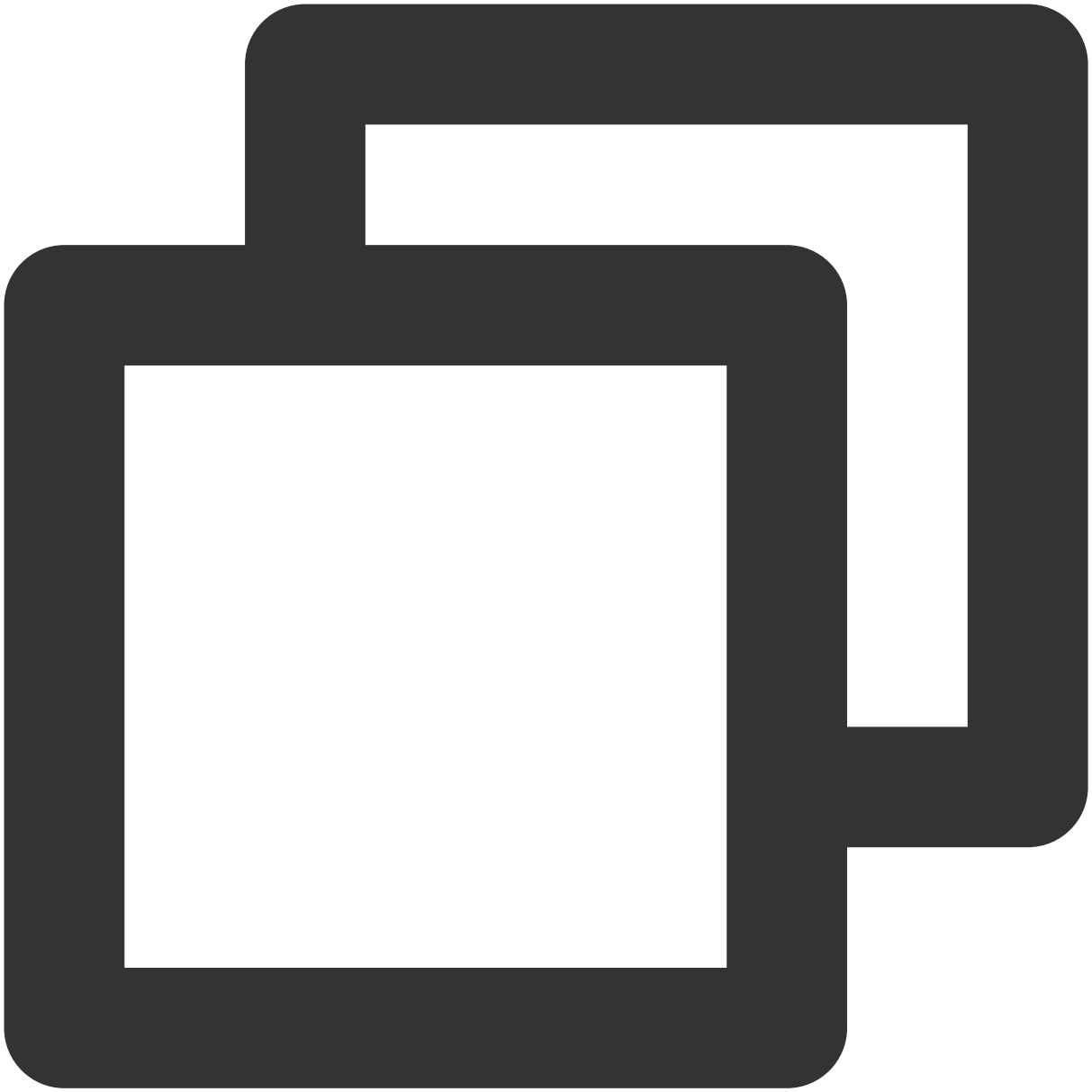
注意：

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。

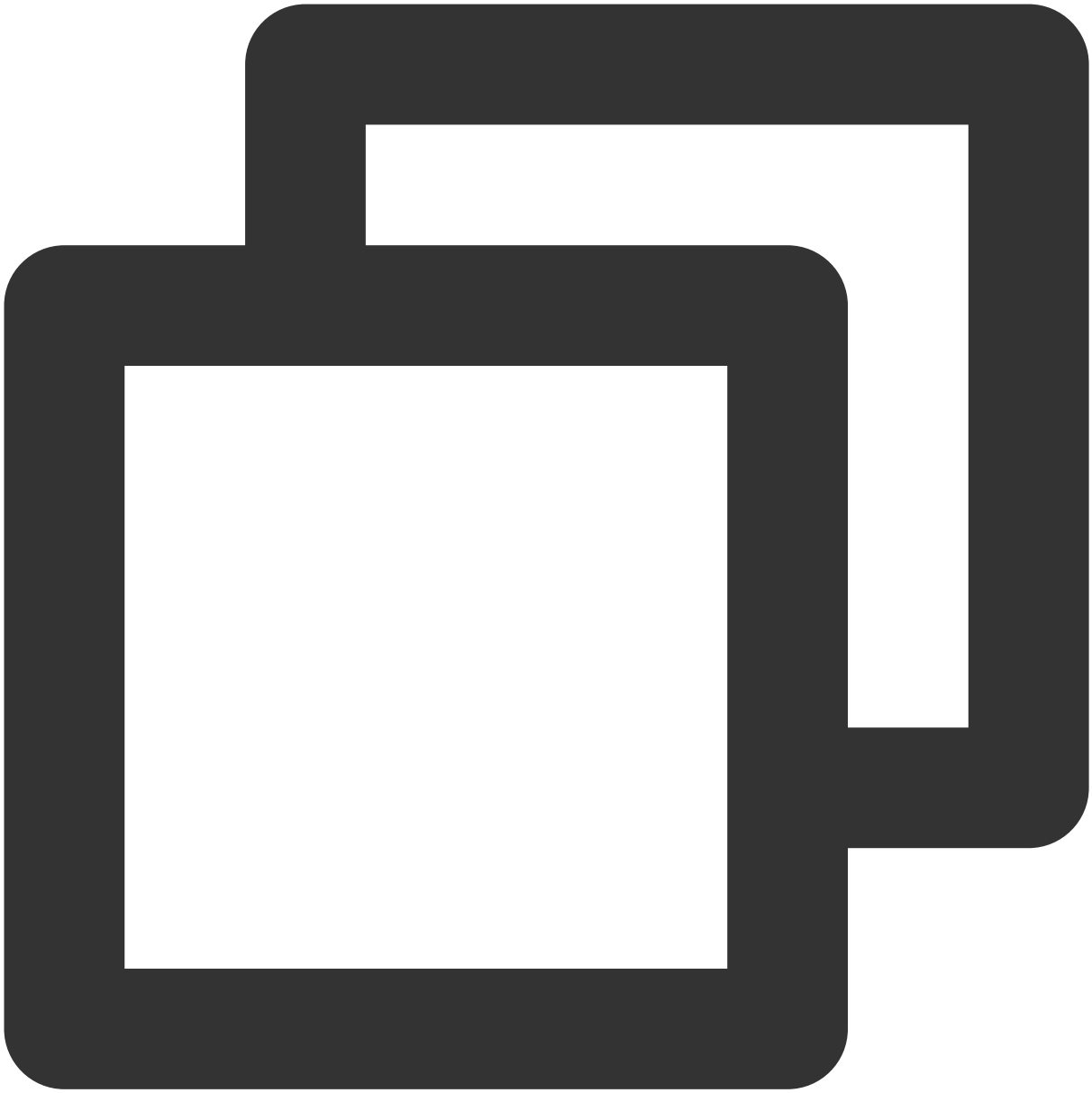
4. 实现快直播播放

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）：

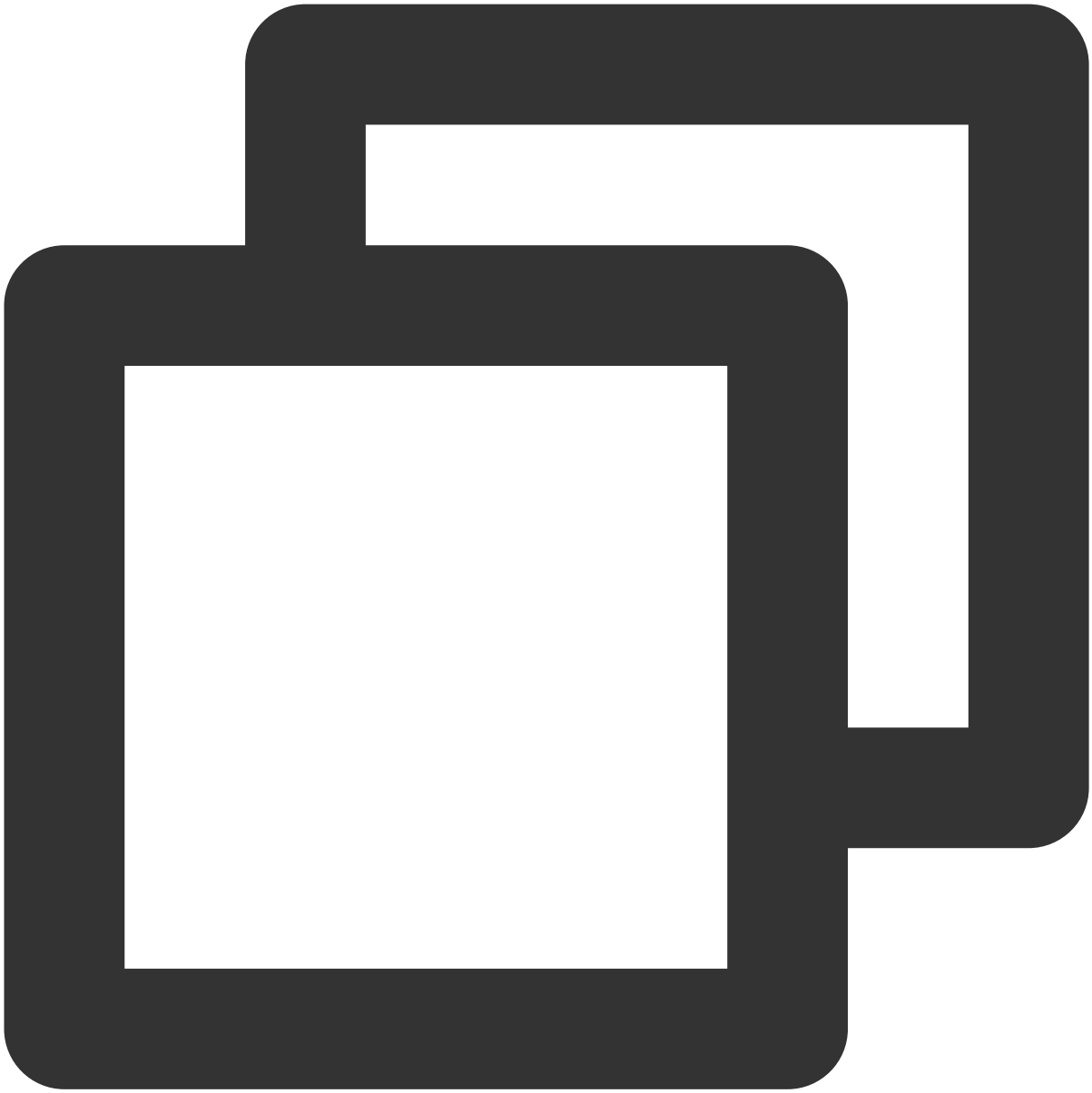
示例代码

Android

iOS



```
// 创建一个 V2TXLivePlayer 对象；
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传入低延时协议播放地址，即可开始播放；
player.startPlay("webrtc://{Domain}/{AppName}/{StreamName}");
```



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

Android
标准直播拉流

最近更新时间：2024-01-13 15:53:49

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
HLS	成熟度高、高并发无压力	需集成 SDK 才能播放	3s - 5s
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、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

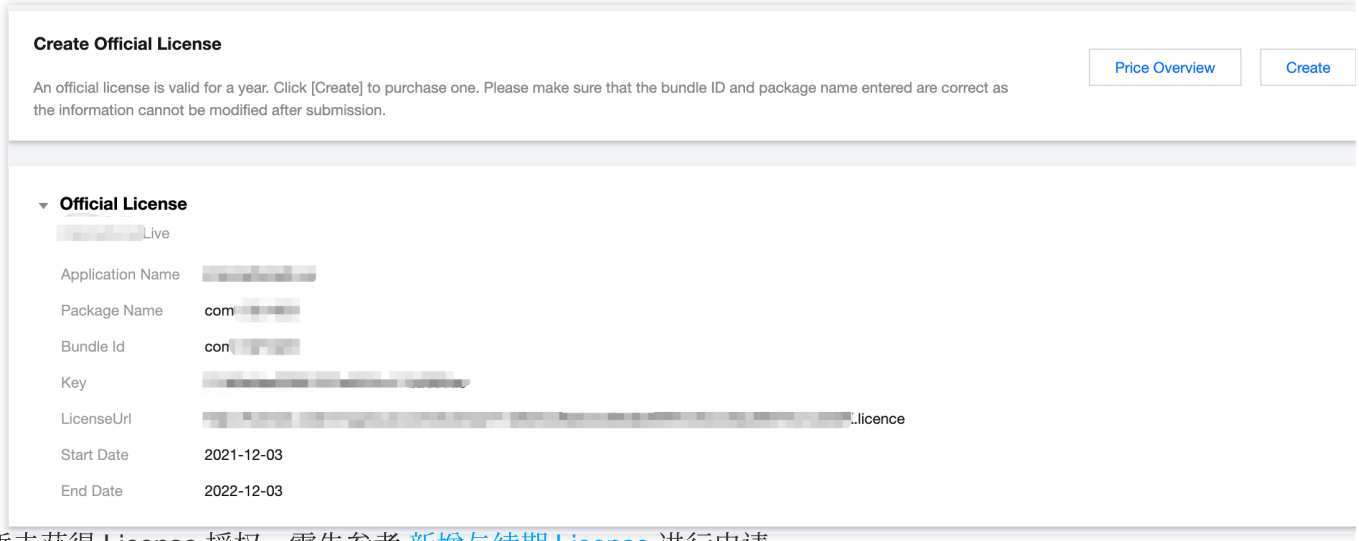
对接攻略

1. 下载 SDK 开发包

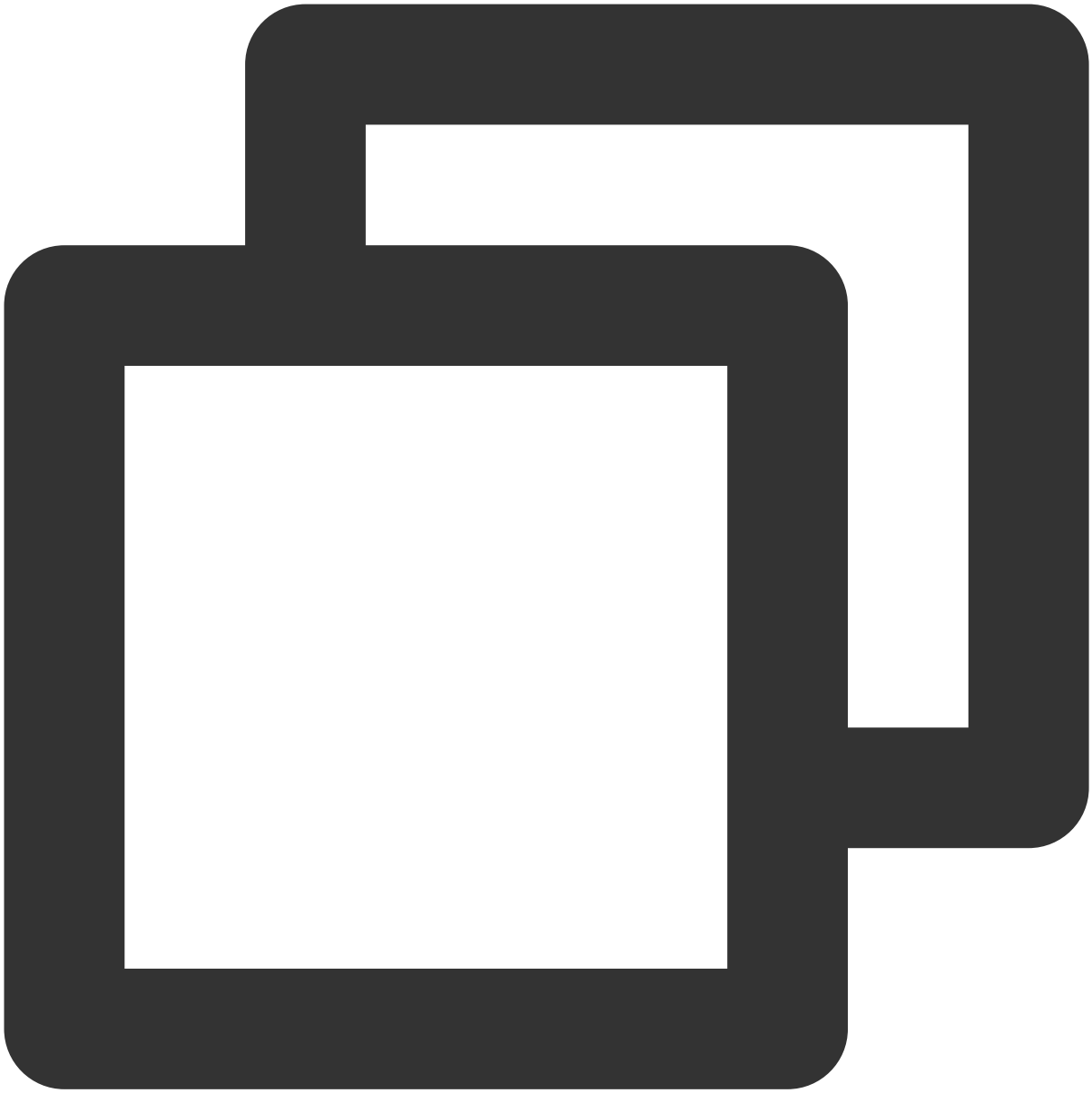
下载 SDK 开发包，并按照 SDK 集成指引 将 SDK 嵌入您的 App 工程中。

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 SDK 相关功能之前（建议在 Application 类中）进行如下设置：



```
public class MyApplication 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.setEnvironment("GDPR");  // 设置环境
 V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
 V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
        @Override
```

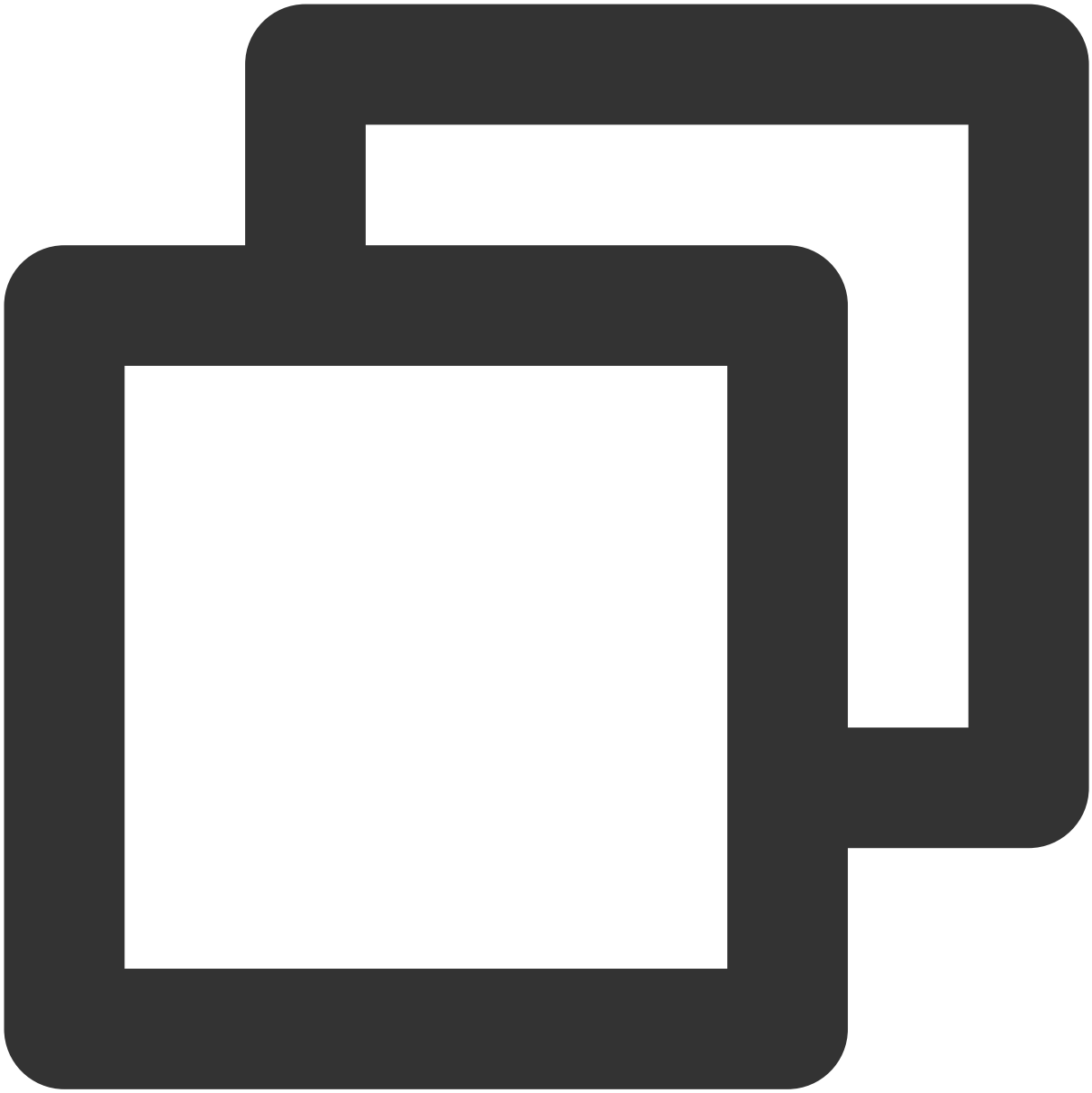
```
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason)
            }
        });
    }
```

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

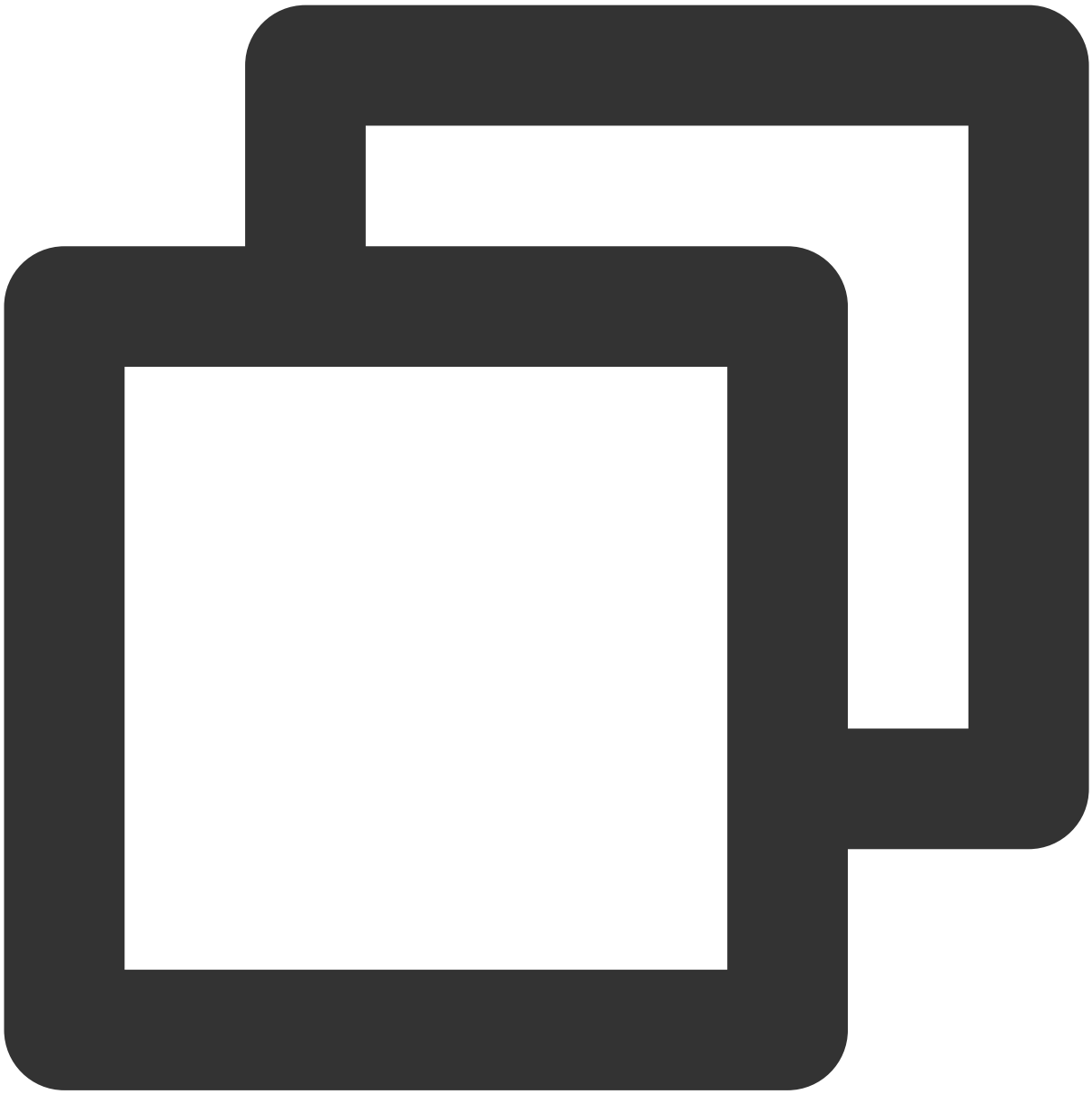
为了能够展示播放器的视频画面，我们第一步要做的就是在布局 xml 文件里加入渲染 View：



```
<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"/>
```

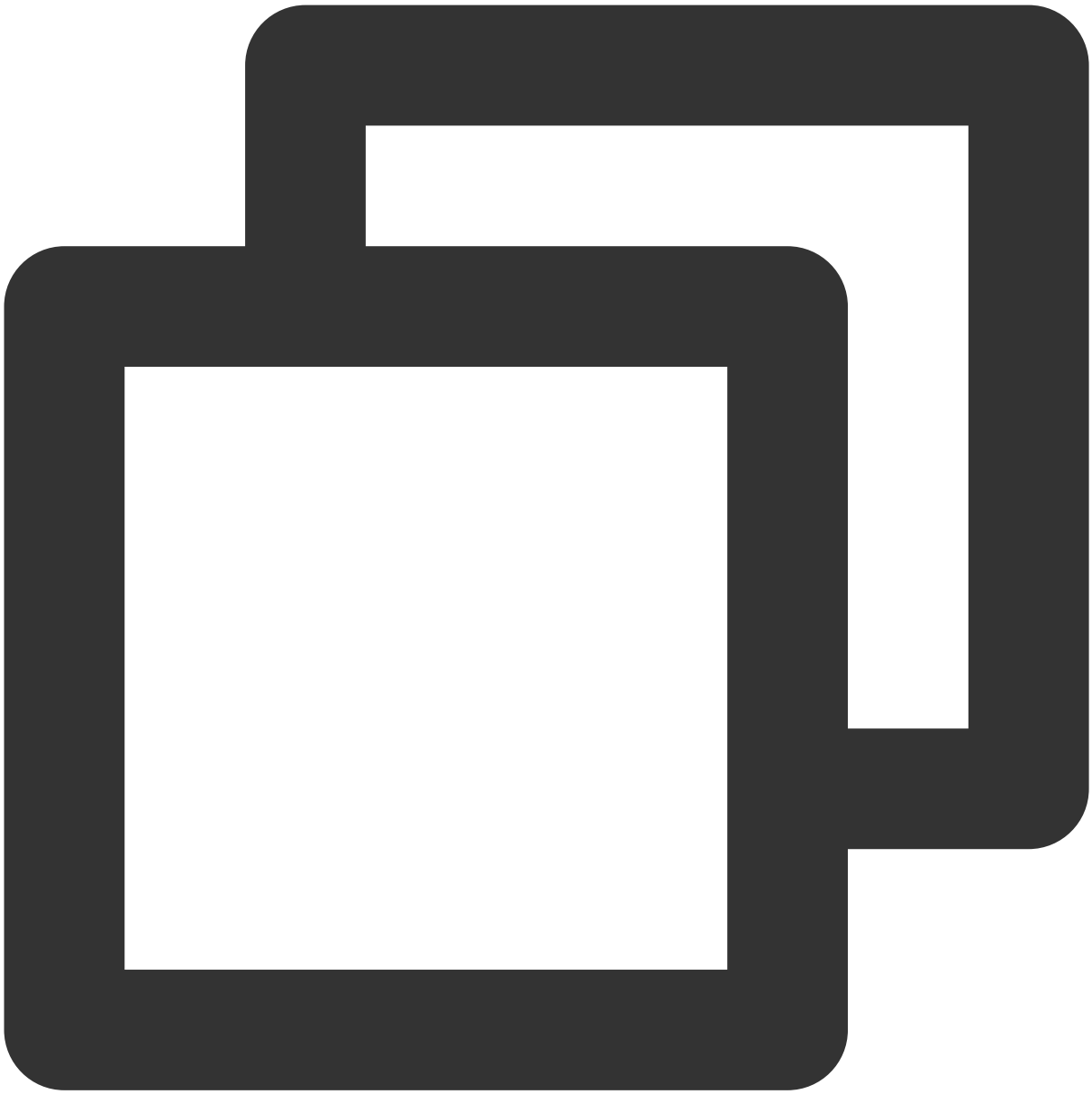
4. 创建 Player

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.startPlay(flvUrl);
```

6. 画面调整

view：大小和位置

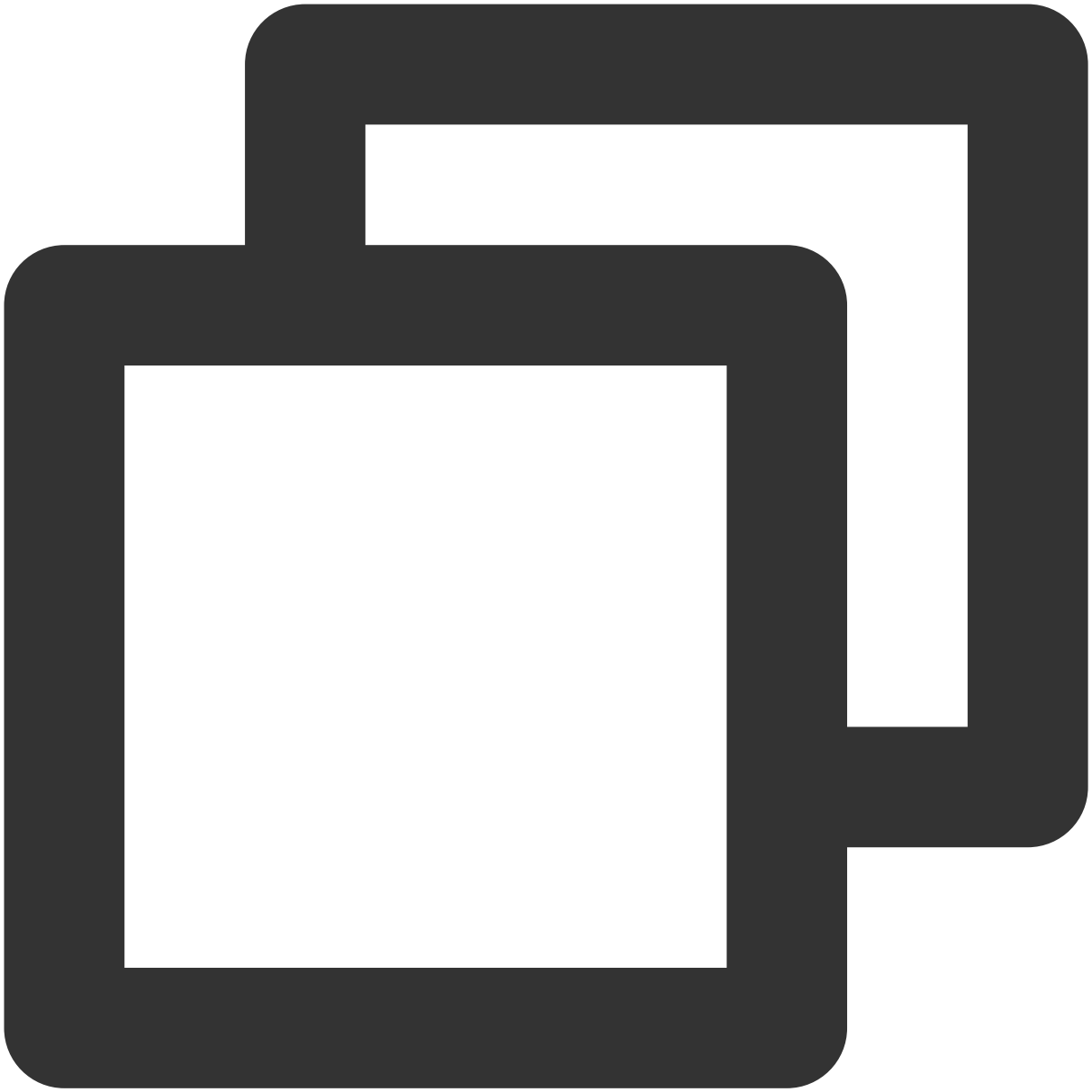
如需修改画面的大小及位置，直接调整 step3 中添加的 `video_view` 控件的大小和位置即可。

setRenderFillMode：铺满 or 适应

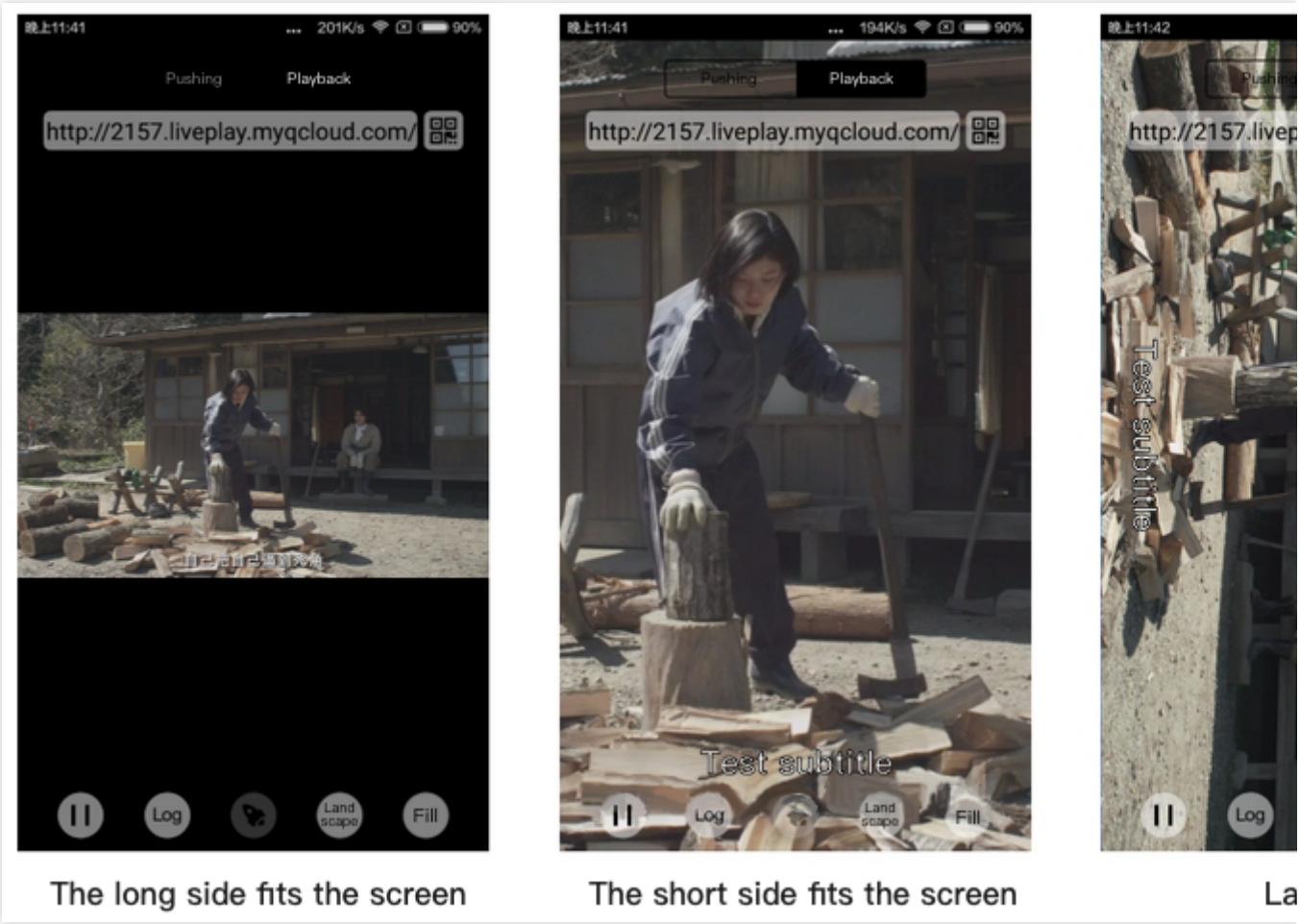
可选值	含义
V2TXLiveFillModeFill	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全
V2TXLiveFillModeFit	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边

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

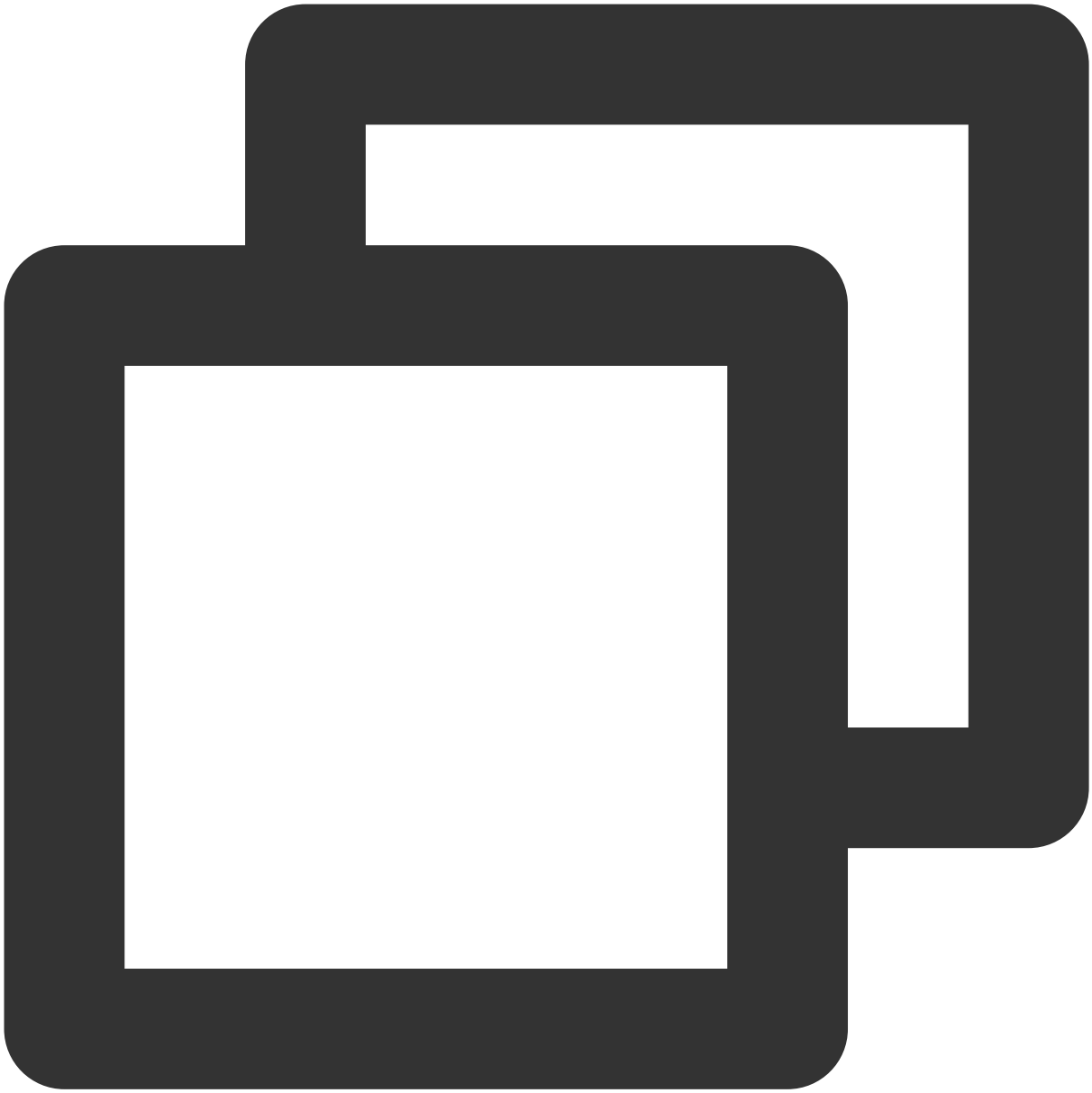


```
// 设置填充模式
mLivePlayer.setRenderFillMode(V2TXLiveFillModeFit);
// 设置画面渲染方向
mLivePlayer.setRenderRotation(V2TXLiveRotation0);
```



7. 暂停播放

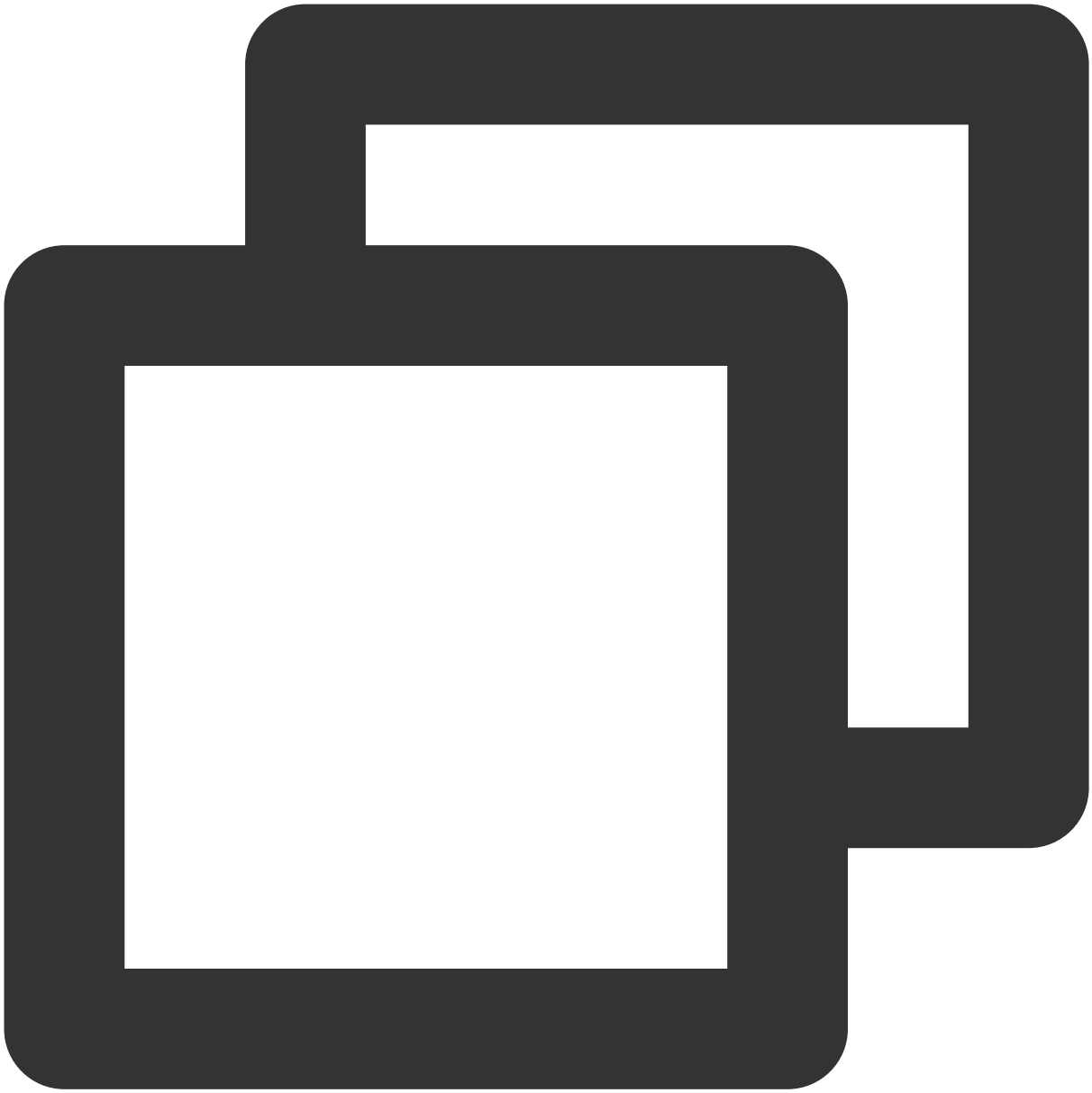
对于直播播放而言，并没有真正意义上的暂停，所谓的直播暂停，只是**画面冻结**和**关闭声音**，而云端的视频源还在不断地更新着，所以当您调用 resume 的时候，会从最新的时间点开始播放，这是和点播对比的最大不同点（点播播放器的暂停和继续与播放本地视频文件时的表现相同）。



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

8. 结束播放

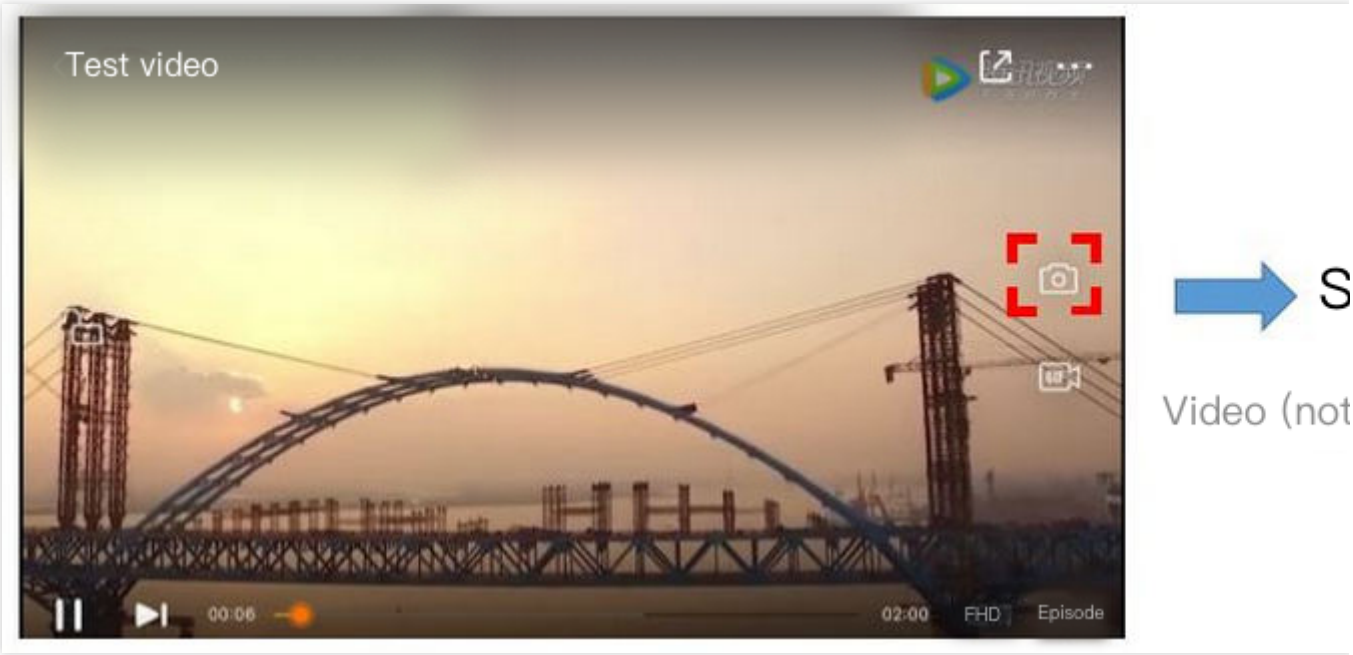
结束播放非常简单，直接调用 `stopPlay` 即可。

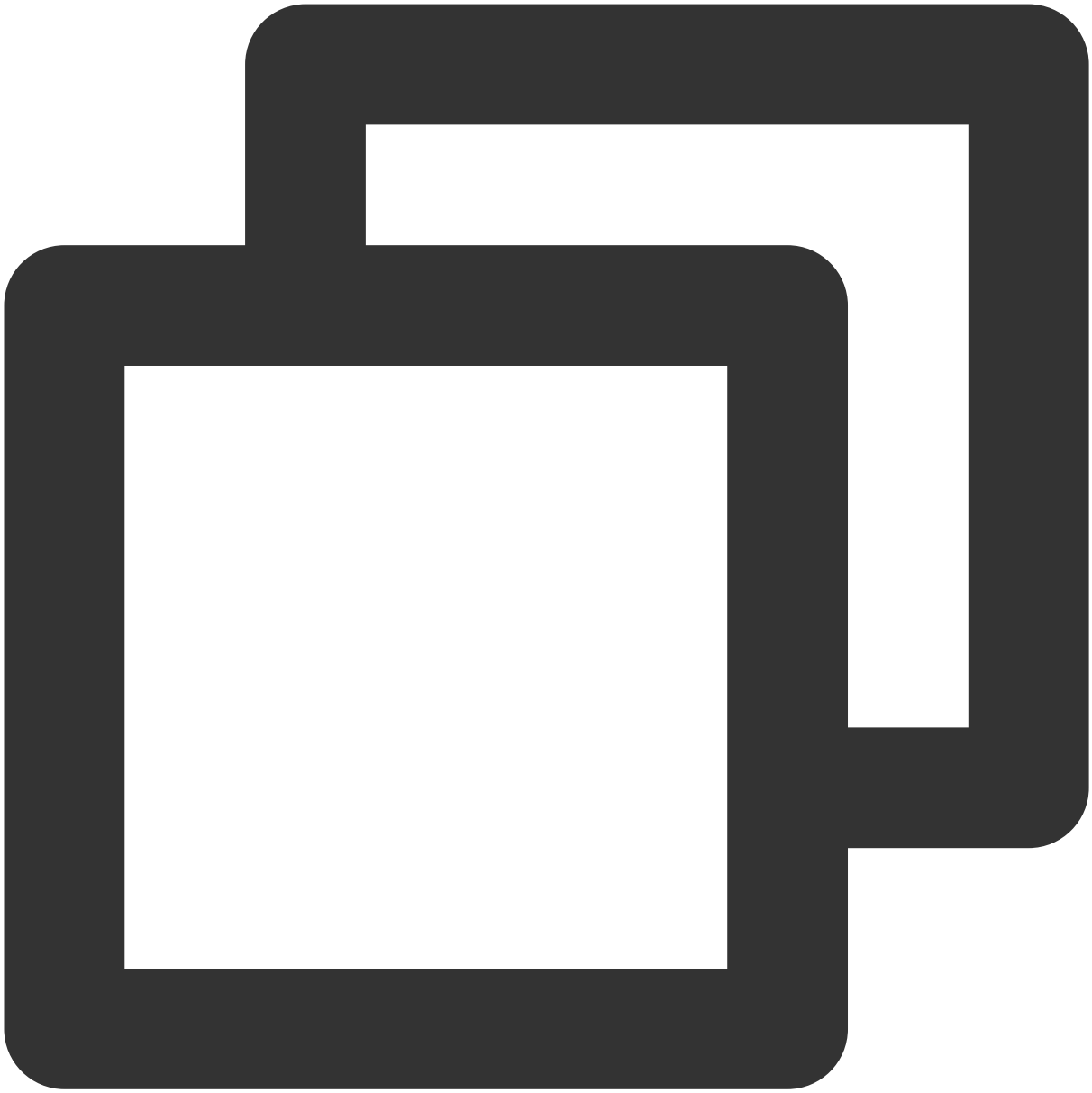


```
mLivePlayer.stopPlay();
```

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。


Test video
00:06
02:00
FHD
Episode
S
Video (not



```
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) {
    }
    ...
}
```

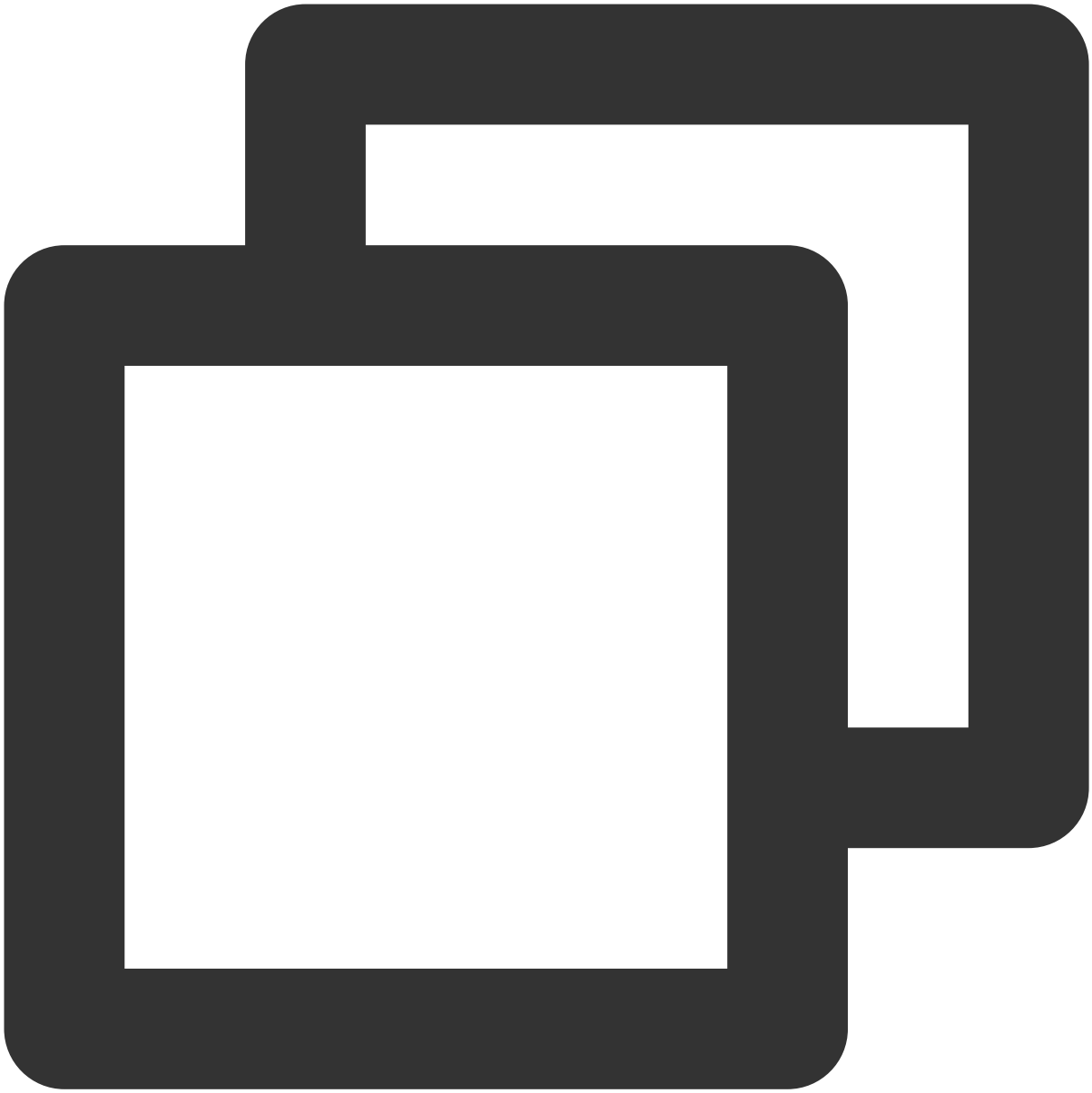
延时调节

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：2024-01-13 15:53:49

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极致直播观看体验。 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。

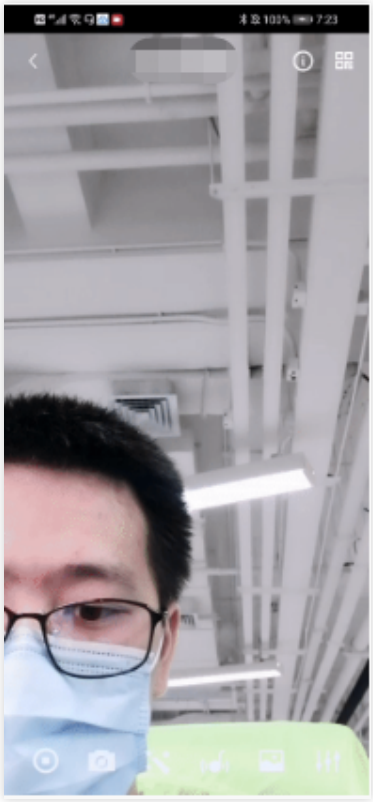
体验快直播

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
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。

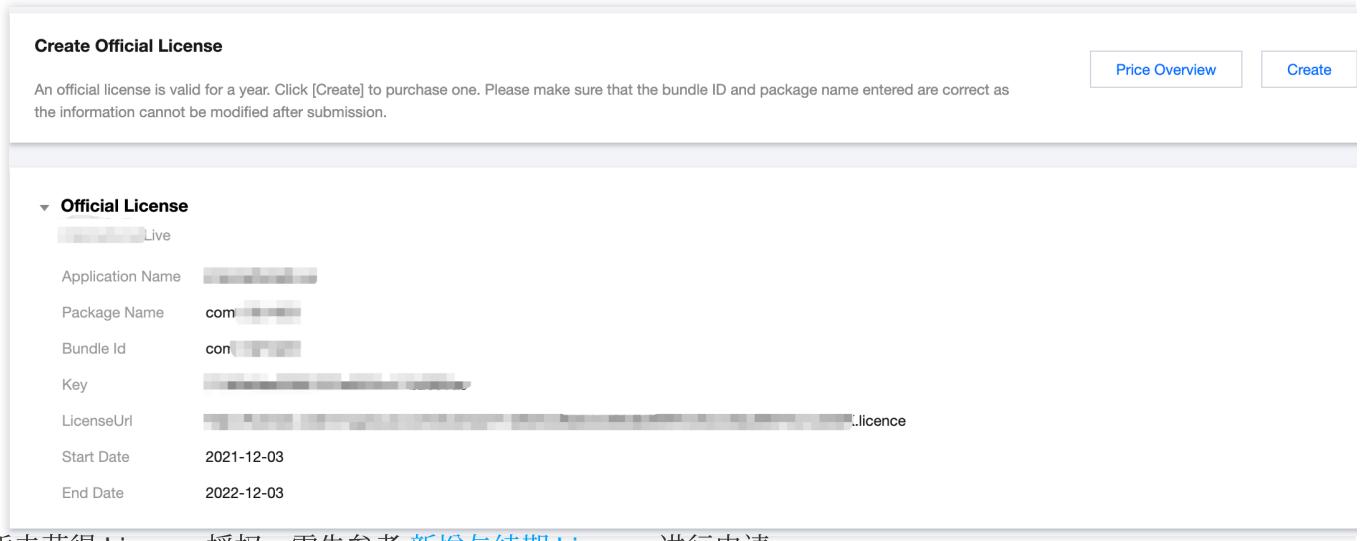
1. 下载 SDK

可以在 SDK 下载 页面选择全功能版下载。

2. 给 SDK 配置 License 授权

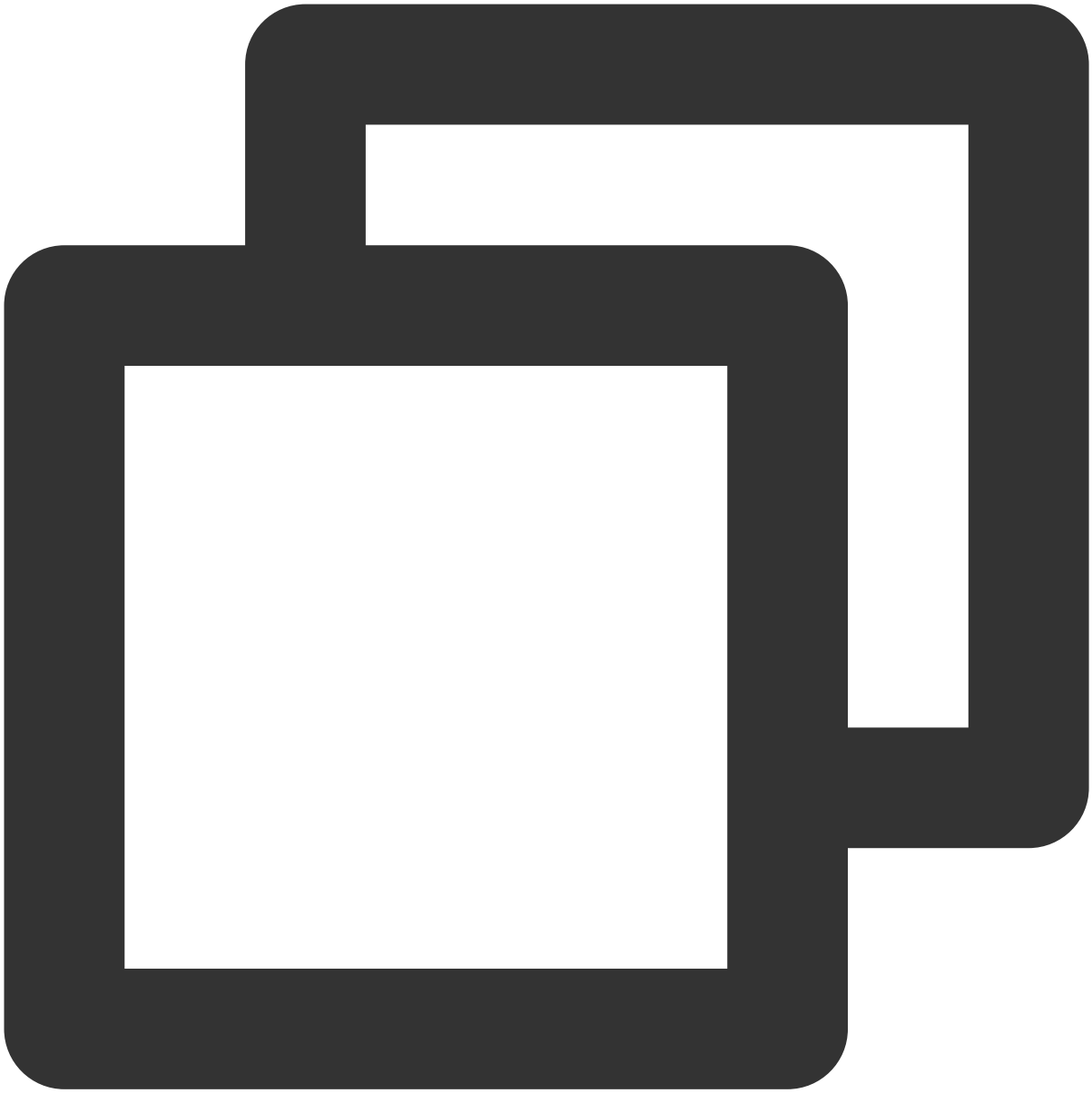
1. 获取 License 授权：

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` 中）进行如下设置：



```
public class MyApplication 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.setEnvironment("GDPR");  // 设置环境
    V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
```

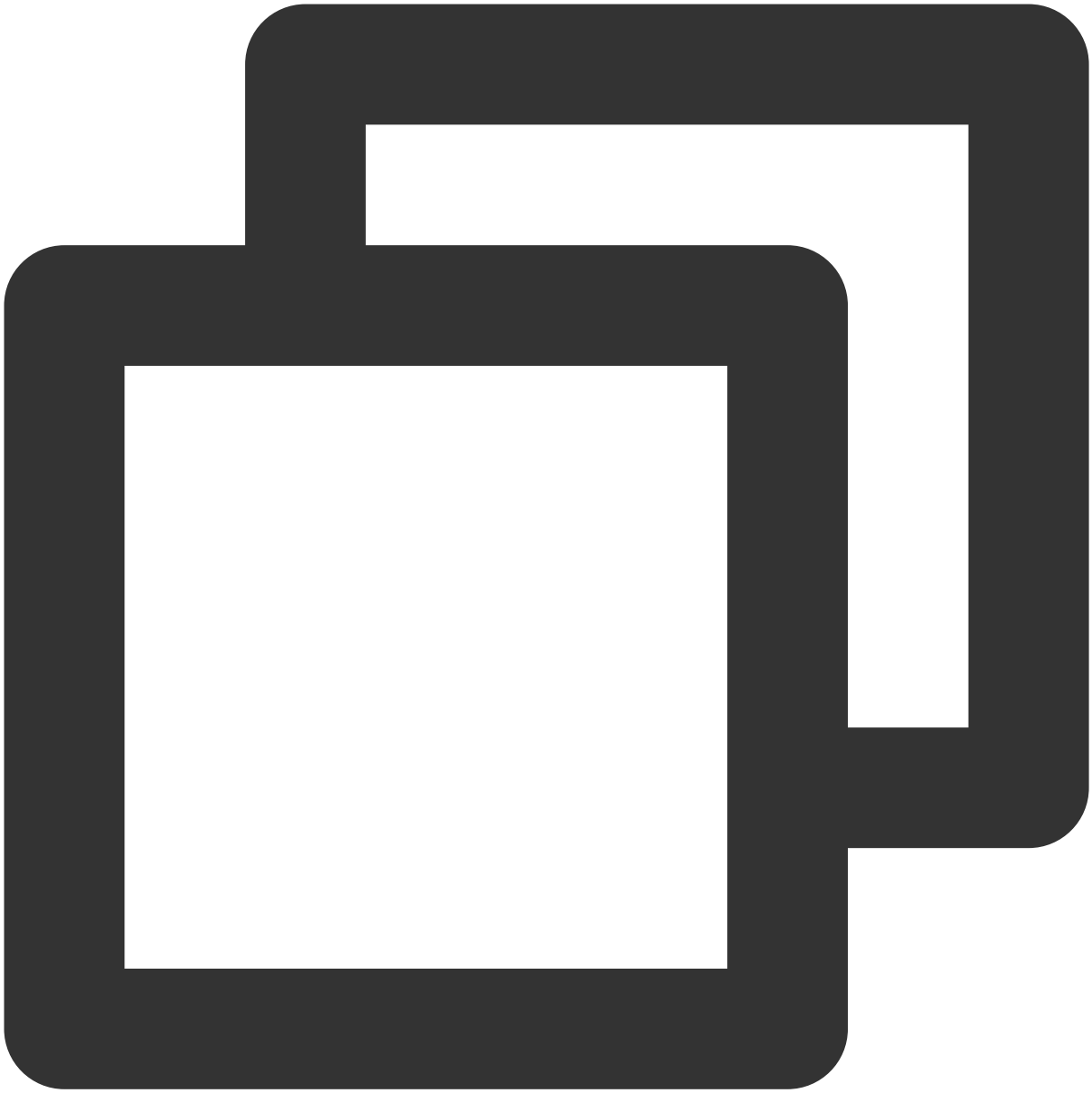
```
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reas
            }
        });
}
```

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。

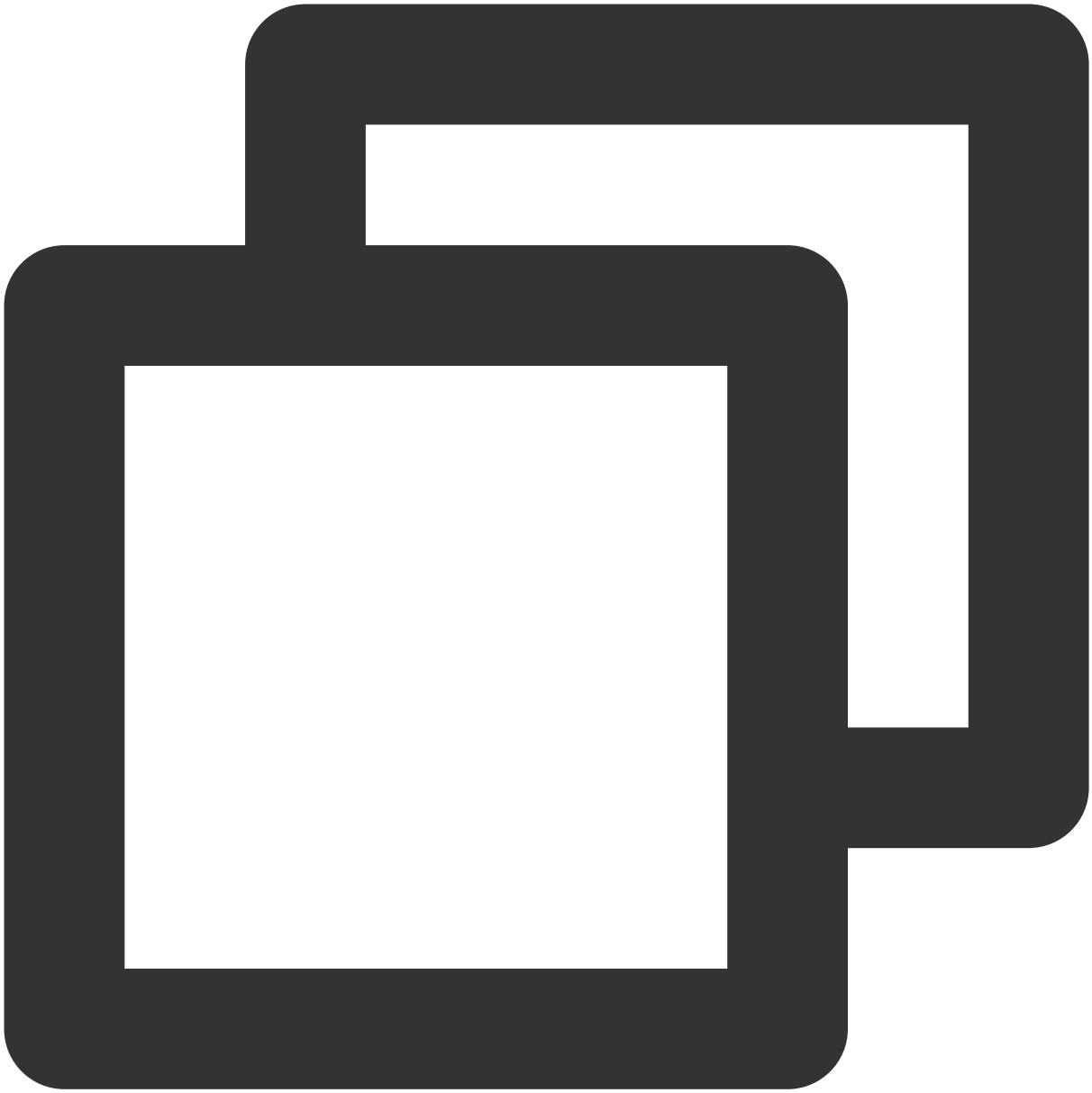
4. 实现快直播播放

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）：

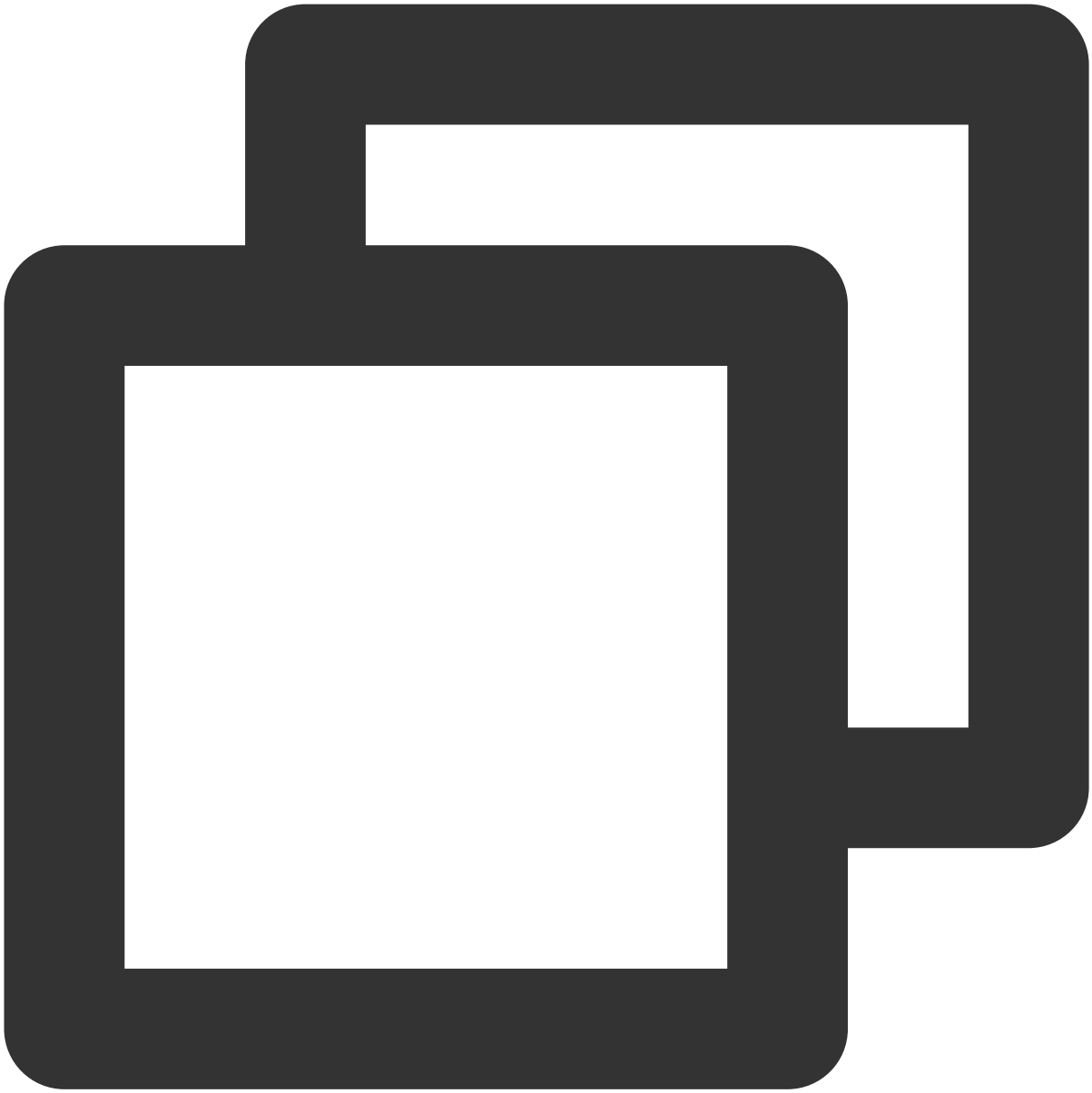
示例代码

Android

iOS



```
// 创建一个 V2TXLivePlayer 对象；
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传入低延时协议播放地址，即可开始播放；
player.startPlay("webrtc://{Domain}/{AppName}/{StreamName}");
```



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

Flutter
标准直播拉流

最近更新时间：2024-01-13 15:53:49

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
HLS	成熟度高、高并发无压力	需集成 SDK 才能播放	3s - 5s
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、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

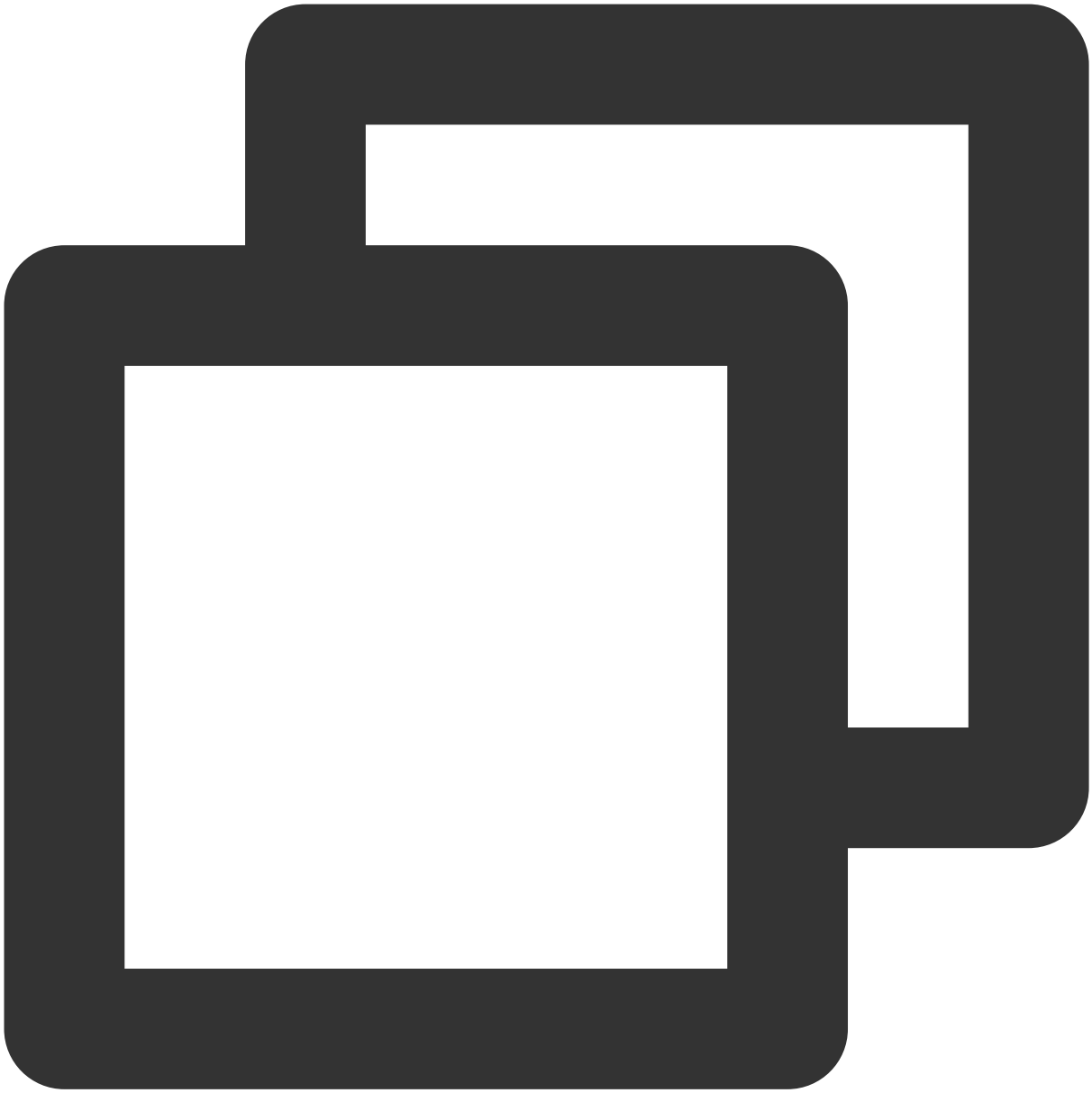
针对开发者的接入反馈的高频问题，腾讯云提供有更加简洁的 API-Example 工程，方便开发者可以快速的了解相关 API 的使用，欢迎使用。

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。

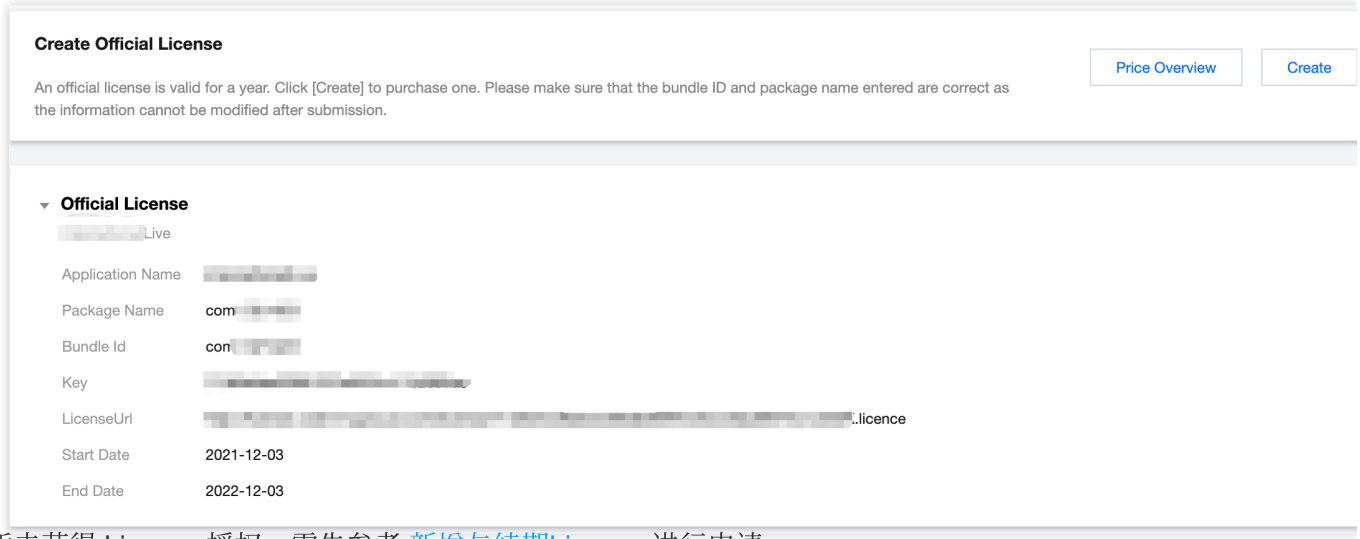


```
dependencies:
  live_flutter_plugin: latest version number
```

2. 给 SDK 配置 License 授权

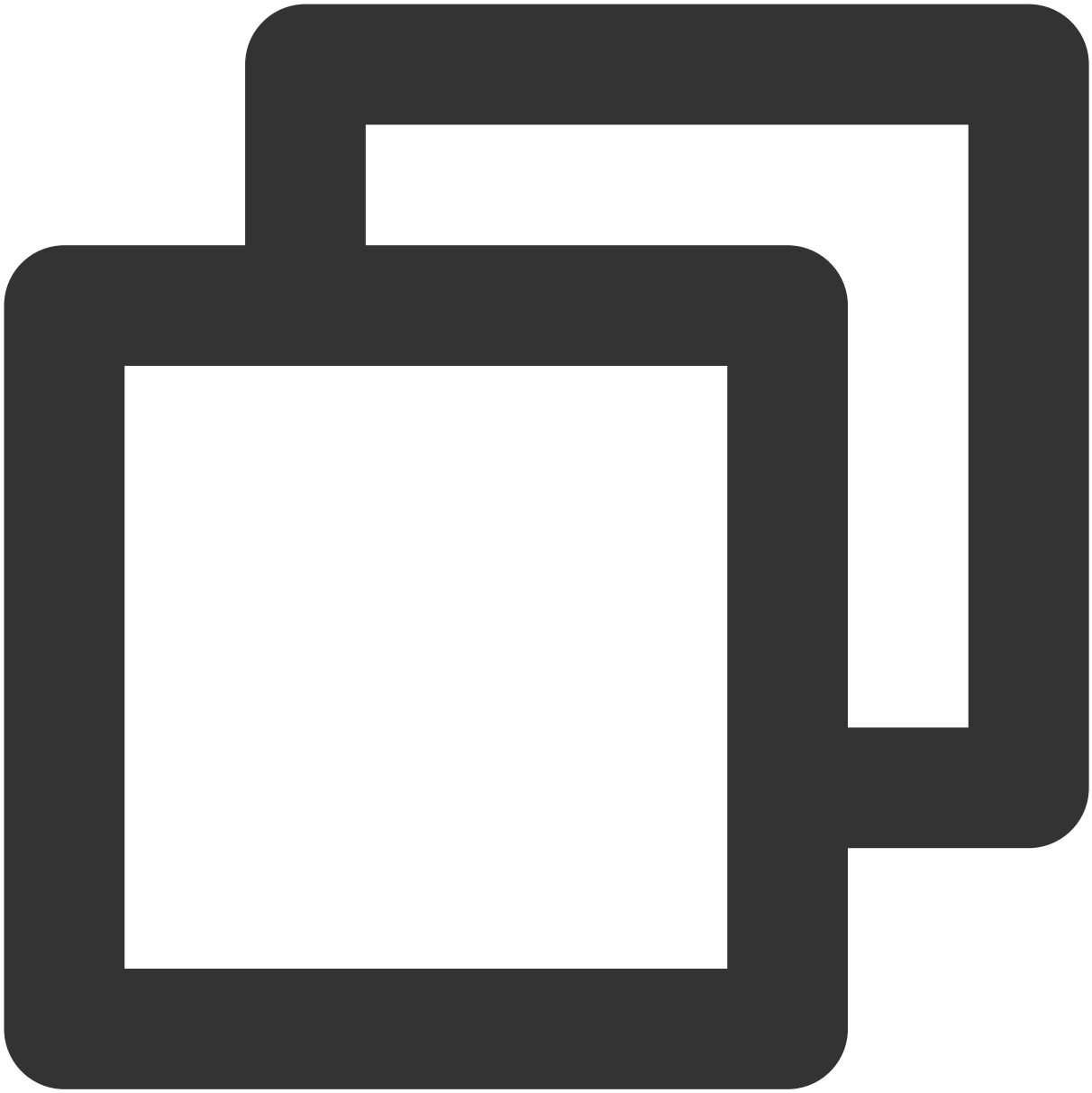
1. 获取 License 授权：

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：



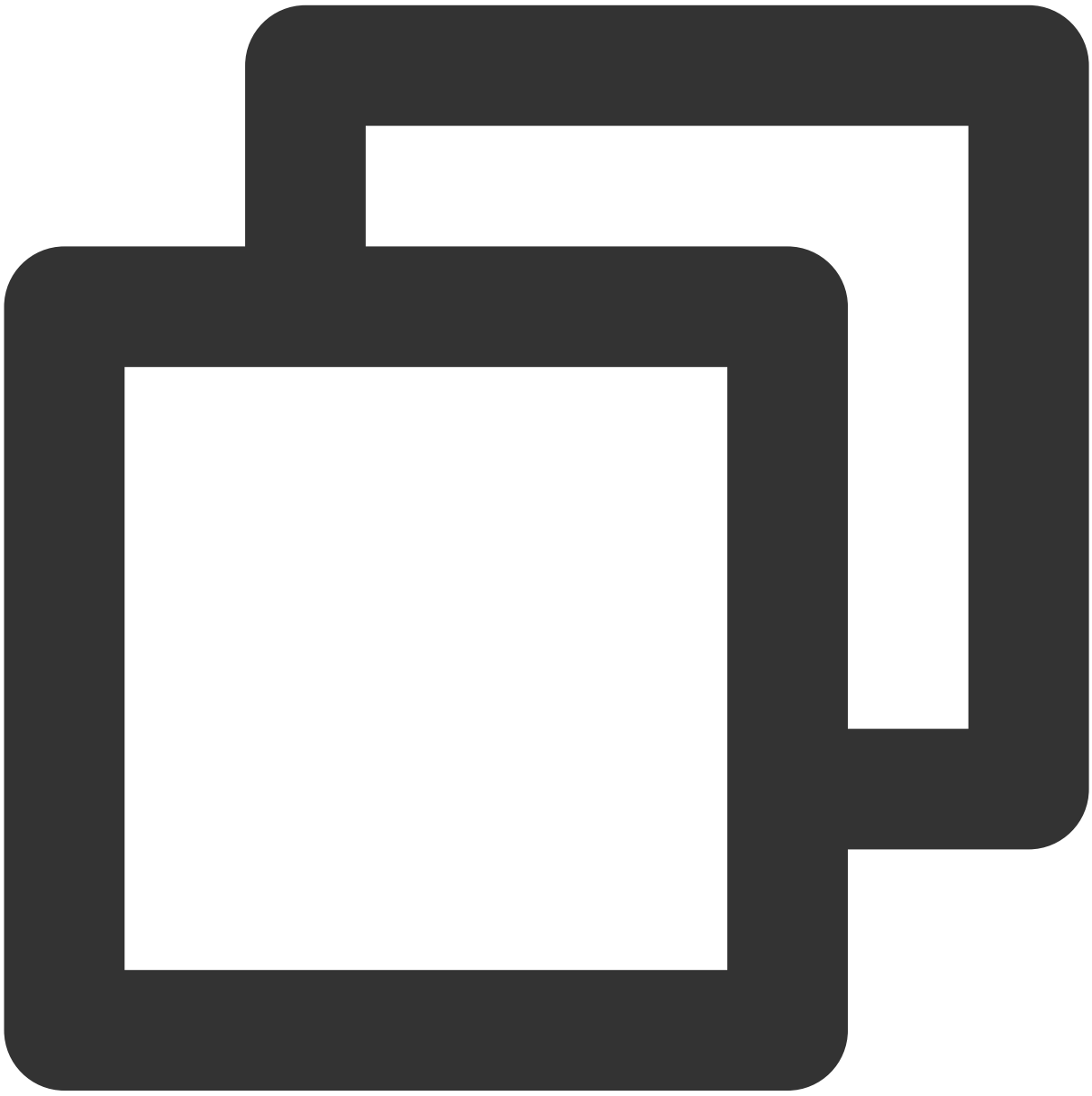
```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

 /// 腾讯云License管理页面(https://console.tencentcloud.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。

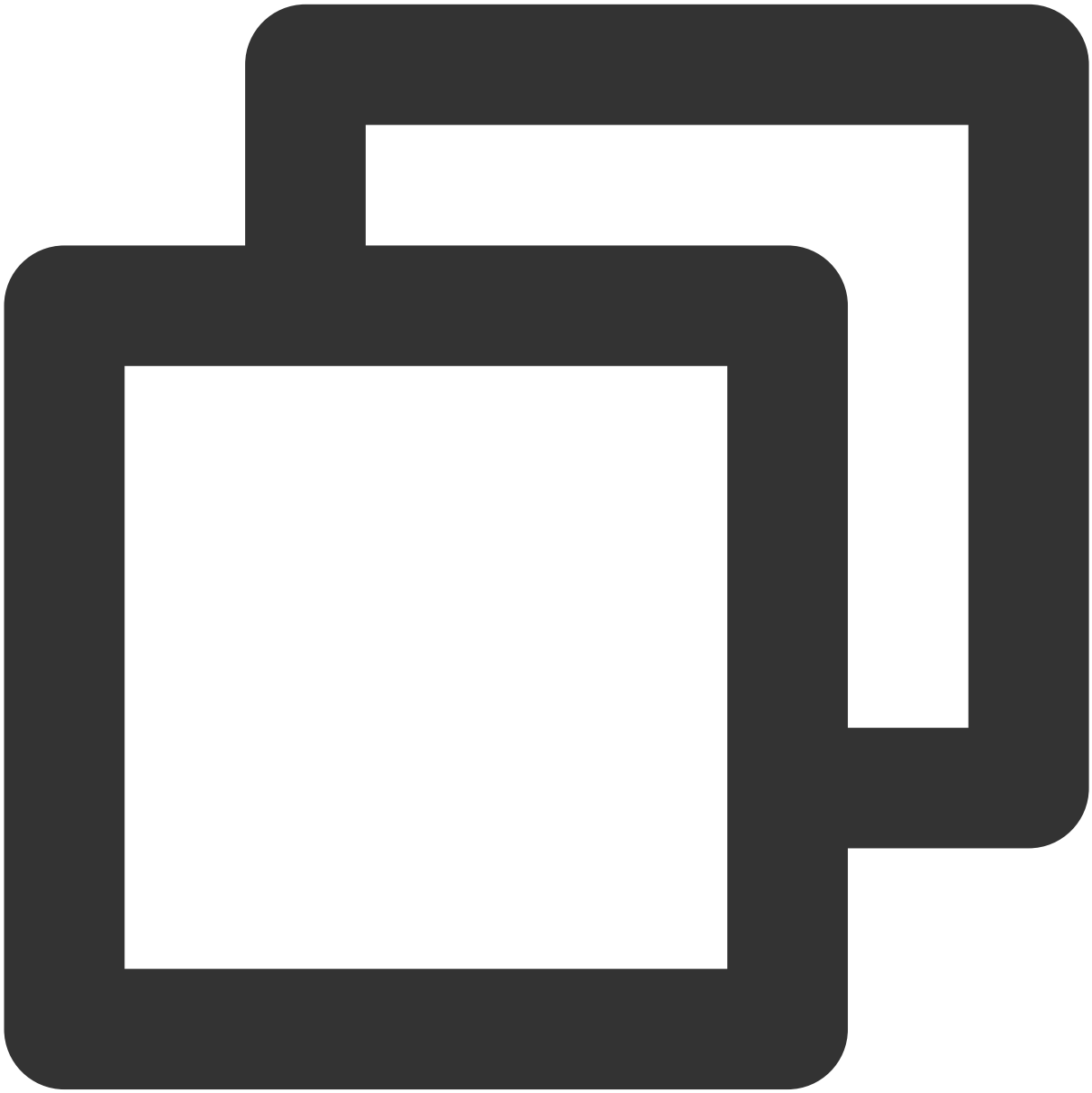


```
import 'package:live_flutter_plugin/v2_tx_live_player.dart';
/// 初始化V2TXLivePlayer
initPlayer() {
  _livePlayer = V2TXLivePlayer();
```

```
  _livePlayer.addListener(onPlayerObserver);
}
```

4. 渲染 View

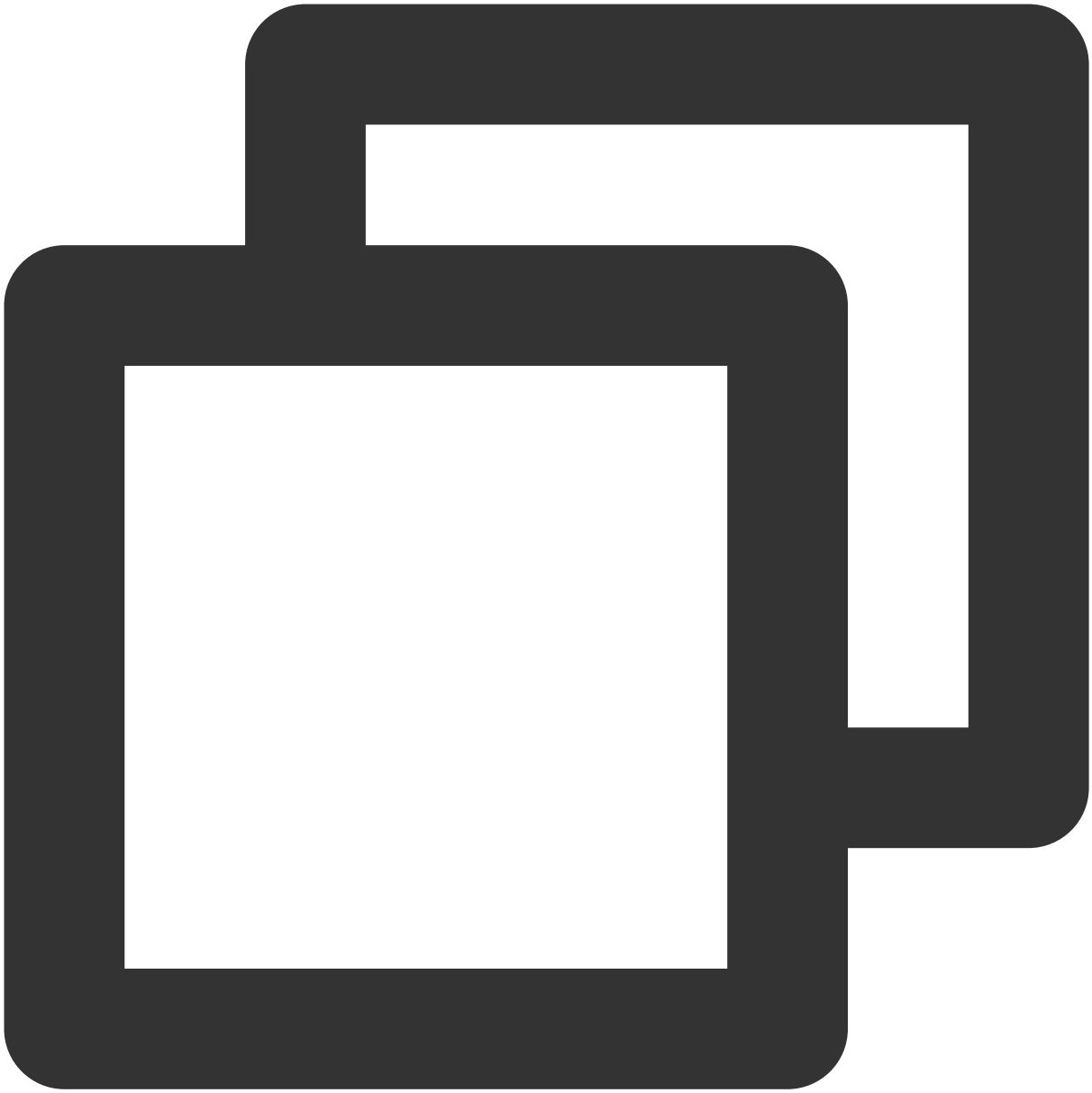
接下来我们要给播放器的视频画面找个地方来显示，Flutter 需要依赖 `v2_tx_live_video_widget` 创建视频渲染 View。



```
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);
    },
  );
}
```

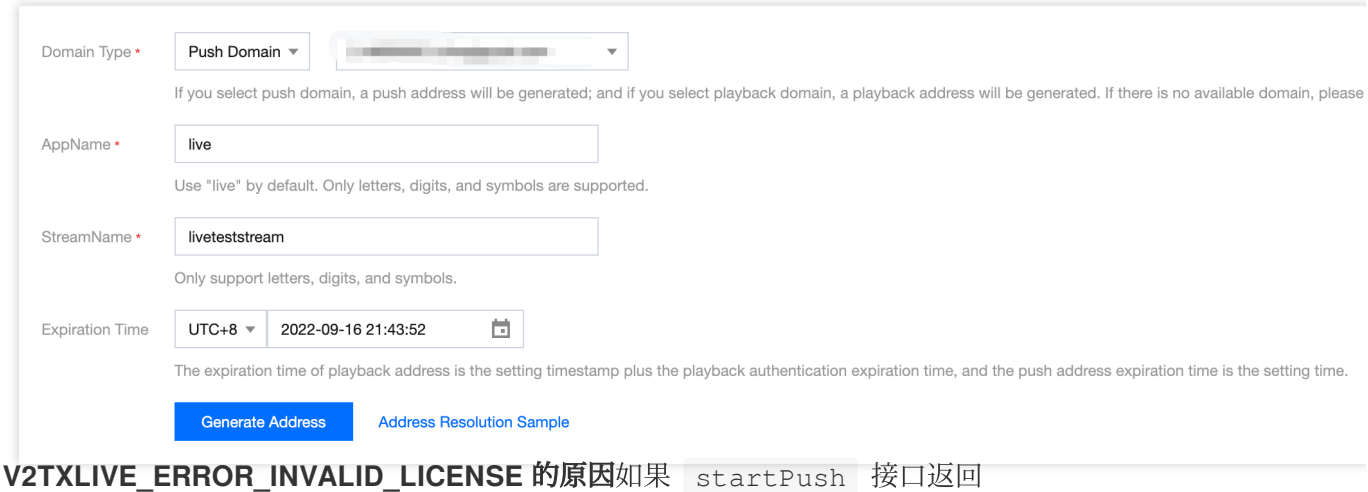
5. 启动播放



```
/// 开始拉流
startPlay() async {
  // 生成拉流url RTMP/TRTC/LEB
  var url = ""
  // 开始拉流
  await _livePlayer?.startPlay(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** > **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。

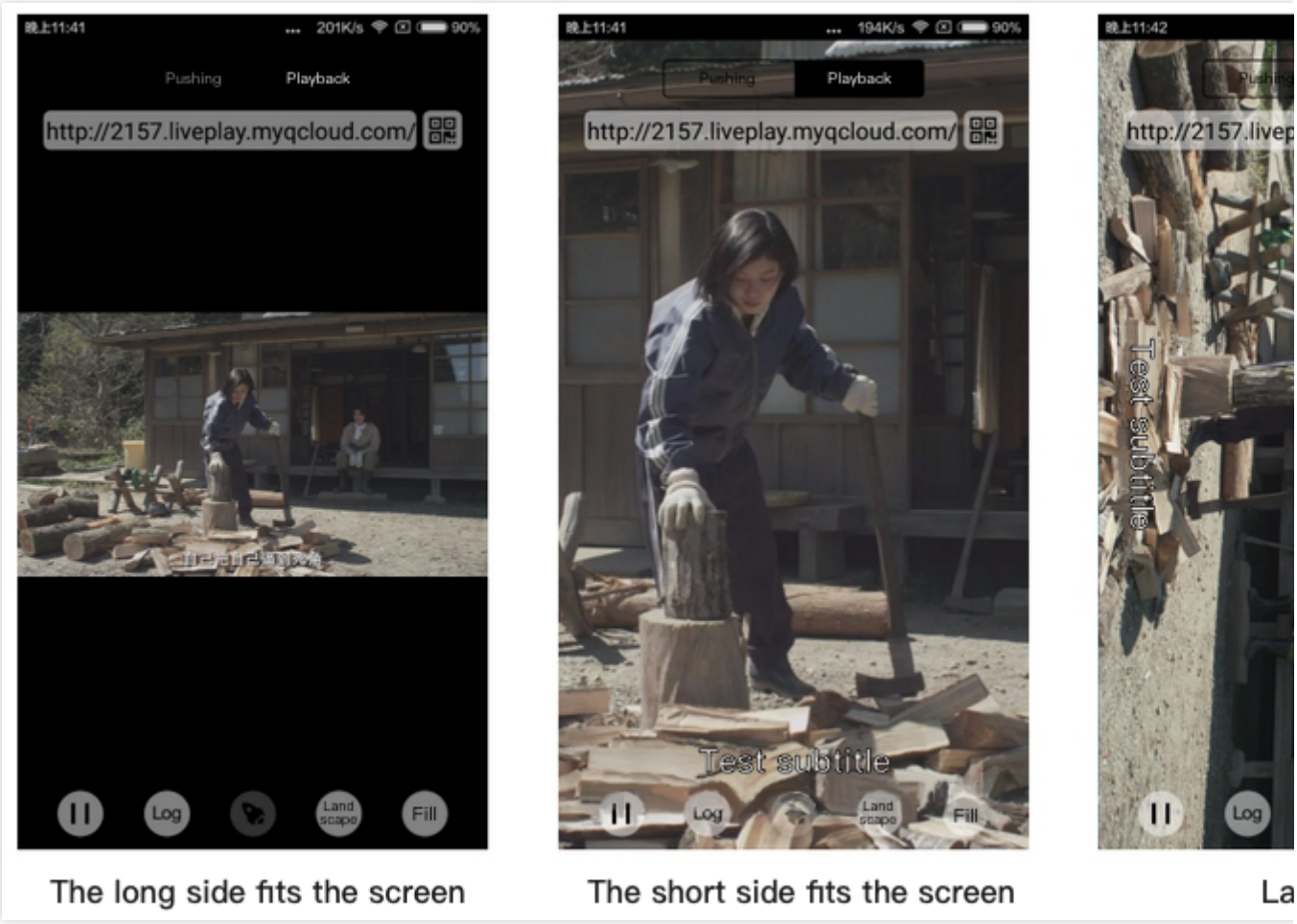
6. 画面调整

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

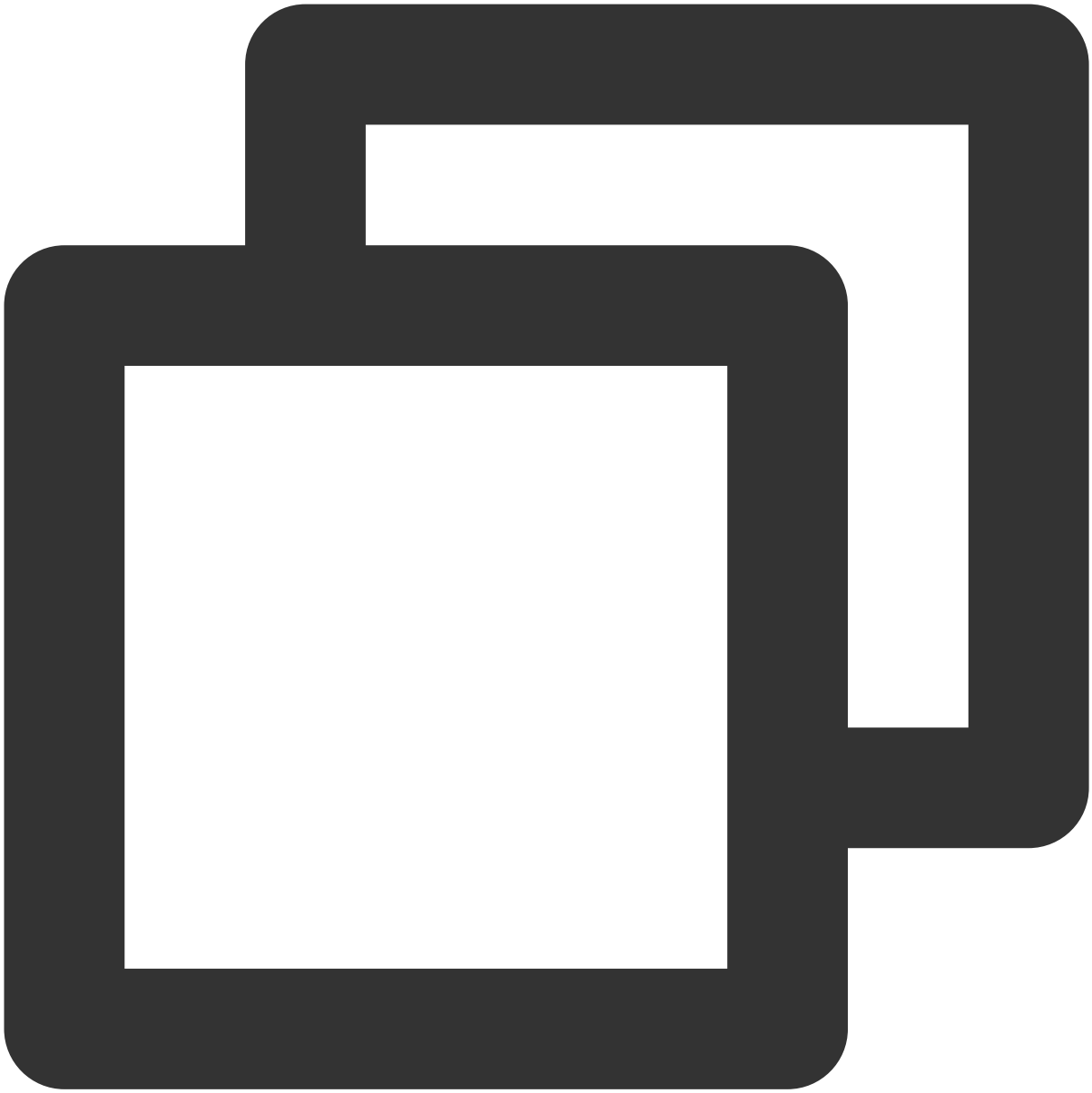
setRenderRotation：视频画面顺时针旋转角度

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



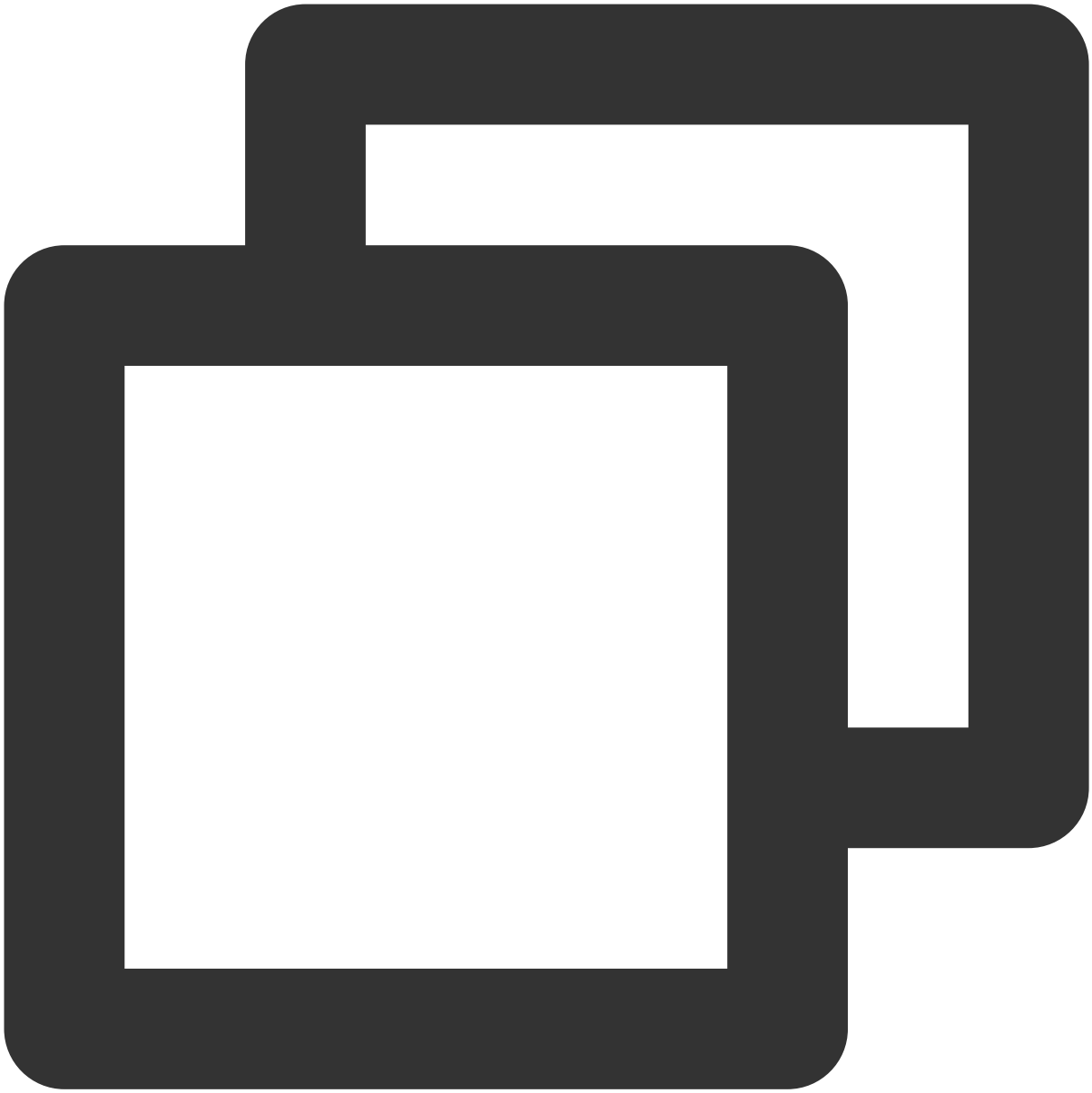
7. 暂停播放

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

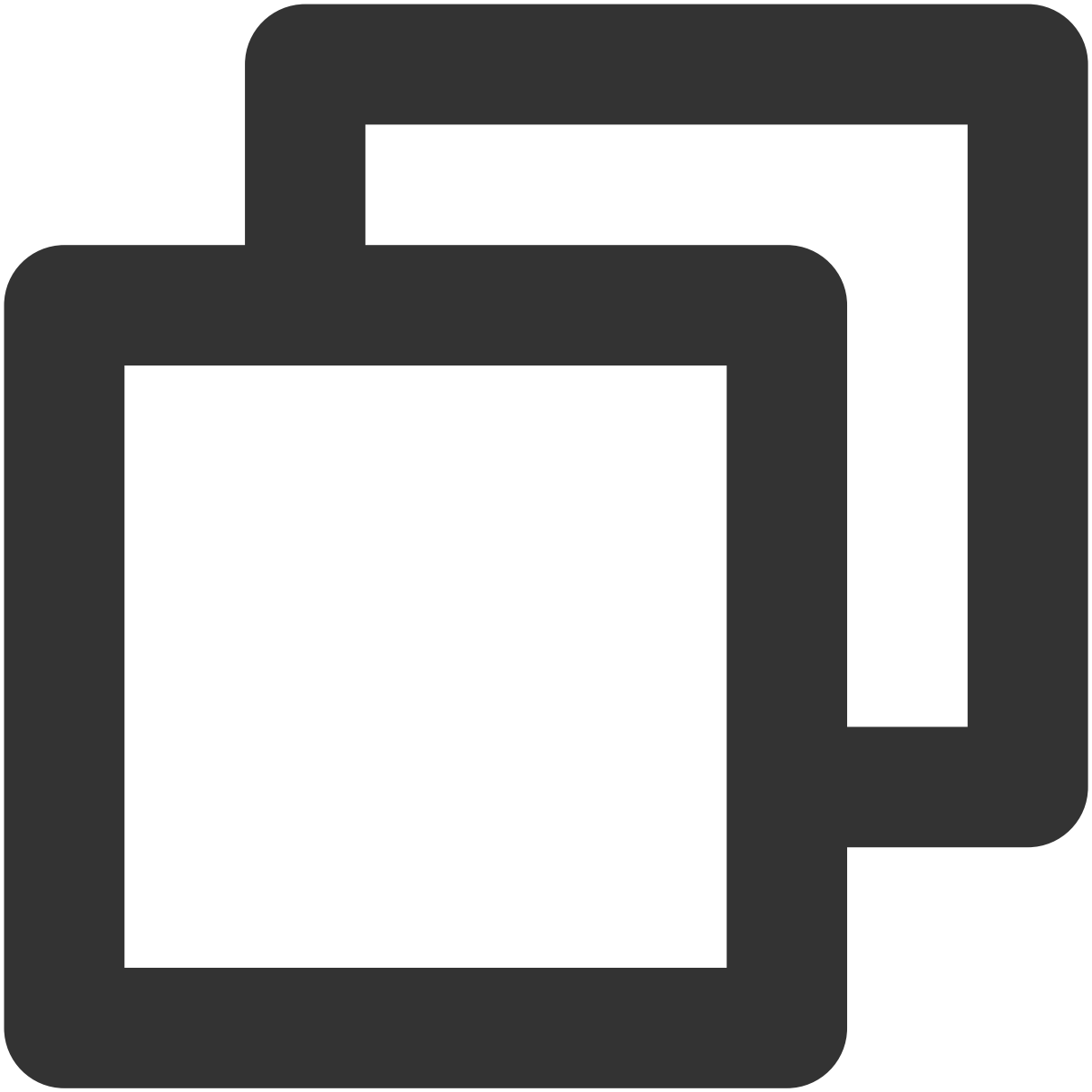
腾讯云 SDK 的直播播放功能，并非基于 ffmpeg 做二次开发， 而是采用了自研的播放引擎，所以相比于开源播放器，在直播的延迟控制方面有更好的表现，我们提供了三种延迟调节模式，分别适用于：秀场，游戏以及混合场

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。