云点播

播放器 SDK 文档

产品文档

腾讯云

文档目录

播放器 SDK 文档
概述

最近更新时间：2023-08-16 11:27:15

产品说明

播放器 Player SDK 是腾讯云专为用户的点播业务打造，提供全面、稳定、流畅的视频播放服务，帮助用户连接云端服务、打造云端一体化能力。播放器 Player 在点播播放场景提供了播放器和第三方播放器插件，同时支持 Web 端、iOS 端、Android 端、Flutter 端等多个平台。此外，云点播为客户提供多种视频播放解决方案，赋能客户的不同场景，满足客户多样化需求。

快速了解

为了保证用户可以快速了解播放器 Player，在正式使用播放器和第三方播放器插件前，建议所有客户首先阅读播放器的 基本概念 并通过 如何选择点播播放器 快速选择匹配自身业务的播放器类型。

核心优势

云端一体化服务

播放器融合强大的云点播音视频服务能力，打造功能完备的云端一体化能力，为用户业务提供更有价值的业务运营能力。

全方位视频安全

播放器支持防盗链、URL 鉴权、HLS 加密、私有协议加密、离线下载等视频安全方案，同时支持动态水印等视频安全能力，全方位保证用户媒体安全，满足业务不同场景的安全需求。

完整的数据支撑

播放器支持全链路视频播放质量监控，包含播放性能、用户行为、文件特征等多维度数据指标，助力业务高效运营。

极致的播放体验

依靠腾讯云海量加速节点，提供完备的视频加速能力，毫秒级的延迟让用户无延迟体验极速视频播放，为用户业务保驾护航。

多样化播放能力

提供首屏秒开、边播边缓存、倍速播放、视频打点、媒体弹幕和外挂字幕等多种功能，用户可以根据业务需求选择功能，助力业务生态建设。

常见场景

短视频播放

播放器结合云点播平台提供的内容审核、媒资管理、无缝切换、首屏秒开互动浮窗等功能，常用于用户打造短视频相关场景，同时云点播提供 短视频 UGSV Demo 供用户参考。

长视频播放

播放器结合云点播的自适应码流、无缝清晰度切换、缩略图、截图、倍速播放等功能，长视频用户打造视频剧集和门户类场景，同时云点播提供 视频播放方案 供用户参考。

视频版权保护

播放器支持云点播的视频安全能力，支持私有协议加密、离线下载、跑马灯和防盗链等功能，助力用户保障自己视频安全，同时云点播提供 视频安全方案教程 供用户参考。

直播录制

播放器支持直播录制文件播放，同时支持直播时移回看、伪直播观看，在音视频直播点播场景下帮助用户打造一体化观看体验。

SDK下载

您可以体验播放器 Demo，更多信息，请参见 Demo 体验。
您可以下载对应的 SDK 进行集成操作，更多信息，请参见 SDK 下载。

您可以通过 功能列表，查询播放器是否满足自己的能力需求。

接入指引

为了帮助您快速接入播放器，我们为您提供了播放器接入指引，以示例的方式为您讲解接入步骤。
如遇到播放问题，请参见 常见问题文档。

更多终端SDK

除了播放器外，我们还为您提供 短视频制作、直播 和 美颜 等终端SDK来快速满足不同功能场景需求，您可以根据您的需要选择相应的SDK，如果您同时需要 直播、播放、短视频、实时音视频多个SDK功能，也可以选择下载 全功能（All-In-One）版本SDK。
播放器SDK
直播SDK
短视频SDK
实时音视频SDK
全功能SDK

基本概念

最近更新时间：2023-06-16 10:59:22

本文对播放器 Player 的点播播放场景中涉及的基本概念进行说明，帮助您快速的理解和使用播放器 Player 下的视频点播能力。

基本概念

播放器 Player 在云点播平台支持播放器和第三方播放器插件两种方式接入点播服务。

播放器

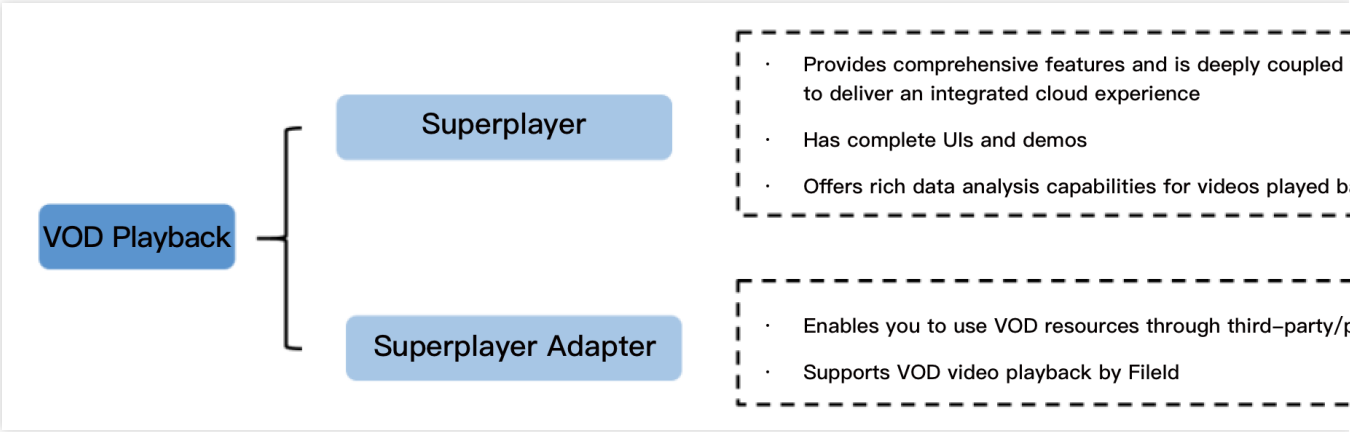
播放器是一款独立完整的视频播放器，具备视频加密、缩略图预览、清晰度切换等全面的视频播放功能；该播放器拥有完整 UI 和体验 Demo，同时深度融合腾讯云点播业务，可通过使用点播文件标识 FileID 播放云点播资源，此外，播放器还提供云点播全链路视频播放质量数据服务。如果您想要快速接入播放器，请参见 播放器教程。

第三方播放器插件

第三方播放器插件是一款用于连接第三方播放器与腾讯云点播资源的播放器插件，具备视频播放、视频加密等能力。将第三方播放器插件集成在用户第三方播放器中，即可通过点播文件标识 FileID 播放云点播资源。如果您想要快速接入第三方播放器插件，请参见各端集成文档。

如何选择播放器

为降低用户的接入难度、匹配用户自身业务场景，云点播建议用户在接入播放器服务时，选择最适合自己的类型接入：



播放器：适用于尚未集成播放器，但需要快速搭建点播视频播放能力的用户。播放器 Player 功能全面，接入便捷，云点播为用户提供了详细的播放器接入教程，详情请参见 播放器教程。

第三方播放器插件：适用于需要使用自研/第三方播放器播放云点播资源的用户。云点播提供第三方播放器插件帮助客户顺利接入和使用云点播平台资源。

各类型播放器支持的平台端类型如下：

播放器类别	播放器	第三方播放器插件
Web 端	✓	✓
iOS 端	✓	✓
Android 端	✓	✓
Flutter 端	✓	-
UI	✓	-
Demo	✓	-

产品功能

最近更新时间：2024-06-28 17:42:12

播放器 SDK 提供直播、点播场景的视频播放能力，支持 Web/H5、iOS、Android、Flutter 等平台，支持的功能详情如下：

功能模块	功能项	功能简介	Web	iOS & Android	Flutter
播放协议/格式	点播或直播支持	同时支持点播播放和直播播放能力	✓	✓	✓
	支持的直播播放格式	支持 RTMP、FLV、HLS、DASH 和 WebRTC 等直播视频格式	WebRTC、FLV、HLS、DASH	RTMP、FLV、HLS、WebRTC	RTMP、FLV、HLS、WebRTC
	支持的点播播放格式	支持 HLS、DASH、MP4 和 MP3 等点播音视频格式	HLS、MP4、MP3、FLV、DASH	MP4、MP3、HLS、DASH（DASH 仅高级版支持）	MP4、MP3、HLS
	URL 播放	支持网络视频的 URL 方式播放，URL 可以为点播播放地址也可以为直播拉流地址	✓	✓	✓
	FileID 播放	支持通过云点播文件标识 FileID 播放视频，包含多个清晰度的视频、缩略图、打点等信息	✓	✓	✓
	本地视频播放	支持播放存储在本地的视频文件	-	✓	✓
	快直播	支持腾讯云毫秒级超低时延快直播播放	✓	✓	✓
	DASH 协议支持	支持标准协议的 DASH 视频播放	✓	√（仅高级版支持）	×
	全景VR视频	支持播放全景VR视频源，移动端设备支持手指拖动或陀螺仪操作以查看全景视频内容，PC端设备支持鼠标在界面上拖动画面查看	√（仅高级版支持）	×	×

	Quic 加速	支持 Quic 传输协议，有效提升视频传输效率	-	✓（仅高级版支持）	✓
	SDR/HDR 视频	支持播放 SDR 视频和 HDR 10/HLG 标准的 HDR 视频	-	✓	✓
	H.264播放及软硬解	支持播放H.264视频源，并支持软硬解	✓	✓	✓
	H.265 硬解	支持对 H.265 视频源的硬解码播放	-	✓	✓
	AV1	支持播放 AV1 编码格式的视频	部分支持	部分支持（仅高级版支持）	×
	纯音频播放	支持 MP3 等文件纯音频播放	✓	✓	✓
	双声道音频	支持播放双声道音频	×	✓	✓
	多音轨	支持播放含多音轨的视频文件，播放时可切换音轨，如英文切换中文。	✓	✓（仅高级版支持）	×
	设置 Http Header	请求视频资源时，自定义 HTTP Headers 内容	×	✓	✓
	支持 HTTPS	支持播放 HTTPS 的视频资源	✓	✓	✓
	HTTP 2.0	支持 HTTP 2.0协议	✓	✓	✓
播放性能	短视频播放组件	以极低的接入成本，实现极速首帧、无感启播、丝滑切换的短视频播放体验。结合预播放、预下载、播放器复用、精准流量控制、加载策略等技术，在保证低能耗的前提下实现极致流畅的播放效果	-	✓（仅高级版支持）	×
	预下载	支持提前下载指定视频文件内容，并支持配置预下载视频文件的大小及分辨率。	✓	✓	✓
	边播边缓存	支持播放的同时缓存下载后面的内容，降低网络占用，可设置缓存策略	✓	✓	✓

	精准 seek	支持在进度条上跳转到指定位置进行播放，移动端可精确到帧级别，Web 端精准到毫秒级	✓	✓	✓
	自适应码率	支持播放 HLS、DASH 和 WebRTC 的自适应码流，可根据网络带宽自动选择合适的码率进行播放	✓	✓（DASH 仅高级版支持）	✓（不支持 DASH）
	实时下载网速	支持获取实时下载网速，既可根据业务需要给 C 端用户在卡顿时展示下载网速，也是使用自适应码率带宽预测模块的必要前提	✓	✓	✓
	多实例	支持在一个界面添加多个播放器同时播放	✓	✓	✓
	动态追帧	发生卡顿时，通过类似“快进”的方式追赶上当前直播进度，保证直播画面实时性	✓	×	×
	PDT Seek	跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能	×	✓（仅高级版支持）	✓
播放控制	基础控制	支持开始、结束、暂停和恢复等播放控制功能	✓	✓	✓
	基础画中画	支持切换到画中画以小窗形式播放，移动端同时支持在集成 APP 内或 APP 外的画中画播放。	✓	✓	✓
	高级画中画组件	相对基础画中画，新增支持加密视频画中画、离线播放画中画和“秒切”效果。	-	✓（仅高级版支持）	×
	缓存内 seek	支持已经缓存的视频内容在 seek 时不清除缓存内容并快速 seek	✓	✓	✓
	直播时移	支持直播时移视频流播放，可设置开始、结束和当前支持时间，支持拖动	✓	×	×
	进度条标记及缩略图预览	支持在进度条上添加标记信息，并支持缩略图（雪碧图）预览	✓	✓	✓
	设置封面	支持设置播放视频的封面	✓	✓	✓

重播、循环播放、列表播放	支持视频播放结束后自动或手动重播；同时支持依次播放视频列表中的视频，并支持轮播，即在视频列表最后一个视频播放完成后，播放列表的第一个视频。	✓	✓	✓
断点续播	支持从上次播放结束位置开始播放	✓	✓	✓
自定义启播时间	支持自定义视频开启播放的时间	✓	✓	✓
倍速播放	支持0.5~3倍的变速播放，音频实现变速不变调	✓	✓	✓
后台播放	支持界面切到后台后继续播放音频和视频	-	✓	✓
播放回调	支持对播放状态回调、首帧回调、播放完成或失败回调	✓	✓	✓
播放失败重试	播放失败时自动重试，支持直播的自动重连功能	✓	✓	✓
音量设置	支持实时调节系统音量和静音操作	✓	✓	✓
清晰度切换和命名	支持用户流畅无卡顿的切换 HLS 视频的多路清晰度流，并支持为不同清晰度流自定义命名	✓	✓	✓
截图功能	支持截取播放画面的任意一帧	-	✓	×
试看功能	支持播放开启试看功能的视频	✓	✓	×
弹幕	支持在视频上方展示弹幕	✓	✓	×
外挂字幕	支持导入自定义字幕文件，Web 端支持WebVTT 格式，移动端支持 VTT、SRT 格式	✓	✓（仅高级版支持）	x
SEI 回调	解析视频流中的 SEI 帧，并进行事件回调	×	✓（仅高级版支持）	×
HEVC 降级播放	播放器支持同时传入 HEVC 和其它视频编码格式。例如： H.264	×	✓（仅高级版	×

		的播放链接， 当播放机型不支持 HEVC 格式时，将自动降级为配置的其它编码格式（如： H.264 ）的视频播放。		支持）	
	音量均衡	在播放音频时自动调整音量，使得所有音频的音量保持一致。	×	✓（仅高级版支持）	×
视频安全	referer 黑白名单	支持通过播放请求中携带的 Referer 字段识别请求的来源，以黑名单或白名单方式对来源请求进行控制	✓	✓	✓
	Key 防盗链	支持在播放链接中加入控制参数，控制链接的有效时间、试看时长、允许播放的 IP 数等	✓	✓	✓
	HLS 加密	支持基于 HLS 提供的 AES encryption 方案，使用密钥对视频数据加密	✓	✓	✓
	HLS 私有加密	支持在云点播的私有协议对视频进行加密，且仅能通过播放器 SDK 对加密后的视频进行解密播放，可有效防范多种浏览器插件和灰产工具的破解	✓	✓	✓
	商业 DRM	提供苹果 Fairplay、谷歌 Widevine 原生加密方案	✓	✓（仅高级版支持）	×
	安全下载	支持离线下载加密视频后，仅可通过播放器 SDK 对视频进行解密播放	-	✓	✓
	动态水印	支持在播放界面添加不规则跑动的文字水印，有效防盗录	✓	✓	×
	幽灵水印	随机时间、随机位置、短暂的出现在播放界面上，同时一旦检测到水印被异常去除，将自动停止视频播放；可在几乎不影响观看体验的同时保证视频安全。	✓	×	×
	Web 安全插件	检测 Web 端播放环境和播放状态是否正常，异常环境下将中断视	✓	-	-

		频播放，保护视频安全。插件包含 MSE 环境检测、安全结构检查和接口响应完整性校验。	（仅高级版）		
显示效果	自定义 UI	SDK 提供含 UI 集成方案，提供包含 UI 界面的常用播放组件，可以根据自身需求选用	✓	✓	✓
	屏幕填充	支持为视频画面选择不同填充模式，适应屏幕大小	✓	✓	×
	设置播放器尺寸	支持自定义设置播放器的宽高	✓	✓	✓
	图片贴片	支持暂停时，增加图片贴片用于广告展示	✓	✓	×
	视频镜像	支持水平、垂直等方向的镜像	✓	✓	×
	视频旋转	支持对视频画面按角度旋转，同时支持根据视频文件内部 rotate 参数自动旋转视频	×	✓	×
	锁定屏幕	支持锁屏功能，包含锁定旋转和隐藏界面元素	-	✓	×
	亮度调节	支持播放视频时调节系统亮度	-	✓	✓

注意：

表中“-”表示该端无需具备相应功能或不存在相关概念。

未注明“仅高级版支持”的“✓”代表基础版即支持。

Demo 体验

最近更新时间：2024-04-11 16:11:37

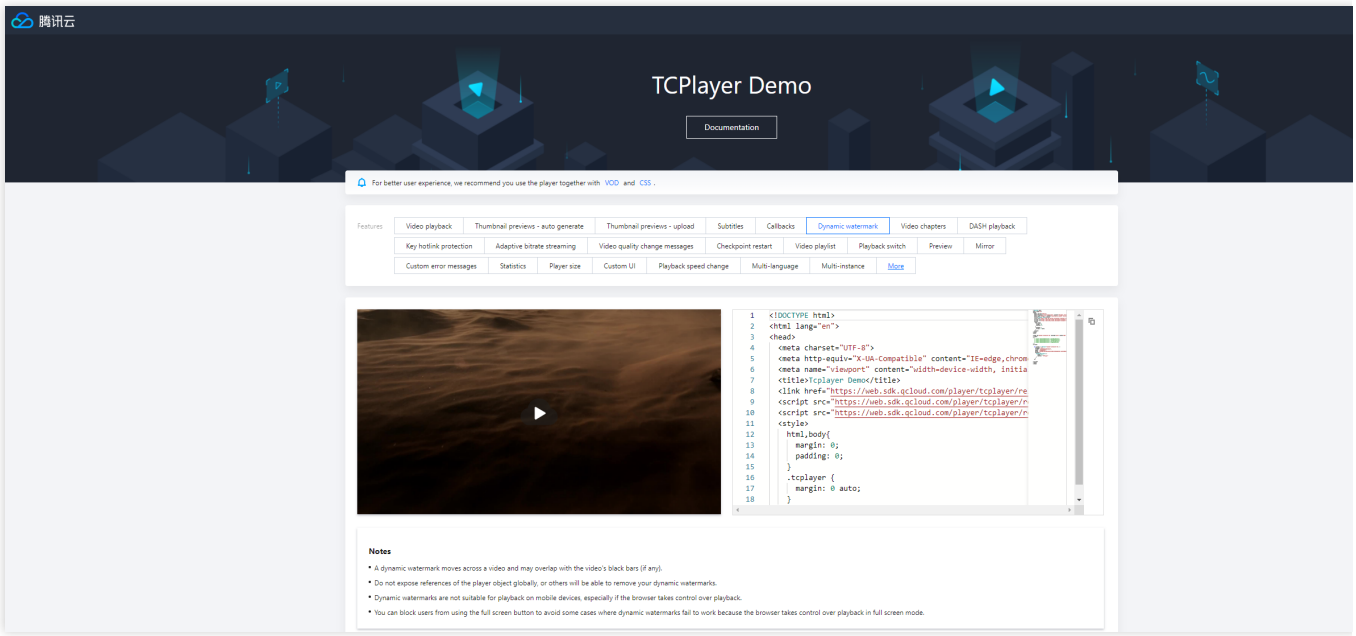
播放器 SDK Demo 提供完整的产品级交互界面和业务源码，开发者可按需取用。

功能体验 Demo

您可通过下述地址/二维码获得 Demo 进行功能体验。移动端扫码下载腾讯云音视频 App 后，在**视频播放**卡片中体验。

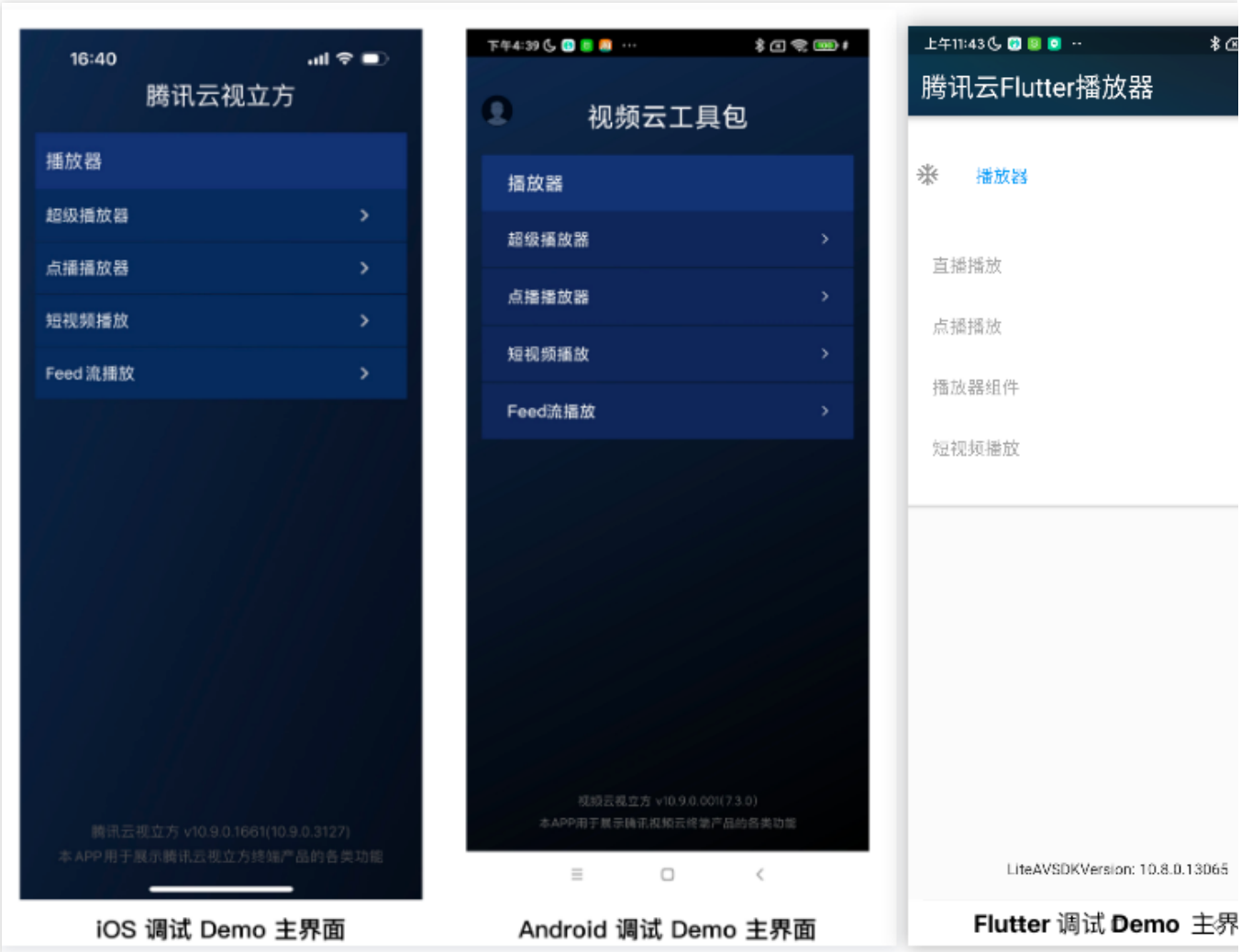
Web端（TCPlayer）

Web 端播放器支持 PC 端和移动端的浏览器视频播放，Web 播放器 Demo 提供了可对比查看视频播放功能效果及其配套代码的 Demo 体验页面，您可以通过修改示例代码，即时的在播放区域内查看修改后的功能效果。



开发调试 Demo

为了帮助开发者更好的理解播放器 SDK 的使用方式，播放器 SDK 移动端提供了可供开发调试的 Demo 源代码及接口使用说明，您可参考下述步骤使用。



iOS 调试 Demo 主界面　Android 调试 Demo 主界面　Flutter 调试 Demo 主界

步骤一：获取 Demo 工程源码

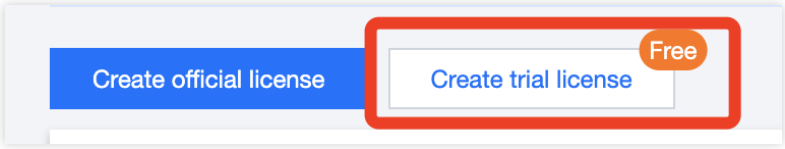
您可访问下述 GitHub 地址获取调试 Demo 源代码，或者下载对应的 ZIP 包。

平台	源码地址	ZIP 包下载
iOS	GitHub	ZIP 包
Android	GitHub	ZIP 包
Flutter	GitHub	-

步骤二：配置 License

播放器 SDK 移动端（iOS & Android & Flutter）需获取 License 后方可使用。

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



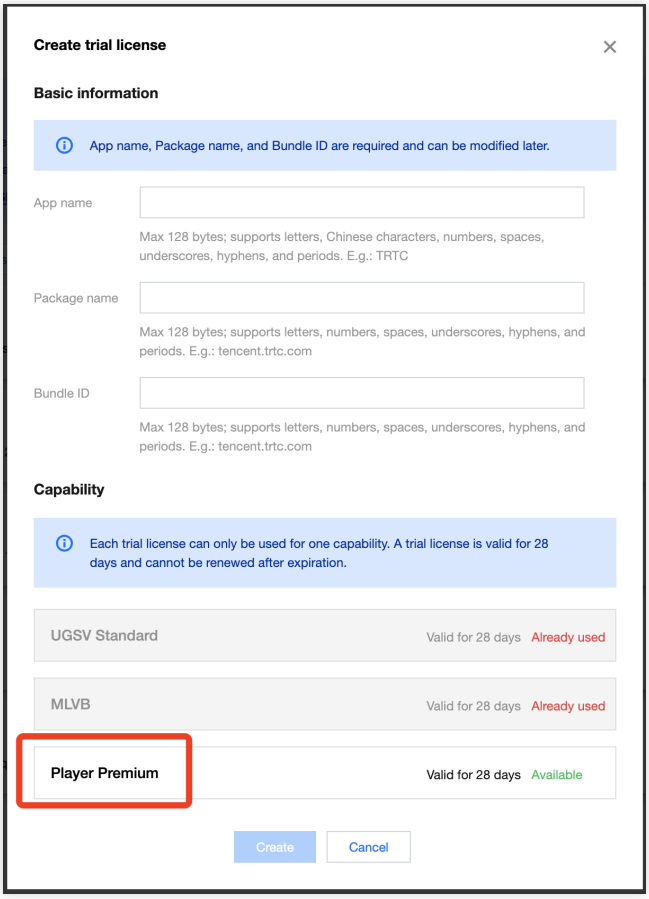
2. 根据实际需求填写 `App Name` 、 `Package Name` 和 `Bundle ID` ，勾选功能模块 **播放器高级版**，单击**确定**。

Package Name：请在 App 目录下的 **build.gradle** 文件查看 **applicationId** 。

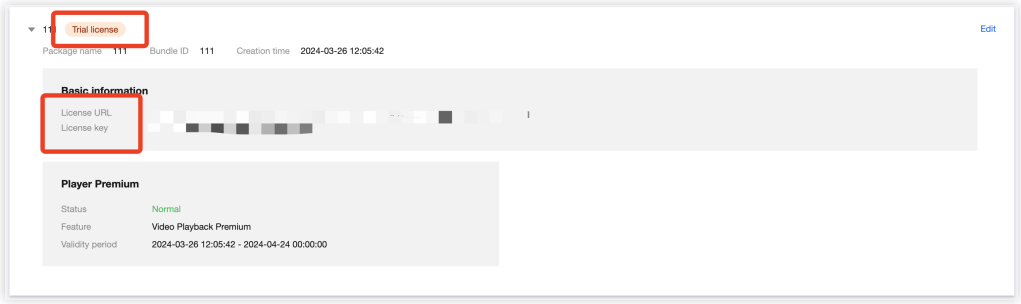
Bundle ID：请在 **xcode** 中查看项目的 **Bundle Identifier** 。

注意：

如果在腾讯云控制台申请 Package Name 或 Bundle ID ，和工程中实际的 Package Name 或 Bundle ID 不一致，将会播放失败。



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 License URL 和 License Key 两个参数，请妥善保存以下信息。**



4. 获取到 Licence URL 和 Licence Key 后，请参考下面的教程把它们配置到 Demo 工程。

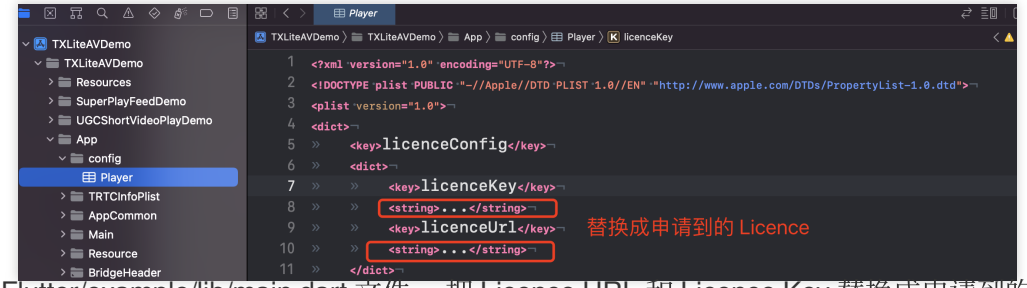
Android 端配置 Licence

iOS 端配置 Licence

Flutter 端配置 Licence

打开 Demo/app/src/main/java/com/tencent/liteav/demo/TXCSDKService.java 文件， 把 Licence URL 和 License Key 替换成申请到的 Licence 内容。

打开 Demo/TXLiteAVDemo/App/config/Player.plist 文件， 把 Licence URL 和 Licence Key 替换成申请到的 Licence 内容。



打开 Flutter/example/lib/main.dart 文件， 把 Licence URL 和 Licence Key 替换成申请到的 Licence 内容。

免费测试

最近更新时间：2024-04-11 16:11:38

播放器 SDK 提供测试版 License，包含播放器**移动端**测试 License 和播放器 **Web 端**测试 License，您可参照本文档指引，按需免费申请使用。

测试 License 类型	可授权的功能	有效期
播放器移动端测试 License	播放器 SDK **移动端**所有功能（含高级版功能）	申请后默认28天
播放器 Web 端测试 License	播放器 SDK **Web 端**所有功能（含高级版功能）	申请后默认14天，可续期1次，共28天

注意

购买后可在 云点播控制台 或 云直播控制台 对播放器 License 进行新增和续期等操作，详情请参见 新增与续期正式版 License。

移动端测试 License

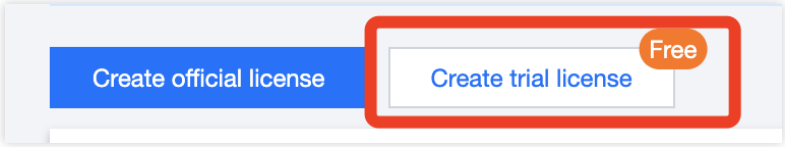
申请测试版 License

您可以免费申请播放器移动端测试 License （有效期共28天）体验测试。申请测试模块时，您可以选择**新建测试 License 并选择播放器测试 License** 或在**已创建的测试应用中申请测试新 License** 两种方式创建测试 License。

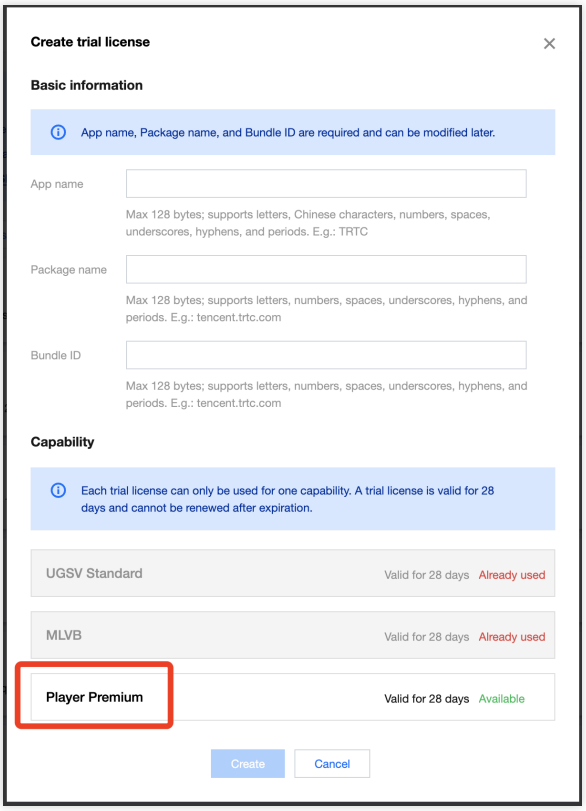
方式一：新建测试 License 并选择播放器测试 License

方法二：已创建的测试应用中申请测试新 License

1. 登录 **云点播控制台 或 云直播控制台 > License 管理 > 移动端 License**，单击**新建测试 License**。



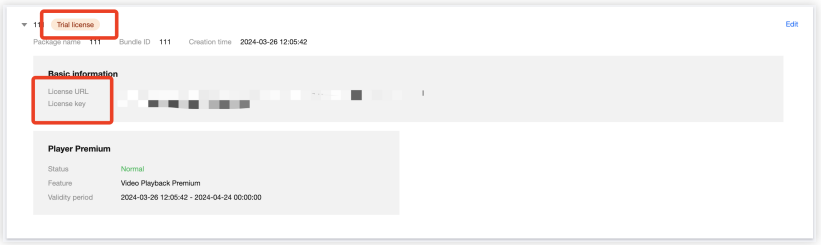
2. 根据实际需求填写 `App Name` 、 `Package Name` 和 `Bundle ID` ，选择**播放器高级版**，单击**确定**。



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。**

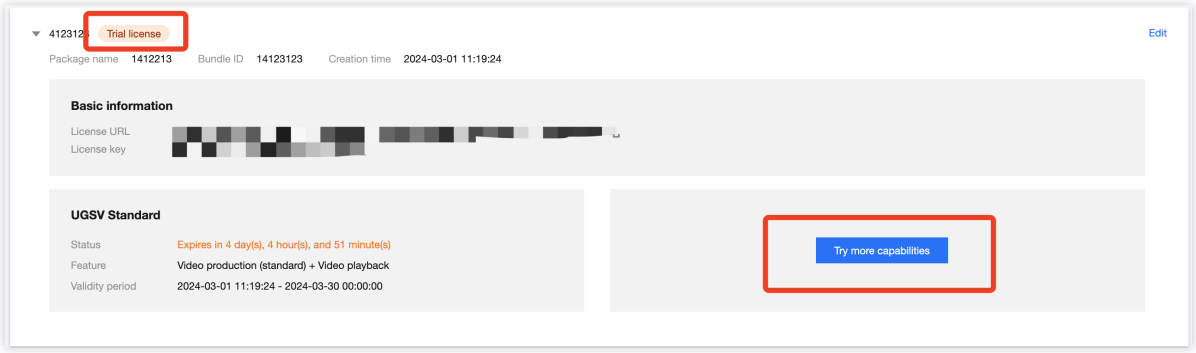
注意：

同一应用的 License URL 和 Key 唯一，测试版 License 升级为正式版后 License URL 和 Key 不变。

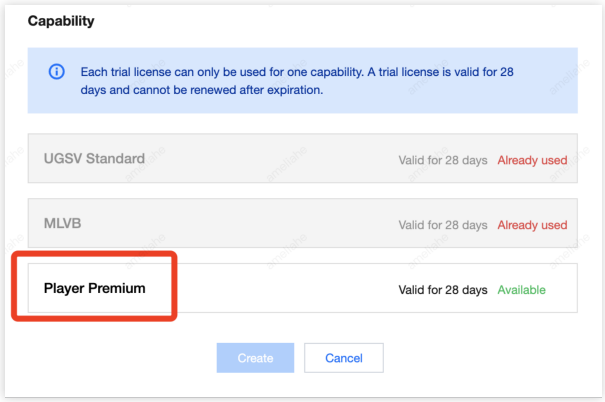


若您想在已创建的测试应用中申请测试**播放器**功能，步骤如下：

1. 登录 **云点播控制台** 或 **云直播控制台** > **License 管理** > **移动端 License**，选择您想测试的应用，单击**测试新功能**。



2. 选择**播放器高级版**，单击**确定**。



说明

测试版 License 有效期内可单击右侧的**编辑**，进入修改 Bundle ID 和 Package Name 信息，单击**确定**即可保存。

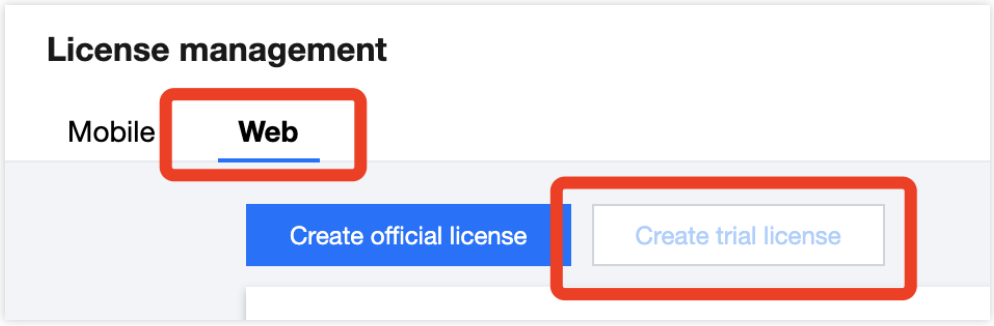
若无 Package Name 或 Bundle Id，可填写“-”。

Web 端测试 License

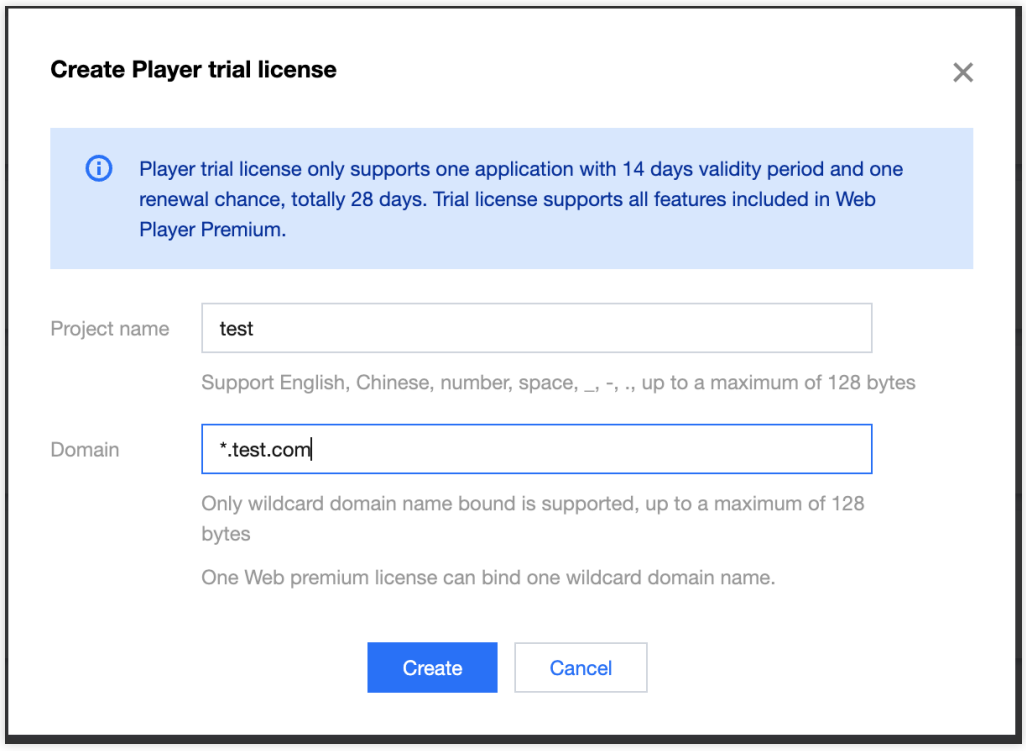
申请测试版 License

您可以免费申请播放器 Web 端测试 License （有效期为14天，可续期1次，共28天）体验测试。

1. 登录 **云点播控制台** 或 **云直播控制台** > **License 管理** > **Web 端**，单击**新建测试 License**。



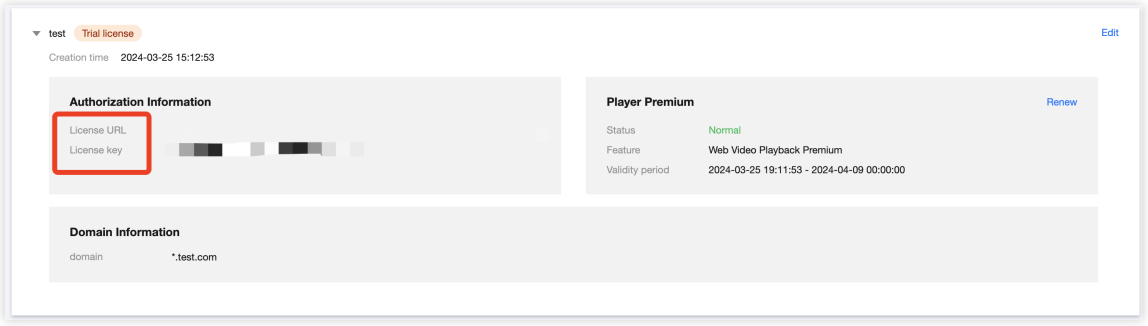
2. 填写**项目名**和**域名**，单击**创建**完成申请。



注意：

仅 5.1.0 及其以上版本的 Web 端播放器 SDK （TCPlayer）支持使用泛域名授权。

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。**



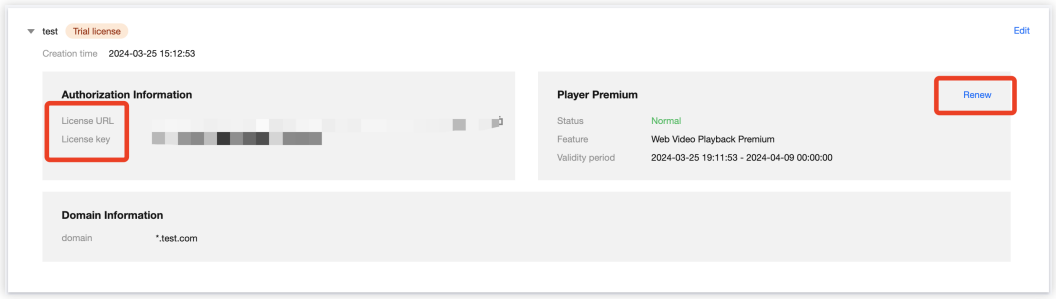
注意：

1个 Web 端高级版 License 仅可关联 1 个泛域名。

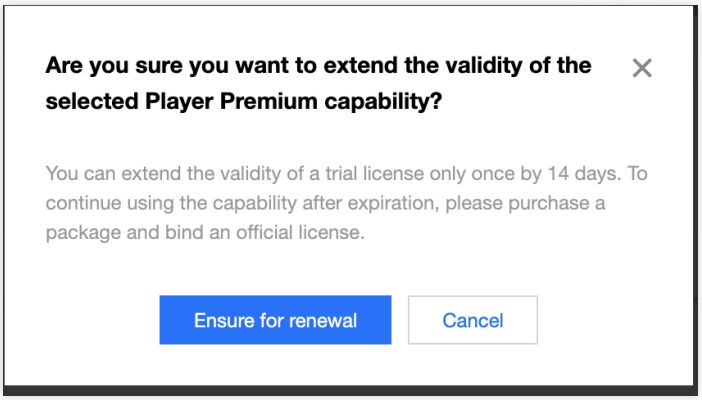
可单击右侧**编辑**修改域名。

续期测试版 License

1. Web 端测试版 License 初次申请默认有效期默认为14天，您可续期**1次**。单击**播放器高级版**功能右侧的**续期**。



2. 单击**确定续期**即可续期该功能14天。



说明：

测试版 License 有效期共28天，**只能续期一次**。若您需继续使用，请 购买 后 绑定 Web 端正式版 License。

购买指南
价格总览

最近更新时间：2024-04-11 16:11:38

播放器 SDK 的费用包含以下几个部分：

费用类型	说明
SDK 授权费用	播放器 SDK的使用授权费用。
其他相关云服务费用	配合 腾讯云点播、腾讯云直播等云端服务使用播放器 SDK 时，对应产生的费用。

说明：
播放器 SDK 移动端高级版和 Web 端高级版均提供累计28天的测试 License，您可以前往控制台免费申请。

SDK 授权费用

播放器 SDK 移动端和 Web 端分开授权，移动端和 Web 端均提供基础版和高级版两种版本，通过购买指定 License 获得对应版本的使用授权。不同端及不同版本的差异详见 产品功能。

获取方式

播放器 SDK 平台	功能版本	所需 License 类型	定价	授权单位
播放器移动端 (iOS & Android & Flutter)	基础版	播放器移动端基础版 License /短视频 License / 直播 License	0 美金（1年，可免费续期） 免费申请	1个 License 可授权一个 iOS 应用包名（Bundle ID）和一个 Android 应用包名（Package Name）
	高级版	播放器移动端高级版 License	499 美金/月 立即购买	
播放器 Web 端	基础版	播放器 Web 端基础版 License	0 美金（1年，可免费续期） 免费申请	精准域名（1个 License 最多可授权10个精准域名）
	高级版	播放器 Web 端高级版 License	99 美金/月 立即购买	泛域名（1个 License 最多可授权1个泛域名）

若您已经购买了以下三种 License，您也可以获得播放器移动端基础版 SDK 授权，无需再额外购买播放器 License。

正式版 License 类型	有效期	价格（美元/年）
短视频 License 精简版	1年（自购买之日起）	1,899
短视频 License 基础版	1年（自购买之日起）	9,999
直播 License	1年（自购买之日起）	5,988

计费规则

每个账号可通过 云点播控制台 或 云直播控制台 免费申请 1 次播放器移动端测试 License 和1次播放器 Web 端测试 License 进行产品体验和测试，首次申请会先提供 14 天有效期，可在控制台中再续期 14 天，共计 28 天。移动端和 Web 端的测试 License 的申请次数和时长各自独立。

移动端一个 License 可以绑定一个 iOS 应用包名（Bundle ID）和一个 Android 应用包名（Package Name），Web 端一个 License 最多可绑定 10 个精准域名；绑定 License 的应用/域名即可获得相应授权，不区分业务环境。若有多款应用需要接入，需要购买对应数量的 License 进行绑定。

域名说明：

精准域名是指某个固定的域名地址，如：`a.com`，`a.b.com`，`a.b.c.com` 等唯一且固定的域名地址。

泛域名是指具有相同域名后缀的一系列域名地址，如泛域名 `*.a.com` 绑定 License 后，`a.a.com`、`b.a.com`、`c.a.com` 等相同域名后缀的域名均可解锁套餐能力。`*` 位置可以支持多级域名的自定义，例如 `b.c.a.com`、`b.c.d.a.com` 等。

应用支持更换绑定的 License，更换应用的授权有效期为新绑定的 License 的有效期。替换下的 License 可用于绑定其他应用，且有效期不变。

退费相关内容参见 退费说明。

新增 License 有效期示例

客户在 2022年07月01日 11:36:59 购买了播放器移动端高级版 License A，并绑定了 iOS 应用 A1 和 Android 应用 A2。随后，客户在 2022年07月02日11:36:59 购买了一个播放器移动端高级版 License，获得了播放器移动端高级版 License B，并绑定了 iOS 应用 B1 和 Android 应用 B2。则：

播放器移动端高级版 License A 有效期为：2022年07月01日11:36:59 到 2022年08月02日00:00:00。

播放器移动端高级版 License B 有效期为：2022年07月02日11:36:59 到 2022年08月03日 00:00:00。

应用 A1、A2、B1、B2 均获得播放器 SDK 移动端高级版功能的使用授权，同时应用 A1、A2 的授权有效期为 License A 的有效期，应用 B1、B2 的授权有效期为 License B 的有效期。

更新有效期示例

在上一示例基础上，同一客户于 2022年08月01日15:48:12 购买了播放器移动端高级版 License C，并使用 License C 为应用 A1、A2 续期，则：

播放器移动端高级版 License C 有效期为：2022年08月01日15:48:12 到 2022年09月02日00:00:00。为应用 A1、A2 续期时将会替换掉原本与应用 A1、A2 绑定的 License A。
因为应用的授权有效期为实际绑定的 License 的有效期，因此续期后应用 A1、A2 的有效期变更为 License C 的有效期，即2022年08月01日15:48:12 到 2022年09月02日00:00:00。此时应用 A1、A2 的授权有效期就从原本的 2022年08月02日00:00:00 延长到了 2022年09月02日00:00:0。
License C 与应用 A1、A2 绑定后，原本与应用 A1、A2 绑定的 License A 将自然解除绑定，同时 License A 的有效期不变，仍为可用的 License 资源，可用于绑定其他的应用。

其他相关云服务费用

根据各云服务的计费规则收取对应费用，未使用相关服务不会产生相关费用。

云点播服务（VOD）

云点播提供视频上传、存储、转码、加速分发播放、版权保护、播放质量监控服务等功能，若您需要使用相关功能，则需开通 云点播 服务。
云点播产品的计费主要包括：
视频存储：上传到云点播服务的视频源文件和转码后的视频文件占用的存储空间，按存储容量计费。
视频转码：存储在云点播服务的视频源文件进行转码处理时，按目标文件的规格和时长计费。
播放分发加速：视频进行播放时，使用云点播分发加速产生的费用，按下行流量计费。
说明：
更多费用相关说明，请参见 云点播价格总览。

云直播服务（CSS）

云直播提供直播播放分发、云端录制、实时转码、实时截图、播放质量监控等功能，若您需要使用相关功能，则需开通 云直播 服务，对应的计费包括：
基础服务费用：正常直播推流接流和直播播放产生的消耗，按照流量/带宽计费。
增值服务费用：包括直播转码（含混流和水印）、直播录制、实时截图、新版直播连麦和拉流转推等增值服务产生的用量消耗。
说明：
更多费用相关说明，请参见 云直播价格总览。

退费说明

最近更新时间：2024-04-11 16:11:37

5 天无理由退款

为了更方便您使用播放器 SDK，如果您购买的 License满足下述条件，则腾讯云支持 5 天内无理由退款：
移动端 License 未绑定应用包名（包含新增和续期两种绑定应用包名的方式），Web 端 License 未绑定域名。
License 购买之日到目前为止不超过 5 天（包含 5 天）。

退款规则

满足 5 天无理由退款规则的订单，可通过 工单 方式提交退款申请，具体参见 退款步骤。
如出现疑似异常或恶意退货，腾讯云有权拒绝您的退货申请。
符合 5 天无理由退款场景的订单，退款金额为购买时花费的全部消费金额，包括现金账户金额、收益转入账户金额以及赠送账户金额。

注意

抵扣或代金券不予以退还。
退还金额将全部原路退回到您的腾讯云账户。

退款步骤

1. 单击 提交工单，进入工单申请页。
2. 在右侧的搜索框中搜索并选择“**其他腾讯云产品**”。
3. 选择问题类型为“**功能咨询**”。
4. 进入工单创建页，填写相应信息并单击**提交工单**即可。

注意

退还资源前，请您查看该资源是否符合 5 天无理由退款，若不符合则无法退还。

欠费停服说明

最近更新时间：2024-04-11 16:18:08

腾讯云计费平台检测到腾讯云账户余额不足时，将提醒您充值。欠费后您有24小时充值时间。若逾期未进行账户充值，云直播、云点播以及直播连麦等功能将暂停提供服务。若您在欠费后24小时内进行账户充值，您的连麦服务将不受影响。

SDK 下载

SDK 下载

最近更新时间：2024-04-11 16:11:38

播放器 SDK

发布日志（Web）

最近更新时间：2024-04-11 16:11:38

TCplayer 更新日志

TCplayer 5.1.0 @ 2023.11.13

新增支持配置原生解码策略
MacOS iOS优化字幕显示效果
全场景支持 loadVideoByID 方法
支持解析泛域名
新增服务端新的错误码和 license 相关的错误码
修复问题

TCplayer 5.0.0 @ 2023.08.16 Breaking Change

新增 License。
新增支持 vr、安全环境检查插件。
快直播 abr 接口调整。
快直播降级逻辑调整，增加自动降级。
开放 m4a。
伪全屏效果优化。
macOS safari 支持私有加密。

TCplayer 4.8.0 @ 2023.04.20

新增支持 P2P。
新增 WebRTC 是否降级的开关。
修复问题。

TCplayer 4.7.2 @ 2023.1.10

修复 iOS 环境判断错误问题。

TCplayer 4.7.0 @ 2022.12.20

新增自动异步加载依赖 SDK。
新增元素检查。
新增支持幽灵水印。
完善代码混淆。

TCplayer 4.6.0 @ 2022.11.04

新增支持多音轨。
新增支持 URL 形式的续播。
修复部分问题。

TCplayer 4.5.4 @ 2022.08.26

新增支持快直播 abr，更新 txliveplayer。
新增支持 av1 in flv，更新 flv.js。
新增支持华曦达 DRM。
新增支持快直播和标准直播统一自动播放被阻止事件到 blocked。
新增支持打点回调事件。
修复部分问题。

TCplayer 4.5.3 @ 2022.06.15

新增支持点播商业级 DRM。

TCplayer 4.5.2 @ 2022.04.15

修复通过劫持 mse 绕过私有加密方案盗取视频的漏洞。
修复部分上报数据问题。

TCplayer 4.5.0 @ 2022.01.14

新增支持标准直播和快直播数据上报功能。
新增支持 MP3 音频格式播放。
新增支持弱网追帧能力。
优化水印功能，支持更多配置项。
修复若干问题。

TCplayer 4.4.0 @ 2021.12.14

新增支持快直播播放。
新增点播场景的数据上报功能。
新增支持通过 v4 接口播放媒体原文件和转码文件。

TCplayer 4.1 @ 2020.07.10

修改默认 hls.js 版本为0.13.2。
支持开启 Key 防盗链功能。
修复其他已知问题。

TCplayer 4.0 @ 2020.06.17

修复试看视频时长保持显示原始时长。
启用后台清晰度配置。
修复其他已知问题。

TCplayerLite 更新日志

TCplayerLite 2.4.1 @ 2021.06.25

新增支持 v1 信令的 WebRTC 的流地址。
增加 webrtcConfig 参数。
增加 WebRTC 卡顿、卡顿结束、推流结束事件。

TCplayerLite 2.4.0 @ 2021.06.03

增加对快直播功能的支持。
修复其他已知问题。

TCplayerLite 2.3.3 @ 2020.07.01

修复 X5 环境下切换全屏时，事件派发异常的问题。
规避 hls 切换源时，相关事件触发时机很慢，导致封面显示异常的问题。

TCplayerLite 2.3.2 @ 2019.08.20

修改默认 hls 版本为0.12.4。
修复其他已知问题。

TCplayerLite 2.3.1 @ 2019.04.26

增加 flvConfig 参数。
默认加载 flv.1.5.js。
修复其他已知问题。

TCplayerLite 2.3.0 @ 2019.04.19

增加部分功能参数选项。
参数 coverpic 改为 poster。
destroy 销毁 flv.js 实例。
修复其他已知问题。

TCplayerLite 2.2.3 @ 2018.12.17

优化播放逻辑。
解决 iOS 微信没有播放事件触发的情况下，出现 loading 动画的问题。

修复其他已知问题。

TCplayerLite 2.2.2 @ 2018.05.03

优化 loading 组件。
优化 Flash destroy 方法。
默认使用 H5 播放。
修复已知问题。

TCplayerLite 2.2.1 @ 2017.12.20

增加可配置清晰度文案功能。
设置默认清晰度。
支持切换清晰度方法。

TCplayerLite 2.2.1 @ 2017.12.07

增加 systemFullscreen 参数。
增加 flashUrl 参数。
修复音量 Max 后进行静音切换的 UI 问题。
修复 iOS 11 微信下需要单击两次才能播放的问题。
修复 safari 11 系统样式被遮挡的问题。
适配在 x5 内核会触发 seeking，但不会触发 seeked 的情况。
修复进度条拖拽到起始位置，设置 currentTime 失败的问题。
切换清晰度保持音量不变。
修复页面宽度为0，播放器宽度判断失败问题。
destroy 方法增加完全销毁播放器节点。

TCplayerLite 2.2.0 @ 2017.06.30

增加控制播放环境判断的参数： Flash、h5_flv、x5_player。
调整播放器初始化逻辑，优化错误提示效果。
增加 flv.js 支持，在符合条件的情况下可以采用 flv.js 播放 FLV。
支持 x5-video-orientation 属性。
增加播放环境判断逻辑，可通过参数调整 H5 与 Flash 的优先级，以及是否启用 TBS 播放。
启用版本号发布方式，避免影响旧版本的使用者。
优化事件触发的时间戳，统一为标准时间。
Bug 修复。

TCplayerLite 2.1.0 @ 2017.03.04

至2017.06.30，经历数次的迭代开发，逐步趋于稳定，目前文档的功能描述中，如果没有特殊说明，皆基于此版本。

Adapter 插件更新日志

Adapter 插件发布 @ 2021.07.16

首次发布播放器 Adapter 版

发布日志（iOS & Android）

最近更新时间：2024-08-07 15:14:34

播放器 SDK

播放器 SDK 12.0 @ 2024.08.01

新功能：

Android&iOS：HEVC 降级播放时，新增对外抛出对应事件 `VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK（2031）` 。

Android&iOS：新增对外抛出收到首帧数据包事件 `VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET（2017）` 。

缺陷修复:

Android&iOS：修复 MP4 试看视频切换到正片播放场景下，播放链接带 exper 导致的缓存复用错误问题。

Android&iOS：修复 HLS 加密视频预加载和自适应播放异常问题。

Android&iOS：修复高并发播放场景下可能出现的内存泄露问题。

iOS：修复通过 fileId 多次发起预下载可能引起崩溃的问题。

iOS：修复播放质量监控上报丢失 app_version 字段问题。

iOS：修复选择字幕轨道后回掉错误码丢失问题。

iOS：修复调用 `TXVodPlayer#stopPlay` 后再添加字幕， 重新启动播放后字幕数据丢失问题。

iOS：修复播放过程中，退到后台一段时间后返回前台， 出现从头开始播放的问题。

iOS：修复预下载失败后，内存不释放的问题。

播放器 SDK 11.9 @ 2024.06.03

缺陷修复:

Android&iOS： 优化首帧耗时统计，使统计更加准确。

Android&iOS： 修复播放器下载器模块在多线程场景下有概率出现 crash 问题。

Android：去掉 `NetworkInfo#getExtraInfo` 调用，避免被误检测出读取 ssid。

播放器 SDK 11.8 @ 2024.05.06

新功能：

Android&iOS：支持 HLS EVENT 直播源（高级版支持）。

Android：支持播放 `content://` 和 `asset://` URI本地视频。

Android&iOS：支持精准和非精准 Seek。

缺陷修复:

Android：修复 TXVodPlayer#setAutoPlay 参数为 false 时，收到 PLAY_EVT_VOD_PLAY_PREPARE 事件后回调进度的问题。

iOS： 修复在画中画播放过程中，不调用 exitPictureInPicture 而直接 stopPlay 导致的异常问题。

播放器 SDK 11.7 @ 2024.03.04

新功能：

Android&iOS：新增播放音量均衡控制功能（高级版支持）。

Android&iOS：HECV 自适应降级播放（高级版支持）。

Android&iOS：支持 HLS EVENT 直播源（高级版支持）。

播放器 SDK 11.6 @ 2024.01.10

新功能：

Android&iOS：播放器高级版本升级网络内核，性能更优（高级版支持）。

Android&iOS：预加载支持 FileId 加密视频（高级版支持）。

Android&iOS：FileId 视频播放支持幽灵水印。

Android&iOS：支持 SEI 信息回调（高级版支持）。

Android&iOS：HLS 的视频格式支持 Program Date Time 实时获取和 Seek 能力（PDT，高级版支持）。

功能优化：

Android&iOS：首帧事件额外携带首帧时长信息。

Android&iOS：播放器高级版本修复内置字幕解析异常问题。

Android&iOS：使用抓包工具无需设置 http 代理绕过 localhost。

缺陷修复:

Android&iOS：修复断网后，播放重试次数失效问题。

Android&iOS：修正部分 mp3 seek 很慢问题。

Android&iOS：修复直出 AES-128 加密 m3u8 离线播放异常问题。

Android：修复短时间内多次 seek，只有首次 seek 生效的问题。

iOS：修复内置字幕导致画中画播放异常的问题

播放器 SDK 11.4 @ 2023.08.29

新功能：

iOS：新增高级画中画能力，支持画中画播放加密视频，离线播放画中画，退后台自动画中画。

功能优化：

Android&iOS：缓存相关接口支持 KB 粒度控制。

Android&iOS：优化播放器的网络调度策略。

缺陷修复:

Android&iOS：修复播放停止后，不能及时停止网络流量消耗问题。

播放器 SDK 11.3 @ 2023.07.07

新功能：

Android&iOS：视频预下载支持指定媒资类型（TXPlayInfoParams#mMediaType），减少底层的类型探测，提升下载效率。

iOS：增加播放过程中网络异常重试机制。

功能优化：

Android&iOS：优化断网场景下播放内存申请过快的问题。

缺陷修复:

Android&iOS：修复断网后播放 HLS 视频，出现重复播放某个片段的问题。

Android&iOS： 修复设置的 preloadSize 较小导致概率播放视频失败问题。

iOS： 修复 httpDns 服务在弱网络下偶现 Crash 的问题。

播放器 SDK 11.2 @ 2023.06.01

新功能：

Android&iOS：点播播放补充命中缓存事件（VOD_PLAY_EVT_HIT_CACHE）回调。

Android：DRM 播放可通过 TXPlayerGlobalSetting#setDrmProvisionEnv 配置 COM 或 CN 证书提供商地址。

功能优化：

Android&iOS：可通过 TXVodPlayConfig#setMediaType 指定播放媒资类型，减少底层的类型探测，提升启播速度。

Android&iOS：补充 MP4 播放过程中音频码率（VIDEO_BITRATE&AUDIO_BITRATE）信息回调。

缺陷修复:

Android&iOS：修复 onNetStatus 回调 VIDEO_CACHE 值缺失问题。

Android&iOS：修复离线下载资源在某些场景下播放失败问题。

Android&iOS：修复快速切视频概率播放失败问题。

播放器 SDK 11.1 @ 2023.04.07

功能优化:

Android&iOS： 新增“判断已下载完成的视频资源是否存在”接口（TXVodDownloadMediaInfo#isResourceBroken）。

Android&iOS： 视频下载支持私有加密视频在防盗链过期之前共享缓存。

缺陷修复:

Android&iOS：修复音频帧解码失败后 CPU 使用率增大的问题。

Android&iOS：修复下载异常退出时进度丢失问题修复。

Android&iOS：修复部分 MP4 文件播放异常问题。

播放器 SDK 11.0 @ 2023.02.24

功能优化:

Android&iOS：优化音视频交织不好的视频播放兼容能力。

iOS：优化视频帧数据内存释放不及时问题。

缺陷修复:

iOS：修复循环播放单次播放结束没有回调 VOD_PLAY_EVT_LOOP_ONCE_COMPLETE（单次循环播放结束）事件异常。

Android&iOS：修复视频下载获取的子流文件大小（TXVodDownloadMediaInfo#getSize）不准确问题。

Android&iOS：修复视频下载嵌套 m3u8 单流下载异常问题。

Android&iOS：修复视频下载嵌套 m3u8 私有加密视频概率播放异常问题。

播放器 SDK 10.9 @ 2022.12.30

新功能:

Android&iOS：支持外挂 HttpDns，解决播放过程中域名被劫持导致播放失败的问题。

Android&iOS：V2TXLivePlayer 支持播放 WebRTC。

功能优化:

Android： 视频下载功能 TXVodDownloadMediaInfo 新增获取网络下载速率 speed 字段。

缺陷修复:

iOS：修复画中画反复切换导致切换失败的问题。

iOS：修复视频下载完成，接口获取还是未完成状态问题。

iOS：修复iOS16上播放画面卡住不动的问题。

Android&iOS：修复某些机型上由于视频帧率过大，导致播放失败的问题。

Android&iOS：修改网络异常场景下， 播放失败耗时过长的问题。

播放器 SDK 10.8 @ 2022.10.27

功能优化:

Android&iOS：循环播放单轮结束增加 VOD_PLAY_EVT_LOOP_ONCE_COMPLETE 事件。

Android：合规优化启动时调用2次：NetworkInfo.getExtraInfo 问题。

缺陷修复:

Android&iOS：修复特殊场景导致私有加密视频播放失败问题。

Andoird&iOS：修复部分视频通过 gzip 传输播放失败问题。

Andoird&iOS：修复播放结束后进度条时长不匹配问题。

iOS：修复 appid&fileid 播放 v2 协议取视频源地址错误问题。

播放器 SDK 10.7 @ 2022.09.20

功能优化:

Android&iOS：点播播放 startPlay 接口变更为 startVodPlay。

Android&iOS： 直播播放 startPlay 接口变更为 startLivePlay。

iOS：修复长时间退到后台，返回播放器，无法继续播放的问题。

Android： 修复低版本 Android 系统部分视频播放失败问题。

播放器 SDK 10.6 @ 2022.08.31

功能优化:

Android&iOS：fileid 播放方式新增雪碧图、URL 等信息回调。

Android&iOS：包体大小优化

缺陷修复

iOS：修复部分场景下私有加密视频离线下载播放失败问题。

播放器 SDK 10.5 @ 2022.08.12

缺陷修复：

Android&iOS：修复播放失败不带视频格式后缀短链异常。

播放器 SDK 10.4.0 @ 2022.07.21

功能优化:

Android&iOS：HLS 直播支持自适应播放。

缺陷修复:

Android：修复 onNetStatus 和进度回调间隔异常 。

Android：修复播放器没有调用 setConfig 引起的空指针异常。

iOS：修复部分场景下重播卡顿问题。

播放器 SDK 10.3.0 @ 2022.07.06

新功能：

iOS：视频播放支持画中画模式。

缺陷修复：

Android：修复硬解后台连续播放视频列表会中断问题。

Android&iOS：修复 Seek 完成事件不回调问题。

播放器 SDK 10.2.0 @ 2022.06.23

功能优化：

Android&iOS：优化播放过程中回调 cachedBytes、IP 地址等参数。

缺陷修复：

Android&iOS：修复硬解播放H265格式视频失败问题。

Android&iOS：修复播放 HLS 直播异常。

iOS：修复某些场景下获取 supportedBitrates 异常。

播放器 SDK 10.1.0 @ 2022.05.31

Android&iOS： 视频超分效果优化

Android&iOS：修复嵌套 m3u8 refer header 子流传递问题

iOS： 解决与第三方 SDK ffmpeg 冲突问题

Android&iOS： 优化播放器内核性能

播放器 SDK 9.5.29040 @ 2022.05.13

Android&iOS：修复播放带封面 mp3 失败的问题。

播放器 SDK 9.5.29036 @ 2022.05.06

Android：修复 SurfaceView 重复 Add 和 Remove 导致黑屏问题。

播放器 SDK Android 9.5.29035, iOS 9.5.29036 @ 2022.04.28

Android&iOS：新增视频预下载功能。
Android&iOS：支持 onPrepared 事件之前播放器暂停（Pause）能力。
Android&iOS：支持 Pause 状态下切码流继续保持 Pause 状态。
Android&iOS：优化播放性能。

播放器 SDK 9.5.29016 @ 2022.03.30

Android&iOS： 支持缓存流量精细化控制，预加载 buffer 和启播 buffer 可以分开控制。
Android&iOS： 支持启播前指定偏好分辨率播放，找最适合的分辨率启播。

播放器 SDK 9.5.29015 @ 2022.03.25

Android：优化播放性能。

播放器 SDK 9.5.29011 @ 2022.03.10

iOS： 优化版本适配问题。

播放器 SDK 9.5.29009 @ 2022.03.03

Android&iOS：支持终端极速高清，可通过插件接入。
Android&iOS：优化视频私有加密。
Android&iOS：优化精准 seek 到帧。
Android&iOS：HLS 支持 EXT-X-DISCONTINUITY 标签。
Android&iOS：优化播放器内核， 提升性能。
Android&iOS：播放器组件提供沉浸式短视频、Feed 视频流、视频试看、视频封面和视频动态水印等功能 Demo。

播放器 SDK 9.5 @ 2022.01.11

Android：修复视频列表中任意视频播放到最后连续切换两次清晰度会重播的问题。
Android&iOS：修复不同时间点回看播放时间点不准的问题。

播放器 SDK 9.4 @ 2021.12.09

iOS：修复切换 HLS 码流出现黑屏的问题。
iOS：修复点播播放器在播放视频的时候频繁 seek 会有杂音的问题。
Android：修复防盗链雪碧图获取失败的问题。

Android：修复点播播放器离线下载 HLS 偶现报错的问题。
Android&iOS：修复播放器精准 seek 不准的问题。

播放器 SDK 9.3 @ 2021.11.04

Android&iOS：修复点播播放器开启预加载，调用 startPlay 后听到声音异常的问题。
Android&iOS：修复点播播放器硬解播放 hevc 视频时回调分辨率为0的问题。

播放器 SDK 9.2 @ 2021.09.26

Android&iOS：点播放器支持 HLS 加固加密播放。
Android：修复播放特殊字符的地址失败的问题。
Android：修复设备频繁切换前后台，偶现有声无画的问题。

播放器 SDK 9.1 @ 2021.09.02

Android：修复 Android5.x 特定设备出现播放崩溃的问题。
Android：优化直播播放特定条件下视频画面过曝的问题。

播放器 SDK 9.0 @ 2021.08.06

iOS：修复在开启 smoothSwitchBitrate 后，反复切换清晰度导致的 Crash 问题。
iOS：优化点播播放器在网络恢复后播放进度异常的问题。

播放器 SDK 8.9 @ 2021.07.15

Android：修复点播播放器断网后的回调事件逻辑错误的问题。

播放器 SDK 8.8 @ 2021.06.21

iOS：修复点播播放器启动停止播放多次触发内存泄漏的问题。
Android：修复在 Android 11上播放 HLS 文件的报错问题。
Android：修复播放器播放默认直播卡顿，其他直播偶现音画加速的问题。
Android&iOS：修复 VodPlayer 播放特定视频 seek 慢的问题。
Android&iOS：修复点播暂停播放后，调 seek 设置进度，画面显示慢的问题。

播放器 Adapter

播放器 Adapter 1.4.0 @ 2023.04.18

Android&iOS：支持对点播 CDN 的加密进行解密。

播放器 Adapter 1.2.0 @ 2022.03.10

Android&iOS：支持通过 FileId 播放自适应码流、转码和原始视频。

播放器 Adapter 发布 @ 2021.07.22

首次发布 iOS & Android 播放器组件 Adapter。

发布日志（Flutter）

最近更新时间：2023-07-13 14:37:32

播放器 SDK Flutter 端 V11.3.0 @ 2023.07.07

移除已经废弃的直播时移接口。
修复已知问题

播放器 SDK Flutter 端 V11.2.0 @ 2023.06.05

flutter 开发环境最低需求版本提升。
修复已知问题。

播放器 SDK Flutter 端 V11.1.1 @ 2023.05.08

移除非必要的三方依赖。
修复已知问题。

播放器 SDK Flutter 端 V11.1.0 @ 2023.04.10

原生桥接层改造。
修复已知问题。

播放器 SDK Flutter 端 V11.0.0 @ 2023.02.28

增加离线下载 demo。
修复雪碧图错乱问题、IOS 画中画打开失败等已知问题。

播放器 SDK Flutter 端 V10.9.0 @ 2023.01.03

修复 Android 端画中画不支持播放加密视频的问题。
修复直播、点播暂停后，退出后台回到应用界面，会自动播放的问题。

优化 log 输出，方便定位问题。

播放器 SDK Flutter 端 V10.8.0 @ 2022.10.20

调整播放器组件画中画交互，支持开启画中画后能返回到上一级页面。

播放器 SDK Flutter 端 V10.7.0 @ 2022.09.20

点播播放 startPlay 接口变更为 startVodPlay。
直播播放 startPlay 接口变更为 startLivePlay。
播放器组件 playWithModel 接口变更为 playWithModelNeedLicence。

播放器 SDK Flutter 端 V1.0.3 @ 2022.07.05

Android&iOS 端新增画中画（PIP) 功能。

播放器 SDK Flutter 端 V1.0.2 @ 2022.06.24

新增含UI集成方案SuperPlayer 组件
完善直播和点播 SDK 接口。
修复通过 fileId 和 psign 播放失败问题。

播放器 SDK Flutter 端发布 @ 2021.07.16

首次发布 Flutter 端播放器 SDK。

License 指引
新增与续期 License

最近更新时间：2024-05-15 17:21:08

概述

播放器 SDK 提供直播播放和点播播放能力，移动端和 Web 端各自分开独立授权计费，您需要获取对应 License 后方可使用对应功能。
License 获取和使用方式如下：

1. **购买 License**
您可参考 产品功能 和 购买指南 确认并购买您需要的 License。若您需要申请测试 License 进行体验，可参见 免费测试 指引。

2. **绑定 License**
购买完 License 后，您需将您新购的 License 与您需要授权的应用/域名进行绑定，以实现对应的应用/域名的授权。移动端以应用包名为单位授权，Web 端以域名为单位授权，移动端绑定操作 和 Web 端绑定操作 指引见下文。

3. **配置 License**
完成绑定后您将在控制台获得授权凭证 **License URL** 和 **License Key**，您在集成 SDK 的过程中需传入对应信息，请妥善保管。具体传入方式参见各端的集成文档。

注意：
直播 License 和短视频 License 包含了播放器移动端基础版 License 全部能力，因此也可用于解锁播放器 SDK 移动端基础版的功能。直播 License 相关参见 License 计费购买 和 新增与续期，短视频 License 相关参见 License 计费购买 和 新增与续期。

移动端 License 新增与续期

购买正式版 License

在进行 License 绑定前，您可参考以下方式获取播放器 License ：

License 类型	获取方式	价格（美金）	有效期
播放器移动端基础版 License	免费申请	0	1 年
播放器移动端高级版 License	直接购买	499	1 个月

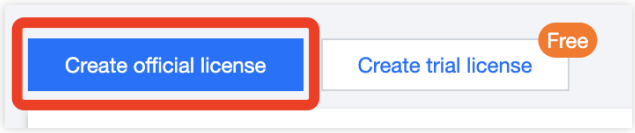
绑定正式版 License

购买后，您需要在 云点播控制台 或 云直播控制台 对应用进行 License 绑定使其生效。您可以选择**新建正式应用并选择播放器 License** 或在**已创建的应用中解锁播放器功能并绑定 License** 两种方式进行正式版 License 绑定 。

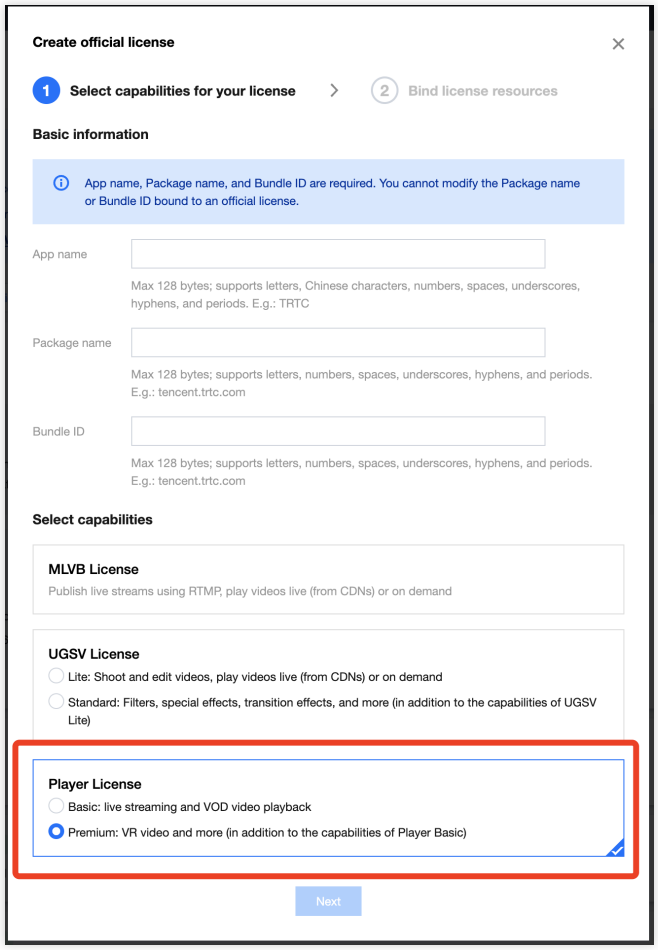
方式一：新建正式应用并选择播放器 License

方式二：已创建的正式版应用中解锁播放器功能并绑定 License

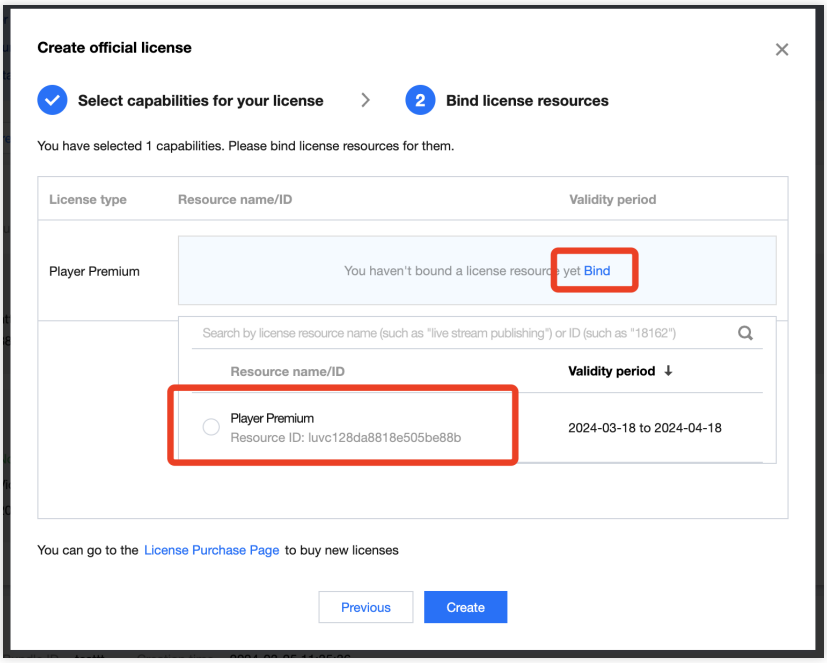
1. 进入 云点播控制台 或 云直播控制台> **License 管理 > 移动端 License**，单击**新建正式 License**。



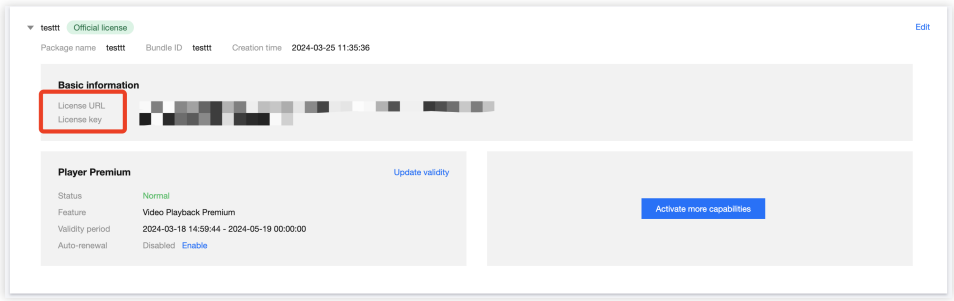
2. 填写正式应用的 `App Name` 、 `Package Name` 和 `Bundle ID` 信息，选择**播放器 License**，选择**基础版**或**高级版**，单击**下一步**。



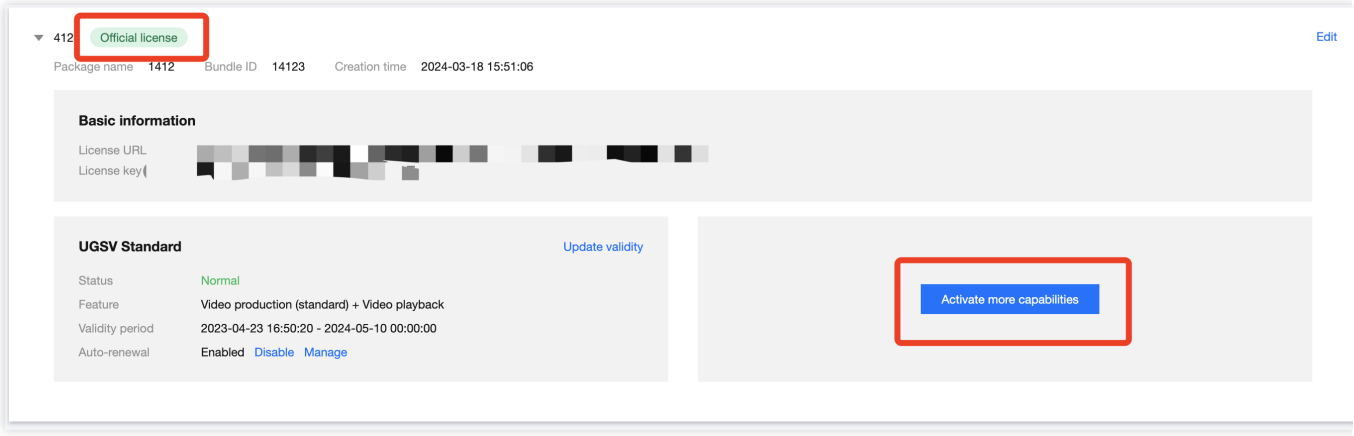
3. 进入**选择 License 资源并绑定**界面，单击**立即绑定**，选择**未绑定**的播放器 License（若没有可绑定的 License 资源，可参考 购买播放器 License），并单击**创建**即可创建应用并生成正式版 License。



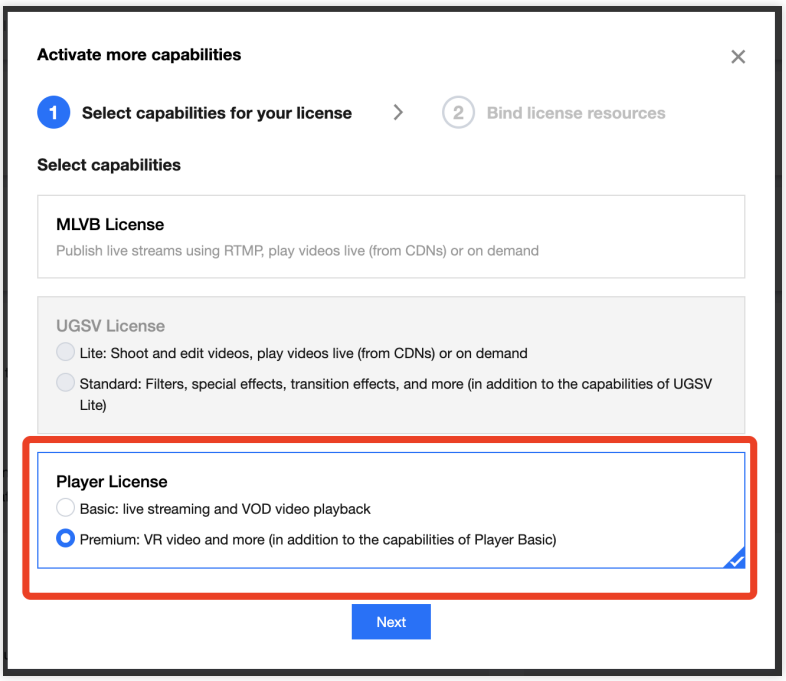
4. 正式版 License 成功创建后，页面会显示生成的正式版 License 信息。在 SDK 初始化配置时需要传入 License URL 和 License Key 两个参数，请妥善保存以下信息。参见 配置查看 License 指引在 SDK 内传入您的 License URL 和 License Key 即可完成 License 授权。



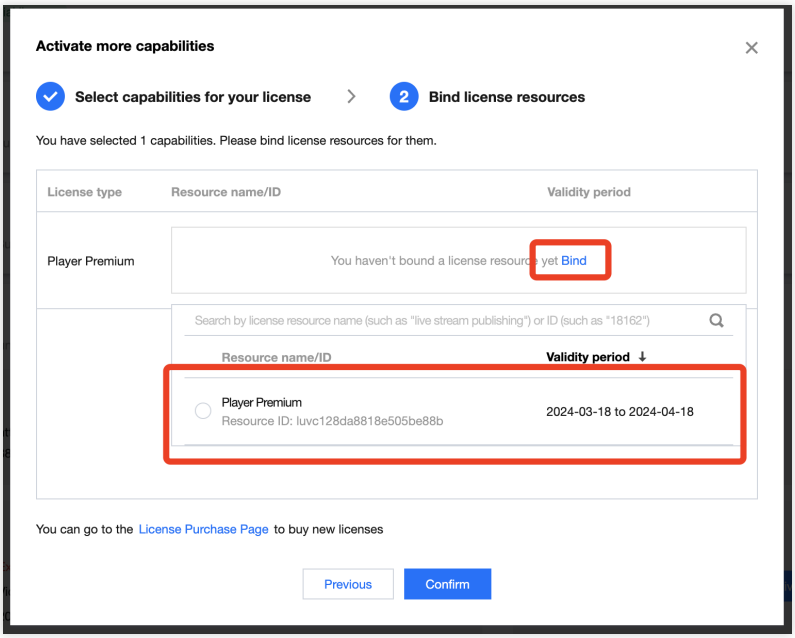
1. 进入 **云点播控制台** 或 **云直播控制台**> **License 管理** > **移动端 License**。
2. 选择您需要增加播放器功能的正式应用，单击**解锁新功能**。



3. 选择**播放器 License**，单击**下一步**。



4. 进入**选择 License 资源并绑定**界面，单击**立即绑定**，选择**未绑定**的播放器 License（若没有可绑定的 License 资源，可参考 购买播放器 License），并单击**确定**即可在应用下生成正式版播放器功能。



说明：

单击**确定**前需要再次确认 Bundle ID 和 Package Name 与业务使用包名信息一致，如与提交到商店的不一致，请在提交前进行修改，**正式版 License 一旦提交成功将无法再修改 License 信息**。

更新正式版 License 有效期

您可以登录**云点播控制台** 或 **云直播控制台**> **License 管理** > **移动端 License** 页面查看播放器移动端正式版 License 的有效期，也可通过在 消息订阅 中订阅音视频终端 SDK，配置**站内信/邮件/短信**等消息接收渠道，接收正式版 License 到期提醒，播放器正式版 License 将在到期时间距离当前时间为32天、7天、3天、1天时各向您发送一次到期提醒，提示您及时续费以免影响正常业务运行。

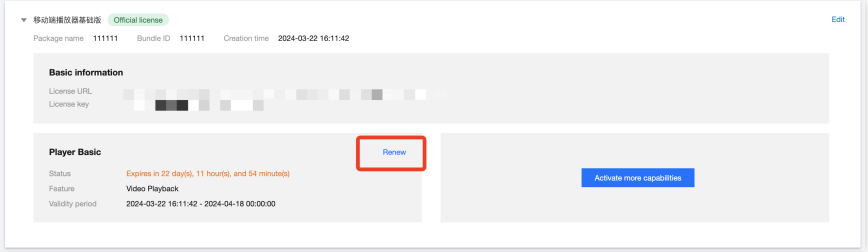
若您的播放器移动端正式版 License 已到期，请参考下方指引完成 License 续费：

更新基础版 License 有效期

移动端**基础版**为免费使用，选择您需要更新有效期的 License，剩余有效期 30 天内可单击**续期**免费延长有效期。

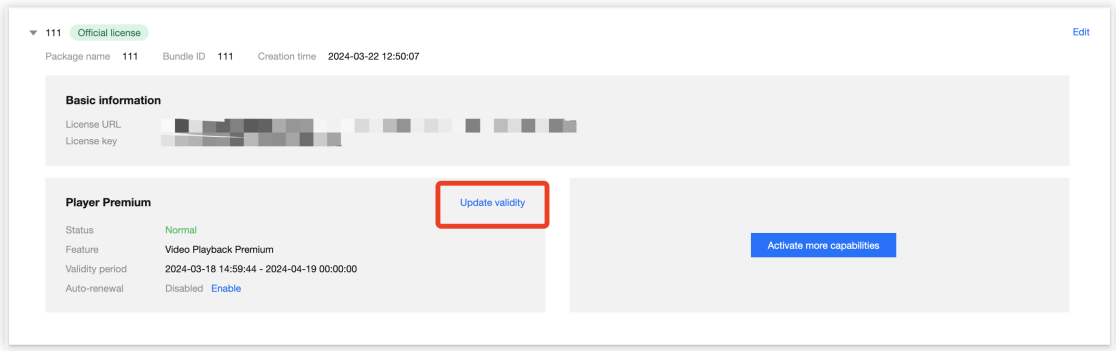
注意：

播放器移动端基础版 License 不支持开启自动续费。



更新高级版 License 有效期

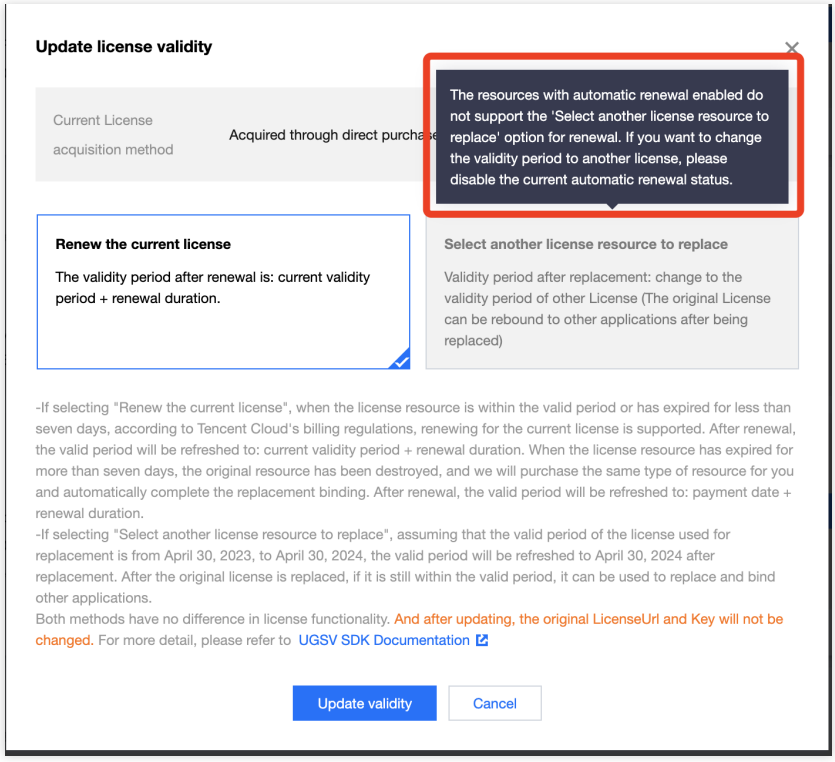
1. 选择您需要更新有效期的 License，单击**播放器**功能内的**更新有效期**。



2. 您可以选择**为当前 License 续费**和**选择其他 License 资源替换**两种方式更新 License 有效期，具体如下。

注意：

已开启自动续费的资源不支持选择其他 License 资源替换进行续期，若您想变更为其他 License 的有效期，请将当前自动续费状态关闭。



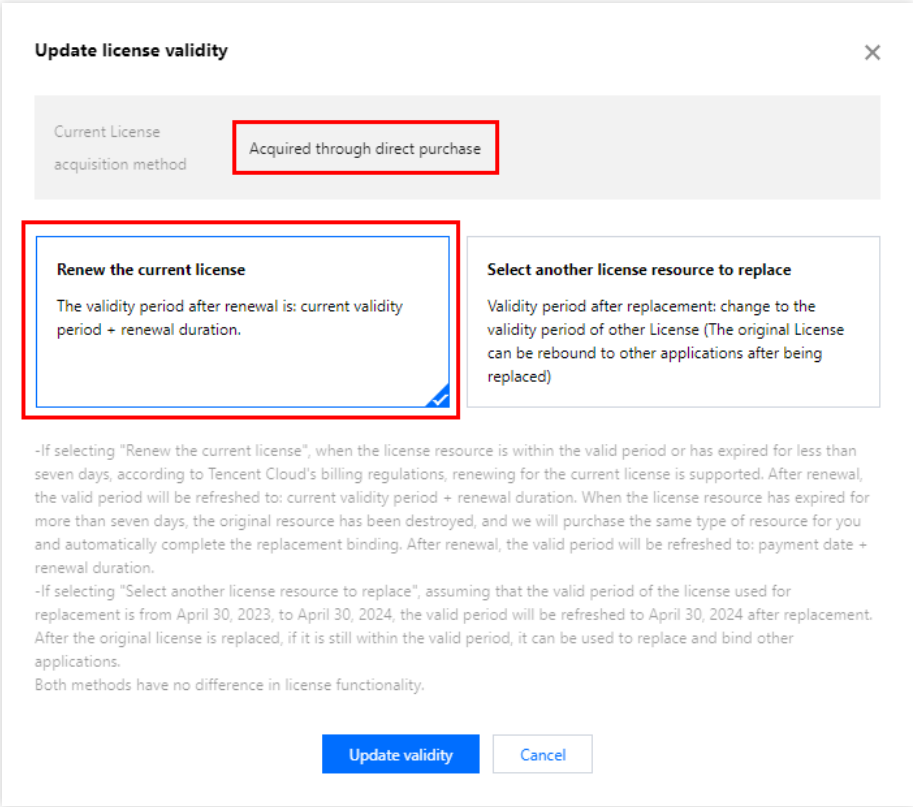
为当前 License 续费

选择其他 License 资源替换

1. 点击

为当前 License 续费

，点击**更新有效期**。



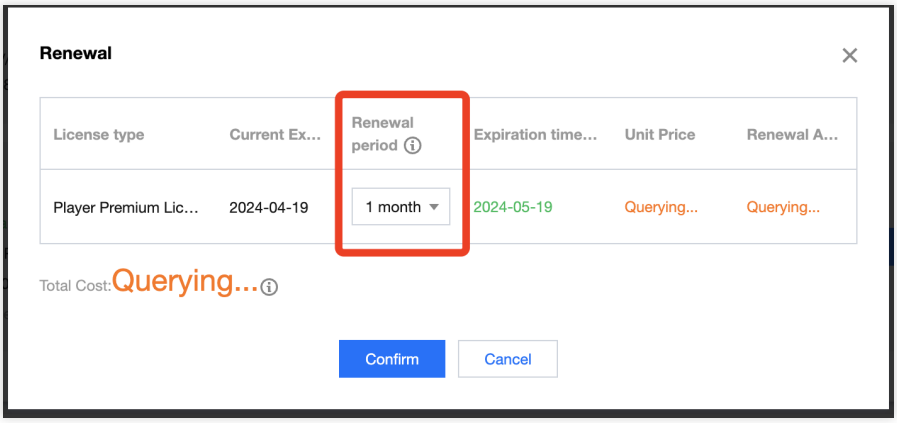
注意：

若选择**为当前 License 续费**：

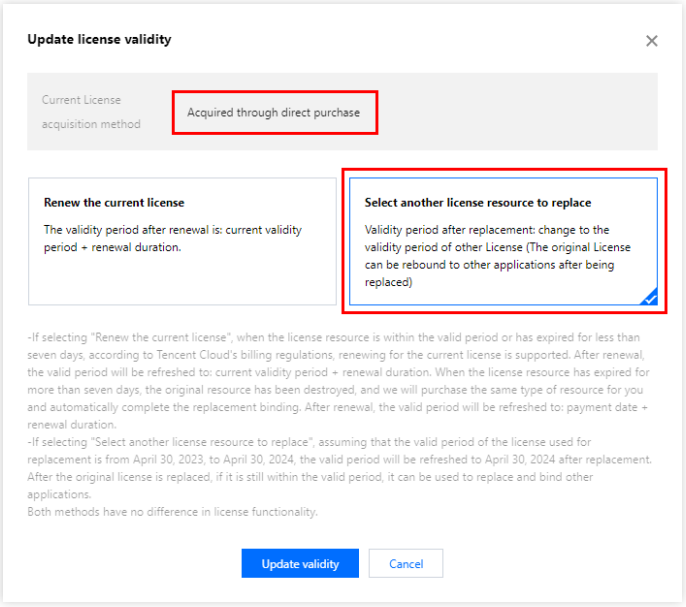
当 License 资源**在有效期内或者资源过期未超过七天**时，根据腾讯云计费相关规定，都支持为当前 License 续费，续期后有效期刷新为：**当前有效期 + 续费时长**；

当 License 资源**过期超过七天**时，原有资源已经销毁，我们将为您新购相同类型资源并自动完成替换绑定，续期后有效期刷新为：**付款日期 + 续费时长**。

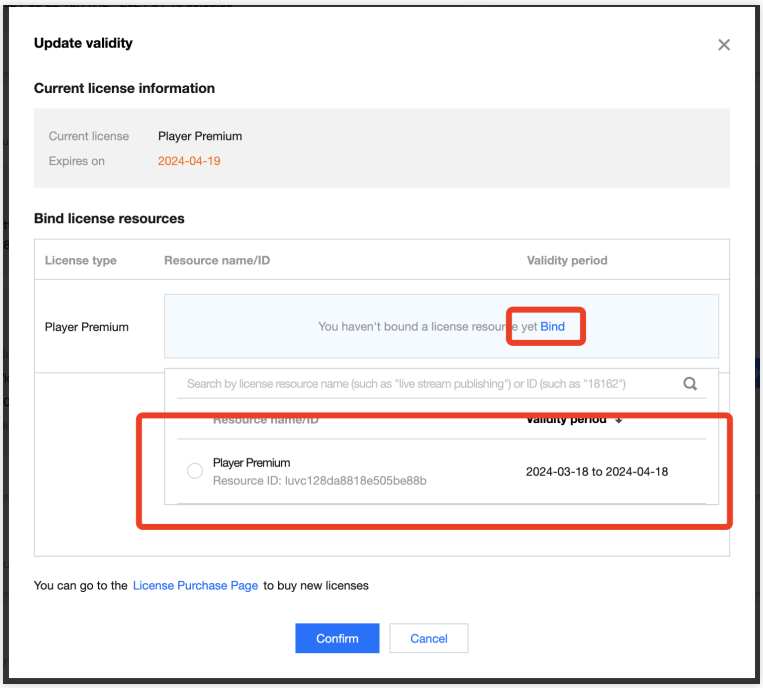
2. 在续费界面**选择续费时长**，播放器基础版 License **以月**为周期续费。点击**确定续费**延长 License 有效期。



1. 点击
选择其他 License 资源替换
，点击**更新有效期**。



2. 在**更新功能有效期**界面，点击**立即绑定**，选择**未绑定**的播放器高级版 License （若没有可绑定的 License 资源，可前往 音视频终端 License 购买页 购买），单击**确定**即可。



3. 查看更新后的有效期情况。

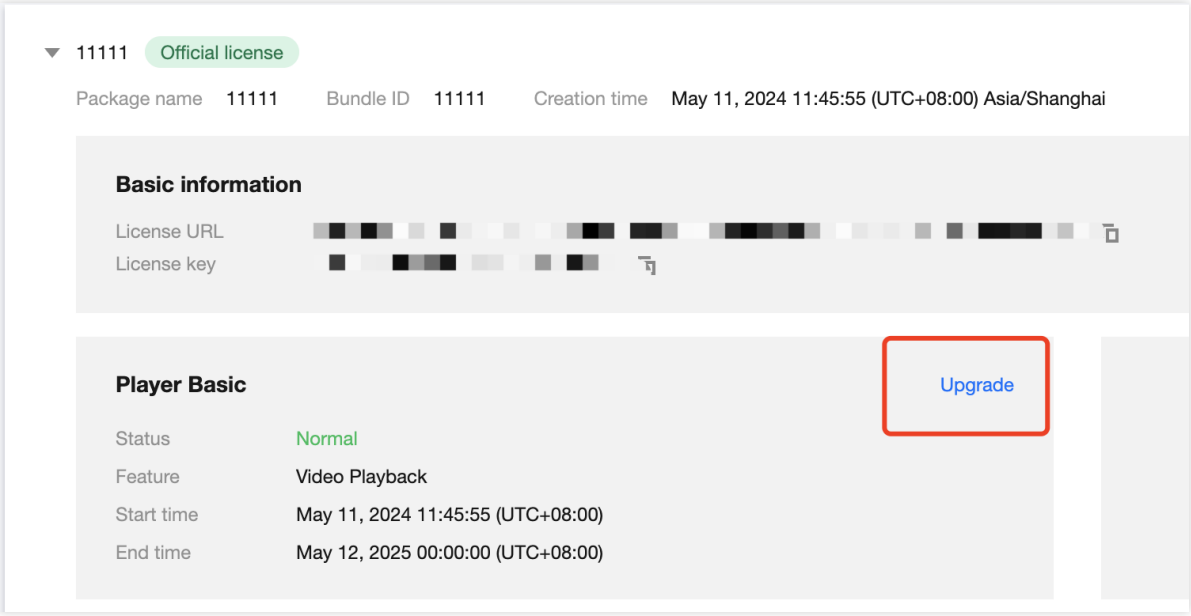
注意：

播放器正式版 License 不支持信息修改，若您需要修改 License 信息，购买 License 后请勿用于 License 有效期的更新，请单击**新增 License** 重新新增 License 绑定新的包名信息。

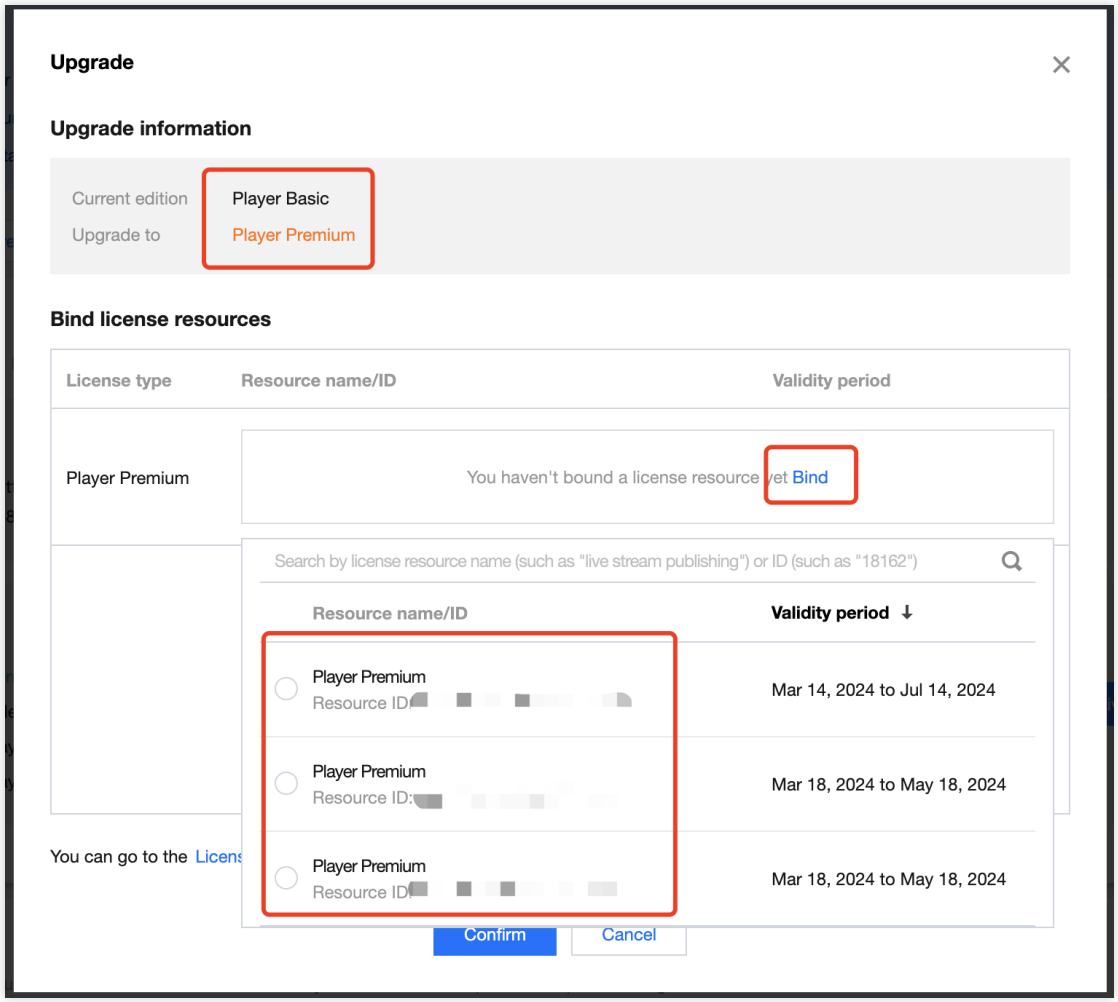
基础版升级为高级版 License

若您已开通播放器**移动端基础版 License**，且需要支持外挂字幕、高级画中画组件等功能，您可以通过以下操作指引升级为播放器**移动端高级版 License**，解锁更多功能：

1. 选择需要升级的正式**移动端基础版 License**，单击**播放器基础版**功能内的升级。



2. 进入升级功能界面，单击**立即绑定**，选择需要绑定的播放器高级版 License，单击**确定**即可升级到播放器**移动端高级版 License**。



Web 端 License 新增与续期

购买正式版 License

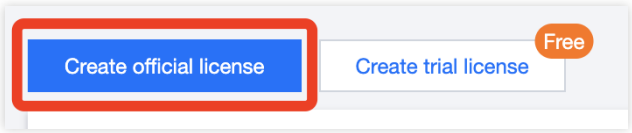
在进行 License 绑定前，您可参考以下方式获取播放器 License ：

License 类型	授权单位	获取方式	价格（美金）	有效期
播放器 Web 端基础版 License	精准域名（1 个 License 最多可授权 10 个精准域名）	免费申请	0	1 年
播放器 Web 端高级版 License	泛域名（1 个 License 最多可授权 1 个泛域名）	直接购买	99	1 个月

绑定正式版 License

购买后，您需要在 云点播控制台 或 云直播控制台 > **License 管理** > **Web 端 License** 对应用进行 License 绑定使其生效。

1. 进入 **Web 端 License**，单击**新建正式 License**。



2. 选择**版本**。播放器 Web 端正式版 License 包括**基础版**和**高级版**。

基础版可免费申请，有效期一年；仅支持**精准域名**绑定，最多可关联 **10个**。

高级版为包月付费使用；仅支持**泛域名**绑定，可关联 **1个**。

新增基础版 License

新增高级版 License

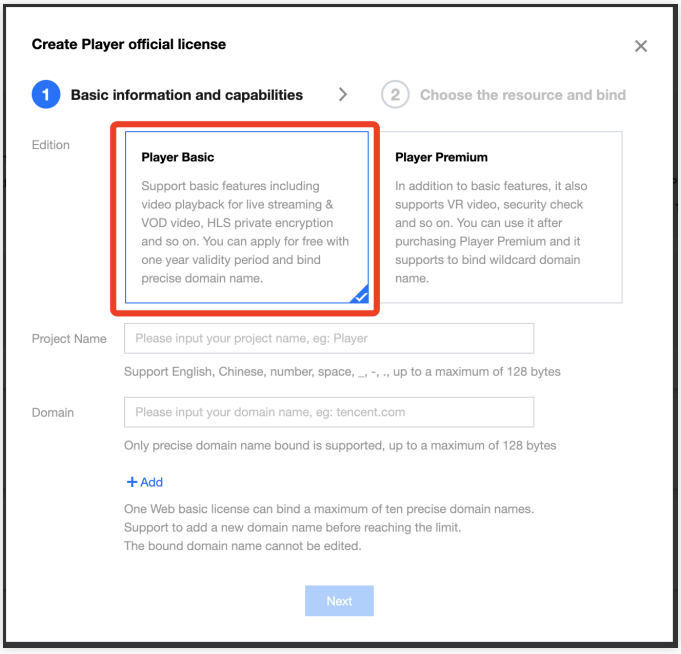
注意：

1 个 Web 端基础版 License 最多可关联 10 个精准域名。

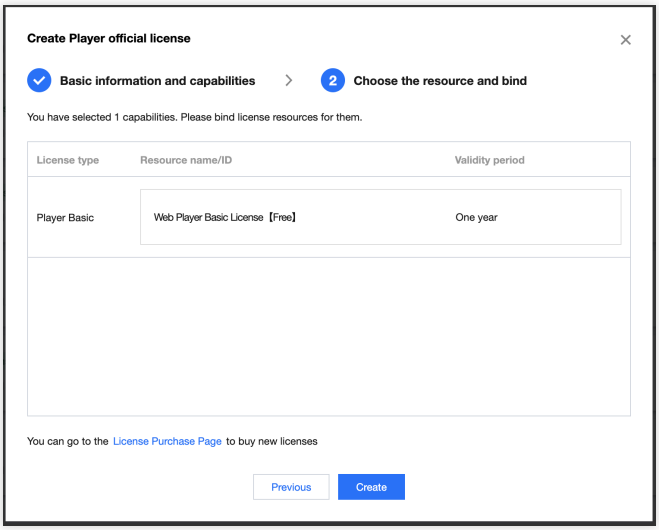
到达次数上限前随时可以新增域名。

已绑定域名无法修改。

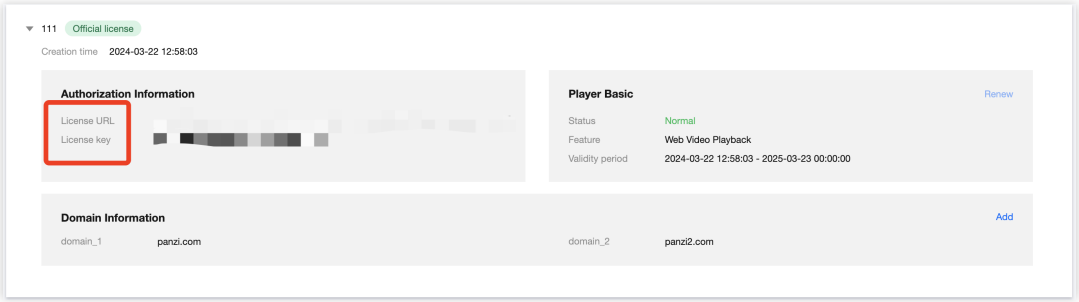
1. 选择**播放器基础版**，填写**项目名**和**域名**，单击**下一步**。



2. 自动选择 **1 年免费 Web 播放器基础版 License** 资源，单击**创建**即可创建应用并生成基础版 License。



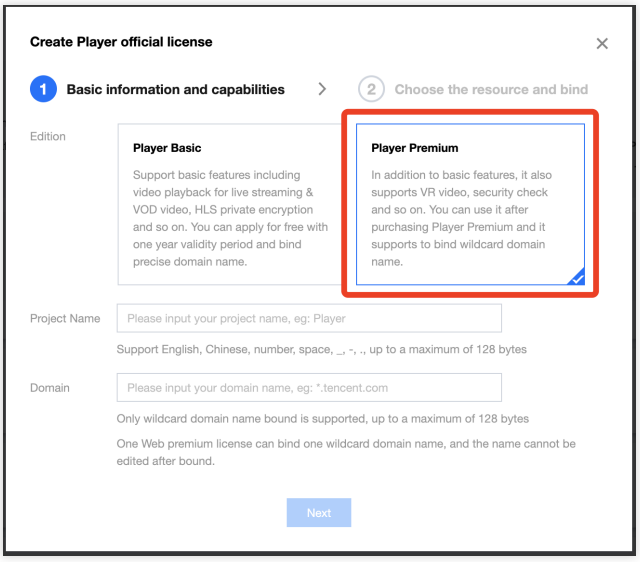
3. 正式版 License 成功创建后，页面会显示生成的正式版 License 信息。在 SDK 初始化配置时需要传入 License URL 和 License Key 两个参数，请妥善保存以下信息。



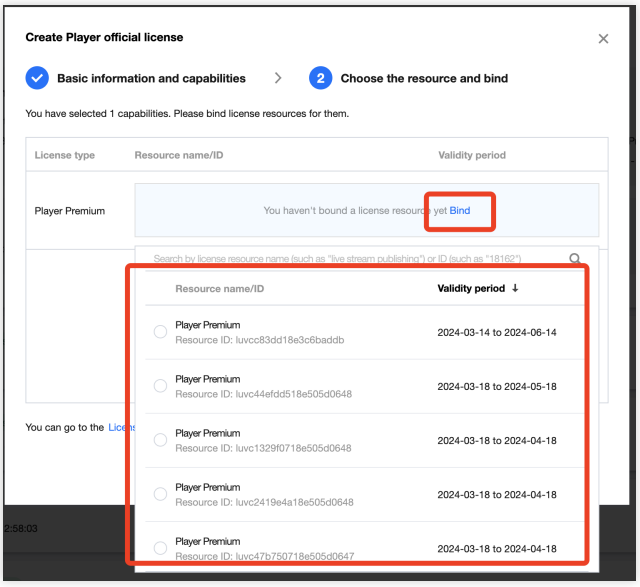
注意：

1 个 Web 端高级版 License 可关联 1 个泛域名，域名一经绑定无法修改。

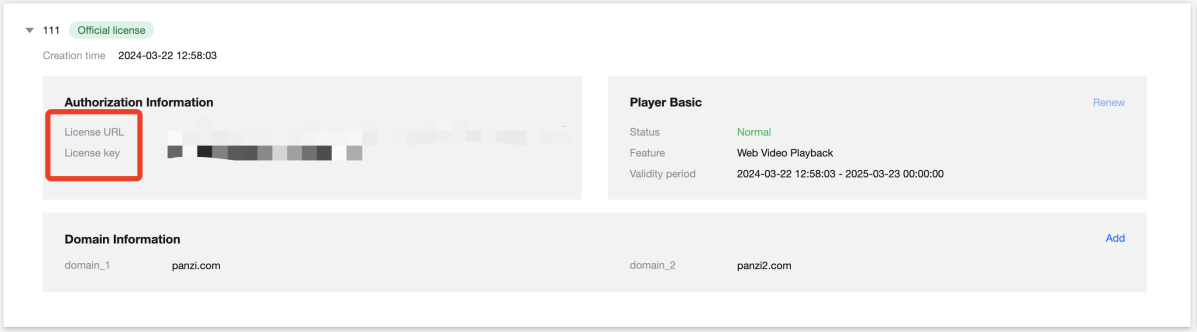
1. 选择**播放器高级版**，填写项目名和域名，单击**下一步**。



2. 进入**选择 License 资源并绑定**界面，单击**立即绑定**，选择**未绑定**的播放器 License（若没有可绑定的 License 资源，可参考 购买播放器 License），并单击**创建**即可创建应用并生成高级版 License。



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

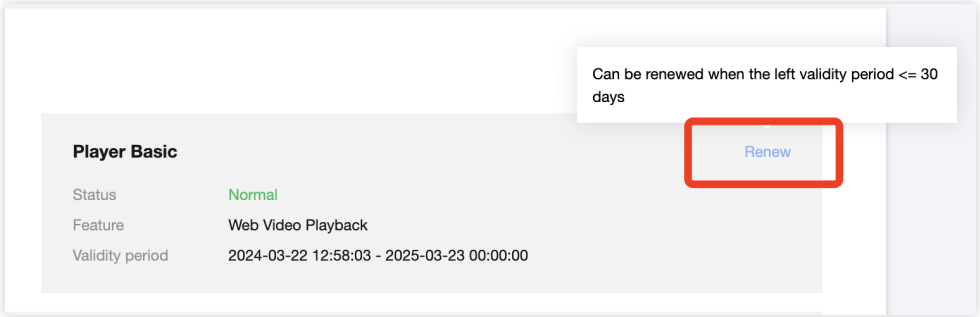


更新正式版 License 有效期

您可以登录 **云点播控制台** 或 **云直播控制台** > **License 管理** > **Web 端 License** 页面查看播放器 Web 端正式版 License 的有效期，也可通过在 消息订阅 中订阅音视频终端 SDK，配置**站内信/邮件/短信**等消息接收渠道，接收正式版 License 到期提醒，播放器正式版 License 将在到期时间距离当前时间为32天、7天、3天、1天时各向您发送一次到期提醒，提示您及时续费以免影响正常业务运行。
若您的播放器 Web 正式版 License 已到期，参考下文指引进行续期。

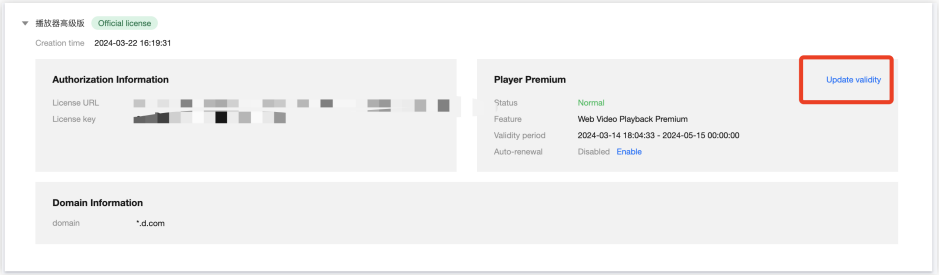
基础版更新 License 有效期

Web 端**基础版**为免费使用，选择您需要更新有效期的 License，剩余有效期 30 天内可单击**续期**免费延长有效期。
注意：
播放器 Web 端基础版 License 不支持开启自动续费。



高级版更新 License 有效期

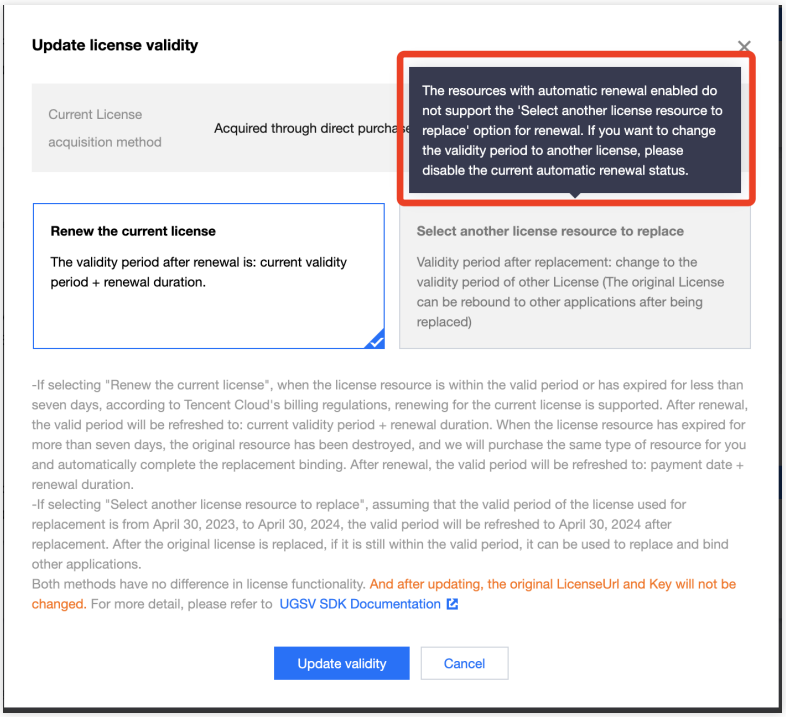
1. 选择您需要更新有效期的 License，单击**播放器高级版**功能内的**更新有效期**。



2. 播放器高级版 License 支持**为当前 License 续费**和**选择其他 License 资源替换** 2种更新有效期的方式，具体如下。

注意：

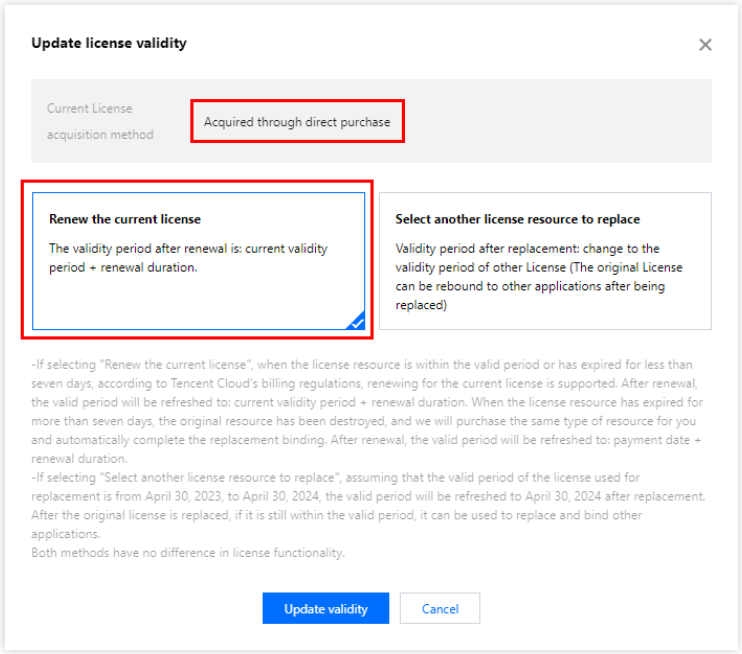
已开启自动续费的资源不支持选择其他 License 资源替换进行续期，若您想变更为其他 License 的有效期，请将当前自动续费状态关闭。



为当前 License 续费

选择其他 License 资源替换

1. 点击**为当前 License 续费**，点击**更新有效期**。



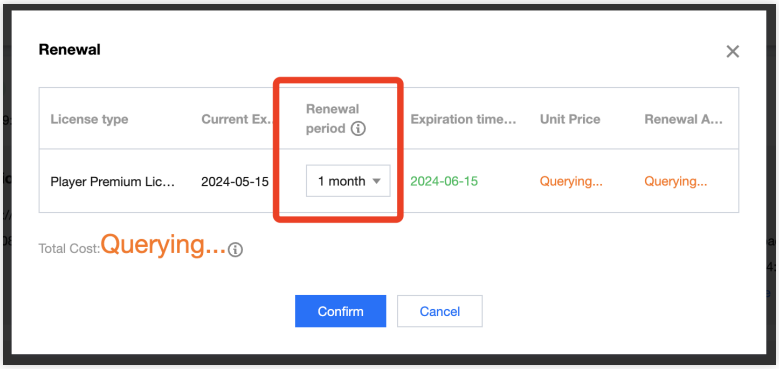
注意：

若选择**为当前 License 续费**：

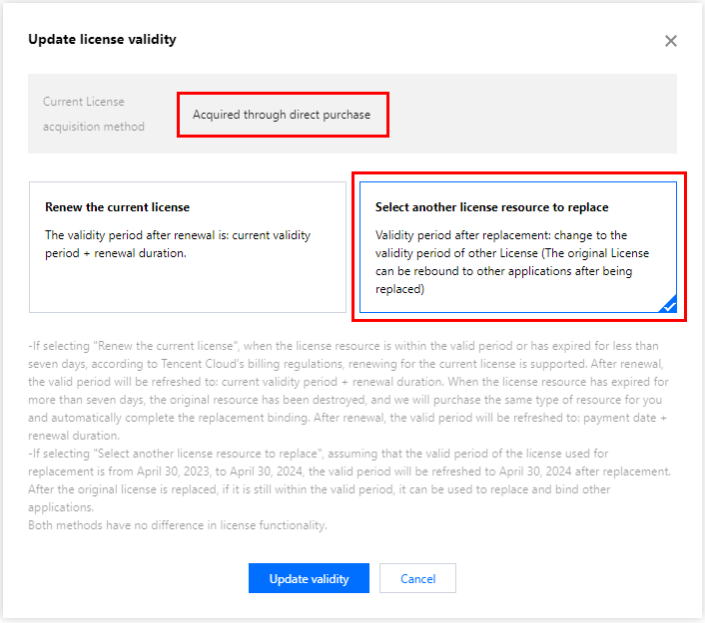
当 License 资源**在有效期内或者资源过期未超过七天**时，根据腾讯云计费相关规定，都支持为当前 License 续费，续期后有效期刷新为：**当前有效期 + 续费时长**；

当 License 资源**过期超过七天**时，原有资源已经销毁，我们将为您新购相同类型资源并自动完成替换绑定，续期后有效期刷新为：**付款日期 + 续费时长**。

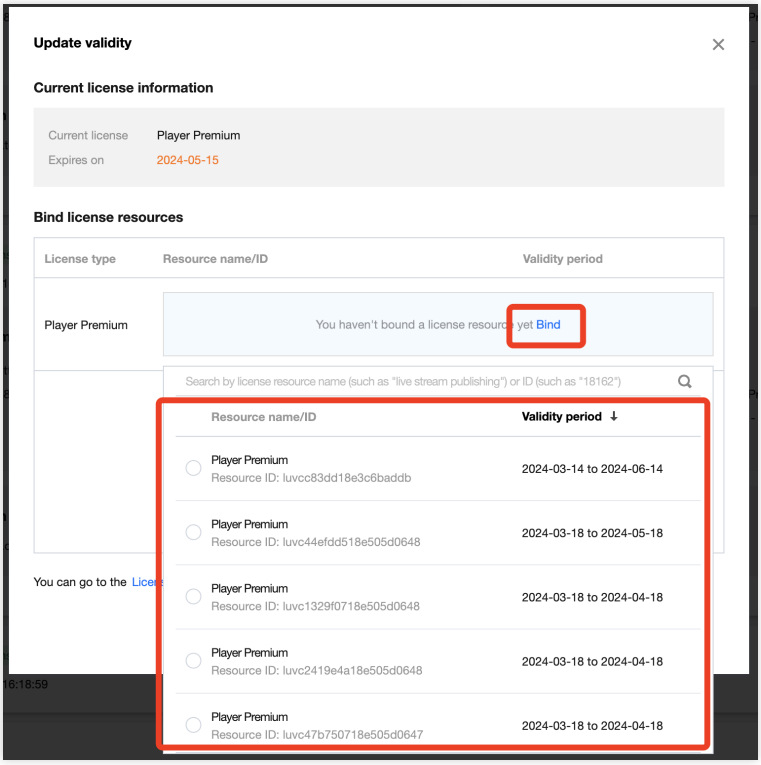
2. 在续费界面**选择续费时长**，播放器高级版 License **以月**为周期续费。点击**确定续费**延长 License 有效期。



1. 点击**选择其他 License 资源替换**，点击**更新有效期**。



2. 在**更新功能有效期**界面，点击**立即绑定**，选择**未绑定**的播放器高级版 License（若没有可绑定的资源包，可前往音视频终端 License 购买页 购买），单击**确定**即可。



3. 查看更新后的有效期情况。

注意：

播放器正式版 License 不支持信息修改，若您需要修改 License 信息，购买资源包后请勿用于 License 有效期的更新，请单击**创建应用并绑定 License** 重新创建应用新增 License 绑定新的包名信息。

自动续费

您可以通过**控制台管理自动续费**和**费用中心管理自动续费** 2种方式管理自动续费，具体如下。

控制台管理自动续费

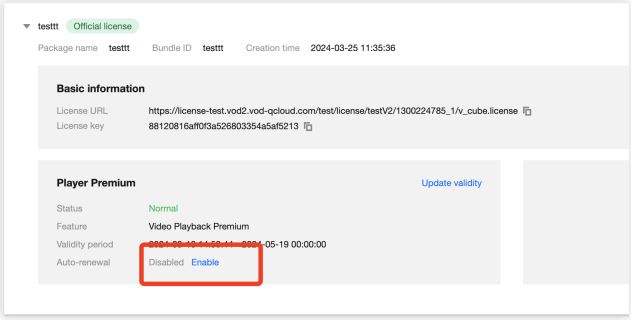
费用中心管理自动续费

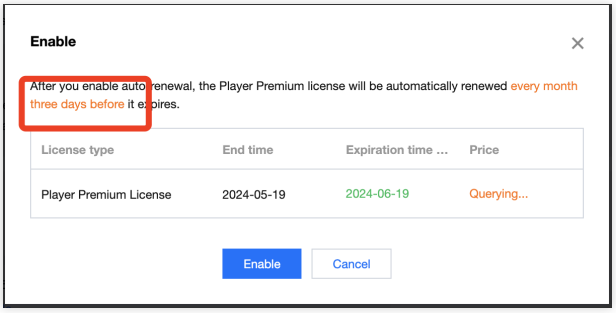
License 支持开启自动续费，开启自动续费的 License 资源**到期前 3 天将按月进行自动续费**，开启前需保证账户可用余额的充足，否则可能导致续期失败影响您的使用。

登录 云点播控制台 **或** 云直播控制台 **> License 管理** 页面，找到您需要管理自动续费的 License ：

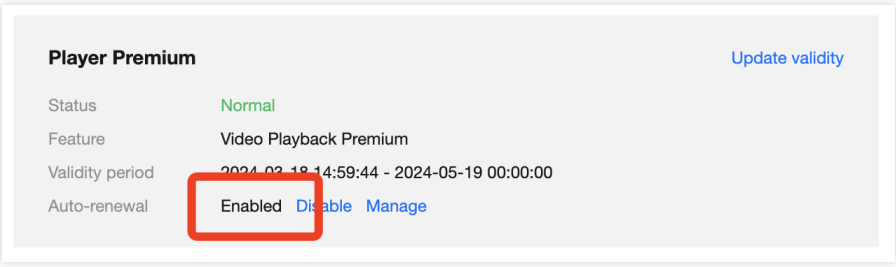
1. 开启自动续费。

1.1 License 的自动续费在**未开启状态**下可点击**开启自动续费**，到期前3天将**按月**自动扣款并续期。

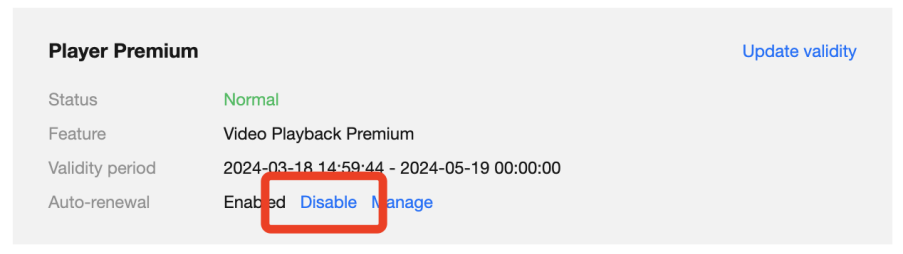


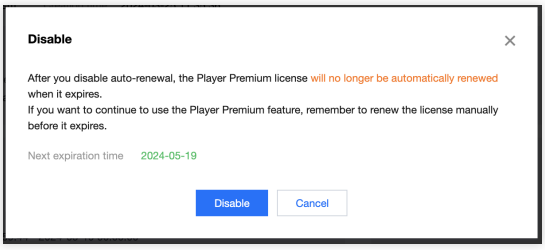


1.2 自动续费状态转为已开启。



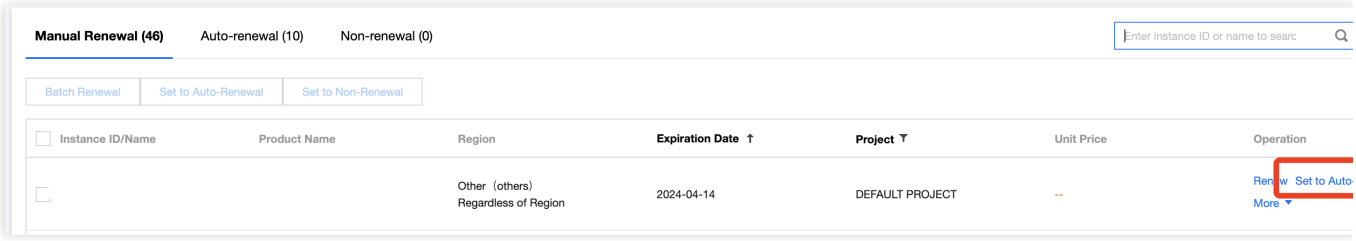
2. 关闭自动续费。License 的自动续费在**已开启状态**下可点击**关闭自动续费**，到期后将不再自动续费。





您可以前往 续费管理 将资源设为自动续费。

在右侧搜索框中搜索**播放器**，找到对应资源，单击**设为自动续费**即可。



配置查看 License

最近更新时间：2024-04-18 17:06:54

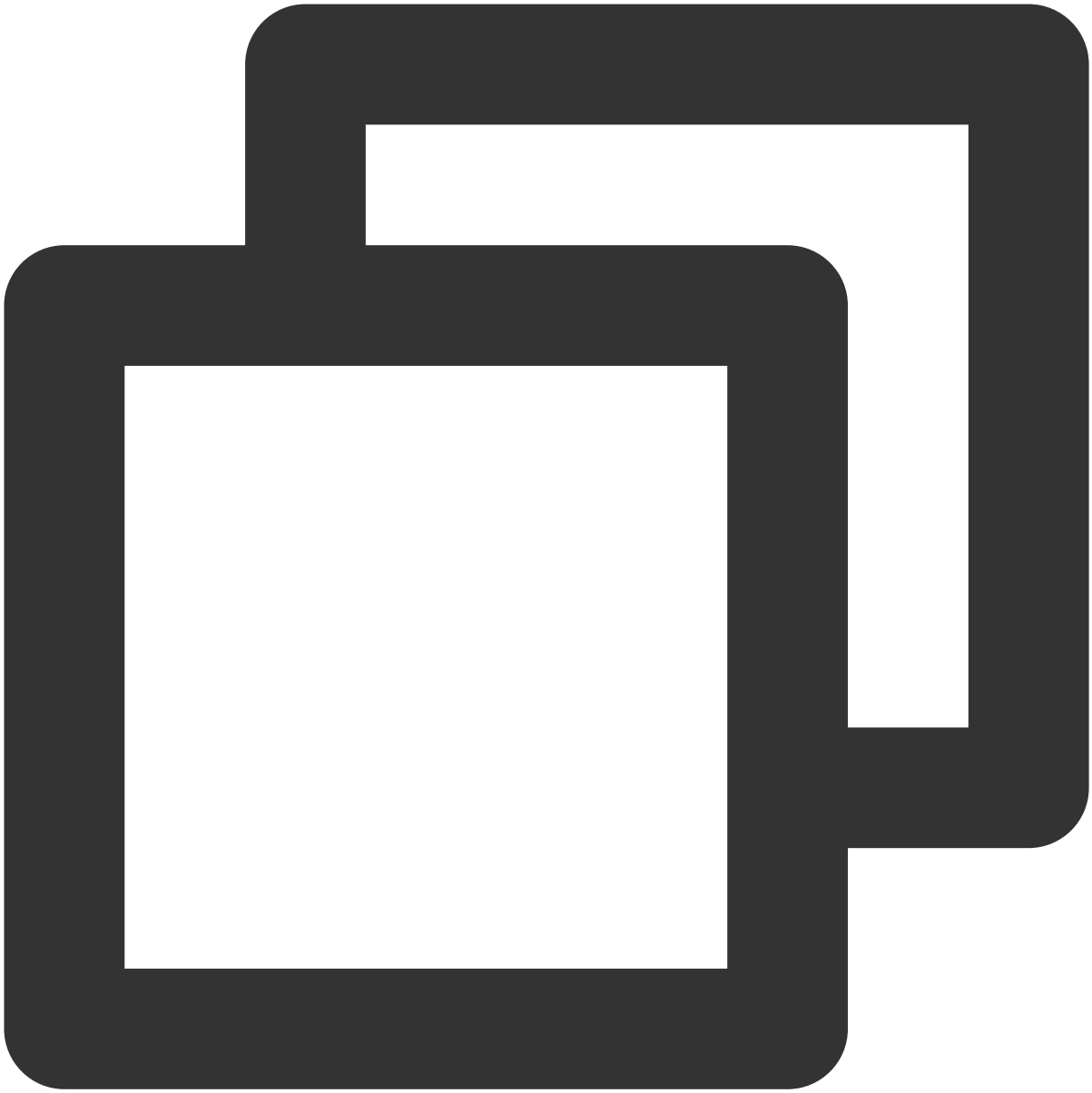
视频播放 License

配置方法

调用 SDK 的相关接口前，您需要调用如下方法配置 License：

iOS

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

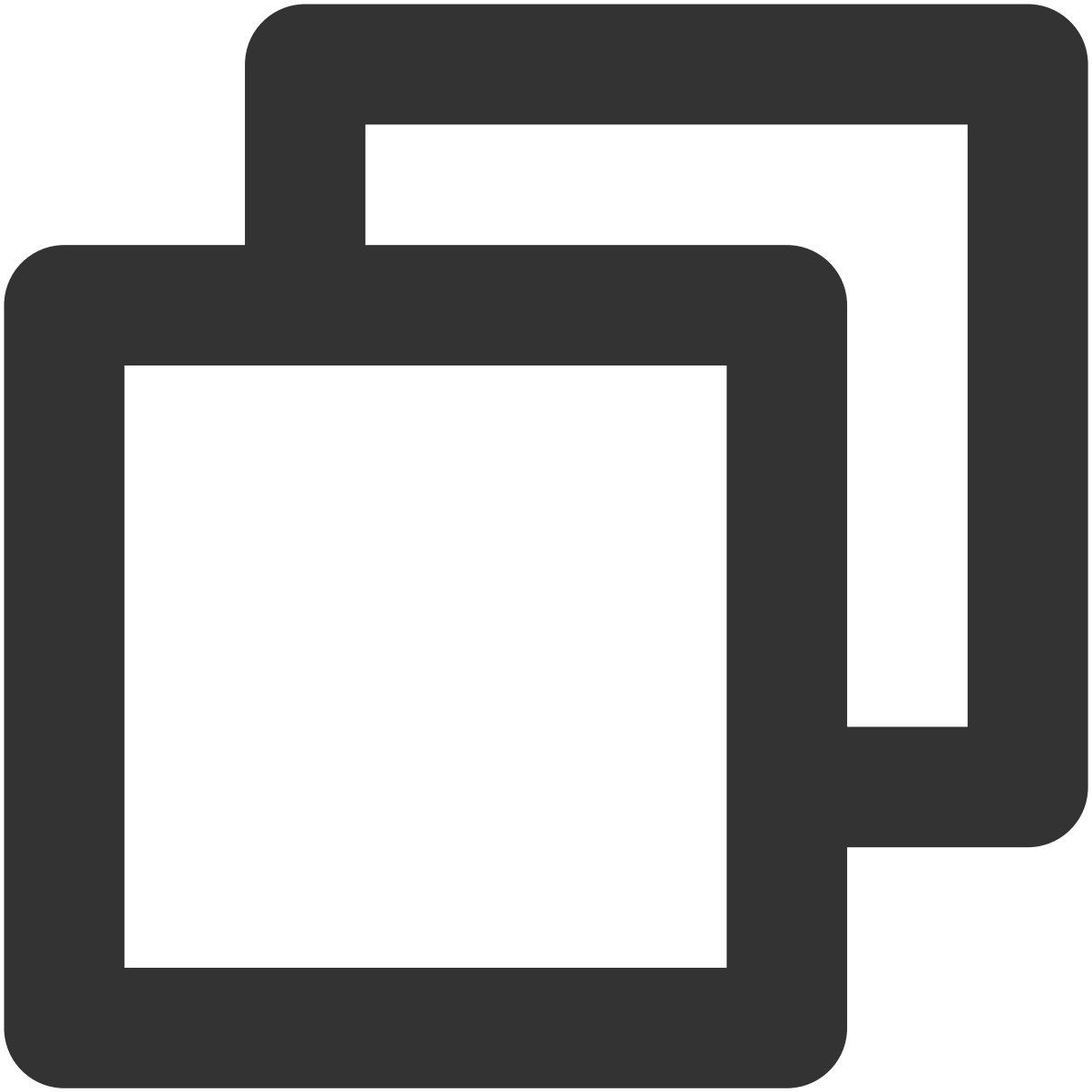


```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    NSString * const licenceURL = @"<获取到的licenseUrl>";
    NSString * const licenceKey = @"<获取到的key>";

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

```
#pragma mark - TXLiveBaseDelegate
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
    // 如果 result 不为 0，表示设置失败，需要进行重试
    if (result != 0) {
        [TXLiveBase setLicence:licenceURL key:licenceKey];
    }
}
@end
```

Android

建议在 application 中添加：



```
public class MApplication extends Application {
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // 获取到的 licence url
        String licenceKey = ""; // 获取到的 licence key
        TXLiveBase.getInstance().setLicence(appContext, licenceURL, licenceKey);
        TXLiveBase.setListener(new TXLiveBaseListener() {
            @Override
            public void onLicenceLoaded(int result, String reason) {
                Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reas
                if (result != 0) {
```

```
                        // 如果 result 不为 0，表示设置失败，需要进行重试
                        TXLiveBase.getInstance().setLicence(appContext, licenceURL, lic
                    }
                }
            });
        }
    }
```

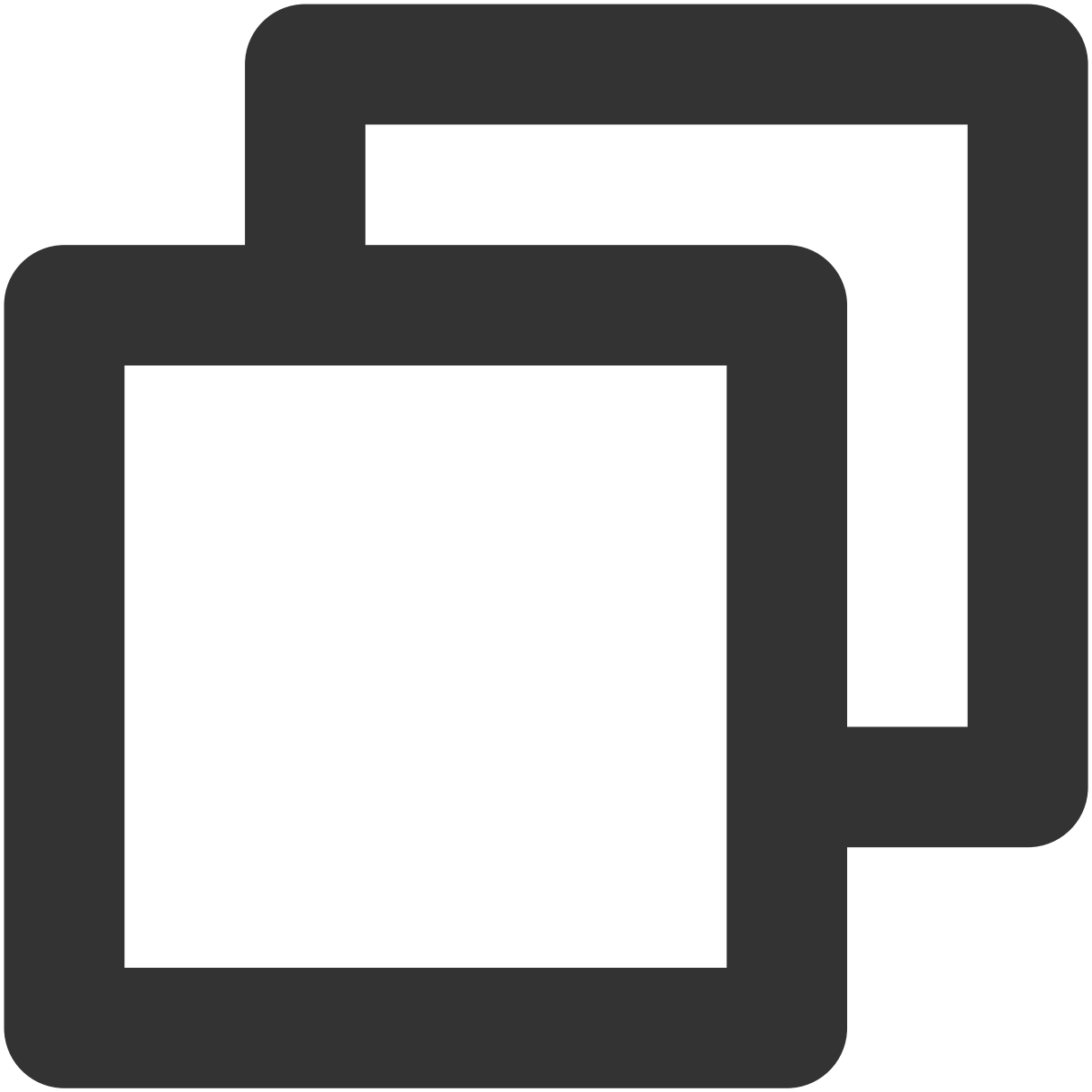
注意：

1. License 是强线上检验逻辑，应用首次启动后调用 TXLiveBase#setLicence 时，需确保网络可用。 在App首次启动时，可能还没有授权联网权限，则需要等授予联网权限后，再次调用 TXLiveBase#setLicence 。
2. 监听 TXLiveBase#setLicence 加载结果：onLicenceLoaded 接口，如果失败要根据实际情况做对应重试及引导，如果多次失败后，可以限频，并业务辅以产品弹窗等引导，让用户检查网络情况。
3. TXLiveBase#setLicence 可以多次调用，建议在进入App 主界时面调用 TXLiveBase#setLicence， 确保加载成功。
4. 对于多进程的 App， 确保每个使用播放器的进程启动时，都调用了 TXLiveBase#setLicence。例如： Android 端使用独立进程播放视频的 App， 后台播放时进程被系统 kill 掉重启时，也要调用 TXLiveBase#setLicence。

查看方法

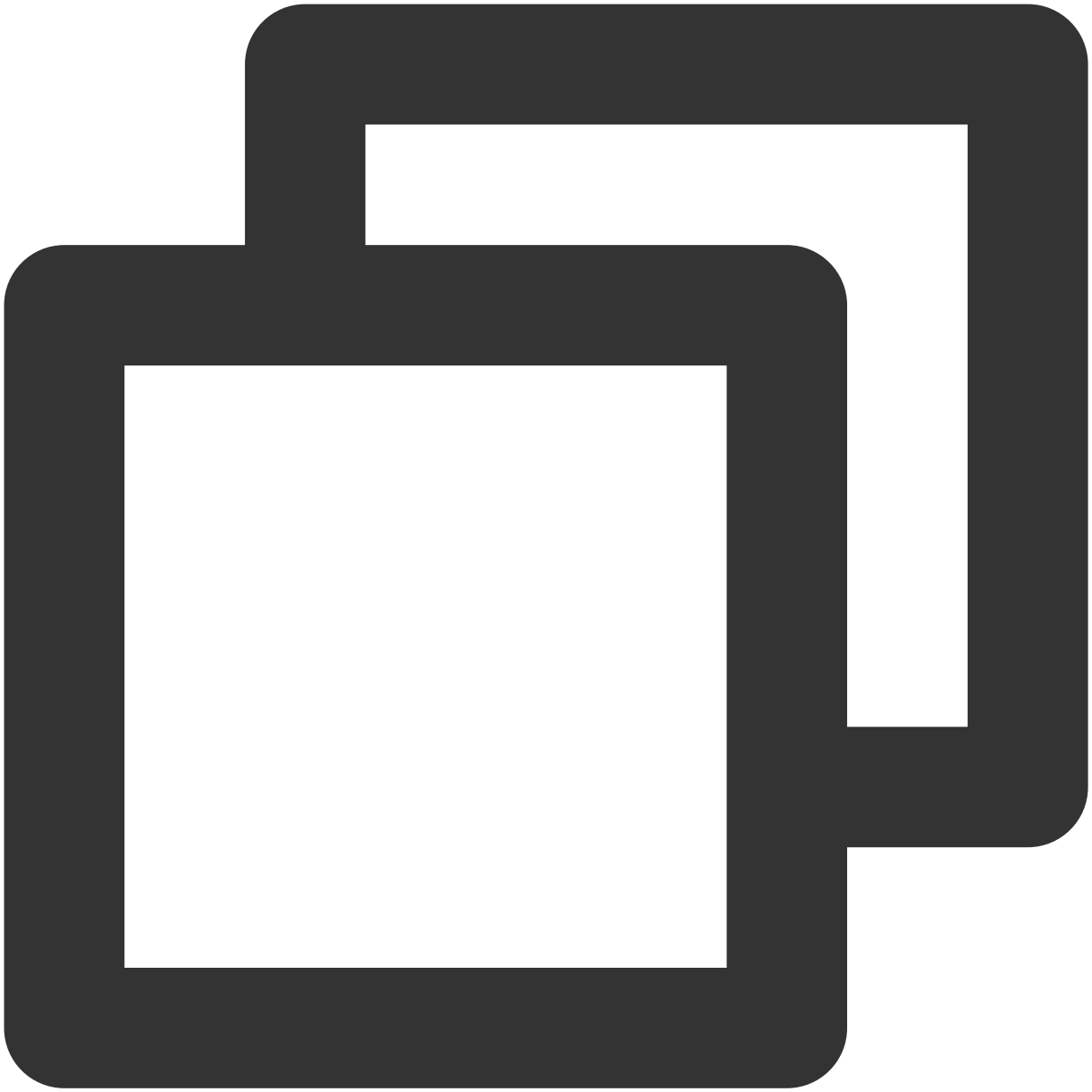
License 设置成功后（需稍等一段时间，具体时间长短依据网络情况而定），您可以通过调用如下方法查看 License 信息：

iOS：



```
NSLog(@"%@", [TXLiveBase getLicenceInfo]);
```

Android：



```
TXLiveBase.getInstance().getLicenceInfo();
```

播放器教程
阶段1：播放原始视频

最近更新时间：2023-05-15 17:15:06

学习目标

通过本阶段的教程后，您将掌握上传一个视频到云点播，并在播放器中播放的技能。

前置条件

在开始本教程之前，请您确保已满足以下前置条件。

开通云点播

您需要开通云点播，步骤如下：

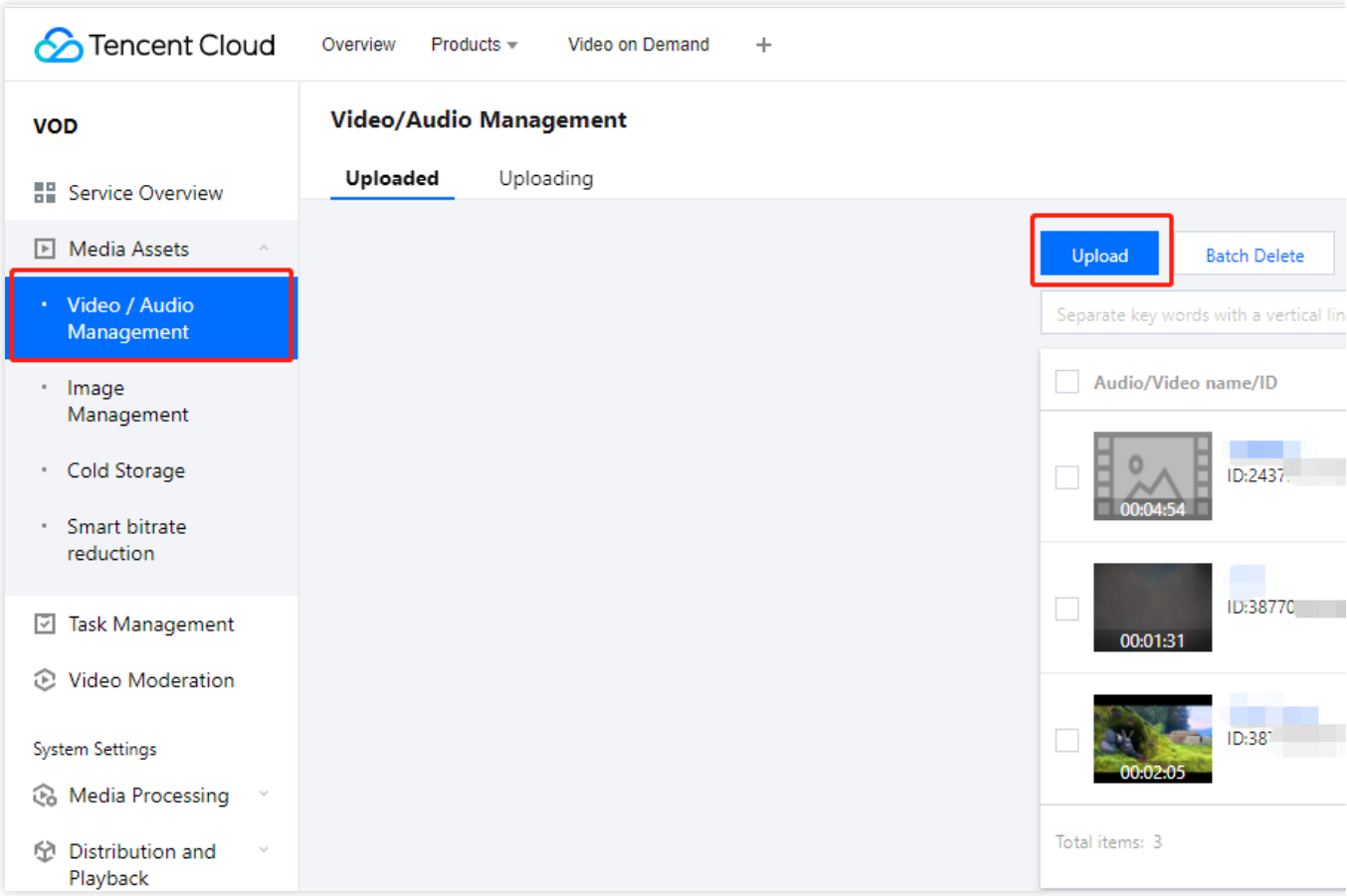
1. 注册 腾讯云账号。
2. 购买云点播服务，具体请参见 计费概述。
3. 选择**云产品** > **视频服务** > **云点播**，进入云点播控制台。

至此，您已经完成了云点播的开通步骤。

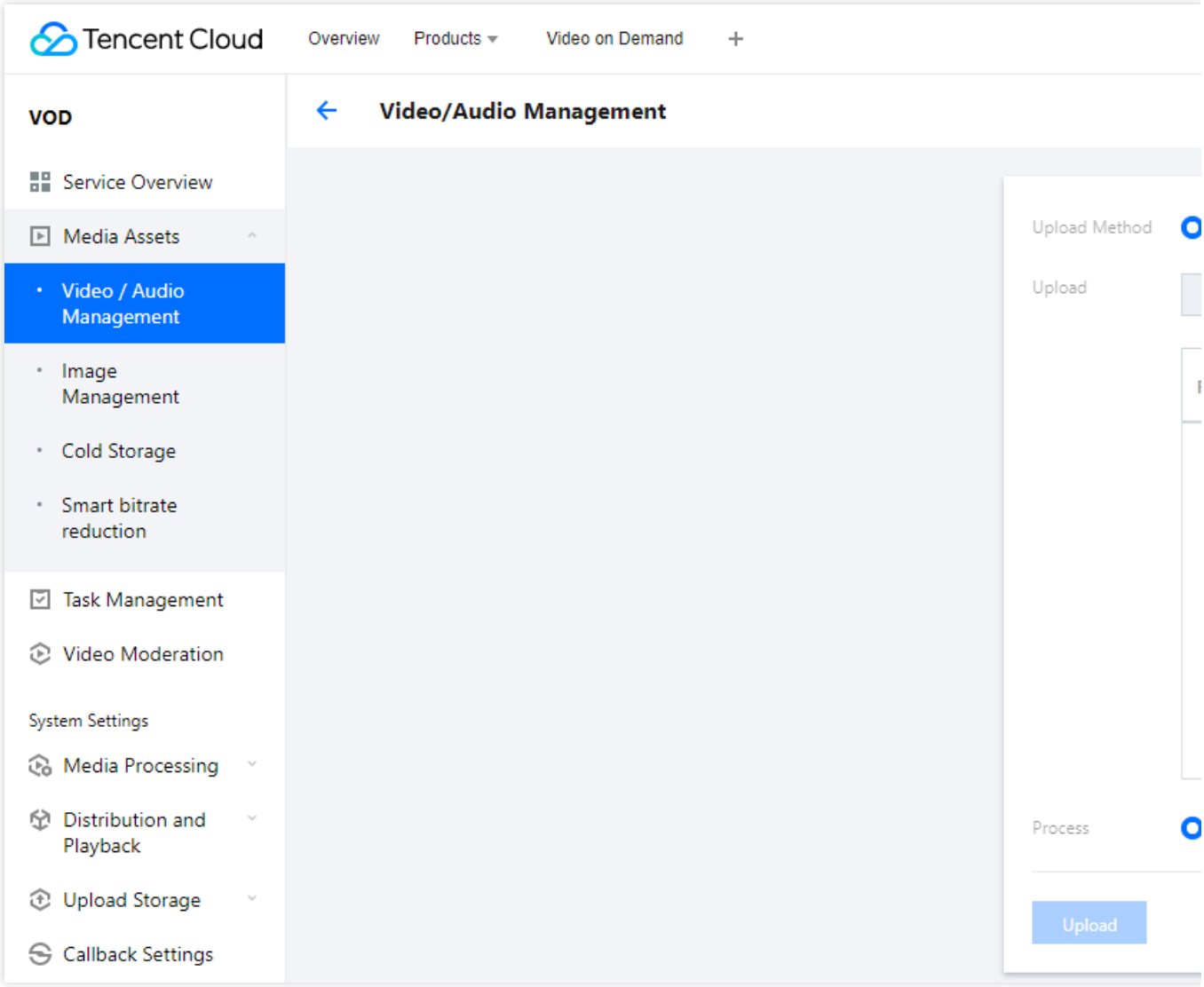
步骤1：上传视频

本步骤，我们将指导您如何上传视频。

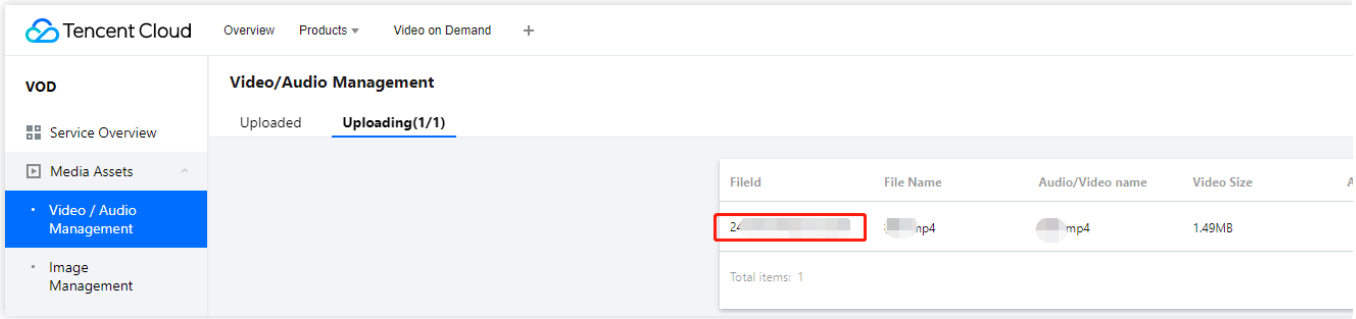
1. 登录**云点播控制台** > 应用管理，单击目标应用名称进入**媒资管理** > **音视频管理**页面，单击**上传音视频**。



2. 在上传界面，选择“**本地上传**”，并单击**选择文件** 上传本地视频，其他设置如下：
视频处理选择“**只上传，暂不进行视频处理**”。



3. 单击**开始上传**，进入“正在上传”页，当**状态**变为“**上传成功**”即表示上传完成，**文件 ID** 即为上传音视频的 FileId（这里为 387xxxxx8142975036）。



步骤2：生成播放器签名

本步骤，我们使用签名工具快速生成播放器签名，用于播放器播放视频。

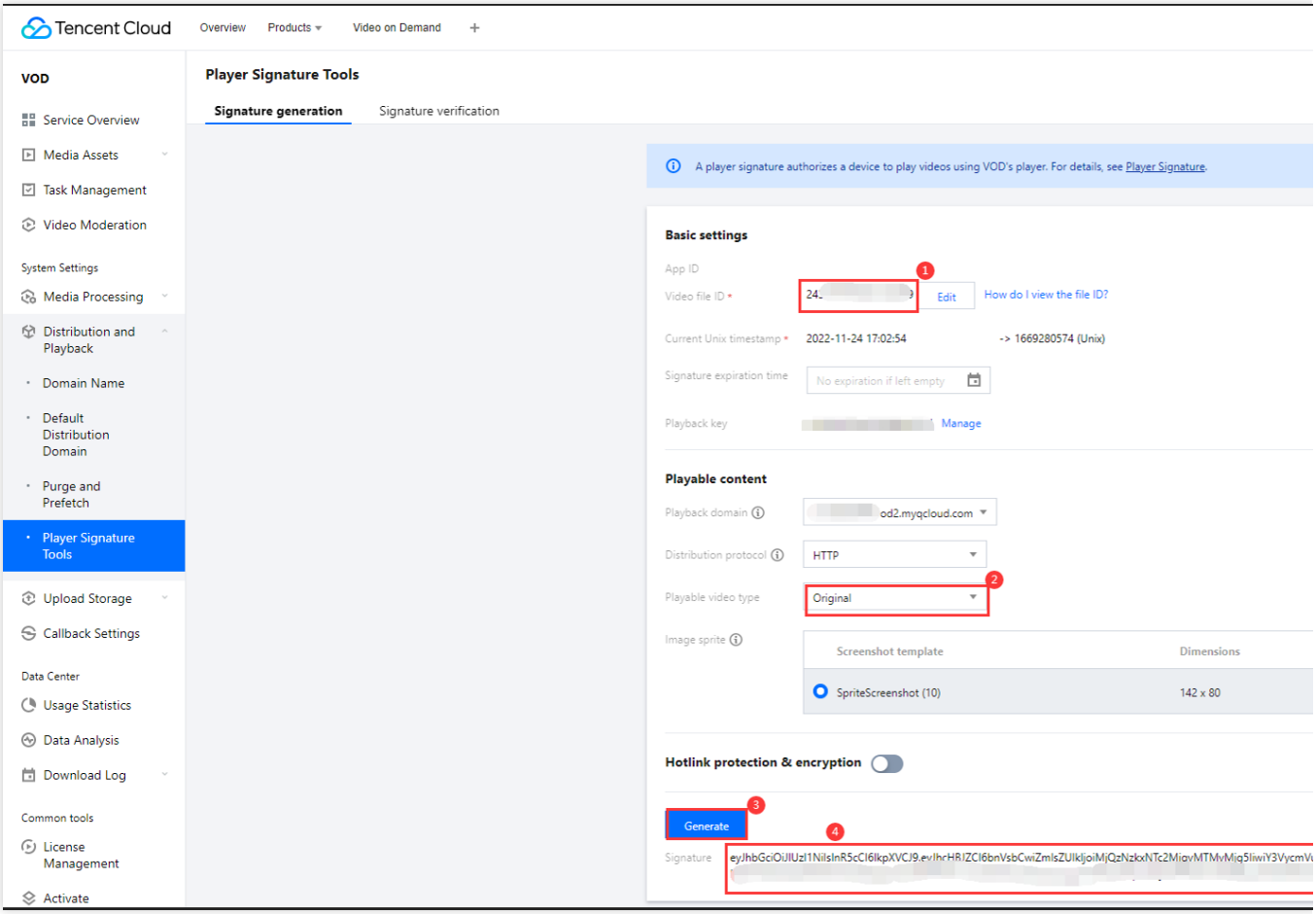
1. 在应用管理页，选择菜单栏的**分发播放设置 > 播放器签名工具**，填写如下信息：

视频 fileId 填写 步骤1 的 FileId（387xxxxx8142975036）。

签名过期时间戳播放器签名过期时间，不填表示签名不过期。

可播放的视频类型选择“**原始视频**”。

2. 单击**生成签名结果**，得到签名结果字符串。



步骤3：播放视频

经过步骤2，我们得到播放视频所需的三个参数：`appId`、`fileId`以及播放器签名（`psign`），下面将展示 Web 端播放视频。

Web 端播放示例

1. 打开 Web端播放器体验，配置如下：

播放器功能选择“**视频播放**”。

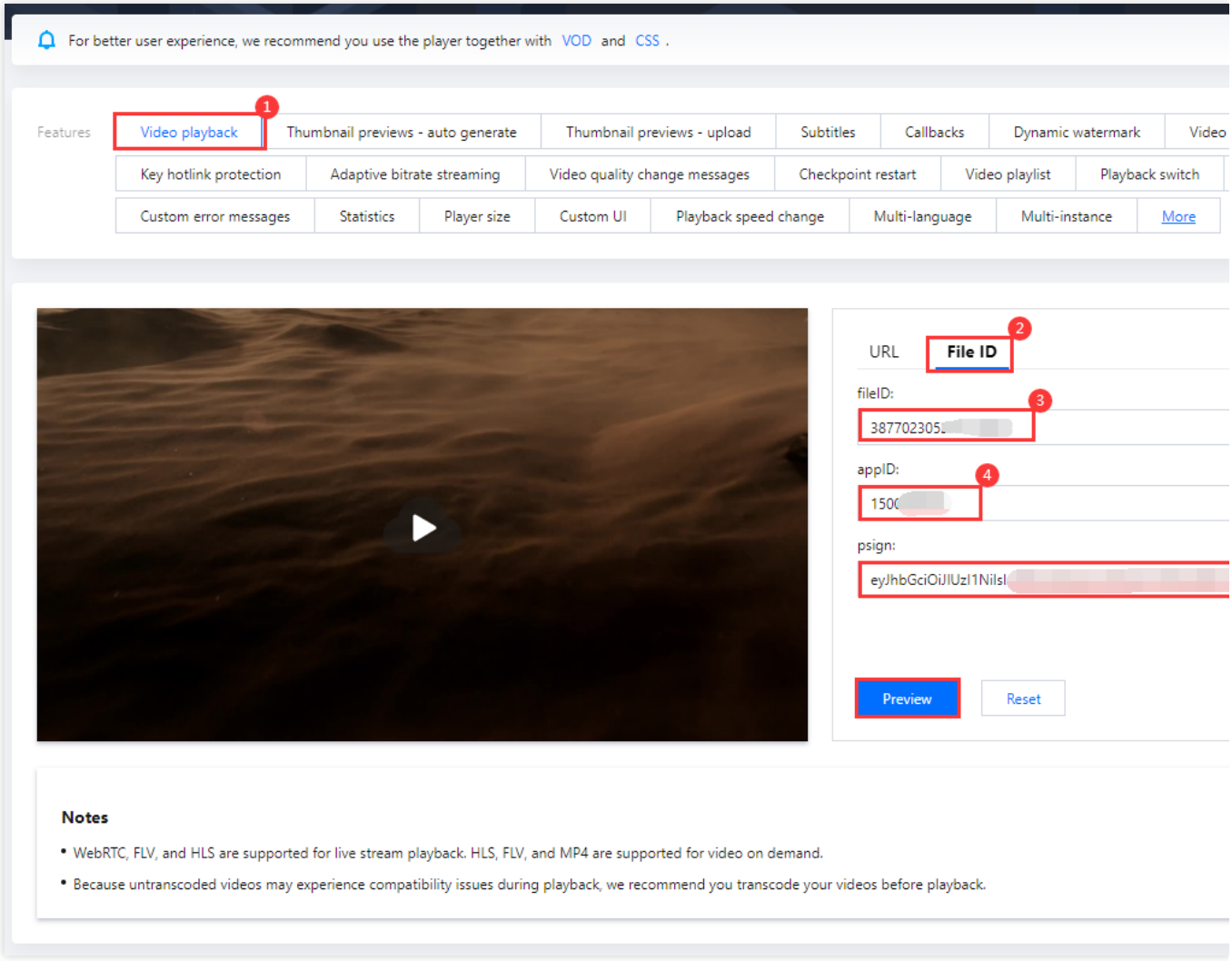
单击 **FileID 播放**标签页。

fileID 填写上一步的 FileId（387xxxxx8142975036）。

appID 填写文件所属的 appId（即上一步生成播放器签名页面的 appID）。

psign 填写上一步生成的签名结果字符串。

2. 单击**预览**即可播放视频。



多端播放器 Demo

获取播放器签名后，您可以分别使用 Web、Android 和 iOS 三端的播放器 Demo 进行验证，具体请参考 Demo 的源码。

总结

学习本教程后，您已经掌握如何上传一个视频到云点播，并在播放器中播放。

如果您希望：

播放转码视频，请参考 阶段2：播放转码视频

播放自适应码流视频，请参考 阶段3：播放自适应码流视频

播放加密视频，请参考 阶段4：播放加密视频

播放长视频方案，请参考 阶段5：播放长视频方案

阶段2：播放转码视频

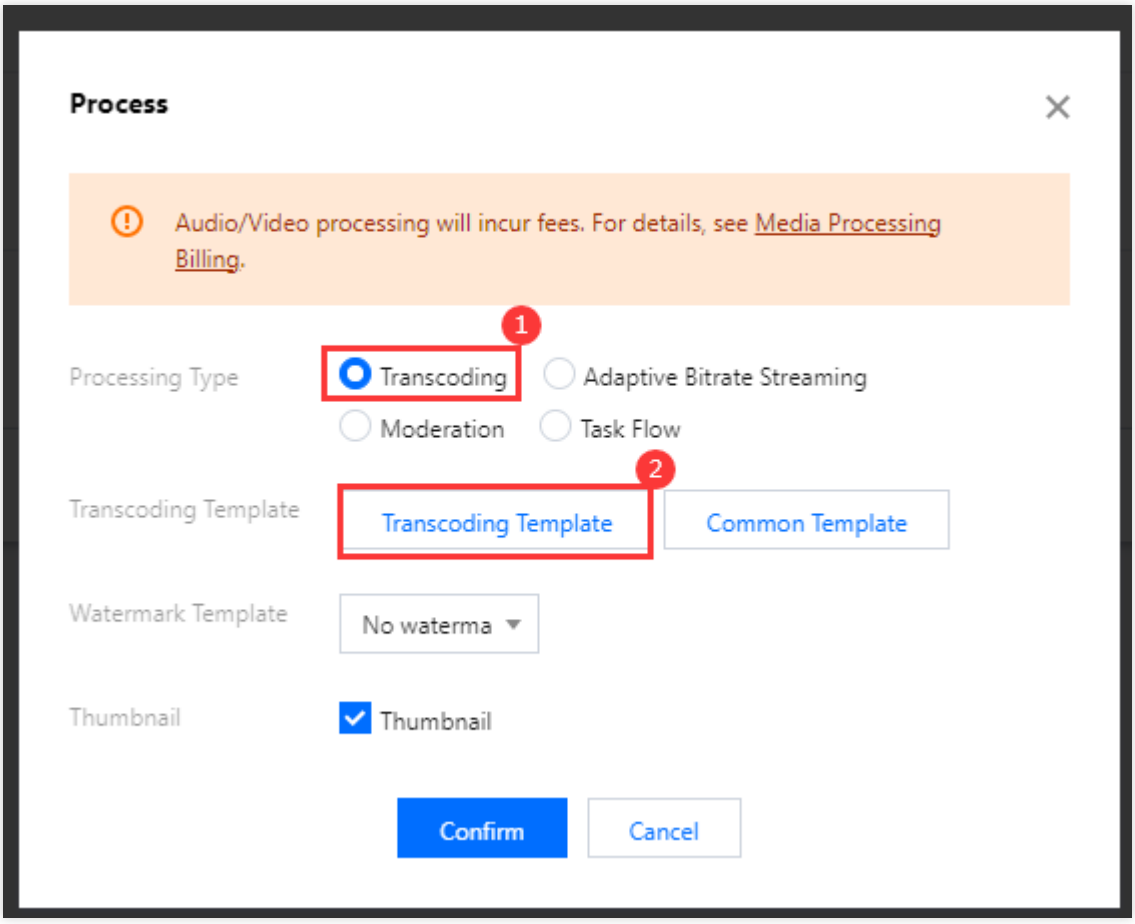
最近更新时间：2023-05-15 17:15:06

学习目标

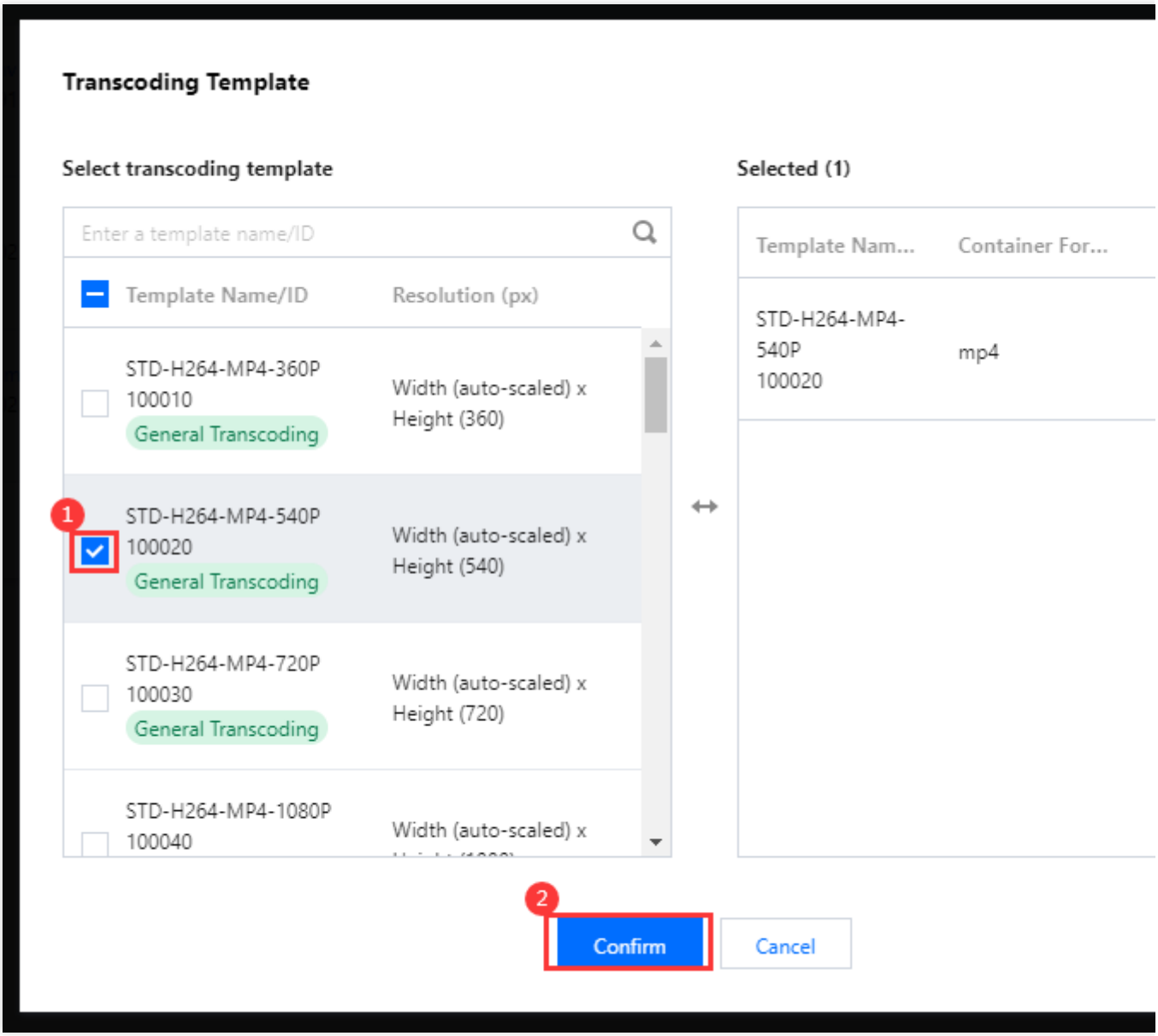
学习本阶段教程，您将了解并掌握如何对视频转码，并使用播放器播放转码视频。
阅读之前，请先确保已经学习播放器指引的 阶段1：播放原始视频 篇部分，本教程使用了 阶段1 篇开通的账号以及上传的视频。

步骤1：视频转码

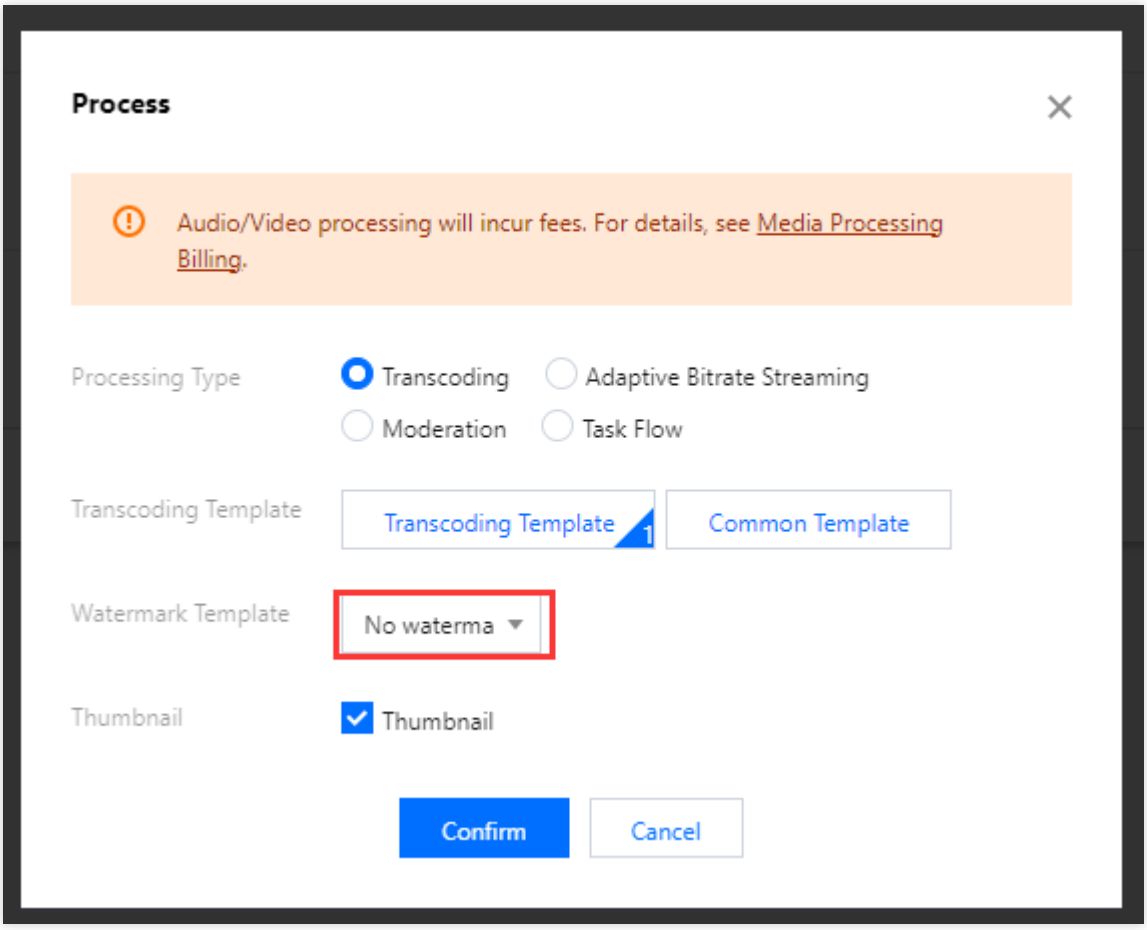
1. 登录**云点播控制台 > 应用管理**，单击目标应用名称后默认进入**媒资管理 > 音视频管理**页面，勾选要处理的视频（FileId 为 387xxxxx8142975036），单击 **转码。**
2. 在**媒体处理**界面：
2.1 **处理类型**选择“**转码**”。
2.2 **转码模板**单击**选择模板：**



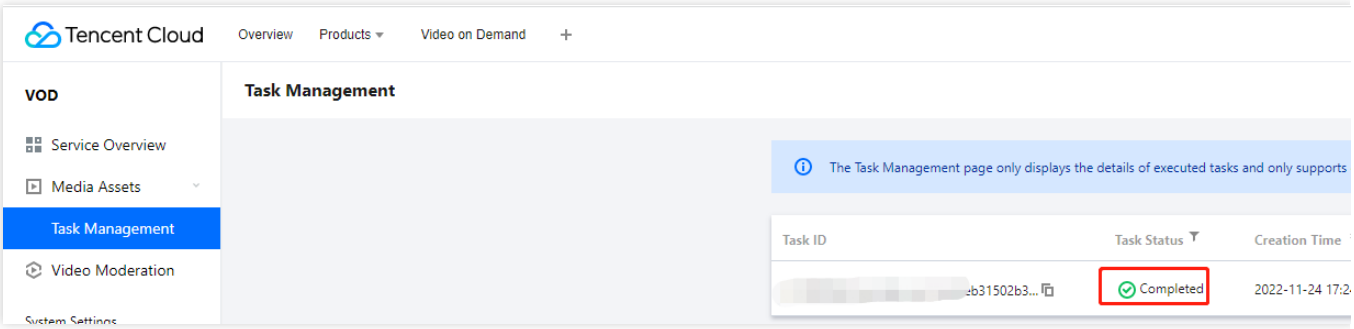
2.3 在弹出的**选择转码模板**页面选择模板（模板名称为 `STD-H264-MP4-540P`，ID 为 `100020`）：



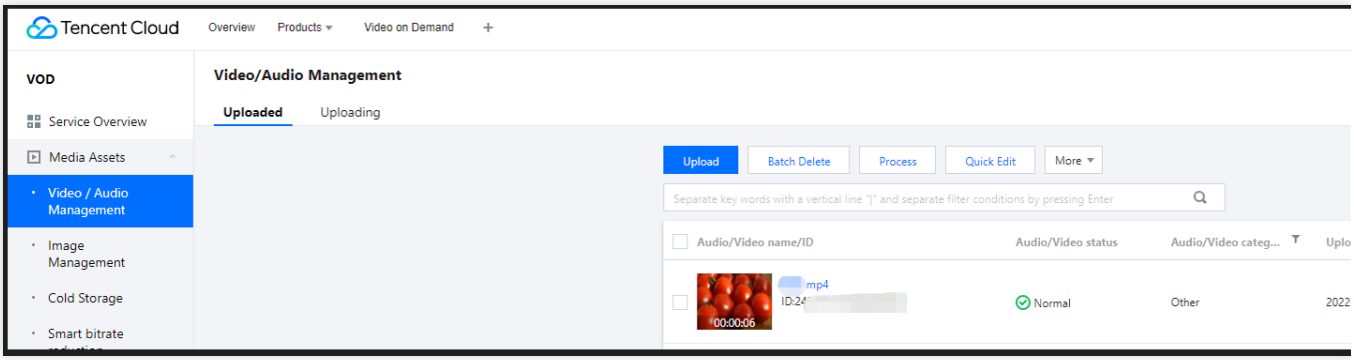
2.4 单击**确定**发起转码任务。



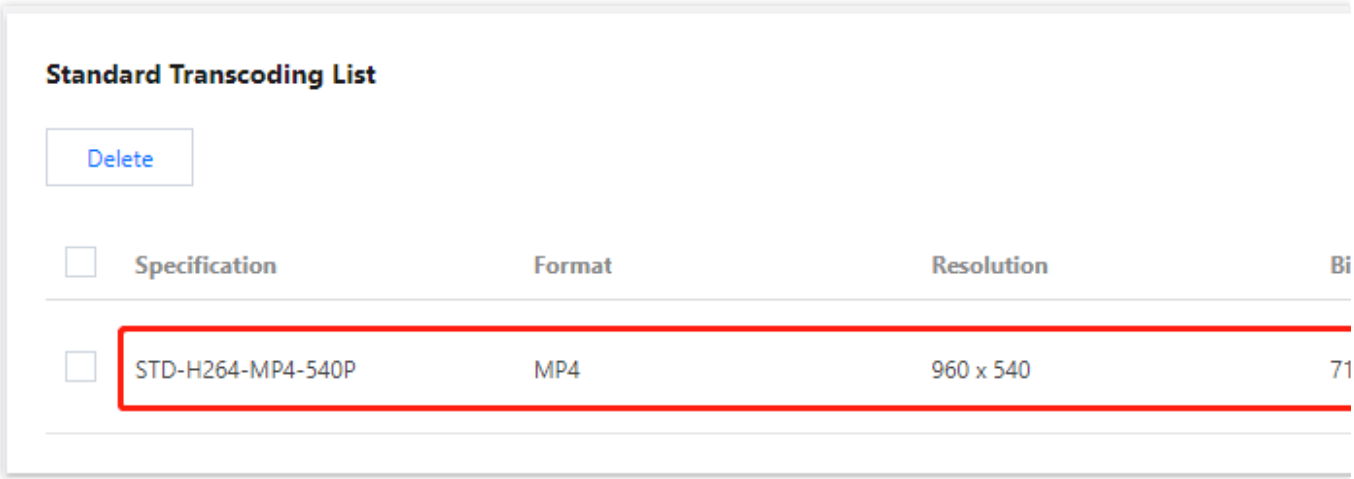
3. 进入左侧导航栏**任务中心**，列表中的“任务状态”从“处理中”变为“已完成”，表示视频已处理完毕。



4. 进入**媒资管理** > **音视频管理**，单击发起转码视频条目右侧的**管理**，进入管理页面。



转码信息页面可以看到转码成功的转码模板列表。



步骤2：生成播放器签名

本步骤，我们使用签名工具快速生成播放器签名，用于播放器播放视频。

1. 选择**分发播放设置** > **播放器签名工具**，填写如下信息：

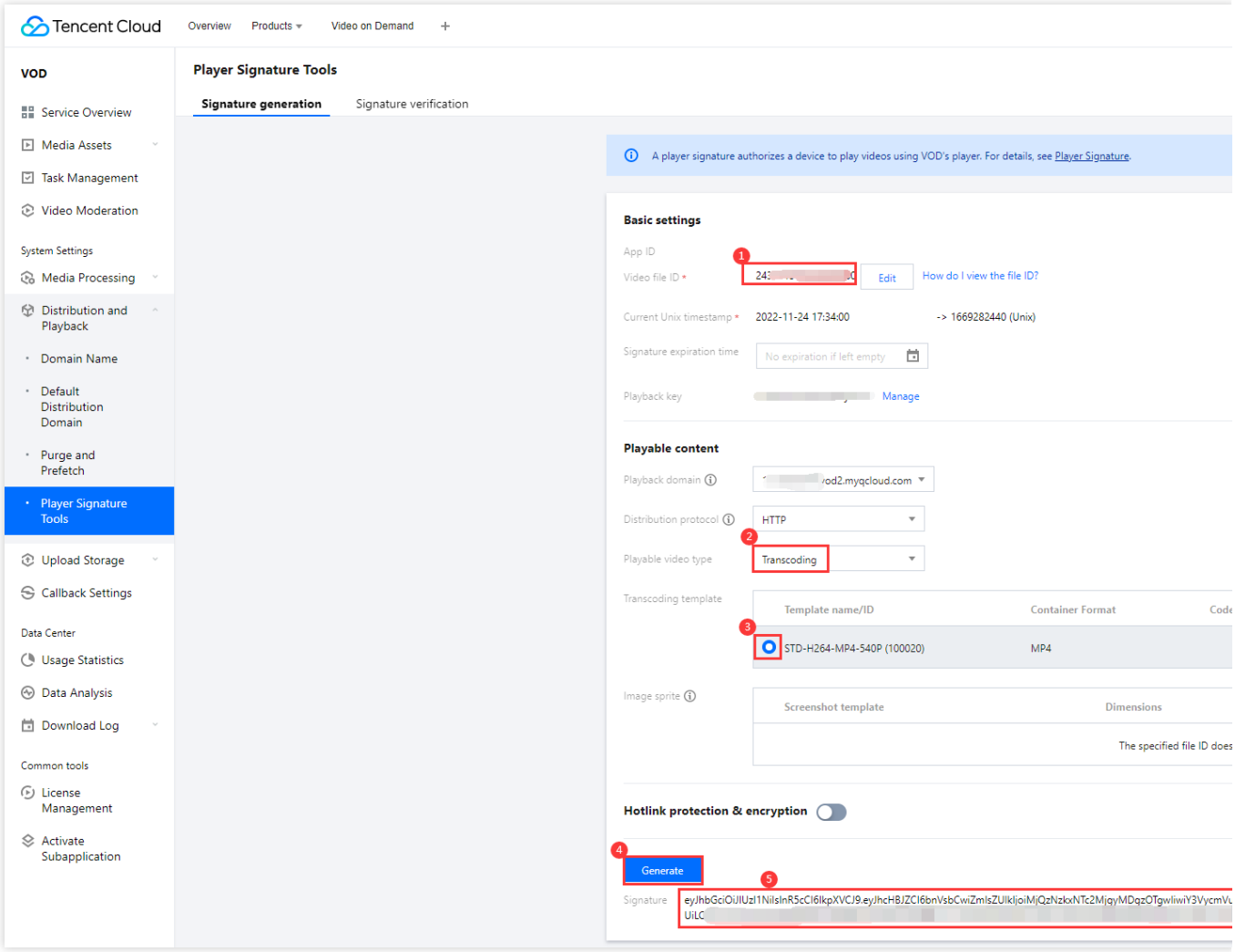
视频 fileId 填写 步骤1 的 FileId（387xxxxx8142975036）。

签名过期时间戳播放器签名过期时间，不填表示签名不过期。

可播放的视频类型选择“**转码**”。

可播放的转码模板选择 `STD-H264-MP4-540P (100020)` 。

2. 单击**生成签名结果**，得到签名结果字符串。

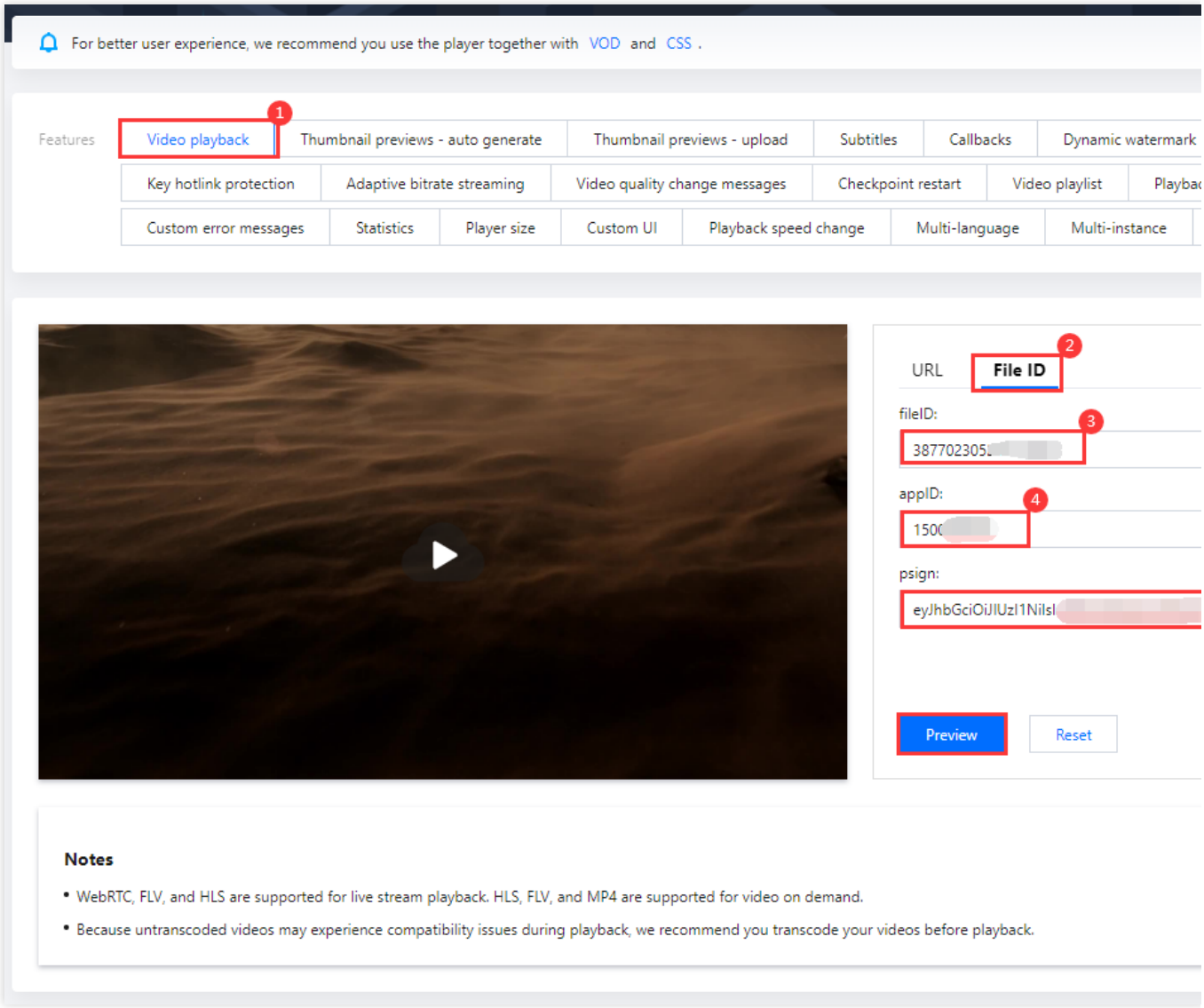


步骤3：播放视频

经过步骤2，我们得到播放视频所需的三个参数：`appId`、`fileId`以及播放器签名（`psign`），下面将展示 Web 端播放视频。

Web 端播放示例

1. 打开 Web端播放器体验，配置如下：
播放器功能选择“**视频播放**”。
单击 **FileID 播放** 标签页。
fileID 填写上一步的 FileId（387xxxxx8142975036）。
appID 填写文件所属的 appId（即上一步生成播放器签名页面的 appID）。
psign 填写上一步生成的签名结果字符串。
2. 单击**预览**即可播放视频。



多端播放器 Demo

获取播放器签名后，您可以分别使用 Web、Android 和 iOS 三端的播放器 Demo 进行验证，具体请参考 Demo 的源码。

总结

学习本教程后，您已经掌握如何对视频转码，并在播放器中播放。

阶段3：播放自适应码流视频

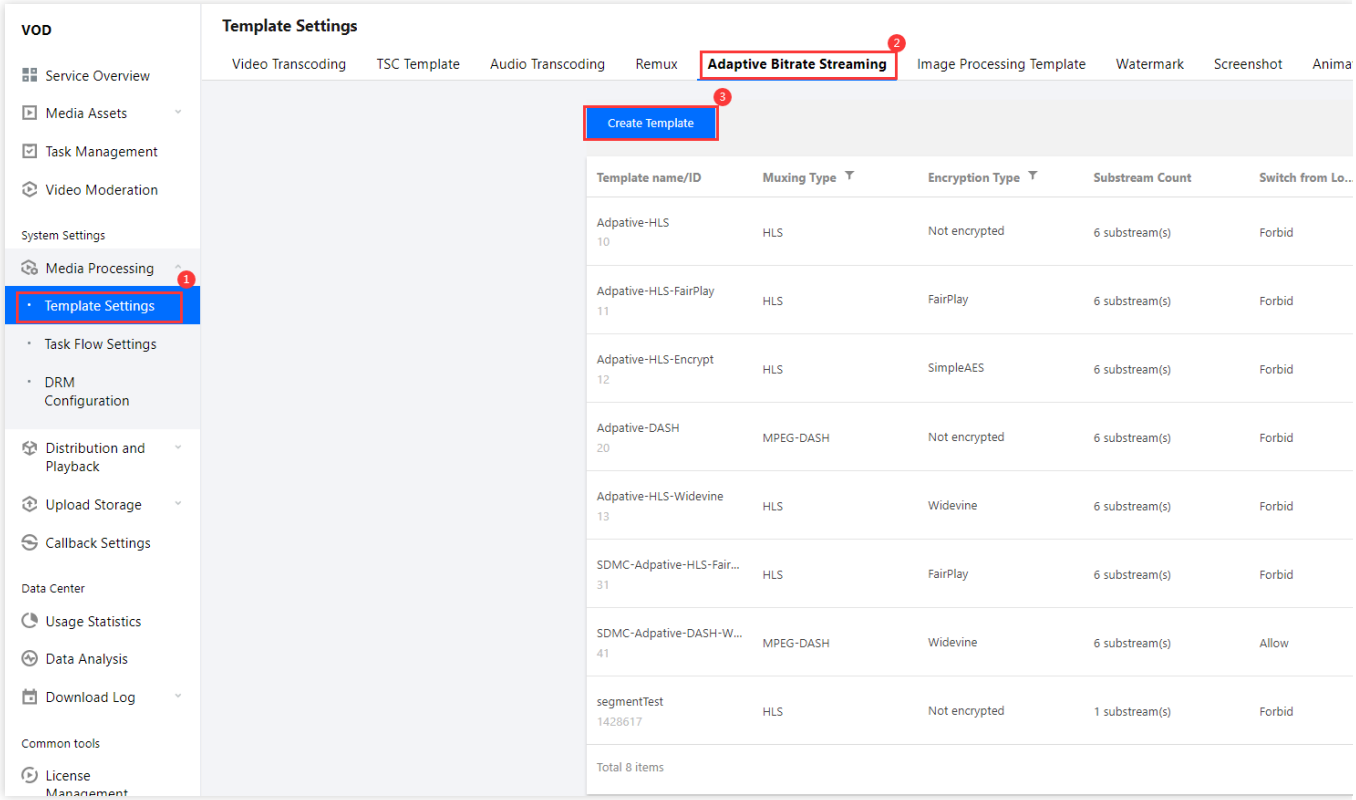
最近更新时间：2023-05-15 17:15:06

学习目标

学习本阶段教程，您将了解如何播放自适应码流视频，包括：
播放子流规格中最小分辨率为480p，最大分辨率1080p。
使用视频中间部分的截图作为视频封面。
进度条上的缩略图预览，调整为20%的间隔。
阅读之前，请先确保已经学习播放器指引的 阶段1：播放原始视频 篇部分，本教程使用了 阶段1 篇开通的账号以及上传的视频。

步骤1：创建自适应码流模板

1. 登录**云点播控制台 >** 应用管理，单击目标应用名称进入应用管理页，选择**媒体处理设置 > 模板设置**，单击“转自适应码流模板”页签下的**创建转自适应码流模板**。



2. 进入“模板设置”页面后，单击**添加子流**，新建子流1、子流2和子流3，填写参数如下：

基本信息模块：

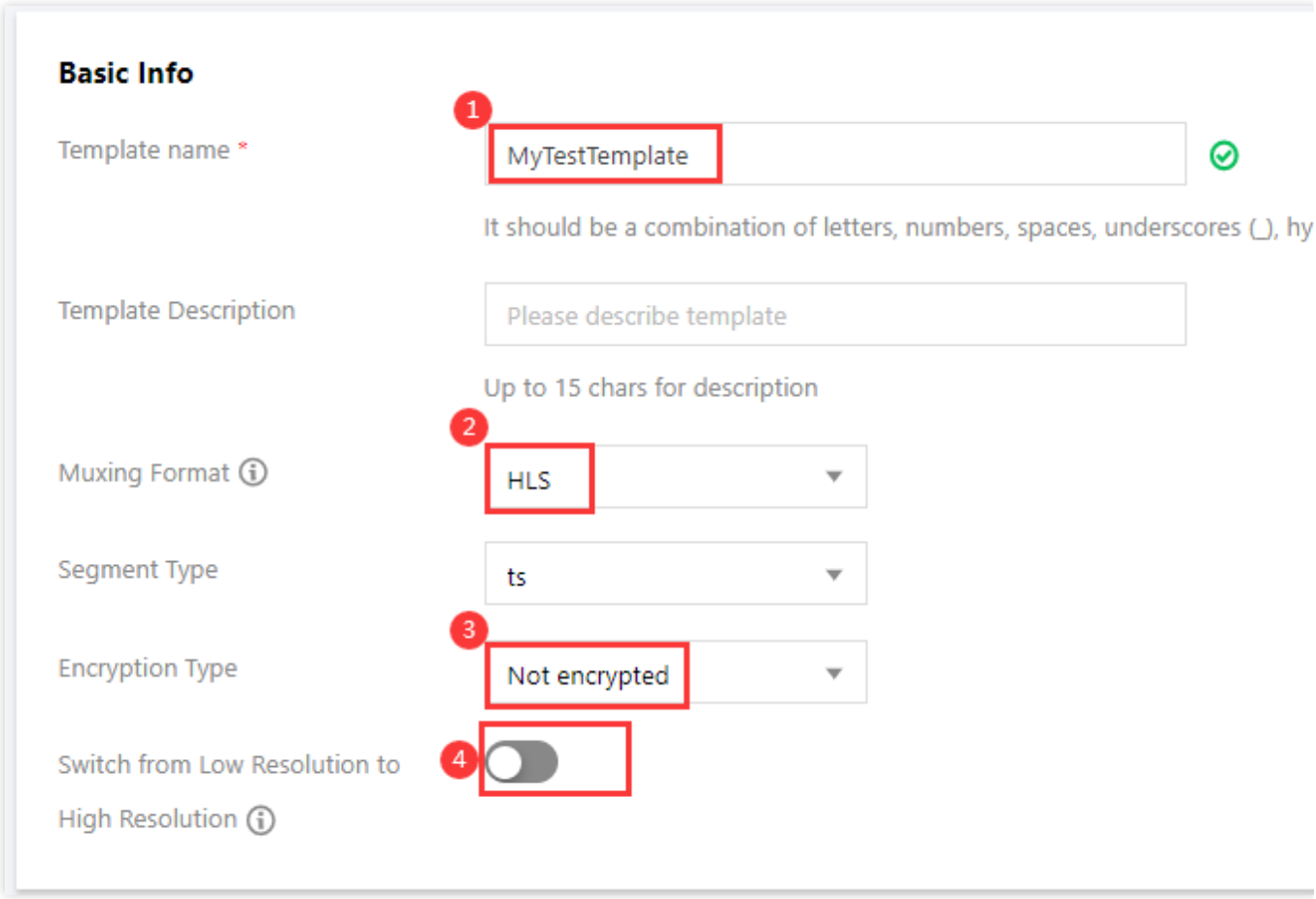
模板名称：填写 MyTestTemplate。

打包格式：选择 “HLS”。

加密类型：选择“不加密”。

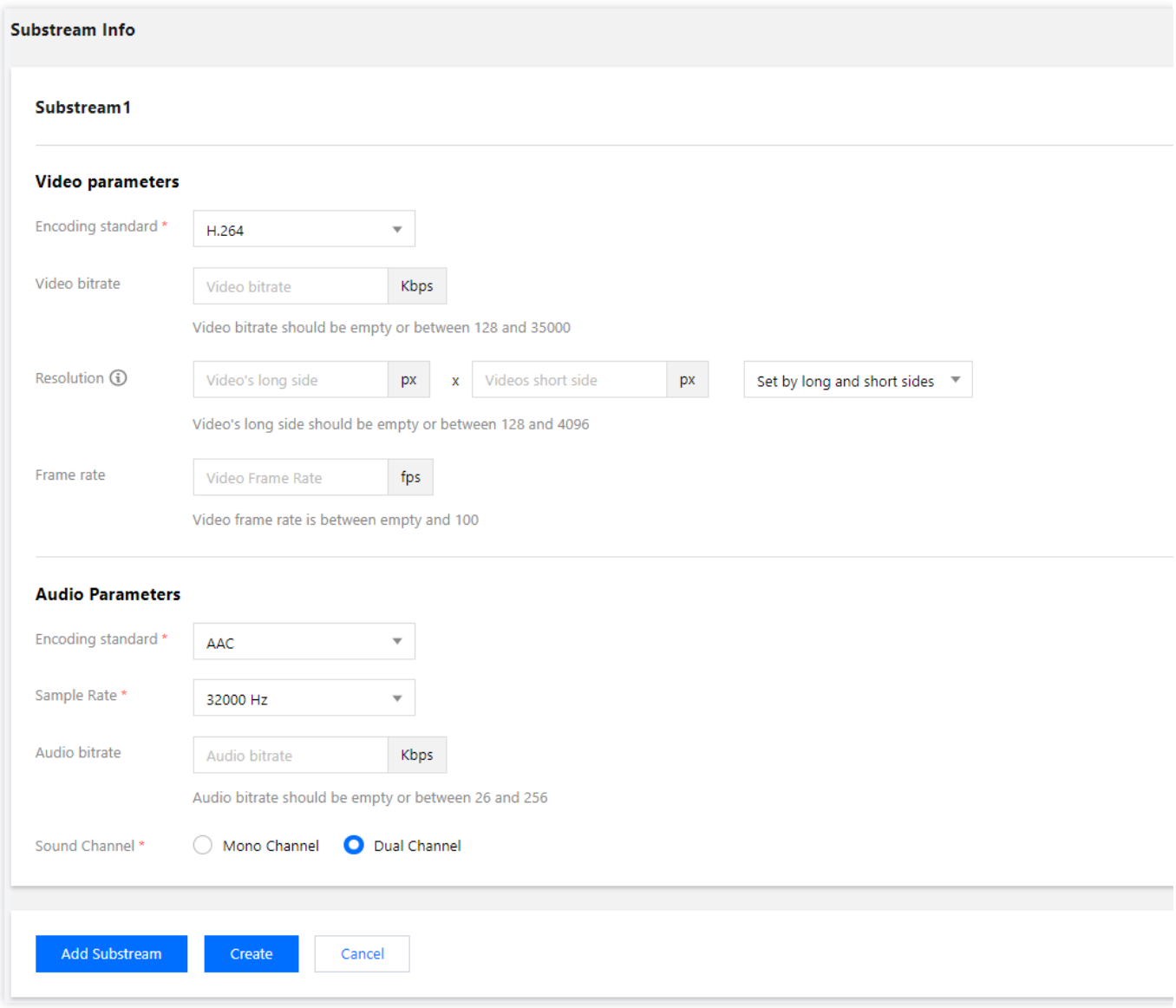
低分辨率转高分辨率：不开启。

转码方式：选择“普通转码”。

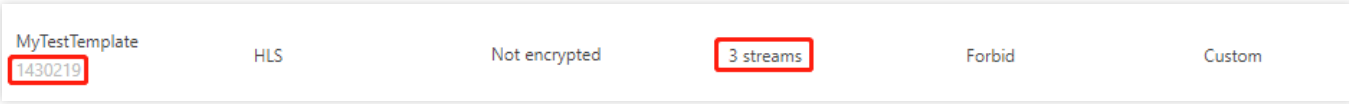


子流信息模块：

子流编号	视频码率	分辨率	帧率	音频码率	声道
子流1	512kbps	视频长边0px，视频短边480px	24fps	48kbps	双声道
子流2	512kbps	视频长边0px，视频短边720px	24fps	48kbps	双声道
子流3	1024kbps	视频长边0px，视频短边1080px	24fps	48kbps	双声道

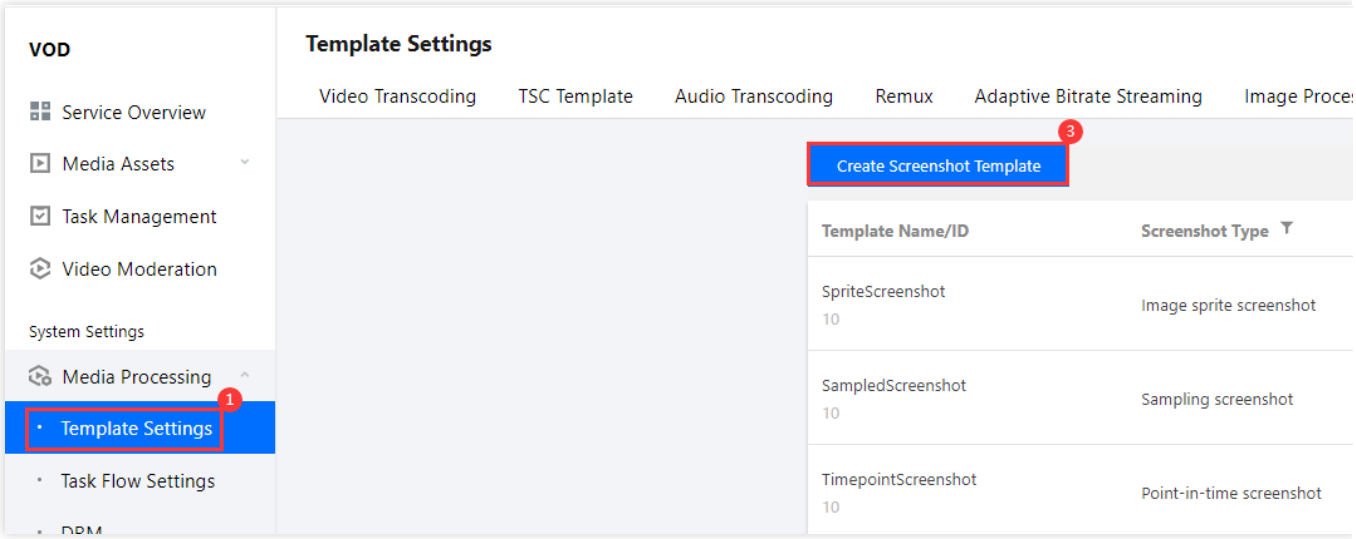


3. 单击**创建**，则生成了一个包含3个子流的自适应码流模板，模板 ID 为 `1430219` 。



步骤2：创建雪碧图模板

1. 在**媒体处理设置** > **模板设置**页，单击“截图模板”页签下的**创建截图模板**。



2. 进入“模板设置”页面后，填写模板参数：

模板名称：填写 MyTestTemplate。

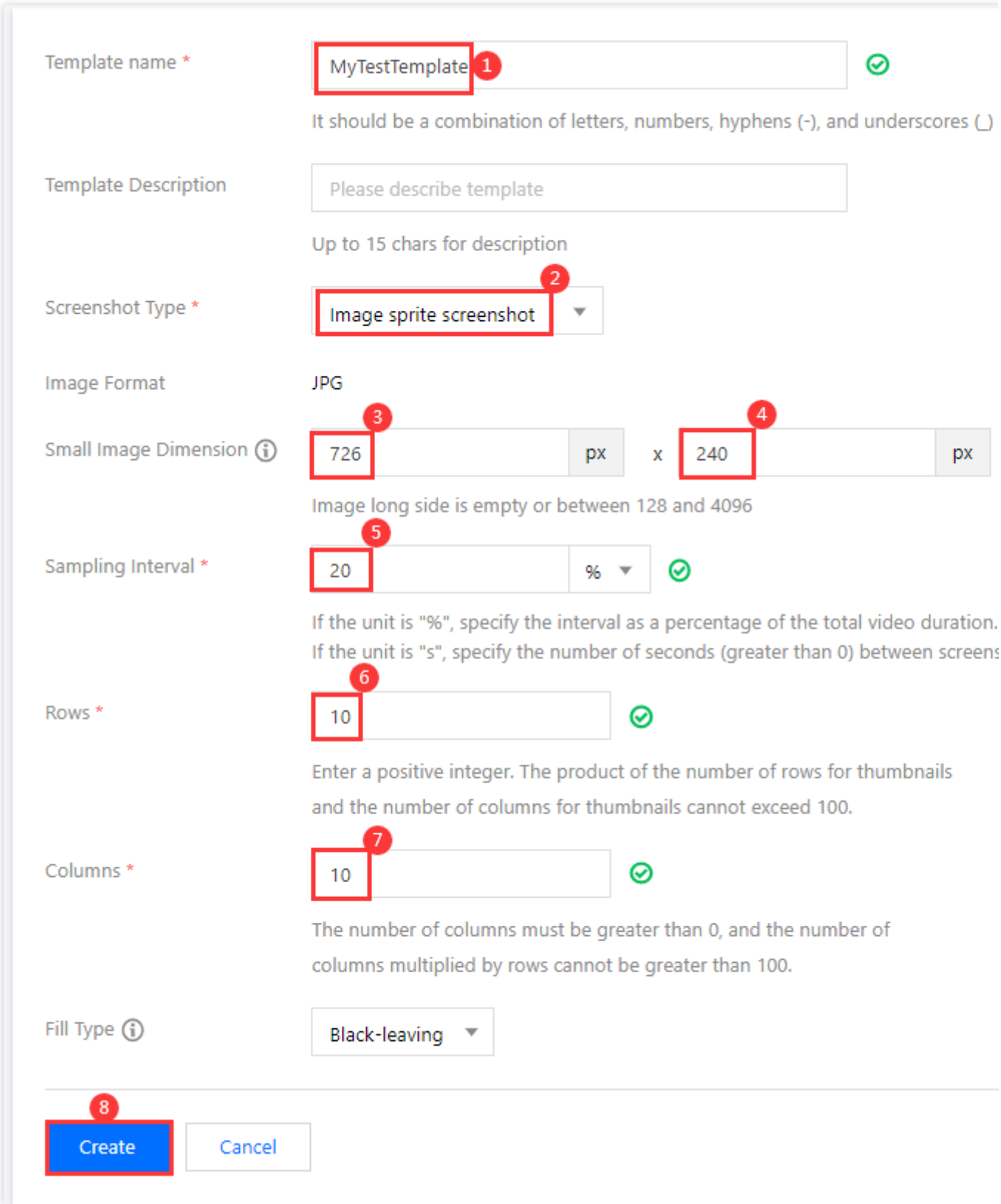
模板类型：选择“雪碧图截图”。

小图尺寸：726px × 240px。

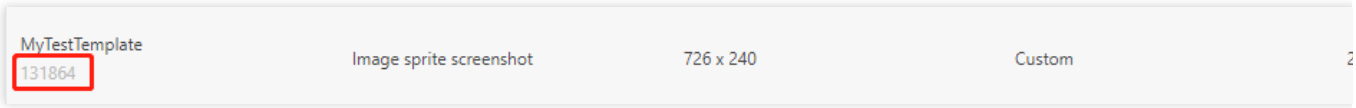
采样间隔：20%。

小图行数：10。

小图列数：10。



3. 单击**创建**，则生成了一个模板 ID 为 131864 的雪碧图模板。



步骤3：创建任务流并发起处理

创建新的转自适应码流模板（ID 为 1430219）和雪碧图模板（ID 为 131864）后，还需要创建一个新的任务流。

1. 在**媒体处理设置 > 模板设置**页，单击**创建任务流**：

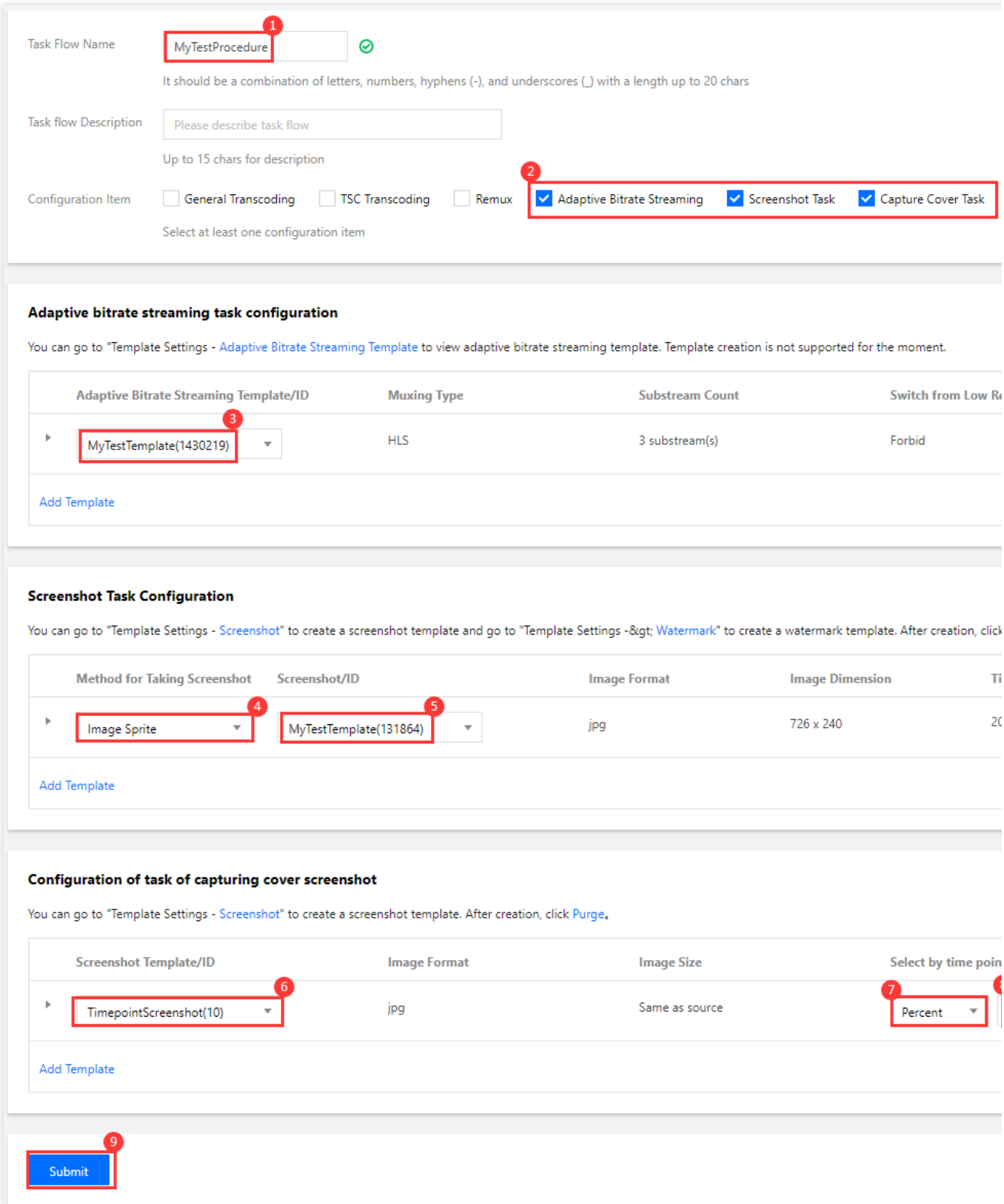
任务流名称：填写 MyTestProcedure。

任务类型配置：勾选“转自适应码流”、“截图”和“截取封面”：

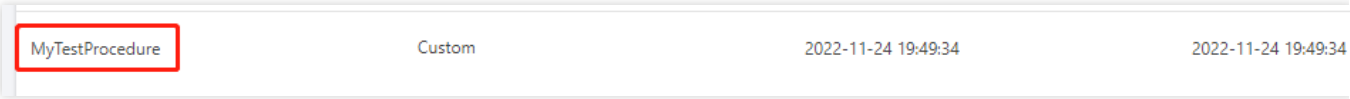
在**自适应码流任务配置**选项卡，单击**添加自适应码流模板**，在“自适应码流模板/ID”栏选择**步骤1**创建的自定义转自适应码流模板 MyTestTemplate(1430219)。

在**截图任务配置**选项卡，单击**添加截图模板**，“截图方式”栏选择“雪碧图”，“截图模板”栏选择**步骤2**创建的自定义雪碧图模板 MyTestTemplate(131864)。

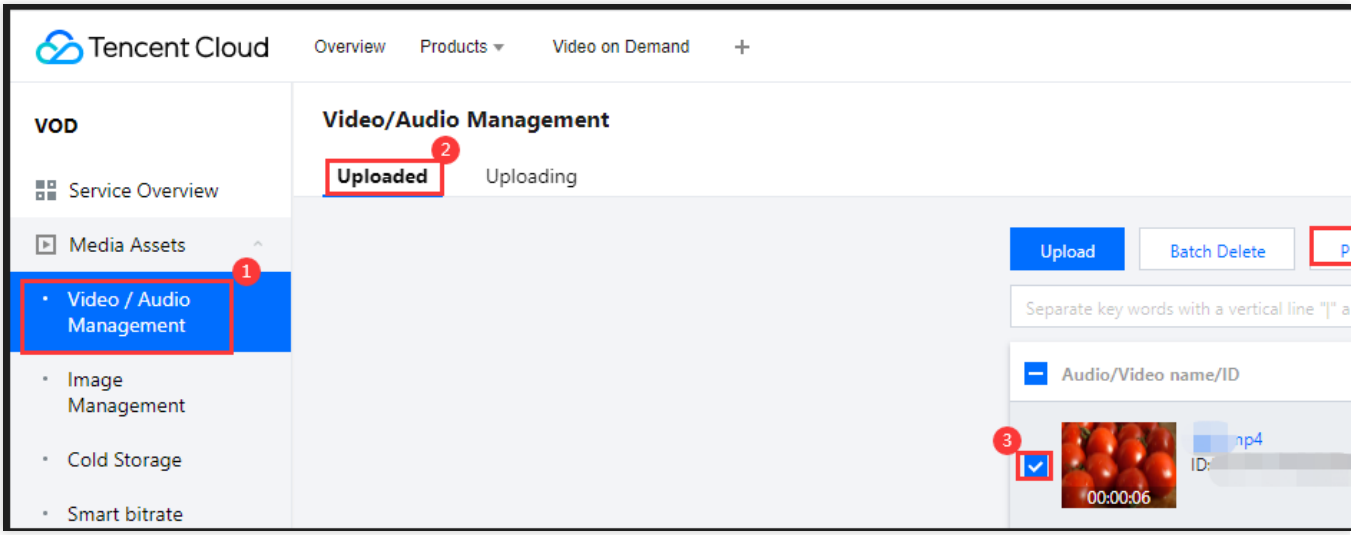
在**截取封面图任务配置**选项卡，单击**添加截图模板**，“截图模板”栏选择“TimepointScreenshot”，“时间点选取”栏选择“百分比”，填写50%。



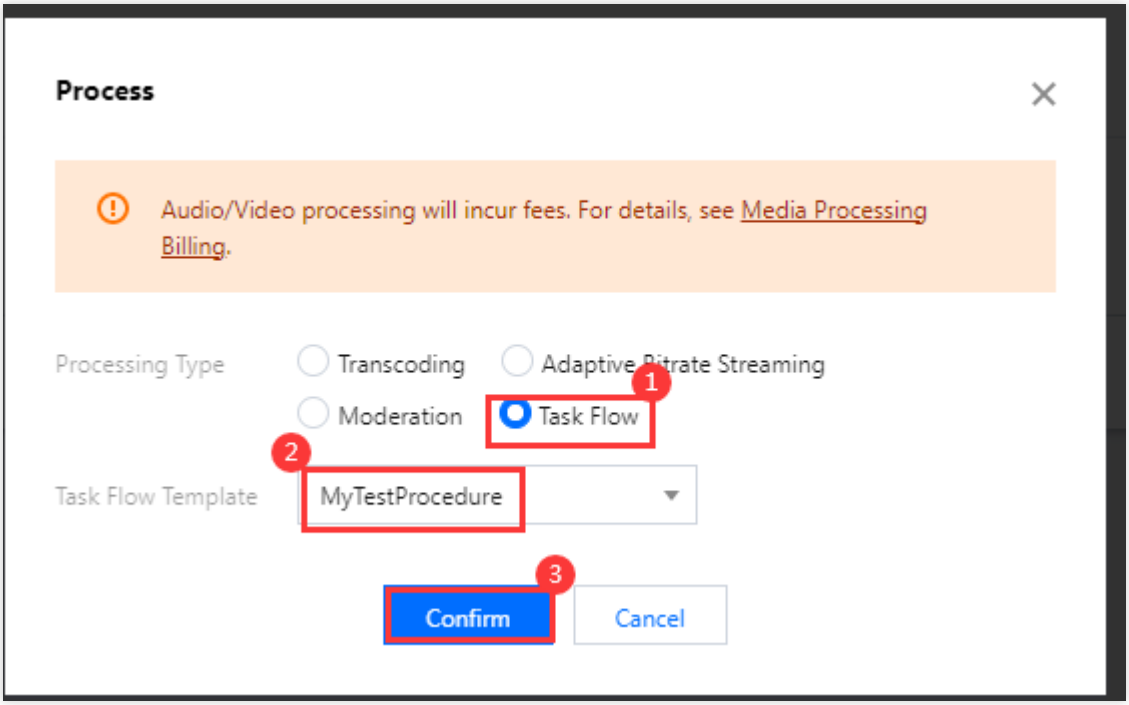
2. 单击**提交**，生成了一个名为 MyTestProcedure 的任务流。



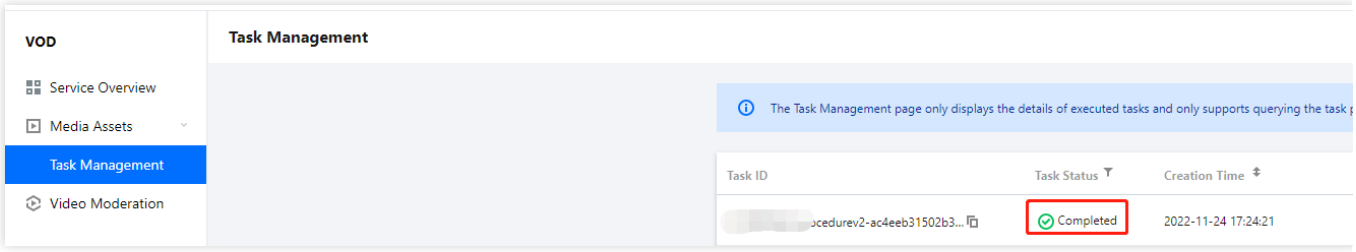
3. 在控制台选择**媒资管理** > **音视频管理**，勾选要处理的视频（FileId 为 243xxx814xxxxx416），单击任务流。



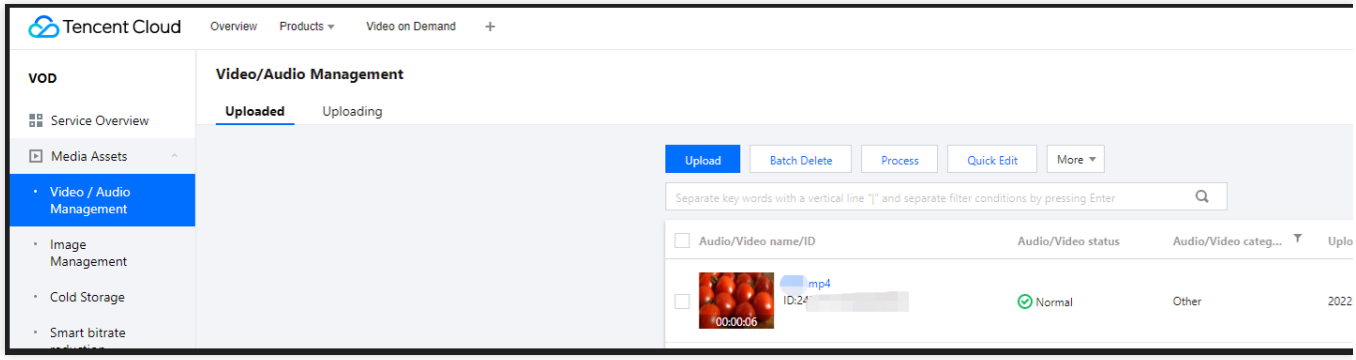
4. 在视频处理弹框：
处理类型选择“任务流”。
任务流模板选择“MyTestProcedure”。



5. 单击**确定**，等待列表中的“任务状态”从“处理中”变为“已完成”，表示视频已处理完毕。

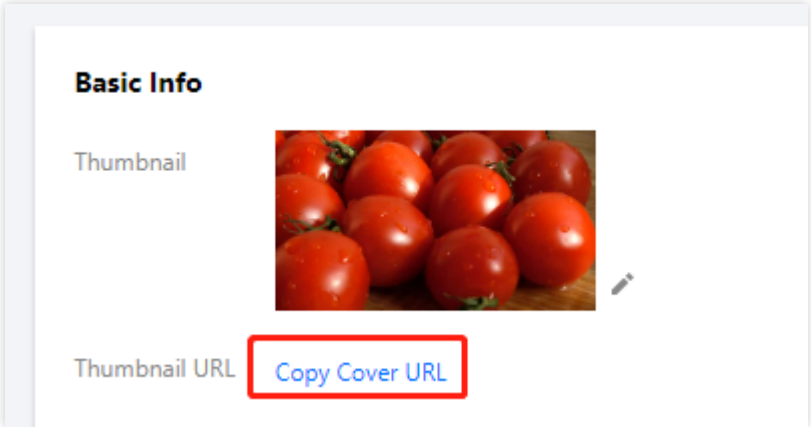


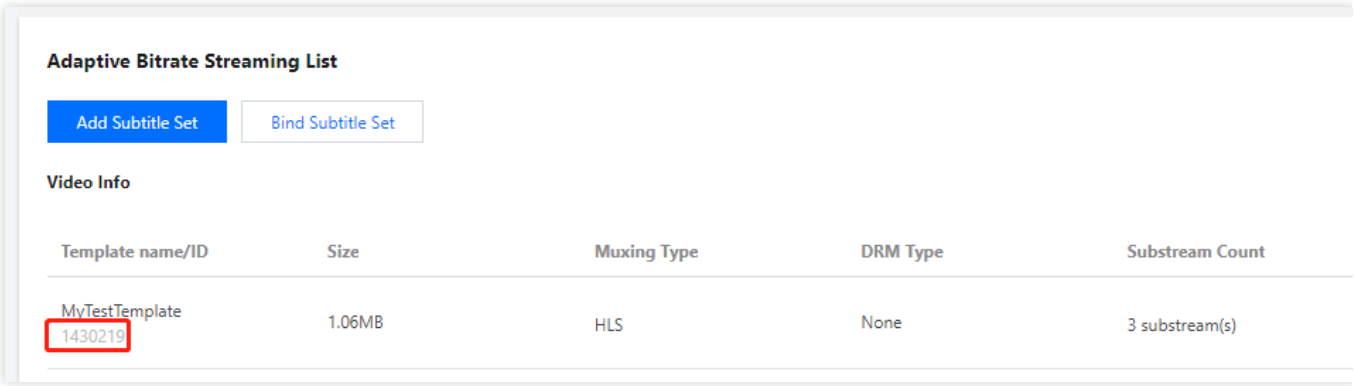
6. 进入**媒资管理 > 音视频管理**，单击发起转自适应码流的视频条目右侧**管理**，进入管理页面。



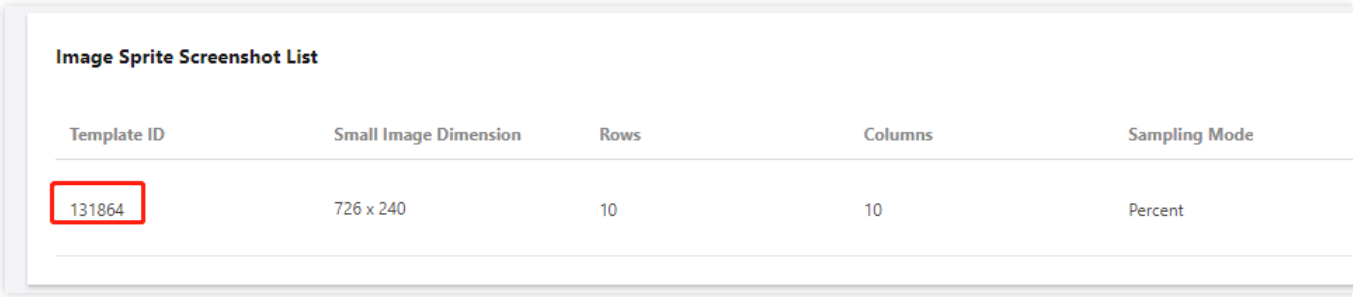
6.1 在**基本信息**模块可以查看：

可以看到生成的封面，以及自适应码流输出（模板 ID 为 1430219）。





6.2 选择**截图信息**页签：
可以看到生成的雪碧图（模板 ID 为 131864）。



步骤4：生成播放器签名

本步骤，我们使用签名工具快速生成播放器签名，用于播放器播放视频。
选择 **分发播放设置** > **播放器签名工具**，填写如下信息：
视频 fileId 填写**步骤3**的 FileId（243xxx814xxxxx416）。
签名过期时间戳播放器签名过期时间，不填表示签名不过期。
可播放的视频类型选择 **转自适应码流(不加密)。**
可播放的自适应码流模板选择 `MyTestTemplate (1430219)` 。
用于缩略图预览的雪碧图选择 `MyTestTemplate (131864)` 。
单击**生成签名结果**，得到签名结果字符串。

步骤5：播放视频

经过步骤4，我们得到播放视频所需的三个参数： `appId` 、 `fileId` 以及播放器签名（ `psign` ），下面将展示 Web 端播放视频。

Web 端播放示例

打开 Web端播放器体验，配置如下：

播放器功能 选择 **视频播放**

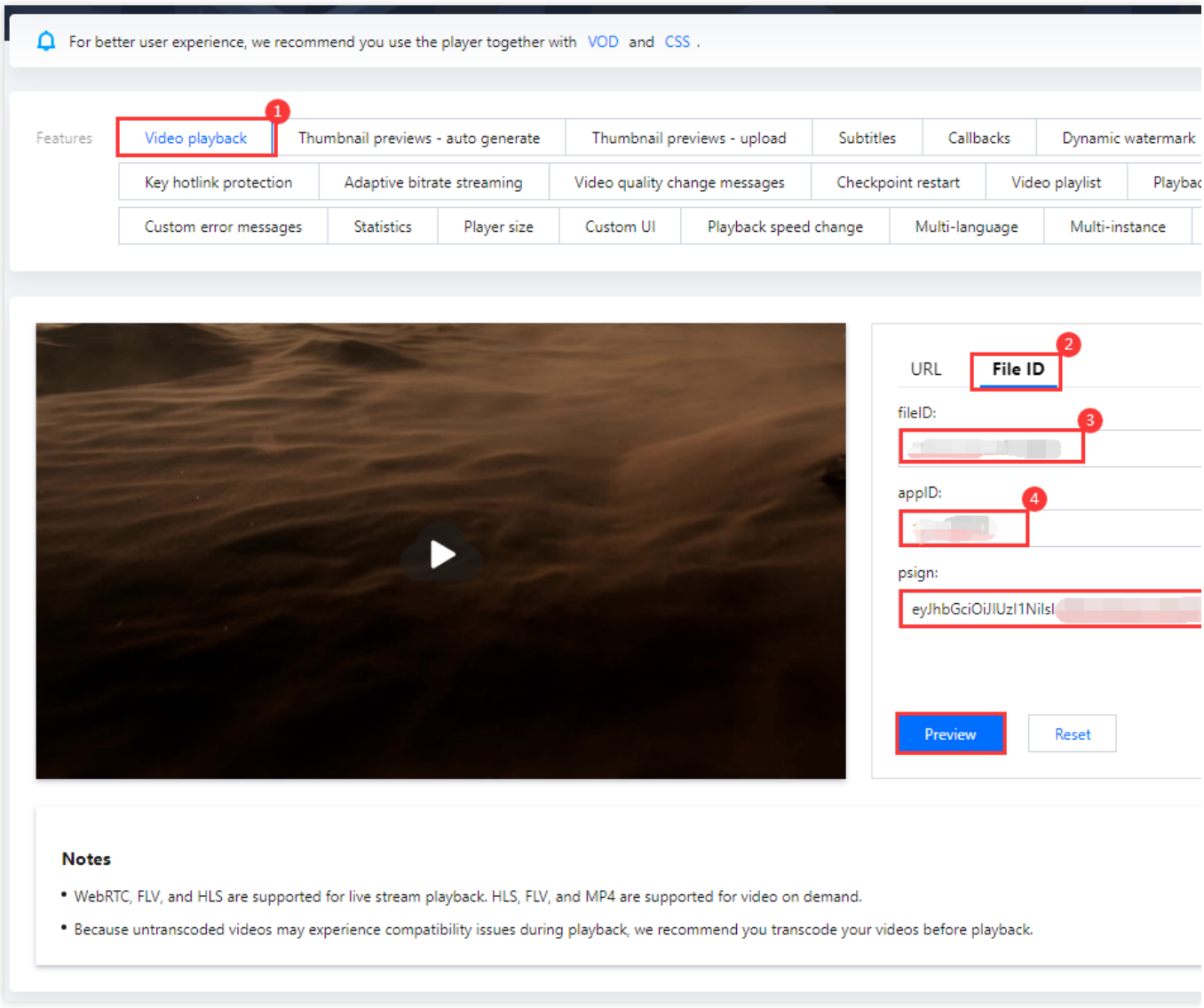
单击 **FileID 播放** 标签页

fileID 填写上一步的 FileId（243xxx814xxxxx416）

appID 填写文件所属的 appId（即上一步生成播放器签名页面的 appID）

psign 填写上一步生成的签名结果字符串

单击**预览**即可播放视频。



多端播放器 Demo

获取播放器签名后，您可以分别使用 Web、Android 和 iOS 三端的播放器 Demo 进行验证，具体请参考 Demo 的源码。

总结

学习本教程后，您已经掌握如何播放自适应码流视频。
如果您希望对视频进行加密，并播放加密后的视频，请参考 阶段4：播放加密视频。

阶段4：播放加密视频

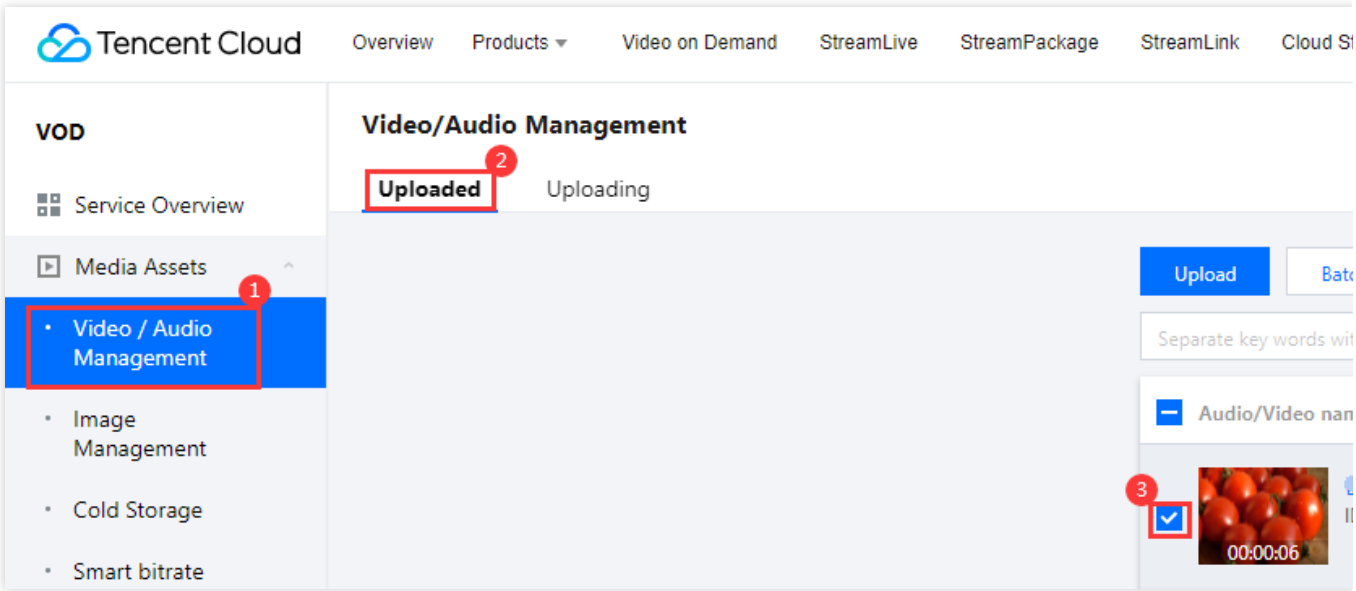
最近更新时间：2023-05-15 17:15:06

学习目标

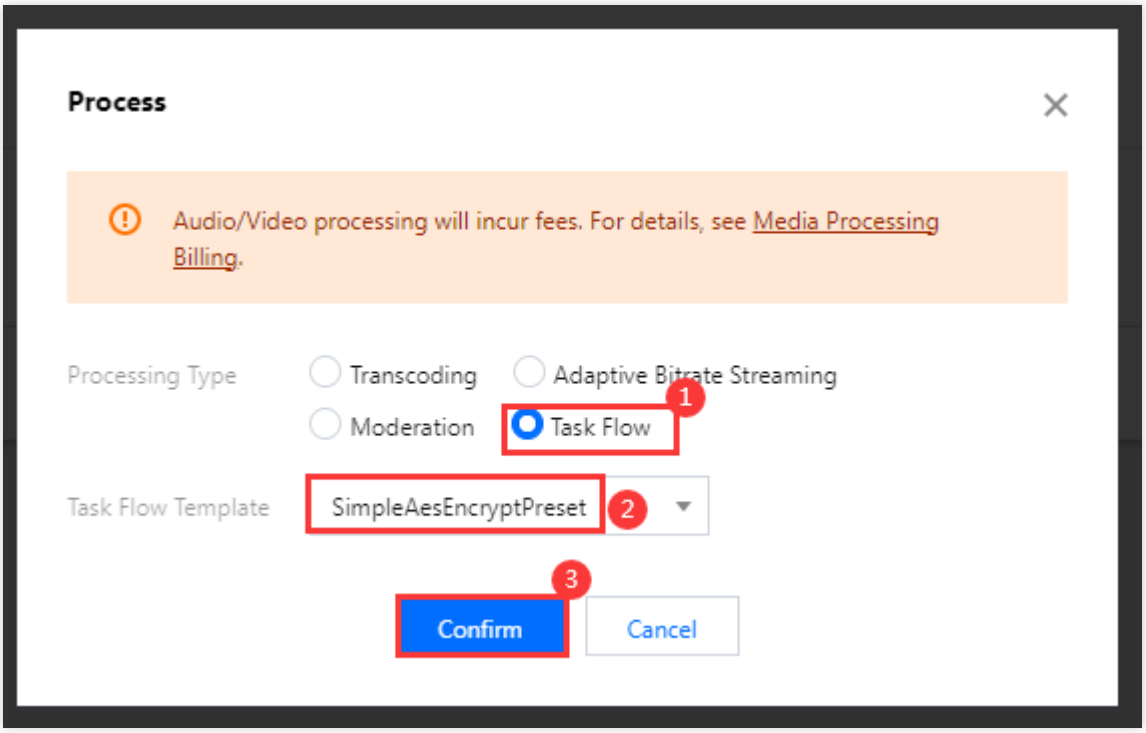
学习本阶段教程，您将了解并掌握如何对视频加密，并使用播放器播放加密视频。
阅读之前，请先确保已经学习播放器指引的 阶段1：播放原始视频 篇部分，本教程使用了 阶段1 篇开通的账号以及上传的视频。

步骤1：视频加密

1. 登录**云点播控制台 >** 应用管理，单击目标应用名称进入**媒资管理 > 音视频管理**页，勾选要处理的视频（FileId 为 243xxx814xxxxx416），单击**任务流**。



2. 在**媒体处理**界面：
处理类型 选择 **任务流**。
任务流模板 选择 **SimpleAesEncryptPreset**。



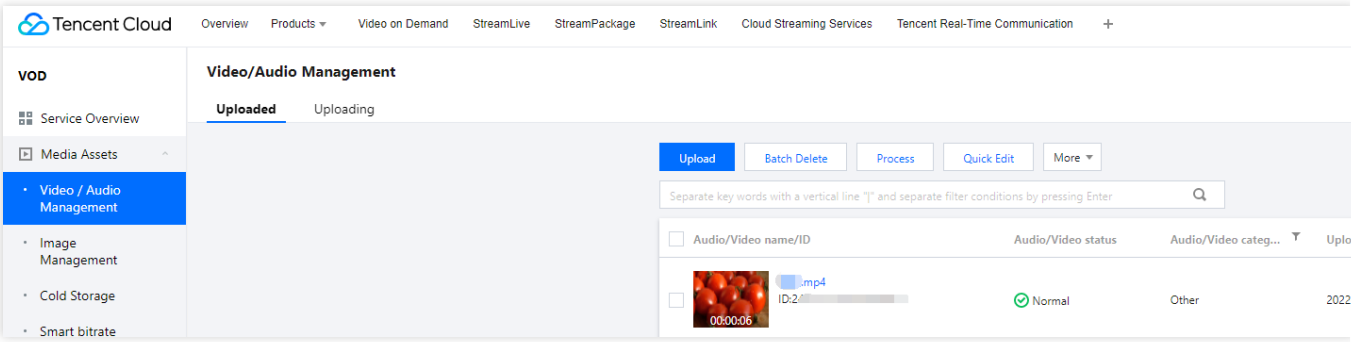
说明：

SimpleAesEncryptPreset 是预置任务流：使用12模板转自适应码流，10模板截图做封面，10模板截雪碧图。

12模板自适应码流是转出加密的多码率输出。

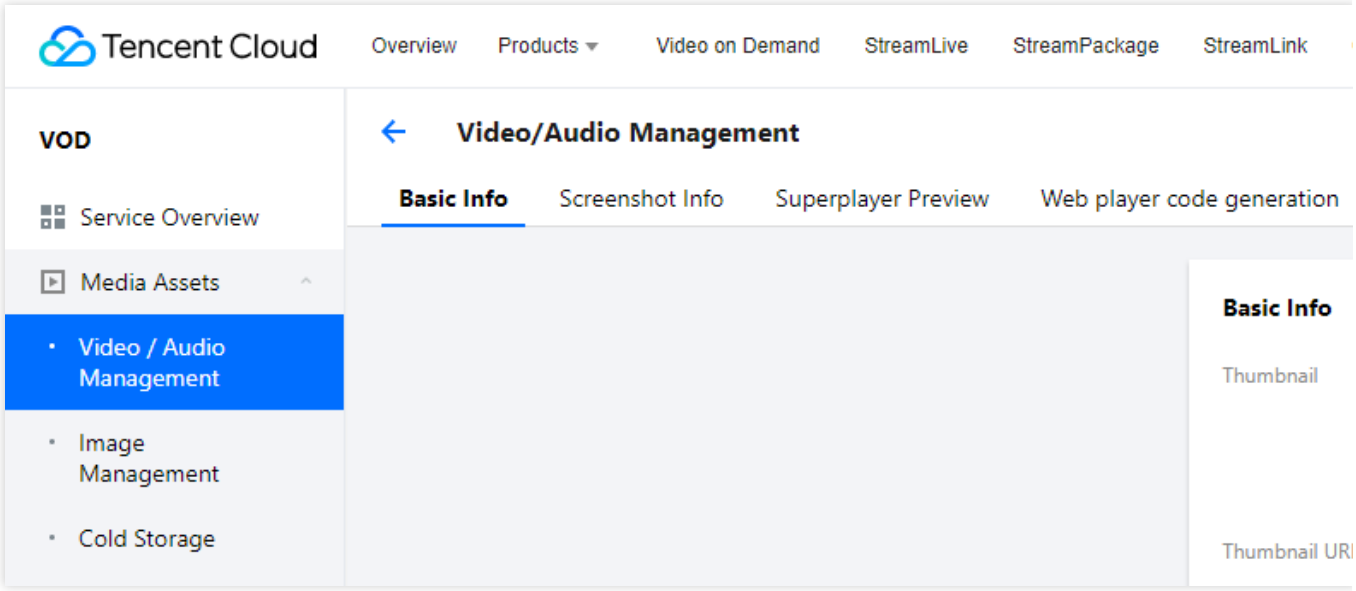
3. 单击**确定**，等待列表中的“任务状态”从“处理中”变为“已完成”，表示视频已处理完毕。

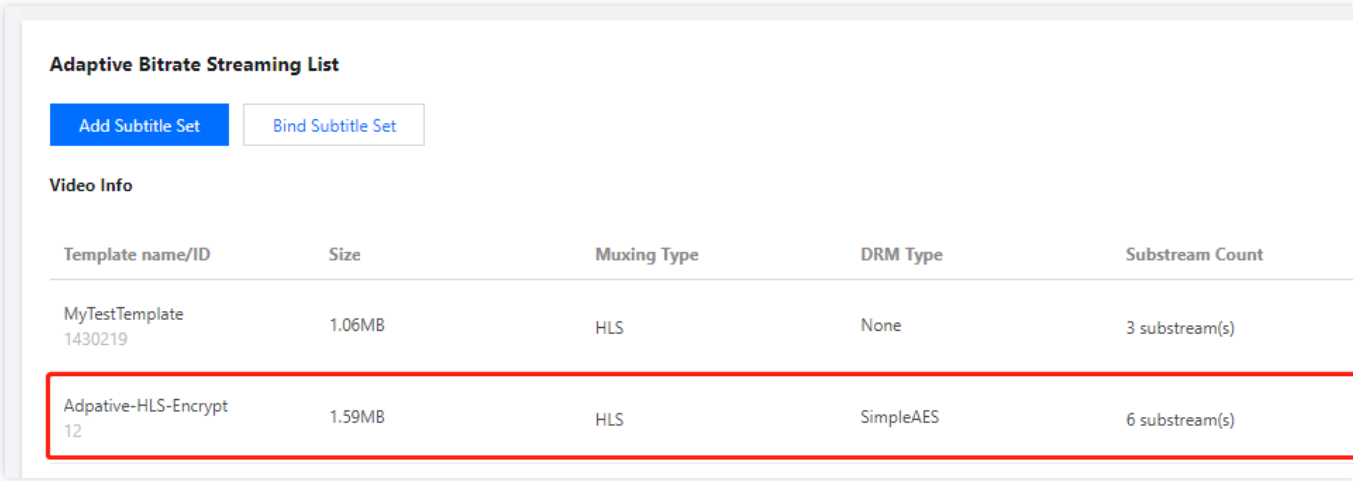
4. 进入**媒资管理** > **音视频管理**，单击发起加密的视频条目右侧的**管理**，进入管理页面。



4.1 在**基本信息**模块可以查看：

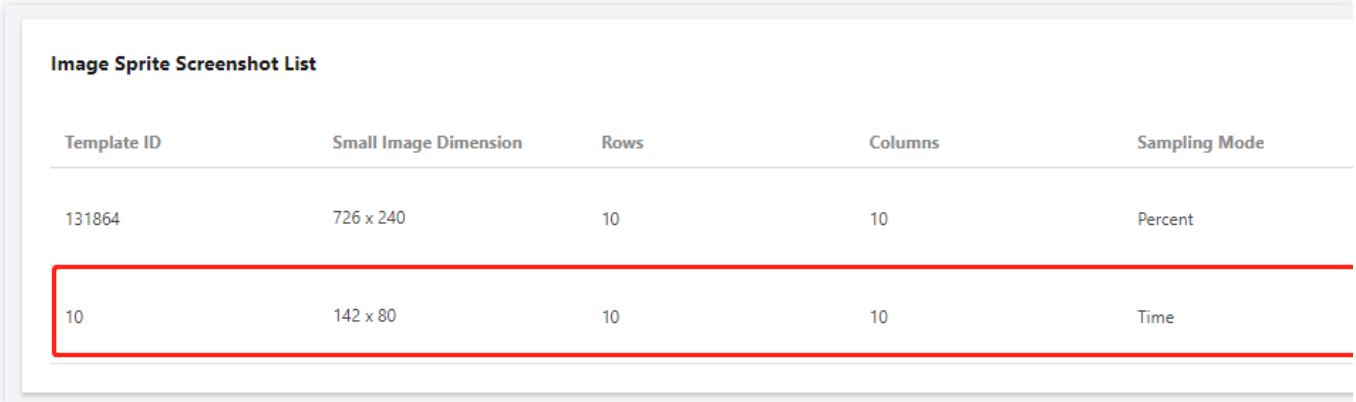
可以看到生成的封面，以及加密的自适应码流输出（模板 ID 为 12）。





4.2 选择**截图信息**页签：

可以看到生成的雪碧图（模板 ID 为 10）。



步骤2：生成播放器签名

本步骤，我们使用签名工具快速生成播放器签名，用于播放器播放视频。

1. 登录**云点播控制台 >** 应用管理，单击目标应用名称进入应用管理页，选择**分发播放设置 > 播放器签名工具**，填写如下信息：

视频 fileId 填写 步骤1 的 FileId（243xxx814xxxxx416）。

签名过期时间戳 播放器签名过期时间，不填表示签名不过期。

可播放的视频类型 选择 **转自适应码流(加密)。**

加密类型 选择 **私有加密（SimpleAES）。**

可播放的自适应码流模板 选择 `Adpative-HLS-Encrypt (12)` 。

用于缩略图预览的雪碧图 选择 `SpriteScreenshot (10)` 。

2. 单击**生成签名结果**，得到签名结果字符串。

步骤3：播放视频

经过步骤2，我们得到播放视频所需的三个参数： `appId` 、 `fileId` 以及播放器签名（ `psign` ），下面将展示 Web 端播放视频。

Web 端播放示例

1. 打开 Web端播放器体验，配置如下：

播放器功能选择**视频播放。**

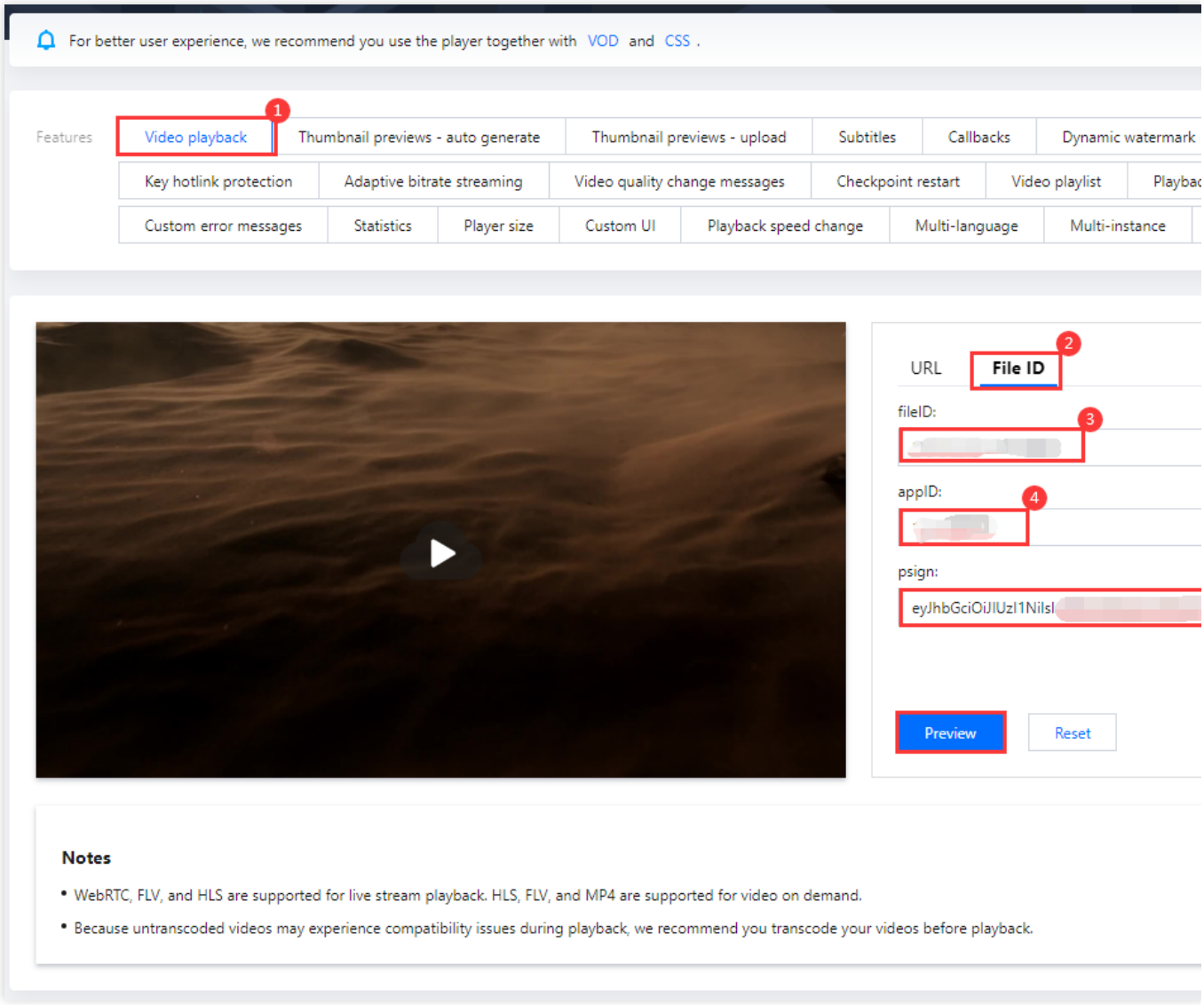
单击 **FileID 播放** 标签页。

fileID 填写上一步的 FileId（387xxxxx8142975036）。

appID 填写文件所属的 appId（即上一步生成播放器签名页面的 appID）。

psign 填写上一步生成的签名结果字符串。

2. 单击**预览**即可播放视频。



多端播放器 Demo

获取播放器签名后，您可以分别使用 Web、Android 和 iOS 三端的播放器 Demo 进行验证，具体请参考 Demo 的源码。

总结

学习本教程后，您已经掌握如何对视频加密，并在播放器中播放。

阶段5：播放长视频方案

最近更新时间：2022-12-30 16:30:24

本文针对音视频平台常见的长视频播放场景，推出播放器播放长视频教程。用户将掌握如何在 Web 端、iOS 端、Android 端播放器上播放视频，同时开启 KEY 防盗链、自动切换自适应码流、预览视频缩略图、添加视频打点信息。

学习目标

学习本阶段教程，您将掌握：

- 如何在云点播设置 KEY 防盗链，实现有效时间、播放人数、播放时长等控制
- 如何在云点播转出自适应码流（播放器能够根据当前带宽，动态选择最合适的码率播放）
- 如何在云点播设置视频打点信息
- 如何在云点播使用雪碧图做缩略图
- 如何使用播放器进行播放

阅读之前，请先确保已经学习播放器指引的 阶段1：用播放器播放视频 篇部分，了解云点播 fileid 的概念。

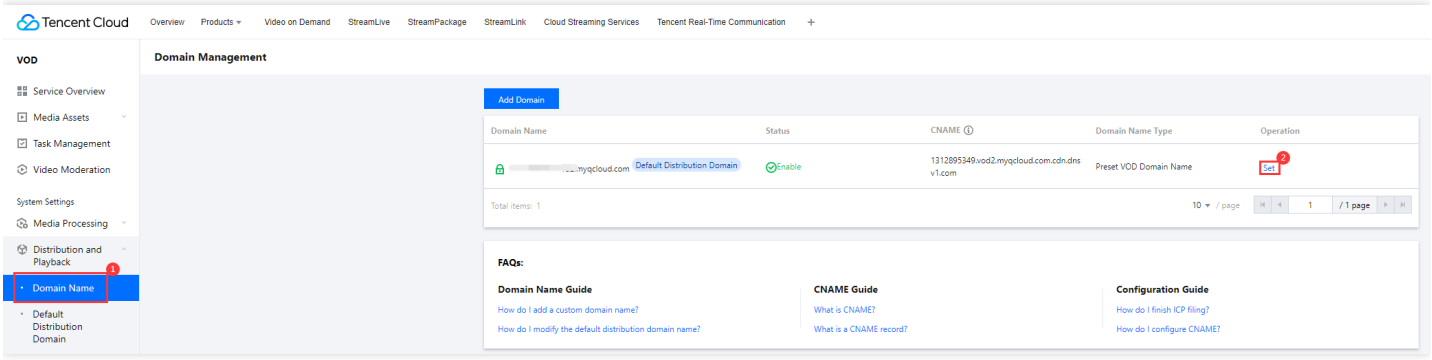
操作步骤

步骤1：开启 KEY 防盗链

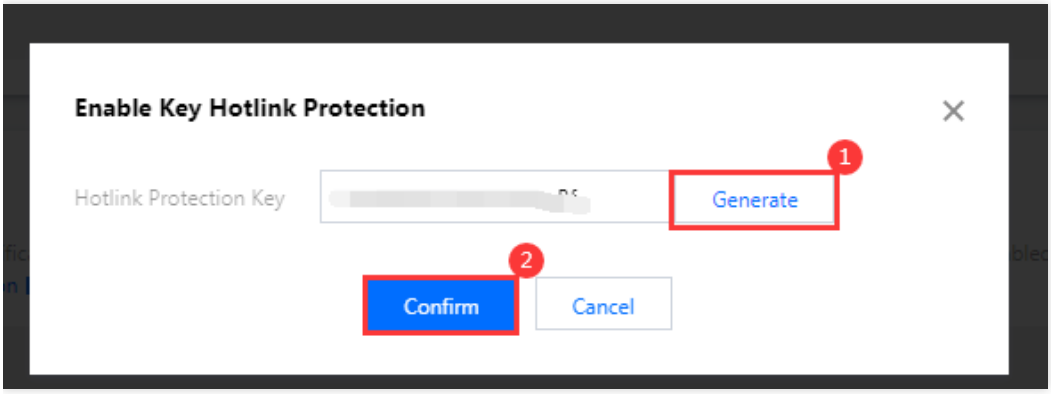
以您账号下的默认分发域名开启 KEY 防盗链为例：

> 注意：
> 请避免直接对生产环境的现网域名开启防盗链，否则可能造成现网的视频无法播放。

1. 登录云点播控制台，选择【分发播放设置】>【域名管理】，进入设置页面。



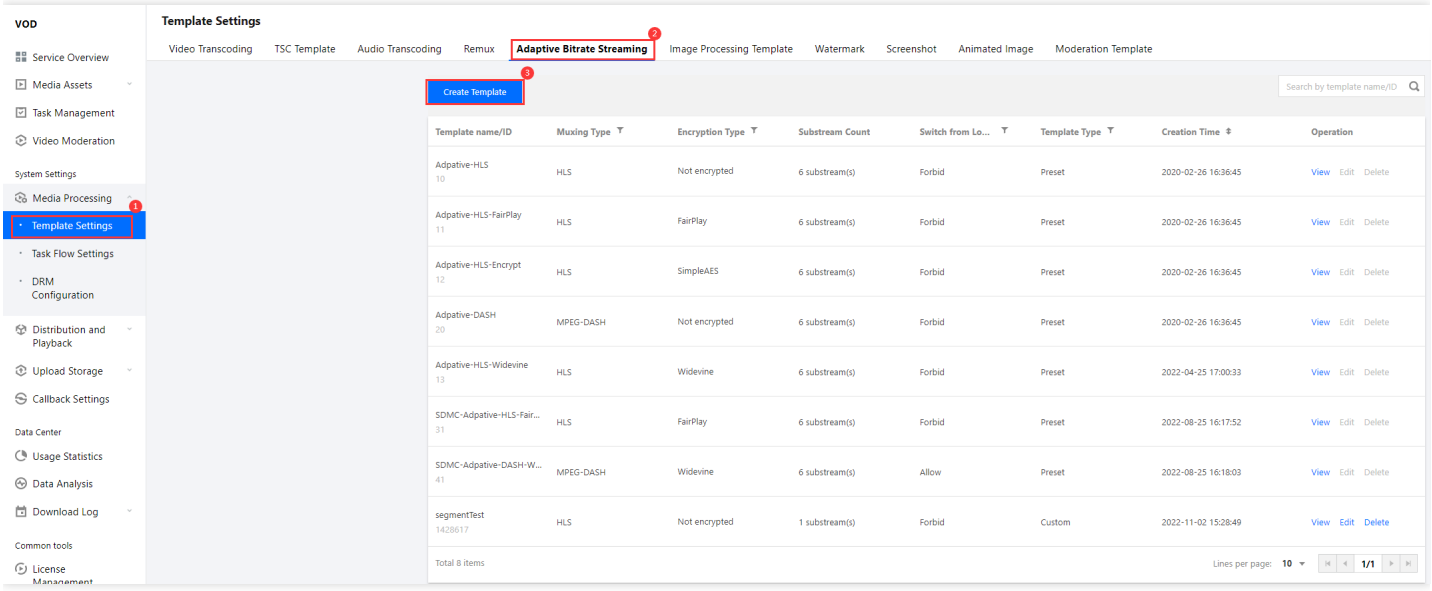
2. 单击“访问控制”页签，找到【Key 防盗链】，点击灰色按钮开启，在弹出的设置页面并单击【随机生成】生成一个随机的 Key，然后单击【确定】保存生效。



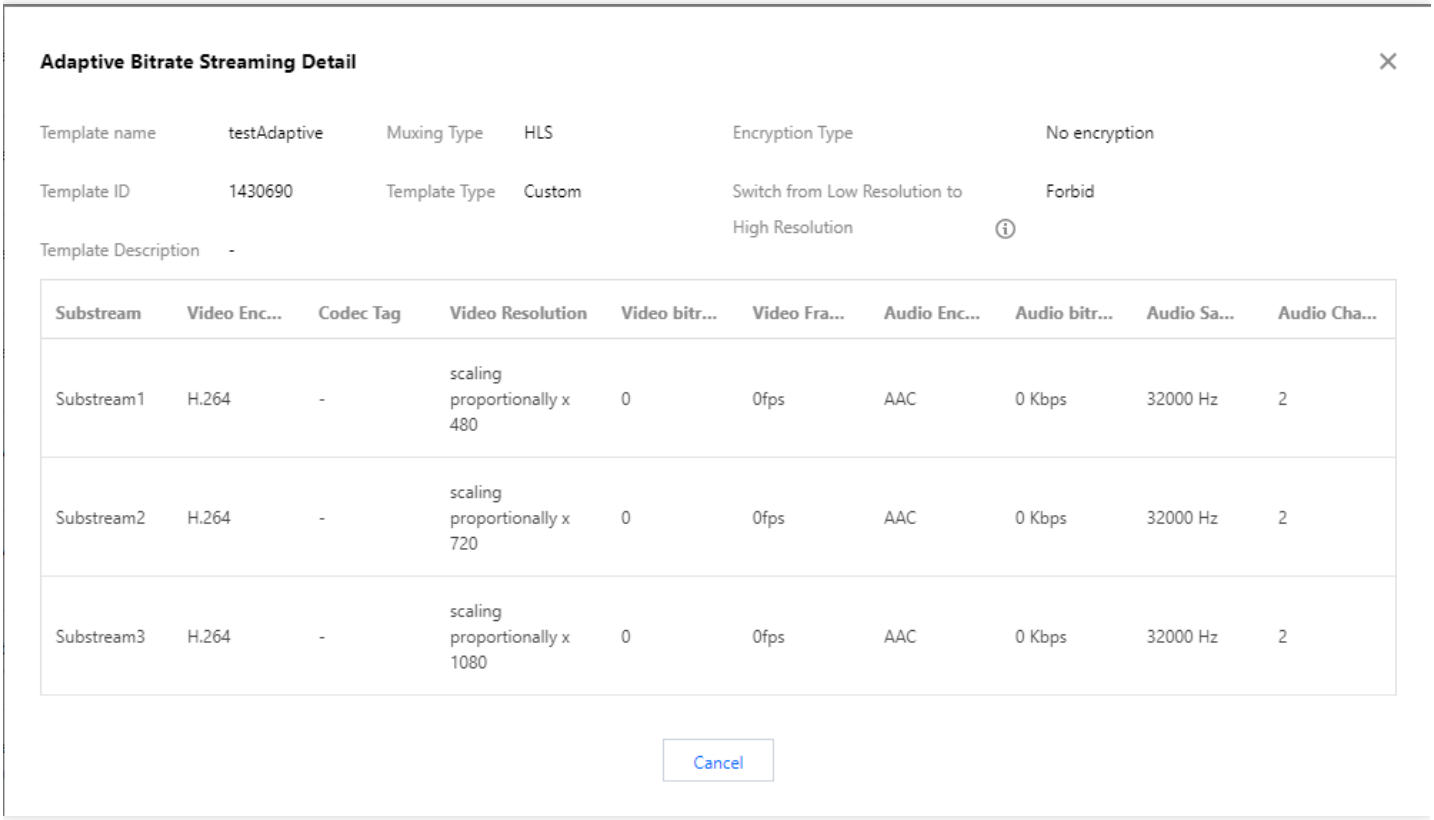
步骤2：转出自适应码流与雪碧图

本步骤，我们将指导您如何对视频转出自适应码流与雪碧图。

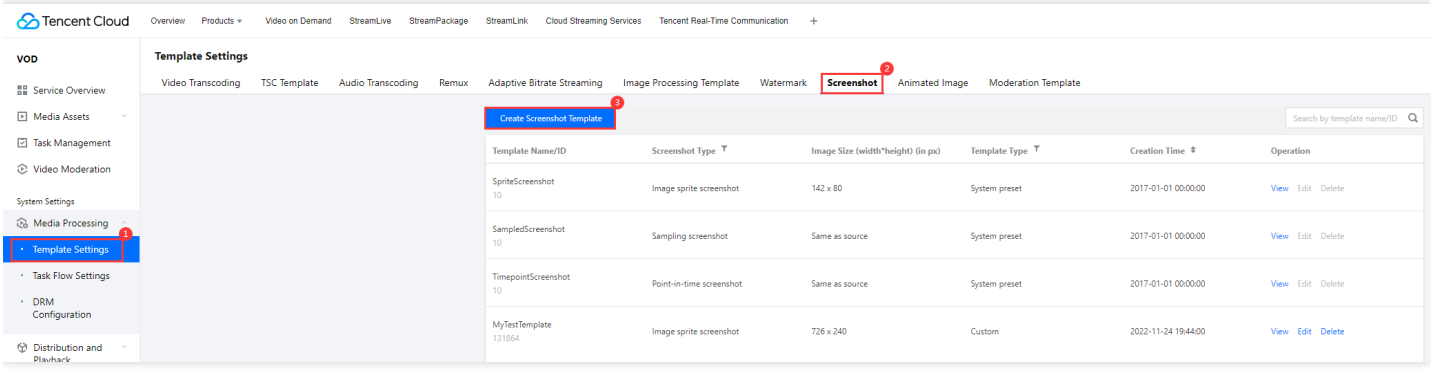
1. 登录 云点播控制台，选择**视频处理设置>模板设置>转自适应码流模板**，单击**创建转自适应码流模板**。



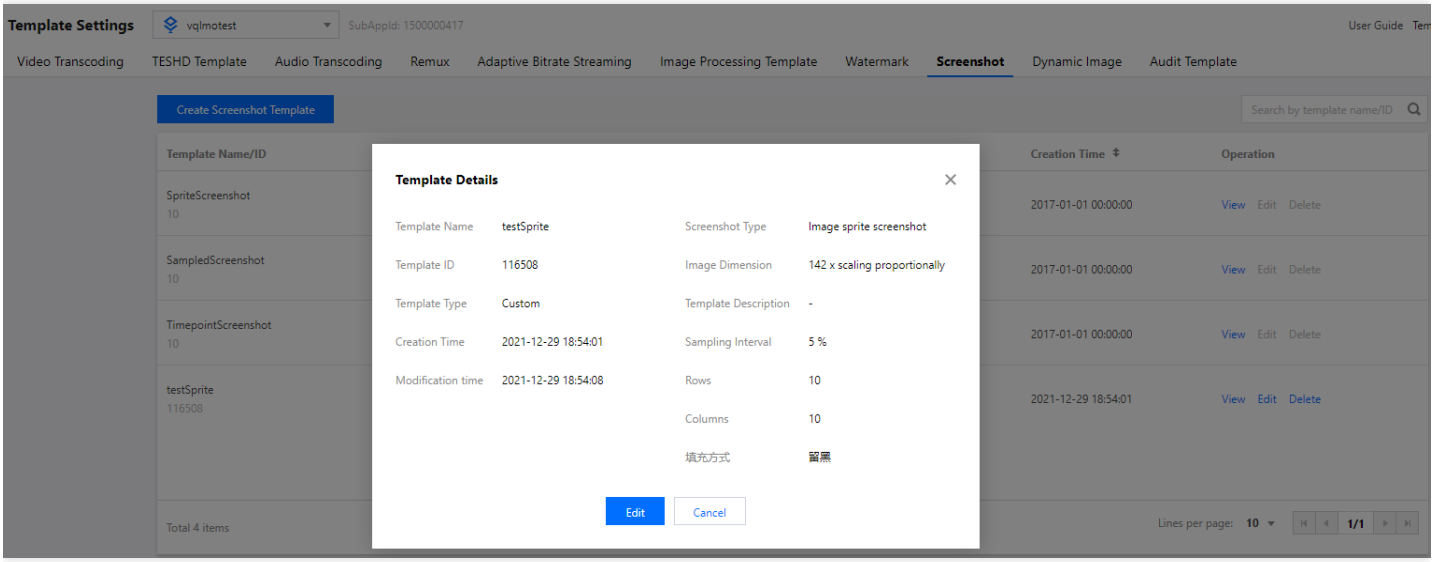
通过模板创建用户需要的自适应码流，本文创建一条名为 testAdaptive 的自适应码流模板，总共包含三条子流，分辨率分别为480p,720p和1080p。视频码率、视频帧率、音频码率则保持与原视频一致。



2. 选择**视频处理设置>模板设置>截图模板**，单击**创建截图模板**。

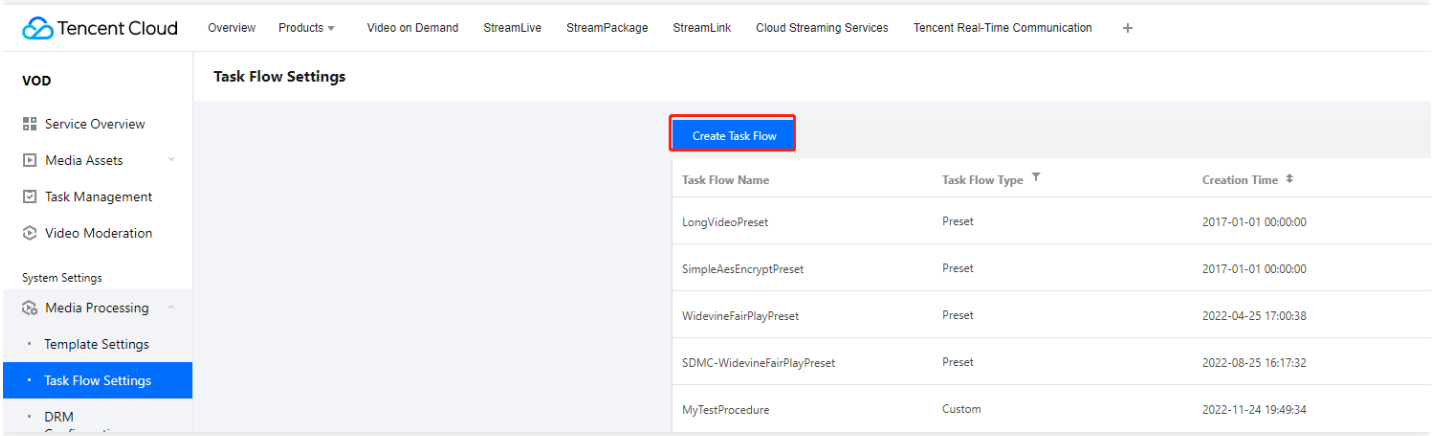


通过模板创建用户需要的雪碧图，本文创建一个名为 testSprite 的雪碧图模板，采样间隔为5%，小图行数：10，小图列数：10。

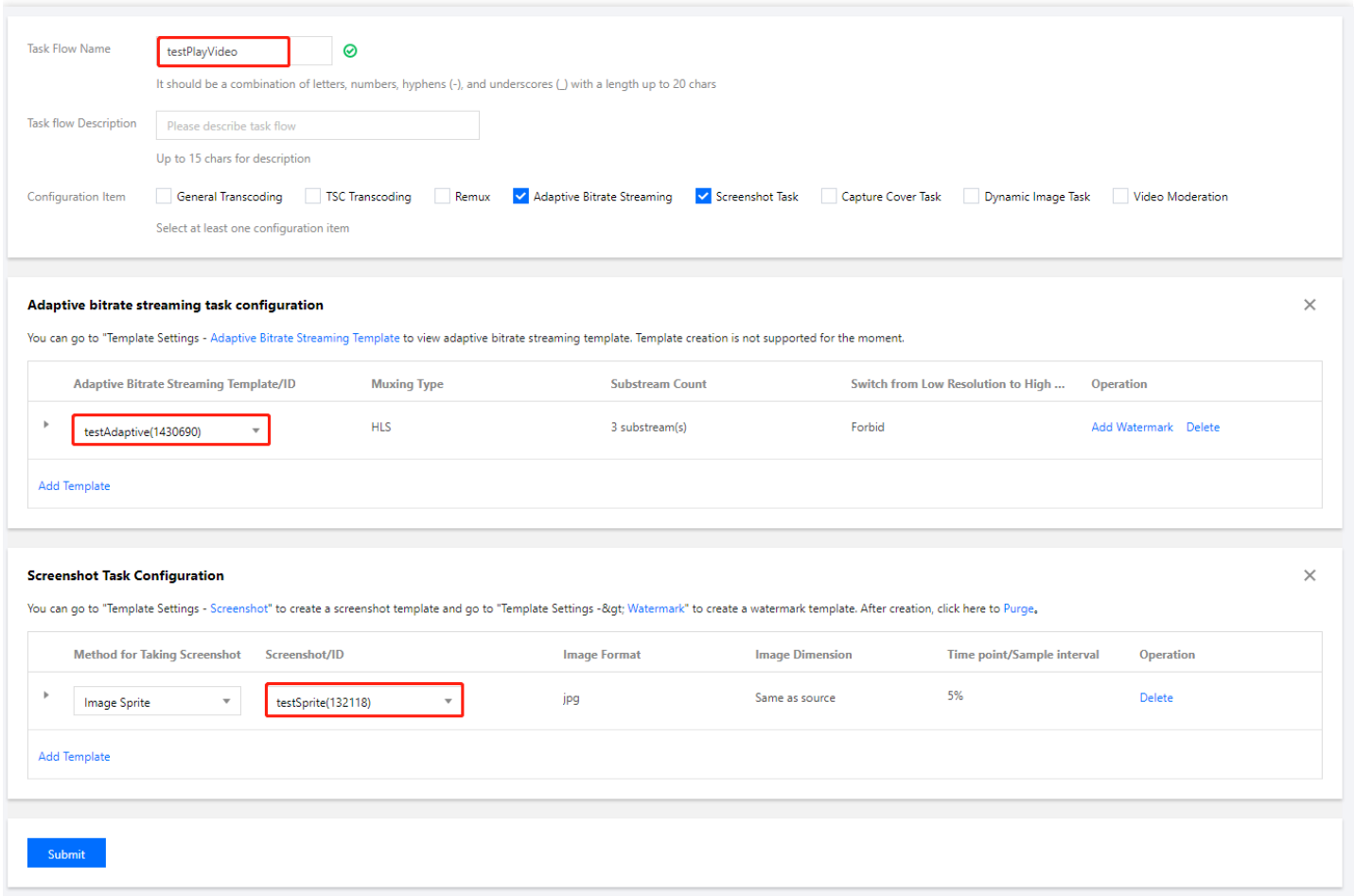


3. 通过任务流，添加自适应码流模板与雪碧图模板。

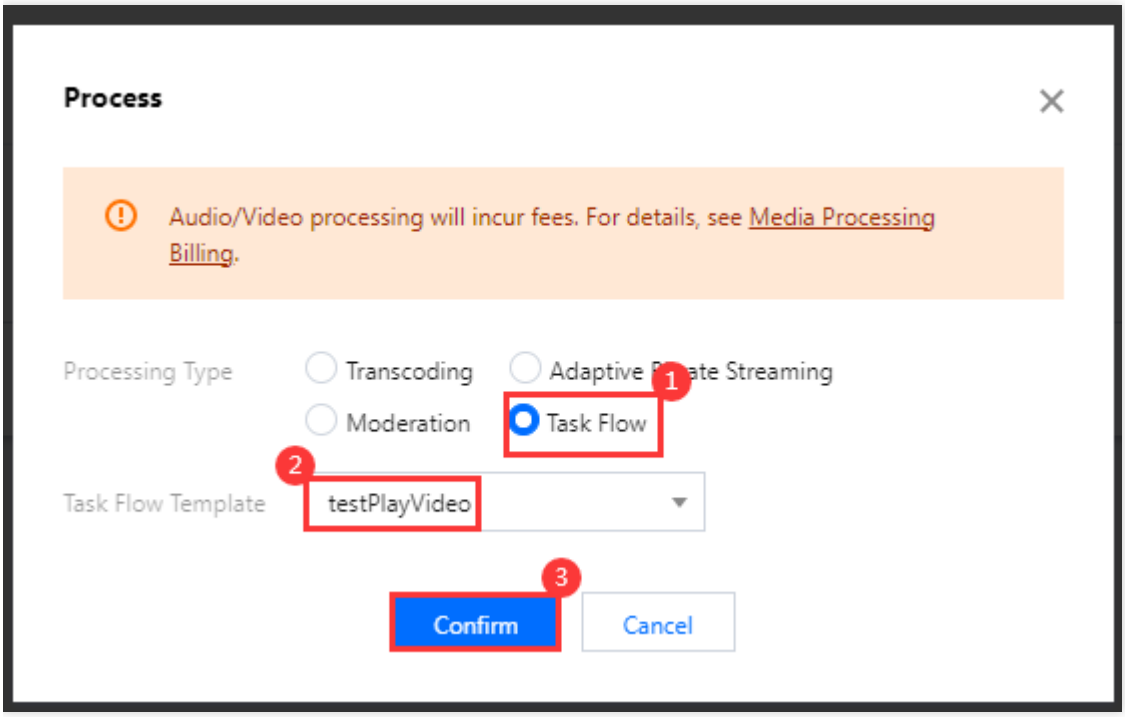
   选择**视频处理设置>任务流设置**，单击**创建任务流**。



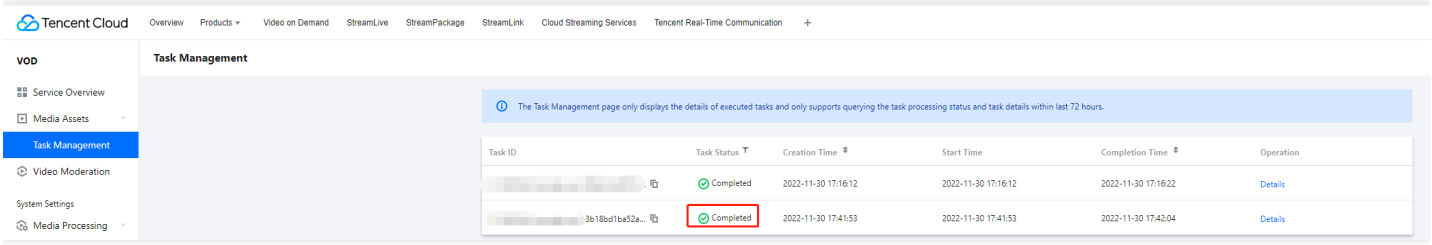
通过任务流，添加用户需要处理的任务，本文为展示播放自适应码流过程，创建了一条 testPlayVideo 的任务流，该任务流仅增加了自适应码流模板和雪碧图模板。



4. 选择**媒资管理>音视频管理**，勾选需要处理的视频（FileId 为 387xxxxx8142975036），单击**任务流**，选择任务流模板，发起任务。



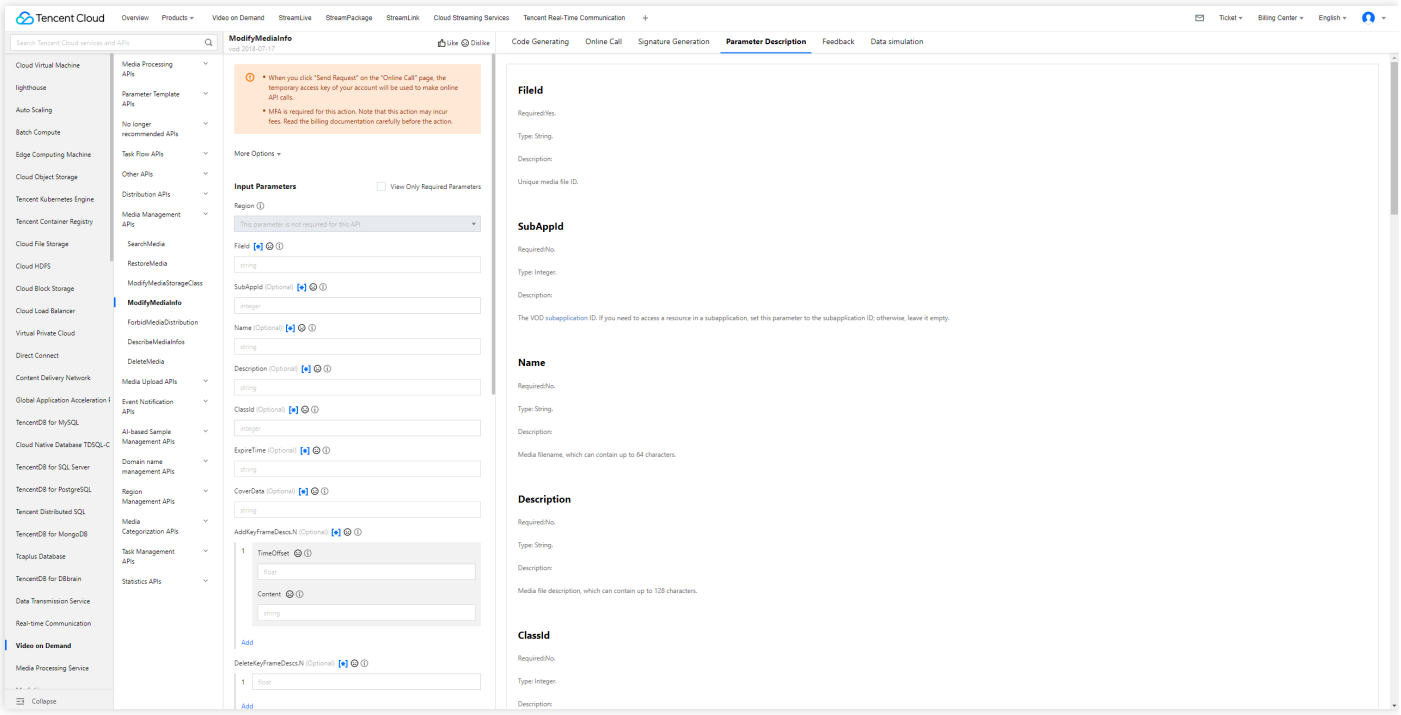
5. 至此，我们可以在**任务中心>音视频管理**中，查看任务执行情况，完成后获取任务结果。



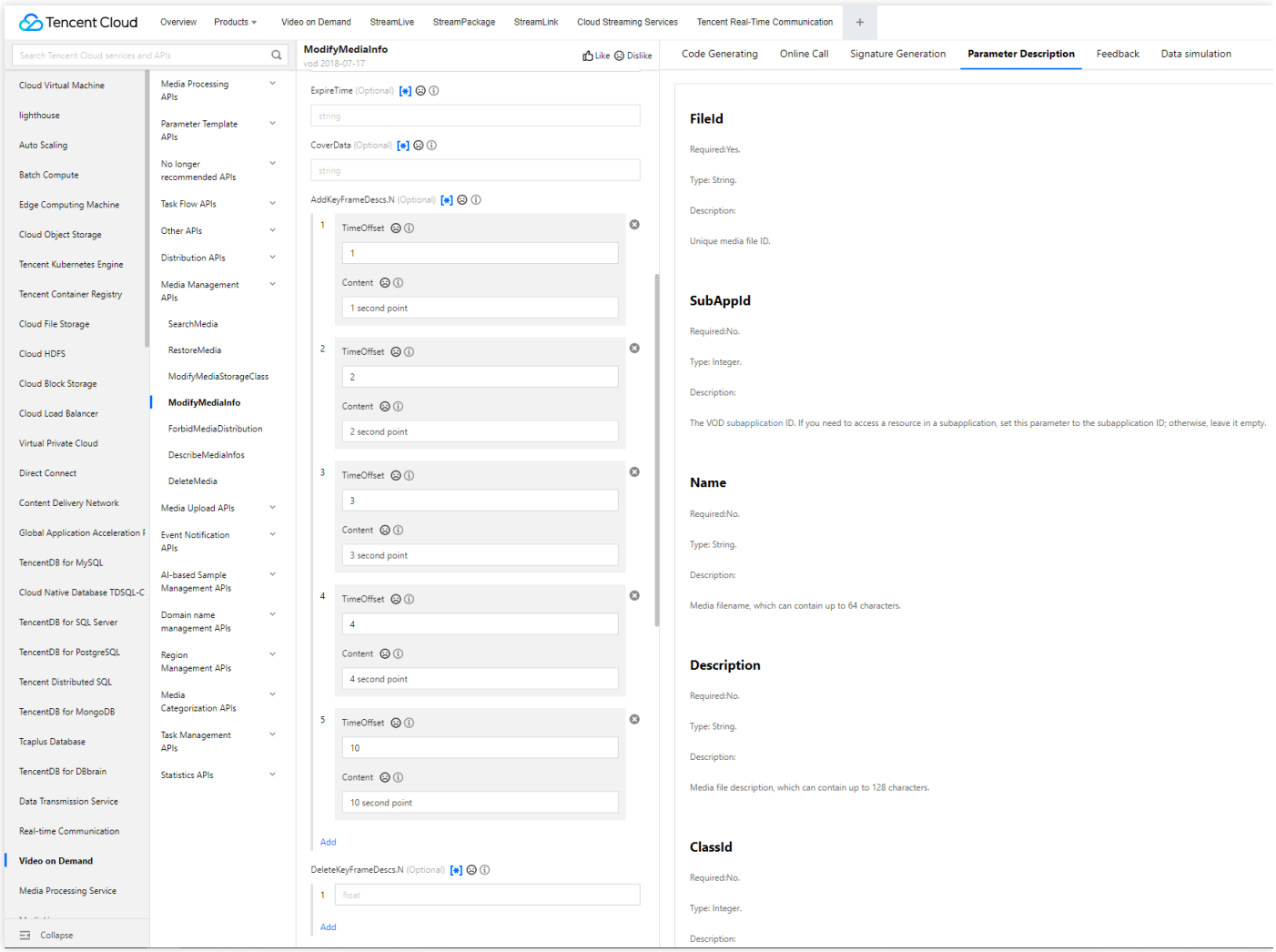
步骤3：增加视频打点信息

本步骤，我们将指导您新增的一组视频打点信息。

1. 进入云点播服务端 API 文档>**媒资管理相关接口**>**修改媒体文件属性**，单击调试，进入云 API 控制台进行调试。



2. 通过参数名称 **AddKeyFrameDescs.N** 添加指定视频打点信息



至此您已经完成了在云端上的操作，此时您在云点播已经转出自适应码流，视频雪碧图和添加了相关视频打点信息。

步骤4：生成播放器签名

本步骤，我们使用签名工具快速生成播放器签名，用于播放器播放视频。
选择 **分发播放设置>播放器签名工具**，填写如下信息：

- **视频 fileId** 填写 **步骤2** 使用的 FileId（387xxxxx8142975036）
- **签名过期时间戳** 播放器签名过期时间，不填表示签名不过期
- **可播放的视频类型** 选择 **转自适应码流(不加密)**
- **可播放的自适应码流模板** 选择 `testAdaptive (1429229)`
- **用于缩略图预览的雪碧图** 选择 `testSprite (131353)`
- **防盗链&加密配置** 开关打开，配置如下：
- **链接过期时间** 设置获取的播放链接的防盗链签名过期时间
- **最多可播放 IP 数** 设置最多允许多少个 IP 不同的终端播放

点击 **生成签名结果**，得到签名结果字符串。

步骤5：播放器端集成

经过步骤4，我们得到播放视频所需的三个参数： `appId` 、 `fileId` 以及播放器签名（ `psign` ）
本步骤，我们将指导您在 Web 端、iOS 端、Android 端播放器播放自适应码流、添加缩略图与打点信息。

- Web 端
- iOS 端
- Android 端

用户需要集成视立方播放器请参见 集成指引，引入播放器 SDK 文件之后，使用 `appId` 、 `fileId` 以及播放器签名（ `psign` ）进行播放。

播放器的构建方法为 `TCPlayer` , 通过其创建播放器实例即可播放。

1. 在 html 文件放置播放器容器

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

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

2. 使用 fileID 播放

在 index.html 页面初始化的代码中加入以下初始化脚本，传入获取到的 fileID 与 appID 即可播放。

```
var player = TCPlayer('player-container-id', { // player-container-id 为播放器容器
ID，必须与 html 中一致
fileID: '387xxxxx8142975036', // 要播放的视频 fileID
appID: '1400329073', // 要播放视频的点播账号 appID
psign:'psignxxxx' // psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencent
cloud.com/document/product/266/38099
});
```

总结

至此，您就可以在播放器播放开启了 KEY 防盗链的点播帐号下的媒体文件，查看雪碧图预览、视频打点信息和自动切换动态自适应码流。
更多播放器功能请参见 功能说明。

含 UI 集成方案
Web 接入指引
TCPlayer 集成指引

最近更新时间：2024-06-25 11:47:51

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

概述

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

协议支持

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

说明：
视频编码格式仅支持 H.264 编码。
播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。
HLS、FLV 视频在部分浏览器环境播放需要依赖 Media Source Extensions。
在不支持 WebRTC 的浏览器环境，传入播放器的 WebRTC 地址会自动进行协议转换来更好的支持媒体播放。

功能支持

功能\\浏览器	Chrome	Firefox	Edge	QQ浏览器	Mac Safari	iOS Safari	微信	Android Chrome	IE 11
播放器尺寸设置	✓	✓	✓	✓	✓	✓	✓	✓	✓
续播功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
倍速播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
缩略图预览	✓	✓	✓	✓	-	-	-	-	✓
切换 fileID 播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
镜像功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
进度条标记	✓	✓	✓	✓	✓	-	-	-	✓
HLS 自适应码率	✓	✓	✓	✓	✓	✓	✓	✓	✓
Referer 防盗链	✓	✓	✓	✓	✓	✓	✓	-	✓
清晰度切换提示	✓	✓	✓	✓	-	-	-	✓	✓
试看功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 标准加密播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 私有加密播放	✓	✓	✓	-	-	-	Android：✓ iOS: -	✓	✓
视频统计信息	✓	✓	✓	✓	-	-	-	-	-
视频数据监控	✓	✓	✓	✓	-	-	-	-	-
自定义提	✓	✓	✓	✓	✓	✓	✓	✓	✓

示文案									
自定义UI	✓	✓	✓	✓	✓	✓	✓	✓	✓
弹幕	✓	✓	✓	✓	✓	✓	✓	✓	✓
水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
幽灵水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
视频列表	✓	✓	✓	✓	✓	✓	✓	✓	✓
弱网追帧	✓	✓	✓	✓	✓	✓	✓	✓	✓

说明：

视频编码格式仅支持 H.264 编码。

Chrome、Firefox 包括 Windows、macOS 平台。

Chrome、Firefox、Edge 及 QQ 浏览器播放 HLS 需要加载 `hls.js` 。

Referer 防盗链功能是基于 HTTP 请求头的 Referer 字段实现的，部分 Android 浏览器发起的 HTTP 请求不会携带 Referer 字段。

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

准备工作

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

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

说明：

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

集成指引

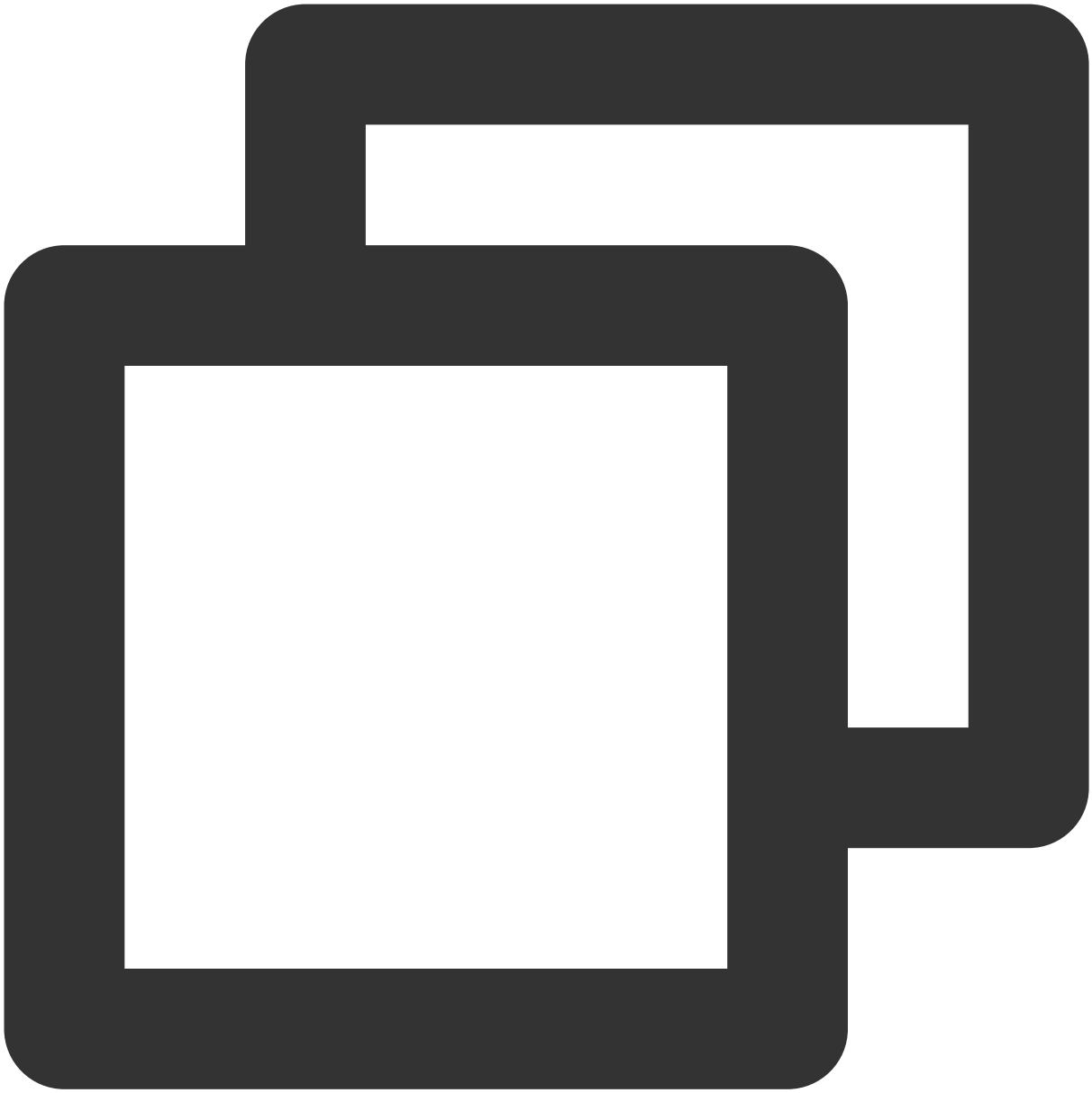
通过以下步骤，您就可以在网页上添加一个视频播放器。

步骤1：在页面中引入文件

播放器 SDK 支持 cdn 和 npm 两种集成方式：

1. 通过 cdn 集成

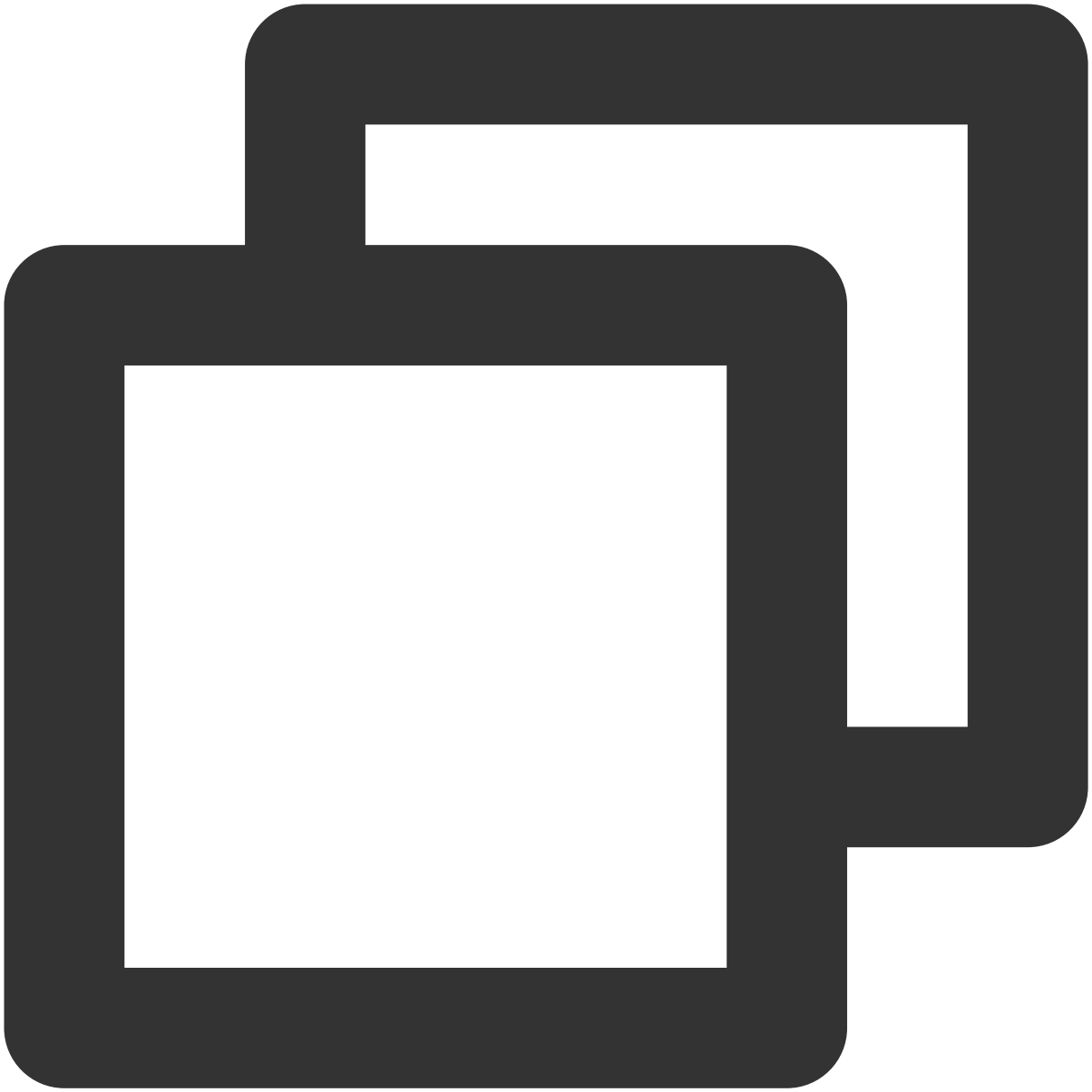
在本地的项目工程内新建 index.html 文件，html 页面内引入播放器样式文件与脚本文件。
建议在使用播放器 SDK 的时候自行部署资源，单击下载播放器资源。部署解压后的文件夹，不能调整文件夹里面的目录，避免资源互相引用异常。
如果您部署的地址为 `aaa.xxx.ccc` ，在合适的地方引入播放器样式文件与脚本文件，自行部署情况下，需要手动引用资源包文件夹 libs 下面的依赖文件，否则会默认请求腾讯云 cdn 文件。



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

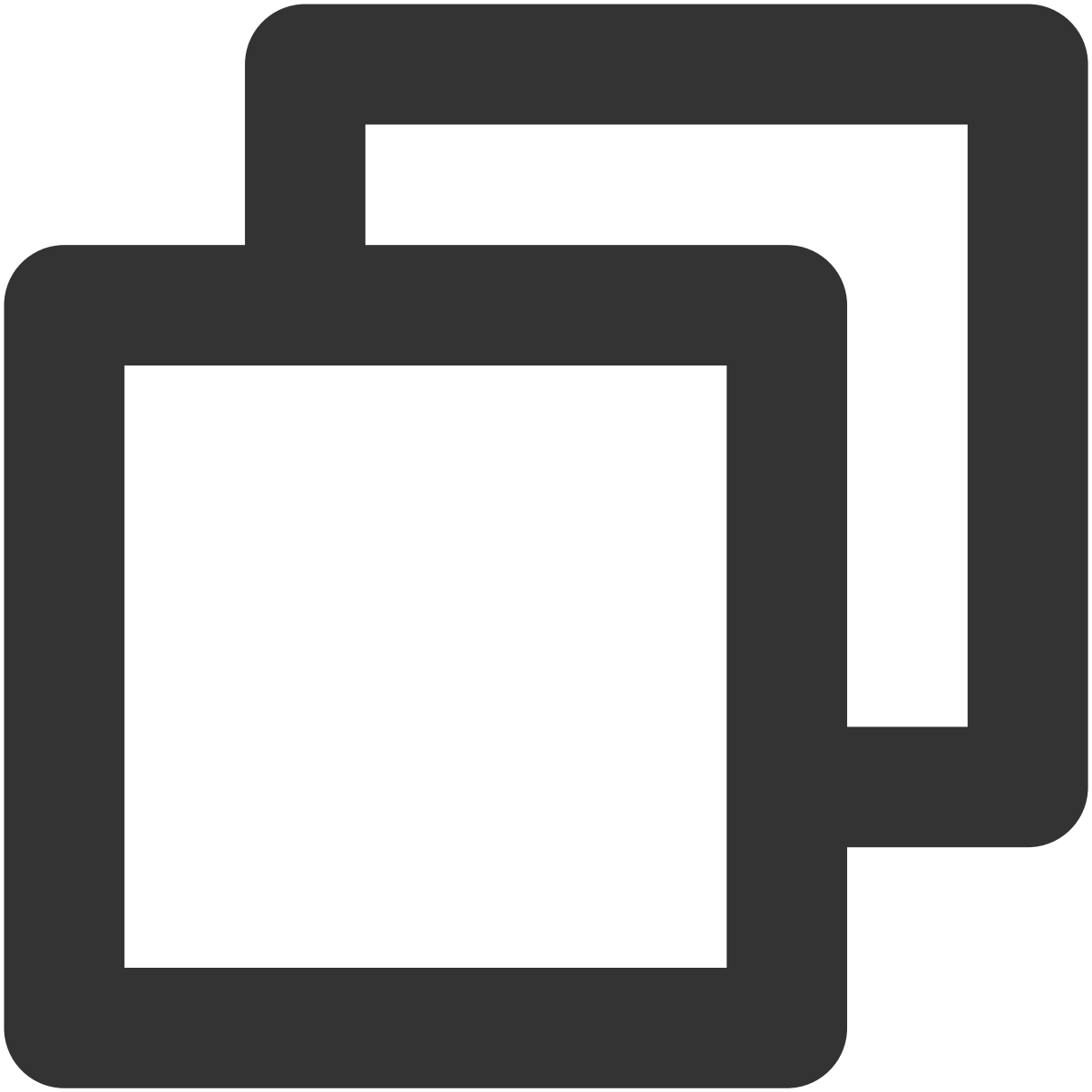
2. 通过 npm 集成

首先安装 tcplayer 的 npm 包：



```
npm install tcplayer.js
```

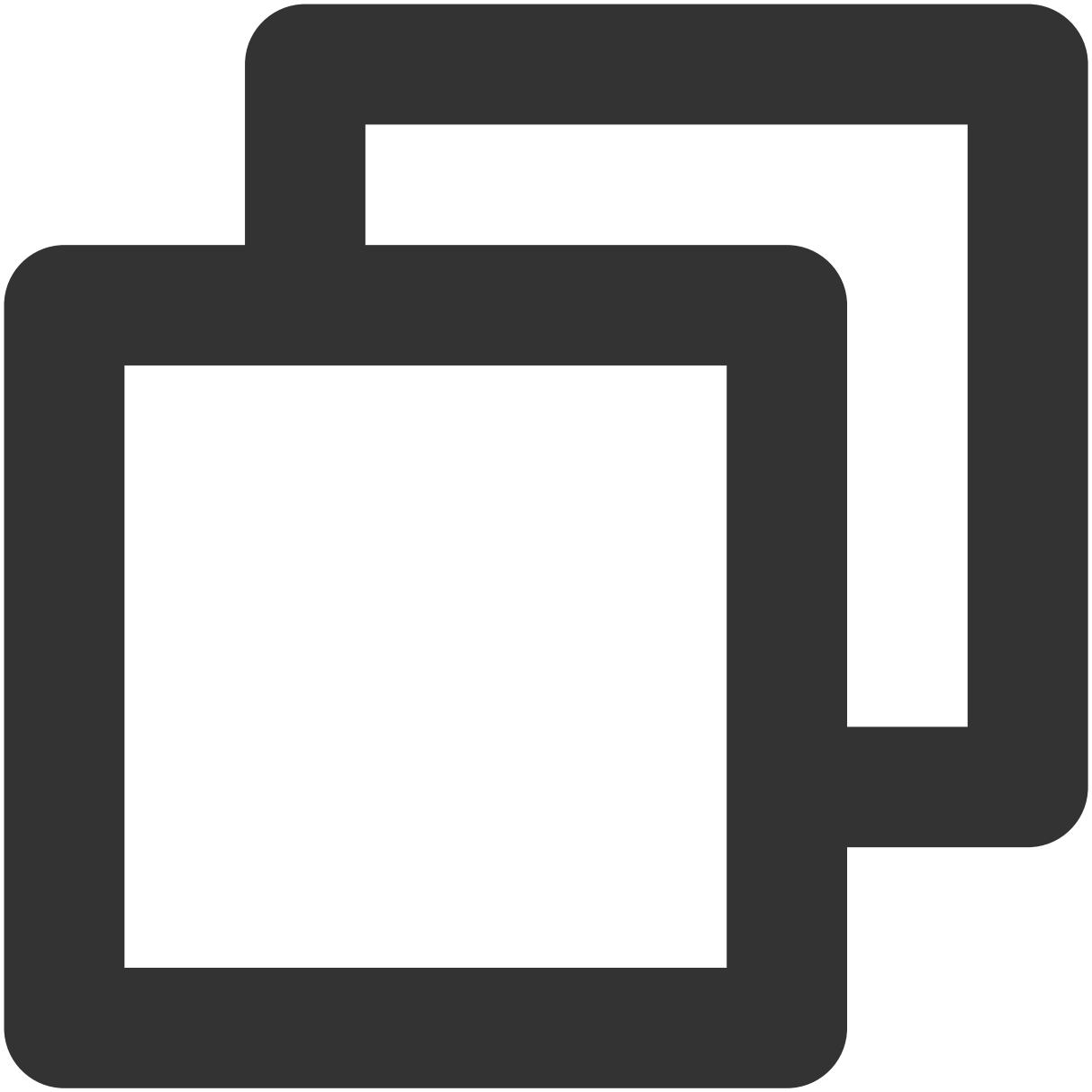
导入 SDK 和样式文件：



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

步骤2：放置播放器容器

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



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

说明：

播放器容器必须为 `<video>` 标签。

示例中的 `player-container-id` 为播放器容器的 ID，可自行设置。

播放器容器区域的尺寸，建议通过 CSS 进行设置，通过 CSS 设置比属性设置更灵活，可以实现例如铺满全屏、容器自适应等效果。

示例中的 `preload` 属性规定是否在页面加载后载入视频，通常为了更快的播放视频，会设置为 `auto` ，其他可选值： `meta` （当页面加载后只载入元数据）， `none` （当页面加载后不载入视频），移动端由于系统限制不会自动加载视频。

`playsinline` 和 `webkit-playsinline` 这几个属性是为了在标准移动端浏览器不劫持视频播放的情况下实现行内播放，此处仅作示例，请按需使用。

设置 `x5-playsinline` 属性在 TBS 内核会使用 X5 UI 的播放器。

步骤3：播放器初始化

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

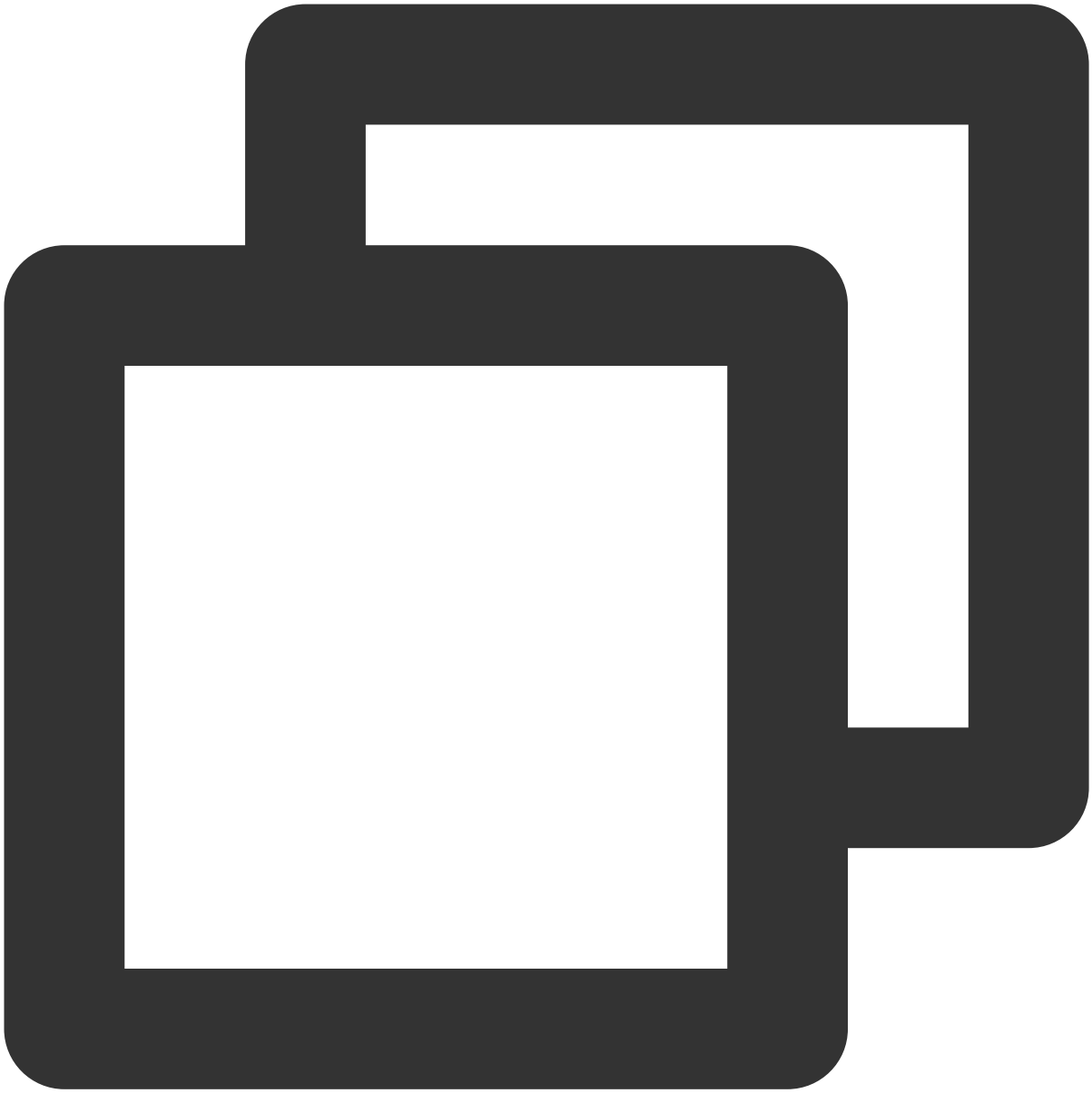
点播播放：播放器可以通过 FileID 播放腾讯云点播媒体资源，云点播具体流程请参见 使用播放器播放 文档。

直播播放：播放器通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播 URL 生成方式可参见 自主拼装直播 URL。

通过 URL 播放（直播、点播）

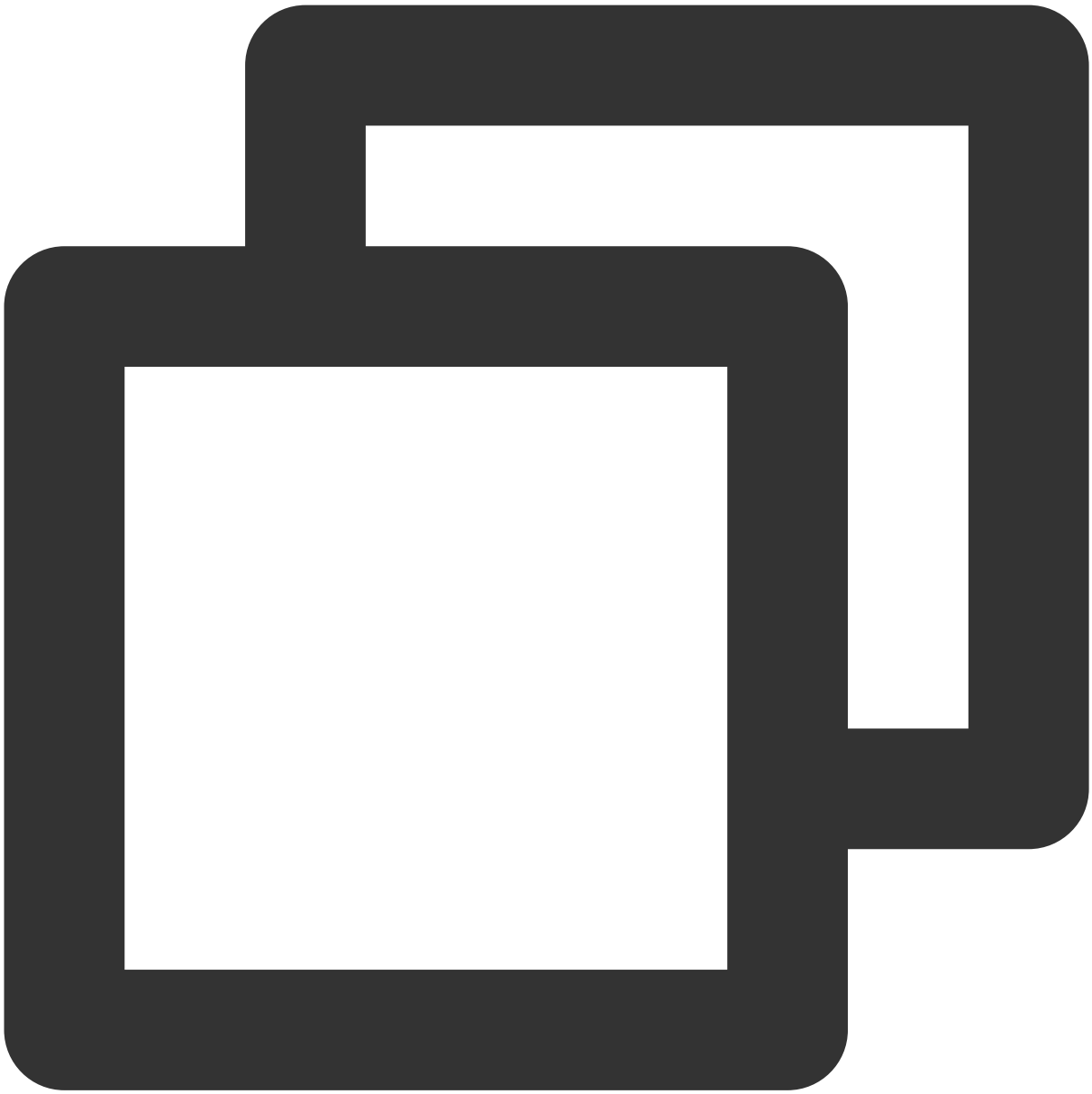
通过 FileID 播放（点播）

在页面初始化之后，调用播放器实例上的方法，将 URL 地址传入方法。



```
// player-container-id 为播放器容器 ID，必须与 html 中一致
var player = TCPlayer('player-container-id', {
    sources: [{
      src: '请替换你的播放地址',
    }],
    licenseUrl: '请替换你的 licenseUrl', // license 地址，参考准备工作部分，在视立方控制台申
    language: '请替换你的设置语言', // 设置语言 en | zh-CN
});

// player.src(url); // url 播放地址
```

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



```
var player = TCPlayer('player-container-id', { // player-container-id 为播放器容器 ID
    fileID: '请传入你的 fileID', // 请传入需要播放的视频 fileID
    appID: '请传入你的 appID', // 请传入点播账号的 appID
    // 请传入播放器签名 psign，签名介绍和生成方式参见链接：https://cloud.tencent.com/docume
    psign:'请传入你的播放器签名 psign',
    licenseUrl: '请传入你的 licenseUrl', // 参考准备工作部分，在视立方控制台申请 license 后
    language: '请替换你的设置语言', // 设置语言 en | zh-CN
});
```

注意：

要播放的视频建议使用腾讯云转码，原始视频无法保证在浏览器中正常播放。

步骤4: 更多功能

播放器可以结合云点播的服务端能力实现高级功能，比如自动切换自适应码流、预览视频缩略图、添加视频打点信息等。这些功能在 播放长视频方案 中有详细的说明，可以参考文档实现。

此外，播放器还提供更多其他功能，功能列表和使用方法请参见 功能展示 页面。

TCPlayer 清晰度配置说明

最近更新时间：2024-05-13 17:49:25

在播放过程中，您可以通过自动或手动切换视频清晰度，以适应不同尺寸的播放设备和网络环境，从而提高观看体验。本文将从以下几个场景进行说明。

直播场景

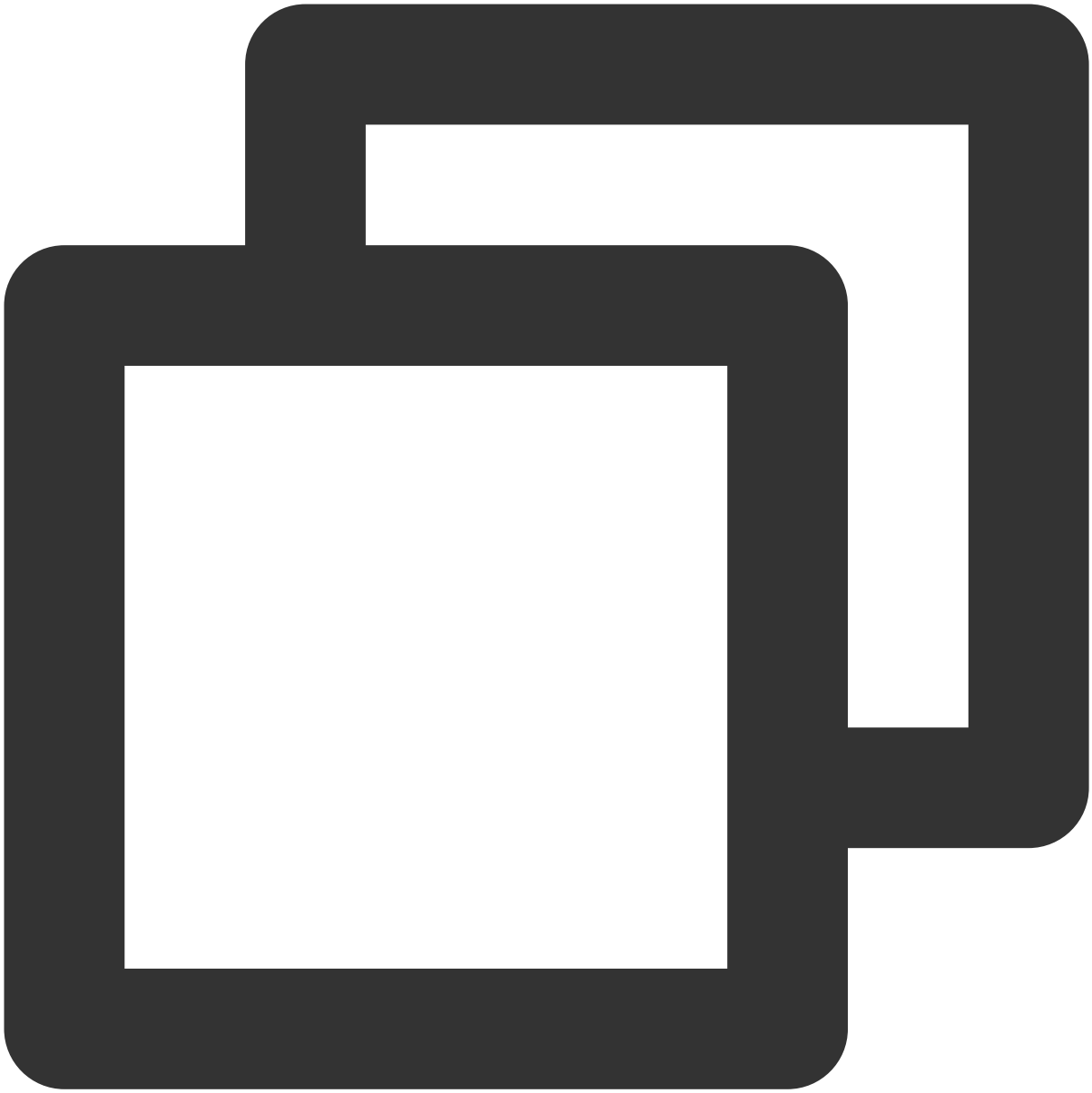
直播场景以 URL 的形式来播放视频，初始化播放器时，可以通过 sources 字段指定所要播放的 URL。或者在初始化播放器之后，调用播放器实例上的 src 方法进行播放。

1. 自适应码率（ABR）

自适应码率地址在切换时可以做到无缝衔接，切换过程不会出现中断或跳变，实现了观感和听感的平滑过渡。使用该技术也比较简单，仅需将播放地址传给播放器，播放器会自动解析子流，并将清晰度切换组件渲染到控制条上。

示例1: 播放 HLS 自适应码率地址

在初始化播放器时，传入自适应码率地址，播放器将自动生成清晰度切换组件，并会根据网络状况进行自动切换。



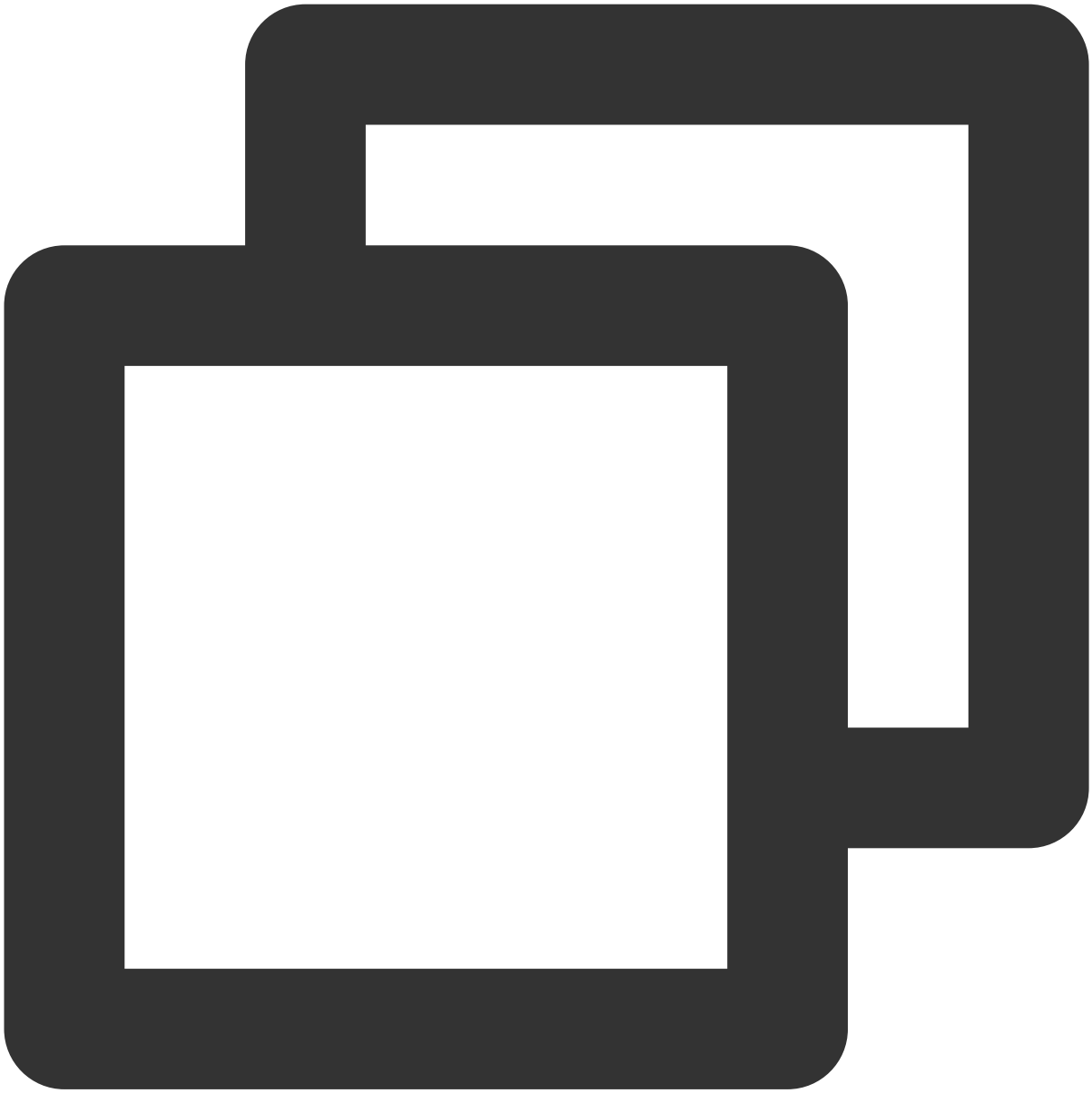
```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器I
  sources: [{
    src: 'https://hls-abr-url', // hls 自适应码率地址
  }],
});
```

注意：

解析 HLS 自适应码率的子流，需要依赖播放环境的 MSE API。在不支持 MSE 的浏览器环境（例如 iOS 的 Safari 浏览器），会由浏览器内部处理，根据网络情况自动切换清晰度，但无法解析出多种清晰度来供您手动切换。

示例2：播放 WebRTC 自适应码率地址

在 WebRTC 自适应码率场景，传入地址后，播放器会根据地址中的 ABR 模板自动拆解子流地址。



```
const player = TCPlayer('player-container-id',{
  sources: [{
    src: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b2
  }],

  webrtcConfig: {
    // 是否渲染多清晰度的开关，默认开启，可选
    enableAbr: true,
```

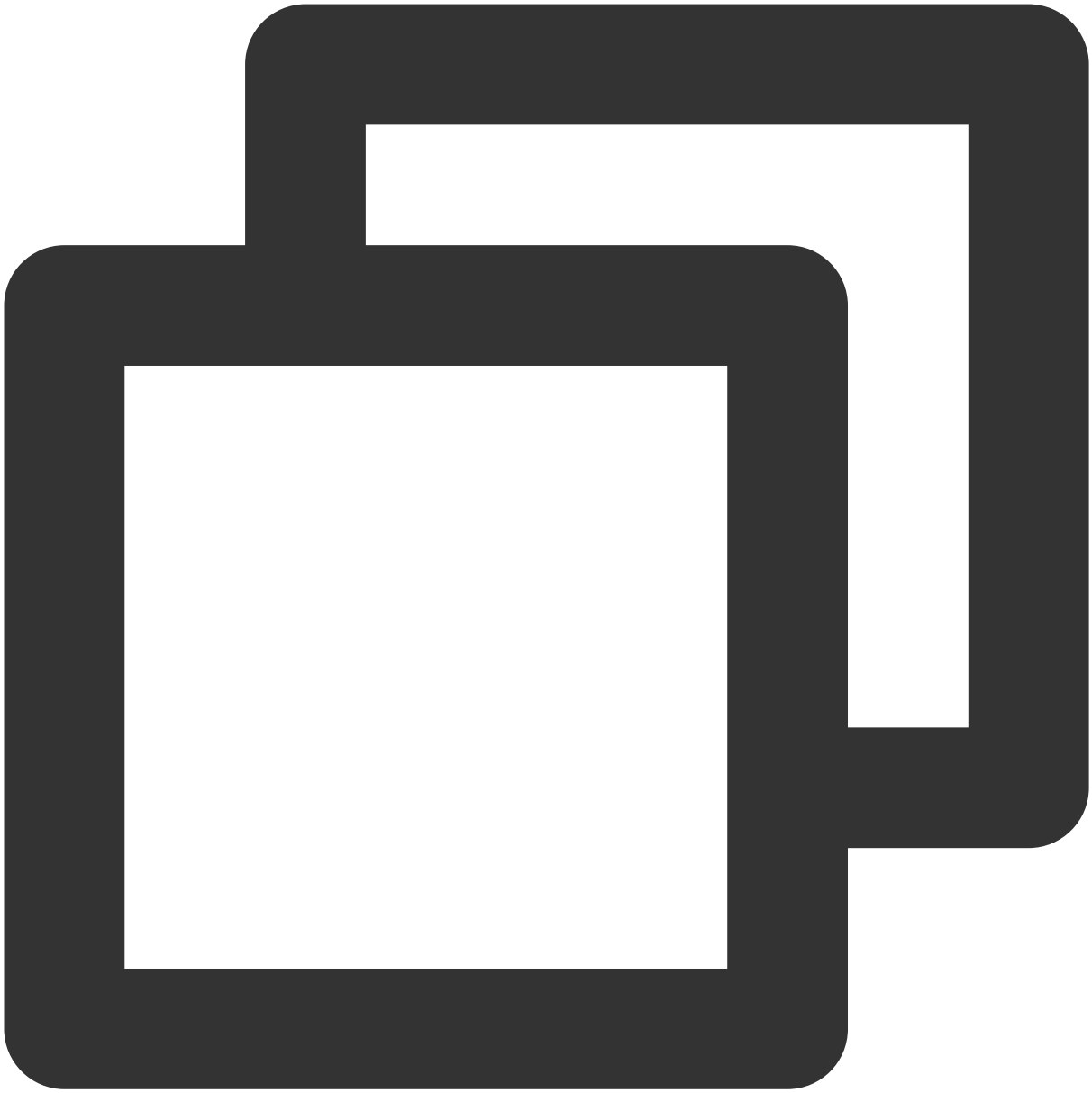
```
      // 模板名对应的label名，可选
      abrLabels: {
        d1080p: 'FHD',
        d540p: 'HD',
        d360p: 'SD',
        auto: 'AUTO',
      },
    },
  });
```

这里对 WebRTC 地址中的参数做以下说明：

1. tabr_bitrates 指定了 ABR 模板，有几个模板就会渲染出几个清晰度。如果没有单独设置清晰度的 label，模板名称（如 d1080p）将被设为清晰度名称。
2. tabr_start_bitrate 指定了起播的清晰度规格。
3. tabr_control 设置是否开启自动切换清晰度。开启后，播放器会单独渲染出自动清晰度选项。

2. 手动设置清晰度

如果播放地址不是自适应码率地址，也可以手动设置清晰度。参见如下代码：

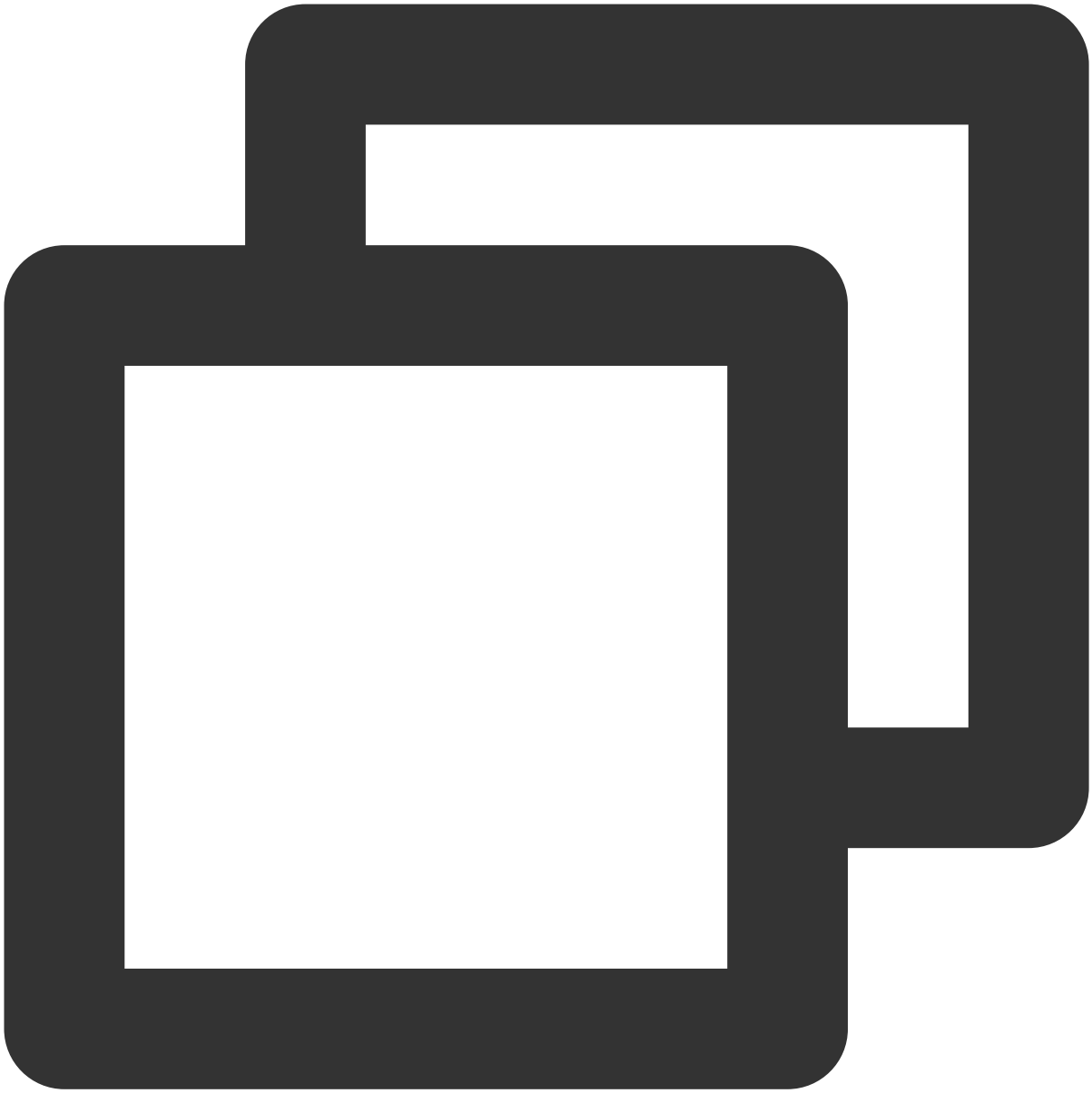


```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器I
  multiResolution:{
    // 配置多个清晰度地址
    sources:{
      'SD':[{
        src: 'http://video-sd-url',
      }],
      'HD':[{
        src: 'http://video-hd-url',
      }],
      'FHD':[{
```

```
          src: 'http://video-fhd-url',
        }]
      },
      // 配置每个清晰度标签
      labels:{
        'SD':'标清','HD':'高清','FHD':'超清'
      },
      // 配置各清晰度在播放器组件上的顺序
      showOrder:['SD','HD','FHD'],
      // 配置默认选中的清晰度
      defaultRes: 'SD',
    },
  });
```

点播场景

在点播场景下，如果通过 fileID 播放，播放哪种规格的文件（原始文件、转码文件、自适应码率文件）以及自适应码率文件子流的清晰度，都是在播放器签名中设置的。您可以参见指引 播放自适应码流视频，以便于您了解点播场景下播放视频的整个流程。
计算播放器签名时，可以通过 contentInfo 字段中的 resolutionNames 来设定不同分辨率的子流的展示名字。不填或者填空数组则使用默认配置。



```
resolutionNames: [{
  MinEdgeLength: 240,
  Name: '240P',
}, {
  MinEdgeLength: 480,
  Name: '480P',
}, {
  MinEdgeLength: 720,
  Name: '720P',
}, {
  MinEdgeLength: 1080,
```

```
    Name: '1080P',
  }, {
    MinEdgeLength: 1440,
    Name: '2K',
  }, {
    MinEdgeLength: 2160,
    Name: '4K',
  }, {
    MinEdgeLength: 4320,
    Name: '8K',
  }]
```

播放时的子流数量取决于转码时根据不同的自适应码率模板转换出的子流数。这些子流会依据短边长度落在由 resolutionNames 设定的哪个 MinEdgeLength 范围，再以对应的 Name 作为清晰度名称进行展示。

若您需要快速体验生成播放器签名，可以使用腾讯云点播控制台的 播放器签名生成工具。

TCPlayer 快直播降级说明

最近更新时间：2024-05-13 17:49:25

降级场景

快直播基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持。
目前，SDK 对以下操作系统和浏览器进行了测试，测试结果如下：

操作系统	操作系统版本	浏览器类型	浏览器版本	是否支持拉流
Windows	win 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		微信内嵌	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		微信内嵌	X5 内核	✓
		微信内嵌	XWeb 内核	✓

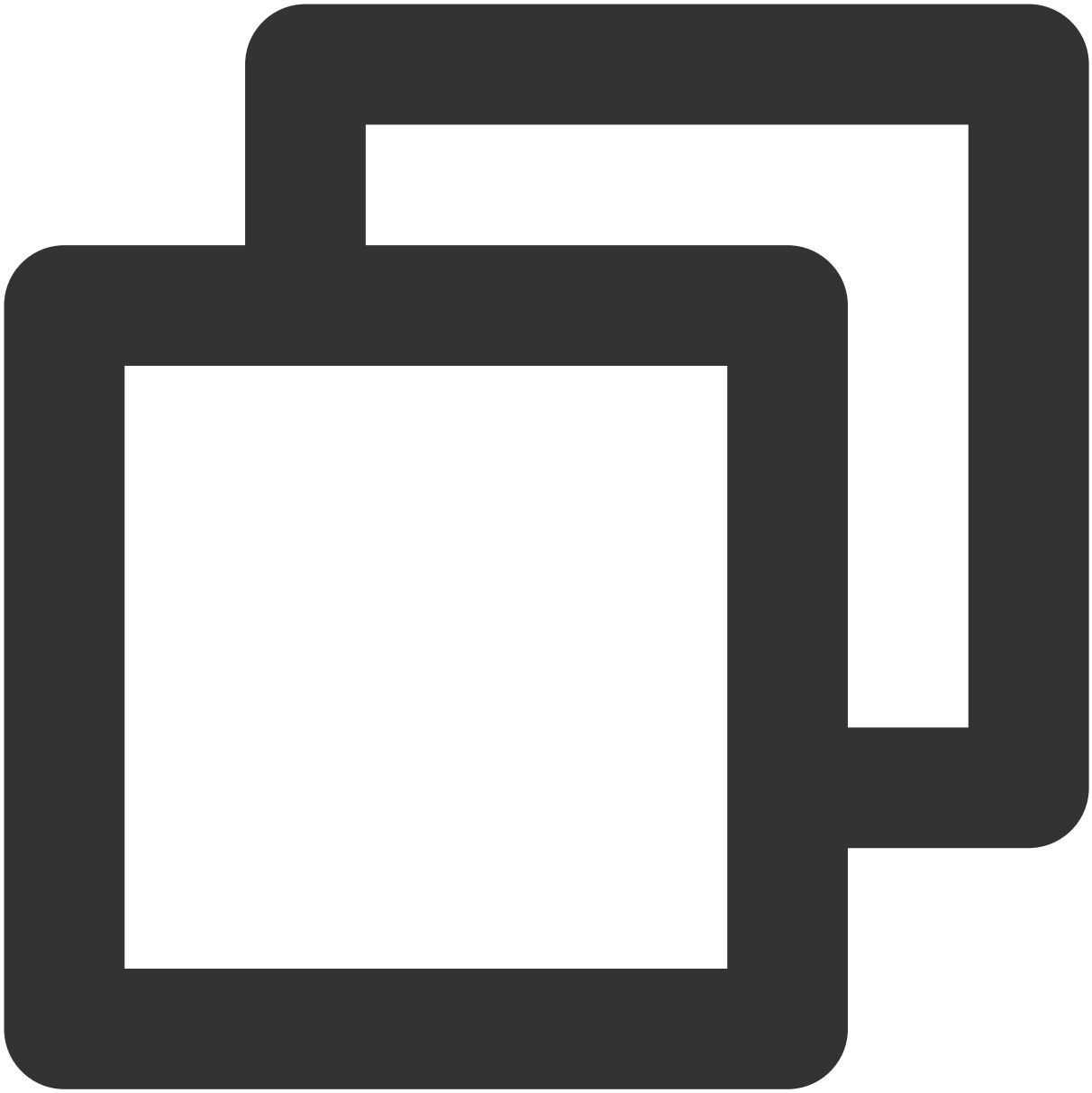
此外，在部分支持 WebRTC 的浏览器，也会出现解码失败或者服务端问题，这些情况下，播放器都会将 WebRTC 地址转换为兼容性较好的 HLS 地址来播放，这个行为称为降级处理。

总结一下，会触发降级的场景有以下几个：
浏览器环境不支持 WebRTC 。
连接服务器失败，并且连接重试次数已超过设定值 （内部状态码 -2004）。
播放过程解码失败（内部状态码 -2005）。
其他 WebRTC 相关错误（内部状态码 -2001）。

降级方式

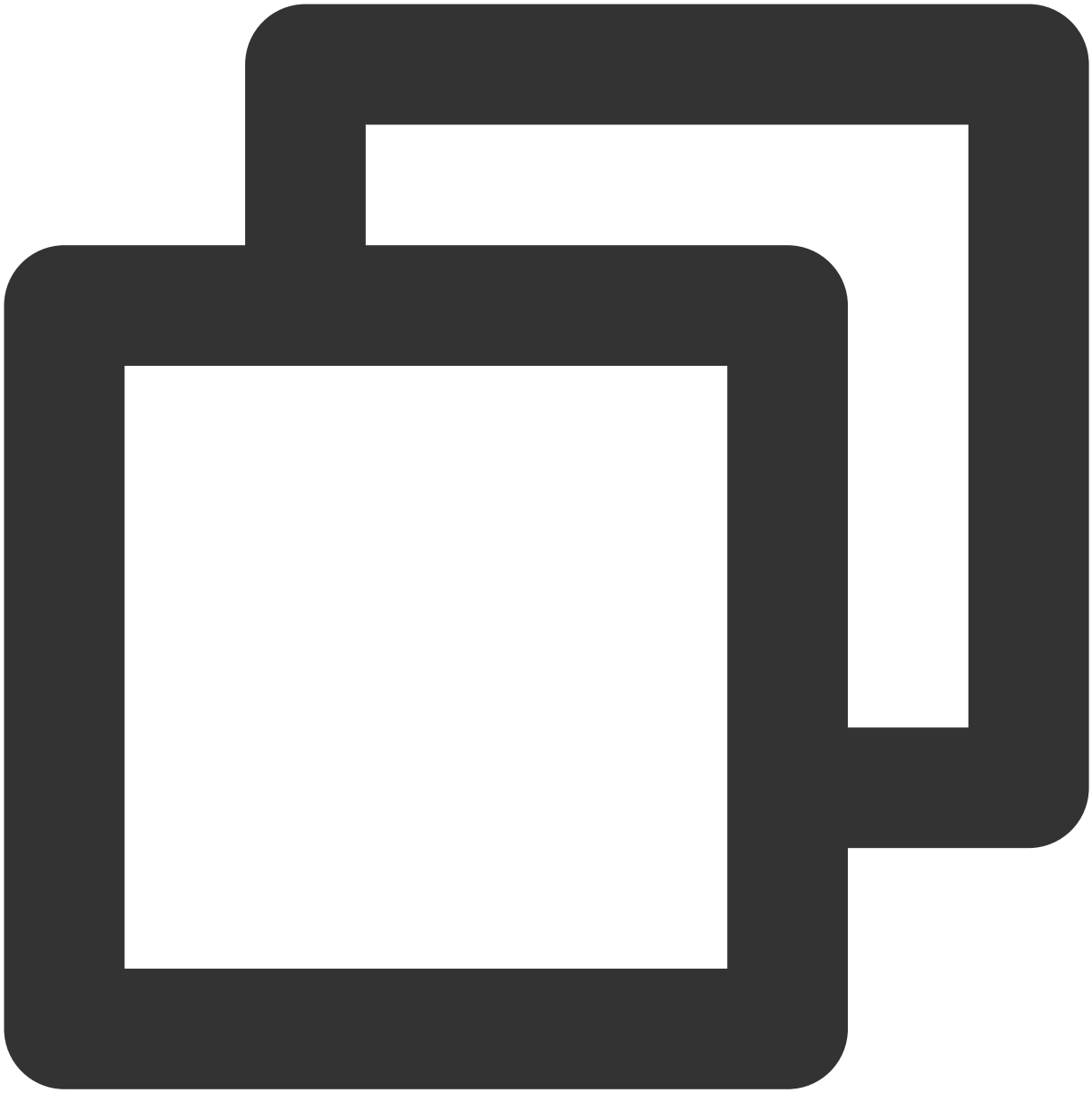
1. 自动降级

初始化播放器时，通过 sources 字段传入了快直播地址，在需要降级处理的环境，播放器自动会进行协议的转换，将快直播地址转换为 HLS 协议地址。
例如，快直播地址：



```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b284137ed10d
```

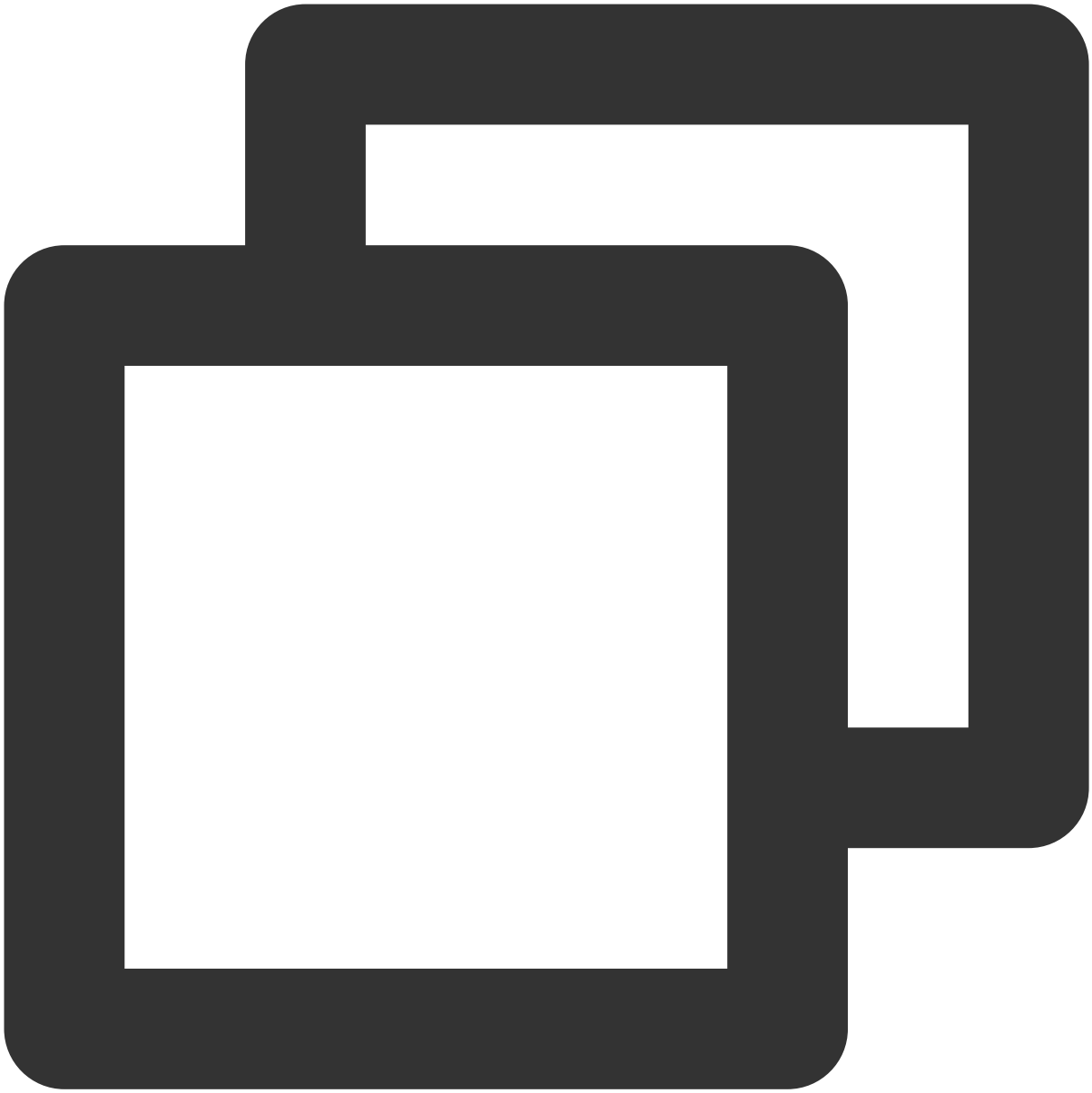
会自动转换为：



```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?txSecret=f22a813b284137e
```

2. 指定降级

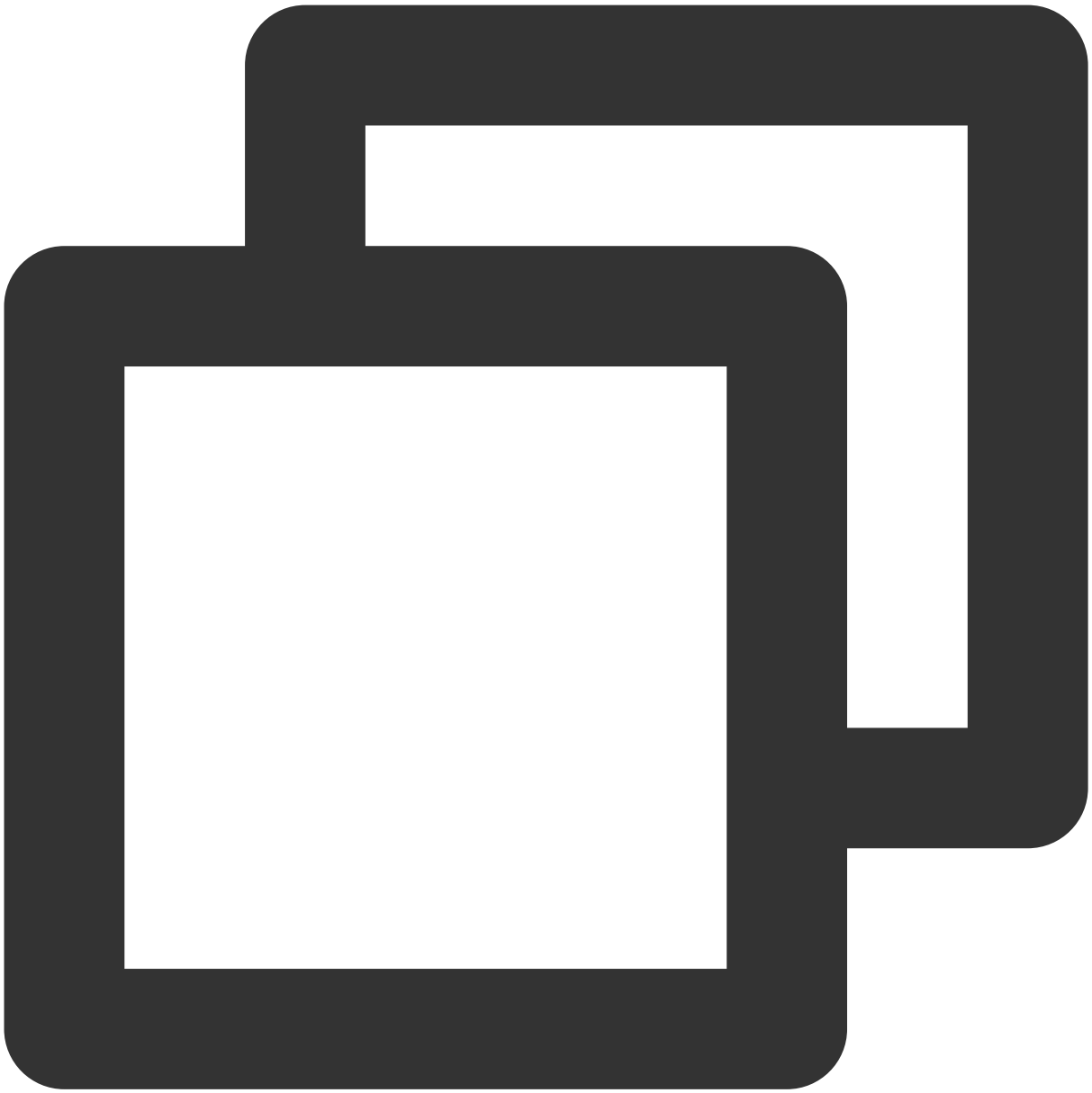
在播放自适应码率（ABR）场景，如果需要降级，并不能直接通过格式转换得到自适应码率的 HLS 地址，需要手动指定。又或者是在用户希望手动指定的其他场景，都可以通过如下方式指定降级地址，这里的地址并不局限于 HLS 协议，也可以是其他协议地址：



```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a8
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3
    },
});
```

降级回调

当触发降级时，播放器会触发回调：



```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

iOS 接入指引

最近更新时间：2024-04-18 17:06:53

产品概述

腾讯云 iOS 播放器组件是腾讯云开源的一款播放器组件，简单几行代码即可拥有类似腾讯视频强大的播放功能，包括横竖屏切换、清晰度选择、手势和小窗等基础功能，还支持视频缓存，软硬解切换和倍速播放等特殊功能，相比系统播放器，支持格式更多，兼容性更好，功能更强大，同时还具备首屏秒开、低延迟的优点，以及视频缩略图等高级能力。
若播放器组件满足不了您的业务的个性化需求，且您具有一定的开发经验，可以集成 播放器 SDK，自定义开发播放器界面和播放功能。

准备工作

1. 开通 云点播 相关服务，未注册用户可注册账号 试用。
2. 下载 Xcode，如您已下载可略过该步骤，您可以进入 App Store 下载安装。
3. 下载 Cocoapods，如您已下载可略过该步骤，您可以进入 Cocoapods官网 按照指引进行安装。

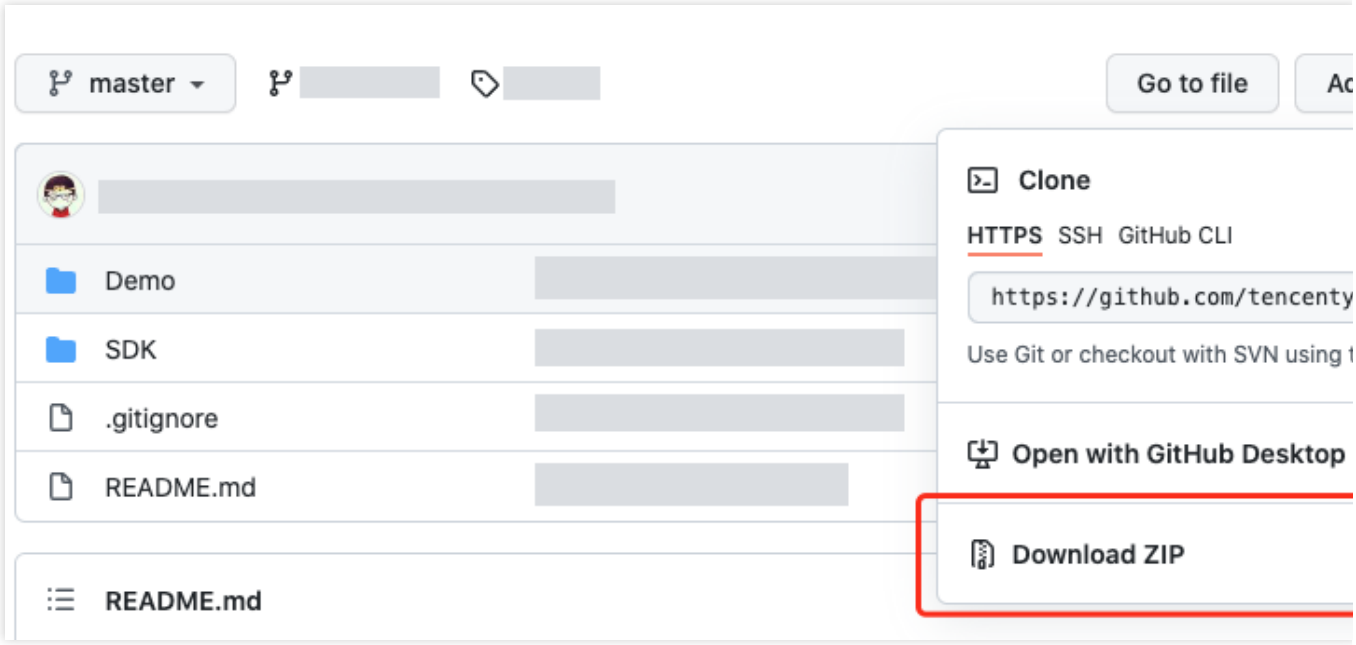
通过本文您可以学会

如何集成腾讯云 iOS 播放器组件
如何创建和使用播放器

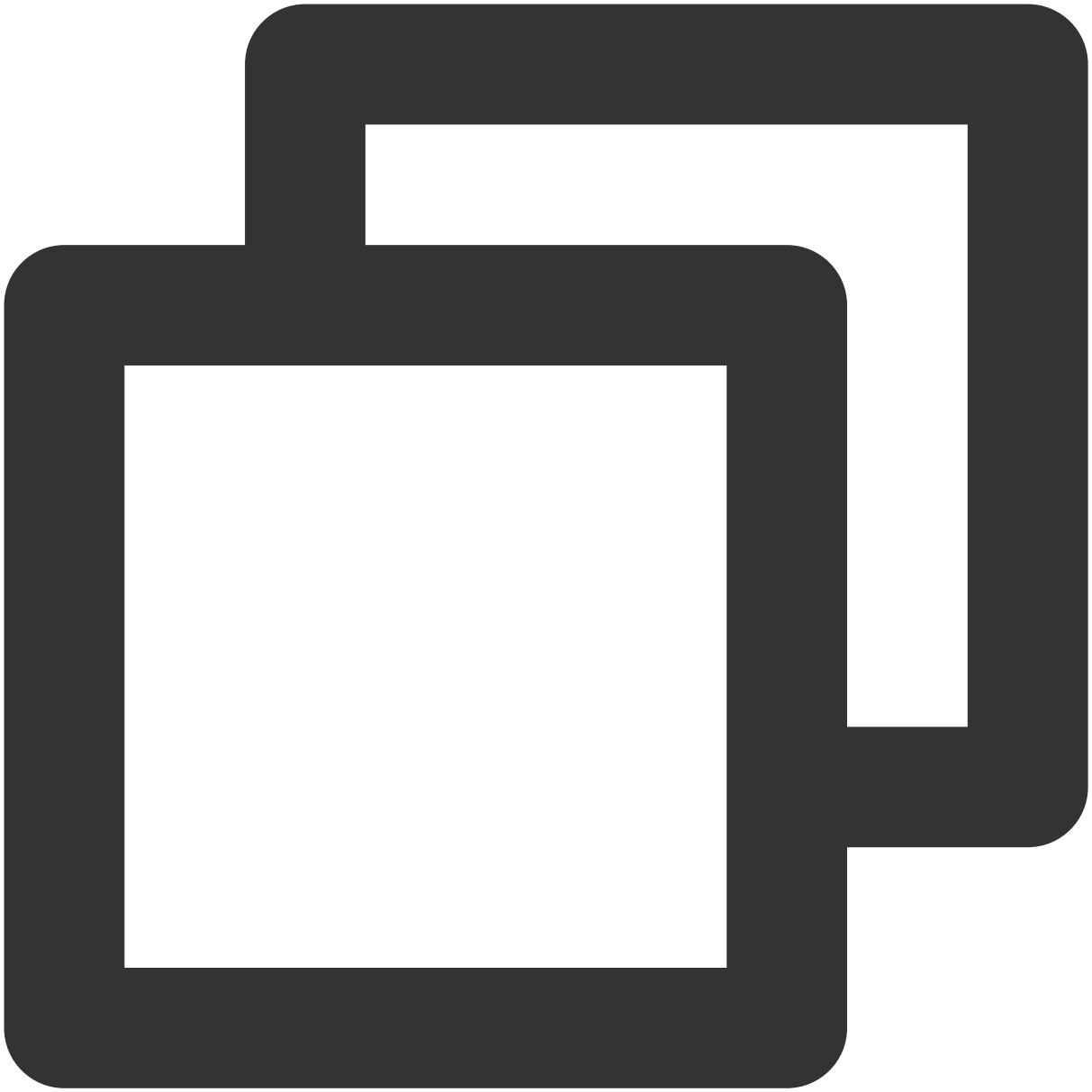
集成准备

步骤1：项目下载

腾讯云 iOS 播放器的项目地址是 LiteAVSDK/Player_iOS 。
您可通过 **下载播放器组件 ZIP 包** 或 **Git 命令下载** 的方式下载腾讯云 iOS 播放器组件项目工程。
下载播放器组件 ZIP 包
Git 命令下载
您可以直接下载播放器组件 ZIP 包，单击页面的 **Code** > **Download ZIP** 下载。

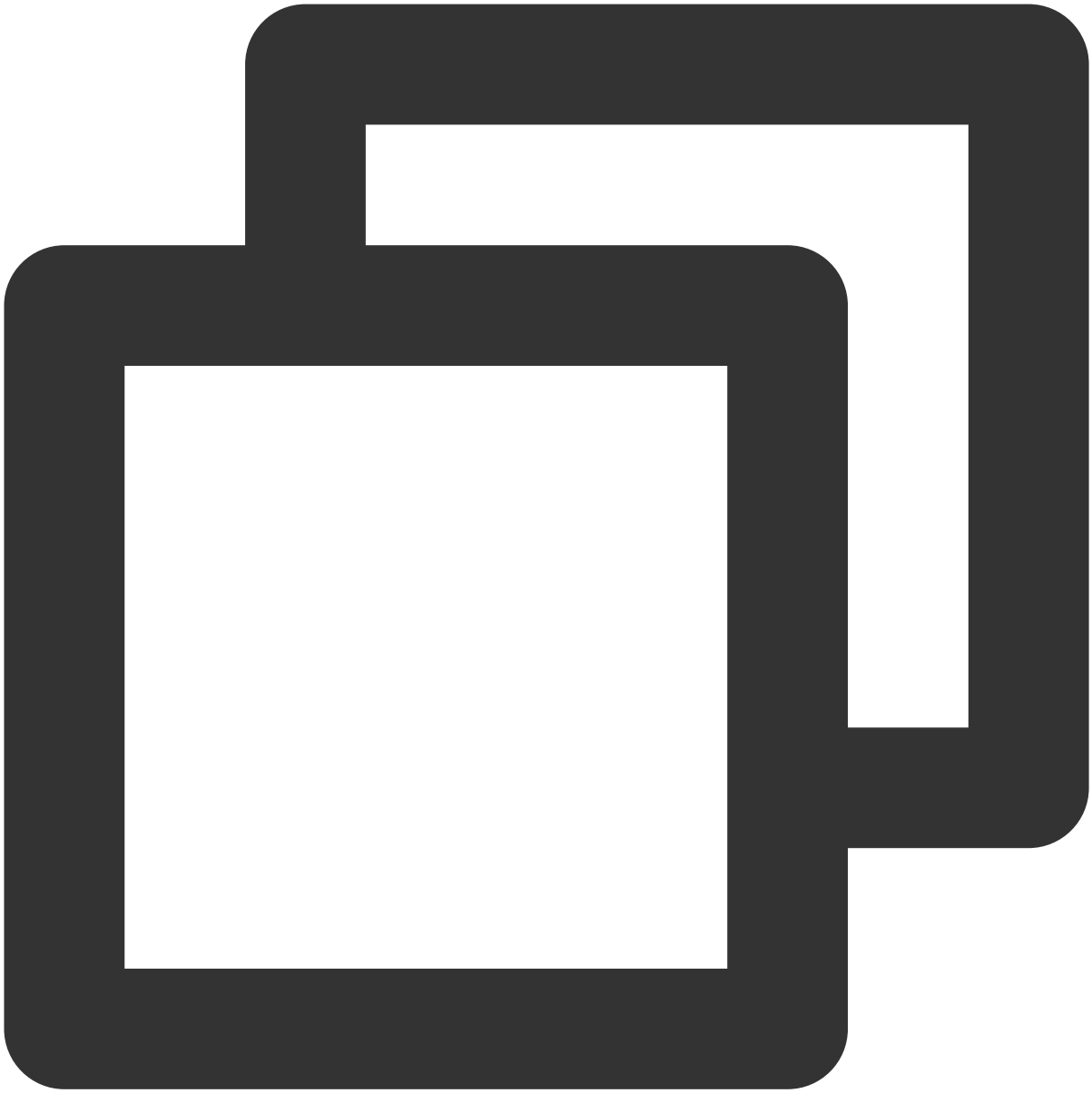


1. 首先确认您的电脑上安装了 Git。如果没有安装，可以参见 Git 安装教程 进行安装。
2. 执行下面的命令把播放器组件工程代码 clone 到本地。



```
git clone git@github.com:tencentyun/SuperPlayer_iOS.git
```

提示下面的信息表示成功 clone 工程代码到本地。



```
正克隆到 'SuperPlayer_iOS'...
remote: Enumerating objects: 2637, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (333/333), done.
remote: Total 2637 (delta 227), reused 524 (delta 170), pack-reused 1993
接收对象中: 100% (2637/2637), 571.20 MiB | 3.94 MiB/s, 完成.
处理 delta 中: 100% (1019/1019), 完成.
```

下载工程后，工程源码解压后的目录如下：

文件名	作用

SDK	存放播放器的 framework 静态库
Demo	存放超级播放器 Demo
App	程序入口界面
SuperPlayerDemo	超级播放器 Demo
SuperPlayerKit	超级播放器组件

步骤2：集成指引

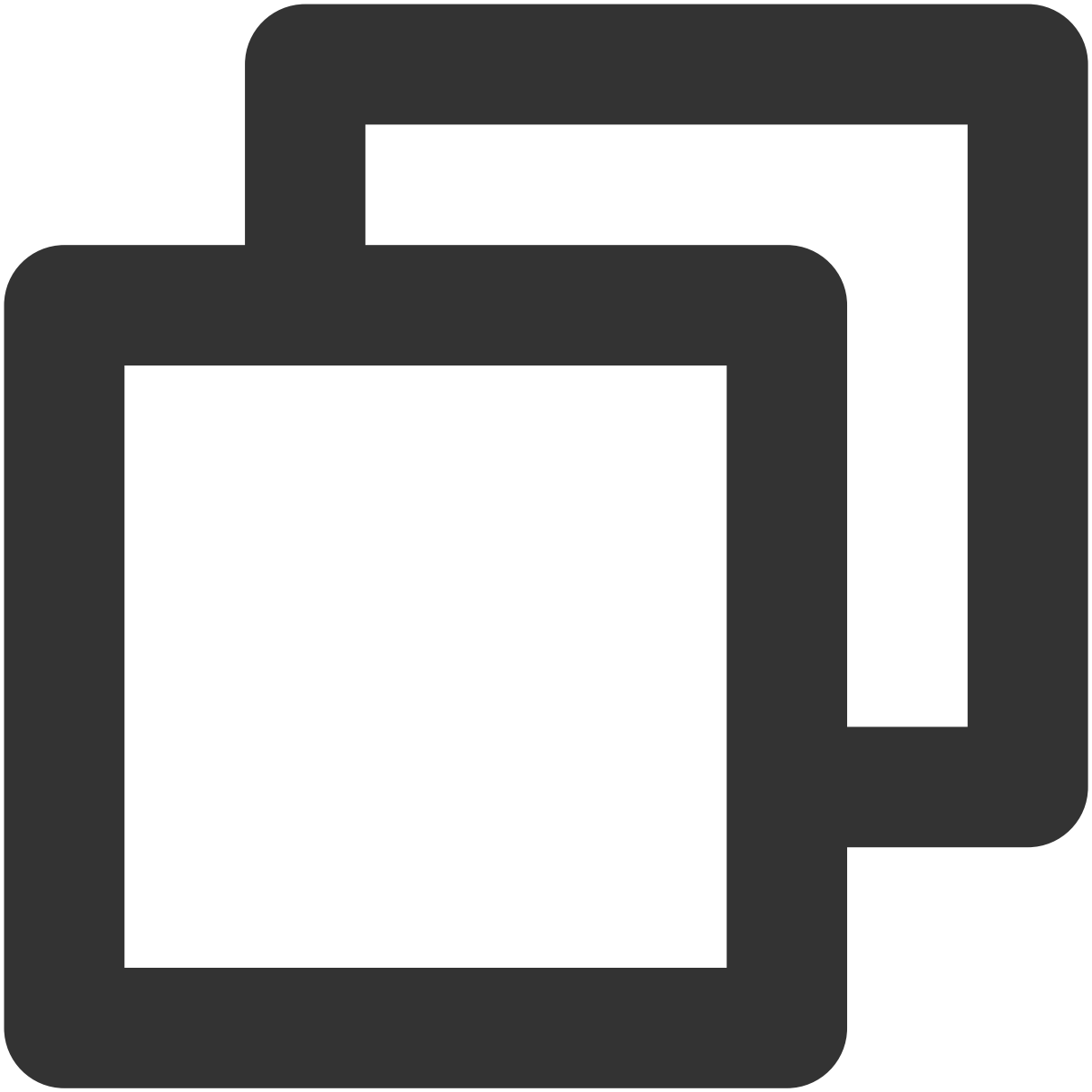
本步骤，用于指导用户如何集成播放器，推荐用户选择使用 **Cocoapods 集成** 或者 **手动下载 SDK** 再将其导入到您当前的工程项目中。

Cocoapods 集成

手动下载 SDK

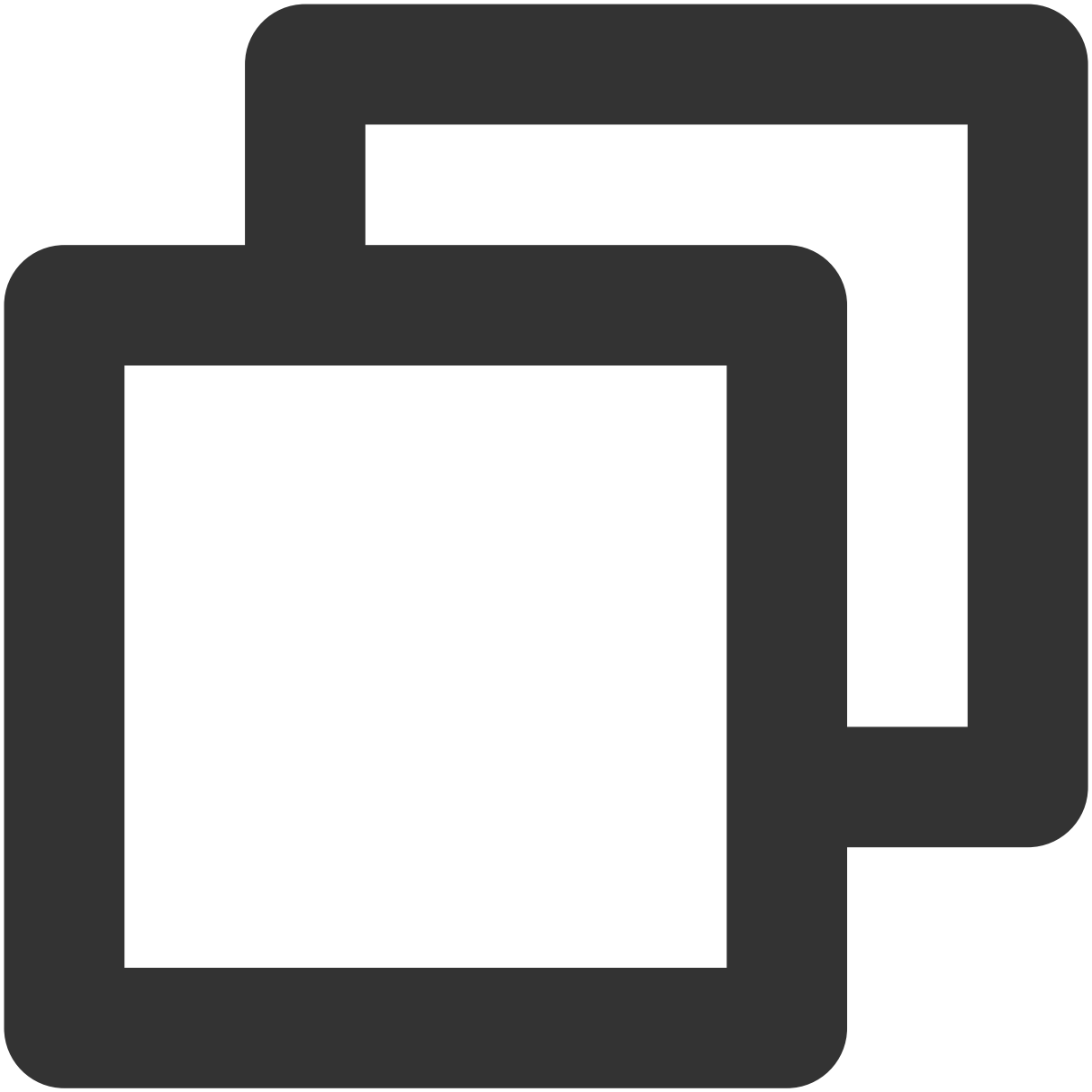
1. 本项目支持 Cocoapods 安装，只需要将如下代码添加到 Podfile 中：

Pod 方式直接集成 SuperPlayer。



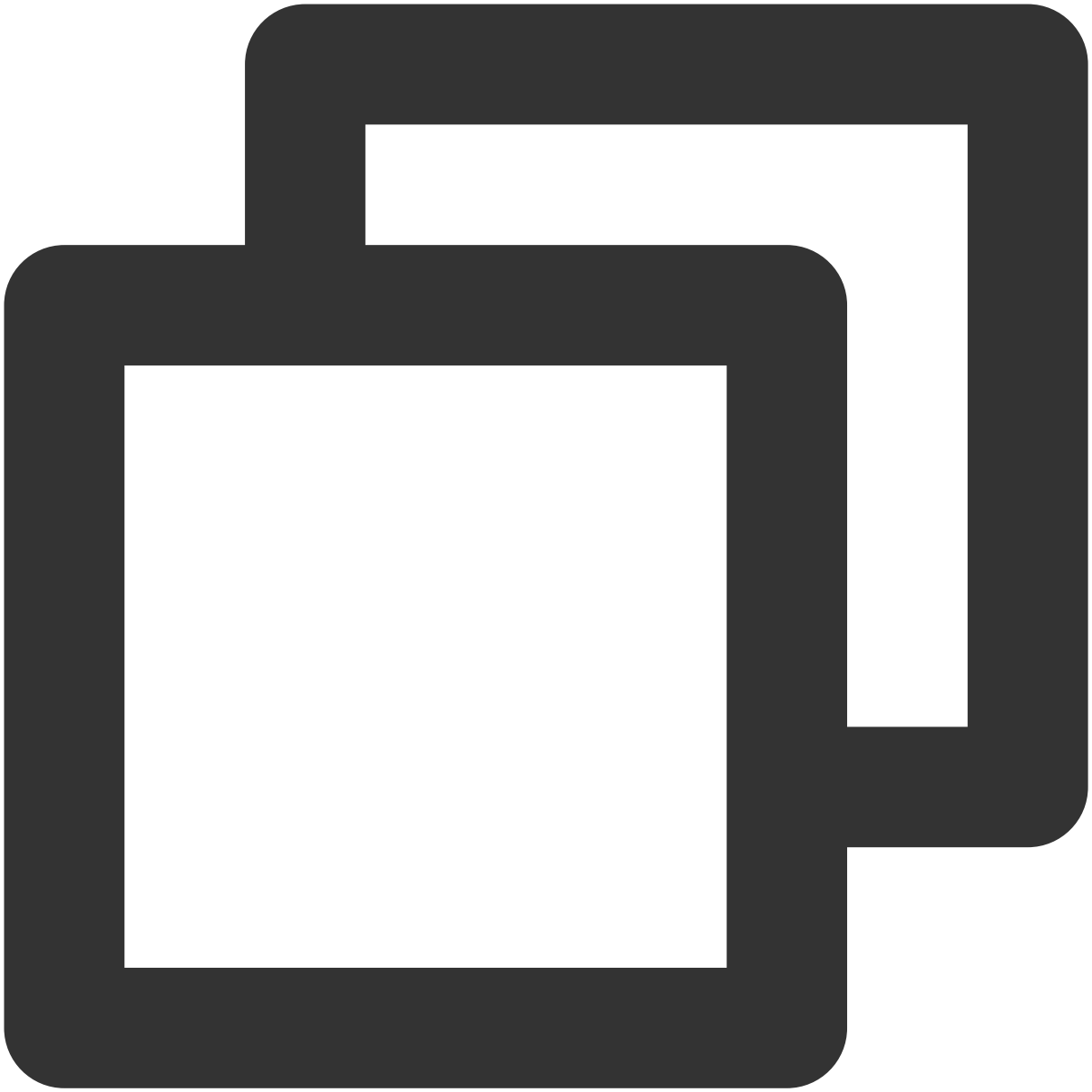
```
pod 'SuperPlayer
```

如果您需要依赖 Player 版，可以在 podfile 文件中添加如下依赖：



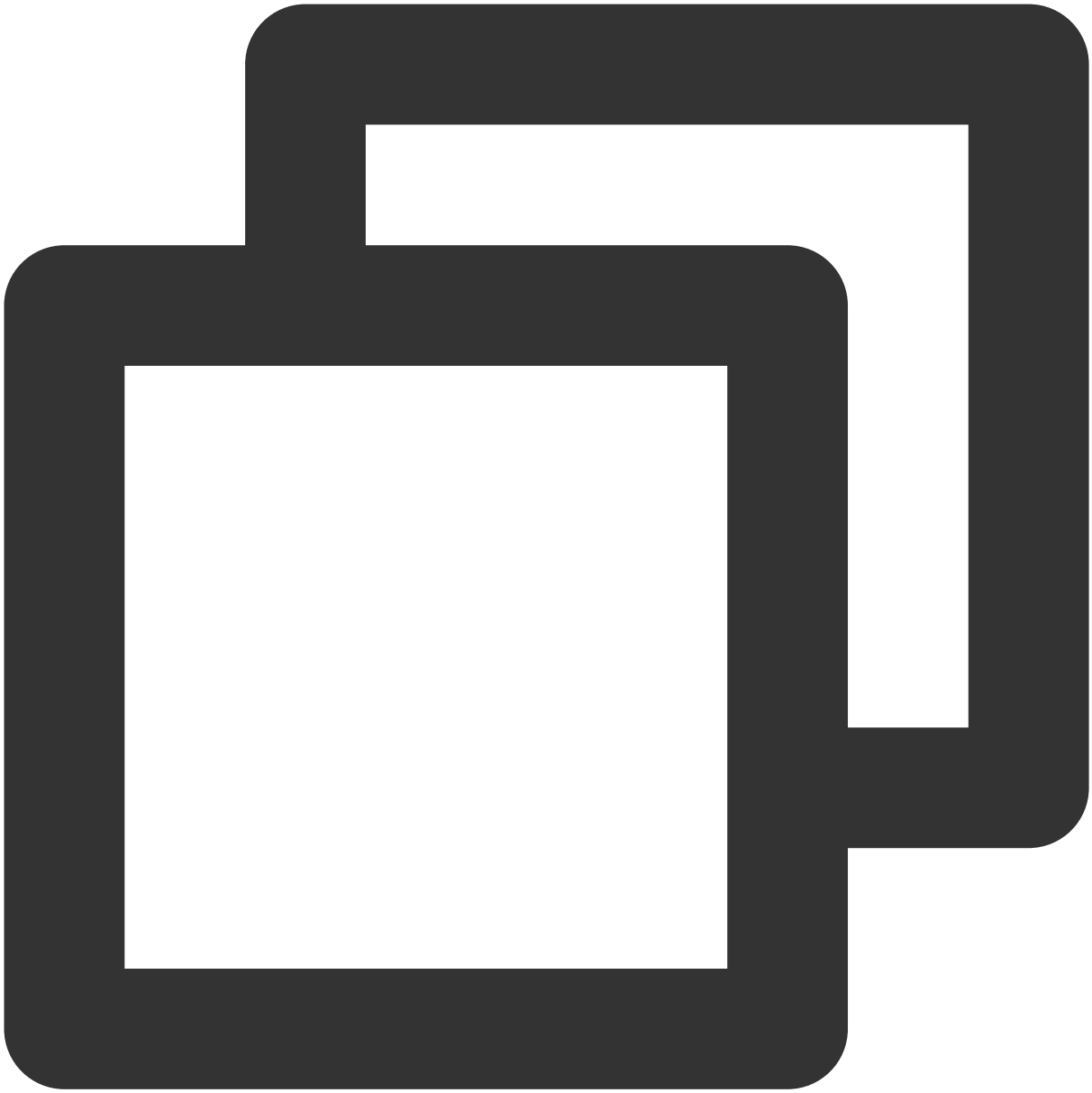
```
pod 'SuperPlayer/Player'
```

如果您需要依赖 Player_Premium 版，可以在 podfile 文件中添加如下依赖：



```
pod 'SuperPlayer/Player_Premium'
```

如果您需要依赖专业版，可以在 podfile 文件中添加如下依赖：



```
pod 'SuperPlayer/Professional'
```

2. 执行 `pod install` 或 `pod update` 。

1. 下载 SDK + Demo 开发包，腾讯云视立方 iOS 播放器项目为 LiteAVSDK/Player_iOS。

2. 导入 `TXLiteAVSDK_Player_Premium.framework` 到工程中，并勾选 `Do Not Embed` 。

3. 将 Demo/TXLiteAVDemo/SuperPlayerKit/SuperPlayer 拷贝到自己的工程目录下。

4. SuperPlayer依赖第三方库包括：AFNetworking、SDWebImage、Masonry、TXLiteAVSDK_Player。

如果是手动集成 TXLiteAVSDK_Player，需要添加所需要的系统库和 library：

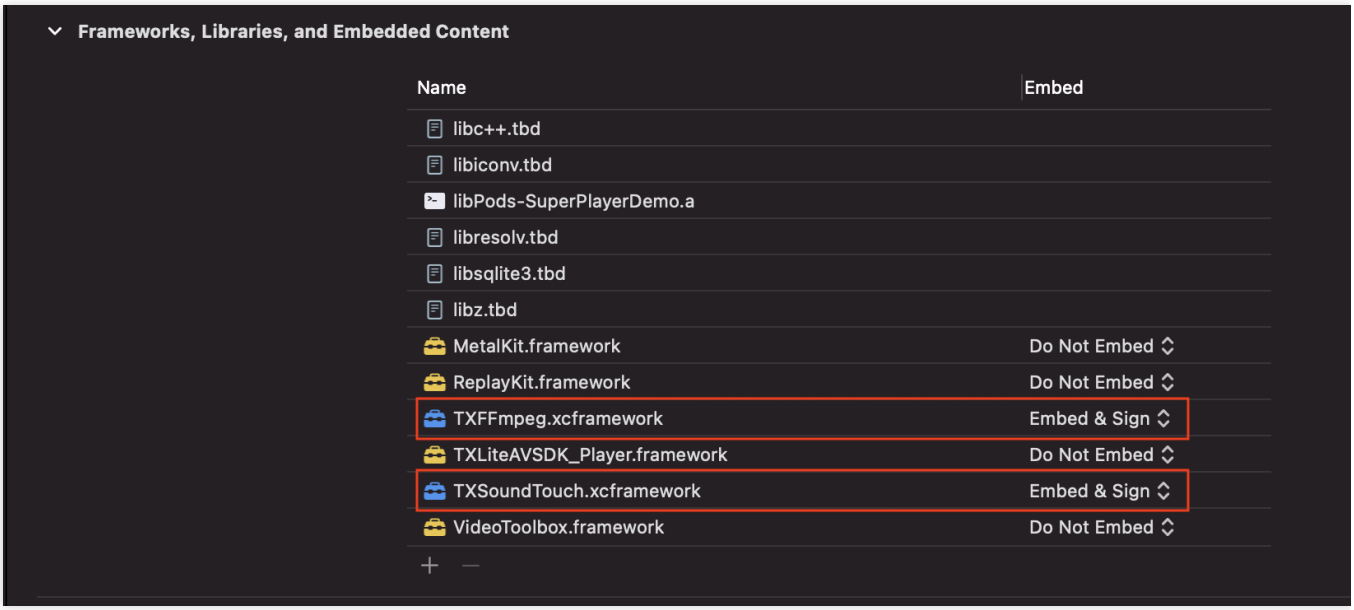
系统 Framework 库：MetalKit, ReplayKit, SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics,

AVFoundation, Accelerate, MobileCoreServices, ,VideoToolbox。

系统 Library 库: libz, libresolv， libiconv, libc++, libsqlite3。

具体操作步骤可以 参考：定制开发 - 点播场景 - 接入文档 - SDK集成 步骤1 - 手动集成 SDK。

此外还需要把 TXLiteAVSDK_Player 文件下的 TXFFmpeg.xcframework 和 TXSoundTouch.scframework 以动态库的方式加进来如下图所示：



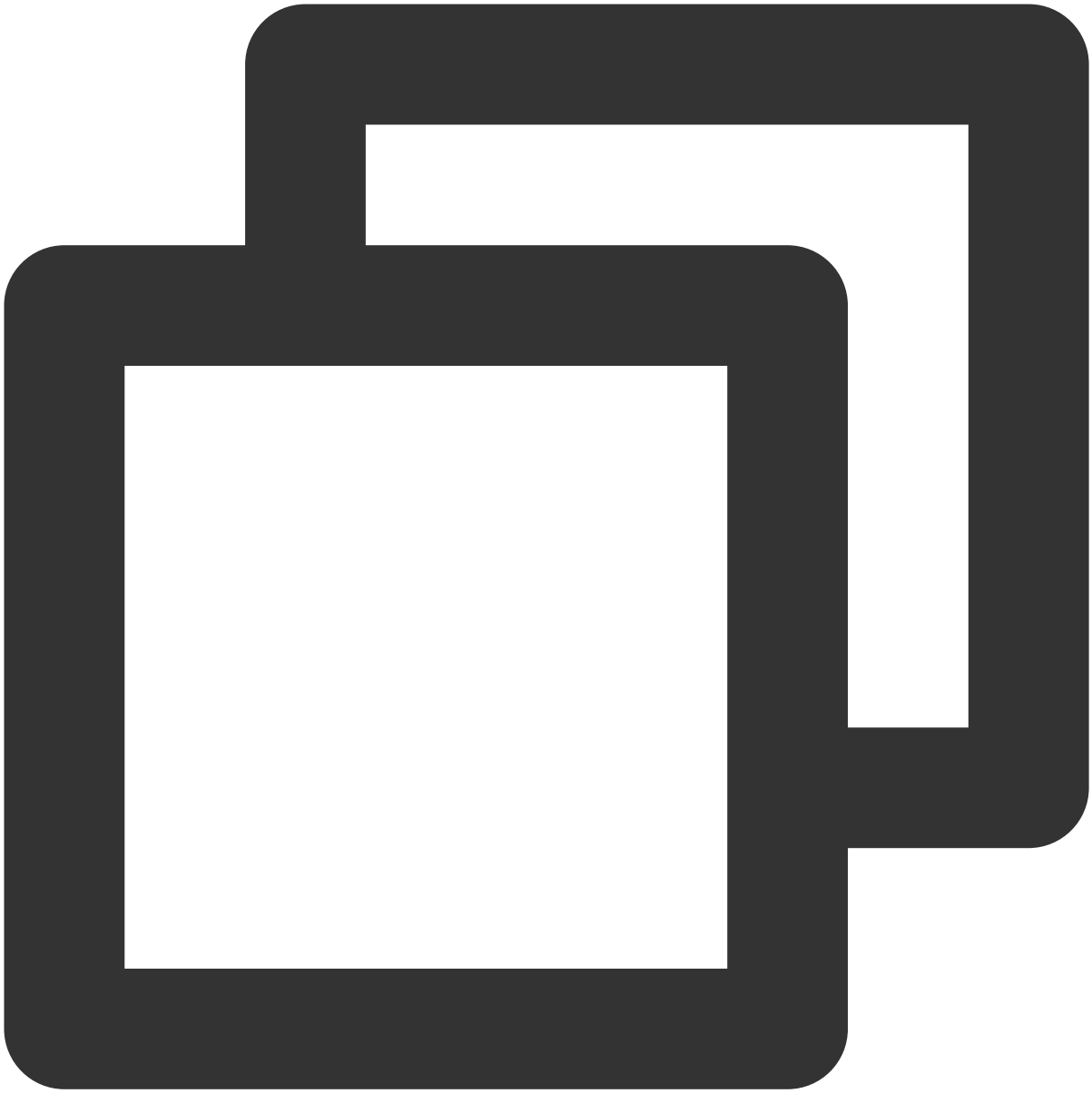
5. 如果是用 Pod 的方式集成 TXLiteAVSDK_Player，不需要添加任何库。

步骤3：使用播放器功能

本步骤，用于指导用户创建和使用播放器，并使用播放器进行视频播放。

1. **创建播放器：**

播放器主类为 `SuperPlayerView`，创建后即可播放视频。



```
// 引入头文件
#import <SuperPlayer/SuperPlayer.h>

// 创建播放器
_playerView = [[SuperPlayerView alloc] init];
// 设置代理，用于接受事件
_playerView.delegate = self;
// 设置父 View，_playerView 会被自动添加到 holderView 下面
_playerView.fatherView = self.holderView;
```

2. **配置 License 授权**

若您已获得相关 License 授权，需在 云点播控制台 获取 License URL 和 License Key。

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

3. **播放视频：**

本步骤，用于指导用户播放视频，腾讯云视立方 iOS 播放器组件支持 云点播 FileId 或者 使用 URL 进行播放，推荐您选择**集成 FileId** 使用更完善的能力。

云点播 FileId 播放

使用 URL 播放

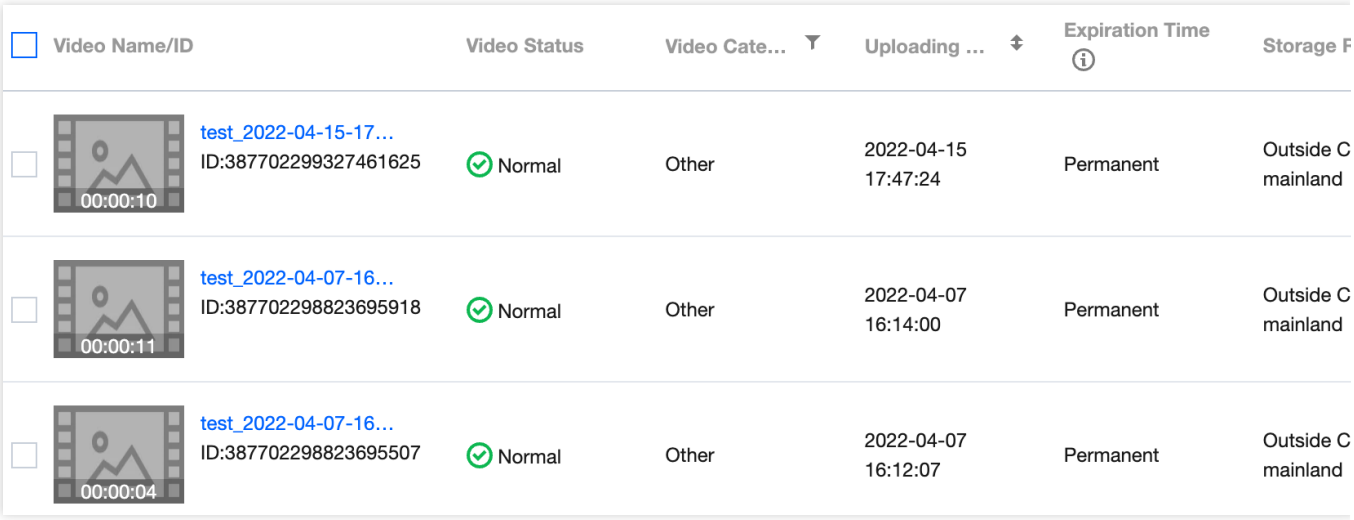
本地视频播放

视频 FileId 在一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 FileId 到客户端。

2. 服务端视频上传时，在 确认上传 的通知中包含对应的 FileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件，查看 FileId。如下图所示，ID 即表示 FileId：

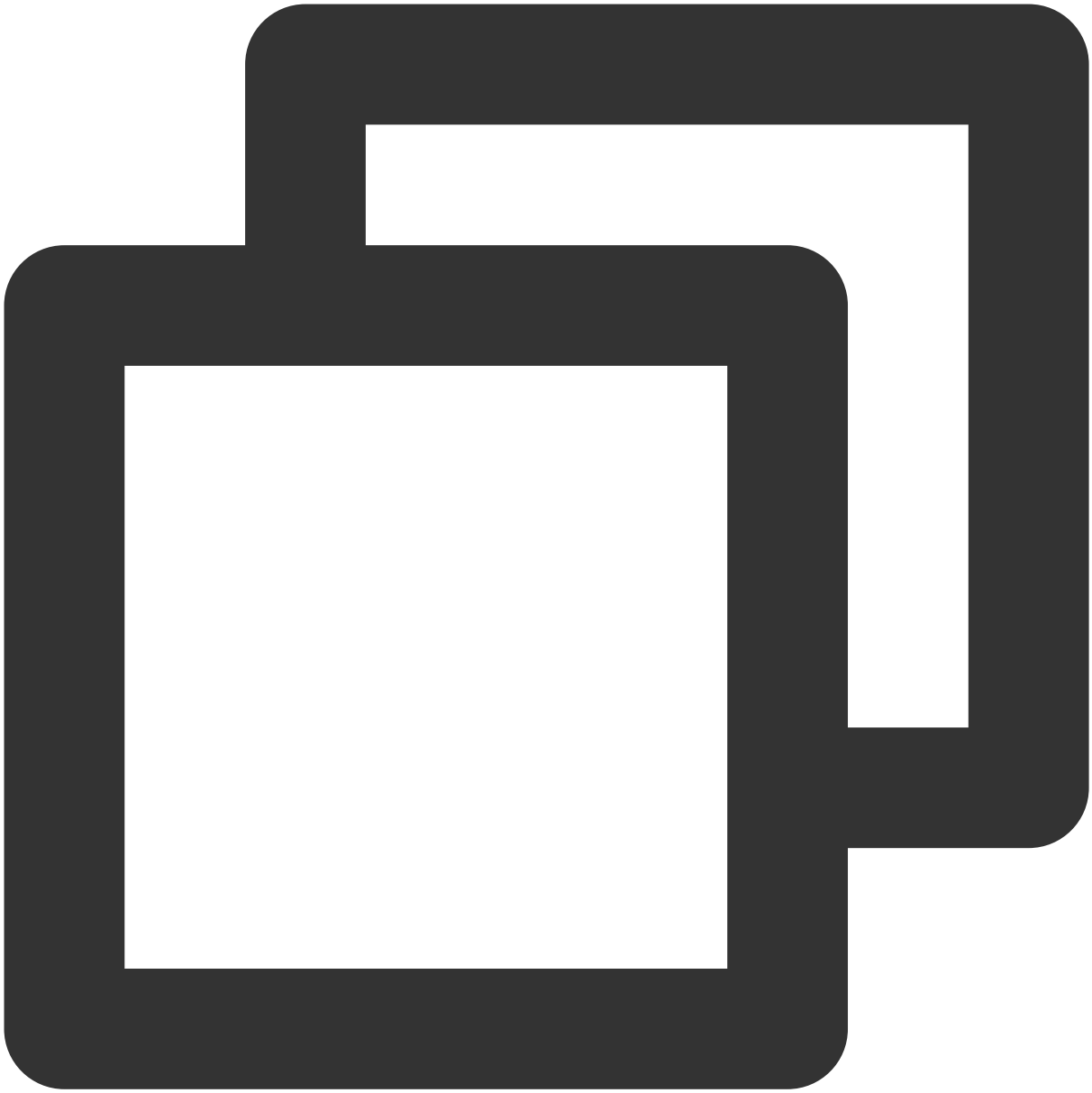


注意：

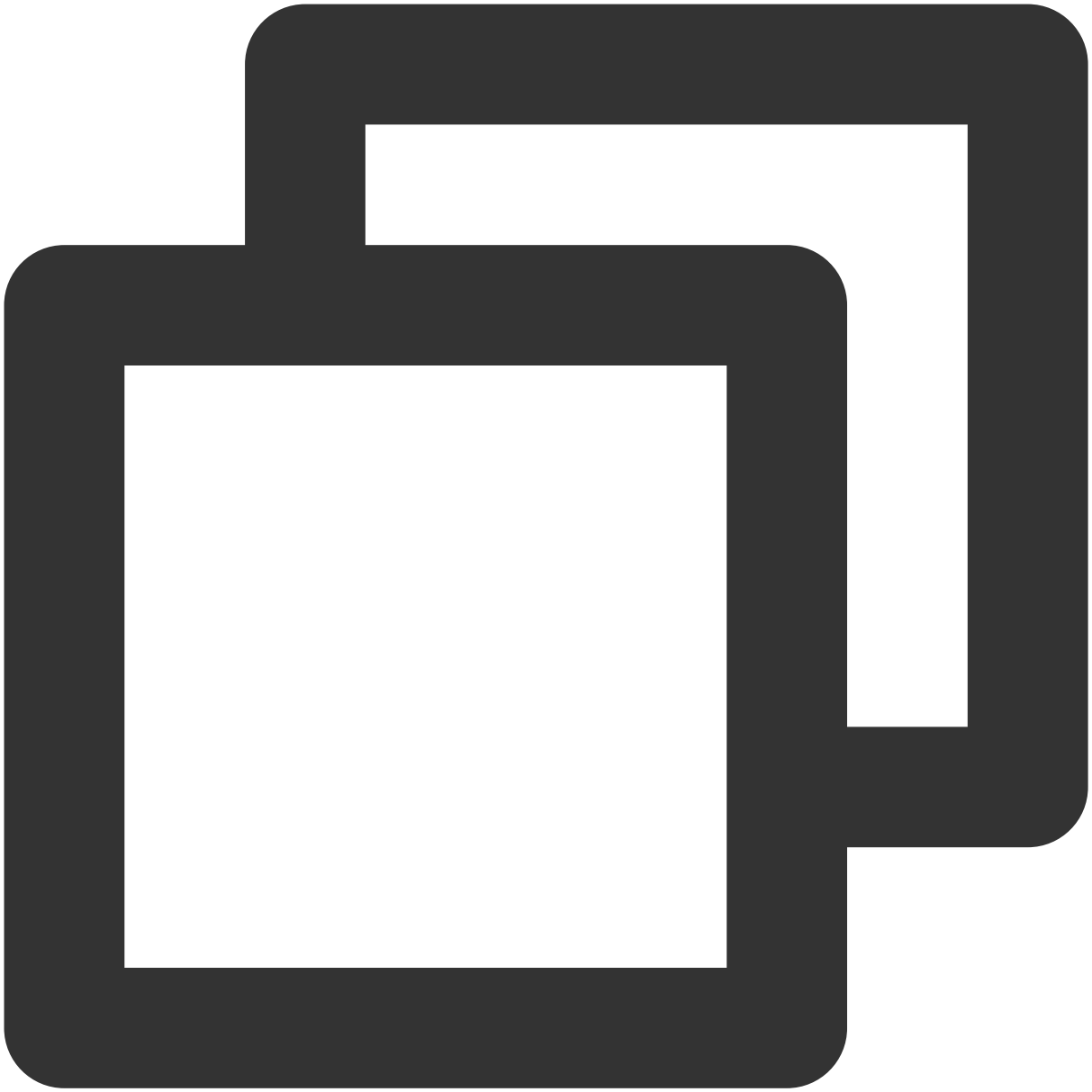
1. 通过 FileId 播放时，需要首先使用 Adaptive-HLS(10) 转码模板对视频进行转码，或者使用播放器组件签名 psign 指定播放的视频，否则可能导致视频播放失败。转码教程和说明可参见 用播放器组件播放视频，psign 生成教程可参见 psign 教程。

2. 若您在通过 FileId 播放时出现“no v4 play info”异常，则说明您可能存在上述问题，建议您根据上述教程调整。同时您也可以直接获取源视频播放链接，通过 URL 播放 的方式实现播放。

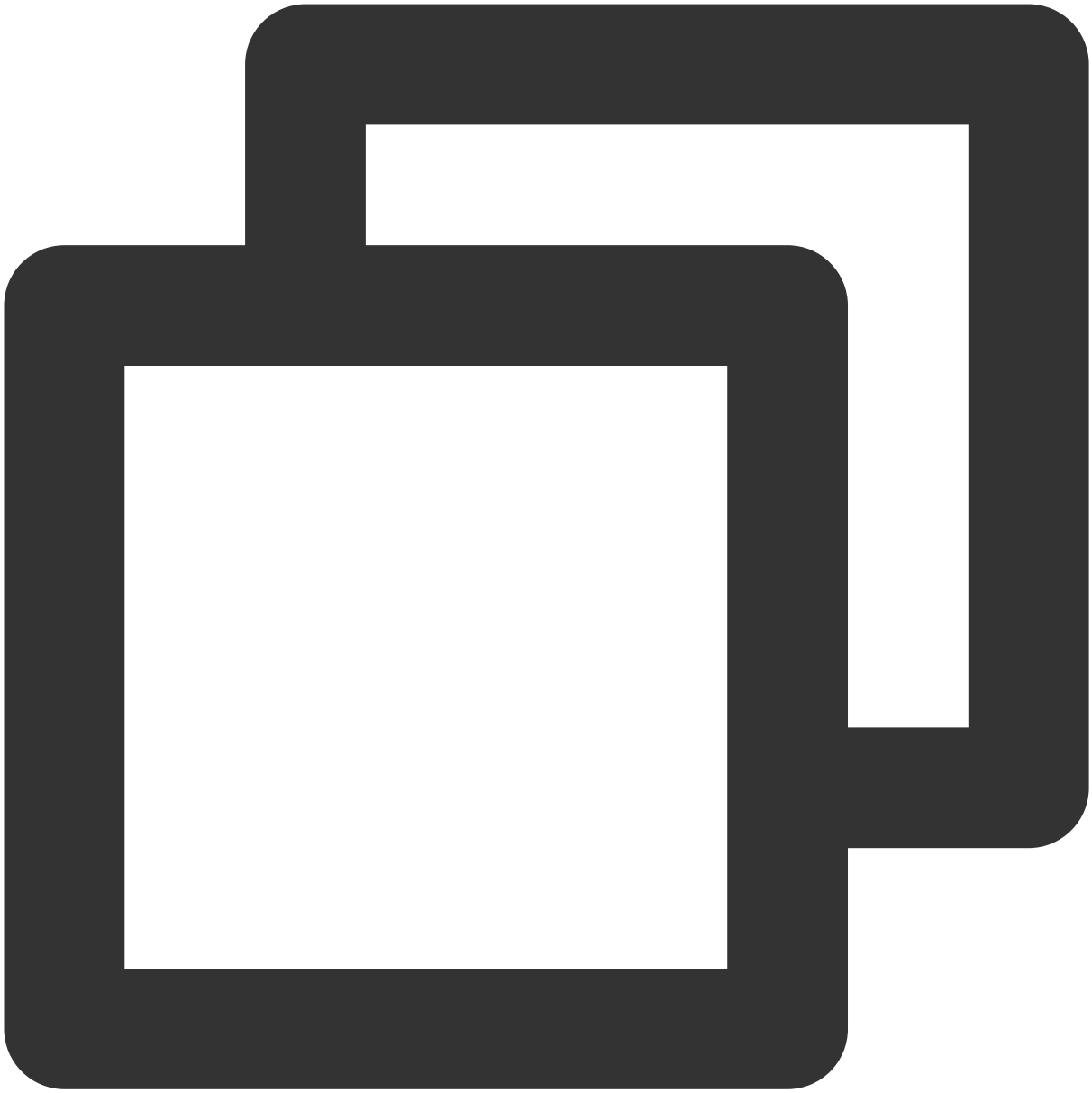
3. **未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放。**



```
//在未开启防盗链进行播放的过程中，如果出现了“no v4 play info”异常，建议您使用 Adaptive-HLS(

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// 配置 AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // 配置 FileId
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
model.videoId.pSign = @"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOT
[_playerView playWithModelNeedLicence:model];
```



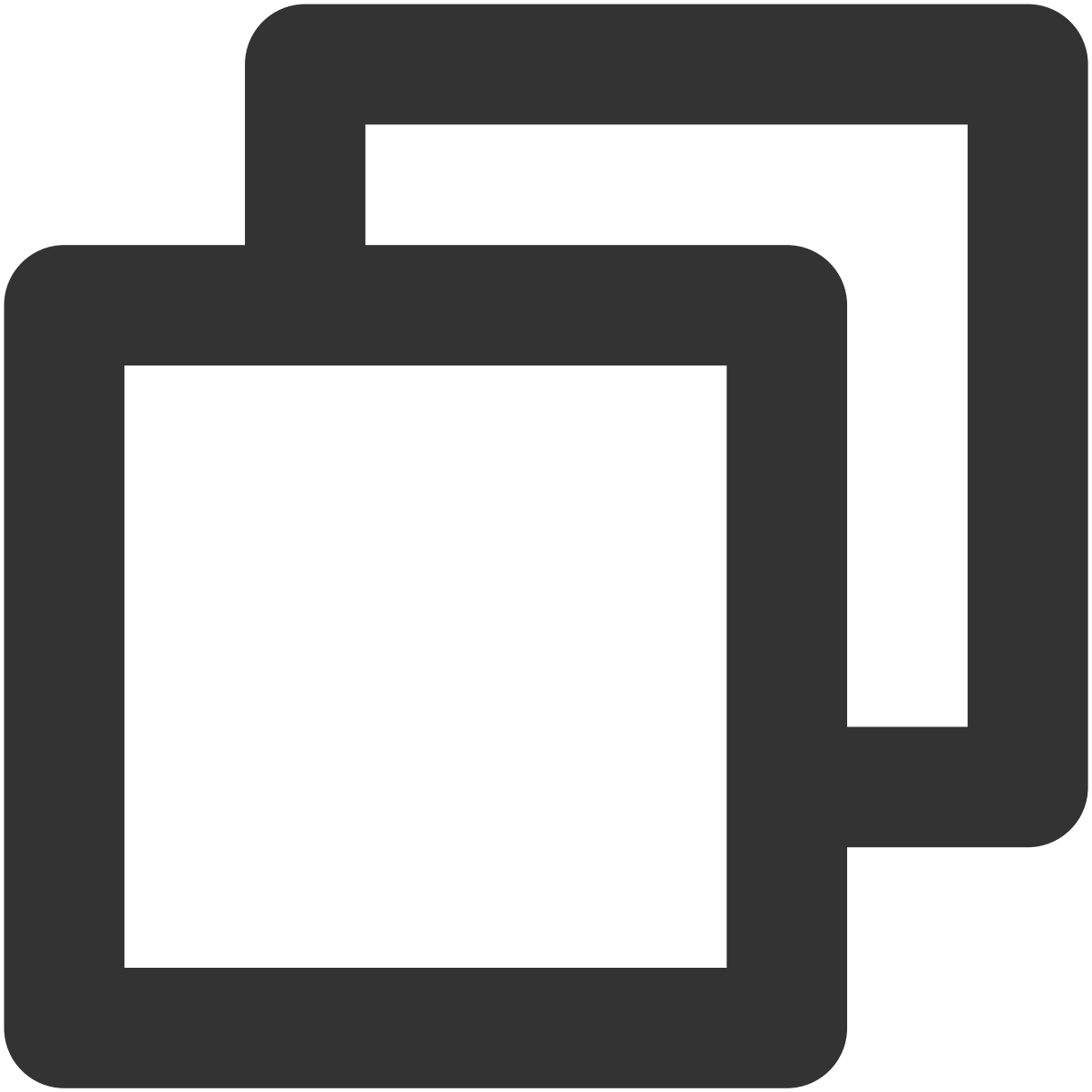
```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.videoURL = @"http://your_video_url.mp4";    // 配置您的播放视频 url
[_playerView playWithModelNeedLicence:model];
```



```
SuperPlayerModel *model = [[SuperPlayerModel alloc]init];
//把您的视频文件添加到工程，然后通过NSBundle获取视频的文件路径
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"your_video_name" ofTy
model.videoURL = [filePath stringByReplacingOccurrencesOfString:@"file://" withStri
[_playerView playWithModelNeedLicence:model];
```

4. **退出播放：**

当不需要播放器时，调用 `resetPlayer` 清理播放器内部状态，释放内存。



```
[_playerView resetPlayer];
```

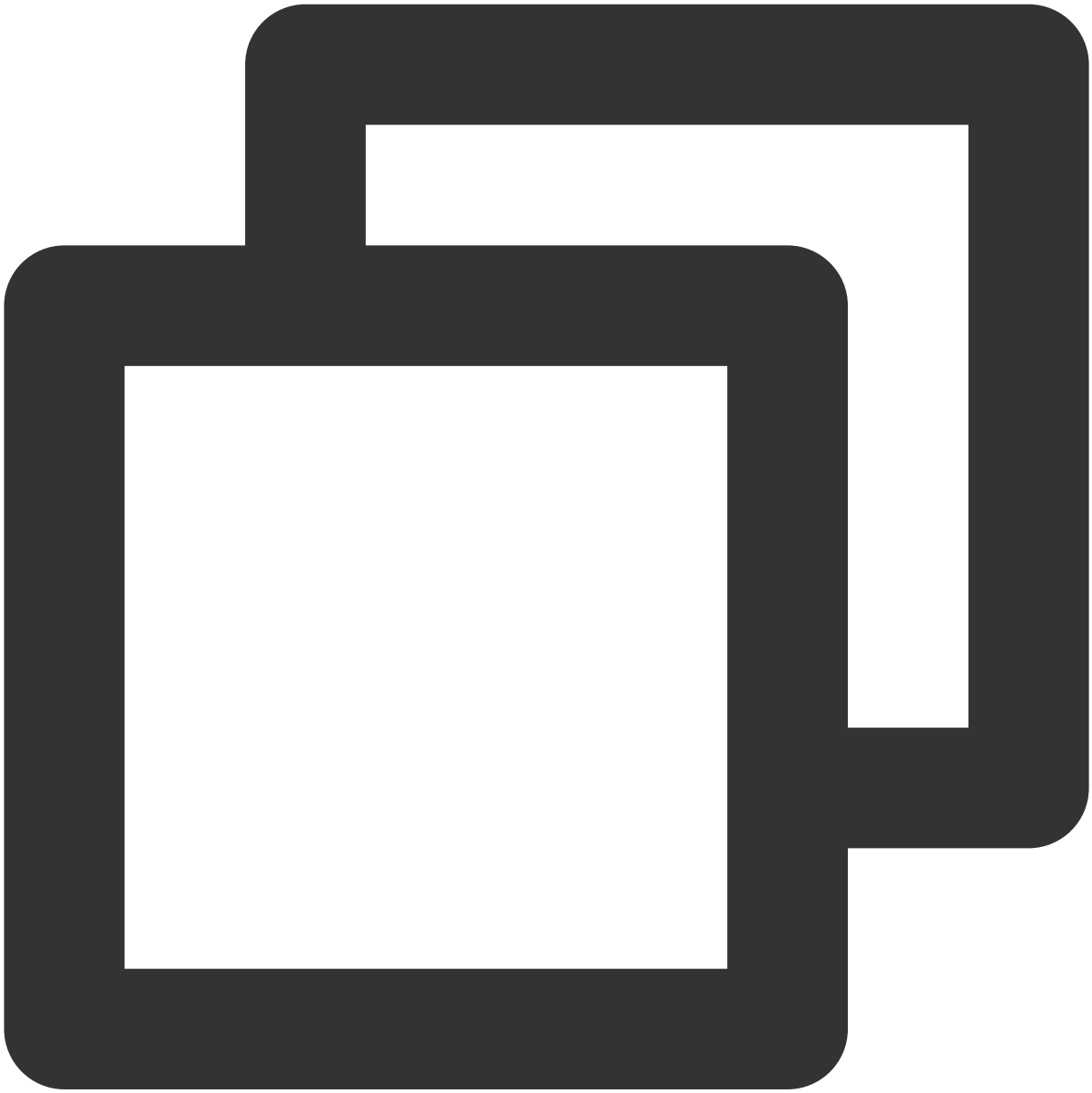
您已完成了腾讯云 iOS 播放器组件的创建、播放视频和退出播放的能力集成。

功能使用

1、全屏播放

播放器组件支持全屏播放，在全屏播放场景内，同时支持锁屏、手势控制音量和亮度、弹幕、截屏、清晰度切换等功能设置。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件** 中体验，单击界面右下角**全屏**即可进入全屏播放界面。

在窗口播放模式下，可通过调用下述接口进入全屏播放模式：



```
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player {
  //用户可在此自定义切换全屏后的逻辑
}
```

全屏播放界面功能介绍

返回窗口模式

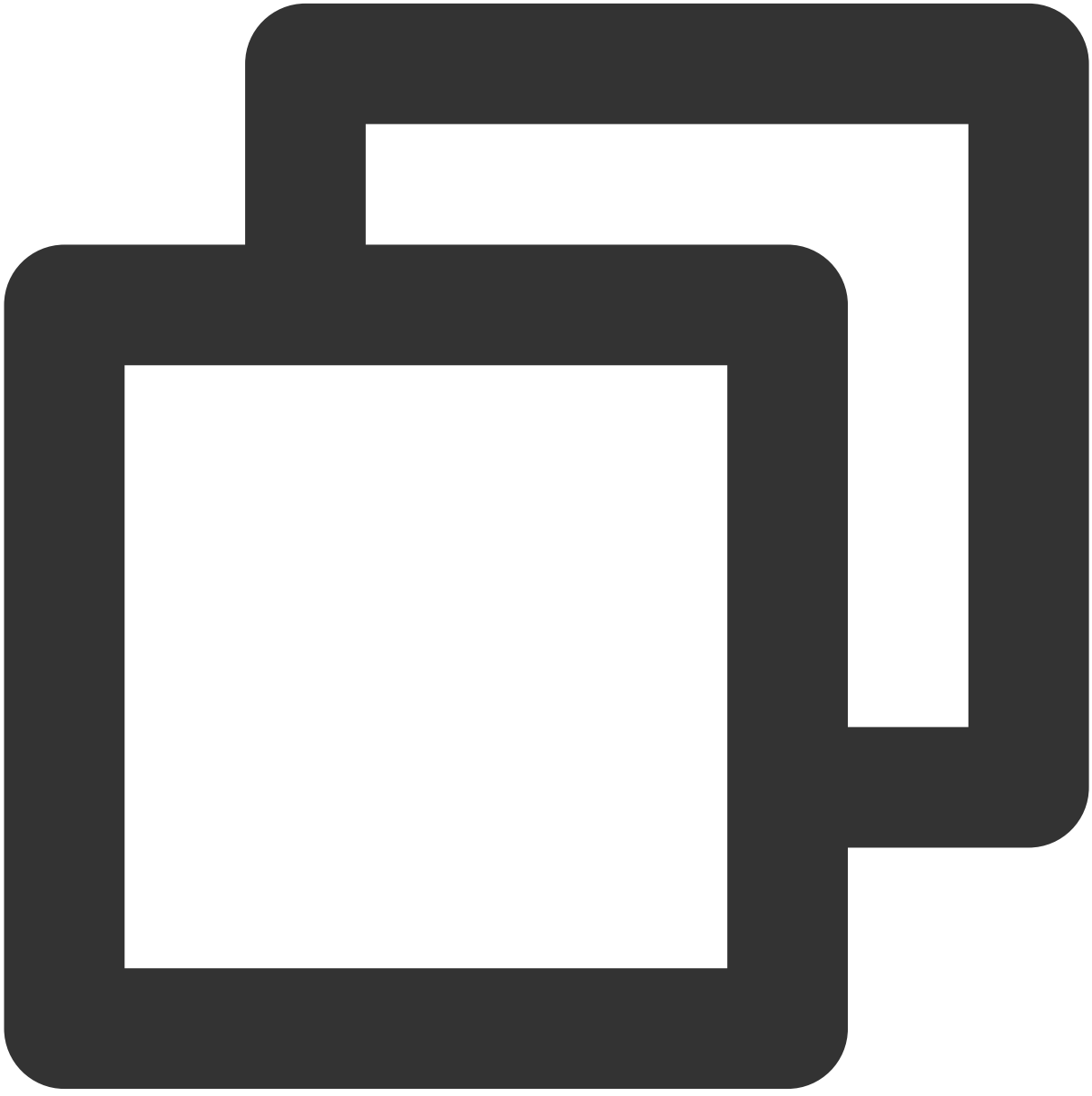
锁屏

弹幕

截屏

清晰度切换

通过**返回**即可返回窗口播放模式，单击后 SDK 处理完全屏切换的逻辑后会触发的代理方法为：

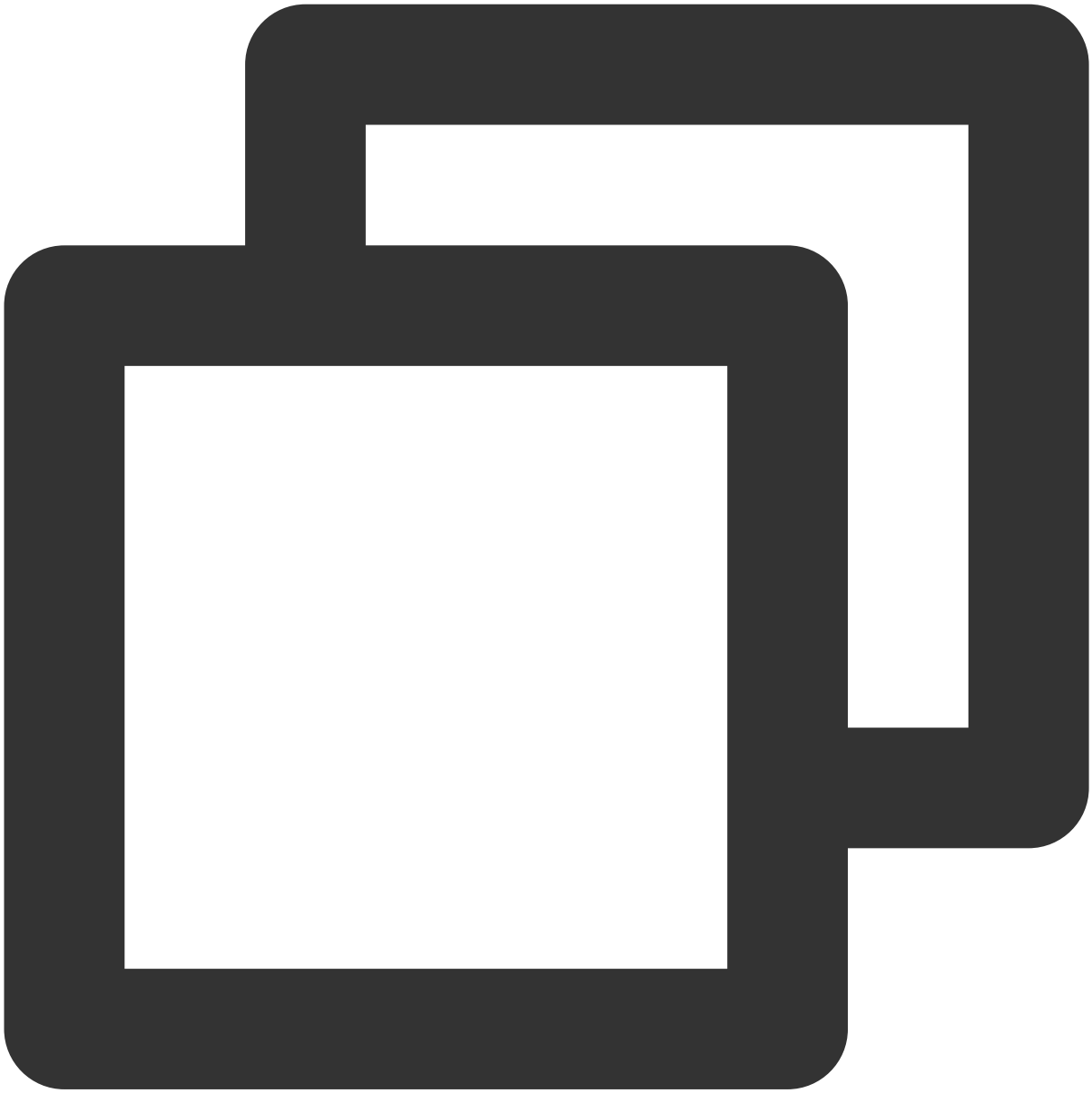


```
// 返回事件
- (void)superPlayerBackAction:(SuperPlayerView *)player;
```

单击左上角返回按钮触发

```
// 全屏改变通知
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player;
```

锁屏操作可以让用户进入沉浸式播放状态。单击后由 SDK 自己处理，无回调。

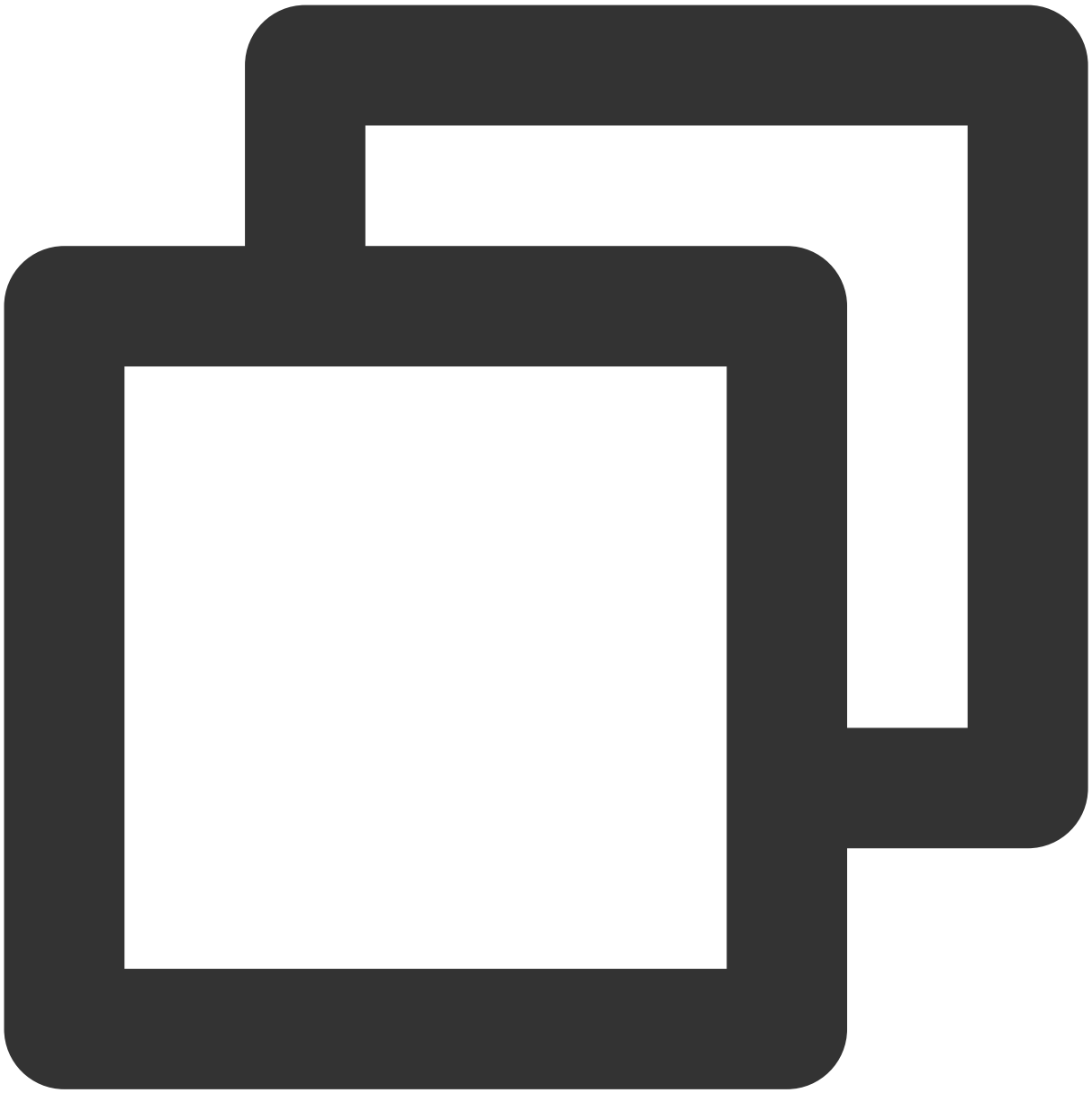


```
// 用户可通过以下接口控制是否锁屏
@property(nonatomic, assign) BOOL isLockScreen;
```

打开弹幕功能后屏幕上会有用户发送的文字飘过。

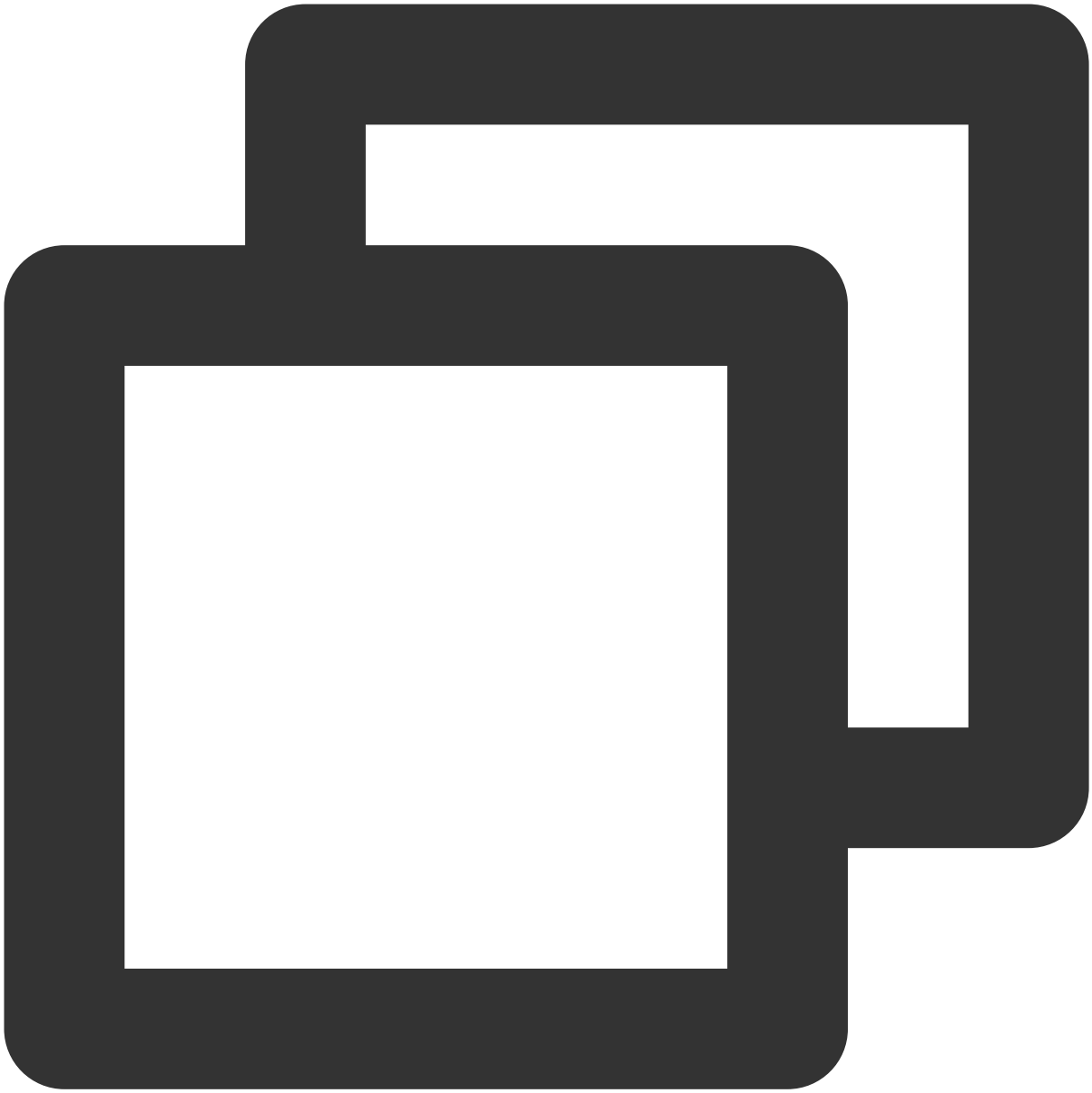
在这里拿到 SPDefaultControlView 对象，播放器 view 初始化的时候去给 SPDefaultControlView 的弹幕按钮设置事件，弹幕内容和弹幕 view 需要用户自己自定义，详细参见 SuperPlayerDemo 下的 CFDanmakuView、

CFDanmakuInfo、CFDanmaku。



```
SPDefaultControlView *dv = (SPDefaultControlView *)**self**.playerView.controlView;
[dv.danmakuBtn addTarget:**self** action:**@selector**(danmakuShow:) forControlEven
```

CFDanmakuView：弹幕的属性在初始化时配置。



```
// 以下属性都是必须配置的--------
// 弹幕动画时间
@property(nonatomic, assign) CGFloat duration;
// 中间上边/下边弹幕动画时间
@property(nonatomic, assign) CGFloat centerDuration;
// 弹幕弹道高度
@property(nonatomic, assign) CGFloat lineHeight;
// 弹幕弹道之间的间距
@property(nonatomic, assign) CGFloat lineMargin;

// 弹幕弹道最大行数
```

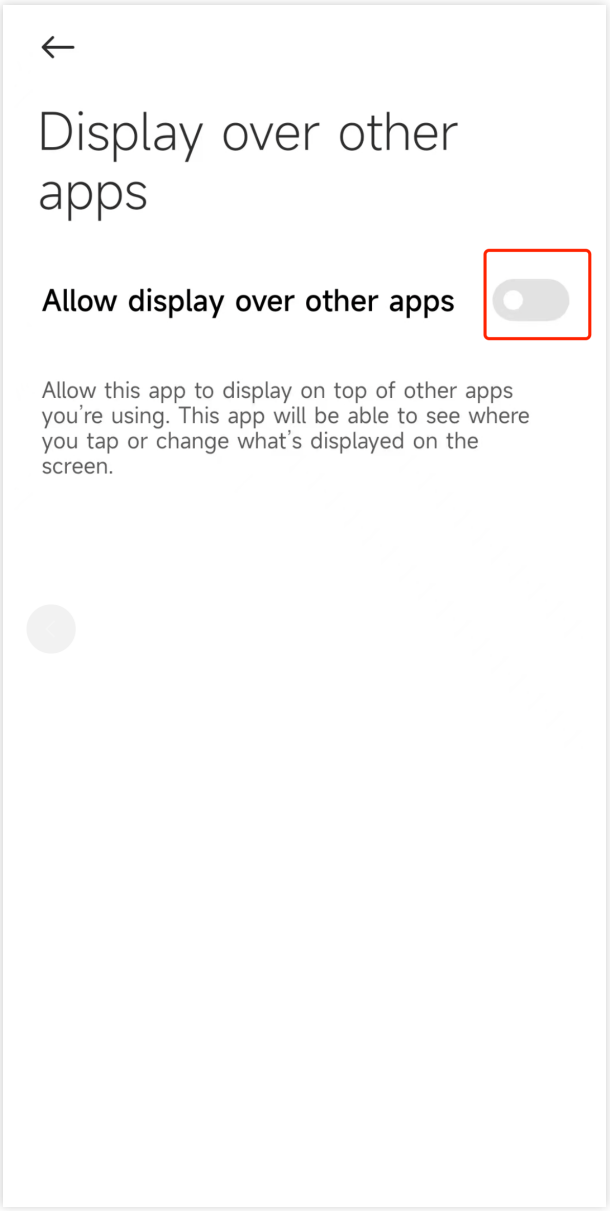
```
@property(nonatomic, assign) NSInteger maxShowLineCount;

// 弹幕弹道中间上边/下边最大行数
@property(nonatomic, assign) NSInteger maxCenterLineCount;
```

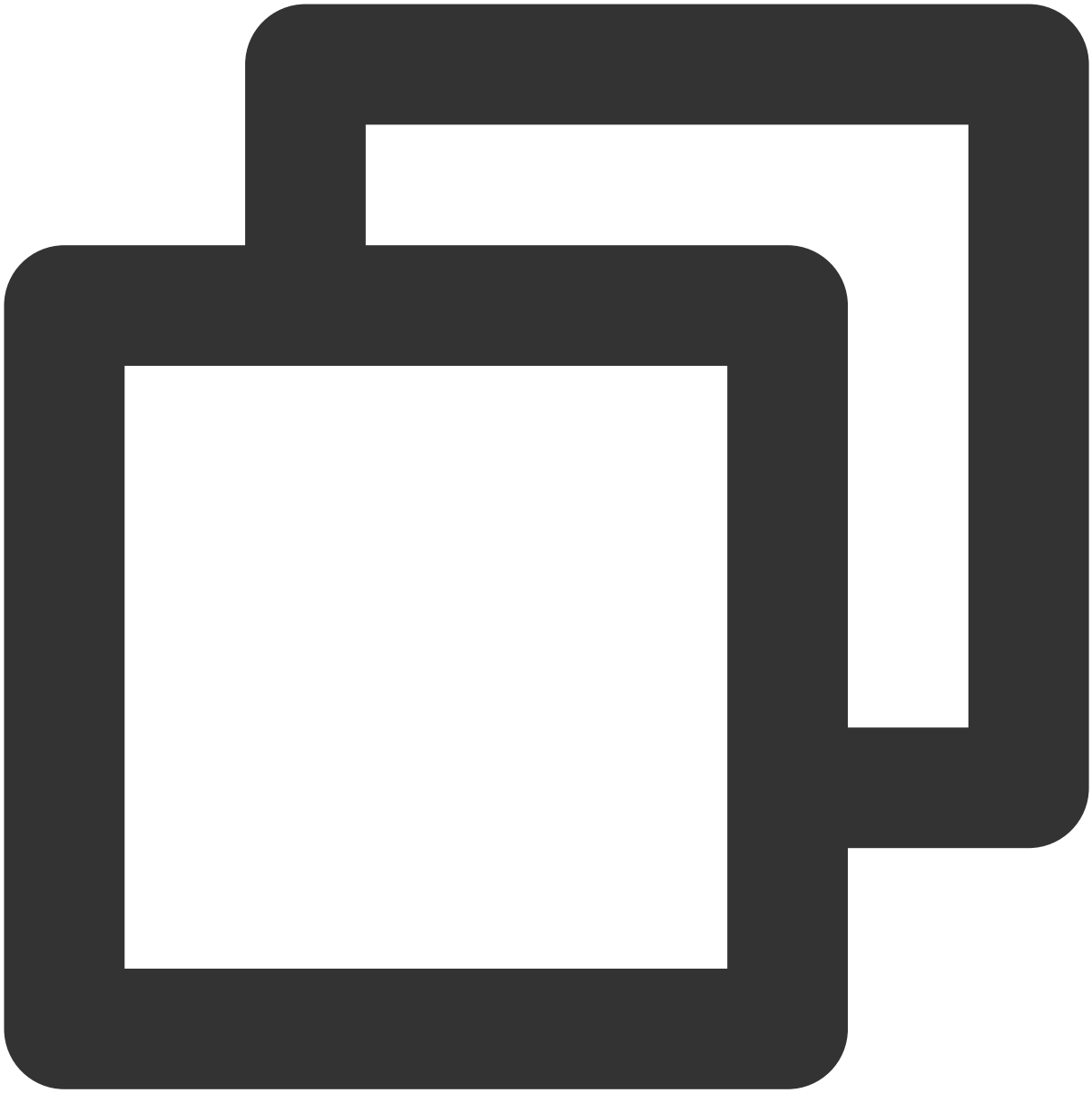
播放器组件提供播放过程中截取当前视频帧功能，您可以把图片保存起来进行分享。单击截屏按钮后，由 SDK 内部处理，无截屏成功失败的回调，截取到的图片目录为手机相册。

用户可以根据需求选择不同的视频播放清晰度，如高清、标清或超清等。单击后触发的显示清晰度view以及单击清晰度选项均由 SDK 内部处理，无回调。

2、悬浮窗播放

播放器组件支持悬浮窗小窗口播放，可以在切换到应用内其它页面时，不打断视频播放功能。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件** 中体验，单击界面左上角**返回**，即可体验悬浮窗播放功能。


Display over other apps
Allow display over other apps
Allow this app to display on top of other apps you're using. This app will be able to see where you tap or change what's displayed on the screen.

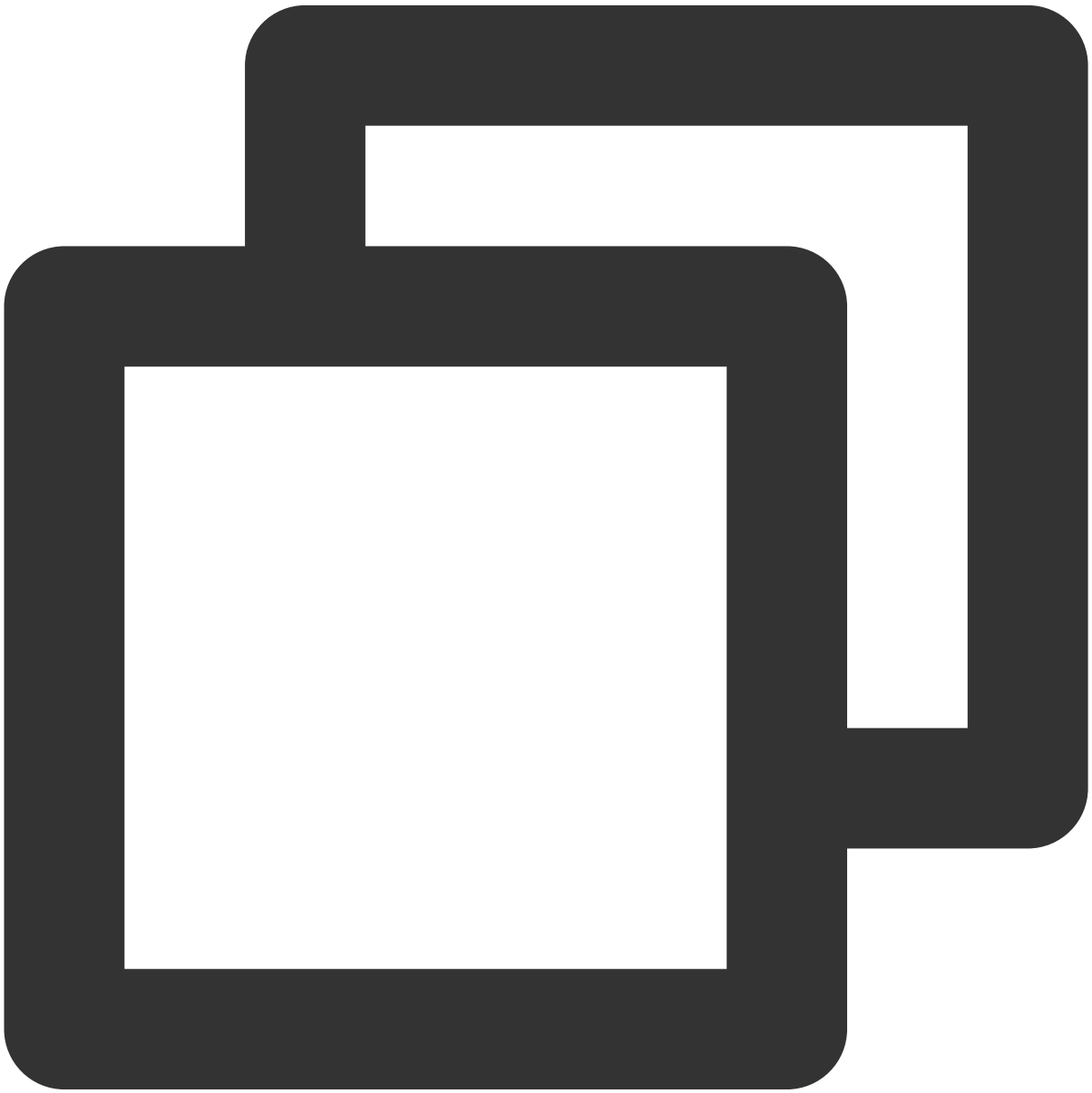


```
// 如果在竖屏且正在播放的情况下单击返回按钮会触发接口
[SuperPlayerWindowShared setSuperPlayer:self.playerView];
[SuperPlayerWindowShared show];
// 单击浮窗返回窗口触发的代码接口
SuperPlayerWindowShared.backController = self;
```

3、视频封面

播放器组件支持用户自定义视频封面，用于在视频接收到首帧画面播放回调前展示。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件 > 自定义封面演示** 视频中体验。

当播放器组件设置为自动播放模式 `PLAY_ACTION_AUTO_PLAY` 时，视频自动播放，此时将在视频首帧加载出来之前展示封面。
当播放器组件设置为手动播放模式 `PLAY_ACTION_MANUAL_PLAY` 时，需用户单击 **播放** 后视频才开始播放。在单击 **播放** 前将展示封面；在单击 **播放** 后到视频首帧加载出来前也将展示封面。
视频封面支持使用网络 URL 地址或本地 File 地址，使用方式可参见下述指引。若您通过 FileID 的方式播放视频，则可直接在云点播内配置视频封面。



```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
```

```
model.appId = 1400329071;
model.videoId = videoId;
//播放模式，可设置自动播放模式：PLAY_ACTION_AUTO_PLAY，手动播放模式：PLAY_ACTION_MANUAL_PLA
model.action  = PLAY_ACTION_MANUAL_PLAY;
//设定封面的地址为网络 url 地址，如果 coverPictureUrl 不设定，那么就会自动使用云点播控制台设置的
model.customCoverImageUrl = @"http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500
[self.playerView playWithModelNeedLicence:model];
```

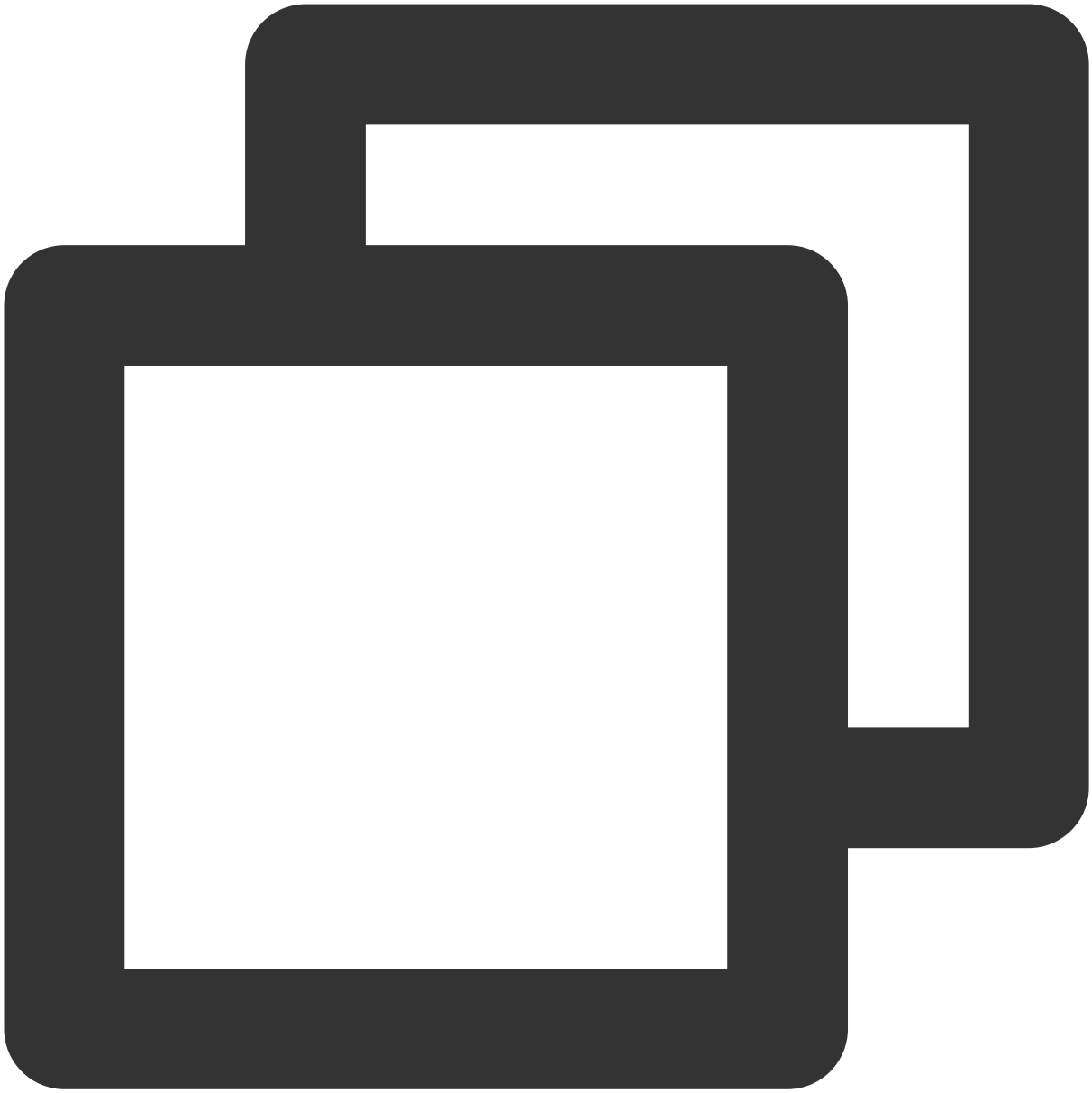
4、视频列表轮播

播放器组件支持视频列表轮播，即在给定一个视频列表后：

支持按顺序循环播放列表中的视频，播放过程中支持自动播放下一集也支持手动切换到下一个视频。

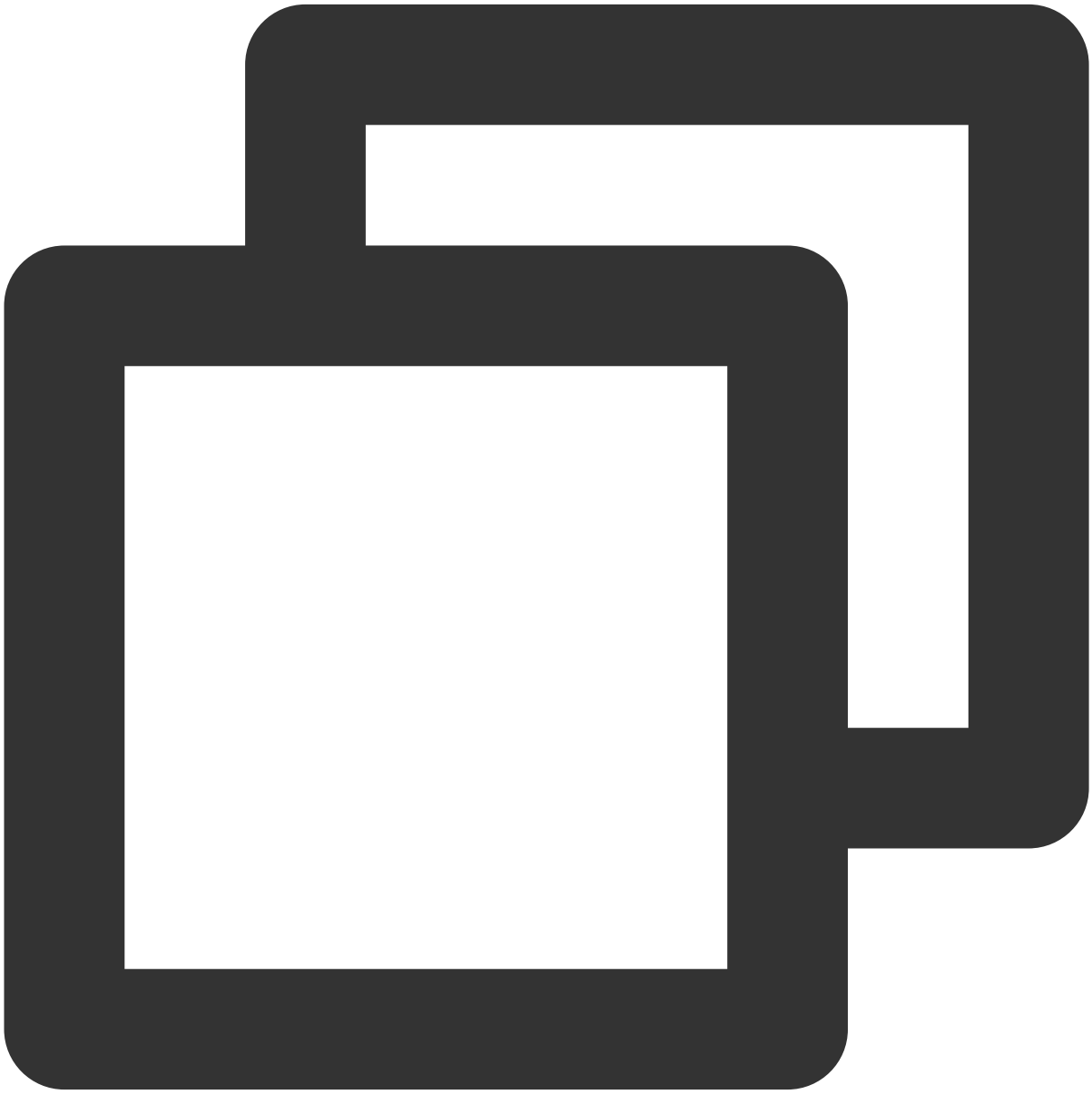
列表中最后一个视频播放完成后将自动开始播放列表中的第一个视频。

功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件 > 视频列表轮播演示** 视频中体验。



```
//步骤1:构建轮播数据的 NSMutableArray
NSMutableArray *modelArray = [NSMutableArray array];
SuperPlayerModel *model = [SuperPlayerModel new];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];

model = [SuperPlayerModel new];
videoId = [SuperPlayerVideoId new];
```

```
videoId.fileId = @"4564972819219071679";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];

//步骤2：调用 SuperPlayerView 的轮播接口
[self.playerView playWithModelListNeedLicence:modelArray isLoopPlayList:YES startIn
```



```
(void)playWithModelListNeedLicence:(NSArray *)playModelList isLoopPlayList:(BOOL)is
```

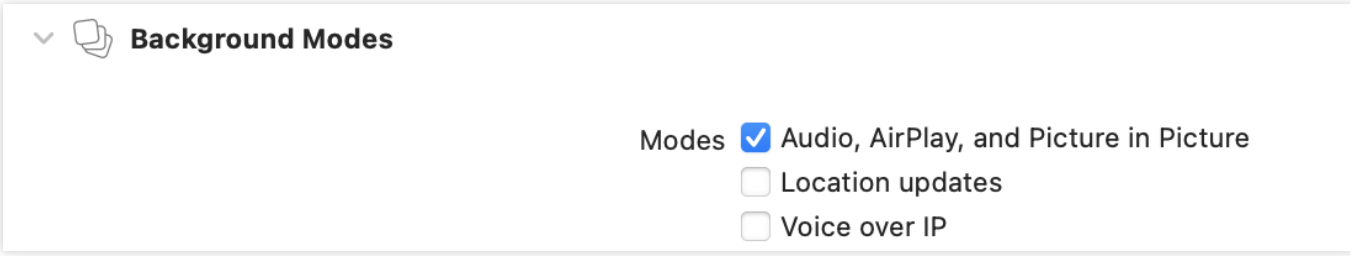
接口参数说明

参数名	类型	描述
playModelList	NSArray *	轮播数据列表
isLoop	Boolean	是否循环
index	NSInteger	开始播放的视频索引

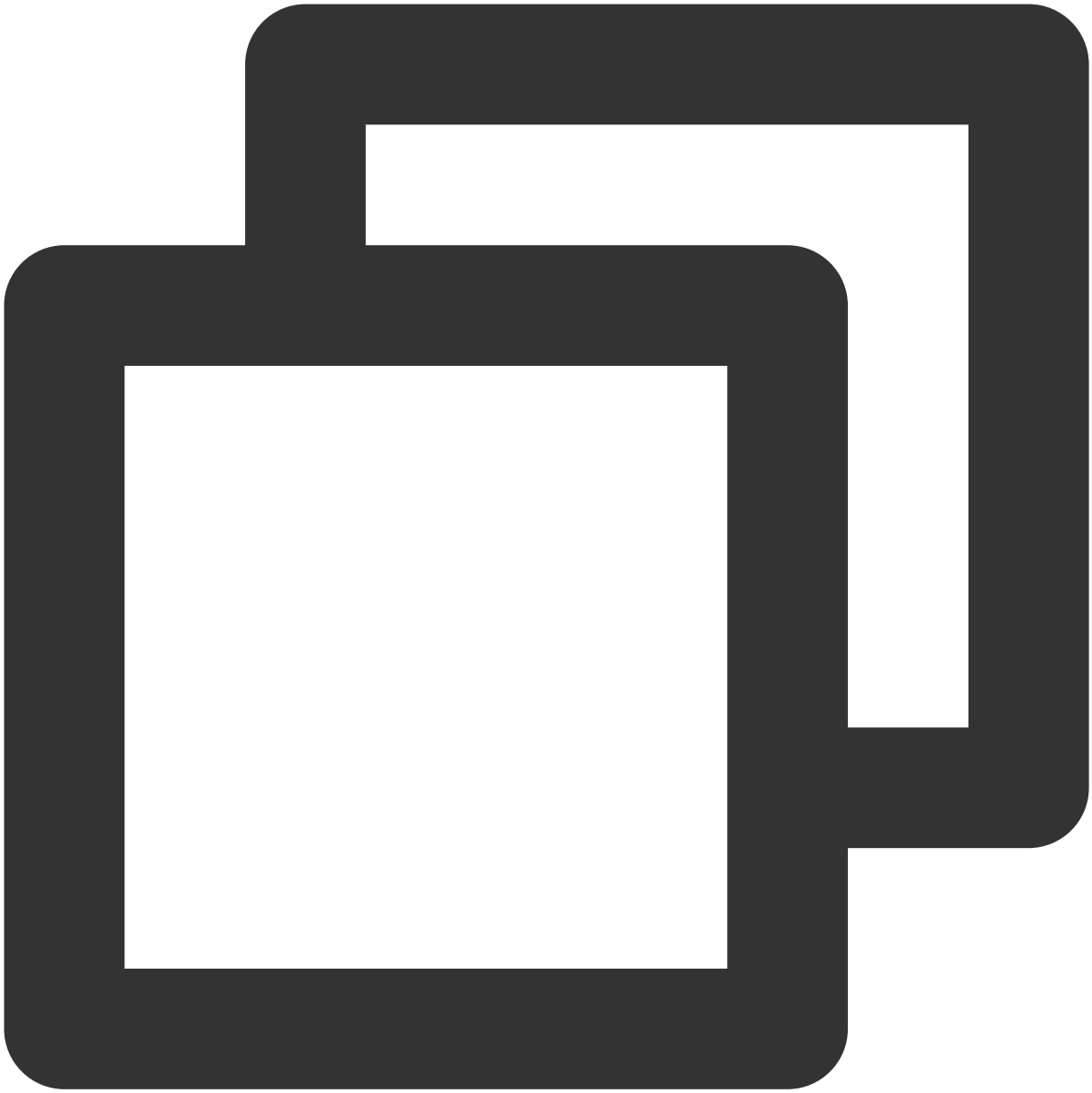
5、画中画功能

画中画（PictureInPicture）在 iOS 9就已经推出了，不过之前都只能在 iPad 上使用，iPhone 要使用画中画需更新到 iOS 14才能使用。

目前腾讯云播放器可以支持应用内和应用外画中画能力，极大的满足用户的诉求。使用前需要开通后台模式，步骤为：XCode 选择对应的 Target -> Signing & Capabilities -> Background Modes，勾选“Audio, AirPlay, and Picture in Picture”。



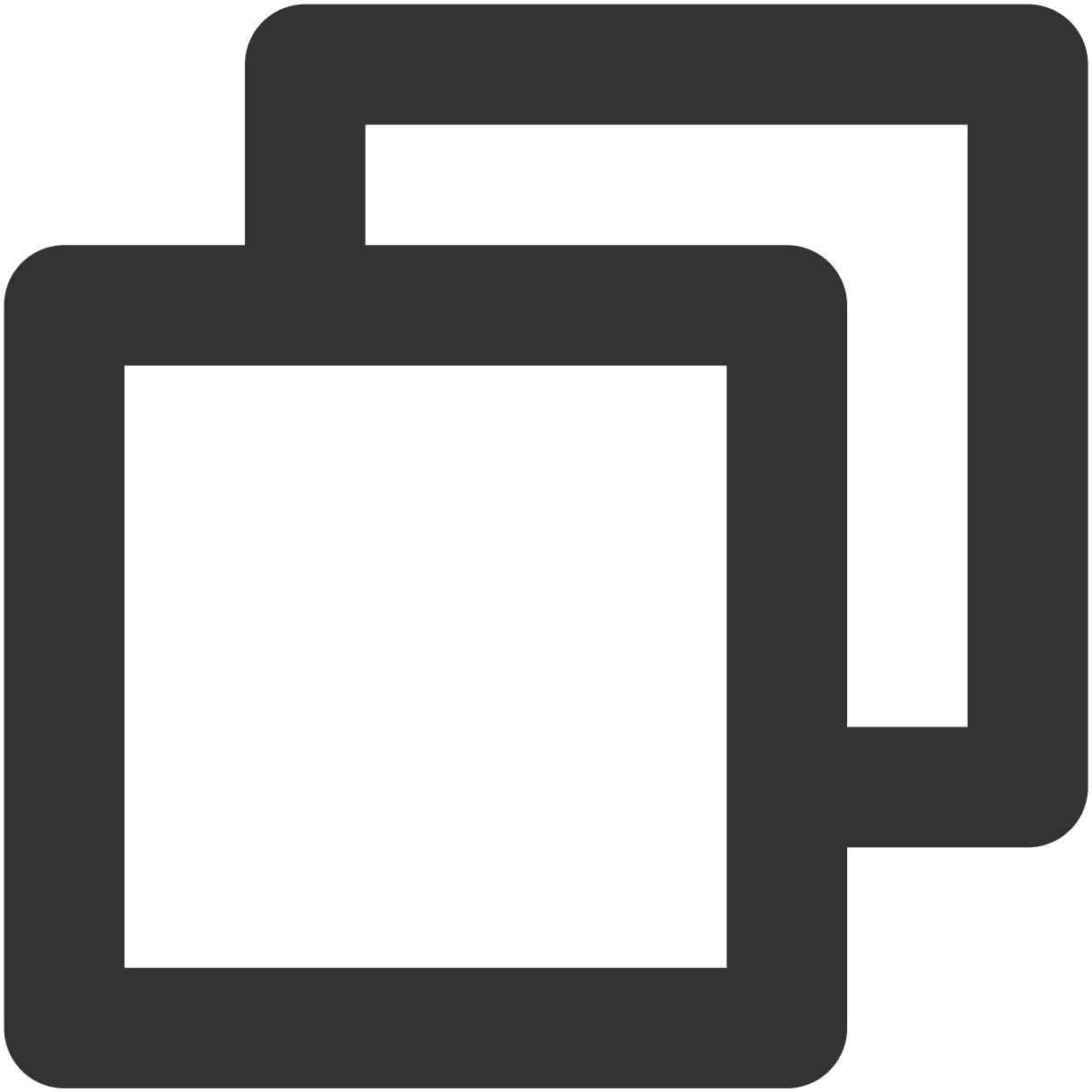
使用画中画能力代码示例：



```
// 进入画中画
 if (![TXVodPlayer isSupportPictureInPicture]) {
    return;
 }
 [_vodPlayer enterPictureInPicture];

// 退出画中画
[_vodPlayer exitPictureInPicture];
```

6、视频试看

播放器组件支持视频试看功能，可以适用于非 VIP 试看等场景，开发者可以传入不同的参数来控制视频试看时长、提示信息、试看结束界面等。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件 > 试看功能演示** 视频中体验。



```
//步骤1：创建试看 model
TXVipWatchModel *model = [[TXVipWatchModel alloc] init];
model.tipTtitle = @"可试看15秒，开通 VIP 观看完整视频";
model.canWatchTime = 15;
//步骤2：设置试看 model
self.playerView.vipWatchModel = model;
//步骤3：调用方法展示试看功能
```

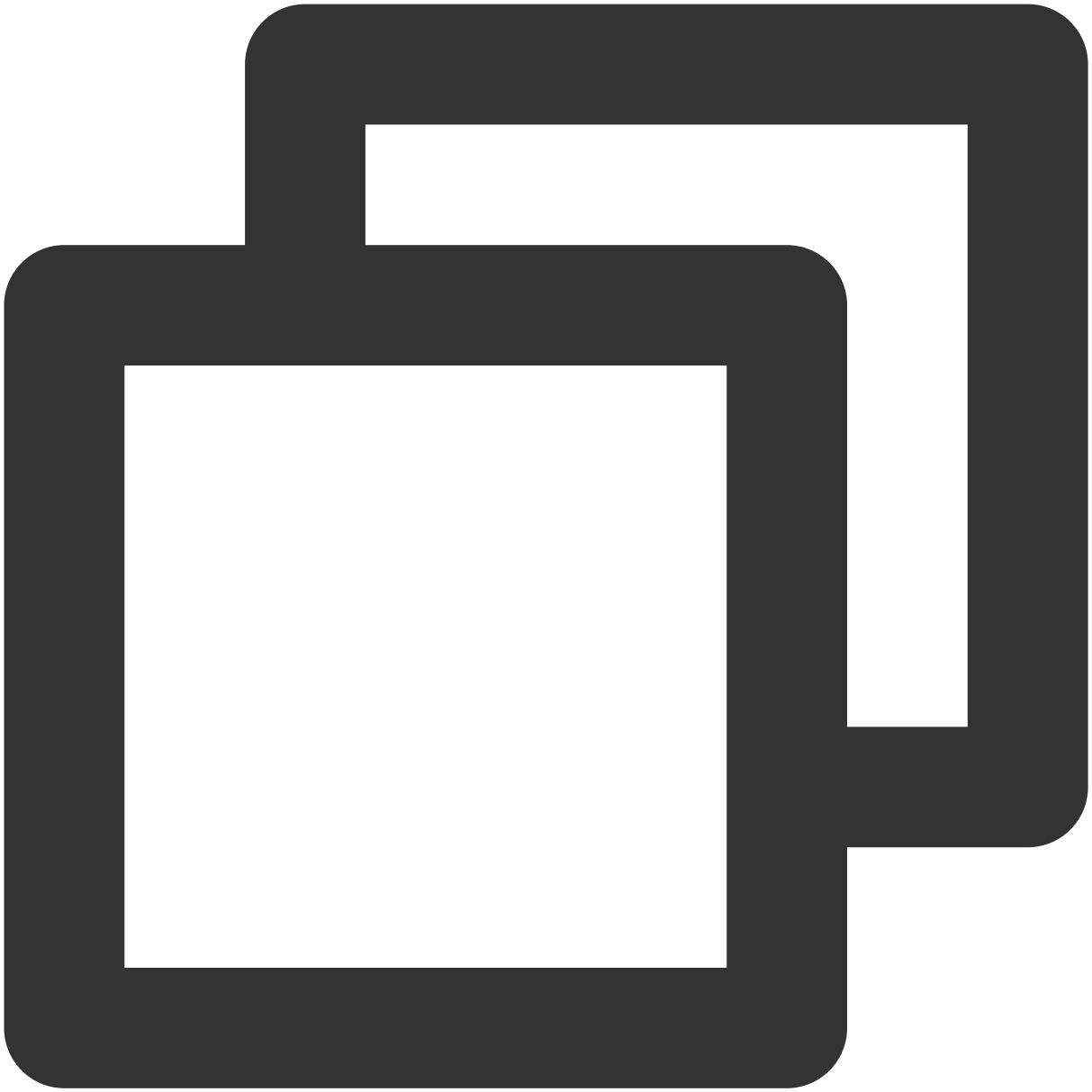
```
[self.playerView showVipTipView];
```

TXVipWatchModel 类参数说明：

参数名	类型	描述
tipTtitle	NSString	试看提示信息
canWatchTime	float	试看时长，单位为妙

7、动态水印

播放器组件支持在播放界面添加不规则跑动的文字水印，有效防盗录。全屏播放模式和窗口播放模式均可展示水印，开发者可修改水印文本、文字大小、颜色。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** > **动态水印演示** 视频中体验。



```
//步骤1：创建视频源信息 model
SuperPlayerModel *  playermodel   = [SuperPlayerModel new];
//添加视频源其他信息
//步骤2：创建动态水印 model
DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
//步骤3：设置动态水印的数据
model.dynamicWatermarkTip = @"shipinyun";
model.textFont = 30;
model.textColor = [UIColor colorWithRed:255.0/255.0 green:255.0/255.0 blue:255.0/25
playermodel.dynamicWaterModel = model;
//步骤4：调用方法展示动态水印
```

```
[self.playerView playWithModelNeedLicence:playermodel];
```

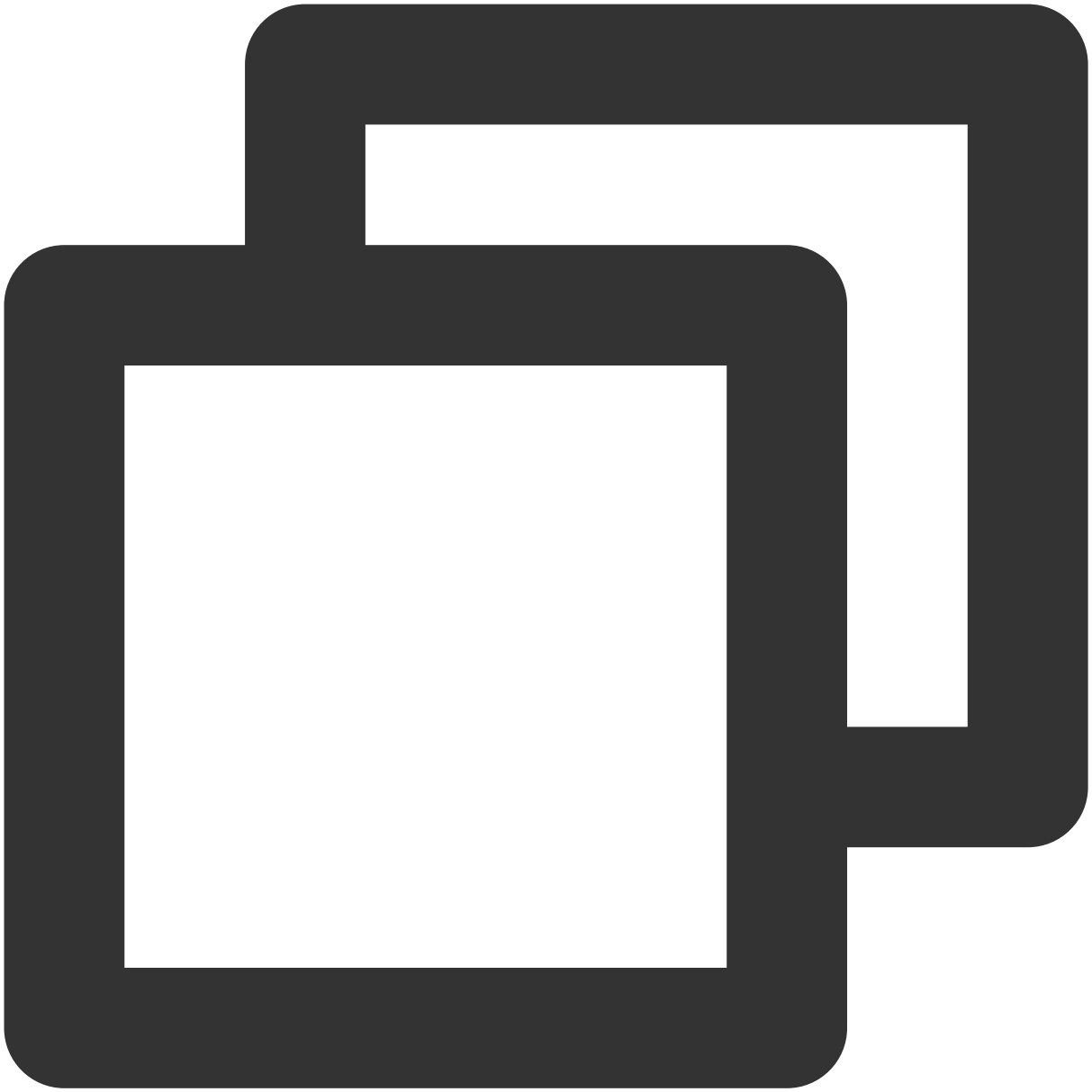
DynamicWaterModel 类参数说明：

参数名	类型	描述
dynamicWatermarkTip	NSString	水印文本信息
textFont	CGFloat	文字大小
textColor	UIColor	文字颜色

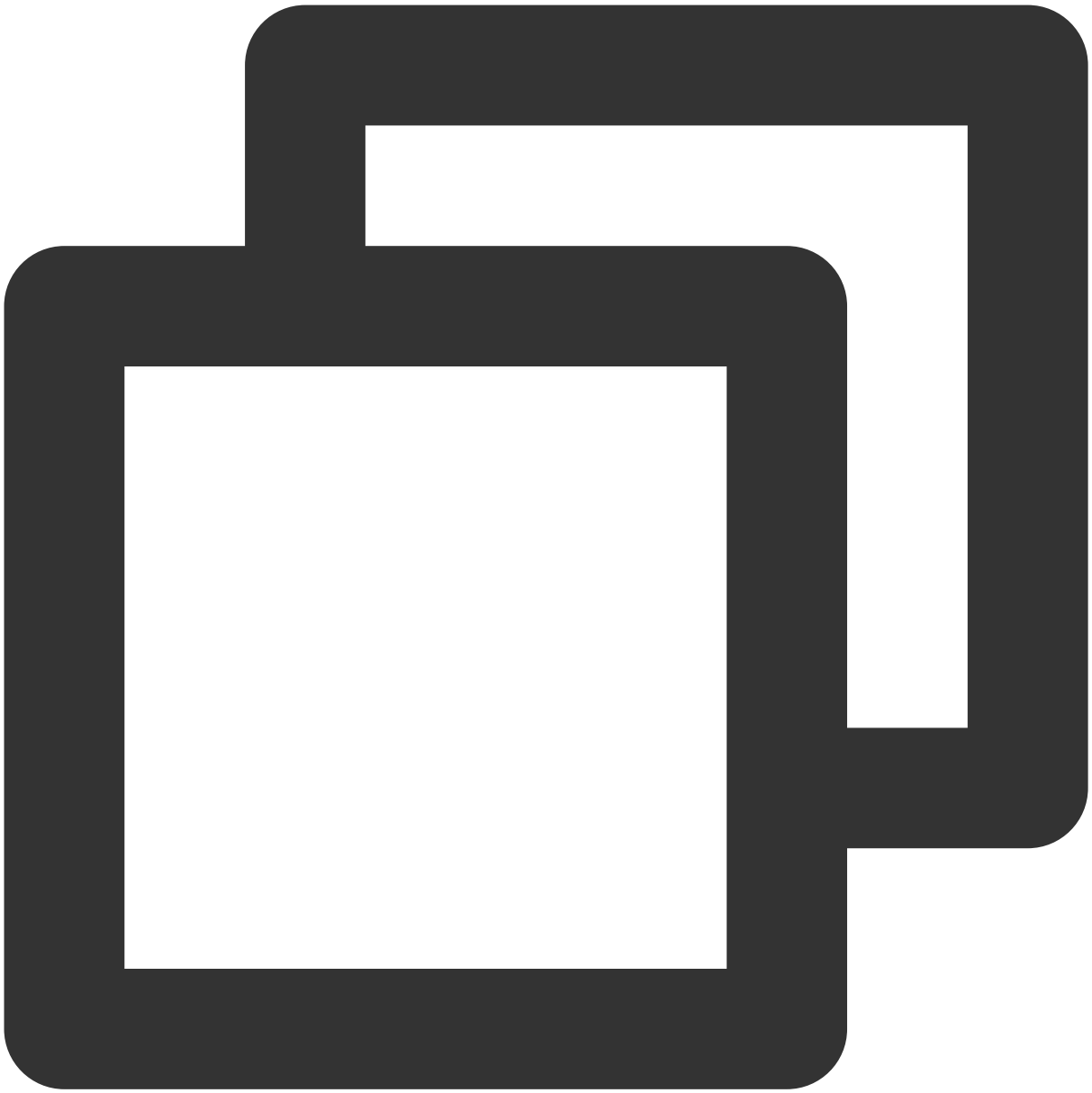
8、视频下载

支持用户在有网络的条件下缓存视频，随后在无网络的环境下观看；同时离线缓存的视频仅可在客户端内观看，不可被下载至本地，可有效防止下载视频的非法传播，保护视频安全。
你可在腾讯云视立方 App > 播放器 > 播放器组件 > 离线缓存（全屏）演示视频中，使用全屏观看模式后体验。

VideoCacheView（缓存选择列表视图）
用于选择下载对应清晰度的视频。左上角选择清晰度后，再点击要下载的视频选项，出现对勾后，代表开始了下载。点击下方的 video download list 按钮后会跳转到 VideoDownloadListView 所在的 Activity。



```
// 步骤1：初始化缓存选择列表视图
//@property (nonatomic, strong) VideoCacheView *cacheView;
_cacheView = [[VideoCacheView alloc] initWithFrame:CGRectZero];
_cacheView.hidden = YES;
[self.playerView addSubview:_cacheView];

// 步骤2：设置正在播放的视频选项
[_cacheView setVideoModels:_currentPlayVideoArray currentPlayingModel:player.player

// video download list 按钮的点击事件
- (UIButton *)viewCacheListBtn;
```



```
- (void)setVideoModels:(NSArray *)models currentPlayingModel:(SuperPlayerModel *)cu
```

接口参数说明：

参数名	类型	描述
models	NSArray	下载列表的视频数据模型
SuperPlayerModel	currentModel	当前在播放的视频数据模型

VideoCacheListView（视频下载列表）

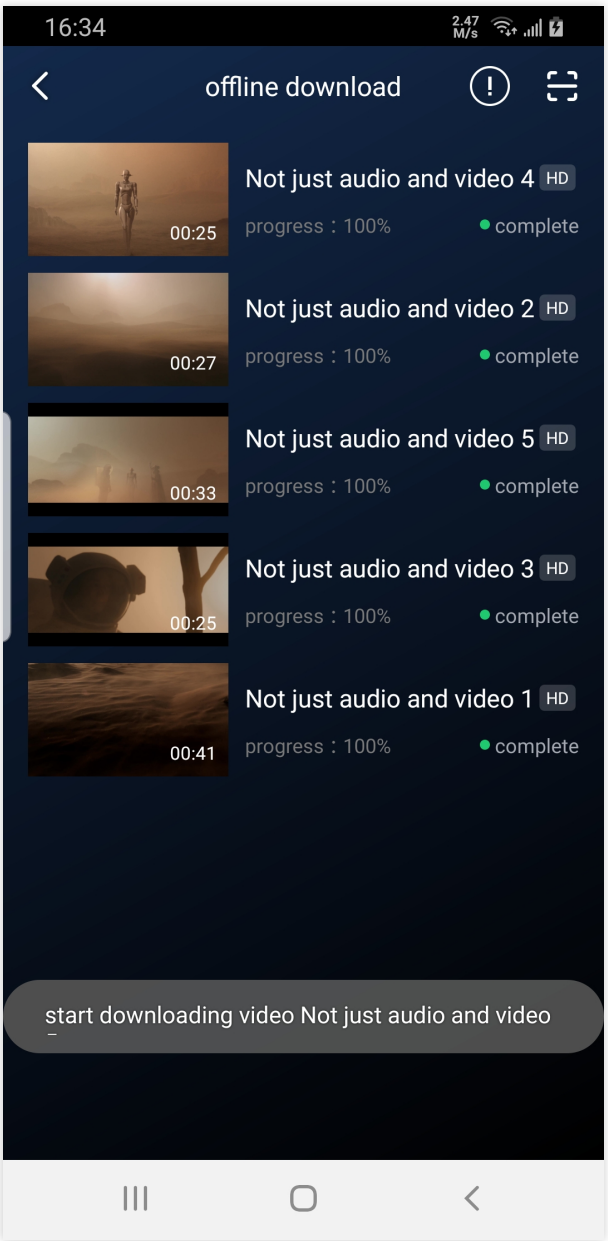
显示所有正在下载的和下载完成视频的列表 View。

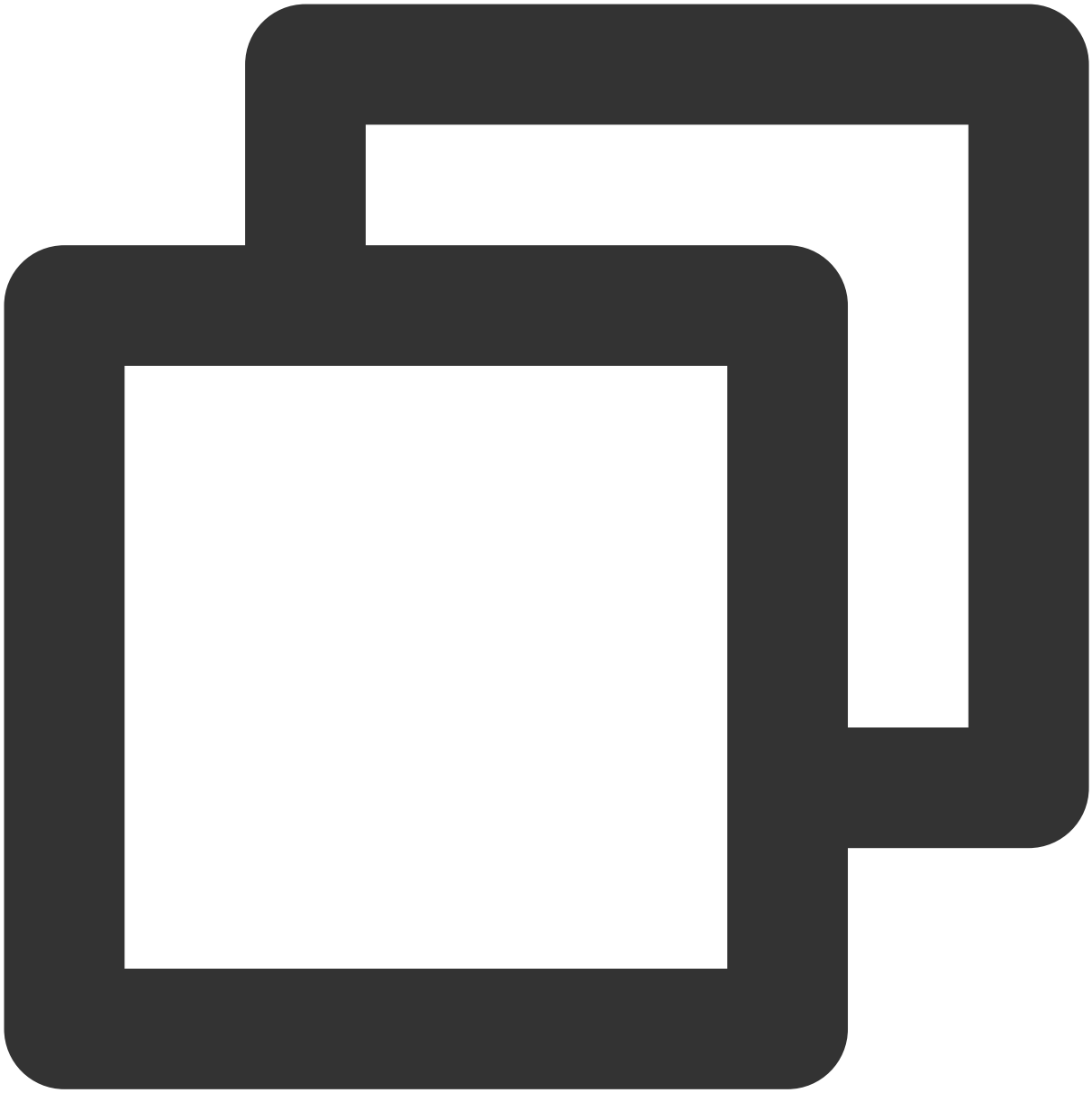
单击时：

如果正在下载，会暂停下载。

如果暂时下载，会继续下载。

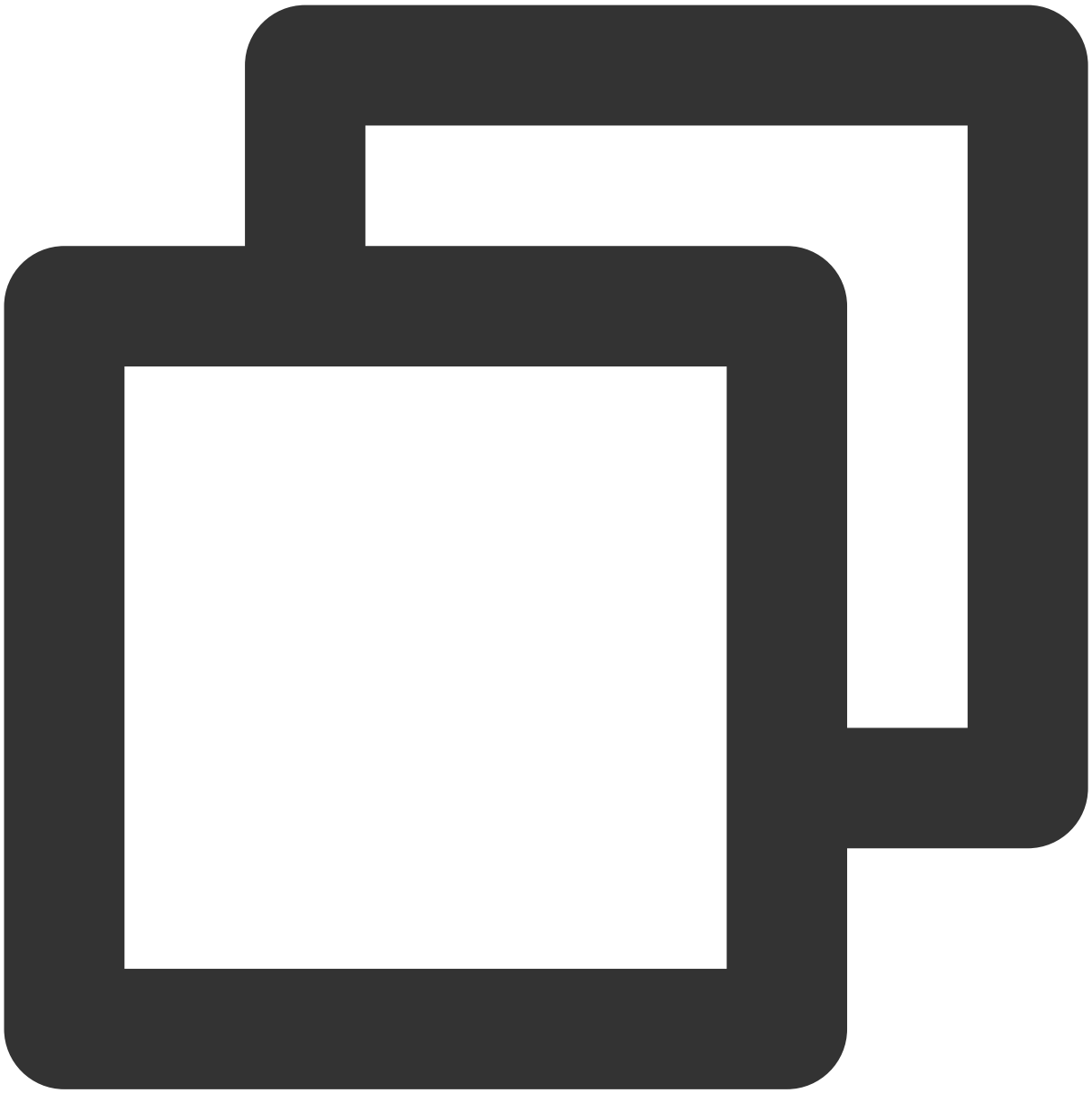
如果下载完成，会跳转播放。





```
// 添加数据，数据从TXVodDownloadManager#getDownloadMediaInfoList 接口获取到
NSArray<TXVodDownloadMediaInfo *> *array = [[[TXVodDownloadManager shareInstance] g
for (TXVodDownloadMediaInfo *info in array) {
    VideoCacheListModel *model = [[VideoCacheListModel alloc] init];
    model.mediaInfo = info;
    [self.videoCacheArray addObject:model];
}

// 列表项支持点击播放、长按删除等操作
- (void)longPress:(UILongPressGestureRecognizer *)longPress;  // 长按
```

下载后的视频支持无网络情况下进行播放，播放时请参考如下代码：



```
NSArray<TXVodDownloadMediaInfo *> *mediaInfoList = [[TXVodDownloadManager shareInst
TXVodDownloadMediaInfo *mediaInfo = [mediaInfoList firstObject];
SuperPlayerUrl *superPlayerUrl = [[SuperPlayerUrl alloc] init];
superPlayerUrl.title = @"*********";
superPlayerUrl.url = mediaInfo.playpath;
NSArray<SuperPlayerUrl *> *multiVideoURLs = @[superPlayerUrl];
SuperPlayerModel *playerModel = [[SuperPlayerModel alloc] init];
playerModel.multiVideoURLs = multiVideoURLs;
[self.playerView playWithModelNeedLicence:playerModel];
```

注意：

视频文件下载无网络播放时，一定要通过获取下载列表并通过下载列表视频对象 TXVodDownloadMediaInfo 的 PlayPath 进行播放，切勿直接保存 PlayPath 对象。

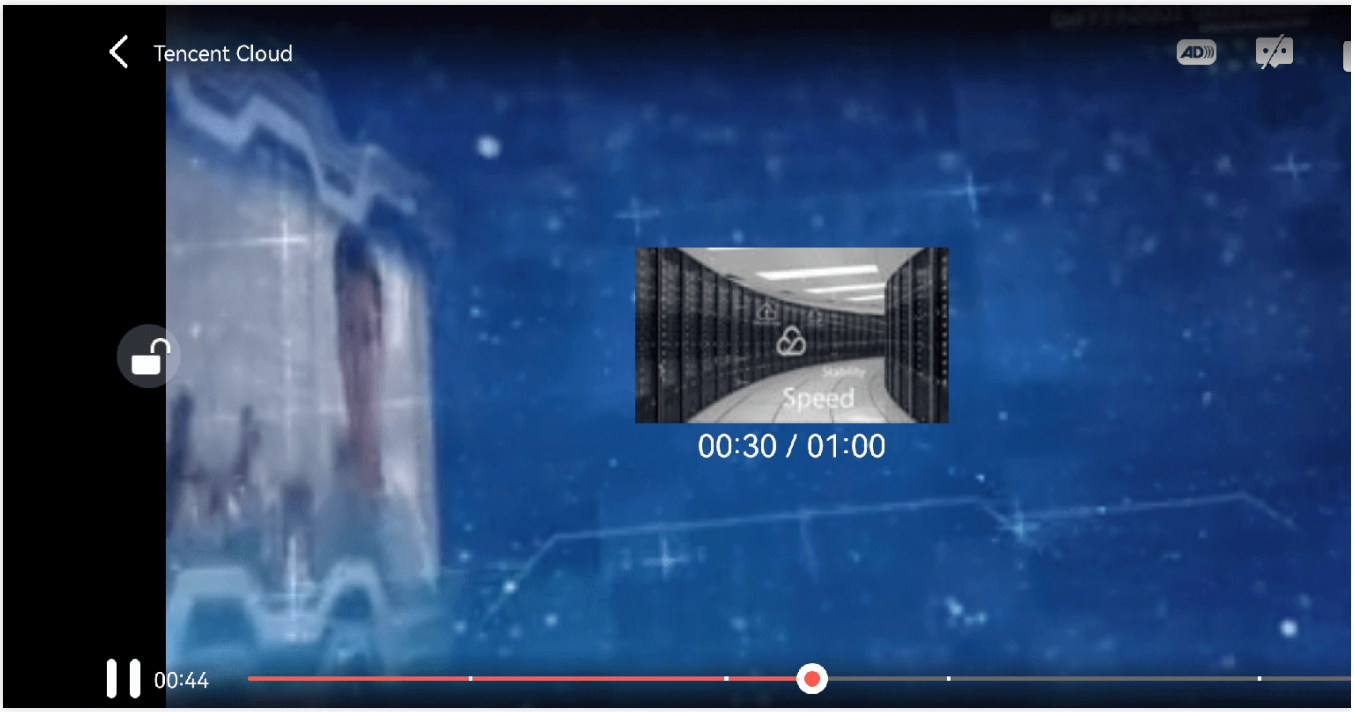
9、雪碧图和打点信息

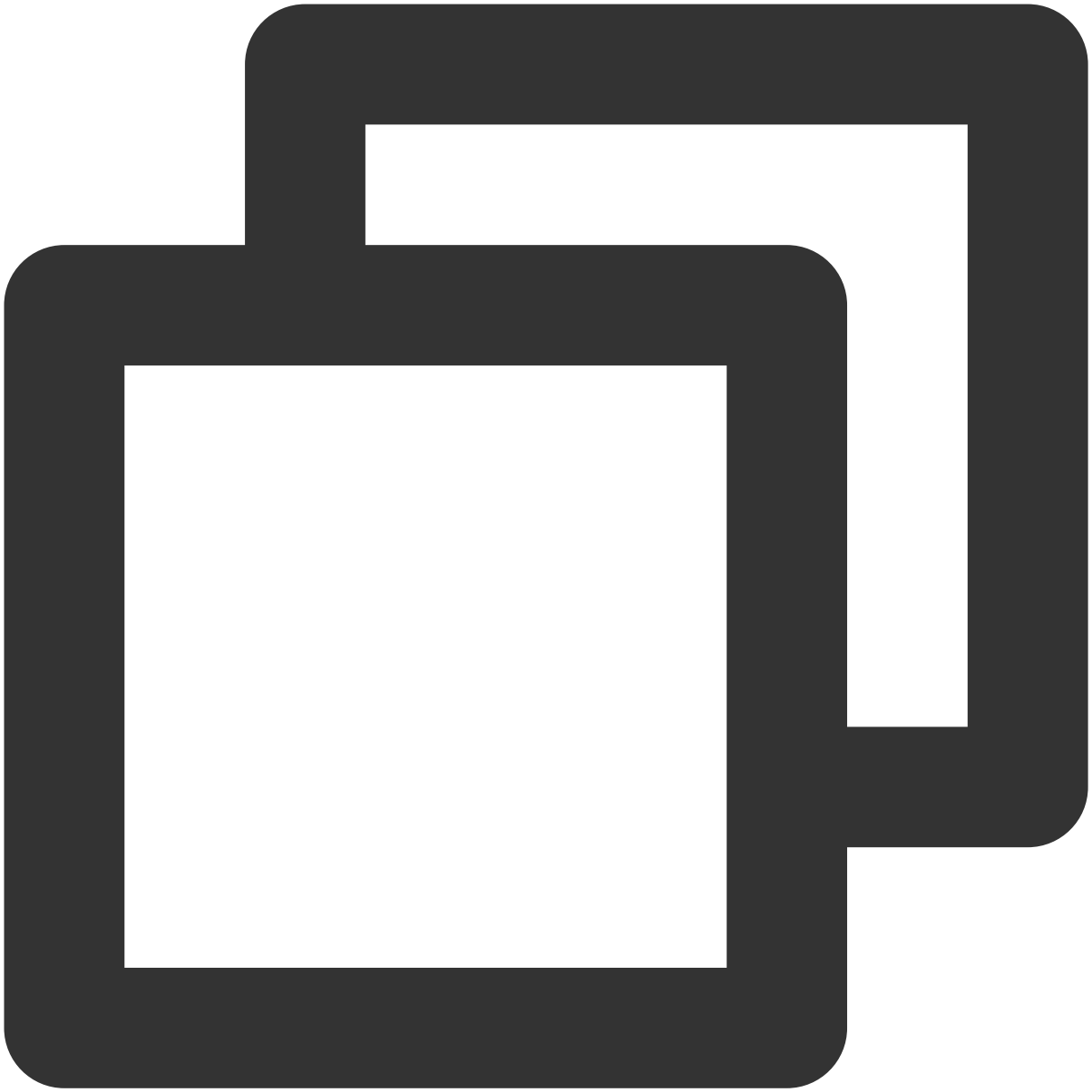
打点信息

支持在进度条关键位置添加文字介绍，用户点击后可显示打点位置的文字信息，以快速了解当前位置的视频信息。
点击视频信息后，可以seek到打点信息位置。
您可在**腾讯云视立方 App > 播放器 > 播放器组件 > 腾讯云视频**中，使用全屏观看模式后体验。

雪碧图

支持用户在拖拽进度条或执行快进操作时查看视频缩略图，以快速了解指定进度的视频内容。缩略图预览基于视频雪碧图实现，您可以在云点播控制台中生成视频文件雪碧图，或直接生成雪碧图文件。
您可在**腾讯云视立方 App > 播放器 > 播放器组件 > 腾讯云视频**中，使用全屏观看模式后体验。


Tencent Cloud
Speed
00:30 / 01:00
00:44



```
// 步骤1：通过 playWithModelNeedLicence 播放器视频，才能在 onPlayEvent 回调中获取到雪碧图和
[self.playerView playWithModelNeedLicence:playerModel];

// 步骤2: playWithModelNeedLicence 在 VOD_PLAY_EVT_GET_PLAYINFO_SUCC 回调事件中取得关键
NSString *imageSpriteVtt = [param objectForKey:VOD_PLAY_EVENT_IMAGESPRIT_WEBVTTURL]
NSArray<NSString *> *imageSpriteList = [param objectForKey:VOD_PLAY_EVENT_IMAGESPRI
NSArray<NSURL *> *imageURLs = [self convertImageSpriteList:imageSpriteList];
[self.imageSprite setVTTUrl:[NSURL URLWithString:imageSpriteVtt] imageUrls:imageURL

// 步骤3: 将拿到的打点信息和雪碧图，并显示到界面上
if (self.isFullScreen) {
```

```
    thumbnail = [self.imageSprite getThumbnail:draggedTime];
}
if (thumbnail) {
    [self.fastView showThumbnail:thumbnail withText:timeStr];
}
```

10、外挂字幕

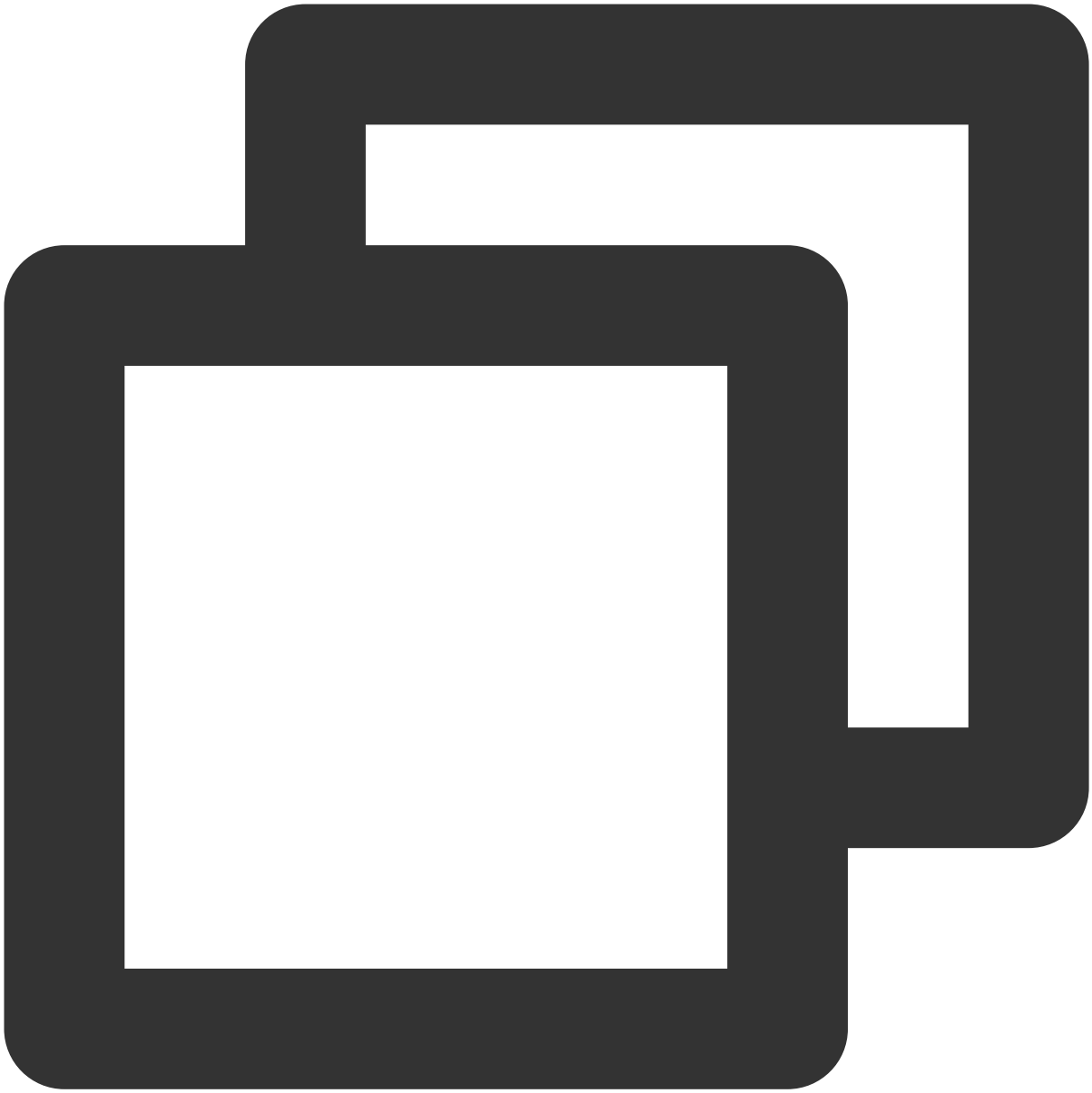
注意：

外挂字幕依赖播放器高级版本 SDK 且 SDK 需要11.3版本以上才支持。

目前支持 SRT 和 VTT 这两种格式的字幕。用法如下：

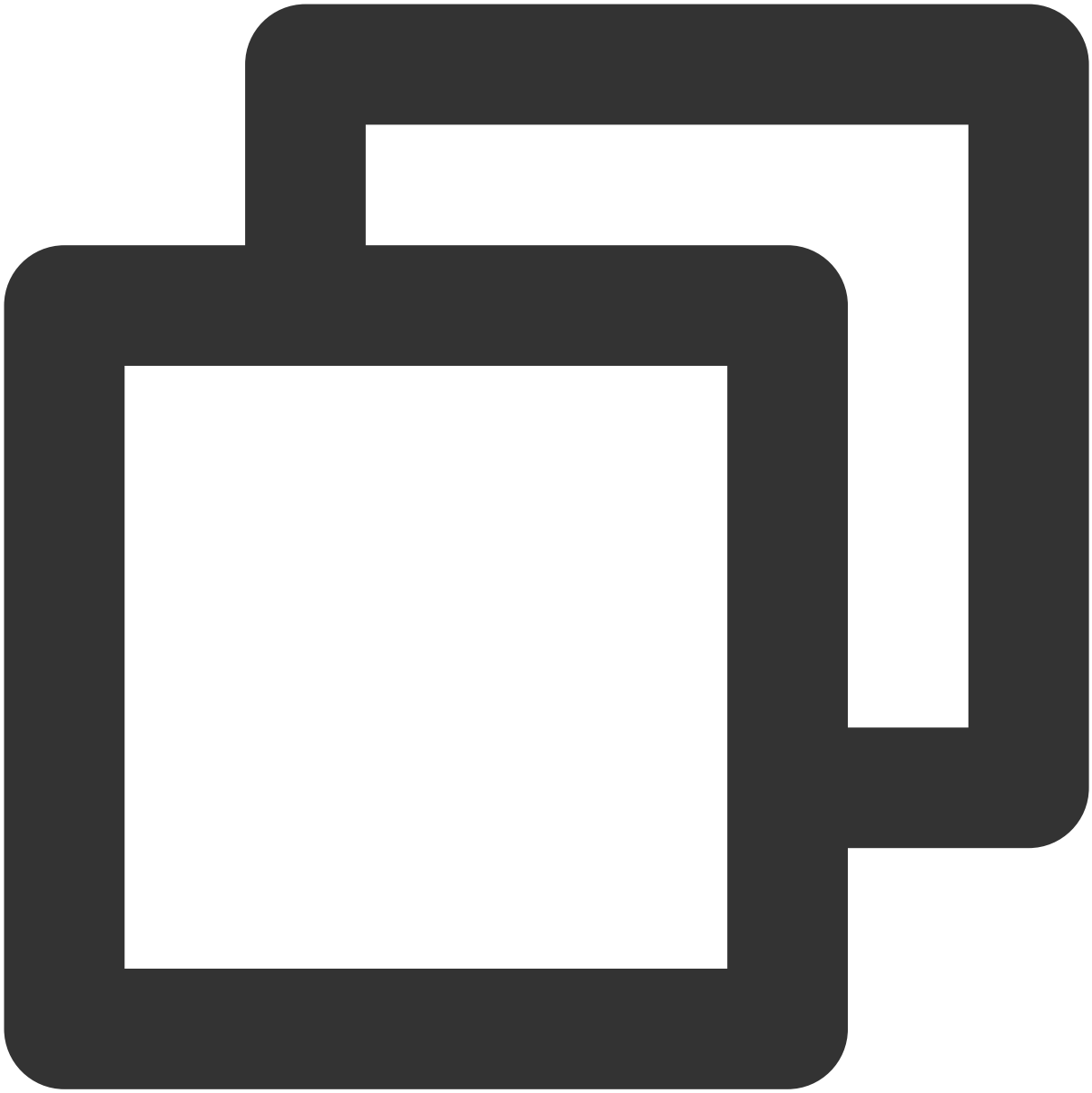
步骤1：添加外挂字幕。

往 `SuperPlayerModel#subtitlesArray` 传入外挂字幕类别字段。



```
// 传入 字幕url，  字幕名称，  字幕类型
SuperPlayerSubtitles *subtitleModel = [[SuperPlayerSubtitles alloc] init];
subtitleModel.subtitlesUrl = @"https://mediacloud-76607.gzc.vod.tencent-cloud.com/D
subtitleModel.subtitlesName = @"ex-cn-srt";
subtitleModel.subtitlesType = 0;
[subtitlesArray addObject:subtitleModel];

// 播放
[self.playerView playWithModelNeedLicence:model];
```

步骤2：播放后切换字幕。

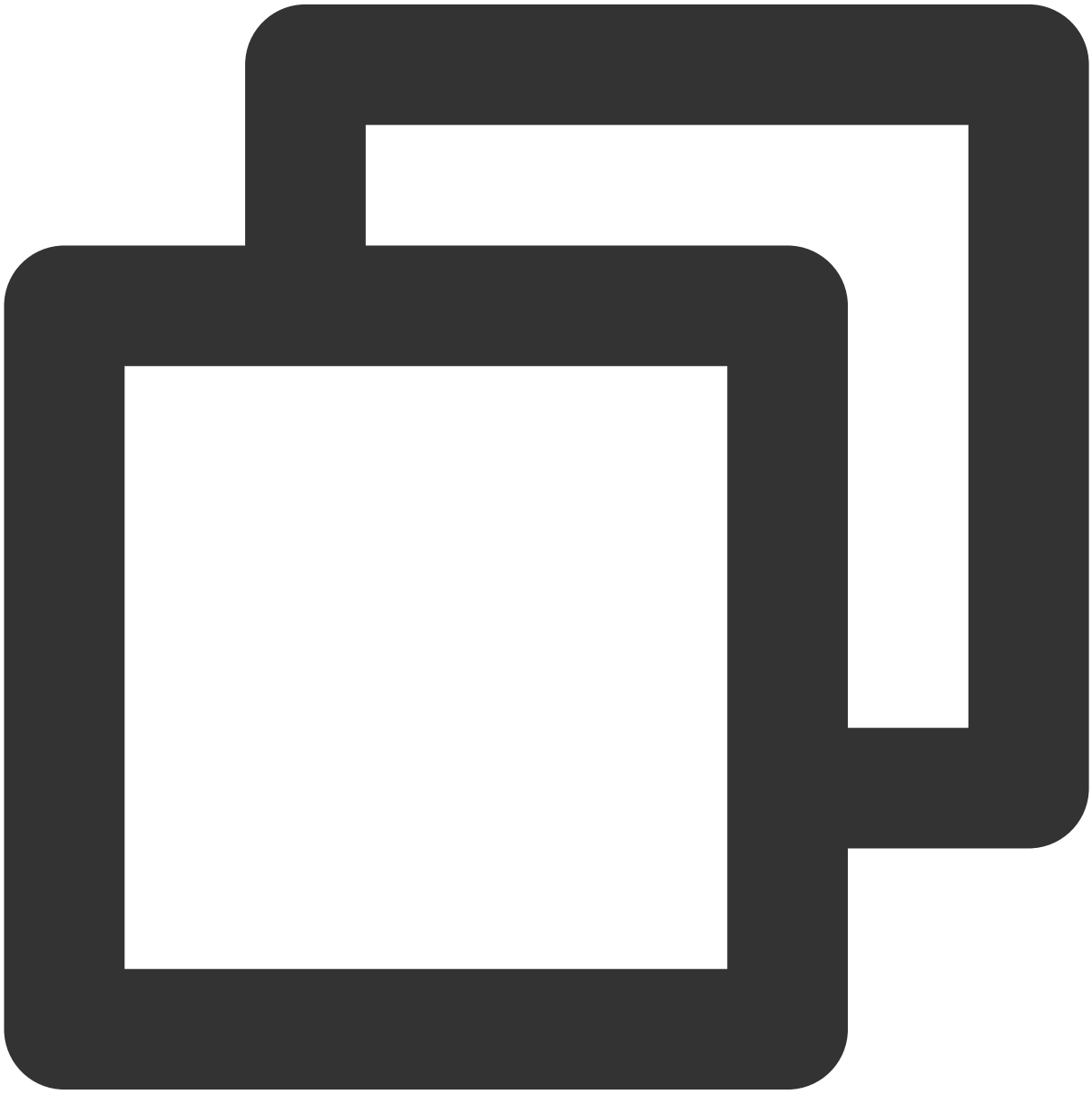


```
// 开始播放视频后，选中添加的外挂字幕
- (void)controlViewSwitch:(UIView *)controlView withSubtitlesInfo:(TXTrackInfo *)in
    if (info.trackIndex == -1) {
        [self.vodPlayer deselectTrack:preInfo.trackIndex];
        self->_lastSubtitleIndex = -1;
    } else {
        if (preInfo.trackIndex != -1) {
            // 其它字幕不需要的话， 进行deselectTrack
            [self.vodPlayer deselectTrack:preInfo.trackIndex];
        }
        // 选中字幕
```

```
        [self.vodPlayer selectTrack:info.trackIndex];
        self->_lastSubtitleIndex = info.trackIndex;
    }
}
```

步骤3：配置字幕样式。

字幕样式支持在播放前或者播放过程中配置。



```
TXPlayerSubtitleRenderModel *model = [[TXPlayerSubtitleRenderModel alloc] init];
model.canvasWidth = 1920;  // 字幕渲染画布的宽
model.canvasHeight = 1080;  // 字幕渲染画布的高
```

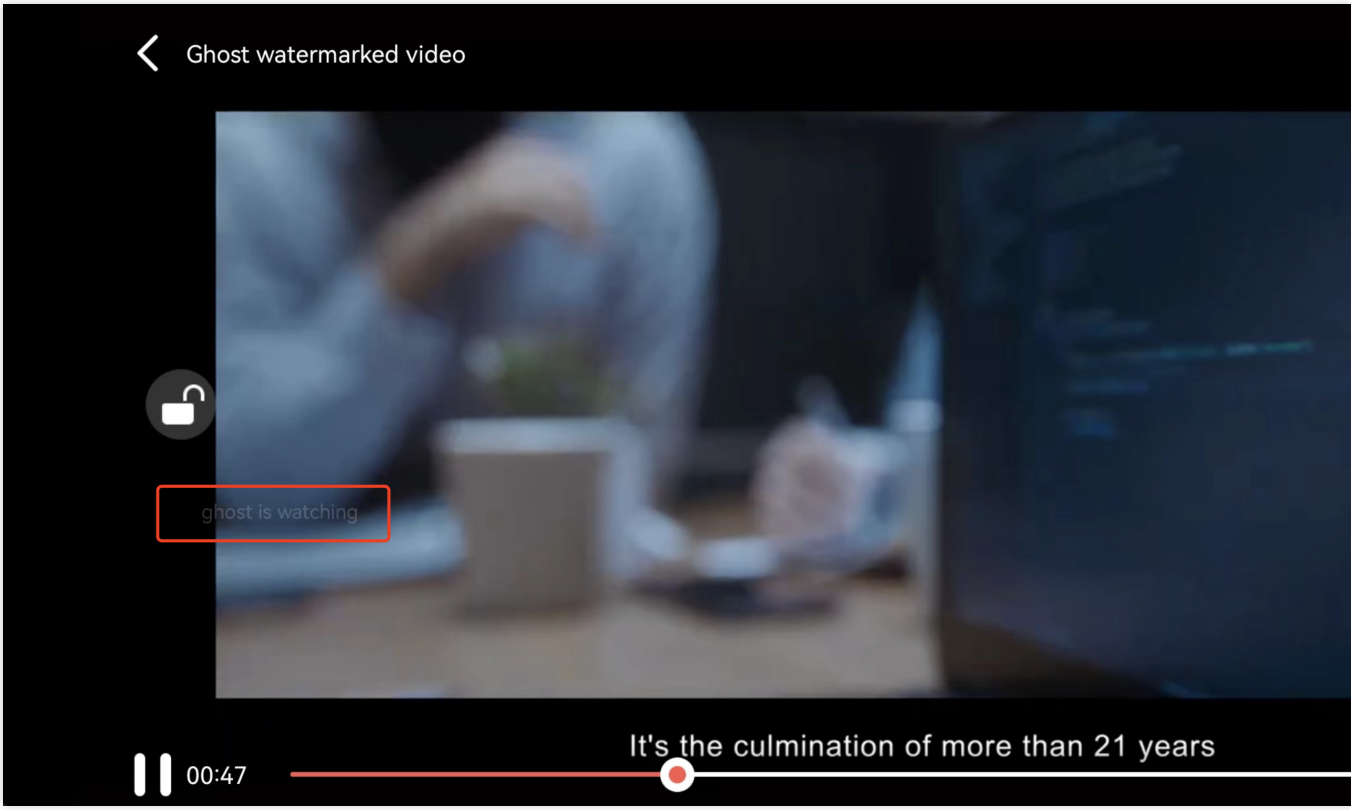
```
model.isBondFontStyle = NO;  // 设置字幕字体是否为粗体
model.fontColor = 0xFF000000; // 设置字幕字体颜色，默认白色不透明
[_txVodPlayer setSubtitleStyle:model];
```

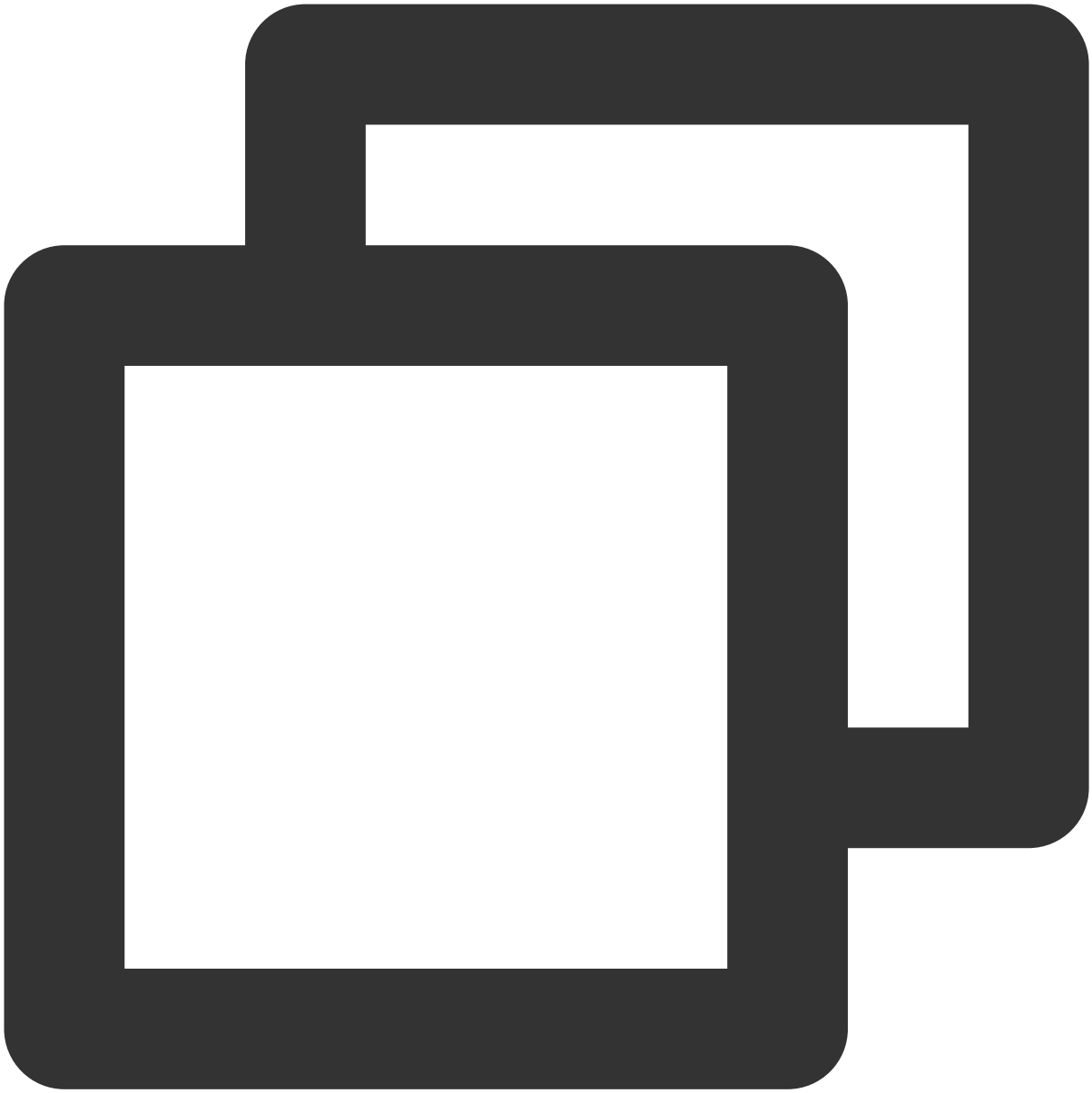
11、幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程。幽灵水印仅在视频上出现一段很短的时间，这种闪现对视频的观看影响很微小。每次水印出现的画面位置都不固定，杜绝了他人遮挡水印的企图。效果如下图所示，在视频开始播放时，就会出现一次水印，然后消失。等到下一次再出现，再消失。

幽灵水印的内容在收到播放器的 `VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `[param objectForKey:@"EVT_KEY_WATER_MARK_TEXT"]` 获取。

注意： 播放器 11.6 版本开始支持。





```
// 步骤 1： 配置支持幽灵水印的 FileId 播放视频
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1500006438;
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"387702307847129127";
model.videoId.pSign =
@"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwNjQzOCwiZmlsZUlkIjoiMzg3
[_playerView playWithModelNeedLicence:model];

// 步骤 2： 在 SuperPlayerView 收到 PLAY_EVT_GET_PLAYINFO_SUCC事件后，获取幽灵水印内容展示
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
```

```
    dispatch_async(dispatch_get_main_queue(), ^{
        if (EvtID == PLAY_EVT_PLAY_EVT_GET_PLAYINFO_SUCCPLAY_PROGRESS) {
            NSString *ghostWaterText = [param objectForKey:@"EVT_KEY_WATER_MARK_TEX
            if (ghostWaterText && ghostWaterText.length > 0) {
                DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
                model.showType = ghost;
                model.duration = self.playerModel.duration;
                model.dynamicWatermarkTip = ghostWaterText;
                model.textFont = 30;
                model.textColor = [UIColor redColor];
                if (![self.subviews containsObject:self.watermarkView]) {
                    [self addSubview:self.watermarkView];
                    [self.watermarkView mas_makeConstraints:^(MASConstraintMaker *m
                        make.edges.equalTo(self);
                    }];
                }
                [self.watermarkView setDynamicWaterModel:model];
            }
        }
    };
}
```

Demo 体验

更多完整功能可直接运行工程 Demo，或扫码下载移动端 Demo 腾讯云视立方 App体验。

运行工程 Demo

1. 在 Demo 目录，执行命令行 `pod update`，重新生成 `TXLiteAVDemo.xcworkspace` 文件。
2. 双击打开工程，修改证书选择真机运行。
3. 成功运行 Demo 后，进入 **播放器** > **播放器组件**，可体验播放器功能。

Android 接入指引

最近更新时间：2024-04-18 17:06:54

产品概述

腾讯云 Android 播放器组件是腾讯云开源的一款播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美市面上各种流行视频 App 的播放软件。
若播放器组件满足不了您的业务的个性化需求，且您具有一定的开发经验，可以集成 播放器 SDK，自定义开发播放器界面和播放功能。

准备工作

1. 为了您体验到更完整全面的播放器功能，建议您开通 云点播 相关服务，未注册用户可注册账号 试用。若您不使用云点播服务，可略过此步骤，但集成后仅可使用播放器基础能力。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

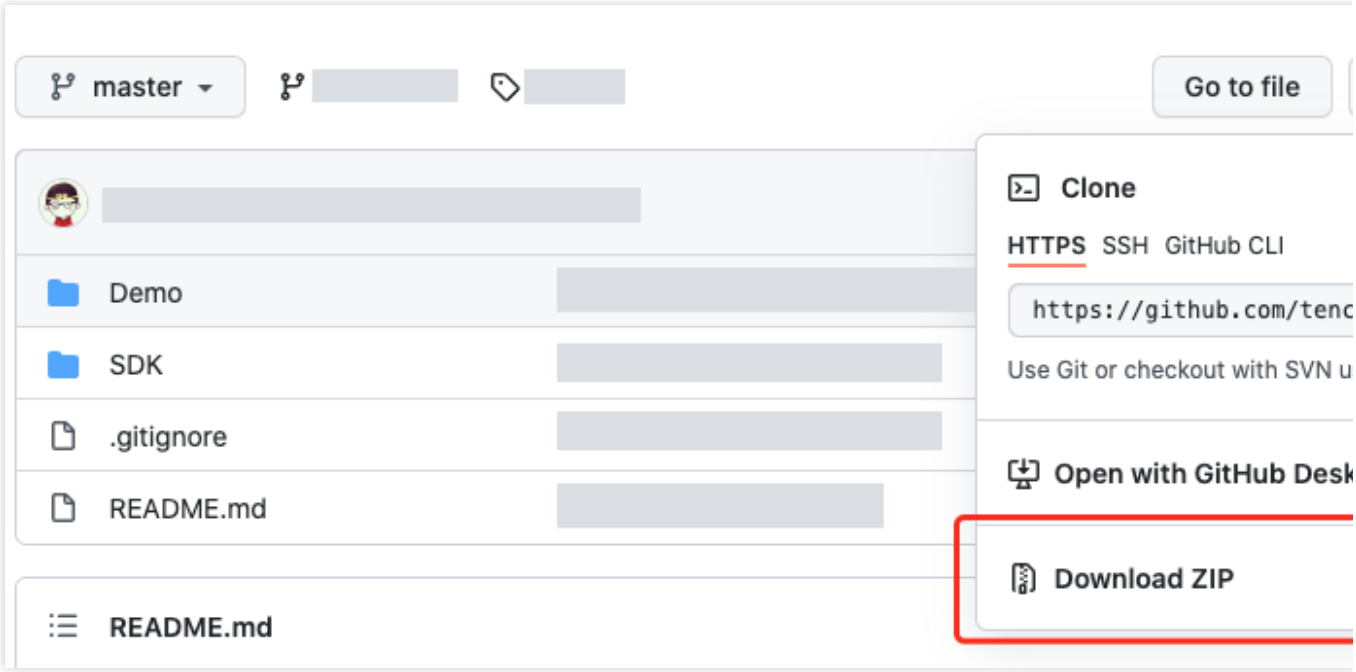
通过本文您可以学会

1. 如何集成腾讯云 Android 播放器组件
2. 如何创建和使用播放器

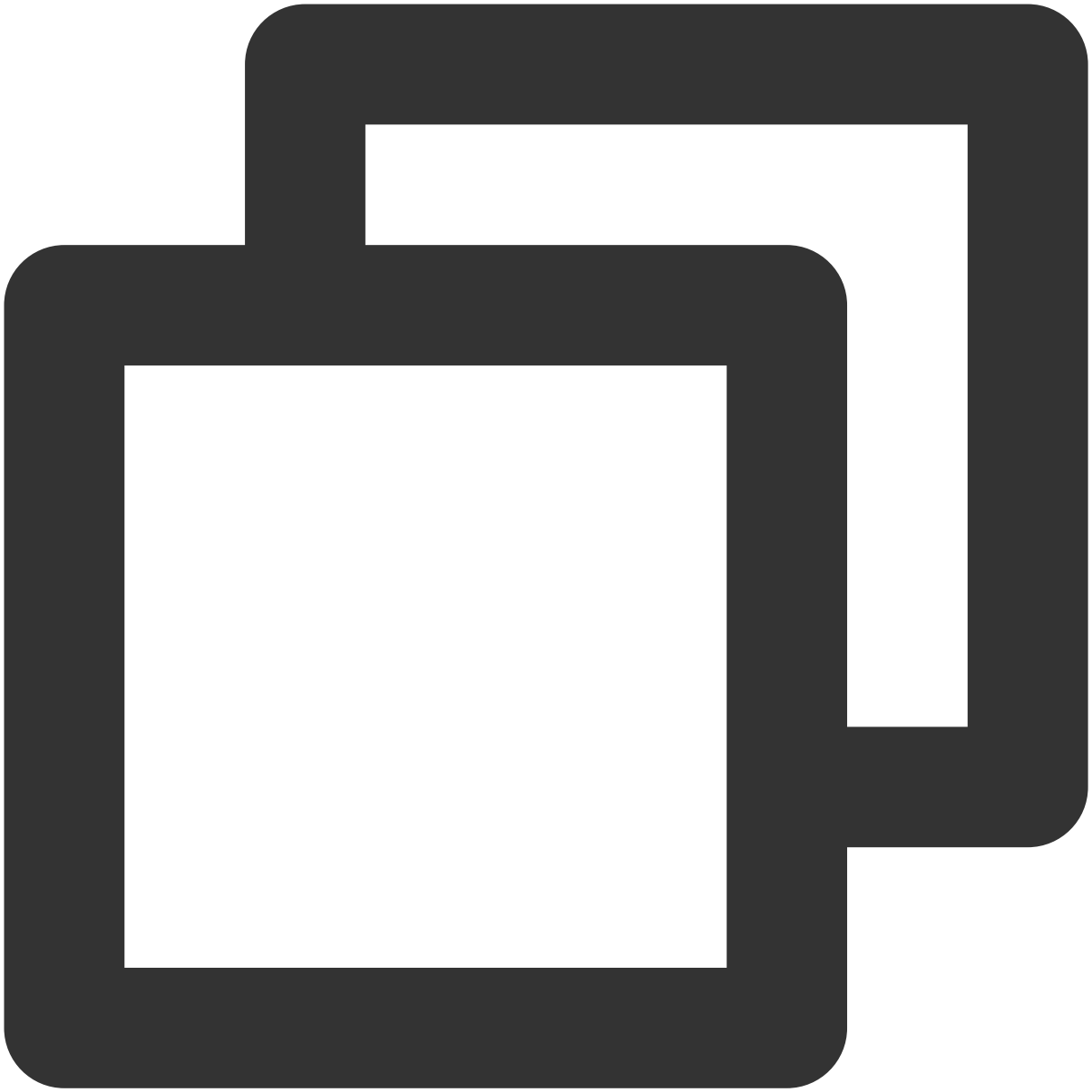
集成准备

步骤1：项目下载

腾讯云视立方 Android 播放器组件的项目地址是 SuperPlayer_Android。
您可通过 **下载播放器组件 ZIP 包** 或 **Git 命令下载** 的方式下载腾讯云视立方 Android 播放器组件项目工程。
下载播放器组件 ZIP 包
Git 命令下载
您可以直接下面播放器组件 ZIP包，单击页面的 **Code** > **Download ZIP** 下载。

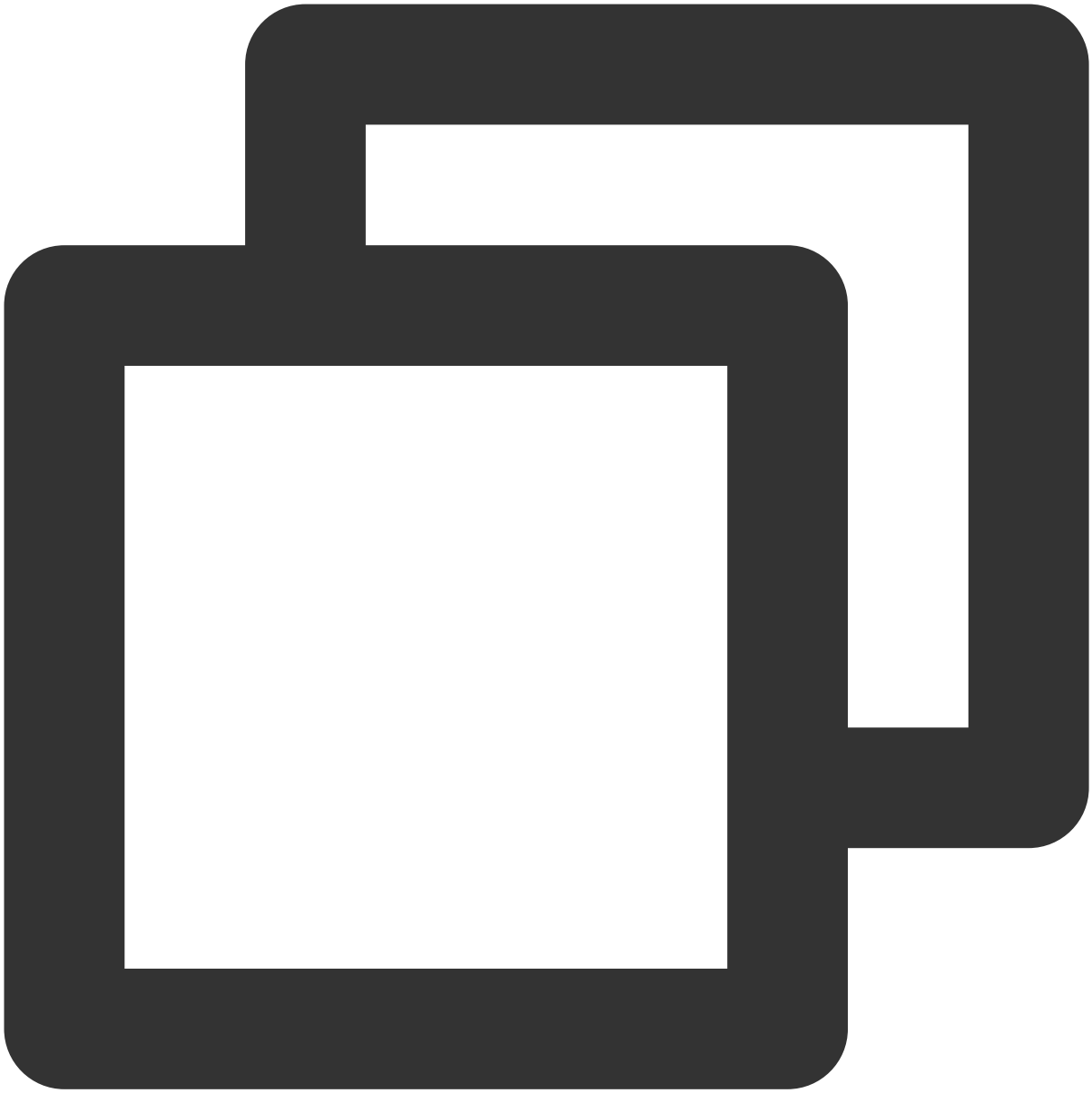


1. 首先确认您的电脑上安装了 Git，如果没有安装，可以参见 Git 安装教程 进行安装。
2. 执行下面的命令把播放器组件组件工程代码 clone 到本地。



```
git clone git@github.com:tencentyun/SuperPlayer_Android.git
```

提示下面的信息表示成功 clone 工程代码到本地。



```
正克隆到 'SuperPlayer_Android'...
remote: Enumerating objects: 2637, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (333/333), done.
remote: Total 2637 (delta 227), reused 524 (delta 170), pack-reused 1993
接收对象中: 100% (2637/2637), 571.20 MiB | 3.94 MiB/s, 完成.
处理 delta 中: 100% (1019/1019), 完成.
```

下载工程后，源码解压后的目录如下：

文件名	作用

LiteAVDemo(Player)	播放器组件 Demo 工程，导入到 Android Studio 后可以直接运行
app	主界面入口
superplayerkit	播放器组件（SuperPlayerView），具备播放、暂停、手势控制等常见功能
superplayerdemo	播放器组件 Demo 代码
common	工具类模块
SDK	视立方播放器 SDK，包括：LiteAVSDK_Player_x.x.x.aar，aar 格式提供的 SDK；LiteAVSDK_Player_x.x.x.zip，lib 和 jar 格式提供的 SDK
Player 说明文档 (Android).pdf	播放器组件使用文档

步骤2：集成指引

本步骤可指导您如何集成播放器，您可选择使用 Gradle 自动加载的方式，手动下载 aar 再将其导入到您当前的工程或导入 jar 和 so 库的方式集成项目。

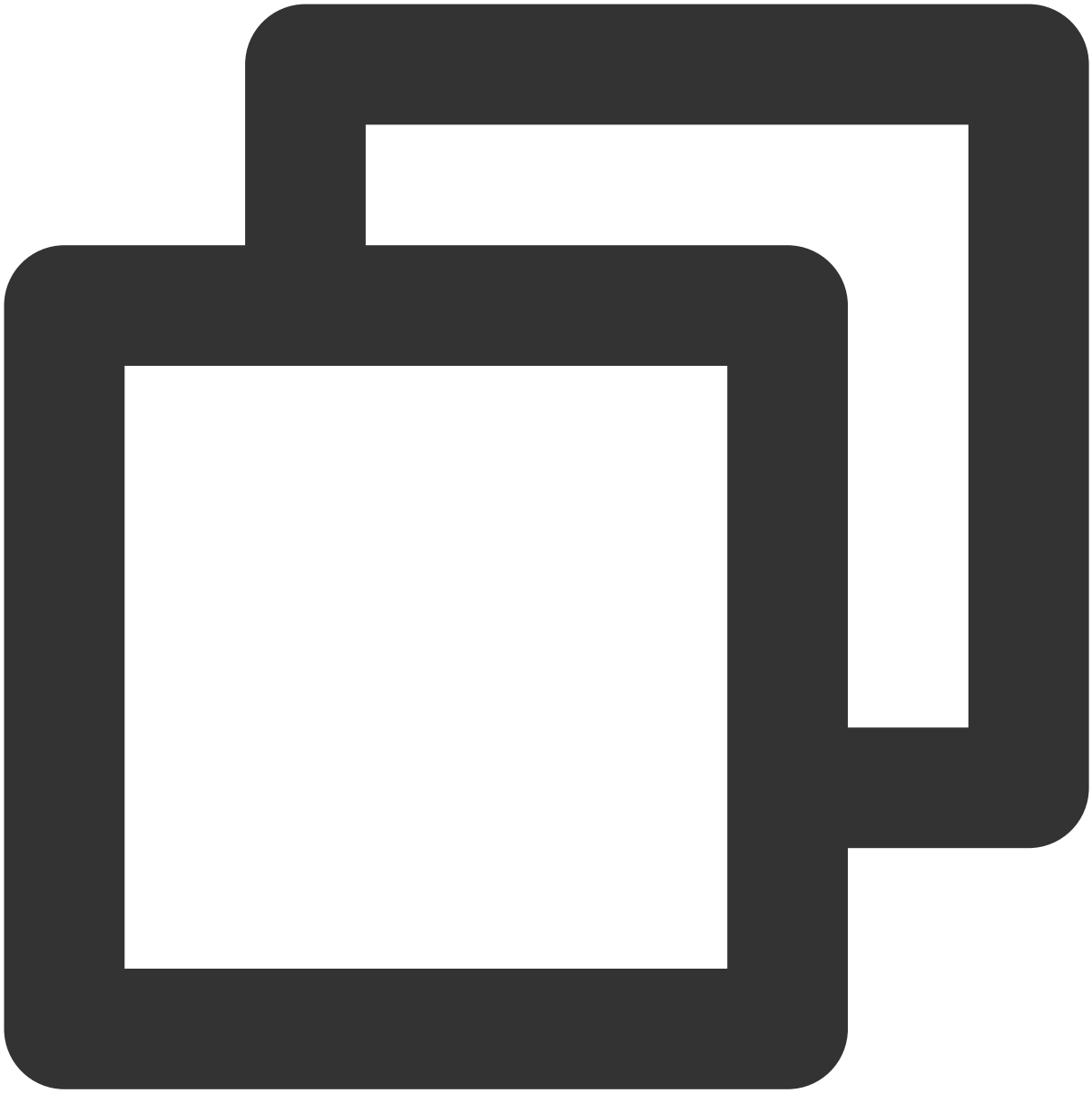
Gradle 自动加载（AAR）

Gradle 手动下载（AAR）

集成 SDK（jar+so）

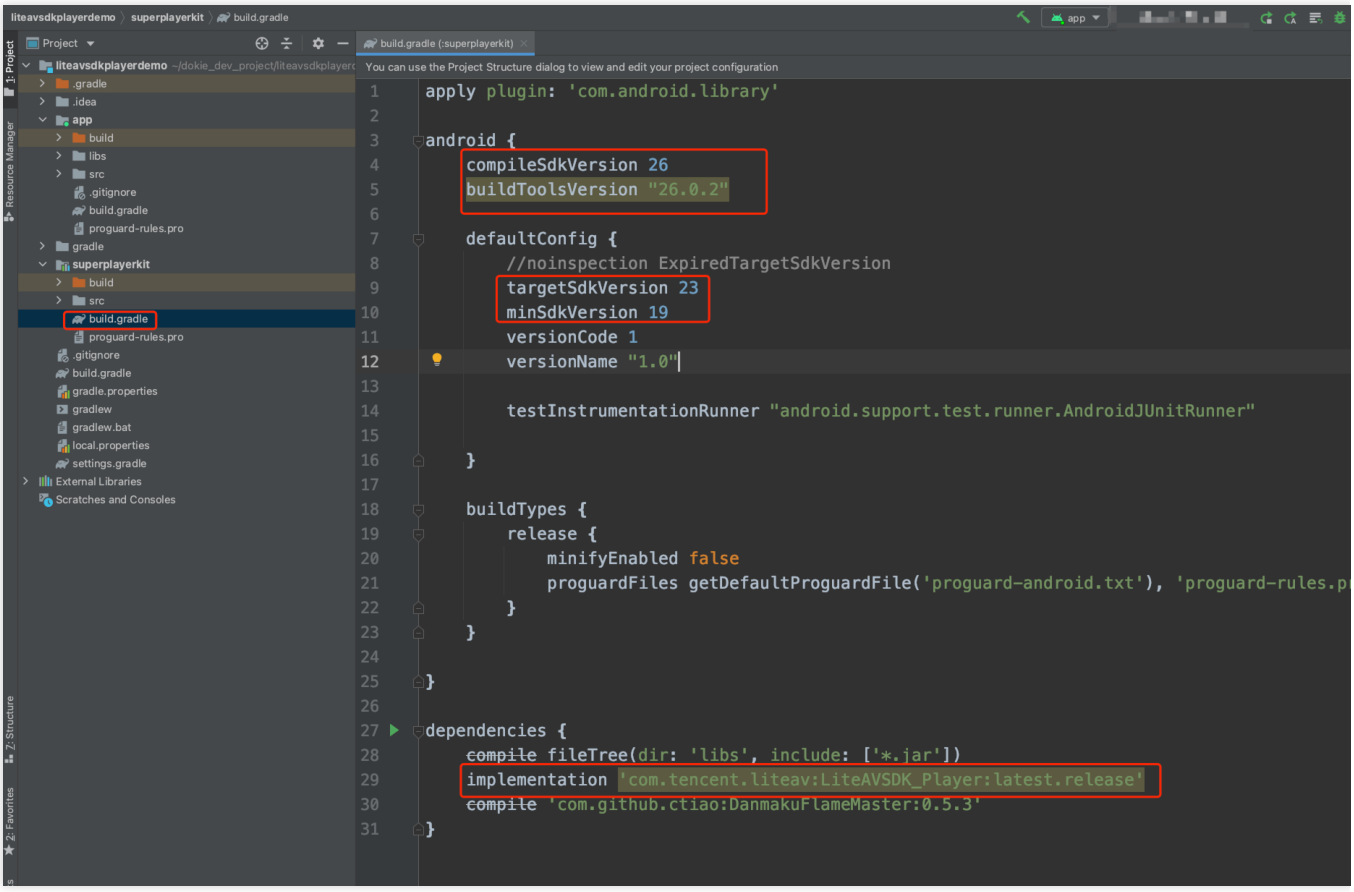
1. 下载 SDK + Demo 开发包，项目地址为 Android。
2. 把 `Demo/superplayerkit` 这个 module 复制到工程中，然后进行下面的配置：

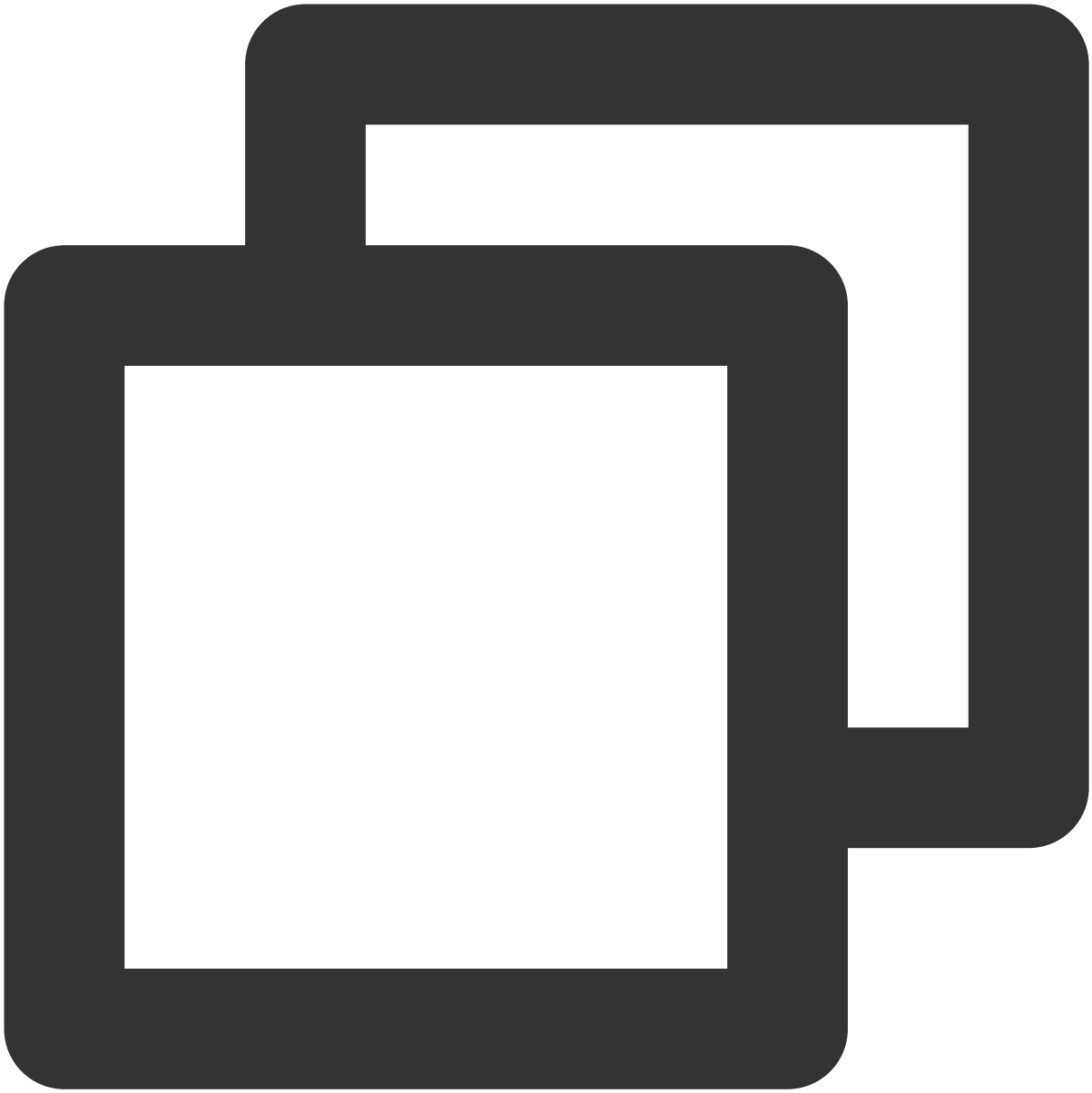
在工程目录下的 setting.gradle 导入 `superplayerkit` 。



```
include ':superplayerkit'
```

打开 `superplayerkit` 工程的 build.gradle 文件修改 compileSdkVersion，buildToolsVersion，minSdkVersion，targetSdkVersion 和 rootProject.ext.liteavSdk 的常量值。


liteavsdkplayerdemo
superplayerkit
build.gradle
Project
build.gradle (:superplayerkit)
You can use the Project Structure dialog to view and edit your project configuration
apply plugin: 'com.android.library'
android {
compileSdkVersion 26
buildToolsVersion "26.0.2"
defaultConfig {
//noinspection ExpiredTargetSdkVersion
targetSdkVersion 23
minSdkVersion 19
versionCode 1
versionName "1.0"
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.p
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
compile 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
}

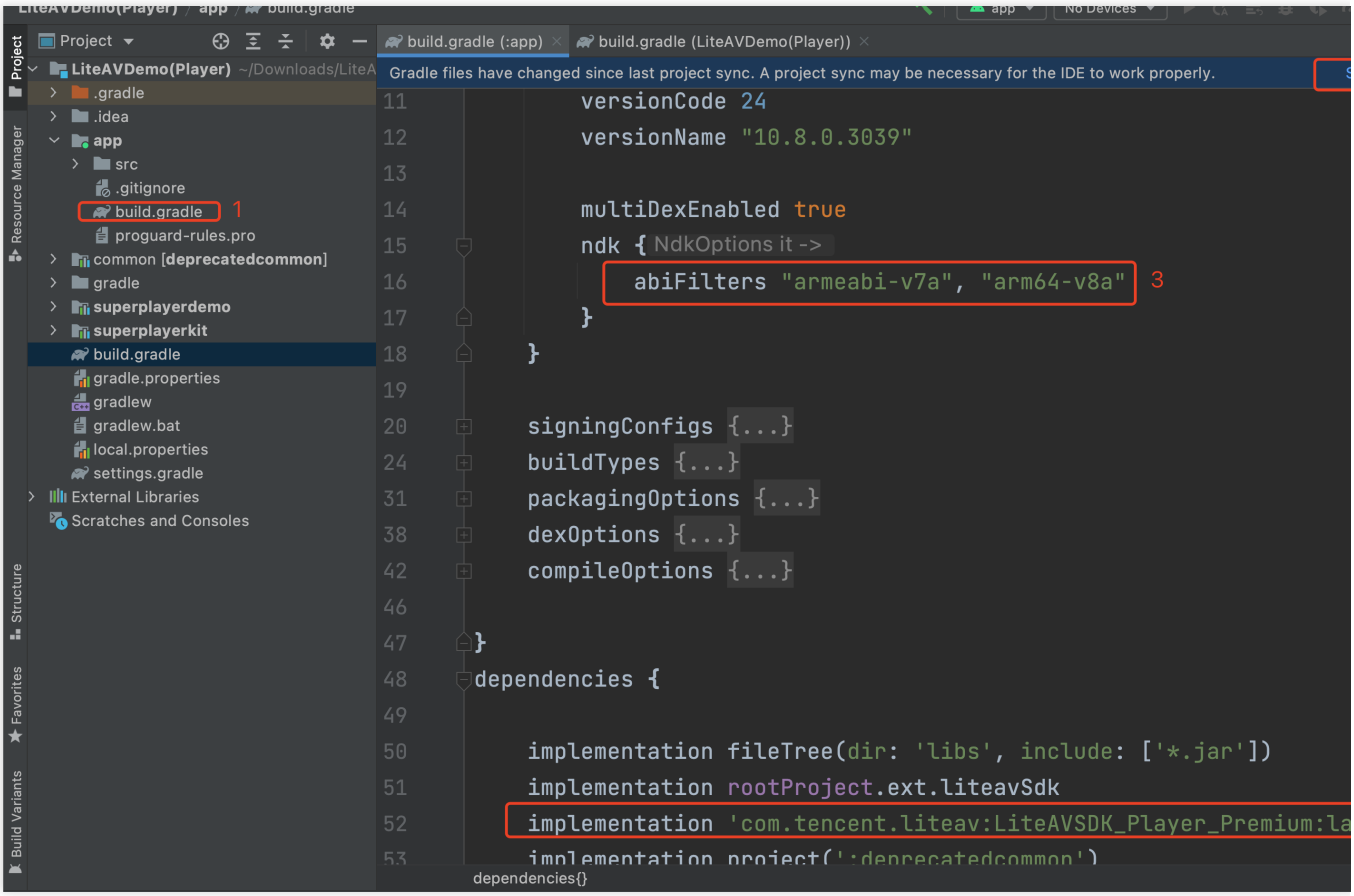


```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  //如果要集成播放器高级版历史版本，可将 latest.release 修改为对应的版本，例如：10.8.0.29000
  implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
```

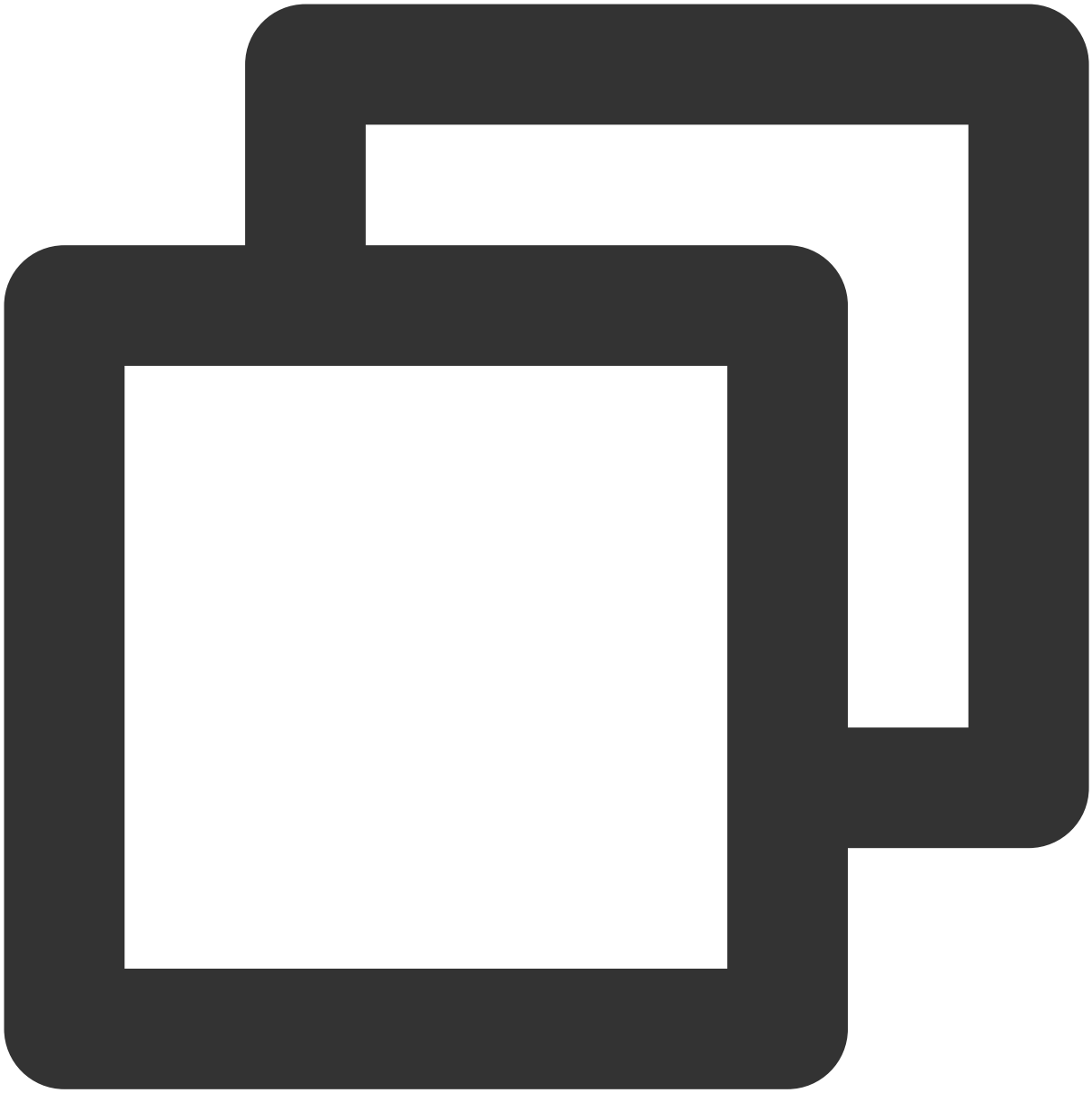
```
    // 如果要集成播放器基础版
    // implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

请参见上面的步骤，把 `common` 模块导入到项目，并进行配置。

3. 通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK，打开 `app/build.gradle` ，进行下面的配置：

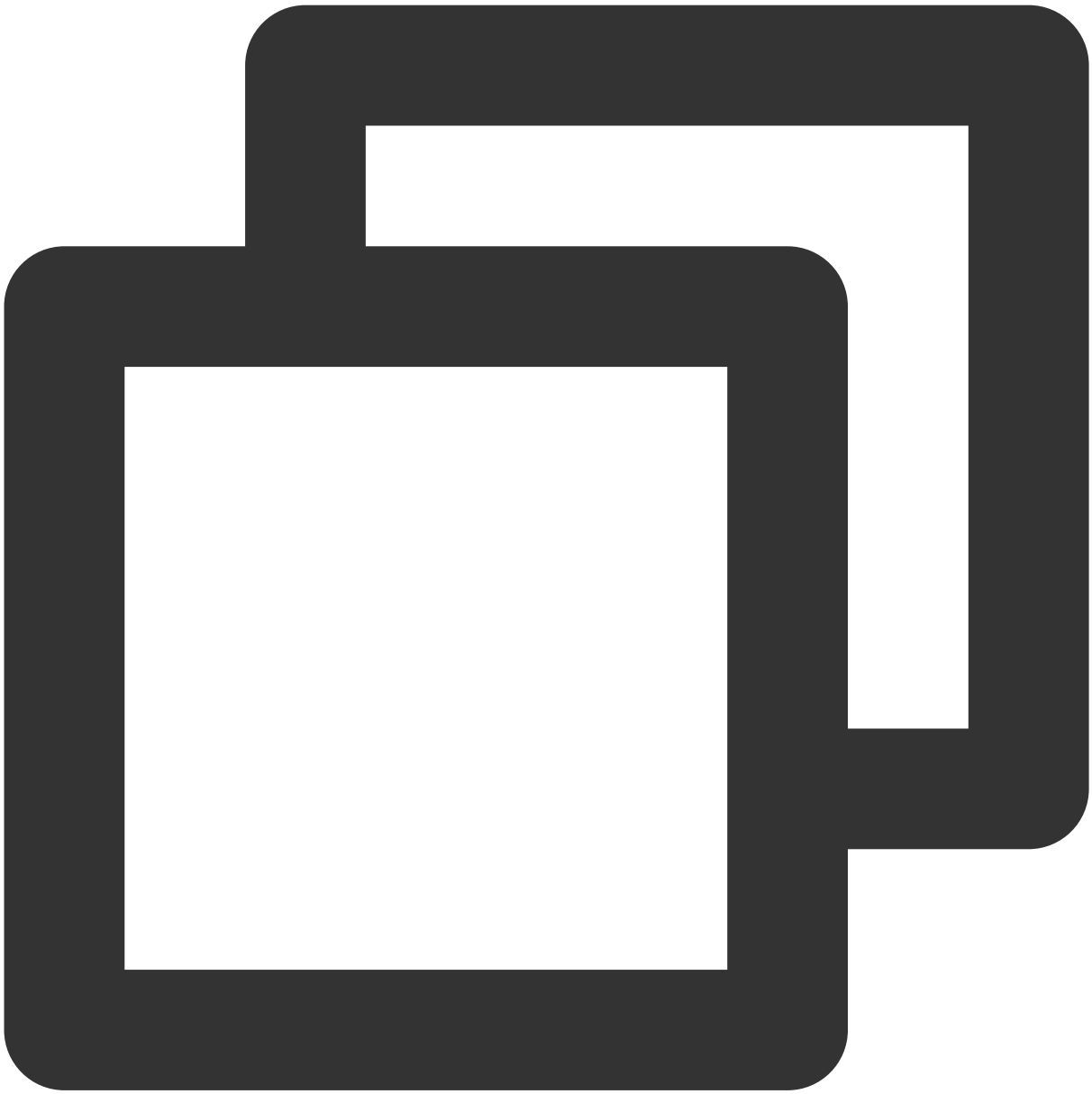


3.1 在 dependencies 中添加 LiteAVSDK_Player_Premium 的依赖。



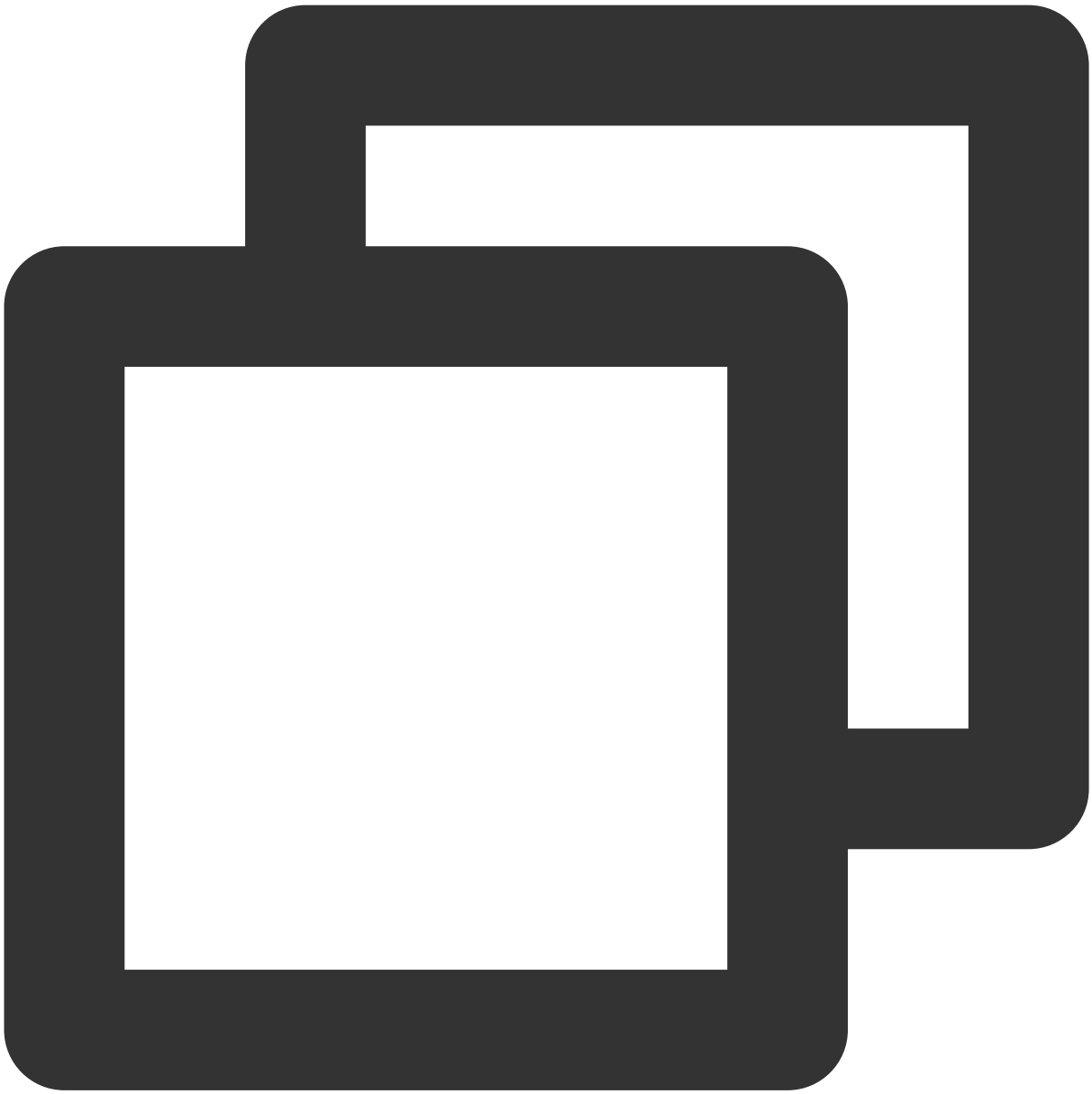
```
dependencies {
     implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
      // 如果要集成播放器基础版
      // implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
     implementation project(':superplayerkit')
     // 播放器组件弹幕集成的第三方库
     implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
}
```

如果您需要集成历史版本的 LiteAVSDK_Player_Premium SDK，可以在 MavenCentral 查看历史版本，然后通过下面的方式进行集成：



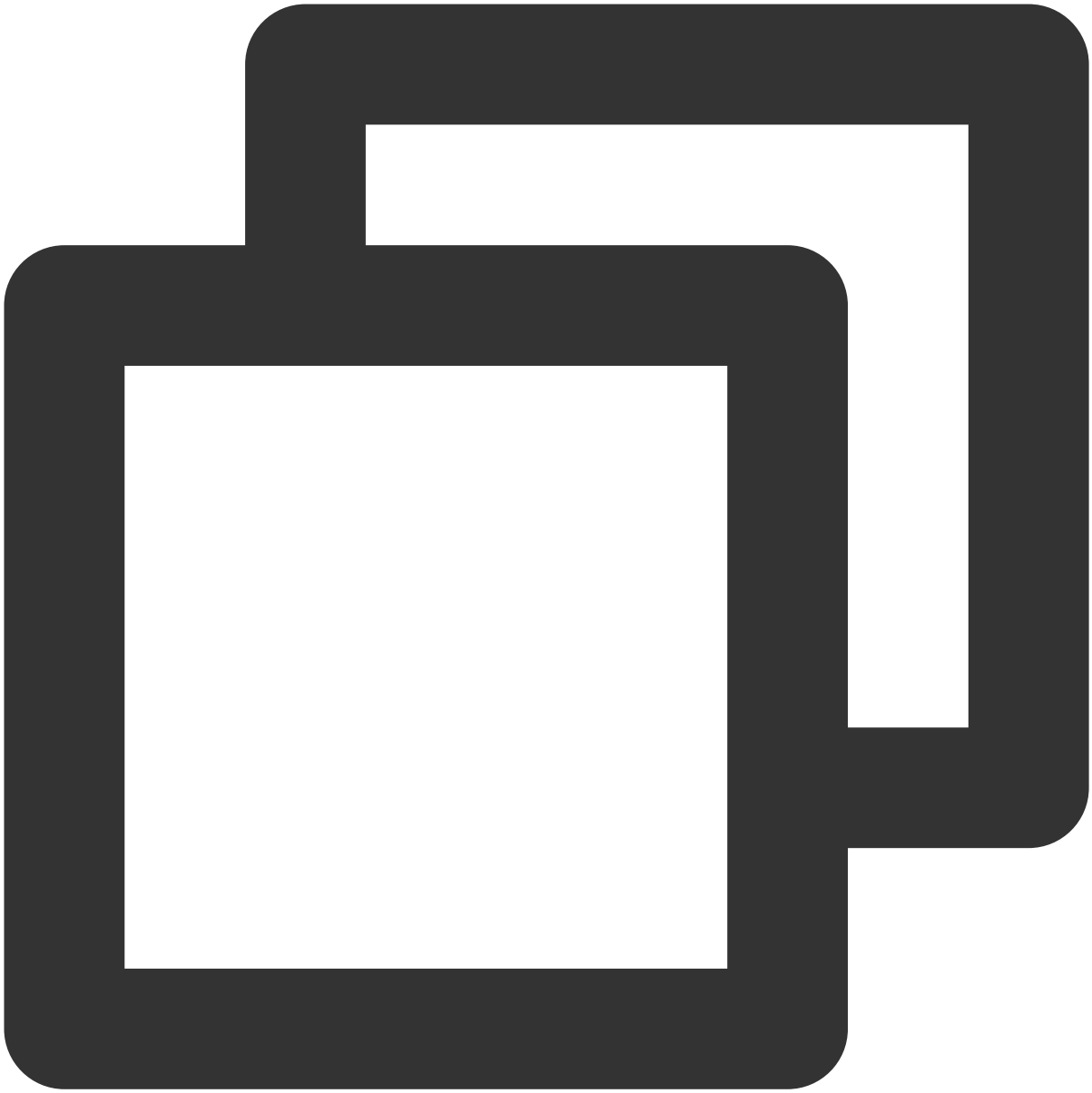
```
dependencies {
    // 集成 10.8.0.29000 版本 LiteAVSDK_Player_Premium SDK
    implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:10.8.0.29000'
    // 如果要集成播放器基础版
    // implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

4. 在 `app/build.gradle` defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi 、armeabi-v7a 和 arm64-v8a，可根据项目需求配置）。



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

5. 在工程目录的 build.gradle 添加 mavenCentral 库。



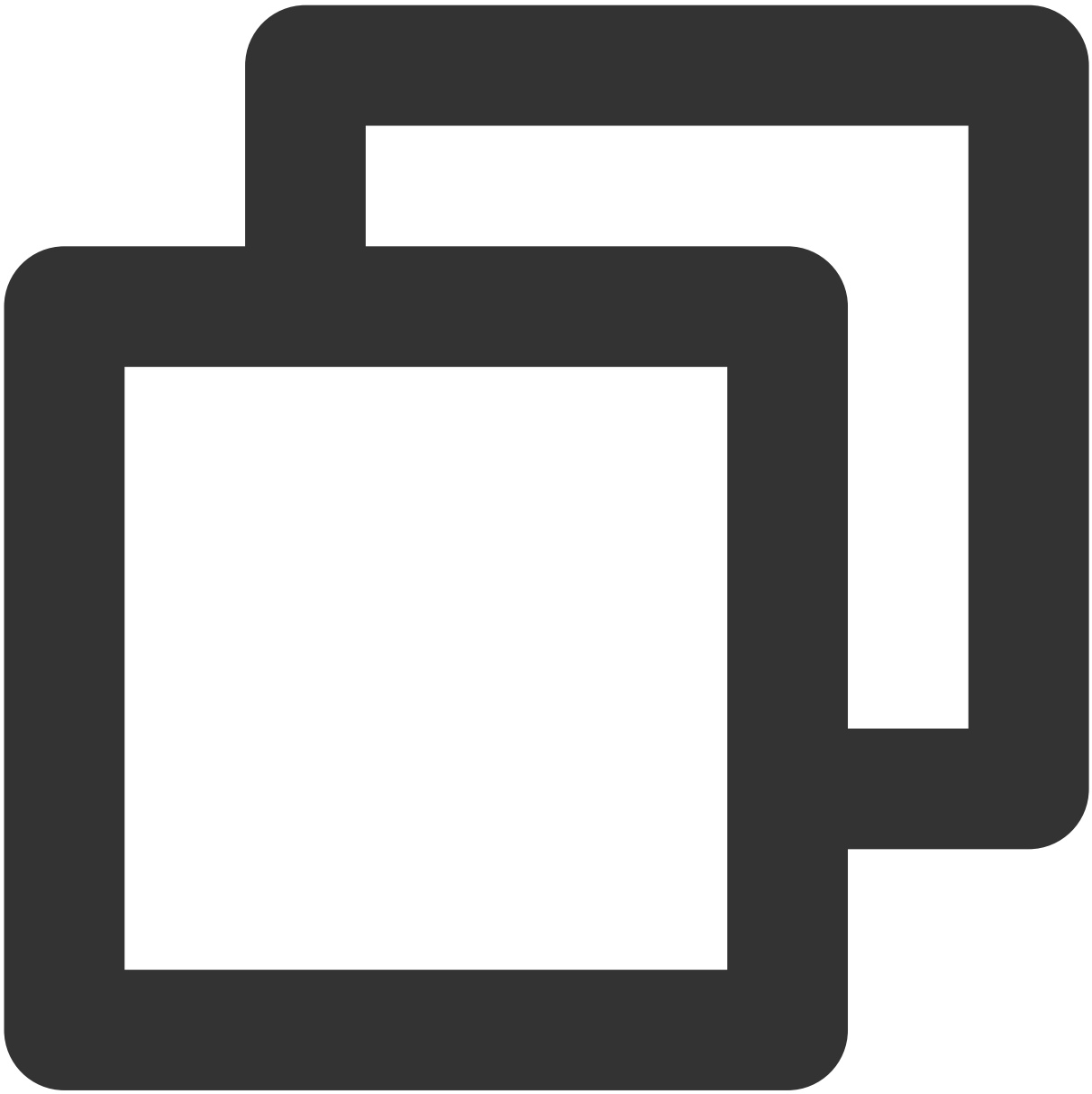
```
repositories {
  mavenCentral()
}
```

6. 单击

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

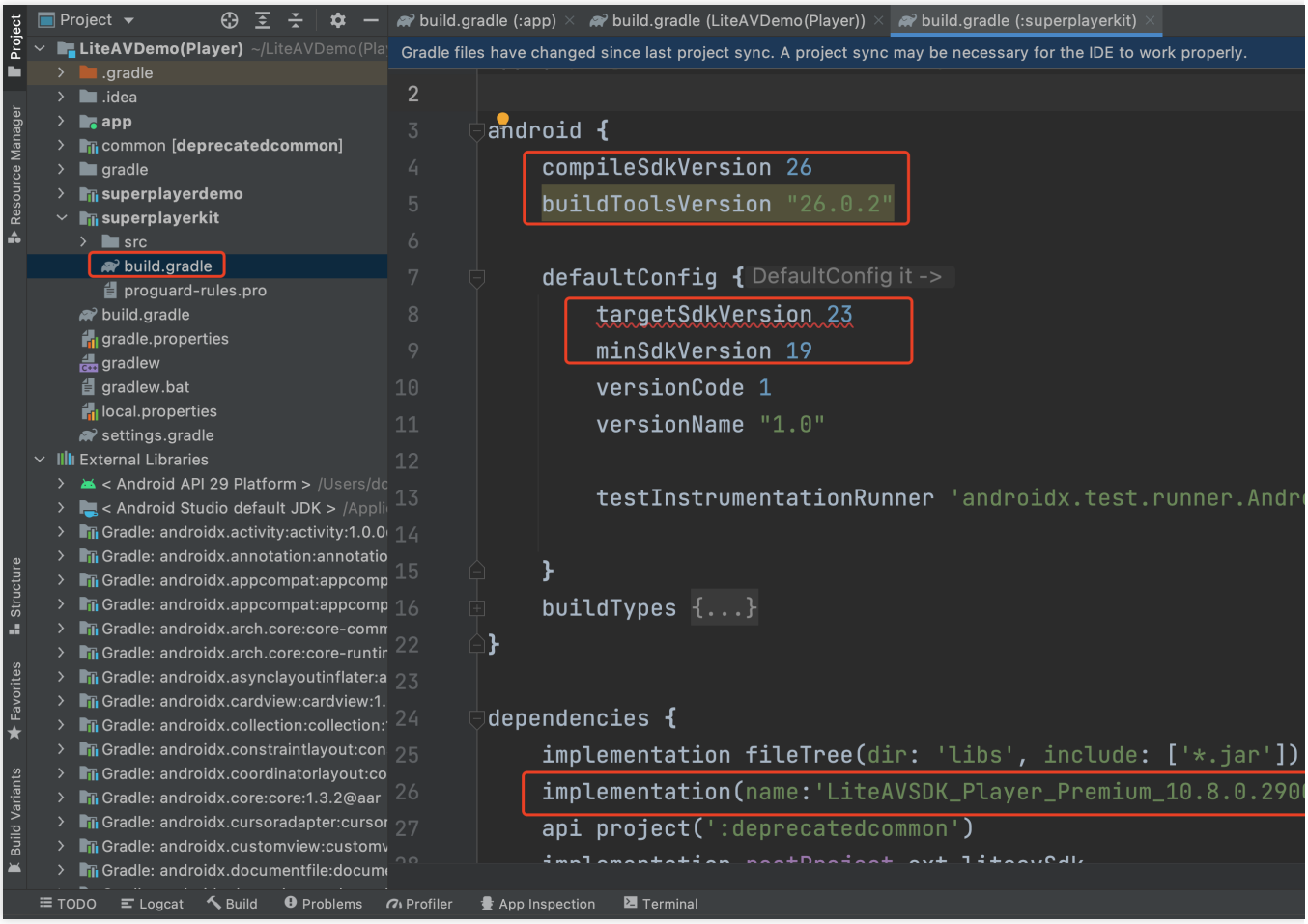
1. 下载 SDK + Demo 开发包，项目地址为 Android。

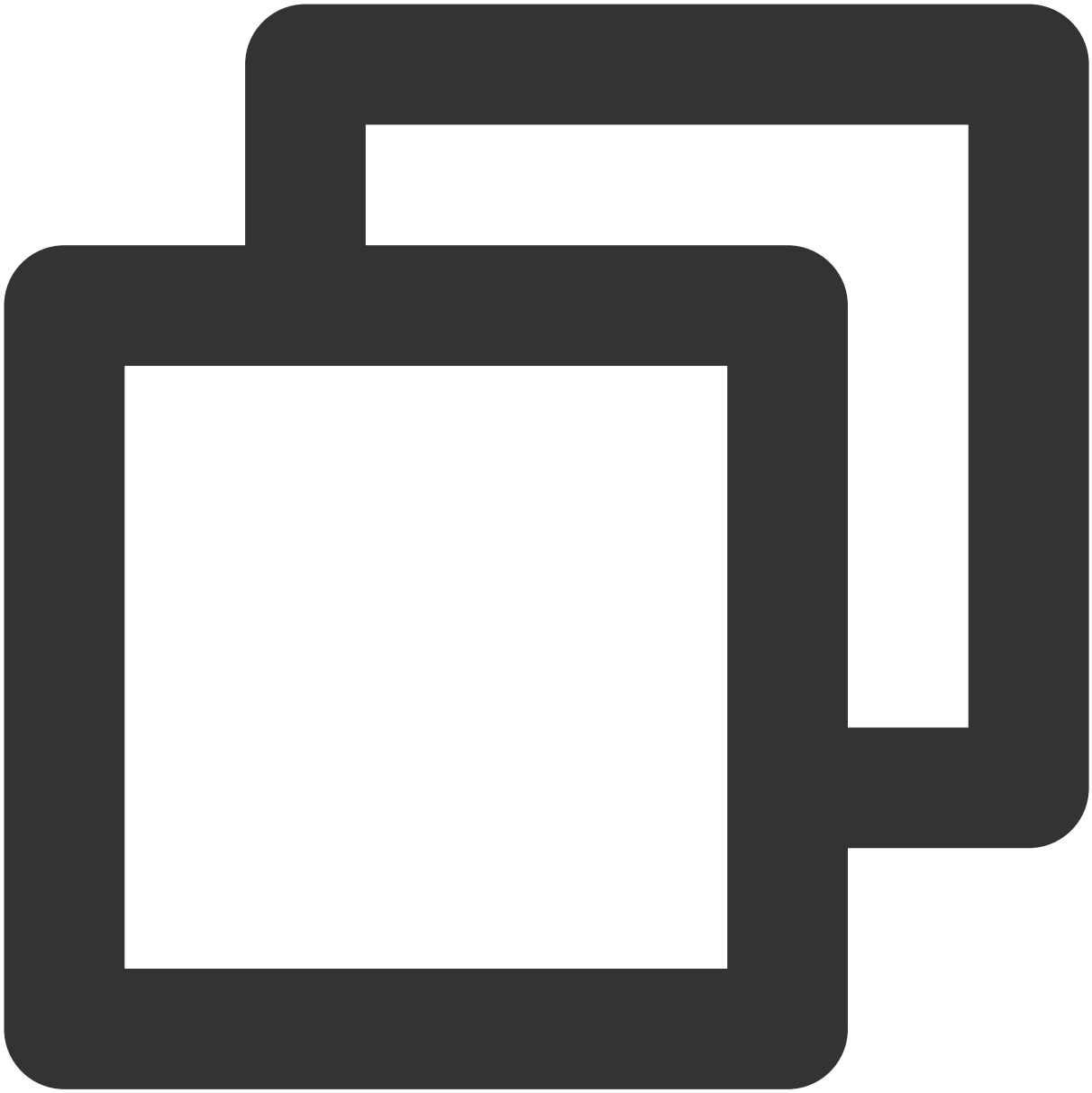
2. 导入 `SDK/LiteAVSDK_Player_Premium_XXX.aar`（其中 XXX 为版本号）到 app 下面的 libs 文件夹以及复制 `Demo/superplayerkit` 这个 module 到工程中。
3. 在工程目录下的 setting.gradle 导入 `superplayerkit` 。



```
include ':superplayerkit'
```

4. 打开 `superplayerkit` 工程的 build.gradle 文件修改 compileSdkVersion，buildToolsVersion，minSdkVersion，targetSdkVersion 和 rootProject.ext.liteavSdk 的常量值。

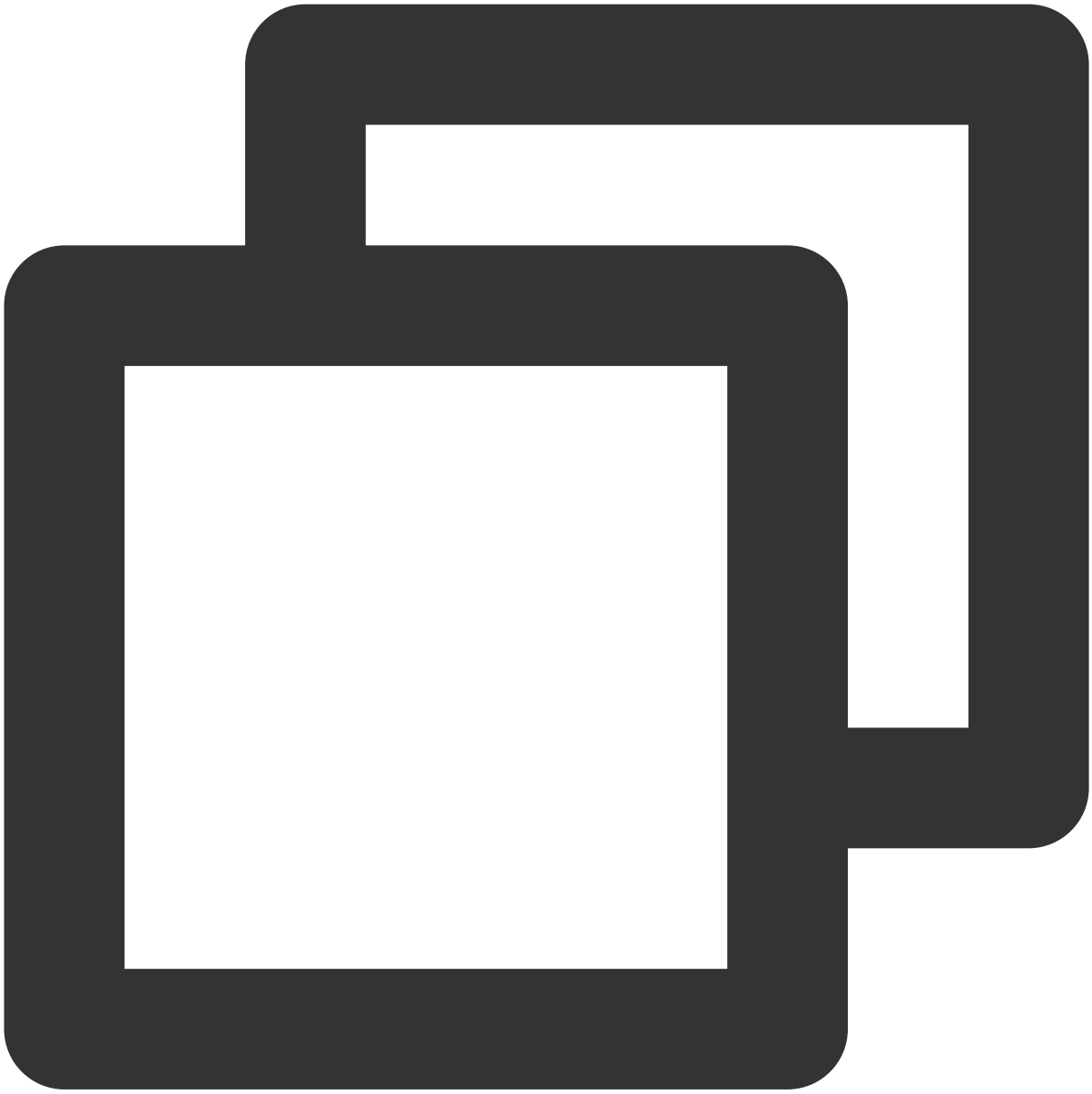

Project
LiteAVDemo(Player) ~/LiteAVDemo(Pla
.gradle
.idea
app
common [deprecatedcommon]
gradle
superplayerdemo
superplayerkit
src
build.gradle
proguard-rules.pro
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries
build.gradle (:app)
build.gradle (LiteAVDemo(Player))
build.gradle (:superplayerkit)
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
android {
compileSdkVersion 26
buildToolsVersion "26.0.2"
defaultConfig { DefaultConfig it ->
targetSdkVersion 23
minSdkVersion 19
versionCode 1
versionName "1.0"
testInstrumentationRunner 'androidx.test.runner.Andr
}
buildTypes {...}
}
dependencies {
implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation(name:'LiteAVSDK_Player_Premium_10.8.0.290
api project(':deprecatedcommon')
TODO
Logcat
Build
Problems
Profiler
App Inspection
Terminal



```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  implementation(name:'LiteAVSDK_Player_Premium_10.8.0.29000', ext:'aar')
}
```

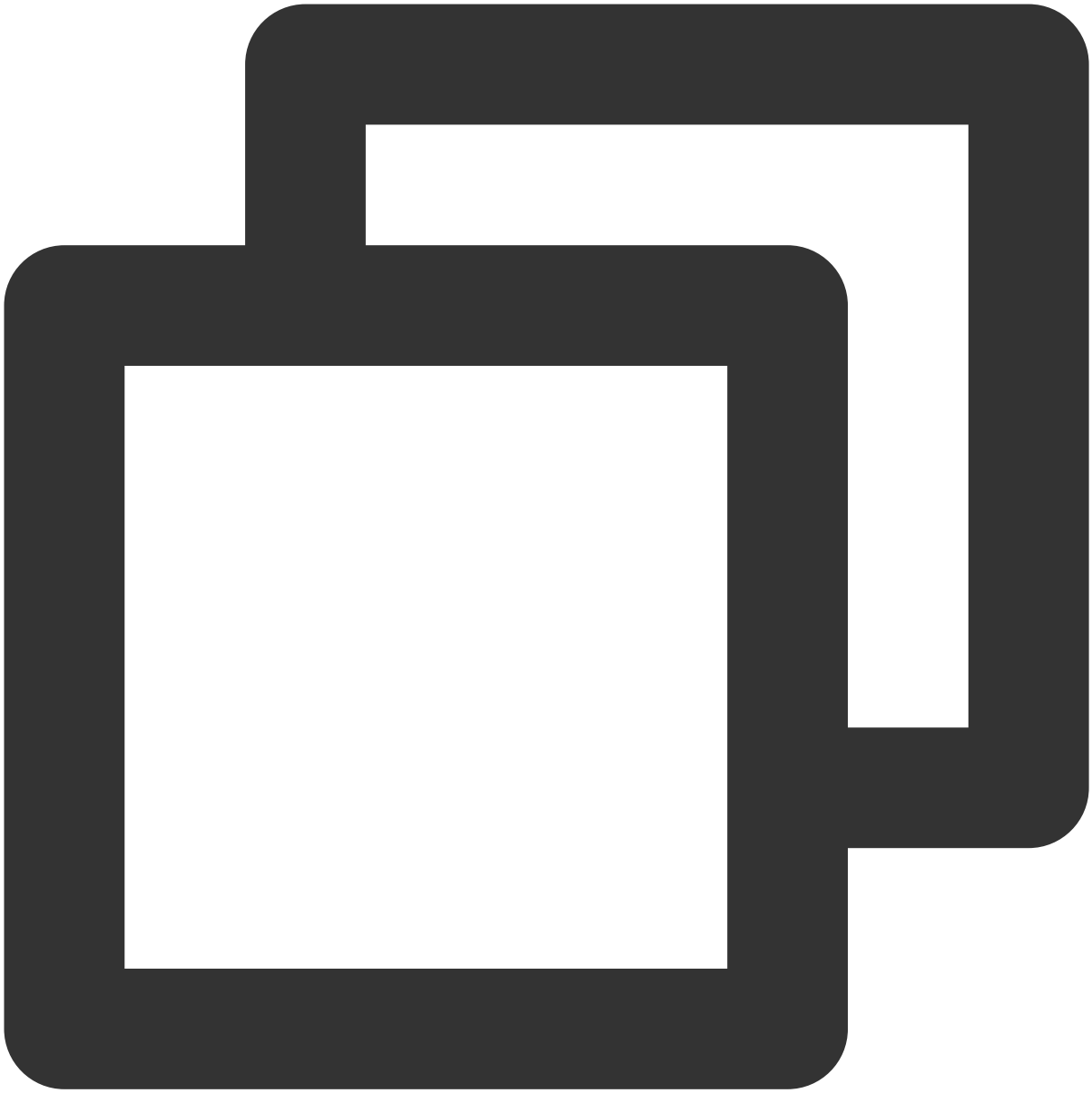
请参见上面的步骤，把 `common` 模块导入到项目，并进行配置。

配置 repositories。



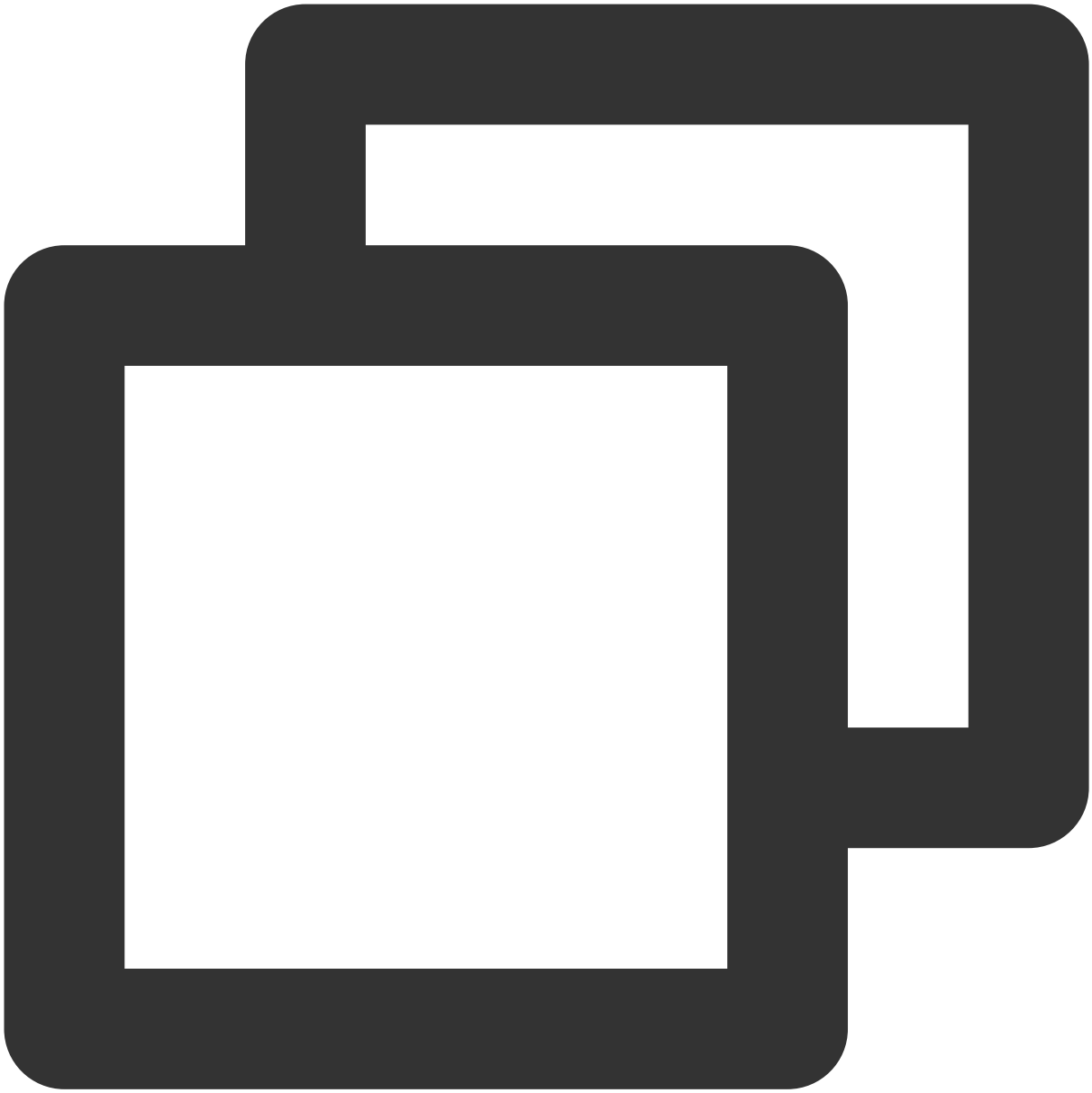
```
repositories {
flatDir {
        dirs '../app/libs'
}
}
```

5. 在 `app/build.gradle` 中添加依赖：



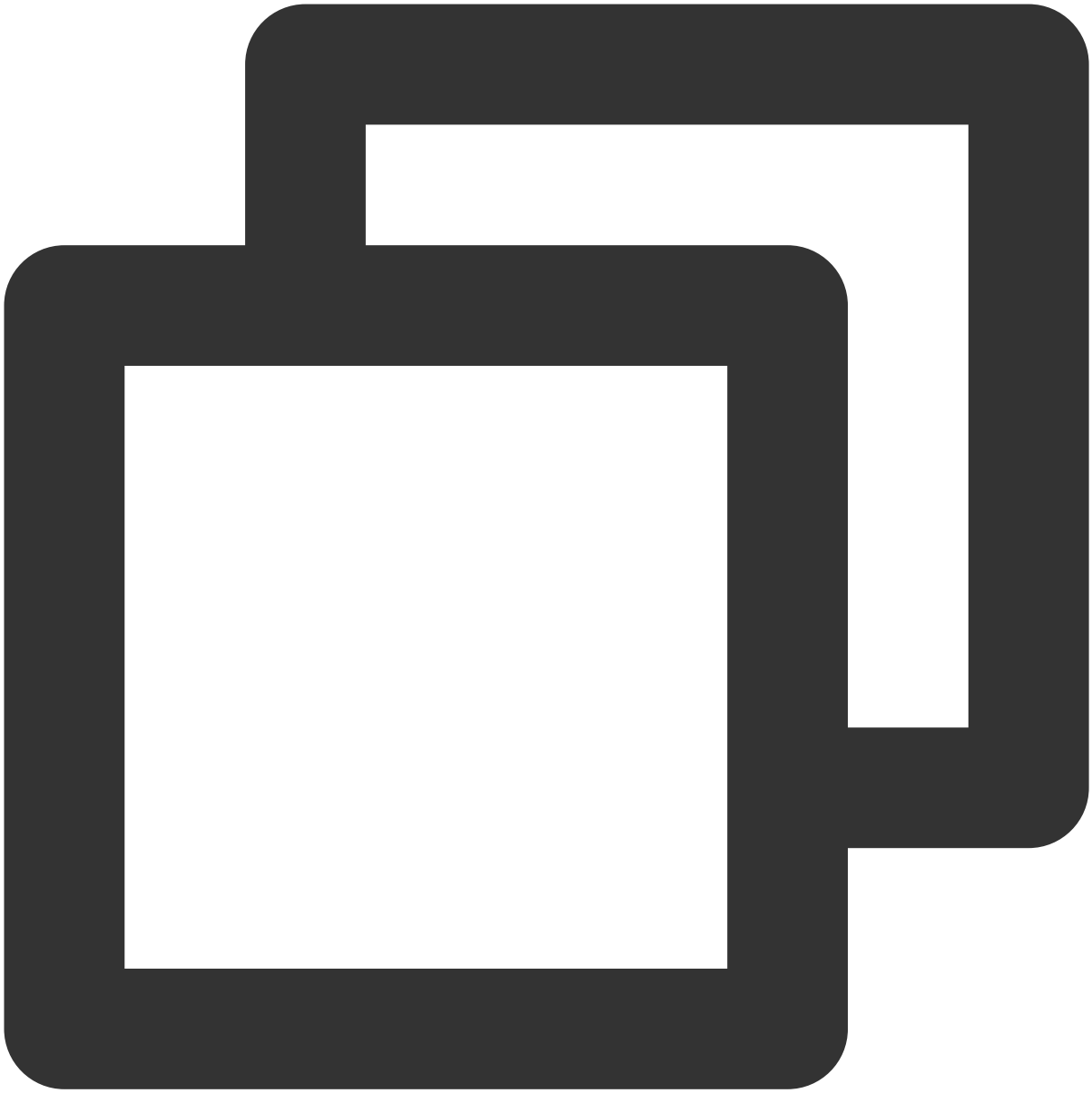
```
compile(name:'LiteAVSDK_Player_Premium_10.8.0.29000', ext:'aar')
implementation project(':superplayerkit')
// 播放器组件弹幕集成的第三方库
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
```

6. 在项目 `build.gradle` 中添加：



```
allprojects {
  repositories {
          flatDir {
                  dirs 'libs'
          }
  }
}
```

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

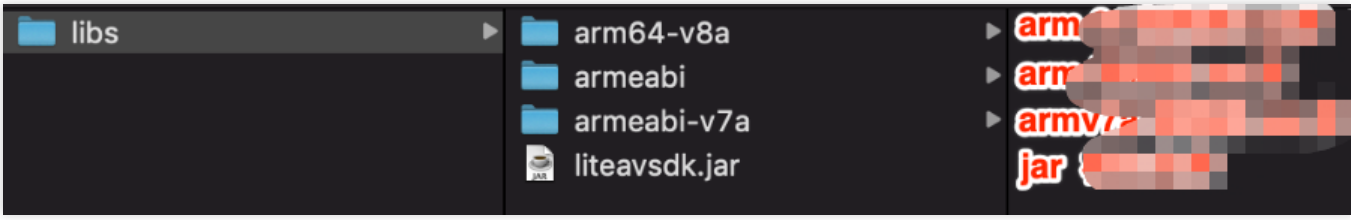


```
ndk {
  abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

8. 单击 Sync Now 按钮同步 SDK，完成播放器组件的集成工作。

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 SDK + Demo 开发包，项目地址为 Android，下载完成后进行解压。在 SDK 目录找到 SDK/LiteAVSDK_Player_Premium_XXX.zip（其中 XXX 为版本号），解压得到 libs 目录，里面包含 jar 文件和 so 文件夹，文件清单如下：

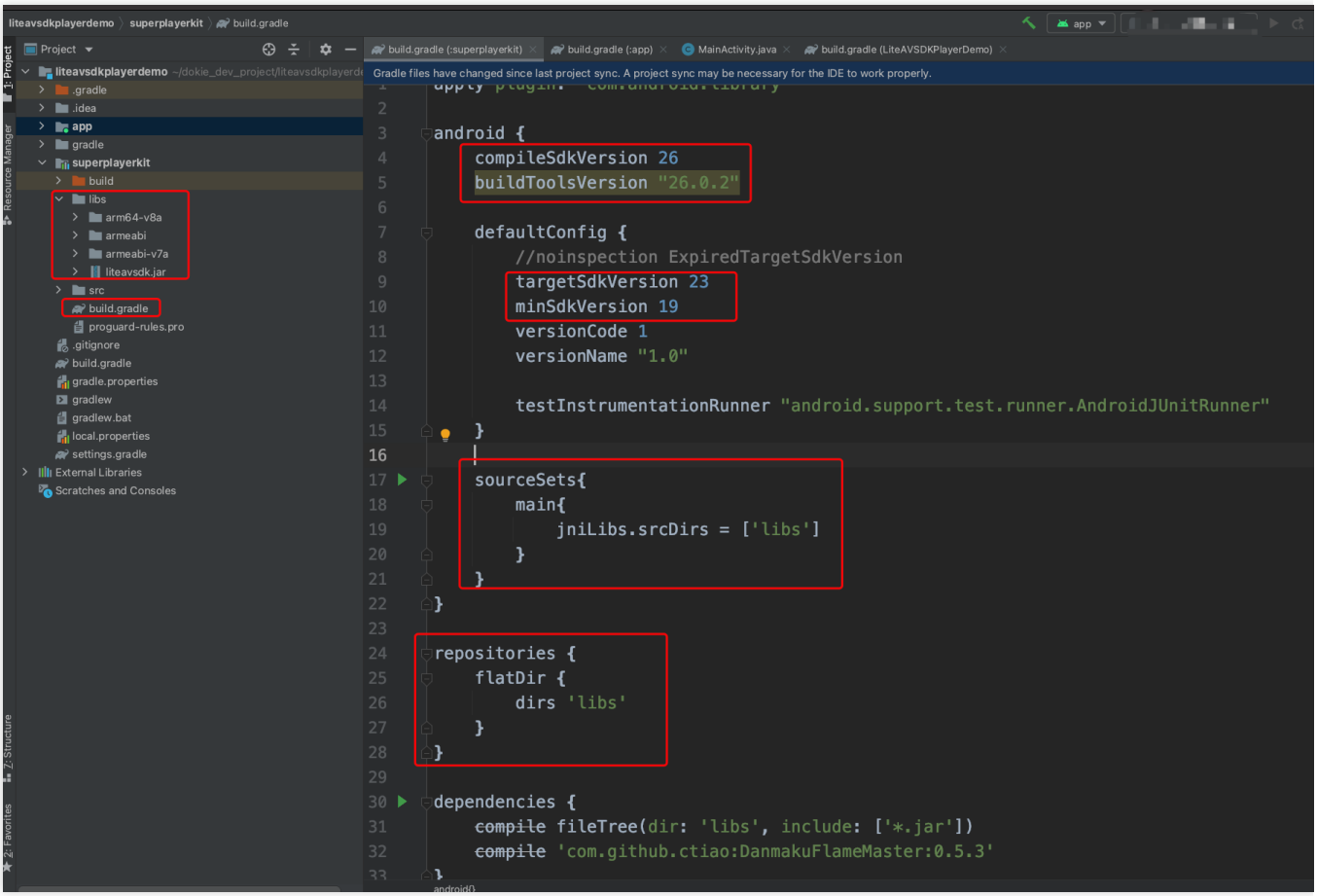


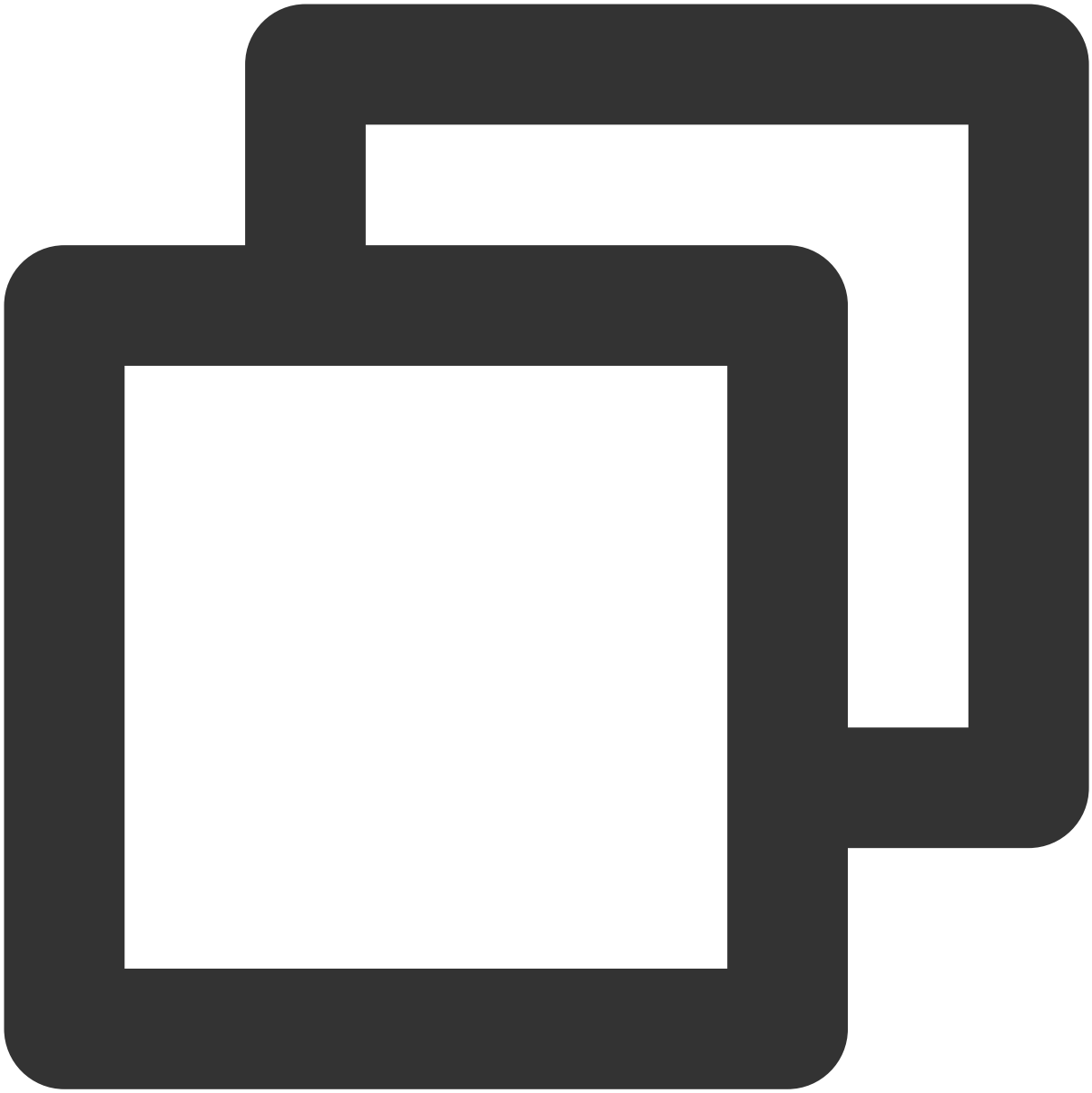
2. 把 `Demo/superplayerkit` 这个 module 复制到工程中，然后在工程目录下的 setting.gradle 导入 `superplayerkit` 。

```
include ':superplayerkit'
```

3. 把 步骤1 解压得到的 libs 文件夹复制 `superplayerkit` 工程根目录。

4. 修改 `superplayerkit/build.gradle` 文件：

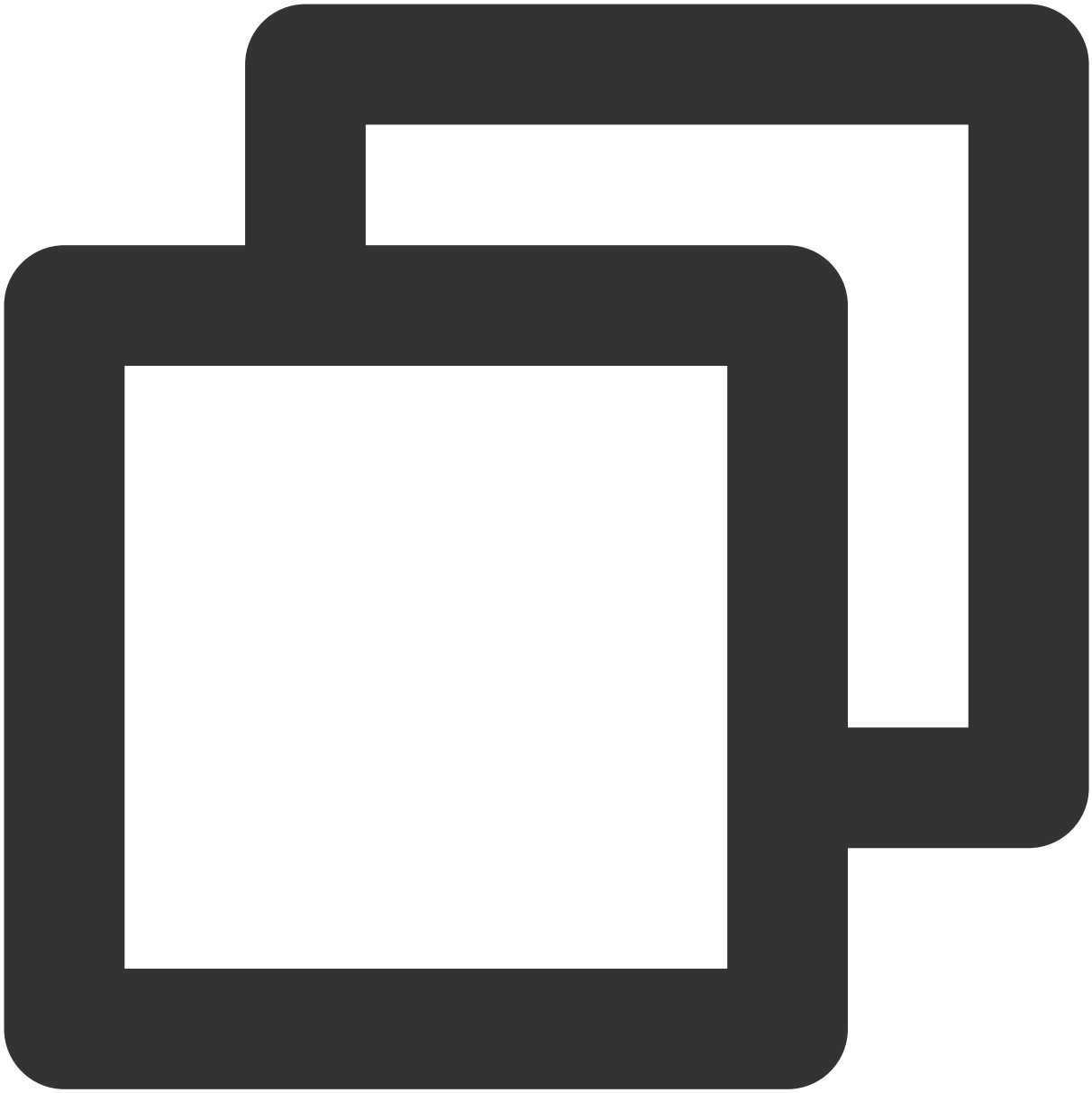




```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}
```

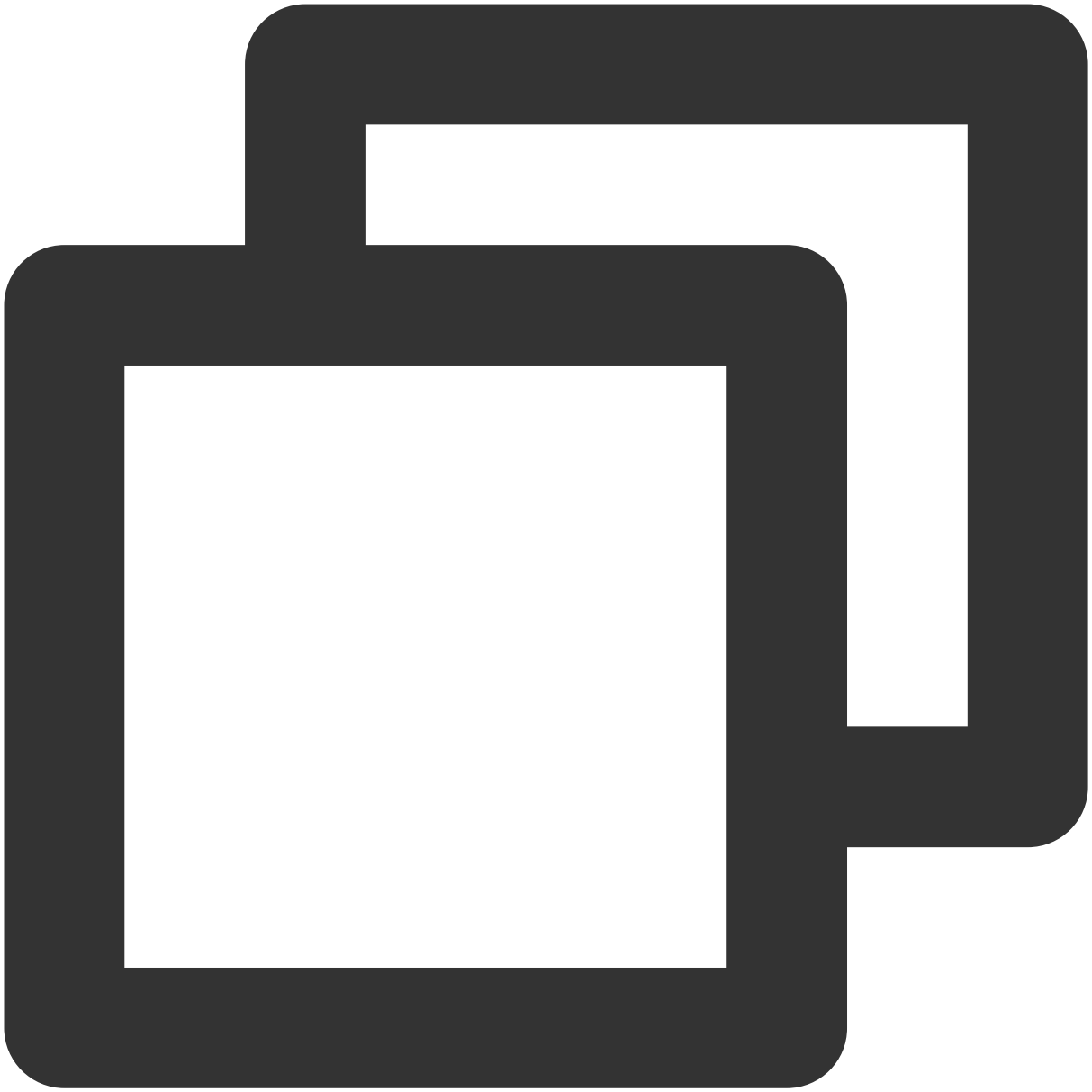
请参见上面的步骤，把 `common` 模块导入到项目，并进行配置。

配置 sourceSets，添加 so 库引用代码。



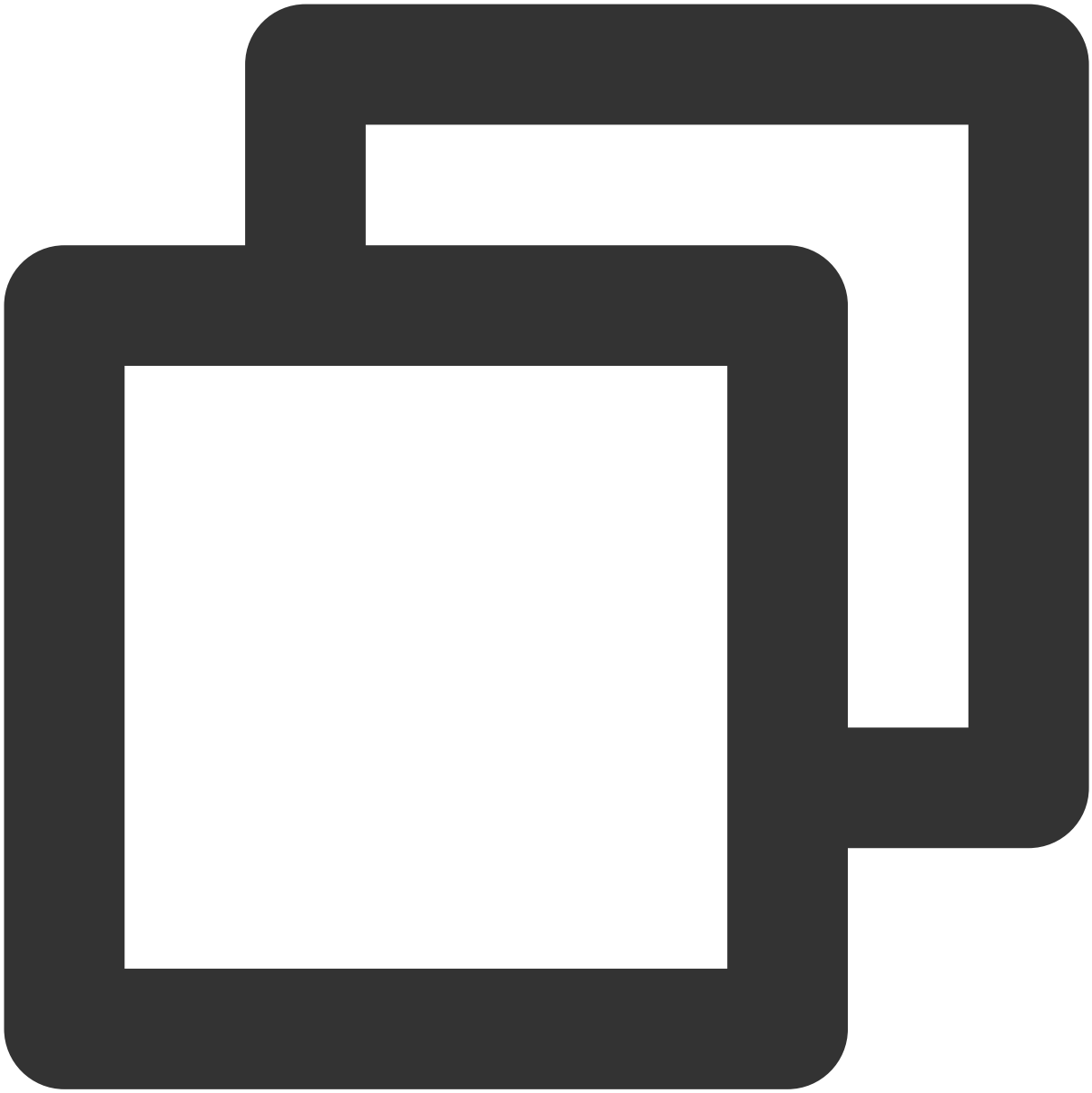
```
sourceSets{
    main{
            jniLibs.srcDirs = ['libs']
    }
 }
```

配置 repositories，添加 flatDir，指定本地仓库路径。



```
repositories {
flatDir {
        dirs 'libs'
}
}
```

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



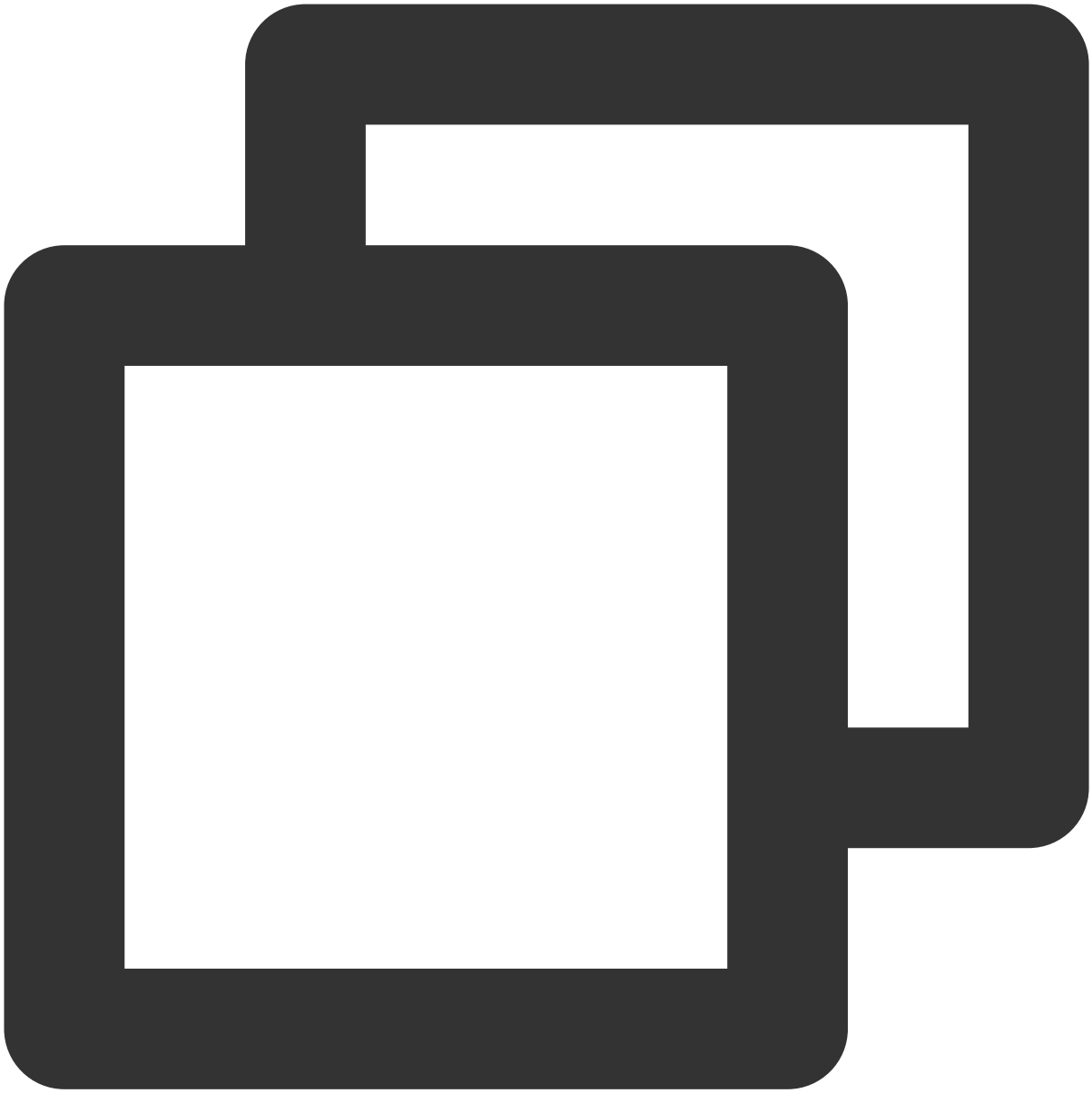
```
ndk {
  abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

6. 单击 Sync Now 按钮同步 SDK，完成 播放器组件的集成工作。

您已经完成了腾讯云 Android 播放器组件项目集成的步骤。

步骤3：配置 App 权限

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

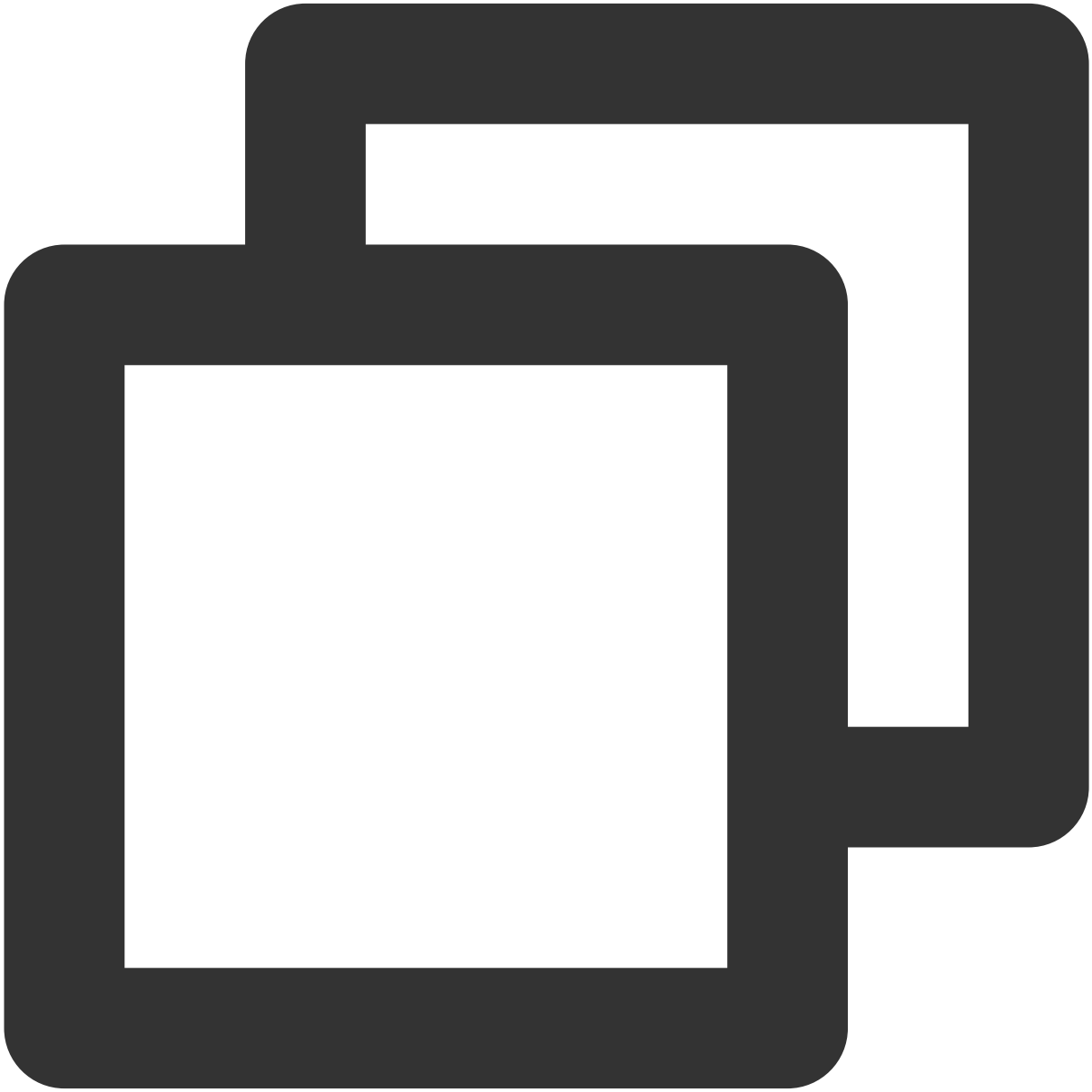


```
<!--网络权限-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--点播播放器悬浮窗权限-->
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

网络安全配置允许 App 发送 http 请求

出于安全考虑，从 Android P 开始，Google 要求 App 的请求都使用加密链接。播放器 SDK 会启动一个 localsever 代理 http 请求，如果您的应用 **targetSdkVersion 大于或等于28**，可以通过 网络安全配置 来开启允许向127.0.0.1发送 http 请求。 否则播放时将出现 "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" 错误， 导致无法播放视频。配置步骤如下：

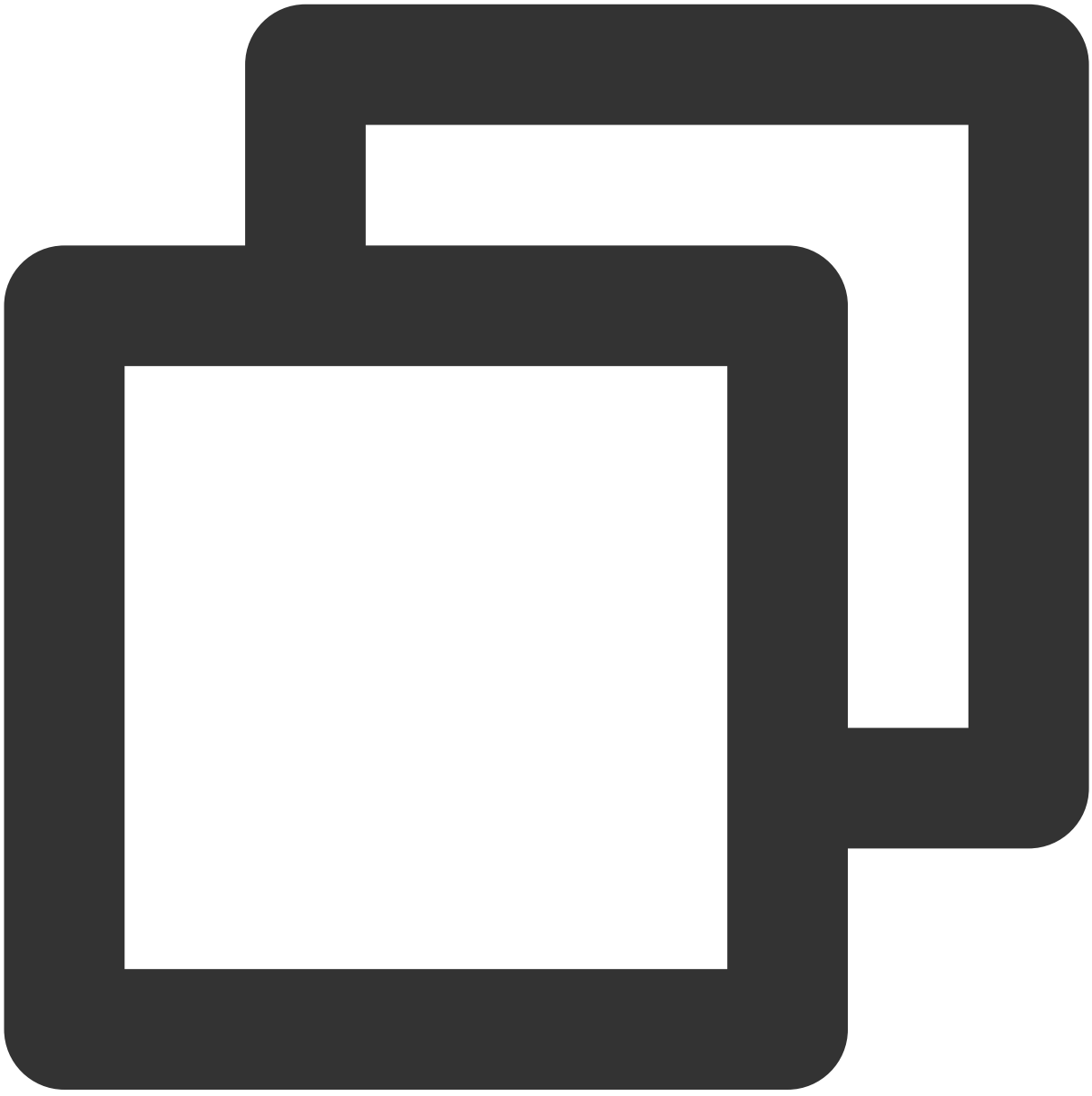
1. 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置。



```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
```

```
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

2. 在 AndroidManifest.xml 文件下的 application 标签增加以下属性。

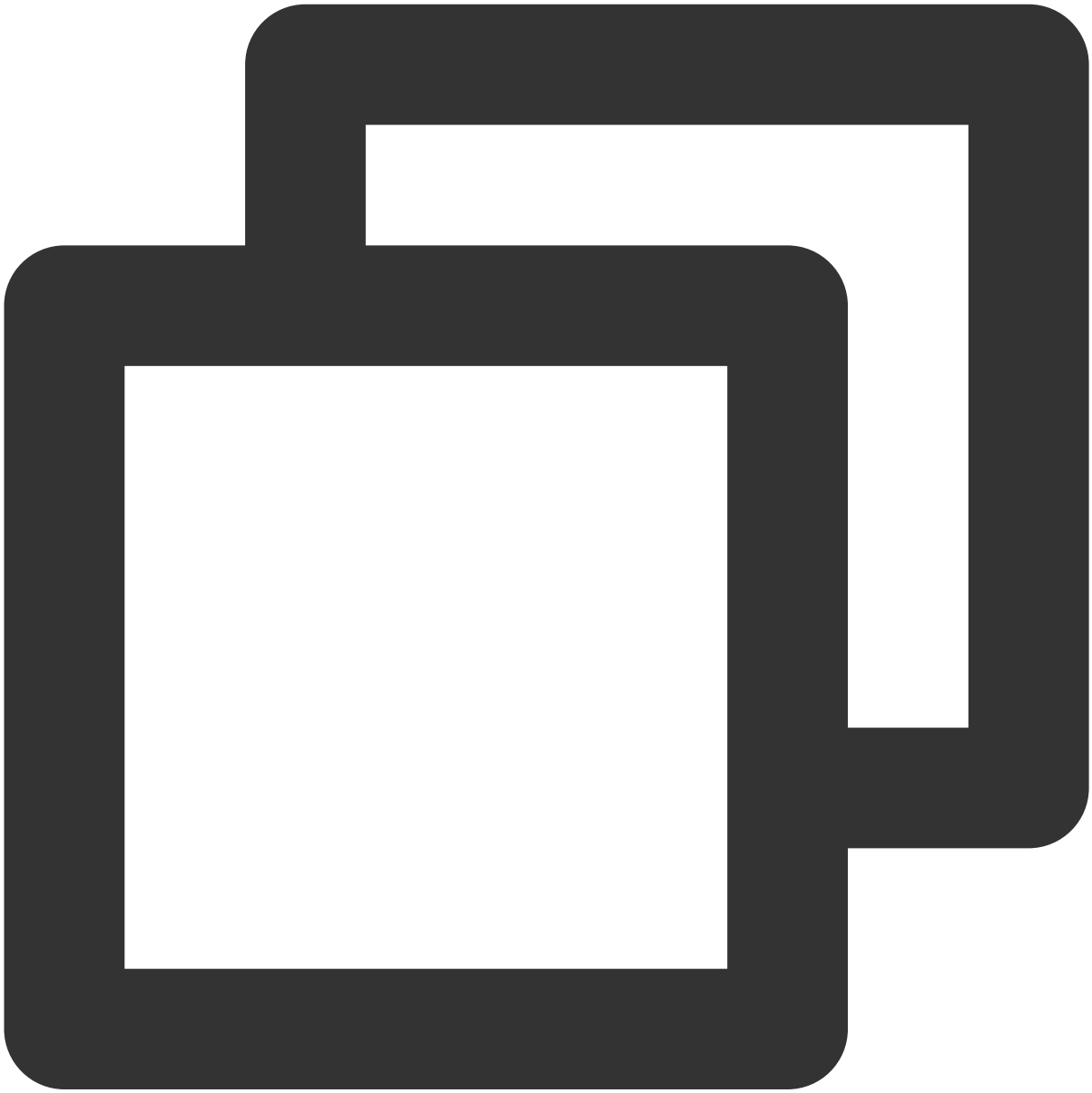


```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
```

```
</manifest>
```

步骤4：设置混淆规则

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



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

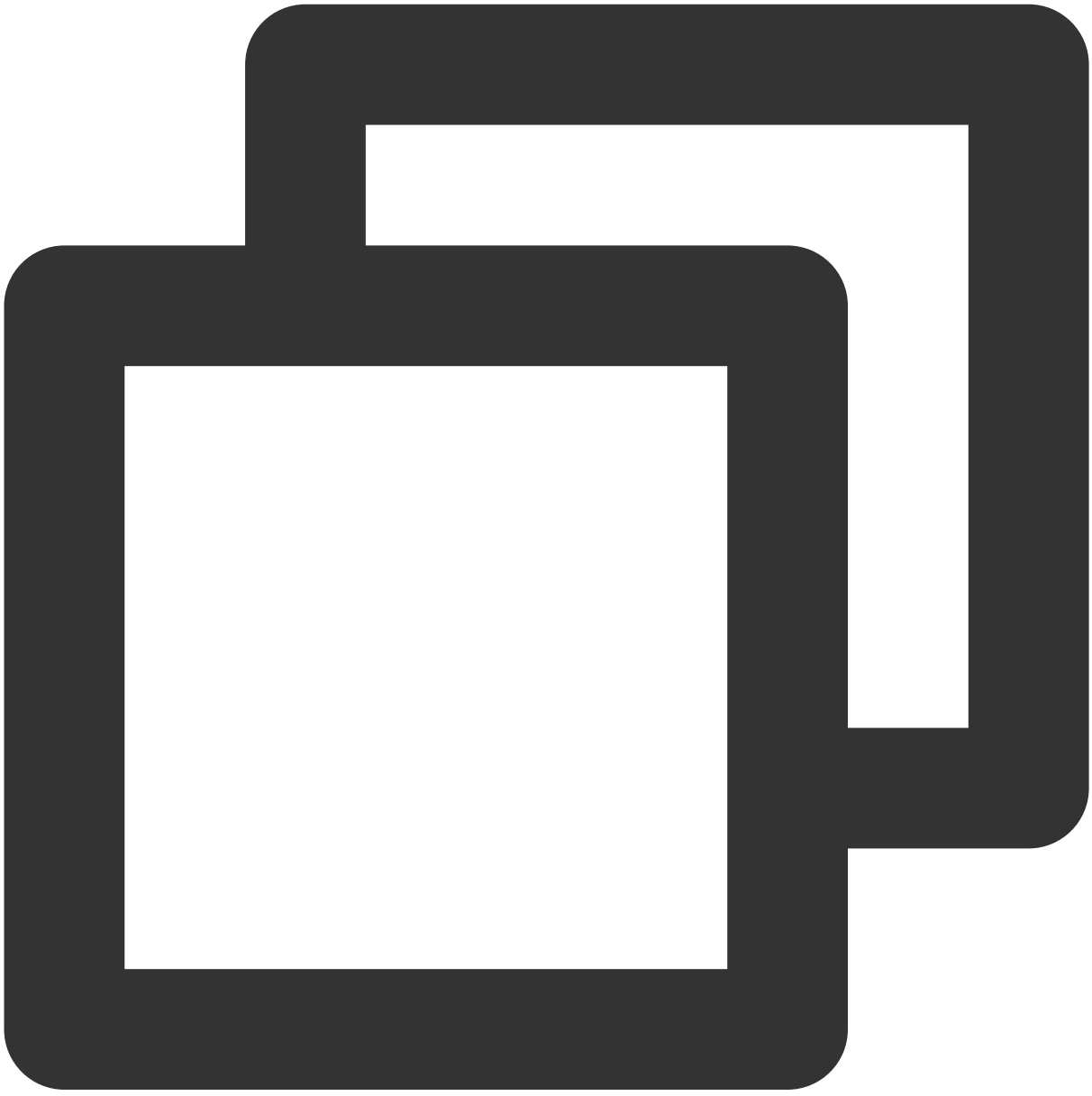
您已经完成了腾讯云视立方 Android 播放器组件 app 权限配置的步骤。

步骤5：使用播放器功能

本步骤，用于指导用户创建和使用播放器，并使用播放器进行视频播放。

1. 创建播放器

播放器主类为 `SuperPlayerView` ，创建后即可播放视频， 支持集成 FileID 或者 URL 进行播放。在布局文件创建 SuperPlayerView：



```
<!-- 播放器组件-->
<com.tencent.liteav.demo.superplayer.SuperPlayerView
 android:id="@+id/superVodPlayerView"
 android:layout_width="match_parent"
 android:layout_height="200dp" />
```

2. **配置 License 授权**

若您已获得相关 License 授权，需在 云点播控制台获取 License URL 和 License Key。

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

3. **播放视频**

本步骤用于指导用户播放视频。腾讯云视立方 Android 播放器组件可用于直播和点播两种播放场景，具体如下：

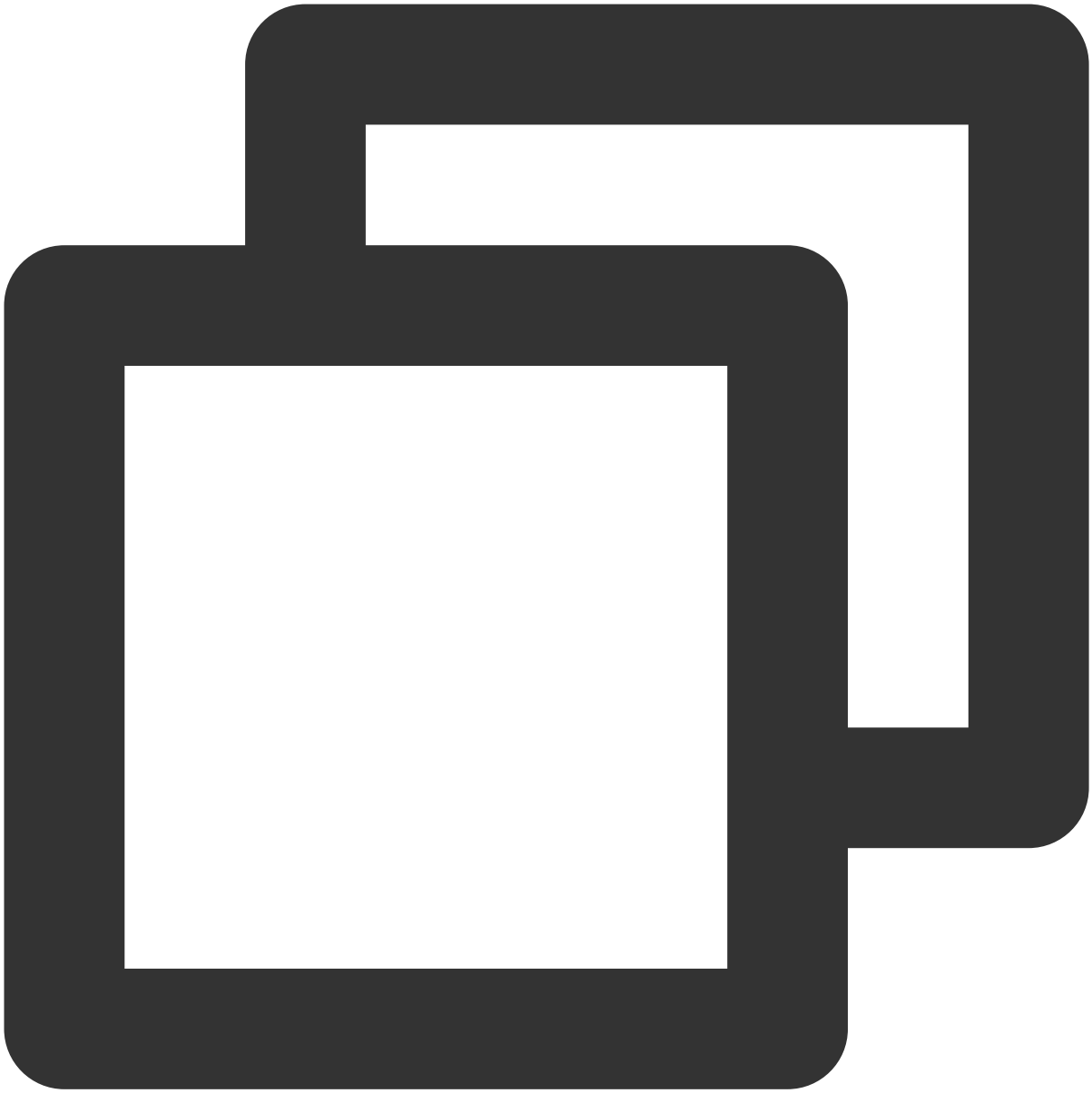
点播播放：播放器组件支持两种点播播放方式，可以 通过 FileID 播放 腾讯云点播媒体资源，也可以直接使用 [URL 播放] (#url) 地址进行播放。

直播播放：播放器组件可使用 URL 播放 的方式实现直播播放。通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播URL生成方式可参见 自主拼装直播 URL。

通过 URL 播放（直播、点播）

通过 FileID 播放（点播）

URL 可以是点播文件播放地址，也可以是直播拉流地址，传入相应 URL 即可播放相应视频文件。



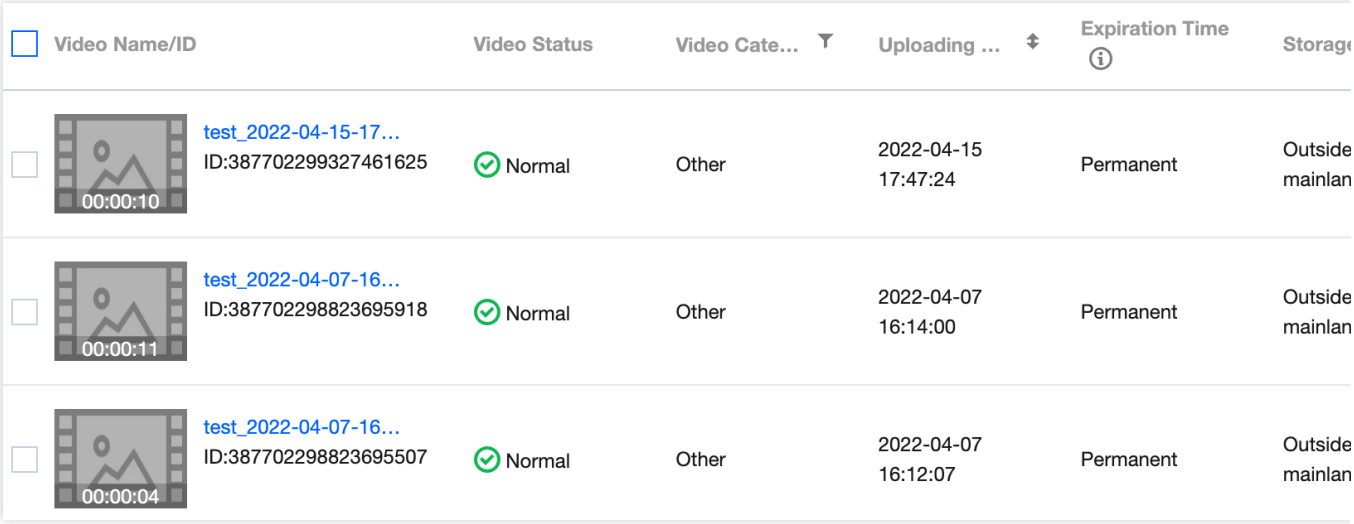
```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073; // 配置 AppId
model.url = "http://your_video_url.mp4";    // 配置您的播放视频 url
mSuperPlayerView.playWithModelNeedLicence(model);
```

视频 FileId 在一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 FileId 到客户端。

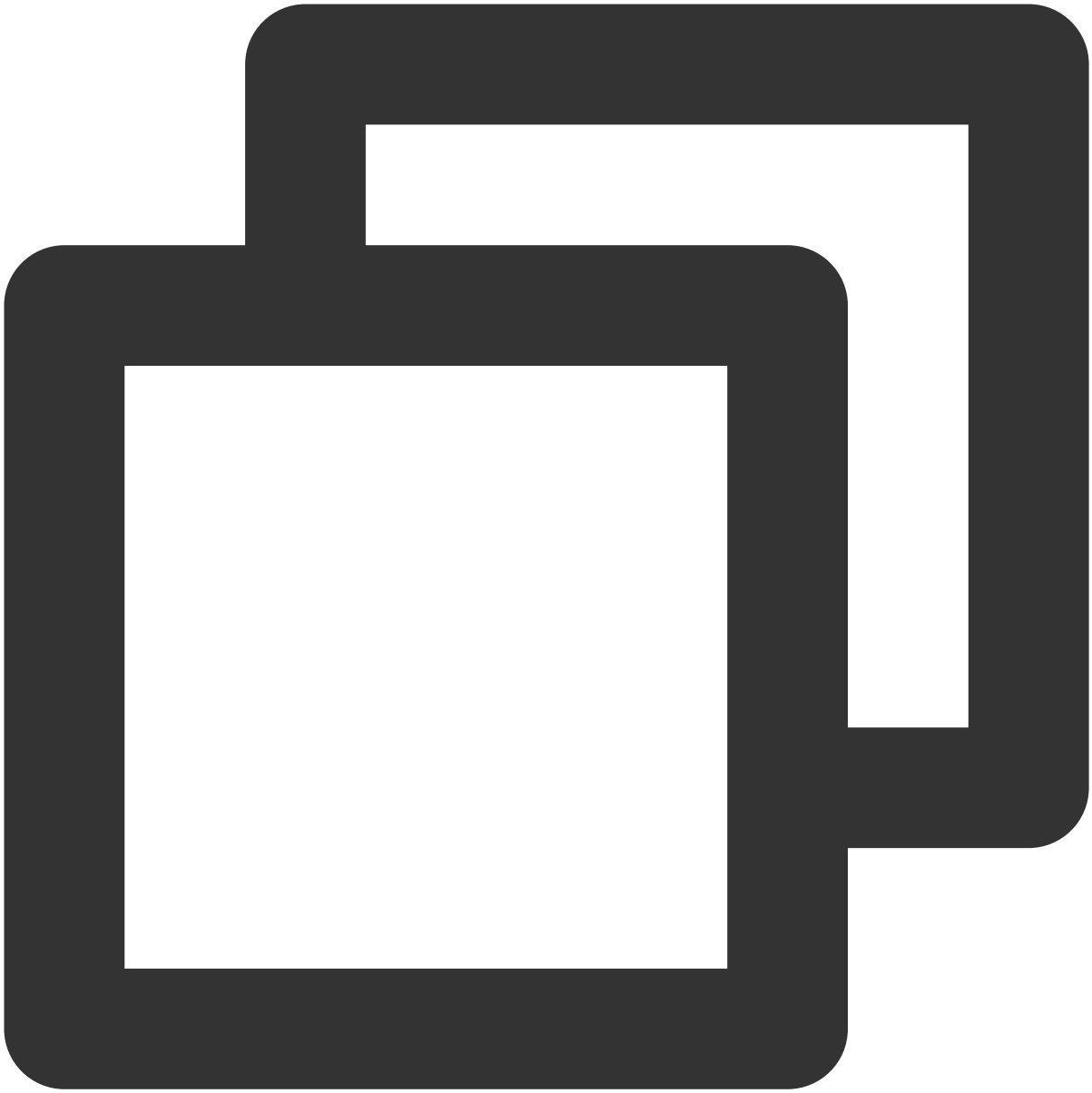
2. 服务端视频上传时，在 确认上传 的通知中包含对应的 FileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件，查看 FileId。如下图所示，ID 即表示 FileId：



注意：

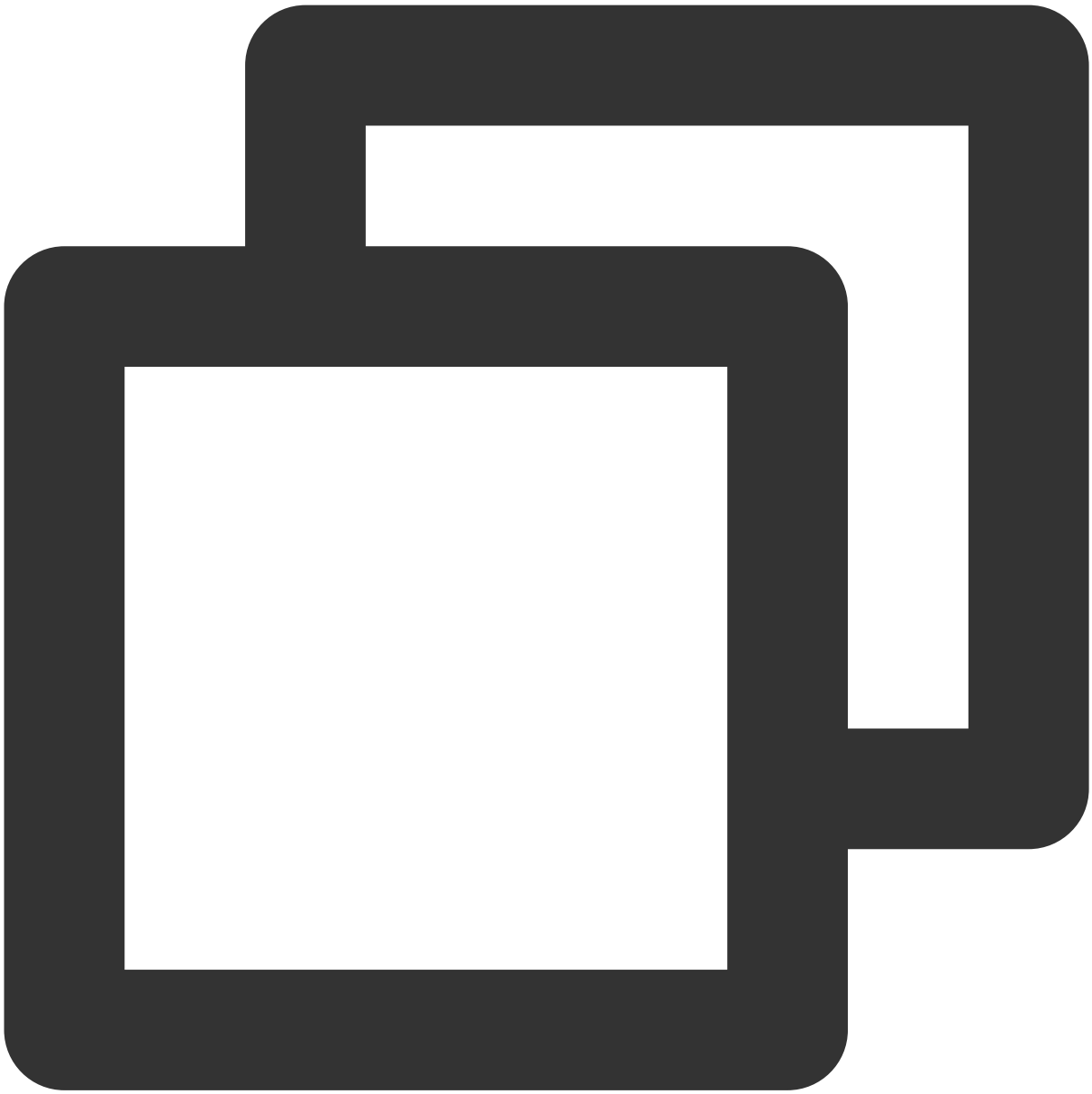
1. 通过 FileID 播放时，需要首先使用 Adaptive-HLS(10) 转码模板对视频进行转码，或者使用播放器组件签名 psign 指定播放的视频，否则可能导致视频播放失败。转码教程和说明可参见 用播放器组件播放视频，psign 生成教程可参见 psign 教程。
2. 若您在通过 FileID 播放时出现 “no v4 play info” 异常，则说明您可能存在上述问题，建议您根据上述教程调整。同时您也可以直接获取源视频播放链接，通过 URL 播放的方式实现播放。
3. 未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放。



```
//在未开启防盗链进行播放的过程中，如果出现了“no v4 play info”异常，建议您使用 Adaptive-HLS(

SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329071;// 配置 AppId
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "5285890799710173650"; // 配置 FileId
// psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/prod
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA
mSuperPlayerView.playWithModelNeedLicence(model);
```

4. 退出播放

当不需要播放器时，调用 `resetPlayer` 清理播放器内部状态，释放内存。



```
mSuperPlayerView.resetPlayer();
```

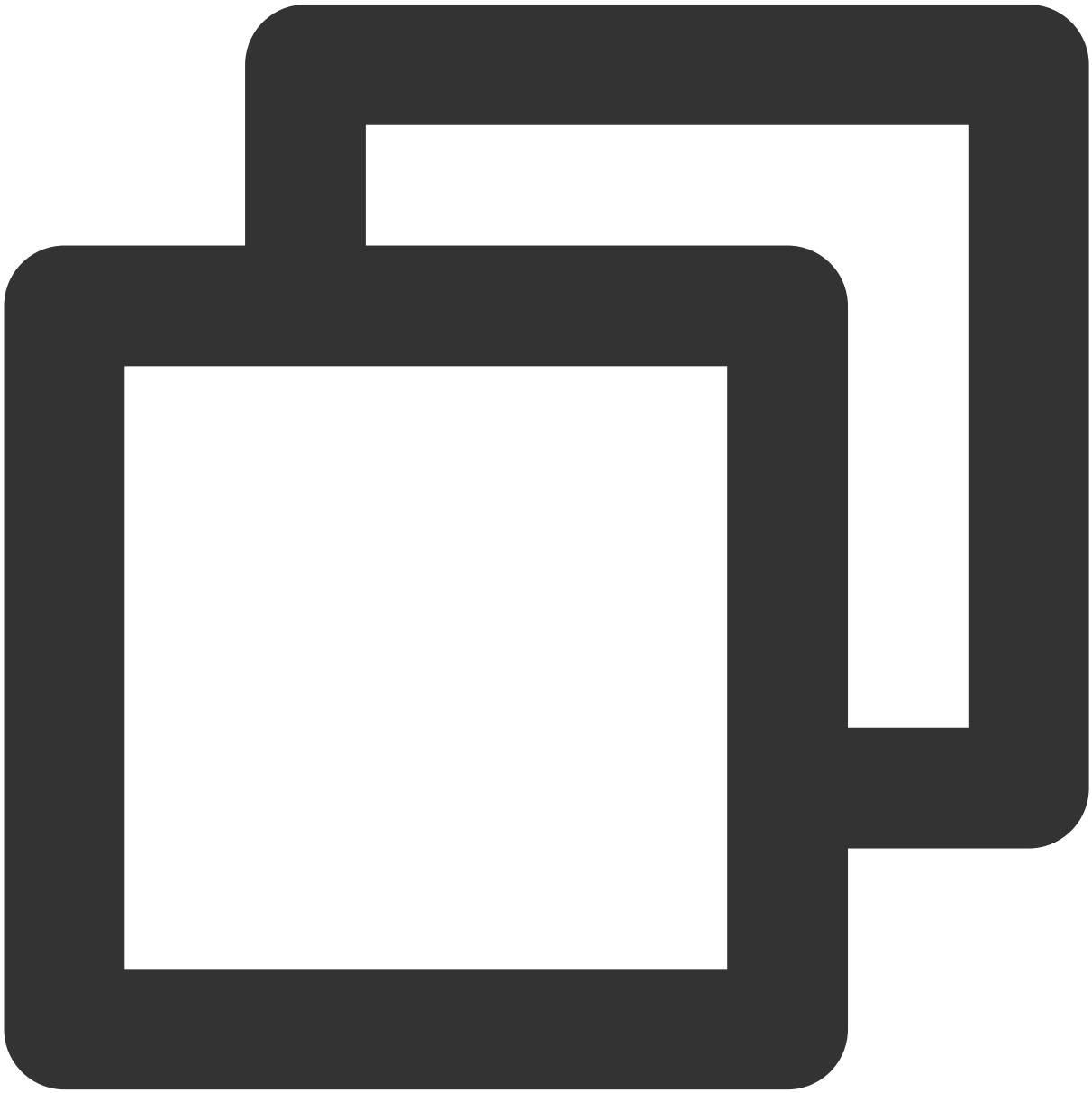
至此，您已经完成了腾讯云视立方 Android 播放器组件创建、播放视频和退出播放的能力即成。

功能使用

本章将为您介绍几种常见的播放器功能使用方式，更为完整的功能使用方式可参见 Demo 体验，播放器组件支持的功能可参见 能力清单。

1、全屏播放

播放器组件支持全屏播放，在全屏播放场景内，同时支持锁屏、手势控制音量和亮度、弹幕、截屏、清晰度切换等功能设置。功能效果可在 腾讯云视立方 App > **播放器** > **播放器组件** 中体验，单击界面右下角即可进入全屏播放界面。
在窗口播放模式下，可通过调用下述接口进入全屏播放模式：



```
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.FULLSCREEN);
```

全屏播放界面功能介绍

返回窗口

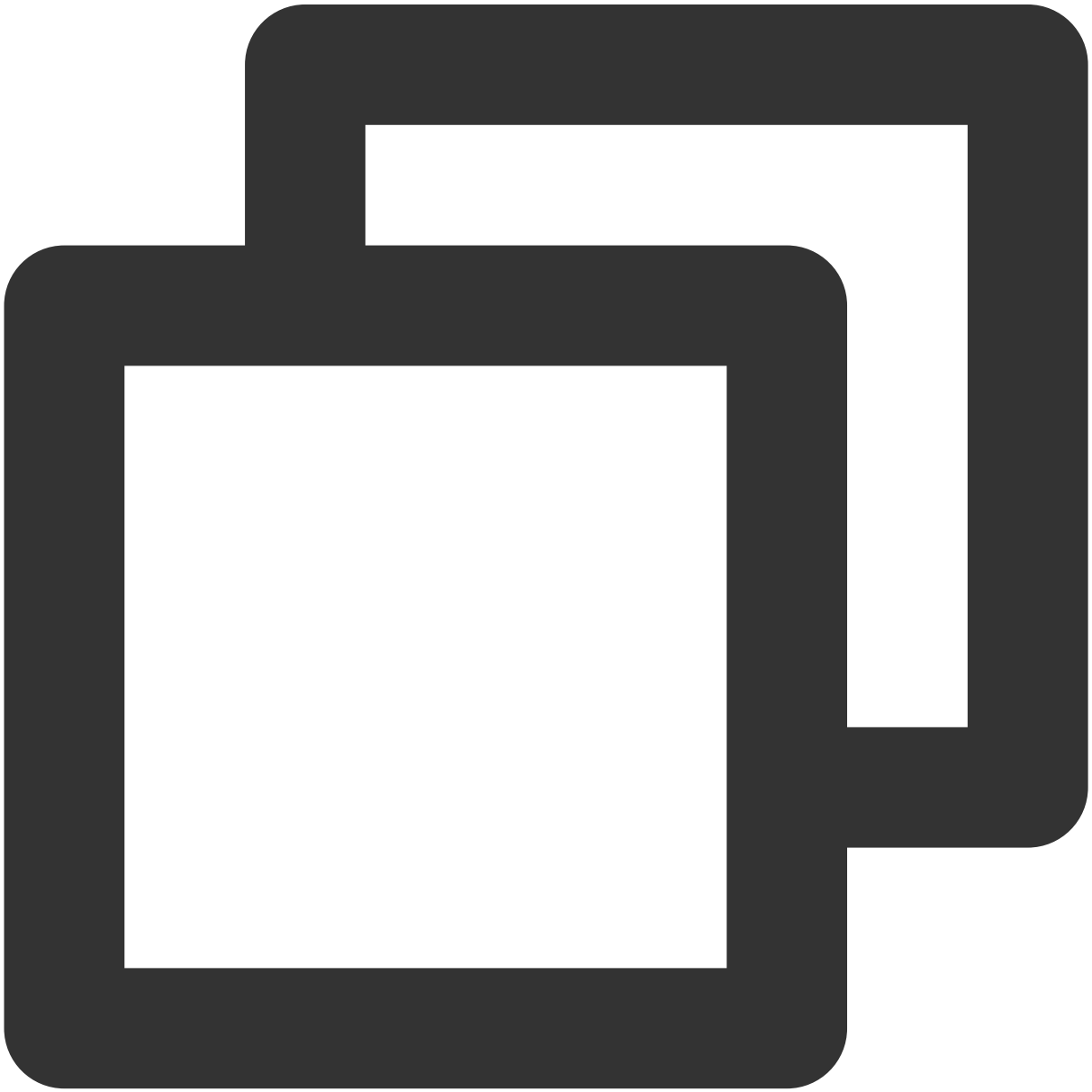
锁屏

弹幕

截屏

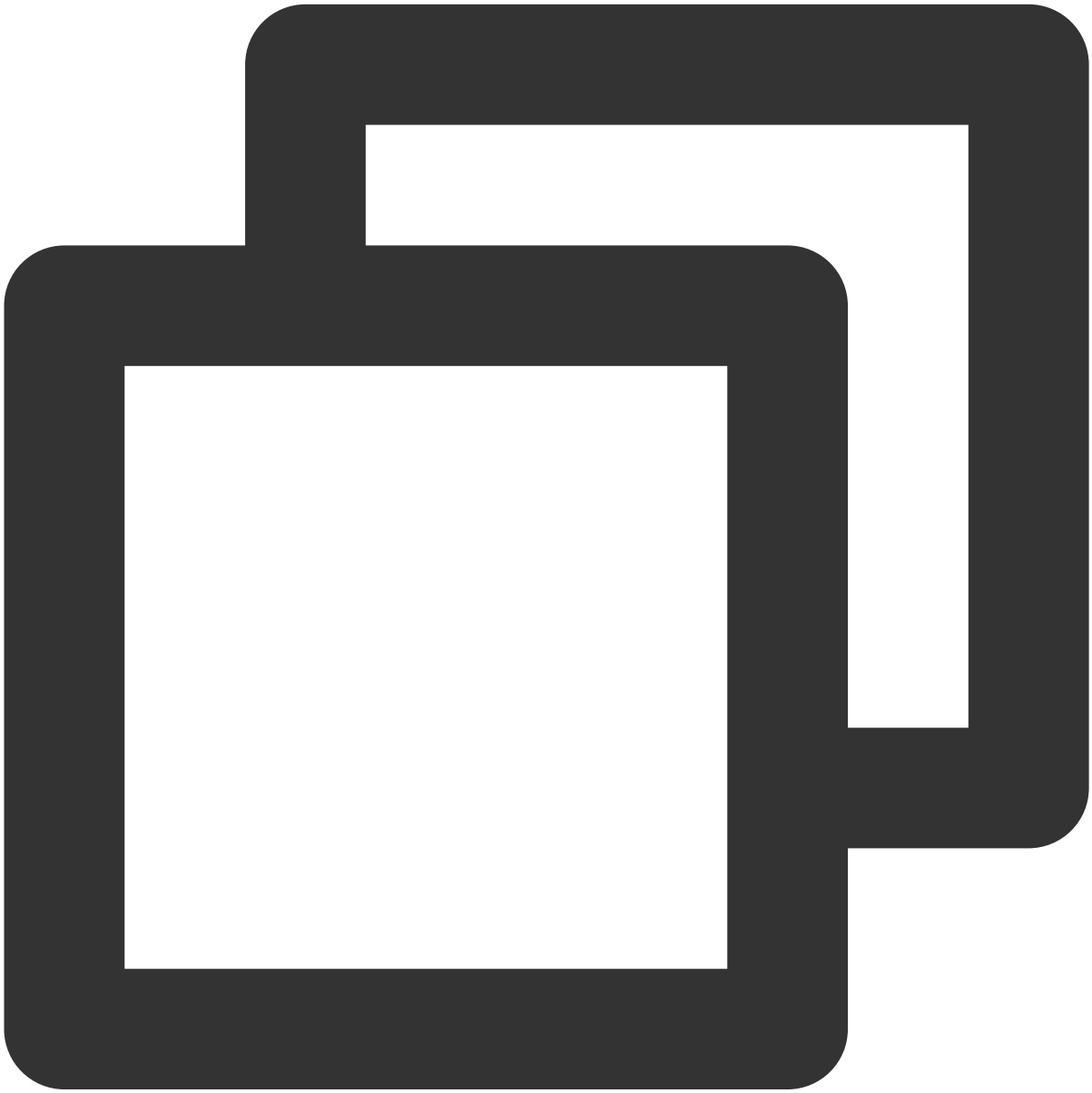
清晰度切换

单击 **返回**，即可返回至窗口播放模式。



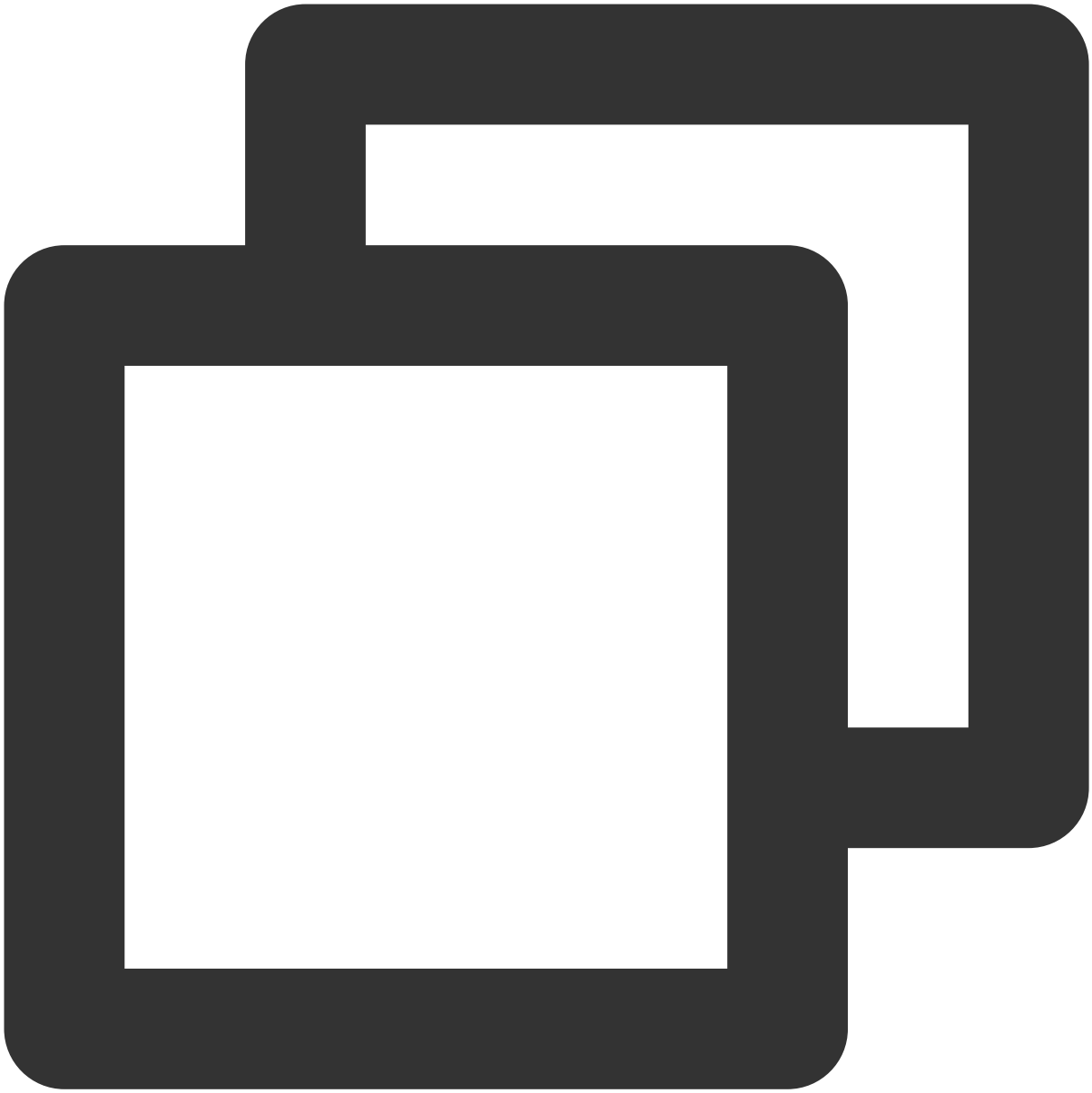
```
//单击后触发下面的接口
mControllerCallback.onBackPressed(SuperPlayerDef.PlayerMode.FULLSCREEN);
onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

锁屏操作可以让用户进入沉浸式播放状态。



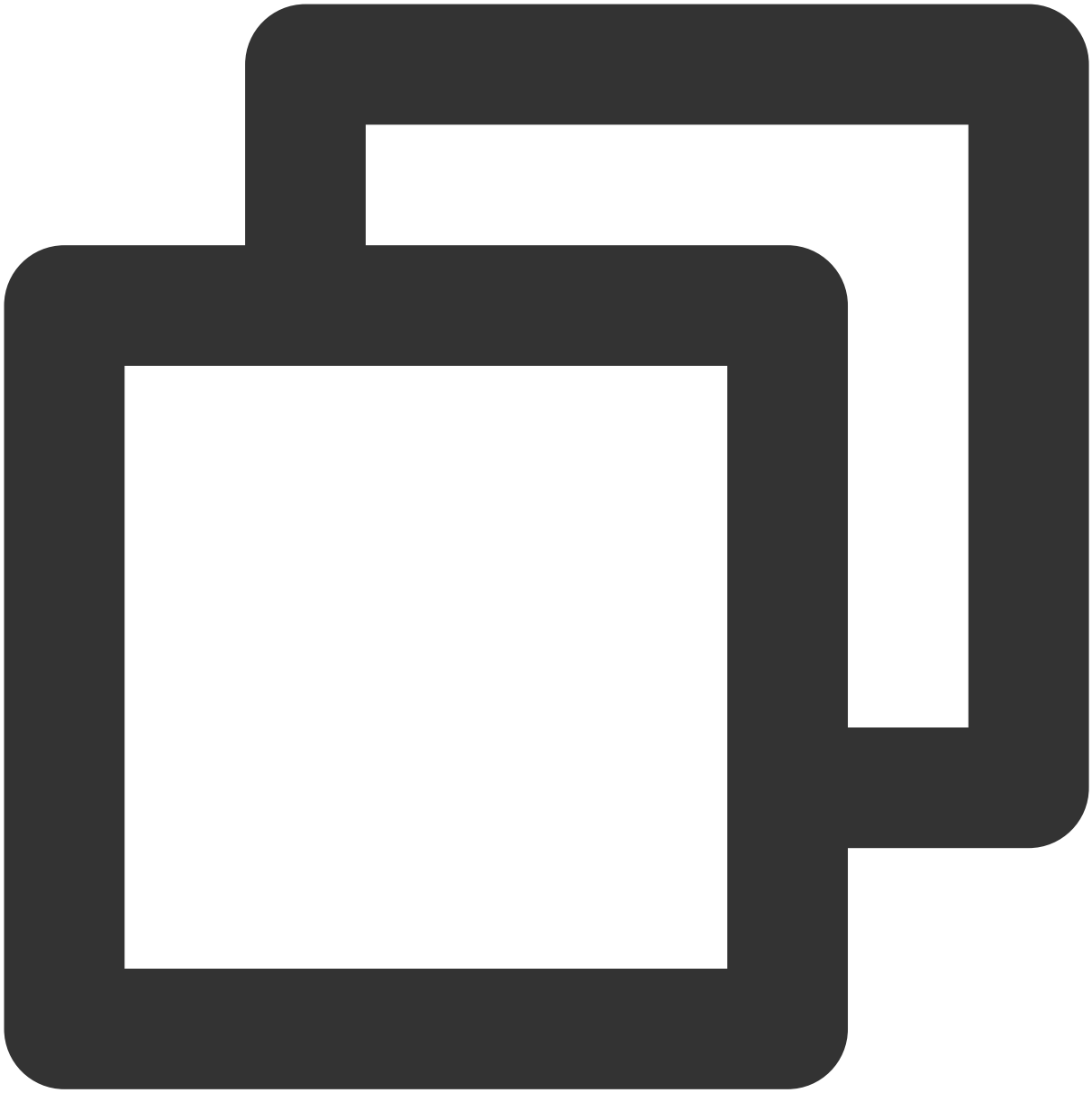
```
//单击后触发的接口
toggleLockState();
```

打开弹幕功能后屏幕上会有用户发送的文字飘过。



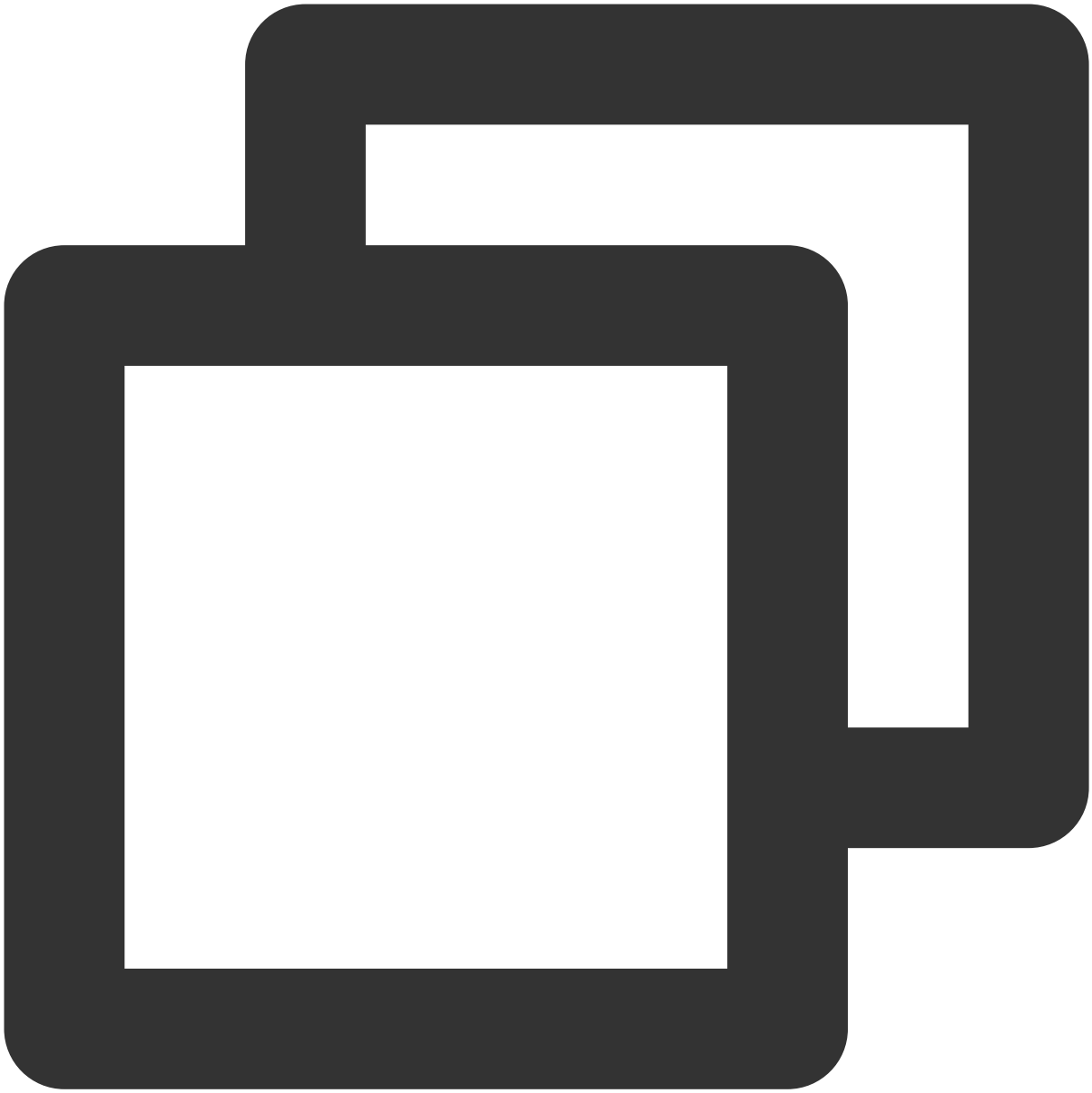
```
// 步骤一：向弹幕View中添加一条弹幕
addDanmaku(String content, boolean withBorder);
// 步骤二：打开或者关闭弹幕
toggleBarrage();
```

播放器组件提供播放过程中截取当前视频帧功能，您可以把图片保存起来进行分享。单击图片4处按钮可以截屏，您可以在 mSuperPlayer.snapshot 接口进行保存截取的图片。



```
mSuperPlayer.snapshot(new TXLivePlayer.ITXSnapshotListener() {
  @Override
  public void onSnapshot(Bitmap bitmap) {
        //在这里可以保存截图
  }
});
```

用户可以根据需求选择不同的视频播放清晰度，如高清、标清或超清等。

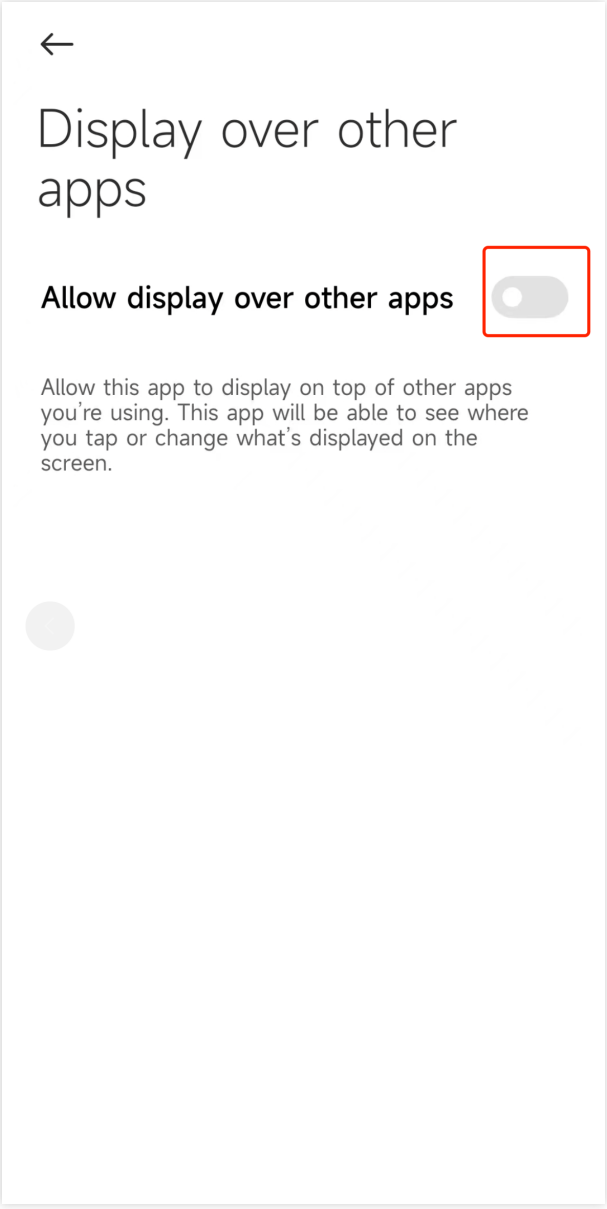


```
//单击后触发的显示清晰度 view 代码接口
showQualityView();
//单击清晰度选项的回调接口为
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int position, long id)
    // 清晰度ListView的单击事件
    VideoQuality quality = mList.get(position);
    mCallback.onQualitySelect(quality);
  }
});
```

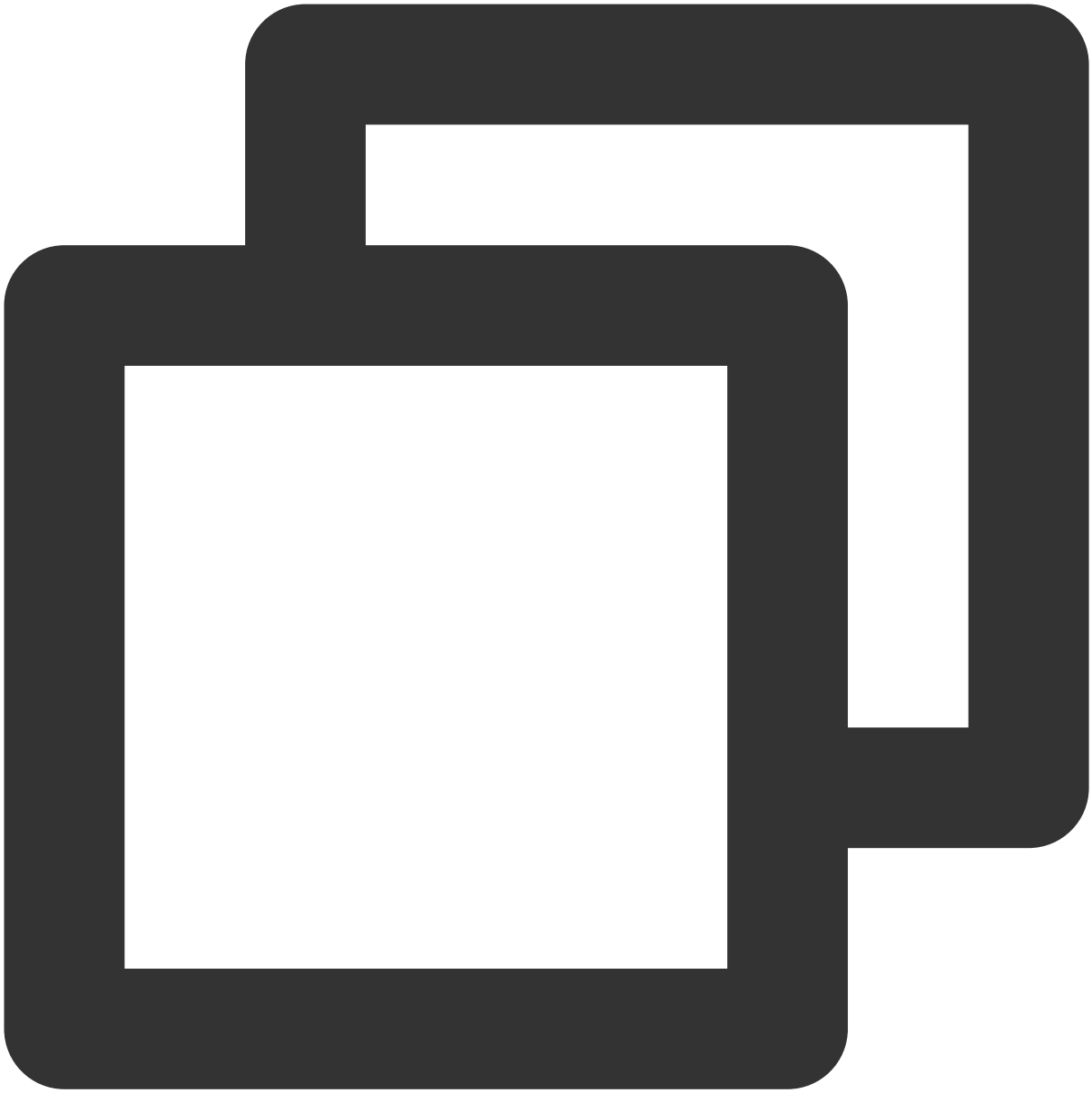
```
//最终改变清晰度的回调
@Override
public void onQualityChange(VideoQuality quality) {
    mFullScreenPlayer.updateVideoQuality(quality);
    mSuperPlayer.switchStream(quality);
}
```

2、悬浮窗播放

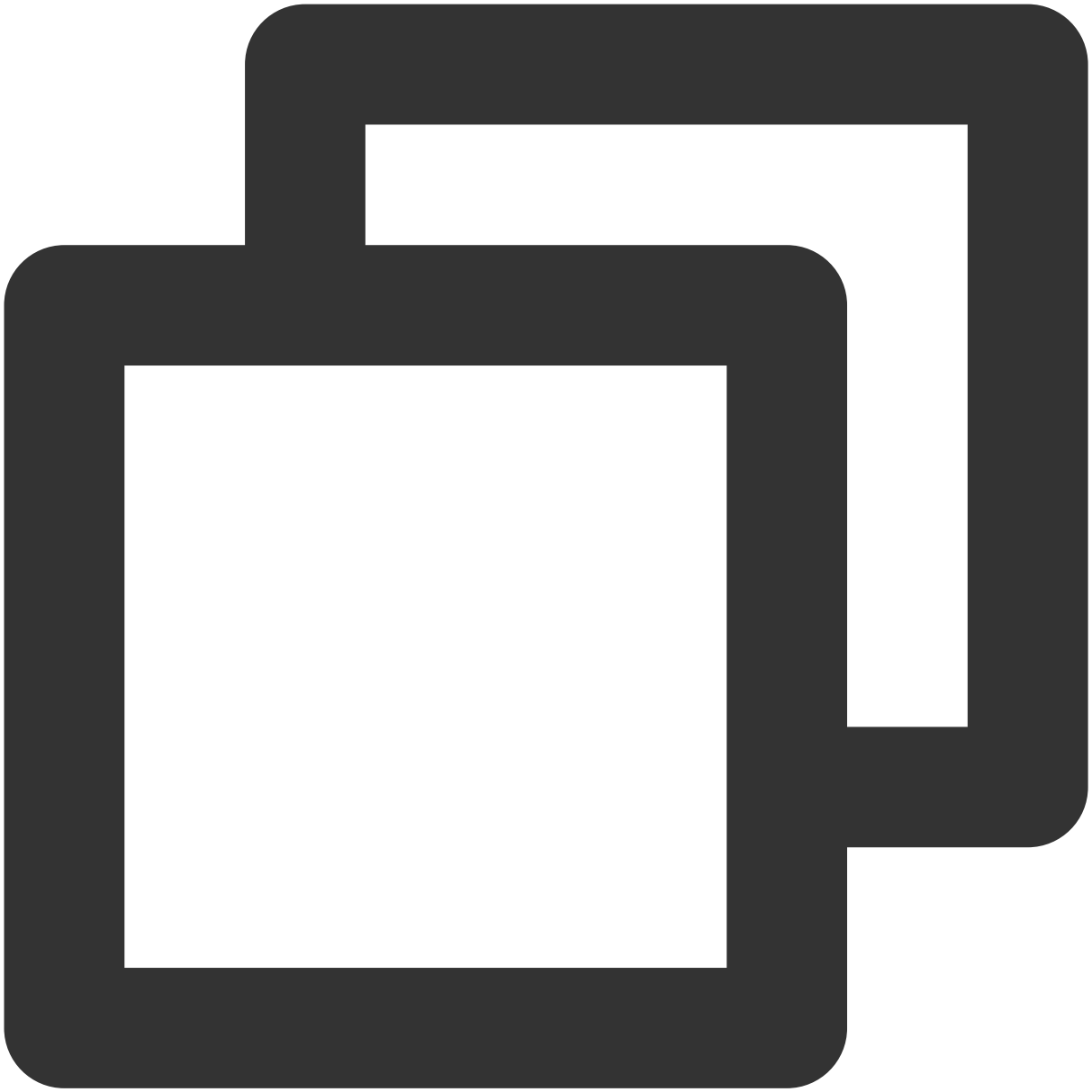
播放器组件支持悬浮窗小窗口播放，可在切换到其它应用时，不打断视频播放功能。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件** 中体验，单击界面左上角**返回**，即可体验悬浮窗播放功能。



悬浮窗播放依赖于 AndroidManifest 中的以下权限：



```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
```

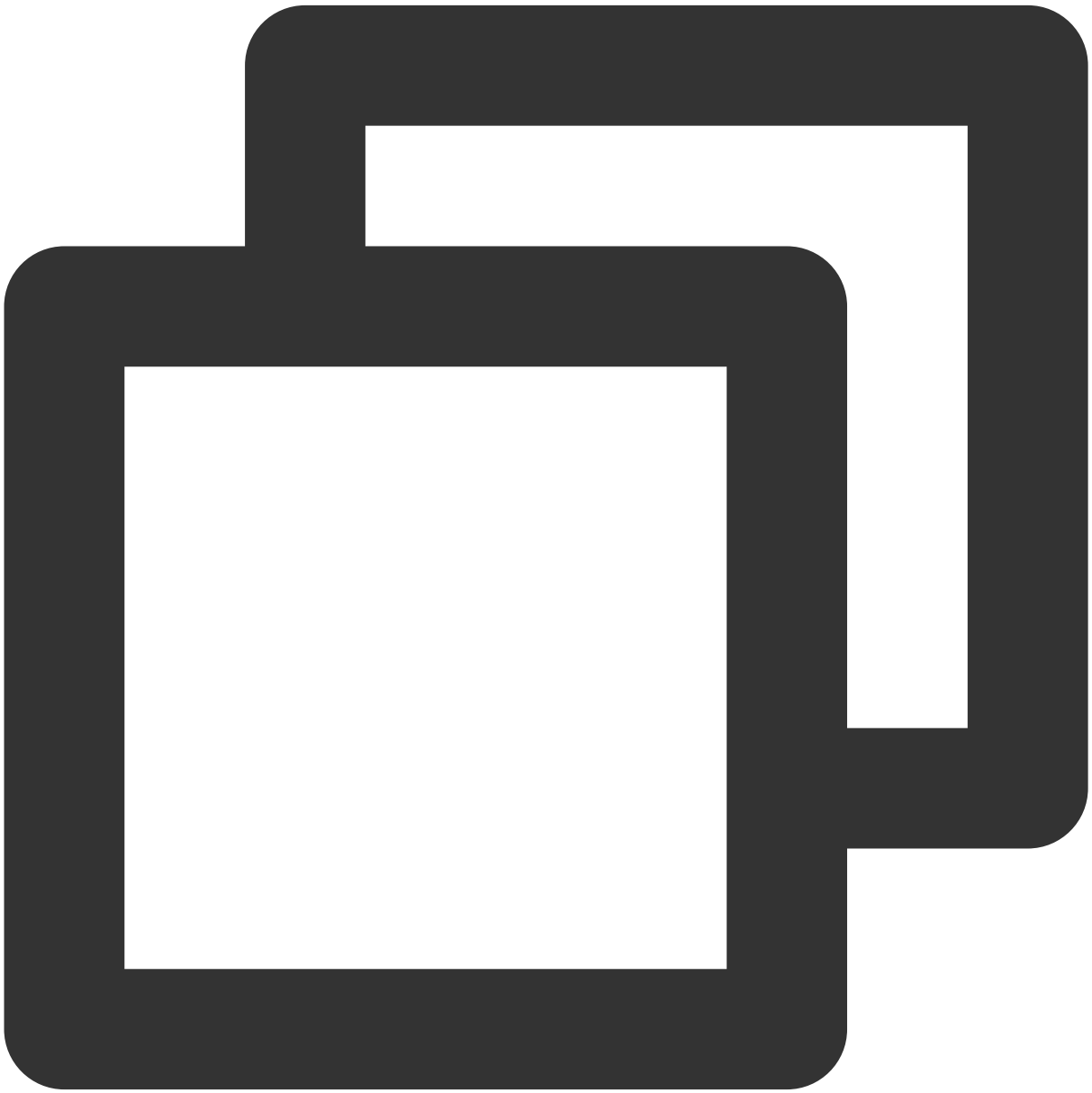


```
// 切换悬浮窗触发的代码接口
mSuperPlayerView.switchPlayMode(SuperPlayerDef.PlayerMode.FLOAT);
//单击浮窗返回窗口触发的代码接口
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

3、视频封面

播放器组件支持用户自定义视频封面，用于在视频接收到首帧画面播放回调前展示。功能效果可在 **腾讯云视立方 App> 播放器 > 播放器组件 > 自定义封面演示** 视频中体验。

当播放器组件设置为自动播放模式 `PLAY_ACTION_AUTO_PLAY` 时，视频自动播放，此时将在视频首帧加载出来之前展示封面；
当播放器组件设置为手动播放模式 `PLAY_ACTION_MANUAL_PLAY` 时，需用户单击**播放**后视频才开始播放。在单击**播放**前将展示封面；在单击**播放**后到视频首帧加载出来前也将展示封面。
视频封面支持使用网络 URL 地址或本地 File 地址，使用方式可参见下述指引。若您通过 FileID 的方式播放视频，则可直接在云点播内配置视频封面。



```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = "您的 appid";
model.videoId = new SuperPlayerVideoId();
```

```
model.videoId.fileId = "您的 fileId";
//播放模式，可设置自动播放模式：PLAY_ACTION_AUTO_PLAY，手动播放模式：PLAY_ACTION_MANUAL_PLA
model.playAction = PLAY_ACTION_MANUAL_PLAY;
//设定封面的地址为网络 url 地址，如果 coverPictureUrl 不设定，那么就会自动使用云点播控制台设置的
model.coverPictureUrl = "http://1500005830.vod2.myqcloud.com/6c9a5118vodcq150000583
mSuperPlayerView.playWithModelNeedLicence(model);
```

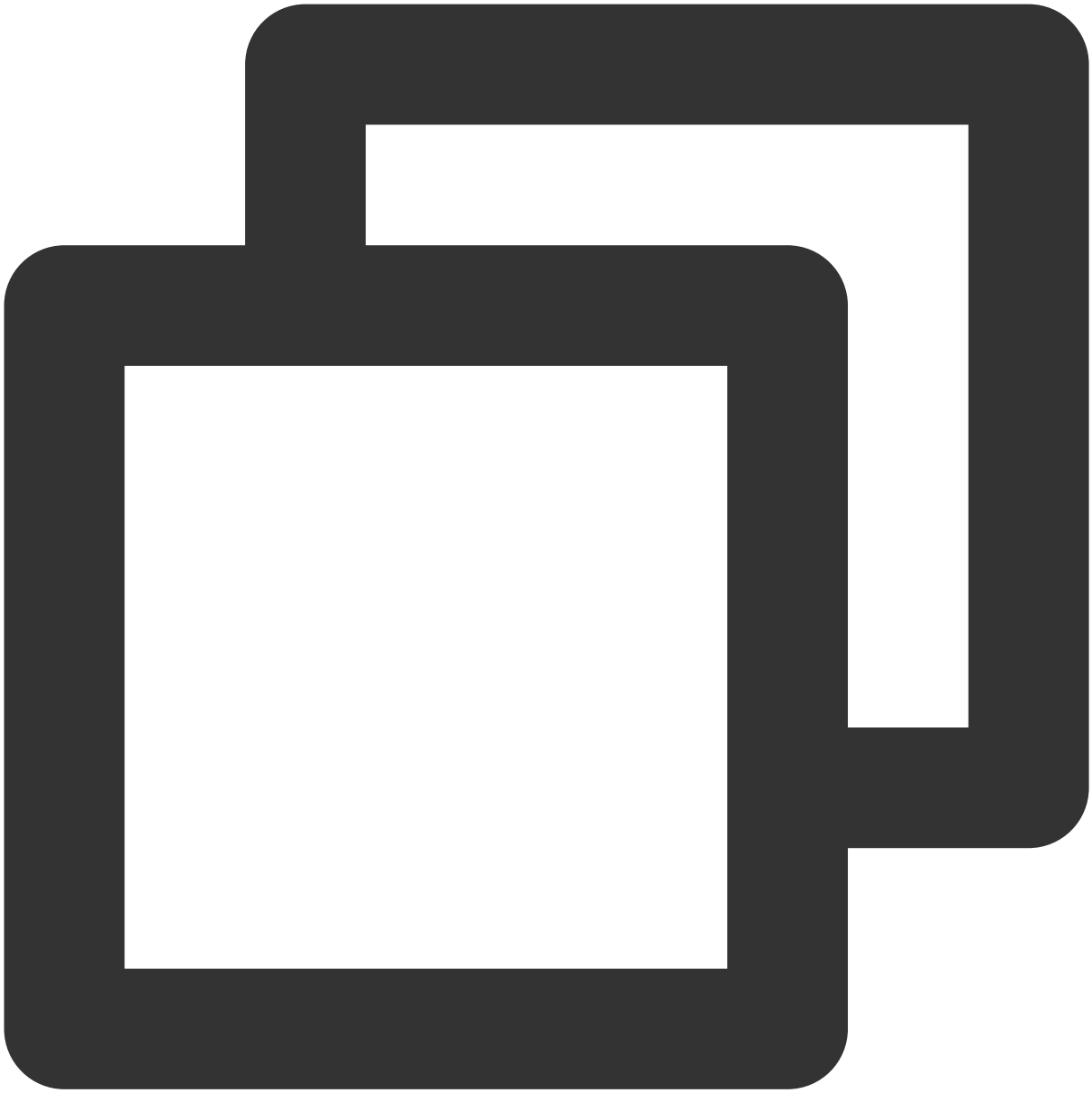
4、视频列表轮播

播放器组件支持视频列表轮播，即在给定一个视频列表后：

支持按顺序循环播放列表中的视频，播放过程中支持自动播放下一集也支持手动切换到下一个视频。

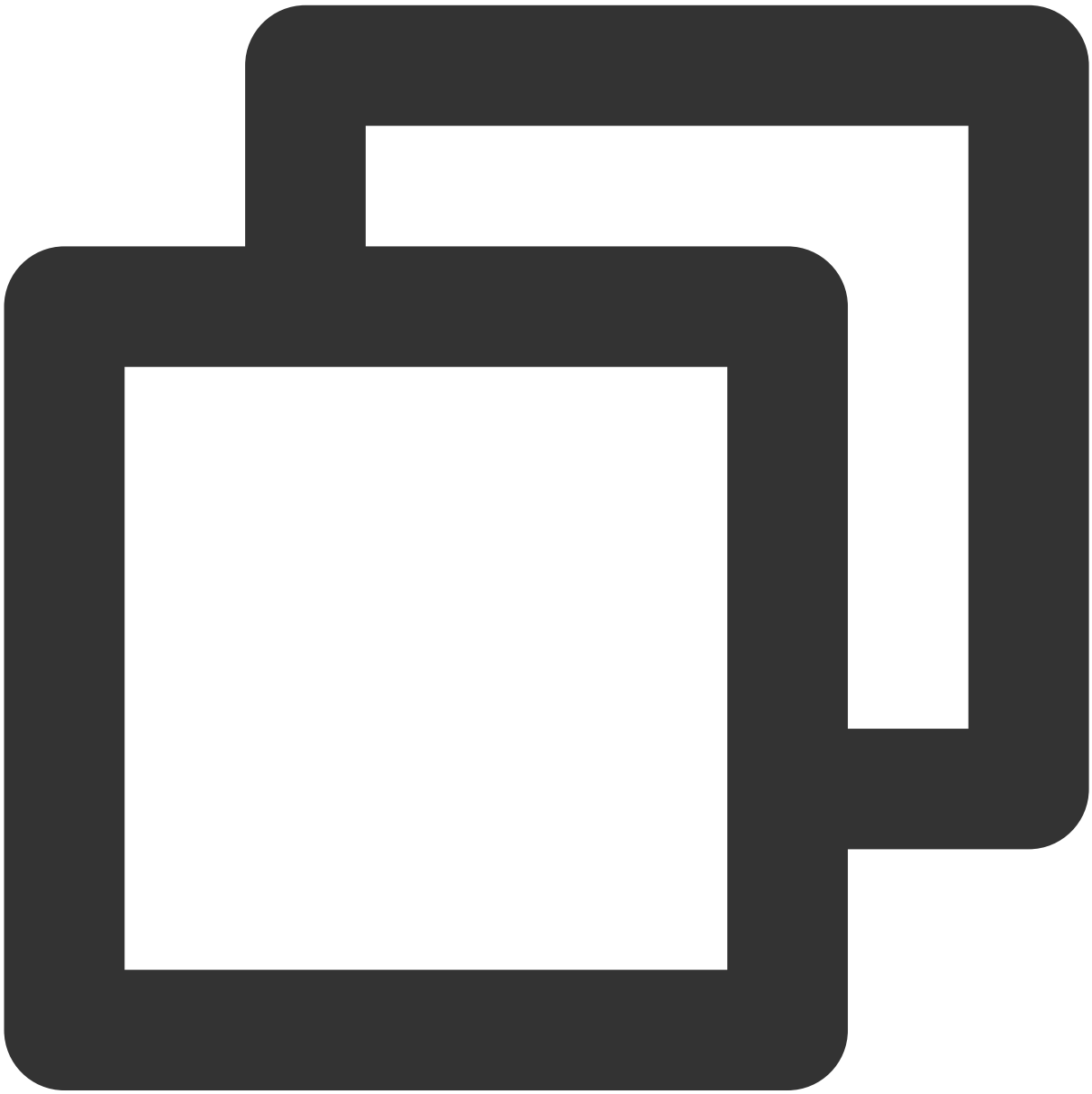
列表中最后一个视频播放完成后将自动开始播放列表中的第一个视频。

功能效果可在 **腾讯云视立方 App> 播放器 > 播放器组件 > 视频列表轮播演示**视频中体验。



```
//步骤1:构建轮播的 List<SuperPlayerModel>
ArrayList<SuperPlayerModel> list = new ArrayList<>();
SuperPlayerModel model = new VideoModel();
model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071568";
list.add(model);

model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
```

```
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071679";
list.add(model);
//步骤2：调用轮播接口
mSuperPlayerView.playWithModelListNeedLicence(list, true, 0);
```



```
public void playWithModelListNeedLicence(List<SuperPlayerModel> models, boolean isL
```

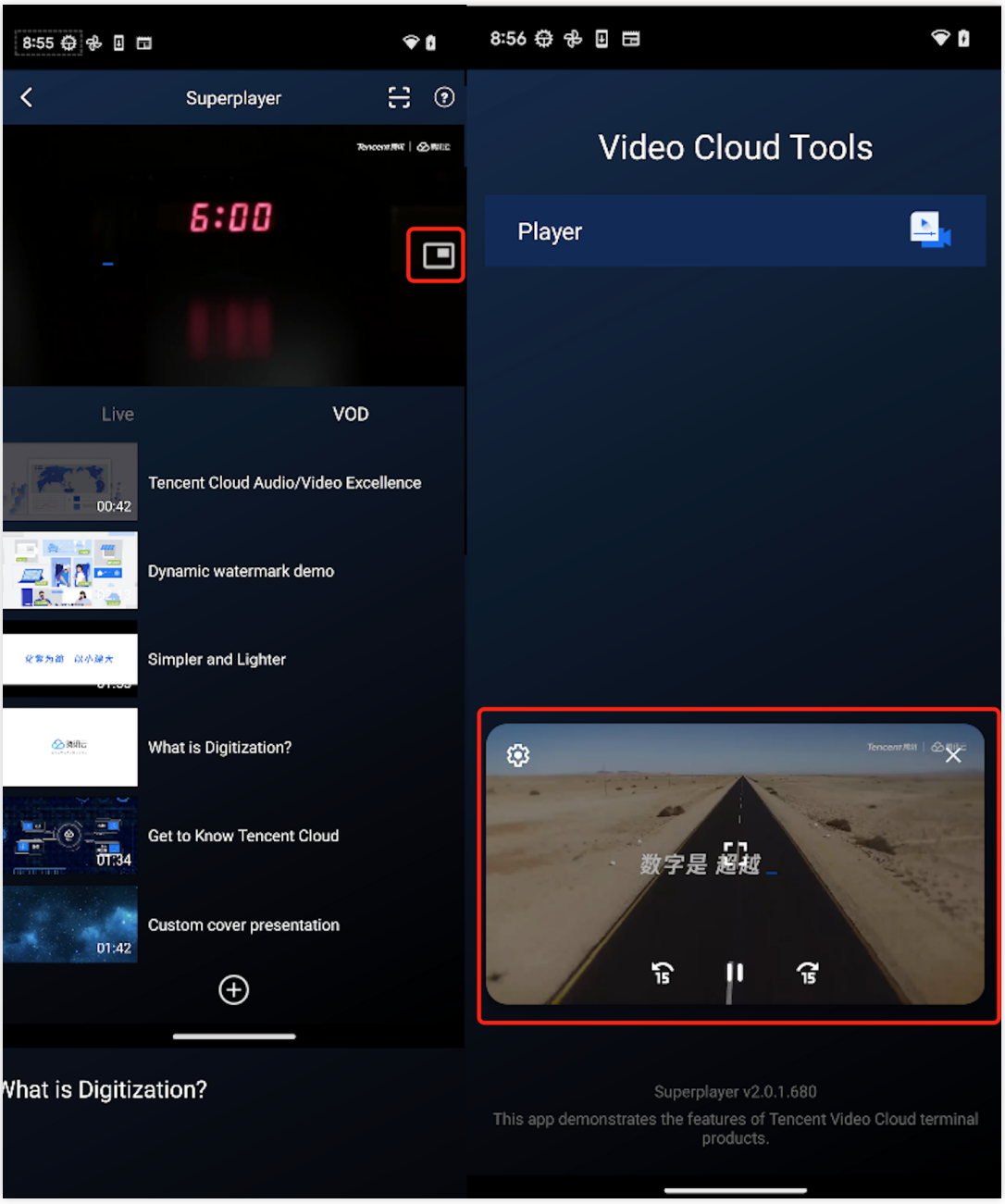
接口参数说明

参数名	类型	描述

models	List<SuperPlayerModel>	轮播数据列表
isLoopPlayList	boolean	是否循环
index	int	开始播放的 SuperPlayerModel 索引

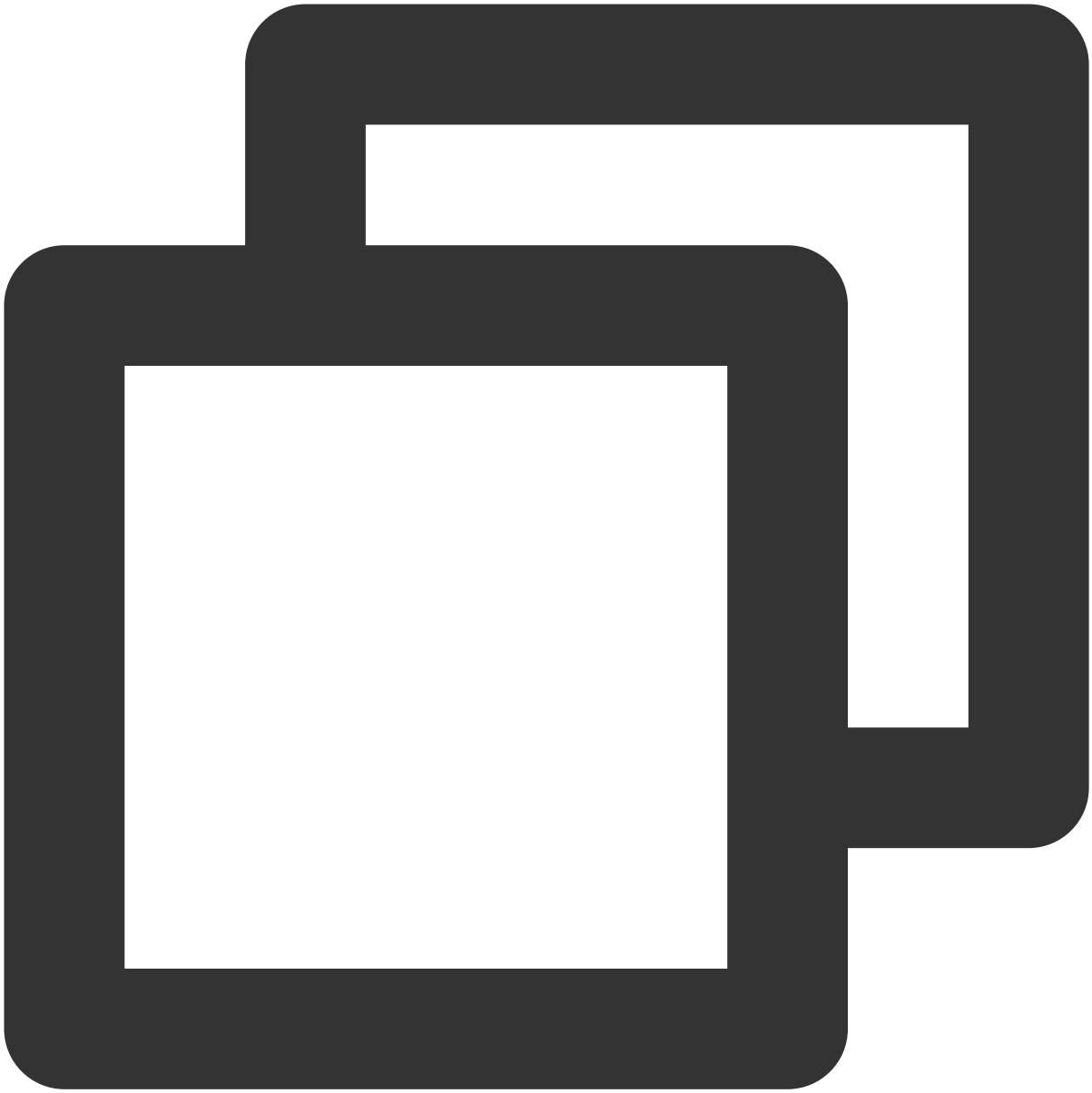
5、画中画功能

从 Android 8.0(API 级别 26)开始，Android 允许以画中画 (PiP) 模式启动 Activity。



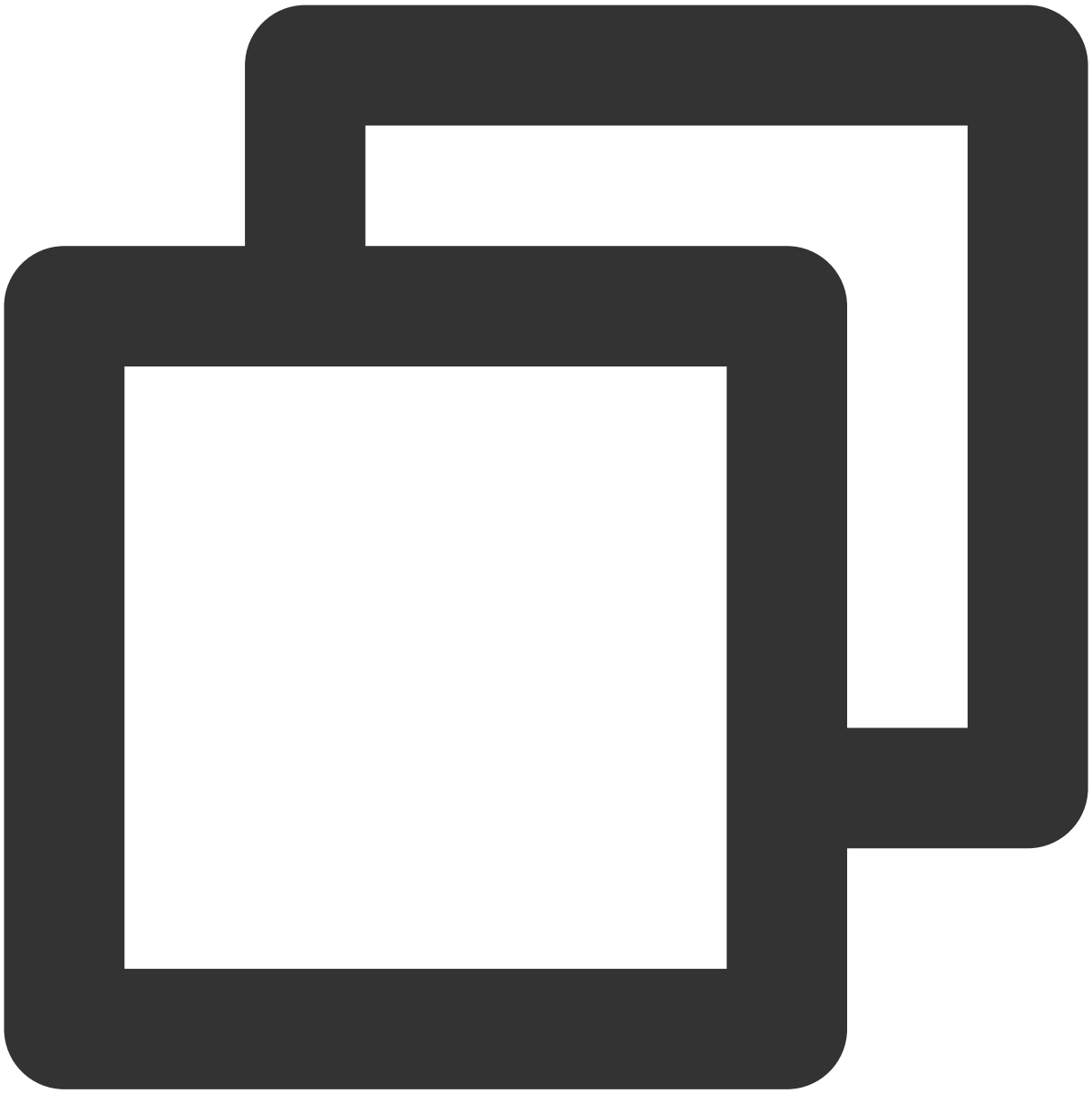
如果您需要启用或者禁用画中画，只需更改 SuperPlayerGlobalConfig 中 enablePIP 的值。

要将画中画添加到您的应用中，需要对支持画中画的 Activity 在 AndroidManifest 中加上以下属性。



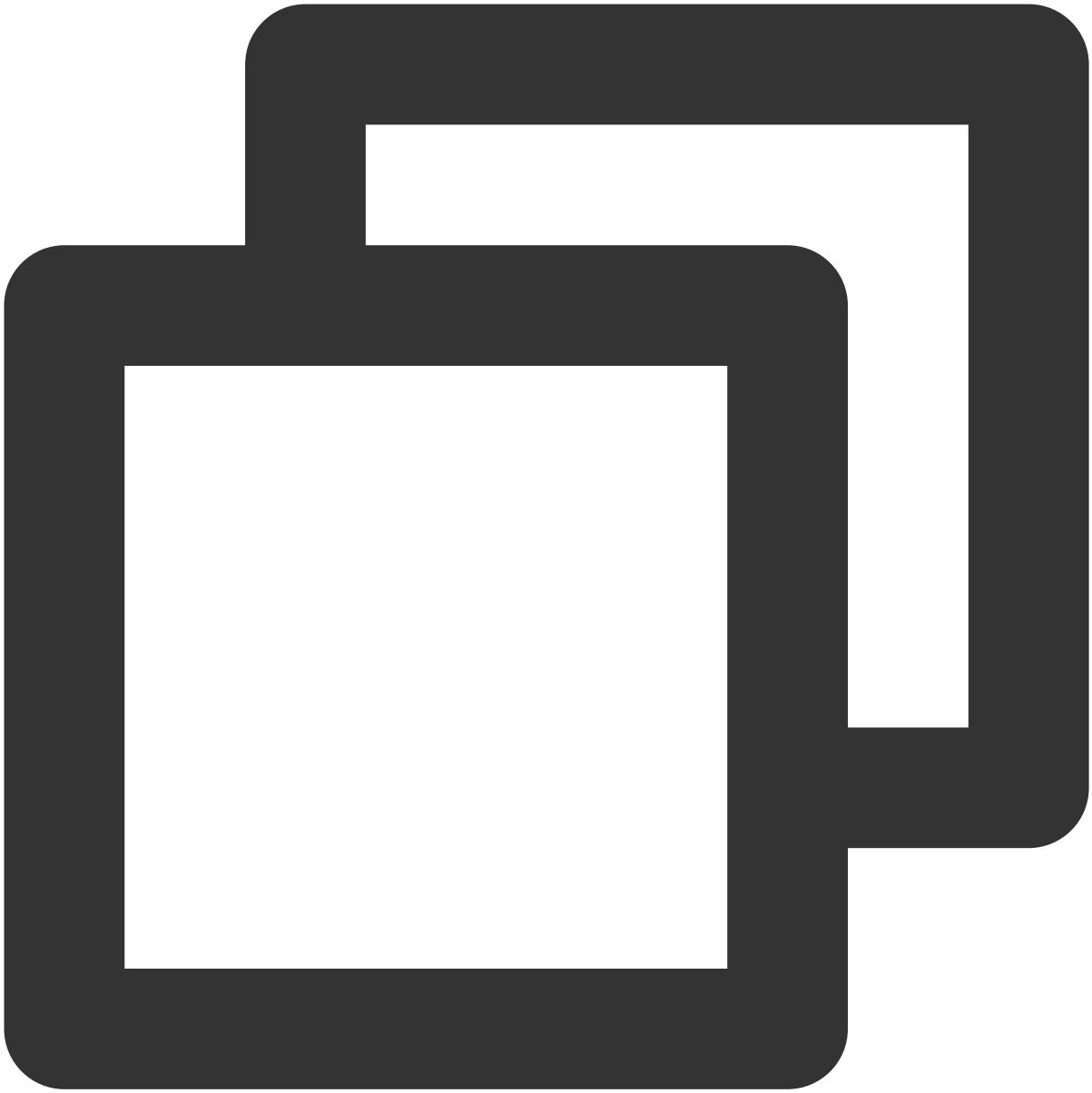
```
<activity>
    android:name=".demo.SuperPlayerActivity"
    android:resizeableActivity="true"
    android:supportsPictureInPicture="true"
    android:documentLaunchMode="intoExisting"
    android:excludeFromRecents="true"
    android:configChanges="orientation|keyboardHidden|screenSize|smallestScreenSize

</activity>
```

同时对支持对支持画中画的 Activity 的生命周期需要参照 SuperPlayerActivity 做下特殊处理。开启画中画，在 SuperPlayerView 中使用PictureInPictureHelper。



```
PictureInPictureHelper mPictureInPictureHelper = new PictureInPictureHelper(mContex
mPictureInPictureHelper.setListener(this);
mPictureInPictureHelper.enterPictureInPictureMode(getPlayerState(),mTXCloudVideoVie
```

在退出时需要在 SuperPlayerView 中释放。

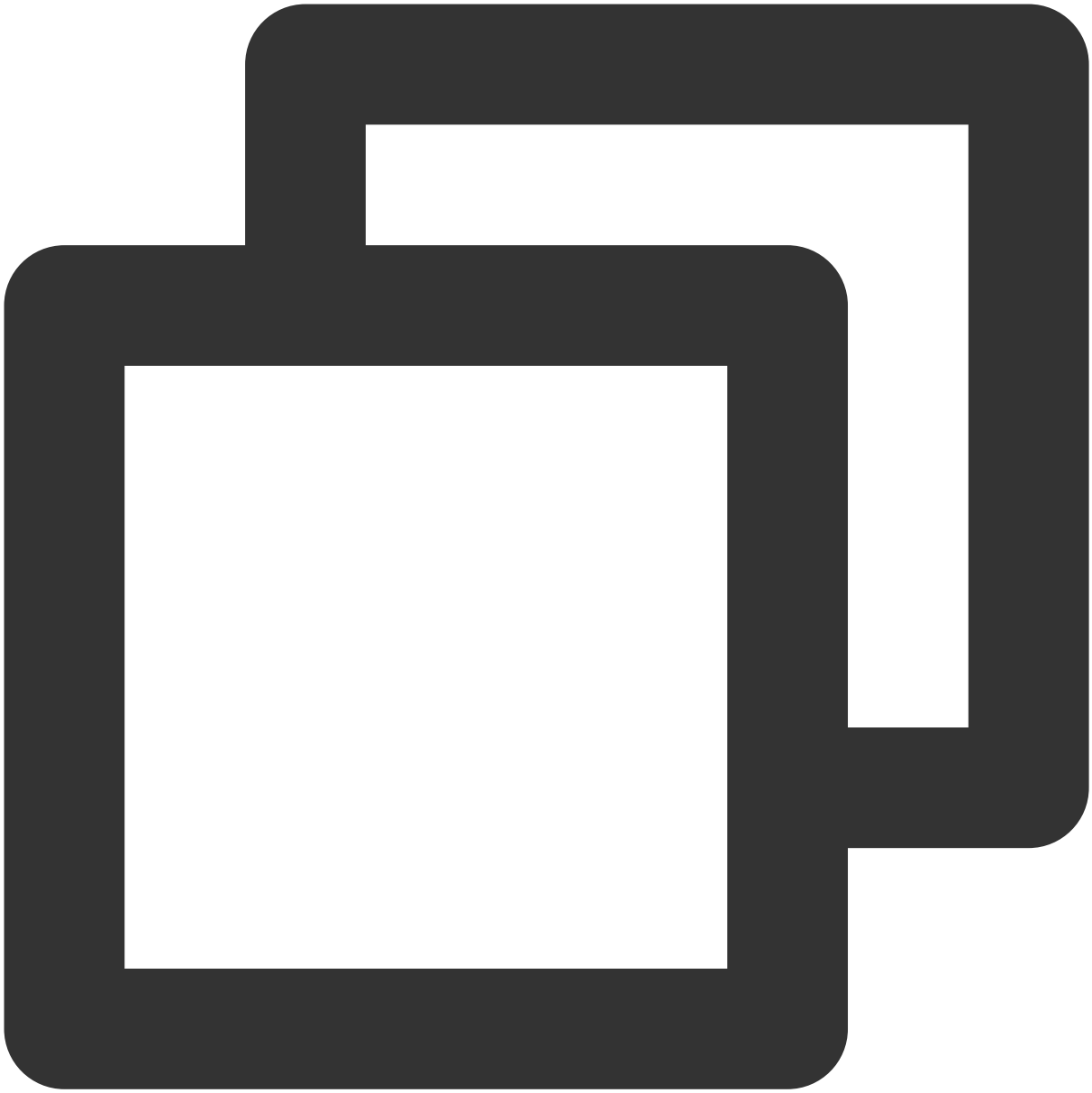


```
mPictureInPictureHelper.release();
```

如果需要对画中画中自定义按钮前移后移的时间间隔修改，只需要修改 PictureInPictureHelper 中 PIP_TIME_SHIFT_INTERVAL 的值。

6、视频试看

播放器组件支持视频试看功能，可以适用于非 VIP 试看等场景，开发者可以传入不同的参数来控制视频试看时长、提示信息、试看结束界面等。功能效果可在 **腾讯云视立方 App> 播放器 > 播放器组件 > 试看功能演示** 视频中体验。

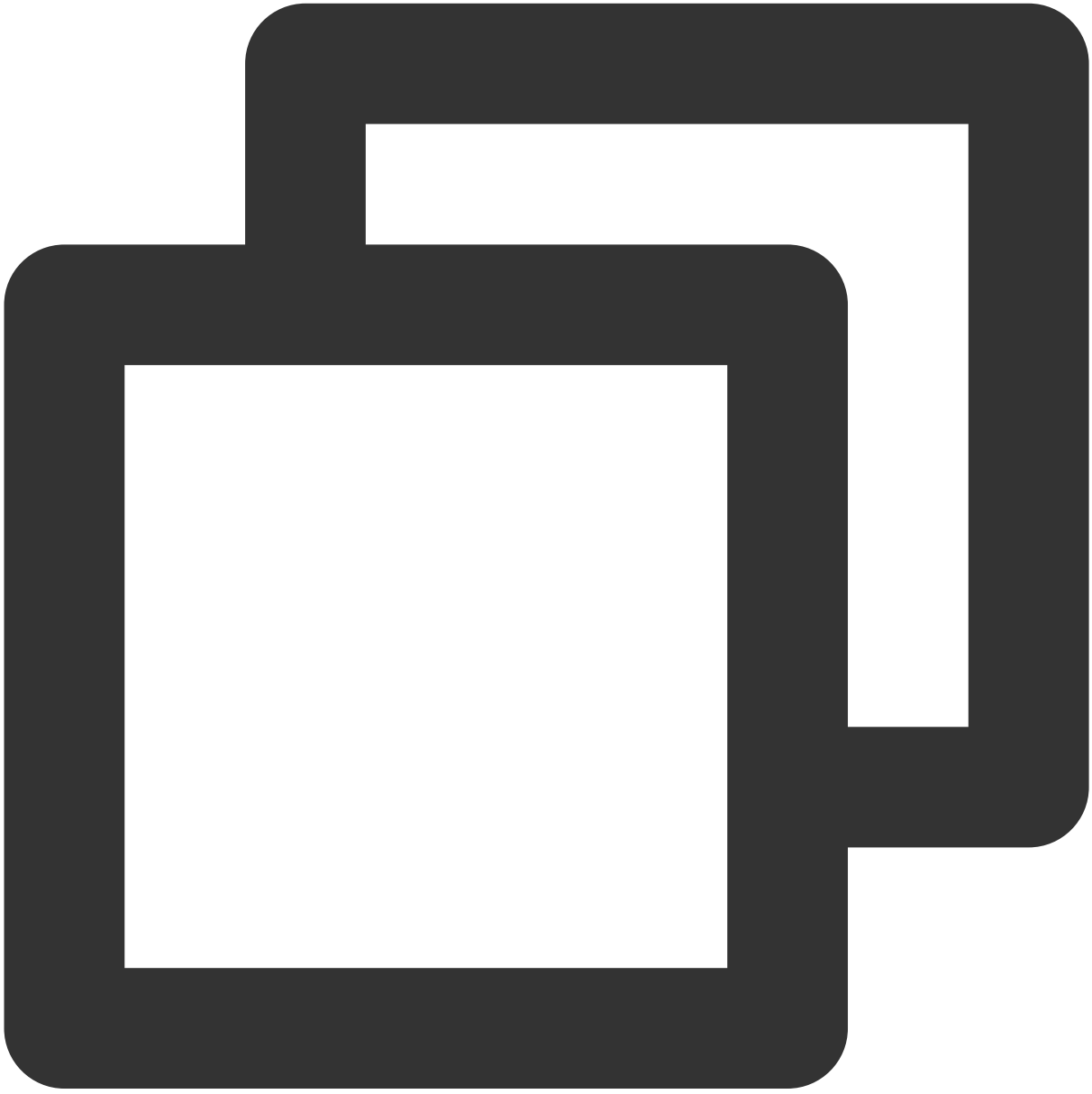


方法一：

```
//步骤1：创建视频 mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss，开通 VIP 观看完整视频",15)
mode.vipWatchMode = vipWatchModel;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModelNeedLicence(mode);
```

方法二：

```
//步骤1：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss, 开通 VIP 观看完整视频",15)
 //步骤2：调用设置试看功能方法
mSuperPlayerView.setVipWatchModel(vipWatchModel);
```



```
public VipWatchModel(String tipStr, long canWatchTime)
```

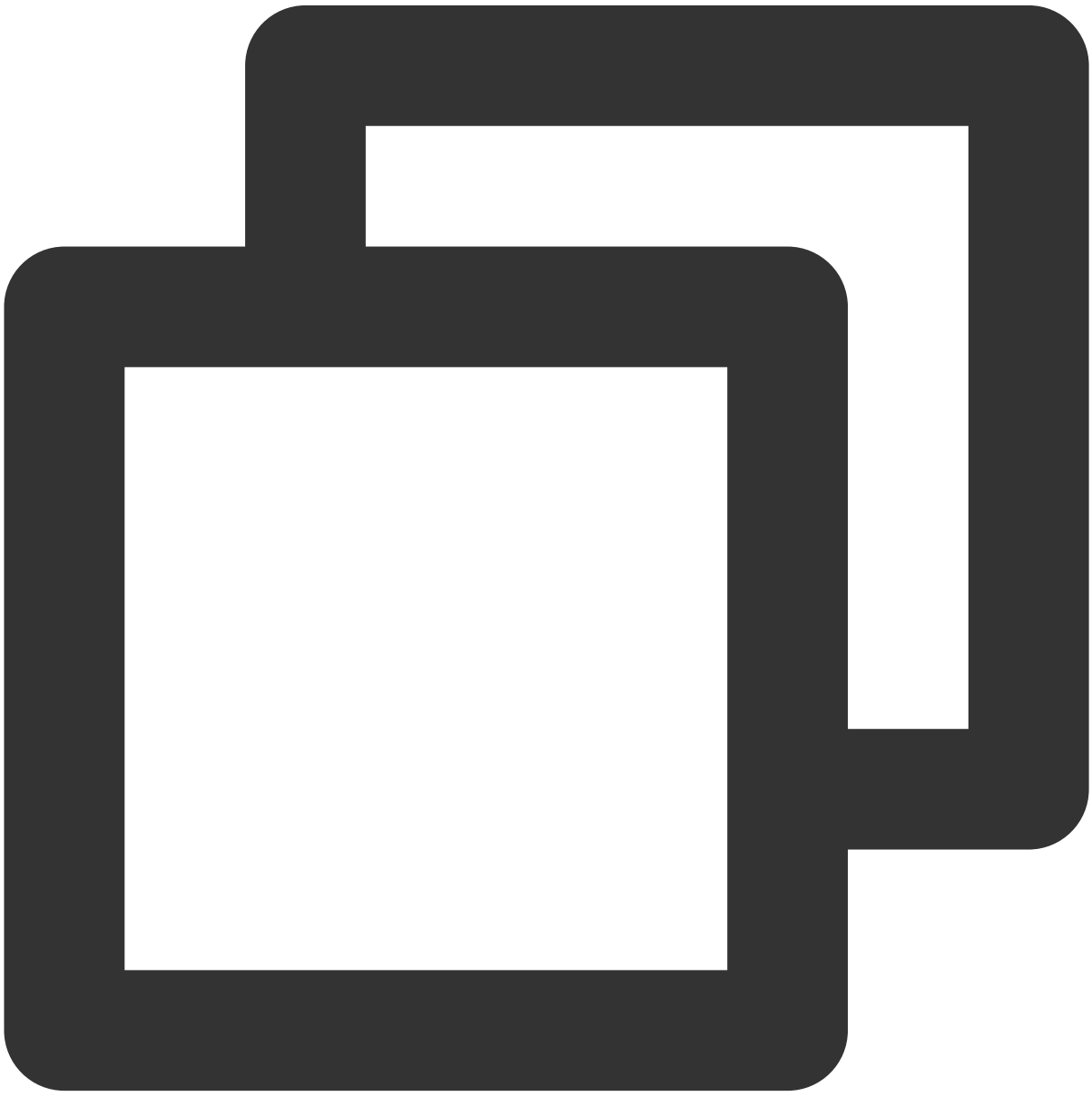
VipWatchModel 接口参数说明：

参数名	类型	描述

tipStr	String	试看提示信息
canWatchTime	Long	试看时长，单位为妙

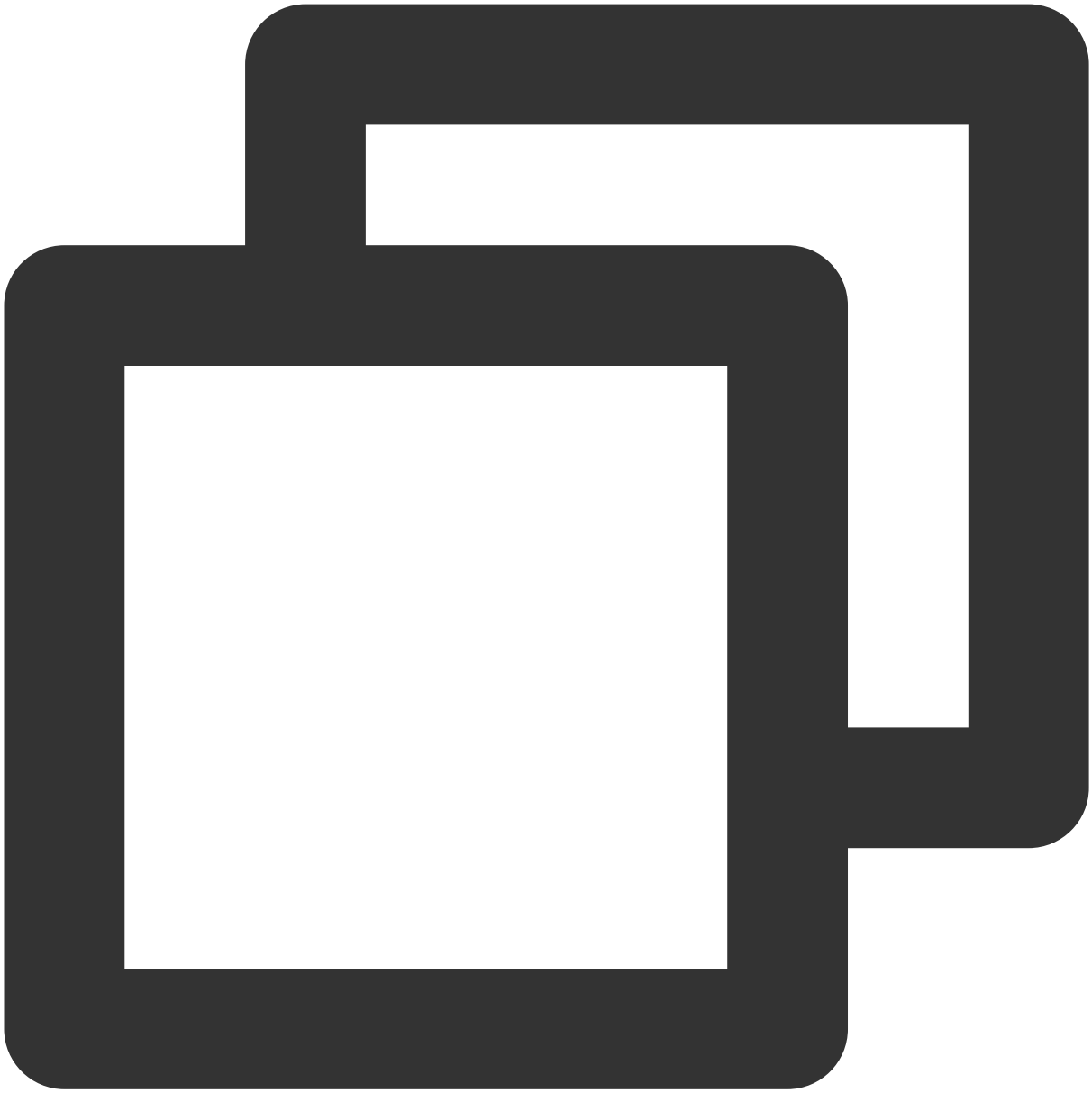
7、动态水印

播放器组件支持在播放界面添加不规则跑动的文字水印，有效防盗录。全屏播放模式和窗口播放模式均可展示水印，开发者可修改水印文本、文字大小、颜色。功能效果可在 **腾讯云视立方 App> 播放器 > 播放器组件 > 动态水印** 演示视频中体验。



方法一：

```
//步骤1：创建视频 mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建水印信息 mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, Co
mode.dynamicWaterConfig = dynamicWaterConfig;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModelNeedLicence(mode);

方法二：
//步骤1：创建水印信息 mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, Co
 //步骤2：调用设置动态水印功能方法
mSuperPlayerView.setDynamicWatermarkConfig(dynamicWaterConfig);
```



```
public DynamicWaterConfig(String dynamicWatermarkTip, int tipTextSize, int tipTextC
```

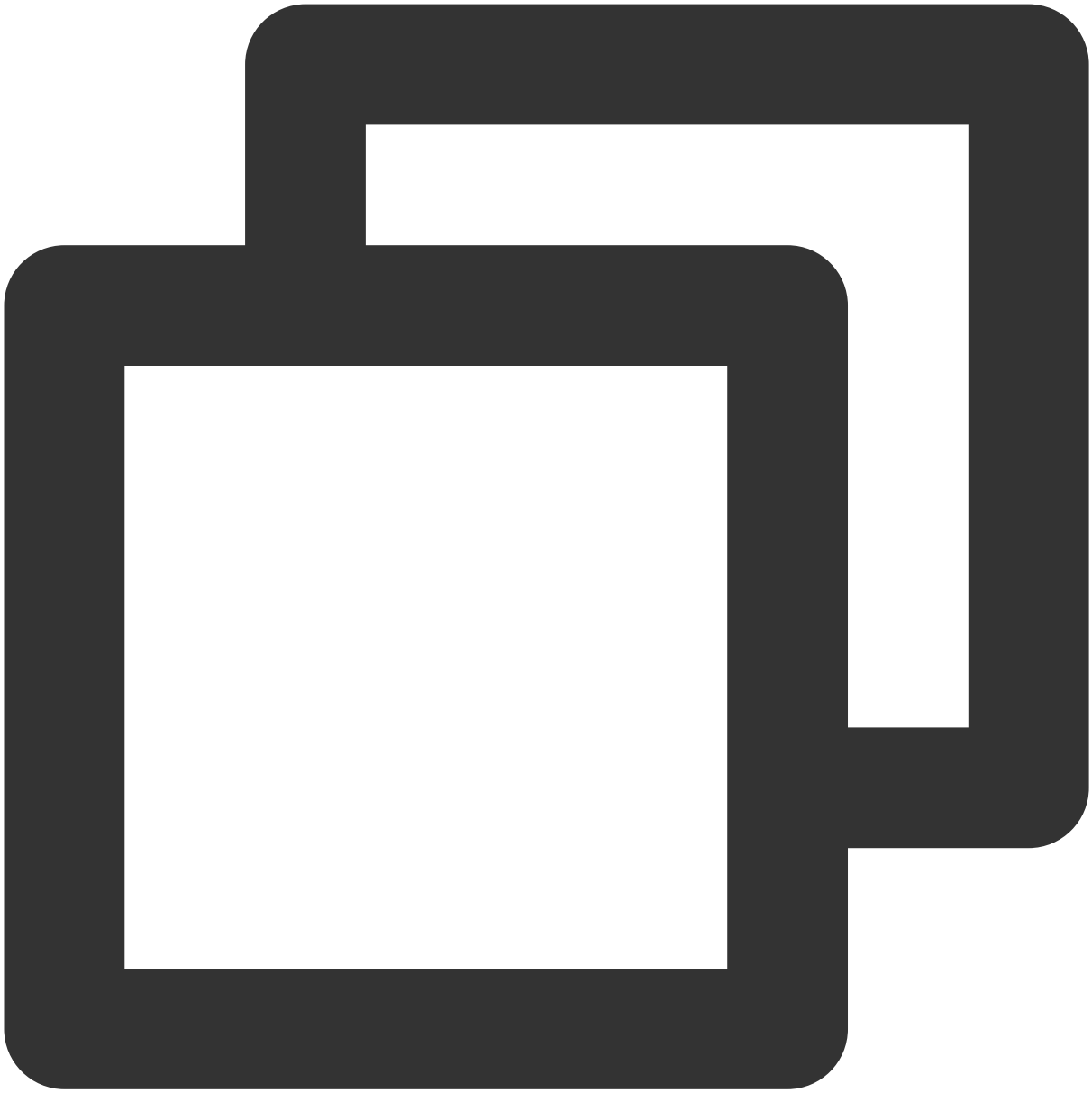
接口参数说明

参数名	类型	描述
dynamicWatermarkTip	String	水印文本信息
tipTextSize	int	文字大小
tipTextColor	int	文字颜色

8、视频下载

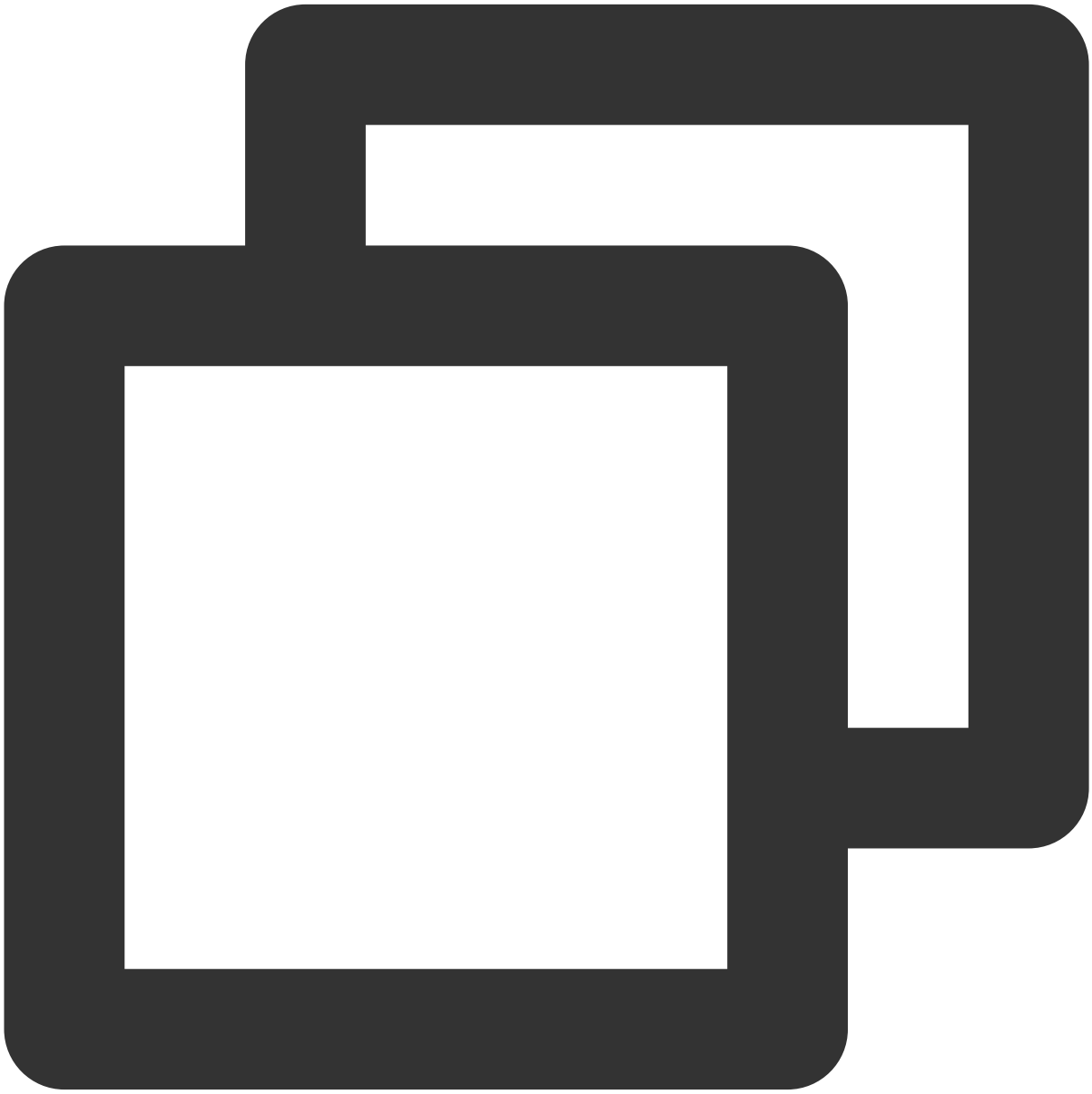
支持用户在有网络的条件下缓存视频，随后在无网络的环境下观看；同时离线缓存的视频仅可在客户端内观看，不可被下载至本地，可有效防止下载视频的非法传播，保护视频安全。
你可在 腾讯云视立方 App > 播放器 > 播放器组件 > 离线缓存（全屏）演示视频中，使用全屏观看模式后体验。

DownloadMenuListView（缓存选择列表视图），用于选择下载对应清晰度的视频。左上角选择清晰度后，再点击要下载的视频选项，出现对勾后，代表开始了下载。点击下方的 video download list 按钮后会跳转到 VideoDownloadListView 所在的 Activity。



```
// 步骤1：初始化下载数据 参数见下方列表
DownloadMenuListView mDownloadMenuView = findViewById(R.id.superplayer_cml_cache_me
mDownloadMenuView.initDownloadData(superPlayerModelList, mVideoQualityList, mDefaul

// 步骤2：设置正在播放的视频选项
mDownloadMenuView.setCurrentPlayVideo(mSuperplayerModel);

// 步骤3：设置 video download list 按钮的点击事件
mDownloadMenuView.setOnCacheListClick(new OnClickListener() {
     @Override
     public void onClick(View v) {
```

```
        // 跳转到 VideoDownloadListView 所在的 Activity
        startActivity(DownloadMeduListActivity.this,VideoDownloadListActivity.class)
      }
});

// 步骤4:通过动画展示 view
mDownloadMenuView.show();
```



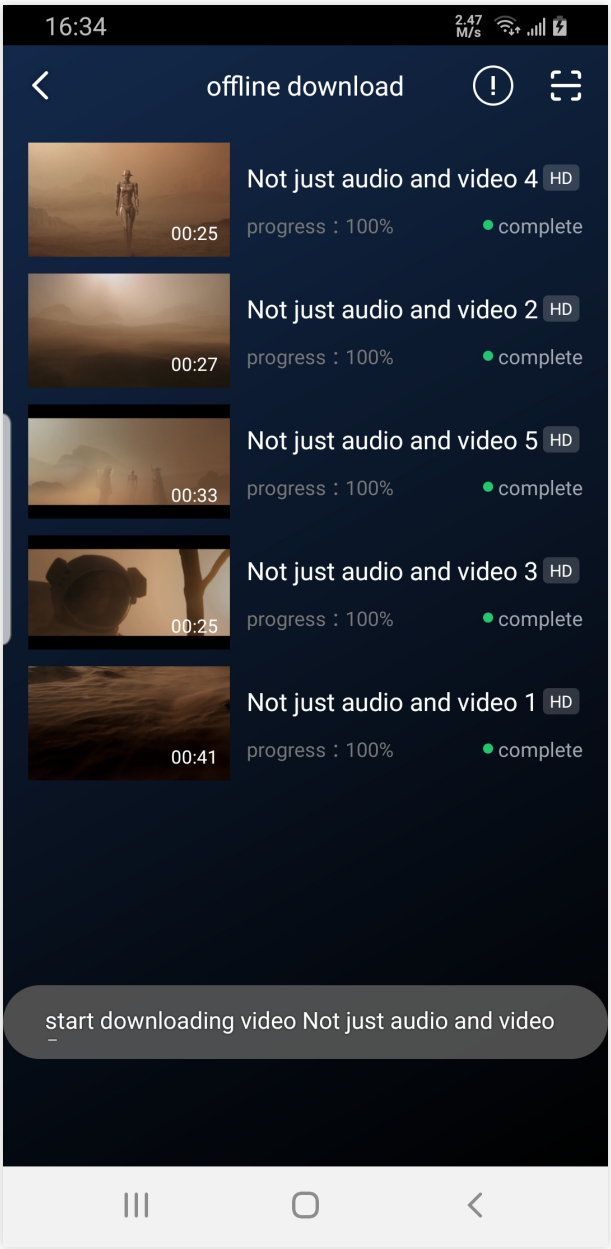
```
public void initDownloadData(List<SuperPlayerModel> superPlayerModelList,
                             List<VideoQuality> qualityList,
                             VideoQuality currentQuality,
```

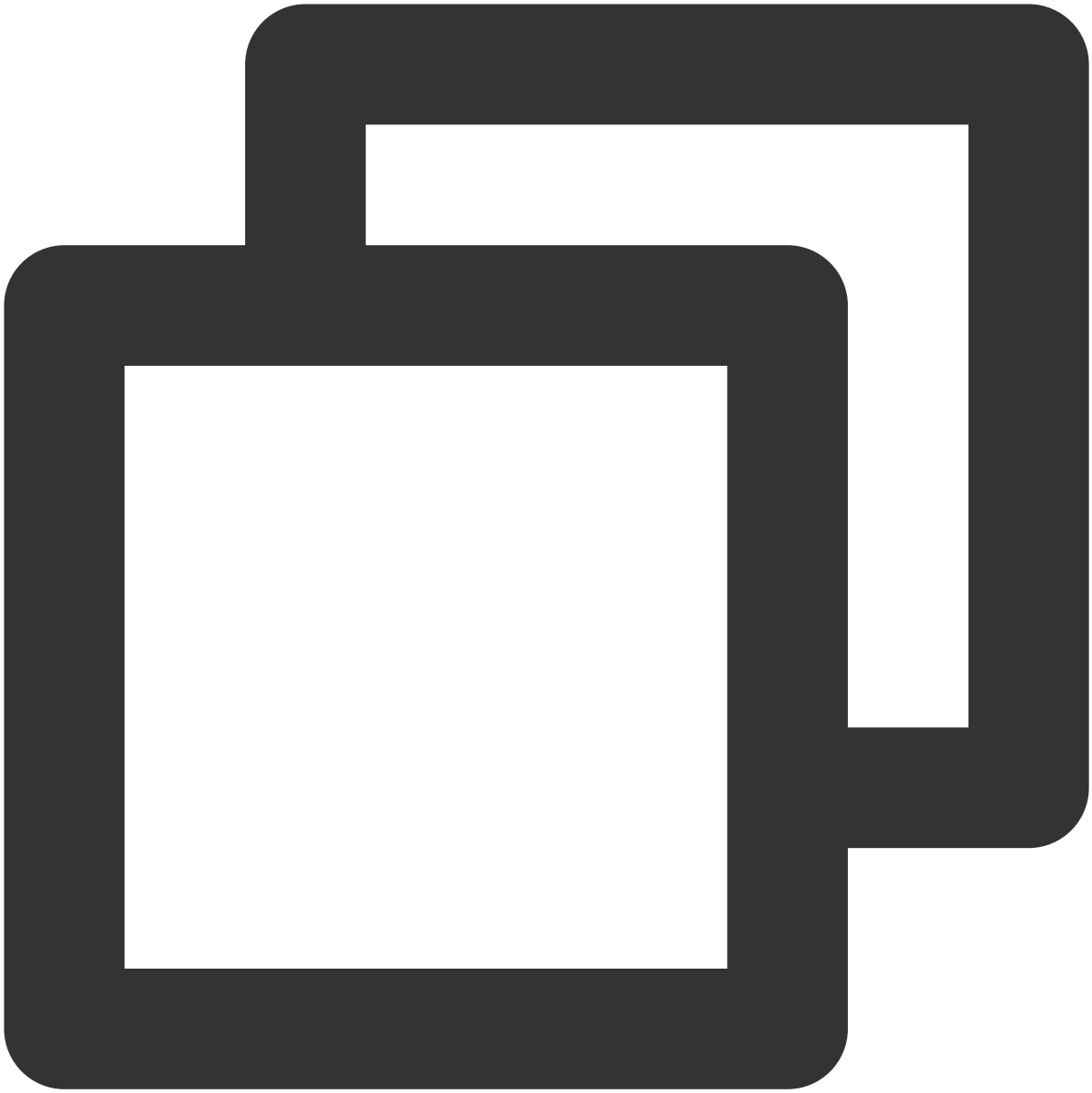
```
                                String userName)
```

接口参数说明

参数名	类型	描述
superPlayerModelList	List<SuperPlayerModel>	下载的视频数据
qualityList	List<VideoQuality>	视频清晰度数据
currentQuality	VideoQuality	当前的视频清晰度
userName	String	用户名

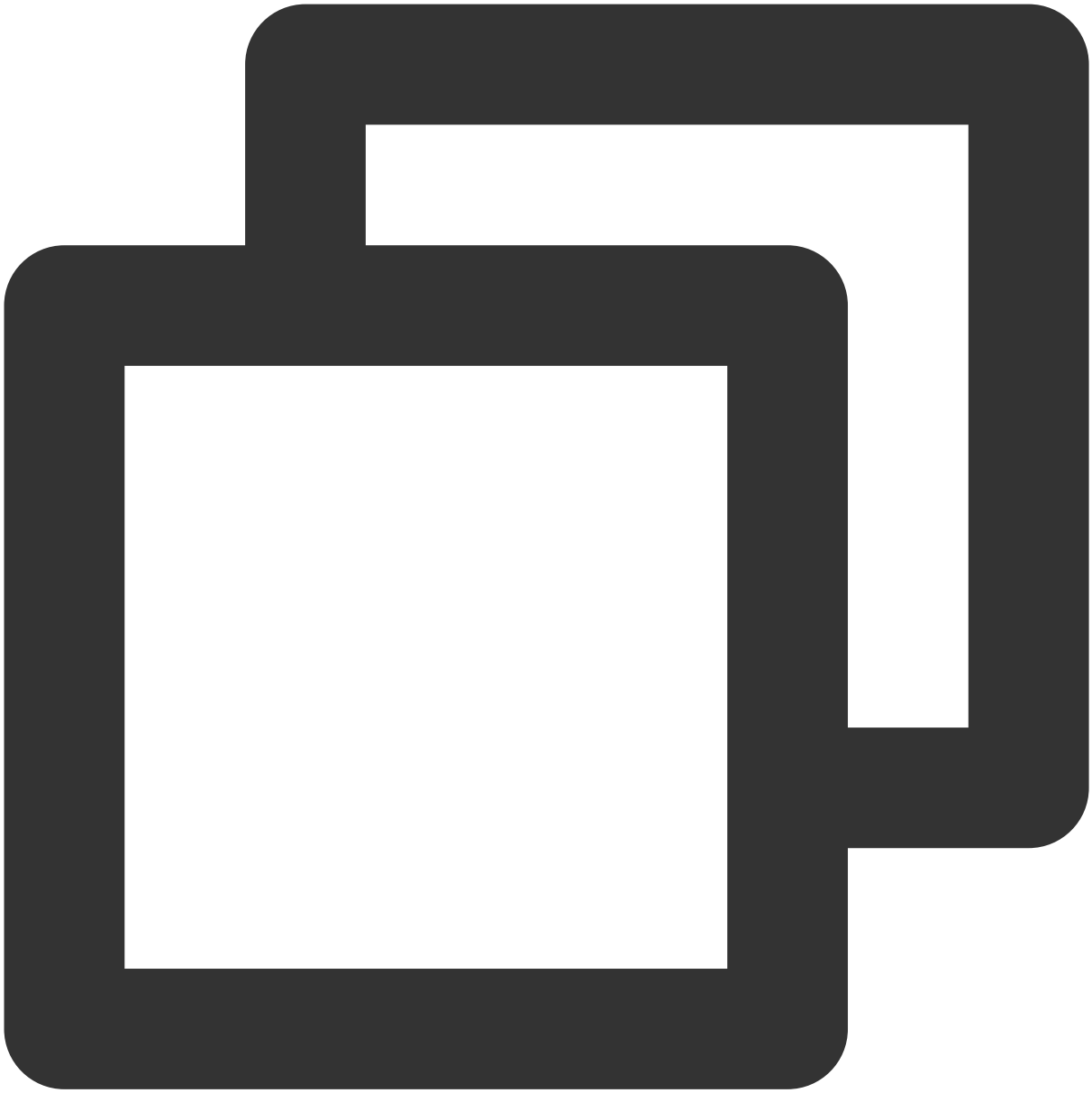
VideoDownloadListView（视频下载列表），显示所有正在下载的和下载完成视频的列表 View。点击时，如果正在下载，会暂停下载；如果暂时下载，会继续下载；如果下载完成，会跳转播放。


16:34
2.47 M/s
offline download
Not just audio and video 4 HD
00:25
progress：100%
complete
Not just audio and video 2 HD
00:27
progress：100%
complete
Not just audio and video 5 HD
00:33
progress：100%
complete
Not just audio and video 3 HD
00:25
progress：100%
complete
Not just audio and video 1 HD
00:41
progress：100%
complete
start downloading video Not just audio and video



```
// 步骤1：绑定控件
VideoDownloadListView mVideoDownloadListView = findViewById(R.id.video_download_lis

//步骤2: 添加数据
mVideoDownloadListView.addCacheVideo(mDataList, true);
```

接口参数说明：



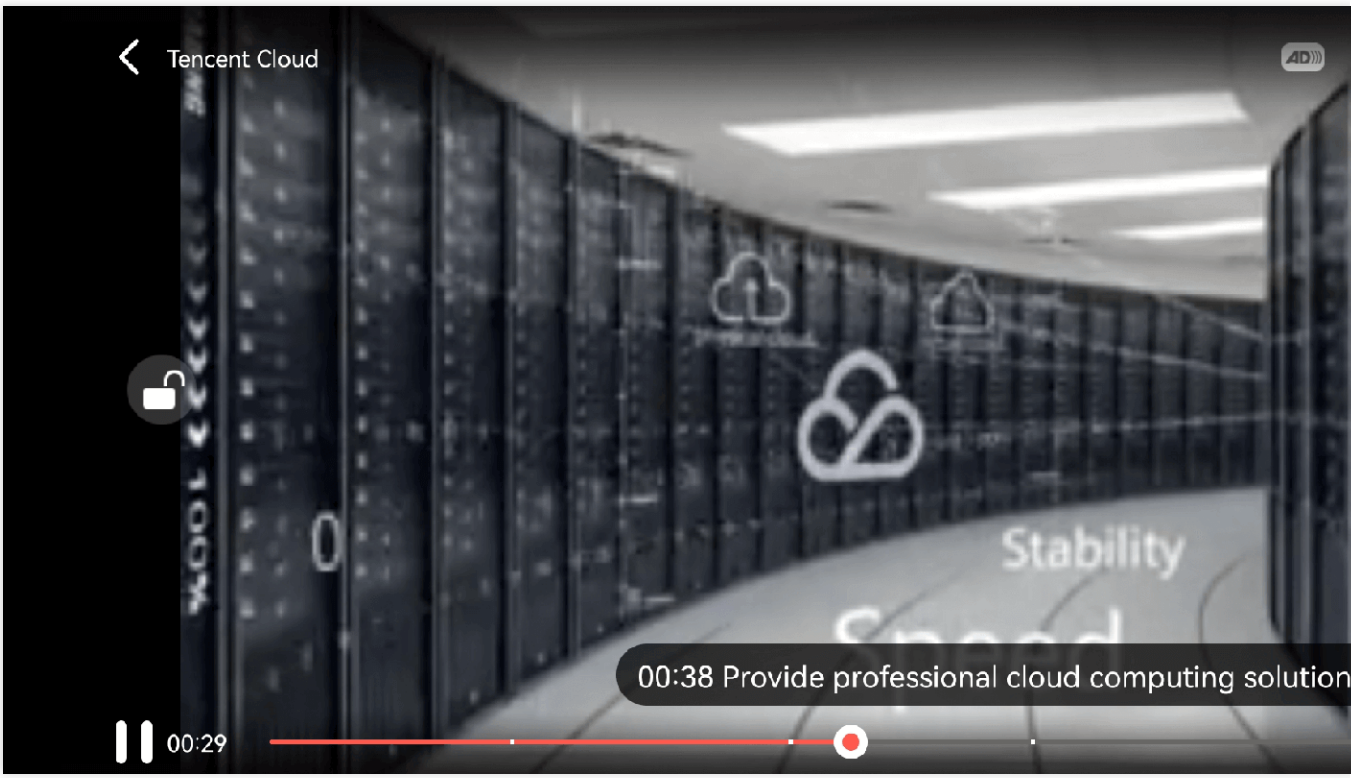
```
public void addCacheVideo(List<TXVodDownloadMediaInfo> mediaInfoList, boolean isNee
```

参数名	类型	描述
mediaInfoList	List<TXVodDownloadMediaInfo>	添加的视频数据类型
isNeedClean	boolean	是否清除之前的数据

9、雪碧图和打点信息

打点信息

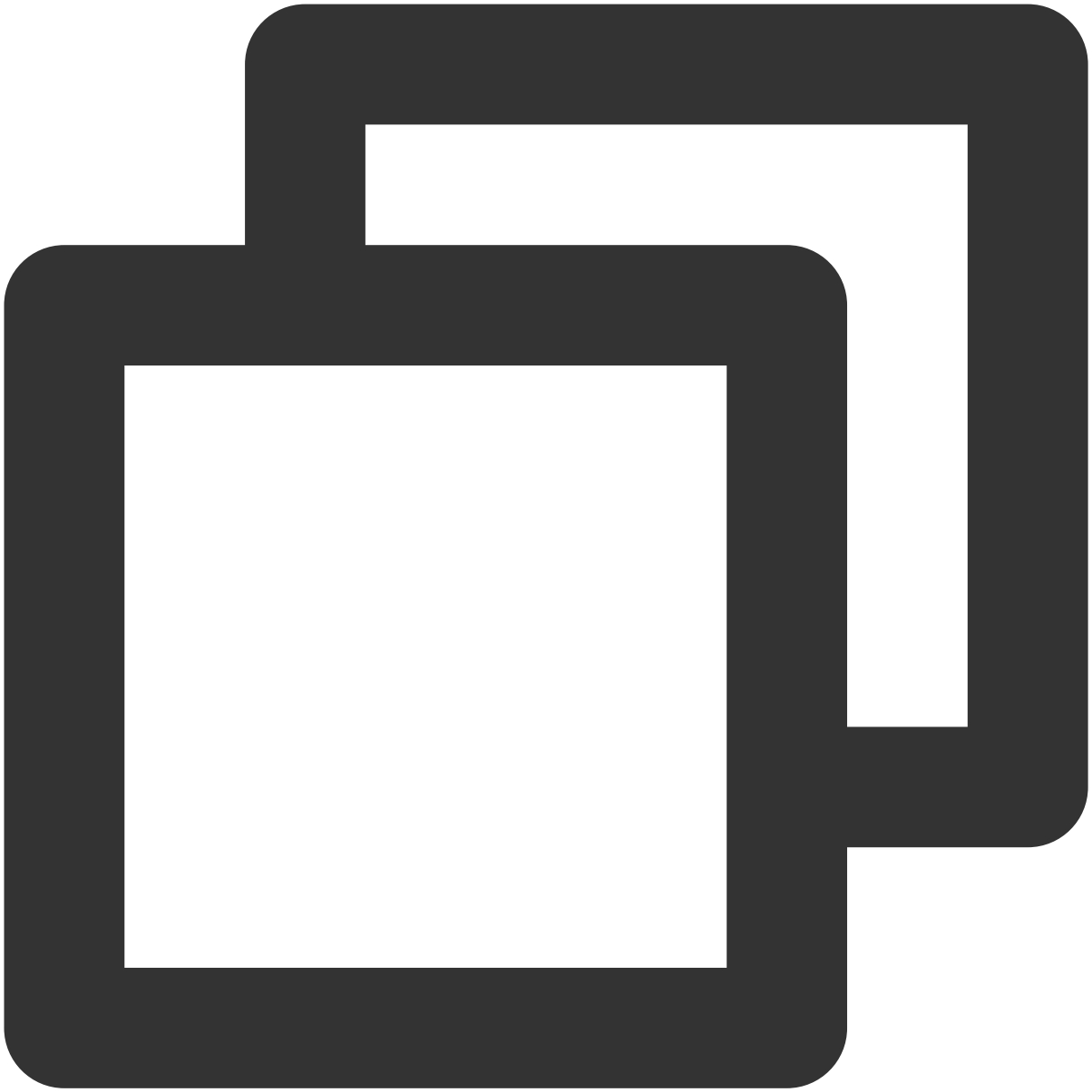
支持在进度条关键位置添加文字介绍，用户点击后可显示打点位置的文字信息，以快速了解当前位置的视频信息。点击视频信息后，可以 seek 到打点信息位置。
您可在腾讯云视立方 App > 播放器 > 播放器组件 > 腾讯云 视频中，使用全屏观看模式后体验。



雪碧图

支持用户在拖拽进度条或执行快进操作时查看视频缩略图，以快速了解指定进度的视频内容。缩略图预览基于视频雪碧图实现，您可以在云点播控制台中生成视频文件雪碧图，或直接生成雪碧图文件。
您可在腾讯云视立方 App > 播放器 > 播放器组件 > 腾讯云 视频中，使用全屏观看模式后体验。


Tencent Cloud
Speed
00:30 / 01:00
00:44



```
// 步骤1：播放视频 superplayerModel的url 变量需要为空，且 videoId 不为空，这样才会通过 PlayW
mSuperplayerView.play(superplayerModel);

// 步骤2: PlayWithFileId 播放时候 在 VOD_PLAY_EVT_GET_PLAYINFO_SUCC 回调事件中取得打点信息
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    switch (event) {
        case TXVodConstants.VOD_PLAY_EVT_GET_PLAYINFO_SUCC:

            // 获取 雪碧图 图片链接 URL
            playImageSpriteInfo.imageUrls = param.getStringArrayList(TXVodConstants
            // 获取 雪碧图 web vtt 描述文件下载 URL
```

```
            playImageSpriteInfo.webVttUrl = param.getString(TXVodConstants.EVT_IMAG
            // 获取 打点信息
            ArrayList<String> keyFrameContentList =
                    param.getStringArrayList(TXVodConstants.EVT_KEY_FRAME_CONTENT_L
            // 获取 打点信息时间信息
            float[] keyFrameTimeArray = param.getFloatArray(TXVodConstants.EVT_KEY_

            // 构建 打点信息数据列表
            if (keyFrameContentList != null && keyFrameTimeArray != null
                    && keyFrameContentList.size() == keyFrameTimeArray.length) {
                for (int i = 0; i < keyFrameContentList.size(); i++) {
                    PlayKeyFrameDescInfo frameDescInfo = new PlayKeyFrameDescInfo()
                    frameDescInfo.content = keyFrameContentList.get(i);
                    frameDescInfo.time = keyFrameTimeArray[i];
                    mKeyFrameDescInfoList.add(frameDescInfo);
                }
            }
            break;
        default:
            break;
    }
}

// 步骤3：将拿到的打点信息和雪碧图信息通过 updateVideoImageSpriteAndKeyFrame 方法赋值给对应
// 雪碧图的 view 对应 VideoProgressLayout 组件中 mIvThumbnail。
// 打点信息的 view 对应 PointSeekBar 组件中的 TCPointView。
updateVideoImageSpriteAndKeyFrame(playImageSpriteInfo,keyFrameDescInfoList);
```

10、外挂字幕

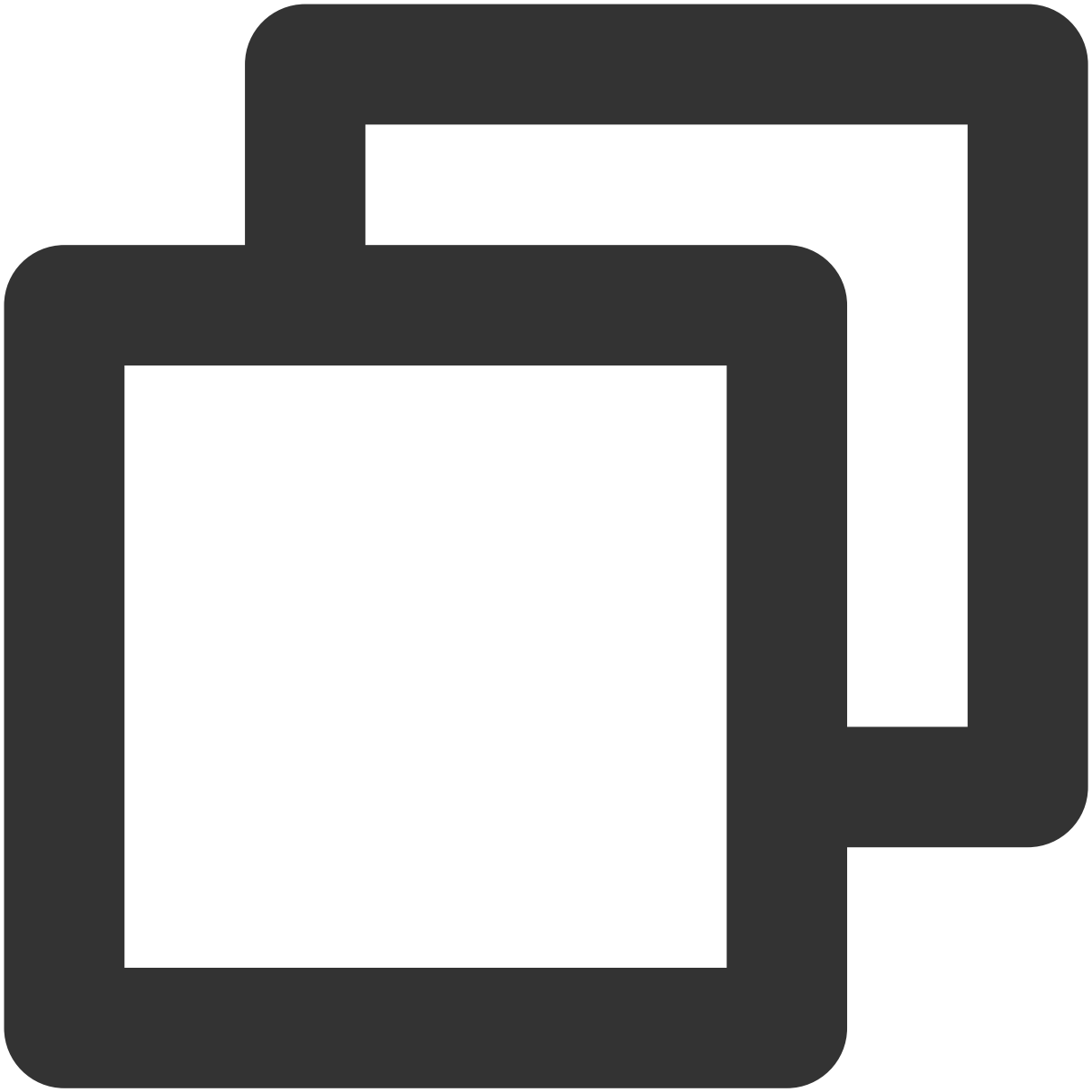
注意：

外挂字幕依赖播放器高级版本SDK 且 SDK 需要11.3 版本以上才支持。

目前支持 SRT 和 VTT 这两种格式的字幕。用法如下：

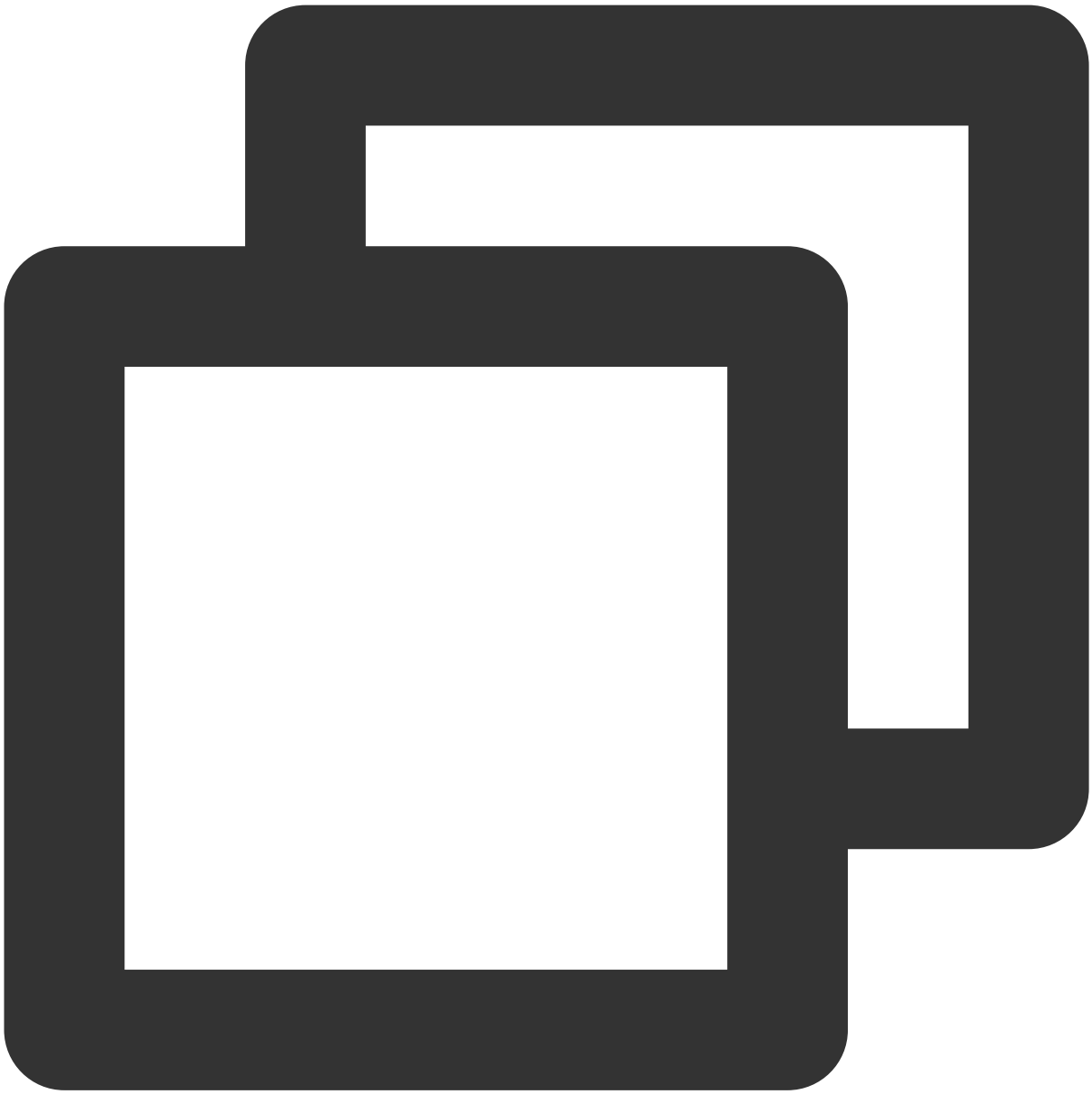
步骤1：添加外挂字幕。

往 `SuperPlayerModel#subtitleSourceModelList` 传入外挂字幕类别字段。



```
// 传入 字幕url，  字幕名称，  字幕类型
SubtitleSourceModel subtitleSourceModel = new SubtitleSourceModel();
subtitleSourceModel.name = "ex-cn-srt";
subtitleSourceModel.url = "https://mediacloud-76607.gzc.vod.tencent-cloud.com/DemoR
subtitleSourceModel.mimeType = TXVodConstants.VOD_PLAY_MIMETYPE_TEXT_SRT;
model.subtitleSourceModelList.add(subtitleSourceModel);


// 播放
mSuperPlayerView.playWithModelNeedLicence(model);
```

步骤2：播放后切换字幕。

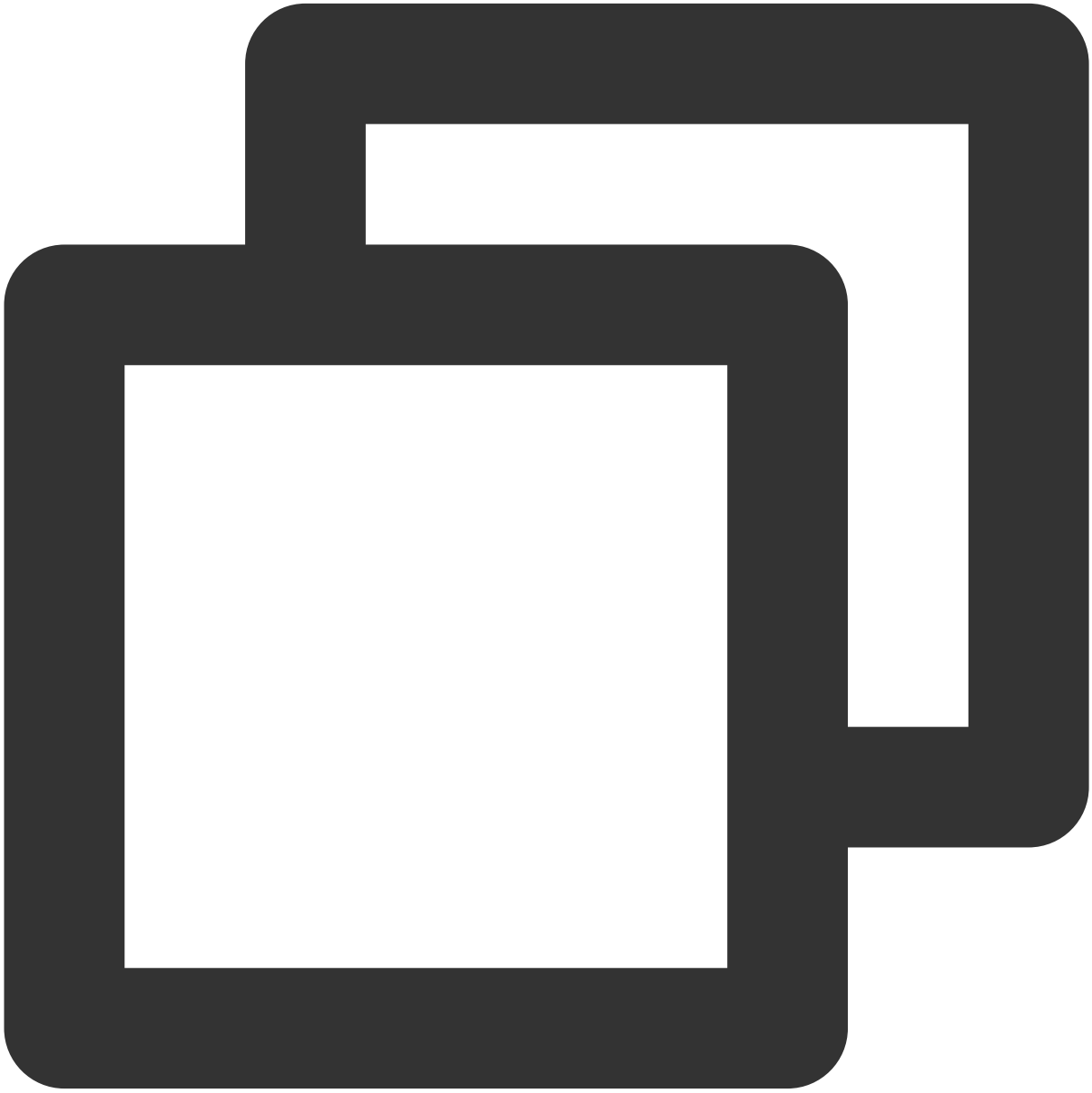


```
// 开始播放视频后，选中添加的外挂字幕
public void onClickSubTitleItem(TXTrackInfo clickInfo) {
    List<TXTrackInfo> subtitleTrackInfoList = mVodPlayer.getSubtitleTrackInfo();
    for (TXTrackInfo trackInfo : subtitleTrackInfoList) {
        if (trackInfo.trackIndex == clickInfo.trackIndex) {
            // 选中字幕
            mVodPlayer.selectTrack(trackInfo.trackIndex);
            mSelectedSubtitleTrackInfo = trackInfo;
        } else {
```

```
            // 其它字幕不需要的话，  进行deselectTrack
            mVodPlayer.deselectTrack(trackInfo.trackIndex);
        }
    }
}
```

步骤3：配置字幕样式。

字幕样式支持在播放前或者播放过程中配置。



```
TXSubtitleRenderModel model = new TXSubtitleRenderModel();
model.canvasWidth = 1920;  // 字幕渲染画布的宽
```

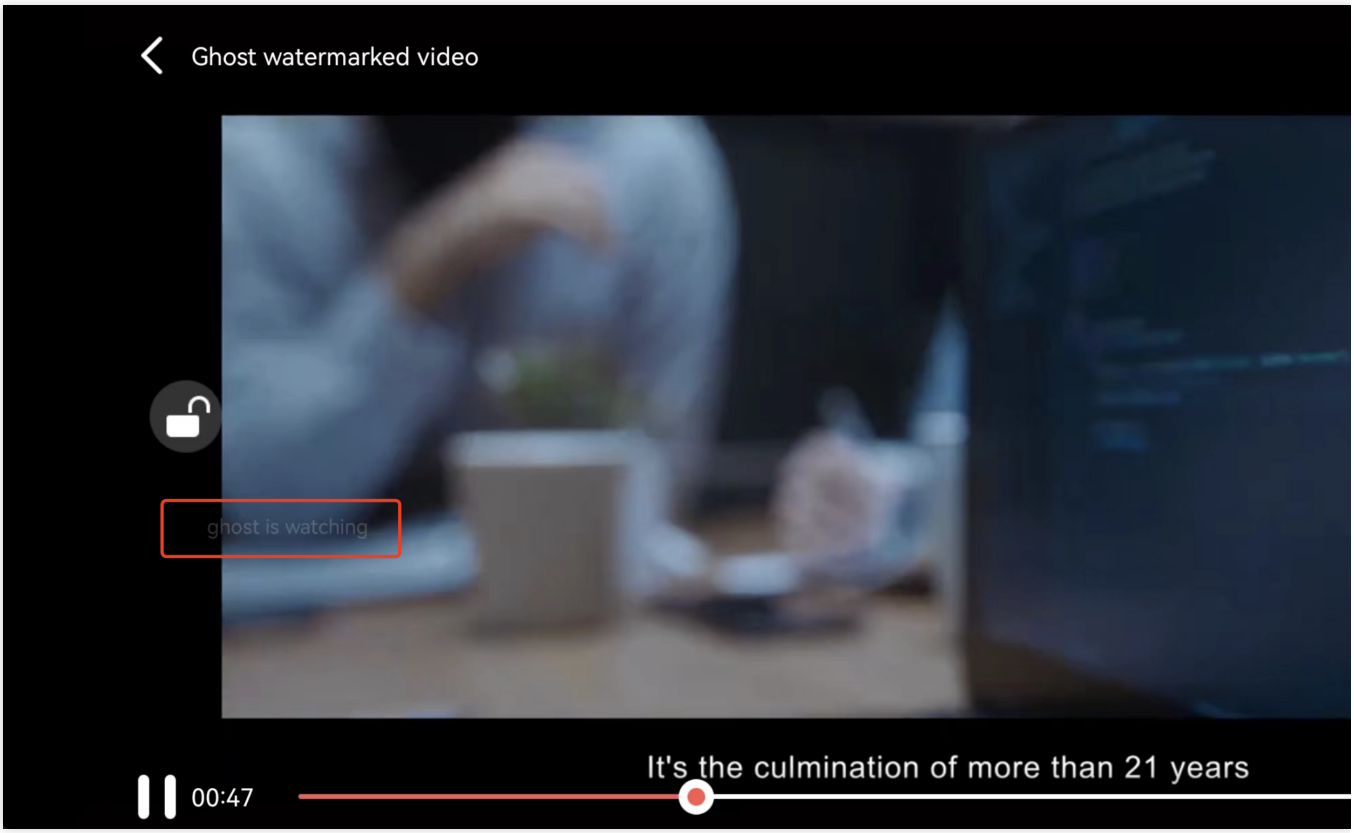
```
model.canvasHeight = 1080;  // 字幕渲染画布的高
model.fontColor = 0xFFFFFFFF; // 设置字幕字体颜色，默认白色不透明
model.isBondFontStyle = false;  // 设置字幕字体是否为粗体
mVodPlayer.setSubtitleStyle(model);
```

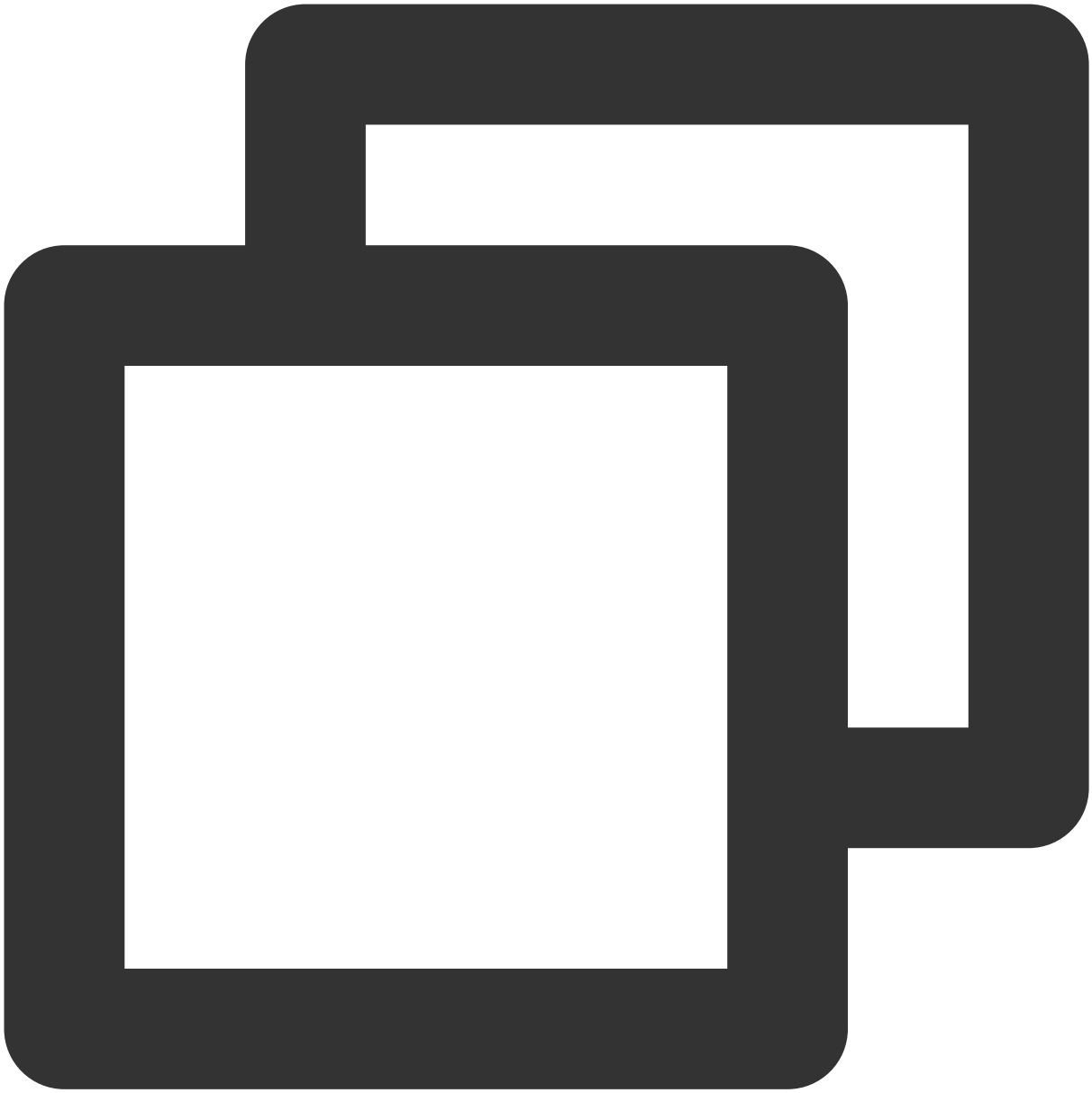
11、幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程。幽灵水印仅在视频上出现一段很短的时间，这种闪现对视频的观看影响很微小。每次水印出现的画面位置都不固定，杜绝了他人遮挡水印的企图。效果如下图所示，在视频开始播放时，就会出现一次水印，然后消失。等到下一次再出现，再消失。

幽灵水印的内容在收到播放器的 `TXVodConstants#VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `param.getString(TXVodConstants.EVT_KEY_WATER_MARK_TEXT)` 获取。

注意： 播放器 11.6 版本开始支持。





```
// 步骤 1： 配置支持幽灵水印的 FileId 播放视频
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1500006438;
model.videoId.fileId = "387702307847129127";
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDA"
                    + "wNjQzOCwiZmlsZUlkIjoiMzg3NzAyMzA3ODQ3MTI5MTI3IiwiY29udG"
                    + "VudEluZm8iOnsiYXVkaW9WaWRlb1R5cGUiOiJSYXdBZGFwdGl2ZSIsIn"
                    + "Jhd0FkYXB0aXZlRGVmaW5pdGlvbiI6MTB9LCJjdXJyZW50VGltZVN0YW1w"
                    + "IjoxNjg2ODgzMzYwLCJnaG9zdFdhdGVybWFya0luZm8iOnsidGV4dCI6I"
                    + "mdob3N0IGlzIHdhdGNoaW5nIn19.0G2o4P5xVZ7zF"
```

```
                    + "lFUgBLntfX03iGxK9ntD_AONClUUno";
mSuperPlayerView.playWithModelNeedLicence(model);

// 步骤 2： 在 SuperPlayerView#onRcvWaterMark 中收到幽灵水印内容回调后，展示幽灵水印
public void onRcvWaterMark(String text, long duration) {
    if (!TextUtils.isEmpty(text)) {
        DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig(text, 30, Co
        dynamicWaterConfig.durationInSecond = duration;
        dynamicWaterConfig.setShowType(DynamicWaterConfig.GHOST_RUNNING);
        setDynamicWatermarkConfig(dynamicWaterConfig);
    }
}
```

Demo 体验

更多完整功能可直接运行工程 Demo，或扫码下载移动端 Demo 腾讯云视立方 App 体验。

运行工程 Demo

1. 在 Android Studio 的导航栏选择 **File** > **Open**，在弹框中选择 **Demo** 工程目录：`$SuperPlayer_Android/Demo`，待成功导入 Demo 工程后，单击 **Run app**，即可成功运行 Demo 。
2. 成功运行 Demo 后如下图，进入**播放器** > **播放器组件**，可体验播放器功能。

Flutter 接入指引

最近更新时间：2024-04-11 16:11:38

SDK 下载

腾讯云 Flutter 播放器 SDK 的地址是 Player Flutter。

阅读对象

本文档部分内容为腾讯云专属能力，使用前请开通 腾讯云 相关服务，未注册用户可注册账号 免费试用。

通过本文你可以学会

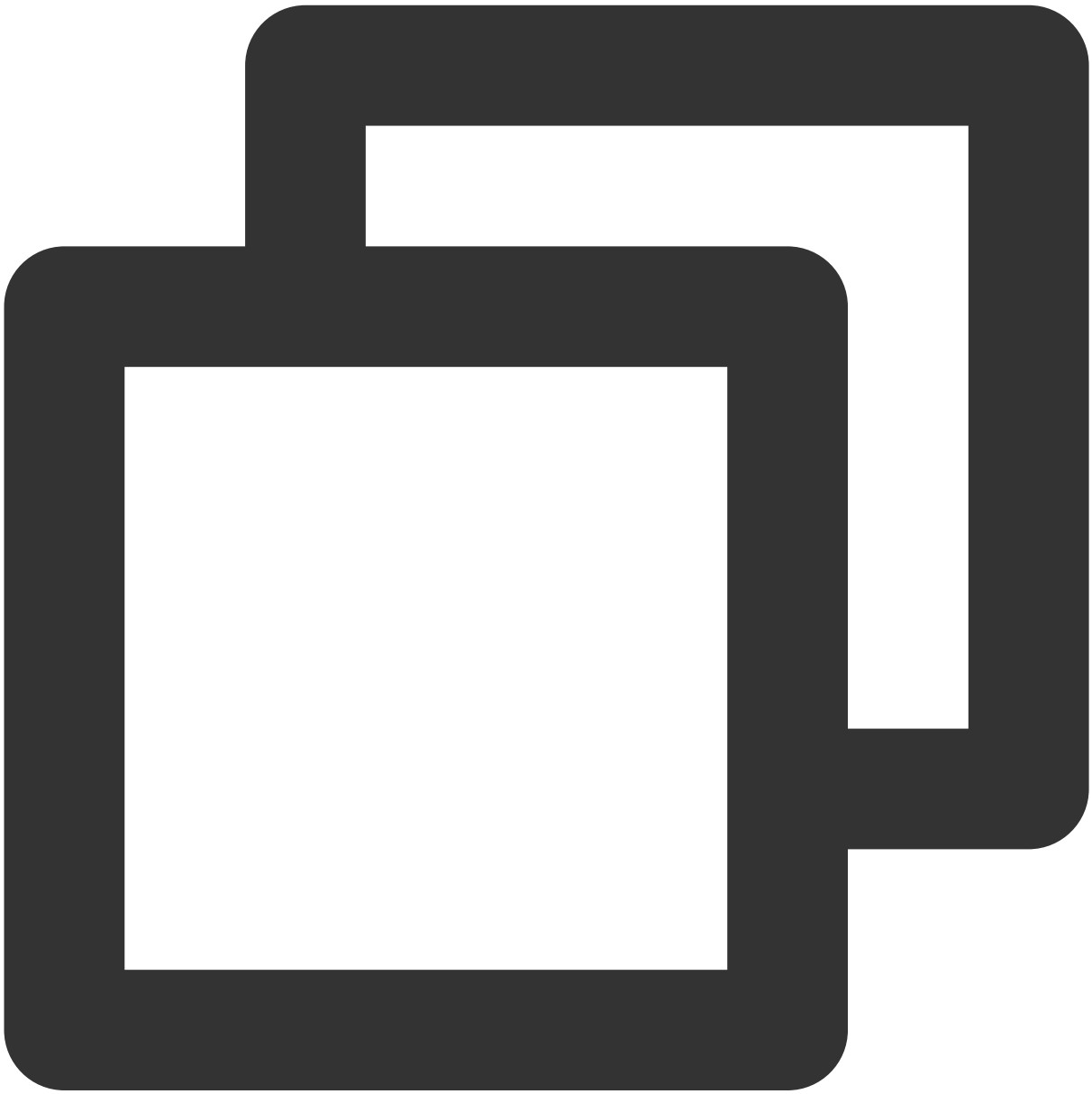
如何集成腾讯云 Flutter 播放器 SDK。
如何使用播放器组件进行点播播放。

播放器组件简介

Flutter 播放器组件是基于 Flutter 播放器 SDK 的扩展，播放器组件对于点播播放器，集成了更多的功能，包括全屏切换、清晰度切换、
进度条、播放控制、封面标题展示等常用功能，并且相对于点播播放器使用起来更加方便，如果想更加方便快捷的集成 Flutter 视频播放能力，可以选择 Flutter 播放器组件使用。
支持功能列表：
全屏播放
播放过程中屏幕旋转自适应
自定义视频封面
清晰度切换
声音和亮度调节
倍速播放
硬件加速开启\\关闭
画中画（PIP）（支持 Android 和 iOS 平台）
雪碧图和关键帧打点信息
更多功能正在逐步开发中.....

集成指引

1. 将项目中 `superplayer_widget` 目录复制到自己的 flutter 工程下。
2. 在自己项目的配置文件 `pubspec.yaml` 下添加依赖。



```
superplayer_widget:
    # 该路径根据superplayer_widget存放路径改变
    path: ../superplayer_widget
super_player:
  git:
```

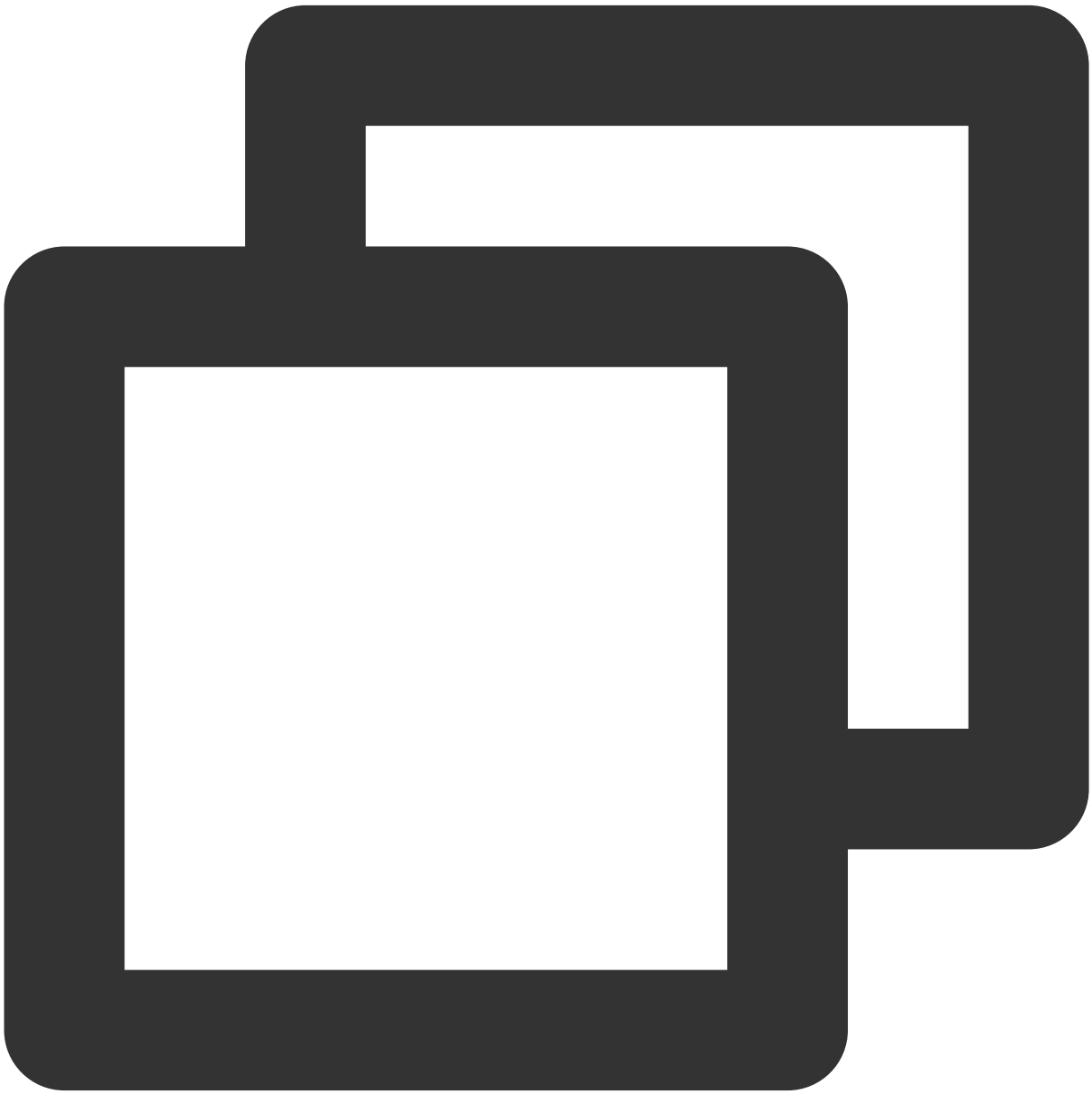
```
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: main
```

ref 可以根据自身项目需要，替换为对应的版本或分支。

3. 修改 superplayer_widget 的 `superPlayer` 依赖。

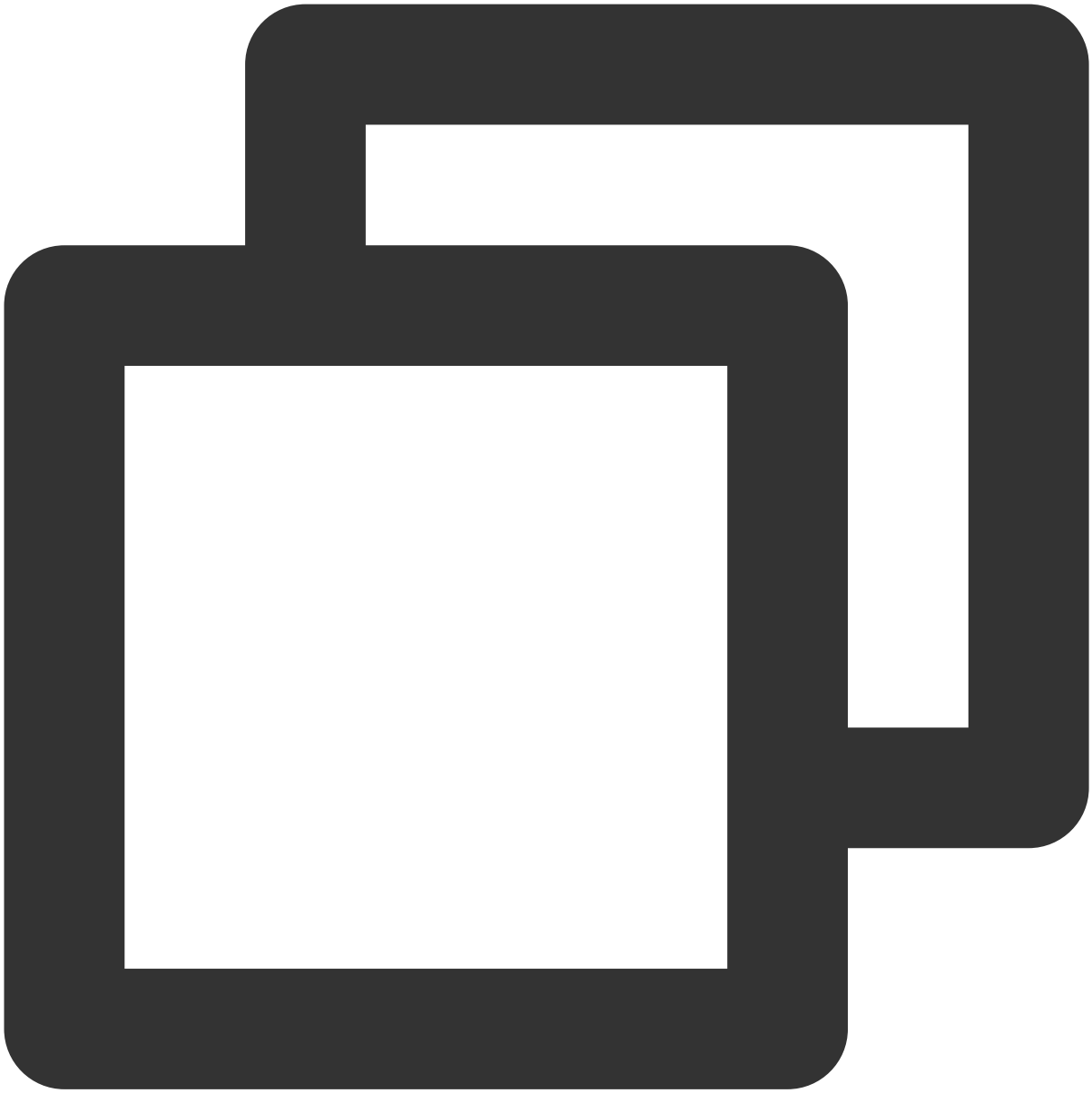
进入修改 superplayer_widget 的 `pubspec.yaml` 。

将如下配置进行替换：



```
super_player:
  path: ../
```

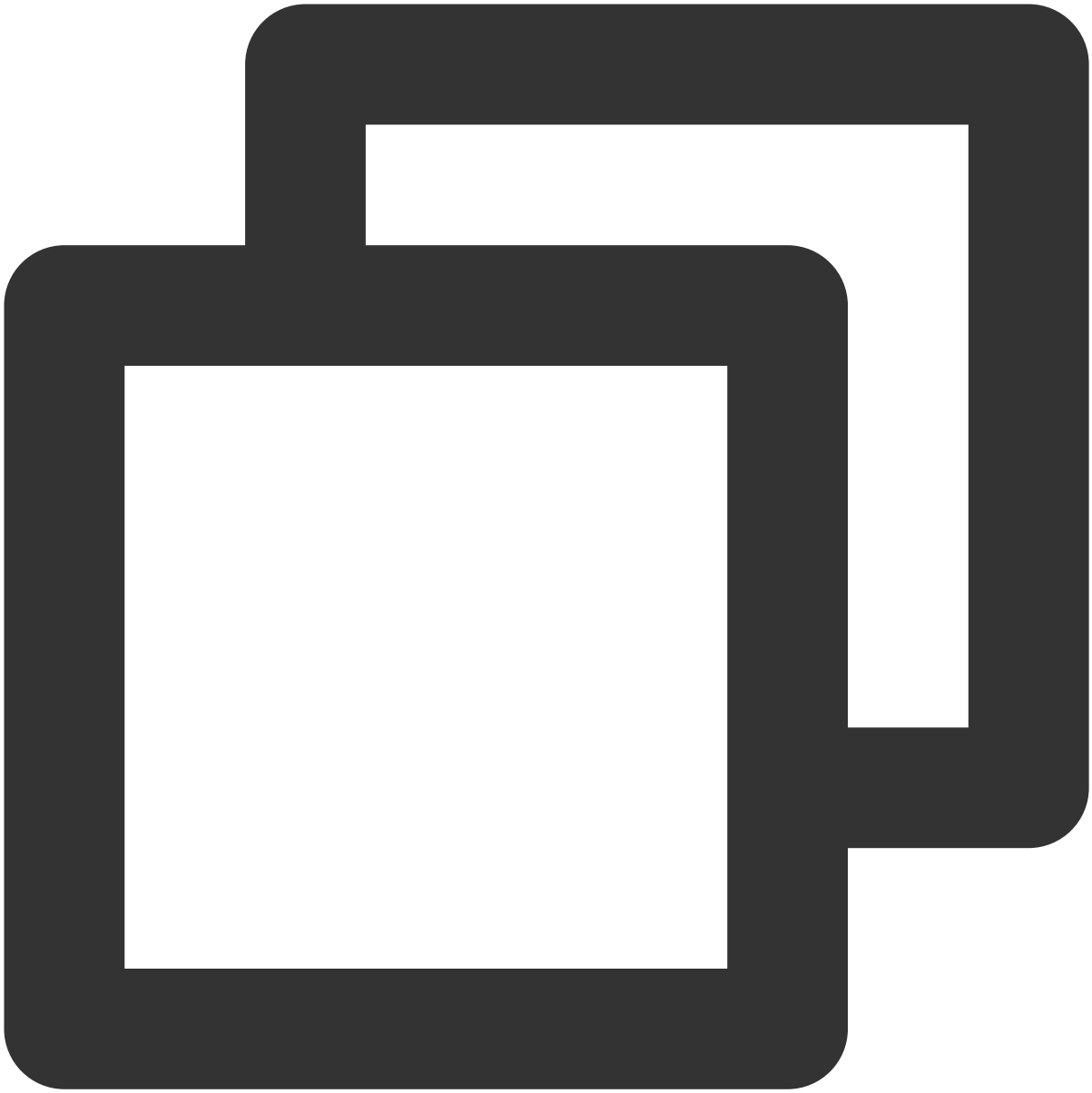
替换为：



```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: main
```

ref 可以根据自身项目需要，替换为对应的版本或分支。

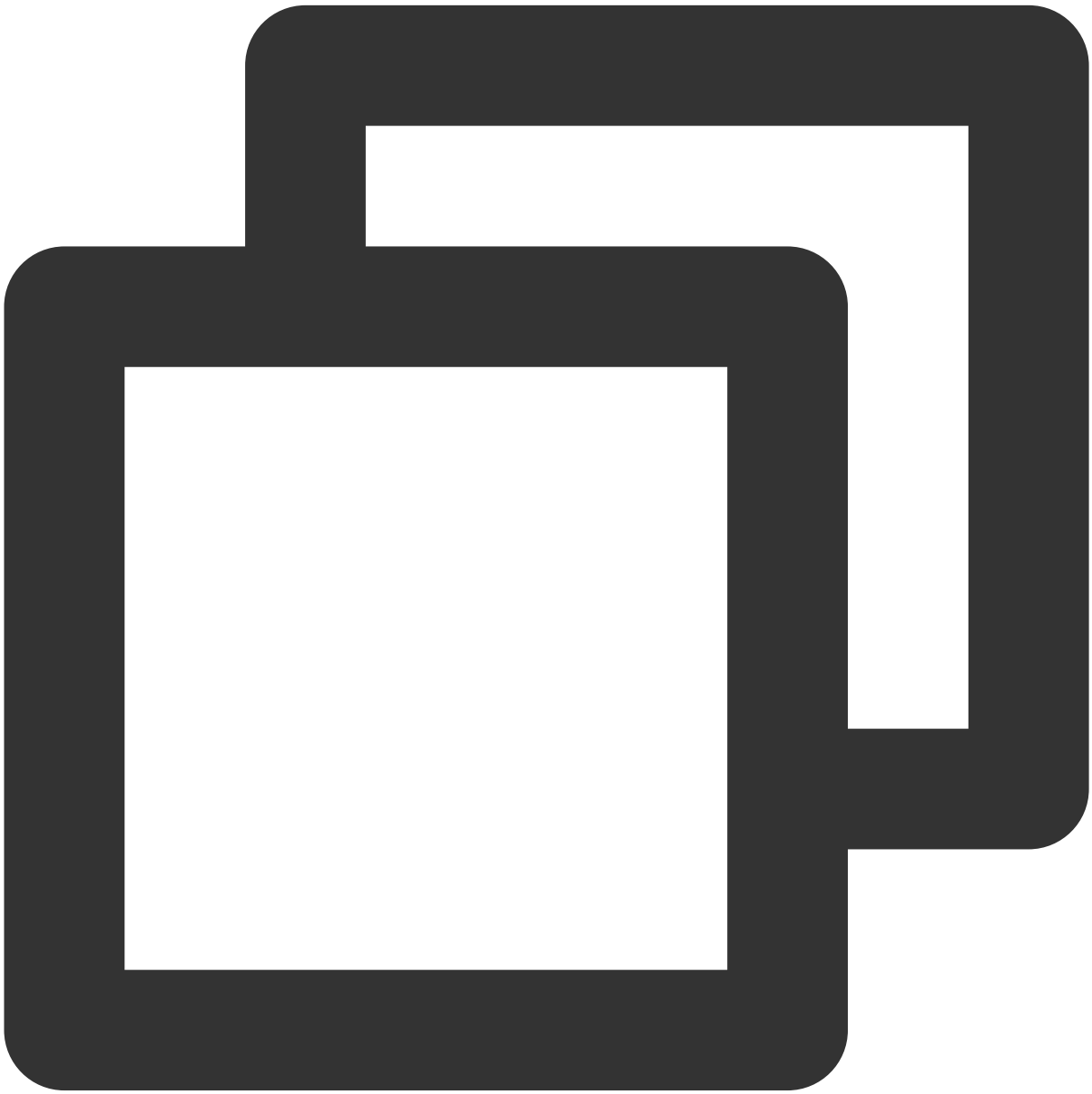
4. 由于目前播放器组件接入了国际化，需要在入口函数中添加国际化组件，如下示例：



```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    localizationsDelegates: [
      SuperPlayerWidgetLocals.delegate,
      // ..... your app other delegate
    ],
    supportedLocales: [
      Locale.fromSubtags(languageCode: 'en'),
      Locale.fromSubtags(languageCode: 'zh'),
```

```
        // ....... other language
      ],
      // ...... your app other code
    );
  }
```

5. 在需要使用到的页面，导入 `superplayer_widget` 的依赖包，如下所示：



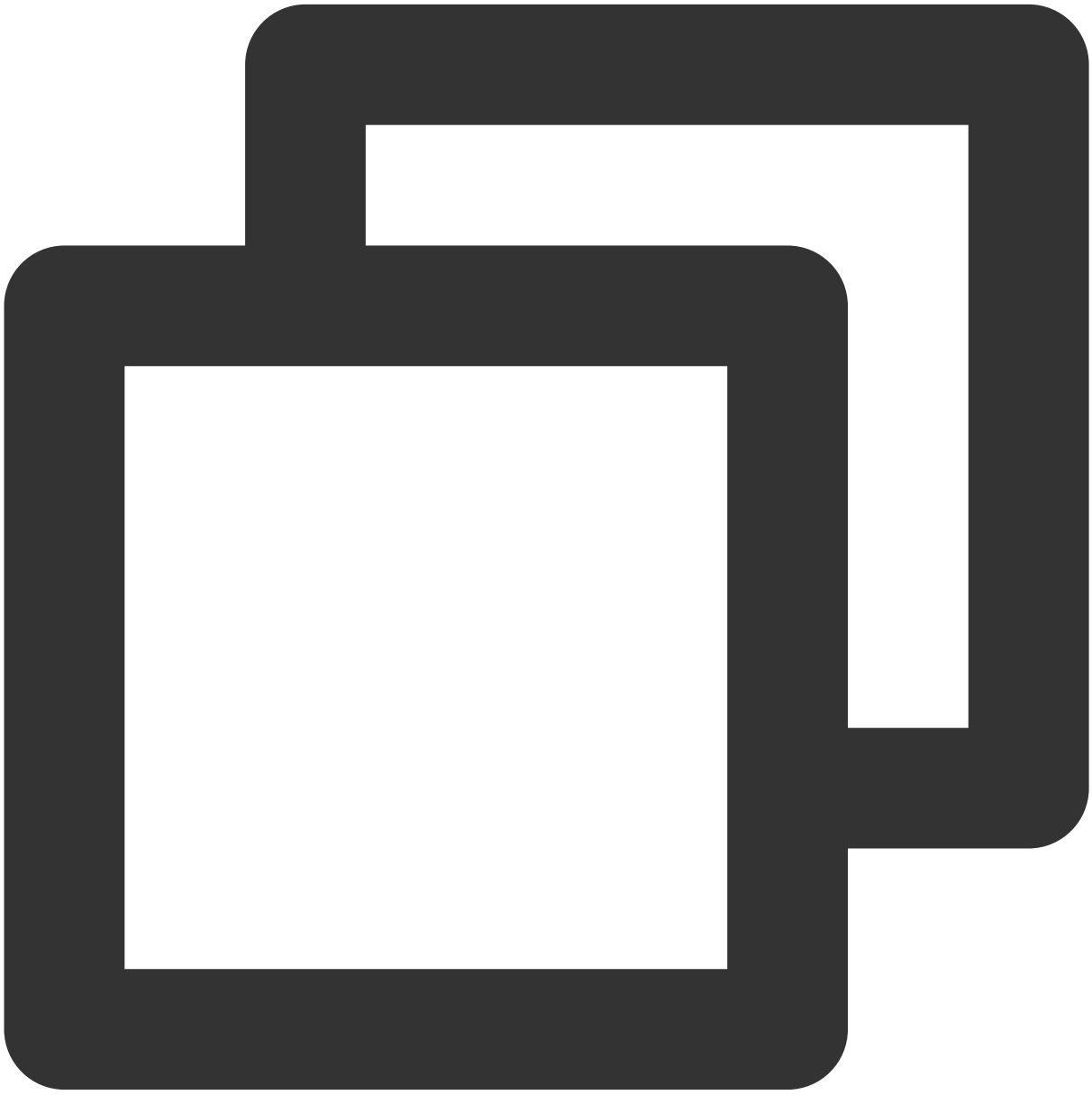
```
import 'package:superplayer_widget/demo_superplayer_lib.dart';
```

6. 其余原生相关配置，可以参考 集成指引。

SDK 集成

步骤1：申请视频播放能力 License 和集成

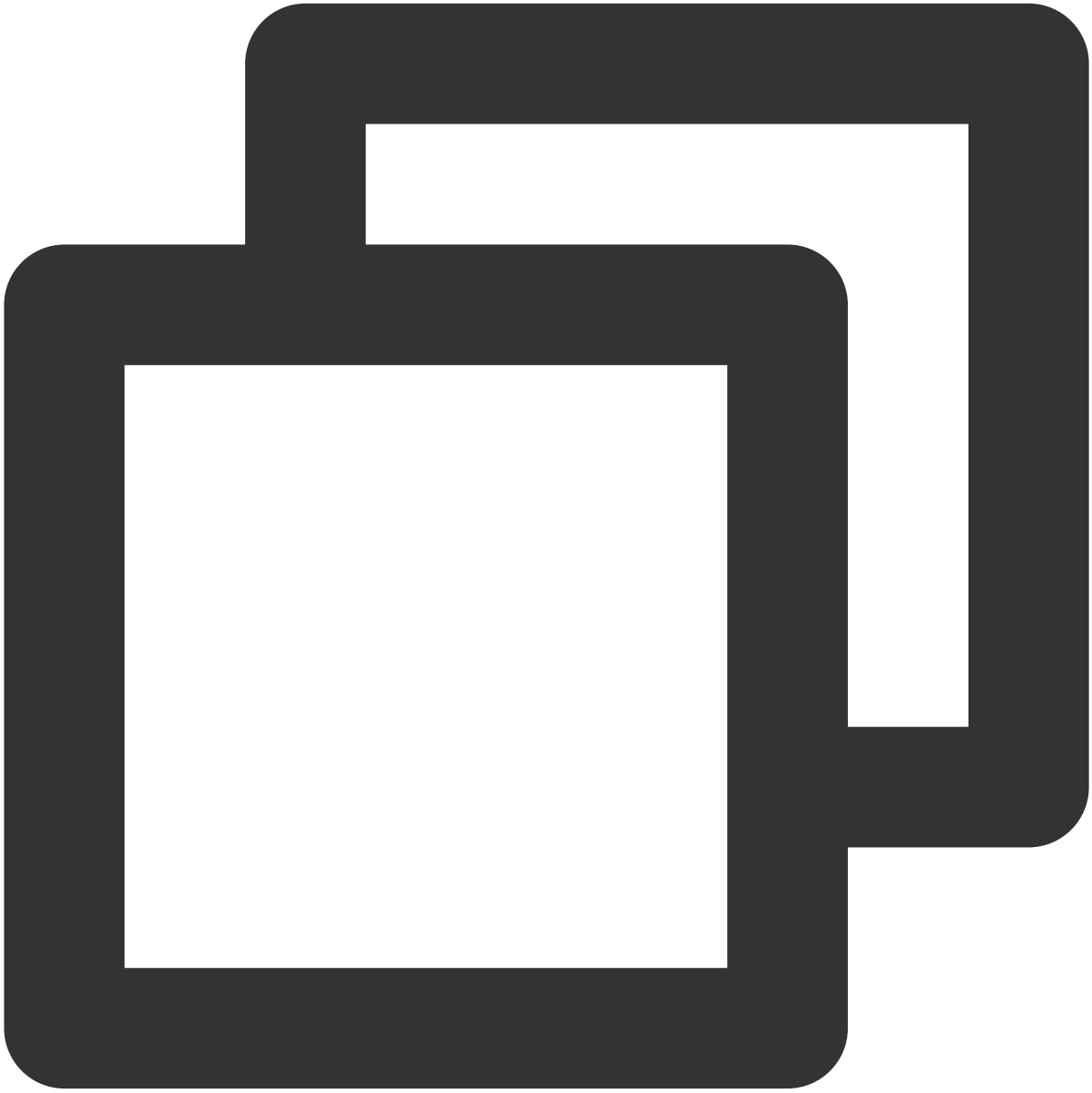
集成播放器前，需要 注册腾讯云账户，注册成功后申请视频播放能力 License， 然后通过下面方式集成，建议在应用启动时进行。
如果没有集成 License，播放过程中可能会出现异常。



```
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
```

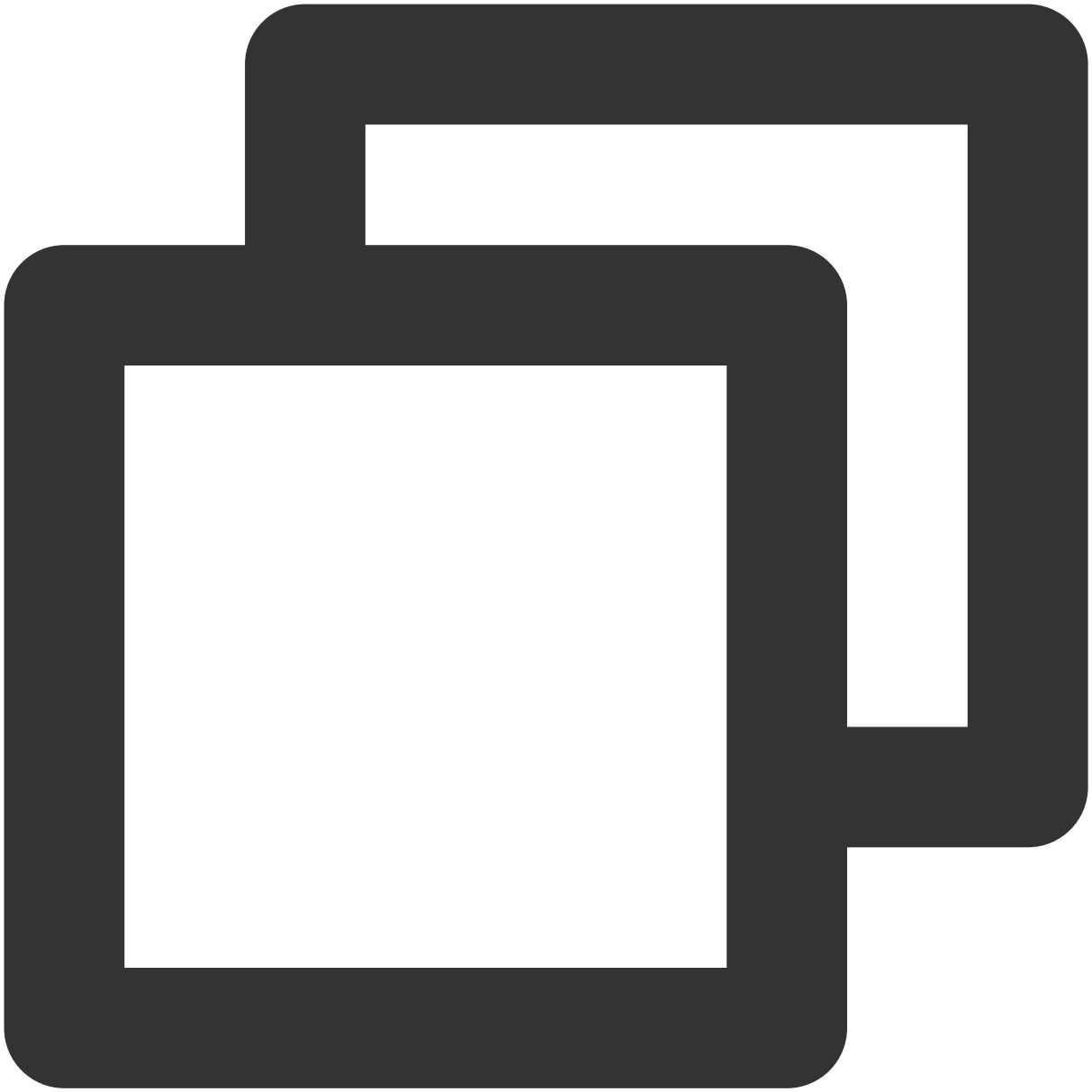
步骤2：设置 SDK 接入环境

为服务客户更高质量、更安全合规地开展业务，符合各国家和地区的法律法规要求，腾讯云提供两套SDK接入环境。若您服务全球用户，推荐您使用以下接口配置全球接入环境。



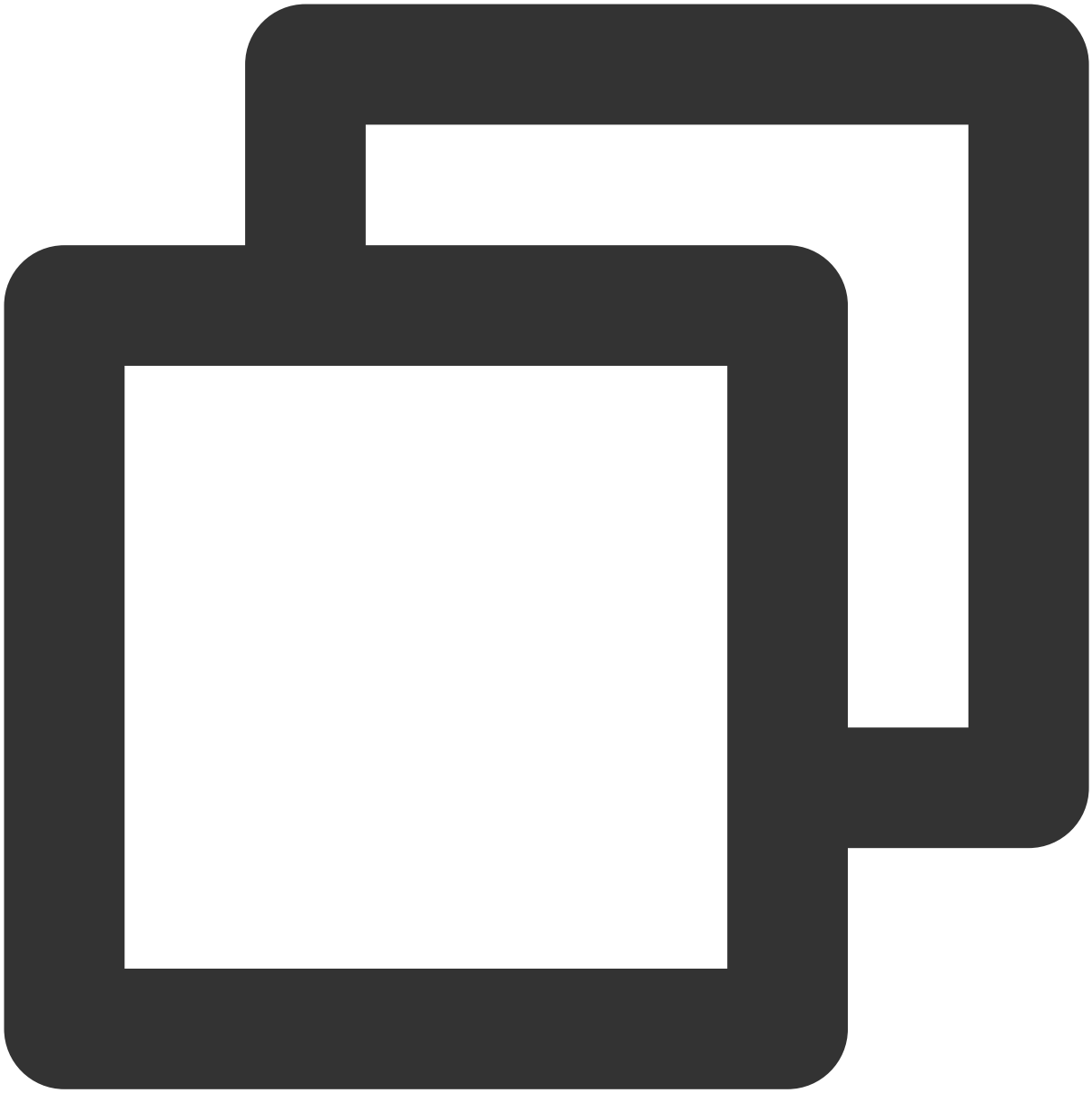
```
SuperPlayerPlugin.setGlobalEnv("GDPR");
```

步骤3：创建 controller



```
SuperPlayerController _controller = SuperPlayerController(context);
```

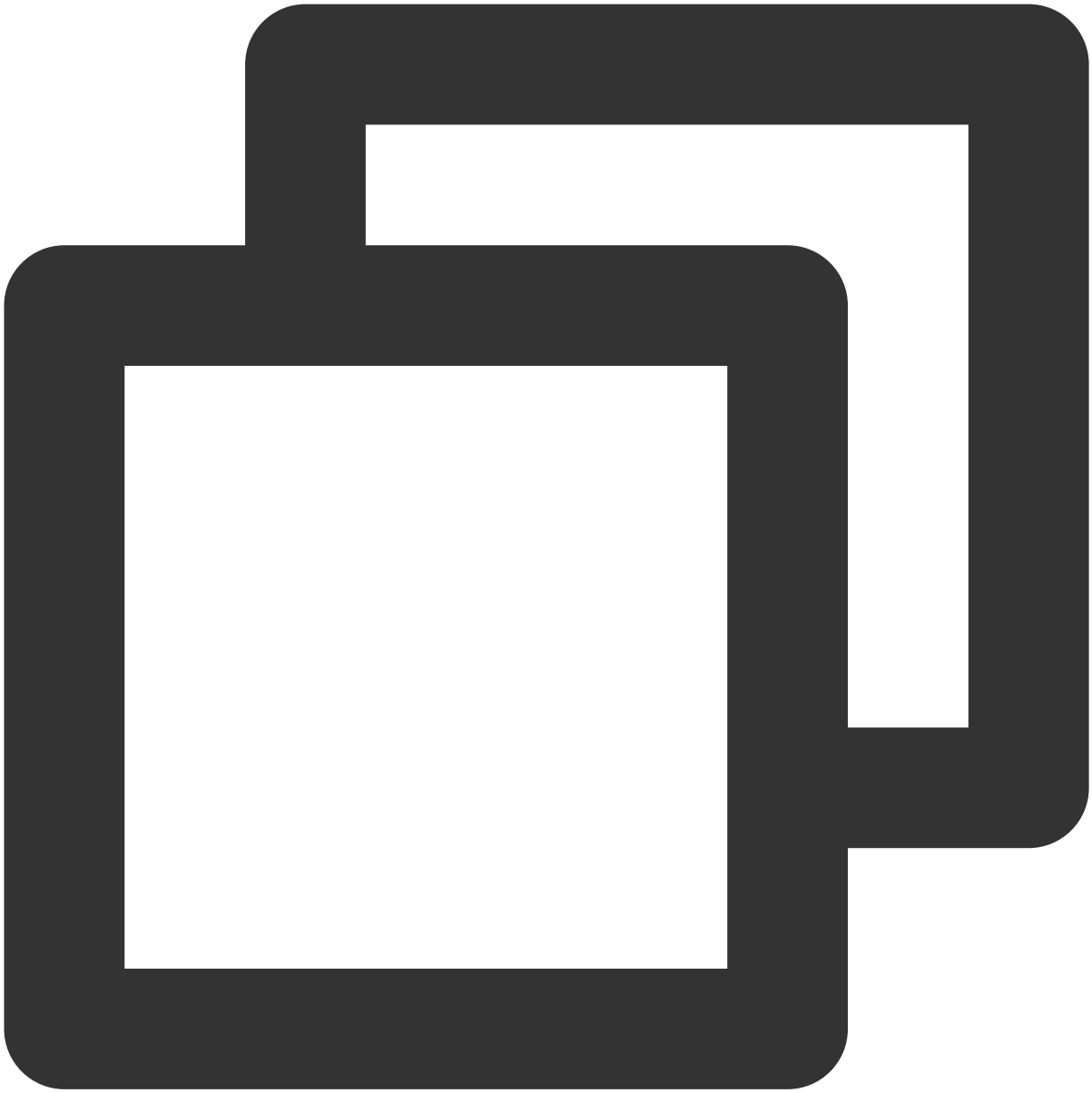
步骤4：配置播放器



```
FTXVodPlayConfig config = FTXVodPlayConfig();
// 如果不配置preferredResolution，则在播放多码率视频的时候优先播放720 * 1280分辨率的码率
config.preferredResolution = 720 * 1280;
_controller.setPlayConfig(config);
```

FTXVodPlayConfig 中的详细配置可参考 Flutter 点播播放器的配置播放器接口。

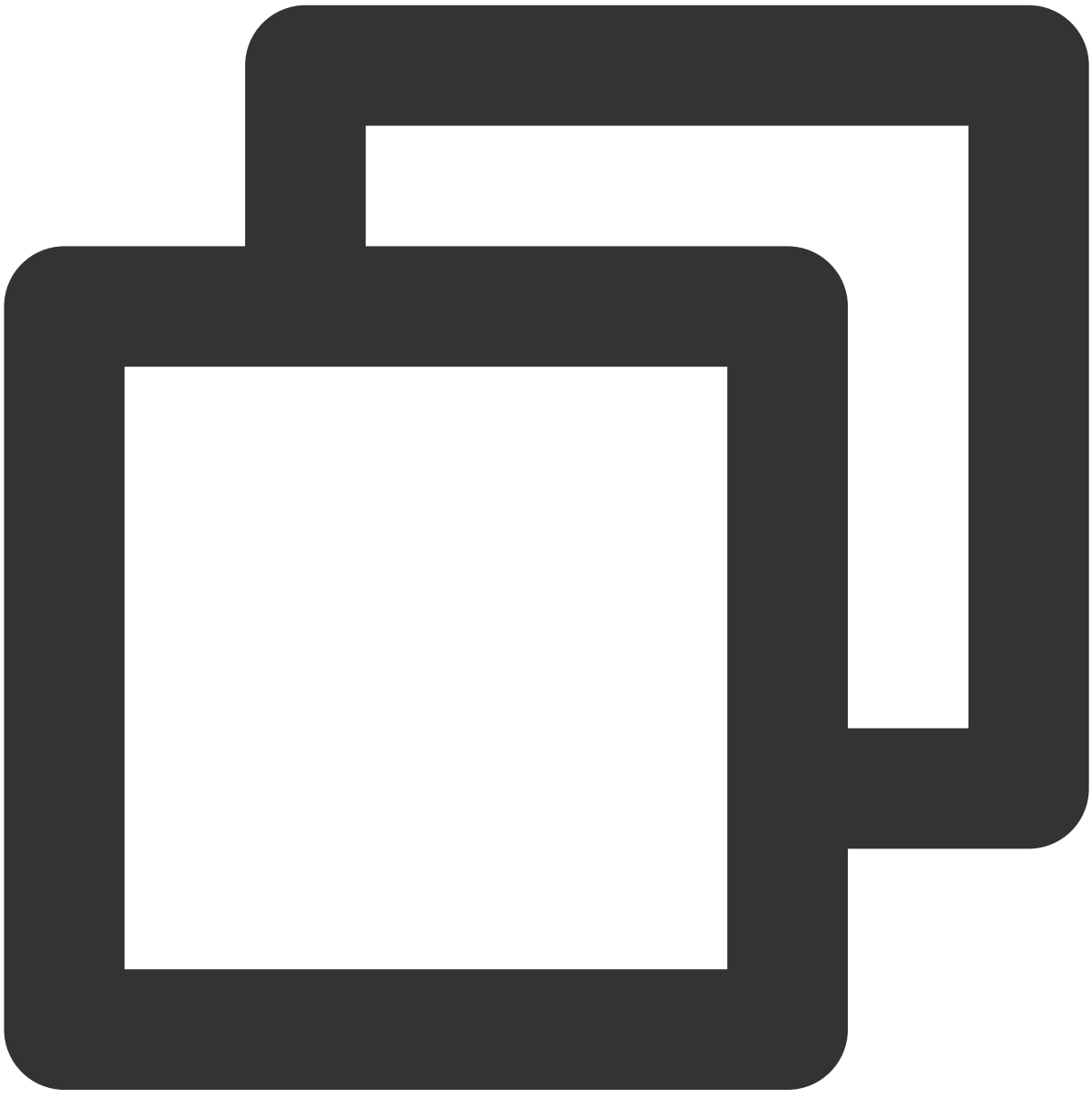
步骤5：设置监听事件



```
_controller.onSimplePlayerEventBroadcast.listen((event) {
  String evtName = event["event"];
  if (evtName == SuperPlayerViewEvent.onStartFullScreenPlay) {
    setState(() {
      _isFullScreen = true;
    });
  } else if (evtName == SuperPlayerViewEvent.onStopFullScreenPlay) {
    setState(() {
      _isFullScreen = false;
    });
  } else {
```

```
    print(evtName);
  }
});
```

步骤6：添加布局



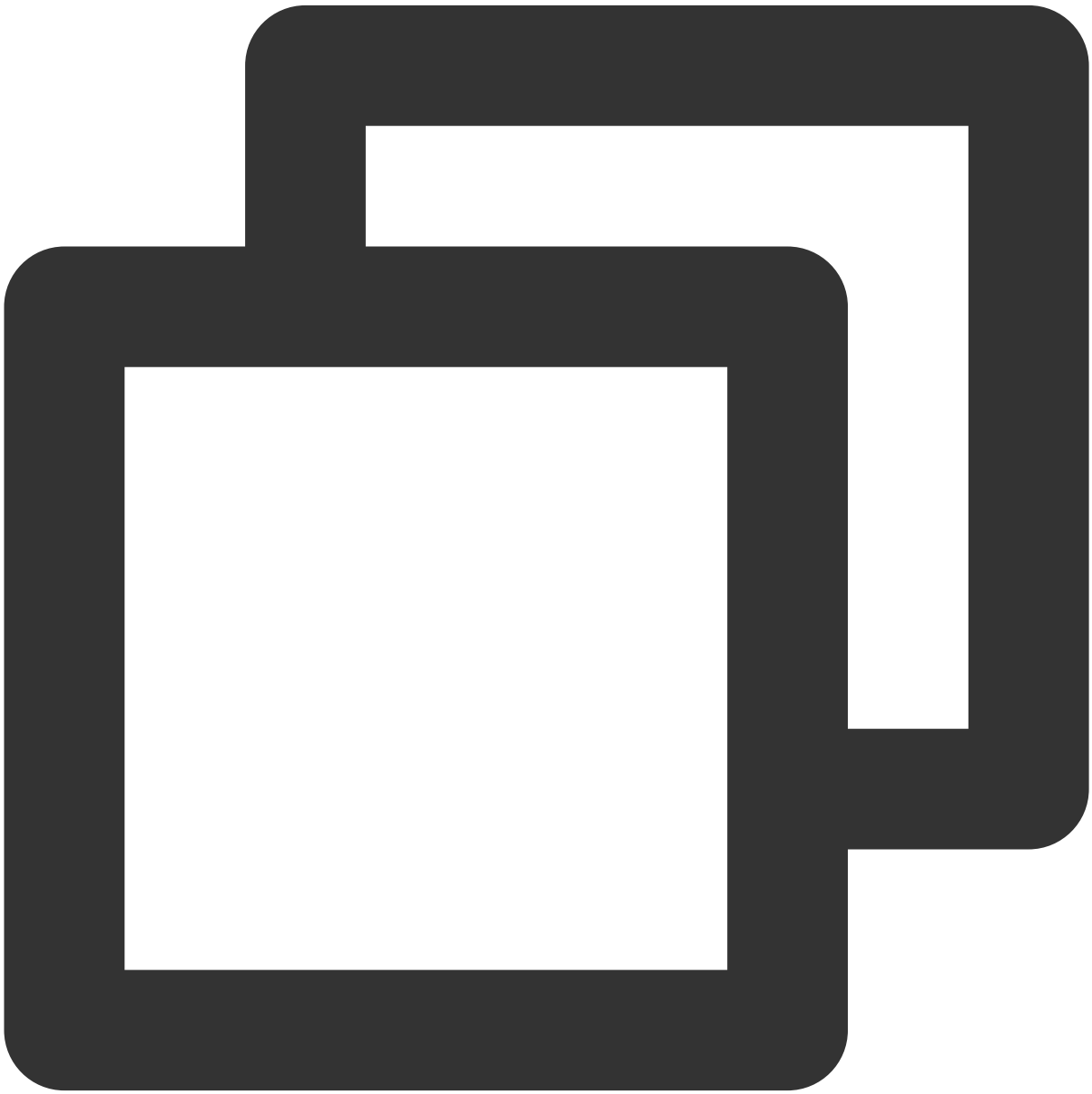
```
Widget _getPlayArea() {
    return Container(
    height: 220,
    child: SuperPlayerView(_controller),
```

```
    );
  }
```

步骤7：添加返回事件监听

添加返回事件监听，确保用户在触发返回事件的时候，如果播放器处于全屏等状态，可以优先退出全屏，再次触发才会退出页面。

如果全屏播放状态下需要直接退出页面，可以不实现该监听。



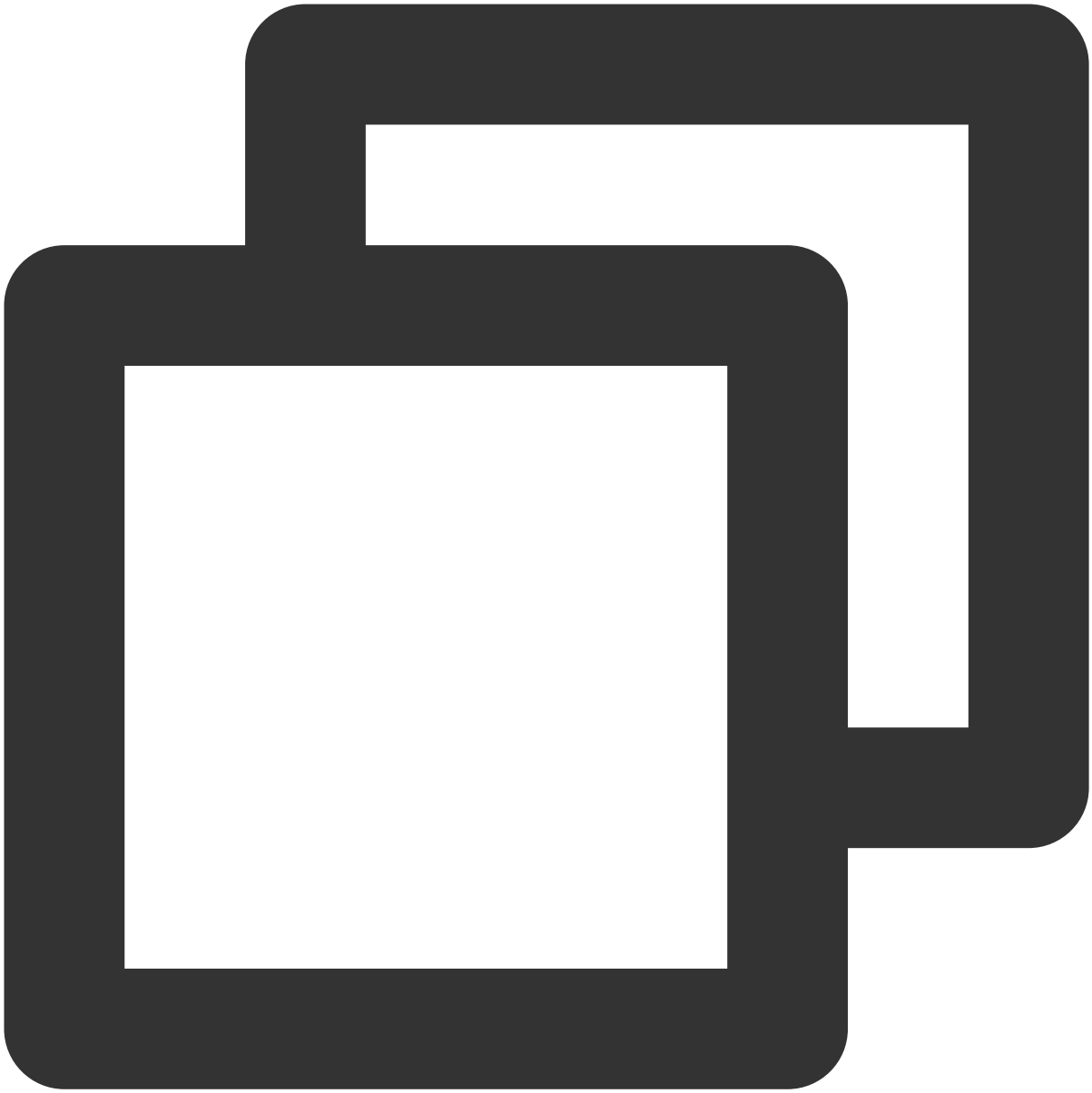
```
  @override
Widget build(BuildContext context) {
```

```
  return WillPopScope(
      child: Container(
        decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage("images/ic_new_vod_bg.png"),
              fit: BoxFit.cover,
            )),
        child: Scaffold(
          backgroundColor: Colors.transparent,
          appBar: _isFullScreen
              ? null
              : AppBar(
            backgroundColor: Colors.transparent,
            title: const Text('SuperPlayer'),
          ),
          body: SafeArea(
            child: Builder(
              builder: (context) => getBody(),
            ),
          ),
        ),
      ),
      onWillPop: onWillPop);
}

Future<bool> onWillPop() async {
  return !_controller.onBackPress();
}
```

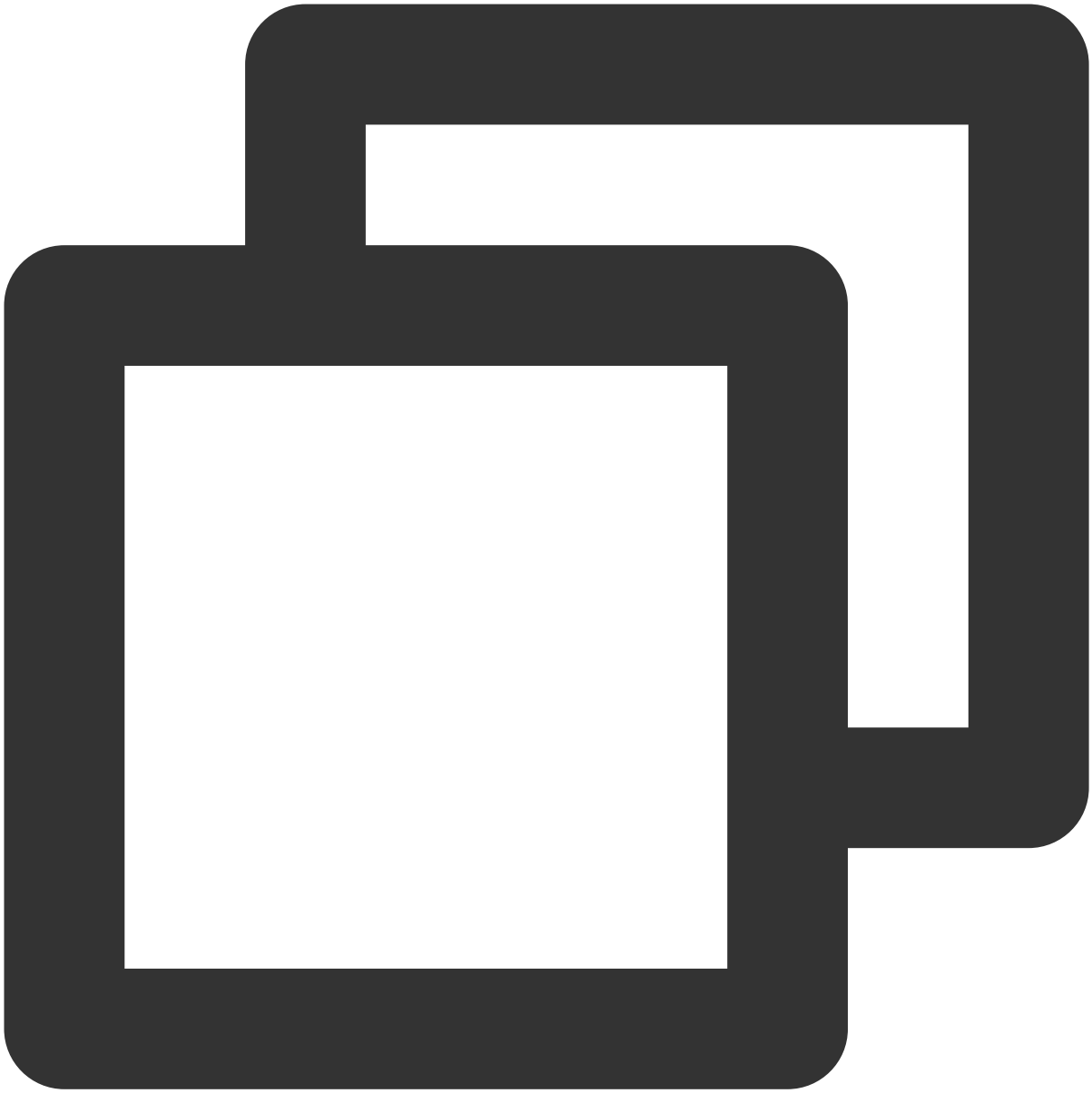
步骤8：启动播放

通过 url 方式

通过 fileId 方式



```
SuperPlayerModel model = SuperPlayerModel();
model.videoURL = "http://1400329073.vod2.myqcloud.com/d62d88a7vodtranscq1400329073/
_controller.playWithModelNeedLicence(model);
```



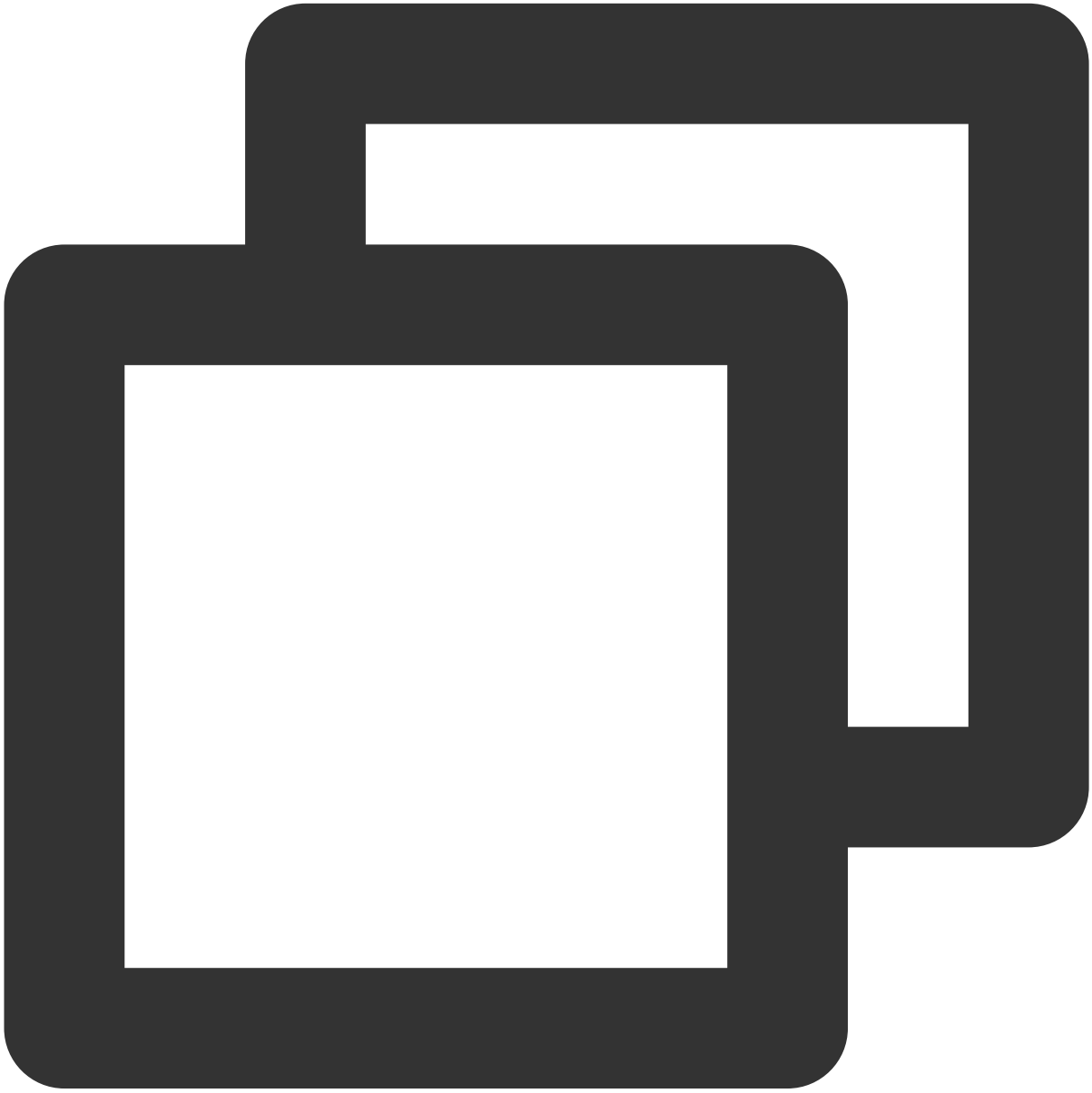
```
SuperPlayerModel model = SuperPlayerModel();
model.appId = 1500005830;
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "8602268011437356984";
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
model.videoId.pSign = "psignXXX"
_controller.playWithModelNeedLicence(model);
```

在 媒资管理 找到对应的视频文件。在文件名下方可以看到 FileId。

通过 FileId 方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或 FileId 不存在，则会收到 SuperPlayerViewEvent.onSuperPlayerError 事件。

步骤9：结束播放

结束播放时**记得调用 controller 的销毁方法**，尤其是在下次 startVodPlay 之前，否则可能会产生大量的内存泄露以及闪屏问题。也确保在退出页面的时候，能够结束视频播放。



```
  @override
void dispose() {
  // must invoke when page exit.
```

```
    _controller.releasePlayer();
    super.dispose();
  }
```

播放器组件接口列表

1、视频播放

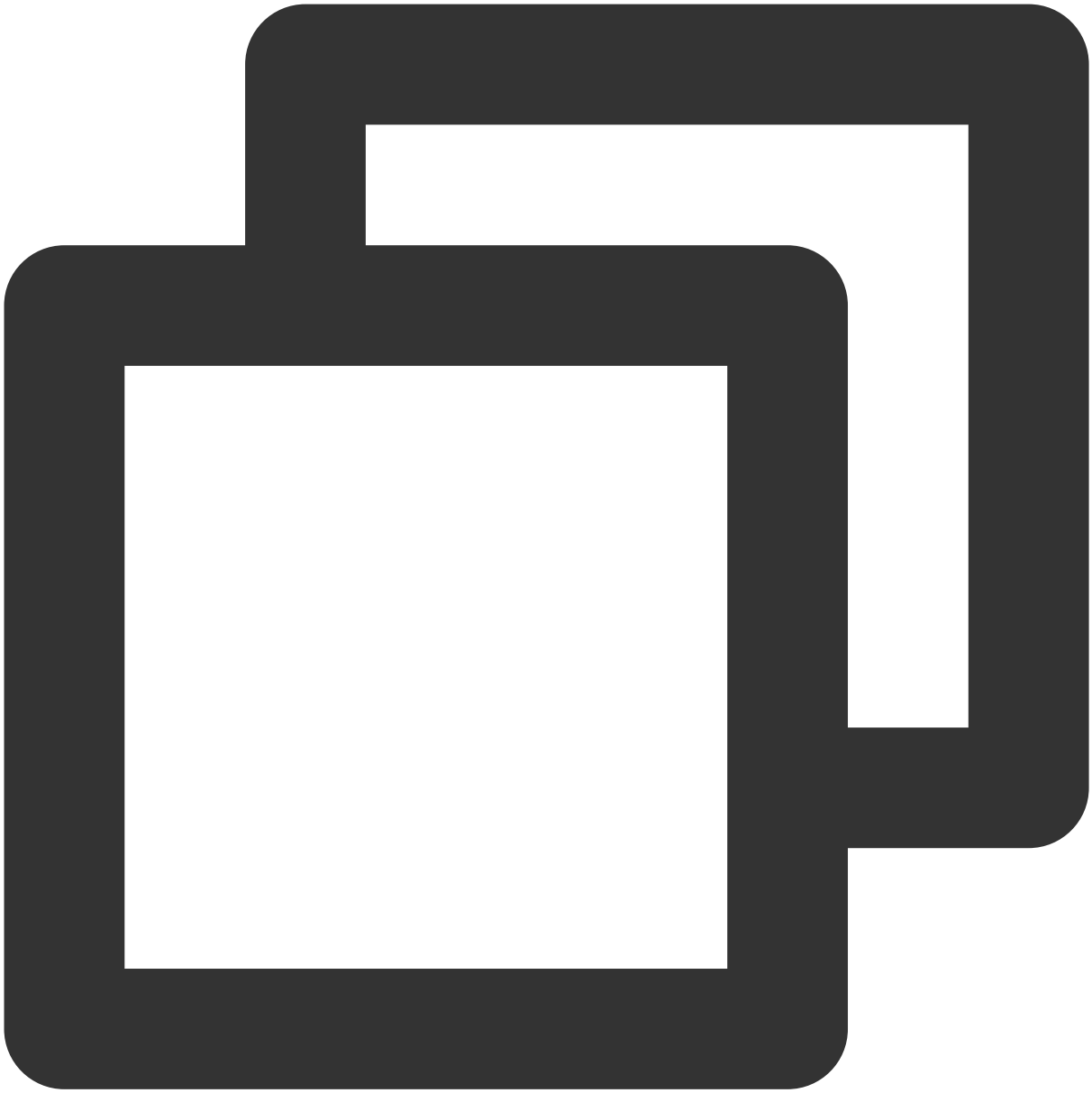
注意

10.7版本开始，startPlay变更为startVodPlay，需要通过 {@link SuperPlayerPlugin#setGlobalLicense} 设置 Licence 后方可成功播放， 否则将播放失败（黑屏），全局仅设置一次即可。直播 Licence、短视频 Licence 和视频播放 Licence 均可使用，若您暂未获取上述 Licence ，可快速免费申请 Licence 以正常播放。

说明

开始播放视频。

接口



```
_controller.playWithModelNeedLicence(model);
```

参数说明

1. SuperPlayerModel。

参数名	类型	描述
appId	int	应用 appId。fileId 播放必填。
videoURL	String	视频 url，url 播放必填。

multiVideoURLs	List<String>	多码率 url，多码率 url 播放必填。
defaultPlayIndex	int	默认播放码率序号，配合 multiVideoURLs 使用。
videoId	SuperPlayerVideoId	fileId 存储对象，以下会有详细介绍。
title	String	视频标题，用户可设置该字段来自定义标题，从而覆盖播放器内部从服务器请求的标题。
coverUrl	String	从腾讯服务器拉取的封面图片,该值会在 SuperVodDataLoader 中被自动赋值。
customeCoverUrl	String	自定义视频封面，该字段会被优先判断，可以通过定义该参数来实现自定义封面。
duration	int	视频时长，单位秒。
videoDescription	String	视频描述。
videoMoreDescription	String	视频详细描述。
playAction	int	action 包括 PLAY_ACTION_AUTO_PLAY、PLAY_ACTION_MANUAL_PLAY和PLAY_ACTION_PRELOAD，以下对参数含义会有详细介绍。

2. SuperPlayerVideoId。

参数名	类型	描述
fileId	String	文件 id。必填。
psign	String	播放器签名，签名介绍和生成方式。

3. playAction。

PLAY_ACTION_AUTO_PLAY：调用 playWithModel 之后，会自动开始播放视频。

PLAY_ACTION_MANUAL_PLAY：调用 playWithModel 之后，需要手动播放，并且播放器实质上并未加载视频，只会显示封面图，相对于 PLAY_ACTION_PRELOAD 没有任何视频播放资源消耗。

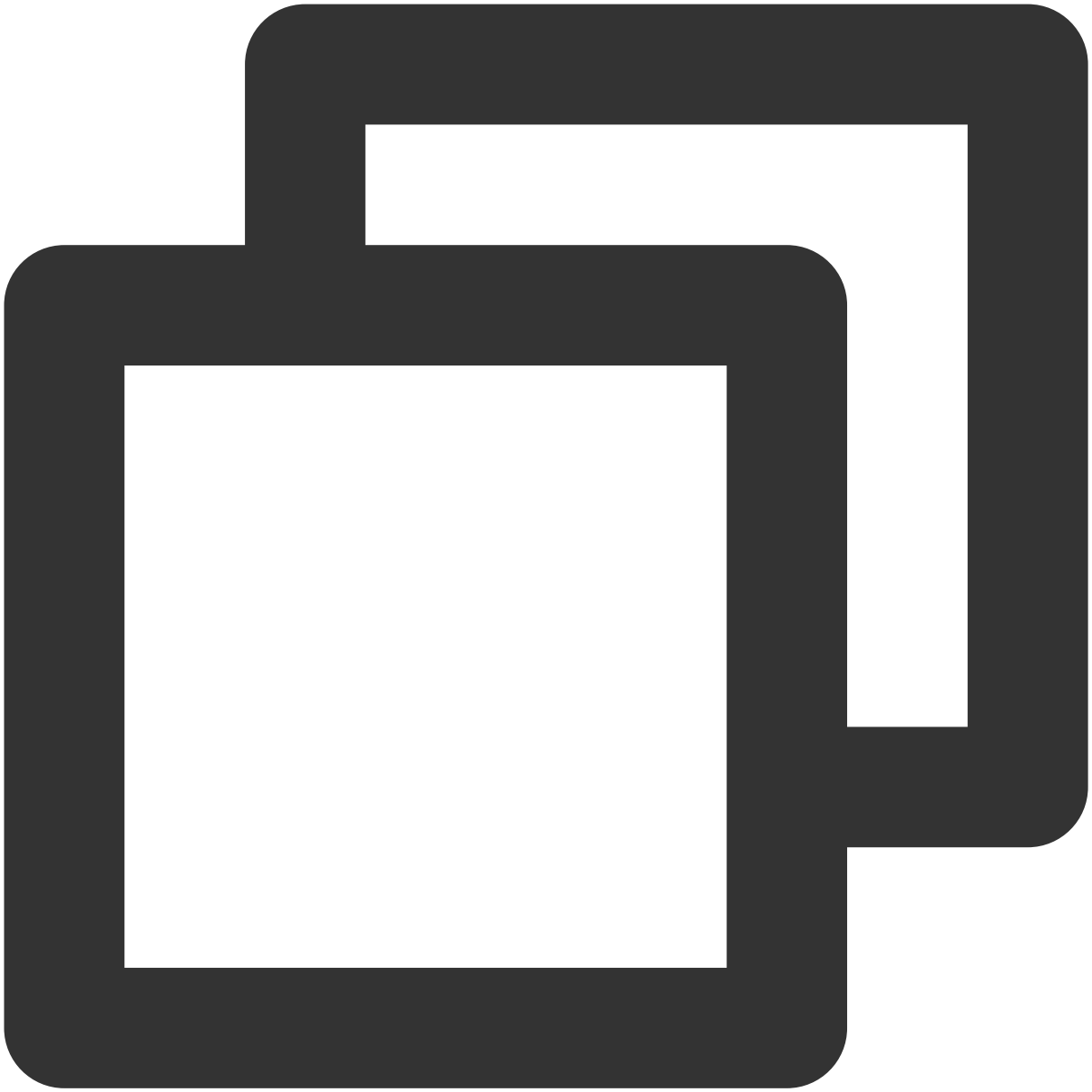
PLAY_ACTION_PRELOAD：调用 playWithModel 之后，会显示封面图，不会开始播放视频，不过播放器实质上已经加载了视频，相对于 PLAY_ACTION_MANUAL_PLAY，起播速度会更快。

2、暂停播放

说明

暂停播放视频。

接口



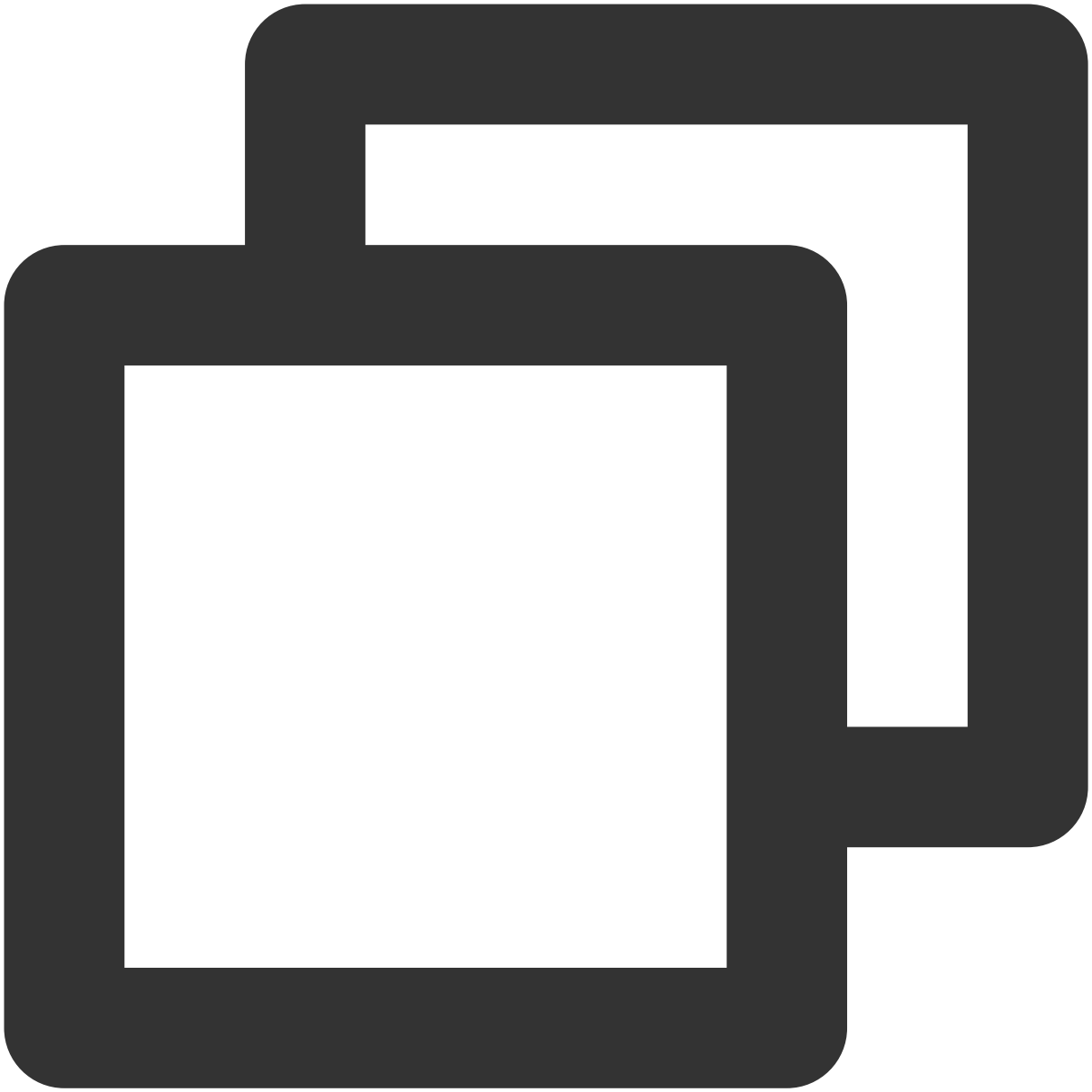
```
_controller.pause();
```

3、继续播放

说明

继续播放视频。

接口



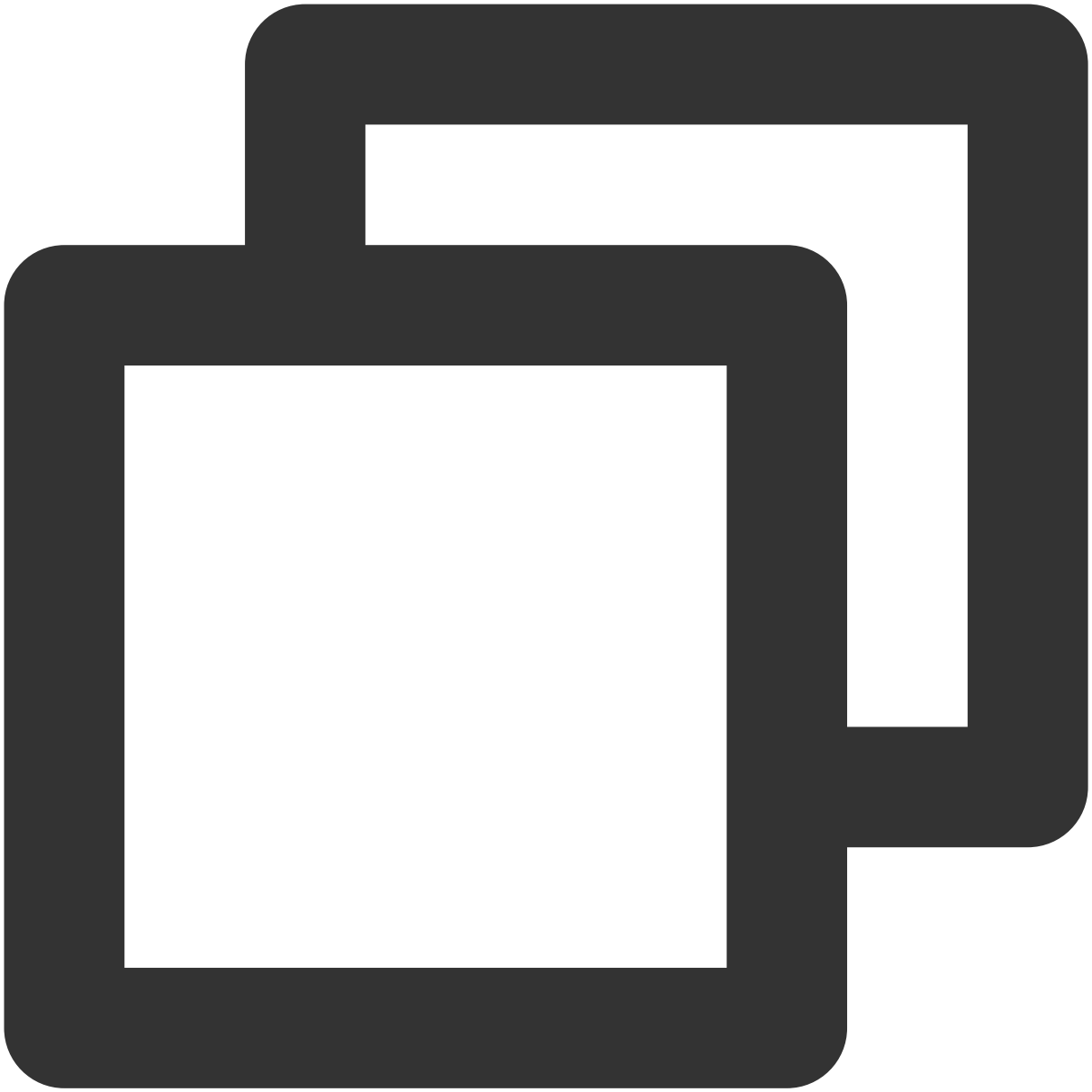
```
_controller.resume();
```

4、重新开始播放

说明

重新开始播放视频。

接口



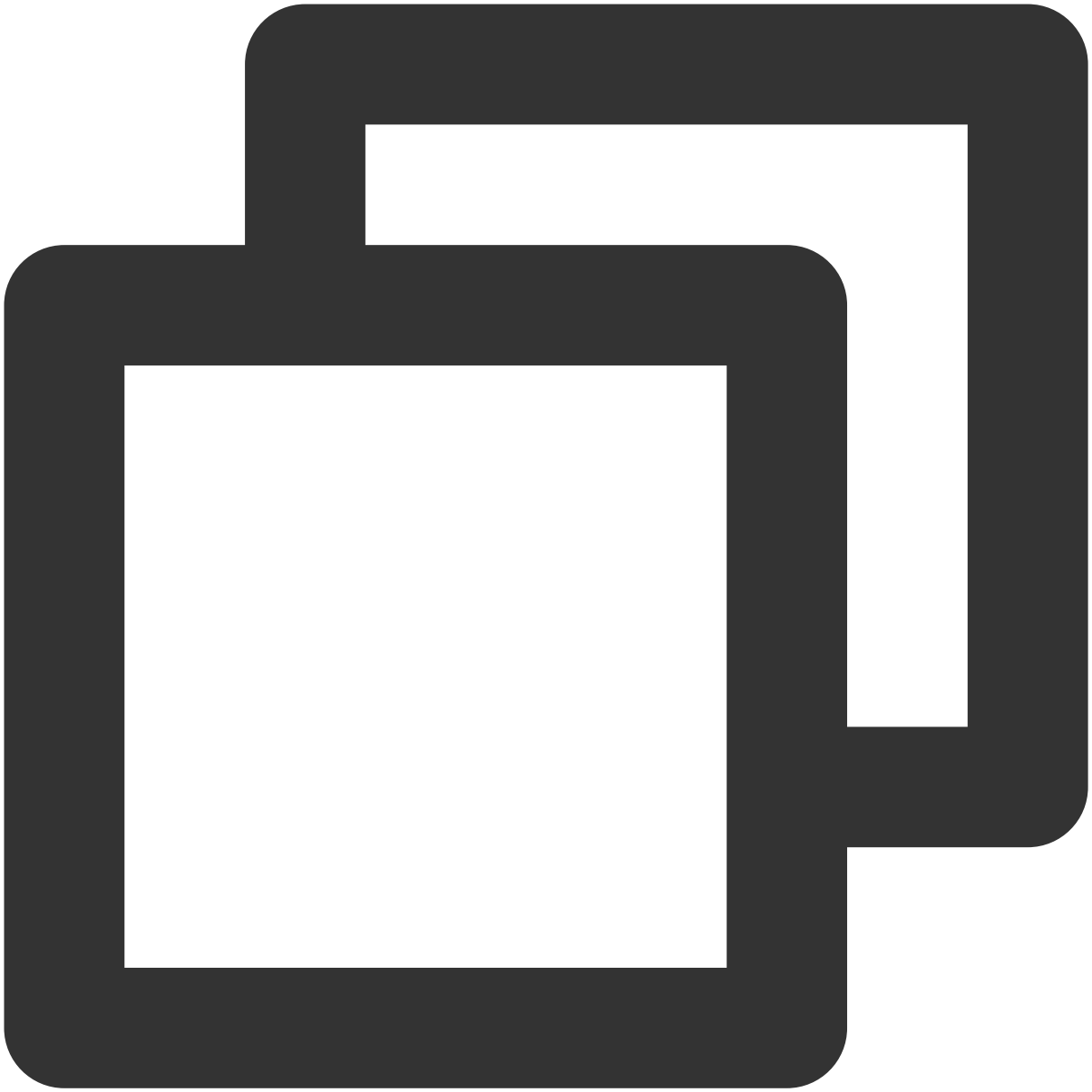
```
_controller.reStart();
```

5、重置播放器

说明

重置播放器状态，并停止播放视频。

接口



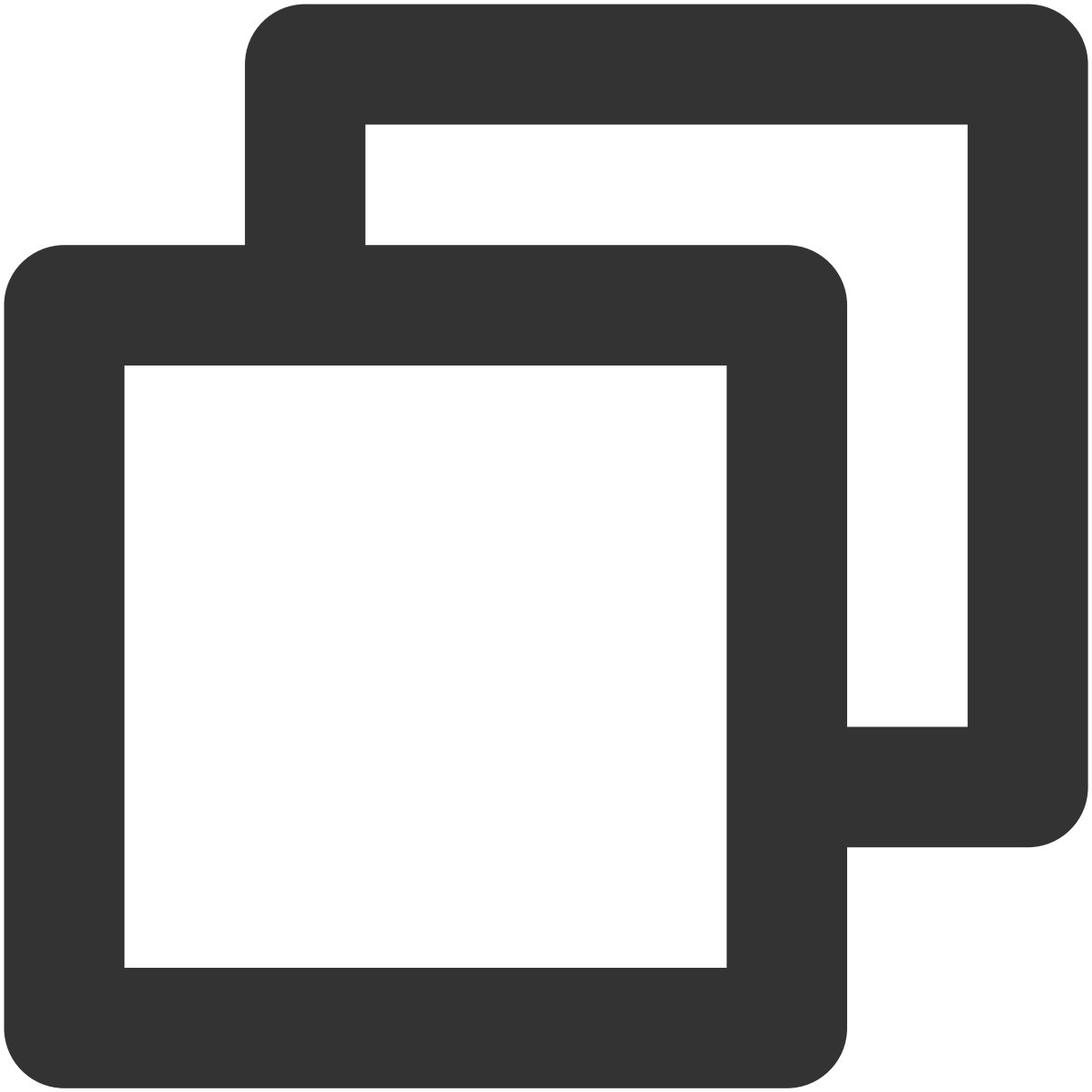
```
_controller.resetPlayer();
```

6、释放播放器

说明

释放播放器资源，并停止播放，调用该方法之后，controller 将不可再复用。

接口



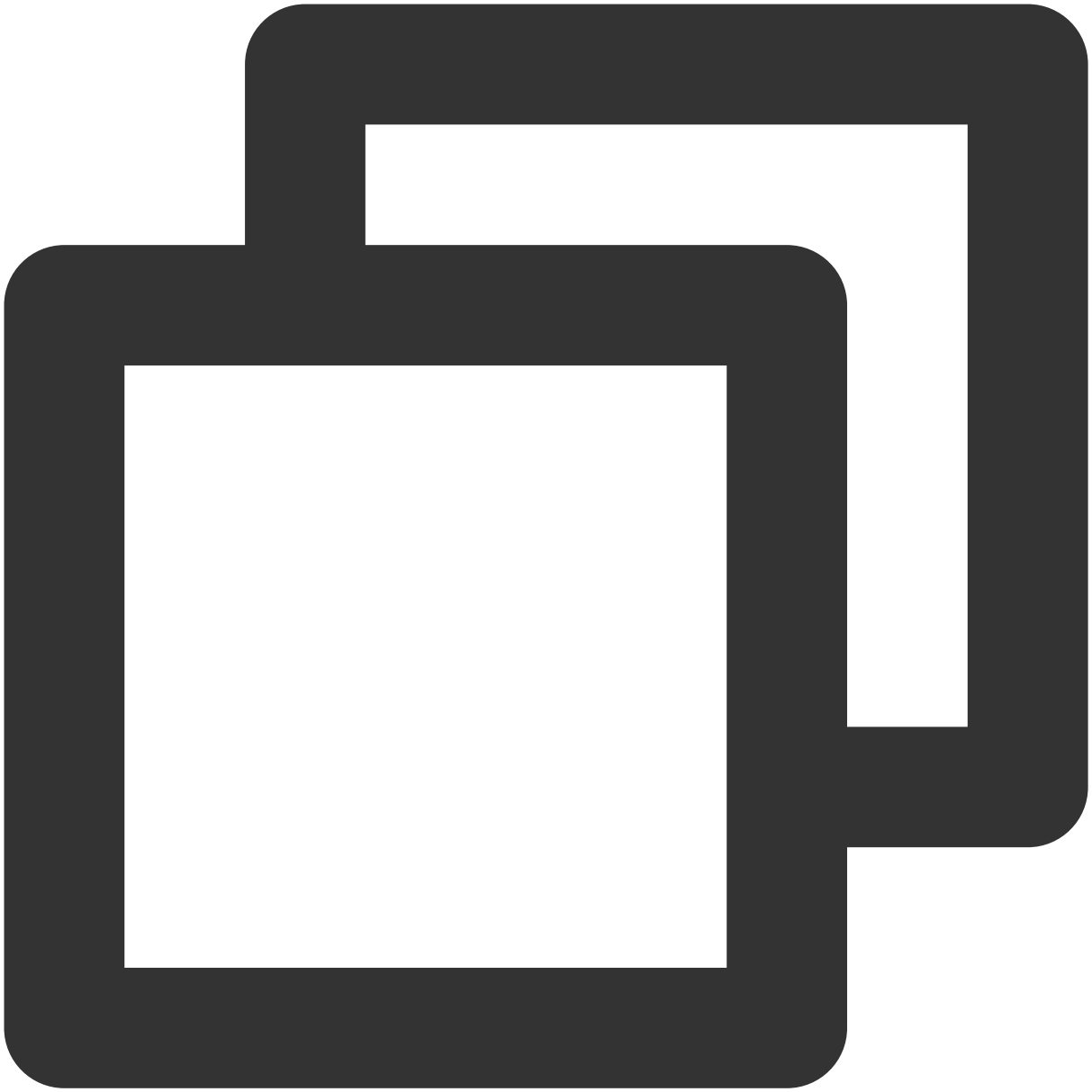
```
_controller.releasePlayer();
```

7、播放器返回事件

说明

触发播放器返回事件，该方法主要用于全屏播放模式下的返回判断和处理，返回 true：执行了退出全屏等操作，消耗了返回事件 false：未消耗事件。

接口



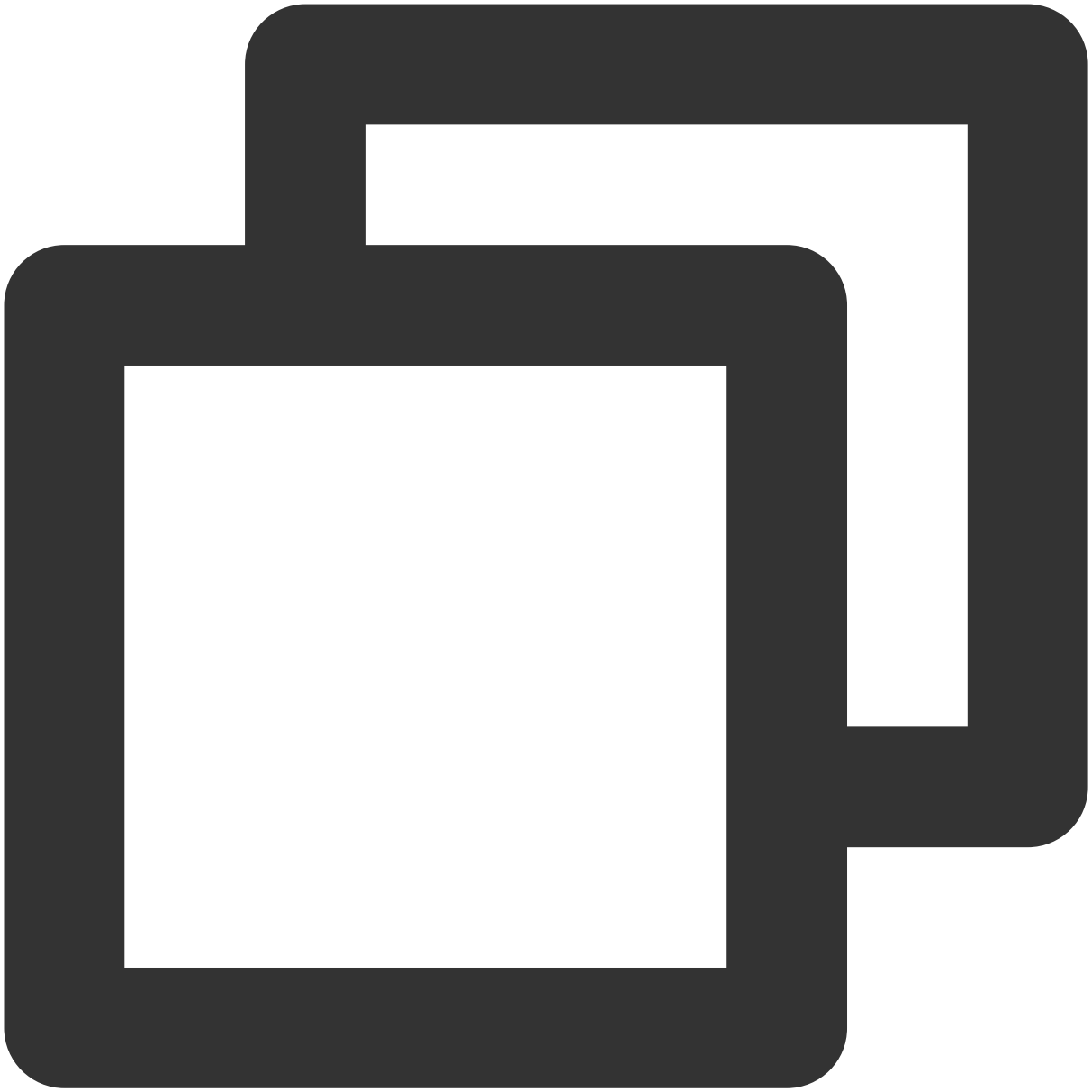
```
_controller.onBackPress();
```

8、切换清晰度

说明

实时切换当前正在播放的视频的清晰度。

接口



```
_controller.switchStream(videoQuality);
```

参数说明

videoQuality 在开始播放之后，一般可通过_controller.currentQualiyList和_controller.currentQuality来获取，前者为清晰度列表，后者为默认清晰度。**清晰度切换能力在超级播放器中已经集成，切换到全屏之后可点击右下角清晰度进行切换。**

参数名	类型	描述
index	int	清晰度序号

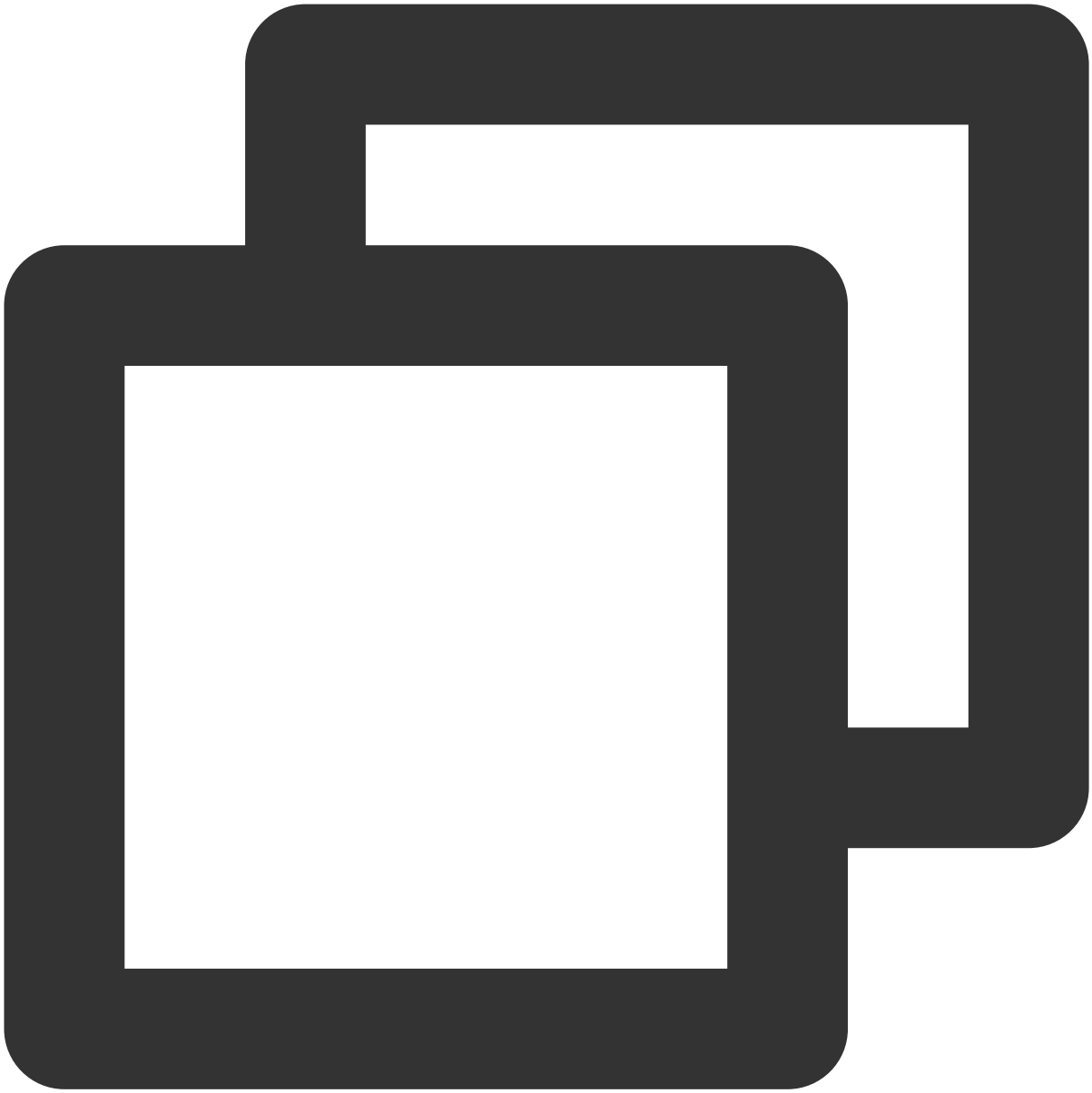
bitrate	int	清晰度码率
width	int	该清晰度下视频的宽度
height	int	该清晰度下视频的高度
name	String	清晰度简称
title	String	用于显示的清晰度名称
url	String	清晰度 url，用于多码率下的清晰度 url，非必填

9、调整进度(seek)

说明

调整当前视频的播放进度。

接口



```
_controller.seek(progress);
```

参数说明

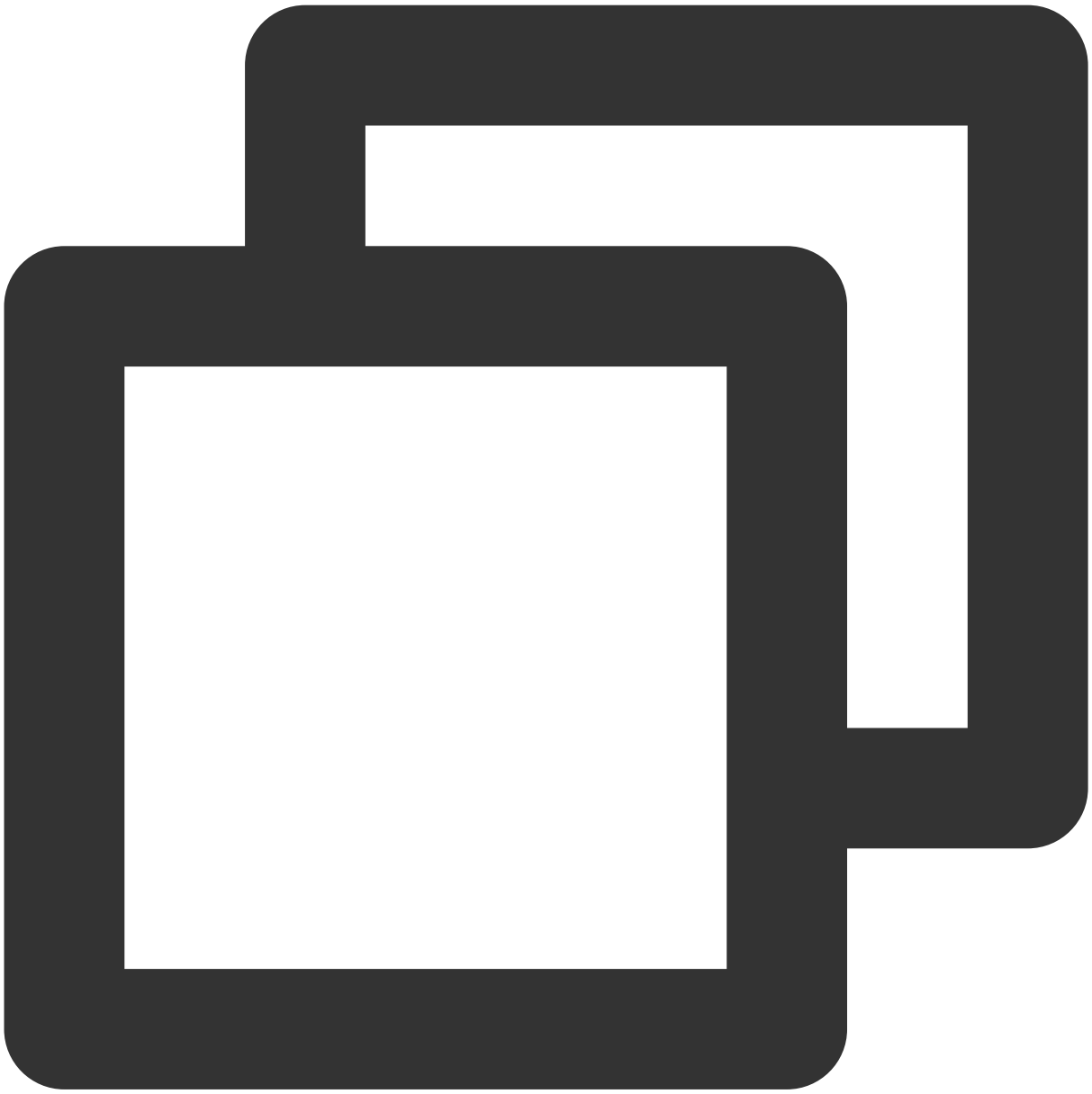
参数名	类型	描述
progress	double	需要调整到的时间，单位 秒

10、配置超级播放器

说明

配置超级播放器

接口



```
_controller.setPlayConfig(config);
```

参数说明

参数名	类型	描述
connectRetryCount	int	播放器重连次数，当 SDK 与服务器异常断开连接时,SDK 会尝试与服务器重连.通过该值设置SDK重连次数

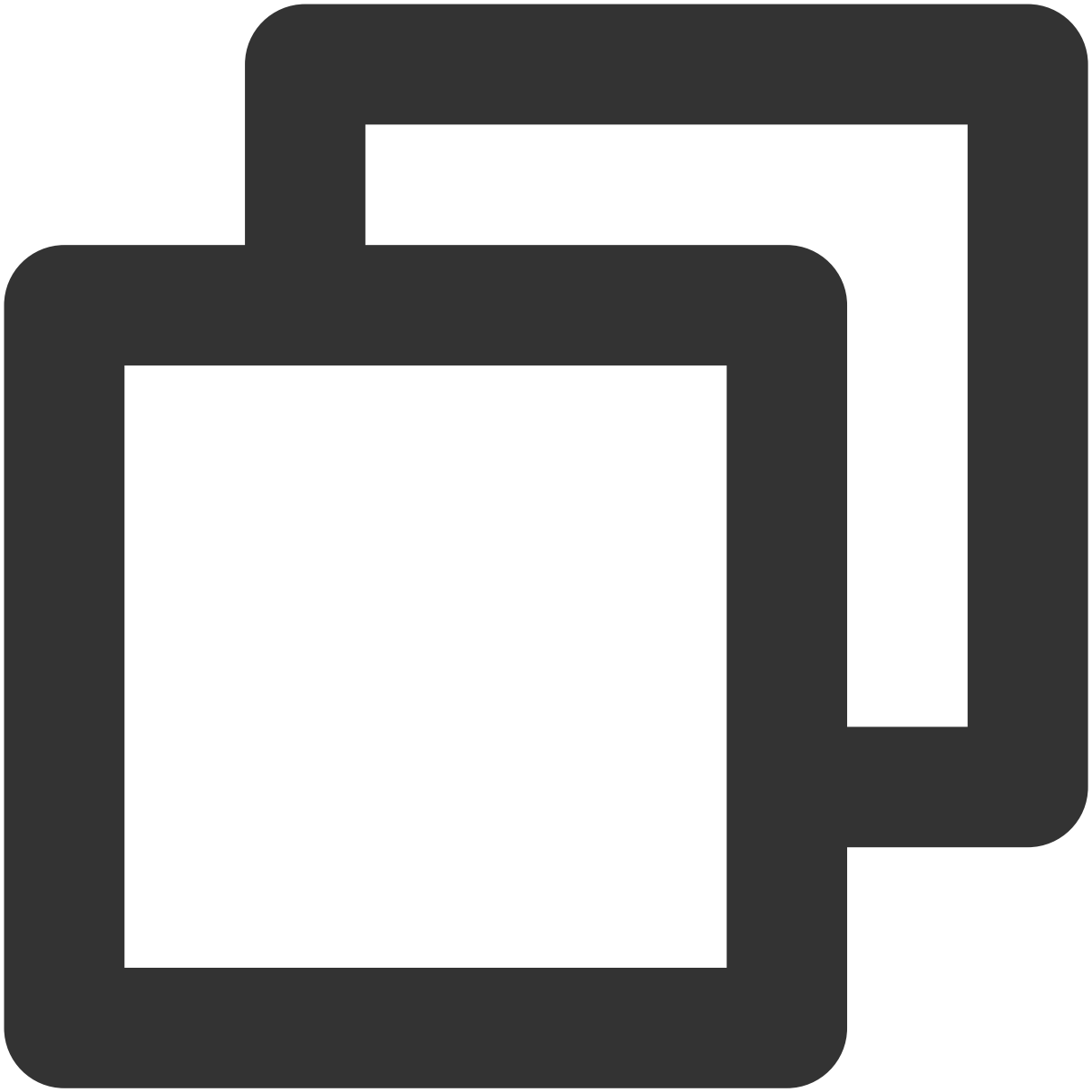
connectRetryInterval	int	播放器重连间隔，当 SDK 与服务器异常断开连接时,SDK 会尝试与服务器重连.通过该值设置两次重连间隔时间
timeout	int	播放器连接超时时间
playerType	int	播放器类型,0 点播，1 直播，2 直播回看
headers	Map	自定义http headers
enableAccurateSeek	bool	是否精确seek，默认true
autoRotate	bool	播放mp4文件时，若设为 true则根据文件中的旋转角度自动旋转。旋转角度可在PLAY_EVT_CHANGE_ROTATION事件中获得。默认 true
smoothSwitchBitrate	bool	平滑切换多码率 HLS，默认false。设为false时，可提高多码率地址打开速度; 设为true，在 IDR 对齐时可平滑切换码率
cacheMp4ExtName	String	缓存 mp4 文件扩展名,默认mp4
progressInterval	int	设置进度回调间隔,若不设置，SDK默认间隔0.5秒回调一次,单位毫秒
maxBufferSize	int	最大播放缓冲大小，单位 MB。此设置会影响 playableDuration，设置越大，提前缓存的越多
maxPreloadSize	int	预加载最大缓冲大小，单位：MB
firstStartPlayBufferTime	int	首缓需要加载的数据时长，单位ms，默认值为100ms
nextStartPlayBufferTime	int	缓冲时（缓冲数据不够引起的二次缓冲，或者 seek 引起的拖动缓冲）最少要缓存多长的数据才能结束缓冲，单位ms，默认值为250ms
overlayKey	String	HLS 安全加固加解密key
overlayIv	String	HLS 安全加固加解密Iv
extInfoMap	Map	一些不必周知的特殊配置
enableRenderProcess	bool	是否允许加载后渲染后处理服务,默认开启，开启后超分插件如果存在，默认加载
preferredResolution	int	优先播放的分辨率，preferredResolution = width * height

11、开关硬解

说明

开启或关闭硬解播放能力。

接口



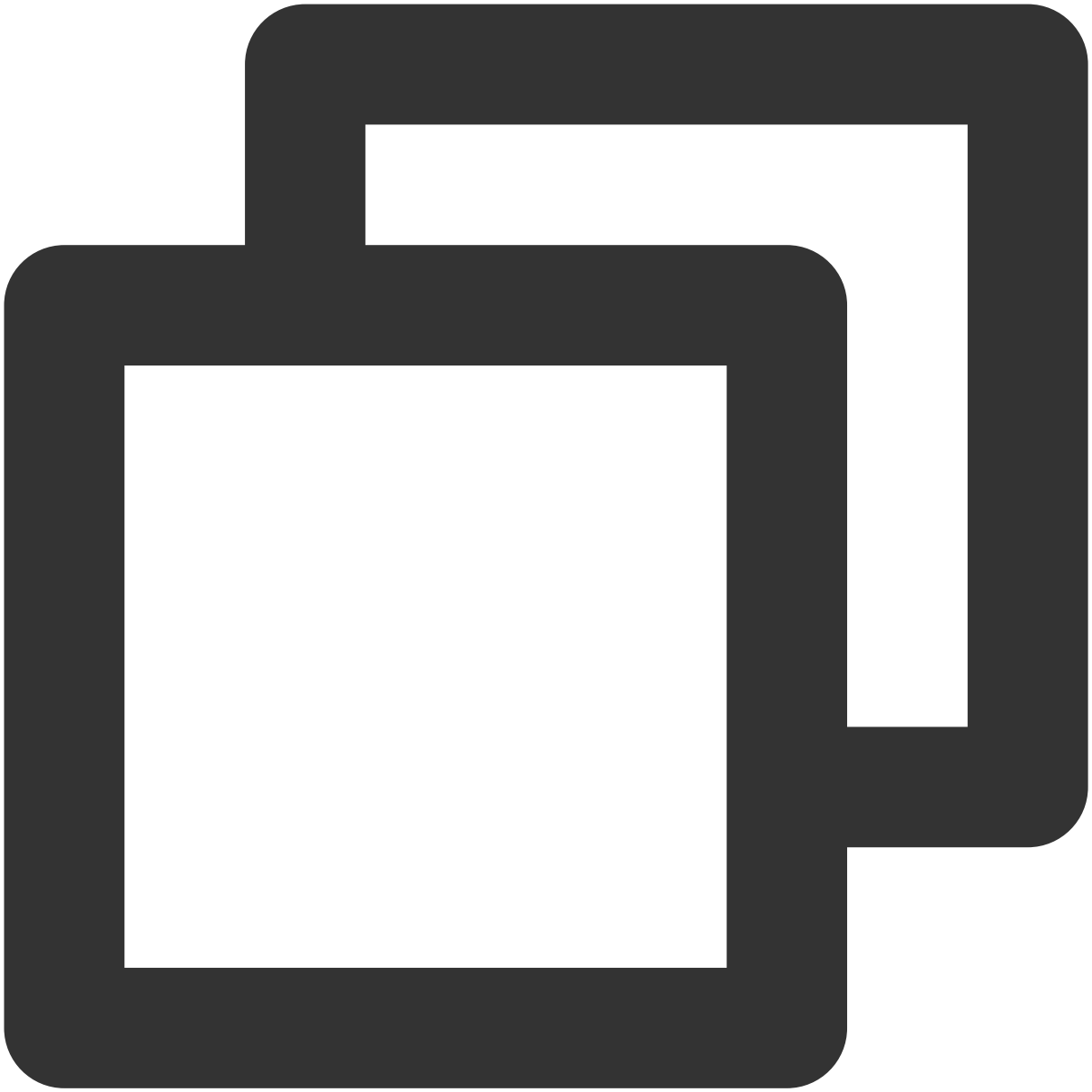
```
_controller.enableHardwareDecode(enable);
```

12、获得播放状态

说明

获得当前播放器的播放状态。

接口



```
SuperPlayerState superPlayerState = _controller.getPlayerState();
```

参数说明

参数名	类型	描述
INIT	SuperPlayerState	初始状态
PLAYING	SuperPlayerState	播放中
PAUSE	SuperPlayerState	暂停中

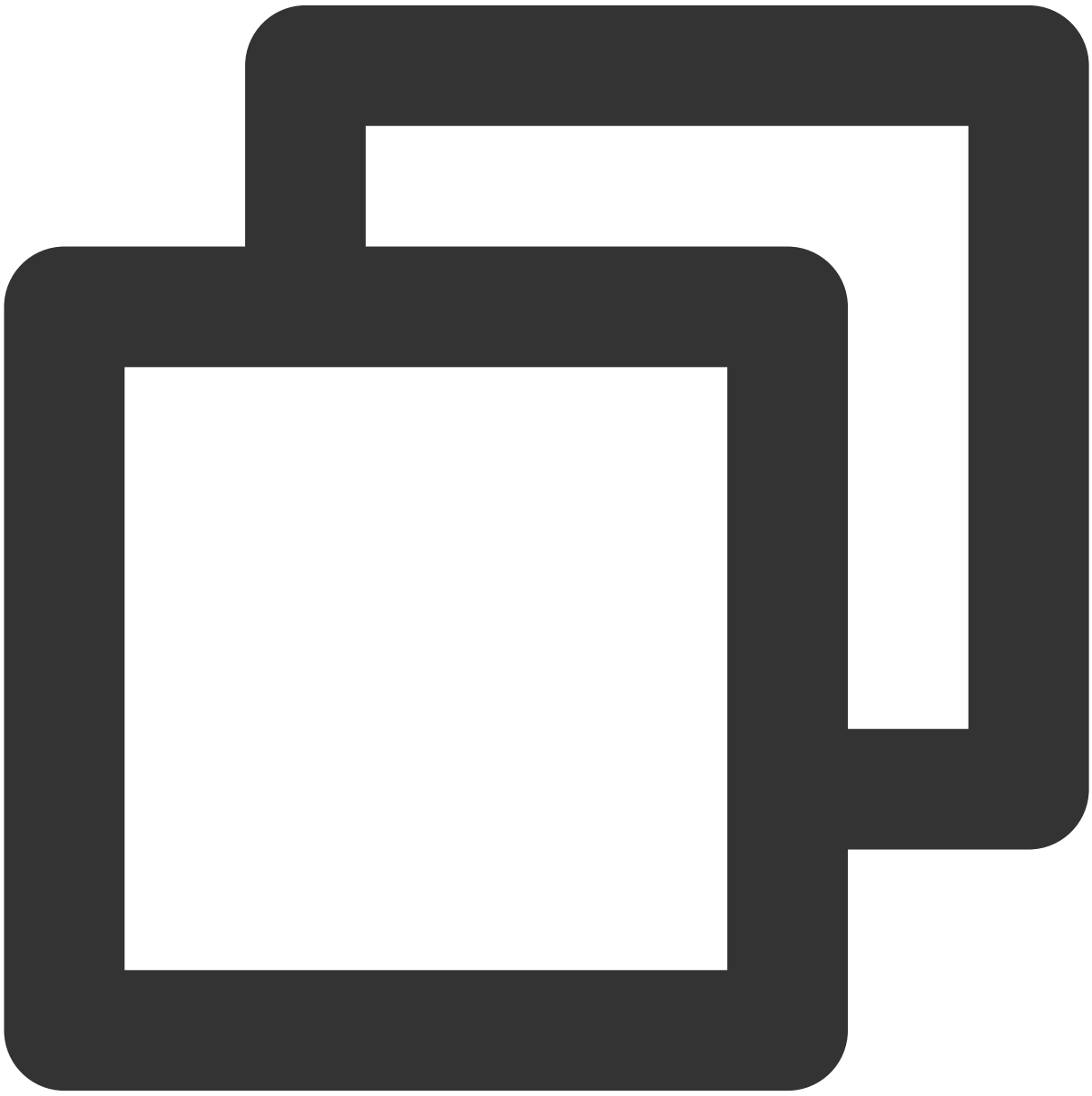
LOADING	SuperPlayerState	缓冲中
END	SuperPlayerState	播放结束

13、进入画中画模式

说明

调动该方法之后，视频将会进入画中画模式，该模式只支持 android 7.0以上，并且支持画中画模式的机型。

接口



```
_controller.enterPictureInPictureMode(
```

```
backIcon: "images/ic_pip_play_replay.png",
playIcon: "images/ic_pip_play_normal.png",
pauseIcon: "images/ic_pip_play_pause.png",
forwardIcon: "images/ic_pip_play_forward.png");
```

参数说明

该参数只适用于android平台。

参数名	类型	描述
backIcon	String	回退按钮图标，由于android平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
playIcon	String	播放按钮图标，由于android平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
pauseIcon	String	暂停按钮图标，由于android平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
forwardIcon	String	快进按钮图标，由于android平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标

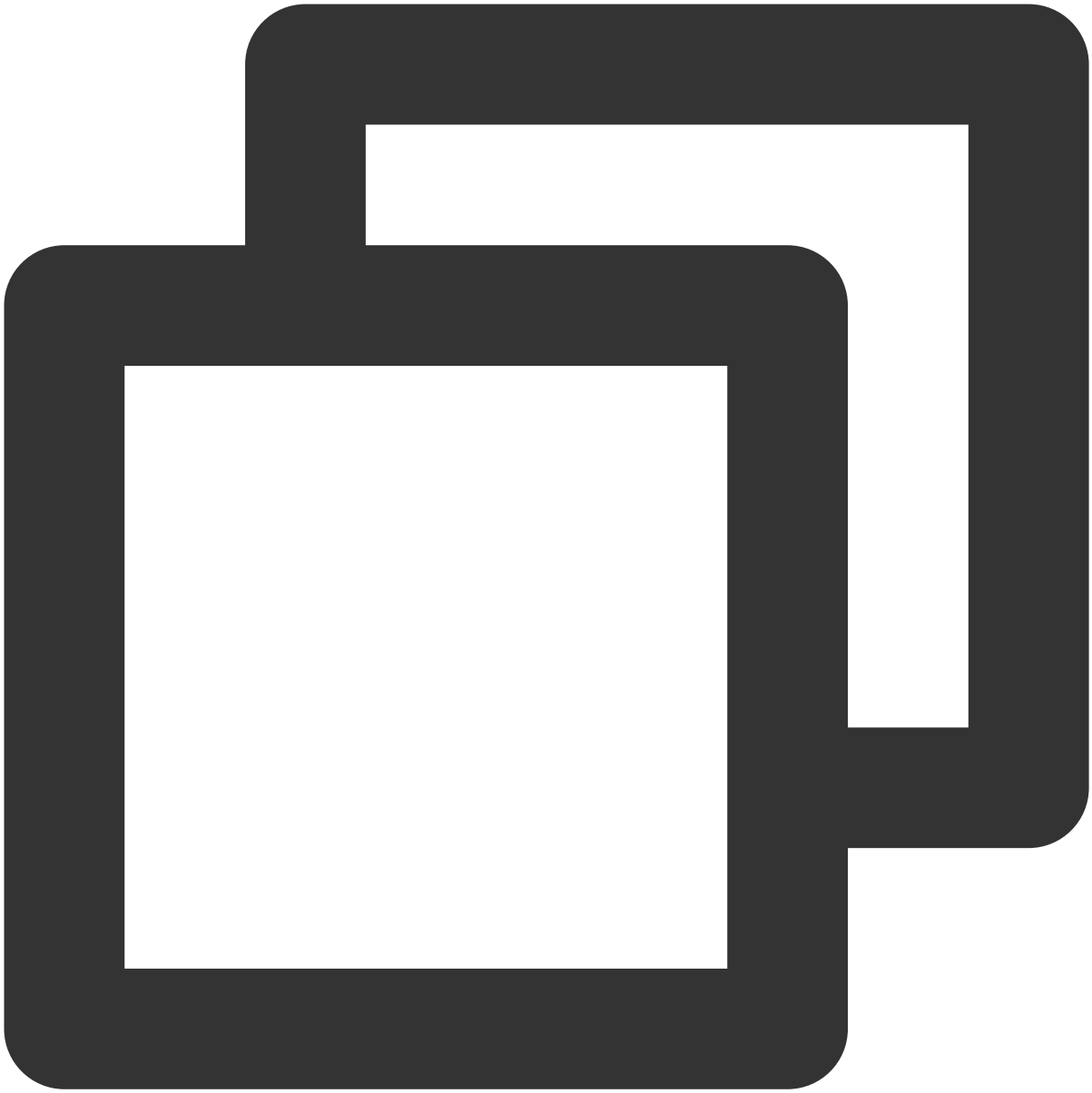
事件通知

1、播放事件监听

说明

监听播放器的操作事件。

代码



```
_controller.onSimplePlayerEventBroadcast.listen((event) {
    String evtName = event["event"];
    if (evtName == SuperPlayerViewEvent.onStartFullScreenPlay) {
        setState(() {
        _ isFullScreen = true;
        });
    } else if (evtName == SuperPlayerViewEvent.onStopFullScreenPlay) {
        setState(() {
          _isFullScreen = false;
        });
    } else {
```

```
            print(evtName);
        }
    });
```

事件说明

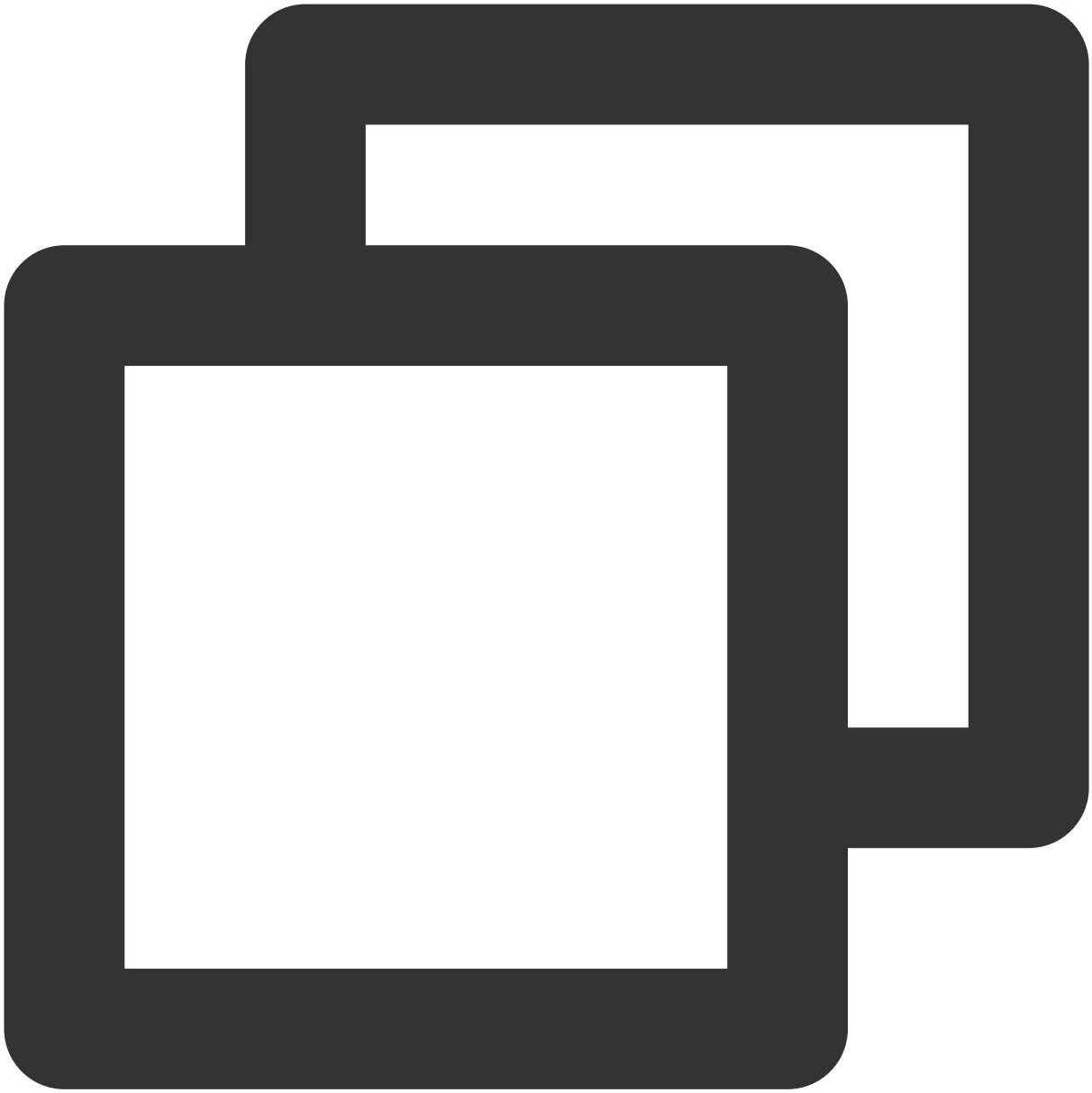
状态	含义
onStartFullScreenPlay	进入全屏播放。
onStopFullScreenPlay	退出全屏播放。
onSuperPlayerDidStart	播放开始通知。
onSuperPlayerDidEnd	播放结束通知。
onSuperPlayerError	播放错误通知。
onSuperPlayerBackAction	返回事件。

高级功能

1、通过 fileId 提前请求视频数据

可通过 SuperVodDataLoader 提前将视频数据请求下来，提高起播速度。

代码示例



```
SuperPlayerModel model = SuperPlayerModel();
model.appId = 1500005830;
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "8602268011437356984";
model.title = "云点播";
SuperVodDataLoader loader = SuperVodDataLoader();
// model中的必要参数会在SuperVodDataLoader中直接赋值
loader.getVideoData(model, (resultModel) {
  _controller.playWithModelNeedLicence(resultModel);
})
```

2、画中画模式的使用

1. **平台配置**

Android

iOS

在自己项目 android 包下，找到 build.gradle，确保 compileSdkVersion 和 targetSdkVersion 的版本为31或以上。

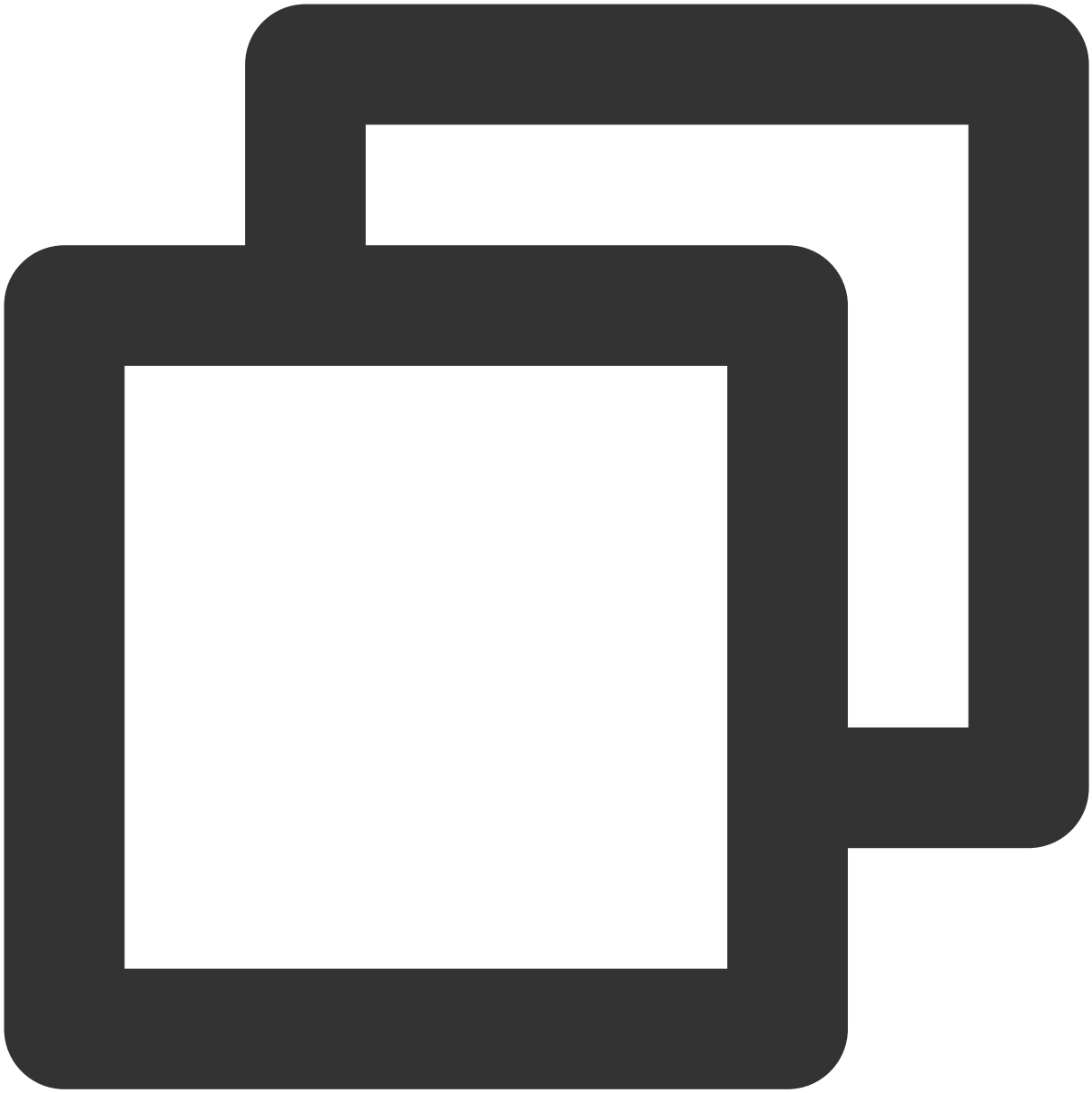
在自己项目的 target 下选择 **Signing & Capabilities** 添加 **Background Modes**，勾选 “**Audio,AirPlay,and Picture in Picture**”。

2. **复制 superPlayer 示例代码**

将 github 项目中 superplayer_widget 导入到自己的 lib 目录下，仿照示例代码中的 demo_superplayer.dart 集成播放器组件。然后就可以在播放器组件的播放界面右边中间看到画中画模式按钮，点击即可进入画中画模式。

3. **监听画中画模式生命周期**

使用 SuperPlayerPlugin 中的 onExtraEventBroadcast 可监听到画中画模式的生命周期，示例代码如下：



```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
  int eventCode = event["event"];
  if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_ALREADY_EXIT) {
    // exit pip mode
  } else if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_REQUEST_START) {
    // enter pip mode
  } else if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_ALREADY_ENTER) {
    // already enter pip mode
  } else if (eventCode == TXVodPlayEvent.EVENT_IOS_PIP_MODE_WILL_EXIT) {
    // will exit pip mode
  } else if (eventCode == TXVodPlayEvent.EVENT_IOS_PIP_MODE_RESTORE_UI) {
```

```
      // restore UI only support iOS
    }
  });
```

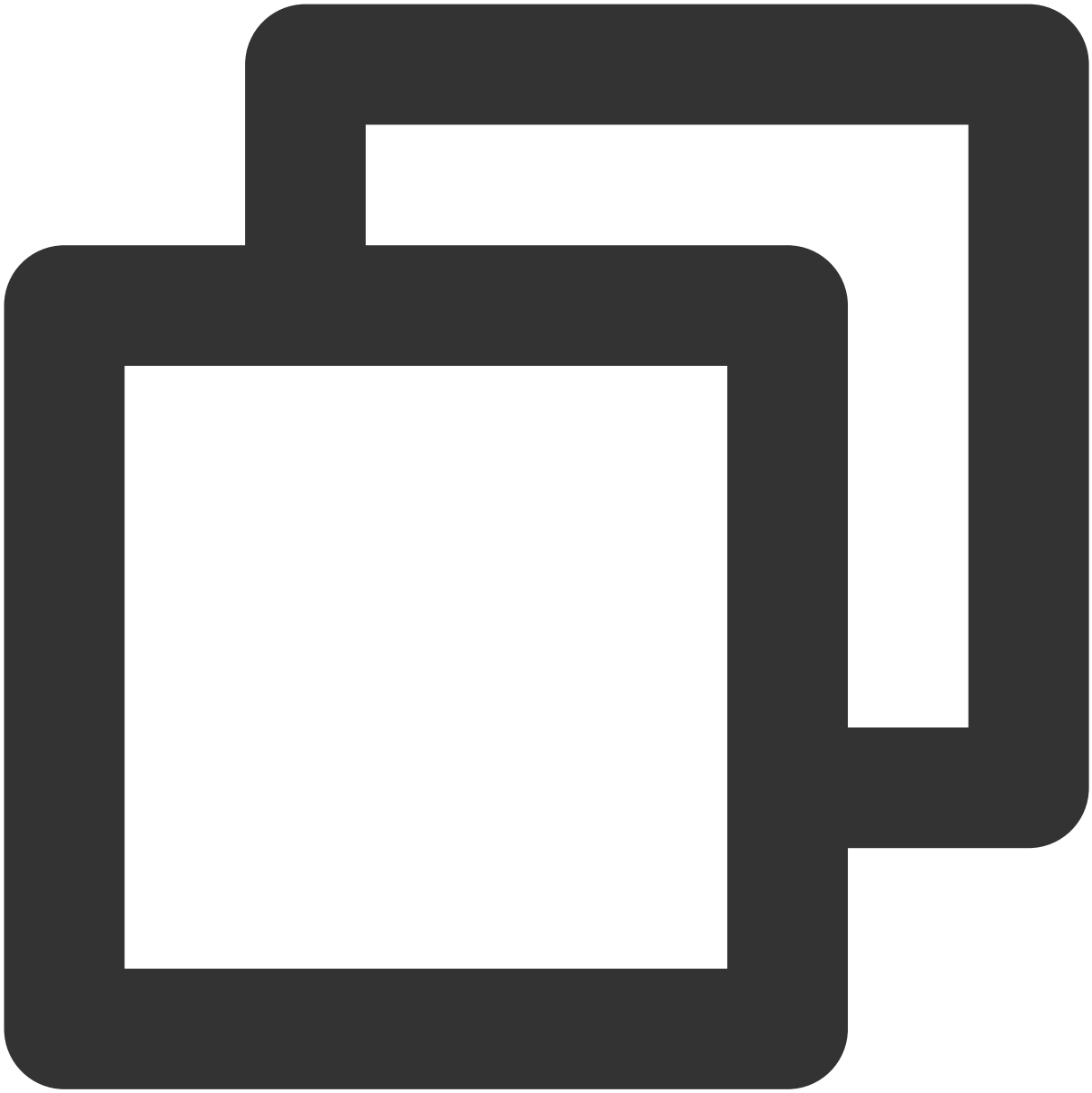
4. 画中画模式进入错误码

进入画中画模式失败的时候，除了有 log 提示以外，还会有 toast 提示，可在 superplayer_widget.dart 的 _onEnterPipMode 方法内修改错误情况处理。错误码含义如下：

参数名	值	描述
NO_ERROR	0	启动成功，没有错误。
ERROR_PIP_LOWER_VERSION	-101	Android 版本过低，不支持画中画模式。
ERROR_PIP_DENIED_PERMISSION	-102	画中画模式权限未打开，或者当前设备不支持画中画。
ERROR_PIP_ACTIVITY_DESTROYED	-103	当前界面已经销毁。
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	设备或系统版本不支持（iPad iOS9+ 才支持 PIP）。
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	播放器不支持。
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	视频不支持。
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	PIP 控制器不可用。
ERROR_IOS_PIP_FROM_SYSTEM	-108	PIP 控制器报错。
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	播放器对象不存在。
ERROR_IOS_PIP_IS_RUNNING	-110	PIP 功能已经运行。
ERROR_IOS_PIP_NOT_RUNNING	-111	PIP 功能没有启动。

5. 判断当前设备是否支持画中画

使用 SuperPlayerPlugin 中的 isDeviceSupportPip 可判断当前是否能够开启画中画，代码示例如下：



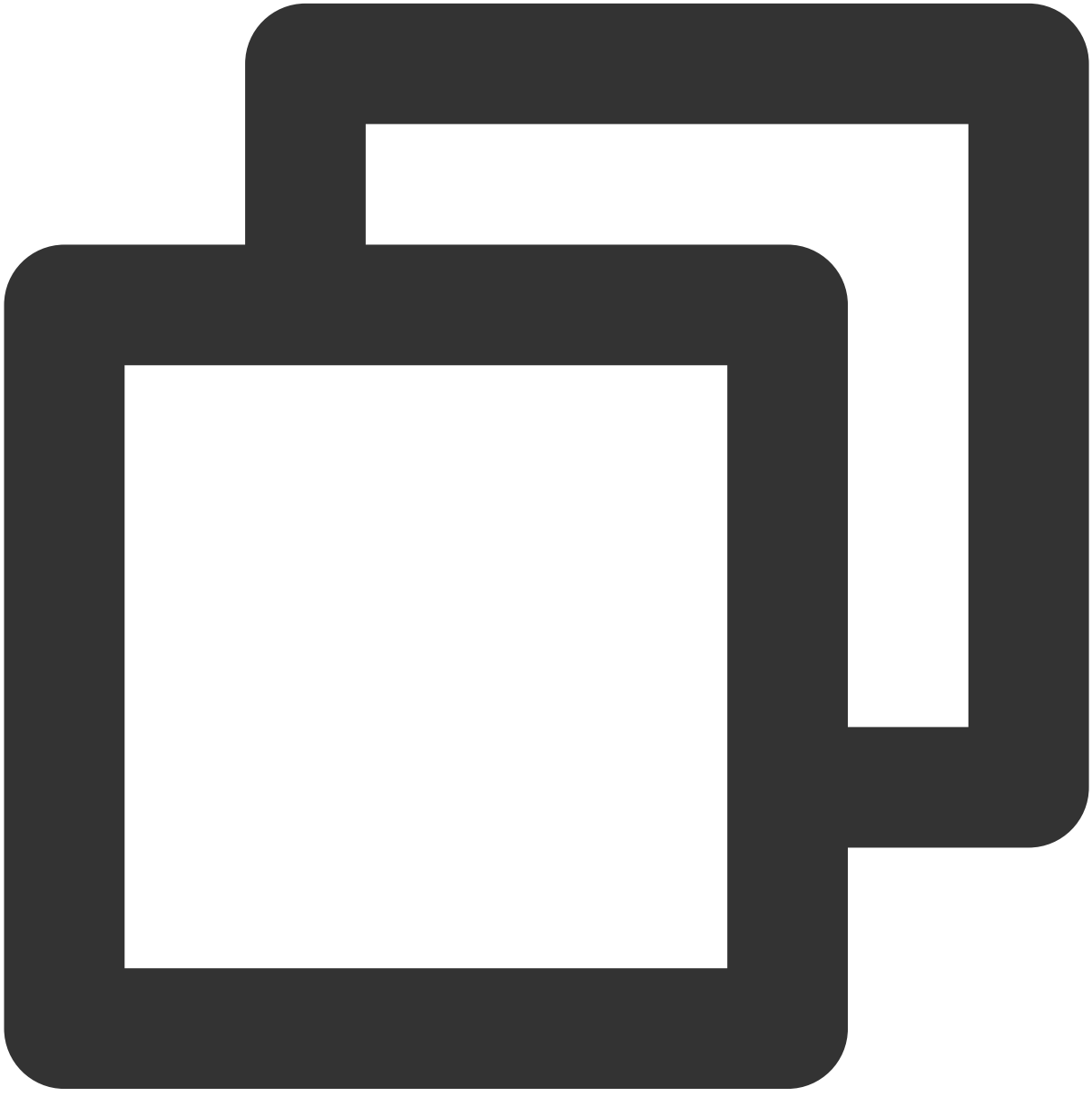
```
int result = await SuperPlayerPlugin.isDeviceSupportPip();
if(result == TXVodPlayEvent.NO_ERROR) {
  // pip support
}
```

result 的返回结果的含义和画中画模式错误码一致。

6. **使用画中画控制器管理画中画**

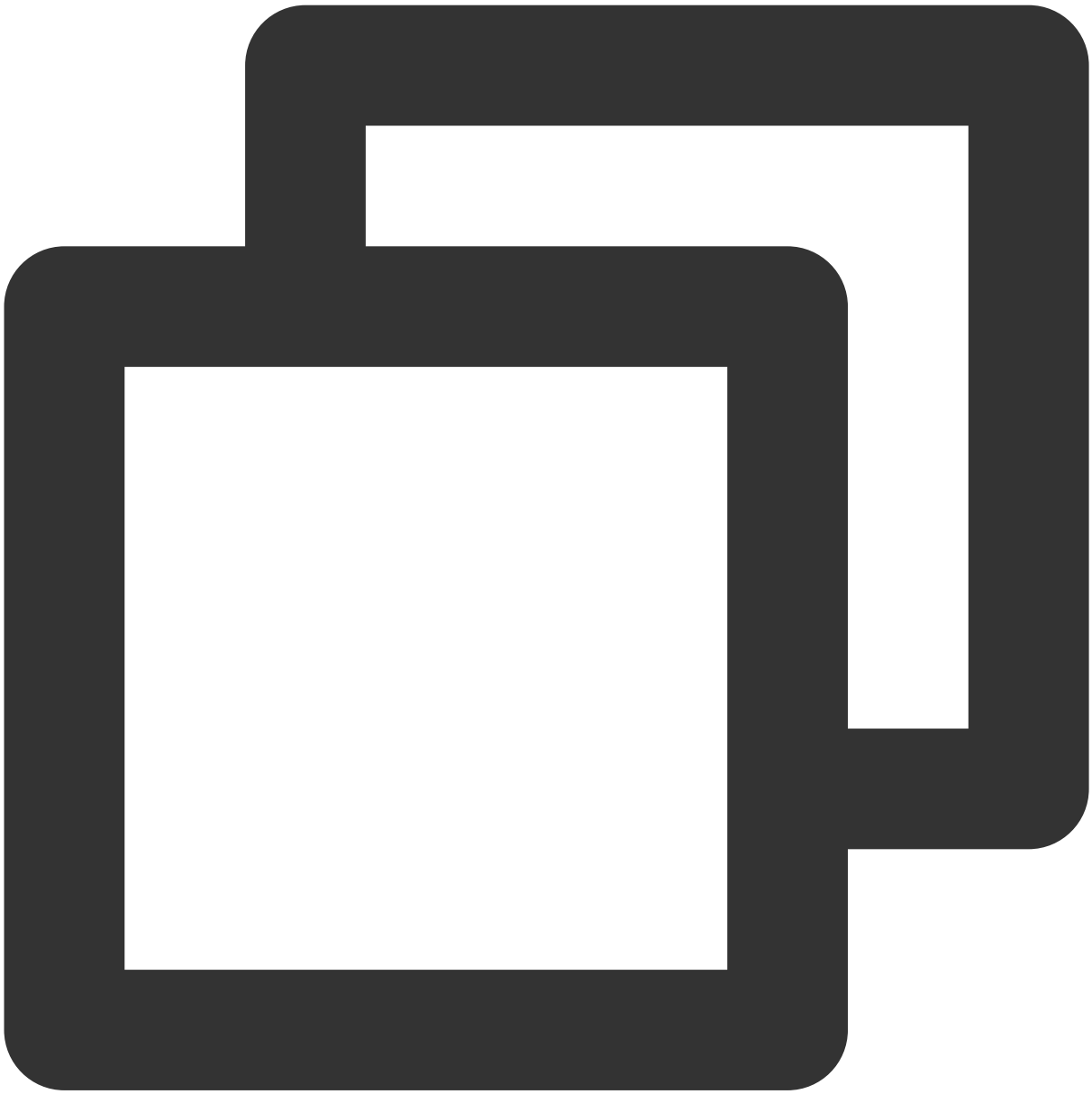
画中画控制器 TXPipController 为 superplayer_widget 中封装的画中画工具，**必须与 SuperPlayerView 搭配起来使用**。

进入画中画会自动关闭当前界面，并回调提前设置的监听方法，在回调的方法中可以保存播放器当前界面的必要参数。画中画还原之后，会重新将之前的界面 push 回来，并传递之前保存的参数。
使用该控制器的时候，画中画和播放器只能存在一个实例，当重新进入播放器界面的时候，画中画会自动关闭。
6.1 在自己的项目的入口处，如main.dart，调用TXPipController设置画中画控制跳转，跳转的页面为用于进入画中画的播放器页面。
可根据自身项目情况设置不同的界面，代码实例如下：



```
TXPipController.instance.setNavigatorHandle((params) {
  navigatorKey.currentState?.push(MaterialPageRoute(builder: (_) => DemoSuperPlayer
});
```

6.2 设置画中画的播放页面监听，需要实现 `TXPipPlayerRestorePage` 方法，设置之后，当即将进入画中画时，控制器会回调 `void onNeedSavePipPageState(Map<String, dynamic> params)` 方法，此时可以在 params 中存入当前页面需要的参数。



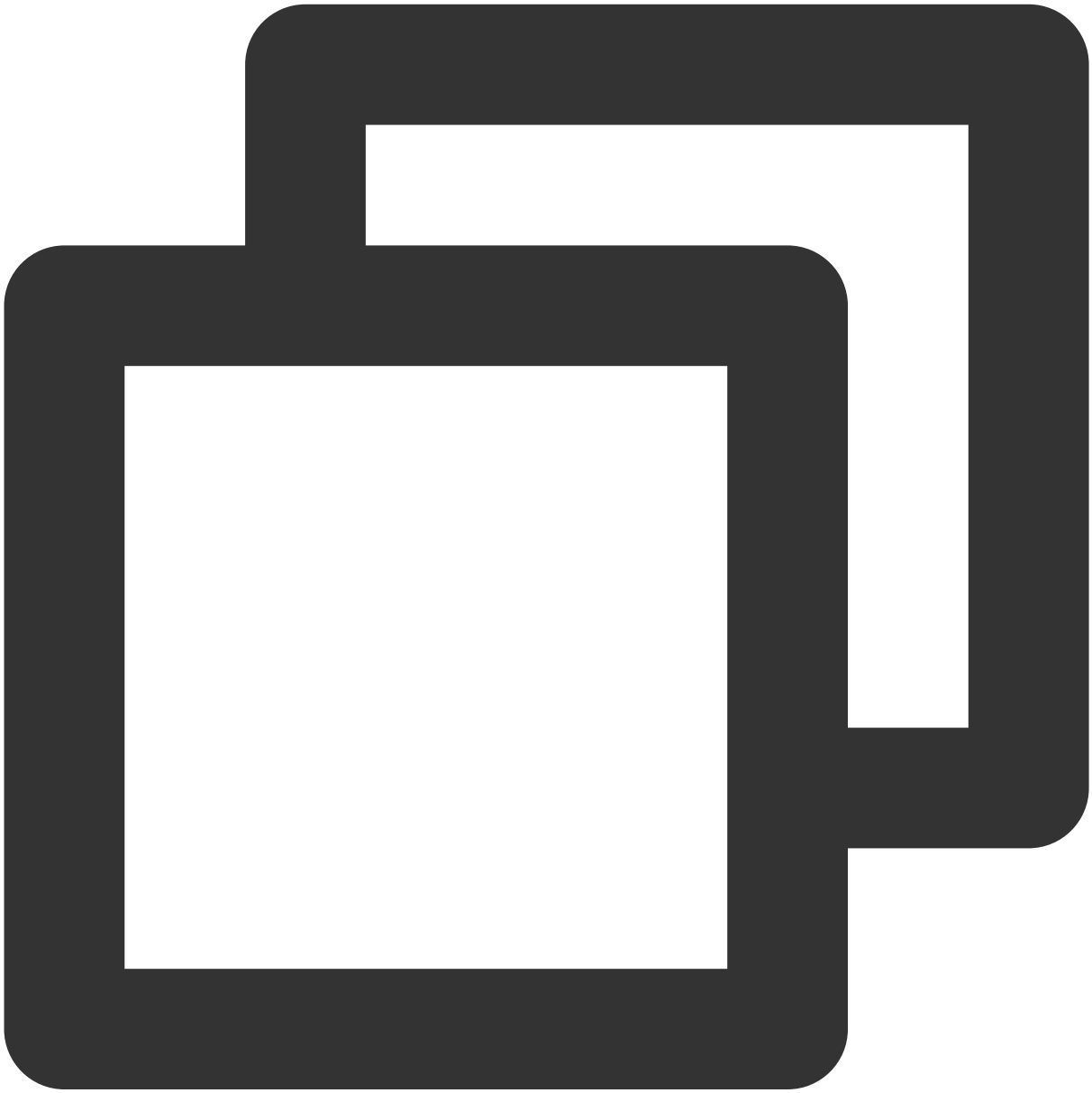
```
TXPipController.instance.setPipPlayerPage(this);
```

随后，当用户点击 SuperPlayerView 上的进入画中画按钮的时候，会调用 `SuperPlayerView` 的 `_onEnterPipMode` 内部方法进入画中画，也可以自行调用 `SuperPlayerController` 的 `enterPictureInPictureMode` 方法进入。

3、视频下载

下载视频

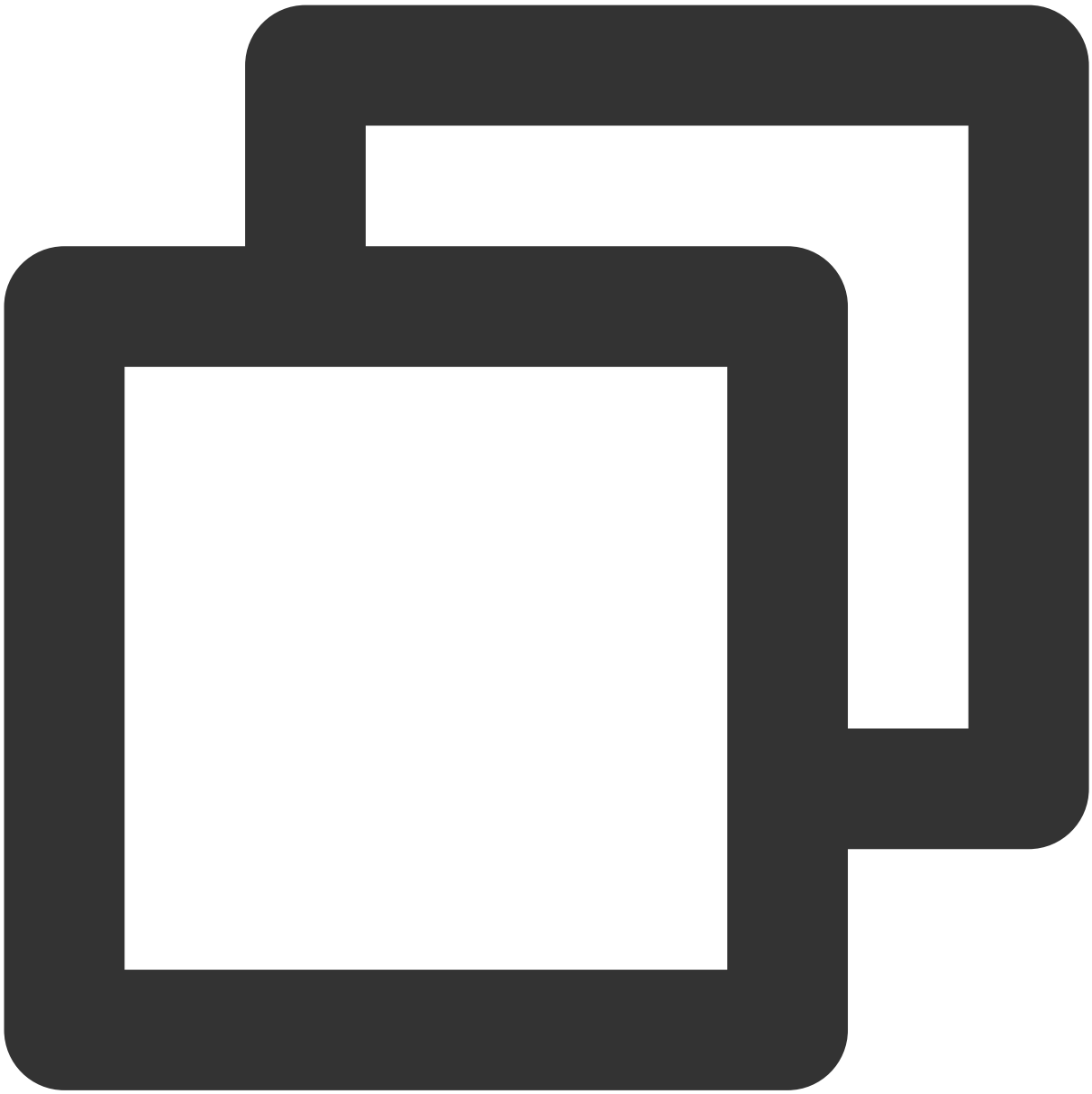
1. 使用播放器组件的视频下载，首先需要把SuperPlayerModel中的 `isEnableDownload` 打开，该字段默认关闭。



```
SuperPlayerModel model = SuperPlayerModel();
// 打开视频下载能力
model.isEnableDownload = true;
```

播放器组件目前只会在点播播放模式下启用下载。

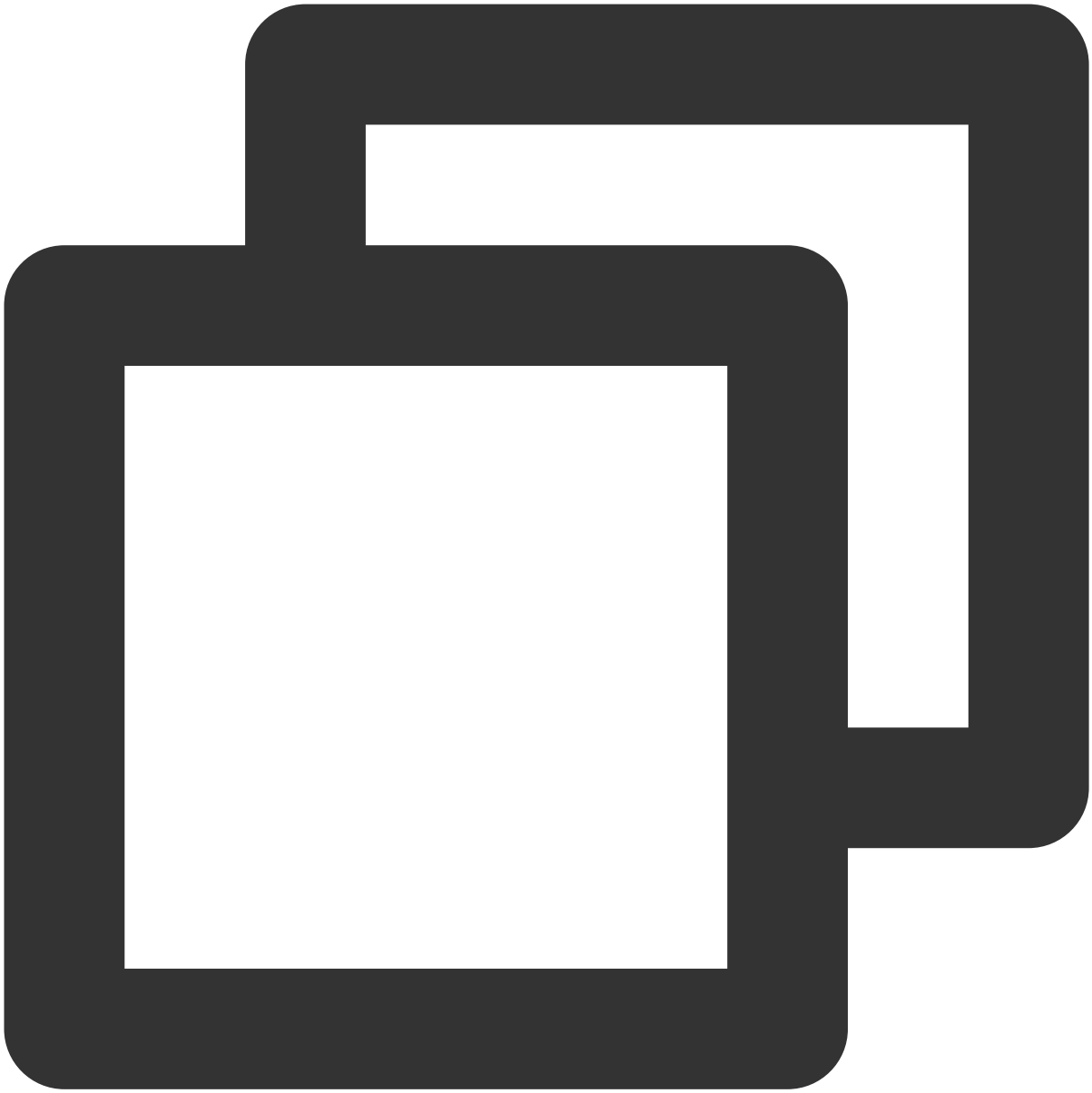
2. 使用 `SuperPlayerController` 的 `startDownload` 方法，可以直接下载当前播放器正在播放的视频，对应的是当前播放视频的清晰度。也可是使用 `DownloadHelper` 下载指定视频，如下：



```
DownloadHelper.instance.startDownloadBySize(videoModel, videoWidth, videoHeight);
```

使用 `DownloadHelper` 的 startDownloadBySize，可下载指定分辨率的视频，如果没有该分辨率，会下载相近分辨率的视频。

除了以上接口以外，也可选择传入画质 ID 或者 mediaInfo 直接下载。

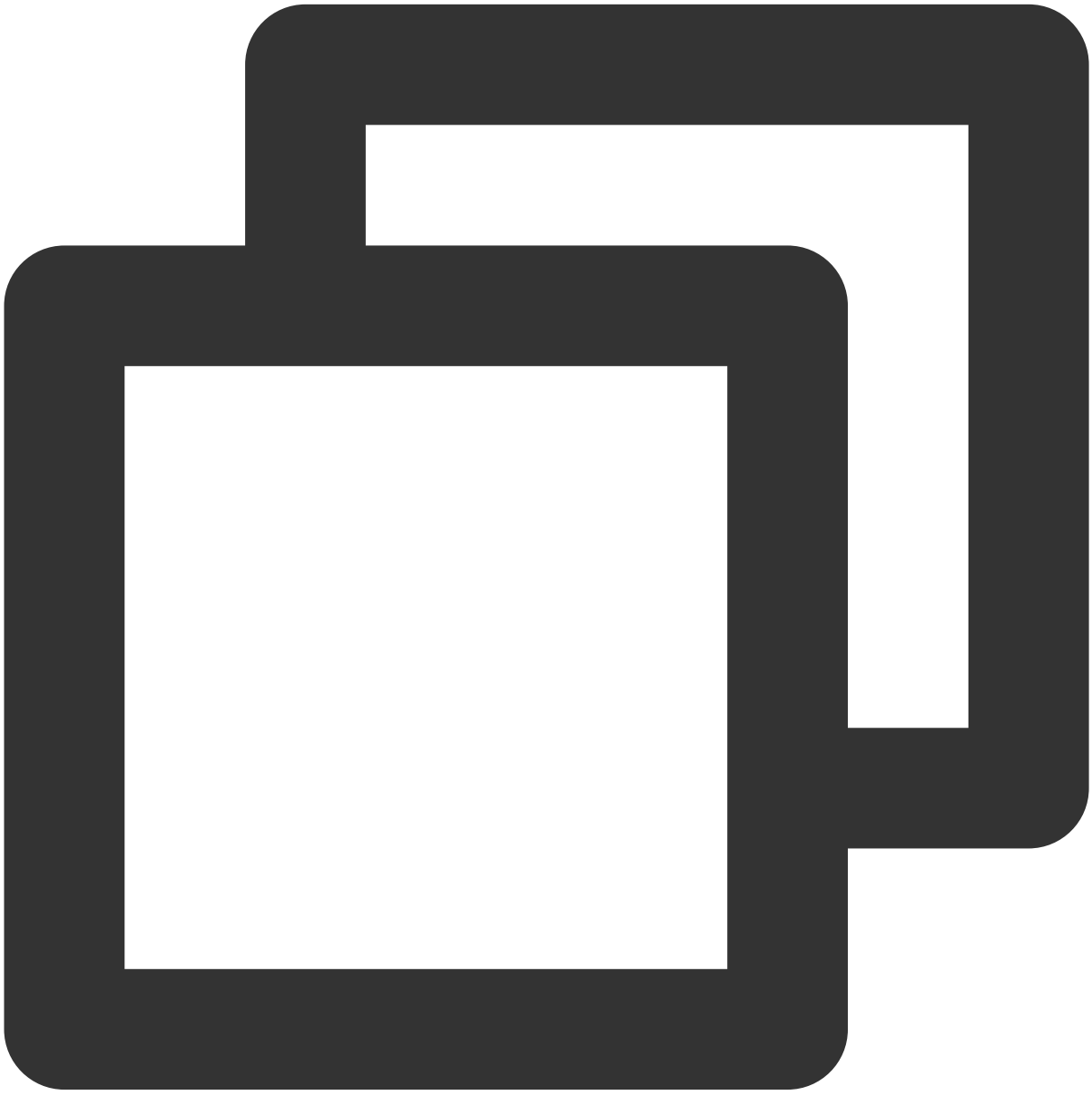


```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720，期望下载此分辨率的流，quality
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
// 使用画质ID下载
```

```
DownloadHelper.instance.startDownload(videoModel, qualityId);
// 使用mediaInfo下载
DownloadHelper.instance.startDownloadOrg(mediaInfo);
```

3. 画质 ID 转换。

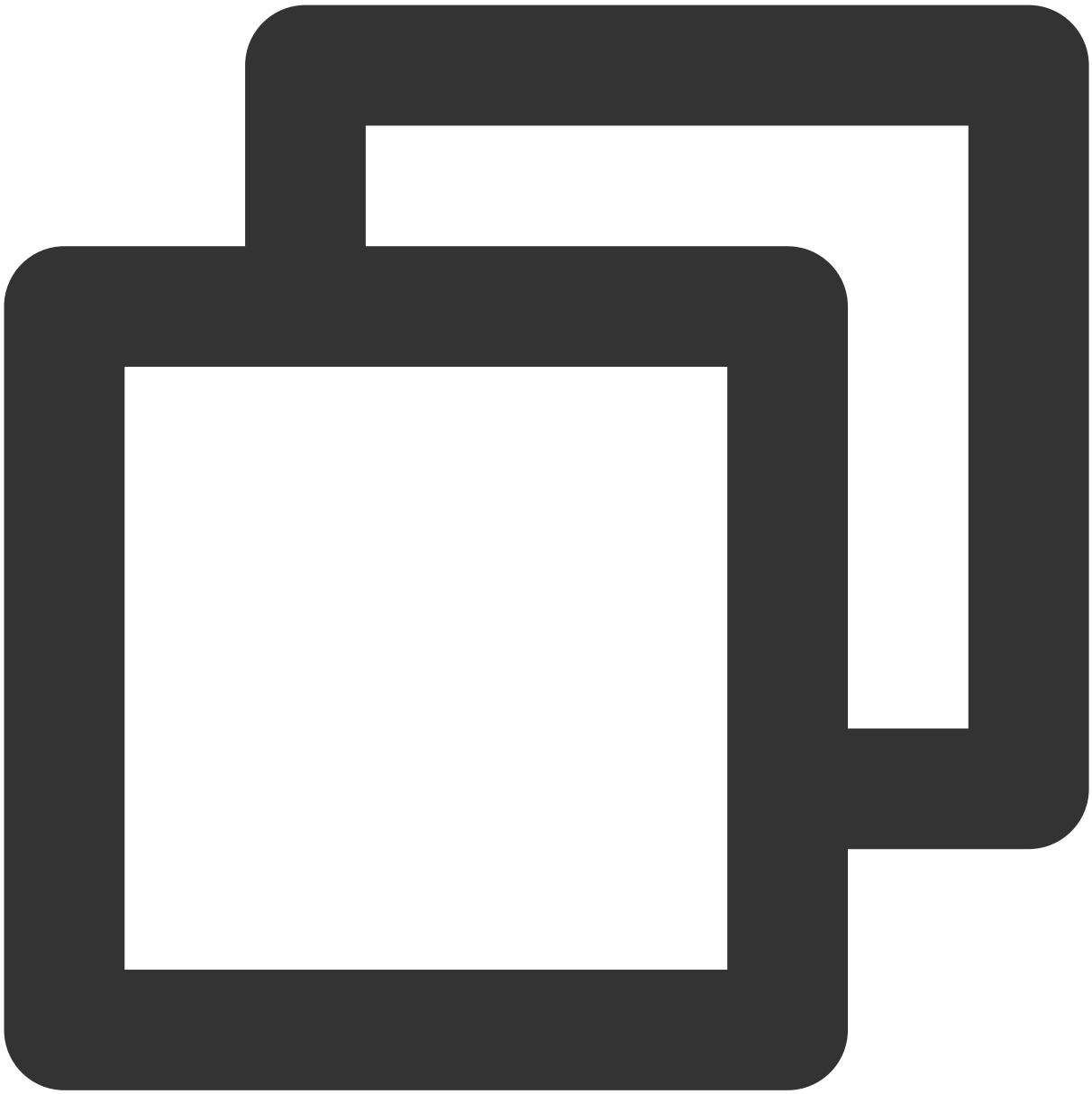
点播的 `CommonUtils` 提供了 `getDownloadQualityBySize` 方法，用于将分辨率转为对应的画质 ID。



```
CommonUtils.getDownloadQualityBySize(width, height);
```

停止下载视频

使用 `DownloadHelper` 的 `stopDownload` 方法可以停止对应的视频下载，示例如下：

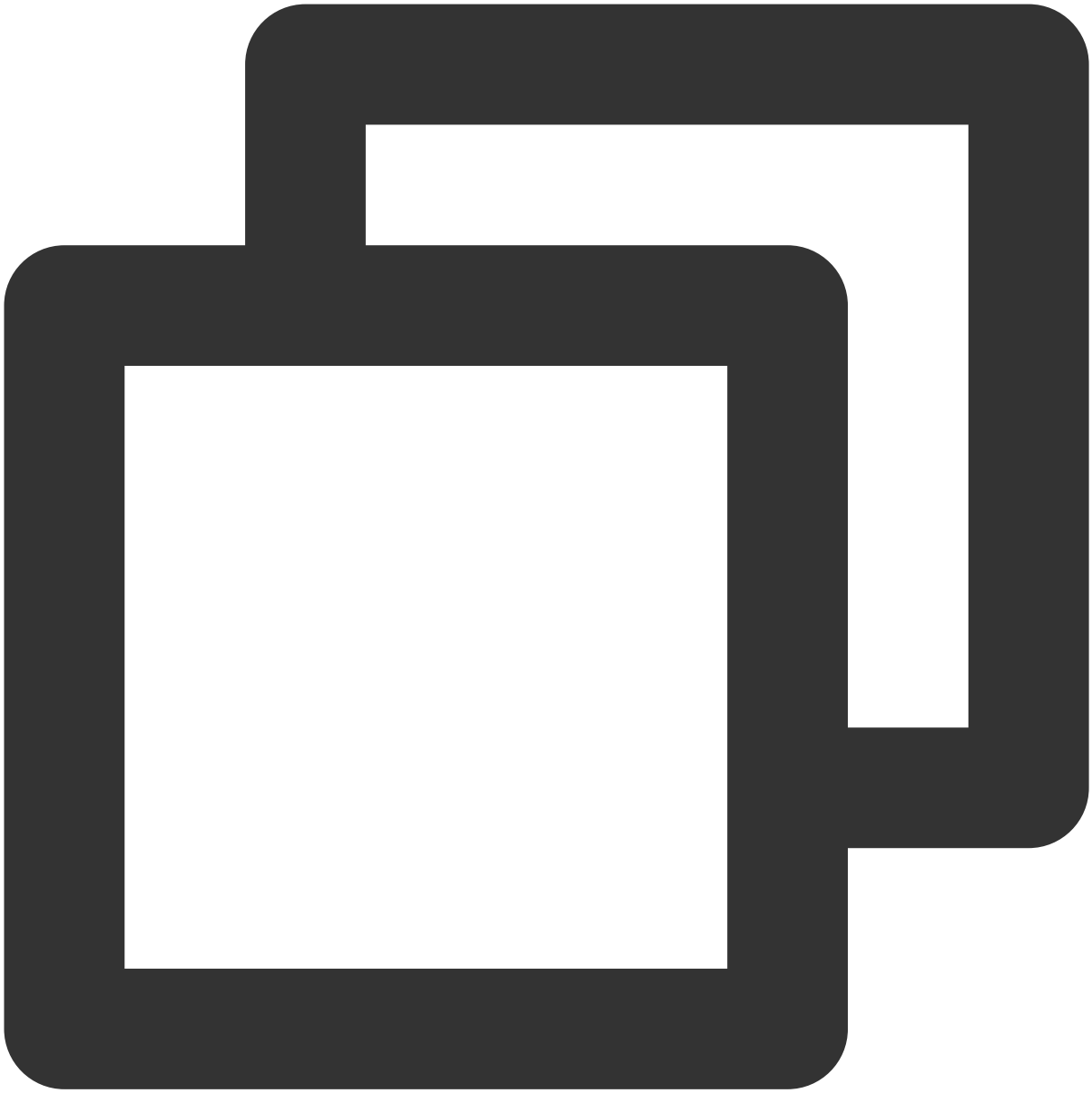


```
DownloadHelper.instance.stopDownload(mediaInfo);
```

mediaInfo 可通过 `DownloadHelper` 的 `getMediaInfoByCurrent` 方法获取，或者使用 `TXVodDownloadController` 的 `getDownloadList` 获得下载信息。

删除下载视频

使用 `DownloadHelper` 的 `deleteDownload` 方法，可以删除对应的视频。



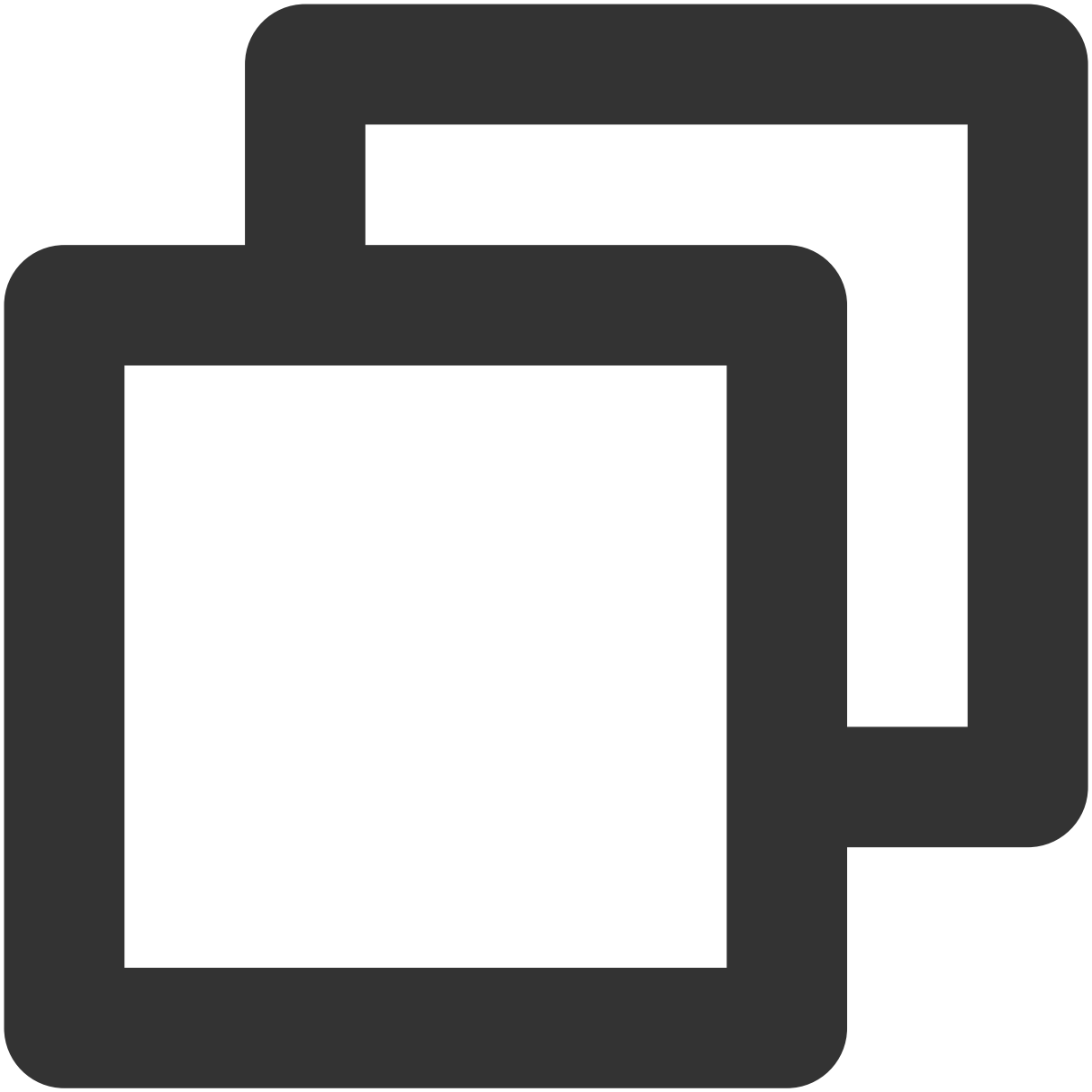
```
bool deleteResult = await DownloadHelper.instance.deleteDownload(downloadModel.medi
```

deleteDownload 会返回删除的结果，来判断是否删除成功。

下载状态

`DownloadHelper` 提供了基本的 `isDownloaded` 方法判断视频是否已经下载。也可以注册监听来实时判断下载状态。

`DownloadHelper` 对下载事件进行了分发，可通过如下代码进行事件注册。

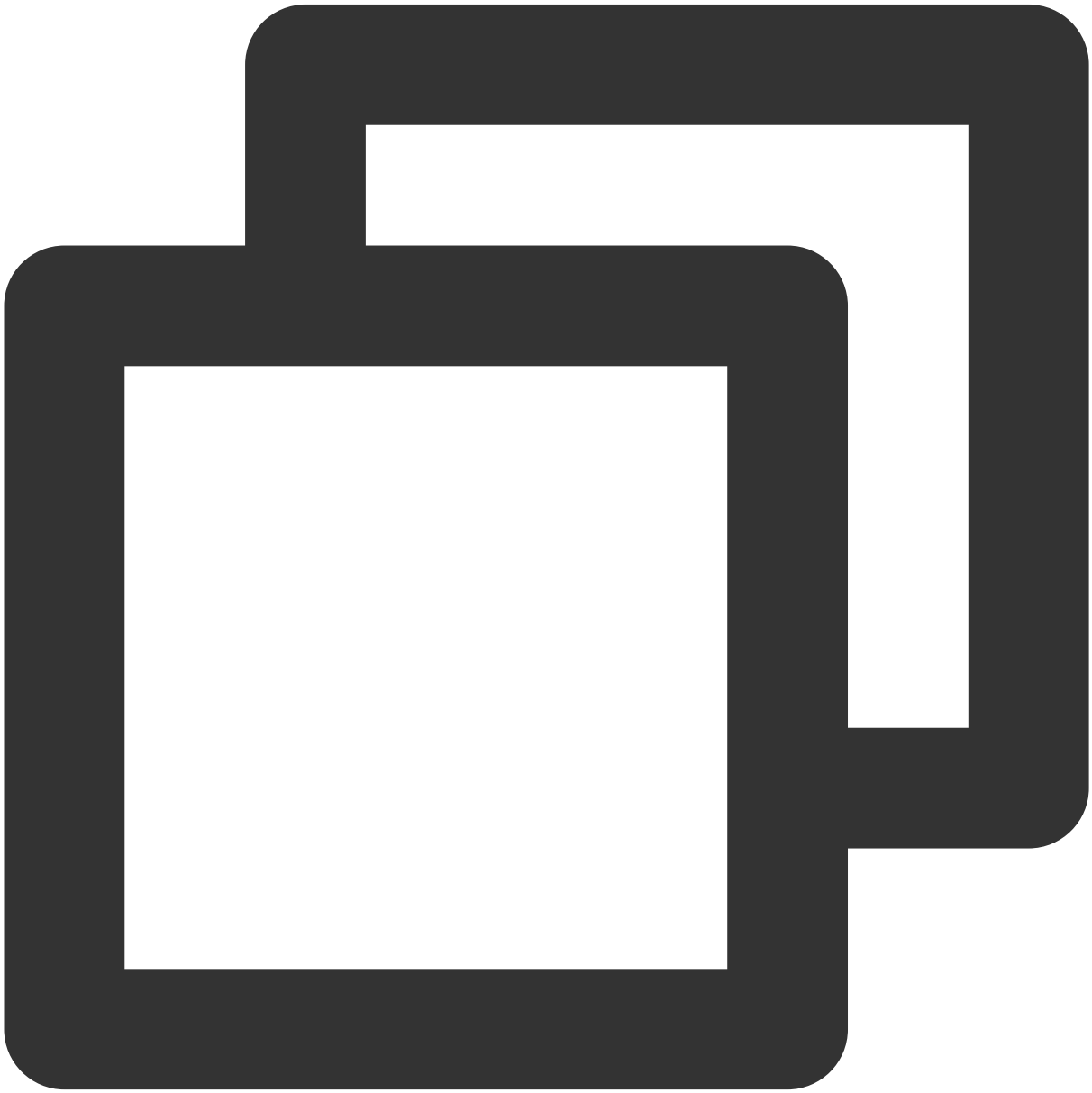


```
// 注册下载事件监听
DownloadHelper.instance.addDownloadListener(FTXDownloadListener((event, info) {
      // 下载状态变化
    }, (errorCode, errorMsg, info) {
      // 下载错误回调
    }));
// 移除下载事件监听
DownloadHelper.instance.removeDownloadListener(listener);
```

此外，还可以通过 `TXVodDownloadController.instance.getDownloadInfo(mediaInfo)` 方法或者 `TXVodDownloadController.instance.getDownloadList()` 方法直接查询 mediaInfo 中的 downloadState 来判断下载状态。

播放下载的视频

`TXVodDownloadController.instance.getDownloadInfo(mediaInfo)` 和 `TXVodDownloadController.instance.getDownloadList()` 获得到的视频信息中有个 playPath 字段，使用 TXVodPlayerController 直接播放即可。

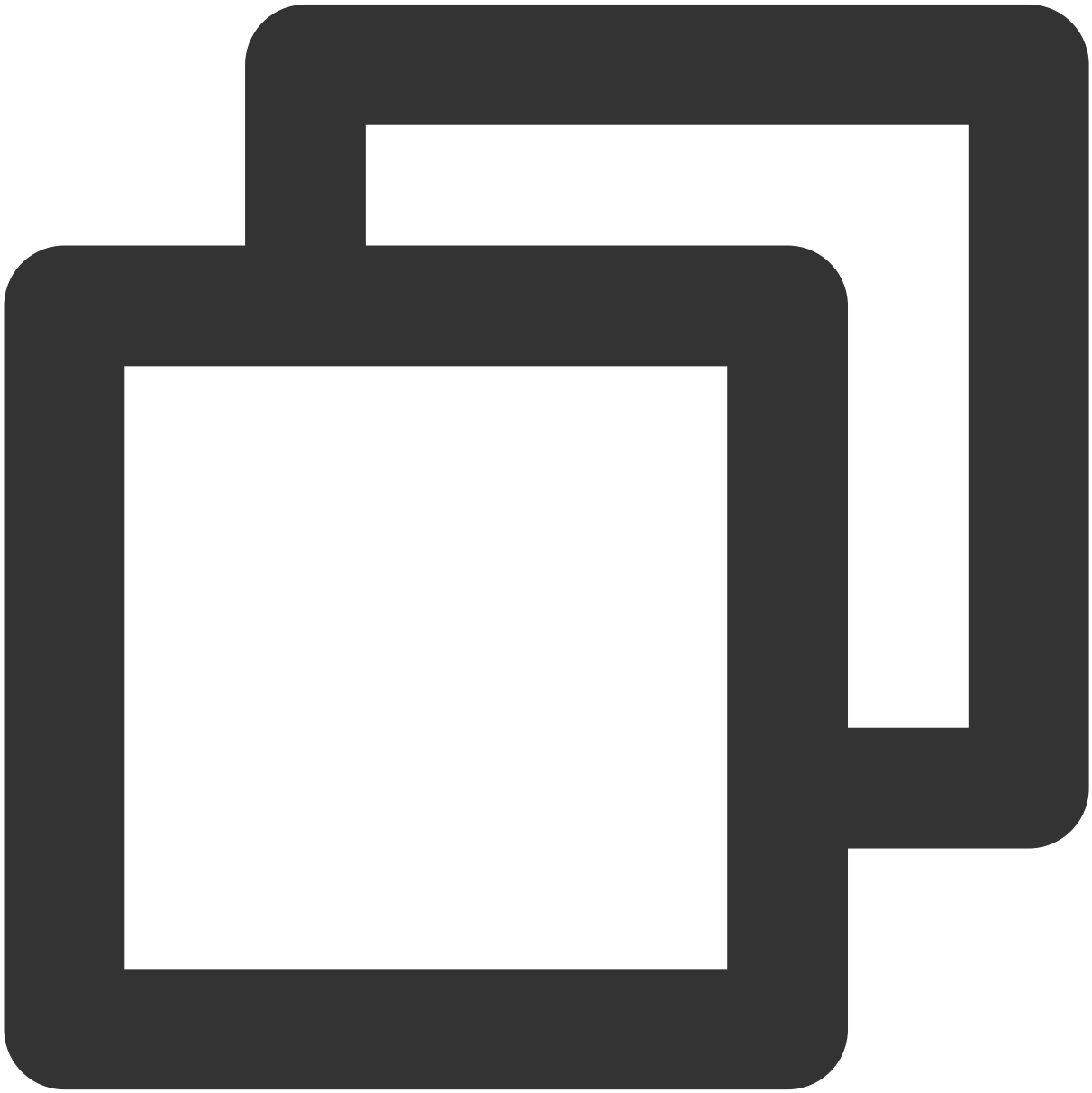


```
controller.startVodPlay(mediaInfo.playPath);
```

4、横竖屏的使用

横竖屏切换配置

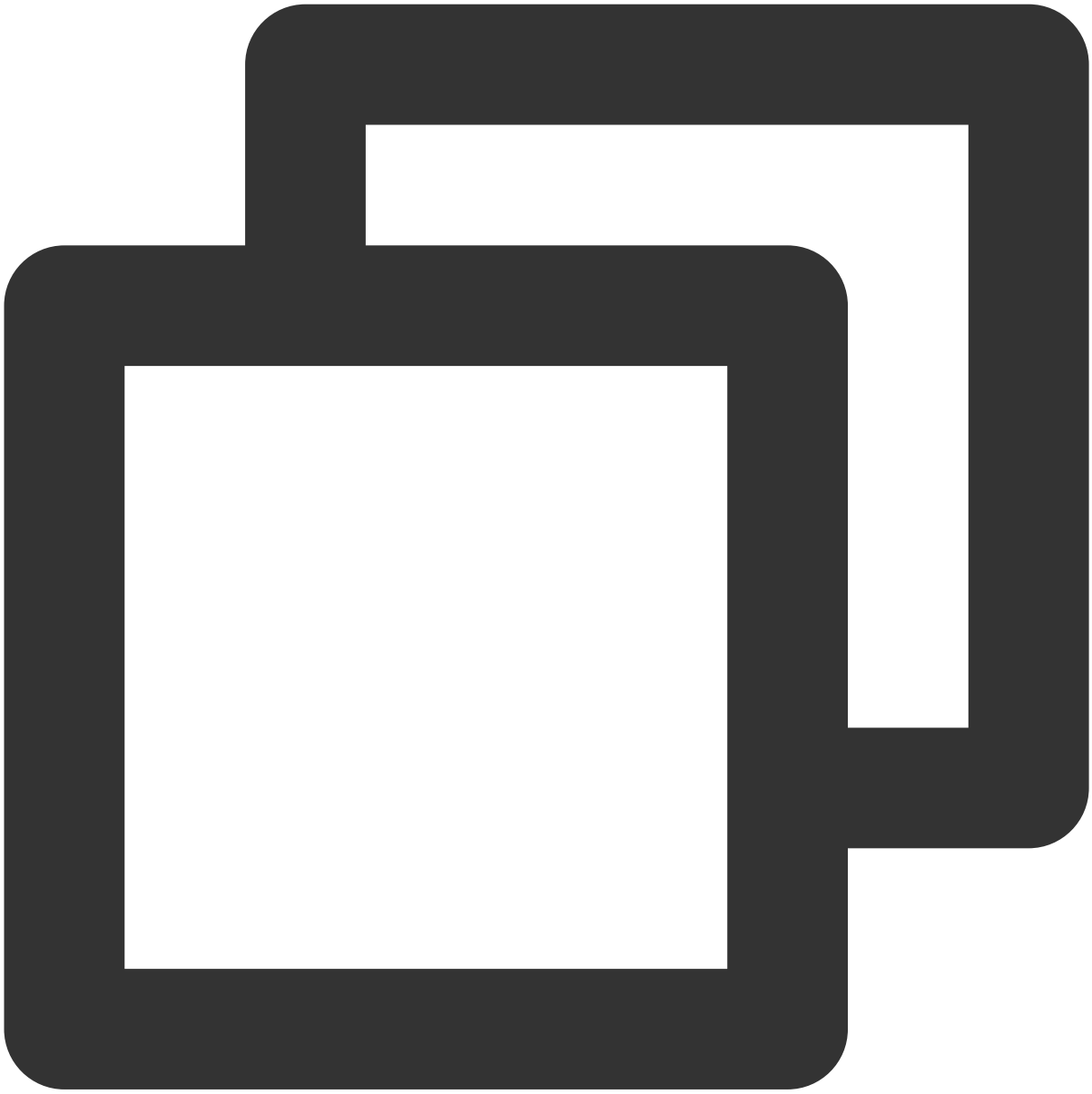
播放器组件横竖屏的切换，iOS 需要使用 Xcode 打开，打开项目配置，General 分页下的 Deployment 标签下，勾选上 `Landscape left` 和 `Landscaoe right` 。确保 iOS 设备能够支持横屏。
如果希望自己的 App 其他页面稳定保持竖屏，不受横竖屏自动旋转影响，需要在自己项目下的入口处，配置竖屏。
代码如下：



```
SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
```

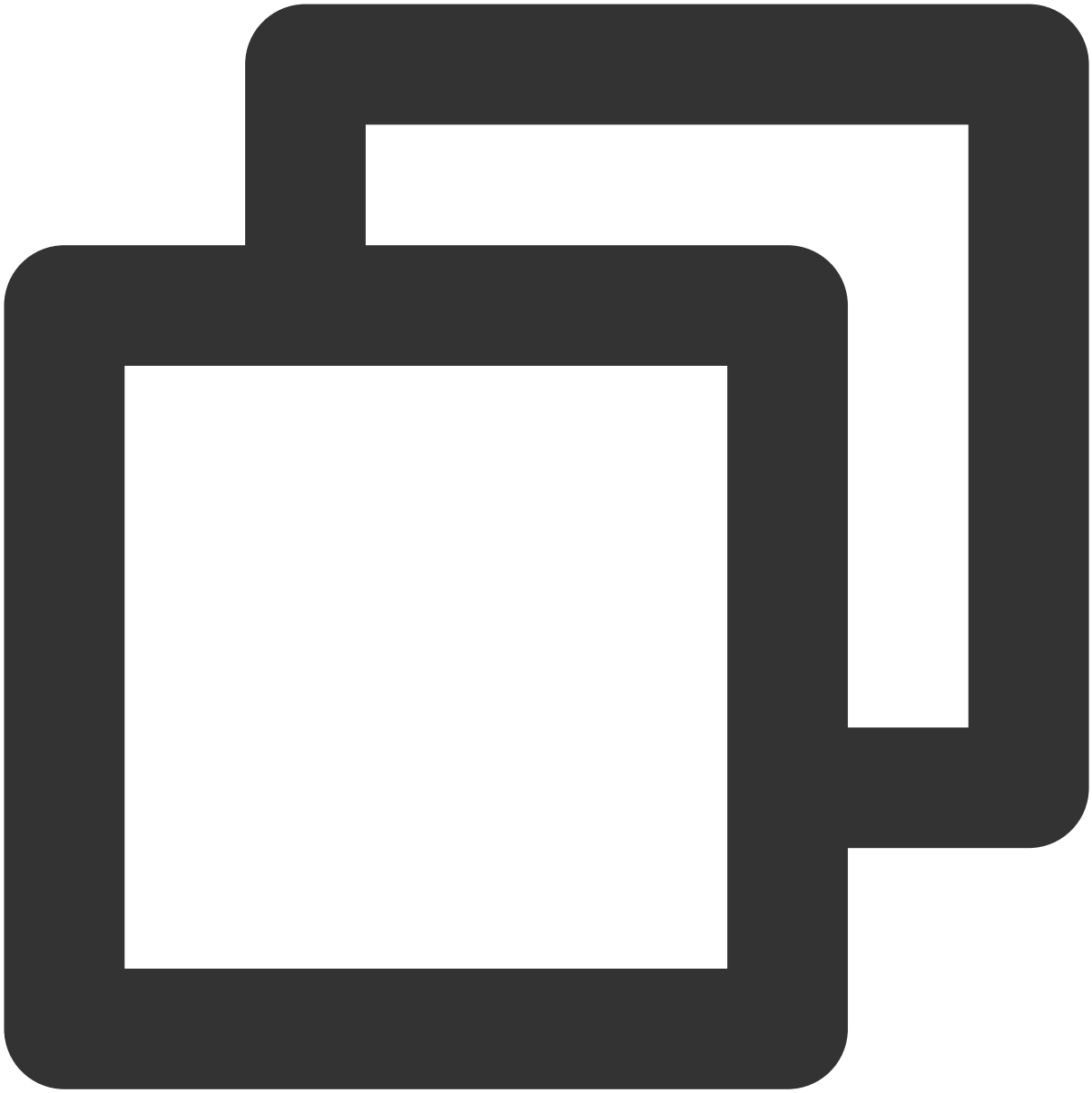
根据 sensor 配置自动全屏

Android 端需要调用如下方法，来开启对 sensor 的监听。



```
SuperPlayerPlugin.startVideoOrientationService();
```

调用之后，在 Android 设备上，将会对 Android sensor 进行监听，会通过 `SuperPlayerPlugin.instance.onEventBroadcast` 对 flutter 侧发送旋转事件。播放器组件内部也会自动根据该事件旋转播放器。监听使用范例如下：



```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
    int eventCode = event["event"];
    if (eventCode == TXVodPlayEvent.EVENT_ORIENTATION_CHANGED ) {
      int orientation = event[TXVodPlayEvent.EXTRA_NAME_ORIENTATION];
      // do orientation
    }
  });
```

Demo 体验

更多功能和调试 Demo 体验，请 单击这里，**运行该 demo 的时候，需要在 demo_config 中设置自己的播放器 license，并在 Android 和 iOS 配置中，将包名和 bundleId 修改为自己签名的包名和 bundleId。**

无 UI 集成方案
Web 端集成
TCPlayer 集成指引

最近更新时间：2024-04-11 16:48:50

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

概述

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

协议支持

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

说明：
视频编码格式仅支持 H.264 编码。
播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。
HLS、FLV 视频在部分浏览器环境播放需要依赖 Media Source Extensions。
在不支持 WebRTC 的浏览器环境，传入播放器的 WebRTC 地址会自动进行协议转换来更好的支持媒体播放。

功能支持

功能\\浏览器	Chrome	Firefox	Edge	QQ浏览器	Mac Safari	iOS Safari	微信	Android Chrome	IE 11
播放器尺寸设置	✓	✓	✓	✓	✓	✓	✓	✓	✓
续播功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
倍速播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
缩略图预览	✓	✓	✓	✓	-	-	-	-	✓
切换 fileID 播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
镜像功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
进度条标记	✓	✓	✓	✓	✓	-	-	-	✓
HLS 自适应码率	✓	✓	✓	✓	✓	✓	✓	✓	✓
Referer 防盗链	✓	✓	✓	✓	✓	✓	✓	-	✓
清晰度切换提示	✓	✓	✓	✓	-	-	-	✓	✓
试看功能	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 标准加密播放	✓	✓	✓	✓	✓	✓	✓	✓	✓
HLS 私有加密播放	✓	✓	✓	-	-	-	Android：✓ iOS: -	✓	✓
视频统计信息	✓	✓	✓	✓	-	-	-	-	-
视频数据监控	✓	✓	✓	✓	-	-	-	-	-
自定义提	✓	✓	✓	✓	✓	✓	✓	✓	✓

示文案									
自定义UI	✓	✓	✓	✓	✓	✓	✓	✓	✓
弹幕	✓	✓	✓	✓	✓	✓	✓	✓	✓
水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
幽灵水印	✓	✓	✓	✓	✓	✓	✓	✓	✓
视频列表	✓	✓	✓	✓	✓	✓	✓	✓	✓
弱网追帧	✓	✓	✓	✓	✓	✓	✓	✓	✓

说明：

视频编码格式仅支持 H.264 编码。

Chrome、Firefox 包括 Windows、macOS 平台。

Chrome、Firefox、Edge 及 QQ 浏览器播放 HLS 需要加载 `hls.js` 。

Referer 防盗链功能是基于 HTTP 请求头的 Referer 字段实现的，部分 Android 浏览器发起的 HTTP 请求不会携带 Referer 字段。

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

准备工作

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

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

说明：

1. 播放器 Web 端基础版 License 可免费申请，申请后有效期默认1年；在有效期剩余时间小于30天时，可免费续期。
2. 为方便本地开发，播放器不会校验 localhost 或者 127.0.01，因此申请 License 时不需要申请这类本地服务域名。

集成指引

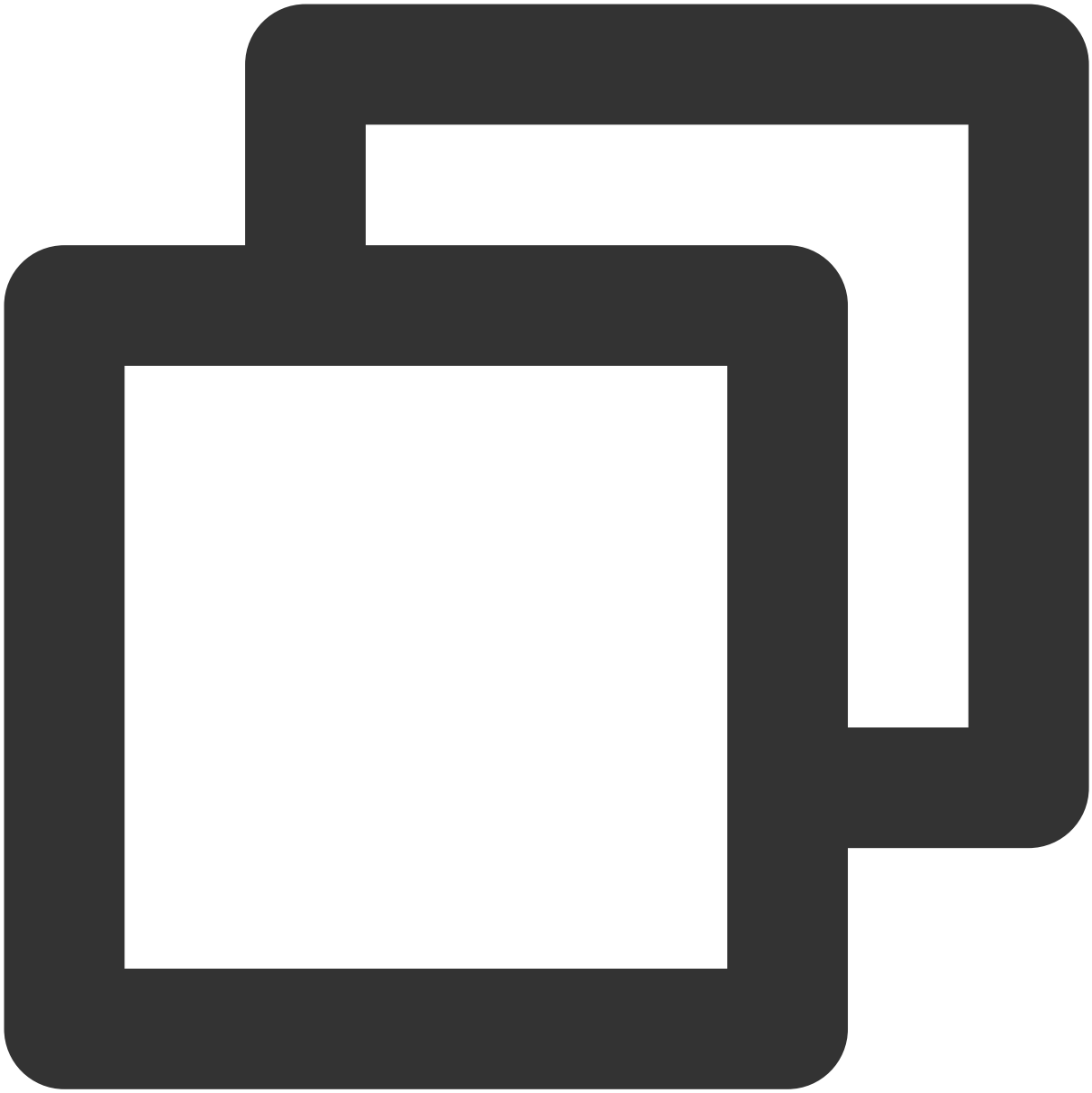
通过以下步骤，您就可以在网页上添加一个视频播放器。

步骤1：在页面中引入文件

播放器 SDK 支持 cdn 和 npm 两种集成方式：

1. 通过 cdn 集成

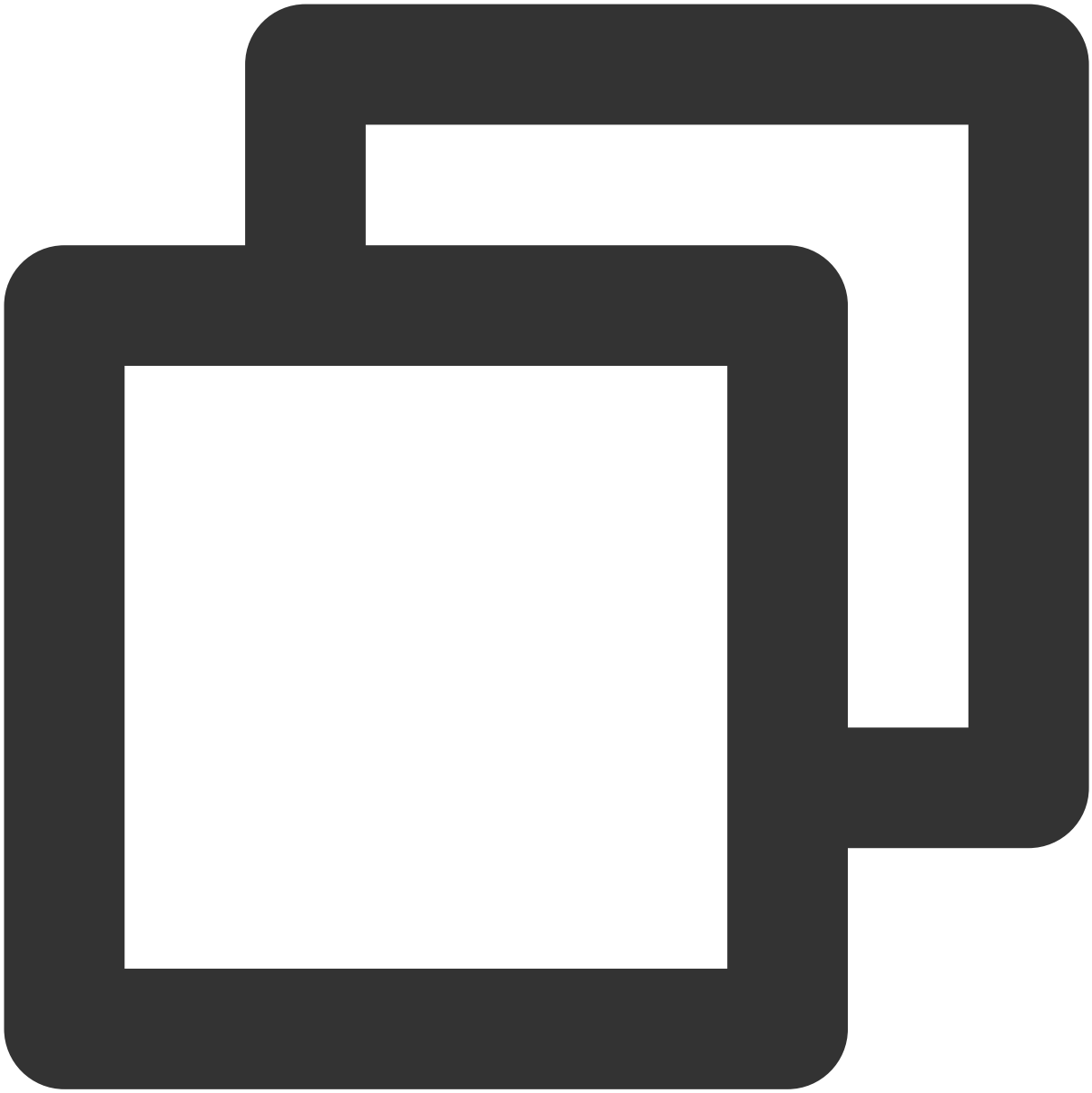
在本地的项目工程内新建 index.html 文件，html 页面内引入播放器样式文件与脚本文件：



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

建议在使用播放器 SDK 的时候自行部署资源，单击下载播放器资源。部署解压后的文件夹，不能调整文件夹里面的目录，避免资源互相引用异常。

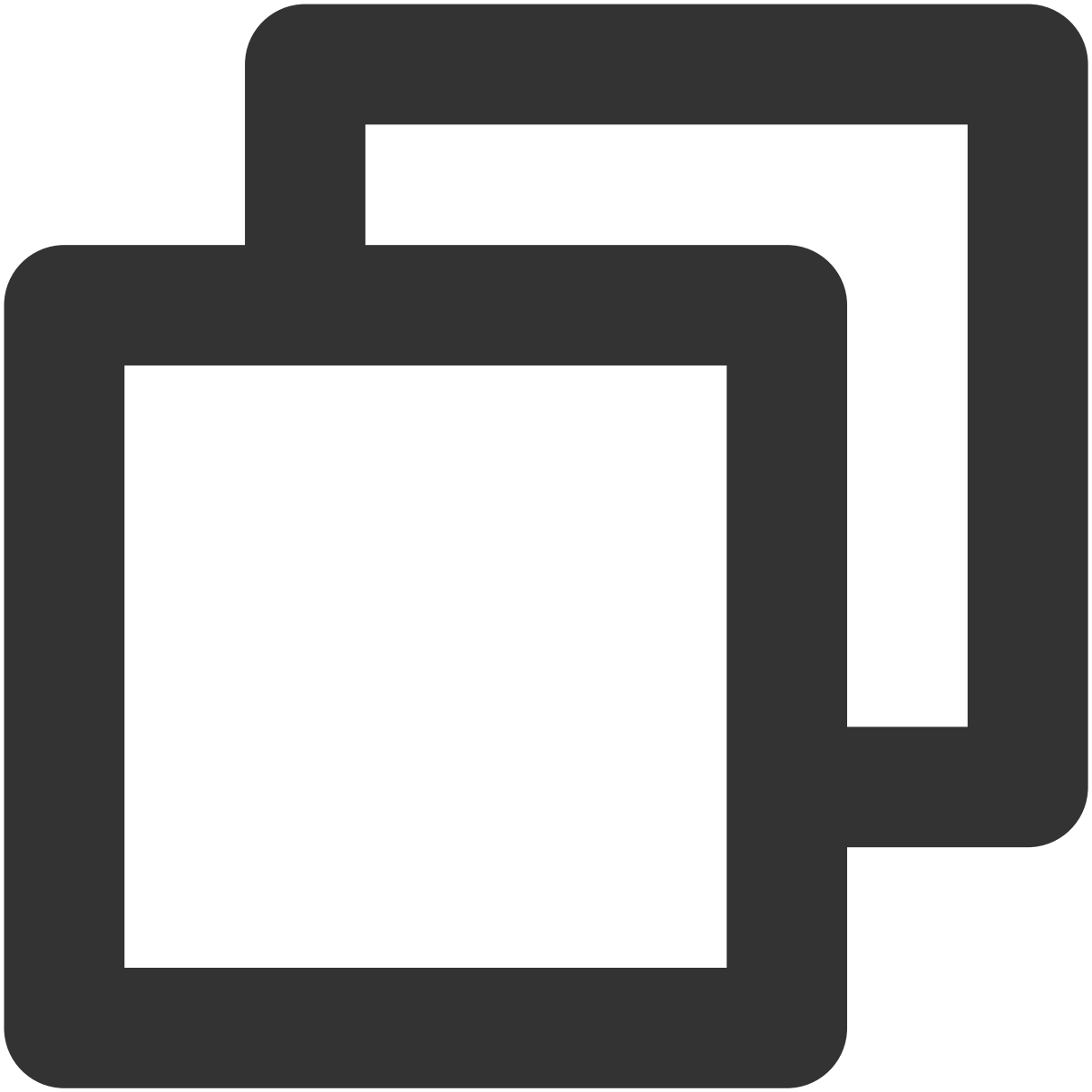
如果您部署的地址为 `aaa.xxx.ccc` ，在合适的地方引入播放器样式文件与脚本文件，自行部署情况下，需要手动引用资源包文件夹 libs 下面的依赖文件，否则会默认请求腾讯云 cdn 文件。



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

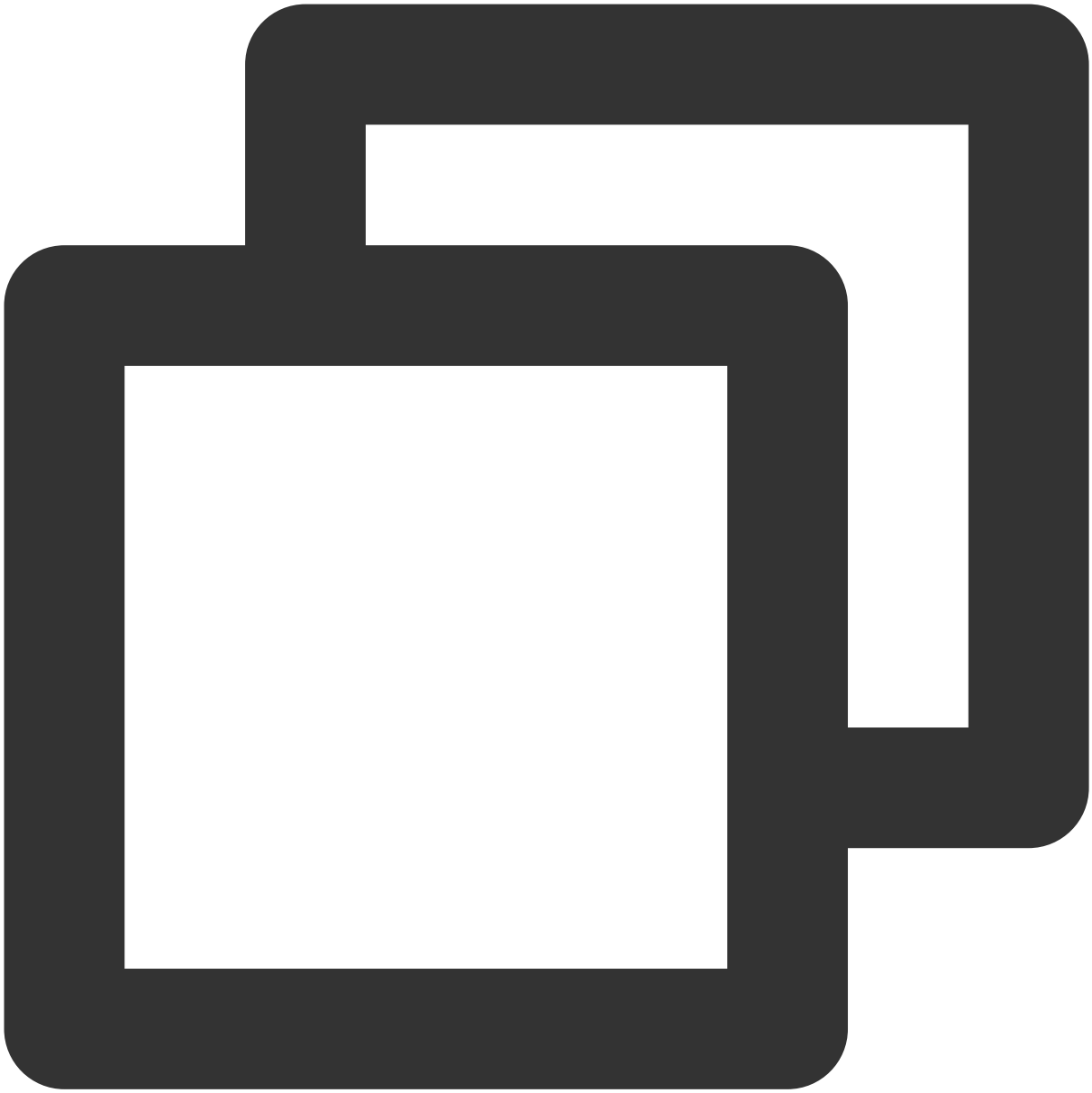
2. 通过 npm 集成

首先安装 tcplayer 的 npm 包：



```
npm install tcplayer.js
```

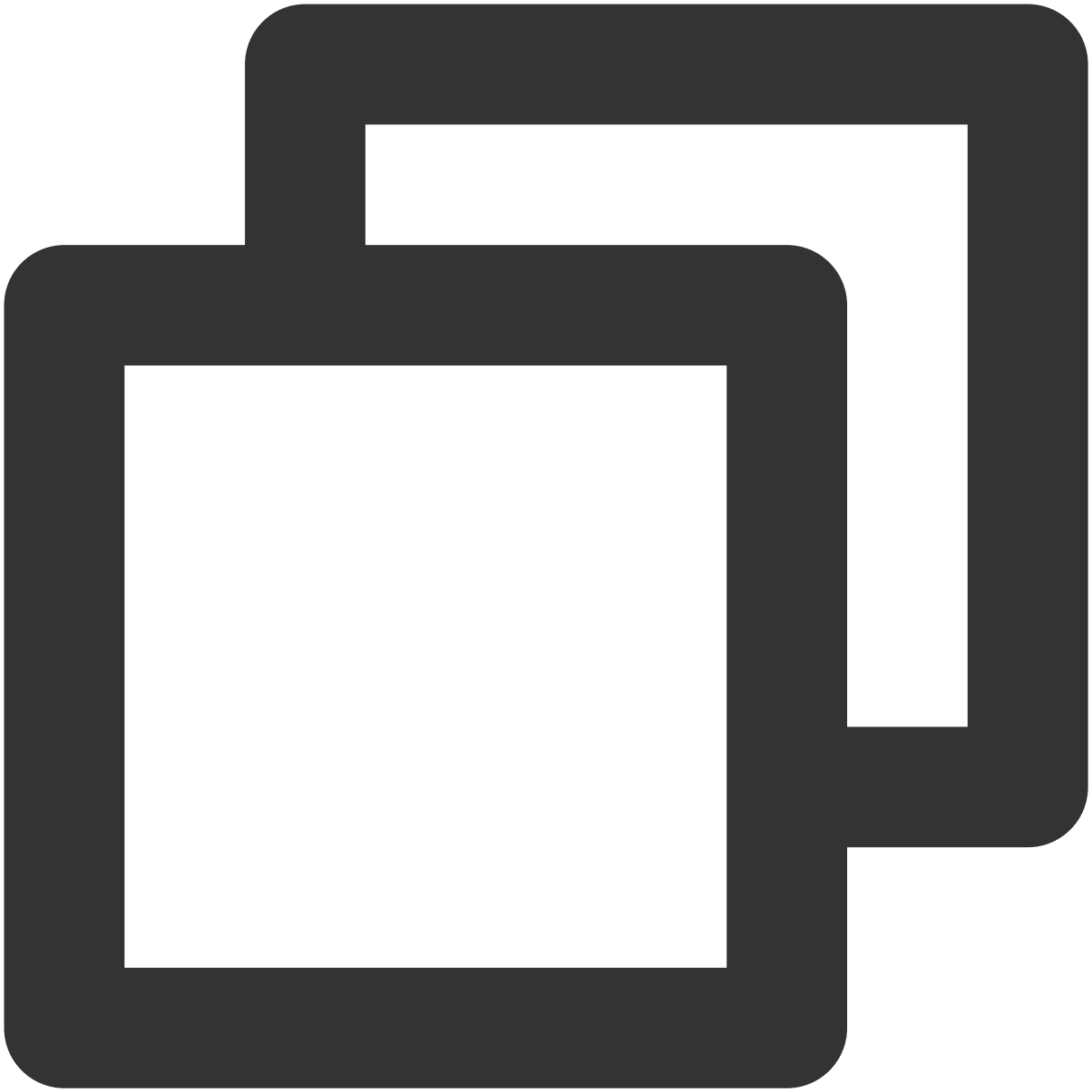
导入 SDK 和样式文件：



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

步骤2：放置播放器容器

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



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

说明：

播放器容器必须为 `<video>` 标签。

示例中的 `player-container-id` 为播放器容器的 ID，可自行设置。

播放器容器区域的尺寸，建议通过 CSS 进行设置，通过 CSS 设置比属性设置更灵活，可以实现例如铺满全屏、容器自适应等效果。

示例中的 `preload` 属性规定是否在页面加载后载入视频，通常为了更快的播放视频，会设置为 `auto` ，其他可选值： `meta` （当页面加载后只载入元数据）， `none` （当页面加载后不载入视频），移动端由于系统限制不会自动加载视频。

`playsinline` 和 `webkit-playsinline` 这几个属性是为了在标准移动端浏览器不劫持视频播放的情况下实现行内播放，此处仅作示例，请按需使用。

设置 `x5-playsinline` 属性在 TBS 内核会使用 X5 UI 的播放器。

步骤3：播放器初始化

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

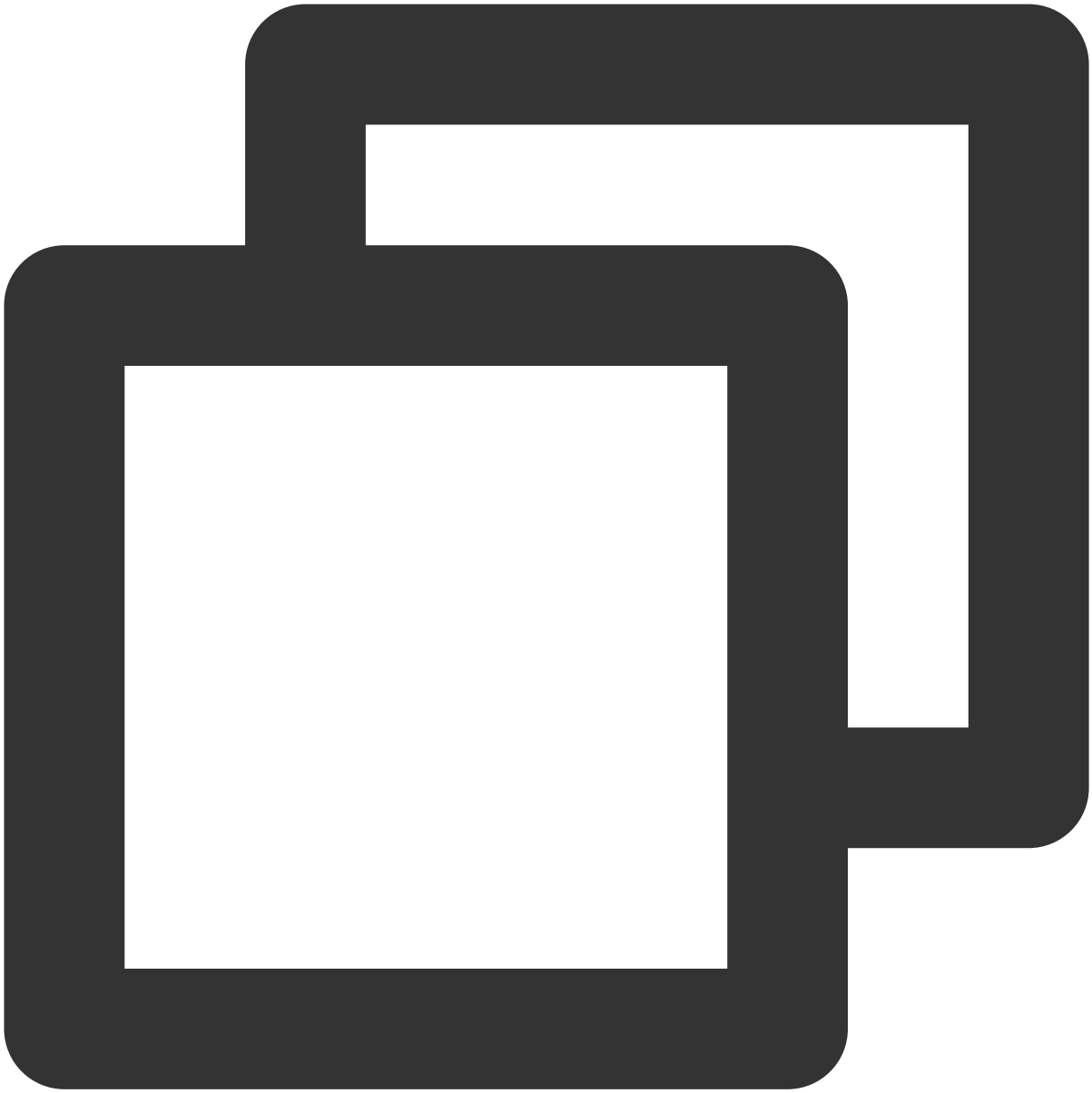
点播播放：播放器可以通过 FileID 播放腾讯云点播媒体资源，云点播具体流程请参见 使用播放器播放 文档。

直播播放：播放器通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播 URL 生成方式可参见 自主拼装直播 URL。

通过 URL 播放（直播、点播）

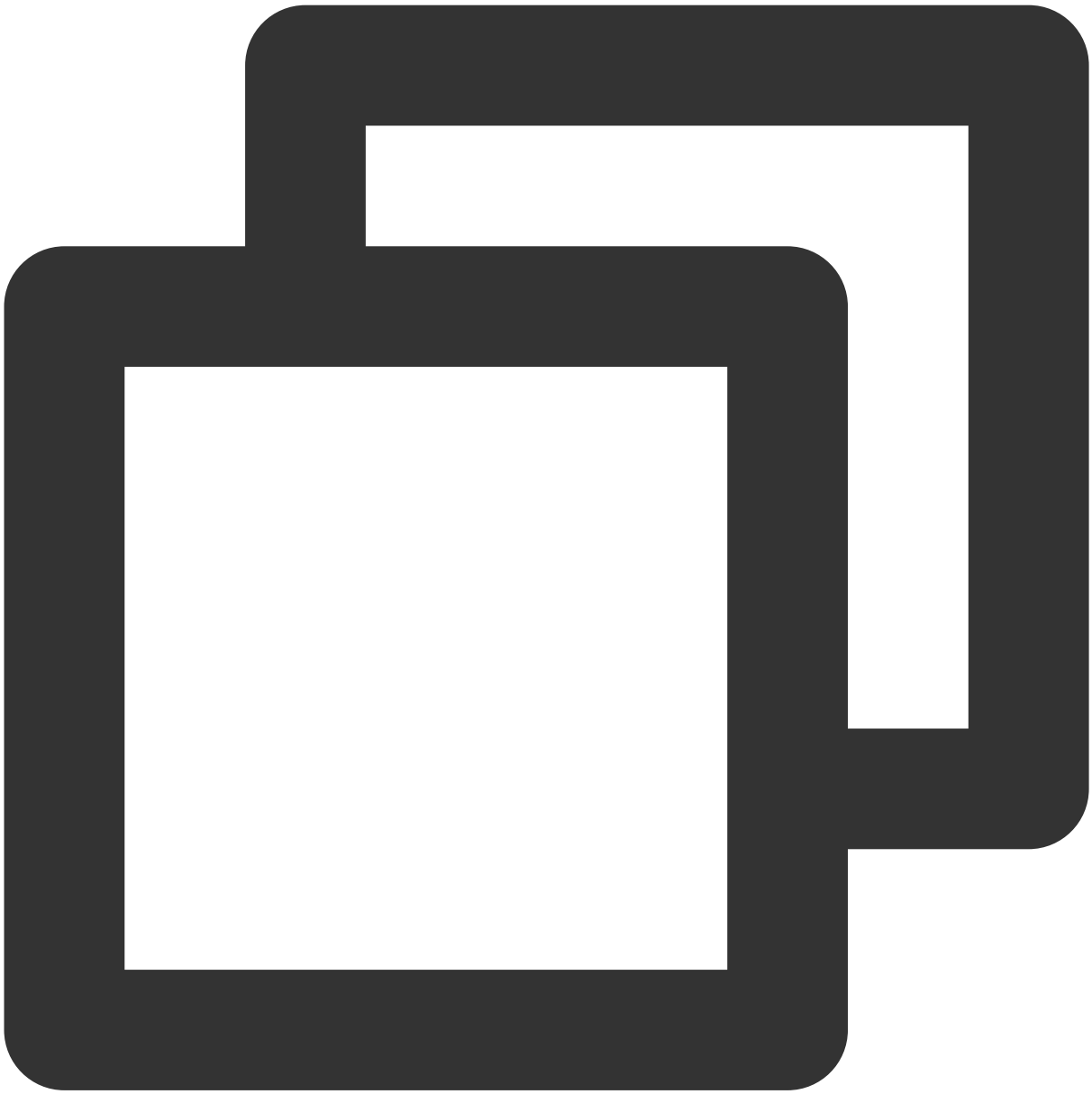
通过 FileID 播放（点播）

在页面初始化之后，调用播放器实例上的方法，将 URL 地址传入方法。



```
// player-container-id 为播放器容器 ID，必须与 html 中一致
var player = TCPlayer('player-container-id', {
    sources: [{
      src: '请替换你的播放地址',
    }],
    licenseUrl: '请替换你的 licenseUrl', // license 地址，参考准备工作部分，在视立方控制台申
    language: '请替换你的设置语言', // 设置语言 en | zh-CN
});

// player.src(url); // url 播放地址
```

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



```
var player = TCPlayer('player-container-id', { // player-container-id 为播放器容器 ID
    fileID: '请传入你的 fileID', // 请传入需要播放的视频 fileID
    appID: '请传入你的 appID', // 请传入点播账号的 appID
    // 请传入播放器签名 psign，签名介绍和生成方式参见链接：https://cloud.tencent.com/docume
    psign:'请传入你的播放器签名 psign',
    licenseUrl: '请传入你的 licenseUrl', // 参考准备工作部分，在视立方控制台申请 license 后
    language: '请替换你的设置语言', // 设置语言 en | zh-CN
});
```

注意：

要播放的视频建议使用腾讯云转码，原始视频无法保证在浏览器中正常播放。

步骤4: 更多功能

播放器可以结合云点播的服务端能力实现高级功能，比如自动切换自适应码流、预览视频缩略图、添加视频打点信息等。这些功能在 播放长视频方案 中有详细的说明，可以参考文档实现。

此外，播放器还提供更多其他功能，功能列表和使用方法请参见 功能展示 页面。

TCPlayer 清晰度配置说明

最近更新时间：2024-04-11 16:49:44

在播放过程中，您可以通过自动或手动切换视频清晰度，以适应不同尺寸的播放设备和网络环境，从而提高观看体验。本文将从以下几个场景进行说明。

直播场景

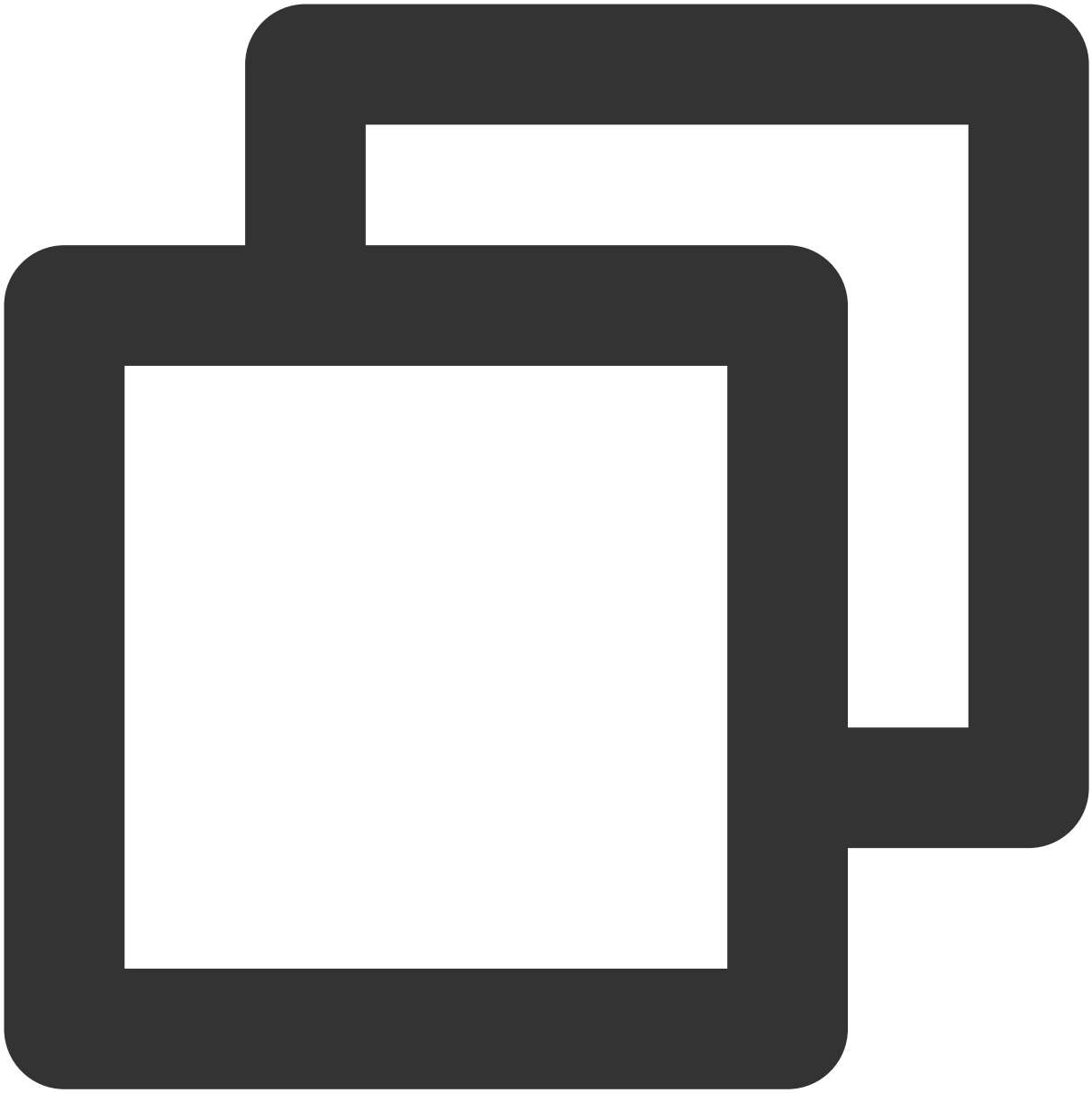
直播场景以 URL 的形式来播放视频，初始化播放器时，可以通过 sources 字段指定所要播放的 URL。或者在初始化播放器之后，调用播放器实例上的 src 方法进行播放。

1. 自适应码率（ABR）

自适应码率地址在切换时可以做到无缝衔接，切换过程不会出现中断或跳变，实现了观感和听感的平滑过渡。使用该技术也比较简单，仅需将播放地址传给播放器，播放器会自动解析子流，并将清晰度切换组件渲染到控制条上。

示例1: 播放 HLS 自适应码率地址

在初始化播放器时，传入自适应码率地址，播放器将自动生成清晰度切换组件，并会根据网络状况进行自动切换。



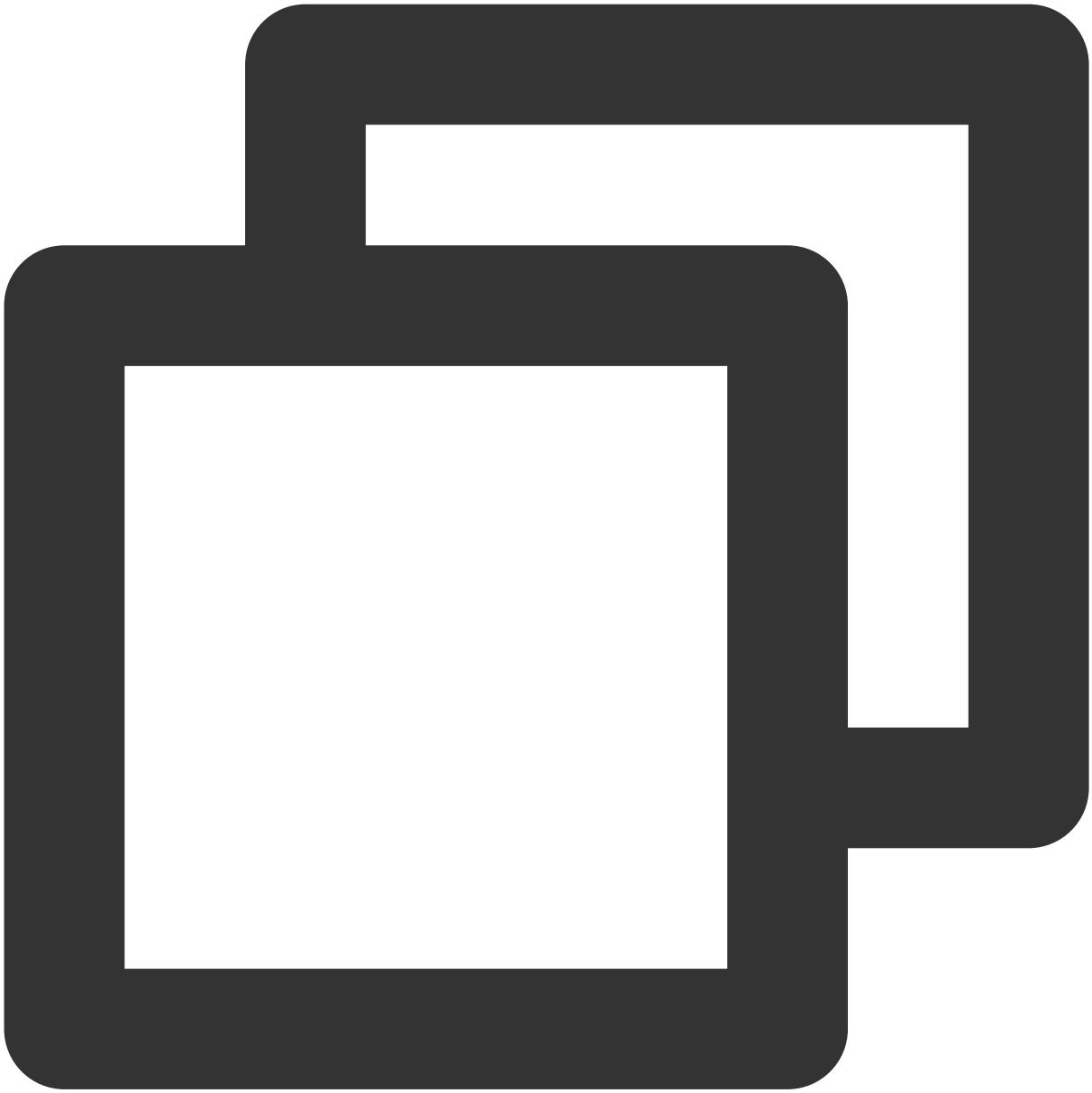
```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器I
  sources: [{
    src: 'https://hls-abr-url', // hls 自适应码率地址
  }],
});
```

注意：

解析 HLS 自适应码率的子流，需要依赖播放环境的 MSE API。在不支持 MSE 的浏览器环境（例如 iOS 的 Safari 浏览器），会由浏览器内部处理，根据网络情况自动切换清晰度，但无法解析出多种清晰度来供您手动切换。

示例2：播放 WebRTC 自适应码率地址

在 WebRTC 自适应码率场景，传入地址后，播放器会根据地址中的 ABR 模板自动拆解子流地址。



```
const player = TCPlayer('player-container-id',{
  sources: [{
    src: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b2
  }],

  webrtcConfig: {
    // 是否渲染多清晰度的开关，默认开启，可选
    enableAbr: true,
```

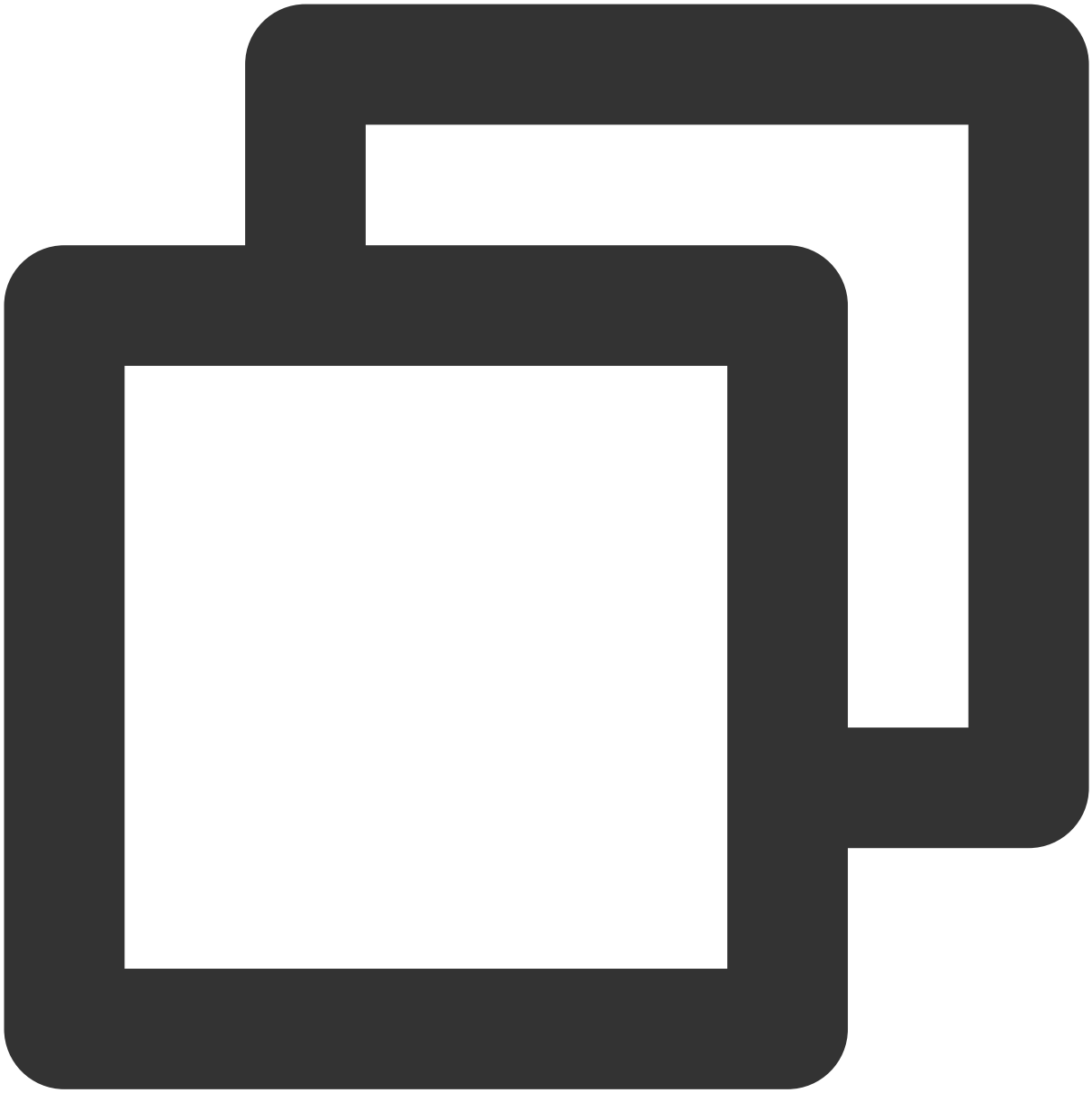
```
      // 模板名对应的label名，可选
      abrLabels: {
        d1080p: 'FHD',
        d540p: 'HD',
        d360p: 'SD',
        auto: 'AUTO',
      },
    },
  });
```

这里对 WebRTC 地址中的参数做以下说明：

1. tabr_bitrates 指定了 ABR 模板，有几个模板就会渲染出几个清晰度。如果没有单独设置清晰度的 label，模板名称（如 d1080p）将被设为清晰度名称。
2. tabr_start_bitrate 指定了起播的清晰度规格。
3. tabr_control 设置是否开启自动切换清晰度。开启后，播放器会单独渲染出自动清晰度选项。

2. 手动设置清晰度

如果播放地址不是自适应码率地址，也可以手动设置清晰度。参见如下代码：

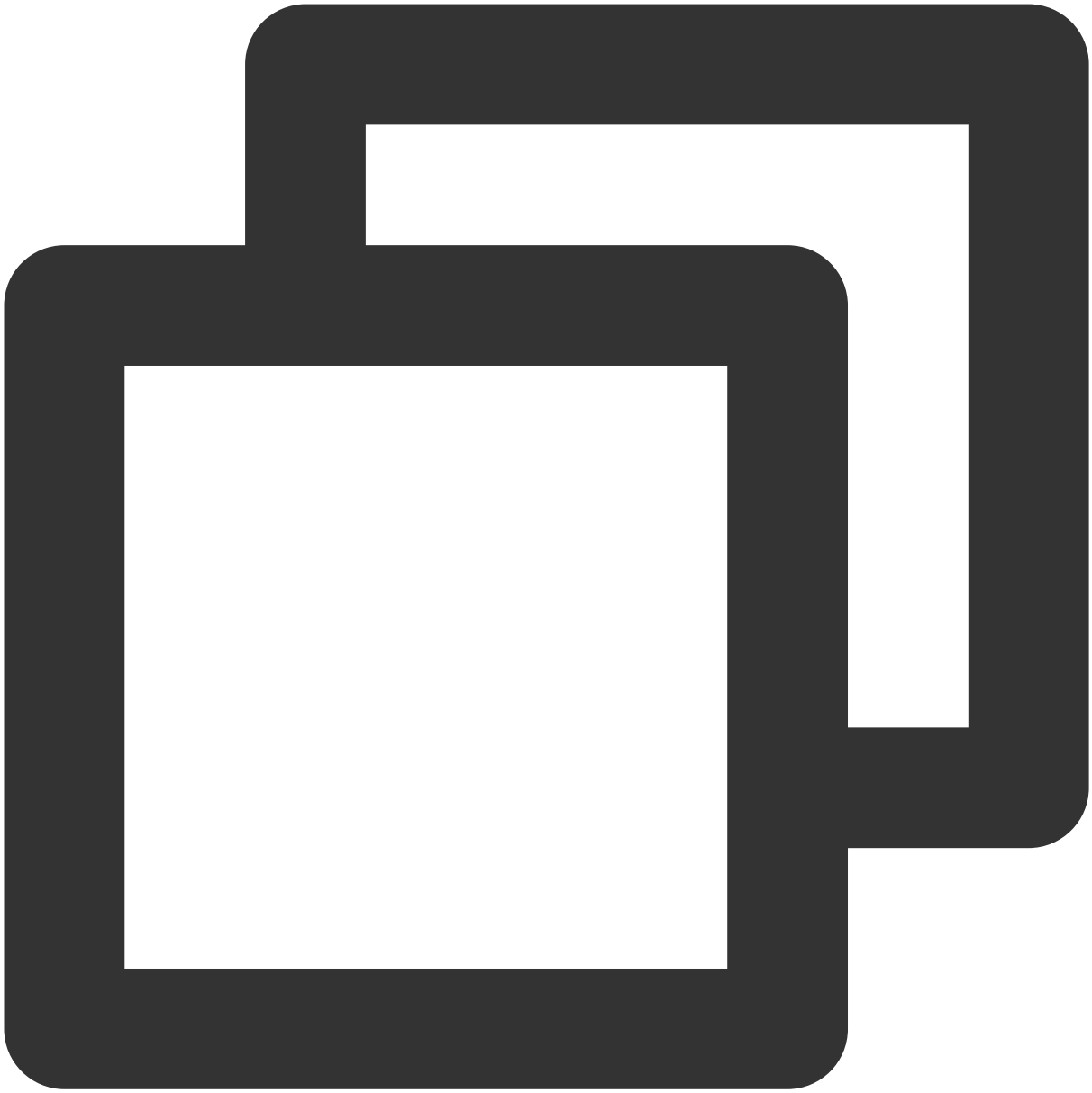


```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器I
  multiResolution:{
    // 配置多个清晰度地址
    sources:{
      'SD':[{
        src: 'http://video-sd-url',
      }],
      'HD':[{
        src: 'http://video-hd-url',
      }],
      'FHD':[{
```

```
          src: 'http://video-fhd-url',
        }]
      },
      // 配置每个清晰度标签
      labels:{
        'SD':'标清','HD':'高清','FHD':'超清'
      },
      // 配置各清晰度在播放器组件上的顺序
      showOrder:['SD','HD','FHD'],
      // 配置默认选中的清晰度
      defaultRes: 'SD',
    },
  });
```

点播场景

在点播场景下，如果通过 fileID 播放，播放哪种规格的文件（原始文件、转码文件、自适应码率文件）以及自适应码率文件子流的清晰度，都是在播放器签名中设置的。您可以参见指引 播放自适应码流视频，以便于您了解点播场景下播放视频的整个流程。
计算播放器签名时，可以通过 contentInfo 字段中的 resolutionNames 来设定不同分辨率的子流的展示名字。不填或者填空数组则使用默认配置。



```
resolutionNames: [{
  MinEdgeLength: 240,
  Name: '240P',
}, {
  MinEdgeLength: 480,
  Name: '480P',
}, {
  MinEdgeLength: 720,
  Name: '720P',
}, {
  MinEdgeLength: 1080,
```

```
    Name: '1080P',
  }, {
    MinEdgeLength: 1440,
    Name: '2K',
  }, {
    MinEdgeLength: 2160,
    Name: '4K',
  }, {
    MinEdgeLength: 4320,
    Name: '8K',
  }]
```

播放时的子流数量取决于转码时根据不同的自适应码率模板转换出的子流数。这些子流会依据短边长度落在由 resolutionNames 设定的哪个 MinEdgeLength 范围，再以对应的 Name 作为清晰度名称进行展示。

若您需要快速体验生成播放器签名，可以使用腾讯云点播控制台的 播放器签名生成工具。

TCPlayer 快直播降级说明

最近更新时间：2024-04-11 16:50:28

降级场景

快直播基于 WebRTC 实现，依赖于操作系统和浏览器对于 WebRTC 的支持。
目前，SDK 对以下操作系统和浏览器进行了测试，测试结果如下：

操作系统	操作系统版本	浏览器类型	浏览器版本	是否支持拉流
Windows	win 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		微信内嵌	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		微信内嵌	X5 内核	✓
		微信内嵌	XWeb 内核	✓

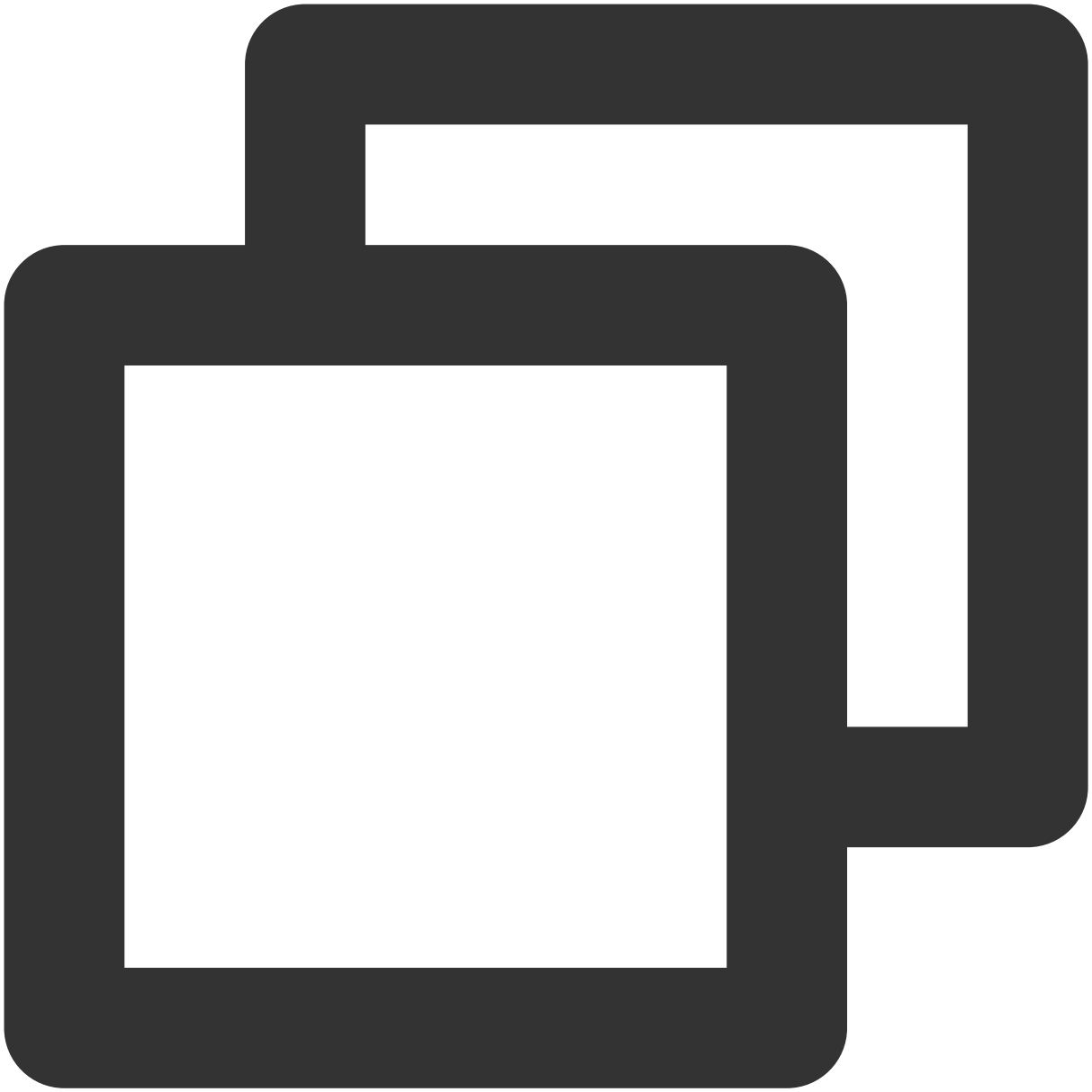
此外，在部分支持 WebRTC 的浏览器，也会出现解码失败或者服务端问题，这些情况下，播放器都会将 WebRTC 地址转换为兼容性较好的 HLS 地址来播放，这个行为称为降级处理。

总结一下，会触发降级的场景有以下几个：
浏览器环境不支持 WebRTC 。
连接服务器失败，并且连接重试次数已超过设定值 （内部状态码 -2004）。
播放过程解码失败（内部状态码 -2005）。
其他 WebRTC 相关错误（内部状态码 -2001）。

降级方式

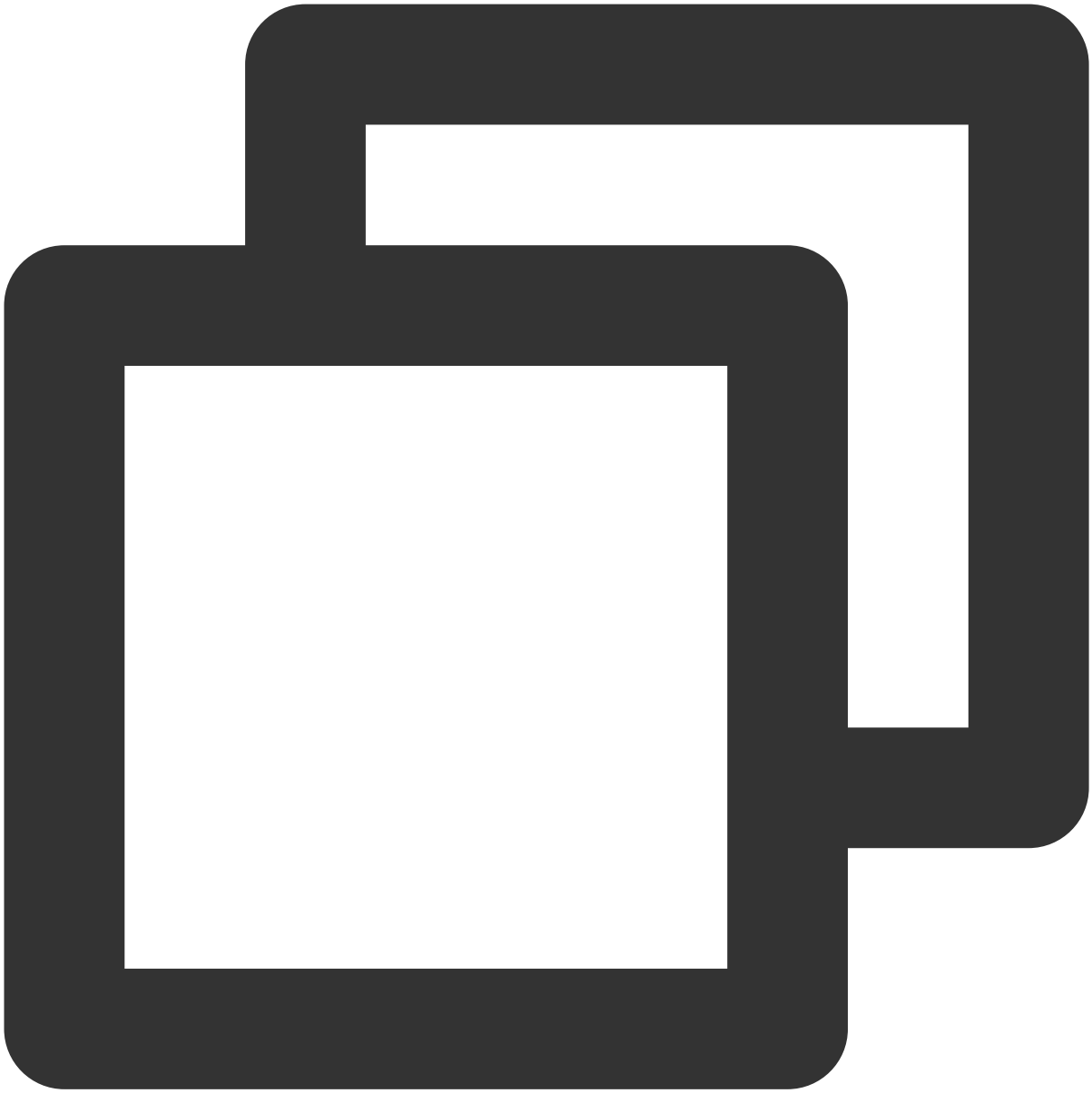
1. 自动降级

初始化播放器时，通过 sources 字段传入了快直播地址，在需要降级处理的环境，播放器自动会进行协议的转换，将快直播地址转换为 HLS 协议地址。
例如，快直播地址：



```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b284137ed10d
```

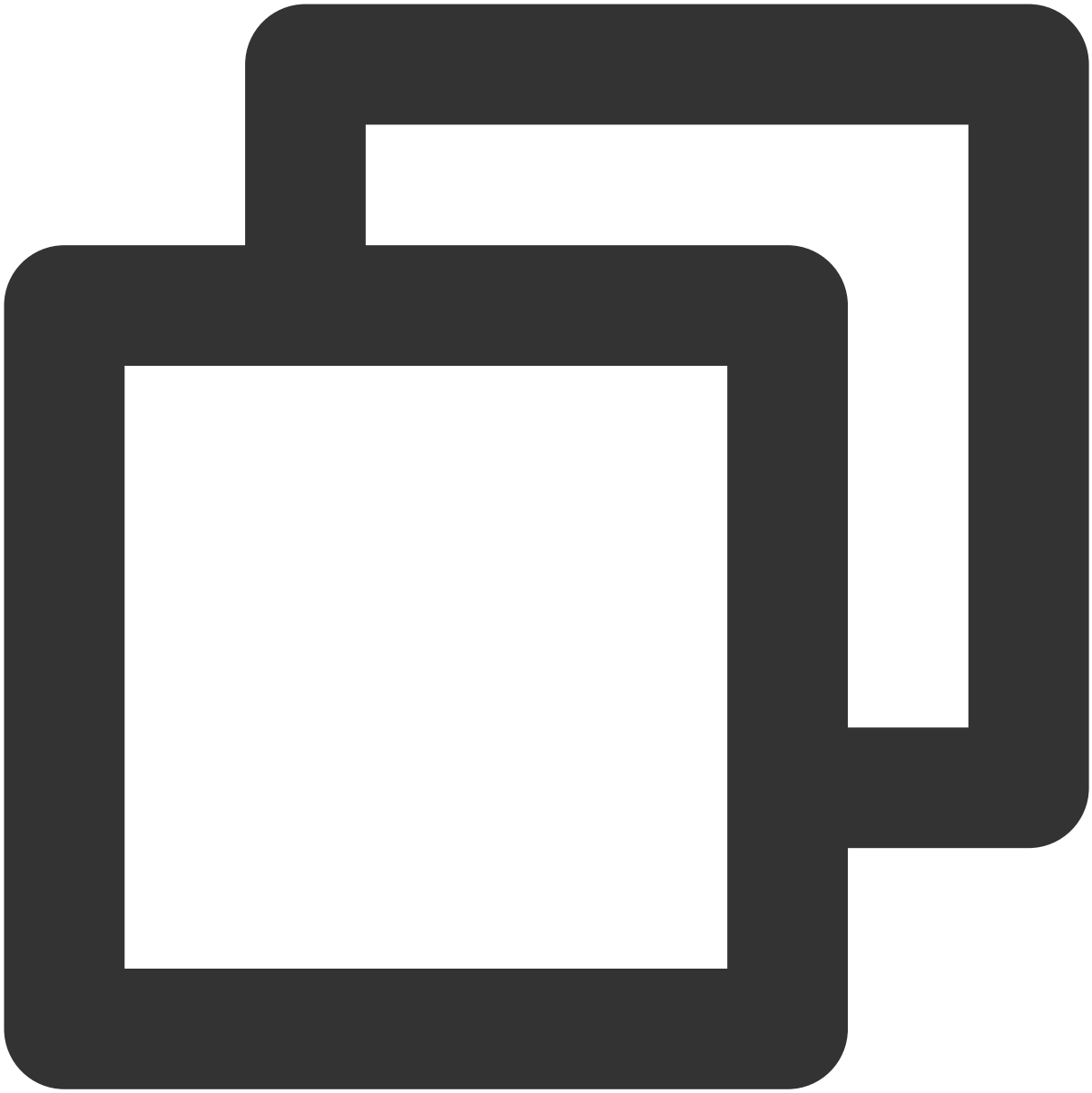
会自动转换为：



```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?txSecret=f22a813b284137e
```

2. 指定降级

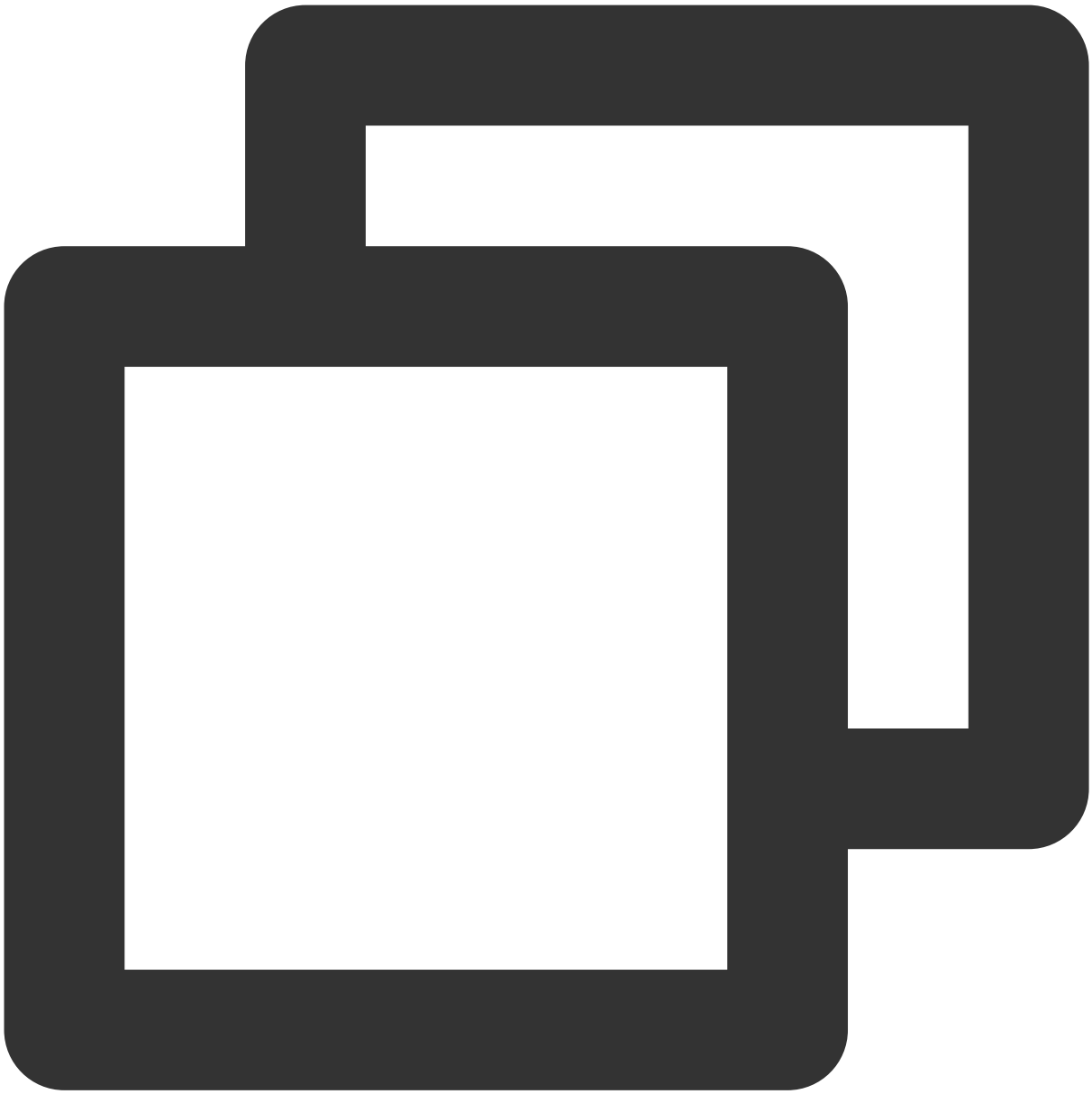
在播放自适应码率（ABR）场景，如果需要降级，并不能直接通过格式转换得到自适应码率的 HLS 地址，需要手动指定。又或者是在用户希望手动指定的其他场景，都可以通过如下方式指定降级地址，这里的地址并不局限于 HLS 协议，也可以是其他协议地址：



```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a8
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3
    },
});
```

降级回调

当触发降级时，播放器会触发回调：



```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

iOS 端集成
集成指引

最近更新时间：2024-04-26 11:09:31

本文主要介绍如何快速地将腾讯云播放器 LiteAVSDK_Player（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。

开发环境要求

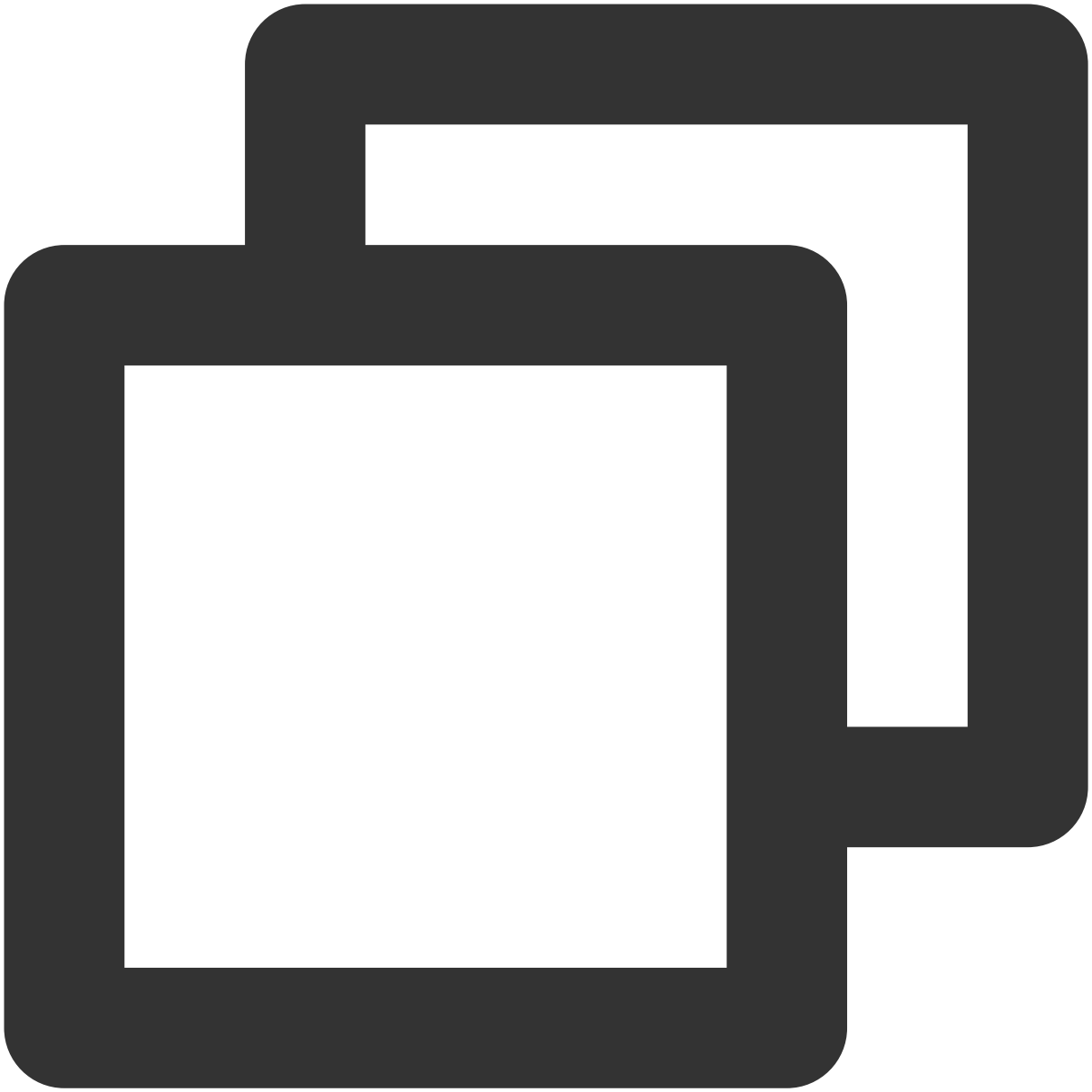
Xcode 9.0+。
iOS 9.0 以上的 iPhone 或者 iPad 真机。
项目已配置有效的开发者签名。

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集成

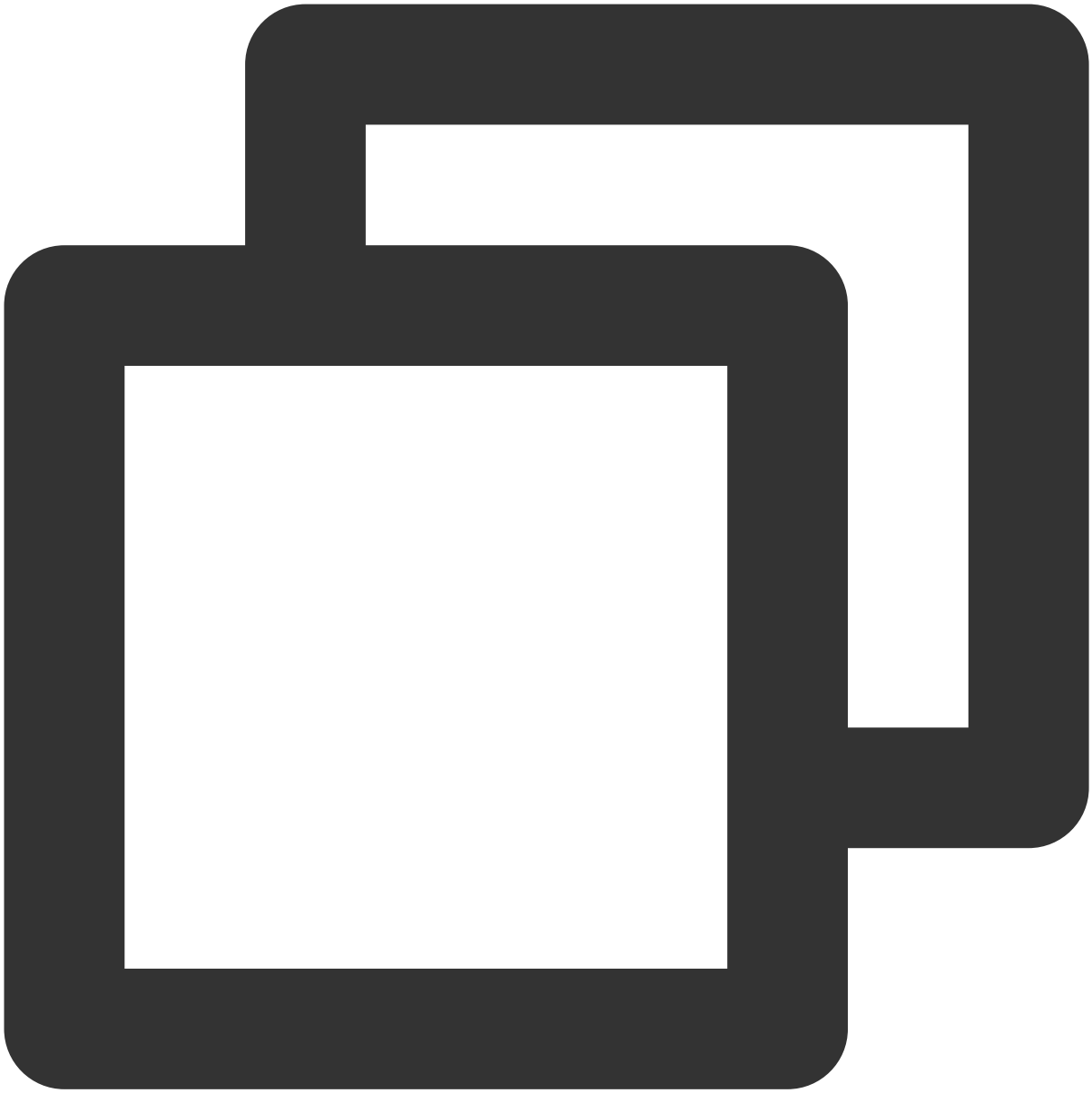
1. **安装 CocoaPods**
在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：



```
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。

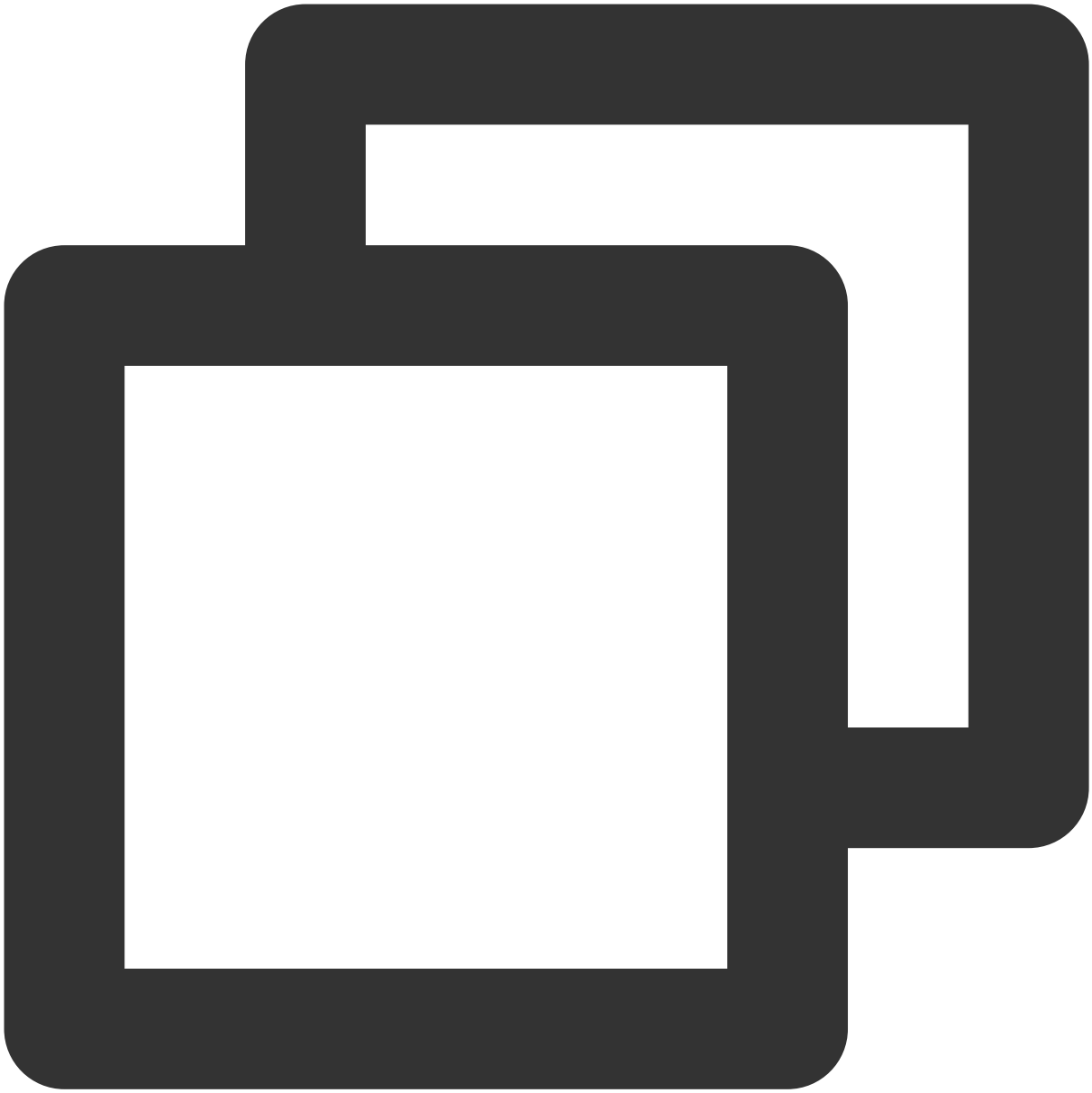


```
pod init
```

3. **编辑 Podfile 文件**

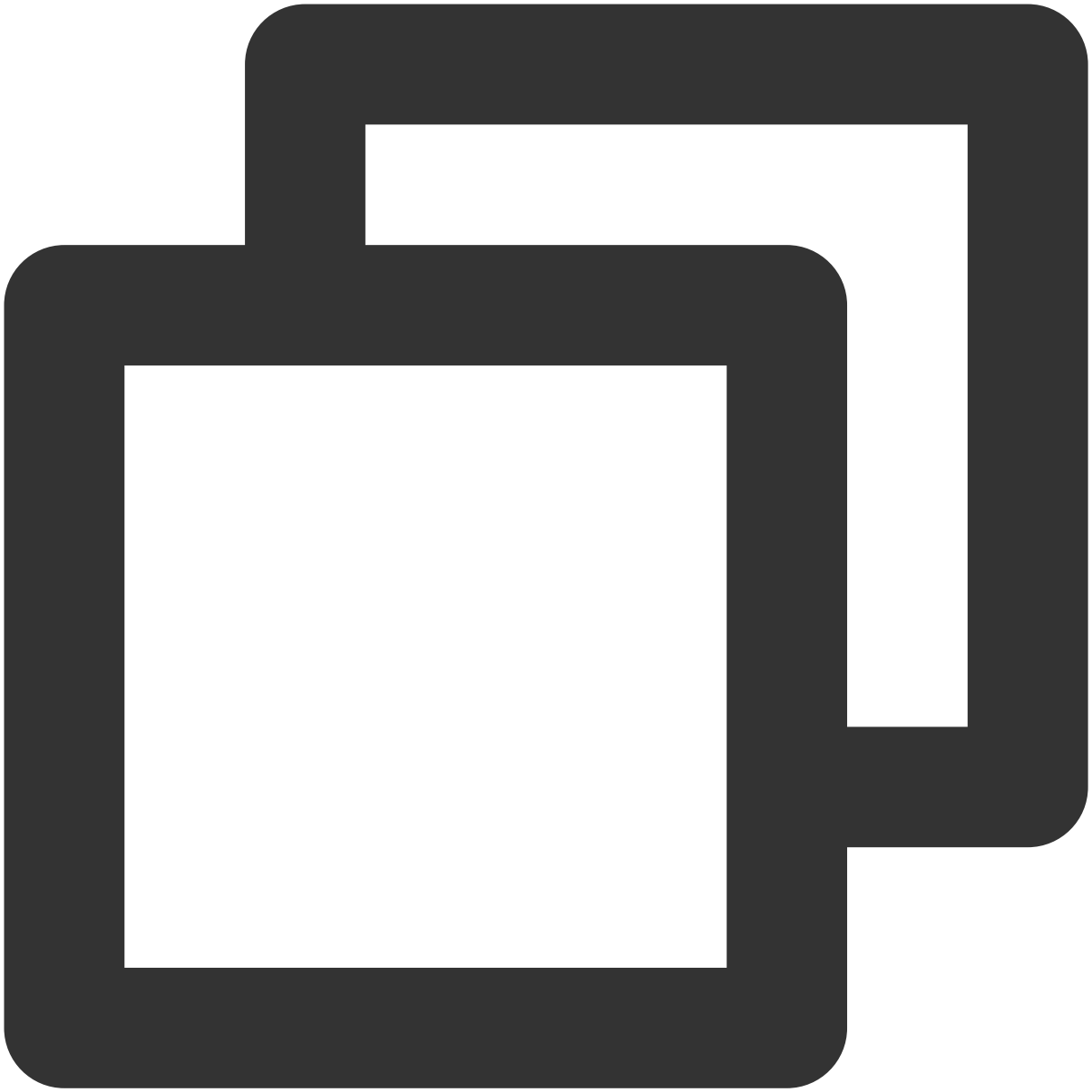
使用 CocoaPod 官方源，支持选择版本号。编辑 Podfile 文件：

Pod 方式直接集成最新版本 TXLiteAVSDK_Player_Premium：



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

target 'App' do
pod 'TXLiteAVSDK_Player_Premium'
end
```

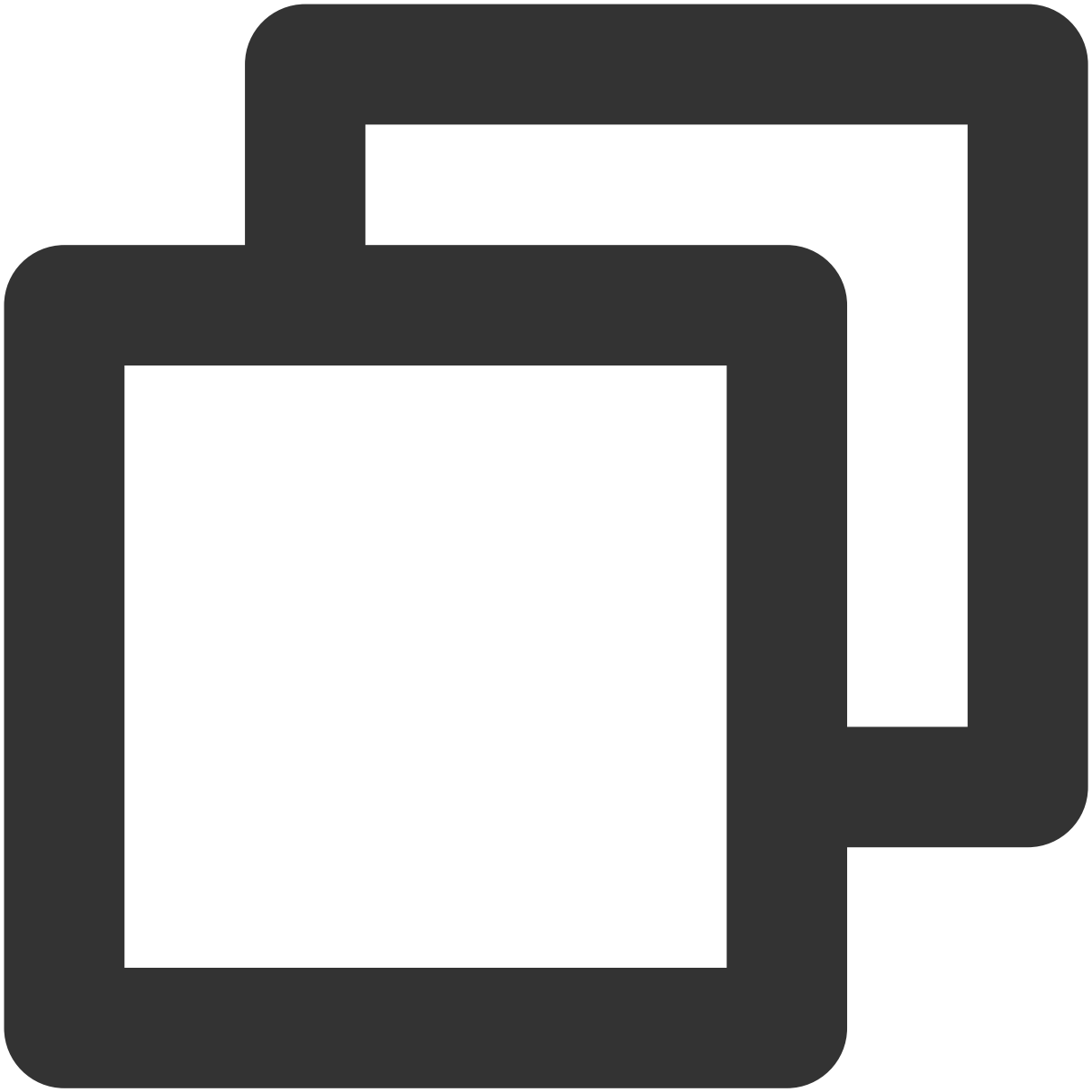
如果您需要指定某一个特定版本，可以在 podfile 文件中添加如下依赖：



```
pod 'TXLiteAVSDK_Player_Premium', '~> 110.8.29000'
```

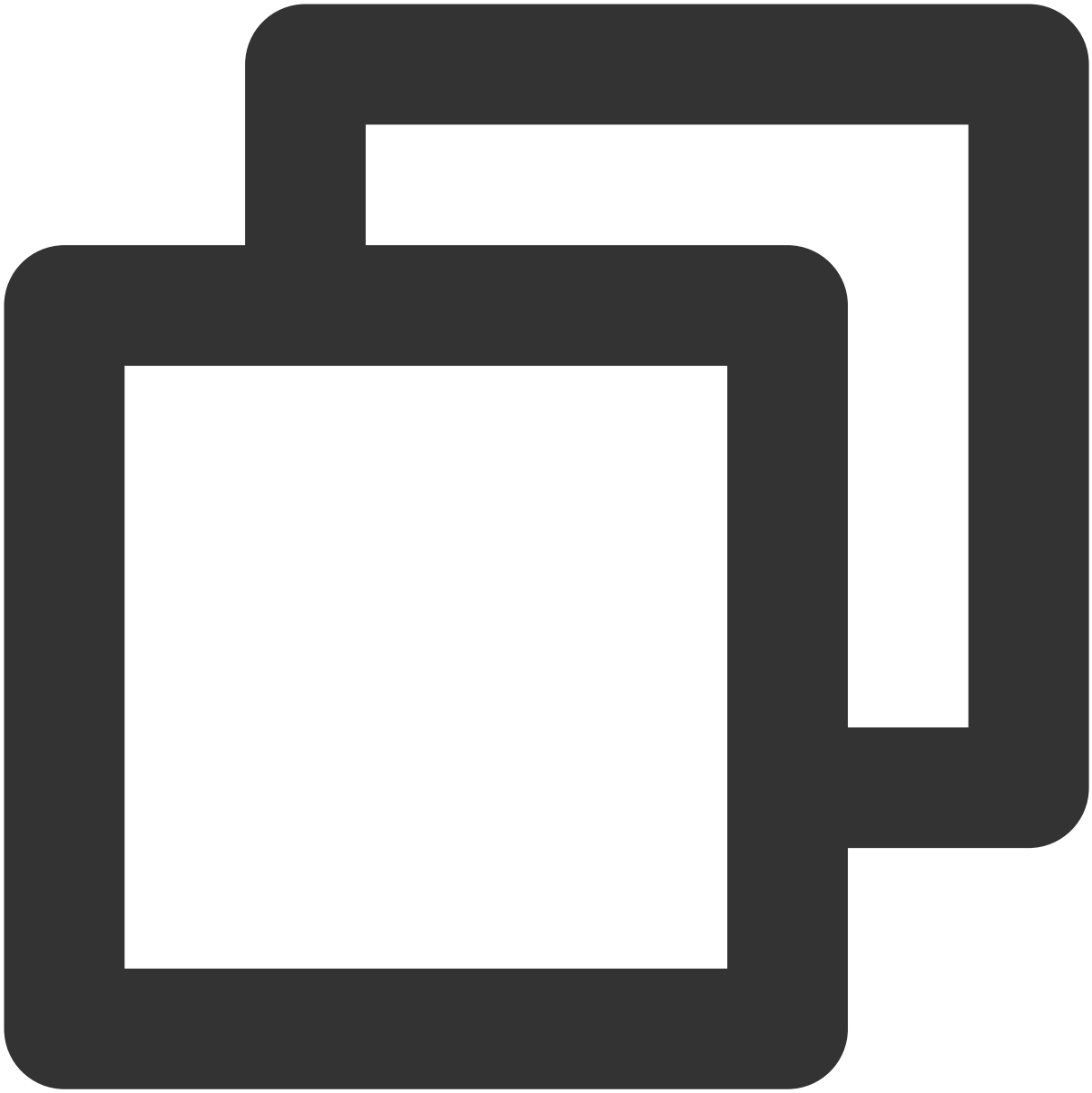
4. 更新并安装 SDK

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：

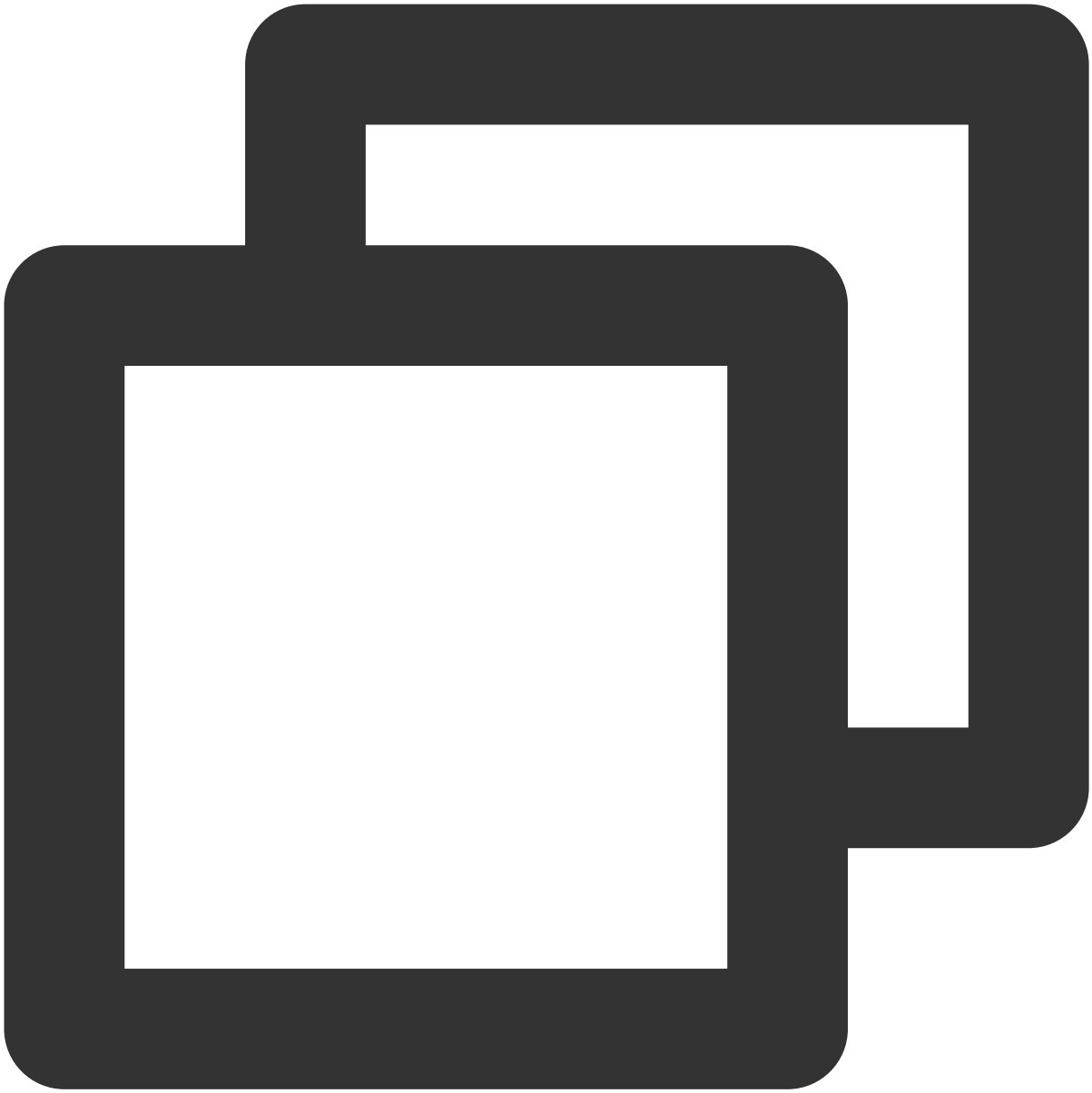


```
pod update
```

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SDK

1. 下载 最新版本 TXLiteAVSDK_Player_Premium 的 SDK + Demo 开发包。
2. 将 SDK/TXLiteAVSDK_Player_Premium.framework 添加到待集成的工程中，并勾选 `Do Not Embed` 。
3. 需要配置项目 Target 的 -ObjC，否则会因为加载不到 SDK 的类别而导致 Crash。

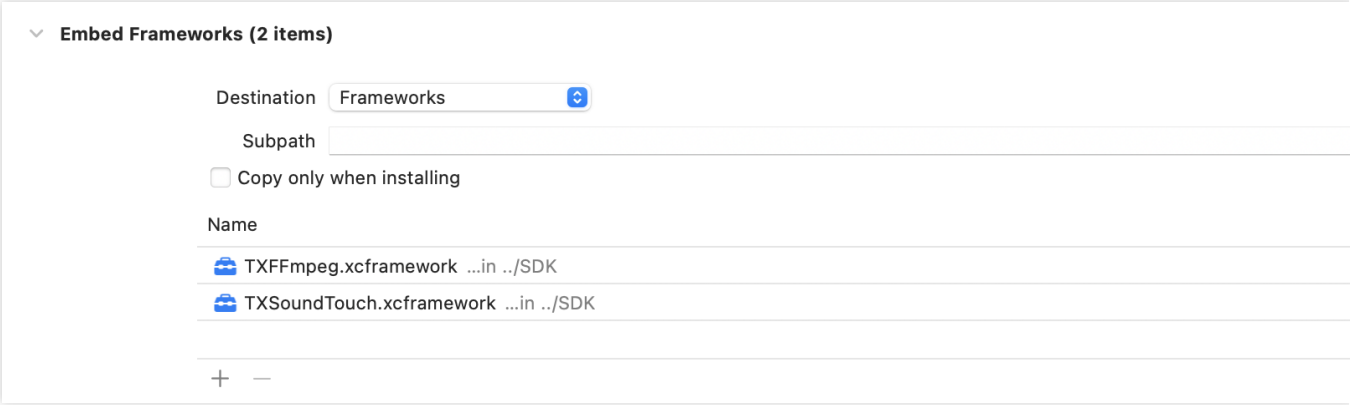


```
打开 Xcode -> 选择对应的 Target -> 选择"Build Setting" Tab -> 搜索"Other Link Flag" ->
```

4. 添加相应的库文件（SDK 目录里）

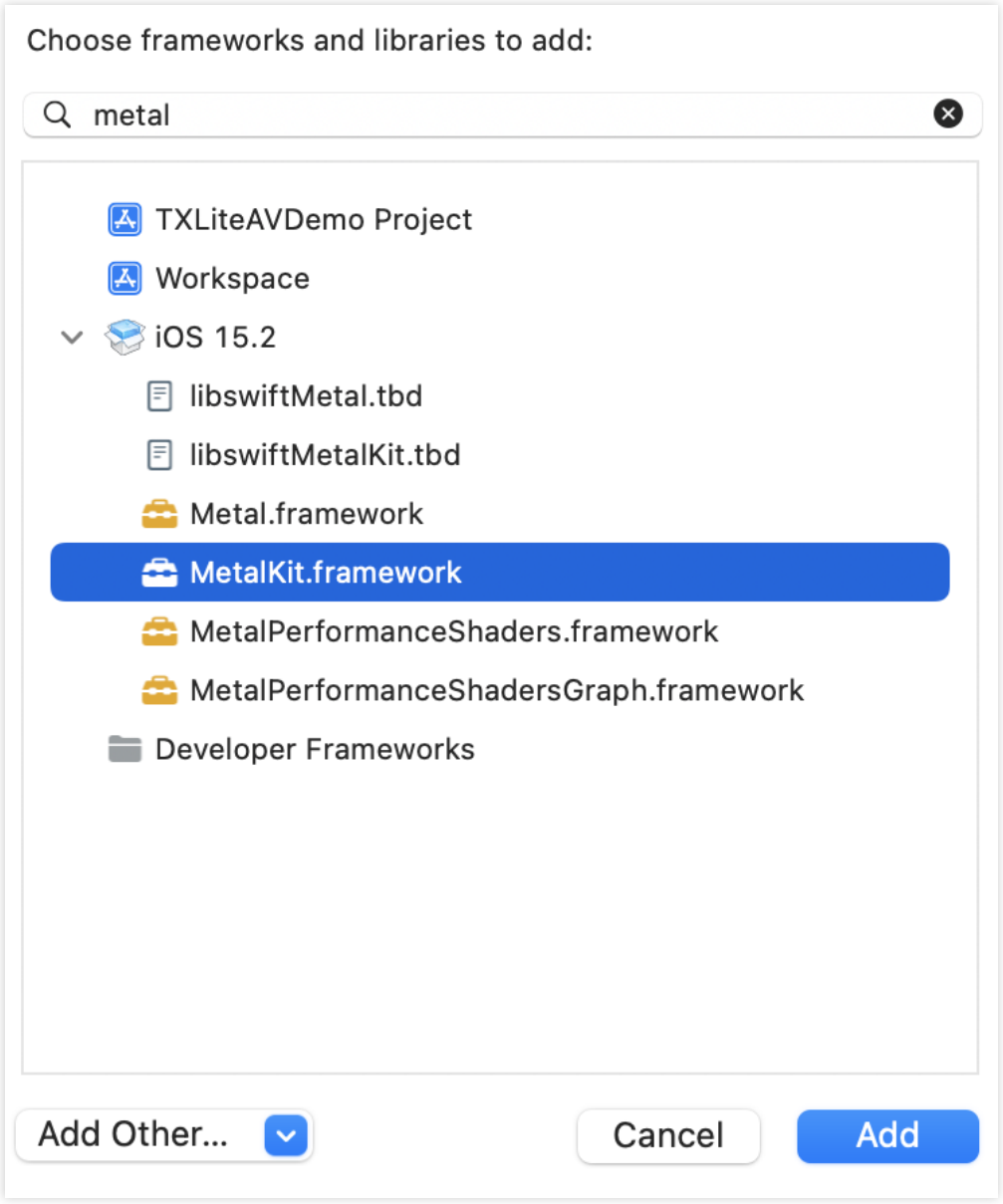
TXFFmpeg.xcframework：将.xcframework 文件添加到项目工程中，并在“General - Frameworks, Libraries, and Embedded Content”中将其设置为“Embed&Sign”，并在“Project Setting - Build Phases - Embed Frameworks”中进行检查，设置“Code Sign On Copy”选项为勾选状态，如下图所示：

TXSoundTouch.xcframework：将.xcframework 文件添加到项目工程中，并在“General - Frameworks, Libraries, and Embedded Content”中将其设置为“Embed&Sign”，并在“Project Setting - Build Phases - Embed Frameworks”中进行检查，设置“Code Sign On Copy”选项为勾选状态，如下图所示：

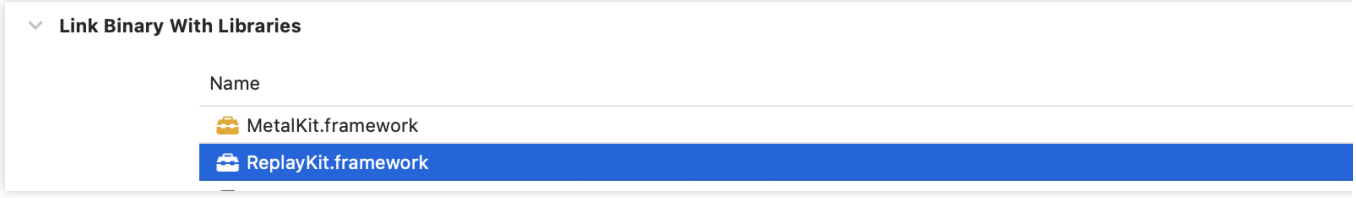


同时，切换到 Xcode 的 “Build Settings - Search Paths”，在“Framework Search Paths”中添加上述 Framework 所在的路径。

MetalKit.framework：打开 Xcode，切换到“project setting - Build Phases - Link Binary With Libraries”，选择左下角的“+”号，并输入“MetalKit”，并加入项目工程中，如下图所示：



ReplayKit.framework：打开 Xcode，切换到“project setting - Build Phases - Link Binary With Libraries”，选择左下角的“+”号，并输入“ReplayKit”，并加入项目工程中，如下图所示：



使用同样的方式添加如下系统库：

系统 Framework 库：

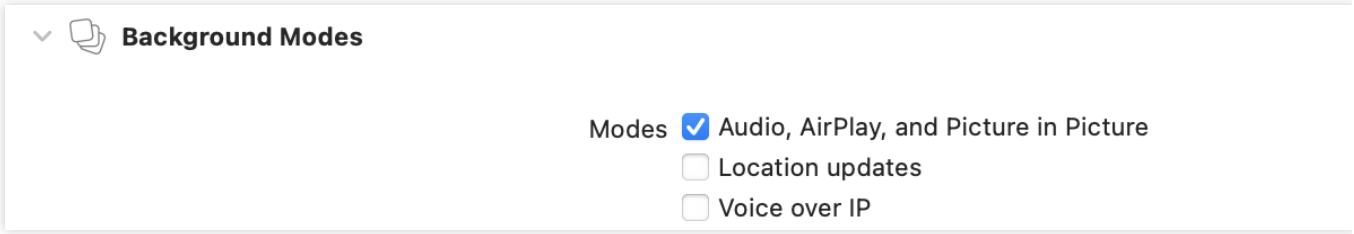
SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics, AVFoundation, Accelerate, MobileCoreServices。

系统 Library 库: libz, libresolv，libiconv, libc++, libsqlite3。

画中画功能

如果需要使用画中画能力，请按如下图的方式进行配置，若无此部分需求可以忽略。

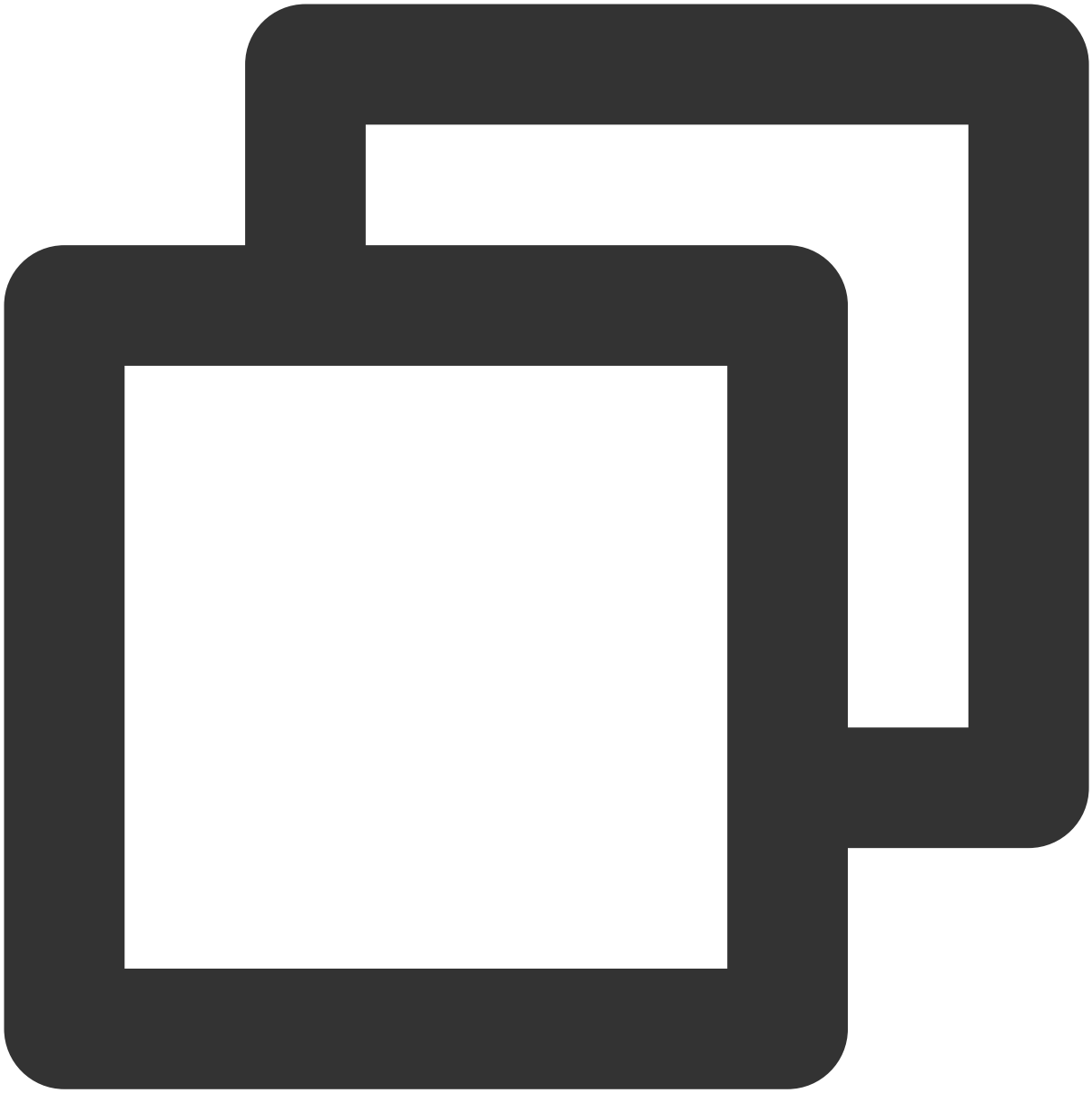
1. 为了使用 iOS 的画中画（Picture-In-Picture），请将 SDK 升级到10.3版本及以上。
2. 使用画中画能力时，需要开通后台模式。XCode 选择对应的Target -> Signing & Capabilities -> Background Modes，勾选“Audio, AirPlay, and Picture in Picture”，如图所示：



在工程中引入 SDK

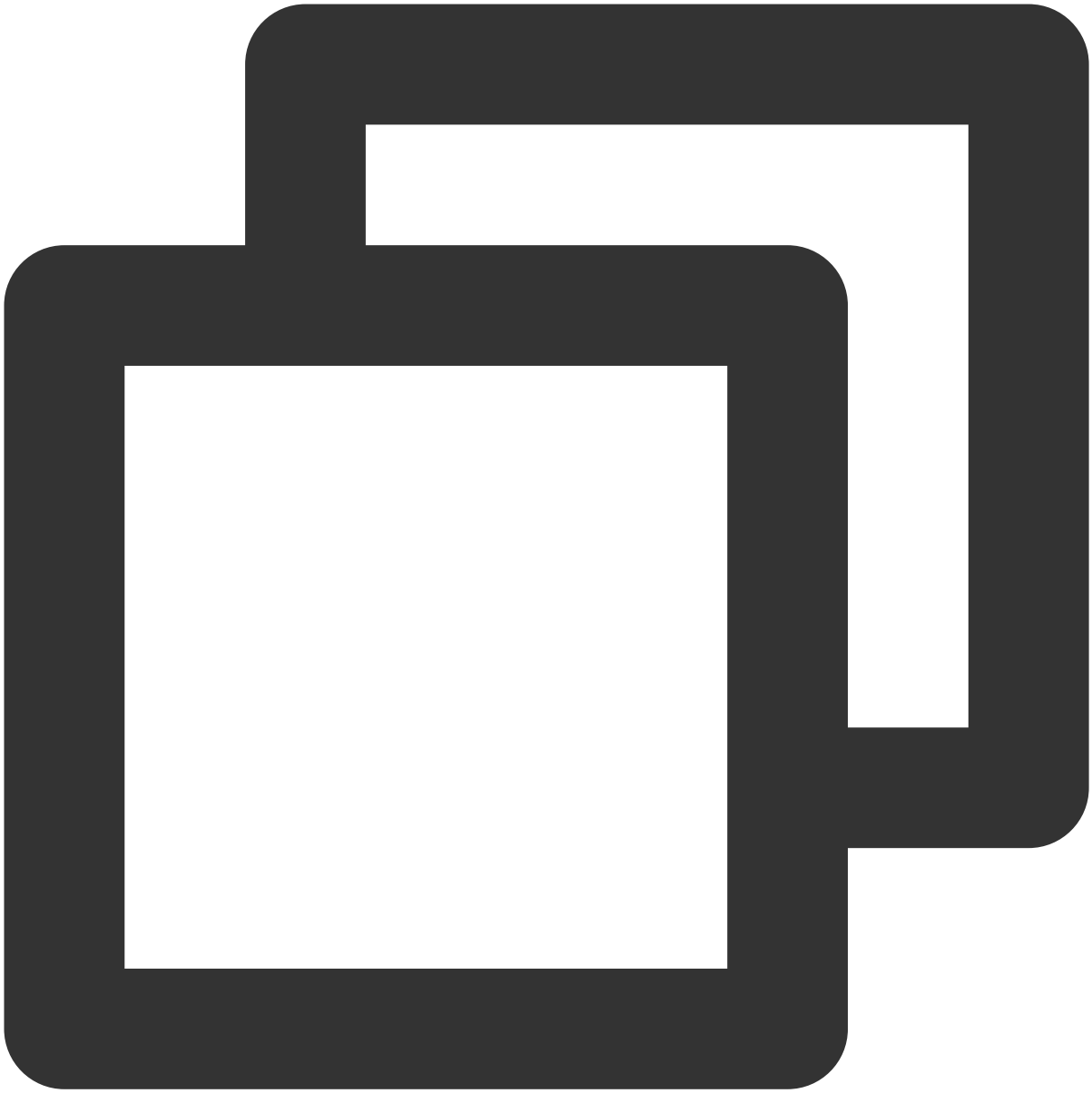
项目代码中使用 SDK 有两种方式：

方式一： 在项目需要使用 SDK API 的文件里，添加模块引用。



```
@import TXLiteAVSDK_Player_Premium;
// 如果您使用的是播放器基础版本，请用： @import TXLiteAVSDK_Player;
```

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_Player_Premium/TXLiteAVSDK.h"
// 如果您使用的是播放器基础版本，请用：#import "TXLiteAVSDK_Player/TXLiteAVSDK.h"
```

给 SDK 配置 License 授权

1. 进入 云点播控制台 获取测试用 License，不配置 License 将会播放时视频失败，具体操作请参见 新增与续期 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. 获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

常见问题

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。

Swift 项目工程里怎么调用 SDK 的 API 方法？

如果在 Swift 的项目工程里想调用 SDK 的 API 接口，有下面两种方式：

方式一：使用桥接头文件

1. 创建桥接头文件。例如***-Bridging-Header.h，并添加如下代码#import <TXLiteAVSDK_Player_Premium/TXLiteAVSDK.h>。
2. 配置工程BuildSetting的Objective-c Bridging header选项。设置桥接文件的路径并添加到 Objective-c Bridging header中（如：$(SRCROOT)/SwiftCallOC/***-Bridging-Header.h，根据项目具体路径确定），编译运行即可。

方式二：使用 SDK 内的 module.modulemap 文件

1. 检查TXLiteAVSDK_Player_Premium.framework里是否有包含Modules - module.modulemap文件（Player SDK 默认都提供）。
2. 配置工程BuildSetting的Swift Compiler - Search Paths选项。添加module.modulemap文件所在的目录路径或其上层目录路径，此处可为：${PODS_ROOT}/TXLiteAVSDK_Player_Premium/TXLiteAVSDK_Player_Premium/TXLiteAVSDK_Player_Premium.framework/Modules（根据项目具体路径确定）。
3. 在需要调用的类顶部，使用import TXLiteAVSDK_Player_Premium来进行引入并调用相关的方法。

以上集成的方式及 Demo，可以具体参考 GitHub Demo。

点播场景

最近更新时间：2024-08-07 15:15:40

准备工作

1. 开通 云点播 相关服务，未注册用户可注册账号 试用。
2. 下载 Xcode，如您已下载可略过该步骤，您可以进入 App Store 下载安装。
3. 下载 Cocoapods，如您已下载可略过该步骤，您可以进入 Cocoapods官网 按照指引进行安装。

通过本文你可以学会

如何集成腾讯云 iOS 播放器 SDK
如何使用播放器 SDK 进行点播播放
如何使用播放器 SDK 底层能力实现更多功能

SDK 集成

步骤1：集成 SDK 开发包

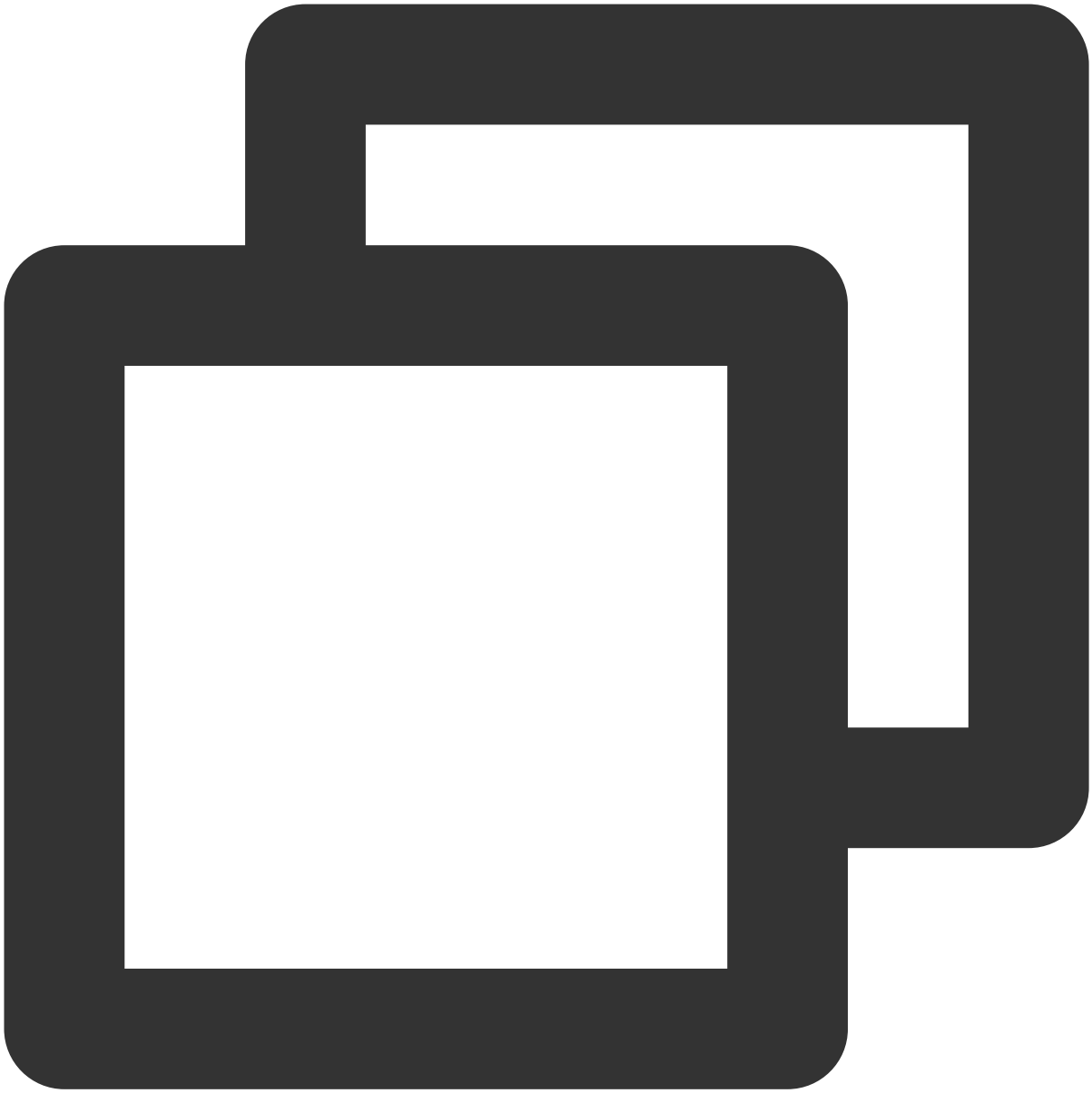
下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引。

步骤2：配置 License 授权

若您已获得相关 License 授权，需在 云点播控制台 获取 License URL 和 License Key。
若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。
获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

步骤3：创建 Player

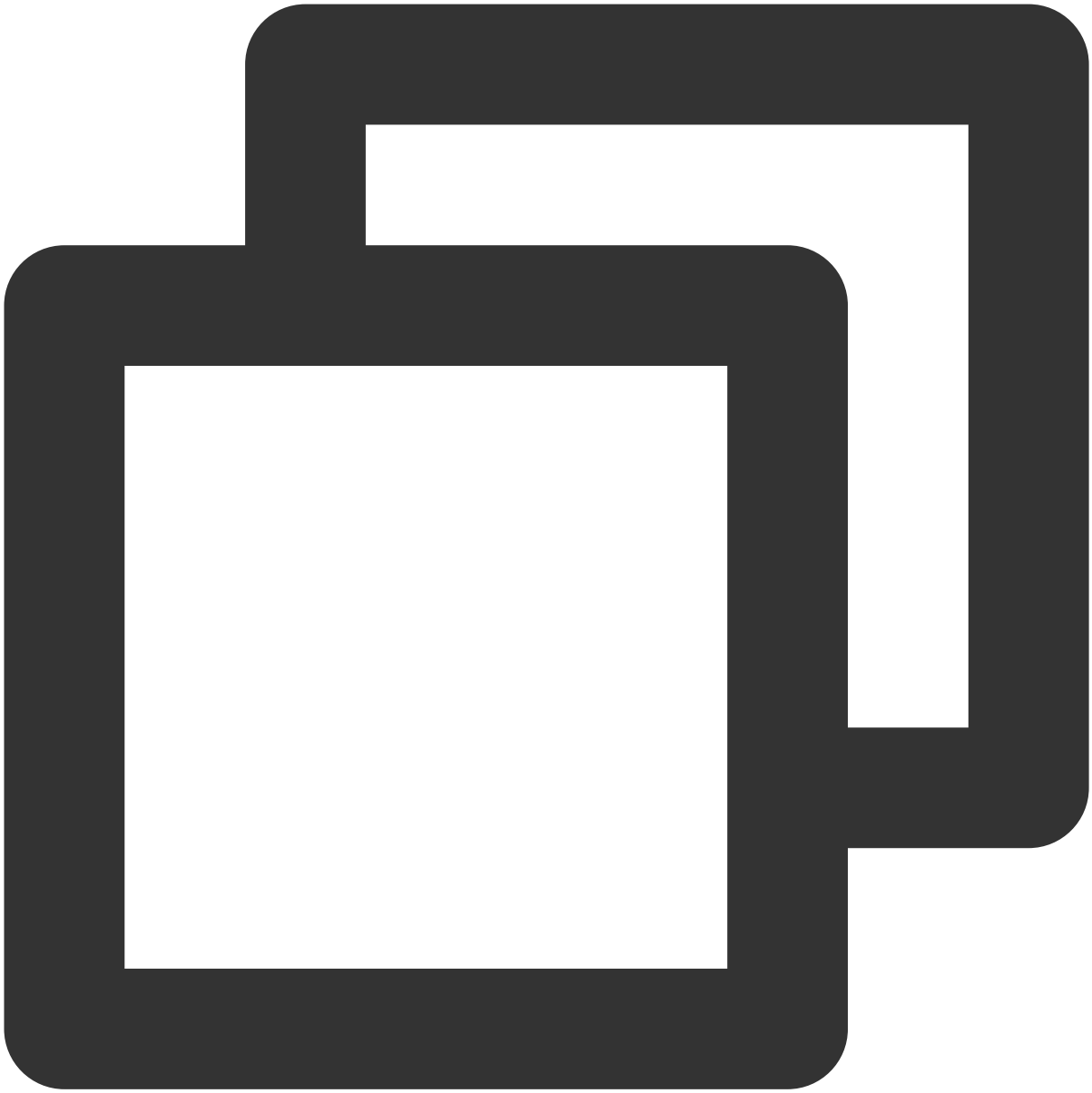
视频云 SDK 中的 TXVodPlayer 模块负责实现点播播放功能。



```
TXVodPlayer *_txVodPlayer = [[TXVodPlayer alloc] init];
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

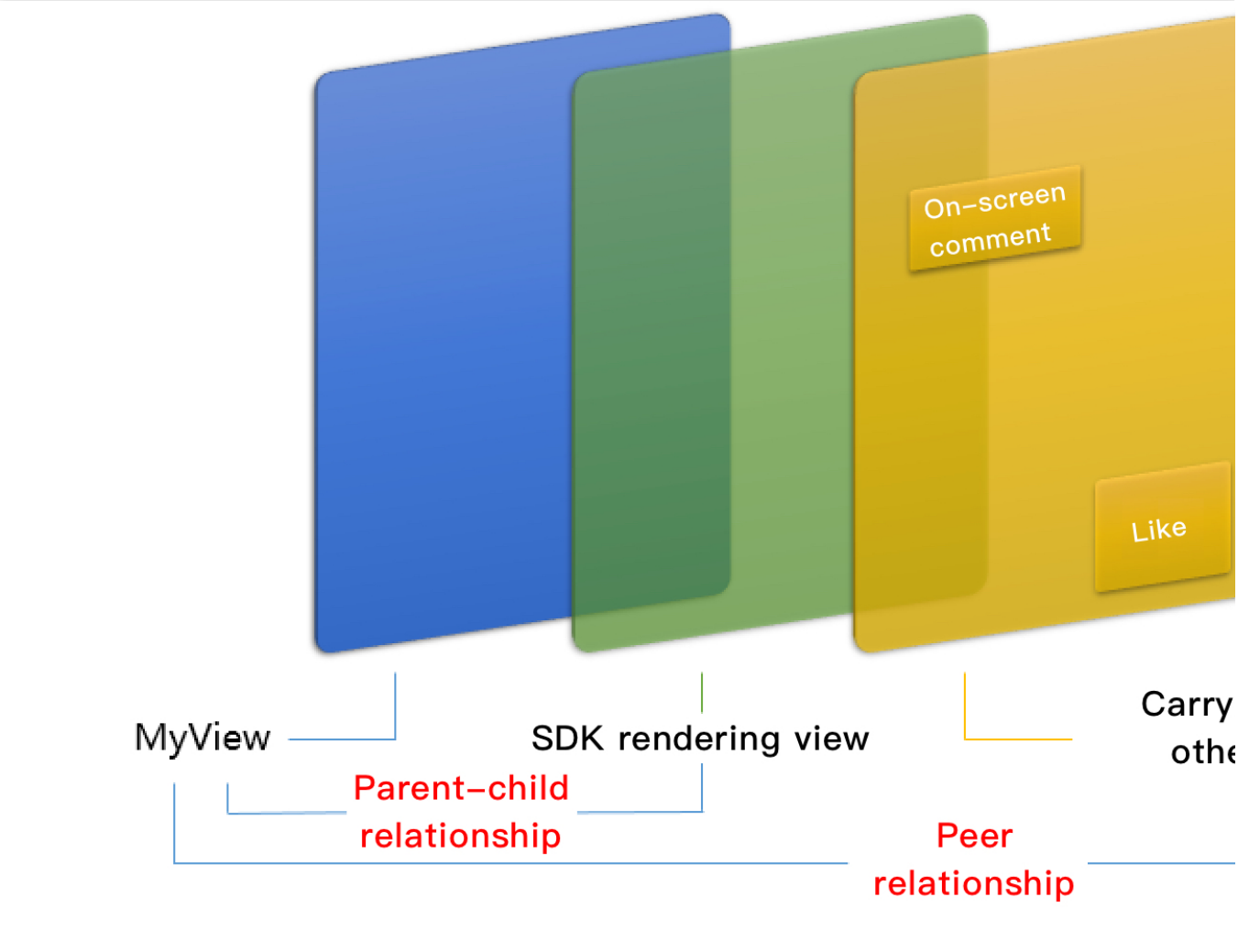
步骤4：渲染 View

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。



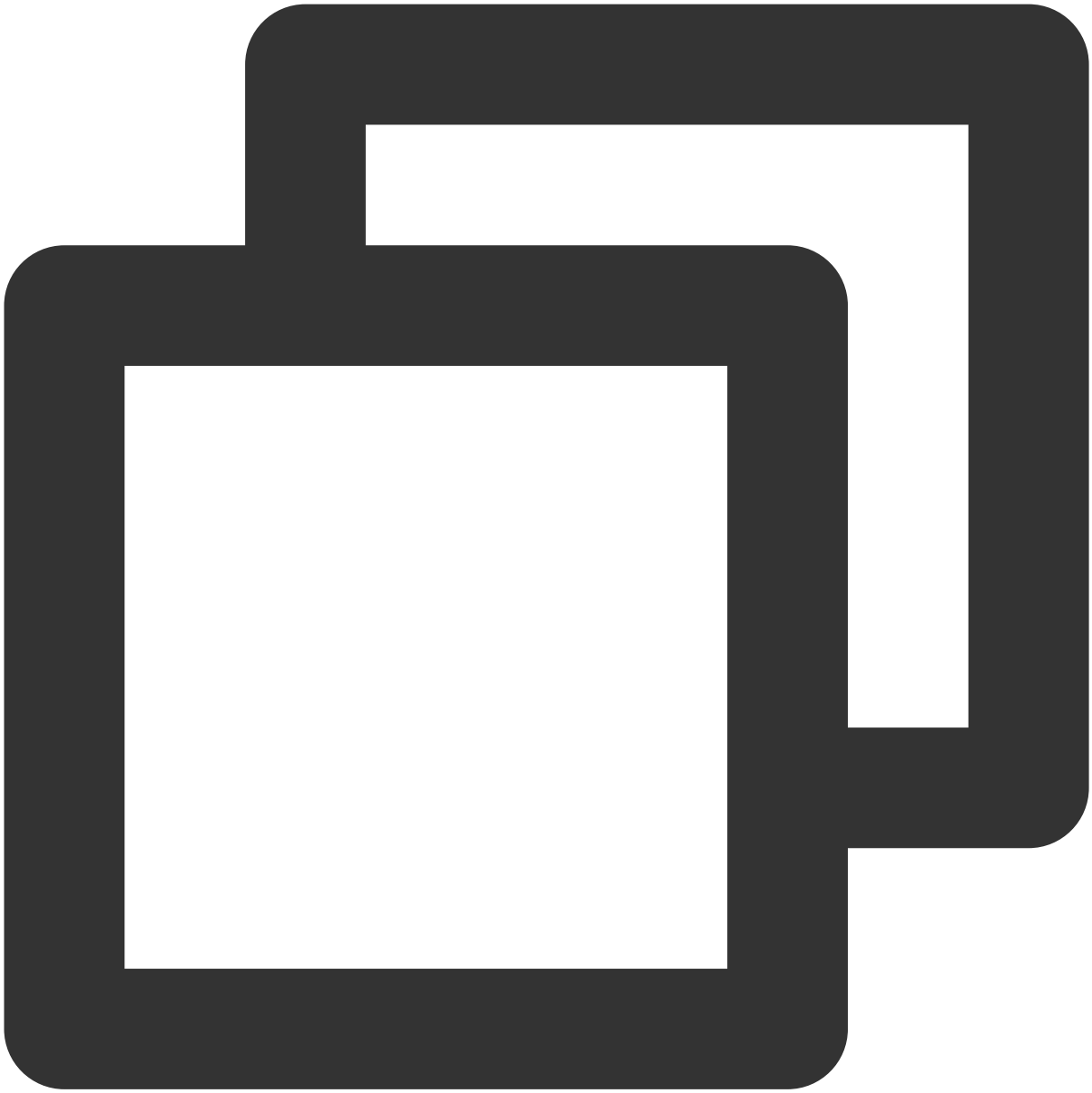
```
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

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



如何做动画

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



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

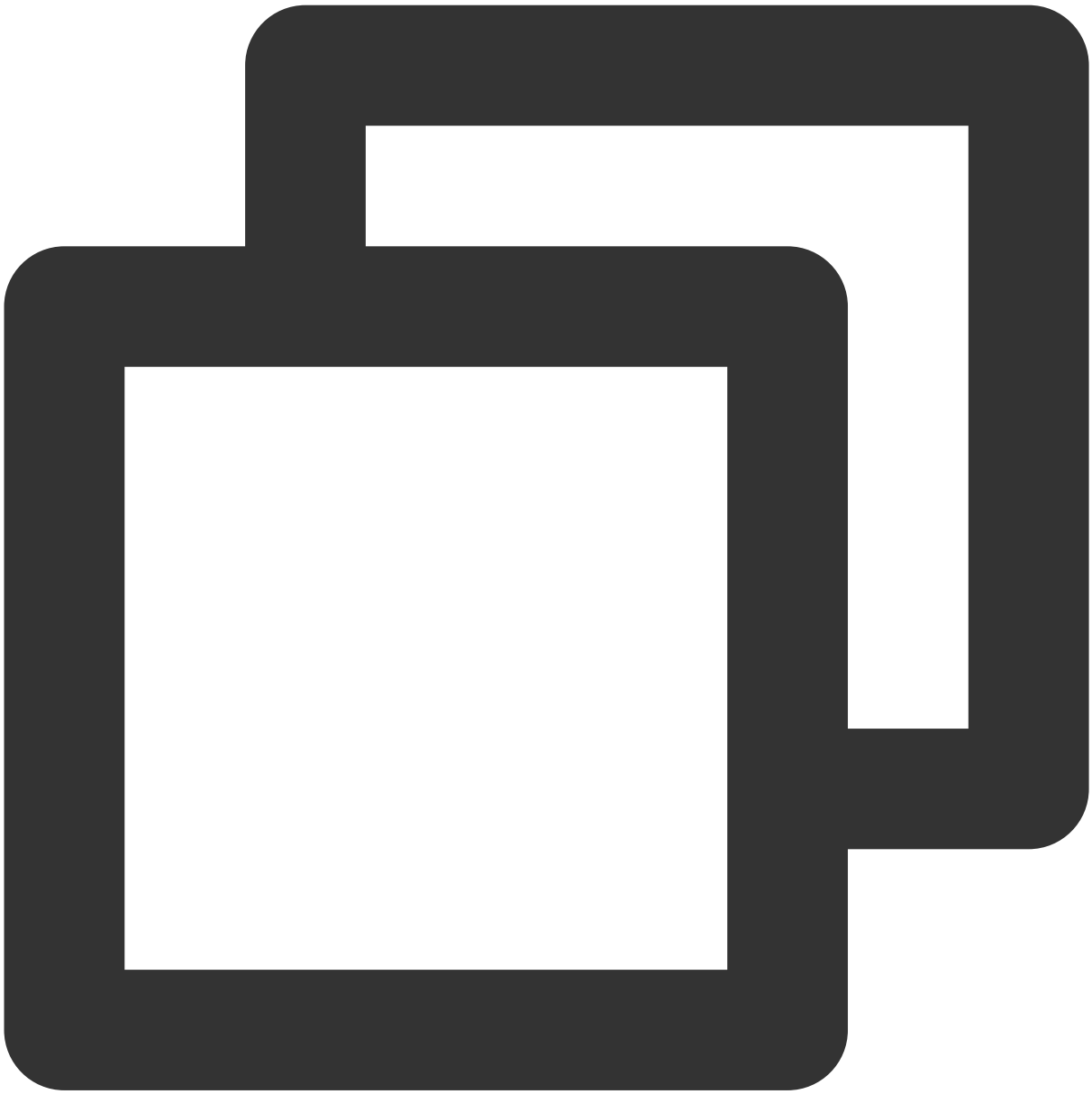
步骤5：启动播放

TXVodPlayer 支持两种播放模式，您可以根据需要自行选择：

通过 URL 方式

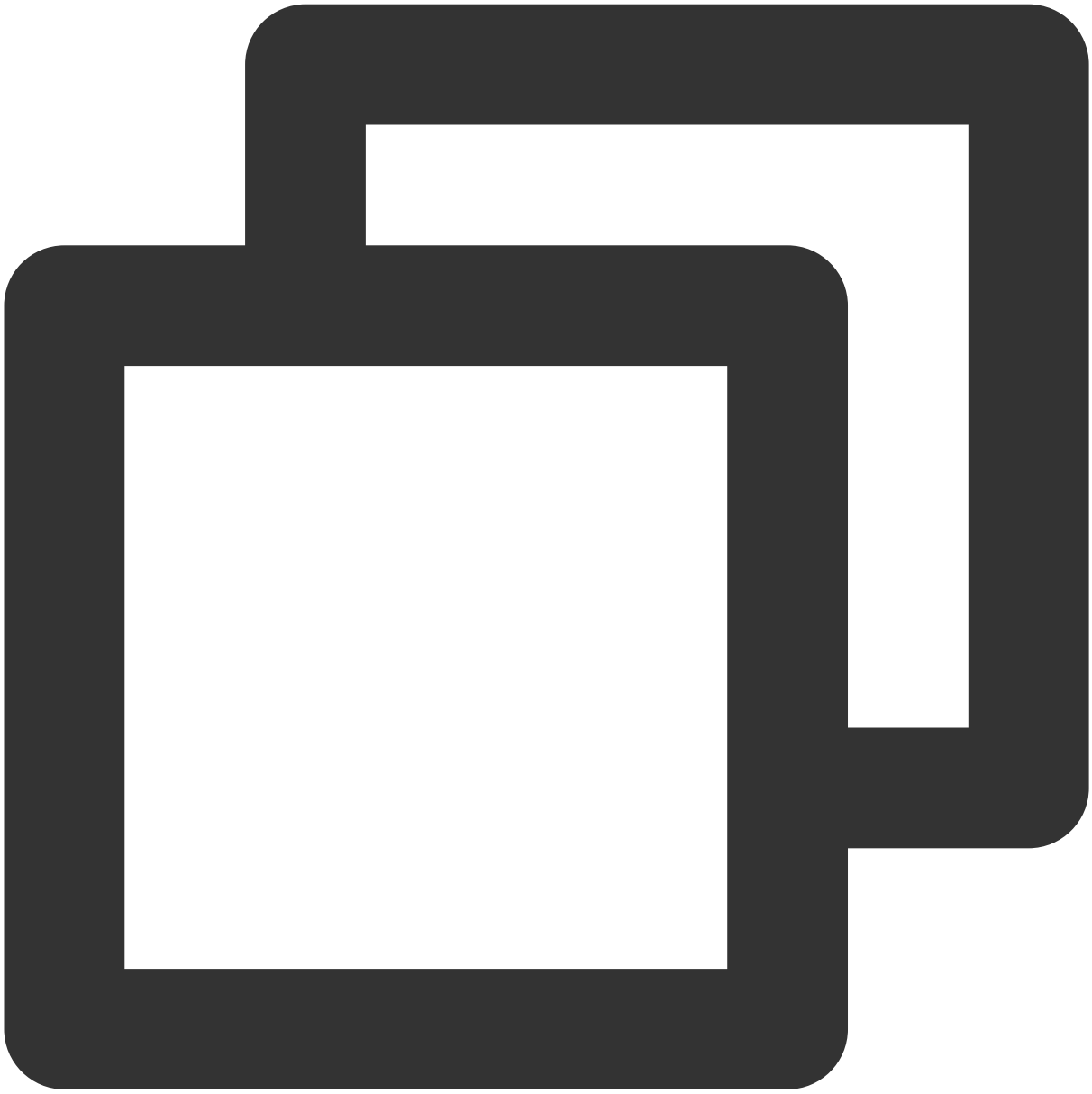
通过 fileId 方式

TXVodPlayer 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startPlay 函数即可。



```
// 播放 URL 视频资源
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startVodPlay:url];

// 播放沙盒本地视频资源
// 获取 Documents路径
NSString *documentPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
// 获取本地视频路径
NSString *videoPath = [NSString stringWithFormat:@"%@/video1.m3u8",documentPath];
```

```
[_txVodPlayer startVodPlay:videoPath];
```

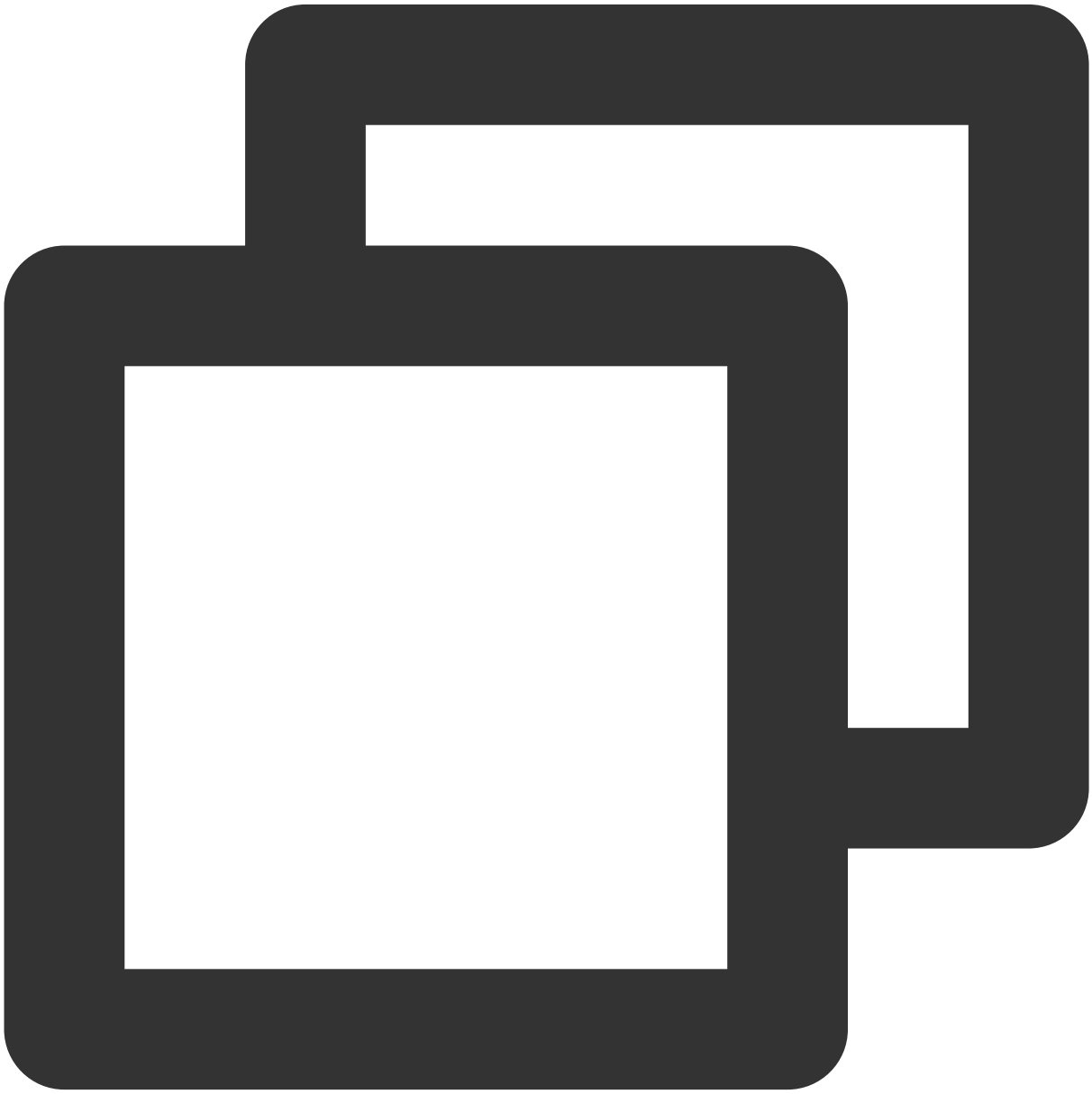


```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788;
p.fileId = @"4564972819220421305";
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
p.sign = @"psignxxxxx"; // 播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

在 媒资管理 找到对应的文件。点开后在右侧视频详情中，可以看到 fileId。
通过 fileId 方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或 fileId 不存在，则会收到 `PLAY_ERR_GET_PLAYINFO_FAIL` 事件，反之收到 `PLAY_EVT_GET_PLAYINFO_SUCC` 表示请求成功。

步骤6：结束播放

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



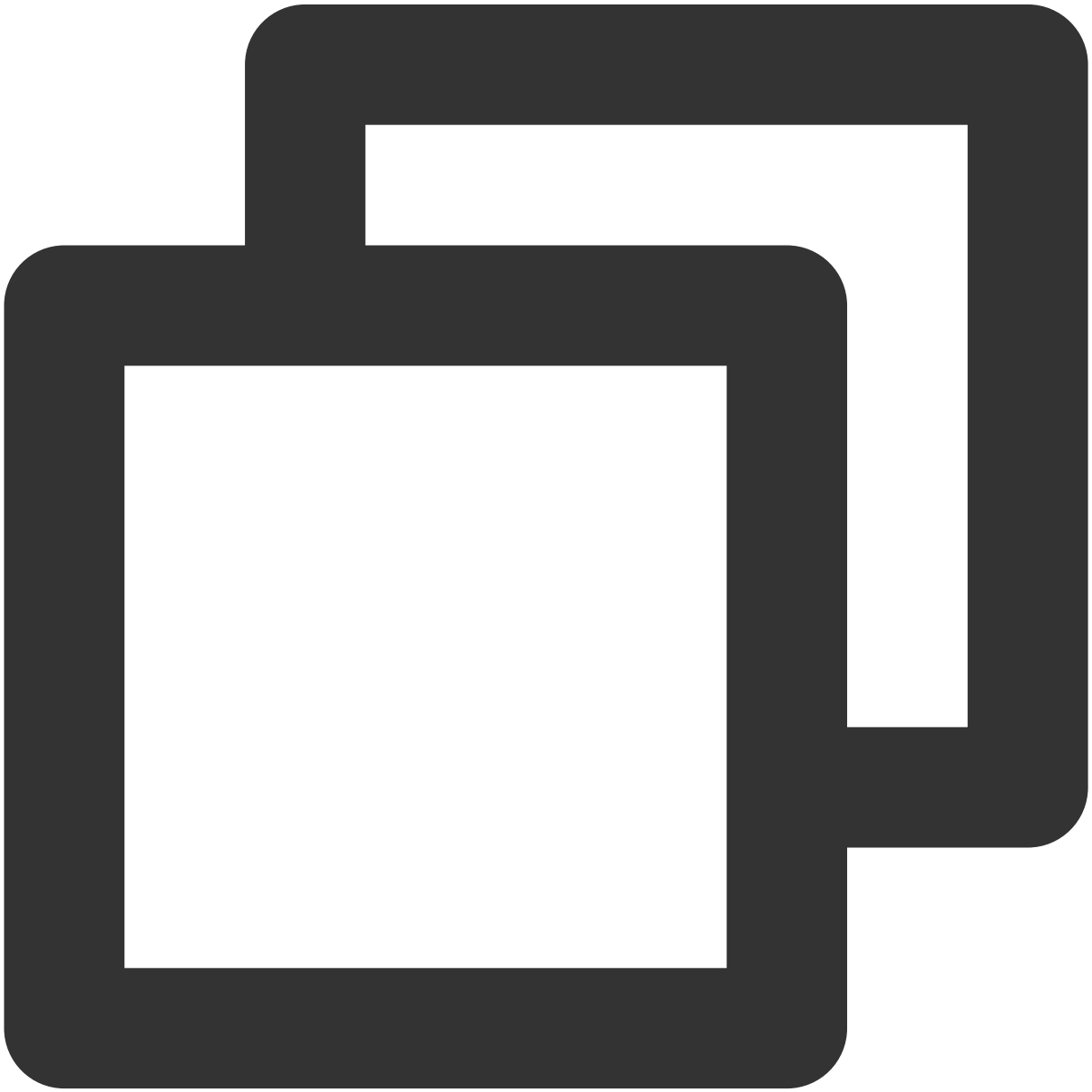
```
// 停止播放
[_txVodPlayer stopPlay];
```

```
[_txVodPlayer removeVideoWidget]; // 记得销毁 view 控件
```

基础功能使用

1、播放控制

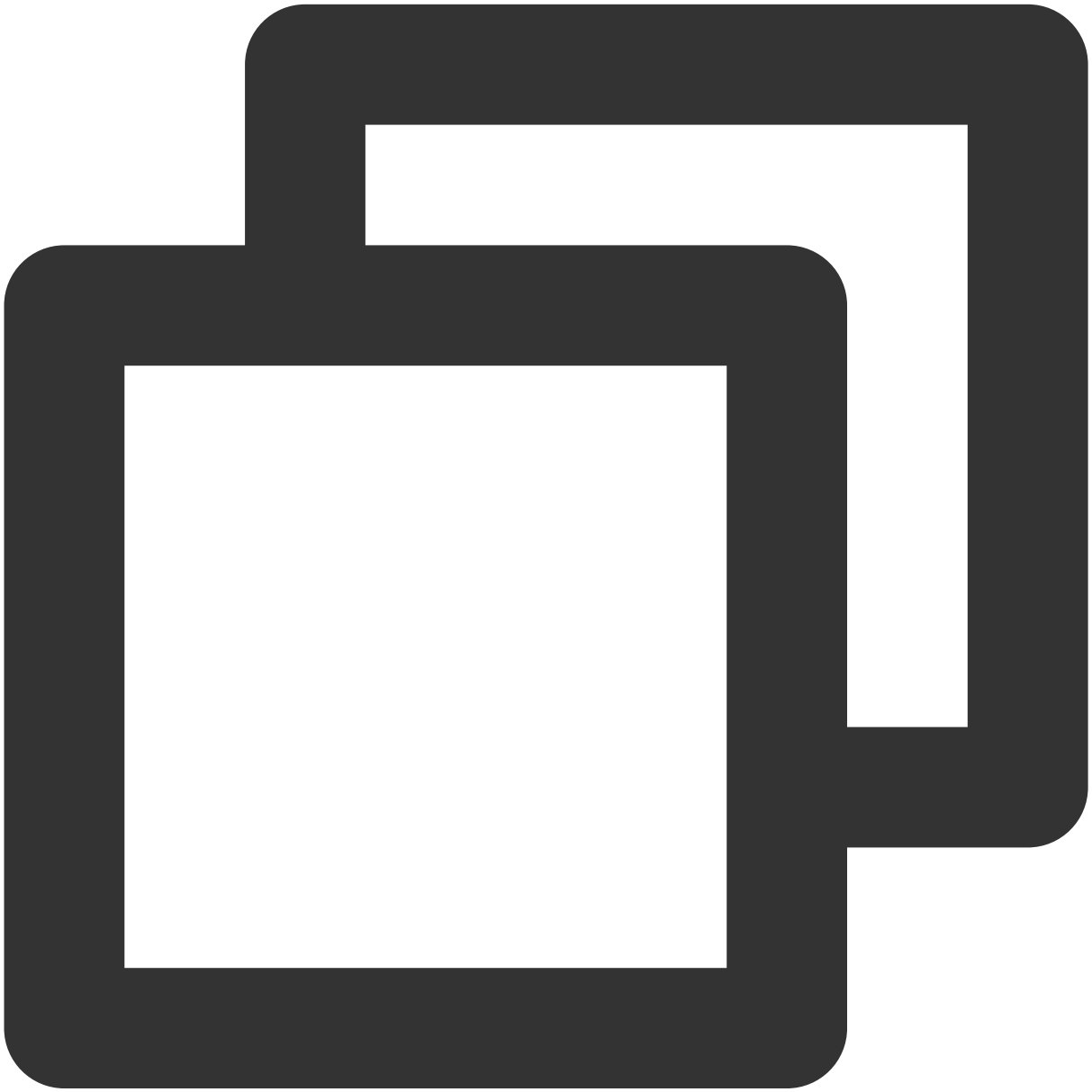
开始播放



```
// 开始播放
```

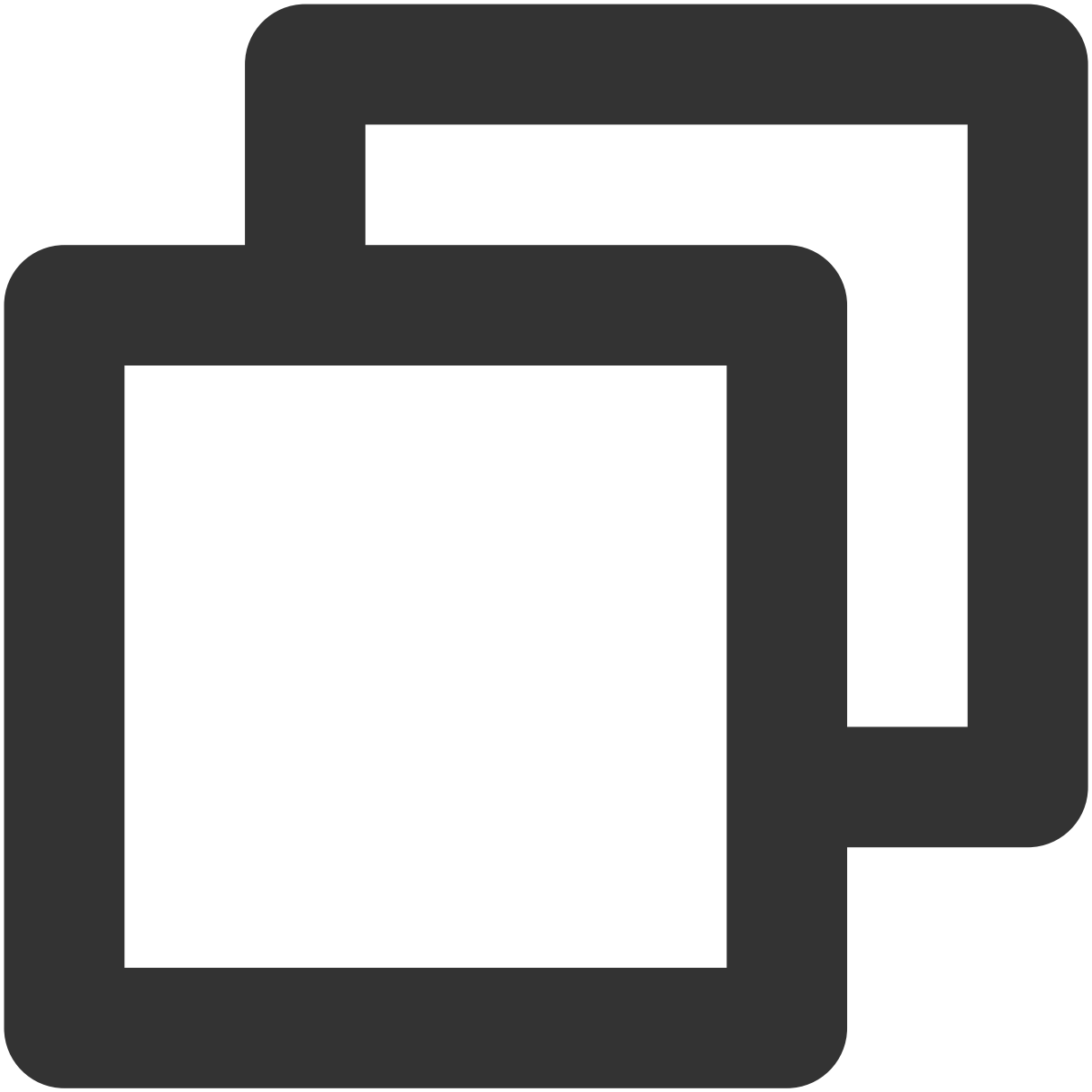
```
[_txVodPlayer startVodPlay:url];
```

暂停播放



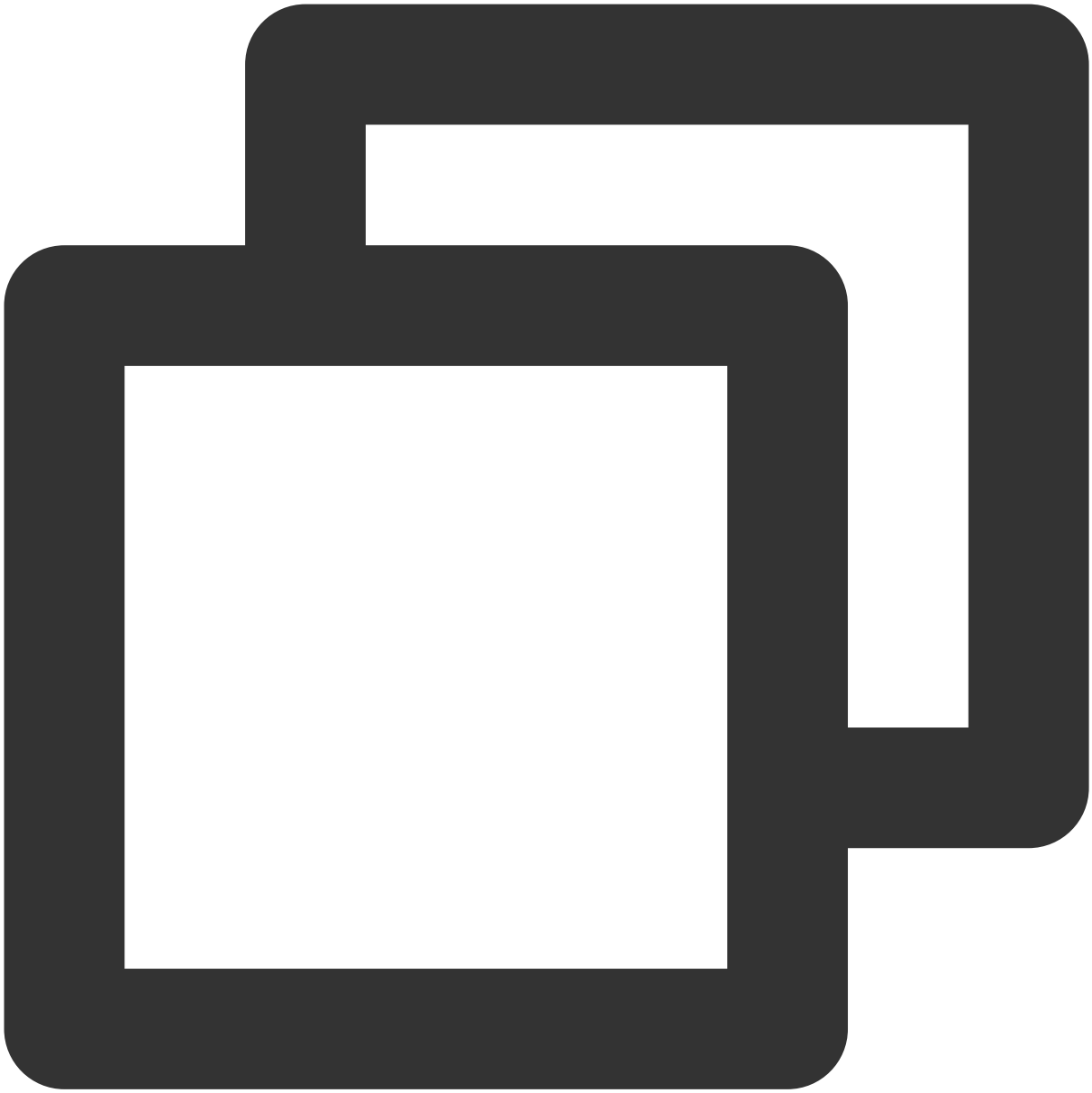
```
// 暂停播放
[_txVodPlayer pause];
```

恢复播放



```
// 恢复播放
[_txVodPlayer resume];
```

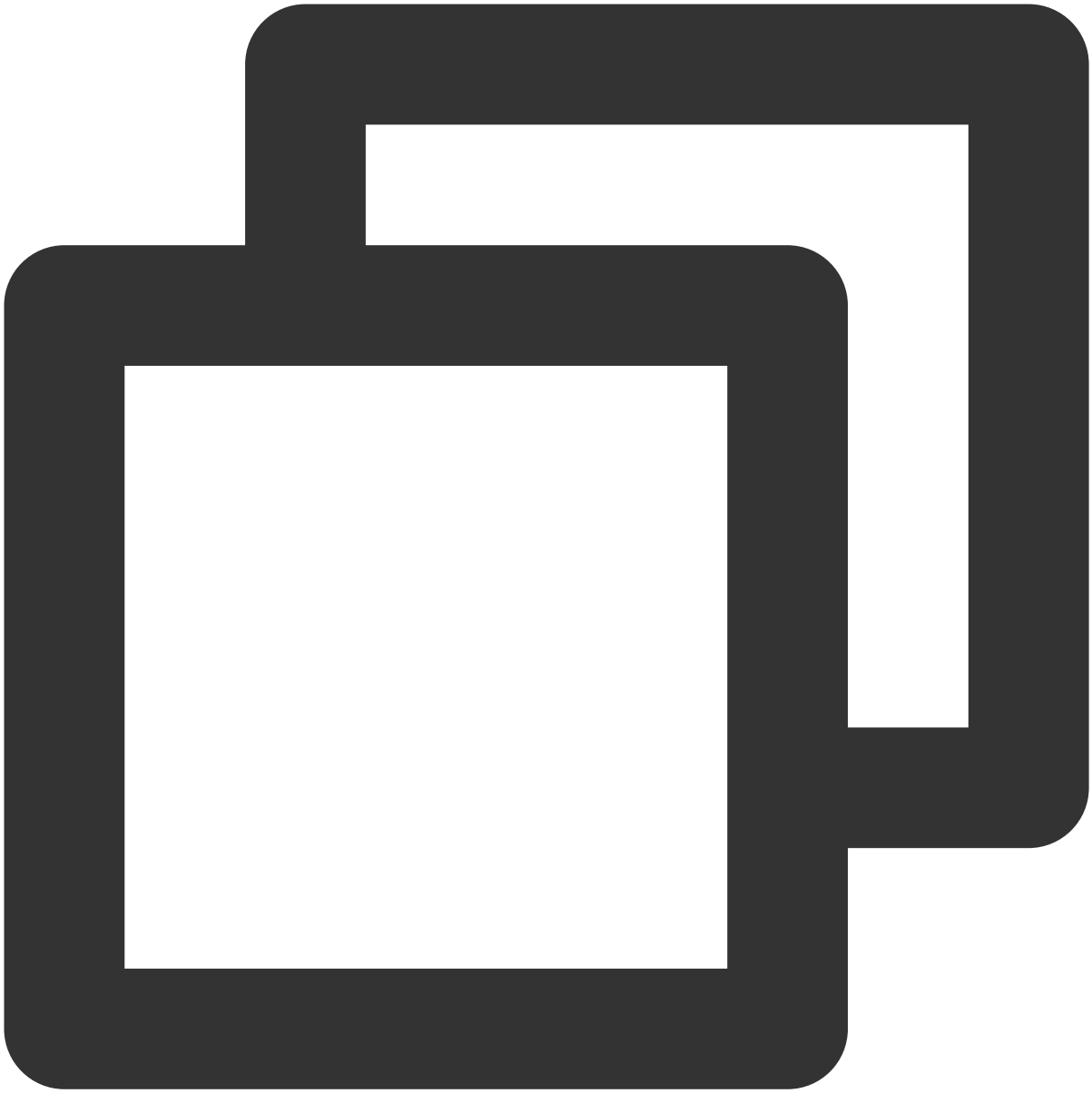
结束播放



```
// 结束播放
[_txVodPlayer stopPlay];
```

调整进度（Seek）

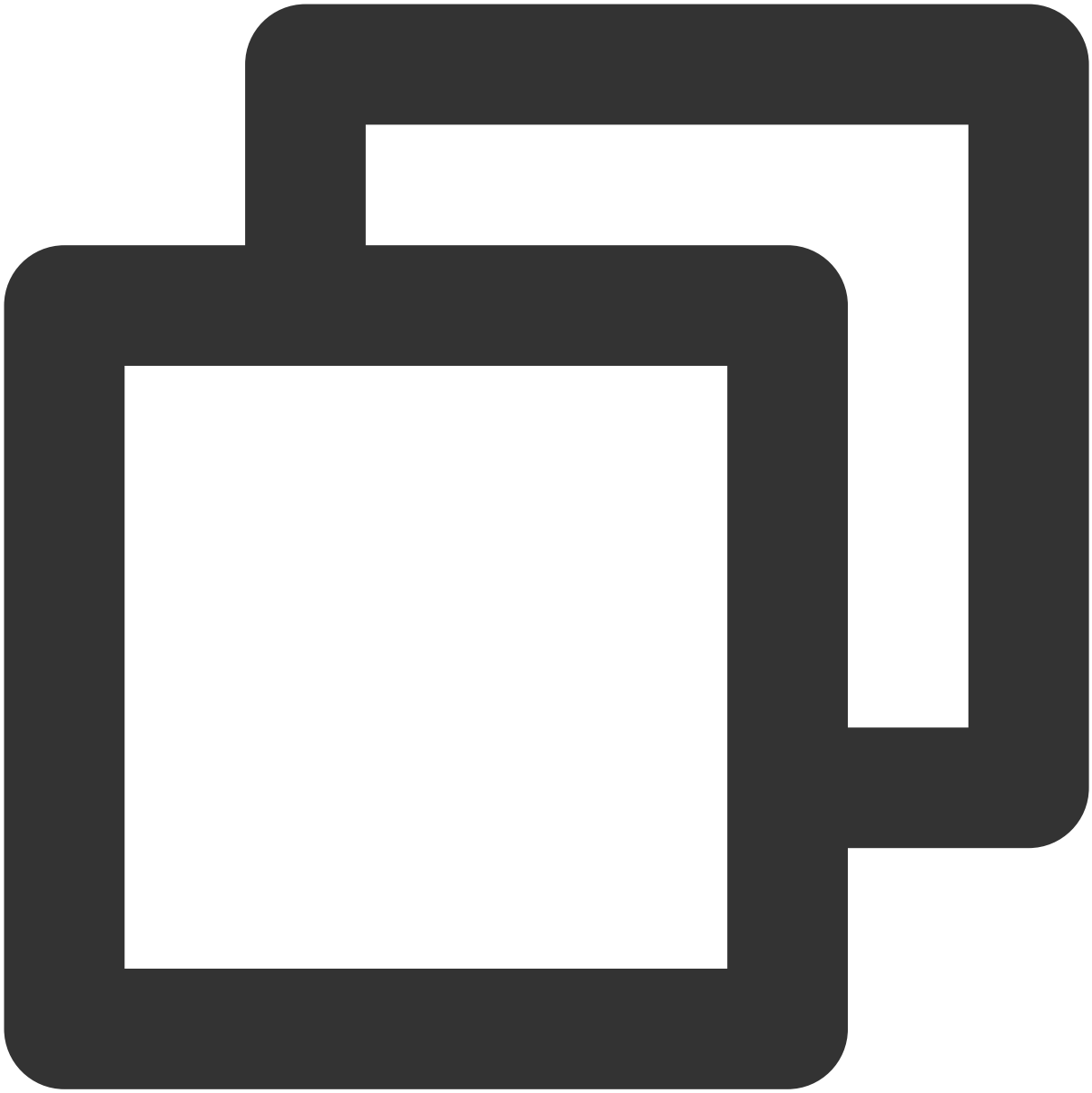
当用户拖拽进度条时，可调用 seek 从指定位置开始播放，播放器 SDK 支持精准 seek。



```
int time = 600; // int类型时，单位为 秒
// 调整进度
[_txVodPlayer seek:time];
```

精准和非精准 Seek

播放器 SDK 11.8 版本开始，支持调用 seek 接口时，指定精准或非精准 seek。

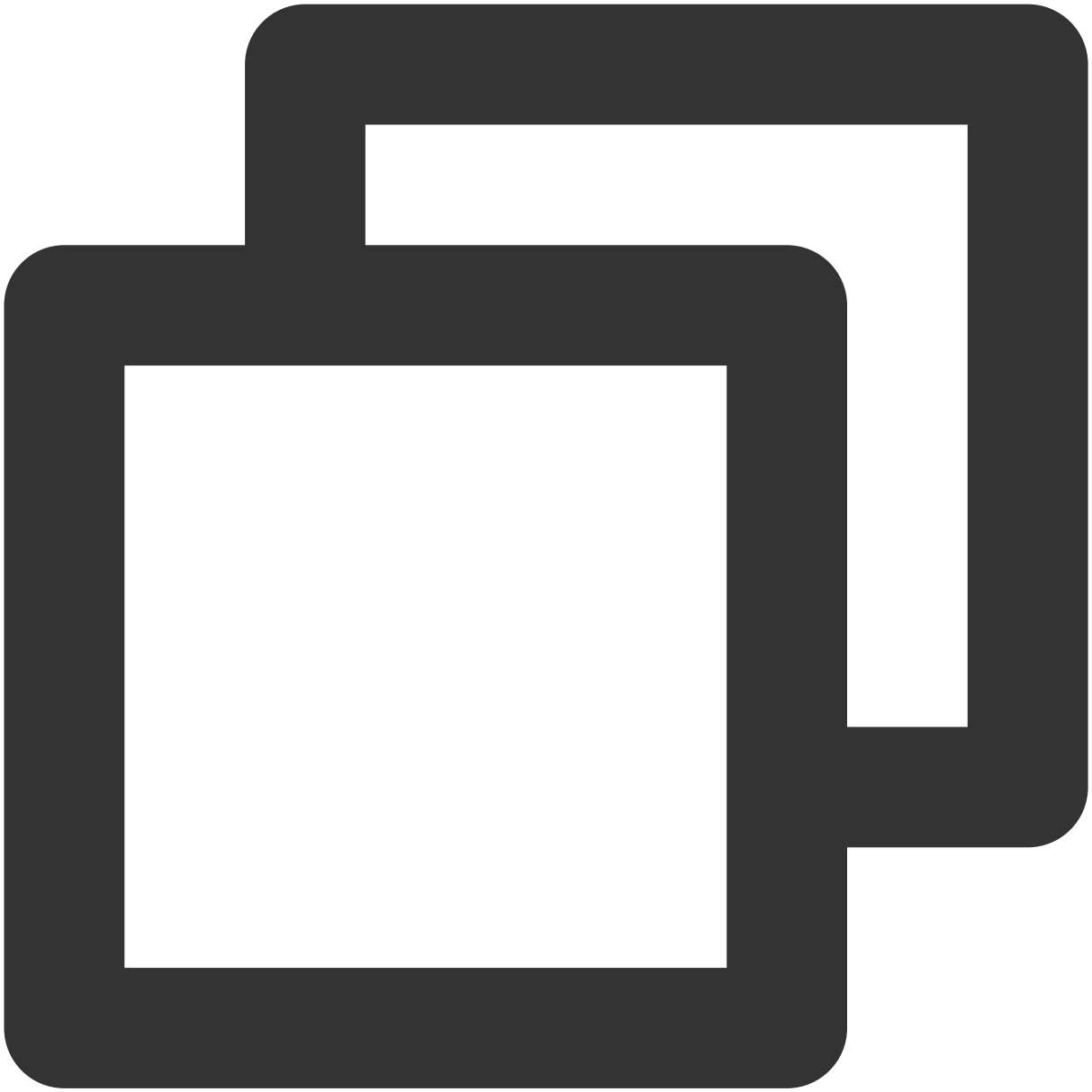


```
float time = 600; // float 类型时单位为 秒
// 调整进度
[_txVodPlayer seek:time accurateSeek:YES];  // 精准 seek
[_txVodPlayer seek:time accurateSeek:NO];  // 非精准 seek
```

Seek 到视频流指定 PDT 时间点

跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。

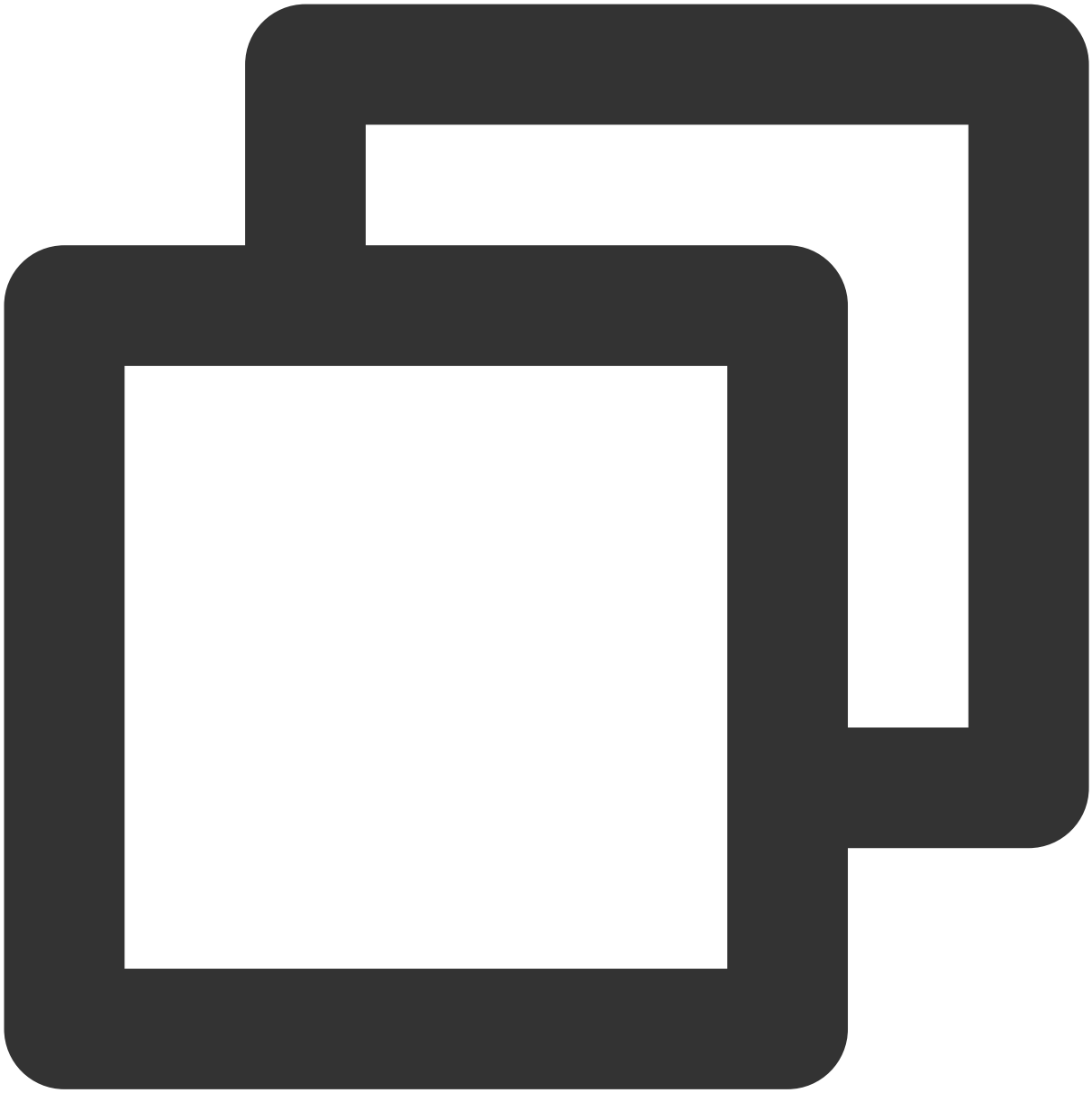
注意：播放器高级版 11.6 版本开始支持。



```
long long pdtTimeMs = 600; // 单位为 毫秒
[_txVodPlayer seekToPdtTime:time];
```

从指定时间开始播放

首次调用 startVodPlay之前，支持从指定时间开始播放。



```
float startTimeInSecond = 60; // 单位：秒
[_txVodPlayer setStartTime:startTimeInSecond];  // 设置开始播放时间
[_txVodPlayer startVodPlay:url];
```

2、画面调整

view：大小和位置

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

setRenderMode：铺满或适应

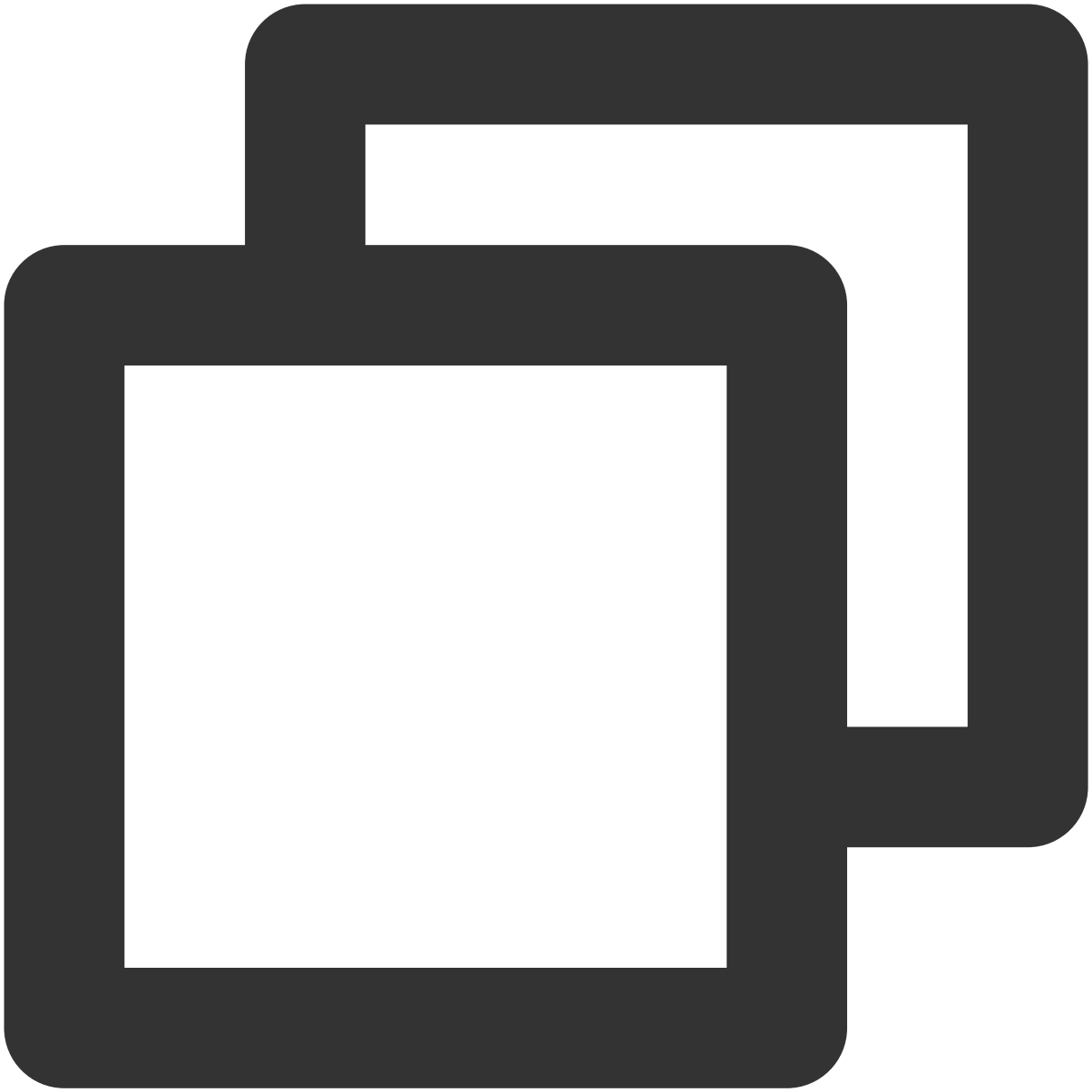
可选值	含义
RENDER_MODE_FILL_SCREEN	将图像等比例铺满整个屏幕，多余部分裁剪掉，此模式下画面不会留黑边，但可能因为部分区域被裁剪而显示不全。
RENDER_MODE_FILL_EDGE	将图像等比例缩放，适配最长边，缩放后的宽和高都不会超过显示区域，居中显示，画面可能会留有黑边。

setRenderRotation：画面旋转

可选值	含义
HOME_ORIENTATION_RIGHT	home 在右边
HOME_ORIENTATION_DOWN	home 在下面
HOME_ORIENTATION_LEFT	home 在左边
HOME_ORIENTATION_UP	home 在上面

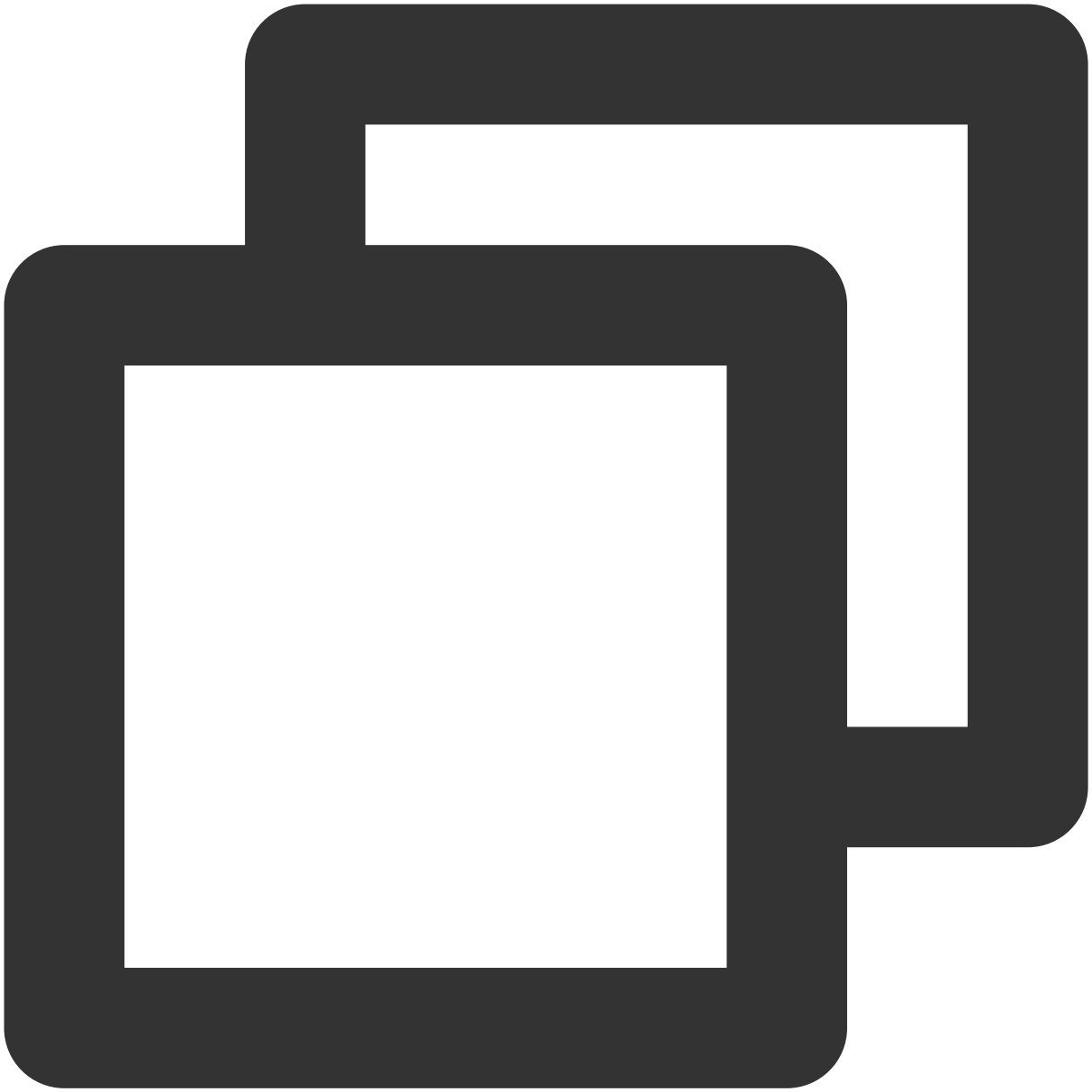
3、变速播放

点播播放器支持变速播放，通过接口 `setRate` 设置点播播放速率来完成，支持快速与慢速播放，如0.5X、1.0X、1.2X、2X等。



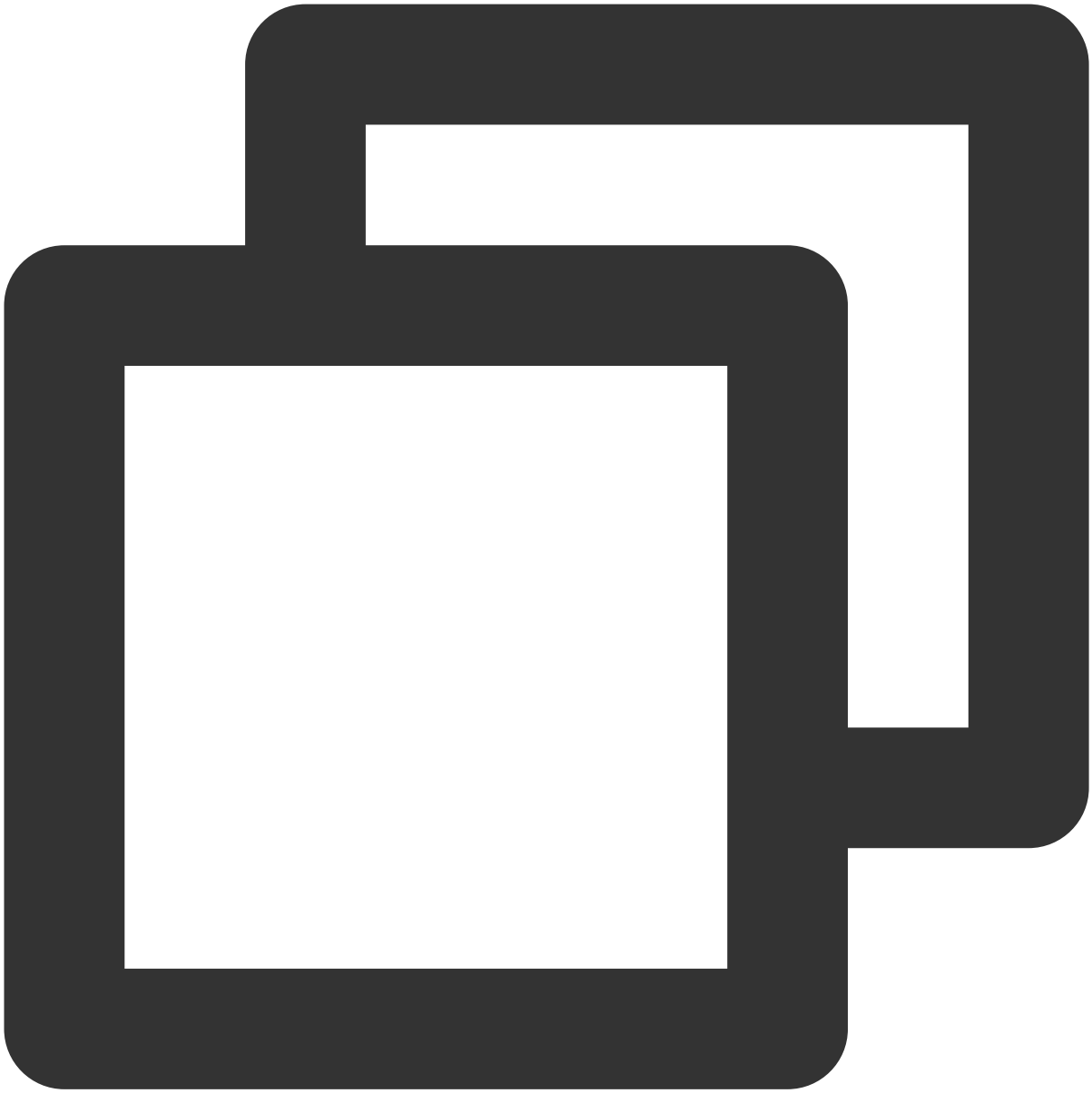
```
// 设置1.2倍速播放
[_txVodPlayer setRate:1.2];
// 开始播放
[_txVodPlayer startVodPlay:url];
```

4、循环播放



```
// 设置循环播放
[_txVodPlayer setLoop:true];
// 获取当前循环播放状态
[_txVodPlayer loop];
```

5、静音设置



```
// 设置静音，true 表示开启静音， false 表示关闭静音
[_txVodPlayer setMute:true];
```

6、屏幕截图

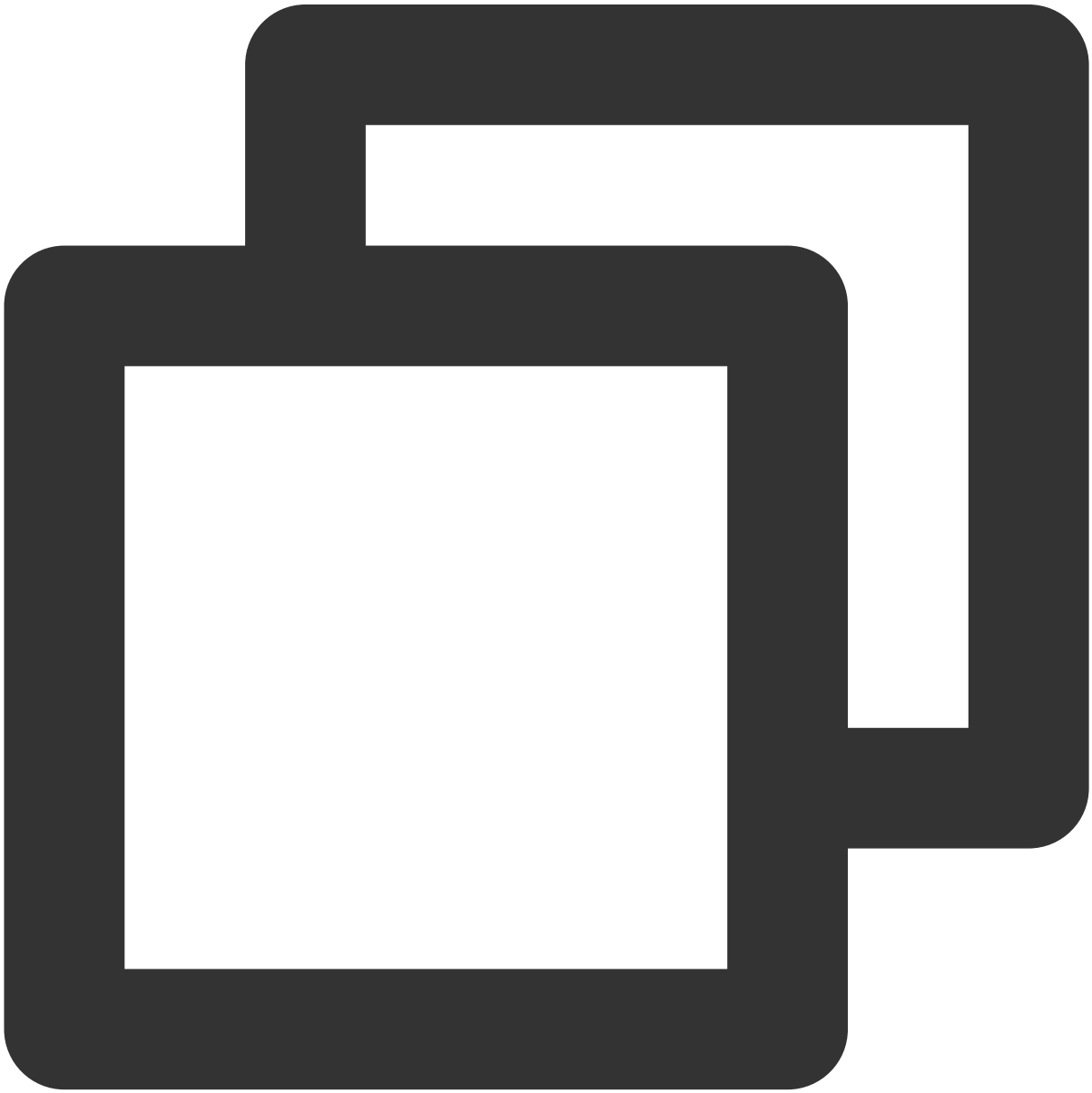
通过调用 **snapshot** 您可以截取当前视频为一帧画面，此功能只会截取当前直播流的视频画面，如果您需要截取当前的整个 UI 界面，请调用 iOS 的系统 API 来实现。

7、贴片广告

播放器 SDK 支持在界面添加图片贴片，用于广告宣传等。实现方式如下：
将 `autoPlay` 为 NO，此时播放器会正常加载，但视频不会立刻开始播放。
在播放器加载出来后、视频尚未开始时，即可在播放器界面查看图片贴片广告。
待达到广告展示结束条件时，使用 resume 接口启动视频播放。

8、HTTP-REF

TXVodPlayConfig 中的 headers 可以用来设置 HTTP 请求头，例如常用的防止 URL 被到处拷贝的 Referer 字段（腾讯云可以提供更加安全的签名防盗链方案），以及用于验证客户端身份信息的 Cookie 字段。

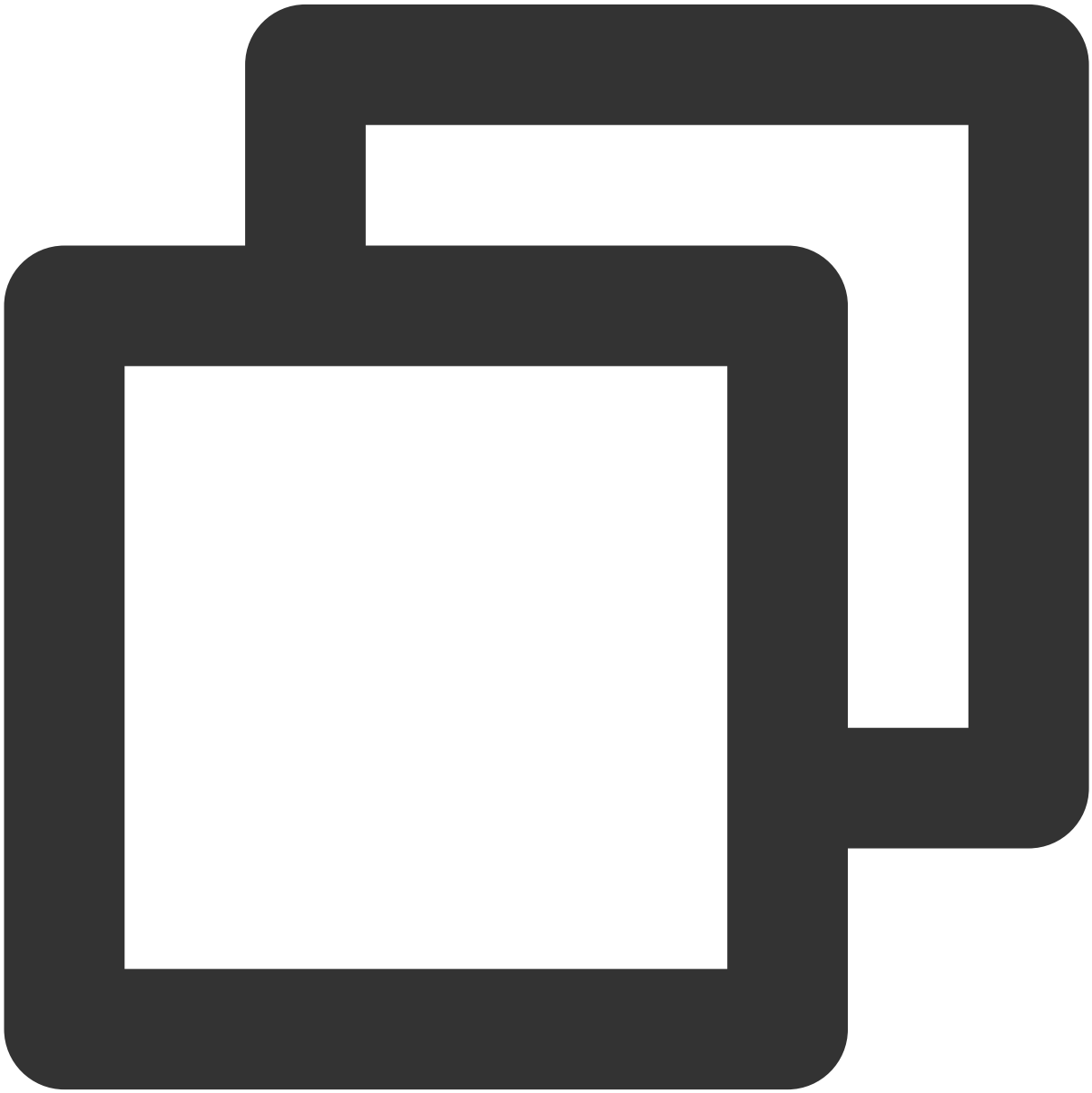


```
NSMutableDictionary<NSString *, NSString *> *httpHeader = [[NSMutableDictionary all
```

```
[httpHeader setObject:@"${Referer Content}" forKey:@"Referer"];
[_config setHeaders:httpHeader];
[_txVodPlayer setConfig:_config];
```

9、硬件加速

对于蓝光级别（1080p）的画质，简单采用软件解码的方式很难获得较为流畅的播放体验，所以如果您的场景是以游戏直播为主，一般都推荐开启硬件加速。
软解和硬解的切换需要在切换之前先 **stopPlay**，切换之后再 **startVodPlay**，否则会产生比较严重的花屏问题。

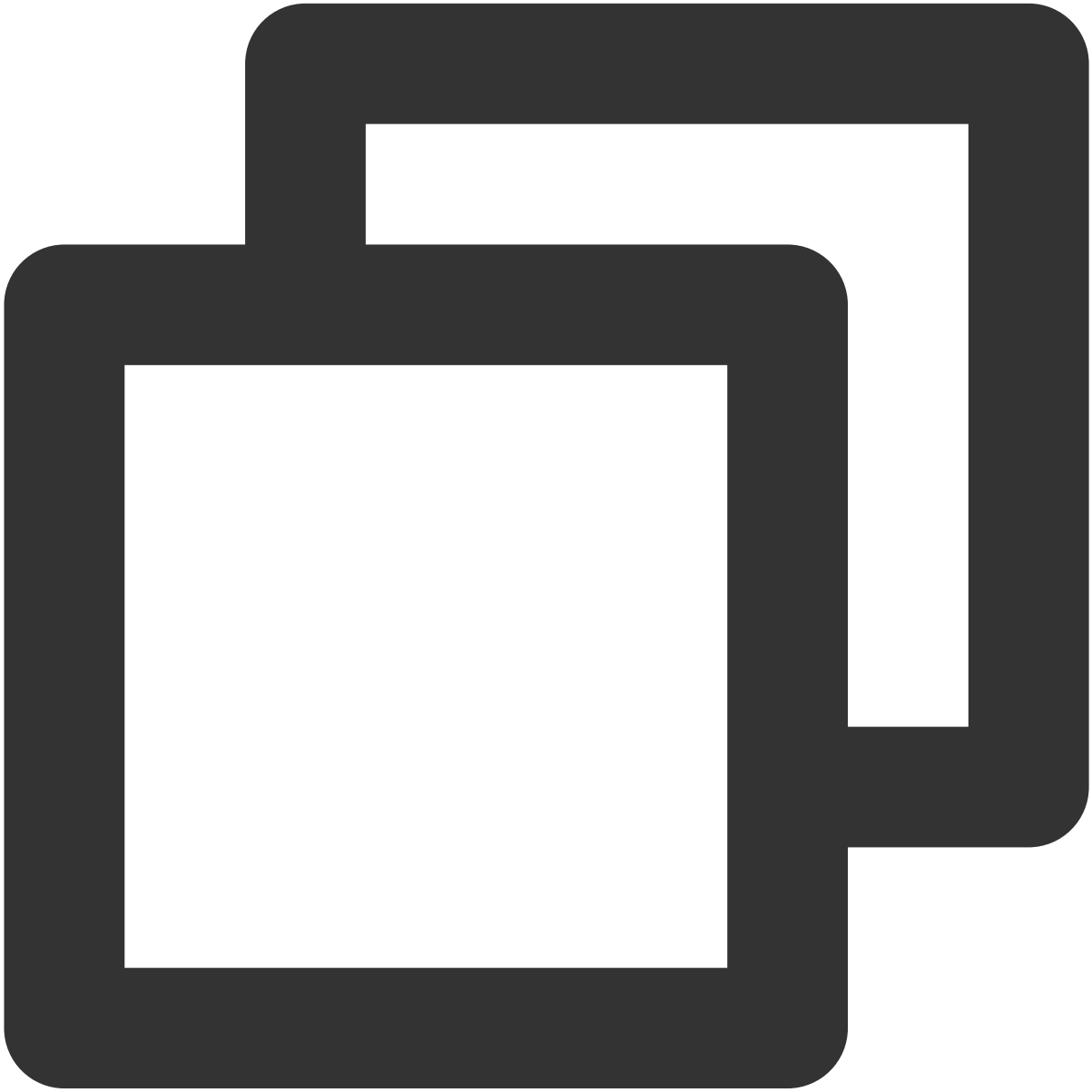


```
[_txVodPlayer stopPlay];
```

```
_txVodPlayer.enableHWAcceleration = YES;
[_txVodPlayer startVodPlay:_flvUrl type:_type];
```

10、清晰度设置

SDK 支持 hls 的多码率格式，方便用户切换不同码率的播放流。可以通过下面方法获取多码率数组：



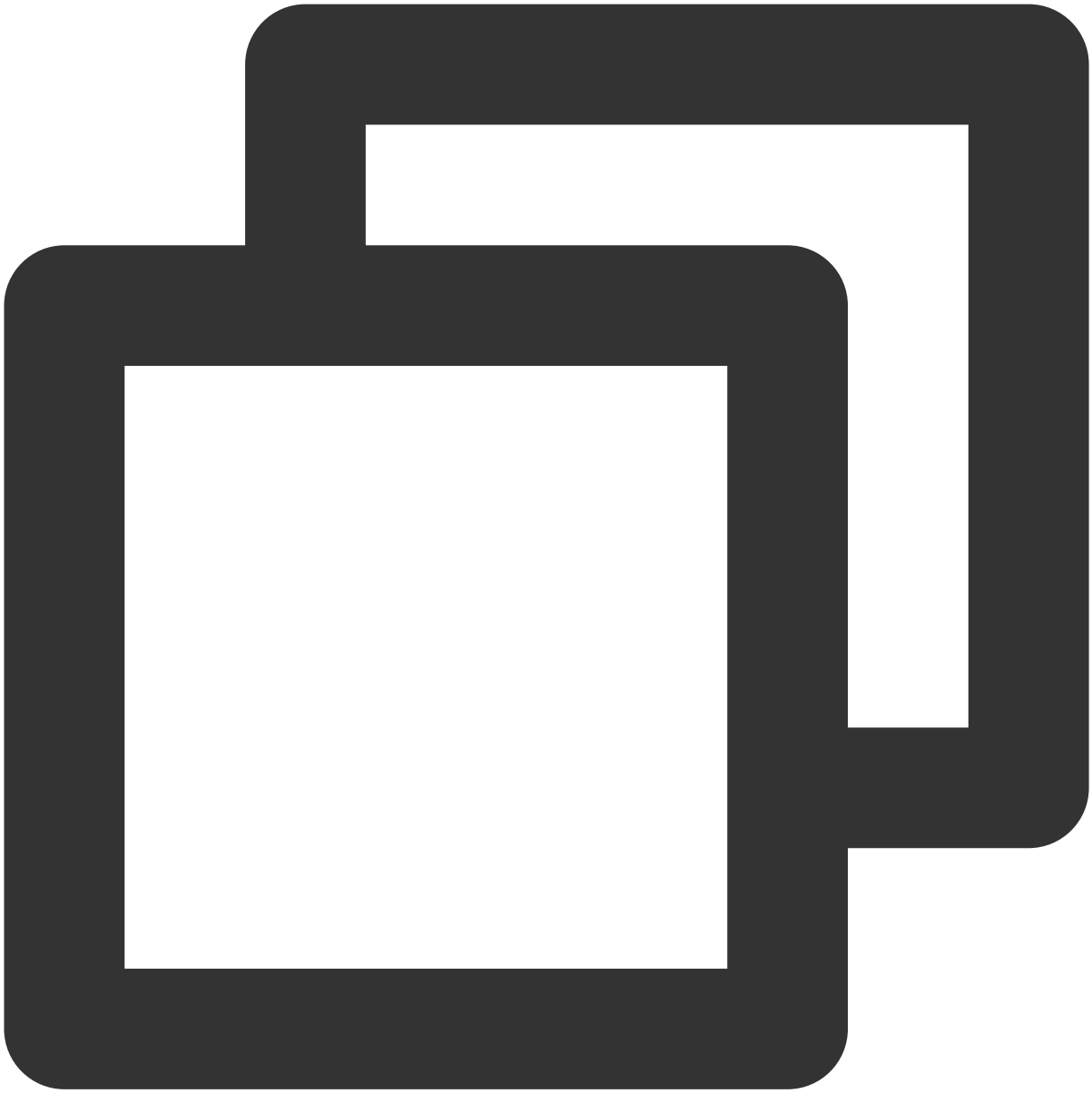
```
NSArray *bitrates = [_txVodPlayer supportedBitrates]; //获取多码率数组
// TXBitrateItem 类字段含义：index-码率下标；width-视频宽；height-视频高；birate-视频码率
TXBitrateItem *item = [bitrates objectAtIndex:i];
[_txVodPlayer setBitrateIndex:item.index]; // 切换码率到想要的清晰度
```

```
// 获取当前播放的码率下标，返回值 -1000 为默认值，表示没有设置过码率标；返回值 -1 表示开启了自适应
int index = [_txVodPlayer bitrateIndex];
```

在播放过程中，可以随时通过 `-[TXVodPlayer setBitrateIndex:]` 切换码率。切换过程中，会重新拉取另一条流的数据，因此会有稍许卡顿。SDK 针对腾讯云的多码率文件做过优化，可以做到切换无卡顿。
如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率，从而避免播放后切换码流。详细方法参考 播放器配置#启播前指定分辨率。

11、码流自适应

SDK 支持 HLS 的多码流自适应，开启相关能力后播放器能够根据当前带宽，动态选择最合适的码率播放。可以通过下面方法开启码流自适应：

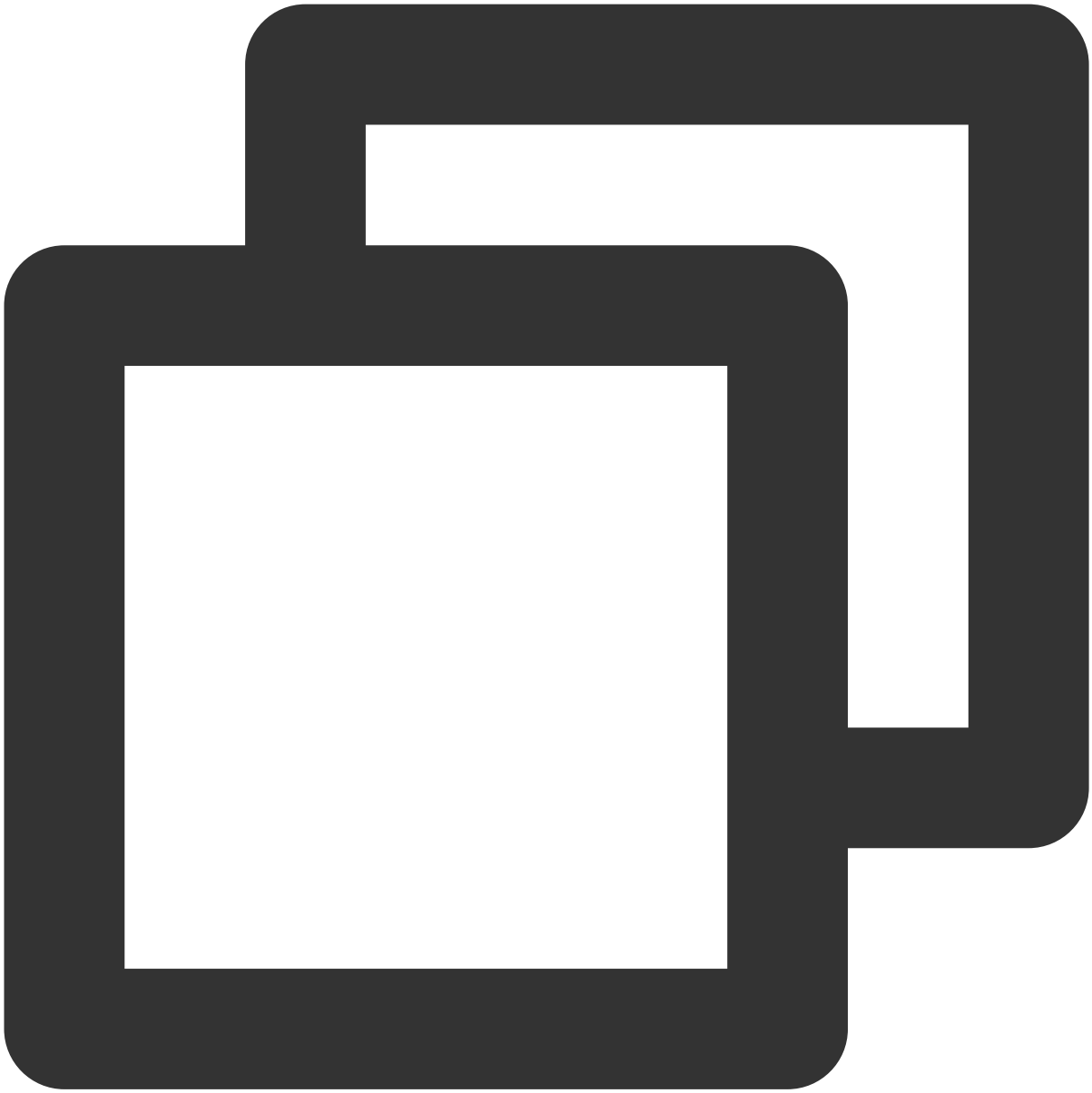


```
[_txVodPlayer setBitrateIndex:-1]; //index 参数传入-1
```

在播放过程中，可以随时通过 `-[TXVodPlayer setBitrateIndex:]` 切换其它码率，切换后码流自适应也随之关闭。

12、开启平滑切换码率

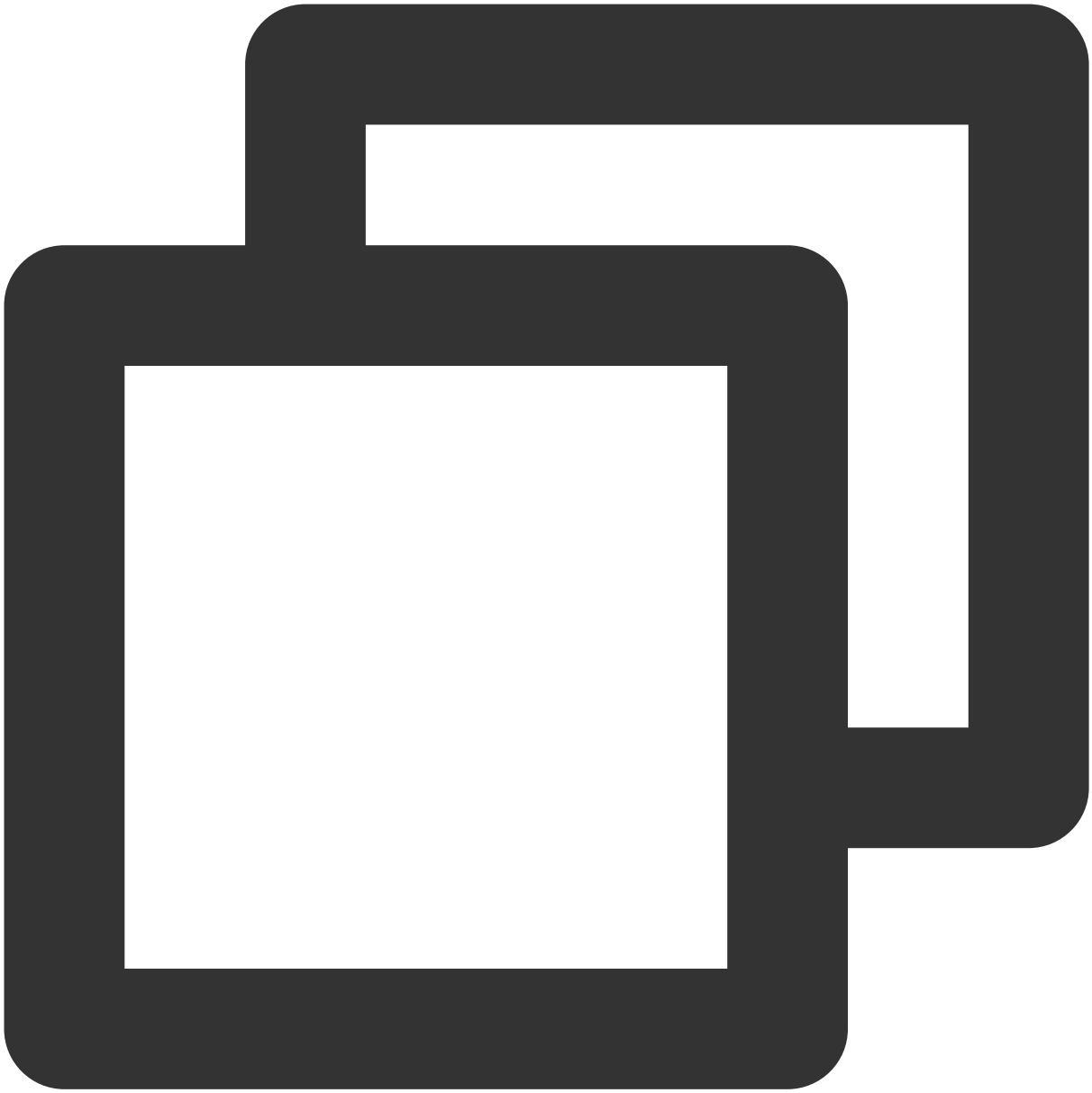
在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
// 设为YES，在IDR对齐时可平滑切换码率，设为NO时，可提高多码率地址打开速度
[_config setSmoothSwitchBitrate:YES];
[_txVodPlayer setConfig:_config];
```

13、播放进度监听

点播播放中的进度信息分为2种：**加载进度**和**播放进度**，SDK 目前是以事件通知的方式将这两个进度实时通知出来的。更多事件通知内容参见 事件监听。



```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
    if (EvtID == PLAY_EVT_PLAY_PROGRESS) {
            // 加载进度，单位是秒，小数部分为毫秒
            float playable = [param[EVT_PLAYABLE_DURATION] floatValue];
                [_loadProgressBar setValue:playable];

            // 播放进度，单位是秒，小数部分为毫秒
            float progress = [param[EVT_PLAY_PROGRESS] floatValue];
                [_seekProgressBar setValue:progress];

            // 视频总长，单位是秒，小数部分为毫秒
```

```
            float duration = [param[EVT_PLAY_DURATION] floatValue];
            // 可以用于设置时长显示等等

            // 获取 PDT 时间，播放器高级版 11.6 版本开始支持
            long long pdt_time_ms = [param[VOD_PLAY_EVENT_PLAY_PDT_TIME_MS] longLon
    }
}
```

14、播放网速监听

通过 事件监听 方式，可以在视频播放卡顿时在显示当前网速。

通过 `onNetStatus` 的 `NET_SPEED` 获取当前网速。具体使用方法见 状态反馈（onNetStatus）。

监听到 `PLAY_EVT_PLAY_LOADING` 事件后，显示当前网速。

收到 `PLAY_EVT_VOD_LOADING_END` 事件后，对显示当前网速的 view 进行隐藏。

15、获取视频分辨率

播放器 SDK 通过 URL 字符串播放视频，URL 中本身不包含视频信息。为获取相关信息，需要通过访问云端服务器加载到相关视频信息，因此 SDK 只能以事件通知的方式将视频信息发送到您的应用程序中，更多内容参见事件监听。

分辨率信息

方法1

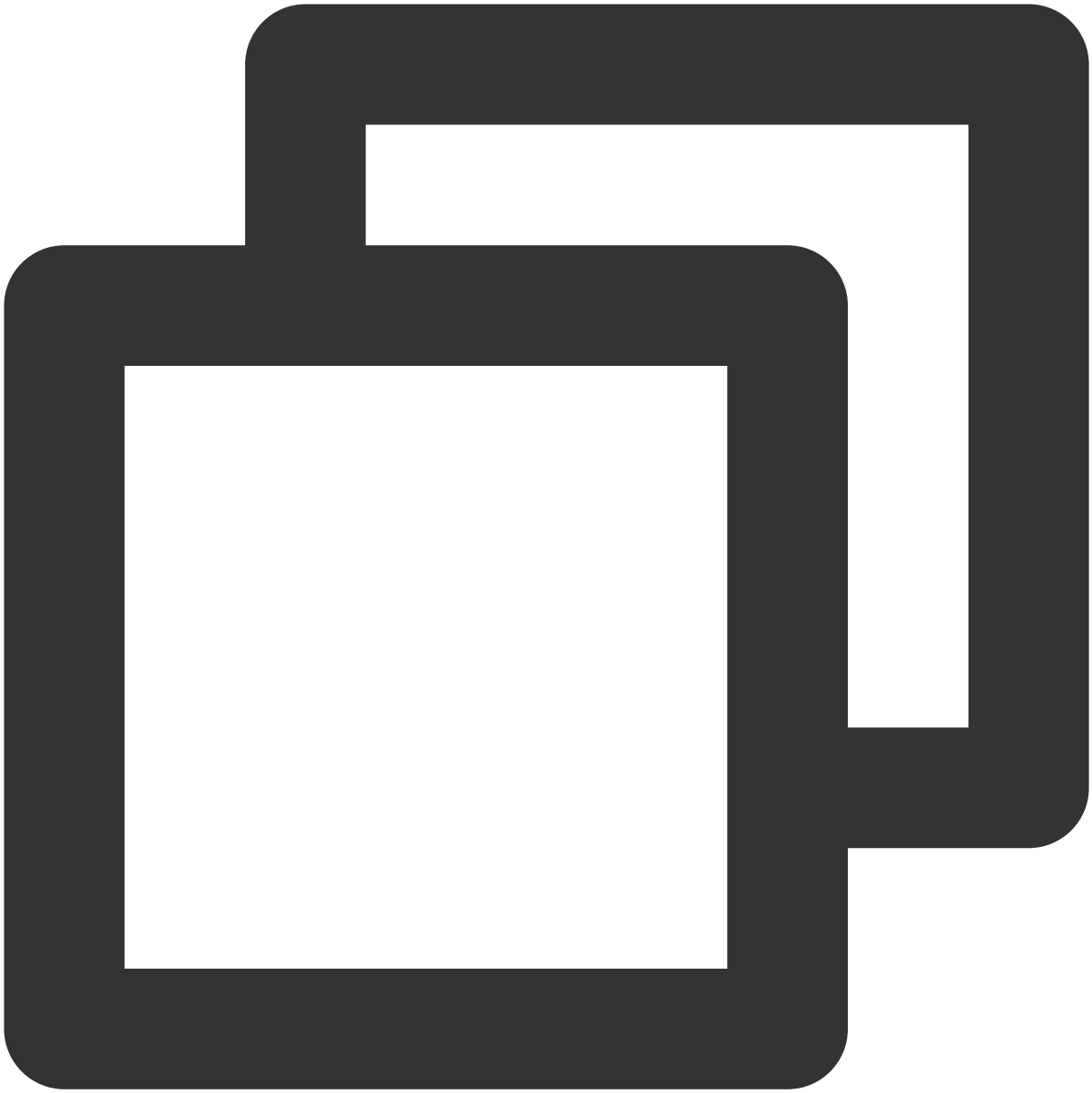
方法2

通过 `onNetStatus` 的 `VIDEO_WIDTH` 和 `VIDEO_HEIGHT` 获取视频的宽和高。具体使用方法见状态反馈（onNetStatus）。

直接调用 `-[TXVodPlayer width]` 和 `-[TXVodPlayer height]` 获取当前宽高。

16、播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



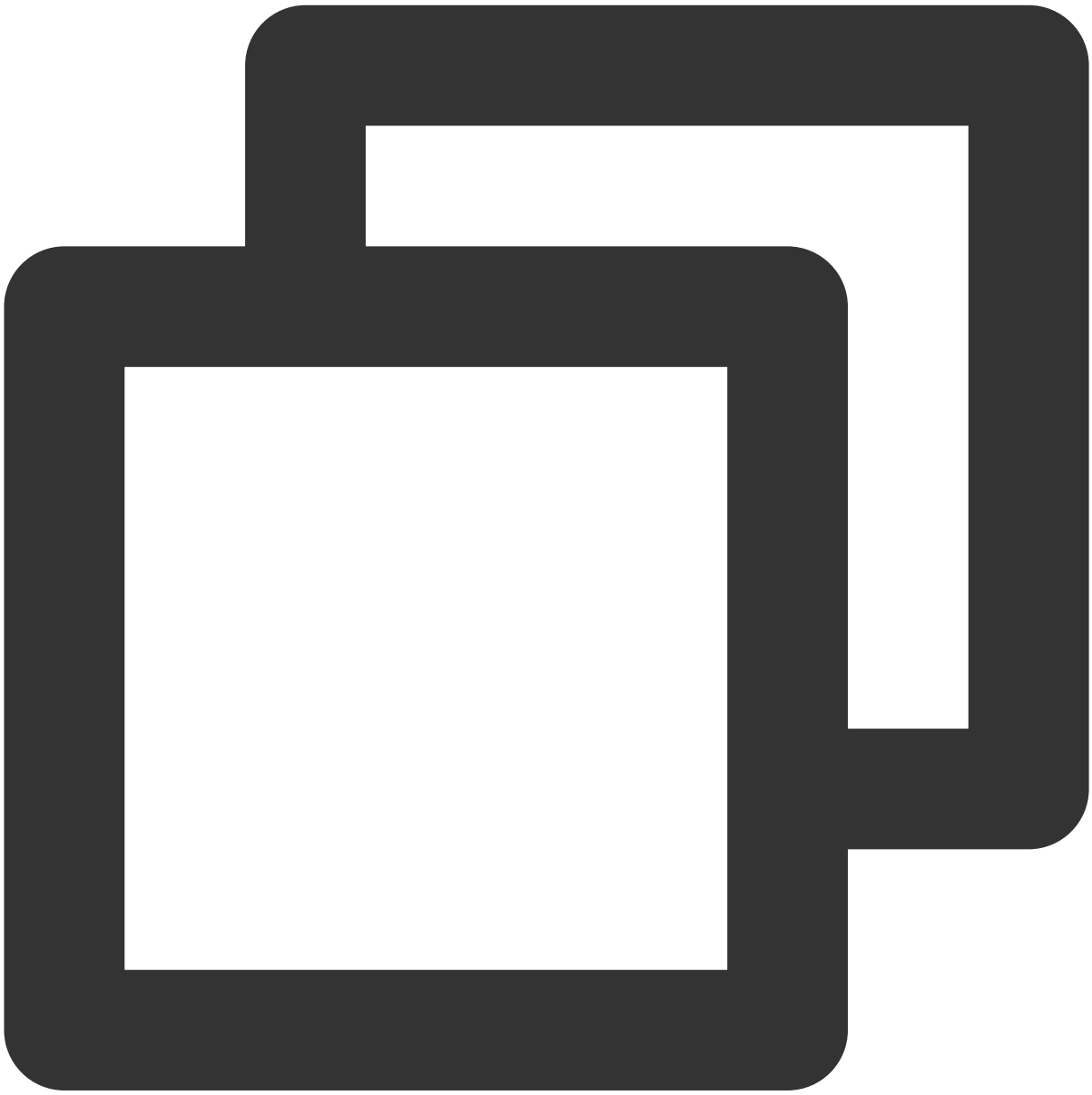
```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10];  // 播放时最大缓冲大小。单位：MB
[_txVodPlayer setConfig:_config];  // 把config 传给 _txVodPlayer
```

17、视频本地缓存

在短视频播放场景中，视频文件的本地缓存是很刚需的一个特性，对于普通用户而言，一个已经看过的视频再次观看时，不应该再消耗一次流量。

格式支持：SDK 支持 HLS（m3u8）和 MP4 两种常见点播格式的缓存功能。

开启时机：SDK 并不默认开启缓存功能，对于用户回看率不高的场景，也并不推荐您开启此功能。
开启方法：开启此功能需要配置两个参数：本地缓存目录及缓存大小。



```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
   [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                            withIntermediateDirectories:NO
                                             attributes:nil
```

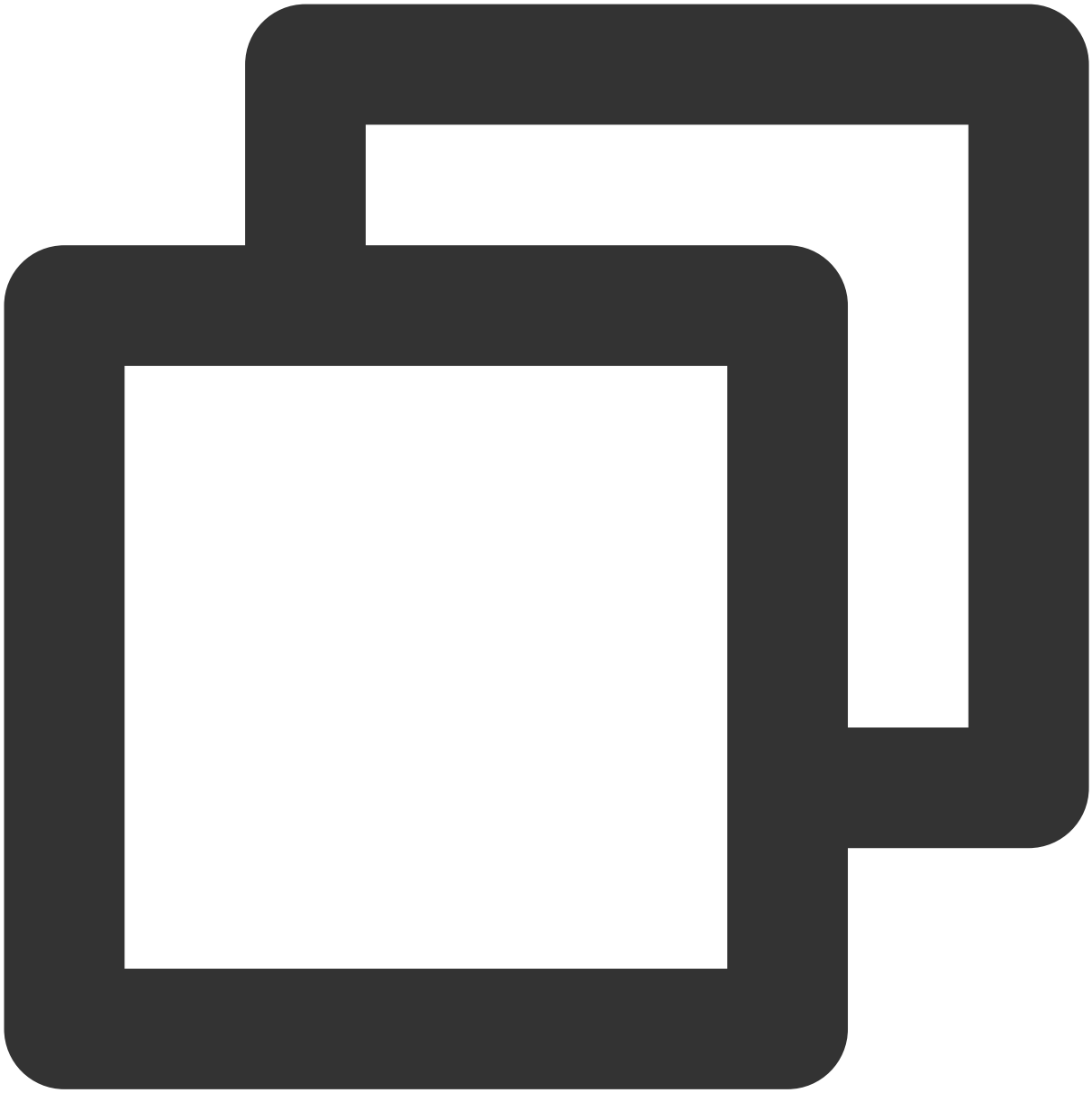
```
                                                        error:&error];
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];
// 开始播放
[_txVodPlayer startVodPlay:url];
```

说明：

旧版本通过 TXVodPlayConfig#setMaxCacheItems 接口配置已经废弃，不推荐使用。

18、屏幕控制（亮屏和灭屏）

由于手机个性化设置中经常会设置屏幕锁定的时间，在视频播放场景中可能会出现屏幕灭屏（或被锁定）的情况，这极大影响了用户体验。因此，为了解决这一情况需要在播放过程中的相关时机添加如下代码，以便使屏幕一直处于亮屏状态。



```
（1）亮屏(禁止灭屏)
// 启动播放（startVodPlay / startPlayDrm / startVodPlayWithParams）
// 恢复播放（resume）
 [[UIApplication sharedApplication] setIdleTimerDisabled:YES];

（2）灭屏（恢复灭屏）
// 停止（stopPlay）
// 暂停（pause）
[[UIApplication sharedApplication] setIdleTimerDisabled:NO];
```

注意：

以上接口的使用请注意在 主线程 调用。

19、DRM 加密视频播放

注意：

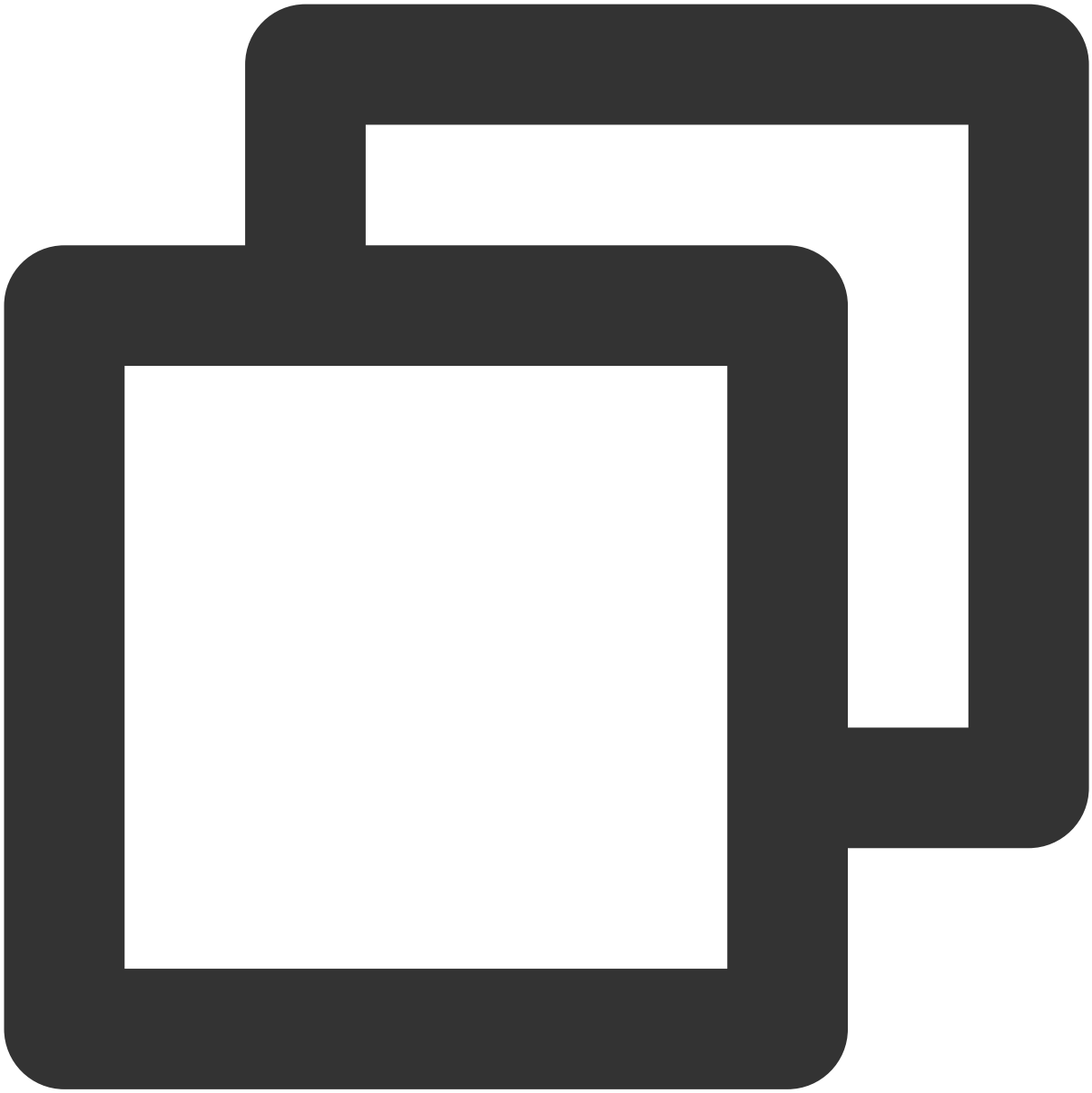
此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持播放商业级 DRM 加密视频，目前支持 WideVine 和 Fairplay 2种 DRM 方案。更多的商业级 DRM 信息，请参考产品介绍。

可通过下面 2 种播放 DRM 加密视频：

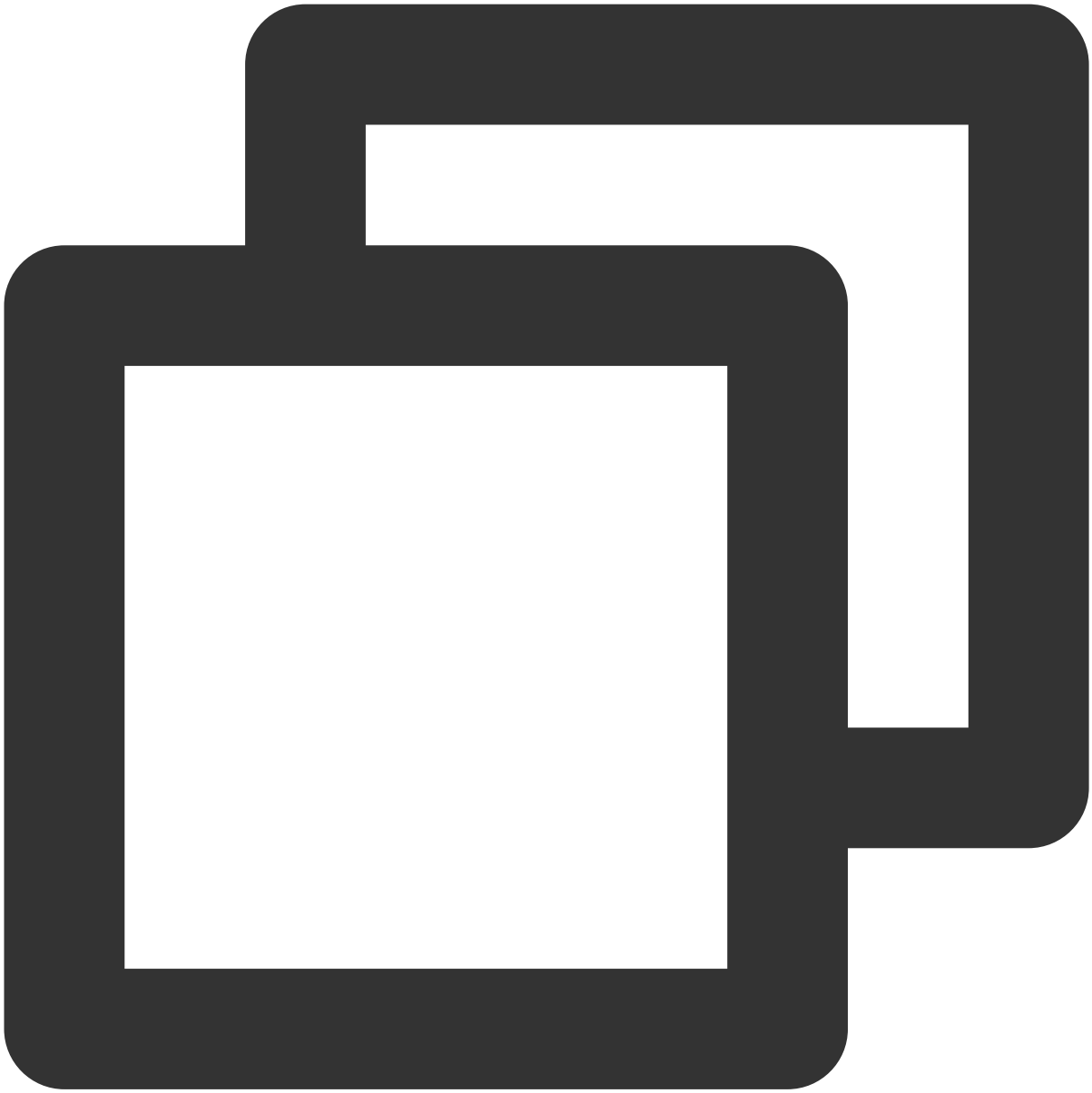
通过FileId播放

自定义配置播放



```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = ${appId}; // 腾讯云账户的 appId
p.fileId = @"${fieId}"; // DRM 加密视频的 fileId
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/zh/documen
p.sign = @"${psgin}"; // 加密视频的播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

通过 FileId 播放适用于接入云点播后台。这种方式和播放普通 FileId 文件没有差别，需要在云点播先配置资源为 DRM 类型，SDK 会在内部识别并处理。



```
// 通过 TXVodPlayer#startPlayDrm 接口播放
// @param certificateUrl 证书提供商url
// @param licenseUrl 解密的key url
// @param videoUrl 待播放视频的Url地址
TXPlayerDrmBuilder *builder = [[TXPlayerDrmBuilder alloc] initWithDeviceCertificate
[_txVodPlayer startPlayDrm:builder];
```

20、外挂字幕

注意：

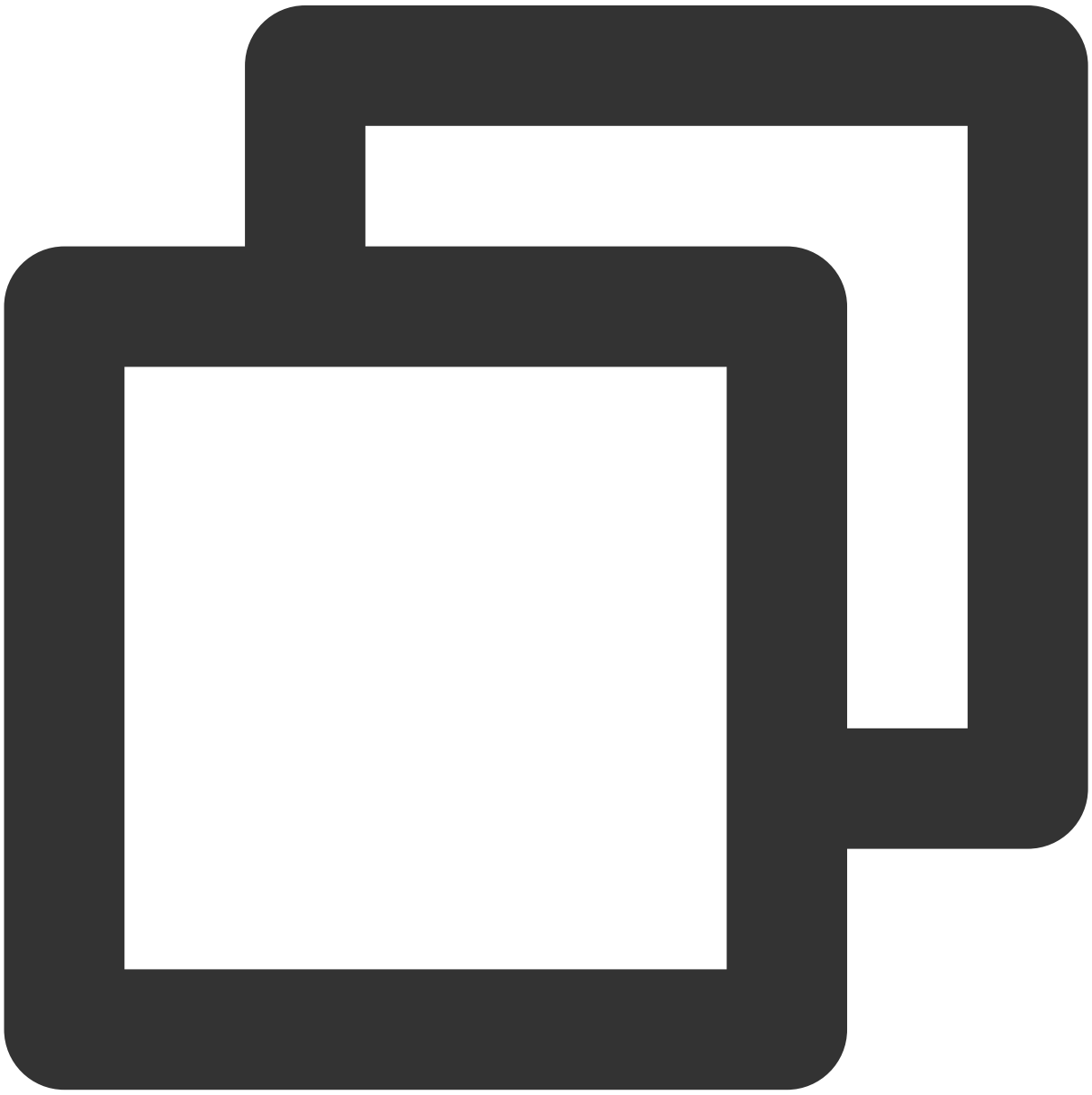
此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这 2 种格式的字幕。

最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。 添加字幕后，并不会主动加载字幕， 调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。

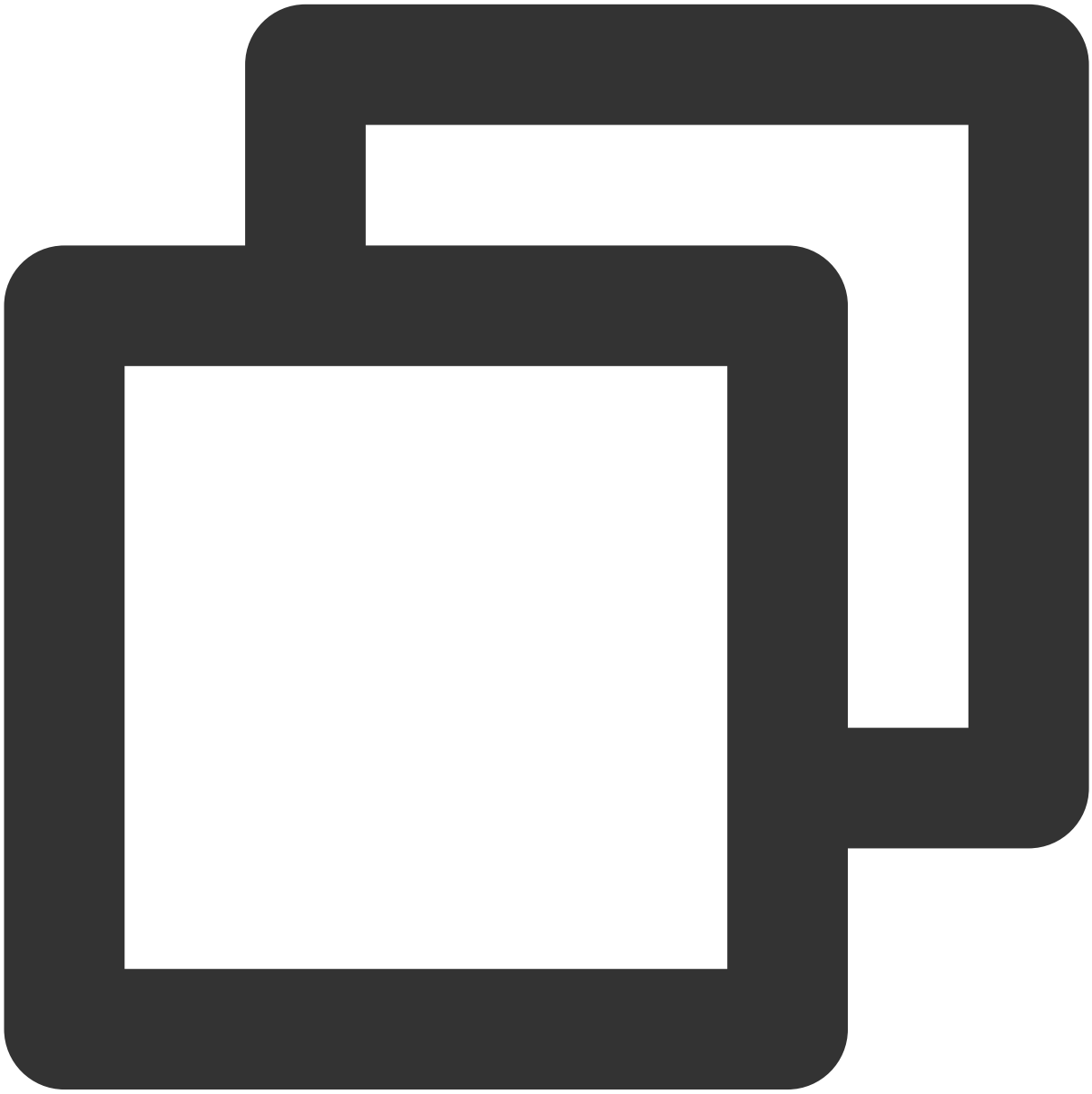
用法如下：

步骤1：添加外挂字幕。



```
// 传入 字幕url， 字幕名称， 字幕类型，建议在启动播放前添加
[_txVodPlayer addSubtitleSource:@"https://mediacloud-76607.gzc.vod.tencent-cloud.co
```

步骤2：播放后切换字幕。

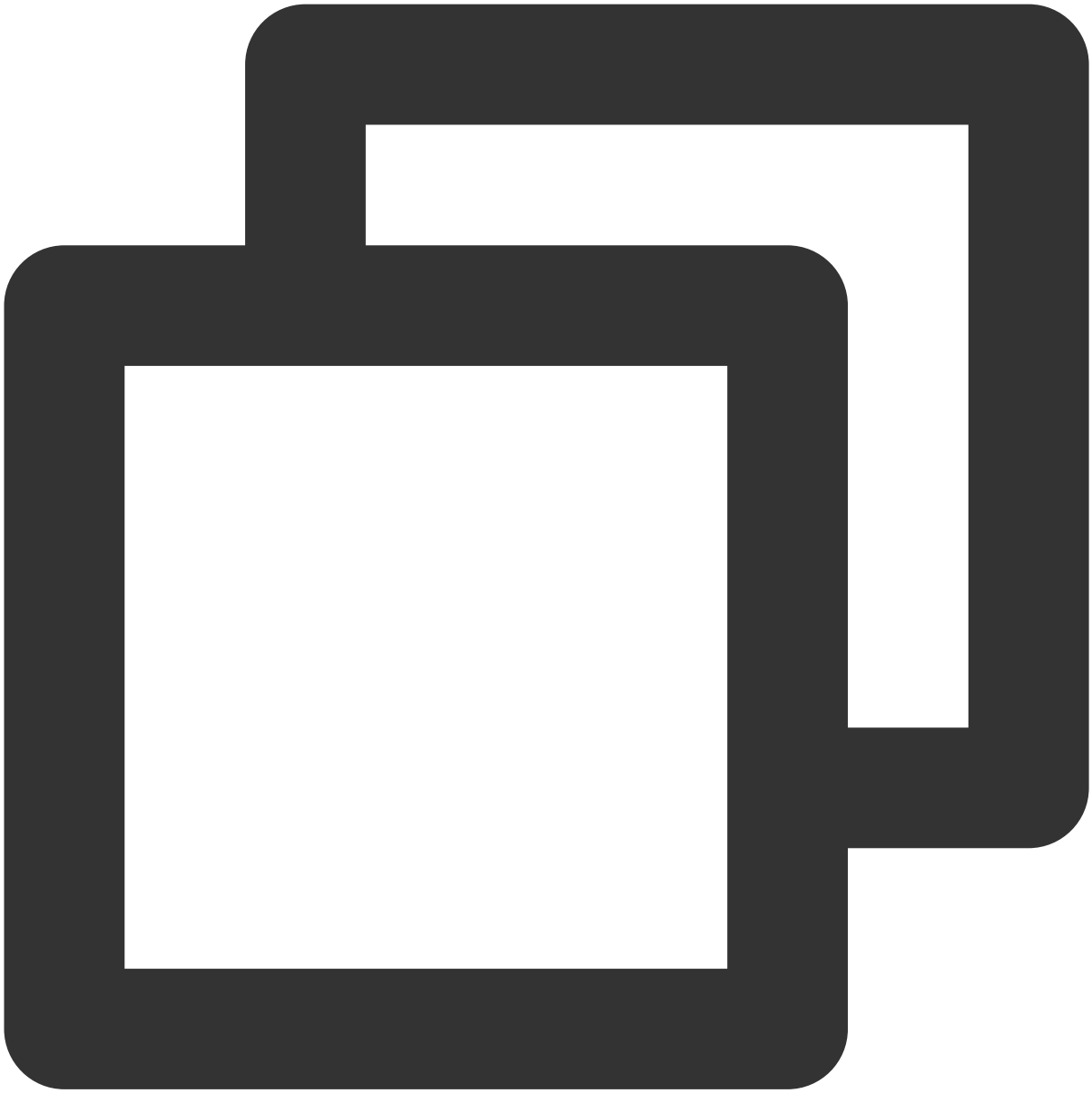


```
// 开始播放视频后，选中添加的外挂字幕，请在收到 VOD_PLAY_EVT_SELECT_TRACK_COMPLETE 事件后调用
NSArray<TXTrackInfo *> *subtitlesArray = [_txVodPlayer getSubtitleTrackInfo];
for (int i = 0; i < subtitlesArray.count; i++) {
    TXTrackInfo *info = subtitlesArray[i];
    if (info.trackIndex == 0) {
```

```
        [_txVodPlayer selectTrack:info.trackIndex];  // 选中字幕
    } else {
        // 其它字幕不需要的话，  进行deselectTrack
        [_txVodPlayer deselectTrack:info.trackIndex];
    }
}

// 监听轨道切换消息
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
    if (EvtID == VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
        int trackIndex = [(NSNumber *)[param valueForKey:EVT_KEY_SELECT_TRACK_INDEX
        int errorCode = [(NSNumber *)[param valueForKey:EVT_KEY_SELECT_TRACK_ERROR_
        NSLog(@"receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=%d , errorCo
    }
}
```

步骤3：配置字幕样式。

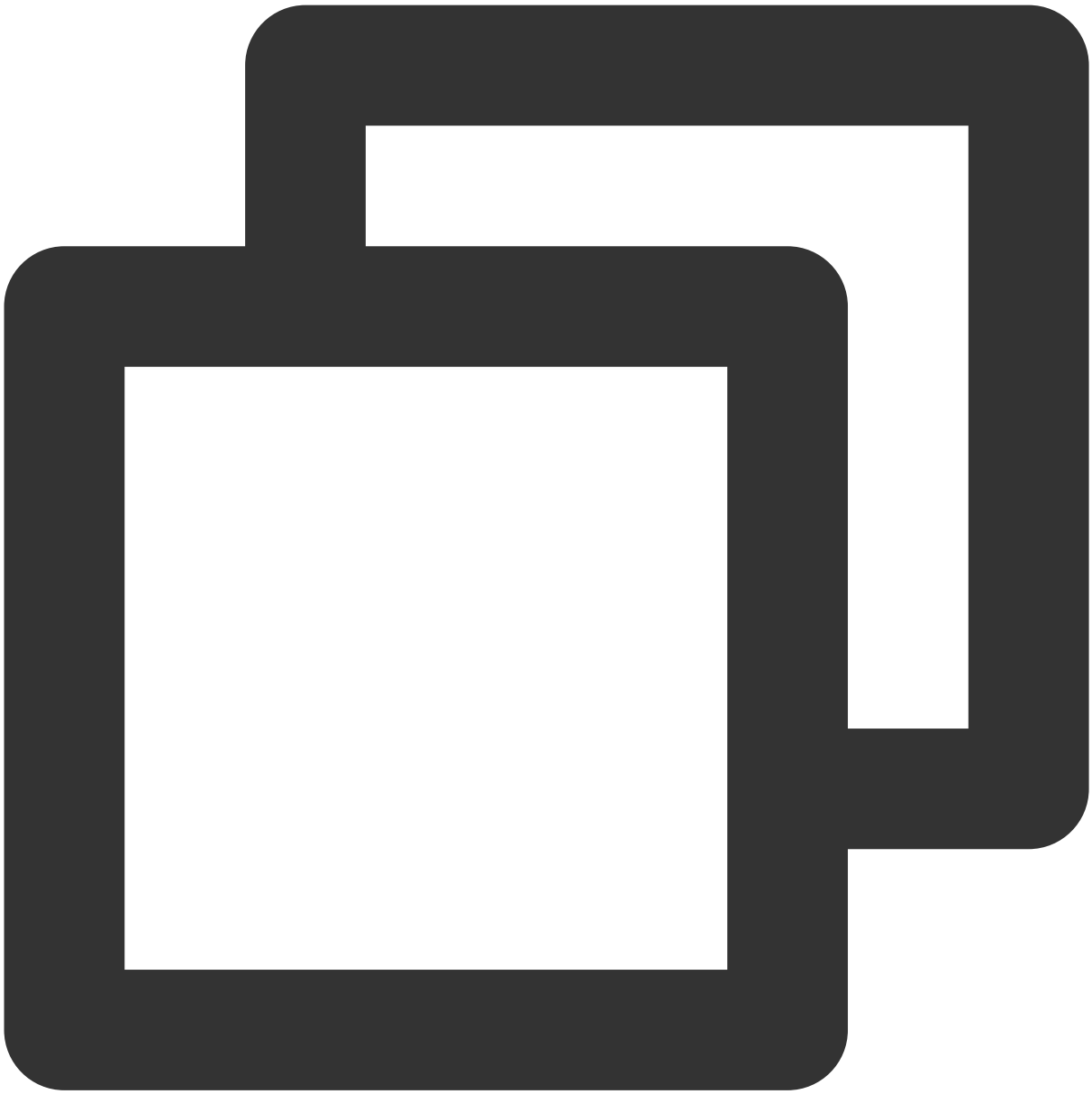
字幕样式支持在播放前或者播放过程中配置。



```
// 详细参数配置请参考 API 文档
TXPlayerSubtitleRenderModel *model = [[TXPlayerSubtitleRenderModel alloc] init];
model.canvasWidth = 1920;  // 字幕渲染画布的宽
model.canvasHeight = 1080;  // 字幕渲染画布的高
model.isBondFontStyle = NO;  // 设置字幕字体是否为粗体
model.fontColor = 0xFF000000; // 设置字幕字体颜色，默认白色不透明
[_txVodPlayer setSubtitleStyle:model];
```

21、多音轨切换

播放器高级版 SDK 支持切换视频内置的多音轨。用法如下：



```
NSArray<TXTrackInfo *> *soundTrackArray = [_txVodPlayer getAudioTrackInfo];
for (int i = 0; i < soundTrackArray.count; i++) {
    TXTrackInfo *info = soundTrackArray[i];
    if (info.trackIndex == 0) {
        // 通过判断 trackIndex 或者 name 切换到需要的音轨
        [_txVodPlayer selectTrack:info.trackIndex];
    } else {
        // 其它字幕不需要的话，  进行 deselectTrack
        [_txVodPlayer deselectTrack:info.trackIndex];
```

```
        }
    }
```

高级功能使用

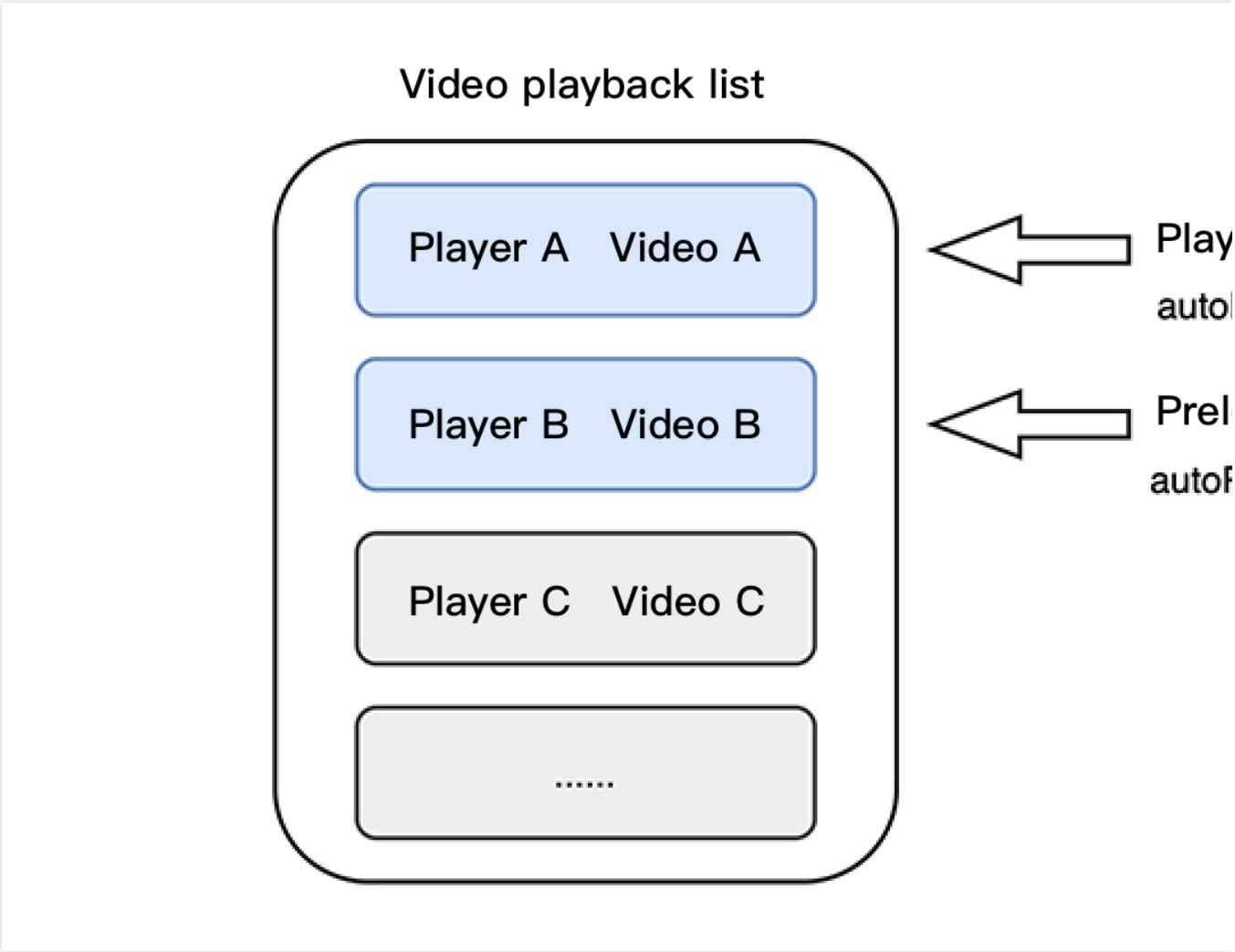
1、视频预播放

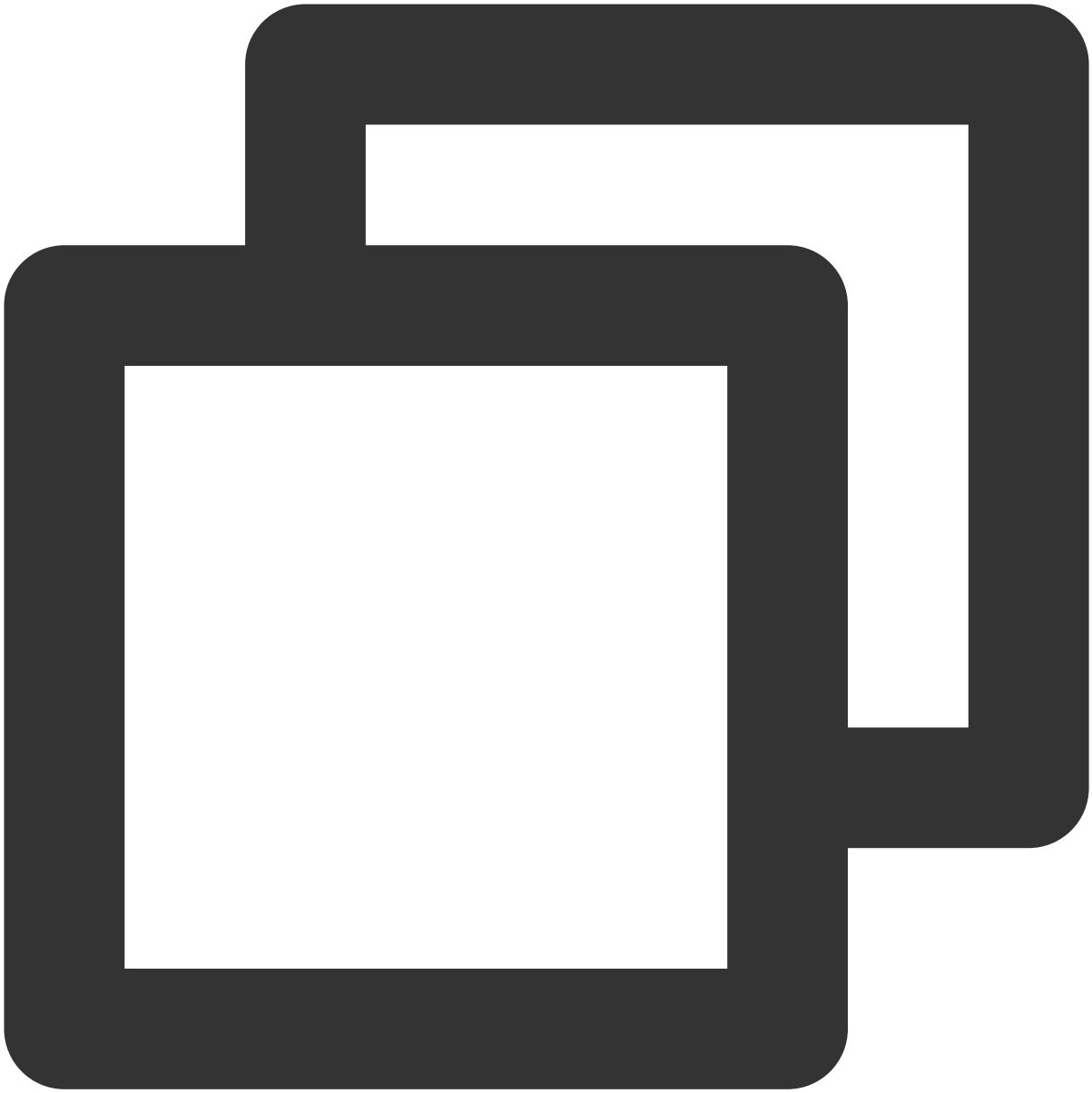
步骤1：视频预播放使用

在短视频播放场景中，预加载功能对于流畅的观看体验很有帮助：在观看当前视频的同时，在后台加载即将要播放的下一个视频 URL，这样一来，当用户真正切换到下一个视频时，已经不需要从头开始加载了，而是可以做到立刻播放。

预播放视频会有很好的秒开效果，但有一定的性能开销，如果业务同时有较多的视频预加载需求，建议结合 视频预下载 一起使用。

这就是视频播放中无缝切换的背后技术支撑，您可以使用 TXVodPlayer 中的 isAutoPlay 开关来实现这个功能，具体做法如下：


Video playback list
Player A Video A
Player B Video B
Player C Video C
.....
Play
auto
Prel
autoF

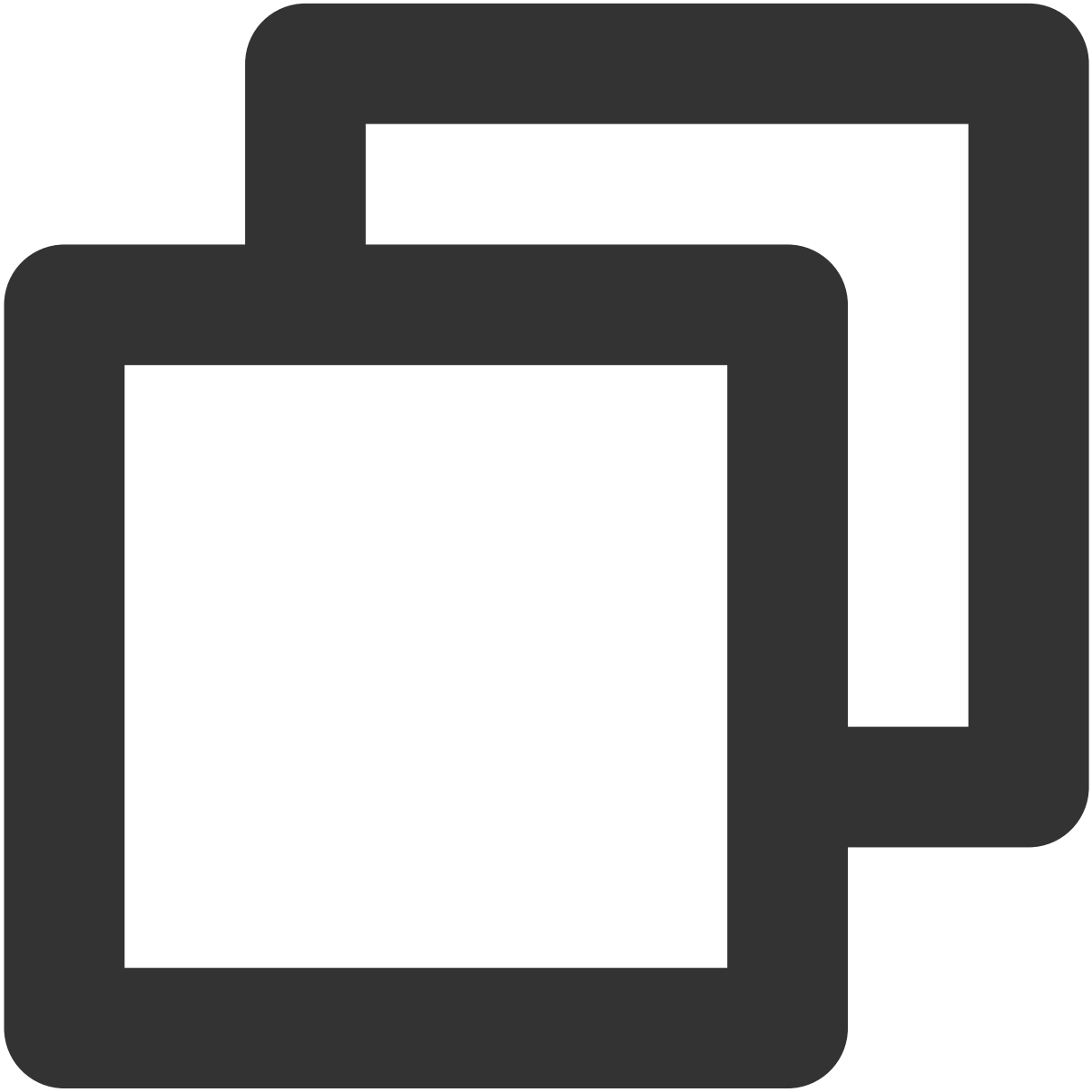


```
// 播放视频 A：如果将 isAutoPlay 设置为 YES，那么 startVodPlay调用会立刻开始视频的加载和播放
NSString* url_A = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
_player_A.isAutoPlay = YES;
[_player_A startVodPlay:url_A];

// 在播放视频 A 的同时，预加载视频 B，做法是将 isAutoPlay 设置为 NO
NSString* url_B = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
_player_B.isAutoPlay = NO;
[_player_B startVodPlay:url_B];
```

等到视频 A 播放结束，自动（或者用户手动切换到）视频 B 时，调用 resume 函数即可实现立刻播放。

注意：

设置了 autoPlay 为 false 之后，调用 resume 之前需要保证视频 B 已准备完成，即需要在监听到视频 B 的 PLAY_EVT_VOD_PLAY_PREPARED （2013，播放器已准备完成，可以播放）事件后调用。



```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
{
    // 在视频 A 播放结束的时候，直接启动视频 B 的播放，可以做到无缝切换
    if (EvtID == PLAY_EVT_PLAY_END) {
            [_player_A stopPlay];
            [_player_B setupVideoWidget:mVideoContainer insertIndex:0];
            [_player_B resume];
```

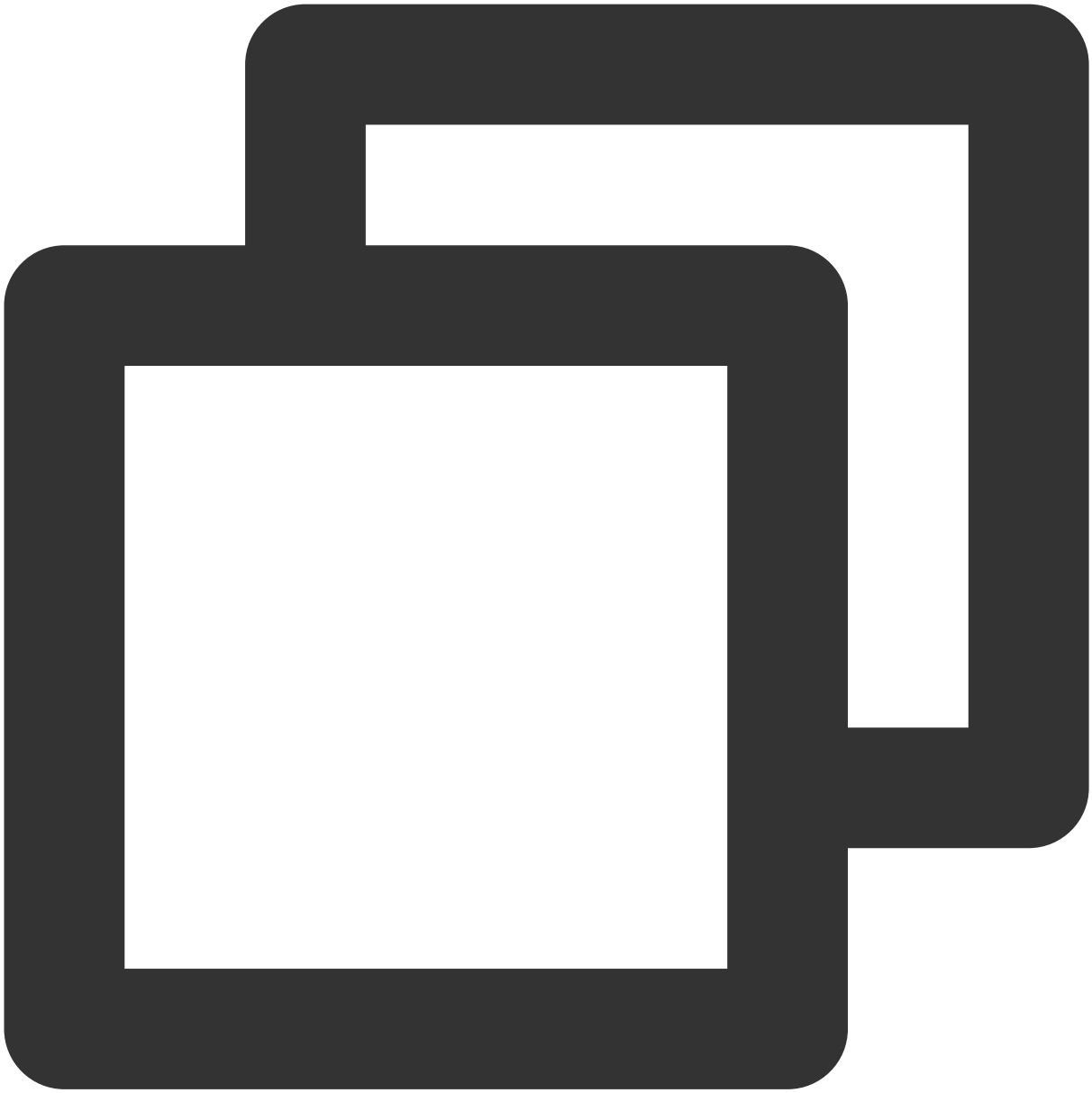
```
            }
    }
```

步骤2：视频预播放缓冲配置

设置较大的缓冲可以更好的应对网络的波动，达到流畅播放的目的。
设置较小的缓冲可以帮助节省流量消耗。

预播放缓冲大小

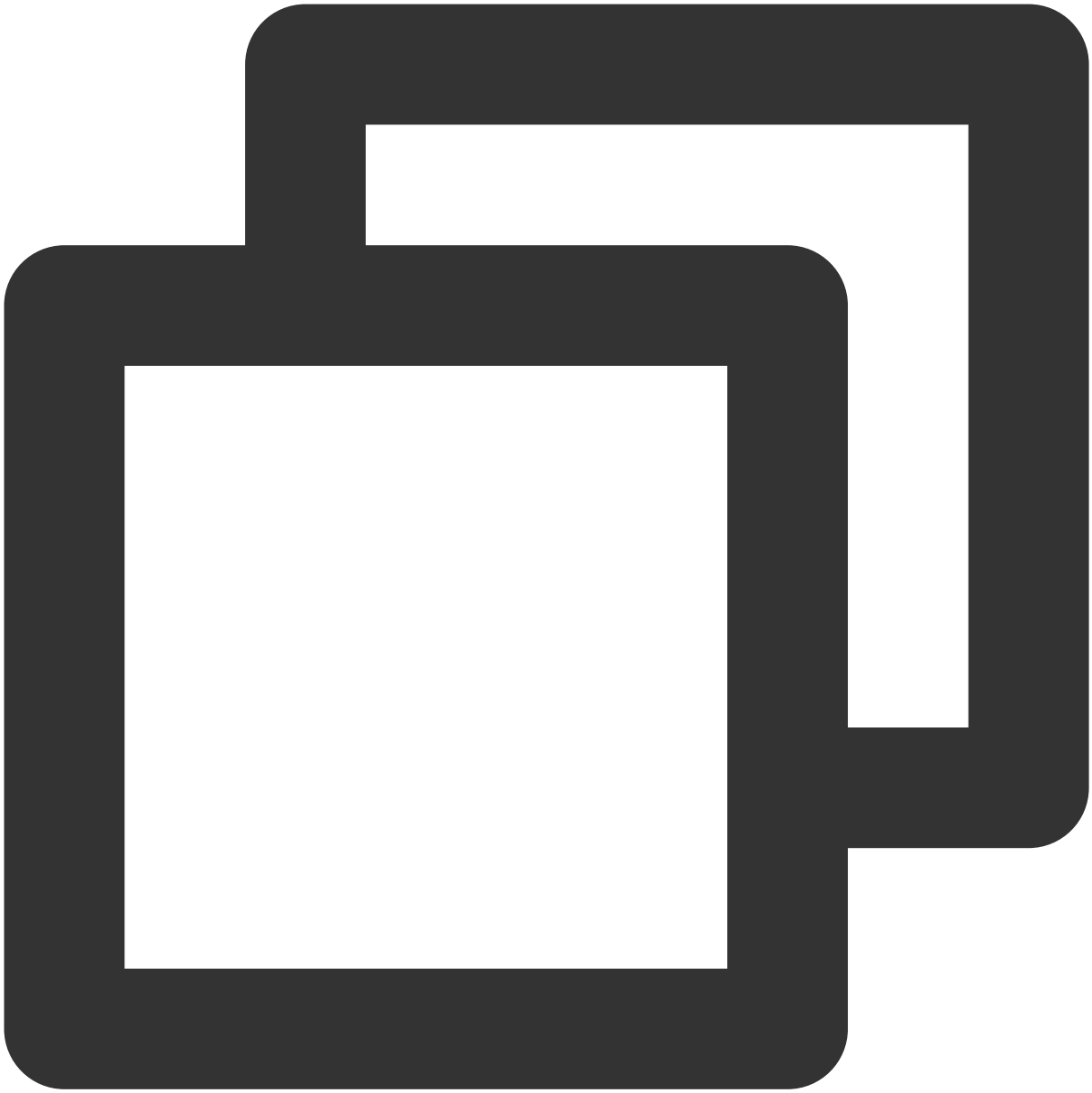
此接口针对预加载场景（即在视频启播前，且设置 player 的 AutoPlay 为 false），用于控制启播前阶段的最大缓冲大小。



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxPreloadSize:(2)];;   // 预播放最大缓冲大小。单位：MB，根据业务情况设置去节省流
[_txVodPlayer setConfig:_config];   // 把config 传给 _txVodPlayer
```

播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10];  // 播放时最大缓冲大小。单位：MB
[_txVodPlayer setConfig:_config];  // 把config 传给 _txVodPlayer
```

2、视频预下载

不需要创建播放器实例，预先下载视频部分内容，使用播放器时，可以加快视频启播速度，提供更好的播放体验。在使用播放服务前，请确保先设置好视频缓存。

说明：

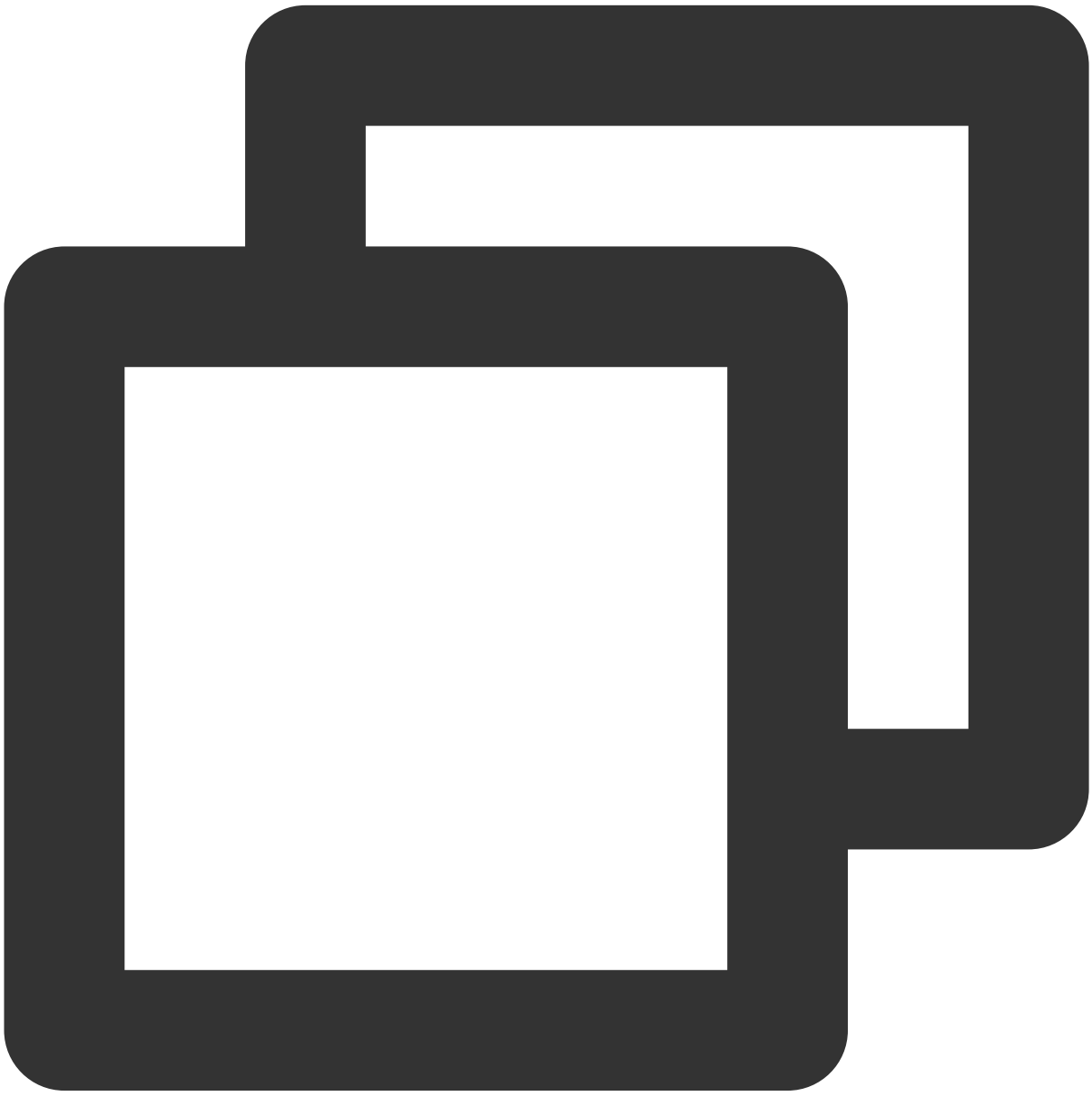
TXPlayerGlobalSetting 是全局缓存设置接口，原有 TXVodConfig 的缓存配置接口废弃。

全局缓存目录和大小设置的优先级高于播放器 TXVodConfig 配置的缓存设置。

通过媒体URL预下载

通过媒体FileId预下载

通过媒资 URL 预下载视频代码示例如下：



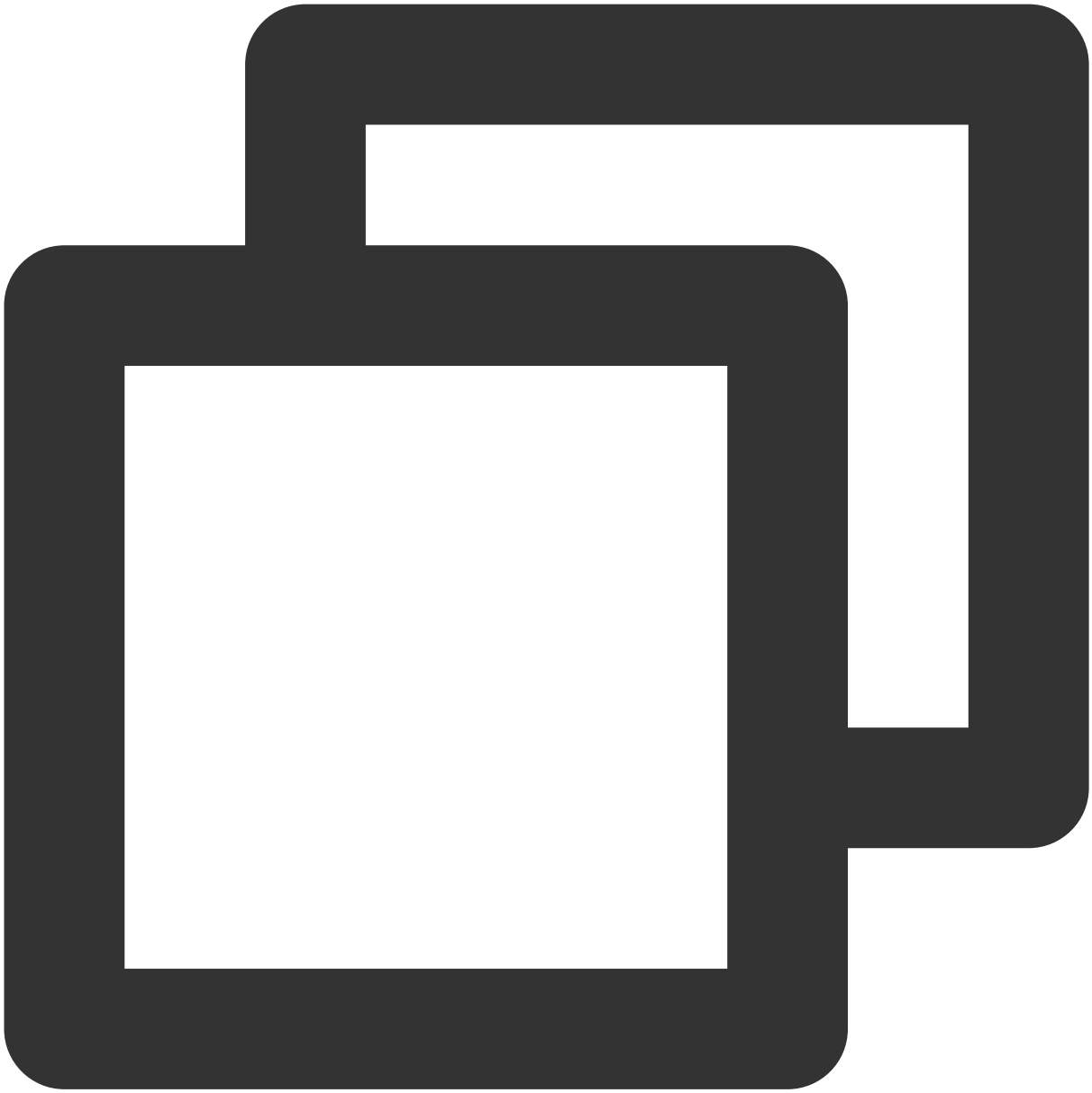
```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
```

```
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                              withIntermediateDirectories:NO
                                               attributes:nil
                                                    error:&error]; //Create folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];

//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];
NSString *m3u8url = "http://****";
int taskID = [[TXVodPreloadManager sharedManager] startPreload:m3u8url
                                                   preloadSize:10
                                          preferredResolution:1920*1080
                                                      delegate:self];


//取消预下载
[[TXVodPreloadManager sharedManager] stopPreload:taskID];
```

注意：

通过 fileId 预下载从 11.3 版本开始支持。

通过 fileId 预下载是耗时操作， 请不要在主线程调用，否则会抛出非法调用异常。startPreload 时传入的 preferredResolution 要和启播时设置的优先启播分辨率保持一致，否则将达不到预期的效果。 使用示例如下：



```
//设置播放引擎的全局缓存目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                              withIntermediateDirectories:NO
                                               attributes:nil
                                                    error:&error]; //Create folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
```

```
//设置播放引擎缓存大小
[TXPlayerGlobalSetting setMaxCacheSize:200];

TXPlayerAuthParams *params = [[TXPlayerAuthParams alloc] init];
params.appId = ${appId};
params.fileId = @"${fileId}";
params.sign = @"${psign}";
// 注意： 耗时操作，请不要在主线程调用！在主线程调用将会抛出非法调用异常。
int taskID = [[TXVodPreloadManager sharedManager] startPreload:params
preloadSize:10
preferredResolution:1920*1080
delegate:self]; // TXVodPreloadManagerDelegate

//取消预下载
[[TXVodPreloadManager sharedManager] stopPreload:taskID];
```

3、视频下载

视频下载支持用户在有网络的条件下下载视频，随后在无网络的环境下观看。如果是加密视频，通过播放器 SDK 下载后的视频在本地保持为加密状态，仅可通过腾讯云播放器 SDK 进行解密播放，可有效防止下载后视频的非法传播，保护视频安全。

由于 HLS 流媒体无法直接保存到本地，因此也无法通过播放本地文件的方式实现 HLS 离线播放，对于该问题，您可以通过基于 `TXVodDownloadManager` 的视频下载方案实现 HLS 的离线播放。

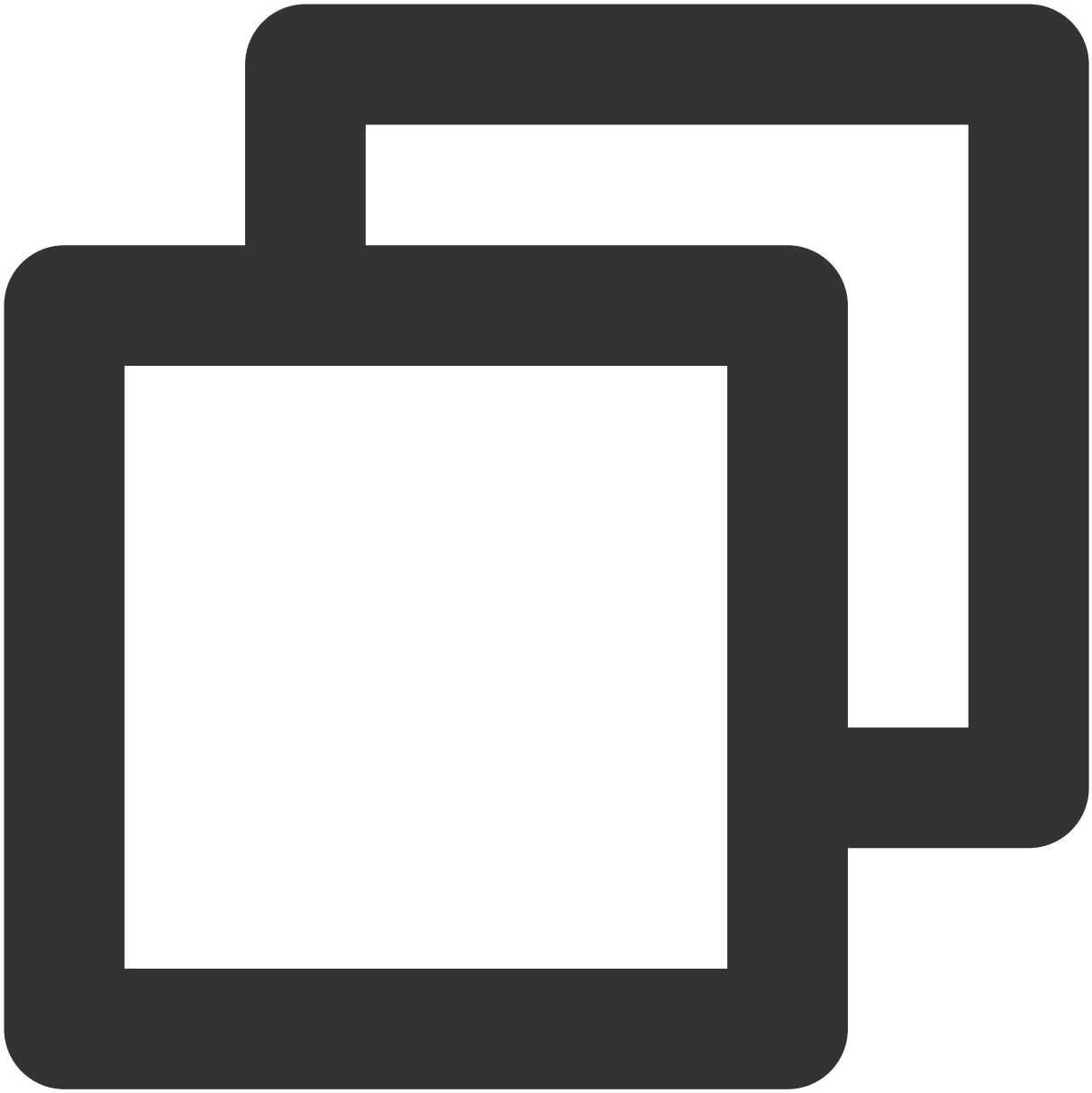
注意：

`TXVodDownloadManager` 暂不支持缓存 MP4 。

播放器 SDK 已支持 MP4 和 FLV 格式的本地文件播放。

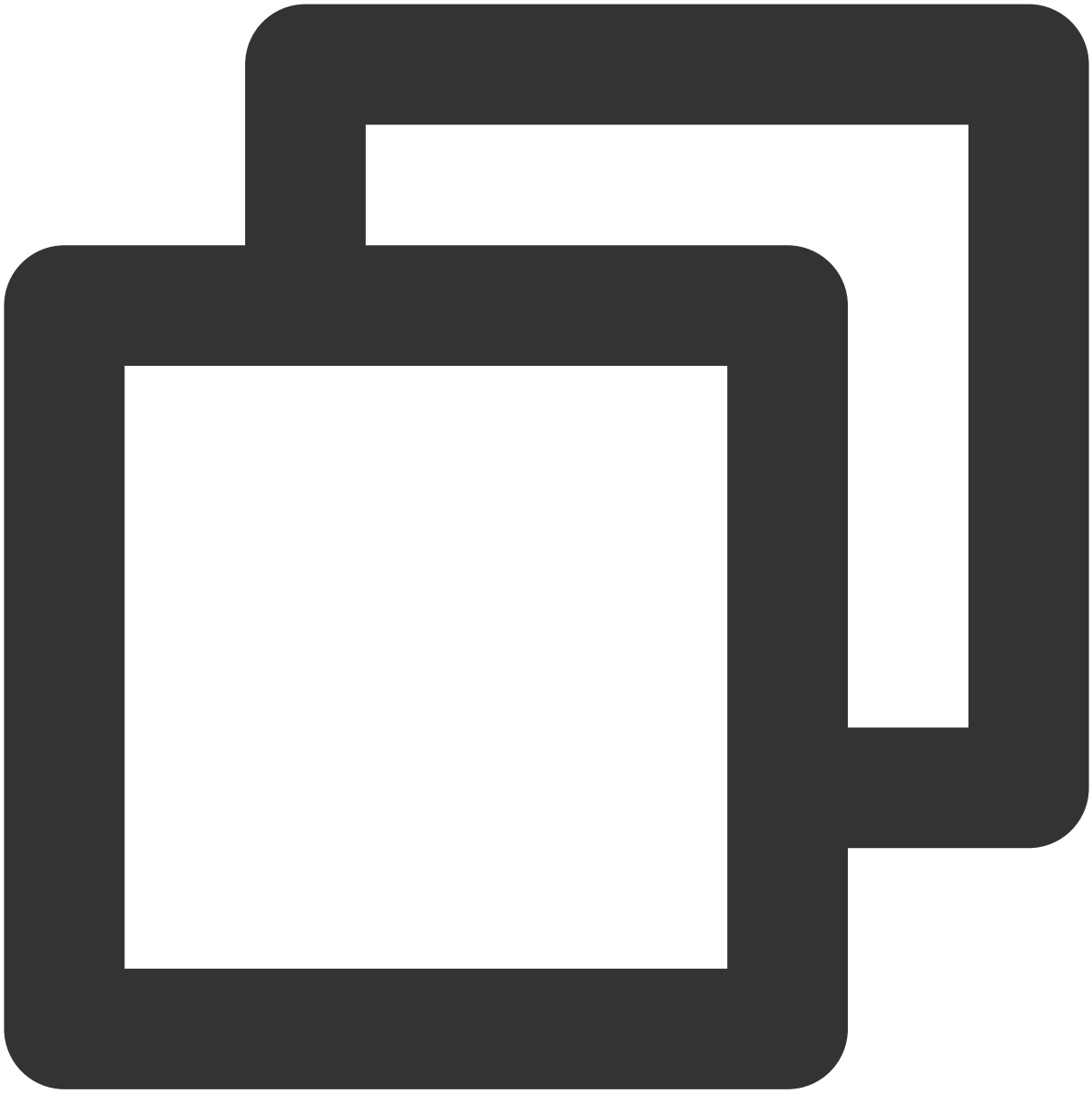
步骤1：准备工作

SDK 初始化时，设置全局存储路径，用于视频下载，预加载，和缓存等功能。用法如下：



```
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSU
NSString downloadPath = [NSString stringWithFormat:@"%@/txdownload",cachesDir];
[TXPlayerGlobalSetting setCacheFolderPath:downloadPath];
```

`TXVodDownloadManager` 被设计为单例，因此您不能创建多个下载对象。用法如下：



```
TXVodDownloadManager *downloader = [TXVodDownloadManager shareInstance];
```

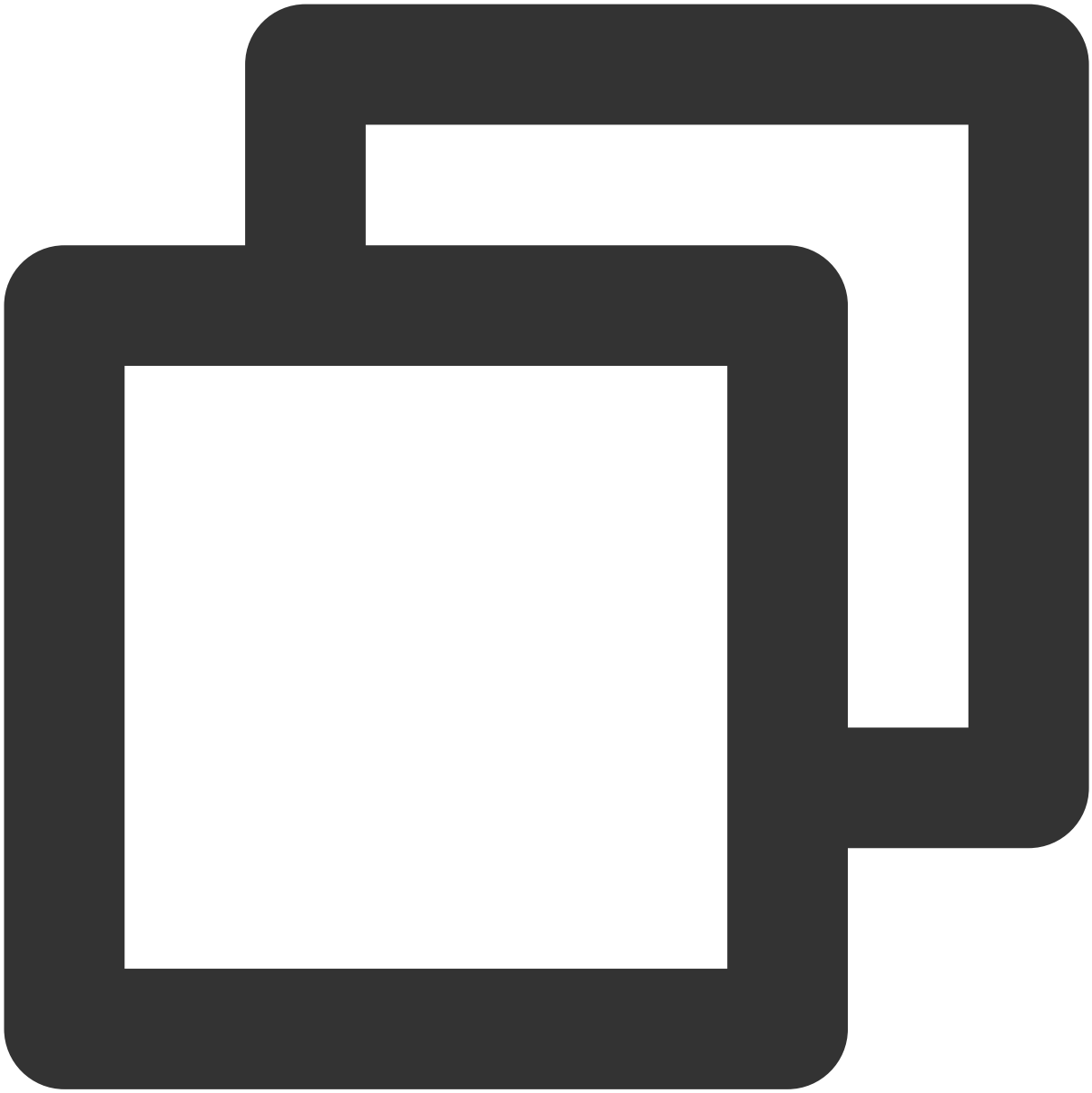
步骤2：开始下载

开始下载有两种方式：fileid 和 URL。

fileid 方式

URL 方式

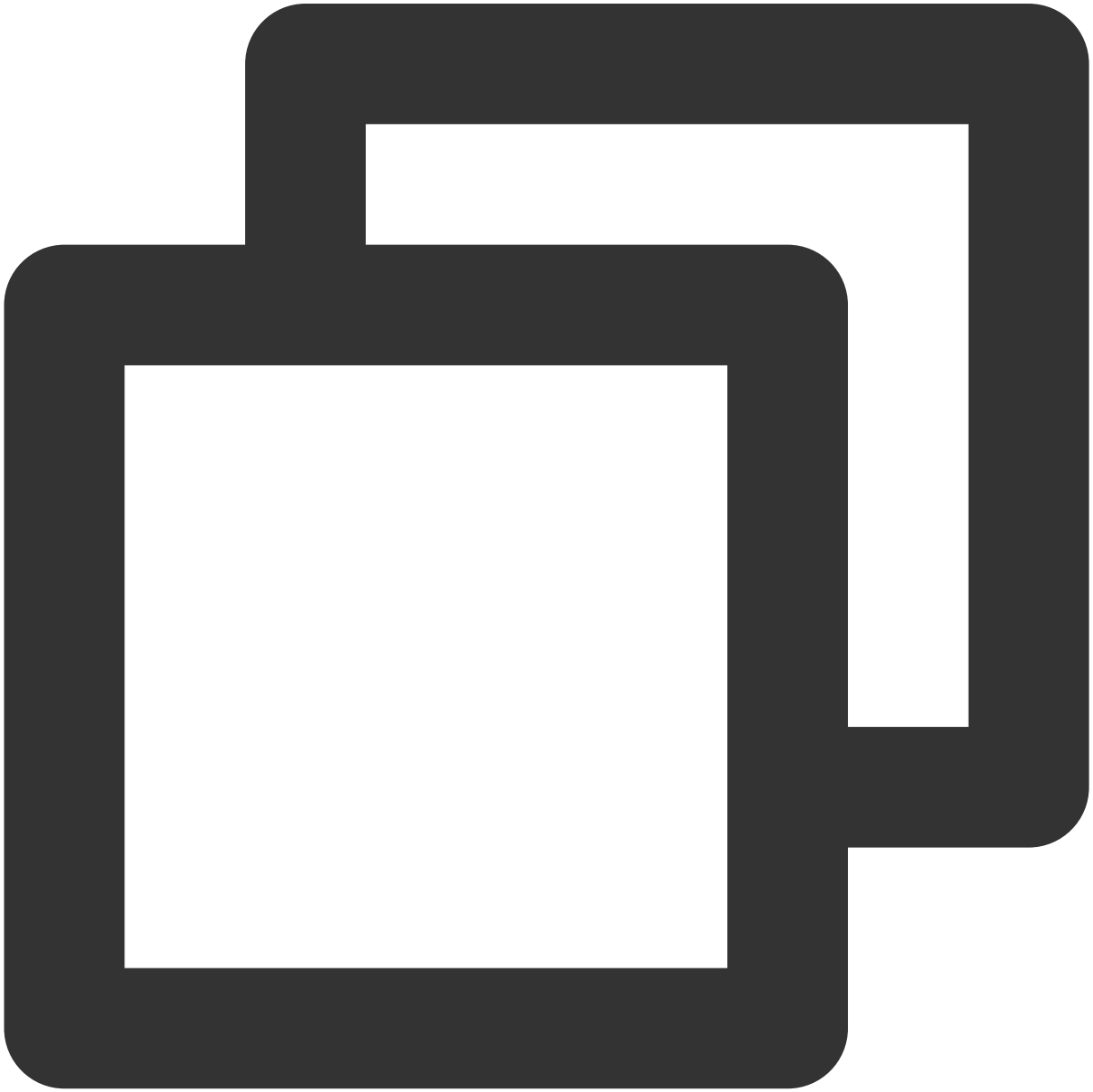
fileid 下载至少需要传入 appId 和 fileId，userName 不传入具体值时，默认为“default”。**注意：加密视频只能通过 Fileid 下载。**



```
TXVodDownloadDataSource *source = [[TXVodDownloadDataSource alloc] init];
source.appId = 1252463788;
source.fileId = @"4564972819220421305";
// // psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/documen
source.pSign = @"xxxxxxxxxx";

// 指定下载清晰度
// 可用枚举值有：240p:TXVodQuality240P, 360p:TXVodQuality360P, 480p:TXVodQuality480P,
// 1080p:TXVodQuality1080P,2K:TXVodQuality2K,4K:TXVodQuality4K
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720，期望下载此分辨率的流，quality
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
```

```
source.quality = TXVodQualityHD;  // 高清

// ** 注意如果是使用旧的v2协议下载，请通过TXVodDownloadDataSource中的 auth 属性来设置appId
// source.auth = auth;  ** 默认无需设置 **

[downloader startDownload:dataSource];
```

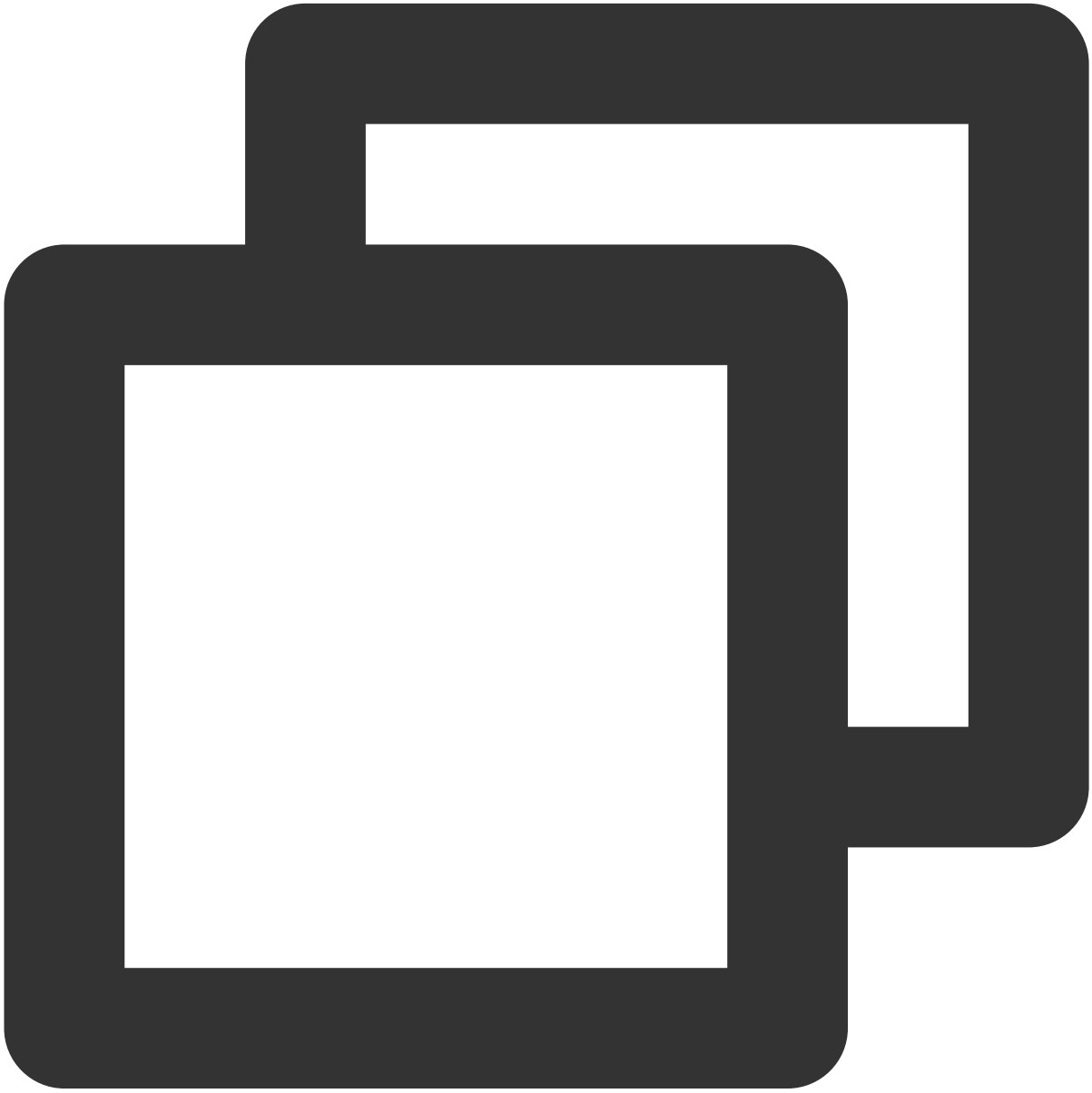
至少需要传入下载地址 URL。preferredResolution 取值为视频分辨率宽和高的乘积：preferredResolution=width * height。如果是嵌套 HLS 格式，preferredResolution 不传入具体值时，默认值为921600。userName 不传入具体值时，默认为"default"。私有加密请使用 fileid 形式。



```
[downloader startDownloadUrl:@"" resolution:@921600 userName:@""];
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 delegate。



```
downloader.delegate = self;
```

可能收到的任务回调有：

回调信息	含义

-[TXVodDownloadDelegate onDownloadStart:]	任务开始，表示 SDK 已经开始下载
-[TXVodDownloadDelegate onDownloadProgress:]	任务进度，下载过程中，SDK 会频繁回调此接口，您可以在这里更新进度显示
-[TXVodDownloadDelegate onDownloadStop:]	任务停止，当您调用是stopDownload停止下载，收到此消息表示停止成功
-[TXVodDownloadDelegate onDownloadFinish:]	下载完成，收到此回调表示已全部下载。此时下载文件可以给 TXVodPlayer 播放
-[TXVodDownloadDelegate onDownloadError:errorMsg:]	下载错误，下载过程中遇到网络断开会回调此接口，同时下载任务停止。所有错误码请参考TXDownloadError

下载错误码

错误码	数值	含义说明
TXDownloadSuccess	0	下载成功。
TXDownloadAuthFaild	-5001	向云点播控制台请求视频信息失败，建议检查fileId、psign参数是否正确。
TXDownloadNoFile	-5003	无此清晰度文件。
TXDownloadFormatError	-5004	下载文件格式不支持。
TXDownloadDisconnet	-5005	网络断开，建议检查网络是否正常。
TXDownloadHlsKeyError	-5006	获取 HLS 解密 Key 失败。
TXDownloadPathError	-5007	下载目录访问失败，建议检查是否有访问下载目录的权限。

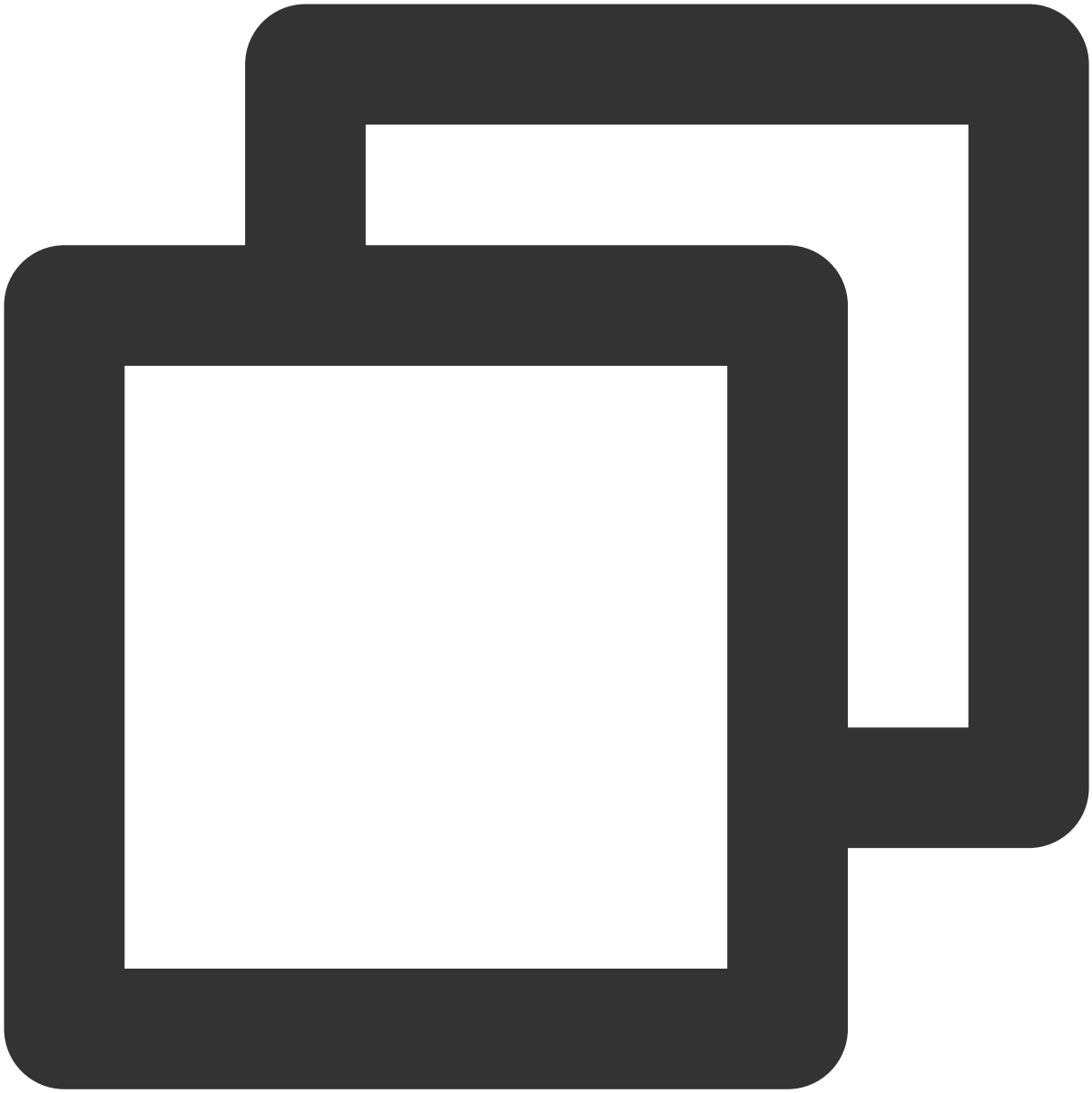
由于 downloader 可以同时下载多个任务，所以回调接口里带上了 `TXVodDownloadMediaInfo` 对象，您可以访问 URL 或 dataSource 判断下载源，同时还可以获取到下载进度、文件大小等信息。

步骤4：中断下载

停止下载请调用 `-[TXVodDownloadManager stopDownload:]` 方法，参数为 `-[TXVodDownloadManager sartDownloadUrl:]` 返回的对象。**SDK 支持断点续传**，当下载目录没有发生改变时，下次下载同一个文件时会从上次停止的地方重新开始。

步骤5：管理下载

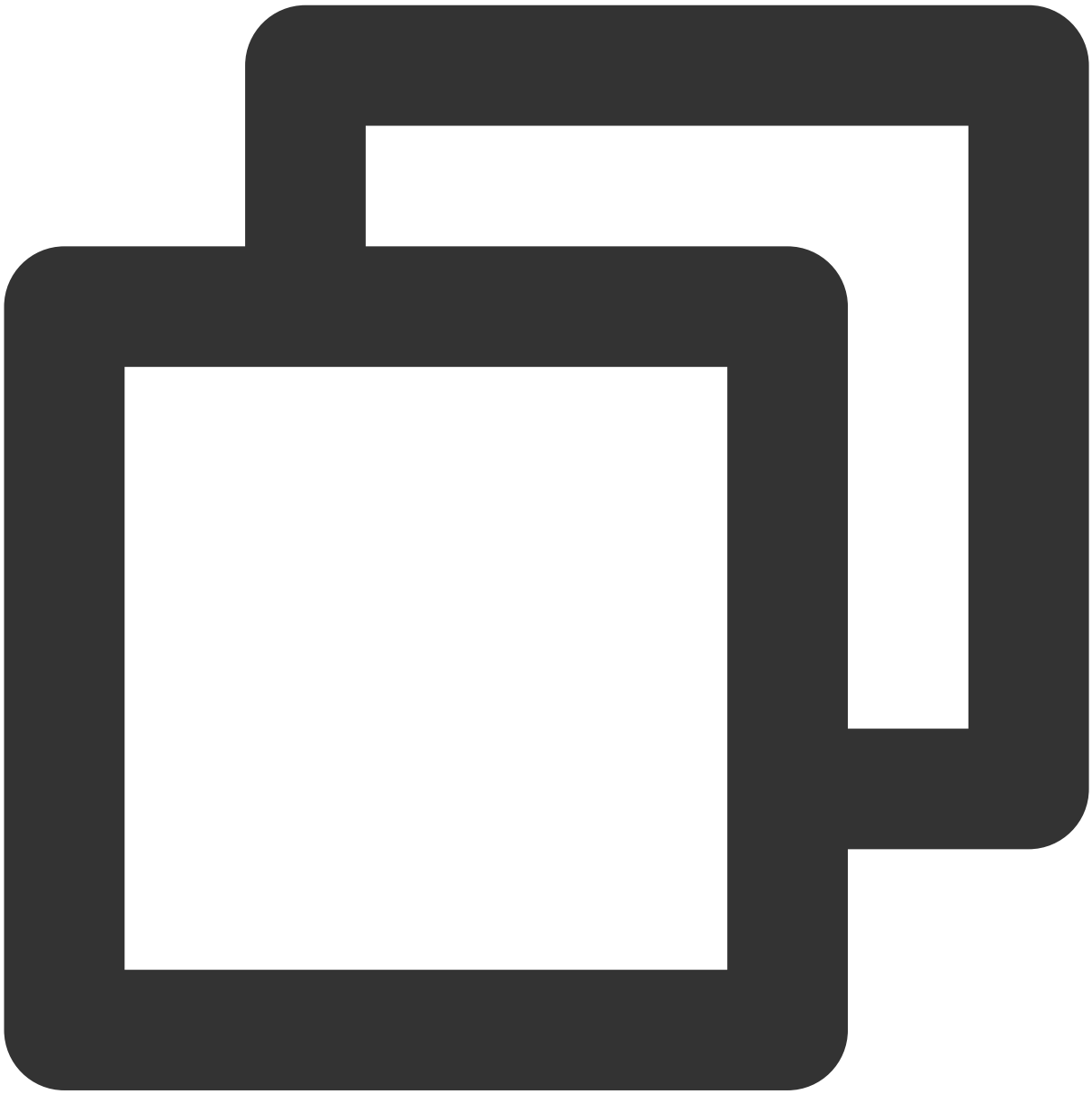
1. 获取所有用户账户的下载列表信息, 也可获取指定用户账户的下载列表信息。



```
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
NSArray<TXVodDownloadMediaInfo *> *array = [[[TXVodDownloadManager shareInstance] g
// 获取默认“default”用户的下载列表
for (TXVodDownloadMediaInfo *info in array) {
    if ([info.userName isEqualToString:@"default"]) {
        // 保存“default”用户的下载列表
    }
}
```

2. 获取 FileId 或 URL 相关的下载信息：

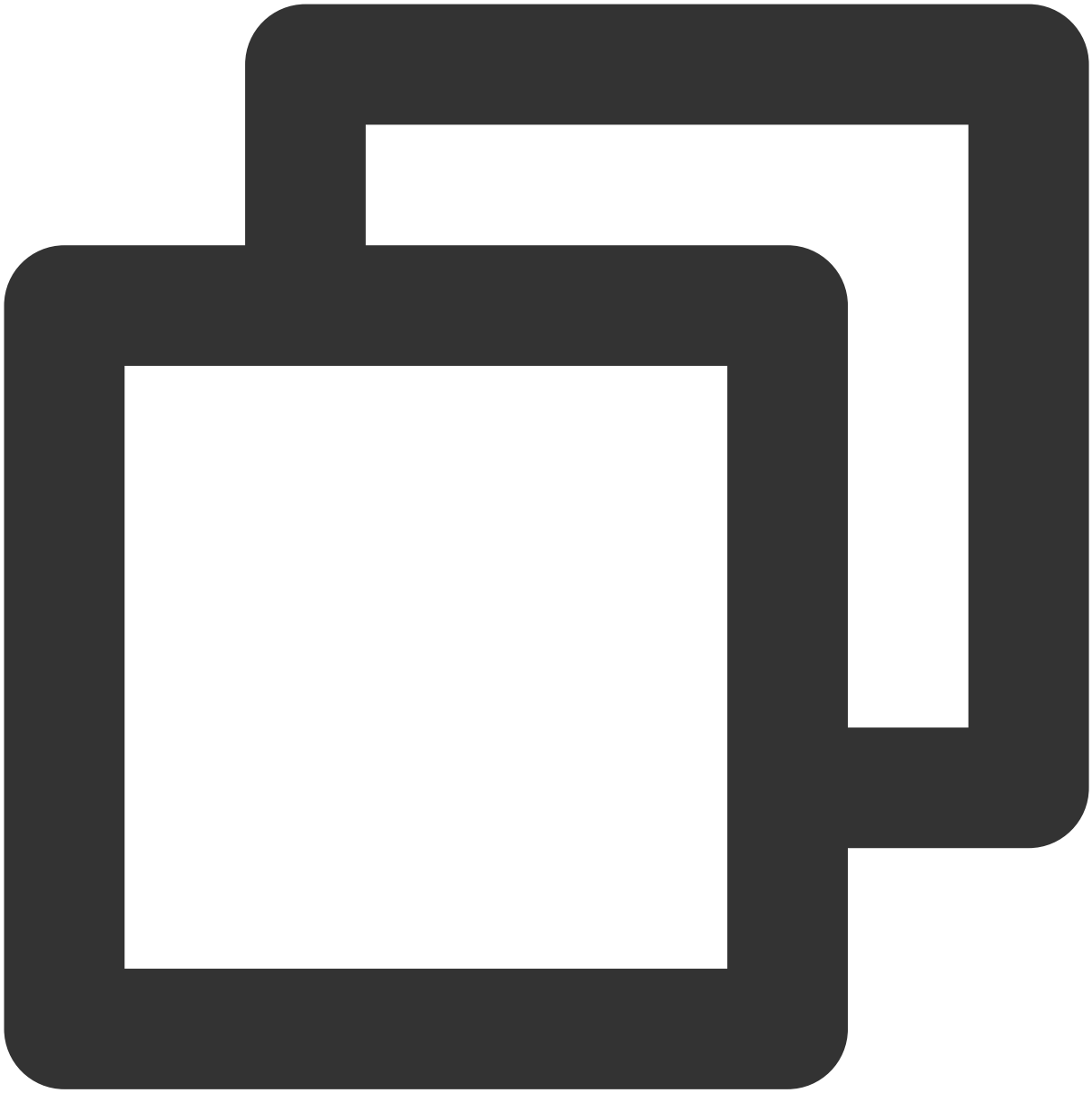
2.1. 通过接口 `-[TXVodDownloadManager getDownloadMediaInfo:]` 获取某个 Fileid 相关下载信息，包括当前下载状态，获取当前下载进度，判断是否下载完成等，需要传入 AppID、 Fileid 和 qualityId。



```
// 获取某个fileId相关下载信息
TXVodDownloadMediaInfo *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc] init];
TXVodDownloadDataSource *dataSource = [[TXVodDownloadDataSource alloc] init];
dataSource.appId = 1252463788;
dataSource.fileId = @"4564972819220421305";
dataSource.pSign = @"psignxxxx";
dataSource.quality = TXVodQualityHD;
sourceMediaInfo.dataSource = dataSource;
```

```
TXVodDownloadMediaInfo *downlaodMediaInfo =  [[TXVodDownloadManager shareInstance]

// 获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。
// 备注：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大小，暂时无法获取。
downlaodMediaInfo.size; //  获取下载文件总大小
downlaodMediaInfo.duration;    // 获取总时长
downlaodMediaInfo.playableDuration; // 获取已下载的可播放时长
downlaodMediaInfo.progress;    // 获取下载进度
downlaodMediaInfo.playPath;    // 获取离线播放路径，传给播放器即可离线播放
downlaodMediaInfo.downloadState;  // 获取下载状态，具体参考STATE_xxx常量
[downlaodMediaInfo isDownloadFinished];  // 返回YES表示下载完成
```

2.2. 获取某个 URL 相关下载信息，需要传入 URL 信息即可。



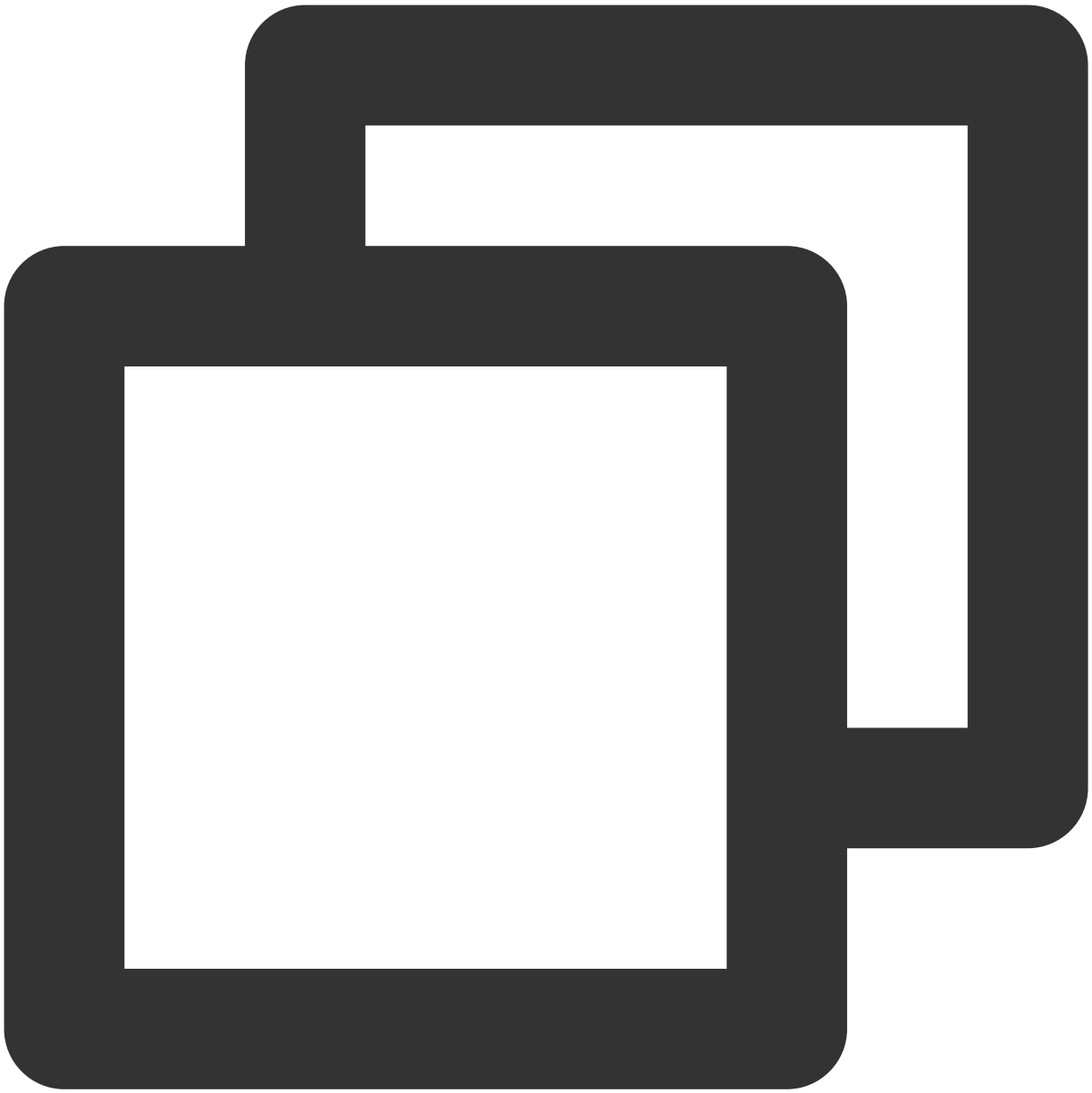
```
// 获取某个fileId相关下载信息
TXVodDownloadMediaInfo *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc] init];
mediaInfo.url = @"videoURL";
TXVodDownloadMediaInfo *downlaodMediaInfo =  [[TXVodDownloadManager shareInstance]
```

3. 删除下载信息和相关文件：

如果您不需要重新下载，请调用 `-[TXVodDownloadManager deleteDownloadFile:]` 方法删除文件，以释放存储空间。

步骤6：下载后离线播放

下载后的视频支持无网络的情况下进行播放，无需进行联网。下载完成后，即可进行播放。



```
NSArray<TXVodDownloadMediaInfo *> *mediaInfoList = [[TXVodDownloadManager shareInst
TXVodDownloadMediaInfo *mediaInfo = [mediaInfoList firstObject]; // 根据情况找到当前的
if (mediaInfo.downloadState == TXVodDownloadMediaInfoStateFinish) { // 判断是否下载完
    [self.player startVodPlay:mediaInfo.playPath];
}
```

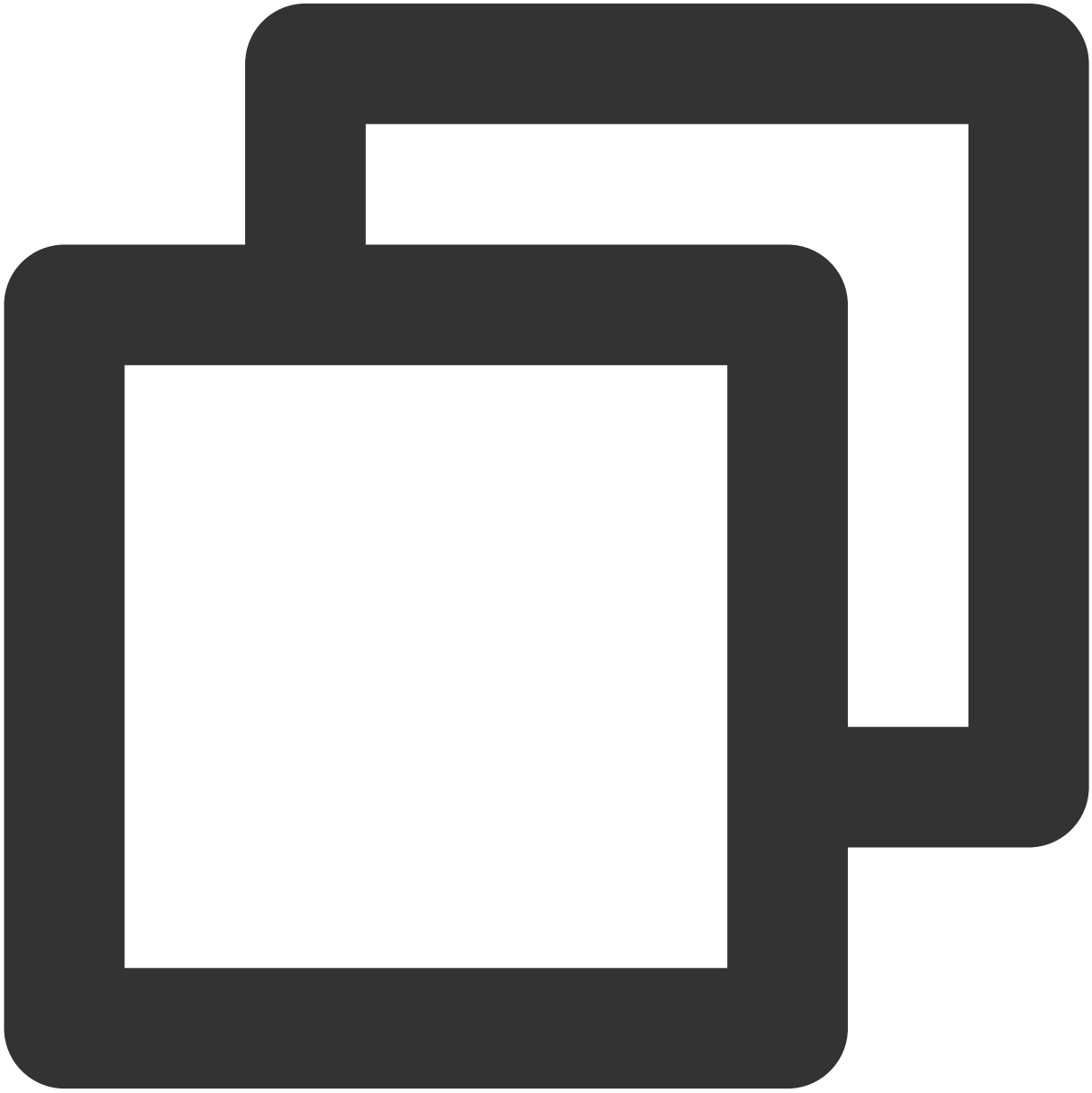
注意：

离线下载播放时，一定要通过获取下载列表并通过下载列表视频对象 TXVodDownloadMediaInfo 的 PlayPath 进行播放，切勿直接保存 PlayPath 对象。

4、加密播放

视频加密方案主要用于在线教育等需要对视频版权进行保护的场景。如果要对您的视频资源进行加密保护，就不仅需要在播放器上做改造，还需要对视频源本身进行加密转码，亦需要您的后台和终端研发工程师都参与其中。在 视频加密解决方案 中您会了解到全部细节内容。

在腾讯云控制台提取到 appId， 加密视频的 fileId 和 psign 后，可以通过下面的方式进行播放：

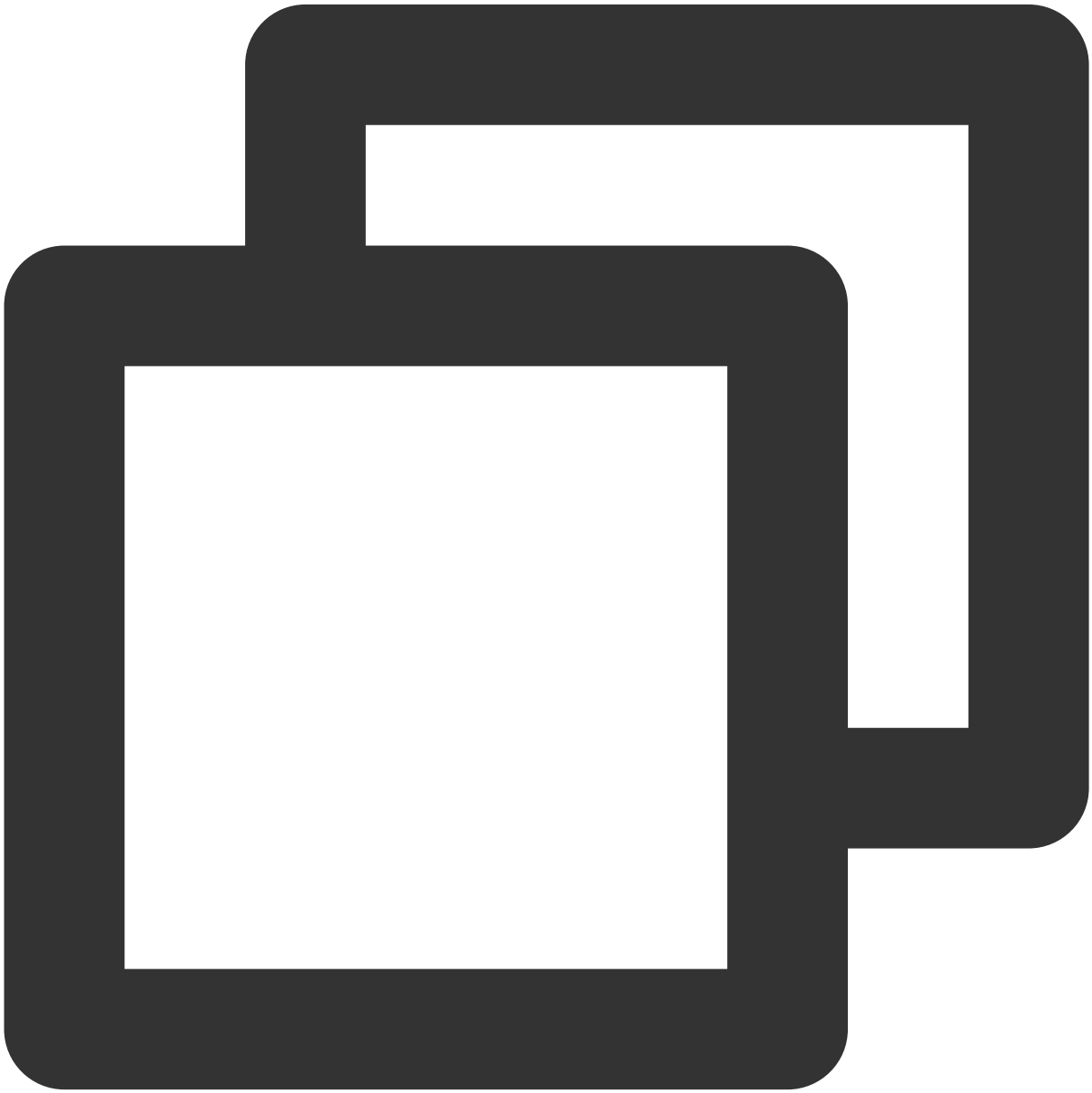


```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788; // 腾讯云账户的appId
p.fileId = @"4564972819220421305"; // 视频的fileId
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
```

```
p.sign = @"psignxxxxx"; // 播放器签名
[_txVodPlayer startVodPlayWithParams:p];
```

5、播放器配置

在调用 statPlay 之前可以通过 setConfig 对播放器进行参数配置，比如：设置播放器连接超时时间、设置进度回调间隔、设置缓存文件个数等配置，TXVodPlayConfig 支持配置的详细参数请单击 基础配置接口 了解。使用示例：

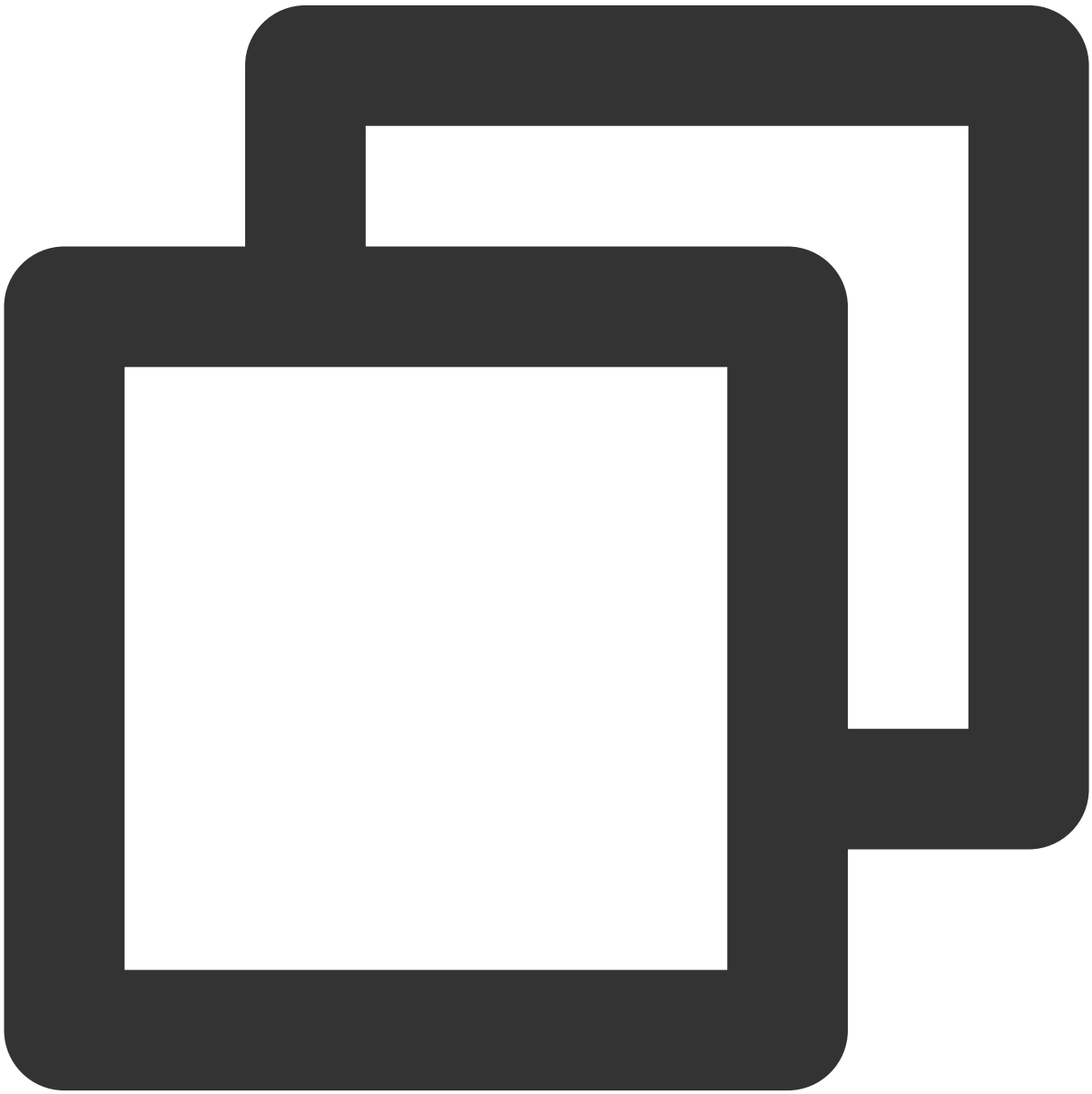


```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setEnableAccurateSeek:true];  // 设置是否精确 seek，默认 true
[_config setMaxCacheItems:5];  // 设置缓存文件个数为5
```

```
[_config setProgressInterval:200];  // 设置进度回调间隔，单位毫秒
[_config setMaxBufferSize:50];  // 最大预加载大小，单位 MB
[_txVodPlayer setConfig:_config];  // 把config 传给 _txVodPlayer
```

启播时指定分辨率

播放 HLS 的多码率视频源，如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率。 播放器会查找小于或等于偏好分辨率的流进行启播，启播后没有必要再通过 setBitrateIndex 切换到需要的码流。



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
// 传入参数为视频宽和高的乘积(宽 *  高)，可以自定义值传入
```

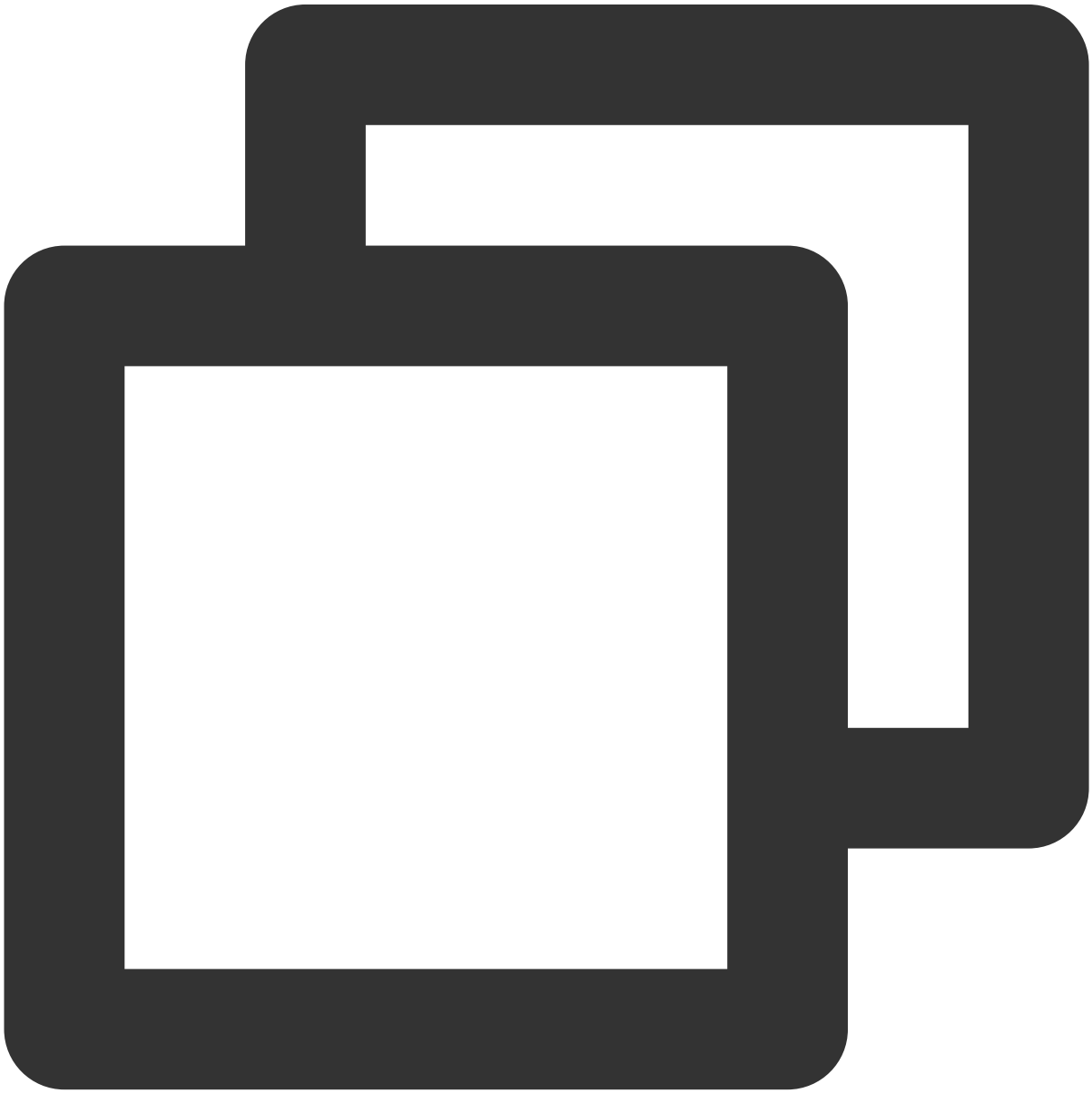
```
[_config setPreferredResolution:720*1280];
[_txVodPlayer setConfig:_config];  // 把config 传给 _txVodPlayer
```

启播前指定媒体类型

当提前知道播放的媒资类型时，可以通过配置TXVodPlayConfig#setMediaType减少播放器SDK内部播放类型探测，提升启播速度。

注意：

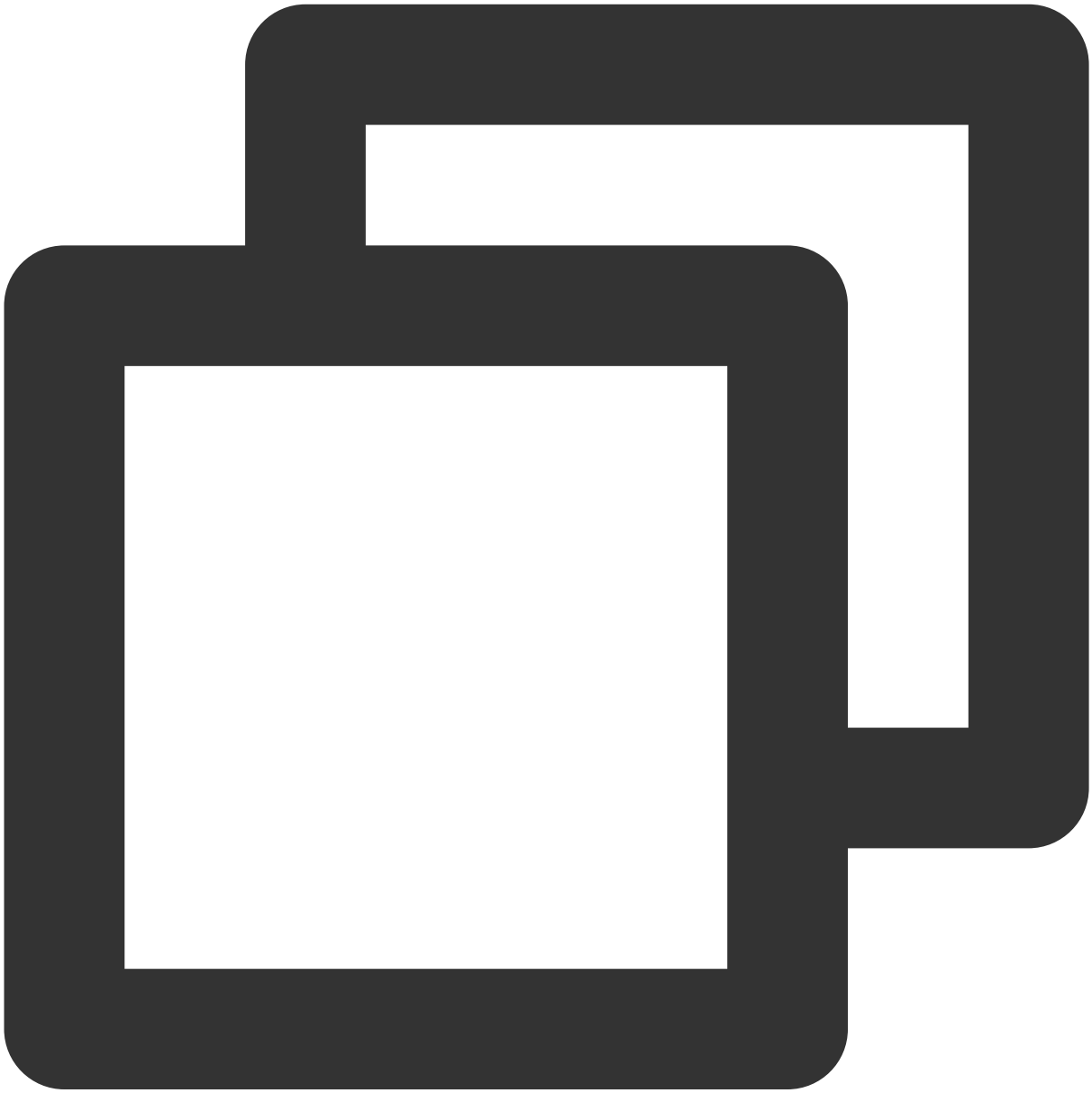
TXVodPlayConfig#setMediaType11.2 版本开始支持



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
```

```
[_config setMediaType:MEDIA_TYPE_FILE_VOD];  // 用于提升MP4启播速度
// [_config setMediaType:MEDIA_TYPE_HLS_VOD]; // 用于提升HLS启播速度
[_txVodPlayer setConfig:_config];
```

设置播放进度回调时间间隔



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setProgressInterval:200];  // 设置进度回调间隔，单位毫秒
[_txVodPlayer setConfig:_config];  // 把config 传给 _txVodPlayer
```

6、HttpDNS 解析服务

移动解析（HTTPDNS）基于 HTTP 协议向 DNS 服务器发送域名解析请求，替代了基于 DNS 协议向运营商 Local DNS 发起解析请求的传统方式，可避免 Local DNS 造成域名劫持和跨网访问问题，解决移动互联网服务中域名解析异常带来的视频播放失败困扰。

注意：

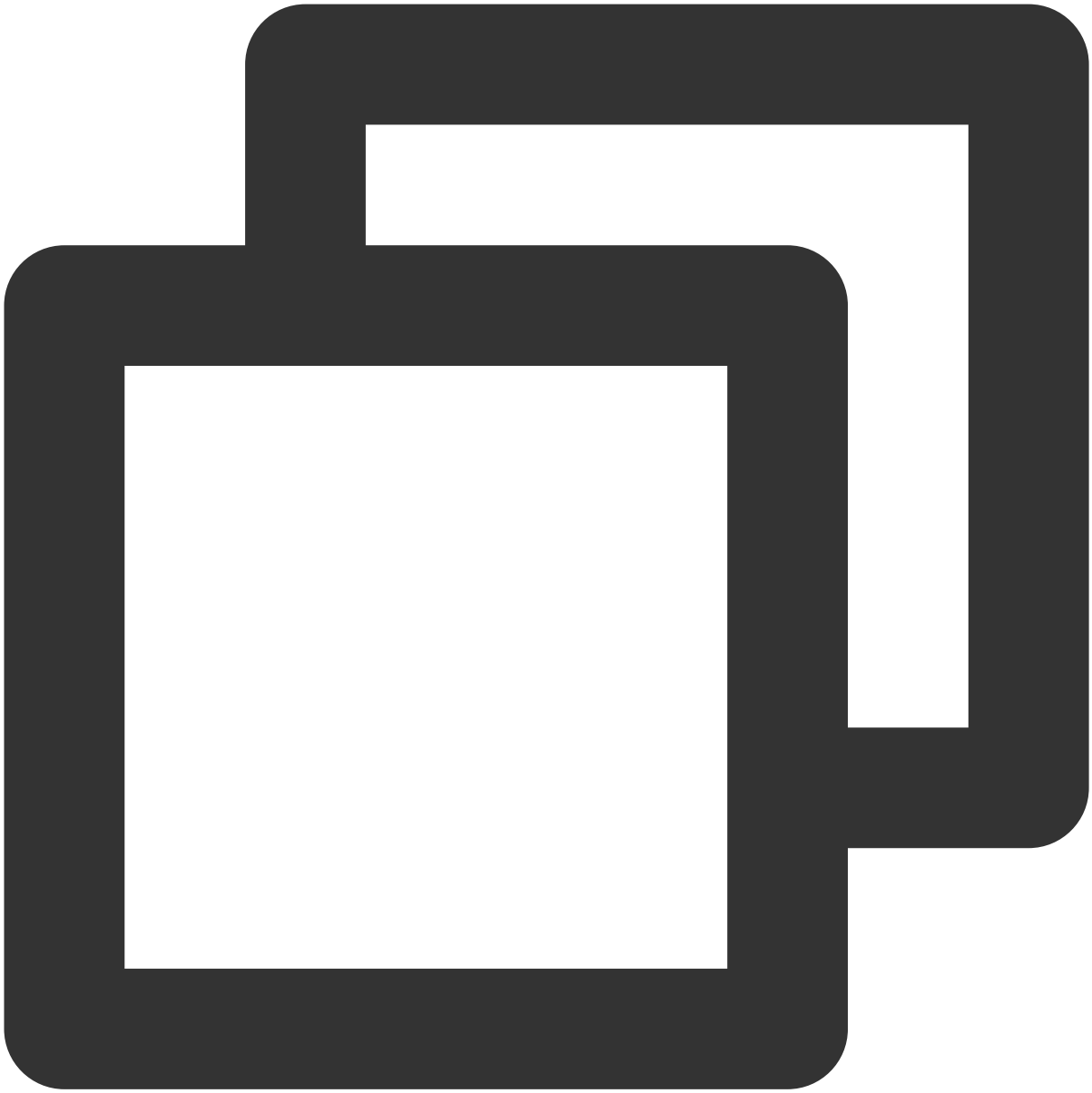
HttpDNS 解析服务从 10.9 版本开始支持。

1. 开通 HTTPDNS 解析服务

您可以选择腾讯云或其它云提供商，开通 HTTPDNS 解析服务，确保开通成功后， 再集成到播放 SDK。

2. 在播放 SDK 接入 HTTPDNS 解析服务

下面以接入 腾讯云 HTTPDNS 为例子，展示如何在播放器 SDK 接入：

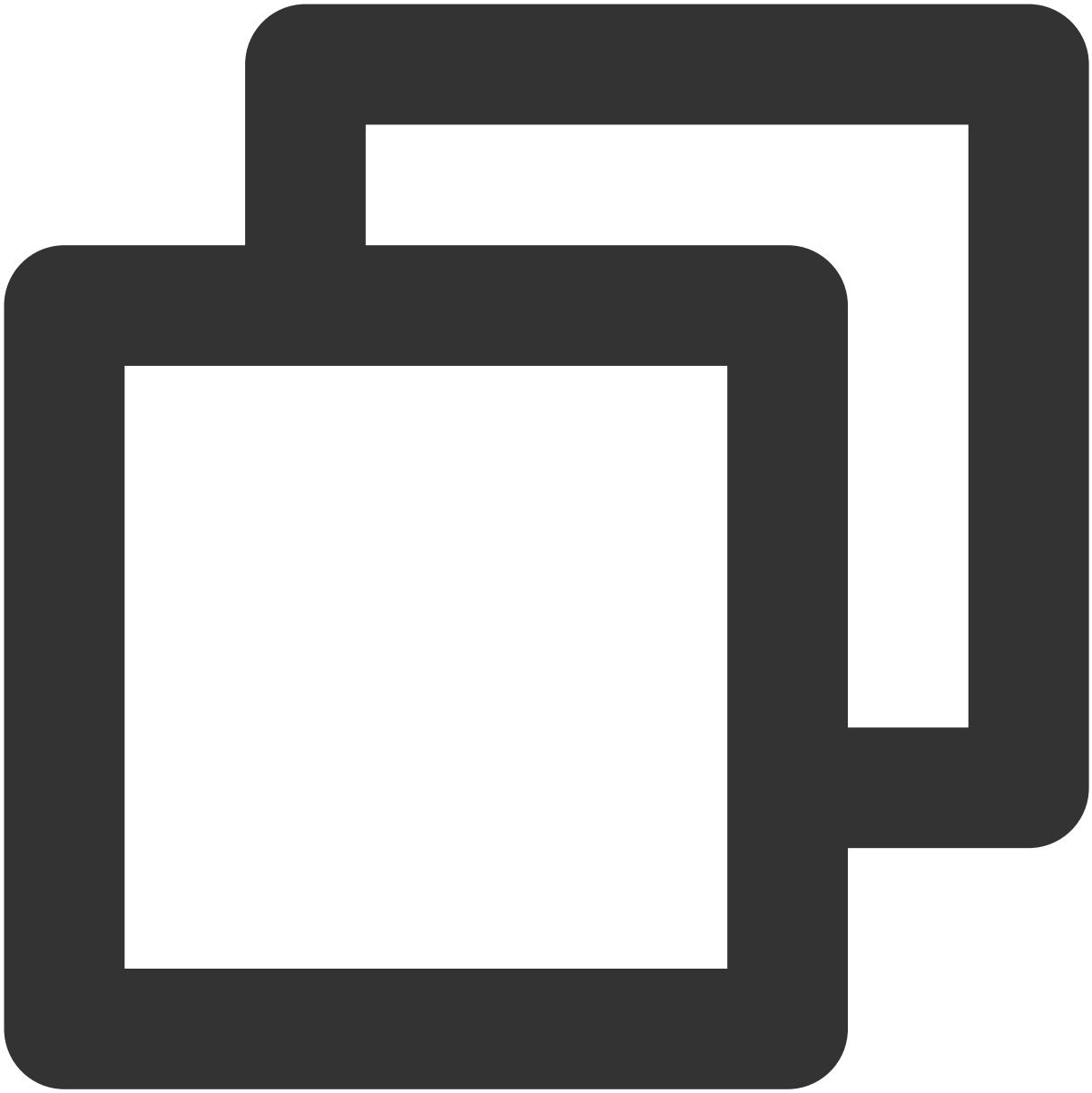


```
// 步骤1： 打开 HttpDNS 解析开关
[TXLiveBase enableCustomHttpDNS:YES];
// 步骤2： 实现 HttpDNS 解析代理：TXLiveBaseDelegate#onCustomHttpDNS
- (void)onCustomHttpDNS:(NSString *)hostName ipList:(NSMutableArray<NSString *> *)l
    // 把 hostName 解析到 ip 地址后，保存到 iPList，返回给 SDK 内部。注意：这里不要进行耗时的
    // MSDKDnsResolver 是腾讯云提供的 HTTPDNS SDK 解析接口
    NSArray *result = [[MSDKDns sharedInstance] WGGetHostByName:hostName];
    NSString *ip = nil;
    if (result && result.count > 1) {
        if (![result[1] isEqualToString:@"0"]) {
            ip = result[1];
```

```
            } else {
                ip = result[0];
            }
        }
        [list addObject:ip];
    }

    // 步骤3： 设置 HttpDNS 解析代理
    [TXLiveBase sharedInstance].delegate = self;
```

7、HEVC 自适应降级播放

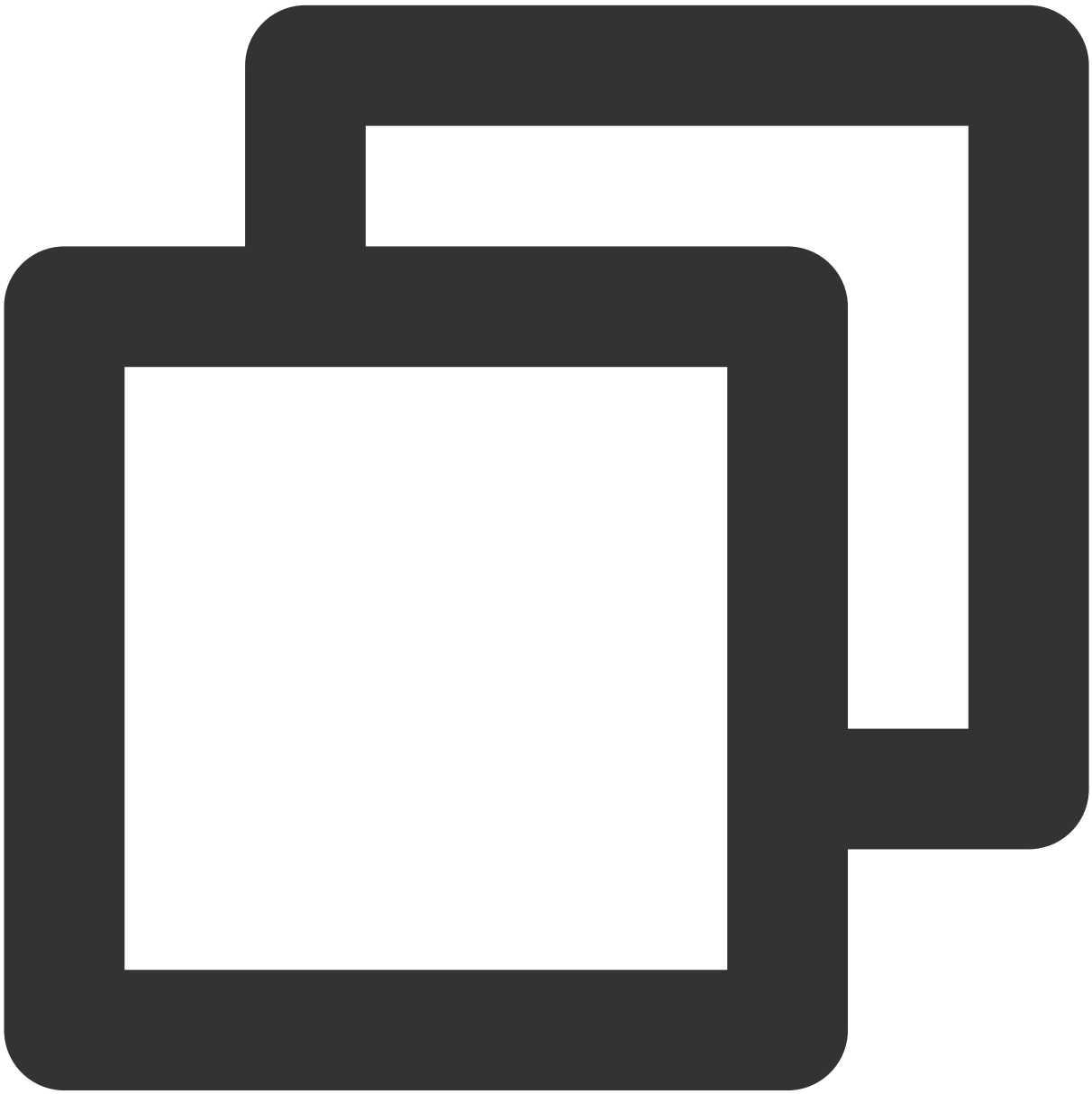
播放器支持同时传入 HEVC 和其它视频编码格式比如： H.264 的播放链接， 当播放机型不支持 HEVC 格式时，将自动降级为配置的其它编码格式（如： H.264 ）的视频播放。
注意：播放器高级版 11.7 版本开始支持。



```
#import <CoreMedia/CoreMedia.h> // 引入头文件

NSDictionary *dic = @{
    VOD_KEY_VIDEO_CODEC_TYPE:@(kCMVideoCodecType_HEVC),  // 指定原始 HEVC 视频编码类型
    VOD_KEY_BACKUP_URL:@"${backupPlayUrl}"};  // 设置 H.264 格式等备选播放链接地址
[_txVodPlayer setExtentOptionInfo:dic];

// 设置原始 HEVC 播放链接
[_txVodPlayer startVodPlay:@"${hevcPlayUrl}"];
```

8、音量均衡

播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。通过 `TXVodPlayer#setAudioNormalization` 设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。
注意：播放器高级版 11.7 版本开始支持。



```
/**
 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h）
 关：AUDIO_NORMALIZATION_OFF
 开：AUDIO_NORMALIZATION_STANDARD（标准）
```

```
    AUDIO_NORMALIZATION_LOW （低）
    AUDIO_NORMALIZATION_HIGH（高）
 可填自定义数值：从低到高，范围-70 - 0 LUFS
*/
[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_STANDARD];              //后

[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_OFF];
```

播放器事件监听

您可以为 TXVodPlayer 对象绑定一个 TXVodPlayListener 监听器，即可通过 onPlayEvent（事件通知） 和 onNetStatus（状态反馈）向您的应用程序同步信息。

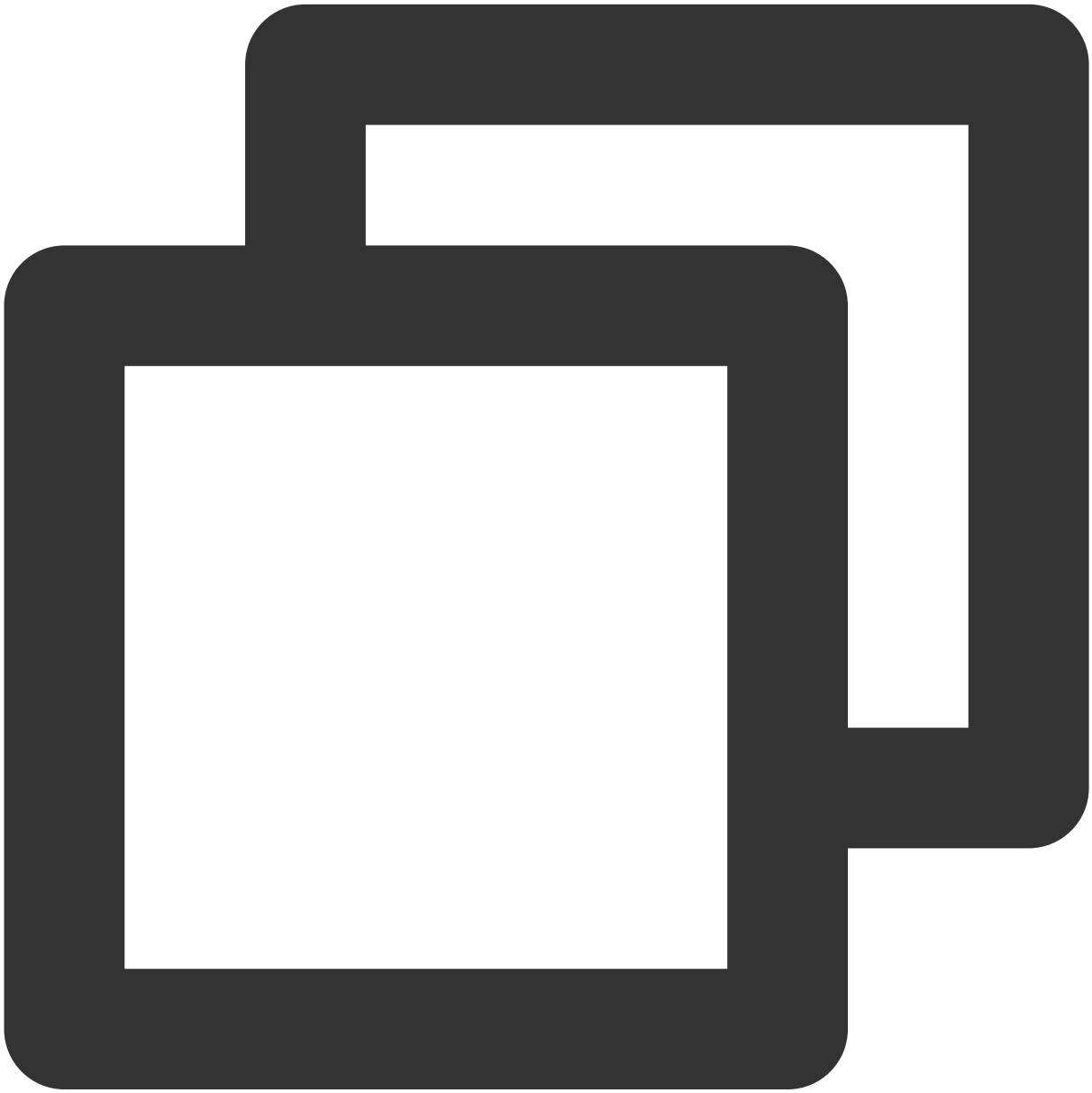
事件通知（onPlayEvent）

播放事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度和总体时长
PLAY_EVT_PLAY_LOADING	2007	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件
PLAY_EVT_VOD_LOADING_END	2014	视频播放 loading 结束，视频继续播放
VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seek 完成，10.3版本开始支持
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	循环播放，一轮播放结束（10.8 版本开始支持）
VOD_PLAY_EVT_HIT_CACHE	2002	启播时命中缓存事件（11.2 版本开始支持）
VOD_PLAY_EVT_VIDEO_SEI	2030	收到 SEI 帧事件（ 播放器高级版11.6 版本开始支持）。
VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	2031	发生 HEVC 降级播放（播放器高级版12.0版本开始支持）。
VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET	2017	播放器收到首帧数据包事件（12.0 版本开始支持）。

SEI 帧

SEI（Supplemental Enhancement Information）帧是一种用于传递附加信息的帧类型，播放器高级版会解析视频流中的 SEI 帧，通过
VOD_PLAY_EVT_VIDEO_SEI 事件回调， 注意：播放器高级版 11.6 版本开始支持。



```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
    if (EvtID == VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = [param objectForKey:EVT_KEY_SEI_TYPE]; //  the type of video
        int seiSize = [param objectForKey:EVT_KEY_SEI_SIZE]; // the data size of vi
        NSData *seiData = [param objectForKey:EVT_KEY_SEI_DATA]; // the byte array
```

```
        }
    }
```

警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件 ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连，已启动自动重连（重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了）
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解

连接事件

此外还有几个连接服务器的事件，主要用于测定和统计服务器连接时间：

事件 ID	数值	含义说明
PLAY_EVT_VOD_PLAY_PREPARED	2013	播放器已准备完成，可以播放。设置了 autoPlay 为 false 之后，需要在收到此事件后，调用 resume 才会开始播放
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包（IDR）

画面事件

以下事件用于获取画面变化信息：

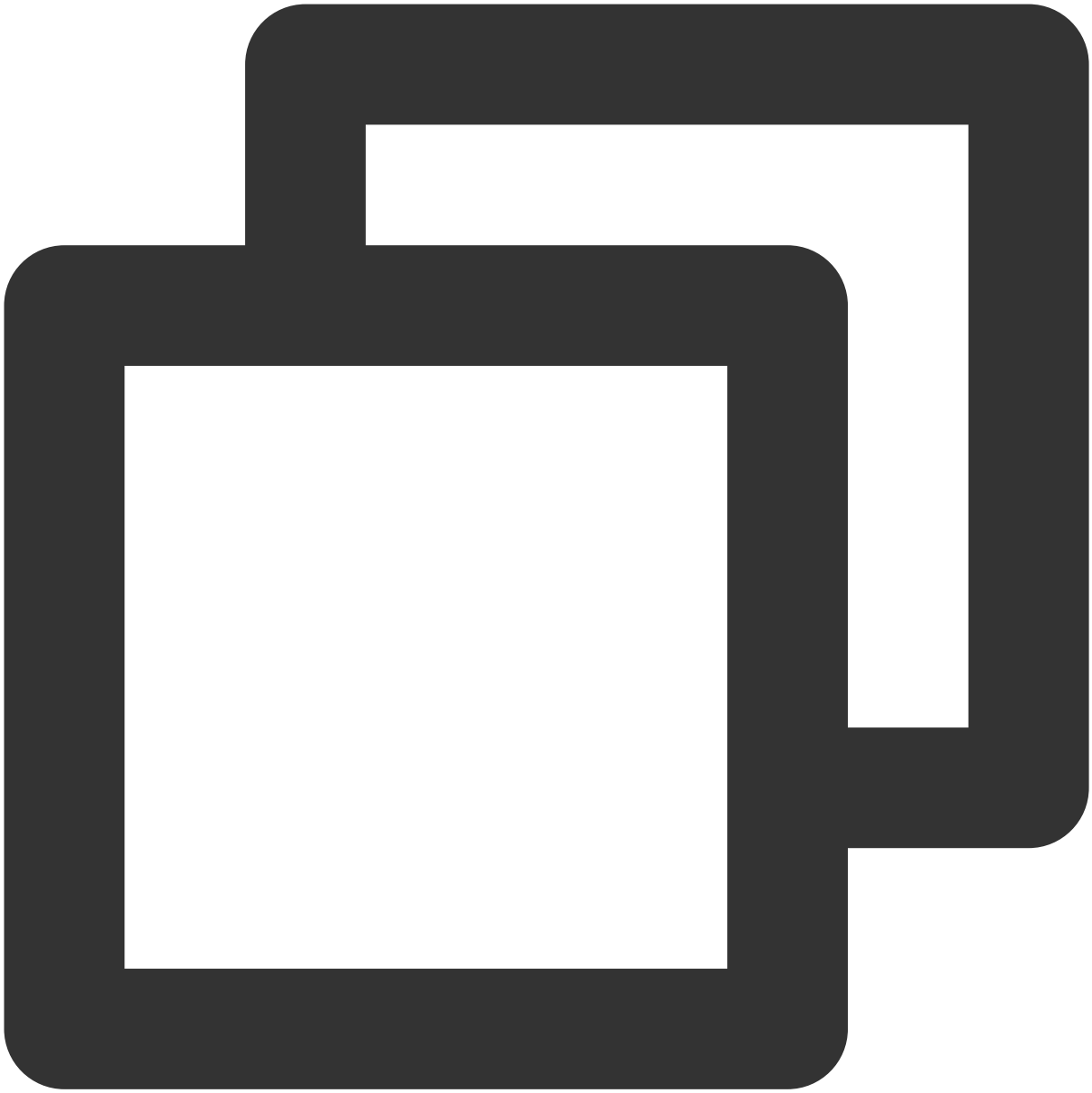
事件 ID	数值	含义说明
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率改变
PLAY_EVT_CHANGE_ROATION	2011	MP4 视频旋转角度

视频信息事件

事件 ID	数值	含义说明
PLAY_EVT_GET_PLAYINFO_SUCC	2010	成功获取播放文件信息

如果通过 fileId 方式播放且请求成功，SDK 会将一些请求信息通知到上层。您可以在收到 PLAY_EVT_GET_PLAYINFO_SUCC 事件后，解析 param 获取视频信息。

视频信息	含义说明
EVT_PLAY_COVER_URL	视频封面地址
EVT_PLAY_URL	视频播放地址
EVT_PLAY_DURATION	视频时长
EVT_KEY_WATER_MARK_TEXT	幽灵水印文本内容（11.6 版本开始支持）



```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
{
    if (EvtID == PLAY_EVT_VOD_PLAY_PREPARED) {
        //收到播放器已经准备完成事件，此时可以调用pause、resume、getWidth、getSupportedBitr
    } else if (EvtID == PLAY_EVT_PLAY_BEGIN) {
        // 收到开始播放事件
    } else if (EvtID == PLAY_EVT_PLAY_END) {
        // 收到开始结束事件
    }
}
```

幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程。幽灵水印的内容在收到播放器的 `VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `[param objectForKey:@"EVT_KEY_WATER_MARK_TEXT"]` 获取。详细使用教程参见 超级播放器组件 > 幽灵水印。

注意： 播放器 11.6 版本开始支持

播放错误事件

说明：

[-6004 , -6010] 错误事件 11.0 版本开始支持。

事件ID	数值	含义说明
PLAY_ERR_NET_DISCONNECT	-2301	视频数据错误导致重试亦不能恢复正常播放。如：网络异常或下载数据错误，导致解封装超时或失败
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	系统播放器播放错误
VOD_PLAY_ERR_DECODE_VIDEO_FAIL	-6006	视频解码错误，视频格式不支持
VOD_PLAY_ERR_DECODE_AUDIO_FAIL	-6007	音频解码错误，音频格式不支持
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	字幕解码错误
VOD_PLAY_ERR_RENDER_FAIL	-6009	视频渲染错误
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	视频后处理错误
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	获取点播文件信息失败，建议检查AppId、FileId或Psign填写是否正确。

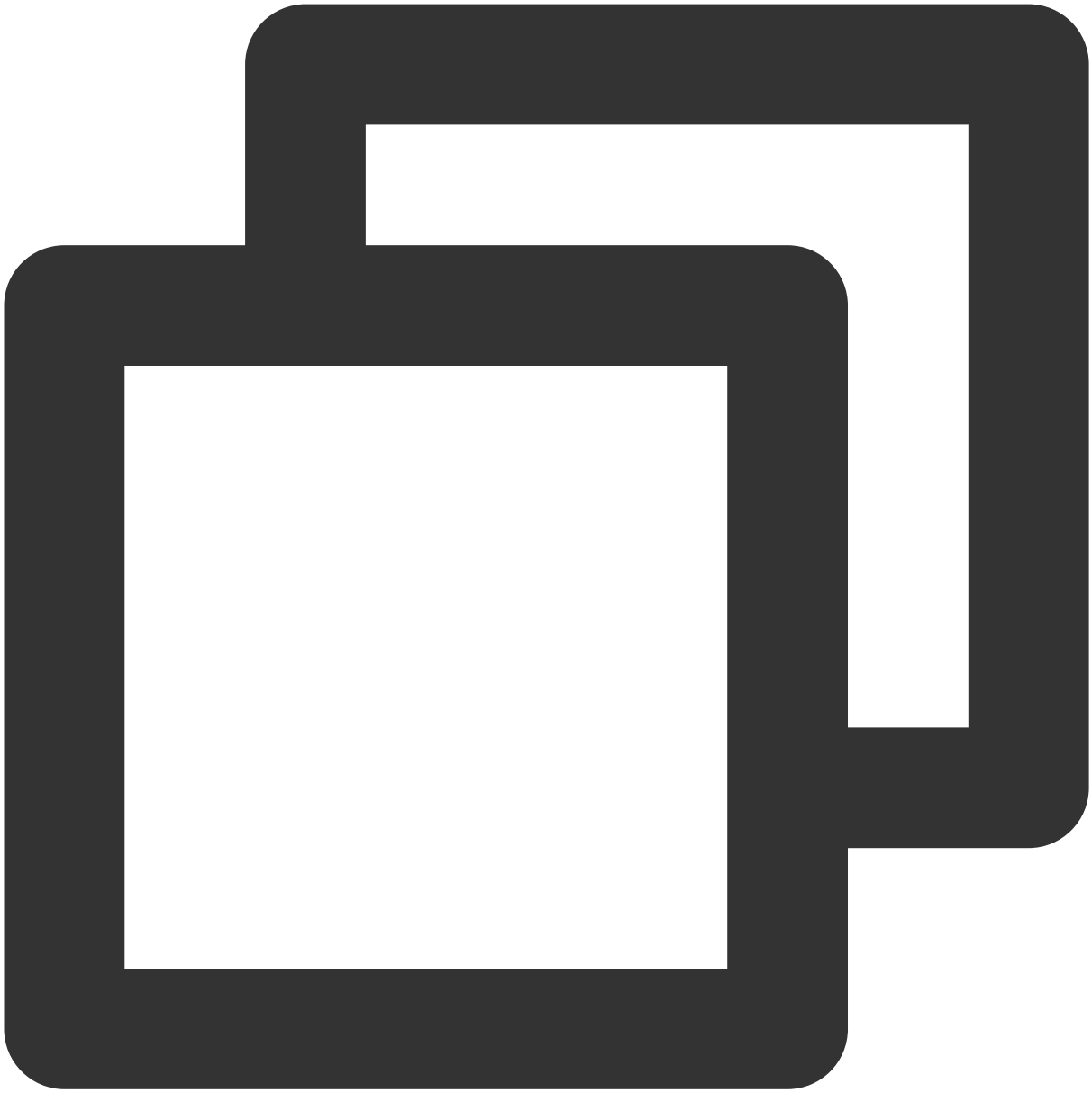
状态反馈（onNetStatus）

状态反馈每0.5秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前 SDK 内部的一些具体情况，以便您能对当前视频播放状态等有所了解。

评估参数	含义说明
CPU_USAGE	当前瞬时 CPU 使用率
VIDEO_WIDTH	视频分辨率 - 宽
VIDEO_HEIGHT	视频分辨率 - 高

NET_SPEED	当前的网络数据接收速度，单位：KBps
VIDEO_FPS	当前流媒体的视频帧率
VIDEO_BITRATE	当前流媒体的视频码率，单位 bps
AUDIO_BITRATE	当前流媒体的音频码率，单位 bps
V_SUM_CACHE_SIZE	缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
SERVER_IP	连接的服务器 IP

通过 onNetStatus 获取视频播放过程信息示例：



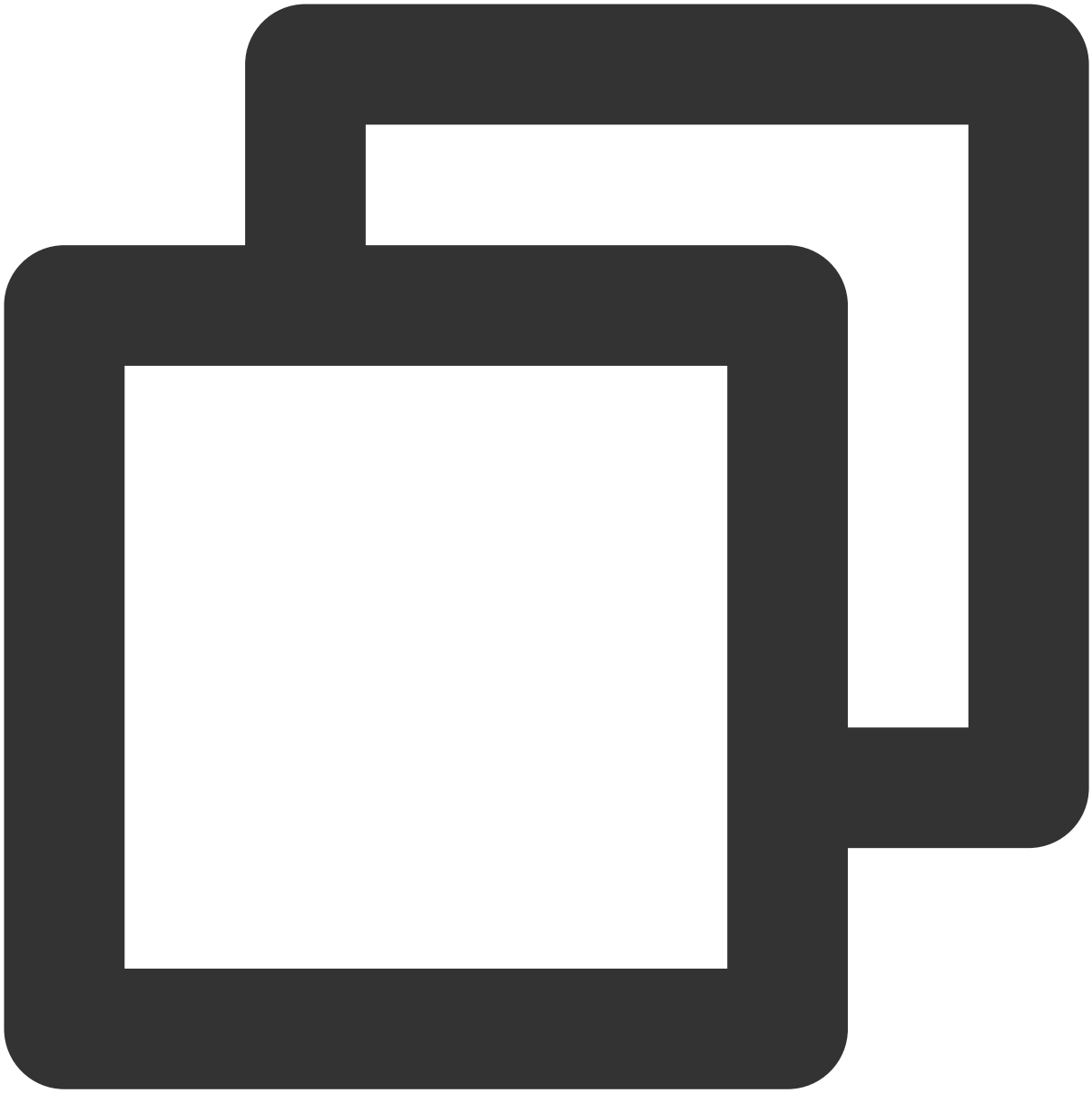
```
- (void)onNetStatus:(TXVodPlayer *)player withParam:(NSDictionary *)param {
    //获取当前CPU使用率
    float cpuUsage = [[param objectForKey:@"CPU_USAGE"] floatValue];
    //获取视频宽度
    int videoWidth = [[param objectForKey:@"VIDEO_WIDTH"] intValue];
    //获取视频高度
    int videoHeight = [[param objectForKey:@"VIDEO_HEIGHT"] intValue];
    //获取实时速率
    int  speed = [[param objectForKey:@"NET_SPEED"] intValue];
    //获取当前流媒体的视频帧率
    int fps = [[param objectForKey:@"VIDEO_FPS"] intValue];
```

```
    //获取当前流媒体的视频码率，单位 kbps
    int videoBitRate = [[param objectForKey:@"VIDEO_BITRATE"] intValue];
    //获取当前流媒体的音频码率，单位 kbps
    int audioBitRate = [[param objectForKey:@"AUDIO_BITRATE"] intValue];
    //获取缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
    int jitterbuffer = [[param objectForKey:@"V_SUM_CACHE_SIZE"] intValue];
    //获取连接的服务器的IP地址
    NSString *ip = [param objectForKey:@"SERVER_IP"];
}
```

其它功能使用

HLS 直播视频源播放

播放器高级版本支持播放 HLS 直播视频源，从 11.8 版本开始支持带 HLS EVENT 直播视频源。用法如下：



```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMediaType:MEDIA_TYPE_HLS_LIVE];    // 指定HLS直播媒资类型
[_txVodPlayer setConfig:_config];
[_txVodPlayer startVodPlay:${YOUR_HSL_LIVE_URL}];
```

场景化功能

1、动态设置 AudioSession

有时候需要根据场景来动态设置播放音频输出方式，特别是对于 iPhone 来说，天然支持多音频播放及后台模式。因此，我们根据用户的场景支持了以下三种主要的模式：

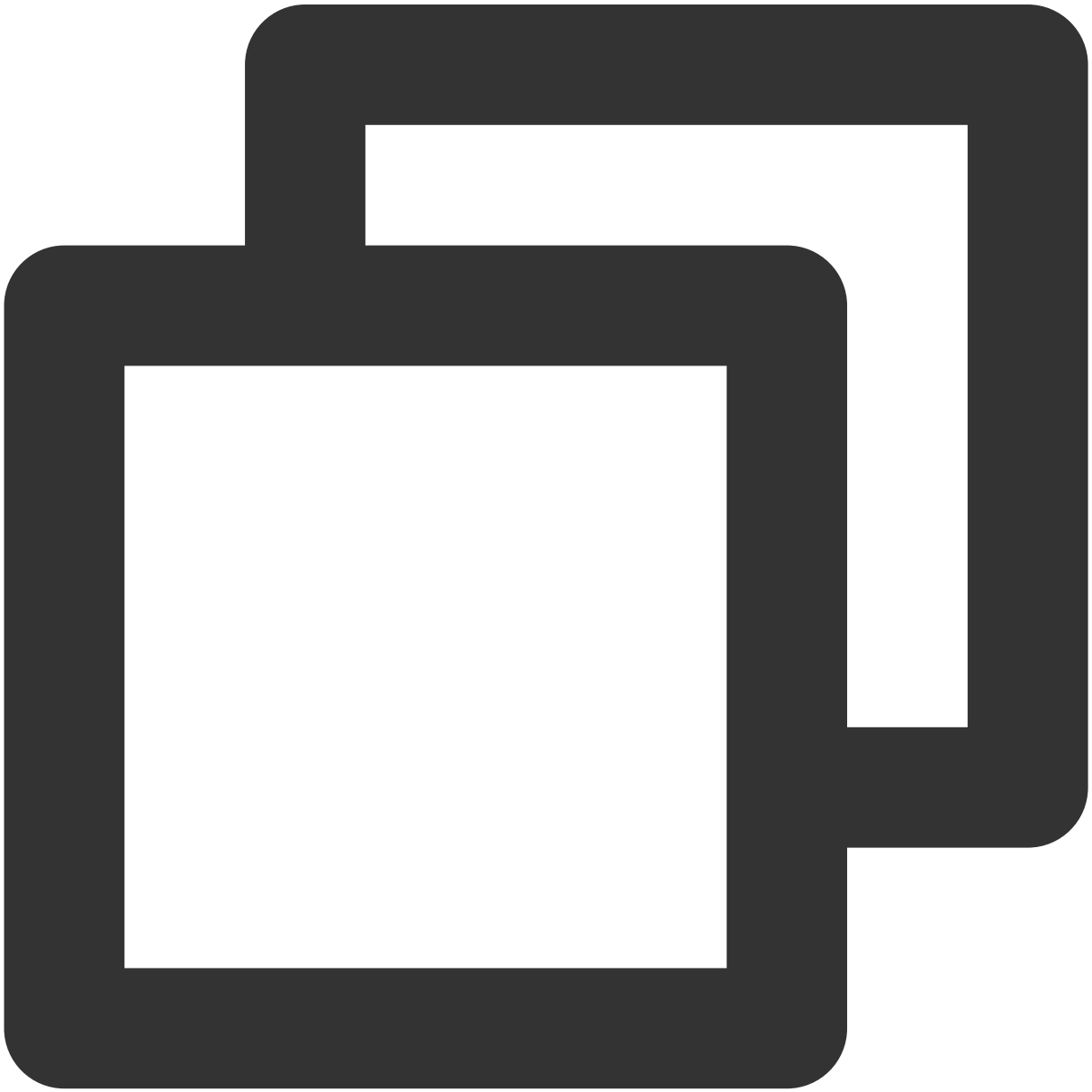
AVAudioSessionCategoryPlayback：后台独占播放。

AVAudioSessionCategoryPlayAndRecord：后台独占播放。

AVAudioSessionCategoryAmbient：混合播放。

可以根据所处的场景，利用上面的模式来设置 AudioSession 的 Category 和 Option 来达到自己的目的。下面罗列了两种场景的设置(以下设置可以根据自己的场景进行动态调整和设置)：

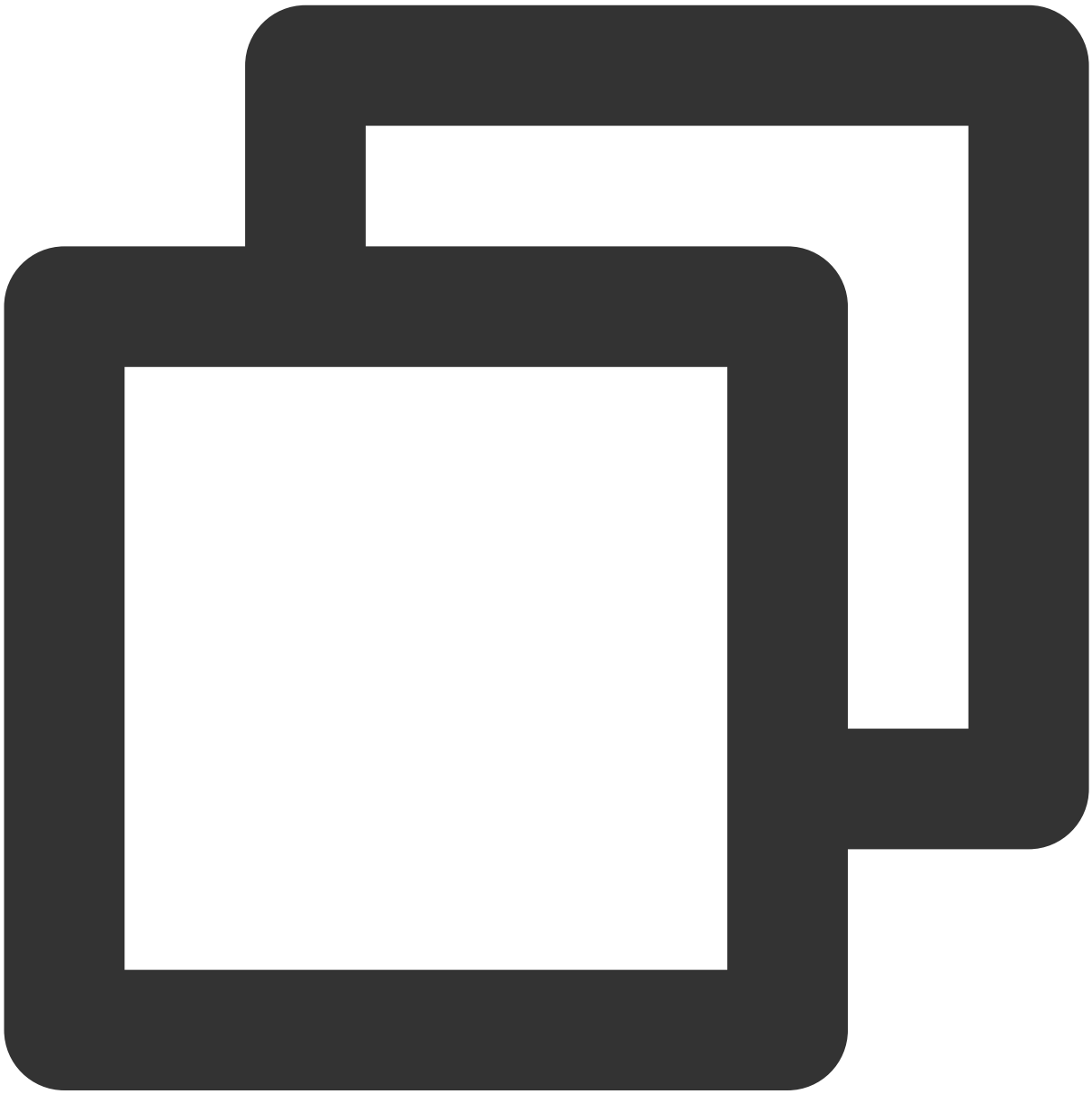
场景一 ：播放列表场景（视频播放需要支持列表里静音播放，并且不打断外部音频播放）。



```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback withOpt
```

```
[[AVAudioSession sharedInstance] setActive:YES error:nil];
```

场景二：播放详情场景（视频详情有声音，并且暂时打断外部音频，当视频播放完成以后，就恢复外部音频）。



```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryAmbient withOpti
[[AVAudioSession sharedInstance] setActive:NO withOptions:AVAudioSessionSetActiveOp
```

2、基于 SDK 的 Demo 组件

基于播放器SDK，腾讯云研发了一款 播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美

市面上各种流行视频 App 的播放软件。

3、开源 Github

基于播放器 SDK，腾讯云研发了沉浸式视频播放器组件、视频 Feed 流、多播放器复用组件等，而且随着版本发布，我们会提供跟多的基于用户场景的组件。您可以通过 Player_iOS 下载体验。

Android 端集成
集成指引

最近更新时间：2024-04-26 11:09:31

本文主要介绍如何快速地将腾讯云播放器SDK集成到您的项目中，不同版本的 SDK 集成方式都通用，按照如下步骤进行配置，就可以完成 SDK 的集成工作。

开发环境要求

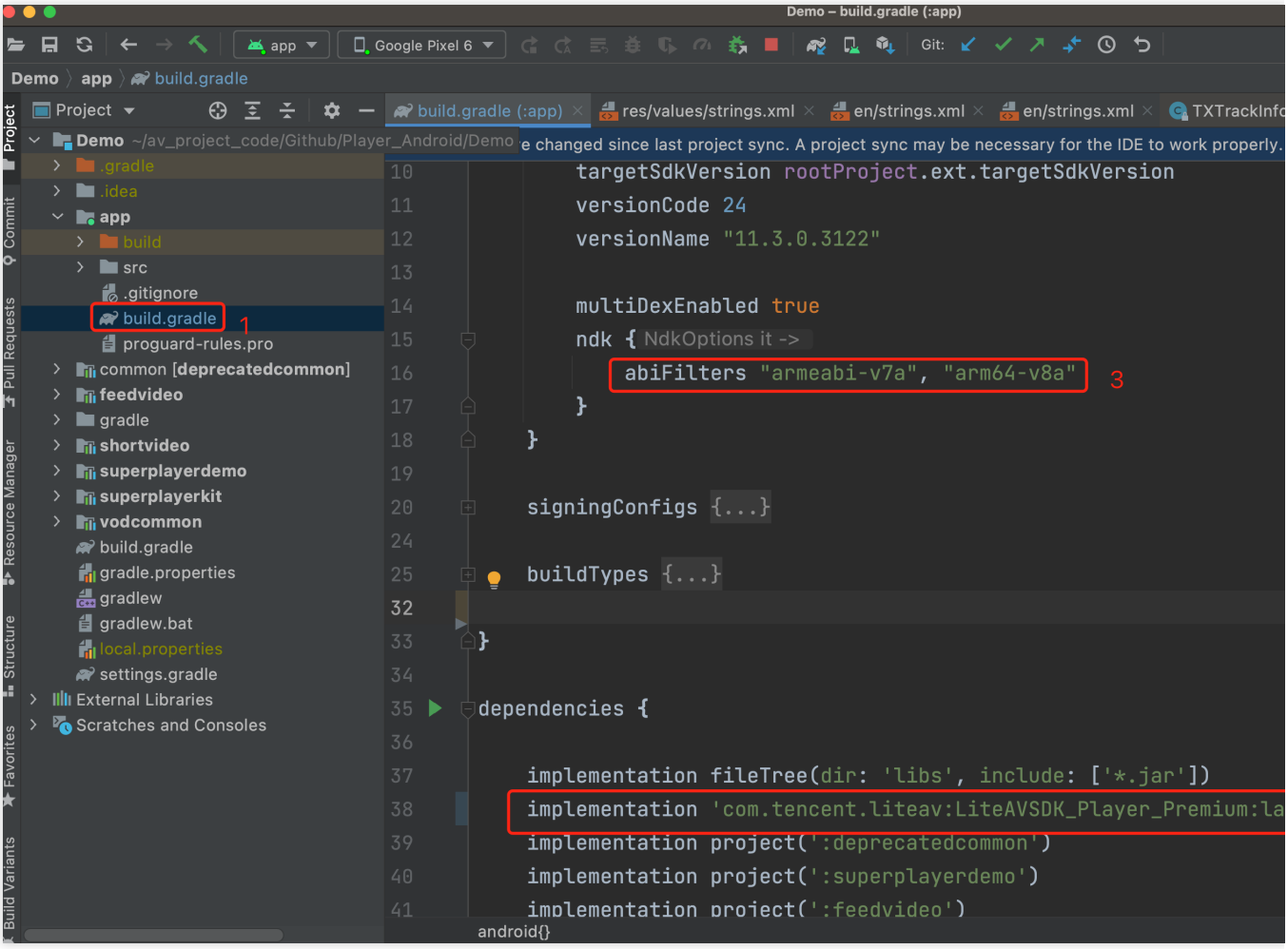
Android Studio 2.0+。
Android 4.1（SDK API 16）及以上系统。

集成 SDK（aar）

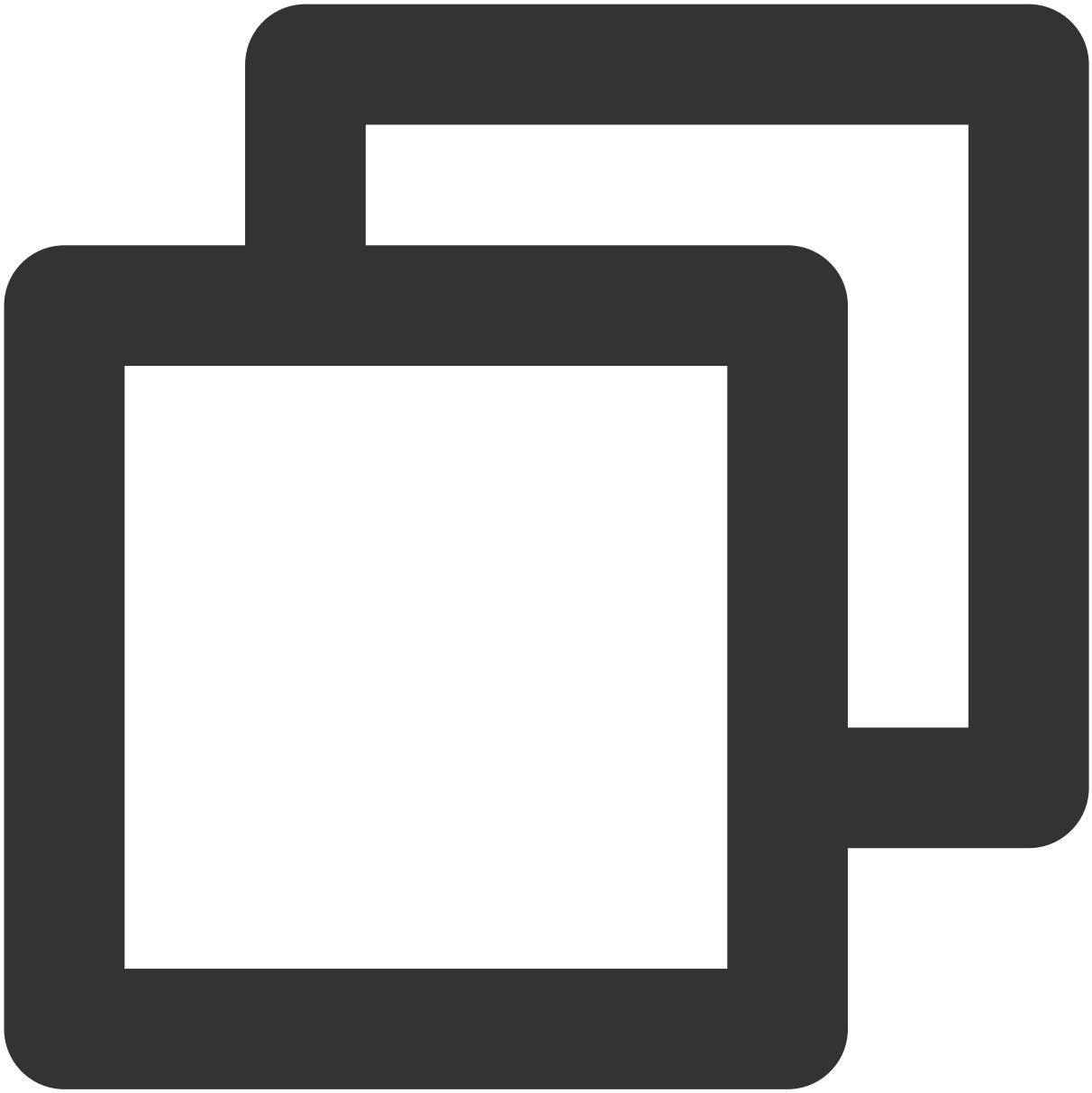
您可以选择使用 Gradle 自动加载的方式，或者手动下载 aar 再将其导入到您当前的工程项目中。

方法一：自动加载（aar）

播放器 SDK 已经发布到 mavenCentral 库，您可以通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK_Player_Premium。
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：

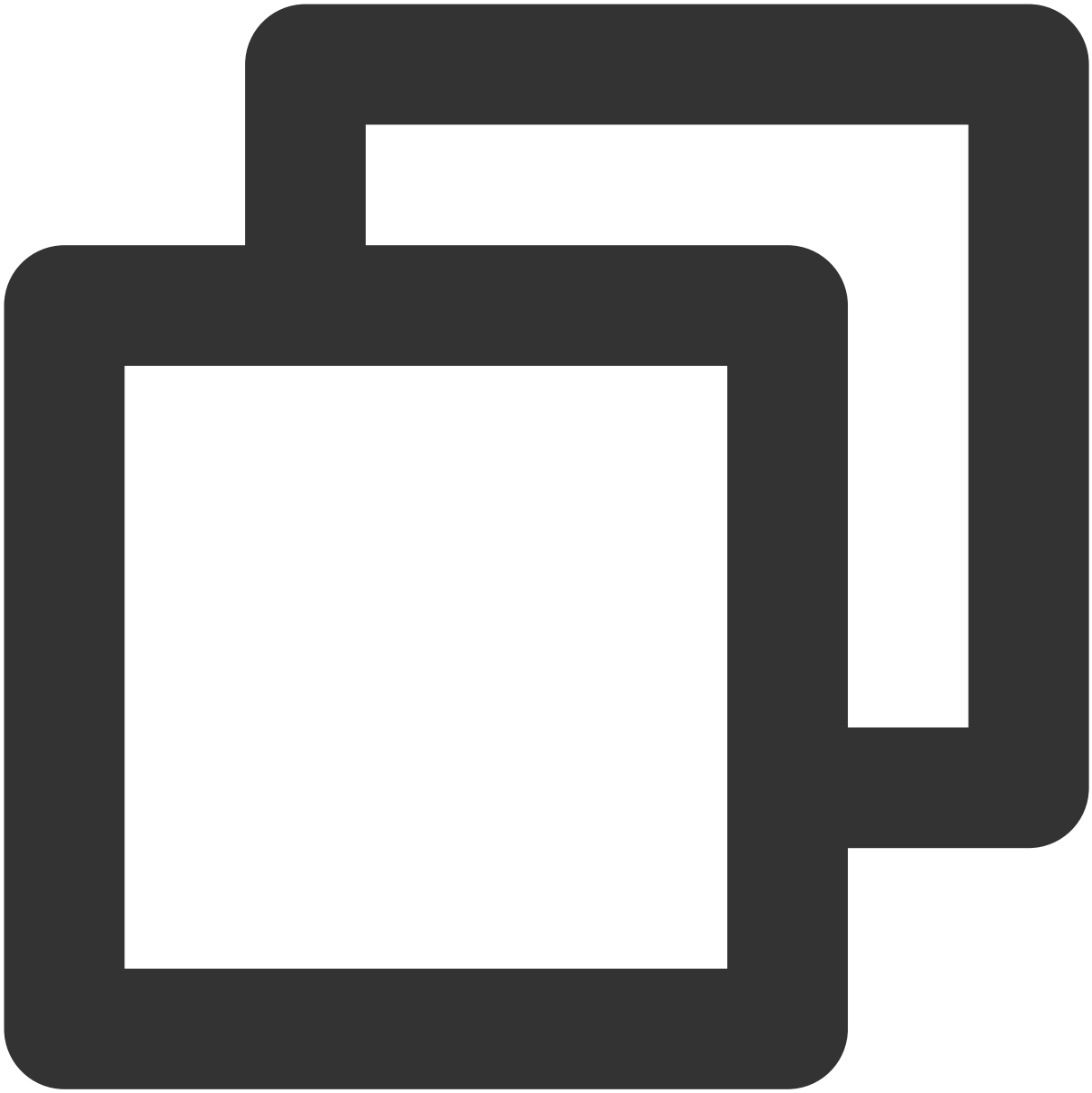


1. 打开工程根目录下的 build.gradle，添加mavenCentral库。



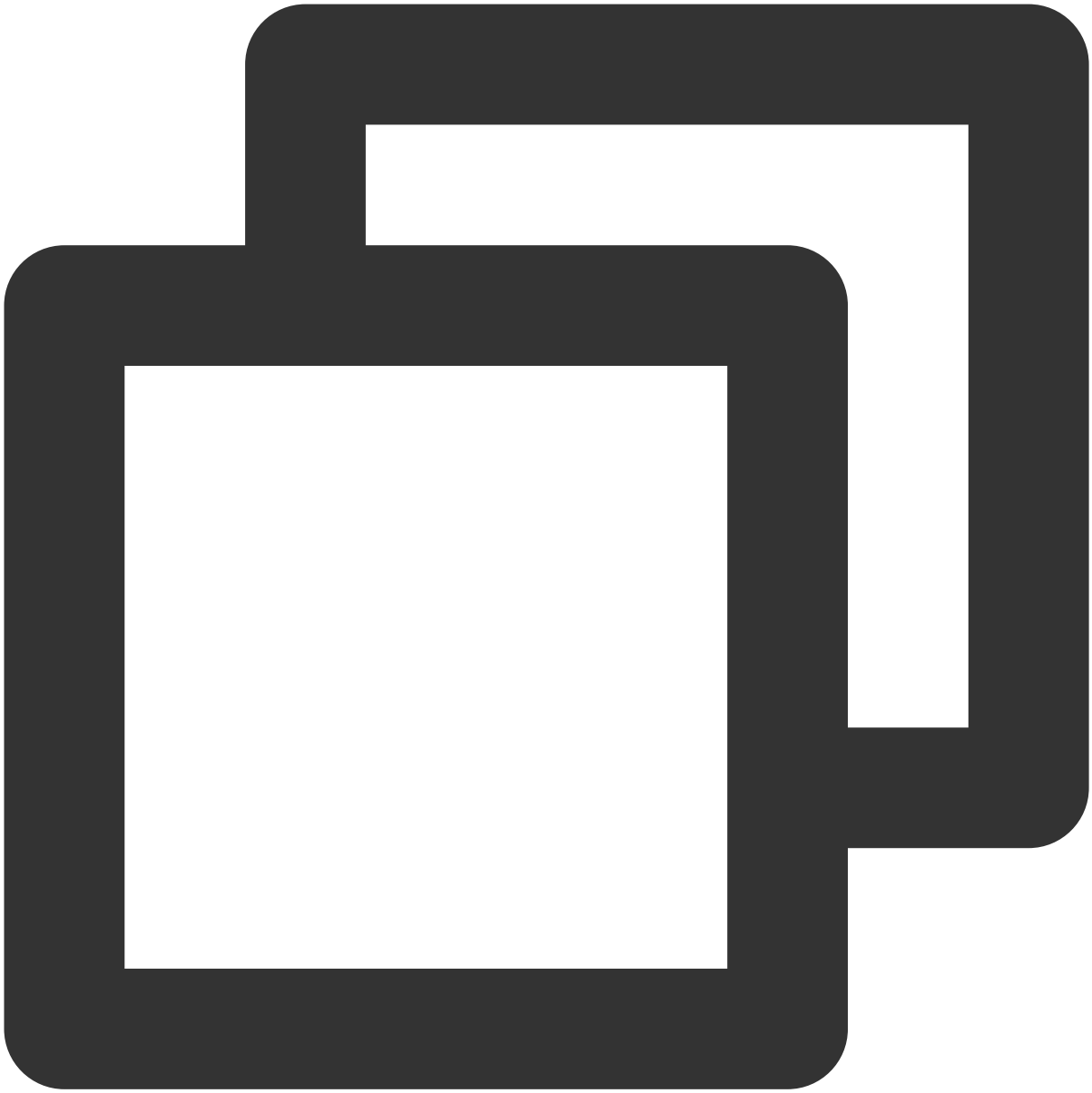
```
repositories {
 mavenCentral()
}
```

2. 打开 app 下的 build.gradle，在 dependencies 中添加 LiteAVSDK_Player 的依赖。



```
  dependencies {
    // 此配置默认集成 LiteAVSDK_Player_Premium 最新版本
  implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
    // 集成历史版，如：10.7.0.13038 版本，可通过下面方式集成
    // implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:10.7.0.13038'
}
```

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



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

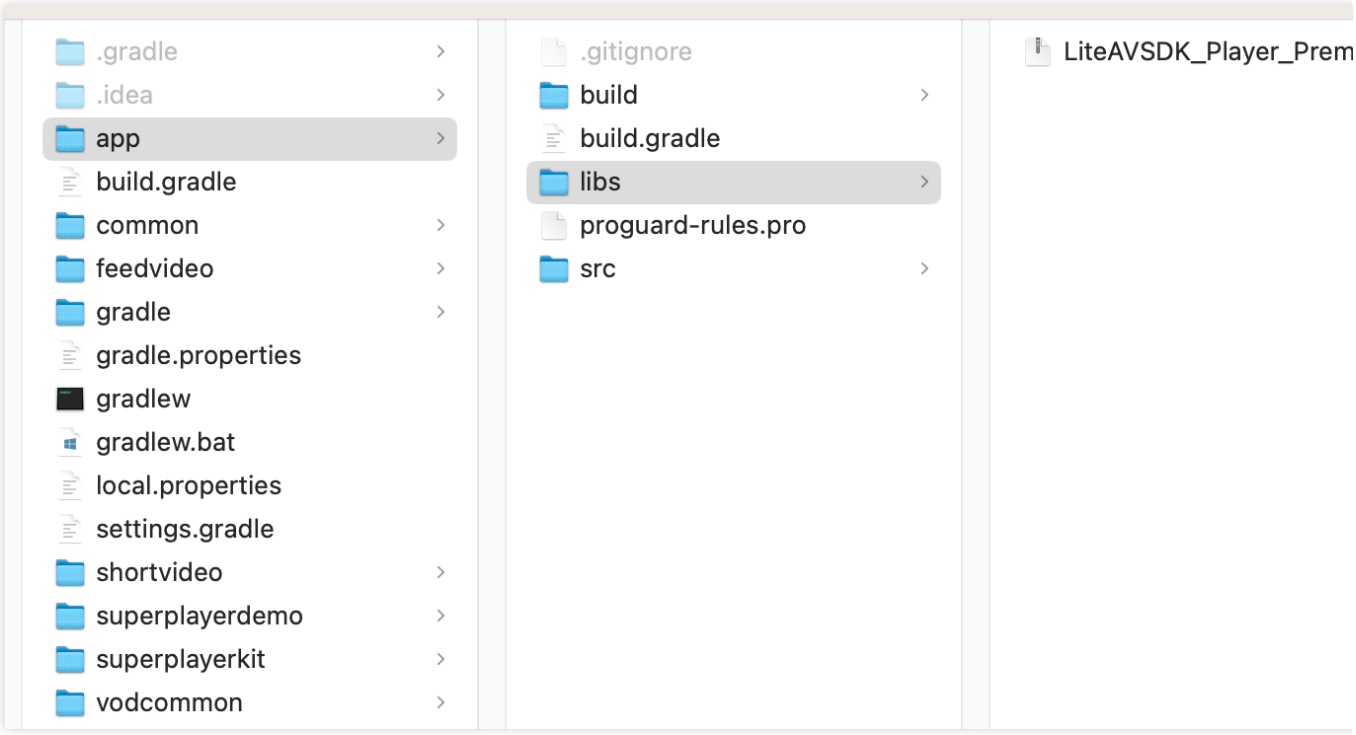
4. 单击

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

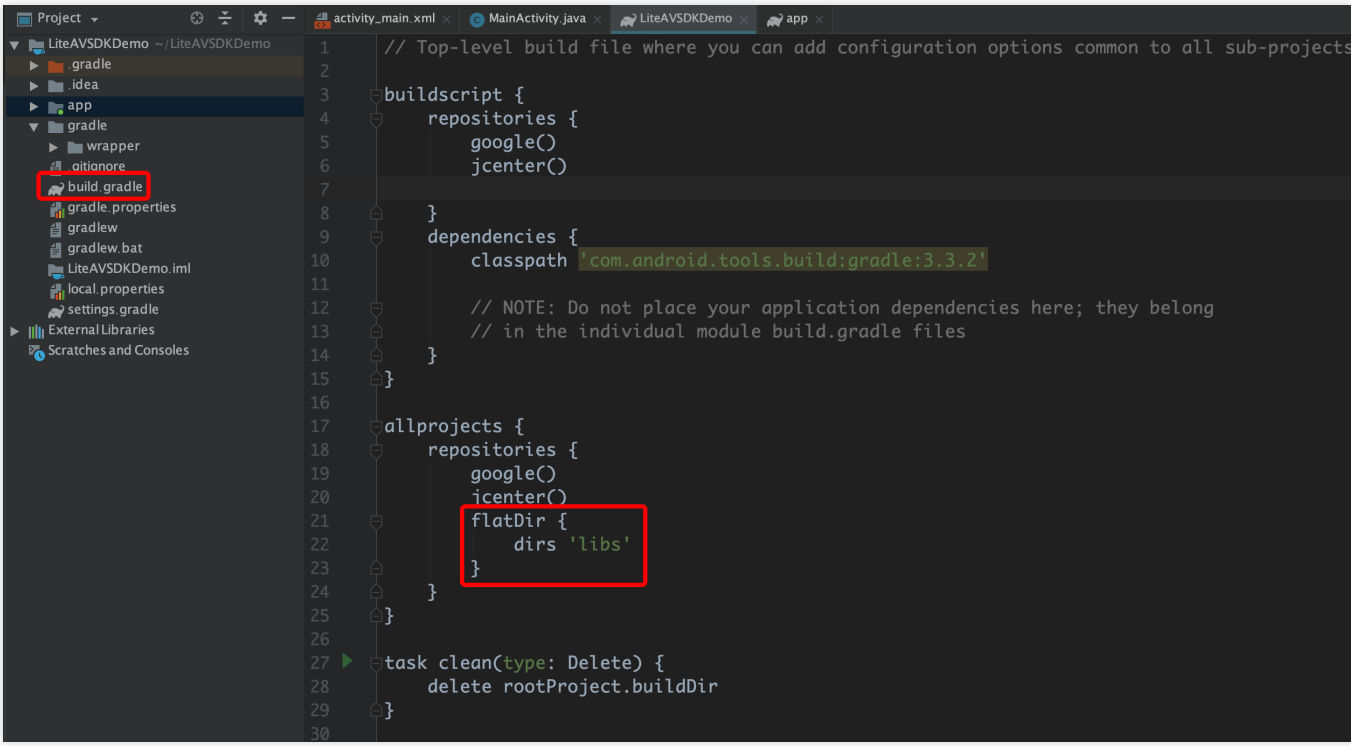
方法二：手动下载（aar）

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

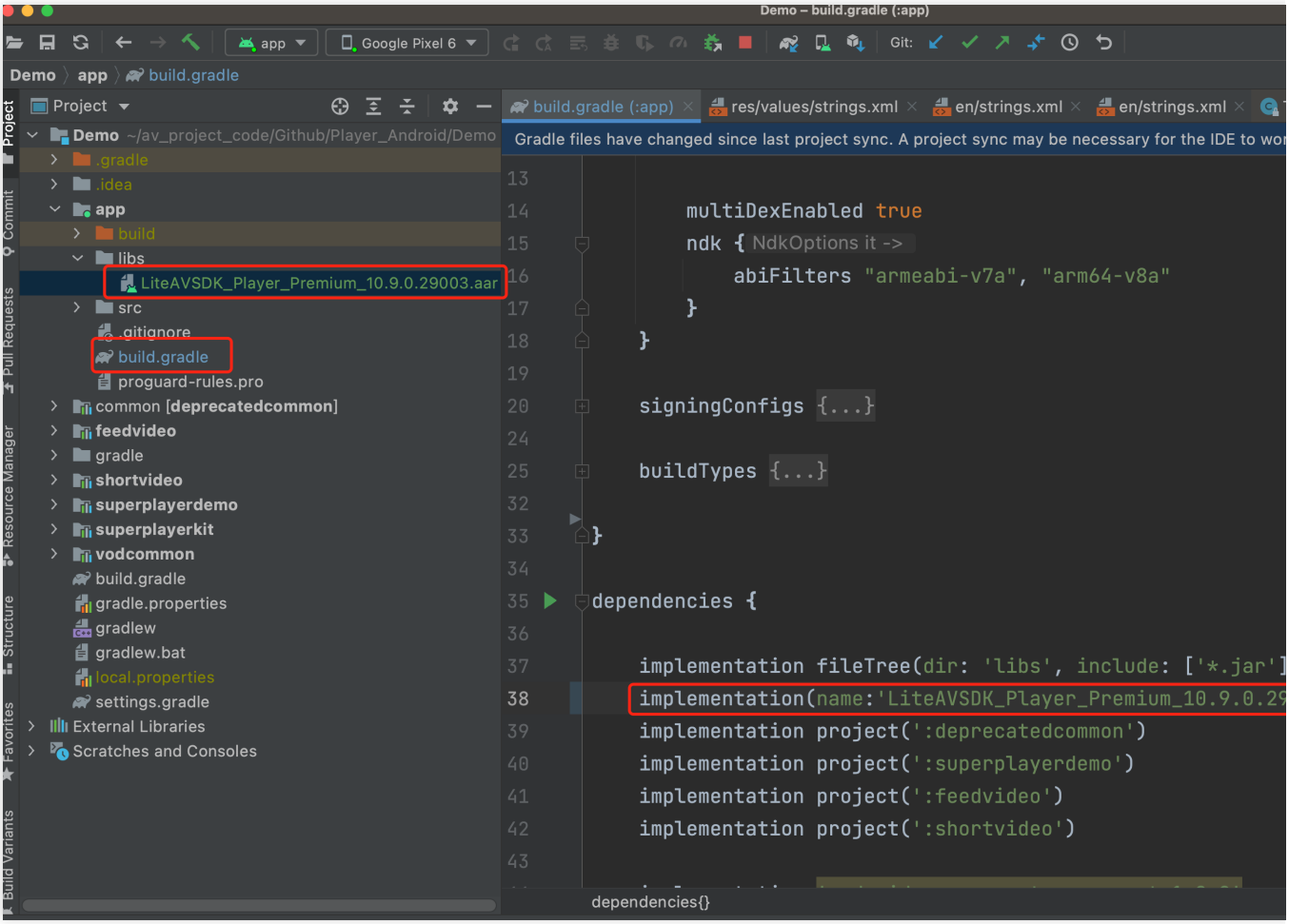
1. 下载 LiteAVSDK_Player_Premium ，下载完成后进行解压。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 **app/libs** 目录下：

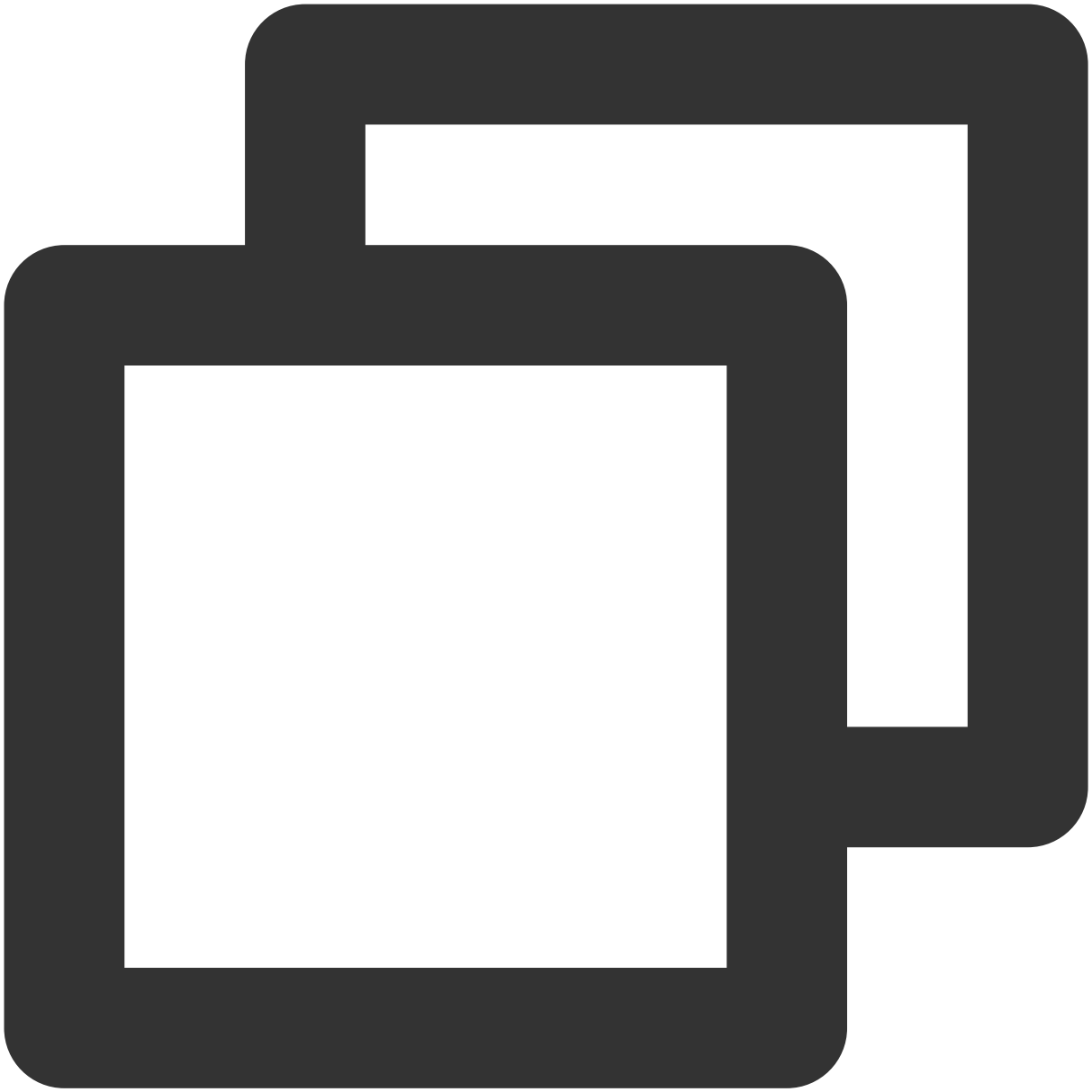


3. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



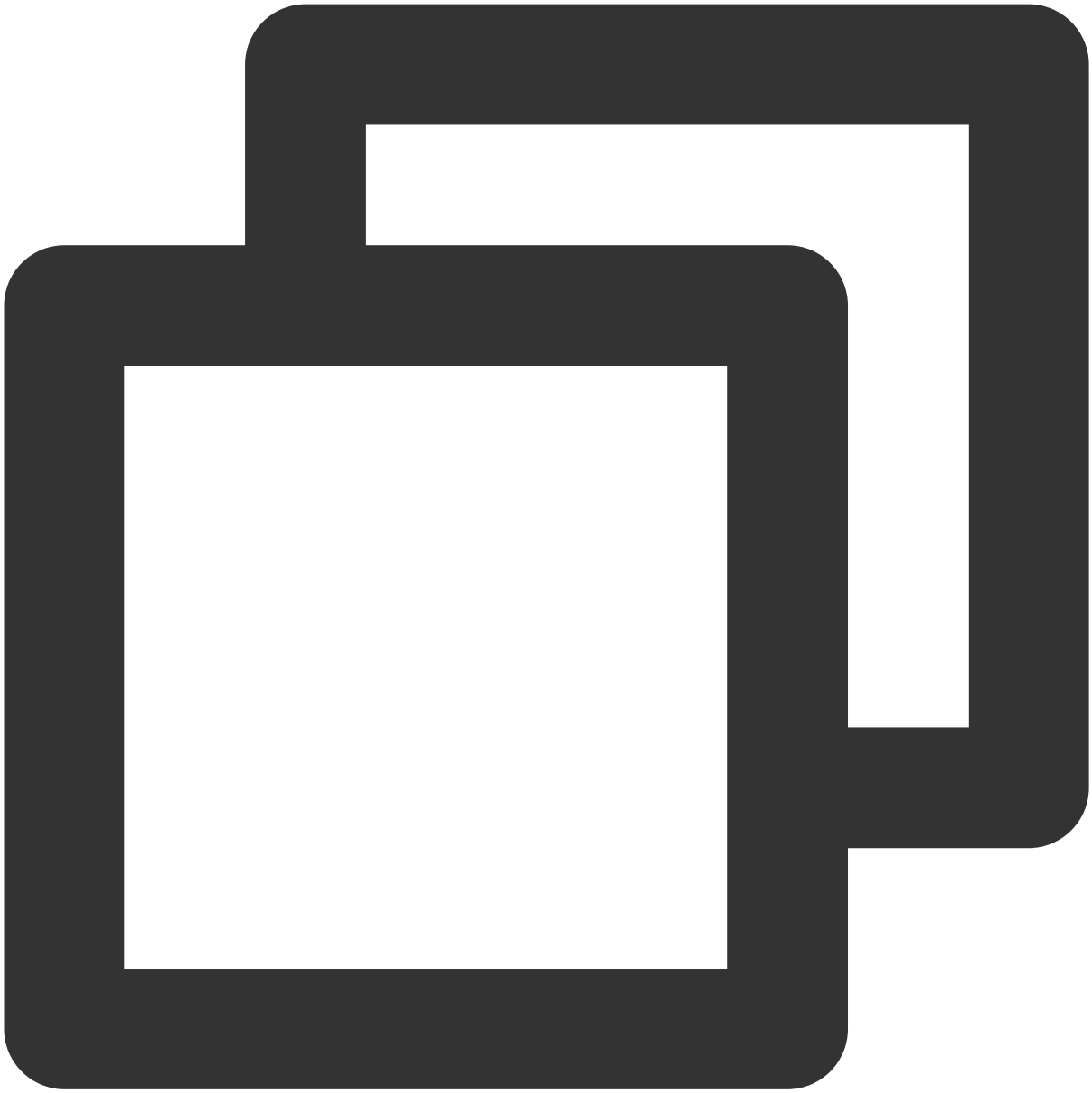
4. 添加 LiteAVSDK_Player_Premium 依赖，在 app/build.gradle 中，添加引用 aar 包的代码。


Demo – build.gradle (:app)
Demo app build.gradle
Project
Demo ~/av_project_code/Github/Player_Android/Demo
.gradle
.idea
app
build
libs
LiteAVSDK_Player_Premium_10.9.0.29003.aar
src
.gitignore
build.gradle
proguard-rules.pro
common [deprecatedcommon]
feedvideo
gradle
shortvideo
superplayerdemo
superplayerkit
vodcommon
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries
Scratches and Consoles
build.gradle (:app)
res/values/strings.xml
en/strings.xml
en/strings.xml
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to wo
multiDexEnabled true
ndk { NdkOptions it ->
abiFilters "armeabi-v7a", "arm64-v8a"
}
}
signingConfigs {...}
buildTypes {...}
}
dependencies {
implementation fileTree(dir: 'libs', include: ['*.jar']
implementation(name:'LiteAVSDK_Player_Premium_10.9.0.29
implementation project(':deprecatedcommon')
implementation project(':superplayerdemo')
implementation project(':feedvideo')
implementation project(':shortvideo')
dependencies{}



```
implementation(name:'LiteAVSDK_Player_Premium_10.9.0.29003', ext:'aar')
```

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



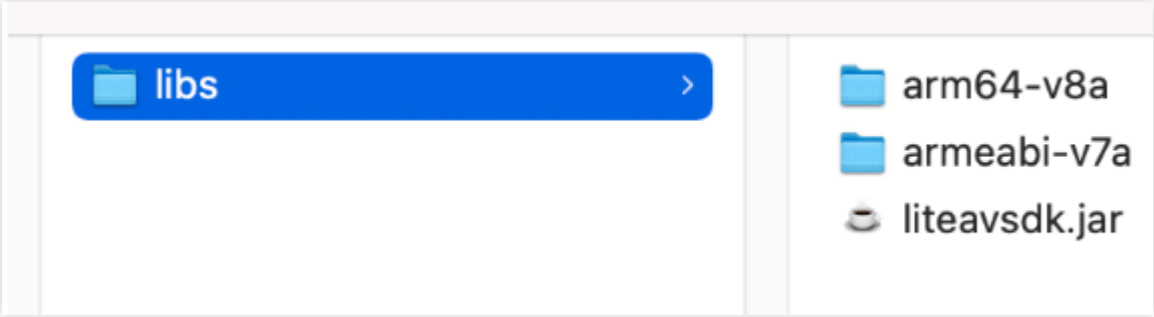
```
defaultConfig {
 ndk {
     abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
 }
}
```

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

集成 SDK（jar）

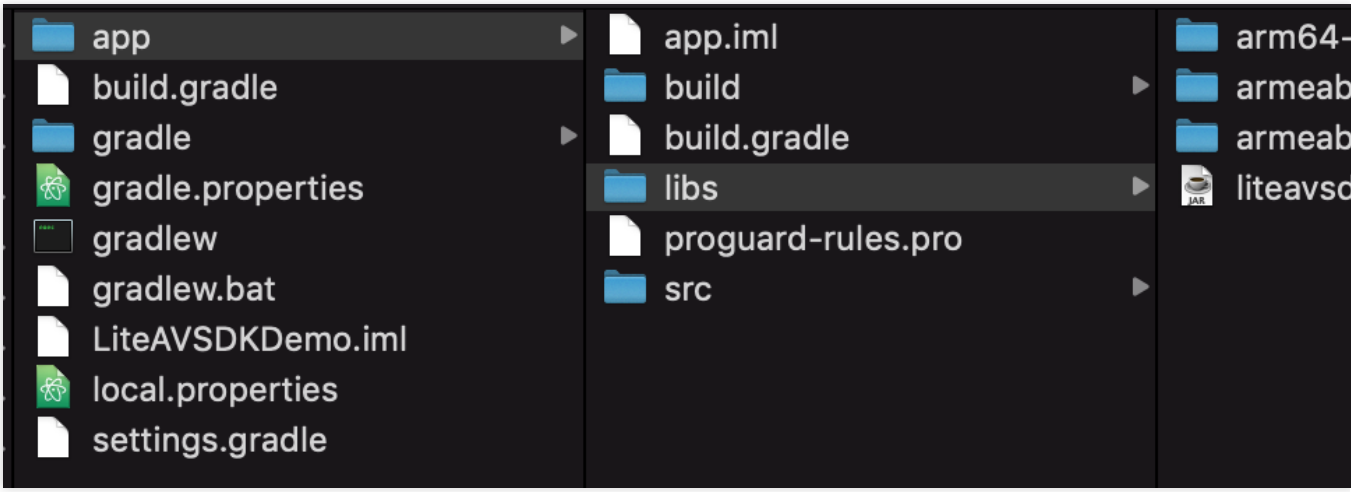
如果您不想集成 aar 库，也可以通过导入 jar 和 so 库的方式集成 LiteAVSDK：

1. 下载 LiteAVSDK_Player_Premium ，下载完成后进行解压。在 SDK 目录下找到 `LiteAVSDK_Player_Premium_xxx.zip` （其中 `xxx` 为 LiteAVSDK 的版本号），解压后得到 libs 目录，里面主要包含 jar 文件和 so 文件夹，文件清单如下：

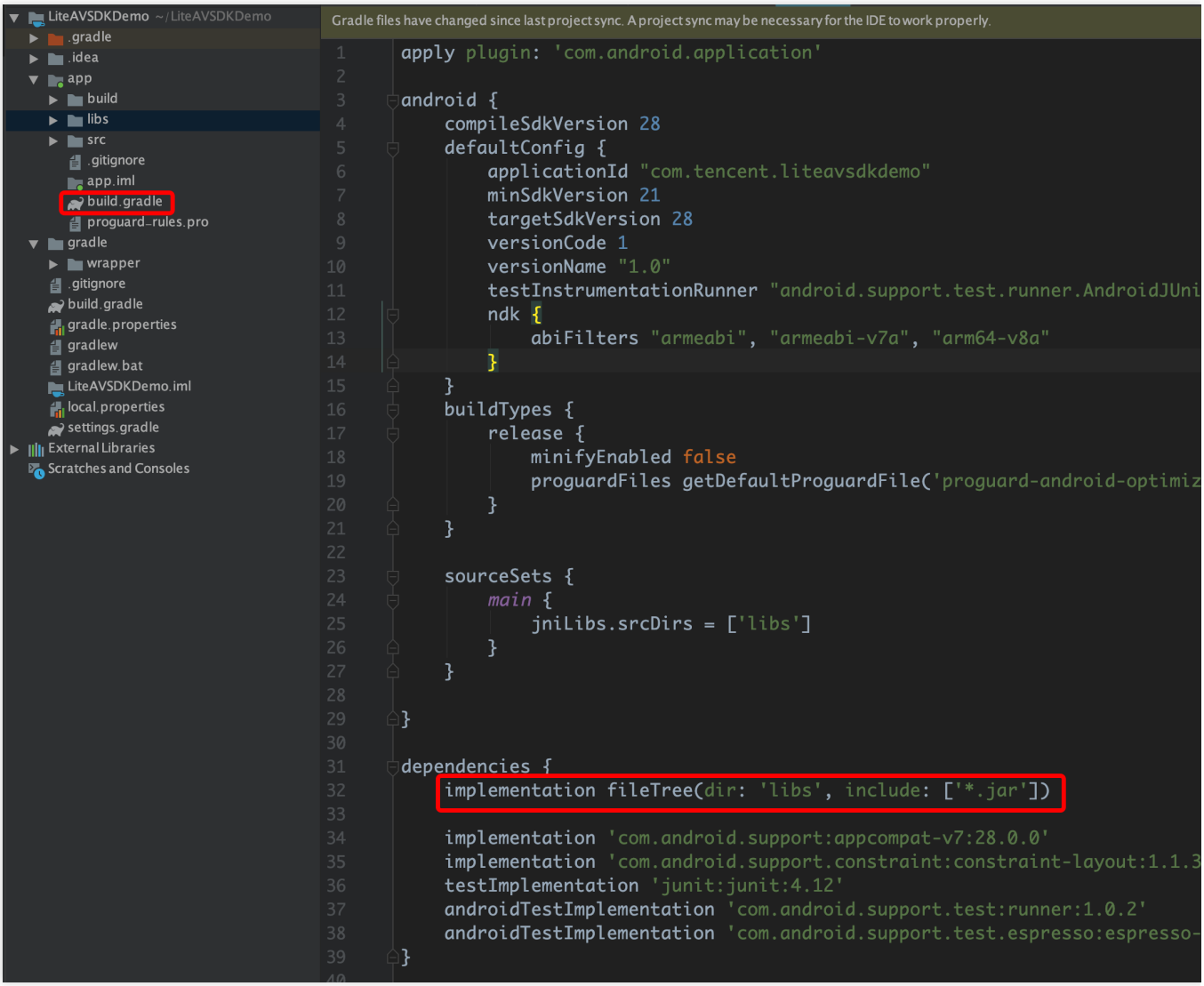


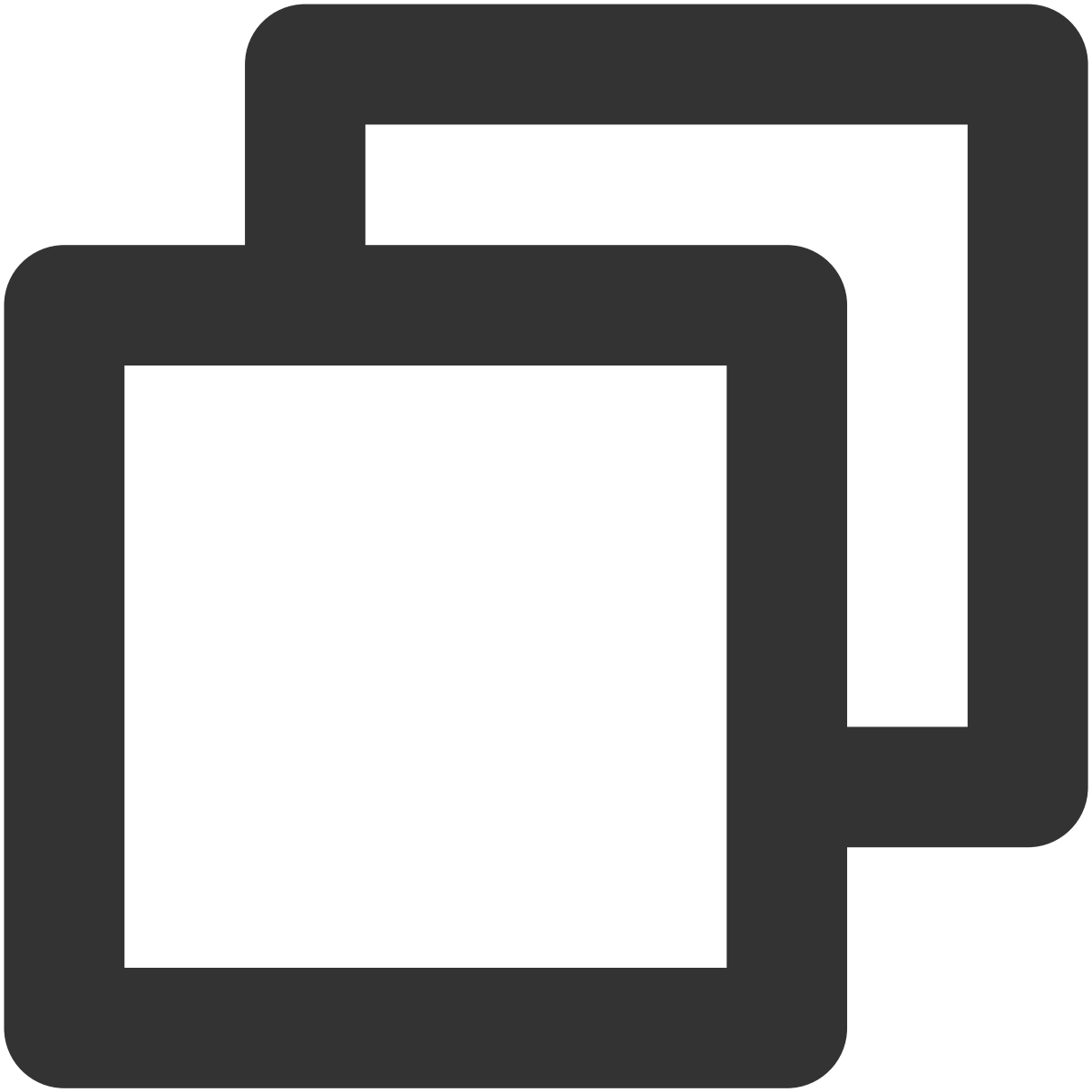
如果你还需要 armeabi 架构 so，复制一份 armeabi-v7a 目录，重命名为 armeabi 即可。

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



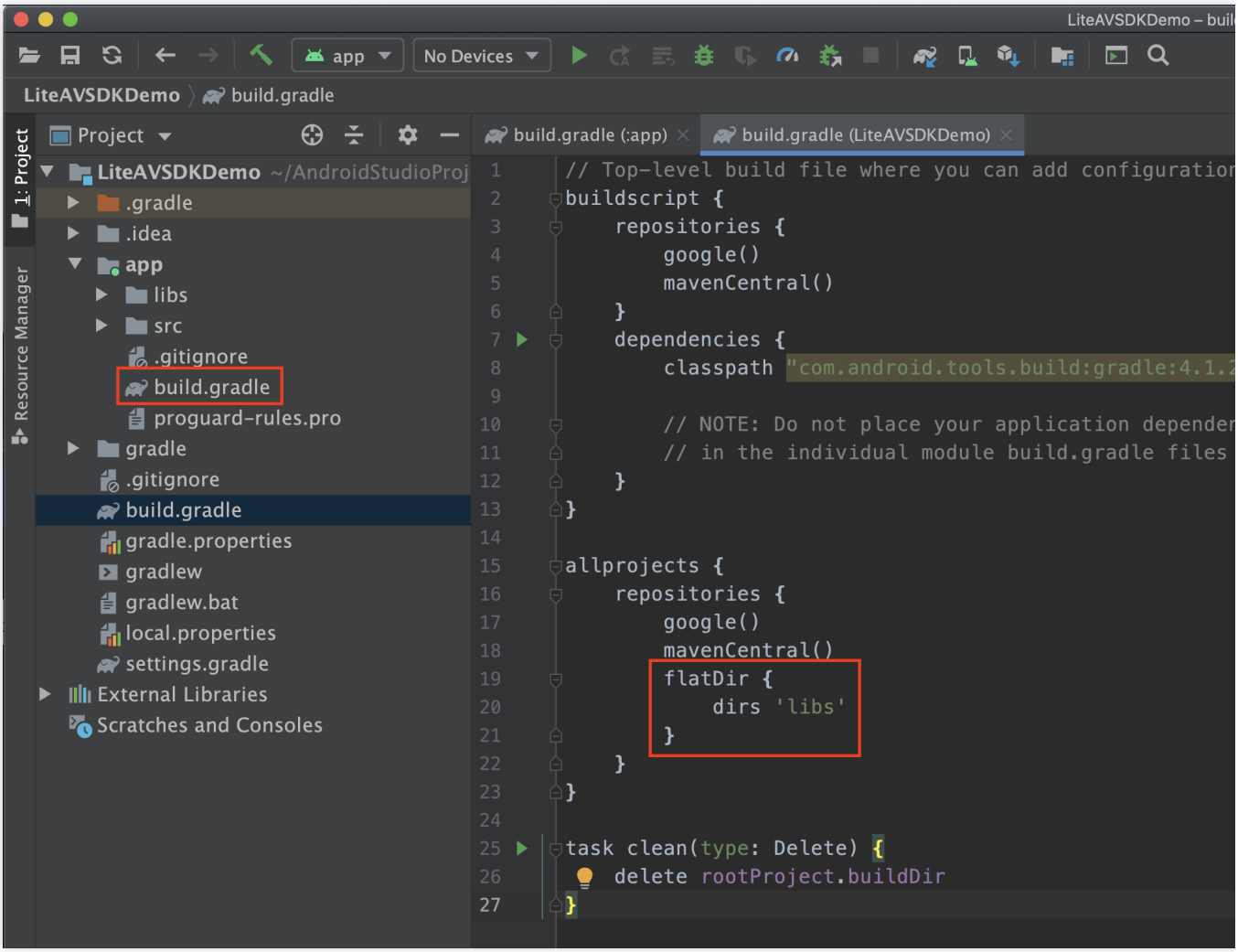
3. 在 `app/build.gradle` 中，添加引用 jar 库的代码。


LiteAVSDKDemo ~/LiteAVSDKDemo
.gradle
.idea
app
build
libs
src
.gitignore
app.iml
build.gradle
proguard-rules.pro
gradle
wrapper
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
LiteAVSDKDemo.iml
local.properties
settings.gradle
External Libraries
Scratches and Consoles
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
apply plugin: 'com.android.application'
android {
compileSdkVersion 28
defaultConfig {
applicationId "com.tencent.liteavsdkdemo"
minSdkVersion 21
targetSdkVersion 28
versionCode 1
versionName "1.0"
testInstrumentationRunner "android.support.test.runner.AndroidJUni
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android-optimiz
}
}
sourceSets {
main {
jniLibs.srcDirs = ['libs']
}
}
}
dependencies {
implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support.constraint:constraint-layout:1.1.3
testImplementation 'junit:junit:4.12'
androidTestImplementation 'com.android.support.test:runner:1.0.2'
androidTestImplementation 'com.android.support.test.espresso:espresso-
}

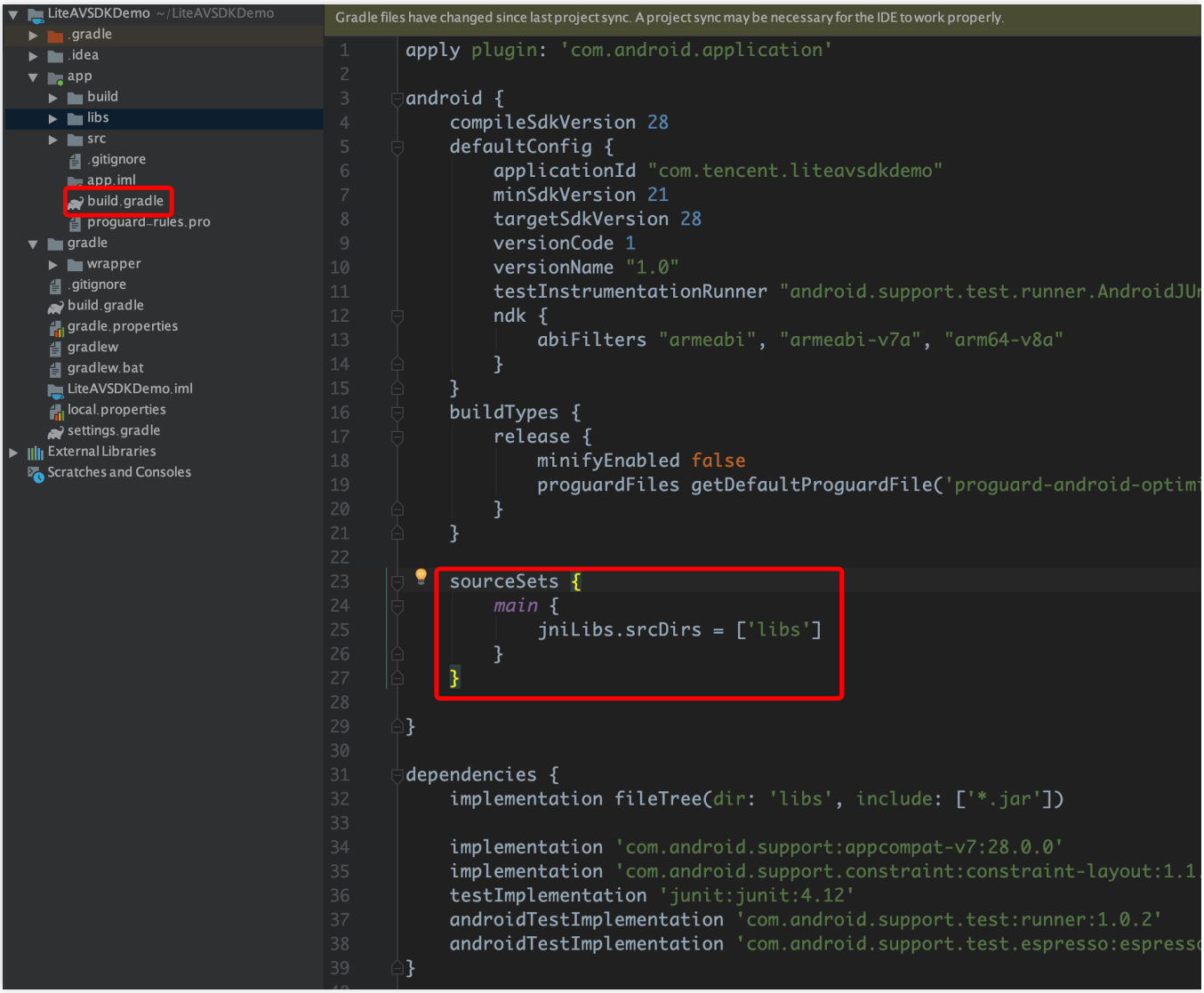


```
dependencies{
 implementation fileTree(dir:'libs',include:['*.jar'])
}
```

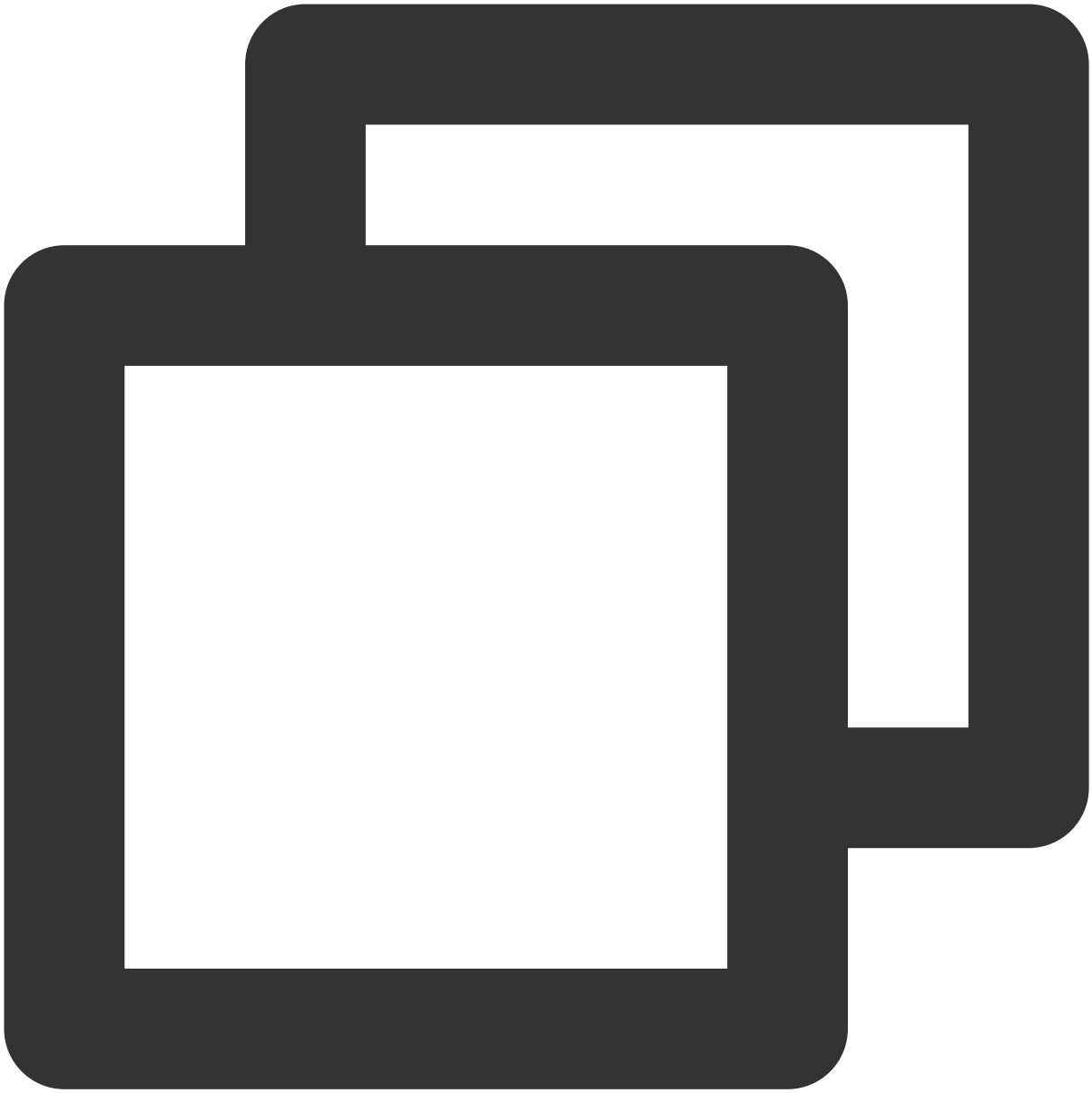
4. 在工程根目录下的 build.gradle 中，添加 **flatDir**，指定本地仓库路径。



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



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

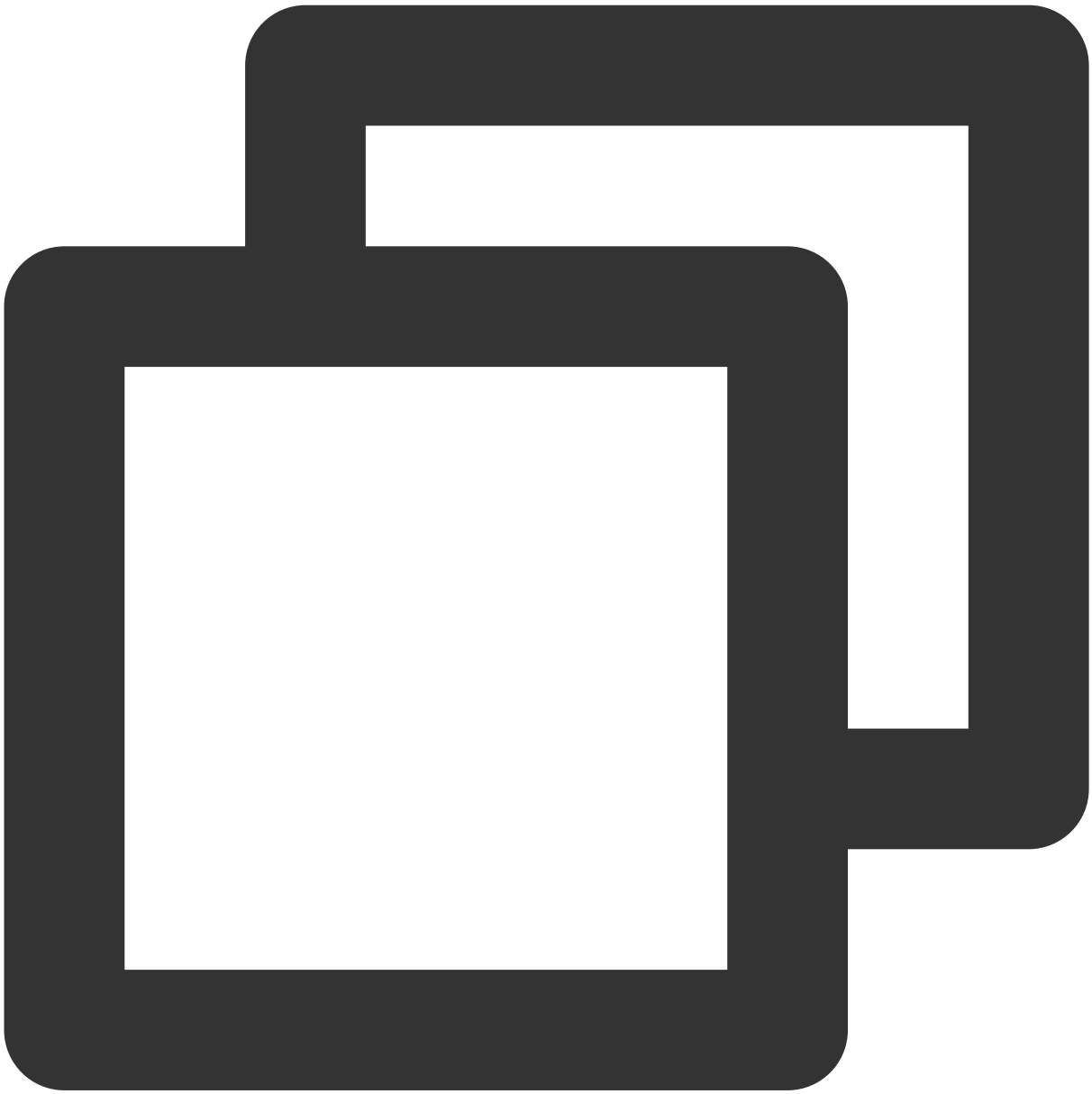


```
defaultConfig {
 ndk {
     abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
 }
}
```

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

配置 App 权限

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

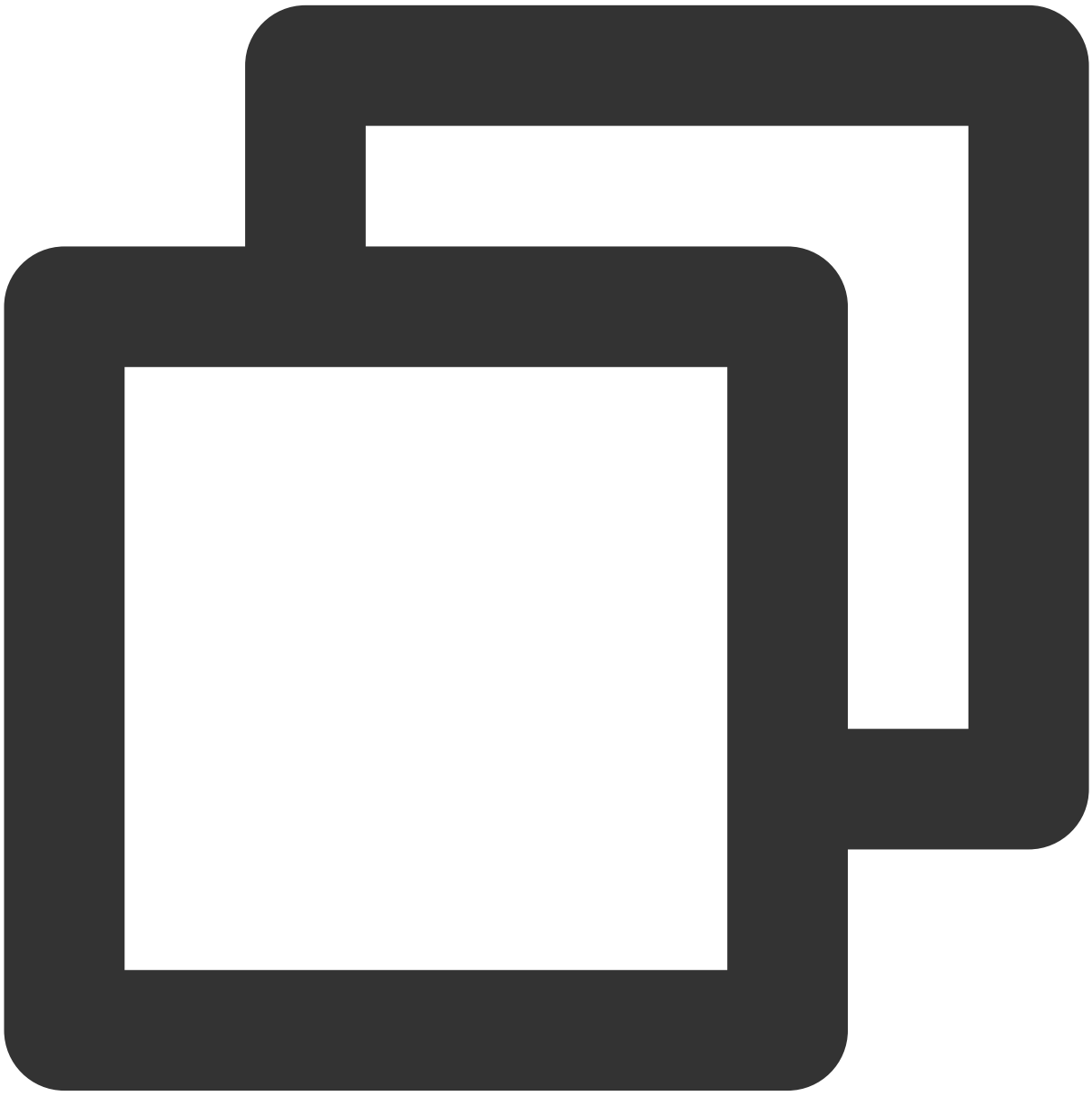


```
<!--网络权限-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

网络安全配置允许 App 发送 http 请求

出于安全考虑，从 Android P 开始，Google 要求 App 的请求都使用加密链接。播放器 SDK 会启动一个 localsever 代理 http 请求，如果您的应用 **targetSdkVersion 大于或等于28**，可以通过 网络安全配置 来开启允许向127.0.0.1发送 http 请求。 否则播放时将出现 "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" 错误， 导致无法播放视频。配置步骤如下：

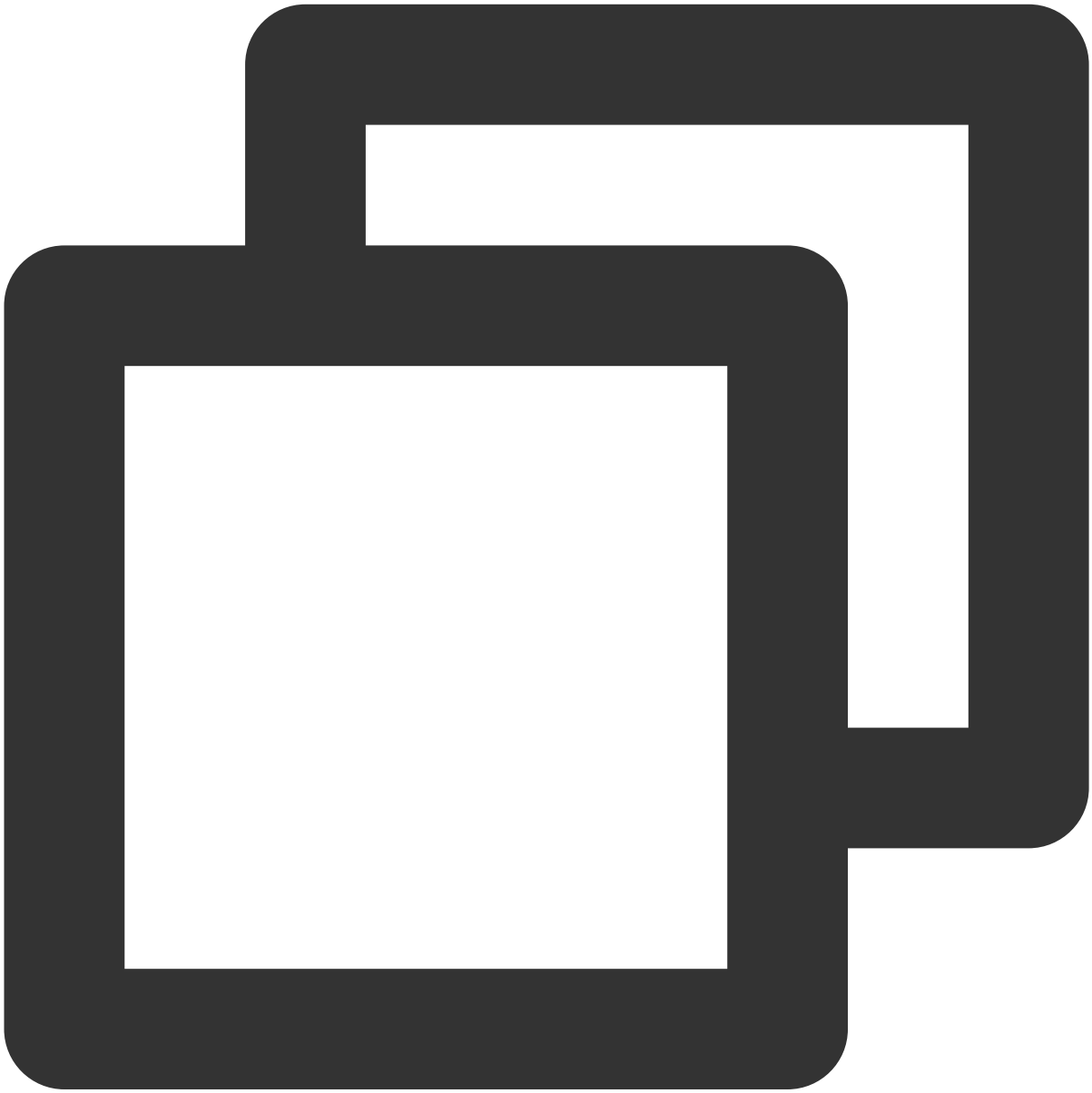
1. 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置。



```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
```

```
    </domain-config>
</network-security-config>
```

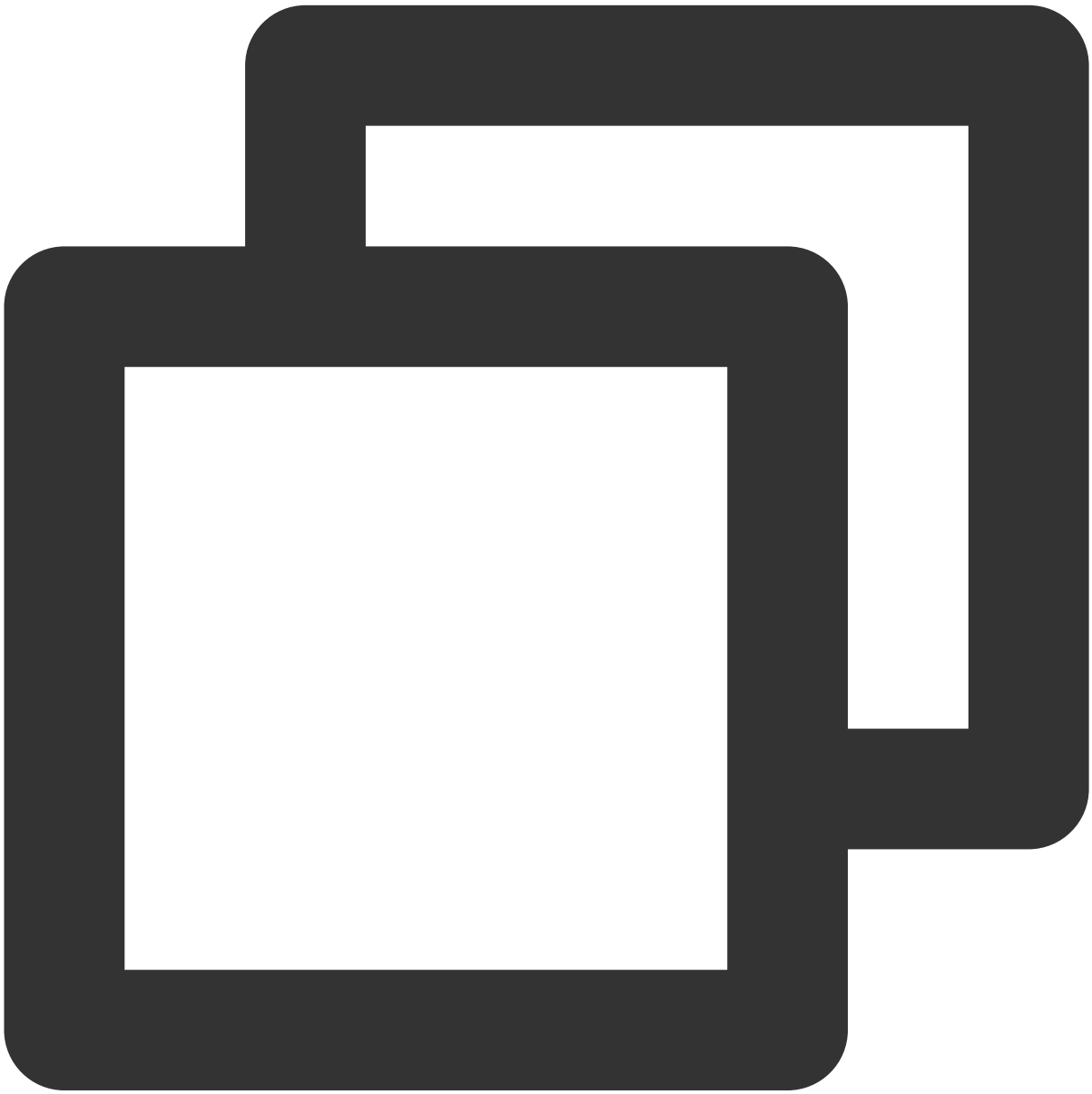
2. 在 AndroidManifest.xml 文件下的 application 标签增加以下属性。



```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

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.** { *; }
```

配置 License 授权

进入 云点播控制台 获取测试用 License，不配置 License 将会播放时视频失败，具体操作请参见 新增与续期 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。
获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

常见问题

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。

点播场景

最近更新时间：2024-08-07 15:16:42

准备工作

1. 为了您体验到更完整全面的播放器功能，建议您开通 云点播 相关服务，未注册用户可注册账号 试用。若您不使用云点播服务，可略过此步骤，但集成后仅可使用播放器基础能力。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

通过本文你可以学会

如何集成腾讯云 Android 播放器SDK
如何使用播放器 SDK 进行点播播放
如何使用播放器 SDK 底层能力实现更多功能

SDK 集成

步骤1：集成 SDK 开发包

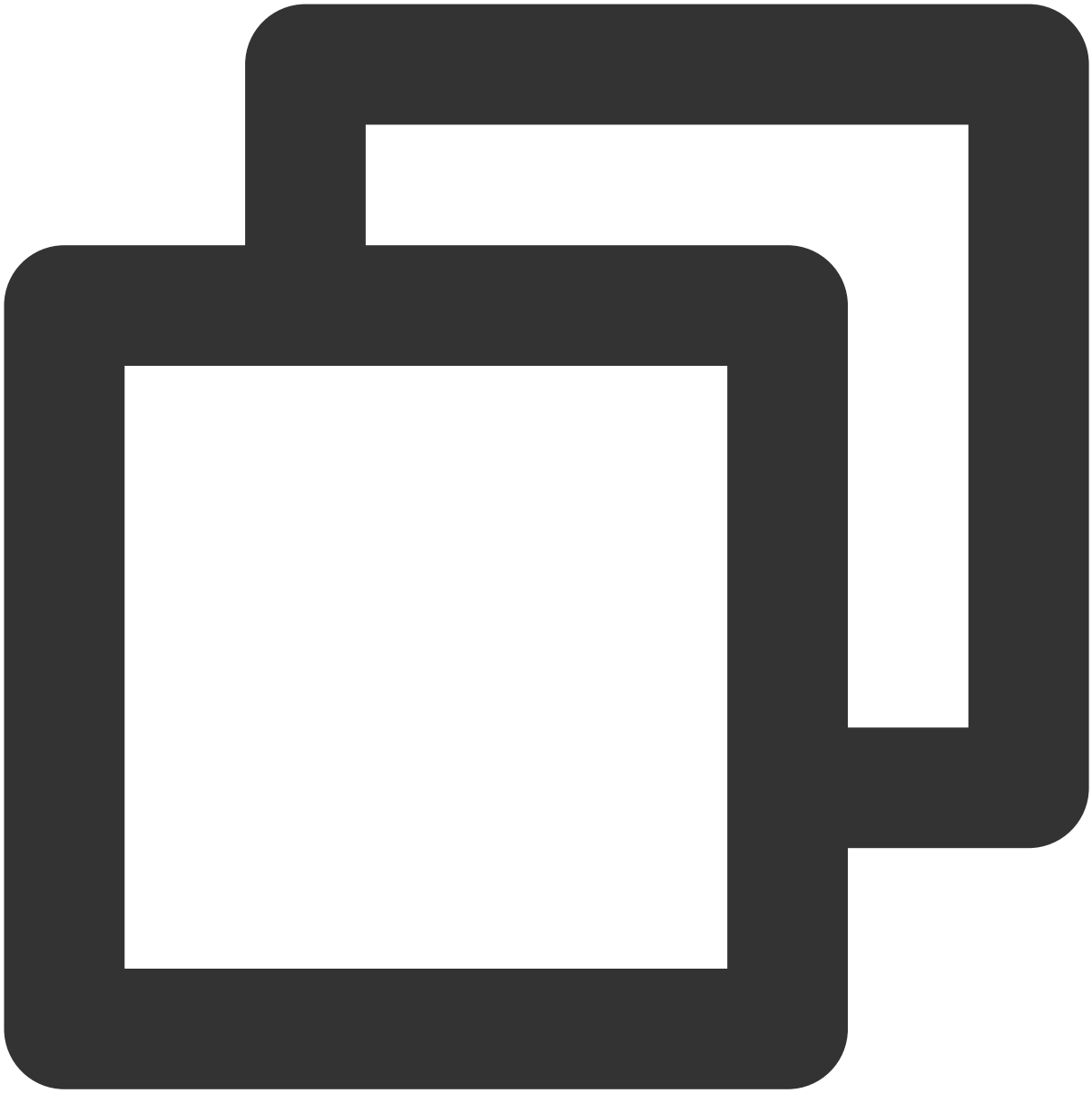
下载和集成 SDK 开发包，请参考同目录下的 SDK 集成指引。

步骤2：配置 License 授权

若您已获得相关 License 授权，需在 云点播控制台 获取 License URL 和 License Key。
若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。
获取到 License 信息后，在调用 SDK 的相关接口前，需要初始化配置 License，详细教程请参见 配置查看 License。

步骤3: 添加 View

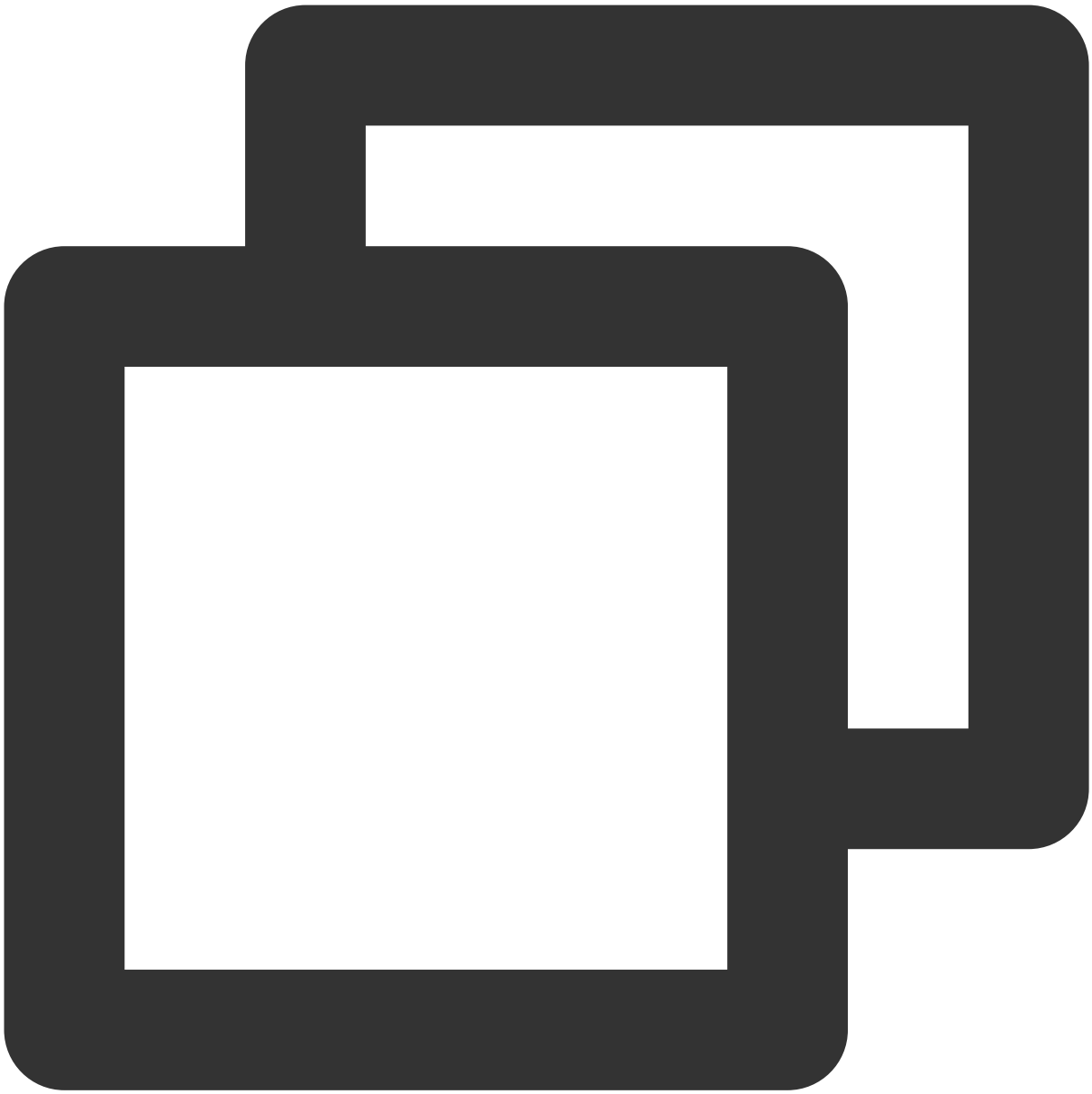
SDK 默认提供 TXCloudVideoView 用于视频渲染，我们第一步要做的就是在布局 xml 文件里加入如下一段代码：



```
<com.tencent.rtmp.ui.TXCloudVideoView
            android:id="@+id/video_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_centerInParent="true"
            android:visibility="gone"/>
```

步骤4：创建 Player

接下来创建一个 **TXVodPlayer** 的对象，并使用 setPlayerView 接口将它与我们刚添加到界面上的 **video_view** 控件进行关联。



```
//mPlayerView 即步骤3中添加的视频渲染 view
TXCloudVideoView mPlayerView = findViewById(R.id.video_view);
//创建 player 对象
TXVodPlayer mVodPlayer = new TXVodPlayer(getActivity());
//关联 player 对象与视频渲染 view
mVodPlayer.setPlayerView(mPlayerView);
```

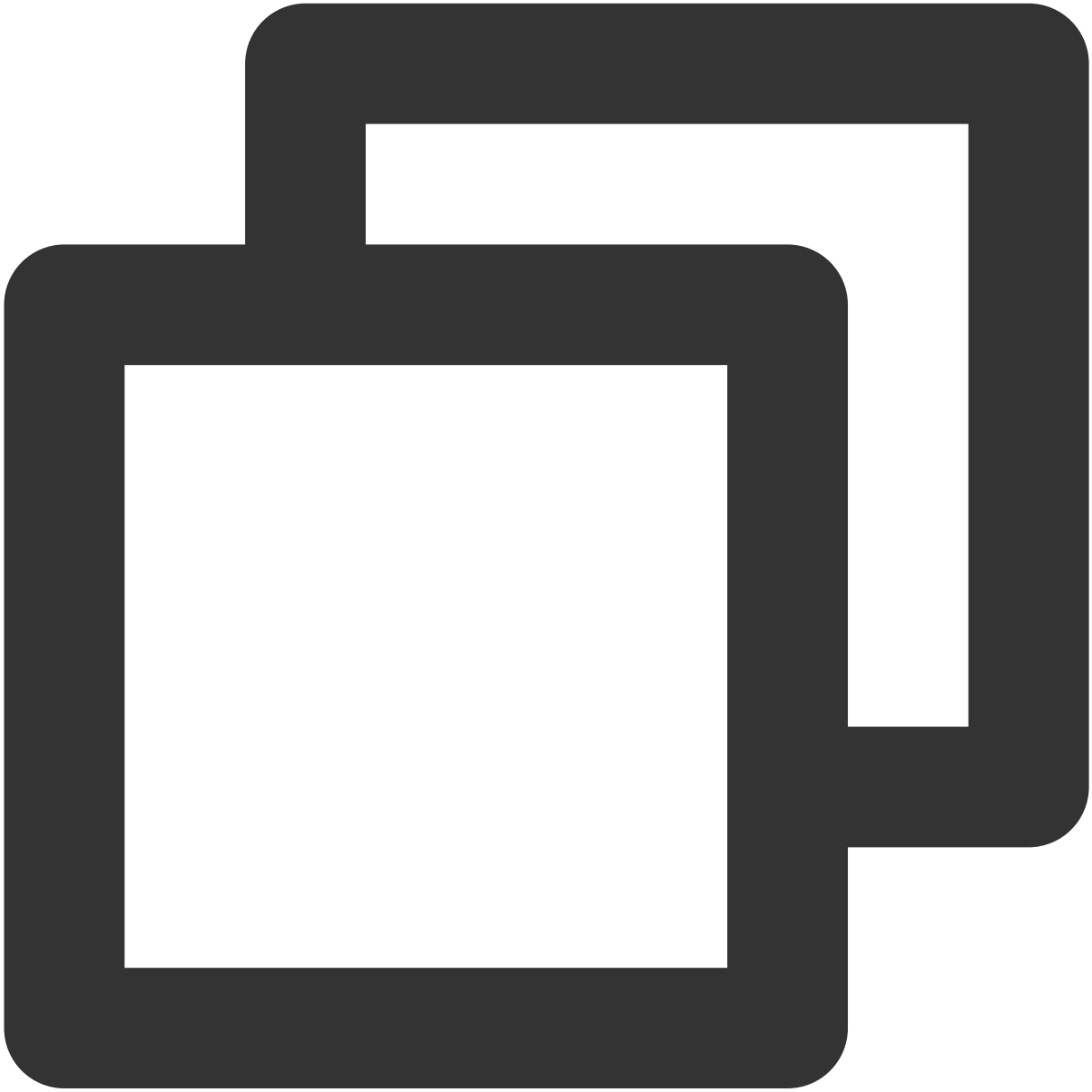
步骤5：启动播放

TXVodPlayer 支持两种播放模式，您可以根据需要自行选择：

通过 URL 方式

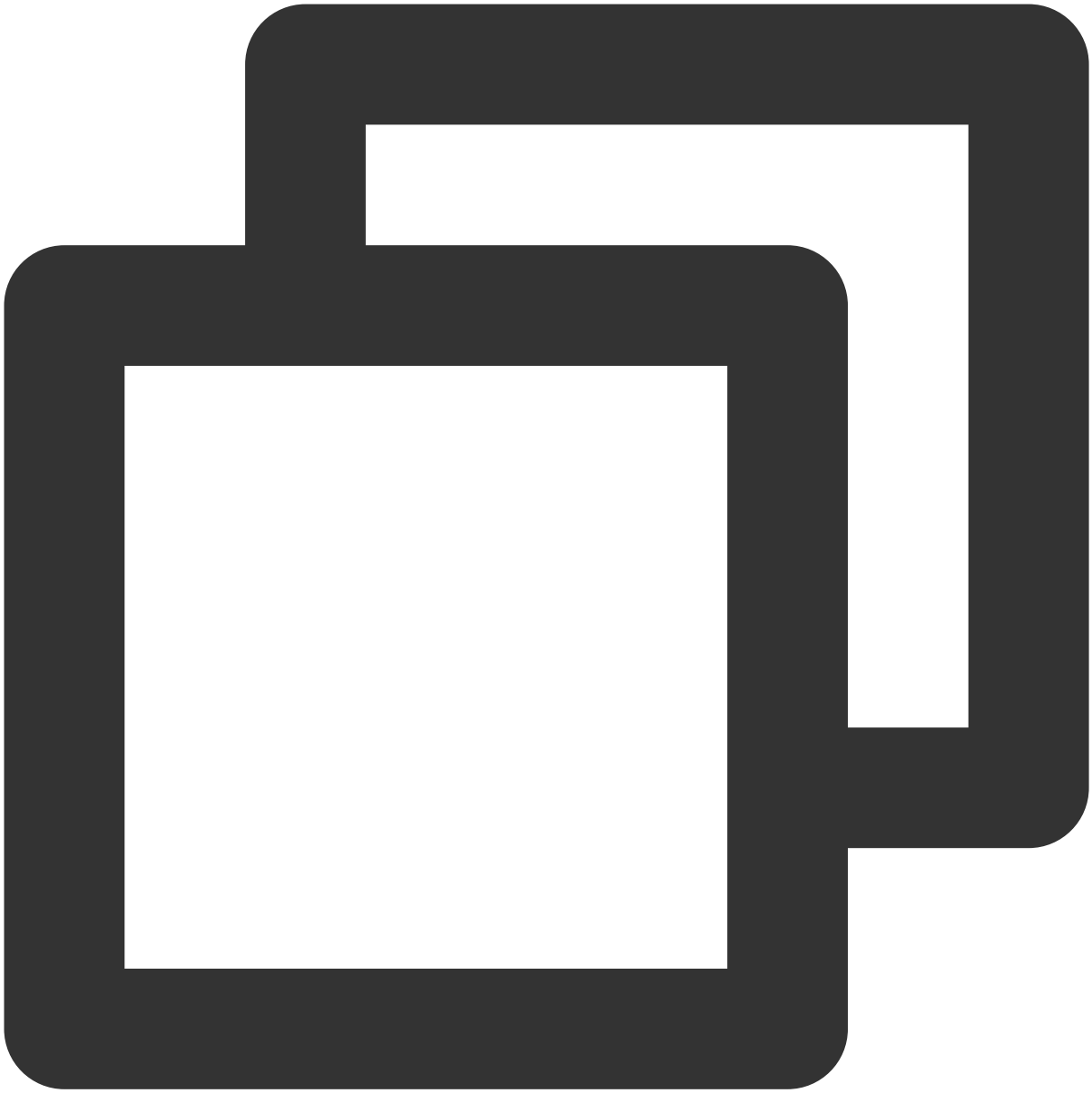
通过 FileId 方式

TXVodPlayer 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startVodPlay 函数即可。



```
// 播放 URL 视频资源
String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
mVodPlayer.startVodPlay(url);
```

```
// 播放本地视频资源
String localFile = "/sdcard/video.mp4";
mVodPlayer.startVodPlay(localFile);

// 从 11.8 版本开始支持播放器 content:// URI 视频资源和 asset 目录视频资源
// 播放 content:// URI 视频资源
String localFile = "content://xxx/xxx/video.mp4";
mVodPlayer.startVodPlay(localFile);

// 播放 asset 目录视频资源,  传入的地址必须以 asset:// 开头
String localFile = "asset://video.mp4";
mVodPlayer.startVodPlay(localFile);
```



```
// 推荐使用下面的新接口
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // 腾讯云账户的appI
    "4564972819220421305", // 视频的fileId
    "psignxxxxxxx"); // 播放器签名
mVodPlayer.startVodPlay(playInfoParam);

// 旧接口，不推荐使用
TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
authBuilder.setAppId(1252463788);
authBuilder.setFileId("4564972819220421305");
```

```
mVodPlayer.startVodPlay(authBuilder);
```

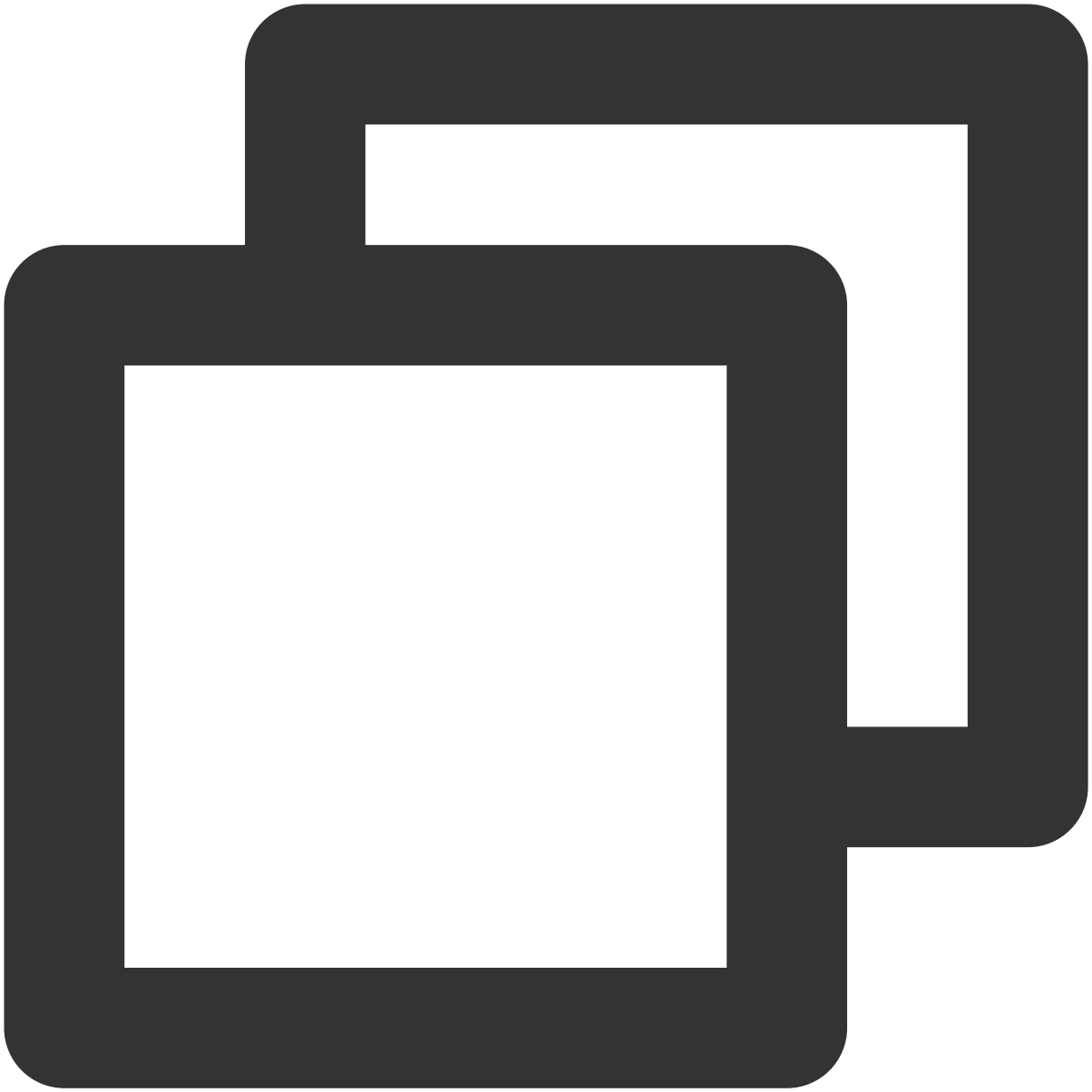
在 媒资管理 找到对应的视频文件。在文件名下方可以看到 FileId。

通过 FileId 方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或 FileId 不存在，则会收到 `TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL` 事件，反之收到 `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` 表示请求成功。

步骤6：结束播放

结束播放时**记得销毁 view 控件**，尤其是在下次 startVodPlay 之前，否则会产生大量的内存泄露以及闪屏问题。

同时，在退出播放界面时，记得一定要调用渲染 View 的 `onDestroy()` 函数，否则可能会产生内存泄露和“Receiver not registered”报警。



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

说明：

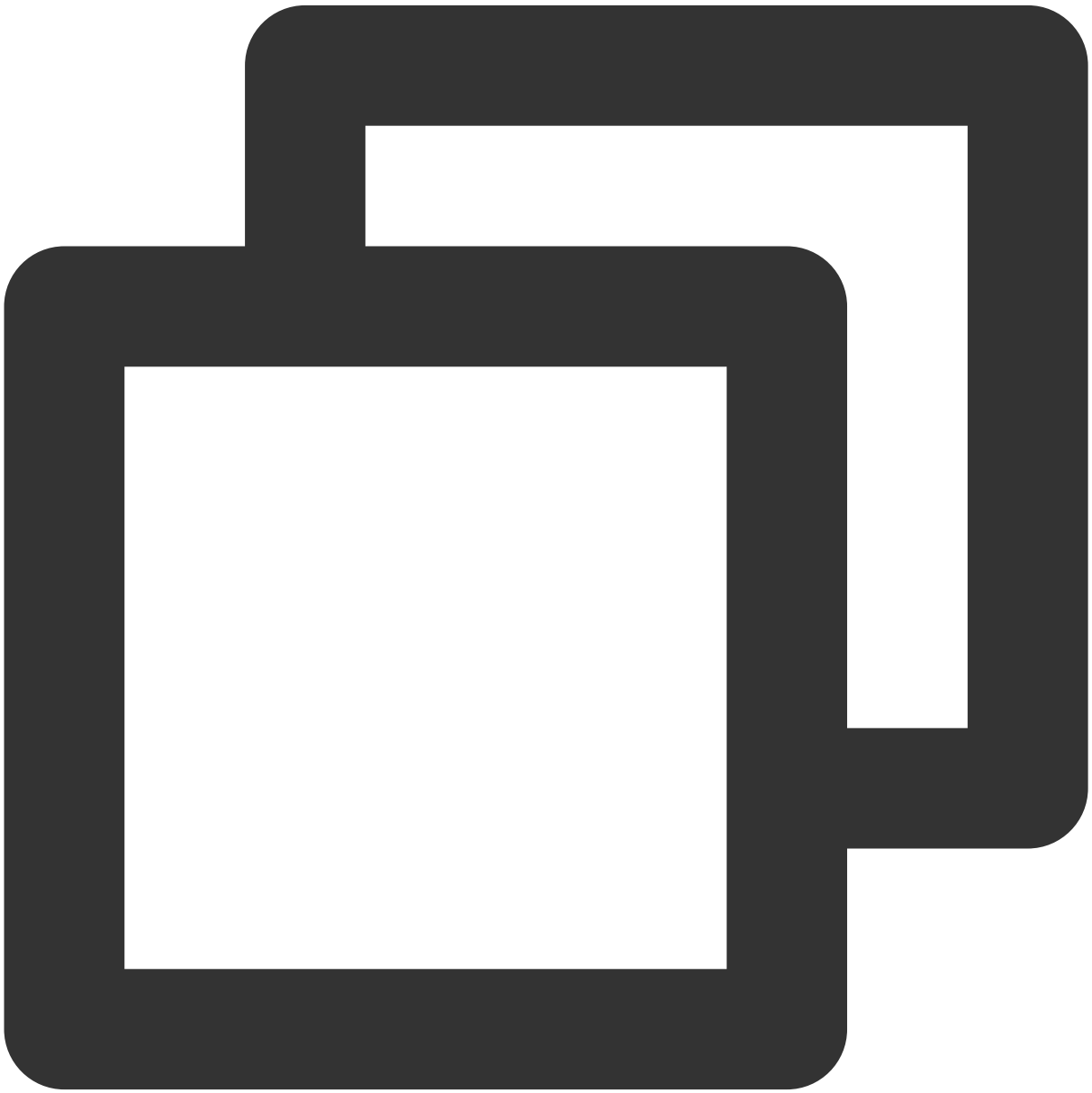
stopPlay 的布尔型参数含义为：“是否清除最后一帧画面”。早期版本的 RTMP SDK 的直播播放器没有 pause 的概念，所以通过这个布尔值来控制最后一帧画面的清除。

如果是点播播放结束后，也想保留最后一帧画面，您可以在收到播放结束事件后什么也不做，默认停在最后一帧。

基础功能使用

1、播放控制

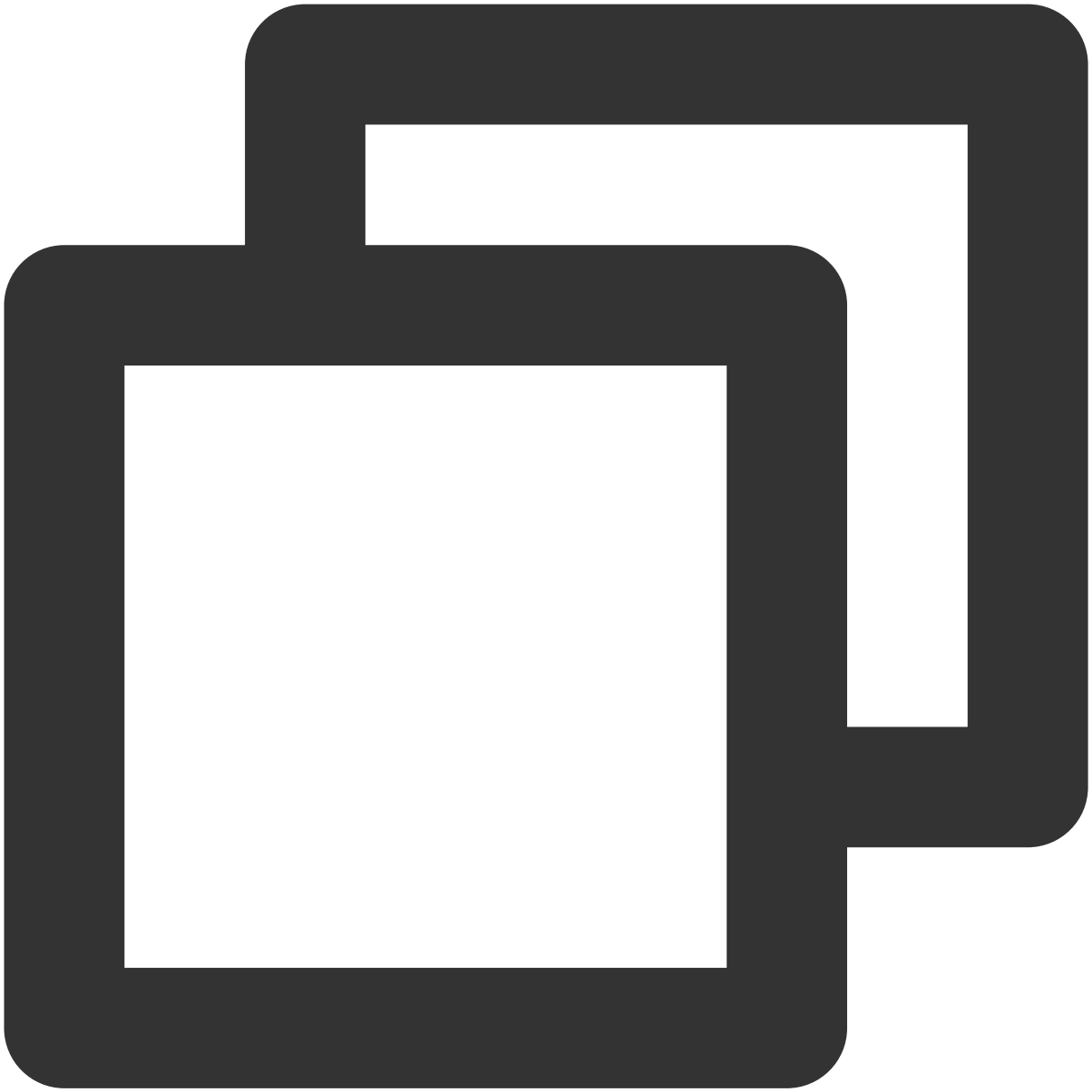
开始播放



```
// 开始播放
```

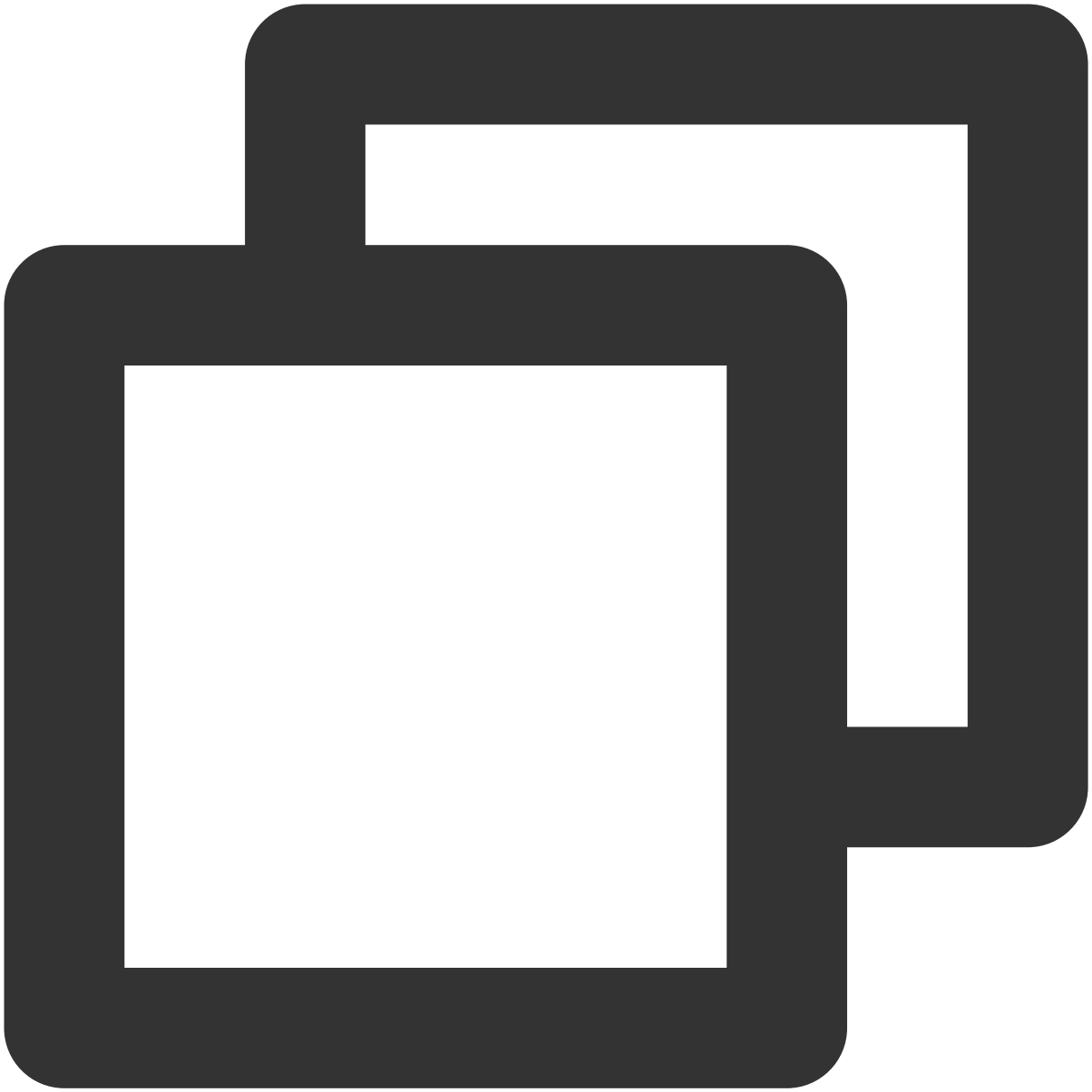
```
mVodPlayer.startVodPlay(url)
```

暂停播放



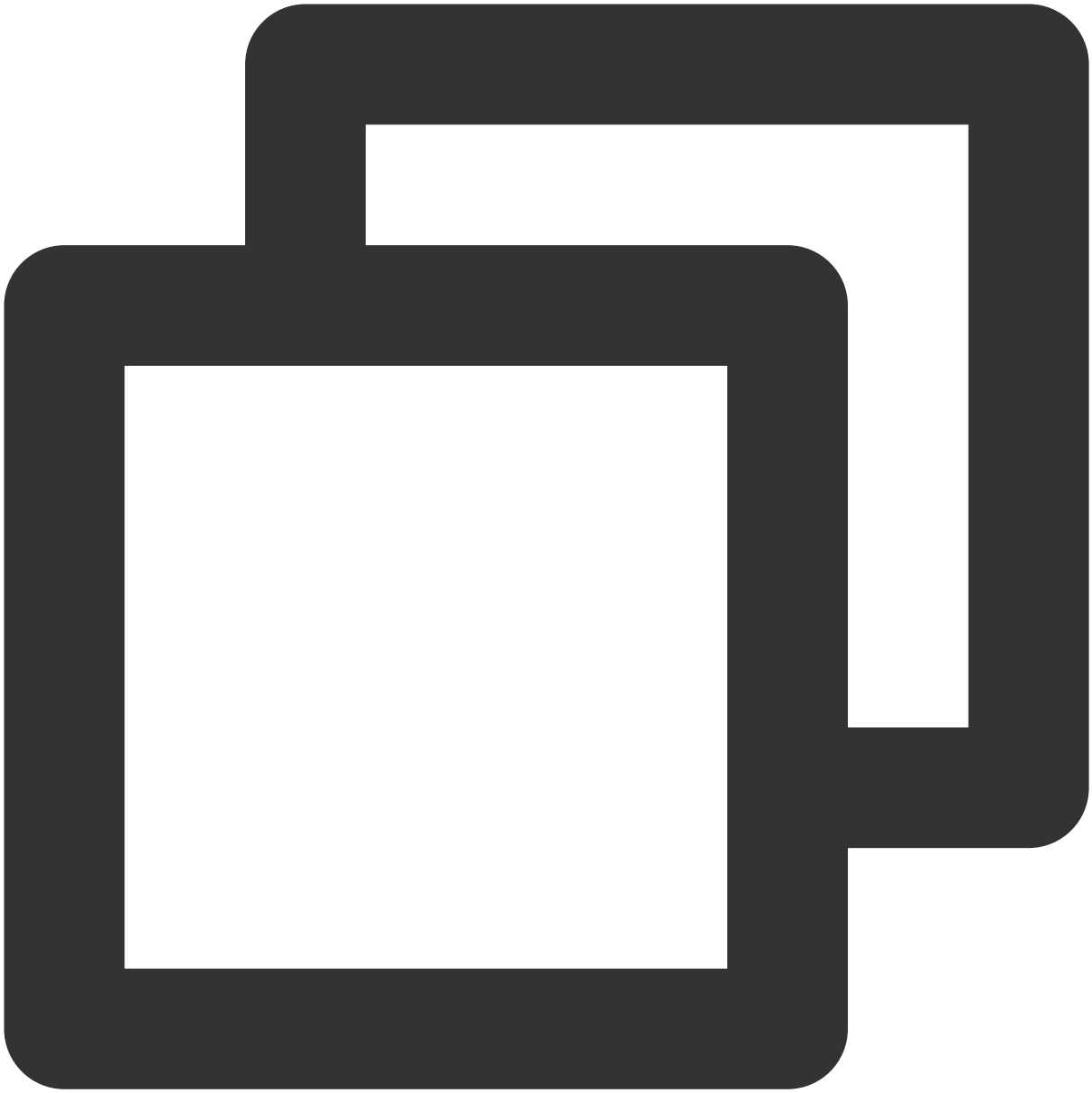
```
// 暂停播放
mVodPlayer.pause();
```

恢复播放



```
// 恢复播放
mVodPlayer.resume();
```

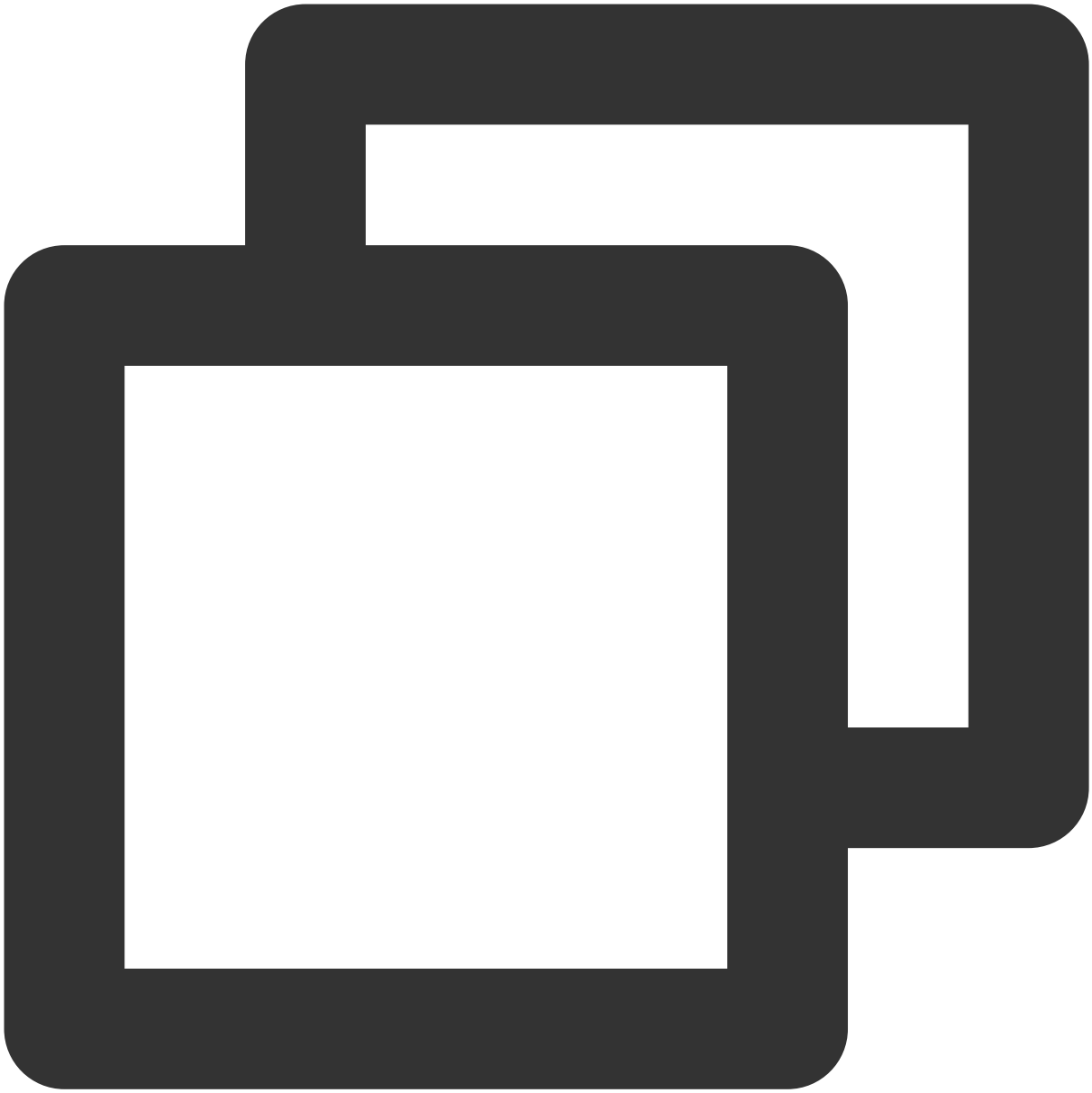
结束播放



```
// 结束播放
mVodPlayer.stopPlay(true);
```

调整进度（Seek）

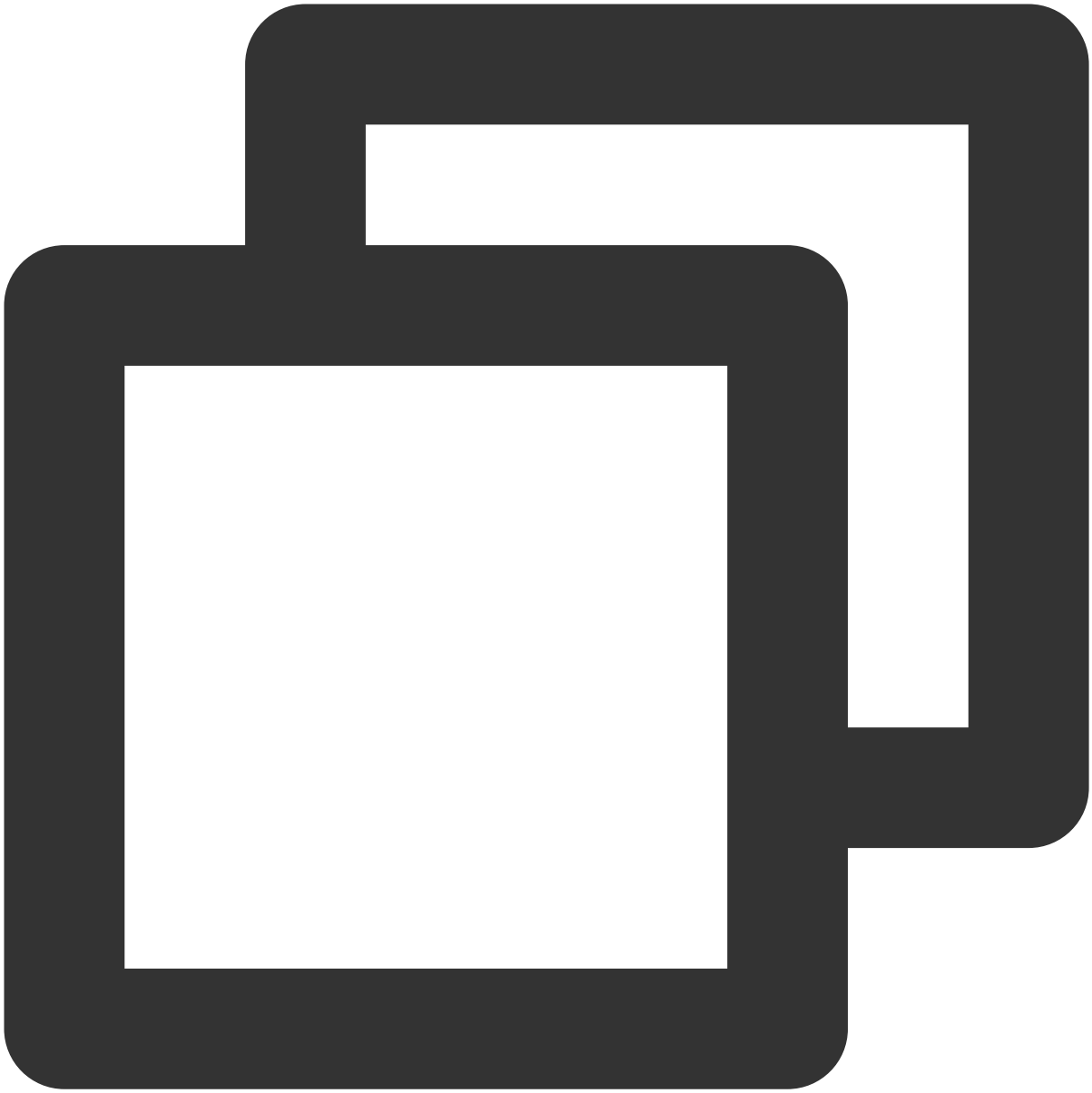
当用户拖拽进度条时，可调用 seek 从指定位置开始播放，播放器 SDK 支持精准 seek。



```
int time = 600; // int类型时，单位为 秒
// float time = 600; // float 类型时单位为 秒
// 调整进度
mVodPlayer.seek(time);
```

精准和非精准 Seek

播放器 SDK 11.8 版本开始，支持调用 seek 接口时，指定精准或非精准 seek。



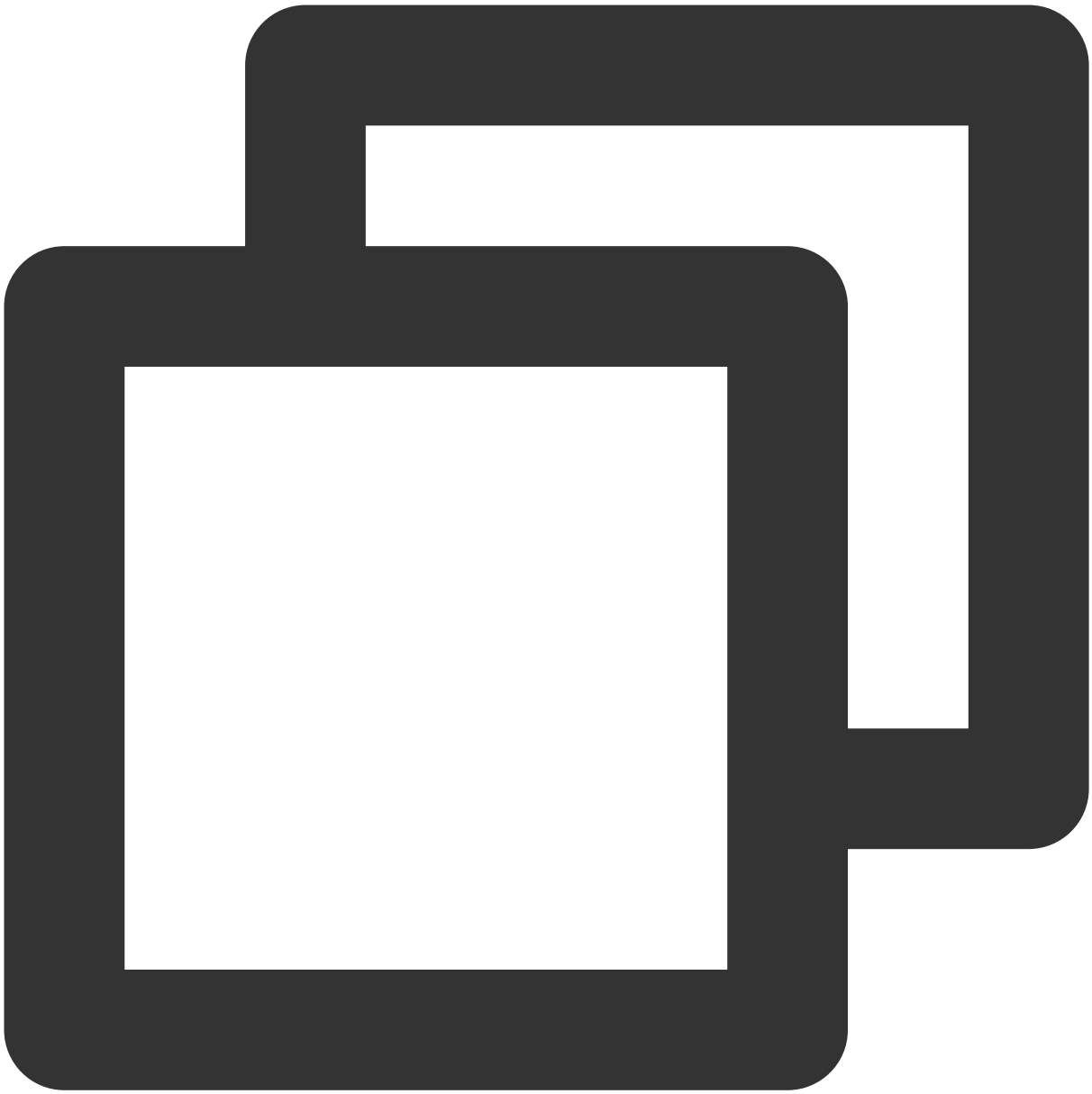
```
float time = 600; // float 类型时单位为 秒
// 调整进度
mVodPlayer.seek(time, true);  // 精准 seek
mVodPlayer.seek(time, false);  // 非精准 seek
```

Seek 到视频流指定 PDT 时间点

跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。

注意：

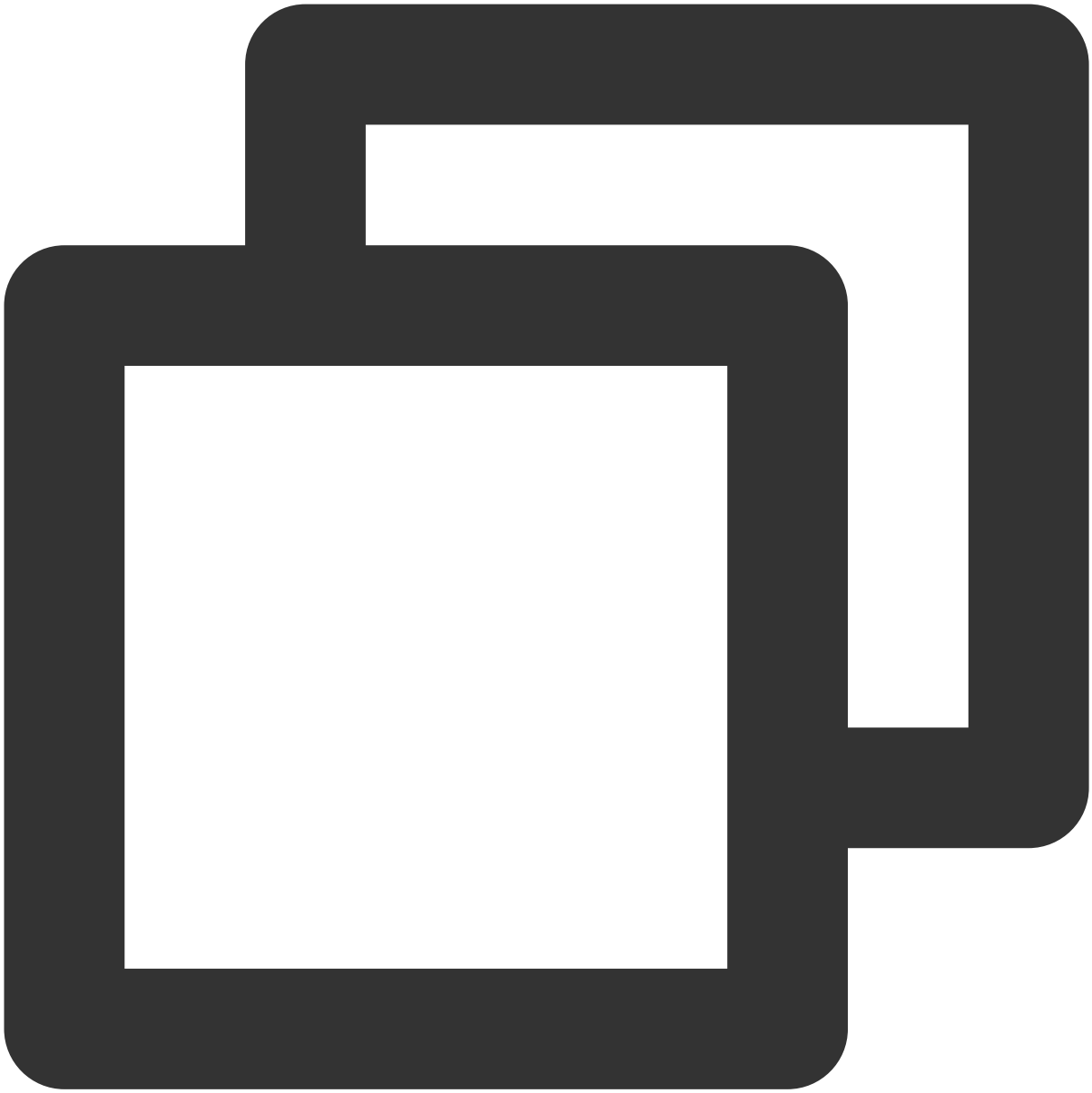
此功能为播放器高级版 11.6 版本开始支持。



```
long pdtTimeMs = 600; // 单位为 毫秒
mVodPlayer.seekToPdtTime(time);
```

从指定时间开始播放

首次调用 startVodPlay 之前，支持从指定时间开始播放。



```
float startTimeInSecond = 60; // 单位：秒
mVodPlayer.setStartTime(startTimeInSecond);    // 设置开始播放时间
mVodPlayer.startVodPlay(url);
```

2、画面调整

view：大小和位置

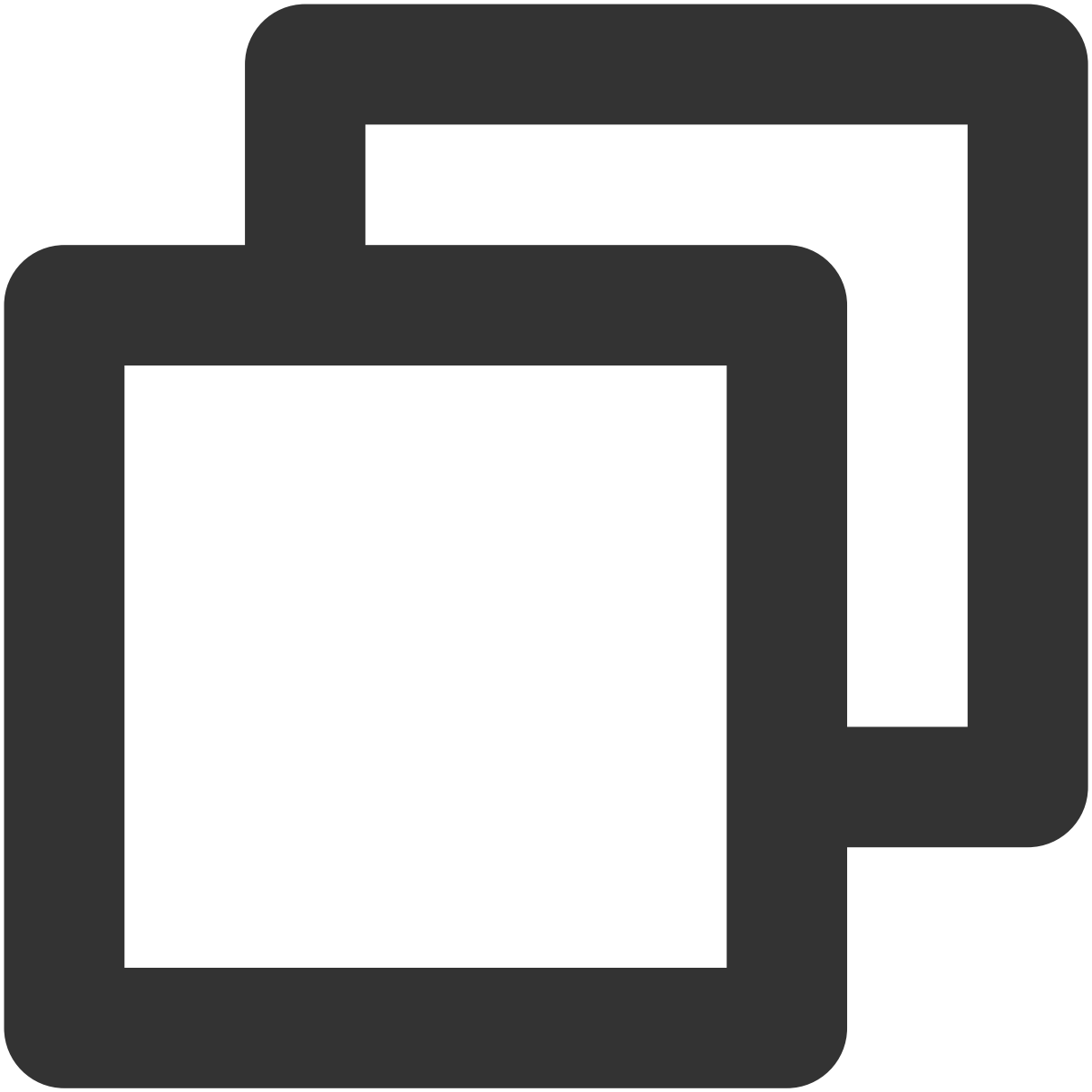
如需修改画面的大小及位置，直接调整 SDK 集成时 添加 View 中添加的 “video_view” 控件的大小和位置即可。

setRenderMode：铺满或适应

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

setRenderRotation：画面旋转

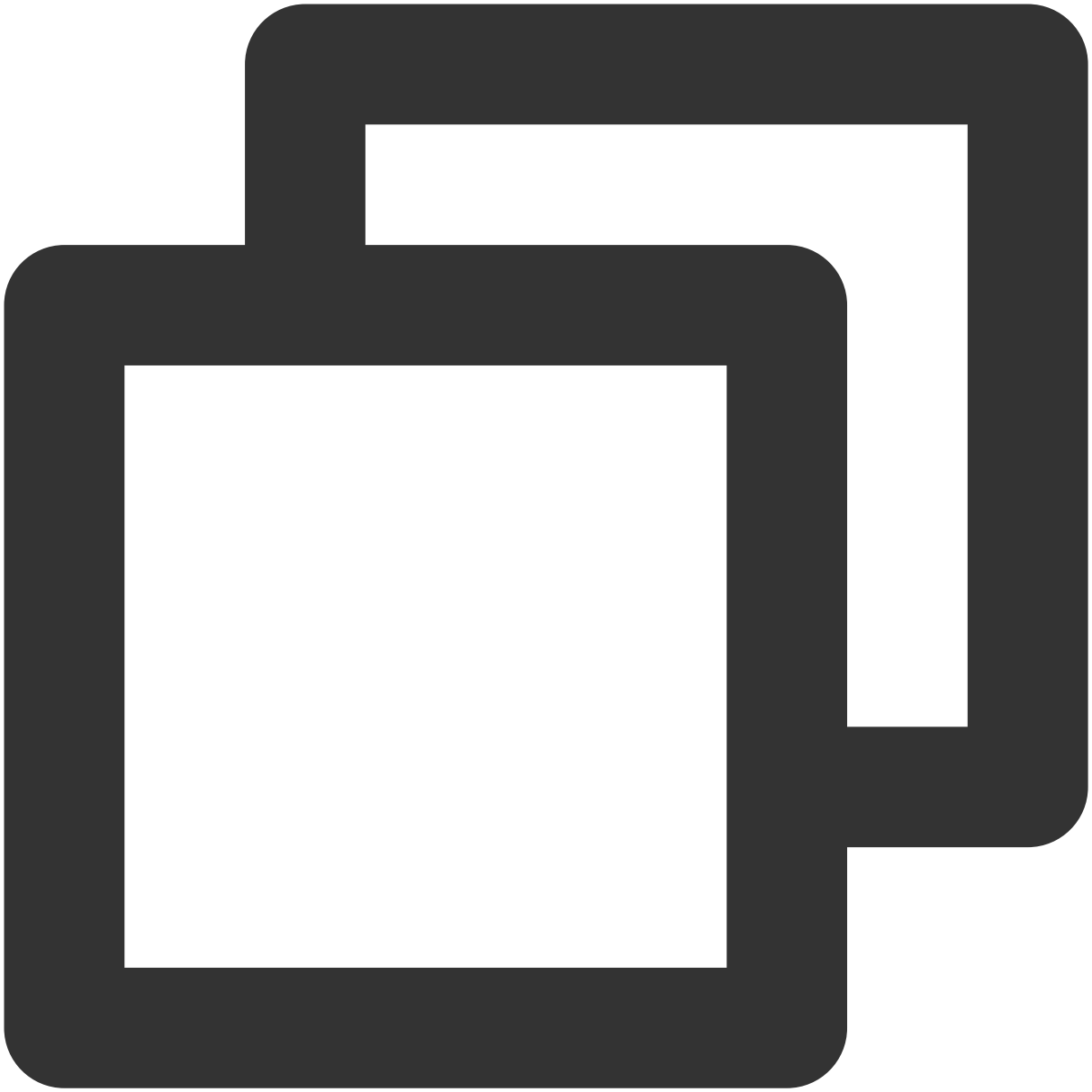
可选值	含义
RENDER_ROTATION_PORTRAIT	正常播放（Home 键在画面正下方）
RENDER_ROTATION_LANDSCAPE	画面顺时针旋转270度（Home 键在画面正左方）



```
 // 将图像等比例铺满整个屏幕
mVodPlayer.setRenderMode(TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN);
// 正常播放（Home 键在画面正下方）
mVodPlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
```

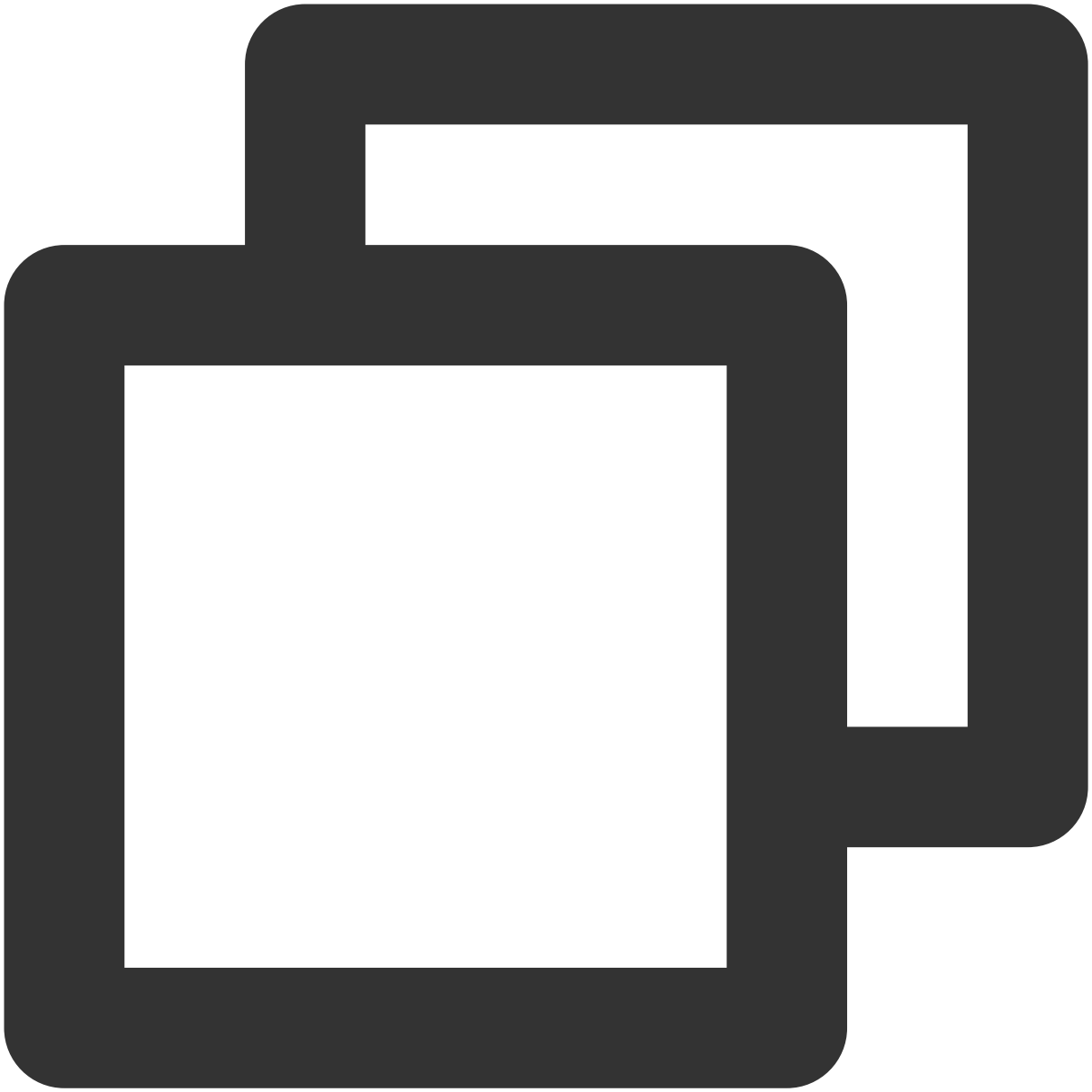
3、变速播放

点播播放器支持变速播放，通过接口 `setRate` 设置点播播放速率来完成，支持快速与慢速播放，如0.5X、1.0X、1.2X、2X等。



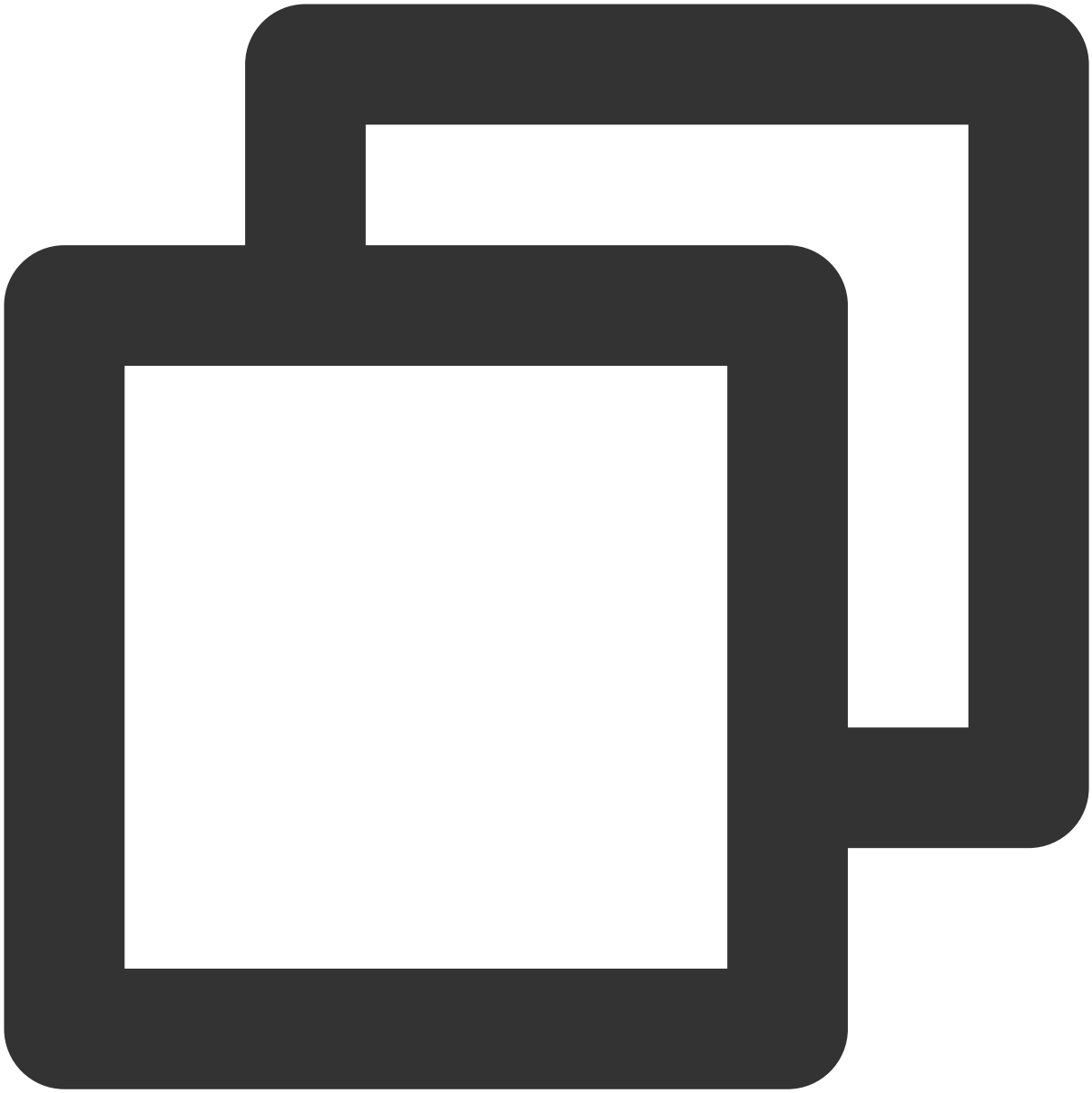
```
// 设置1.2倍速播放
mVodPlayer.setRate(1.2);
```

4、循环播放



```
// 设置循环播放
mVodPlayer.setLoop(true);
// 获取当前循环播放状态
mVodPlayer.isLoop();
```

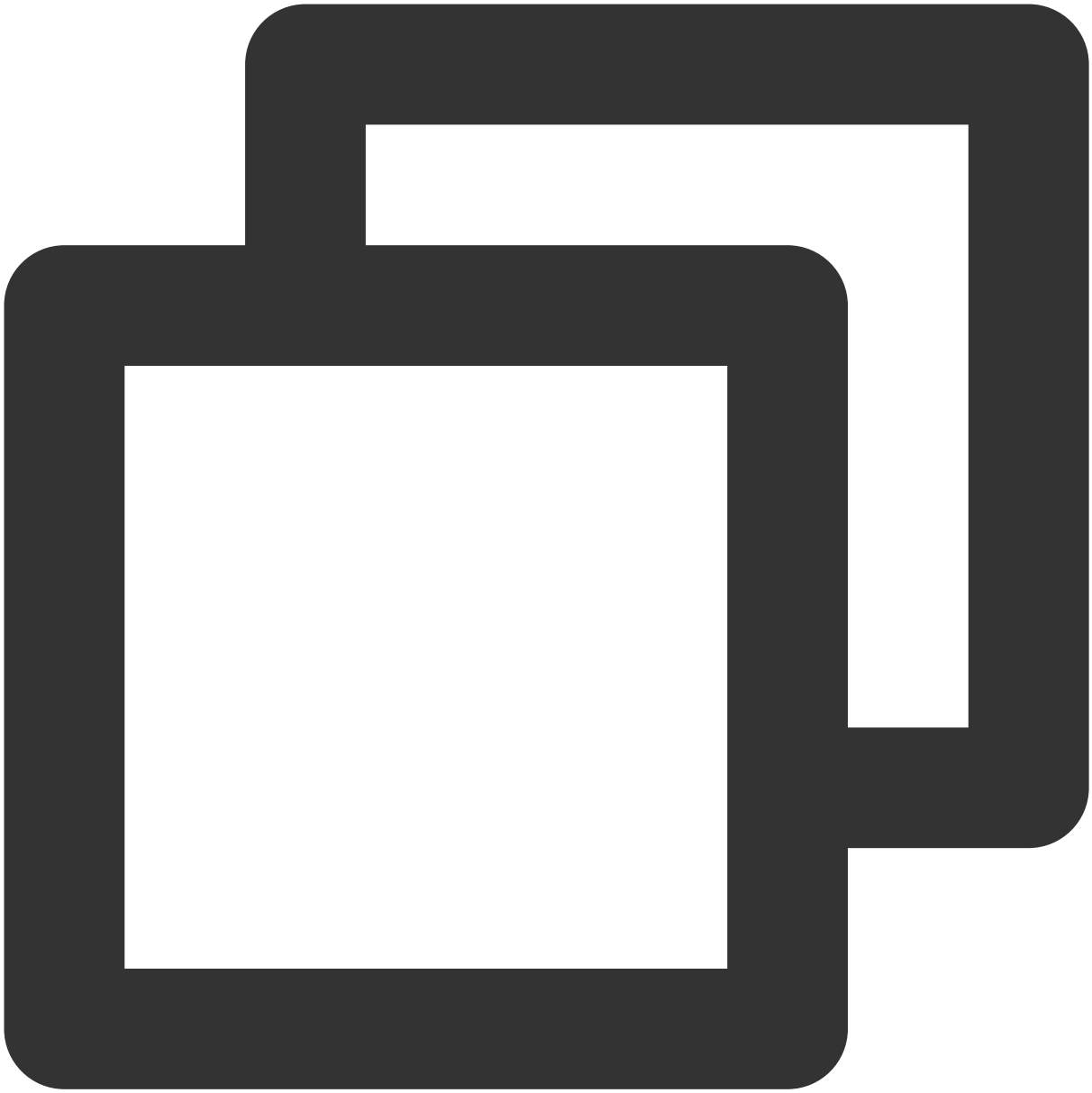
5、静音设置



```
// 设置静音，true 表示开启静音， false 表示关闭静音
mVodPlayer.setMute(true);
```

6、屏幕截图

通过调用 **snapshot** 您可以截取当前视频为一帧画面，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。



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

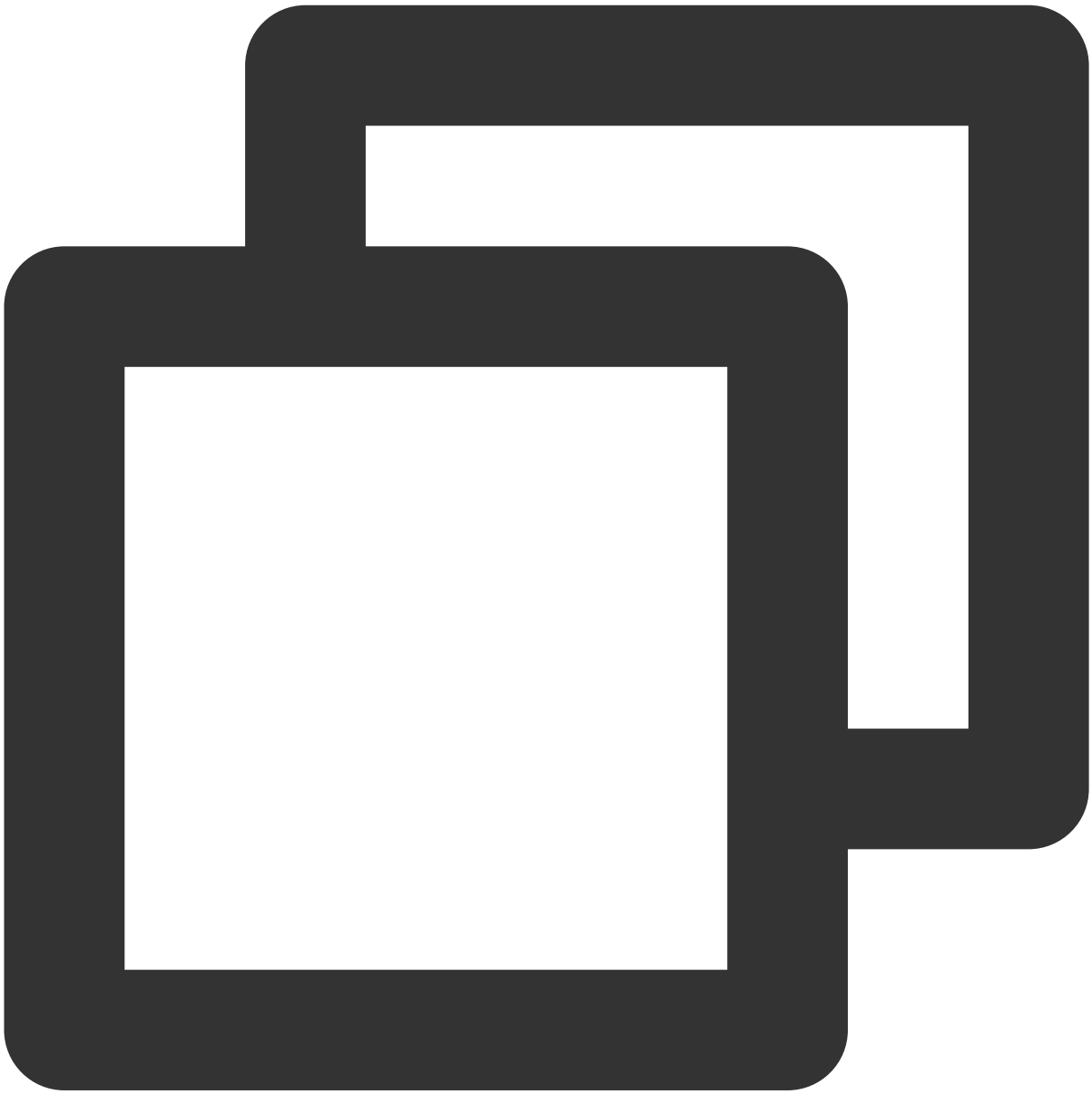
7、贴片广告

播放器SDK支持在界面添加图片贴片，用于广告宣传等。实现方式如下：

将 `autoPlay` 为 NO，此时播放器会正常加载，但视频不会立刻开始播放。

在播放器加载出来后、视频尚未开始时，即可在播放器界面查看图片贴片广告。

待达到广告展示结束条件时，使用resume接口启动视频播放。

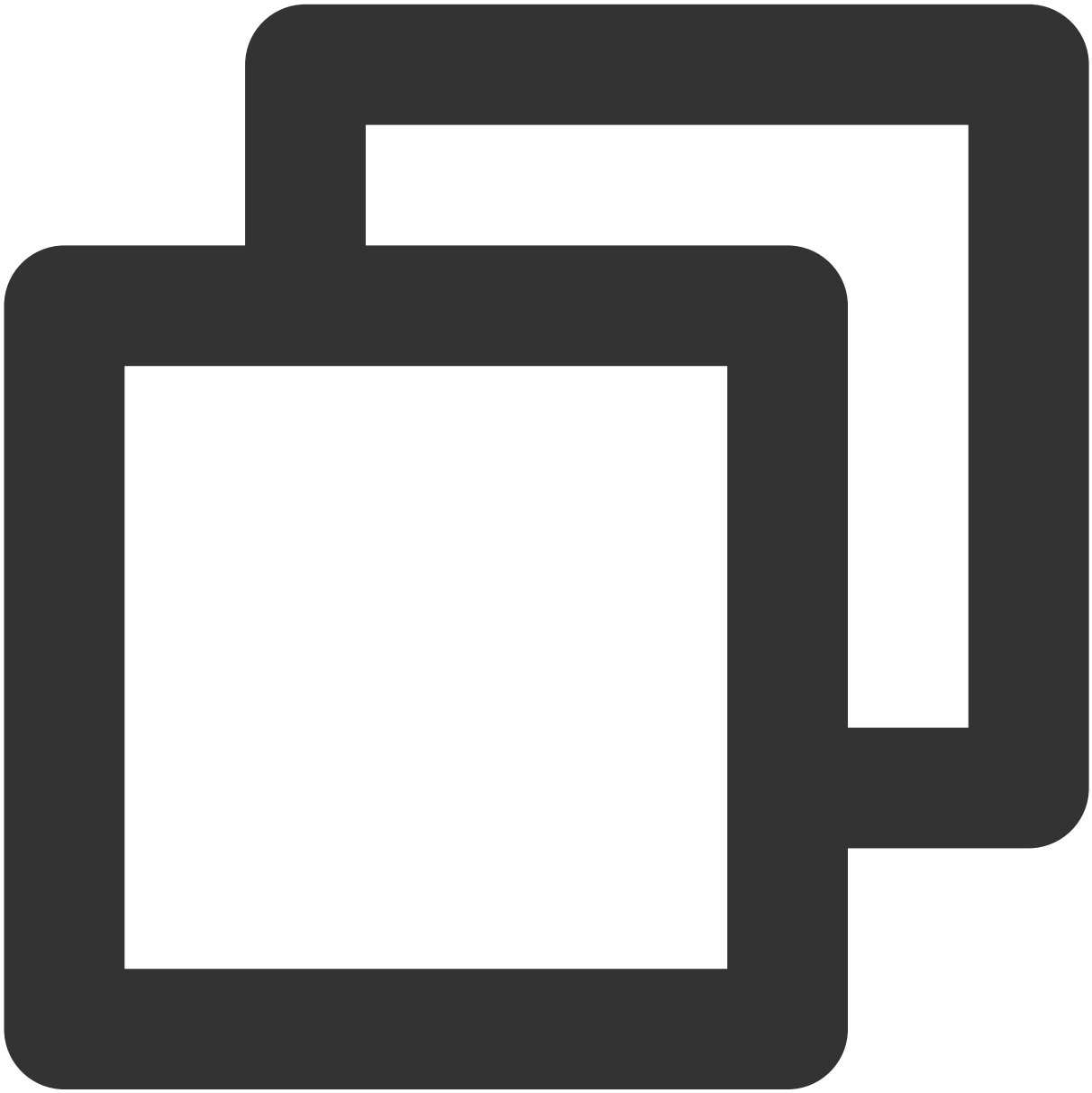


```
mVodPlayer.setAutoPlay(false);   // 设置为非自动播放
mVodPlayer.startVodPlay(url);     // startVodPlay 后会加载视频，加载成功后不会自动播放
// ......
// 在播放器界面上展示广告
```

```
// ......
mVodPlayer.resume();  // 广告展示完调用 resume 开始播放视频
```

8、HTTP-REF

TXVodPlayConfig 中的 headers 可以用来设置 HTTP 请求头，例如常用的防止 URL 被到处拷贝的 Referer 字段（腾讯云可以提供更加安全的签名防盗链方案），以及用于验证客户端身份信息的 Cookie 字段。

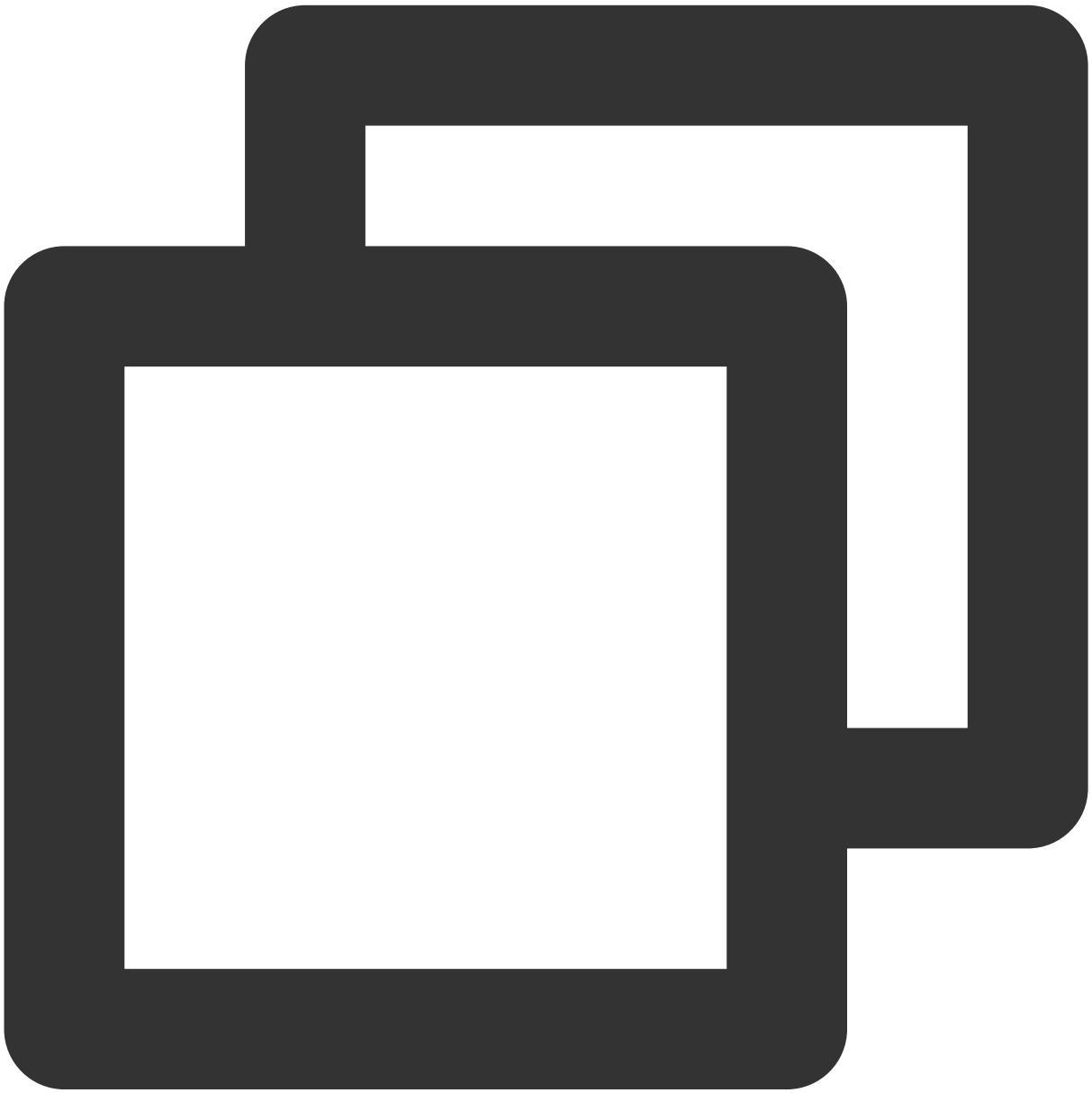


```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
Map<String, String> headers = new HashMap<>();
headers.put("Referer", "${Refer Content}");
```

```
mPlayConfig.setHeaders(headers);
mVodPlayer.setConfig(mPlayConfig);
```

9、硬件加速

对于蓝光级别（1080p）的画质，简单采用软件解码的方式很难获得较为流畅的播放体验，所以如果您的场景是以游戏直播为主，一般都推荐开启硬件加速。
软解和硬解的切换需要在切换之前先 **stopPlay**，切换之后再 **startVodPlay**，否则会产生比较严重的花屏问题。

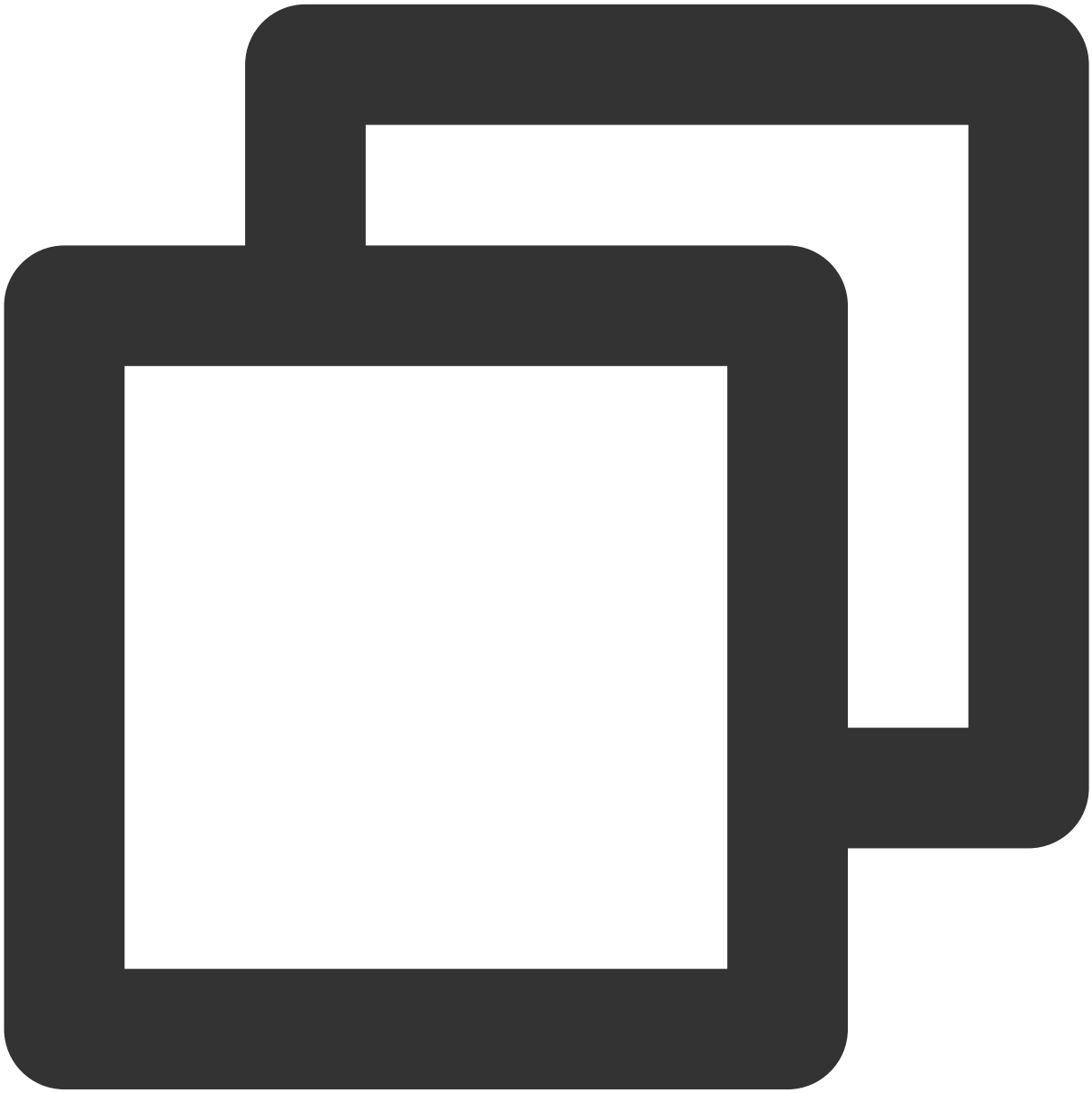


```
mVodPlayer.stopPlay(true);
mVodPlayer.enableHardwareDecode(true);
```

```
mVodPlayer.startVodPlay(flvUrl, type);
```

10、清晰度设置

SDK 支持 HLS 的多码率格式，方便用户切换不同码率的播放流，从而达到播放不同清晰的目标。可以通过下面方法进行清晰度设置。



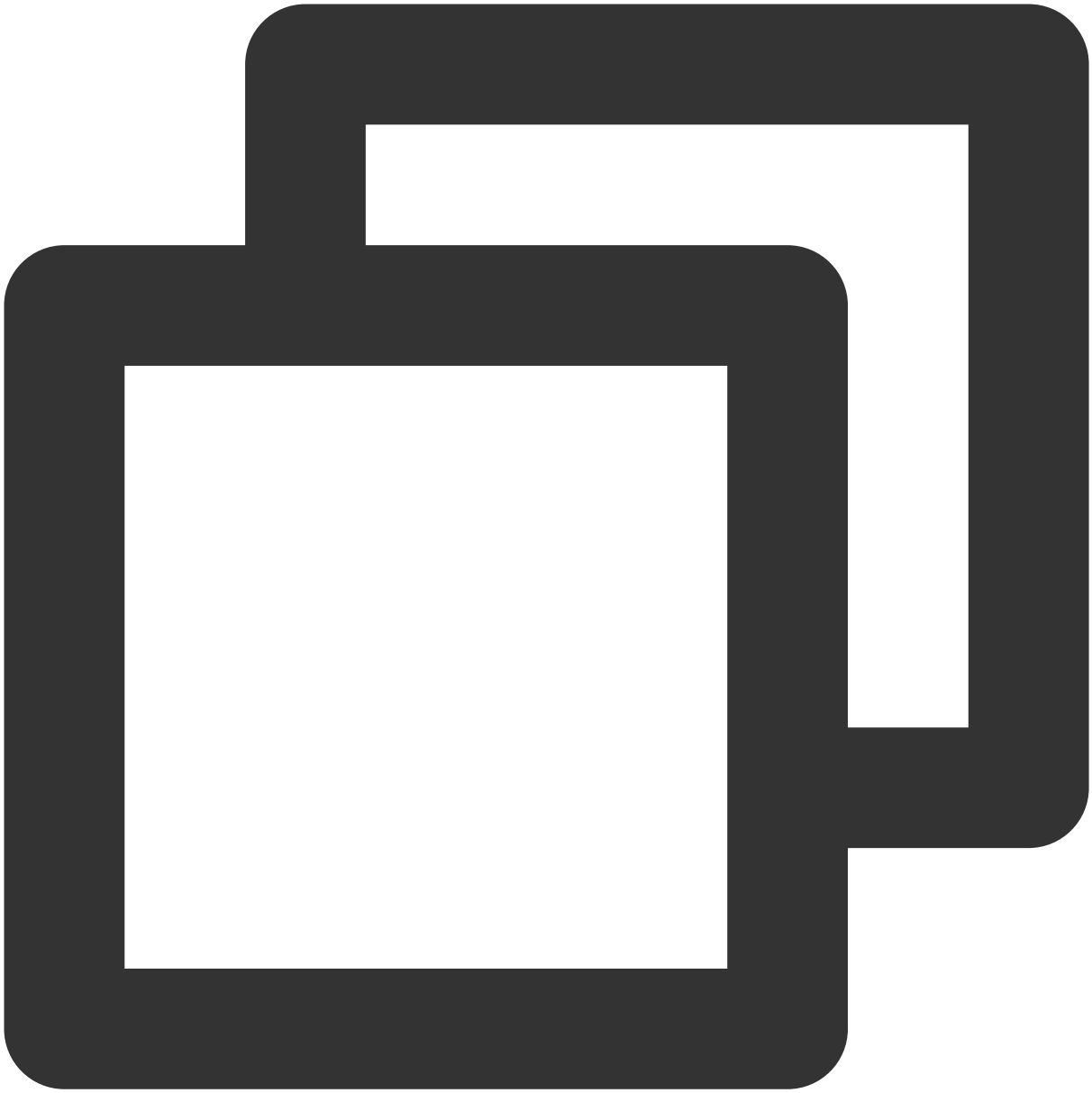
```
// 获取多码率数组，TXBitrateItem 类字段含义：index-码率下标；width-视频宽；height-视频高；bi
ArrayList<TXBitrateItem> bitrates = mVodPlayer.getSupportedBitrates();
int index = bitrates.get(i).index;  // 指定要播的码率下标
mVodPlayer.setBitrateIndex(index);  // 切换码率到想要的清晰度
```

```
// 获取当前播放的码率下标，返回值 -1000 为默认值，表示没有设置过码率标；返回值 -1 表示开启了自适应
int index = mVodPlayer.getBitrateIndex();
```

在播放过程中，可以随时通过 `mVodPlayer.setBitrateIndex(int)` 切换码率。切换过程中，会重新拉取另一条流的数据，SDK 针对腾讯云的多码率文件做过优化，可以做到切换无卡顿。
如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率，从而避免播放后切换码流。详细方法参考 播放器配置#启播前指定分辨率。

11、码流自适应

SDK 支持 HLS 的多码流自适应，开启相关能力后播放器能够根据当前带宽，动态选择最合适的码率播放。可以通过下面方法开启码流自适应。

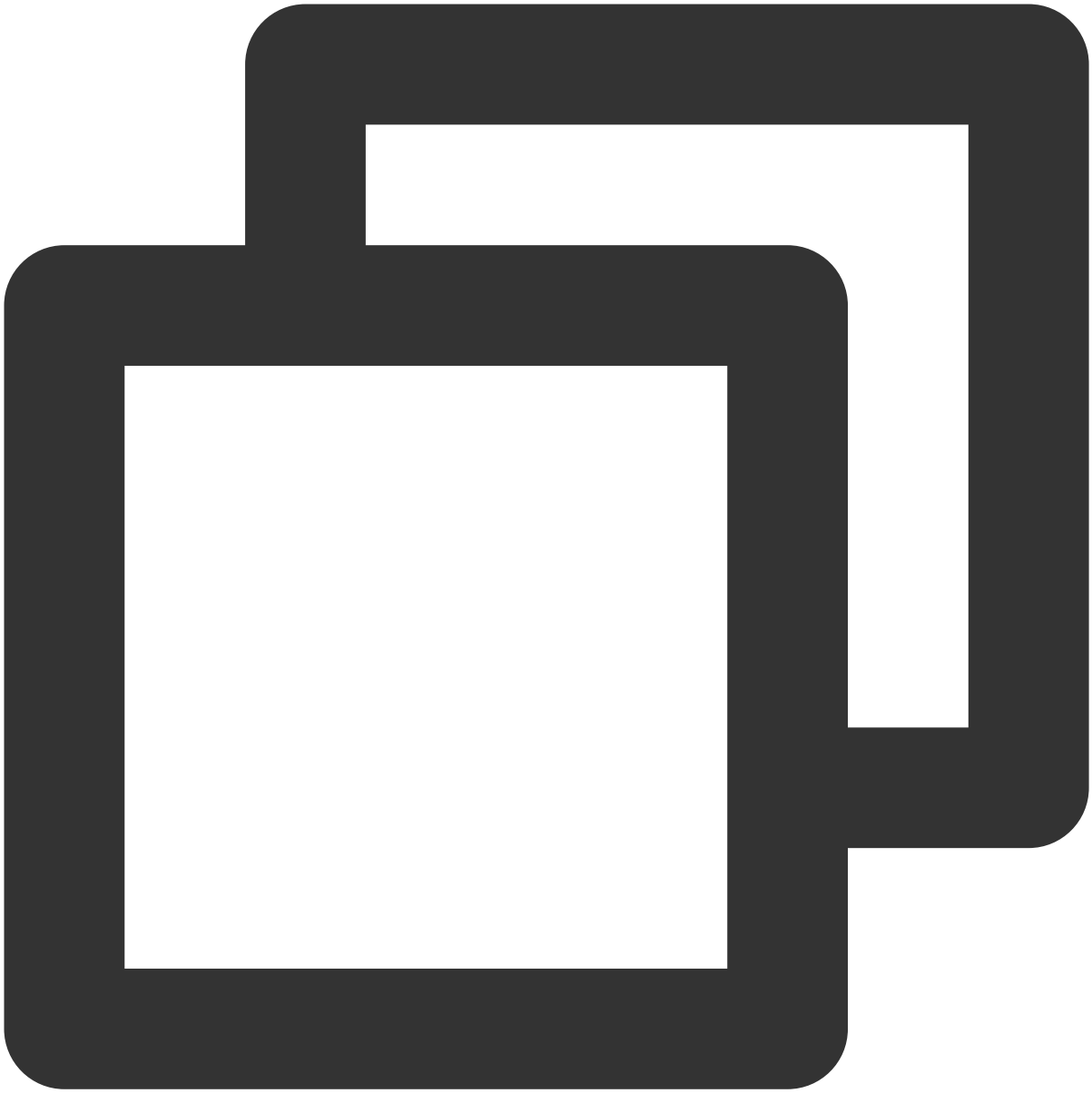


```
mVodPlayer.setBitrateIndex(-1); //index 参数传入-1
```

在播放过程中，可以随时通过 `mVodPlayer.setBitrateIndex(int)` 切换其它码率，切换后码流自适应也随之关闭。

12、开启平滑切换码率

在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。

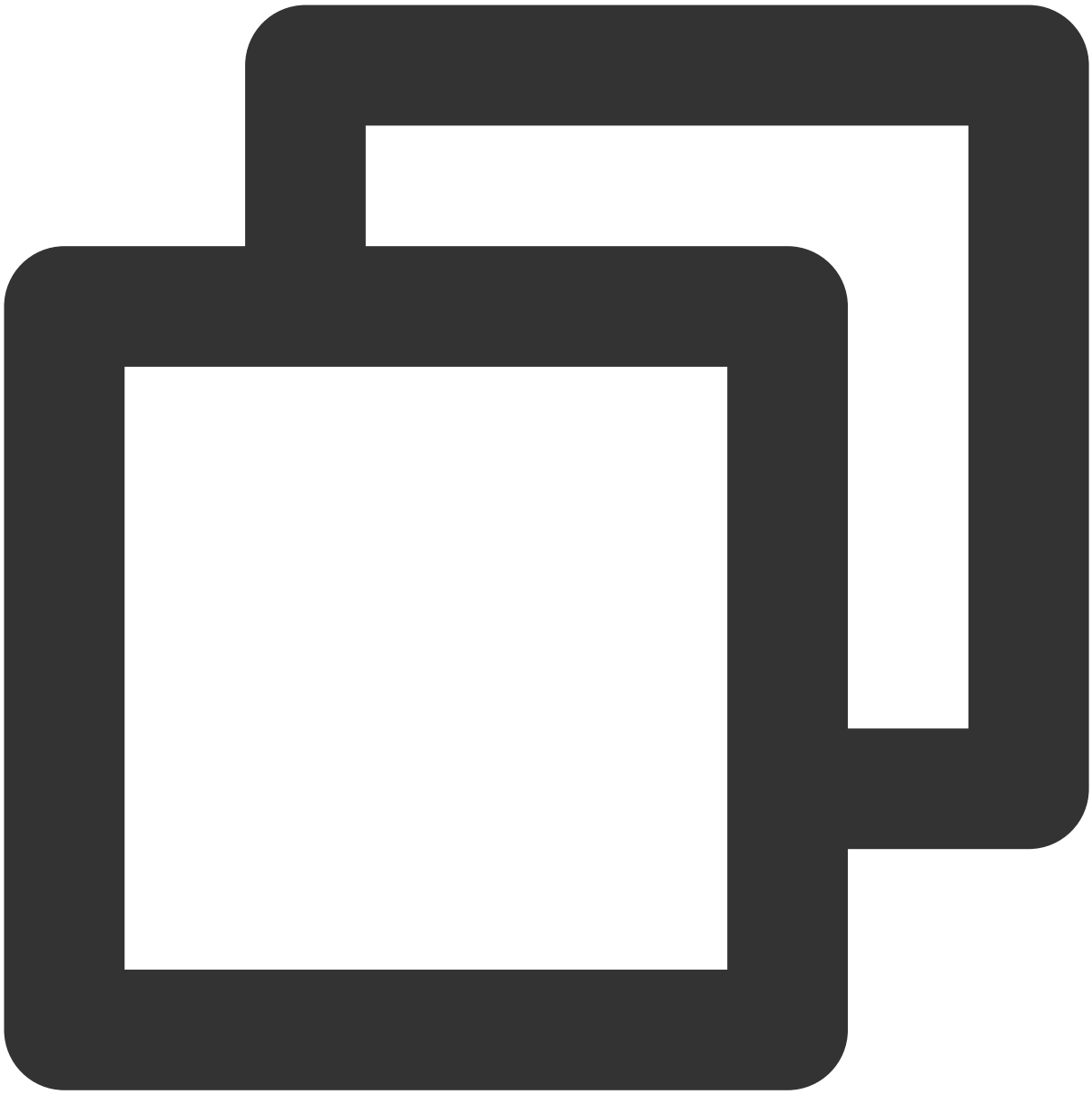


```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
// 设为true，在IDR对齐时可平滑切换码率，设为false时，可提高多码率地址打开速度
mPlayConfig.setSmoothSwitchBitrate(true);
mVodPlayer.setConfig(mPlayConfig);
```

13、播放进度监听

点播播放中的进度信息分为2种：**加载进度**和**播放进度**，SDK 目前是以事件通知的方式将这两个进度实时通知出来的。更多事件通知内容参见事件监听。

您可以为 TXVodPlayer 对象绑定一个 **TXVodPlayerListener** 监听器，进度通知会通过 **PLAY_EVT_PLAY_PROGRESS** 事件回调到您的应用程序，该事件的附加信息中即包含上述两个进度指标。



```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_PLAY_PROGRESS) {
            // 加载进度，单位是毫秒
            int playable_duration_ms = param.getInt(TXLiveConstants.EVT_PLAYABLE_DU
            mLoadBar.setProgress(playable_duration_ms); // 设置loading 进度条
```

```
            // 播放进度，单位是毫秒
            int progress_ms = param.getInt(TXLiveConstants.EVT_PLAY_PROGRESS_MS);
            mSeekBar.setProgress(progress_ms);  // 设置播放进度条

            // 视频总长，单位是毫秒
            int duration_ms = param.getInt(TXLiveConstants.EVT_PLAY_DURATION_MS);
            // 可以用于设置时长显示等等

            // 获取 PDT 时间，播放器高级版 11.6 版本开始支持
            long pdt_time_ms = param.getLong(TXVodConstants.EVT_PLAY_PDT_TIME_MS);
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
    }
});
```

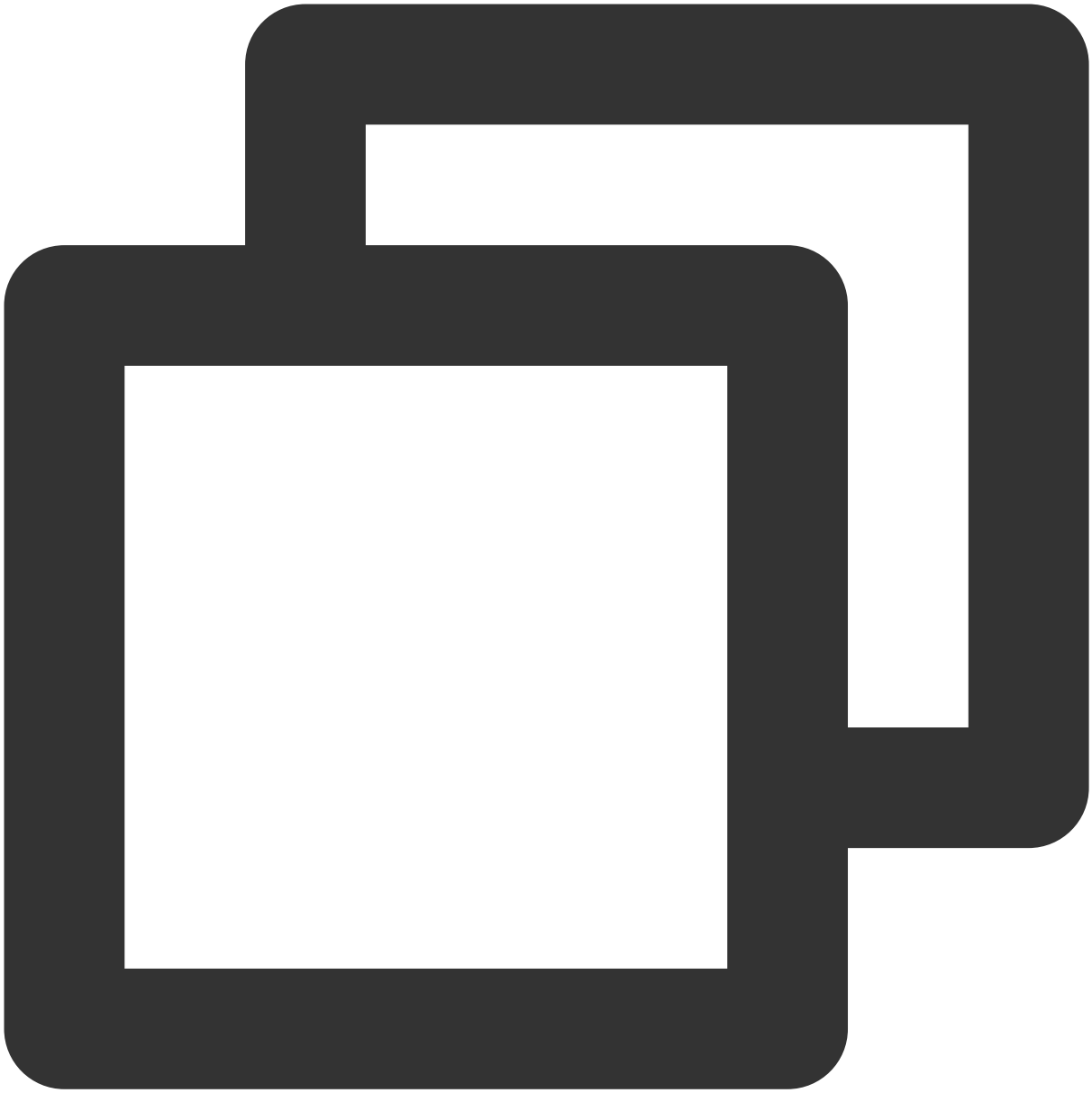
14、播放网速监听

通过 事件监听 方式，可以在视频播放卡顿时在显示当前网速。

通过 `onNetStatus` 的 `NET_STATUS_NET_SPEED` 获取当前网速。具体使用方法见 状态反馈（onNetStatus）。

监听到 `PLAY_EVT_PLAY_LOADING` 事件后，显示当前网速。

收到 `PLAY_EVT_VOD_LOADING_END` 事件后，对显示当前网速的 view 进行隐藏。



```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_PLAY_LOADING) {
            // 显示当前网速

        } else if (event == TXLiveConstants.PLAY_EVT_VOD_LOADING_END) {
            // 对显示当前网速的 view 进行隐藏
        }
    }
```

```
    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        // 获取实时速率，单位：kbps
        int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
    }
});
```

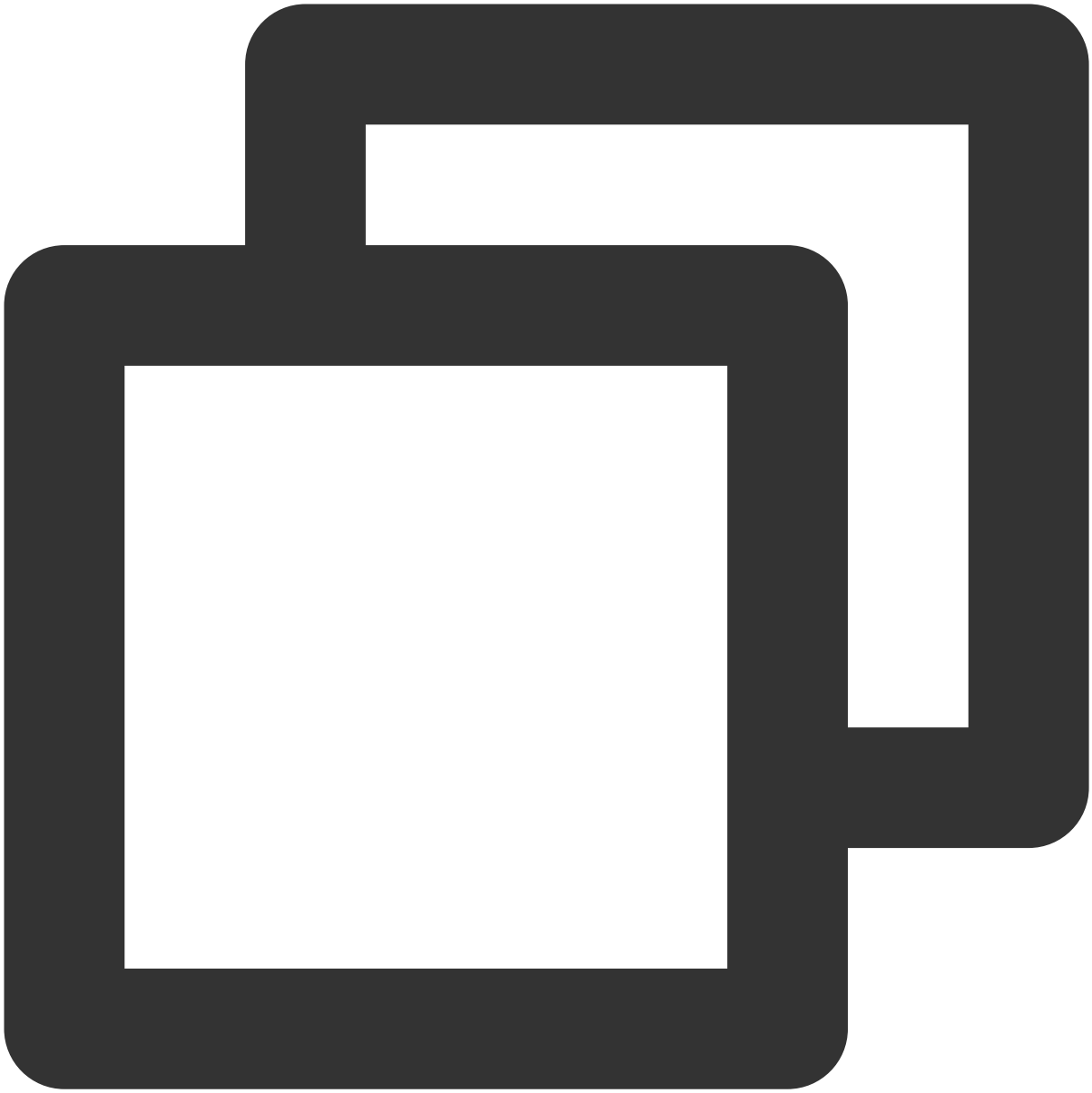
15、获取视频分辨率

播放器 SDK 通过 URL 字符串播放视频，URL 中本身不包含视频信息。为获取相关信息，需要通过访问云端服务器加载到相关视频信息，因此 SDK 只能以事件通知的方式将视频信息发送到您的应用程序中，更多内容参见 事件监听。

可以通过下面两种方法获取分辨率信息

方法1：通过 `onNetStatus` 的 `NET_STATUS_VIDEO_WIDTH` 和 `NET_STATUS_VIDEO_HEIGHT` 获取视频的宽和高。具体使用方法见 状态反馈（onNetStatus）。

方法2：在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后，直接调用 `TXVodPlayer.getWidth()` 和 `TXVodPlayer.getHeight()` 获取当前宽高。

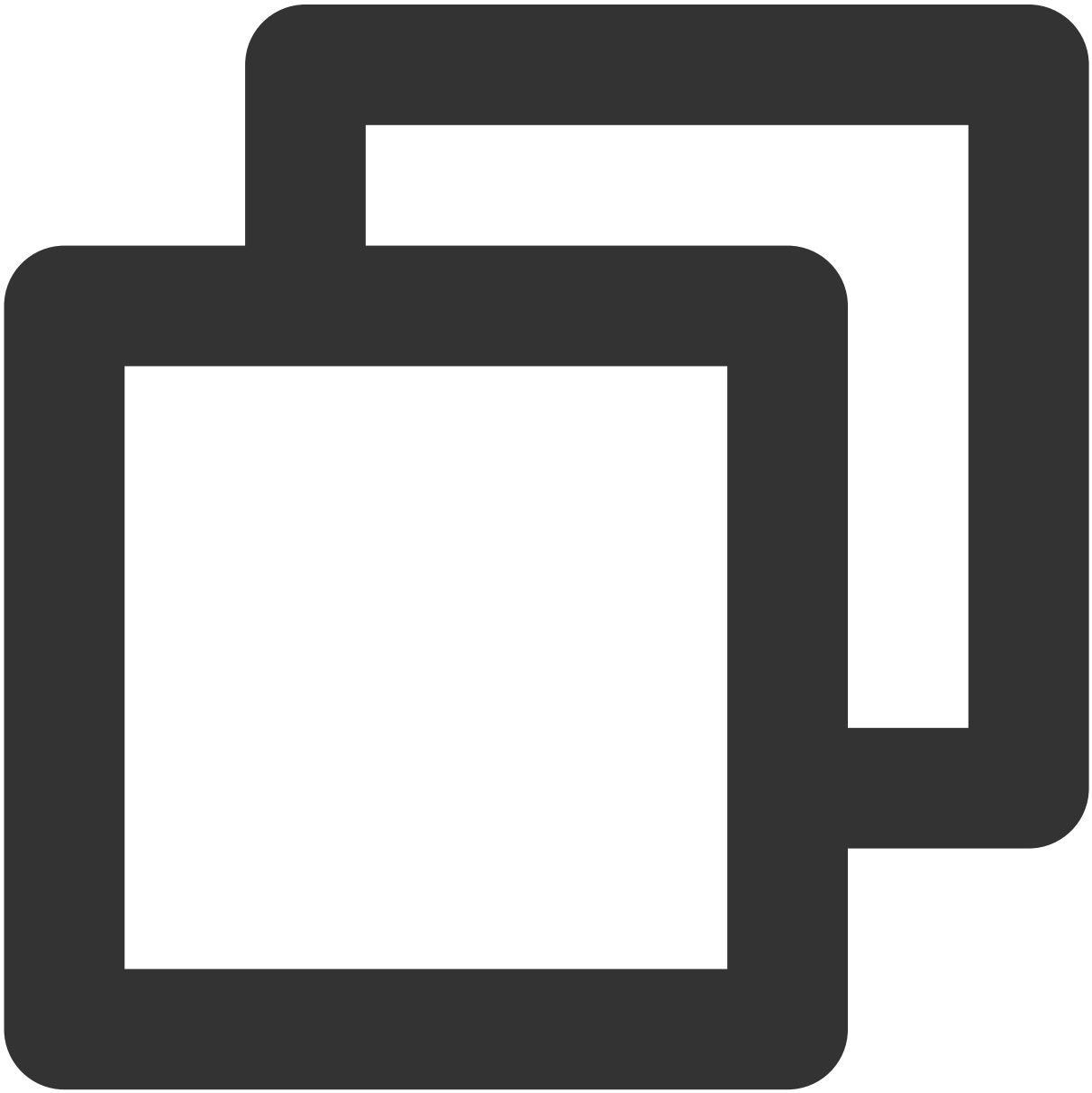


```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        //获取视频宽度
        int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
        //获取视频高度
        int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
```

```
    }
});

// 获取视频宽高，需要在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后才返回值
mVodPlayer.getWidth();
mVodPlayer.getHeight();
```

16、播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10);  // 播放时最大缓冲大小。单位：MB
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

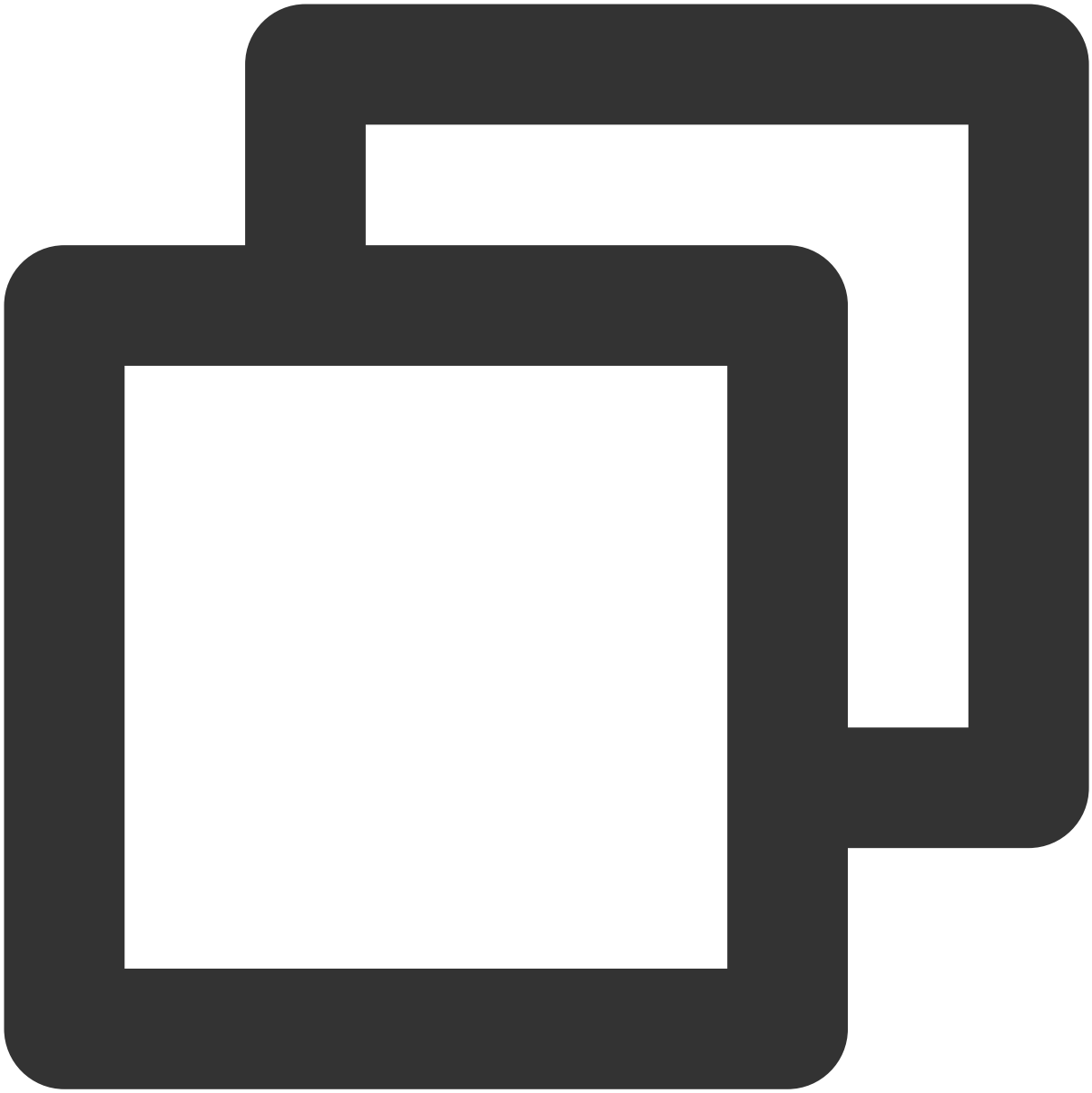
17、视频本地缓存

在短视频播放场景中，视频文件的本地缓存是很刚需的一个特性，对于普通用户而言，一个已经看过的视频再次观看时，不应该再消耗一次流量。

格式支持：SDK 支持 HLS(m3u8) 和 MP4 两种常见点播格式的缓存功能。

开启时机：SDK 并不默认开启缓存功能，对于用户回看率不高的场景，也并不推荐您开启此功能。

开启方式：全局生效，在使用播放器开启。开启此功能需要配置两个参数：本地缓存目录及缓存大小。



```
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
if (sdcardDir != null) {
    //设置播放引擎的全局缓存目录
   TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
    //设置播放引擎的全局缓存目录和缓存大小，//单位MB
    TXPlayerGlobalSetting.setMaxCacheSize(200);
}

//使用播放器
```

说明：

旧版本通过 TXVodPlayConfig#setMaxCacheItems 接口配置已经废弃，不推荐使用。

18、DRM 加密视频播放

注意：

此功能需要播放器高级版本才支持。

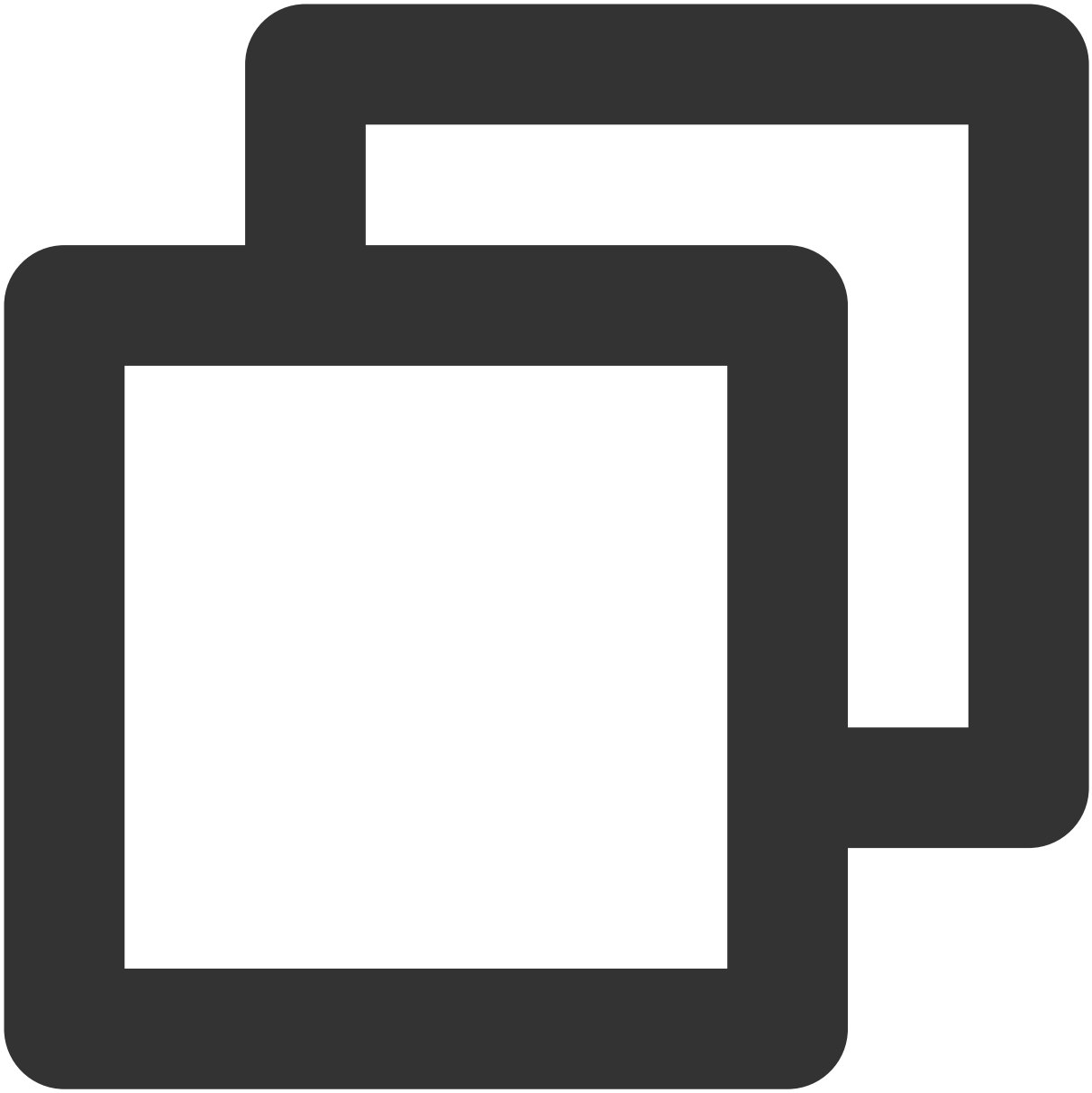
播放器高级版 SDK 支持播放商业级 DRM 加密视频，目前支持 WideVine 和 Fairplay 2种 DRM 方案。更多的商业级 DRM 信息，请参考产品介绍。

注意：DRM 播放请通过 `TXVodPlayer#setSurface` 配置后播放，否则会出现播放异常。

可通过下面 2 种播放 DRM 加密视频：

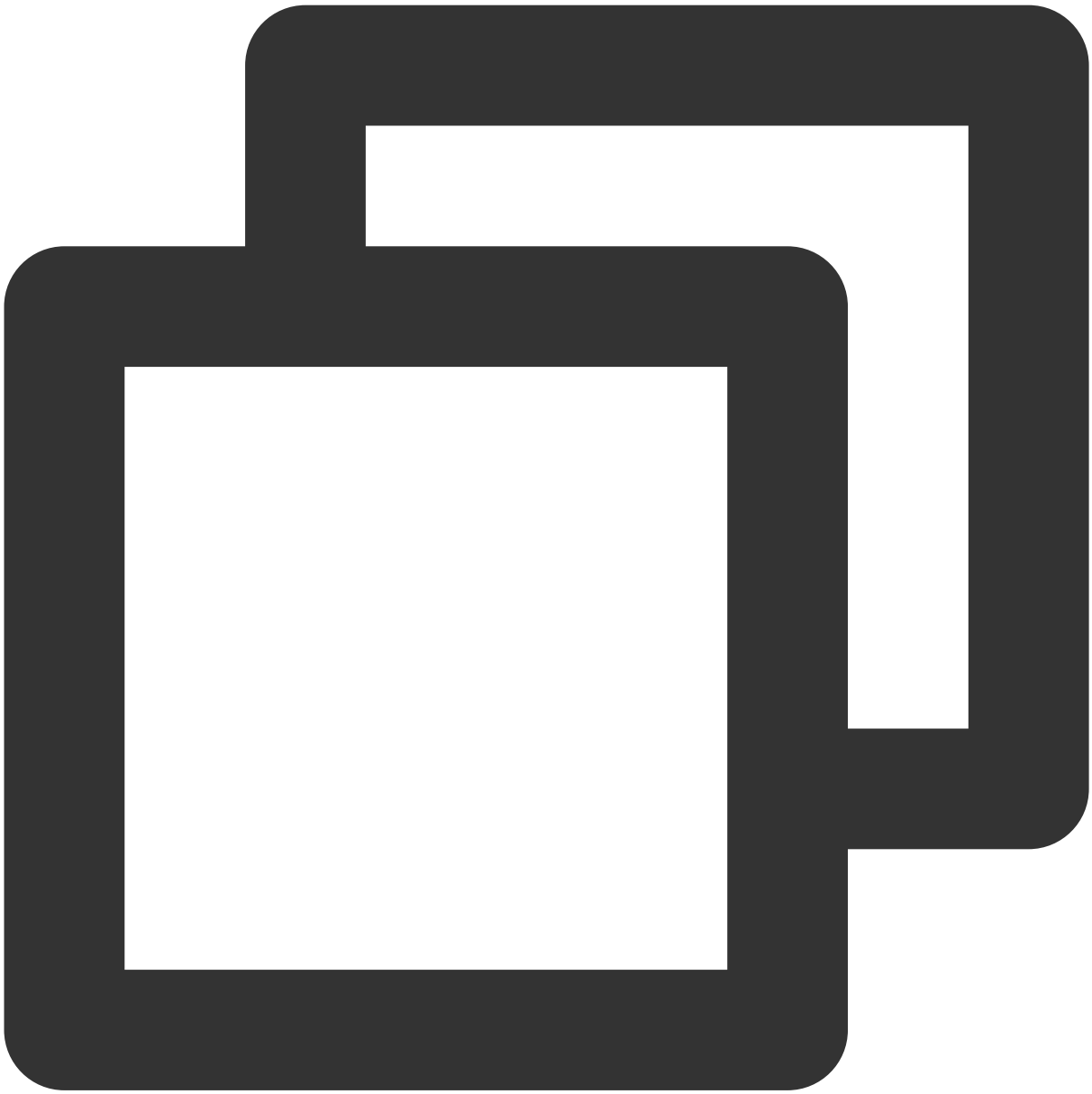
通过 FileId 播放

自定义配置播放



```
// DRM 播放请通过 TXVodPlayer#setSurface配置后播放，否则会出现播放异常
mVodPlayer.setSurface(mPlayerView.getHolder().getSurface());

// psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/prod
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(${appId}, // 腾讯云账户的 appId
    ${fieId}, // DRM 加密视频的 fileId
    ${psgin}); // 加密视频的播放器签名
mVodPlayer.startVodPlay(playInfoParam);
```

通过 FileId 播放适用于接入云点播后台。这种方式和播放普通 FileId 文件没有差别，需要在云点播先配置资源为 DRM 类型，SDK 会在内部识别并处理。



```
// DRM 播放请通过 TXVodPlayer#setSurface配置后播放，否则会出现播放异常
mVodPlayer.setSurface(mPlayerView.getHolder().getSurface());

// 步骤一： 设置 Drm 证书提供商环境，此步骤默认情况下不需要配置
// Google Drm 证书提供商环境默认使用 googleapis.com 域名，可通过下面接口设置为 googleapis.c
TXPlayerGlobalSetting.setDrmProvisionEnv(TXPlayerGlobalSetting.DrmProvisionEnv.DRM_

// 步骤二：通过 TXVodPlayer#startPlayDrm 接口播放
```

```
TXPlayerDrmBuilder builder = new TXPlayerDrmBuilder();
builder.setPlayUrl(${url});  // 设置播放视频的 url
builder.setKeyLicenseUrl(${keyLicneseUrl});    /// 设置解密 key url
mVodPlayer.startPlayDrm(builder);
```

19、外挂字幕

注意：

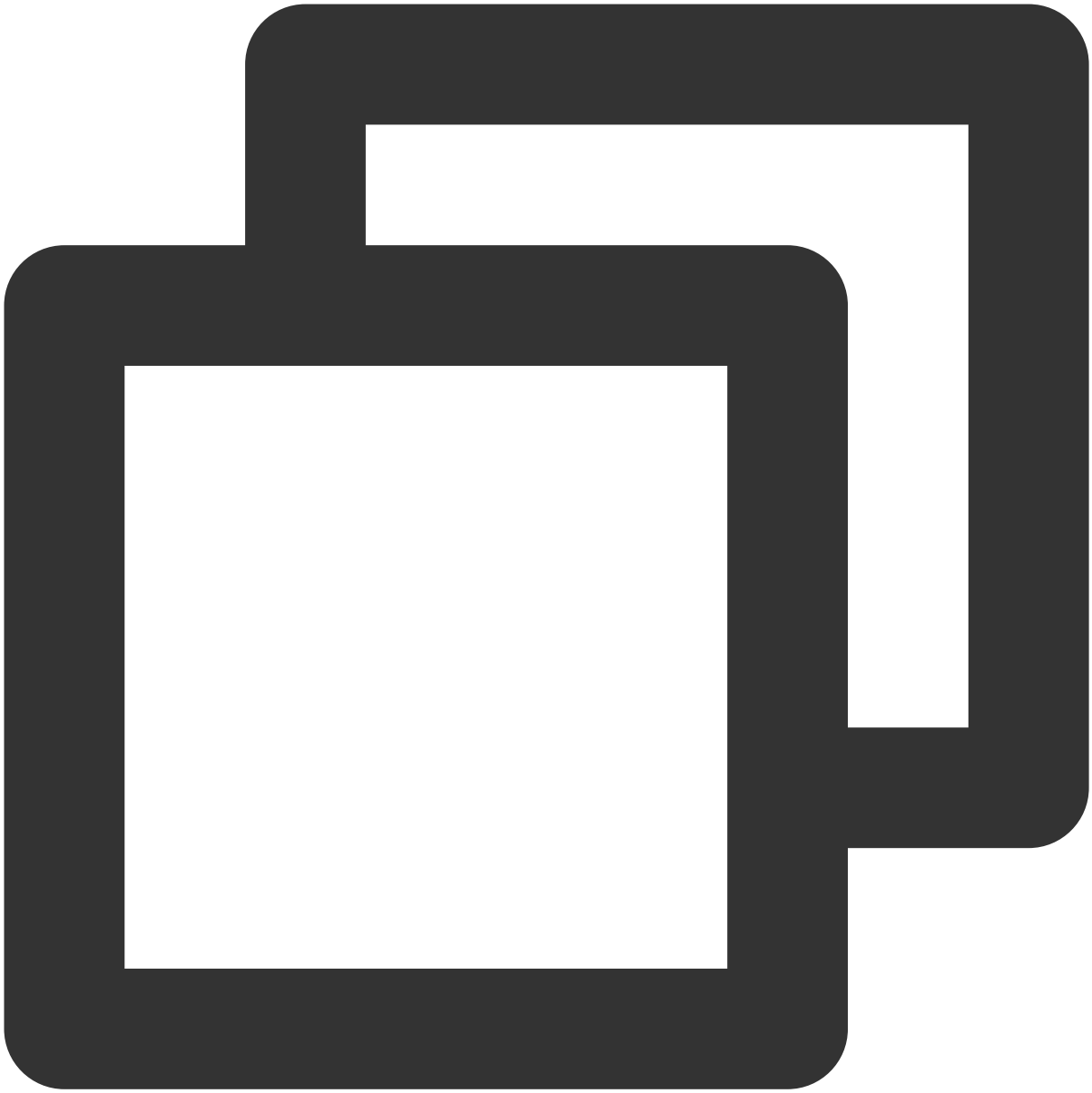
此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这 2 种格式的字幕。

最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `VOD_PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。 添加字幕后，并不会主动加载字幕， 调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。

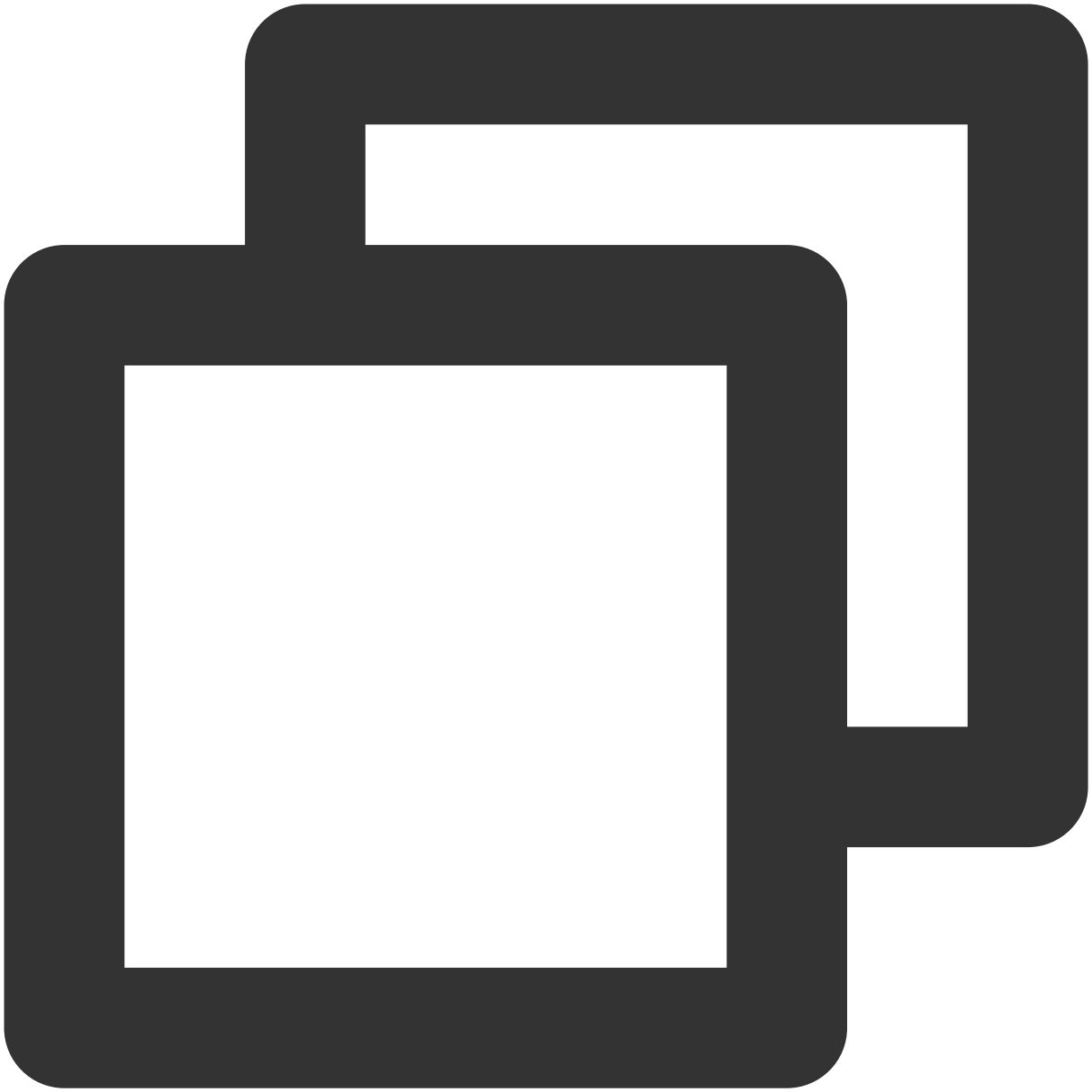
用法如下：

步骤1：设置字幕渲染目标对象 View。



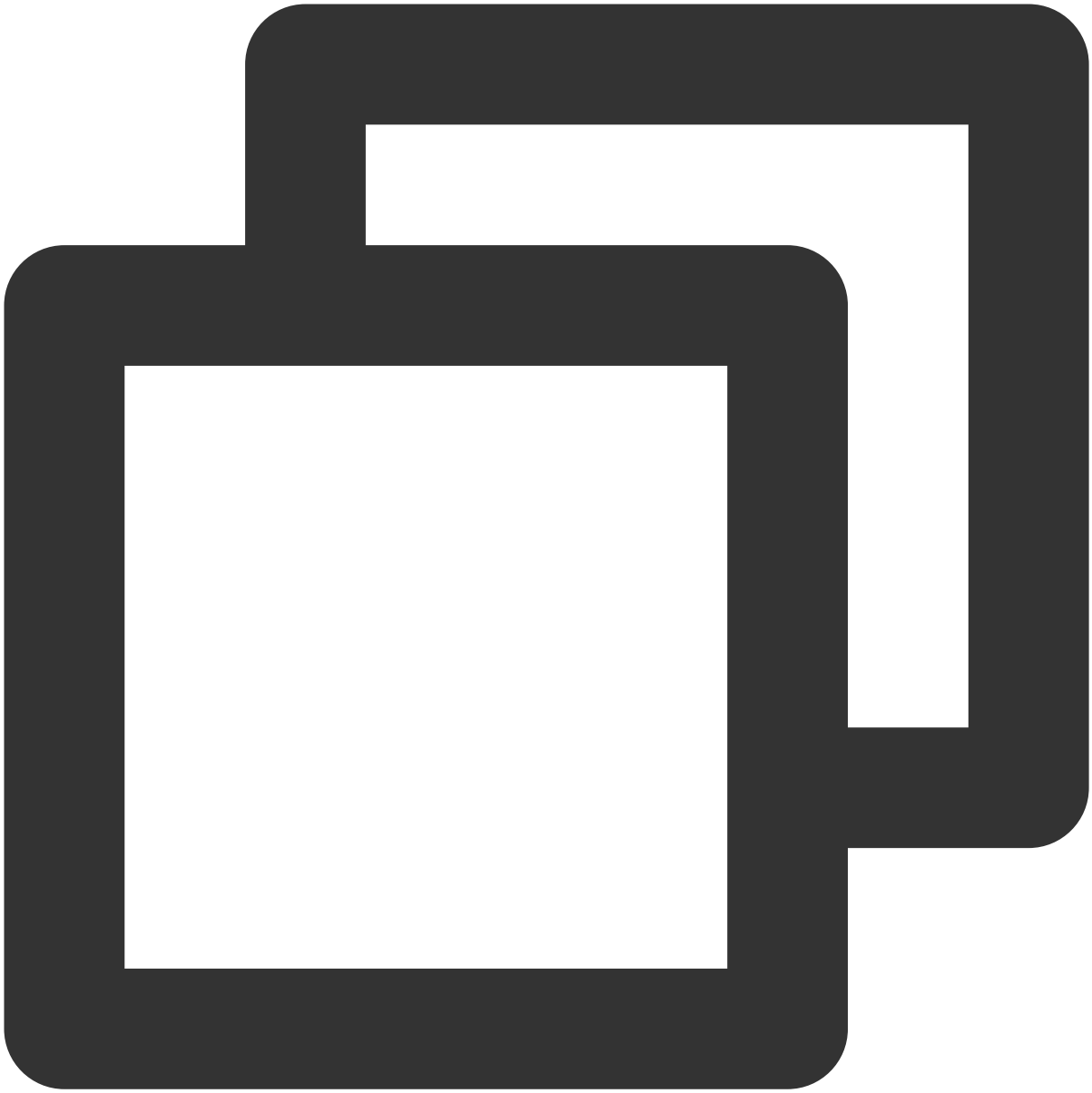
```
// 把TXSubtitleView添加到播放器所在布局xml
<com.tencent.rtmp.ui.TXSubtitleView
            android:id="@+id/subtitle_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

mSubtitleView = findViewById(R.id.subtitle_view);
// 设置字幕渲染目标对象
mVodPlayer.setSubtitleView(mSubtitleView);
```

步骤2：添加外挂字幕。



```
// 传入 字幕url， 字幕名称， 字幕类型，建议在启动播放前添加
mVodPlayer.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-cloud.com/De
```

步骤3：播放后切换字幕。

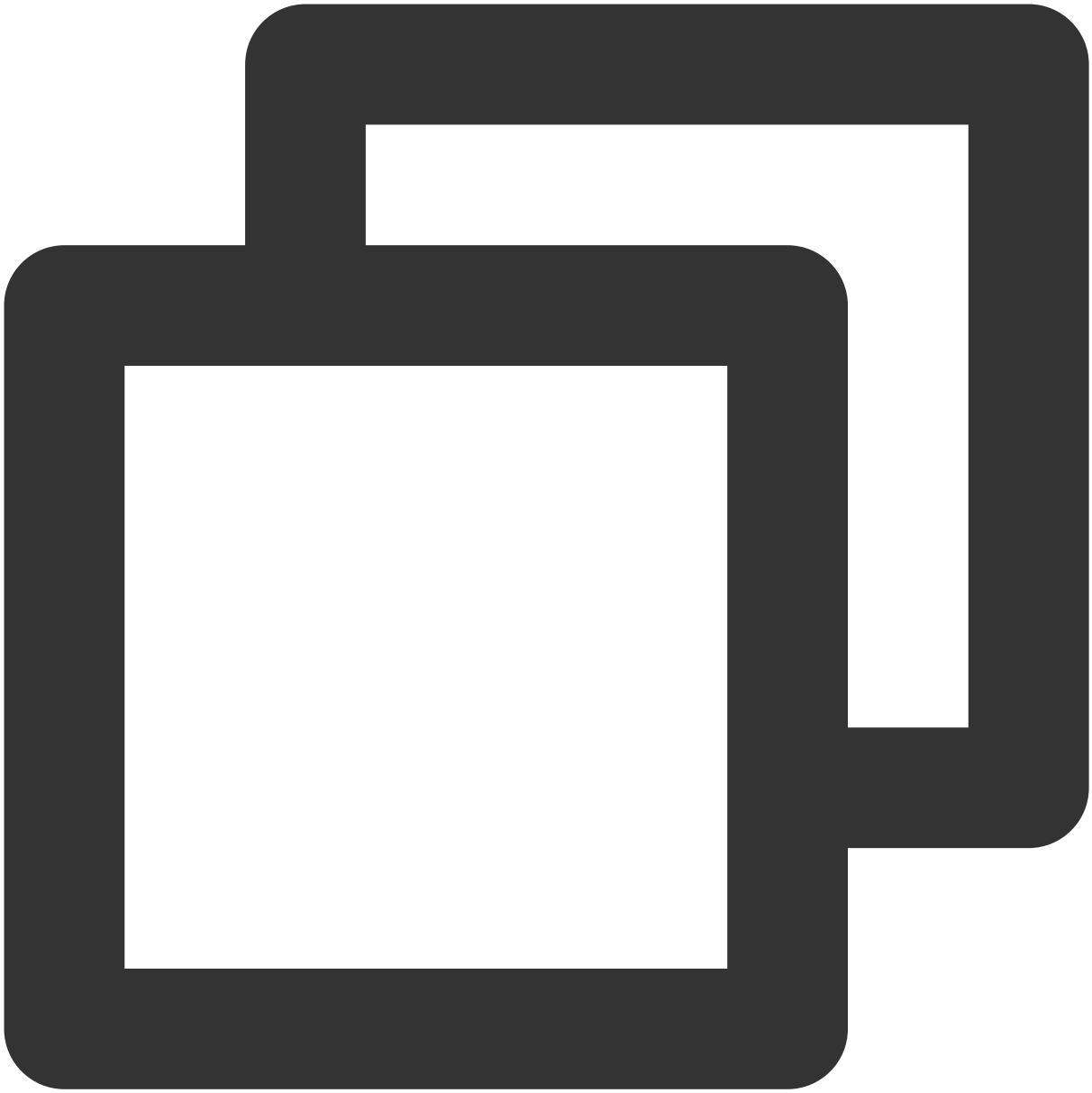


```
// 开始播放视频后，选中添加的外挂字幕，  在收到 VOD_PLAY_EVT_VOD_PLAY_PREPARED 事件后调用
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VOD_PLAY_PREPARED) {
        List<TXTrackInfo> subtitleTrackInfoList = mVodPlayer.getSubtitleTrackInfo()
        for (TXTrackInfo track : subtitleTrackInfoList) {
            Log.d(TAG, "TrackIndex= " + track.getTrackIndex() + " ,name= " + track.
            if (TextUtils.equals(track.getName(), "subtitleName")) {
                // 选中字幕
                mVodPlayer.selectTrack(track.trackIndex);
            } else {
```

```
                // 其它字幕不需要的话，  进行deselectTrack
                mVodPlayer.deselectTrack(track.trackIndex);
            }
        }
    }
}


// 如果需要，可以监听轨道切换消息
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXVodConstants.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
            int trackIndex = param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_INDEX
            int errorCode = param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_ERROR_
            Log.d(TAG, "receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=" +
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle status) {

    }
});
```

步骤4：配置字幕样式。

字幕样式支持在播放前或者播放过程中配置。



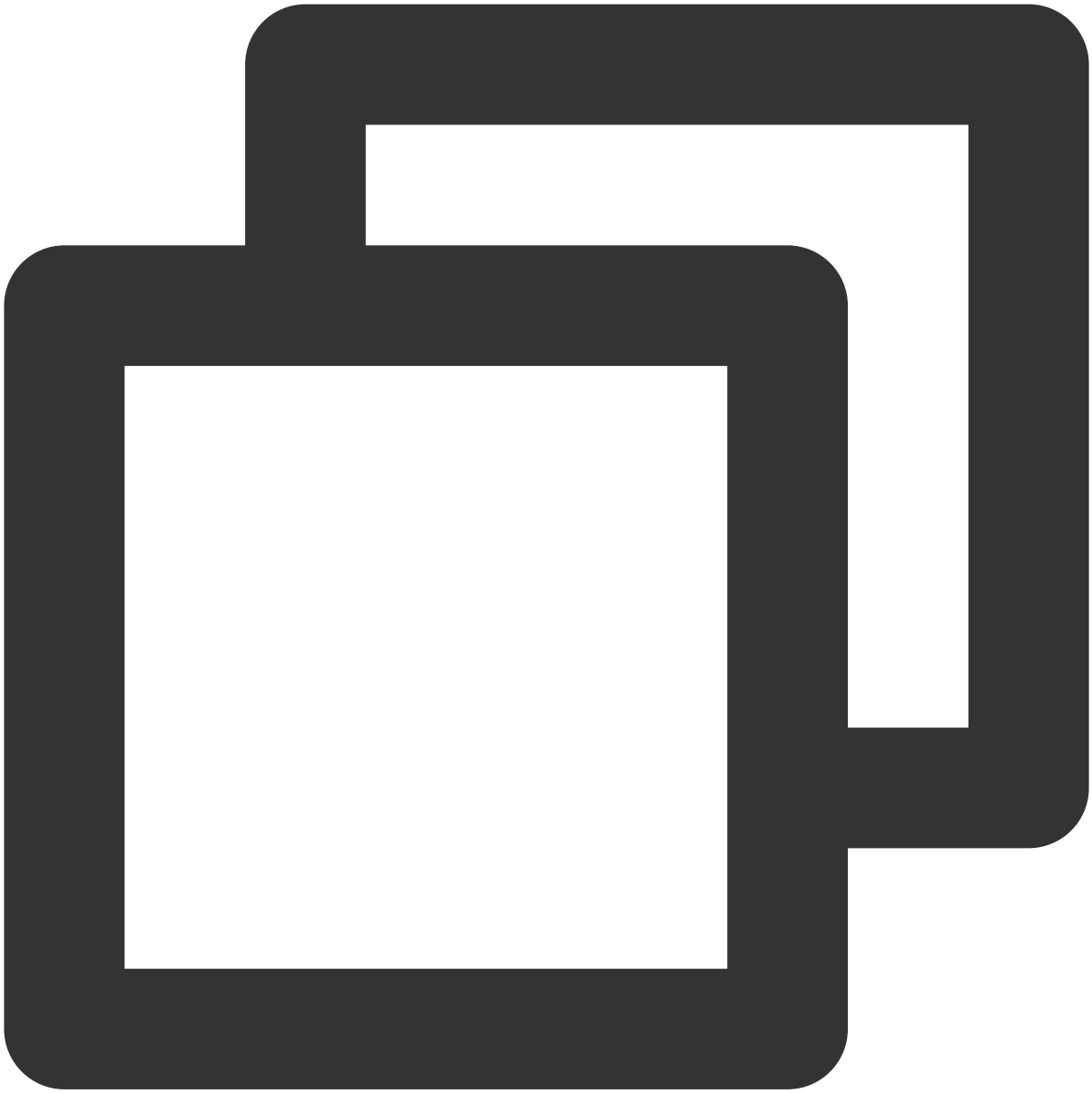
```
// 详细参数配置请参考 API 文档
TXSubtitleRenderModel model = new TXSubtitleRenderModel();
model.canvasWidth = 1920;  // 字幕渲染画布的宽
model.canvasHeight = 1080;  // 字幕渲染画布的高
model.fontColor = 0xFFFFFFFF; // 设置字幕字体颜色，默认白色不透明
model.isBondFontStyle = false;  // 设置字幕字体是否为粗体
mVodPlayer.setSubtitleStyle(model);
```

20、多音轨切换

注意：

此功能需要播放器高级版本才支持。

播放器高级版 SDK 支持切换视频内置的多音轨。用法如下：



```
// 返回音频轨道信息列表
List<TXTrackInfo> soundTrackInfoList = mVodPlayer.getAudioTrackInfo();
for (TXTrackInfo trackInfo : soundTrackInfoList) {
    if (trackInfo.trackIndex ==  0) {
        // 通过判断 trackIndex 或者 name 切换到需要的音轨
        mVodPlayer.selectTrack(trackInfo.trackIndex);
    } else {
```

```
            // 不需要的音轨进行 deselectTrack
            mVodPlayer.deselectTrack(trackInfo.trackIndex);
        }
    }
```

进阶功能使用

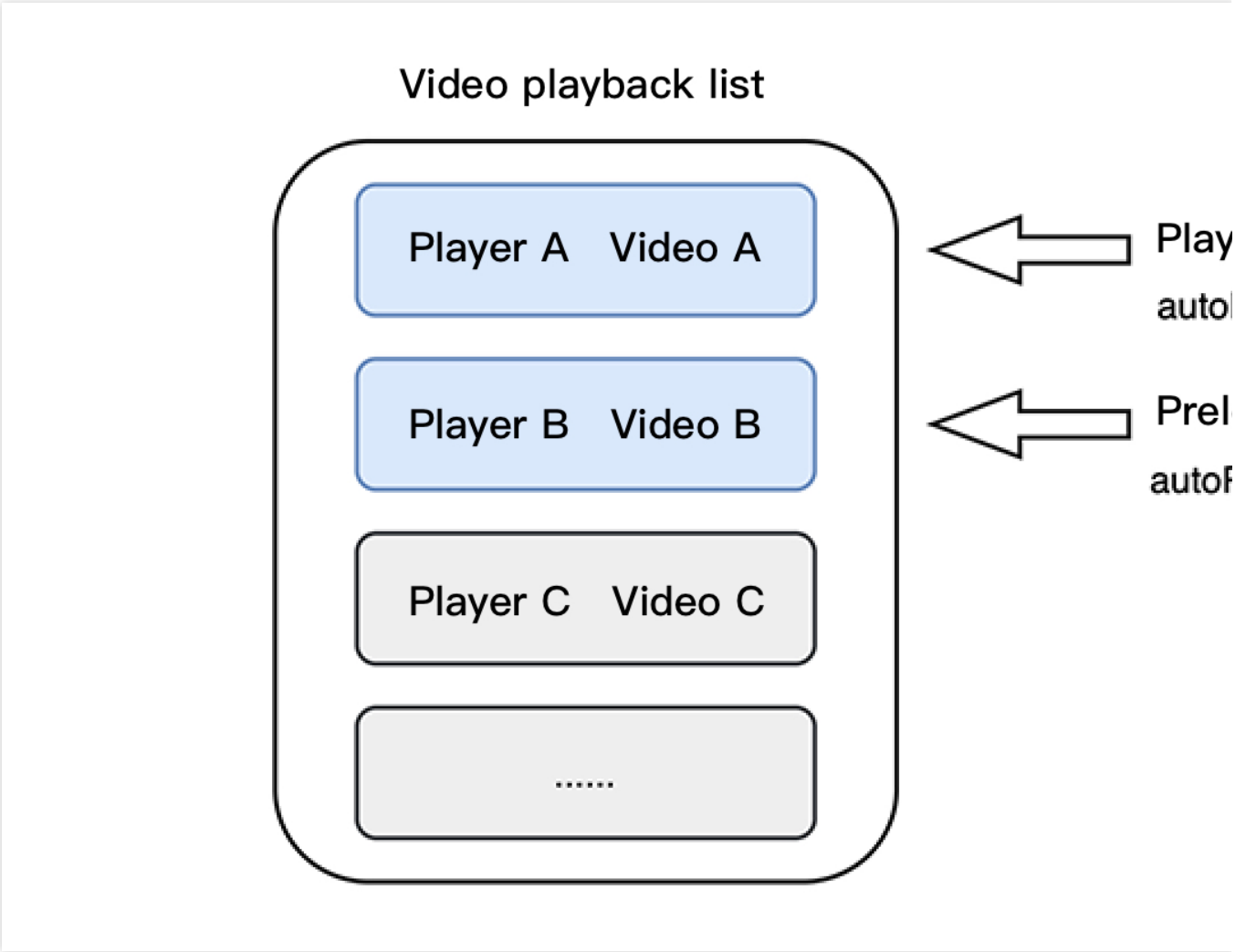
1、视频预播放

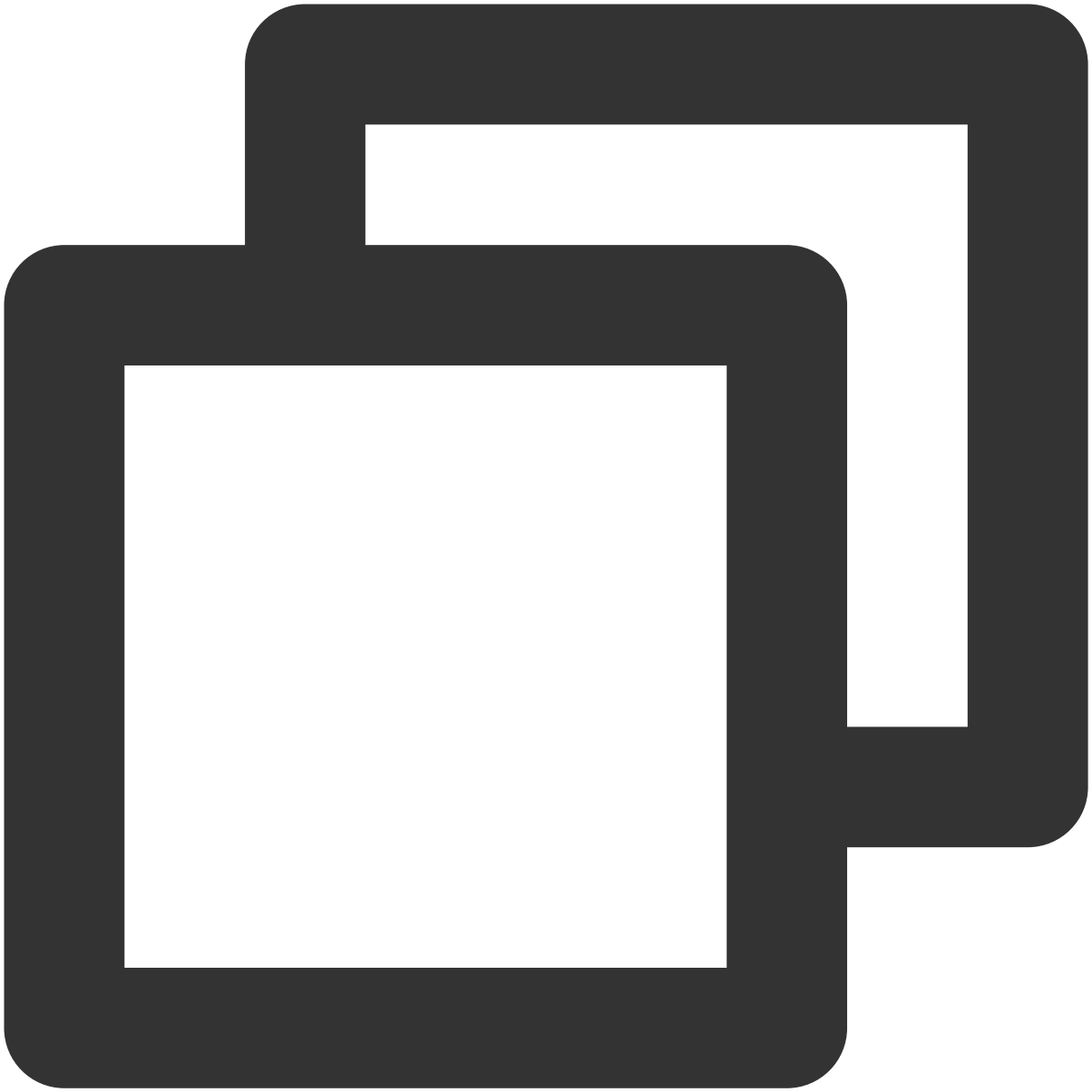
步骤1：视频预播放使用

在短视频播放场景中，视频预播放功能对于流畅的观看体验很有帮助：在观看当前视频的同时，在后台加载即将要播放的下一个视频，这样一来，当用户真正切换到下一个视频时，已经不需要从头开始加载了，而是可以做到立刻播放。

预播放视频会有很好的秒开效果，但有一定的性能开销，如果业务同时有较多的视频预加载需求，建议结合 视频预下载 一起使用。

这就是视频播放中无缝切换的背后技术支撑，您可以使用 TXVodPlayer 中的 setAutoPlay 开关来实现这个功能，具体做法如下：


Video playback list
Player A Video A
Player B Video B
Player C Video C
.....
Play
auto
Prel
autoF

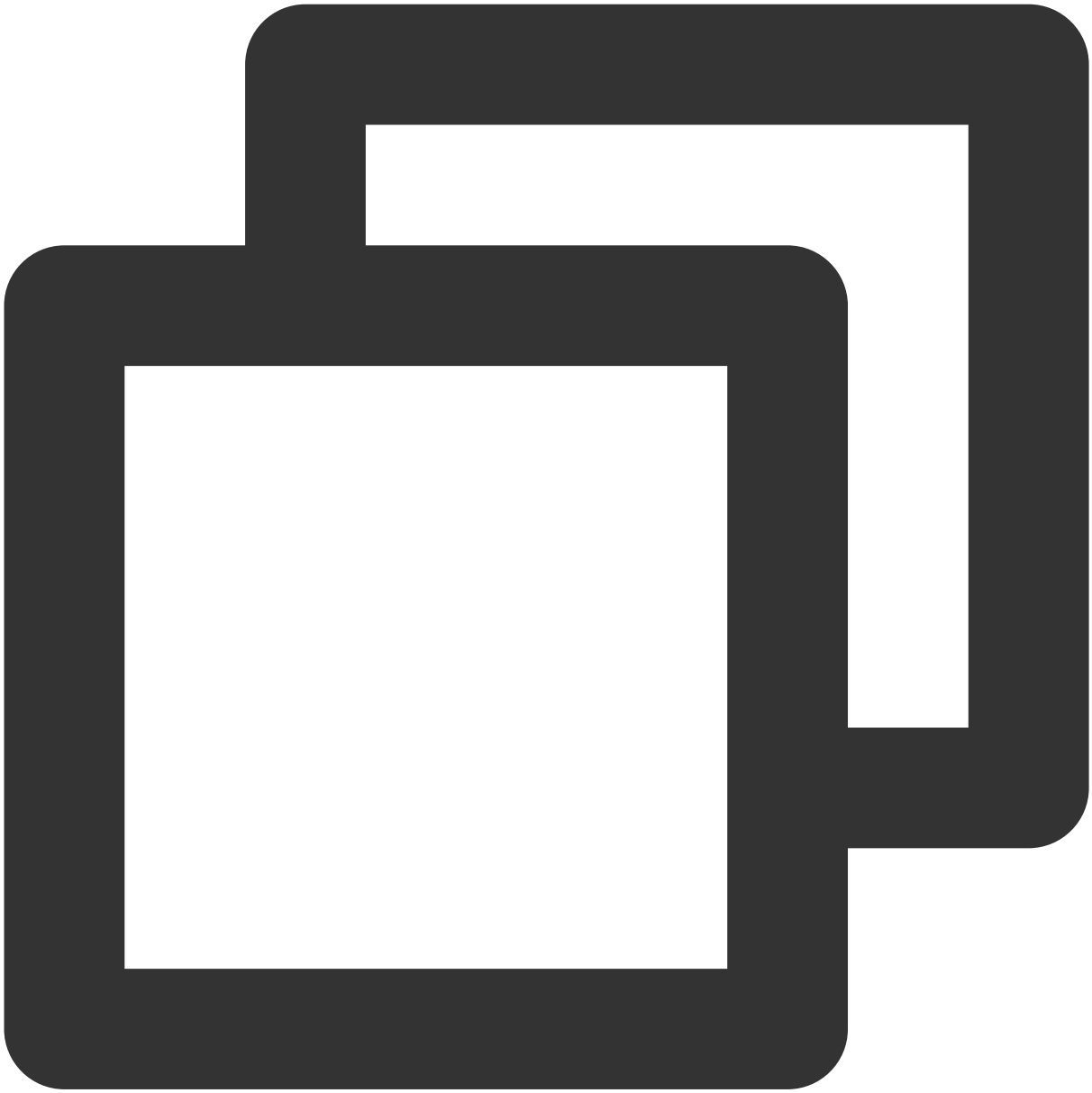


```
// 播放视频 A：如果将 autoPlay 设置为 true， 那么 startVodPlay 调用会立刻开始视频的加载和播放
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
playerA.setAutoPlay(true);
playerA.startVodPlay(urlA);

// 在播放视频 A 的同时，预加载视频 B，做法是将 setAutoPlay 设置为 false
String urlB = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
playerB.setAutoPlay(false);
playerB.startVodPlay(urlB); // 不会立刻开始播放，而只会开始加载视频
```

等到视频 A 播放结束，自动（或者用户手动切换到）视频 B 时，调用 resume 函数即可实现立刻播放。

注意：

设置了 autoPlay 为 false 之后，调用 resume 之前需要保证视频 B 已准备完成，即需要在监听到视频 B 的 PLAY_EVT_VOD_PLAY_PREPARED （2013，播放器已准备完成，可以播放）事件后调用。



```
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    // 在视频 A 播放结束的时候，直接启动视频 B 的播放，可以做到无缝切换
    if (event == PLAY_EVT_PLAY_END) {
            playerA.stop();
            playerB.setPlayerView(mPlayerView);
            playerB.resume();
        }
```

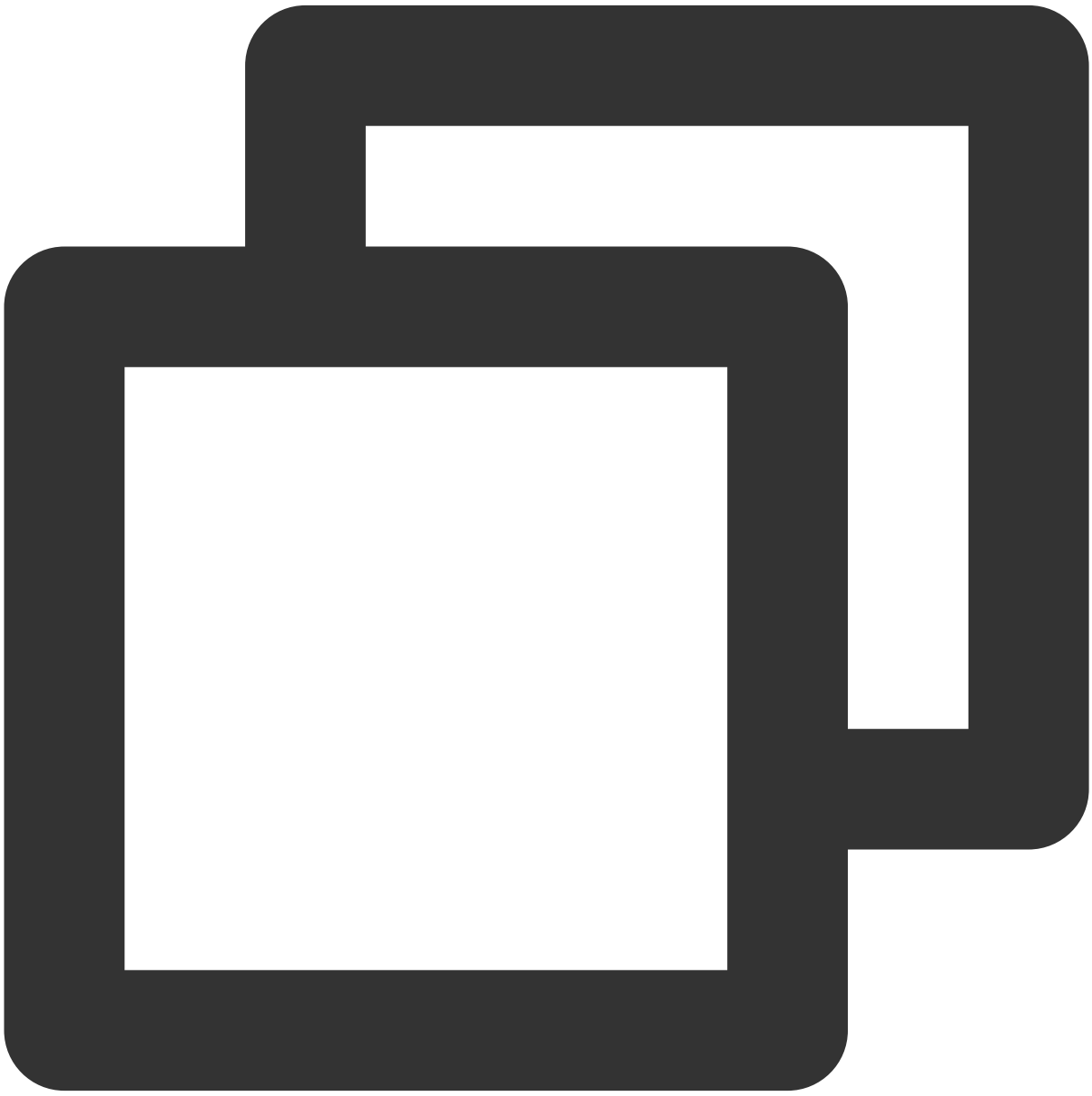
```
}
```

步骤2：视频预播放缓冲配置

设置较大的缓冲可以更好的应对网络的波动，达到流畅播放的目的。
设置较小的缓冲可以帮助节省流量消耗。

预播放缓冲大小

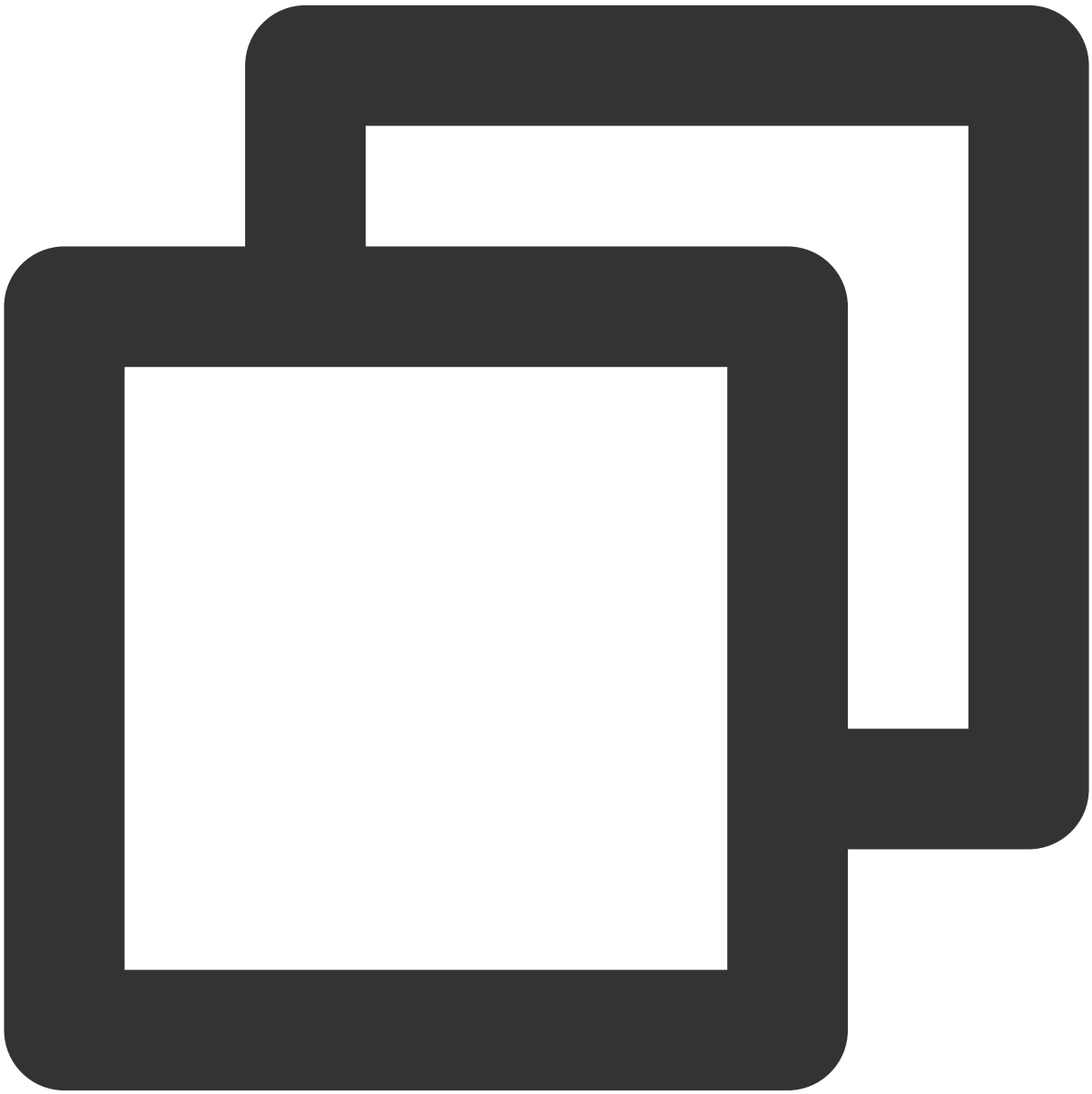
此接口针对预加载场景（即在视频启播前，且设置 player 的 AutoPlay 为 false），用于控制启播前阶段的最大缓冲大小。



```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxPreloadSize(2);   // 预播放最大缓冲大小。单位：MB，根据业务情况设置去节省流量
mVodPlayer.setConfig(config);   // 把config 传给 mVodPlayer
```

播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10);   // 播放时最大缓冲大小。单位：MB
```

```
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

2、视频预下载

不需要创建播放器实例，预先下载视频部分内容，使用播放器时，可以加快视频启播速度，提供更好的播放体验。
在使用播放服务前，请确保先设置好 视频缓存。

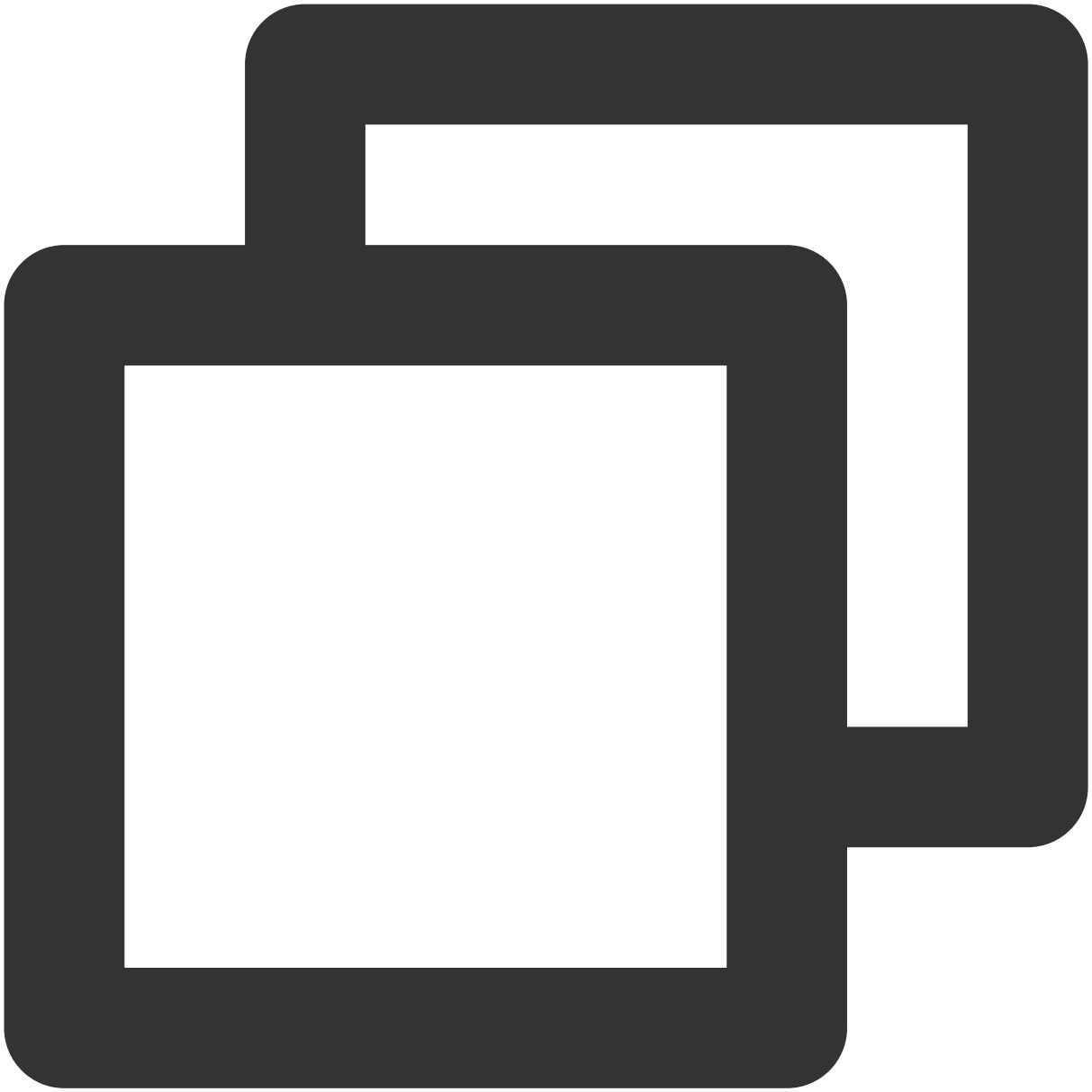
说明：

1. TXPlayerGlobalSetting 是全局缓存设置接口，原有 TXVodConfig 的缓存配置接口废弃。
2. 全局缓存目录和大小设置的优先级高于播放器 TXVodConfig 配置的缓存设置。

通过媒资 URL 预下载

通过媒资 FileId 预下载

通过媒资 URL 预下载视频代码示例如下：



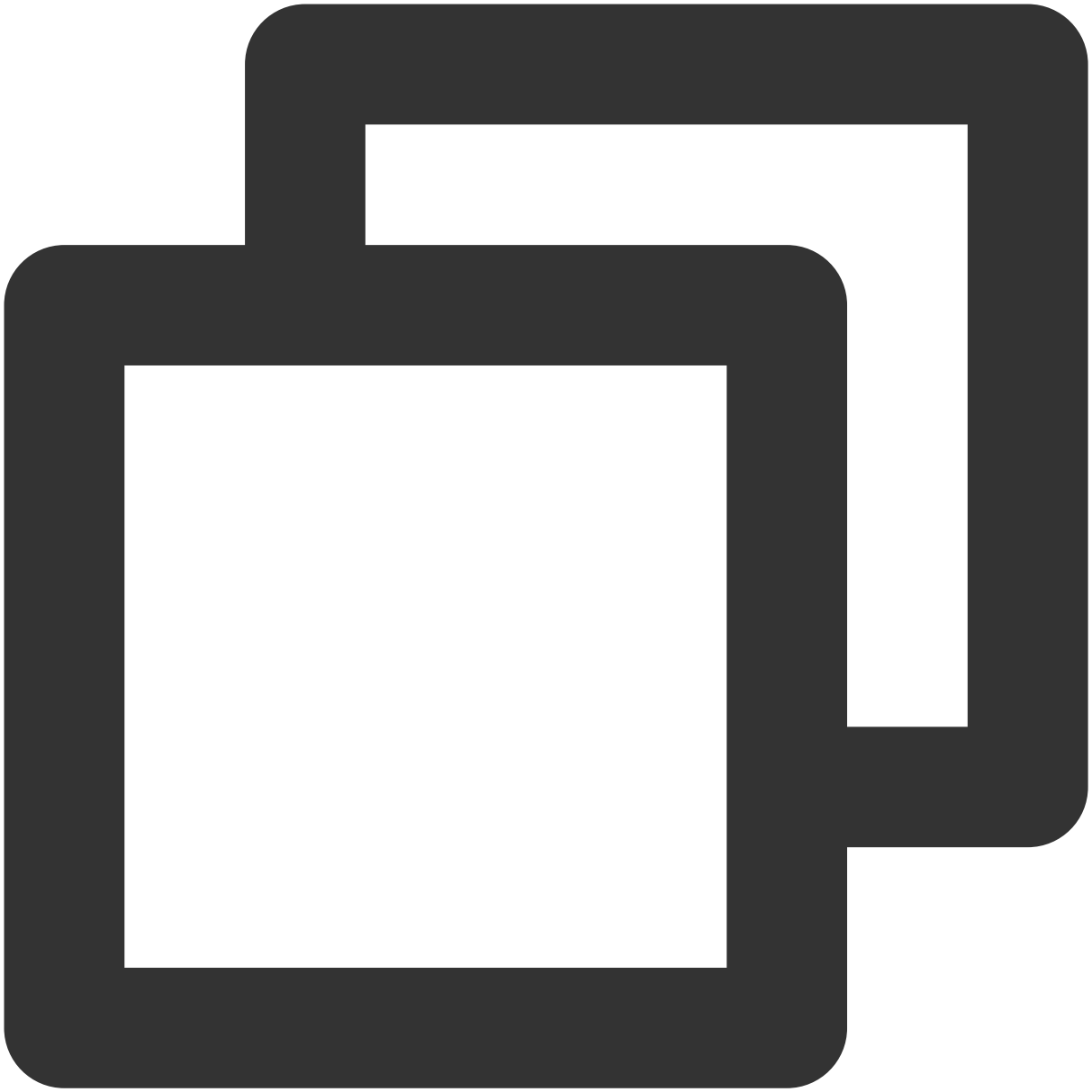
```
//先设置播放引擎的全局缓存目录和缓存大小
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
//设置播放引擎的全局缓存目录和缓存大小
if (sdcardDir != null) {
    TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
    TXPlayerGlobalSetting.setMaxCacheSize(200); //单位MB
}

String palyrl = "http://****";
//启动预下载
final TXVodPreloadManager downloadManager = TXVodPreloadManager.getInstance(getAppl
```

```
final int taskID = downloadManager.startPreload(playUrl, 3, 1920*1080, new ITXVodPr
    @Override
    public void onComplete(int taskID, String url) {
        Log.d(TAG, "preload: onComplete: url: " + url);
    }

    @Override
    public void onError(int taskID, String url, int code, String msg) {
        Log.d(TAG, "preload: onError: url: " + url + ", code: " + code + ", msg: "
    }

});

//取消预下载
downloadManager.stopPreload(taskID);
```

注意：

通过 fileId 预下载从 11.3 版本开始支持。

通过 fileId 预下载是耗时操作， 请不要在主线程调用，否则会抛出非法调用异常。startPreload 时传入的 preferredResolution 要和启播时设置的优先启播分辨率保持一致，否则将达不到预期的效果。 使用示例如下：



```
//先设置播放引擎的全局缓存目录和缓存大小
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
//设置播放引擎的全局缓存目录和缓存大小
if (sdcardDir != null) {
    TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
    TXPlayerGlobalSetting.setMaxCacheSize(200); //单位MB
}

// 启动预下载
Runnable task = new Runnable() {
    @Override
```

```
    public void run() {
        TXPlayInfoParams playInfoParams = new TXPlayInfoParams(${appId}, "${fileId}
                "${psign}");
        // 注意： 耗时操作，请不要在主线程调用！在主线程调用将会抛出非法调用异常。
        mPreLoadManager.startPreload(playInfoParams, 3, 1920 * 1080, new ITXVodFile

            @Override
            public void onStart(int taskID, String fileId, String url, Bundle bundl
                // 通过 fileId 换链成功后会回调 onStart
                Log.d(TAG, "preload: onStart: taskID: " + taskID + ", fileId: " + f
            }

            @Override
            public void onComplete(int taskID, String url) {
                Log.d(TAG, "preload: onComplete: url: " + url);
            }

            @Override
            public void onError(int taskID, String url, int code, String msg) {
                Log.d(TAG, "preload: onError: url: " + url + ", code: " + code + ",
            }
        });
    }
};
new Thread(task).start();


//取消预下载
downloadManager.stopPreload(taskID);
```

3、视频下载

视频下载支持用户在有网络的条件下下载视频，随后在无网络的环境下观看。如果是加密视频，通过播放器 SDK 下载后的视频在本地保持为加密状态，仅可通过腾讯云播放器 SDK 进行解密播放，可有效防止下载后视频的非法传播，保护视频安全。

由于 HLS 流媒体无法直接保存到本地，因此也无法通过播放本地文件的方式实现 HLS 下载到本地后播放，对于该问题，您可以通过基于 `TXVodDownloadManager` 的视频下载方案实现 HLS 的离线播放。

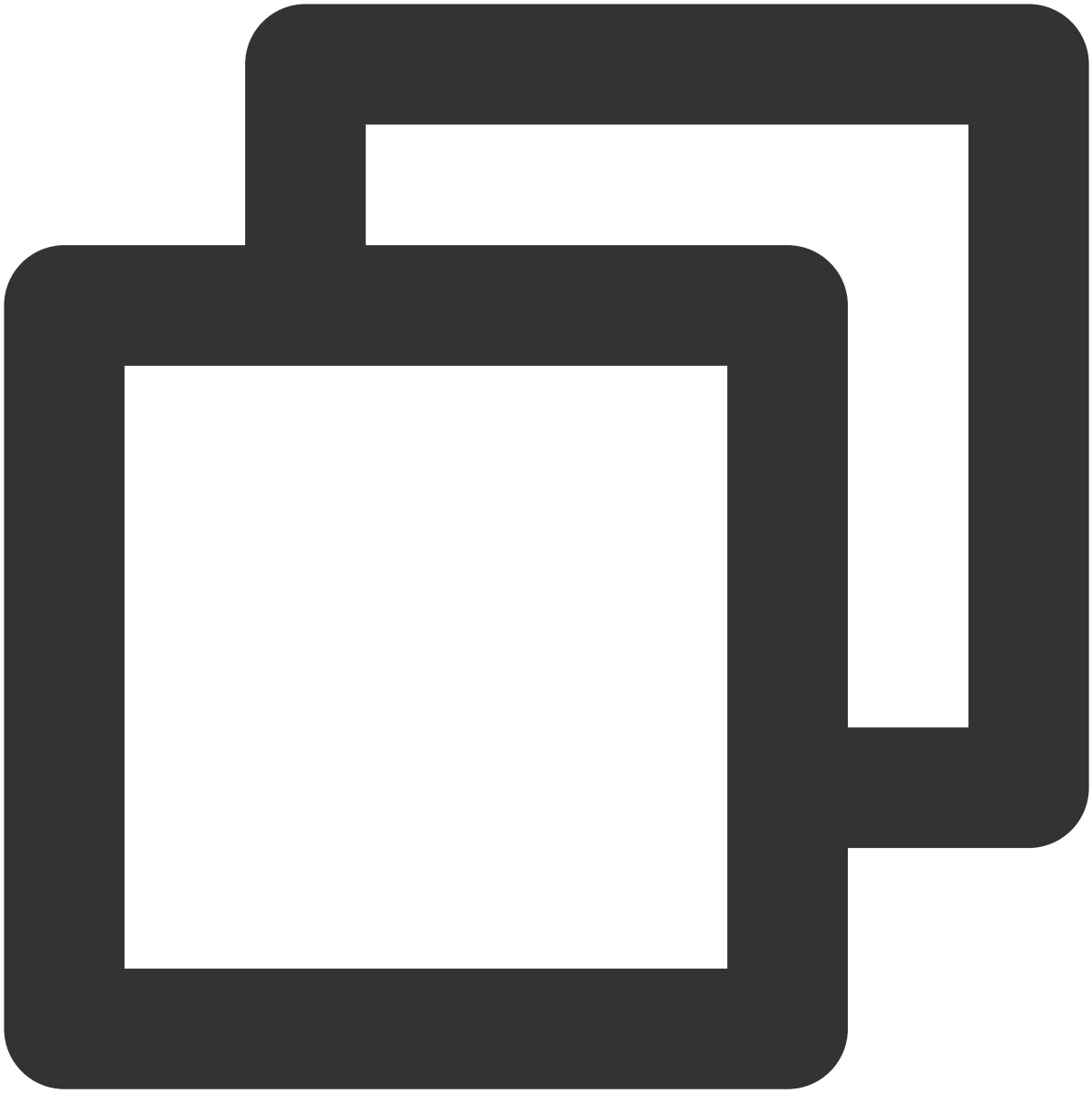
注意：

`TXVodDownloadManager` 暂不支持缓存 MP4 和 FLV 格式的文件。

播放器 SDK 已支持 MP4 和 FLV 格式的本地文件播放。

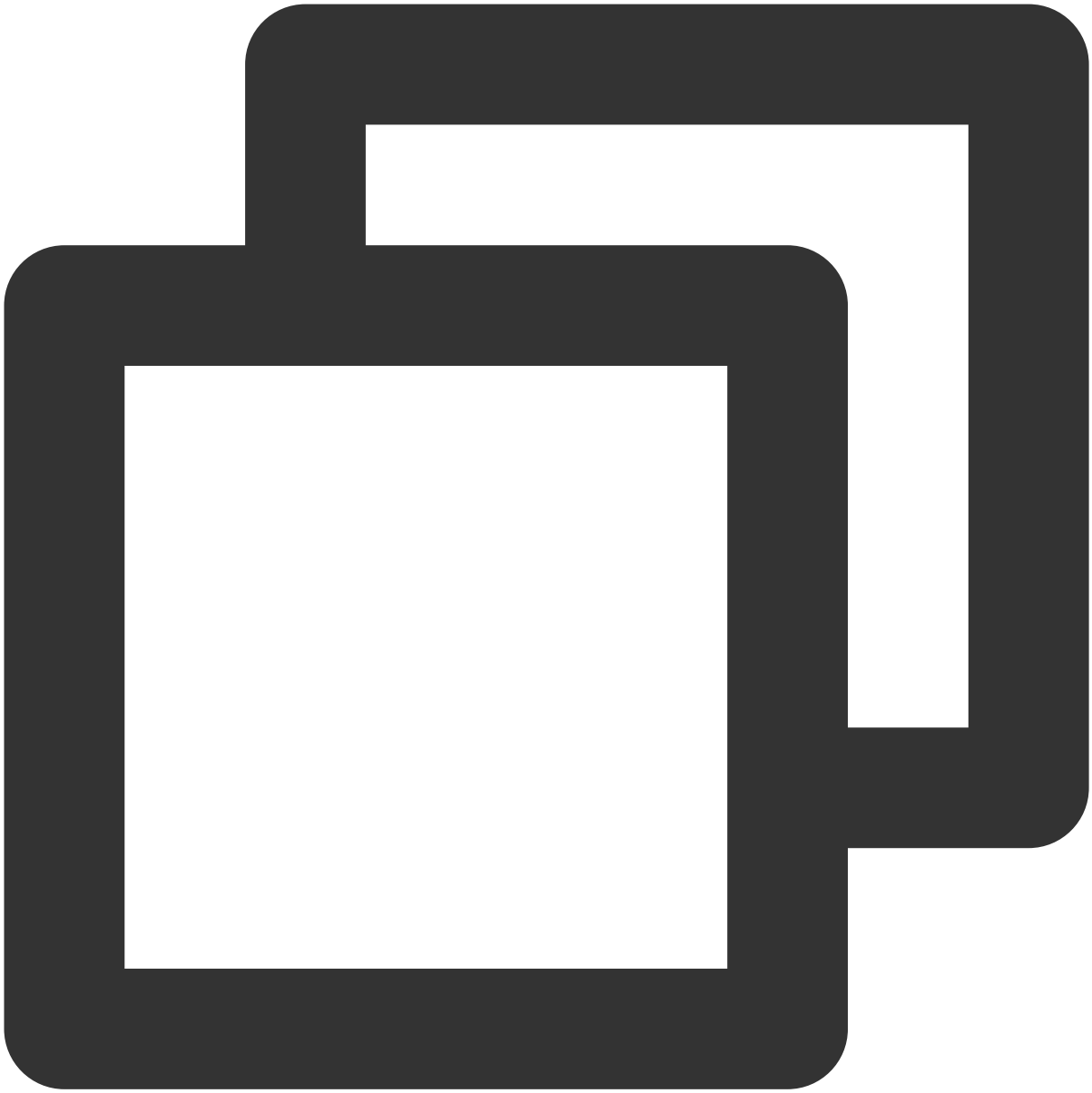
步骤1：准备工作

SDK 初始化时，设置全局存储路径，用于视频下载，预加载，和缓存等功能。用法如下：



```
File sdcardDir = context.getExternalFilesDir(null);
TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
```

`TXVodDownloadManager` 被设计为单例，因此您不能创建多个下载对象。用法如下：



```
TXVodDownloadManager downloader = TXVodDownloadManager.getInstance();
```

步骤2: 开始下载

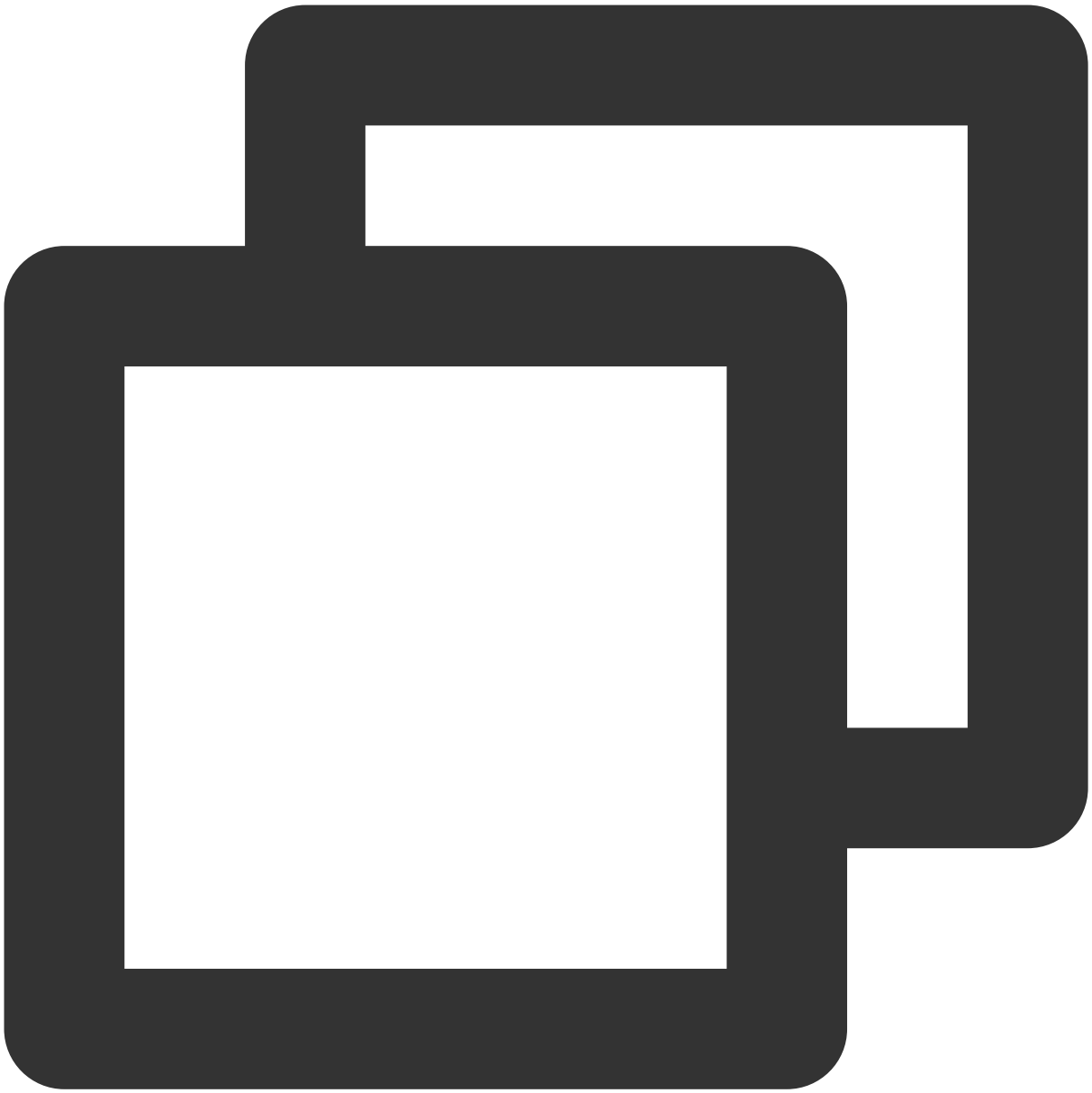
开始下载有 Fileid 和 URL 两种方式，具体操作如下：

Fileid 方式

URL 方式

Fileid 下载至少需要传入 AppID、 Fileid 和 qualityId。带签名视频需传入 pSign，userName 不传入具体值时，默认为“default”。

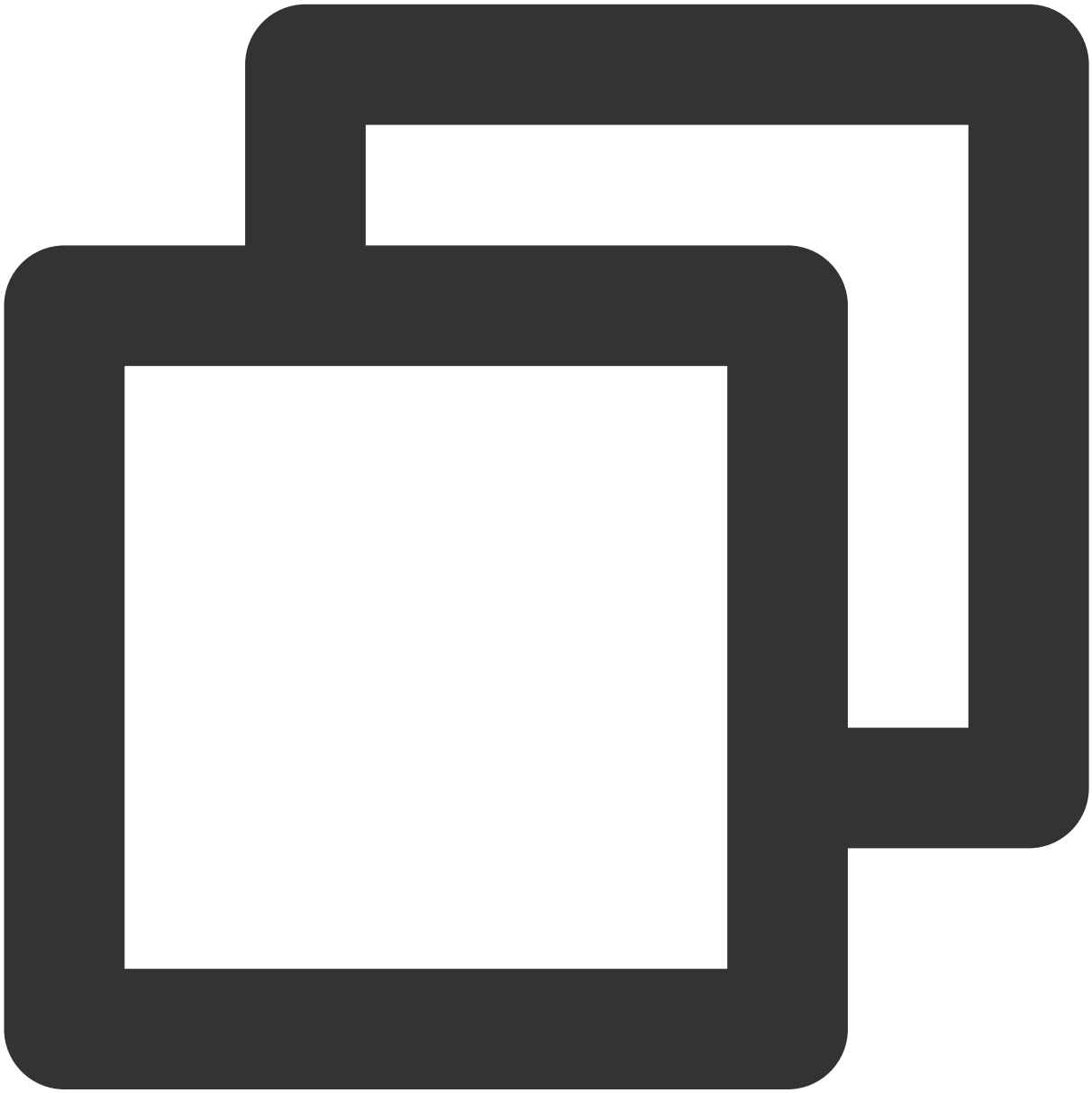
注意：加密视频只能通过Fileid下载，psign参数必须填写。



```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720，期望下载此分辨率的流，quality
```

```
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
TXVodDownloadDataSource source = new TXVodDownloadDataSource(1252463788, "456497281
downloader.startDownload(source)
```

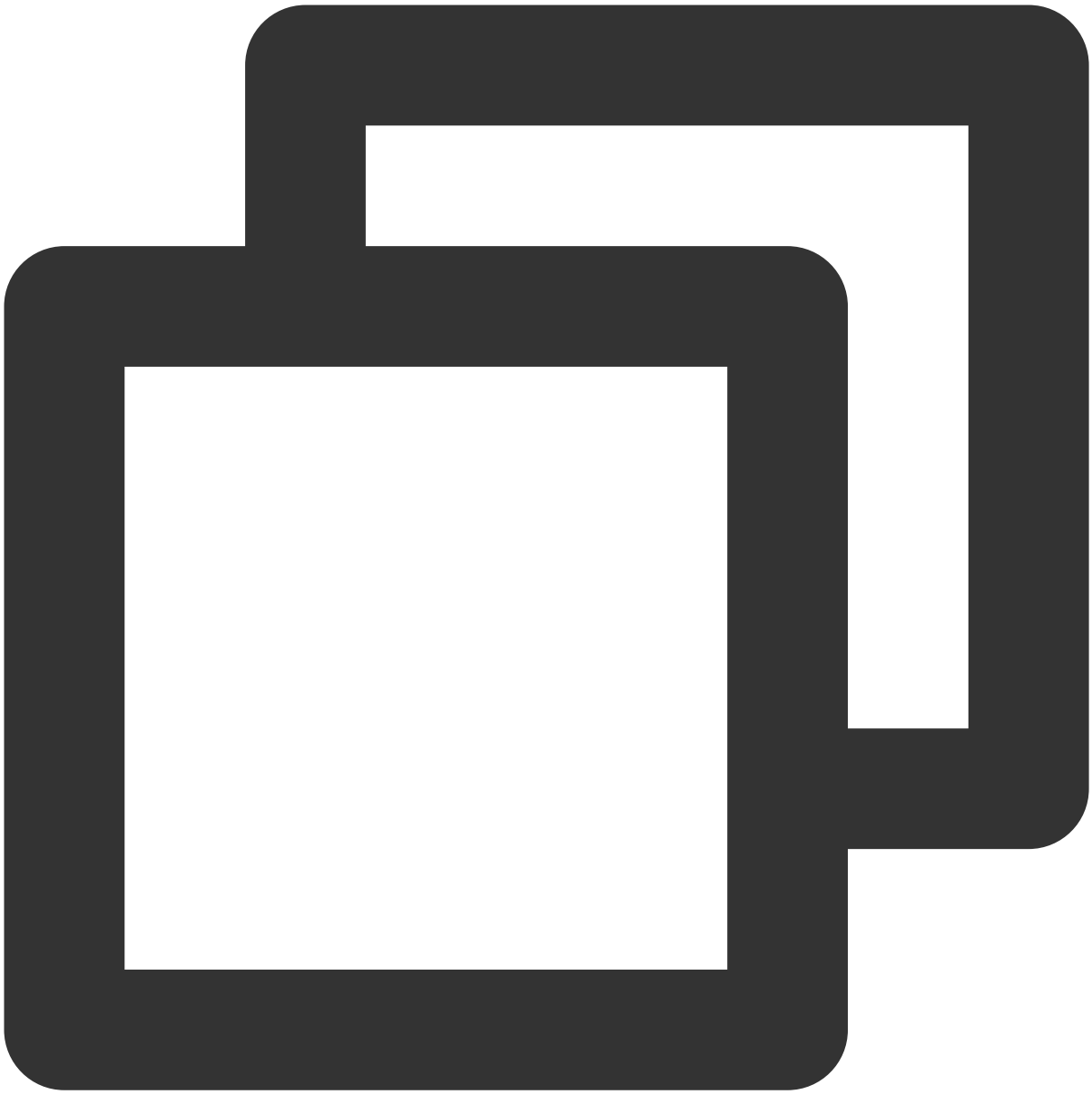
至少需要传入下载地址 URL，不支持嵌套 HLS 格式，仅支持单码流的 HLS 下载。userName 不传入具体值时，默认为"default"。



```
downloader.startDownloadUrl("http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 listener。



```
downloader.setListener(this);
```

可能收到的任务回调有：

回调信息	说明
void onDownloadStart(TXVodDownloadMediaInfo mediaInfo)	任务开始，表示 SDK 已经开始下载

void onDownloadProgress(TXVodDownloadMediaInfo mediaInfo)	任务进度，下载过程中，SDK 会频繁回调此接口，您可以通过 `mediaInfo.getProgress()` 获取当前进度
void onDownloadStop(TXVodDownloadMediaInfo mediaInfo)	任务停止，当您调用 `stopDownload` 停止下载，收到此消息表示停止成功
void onDownloadFinish(TXVodDownloadMediaInfo mediaInfo)	下载完成，收到此回调表示已全部下载。此时下载文件可以给 TXVodPlayer 播放
void onDownloadError(TXVodDownloadMediaInfo mediaInfo, int error, String reason)	下载错误，下载过程中遇到网络断开会回调此接口，同时下载任务停止。错误码位于 `TXVodDownloadManager` 中

下载错误码

错误码	数值	含义说明
DOWNLOAD_SUCCESS	0	下载成功。
DOWNLOAD_AUTH_FAILED	-5001	向云点播控制台请求视频信息失败，建议检查fileId、psign参数是否正确。
DOWNLOAD_NO_FILE	-5003	无此清晰度文件。
DOWNLOAD_FORMAT_ERROR	-5004	下载文件格式不支持。
DOWNLOAD_DISCONNECT	-5005	网络断开，建议检查网络是否正常。
DOWNLOAD_HLS_KEY_ERROR	-5006	获取 HLS 解密 Key 失败。
DOWNLOAD_PATH_ERROR	-5007	下载目录访问失败，建议检查是否有访问下载目录的权限。
DOWNLOAD_403FORBIDDEN	-5008	请求下载时，鉴权信息不通过，建议检查签名(psign) 是否已过期。

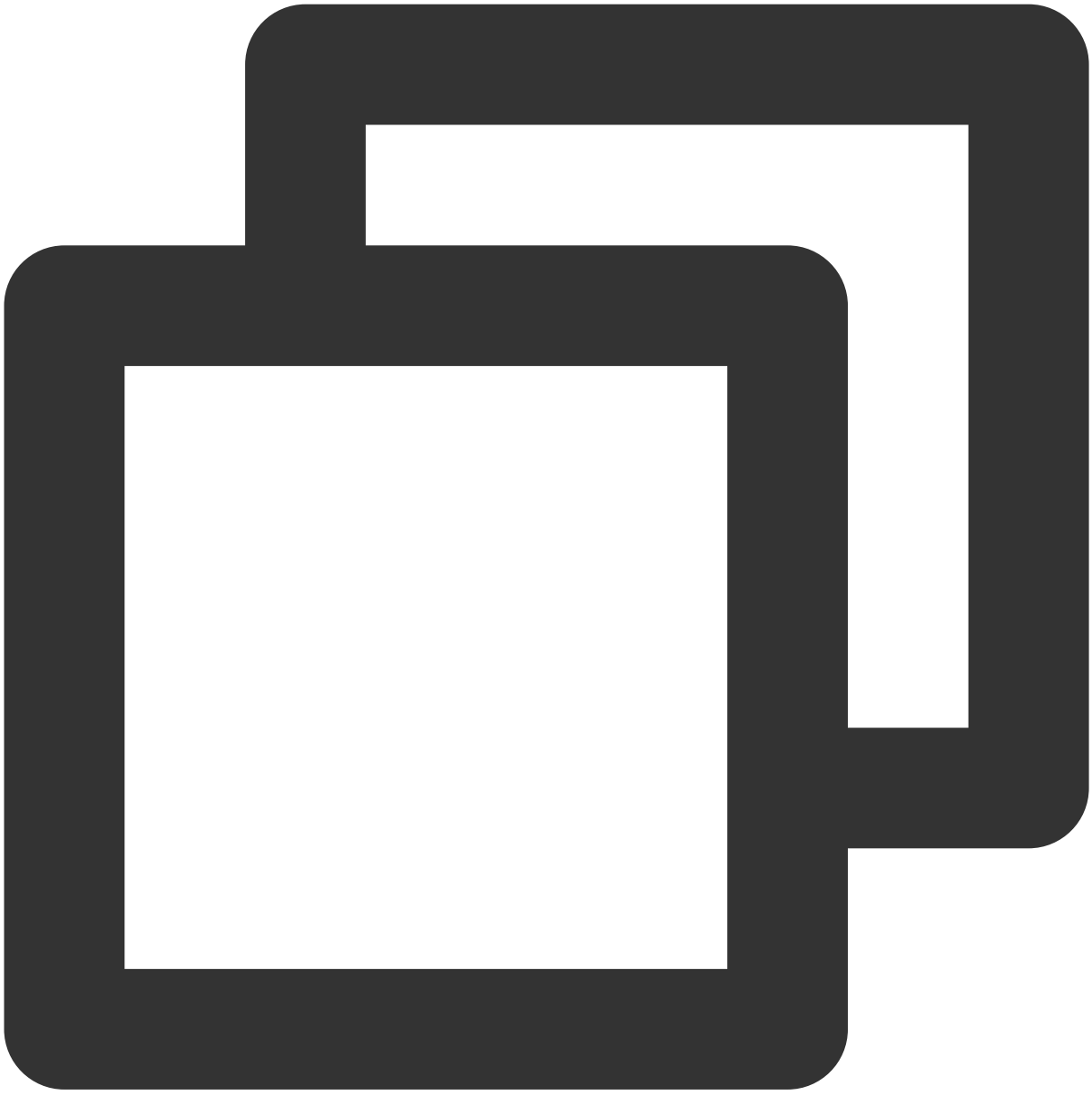
由于 downloader 可以同时下载多个任务，所以回调接口里带上了 `TXVodDownloadMediaInfo` 对象，您可以访问 URL 或 dataSource 判断下载源，同时还可以获取到下载进度、文件大小等信息。

步骤4：中断下载

停止下载请调用 `downloader.stopDownload()` 方法，参数为 `downloader.startDownload()` 返回的对象。**SDK 支持断点续传**，当下载目录没有发生改变时，下次下载同一个文件时会从上次停止的地方重新开始。

步骤5：管理下载

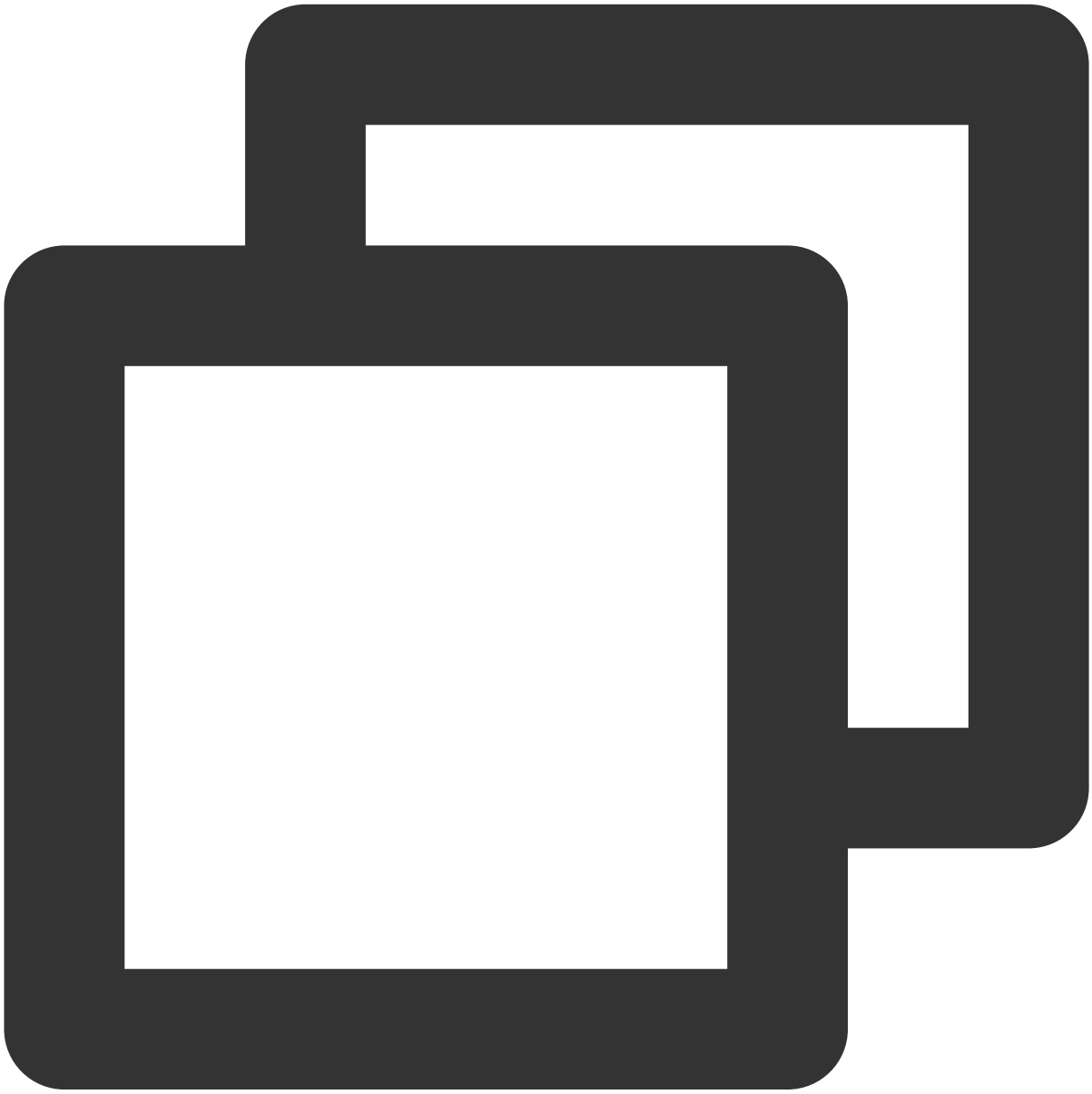
获取所有用户账户的下载列表信息, 也可获取指定用户账户的下载列表信息。



```
// 获取所有用户的下载列表信息
// 接入方可根据下载信息中的userName区分不同用户的下载列表信息
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
List<TXVodDownloadMediaInfo> downloadInfoList = downloader.getDownloadMediaInfoList
if (downloadInfoList == null || downloadInfoList.size() <= 0) return;
// 获取默认“default”用户的下载列表
List<TXVodDownloadMediaInfo> defaultUserDownloadList = new ArrayList<>();
for(TXVodDownloadMediaInfo downloadMediaInfo : downloadInfoList) {
```

```
    if ("default".equals(downloadMediaInfo.getUserName())) {
        defaultUserDownloadList.add(downloadMediaInfo);
    }
}
```

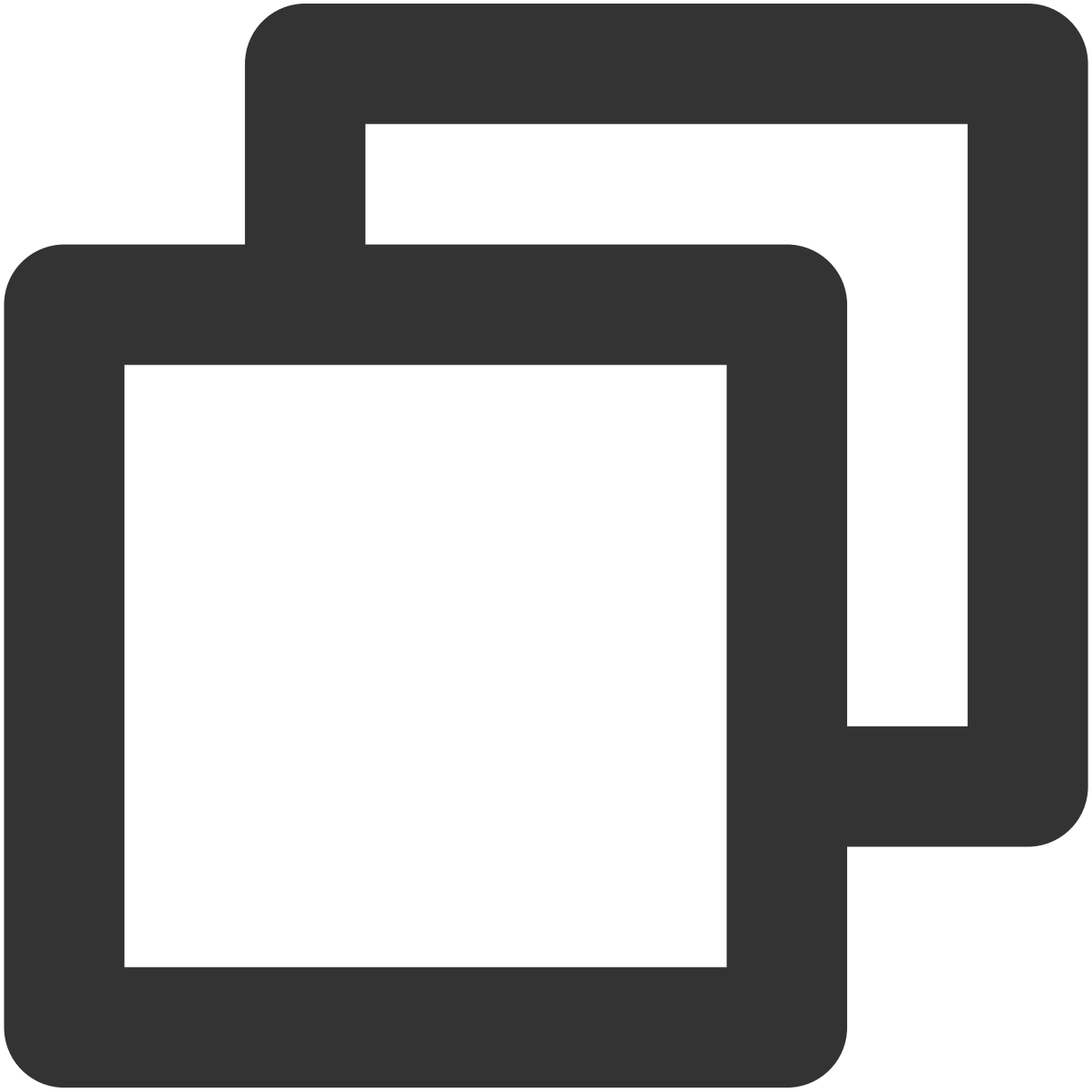
获取某个 Fileid 相关下载信息，包括当前下载状态，获取当前下载进度，判断是否下载完成等，需要传入 AppID、Fileid 和 qualityId。



```
// 获取某个fileId相关下载信息
// getDownloadMediaInfo 是耗时接口，请不要在主线程调用
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo(1252463788, "
```

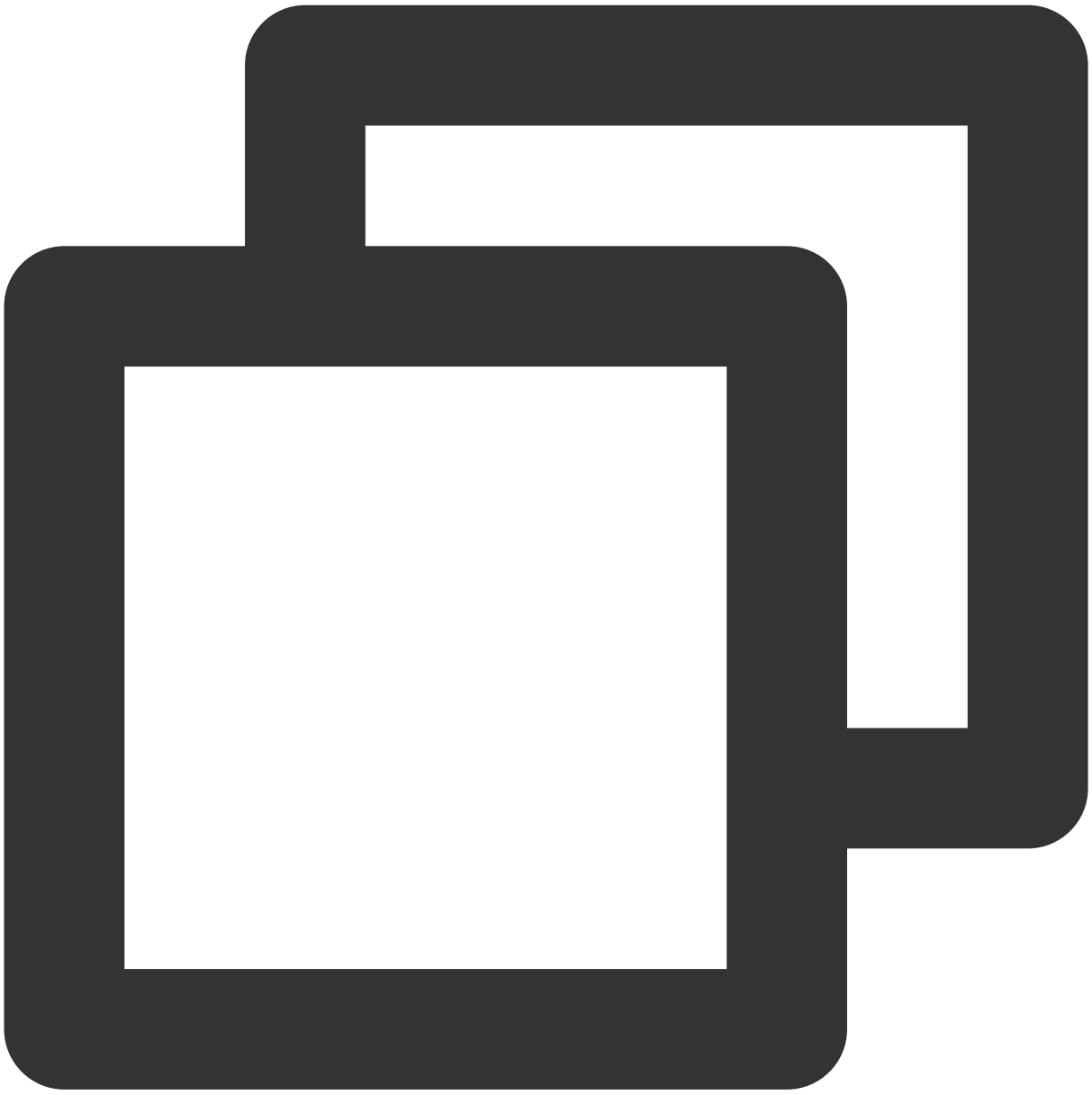
```
// 获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。
// 备注：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大小，暂时无法获取。
int size = downloadInfo.getSize();  // 获取下载文件总大小
int duration = downloadInfo.getDuration();       // 获取总时长
int playableDuration = downloadInfo.getPlayableDuration(); // 获取已下载的可播放时长
float progress = downloadInfo.getProgress();     // 获取下载进度
String playPath = downloadInfo.getPlayPath(); // 获取离线播放路径，传给播放器即可离线播放
int downloadState = downloadInfo.getDownloadState(); // 获取下载状态，具体参考STATE_xxx
boolean isDownloadFinished = downloadInfo.isDownloadFinished(); // 返回true表示下载完
```

获取某个 URL 相关下载信息，需要传入 URL 信息。



```
// 获取某个url下载信息， 耗时接口，请不要在主线程调用
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo("http://12531
```

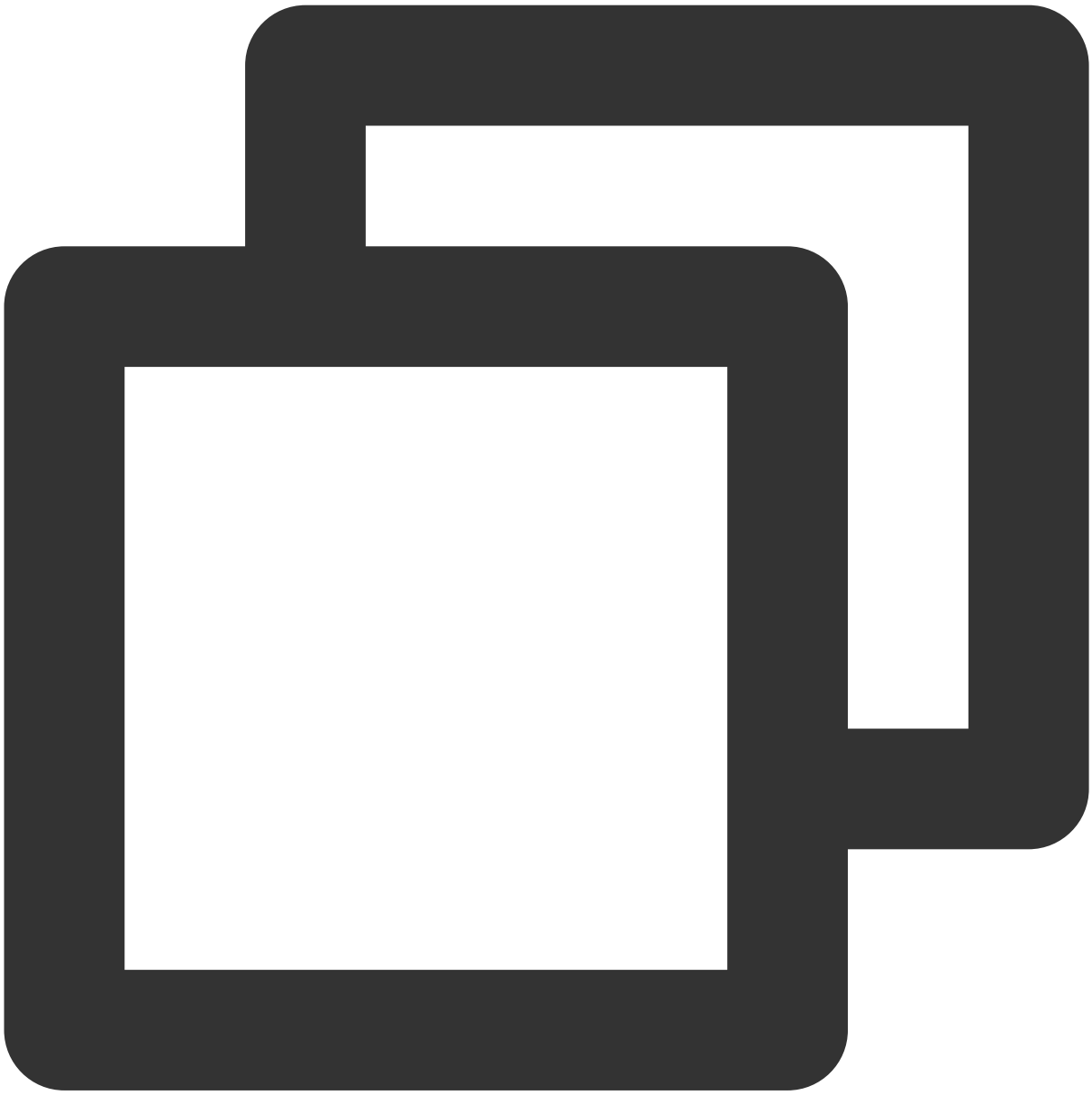
删除下载信息和相关文件，需传入 TXVodDownloadMediaInfo 参数。



```
// 删除下载信息
boolean deleteRst = downloader.deleteDownloadMediaInfo(downloadInfo);
```

步骤6：下载后离线播放

下载后的视频支持无网络的情况下进行播放，无需进行联网。下载完成后，通过 `TXVodDownloadMediaInfo#getPlayPath` 获取到下载地址即可进行播放。

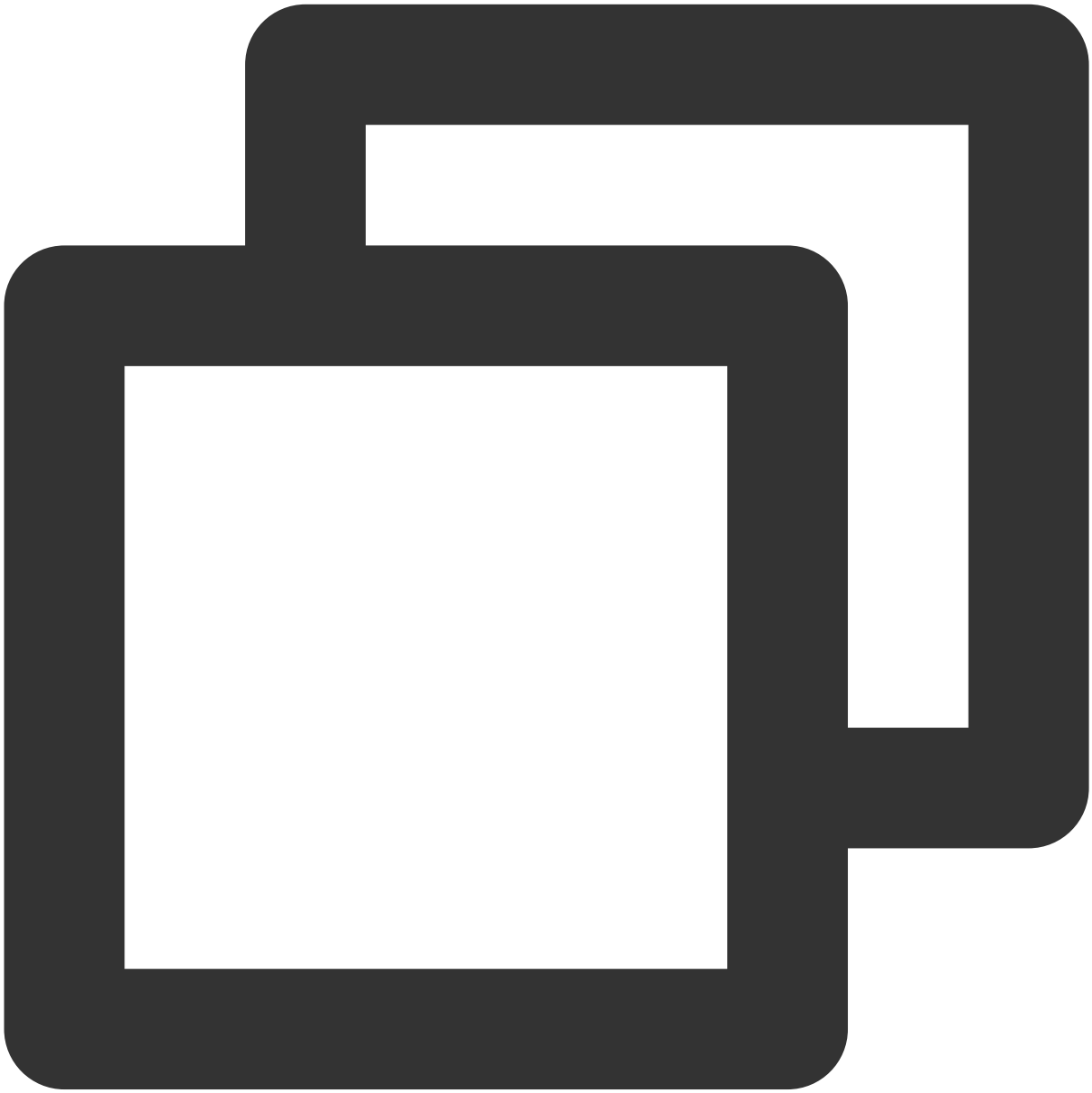


```
// getDownloadMediaInfoList 是耗时接口，请不要在主线程调用
List<TXVodDownloadMediaInfo> mediaInfoList = TXVodDownloadManager.getInstance().get
// 业务侧根据实际需求查找到需要播放的 media 对象
for (TXVodDownloadMediaInfo mediaInfo : mediaInfoList) {
    if (mediaInfo.getDownloadState() == TXVodDownloadMediaInfo.STATE_FINISH) {  //
        mVodPlayer.startVodPlay(mediaInfo.getPlayPath());  // 播放已经下载完成的视频
    }
}
```

4、加密播放

视频加密方案主要用于在线教育等需要对视频版权进行保护的场景。如果要对您的视频资源进行加密保护，就不仅需要在播放器上做改造，还需要对视频源本身进行加密转码，亦需要您的后台和终端研发工程师都参与其中。在 视频加密解决方案 中您会了解到全部细节内容。

在腾讯云控制台提取到appId， 加密视频的fileId 和 psign后，可以通过下面的方式进行播放：

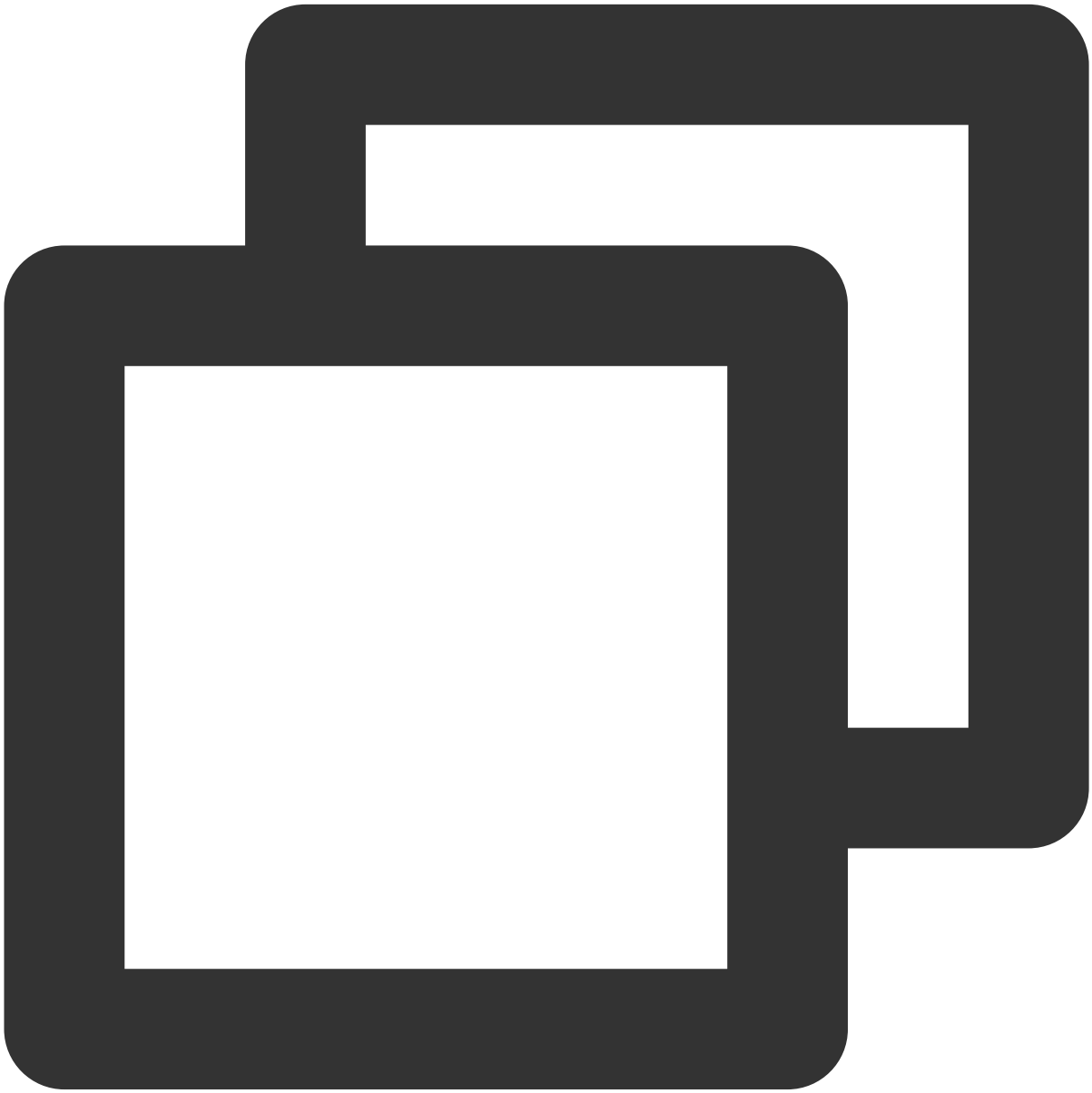


```
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // 腾讯云账户的appI
    "4564972819220421305", // 视频的fileId
```

```
    "psignxxxxxxx"); // 播放器签名
mVodPlayer.startVodPlay(playInfoParam);
```

5、播放器配置

在调用 statPlay 之前可以通过 setConfig 对播放器进行参数配置，例如：设置播放器连接超时时间、设置进度回调间隔、设置缓存文件个数等配置，TXVodPlayConfig 支持配置的详细参数请单击 基础配置接口 了解。使用示例：

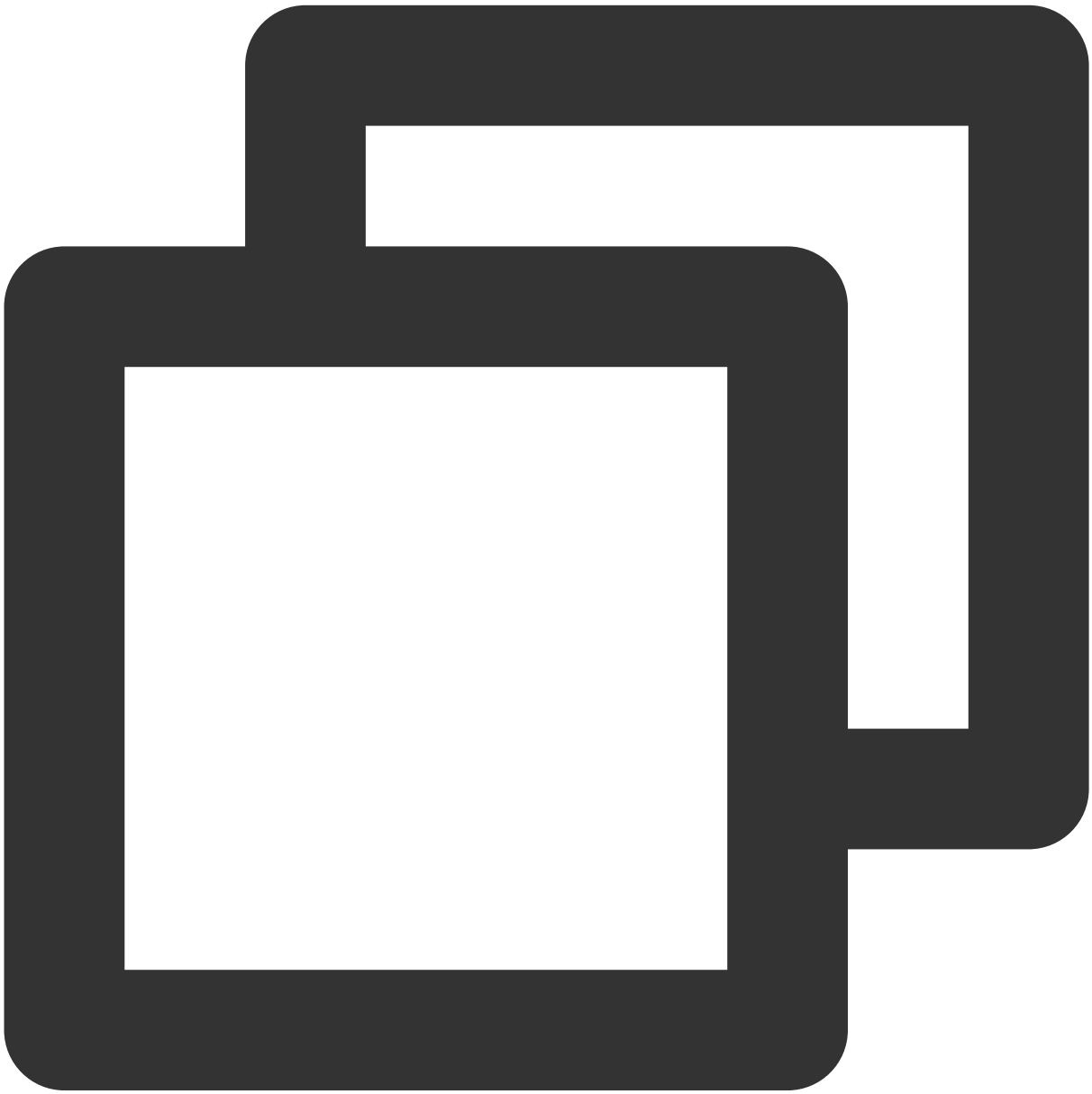


```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setEnableAccurateSeek(true);  // 设置是否精确 seek，默认 true
config.setMaxCacheItems(5);  // 设置缓存文件个数为5
```

```
config.setProgressInterval(200);  // 设置进度回调间隔，单位毫秒
config.setMaxBufferSize(50);  // 最大预加载大小，单位 MB
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

启播时指定分辨率

播放 HLS 的多码率视频源，如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率。 播放器会查找小于或等于偏好分辨率的流进行启播，启播后没有必要再通过 setBitrateIndex 切换到需要的码流。



```
TXVodPlayConfig config = new TXVodPlayConfig();
// 传入参数为视频宽和高的乘积(宽 *  高)，可以自定义值传入
```

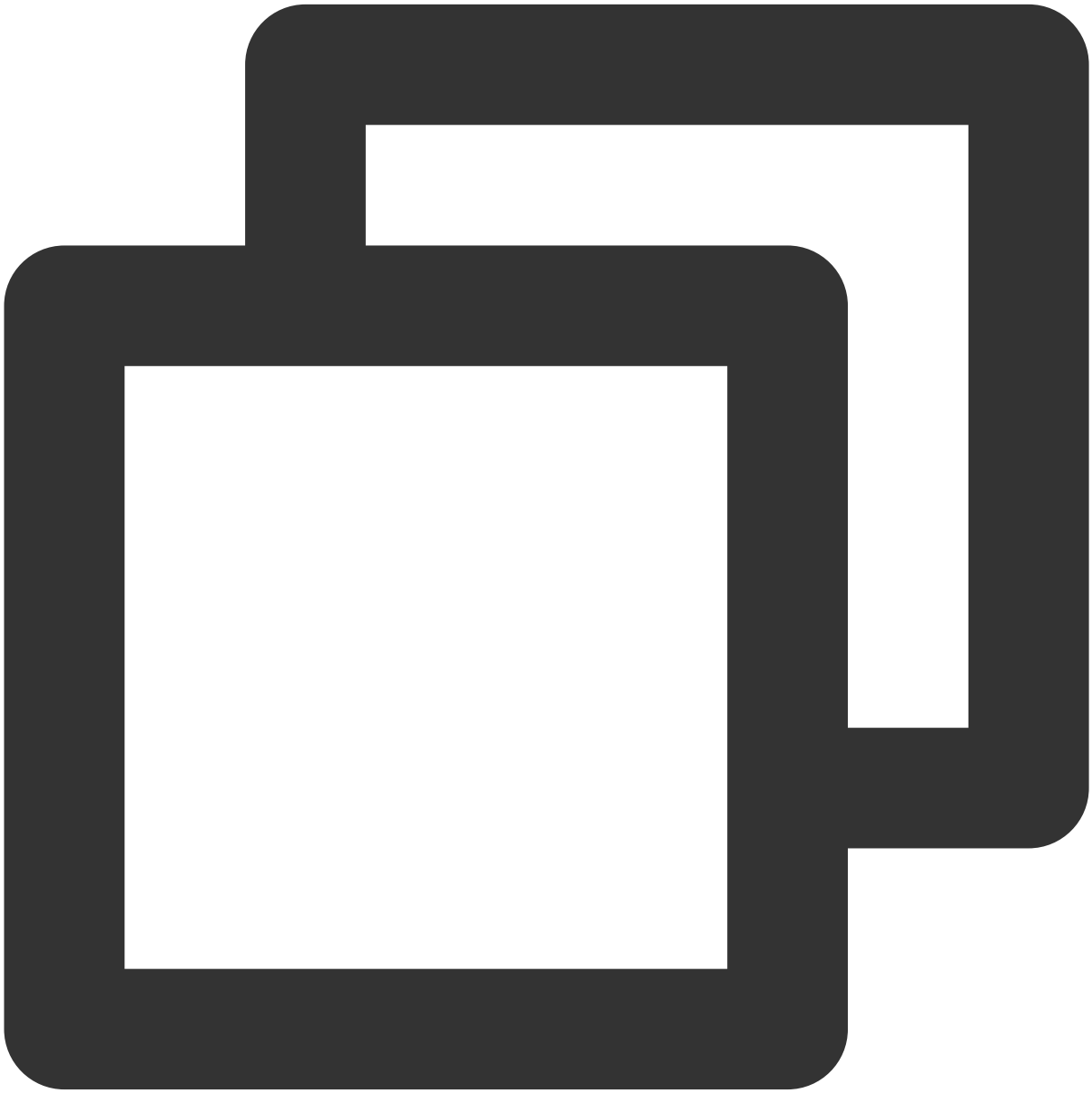
```
config.setPreferredResolution(TXLiveConstants.VIDEO_RESOLUTION_720X1280);
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

启播前指定媒资类型

当提前知道播放的媒资类型时，可以通过配置 `TXVodPlayConfig#setMediaType` 减少播放器SDK内部播放类型探测，提升启播速度。

注意：

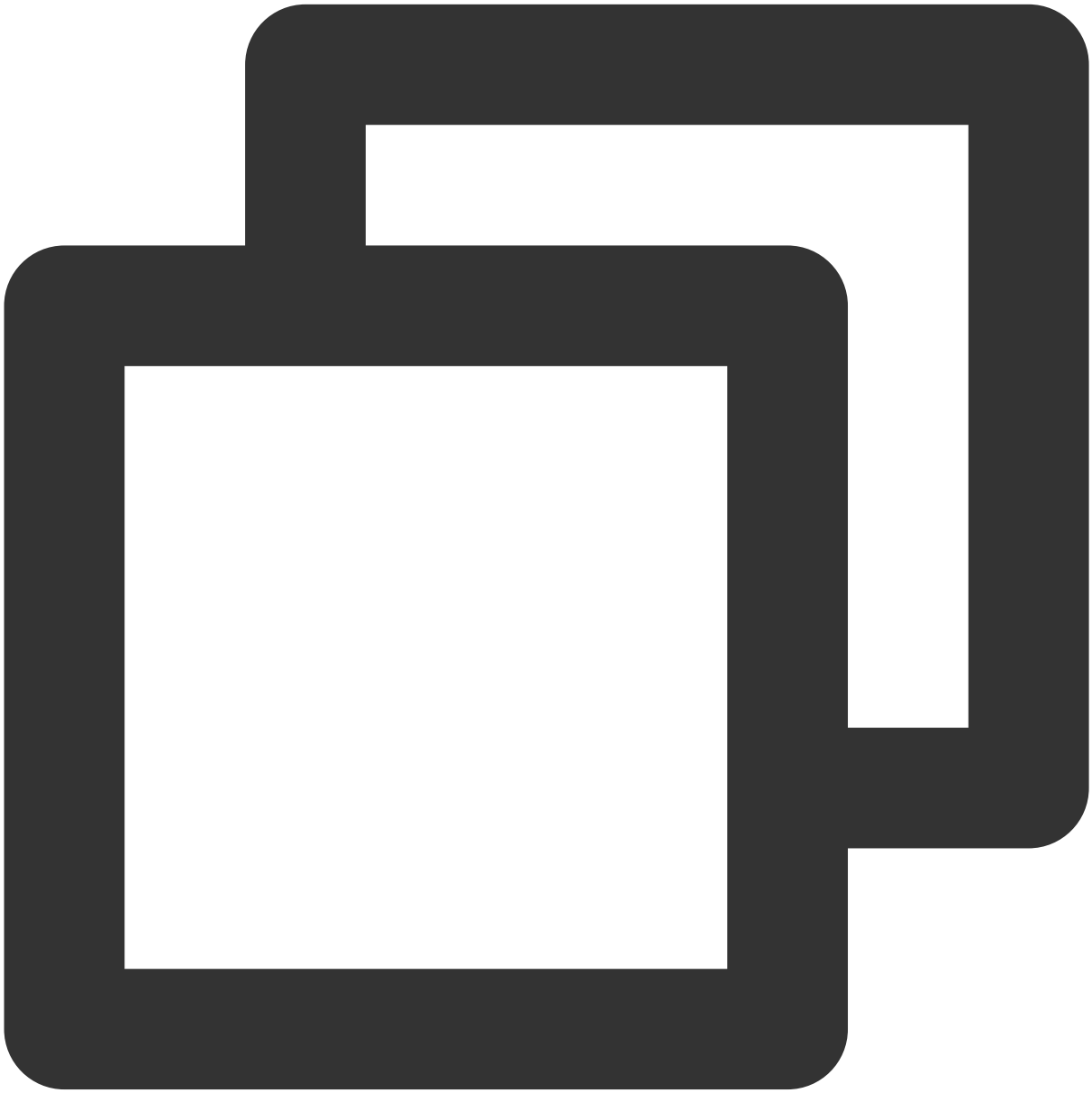
`TXVodPlayConfig#setMediaType` 11.2 版本开始支持。



```
TXVodPlayConfig config = new TXVodPlayConfig();
```

```
config.setMediaType(TXVodConstants.MEDIA_TYPE_FILE_VOD);  // 用于提升MP4启播速度
// config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_VOD);   // 用于提升HLS启播速度
mVodPlayer.setConfig(config);
```

设置播放进度回调时间间隔



```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setProgressInterval(200);  // 设置进度回调间隔，单位毫秒
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

6、HttpDNS 解析服务

移动解析（HTTPDNS）基于 HTTP 协议向 DNS 服务器发送域名解析请求，替代了基于 DNS 协议向运营商 Local DNS 发起解析请求的传统方式，可避免 Local DNS 造成域名劫持和跨网访问问题，解决移动互联网服务中域名解析异常带来的视频播放失败困扰。

注意：

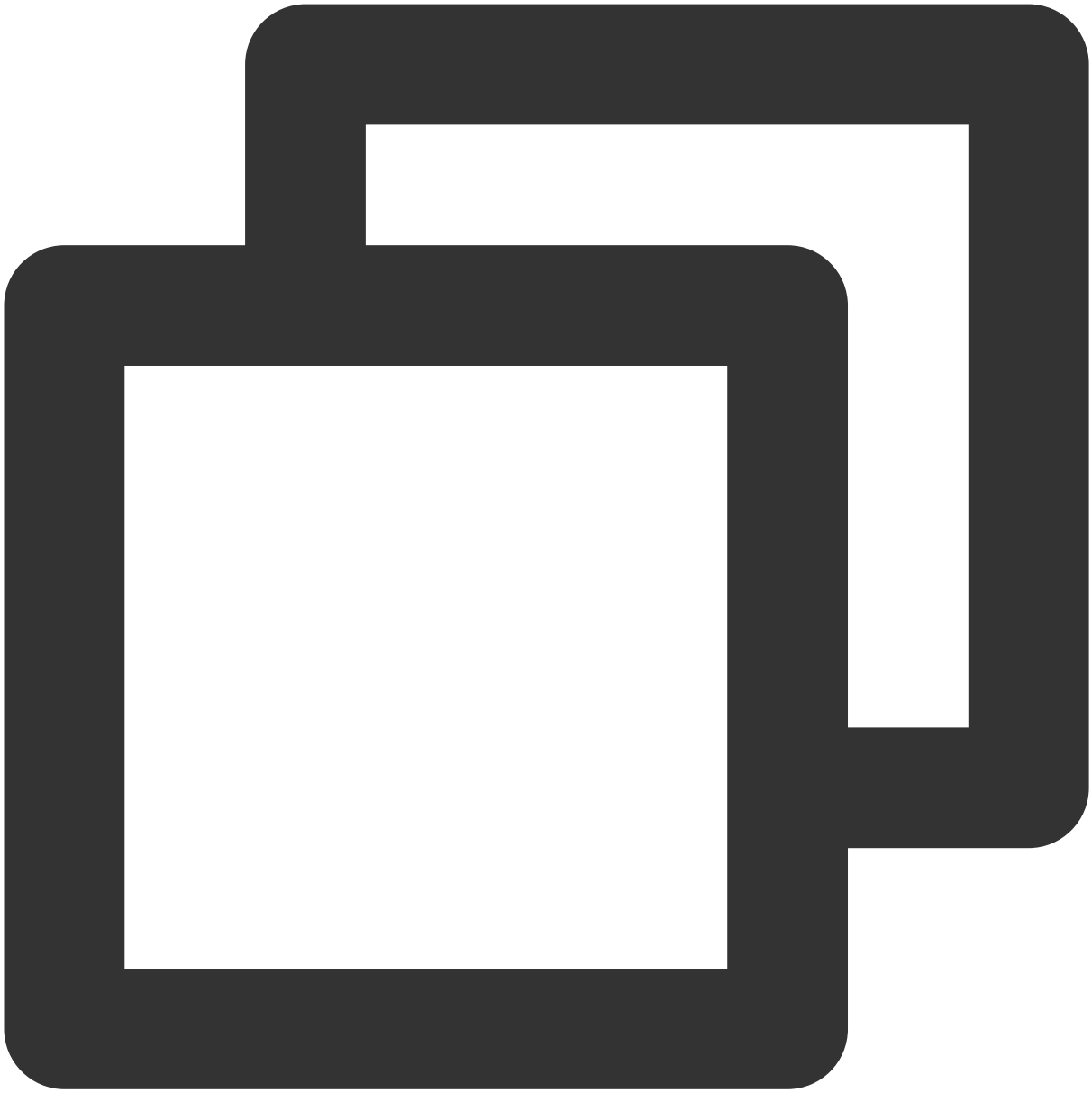
HttpDNS 解析服务从 10.9 版本开始支持。

1. 开通 HTTPDNS 解析服务

您可以选择腾讯云或其它云提供商，开通 HTTPDNS 解析服务，确保开通成功后， 再集成到播放 SDK。

2. 在播放 SDK 接入 HTTPDNS 解析服务

下面以接入 腾讯云 HTTPDNS 为例子，展示如何在播放器 SDK 接入：

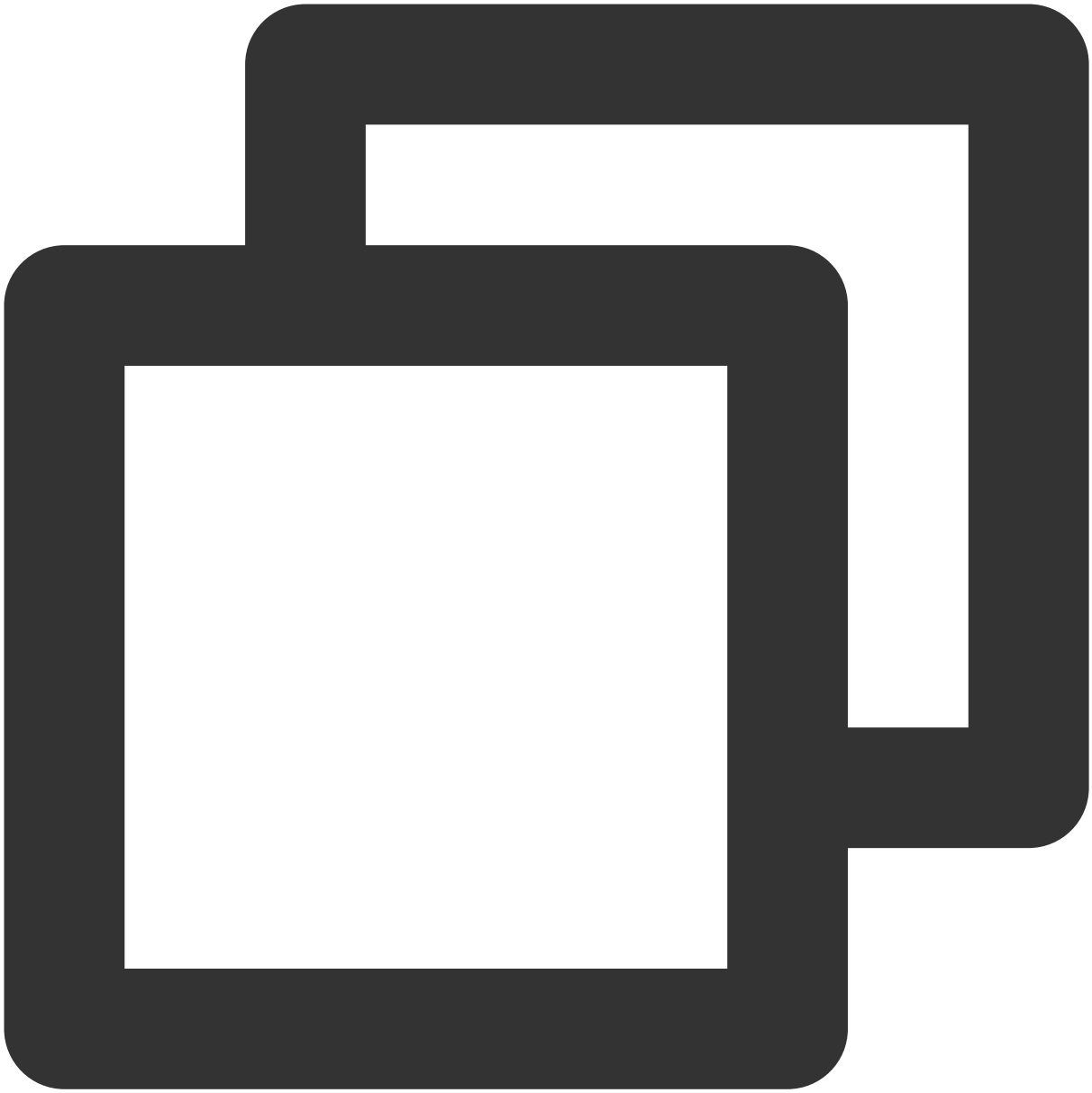


```
// 步骤1： 打开 HttpDNS 解析开关
TXLiveBase.enableCustomHttpDNS(true);
// 步骤2： 设置 HttpDNS 解析回调， TXLiveBaseListener#onCustomHttpDNS
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onCustomHttpDNS(String hostName, List<String> ipList) {
        // 把 hostName 解析到 ip 地址后，保存到 iPList，返回给 SDK 内部。注意：这里不要进行异
        // MSDKDnsResolver 是腾讯云提供的 HTTPDNS SDK 解析接口
        String ips = MSDKDnsResolver.getInstance().getAddrByName(hostname);
        String[] ipArr = ips.split(";");
        if (0 != ipArr.length) {
```

```
                for (String ip : ipArr) {
                    if ("0".equals(ip)) {
                        continue;
                    }
                    ipList.add(ip);
                }
            }
        }
    });
```

7、HEVC 自适应降级播放

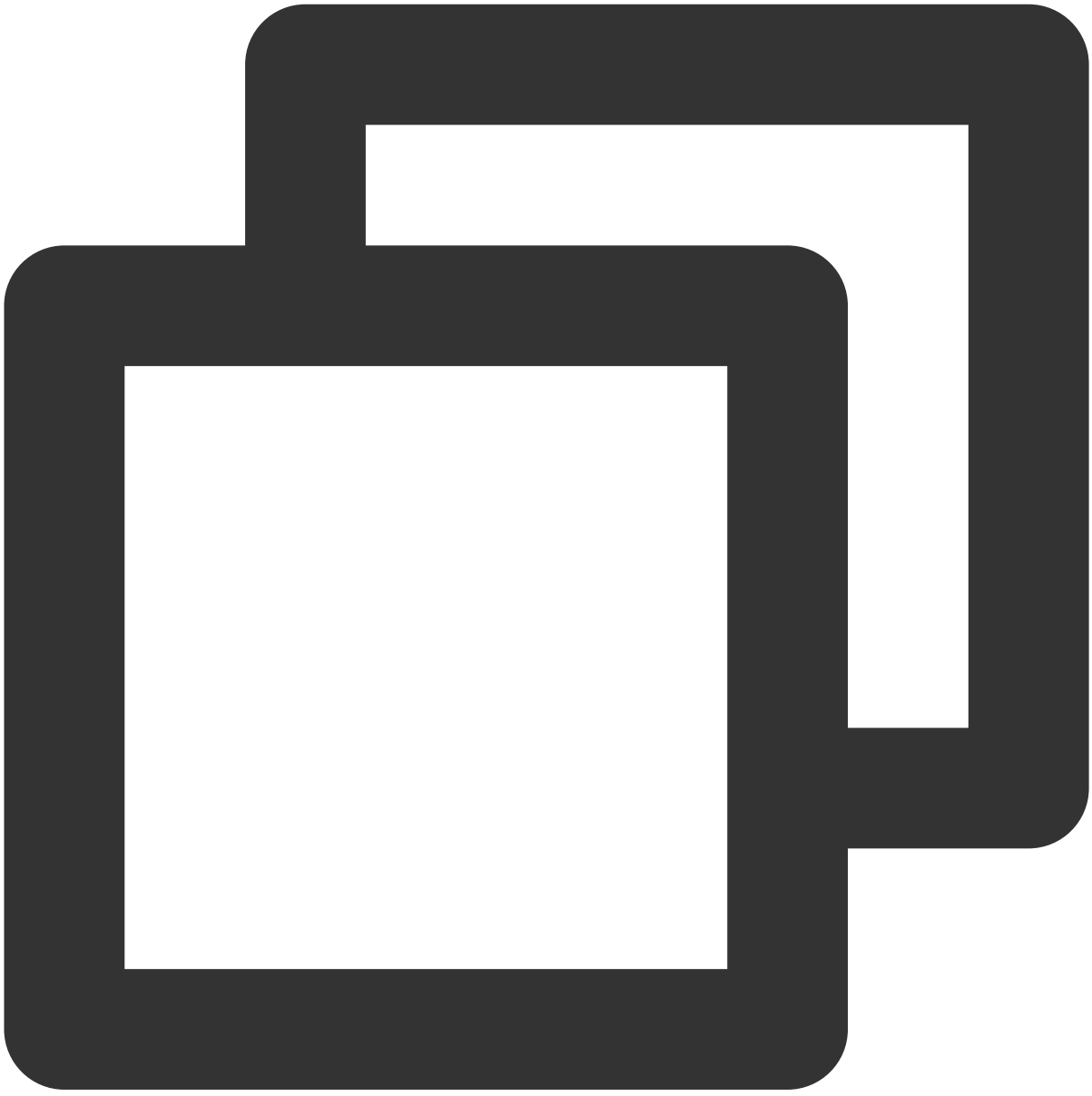
播放器支持同时传入 HEVC 和其它视频编码格式例如： H.264 的播放链接， 当播放机型不支持 HEVC 格式时，将自动降级为配置的其它编码格式（如： H.264 ）的视频播放。
注意：播放器高级版 11.7 版本开始支持。



```
//设置备选播放链接
String backupPlayUrl = "${backupPlayUrl}";  // 备选播放链接
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_MIMETYPE, TXVodConstants.VOD_PLAY
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_BACKUP_URL, backupPlayUrl); // 设

// 设置原始 HEVC 播放链接
String hevcPlayUrl = "${hevcPlayUrl}";
mVodPlayer.startVodPlay(hevcPlayUrl);
```

8、音量均衡

播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。通过 `TXVodPlayer#setAudioNormalization` 设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。
注意：播放器高级版 11.7 版本开始支持。



```
/**
 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h）
 关：AUDIO_NORMALIZATION_OFF
 开：AUDIO_NORMALIZATION_STANDARD（标准）
    AUDIO_NORMALIZATION_LOW （低）
    AUDIO_NORMALIZATION_HIGH（高）
```

```
 可填自定义数值：从低到高，范围-70 - 0 LUFS
*/
mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD);  //启

mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_OFF);       //关
```

播放器事件监听

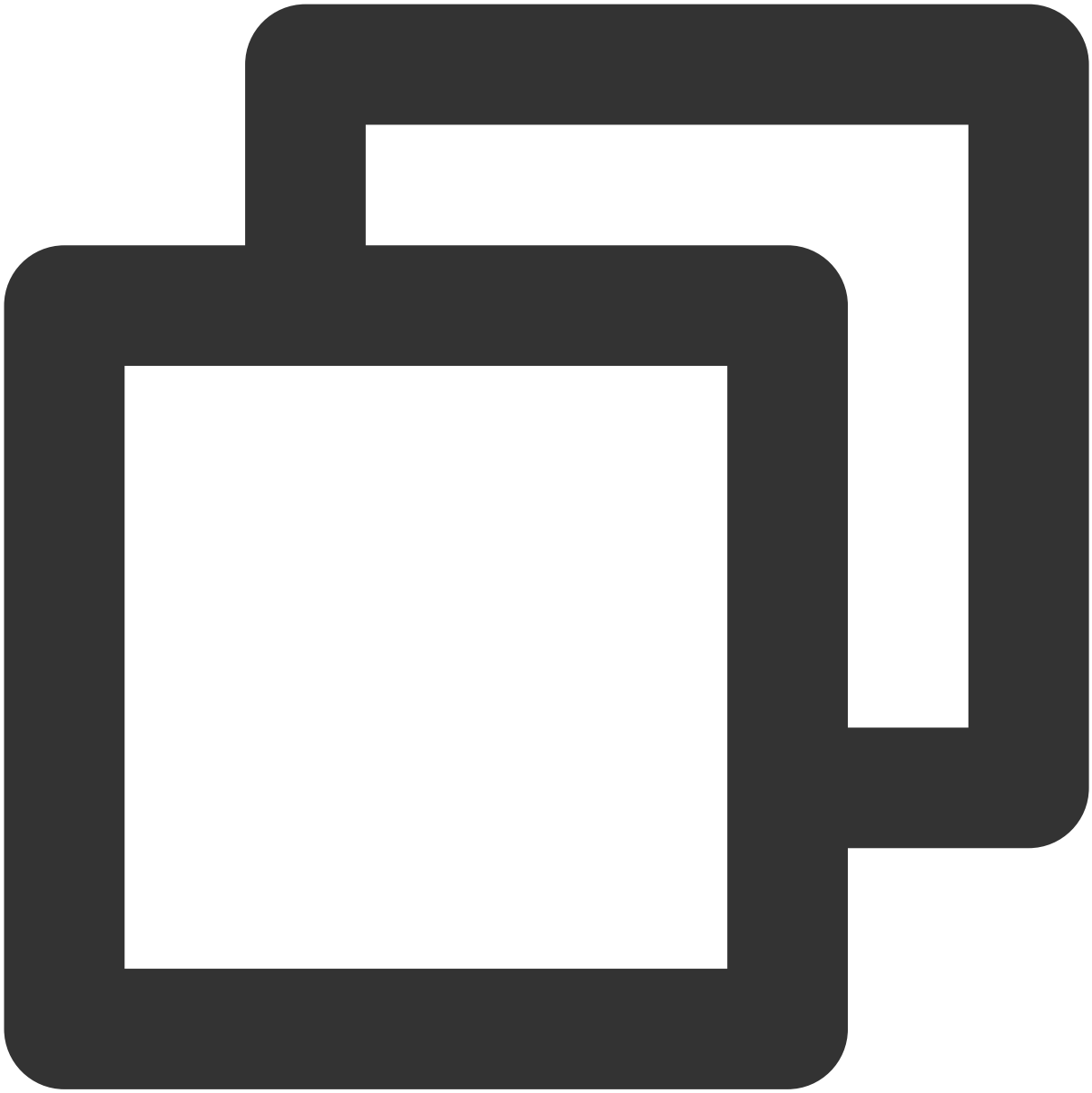
您可以为 TXVodPlayer 对象绑定一个 TXVodPlayListener 监听器，即可通过onPlayEvent（事件通知） 和 onNetStatus（状态反馈）向您的应用程序同步信息。

播放事件通知（onPlayEvent）

事件 ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度 和总体时长
PLAY_EVT_PLAY_LOADING	2007	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件
PLAY_EVT_VOD_LOADING_END	2014	视频播放 loading 结束，视频继续播放
TXVodConstants.VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seek 完成，10.3版本开始支持
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	循环播放，一轮播放结束（10.8 版本开始支持）
TXVodConstants.VOD_PLAY_EVT_HIT_CACHE	2002	启播时命中缓存事件（11.2 版本开始支持）。
TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI	2030	收到 SEI 帧事件（ 播放器高级版11.6 版本开始支持）。
VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	2031	发生 HEVC 降级播放（播放器高级版12.0版本开始支持）。
VOD_PLAY_EVT_FIRST_VIDEO_PACKET	2017	播放器收到首帧数据包事件（12.0 版本开始支持）。

SEI 帧

SEI（Supplemental Enhancement Information）帧是一种用于传递附加信息的帧类型，播放器高级版会解析视频流中的 SEI 帧，通过
VOD_PLAY_EVT_VIDEO_SEI 事件回调， 注意：播放器高级版 11.6 版本开始支持。



```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = param.getInt(TXVodConstants.EVT_KEY_SEI_TYPE);  //  the type
        int seiSize = param.getInt(TXVodConstants.EVT_KEY_SEI_SIZE);  // the data s
        byte[] seiData = param.getByteArray(TXVodConstants.EVT_KEY_SEI_DATA);  // t
```

```
        }
    }
```

结束事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败

警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件 ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解

连接事件

连接服务器的事件，主要用于测定和统计服务器连接时间：

事件 ID	数值	含义说明
PLAY_EVT_VOD_PLAY_PREPARED	2013	播放器已准备完成，可以播放。设置了 autoPlay 为 false 之后，需要在收到此事件后，调用 resume 才会开始播放
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包（IDR）

画面事件

以下事件用于获取画面变化信息：

事件 ID	数值	含义说明
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率改变

PLAY_EVT_CHANGE_ROTATION	2011	MP4 视频旋转角度

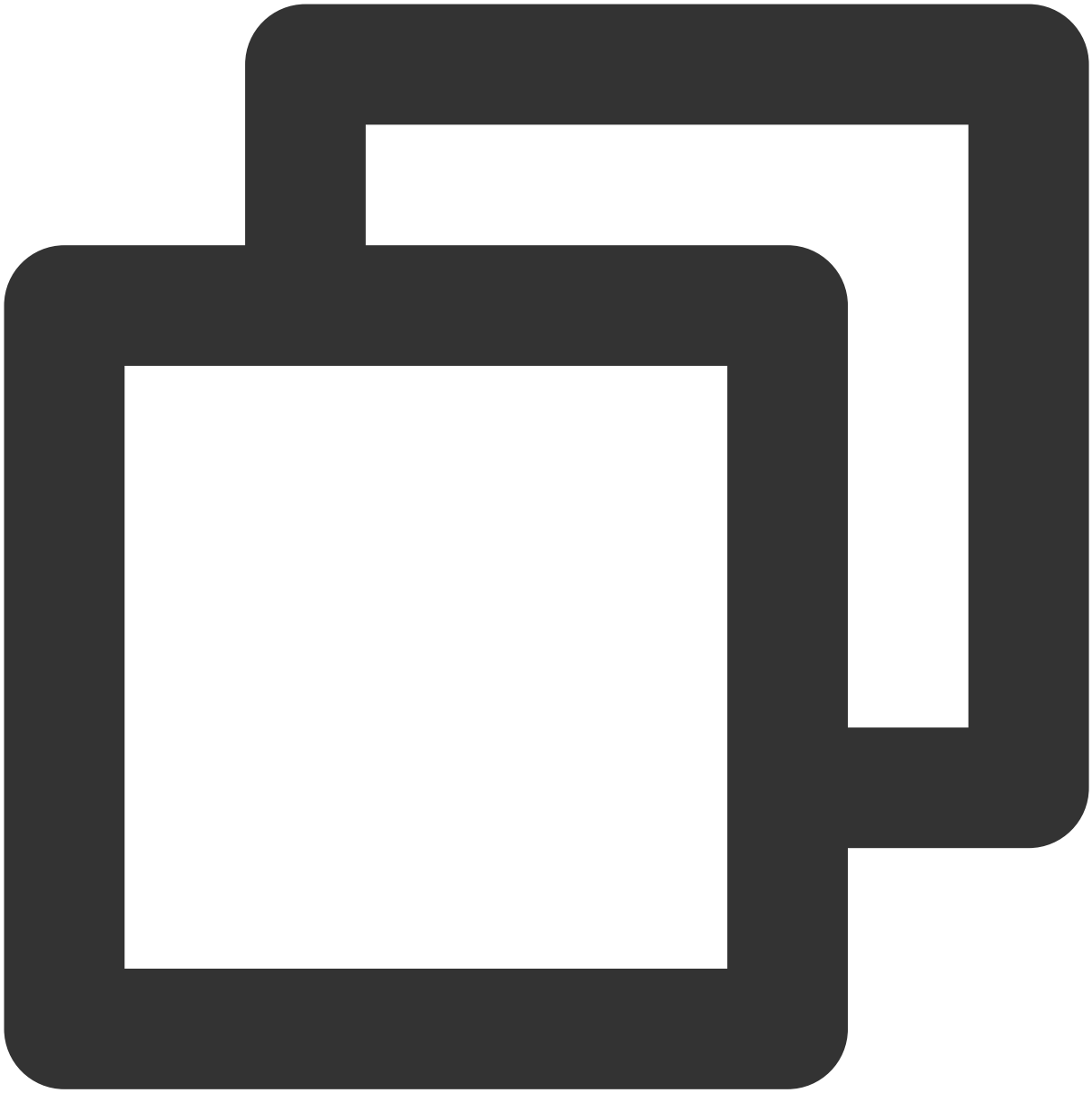
视频信息事件

事件 ID	数值	含义说明
TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC	2010	成功获取播放文件信息

如果通过 fileId 方式播放且请求成功（接口：startVodPlay(TXPlayInfoParams playInfoParams)），SDK 会将一些请求信息通知到上层。您可以在收到 `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` 事件后，解析 param 获取视频信息。

视频信息	含义说明
EVT_PLAY_COVER_URL	视频封面地址
EVT_PLAY_URL	视频播放地址
EVT_PLAY_DURATION	视频时长
EVT_DESCRIPTION	事件说明
EVT_PLAY_NAME	视频名称
TXVodConstants.EVT_IMAGESPRIT_WEBVTTURL	雪碧图 web vtt 描述文件下载 URL，10.2版本开始支持
TXVodConstants.EVT_IMAGESPRIT_IMAGEURL_LIST	雪碧图图片下载 URL，10.2版本开始支持
TXVodConstants.EVT_DRM_TYPE	加密类型，10.2版本开始支持
TXVodConstants.EVT_KEY_WATER_MARK_TEXT	幽灵水印文本内容（11.6 版本开始支持）。

通过 onPlayEvent 获取视频播放过程信息示例：



```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
            // 收到播放器已经准备完成事件，此时可以调用pause、resume、getWidth、getSupporte
        } else if (event == TXLiveConstants.PLAY_EVT_PLAY_BEGIN) {
            // 收到开始播放事件
        } else if (event == TXLiveConstants.PLAY_EVT_PLAY_END) {
            // 收到开始结束事件
        }
    }
```

```
    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
    }
});
```

幽灵水印

幽灵水印内容在播放器签名中填写，经云点播后台，最终展示到播放端上，整个传输链路过程由云端和播放端共同协作，确保水印的安全。在播放器签名中 配置幽灵水印教程。幽灵水印的内容在收到播放器的 `TXVodConstants#VOD_PLAY_EVT_GET_PLAYINFO_SUCC` 事件后， 通过 `param.getString(TXVodConstants.EVT_KEY_WATER_MARK_TEXT)` 获取。详细使用教程参见 超级播放器组件 > 幽灵水印。

注意： 播放器 11.6 版本开始支持。

播放错误事件

说明：

[-6004 , -6010] 错误事件 11.0 版本开始支持。

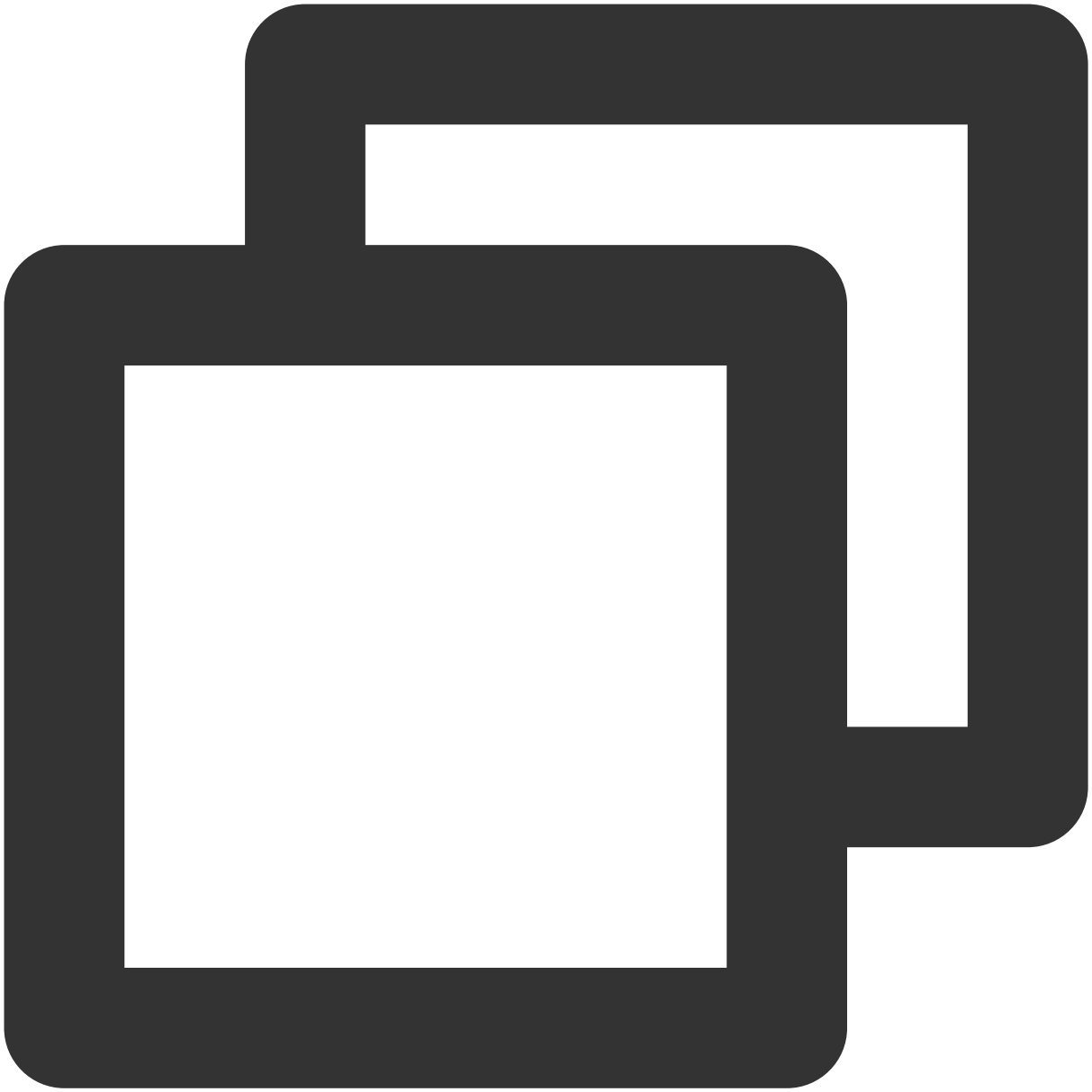
事件 ID	数值	含义说明
PLAY_ERR_NET_DISCONNECT	-2301	视频数据错误导致重试亦不能恢复正常播放。如：网络异常或下载数据错误，导致解封装超时或失败。
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败。
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	系统播放器播放错误。
VOD_PLAY_ERR_DECODE_VIDEO_FAIL	-6006	视频解码错误，视频格式不支持。
VOD_PLAY_ERR_DECODE_AUDIO_FAIL	-6007	音频解码错误，音频格式不支持。
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	字幕解码错误。
VOD_PLAY_ERR_RENDER_FAIL	-6009	视频渲染错误。
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	视频后处理错误。
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	获取点播文件信息失败，建议检查 AppId、FileId 或 Psign 填写是否正确。

播放状态反馈（onNetStatus）

状态反馈每0.5秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前 SDK 内部的一些具体情况，以便您能对当前视频播放状态等有所了解。

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时 CPU 使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 - 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度，单位：KBps。
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 bps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 bps
NET_STATUS_VIDEO_CACHE	缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器 IP

通过 onNetStatus 获取视频播放过程信息示例：



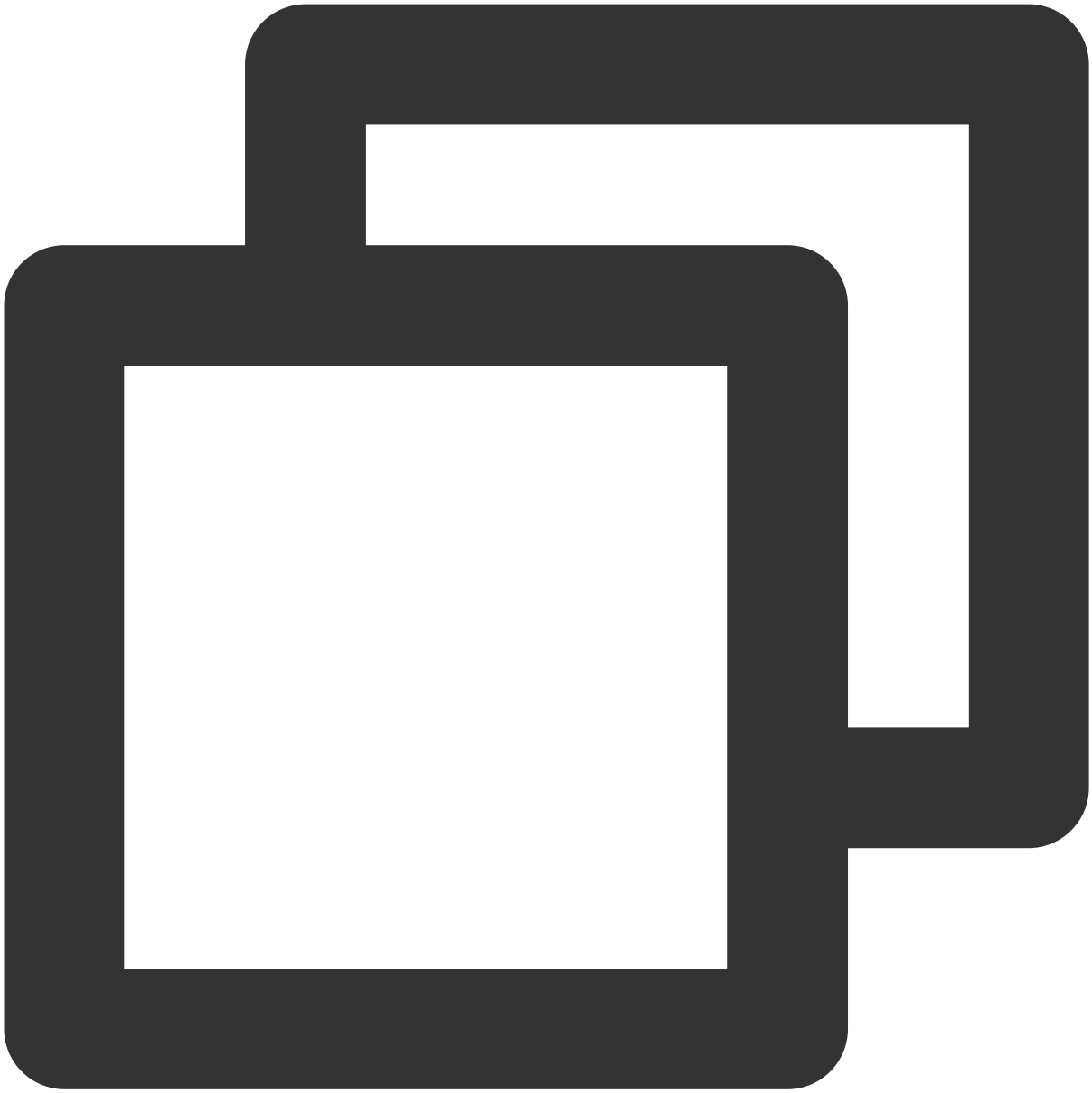
```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        //获取当前CPU使用率
        CharSequence cpuUsage = bundle.getCharSequence(TXLiveConstants.NET_STATUS_C
        //获取视频宽度
        int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
```

```
            //获取视频高度
            int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
            //获取实时速率,  单位：kbps
            int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
            //获取当前流媒体的视频帧率
            int fps = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_FPS);
            //获取当前流媒体的视频码率，单位 bps
            int videoBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_BITRATE);
            //获取当前流媒体的音频码率，单位 bps
            int audioBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_AUDIO_BITRATE);
            //获取缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
            int jitterbuffer = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_CACHE);
            //获取连接的服务器的IP地址
            String ip = bundle.getString(TXLiveConstants.NET_STATUS_SERVER_IP);
        }
    });
```

其它功能使用

HLS 直播视频源播放

播放器高级版本支持播放 HLS 直播视频源，从 11.8 版本开始支持带 HLS EVENT 直播视频源。用法如下：



```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_LIVE); // 指定HLS直播媒资类型
mVodPlayer.setConfig(config);
mVodPlayer.startVodPlay(${YOUR_HSL_LIVE_URL});
```

场景化功能

1、基于 SDK 的 Demo 组件

基于播放器 SDK，腾讯云研发了一款 播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美市面上各种流行视频 App 的播放软件。

2、开源 Github

基于播放器 SDK，腾讯云研发了沉浸式视频播放器组件、视频 Feed 流、多播放器复用组件等，而且随着版本发布，我们会提供跟多的基于用户场景的组件。您可以通过 Player_Android 下载体验。

Flutter 端集成
集成指引

最近更新时间：2024-04-26 11:09:31

环境准备

Flutter 3.0 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 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。

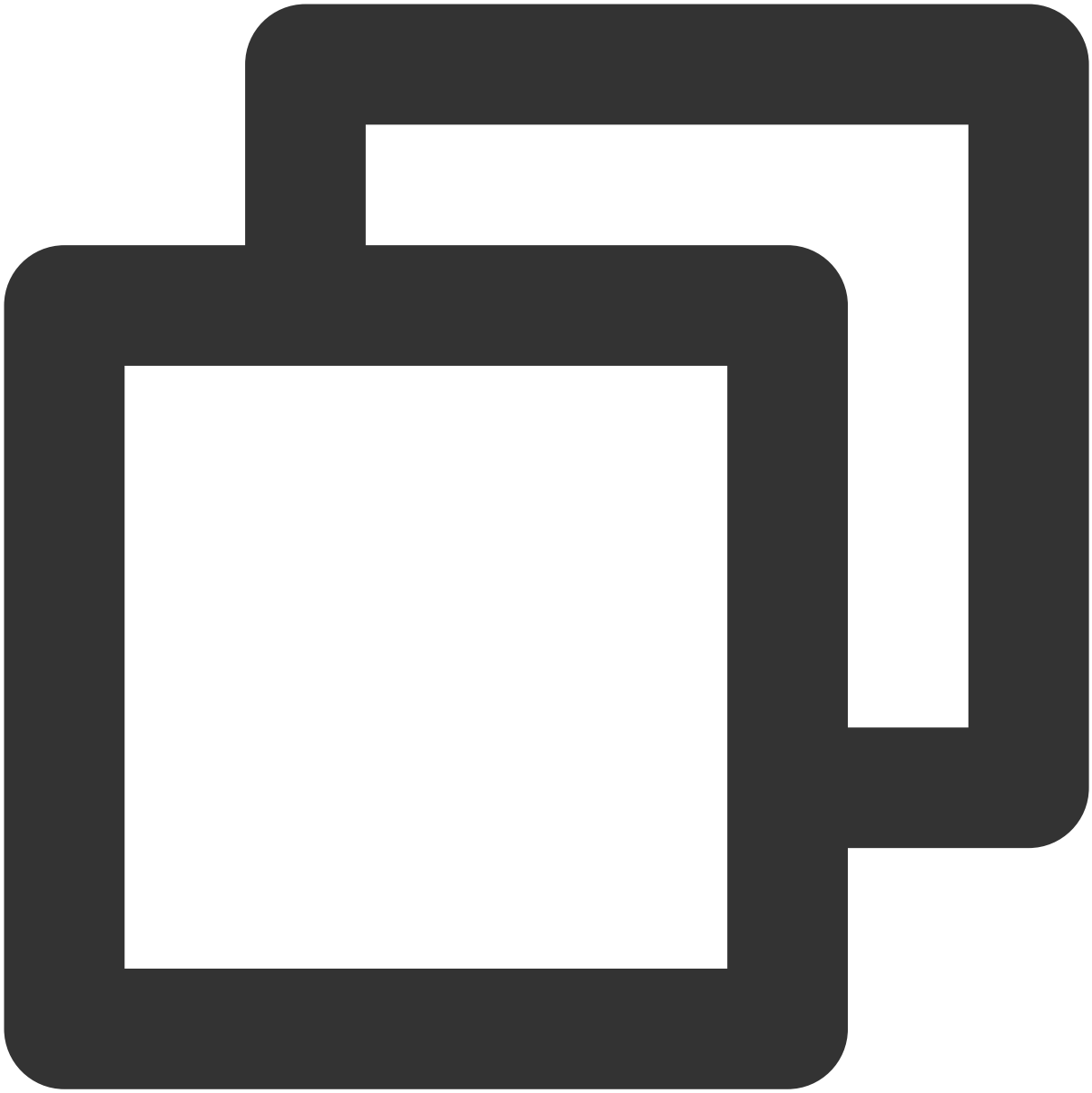
SDK 下载

腾讯云视立方 Flutter 播放器项目的地址是 Player Flutter 。
注意：
运行该 demo 的时候，需要在 demo_config 中设置自己的播放器 license，并在 Android 和 iOS 配置中，将包名和 bundleId 修改为自己签名的包名和 bundleId。

快速集成

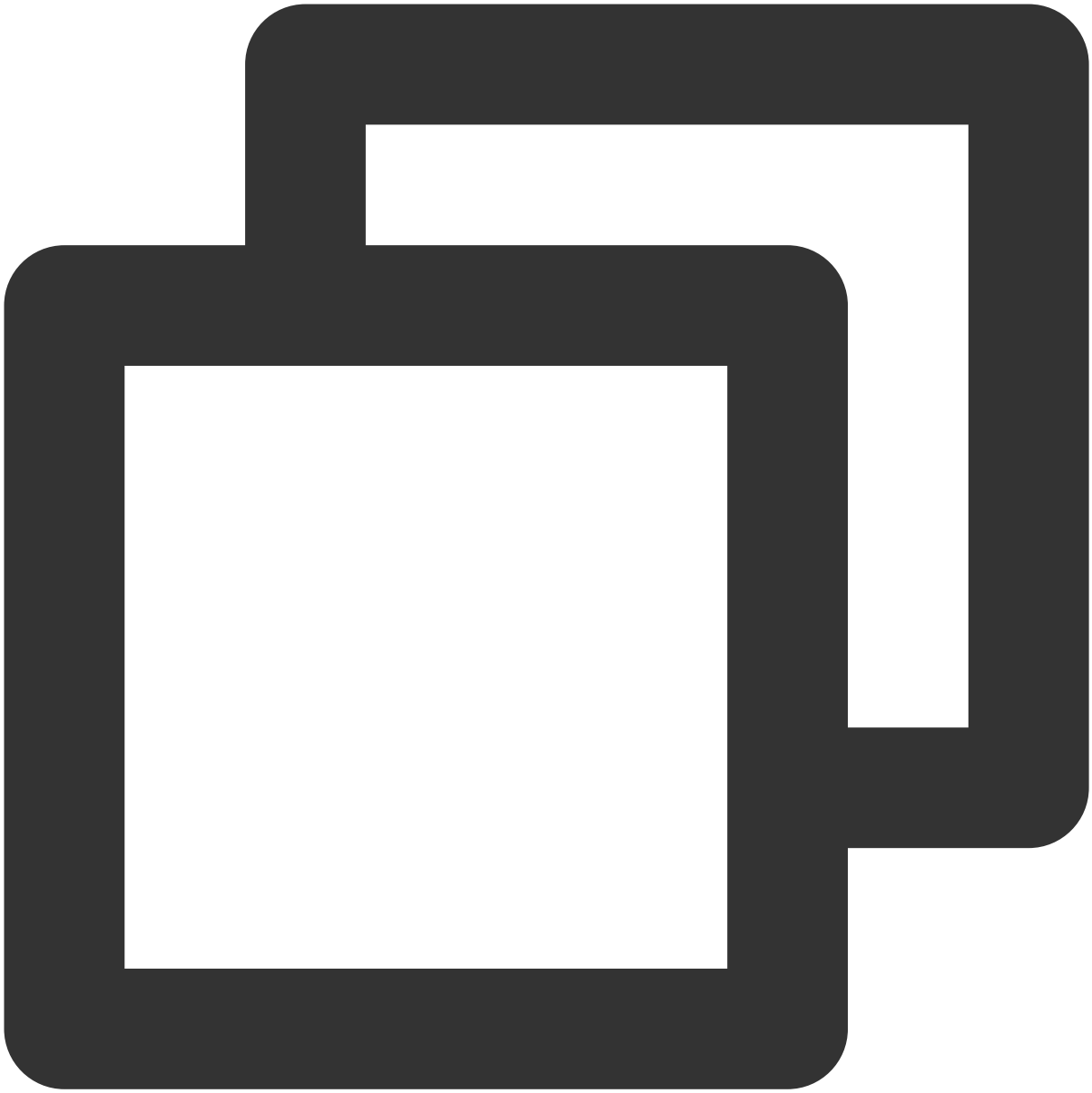
在项目的 pubspec.yaml 中添加依赖

支持基于 LiteAVSDK Player 或 Professional 版本集成，你可以根据项目需要进行集成。
1. 集成 LiteAVSDK_Player_Premium (播放器高级版) 最新版本，则 `pubspec.yaml` 中增加配置：



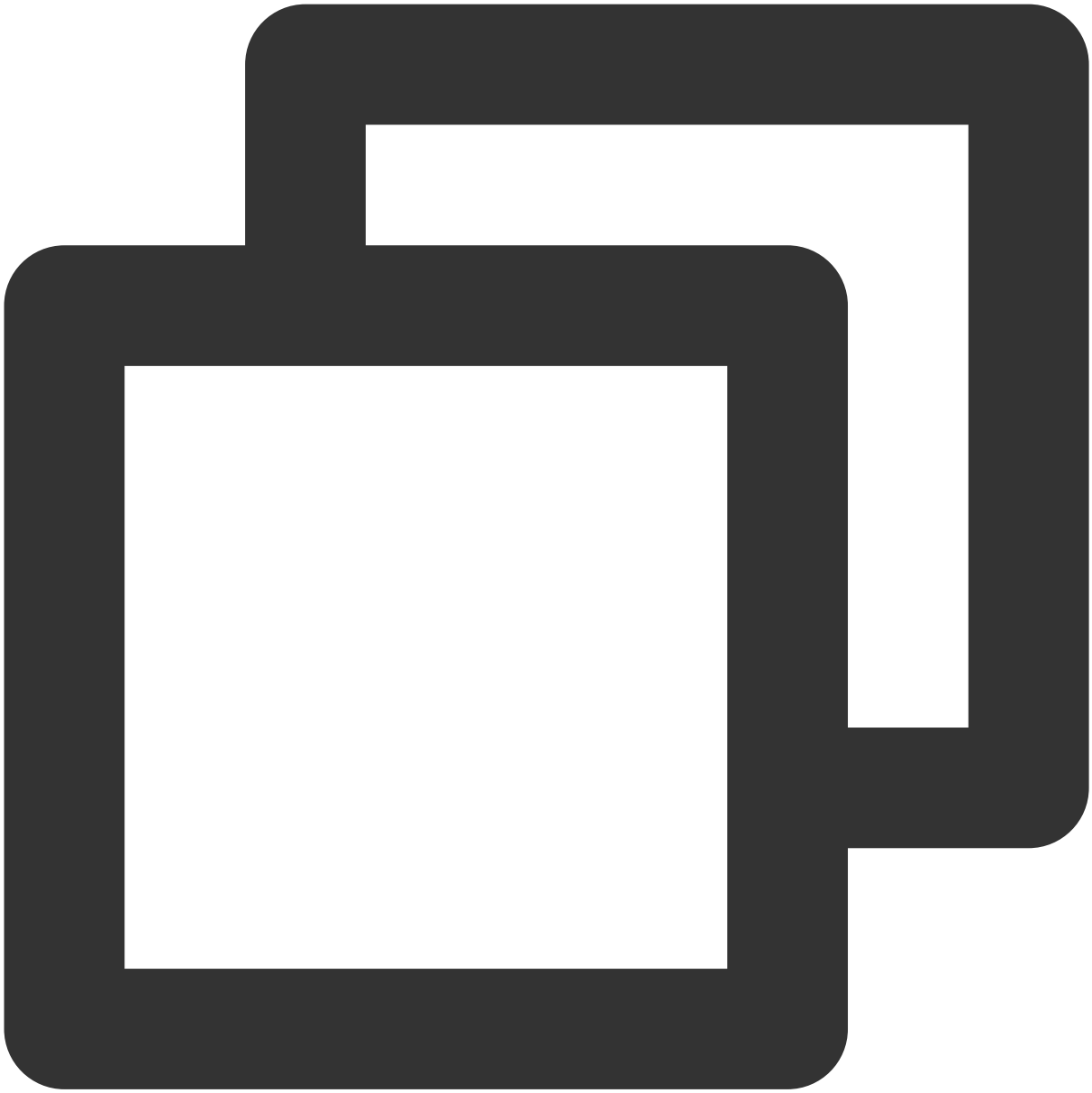
```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: Player_Premium
```

集成 LiteAVSDK_Player 版本最新版本，在 `pubspec.yaml` 中增加配置：



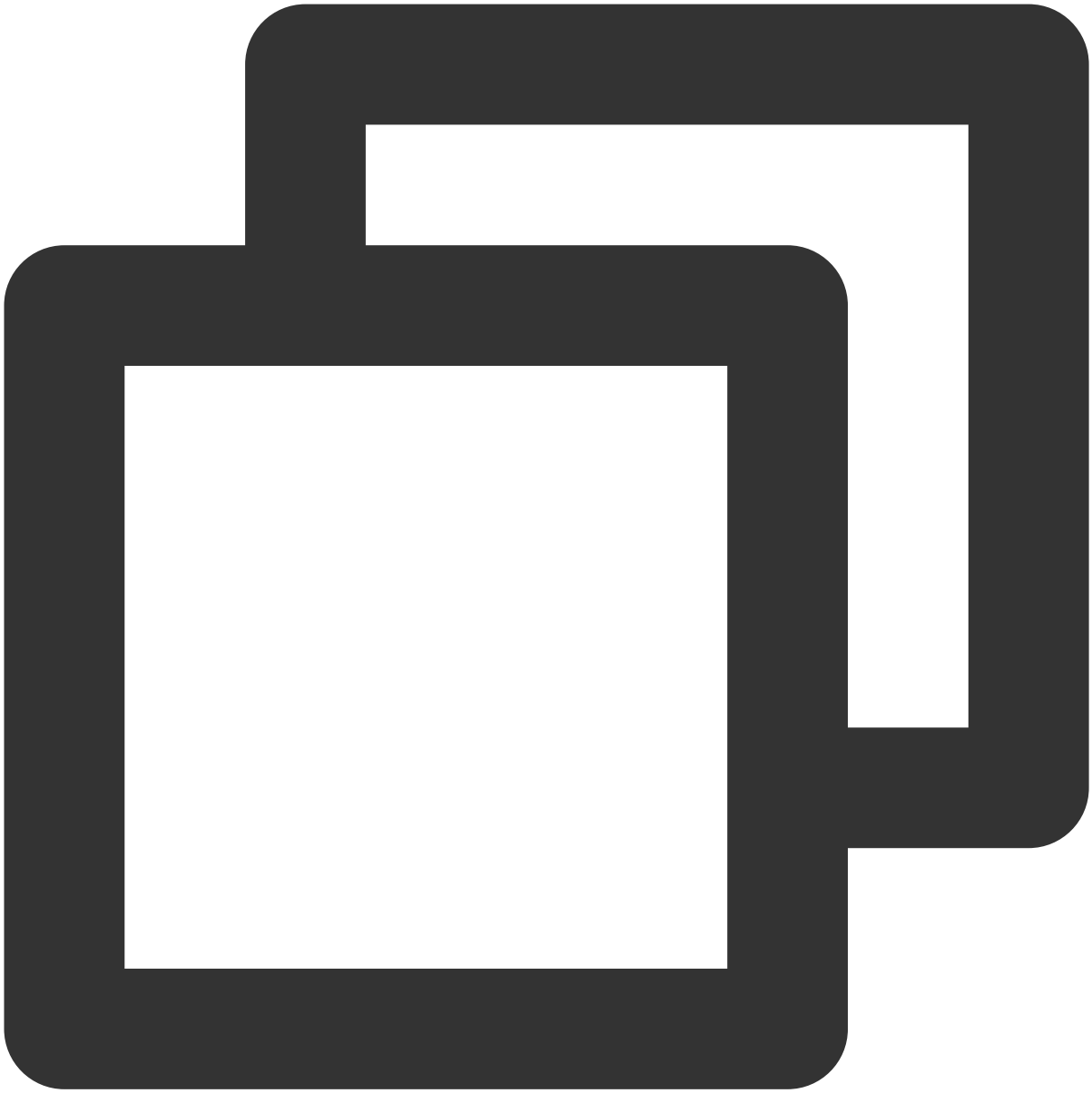
```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
```

集成 LiteAVSDK_Professional 最新版本，则 `pubspec.yaml` 中配置改为：



```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: Professional
```

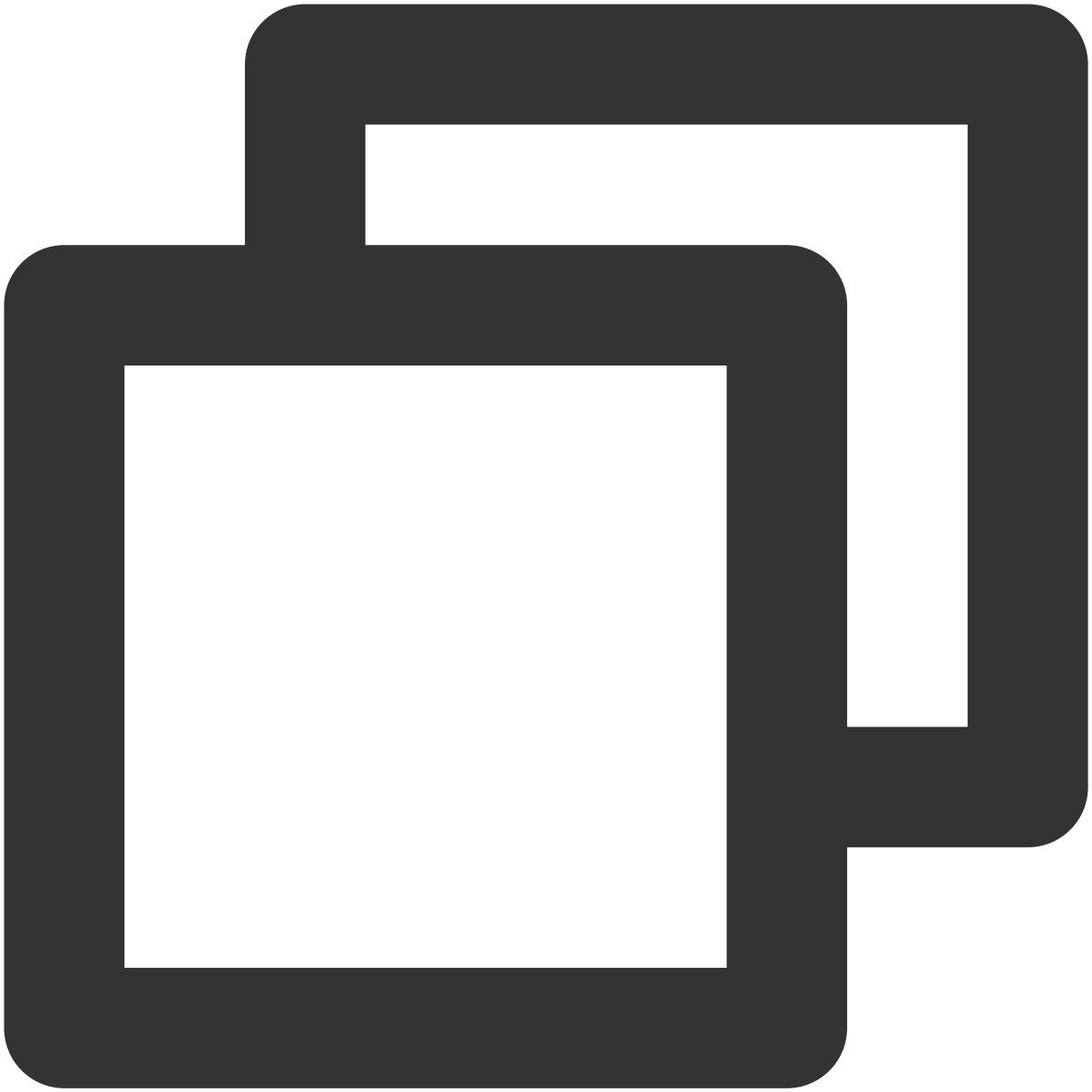
如果需要集成指定播放器版本的 SDK，可以指定通过 ref 依赖的 tag 来指定到对应版本，如下所示：



```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: release_player_v1.0.6

# release_player_v1.0.6 表示将集成Android端TXLiteAVSDK_Player_10.6.0.11182 版本，iOS 端
```

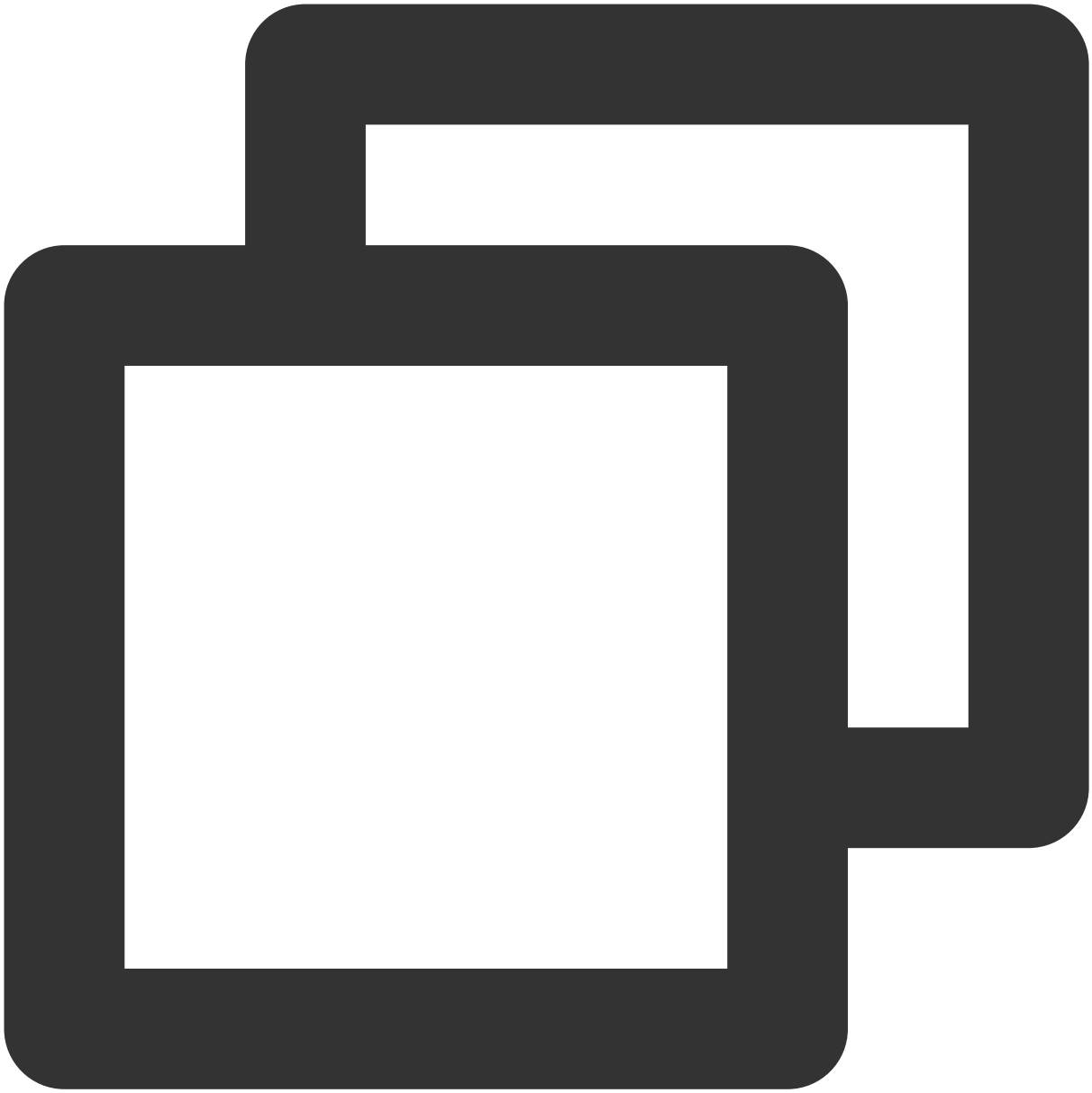
更多归档的 tag 请参考 release 列表 。

2. 集成之后，可以通过代码编辑器自带的 UI 界面来获取 Flutter 依赖，也可以直接使用如下命令获取：



```
flutter pub get
```

3. 使用过程中，可以通过以下命令来更新现有 Flutter 依赖：

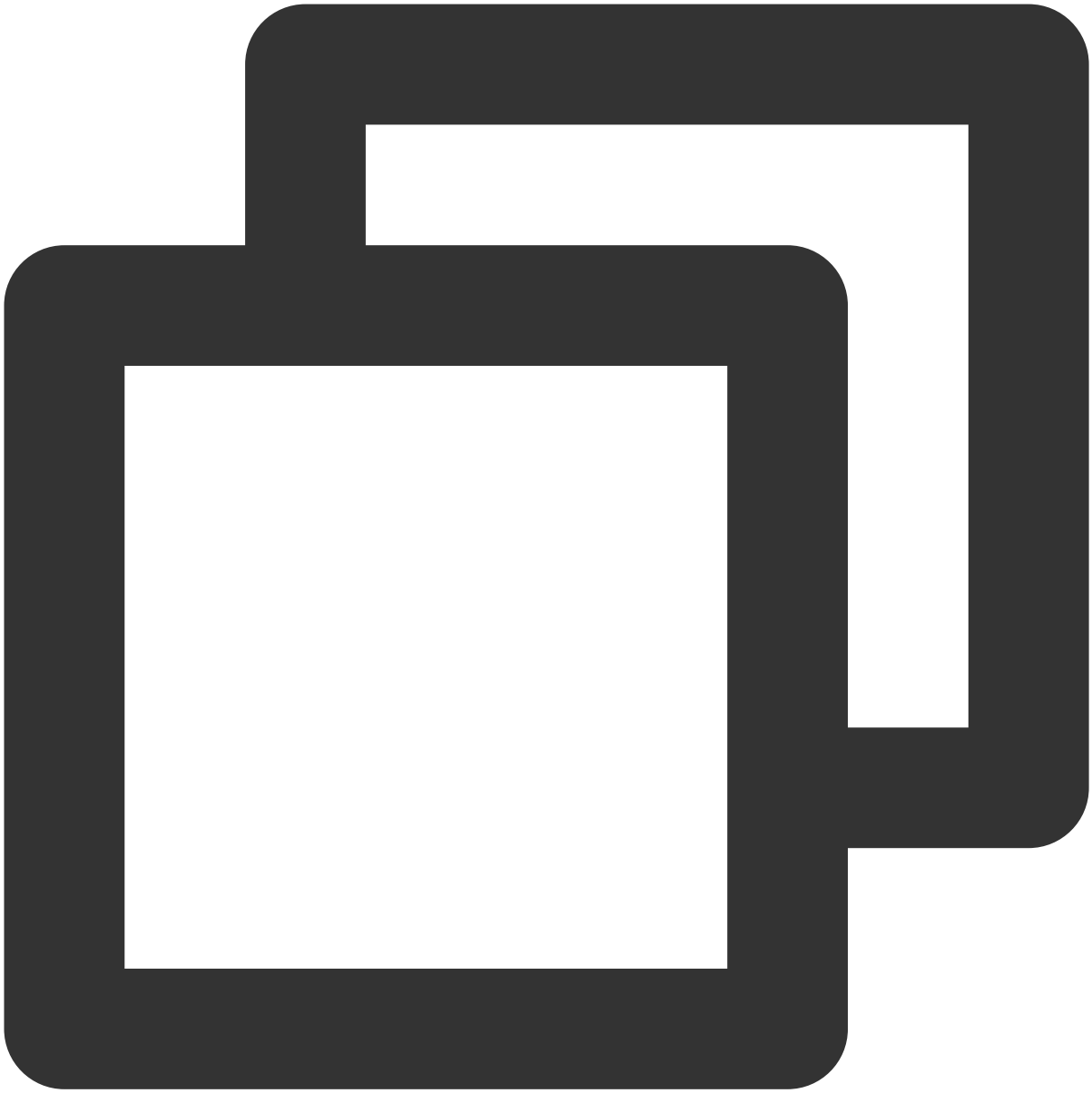


```
flutter pub upgrade
```

添加原生配置

Android 端配置

1. 在 Android 的 `AndroidManifest.xml` 中增加如下配置：

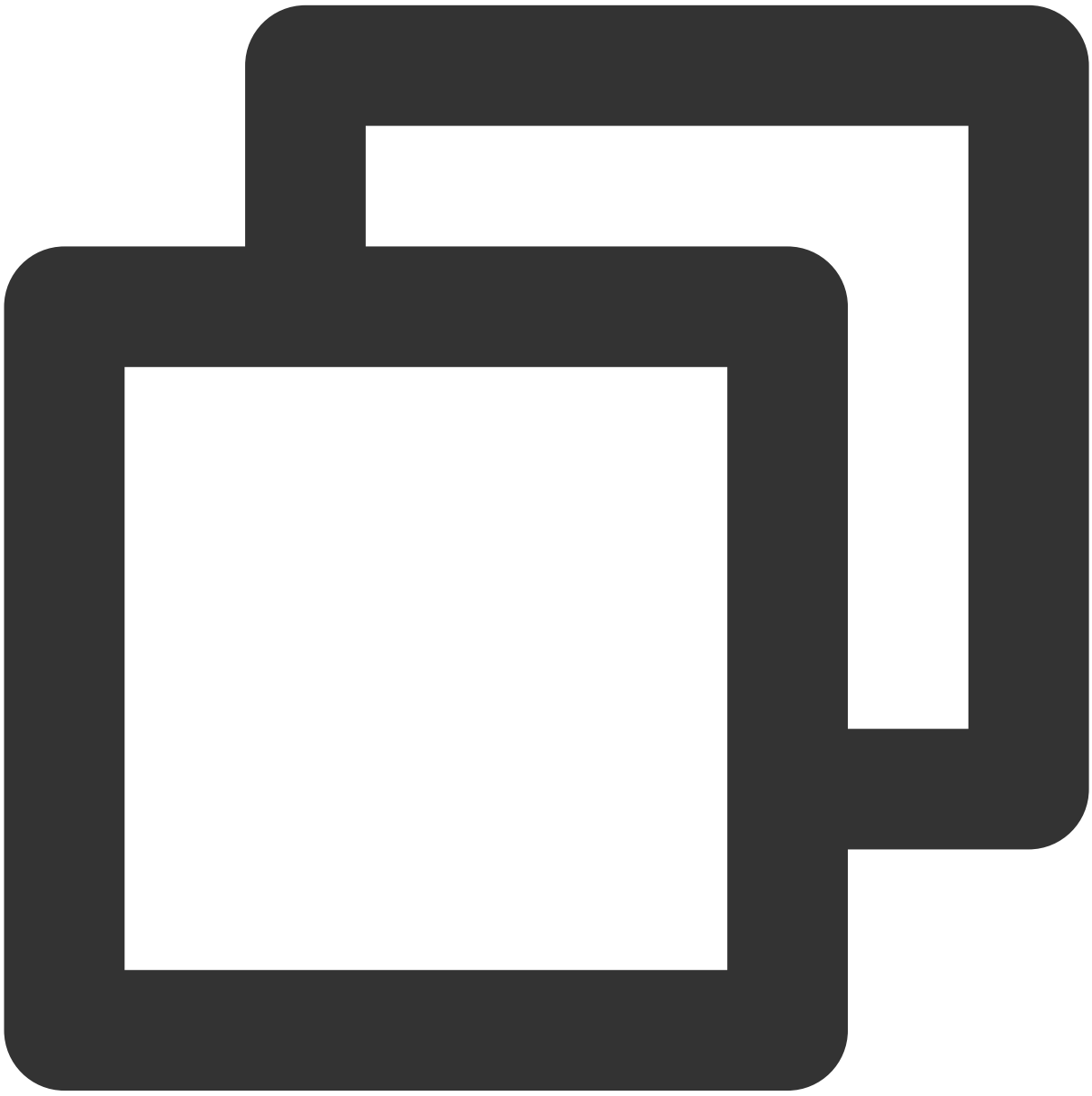


```
<!--网络权限-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

网络安全配置允许 App 发送 http 请求

出于安全考虑，从 Android P 开始，Google 要求 App 的请求都使用加密链接。播放器 SDK 会启动一个 localsever 代理 http 请求，如果您的应用 **targetSdkVersion 大于或等于28**，可以通过 网络安全配置 来开启允许向127.0.0.1发送 http 请求。 否则播放时将出现 "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" 错误， 导致无法播放视频。配置步骤如下：

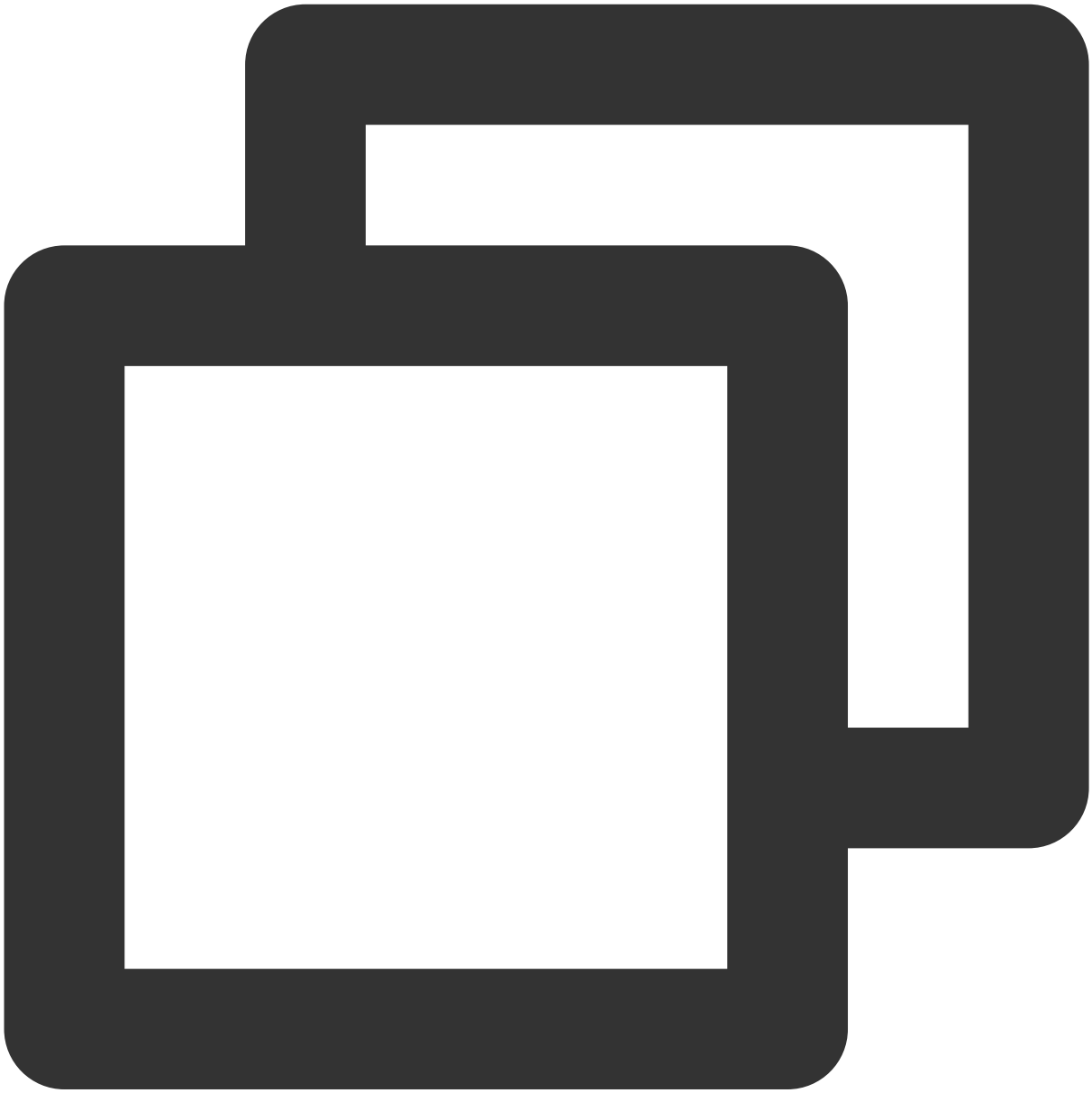
1.1 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置。



```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
```

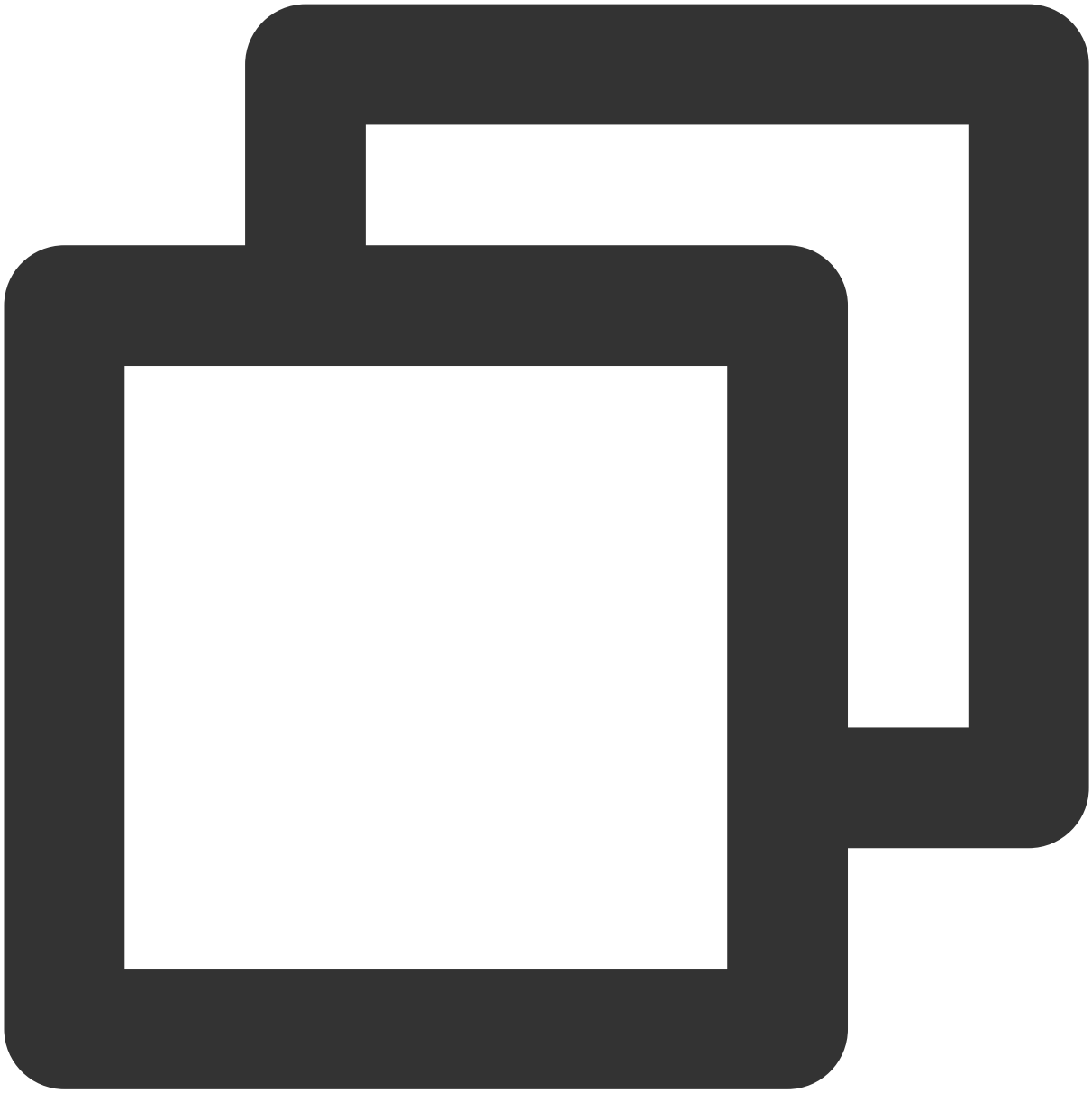
```
    </domain-config>
</network-security-config>
```

1.2 在 AndroidManifest.xml 文件下的 application 标签增加以下属性。



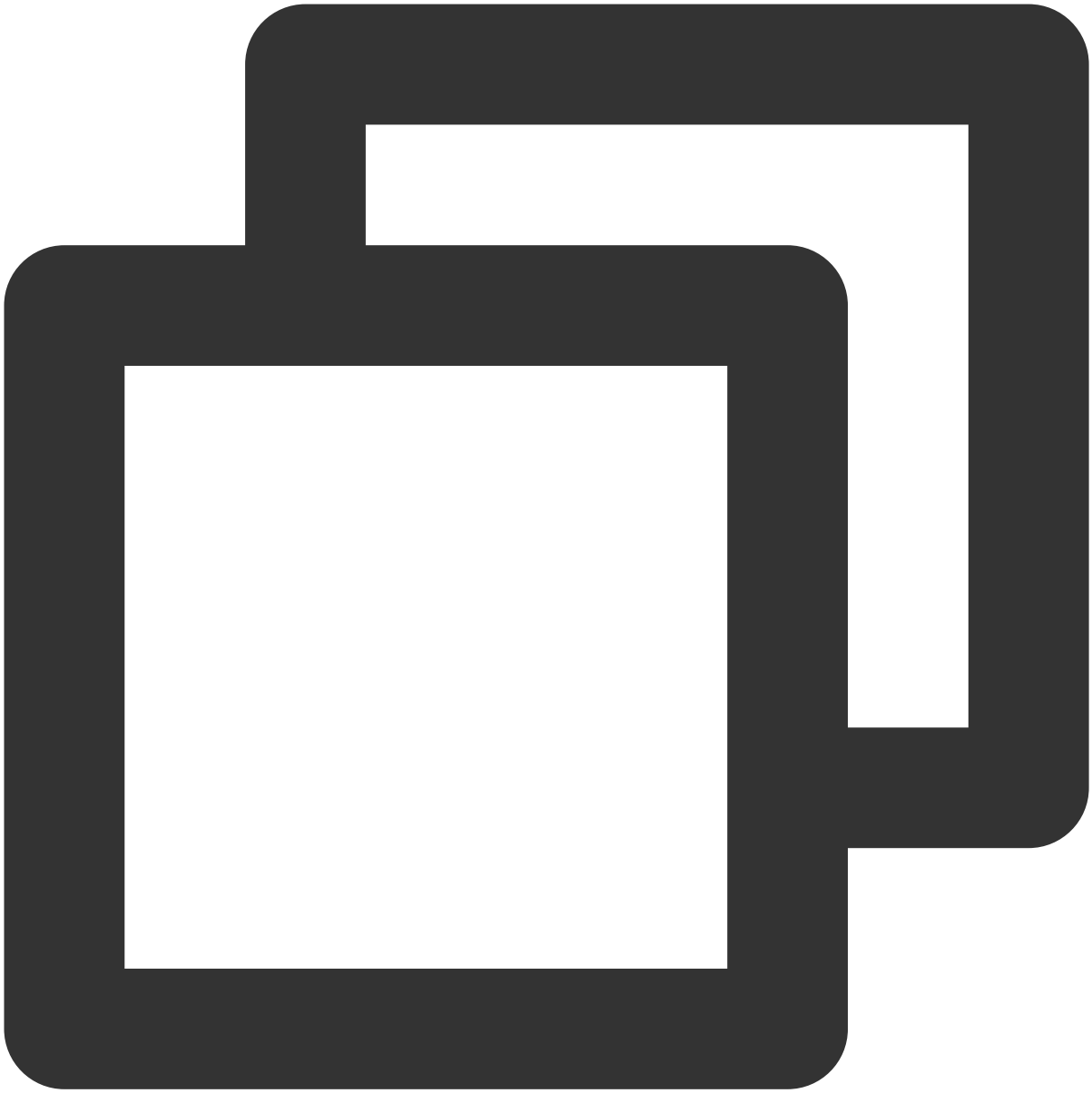
```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

2. 确保 Android 目录下的 `build.gradle` 使用了 mavenCenter，能够成功下载到依赖。



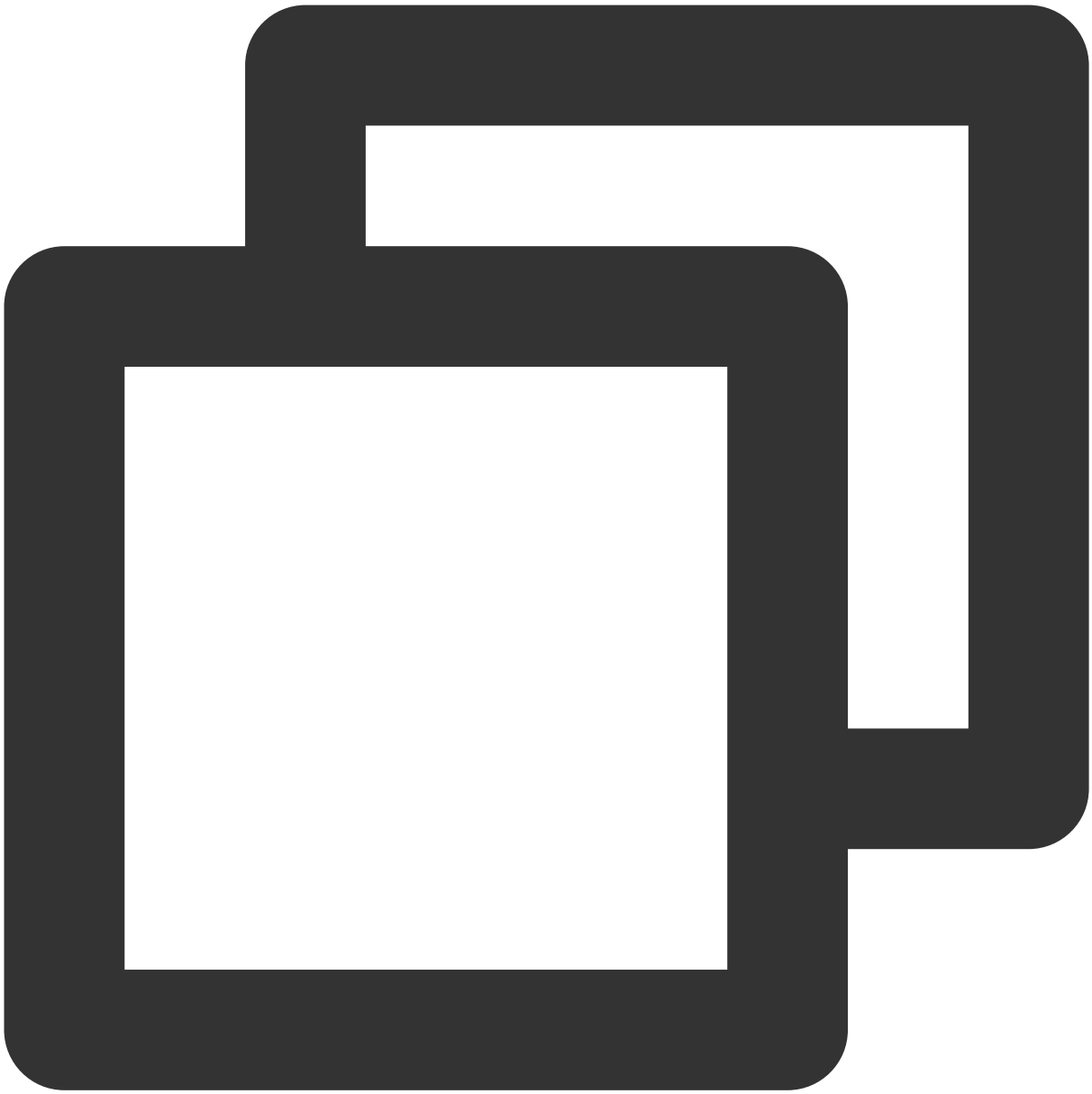
```
repositories {
  mavenCentral()
}
```

3. 配置 Android 最小 SDK 版本，由于 flutter 默认配置的 Android 最小版本过低，需要手动更改为至少19，如果需要使用画中画能力，compileSdkVersion 和targetSdkVersion 则需要修改为至少31。



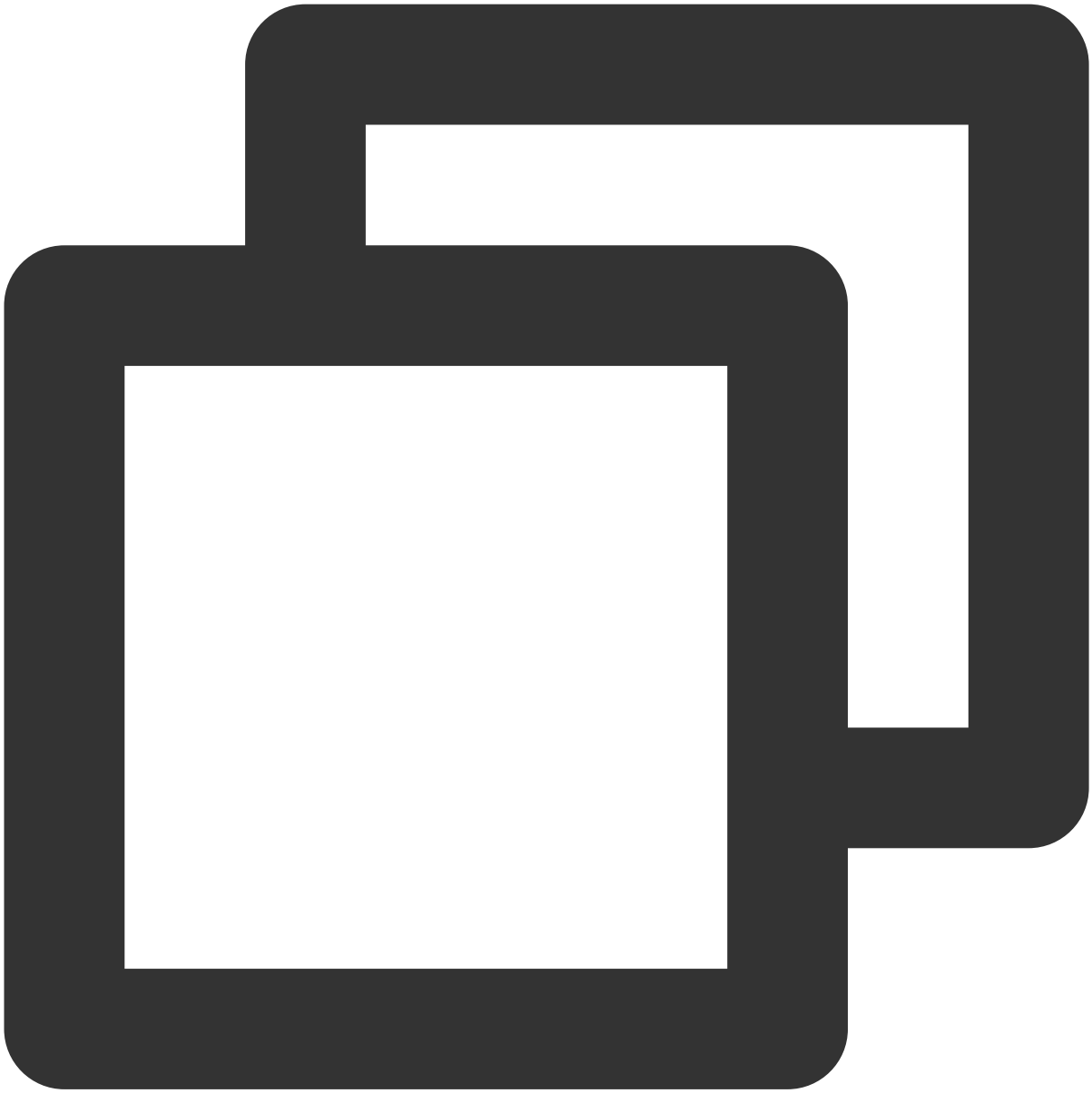
```
compileSdkVersion 31
defaultConfig {
    applicationId "com.tencent.liteav.demo"
    minSdkVersion 19
    targetSdkVersion 31
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

4. `AndroidManifest.xml` 根节点 manifest 标签内增加如下配置 `xmlns:tools="http://schemas.android.com/tools";` ，示例如下：



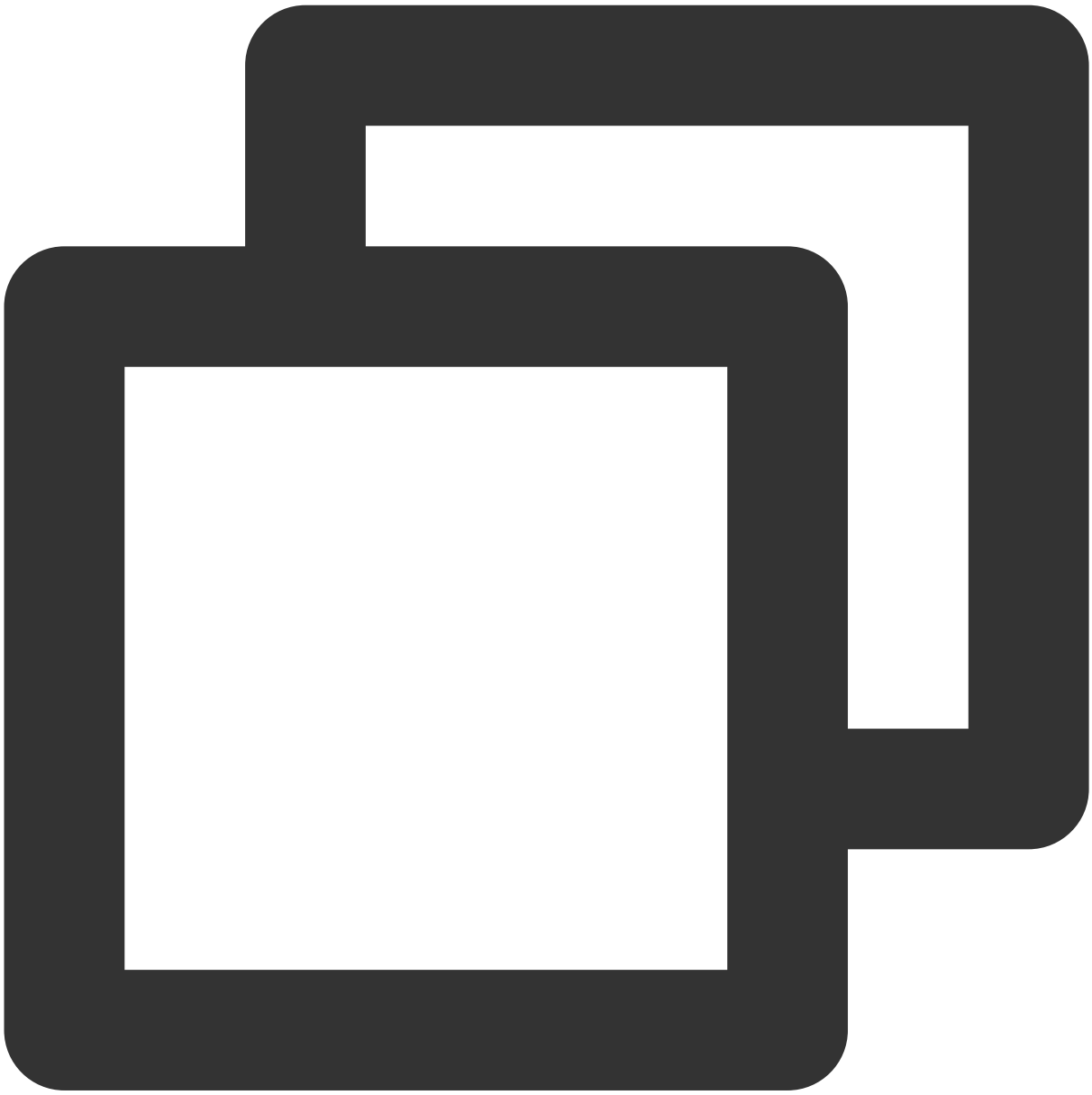
```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.player">
 <!-- your config...... -->
</manifest>
```

application 节点下增加 `tools:replace="android:label"`，示例如下：



```
<application
    android:label="super_player_example"
    android:icon="@mipmap/ic_launcher"
    android:requestLegacyExternalStorage="true"
    tools:replace="android:label">
<!-- your config..... -->
</application>
```

5. 如果需要更新原生 SDK 依赖版本，可手动删除 Android 目录下的 `build` 文件夹，也可以使用如下命令强制刷新。



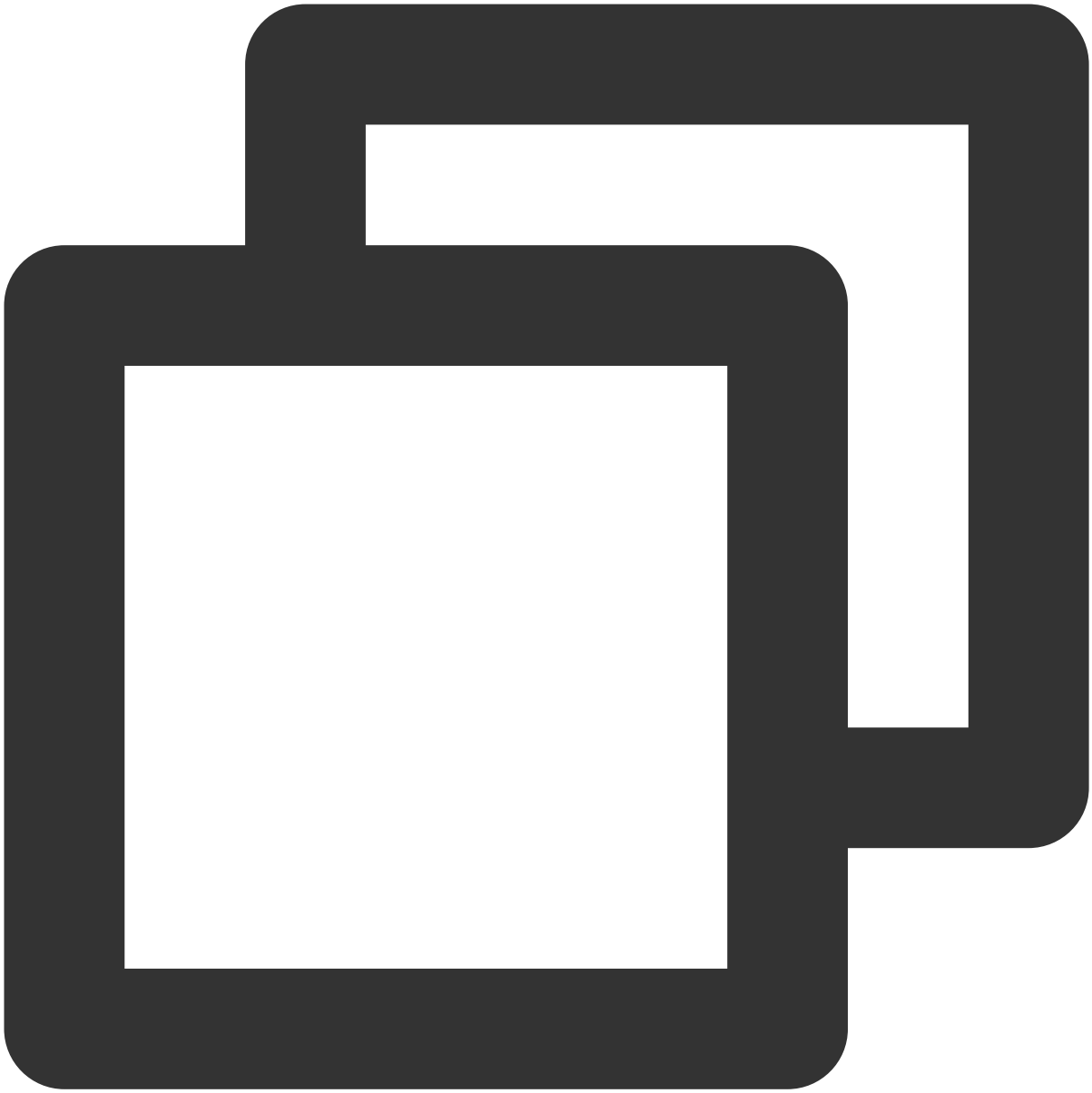
```
./gradlew build
```

iOS 端配置

注意：

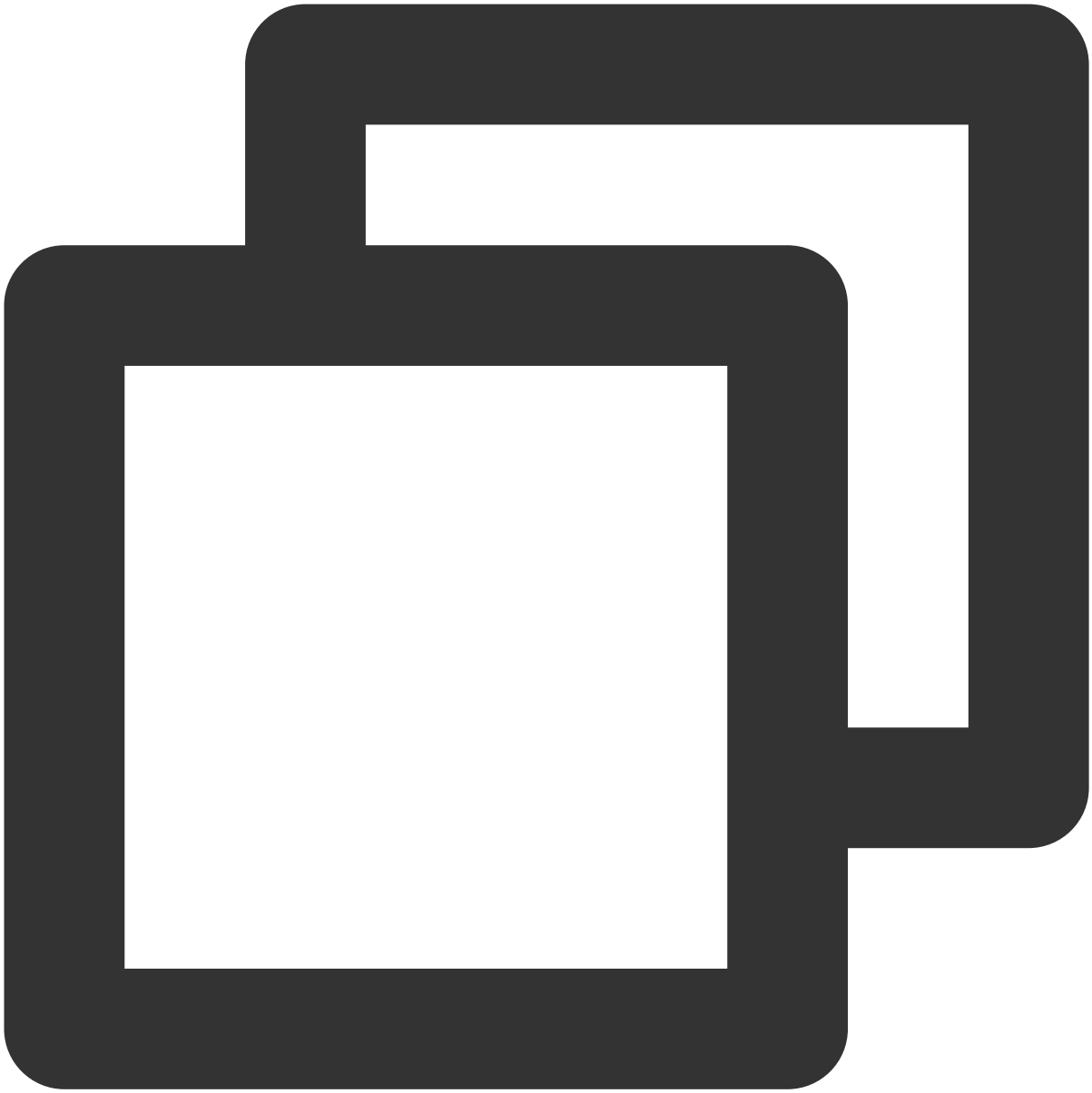
iOS 端目前暂不支持模拟器运行调试，建议在真机下进行开发调试。

1. 在 iOS 的 `Info.plist` 中增加如下配置：



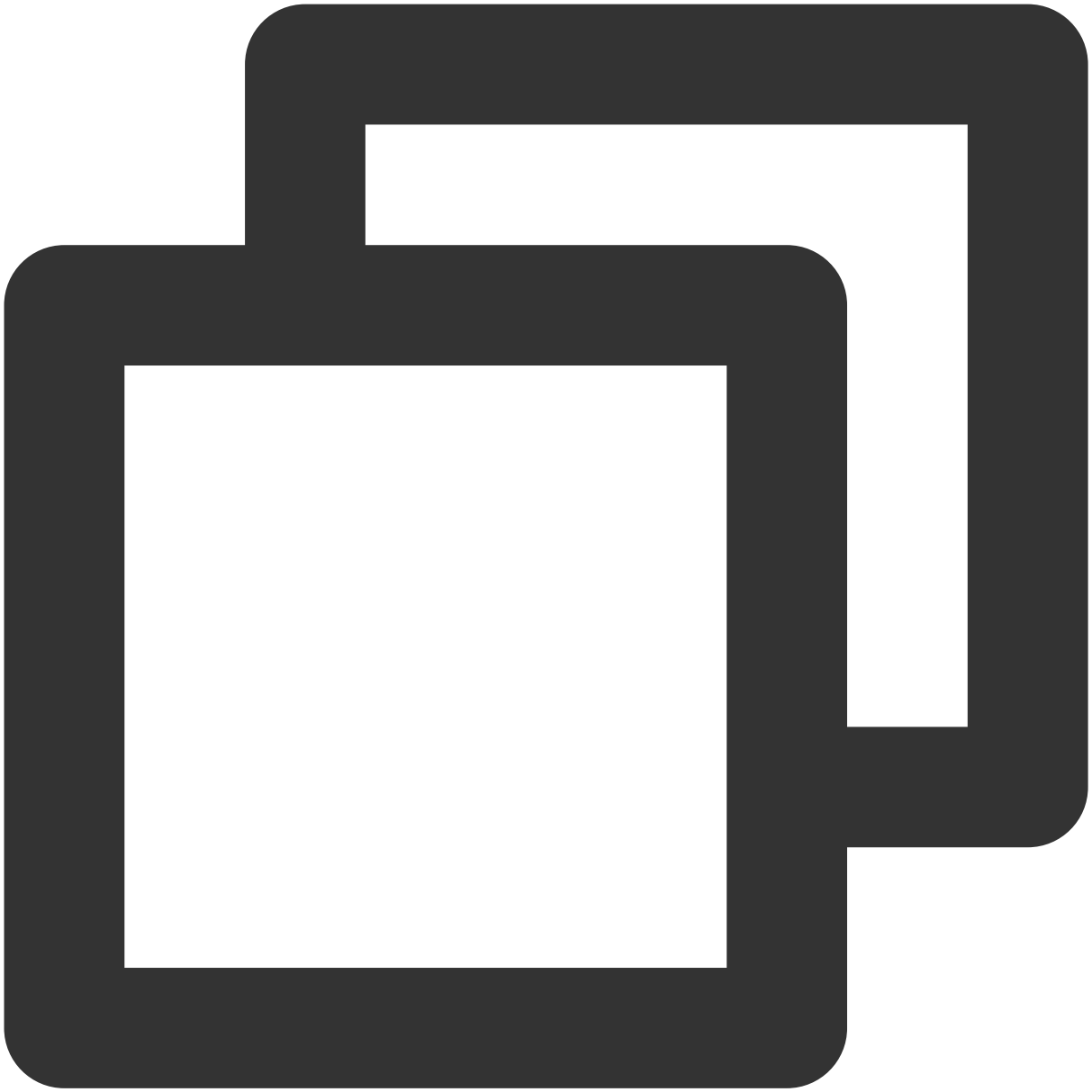
```
<key>NSAppTransportSecurity</key>
<dict>
 <key>NSAllowsArbitraryLoads</key>
 <true/>
</dict>
```

2. iOS 原生采用 `pod` 方式进行依赖，编辑 `podfile` 文件，指定你的播放器 SDK 版本，集成的是 Player_Premium 版SDK。



```
pod 'TXLiteAVSDK_Player_Premium'          //Player_Premium版
#pod 'TXLiteAVSDK_Player'                 //Player版
```

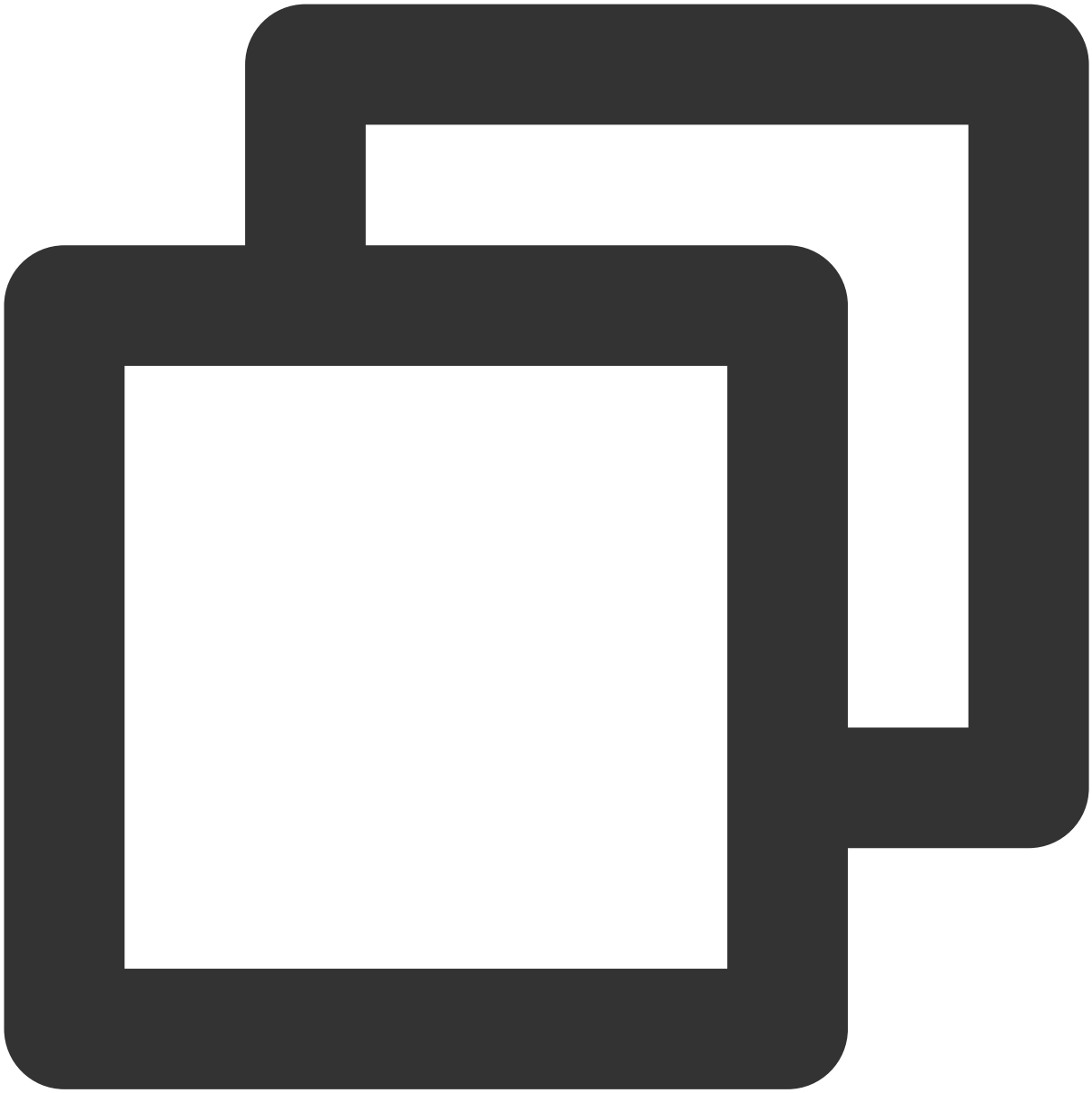
Professional 版 SDK 集成：



```
pod 'TXLiteAVSDK_Professional'  //Professional版
```

如果不指定版本，默认会安装最新的 `TXLiteAVSDK_Player` 最新版本。

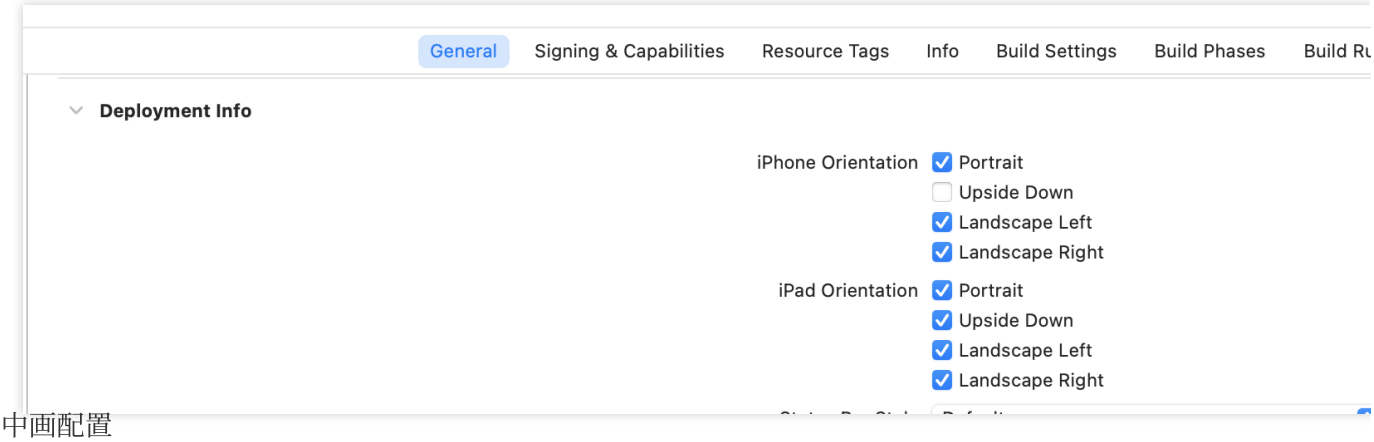
3. 部分情况下（如：发布了新版本），需要强制更新 iOS 播放器依赖，可以在 iOS 目录下使用如下命令进行更新：



```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

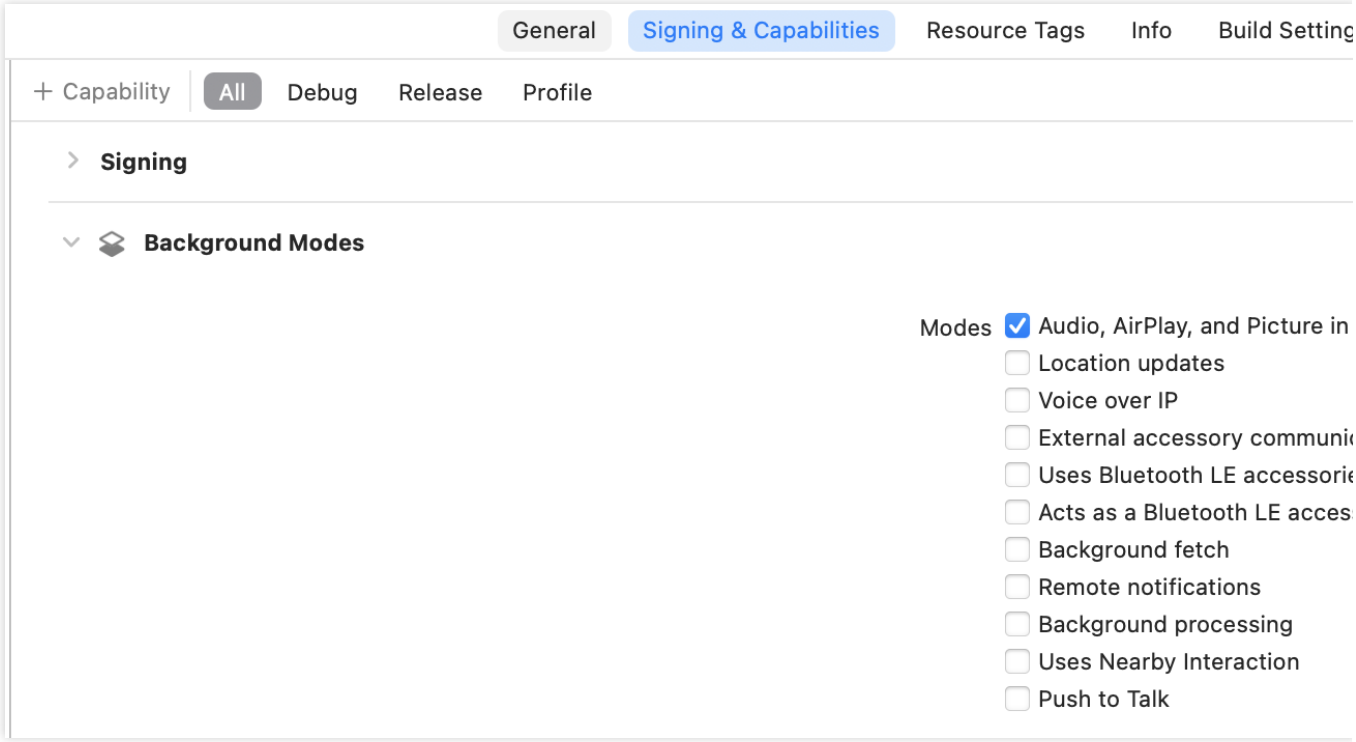
4. 横屏配置

如果需要应用支持横屏，需要在IOS项目配置 `General` 页面的 `Deployment Info` 标签下设置横竖屏的支持方向，可以全部进行勾选，如下图所示：



5. 画中画配置

如果需要项目支持画中画，需要在IOS项目配置 `Signing & Capabilities` 页面的 `Background Modes` 标签下，勾选 "Audio, AirPlay, and Picture in Picture" 来让项目支持画中画能力，如下图所示：



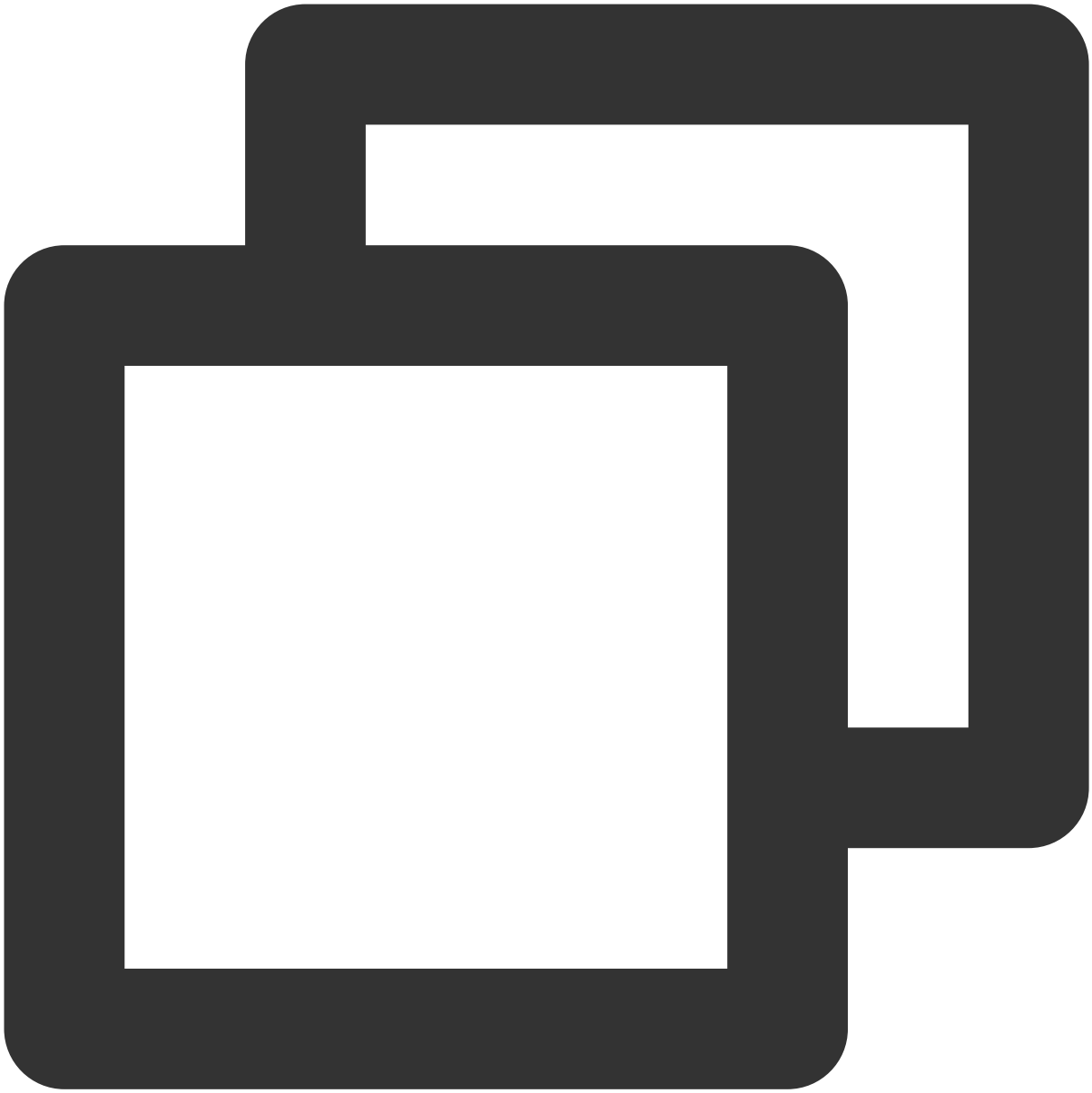
集成视频播放 License

若您已获得相关 License 授权，需在 云点播控制台 获取 License URL 和 License Key。

若您暂未获得 License 授权，需先参见 视频播放 License 获取相关授权。

集成播放器前，需要 注册腾讯云账户 ，注册成功后申请视频播放能力 License， 然后通过下面方式集成，建议在应用启动时进行。

如果没有集成 License，播放过程中可能会出现异常。



```
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
```

深度定制开发指引

腾讯云播放器 SDK Flutter 插件对原生播放器能力进行了封装， 如果您要进行深度定制开发，建议采用如下方法：

基于点播播放，接口类为 `TXVodPlayerController` 或直播播放，接口类为 `TXLivePlayerController` ，进行定制开发，项目中提供了定制开发 Demo，可参考 example 工程里的 `DemoTXVodPlayer` 和 `DemoTXLivePlayer` 。

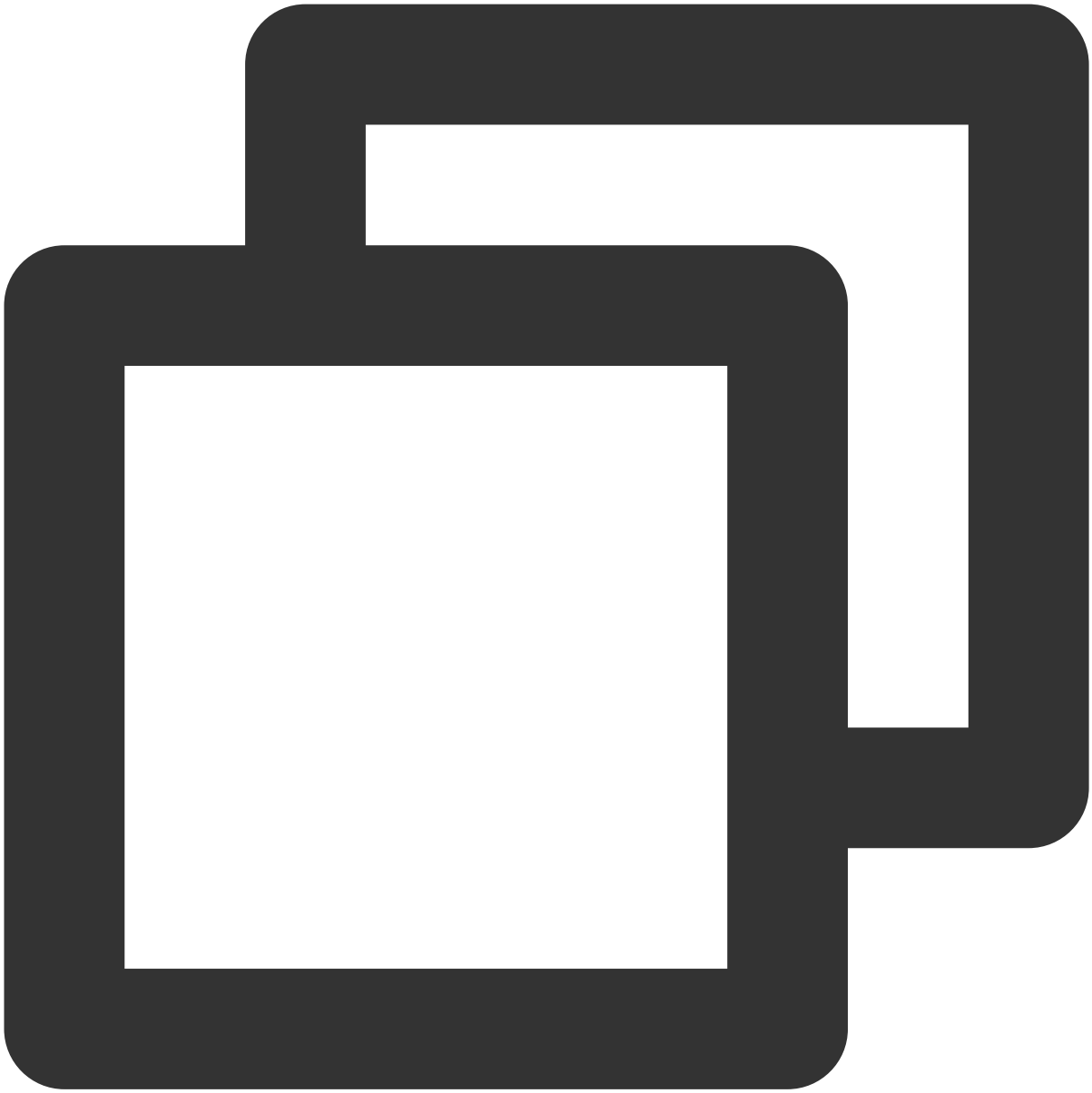
播放器组件 `SuperPlayerController` 对点播和直播进行了封装，同时提供了简单的 UI 交互， 由于此部分代码在 example 目录。如果您有对播放器组件定制化的需求，您可以进行如下操作：

把播放器组件相关的代码，代码目录： `Flutter/superplayer_widget` ， 复制到您的项目中，进行定制化开发。

常见问题

1. iOS 端运行，出现 `No visible @interface for 'TXLivePlayer' declares the selector 'startLivePlay:type:'` 等类似找不到接口错误。

可以使用如下命令，更新 iOS SDK：



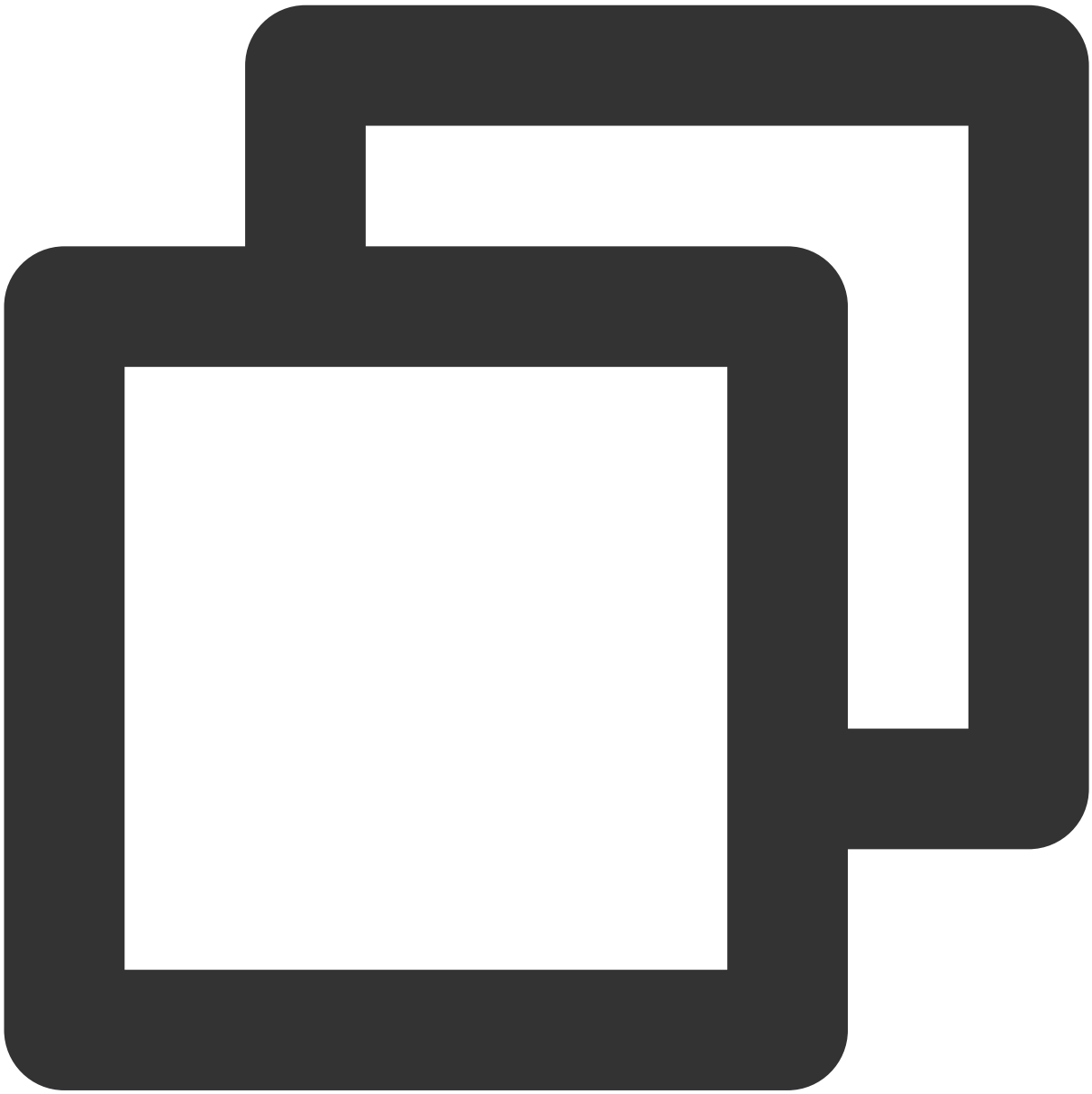
```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

2. 同时集成 tencent_trtc_cloud 和 Flutter 播放器出现 SDK 或 符号冲突。

常见异常日志：`java. lang.RuntimeException: Duplicate class com.tencent.liteav.TXLiteAVCode found in modules classes.jar`

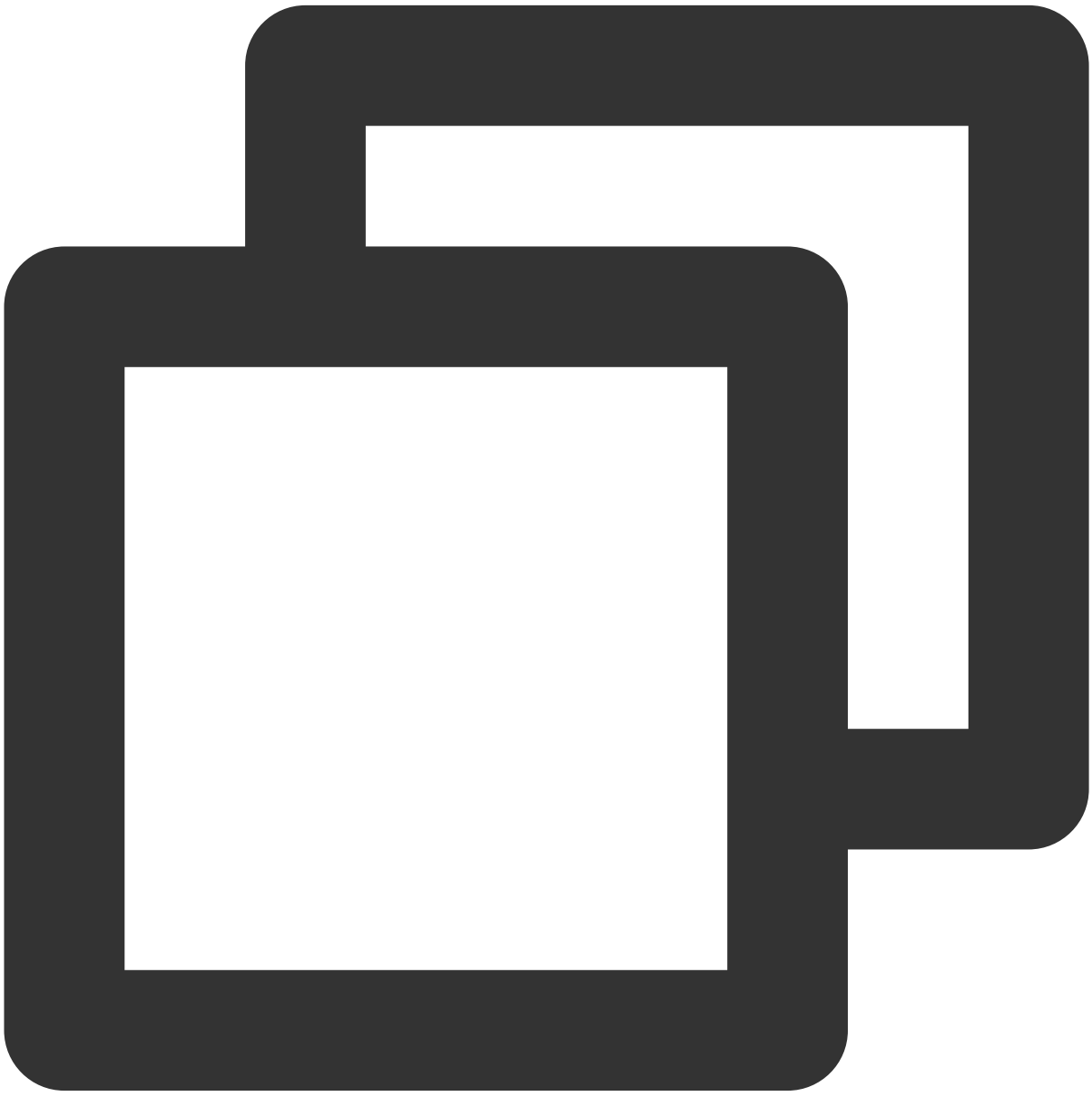
此时需要集成 flutter 播放器的 Professional 版本，让tencent_trtc_cloud 和 flutter 播放器共同依赖于同一个版的 LiteAVSDK_Professional。注意确保依赖的LiteAVSDK_Professional 的版本必须一样。

如：依赖 Android 端 TXLiteAVSDK_Professional_10.3.0.11196 和 iOS 端TXLiteAVSDK_Professional to 10.3.12231 版本，依赖声明如下：



```
tencent_trtc_cloud：2.3.8

super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: release_pro_v1.0.3.11196_12231
```

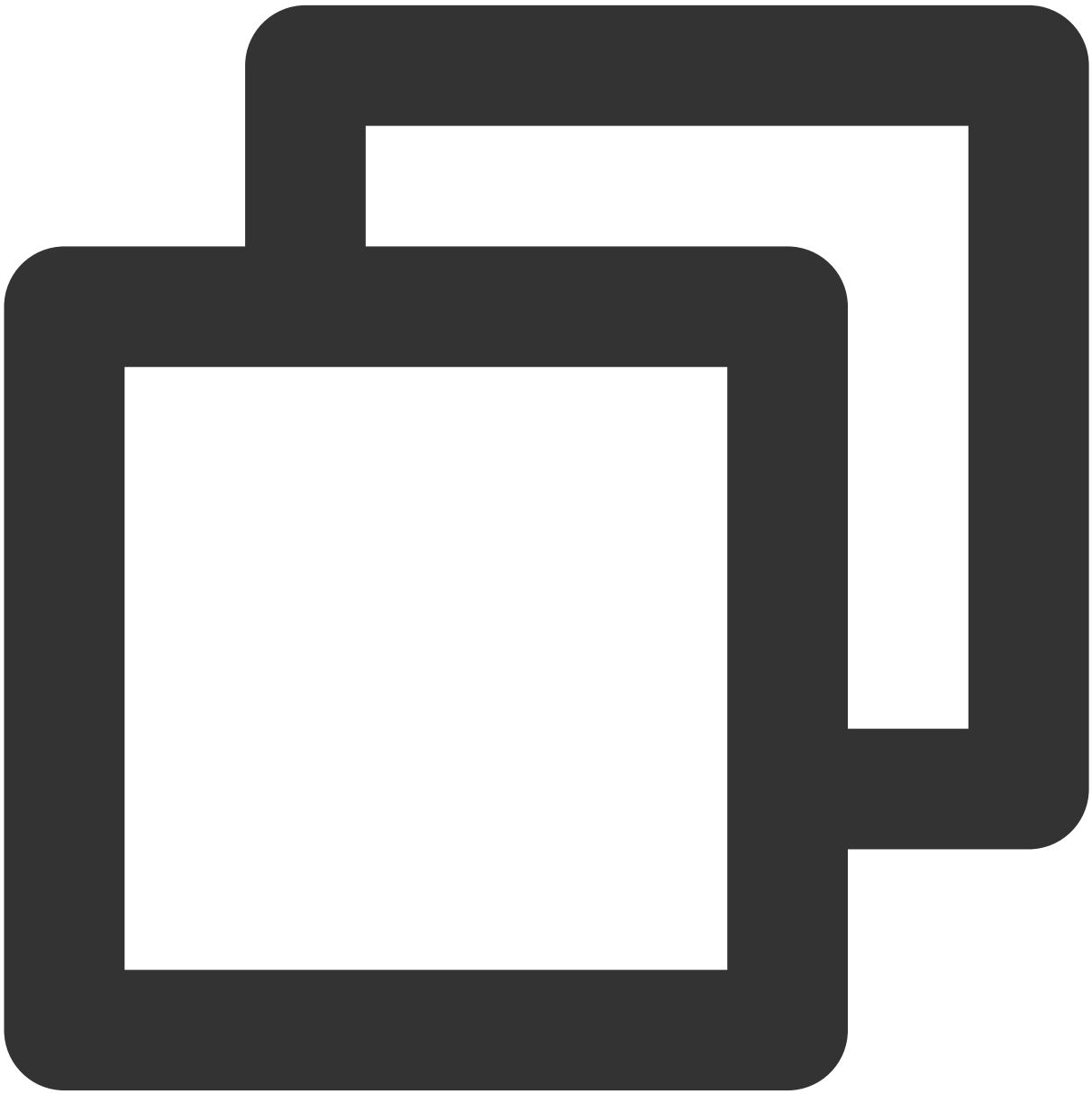
3. 需要同时使用多个播放器实例的时候，频繁切换播放视频，画面呈现模糊。
在每个播放器组件容器销毁的时候，调用播放器的 `dispose` 方法，将播放器释放。
4. 其余通用 Flutter 依赖问题：
执行 `flutter doctor` 命令检查运行环境，直到出现“No issues found!”。
执行 `flutter pub get` 确保所有依赖的组件都已更新成功。
5. 集成 superPlayer 之后，出现如下 manifest 错误：



```
Attribute application@label value=(super_player_example) from AndroidManifest.xml:9
is also present at [com.tencent.liteav:LiteAVSDK_Player:10.8.0.13065] AndroidManife
Suggestion: add 'tools:replace="android:label"' to <application> element at Android
```

解决方法：由于播放器 Android SDK 的 AndroidManifest 已经定义过 label，而 flutter 新建项目之后，在 Android 目录的 AndroidManifest 也会定义 label，此处建议根据错误提示，进入您的 Android 项目目录，在 AndroidManifest 的根节点 `manifest` 节点下增加 `xmlns:tools="http://schemas.android.com/tools"`，并在 application 节点下增加 `tools:replace="android:label"`。

6. 集成 superPlayer 之后，出现如下版本错误：



```
uses-sdk:minSdkVersion 16 cannot be smaller than version 19 declared in library [:s
```

解决方法：目前播放器 Android SDK 最小支持版本为 android 19，flutter 部分版本默认 Android 最小支持版本为 android 16。建议您将最小支持版本提高到 android 19。具体修改方法为：进入您的 Android 项目的主 module 下，

一般为 `app` 目录，将该目录下的 `build.gradle` 中的 `minSdkVersion` 修改为19。

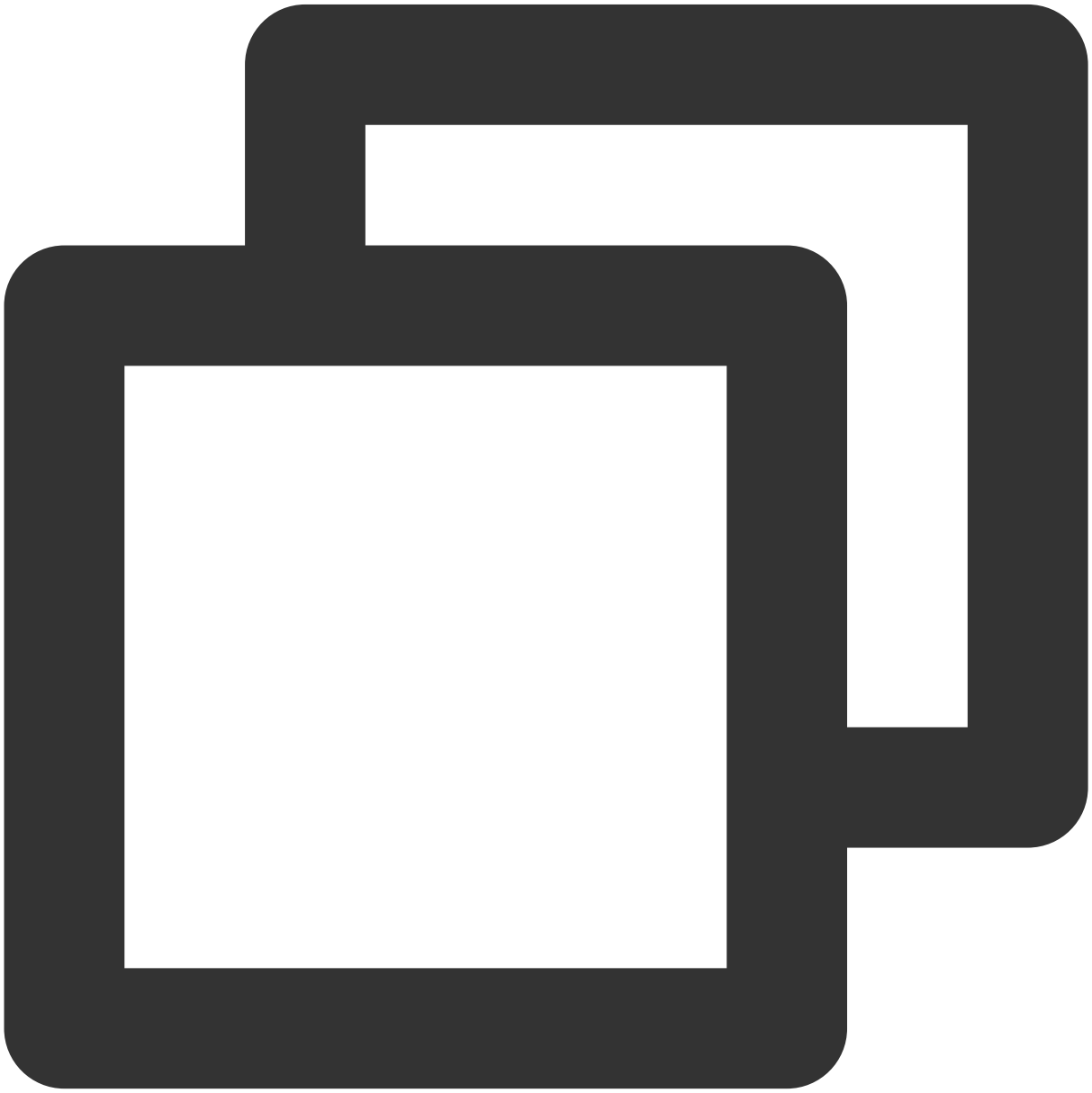
7. 如何提取播放器 SDK 的运行 Log ？

解决方法：播放器 SDK 默认把运行的 log 输出到本地文件，腾讯云技术支持 在帮忙定位问题时，需要这些运行 log 分析问题。Andorid 平台 log 保存在目录：`/sdcard/Android/data/packagename/files/log/tencent/liteav`，iOS 平台 log 保存在目录：`sandbox的Documents/log` 。

8. 如何减少控制台 log 输出？

解决方法：可以通过下面的接口设置 log 输出级别：

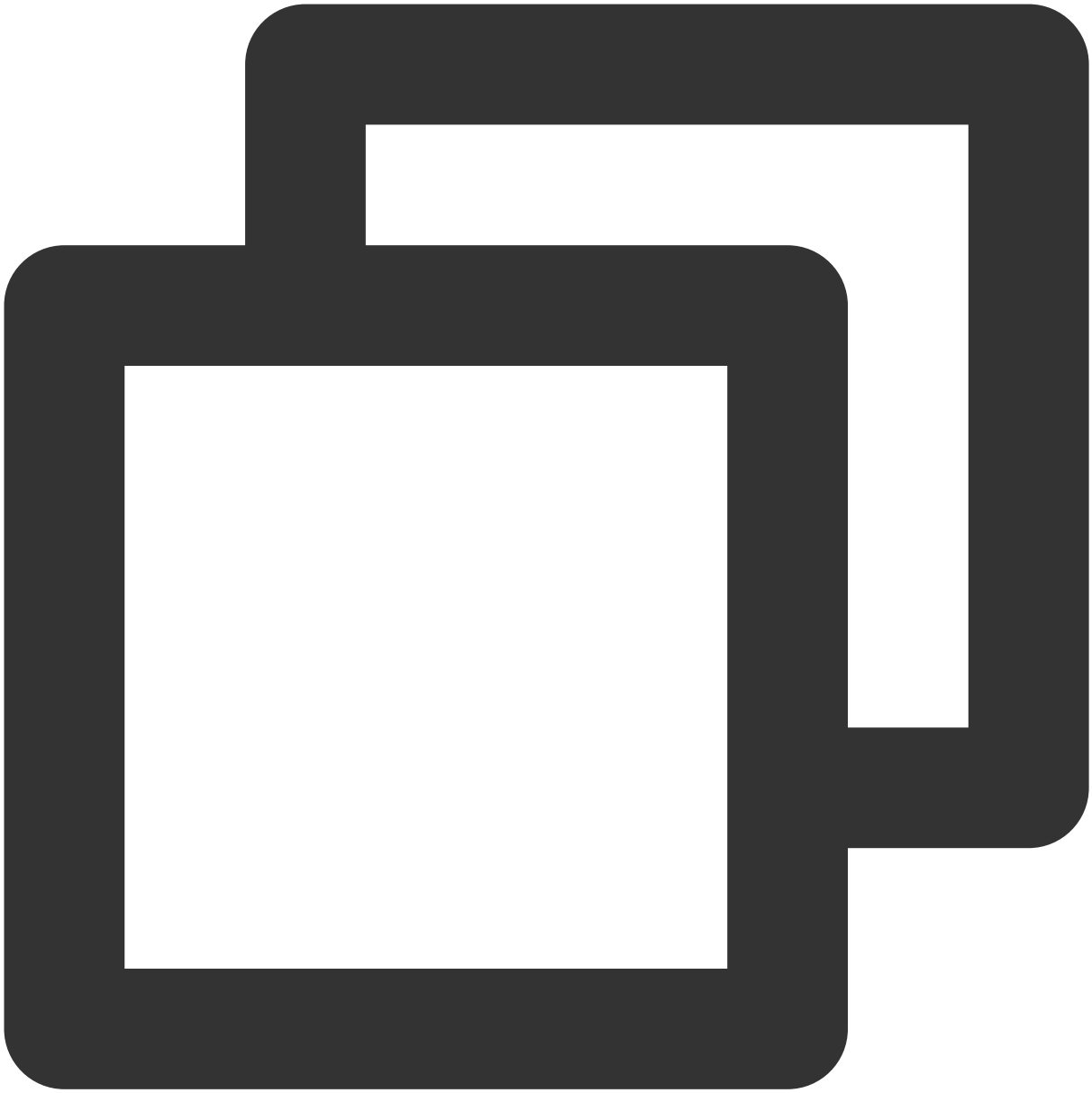
SuperPlayerPlugin.setLogLevel(TXLogLevel.LOG_LEVEL_NULL)，支持以下 log 级别：



```
class TXLogLevel {
  static const LOG_LEVEL_VERBOSE = 0; // 输出所有级别的log
  static const LOG_LEVEL_DEBUG = 1; // 输出 DEBUG,INFO,WARNING,ERROR 和 FATAL 级别的l
  static const LOG_LEVEL_INFO = 2; // 输出 INFO,WARNNING,ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_WARN = 3; // 输出WARNNING,ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_ERROR = 4; // 输出ERROR 和 FATAL 级别的log
  static const LOG_LEVEL_FATAL = 5; // 只输出FATAL 级别的log
  static const LOG_LEVEL_NULL = 6; // 不输出任何sdk log
}
```

9. 项目使用过程中，出现原生相关报错，例如 `错误：不兼容的类型`、`error: initializing 'BOOL' (aka 'bool') with an expression of incompatible type 'void'` 等错误，是由于 SDK 更新，导致 SDK 与 flutter 端原生代码不兼容。此时只需要更新 SDK 版本即可。

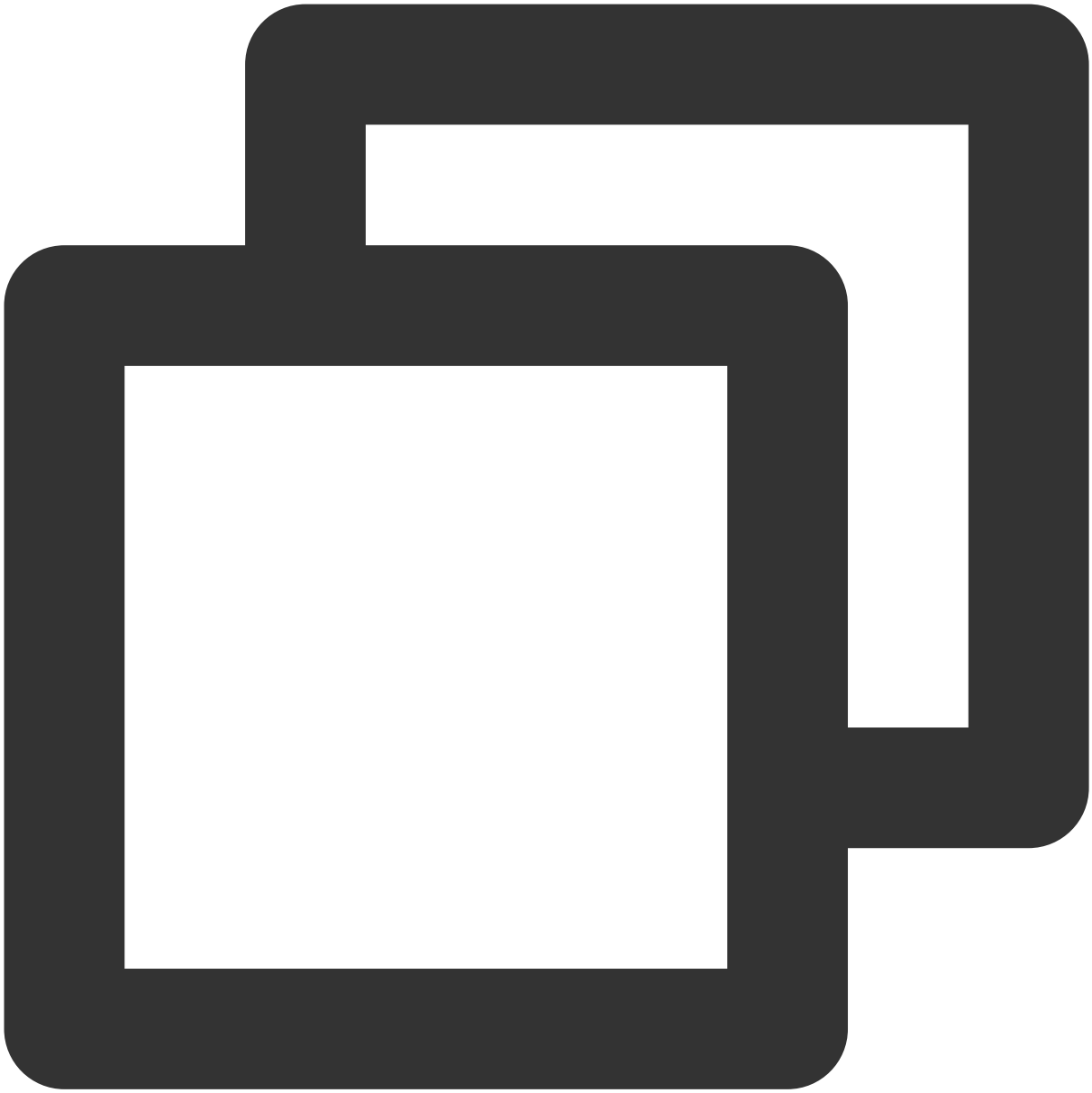
解决方法：在项目目录下，打开终端，依次输入如下命令：



```
flutter pub cache clean
flutter clean
flutter pub upgrade
flutter pub get
```

确保命令执行成功，更新本地 flutter 依赖。

然后在 ios 目录下，打开终端，输入如下命令，更新 IOS 依赖：



```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

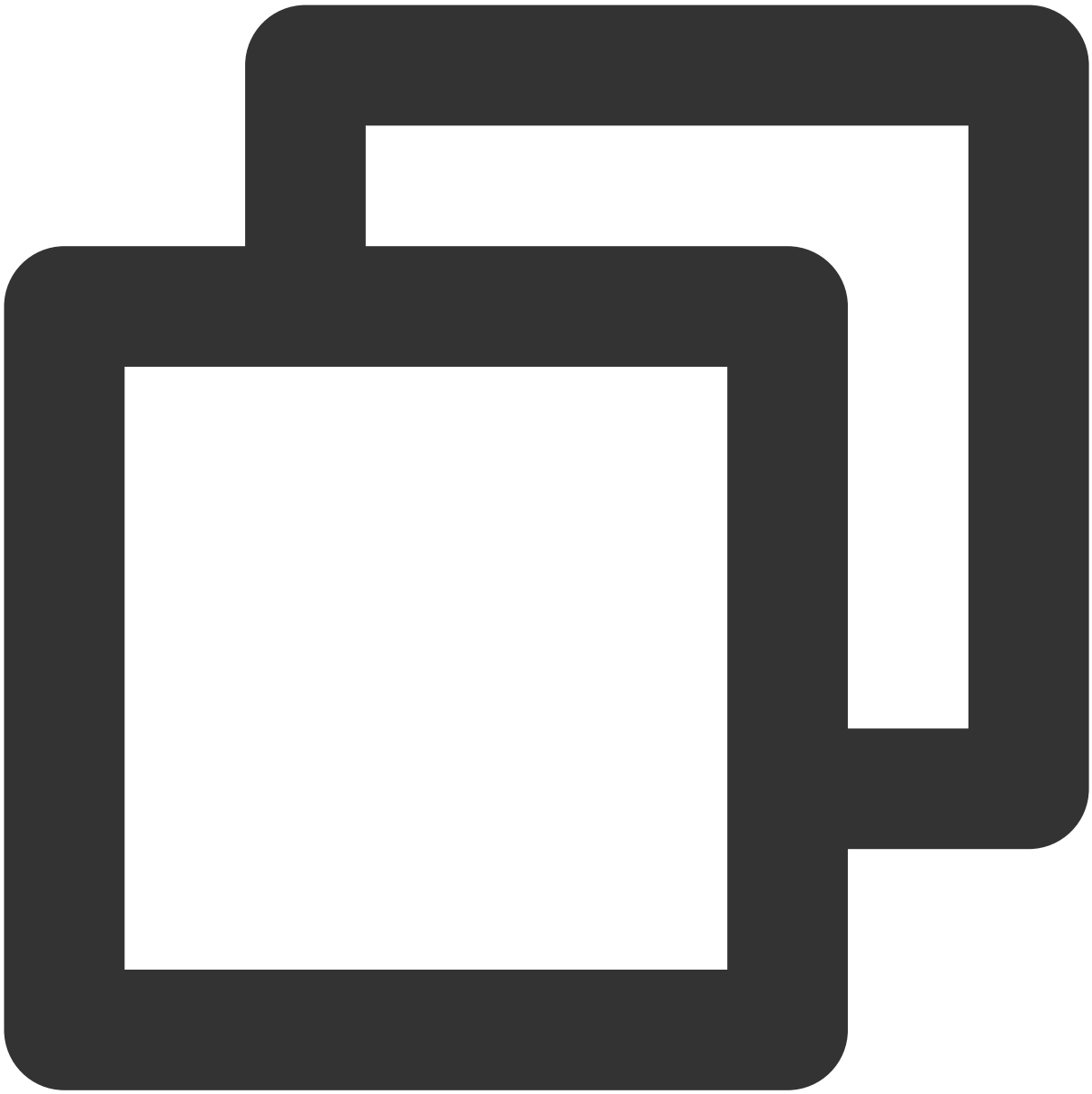
如果问题依然存在，可以尝试删除项目 build 文件夹，并且手动删除您电脑中的 flutter 依赖缓存文件夹 `.pubcache` 。然后重新刷新 flutter pub 依赖再进行编译运行。

10. 安卓点播播放器播放视频，播放器边缘出现平铺拉伸现象。

该问题是 flutter 端 sdk 的纹理渲染问题，可以将 flutter 版本升级到 flutter 3.7.0以上。

11. flutter 调试和测试包运行没问题，但是打正式包会闪退。

flutter 打正式包默认是开启混淆的，播放器SDK需要配置如下混淆规则：



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

12. 播放本地视频无法播放

flutter 播放器支持本地视频播放，需要将正确的本地视频地址传入到视频播放接口中，出现无法播放现象，首先需要检查本地视频地址是否可用，文件是否损坏，如果本地视频没有问题，需要检查应用是否具有存储或者图片/视频读取权限。

13. 运行 iOS 项目出现 `CocoaPods could not find compatible versions for pod "Flutter"` 等类似报错

该问题是由于在高 flutter 开发环境中，已经不再支持 iOS 低版本，可以检查项目中 Minimum Deployments 配置的 iOS 版本是否过小，或者是否继承了只支持低 iOS 版本的依赖。

更多功能

你可以通过运行项目中的 example 体验完整功能，example 运行指引。
播放器 SDK 官网提供了 iOS、Android 和 Web 端的 Demo 体验，请单击这里 。

点播场景

最近更新时间：2024-04-26 11:09:31

阅读对象

本文档部分内容为腾讯云专属能力，使用前请开通 腾讯云 相关服务，未注册用户可注册账号 免费试用。

通过本文你可以学会

如何集成腾讯云视立方 Flutter 播放器 SDK。
如何使用播放器 SDK 进行点播播放。
如何使用播放器 SDK 底层能力实现更多功能。

基础知识

本文主要介绍视频云 SDK 的点播播放功能，在此之前，先了解如下一些基本知识会大有裨益：
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
通常使用的点播协议如下，现在比较流行的是 HLS（以“http”打头，以“.m3u8”结尾）的点播地址。

特别说明

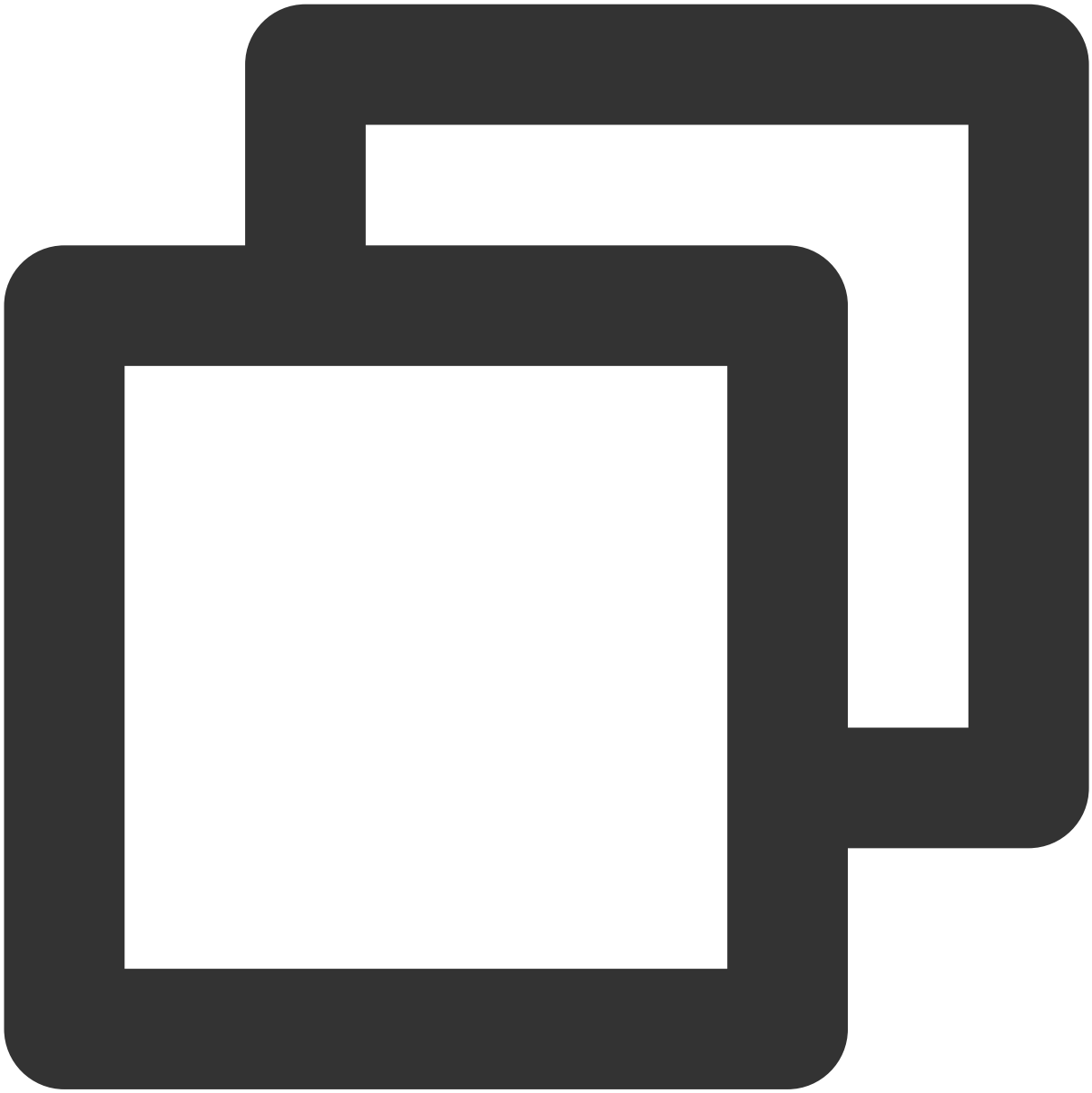
视频云 SDK **不会对播放地址的来源做限制**，即您可以用它来播放腾讯云或非腾讯云的播放地址。但视频云 SDK 中的播放器只支持 FLV 、RTMP 和 HLS（m3u8）三种格式的直播地址，以及 MP4、HLS（m3u8）和 FLV 三种格式的点播地址。

SDK 集成

步骤1：集成 SDK 开发包

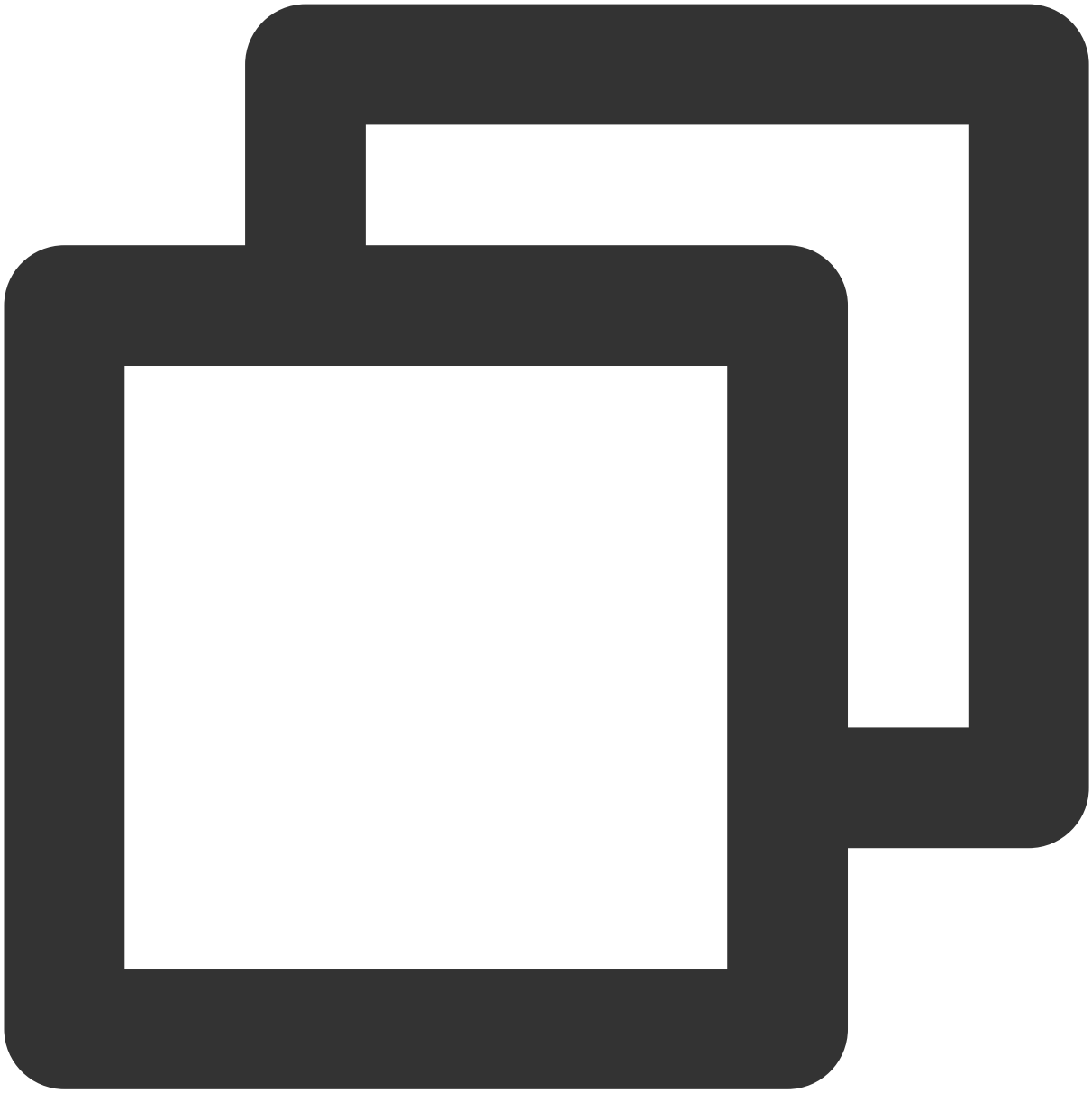
下载和集成 SDK 开发包，请参考 集成指引。

步骤2：创建 controller



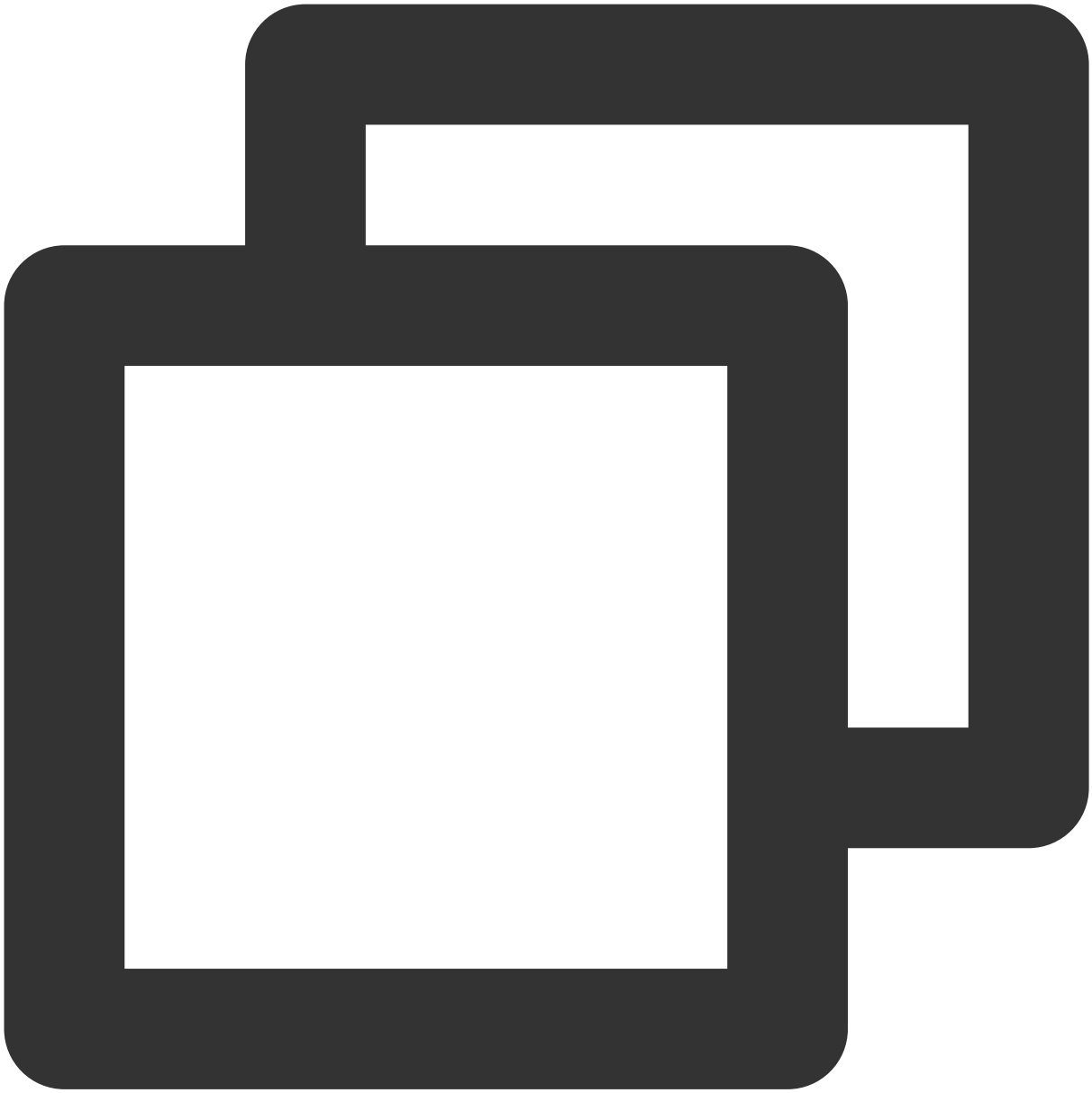
```
TXVodPlayerController _controller = TXVodPlayerController();
```

步骤3：设置监听事件



```
// 监听视频宽高变化，设置合适的宽高比例，也可自行设置宽高比例，视频纹理也会根据比例进行相应拉伸
_controller.onPlayerNetStatusBroadcast.listen((event) async {
  double w = (event["VIDEO_WIDTH"]).toDouble();
  double h = (event["VIDEO_HEIGHT"]).toDouble();

  if (w > 0 && h > 0) {
    setState(() {
      _aspectRatio = 1.0 * w / h;
    });
  }
});
```

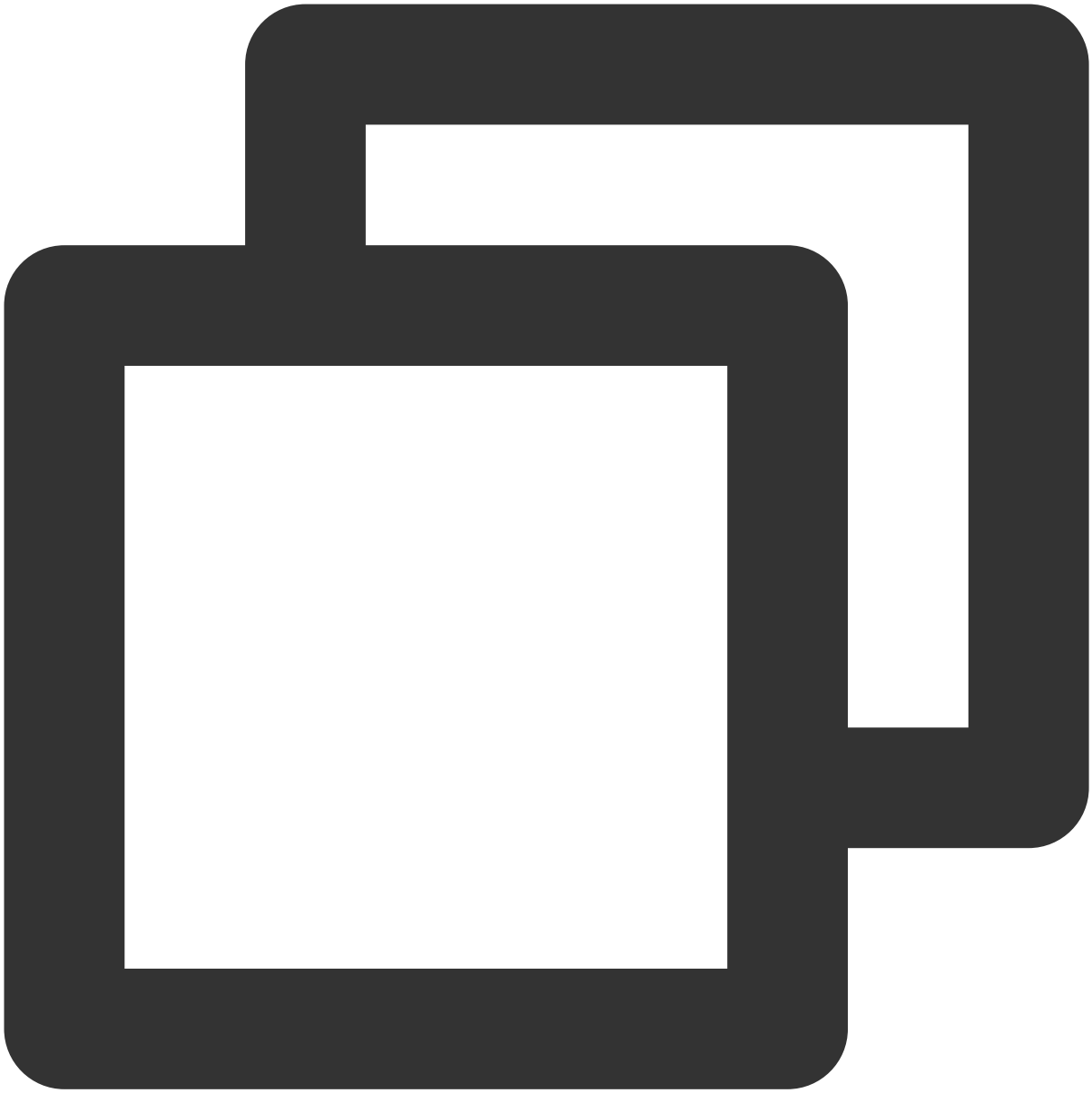
步骤4：添加布局



```
@override
Widget build(BuildContext context) {
return Container(
  decoration: BoxDecoration(
      image: DecorationImage(
        image: AssetImage("images/ic_new_vod_bg.png"),
        fit: BoxFit.cover,
```

```
        )),
    child: Scaffold(
        backgroundColor: Colors.transparent,
        appBar: AppBar(
          backgroundColor: Colors.transparent,
          title: const Text('点播'),
        ),
        body: SafeArea(
            child: Container(
              height: 150,
              color: Colors.black,
              child: Center(
                child: _aspectRatio > 0
                    ? AspectRatio(
                  aspectRatio: _aspectRatio,
                  child: TXPlayerVideo(controller: _controller),
                ) : Container(),
              ),
            ))));
}
```

步骤5：播放器初始化



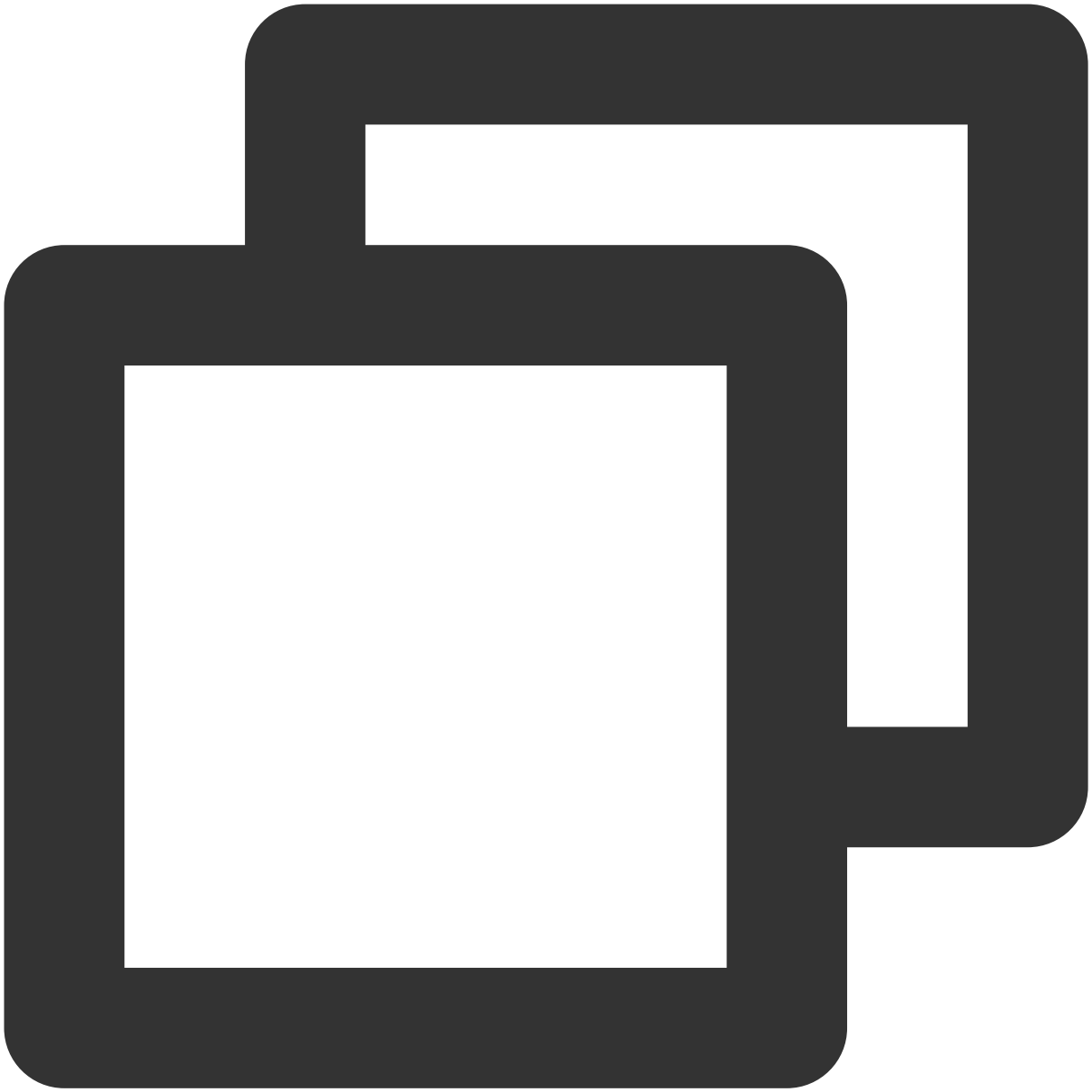
```
// 初始化播放器，分配共享纹理
await _controller.initialize();
```

步骤6：启动播放

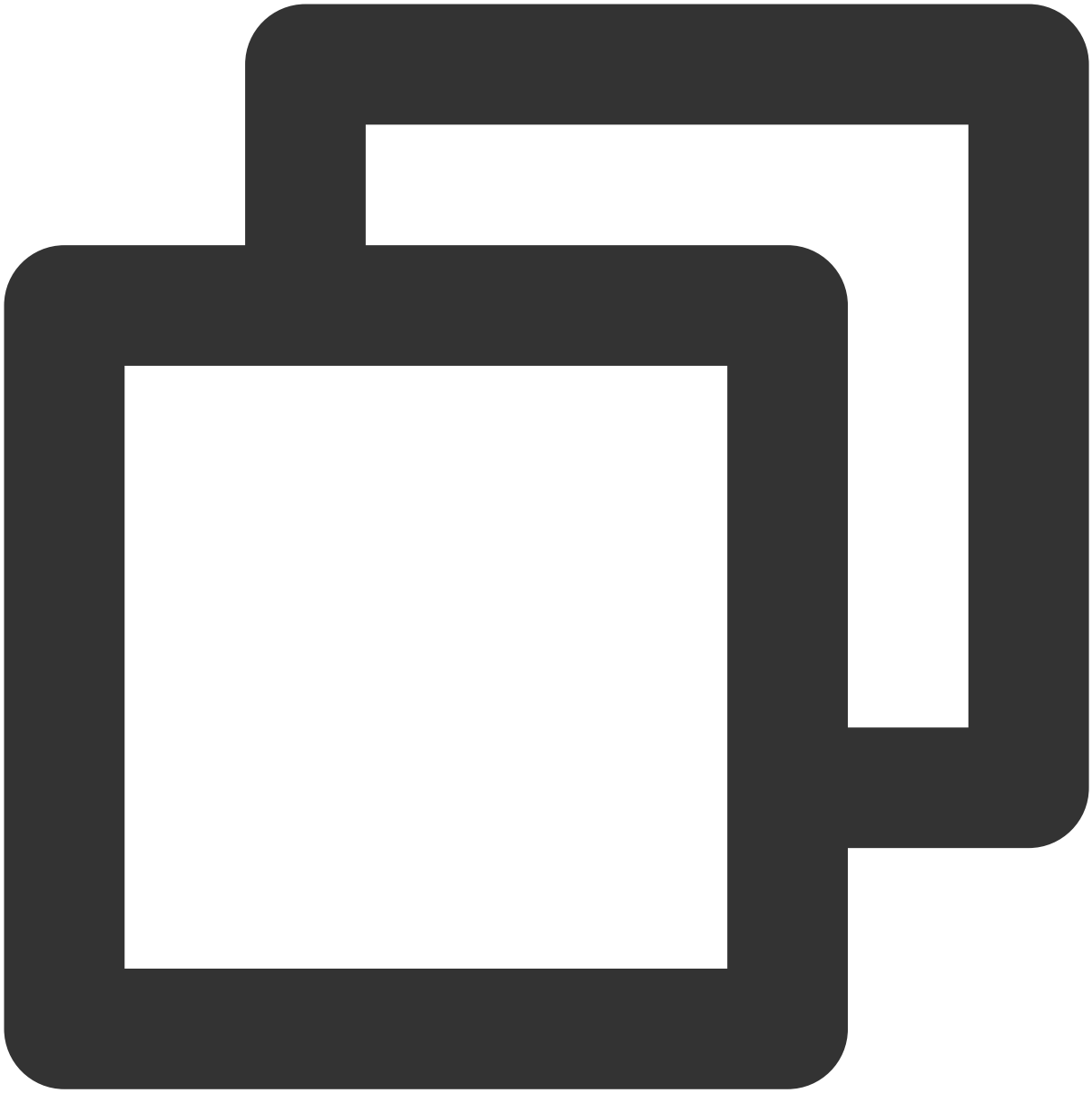
通过 url 方式

通过 field 方式

TXVodPlayerController 内部会自动识别播放协议，您只需要将您的播放 URL 传给 startVodPlay 函数即可。



```
// 播放视频资源
String _url =
    "http://1400329073.vod2.myqcloud.com/d62d88a7vodtranscq1400329073/59c68fe752858
await _controller.startVodPlay(_url);
```



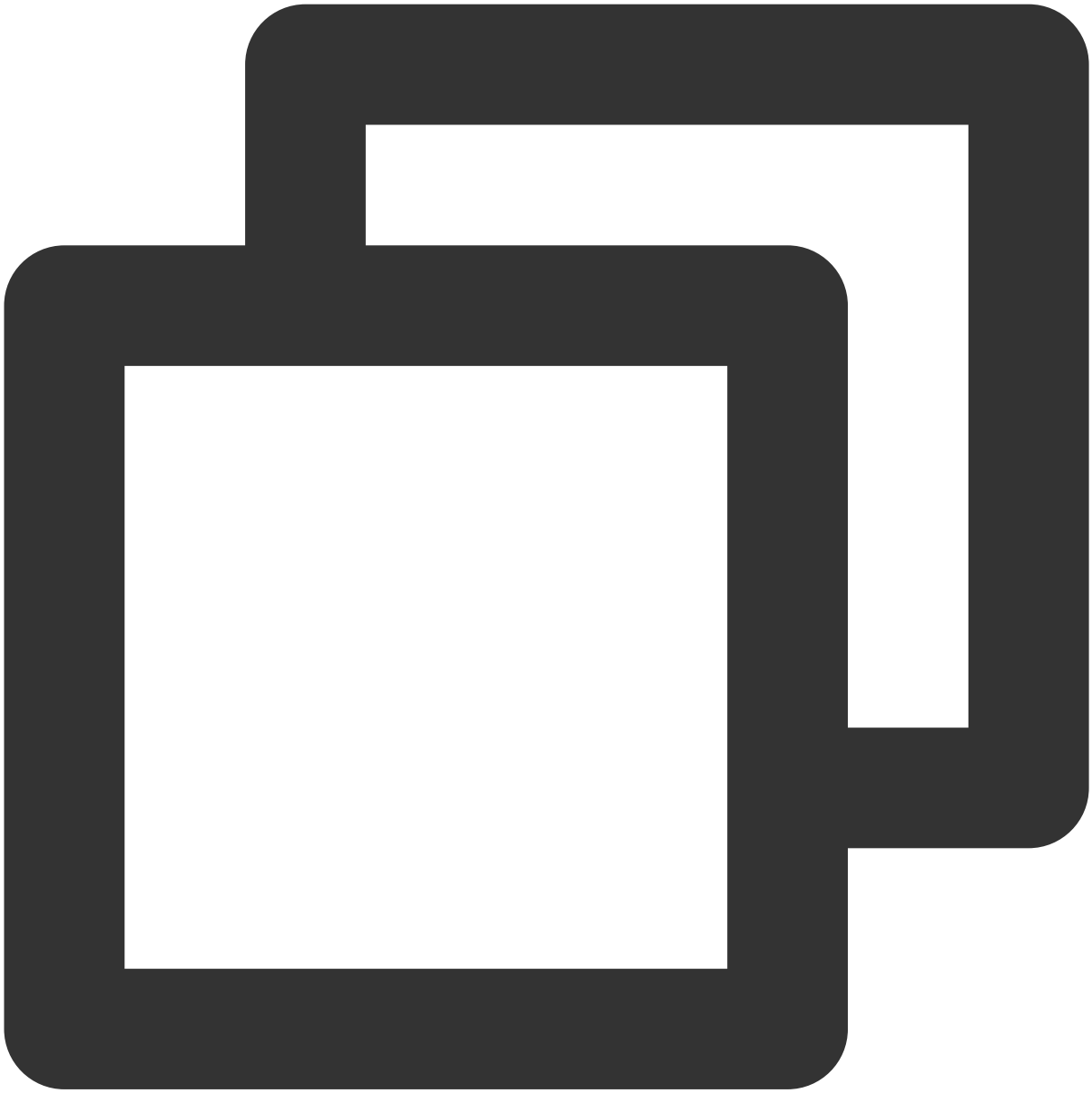
```
// psign 即播放器签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/p
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
        fileId: "4564972819220421305", psign: "psignxxxxxxx");
await _controller.startVodPlayWithParams(params);
```

在 媒资管理 找到对应的视频文件。在文件名下方可以看到 FileId。

通过 FileId 方式播放，播放器会向后台请求真实的播放地址。如果此时网络异常或 FileId 不存在，则会收到 `TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL` 事件，反之收到 `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` 表示请求成功。

步骤7：结束播放

结束播放时**记得调用 controller 的销毁方法**，尤其是在下次 startVodPlay 之前，否则可能会产生大量的内存泄露以及闪屏问题。

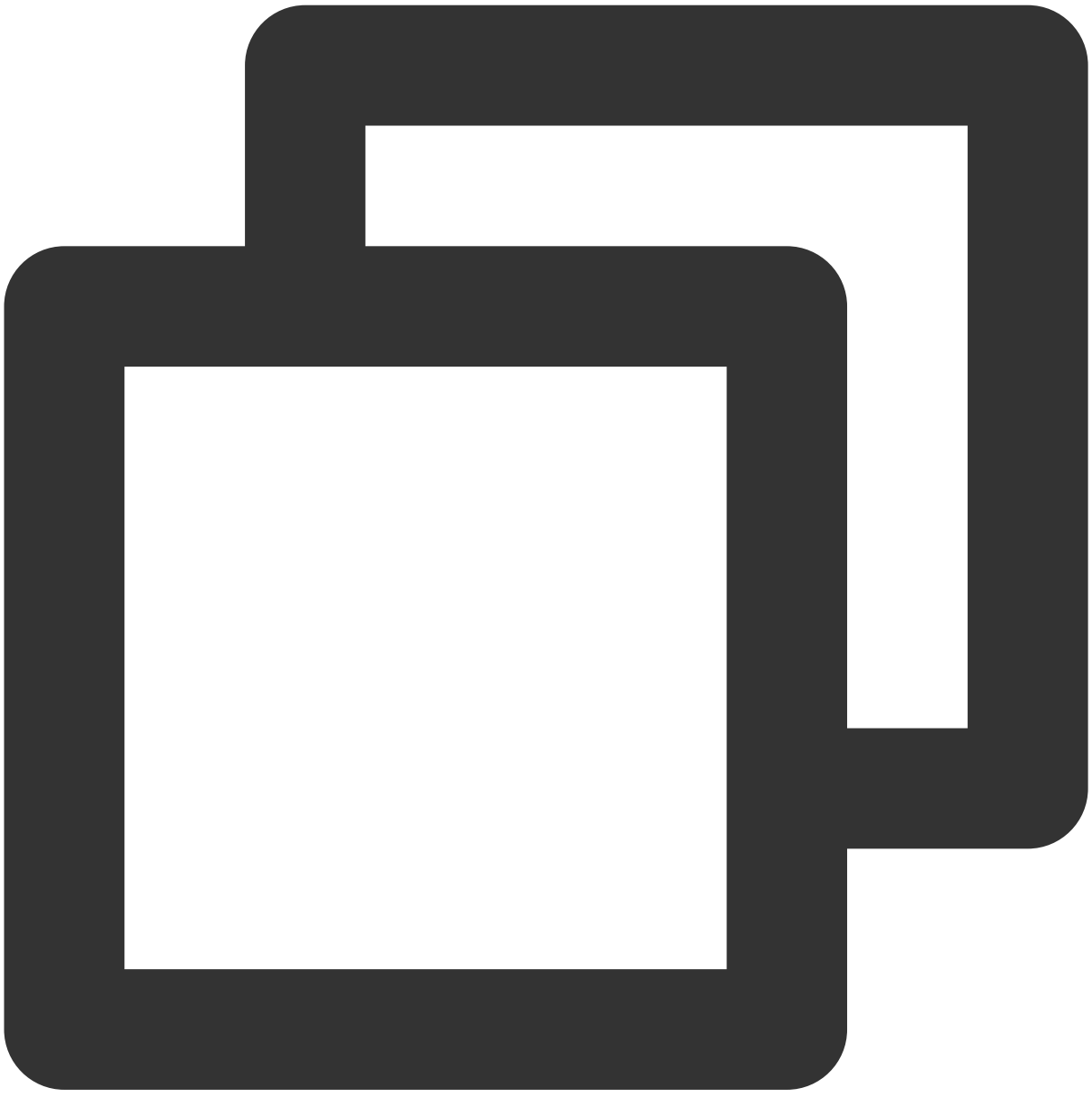


```
@override
void dispose() {
  _controller.dispose();
  super.dispose();
}
```

基础功能使用

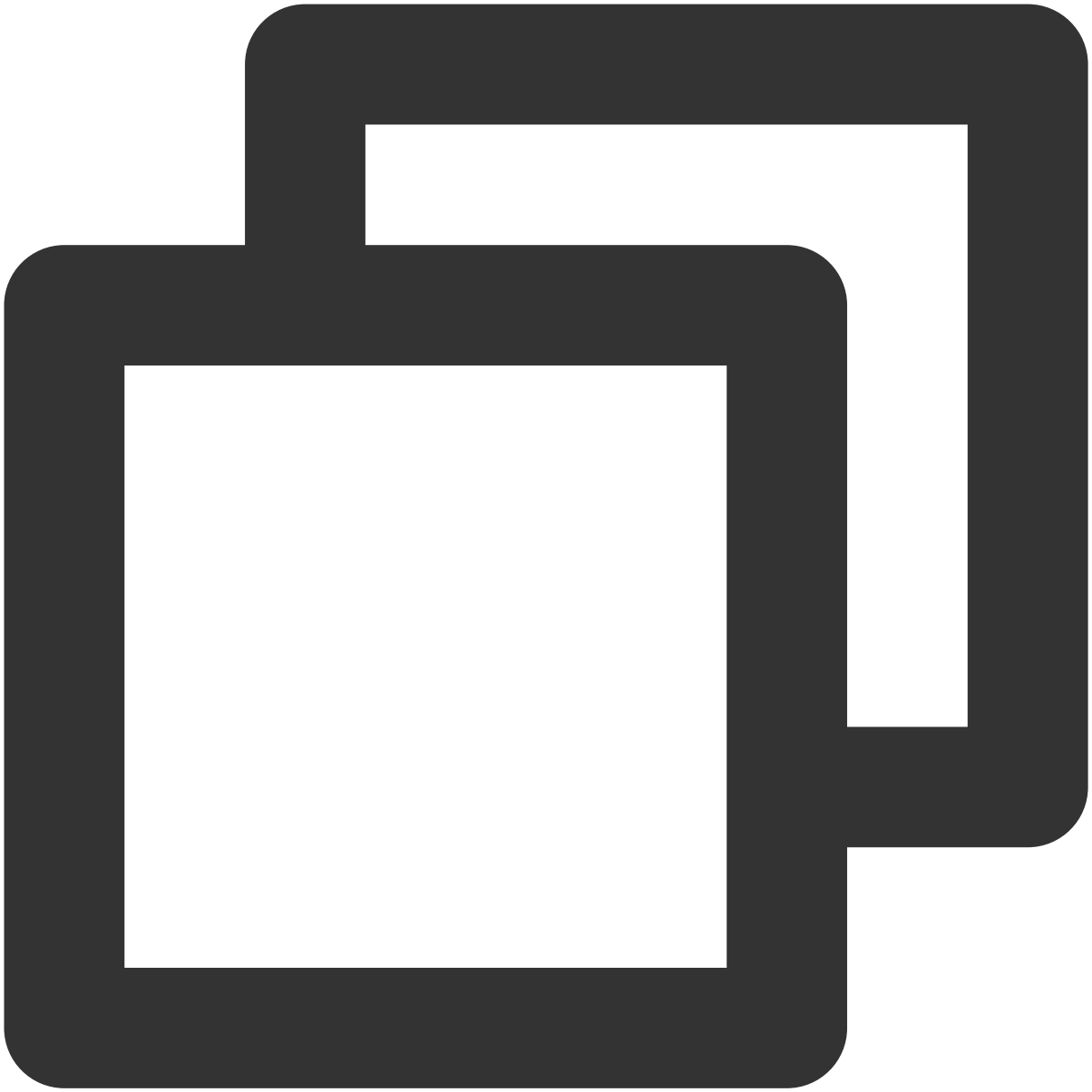
1、播放控制

开始播放



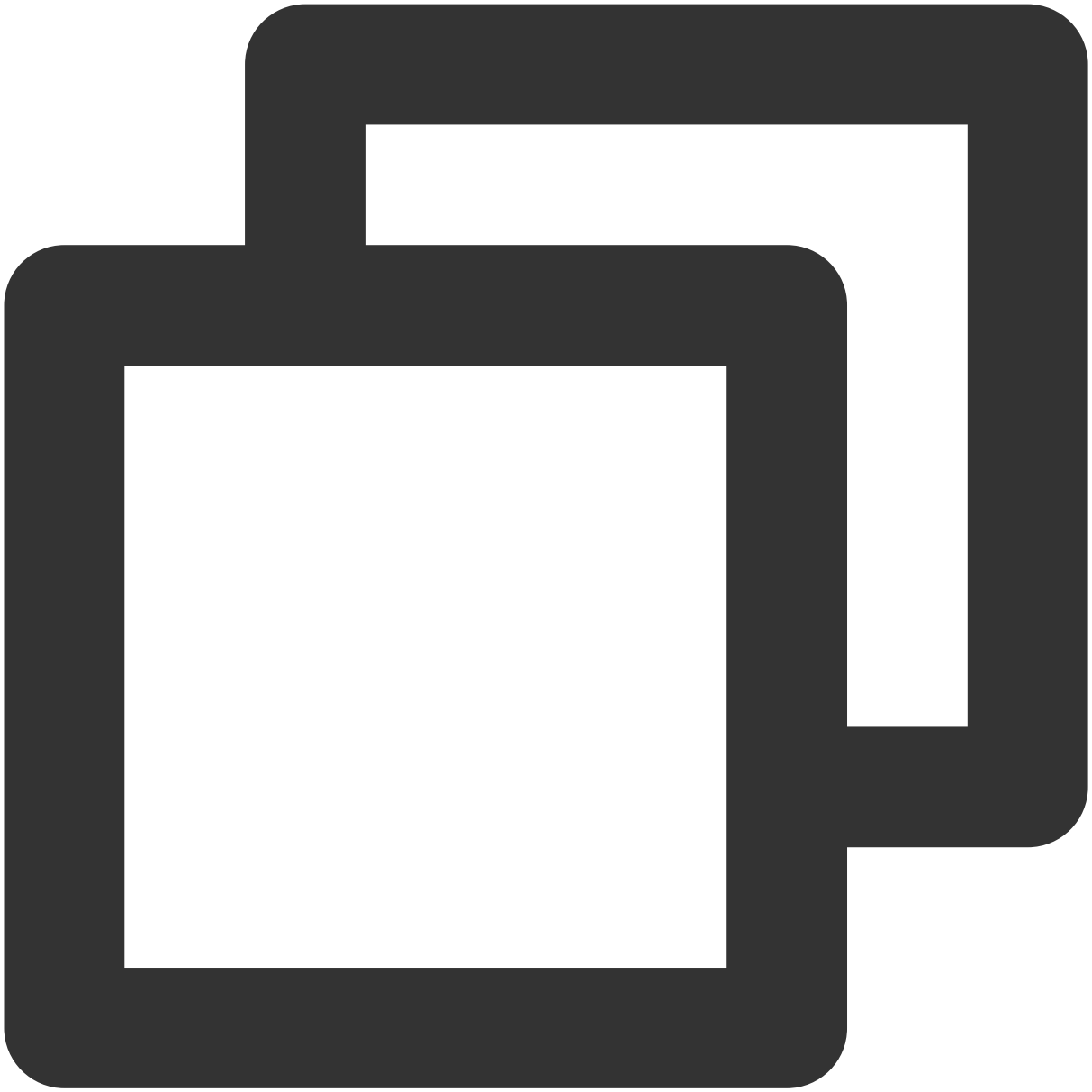
```
// 开始播放
_controller.startVodPlay(url)
```

暂停播放



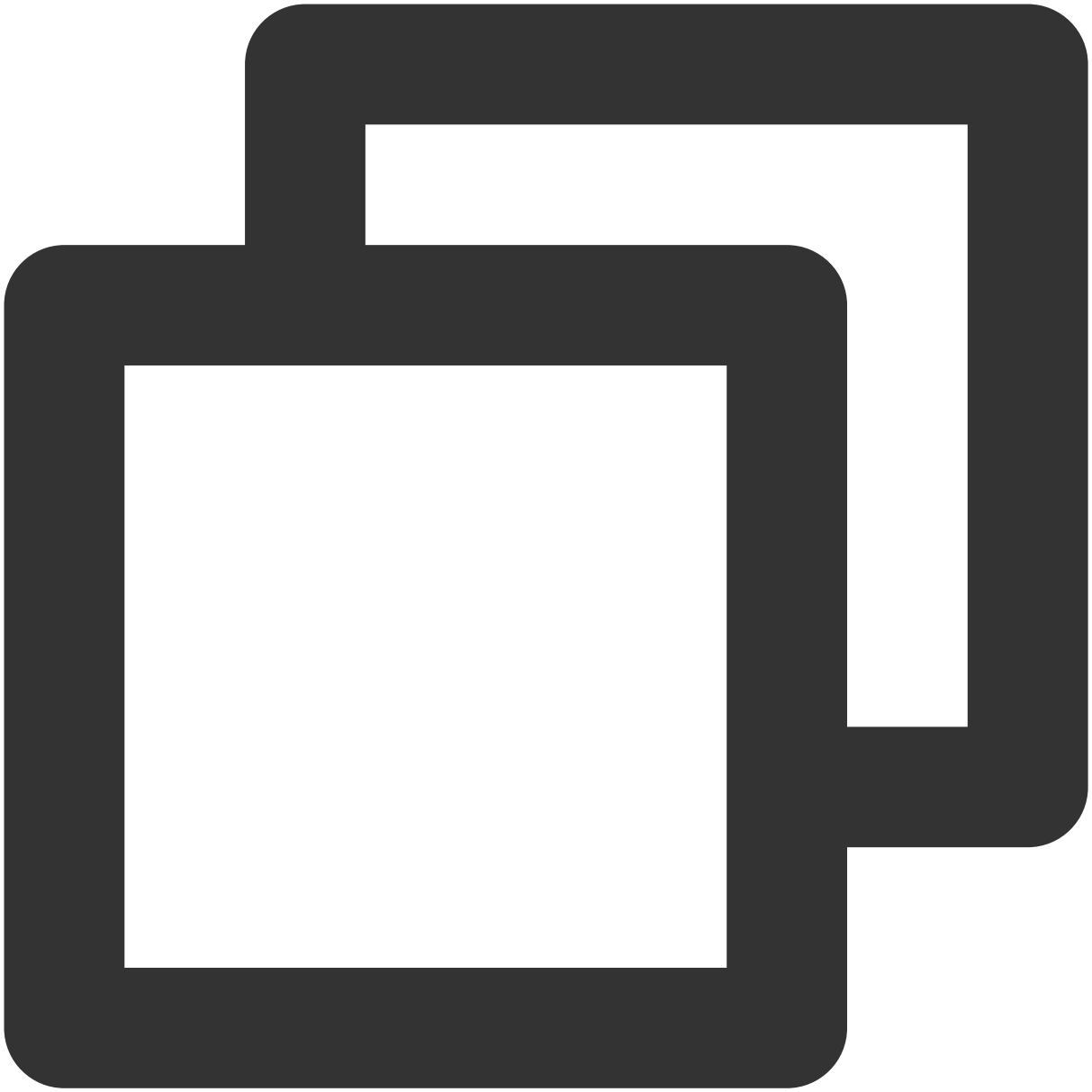
```
// 暂停播放
_controller.pause();
```

恢复播放



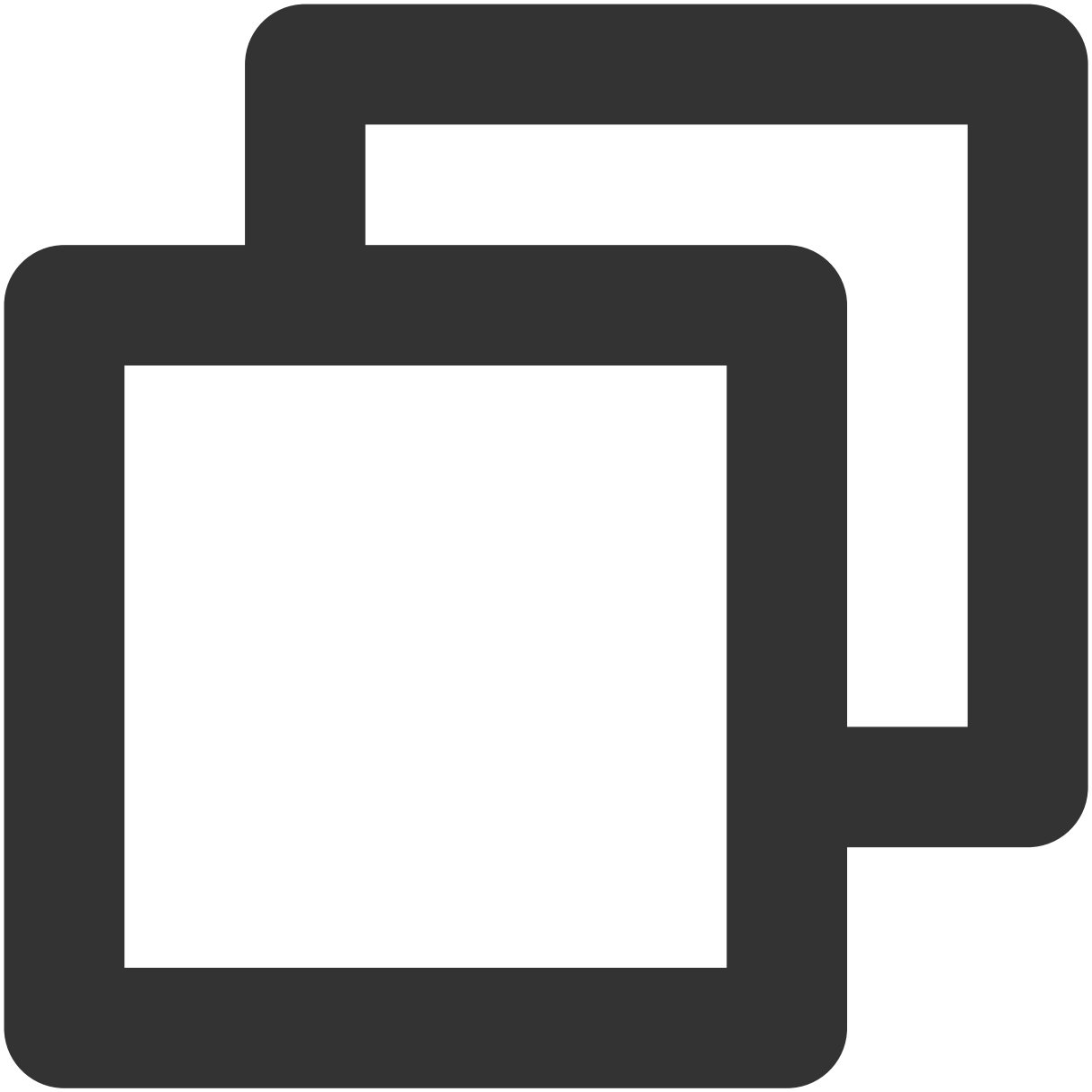
```
// 恢复播放
_controller.resume();
```

结束播放



```
// 结束播放
_controller.stopPlay(true);
```

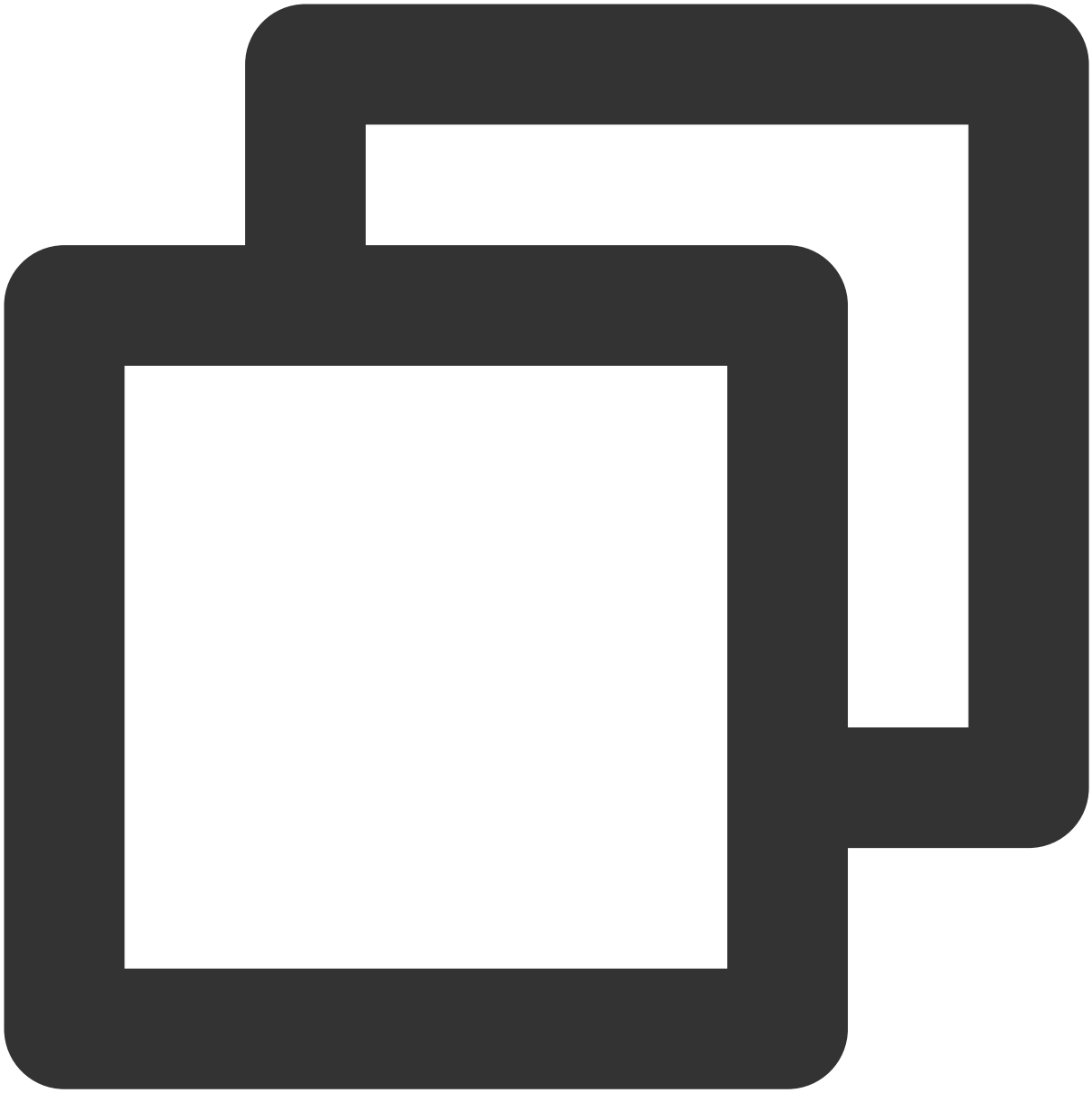
结束播放器



```
// 释放controller
_controller.dispose();
```

调整进度（Seek）

当用户拖拽进度条时，可调用 seek 从指定位置开始播放，播放器 SDK 支持精准 seek。

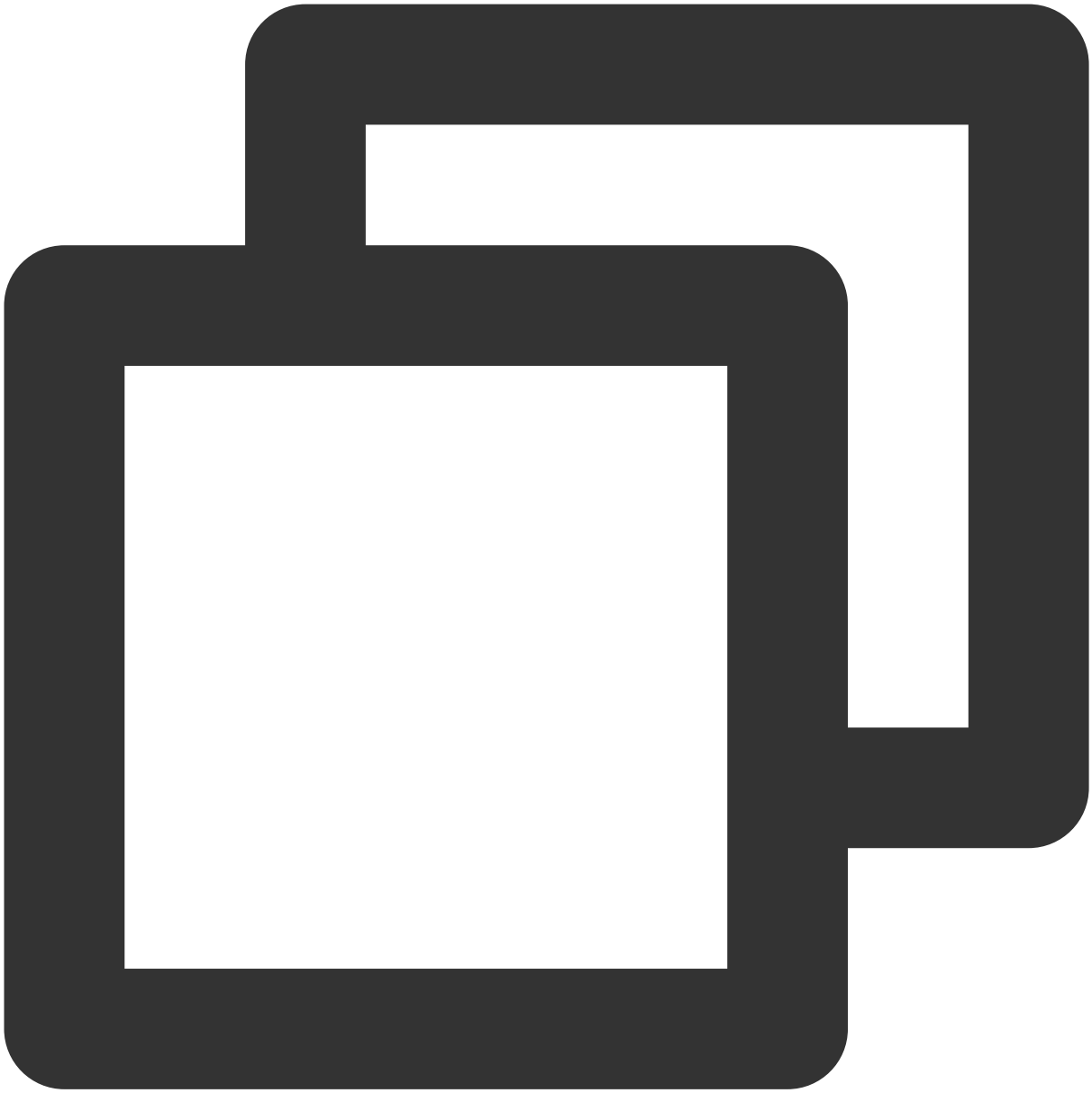


```
double time = 600; // double, 单位为 秒
// 调整进度
_controller.seek(time);
```

Seek 到视频流指定 PDT 时间点

跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。

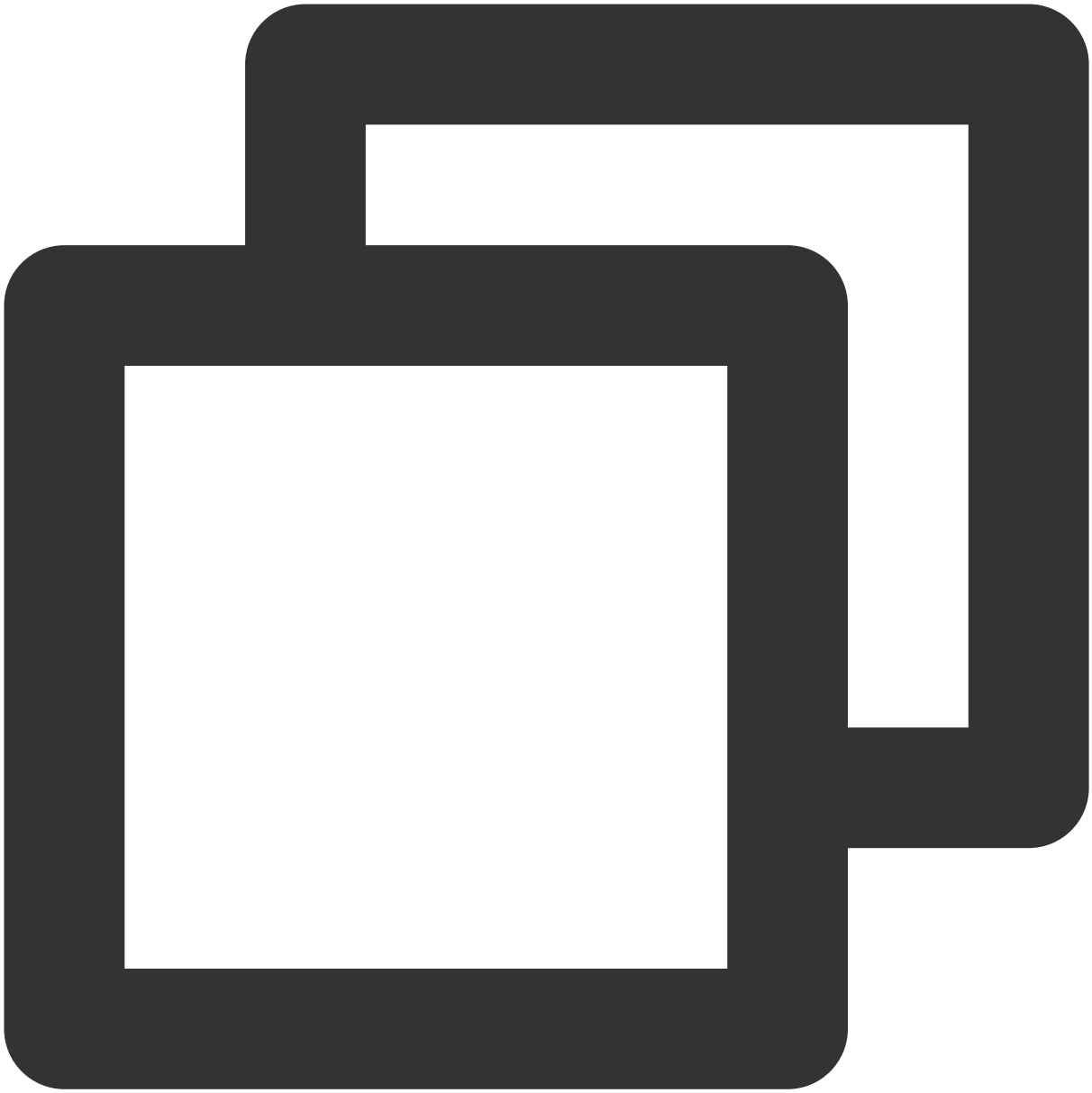
注意：播放器高级版 11.7 版本开始支持。



```
int pdtTimeMs = 600; // 单位为 毫秒
_controller.seekToPdtTime(time);
```

从指定时间开始播放

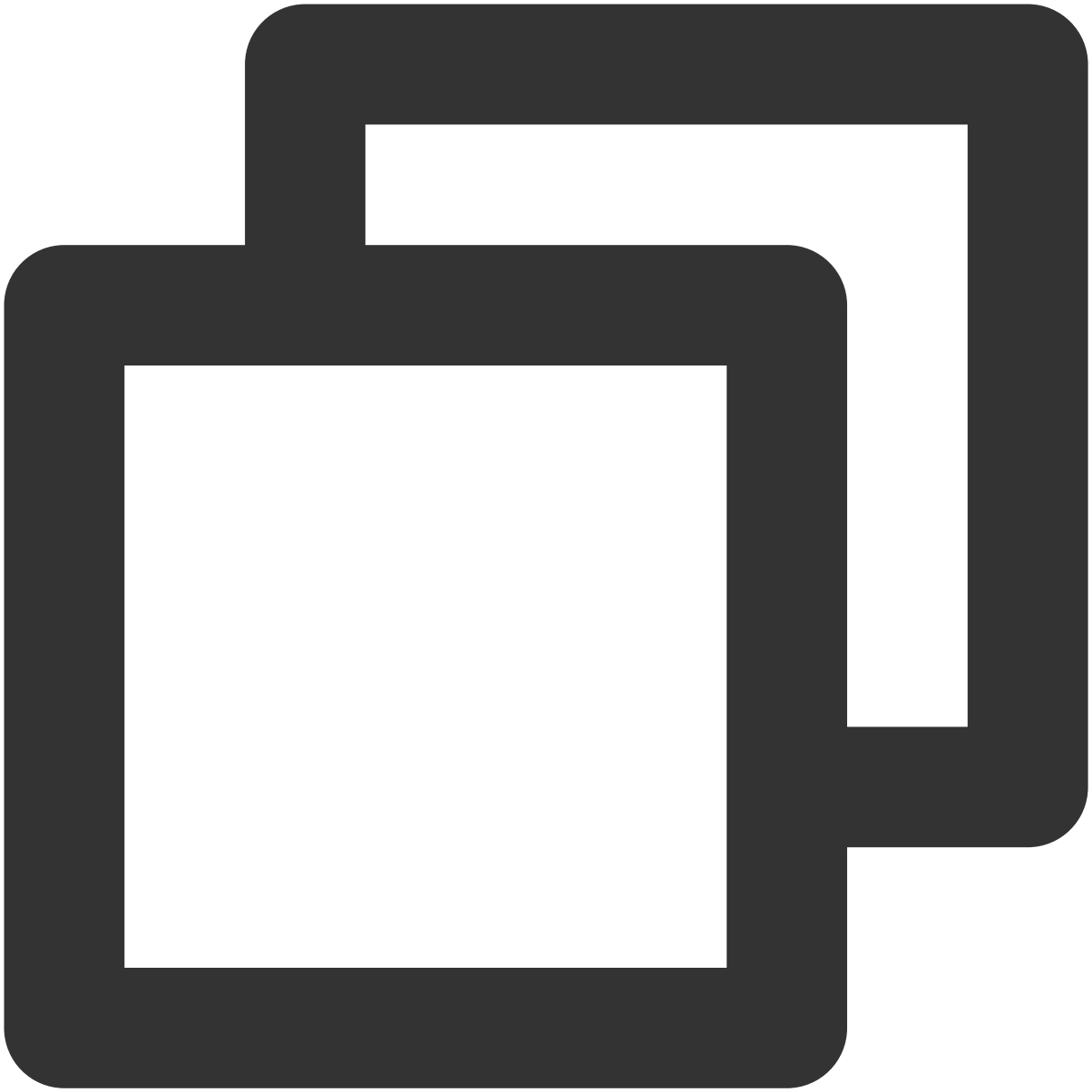
首次调用 startVodPlay 之前，支持从指定时间开始播放。



```
double startTimeInSecond = 60; // 单位：秒
_controller.setStartTime(startTimeInSecond);    // 设置开始播放时间
_controller.startVodPlay(url);
```

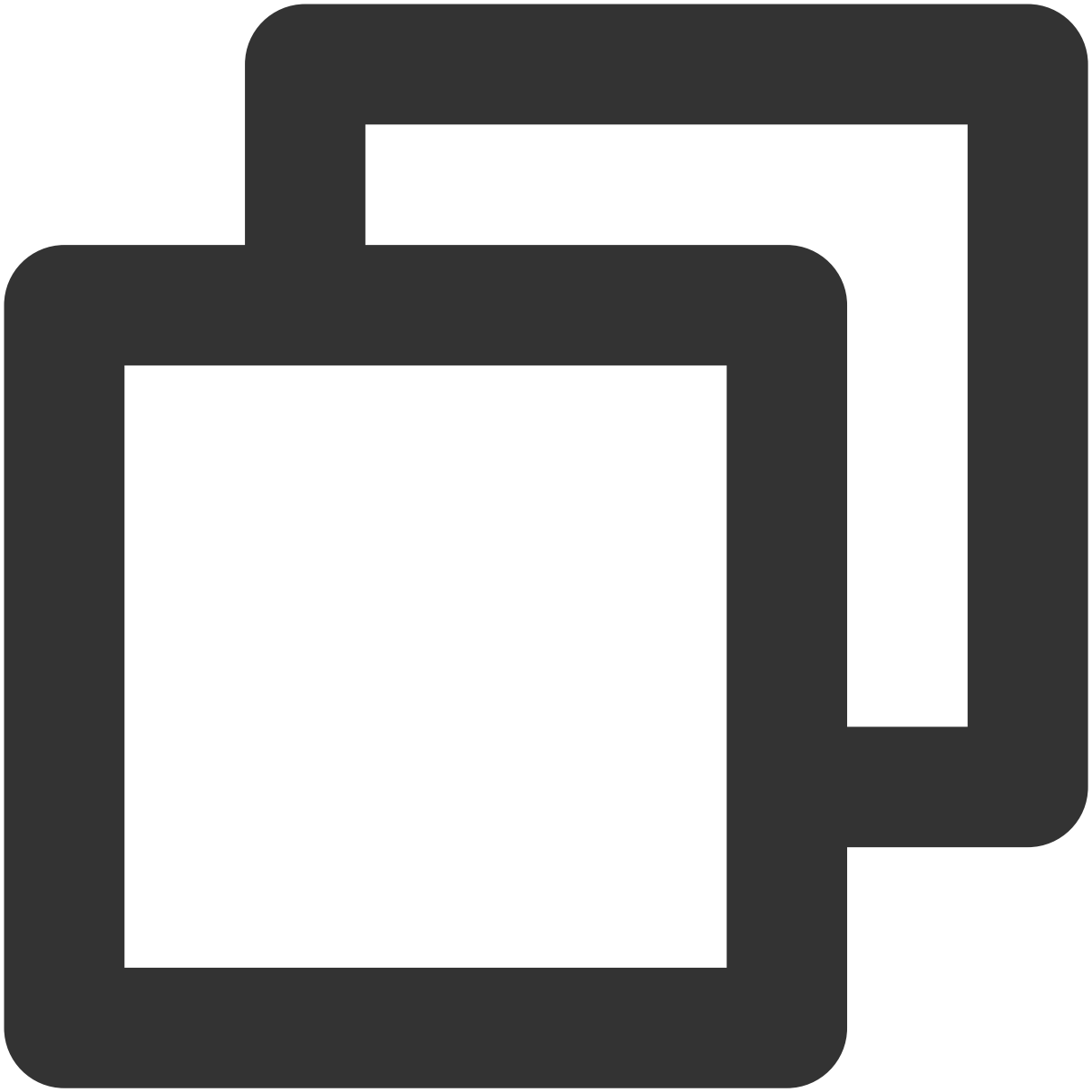
2、变速播放

点播播放器支持变速播放，通过接口 `setRate` 设置点播播放速率来完成，支持快速与慢速播放，如0.5X、1.0X、1.2X、2X等。



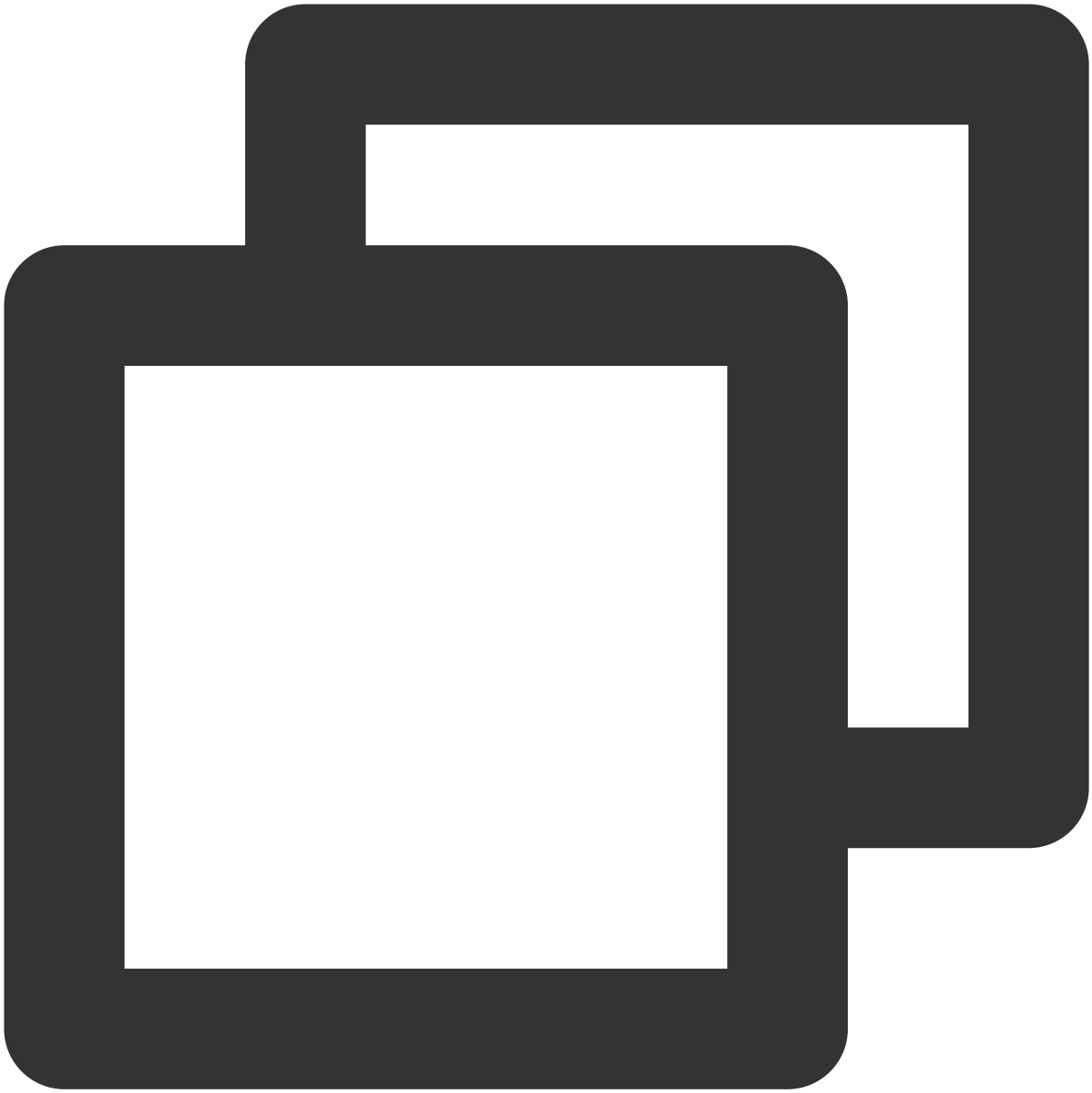
```
// 设置1.2倍速播放
_controller.setRate(1.2);
```

3、循环播放



```
// 设置循环播放
_controller.setLoop(true);
// 获取当前循环播放状态
_controller.isLoop();
```

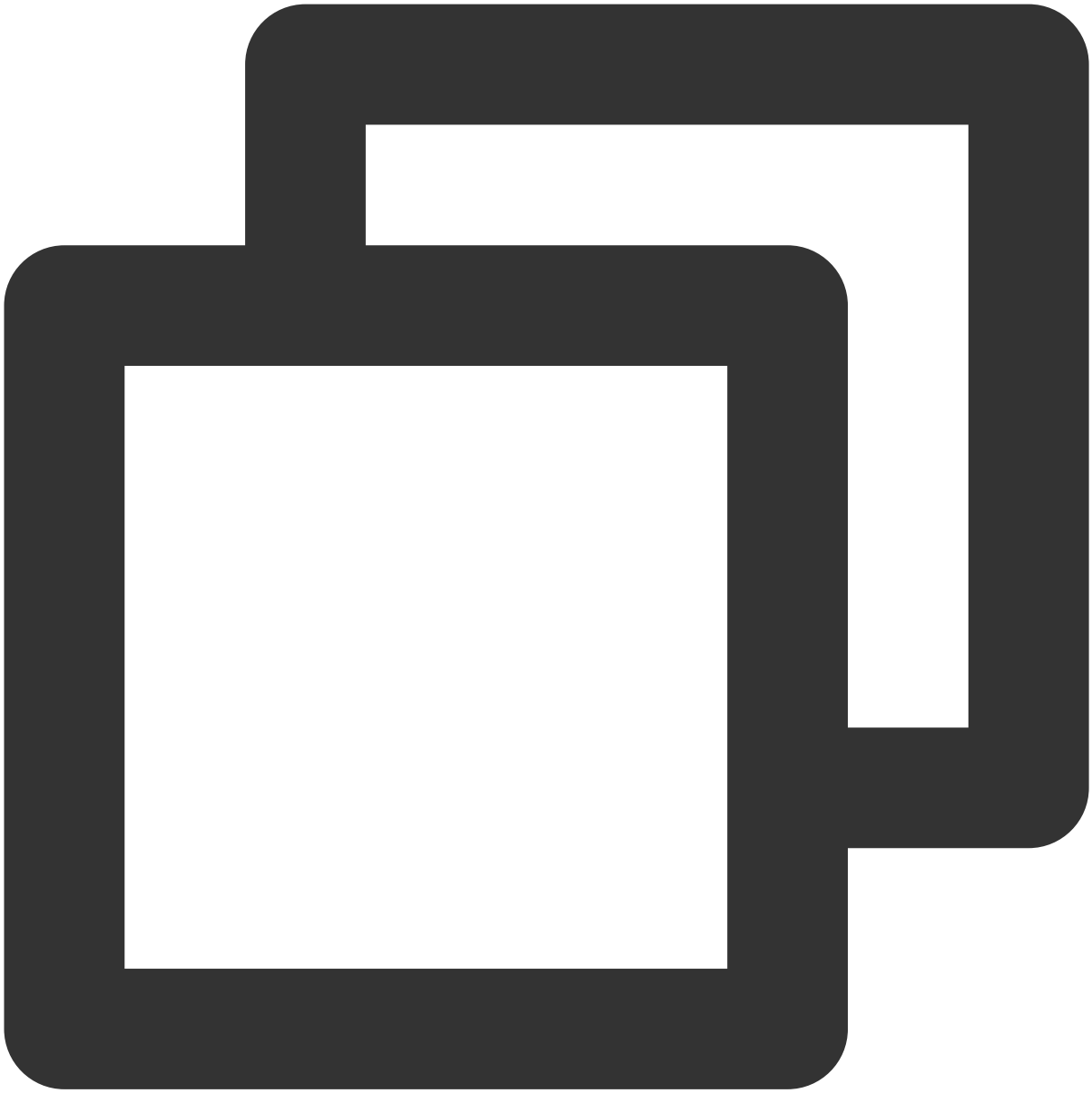
4、静音设置



```
// 设置静音，true 表示开启静音， false 表示关闭静音
_controller.setMute(true);
```

5 、贴片广告

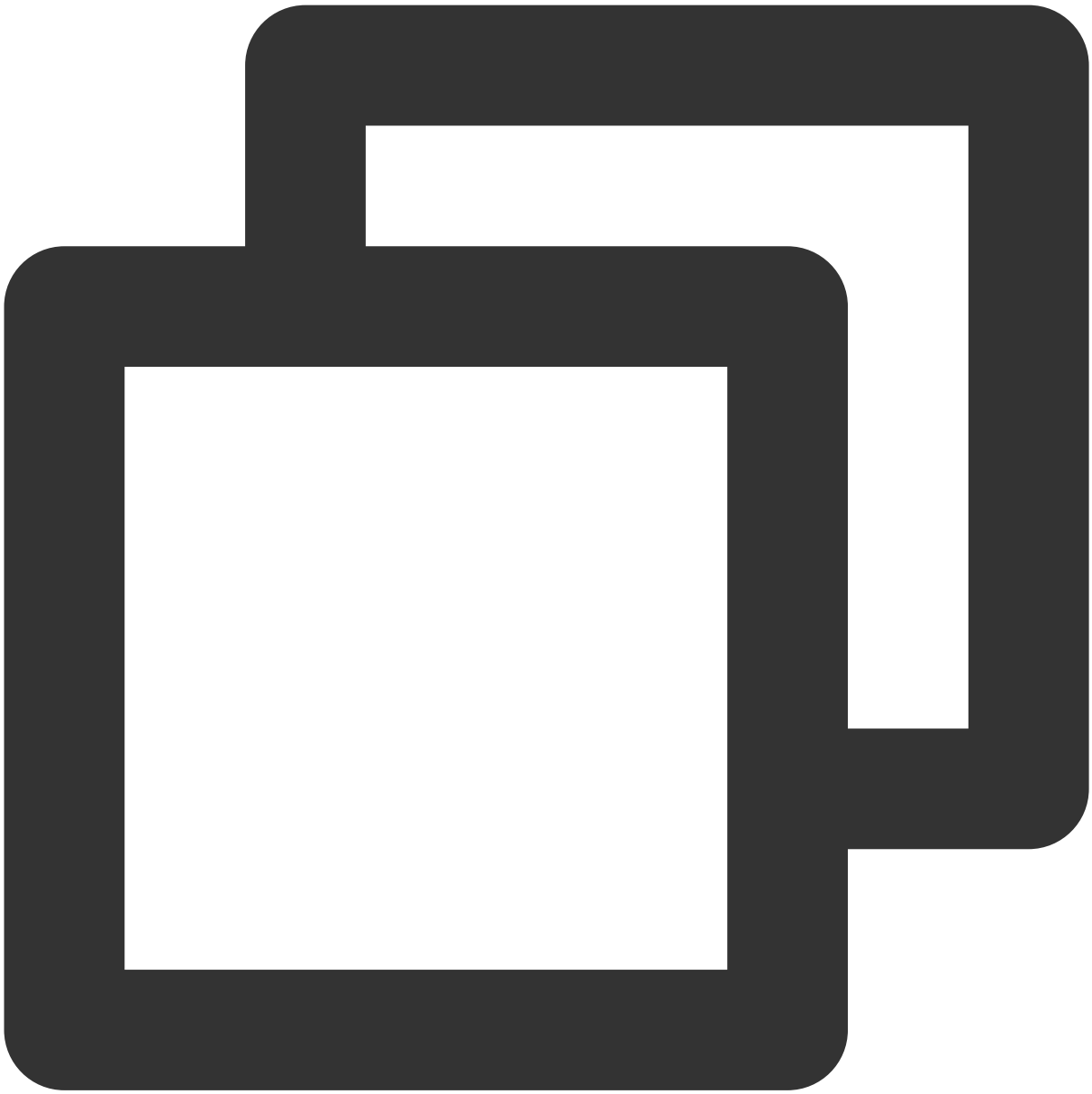
播放器SDK支持在界面添加图片贴片，用于广告宣传等。实现方式如下：
将 `autoPlay` 为 NO，此时播放器会正常加载，但视频不会立刻开始播放。
在播放器加载出来后、视频尚未开始时，即可在播放器界面查看图片贴片广告。
待达到广告展示结束条件时，使用 resume 接口启动视频播放。



```
_controller.setAutoPlay(false);  // 设置为非自动播放
_controller.startVodPlay(url);    // startVodPlay 后会加载视频，加载成功后不会自动播放
// ......
// 在播放器界面上展示广告
// ......
_controller.resume();  // 广告展示完调用 resume 开始播放视频
```

6、HTTP-REF

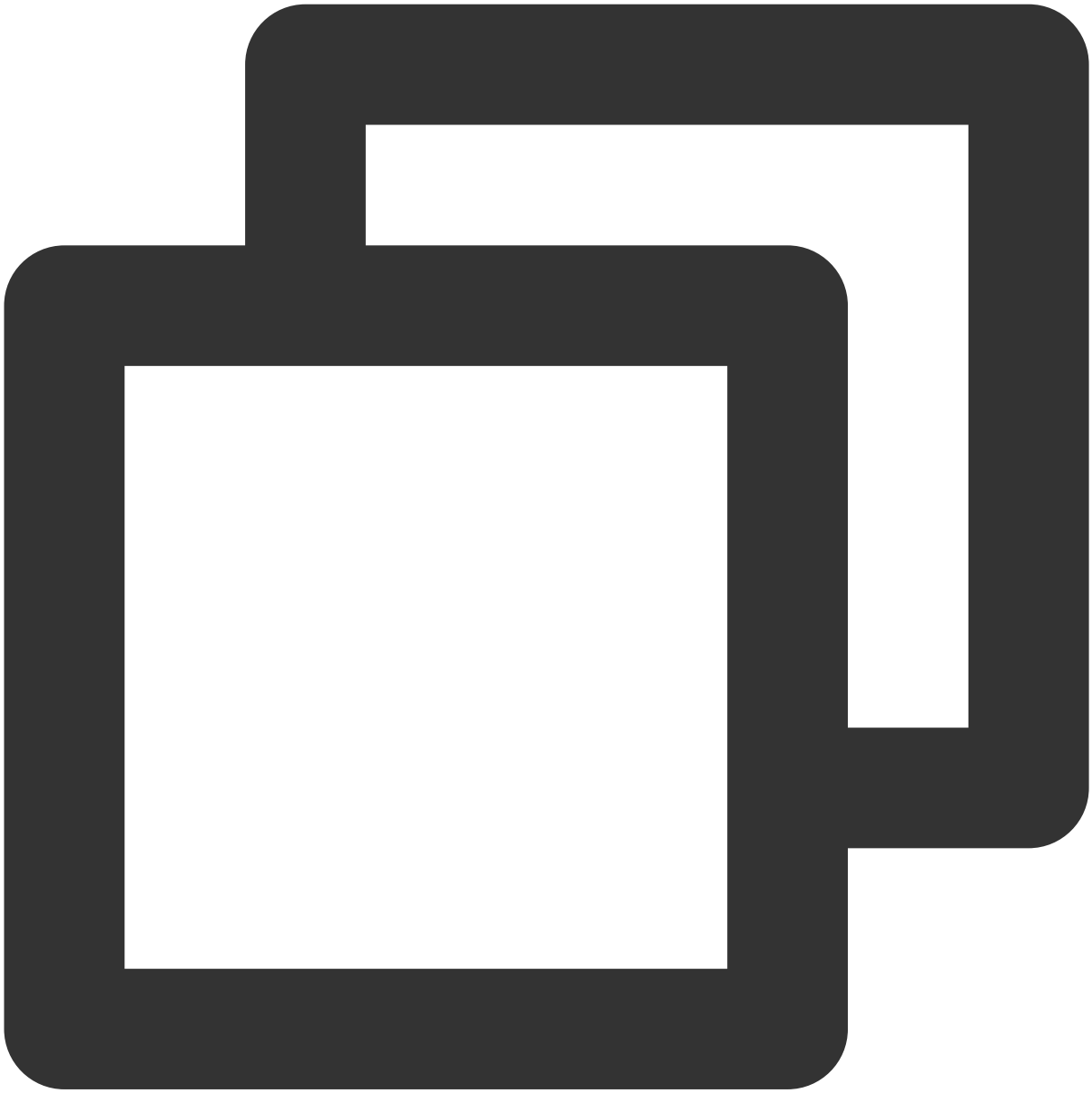
TXVodPlayConfig 中的 headers 可以用来设置 HTTP 请求头，例如常用的防止 URL 被到处拷贝的 Referer 字段（腾讯云可以提供更加安全的签名防盗链方案），以及用于验证客户端身份信息的 Cookie 字段。



```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
Map<String, String> httpHeaders = {'Referer': 'Referer Content'};
playConfig.headers = httpHeaders;
_controller.setConfig(playConfig);
```

7、硬件加速

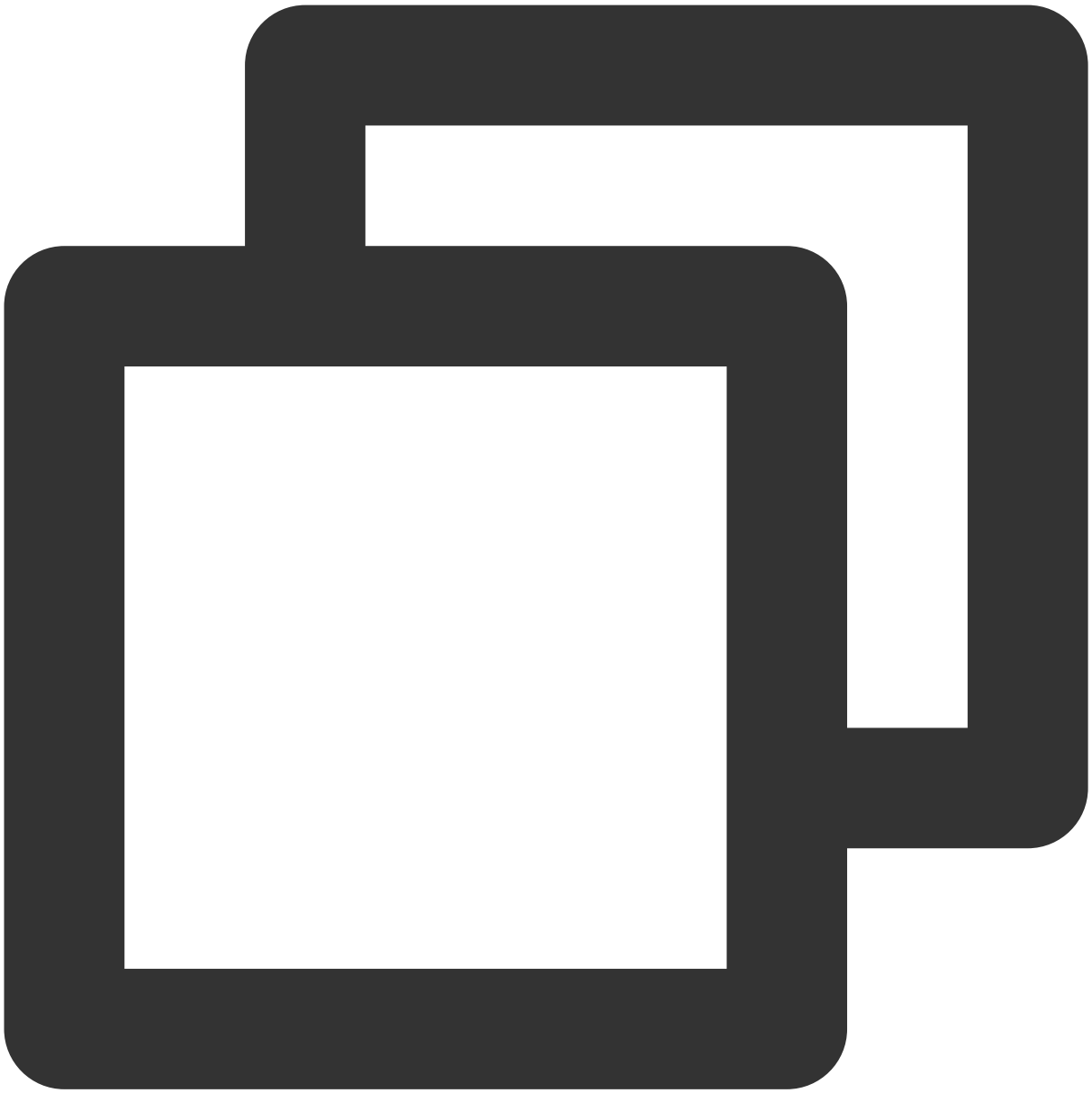
对于蓝光级别（1080p）的画质，简单采用软件解码的方式很难获得较为流畅的播放体验，所以如果您的场景是以游戏直播为主，一般都推荐开启硬件加速。
软解和硬解的切换需要在切换之前先 **stopPlay**，切换之后再 **startVodPlay**，否则会产生比较严重的花屏问题。



```
_controller.stopPlay(true);
_controller.enableHardwareDecode(true);
_controller.startVodPlay(url);
```

8、清晰度设置

SDK 支持 HLS 的多码率格式，方便用户切换不同码率的播放流，从而达到播放不同清晰的目标。可以通过下面方法进行清晰度设置。



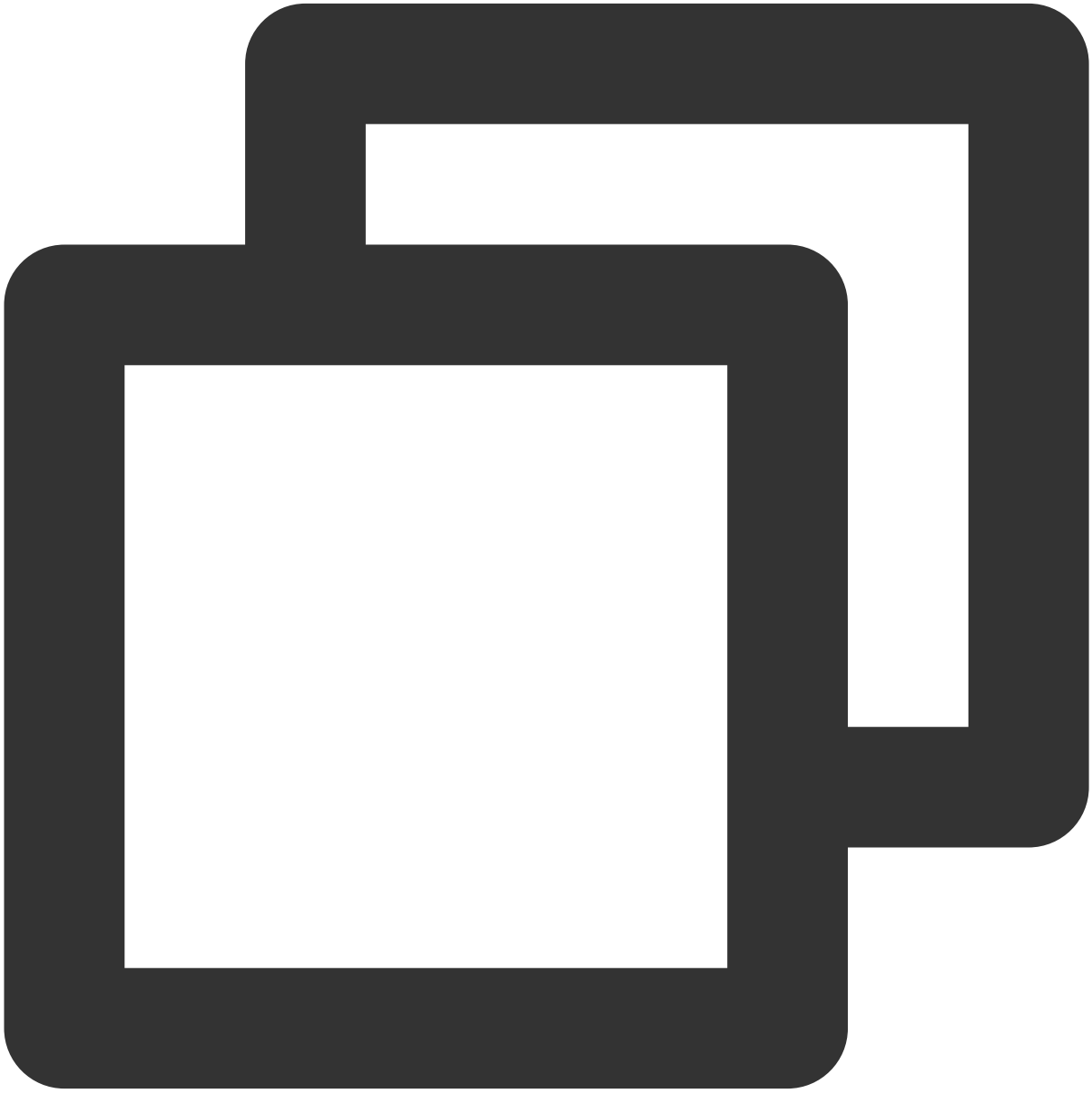
```
List _supportedBitrates = (await _controller.getSupportedBitrates())!;; //获取多码率数
int index = _supportedBitrates[i];  // 指定要播的码率下标
_controller.setBitrateIndex(index);  // 切换码率到想要的清晰度
```

在播放过程中，可以随时通过 `_controller.setBitrateIndex(int)` 切换码率。切换过程中，会重新拉取另一条流的数据，SDK 针对腾讯云的多码率文件做过优化，可以做到切换无卡顿。

如果您提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率，从而避免播放后切换码流。详细方法参见 播放器配置#启播前指定分辨率。

9、码流自适应

SDK 支持 HLS 的多码流自适应，开启相关能力后播放器能够根据当前带宽，动态选择最合适的码率播放。可以通过下面方法开启码流自适应。

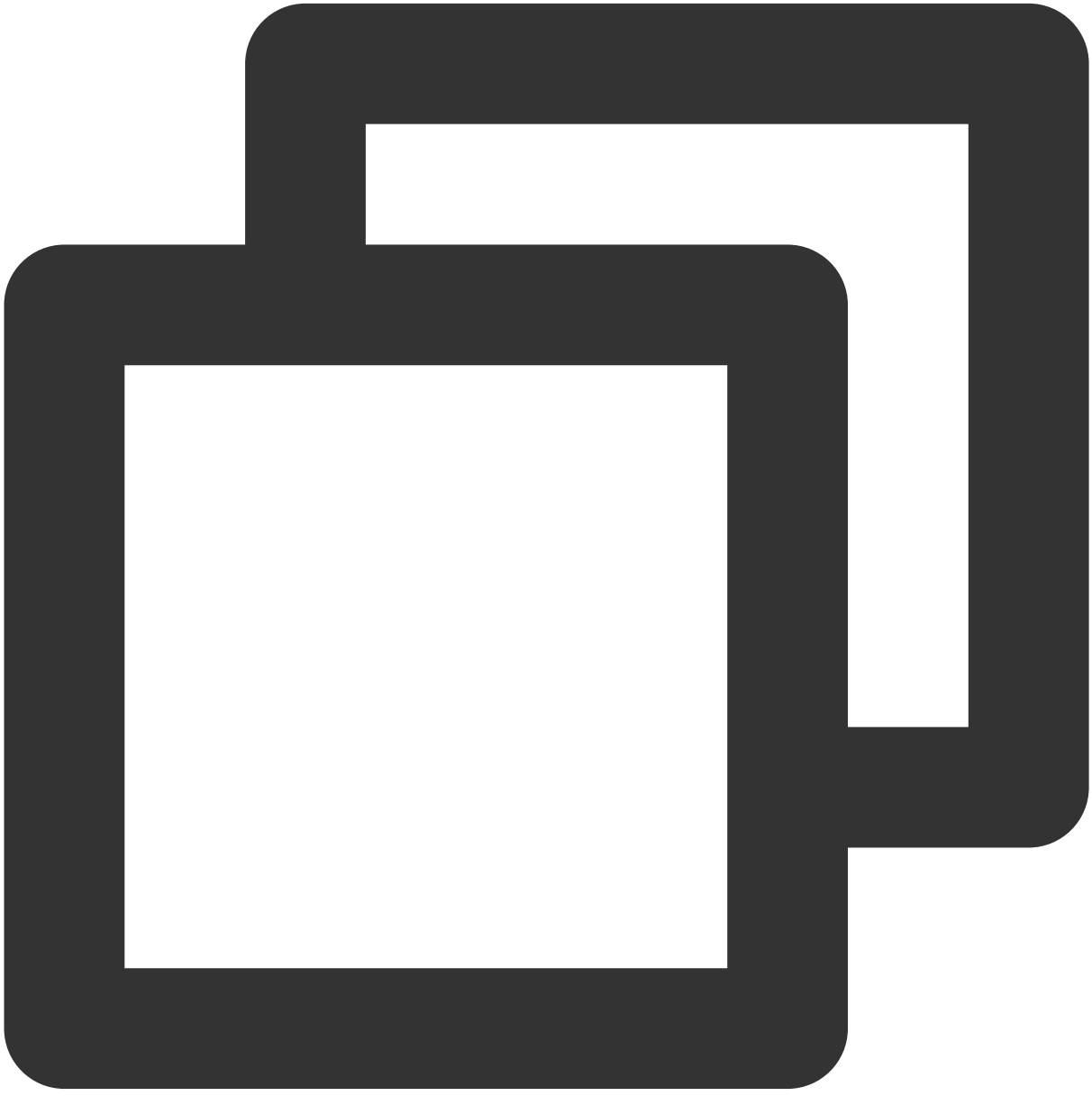


```
_controller.setBitrateIndex(-1); //index 参数传入-1
```

在播放过程中，可以随时通过 `_controller.setBitrateIndex(int)` 切换其它码率，切换后码流自适应也随之关闭。

10、开启平滑切换码率

在启动播放前，通过开启平滑切换码率，在播放过程中切换码率，可以达到无缝平滑切换不同清晰度。对比关闭平滑切换码率，切换过程比较耗时、体验更好，可以根据需求进行设置。



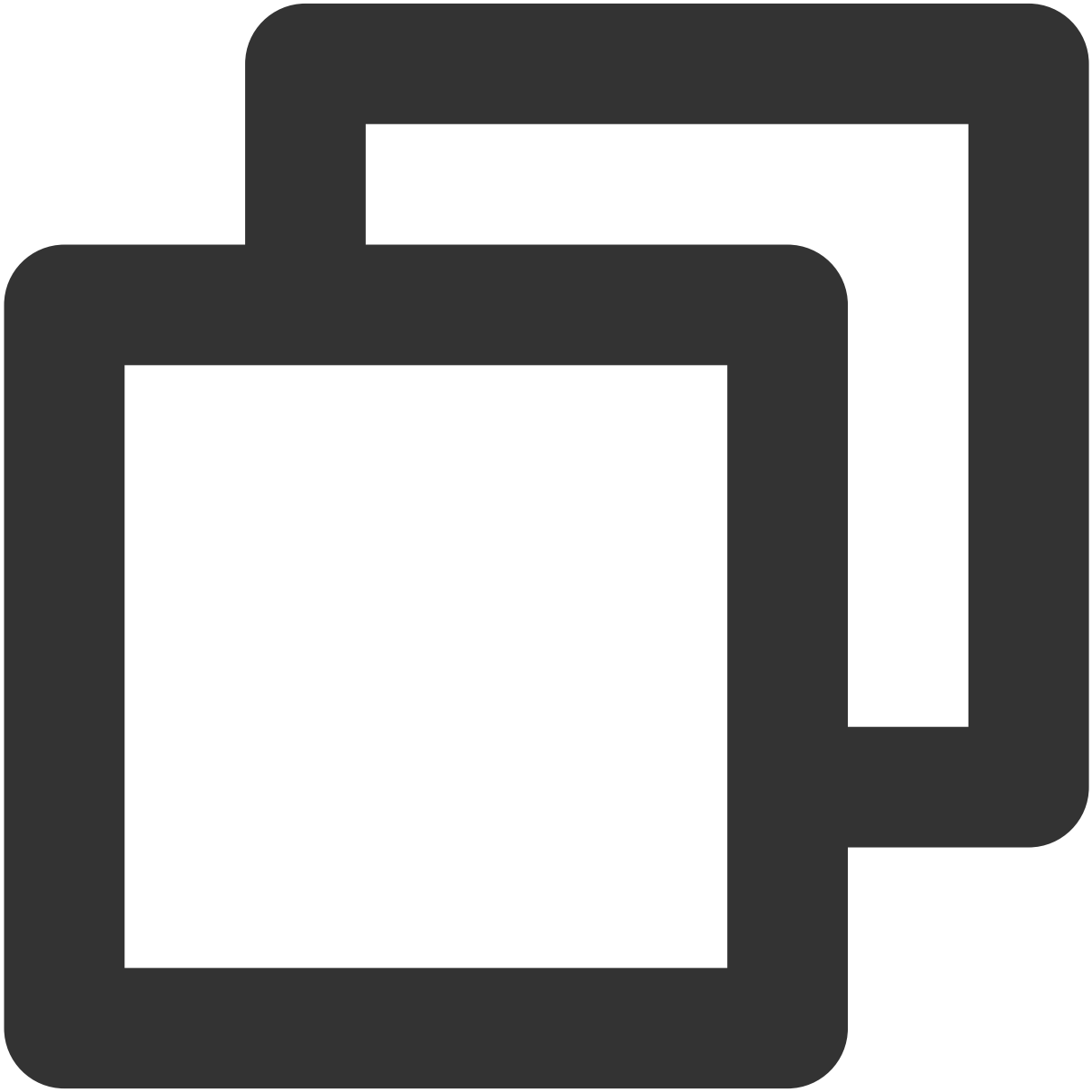
```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
/// 设为true，可平滑切换码率，设为false时，可提高多码率地址打开速度
playConfig.smoothSwitchBitrate = true;
```

```
_controller.setConfig(playConfig);
```

11、播放进度监听

点播播放中的进度信息分为2种：**加载进度**和**播放进度**，SDK 目前是以事件通知的方式将这两个进度实时通知出来的。

通过 `onPlayerEventBroadcast` 接口监听播放器事件，进度通知会通过 **PLAY_EVT_PLAY_PROGRESS** 事件回调到您的应用程序。

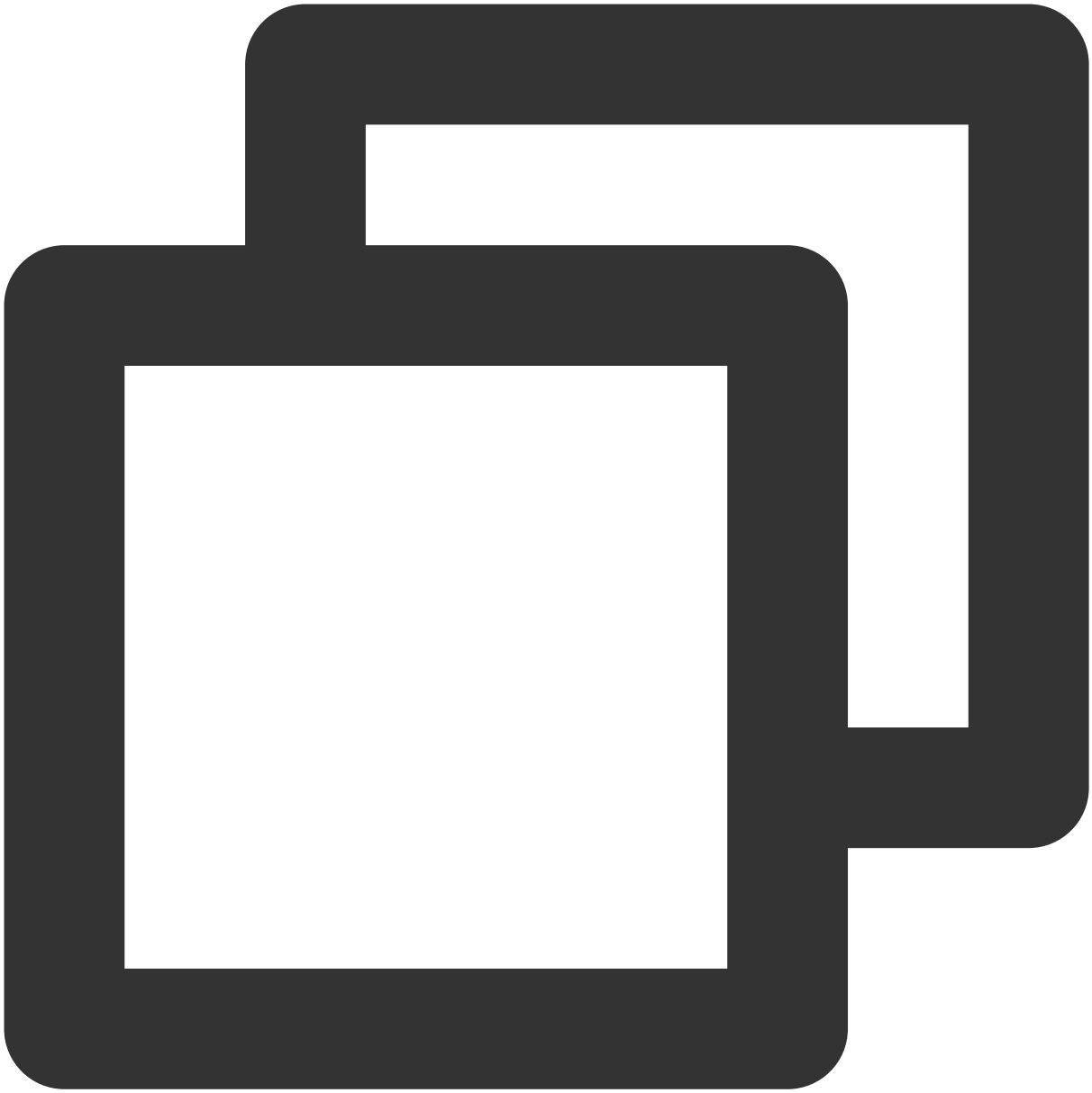


```
_controller.onPlayerEventBroadcast.listen((event) async {
```

```
    if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {// 更多详细请查看iOS或者
      // 可播放时长，即加载进度，单位是毫秒
      double playableDuration = event[TXVodPlayEvent.EVT_PLAYABLE_DURATION_MS].toDoub
      // 播放进度，单位是秒
      int progress = event[TXVodPlayEvent.EVT_PLAY_PROGRESS].toInt();
      // 视频总长，单位是秒
      int duration = event[TXVodPlayEvent.EVT_PLAY_DURATION].toInt();
    }
  });
```

12、播放网速监听

通过 `onPlayerNetStatusBroadcast` 接口监听播放器的网络状态，如： `NET_STATUS_NET_SPEED` 。



```
_controller.onPlayerNetStatusBroadcast.listen((event) {
    (event[TXVodNetEvent.NET_STATUS_NET_SPEED]).toDouble();
});
```

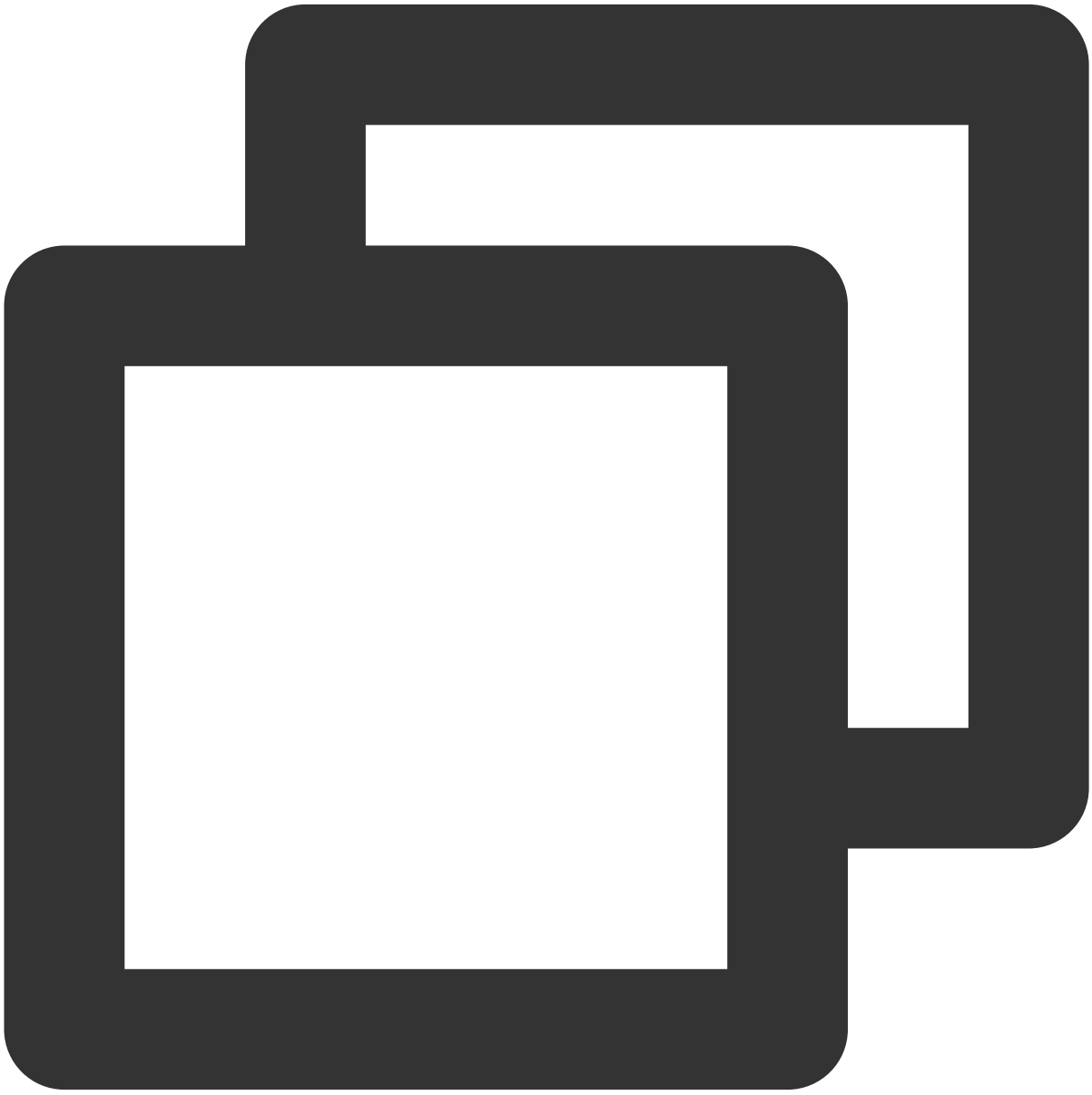
13、获取视频分辨率

播放器 SDK 通过 URL 字符串播放视频，URL 中本身不包含视频信息。为获取相关信息，需要通过访问云端服务器加载到相关视频信息，因此 SDK 只能以事件通知的方式将视频信息发送到您的应用程序中.

可以通过下面两种方法获取分辨率信息

方法1：通过 `onPlayerNetStatusBroadcast` 的 `NET_STATUS_VIDEO_WIDTH` 和 `NET_STATUS_VIDEO_HEIGHT` 获取视频的宽和高

方法2：在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后，直接调用 `getWidth()` 和 `getHeight()` 获取当前宽高。

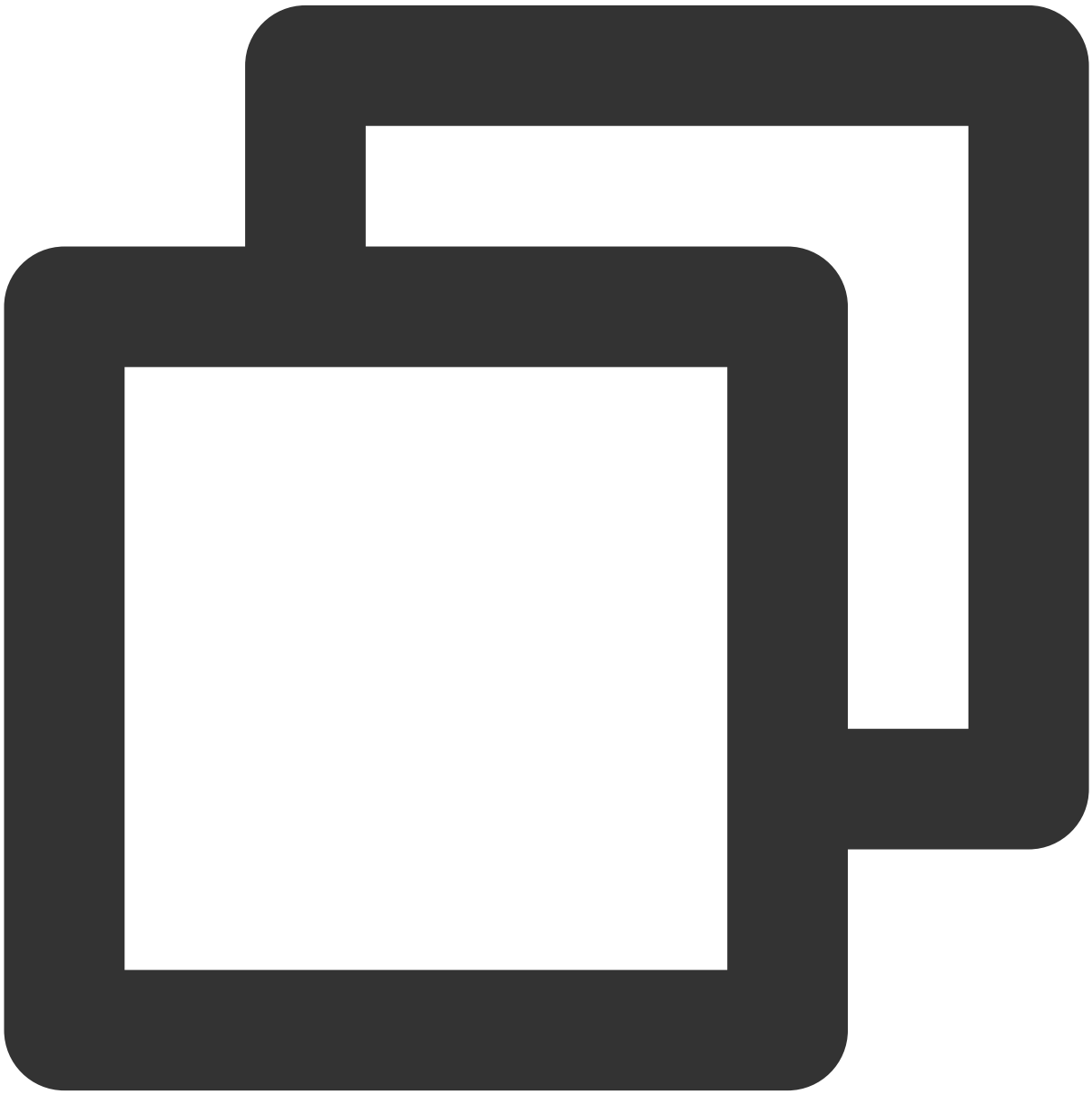


```
 _controller.onPlayerNetStatusBroadcast.listen((event) {
  double w = (event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH]).toDouble();
  double h = (event[TXVodNetEvent.NET_STATUS_VIDEO_HEIGHT]).toDouble();
});
```

```
// 获取视频宽高，需要在收到播放器的PLAY_EVT_VOD_PLAY_PREPARED 事件回调后才返回值
_controller.getWidth();
_controller.getHeight();
```

14、播放缓冲大小

在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
```

```
playConfig.maxBufferSize = 10;  ///  播放时最大缓冲大小。单位：MB
_controller.setConfig(playConfig);
```

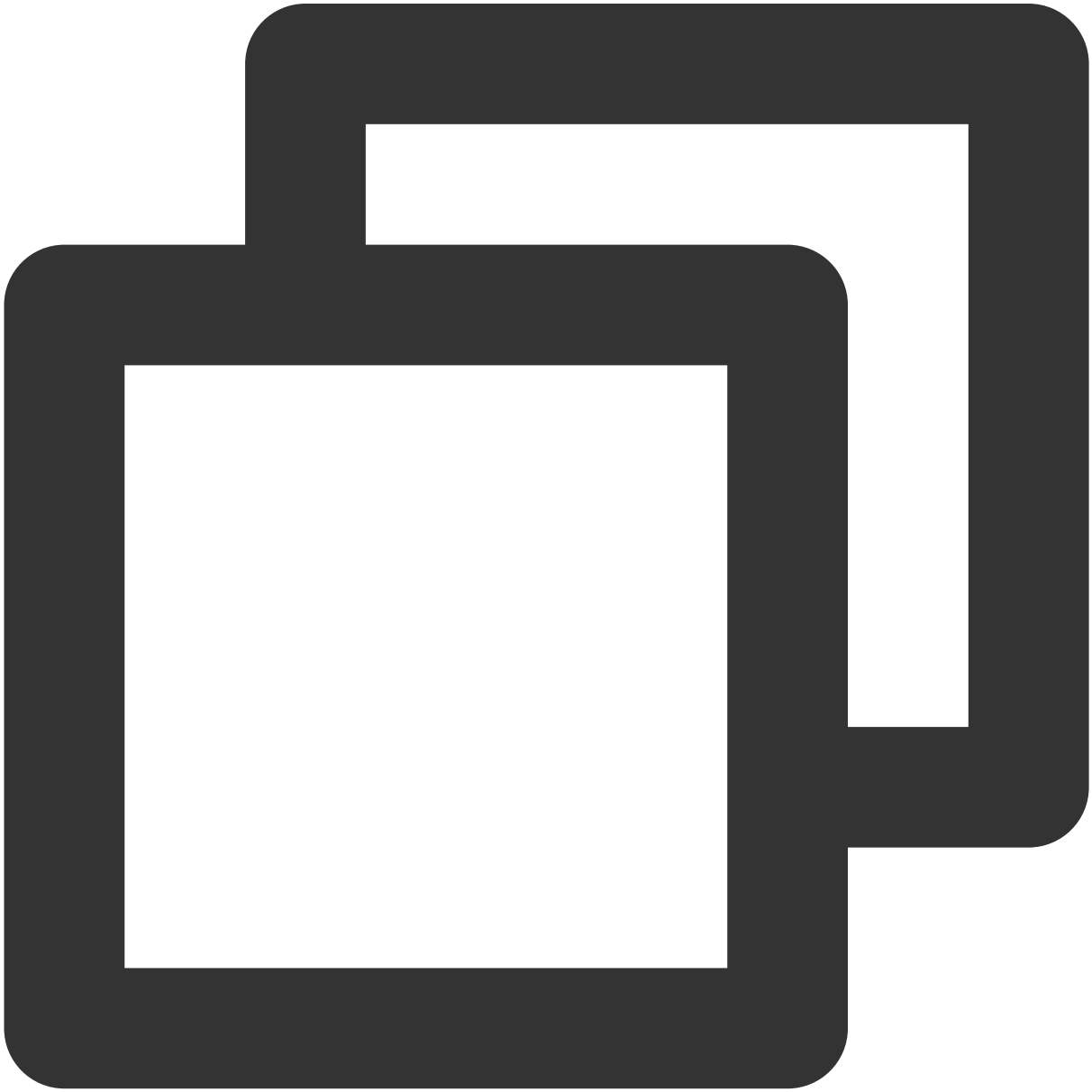
15、视频本地缓存

在短视频播放场景中，视频文件的本地缓存是很刚需的一个特性，对于普通用户而言，一个已经看过的视频再次观看时，不应该再消耗一次流量。

格式支持：SDK 支持 HLS(m3u8) 和 MP4 两种常见点播格式的缓存功能。

开启时机：SDK 并不默认开启缓存功能，对于用户回看率不高的场景，也并不推荐您开启此功能。

开启方式：全局生效，在使用播放器开启。开启此功能需要配置两个参数：本地缓存目录及缓存大小。



```
//设置播放引擎的全局缓存目录和缓存大小，//单位MB
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
//设置播放引擎的全局缓存目录
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

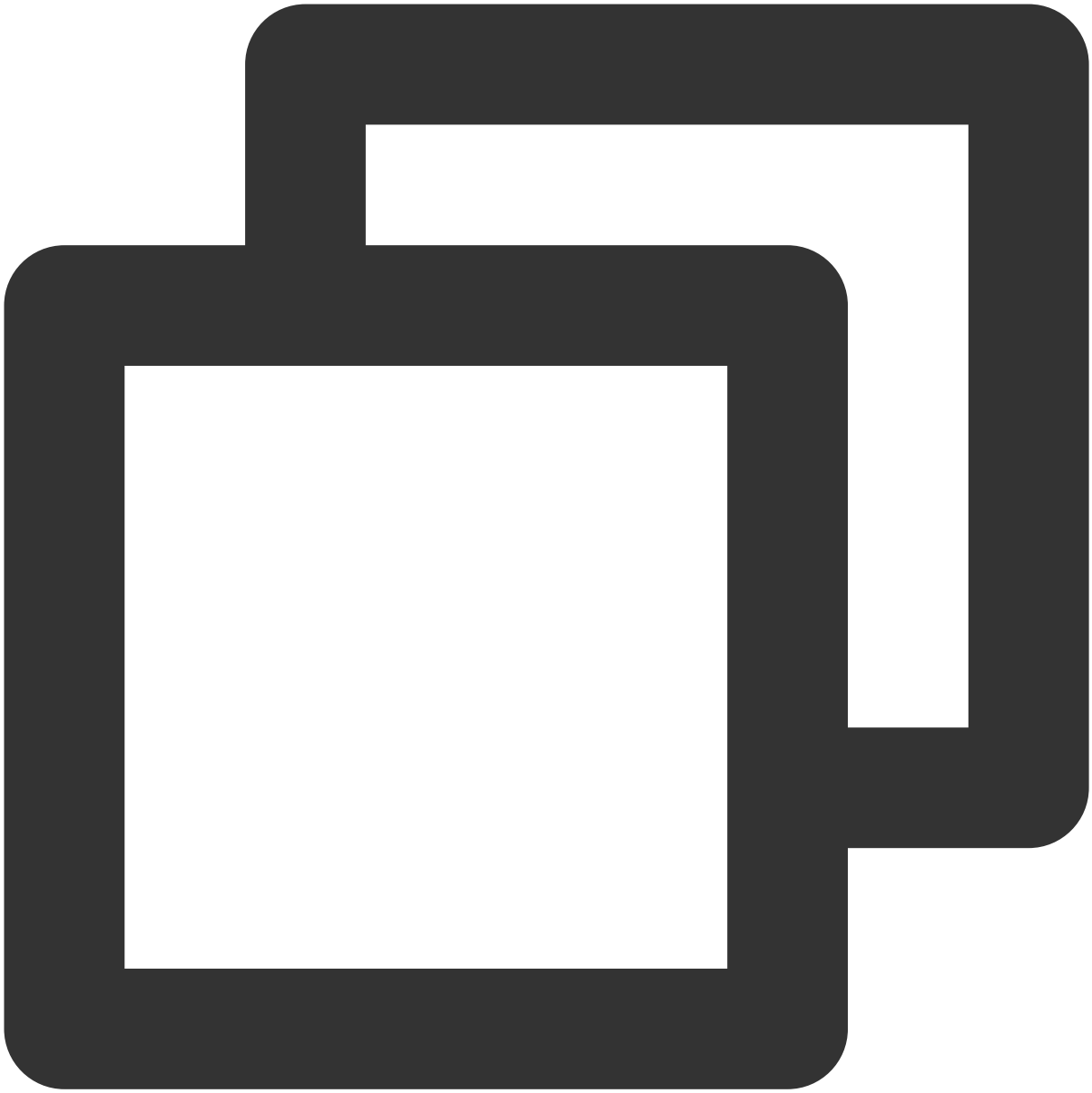
16、外挂字幕

注意：此功能需要播放器高级版 11.7 版本开始支持。

播放器高级版 SDK 支持添加和切换外挂字幕，现已支持 SRT 和 VTT 这两种格式的字幕。

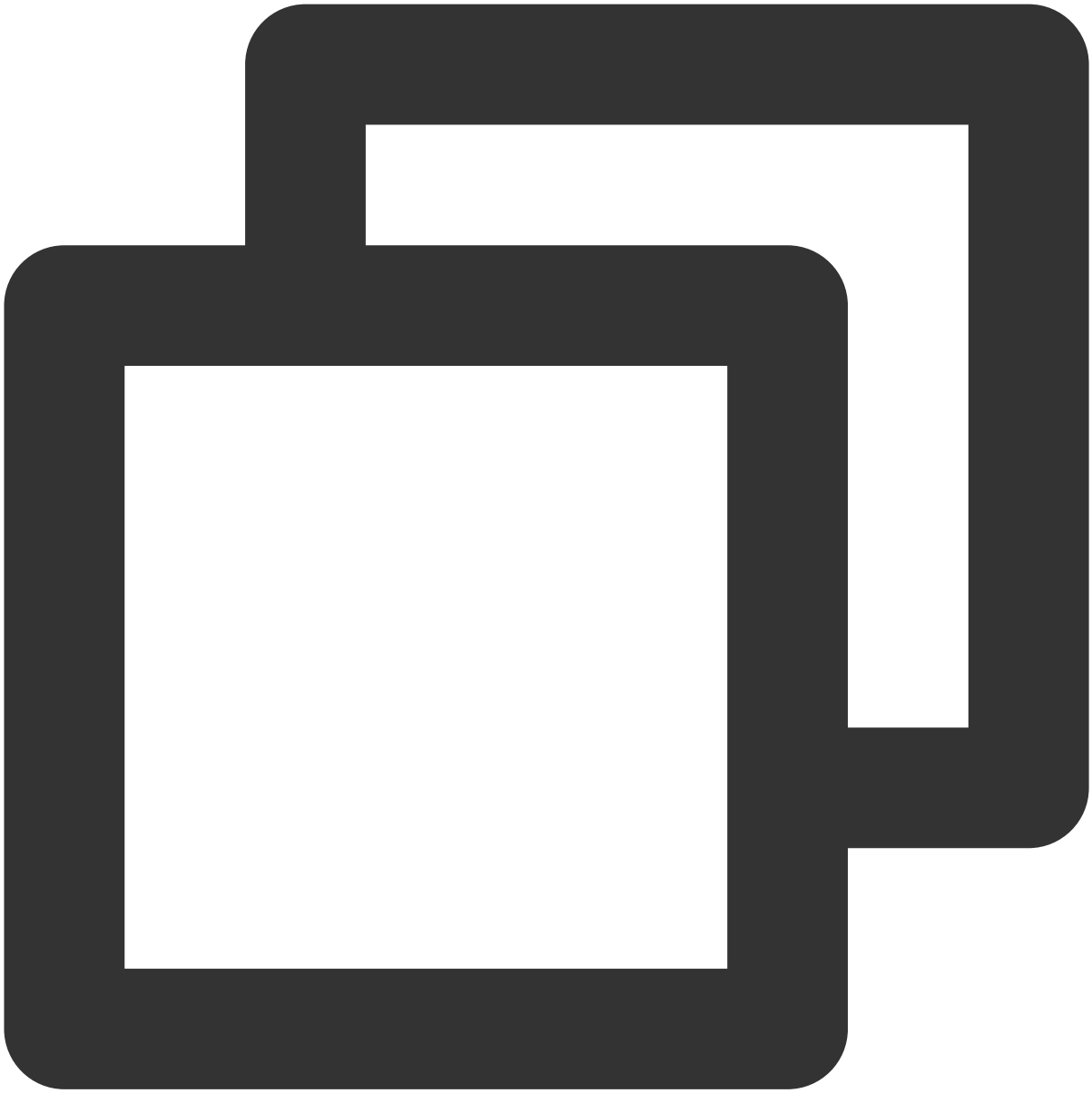
最佳实践：建议在 `startVodPlay` 之前添加字幕和配置字幕样式，在收到 `VOD_PLAY_EVT_VOD_PLAY_PREPARED` 事件后，调用 selectTrack 选择字幕。 添加字幕后，并不会主动加载字幕， 调用 `selectTrack` 后，才会加载字幕，字幕选择成功会有 `VOD_PLAY_EVT_SELECT_TRACK_COMPLETE` 事件回调。选择字幕后，字幕文本内容会通过 `TXVodPlayEvent.EVENT_SUBTITLE_DATA` 。事件回调，字幕的显示在需要业务方自行处理。

步骤 1：添加外挂字幕



```
// 添加外挂字幕，传入 字幕url， 字幕名称， 字幕类型， 建议在启动播放器前添加
controller.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-cloud.com/De

// 开始播放视频后，监听字幕文本内容回调
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
  // 字幕文本内容，可用于显示
  String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

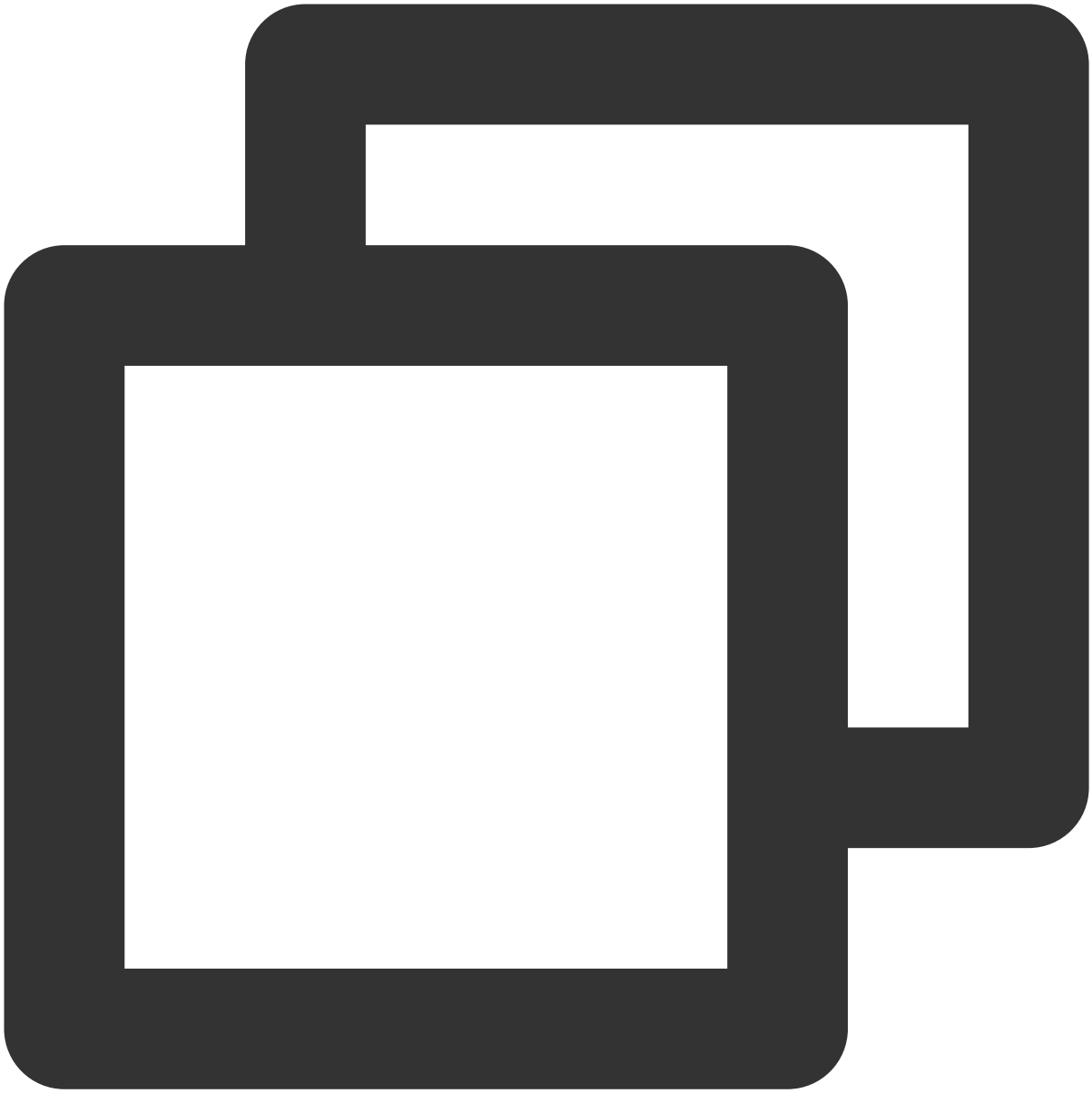
步骤2：播放后切换字幕



```
// 开始播放视频后，选中添加的外挂字幕， 在收到 VOD_PLAY_EVT_VOD_PLAY_PREPARED 事件后调用
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.PLAY_EVT_VOD_PLAY_PREPARED) {
    List<TXTrackInfo> trackInfoList = await _vodPlayerController.getSubtitleTrackIn
      for (TXTrackInfo tempInfo in trackInfoList) {
        if(tempInfo.name == "subtitleName") {
          // 选中字幕
          _vodPlayerController.selectTrack(tempInfo.trackIndex);
        } else {
```

```
            // 其它字幕不需要的话，  进行deselectTrack
            _vodPlayerController.deselectTrack(tempInfo.trackIndex);
          }
        }
      }
});

// 如果需要，可以监听轨道切换消息
_controller.onPlayerEventBroadcast.listen((event) async {
  if(event["event"] == TXVodPlayEvent.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
    int trackIndex = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_INDEX];
    int errorCode = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_ERROR_CODE];
  }
});
```

步骤3：监听字幕文本内容

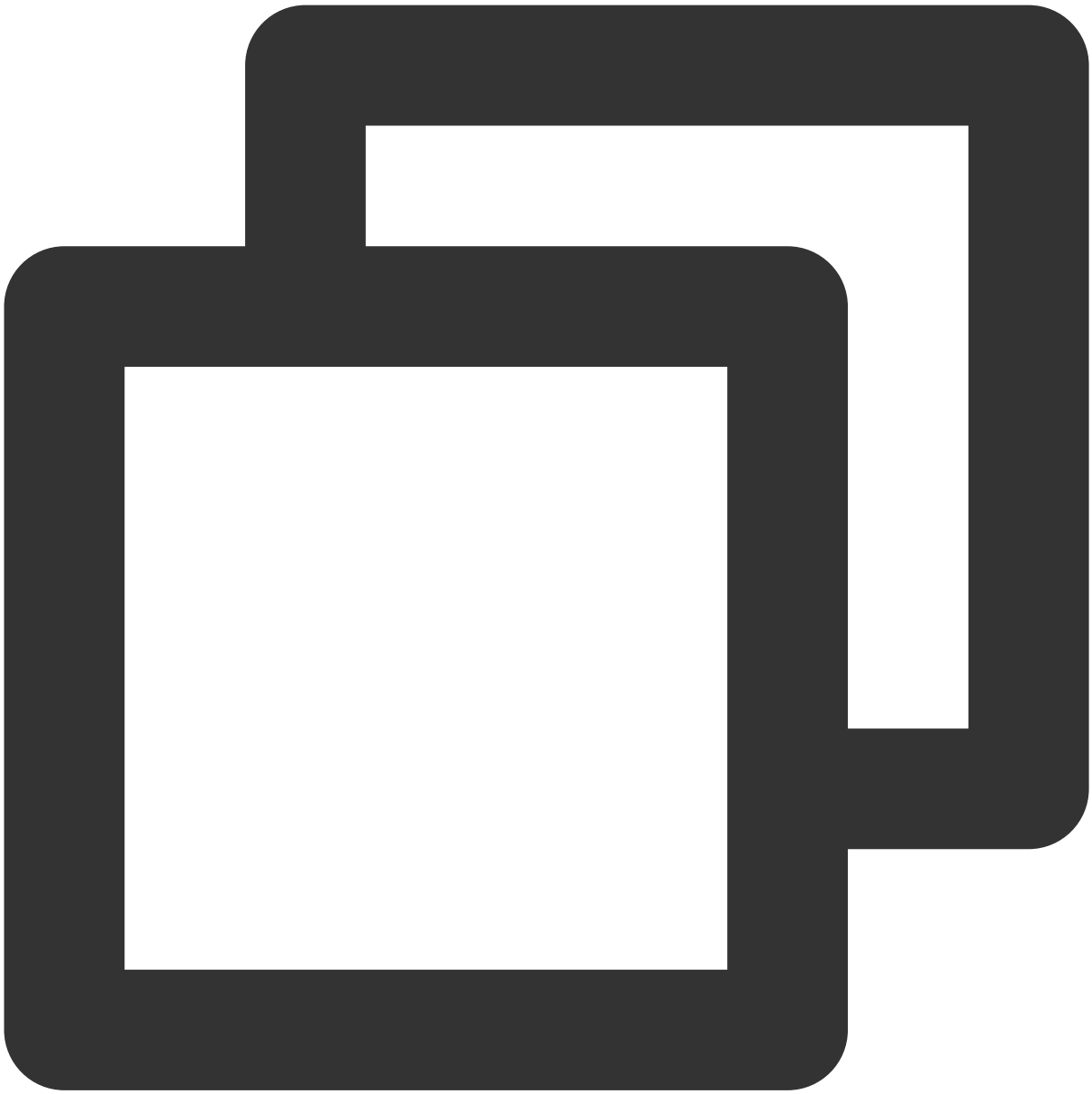


```
// 开始播放视频后，监听字幕文本内容回调
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
  // 字幕文本内容，可用于显示
  String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

17、多音轨切换

注意：此功能需要播放器高级版 11.7 版本开始支持。

播放器高级版 SDK 支持切换视频内置的多音轨。用法参见如下代码：



```
// 返回音频轨道信息列表
List<TXTrackInfo> trackInfoList = await _vodPlayerController.getAudioTrackInfo();
for (TXTrackInfo tempInfo in trackInfoList) {
  if(tempInfo.trackIndex == 0) {
    // 通过判断 trackIndex 或者 name 切换到需要的音轨
    _vodPlayerController.selectTrack(tempInfo.trackIndex);
  } else {
    // 不需要的音轨进行 deselectTrack
```

```
        _vodPlayerController.deselectTrack(tempInfo.trackIndex);
    }
}
```

进阶功能使用

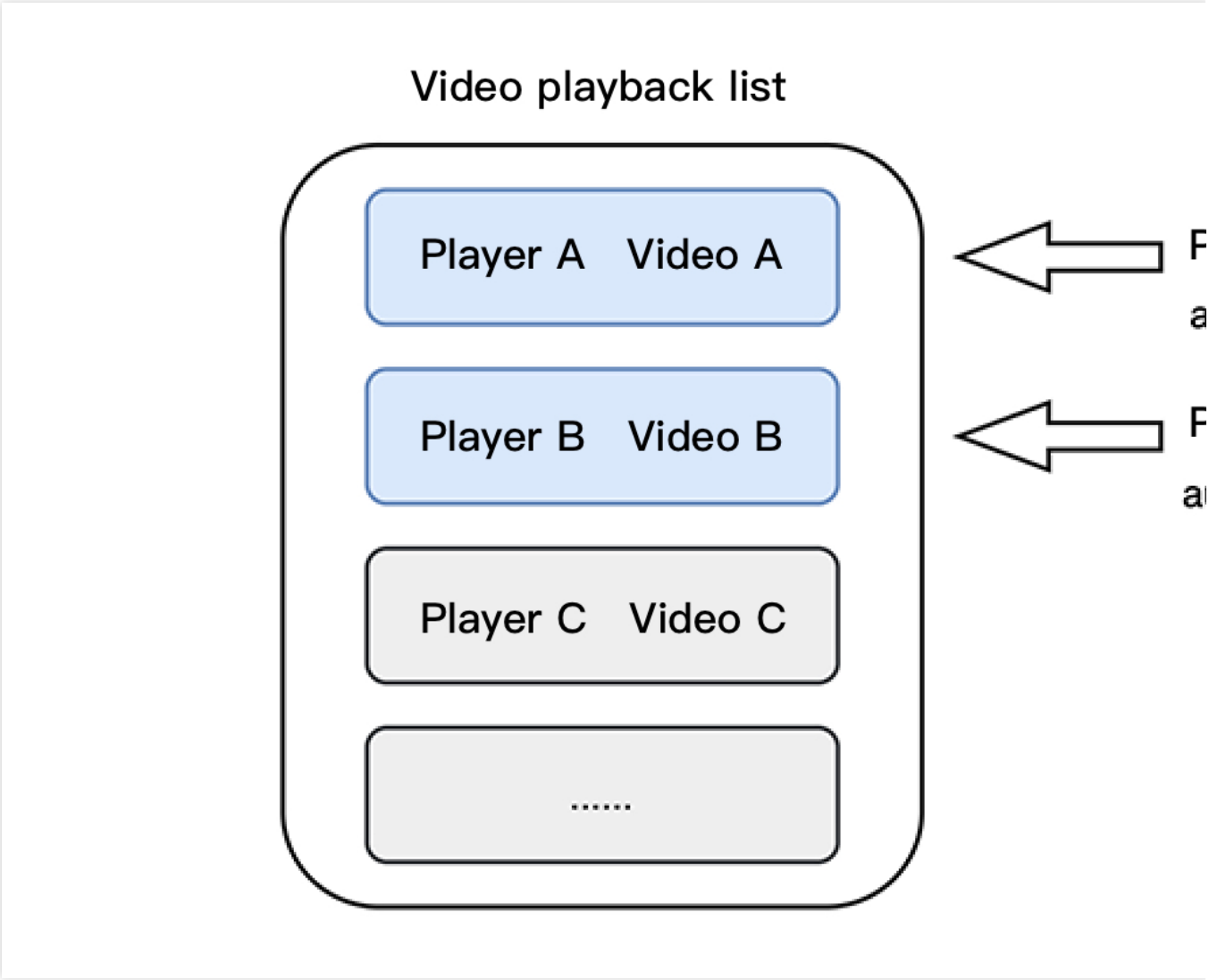
1、视频预播放

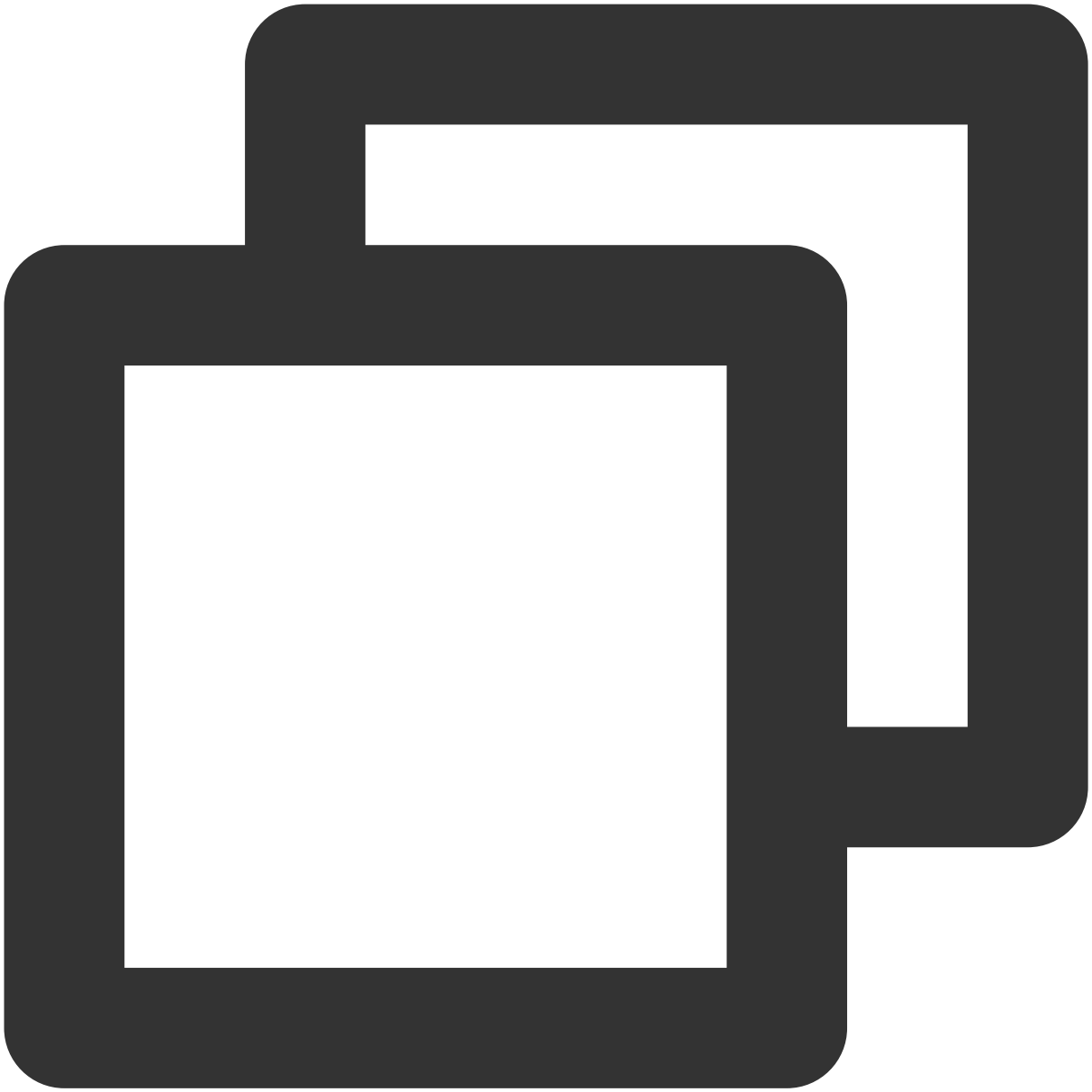
步骤1：视频预播放使用

在短视频播放场景中，视频预播放功能对于流畅的观看体验很有帮助：在观看当前视频的同时，在后台加载即将要播放的下一个视频，这样一来，当用户真正切换到下一个视频时，已经不需要从头开始加载了，而是可以做到立刻播放。

预播放视频会有很好的秒开效果，但有一定的性能开销，如果业务同时有较多的视频预加载需求，建议结合 视频预下载 一起使用。

这就是视频播放中无缝切换的背后技术支撑，您可以使用 TXVodPlayerController 中的 setAutoPlay 开关来实现这个功能，具体做法如下：


Video playback list
Player A Video A
Player B Video B
Player C Video C
.....

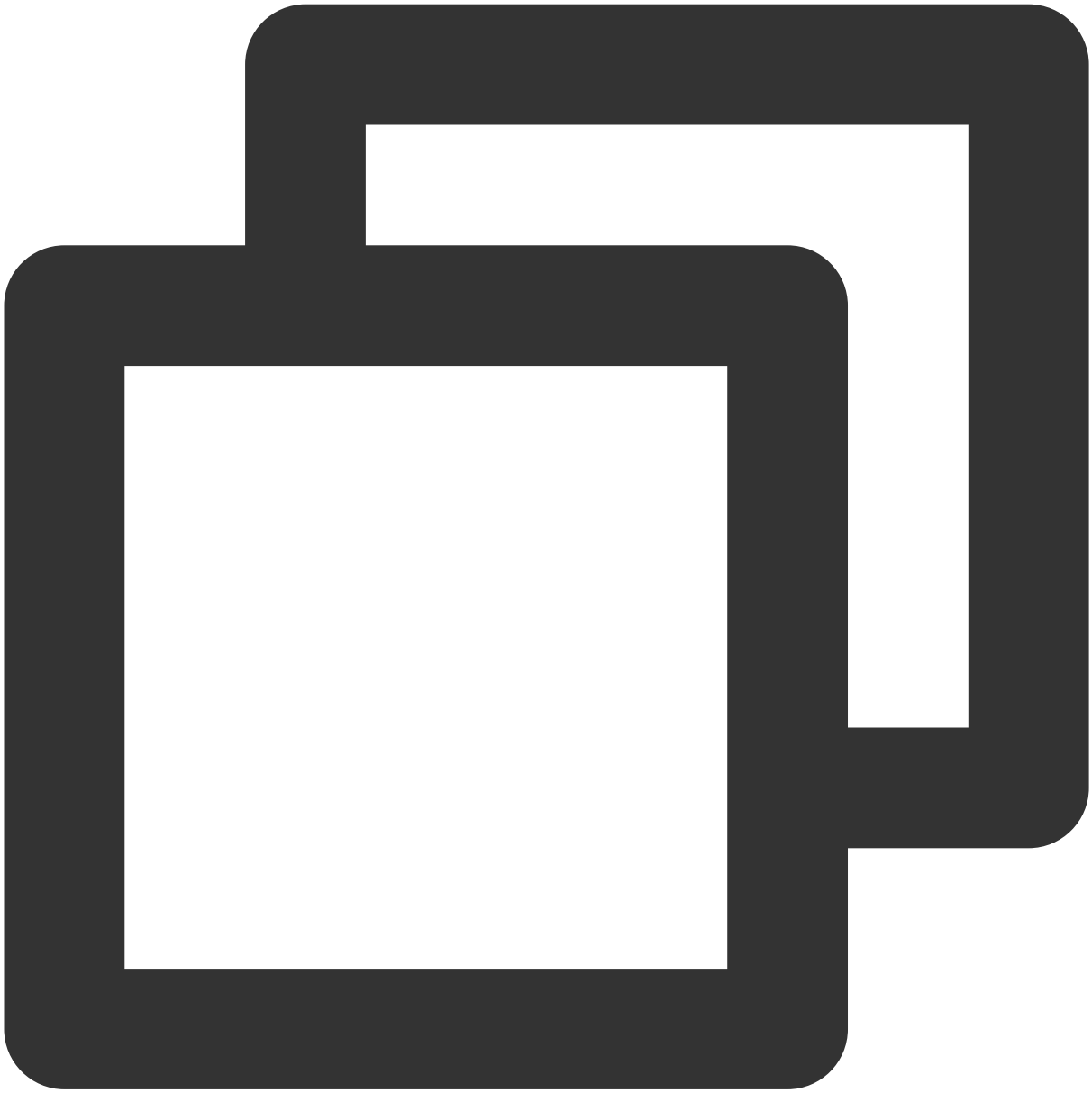


```
// 播放视频 A：如果将 autoPlay 设置为 true， 那么 startVodPlay 调用会立刻开始视频的加载和播放
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
controller.setAutoPlay(isAutoPlay: true);
controller.startVodPlay(urlA);

// 在播放视频 A 的同时，预加载视频 B，做法是将 setAutoPlay 设置为 false
String urlB = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
controller.setAutoPlay(isAutoPlay: false);
controller.startVodPlay(urlB); // 不会立刻开始播放，而只会开始加载视频
```

等到视频 A 播放结束，自动（或者用户手动切换到）视频 B 时，调用 resume 函数即可实现立刻播放。

注意：

设置了 autoPlay 为 false 之后，调用 resume 之前需要保证视频 B 已准备完成，即需要在监听到视频 B 的 PLAY_EVT_VOD_PLAY_PREPARED （2013，播放器已准备完成，可以播放）事件后调用。



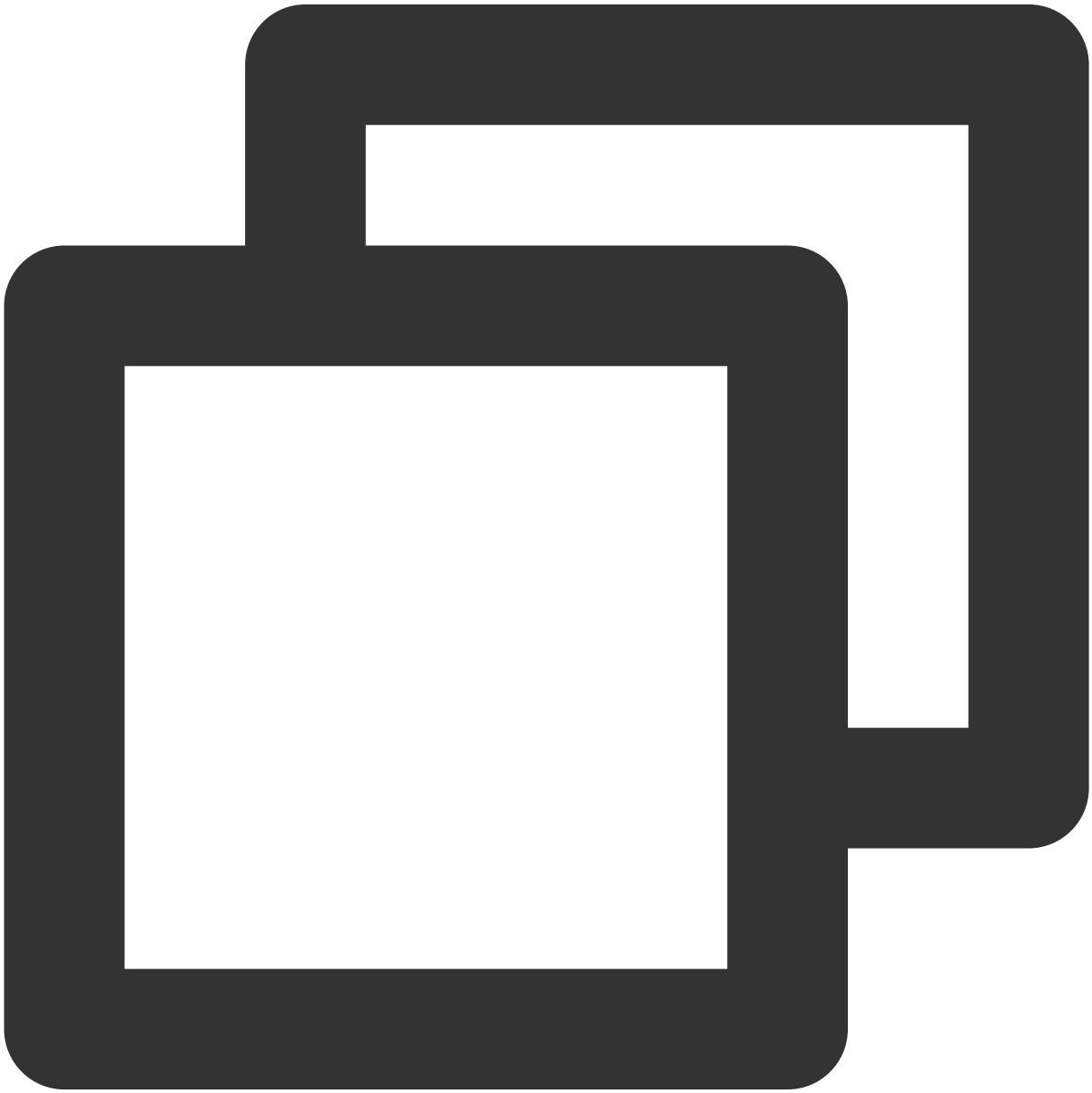
```
controller.onPlayerEventBroadcast.listen((event) async {//订阅状态变化
  if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_END) {
    await _controller_A.stop();
    await _controller_B.resume();
  }
});
```

步骤2：视频预播放缓冲配置

设置较大的缓冲可以更好的应对网络的波动，达到流畅播放的目的。
设置较小的缓冲可以帮助节省流量消耗。

预播放缓冲大小

此接口针对预加载场景（即在视频启播前，且设置 player 的 AutoPlay 为 false），用于控制启播前阶段的最大缓冲大小。

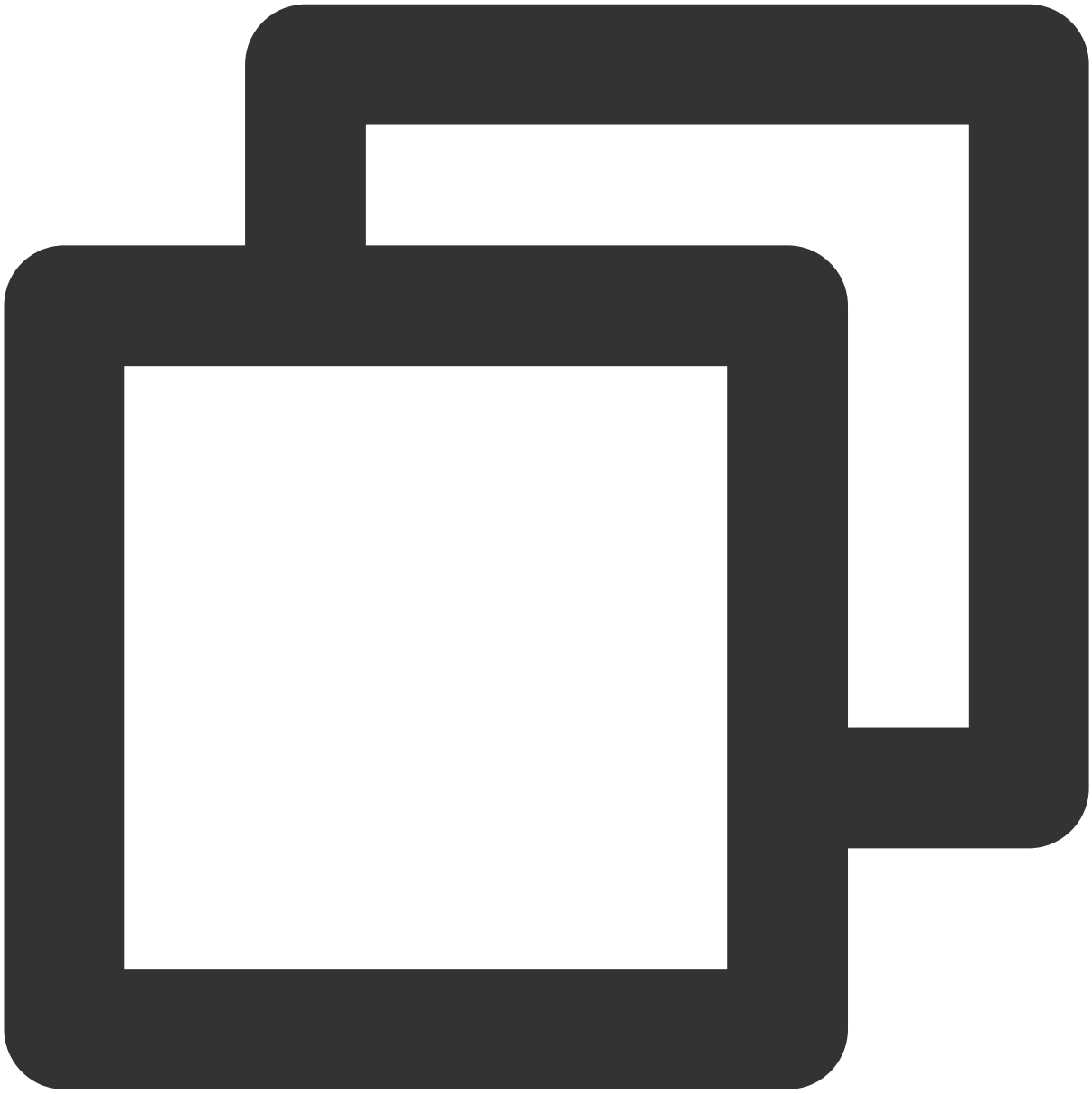


```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxPreloadSize(2);   // 预播放最大缓冲大小。单位：MB，根据业务情况设置去节省流量
```

```
mVodPlayer.setConfig(config);  // 把config 传给 mVodPlayer
```

播放缓冲大小

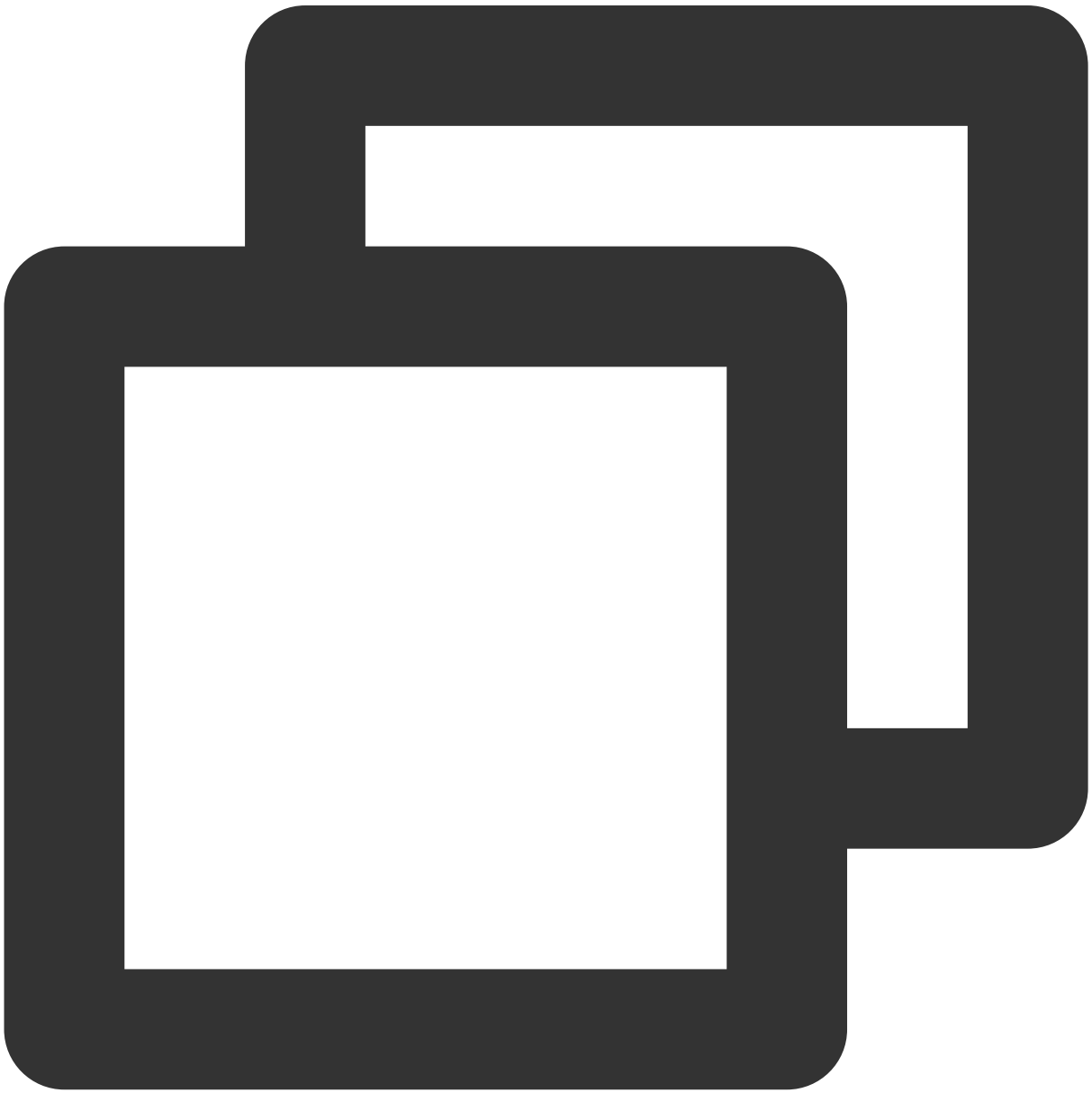
在视频正常播放时，控制提前从网络缓冲的最大数据大小。如果不配置，则走播放器默认缓冲策略，保证流畅播放。



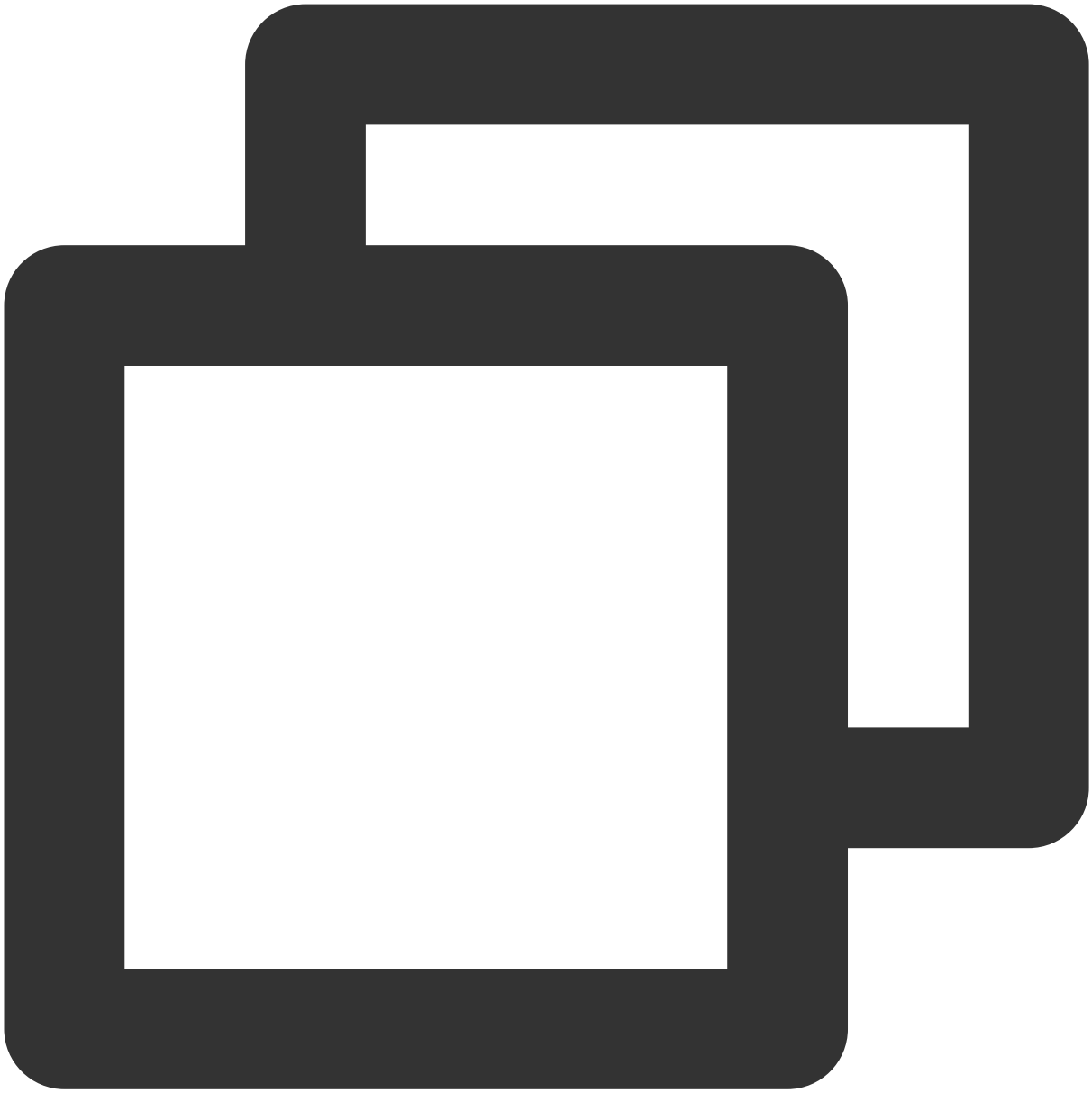
```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.maxBufferSize = 10; // 播放时最大缓冲大小。单位：MB
_controller.setPlayConfig(config); // 把config 传给 controller
```

2、视频预下载

不需要创建播放器实例，预先下载视频部分内容，使用播放器时，可以加快视频启播速度，提供更好的播放体验。
在使用播放服务前，请确保先设置好 视频缓存。
使用示例：
说明：
1. `TXPlayerGlobalSetting` 是全局缓存设置API，原 `TXVodConfig` API 已被弃用。
2. 全局缓存目录和大小设置的优先级高于播放器中 `TXVodConfig` 中的配置。
通过媒资URL预下载
通过媒资FileId预下载



```
//设置播放引擎的全局缓存目录和缓存大小
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache 目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

String palyrl = "http://****";
//启动预下载
int taskId = await TXVodDownloadController.instance.startPreLoad(palyrl, 3, 1920*10
  onCompleteListener:(int taskId,String url) {
    print('taskID=${taskId} ,url=${url}');
  }, onErrorListener: (int taskId, String url, int code, String msg) {
    print('taskID=${taskId} ,url=${url}, code=${code} , msg=${msg}');
  }
);

//取消预下载
TXVodDownloadController.instance.stopPreLoad(taskId);
```



```
//设置播放引擎的全局缓存目录和缓存大小
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache 目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

int retTaskId = -1;
TXVodDownloadController.instance.startPreload(TXPlayInfoParams(appId: 0, fileId: "y
    onStartListener: (taskId, fileId, url, params) {
      // TXVodDownloadController will call this block for callback taskId and video
```

```
        retTaskId = taskId;
      },
      onCompleteListener: (taskId, url) {
        // preDownload complete
      },
      onErrorListener: (taskId, url, code, msg) {
        // preDownload error
      });

  //取消预下载
  TXVodDownloadController.instance.stopPreLoad(retTaskId);
```

3、视频下载

视频下载支持用户在有网络的条件下下载视频，随后在无网络的环境下观看。同时播放器 SDK 提供本地加密能力，下载后的本地视频仍为加密状态，仅可通过指定播放器对视频进行解密播放，可有效防止下载后视频的非法传播，保护视频安全。

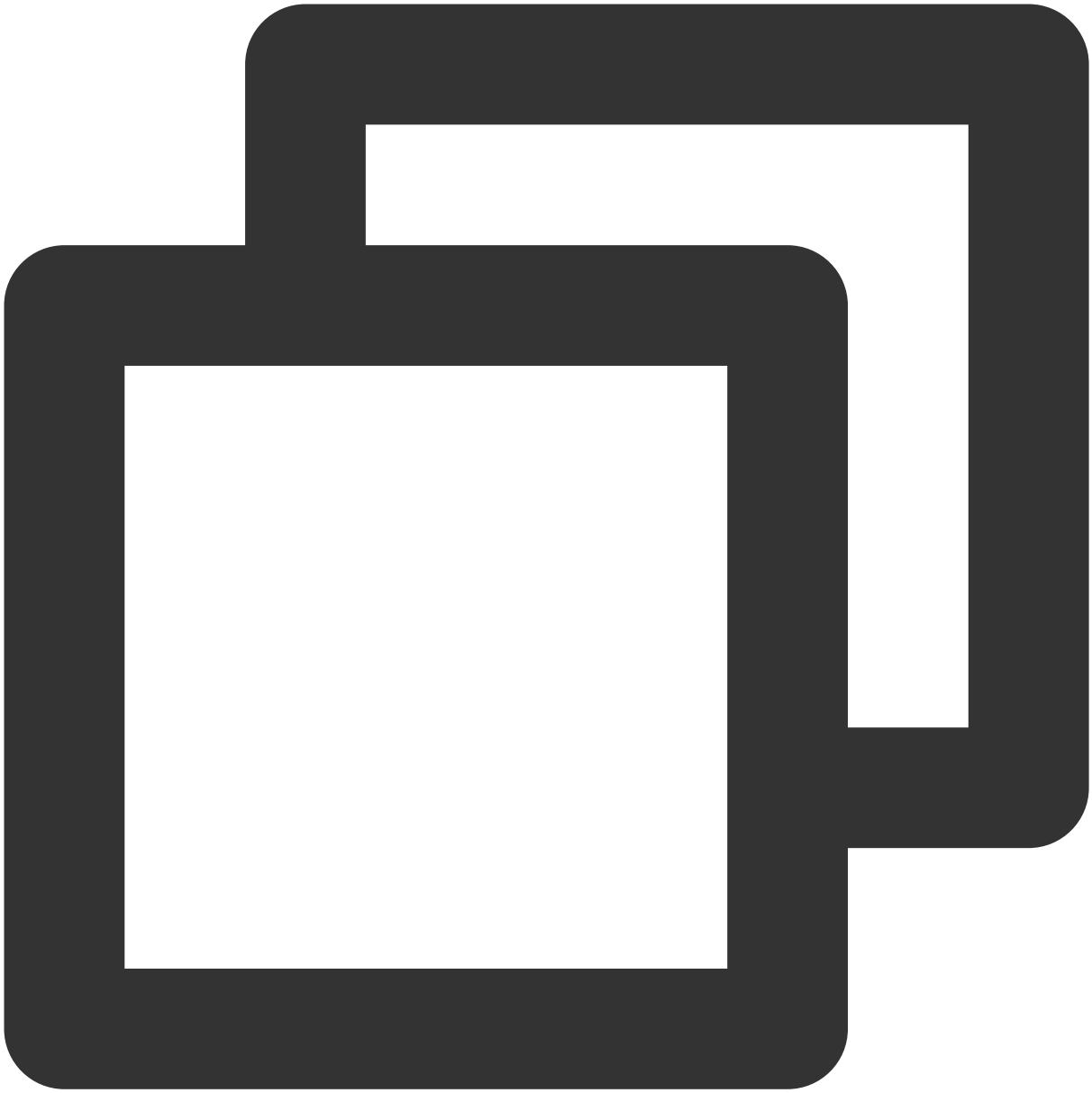
由于 HLS 流媒体无法直接保存到本地，因此也无法通过播放本地文件的方式实现 HLS 下载到本地后播放，对于该问题，您可以通过基于 `TXVodDownloadController` 的视频下载方案实现 HLS 的离线播放。

说明：

`TXVodDownloadController` 暂不支持缓存 MP4 和 FLV 格式的文件，仅支持缓存非嵌套 HLS 格式文件。播放器 SDK 已支持 MP4 和 FLV 格式的本地文件播放。

步骤1：准备工作

`TXVodDownloadController` 被设计为单例，因此您不能创建多个下载对象。用法如下：



```
// 该缓存路径默认设置到app沙盒目录下，postfixPath只需要传递相对缓存目录即可，不需要传递整个绝对路
// Android 平台：视频将会缓存到sdcard的Android/data/your-pkg-name/files/testCache 目录。
// iOS 平台：视频将会缓存到沙盒的Documents/testCache 目录。
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

步骤2: 开始下载

开始下载有 Fileid 和 URL 两种方式，具体操作如下：

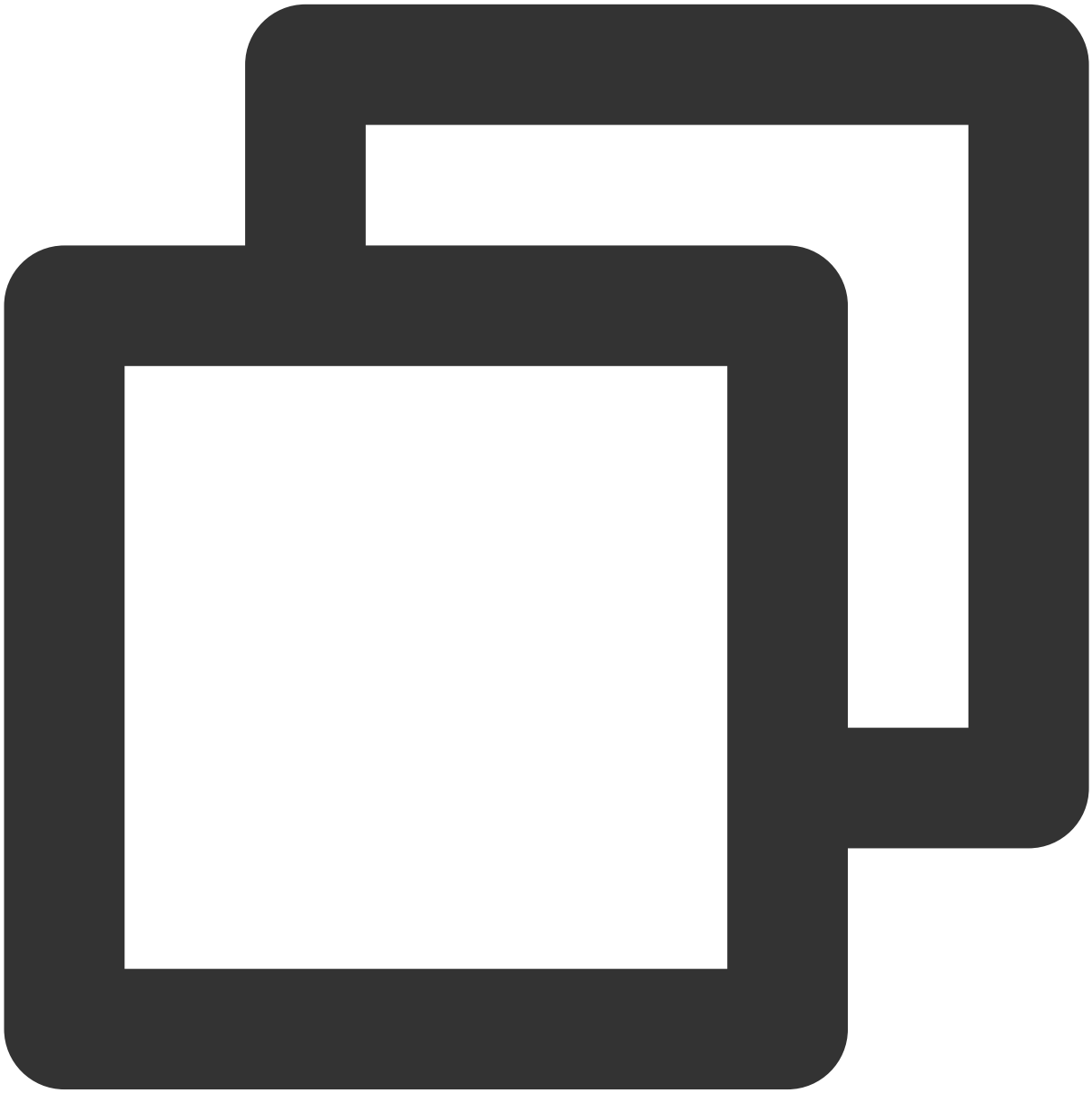
Fileid 方式

URL 方式

Fileid 下载至少需要传入 AppID、 Fileid 和 qualityId。带签名视频需传入 pSign，userName 不传入具体值时，默认为“default”。

注意：

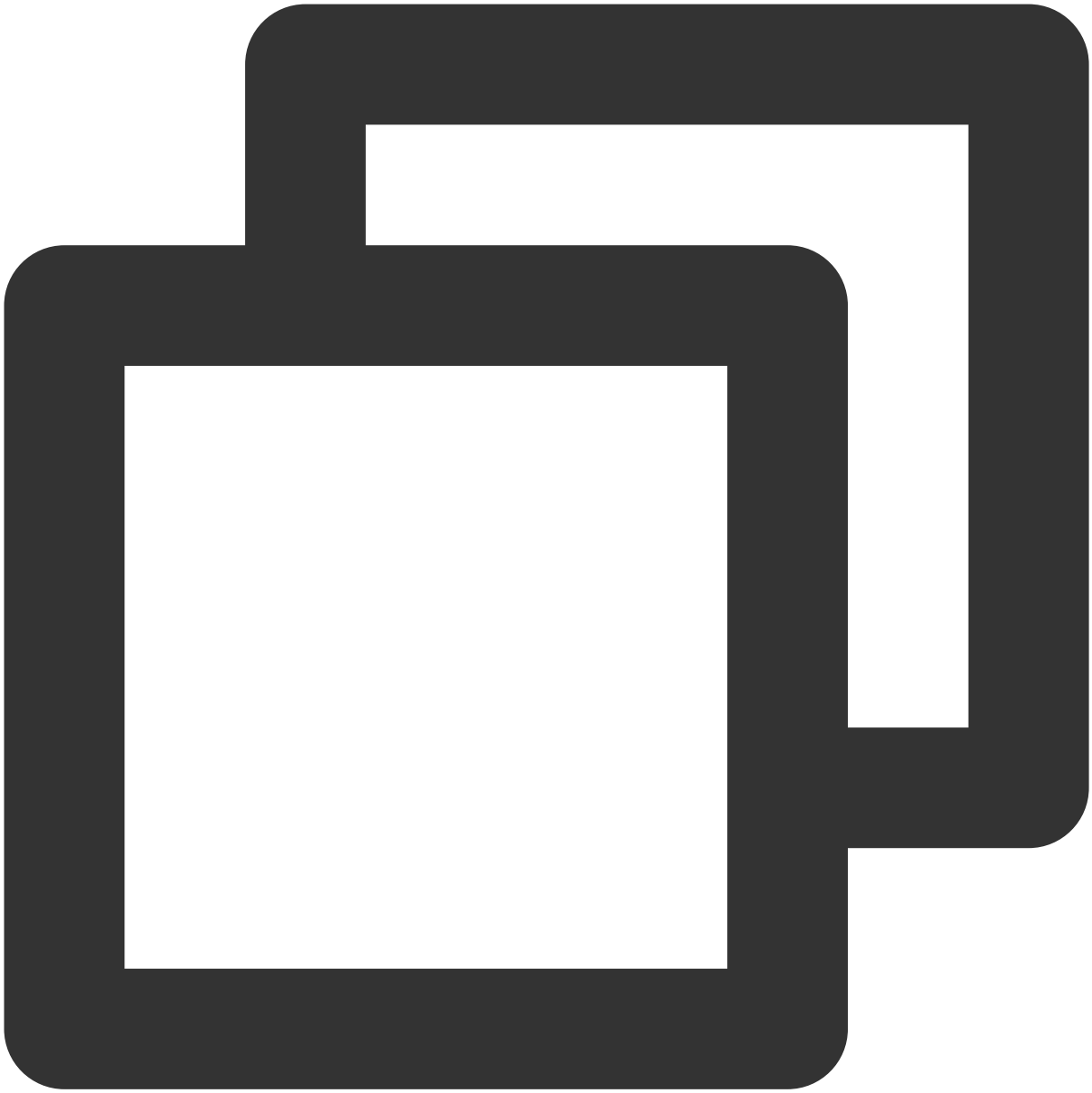
加密视频只能通过 Fileid 下载，psign 参数必须填写。



```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
```

```
// QUALITY_2K    2k
// QUALITY_4K    4k
// quality参数可以自定义，取分辨率宽高最小值(如分辨率为1280*720，期望下载此分辨率的流，quality
// 播放器 SDK 会选择小于或等于传入分辨率的流进行下载
TXVodDownloadMedialnfo medialnfo = TXVodDownloadMedialnfo();
TXVodDownloadDataSource dataSource = TXVodDownloadDataSource();
dataSource.appId = 1252463788;
dataSource.fileId = "4564972819220421305";
dataSource.quality = DownloadQuality.QUALITY_480P;
dataSource.pSign = "pSignxxxx";
medialnfo.dataSource = dataSource;
TXVodDownloadController.instance.startDonwload(medialnfo);
```

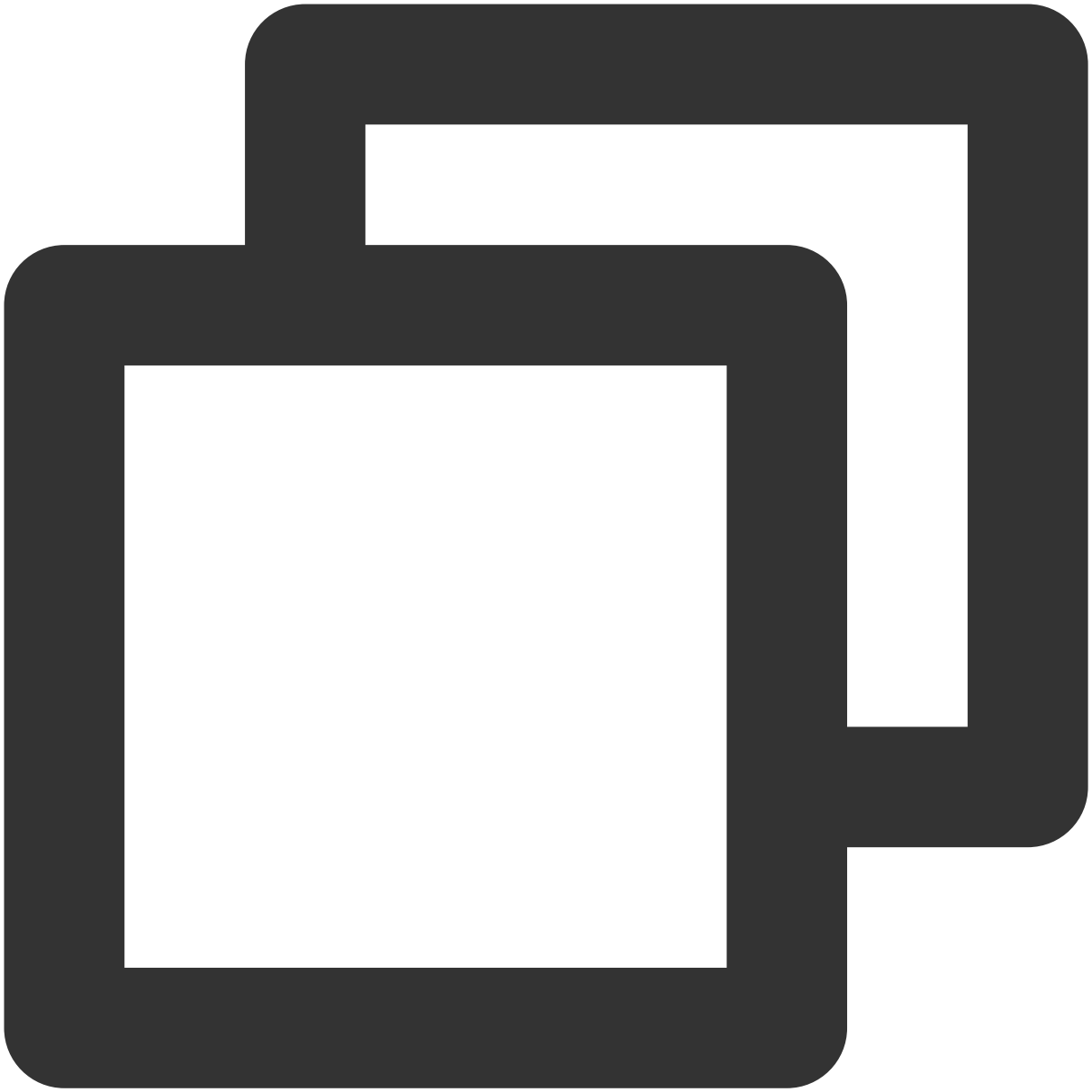
至少需要传入下载地址 URL，不支持嵌套 HLS 格式，仅支持单码流的HLS下载。userName 不传入具体值时，默认为"default"。



```
TXVodDownloadMedialnfo medialnfo = TXVodDownloadMedialnfo();
medialnfo.url = "http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq1500005830/0
TXVodDownloadController.instance.startDonwload(medialnfo);
```

步骤3：任务信息

在接收任务信息前，需要先设置回调 listener。



```
TXVodDownloadController.instance.setDownloadObserver((event, info) {

}, (errorCode, errorMsg, info) {

});
```

可能收到的任务event事件有：

事件	说明
EVENT_DOWNLOAD_START	任务开始，表示 SDK 已经开始下载

EVENT_DOWNLOAD_PROGRESS	任务进度，下载过程中，SDK 会频繁回调此接口，您可以通过 `mediaInfo.getProgress()` 获取当前进度
EVENT_DOWNLOAD_STOP	任务停止，当您调用 `stopDownload` 停止下载，收到此消息表示停止成功
EVENT_DOWNLOAD_FINISH	下载完成，收到此回调表示已全部下载。此时下载文件可以给 TXVodPlayer 播放

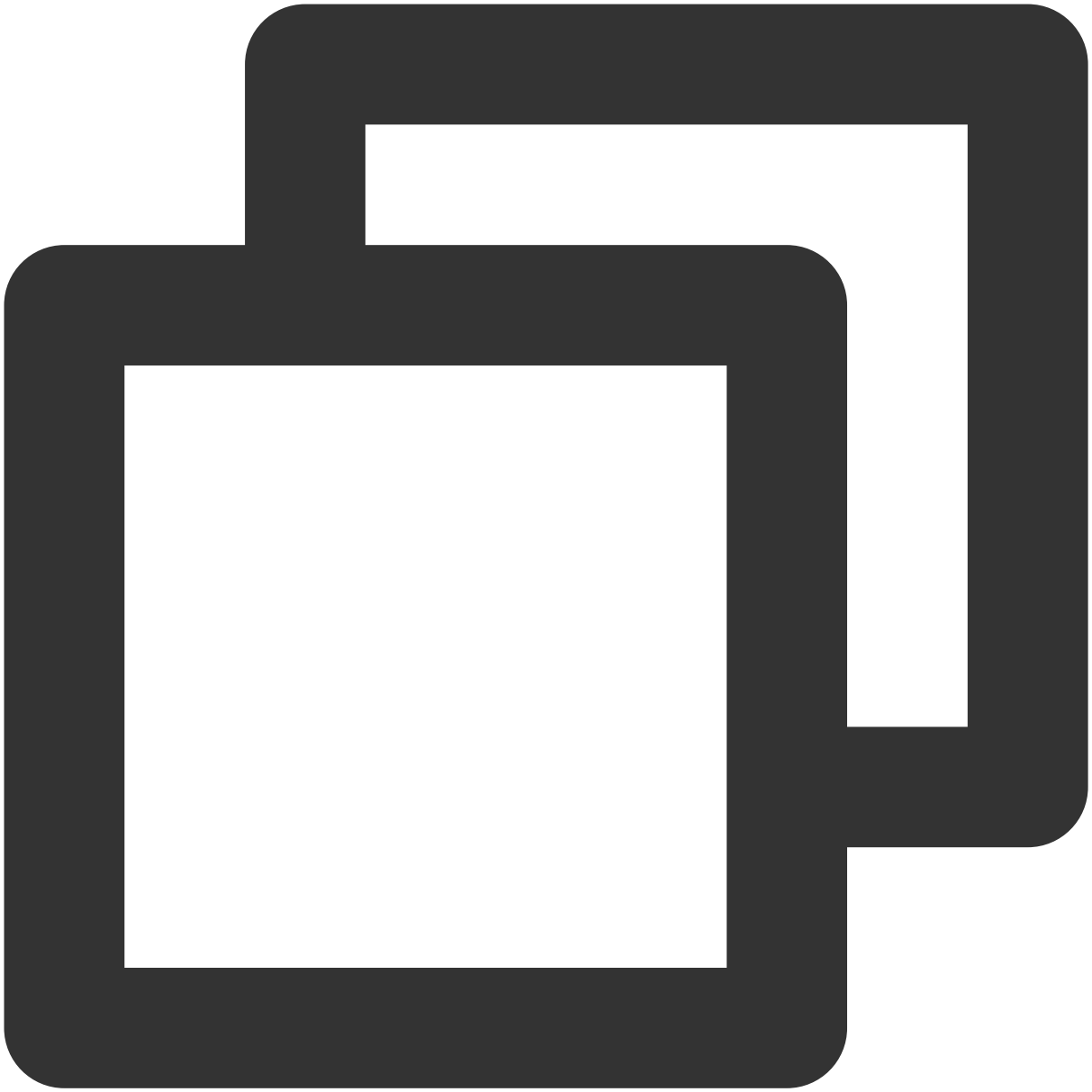
当回调 downlodOnErrorListener 方法的时候，代表下载错误，下载过程中遇到网络断开会回调此接口，同时下载任务停止。
由于 downloader 可以同时下载多个任务，所以回调接口里带上了 `TXVodDownloadMedialnfo` 对象，您可以访问 URL 或 dataSource 判断下载源，同时还可以获取到下载进度、文件大小等信息。

步骤4：中断下载

停止下载请调用 `TXVodDownloadController.instance.stopDownload()` 方法，参数为开始下载传入的 `TXVodDownloadMedialnfo` 对象。**SDK 支持断点续传**，当下载目录没有发生改变时，下次下载同一个文件时会从上次停止的地方重新开始。

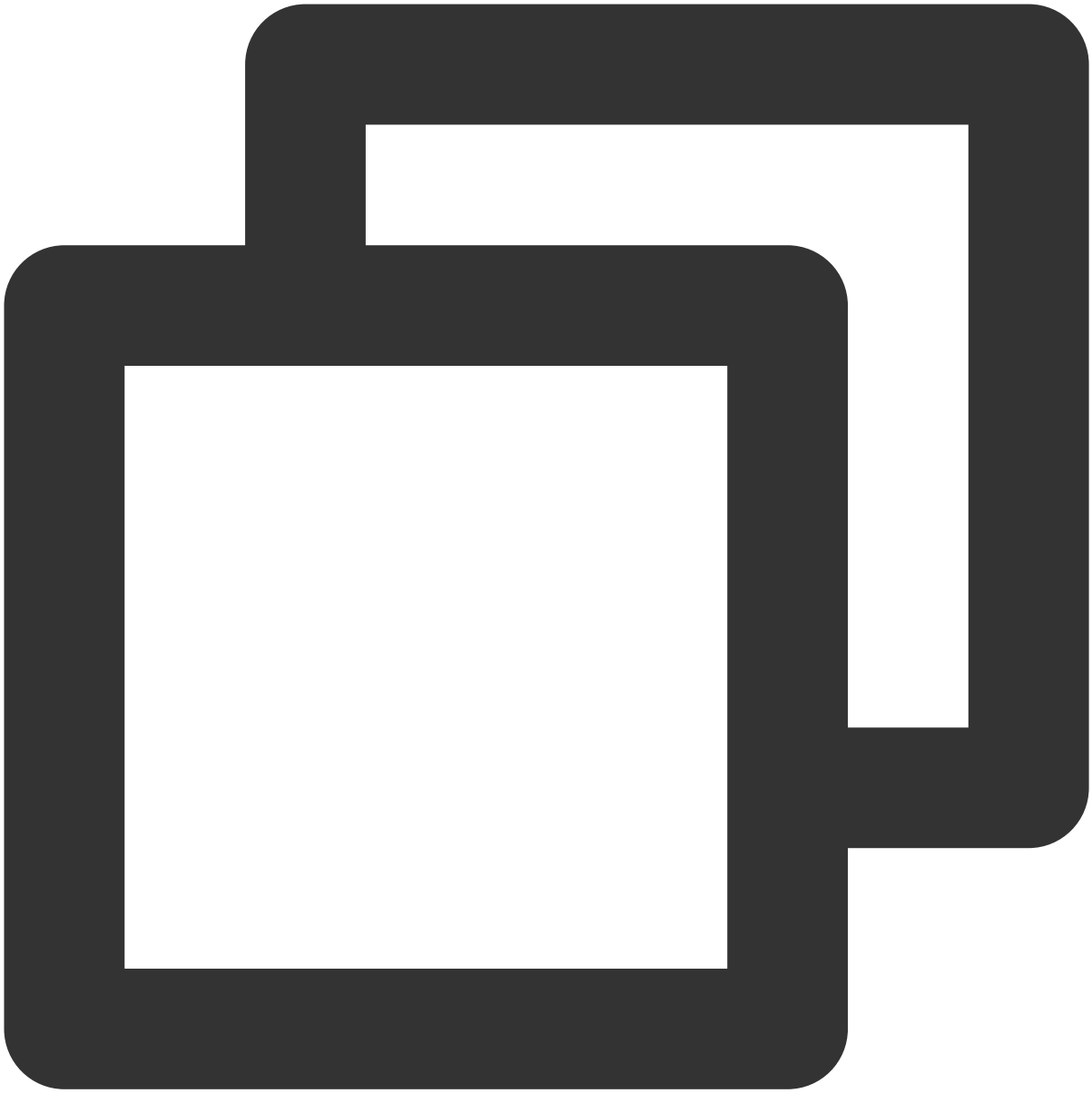
步骤5：管理下载

获取所有用户账户的下载列表信息, 也可获取指定用户账户的下载列表信息。

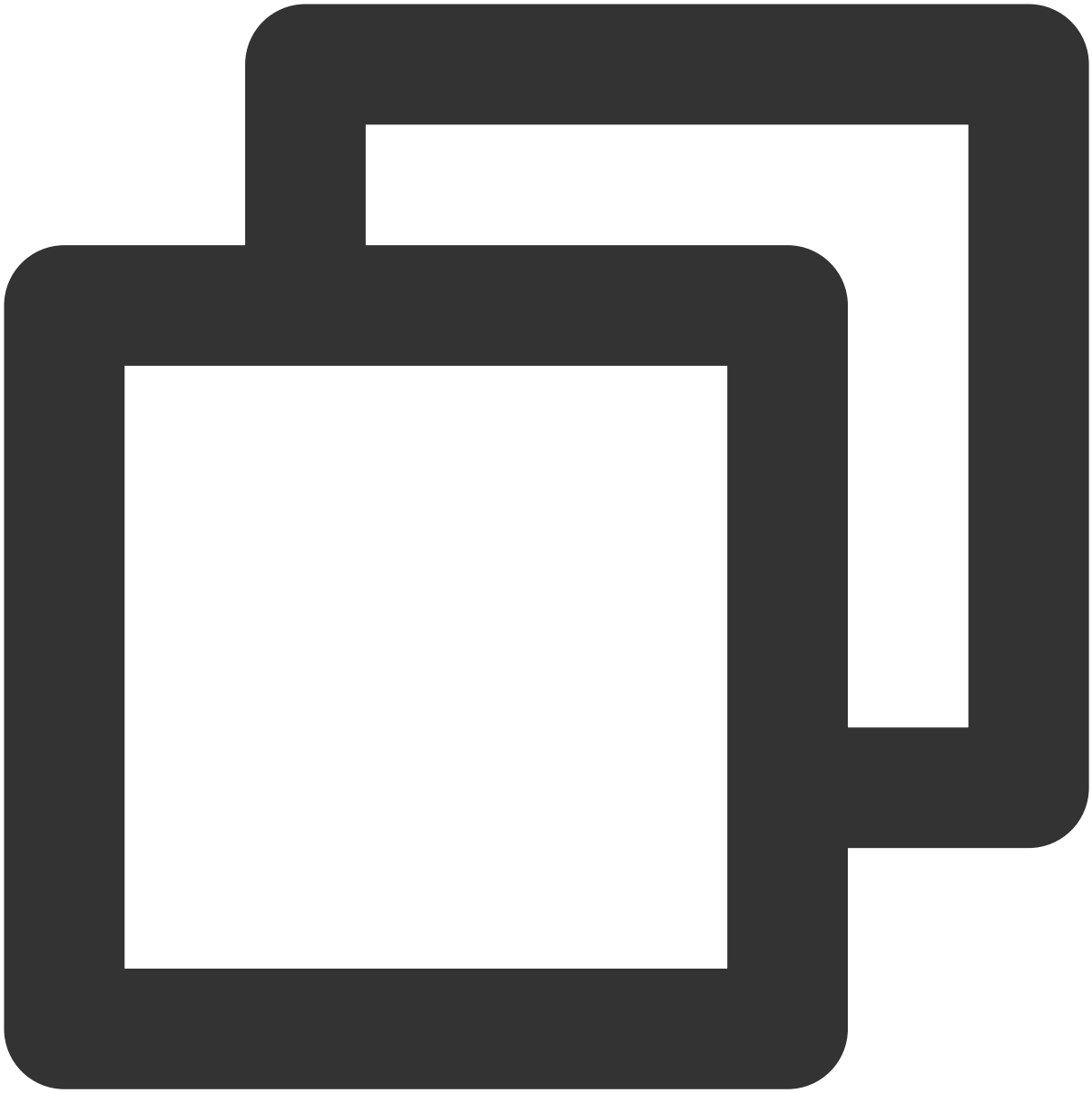


```
// 获取所有用户的下载列表信息
// 接入方可根据下载信息中的userName区分不同用户的下载列表信息
List<TXVodDownloadMedialnfo> downloadInfoList = await TXVodDownloadController.insta
```

获取某个 Fileid 相关下载信息，包括当前下载状态，获取当前下载进度，判断是否下载完成等，需要传入 AppID、Fileid 和 qualityId。



```
// 获取某个视频相关下载信息
TXVodDownloadMedialnfo downloadInfo = await TXVodDownloadController.instance.getDow
int? duration = downloadInfo.duration;  // 获取总时长
int? playableDuration = downloadInfo.playableDuration; // 获取已下载的可播放时长
double? progress = downloadInfo.progress;        // 获取下载进度
String? playPath = downloadInfo.playPath; // 获取离线播放路径，传给播放器即可离线播放
int? downloadState = downloadInfo.downloadState; // 获取下载状态，具体参考STATE_xxx常量
```

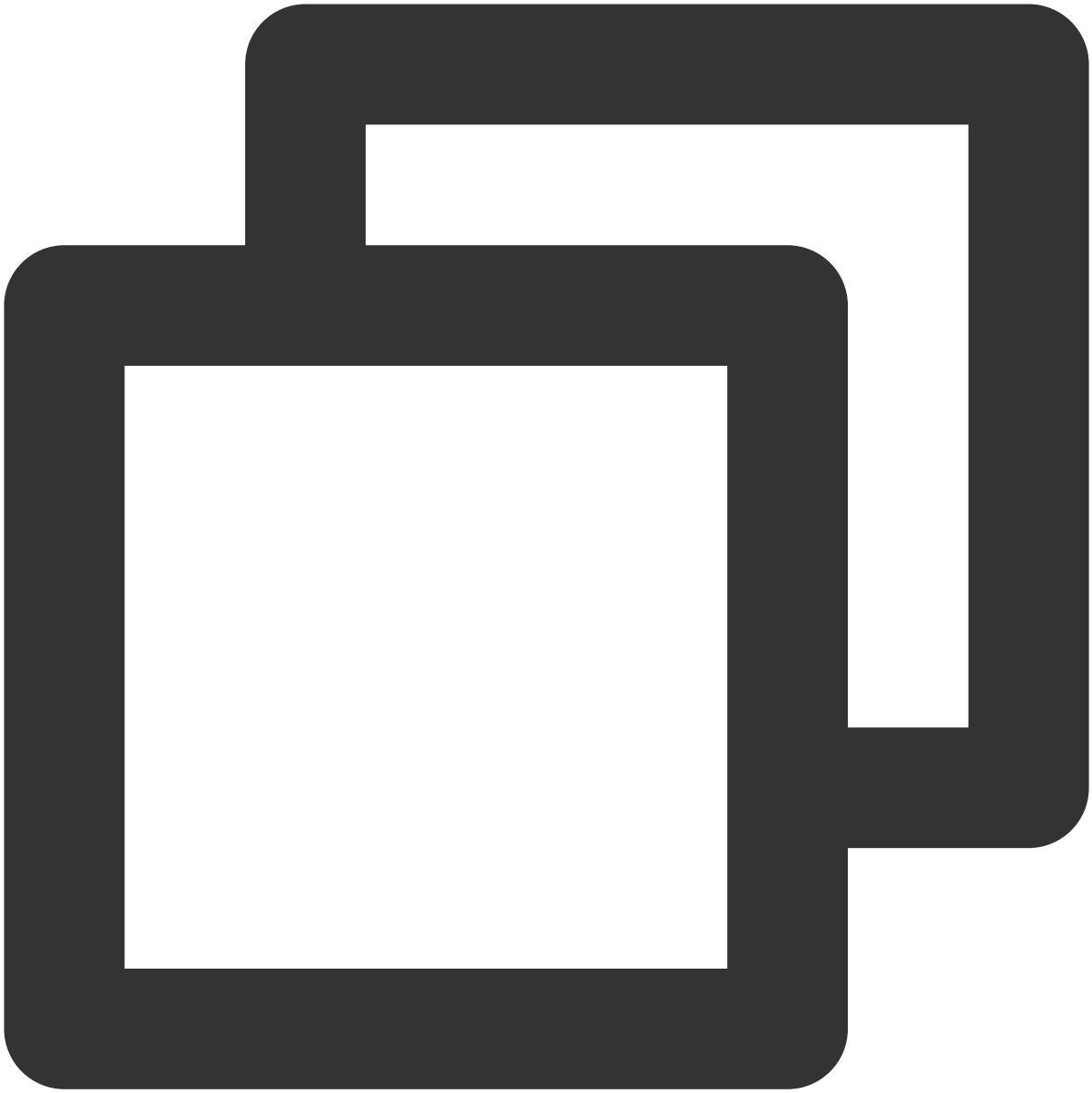


```
// 删除下载信息
bool result = await TXVodDownloadController.instance.deleteDownloadMediaInfo(medial
```

4、加密播放

视频加密方案主要用于在线教育等需要对视频版权进行保护的场景。如果要对您的视频资源进行加密保护，就不仅需要在播放器上做改造，还需要对视频源本身进行加密转码，亦需要您的后台和终端研发工程师都参与其中。在 视频加密解决方案 中您会了解到全部细节内容。

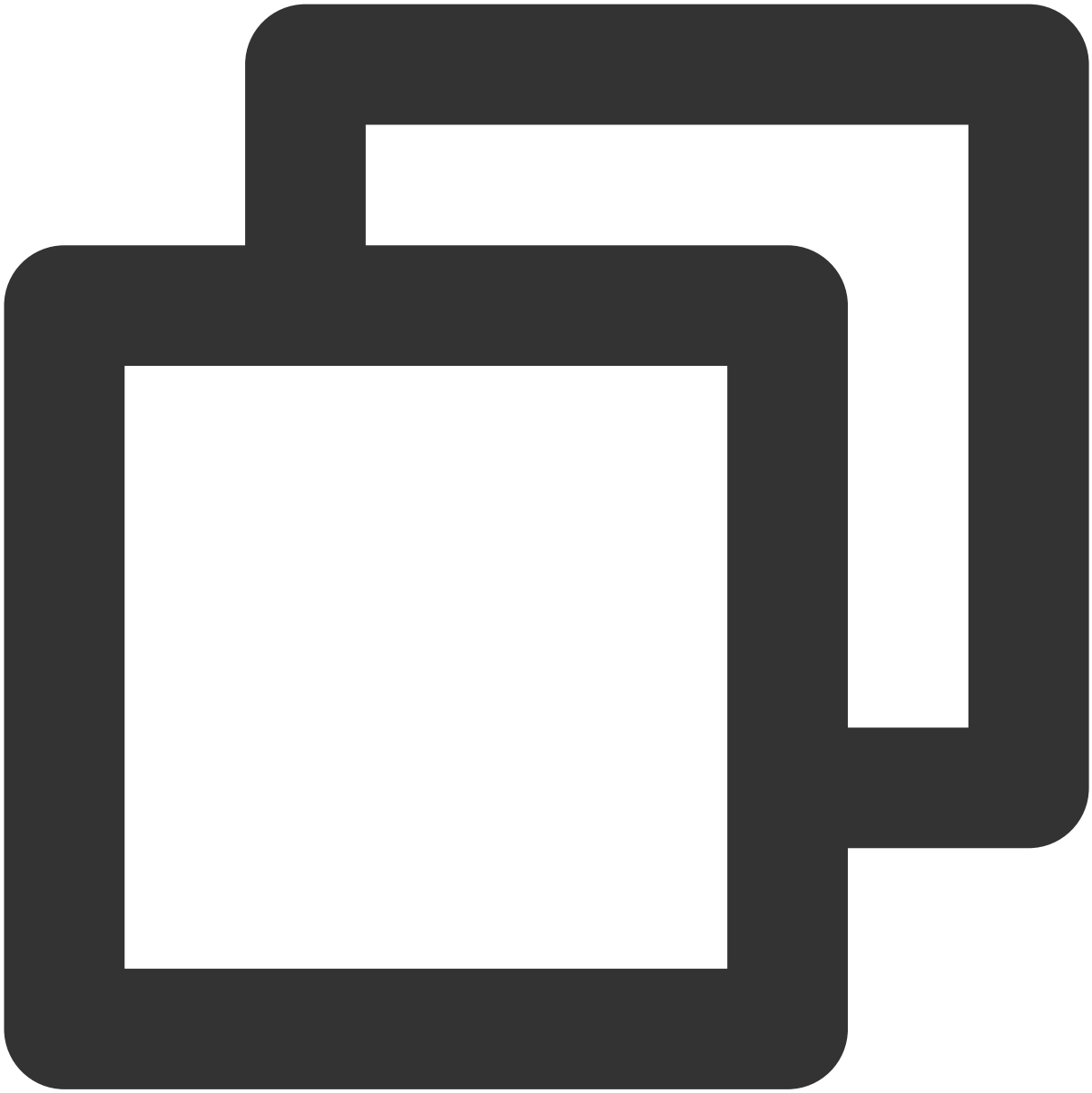
在腾讯云控制台提取到appId， 加密视频的fileId 和 psign后，可以通过下面的方式进行播放：



```
// psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/prod
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
        fileId: "4564972819220421305", psign: "psignxxxxxxx");
_controller.startVodPlayWithParams(params);
```

5、播放器配置

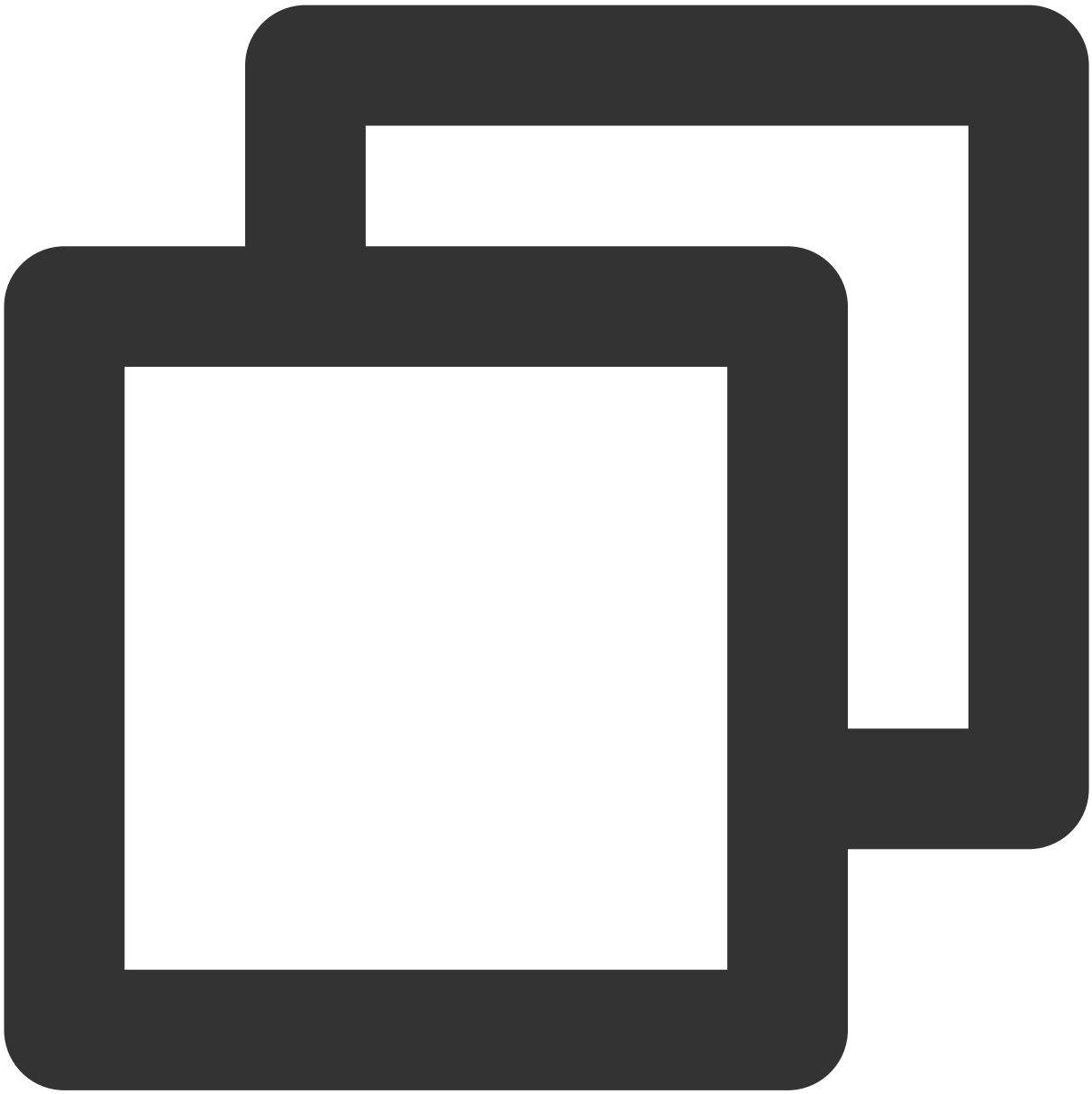
在调用 statPlay 之前可以通过 setConfig 对播放器进行参数配置，例如：设置播放器连接超时时间、设置进度回调间隔、设置缓存文件个数等配置，TXVodPlayConfig 支持配置的详细参数请单击 基础配置接口 了解。使用示例：



```
FTXVodPlayConfig config = FTXVodPlayConfig();
// 如果不配置preferredResolution，则在播放多码率视频的时候优先播放720 * 1280分辨率的码率
config.preferredResolution = 720 * 1280;
config.enableAccurateSeek = true;  // 设置是否精确 seek，默认 true
config.progressInterval = 200;  // 设置进度回调间隔，单位毫秒
config.maxBufferSize = 50;  // 最大预加载大小，单位 MB
_controller.setPlayConfig(config);
```

启播前指定分辨率

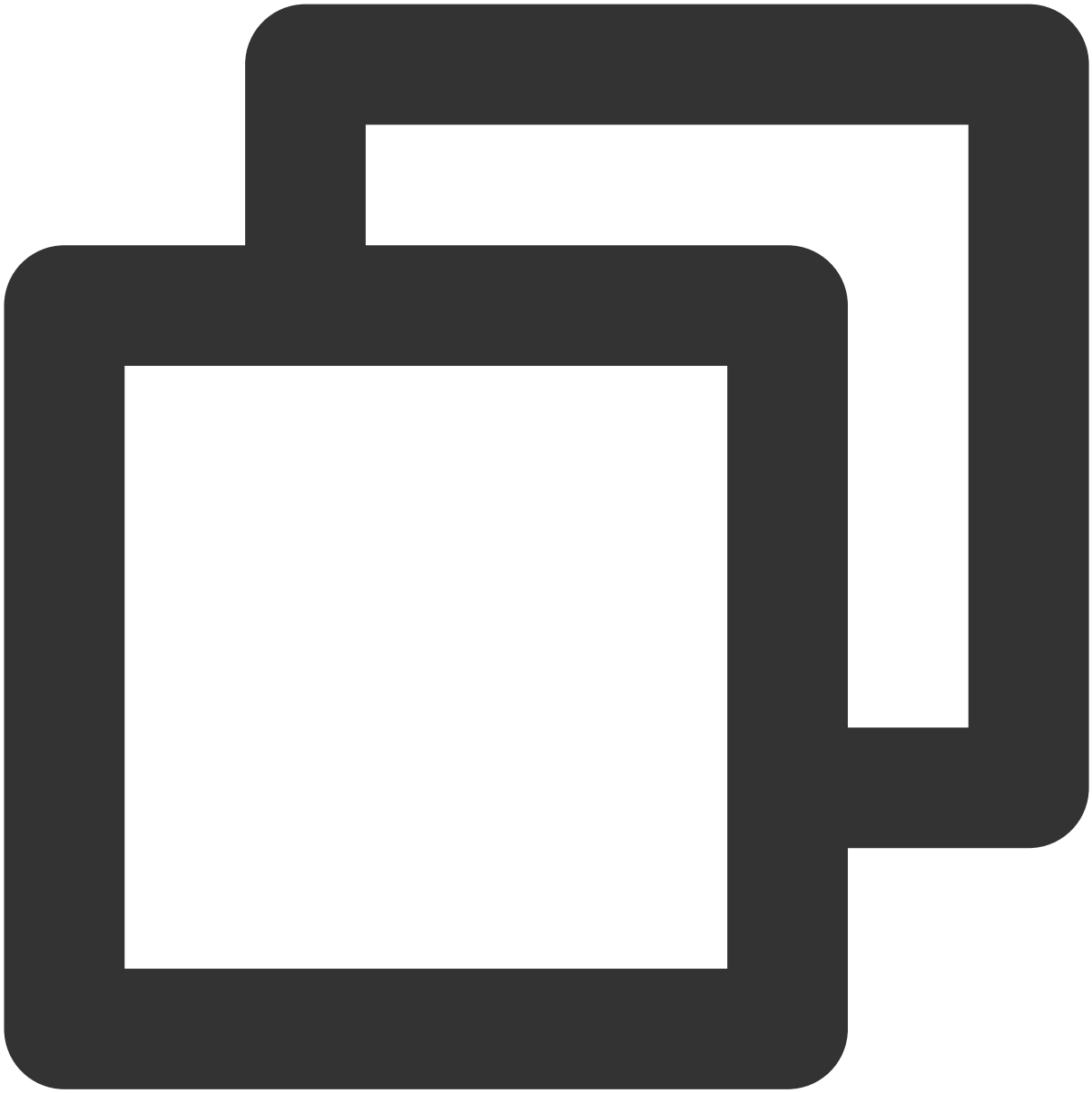
播放 HLS 的多码率视频源，如果你提前知道视频流的分辨率信息，可以在启播前优先指定播放的视频分辨率。 播放器会查找小于或等于偏好分辨率的流进行启播，启播后没有必要再通过 setBitrateIndex 切换到需要的码流。



```
FTXVodPlayConfig config = FTXVodPlayConfig();
// 传入参数为视频宽和高的乘积(宽 *  高)，可以自定义值传入，默认 720 * 1280
config.preferredResolution = 720 * 1280;
_controller.setPlayConfig(config);
```

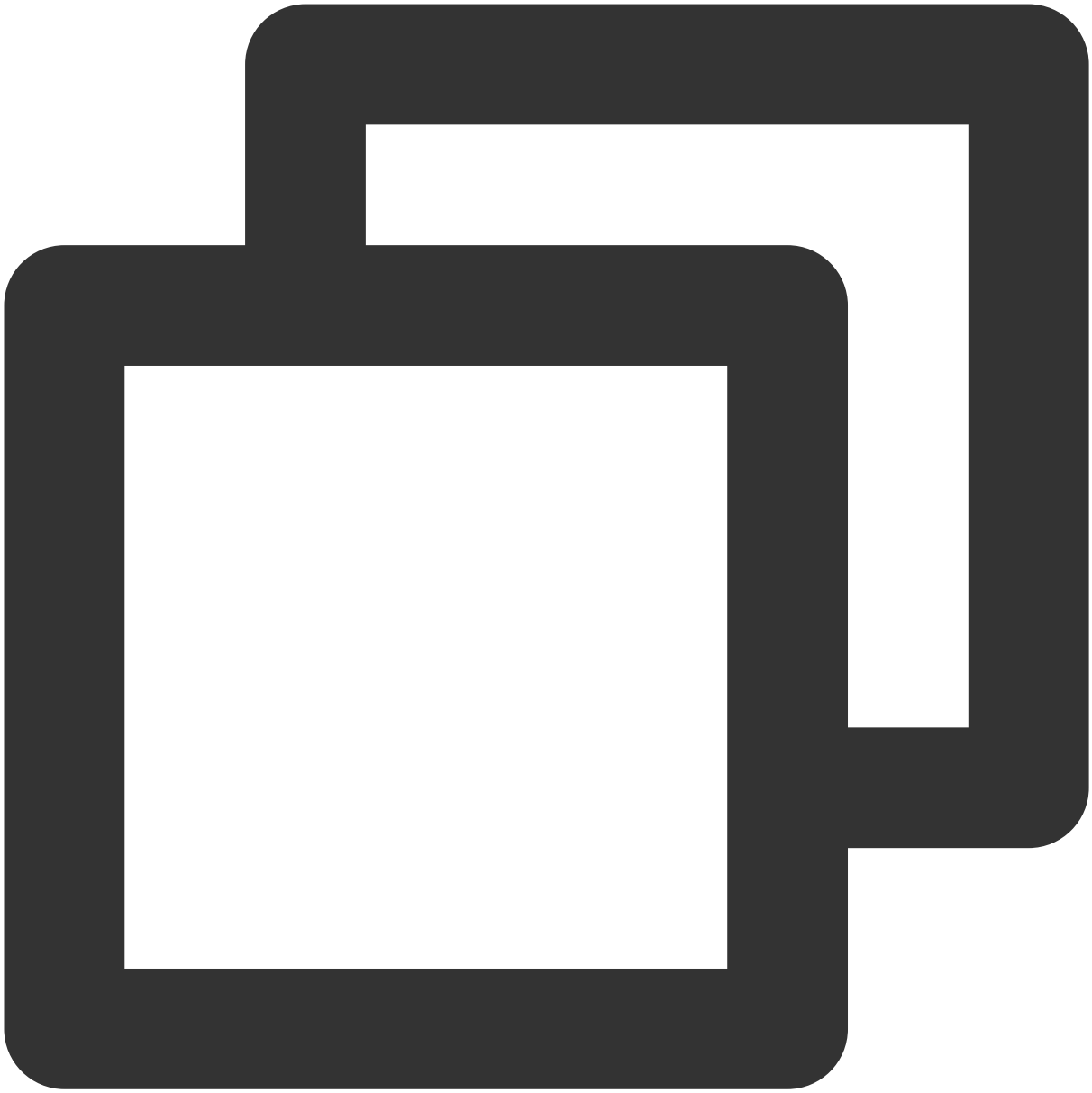
启播前指定媒资类型

当提前知道播放的媒资类型时，可以通过配置 `FTXVodPlayConfig#` mediaType减少播放器SDK内部播放类型探测，提升启播速度。



```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_FILE_VOD;    // 用于提升MP4启播速度
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_HLS_VOD;    // // 用于提升HLS启播速度
_controller.setPlayConfig(config);
```

设置播放进度回调时间间隔



```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.progressInterval = 200;  // 设置进度回调间隔，单位毫秒
_controller.setPlayConfig(config);
```

播放器事件监听

您可以通过 `TXVodPlayerController` 的 `onPlayerEventBroadcast` 来监听播放器的播放事件，来向您的应用程序同步信息。

播放事件通知（onPlayerEventBroadcast）

事件 ID	数值	含义说明
PLAY_EVT_PLAY_BEGIN	2004	视频播放开始
PLAY_EVT_PLAY_PROGRESS	2005	视频播放进度，会通知当前播放进度、加载进度 和总体时长
PLAY_EVT_PLAY_LOADING	2007	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件
PLAY_EVT_VOD_LOADING_END	2014	视频播放 loading 结束，视频继续播放
VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seek 完成，10.3版本开始支持

结束事件

事件 ID	数值	含义说明
PLAY_EVT_PLAY_END	2006	视频播放结束
PLAY_ERR_NET_DISCONNECT	-2301	网络断连,且经多次重连亦不能恢复,更多重试请自行重启播放
PLAY_ERR_HLS_KEY	-2305	HLS 解密 key 获取失败

警告事件

如下的这些事件您可以不用关心，它只是用来告知您 SDK 内部的一些事件。

事件 ID	数值	含义说明
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	当前视频帧解码失败
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	当前音频帧解码失败
PLAY_WARNING_RECONNECT	2103	网络断连, 已启动自动重连 (重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT 了)
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	硬解启动失败，采用软解

连接事件

连接服务器的事件，主要用于测定和统计服务器连接时间：

事件 ID	数值	含义说明
PLAY_EVT_VOD_PLAY_PREPARED	2013	播放器已准备完成，可以播放。设置了 autoPlay 为 false

		之后，需要在收到此事件后，调用 resume 才会开始播放
PLAY_EVT_RCV_FIRST_I_FRAME	2003	网络接收到首个可渲染的视频数据包（IDR）

画面事件

以下事件用于获取画面变化信息：

事件 ID	数值	含义说明
PLAY_EVT_CHANGE_RESOLUTION	2009	视频分辨率改变
PLAY_EVT_CHANGE_ROTATION	2011	MP4 视频旋转角度

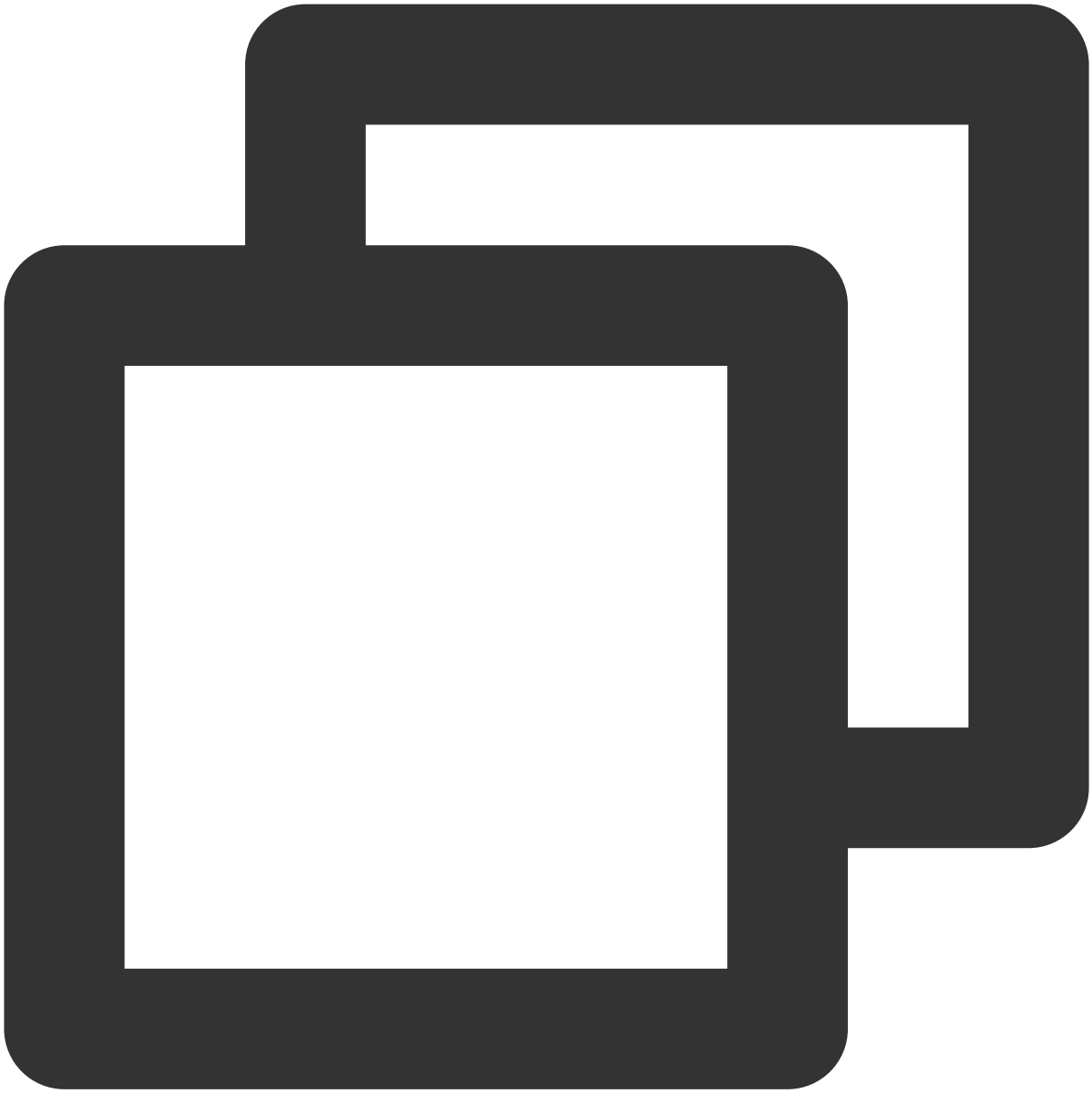
视频信息事件

事件 ID	数值	含义说明
PLAY_EVT_GET_PLAYINFO_SUCC	2010	成功获取播放文件信息

如果通过 fileId 方式播放且请求成功（接口：startVodPlay(TXPlayerAuthBuilder authBuilder)），SDK 会将一些请求信息通知到上层。您可以在收到 `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` 事件后，解析 param 获取视频信息。

视频信息	含义说明
EVT_PLAY_COVER_URL	视频封面地址
EVT_PLAY_URL	视频播放地址
EVT_PLAY_DURATION	视频时长
EVT_TIME	事件发生时间
EVT_UTC_TIME	UTC 时间
EVT_DESCRIPTION	事件说明
EVT_PLAY_NAME	视频名称
EVT_IMAGESPRIT_WEBVTTURL	雪碧图 web vtt 描述文件下载 URL，10.2版本开始支持
EVT_IMAGESPRIT_IMAGEURL_LIST	雪碧图图片下载URL，10.2版本开始支持
EVT_DRM_TYPE	加密类型，10.2版本开始支持

通过 onPlayerEventBroadcast 获取视频播放过程信息示例：



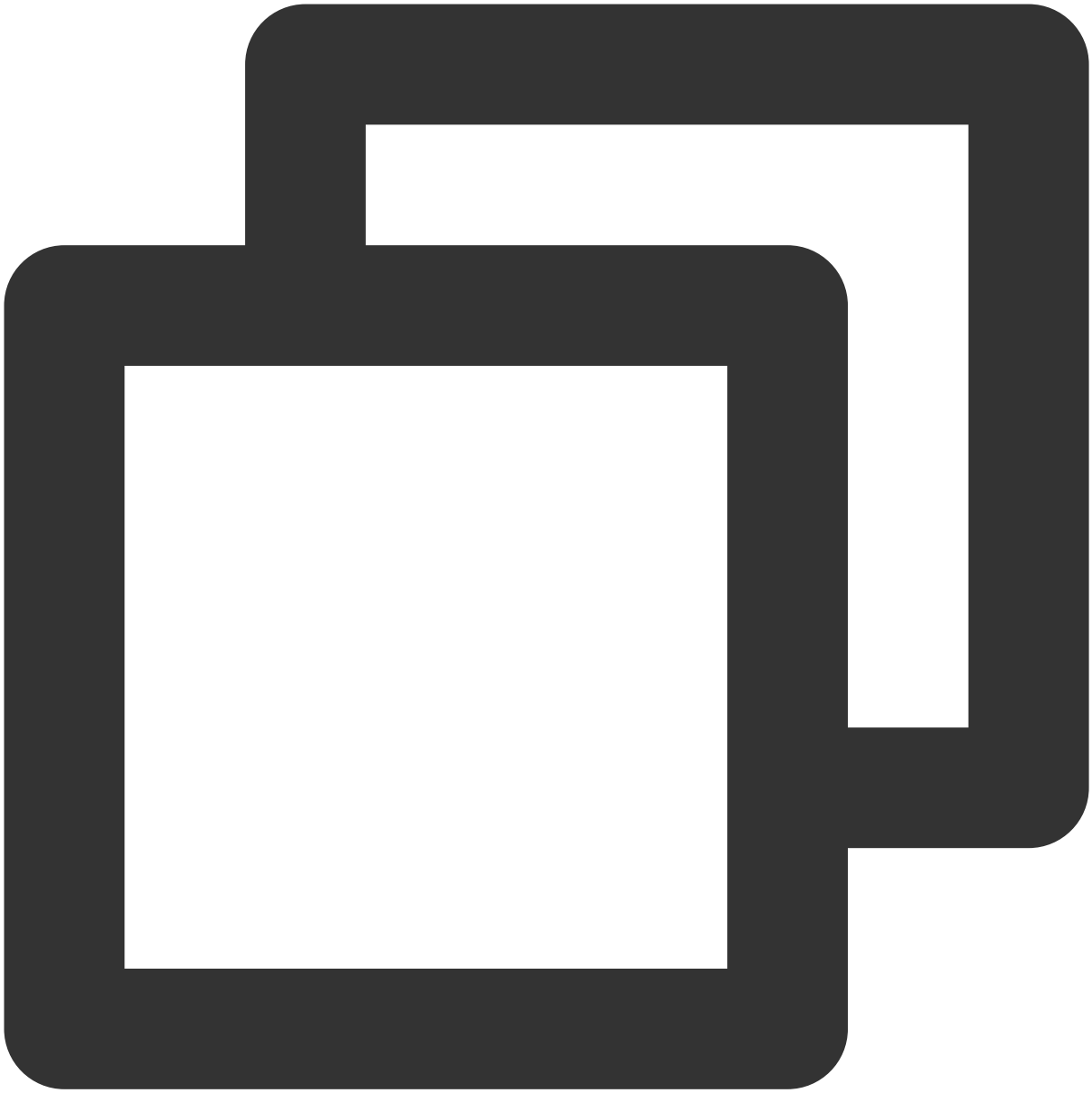
```
_controller.onPlayerEventBroadcast.listen((event) async {
  if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_BEGIN || event["event"] == TXV
  // code ...
  } else if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {
  // code ...
  }
});
```

播放状态反馈（onPlayerNetStatusBroadcast）

状态反馈每0.5秒都会被触发一次，目的是实时反馈当前的推流器状态，它就像汽车的仪表盘，可以告知您目前 SDK 内部的一些具体情况，以便您能对当前视频播放状态等有所了解。

评估参数	含义说明
NET_STATUS_CPU_USAGE	当前瞬时 CPU 使用率
NET_STATUS_VIDEO_WIDTH	视频分辨率 - 宽
NET_STATUS_VIDEO_HEIGHT	视频分辨率 - 高
NET_STATUS_NET_SPEED	当前的网络数据接收速度
NET_STATUS_VIDEO_FPS	当前流媒体的视频帧率
NET_STATUS_VIDEO_BITRATE	当前流媒体的视频码率，单位 kbps
NET_STATUS_AUDIO_BITRATE	当前流媒体的音频码率，单位 kbps
NET_STATUS_VIDEO_CACHE	缓冲区（jitterbuffer）大小，缓冲区当前长度为0，说明离卡顿就不远了
NET_STATUS_SERVER_IP	连接的服务器 IP

通过 onNetStatus 获取视频播放过程信息示例：



```
_controller.onPlayerNetStatusBroadcast.listen((event) async {
  int videoWidth = event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH];
});
```

视频播放状态反馈

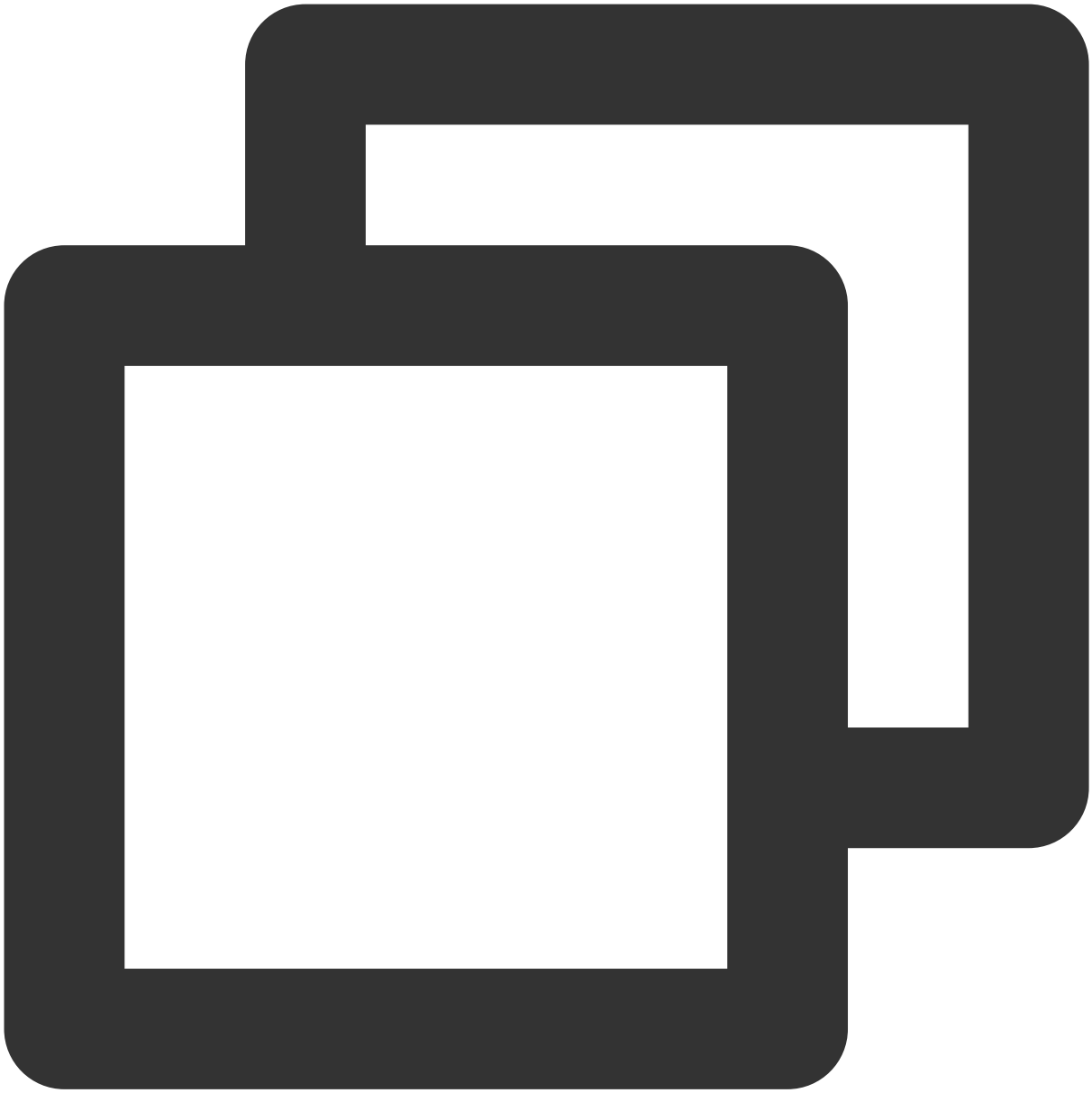
视频播放状态会在每一次播放状态切换的时候进行通知。

事件通过枚举类 TXPlayerState 来传递事件

状态	含义

paused	暂停播放
failed	播放失败
buffering	缓冲中
playing	播放中
stopped	停止播放
disposed	控件释放了

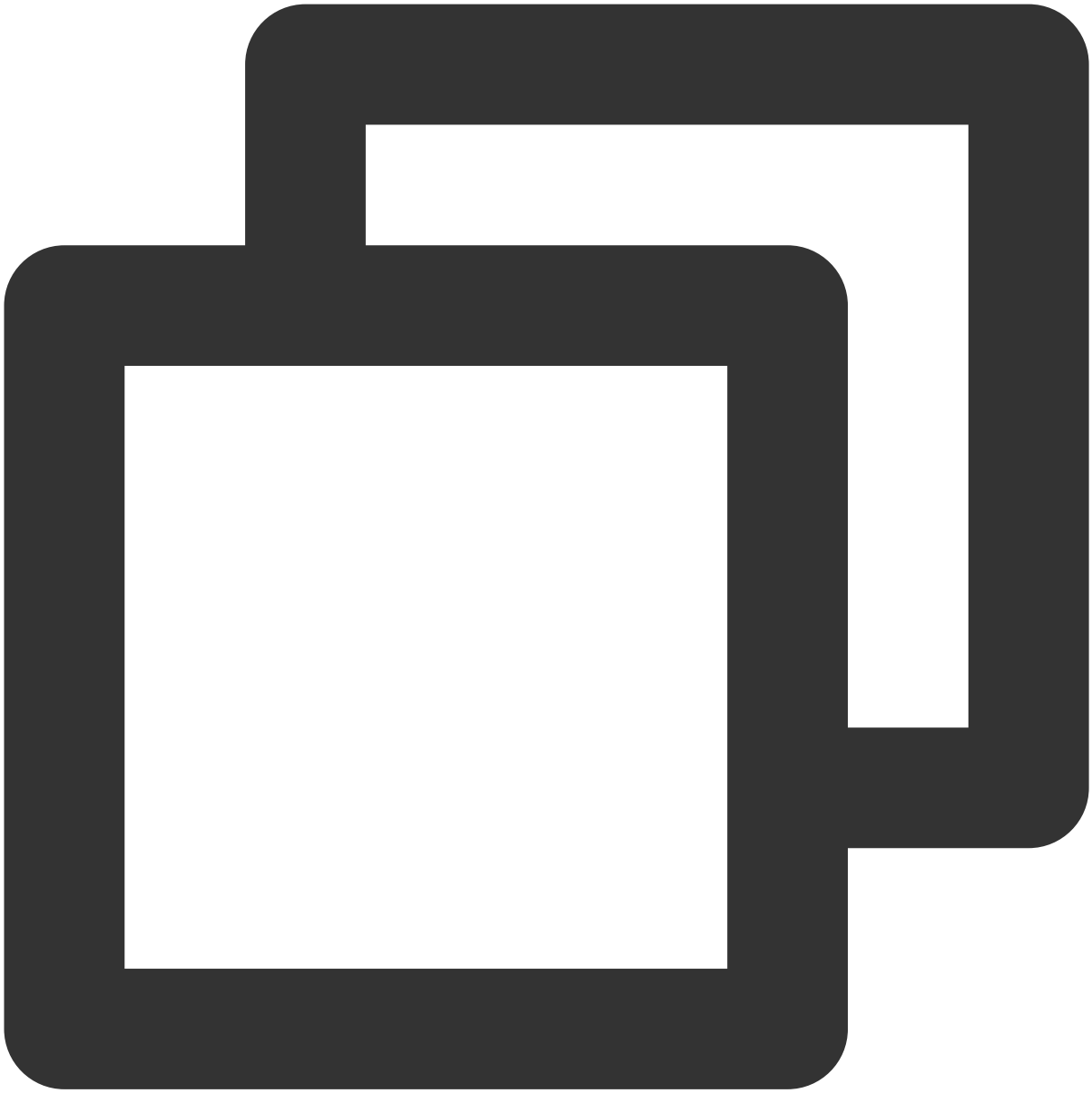
通过 onPlayerState 获取视频播放状态示例：



```
_controller.onPlayerState.listen((val) { });
```

系统音量监听

为了方便监控视频播放音量，SDK 在 flutter 层做了对原生层的音量变化通知进行了事件封装，可以直接通过 SuperPlayerPlugin 来监听当前设备的音量变化。
通过 onEventBroadcast 获取设备音量状态示例：



```
SuperPlayerPlugin.instance.onEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

相关事件含义如下：

状态	值	含义
EVENT_VOLUME_CHANGED	1	音量发生变化
EVENT_AUDIO_FOCUS_PAUSE	2	失去音量输出播放焦点，仅使用Android

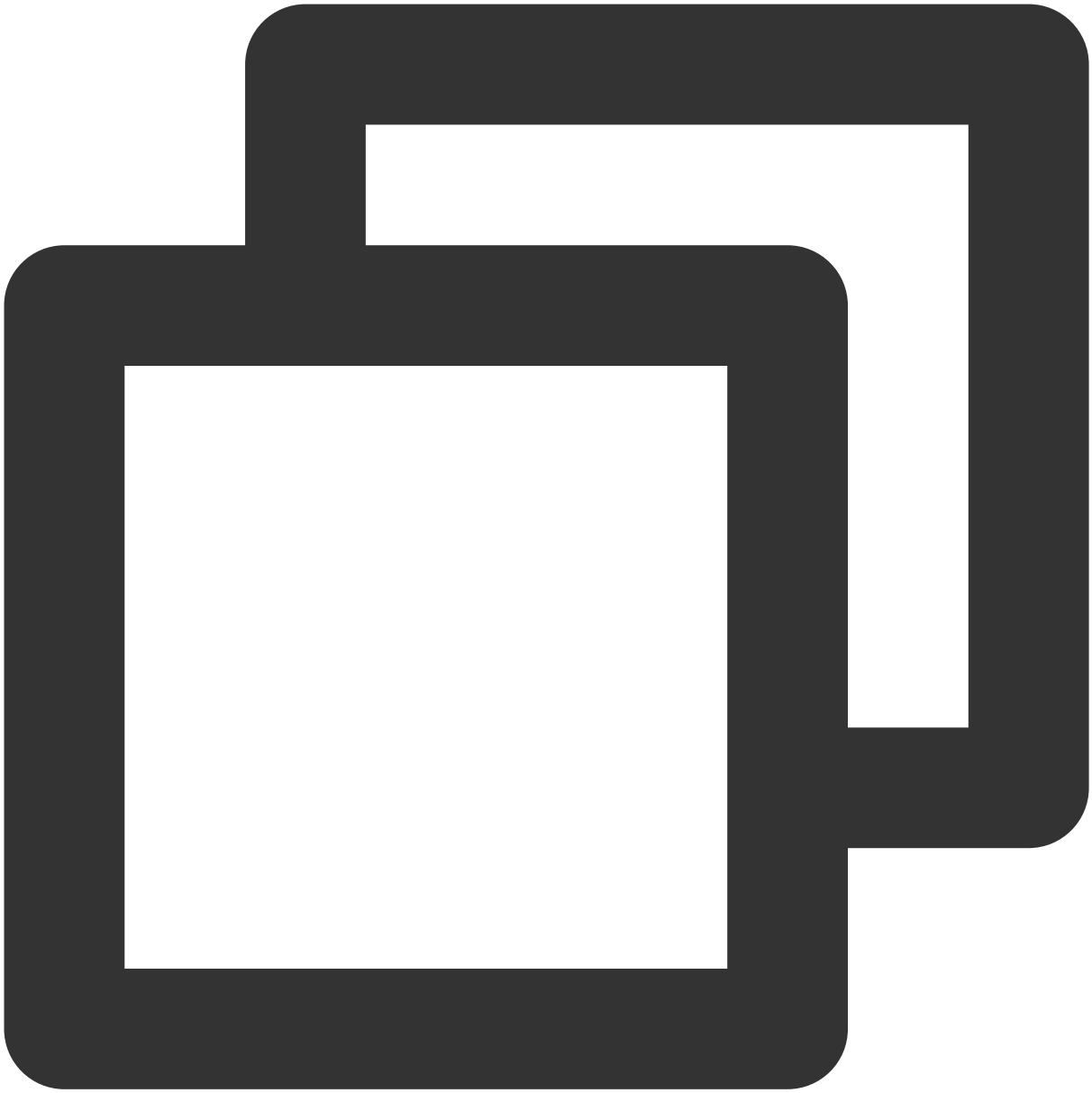
EVENT_AUDIO_FOCUS_PLAY	3	获得音量输出焦点，仅使用Android

画中画事件监听

由于SDK使用的画中画是基于系统提供的画中画能力，进入画中画之后，提供了一系列通知来帮助用户对当前界面做响应的调整。

状态	值	含义
EVENT_PIP_MODE_ALREADY_ENTER	1	已经进入画中画模式
EVENT_PIP_MODE_ALREADY_EXIT	2	已经退出画中画模式
EVENT_PIP_MODE_REQUEST_START	3	开始请求进入画中画模式
EVENT_PIP_MODE_UI_STATE_CHANGED	4	pip UI状态发生变动，仅在 android 31以上生效
EVENT_IOS_PIP_MODE_RESTORE_UI	5	重置UI，仅在 IOS 生效
EVENT_IOS_PIP_MODE_WILL_EXIT	6	将要退出画中画，仅在 IOS 生效

使用 onExtraEventBroadcast 监听画中画事件的示例如下：



```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

高级功能
Web 高级功能
安全检查插件（TCPlayerSafeCheckPlugin）

最近更新时间：2024-04-11 16:18:08

TCPlayerSafeCheckPlugin 插件用于检测播放环境和播放状态是否正常，保护播放安全。需结合播放器 TCPlayer 使用。

使用条件

目前 Web 播放器 SDK 5.0.0 以上版本支持使用 VR 播放插件。
VR 播放需获取 Web 端播放器高级版 License 方可使用。
在长期维护播放器的过程中，遭遇过多种攻击方式，对以下可通过第三方工具盗取视频资源的行为，本插件从以下三个方面进行了针对性防范：

1. MSE 环境检测

部分浏览器插件或脚本可以劫持当前播放环境，通过修改 Media Source Extensions API (MSE) 来截获播放数据，并最终实现下载视频，本插件可以检测并防范此类攻击手段。

2. 安全结构检查

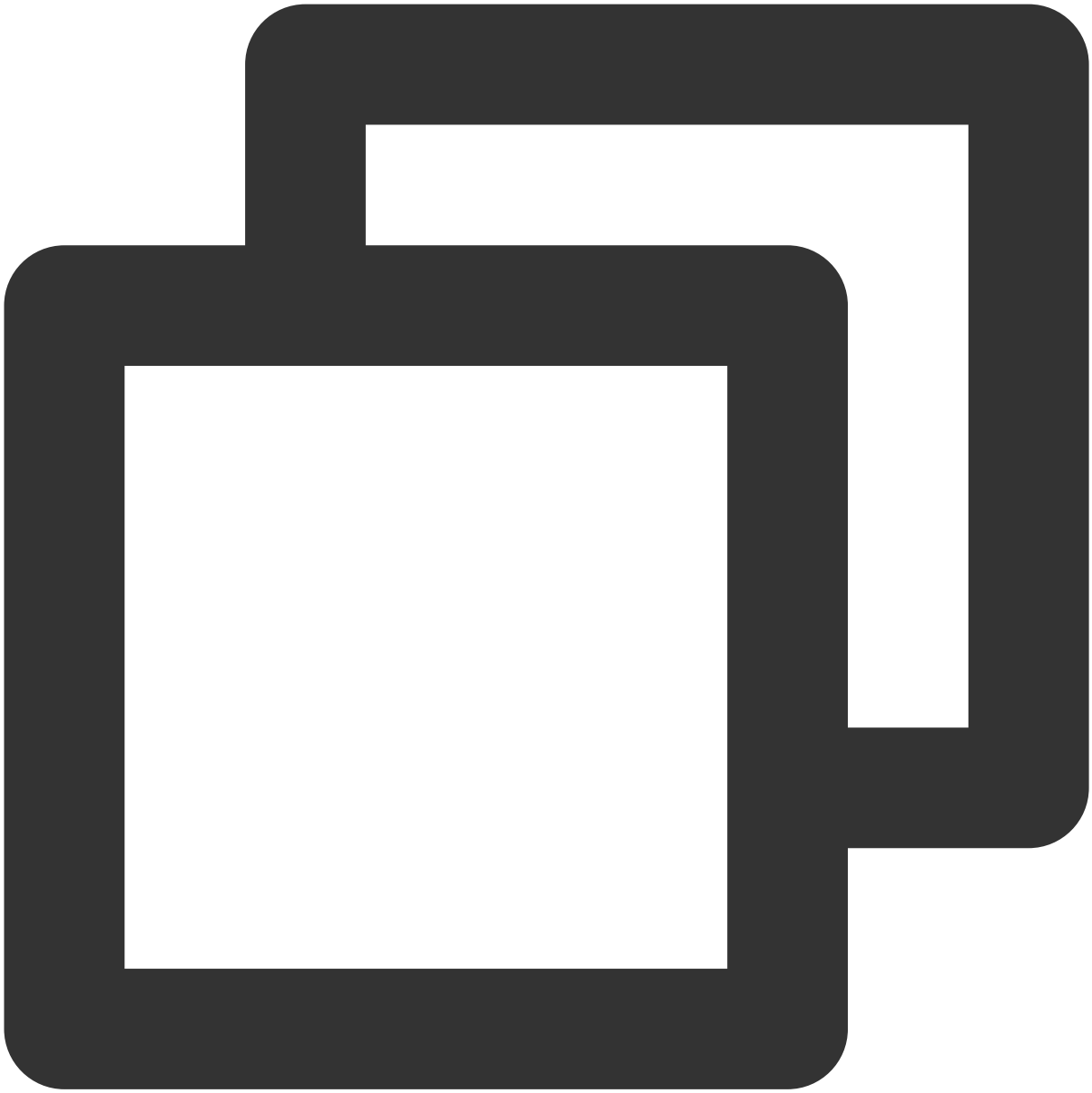
通过第三方工具或脚本可以修改播放器结构，达到去除播放标识、去除水印等目的，并实现录屏，本插件会检测播放器结构是否被侵入篡改，一旦发现此类行为，会及时中止播放。

3. 接口响应完整性校验

播放器使用过程中，需要与云点播服务端进行数据交互，一旦接口数据被修改，会对正常的播放行为产生影响。本插件可以检测并防范此类攻击手段。

使用方式

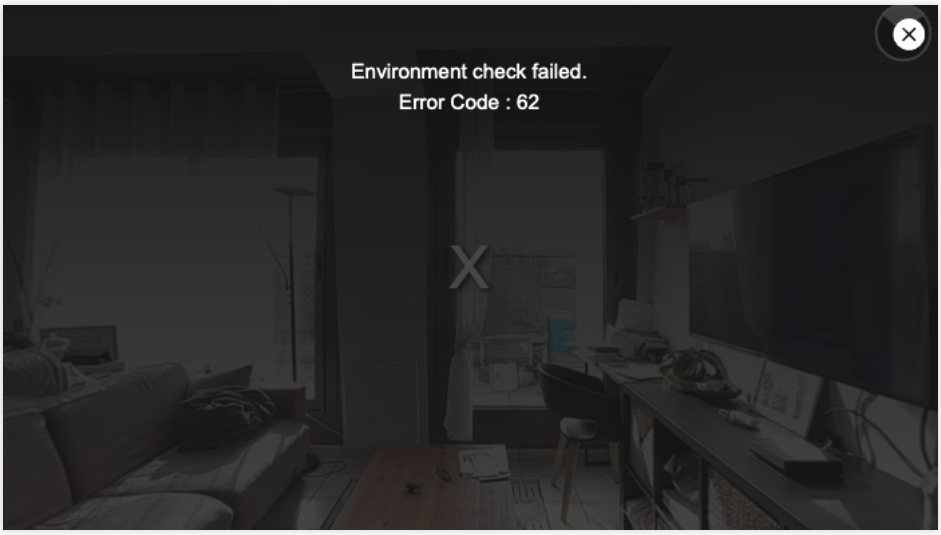
集成 TCPlayer 可以参见 TCPlayer 集成指引 和 API 文档。
创建播放器实例时，可通过声明 plugins 插件的方法开启安全检查能力，开启后播放器会自动加载并使用本插件：



```
const player = TCPlayer('player-container-id', {
  plugins: {
    SafeCheck: true,
  }
});
```

效果

开启插件后，播放器会自动检测环境是否安全，如果遭遇上述攻击手段会自动终止播放，并进行相应提示，如下图：



插件会提示的错误码如下：

错误码	说明
60	安全结构检查异常
61	接口响应完整性异常
62	MSE 环境检测异常

VR 播放插件（TCPlayerVRPlugin）

最近更新时间：2024-04-11 16:18:08

TCPlayerVRPlugin 插件可用于 VR 全景视频播放，播放中可以通过陀螺仪转动或手势操作来改变视角，提供多种属性和方法来控制播放表现，支持 PC 端和移动端。

使用条件

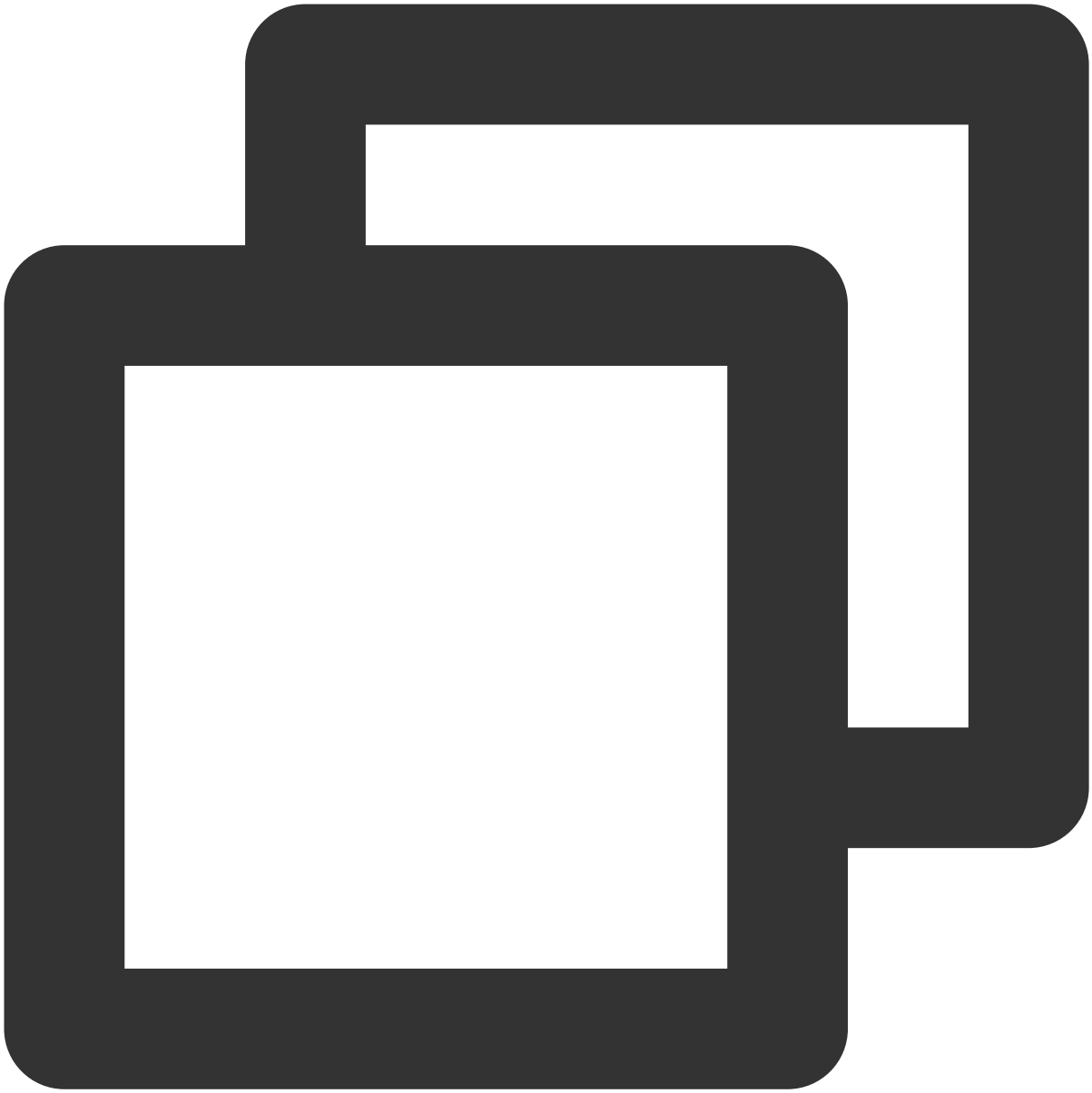
目前 Web 播放器 SDK 5.0.0 以上版本支持使用 VR 播放插件。

VR 播放需获取 Web 端播放器高级版 License 方可使用。

接入方式

播放器初始化过程参见 TCPlayer 集成指引 和 API 文档。

初始化播放器实例时，可通过声明 plugins 插件的方法开启 VR 播放能力，开启后播放器会自动加载并使用本插件：



```
const player = TCPlayer('player-container-id', { // player-container-id 为播放器容器I
  plugins: {
    VR: {
      isEnableController: true,
      ...
    },
  }
});
```

VR 插件属性说明

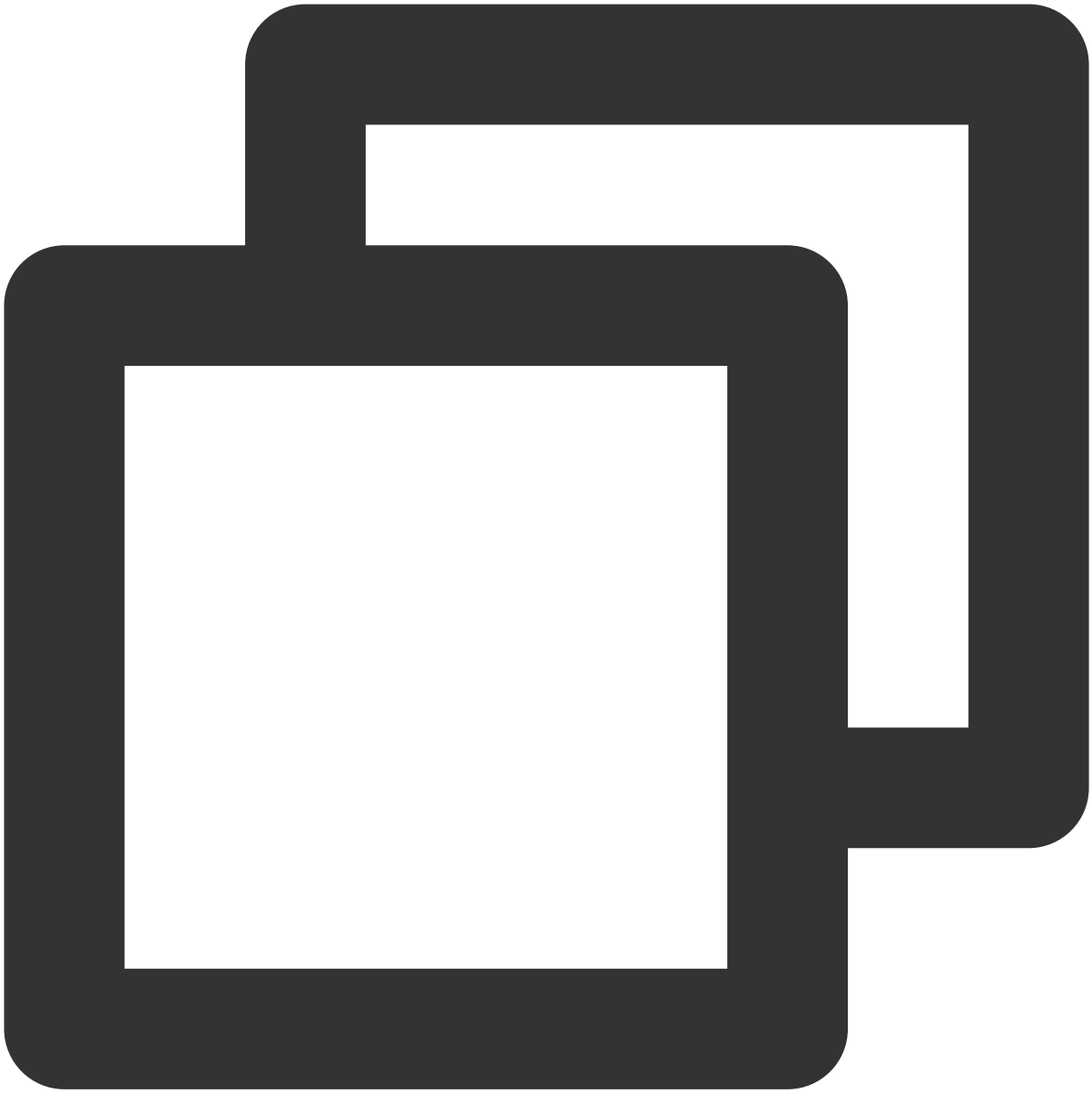
名称	说明	默认值
isEnableController	是否打开 VR 控制器	true
isEnableZoom	是否允许画面缩放	true
yaw	初始化左右视角角度，单位为度	0
pitch	初始化上下视角角度，单位为度	0
fov	初始化可视视角，单位为度	65
yawRange	限制视角活动的范围	[-180, 180]
pitchRange	限制视角活动的范围	[-90, 90]
fovRange	限制视角活动的范围	[30, 110]

VR 插件方法说明

VR 插件初始化后，会生成实例，实例化之后会进入 VR 模式播放视频，可以在播放器实例上找到 VR 实例，通过 VR 实例可以调用相关方法：

- lookAt

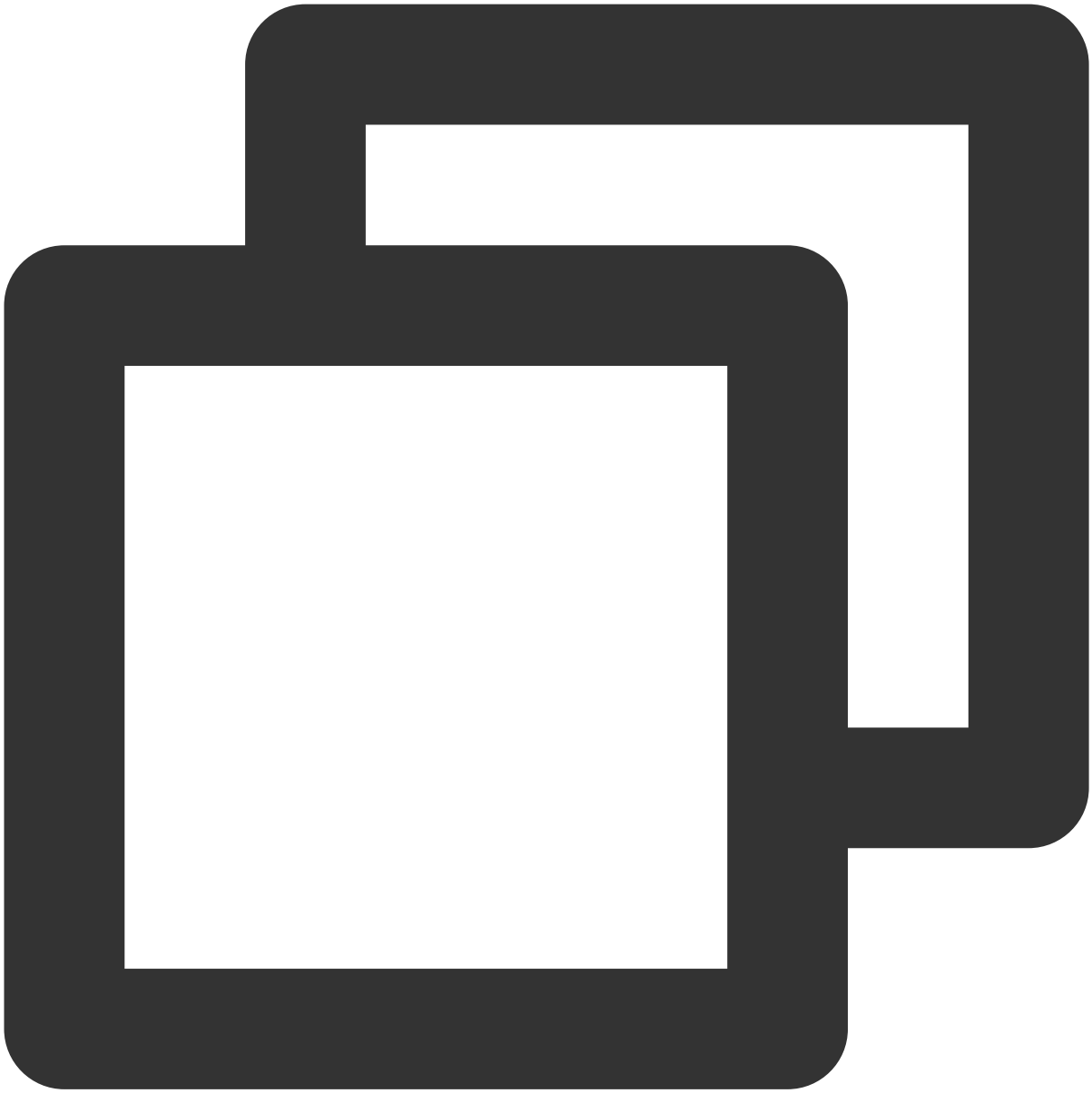
在一段时间通过动画形式移动到特定角度的视角。



```
player.plugins['VR'].lookAt({ yaw: 30 }, 1000);
```

- setGyroMode

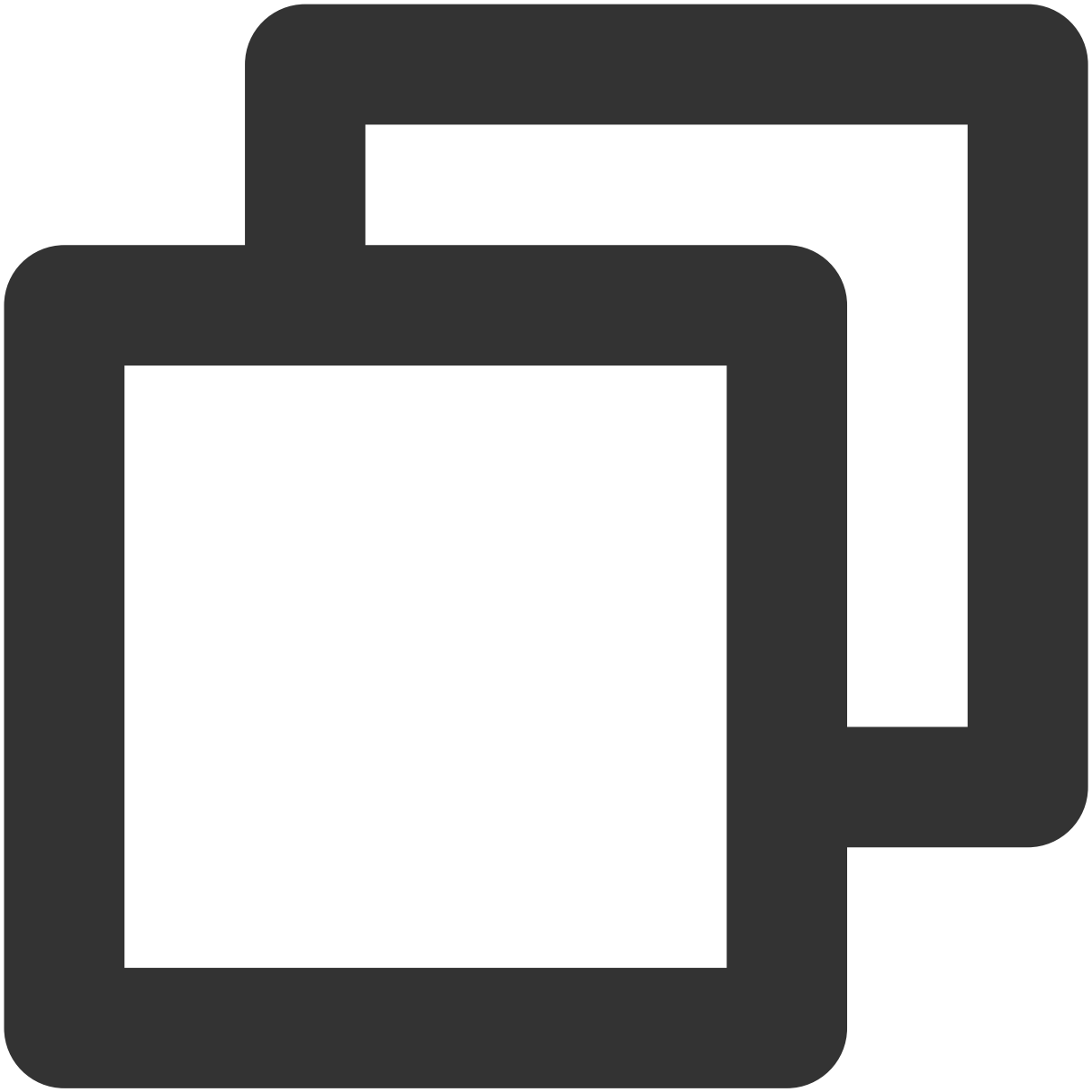
如果您的设备拥有运动传感器（陀螺仪、加速传感器），您可以通过设备的运动来改变视角，这个方法可取值为 `'VR' | 'none' | 'yawPitch'` 。



```
player.plugins['VR'].setGyroMode('none');
```

- enableSensor

获取使用运动传感器的权限。一般在 Android 设备，默认会开启运动传感器，在 iOS 13+，则需要触发人机交互手动获取权限。



```
player.plugins['VR'].enableSensor();
```

说明：

1. 在浏览器劫持播放的环境，无法支持 VR 视频的播放。
2. Android 端播放器初始化后会默认进入 VR 模式，并开启陀螺仪。
3. iOS 端根据系统版本不同，表现会有差异：

3.1 系统版本13+，需要手动点击页面，触发人机交互并获取权限后，才能打开陀螺仪。

3.2 系统版本12.2 - 13，需进入系统设置手动开启运动传感器。通常来说开启路径一般为设置 > Safari > 动作与方向访问，开启传感器后刷新页面，即可打开。

示例

单击 此处 可参考示例。

移动端高级功能
画中画组件（TUIPIP）
iOS

最近更新时间：2024-04-17 11:49:35

简介

应用场景

高级画中画是在原 基础画中画 上进行的升级，主要支持加密视频画中画、离线播放画中画、从前台无缝切换到画中画的场景，优化了实现方式和逻辑，无需长时间等待，实现真正意义的“秒切”效果。
高级画中画优势：
加密视频画中画：现有播放器的加密播放紧密结合，能够顺畅实现基于加密模板的视频画中画播放，全程无需更换播放器类型。
离线播放画中画：支持本地视频画中画播放，包含普通视频、加密视频等。
“秒切”效果：无需点击切换画中画按钮，退后台即可立马启动画中画，实现真正意义的“秒切”。

环境要求

系统版本：iOS ≥ 14.0、iPad ≥ 9.0。
硬件设备：iPhone 8及以上的设备。
SDK 版本：11.4版本及以上。

集成步骤

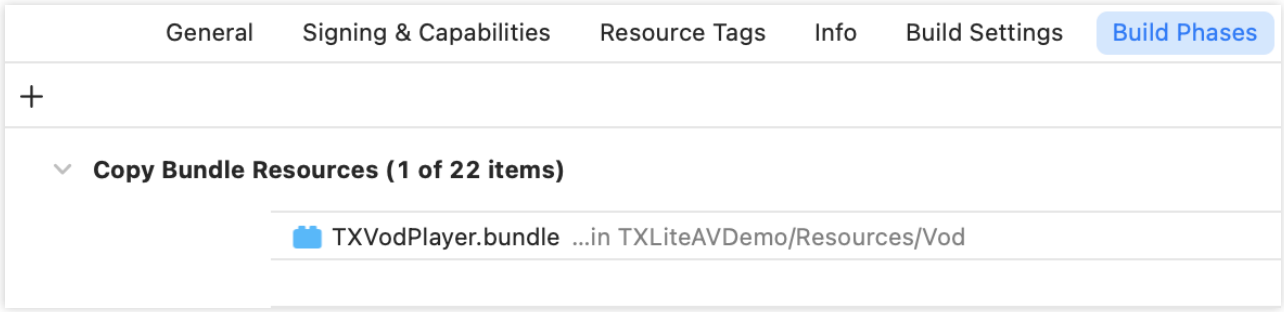
升级版本及配置资源

1. 升级 SDK 版本

高级画中画需要 SDK 配合使用，在使用高级画中画版本功能前需要将 **SDK 的版本升级到11.3及以上的高级版本 或 11.4及以上的基础版本，**否则无法使用。同时，基础画中画 版本和高级画中画版本两者可以兼容性的存在，不会存在功能性冲突。若想升级 SDK 版本，请参见 SDK 集成指引。

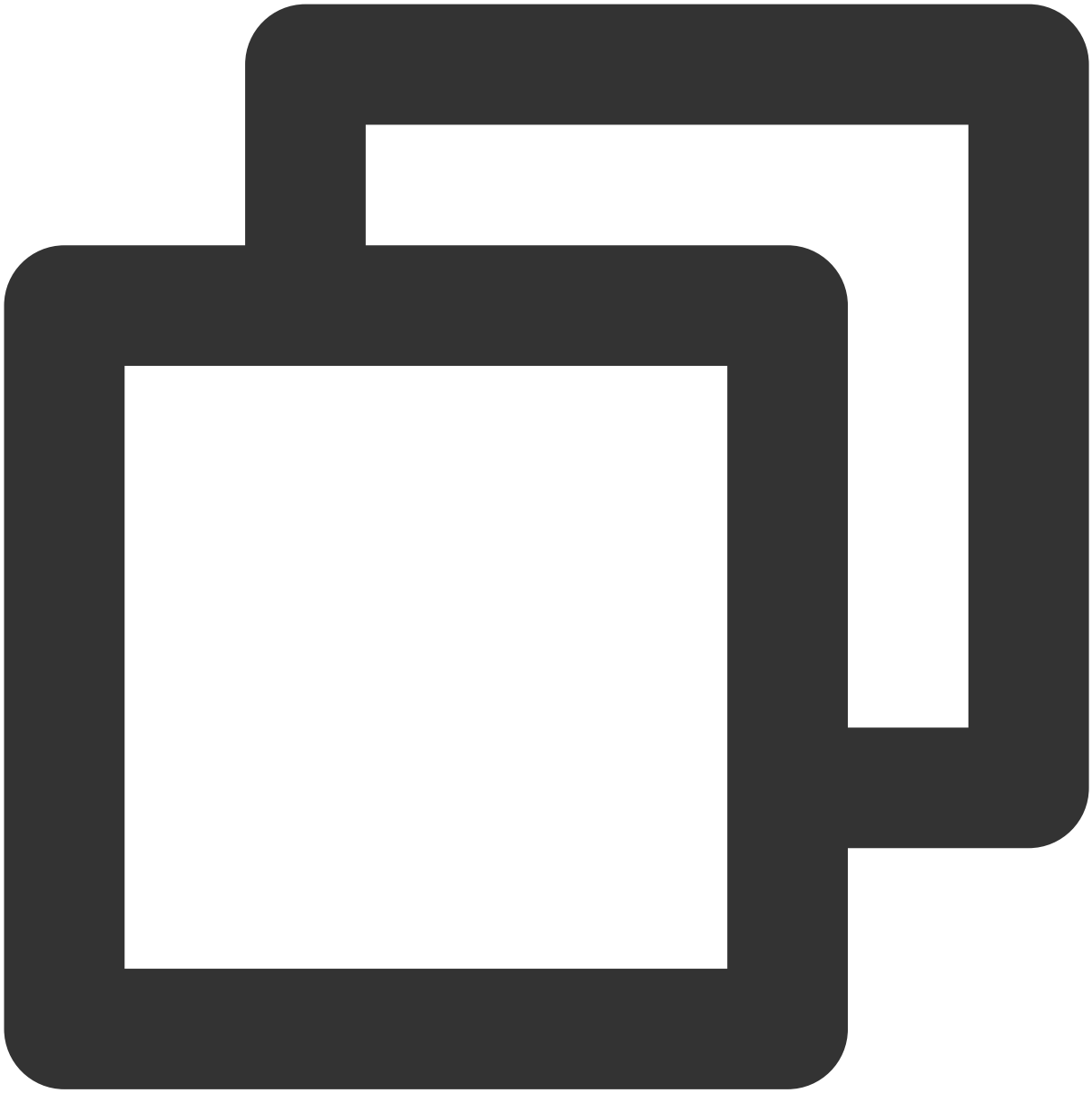
2. 引入 bundle 资源

因为 SDK 内需要使用 TXVodPlayer.bundle 里的资源，需在编译之前将 bundle 文件 下载 引入到项目中，切勿更改 bundle 及其内部使用的资源名称，否则会导致无缝切换画中画失败。



3. 开通播放器高级版 Licence

高级画中画版本需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引获取。若您已获取对应 License，可前往 腾讯云视立方控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请 Player 高级套餐 License，进入画中画将无效。
获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中进行如下设置：

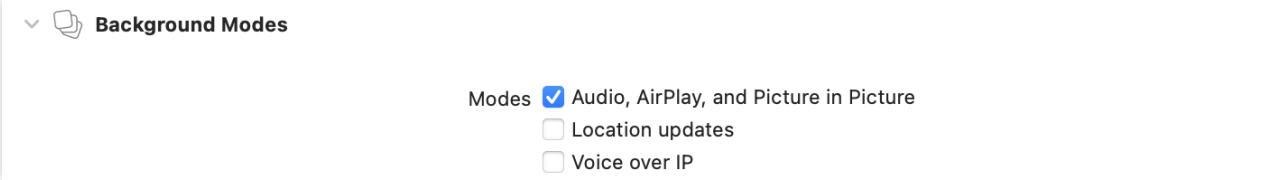


```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    NSString * const licenceURL = @"<获取到的 licenseUrl>";
    NSString * const licenceKey = @"<获取到的 key>";

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

画中画功能快速接入

1. 权限开通

画中画（PictureInPicture）在 iOS 9 就已经推出了，不过之前都只能在 iPad 上使用，iPhone 要使用画中画需更新到 iOS 14 才能使用。 目前腾讯云播放器可以支持应用内和应用外画中画能力，极大的满足用户的诉求。使用前需要开通后台模式，步骤为：**XCode 选择对应的 Target > Signing & Capabilities > Background Modes，勾选“Audio, AirPlay, and Picture in Picture”**。

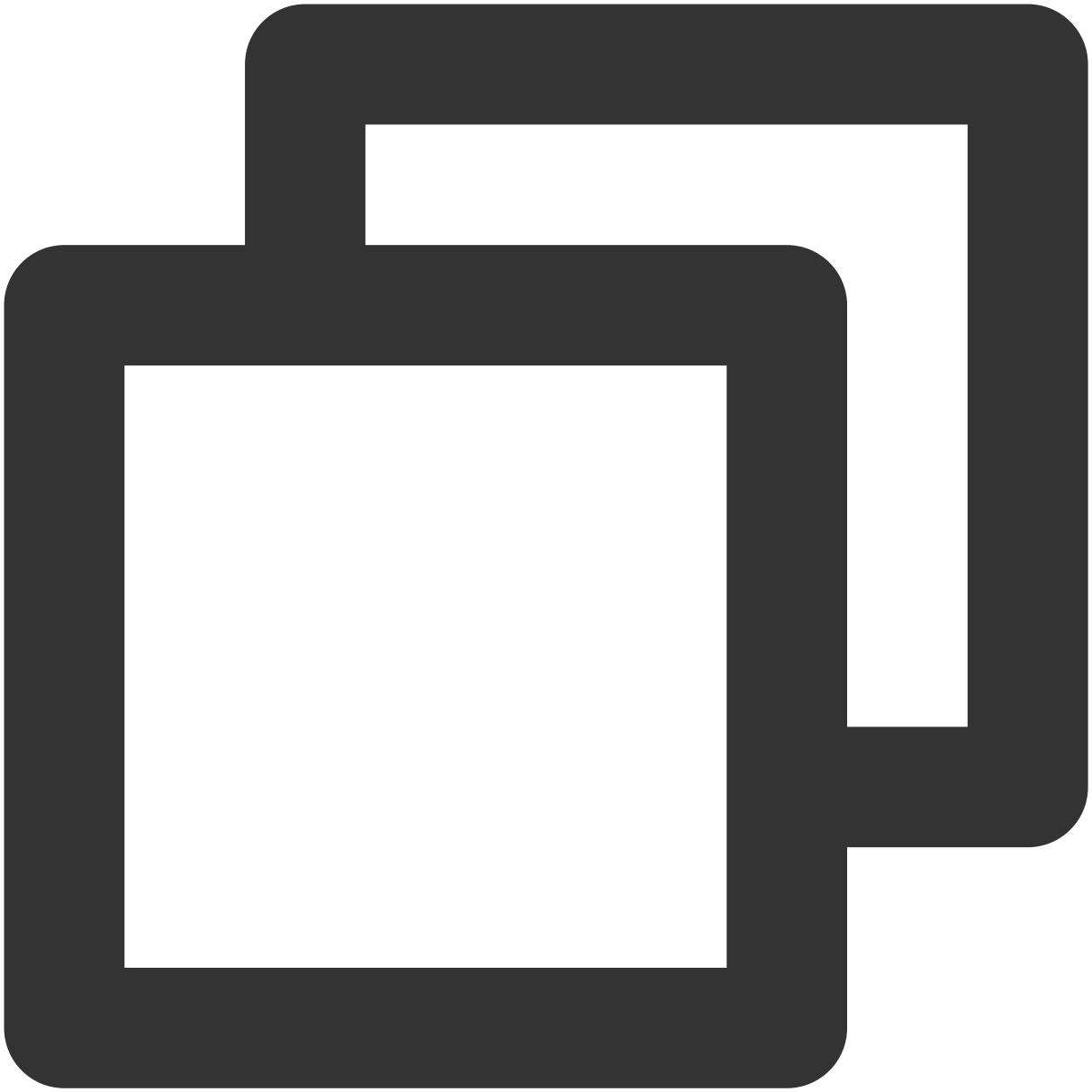


2. 设置配置选项

为了使用自动画中画功能，需要在设置中打开**自动开启画中画**按钮。具体路径为 iPhone 或 iPad上选择：**设置 > 通用 > 画中画 > 自动开启画中画**，选择打开即可。

3. 设置代理

为了便于监听画中画的状态，需要设置 vodDelegate，实现 TXVodPlayListener 中的画中画相关回调。可以根据回调里的各种状态和错误信息，进行相关的业务操作，例如：继续播放、暂停或退出画中画等。



```
/**
 * 画中画状态回调
 *
 * 画中画状态回调
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureStateDidChange:(TX_VOD_PLAYE

/**
 * 画中画错误信息回调
 *
 * 画中画错误信息回调
```

```
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureErrorDidOccur:(TX_VOD_PLAYER
```

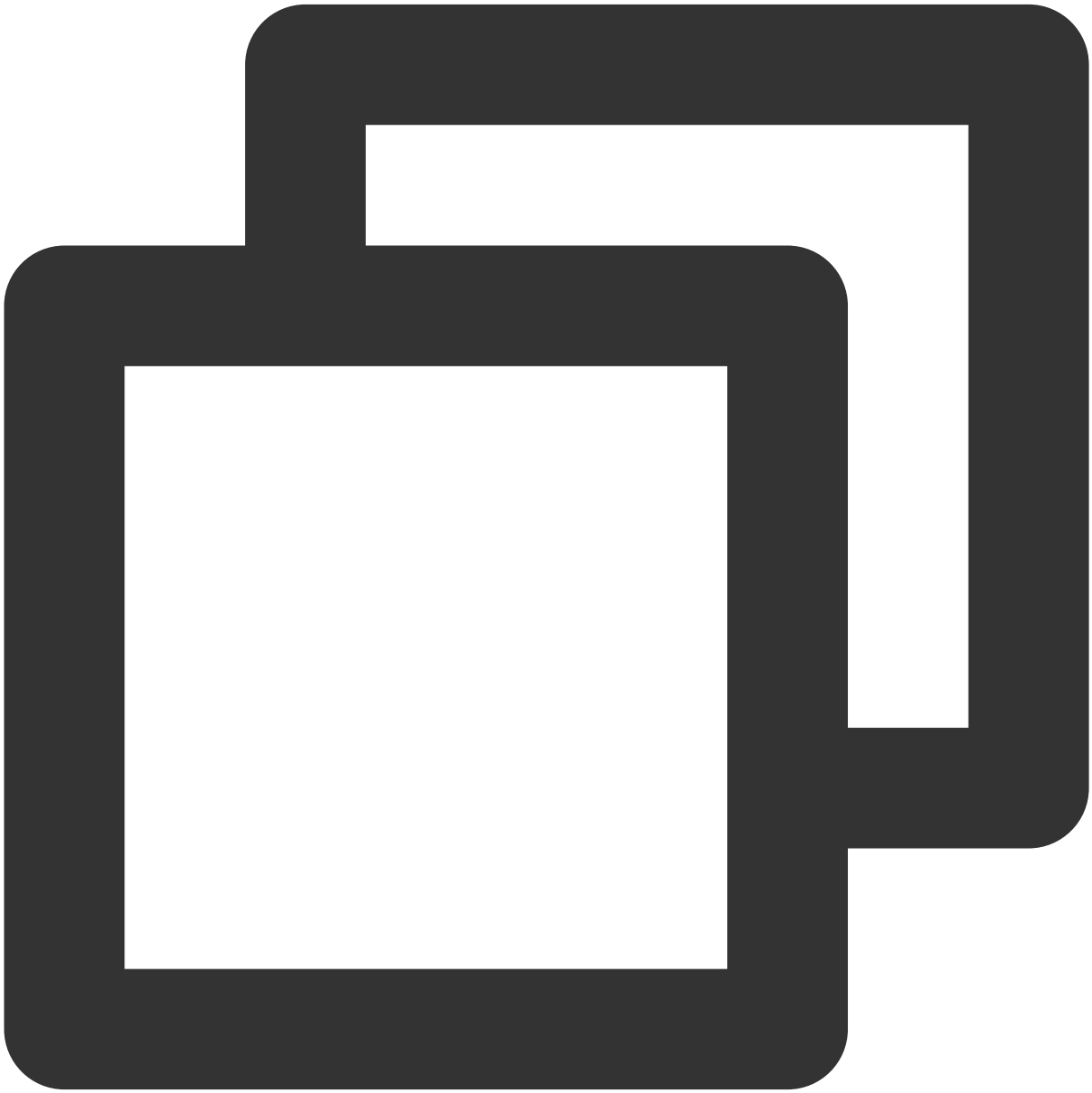
4. 使用画中画能力代码示例

注意：

使用自动画中画功能一定要确保播放器处于播放状态，若播放器是暂停或停止状态时，无法使用自动画中画功能。

isSupportSeamlessPictureInPicture 这个接口，需要在应用程序加载高级版 License 以后才能使用。同时，此接口只能判断设备本身是否支持自动切换画中画，因系统限制，无法判断用户对于自动画中画的设置权限，需自行引导。

播放之前先设置是否允许“自动切换画中画功能”



```
// 1.播放之前先设置“自动切换 Picture-In-Picture功能”是否允许
// YES 表示允许 NO 表示不允许，默认为NO
[TXVodPlayer setPictureInPictureSeamlessEnabled:YES];

// 2、进入画中画
 if (![TXVodPlayer isSupportPictureInPicture]) {
   // 设备不支持画中画直接退出
    return;
 }

// 手动调用进入画中画
 [_vodPlayer enterPictureInPicture];

// 3、退后台操作 如果设备支持无缝切换画中画，退后台不暂停播放。
// 注意：isSupportSeamlessPictureInPicture这个接口，需要在应用程序加载高级版License以后才能
// 是否支持自动切换画中画，因系统限制，无法判断用户对于自动画中画的设置权限，需自行引导。
if ([self.vodplayer isSupportSeamlessPictureInPicture]) {
          // 不做处理
} else {
   // 暂停播放
   [self.vodplayer pause];
}

// 4.退出画中画
[_vodPlayer exitPictureInPicture];
```

短视频组件（TUIPlayerShortVideo）
iOS

最近更新时间：2024-06-17 17:21:56

组件简介

TUIPlayerShortVideo 组件是腾讯云推出的一款性能优异，支持视频极速首帧和流畅滑动，提供优质播放体验的短视频组件。
首帧秒开：首帧时间是短视频类应用核心指标之一，直接影响用户的观看体验。短视频组件通过预播放、预下载、播放器复用和精准流量控制等技术，实现极速首帧、滑动丝滑的优质播放体验，从而提升用户播放量和停留时长。
优异的性能：通过播放器复用和加载策略的优化，在保证极佳流畅度的同时，始终让内存和 CPU 消耗保持在较低的水平。
快速集成：组件对复杂的播放操作进行了封装，提供默认的播放 UI，同时支持 FileId 和 URL 播放，可低成本快速集成到您的项目中。

效果对比

从下方示例视频您可以看到在接入短视频最佳策略前后的对比差异。
优化前有明显的首帧卡顿感。
优化后播放流畅丝滑，优化后起播平均时长达到10毫秒 - 30毫秒。

未优化短视频	优化后短视频

TUIPlayerKit 下载

TUIPlayerKit SDK 和 Demo 可 单击这里 下载。

集成指引

1. 依赖库

TUIPlayerShortVideo 依赖的 SDK 有：

TUIPlayerCore

TXLiteAVSDK ≥ 11.4

SDWebImage

Masonry

2. 环境要求

系统版本：≥ iOS 9.0

开发环境：≥ Xcode 14.0 (推荐使用最新版本)

3. 集成 TUIPlayerCore

解压下载的 TUIPlayerKit 资源包，将 TUIPlayerCore.xcframework 组件 SDK 添加到您项目中 Xcode Project 的合适位置并选择合适的 target，同时勾选 Do Not Embed 。

4. 集成 TUIPlayerShortVideo

解压下载的 TUIPlayerKit 资源包，将 TUIPlayerShortVideo.xcframework 组件 SDK 添加到您项目中 Xcode Project 的合适位置并选择合适的 target，同时勾选 Do Not Embed 。

5. 集成 TXLiteAVSDK

TXLiteAVSDK 集成方法请参见 TXLiteAVSDK 集成指引。

6. 集成 SDWebImage

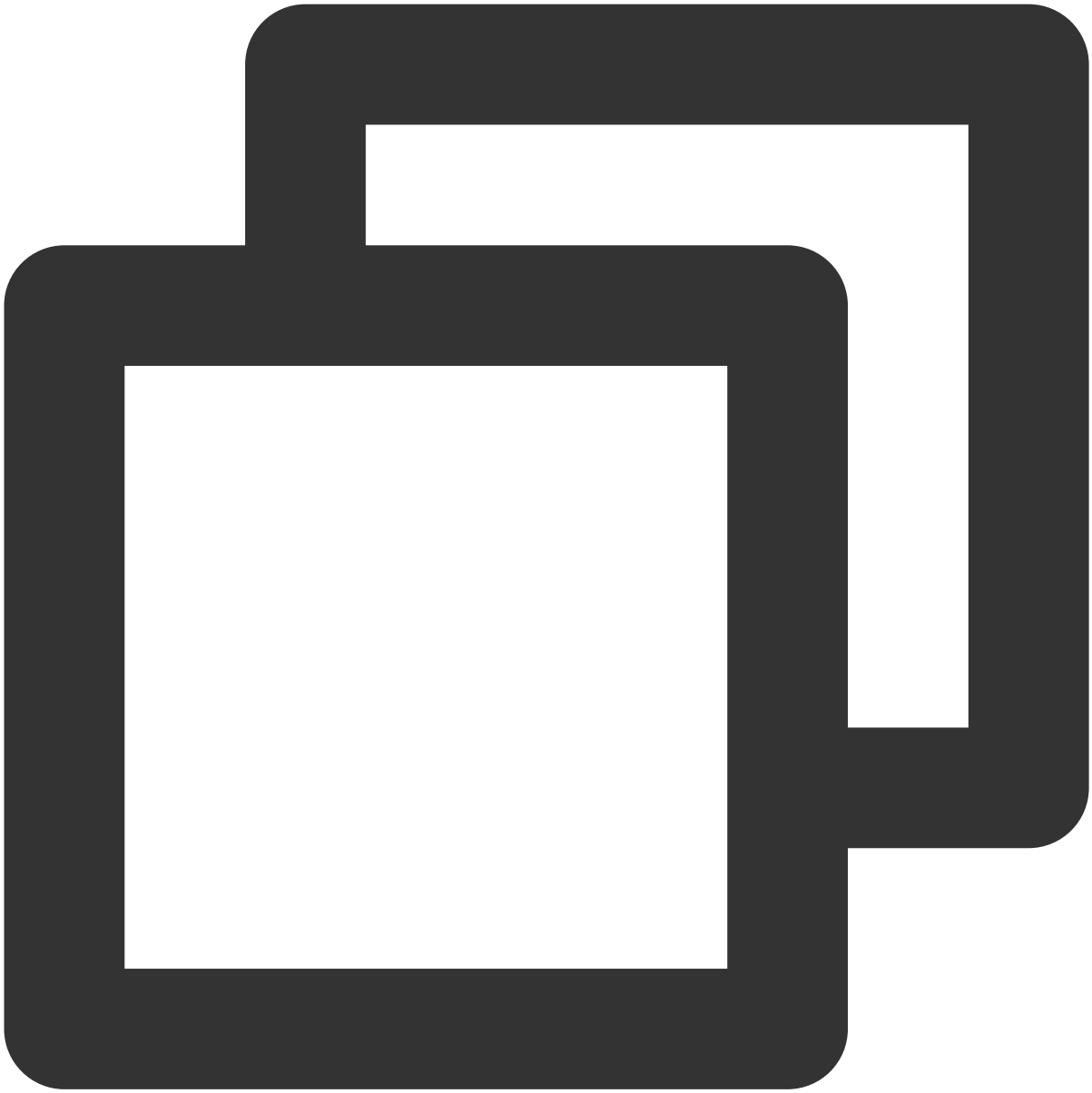
SDWebImage 的下载和集成请参见 GitHub 指引。

7. 集成 Masonry

Masonry 的下载和集成请参见 GitHub 指引。

8. Pod 方式集成

如果您的项目支持 pod，那么也可以通过我们提供的 spec 文件集成，如下：



```
pod 'TUIPlayerShortVideo' ,:path => '../../../SDK/TUIPlayerShortVideoSDK/'
pod 'TUIPlayerCore' ,:path => '../../../SDK/TUIPlayerCoreSDK/'
```

注意：

Path 请根据自己的项目文件路径自行配置。

目前暂不支持远程 Pod 集成。

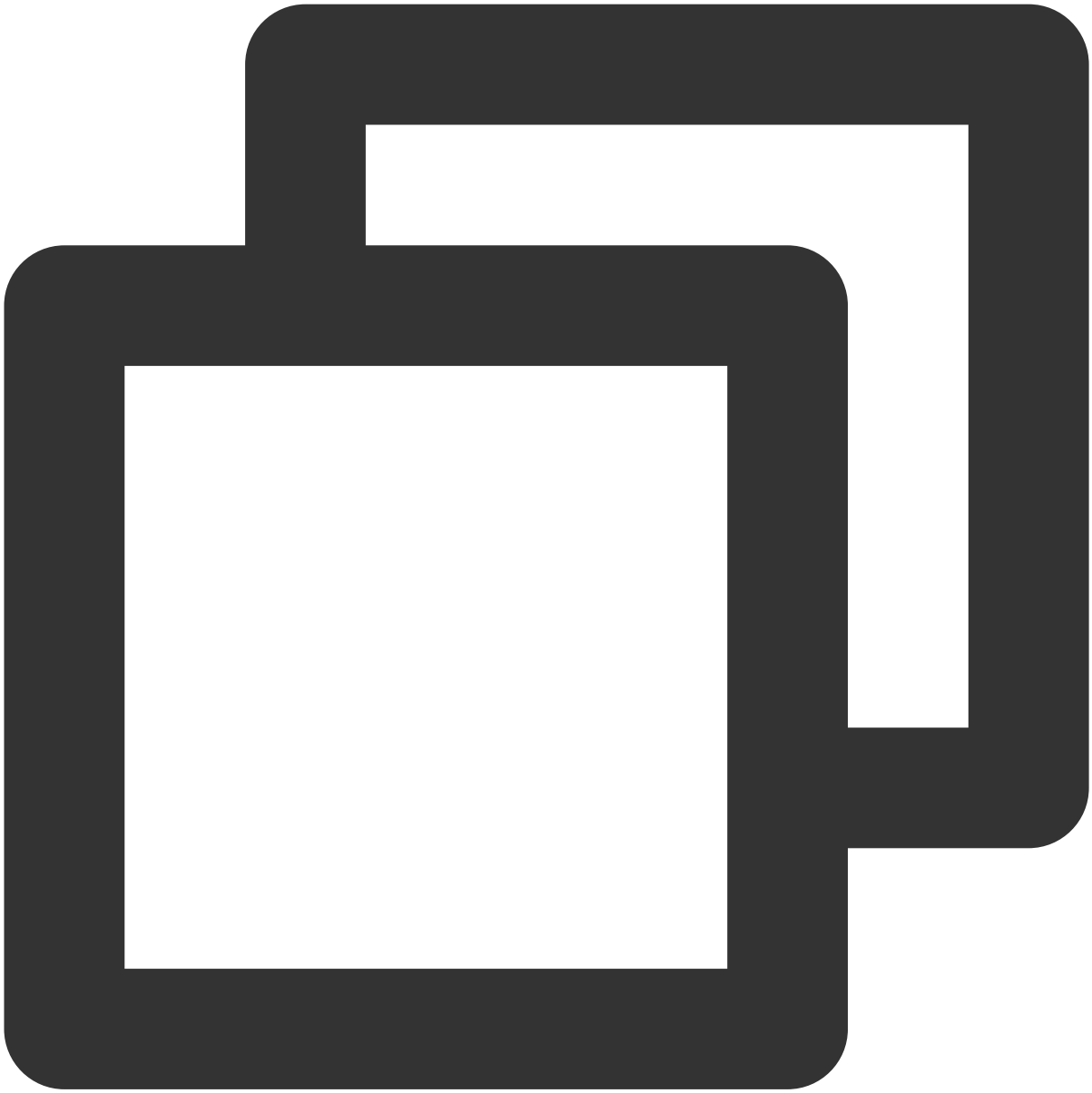
接口使用说明

1. 快速接入

1.1. 配置播放器高级版 Licence

使用 TUIPlayer Kit 组件需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引获取。若您已获取对应 License，可前往 云点播控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请移动端播放器高级版 License，将会出现视频播放失败、黑屏等现象。

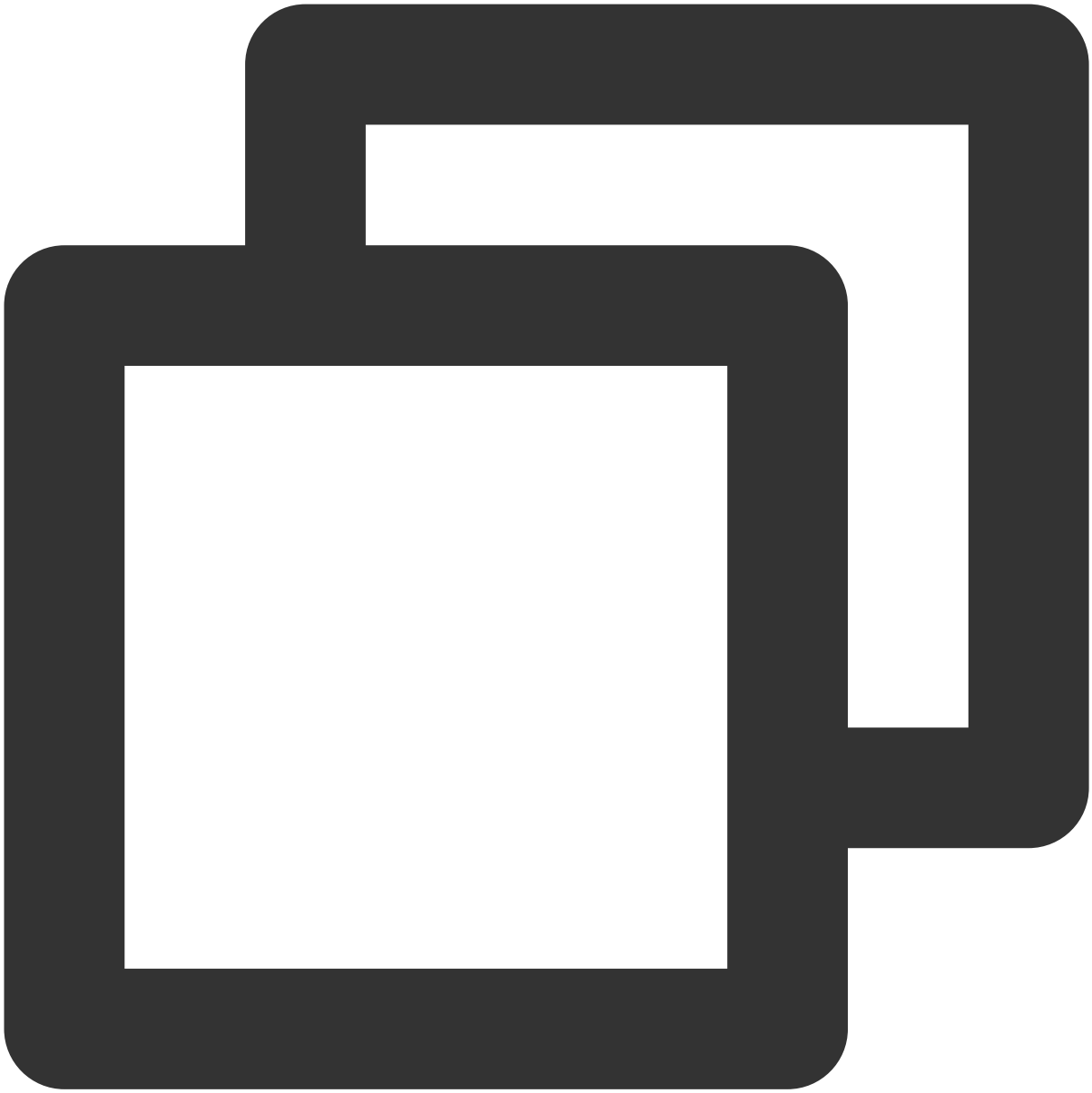
调用相关功能之前，在您的项目中配置 Licence。引用 TUIPlayerCore 模块建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中，做如下配置：



```
NSString * const licenceURL = @"<获取到的licenseUrl>";
NSString * const licenceKey = @"<获取到的key>";
[TXLiveBase setLicenceURL:licenceUrl key:licenceKey];
[[TXLiveBase sharedInstance] setDelegate:self];
```

1.2. 播放

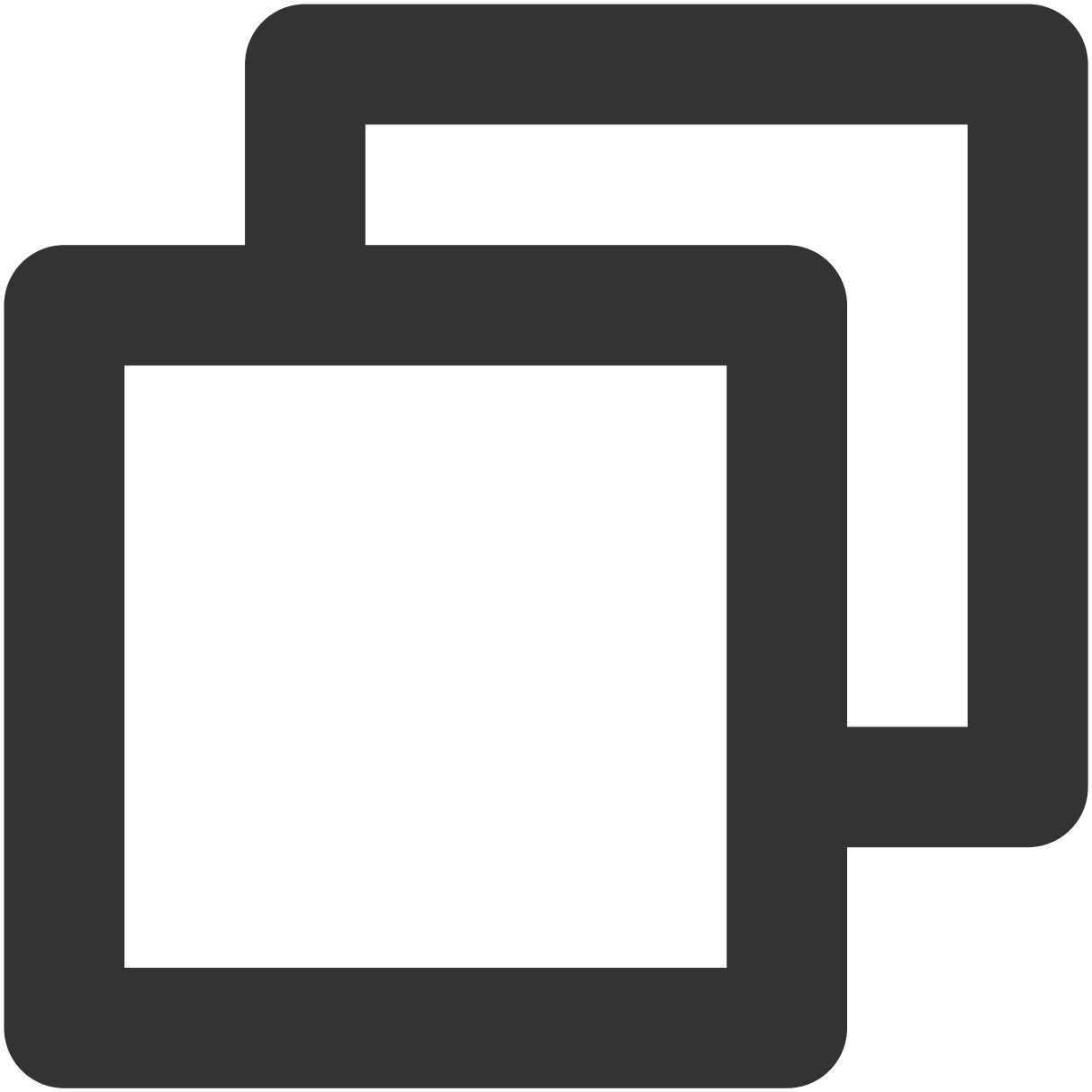
初始化 TUIShortVideoView，举例如下：



```
- (TUIShortVideoView *)videoView {

    if (!_videoView) {
        ///设置自定义UI
        TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager al
        [uiManager setControlViewClass: TUIPlayerShortVideoControlView.class];
        [uiManager setControlViewClass: TUIPSControlLiveView.class viewType:TUI_ITE
        [uiManager setControlViewClass: TUIPSControlCustomView.class viewType:TUI_I
        [uiManager setLoadingView:[[TUIPSDLoadingView alloc] init]];
```

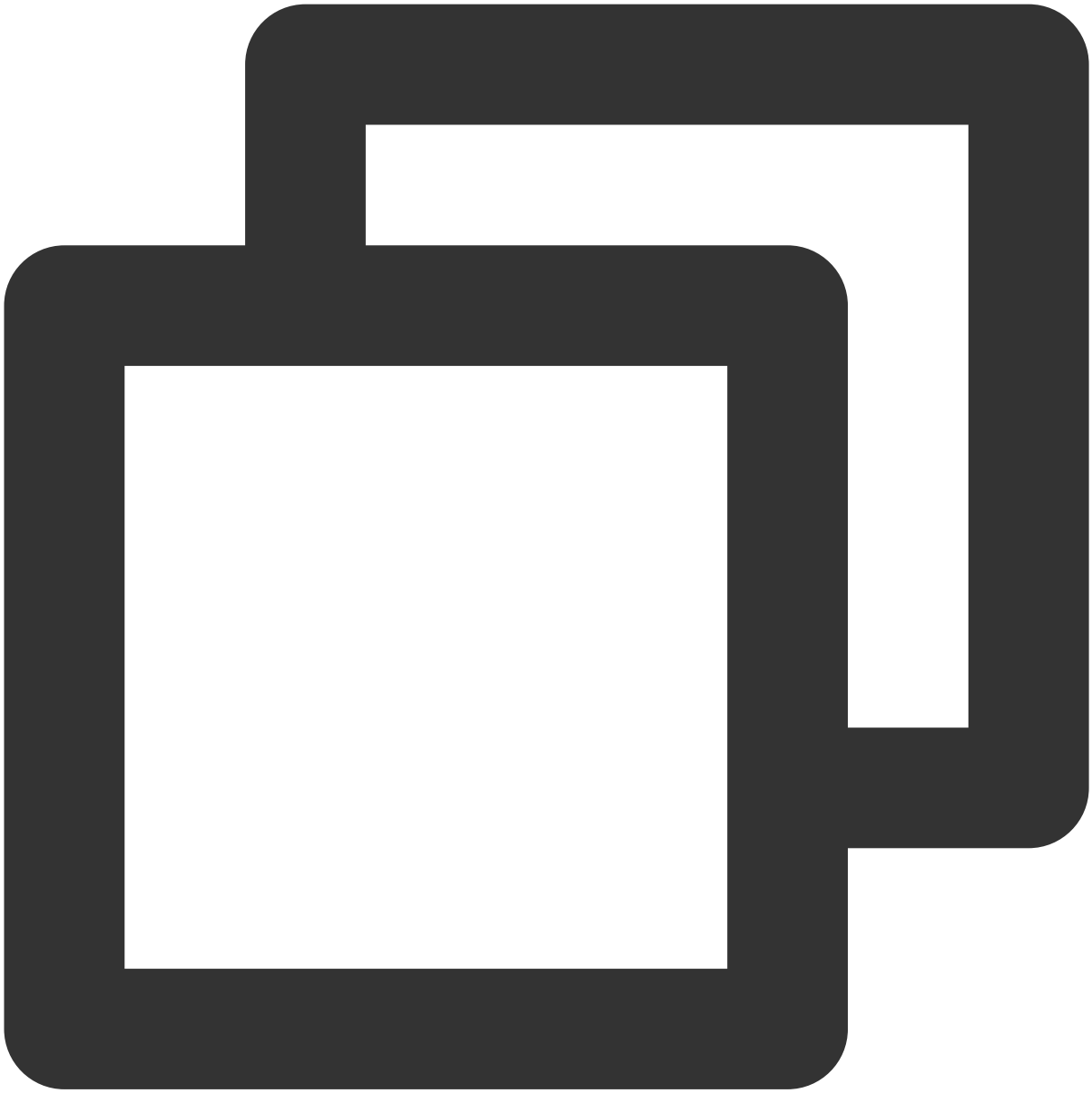
```
        _videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
        _videoView.delegate = self;
        _videoView.customCallbackDelegate = self;
        //_videoView.isAutoPlay = NO;

        // Set your playback strategy
        TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init]
        model.mPreloadConcurrentCount = 1;
        model.preDownloadSize = 1;
        model.enableAutoBitrate = NO;

        // live strategy
        TUIPlayerLiveStrateyModel *liveStrateyModel = [[TUIPlayerLiveStrateyModel a
        [_videoView setShortVideoLiveStrategyModel:liveStrateyModel];
    }
    return _videoView;
}
```

将 TUIShortVideoView 的实例添加到您想呈现的 View 上，参见如下代码：



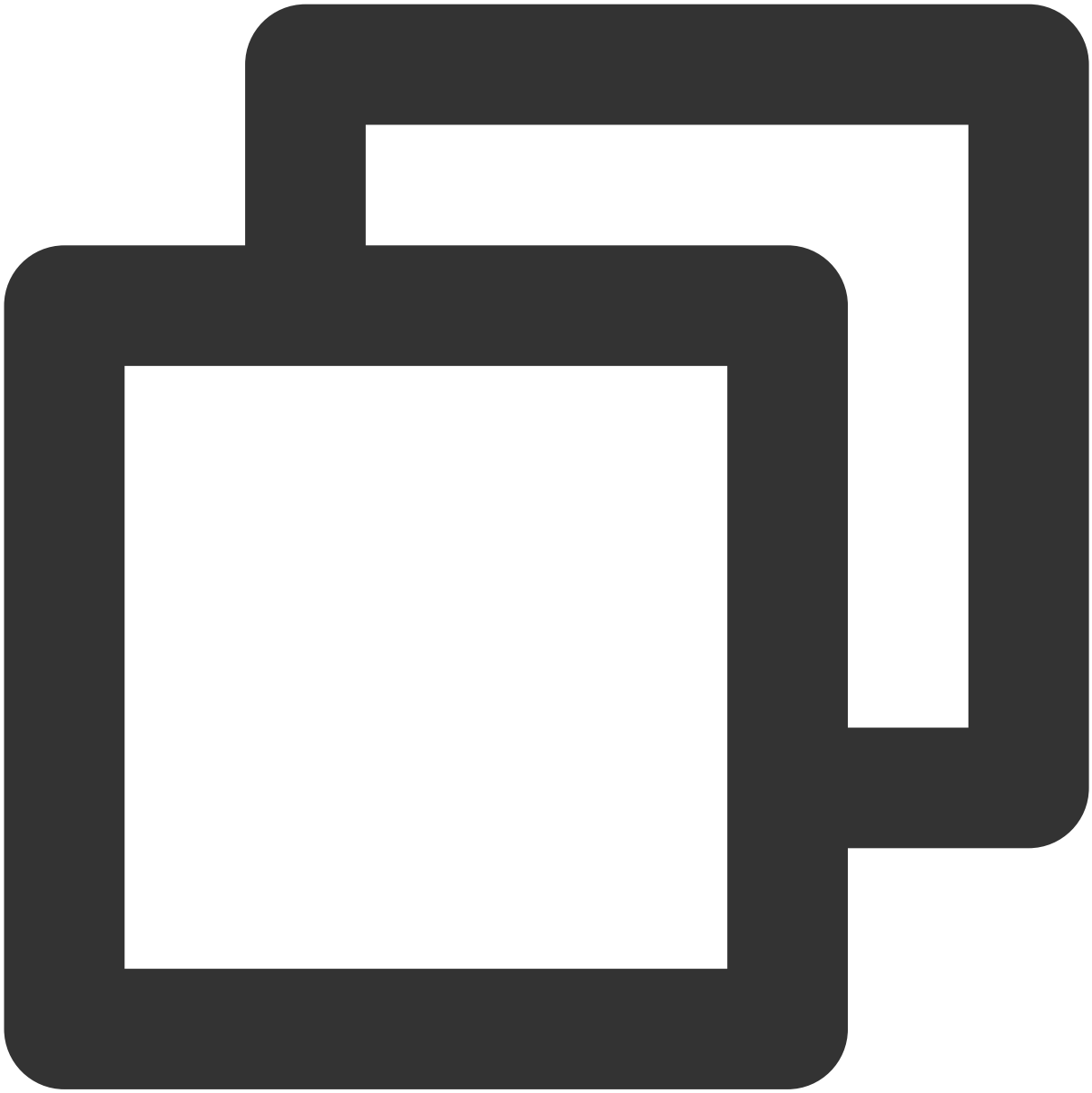
```
videoView.frame = self.view.bounds;
[self.view addSubview:self.videoView];
```

然后添加您的视频数组：



```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///视频数据
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///直播数据
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///自定义数据
/// 这里根据您的业务情况，自行决定每一页的数据量
NSArray *videos1 = @[model1,model2,model3]；
[self.videoView setShortVideoModels:videos1];
```

第一组视频播放完之后您还需要在 TUIShortVideoViewDelegate 的代理方法内继续拼入您的第二组视频数据：



```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///视频数据
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///直播数据
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///自定义数据
/// 这里根据您的业务情况，自行决定每一页的数据量
NSArray *videos2 = @[model1,model2,model3]；
-(void)onReachLast {
    ///这里您可以做数据index索引记录，继续拼入您的第 3 4 5 6....组数据
    [self.videoView appendShortVideoModels:videos2];
}
```

1.3. TUIShortVideoView

TUIShortVideoView 主要接口如下：

参数名称	含义
isAutoPlay	首次加载是否自动播放第一个视频，默认 YES
videos	只读属性，获取当前存在与视频列表中的数据
currentVideoModel	当前正在播放的视频模型
currentVideoIndex	当前正在播放的视频索引
currentPlayerStatus	当前播放器的播放状态
isPlaying	当前播放器是否正在播放
delegate	代理
refreshControl	设置下拉刷新控件
initWithUIManager	初始化（带自定义 UI）
setShortVideoStrategyModel	设置直播播放策略
setShortVideoLiveStrategyModel	设置直播播放策略
setShortVideoModels	首次设置数据源
appendShortVideoModels	追加视频数据源
removeAllVideoModels	删除所有视频数据
setPlaymode	视频播放模式，单个循环或列表循环，默认前者
pause	暂停
resume	继续播放
destoryPlayer	销毁播放器
didScrollToCellWithIndex	跳到指定索引的视频
startLoading	展示 loading 图
stopLoading	隐藏 loading 图
currentPlayerSupportedBitrates	当前正在播放的视频支持的码率
bitrateIndex	获取当前正在播放的码率索引

switchResolution:index:	切换分辨率
pausePreload	暂停预加载
resumePreload	恢复预加载
getDataManager	获取数据管理器
getVodStrategyManager	获取点播策略管理器
getLiveStrategyManager	获取直播策略管理器

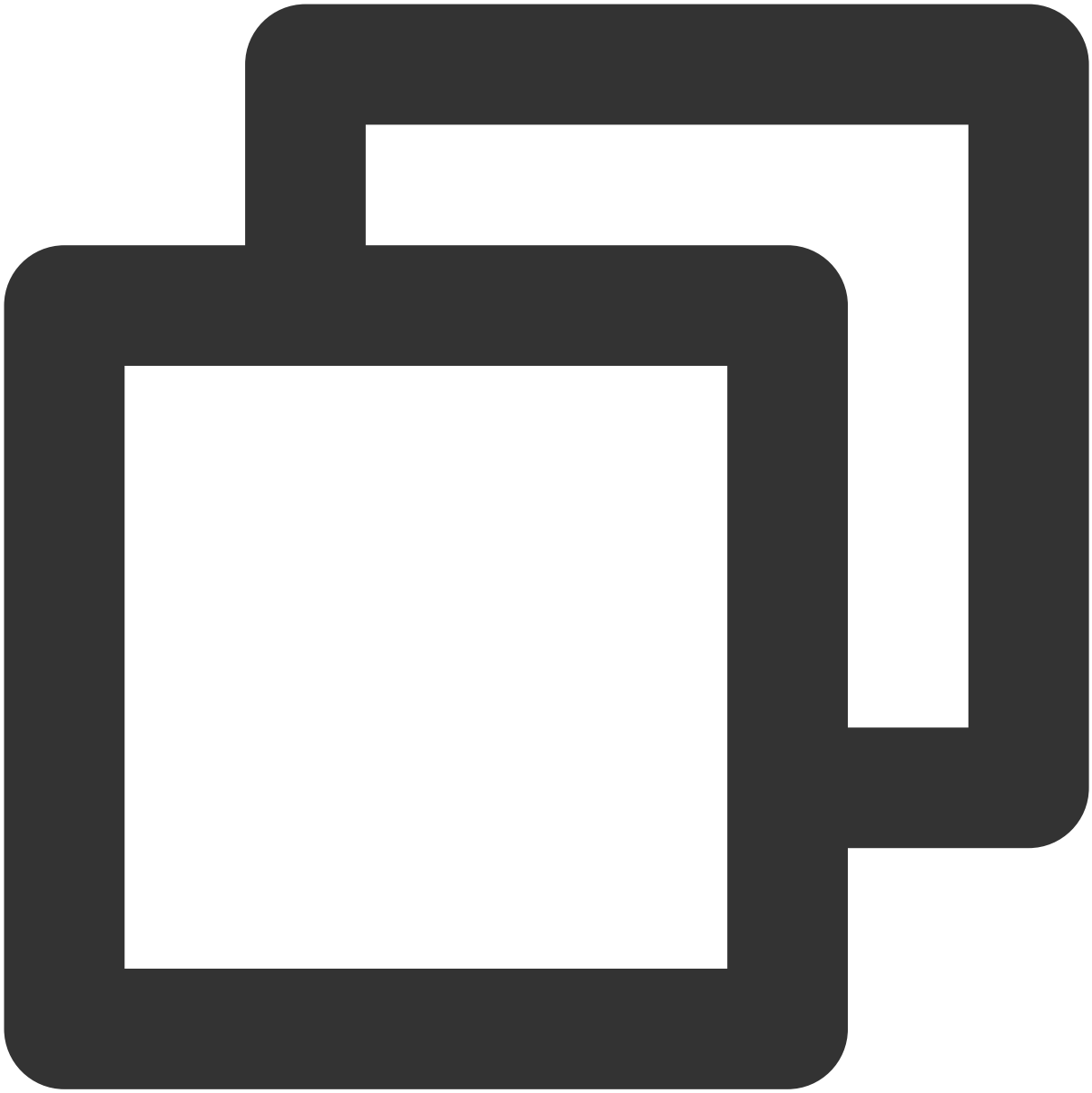
2. 全局配置

您可以通过 TUIPlayerConfig 模型在 TUIPlayerCore 里设置一些全局配置。

TUIPlayerConfig 主要参数参见下表：

参数名称	含义
enableLog	是否允许打印日志，默认 NO

然后通过 TUIPlayerCore 进行全局配置：



```
TUIPlayerConfig *config = [TUIPlayerConfig new];
config.enableLog = YES;
[[TUIPlayerCore shareInstance] setPlayerConfig:config];
```

3. 播放器策略配置

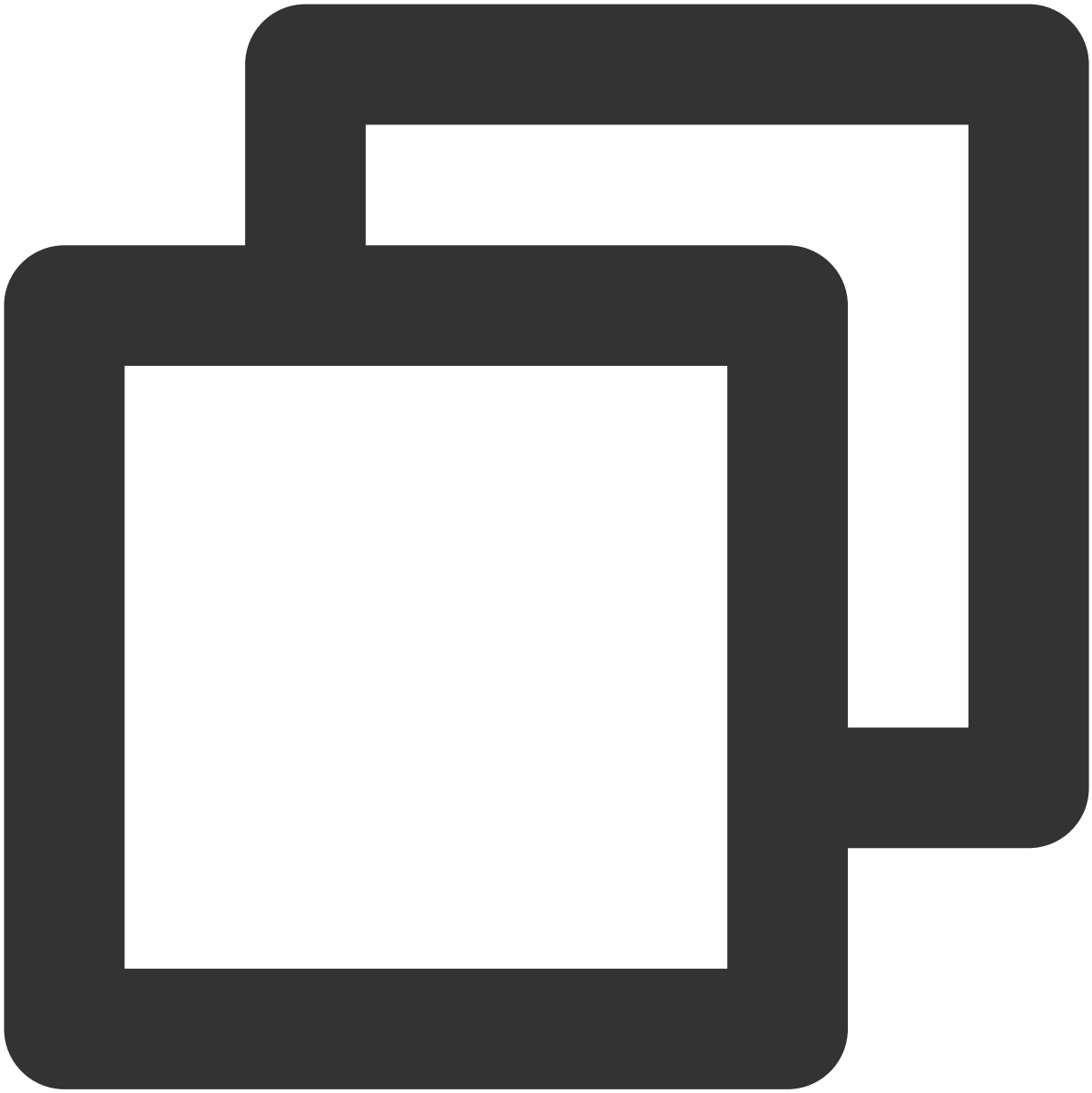
3.1. 点播播放策略设置

您可以通过 TUIPlayerVodStrategyModel 模型配置点播的播放策略。

TUIPlayerVodStrategyModel 主要参数参见下表：

参数名称	含义
mPreloadConcurrentCount	视频缓存个数，默认3
mPreloadBufferSizeInMB	预播放大小，单位 MB，默认0.5MB
preferredResolution	偏好分辨率，默认720 * 1280
progressInterval	进度条回调间隔时长，单位毫秒，默认500ms
renderMode	画布填充样式，默认图像适应屏幕，保持画面完整
extInfoMap	额外参数，预留
enableAutoBitrate	是否开启自适应码率，默认 NO
mediaType	设置媒资类型
maxBufferSize	最大预加载大小，单位 MB，默认10MB，此设置会影响 playableDuration，设置越大，提前缓存的越多
mResumeModel	续播模式，默认 TUI_RESUM_MODEL_NONE
preDownloadSize	预下载大小，单位 MB，默认1MB
enableAccurateSeek	是否精确 seek，默认 YES。开启精确后 seek，seek 的时间平均多出 200ms
audioNormalization	音量均衡。响度范围：-70～0(LUFS)。此配置需要 LiteAVSDK 11.7 及以上版本支持。 以下几种常量供参考使用： 关：AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h) 开（标准响度）：AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h) 开（低响度）：AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h) 开（高响度）：AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h) 默认值为 AUDIO_NORMALIZATION_OFF。
isLastPrePlay	是否保留上一个预播放，默认 NO
subtitleRenderModel	字幕样式

然后进行播放器策略配置：



```
TUIPlayerStrategyModel *model = [[TUIPlayerStrategyModel alloc] init];
model.mPreloadConcurrentCount = 1;
model.preDownloadSize = 1;
model.enableAutoBitrate = NO;
model.mRenderMode = TUI_RENDER_MODE_FILL_SCREEN;
model.mResumeModel = TUI_RESUM_MODEL_LAST;
[_videoView setShortVideoStrategyModel:model];
```

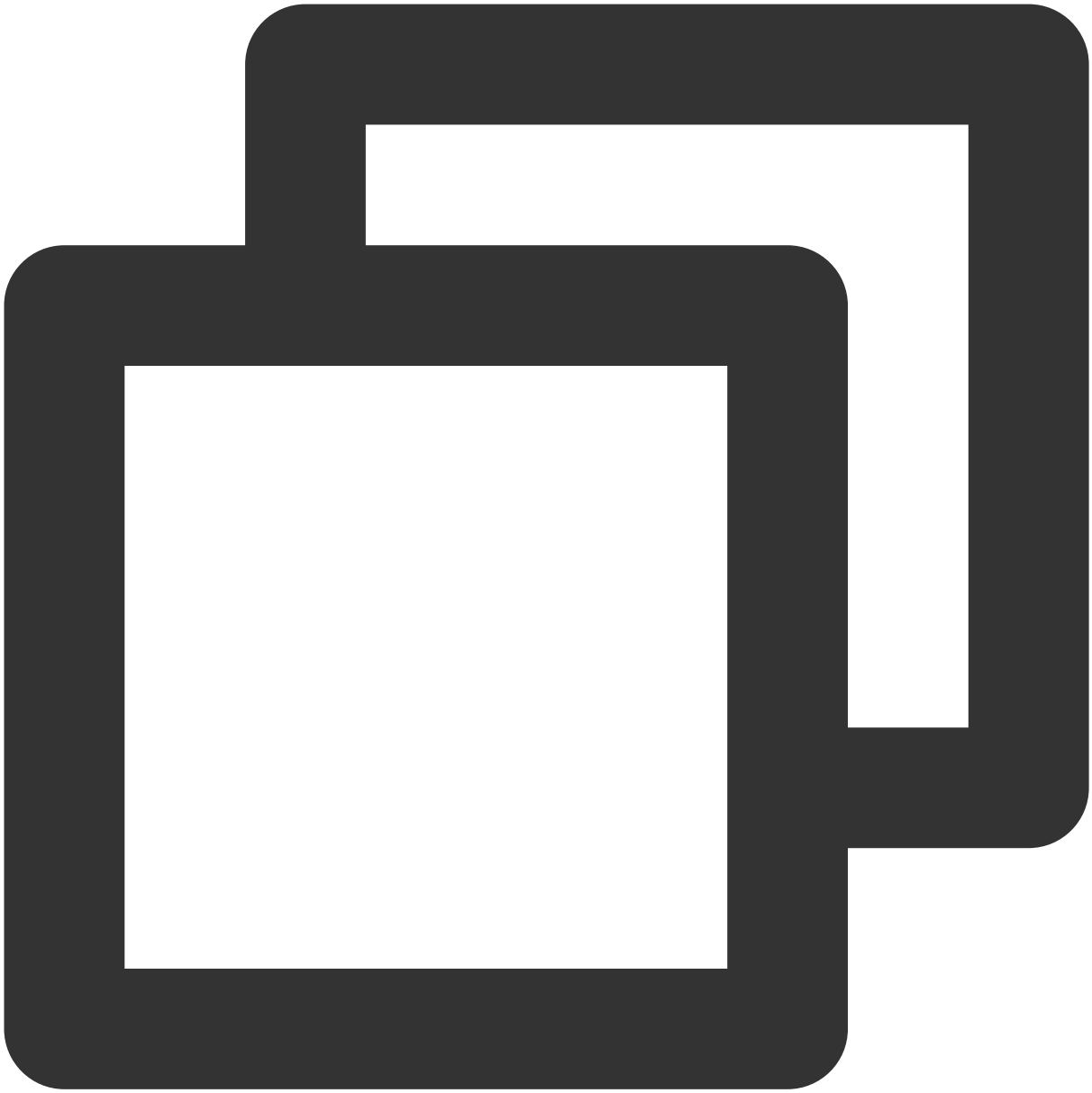
3.2. 直播播放策略设置

您可以通过 TUIPlayerLiveStrategyModel 模型配置点播的播放策略。

TUIPlayerLiveStrategyModel 主要参数参见下表：

参数名称	含义
isLastPrePlay	是否保留上一个预播放，默认 NO
mRenderMode	画布填充样式，默认 V2TXLiveFillModeFill
enablePictureInPicture	YES：开启画中画功能；NO：关闭画中画功能。**默认值**：NO。
volume	播放器音量，取值范围0 - 100。**默认值**：100。
maxAutoAdjustCacheTime	播放器缓存自动调整的最大时间，单位秒，取值需要大于0，默认值：5。
minAutoAdjustCacheTime	播放器缓存自动调整的最小时间，单位秒，取值需要大于0，默认值为1。
isShowDebugView	是否显示播放器状态信息的调试浮层**默认值**：NO。

然后进行播放器策略配置：

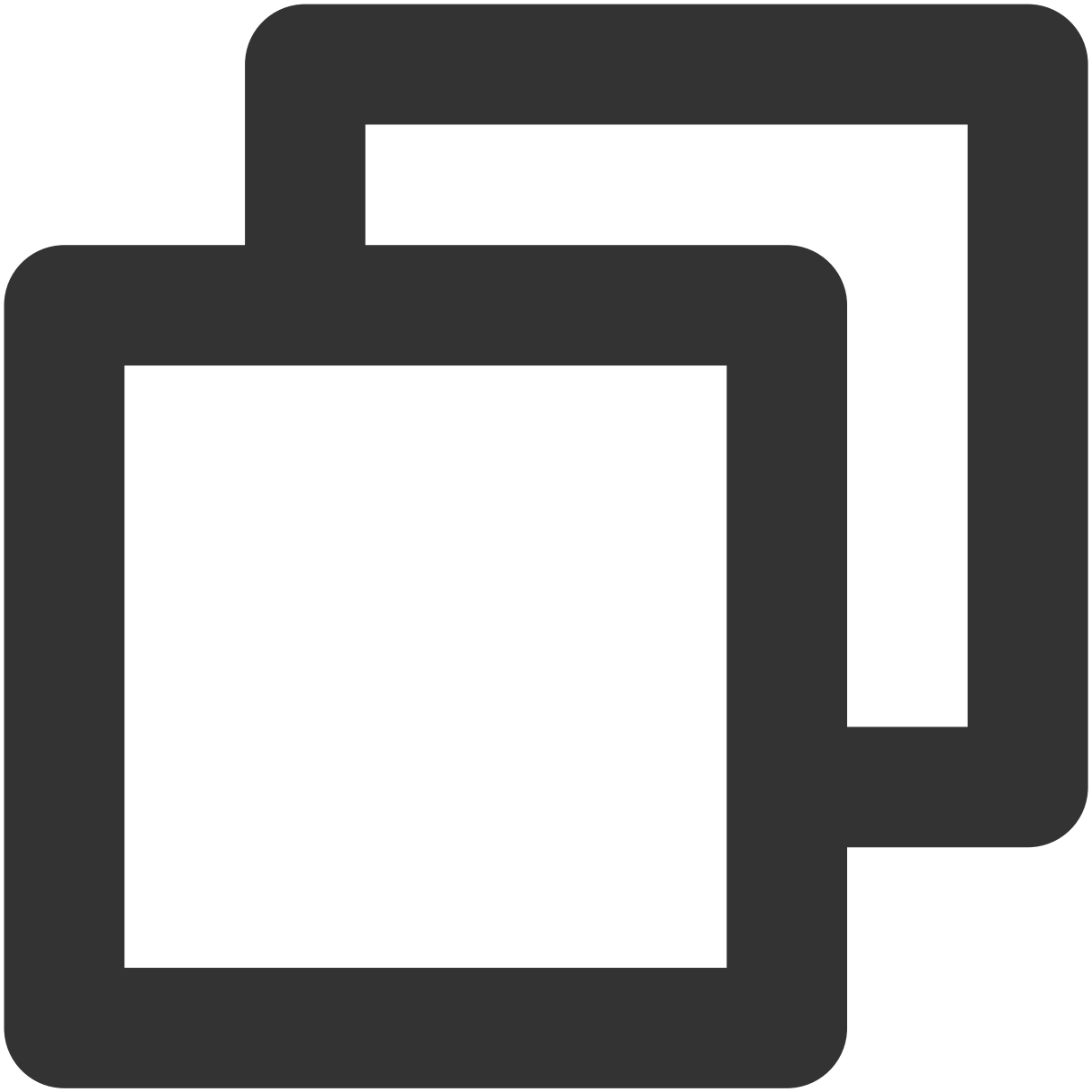


```
TUIPlayerLiveStrategyModel *liveStrategyModel = [[TUIPlayerLiveStrategyModel alloc]
[_videoView setShortVideoLiveStrategyModel:liveStrategyModel];
```

3.3. 动态策略调整

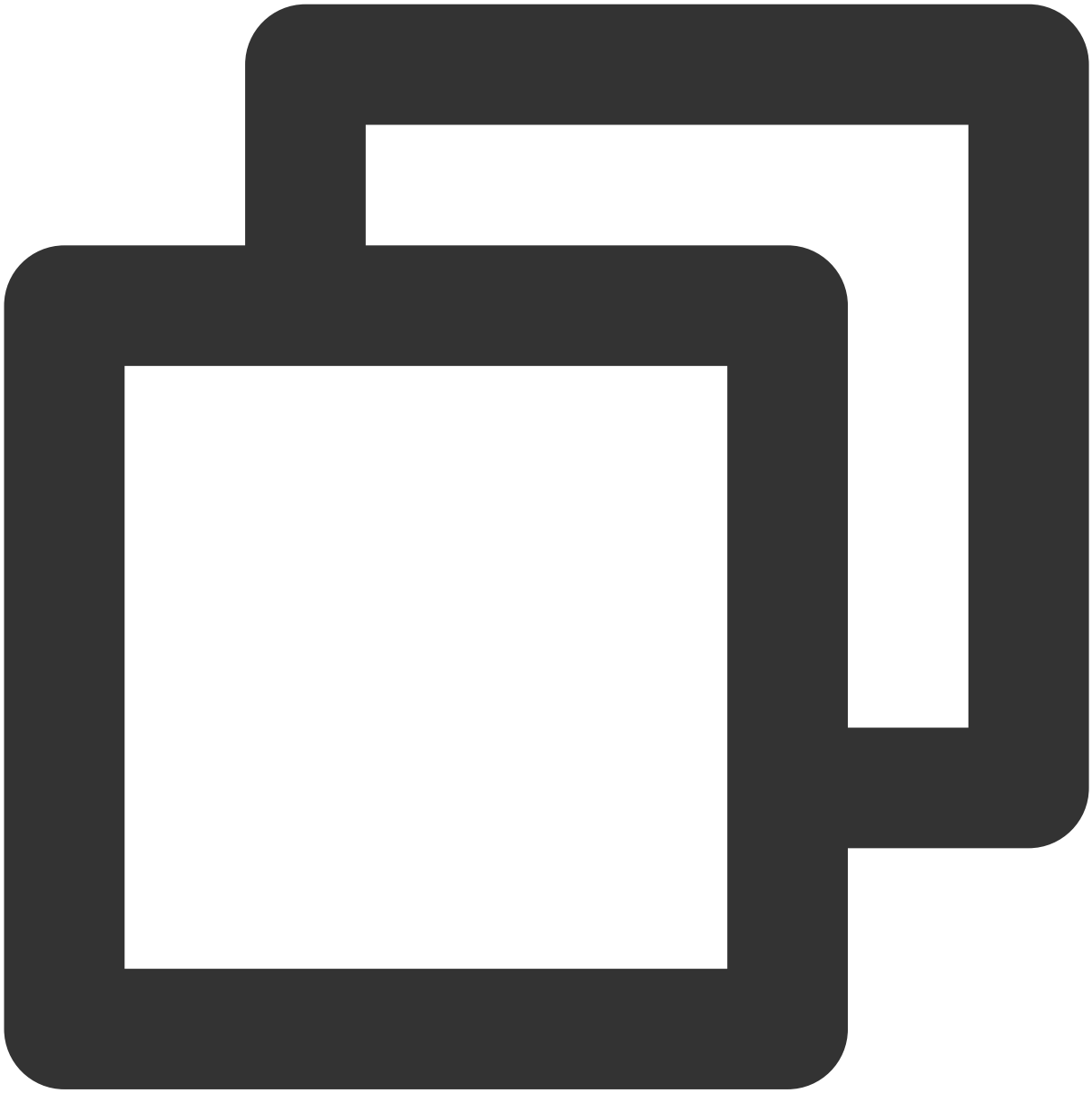
点播和直播的策略均支持动态的调整，步骤如下：

1、通过TUIShortVideoView获取直播&点播策略管理类。



```
TUIPlayerVodStrategyManager *VodStrategyManager = [_videoView getVodStrategyManager
TUIPlayerVodStrategyManager *LiveStrategyManager = [_videoView getLiveStrategyManag
```

2、通过VodStrategyManager 和 LiveStrategyManager调整播放策略。



```
[VodStrategyManager setRenderMode:TUI_RENDER_MODE_FILL_EDGE];
[LiveStrategyManager setRenderMode:V2TXLiveFillModeFill];
```

4.数据管理

4.1. 数据模型

TUIShortVideoView 的原始数据模型包括：

参数说明	含义

TUIPlayerDataModel	基本的数据类型
TUIPlayerVideoModel	点播数据类型，继承于 TUIPlayerDataModel
TUIPlayerLiveModel	直播数据类型，继承于 TUIPlayerDataModel

TUIPlayerDataModel

参数说明	含义
modelType	模型类型
extInfo	extInfo 数据发生改变的 block
onExtInfoChangedBlock	直播数据类型
extInfoChangeNotify	通知 extInfo 数据发生改变
asVodModel	强转为 TUIPlayerVideoModel 类型
asLiveModel	强转为 TUIPlayerLiveModel 类型

TUIPlayerVideoModel

参数说明	含义
videoUrl	视频 Url 地址
coverPictureUrl	封面图
duration	视频时长
appId	appid
fileId	视频的 fileId
pSign	签名字串
subtitles	字幕信息
config	视频的单独配置，详情请看 TUIPlayerVideoConfig

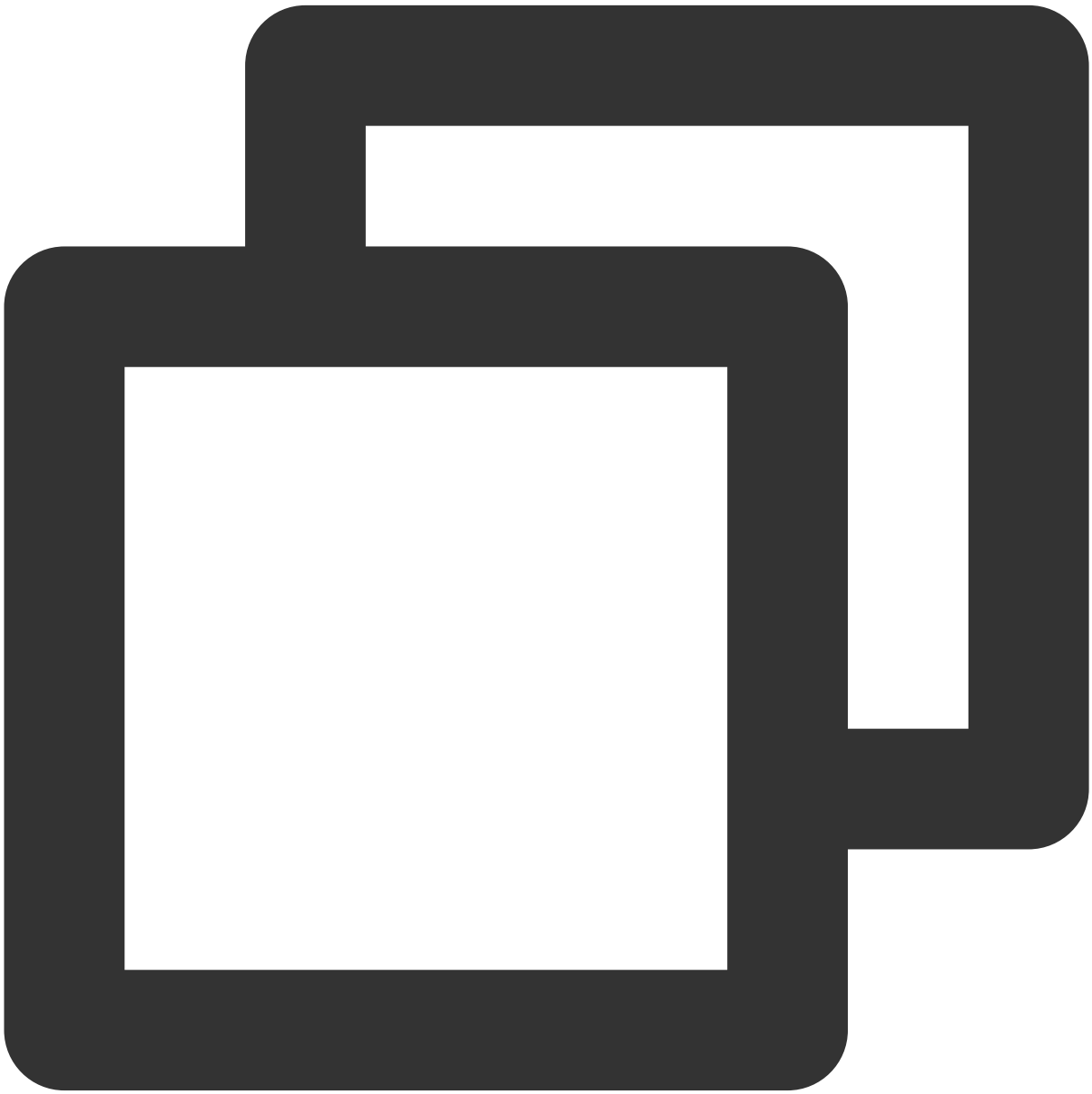
TUIPlayerLiveModel

参数说明	含义
liveUrl	直播 Url

coverPictureUrl	封面图

4.2. 模型的构建

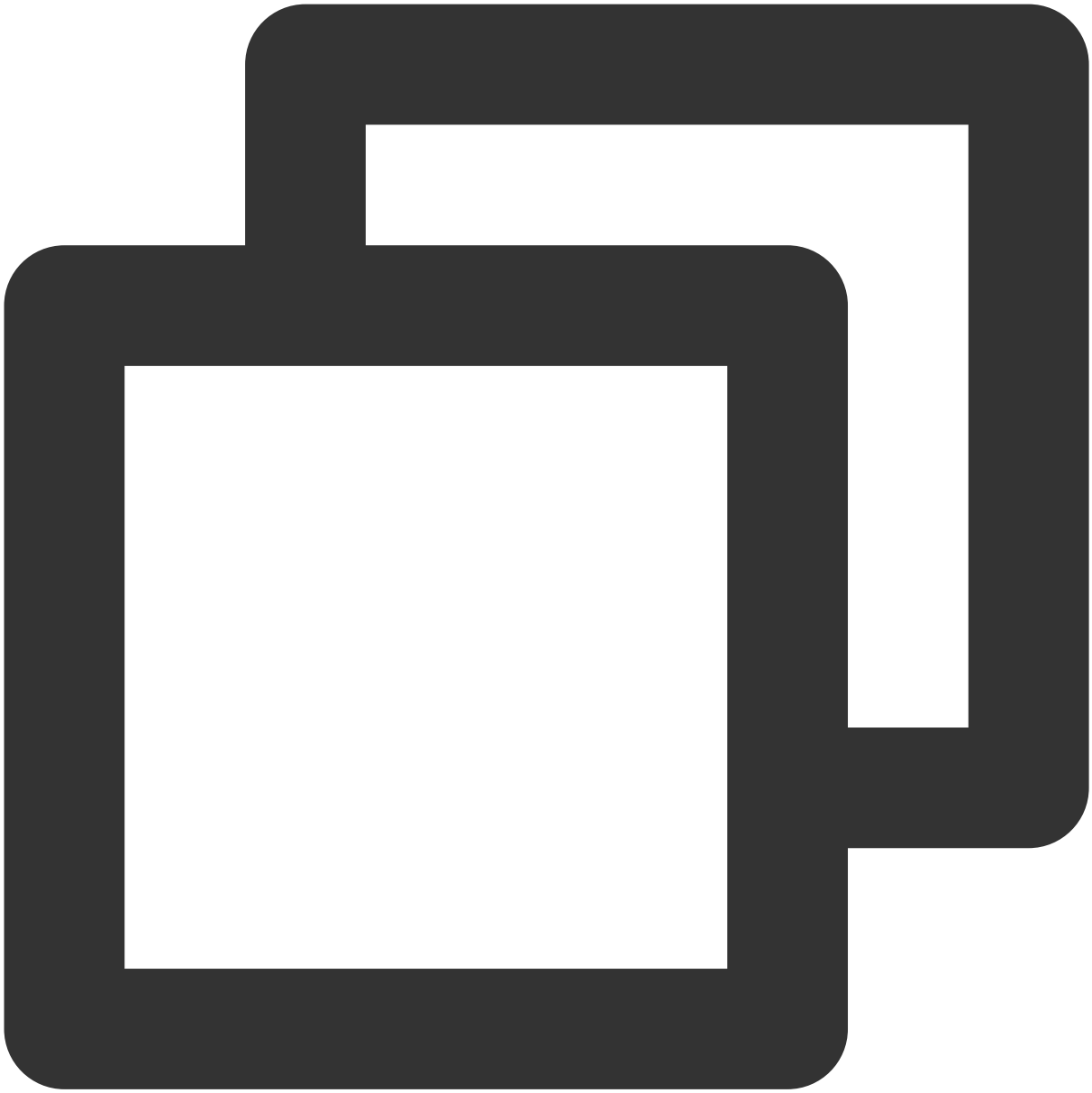
构建一组点播数据模型



```
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.videoUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
model.duration = @"xxxx";
```

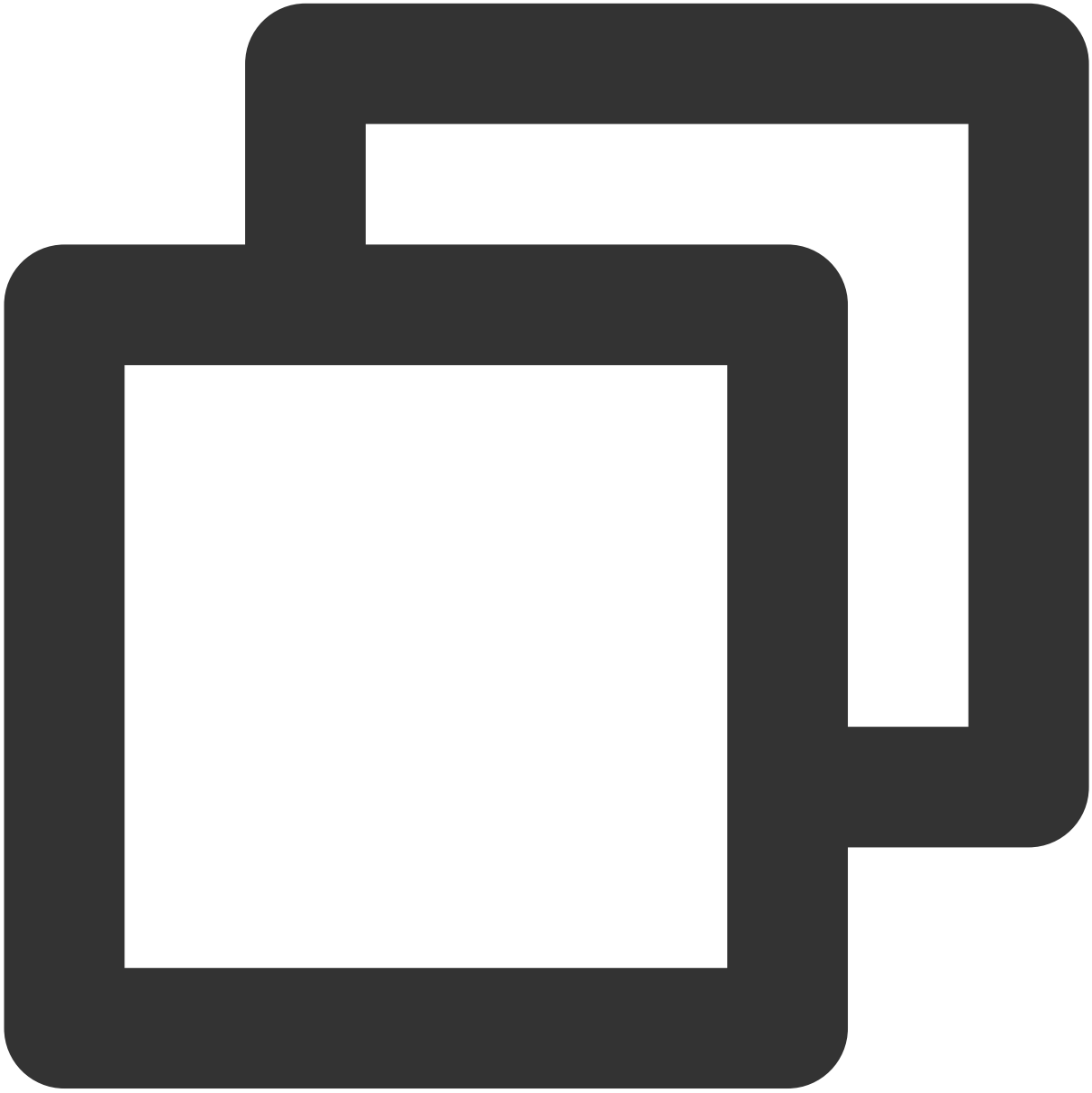
```
model.appId = @"xxxx";
model.fileId = @"xxxx";
model.pSign = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"titile":@"This is a vod broadcast interface",
    @"des":@"This is a vod broadcast interface"
};
model.extInfo = extr;
[modelArray addObject:model];
```

构建一组直播数据模型



```
TUIPlayerLiveModel *model = [[TUIPlayerLiveModel alloc] init];
model.liveUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"liveTitile":@"This is a live broadcast interface",
    @"liveDes":@"This is a live broadcast interface"
};
model.extInfo = extr;
```

构建一组其他类型的数据模型



```
/// 1 轮播图
TUIPlayerDataModel *model = [[TUIPlayerDataModel alloc] init];
NSDictionary *extr = @{
    @"images":@"xxxx",
    @"url":@"https://cloud.tencent.com",
    @"titile":@"This is a picture carousel display interface",
    @"des":@"This is a picture carousel display interface",
    @"name":@"@Mars",
    @"type":@"imageCycle"
```

```
};
model.extInfo = extr;
[modelArray insertObject:model atIndex:1];

/// 2 图文广告
TUIPlayerDataModel *model1 = [[TUIPlayerDataModel alloc] init];
NSDictionary *extr1 = @{
    @"adUrl":@"https://cloud.tencent.com",
    @"adUrl":@"https://cloud.tencent.com/document/product",
    @"adTitile":@"This is a web display interface",
    @"adDes":@"This is a web display interface",
    @"name":@"@Mars",
    @"type":@"ad"
};
model1.extInfo = extr1;
[modelArray insertObject:model1 atIndex:1];
```

注意：

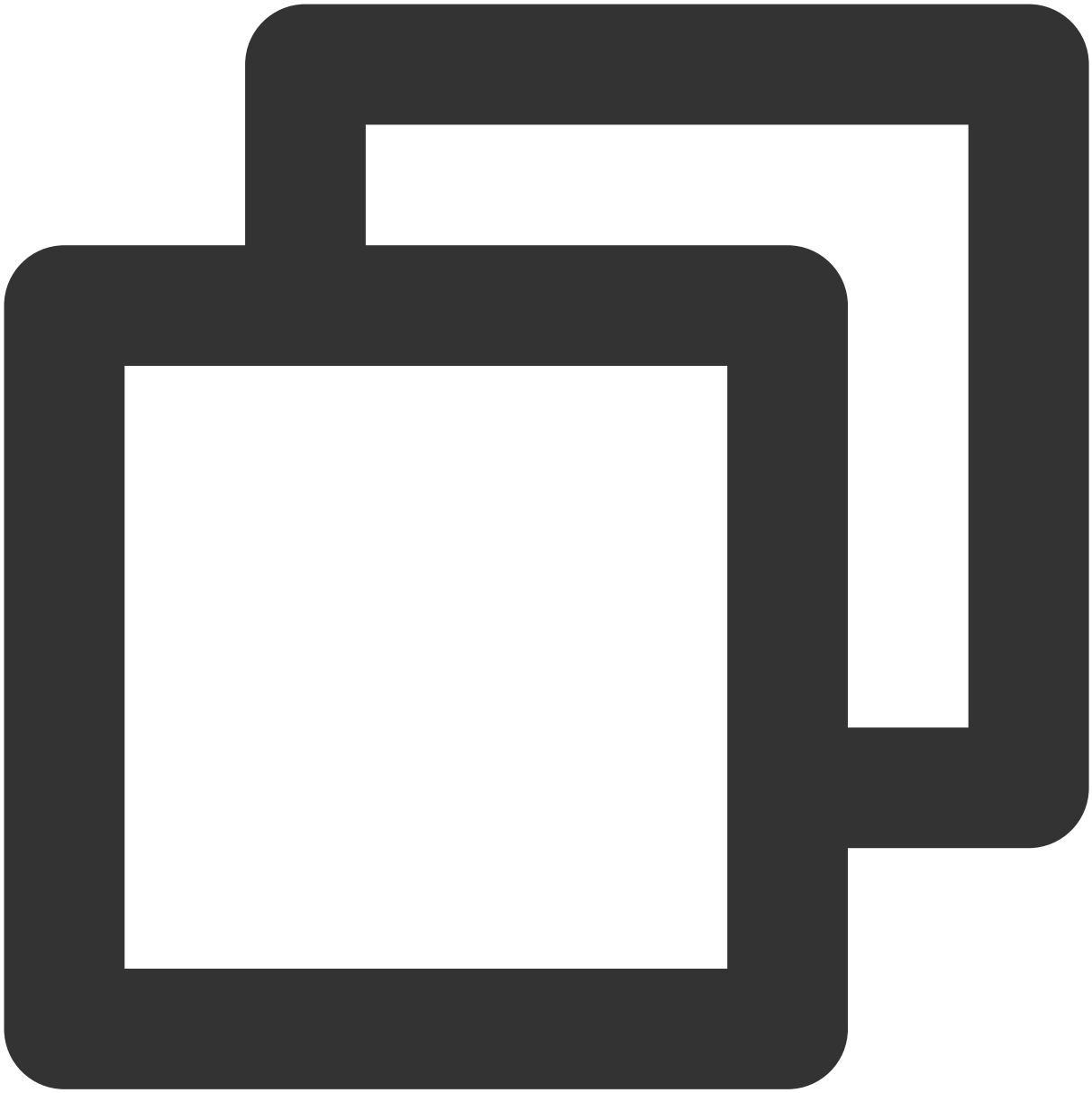
TUIPlayerDataModel 是一个大类，适用于所有非点播和直播的数据类型。

用户可以通过 extInfo 业务字短继续做更细的分类，例如上图通过 TUIPlayerDataModel 分别构建出了“轮播图”和“图文广告”两种类型的数据，可通过 extInfo/type 去自行分类。

extInfo 为一个灵活字短，用户可以随意去设计自己所需要的数据结构。

4.3. 数据的动态调整

TUIShortVideoView 提供了可供外部操作数据的数据管理类 TUIShortVideoDataManager，其作用主要是对当前播放器列表内的数据做基本增删改查等操作，参见如下演示代码：



```
///1、删除索引1处的数据和视图
[[self.videoView getDataManager] removeData:1];
///2、添加一组数据到索引9处
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.viewType = TUI_ITEM_VIEW_TYPE_CUSTOM;
[[self.videoView getDataManager]  addData:model index:9];
```

更详细的接口说明如下：

参数名称	含义

removeData	按索引移除数据
removeRangeData	按范围移除数据
removeDataByIndex	按索引数组移除数据
addData:index	按索引添加数据
addRangeData:startIndex	按模型数组从某个索引处添加数据
replaceData:index	按索引替换数据
replaceRangeData:startIndex	按模型数组从某个索引处替换数据
getDataByPageIndex	读取某个索引处的数据
getCurrentDataCount	获取当前播放列表内的数据总数
getCurrentIndex	获取当前播放界面的数据索引
getCurrentModel	获取当前播放界面的数据模型

说明：

DataManager 的接口调用完后 UI 界面自动刷新。

未操作当前播放界面的情况下，无感刷新。

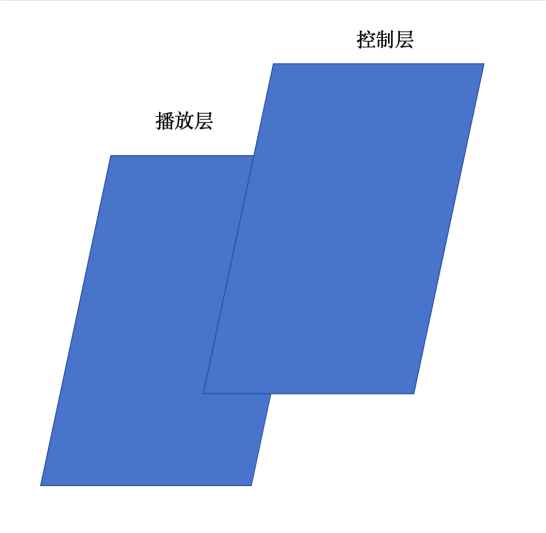
操作了当前界面会刷新当前的界面。

如果删除了当前播放界面，会自动播放下一个，如果下一个无数据（已经到达末尾），将会播放上一个。

5. 自定义 UI 图层

5.1. 层级结构

TUIPlayerShortVideo 的层级结构大致如下：



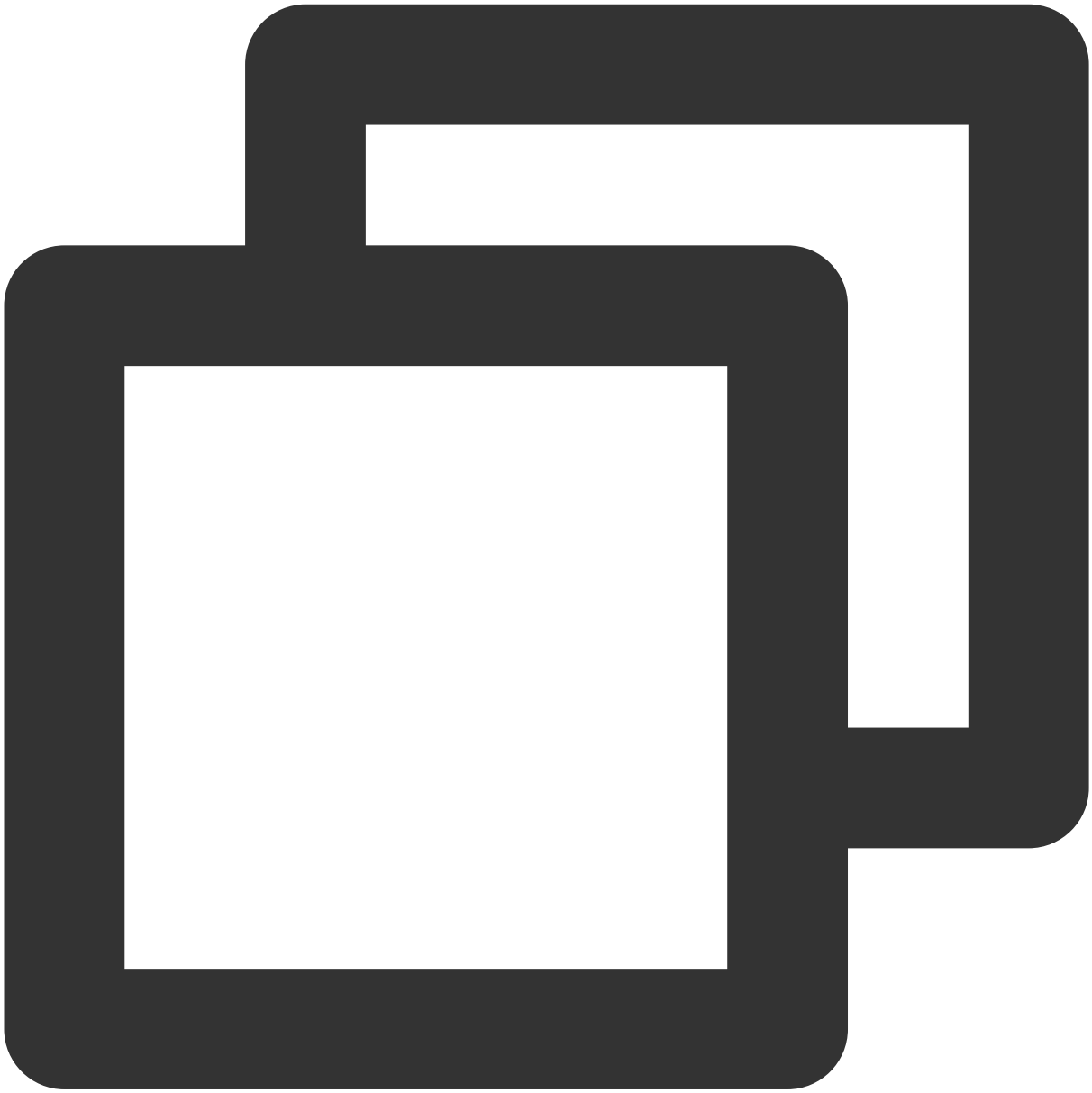
分为显示层和控制层，两者是以堆叠的方式结合。

显示层负责内容展示，点播，直播，广告推广页面等，这层是 SDK 内部管理。

控制层负责交互，点赞，评论，等，这层交给用户去实现高度的自定义。

5.2. TUIPlayerShortVideoUIManager

您可以通过 TUIPlayerShortVideoUIManager 的接口 和 面向协议控制层界面去实现您的自定义 UI。 参见如下代码：



```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc] ini
[uiManager setControlViewClass: TUIPSControlView.class];
[uiManager setLoadingView:[[TUIPSLoadingView alloc] init]];
[uiManager setBackgroundView:[UIView new]];
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

上面的代码通过 TUIPlayerShortVideoUIManager 自定义了名为 TUIPSControlView 的视频控制层（进度条，时间等），以及名为 TUIPSLoadingView 的 loading 加载控件。

TUIPlayerShortVideoUIManager 的接口参见下表：

参数名称	含义

setLoadingView	设置加载图
setBackgroundView	设置背景图
setErrorView	设置错误界面
setControlViewClass	设置视频控制层* @param ViewClass 控制层类，ViewClass 是您封装好的视频控制 View，包含如进度条，时间 lable 等控件 * 它将被整体覆盖在视频窗口上，大小与视频窗口一致
setControlViewClass :viewType	设置不同类型的视频控制层
getLoadingView	获取加载图实 View 实例
getBackgroundView	获取背景图 View 实例
getErrorView	获取错误界面 View 实例
getControlViewClass	获取视频控制界面 View 类
getControlViewClassWithViewType	获取不同类型视频控制界面类

播放层目前支持的类型有两种：

TUI_ITEM_VIEW_TYPE_VOD ///视频

TUI_ITEM_VIEW_TYPE_LIVE ///直播

TUI_ITEM_VIEW_TYPE_CUSTOM /// 自定义类型（例如广告页面）

相应的控制层的协议也支持两种：

TUIPlayerShortVideoControl ///视频控制层协议

TUIPlayerShortVideoLiveControl ///直播控制层协议

TUIPlayerShortVideoCustomControl ///自定义控制层协议

说明

所有非点播和直播的界面，均以 TUI_ITEM_VIEW_TYPE_VOD 和 TUIPlayerShortVideoCustomControl 去呈现，Custom 是一个大类，具体的细分需要用户在此白板上自行定义，如上文 TUIPlayerDataModel 模型构建提到的，可以通过 extInfo/type 或者 extInfo 下别的字段去做更细粒度的划分。

TUIPlayerShortVideoControl 协议具体接口说明如下：

参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互
model	当前播放的视频模型
currentPlayerStatus	当前播放器的播放状态

showCenterView	显示中心 view
hideCenterView	隐藏中心 view
showLoadingView	显示 loading 图
hiddenLoadingView	隐藏 loading 图
setDurationTime	总的视频时常
setCurrentTime	当前的播放时长
setProgress	进度条进度
showSlider	显示进度条
hideSlider	隐藏进度条
reloadControlData	触发视图刷新
getPlayer	获取播放器对象
onPlayEvent	获取播放器事件
getVideoLayerRect	获取视频渲染区域的变化
getVideoWidget	获取视频渲染图层对象

TUIPlayerShortVideoLiveControl 协议具体接口说明如下：

参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互
model	当前播放数据模型
reloadControlData	触发视图刷新
getPlayer	获取播放器对象
getVideoLayerRect	获取视频图层区域
getVideoWidget	获取视频渲染图层

TUIPlayerShortVideoCustomControl 协议具体接口说明如下：

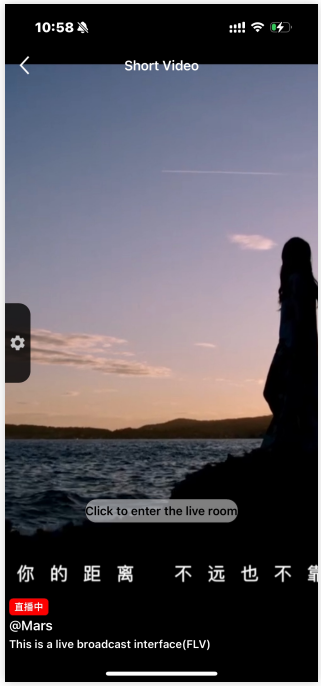
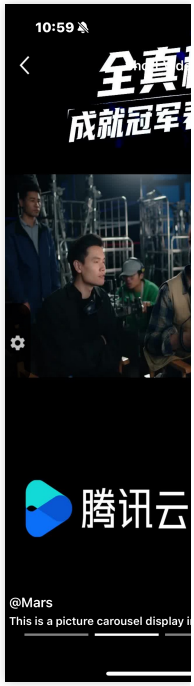
参数说明	含义
delegate	一个反向代理，用于控制层与播放层的交互

model	当前播放数据模型
reloadControlData	触发视图刷新

注意：

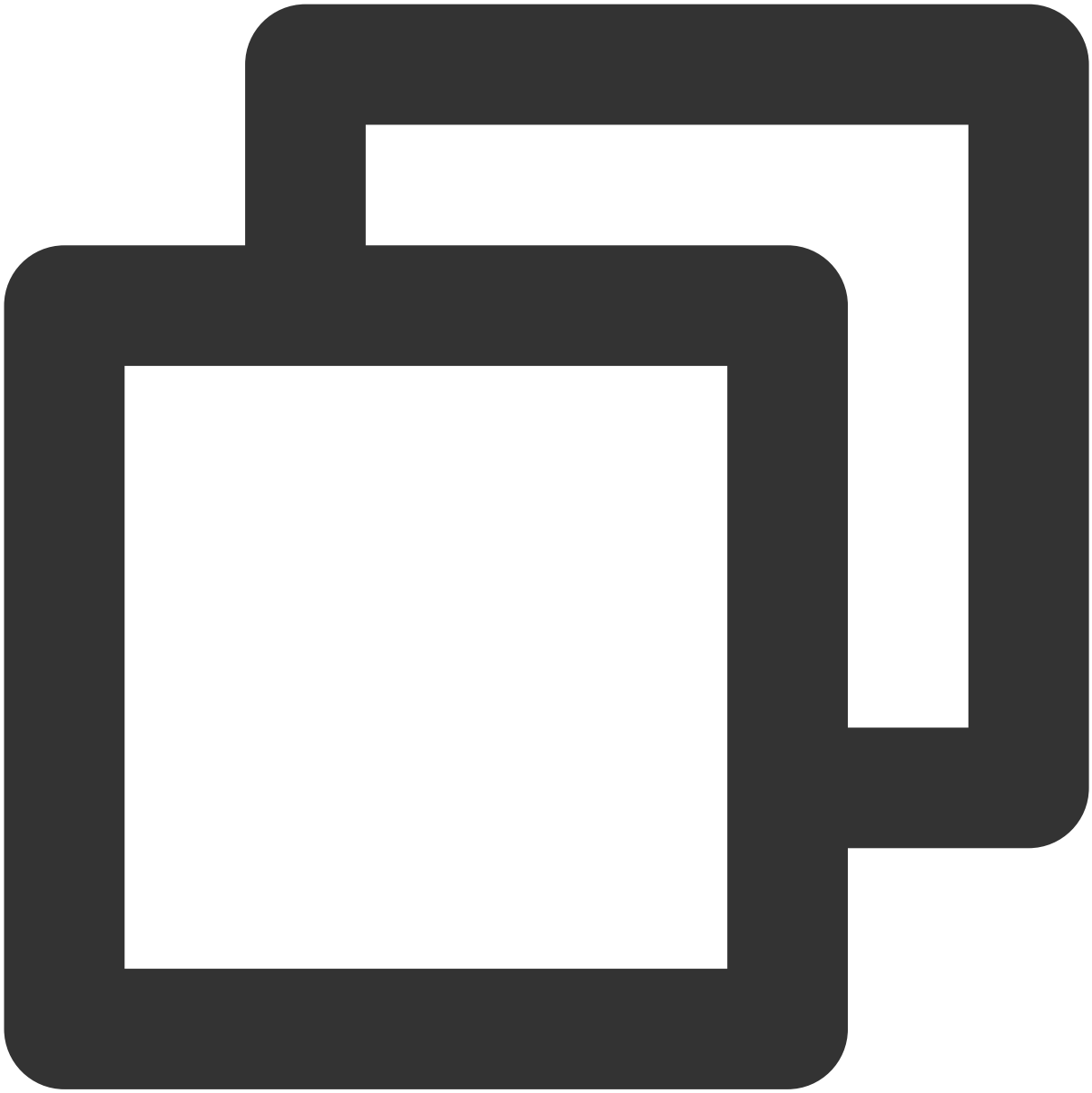
传入的自定义控制层 View 需要遵守相关的协议，否则将会编译失败。

5.3. 示例

类型	点播	直播	Custom（轮播图）
样例			

5.3.1. 定义不同的样式下的 UI 样式

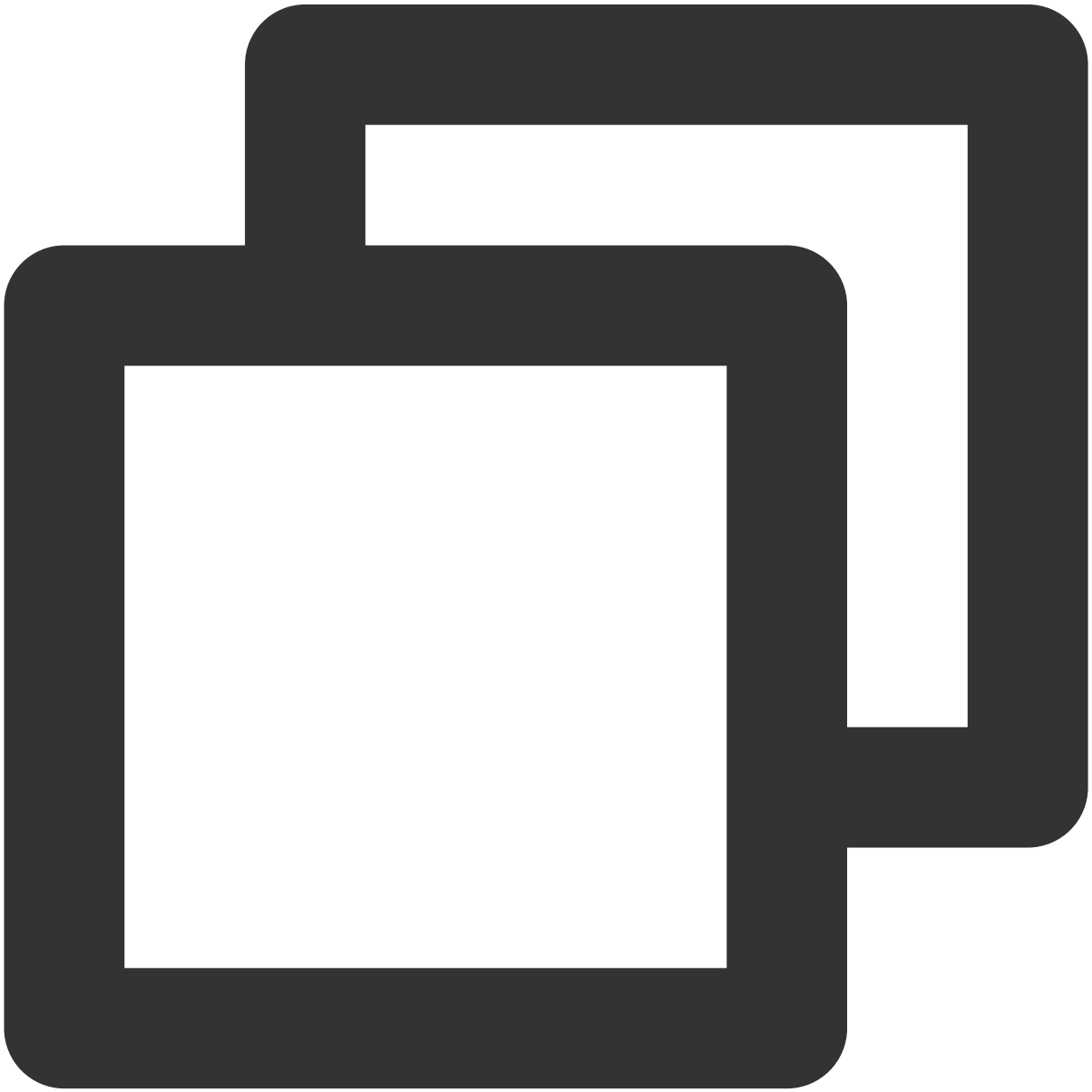
点播样式（TUIPlayerShortVideoControl）



```
@interface TUIPSControlView : UIView<TUIPlayerShortVideoControl>
@property (nonatomic, strong) TUIPlayerVideoModel *videoModel
@end
@implementation TUIPSControlView

-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
```

```
}

-(void)setModel:(TUIPlayerVideoModel *)model {
    _model = model;
    /// 数据
}
@end
```

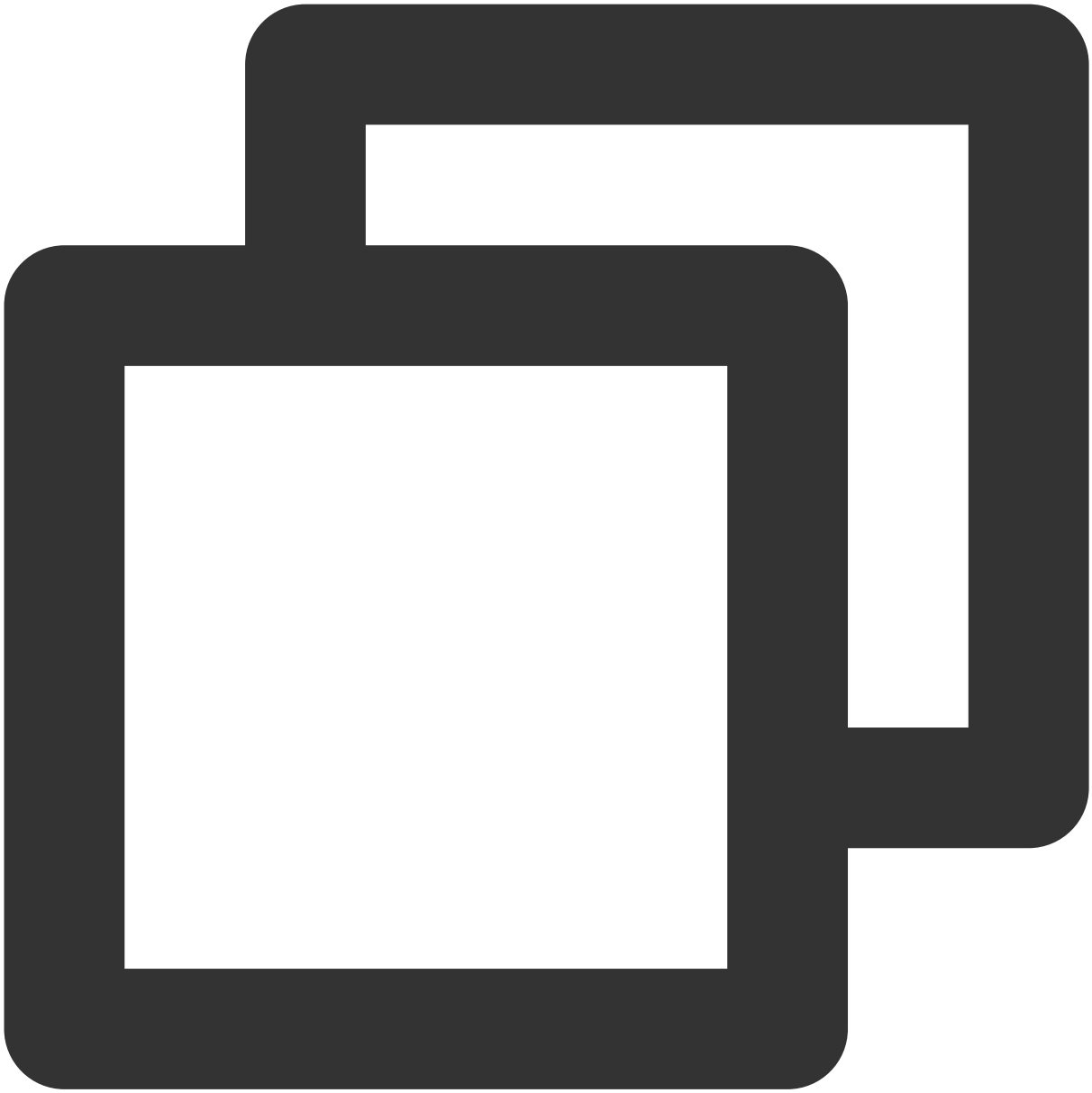
直播样式(TUIPlayerShortVideoLiveControl)



```
@interface TUIPSControlLiveView : UIView<TUIPlayerShortVideoLiveControl>
```

```
@property (nonatomic, strong) TUIPlayerLiveModel *videoModel
@end
@implementation TUIPSControlLiveView
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
}

-(void)setModel:(TUIPlayerLiveModel *)model {
    _model = model;
    /// 数据
}
@end
```

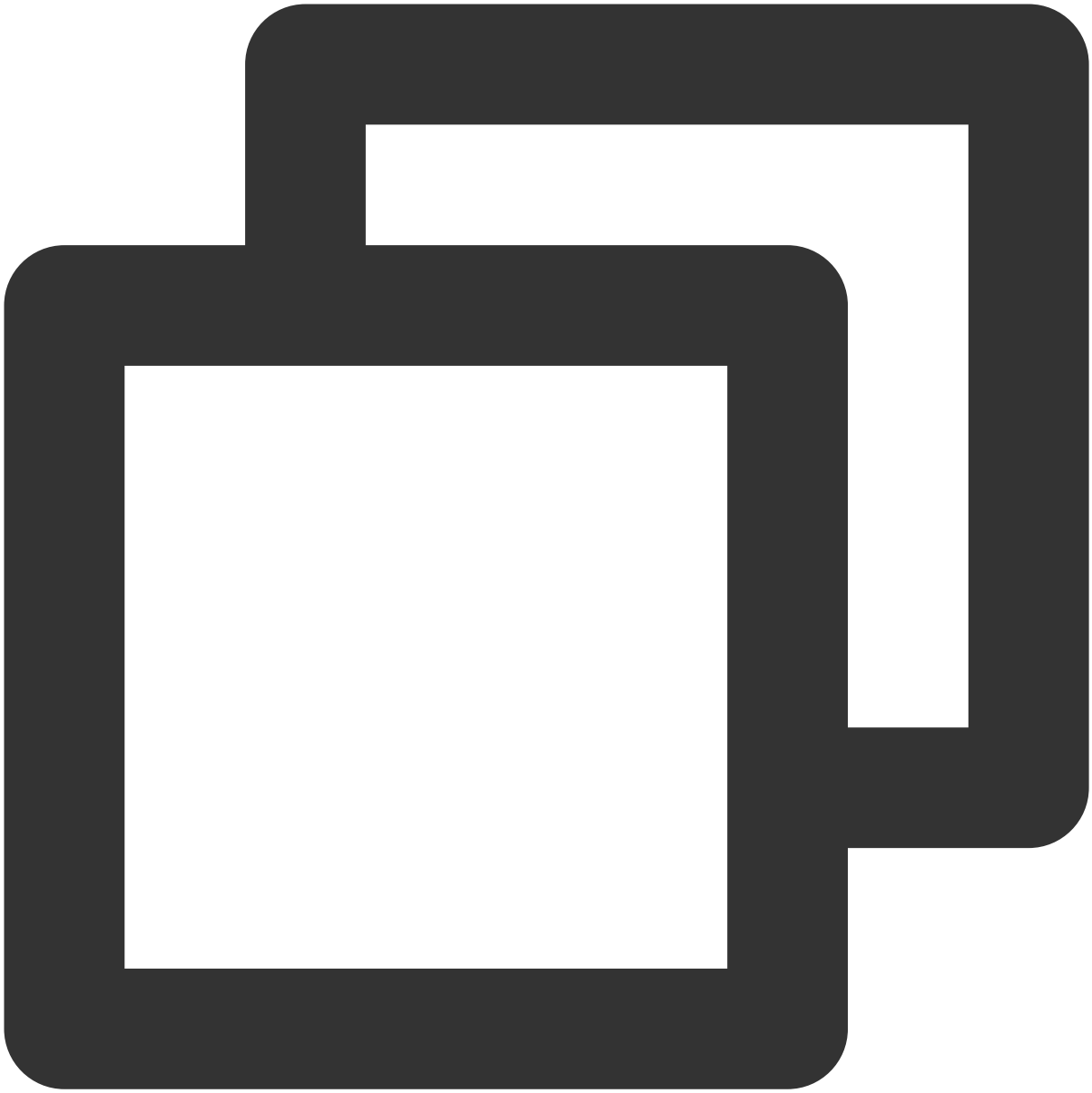
轮播图&图文广告等其他样式(TUIPlayerShortVideoCustomControl)



```
@interface TUIPSControlCustomView : UIView<TUIPlayerShortVideoCustomControl>
@property (nonatomic, strong) TUIPlayerDataModel *videoModel
@end
@implementation TUIPSControlCustomView
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI布局代码
    }
    return self;
}
```

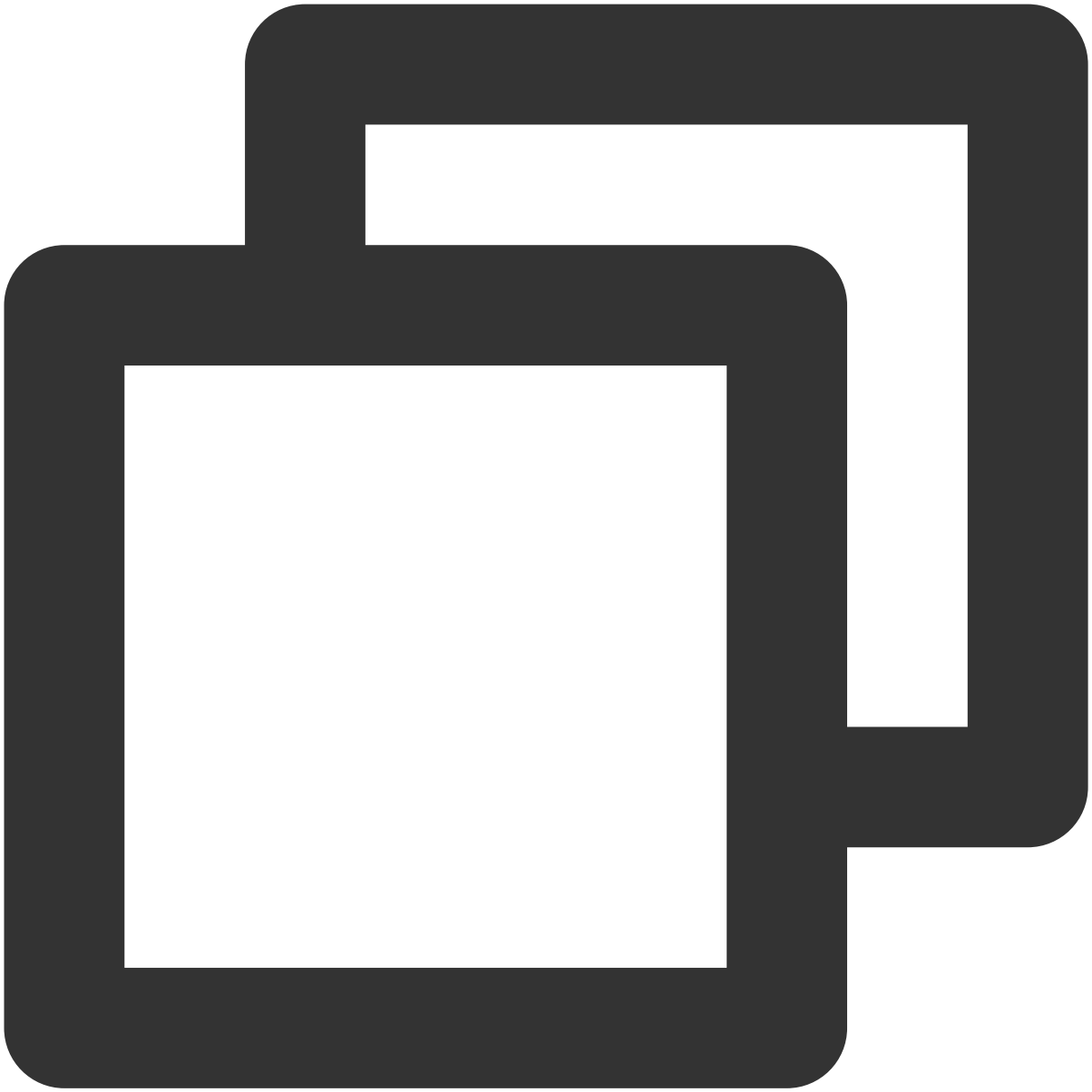
```
-(void)setModel:(TUIPlayerDataModel *)model {
    _model = model;
    /// 数据

    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];

    if ([type isEqualToString:@"ad"]) {  ///图文广告
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString
    } else if ([type isEqualToString:@"imageCycle"]) { ///轮播图
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }

}
@end
```

5.3.2. 将创建好的样式通过 TUIPlayerShortVideoUIManager 注册



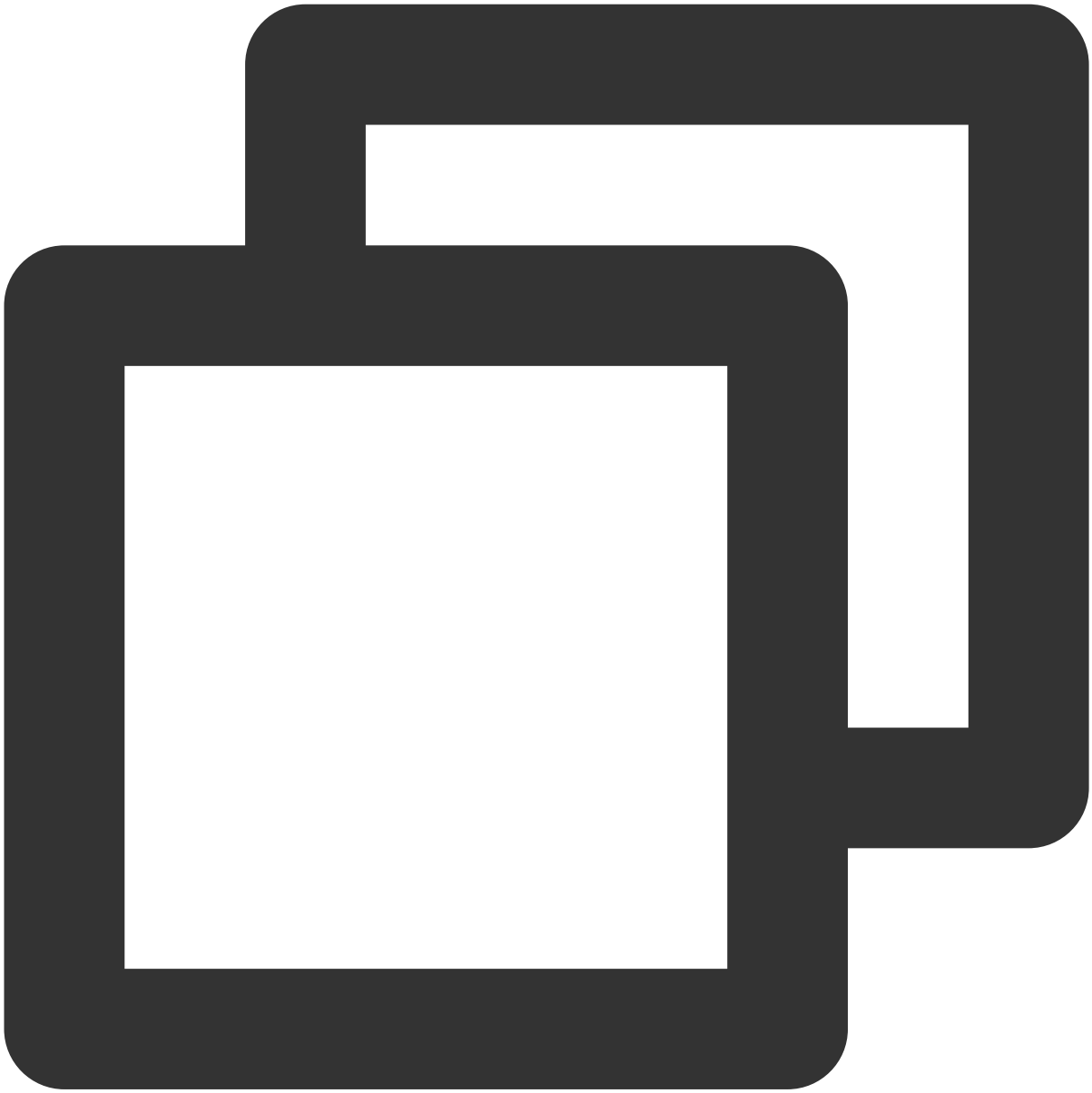
```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc] ini
[uiManager setControlViewClass: TUIPSControlView.class viewType:TUI_ITEM_VIEW_TYPE_
[uiManager setControlViewClass: TUIPSControlLiveView.class viewType:TUI_ITEM_VIEW_T
[uiManager setControlViewClass: TUIPSControlCustomView.class viewType:TUI_ITEM_VIEW
```

5.3.2. 通过 TUIPlayerShortVideoUIManager 初始化 TUIShortVideoView



```
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

5.3.3. 通过 setShortVideoStrategyModel 和 setShortVideoLiveStrategyModel 设置点播和直播的相关策略



```
// Set your playback strategy
 TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
 model.mPreloadConcurrentCount = 1;
 model.preDownloadSize = 1;
 model.enableAutoBitrate = NO;
[_videoView setShortVideoStrategyModel:model];

 // live strategy
 TUIPlayerLiveStrateyModel *liveStrateyModel = [[TUIPlayerLiveStrateyModel alloc] i
 [_videoView setShortVideoLiveStrategyModel:liveStrateyModel];
```

5.3.4. UI 与数据的关系

TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 可以理解为预置模板，初始化 TUIShortVideoView 的时候已经预置好了各种类型的模板，当滑倒相对应的数据类型的时候，即会显示当前模板。

类型	数据模型	UI 模板
点播	TUIPlayerVideoModel	TUIPSControlView
直播	TUIPlayerLiveModel	TUIPSControlLiveView
自定义类型（轮播图，图文广告等）	TUIPlayerDataModel	TUIPSControlCustomView

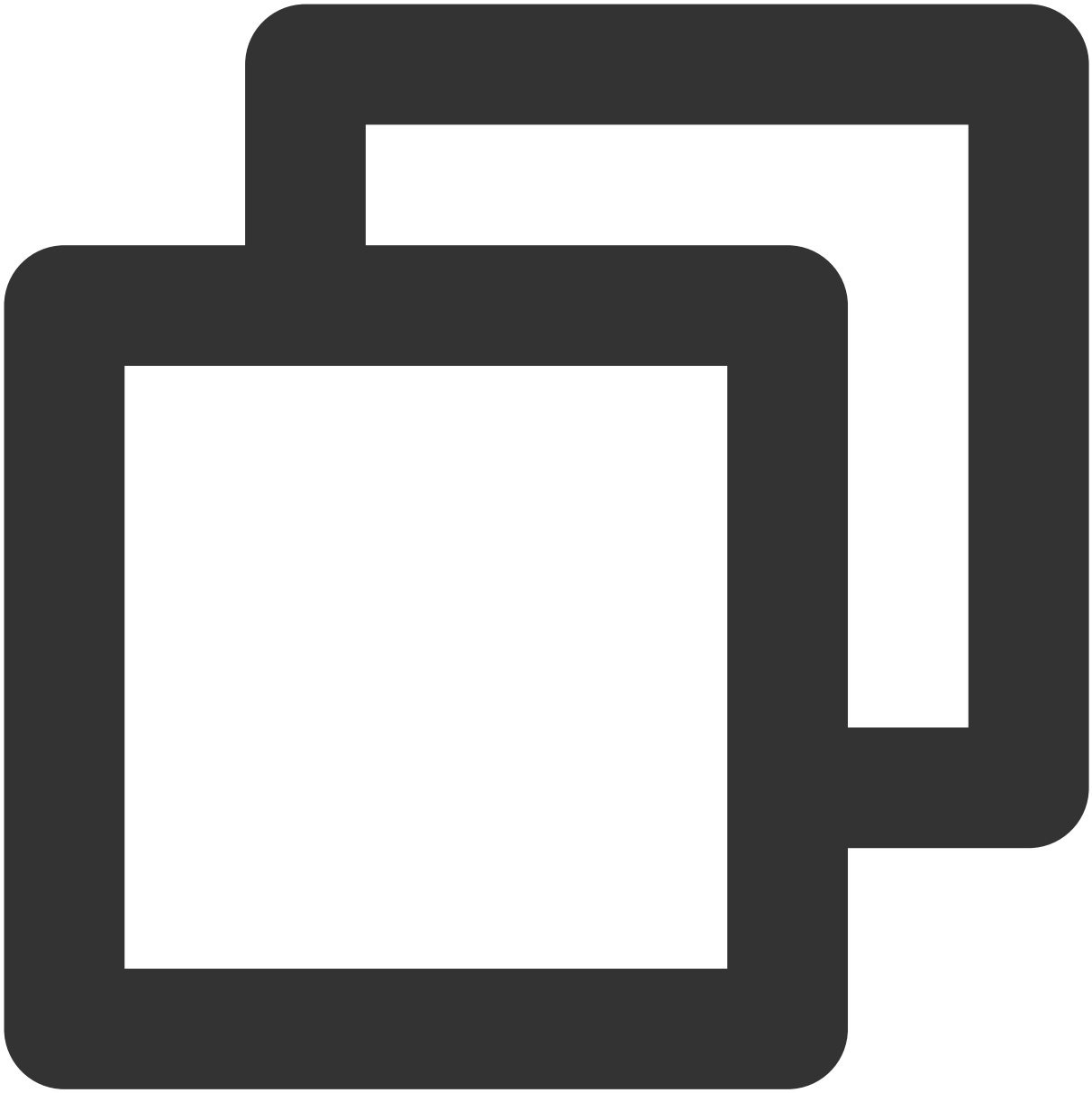
UI 模板是提前预置的

UI 样式的显示则根据相对应的数据类型的驱动

5.3.4. UI 模板与 TUIShortVideoView 的交互

自定义 UI 模板通过对应的协议与 TUIShortVideoView 实现交互 TUIShortVideoView 向 TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 传递消息通过 TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl 的协议方法。

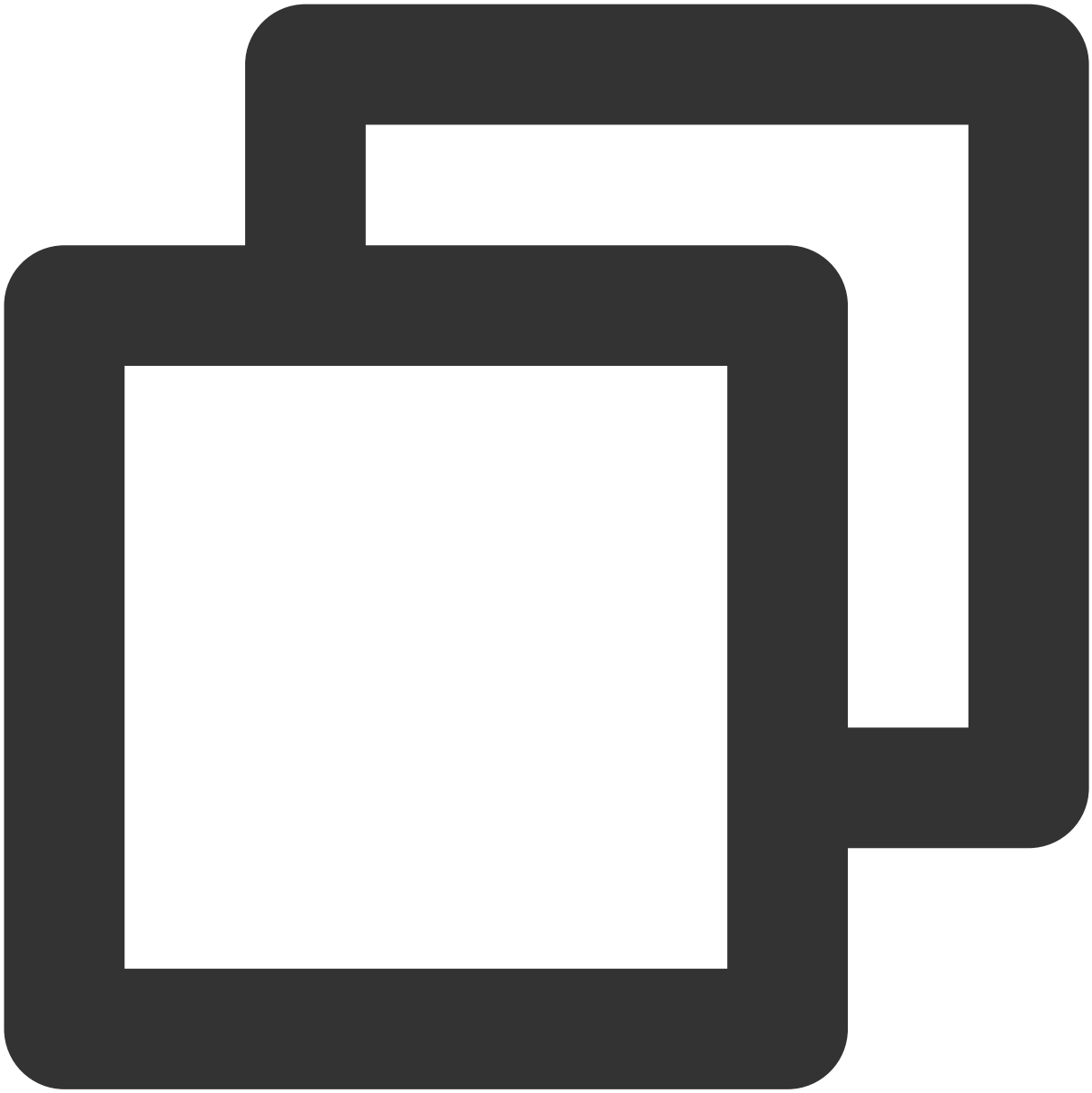
如点播：



```
-(void)setModel:(TUIPlayerVideoModel *)model {

    if ([_model observationInfo]) {

        [_model removeObserver:self forKeyPath:@"preloadState"];

    }
    _model = model;
    [model addObserver:self forKeyPath:@"preloadState" options:NSKeyValueObservingO
```

```
    NSDictionary *dic = model.extInfo;
    NSString *iconUrl = [dic objectForKey:@"iconUrl"];
    NSString *advertise = [dic objectForKey:@"advertise"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *title = [dic objectForKey:@"title"];
    NSString *topic = [dic objectForKey:@"topic"];
    self.iconImageView.image = [UIImage imageNamed:iconUrl];
    [self.adButton setTitle:advertise forState:UIControlStateNormal];
    self.nameLabel.text= name;
    self.themeLabel.text = topic;
    self.desLabel.text = title;
    [self updatePreloadState];
    [self updateLickCount];
    model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
        [self updateLickCount];
    };
}
```

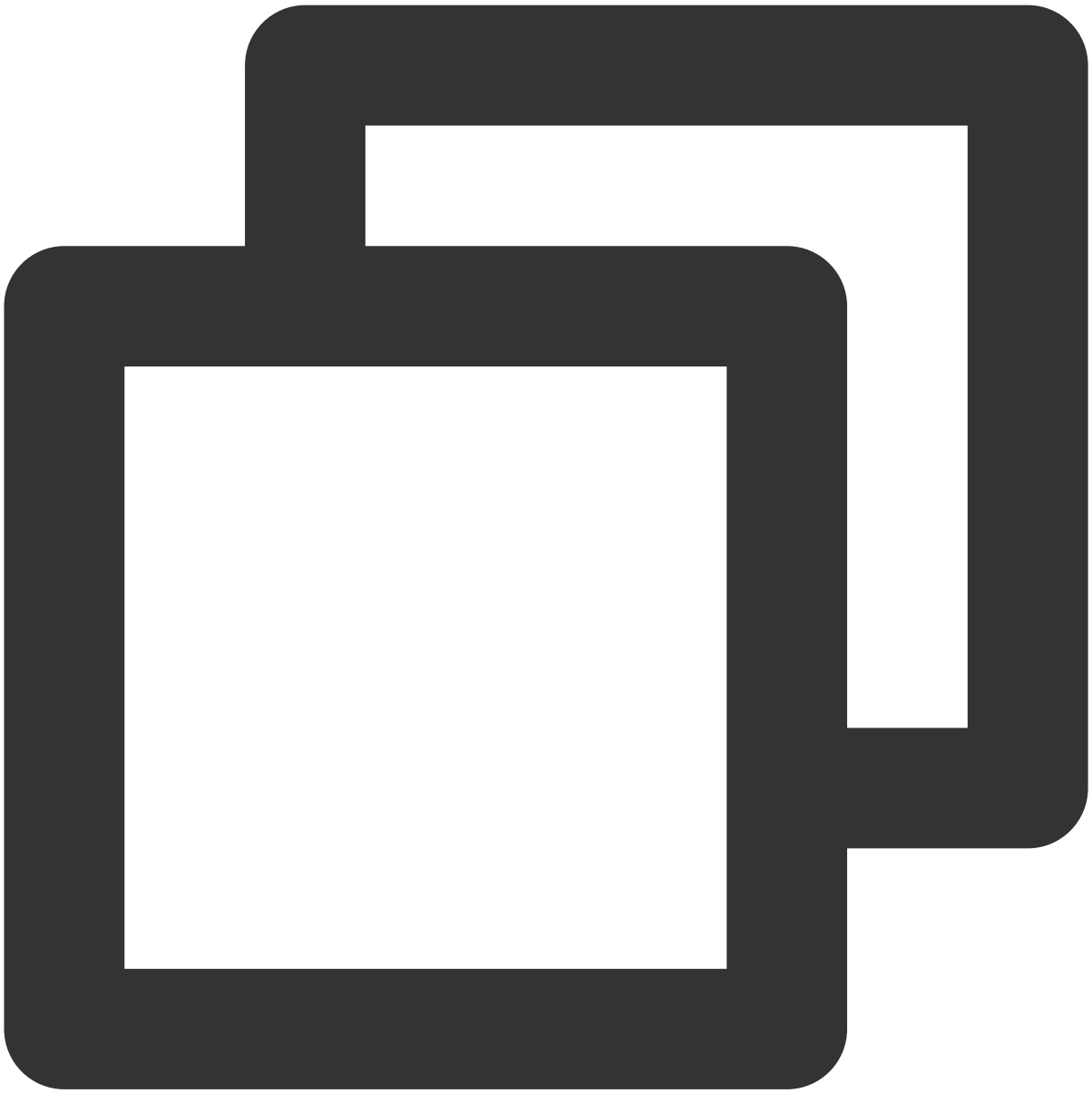
直播：



```
-(void)setModel:(TUIPlayerLiveModel *)model {

    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"liveTitile"];
    NSString *adDes = [dic objectForKey:@"liveDes"];
    NSString *name = [dic objectForKey:@"name"];

    self.nameLabel.text = name;
    self.desLabel.text = adTitile;
```

```
}
```

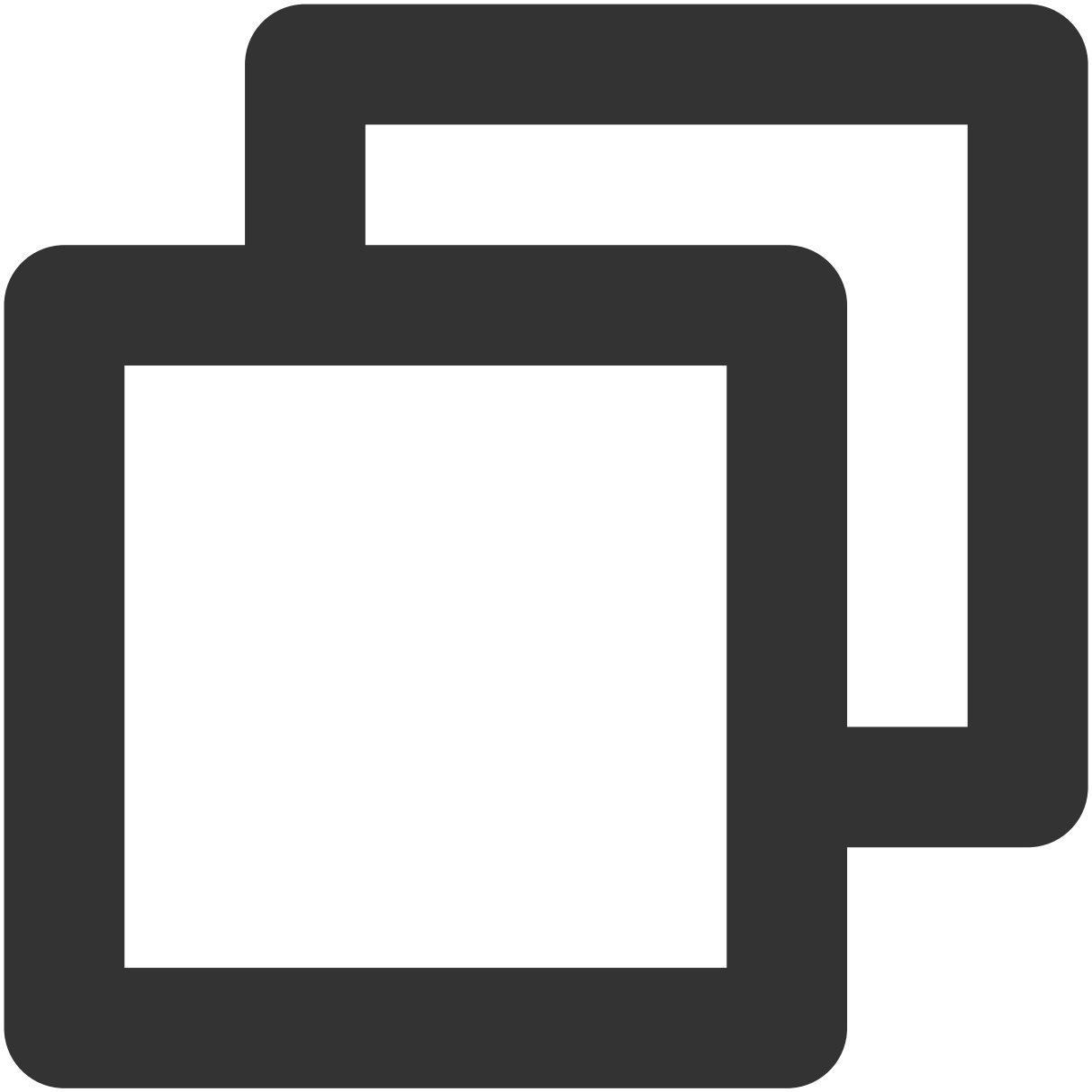
Custom（轮播图，图文等）



```
-(void)setModel:(TUIPlayerDataModel *)model {

    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
```

```
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];
    self.desLabel.text = adTitile;
    self.nameLabel.text = name;
    if ([type isEqualToString:@"web"]) {
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString
    } else if ([type isEqualToString:@"imageCycle"]) {
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }
}
```

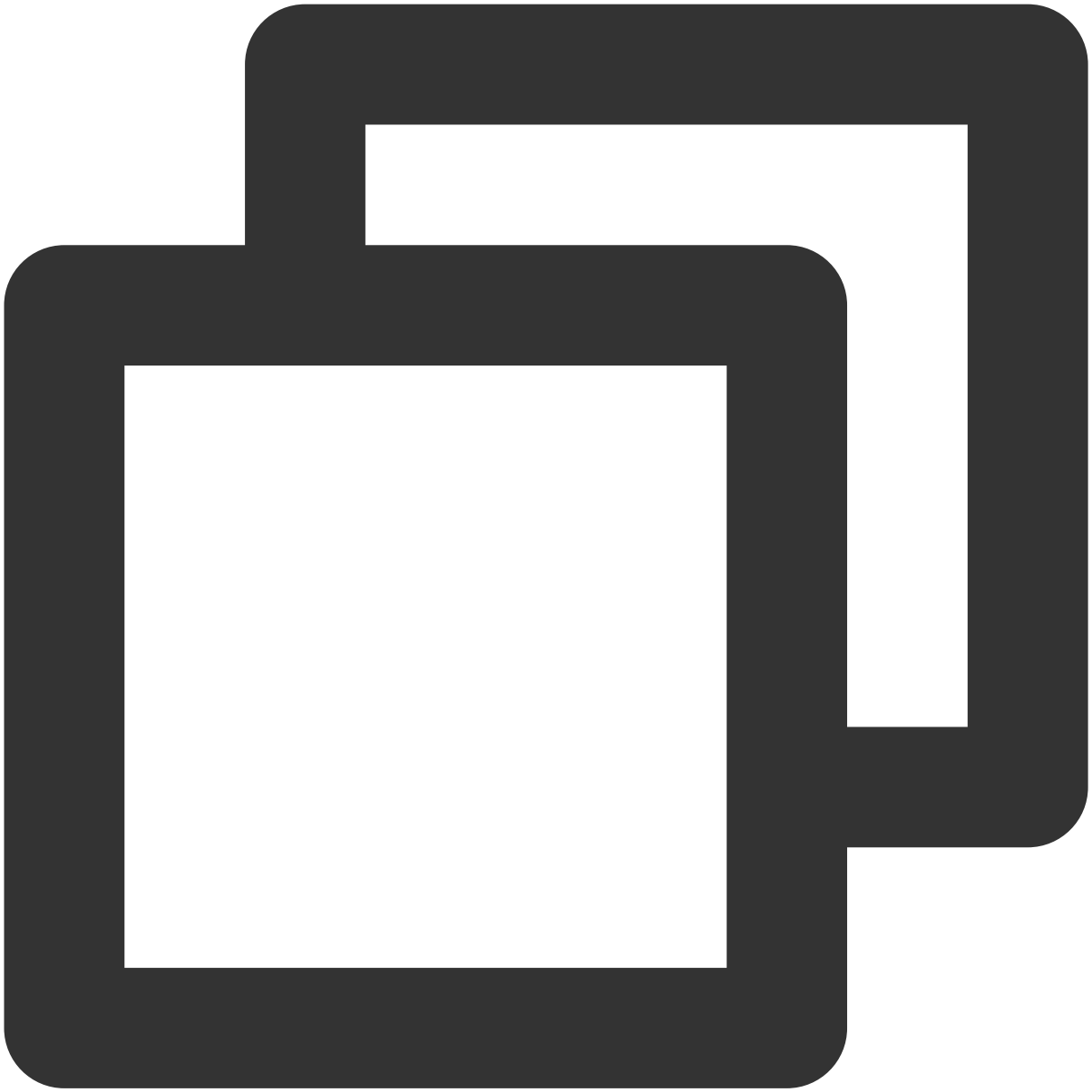
TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView 向 TUIShortVideoView 传递消息则通过 TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl 的 delegate。如点播：



```
@protocol TUIPlayerShortVideoControlDelegate <NSObject>

/**
 * 暂停
 */
- (void)pause;
/**
 * 继续播放
 */
- (void)resume;
```

```
/**
 * 滑动滚动条的处理
 * @param time    滑动的距离
 */
- (void)seekToTime:(float)time;

/**
 * 是否正在播放
 */
- (BOOL)isPlaying;

/**
 * 重置视频播放容器
 * - 用于视频播放容器被移除后需要重置的场景
 */
- (void)resetVideoWeigetContainer;

@optional
/**
 * 自定义回调事件
 */
- (void)customCallbackEvent:(id)info;
@end


/////调用
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

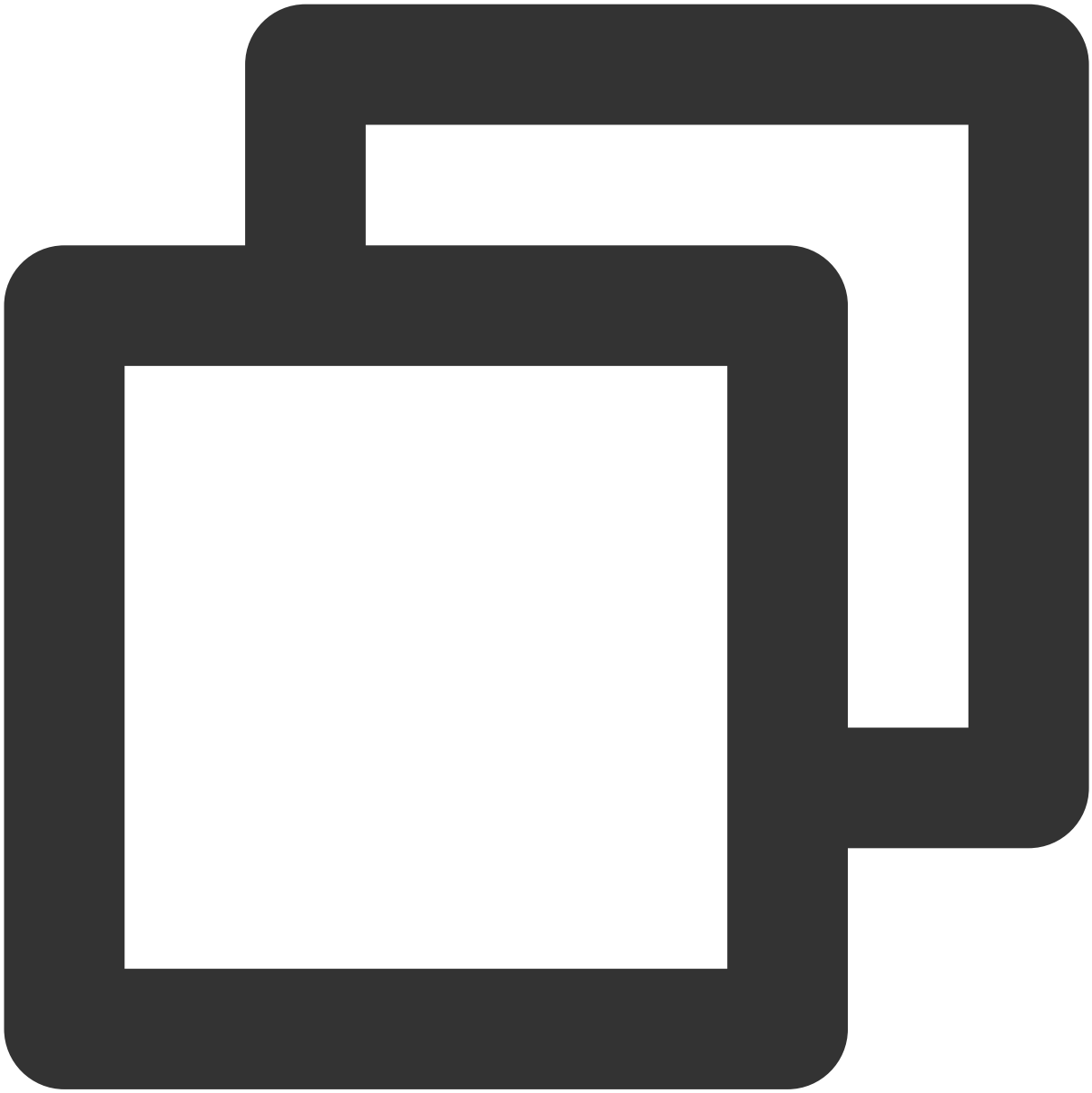
直播



```
@protocol TUIPlayerShortVideoLiveControlDelegate <NSObject>

/**
 * 暂停
 */
- (void)pause;

/**
 * 继续播放
 */
```

```
- (void)resume;

/**
 * 重置视频播放容器
 * - 用于视频播放容器被移除后需要重置的场景
 */
- (void)resetVideoWeigetContainer;

@optional
/**
 * 自定义回调事件
 */
- (void)customCallbackEvent:(id)info;
@end

/////调用
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

Custom（轮播图，图文等）



```
@protocol TUIPlayerShortVideoCustomControlDelegate <NSObject>

@optional
/**
 * 自定义回调事件
 */
- (void)customCallbackEvent:(id)info;
@end

/////调用
```

```
if (self.delegate && [self.delegate respondsToSelector:@selector(customCallbackEven
    [self.delegate customCallbackEvent:@"test"];
}
```

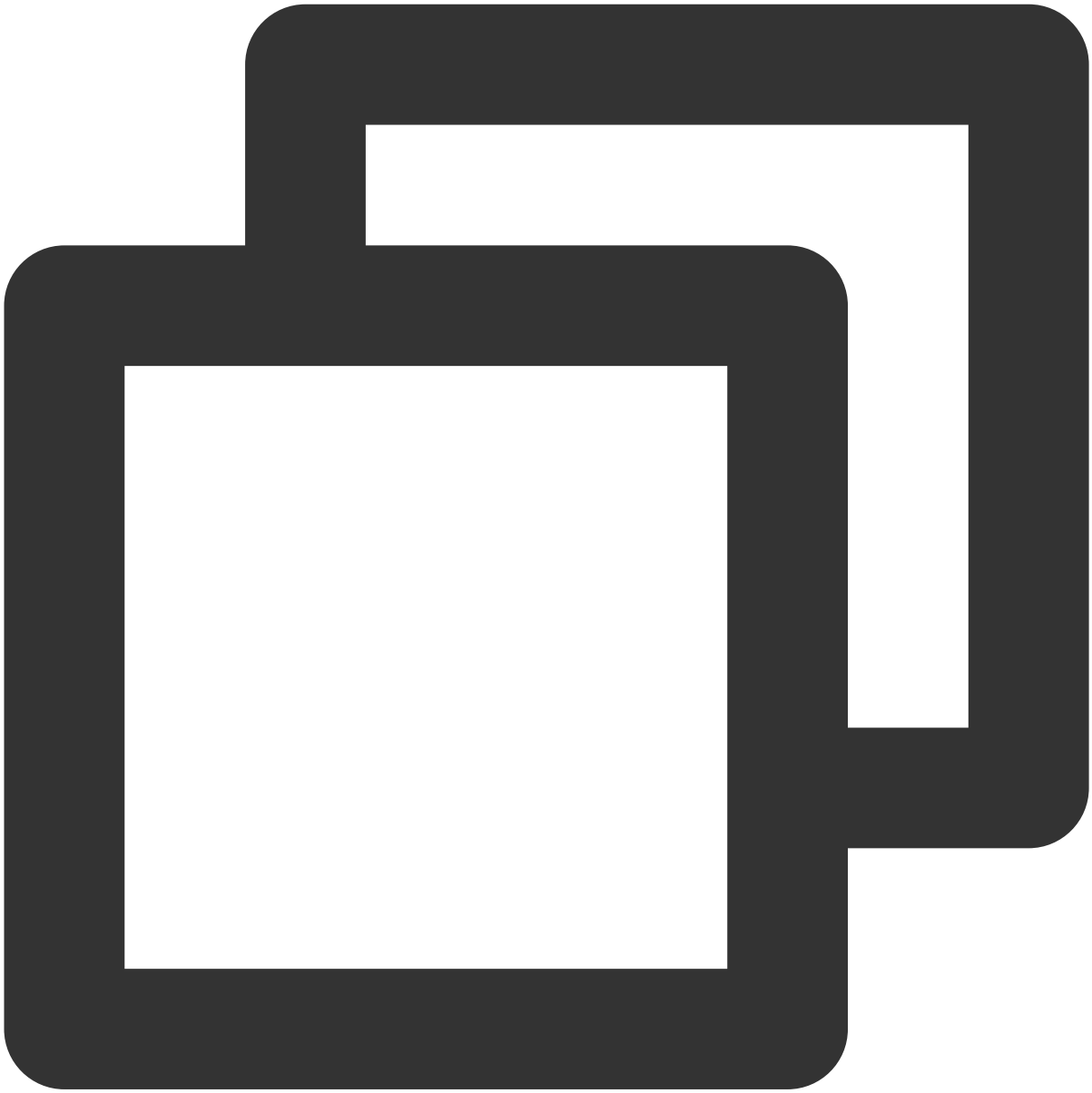
说明：

完整示例请看 Demo。

高级功能

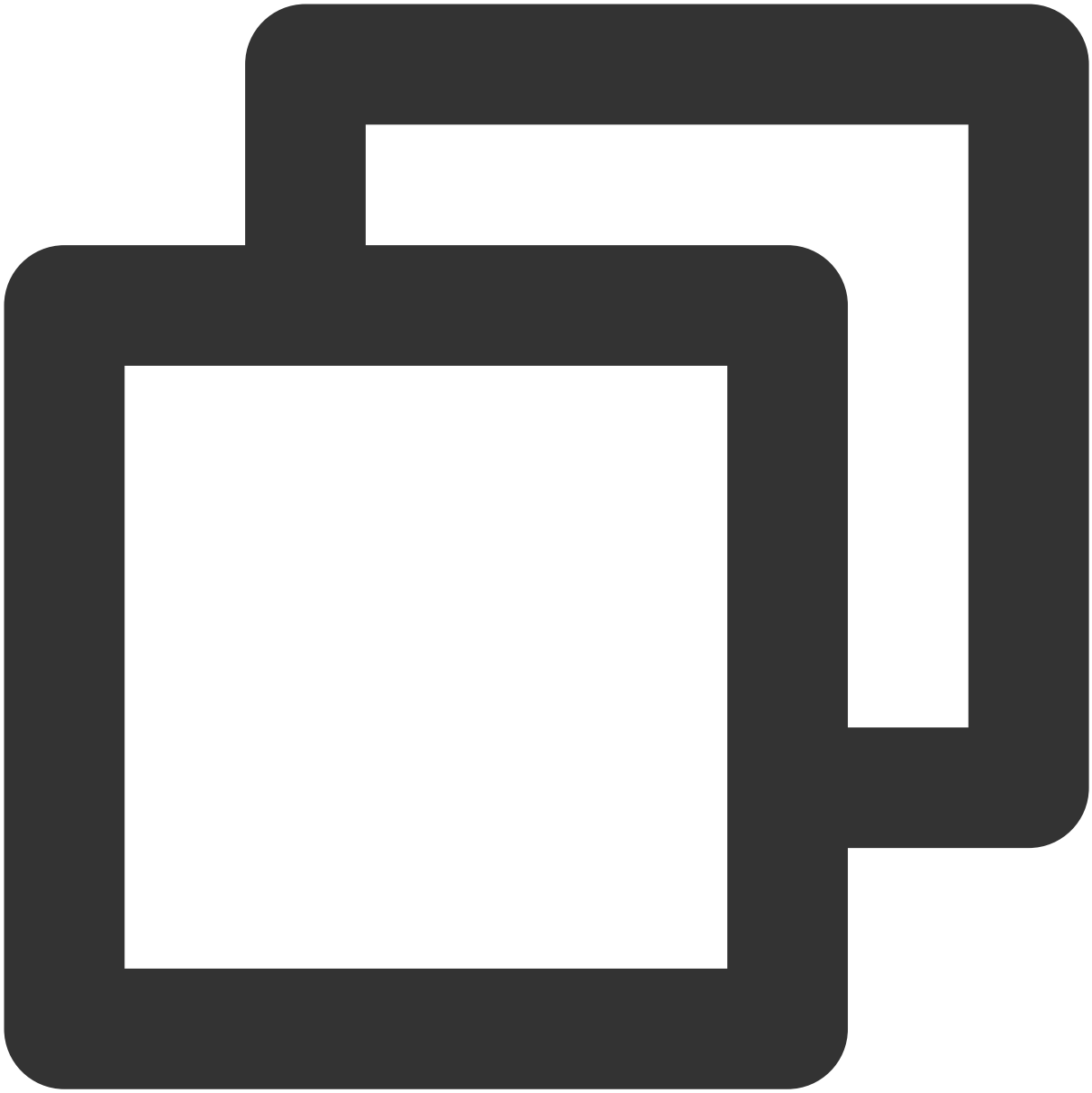
业务通知消息到页面图层

TUI 提供了消息接口供用户把数据实时通知到当前图层，可以通过数据操作对象获取到视频对象之后，进行通知，示例如下：



```
/// 获取数据管理器
TUIShortVideoDataManager *dataManager = [self.videoView getDataManager];
///获取数据模型
TUIPlayerDataModel *model = [dataManager getDataByPageIndex:1];
///修改数据模型
model.extInfo = @{@"key":@"value"}
///通知数据模型发生改变
[model extInfoChangeNotify];
```

随后在 UI 控制层的 onExtInfoChanged 回调中会收到该通知，从而对当前页面进行 UI 修改，示例如下：



```
model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
      [self updateLickCount];
};
```

注意：

此功能只对 extInfo 字段有效

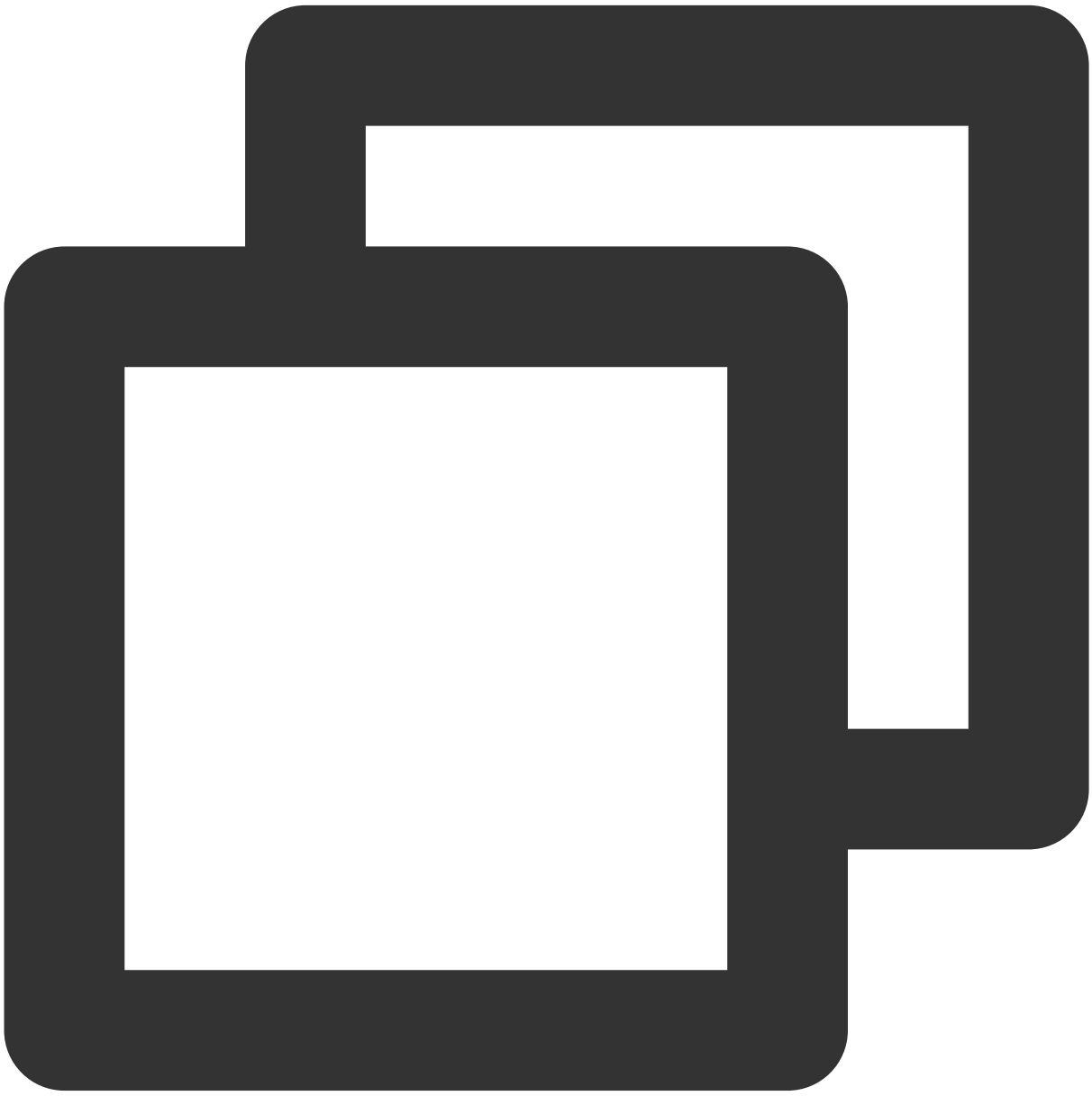
点播设置音量均衡

播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。

通过设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。

注意：

播放器高级版 11.7 版本开始支持。



```
/// 音量均衡 .响度范围：-70～0(LUFS)。此配置需要LiteAVSDK 11.7 及以上版本支持。
/// 以下几种常量供参考使用
/// 关：AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h)
```

```
/// 开（标准响度）：AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h)
/// 开（低响度）：AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h)
/// 开（高响度）：AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h)
/// 默认值为AUDIO_NORMALIZATION_OFF。

TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
model.audioNormalization = AUDIO_NORMALIZATION_STANDARD;

[_videoView setShortVideoStrategyModel:model];
```

Android

最近更新时间：2024-06-17 17:50:34

组件简介

TUIPlayerShortVideo 组件是腾讯云推出的一款性能优异，支持视频极速首帧和流畅滑动，提供优质播放体验的短视频组件。
首帧秒开：首帧时间是短视频类应用核心指标之一，直接影响用户的观看体验。短视频组件通过预播放、预下载、播放器复用和精准流量控制等技术，实现极速首帧、滑动丝滑的优质播放体验，从而提升用户播放量和停留时长。
优秀的性能：通过播放器复用和加载策略的优化，在保证极佳流畅度的同时，始终让内存和 CPU 消耗保持在较低的水平。
快速集成：组件对复杂的播放操作进行了封装，提供默认的播放 UI，同时支持 FileId 和 Url 播放，可低成本快速集成到您的项目中。

效果对比

以下视频演示了，在同等环境下，未经过优化和经过优化之后的短视频使用的对比差异。
优化前，可以明显感觉到视频起播的卡顿感。
优化后，可以达到无感起播的体验，优化后起播平均时长达到10毫秒 - 30毫秒。

未优化短视频	优化后短视频

TUIPlayerKit 下载

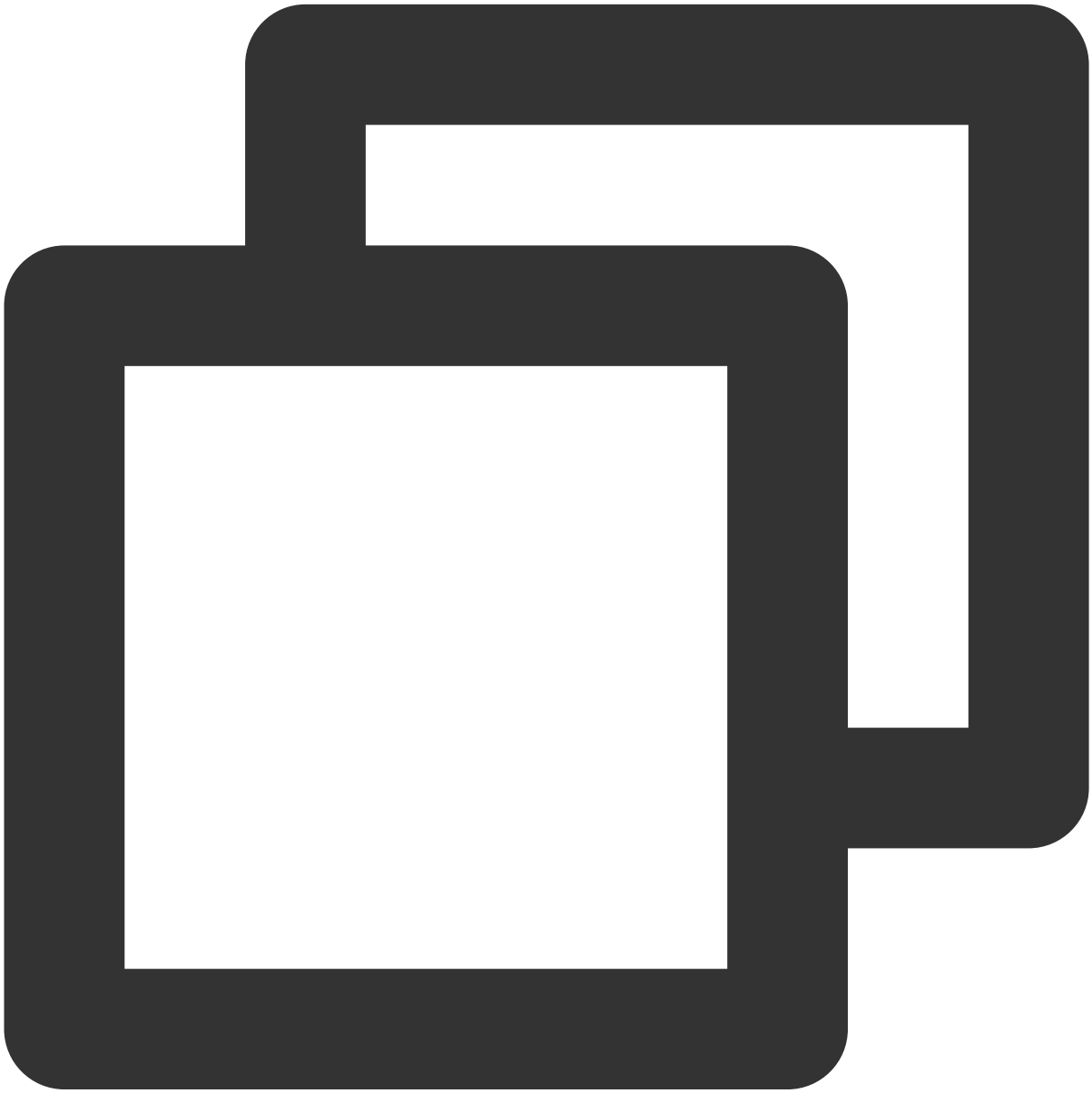
TUIPlayerKit SDK 和 Demo 可单击这里 下载。

集成 TUIPlayerShortVideo 组件

环境准备

Android 系统最低版本要求：Android SDK >= 19

添加短视频需要的依赖：

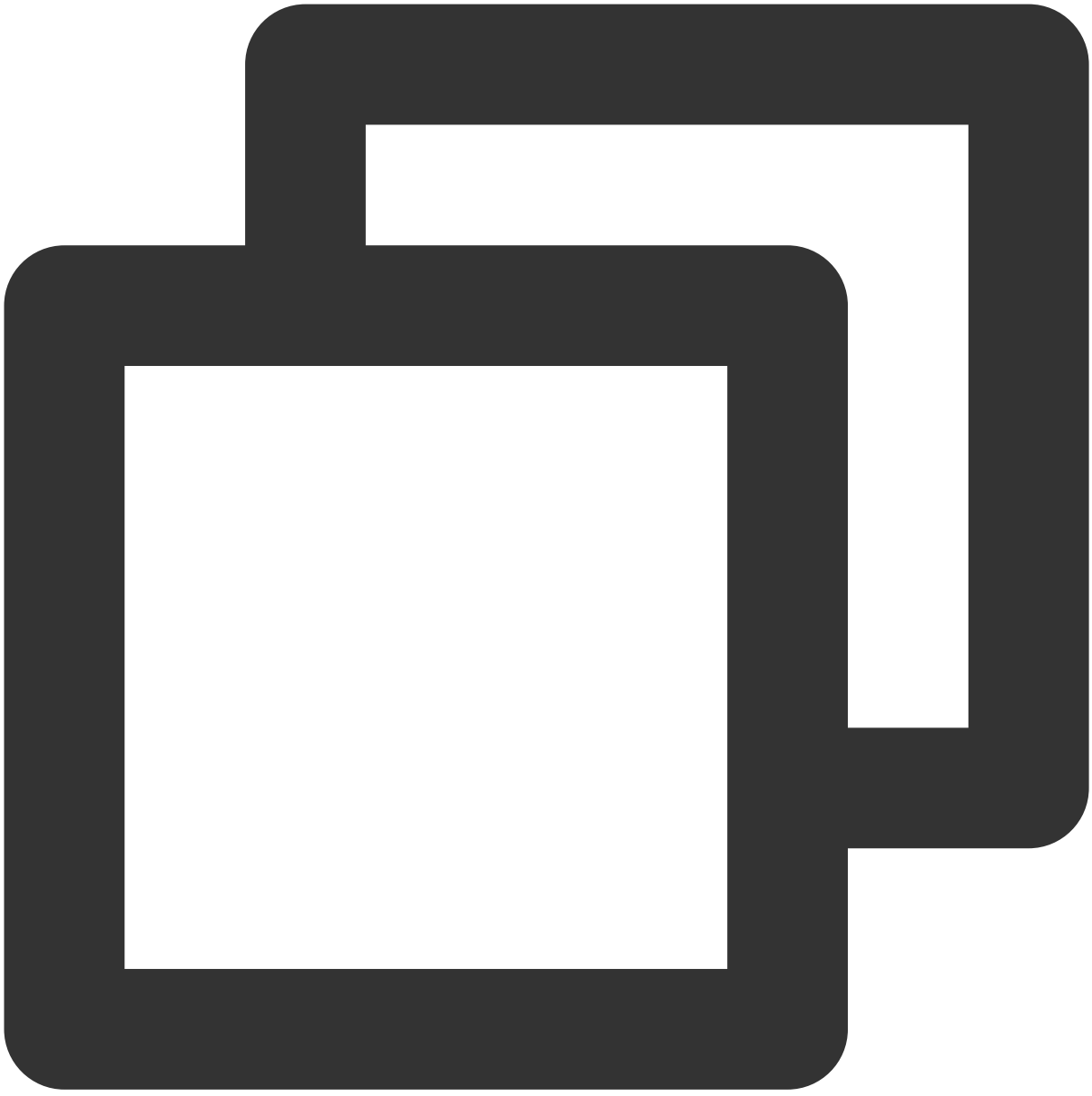


```
// 如果您使用的是专业版本的SDK，则用：api 'com.tencent.liteav:LiteAVSDK_Professional:late
api 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
implementation (name:'tuiplayercore-release_x.x.x', ext:'aar')
implementation (name:'tuiplayershortvideo-release_x.x.x', ext:'aar')
implementation 'androidx.appcompat:appcompat:1.0.0'
implementation 'androidx.viewpager2:viewpager2:1.0.0'
```

注意：

其中 `tuiplayercore-release` 和 `tuiplayershortvideo-release` 中的 `x.x.x` 为版本号，注意2个 aar 的版本号必须一致。

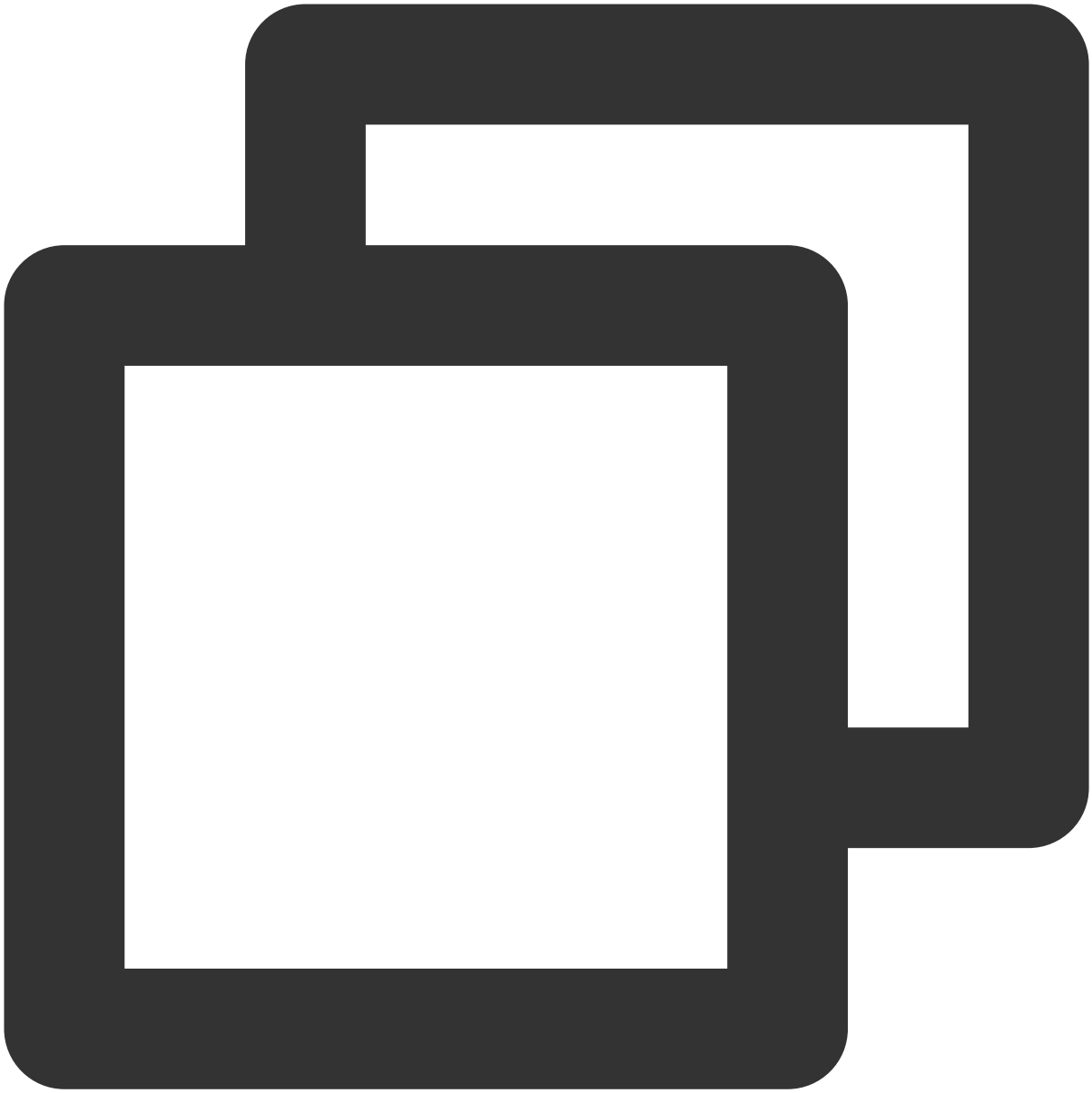
SDK 需要的权限：



```
<uses-permission android:name="android.permission.INTERNET" />
```

设置混淆规则：

在 proguard-rules.pro 文件中，将相关类加入不混淆名单：



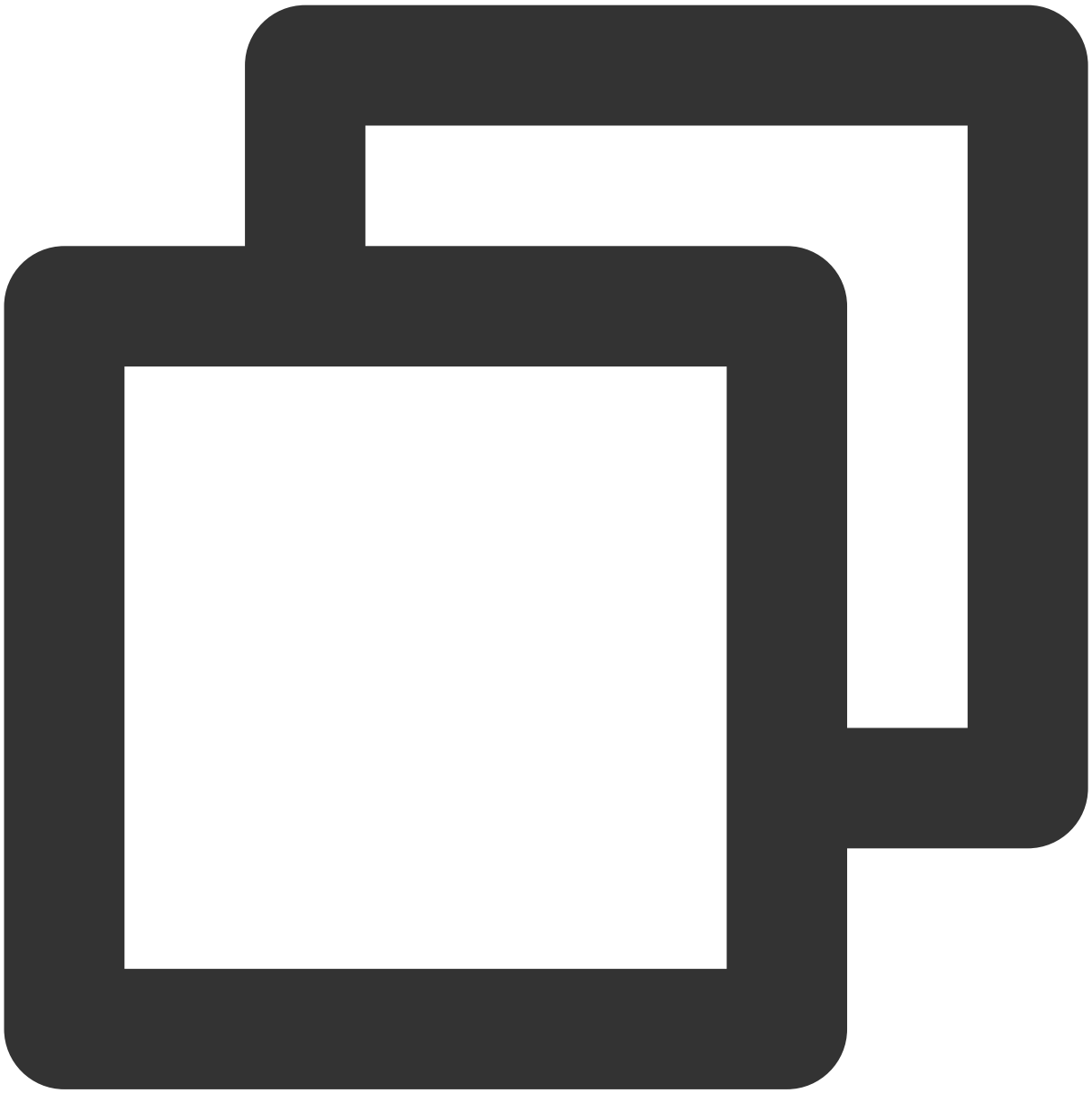
```
-keep class com.tencent.** { *; }
```

申请播放器高级版 License

使用 TUIPlayer Kit 组件需要使用移动端播放器高级版 License，您可参见 移动端播放器 License 指引来获取。若您已获取对应 License，可前往 云点播控制台 > License 管理 > 移动端 License 获取对应 LicenseURL 和 LicenseKey。如果没有申请移动端播放器高级版 License，将会出现视频播放失败、黑屏等现象。

设置 License

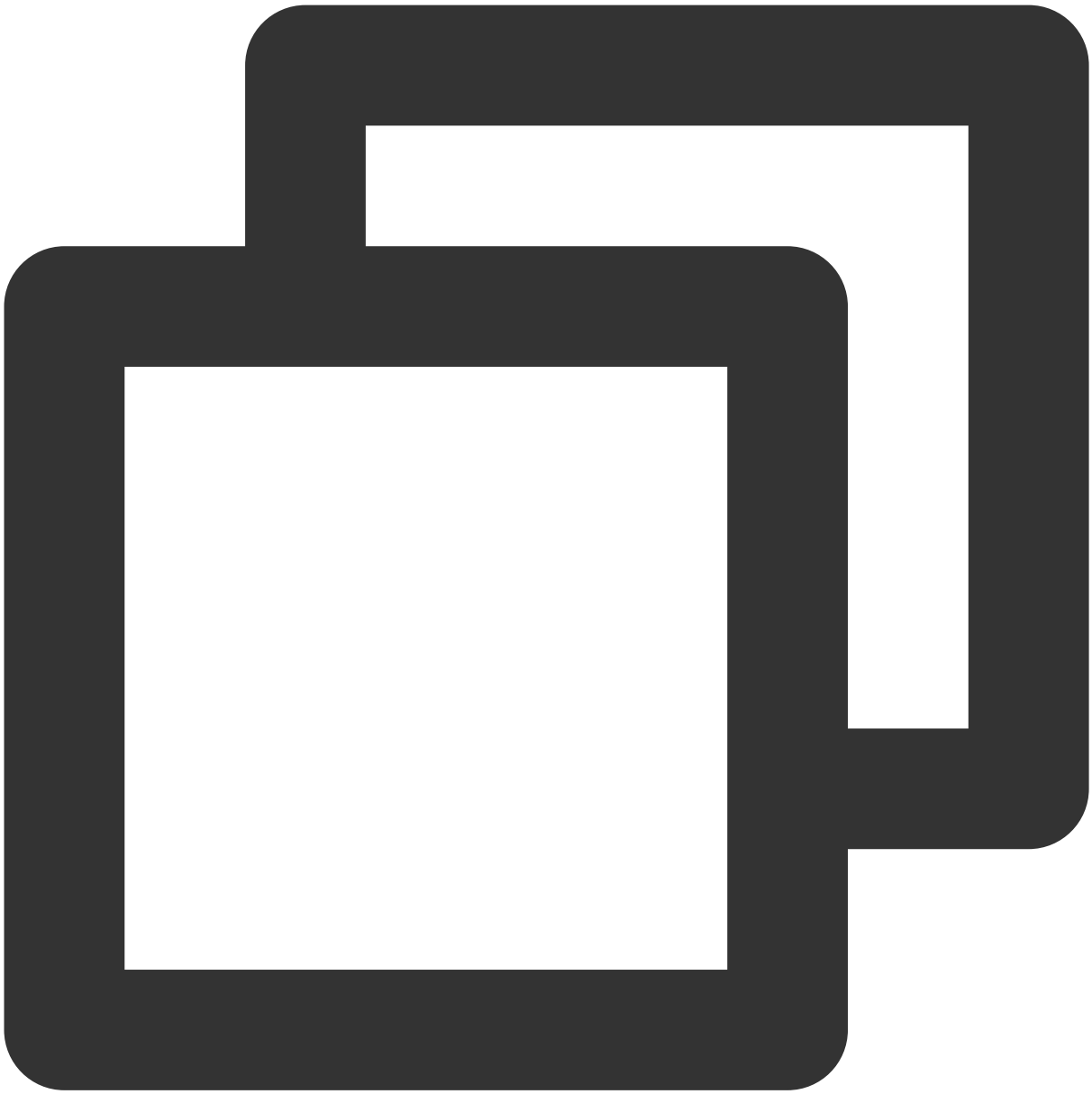
短视频组件需要设置 License 才能使用。



```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
        .licenseKey("Your license key")
        .licenseUrl("Your license url")
        .build();
TUIPlayerCore.init(context, config);
```

添加 UI 组件

在布局文件中，添加短视频 UI 组件。

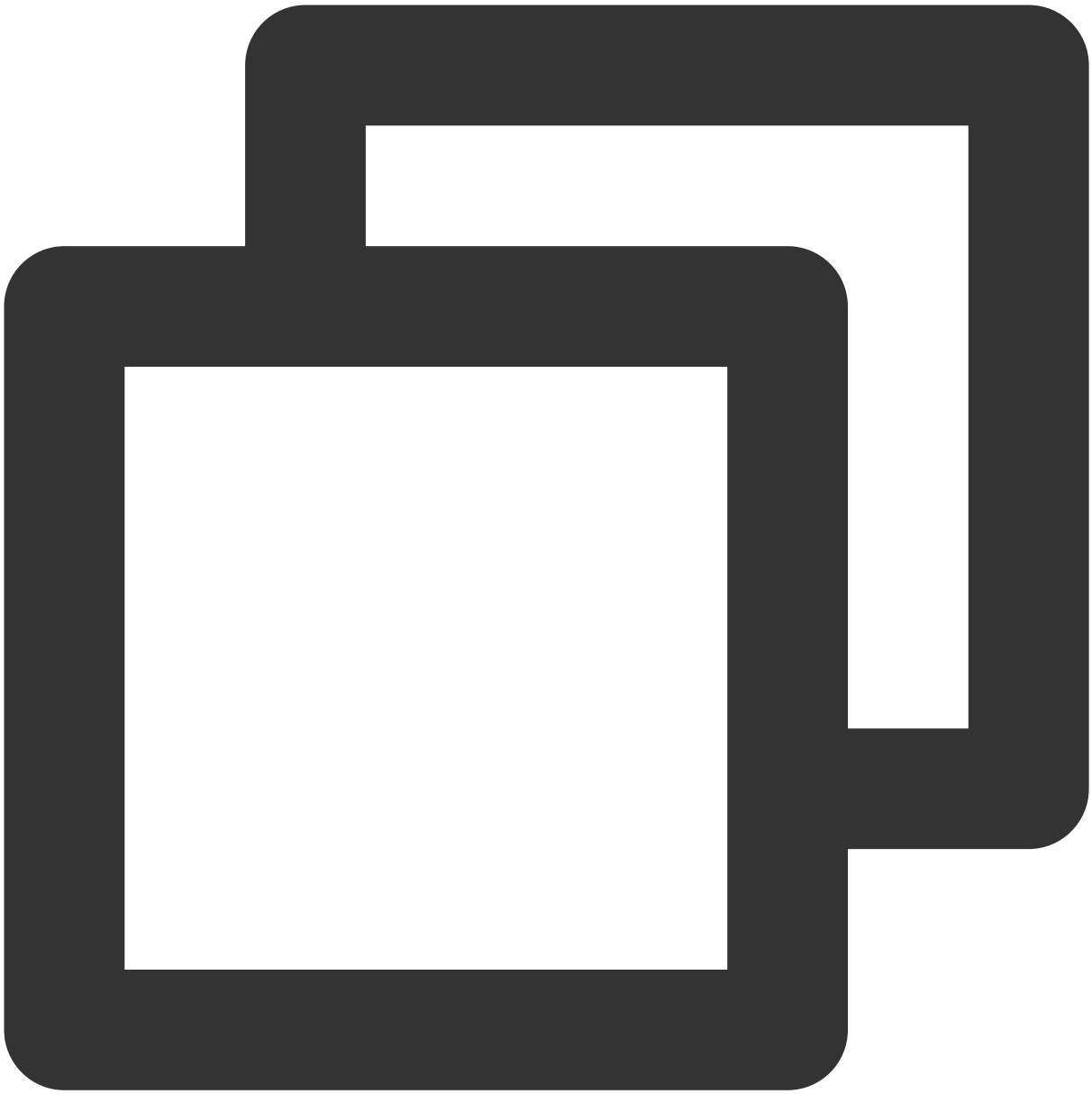


```
<com.tencent.qcloud.tuiplayer.shortvideo.ui.view.TUIShortVideoView
    android:id="@+id/my_video_view"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
```

设置生命周期

设置生命周期后，组件内会根据当前 Activity 生命周期，自行对组件进行暂停、继续播放以及释放。例如在 App 退出后台的时候，短视频会自行暂停播放，当返回 App 之后，会继续播放视频。**也可不设置该周期，根据业务需要自**

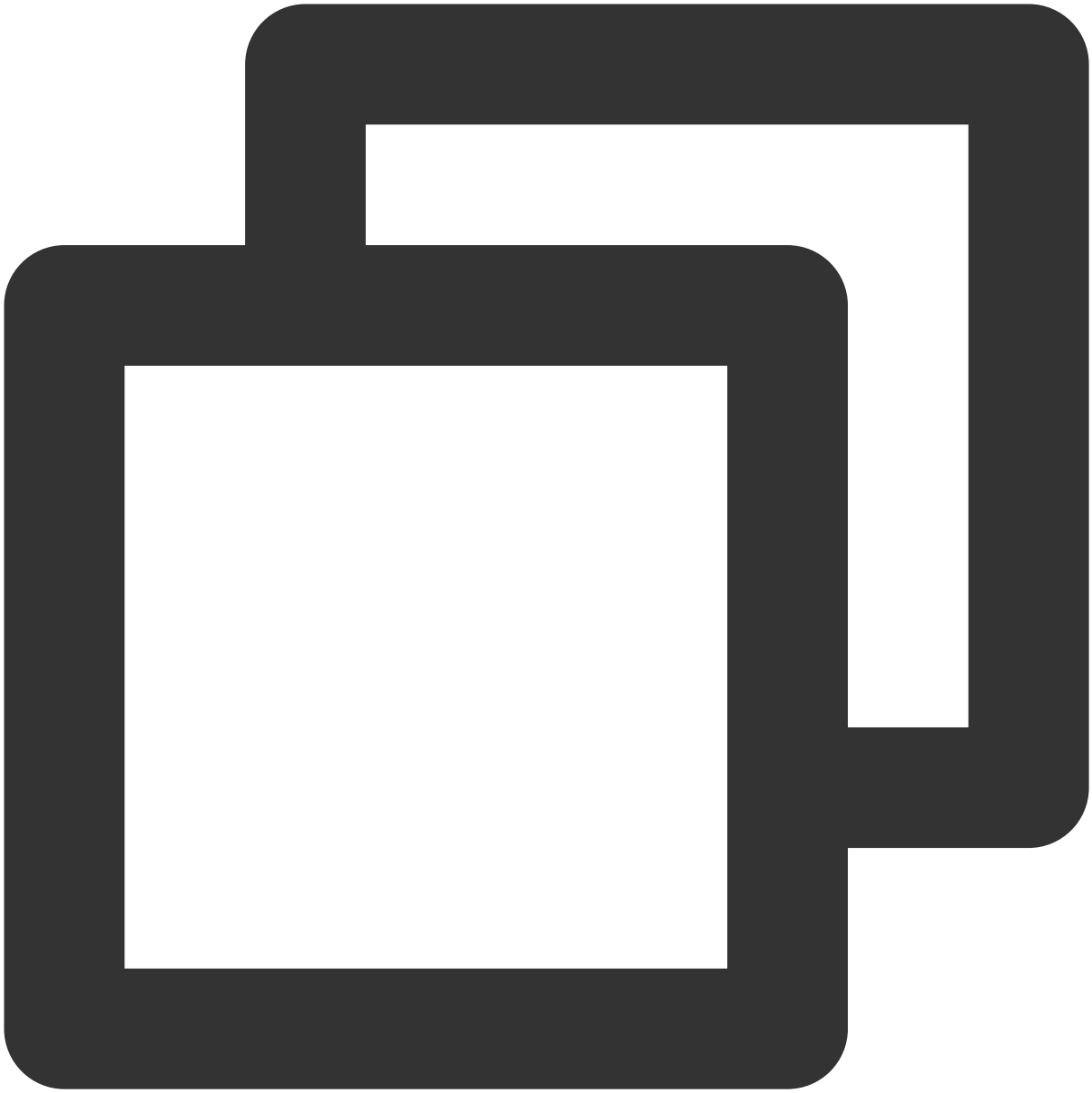
行对短视频进行控制。



```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

配置监听

TUIShortVideoView 的监听提供了多个回调，如下代码演示：



```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
  @Override
  public void onPageChanged(int index, TUIVideoSource videoSource) {
      if (index >= mSuperShortVideoView.getCurrentDataCount() - 1) {
        // append next page data
        mSuperShortVideoView.appendModels(data);
      }
  }

  @Override
  public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
```

```
        // add your vod layer to here
        layerManger.addLayer(new TUICoverLayer());
    }

    @Override
    public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
        // add your live layer to here
    }

    @Override
    public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
        // add your custom layer to here
    }

    @Override
    public void onNetStatus(TUIVideoSource model, Bundle bundle) {
    }
  });
```

当每次页面位置发生变动的时候，会回调 `onPageChanged` 方法，此处可以做类似分页加载的能力。

当列表创建布局的时候，会根据你添加数据的类型，回调 `onCreateVodLayer` 、 `onCreateLiveLayer` 或者 `onCreateCustomLayer` 方法。

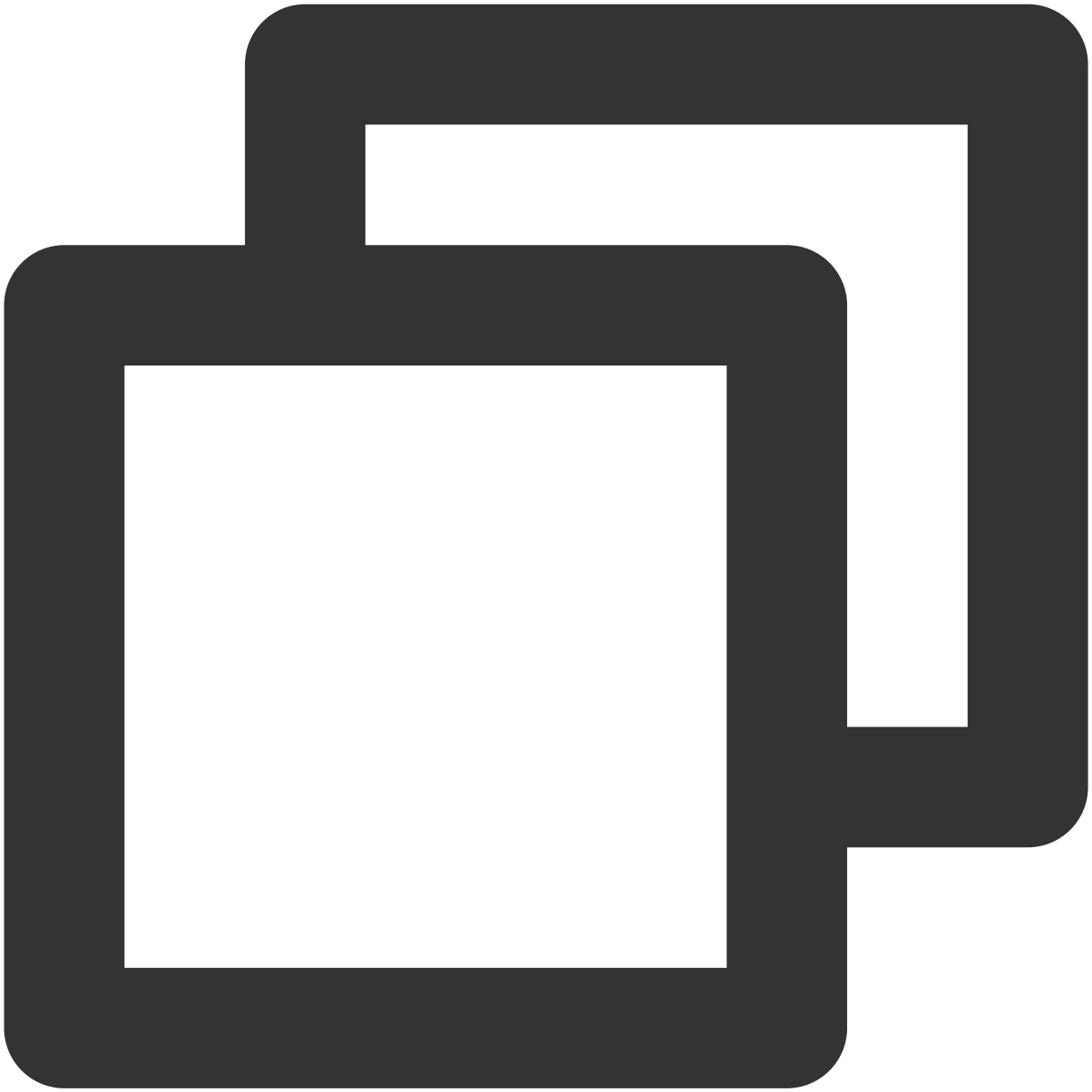
其中如果 setModels 时，数据是 TUIVideoSource 类型的，这里将会回调 `onCreateVodLayer` ，如果是 TUILiveSource 类型的，将会回调 `onCreateLiveLayer` ，如果是继承 TUIPlaySource 实现的自定义数据类型，将会回调 `onCreateCustomLayer` 。

此处还会将 `TUIPlayerSource` 中的 `extViewType` 回调到这里，用于业务根据 viewType 区分不同的 layer 组，layer 的创建会在自定义图层中讲到。

视频开始播放之后，会将当前正在播放的视频的网络状态通过 `onNetStatus` 回调出来，回调详情可查看这里。

填充数据

使用 `setModels` 可以设置数据，并清空掉原本的数据，视频也会从设置的数据源的第一个视频开始重新播放，使用 `appendModels` 可以追加数据，用于分页操作，**当填充完数据之后，会自动从第一个视频开始播放，如果不需要自动播放，可以将第一个视频调用 `setAutoPlay` 设置为 false。**



```
// 点播
TUIVideoSource videoSource = new TUIVideoSource();
videoSource.setCoverPictureUrl(model.placeholderImage);
// fileId填写，fileId和url只需要填写一个
videoSource.setAppId(model.appid);
videoSource.setPSign(model.pSign);
videoSource.setFileId(model.fileid);
// url填写
videoSource.setVideoURL(model.videoURL);
shortVideoData.add(videoSource);
```

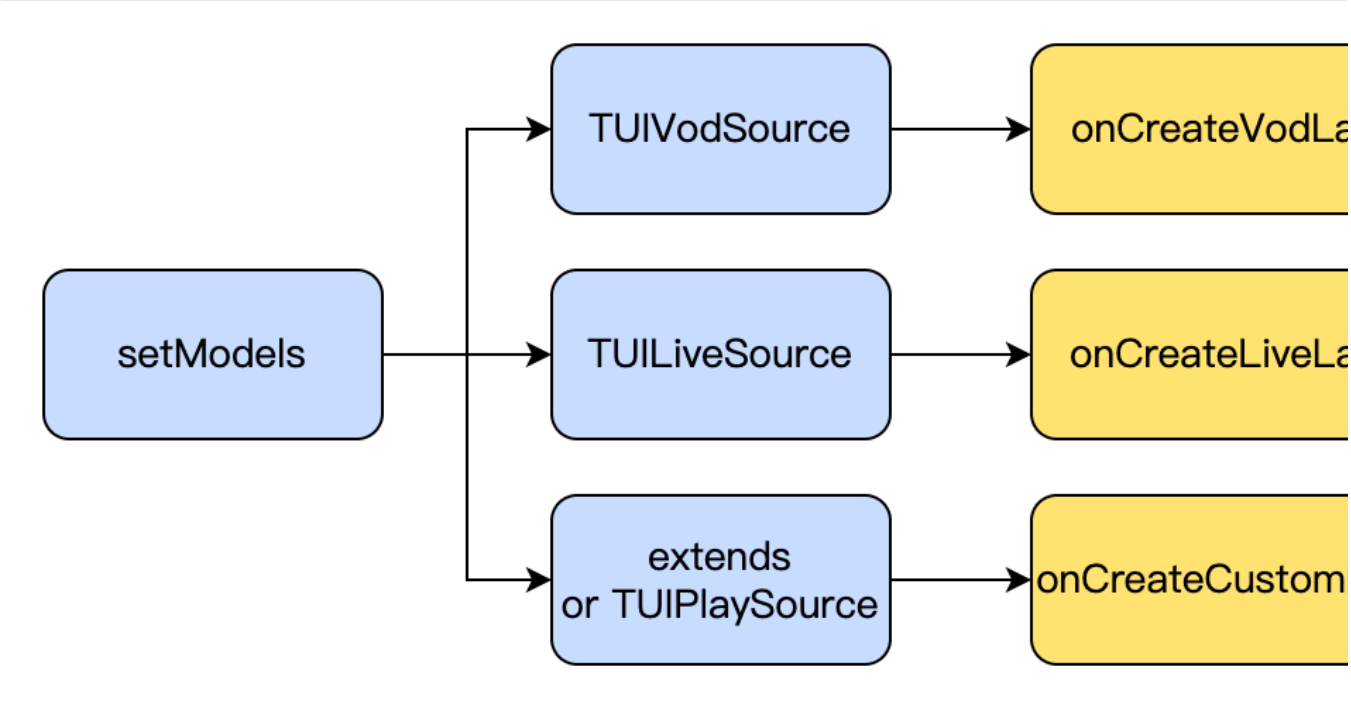
```
// 直播
TUILiveSource liveSource = new TUILiveSource();
// 直播资源url
liveSource.setUrl(liveUrl);
liveSource.setCoverPictureUrl(coverUrl);
shortVideoData.add(liveSource);

// 自定义，DemoImgSource继承自TUIPlaySource,自定义数据，此处可根据业务需求定制不同的数据
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// 设置数据
mSuperShortVideoView.setModels(shortVideoData);

// 如果是分页操作，可选择追加数据
mSuperShortVideoView.appendModels(shortVideoData);
```

自定义图层

一、简介

1. 图层类别

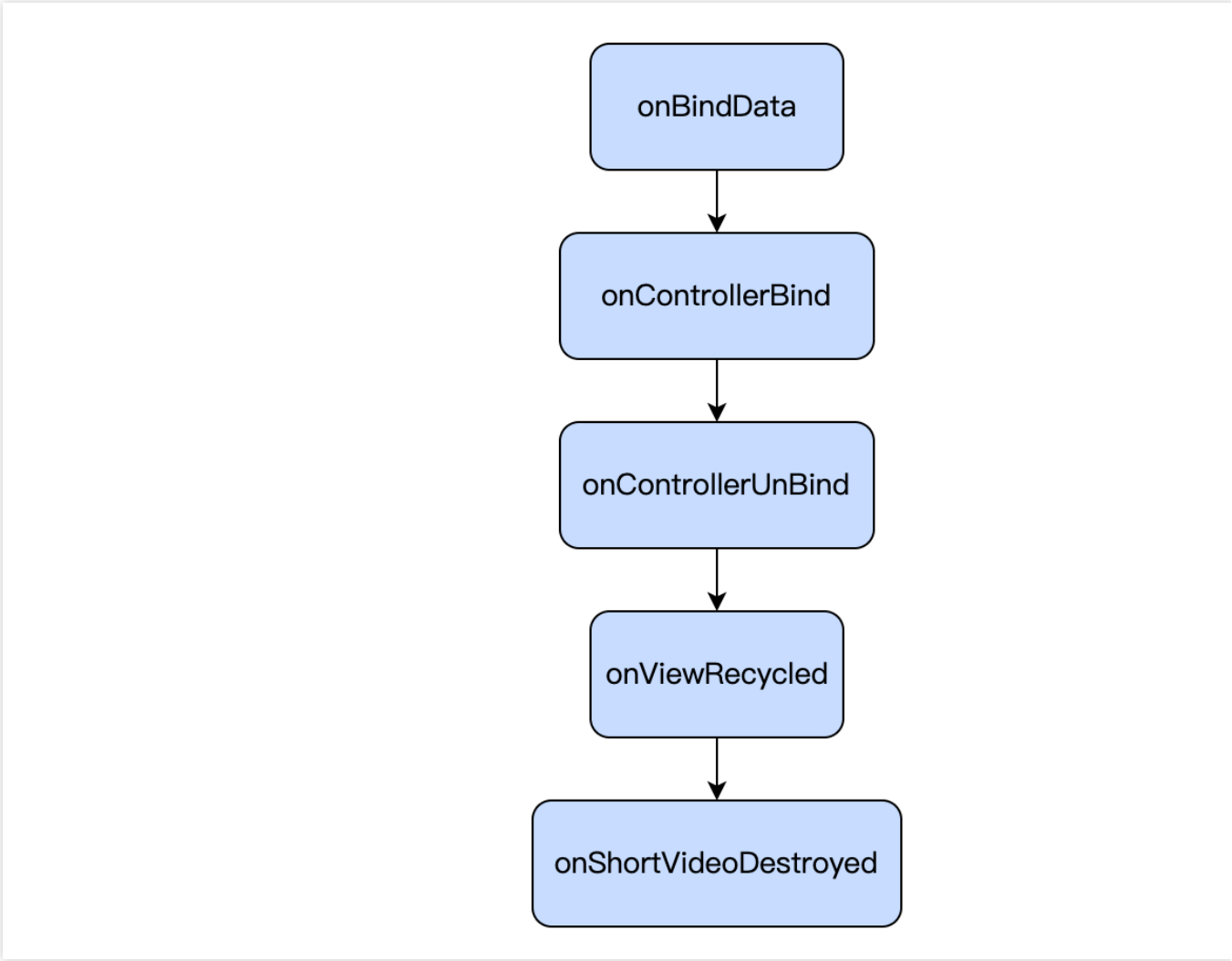
如果需要对 TUI 短视频进行 UI 的定制，需要使用短视频的图层能力。
Android TUI 短视频，采用图层管理的方式，提供给每个短视频页面自定义 UI 能力。通过图层管理器，每个页面可以进行对象化管理，更好的处理视频 UI 与播放器和视频组件之间的交互。
目前有三种页面，点播、直播和自定义页面，三个页面分别对应的 layer 为 `TUIVodLayer` 、 `TUILiveLayer` 和 `TUICustomLayer` 。可以按照需求来继承不同的 layer。不同的 layer 基类都针对其场景提供了符合场景的接口和回调。
对应关系如下图所示：



图层的显示和隐藏，会直接对 View 进行添加和移除，不会造成界面过度渲染。
图层会根据添加的顺序，来决定界面的展示顺序，先添加的在最上层，会优先展示，后添加的在最底层，会被之前添加的覆盖。
短视频列表由于有页面复用机制，当图层中有业务数据相关的UI展示时，需要在 `onBindData` 绑定数据的时候，对界面UI进行重置或者重新设置新值。

2. 图层生命周期

三种 Layer 的生命周期如下图所示



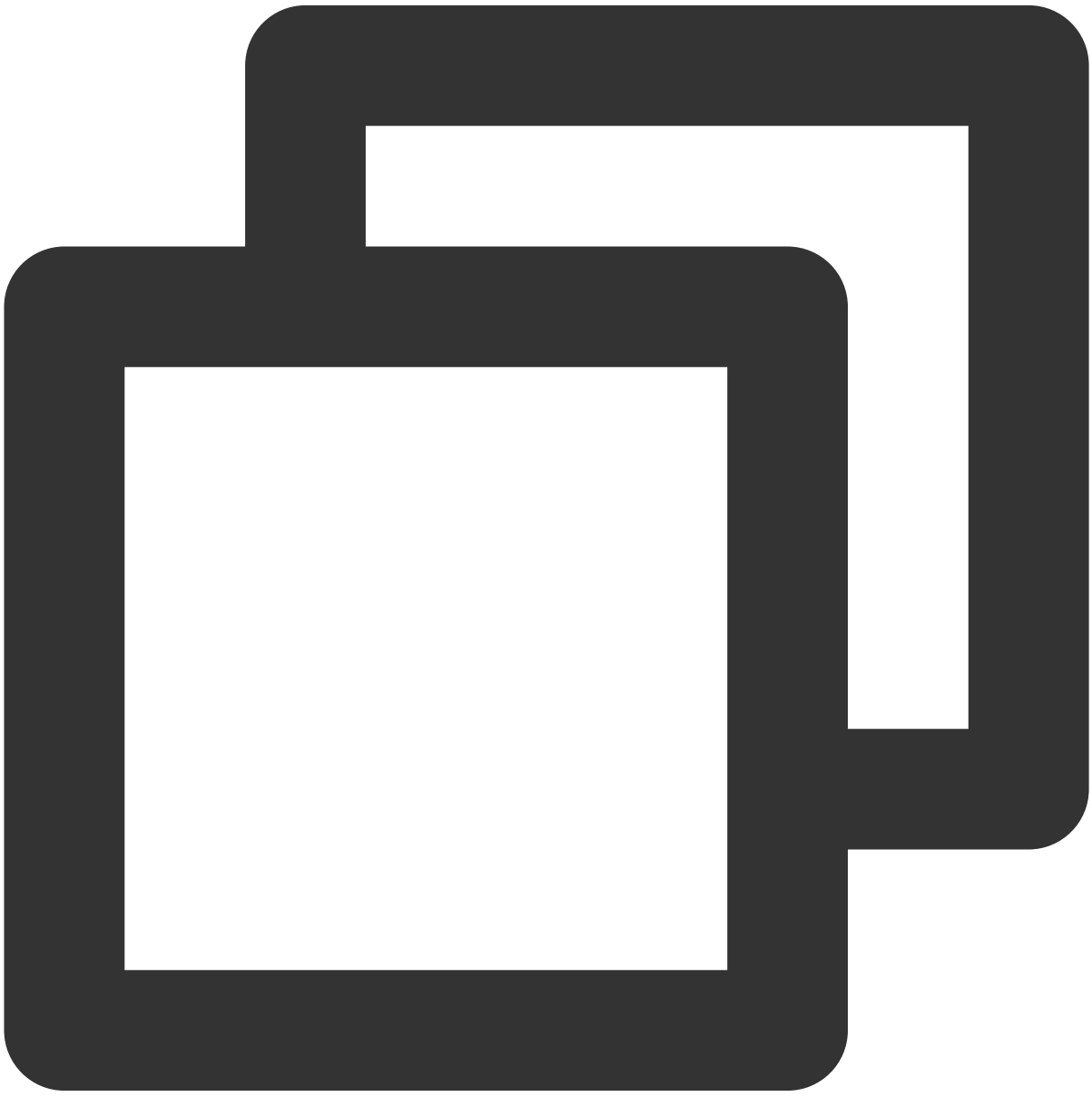
其生命周期含义如下：

生命周期名称	含义
onBindData	代表当前Layer已经绑定数据，可以在该生命周期中，做一些静态UI的数据初始化工作。
onControllerBind	该页面已经是当前短视频列表正在展示的页面，**除了自定义页面 `TUICustomLayer` 以外**，此时调用 `getPlayer()` 和 getController() 将不再为空，可以对播放器、页面容器进行操作。
onControllerUnBind	页面被划走，该生命周期之后将无法再获得到播放器和 videoView，该生命周期可以做一些资源的回收、界面重置工作。
onViewRecycled	页面被回收，会用到其他数据和播放器上，建议在该生命周期中重置界面所有数据，回收相关资源。
onShortVideoDestroyed	TUI 组件被销毁，即 TUIShortVideoView 被调用了release 方法。

二、创建自定义图层

1. 创建自定义图层布局

创建自定义图层，以点播 layer 为例，需要继承 TUIVodLayer，然后实现自己需要的图层。
以视频详情图层为例，首先需要实现 createView 和 tag 方法。createView 是图层 view 的创建方法，tag 是业务侧用来区分图层的字符串标签。
createView 会在 layer 添加到 LayerManager 的时候被调用。



```
@Override
public View createView(ViewGroup parent) {
```

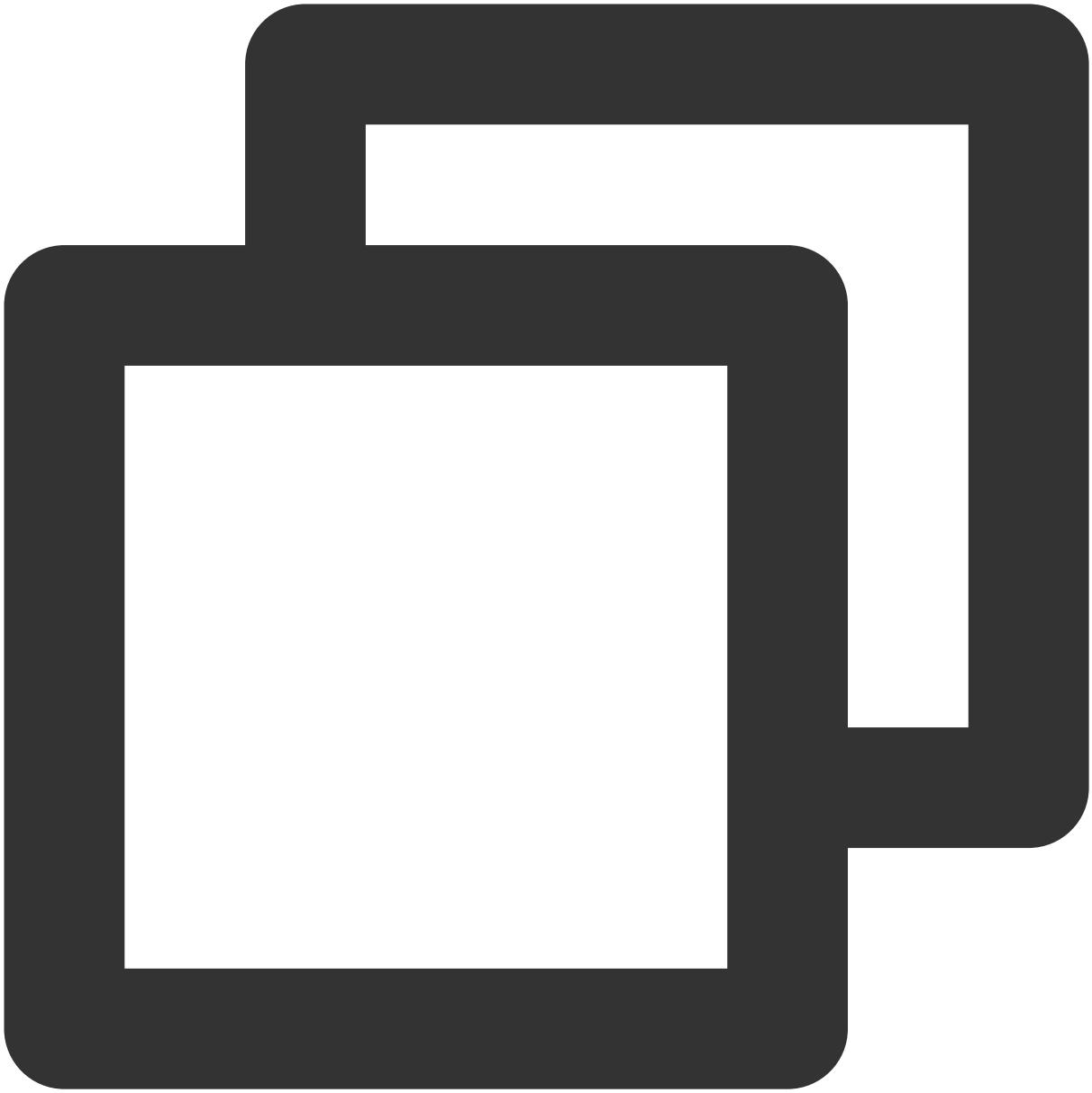
```
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.player_video_info_layer, parent, false);
    mSeekBar = view.findViewById(R.id.vsb_tui_video_progress);
    mTvProgress = view.findViewById(R.id.tv_tui_progress_time);
    mIvPause = view.findViewById(R.id.iv_tui_pause);
    mSeekBar.setListener(this);
    return view;
}

@Override
public String tag() {
    return "TUIVideoInfoLayer";
}
```

createView 中创建了一个 View 并返回。这里可以使用 LayoutInflater 从 XML 中加载布局，也可以使用代码直接创建布局。

2. 展示布局

View 创建完成之后，需要在合适的时机去展示。TUIBaseLayer 提供了丰富的事件回调。详细信息可以参见 图层回调。

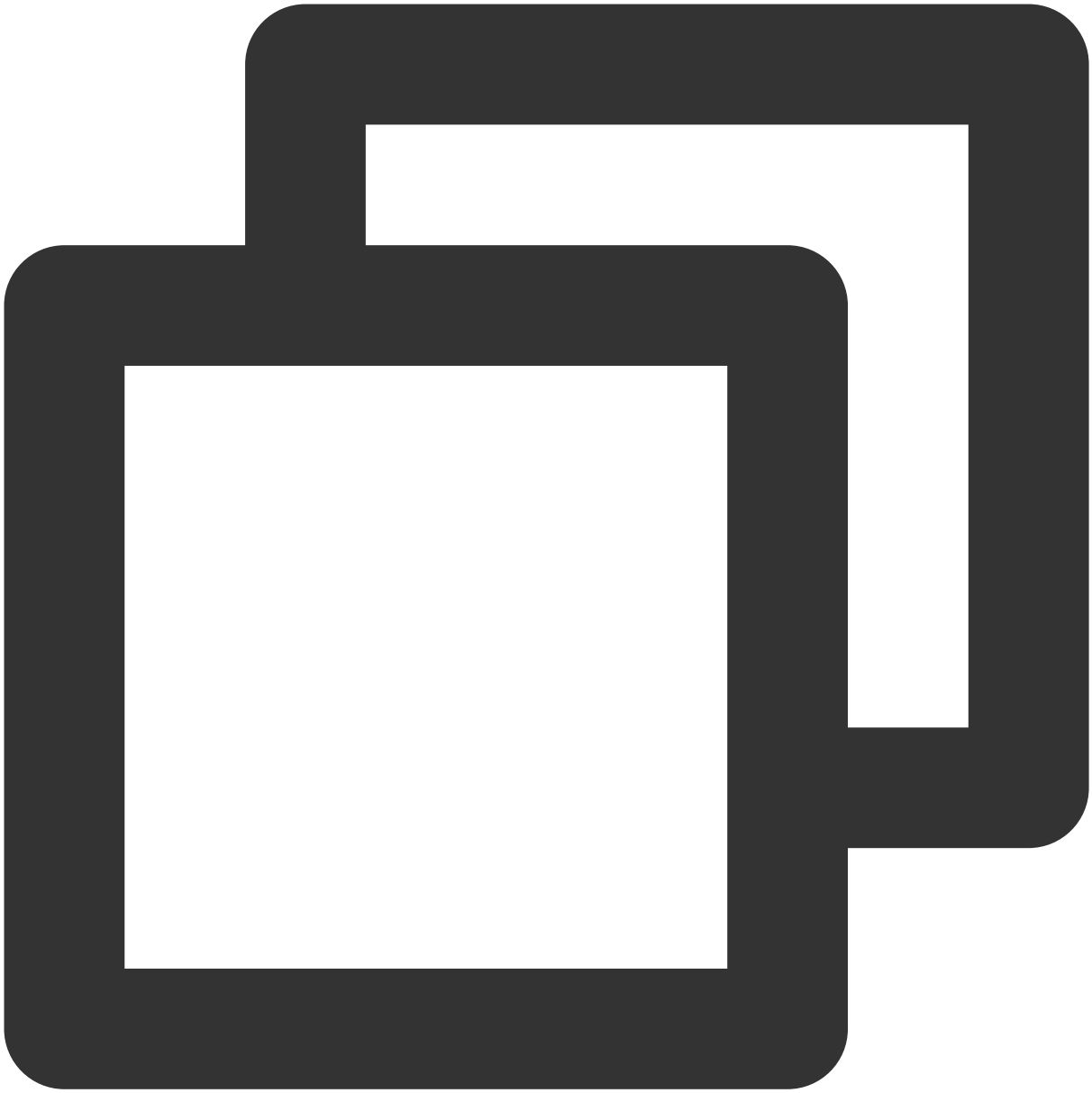
视频详情图层在获得数据的时候就可以展示布局了。所以在 onBindData 把图层展示出来。



```
@Override
public void onBindData(TUIVideoSource videoSource) {
    show();
}
```

3. 操作自己的组件

还可以在其他事件中对自己创建的组件进行操作，这里以点播 layer 为例，需要先将自己的组件声明为成员变量，在 `onCreateView` 中赋值，随后在播放器事件，例如暂停按钮的显示和隐藏，以及播放进度的回调，参见如下代码。

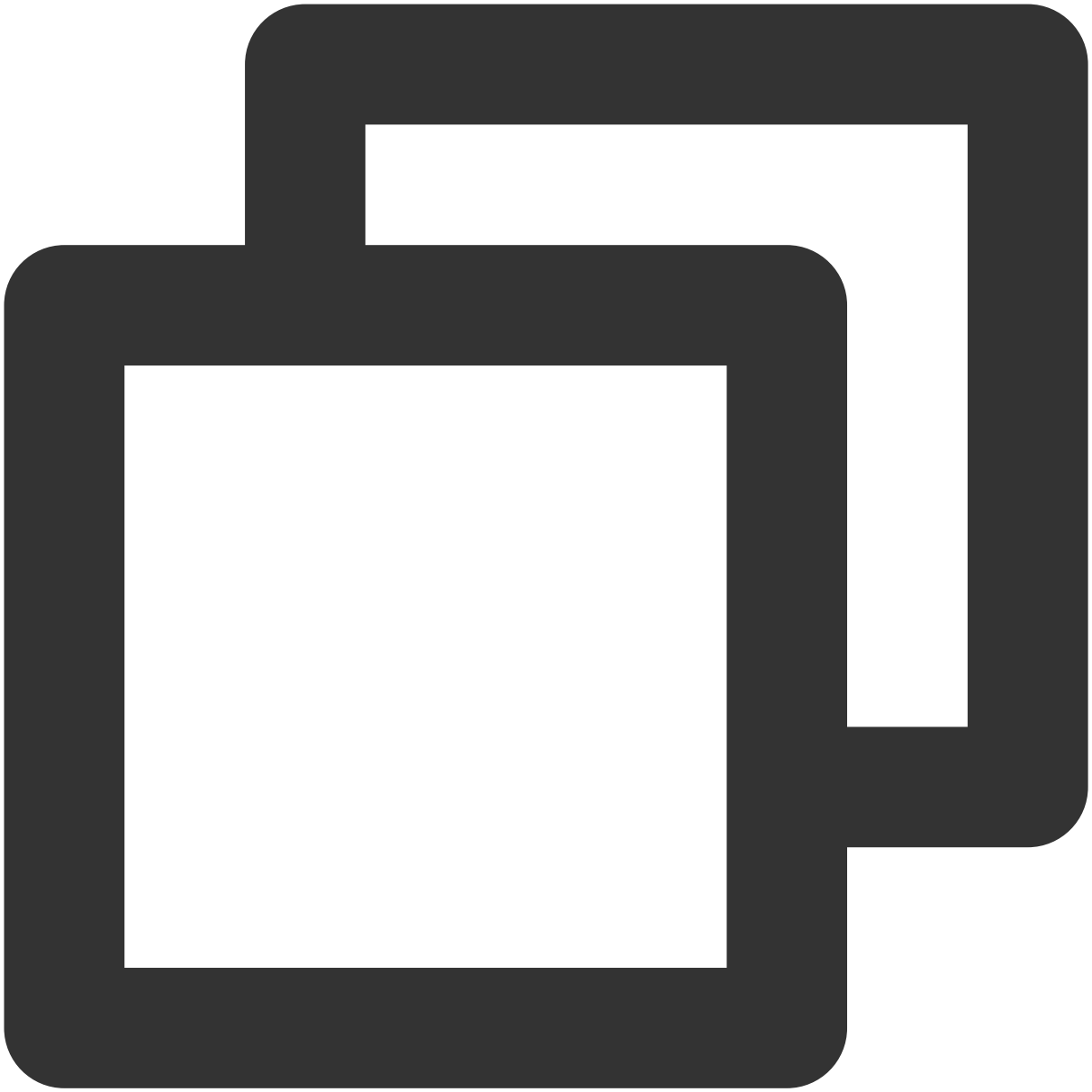


```
@Override
public void onPlayBegin() {
    super.onPlayBegin();
    if (null != mIvPause) {
        mIvPause.setVisibility(View.GONE);
    }
}

@Override
public void onPlayPause() {
    super.onPlayPause();
```

```
    if (null != mIvPause) {
        mIvPause.setVisibility(View.VISIBLE);
    }
}

@Override
public void onPlayProgress(long current, long duration, long playable) {
    videoDuration = duration;
    if (null != mSeekBar) {
        // ensure a refresh at every percentage point
        int progressInt = (int) (((1.0F * current) / duration) * 100);
        if(lastProgressInt != progressInt) {
            setProgress(progressInt / 100F);
            lastProgressInt = progressInt;
        }
    }
}
```

4. 控制播放器

除了接收来自播放器的事件之外，还可以对播放器进行控制。例如调用播放器进行进度的跳转。

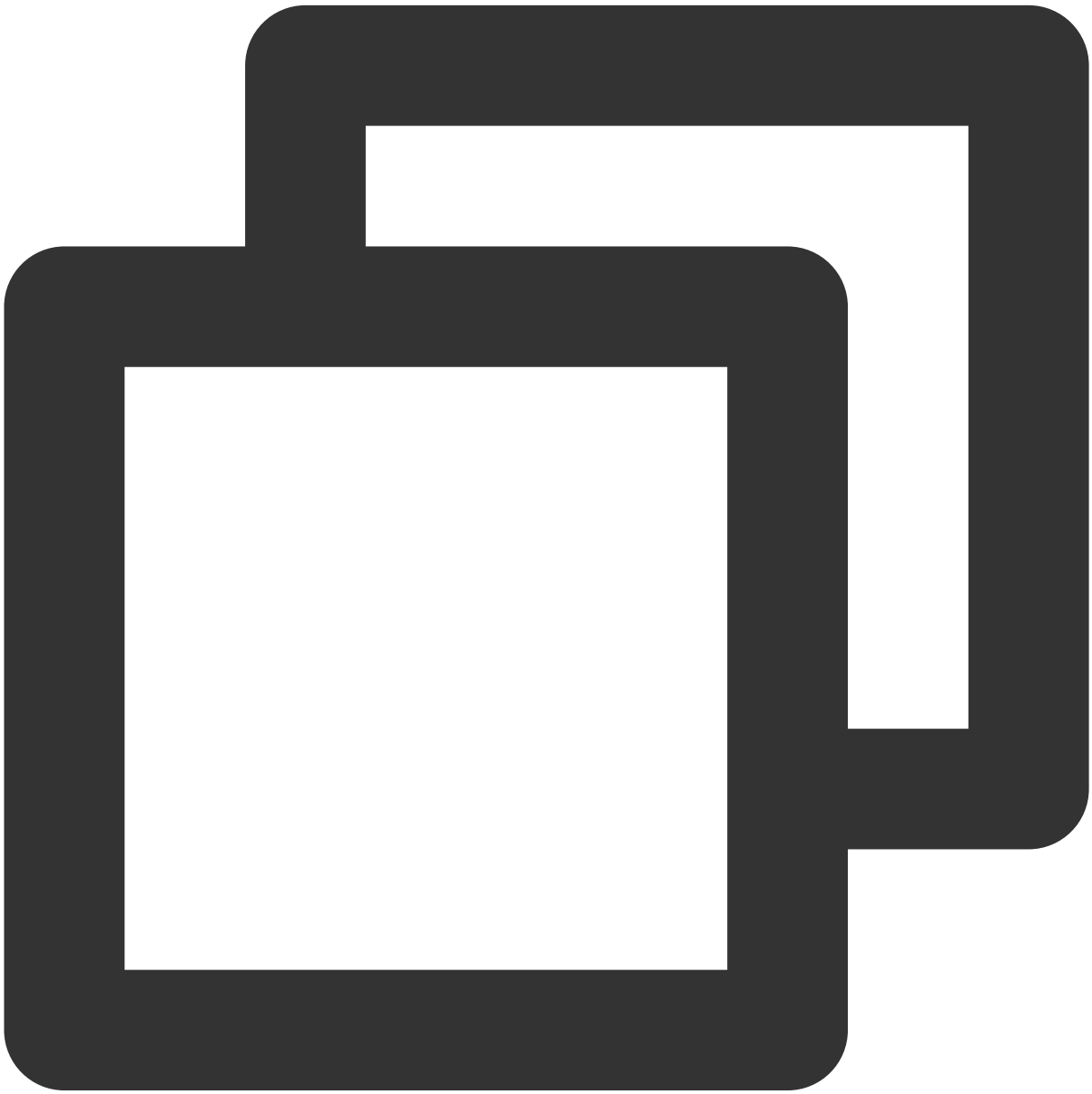


```
@Override
public void onDragDone(VideoSeekBar seekBar) {
    TUIPlayerController controller = getPlayerController();
    if (null != controller && videoDuration > 0) {
        controller.seekTo((int) ((videoDuration * seekBar.getBarProgress()) / 1000)
    }
    if (null != mTvProgress) {
        mTvProgress.setVisibility(View.GONE);
    }
}
```

目前在 TUIBaseLayer 中，可以通过 getVideoView 获得当前 page 的 VideoView 对象，通过 getPlayerController 获得当前视频的播放控制器（只有当前 page 是当前短视频列表正在播放的视频的时候才会有），getPlayer 获得当前播放器对象。由于播放器和视频view会在滑动过程中被释放或者复用，所以这三个对象在获取的时候，都可能会获得空对象，需要进行判空。

5. 图层被回收时进行释放

在图层被回收的时候，需要进行一些释放操作。防止一些外部的对象对图层造成持有，产生了内存泄漏。

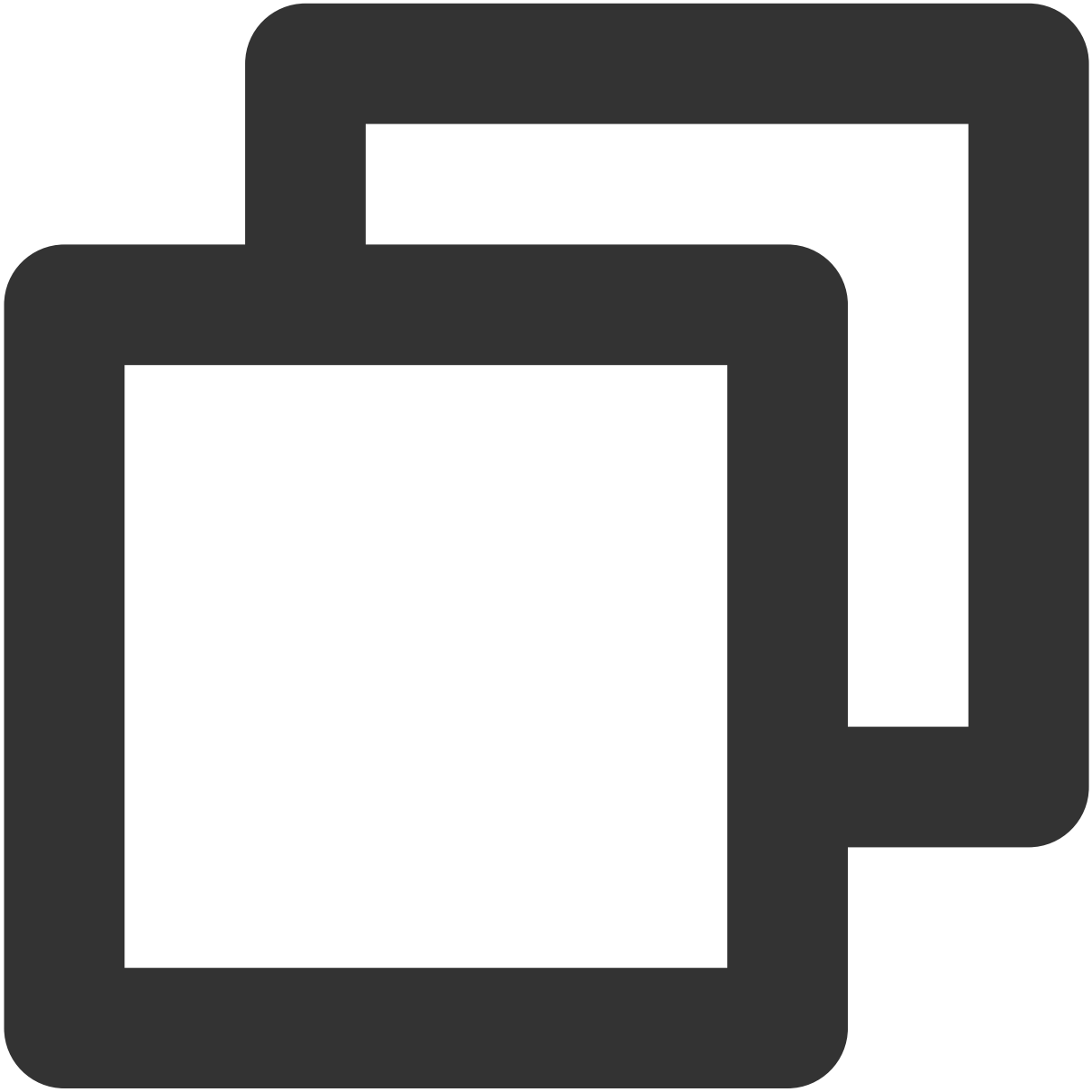


```
@Override
public void onViewRecycled(TUIBaseVideoView videoView) {
```

```
    // release your resource
}
```

6. 监听是否滑动到当前layer

如果需要监听当前 page 是否是当前播放视频，可以对 controller 进行监听，**当触发 `onControllerBind` 的时候，该 layer 被 controller 绑定，代表该 layer 的页面即将展示出来开始播放，当触发 `onControllerUnBind` 的时候，controller 发生解绑，代表页面被划走。**



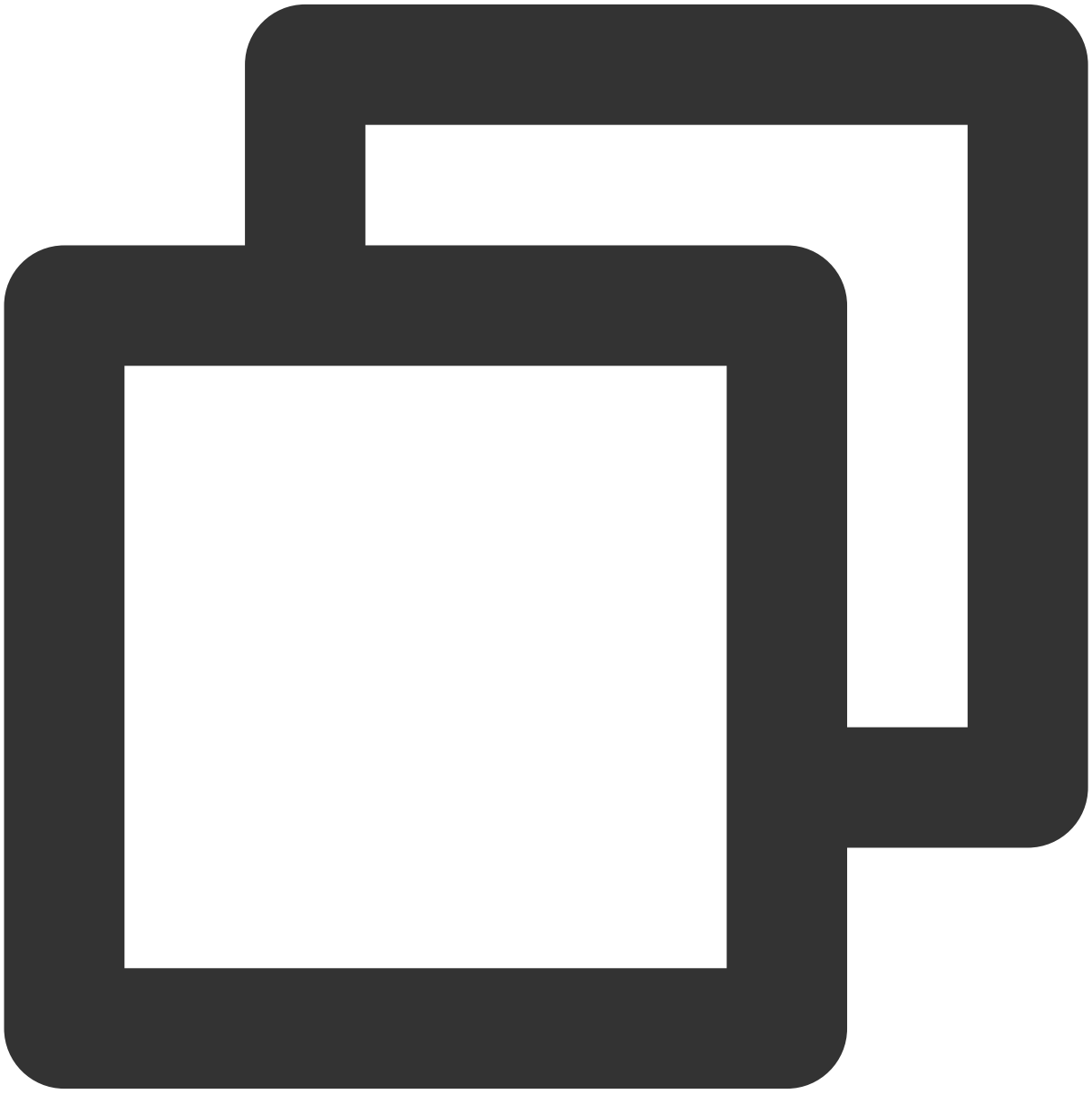
```
@Override
public void onControllerUnBind(TUIPlayerController controller) {
```

```
        super.onControllerUnBind(controller);
        show();
    }
```

以上代码，是在页面划走之后， controller 解绑，为了后续再划回来不是黑屏，触发显示封面图。

7. 点播通过 onRecFileVideoInfo 回调获取视频信息

当设置的 fileId 视频源的时候，layer 会通过 `onRecFileVideoInfo` 将视频的信息回调出来，参见如下代码：



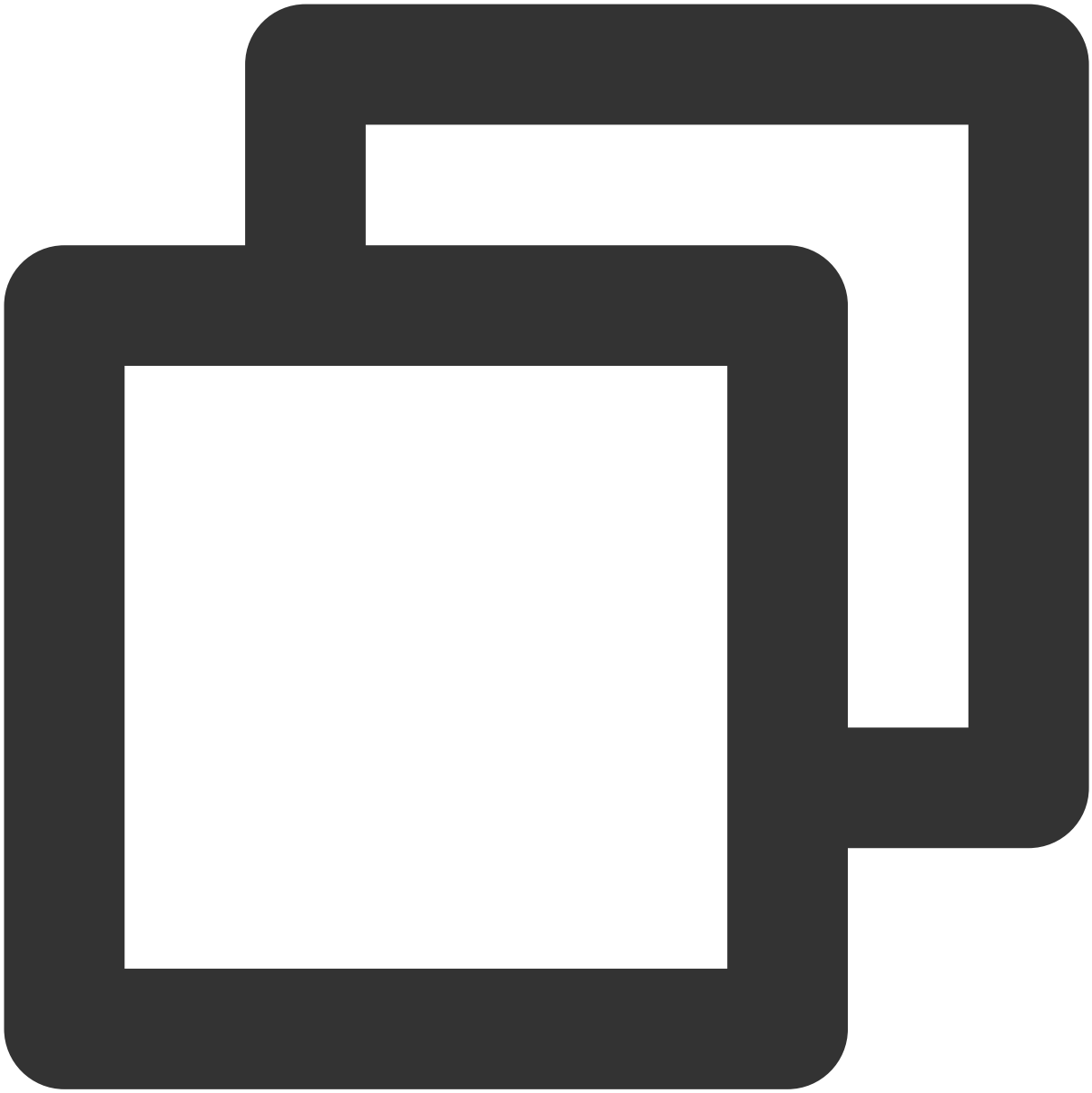
```
@Override
public void onRecFileVideoInfo(TUIFileVideoInfo params) {
```

```
    if(isShowing()) {
        TUIBaseVideoView videoView = getVideoView();
        if (null != videoView && null != params) {
            String coverUrl = params.getCoverUrl();
            if (!TextUtils.isEmpty(coverUrl)) {
                ImageView imageView = getView();
                Glide.with(videoView).load(coverUrl)
                        .centerCrop()
                        .into(imageView);
                coverUrlFromServer = coverUrl;
            }
        }
    }
}
```

该方法会在只使用 fileID 播放的时候回调。会返回视频 URL 链接、封面图、时长、雪碧图等信息。
建议尽可能的通过 URL 传入短视频组件进行播放，并提前赋值好封面图 URL ，这样能够增加短视频加载性能。

8. 通过 onRcvFirstIframe 方法判断视频首帧是否到来

使用方法参见如下代码：

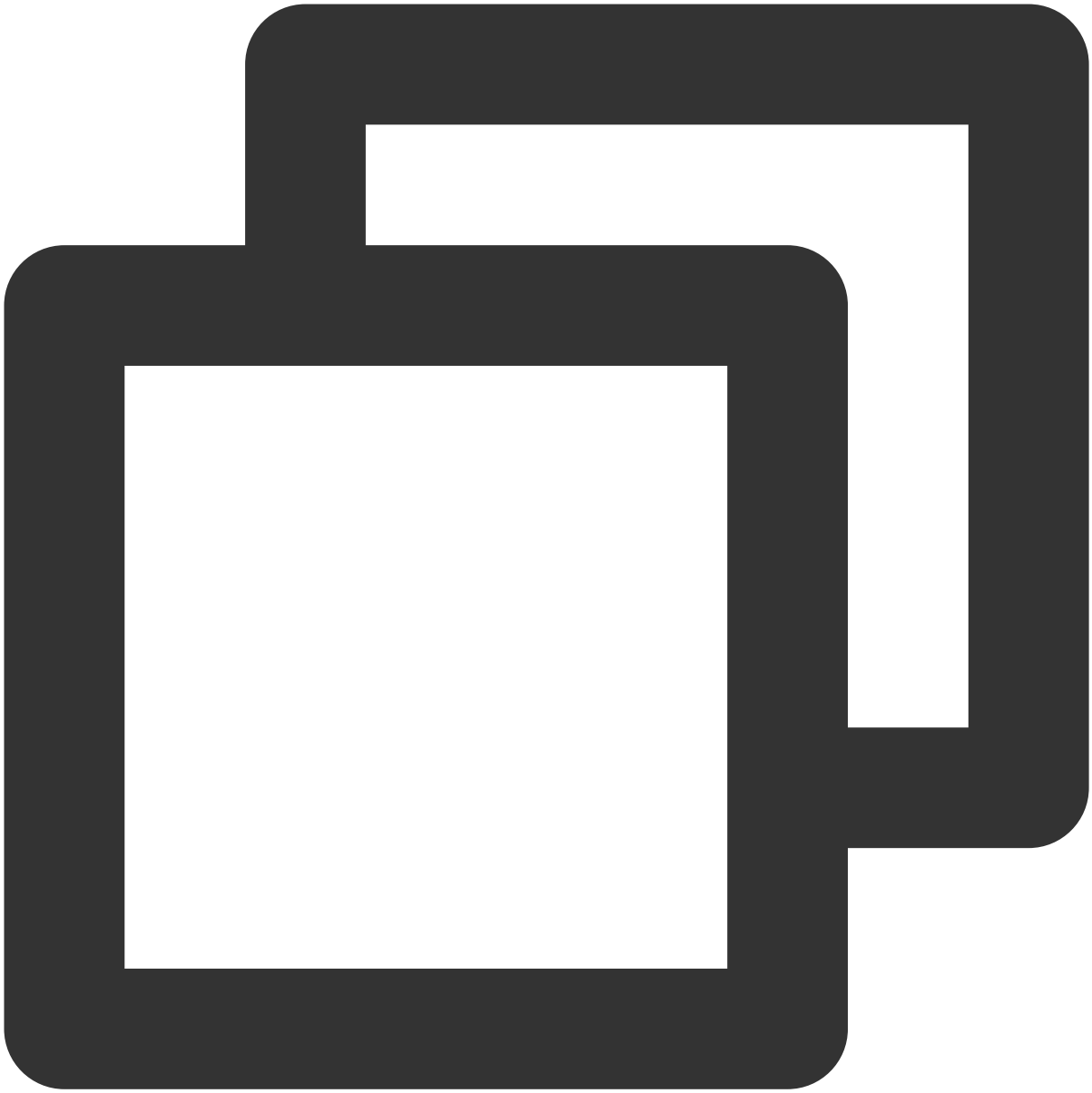


```
@Override
public void onRcvFirstIframe() {
    hidden();
}
```

例如封面图等场景，需要在收到首帧事件后来隐藏封面图。

三、管理图层

当集成短视频组件 TUIShortVideoView 后，使用 TUIShortVideoView 设置监听会在合适的时机回调 item 的创建方法 onCreateItemLayer，来添加或者管理自定义图层。

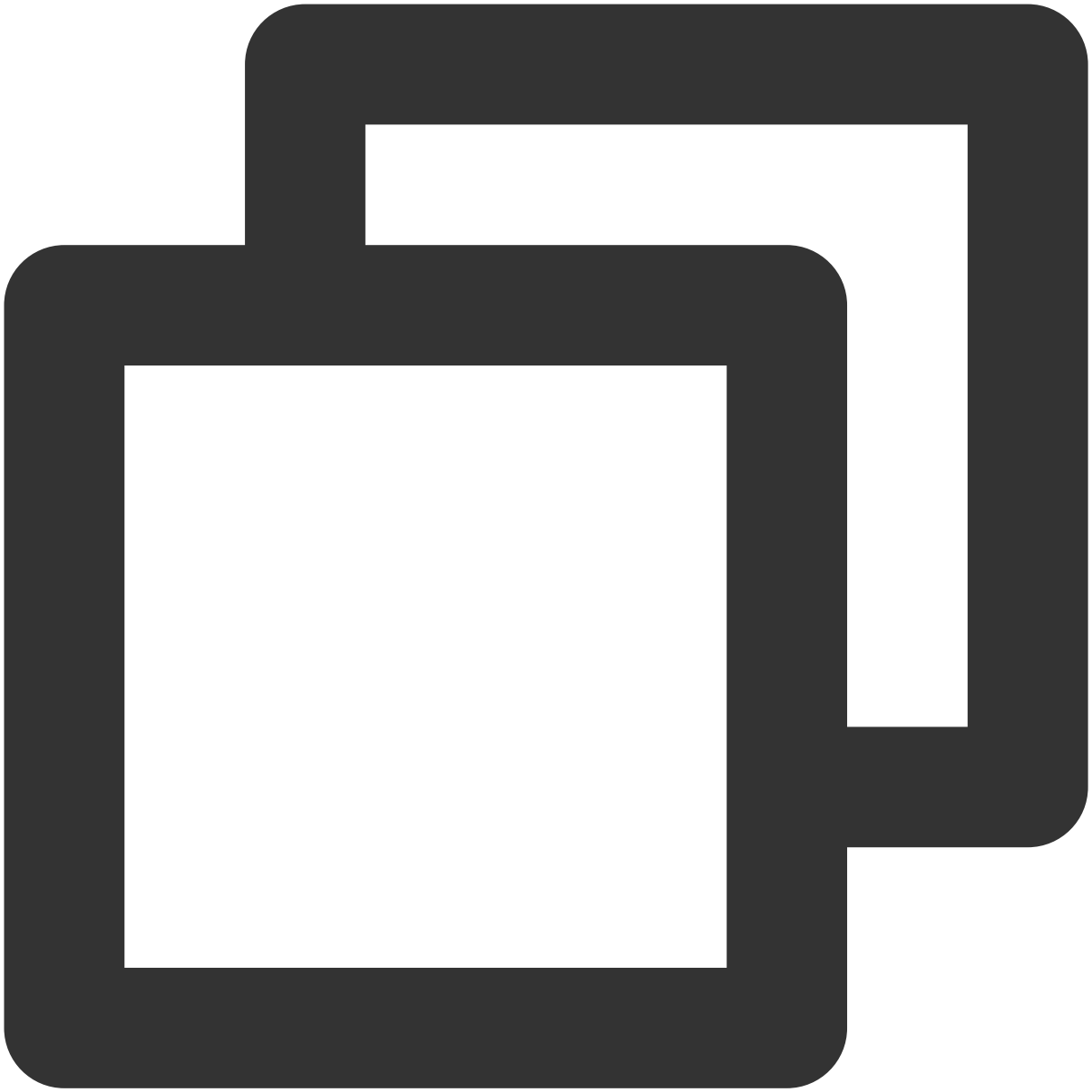


```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {

    // ......

 @Override
  public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
      layerManger.addLayer(new TUIVideoInfoLayer(mShortVideoView, ShortVideoFragmen
      layerManger.addLayer(new TUICoverLayer());
      layerManger.addLayer(new TUILoadingLayer());
      layerManger.addLayer(new TUIErrorLayer());
  }
```

```
    @Override
    public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
        layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView, ShortVideoFra
        layerManager.addLayer(new TUILiveLoadingLayer());
        layerManager.addLayer(new TUILiveErrorLayer());
    }

    @Override
    public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
        if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
            layerManager.addLayer(new PicDisplayLayer());
        }
    }
});
```

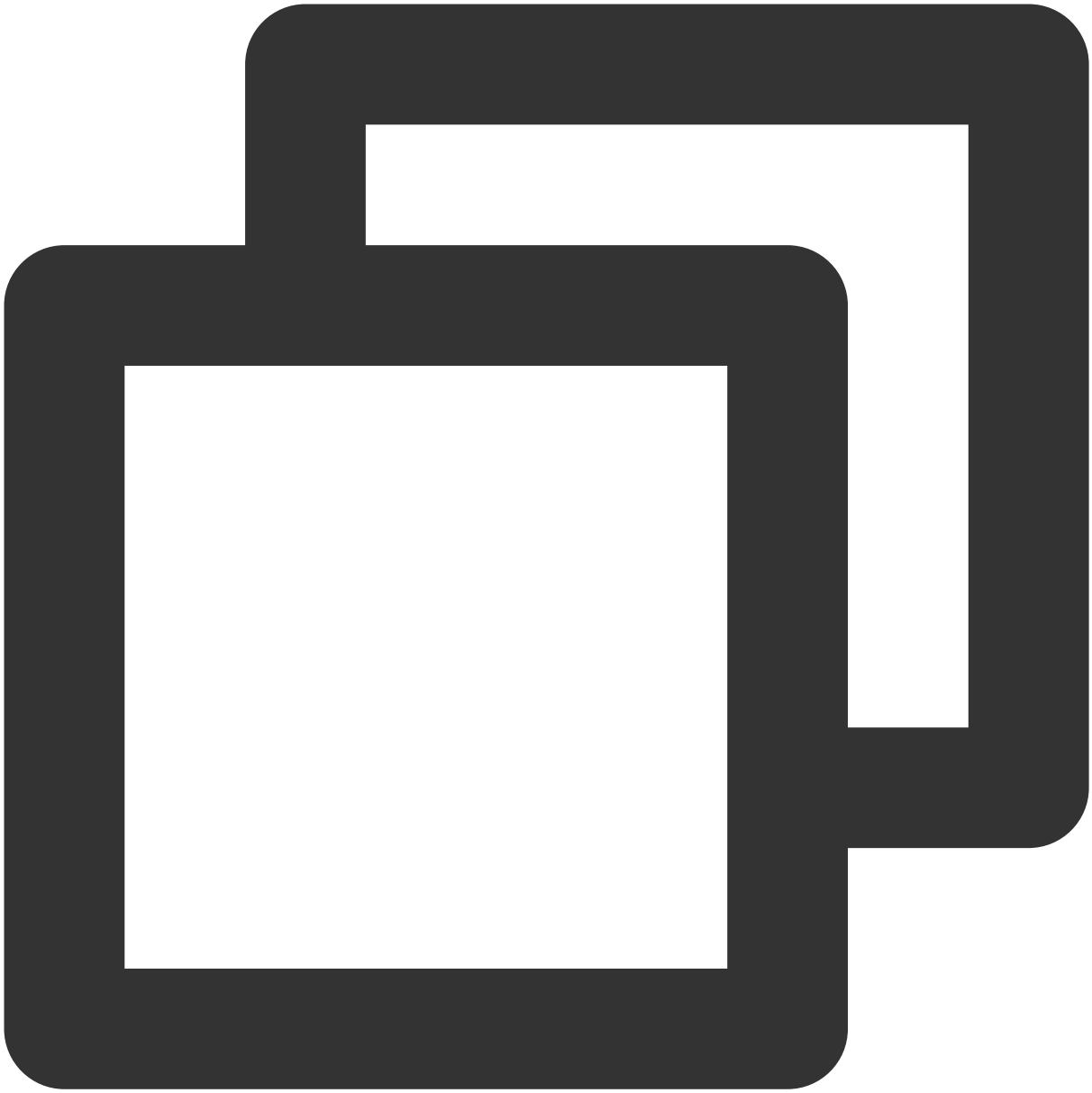
onCreateItemLayer 有两个参数，layerManger 为图层管理器，可以添加、移除、查询图层。添加方式如上图所示。 `viewType` 为当前 page 的视频类型，如果你在 `TUIPlayerSource` 中自定义了 `extViewType` ，这里的 `viewType` 将会是你定义的，如果没有定义，将会根据页面类型返回 `ITEM_TYPE_VOD` 、 `ITEM_TYPE_LIVE` 或者 `ITEM_TYPE_CUSTOM` 。

如果不需要图层，可以将图层移除，**移除之后，layer 中会回调 `unBindLayerManager` 方法。**



```
layerManger.removeLayer(layer);
```

如果需要获得图层的层级，做图层的交互操作，可以通过以下方法获取图层的层级：

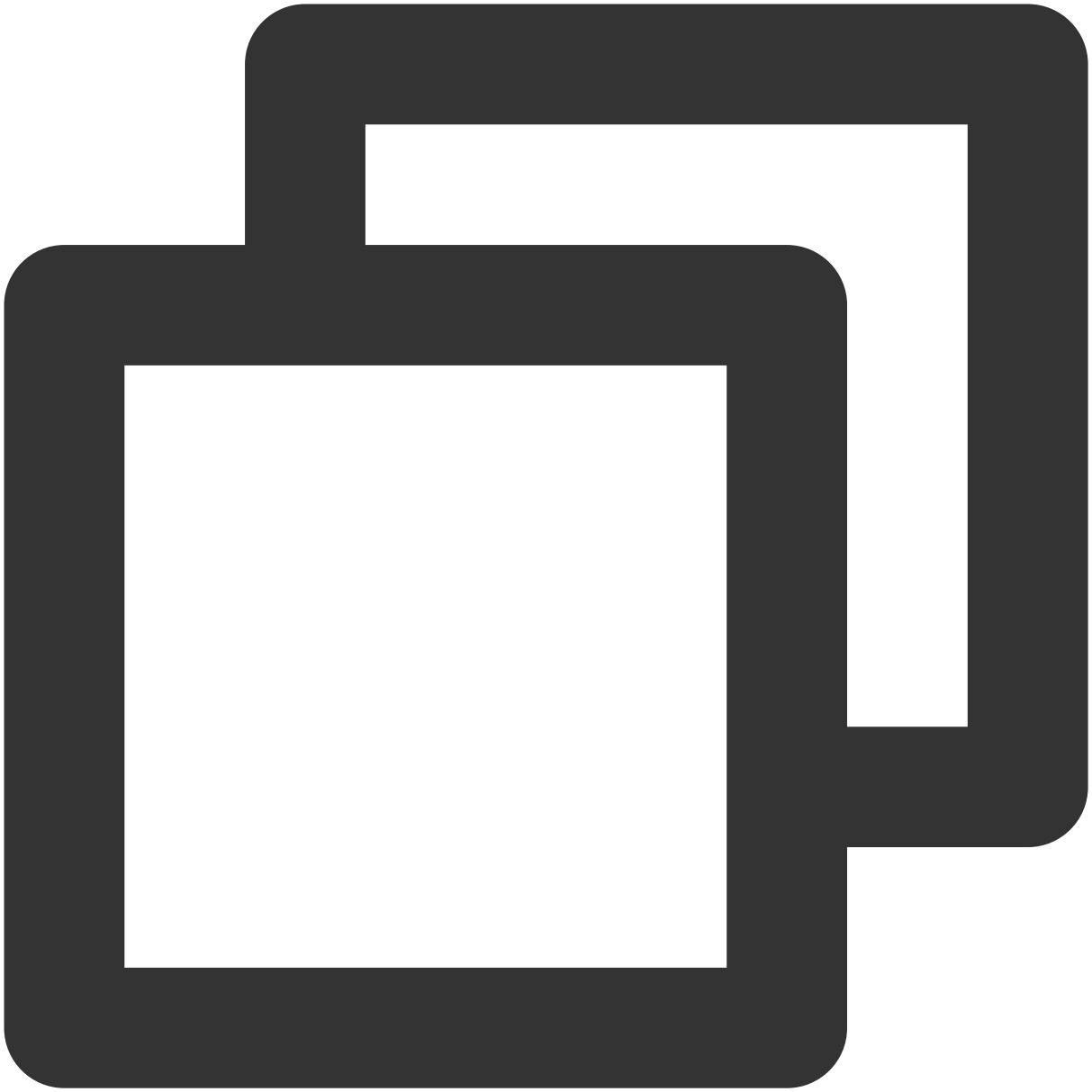


```
layerManger.indexOfLayer(layer);
```

四、使用自定义图层创建图片展示页面

1. 实现自己的自定义数据

这里以展示图片为例，创建一个带图片链接的数据。

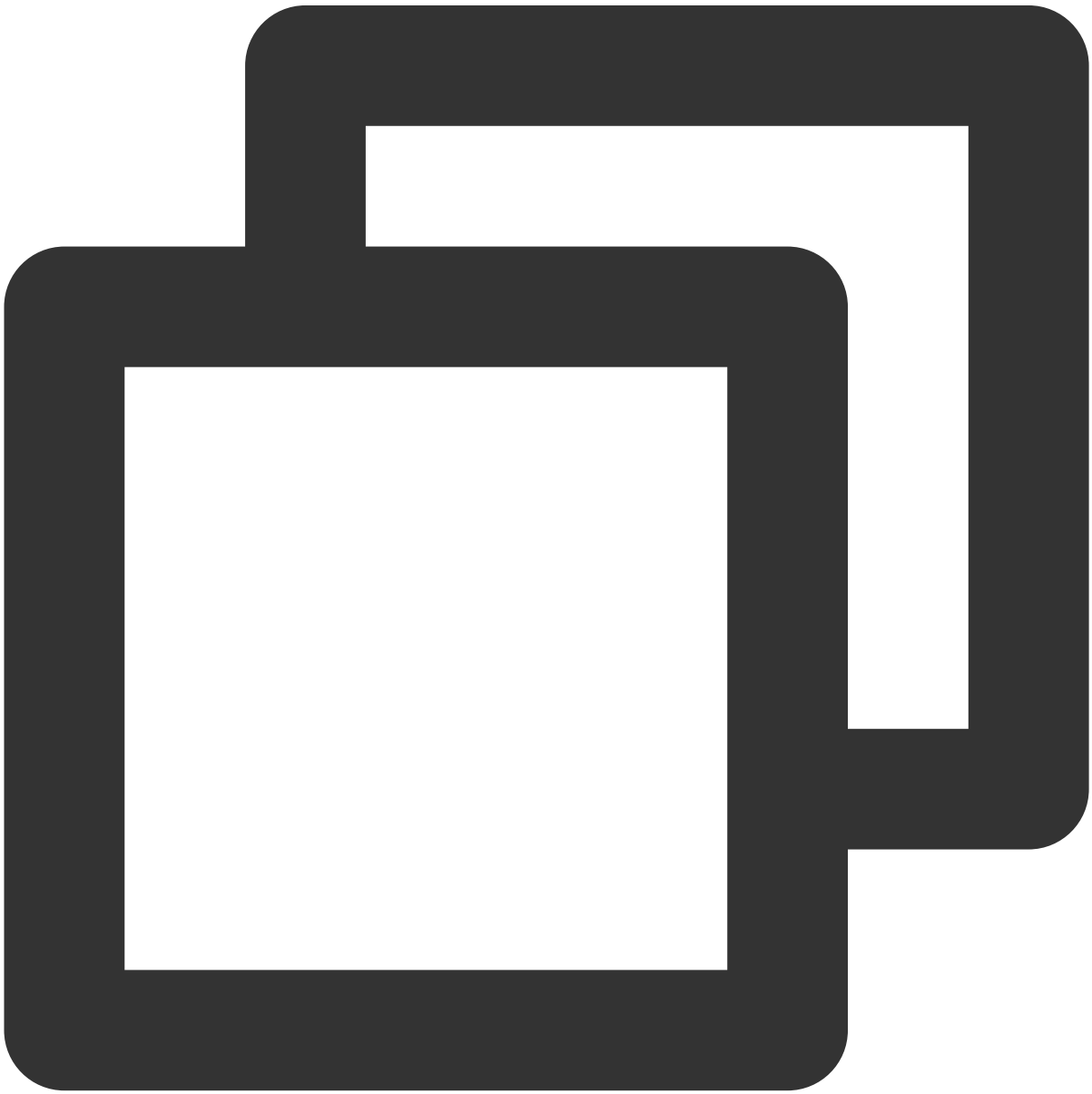


```
public class DemoImgSource extends TUIPlaySource {

    private String mImgUrl;

    public DemoImgSource(String imgUrl) {
        mImgUrl = imgUrl;
        // 你可以指定不同的viewType来区分自定义页面类型
        setExtViewType(SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE);
    }

    public String getImgUrl() {
```

```
        return mImgUrl;
    }

    public void setImgUrl(String imgUrl) {
        this.mImgUrl = imgUrl;
    }
}
```

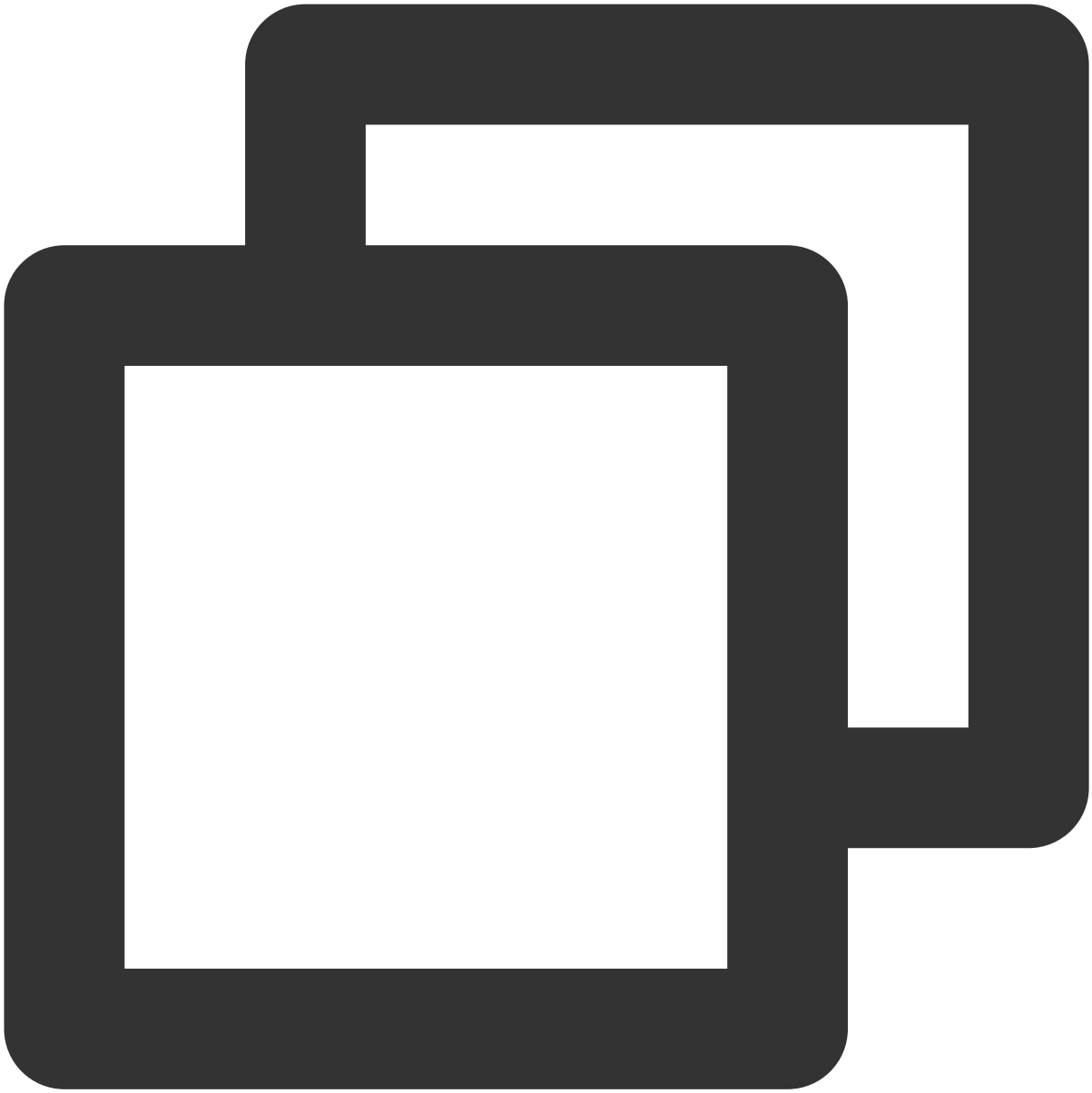
2. 实现自定义页面的 UI

数据实现好后，需要定制自己自定义页面的 UI，以展示图片为例，需要继承 `TUICustomLayer` 实现layer。



```
public class PicDisplayLayer extends TUICustomLayer {

    private ImageView mDisplayImgView;

    @Override
    public View createView(ViewGroup parent) {
        // 创建页面view
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.tuiplayer_img_display_layer, parent,
        mDisplayImgView = view.findViewById(R.id.iv_img_display);
        return view;
    }

    @Override
    public void onBindData(TUIPlaySource videoSource) {
        super.onBindData(videoSource);
        // 数据与页面发生绑定，可以拿到该页面对应的数据源
        if (videoSource.getExtViewType() == SVDemoConstants.CustomSourceType.SINGLE
            DemoImgSource source = (DemoImgSource) videoSource;
            Glide.with(mDisplayImgView).load(source.getImgUrl())
                    .into(mDisplayImgView);
        }
    }

    @Override
    public String tag() {
        return "PicDisplayLayer";
    }
}
```

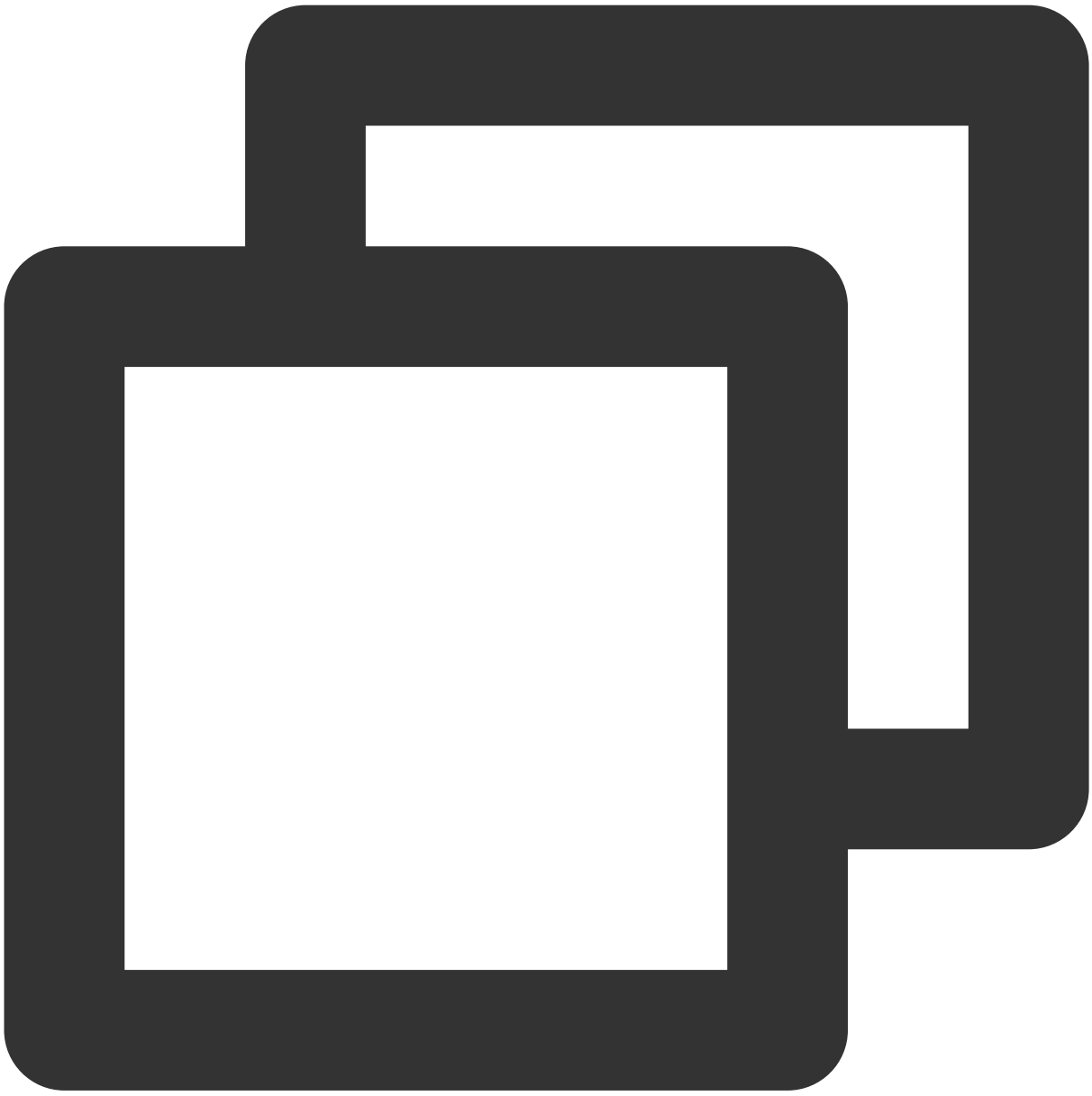
在 TUI 短视频回调中将自己的图层添加进去。



```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {

    // ......

  @Override
  public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
     // custom layer
      if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
          layerManager.addLayer(new PicDisplayLayer());
      }
  }
```

```
});
```

3. 填充数据到 TUI 短视频



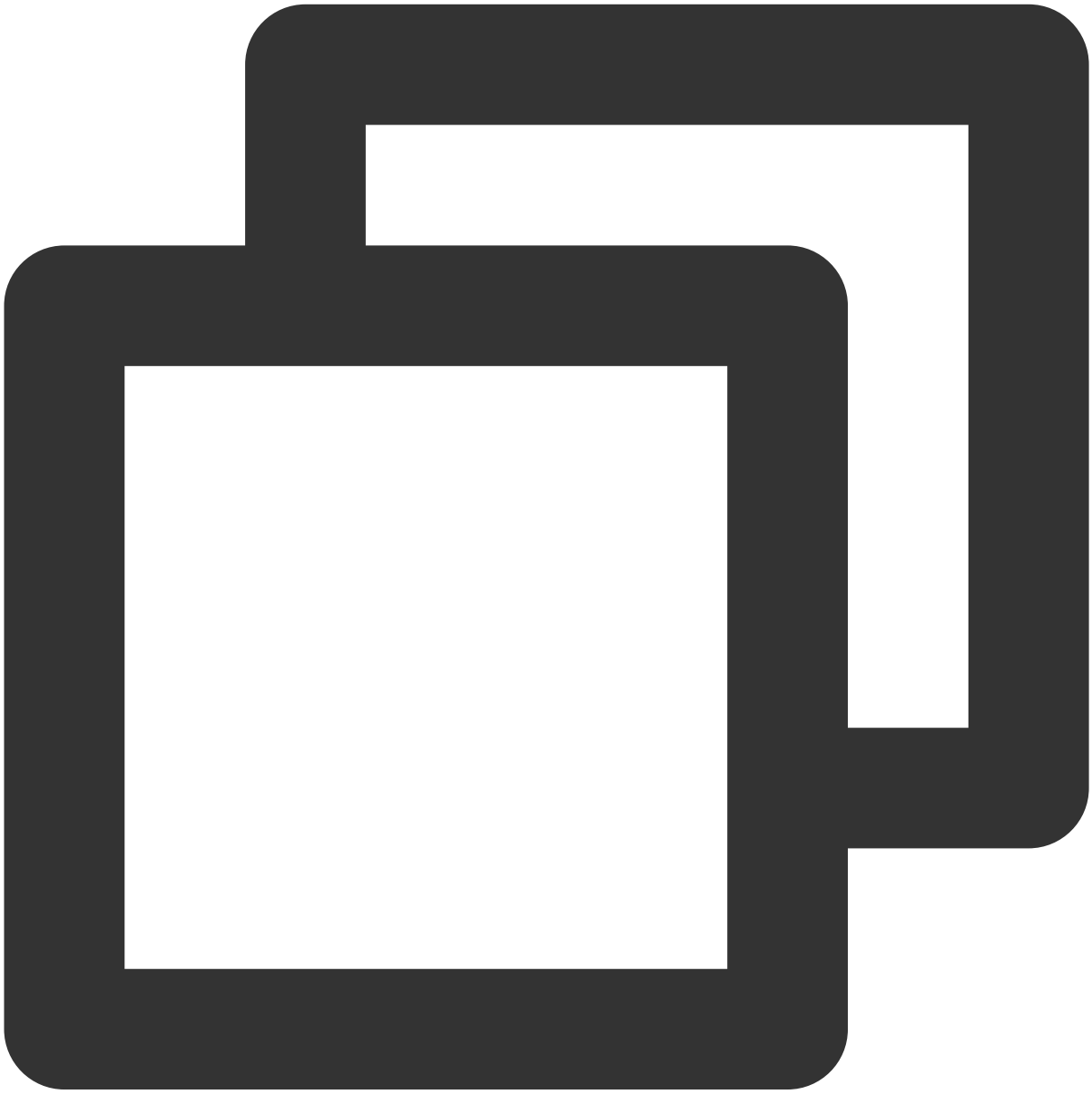
```
// 自定义，DemoImgSource继承自TUIPlaySource,自定义数据，此处可根据业务需求定制不同的数据
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// 设置数据
mSuperShortVideoView.setModels(shortVideoData);
```

随后，该自定义页面，将会根据您在列表中添加的位置，展示在短视频列表对应的页面位置上。

TUI 短视频接口

1. 配置 License

使用 TUI 组件，需要配置对应 premium 的 License，示例如下：

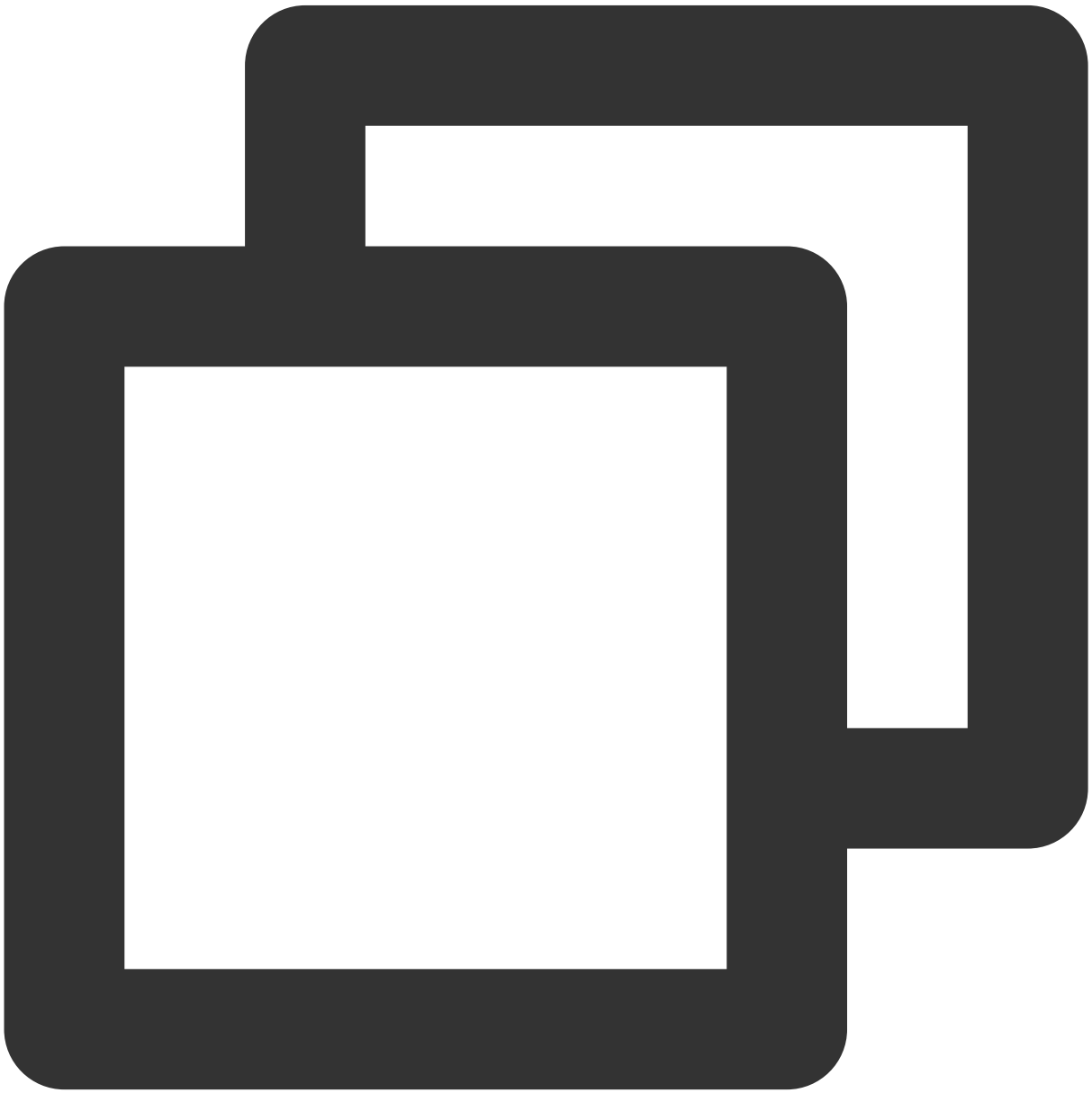


```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
```

```
        .licenseKey(LICENCE_KEY)
        .licenseUrl(LICENCE_URL)
        .build();
TUIPlayerCore.init(getApplicationContext(), config);
```

2. 设置生命周期监听

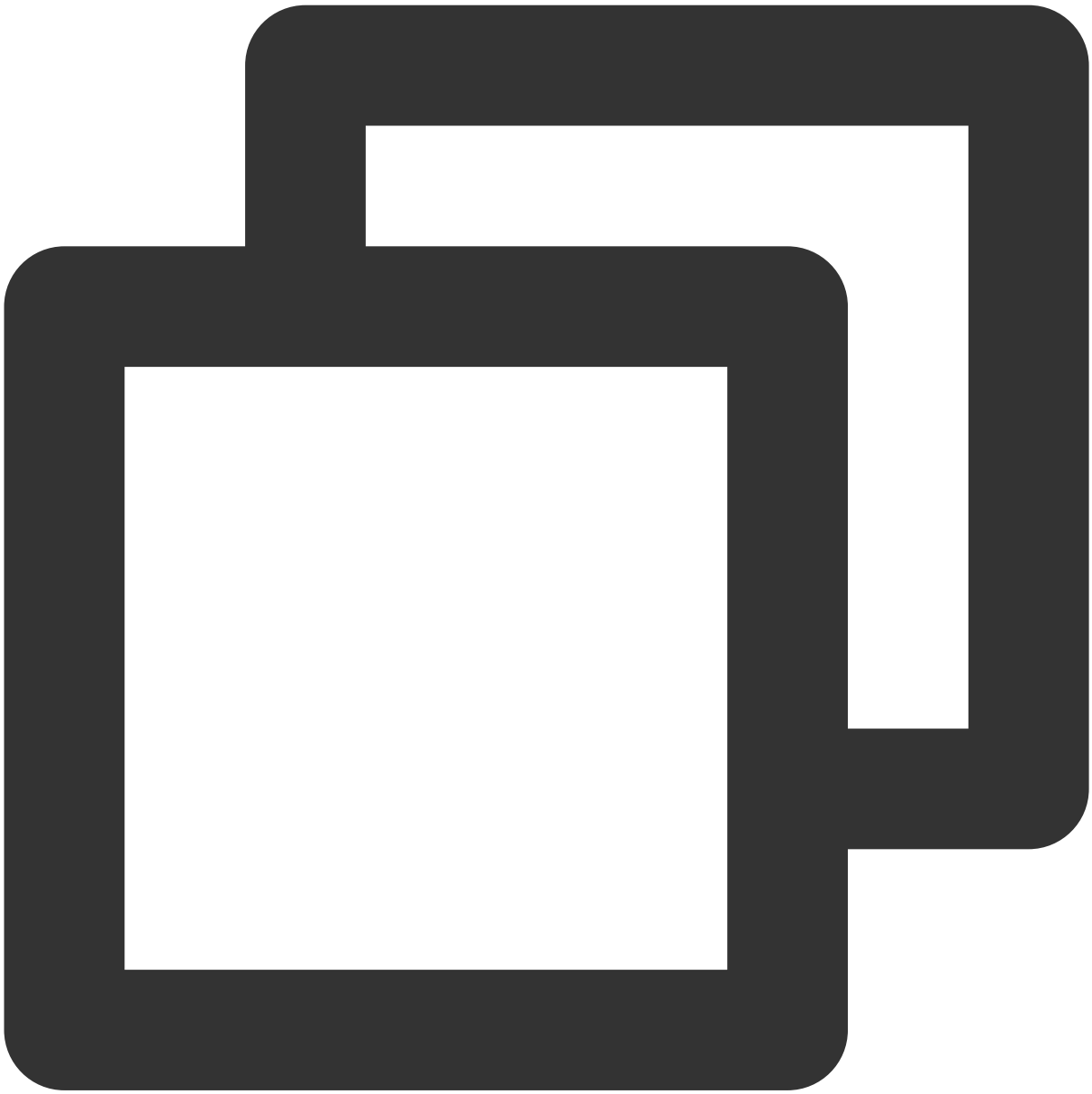
用于 TUIShortVideoView 的生命周期控制，内部自行根据 lifeCycle 状态，进行列表视频的暂停、播放和销毁，该接口可不设置，由业务来接管调用。



```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

3. 设置短视频监听

用于监听 TUIShortVideoView 的事件，其中包括加载分页数据时机，创建 page 的时候回调，可以在该回调添加图层。



```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
    @Override
    public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
        layerManger.addLayer(new TUIVideoInfoLayer(mShortVideoView, ShortVideoFragm
        layerManger.addLayer(new TUICoverLayer());
```

```
            layerManger.addLayer(new TUILoadingLayer());
            layerManger.addLayer(new TUIErrorLayer());
        }

        @Override
        public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
            layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView, ShortVideoF
            layerManager.addLayer(new TUILiveLoadingLayer());
            layerManager.addLayer(new TUILiveErrorLayer());
        }

        @Override
        public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewTyp
            if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
                layerManager.addLayer(new PicDisplayLayer());
            }
        }

        @Override
        public void onPageChanged(int index, TUIPlaySource videoSource) {
            if (index >= mShortVideoView.getCurrentDataCount() - 1) {
                mShortViewRefresh.setRefreshing(true);
                ShortVideoModel.getInstance().loadMore(false);
            }
        }

        @Override
        public void onNetStatus(TUIPlaySource model, Bundle bundle) {
        }
    });
```

4. 设置视频播放策略

设置视频播放过程中的各种策略。

策略 TUIPlayerVodStrategy 参数

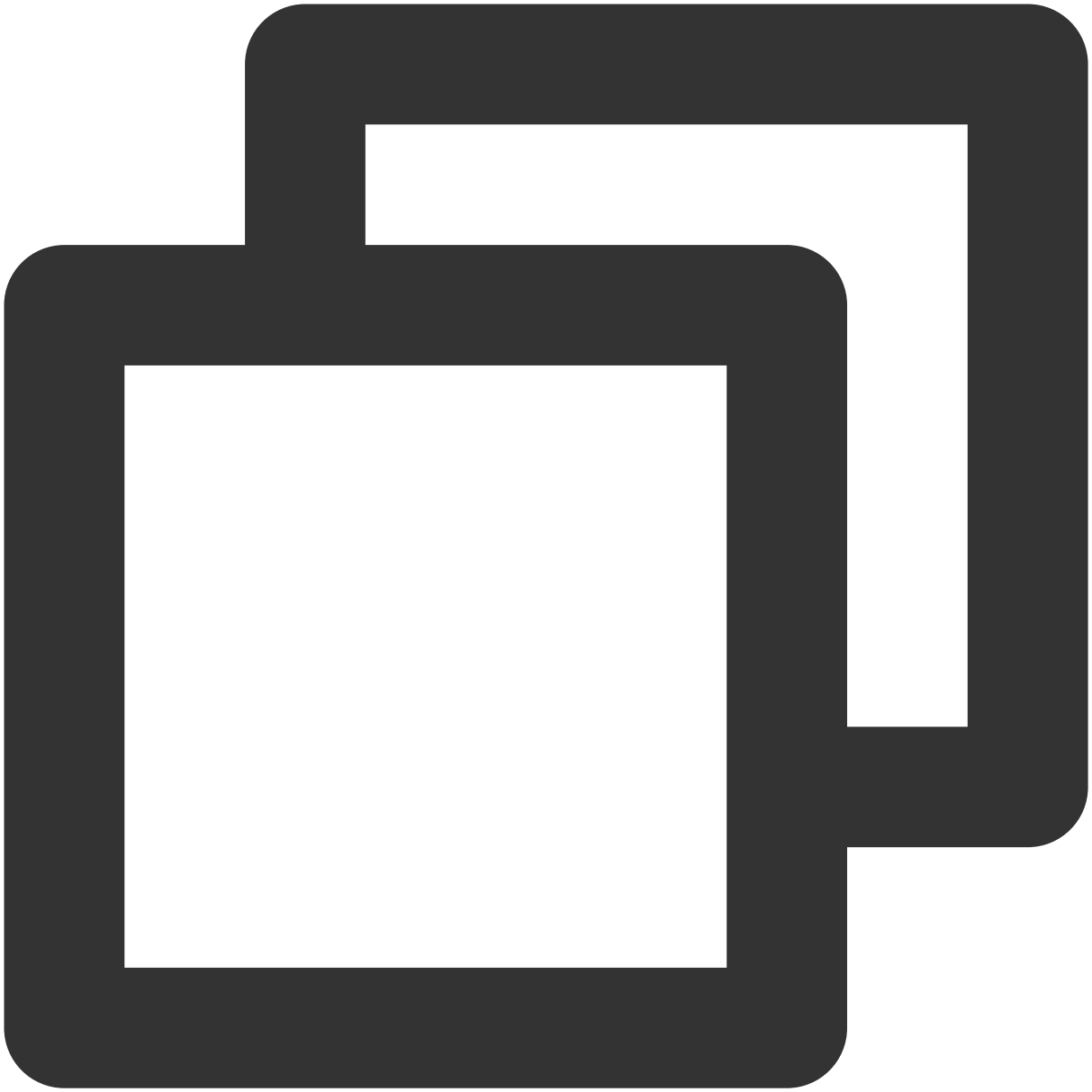
需要使用 Builder 构建。

函数	描述
setPreloadCount	设置预加载最大并发数量，默认3。
setPreDownloadSize	设置预下载缓存大小，默认1MB，单位 MB。
setPreLoadBufferSize	设置预播放缓存大小，默认0.5MB，单位 MB。
setMaxBufferSize	设置播放时视频缓存大小，默认10MB，单位 MB。

setPreferredResolution	设置视频播放的偏好分辨率，默认720 x 1280。
setProgressInterval	播放进度回调间隔，默认500毫秒，单位毫秒。
setRenderMode	渲染平铺模式，默认0。liteavPlayer 中，0代表全屏屏幕，1代表按照视频实际比例渲染，可能会有黑边。
setExtInfo	设置额外信息。
setMediaType	当提前知道播放的媒资类型时，可以通过该接口设置媒资类型，减少播放器 SDK 内部播放类型探测，提升启播速度。
enableAutoBitrate	设置是否启用码率自适应。
setResumeMode	设置续播模式，分为三种模式： TUIConstants.TUIResumeMode.NONE：不续播。 TUIConstants.TUIResumeMode.RESUME_LAST：续播最近一次播放。 TUIConstants.TUIResumeMode.RESUME_PLAYED：续播所有播放过的视频。
setDisplayViewFactory	设置自定义视频图层，可通过实现 IDisplayViewFactory 来自定义视频图层
setEnableAccurateSeek	设置是否开启精准 seek，开启精准 seek 之后，seek 的精准度会大幅提升，但是 seek 会有耗时，关闭之后，实际的seek 时间可能跟预期的有差距，差距根据视频的关键帧分布而定，但是 seek 耗时会变短。
setAudioNormalization	设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。注意：播放器高级版 11.7 版本开始支持。 可填预设值，相关常量类 TXVodConstants， 关：AUDIO_NORMALIZATION_OFF 开： AUDIO_NORMALIZATION_STANDARD（标准） AUDIO_NORMALIZATION_LOW （低） AUDIO_NORMALIZATION_HIGH（高）
setIsRetainPreVod	是否保留上一个播放器，以加快上一个播放器的起播速度

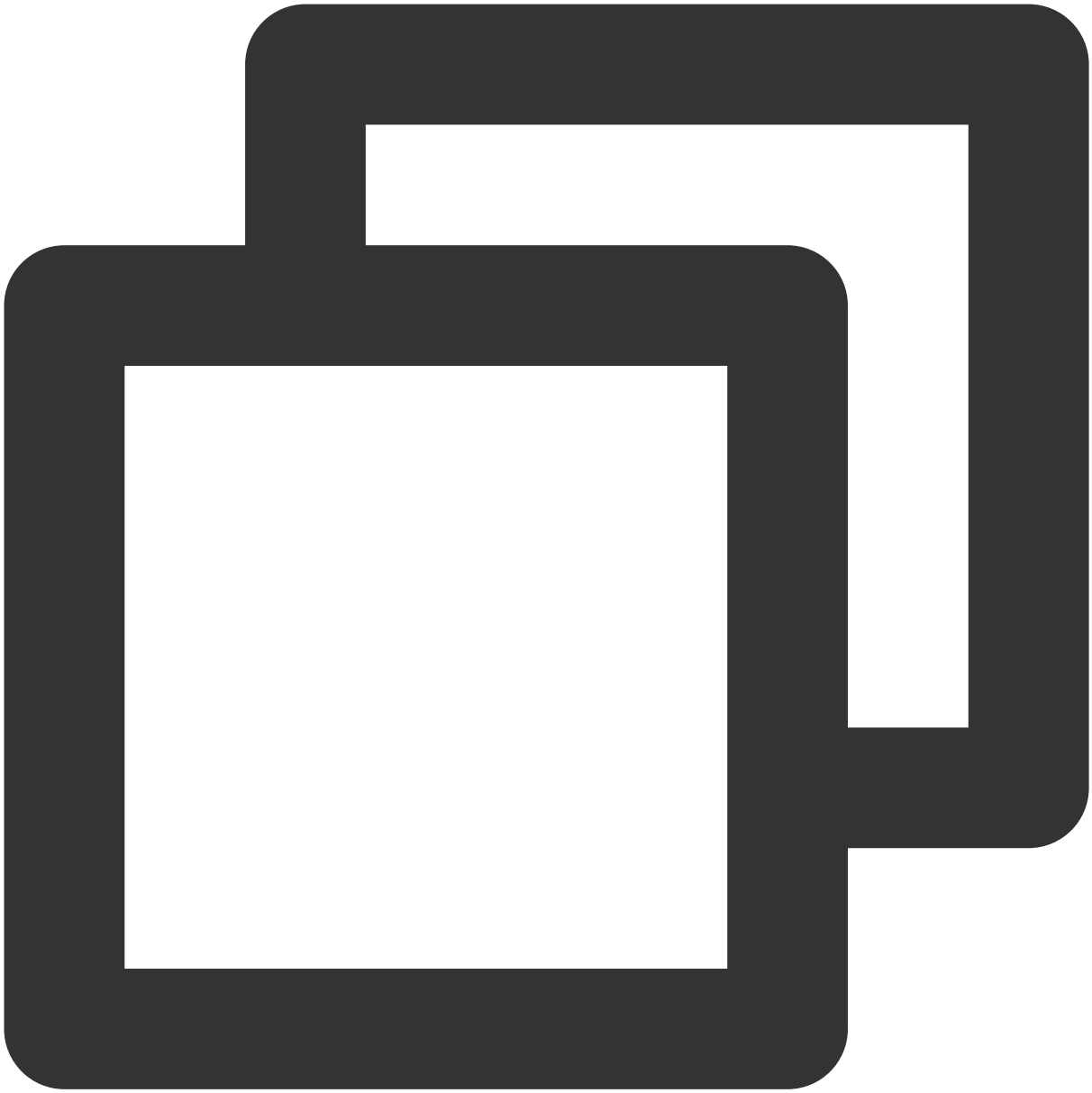
5. 填充数据

往 TUIShortVideoView 中填充数据：



```
mSuperShortVideoView.setModels(shortVideoBeanList);
```

追加数据：



```
mSuperShortVideoView.appendModels(shortVideoBeanList);
```

TUIVideoSource 类

参数/函数	参数类型	描述
setVideoURL	String	视频链接，建议填充该字段，会加快预加载速度。
setCoverPictureUrl	String	视频封面，会回调到 layer，由客户自行处理。

setAppId	int	视频 appId。
setFileId	String	视频 fileId。
setPSign	String	视频加密 pSign。
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。
setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的 layer 进行消息通知。该方法必须通过 TUIDataManager 获取到之后的 Source 调用才会有效。
setVideoConfig	TUIPlayerVideoConfig	视频独立配置。
setExternalSubtitle	List<TUISubtitleSource>	设置外挂字幕，会自行加载到点播播放器中，必须要播放器高级版支持。
setAutoPlay	boolean	设置该视频是否自动播放

TUIPlayerVideoConfig 类

参数	类型	描述
setPreloadBufferSizeInMB	float	设置视频单独的预播放缓存大小，可选。
setPreferredResolution	long	设置视频单独的起播和预加载分辨率，可选。
setPreDownloadSize	float	设置视频单独的预下载缓存大小，可选。

TUILiveSource 类

参数/函数	参数类型	描述
setUrl	String	直播链接
setCoverPictureUrl	String	封面，会回调到 layer，由客户自行处理。
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。
setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的 layer 进行消息通知。该方法必须通过

		TUIDataManager 获取到之后的 Source 调用才会有效。
setLiveConfig	TUIPlayerLiveConfig	独立配置。
setAutoPlay	boolean	设置是否自动播放，直播设置改字段为 false 时，一开始会没有画面，需要使用封面图遮盖

TUIPlayerLiveConfig 类

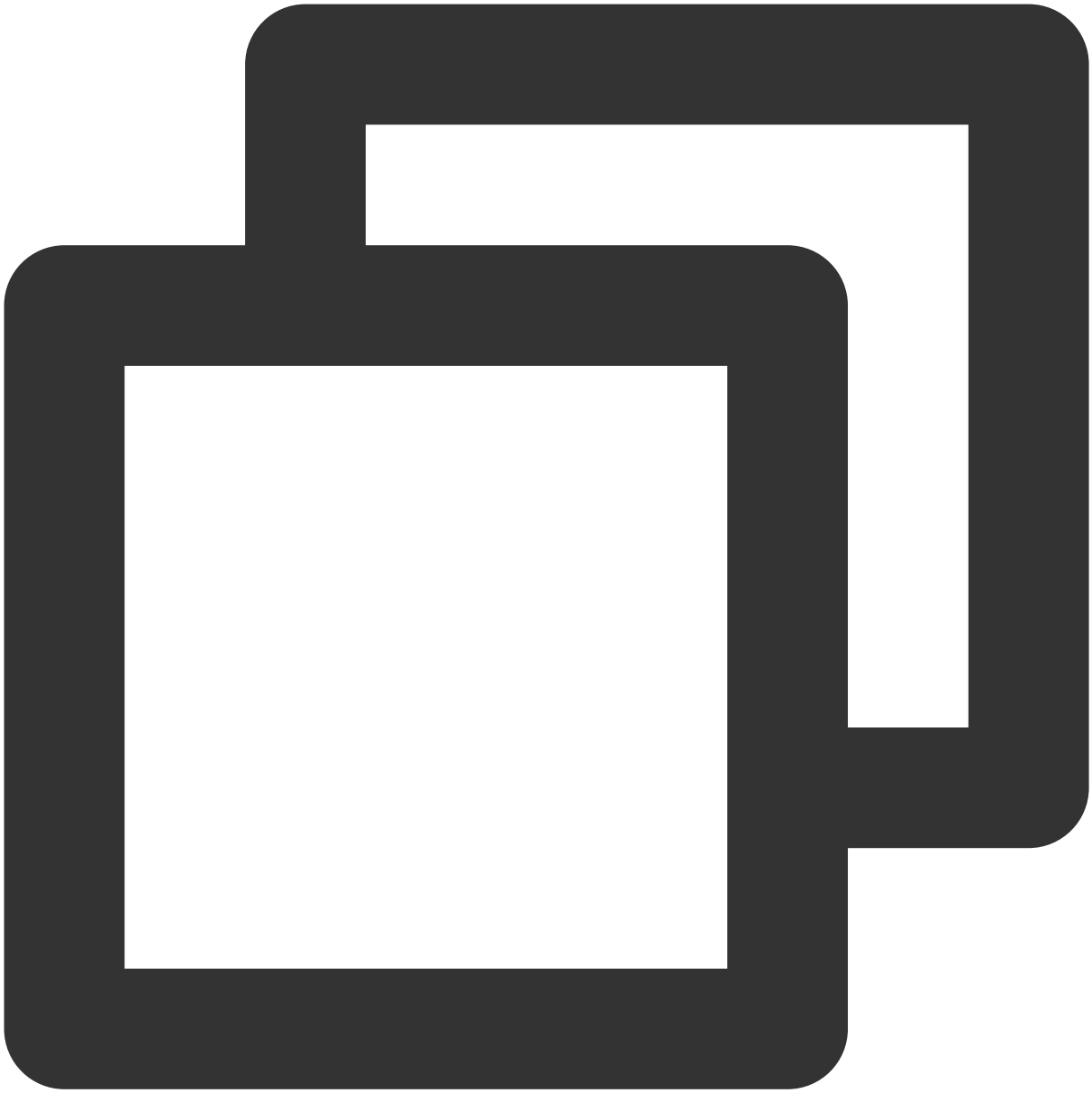
参数	类型	描述
setCacheMinTime	float	缓存自动调整的最小时间，取值需要大于0。【默认值】：1。
setCacheMaxTime	float	缓存自动调整的最大时间，取值需要大于0。【默认值】：5

TUIPlaySource 类

参数/函数	参数类型	描述
setExtViewType	int	自定义页面类型，该值会通过 Layer 创建回调的第二个参数回调出来，供业务区分不同类型的页面。
setExtInfoAndNotify	Object	用于业务自行扩展额外参数，使用该方法可以实时向界面已存在的 layer 进行消息通知。该方法必须通过 TUIDataManager 获取到之后的 Source 调用才会有效。

6. 操作列表数据

TUI短视频提供了数据操作接口，可以实时修改列表内已经添加的数据，通过 `TUIShortVideoView` 的 `getDataManager()` 方法获取数据操作对象，如下所示：



```
// 获取数据操作对象
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
```

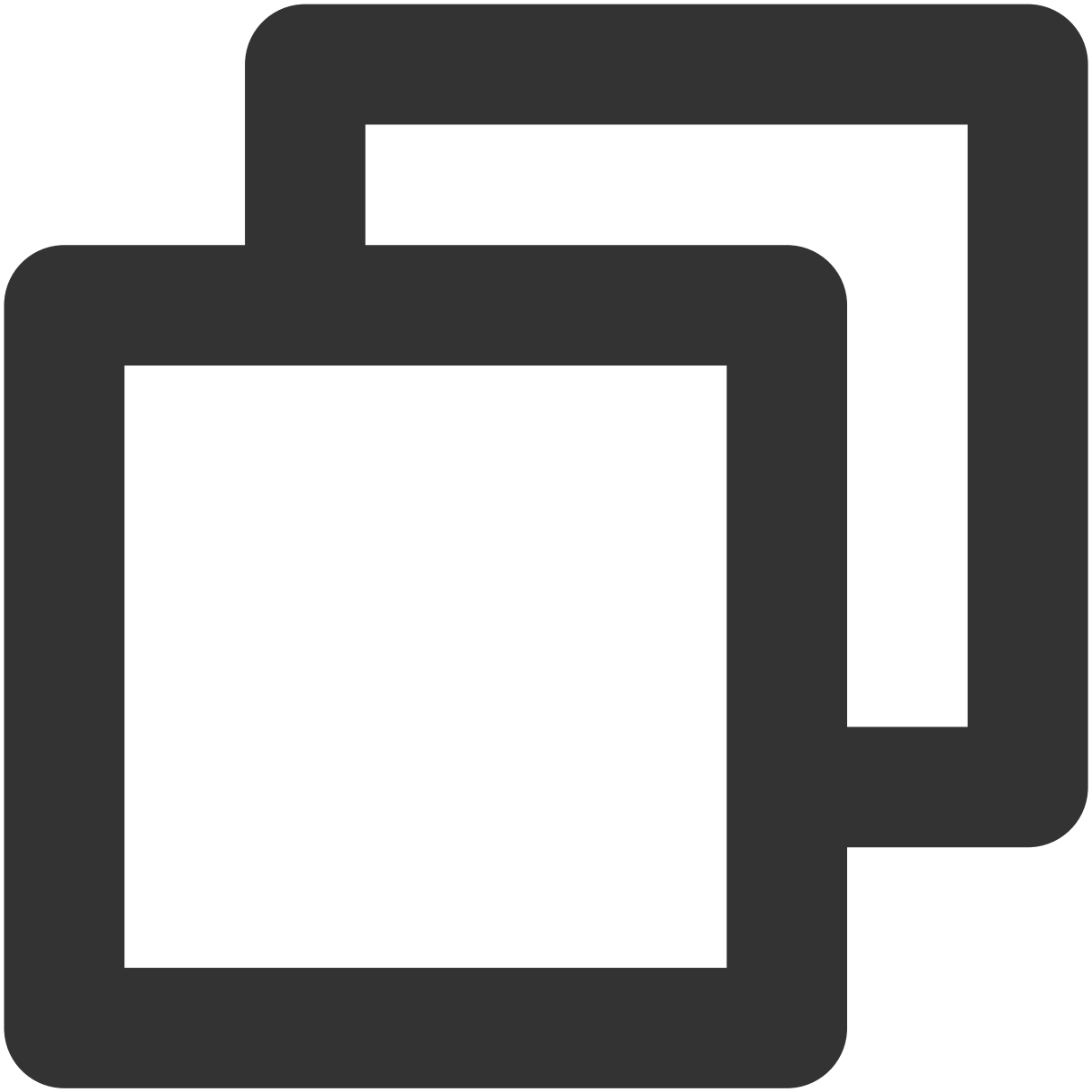
获得数据操作对象之后，可以实时对列表数据进行操作，接口如下：

函数名称	返回参数	传入参数	描述
removeData	void	index：需要移除的页面位置	移除对应的页面和数据
removeRangeData	void	index：需要移除页面的起始位置	移除一个范围的页面和数据

		count：从起始位置开始要移除的数量，不包括 count 的最后一位	
removeDataByIndex	void	removeIndexList：需要移除的页面的位置集合	按照传入的索引，移除索引集合内所有的页面和数据
addData	void	source：需要插入的数据 index：插入数据的位置	根据插入数据的位置，将传入的数据插入到指定位置
addRangeData	void	sources：需要插入的数据集合 startIndex：插入数据的起始位置	根据传入的起始位置，将数据集合插入到指定位置
replaceData	void	source：需要替换的数据 index：需要被替换的位置	根据传入的位置，把指定位置的数据替换为传入的数据
replaceRangeData	void	sources：需要替换的数据集合 startIndex：替换的起始位置	根据传入的起始位置，将传入的数据集合替换到指定位置
getDataByPageIndex	TUIVideoSource	index：页面位置	根据传入的位置，获得指定位置的页面数据
getCurrentDataCount	int	-	获得当前列表所有数据的总数
getCurrentIndex	int	-	获得当前正在显示页面的位置
getCurrentModel	TUIVideoSource	-	获得当前正在显示页面的数据

7. 获取当前正在播放的视频资源

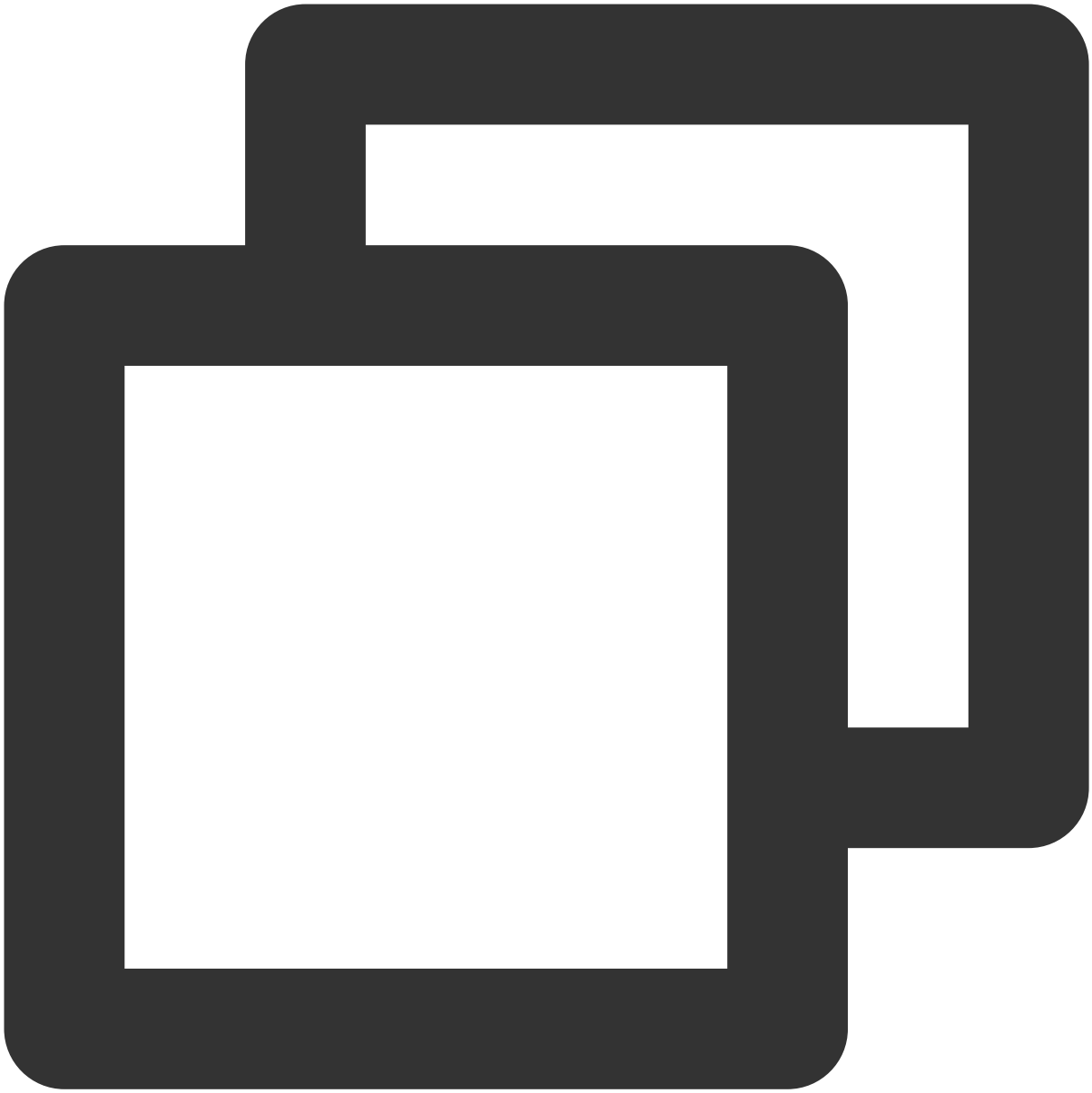
获取当前正在播放的视频资源，使用参见如下代码：



```
mSuperShortVideoView.getCurrentModel()
```

8. 暂停

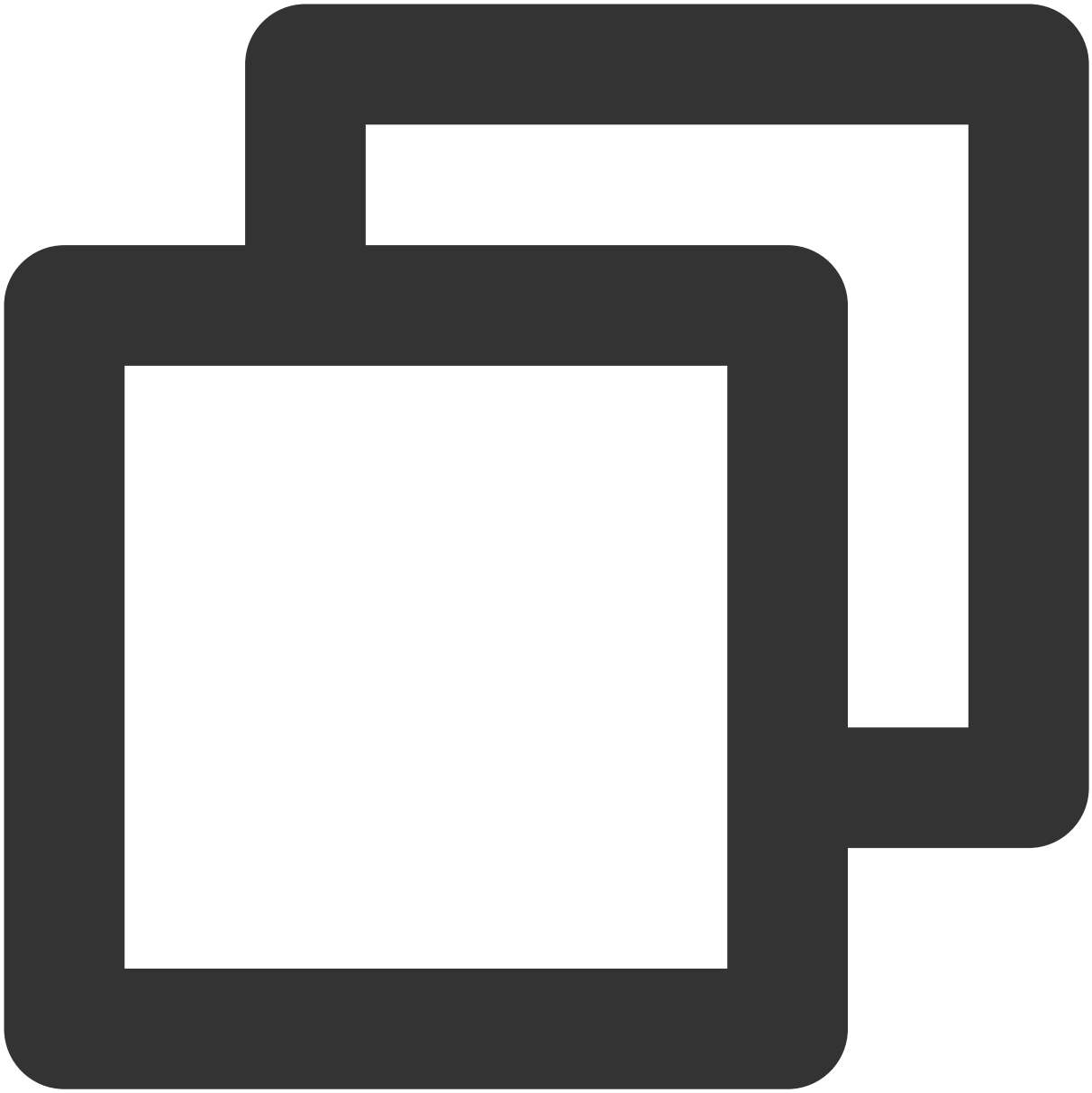
暂停当前正在播放视频。



```
mSuperShortVideoView.pause()
```

9. 从指定位置播放

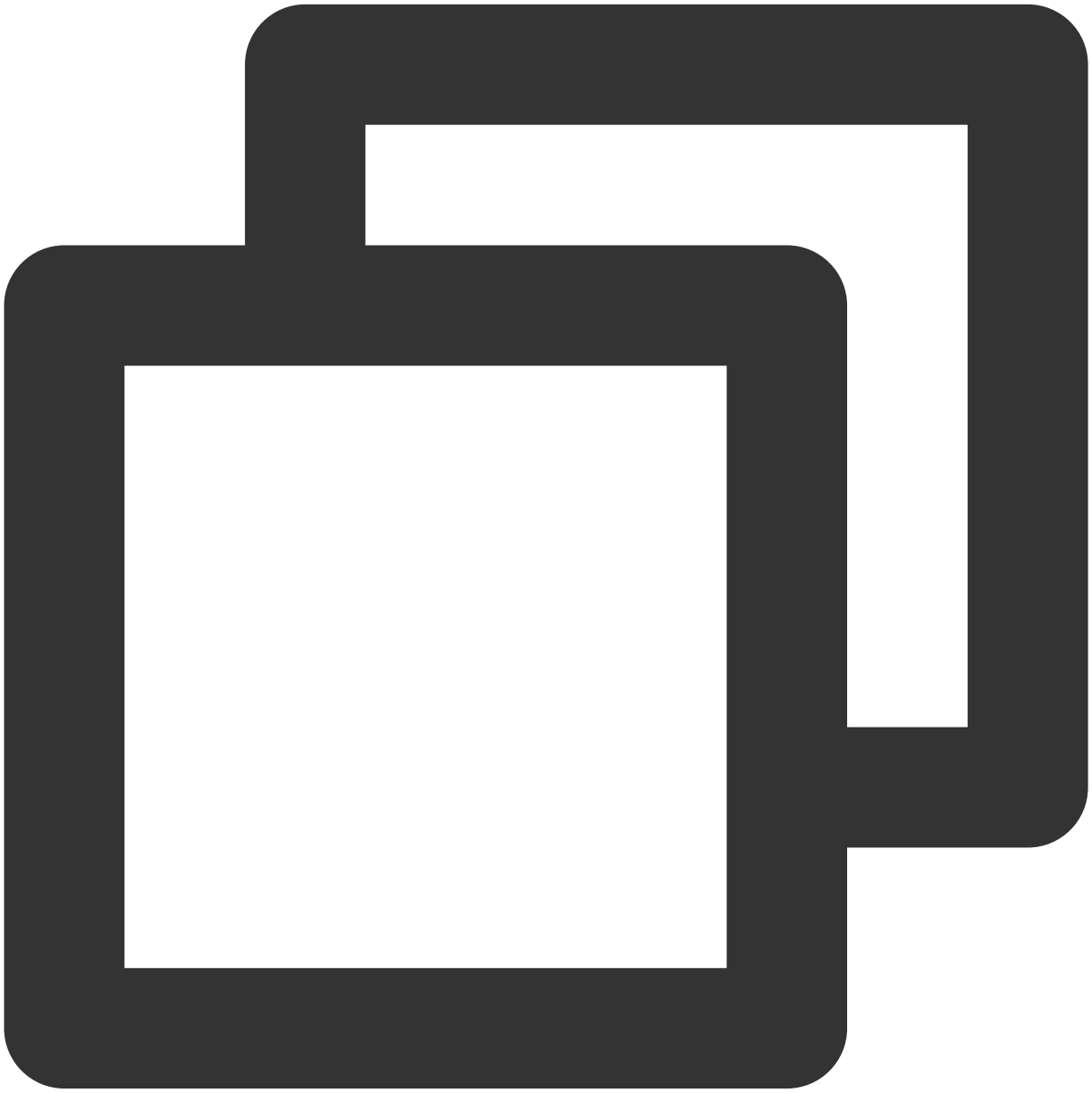
从指定位置开始播放，该方法可以在播放过程中直接跳转到指定位置，默认不平滑跳转，使用参见如下代码：



```
// index 为指定的页面位置
mSuperShortVideoView.startPlayIndex(index);


// index 为需要前往的位置，true 为需要平滑切换，默认为false
mSuperShortVideoView.startPlayIndex(index, true);
```

10. 设置短视频播放模式

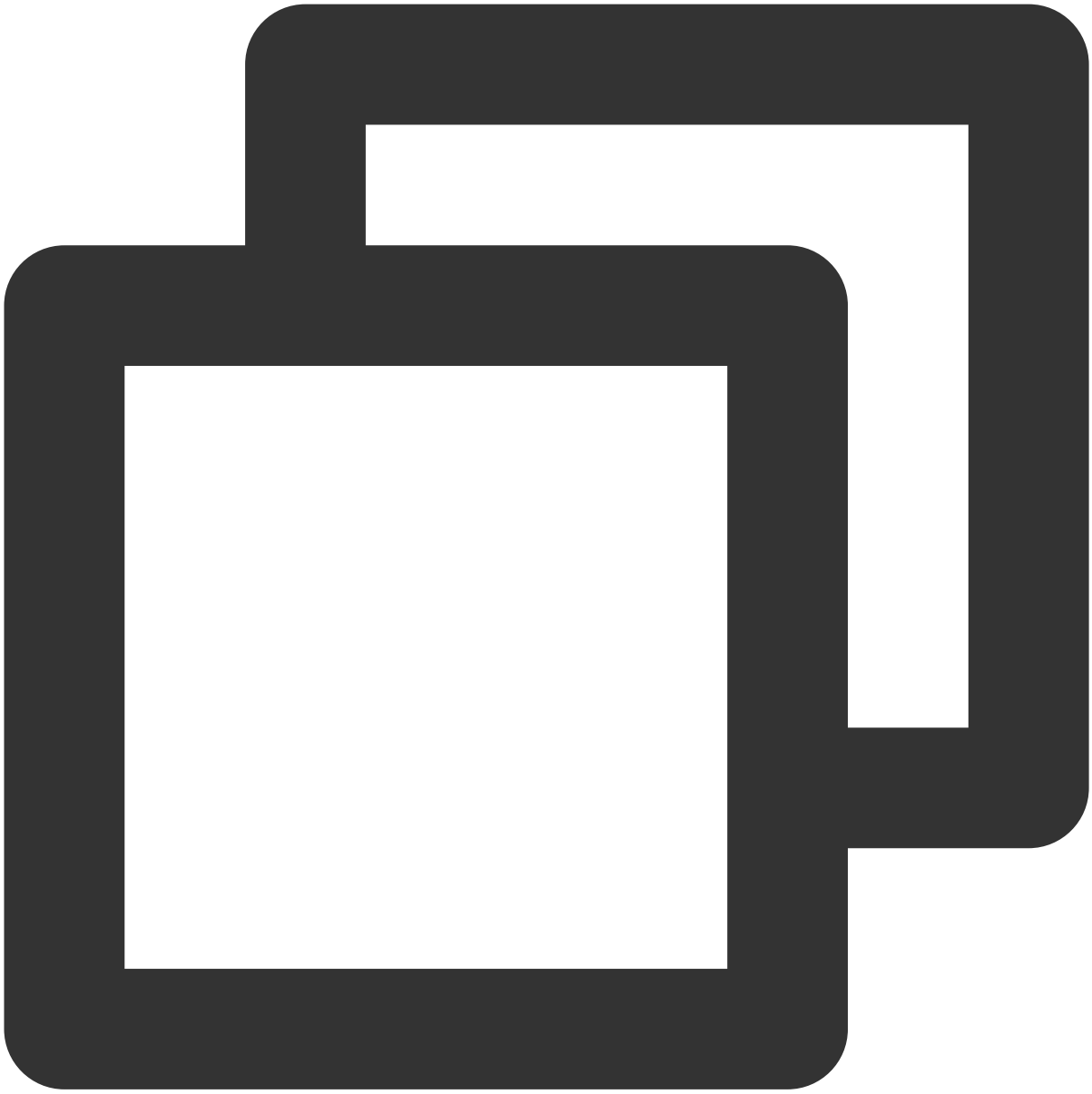
目前短视频播放模式有两种，分别为列表循环播放 `TUIVideoConst.ListPlayMode.MODE_LIST_LOOP` ，当前视频播放完毕之后会自动播放下一个，播放到最后一个之后，会自动回到首个视频继续播放。以及单个视频循环播放 `TUIVideoConst.ListPlayMode.MODE_ONE_LOOP` ，会一直重复播放当前视频直到用户手动去滑动翻页，也可以设置为 `TUIVideoConst.ListPlayMode.MODE_CUSTOM` 由业务接管播放逻辑。当不设置的时候，默认为 `MODE_ONE_LOOP` ，设置方式如下：



```
// set to MODE_ONE_LOOP
mSuperShortVideoView.setPlayMode(TUIVideoConst.ListPlayMode.MODE_ONE_LOOP);
```

11. 销毁控件

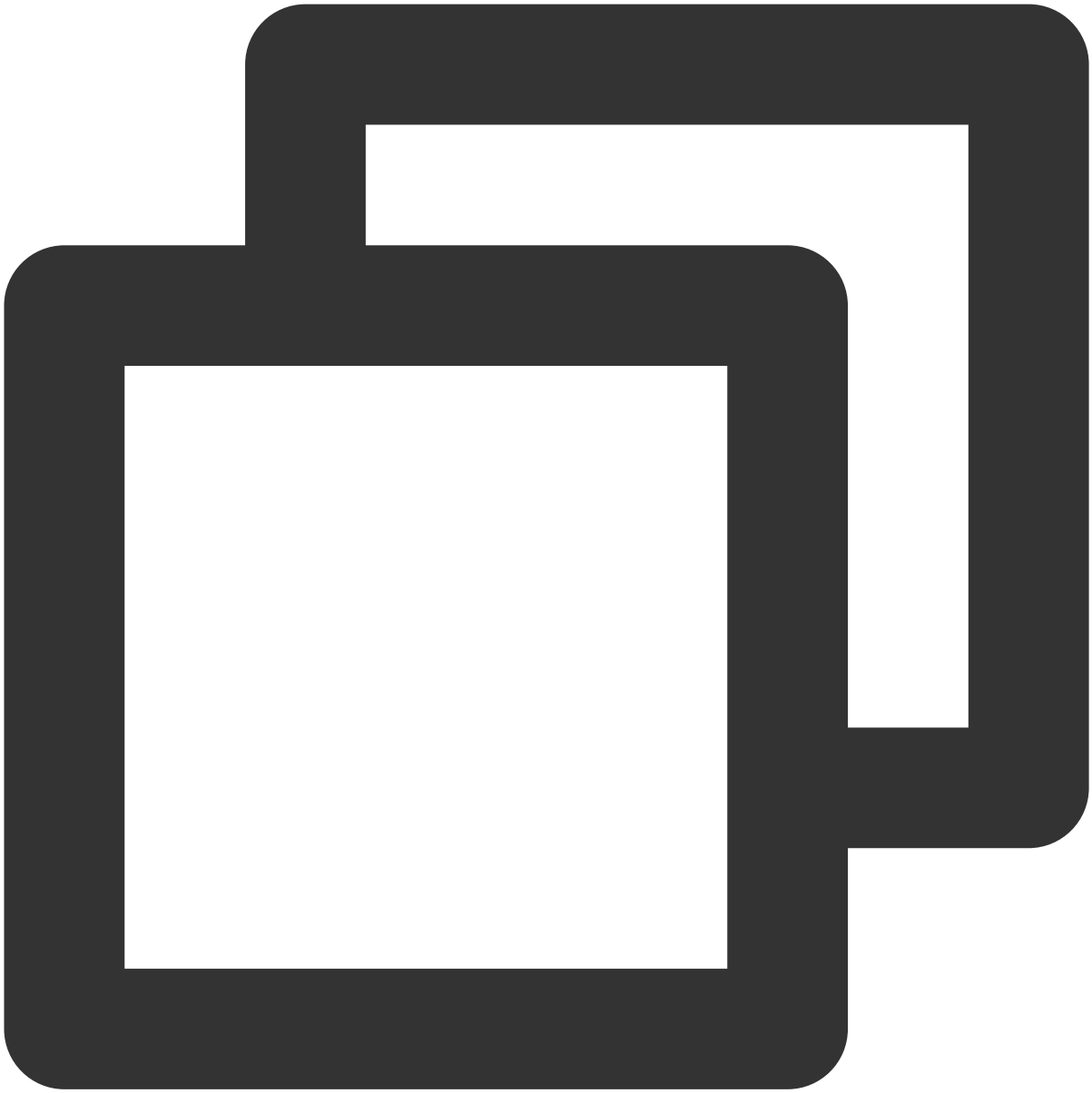
销毁控件和资源。



```
mSuperShortVideoView.release()
```

12. 续播当前视频

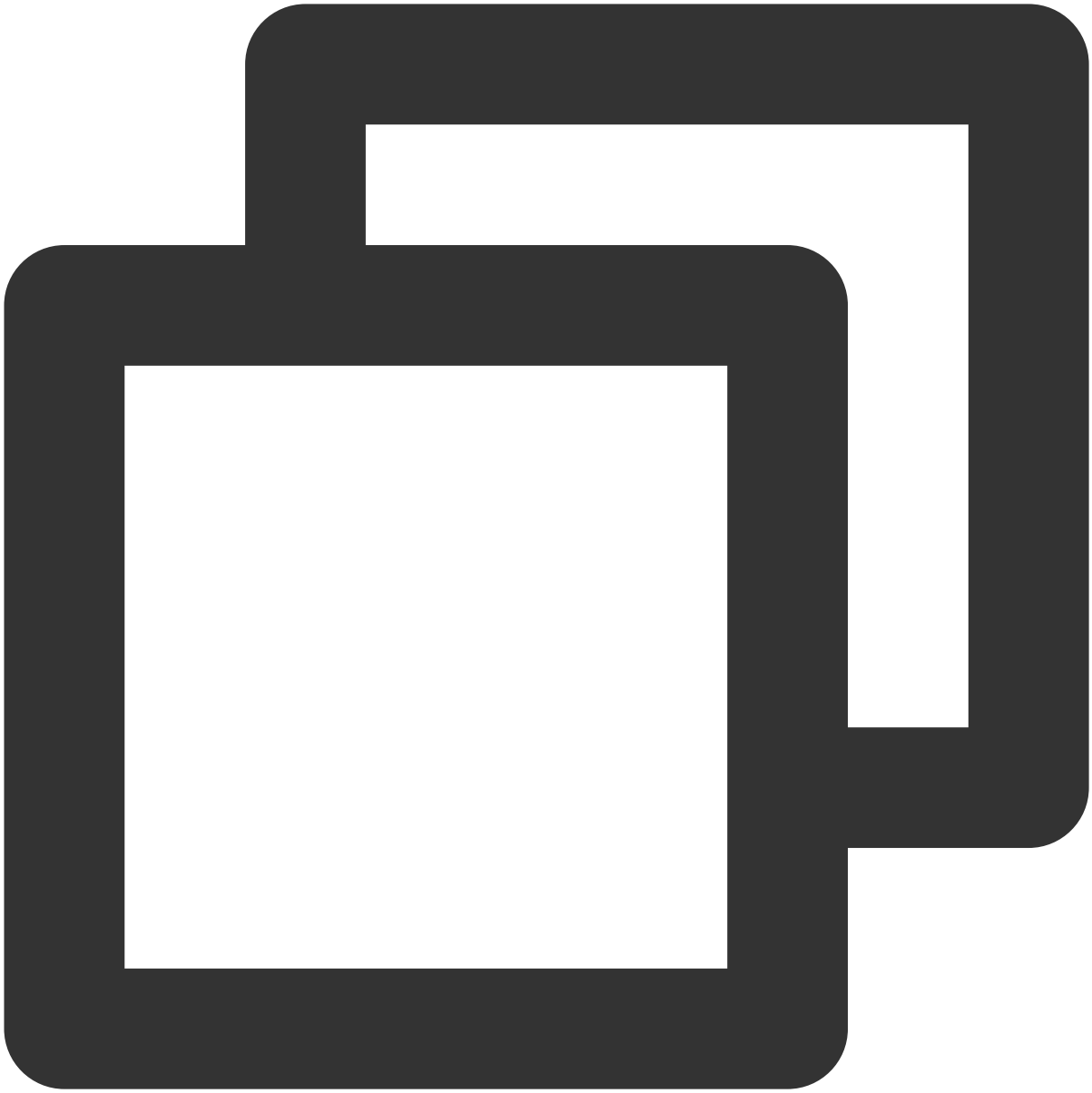
续播当前视频，使用参见如下代码：



```
mSuperShortVideoView.resume()
```

13. 实时切换分辨率

TUI 短视频可以实时切换当前视频分辨率以及全局视频分辨率，接口如下：



```
mSuperShortVideoView.switchResolution(720 * 1080, TUIConstants.TUIResolutionType.CU
```

switchType 除了可以传递 `TUIResolutionType` 以外，也可以直接指定需要切换的视频的 index 来切换视频分辨率。

switchType 含义如下：

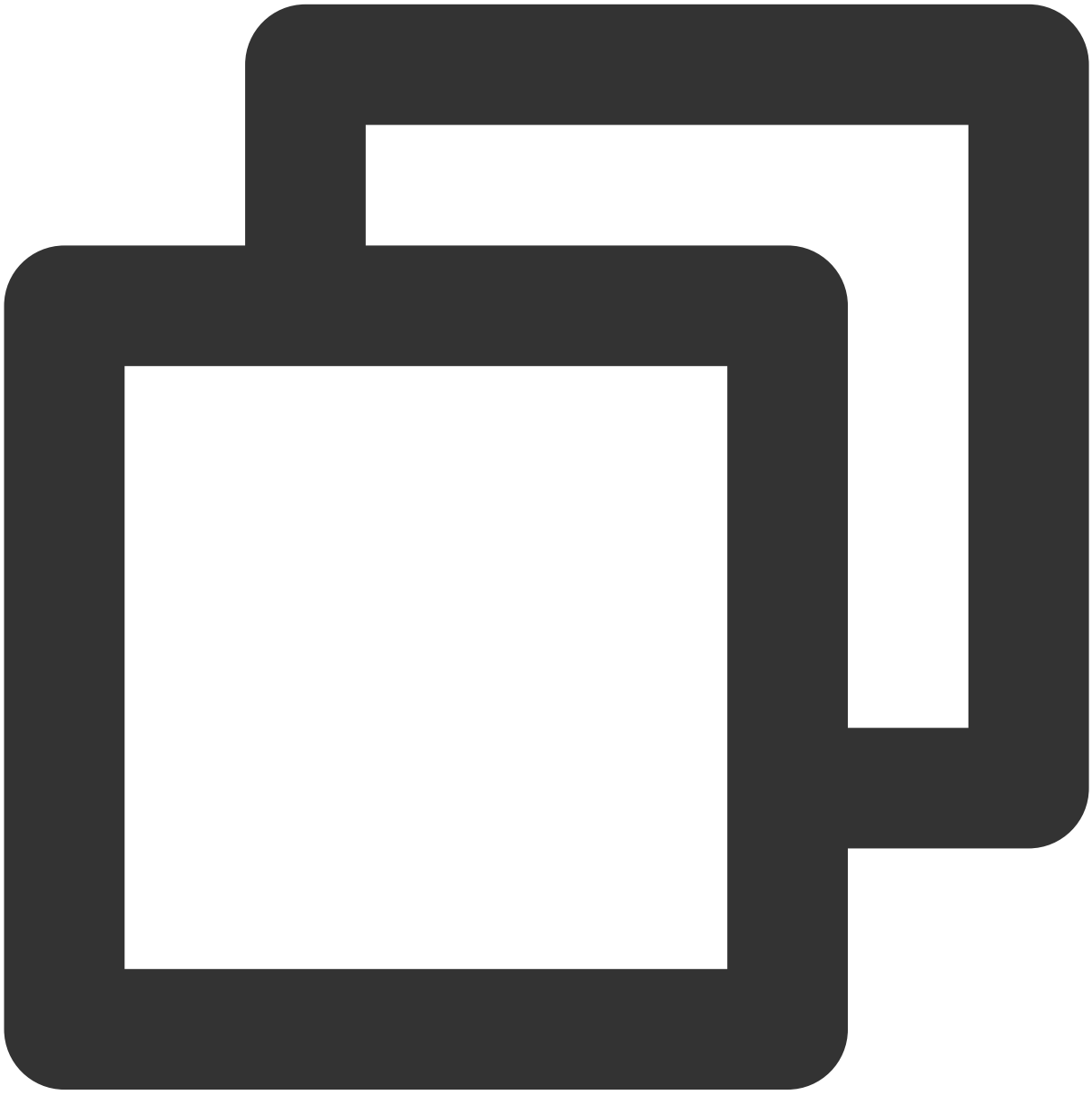
参数	描述
GLOBAL	设置全局分辨率
CURRENT	设置当前视频分辨率

其他大于等于0的值	设置指定位置的分辨率

目前分辨率的优先级，当前视频分辨率的设置优先级大于全局分辨率。

14. 暂停和继续预加载

TUI 短视频可以实时暂停和继续预加载任务



```
// pause all preload
mSuperShortVideoView.pausePreload();
```

```
// start preload from current video index
mSuperShortVideoView.resumePreload();
```

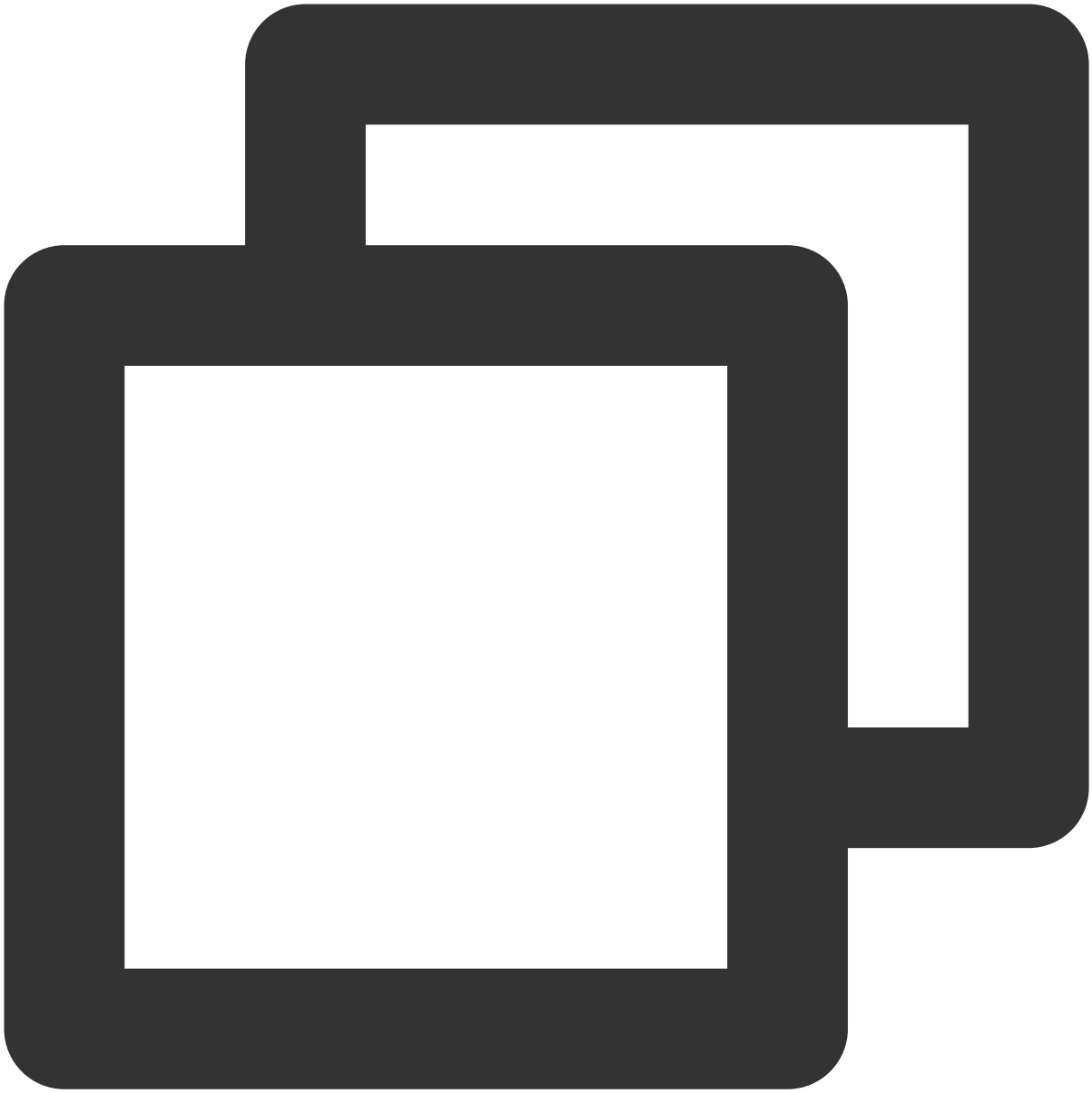
调用继续预加载的时候，会从当前视频开始往后继续预加载。

15. 添加图层

TUI 短视频可以通过添加图层来实现短视频播放界面上的自定义 UI。当 onCreateItemLayer 回调时，就可以通过方法携带的 LayerManager 进行图层的添加和管理。可以通过继承 TUIBaseLayer 来自定义自己需要的图层。图层会浮在视频 VideoView 的上方。
图层的显示和隐藏，都是通过对 View 的添加和移除来操作的，不会产生过度渲染的问题。

onCreateVodLayer、onCreateLiveLayer 和 onCreateCustomLayer 参数

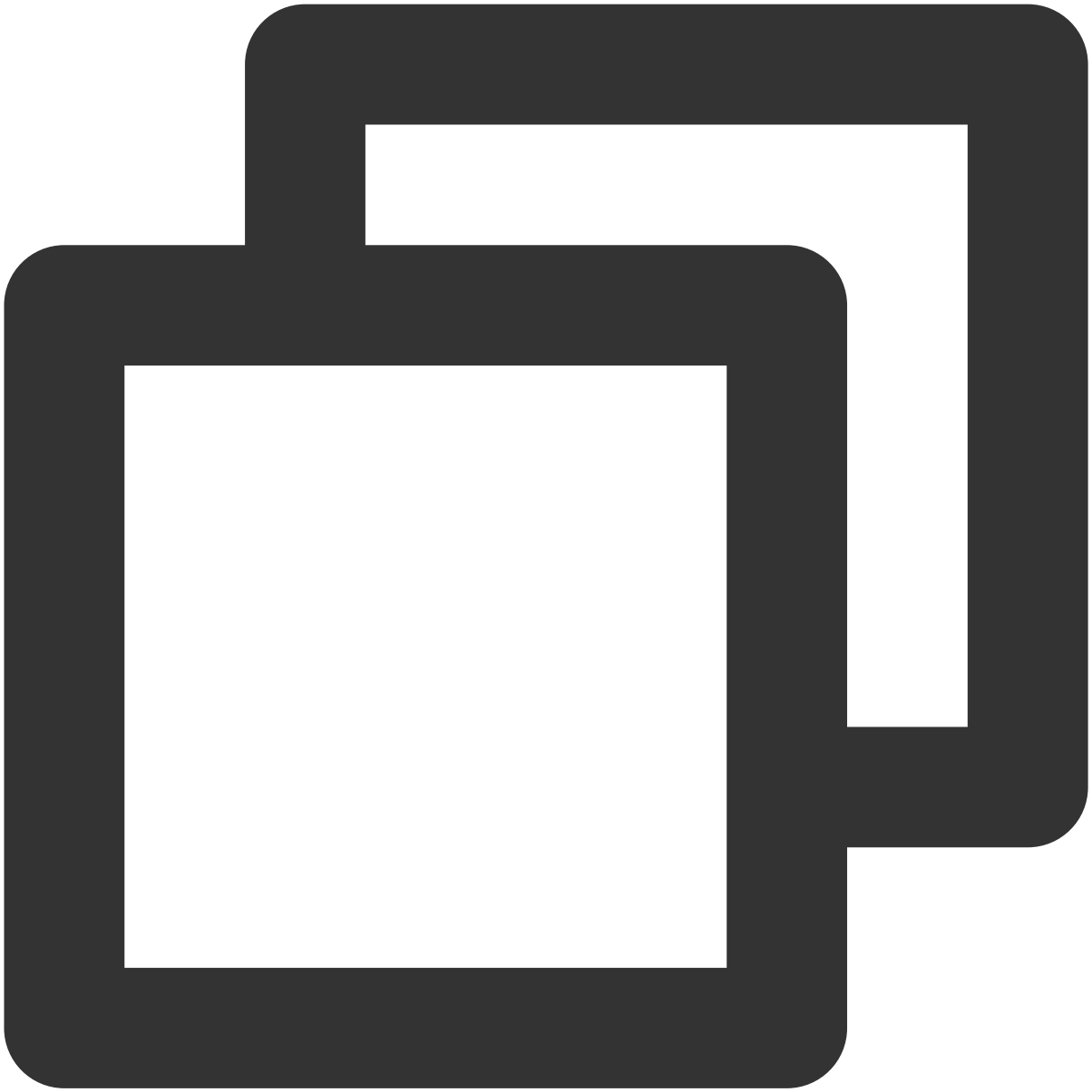
参数	类型	描述
layerManger	TUIVodLayerManager、TUILiveLayerManager 和 TUICustomLayerManager	图层管理对象
viewType	int	当前视频播放类型 TUIVideoConst.ItemType.ITEM_TYPE_VOD：点播 TUIVideoConst.ItemType.ITEM_TYPE_LIVE：直播 TUIVideoConst.ItemType.ITEM_TYPE_CUSTOM：自定义界面 其他由PlayerSource传入的自定义类型



```
layerManger.addLayer(new TUICoverLayer());
layerManger.addLayer(new TUIVideoInfoLayer());
layerManger.addLayer(new TUILoadingLayer());
layerManger.addLayer(new TUIErrorLayer());
```

16. 移除指定图层

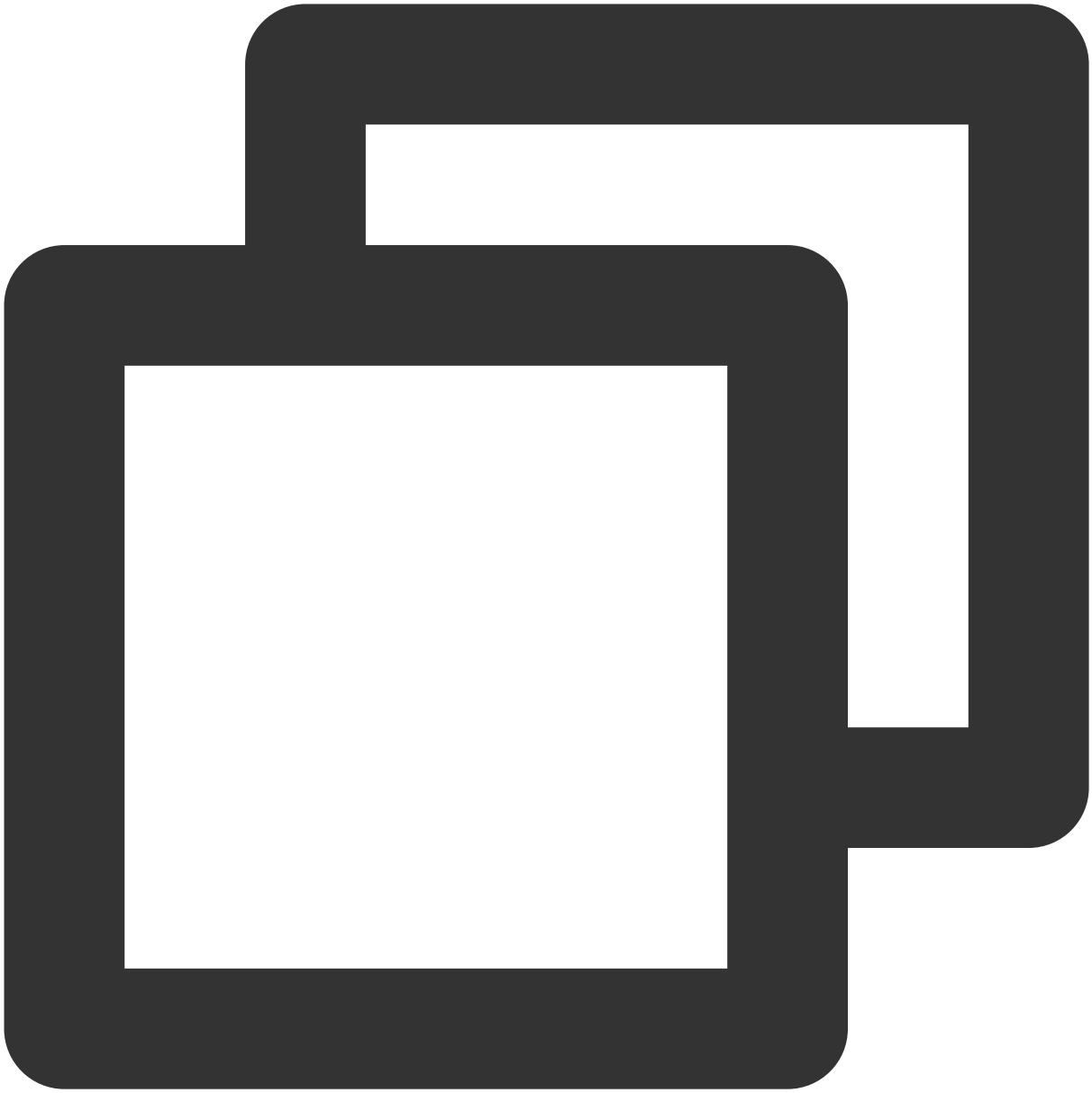
将图层管理器中的图层移除。



```
layerManger.removeLayer(layer);
```

17. 移除所有图层

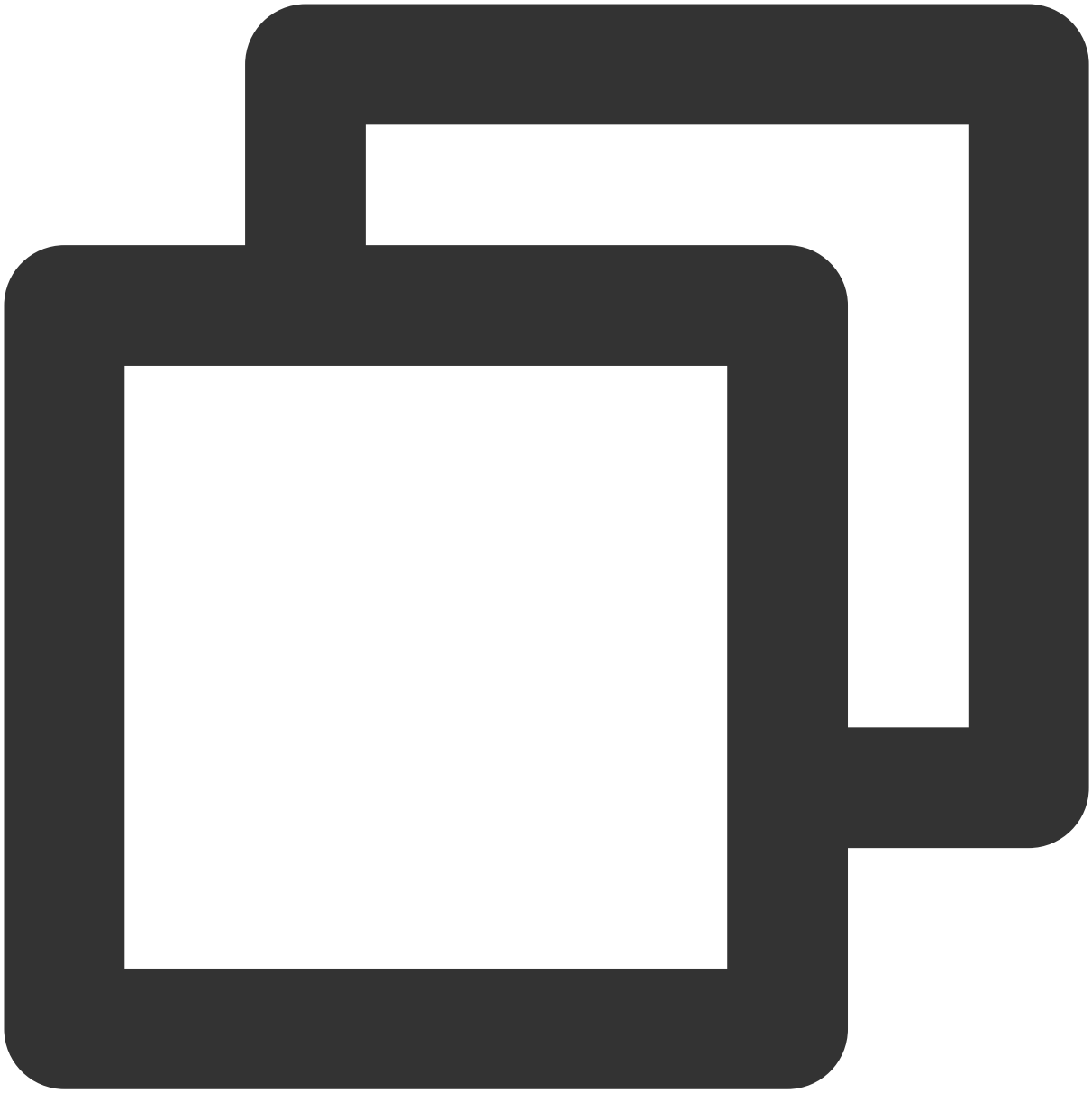
移除图层管理器中的所有图层。



```
layerManger.removeAllLayer();
```

18. 获得图层的当前层级

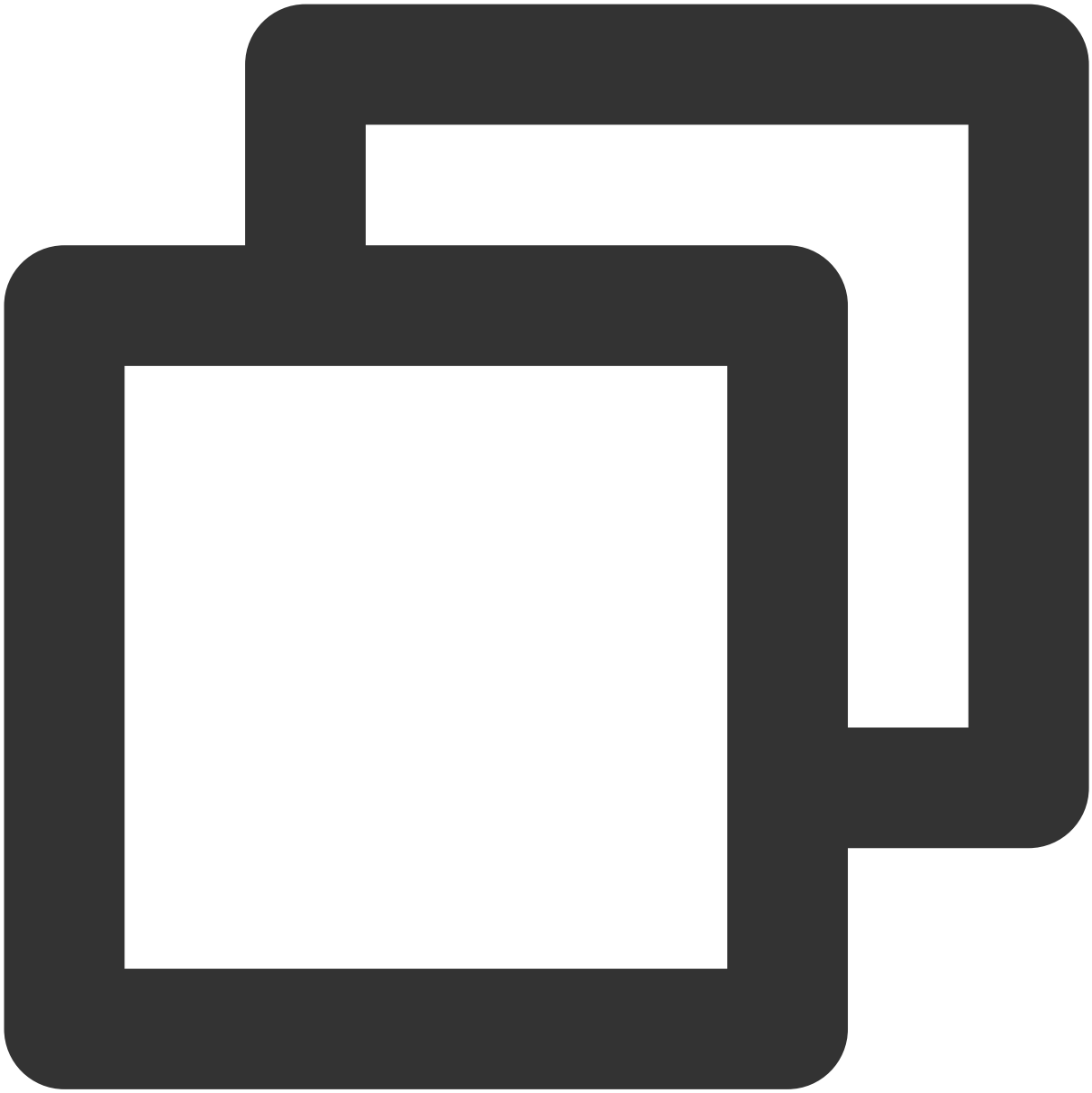
传入图层，获得当前图层的层级，依此可以判断图层显示过程中的先后覆盖顺序，以及触摸事件的先后传递顺序。



```
layerManger.indexOfLayer(layer);
```

19. 自定义列表滑行速率

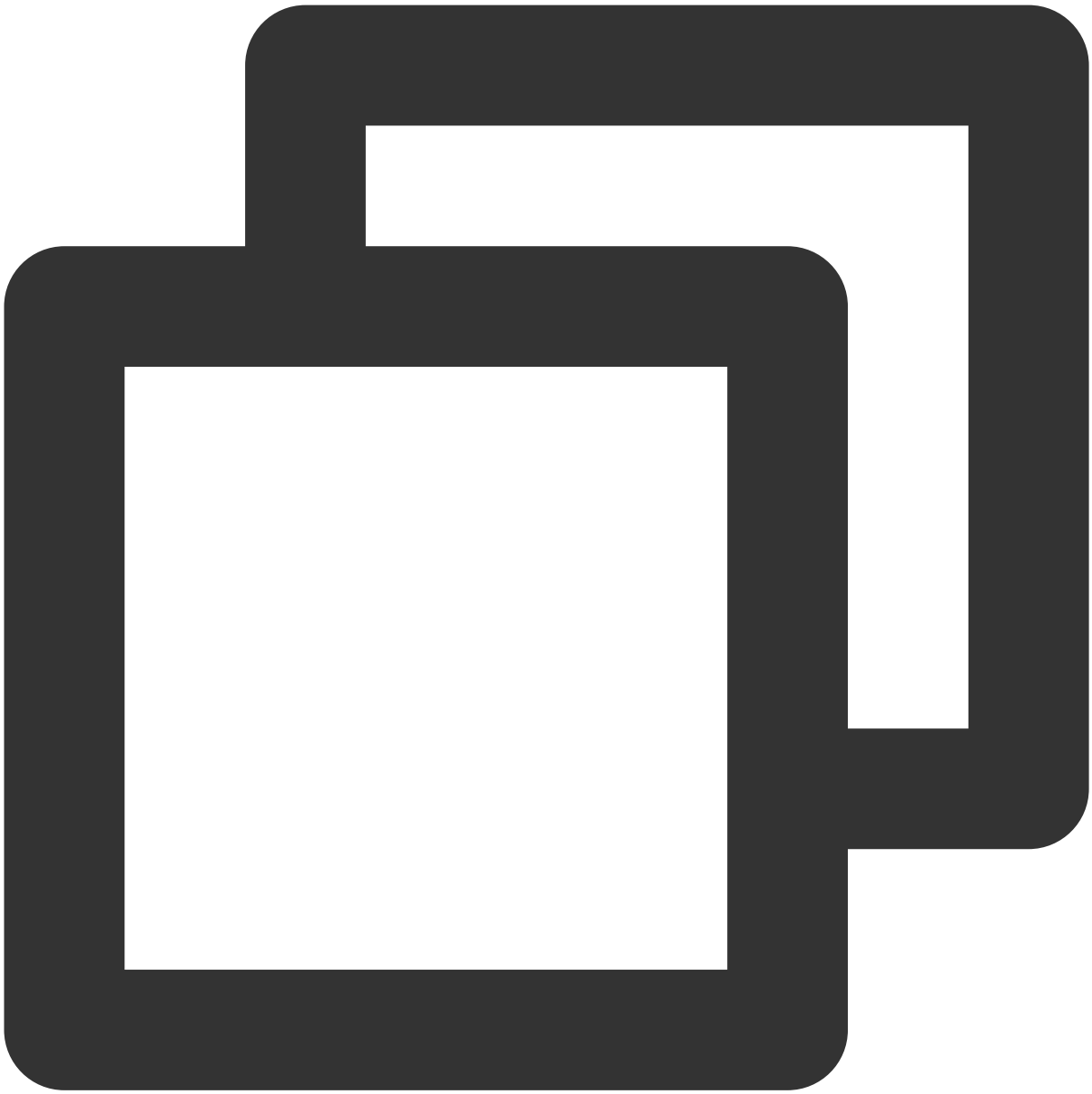
TUI 短视频提供了自定义列表滑行速率的接口，可通过 `TUIShortVideoView` 的 `setPageScroller` 方法来实现，示例如下：



```
mSuperShortVideoView.setPageScroller(new LinearSmoothScroller(getContext()){
    @Override
    protected float calculateSpeedPerPixel(DisplayMetrics displayMetrics) {
        // 返回滑行每一个像素所用的时间，单位：毫秒
        return 25f/displayMetrics.densityDpi;
    }
});
```

20. 禁止列表滑动

可以通过 `TUIShortVideoView` 的 `setUserInputEnabled` 方法来禁止或者允许列表滑动。示例如下：



```
// false 禁止用户滑动
mSuperShortVideoView.setUserInputEnabled(false);
```

21. 图层回调

Layer 回调包括基类的 `ITUILayer` 回调、播放器事件通知，以及来自 videoView 的组件事件，继承 `TUIVodLayer` 、 `TUILiveLayer` 和 `TUICustomLayer` 后，可以根据功能需要来接收视频播放的回调，目前回调函数如下：

TUIVodLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
getVideoView	TUIBaseVideoView	-	获得当前 VideoView，如果 layerManager 与 VideoView 还未绑定会返回空。
getPlayerController	TUIPlayerController	-	获得当前播放 Controller，如果还未与 Controller 发生绑定，会返回空。
getPlayer	ITUIVodPlayer	-	获得当前播放器，如果还未与 Controller 发生绑定，会返回空。
getRenderMode	int	-	获得当前播放器画面渲染填充模式。
onControllerBind	void	controller：当前视频播放控制器	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在播放的视频。
onControllerUnBind	void	controller：当前视频播放控制器	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。
onBindData	void	videoSource：视频数据	当前 VideoView 绑定了视频数据。
onViewRecycled	void	videoView: 当前播放器 view 容器	当前 videoView 被回收。
onExtInfoChanged	void	videoSource: 变化后的视频	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通

		源	知。
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 layer中，供 layer 释放资源。

TUILiveLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
getVideoView	TUIBaseVideoView	-	获得当前 VideoView，如果 layerManager 与 VideoView 还未绑定会返回空。
getPlayerController	TUIPlayerController	-	获得当前播放 Controller，如果还未与 Controller 发生绑定，会返回空。
getPlayer	ITUILivePlayer	-	获得当前播放器，如果还未与 Controller 发生绑定，会返回空。
getRenderMode	int	-	获得当前播放器画面渲染填充模式。
onControllerBind	void	controller：当前视频播放控制器	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在播放的视频。
onControllerUnBind	void	controller：当前视频播放控制器	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。
onBindData	void	videoSource：直播数据	当前 VideoView 绑定了视频数据。

onViewRecycled	void	videoView: 当前播放器 view 容器	当前 videoView 被回收。
onExtInfoChanged	void	source: 变化后的数据	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通知
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 layer 中，供 layer 释放资源

TUICustomLayer 类

函数	返回类型	参数	描述
isShowing	boolean	-	当前图层是否正在显示。
createView	View	parent：图层容器	抽象方法，需要自己实现，用于创建图层的 View。
tag	String	-	图层的 tag，用于区分不同的图层。
unBindLayerManager	void	-	图层与管理器发生解绑，一般发生在图层被移除的时候。
show	void	-	显示当前图层。
hidden	void	-	隐藏当前图层。
getView	T extends View	-	获得当前图层的 View。
onControllerBind	void	-	当前 VideoView 与播放控制器发生绑定，即当前视频变为列表中正在显示。
onControllerUnBind	void	-	VideoView 与播放控制器发生解绑，一般代表该 page 滑出界面。
onBindData	void	videoSource：数据	当前 VideoView 绑定了视频数据。
onViewRecycled	void	videoView: 当前view 容器	当前 videoView 被回收。
onExtInfoChanged	void	source: 变化后的数据	当通过 TUIPlaySource 设置 extInfo 之后，layer 中将通过该回调进行事件通知
onShortVideoDestroyed	void	-	当整个短视频组件被销毁，会通过该回调通知到 laye r中，供 layer 释放资源

TUIVodObserver 类

函数	返回类型	参数	描述
onPlayPrepare	void	-	视频准备完毕
onPlayBegin	void	-	视频开始播放
onPlayPause	void	-	视频暂停
onPlayStop	void	-	视频停止
onPlayLoading	void	-	视频开始加载
onPlayLoadingEnd	void	-	视频加载结束
onPlayProgress	void	current：当前视频播放进度，单位毫秒，long 类型。 duration：当前视频总时长，单位毫秒，long 类型。 playable：当前视频可播放时长，单位毫秒，long 类型	视频播放进度
onSeek	void	position：跳转到的视频进度。单位秒，int 类型。	视频进度发生跳转
onError	void	code：视频错误码 message：错误描述	视频播放发生错误
onRcvFirstIframe	void	-	收到首帧事件
onRcvTrackInformation	void	infoList：视频轨道	收到视频轨道信息
onRcvSubTitleTrackInformation	void	infoList：视频字幕信息	收到视频字幕信息
onRecFileVideoInfo	void	params：视频文件信息	收到视频文件信息，一般只使用 fileId 播放才会触发该回调。
onResolutionChanged	void	width：视频宽度 height：视频高度	当前视频分辨率发生变化
onPlayEvent	void	player：播放器 event：事件id bundle：事件内容	播放器所有事件通知
onPlayEnd	void	-	当前视频播放结束

TUILiveObserver 类

函数	参数	描述
onError	player: 当前直播播放器 code：错误码 msg：错误信息 extraInfo：附加消息	播放发生错误
onWarning	player: 当前直播播放器 code：错误码 msg：警告信息 extraInfo：附加消息	播放器发生警告
onVideoResolutionChanged	player: 当前直播播放器 width: 视频宽度 height: 视频高度	播放器分辨率发生变化
onConnected	player: 当前直播播放器 extraInfo: 附加信息	数据流连接成功
onVideoPlaying	player: 当前直播播放器 firstPlay: 是否是首次播放，**可以用来判断首帧** extraInfo: 附加信息	视频开始播放
onAudioPlaying	player: 当前直播播放器 firstPlay: 是否是首次播放 extraInfo: 附加信息	音频开始播放
onVideoLoading	player: 当前直播播放器 extraInfo: 附加信息	视频开始加载
onAudioLoading	player: 当前直播播放器 extraInfo: 附加信息	音频开始加载
onPlayoutVolumeUpdate	player: 当前直播播放器 volume: 变化后的音量	播放器音量大小回调
onStatisticsUpdate	player: 当前直播播放器 statistics: 数据详情	直播播放器统计数据回调
onSnapshotComplete	player: 当前直播播放器 image: 截图的图片	截图回调
onRenderVideoFrame	player: 当前直播播放器 videoFrame: 图像帧	自定义视频渲染回调，调用enableObserveVideoFrame后开启回调

onPlayoutAudioFrame	player: 当前直播播放器 audioFrame: 音频帧	自定义音频数据回调，调用enableObserveAudioFrame开启回调
onReceiveSeiMessage	player: 当前直播播放器 payloadType: 回调数据的SEI payloadType。 data：数据	收到 SEI 消息的回调，发送端通过V2TXLivePusher中的sendSeiMessage来发送 SEI 消息
onStreamSwitched	player: 当前直播播放器 url: 切换的url code：状态码，0：成功，-1：切换超时，-2：切换失败，服务端错误，-3：切换失败，客户端错误。	分辨率无缝切换回调
onLocalRecordBegin	player: 当前直播播放器 code: 状态码。 0：录制任务启动成功。 -1：内部错误导致录制任务启动失败。 -2：文件后缀名有误（比如不支持的录制格式）。 -6：录制已经启动，需要先停止录制。 -7：录制文件已存在，需要先删除文件。 -8：录制目录无写入权限，请检查目录权限问题。 storagePath: 录制的文件地址。	录制任务开始的事件回调
onLocalRecording	player: 当前直播播放器 durationMs: 录制时长 storagePath: 录制的文件地址。	录制任务正在进行中的进展事件回调
onLocalRecordComplete	player: 当前直播播放器 code: 状态码。 0：结束录制任务成功。 -1：录制失败。 -2：切换分辨率或横竖屏导致录制结束。 -3：录制时间太短，或未采集到任何视频或音频数据，请检查录制时长，或是否已开启音、视频采集。	录制任务已经结束的事件回调

	storagePath: 录制的文件地址。	

22. 播放器函数

在 layer 中，可以通过 `getPlayer` 获得播放器对象，播放器 ITUIVodPlayer 和 ITUILivePlayer 的接口函数如下：

ITUIVodPlayer 类

函数	返回类型	参数	描述
prePlay	void	model：视频数据，类型 TUIVideoSource	对视频进行预播放，内部会有判重机制。
resumePlay	void	-	继续播放当前视频。
seekTo	void	time：需要跳转的时间点，单位秒，int 类型。	跳转到指定播放位置。
isPlaying	boolean	-	当前视频是否正在播放。
startPlay	void	model：视频数据，类型 TUIVideoSource。	播放视频
pause	void	-	暂停播放
stop	void	needClearLastImg：可选参数，是否清除当前画面	停止播放
getCurrentPlayTime	float	-	获得当前播放时间，单位：秒。
setDisplayView	void	videoView：视频渲染 View，类型 TUIVideoRenderView。	设置播放器要渲染的 View。

setSurface	void	surface：画布	设置视频要渲染的surface。
addPlayerObserver	void	observer：播放器监听，类型 PlayerObserver。	设置播放器监听。
removePlayerObserver	void	observer：播放器监听，类型 PlayerObserver。	移除播放器监听。
setConfig	void	config：视频配置，类型 TXVodPlayConfig。	设置视频配置。
setRenderMode	void	renderMode：渲染模式， FULL_FILL_SCREEN：保持宽高比且铺满 ADJUST_RESOLUTION：保持宽高比展示视频	设置平铺模式。
setStartTime	void	startTime：开始播放时间，单位：秒。float 类型	设置开始播放时间，在播放前调用有效，只会生效一次。循环播放后会从头开始。
setLoop	void	isLoop：是否循环	设置视频是否循环播放
setBitrateIndex	void	index：码率角标	设置当前码率
switchResolution	void	resolution：分辨率，即宽 × 高	设置改播放器的分辨率，如果有对应分辨率就会进行切换，仅对本次播放生效

getSupportResolution	List<TUIPlayerBitrateItem>	-	获取当前正在播放视频的分辨率列表
getBitrateIndex	int	-	获取当前正在播放视频的码率角标
setRate	void	rate：视频速率，正常速率为1.0	设置当前播放器播放速率
getCurrentPlaybackTime	float	-	获取当前播放器已播放时间，单位秒
getBufferDuration	float	-	获取当前视频已经缓冲的时间，单位秒
getDuration	float	-	获取当前视频的总时长，单位 秒
getPlayableDuration	float	-	获取当前视频可播放的事件，单位秒
getWidth	int	-	获取当前正在播放视频的宽度
getHeight	int	-	获取当前正在播放视频的高度

setRenderRotation	void	rotation：角度	设置当前视频旋转角度
enableHardwareDecode	boolean	enable：是否开启硬解	设置当前播放器是否开启硬解
setMute	void	mute：是否静音	设置当前播放视频是否静音
setAudioPlayoutVolume	void	volume：视频音量，0到100	设置当前视频的输出音量
setRequestAudioFocus	boolean	requestFocus：是否获取音频焦点	设置当前播放器是否获取音频焦点
snapshot	void	listener：TXLivePlayer.ITXSnapshotListener类型，截图回调	对当前播放器正在播放的视频进行截图
setMirror	void	mirror：是否镜像	设置当前视频是否镜像播放
setToken	void	token：加密 HLS 的 token	设置加密 HLS 的 token
isLoop	boolean	-	当前播放器是否是循环播放
attachTRTC	void	trtcCloud：trtc服务对象	把播放器的音视频流通过 TRTC 进行推送，更多 TRTC 用

			务请参
TRTC
品概述。
detachTRTC	void	-	点播解
TRTC 服
务。
setStringOption	void	key：业务参数键值 value：业务参数值	设置播
器业务
数
setSubtitleView	void	subtitleView：字幕组件	设置视
字幕的
件
addSubtitleSource	void	url：字幕链接 name：字幕名称 mimeType：字幕格式	向视频
加字幕
selectTrack	void	trackIndex：音视频轨道	添加选
音视频
道，目
常用于
音轨、
幕的选
deselectTrack	void	trackIndex：音视频轨道	移除选
音视频
道
setAudioNormalization	void	value: 响度范围：-70～0 (LUFS)，同时支持自定义数值。注意：播放器高级版 11.7 版本开始支持。可填预设值，相关常量TXVodConstants, 关: AUDIO_NORMALIZATION_OFF 开: AUDIO_NORMALIZATION_STANDARD（标准） AUDIO_NORMALIZATION_LOW （低） AUDIO_NORMALIZATION_HIGH（高）	设置音
均衡
setSubtitleStyle	void	renderModel: 字幕格式	设置字
格式
getSubtitleTrackInfo	List<TXTrackInfo>	-	获取导
的字幕

			道信息，prepare事件之后调用才有效
getAudioTrackInfo	List<TXTrackInfo>	-	获取导入的音频轨道信息，prepare事件之后调用才有效

TUIPlayerBitrateItem 类

函数	返回类型	描述
getIndex	int	当前分辨率的码率角标
getWidth	int	当前分辨率的视频宽度
getHeight	int	当前分辨率的视频高度
getBitrate	int	当前分辨率的视频码率

ITUILivePlayer 类

函数	返回类型	参数	描
prePlay	void	model：视频数据，类型 TUILiveSource	对
resumePlay	void	-	继
setConfig	void	config：直播配置	设
addLiveObserver	void	observer: 直播事件观察者	订
removeLiveObserver	void	observer: 直播事件观察者	移
switchStream	int	model：需要切换的数据源	切功
setRenderRotation	int	rotation：旋转方向	设

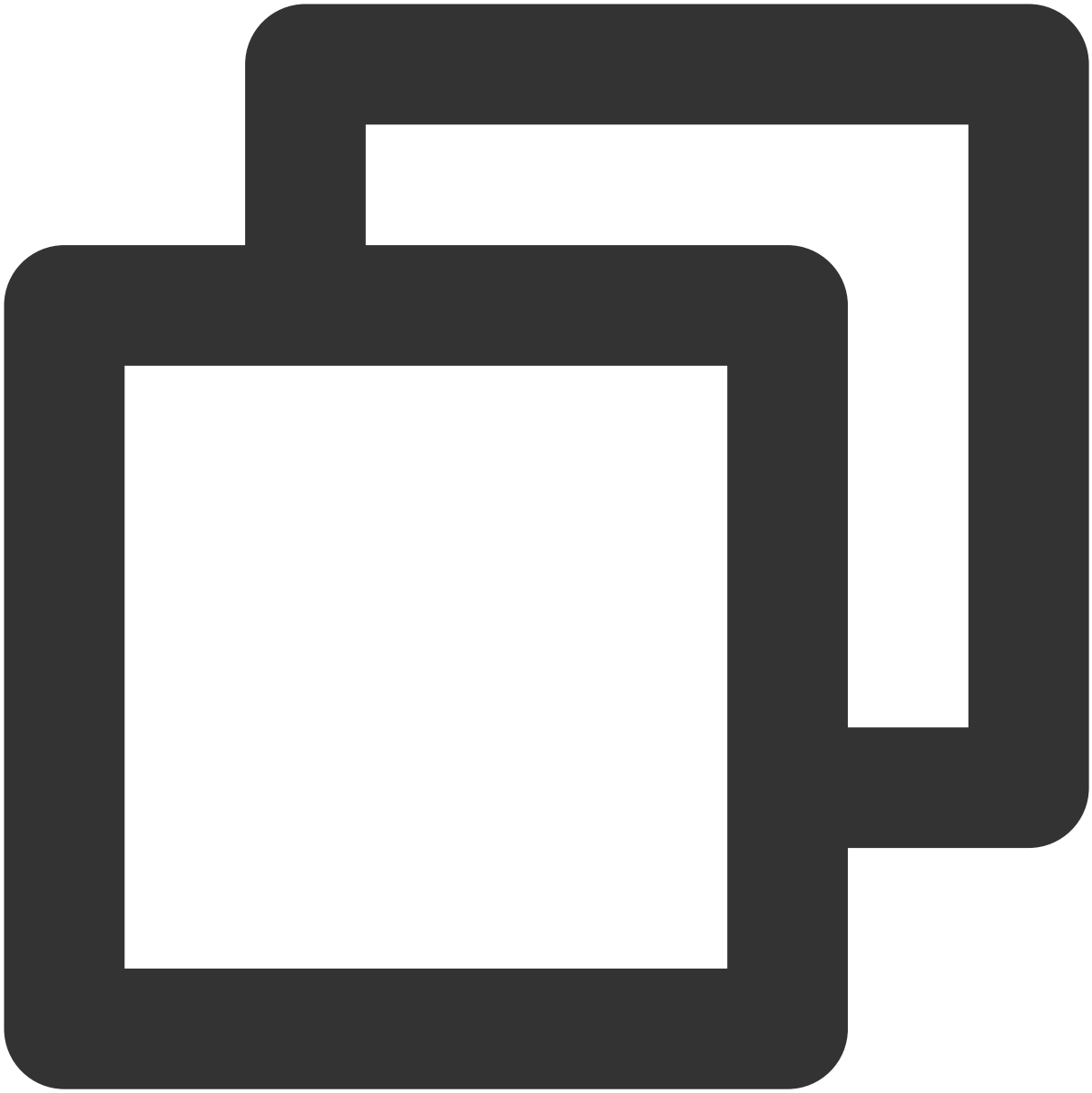
setPlayoutVolume	int	volume：音量大小，取值范围0 - 100。【默认值】: 100	设
getStreamList	ArrayList<V2TXLiveDef.V2TXLiveStreamInfo>	-	获
enableVolumeEvaluation	int	value：决定了onPlayoutVolumeUpdate回调的触发间隔，单位为ms，最小间隔为100ms，如果小于等于0则会关闭回调，建议设置为300ms；【默认值】：0，不开启	启
snapshot	int	-	截 返 V2 V2 停
enableReceiveSeiMessage	int	enable: true: 开启接收 SEI 消息; false: 关闭接收 SEI 消息。【默认值】: false。 payloadType: 指定接收 SEI 消息的 payloadType，支持 5、242，请与发送端的 payloadType 保持一致。	开
showDebugView	void	isShow: 是否显示。【默认值】：false。	是
setProperty	int	key: 高级 API 对应的 key。 value: 调用 key 所对应的高级 API 时，需要的参数。	调
startLocalRecording	int	params:本地录制音视频配置	开 返 V2 V2 参

			V2 拉
stopLocalRecording	void	-	停 还

高级功能

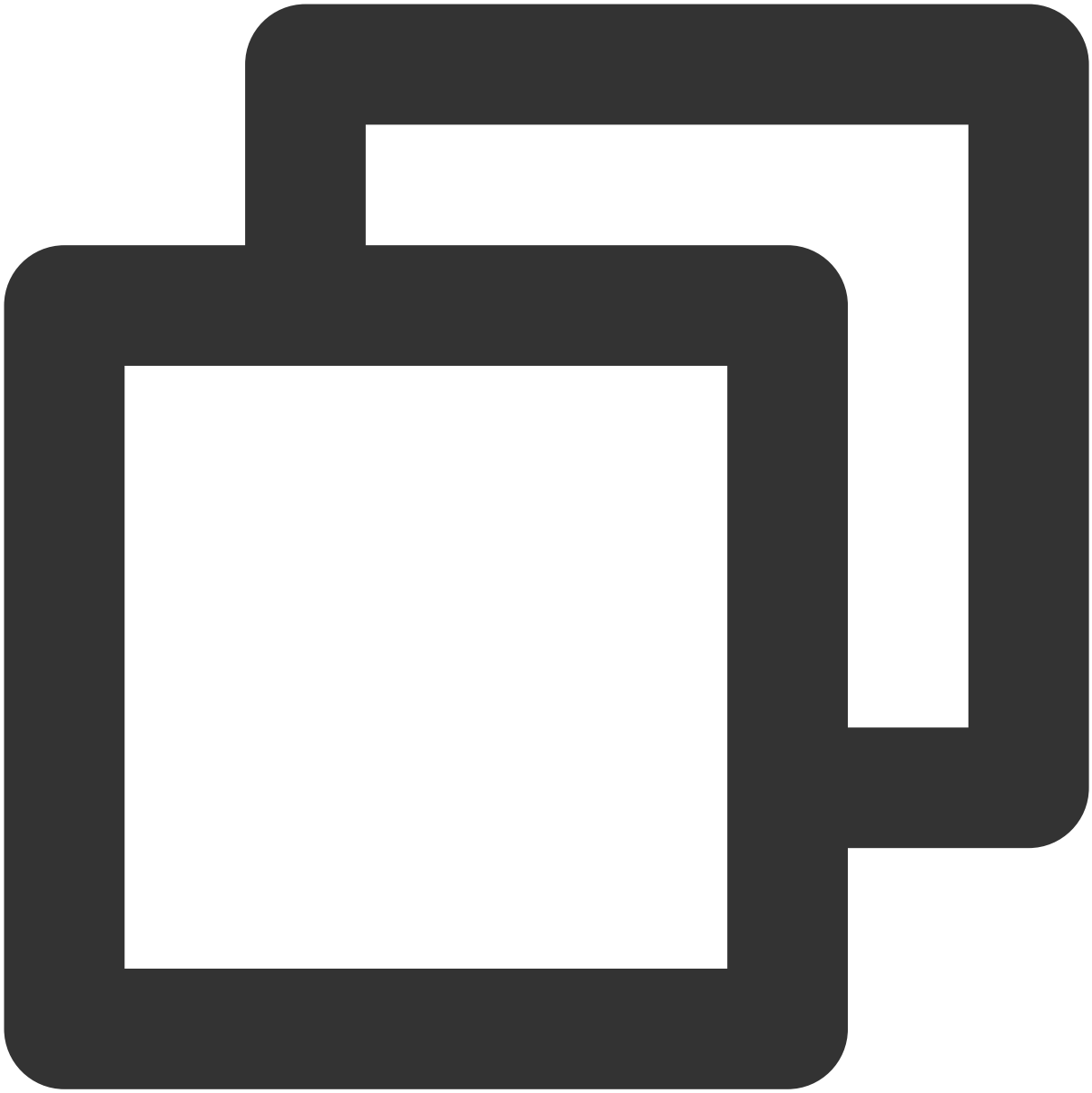
1. 业务通知消息到页面图层

TUI 提供了消息接口供用户把数据实时通知到当前图层，可以通过数据操作对象获取到视频对象之后，进行通知，示例如下：



```
// get data controller
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
// get dataSource
TUIPlaySource videoSource = dataManager.getDataByPageIndex(0);
Map<String, String> data = new HashMap<>();
data.put("key1", "data1");
// set extInfo and notify to layer
videoSource.setExtInfoAndNotify(data);
```

随后在 layer 的 `onExtInfoChanged` 回调中会受到该通知，从而对当前页面进行 UI 修改，示例如下：



```
@Override
public void onExtInfoChanged(TUIVideoSource videoSource) {
    super.onExtInfoChanged(videoSource);
    Map<String, String> data = (Map<String, String>) videoSource.extInfo;
    Log.i("tag", "get data:" + data);
}
```

注意：

`onExtInfoChanged` 只会在 `onBindData` 绑定数据之后才会触发。

2. 点播设置音量均衡

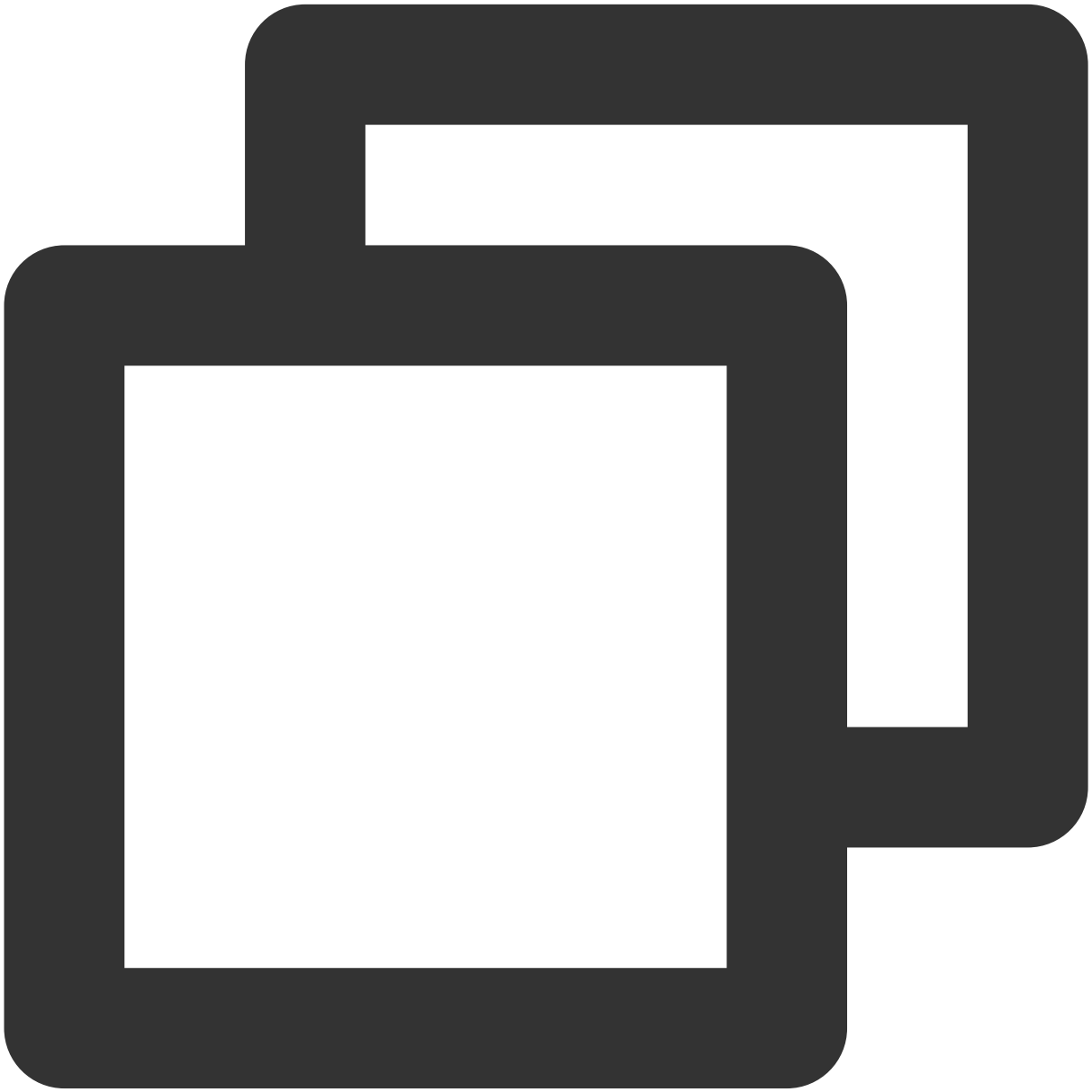
播放器支持在播放音频时自动调整音量，使得所有音频的音量保持一致。这可以避免某些音频过于响亮或过于安静的问题，提供更好的听觉体验。

通过

设置音量均衡，响度范围：-70～0 (LUFS)，同时支持自定义数值。

注意：

播放器高级版 11.7 版本开始支持。



```
/*
 * 支持自定义数值，响度范围：-70～0(LUFS)
```

```
 * 预设值   关：TXVodConstants#AUDIO_NORMALIZATION_OFF
 *         开（低）：TXVodConstants#AUDIO_NORMALIZATION_LOW
 *         开（标准）：TXVodConstants#AUDIO_NORMALIZATION_STANDARD
 *         开（高）：TXVodConstants#AUDIO_NORMALIZATION_HIGH
 */
TUIPlayerVodStrategy vodStrategy = new TUIPlayerVodStrategy.Builder()
        .setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD)
        .build();
mShortVideoView.setVodStrategy(vodStrategy);
```

API 文档

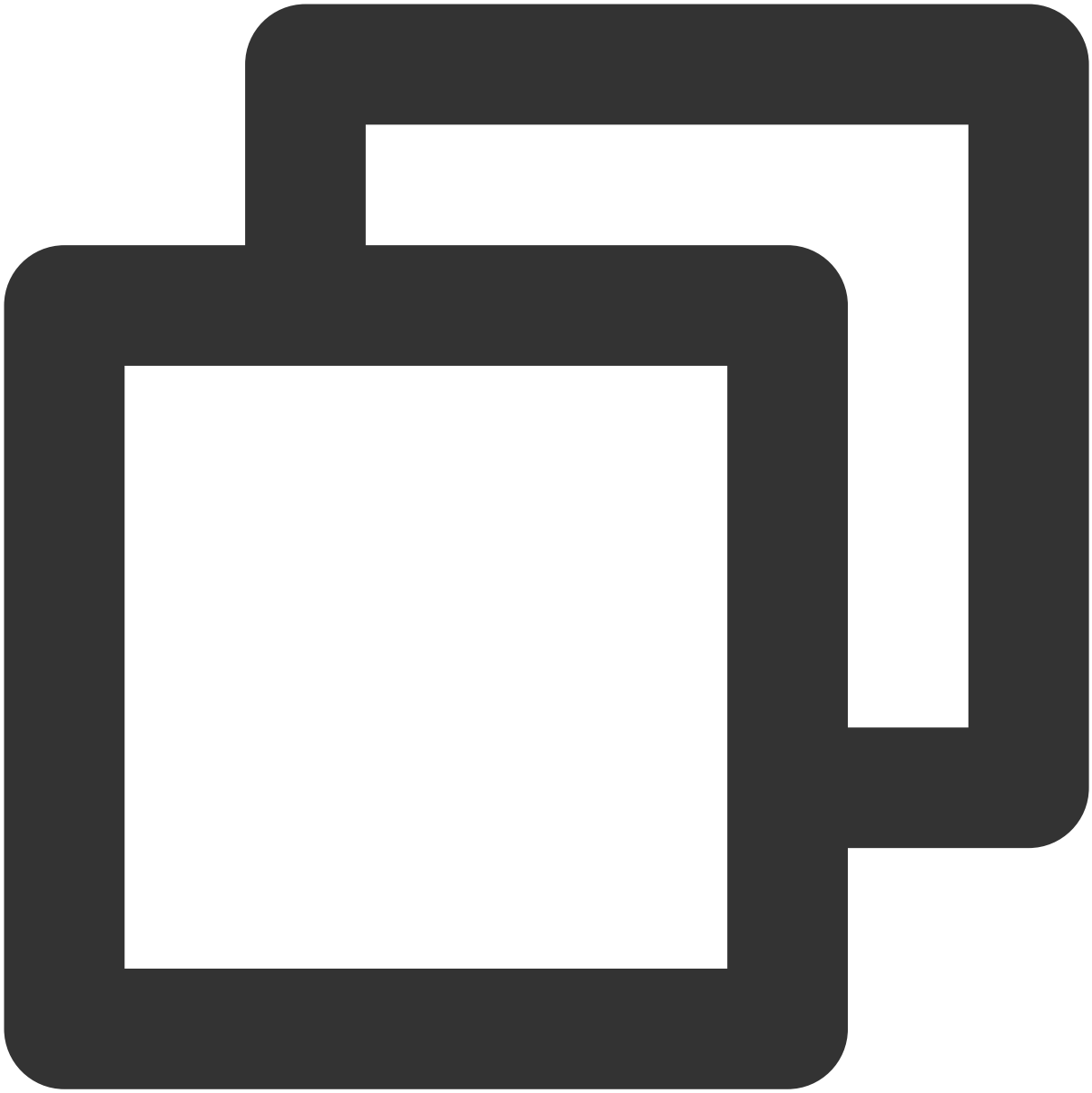
Web

最近更新时间：2024-04-11 16:18:08

本文档是介绍适用于直播和点播播放的 Web 播放器（ TCPlayer ）的相关参数以及 API。本文档适合有一定 Javascript 语言基础的开发人员阅读。

初始化参数

播放器初始化需要传入两个参数，第一个为播放器容器 ID，第二个为功能参数对象。



```
var player = TCPlayer('player-container-id', options);
```

options 参数列表

options 对象可配置的参数如下表：

名称	类型	默认值	说明
appID	String	无	通过 fileID 播放点播媒体文件时必选，为对应腾讯云账号的 appID

fileID	String	无	通过 fileID 播放点播媒体文件时必选，为点播媒体文件的 ID
psign	String	无	播放器签名，通过 fileID 播放时必传，详情参见 播放器签名
licenseUrl	String	无	播放器 License 地址，查看 播放器 License
sources	Array	无	播放器播放地址，格式：[{ src: '//path/to/video.mp4', type: 'video/mp4' }]
width	String/Number	无	播放器区域宽度，单位像素，按需设置，可通过 CSS 控制播放器尺寸。
height	String/Number	无	播放器区域高度，单位像素，按需设置，可通过 CSS 控制播放器尺寸。
controls	Boolean	true	是否显示播放器的控制栏。
poster	String	无	设置封面图片完整地址（如果上传的视频已生成封面图，优先使用生成的封面图，详细请参见 云点播 - 管理视频）。
autoplay	Boolean	false	是否自动播放。
playbackRates	Array	[0.5，1，1.25，1.5，2]	设置变速播放倍率选项，仅 HTML5 播放模式有效。
loop	Boolean	false	是否循环播放。
muted	Boolean	false	是否静音播放。
preload	String	auto	是否需要预加载，有3个属性"auto"，"meta"和"none"，移动端由于系统限制，设置 auto 无效。
swf	String	无	Flash 播放器 swf 文件的 URL
posterImage	Boolean	true	是否显示封面。
bigPlayButton	Boolean	true	是否显示居中的播放按钮（浏览器劫持嵌入的播放按钮无法去除）。
language	String	"zh-CN"	设置语言，可选值为 "zh-CN"/"en"
languages	Object	无	设置多语言词典。
controlBar	Object	无	设置控制栏属性的参数组合，具体参见 controlBar 参数列表。

reportable	Boolean	true	设置是否开启数据上报。
fakeFullscreen	Boolean	false	设置开启伪全屏，通过样式控制来实现全屏效果。
plugins	Object	无	设置插件功能属性的参数组合，具体参见 plugins 插件参数列表。
hlsConfig	Object	无	hls.js 的启动配置，详细内容请参见官方文档 hls.js。
webrtcConfig	Object	无	webrtc 的启动配置，具体参见 webrtcConfig 参数列表。
xp2pConfig	Object	无	P2P 的启动配置，具体参见 xp2pConfig 参数列表。 P2P 功能详情请参见 X-P2P。

注意：

controls、playbackRates、loop、preload、posterImage 这些参数在浏览器劫持播放的状态下将无效。

浏览器劫持视频播放问题参见 常见问题说明。

controlBar 参数列表

controlBar 参数可以配置播放器控制栏的功能，支持的属性如下表：

名称	类型	默认值	说明
playToggle	Boolean	true	是否显示播放、暂停切换按钮。
progressControl	Boolean	true	是否显示播放进度条。
volumePanel	Boolean	true	是否显示音量控制。
currentTimeDisplay	Boolean	true	是否显示视频当前时间。
durationDisplay	Boolean	true	是否显示视频时长。
timeDivider	Boolean	true	是否显示时间分隔符。
playbackRateMenuButton	Boolean	true	是否显示播放速率选择按钮。
fullscreenToggle	Boolean	true	是否显示全屏按钮。
QualitySwitcherMenuButton	Boolean	true	是否显示清晰度切换菜单。

注意：

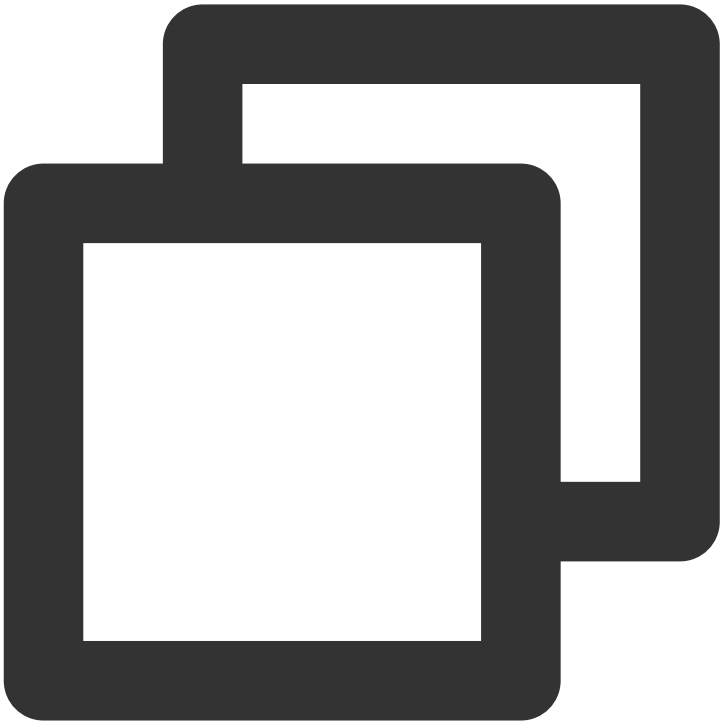
controlBar 参数在浏览器劫持播放的状态下将无效。

浏览器劫持视频播放问题参见 常见问题说明。

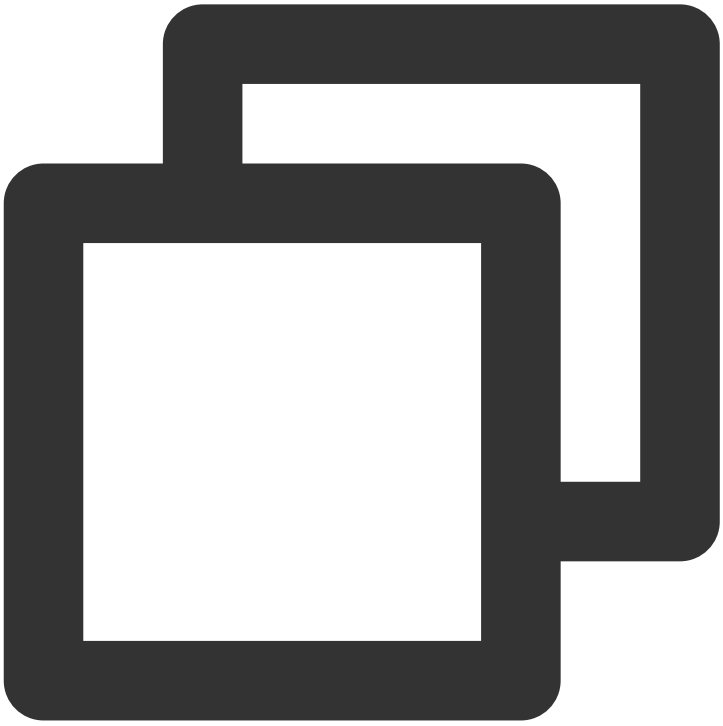
plugins 插件参数列表

plugins 参数可以配置播放器插件的功能，支持的属性有：

名称	类型	默认值	说明
ContinuePlay	Object	无	控制续播功能，支持的属性如下： auto：Boolean 是否在播放时自动续播。 text：String 提示文案。 btnText：String 按钮文案。
VttThumbnail	Object	无	控制缩略图显示，支持的属性如下： vttUrl：String vtt 文件绝对地址，必传。 basePath：String 图片路径，非必须，不传时使用 vttUrl 的 path。 imgUrl：String 图片绝对地址，非必须。
ProgressMarker	Boolean	无	控制进度条显示。
DynamicWatermark	Object	无	控制动态水印显示，支持文字和图片，支持的属性为： type：String 水印类型为文字或图片，取值为 text | image，默认 text，非必传。 content：String 文字水印内容，必传。 speed：Number 水印移动速度，取值范围 0-1, 默认值 0.2，非必传。 opacity：Number 文字水印透明度，取值范围 0-1，非必传。 fontSize：String 文字字体大小，默认12px，非必传。 color： String 文字颜色，非必传。 left：String, 文字位置，支持单位为百分比和 px，该字段设置时，speed 字段无效，非必传。 top、right、bottom：说明同 left。 width：String 图片水印宽度，非必传。 height：String 图片水印高度，非必传 。
ContextMenu	Object	无	可选值如下： mirror：Boolean 控制是否支持镜像显示。 statistic：Boolean 控制是否支持显示数据面板。 levelSwitch：Object 控制切换清晰度时的文案提示。

			
			{
			open: Boolean 是否开启提示
			switchingText: String, 开始切换清晰度时的提示文案
			switchedText: String, 切换成功时的提示文案
			switchErrorText: String, 切换失败时的提示文案
			}

PlayList	Object	无	设置播放列表，支持的属性如下：

			
			{
			// 要播放的视频信息集合
			data: [{
			fileID: String,
			appID: String,
			duration: Number, // 视频时长
			text: String, // 视频名称
			psign: String, // 播放器签名
			img: String, // 封面图
			}],
			title: String, // 列表标题
			loop: Boolean, // 是否循环播放
			}

VR	Object	无	高级版 License 支持，详情参见 Web 高级功能 - VR 播放插件（TCPlayerVRPlugin）
SafeCheck	Object	无	高级版 License 支持，详情参见 Web 高级功能 - 安全检查插件（TCPlayerSafeCheckPlugin）

webrtcConfig 参数列表

webrtcConfig 参数来控制播放 webrtc 过程中的行为表现，支持的属性如下表：

名称	类型	默认值	说明
connectRetryCount	Number	3	SDK 与服务器重连次数
connectRetryDelay	Number	1	SDK 与服务器重连延时
receiveVideo	Boolean	true	是否拉取视频流
receiveAudio	Boolean	true	是否拉取音频流
showLog	Boolean	false	是否在控制台打印日志

xp2pConfig 参数列表

使用 X-P2P 前，需要申请开通，请移步 X-P2P 单击申请，申请后我们会有专门的产品支持人员联系您。

更多详细资料，请参考 X-P2P 产品文档。

公共参数

名称	类型	默认值	说明
useXP2P	Boolean	false	是否开启 XP2P
format	String	无	告知 P2P 需要支持的媒体协议，请根据当前播放的视频格式填写，可选值如下： flv hls webrtc
tencentCloudAppId	Number	无	在腾讯云账号的 APPID (控制台查看路径：账号中心 > **账号信息 > 基本信息 > APPID**)
xp2pAppId	String	无	X-P2P 分配的 ID，由我们邮件提供
xp2pAppKey	String	无	X-P2P 分配的 Key，由我们邮件提供
xp2pPackage	String	无	X-P2P 分配的 Package，由我们邮件提供

flv 协议额外参数

名称	类型	默认值	说明
bizId	String	无	由我们邮件提供
xp2pPlayDomain	String	无	flv 协议的拉流域名，由我们邮件提供
authMode	String	无	鉴权模式，由我们邮件提供
debug	Boolean	false	debug 开关

hls 协议额外参数

名称	类型	默认值	说明
videoType	String	VOD	当前播放的 HLS 是直播还是点播，请准确填写，可选值如下： LIVE VOD
channelId	String	自动生成	可以主动为当前资源生成一个 ID，如果不填，则默认内部自动生成。生成规则参见如下 HLS 资源 ID 生成规则
channelIdWithHost	Boolean	true	默认为 true，可选配置，通常不需要修改这个配置。详细解释参见如下 HLS 资源 ID 生成规则
channelIdWithSearch	Boolean	false	默认为 false，可选配置，通常不需要修改这个配置。详细解释参见如下 HLS 资源 ID 生成规则

HLS 资源 ID 生成规则

资源 ID 是 P2P 分享的单位，相同的资源 ID 的节点才能互相 P2P 分享。不同视频必须确保资源 ID 不同，否则会串流。

1.1 **主动传入**

可以通过设置参数 `channelId` 字段，主动为当前视频指定一个资源 ID，必须保证能唯一标识这个文件，避免串流。

1.2 **默认生成**

如果没有传入 `channelId` 字段，sdk 会默认为每一个 url 生成一个 ID，相同 ID 的会互相 P2P 分享，ID 生成规则如下：

(默认) 截取 host 和 path 部分生成 MD5。

> 例如：https://a.b.com/p1/p2/p3.m3u8?m=1&n=2，则 ID = MD5('a.b.com/p1/p2/p3.m3u8')

1.3 **传入参数控制默认生成规则**

(可选，默认为 true) 通过传入 `channelIdWithHost` 参数，true 表示 ID 包含 host 部分。

(可选，默认为 false) 通过传入 `channelIdWithSearch` 参数，false 表示包含 search 部分。

例如：https://a.b.com/p1/p2/p3.m3u8?m=1&n=2
如果传入 { channelIdWithHost: true, channelIdWithSearch: true } 则 ID = MD5('a.b.com/p1/p2/p3.m3u8?m=1&n=2')
如果传入 { channelIdWithHost: false, channelIdWithSearch: false } 则 ID = MD5('/p1/p2/p3.m3u8')

说明：

可以根据自己业务 url 生成规则，自行选择搭配，目的是确保不能互相 P2P 的视频 url，生成不同的资源 ID，可以互通的 url，生成相同的资源 ID。

1.4 多码率 M3U8 说明

如果播放的视频是多码率 M3U8，我们内部会保证播放不同码率的节点不会互相 P2P。

X-P2P 协议支持

音视频协议	用途	PC 浏览器	Android	iOS
FLV	直播	支持 chrome 55+ firefox 65+ safari 11+	支持 chrome 55+ firefox 65+ 微信浏览器	不支持
HLS	直播，点播	支持 chrome 55+ firefox 65+ safari 11+	支持 chrome 55+ firefox 65+ 微信浏览器	不支持
WebRTC	直播	部分支持 Chrome 94+ edge 94+	部分支持 Chrome 94+ edge 94+	不支持

对象方法

初始化播放器返回对象的方法列表：

名称	参数及类型	返回值及类型	说明
src()	(String)	无	设置播放地址。
loadVideoByID()	(Object)	无	通过 fileID 播放时，可通过这个方法切换视频，参数为由fileID、appID、psign 组成的对象。
ready(function)	(Function)	无	设置播放器初始化完成后的回调。

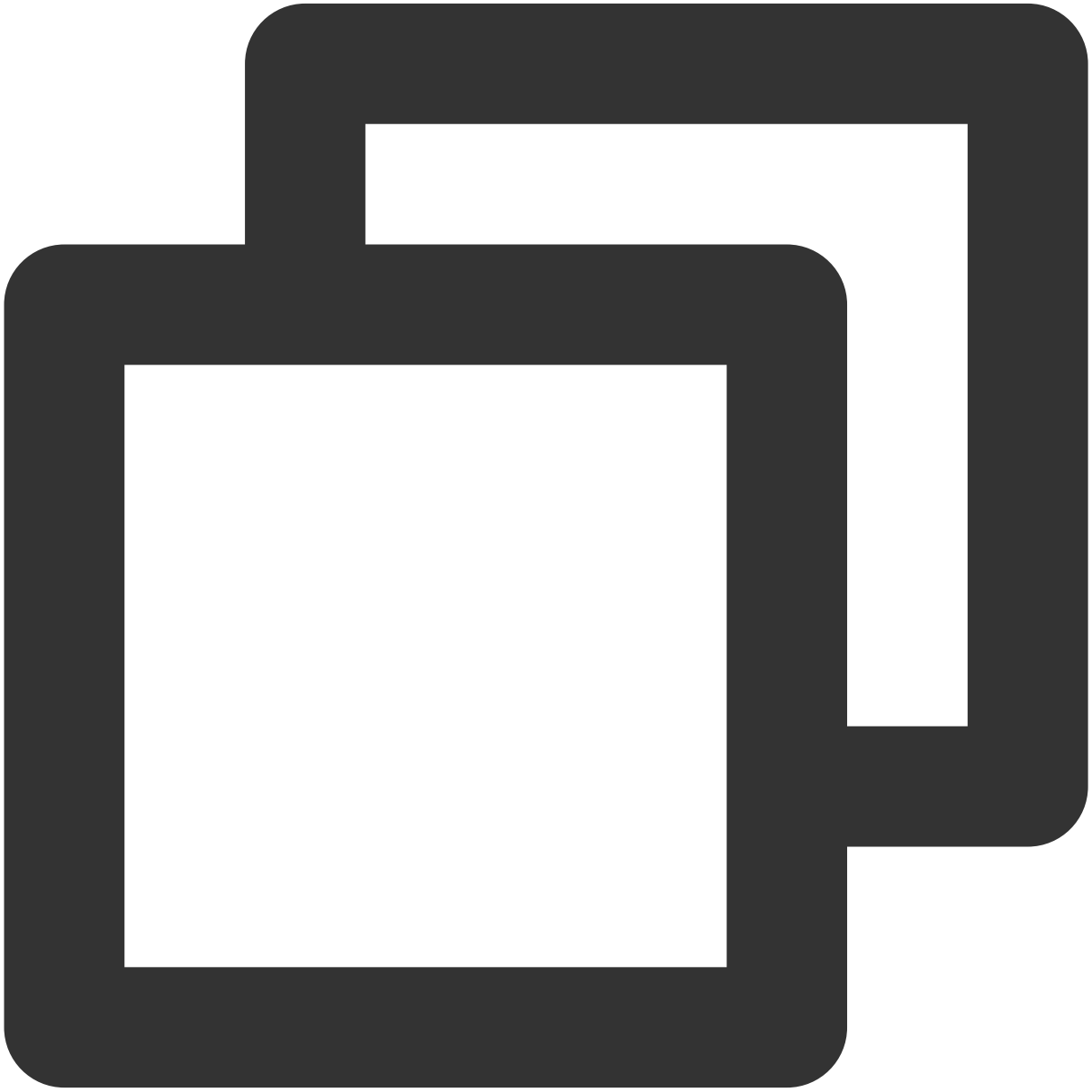
play()	无	无	播放以及恢复播放。
pause()	无	无	暂停播放。
currentTime(seconds)	(Number)	(Number)	获取当前播放时间点，或者设置播放时间点，该时间点不能超过视频时长。
duration()	无	(Number)	获取视频时长。
volume(percent)	(Number)[0，1][可选]	(Number)/设置时无返回	获取或设置播放器音量。
muted()	(Boolean)	(Boolean)	获取或设置播放器是否静音
playbackRate()	(Number)[0, 1]	(Number)	获取或设置播放倍速
poster(src)	(String)	(String)/设置时无返回	获取或设置播放器封面。
requestFullscreen()	无	无	进入全屏模式。
exitFullscreen()	无	无	退出全屏模式。
isFullscreen()	无	Boolean	返回是否进入了全屏模式。
on(type，listener)	(String, Function)	无	监听事件。
one(type，listener)	(String, Function)	无	监听事件，事件处理函数最多只执行1次。
off(type，listener)	(String, Function)	无	解绑事件监听。
buffered()	无	TimeRanges	返回视频缓冲区间。
bufferedPercent()	无	值范围[0，1]	返回缓冲长度占视频时长的百分比。
width()	(Number)[可选]	(Number)/设置时无返回	获取或设置播放器区域宽度，如果通过 CSS 设置播放器尺寸，该方法将无效。
height()	(Number)[可选]	(Number)/设置时无返回	获取或设置播放器区域高度，如果通过 CSS 设置播放器尺寸，该方法将无效。
videoWidth()	无	(Number)	获取视频分辨率的宽度。
videoHeight()	无	(Number)	获取视频分辨率的高度。
dispose()	无	无	销毁播放器。

注意

对象方法不能同步调用，需要在相应的事件（如 loadedmetadata）触发后才可以调用，除了 ready、on、one 以及 off。

事件

播放器可以通过初始化返回的对象进行事件监听，示例：



```
var player = TCPlayer('player-container-id', options);
```

```
// player.on(type, function);
player.on('error', function(error) {
    // 做一些处理
});
```

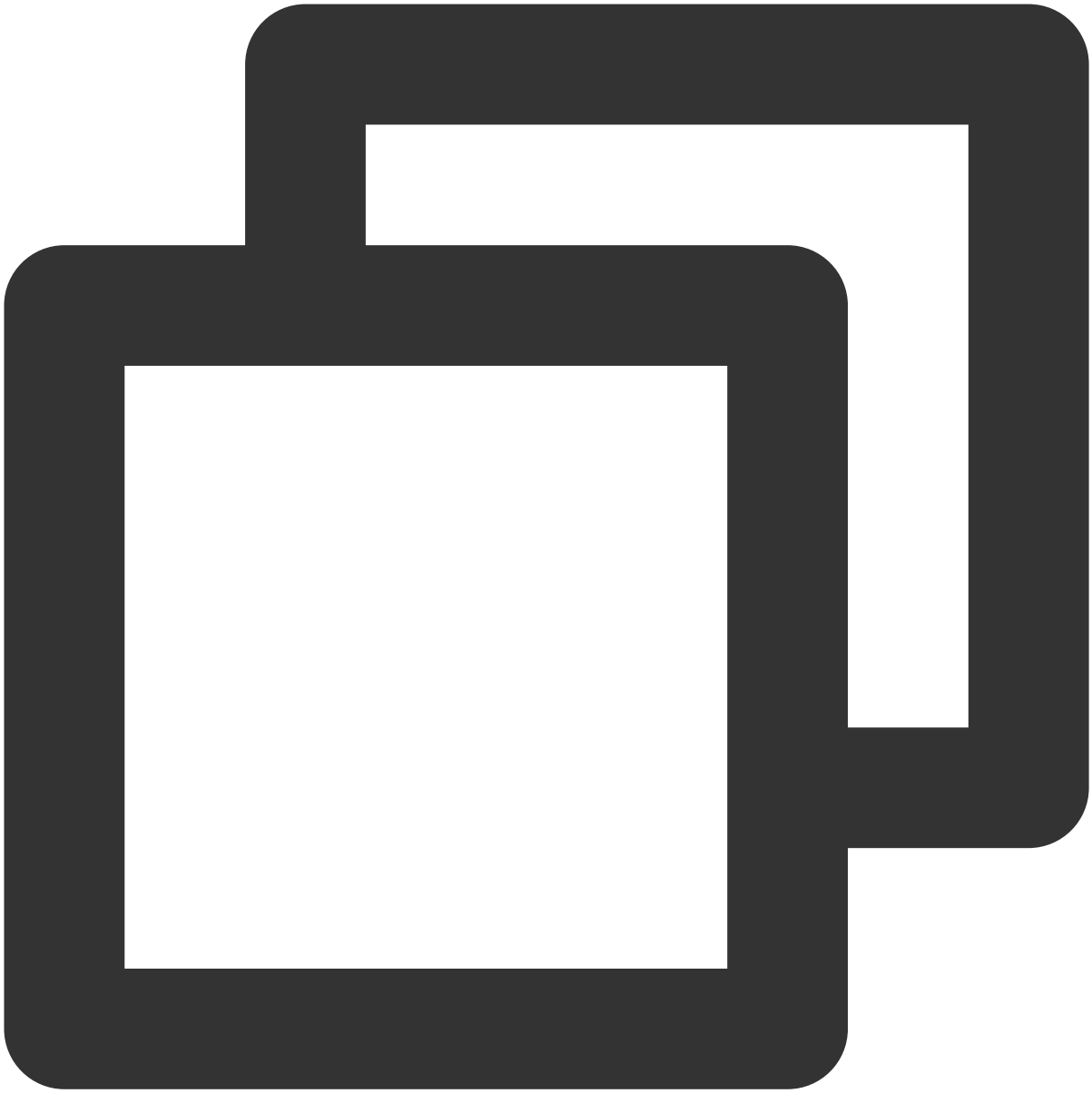
其中 type 为事件类型，支持的事件有：

名称	介绍
play	已经开始播放，调用 play() 方法或者设置了 autoplay 为 true 且生效时触发，这时 paused 属性为 false。
playing	因缓冲而暂停或停止后恢复播放时触发，paused 属性为 false 。通常用这个事件来标记视频真正播放，play 事件只是开始播放，画面并没有开始渲染。
loadstart	开始加载数据时触发。
durationchange	视频的时长数据发生变化时触发。
loadedmetadata	已加载视频的 metadata。
loadeddata	当前帧的数据已加载，但没有足够的数据来播放视频的下一帧时，触发该事件。
progress	在获取到媒体数据时触发。
canplay	当播放器能够开始播放视频时触发。
canplaythrough	当播放器预计能够在不停下来进行缓冲的情况下持续播放指定的视频时触发。
error	视频播放出现错误时触发。
pause	暂停时触发。
blocked	自动播放被浏览器阻止时触发。
ratechange	播放速率变更时触发。
seeked	搜寻指定播放位置结束时触发。
seeking	搜寻指定播放位置开始时触发。
timeupdate	当前播放位置有变更，可以理解为 currentTime 有变更。
volumechange	设置音量或者 muted 属性值变更时触发。
waiting	播放停止，下一帧内容不可用时触发。
ended	视频播放已结束时触发。此时 currentTime 值等于媒体资源最大值。
resolutionswitching	清晰度切换进行中。

resolutionswitched	清晰度切换完毕。
fullscreenchange	全屏状态切换时触发。
webrtcevent	播放 webrtc 时的事件集合。
webrtcstats	播放 webrtc 时的统计数据。
webrtcfallback	播放 webrtc 时触发降级

WebrtcEvent 列表

播放器可以通过 webrtcevent 获取播放 webrtc 过程中的所有事件，示例：



```
var player = TCPlayer('player-container-id', options);
player.on('webrtcevent', function(event) {
   // 从回调参数 event 中获取事件状态码及相关数据
});
```

webrtcevent 状态码如下

状态码	回调参数	介绍
1001	无	开始拉流

1002	无	已经连接服务器
1003	无	视频播放开始
1004	无	停止拉流，结束视频播放
1005	无	连接服务器失败，已启动自动重连恢复
1006	无	获取流数据为空
1007	localSdp	开始请求信令服务器
1008	remoteSdp	请求信令服务器成功
1009	无	拉流卡顿等待缓冲中
1010	无	拉流卡顿结束恢复播放

错误码

当播放器触发 error 事件时，监听函数会返回错误码，其中3位数以上的错误码为媒体数据接口错误码。错误码列表：

名称	描述
-1	播放器没有检测到可用的视频地址。
-2	获取视频数据超时。
1	视频数据加载过程中被中断。 可能原因： 网络中断。 浏览器异常中断。 解决方案： 查看浏览器控制台网络请求信息，确认网络请求是否正常。 重新进行播放流程。
2	由于网络问题造成加载视频失败。 可能原因：网络中断。 解决方案： 查看浏览器控制台网络请求信息，确认网络请求是否正常。 重新进行播放流程。
3	视频解码时发生错误。 可能原因：视频数据异常，解码器解码失败。 解决方案：

	尝试重新转码再进行播放，排除由于转码流程引入的问题。 确认原始视频是否正常。 请联系技术客服并提供播放参数进行定位排查。
4	视频因格式不支持或者服务器或网络的问题无法加载。 可能原因： 获取不到视频数据，CDN 资源不存在或者没有返回视频数据。 当前播放环境不支持播放该视频格式。 解决方案： 查看浏览器控制台网络请求信息，确认视频数据请求是否正常。 确认是否按照使用文档加载了对应视频格式的播放脚本。 确认当前浏览器和页面环境是否支持将要播放的视频格式。 请联系技术客服并提供播放参数进行定位排查。
5	视频解密时发生错误。 可能原因： 解密用的密钥不正确。 请求密钥接口返回异常。 当前播放环境不支持视频解密功能。 解决方案： 确认密钥是否正确，以及密钥接口是否返回正常。 请联系技术客服并提供播放参数进行定位排查。
10	点播媒体数据接口请求超时。在获取媒体数据时，播放器重试3次后仍没有任何响应，会抛出该错误。 可能原因： 当前网络环境无法连接到媒体数据接口，或者媒体数据接口被劫持。 媒体数据接口异常。 解决方案： 尝试打开我们提供的 Demo 页面看是否可以正常播放。 请联系技术客服并提供播放参数进行定位排查。
11	点播媒体数据接口没有返回数据。在获取媒体数据时，播放器重试3次后仍没有数据返回，会抛出该错误。 可能原因： 当前网络环境无法连接到媒体数据接口，或者媒体数据接口被劫持。 媒体数据接口异常。 解决方案： 尝试打开我们提供的 Demo 页面看是否可以正常播放。 请联系技术客服并提供播放参数进行定位排查。
12	点播媒体数据接口返回异常数据。在获取媒体数据时，播放器重试3次后仍返回无法解析的数据，会抛出该错误。 可能原因： 当前网络环境无法连接到媒体数据接口，或者媒体数据接口被劫持。 播放参数有误，媒体数据接口无法处理。

	媒体数据接口异常。 解决方案： 尝试打开我们提供的 Demo 页面看是否可以正常播放。 请联系技术客服并提供播放参数进行定位排查。
13	播放器没有检测到可以在当前播放器播放的视频数据，请对该视频进行转码操作。
14	HTML5 + hls.js 模式下播放 hls 出现网络异常，异常详情可在 event.source 中查看，详细介绍请看 hls.js 的官方文档 Network Errors。
15	HTML5 + hls.js 模式下播放 hls 出现多媒体异常，异常详情可在 event.source 中查看，详细介绍请看 hls.js 的官方文档 Media Errors。
16	HTML5 + hls.js 模式下播放 hls 出现多路复用异常，异常详情可在 event.source 中查看，详细介绍请看 hls.js 的官方文档 Mux Errors。
17	HTML5 + hls.js 模式下播放 hls 出现其他异常，异常详情可在 event.source 中查看，详细介绍请看 hls.js 的官方文档 Other Errors。
1013	播放器签名缺少 contentInfo 字段
10008	媒体数据服务没有找到对应播放参数的媒体数据，请确认请求参数 appID fileID 是否正确，以及对应的媒体数据是否已经被删除。
-2002	快直播拉流接口后台返回报错（例如流不存在、鉴权失败等）
-2006	快直播多分辨率平滑切换接口请求失败

iOS

最近更新时间：2024-04-11 16:11:38

TXVodPlayer

点播播放器

请参见 TXVodPlayer。
主要负责从指定的点播流地址拉取音视频数据，并进行解码和本地渲染播放。
播放器包含如下能力：
支持 FLV、MP4 及 HLS 多种播放格式，支持 基础播放（URL 播放） 和 点播播放（Fileid 播放）两种播放方式 。
屏幕截图，可以截取当前播放流的视频画面。
通过手势操作，调节亮度、声音、进度等。
可以手动切换不同的清晰度，也可根据网络带宽自适应选择清晰度。
可以指定不同倍速播放，并开启镜像和硬件加速。
完整能力，请参见 点播超级播放器 - 能力清单。

播放器配置接口

API	描述
config	点播配置，配置信息请参见 TXVodPlayConfig。
isAutoPlay	startVodPlay 后是否立即播放，默认 YES。
token	加密 HLS 的 token。设置此值后，播放器自动在 URL 中的文件名之前增加 `voddrm.token.TOKEN TextureView` 。
loop	是否循环播放 SurfaceView。
enableHWAcceleration	视频渲染回调。（仅硬解支持）
setExtentOptionInfo	设置播放器业务参数，参数格式为<NSString *, id>
setSubtitleStyle	设置字幕样式信息，可在播放后对字幕样式进行更新（播放器高级版本才支持）。
setAudioNormalization	设置音量均衡，响度范围：-70～0(LUFS) 播放器高级版 11.7 版本开始支持。 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h） 关：AUDIO_NORMALIZATION_OFF 开：AUDIO_NORMALIZATION_STANDARD（标准） AUDIO_NORMALIZATION_LOW （低） AUDIO_NORMALIZATION_HIGH（高）

	可填自定义数值： 从低到高，范围-70 - 0 LUFS

播放基础接口

API	描述
startVodPlay	播放 HTTP URL 形式地址。10.7 版本开始，`startPlay` 变更为 `startVodPlay`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用。
startPlayDrm:	启动一个标准 Fairplay drm 播放。
startVodPlayWithParams	以 fileId 形式播放。播放 HTTP URL 形式地址。10.7 版本开始，`startPlayWithParams` 变更为 `startVodPlayWithParams`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用。
stopPlay	停止播放。
isPlaying	是否正在播放。
pause	暂停播放，停止获取流数据,保留最后一帧画面。
resume	恢复播放，重新获取流数据。
seek	跳转到视频流指定时间点，单位秒。
currentPlaybackTime	获取当前播放位置，单位秒。
duration	获取总时长，单位秒。
playableDuration	获取可播放时长，单位秒。
width	获取视频宽度。
height	获取视频高度。
setStartTime	设置播放开始时间。
setupVideoWidget:insertIndex:	创建 Video 渲染 View，该控件承载着视频内容的展示
removeVideoWidget	移除 Video 渲染 View。

addSubtitleSource:name:mimeType:	添加外挂字幕（播放器高级版本才支持）。
getSubtitleTrackInfo	返回字幕轨道信息列表（播放器高级版本才支持）。
getAudioTrackInfo	返回音频轨道信息列表（播放器高级版本才支持）。
selectTrack:	选择轨道（播放器高级版本才支持）。
deselectTrack:	取消选择轨道（播放器高级版本才支持）。
seekToPdtTime	跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。（播放器高级版 11.6 版本开始支持）。 参数单位毫秒(ms)。

视频相关接口

API	描述
snapshot	获取当前视频帧图像。 **注意：由于获取当前帧图像是比较耗时的操作，所以截图会通过异步回调出来。**
setMirror	设置镜像。
setRate	设置点播的播放速率，默认1.0。
bitrateIndex	返回当前播放的码率索引。
setBitrateIndex	设置当前正在播放的码率索引，无缝切换清晰度。 清晰度切换可能需要等待一小段时间。
supportedBitrates	当播放地址为 master playlist，返回支持的码率（清晰度）。
setRenderMode	设置 图像平铺模式。
setRenderRotation	设置 图像渲染角度。
enterPictureInPicture	进入画中画功能。
exitPictureInPicture	退出画中画功能。

音频相关接口

API	描述
setMute	设置是否静音播放。

setAudioPlayoutVolume	设置音量大小，范围：0 - 100。

事件通知接口

API	描述
delegate	事件回调，建议使用 vodDelegate。
vodDelegate	设置播放器的回调。
videoProcessDelegate	视频渲染回调（仅硬解支持）。

TRTC 相关接口

通过以下接口，可以把点播播放器的音视频流通过 TRTC 进行推送，更多 TRTC 服务请参见 TRTC 产品概述。

API	描述
attachTRTC	点播绑定到 TRTC 服务。
detachTRTC	点播解绑 TRTC 服务。
publishVideo	开始推送视频流。
unpublishVideo	取消推送视频流。
publishAudio	开始推送音频流。
unpublishAudio	取消推送音频流。

类方法

API	描述
getEncryptedPlayKey	获取加固加密播放密钥。
isSupportPictureInPicture	是否支持 Picture In Picture功能（“画中画”功能）。

TXVodPlayListener

腾讯云点播回调通知。

SDK 基础回调

API	描述

onPlayEvent	点播播放事件通知，请参见 播放事件列表、事件参数。
onNetStatus	点播播放器 网络状态通知。
onPlayer:pictureInPictureStateDidChange:withParam:	画中画状态回调。
onPlayer:pictureInPictureErrorDidOccur:withParam:	画中画错误状态回调。

TXVodPlayConfig

点播播放器配置类。

基础配置接口

API	描述
connectRetryCount	设置播放器重连次数。
connectRetryInterval	设置播放器重连间隔，单位秒。
timeout	设置播放器连接超时时间，单位秒。
cacheFolderPath	此接口已废弃，推荐使用TXPlayerGlobalSetting##setCacheFolderPath。 设置点播缓存目录，点播 MP4、HLS 有效。
maxCacheItems	此接口已废弃，推荐使用TXPlayerGlobalSetting#setMaxCacheSizeMB。 设置缓存文件个数。
playerType	设置播放器类型： PLAYER_AVPLAYER：使用系统自带播放器。 PLAYER_THUMB_PLAYER：使用腾讯云自研播放器。
headers	设置自定义 HTTP headers。
enableAccurateSeek	设置是否精确 seek，默认 true。
autoRotate	播放 MP4 文件时，若设为 YES 则根据文件中的旋转角度自动旋转。 旋转角度可在 PLAY_EVT_CHANGE_ROTATION 事件中获得。默认 YES。
smoothSwitchBitrate	平滑切换多码率 HLS，默认 false。
progressInterval	设置进度回调间隔，单位毫秒。
maxBufferSize	最大预加载大小，单位 MB。

maxPreloadSize	设置预加载最大缓冲大小，单位：MB。
firstStartPlayBufferTime	设置首缓需要加载的数据时长，单位ms，默认值为100ms。
nextStartPlayBufferTime	缓冲时（缓冲数据不够引起的二次缓冲，或者seek引起的拖动缓冲）最少要缓存多长的数据才能结束缓冲，单位ms，默认值为250ms。
overlayKey	设置 HLS 安全加固加解密 key。
overlayIv	设置 HLS 安全加固加解密 Iv。
extInfoMap	设置拓展信息。
preferredResolution	播放 HLS 有多条码流时，根据设定的 preferredResolution 选最优的码流进行起播*，preferredResolution 是宽高的乘积（width * height）， 启播前设置才有效。
enableRenderProcess	是否允许加载后渲染后处理服务（如超分插件服务），默认开启。
playerPixelFormatType	视频渲染对象回调的视频格式。
keepLastFrameWhenStop	stopPlay 的时候是否保留最后一帧画面，默认值为 NO。
mediaType	设置媒资类型。可选值有： MEDIA_TYPE_AUTO，AUTO 类型（默认值，自适应码率播放暂不支持）。 MEDIA_TYPE_HLS_VOD，HLS 点播媒资。 MEDIA_TYPE_HLS_LIVE，HLS 直播媒资。 MEDIA_TYPE_HLS_VOD，MP4 等通用文件点播媒资（从 11.2 版本开始支持）。 MEDIA_TYPE_DASH_VOD，DASH 点播媒资（从 11.2 版本开始支持）。

TXPlayerGlobalSetting

点播播放器全局配置类

API	描述
setCacheFolderPath	设置播放引擎的cache目录。设置后，预下载，播放器等会优先从此目录读取和存储
setMaxCacheSize	设置播放引擎的最大缓存大小。设置后会根据设定值自动清理Cache目录的文件。单位MB。

TXVodPreloadManager

点播播放器预下载接口类

API	描述
sharedManager	获取 TXVodPreloadManager 实例对象，单例模式。
startPreload	启动预下载前，请先设置好播放引擎的缓存目录 TXPlayerGlobalSetting#setCacheFolderPath 和缓存大小 TXPlayerGlobalSetting#setMaxCacheSize。
stopPreload	停止预下载

TXVodPreloadManagerDelegate

API	描述
onComplete:url:	下载完成回调。
onError:url:error:	下载错误回调。

TXVodDownloadManager

点播播放器视频下载接口类

TXVodDownloadDataSource

API	描述
auth	fileid 信息。
quality	下载清晰度，默认原画。
token	如地址有加密，请填写 token。
templateName	清晰度模板。
fileId	文件Id。
pSign	签名信息。
appId	应用 appId。必填。
userName	账户名称。
overlayKey	HLS EXT-X-KEY 加解密参数。

overlayIv	加解密参数 overlayIv。

TXVodDownloadMediaInfo

API	描述
isDownloadFinished	是否下载完成。
dataSource	fileid 下载对象（可选）。
url	下载地址。
userName	账户名称 。
duration	时长。
playableDuration	可播放时长。
size	获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。 **注意**：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大小，暂时无法获取。
downloadSize	已下载大小，单位：byte。
segments	分段总数。
downloadSegments	已下载的分段数。
progress	进度。
playPath	播放路径，可传给 TXVodPlayer 播放。
speed	下载速度，byte 每秒。
downloadState	下载状态。
isResourceBroken	判断下载后的视频资源是否损坏，如下载完被删除等情况将返回 YES。（11.0 版本开始支持）

TXVodDownloadManager

API	描述
shareInstance	获取 TXVodDownloadManager 实例对象，单例模式。
setDownloadPath	设置下载根目录。
headers	设置下载 HTTP 头。

delegate	设置下载回调方法，下载前必须设好。
startDownloadUrl	以 URL 方式开始下载。
startDownload	以 FileID 方式开始下载。
stopDownload	停止下载，ITXVodDownloadListener.onDownloadStop 回调时停止成功。
deleteDownloadFile	删除下载文件。
deleteDownloadMediaInfo	删除下载信息。
getDownloadMediaInfoList	获取所有用户的下载列表信息。
getDownloadMediaInfo	获取下载信息。
getOverlayKeyIv	获取 HLS EXT-X-KEY。
genRandomHexStringForHls	获取加密随机数。
encryptHexStringHls:	加密。

TXVodDownloadDelegate

腾讯云视频下载回调通知。

API	描述
onDownloadStart	下载开始。
onDownloadProgress	下载进度更新。
onDownloadStop	下载停止。
onDownloadFinish	下载结束。
onDownloadError	下载过程中遇到错误。
hlsKeyVerify	下载 HLS，遇到加密的文件，将解密 Key 给外部校验。

TXDownloadError

下载错误码。

枚举值	含义说明
TXDownloadSuccess	下载成功。
TXDownloadAuthFaild	fileid 鉴权失败。

TXDownloadNoFile	无此清晰度文件。
TXDownloadFormatError	格式不支持。
TXDownloadDisconnet	网络断开。
TXDownloadHlsKeyError	获取 HLS 解密 key 失败。
TXDownloadPathError	下载目录访问失败。

TXVodDownloadMediaInfoState

下载状态

API	含义说明
TXVodDownloadMediaInfoStateInit	下载初始态。
TXVodDownloadMediaInfoStateStart	下载开始。
TXVodDownloadMediaInfoStateStop	下载停止。
TXVodDownloadMediaInfoStateError	下载出错。
TXVodDownloadMediaInfoStateFinish	下载完成。

TXVodQuality

下载视频的清晰度。

说明：

TXVodQuality240P ~ TXVodQuality1080p 常量在 11.0 版本新增。

枚举值	含义说明
TXVodQualityOD	原画。
TXVodQualityFLU	流畅。
TXVodQualitySD	标清。
TXVodQualityHD	高清。
TXVodQualityFHD	全高清。
TXVodQuality2K	2K。
TXVodQuality4K	4K。

TXVodQuality240P	流畅240P。
TXVodQuality360P	流畅360P。
TXVodQuality480P	标清480P。
TXVodQuality540P	标清540P。
TXVodQuality720P	高清720P。
TXVodQuality1080p	高清720P。

TXPlayerAuthParams

通过fileId播放加密视频配置。

API	描述
appId	应用 appId。
fileId	文件 id。
timeout	加密链接超时时间戳。
exper	试看时长。
us	唯一标识请求。
sign	防盗链签名。
https	是否用 https 请求。

TXBitrateItem

HLS多码率信息。

API	描述
index	m3u8 文件中的序号。
width	此流的视频宽度。
height	此流的视频高度。
bitrate	此流的视频码率。

TXImageSprite

雪碧图解析工具。

API	描述
setVTTUrl	设置雪碧图地址。
getThumbnail	获取缩略图。

TXPlayerDrmBuilder

点播Drm构造器

API	描述
certificateUrl	证书提供商 URL。
keyLicenseUrl	解密 key URL。
playUrl	播放链接。

错误码表

常规事件

code	事件定义	含义说明
2004	PLAY_EVT_PLAY_BEGIN	视频播放开始（若有转圈圈效果，此时将停止）。
2005	PLAY_EVT_PLAY_PROGRESS	视频播放进度，会通知当前播放进度、加载进度和总体时长。
2007	PLAY_EVT_PLAY_LOADING	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件。
2014	PLAY_EVT_VOD_LOADING_END	视频播放 loading 结束，视频继续播放。
2006	PLAY_EVT_PLAY_END	视频播放结束。
2013	PLAY_EVT_VOD_PLAY_PREPARED	播放器已准备完成，可以播放。
2003	PLAY_EVT_RCV_FIRST_I_FRAME	网络接收到首个可渲染的视频数据包（IDR）。

2009	PLAY_EVT_CHANGE_RESOLUTION	视频分辨率改变。
2011	PLAY_EVT_CHANGE_ROTATION	MP4 视频旋转角度。

警告事件

code	事件定义	含义说明
-2301	PLAY_ERR_NET_DISCONNECT	网络断连，且经多次重连亦不能恢复,更多重试请自行重启播放。
-2305	PLAY_ERR_HLS_KEY	HLS 解密 key 获取失败。
2101	PLAY_WARNING_VIDEO_DECODE_FAIL	当前视频帧解码失败。
2102	PLAY_WARNING_AUDIO_DECODE_FAIL	当前音频帧解码失败。
2103	PLAY_WARNING_RECONNECT	网络断连, 已启动自动重连（重连超过三次就直接抛送 PLAY_ERR_NET_DISCONNECT）。
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解。
-2304	PLAY_ERR_HEVC_DECODE_FAIL	H265 解码失败。
-2303	PLAY_ERR_FILE_NOT_FOUND	播放的文件不存在。
-5	VOD_PLAY_ERR_LICENCE_CHECK_FAIL	License 不合法，播放失败。 注意：在startVodPlay之前，需要通过TXLiveBase#setLicence设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。

播放器 SDK 常量

事件码和错误码定义

枚举值	含义
VOD_PLAY_EVT_RCV_FIRST_I_FRAME	播放事件：成功接收到第一个视频帧。
VOD_PLAY_EVT_RCV_FIRST_AUDIO_FRAME	播放事件：成功接收到第一个音频帧。
VOD_PLAY_EVT_PLAY_BEGIN	播放事件：播放已经开始。
VOD_PLAY_EVT_PLAY_PROGRESS	播放事件：播放进度更新，点播播放器（VodPlayer）专用。

VOD_PLAY_EVT_PLAY_END	播放事件：播放已经结束。
VOD_PLAY_EVT_PLAY_LOADING	播放事件：数据缓冲中。
VOD_PLAY_EVT_START_VIDEO_DECODER	播放事件：视频解码器已经启动。
VOD_PLAY_EVT_CHANGE_RESOLUTION	播放事件：视频分辨率发生变化。
VOD_PLAY_EVT_GET_PLAYINFO_SUCC	播放事件：成功获取到点播文件的信息，点播播放器（VodPlayer）专用。
VOD_PLAY_EVT_CHANGE_ROTATION	播放事件：MP4 视频的旋转角度发生变化，点播播放器（VodPlayer）专用。
VOD_PLAY_EVT_VOD_PLAY_PREPARED	播放事件：视频加载完毕，点播播放器（VodPlayer）专用。
VOD_PLAY_EVT_VOD_LOADING_END	播放事件：视频缓冲结束，点播播放器（VodPlayer）专用。
VOD_PLAY_EVT_STREAM_SWITCH_SUCC	播放事件：已经成功完成切流（在不同清晰度的视频流之间进行切换）。
VOD_PLAY_EVT_VOD_PLAY_TCP_CONNECT_SUCC	TCP 连接成功。
VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET	收到首帧数据。
VOD_PLAY_EVT_VOD_PLAY_SEEK_COMPLETE	视频播放 Seek 完成。
VOD_PLAY_EVT_AUDIO_SESSION_INTERRUPT	播放事件: Audio Session 被其他 App 中断（仅适用于 iOS 平台）。
VOD_PLAY_ERR_NET_DISCONNECT	直播错误：网络连接断开（已经经过三次重试并且未能重连成功）。
VOD_PLAY_ERR_FILE_NOT_FOUND	点播错误：播放文件不存在。
VOD_PLAY_ERR_HLS_KEY	点播错误：HLS 解码 KEY 获取失败。
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	点播错误：获取点播文件的文件信息失败。

画中画错误类型

枚举值	含义
TX_VOD_PLAYER_PIP_ERROR_TYPE_NONE	无错误。

TX_VOD_PLAYER_PIP_ERROR_TYPE_DEVICE_NOT_SUPPORT	设备或系统版本不支持（iPad iOS9+ 才支持 PIP）。
TX_VOD_PLAYER_PIP_ERROR_TYPE_PLAYER_NOT_SUPPORT	播放器不支持。
TX_VOD_PLAYER_PIP_ERROR_TYPE_VIDEO_NOT_SUPPORT	视频不支持。
TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_IS_NOT_POSSIBLE	PIP 控制器不可用。
TX_VOD_PLAYER_PIP_ERROR_TYPE_ERROR_FROM_SYSTEM	PIP 控制器报错。
TX_VOD_PLAYER_PIP_ERROR_TYPE_PLAYER_NOT_EXIST	播放器对象不存在。
TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_IS_RUNNING	PIP 功能已经运行。
TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_NOT_RUNNING	PIP 功能没有启动。

画中画控制器状态

枚举值	含义
TX_VOD_PLAYER_PIP_STATE_UNDEFINED	未设置状态。
TX_VOD_PLAYER_PIP_STATE_WILL_START	画中画即将开始。
TX_VOD_PLAYER_PIP_STATE_DID_START	画中画已经开始。
TX_VOD_PLAYER_PIP_STATE_WILL_STOP	画中画即将结束。
TX_VOD_PLAYER_PIP_STATE_DID_STOP	画中画已经结束。
TX_VOD_PLAYER_PIP_STATE_RESTORE_UI	重置 UI。

Android

最近更新时间：2024-04-11 16:11:38

TXVodPlayer

点播播放器

请参见 TXVodPlayer。
主要负责从指定的点播流地址拉取音视频数据，并进行解码和本地渲染播放。
播放器包含如下能力：
支持 FLV、MP4 及 HLS 多种播放格式，支持 基础播放（URL 播放） 和 点播播放（Fileid 播放）两种播放方式 。
屏幕截图，可以截取当前播放流的视频画面。
通过手势操作，调节亮度、声音、进度等。
可以手动切换不同的清晰度，也可根据网络带宽自适应选择清晰度。
可以指定不同倍速播放，并开启镜像和硬件加速。
完整能力，请参见 点播超级播放器 - 能力清单。

播放器配置接口

API	描述
setConfig	设置播放器配置信息，配置信息请参见 TXVodPlayConfig 。
setPlayerView	设置播放器的视频渲染 TXCloudVideoView。
setPlayerView	设置播放器的视频渲染 TextureView。
setSurface	设置播放器的视频渲染 SurfaceView。
setStringOption	设置播放器业务参数，参数格式为 `<String,Object>` 。
setSubtitleStyle	设置字幕样式信息，可在播放后对字幕样式进行更新（播放器高级版本才支持）。
setSubtitleView	设置字幕软解目标对象（播放器高级版本才支持）。
setAudioNormalization	设置音量均衡，响度范围：-70～0(LUFS) 播放器高级版 11.7 版本开始支持。 可填预设值（相关类或文件：Android：TXVodConstants；iOS：TXVodPlayConfig.h） 关：AUDIO_NORMALIZATION_OFF 开：AUDIO_NORMALIZATION_STANDARD（标准） AUDIO_NORMALIZATION_LOW （低） AUDIO_NORMALIZATION_HIGH（高）

	可填自定义数值： 从低到高，范围-70 - 0 LUFS

播放基础接口

API	描述
startVodPlay	播放 HTTP URL 形式地址。10.7 版本开始，`startPlay` 变更为 `startVodPlay`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用。
startVodPlay	以 fileId 形式播放，传入TXPlayInfoParams 参数。10.7 版本开始，`startPlay` 变更为 `startVodPlay`，需要通过 `V2TXLivePremier#setLicence` 或者 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用。
stopPlay	停止播放。
isPlaying	是否正在播放。
pause	暂停播放，停止获取流数据,保留最后一帧画面。
resume	恢复播放，重新获取流数据。
seek	跳转到视频流指定时间点，单位秒。
seek	跳转到视频流指定时间点，单位毫秒。
getCurrentPlaybackTime	获取当前播放位置，单位秒。
getBufferDuration	获取缓存的总时长，单位秒。
getDuration	获取总时长，单位秒。
getPlayableDuration	获取可播放时长，单位秒。
getWidth	获取视频宽度。
getHeight	获取视频高度。
setAutoPlay	设置点播是否 startVodPlay 后自动开始播放，默认自动播放。
setStartTime	设置播放开始时间。
setToken	加密 HLS 的 token。

setLoop	设置是否循环播放。
isLoop	返回是否循环播放状态。
addSubtitleSource	添加外挂字幕（播放器高级版本才支持）。
getSubtitleTrackInfo	返回字幕轨道信息列表（播放器高级版本才支持）。
getAudioTrackInfo	返回音频轨道信息列表（播放器高级版本才支持）。
selectTrack	选择轨道（播放器高级版本才支持）。
deselectTrack	取消选择轨道（播放器高级版本才支持）。
seekToPdtTime	跳转到视频流指定 PDT（Program Date Time ）时间点，可实现视频快进、快退、进度条跳转等功能，目前只支持 HLS 视频格式。（播放器高级版 11.6 版本开始支持）。 参数单位毫秒(ms)。

视频相关接口

API	描述
enableHardwareDecode	启用或禁用视频硬解码。
snapshot	获取当前视频帧图像。 **注意：由于获取当前帧图像是比较耗时的操作，所以截图会通过异步回调出来。**
setMirror	设置镜像。
setRate	设置点播的播放速率，默认1.0。
getBitrateIndex	返回当前播放的码率索引。
setBitrateIndex	设置当前正在播放的码率索引，无缝切换清晰度。清晰度切换可能需要等待一小段时间。
setRenderMode	设置 图像平铺模式。
setRenderRotation	设置 图像渲染角度。

音频相关接口

API	描述
setMute	设置是否静音播放。

setAudioPlayoutVolume	设置音量大小，范围：0 - 100。
setRequestAudioFocus	设置是否自动获取音频焦点 默认自动获取。

事件通知接口

API	描述
setPlayListener	设置播放器的回调（已弃用，建议使用 setVodListener）。
setVodListener	设置播放器的回调。

TRTC 相关接口

通过以下接口，可以把点播播放器的音视频流通过 TRTC 进行推送，更多 TRTC 服务请参见 TRTC 产品概述。

API	描述
attachTRTC	点播绑定到 TRTC 服务。
detachTRTC	点播解绑 TRTC 服务。
publishVideo	开始推送视频流。
unpublishVideo	取消推送视频流。
publishAudio	开始推送音频流。
unpublishAudio	取消推送音频流。

ITXVodPlayListener

腾讯云点播回调通知。

SDK 基础回调

API	描述
onPlayEvent	点播播放事件通知，请参见 播放事件列表、事件参数。
onNetStatus	点播播放器 网络状态通知。

TXVodPlayConfig

点播播放器配置类。

基础配置接口

API	描述
setConnectRetryCount	设置播放器重连次数。
setConnectRetryInterval	设置播放器重连间隔，单位秒。
setTimeout	设置播放器连接超时时间，单位秒。
setCacheFolderPath	设置点播缓存目录，点播 MP4、HLS 有效。
setMaxCacheItems	设置缓存文件个数。接口废弃，请使用TXPlayerGlobalSetting#setMaxCacheSize 进行全局配置。
setPlayerType	设置播放器类型： TXVodConstants#PLAYER_SYSTEM_MEDIA_PLAYER：使用系统自带播放器。 TXVodConstants#PLAYER_THUMB_PLAYER：使用腾讯云自研播放器。
setHeaders	设置自定义 HTTP headers。
setEnableAccurateSeek	设置是否精确 seek，默认 true。
setAutoRotate	播放 MP4 文件时，若设为 YES 则根据文件中的旋转角度自动旋转。 旋转角度可在 PLAY_EVT_CHANGE_ROTATION 事件中获得。默认 YES。
setSmoothSwitchBitrate	平滑切换多码率 HLS，默认 false。
setCacheMp4ExtName	缓存 MP4 文件扩展名。
setProgressInterval	设置进度回调间隔，单位毫秒。
setMaxBufferSize	最大预加载大小，单位 MB。
setMaxPreloadSize	设置预加载最大缓冲大小，单位：MB。
setExtInfo	设置拓展信息。
setPreferredResolution	播放 HLS 有多条码流时，根据设定的 preferredResolution 选最优的码流进行起播，preferredResolution 是宽高的乘积（width * height）， 启播前设置才有效。
setOverlayKey	设置 HLS 安全加固加解密 key。
setOverlayIv	设置 HLS 安全加固加解密 Iv。
setEnableRenderProcess	是否允许加载后渲染后处理服务。

setMediaType	设置媒资类型, 默认为 auto 类型。可选值有： TXVodConstants#MEDIA_TYPE_AUTO，AUTO类型（默认值，自适应码率播放暂不支持）； TXVodConstants#MEDIA_TYPE_HLS_VOD，HLS点播媒资； TXVodConstants#MEDIA_TYPE_HLS_LIVE，HLS直播媒资； TXVodConstants#MEDIA_TYPE_HLS_VOD，MP4等通用文件点播媒资（从 11.2 版本开始支持）； TXVodConstants#MEDIA_TYPE_DASH_VOD，DASH点播媒资（从 11.2 版本开始支持）；

TXPlayerGlobalSetting

点播播放器全局配置类

API	描述
setCacheFolderPath	设置播放引擎的 cache 目录。设置后，离线下载，预下载，播放器等会优先从此目录读取和存储。
setMaxCacheSize	设置播放引擎的最大缓存大小。设置后会根据设定值自动清理Cache目录的文件，单位 MB。
setDrmProvisionEnv	设置 Drm 证书提供商环境（注意：从 11.2 版本开始支持）。可选值有： TXPlayerGlobalSetting.DrmProvisionEnv#DRM_PROVISION_ENV_COM， 代表使用 google COM 域名证书提供商； TXPlayerGlobalSetting.DrmProvisionEnv#DRM_PROVISION_ENV_CN，代表使用 google CN 域名证书提供商；

TXVodPreloadManager

点播播放器预下载接口类

API	描述
getInstance	获取 TXVodPreloadManager 实例对象，单例模式。
startPreload	启动预下载前，请先设置好播放引擎的缓存目录。
stopPreload	停止预下载。

ITXVodPreloadListener

视频预下载回调接口。

API	描述
onComplete	视频预下载完成。
onError	视频预下载出错。

TXVodDownloadManager

点播播放器视频下载接口类。当前只支持下载非嵌套 m3u8 视频源，对 simpleAES 加密视频源将进行腾讯云私有加密算法加密以提升安全性。

API	描述
getInstance	获取 TXVodDownloadManager 实例对象，单例模式。
setHeaders	设置下载 HTTP 头。
setListener	设置下载回调方法，下载前必须设好。
startDownloadUrl	以 URL 方式开始下载。
startDownload	以 fileid 方式开始下载。
stopDownload	停止下载，ITXVodDownloadListener.onDownloadStop 回调时停止成功。
deleteDownloadMediaInfo	删除下载信息。
getDownloadMediaInfoList	获取所有用户的下载列表信息。
getDownloadMediaInfo	获取下载信息。
getDownloadMediaInfo	获取下载信息。

ITXVodDownloadListener

腾讯云视频下载回调通知。

API	描述
onDownloadStart	下载开始。
onDownloadProgress	下载进度更新。

onDownloadStop	下载停止。
onDownloadFinish	下载结束。
onDownloadError	下载过程中遇到错误。
hlsKeyVerify	下载 HLS，遇到加密的文件，将解密 key 给外部校验。

TXVodDownloadDataSource

腾讯云视频fildid下载源，下载时作为参数传入。

API	描述
TXVodDownloadDataSource	构造函数，传入 appid，fileid，quality，pSign，username 等参数。
getAppId	获取传入的 appid。
getFileId	获取传入的 fileid。
getPSign	获取传入的 psign。
getQuality	获取传入的 quality。
getUserName	获取传入的 userName，默认“default”。
getToken	获取传入的 token。
getOverlayKey	获取传入的 overlayKey。
getOverlayIv	获取传入的 overlayIv。

清晰度常量

说明：

TXVodQuality240P ~ TXVodQuality1080p 常量在 11.0 版本新增

code	常量定义	含义说明
0	QUALITY_OD	原画
1	QUALITY_FLU	流畅
2	QUALITY_SD	标清
3	QUALITY_HD	高清

4	QUALITY_FHD	全高清
5	QUALITY_2K	2K
6	QUALITY_4K	4K
1000	QUALITY_UNK	未定义
240	TXVodQuality240P	流畅 240P
360	TXVodQuality360P	流畅 360P
480	TXVodQuality480P	标清 480P
540	TXVodQuality540P	标清 540P
720	TXVodQuality720P	高清 720P
1080	TXVodQuality1080p	全高清 1080P

TXVodDownloadMediaInfo

腾讯云视频下载信息，可获取下载进度，播放链接等信息。

API	描述
getDataSource	fileid 下载时获取传入的 fileid 下载源。
getDuration	获取下载的总时长。
getPlayableDuration	获取已下载的可播放时长。
getSize	获取下载文件总大小，单位：Byte，只针对 fileid 下载源有效。 备注：总大小是指上传到腾讯云点播控制台的原始文件的大小，转自适应码流后的子流大小，暂时无法获取。
getDownloadSize	获取已下载文件大小，只针对 fileid 下载源有效。
getProgress	获取当前下载进度。
getPlayPath	获取当前播放路径，可传给 TXVodPlayer 播放。
getDownloadState	获取下载状态。
isDownloadFinished	判断是否下载完成。
getSpeed	获取下载速度，单位：KByte/秒。（10.9 版本开始支持）

isResourceBroken	判断下载后的视频资源是否损坏，如下载完被删除等情况将返回true。（11.0 版本开始支持）

静态属性

code	属性定义	含义说明
0	STATE_INIT	下载初始态
1	STATE_START	下载开始
2	STATE_STOP	下载停止
3	STATE_ERROR	下载出错
4	STATE_FINISH	下载完成

TXPlayerAuthBuilder

通过fileId播放加密视频配置。

API	描述
setAppId	应用 appId。
setFileId	文件 id。
setTimeout	加密链接超时时间戳。
setExper	试看时长。
setUs	唯一标识请求。
setSign	防盗链签名。
setHttps	是否用 https 请求。

TXBitrateItem

视频码率信息。

API	描述
index	m3u8 文件中的序号。

width	此流的视频宽度。
height	此流的视频高度。
bitrate	此流的视频码率。
compareTo	比较两条流的码率是否相等。

TXImageSprite

雪碧图解析类。

API	描述
TXImageSprite	构造函数。
setVTTUrlAndImageUrls	设置雪碧图地址。
getThumbnail	获取缩略图。
release	使用完毕调用，否则会造成内存泄漏。

TXPlayerDrmBuilder

DRM 播放信息。

API	描述
TXPlayerDrmBuilder	构造 DRM 播放信息对象。
setDeviceCertificateUrl	设置证书提供商 url。
setKeyLicenseUrl	设置解密 key url。
setPlayUrl	设置播放媒体的 url。

TXPlayInfoParams

通过 fileId 播放视频参数信息。

API	描述
TXPlayInfoParams	构造函数。

getAppId	获取应用 id。
getFileId	获取文件 id。
getPSign	获取腾讯云视频加密签名。

错误码表

常规事件

code	事件定义	含义说明
2004	PLAY_EVT_PLAY_BEGIN	视频播放开始（若有转菊花效果，此时将停止）。
2005	PLAY_EVT_PLAY_PROGRESS	视频播放进度，会通知当前播放进度、加载进度和总体时长。
2007	PLAY_EVT_PLAY_LOADING	视频播放 loading，如果能够恢复，之后会有 LOADING_END 事件。
2014	PLAY_EVT_VOD_LOADING_END	视频播放 loading 结束，视频继续播放。
2006	PLAY_EVT_PLAY_END	视频播放结束。
2013	PLAY_EVT_VOD_PLAY_PREPARED	播放器已准备完成，可以播放。
2003	PLAY_EVT_RCV_FIRST_I_FRAME	网络接收到首个可渲染的视频数据包（IDR）。
2009	PLAY_EVT_CHANGE_RESOLUTION	视频分辨率改变。
2011	PLAY_EVT_CHANGE_ROTATION	MP4 视频旋转角度。

警告事件

code	事件定义	含义说明
-2301	PLAY_ERR_NET_DISCONNECT	网络断连，且经多次重连亦不能恢复，更多重试请自行重启播放。
-2305	PLAY_ERR_HLS_KEY	HLS 解密 key 获取失败。
2101	PLAY_WARNING_VIDEO_DECODE_FAIL	当前视频帧解码失败。
2102	PLAY_WARNING_AUDIO_DECODE_FAIL	当前音频帧解码失败。
2103	PLAY_WARNING_RECONNECT	网络断连, 已启动自动重连 (重连超过三次就直

		接抛送 PLAY_ERR_NET_DISCONNECT)。
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解。
-2304	PLAY_ERR_HEVC_DECODE_FAIL	H265 解码失败。
-2303	PLAY_ERR_FILE_NOT_FOUND	播放的文件不存在。

播放器SDK常量

下面的常量10.0 版本开始，通过 `TXVodConstants` 对外提供：

图像平铺模式

code	事件定义	含义说明
0	RENDER_MODE_FULL_FILL_SCREEN	视频画面全屏铺满。
1	RENDER_MODE_ADJUST_RESOLUTION	视频画面自适应屏幕。

图像渲染角度

code	事件定义	含义说明
0	RENDER_ROTATION_PORTRAIT	常规竖屏。
270	RENDER_ROTATION_LANDSCAPE	右旋90度。

播放事件列表

code	事件定义	含义说明
2003	VOD_PLAY_EVT_RCV_FIRST_I_FRAME	网络接收到首个视频数据包（IDR）。
2004	VOD_PLAY_EVT_PLAY_BEGIN	视频播放开始。
2005	VOD_PLAY_EVT_PLAY_PROGRESS	视频播放进度。
2006	VOD_PLAY_EVT_PLAY_END	视频播放结束。
2007	VOD_PLAY_EVT_PLAY_LOADING	视频播放 Loading。
2008	VOD_PLAY_EVT_START_VIDEO_DECODER	解码器启动。
2009	VOD_PLAY_EVT_CHANGE_RESOLUTION	视频分辨率改变。

2010	VOD_PLAY_EVT_GET_PLAYINFO_SUCC	获取点播文件信息成功。
2011	VOD_PLAY_EVT_CHANGE_ROTATION	视频旋转信息。
2013	VOD_PLAY_EVT_VOD_PLAY_PREPARED	视频加载完毕。
2014	VOD_PLAY_EVT_VOD_LOADING_END	loading 结束。
2026	VOD_PLAY_EVT_RCV_FIRST_AUDIO_FRAME	音频首次播放。
2103	VOD_PLAY_WARNING_RECONNECT	网络断连, 已启动自动重连。
-2301	VOD_PLAY_ERR_NET_DISCONNECT	网络断连, 且经多次重连抢救无效。
-2303	VOD_PLAY_ERR_FILE_NOT_FOUND	文件不存在。
-2304	VOD_PLAY_ERR_HEVC_DECODE_FAIL	h265 解码失败。
-2305	VOD_PLAY_ERR_HLS_KEY	HLS 解密 key 获取失败。
-2306	VOD_PLAY_ERR_GET_PLAYINFO_FAIL	获取点播文件信息失败。
2106	VOD_PLAY_WARNING_HW_ACCELERATION_FAIL	硬解启动失败，采用软解。
-5	VOD_PLAY_ERR_INVALID_LICENSE	license 不合法，播放失败。注：在 `startVodPlay` 之前，需要通过 `TXLiveBase#setLicence` 设置 License 后方可成功播放，否则将播放失败（黑屏），全局仅设置一次即可。直播 License、短视频 License 和视频播放 License 均可使用，若您暂未获取上述 License，可 快速免费申请测试版 License 以正常播放，正式版 License 需购买。

播放事件参数

事件定义	含义说明
EVT_UTC_TIME	UTC 时间。
EVT_TIME	事件发生时间。
EVT_DESCRIPTION	事件说明。
EVT_PARAM1	事件参数1。
EVT_PARAM2	事件参数2。

EVT_PLAY_COVER_URL	视频封面。
EVT_PLAY_URL	视频地址。
EVT_PLAY_NAME	视频名称。
EVT_PLAY_DESCRIPTION	视频简介。
EVT_PLAY_PROGRESS_MS	播放进度（毫秒）。
EVT_PLAY_DURATION_MS	播放总长（毫秒）。
EVT_PLAY_PROGRESS	播放进度。
EVT_PLAY_DURATION	播放总长。
EVT_PLAYABLE_DURATION_MS	点播可播放时长（毫秒）。
EVT_PLAYABLE_RATE	播放速率。
EVT_PLAYABLE_DURATION	点播可播放时长。
EVT_IMAGESPRIT_WEBVTTURL	雪碧图 web vtt 描述文件下载 URL。
EVT_IMAGESPRIT_IMAGEURL_LIST	雪碧图图片下载 URL。
EVT_DRM_TYPE	加密类型。
EVT_CODEC_TYPE	视频编码类型。
EVT_KEY_FRAME_CONTENT_LIST	视频关键帧描述信息。
EVT_KEY_FRAME_TIME_LIST	关键帧时间。

播放网络状态通知参数

事件定义	含义说明
NET_STATUS_CPU_USAGE	CPU 使用率。
NET_STATUS_VIDEO_WIDTH	分辨率之 width。
NET_STATUS_VIDEO_HEIGHT	分辨率之 height。
NET_STATUS_VIDEO_FPS	当前视频帧率。
NET_STATUS_VIDEO_GOP	当前视频 GOP。

NET_STATUS_VIDEO_BITRATE	视频数据比特率。
NET_STATUS_AUDIO_BITRATE	音频数据比特率。
NET_STATUS_NET_SPEED	网络速率。
NET_STATUS_AUDIO_CACHE	音频帧数。
NET_STATUS_VIDEO_CACHE	视频帧数。
NET_STATUS_AUDIO_INFO	当前流的音频信息。
NET_STATUS_NET_JITTER	网络抖动情况。
NET_STATUS_SERVER_IP	连接的 Server IP 地址。
NET_STATUS_VIDEO_DPS	当前解码器输出帧率。
NET_STATUS_QUALITY_LEVEL	网络质量。

播放器媒资类型

code	事件定义	含义说明
0	MEDIA_TYPE_AUTO	auto 类型。
1	MEDIA_TYPE_HLS_VOD	自适应码率播放 hls 点播媒资。
2	MEDIA_TYPE_HLS_LIVE	自适应码率播放 hls 直播媒资。

未分类变量

code	事件定义	含义说明
-1	INDEX_AUTO	自适应码率 index 标识。

Flutter

最近更新时间：2024-04-26 11:09:31

SuperPlayerPlugin 类

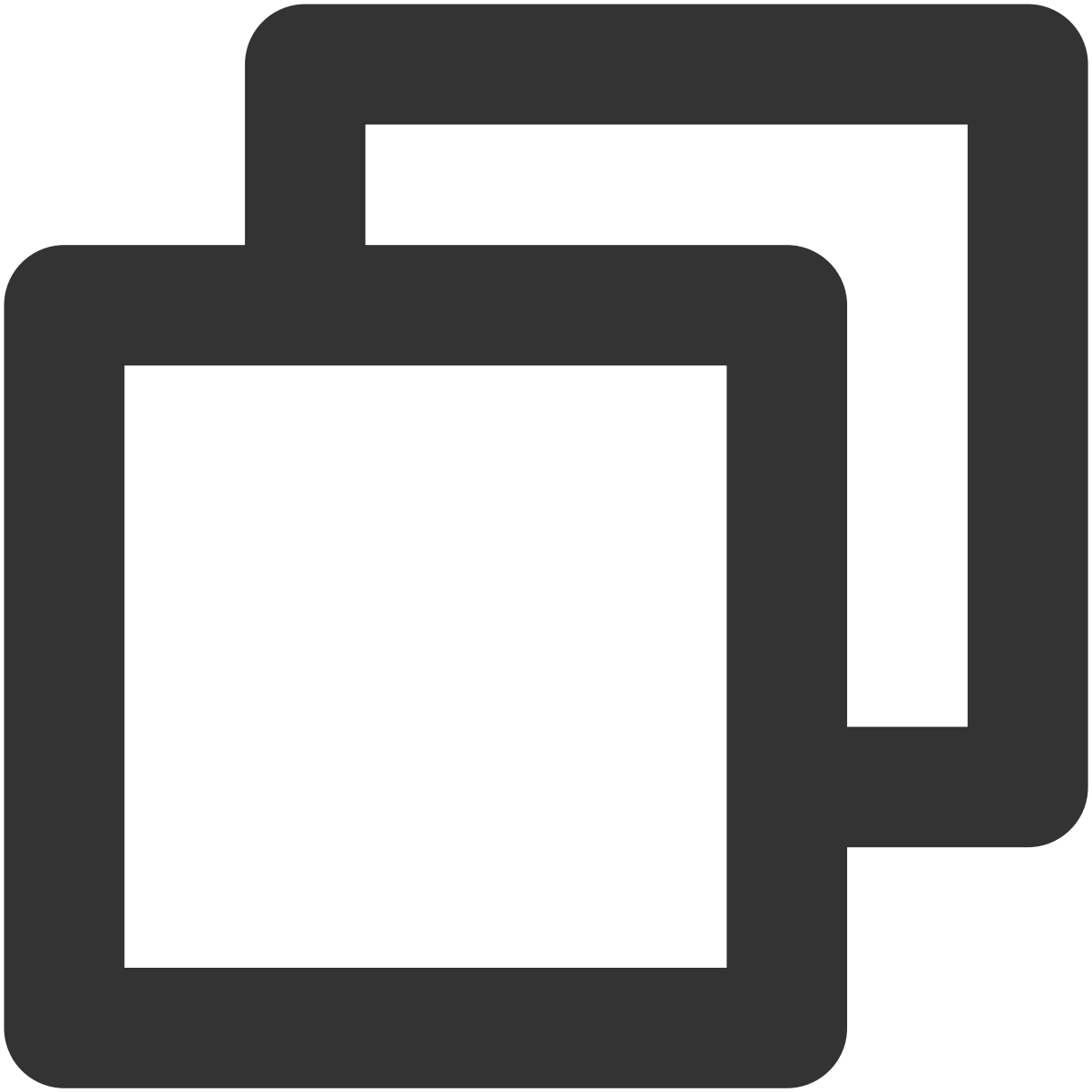
setGlobalLicense

说明

设置 License

申请到 License 后，通过下面的接口初始化 License，建议在启动的时候进行，如果没有设置 License，将会播放视频失败。

接口



```
static Future<void> setGlobalLicense(String licenceUrl, String licenceKey) async;
```

参数说明

参数名	类型	描述
licenceUrl	String	licence 的 url
licenceKey	String	licence 的 key

返回值

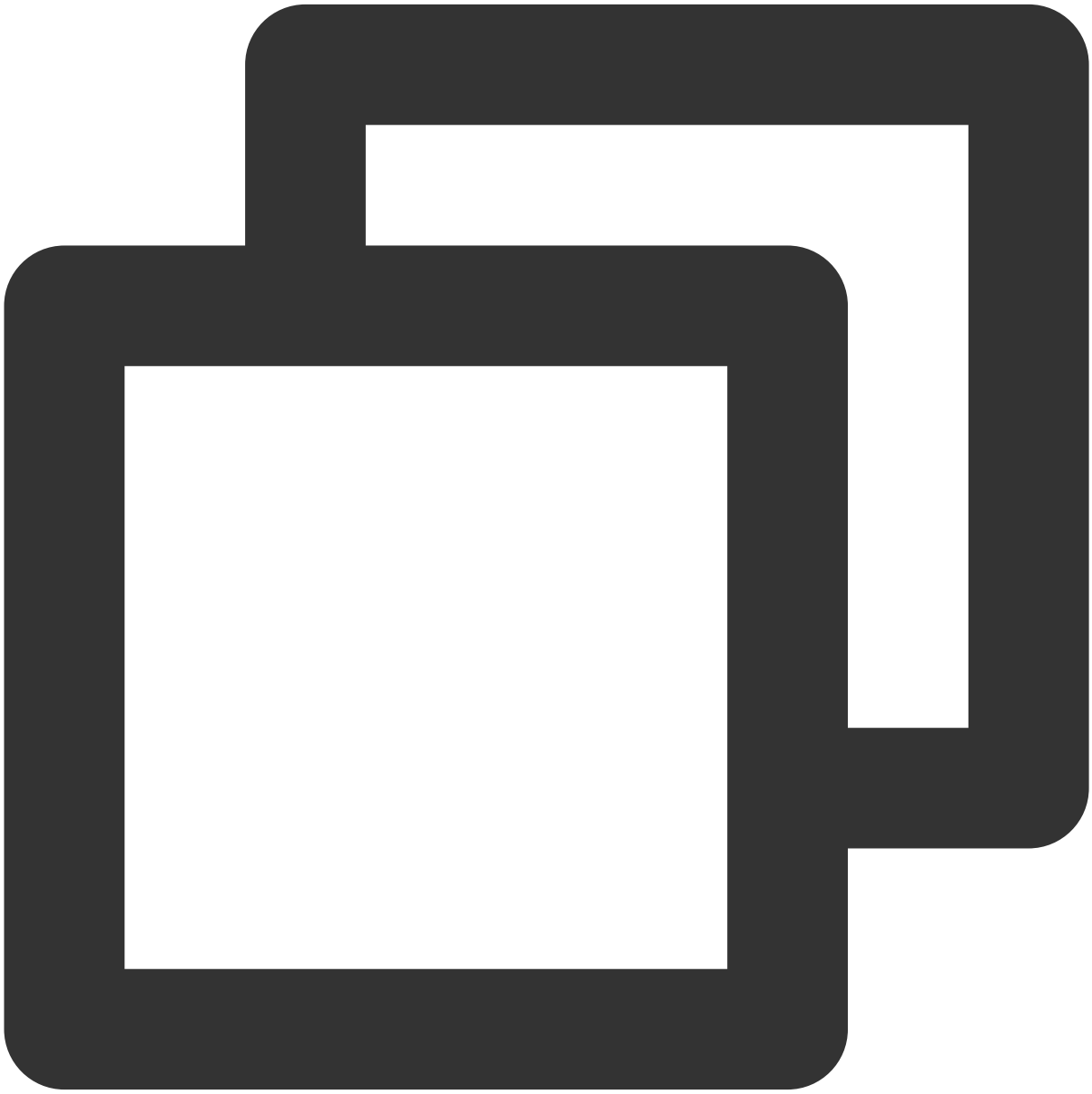
无

createVodPlayer

说明

创建原生层的点播播放器实例，如果使用 `TXVodPlayerController`，其中已经集成，不需要额外创建。

接口



```
static Future<int?> createVodPlayer() async;
```

参数说明

无

返回值

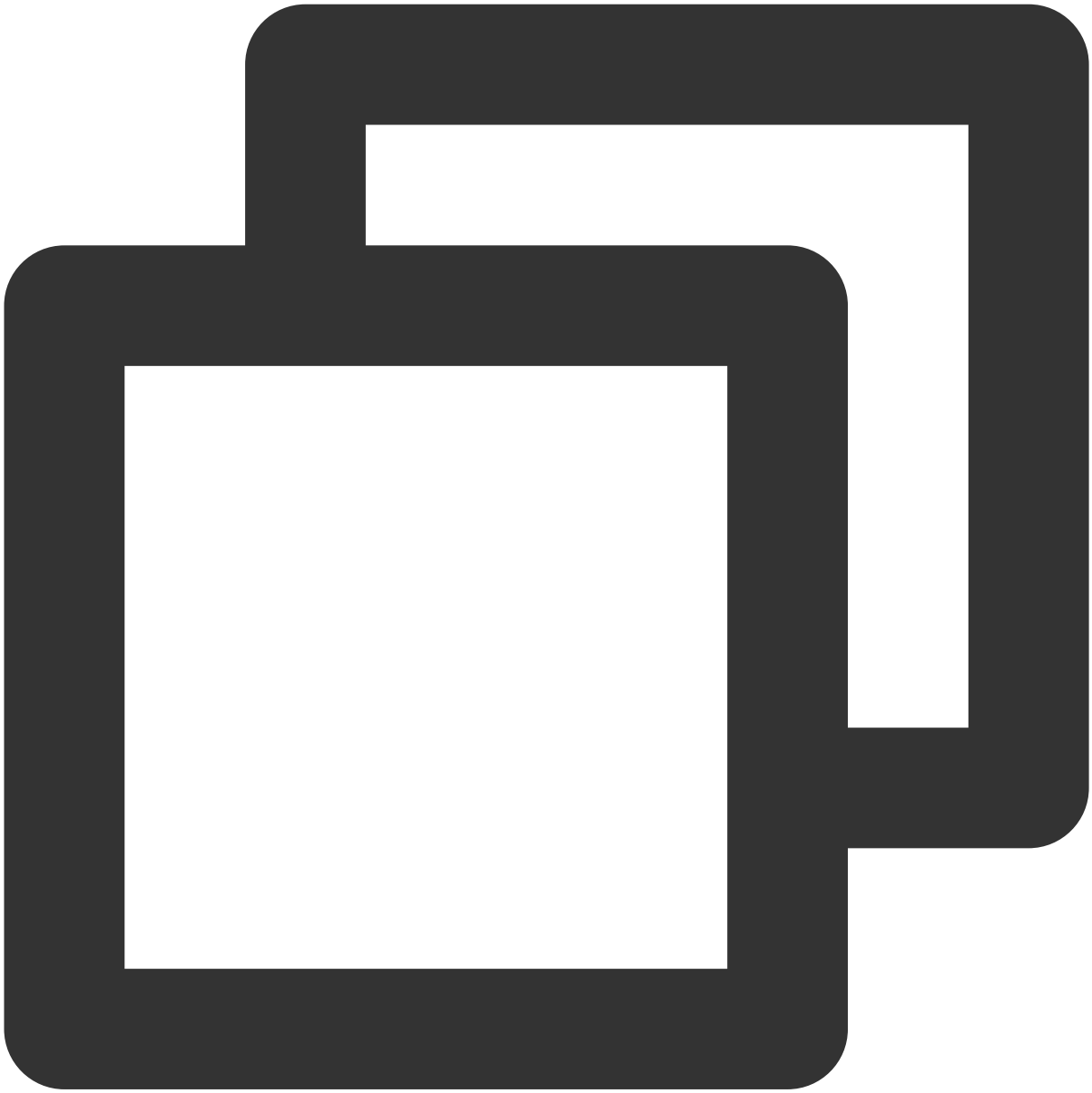
返回值	类型	描述
playerId	int	播放器ID

createLivePlayer

说明

创建原生层的点播播放器实例，如果使用 `TXVodPlayerController` ，其中已经集成，不需要额外创建。

接口



```
static Future<int?> createLivePlayer() async;
```

参数说明

无

返回值

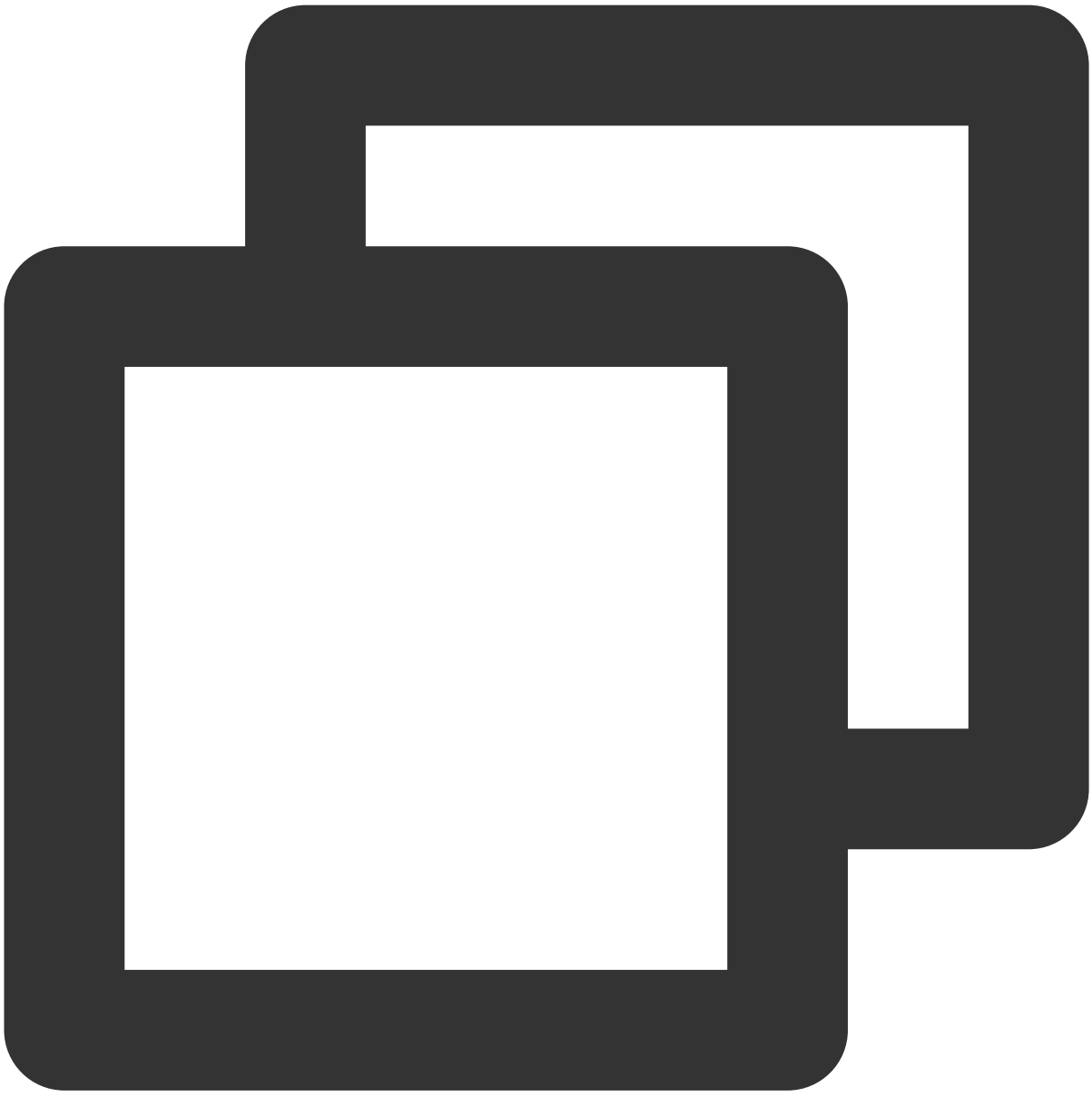
返回值	类型	描述
playerId	int	播放器ID

setConsoleEnabled

说明

打开或关闭播放器原生 log 输出。

接口



```
static Future<int?> setConsoleEnabled() async;
```

参数说明

参数名	类型	描述

enabled	bool	开启或关闭播放器 log

返回值

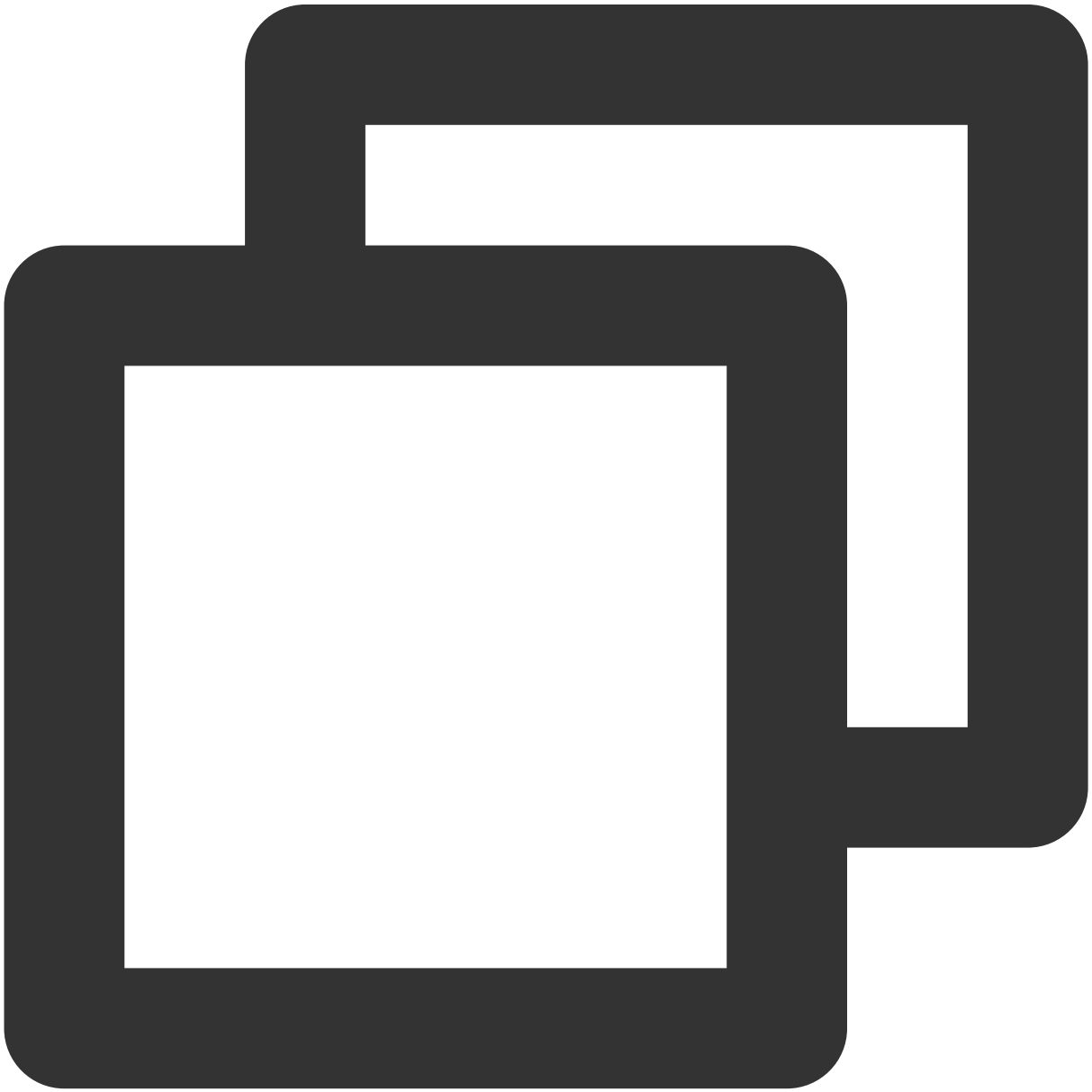
无

releasePlayer

说明

释放对应播放器的资源。

接口



```
static Future<int?> releasePlayer(int? playerId) async;
```

参数说明

无

返回值

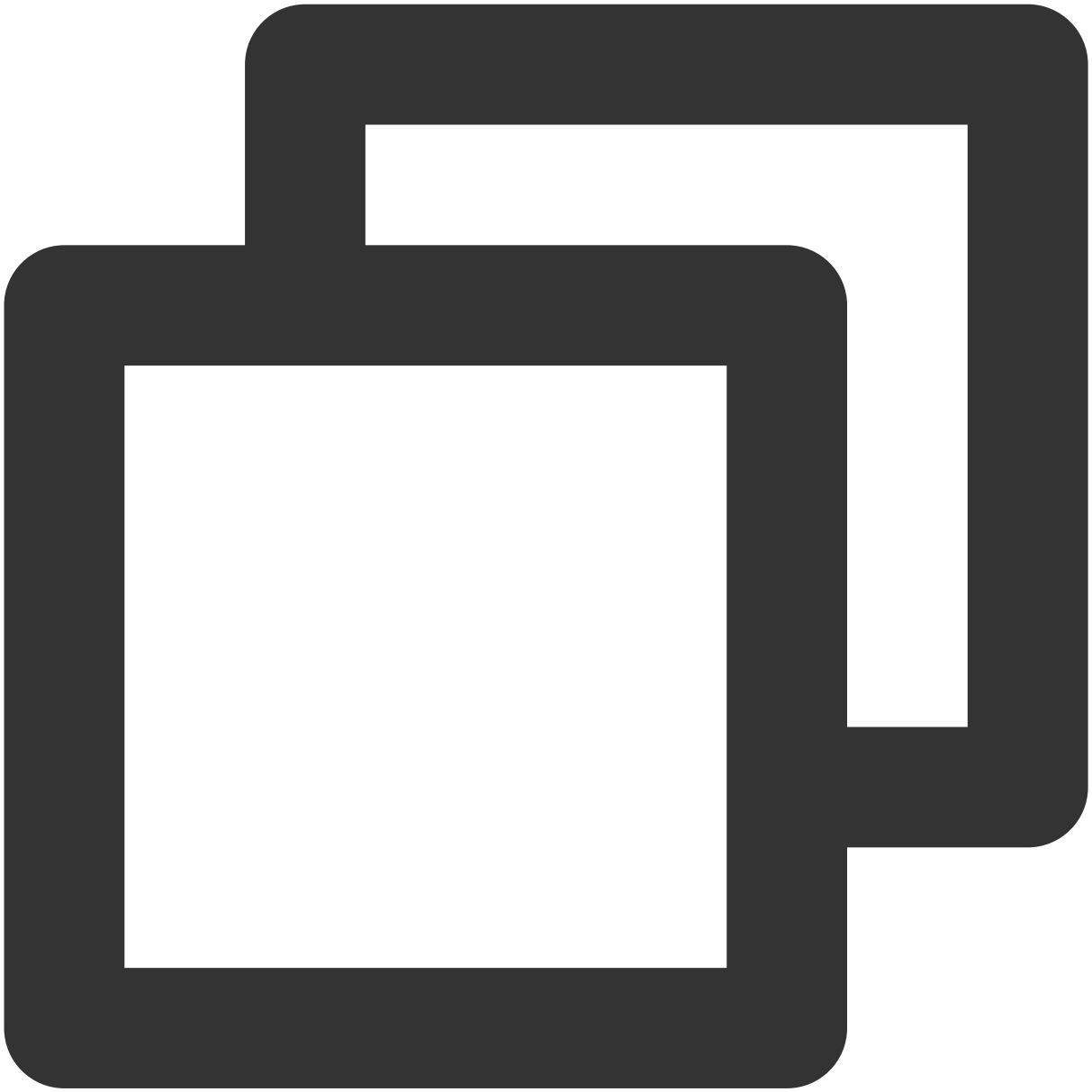
无

setGlobalMaxCacheSize

说明

设置播放引擎的最大缓存大小。设置后会根据设定值自动清理 Cache 目录的文件。

接口



```
static Future<void> setGlobalMaxCacheSize(int size) async;
```

参数说明

参数名	类型	描述
size	int	最大缓存大小（单位：MB）

返回值

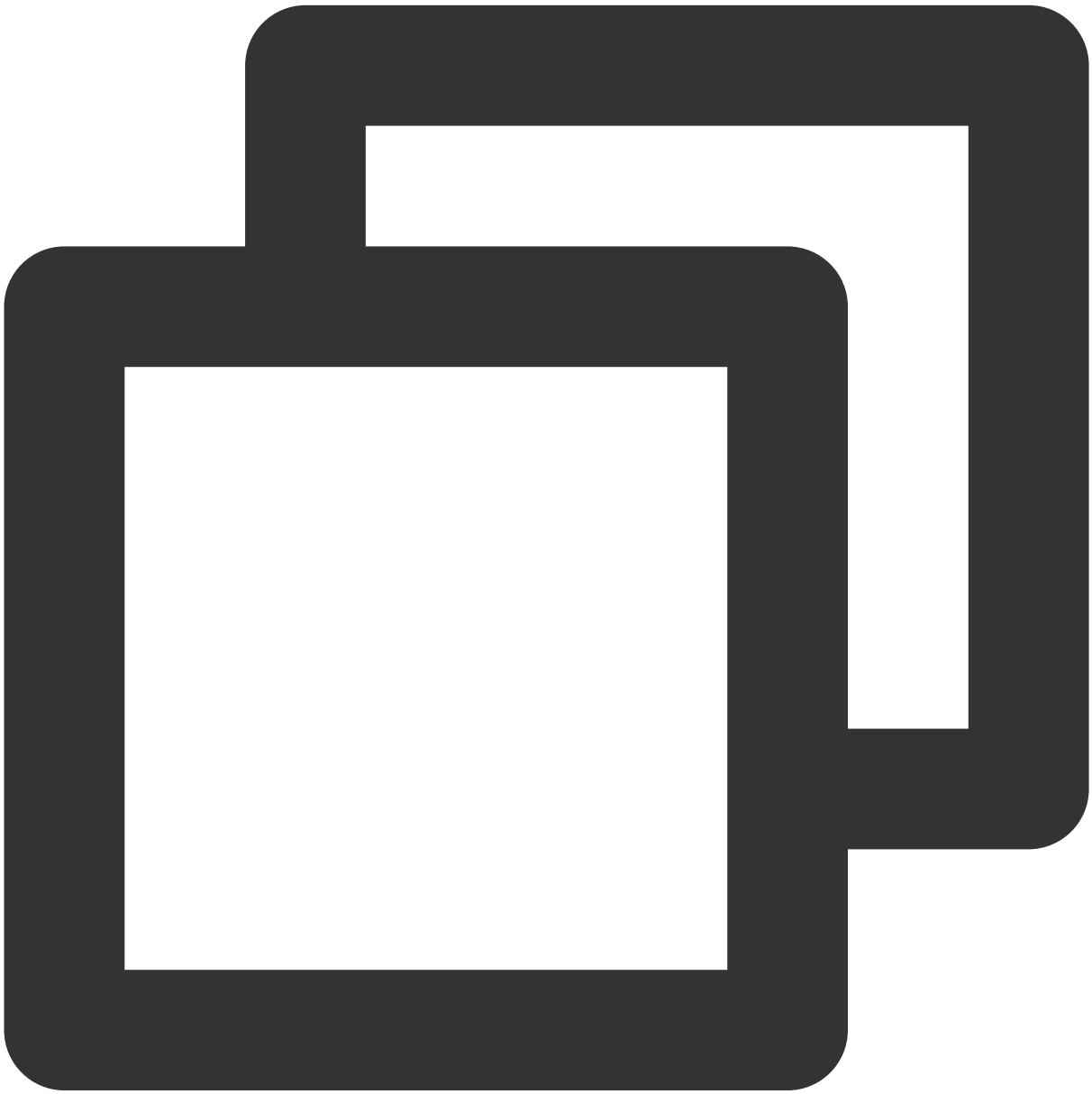
无

setGlobalCacheFolderPath

说明

该缓存路径默认设置到 App 沙盒目录下，参数只需要传递相对缓存目录即可，不需要传递整个绝对路径。

接口



```
static Future<bool> setGlobalCacheFolderPath(String postfixPath) async;
```

参数说明

参数名	类型	描述

postfixPath	String	缓存目录,该缓存路径默认设置到 app 沙盒目录下，postfixPath 只需要传递相对缓存目录即可，不需要传递整个绝对路径。Android 平台：视频将会缓存到 sdcard的 Android/data/your-pkg-name/files/testCache 目录。iOS 平台视频将会缓存到沙盒的 Documents/testCache 目录。

返回值

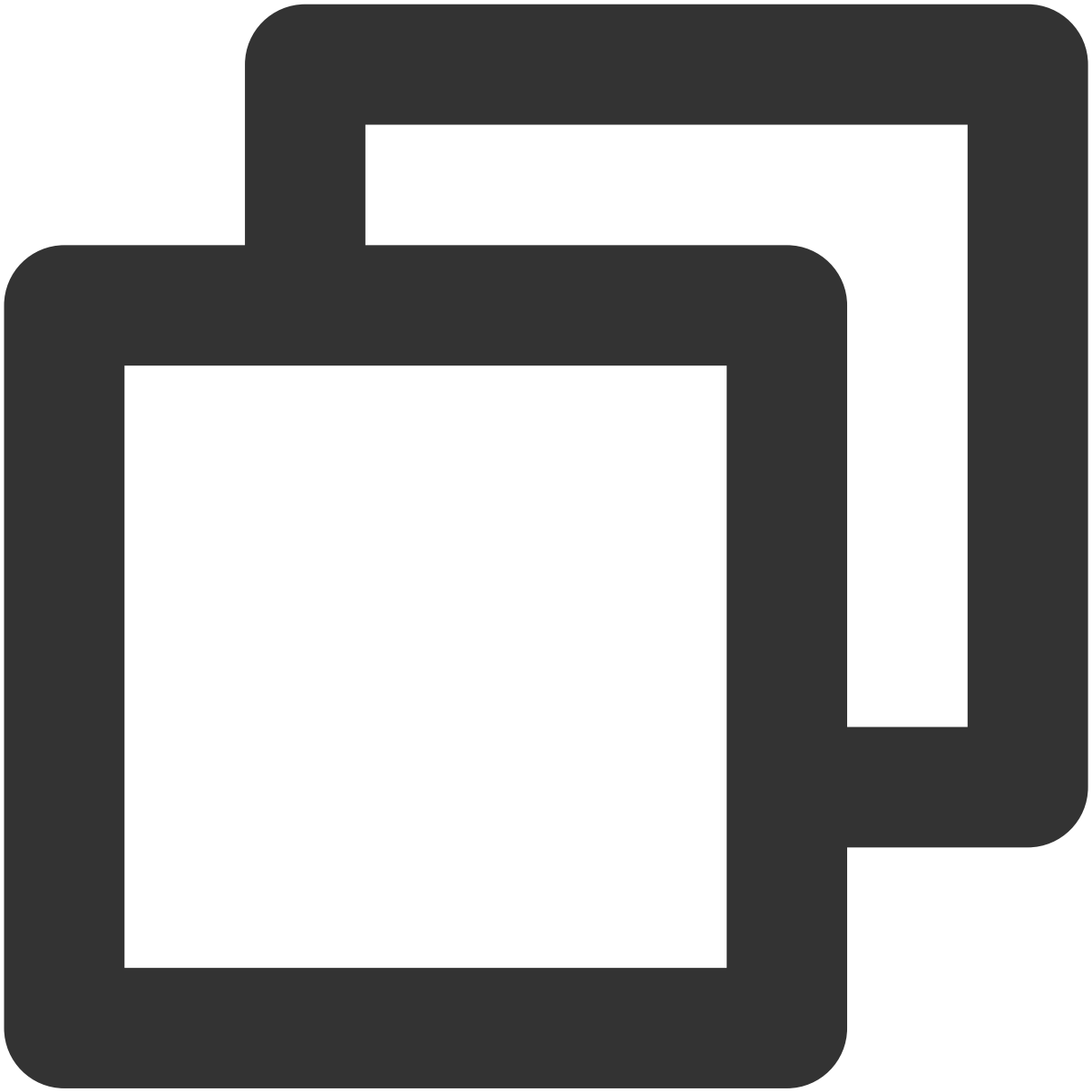
无

setLogLevel

说明

设置 log 输出级别。

接口



```
static Future<void> setLogLevel(int logLevel) async;
```

参数说明

参数名	类型	描述
logLevel	int	0:输出所有级别的 log 1:输出 DEBUG,INFO,WARNING,ERROR 和 FATAL 级别的 log 2:输出 INFO,WARNNING,ERROR 和 FATAL 级别的 log 3:输出 WARNNING,ERROR 和 FATAL 级别的log 4:输出 ERROR 和 FATAL 级别的log

		5:只输出 FATAL 级别的log 6:不输出任何 sdk log

返回值

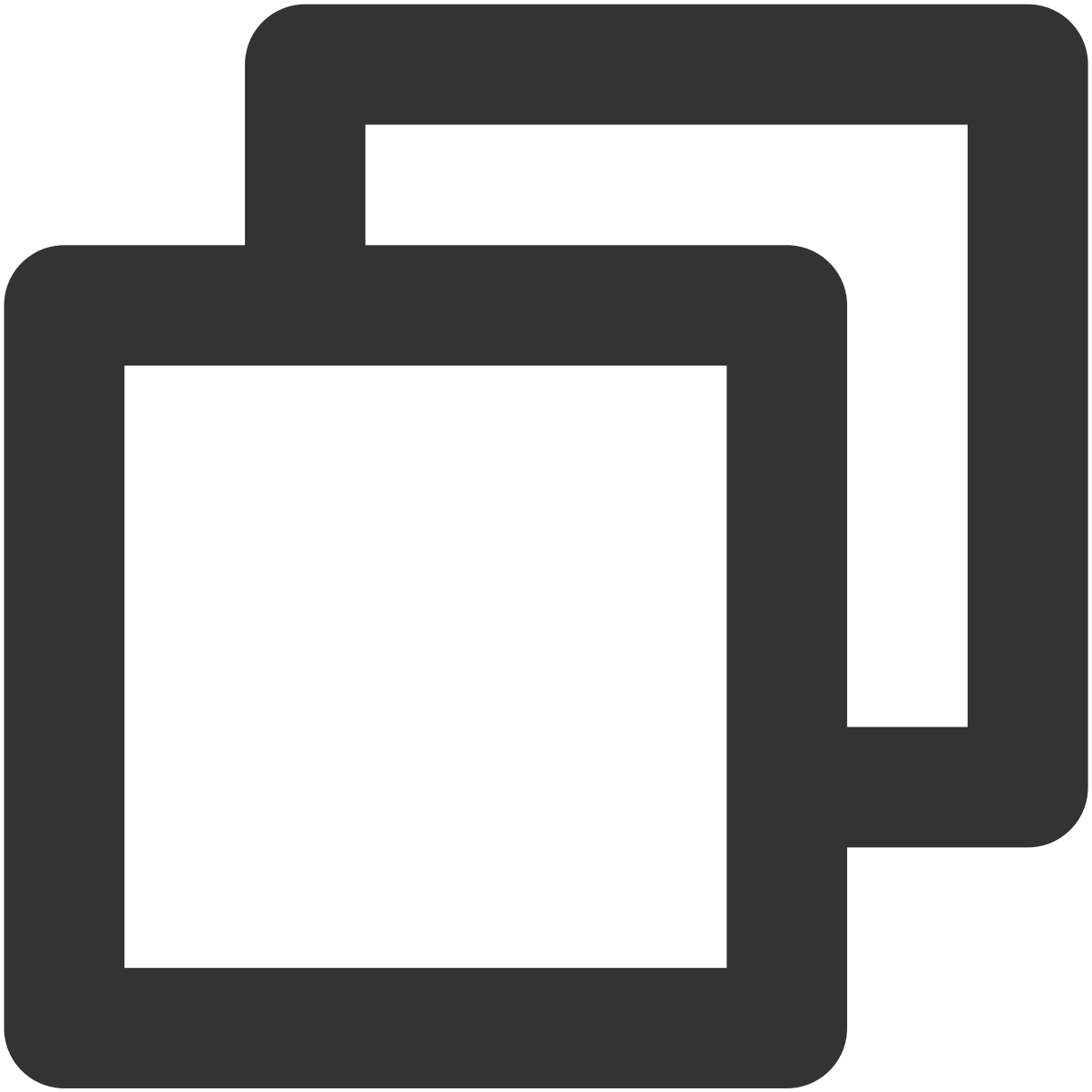
无

setBrightness

说明

设置亮度，仅适用于当前 app。

接口



```
static Future<void> setBrightness(double brightness) async;
```

参数说明

参数名	类型	描述
brightness	double	亮度取值范围 0.0~1.0

返回值

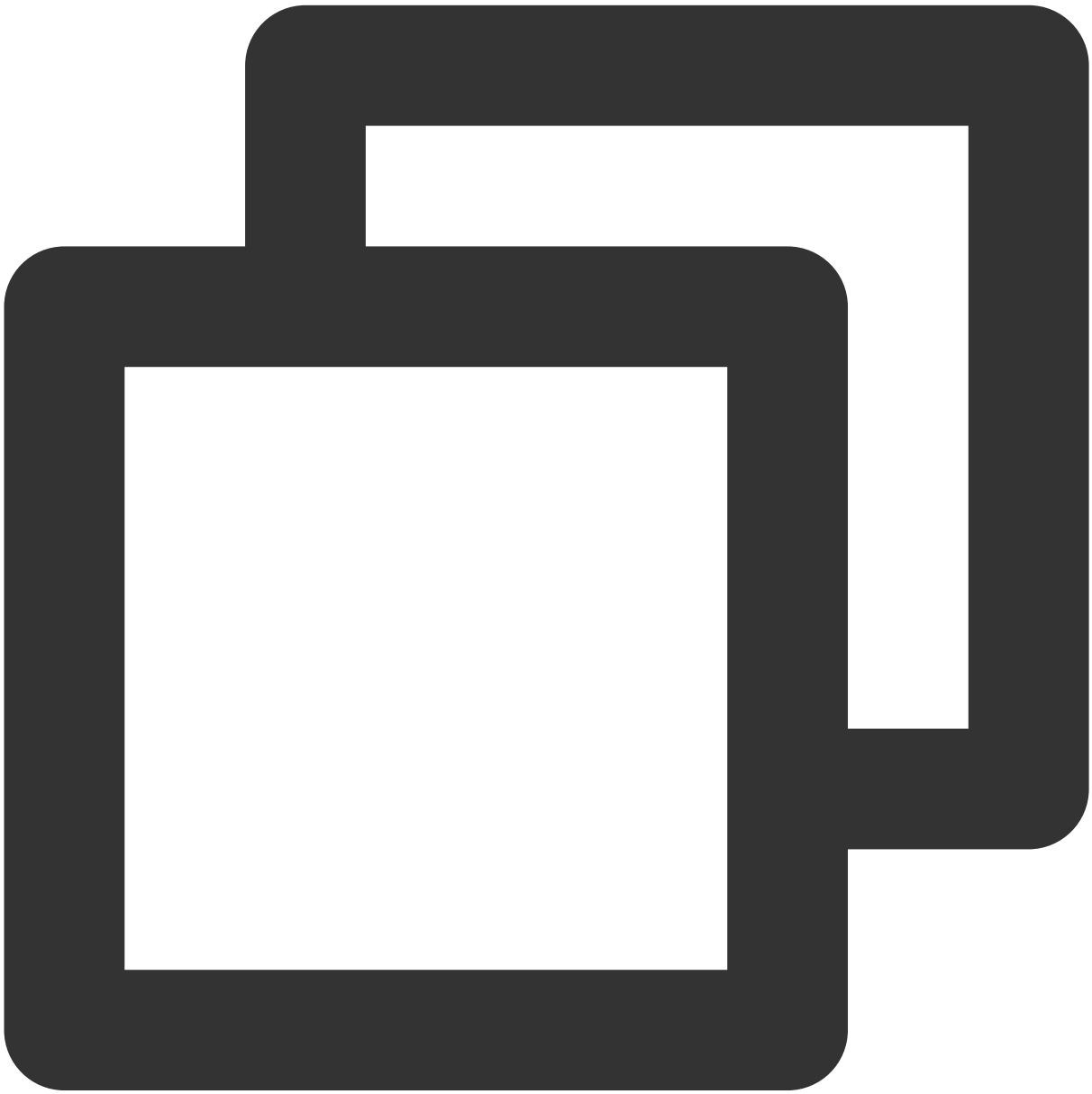
无

restorePageBrightness

说明

恢复界面亮度，仅适用于当前 app。

接口



```
static Future<void> restorePageBrightness() async;
```

参数说明

无

返回值

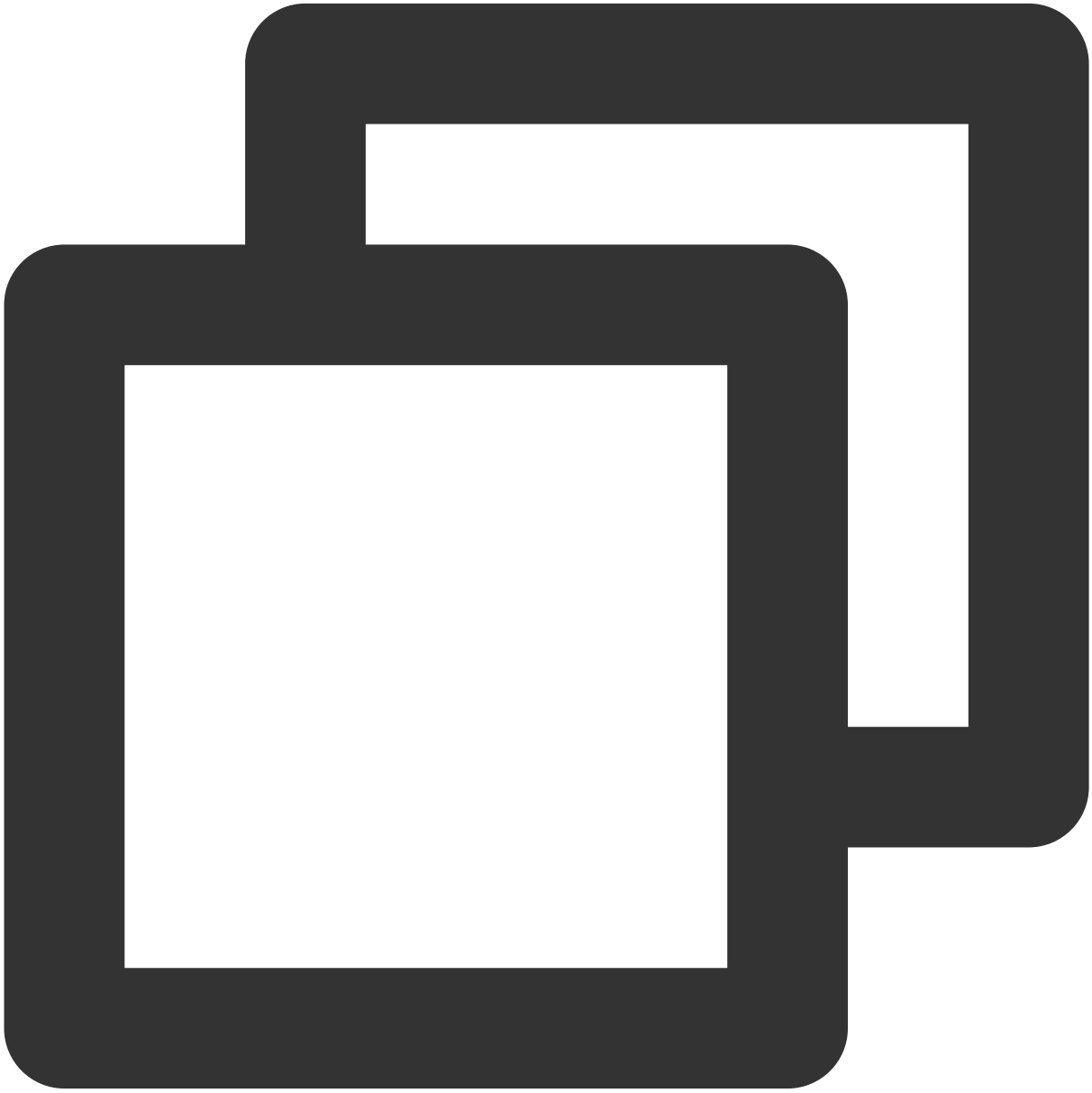
无

getBrightness

说明

获得当前界面的亮度值。

接口



```
static Future<double> getBrightness() async;
```

参数说明

无

返回值

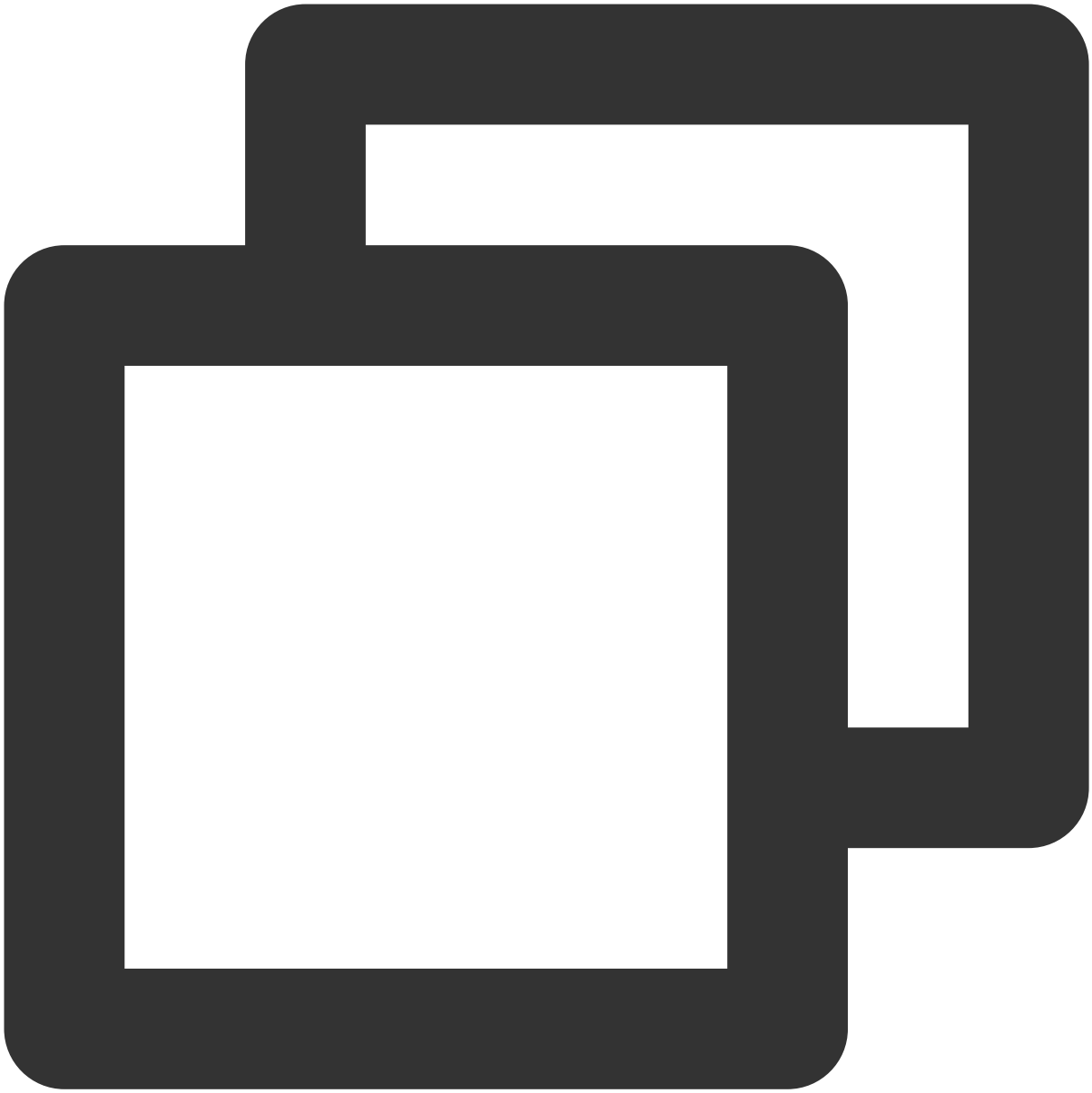
参数名	类型	描述
brightness	double	亮度取值范围 0.0~1.0

setSystemVolume

说明

设置当前系统的音量。

接口



```
static Future<void> setSystemVolume(double volume) async;
```

参数说明

参数名	类型	描述
volume	double	音量取值范围 0.0~1.0

返回值

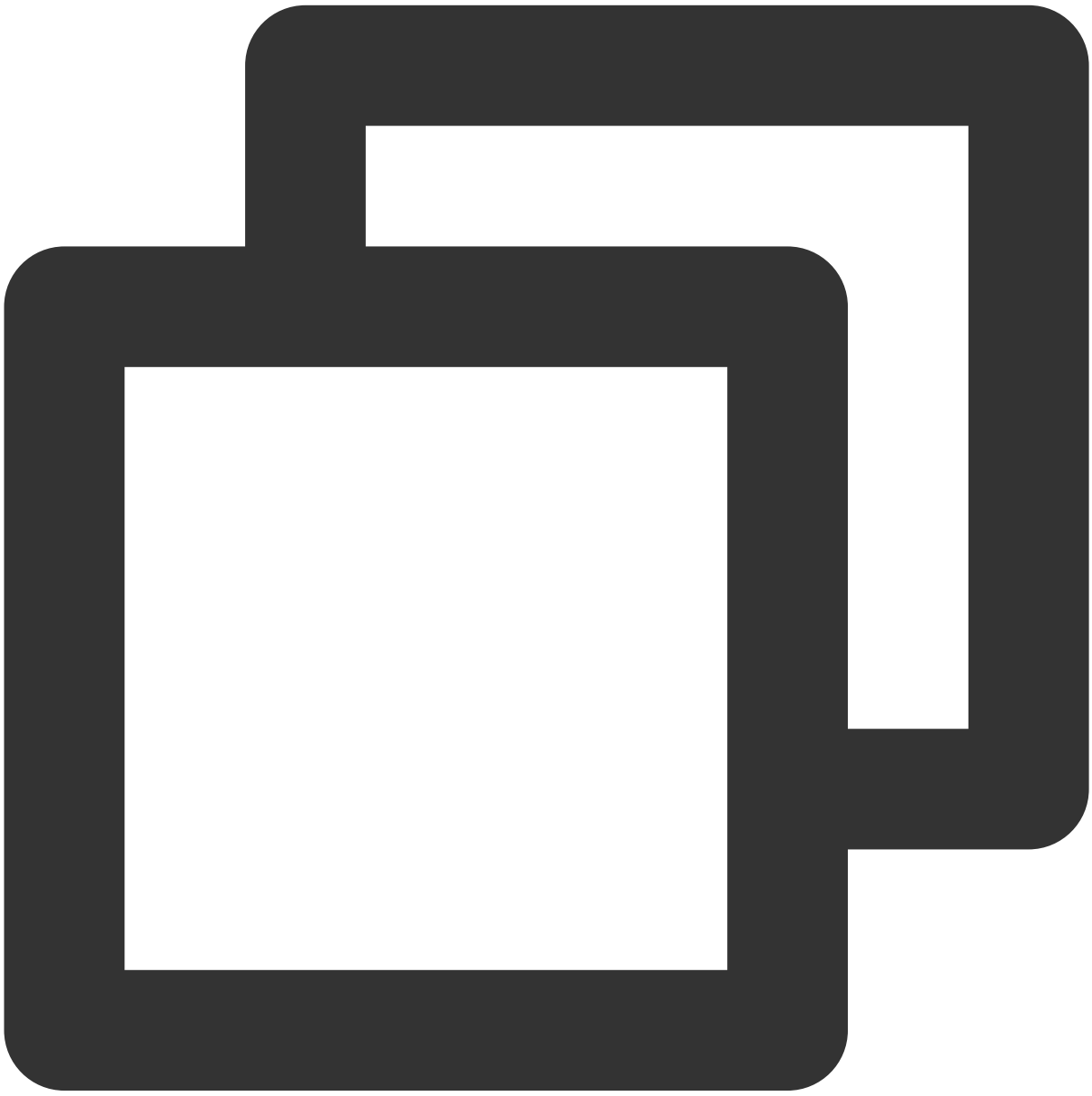
无

getSystemVolume

说明

设置当前系统的音量。

接口



```
static Future<double> getSystemVolume() async;
```

参数说明

无

返回值说明

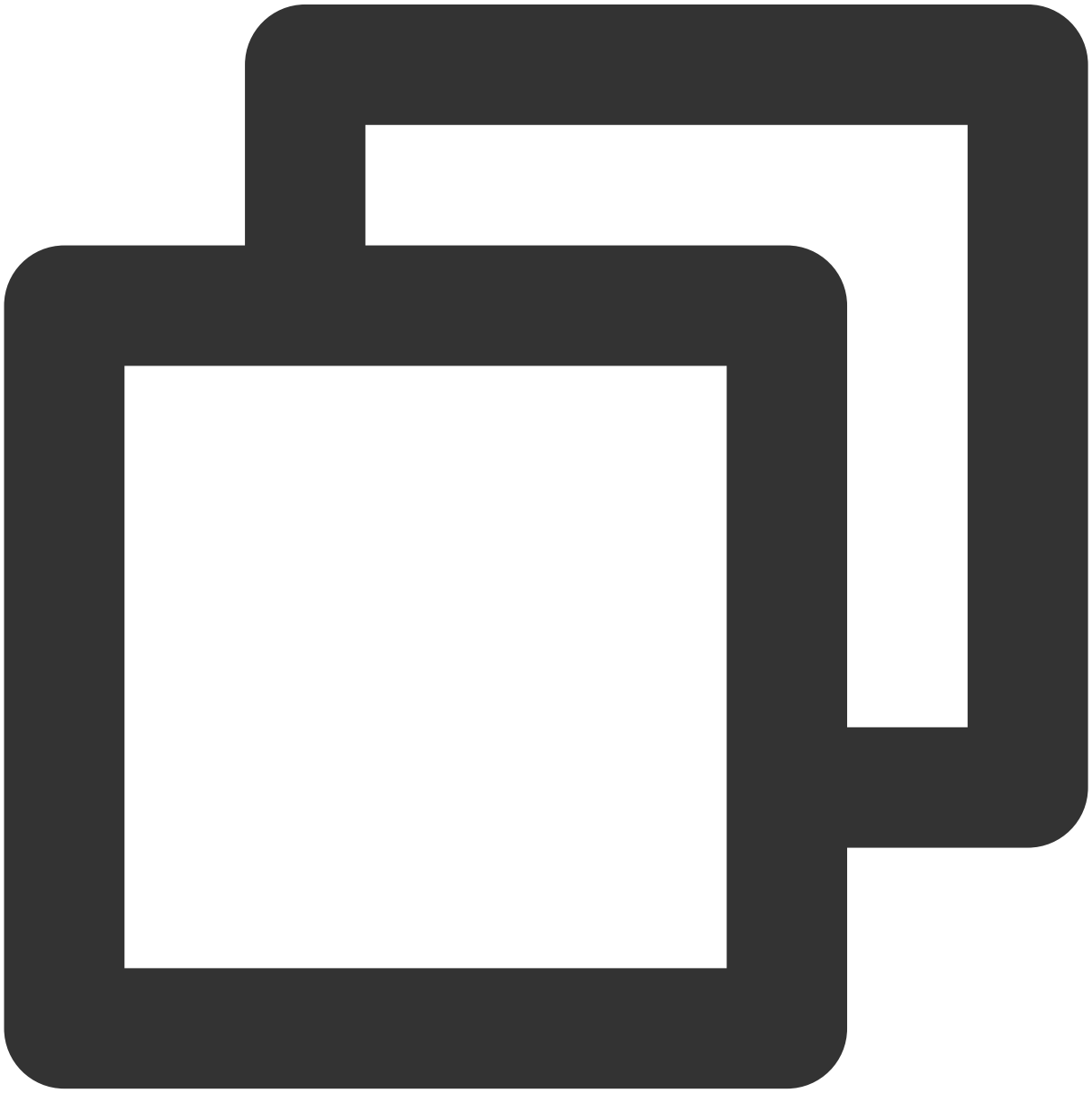
参数名	类型	描述
volume	double	音量取值范围 0.0~1.0

abandonAudioFocus

说明

释放音频焦点，仅适用于Android。

接口



```
static Future<double> abandonAudioFocus() async;
```

参数说明

无

返回值

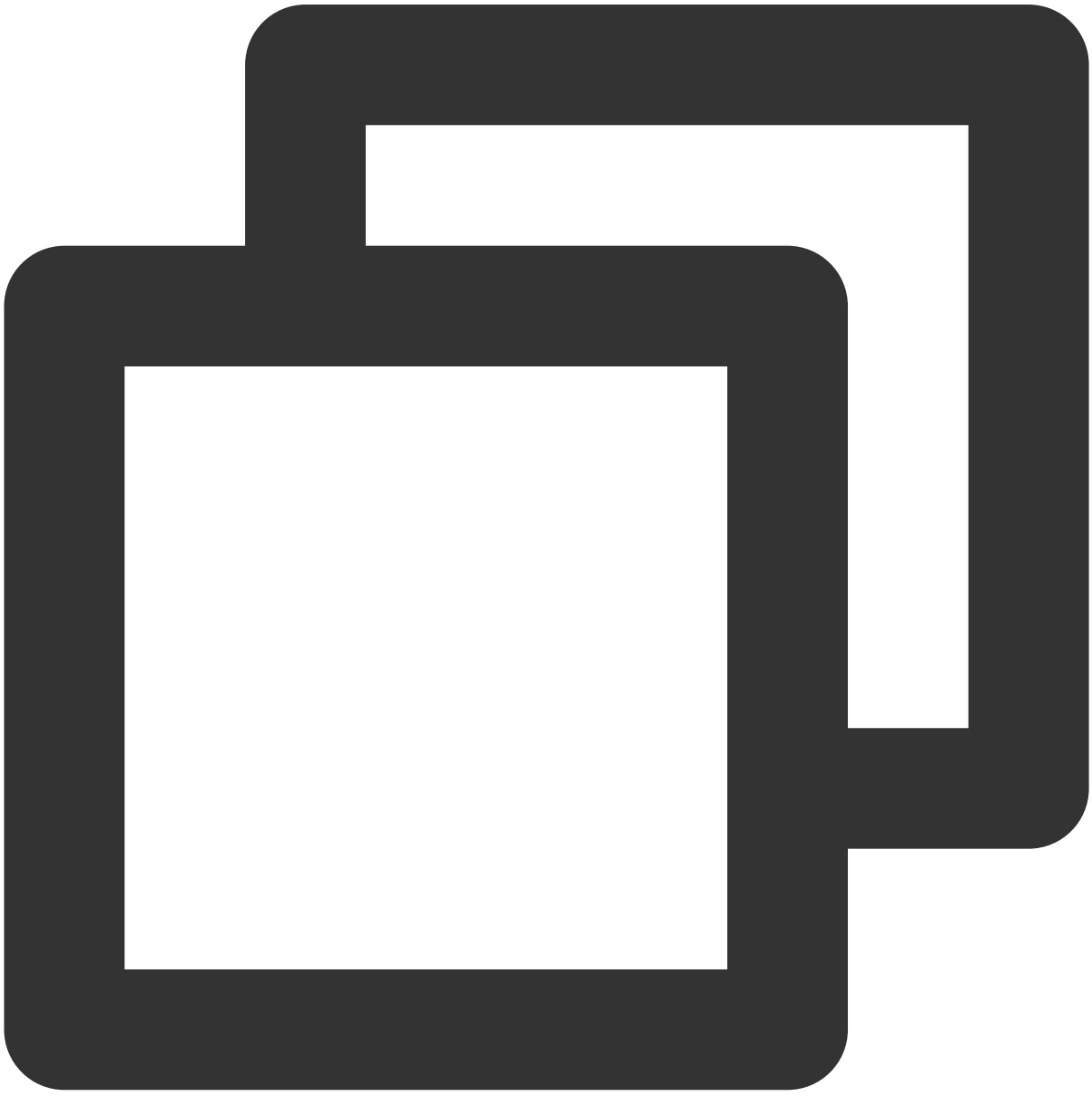
无

requestAudioFocus

说明

请求获取音频焦点，仅适用于Android。

接口



```
static Future<void> requestAudioFocus() async ;
```

参数说明

无

返回值

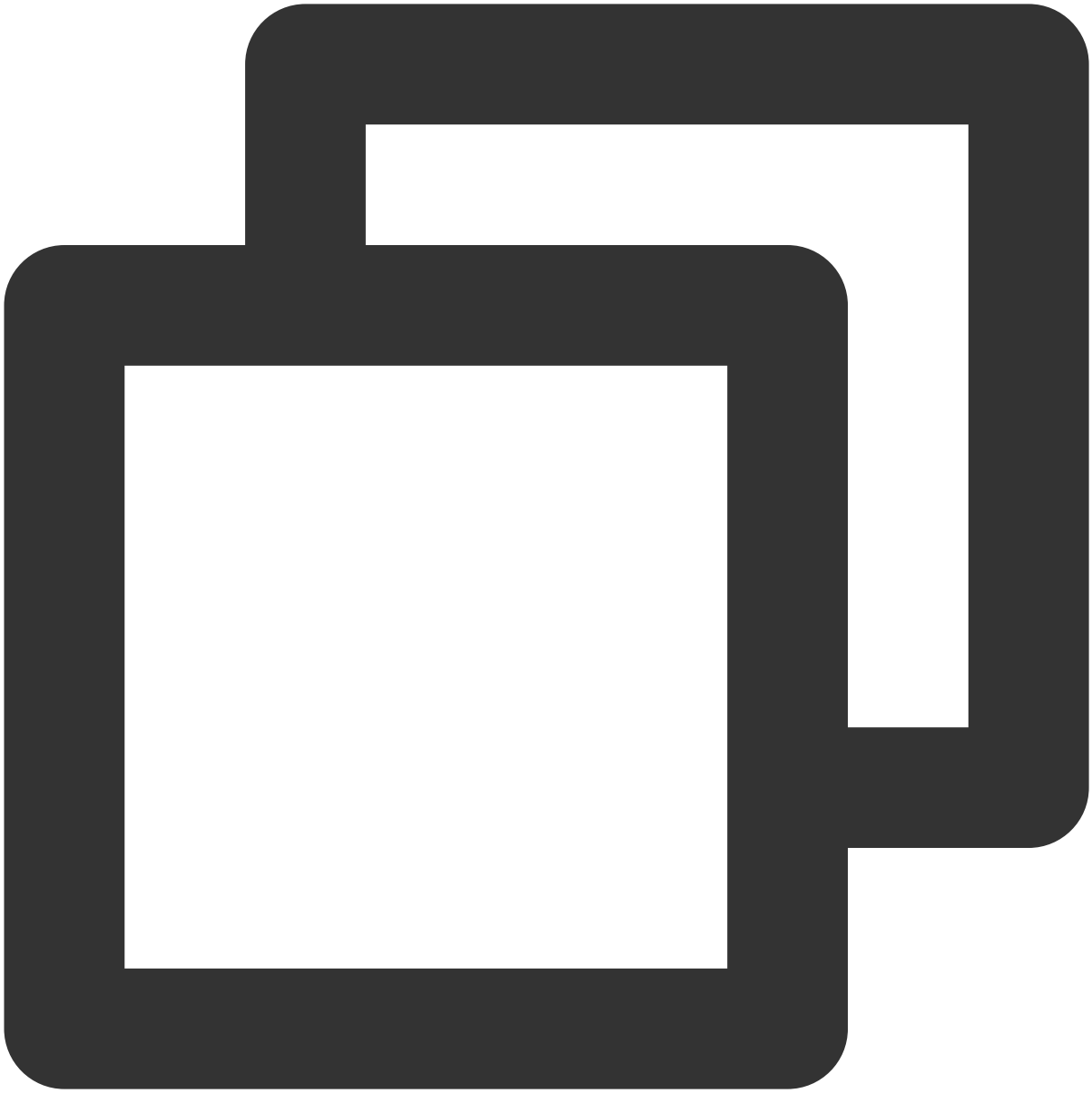
无

isDeviceSupportPip

说明

判断当前设备是否支持画中画模式。

接口



```
static Future<int> isDeviceSupportPip() async;
```

参数说明

无

返回值说明

参数名	类型	描述
isDeviceSupportPip	int	0 可开启画中画模式 -101 android版本过低 -102 画中画权限关闭/设备不支持画中画

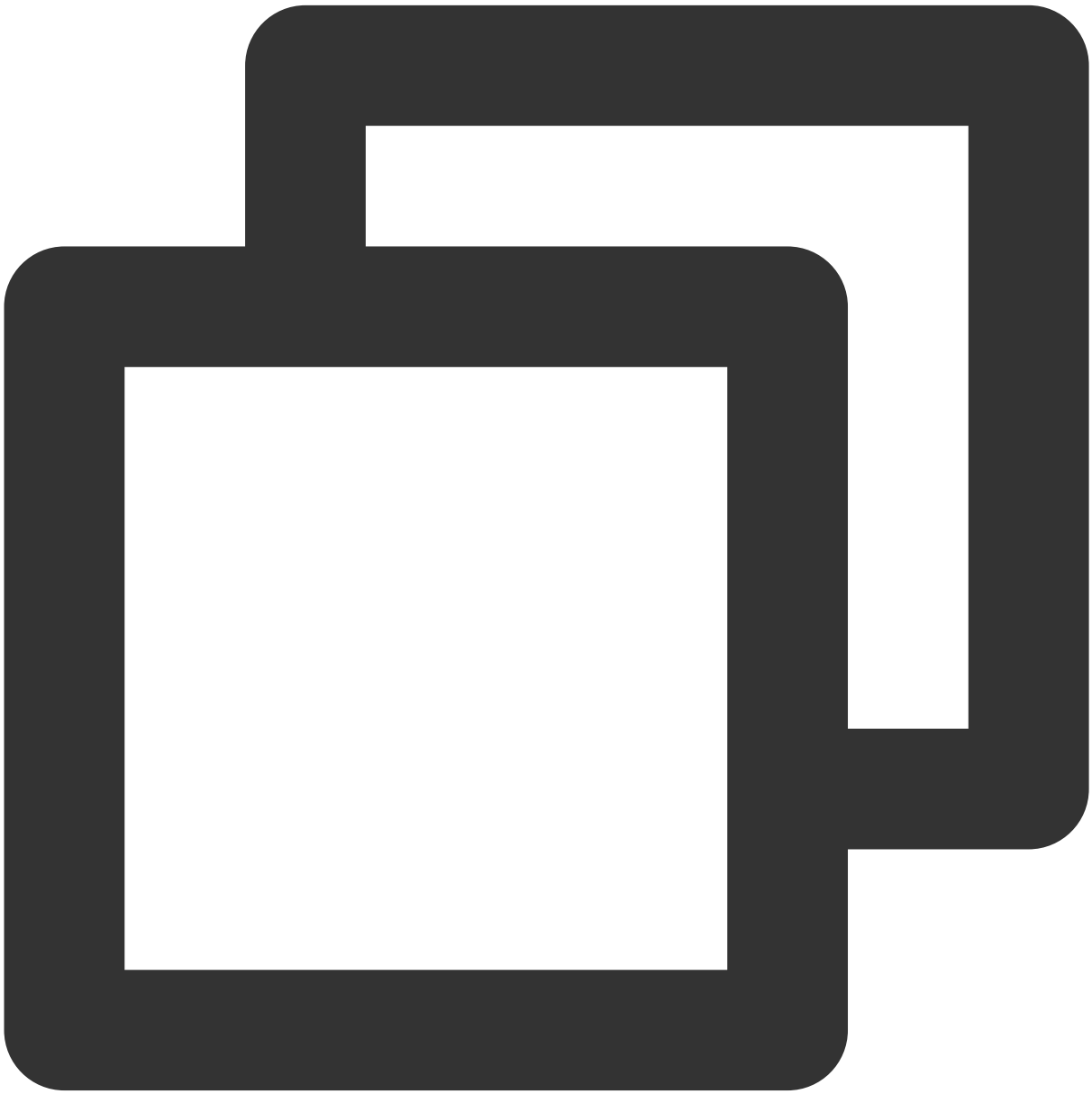
		-103 当前界面已销毁

getLiteAVSDKVersion

说明

获得当前原生层播放器 SDK 的版本号。

接口



```
static Future<String?> getLiteAVSDKVersion() async;
```

参数说明

无

返回值说明

参数名	类型	描述
sdkVersion	String	当前播放器 SDK 版本

startVideoOrientationService

说明

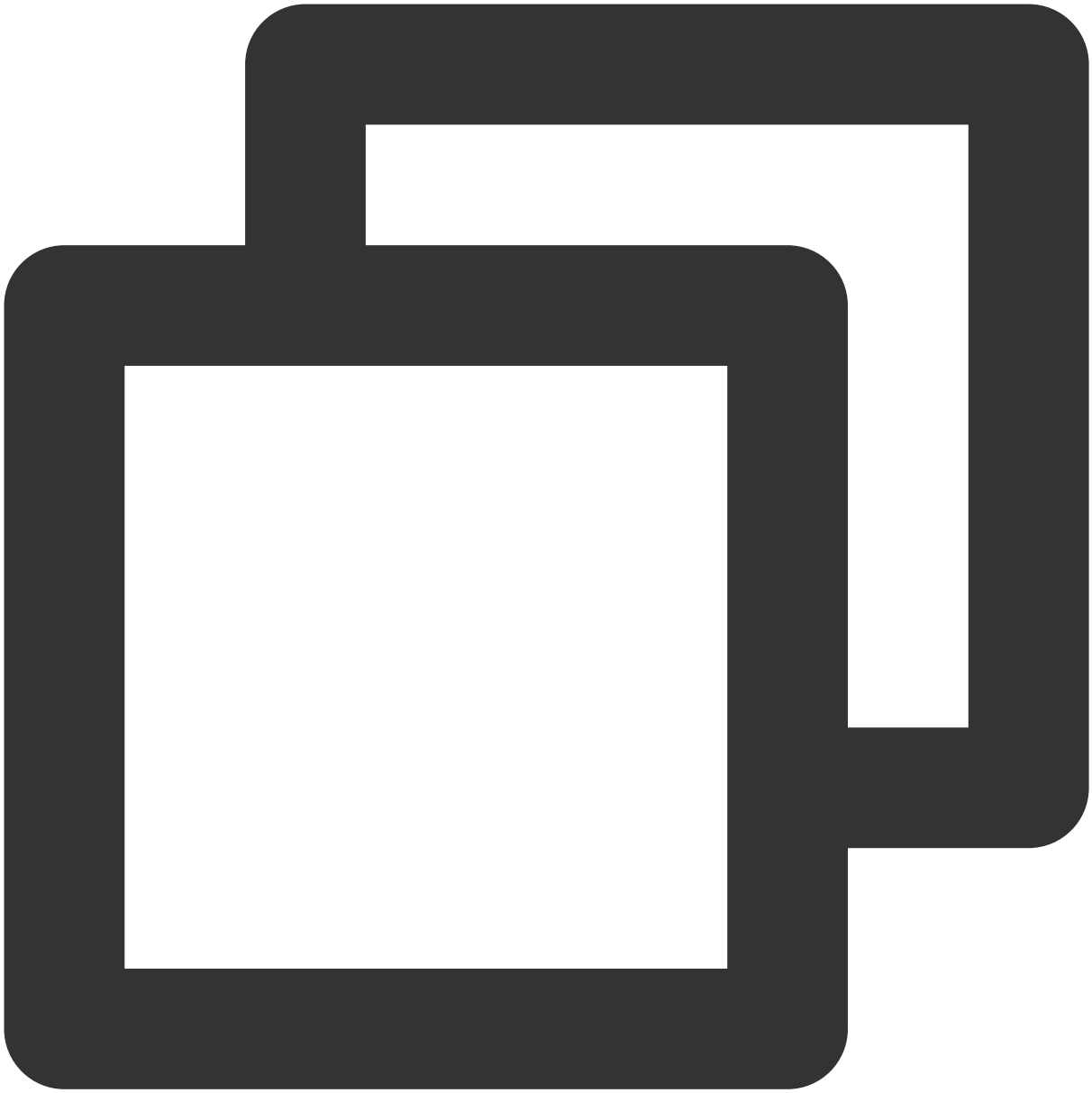
开始监听设备旋转方向，开启之后，如果设备自动旋转打开，播放器会自动根据当前设备方向来旋转视频方向。

该接口目前只适用安卓端，IOS端会自动开启该能力。

注意

在调用该接口前，请务必向用户告知隐私风险。

接口



```
static Future<bool> startVideoOrientationService() async
```

参数说明

无

返回值说明

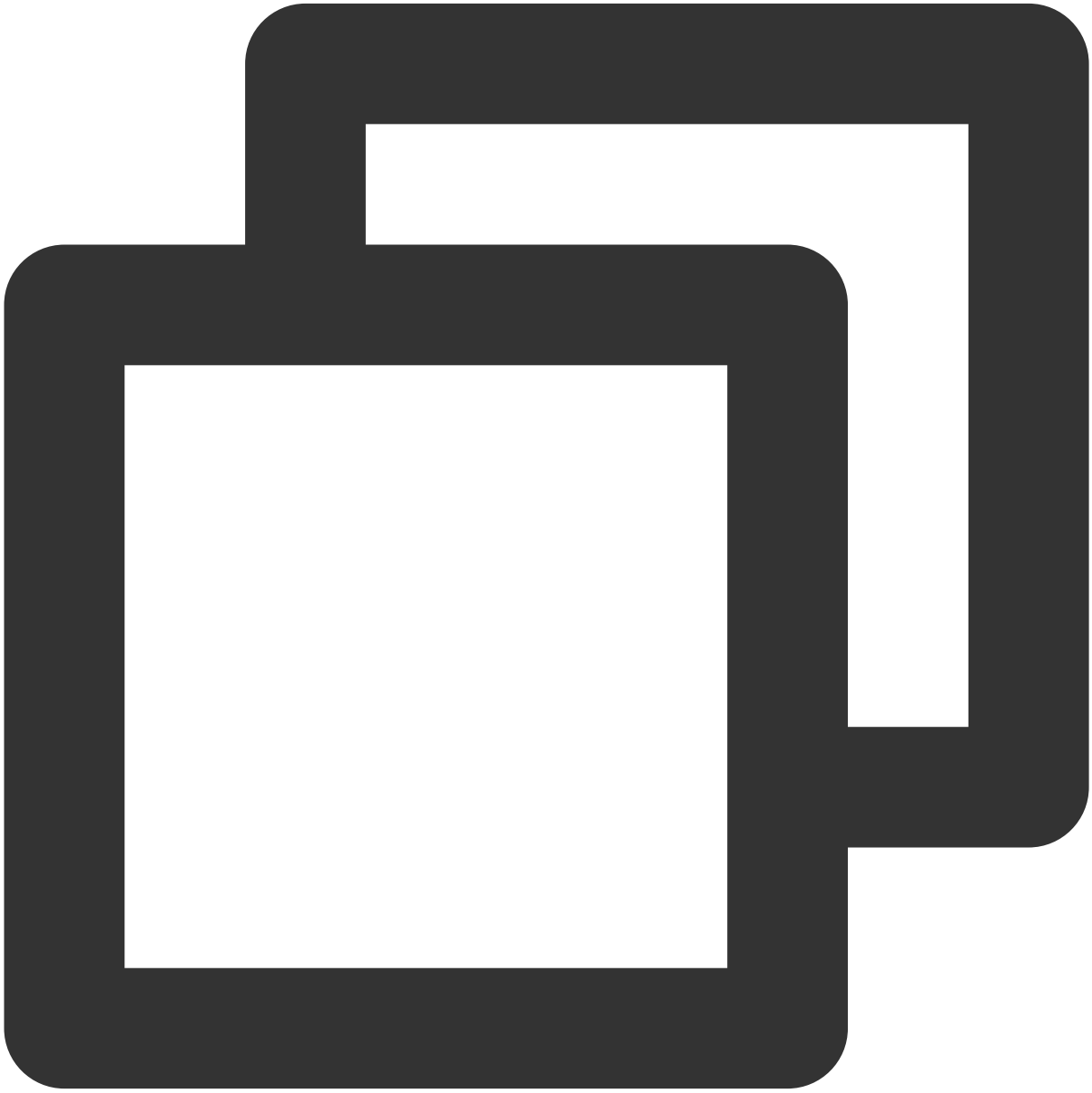
参数名	类型	描述
result	bool	true 开启成功，false 开启失败，如开启过早，还未等到上下文初始化、获取sensor失败等原因

registerSysBrightness

说明

开启或关闭对于系统亮度的监听，如果开启，当系统亮度发生变化，会改变当前 window 亮度，并回调亮度到 flutter 层。该接口需配合 setBrightness 和onExtraEventBroadcast 使用。

接口



```
static Future<void>registerSysBrightness(bool isRegister) async
```

参数说明

参数名	类型	描述

isRegister	bool	true：开启监听。 false：关闭监听。

返回值说明

无

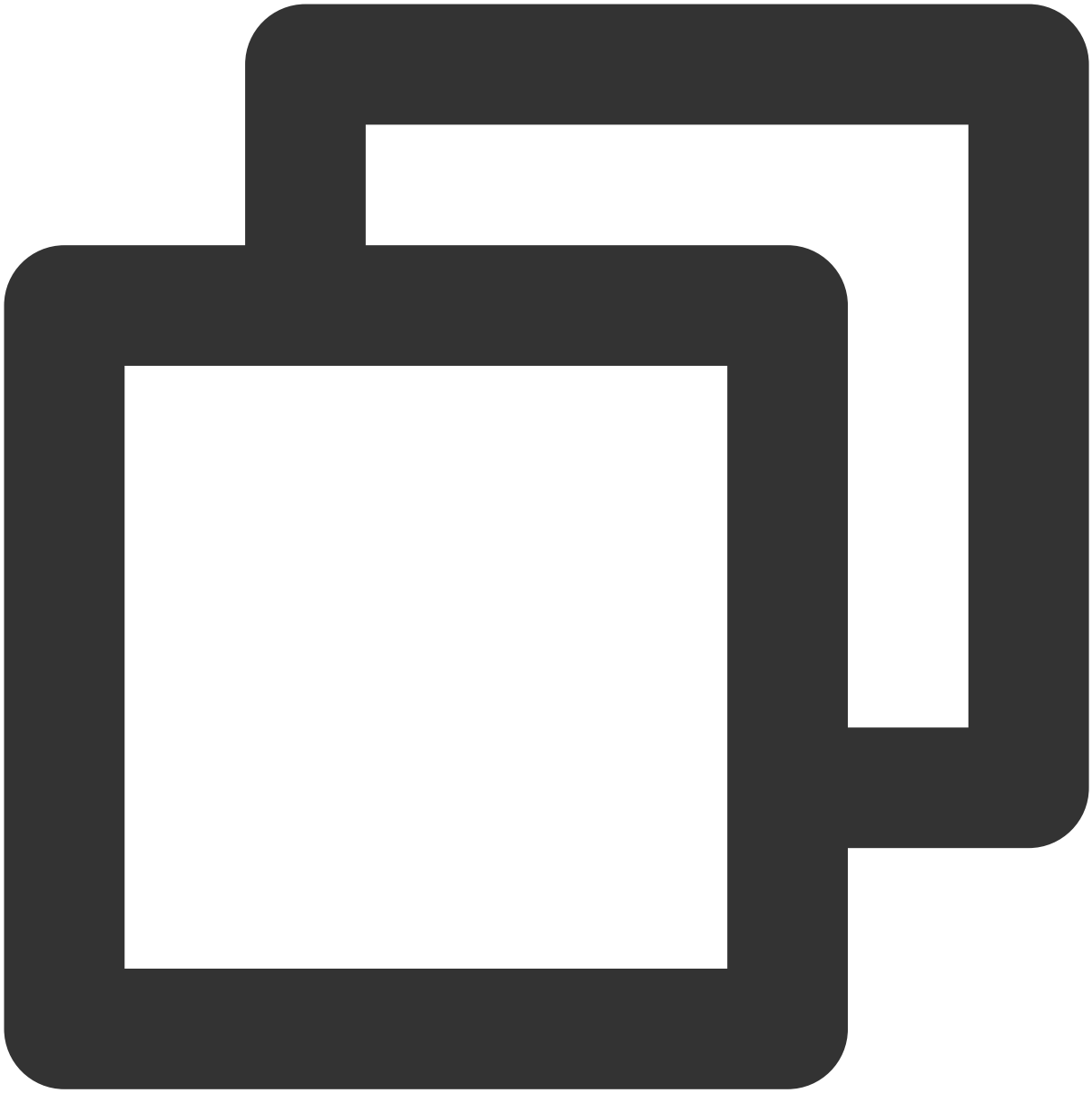
TXVodPlayerController类

initialize

说明

初始化 controller，请求分配共享纹理。

接口



```
Future<void> initialize({bool? onlyAudio}) async;
```

参数说明

参数名	类型	描述
onlyAudio	bool	选填，是否是纯音频播放器

返回值说明

无

startVodPlay

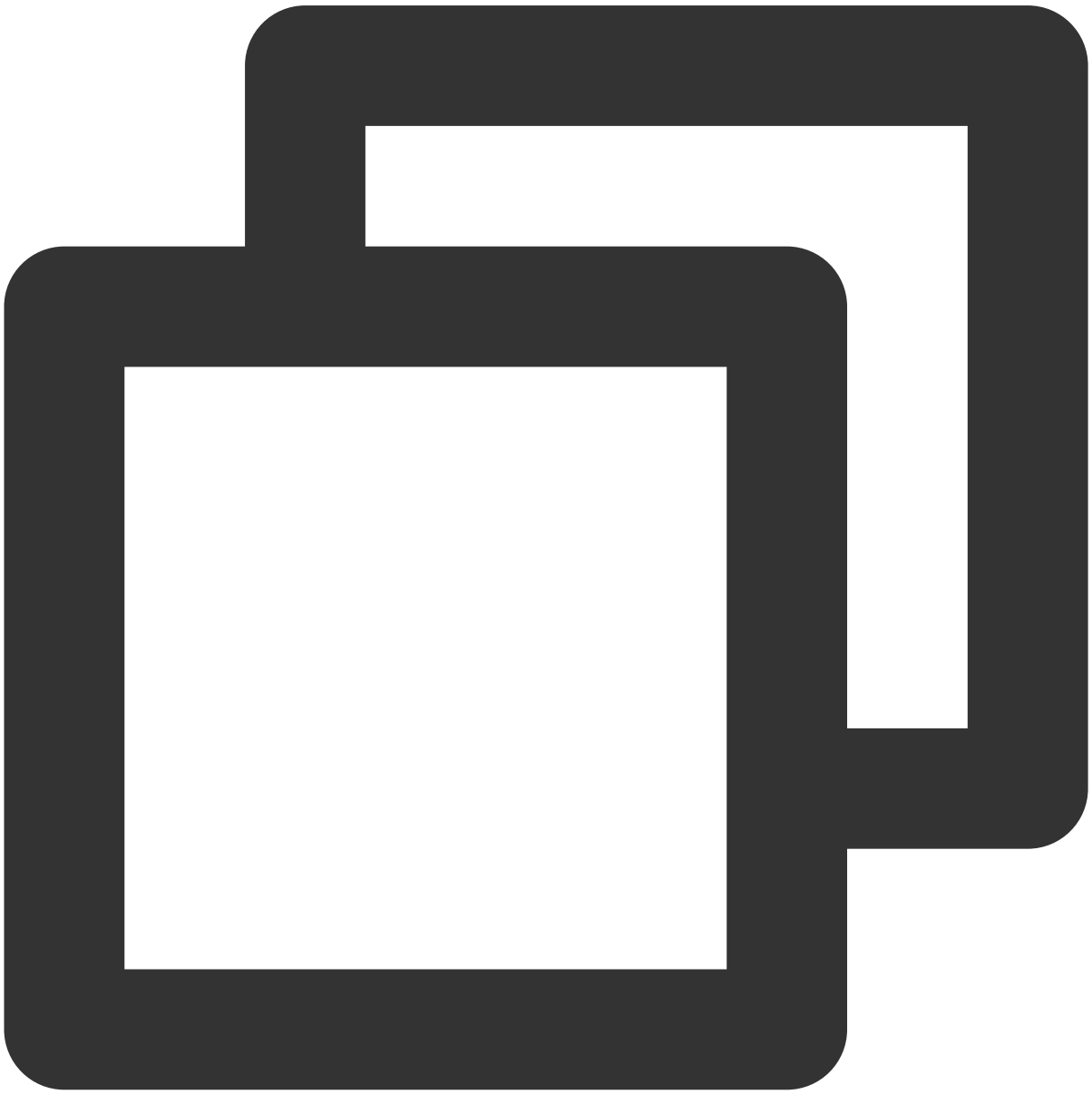
注意

10.7版本开始，startPlay 变更为 startVodPlay，需要通过 {@link SuperPlayerPlugin#setGlobalLicense} 设置 Licence 后方可成功播放， 否则将播放失败（黑屏），全局仅设置一次即可。直播 Licence、短视频 Licence 和视频播放 Licence 均可使用，若您暂未获取上述 Licence ，可快速免费申请测试版 Licence 以正常播放，正式版 License 需 购买。

说明

通过播视频url进行播放。

接口



```
Future<bool> startVodPlay(String url) async;
```

参数说明

参数名	类型	描述
url	String	要播放的视频url

返回值说明

参数名	类型	描述
result	bool	创建是否成功

startVodPlayWithParams

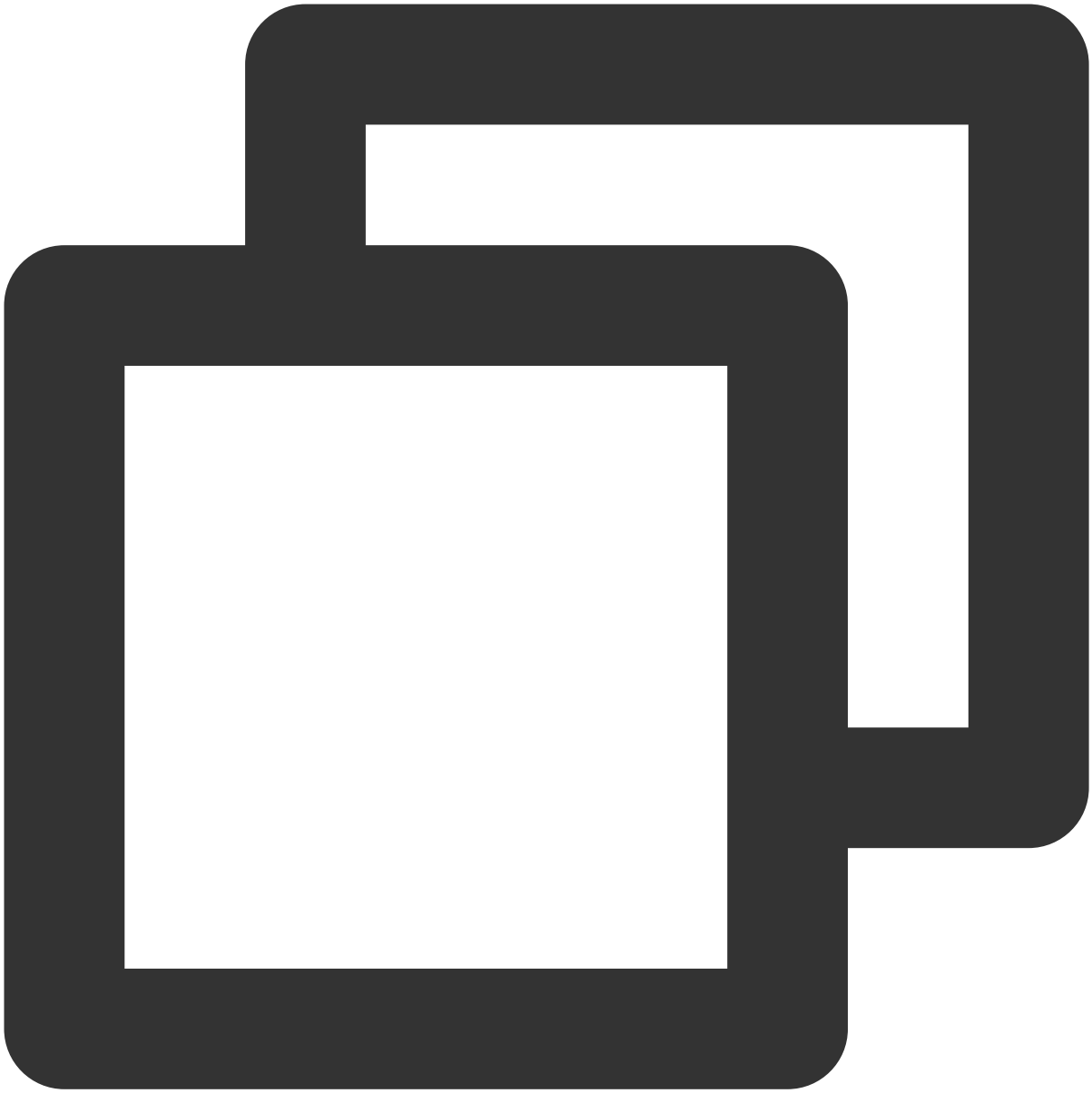
注意

10.7版本开始，startPlay 变更为 startVodPlay，需要通过 {@link SuperPlayerPlugin#setGlobalLicense} 设置 Licence 后方可成功播放， 否则将播放失败（黑屏），全局仅设置一次即可。直播 Licence、短视频 Licence 和视频播放 Licence 均可使用，若您暂未获取上述 Licence ，可 快速免费申请测试版 Licence 以正常播放，正式版 License 需 购买。

说明

通过视频 field 进行播放。

接口



```
Future<void> startVodPlayWithParams(TXPlayInfoParams params) async;
```

参数说明

参数名	类型	描述
appId	int	应用 appId。必填
fileId	String	文件 id。必填
sign	String	防盗链签名，参考 防盗链产品文档

url	String	播放链接，该字段与fileId填写一个即可

返回值说明

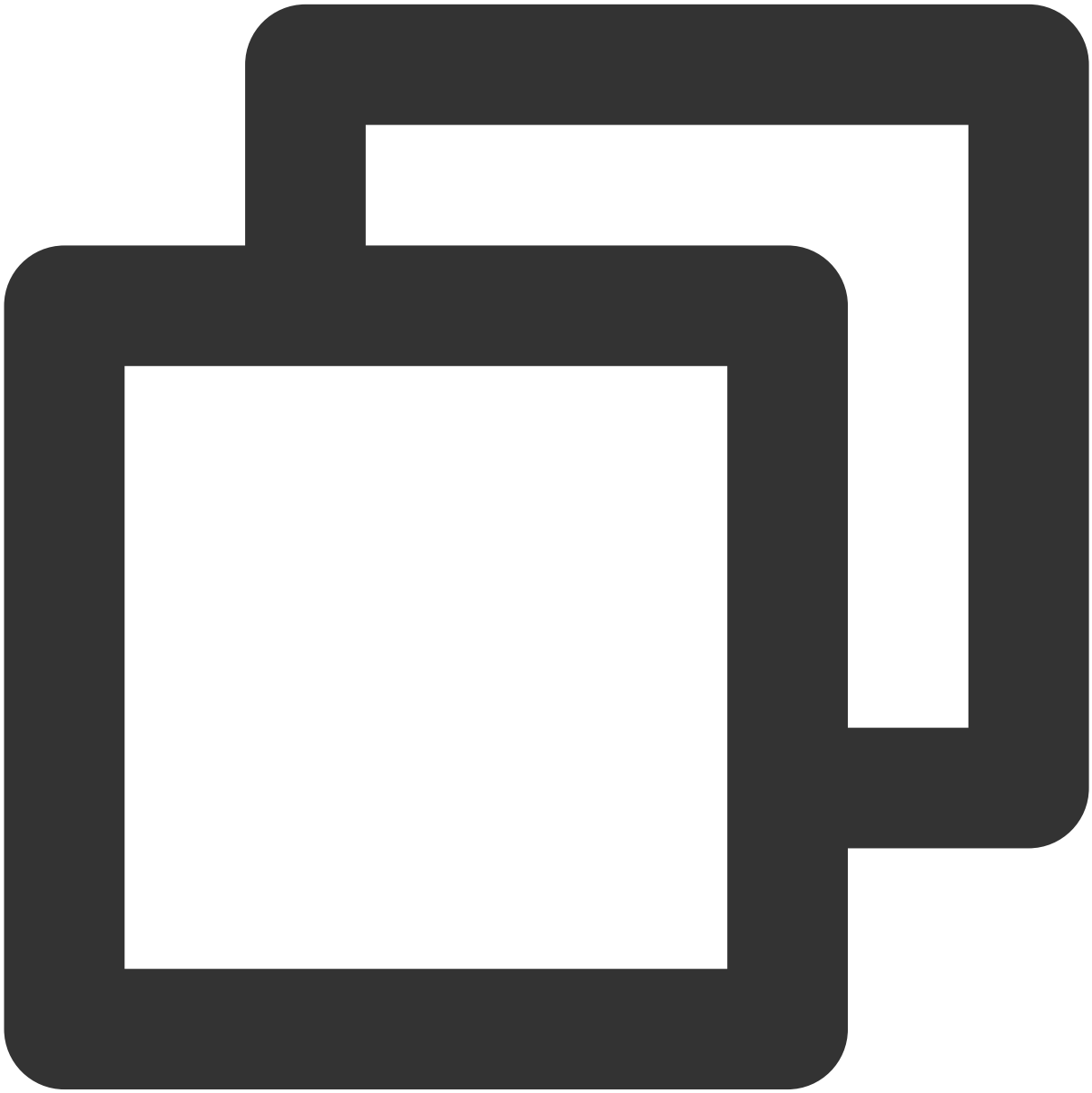
无

pause

说明

暂停当前正在播放的视频。

接口



```
Future<void> pause() async;
```

参数说明

无

返回值说明

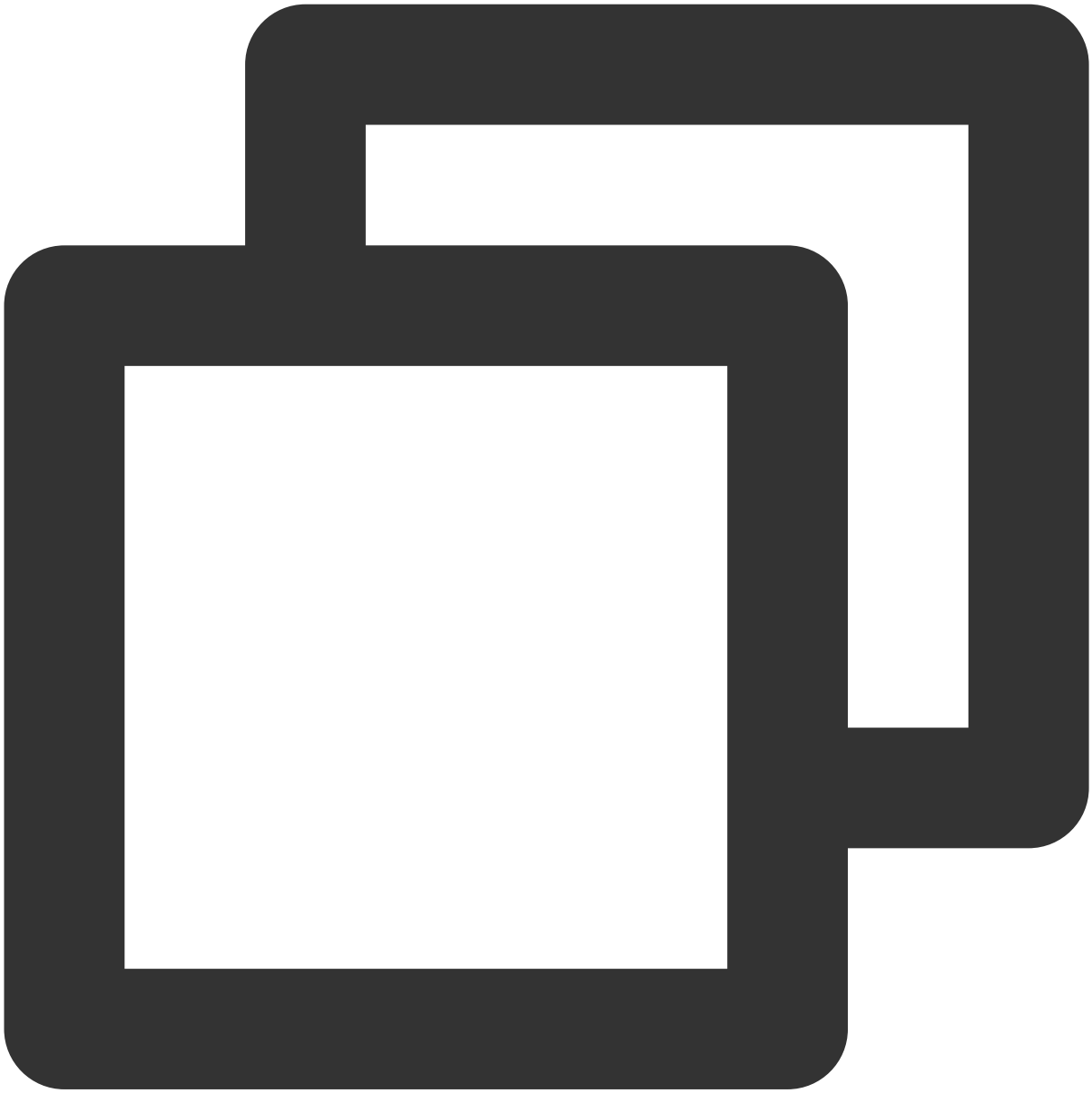
无

resume

说明

将当前处于暂停状态的视频恢复播放。

接口



```
Future<void> resume() async;
```

参数说明

无

返回值说明

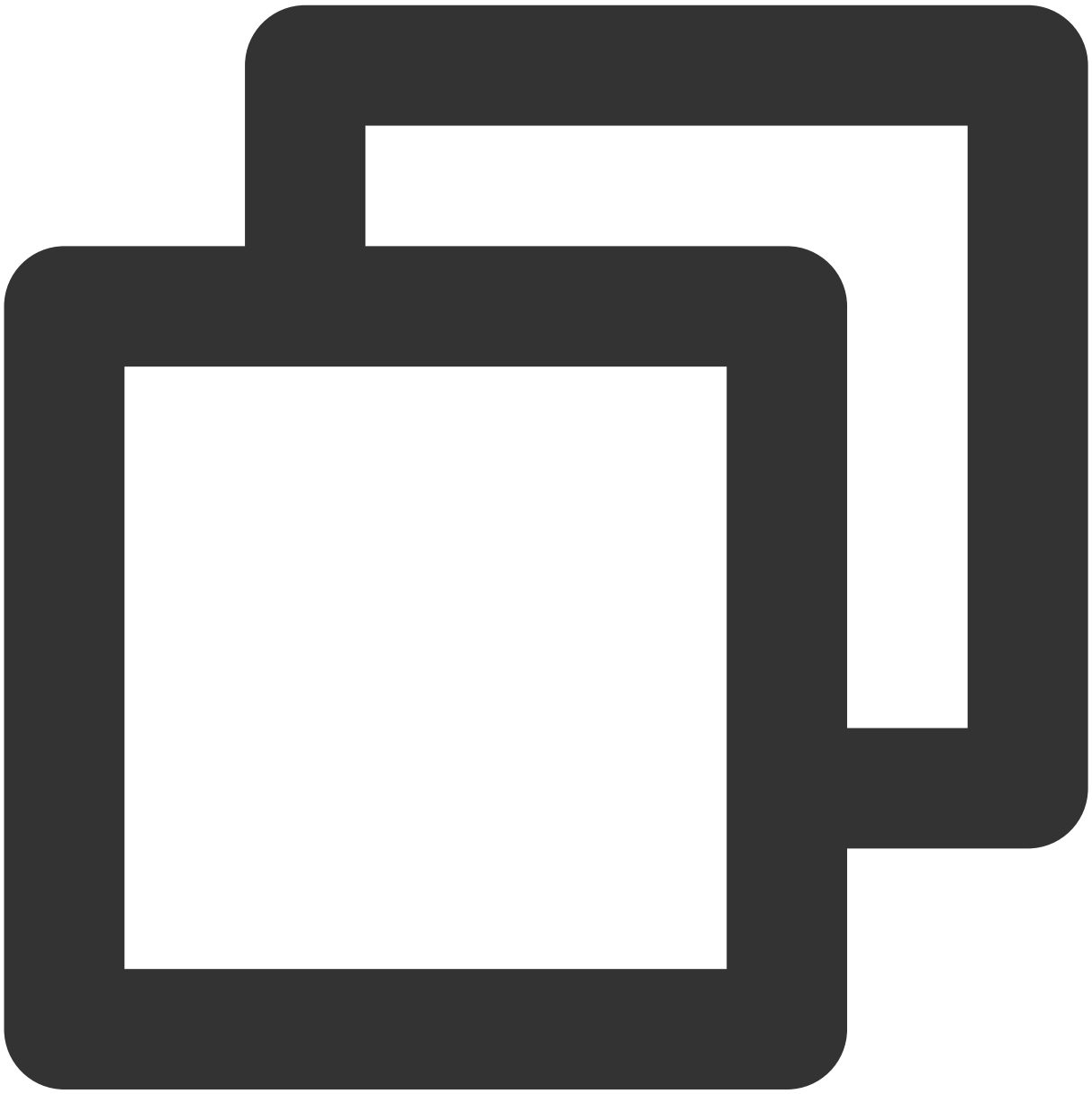
无

stop

说明

停止当前正在播放的视频。

接口



```
Future<bool> stop({bool isNeedClear = false}) async;
```

参数说明

参数名	类型	描述
isNeedClear	bool	是否清除最后一帧画面

返回值说明

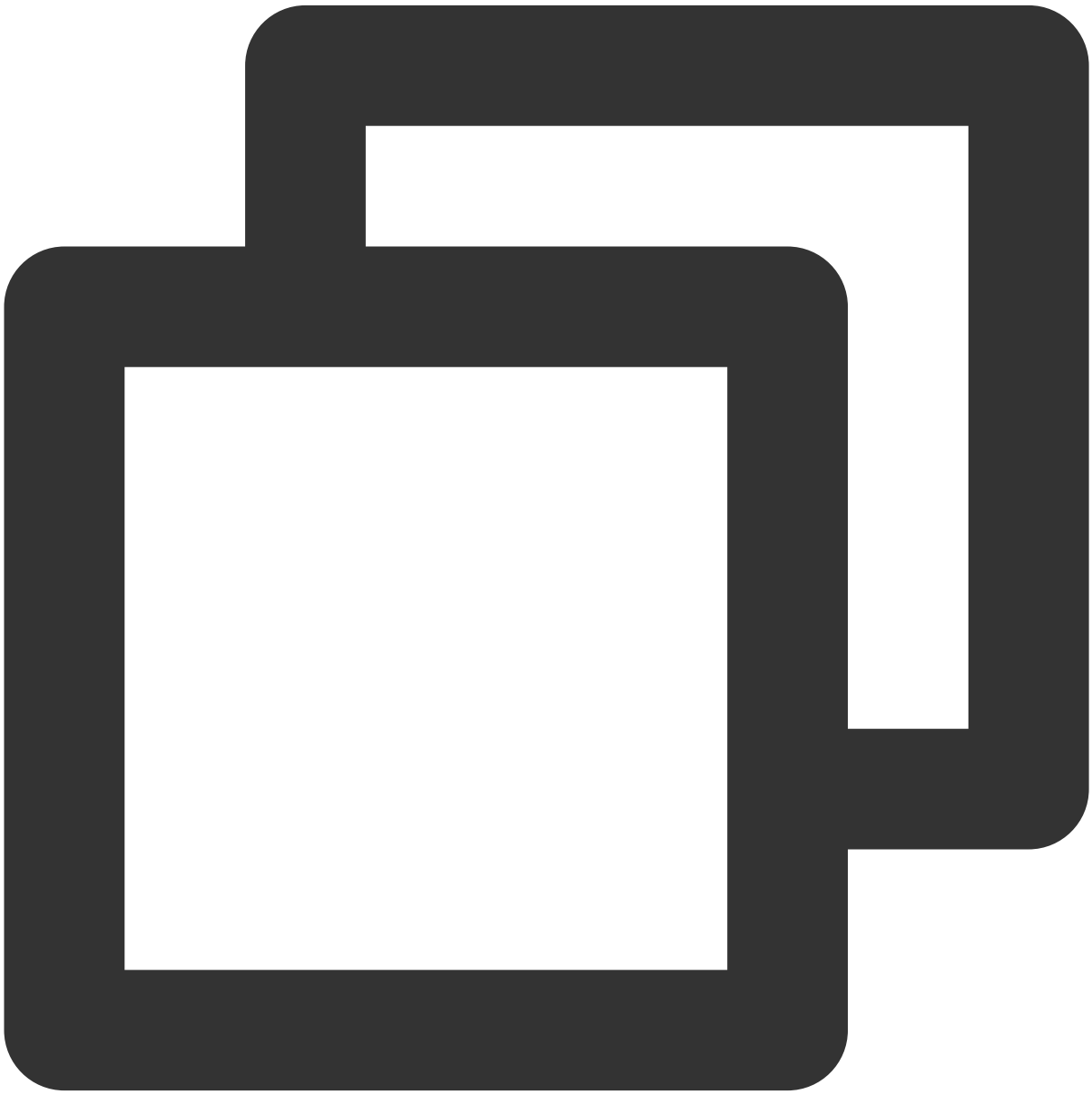
参数名	类型	描述
result	bool	停止是否成功

setIsAutoPlay

说明

设置即将播放的视频，在 startVodPlay 加载视频地址之后，是否直接自动播放。

接口



```
Future<void> setIsAutoPlay({bool? isAutoPlay}) async;
```

参数说明

参数名	类型	描述
isAutoPlay	bool	是否自动播放

返回值说明

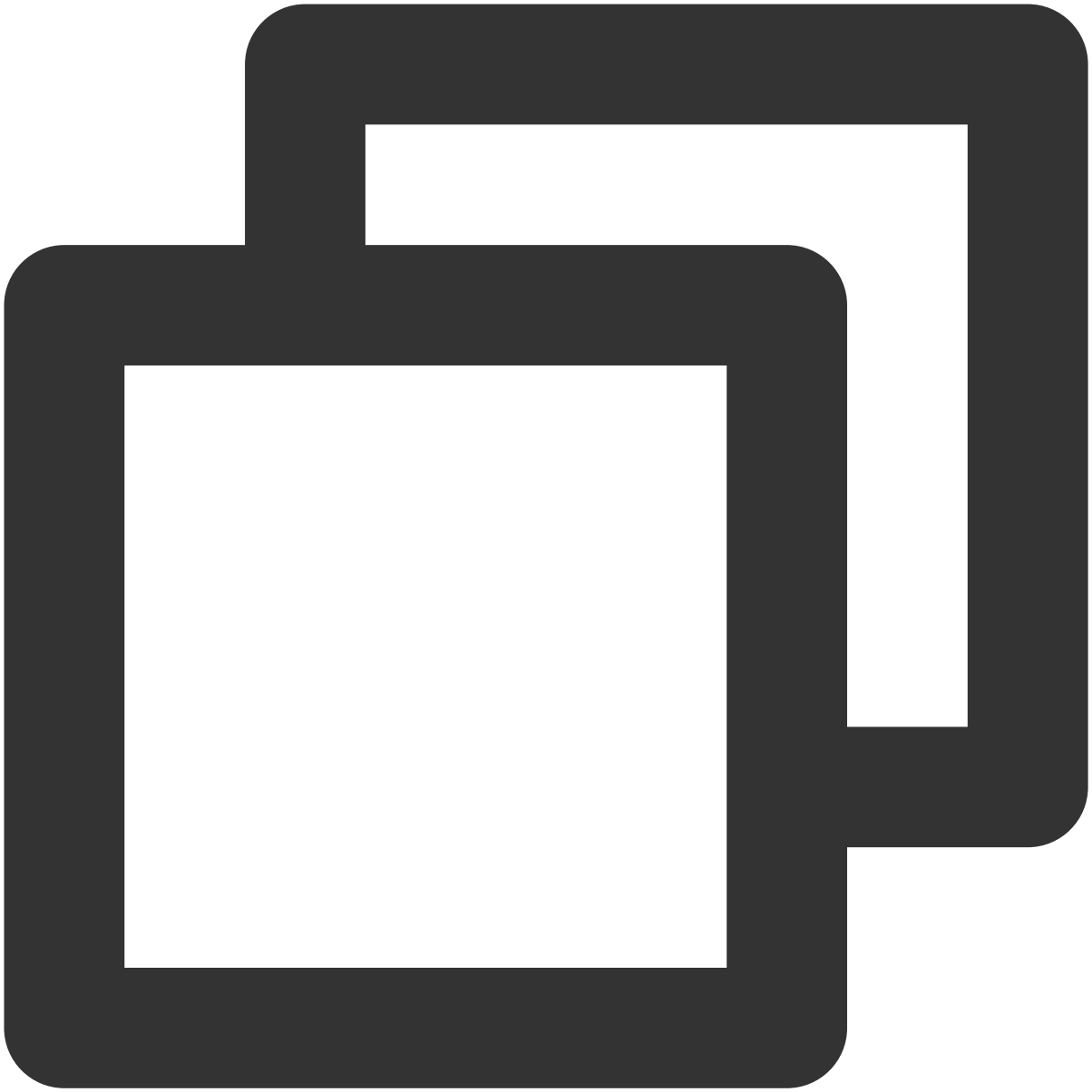
无

isPlaying

说明

当前播放器是否正在播放。

接口



```
Future<bool> isPlaying() async;
```

参数说明

无

返回值说明

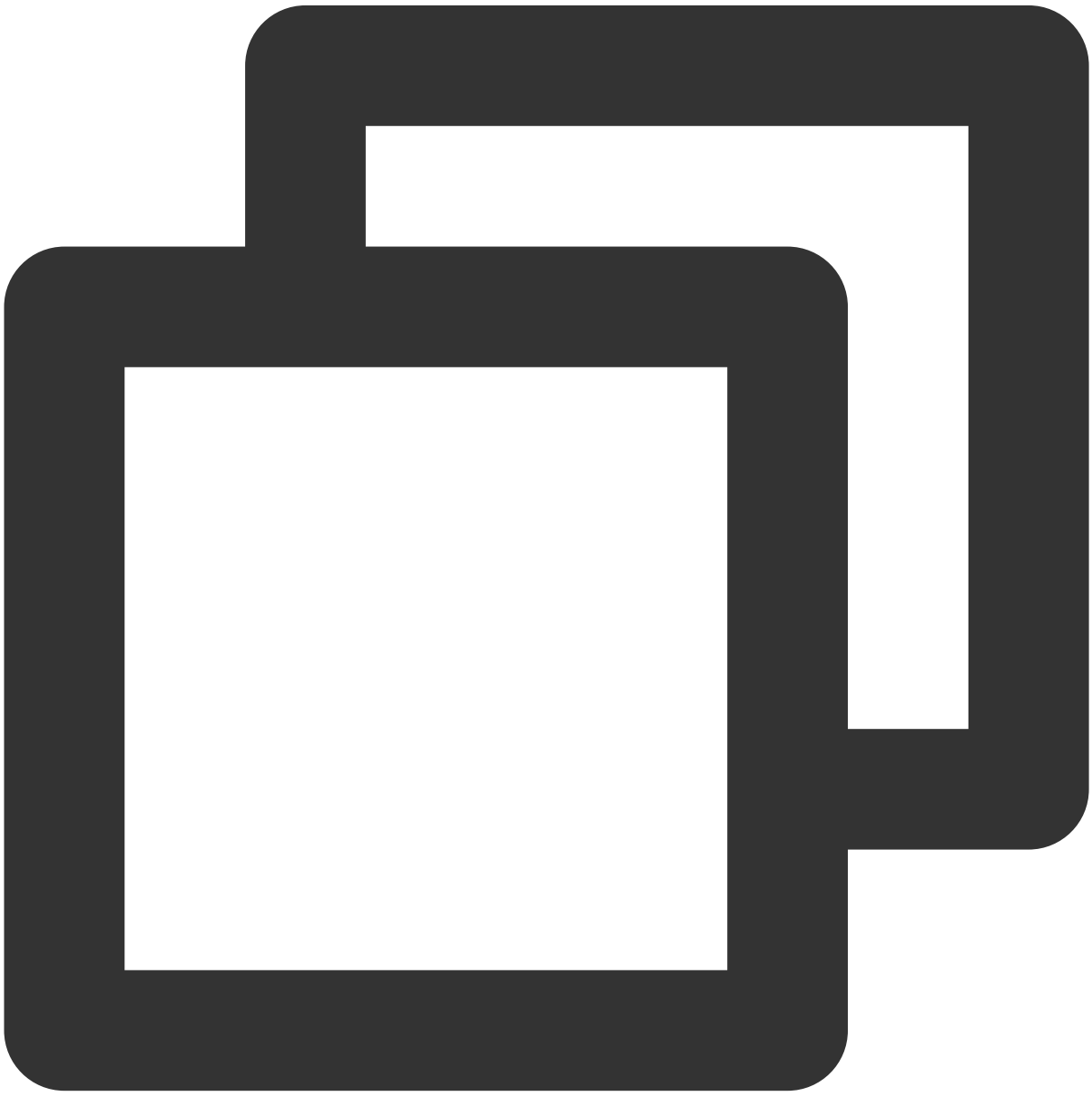
参数名	类型	描述
isPlaying	bool	是否正在播放

setMute

说明

设置当前播放是否静音。

接口



```
Future<void> setMute(bool mute) async;
```

参数说明

参数名	类型	描述

mute	bool	是否静音

返回值说明

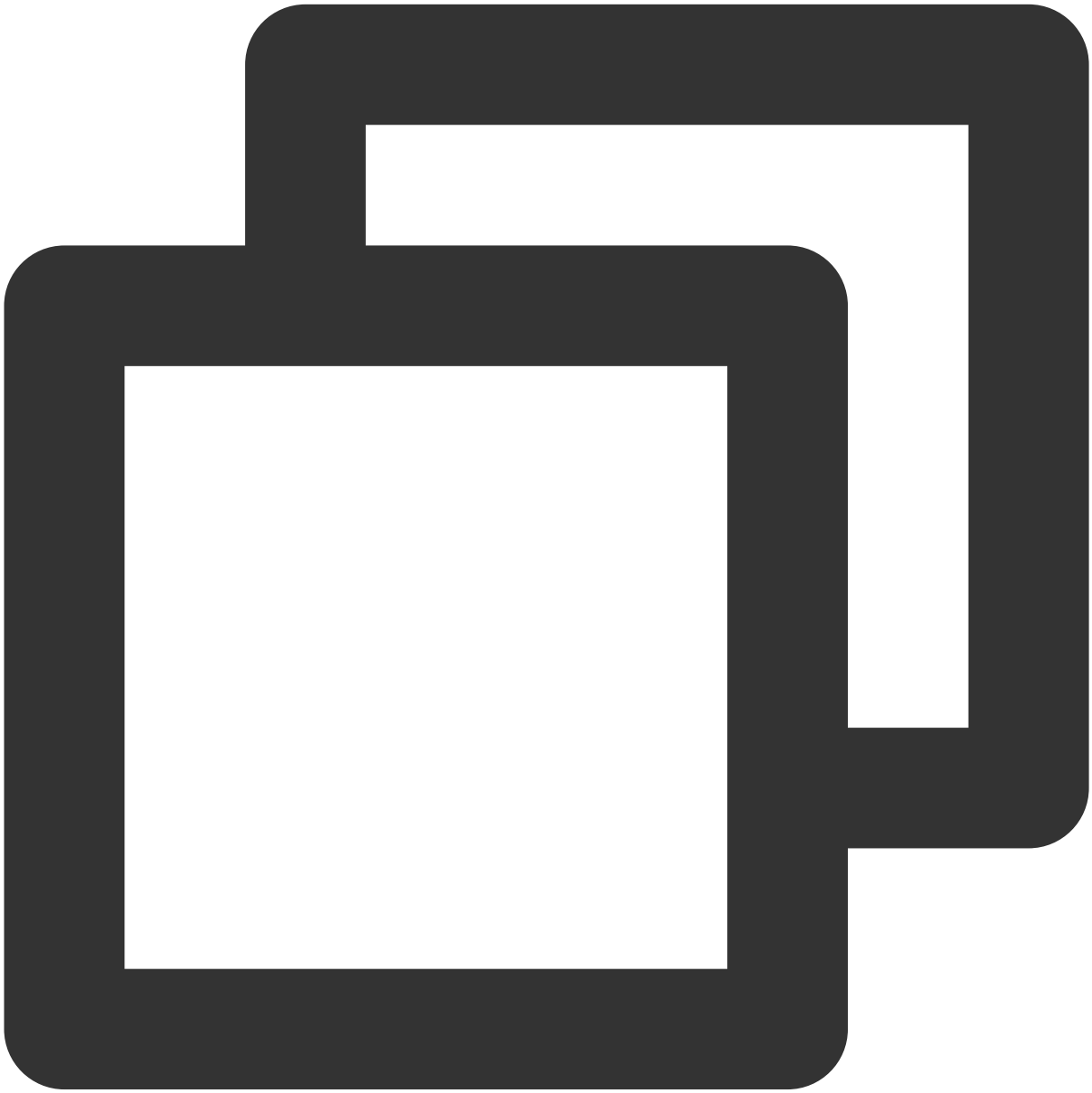
无

setLoop

说明

视频播放完成之后是否循环播放。

接口



```
Future<void> setLoop(bool loop) async;
```

参数说明

参数名	类型	描述
loop	bool	是否循环播放

返回值说明

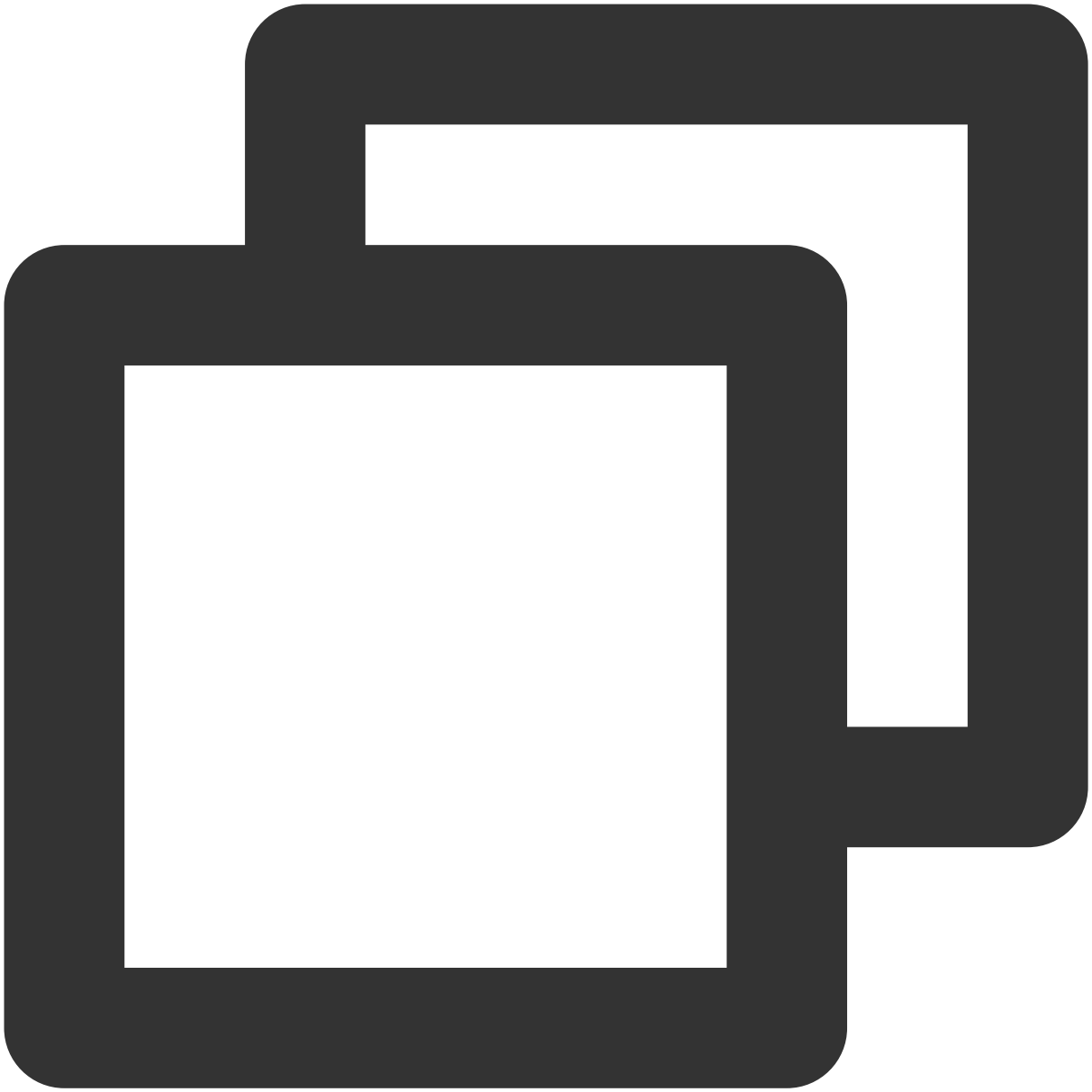
无

seek

说明

将进度调整到指定位置。

接口



```
_controller.seek(progress);
```

参数说明

参数名	类型	描述
progress	double	需要调整到的播放时间，单位 秒。

返回值说明

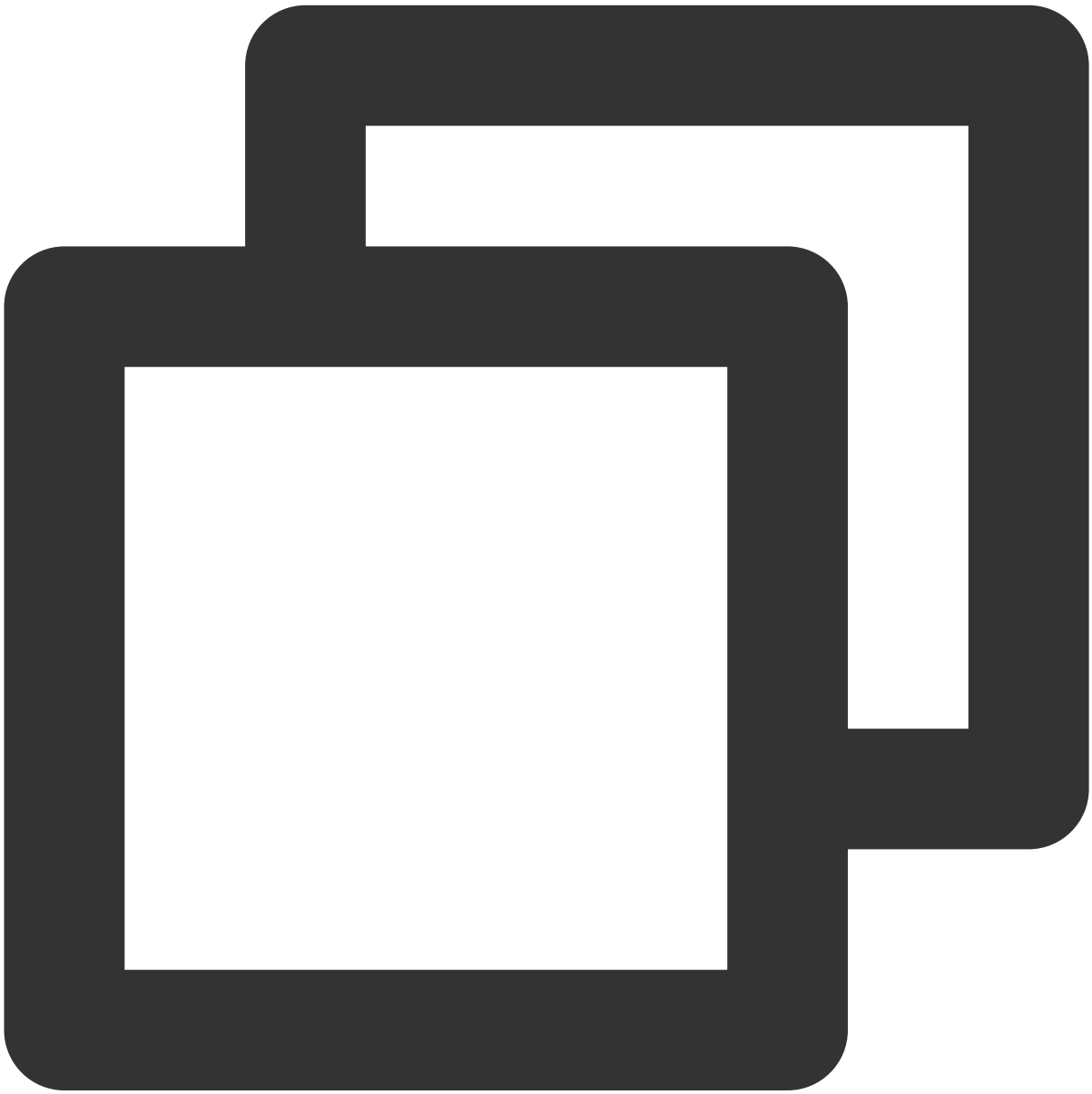
无

setRate

说明

设置视频播放的速率。

接口



```
Future<void> setRate(double rate) async;
```

参数说明

参数名	类型	描述

rate	double	视频的播放速率。默认1.0

返回值说明

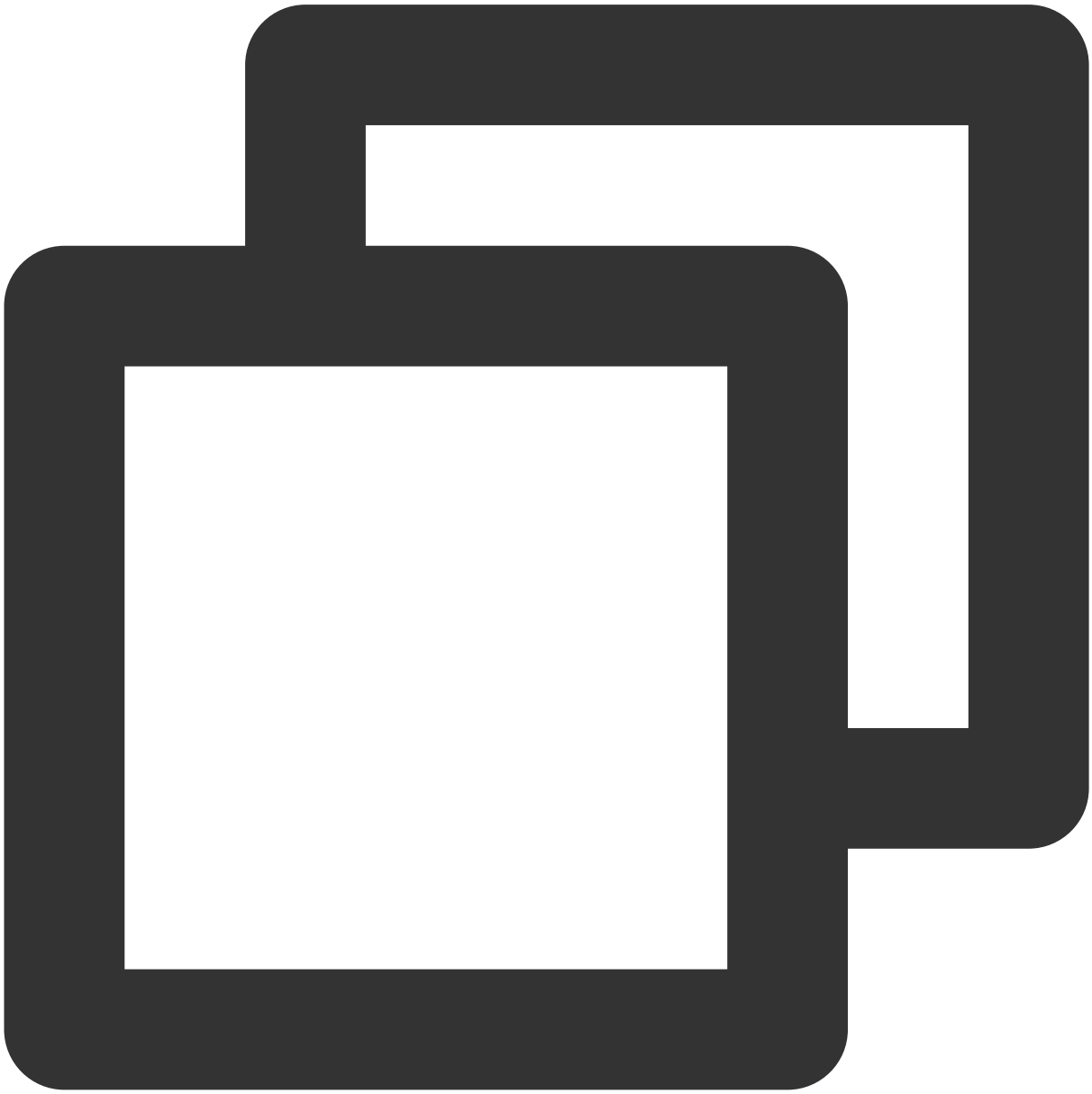
无

getSupportedBitrates

说明

获得当前正在播放的视频支持的码率。

接口



```
Future<List?> getSupportedBitrates() async;
```

参数说明

无

返回值说明

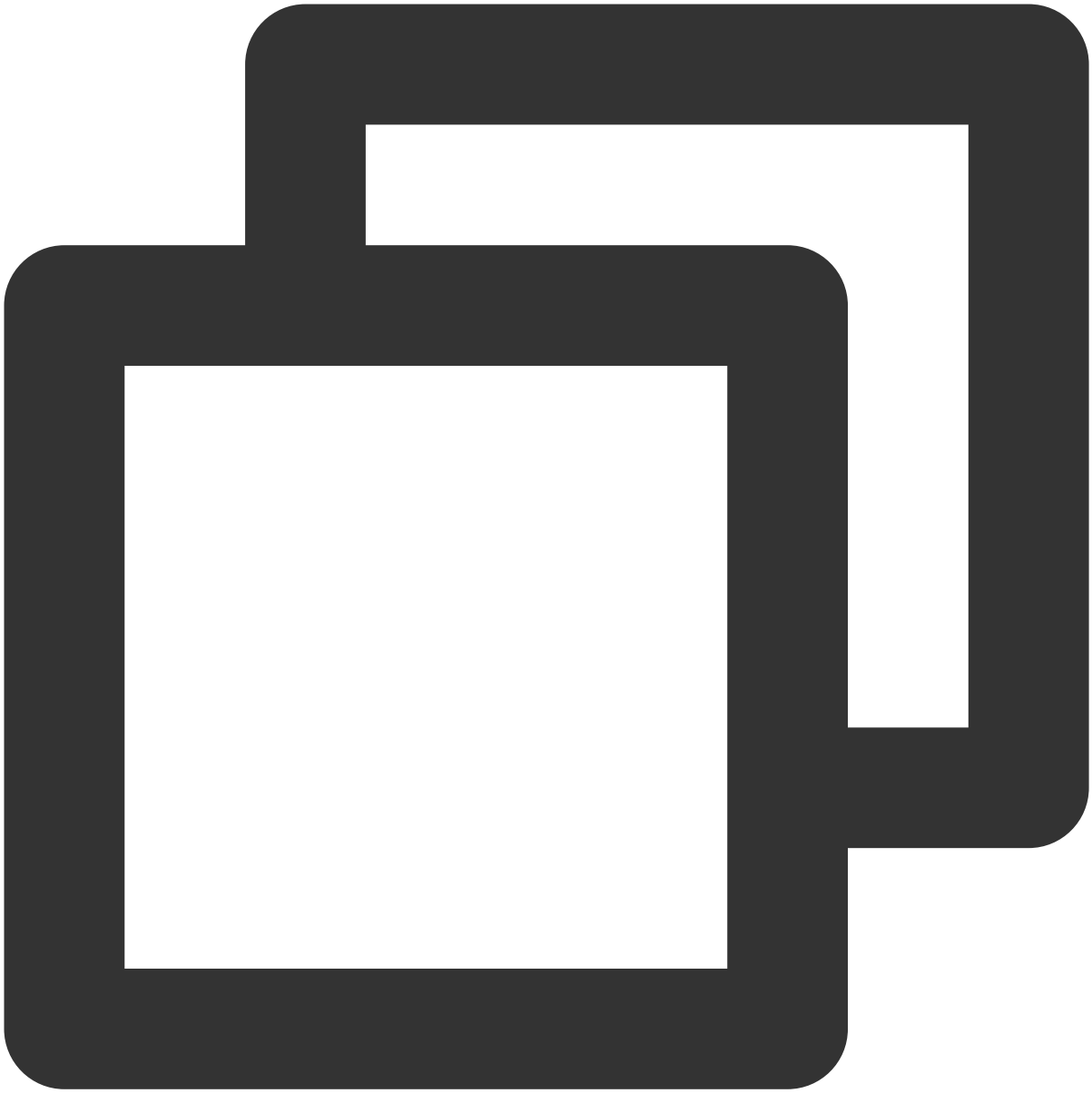
返回值	类型	描述
index	int	码率序号
width	int	码率对应视频宽度
height	int	码率对应视频高度
bitrate	int	码率值

getBitrateIndex

说明

获得设置过的码率序号。

接口



```
Future<int> getBitrateIndex() async;
```

参数说明

无

返回值说明

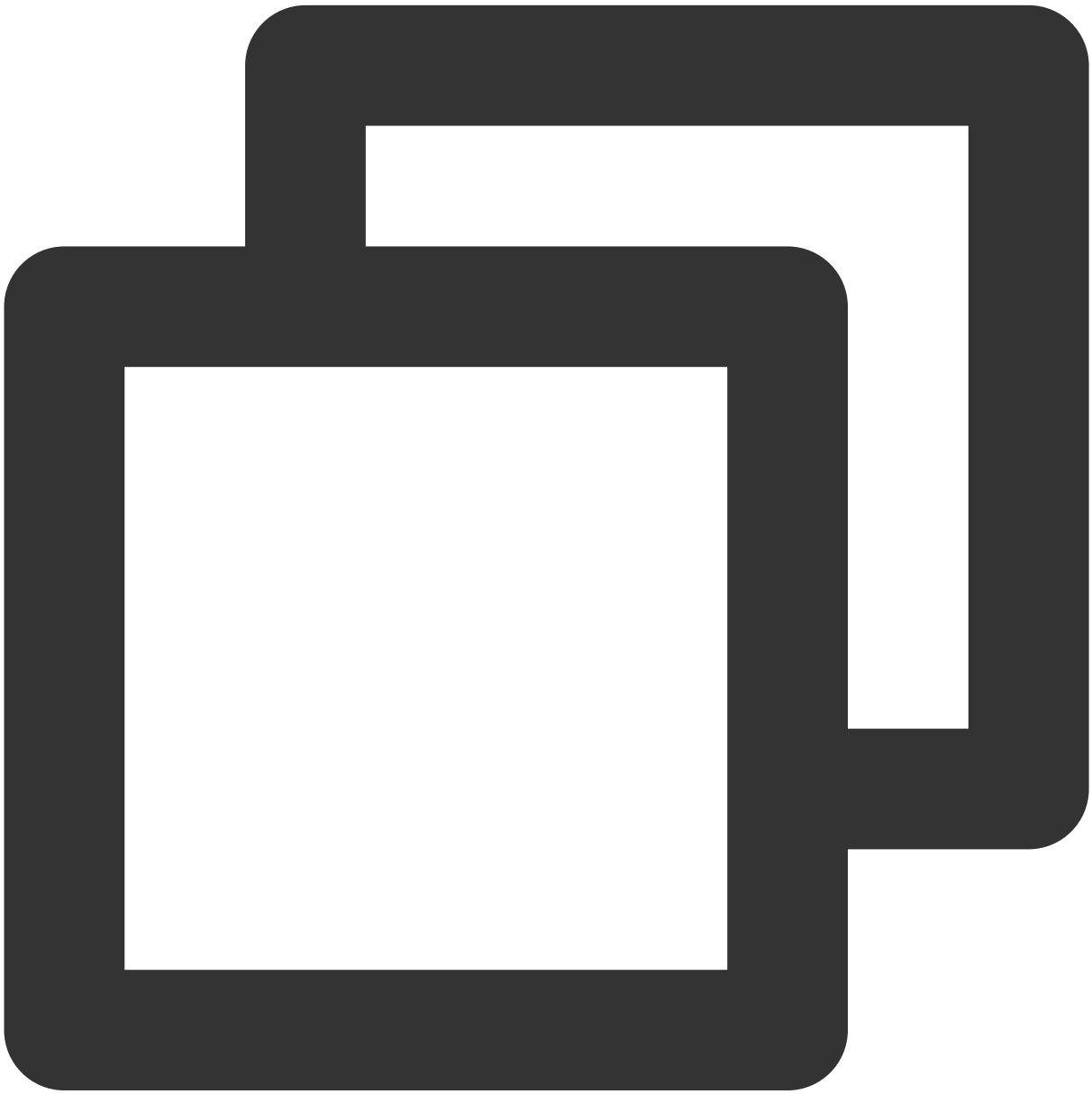
返回值	类型	描述
index	int	码率序号

setBitrateIndex

说明

通过码率序号，设置当前码率。

接口



```
Future<void> setBitrateIndex(int index) async;
```

参数说明

返回值	类型	描述

index	int	码率序号。 传入-1时，表示开启码流自适应。

返回值说明

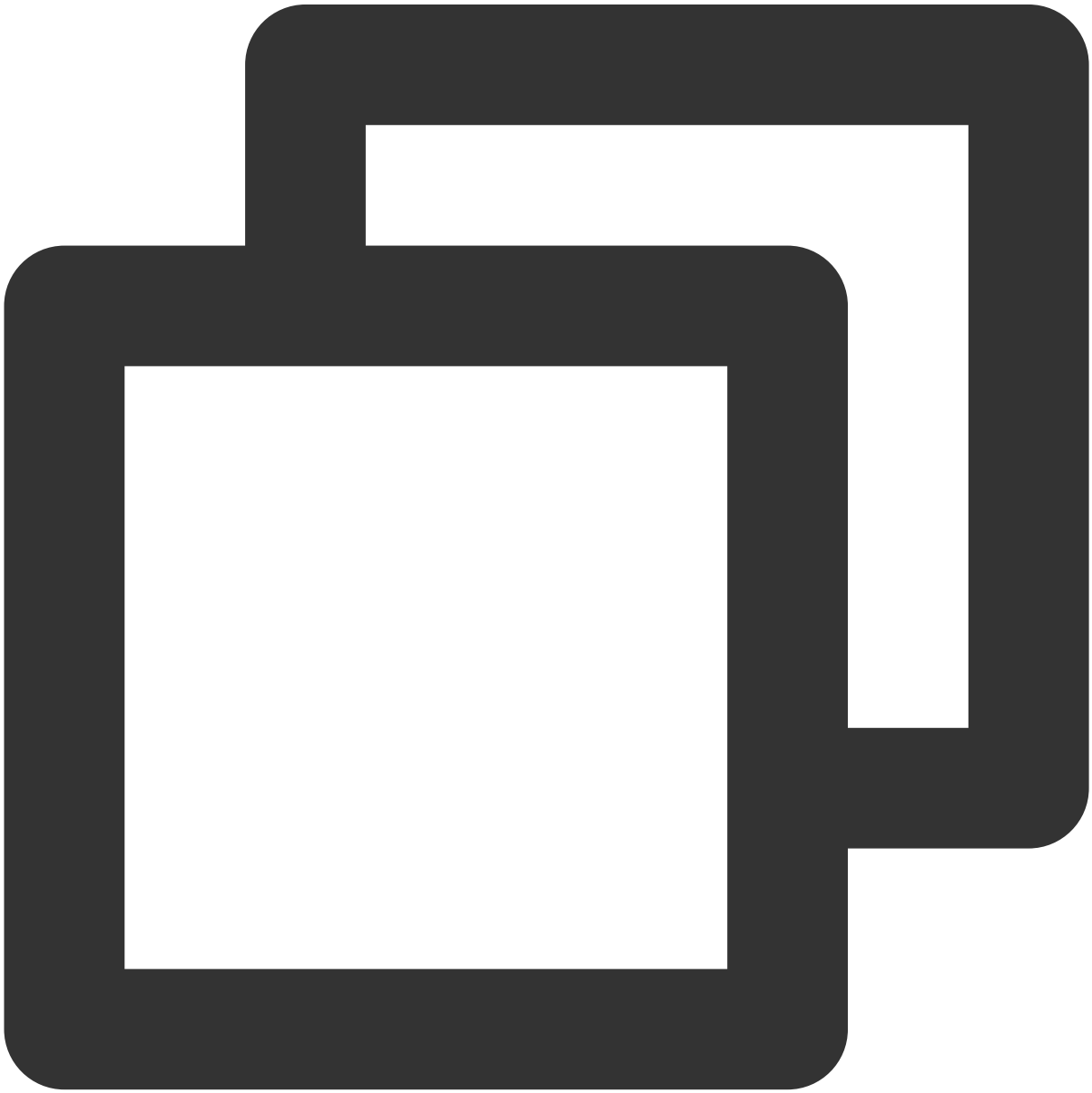
无

setStartTime

说明

指定播放开始时间。

接口



```
Future<void> setStartTime(double startTime) async;
```

参数说明

返回值	类型	描述
startTime	double	播放开始时间，单位 秒

返回值说明

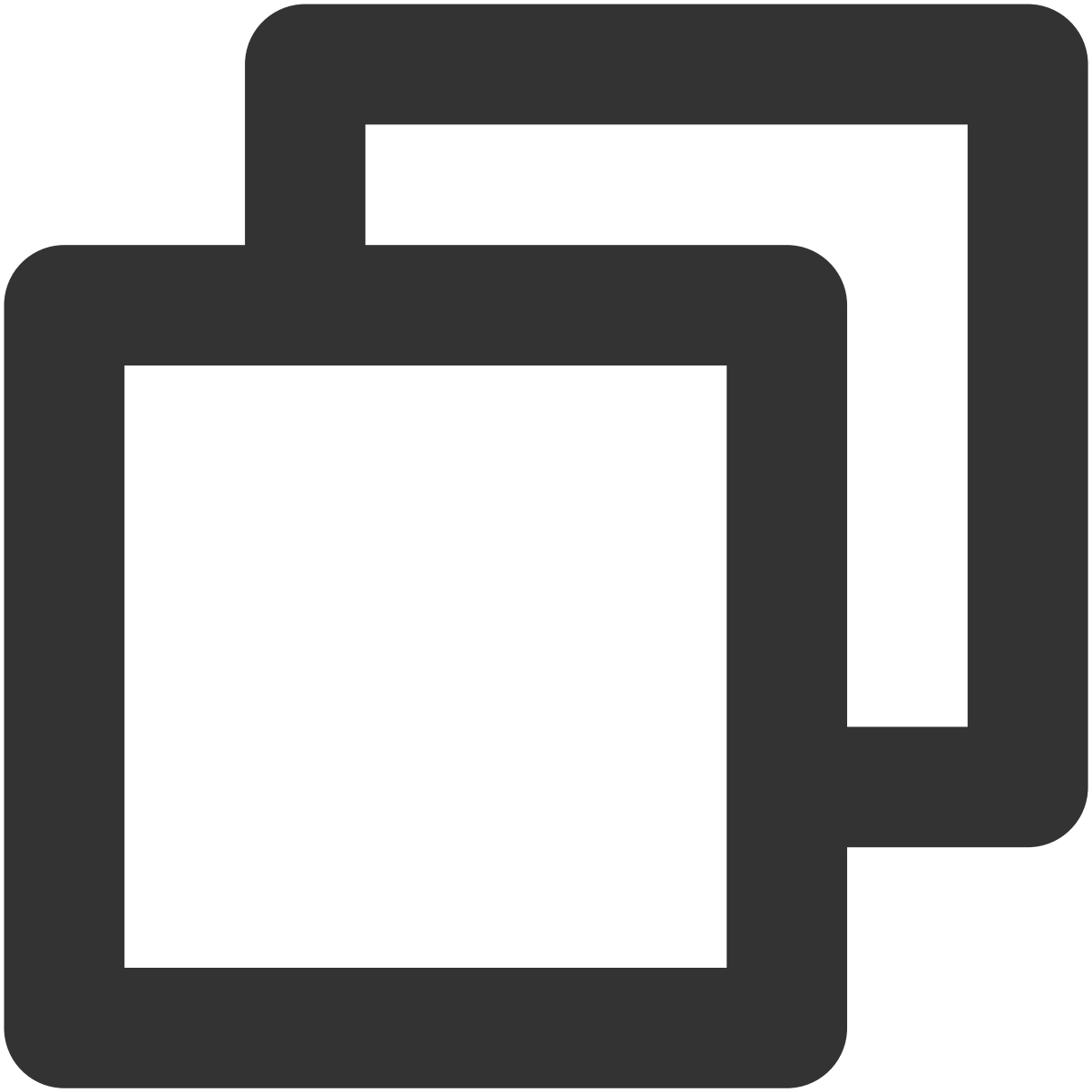
无

setAudioPlayoutVolume

说明

设置视频的声音大小。

接口



```
Future<void> setAudioPlayoutVolume(int volume) async;
```

参数说明

参数名	类型	描述
volume	int	视频声音大小，范围0~100

返回值说明

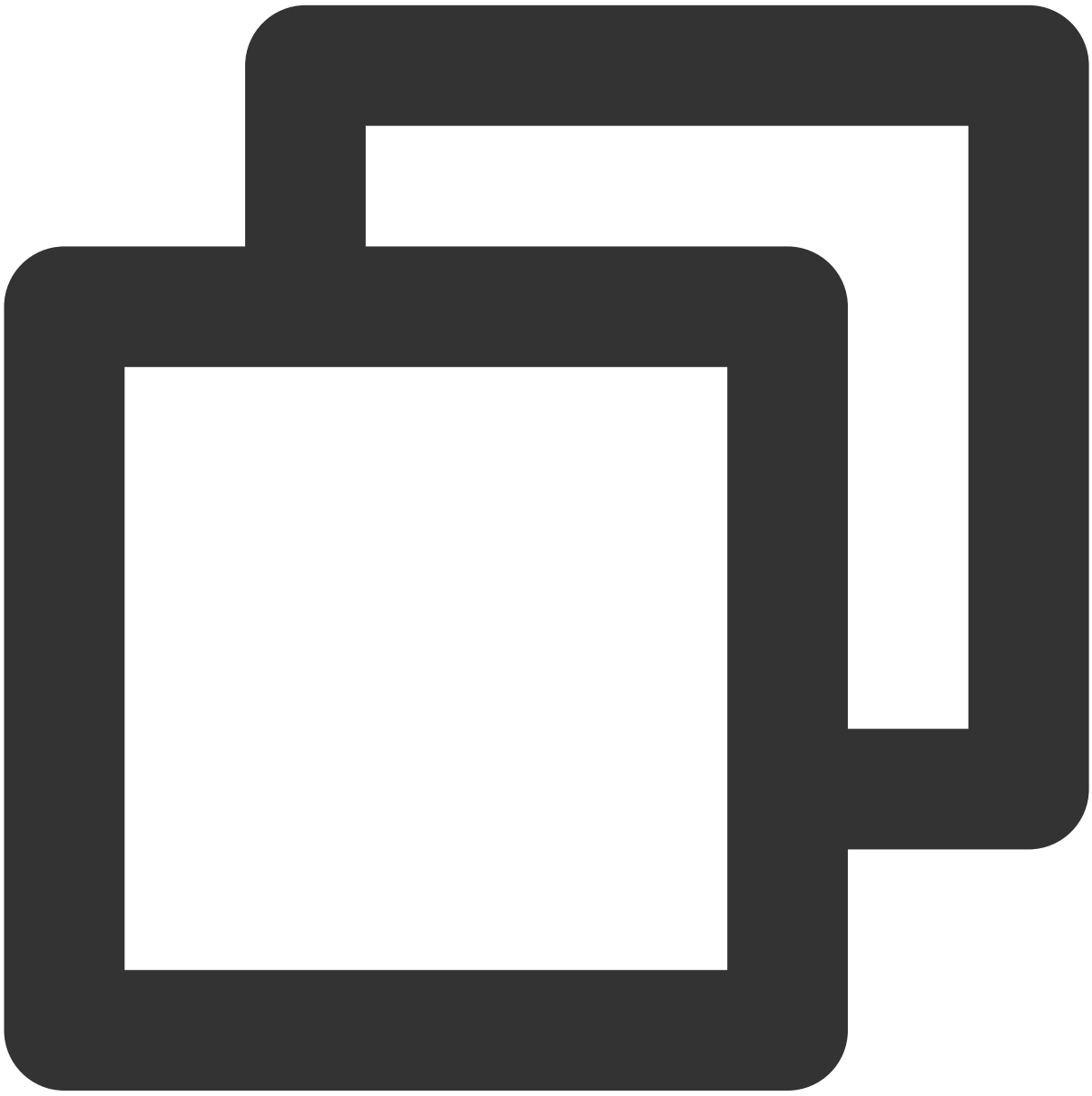
无

setRequestAudioFocus

说明

请求获得音频焦点，适用于Android。

接口



```
Future<bool> setRequestAudioFocus(bool focus) async;
```

参数说明

参数名	类型	描述

focus	bool	是否设置焦点

返回值说明

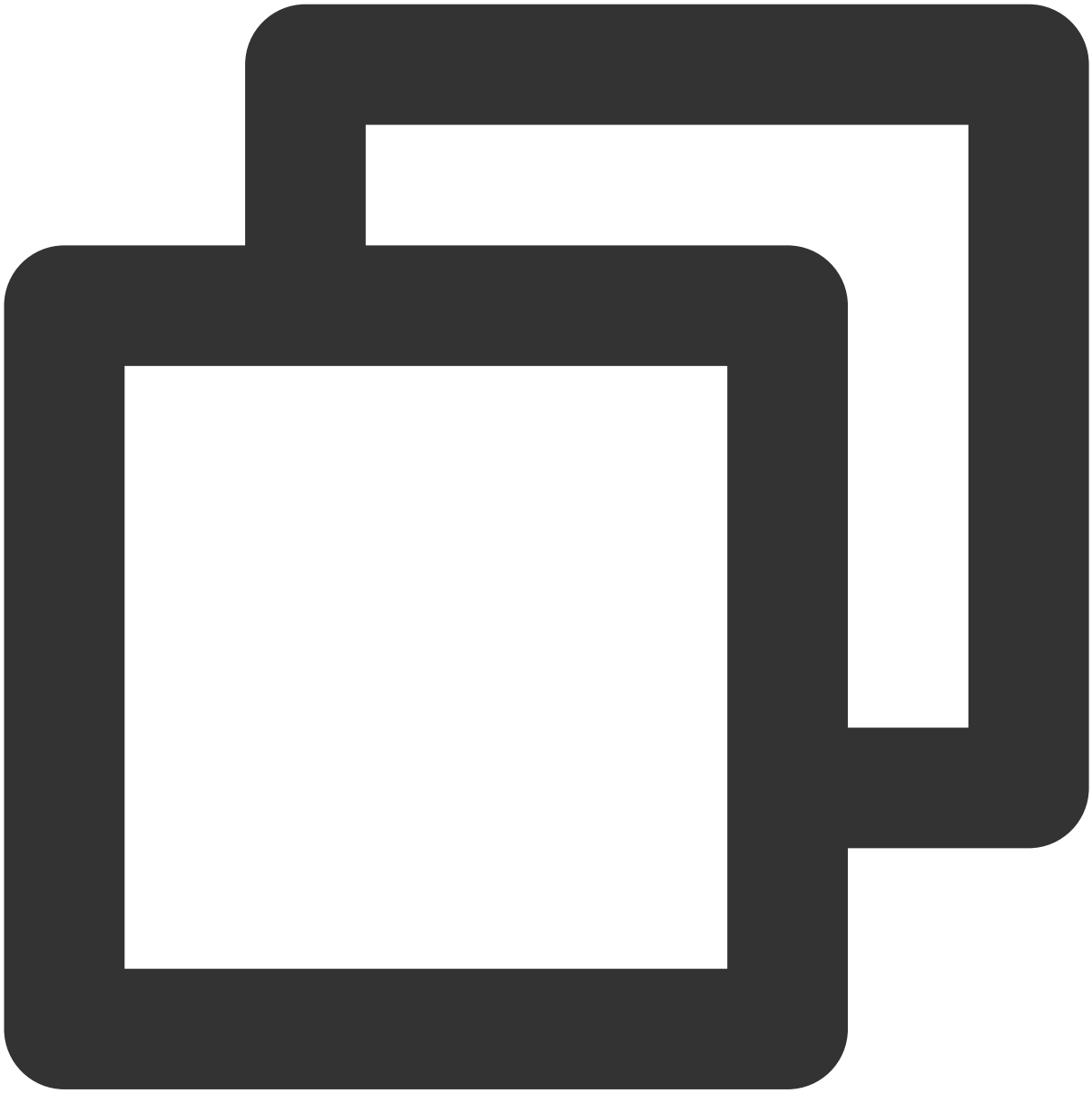
参数名	类型	描述
result	bool	设置焦点是否成功

setConfig

说明

给播放器进行配置。

接口



```
Future<void> setConfig(FTXVodPlayConfig config) async ;
```

参数说明

参数名	类型	描述
config	FTXVodPlayConfig	请参考 `FTXVodPlayConfig类`

返回值说明

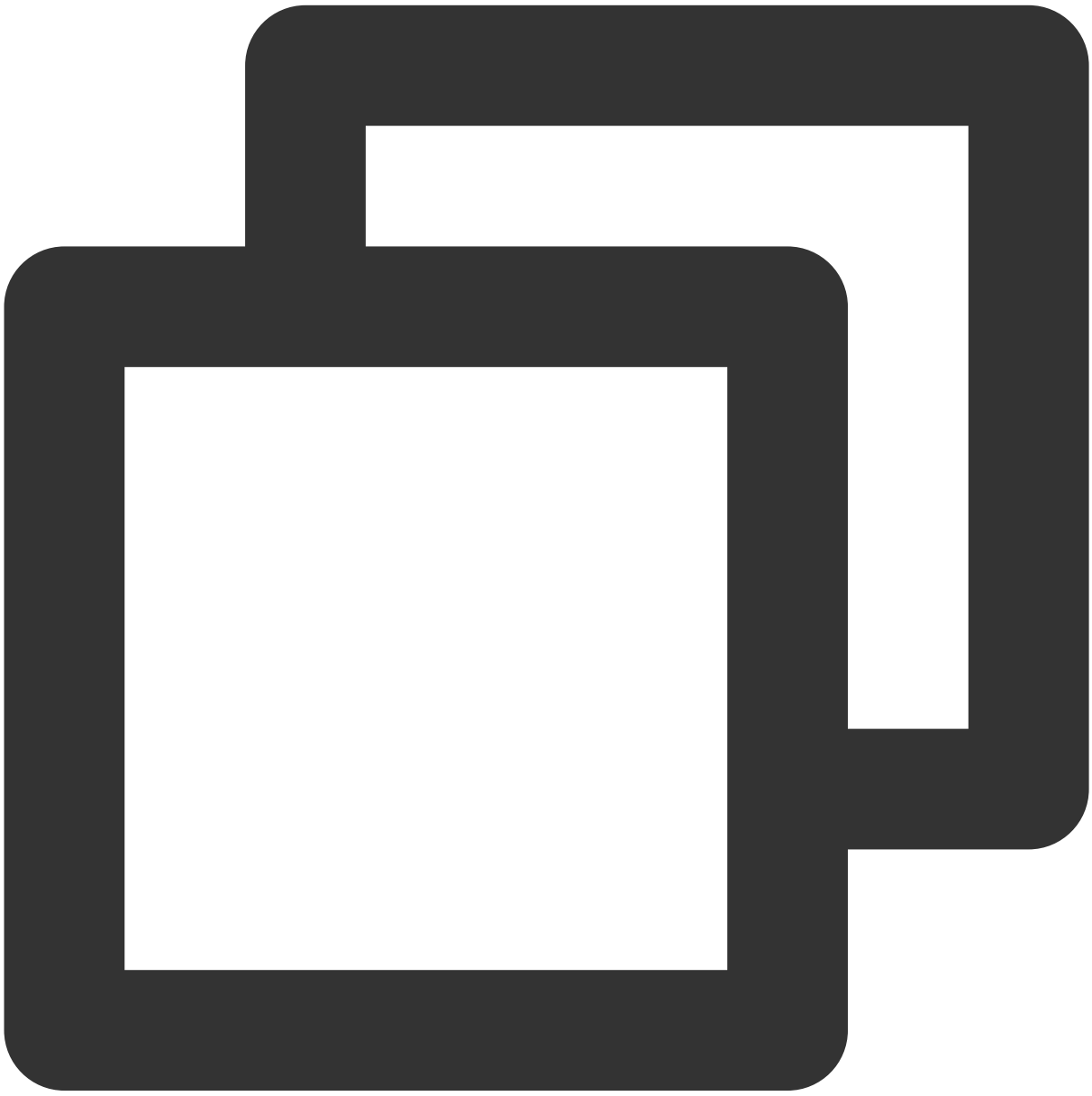
无

getCurrentPlaybackTime

说明

获得当前播放时间，单位**秒**。

接口



```
Future<double> getCurrentPlaybackTime() async;
```

参数说明

无

返回值说明

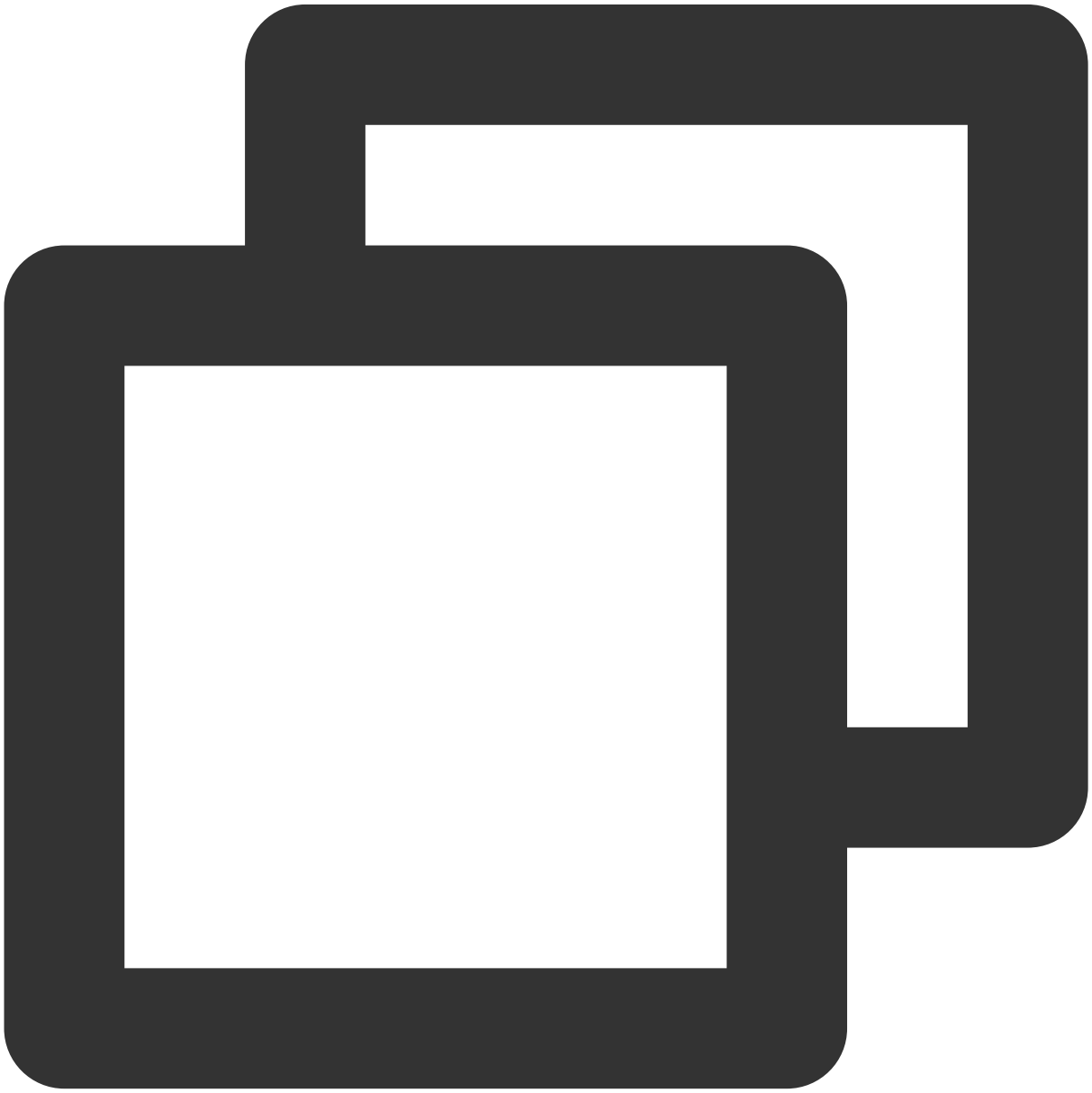
参数名	类型	描述
playbackTime	double	当前播放时间，单位 秒

getBufferDuration

说明

获得当前视频已缓存的时间，单位**秒**。

接口



```
Future<double> getBufferDuration();
```

参数说明

无

返回值说明

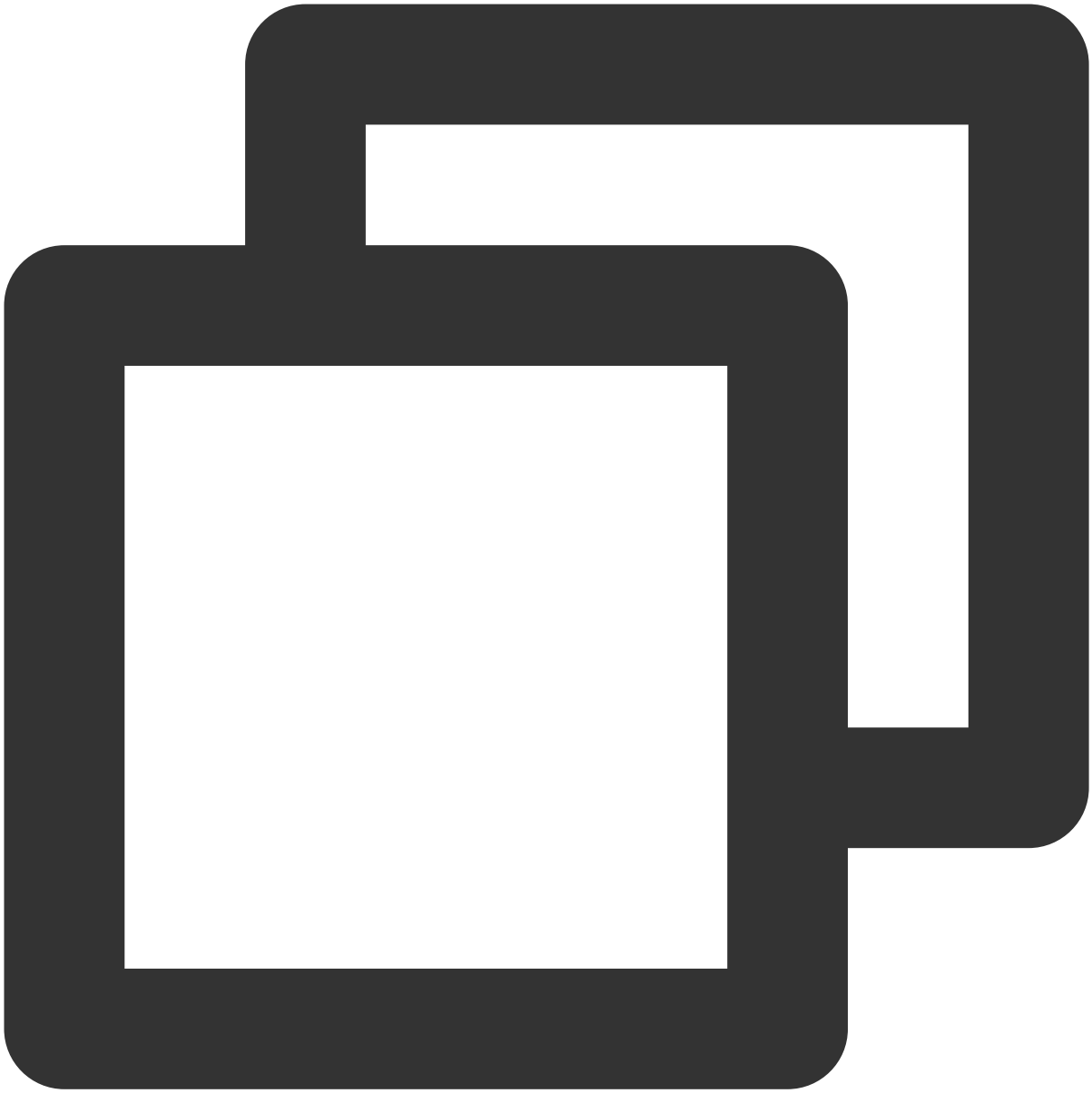
参数名	类型	描述
playbackTime	double	当前视频已缓存的时间，单位 秒

getPlayableDuration

说明

获得当前正在播放视频的可播放时间，单位**秒**。

接口



```
Future<double> getPlayableDuration() async;
```

参数说明

无

返回值说明

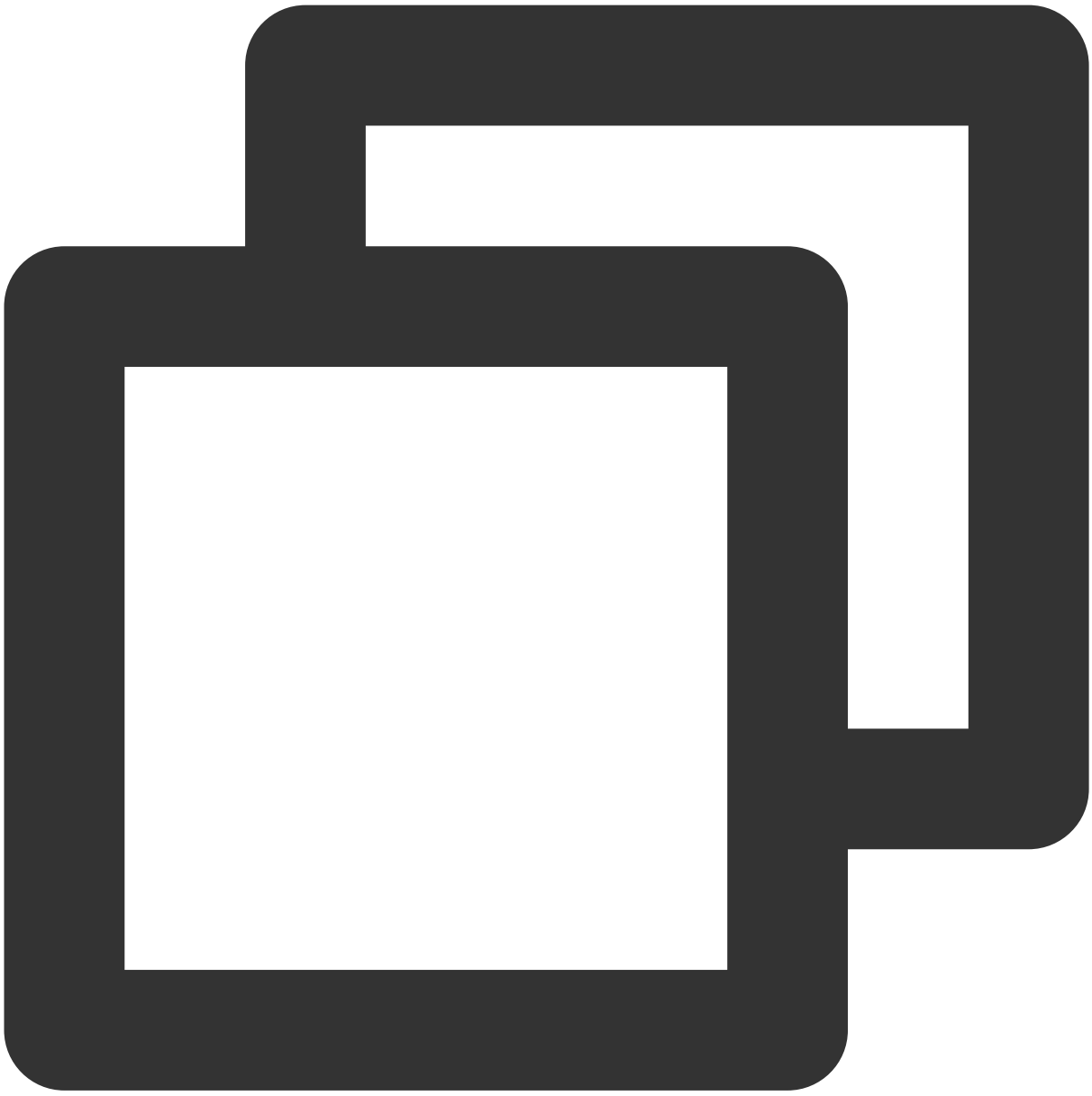
参数名	类型	描述
playableDuration	double	当前视频可播放时，单位 秒

getWidth

说明

获得当前正在播放视频的宽度。

接口



```
Future<int> getWidth() async;
```

参数说明

无

返回值说明

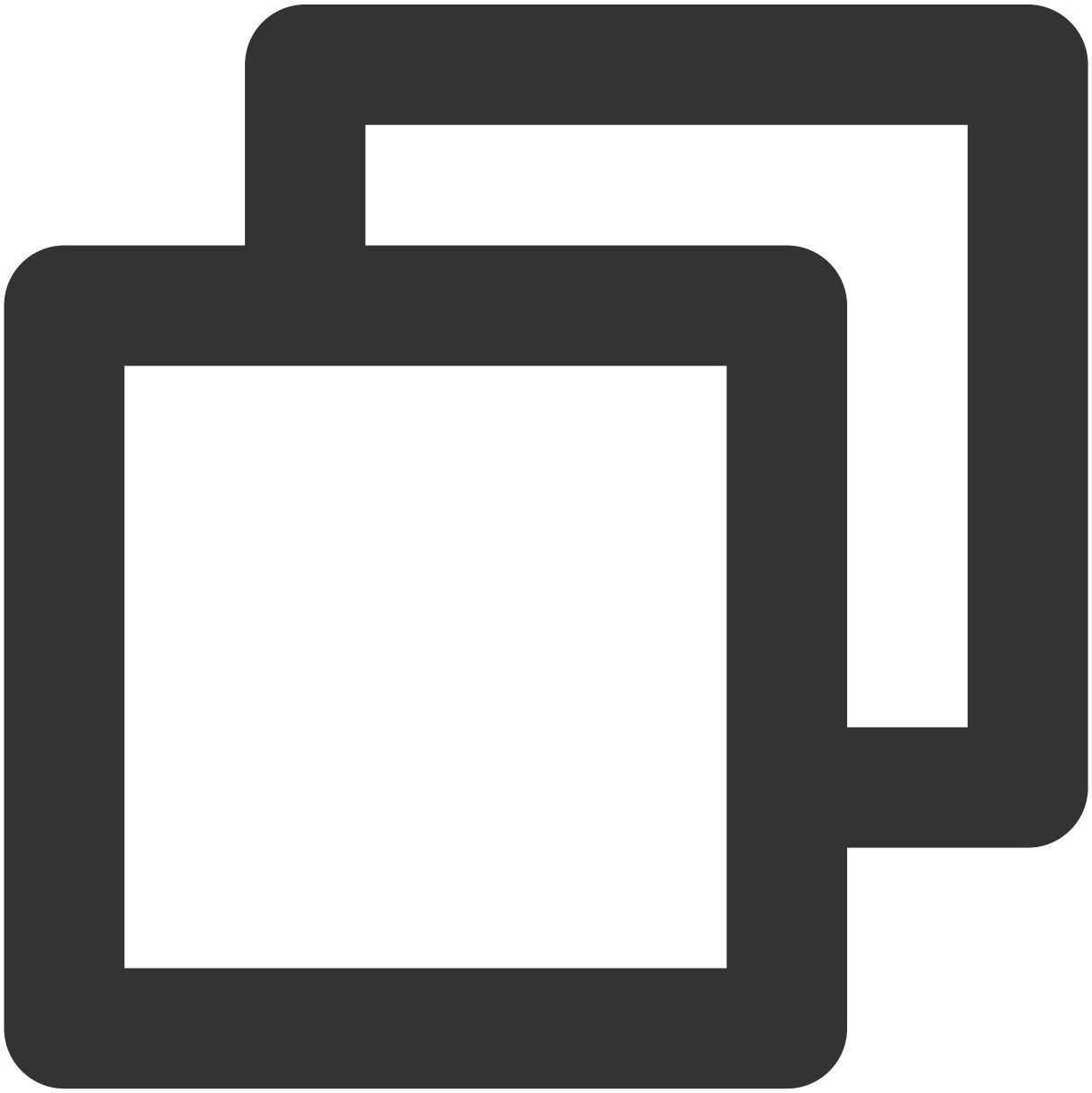
参数名	类型	描述
width	int	当前视频宽度

getHeight

说明

获得当前正在播放视频的高度。

接口



```
Future<int> getHeight() async;
```

参数说明

无

返回值说明

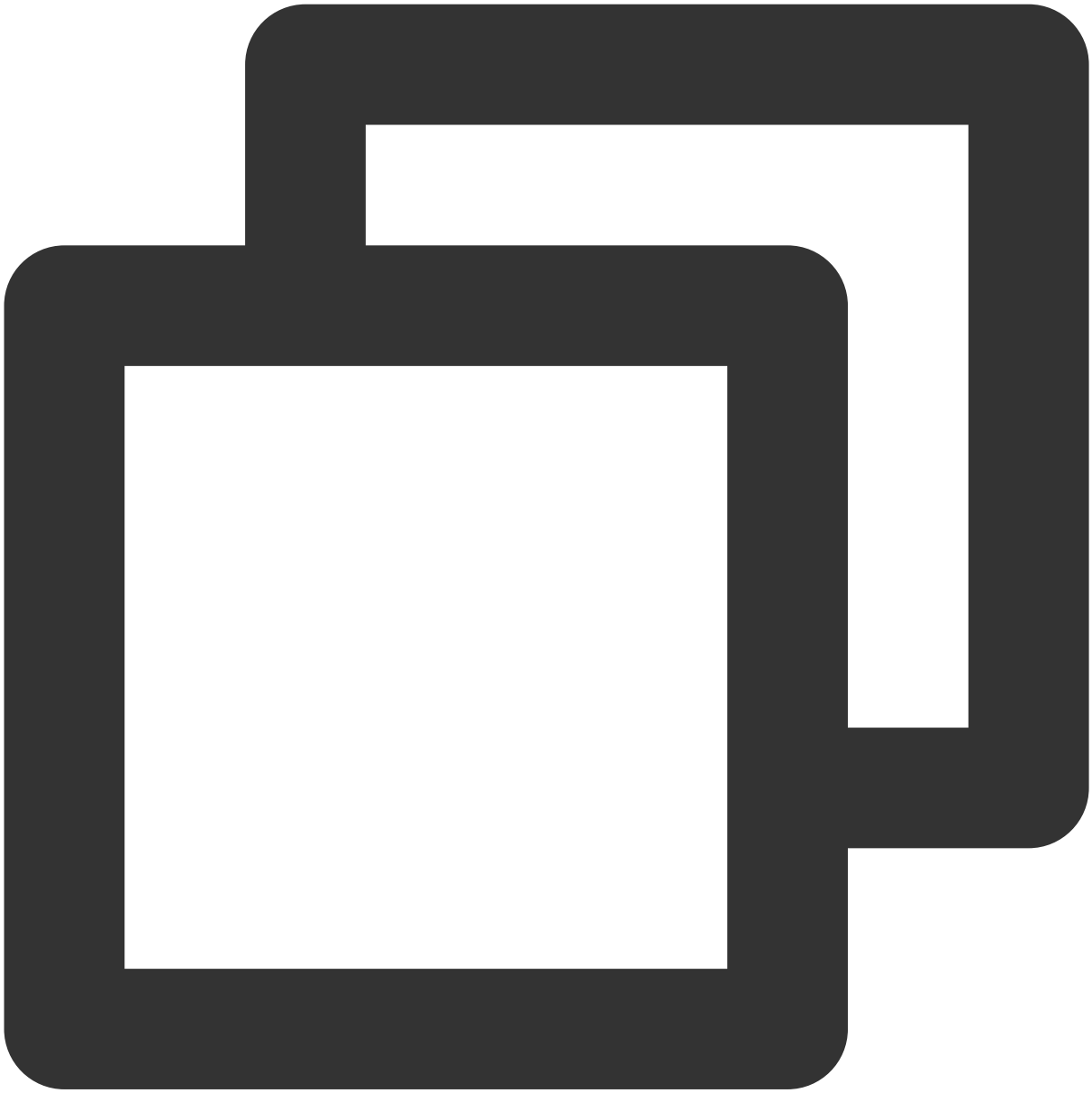
参数名	类型	描述
height	int	当前视频高度

setToken

说明

加密 HLS 的 token。设置此值后，播放器自动在 URL 中的文件名之前增加 `voddrm.token` 。

接口



```
Future<void> setToken(String? token) async;
```

参数说明

参数名	类型	描述
token	String	播放视频的token

返回值说明

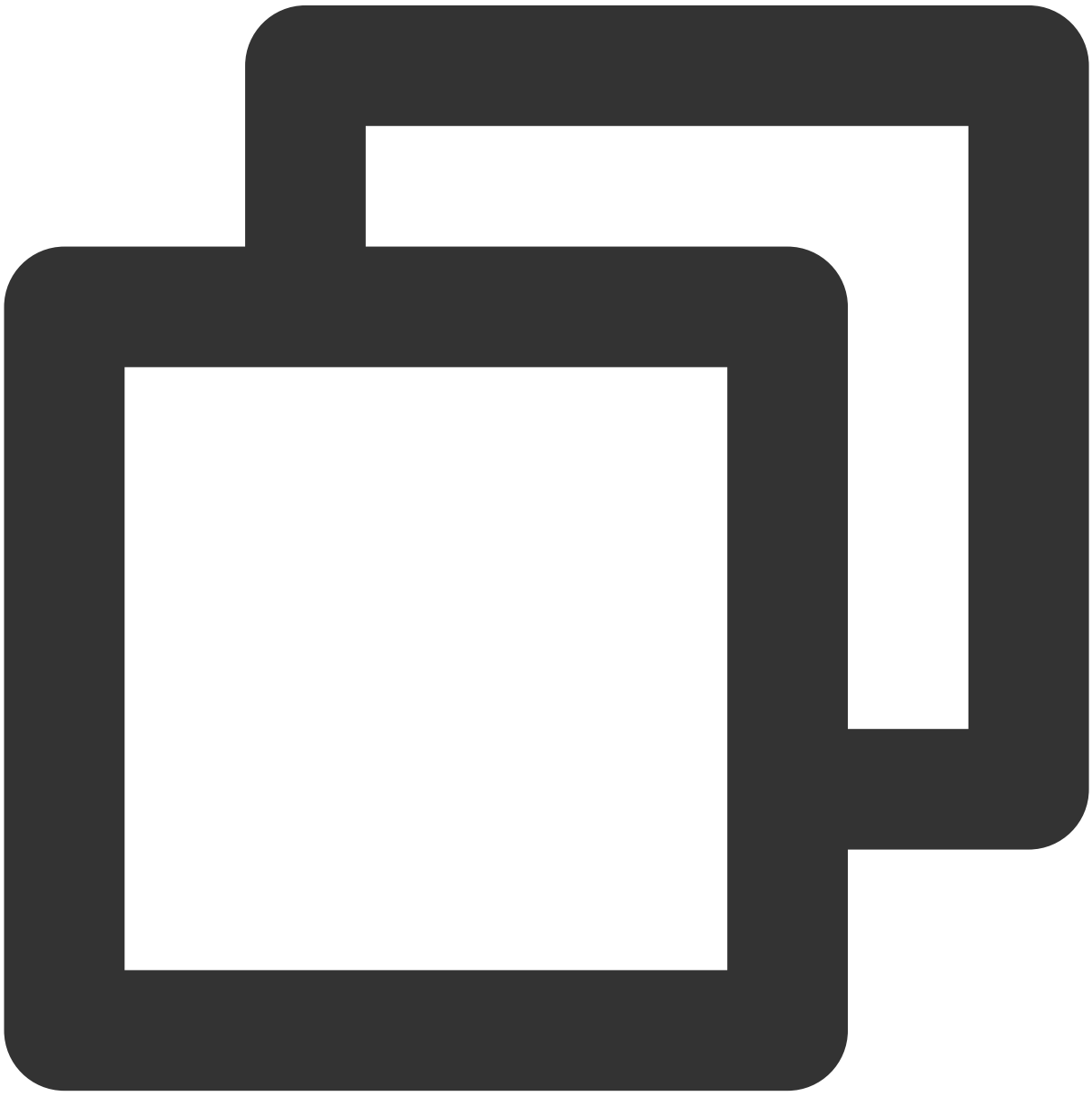
无

isLoop

说明

获得当前播放器是否循环播放的状态。

接口



```
Future<bool> isLoop() async;
```

参数说明

无

返回值说明

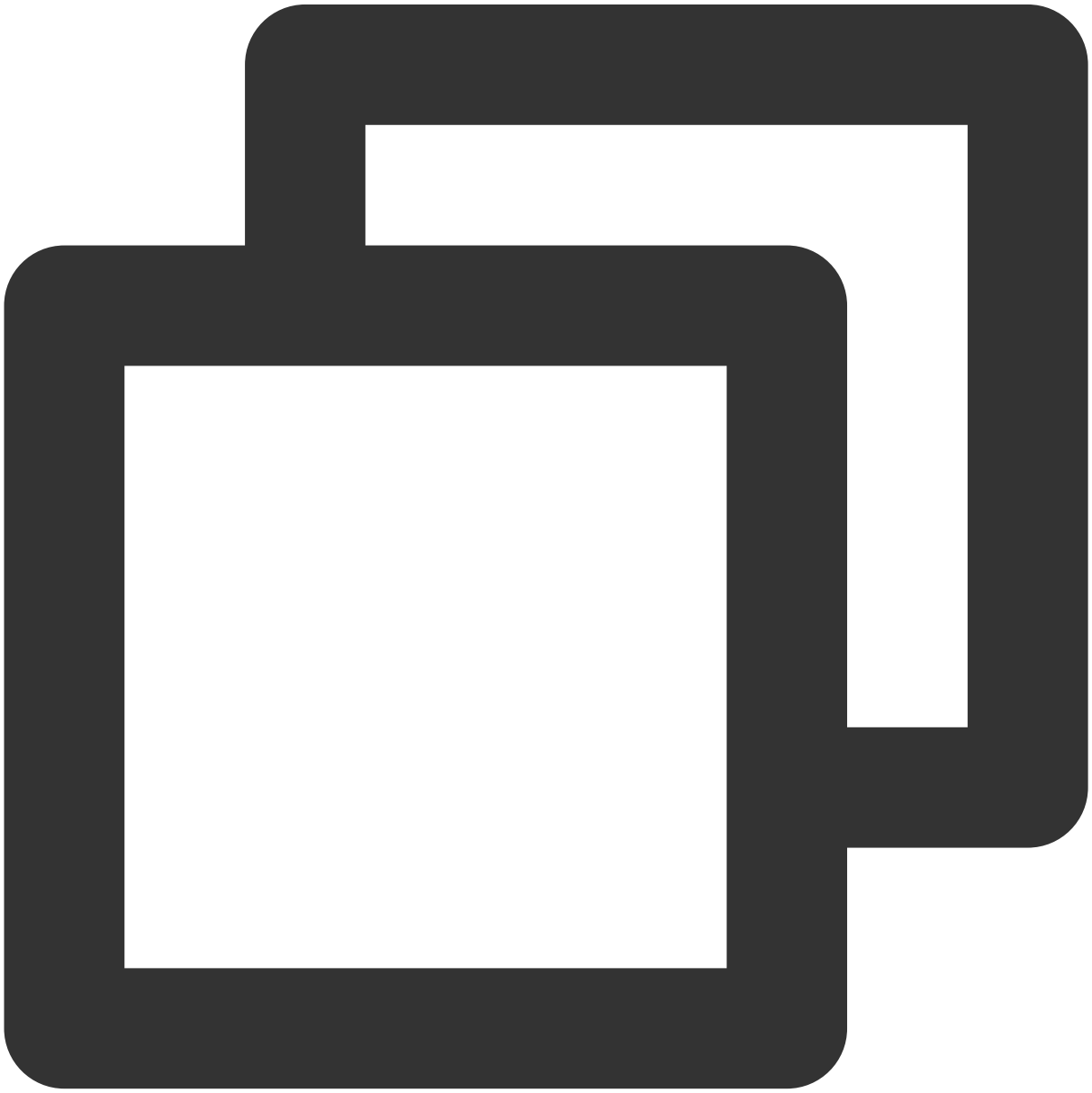
参数名	类型	描述
isLoop	bool	播放器是否处于循环播放状态

enableHardwareDecode

说明

开启/关闭硬解播放，设置后不会立即生效，需要重新播放才可生效。

接口



```
Future<bool> enableHardwareDecode(bool enable);
```

参数说明

参数名	类型	描述
enable	bool	是否开启硬解

返回值说明

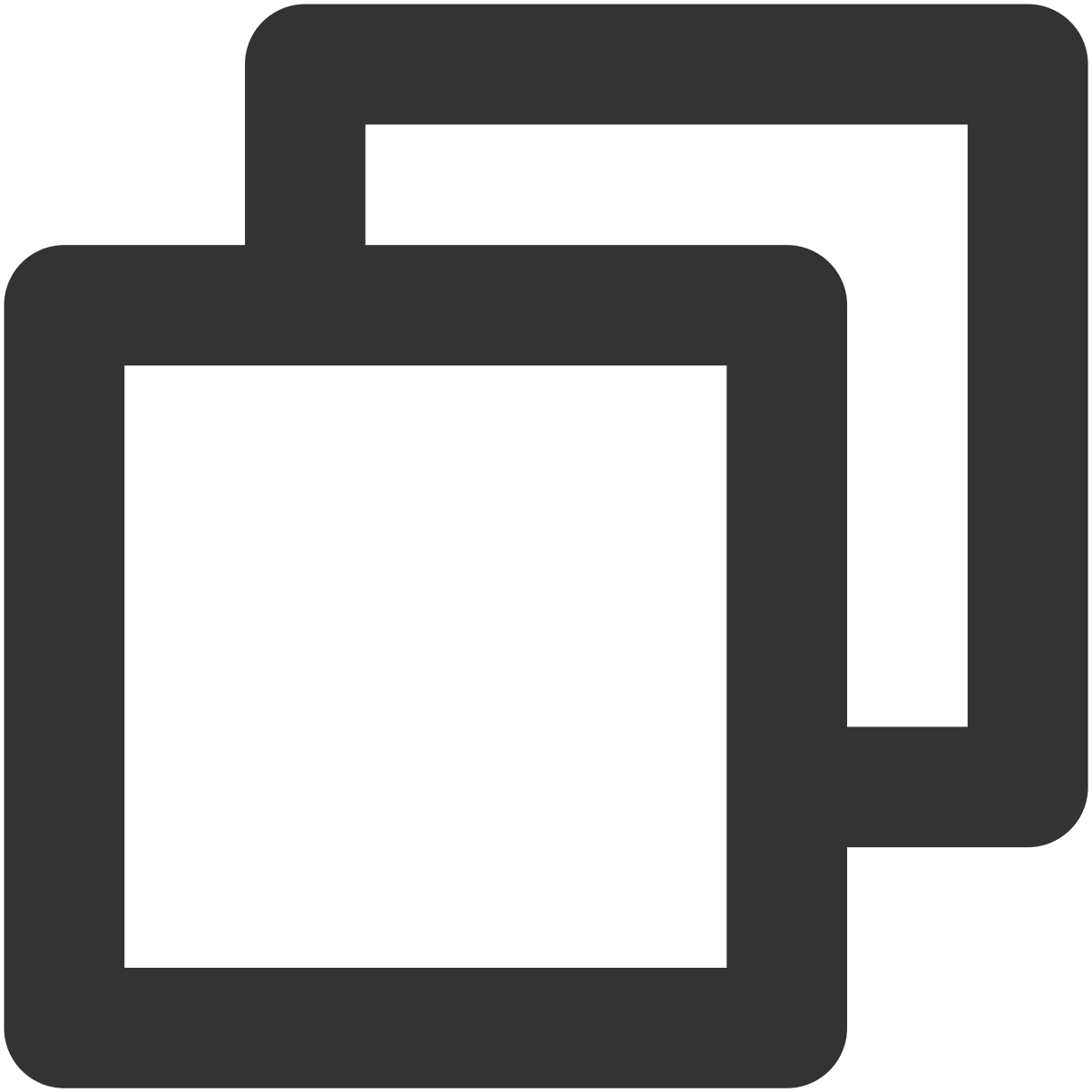
参数名	类型	描述
result	bool	硬解/软解设置结果

dispose

说明

销毁controller，调用该方法会销毁掉所有通知事件，释放掉播放器。

接口



```
Future<void> dispose() async;
```

参数说明

无

返回值说明

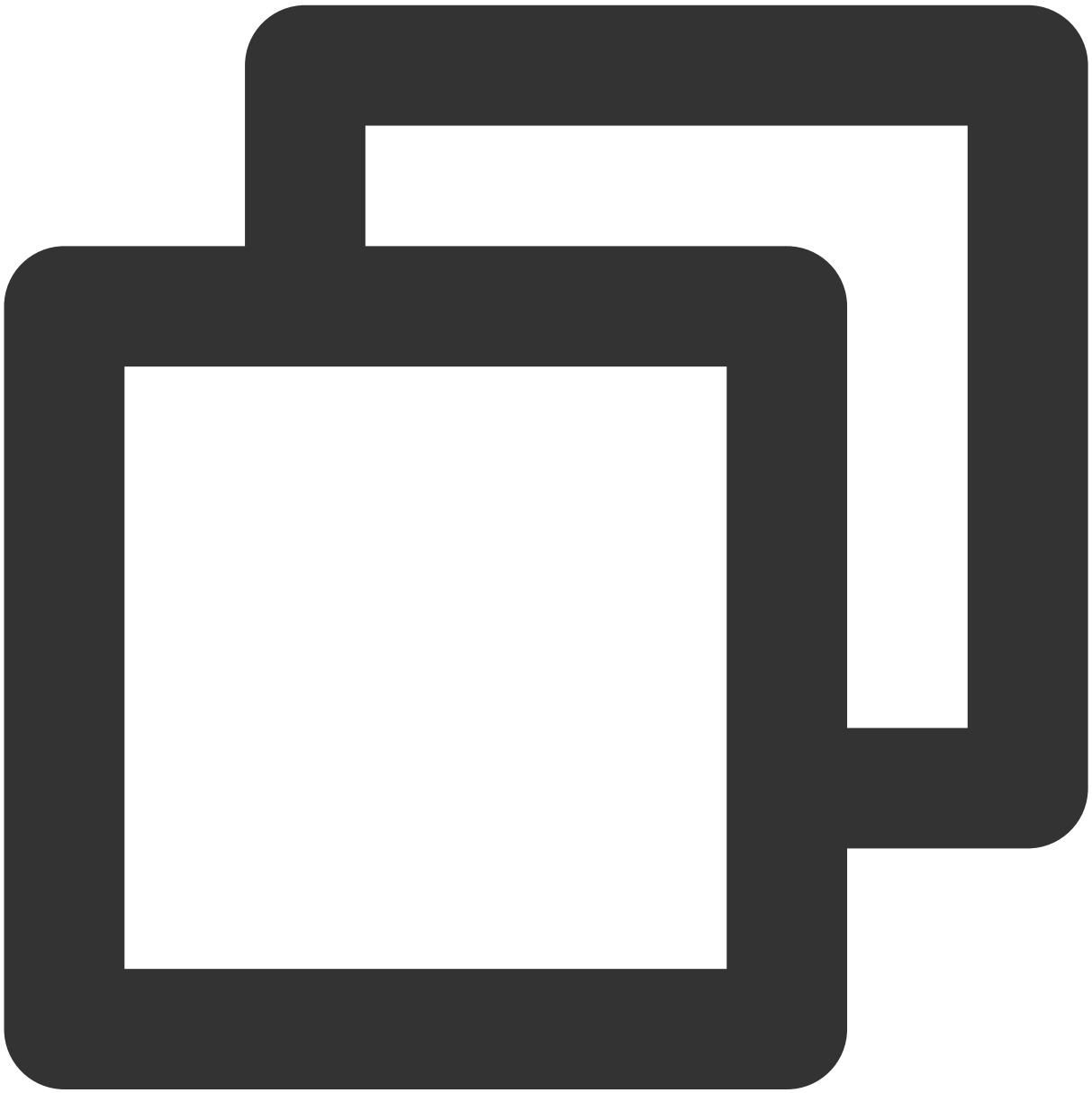
无

getDuration

说明

获取视频总时长。

接口



```
Future<double> getDuration() async;
```

参数说明

无

返回值说明

参数名	类型	描述

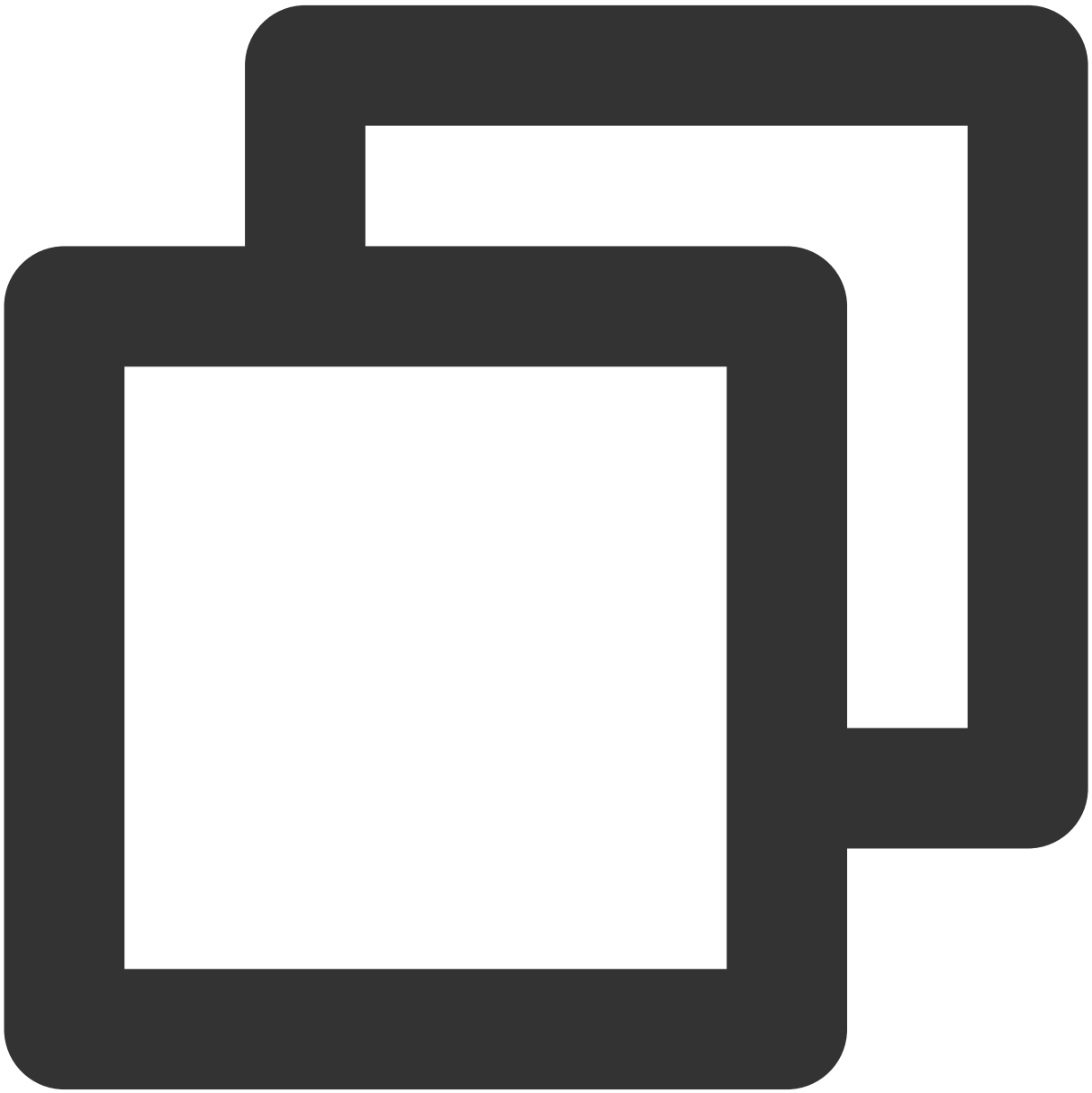
duration	double	视频总时长，单位 秒

enterPictureInPictureMode

说明

进入画中画模式。

接口



```
Future<int> enterPictureInPictureMode({String? backIconForAndroid, String? playIcon
```

参数说明

该参数只适用于 android 平台

参数名	类型	描述
backIcon	String	回退按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
playIcon	String	播放按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
pauseIcon	String	暂停按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
forwardIcon	String	快进按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标

返回值说明

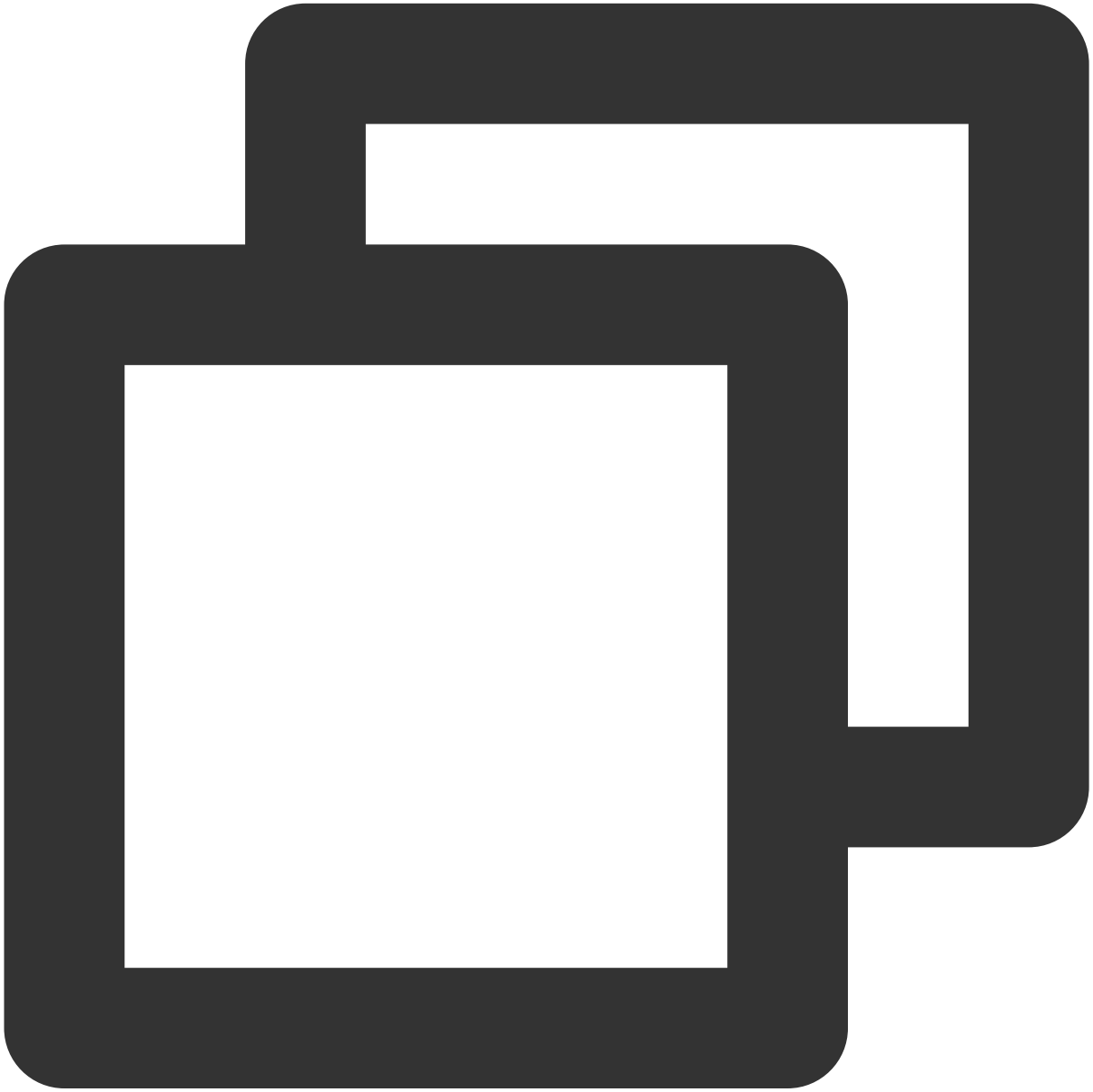
参数名	值	描述
NO_ERROR	0	启动成功，没有错误
ERROR_PIP_LOWER_VERSION	-101	android 版本过低，不支持画中画模式
ERROR_PIP_DENIED_PERMISSION	-102	画中画模式权限未打开，或者当前设备不支持画中画
ERROR_PIP_ACTIVITY_DESTROYED	-103	当前界面已经销毁
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	设备或系统版本不支持（iPad iOS9+ 才支持 PIP），只适用于 iOS
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	播放器不支持，只适用于 iOS
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	视频不支持，只适用于 iOS
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	PIP控制器不可用，只适用于 iOS
ERROR_IOS_PIP_FROM_SYSTEM	-108	PIP控制器报错，只适用于 iOS
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	播放器对象不存在，只适用于 iOS
ERROR_IOS_PIP_IS_RUNNING	-110	PIP功能已经运行，只适用于 iOS
ERROR_IOS_PIP_NOT_RUNNING	-111	PIP功能没有启动，只适用于 iOS

initImageSprite

说明

初始化视频雪碧图。

接口



```
Future<void> initImageSprite(String? vvtUrl, List<String>? imageUrls) async;
```

参数说明

参数名	类型	描述
vvtUrl	String	雪碧图 web vtt 描述文件下载 URL

imageUrls	List<String>	雪碧图图片下载 URL

返回值说明

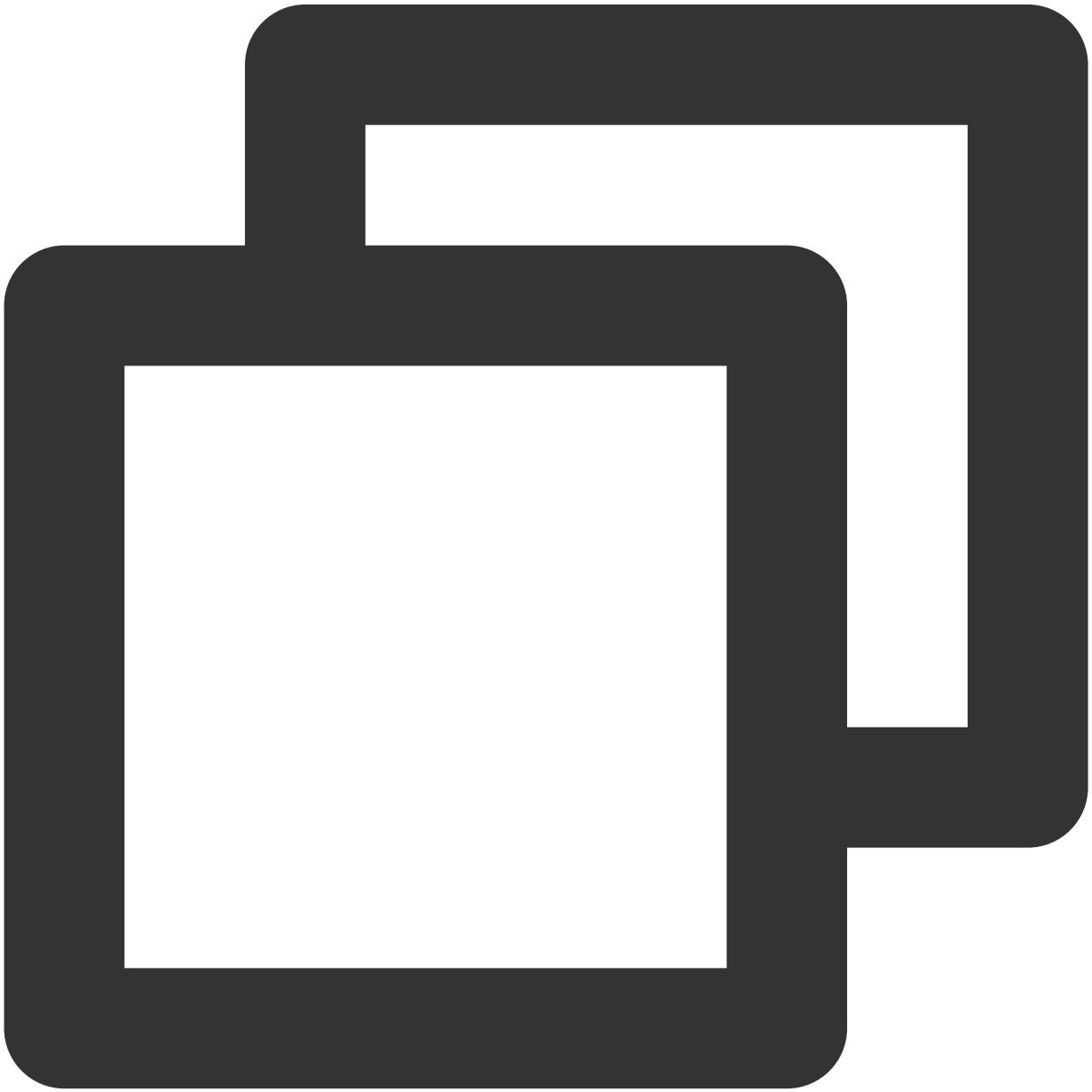
无

getImageSprite

说明

获取加载的雪碧图。

接口



```
Future<Uint8List?> getImageSprite(double time) async;
```

参数说明

参数名	类型	描述
time	double	时间点，单位 秒

返回值说明

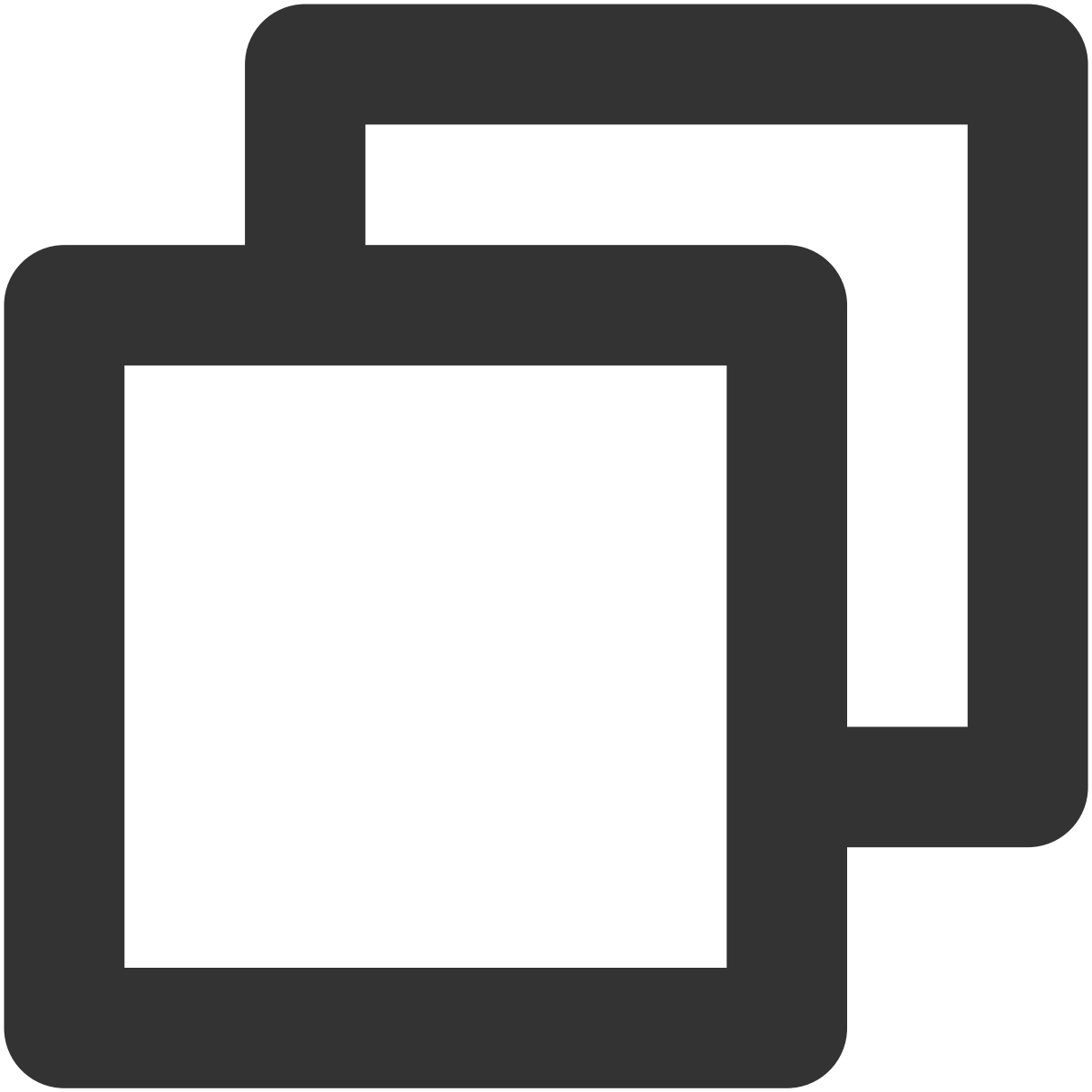
参数名	类型	描述
thumb	Uint8List	雪碧图

exitPictureInPictureMode

说明

退出画中画，如果该播放器处于画中画模式。

接口



```
Future<void> exitPictureInPictureMode() async;
```

参数说明

无

返回值说明

无

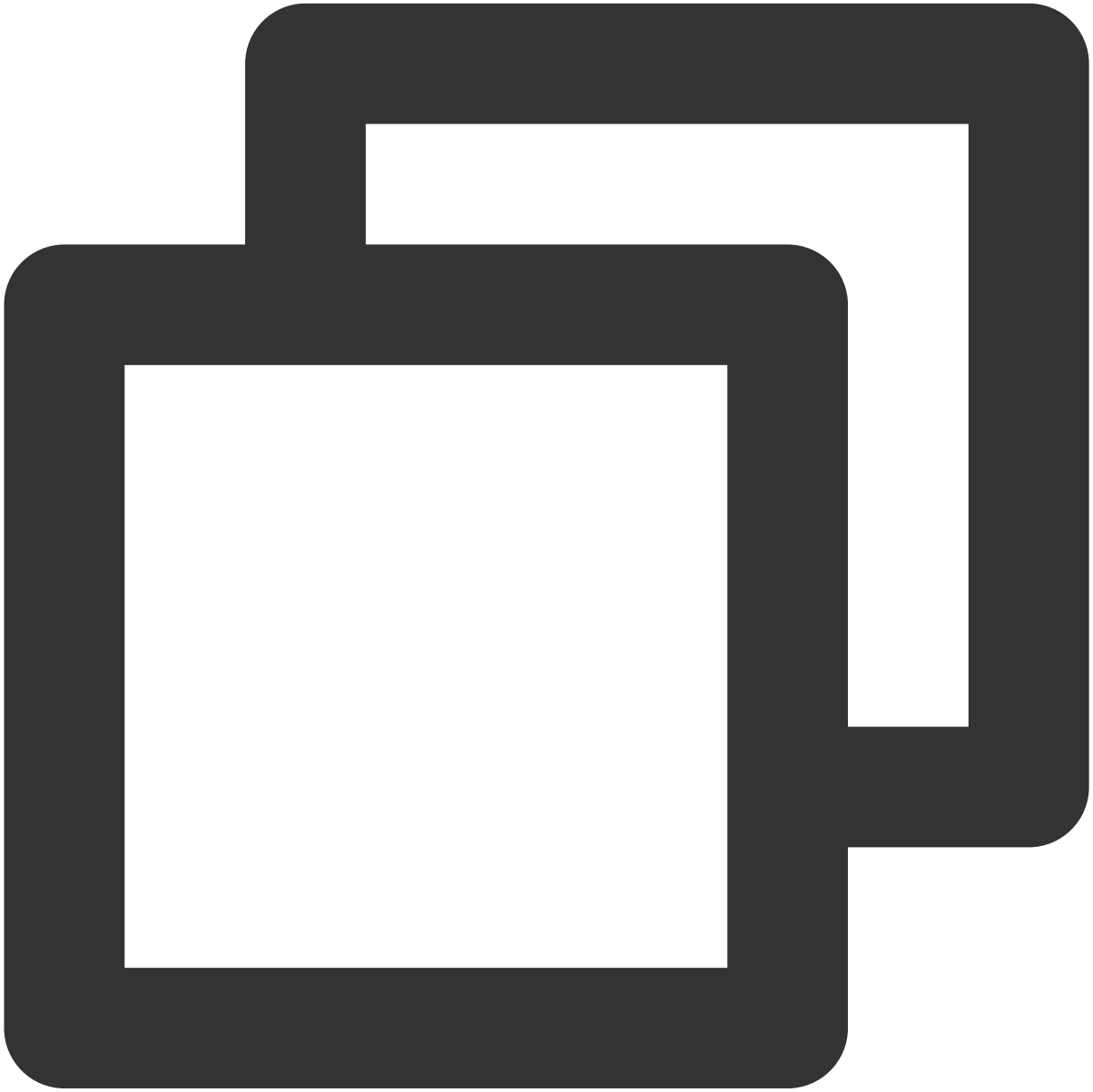
addSubtitleSource

说明

添加外挂字幕。

注意：此功能需要播放器高级版 11.7 版本开始支持。

接口



```
Future<void> addSubtitleSource(String url, String name, {String? mimeType}) async;
```

参数说明

参数名	类型	描述
url	String	字幕url

name	String	字幕名称
mimeType	String	字幕类型，支持SRT（TXVodPlayEvent.VOD_PLAY_MIMETYPE_TEXT_SRT）和VVT（TXVodPlayEvent.VOD_PLAY_MIMETYPE_TEXT_VTT）格式

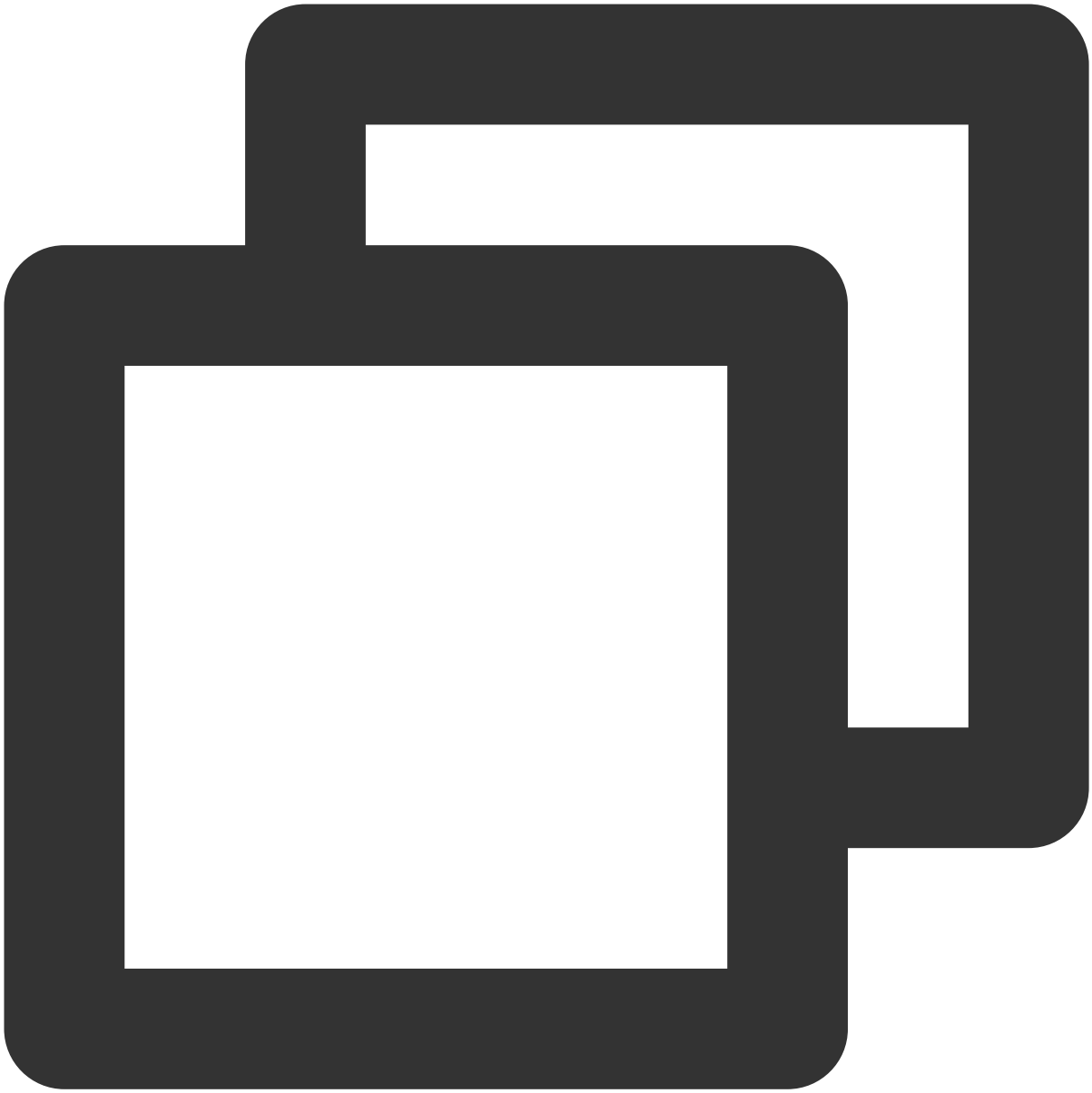
getSubtitleTrackInfo

说明

返回字幕轨道信息列表。

注意：此功能需要播放器高级版 11.7 版本开始支持。

接口



```
Future<List<TXTrackInfo>> getSubtitleTrackInfo() async;
```

参数说明

TXTrackInfo类：

参数名	类型	描述
trackType	int	轨道类型。取值有： 视频轨：TX_VOD_MEDIA_TRACK_TYPE_VIDEO = 1 音频轨： TX_VOD_MEDIA_TRACK_TYPE_AUDIO = 2 字幕轨：TX_VOD_MEDIA_TRACK_TYPE_SUBTITLE = 3

trackIndex	int	轨道index
name	String	轨道名字
isSelected	bool	当前轨道是否被选中
isExclusive	bool	如果是true，该类型轨道每个时刻只有一条能被选中，如果是false，该类型轨道可以同时选中多条
isInternal	bool	当前的轨道是否是内部原始轨道

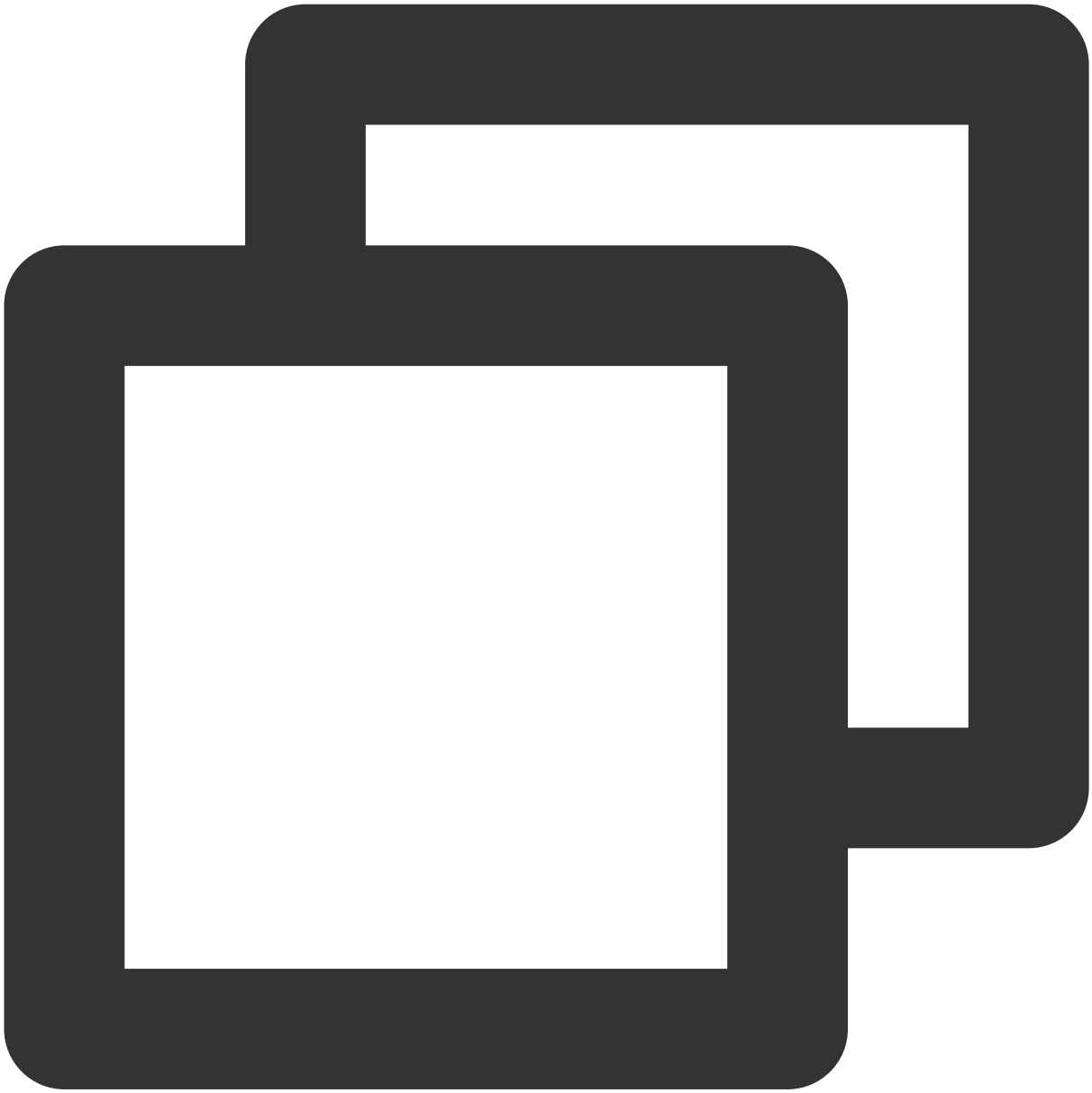
getAudioTrackInfo

说明

返回字幕轨道信息列表。

注意：此功能需要播放器高级版 11.7 版本开始支持。

接口



```
Future<List<TXTrackInfo>> getAudioTrackInfo() async;
```

参数说明

参考TXTrackInfo类

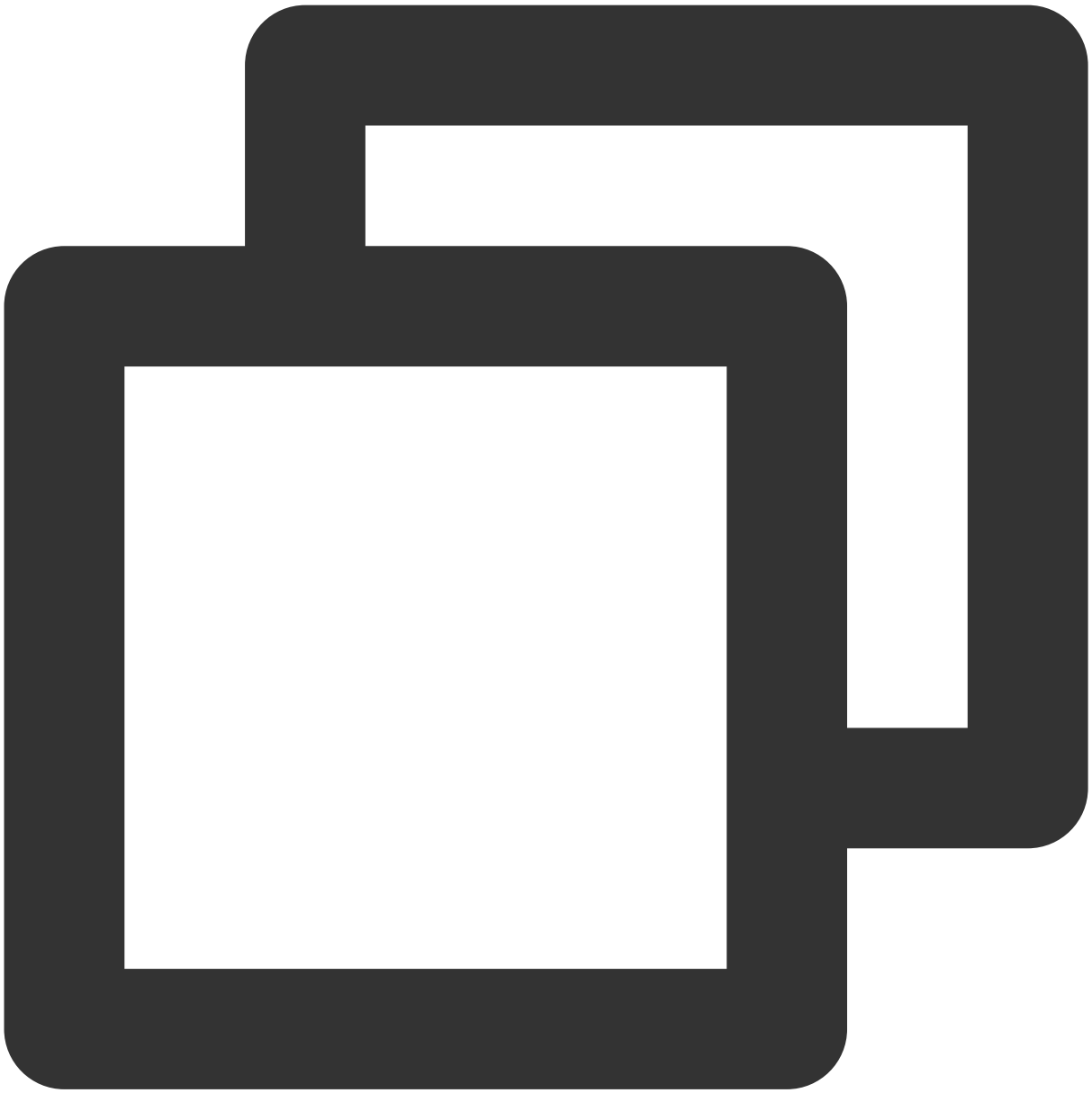
selectTrack

说明

选择轨道。

注意：此功能需要播放器高级版 11.7 版本开始支持。

接口



```
Future<void> selectTrack(int trackIndex) async;
```

参数说明

参数名	类型	描述
trackIndex	int	轨道index，trackIndex 轨道index，通过[TXTrackInfo]的trackIndex获取。

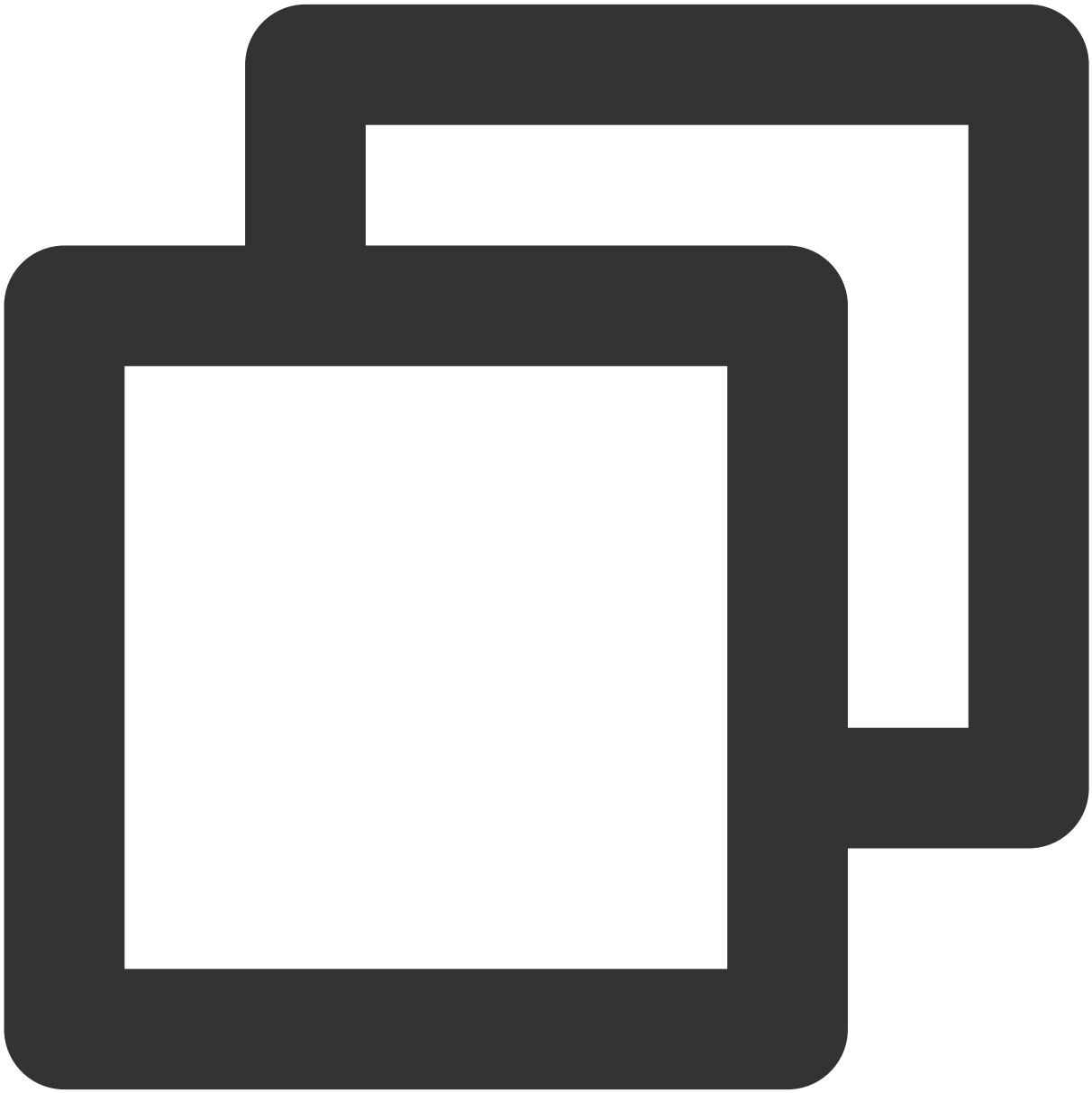
deselectTrack

说明

取消选择轨道。

注意：此功能需要播放器高级版 11.7 版本开始支持。

接口



```
Future<void> deselectTrack(int trackIndex) async;
```

参数说明

参数名	类型	描述

trackIndex	int	轨道index，trackIndex 轨道index，通过[TXTrackInfo]的trackIndex获取。

FTXVodPlayConfig类

属性配置说明

参数名	类型	描述
connectRetryCount	int	播放器重连次数，当 SDK 与服务器异常断开连接时,SDK 会尝试与服务器重连.通过该值设置 SDK 重连次数
connectRetryInterval	int	播放器重连间隔，当 SDK 与服务器异常断开连接时,SDK 会尝试与服务器重连.通过该值设置两次重连间隔时间
timeout	int	播放器连接超时时间
playerType	int	播放器类型,0 点播，1 直播，2 直播回看
headers	Map	自定义 http headers
enableAccurateSeek	bool	是否精确 seek，默认 true
autoRotate	bool	播放mp4文件时，若设为 true 则根据文件中的旋转角度自动旋转。旋转角度可在 PLAY_EVT_CHANGE_ROTATION 事件中获得。默认 true
smoothSwitchBitrate	bool	平滑切换多码率HLS，默认 false。设为 false时，可提高多码率地址打开速度; 设为 true，在 IDR 对齐时可平滑切换码率
cacheMp4ExtName	String	缓存 mp4文件扩展名,默认mp4
progressInterval	int	设置进度回调间隔,若不设置，SDK 默认间隔0.5秒回调一次,单位毫秒
maxBufferSize	int	最大播放缓冲大小，单位 MB。此设置会影响 playableDuration，设置越大，提前缓存的越多
maxPreloadSize	int	预加载最大缓冲大小，单位：MB
firstStartPlayBufferTime	int	首缓需要加载的数据时长，单位ms，默认值为100ms
nextStartPlayBufferTime	int	缓冲时（缓冲数据不够引起的二次缓冲，或者seek引起的拖动缓冲）最少要缓存多长的数据才能结束缓冲，单位ms，默认值为250ms
overlayKey	String	HLS安全加固加解密 key
overlayIv	String	HLS安全加固加解密 Iv
extInfoMap	Map	一些不必周知的特殊配置

enableRenderProcess	bool	是否允许加载后渲染后处理服务,默认开启，开启后超分插件如果存在，默认加载
preferredResolution	int	优先播放的分辨率，preferredResolution = width * height
mediaType	int	设置媒资类型, 默认为 auto 类型。可选值有： TXVodConstants#MEDIA_TYPE_AUTO，AUTO 类型（默认值，自适应码率播放暂不支持）。 TXVodConstants#MEDIA_TYPE_HLS_VOD，HLS 点播媒资。 TXVodConstants#MEDIA_TYPE_HLS_LIVE，HLS 直播媒资。 TXVodConstants#MEDIA_TYPE_FILE_VOD，MP4 等通用文件点播媒资（从 11.7 版本开始支持）。 TXVodConstants#MEDIA_TYPE_DASH_VOD，DASH 点播媒资（从 11.7 版本开始支持）。

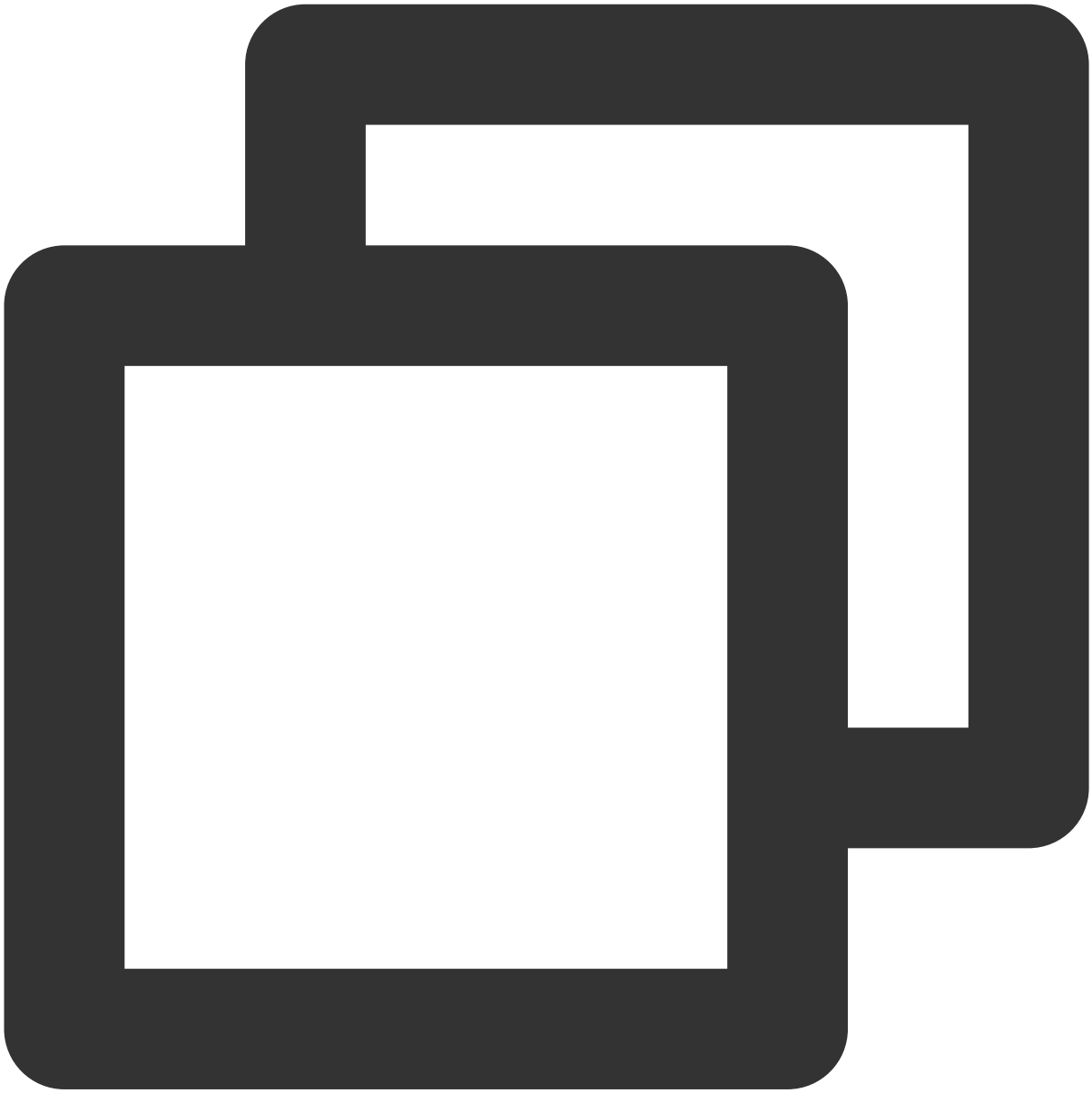
TXLivePlayerController类

initialize

说明

初始化 controller，请求分配共享纹理。

接口



```
Future<void> initialize({bool? onlyAudio}) async;
```

参数说明

参数名	类型	描述
onlyAudio	bool	选填，是否是纯音频播放器

返回值说明

无

startLivePlay

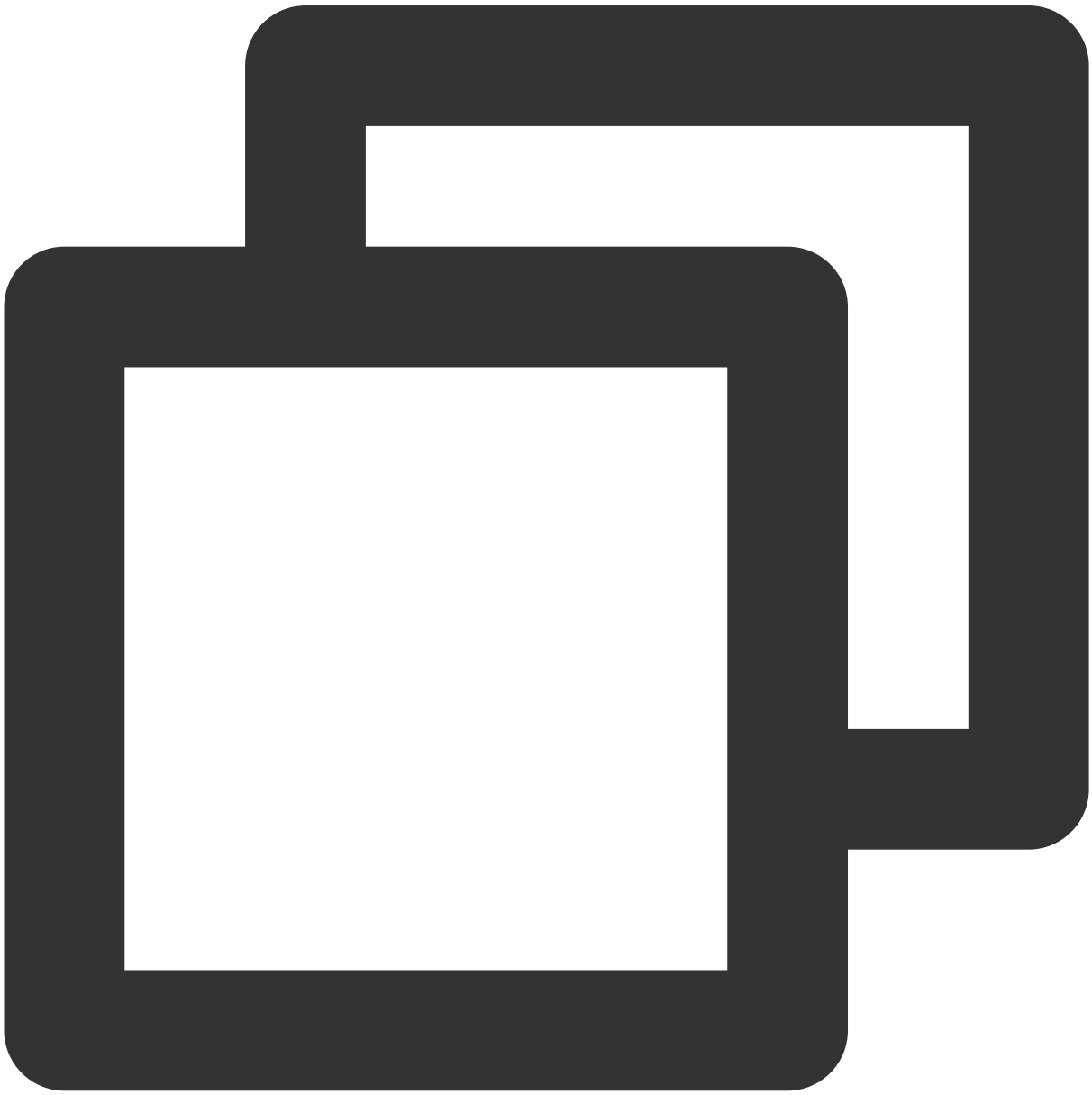
注意

10.7版本开始，startPlay变更为startLivePlay，需要通过 {@link SuperPlayerPlugin#setGlobalLicense} 设置 Licence 后方可成功播放， 否则将播放失败（黑屏），全局仅设置一次即可。直播 Licence、短视频 Licence 和视频播放 Licence 均可使用，若您暂未获取上述 Licence ，可快速免费申请测试版 Licence 以正常播放，正式版 License 需 购买。

说明

通过播视频 url 进行播放。

接口



```
Future<bool> play(String url, {int? playType}) async;
```

参数说明

参数名	类型	描述
url	String	要播放的视频 url
playType	int	选填，支持的播放类型，默认 RTMP 直播，支持 LIVE_RTMP、LIVE_FLV、LIVE_RTMP_ACC以及VOD_HLS，详见 TXPlayType

返回值说明

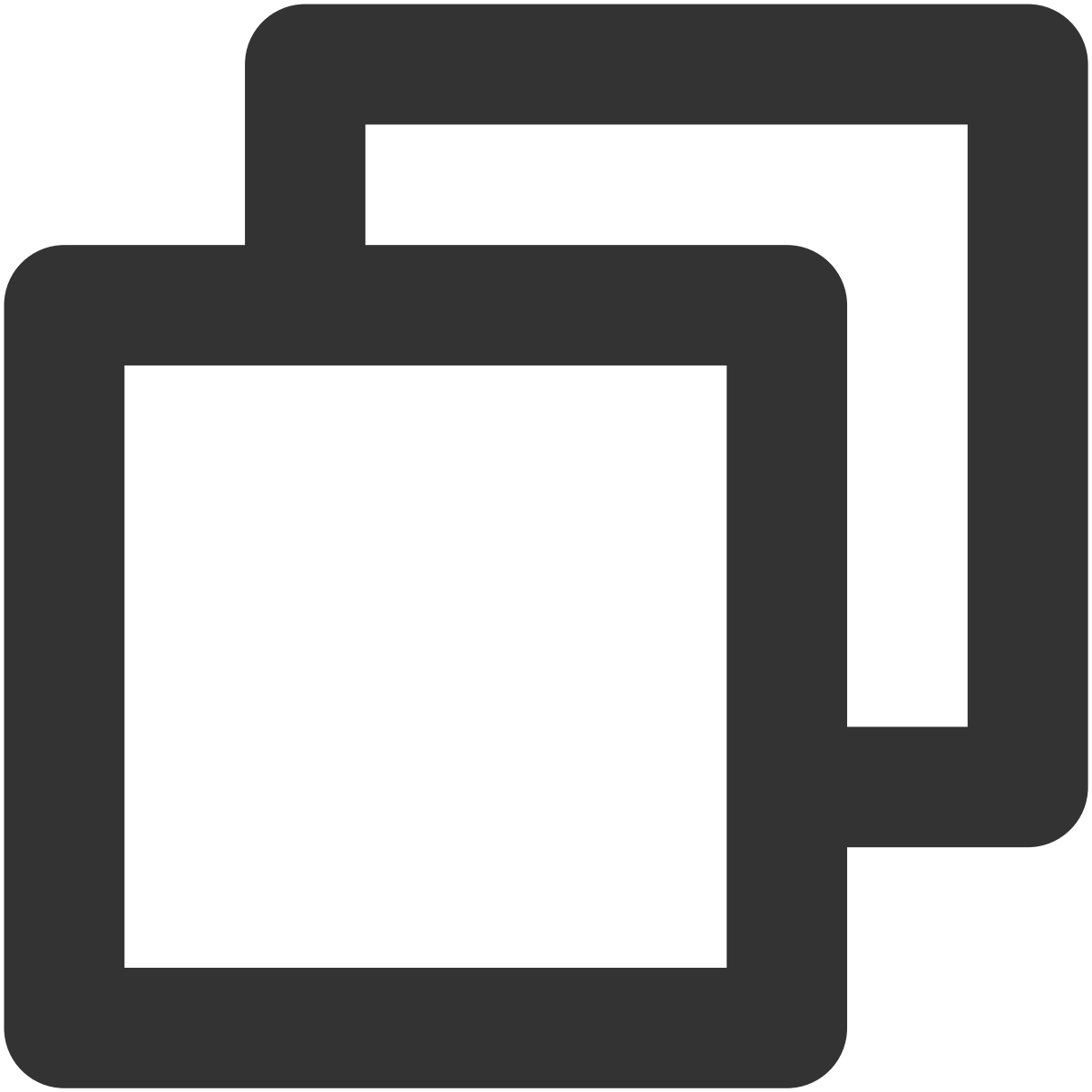
参数名	类型	描述
result	bool	创建是否成功

pause

说明

暂停当前正在播放的视频。

接口



```
Future<void> pause() async;
```

参数说明

无

返回值说明

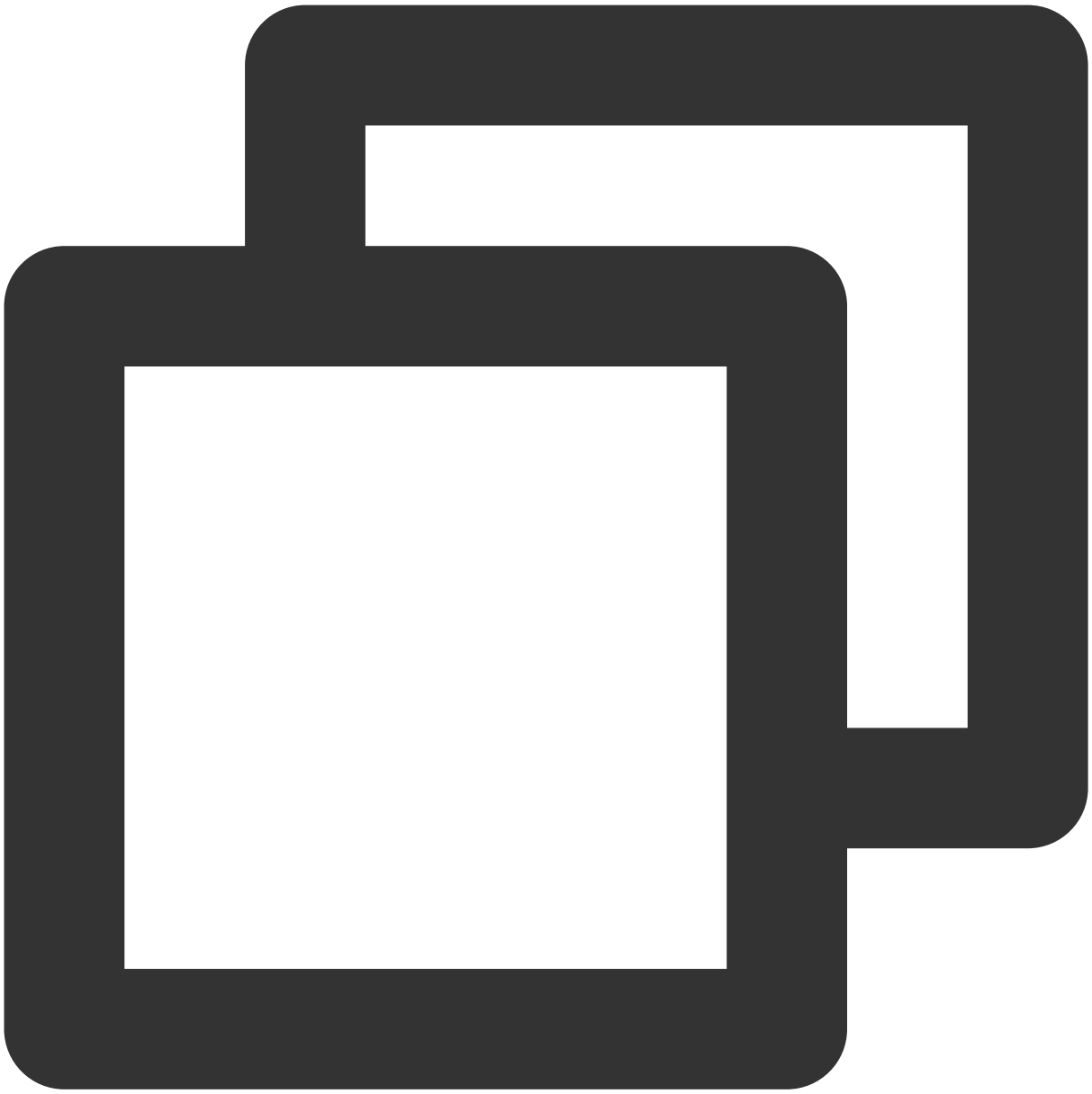
无

resume

说明

将当前处于暂停状态的视频恢复播放。

接口



```
Future<void> resume() async;
```

参数说明

无

返回值说明

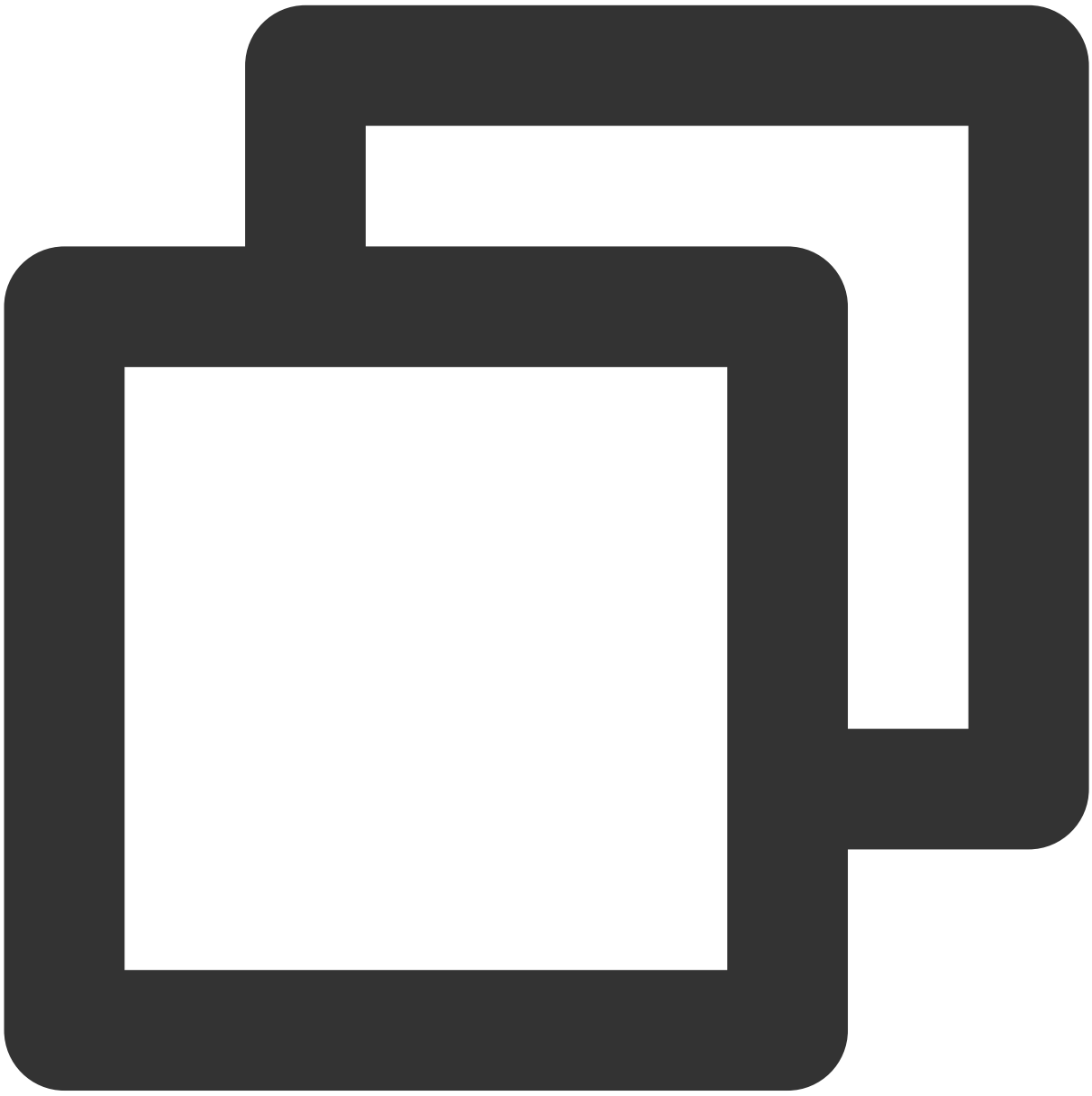
无

stop

说明

停止当前正在播放的视频。

接口



```
Future<bool> stop({bool isNeedClear = false}) async;
```

参数说明

参数名	类型	描述

isNeedClear	bool	是否清除最后一帧画面

返回值说明

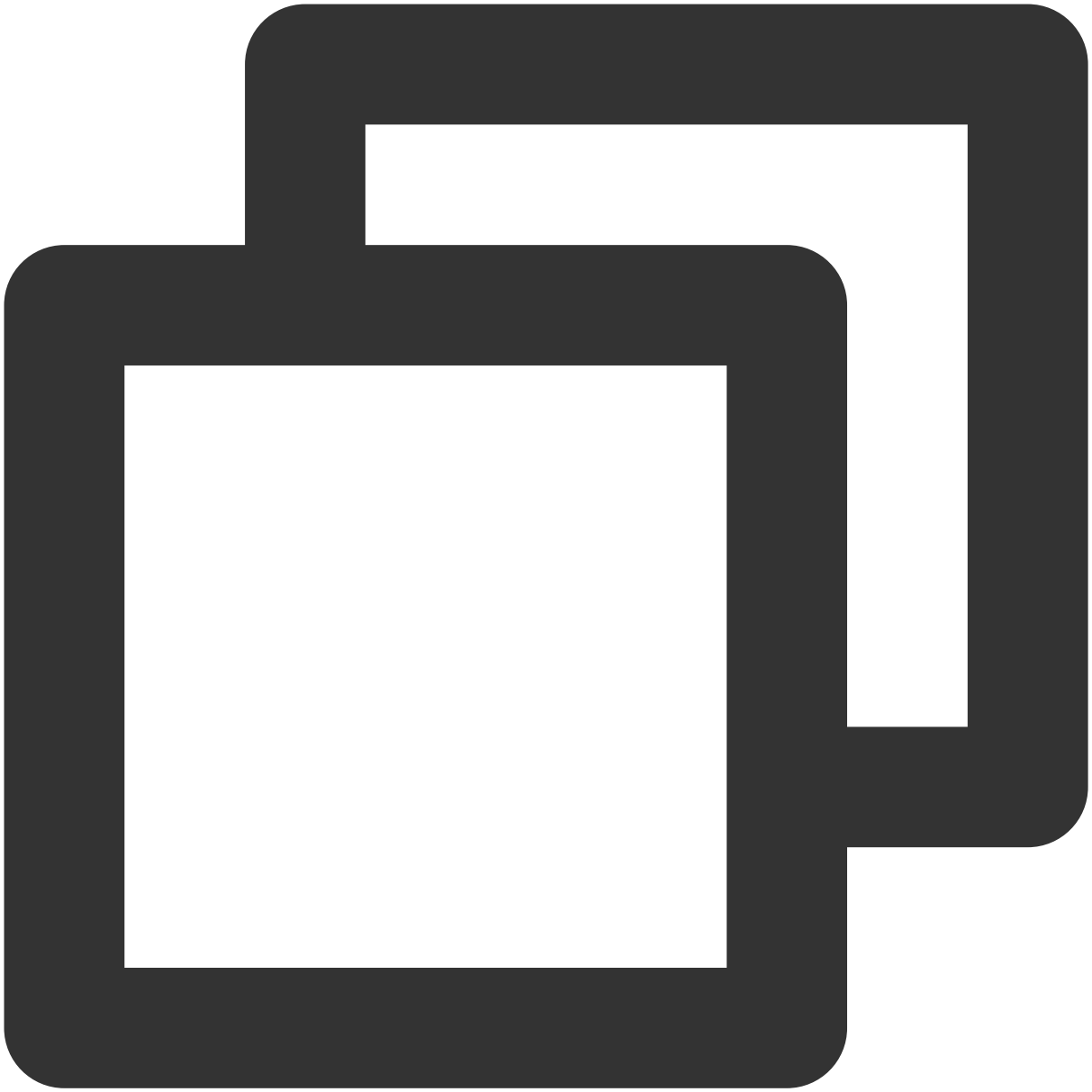
参数名	类型	描述
result	bool	停止是否成功

isPlaying

说明

当前播放器是否正在播放。

接口



```
Future<bool> isPlaying() async;
```

参数说明

无

返回值说明

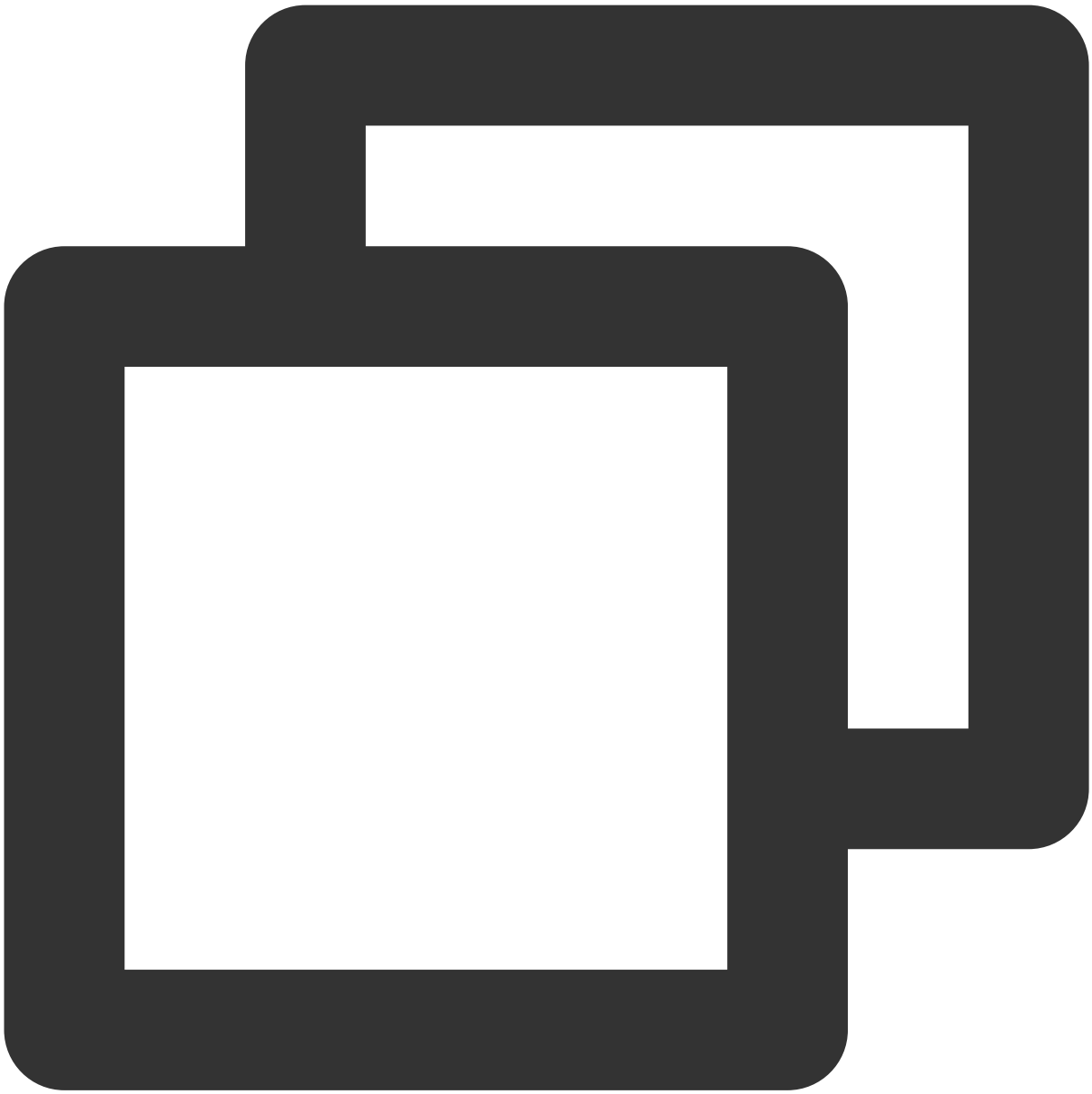
参数名	类型	描述
isPlaying	bool	是否正在播放

setMute

说明

设置当前播放是否静音。

接口



```
Future<void> setMute(bool mute) async;
```

参数说明

参数名	类型	描述

mute	bool	是否静音

返回值说明

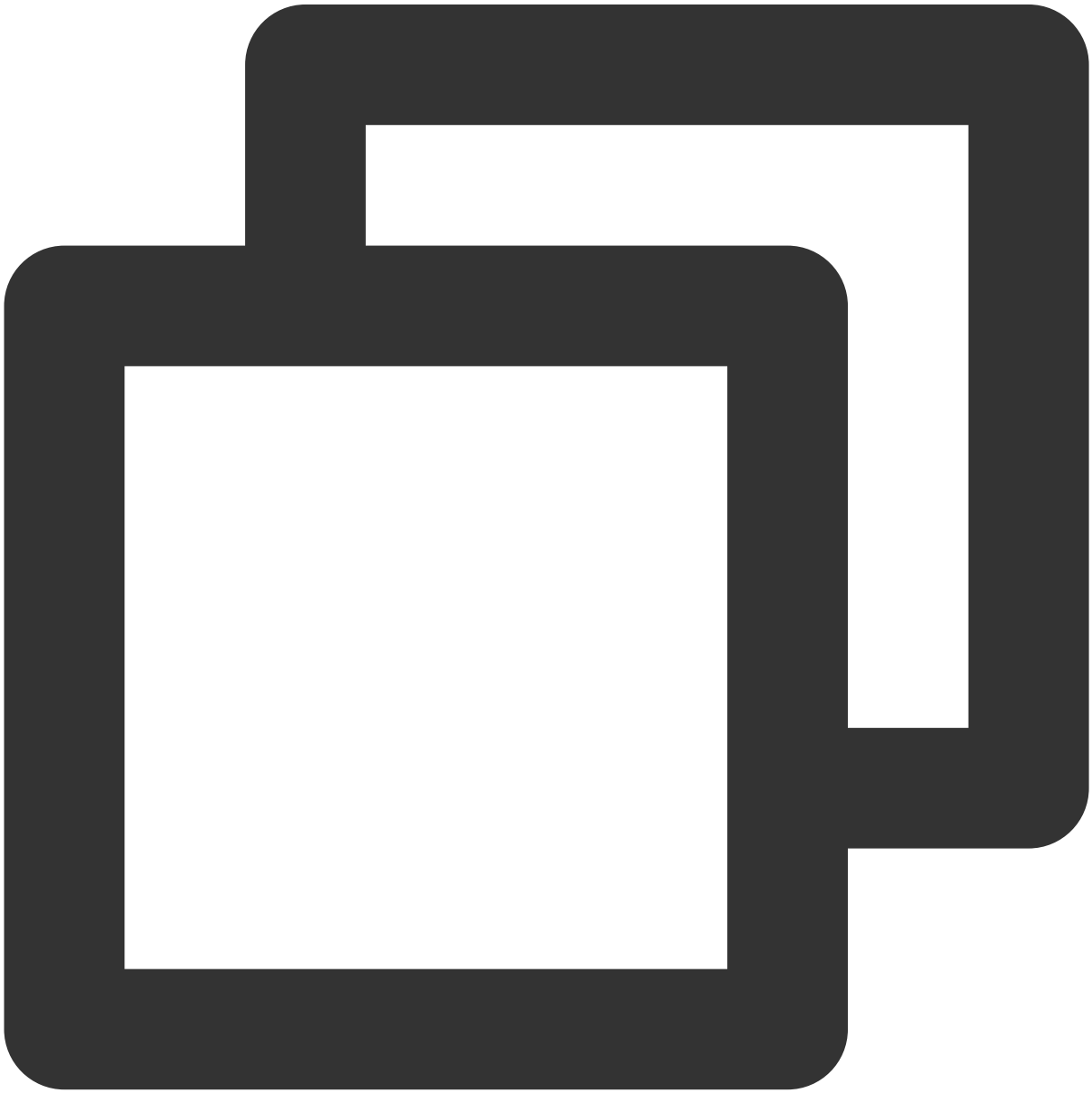
无

setVolume

说明

设置视频的声音大小。

接口



```
Future<void> setVolume(int volume);
```

参数说明

参数名	类型	描述
volume	int	视频声音大小，范围0~100

返回值说明

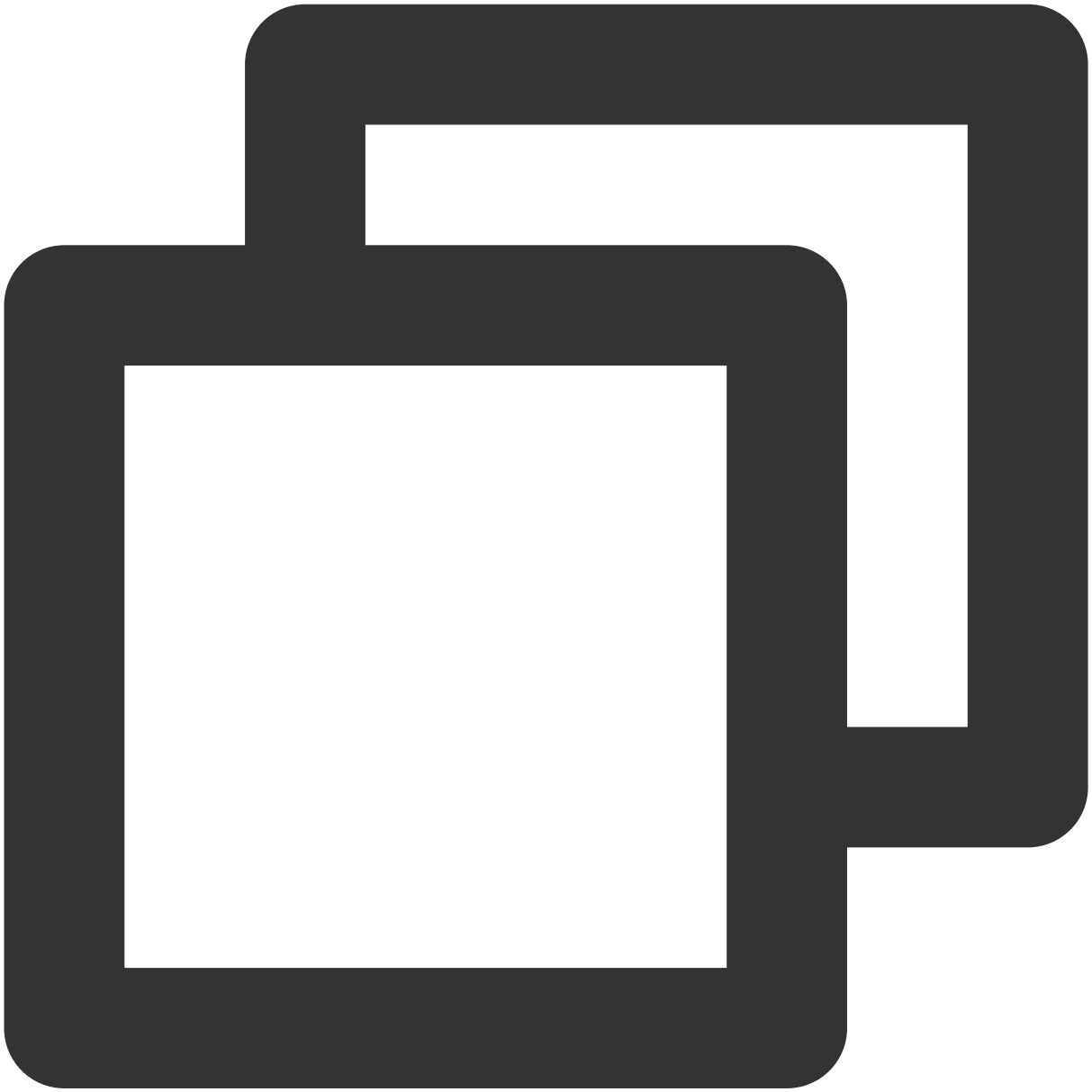
无

setLiveMode

说明

设置直播模式。

接口



```
Future<void> setLiveMode(TXPlayerLiveMode mode) async;
```

参数说明

参数名	类型	描述
mode	int	直播模式，自动模式、极速模式、流畅模式

返回值说明

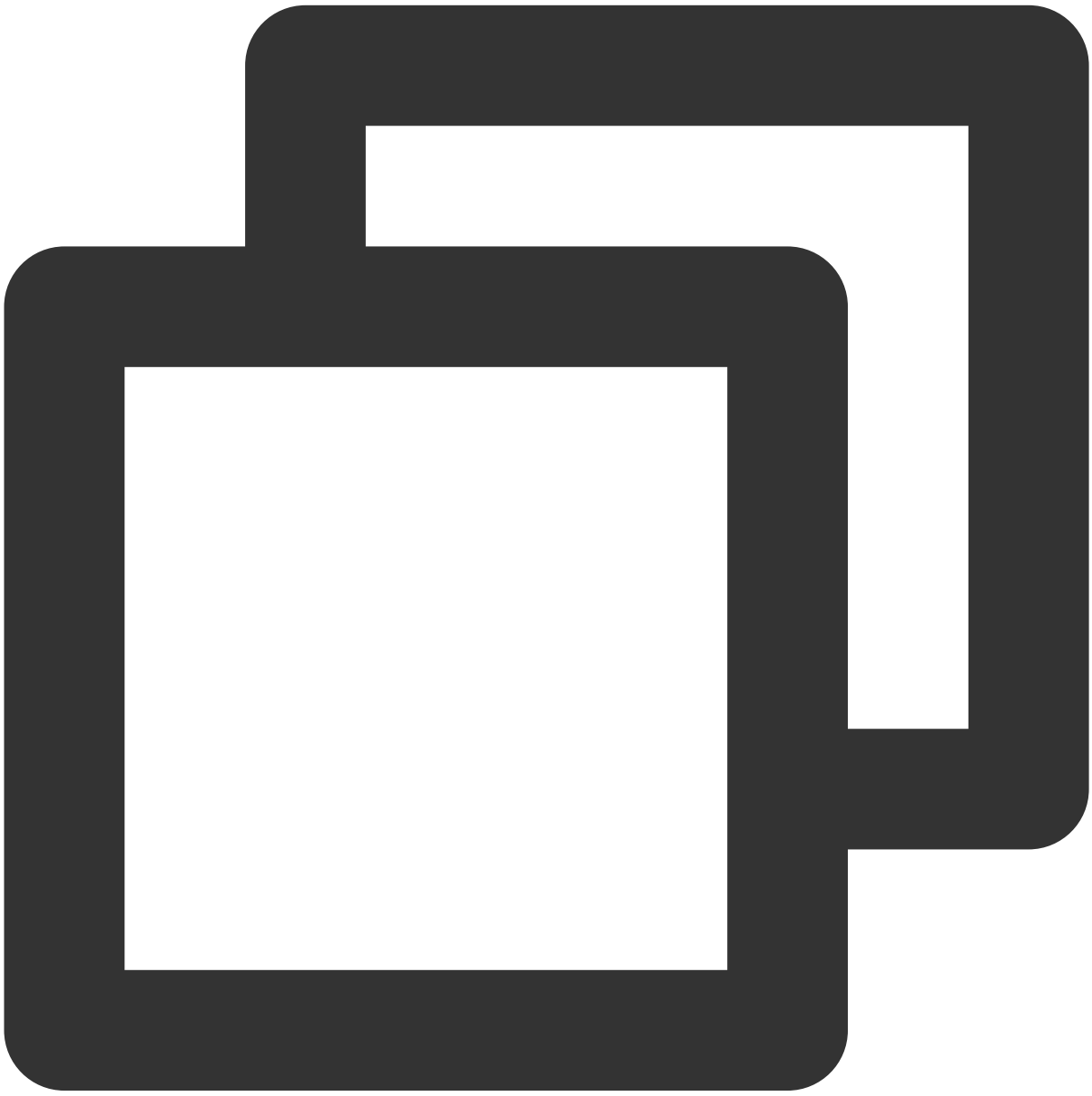
无

setAppID

说明

设置 appID，云控使用。

接口



```
Future<void> setAppID(int appId) async;
```

参数说明

参数名	类型	描述

appId	int	appId

返回值说明

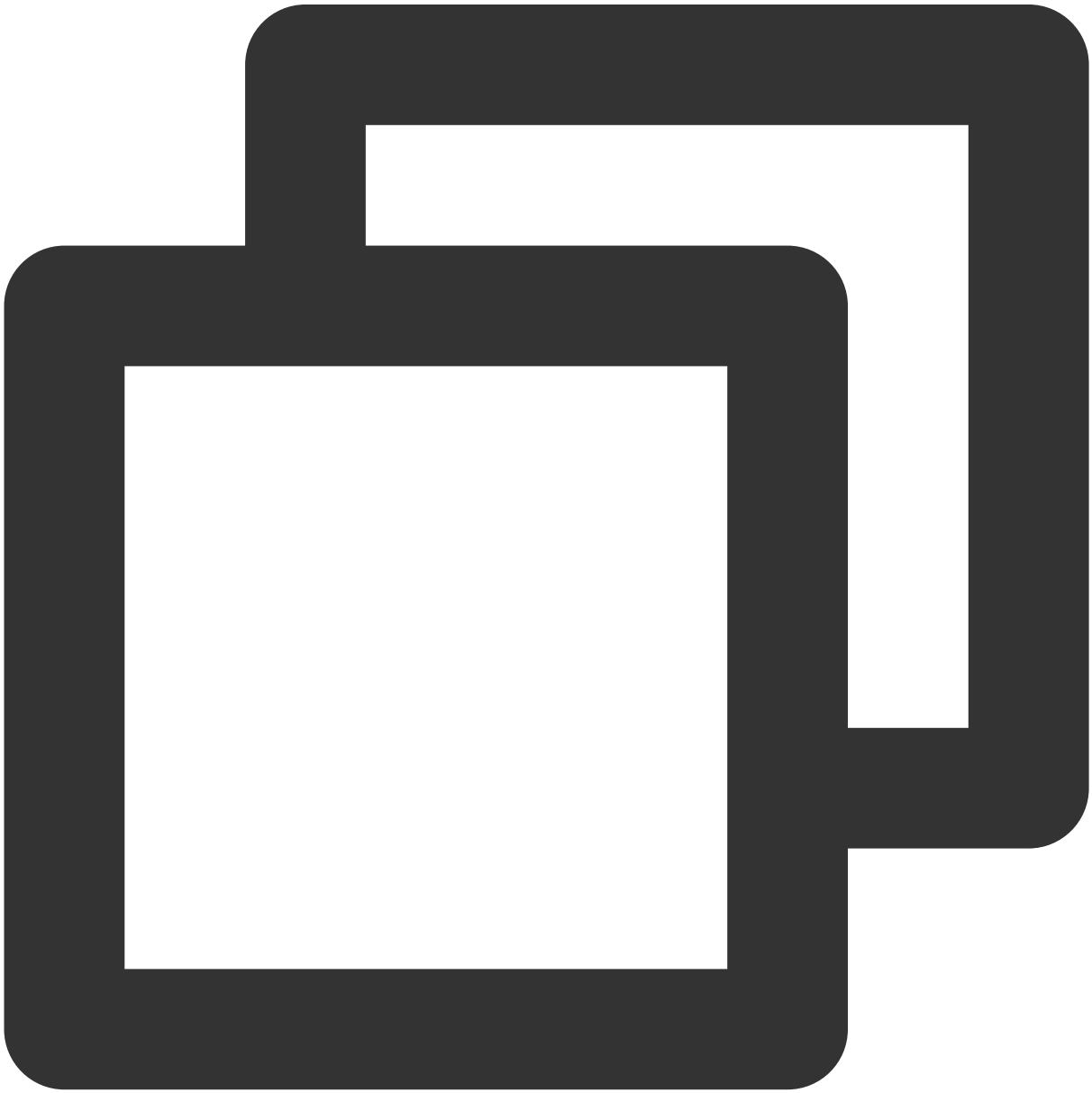
无

setConfig

说明

给播放器进行配置。

接口



```
Future<void> setConfig(FTXLivePlayConfig config) async;
```

参数说明

参数名	类型	描述
config	FTXLivePlayConfig	请参考 `FTXLivePlayConfig类`

返回值说明

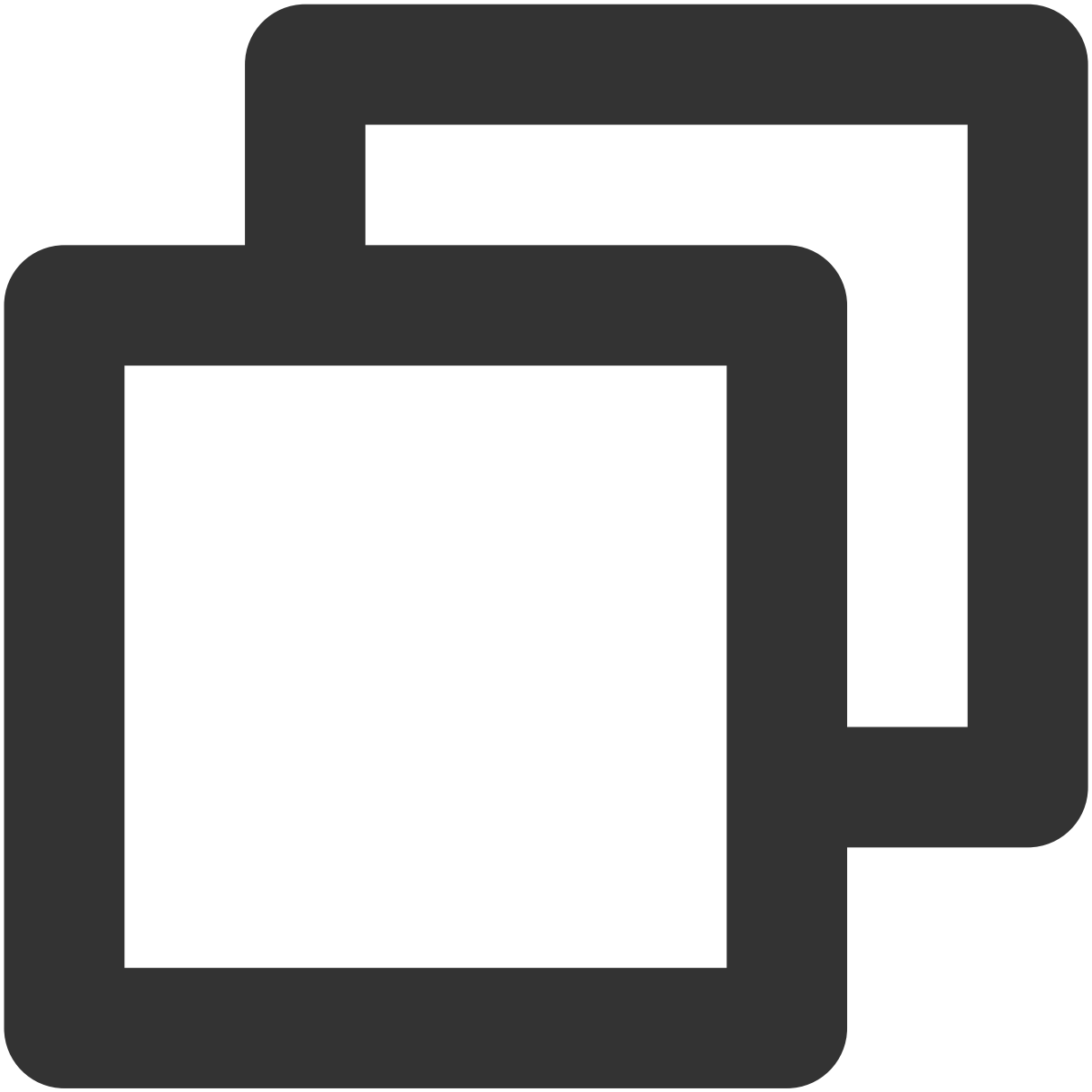
无

enableHardwareDecode

说明

开启/关闭硬解播放，设置后不会立即生效，需要重新播放才可生效。

接口



```
Future<bool> enableHardwareDecode(bool enable);
```

参数说明

参数名	类型	描述
enable	bool	是否开启硬解

返回值说明

参数名	类型	描述

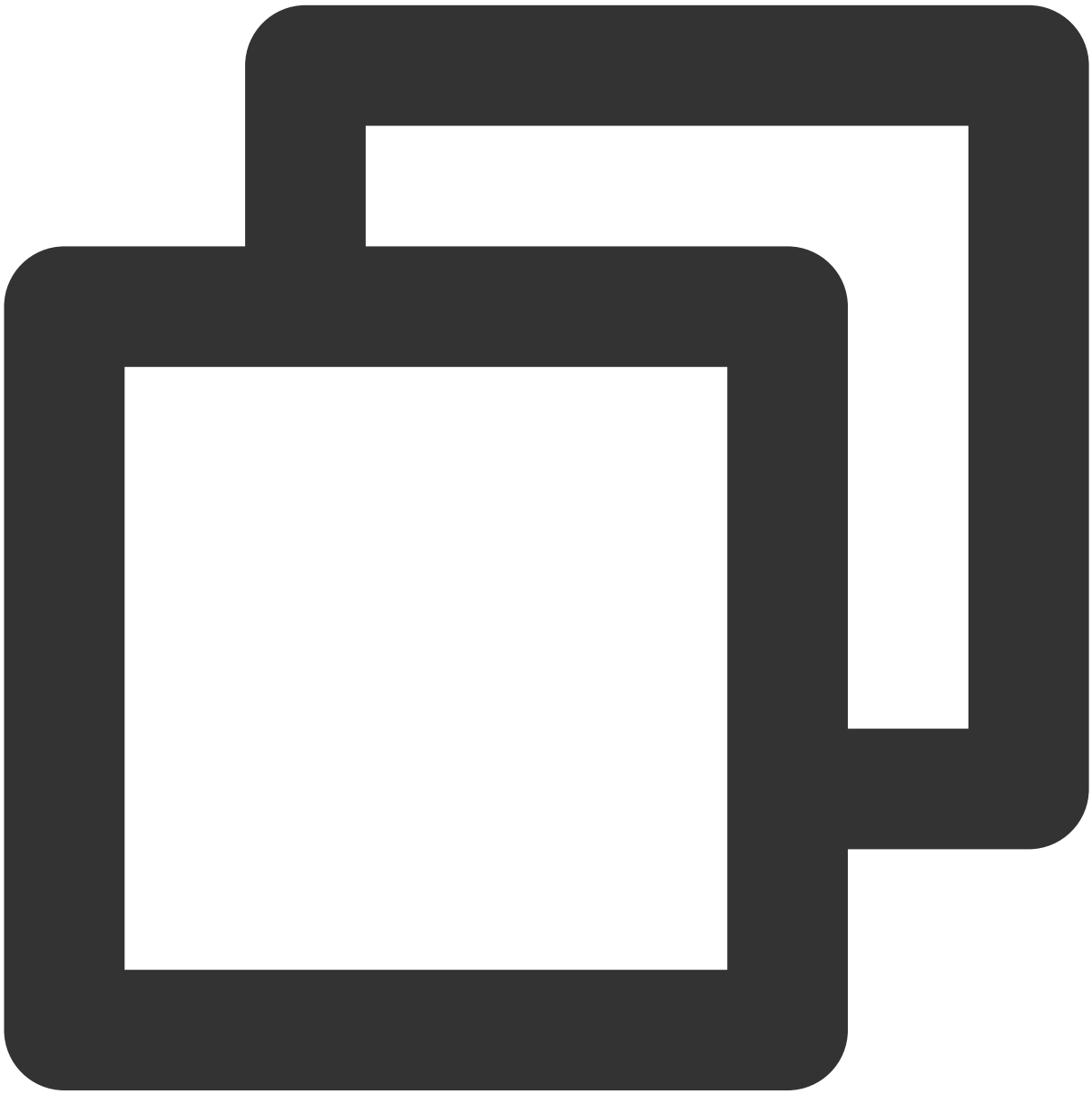
result	bool	硬解/软解设置结果

enterPictureInPictureMode

说明

进入画中画模式，仅支持 Android 端，iOS 端直播目前暂不支持画中画模式。

接口



```
Future<int> enterPictureInPictureMode({String? backIconForAndroid, String? playIcon
```

参数说明

该参数只适用于 android 平台。

参数名	类型	描述
backIcon	String	回退按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
playIcon	String	播放按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
pauseIcon	String	暂停按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标
forwardIcon	String	快进按钮图标，由于 android 平台限制，图标大小不得超过1M，可不传，不传则使用系统自带图标

返回值说明

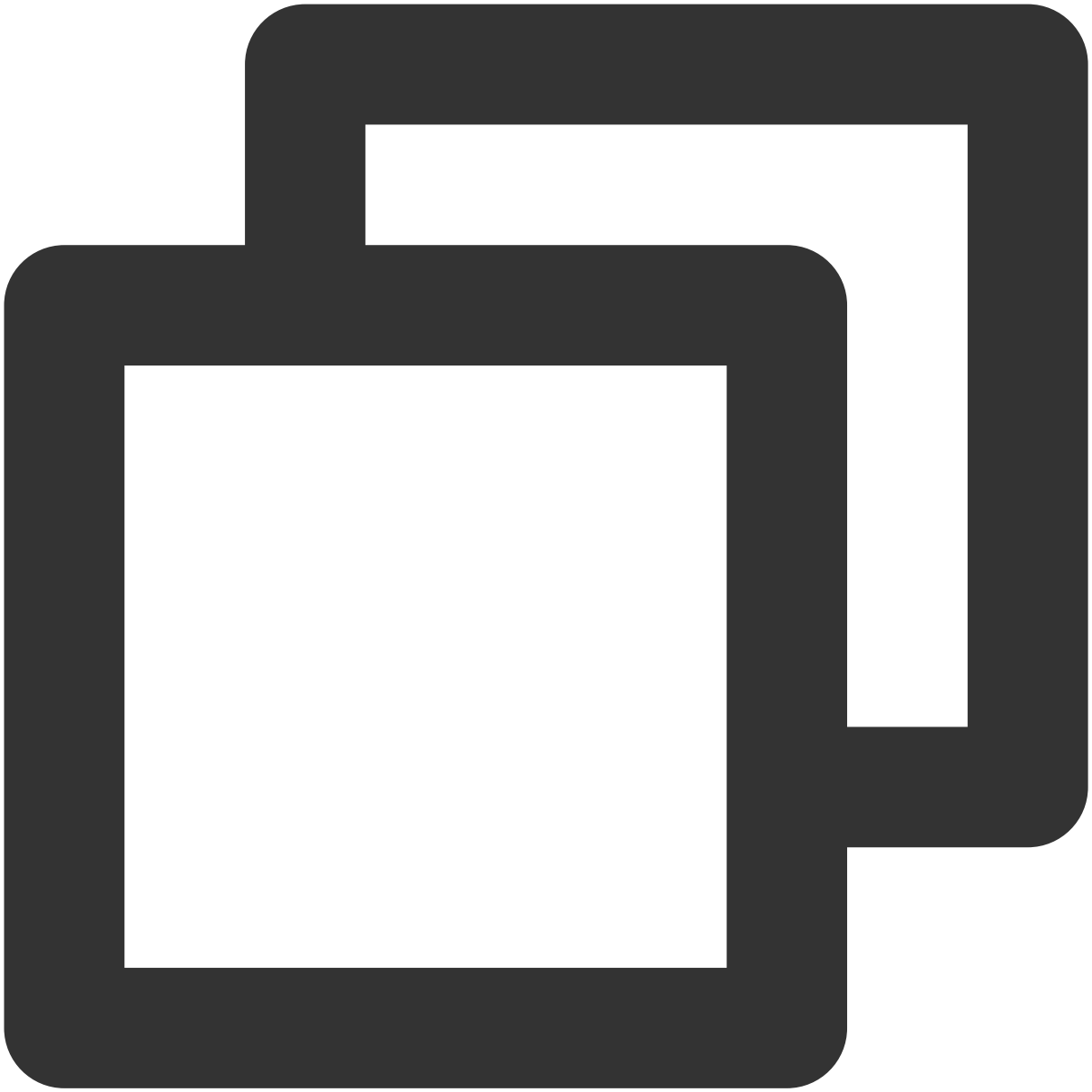
参数名	值	描述
NO_ERROR	0	启动成功，没有错误
ERROR_PIP_LOWER_VERSION	-101	android版本过低，不支持画中画模式
ERROR_PIP_DENIED_PERMISSION	-102	画中画模式权限未打开，或者当前设备不支持画中画
ERROR_PIP_ACTIVITY_DESTROYED	-103	当前界面已经销毁
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	设备或系统版本不支持（iPad iOS9+ 才支持 PIP），只适用于 iOS
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	播放器不支持，只适用于 iOS
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	视频不支持，只适用于 iOS
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	PIP控制器不可用，只适用于 iOS
ERROR_IOS_PIP_FROM_SYSTEM	-108	PIP控制器报错，只适用于 iOS
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	播放器对象不存在，只适用于 iOS
ERROR_IOS_PIP_IS_RUNNING	-110	PIP功能已经运行，只适用于 iOS
ERROR_IOS_PIP_NOT_RUNNING	-111	PIP功能没有启动，只适用于 iOS

dispose

说明

销毁 controller，调用该方法会销毁掉所有通知事件，释放掉播放器。

接口



```
Future<void> dispose() async;
```

参数说明

无

返回值说明

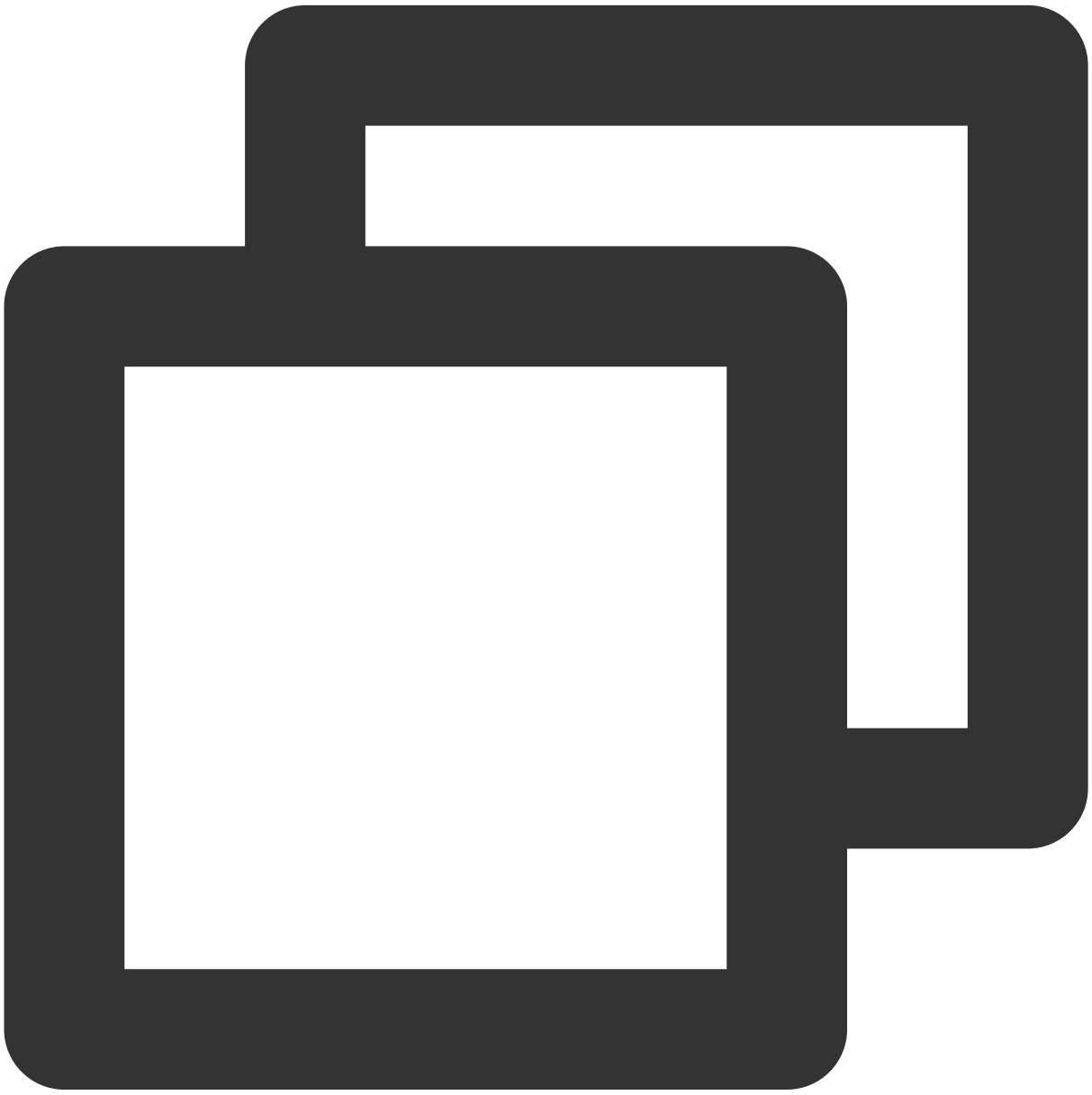
无

switchStream

说明

切换播放流

接口



```
Future<int> switchStream(String url) async;
```

参数说明

参数名	类型	描述

url	String	需要切换的视频源

返回值说明

参数名	类型	描述
result	int	切换结果

FTXLivePlayConfig类

属性配置说明

参数名	类型	描述
cacheTime	double	播放器缓存时间，单位秒，取值需要大于0，默认值：5
maxAutoAdjustCacheTime	double	播放器缓存自动调整的最大时间，单位秒，取值需要大于0，默认值：5
minAutoAdjustCacheTime	double	播放器缓存自动调整的最小时间，单位秒，取值需要大于0，默认值为1
videoBlockThreshold	int	播放器视频卡顿报警阈值，单位毫秒,只有渲染间隔超过这个阈值的卡顿才会有 PLAY_WARNING_VIDEO_PLAY_LAG 通知
connectRetryCount	int	播放器遭遇网络连接断开时 SDK 默认重试的次数，取值范围1 - 10，默认值：3。
connectRetryInterval	int	网络重连的时间间隔，单位秒，取值范围3 - 30，默认值：3。
autoAdjustCacheTime	bool	是否自动调整播放器缓存时间，默认值：true。true：启用自动调整，自动调整的最大值和最小值可以分别通过修改 maxCacheTime 和 minCacheTime 来设置。false：关闭自动调整，采用默认的指定缓存时间(1s)，可以通过修改 cacheTime 来调整缓存时间
enableAec	bool	是否开启回声消除， 默认值为 false
enableMessage	bool	是否开启消息通道， 默认值为 true
enableMetaData	bool	是否开启 MetaData 数据回调，默认值为 NO。 true：SDK 通过 EVT_PLAY_GET_METADATA 消息抛出视频流的 MetaData 数据；false：SDK 不抛出视频流的 MetaData 数据。
flvSessionKey	String	是否开启 HTTP 头信息回调，默认值为 “”

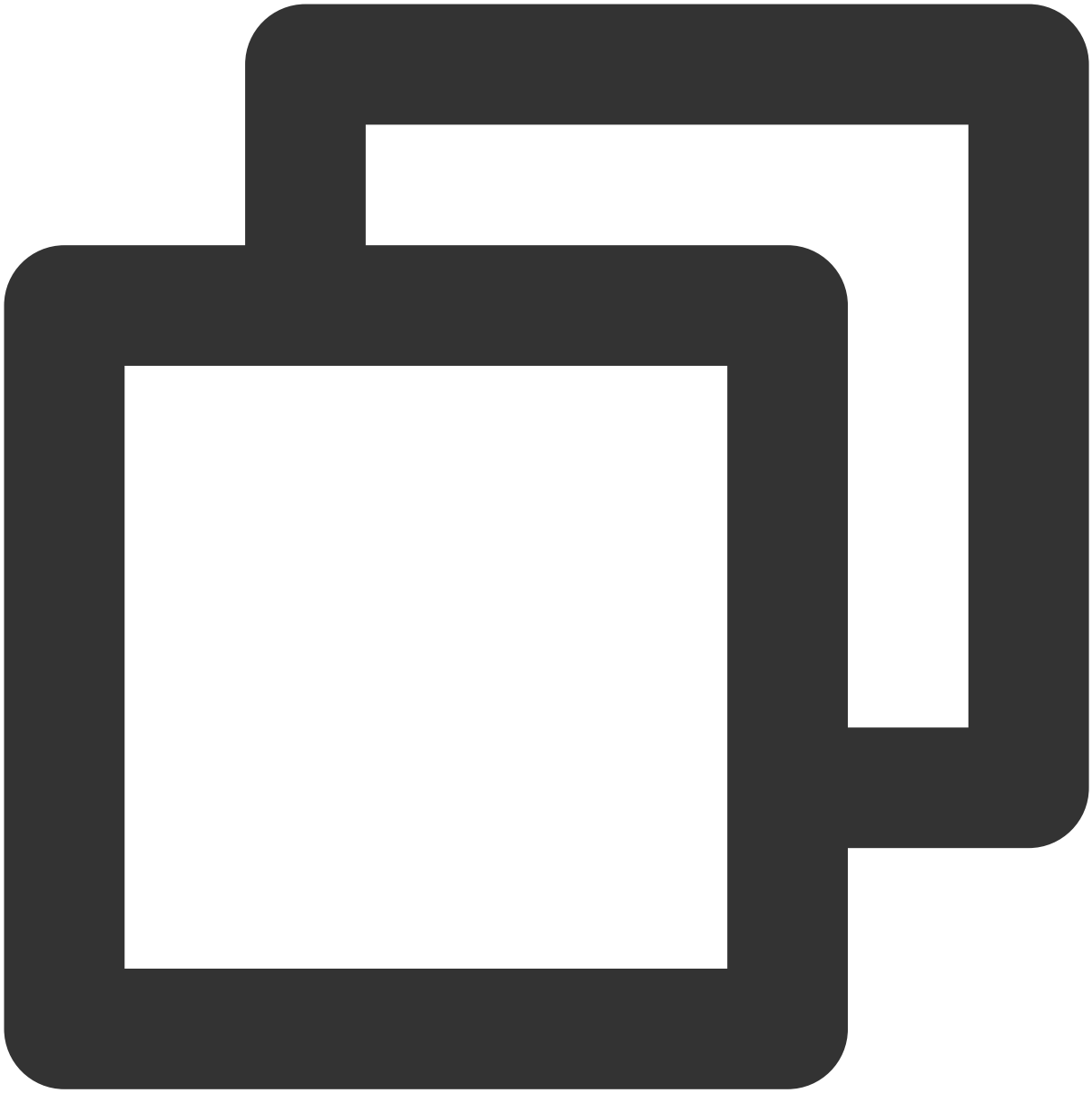
TXVodDownloadController类

startPreLoad

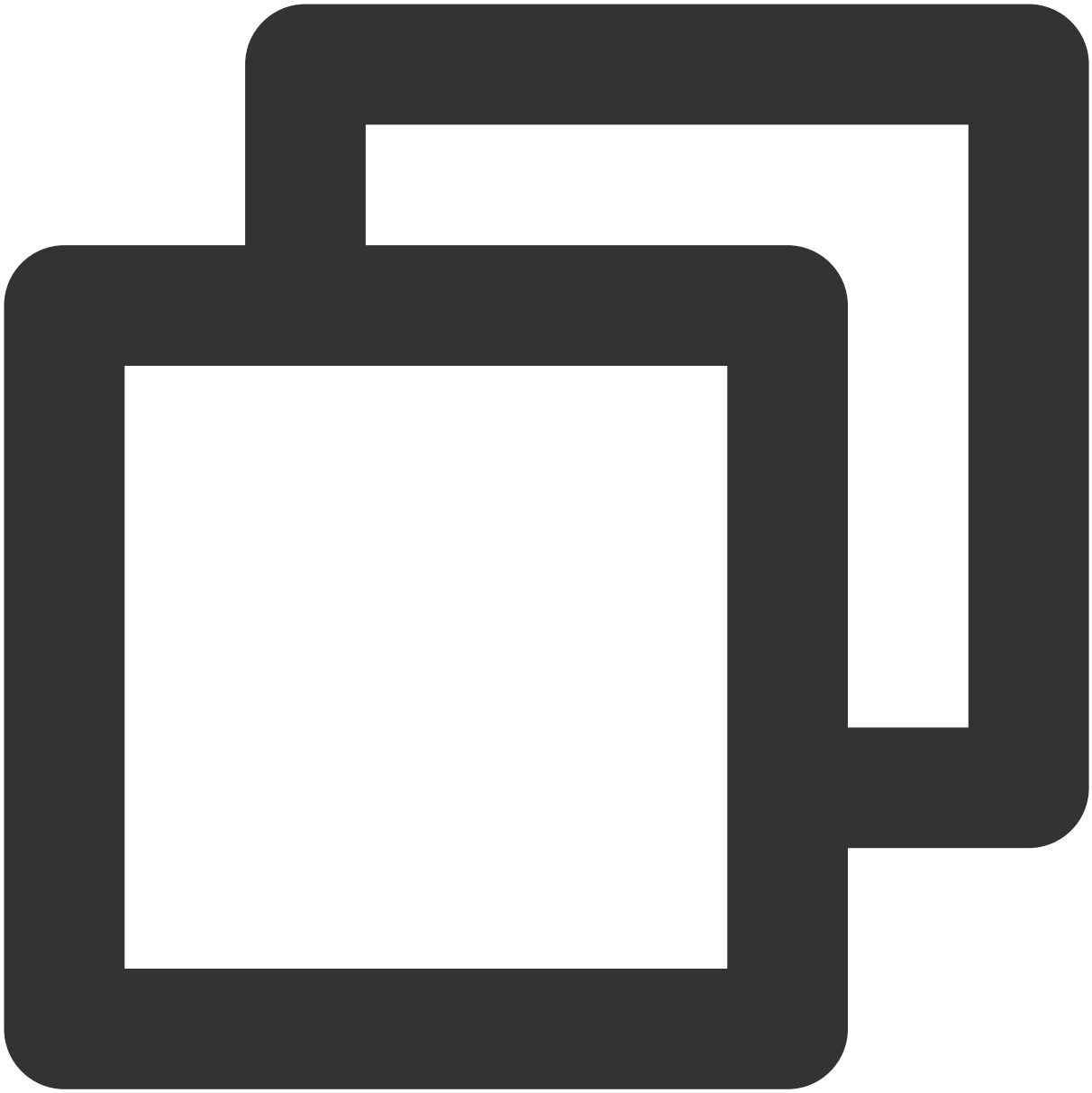
说明

启动预下载。启动预下载前，请先设置好播放引擎的缓存目录 `[SuperPlayerPlugin.setGlobalCacheFolderPath]` 和缓存大小 `[SuperPlayerPlugin.setGlobalMaxCacheSize]`，这个设置是全局配置需和播放器保持一致，否则会造成播放缓存失效。

接口



```
Future<int> startPreLoad(
final String playUrl,
final int preloadSizeMB,
final int preferredResolution, {
FTXPredownlodOnCompleteListener? onCompleteListener,
    FTXPredownlodOnErrorListener? onErrorListener,
}) async
```



```
Future<void> startPreload(TXPlayInfoParams txPlayInfoParams,
    final int preloadSizeMB,
    final int preferredResolution, {
      FTXPredownlodOnCompleteListener? onCompleteListener,
      FTXPredownlodOnErrorListener? onErrorListener,
      FTXPredownlodOnStartListener? onStartListener,
    }) async
```

参数说明

参数名	类型	描述

playUrl	String	要预下载的 url
preloadSizeMB	int	预下载的大小（单位：MB）
preferredResolution	int	期望分辨率，值为高x宽。可参考如720*1080。不支持多分辨率或不需指定时，传-1
onCompleteListener	FTXPredownlodOnCompleteListener?	预下载成功回调，全局
onErrorListener	FTXPredownlodOnErrorListener	预下载失败回调，全局

TXPlayInfoParams:

参数名	类型	描述
appId	int	应用appId。必填
fileId	String	文件id。必填
url	String	视频url，与fileId只用填写一个，如果都填写，url优先
sign	String	防盗链签名，参考防盗链产品文档

返回值说明

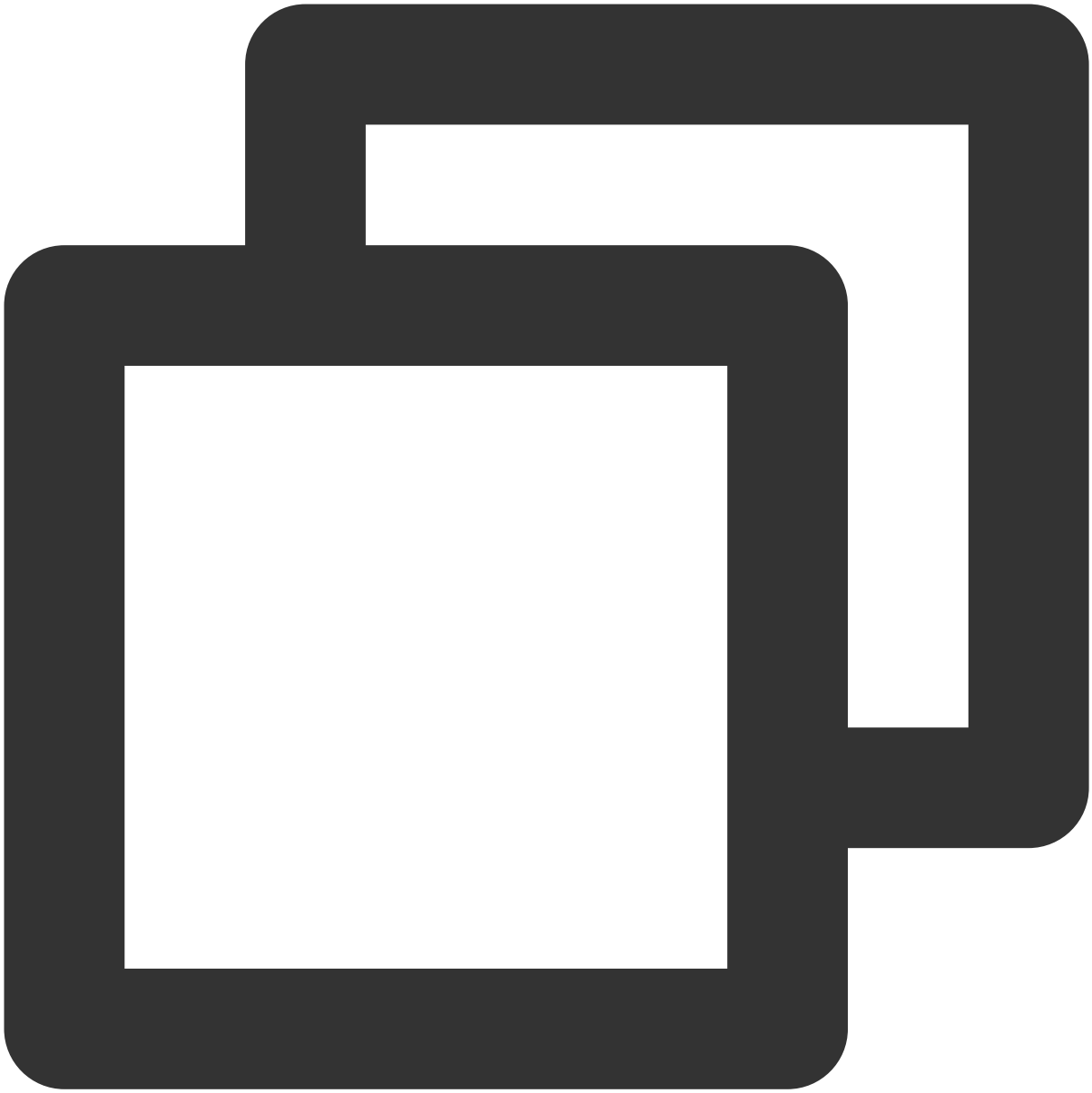
参数名	类型	描述
taskId	int	任务 ID

stopPreLoad

说明

停止预下载。

接口



```
Future<void> stopPreLoad(final int taskId) async
```

参数说明

参数名	类型	描述
taskId	int	任务 ID

返回值说明

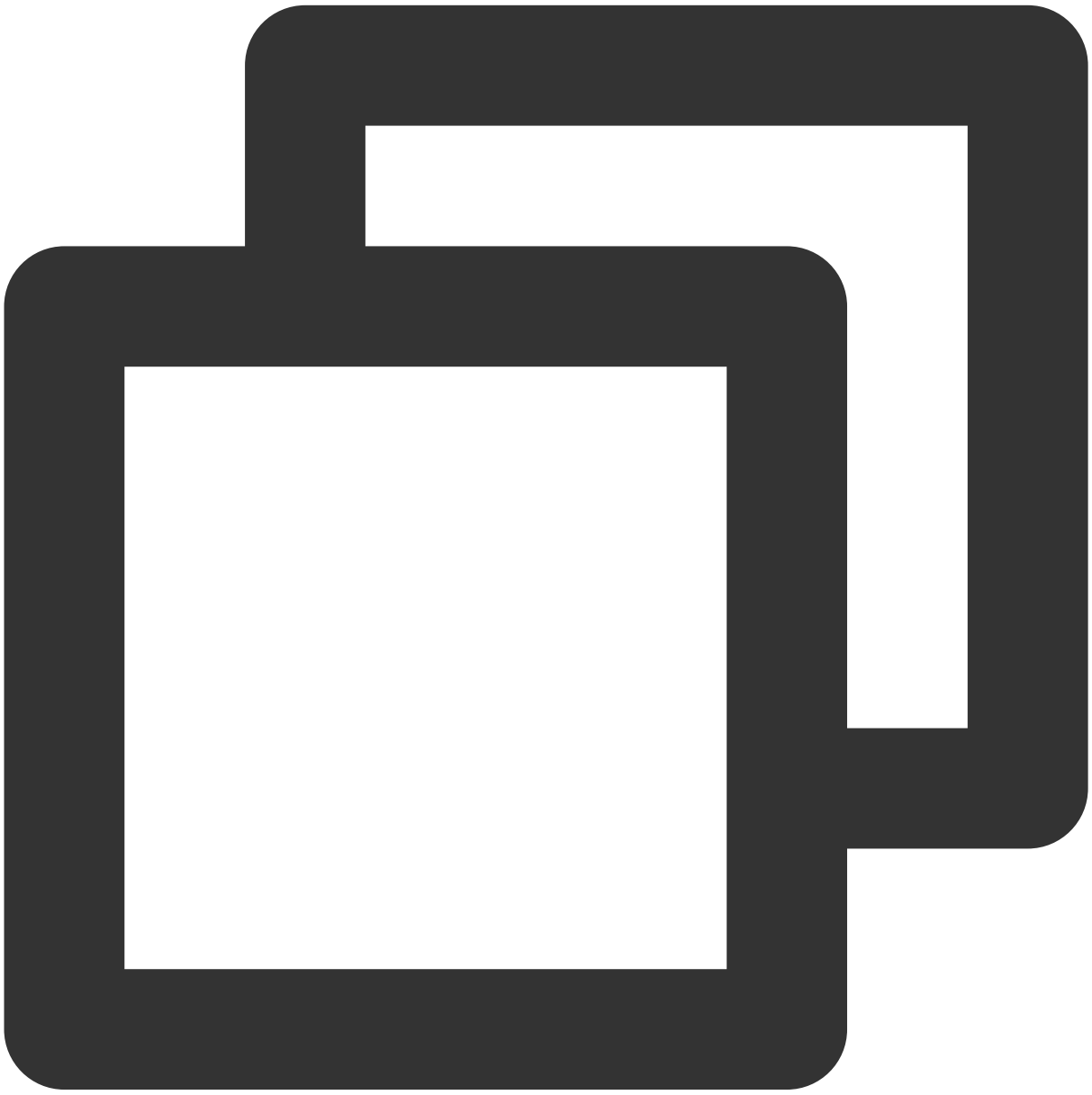
无

startDownload

说明

开始下载视频。

接口



```
Future<void> startDownload(TXVodDownloadMediaInfo mediaInfo) async
```

参数说明

参数名	类型	描述

mediaInfo	TXVodDownloadMediaInfo	下载任务信息

TXVodDownloadMediaInfo

参数名	类型	描述
playPath	String?	缓存地址，获得到的视频缓存会有该值，启动下载可以不赋值
progress	double?	缓存进度，获得到的视频缓存会有该值，启动下载可以不赋值
downloadState	int?	缓存状态，获得到的视频缓存会有该值，启动下载可以不赋值
userName	String?	下载账户名称，用于区分不同账户的下载，传空则为 default
duration	int?	缓存视频总时长，Android 端单位为毫秒，iOS 为秒，获得到的视频缓存会有该值，启动下载可以不赋值
playableDuration	int?	视频已缓存时长，Android 端单位为毫秒，iOS 为秒，获得到的视频缓存会有该值，启动下载可以不赋值
size	int?	文件总大小，单位：byte，获得到的视频缓存会有该值，启动下载可以不赋值
downloadSize	int?	文件已下载的大小，单位：byte，获得到的视频缓存会有该值，启动下载可以不赋值
url	String?	需要下载的视频 url，url 下载必填,不支持嵌套 m3u8 和 mp4 下载
dataSource	TXVodDownloadDataSource?	需要下载的视频 fileId 信息，url 与该参数可只使用一个
speed	int?	下载速度，单位：KByte/秒
isResourceBroken	bool?	资源是否已损坏, 如：资源被删除了

TXVodDownloadDataSource

参数名	类型	描述
appId	int?	下载文件对应的 appId，必填
fileId	String?	下载文件 Id，必填

pSign	String?	加密签名，加密视频必填
quality	int?	清晰度 ID，必传
token	String?	加密 token
userName	String?	下载账户名称，用于区分不同账户的下载，传空则为 default

返回值说明

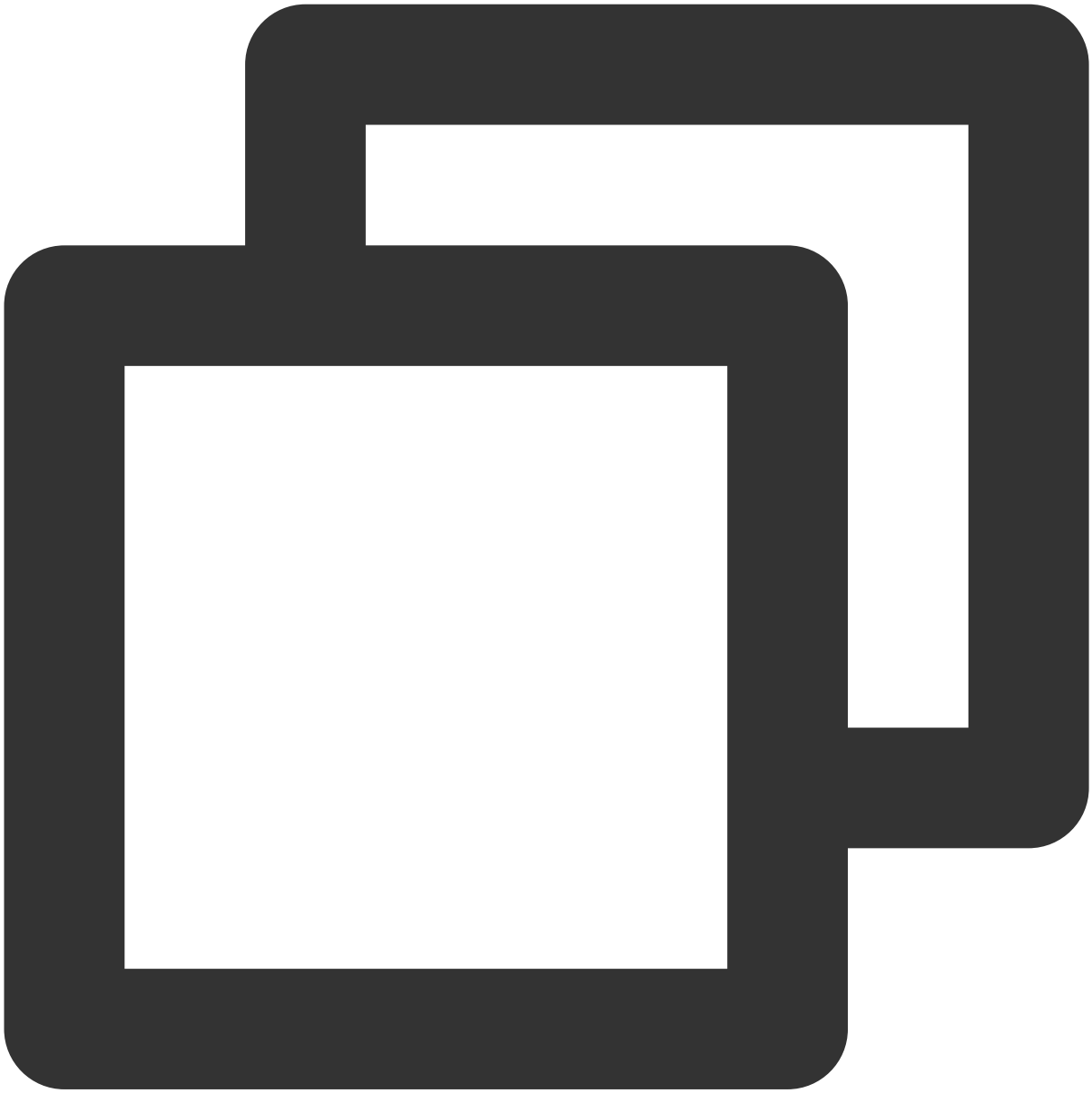
无

stopDownload

说明

停止下载。

接口



```
Future<void> stopDownload(TXVodDownloadMediaInfo mediaInfo) async
```

参数说明

参数名	类型	描述
mediaInfo	TXVodDownloadMediaInfo	任务信息

返回值说明

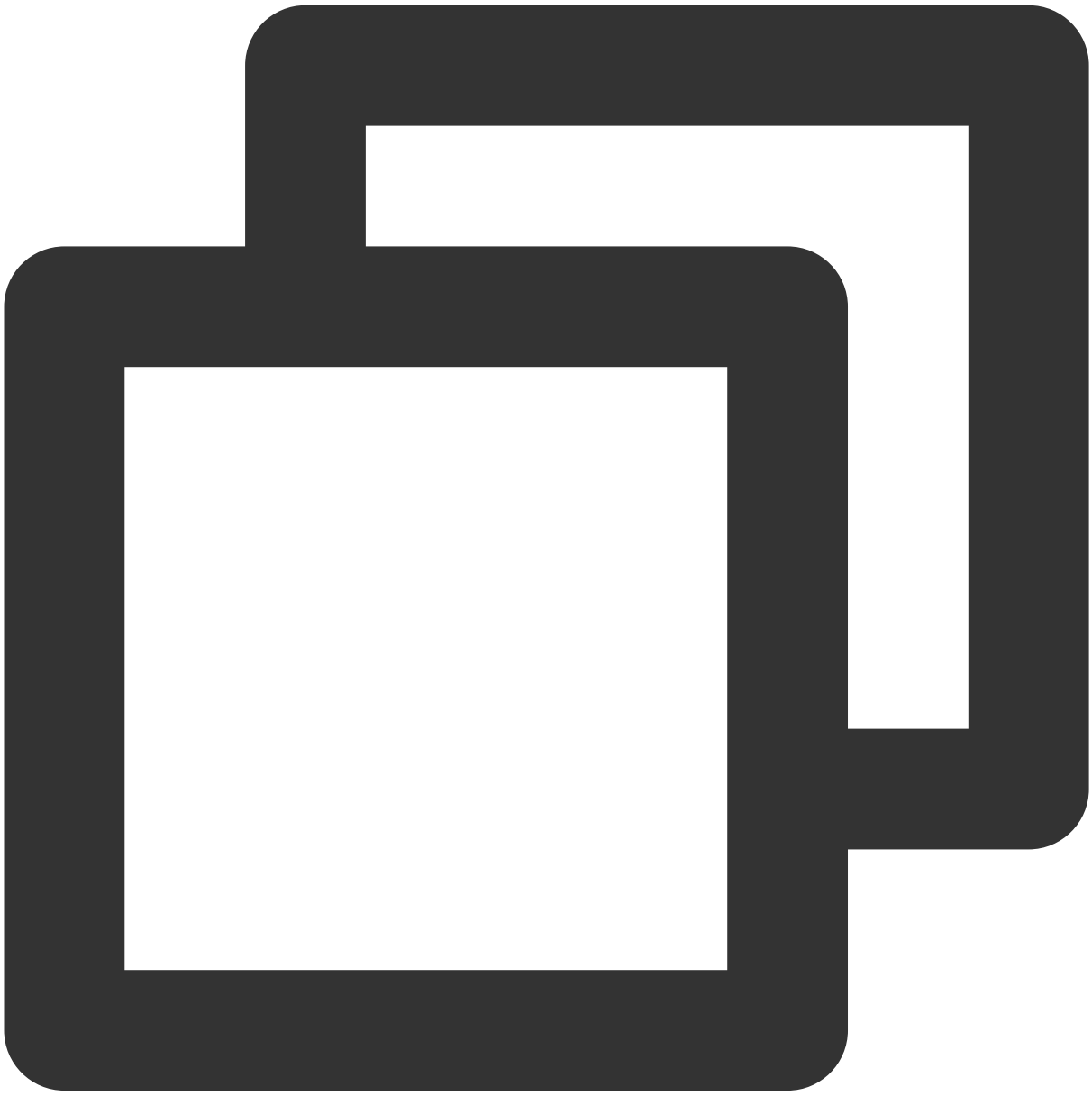
无

setDownloadHeaders

说明

设置下载任务请求头。

接口



```
Future<void> setDownloadHeaders(Map<String, String> headers) async
```

参数说明

参数名	类型	描述

headers	Map<String, String>	请求头信息

返回值说明

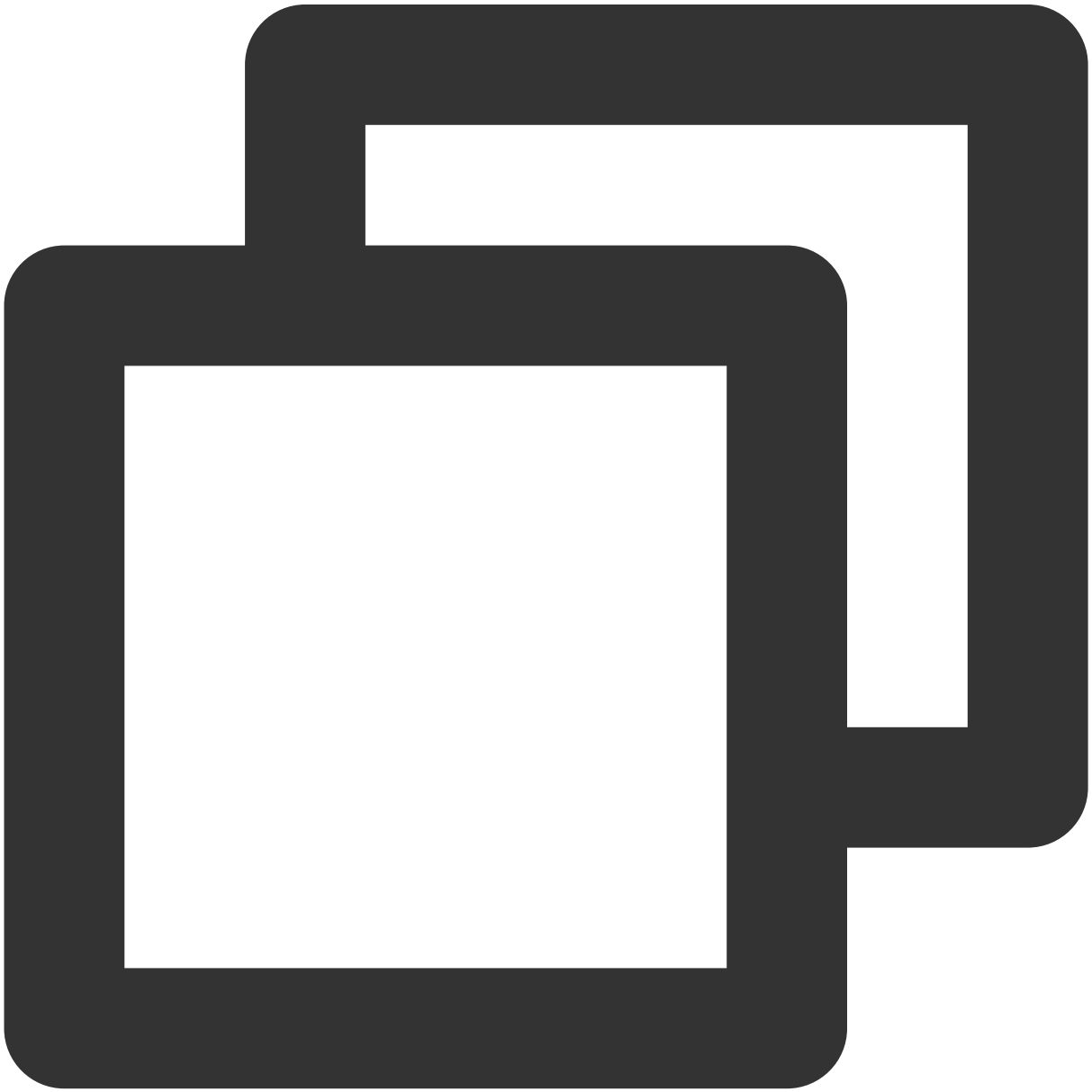
无

getDownloadList

说明

获得所有下载任务，包括已下载、正在下载以及下载错误的任务。

接口



```
Future<List<TXVodDownloadMediaInfo>> getDownloadList() async
```

参数说明

无

返回值说明

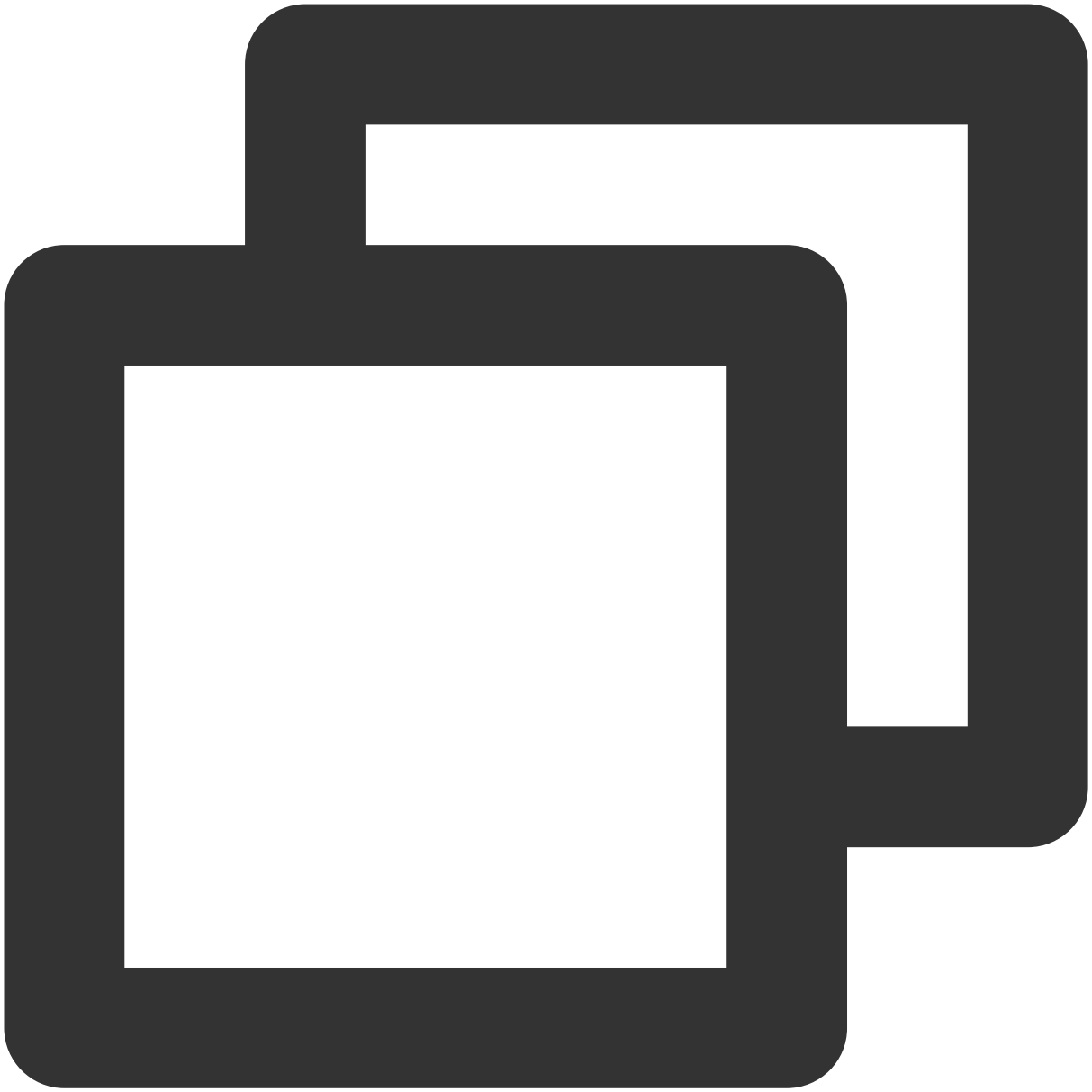
参数名	类型	描述
mediaInfoList	List<TXVodDownloadMediaInfo>	任务列表，可通过对比 userName 来区分不同用户的下载

getDownloadInfo

说明

获得下载任务信息。

接口



```
Future<TXVodDownloadMediaInfo> getDownloadInfo(TXVodDownloadMediaInfo mediaInfo) as
```

参数说明

参数名	类型	描述
mediaInfo	TXVodDownloadMediaInfo	任务信息

返回值说明

参数名	类型	描述

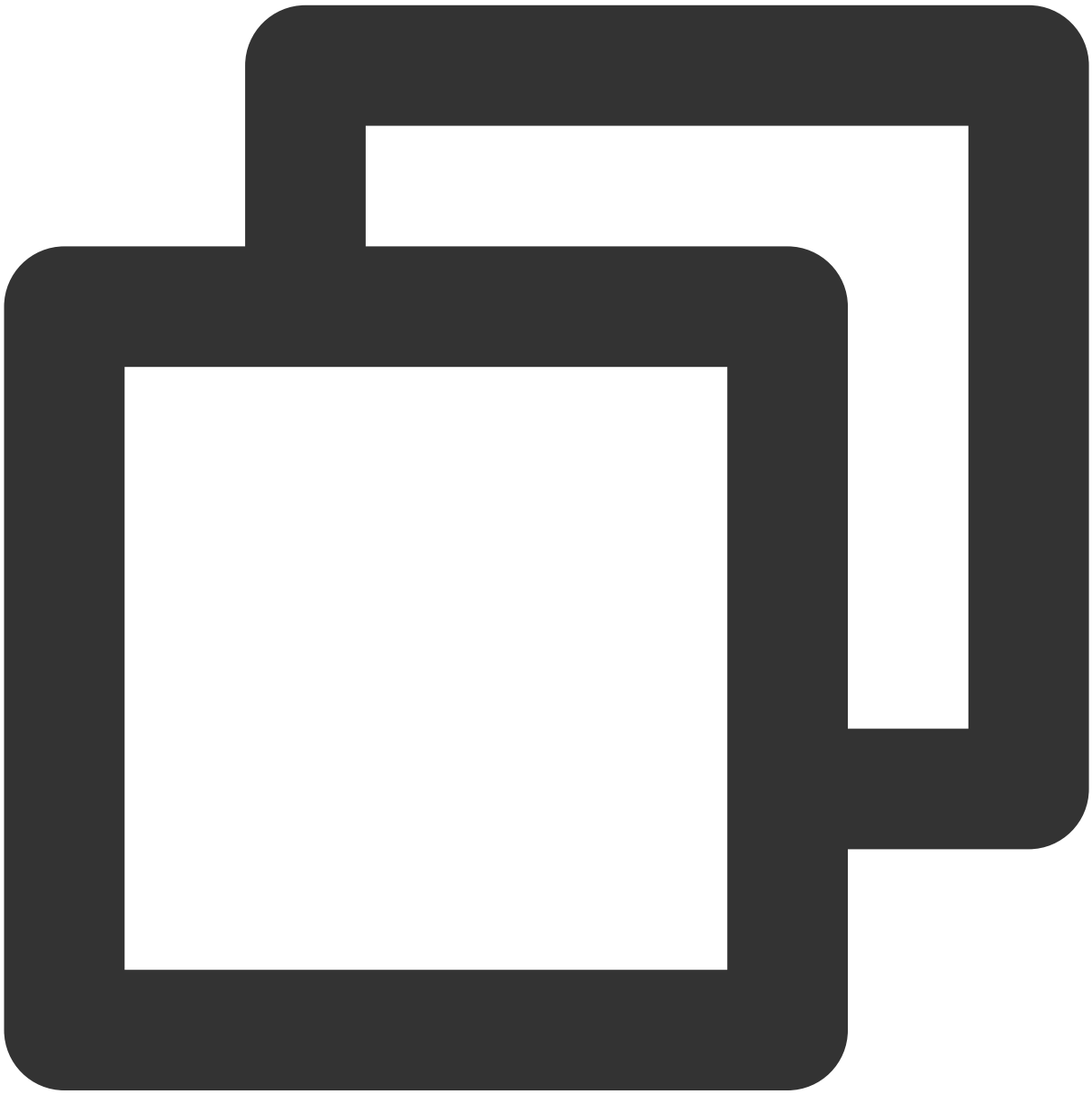
mediaInfo	TXVodDownloadMediaInfo	缓存任务详情信息

setDownloadObserver

说明

获得下载任务信息。

接口



```
void setDownloadObserver(FTXDownlodOnStateChangeListener downlodOnStateChangeListen
```

参数说明

参数名	类型	描述
downlodOnStateChangeListener	FTXDownlodOnStateChangeListener	任务下载状态回调
downlodOnErrorListener	FTXDownlodOnErrorListener	任务下载错误回调

返回值说明

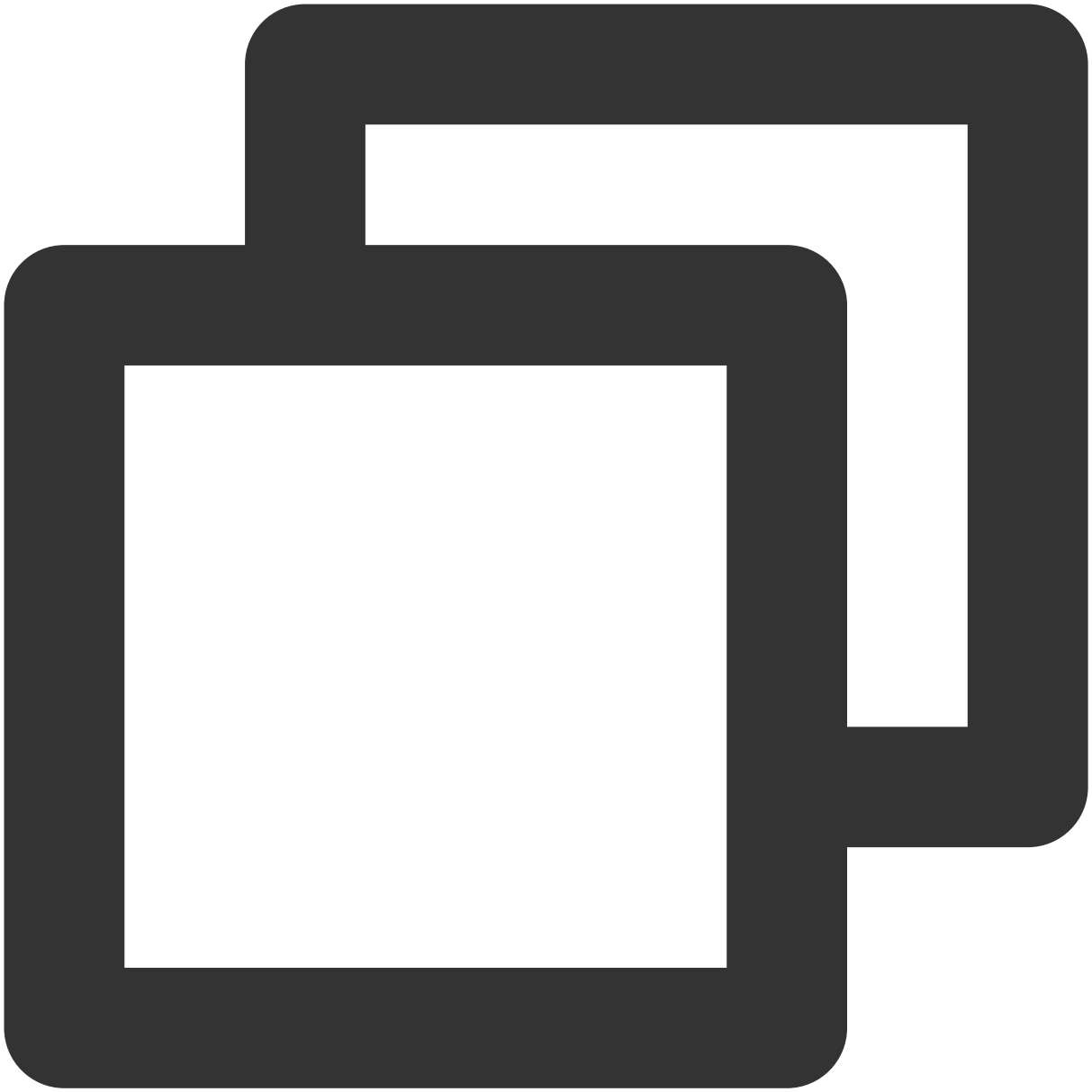
无

deleteDownloadMediaInfo

说明

删除下载的视频。

接口



```
Future<bool> deleteDownloadMediaInfo(TXVodDownloadMediaInfo mediaInfo) async
```

参数说明

参数名	类型	描述
mediaInfo	TXVodDownloadMediaInfo	任务下载信息

返回值说明

参数名	类型	描述

result	bool	删除结果

第三方播放器插件
第三方播放器 iOS 插件

最近更新时间：2022-10-17 11:16:01

第三方播放器 iOS 插件为云点播提供给客户希望使用第三方播放器或自研播放器开发的对接云 PaaS 资源的播放器插件，常用于有自定义播放器功能需求的用户。

SDK下载

第三方播放器 iOS 插件和 Demo 项目，请参见 TXCPlayerAdapterSDK_iOS。

集成指引

环境要求

配置支持 HTTP 请求，需要在项目的 info.plist 文件中添加 `App Transport Security Settings->Allow Arbitrary Loads` 设置为 YES。

组件依赖

添加 `GCDWebServer` 组件依赖。

```
pod "GCDWebServer", "~> 3.0"
```

GCDWebServer 是一个轻量的 HTTP server，它基于 GCD 并可用于 OS X & iOS，该库还实现了基于 Web 的文件上传以及 WebDAV server 等扩展功能。

使用播放器

变量声明，播放器主类为 `TXCPlayerAdapter` ，创建后即可播放视频。

fileId 一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 fileId 到客户端。
2. 服务端视频上传，在 确认上传 的通知中包含对应的 fileId。

如果文件已存在腾讯云，则可以进入 媒资管理 ，找到对应的文件。点开后在右侧视频详情中，可以看到相关参数。

```
NSInteger appId; ////appid 在腾讯云点播申请
NSString *fileId;
//psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
NSString *pSign = self.pSignTextView.text;

TXCPlayerAdapter *adapter = [TXCPlayerAdapter shareAdapterWithAppId:appId];
```

请求视频信息和播放：

```
id<ITXCPlayerAssistorProtocol> assistor = [TXCPlayerAdapter createPlayerAssistorWithFileId:fileId pSign:pSign];
[assistor requestVideoInfo:^(id<ITXCPlayerAssistorProtocol> response, NSError *error) {
if (error) {
NSLog(@"create player assistor error : %@",error);
[self.view makeToast:error.description duration:5.0 position:CSToastPositionBottom];
return;
}
[weakSelf avplayerPlay:response]; //播放视频
}];

- (void)avplayerPlay:(id<ITXCPlayerAssistorProtocol>)response
{
AVPlayerViewController *playerVC = [[AVPlayerViewController alloc] init];
self.playerVC = playerVC;
TXCStreamingInfo *info = response.getStreamingInfo;
AVPlayer *player = [[AVPlayer alloc] initWithURL:[NSURL URLWithString:info.playUrl]];
playerVC.player = player;
playerVC.title = response.getVideoBasicInfo.name;
[self.navigationController pushViewController:playerVC animated:YES];

[player addObserver:self forKeyPath:@"status" options:NSKeyValueObservingOptionNew context:nil];
}
```

使用完后销毁 Player：

```
[TXCPlayerAdapter destroy];
```

SDK 接口说明

初始化 Adatper

初始化 Adapter，单例。

接口

```
+ (instancetype)shareAdapterWithAppId:(NSUInteger)appId;
```

参数说明

appId：填写 appid（如果使用了子应用，则填 subappid）。

销毁 Adatper

销毁 Adapter，当程序退出后调用。

接口

```
+ (void)destroy;
```

创建播放器辅助类

通过播放器辅助类可以获取播放 fileId 相关信息以及处理 DRM 加密接口等。

接口

```
+ (id<ITXCPlayerAssistorProtocol>)createPlayerAssistorWithFileId:(NSString *)file
Id
pSign:(NSString *)pSign;
```

参数说明

参数名	类型	描述
fileId	String	要播放的视频 fileId
pSign	String	播放器签名

请求视频播放信息

本接口会请求腾讯云点播服务器，获取播放视频的流信息等。

接口

```
- (void)requestVideoInfo:(ITXCRequestVideoInfoCallback)completion;
```

参数说明

参数名	类型	描述
completion	ITXCRequestVideoInfoCallback	异步回调函数

销毁播放器辅助类

销毁辅助类，在退出播放器或者切换了下一个视频播放的时候调用。

接口

```
+ (void)destroyPlayerAssistor:(id<ITXCPlayerAssistorProtocol>)assistor;
```

获取视频的基本信息

获取视频信息， 必须是在 `id<itxcplayerassistorprotocol>.requestVideoInfo` 回调之后才生效。

接口

```
- (TXCVideoBasicInfo *)getVideoBasicInfo;
```

参数说明

TXCVideoBasicInfo 参数如下：

参数名	类型	描述
name	String	视频名称
size	Int	视频大小，单位：字节
duration	Float	视频时长，单位：秒
description	String	视频描述
coverUrl	String	视频封面

获取视频流信息

获取视频流信息列表，必须是在 `id<itxcplayerassistorprotocol>.requestVideoInfo` 回调之后才生效。

接口

```
- (TXCStreamingInfo *)getStreamingInfo;
```

参数说明

TXCStreamingInfo 参数如下：

参数名	类型	描述
playUrl	String	播放 URL
subStreams	List	自适应码流子流信息，类型为 TXCSubStreamInfo

TXCSubStreamInfo 参数如下：

参数名	类型	描述
type	String	子流的类型，目前可能的取值仅有 video
width	Int	子流视频的宽，单位：px
height	Int	子流视频的高，单位：px
resolutionName	String	子流视频在播放器中展示的规格名

获取关键帧打点信息

获取视频关键帧打点信息，必须是在 `id<itxcplayerassistorprotocol>.requestVideoInfo` 回调之后才生效。

接口

```
- (NSArray<TXCKeyFrameDescInfo *> *)getKeyFrameDescInfos;
```

参数说明

TXCKeyFrameDescInfo 参数如下：

参数名	类型	描述
timeOffset	Float	1.1
content	String	"片头开始..."

获取缩略图信息

获取缩略图信息，必须是在 `id<itxcplayerassistorprotocol>.requestVideoInfo` 回调之后才生效。

接口

```
- (TXCImageSpriteInfo *)getImageSpriteInfo;
```

参数说明

TCXImageSpriteInfo 参数如下：

参数名	类型	描述
imageUrls	List	缩略图下载 URL 数组，类型为 String
webVttUrl	String	缩略图 VTT 文件下载 URL

第三方播放器 Android 插件

最近更新时间：2022-10-17 11:16:01

第三方播放器 Android 插件为云点播提供给客户希望使用第三方播放器或自研播放器开发的对接云 PaaS 资源的播放器插件，常用于有自定义播放器功能需求的用户。

SDK 下载

第三方播放器 Android 插件和 Demo 项目下载地址 TXCPlayerAdapterSDK_Android。

集成指引

SDK 集成

集成 SDK，拷贝 `TXCPlayerAdapter-release-1.0.0.aar` 到 libs 目录，添加依赖项：

```
implementation(name:'TXCPlayerAdapter-release-1.0.0', ext:'aar')
```

添加混淆脚本：

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

使用播放器

变量声明，播放器主类为 `ITXCPlayerAssistor`，创建后即可播放视频。

fileId 一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 fileId 到客户端。
2. 服务端视频上传，在 确认上传 的通知中包含对应的 fileId。

如果文件已存在腾讯云，则可以进入 媒资管理，找到对应的文件。点开后在右侧视频详情中，可以看到相关参数。

```
//psign 即播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
private String mFileId, mPSign;
ITXCPlayerAssistor mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId, mPSign);
```

初始化：

```
// 初始化
TXCPlayerAdapter.init(appId); //appid 在腾讯云点播申请
TXCPlayerAdapter.setLogEnable(true); //开启log
mSuperPlayerView = findViewById(R.id.sv_videoplayer);
mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId, mPSign);
```

请求视频信息和播放：

```
mPlayerAssistor.requestVideoInfo(new ITXCRequestVideoInfoCallback() {
@Override
public void onError(int errCode, String msg) {
Log.d(TAG, "onError msg = " + msg);
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(VideoActivity.this, "onError msg = " + msg, Toast.LENGTH_SHORT).sh
ow();
}
});
}
@Override
public void onSuccess() {
Log.d(TAG, "onSuccess");
TXCStreamingInfo streamingInfo = mPlayerAssistor.getStreamingInfo();
Log.d(TAG, "streamingInfo = " + streamingInfo);
runOnUiThread(new Runnable() {
@Override
public void run() {
if (mPlayerAssistor.getStreamingInfo() != null) {
//播放视频
mSuperPlayerView.play(mPlayerAssistor.getStreamingInfo().playUrl);
} else {
Toast.makeText(VideoActivity.this, "streamInfo = null", Toast.LENGTH_SHORT).show
();
}
}
});
}
});
```

使用完后销毁 Player。

```
TXCPlayerAdapter.destroy();
```

SDK 接口说明

初始化 TXCPlayerAdatper

初始化 Adapter（每次）。

接口

```
TXCPlayerAdapter.init(String appId);
```

参数说明

appId：填写 appid（如果使用了子应用，则填 subappid）。

销毁 TXCPlayerAdatper

销毁 Adapter，当程序退出后调用。

接口

```
TXCPlayerAdapter.destroy();
```

创建播放器辅助类

通过播放器辅助类可以获取播放 fileId 相关信息以及处理 DRM 加密接口等。

接口

```
ITXCPlayerAssistor playerAssistor = TXCPlayerAdapter.createPlayerAssistor(String fileId, String pSign);
```

参数说明

参数名	类型	描述
fileId	String	要播放的视频 fileId
pSign	String	播放器签名

销毁播放器辅助类

销毁辅助类，在退出播放器或者切换了下一个视频播放的时候调用。

接口

```
TXCPlayerAdapter.destroyPlayerAssistor(ITXCPlayerAssistor assistor);
```

请求视频播放信息

本接口会请求腾讯云点播服务器，获取播放视频的流信息等。

接口

```
playerAssistor.requestVideoInfo(ITXCRequestVideoInfoCallback callback);
```

参数说明

参数名	类型	描述
callback	ITXCRequestVideoInfoCallback	异步回调函数

获取视频的基本信息

获取视频信息， 必须是在 `playerAssistor.requestPlayInfo` 回调之后才生效。

接口

```
TXCVideoBasicInfo playerAssistor.getVideoBasicInfo();
```

参数说明

TXCVideoBasicInfo 参数如下：

参数名	类型	描述
name	String	视频名称
duration	Float	视频时长，单位：秒
description	String	视频描述
coverUrl	String	视频封面

获取视频流信息

获取视频流信息列表，必须是在 `playerAssistor.requestPlayInfo` 回调之后才生效。

接口

```
TXCStreamingInfo playerAssistor.getStreamimgInfo();
```

参数说明

TXCStreamingInfo

参数名	类型	描述
playUrl	String	播放 URL
subStreams	List	自适应码流子流信息，类型为 SubStreamInfo

SubStreamInfo 参数如下：

参数名	类型	描述
type	String	子流的类型，目前可能的取值仅有 video
width	Int	子流视频的宽，单位：px
height	Int	子流视频的高，单位：px
resolutionName	String	子流视频在播放器中展示的规格名

获取关键帧打点信息

获取视频关键帧打点信息，必须是在 `playerAssistor.requestPlayInfo` 回调之后才生效。

接口

```
List<TXCKeyFrameDescInfo> playerAssistor.getKeyFrameDescInfo();
```

参数说明

TXCKeyFrameDescInfo 参数如下：

参数名	类型	描述
timeOffset	Float	1.1
content	String	"片头开始..."

获取缩略图信息

获取缩略图信息，必须是在 `playerAssistor.requestPlayInfo` 回调之后才生效。

接口

```
TXCImageSpriteInfo playerAssistor.getImageSpriteInfo();
```

参数说明

TCXImageSpriteInfo 参数如下：

参数名	类型	描述
imageUrls	List	缩略图下载 URL 数组，类型为 String
webVttUrl	String	缩略图 VTT 文件下载 URL

第三方播放器 Web 插件

最近更新时间：2022-10-17 11:42:21

本文档是介绍第三方播放器 Web 插件，它可以帮助腾讯云客户通过灵活的接口，快速实现第三方播放器与云点播能力的结合，实现视频播放功能。本插件支持获取视频基本信息、视频流信息、关键帧与缩略图信息等，支持私有加密，本文档适合有一定 Javascript 语言基础的开发人员阅读。

SDK 集成

第三方播放器 Web 插件提供 **CDN 集成**和 **npm 集成**两种集成方式：

CDN 集成

在需要播放视频的页面中引入初始化脚本，脚本会在全局下暴露 TcAdapter 变量。

```
<script src="https://cloudcache.tencentcs.com/qcloud/video/dist/tcadapter.1.0.0.min.js"></script>
```

npm 集成

```
// npm install
npm install tcadapter --save
// import TcAdapter
import TcAdapter from 'tcadapter';
```

放置播放器容器

在需要展示播放器的页面加入容器，TcAdapter 仅需要承载播放视频的容器，播放样式和自定义功能可由第三方播放器或使用者自行实现：

```
<video id="player-container-id">
</video>
```

SDK 使用说明

检测开发环境

检测当前环境是否支持 TcAdapter。

```
TcAdapter.isSupported();
```

初始化 Adapter

初始化 Adapter，创建 Adapter 实例。初始化过程会请求腾讯云点播服务器，获取视频文件信息。

接口

```
const adapter = new TcAdapter('player-container-id', {
fileID: string,
appID: string,
psign: string,
hlsConfig: {}
}, callback);
```

参数说明

参数名	类型	描述
appID	String	点播账号的 APPID
fileID	String	要播放的视频 fileId
psign	String	播放器签名
hlsConfig	HlsConfig	HLS 相关设置，可使用 `hls.js` 支持的任意参数
callback	TcAdapterCallBack	初始化完成回调，可以在此方法之后获取视频基本信息

> 注意：
> TcAdapter 底层基于 `hls.js` 实现，可以通过 HlsConfig 接收 `hls.js` 支持的任意参数，用于对播放行为的精细调整。

获取视频基本信息

获取视频的信息， 必须是在初始化之后才生效。

接口

```
VideoBasicInfo adapter.getVideoBasicInfo();
```

参数说明

VideoBasicInfo 参数如下：

参数名	类型	描述
name	String	视频名称
duration	Float	视频时长，单位：秒
description	String	视频描述
coverUrl	String	视频封面

获取视频流信息

接口

```
List<StreamingOutput> adapter.getStreamimgOutputList();
```

参数说明

StreamingOutput 参数如下：

参数名	类型	描述
drmType	String	自适应码流保护类型，目前取值有 plain 和 simpleAES。plain 表示不加密，simpleAES 表示 HLS 普通加密
playUrl	String	播放 URL
subStreams	List	自适应码流子流信息，类型为 SubStreamInfo

SubStreamInfo 参数如下：

参数名	类型	描述
type	String	子流的类型，目前可能的取值仅有 video
width	Int	子流视频的宽，单位：px
height	Int	子流视频的高，单位：px
resolutionName	String	子流视频在播放器中展示的规格名

获取关键帧打点信息

接口

```
List<KeyFrameDescInfo> adapter.getKeyFrameDescInfo();
```

参数说明

KeyFrameDescInfo 参数如下：

参数名	类型	描述
timeOffset	Float	1.1
content	String	"片头开始..."

获取缩略图信息

接口

```
ImageSpriteInfo adapter.getImageSpriteInfo();
```

参数说明

ImageSpriteInfo 参数如下：

参数名	类型	描述
imageUrls	List	缩略图下载 URL 数组，类型为 String
webVttUrl	String	缩略图 VTT 文件下载 URL

监听事件

播放器可以通过初始化返回的对象进行事件监听，示例：

```
const adapter = TcAdapter('player-container-id', options);
adapter.on(TcAdapter.TcAdapterEvents.Error, function(error) {
// do something
});
```

其中 type 为事件类型，支持的事件包括 HLS 原生的事件以及以下事件，可从 `TcAdapter.TcAdapterEvents` 中访问到事件名称：

名称	介绍
LOADEDMETADATA	通过 playcgi 获取到了相应的视频信息，在此事件回调中可以获取视频相关信息

名称	介绍
HLSREADY	hls实例创建完成，可以在此时机调用 hls 实例对象上的各种属性和方法
ERROR	出现错误时触发，可从回调参数中查看失败具体原因

获取 Hls 实例

adapter 底层基于 hls.js 实现，可以通过 adapter 实例访问到 HLS 实例以及实例上的属性和方法，用于实现对播放流程的精细控制。

```
adapter.on('hlsready', () => {
const hls = adapter.hls;
// ...
})
```

> 说明：
> 具体请参见 hls.js。

示例

例1：在 React 中使用 TcAdapter

具体示例，请参见 GitHub。

```
import { useEffect, useRef } from 'react';
import TcAdapter from 'tcadapter';
function App() {
if (!TcAdapter.isSupported()) {
throw new Error('current environment can not support TcAdapter');
}
const videoRef = useRef(null);
useEffect(() => {
const adapter = new TcAdapter(videoRef.current, {
appID: '1500002611',
fileID: '5285890813738446783',
psign: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwMjYxMSwiZmlsZUlk
IjoiNTI4NTg5MDgxMzczODQ0Njc4MyIsImN1cnJlbnRUaW1lU3RhbXAiOjE2MTU5NTEyMzksImV4cGlyZ
VRpbWVTdGFtcCI6MjIxNTY1MzYyMywicGNmZyI6ImJhc2ljRHJtUHJlc2V0IiwidXJsQWNjZXNzSW5mby
I6eyJ0IjoiMjIxNTY1MzYyMyJ9fQ.hRrQYvC0UYtcO-ozB35k7LZI6E3ruvow7DC0XzzdYKE',
hlsConfig: {},
```

```
}, () => {
console.log('basicInfo', adapter.getVideoBasicInfo());
});
adapter.on(TcAdapter.TcAdapterEvents.HLSREADY, () => {
const hls = adapter.hls;
// ...
})
}, []);

const play = () => {
videoRef.current.play();
}
return (
<div>
<div>
<video id="player" ref={ videoRef }></video>
</div>
<button onClick={play}>play</button>
</div>
);
}
export default App;
```

例2：TcAdapter 与 videojs 结合

具体示例，请参见 GitHub。

```
// 1. videojs 播放 hls 会使用 @videojs/http-streaming，所以我们开发一套使用 tcadapter 播放的策略覆盖原有逻辑（也可以直接修改 @videojs/http-streaming 内部逻辑）
// src/js/index.js
import videojs from './video';
import '@videojs/http-streaming';
import './tech/tcadapter'; // 新增逻辑
export default videojs;
// src/js/tech/tcadapter.js
import videojs from '../video.js';
import TcAdapter from 'tcadapter';
class Adapter {
constructor(source, tech, options) {
const el = tech.el();
// 获取参数并初始化实例
const adapter = new TcAdapter(el, {
appID: '1500002611',
fileID: '5285890813738446783',
psign: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwMjYxMSwiZmlsZUlkIjoiNTI4NTg5MDgxMzczODQ0Njc4MyIsImN1cnJlbnRUaW1lU3RhbXAiOjE2MTU5NTEyMzksImV4cGlyZ
```

```
VRpbWVTdGFtcCI6MjIxNTY1MzYyMywicGNmZyI6ImJhc2ljRHJtUHJlc2V0IiwidXJsQWNjZXNzSW5mby
I6eyJ0IjoiMjIxNTY1MzYyMyJ9fQ.hRrQYvC0UYtcO-ozB35k7LZI6E3ruvow7DC0XzzdYKE',
hlsConfig: {},
});
adapter.on(TcAdapter.TcAdapterEvents.LEVEL_LOADED, this.onLevelLoaded.bind(this
));
}
dispose() {
this.hls.destroy();
}
onLevelLoaded(event) {
this._duration = event.data.details.live ? Infinity : event.data.details.totaldur
ation;
}
}
let hlsTypeRE = /^application\/(x-mpegURL|vnd\.apple\.mpegURL)$/i;
let hlsExtRE = /\.m3u8/i;
let HlsSourceHandler = {
name: 'hlsSourceHandler',
canHandleSource: function (source) {
// skip hls fairplay, need to use Safari resolve it.
if (source.skipHlsJs || (source.keySystems && source.keySystems['com.apple.fps.1_
0'])) {
return '';
} else if (hlsTypeRE.test(source.type)) {
return 'probably';
} else if (hlsExtRE.test(source.src)) {
return 'maybe';
} else {
return '';
}
},
handleSource: function (source, tech, options) {
if (tech.hlsProvider) {
tech.hlsProvider.dispose();
tech.hlsProvider = null;
} else {
// hls关闭自动加载后，需要手动加载资源
if (options.hlsConfig && options.hlsConfig.autoStartLoad === false) {
tech.on('play', function () {
if (!this.player().hasStarted()) {
this.hlsProvider.hls.startLoad();
}
});
}
}
tech.hlsProvider = new Adapter(source, tech, options);
```

```
return tech.hlsProvider;
},
canPlayType: function (type) {
if (hlsTypeRE.test(type)) {
return 'probably';
}
return '';
}
};
function mountHlsProvider(enforce) {
if (TcAdapter && TcAdapter.isSupported() || !!enforce) {
try {
let html5Tech = videojs.getTech && videojs.getTech('Html5');
if (html5Tech) {
html5Tech.registerSourceHandler(HlsSourceHandler, 0);
}
} catch (e) {
console.error('hls.js init failed');
}
} else {
//没有引入tcadapter 或者 MSE 不可用或者x5内核禁用
}
}
mountHlsProvider();
export default Adapter;
```

Player SDK Policy
Privacy Policy

最近更新时间：2023-11-30 16:10:20

1. INTRODUCTION

This Module applies if you use Player SDK ("**Feature**"). This Module is incorporated into the privacy policy located at Privacy Policy. Terms used but not defined in this Module shall have the meaning given to them in the Privacy Policy. In the event of any conflict between the Privacy Policy and this Module, this Module shall apply to the extent of the inconsistency.

2. CONTROLLERSHIP

The controller of the personal information described in this Module is as specified in the Privacy Policy.

3. AVAILABILITY

This Feature is available to users globally but primarily intended for users located in the same country/region as the selected service region for optimal performance.

4. HOW WE USE PERSONAL INFORMATION

We will use the information in the following ways and in accordance with the following legal basis:

Personal Information	Use	Legal Basis

Personal Information	Use	Legal Basis
• **Operation Data**: configuration information of customer's Tencent Cloud console, information about the customer's network fluctuations, audio and video quality problems, SDK logs relating to the use of the SDK, customer SDK version number and OS type • **Log Data:** log files of the customer's backend, e.g. network download log, video decoding and encoding log, rendering screen log, API interface call log	We use this information for troubleshooting, operation and maintenance analysis. Please note that this data is stored and backed up in TencentDB for MySQL ("MySQL").	We process this information as it is necessary for us to perform our contract with you to provide the Feature.
End User Network and Device Information: WiFi status, system properties, device model, operating system, sensor information, user IP address and user agent information.	We use this information for: • playback quality analysis; and • troubleshooting, operation and maintenance analysis. Please note that this data is stored and backed up in our MySQL feature.	We process this information as it is necessary for us to perform our contract with you to provide the Feature.

Personal Information	Use	Legal Basis
Administrative Data: customer's app name, application package name (identifier for listing app on application store), bundle ID (identifier for listing app on application store), appID. **Customer License Data**: license type, license url and license key (to allow you to unlock access to the SDK), license details (expiry date, function module, automatic renewal or not).	We use this information to: • provide and allow you to access the Feature; and • for troubleshooting analysis. Please note that this data is stored and backed up in our MySQL feature.	We process this information as it is necessary for us to perform our contract with you to provide the Feature.

5. HOW WE SHARE AND STORE PERSONAL INFORMATION

As specified in the Privacy Policy.

6. DATA RETENTION

We will retain personal information in accordance with the following:

Personal Information	Retention Policy
Operational data	Log data: We retain such data for 14 days. Operational data other than log data: We retain such data for as long as you use the Feature. When your use of the Feature is terminated, we will delete this data within 5 days.
End user network and device data	Automatically deleted after 14 days or for a longer or shorter retention period as requested by you. Under special circumstances, if the user account is attacked or hacked, we will delete the data.
Administrative data	This data will be deleted within 5 days of your request to deactivate your Tencent Cloud account, or immediately upon your data deletion request. Please note that if you submit a data deletion request, we will not be able to continue to provide you with the Feature.