实时音视频

含 UI 集成方案

产品文档



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

含 UI 集成方案
TUIKit 概述
音视频通话
组件介绍（TUICallKit）
开通服务（TUICallKit）
快速接入（TUICallKit）
Android
iOS
Web&H5 (React)
Web&H5 (Vue3)
Flutter
uniapp（客户端）
界面定制（TUICallKit）
Android
iOS
Web
Flutter
离线唤醒（TUICallKit）
iOS
VoIP
APNs
Android
Flutter
云端录制（TUICallKit）
更多特性（TUICallKit）
设置昵称、头像（全平台）
设置分辨率、填充模式(Web)
群组通话
Android&iOS&Flutter
Web&H5
uni-app (Anroid&iOS)
悬浮窗
Android&iOS&Flutter
Web&H5
uni-app (Anroid&iOS)

版权所有：腾讯云计算（北京）有限责任公司

美颜特效
Flutter
自定义铃声
Android
iOS
Web&H5
uni-app (Anroid&iOS)
Flutter
监听通话状态
Android&iOS&Flutter
Web&H5
uni-app（Android & iOS）
客户端 API（TUICallKit）
Android
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义
iOS
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义
Web
API 概览
TUICallKit
TUICallEngine
TUICallEvent
Flutter
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义
uniapp（Android&iOS）
API 概览

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit
TUICallEngine
事件
错误码表
发布日志（TUICallKit）
Web
Android & iOS
Flutter
常见问题（TUICallKit）
Web
全平台
Flutter
多人音视频
组件介绍 (TUIRoomKit)
开通服务（TUIRoomKit）
快速接入 (TUIRoomKit)
iOS
Android
Electron
Windows
Web
Flutter
界面定制 (TUIRoomKit)
Electron
Android
iOS
Windows
Web
Flutter
会议控制 (TUIRoomKit)
API 参考 (TUIRoomKit)
iOS&Mac
API 概览
TUIRoomKit
TUIRoomEngine
TUIRoomObserver
类型定义
Android

API 概览
TUIRoomKit
TUIRoomEngine
TUIRoomObserver
类型定义
Windows
API 概览
TUIRoomEngine
TUIRoomObserver
类型定义
Web
API 概览
TUIRoomKit
TUIRoomEngine
TUIRoomEvents
TUIRoomEngine Defines
Electron
API 概览
TUIRoomKit
TUIRoomEngine
TUIRoomEvent
TUIRoomEngine Defines
Flutter
API概览
TUIRoomObserver
TUIRoomEngine
类型定义
常见问题
Web
iOS
Android
Electron
Flutter
错误码
语音互动直播
集成 TUIVoiceRoom (Android)
集成 TUIVoiceRoom (iOS)
TUIVoiceRoom API 查询

版权所有：腾讯云计算（北京）有限责任公司

TRTCVoiceRoom (iOS)
TRTCVoiceRoom (Android)
视频互动直播
集成 TUILiveRoom (Android)
集成 TUILiveRoom (iOS)
集成 TUIPusher&TUIPlayer (Web)
TUILiveRoom API 查询
TRTCLiveRoom API (iOS)
TRTCLiveRoom API (Android)
在线 K 歌
快速集成（TUIKaraoke）
iOS
Android
方案介绍（TUIKaraoke）
实现步骤
歌曲同步
iOS
Android
歌词同步
iOS
Android
人声同步
iOS
Android
混流方案
iOS
Android
API 参考 （TUIKaraoke）
TRTCKaraoke(iOS)
TRTCKaraoke(Android)
常见问题（TUIKaraoke）
iOS
Android

含 UI 集成方案

TUIKit 概述

最近更新时间：2024-03-06 15:48:27

TUIkit 是腾讯云音视频团队在5000+客户的服务积累中，结合业内主流的音视频场景，提炼出的开源解决方案，包含视频通话组件、直播组件、视频房间组件等多个客户端音视频组件，可以帮助开发者快速搭建诸如通话、客服、直播、语聊、教育等场景解决方案。

说明

TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，在实时音视频控制台创建应用后，我们控制台会自动帮您创建一个相同 SDKAppID 的即时通信 IM 应用，即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

产品动态

音视频通话（TUICallKit）

TUICallKit 是腾讯云推出一款新的音视频通话 UI 组件，是 TUICalling 的升级版本，TUICallKit 体积更小，稳定性更好，也支持更多特性：

更加简单易用的 API、更加全面的通话回调、集成包体体积更小。

全面升级群组通话功能：支持通话过程中邀请他人加入，主动加入通话等全新能力。

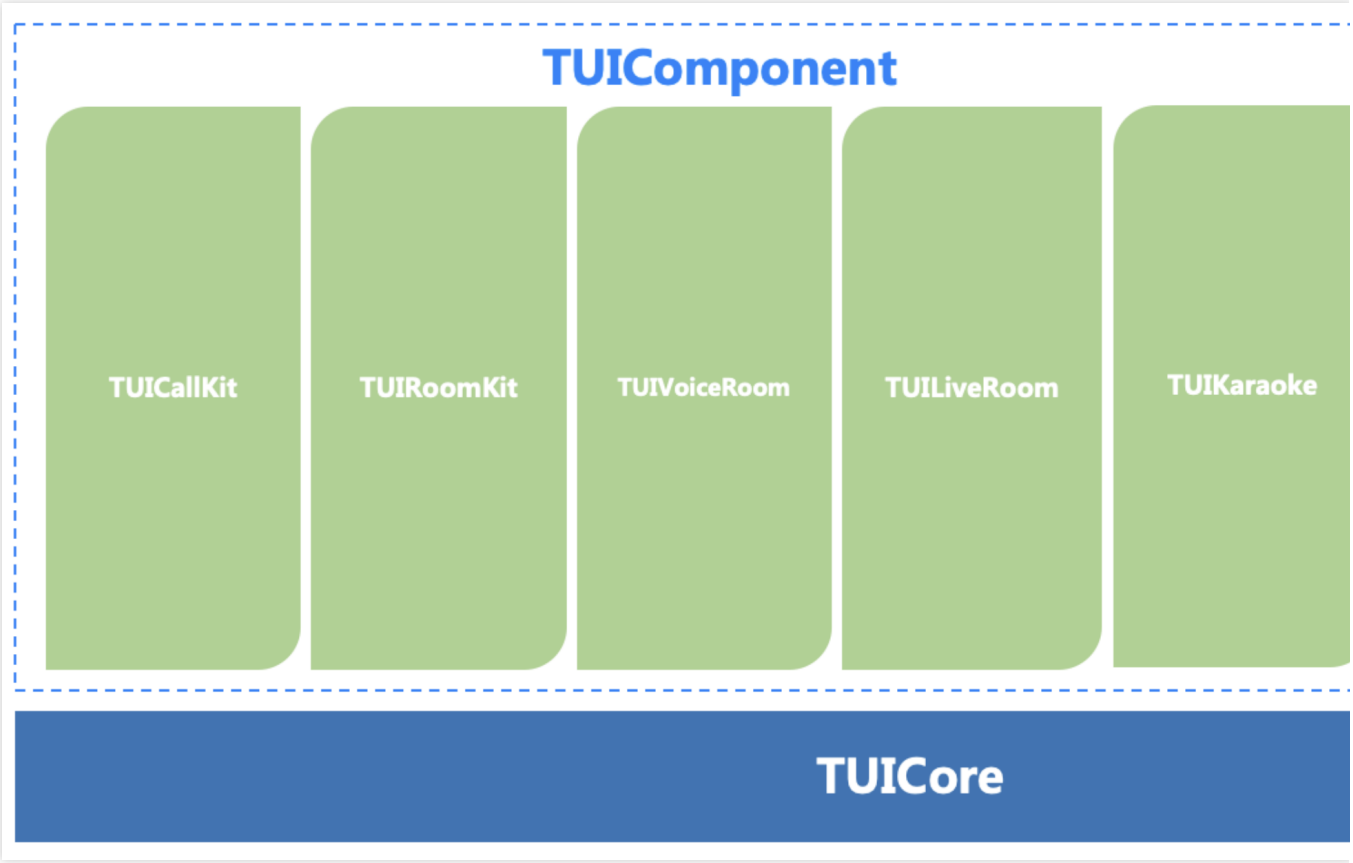
更加灵活的参数设置：新增美颜强度设置，新增编码参数设置，新增渲染参数设置。

新增音视频通话专属功能：新增弱网环境通话卡顿优化、A 降噪能力、通话悬浮窗。

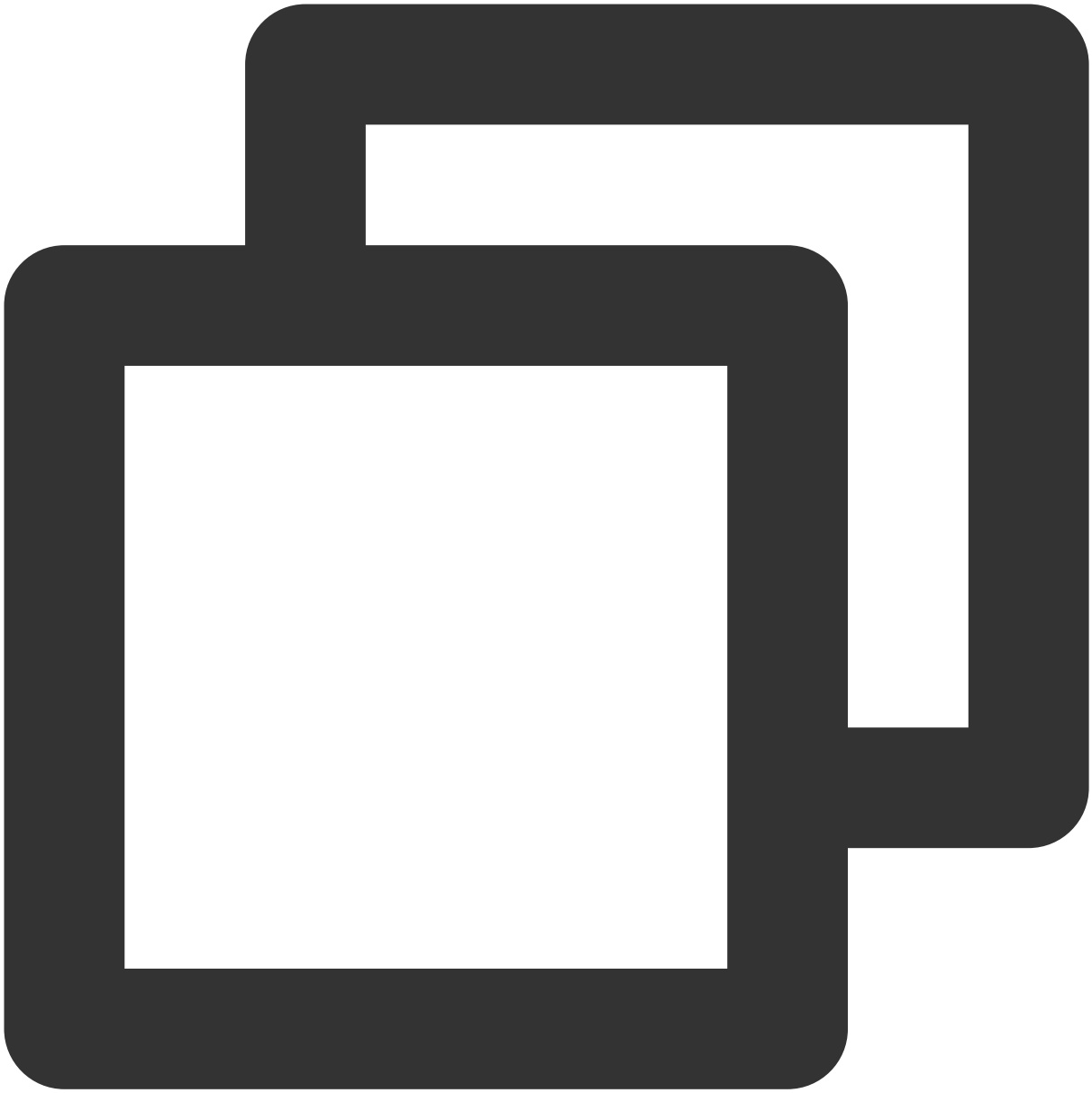
更多特性：支持多端登录、支持服务端结束通话等等。

期待您的的使用，更多信息，请点击这里：组件介绍（TUICallKit）

TUIKit 全家桶



如上图所示，TUIKit 分为 TUIComponent 和 TUIWidget 两种。



```
├── TUIComponent
│   ├── TUICallKit      // 通话组件（类微信通话），针对视频通话、客服、金融审核等音视频场景；
│   ├── TUIRoomKit      // 多人视频房间组件，针对教育、会议、面试等音视频场景，具备禁言禁画等功
│   ├── TUIVoiceRoom    // 语聊直播组件，针对交友语聊、排队语聊，音乐房等音频场景；
│   ├── TUILiveRoom     // 视频互动直播组件，具备连麦、PK、音效等功能；
│   ├── TUIKaraoke      // 创新组件，针对在线KTV、在线合唱等新玩法的音视频的组件，可以搭配腾讯云
├── TUIWidget
│   ├── TUIBeauty       // 美颜挂件
│   ├── TUIAudioEffect  // 音效挂件
│   ├── TUIBarrage      // 弹幕挂件
│   ├── TUIGift         // 礼物挂件；
```

交流&反馈

当然，一样的环境，一样的步骤，可能也会有不一样的报错，这是工程师的逃不开的魔咒，您可以通过如下方式来尝试解决：

常见问题：TUICallKit、TUIRoomKit；

TUIKit 技术交流 QQ 群：592465424

音视频通话
组件介绍（TUICallKit）

最近更新时间：2024-04-03 17:23:11

组件介绍

TUICallKit 是腾讯云推出一款音视频通话 UI 组件，通过集成该组件，您只需要编写几行代码就可以为您的应用添加音视频通话功能。



支持平台

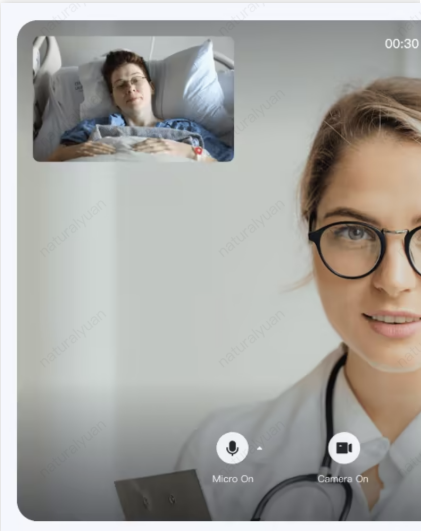
平台	Android	iOS	Web	uni-app 小	Flutter	uni-app 客	微信小程序

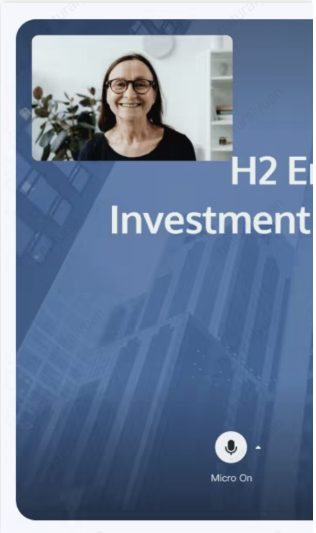
				程序		**户端**	
是否支持							
支持语言/框架	Kotlin Java	Swift Objective-C	Vue3 Vue2 React	Vue3 Vue2	Dart	Vue3 Vue2	

功能介绍

基本功能	**高级功能**	**功能优势**
1v1 语音/视频通话 **群组通话**、中途邀请他人、中途加入 自定义来电铃声 自定义昵称、头像 **开启/关闭悬浮窗** 开启/关闭来电铃声	**离线推送** **虚拟背景** 云端录制 AI 降噪 全球互通 弱网卡顿优化 通话记录	**完善的 UI 交互** **支持多平台互联互通** **支持多设备登录**

适用场景

在线社交	**在线问诊**
	

在线教育	在线销售
	

在线体验

平台	Web	Android	iOS	其它
在线体验				/
demo 集成	Github: web	Github: Andorid	Github: iOS	Github: TUICallKit Demo

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com 。

开通服务（TUICallKit）

最近更新时间：2024-04-03 18:02:21

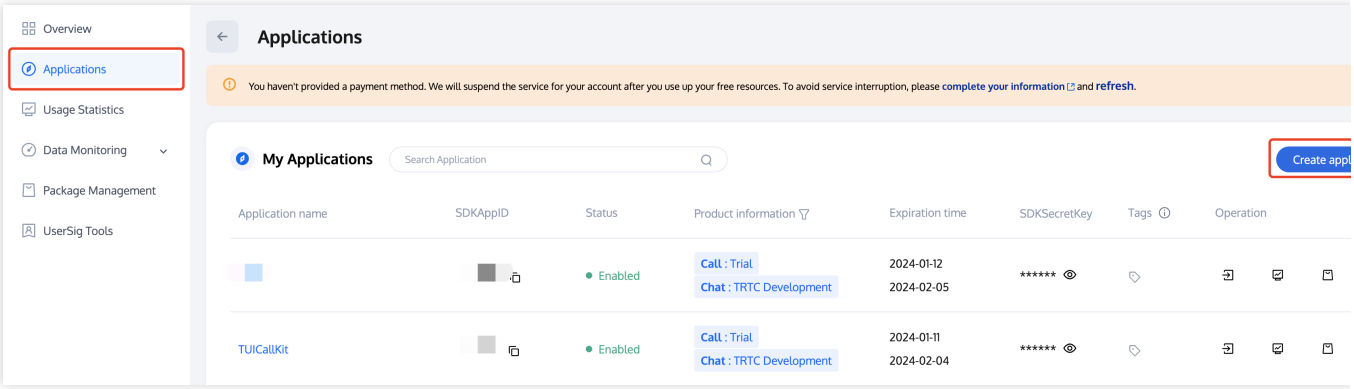
开通和购买

开通体验服务

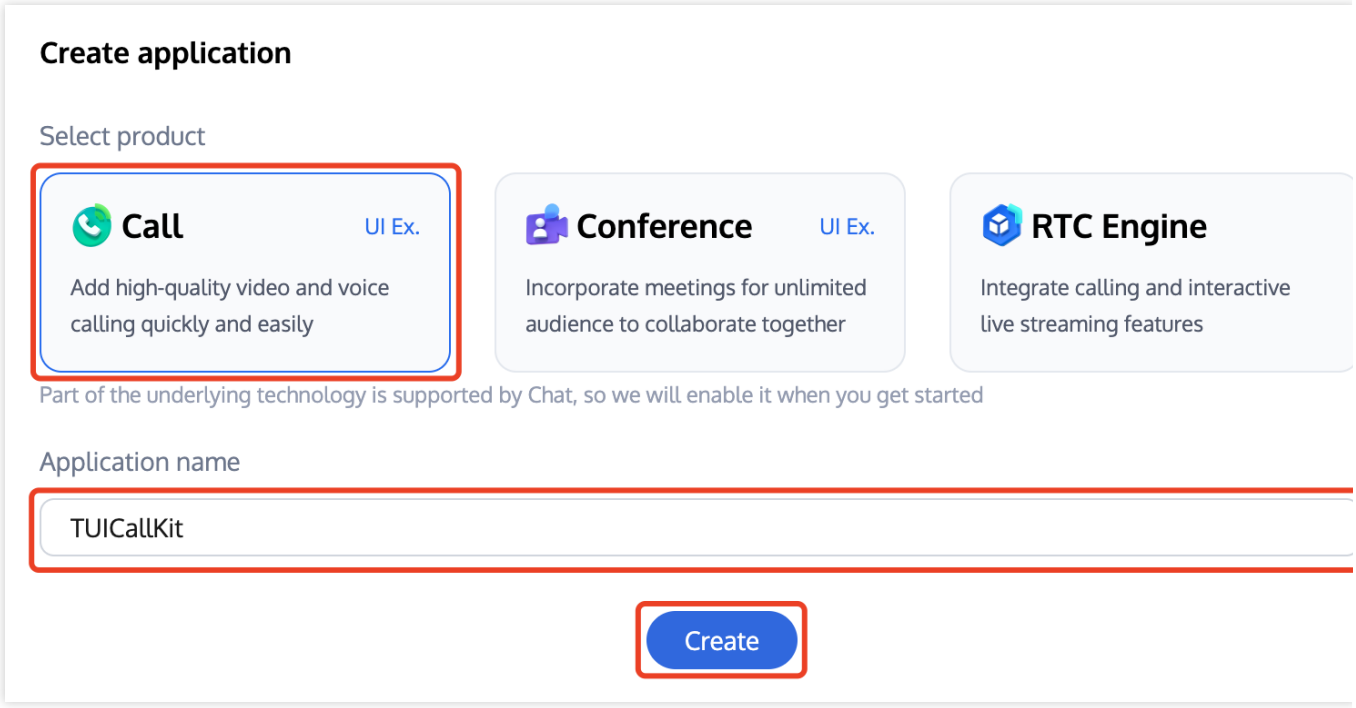
为了您能更好地体验音视频通话 Call 功能，我们免费为每个 SDKAppID 提供了7天体验版（体验版不额外赠送通话时长），每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。

您可以在 实时音视频 TRTC 控制台中开通 Call 体验版，具体操作步骤如下：

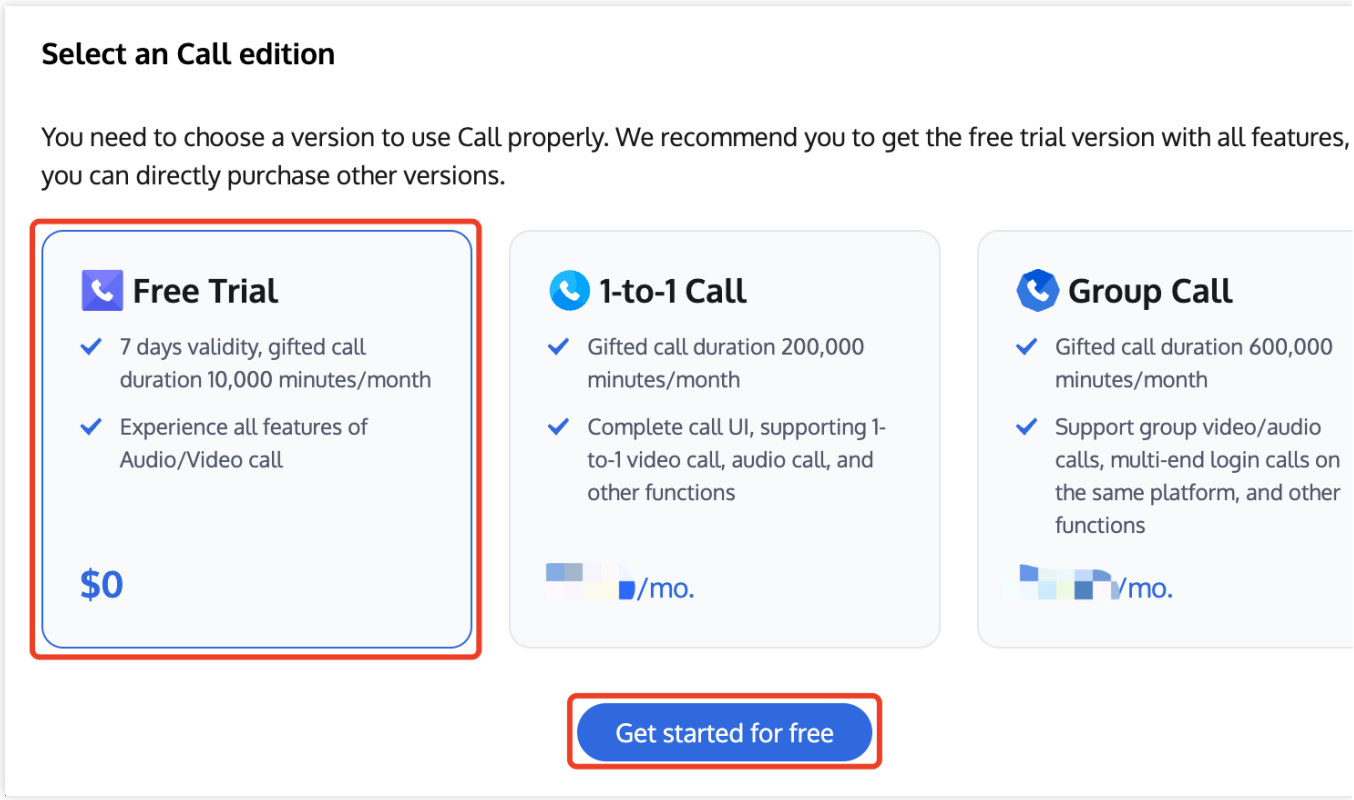
1. 登录到 实时音视频 TRTC 控制台 > Applications，点击 **Create application**。



2. 在创建弹窗中选择 **Call** 并输入应用名称，并点击 **Create**。



3. 完成应用创建后将默认进入应用详情页，在浮窗中选择 **Free Trial**，并点击 **Get started for free**。



4. 确认弹窗内容后，单击 **Free Trial**，即可成功开通音视频通话体验版。

Free trial of TRTC Call

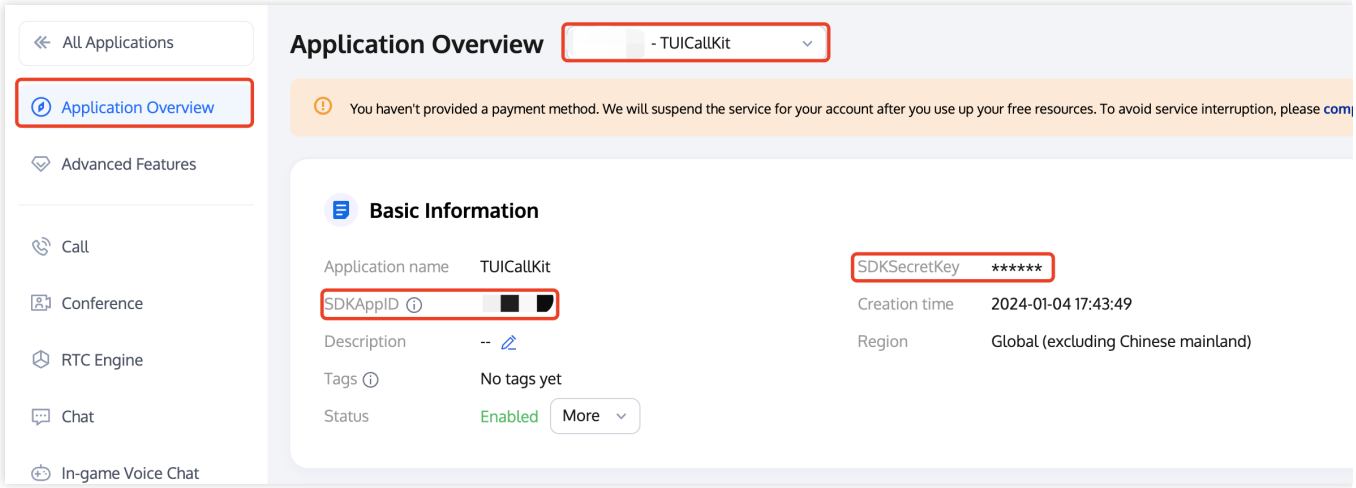
TRTC Call (TUICallKit) is a UI component for audio/video calling. After integrating it and writing a few simple lines of code, your applic will be able to support 1-to-1/Group calls, multi-platform call, and offline call push and other features.



- Call provides a **Free trial with 7 days validity**, allowing you to experience all features of Call for free.
- Some underlying capabilities of Call are based on Chat. If your application has not yet activated Chat, this will activate the Chat development version (free) for you; if your application has already activated Chat, this will not change your existing Chat version and configuration.
- Call Free version **only provides you with the right to use the above functions for free. The Usage time generated during your us will be billed according to Audio/Video call time, and the Chat fees will be billed according to Chat Billing overview**
- Tencent Cloud accounts (UIN) that have activated Call will receive a free 10,000-minute Package of Time every month, which ca used to offset the usage time generated during use. For more information, see Audio and Video Call.

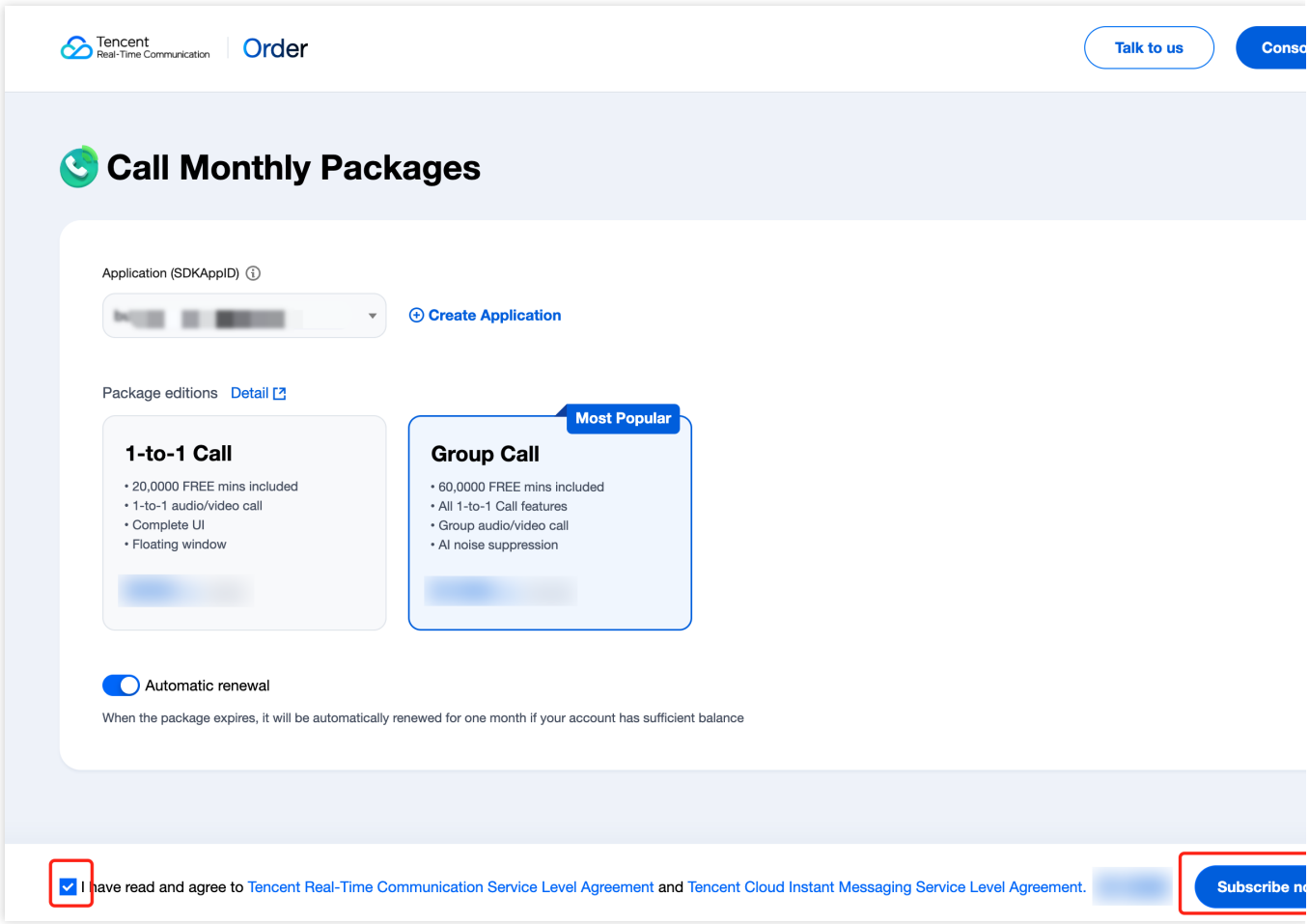
Free Trail

5. 开通完成后可在当前页面查看信息，并可参见 集成指引 进行集成，这里的 `SDKAppID` 、 `SDKSecretKey` 会在集成指引中使用到。

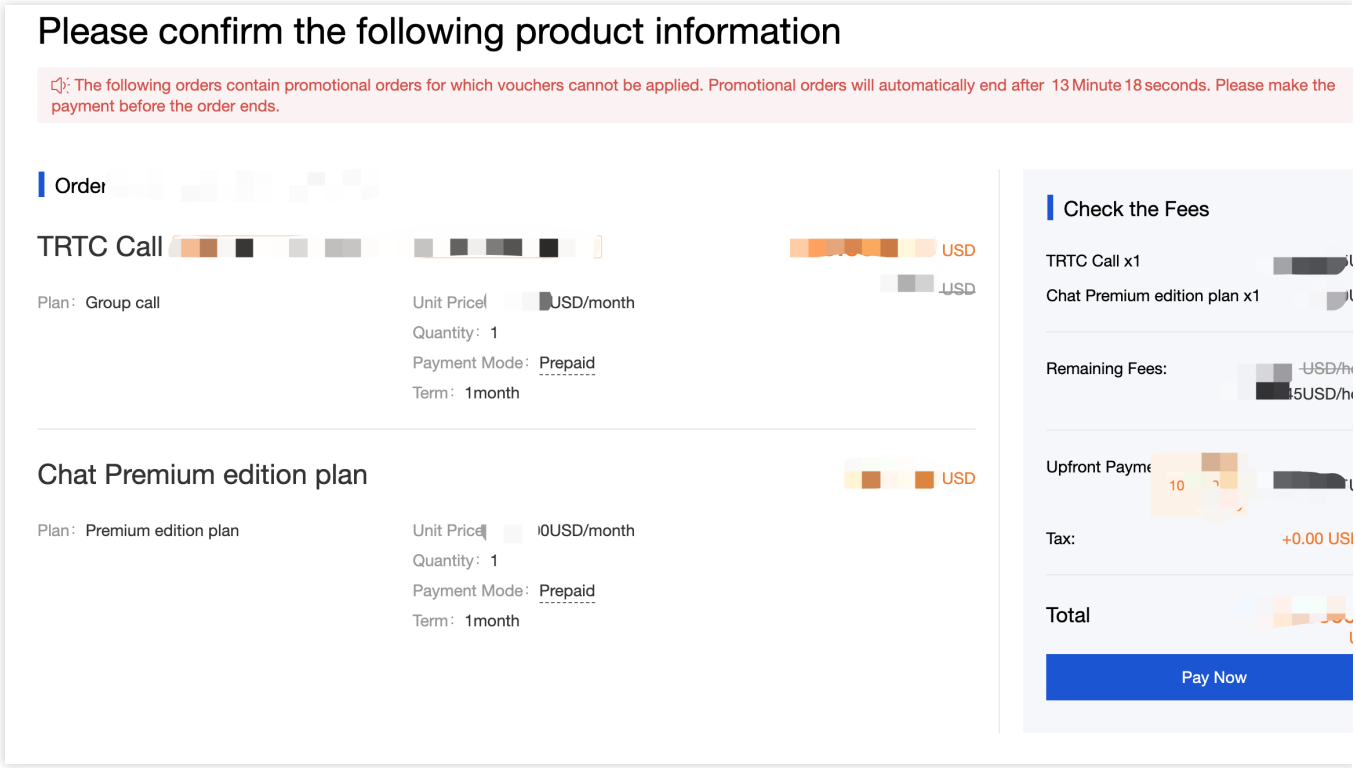


购买正式版

1. 访问 音视频通话购买页面，选择需要购买的应用（SDKAppID）及版本，同时建议您开启自动续期以避免影响业务使用。确认购买信息后并同意相关协议后，勾选协议内容并点击 **Subscribe now**。



2. 前往订单确认页确认商品信息。Call 部分底层服务由即时通信 IM 提供支持，因此 Call 包月套餐中将默认包含一个 IM 基础套餐包，订单中将包含2款商品。（如下图所示）



3. 前往支付页完成支付。购买完成后，您可前往 TRTC 控制台 查看应用版本信息，并可参见 集成指引 进行集成。

说明：

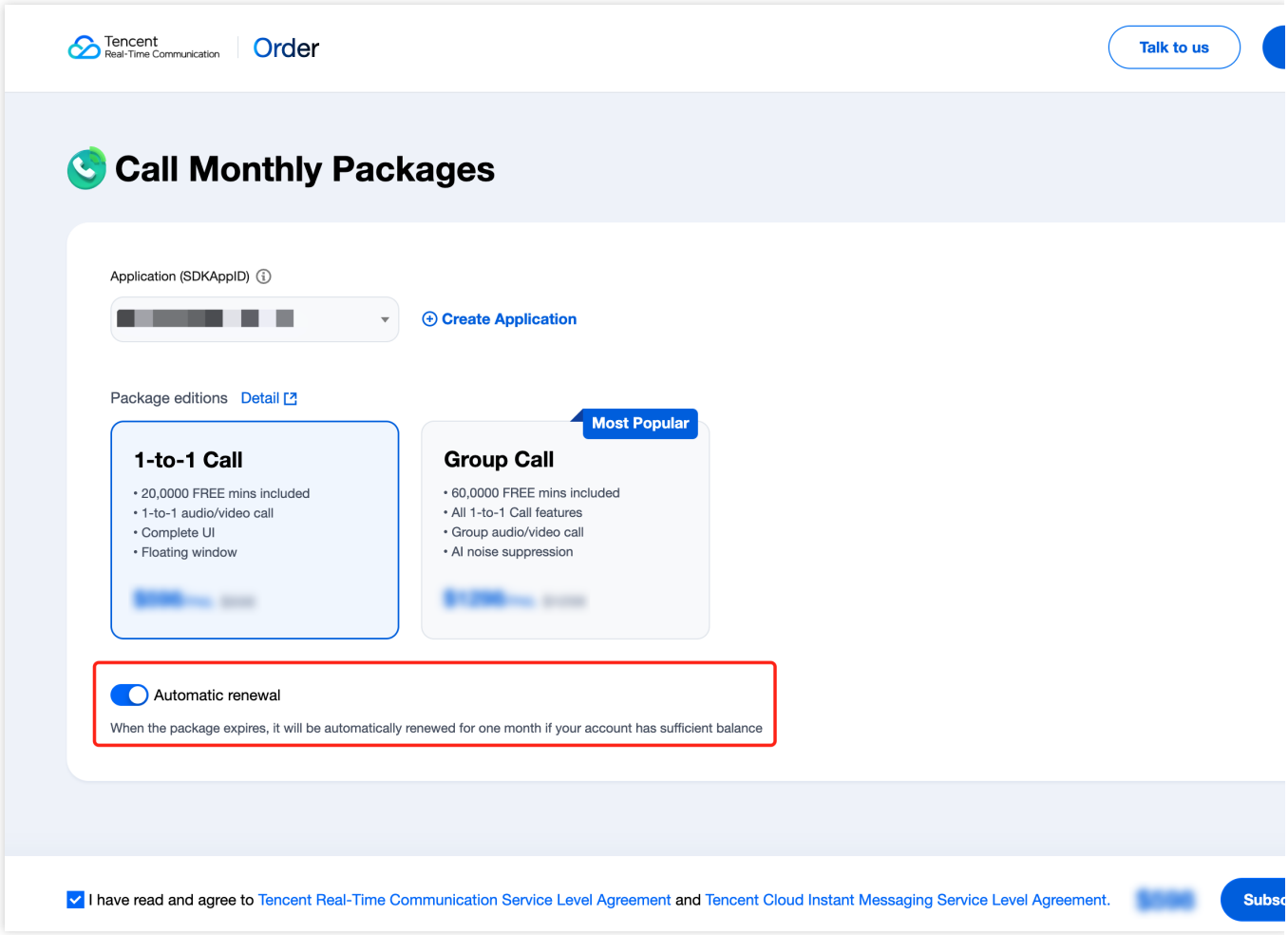
若您已在即时通信 IM 创建过应用并已购买 IM 付费版本，建议**您参考 这里 购买和使用 Call。**

续费正式版

参考 购买正式版 的流程再次购买相同版本的 Call 包月套餐，即可完成续费。

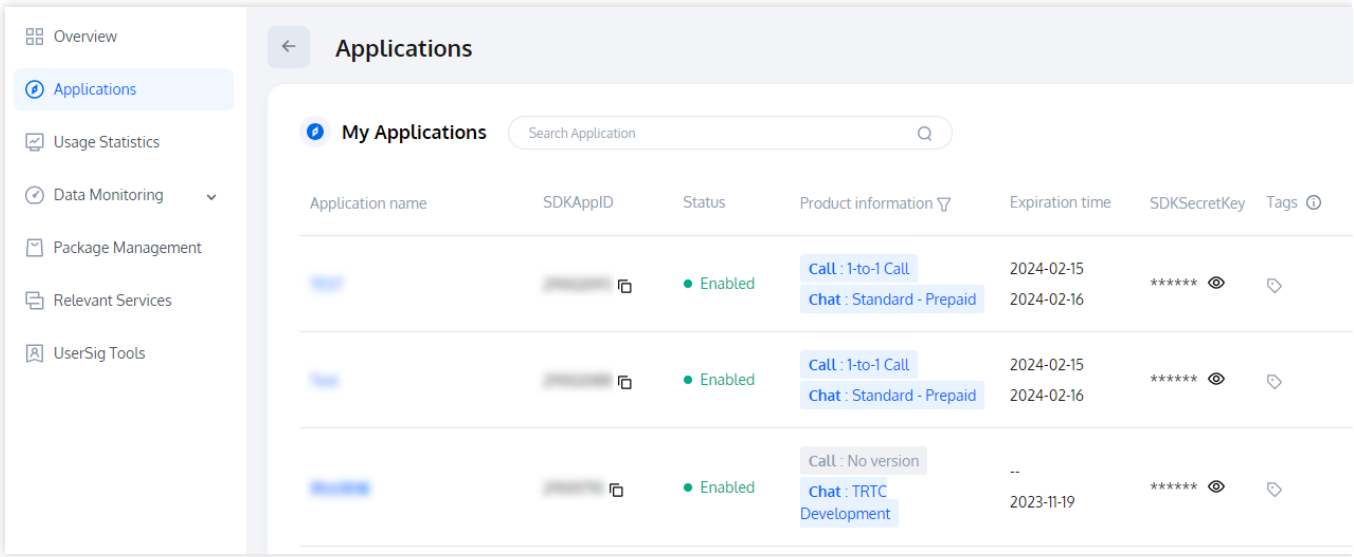
建议您通过开启自动续期的方式为 Call 续费，开启后将账户 **余额足够时，到期后自动按月续费**。由于 Call 部分底层服务由即时通信 IM 提供支持，因此将同时开启 Call 和 IM 的自动续期。您可在购买时开启自动续费，也可在购买完成后在控制台开启。

版权所有：腾讯云计算（北京）有限责任公司

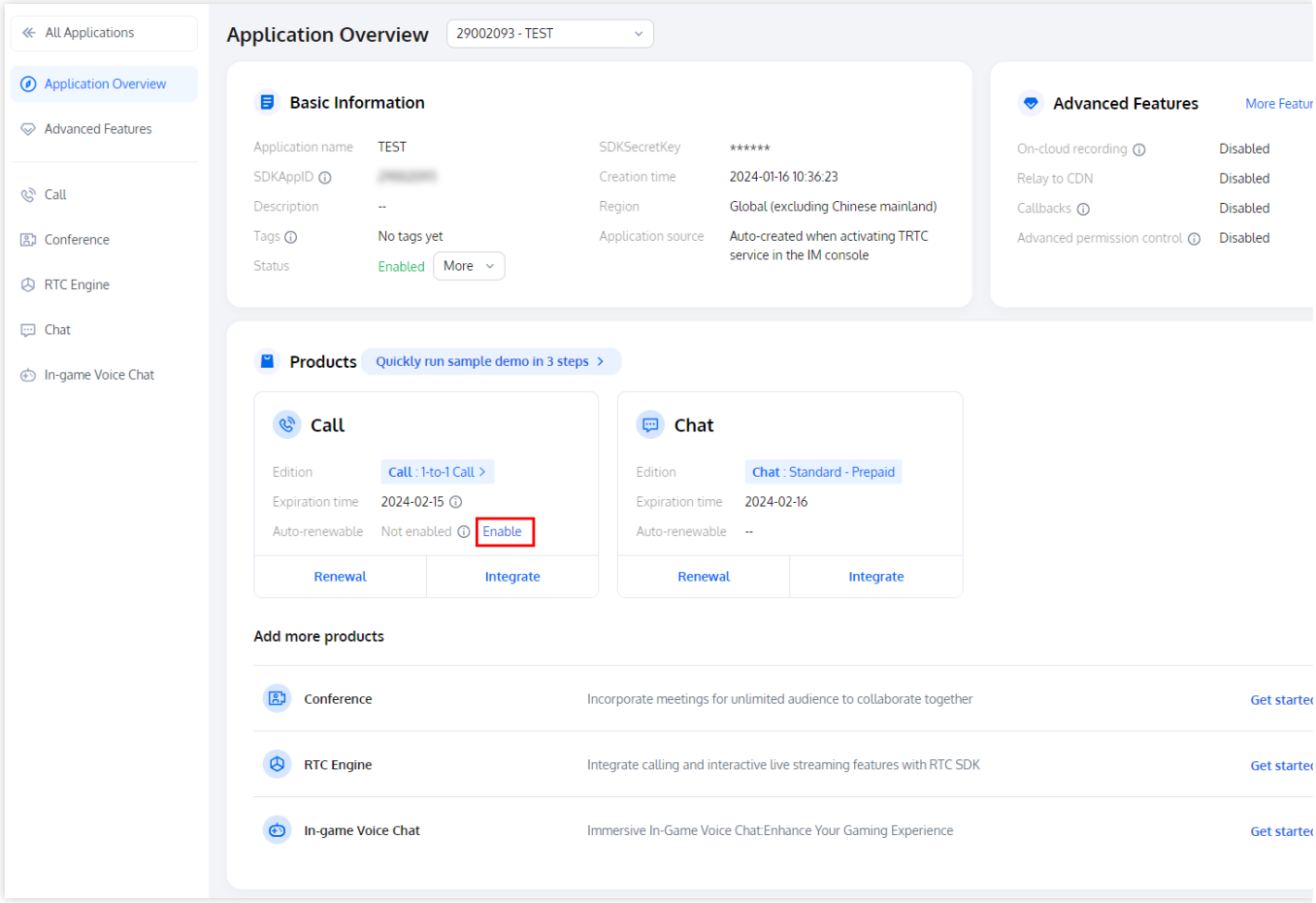


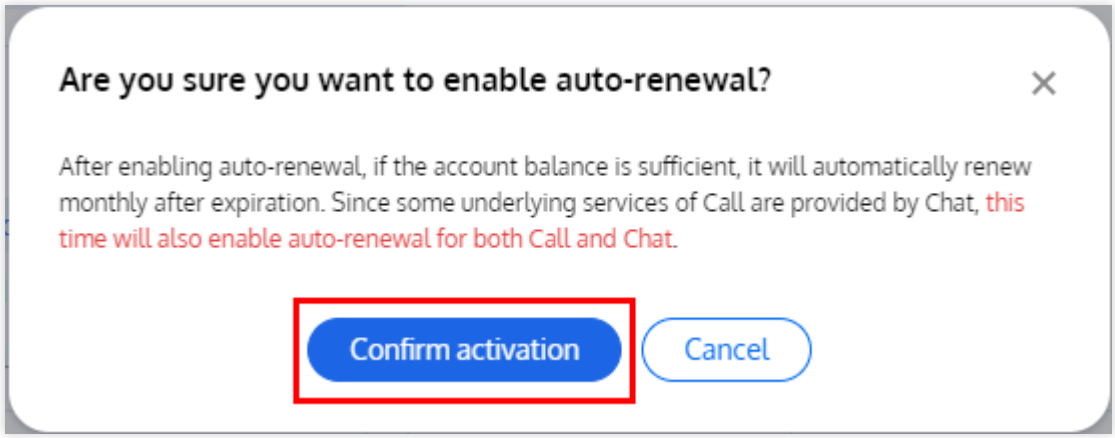
控制台开启自动续期的具体步骤如下：

1. 访问 TRTC 控制台 > Applications，选择需要开启自动续费的应用并点击**管理**进入应用详情页。



2. 在 Call 产品信息中，点击 **开启自动续期** 按钮，将出现确认弹窗，点击**确认开启**。


All Applications
Application Overview
Advanced Features
Call
Conference
RTC Engine
Chat
In-game Voice Chat
Application Overview
29002093 - TEST
Basic Information
Application name
TEST
SDKAppID
Description
--
Tags
No tags yet
Status
Enabled
More
SDKSecretKey
******
Creation time
2024-01-16 10:36:23
Region
Global (excluding Chinese mainland)
Application source
Auto-created when activating TRTC service in the IM console
Advanced Features
More Featur
On-cloud recording
Disabled
Relay to CDN
Disabled
Callbacks
Disabled
Advanced permission control
Disabled
Products
Quickly run sample demo in 3 steps
Call
Edition
Call : 1-to-1 Call
Expiration time
2024-02-15
Auto-renewable
Not enabled
Enable
Renewal
Integrate
Chat
Edition
Chat : Standard - Prepaid
Expiration time
2024-02-16
Auto-renewable
--
Renewal
Integrate
Add more products
Conference
Incorporate meetings for unlimited audience to collaborate together
Get started
RTC Engine
Integrate calling and interactive live streaming features with RTC SDK
Get started
In-game Voice Chat
Immersive In-Game Voice Chat:Enhance Your Gaming Experience
Get started


Are you sure you want to enable auto-renewal?
After enabling auto-renewal, if the account balance is sufficient, it will automatically renew monthly after expiration. Since some underlying services of Call are provided by Chat, this time will also enable auto-renewal for both Call and Chat.
Confirm activation
Cancel

功能与计费说明

TUICallKit 由实时音视频 TRTC 和即时通信 IM 提供底层技术支持，**您需要购买指定套餐后方可使用 TUICallKit，**其中免费体验版的开通方式可参见 开通 TUICallKit 体验版 。

下表为 TUICallKit 推荐套餐对应的功能和价格，您也可以自由组合 TRTC 包月套餐和 IM 版本以获得对应版本的 TUICallKit。TRTC 基础版配合 IM 专业版或旗舰版可使用TUICallKit 的1v1通话版；TRTC 尊享版和旗舰版配合 IM 专业版或旗舰版可使用TUICallKit 的群组通话版。

推荐套餐		体验版	1V1通话版	群组通话版 38万分钟	群组通话版 140万分钟
价格		0元/7天	立即选购	立即选购	立即选购
包含套餐版本		TRTC 体验版 + IM 任意版本	TRTC 基础版 + IM 专业版	TRTC 尊享版 + IM 旗舰版	TRTC 旗舰版 + IM 旗舰版
赠送资源 (超出后付费)	通话时长	-	110,000分钟/月	380,000分钟/月	1400,000分钟/月
	日活跃用户数（DAU）	根据 IM 版本而定	10,000 DAU	10,000 DAU	10,000 DAU
	峰值群组数	根据 IM 版本而定	100000个/月	100000个/月	100000个/月
TUICallKit 专属功能	1v1 视频、音频通话	✓	✓	✓	✓
	微信同款 UI 设计	✓	✓	✓	✓
	通话状态展示	✓	✓	✓	✓
	离线推送（应用未打开时，也能通过消息推送让用户收到音视频通话的响铃呼叫）	✓	✓	✓	✓
	通话悬浮窗（通话界面收起后，可以悬浮窗样式悬浮在应用界面之上）	✓	✓	✓	✓
	自定义呼叫铃声	✓	✓	✓	✓
	呼叫/接听/拒绝/挂断	✓	✓	✓	✓

版权所有：腾讯云计算（北京）有限责任公司

	视频通话切换语音通话	✓	✓	✓	✓
	群组通话	✓	-	✓	✓
	中途呼叫/加入三方通话	✓	-	✓	✓
	通话记录（看板 + API）	✓	-	✓	✓
	小程序通话加速（支持小程序端通话与小程序进房通话加速优化，保障通话效果）	✓	-	✓	✓
	多端登录通话（任一平台接通，自动停止其他端的接入请求）	✓	-	✓	✓
	同平台多端登录通话（支持同一平台，如多个 iOS 设备登录，来电后任一设备接通，自动停止其他设备的接入请求）	✓（仅配合 IM 旗舰版使用）	-	✓	✓
	AI 降噪（借助 AI 能力有效过滤背景噪声，还原通话效果）	✓	-	✓	✓
	弱网通话卡顿优化（针对户外等弱网环境下优化卡顿率，秒开速度更快）	✓	-	✓	✓
即时通信 IM 功能		根据 IM 版本而定 详见 IM 功能	详见 IM 专业版功能	详见 IM 旗舰版功能	详见 IM 旗舰版功能
实时音视频 TRTC 功能		详见 TRTC 体验版功能	详见 TRTC 基础版功能	详见 TRTC 尊享版功能	详见 TRTC 旗舰版功能

超出赠送资源部分后付费	通话时长后付费价格详见音视频时长计费说明，DAU 与峰值群组数后付费价格详见 IM 计费说明
支持平台	iOS、Android、Web、**微信小程序（仅体验版和群组通话版支持）**、uni-app、Flutter

说明：

包含套餐版本：TUICallKit 由 TRTC 和 IM 联合提供，使用 TUICallKit 需同时购买 TRTC 包月套餐和 IM 基础套餐包，不同版本套餐提供的 TUICallKit 功能有所不同。

DAU：单个用户当日登录即时通信 IM 计为1个 DAU，同一用户重复登录时，DAU 不累加。

峰值群组数：单个应用（SDKAppID）下所有用户创建和加入的群组数的总和，以自然月中最高峰值进行结算。

快速接入（TUICallKit）

Android

最近更新时间：2024-04-03 17:23:11

本文将介绍如何用最短的时间完成 TUICallKit 组件的接入，跟随本文档，您可以在 10 分钟内完成接入工作，并最终得到一个包含完备 UI 界面的音视频通话功能。

环境准备

Android 5.0（SDK API Level 21）及以上版本。
Gradle 4.2.1 及以上的版本。
Android 5.0 及以上的手机设备。

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：下载并导入组件

在 Github 中克隆/下载代码，然后拷贝 Android 目录下的 tuicallkit-kt 子目录到您当前工程中的 app 同一级目录中，如下图：

版权所有：腾讯云计算（北京）有限责任公司

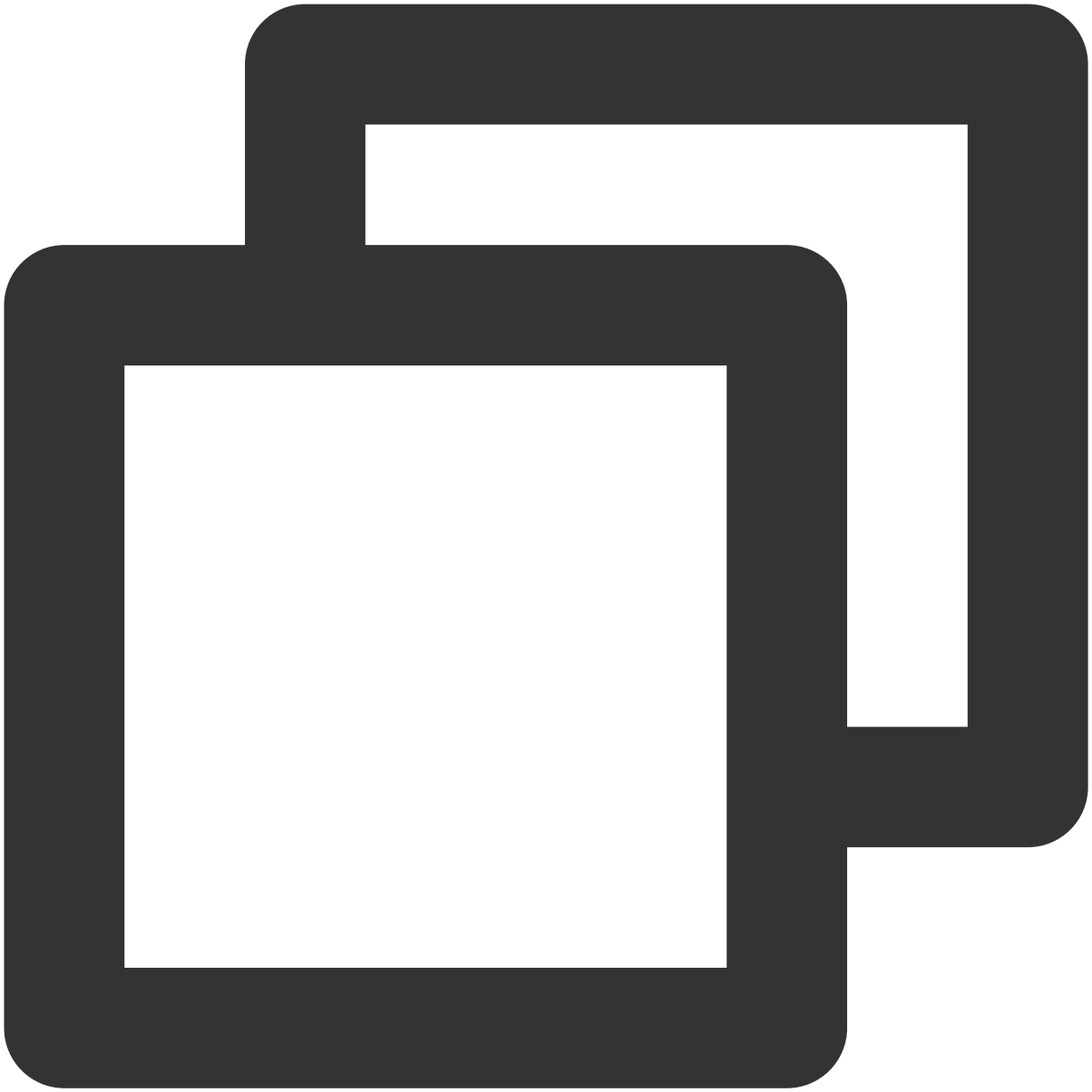


步骤三：工程配置

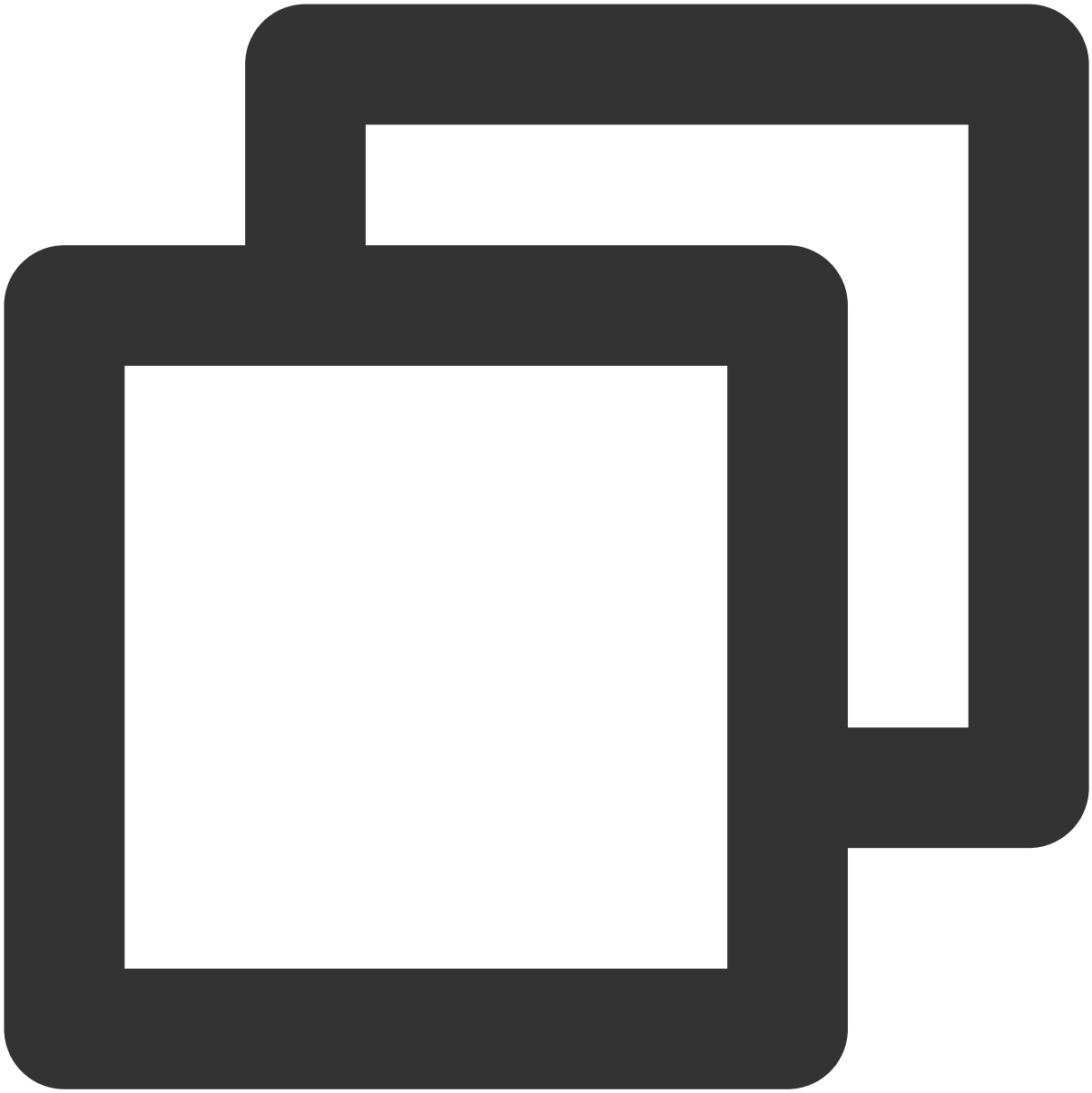
1. 在工程根目录下找到 `settings.gradle（或settings.gradle.kts）` 文件，在其中增加如下代码，它的作用是将 步骤二 中下载的组件导入到您当前的项目中：

settings.gradle

setting.gradle.kts



```
include ':tuicallkit-kt'
```

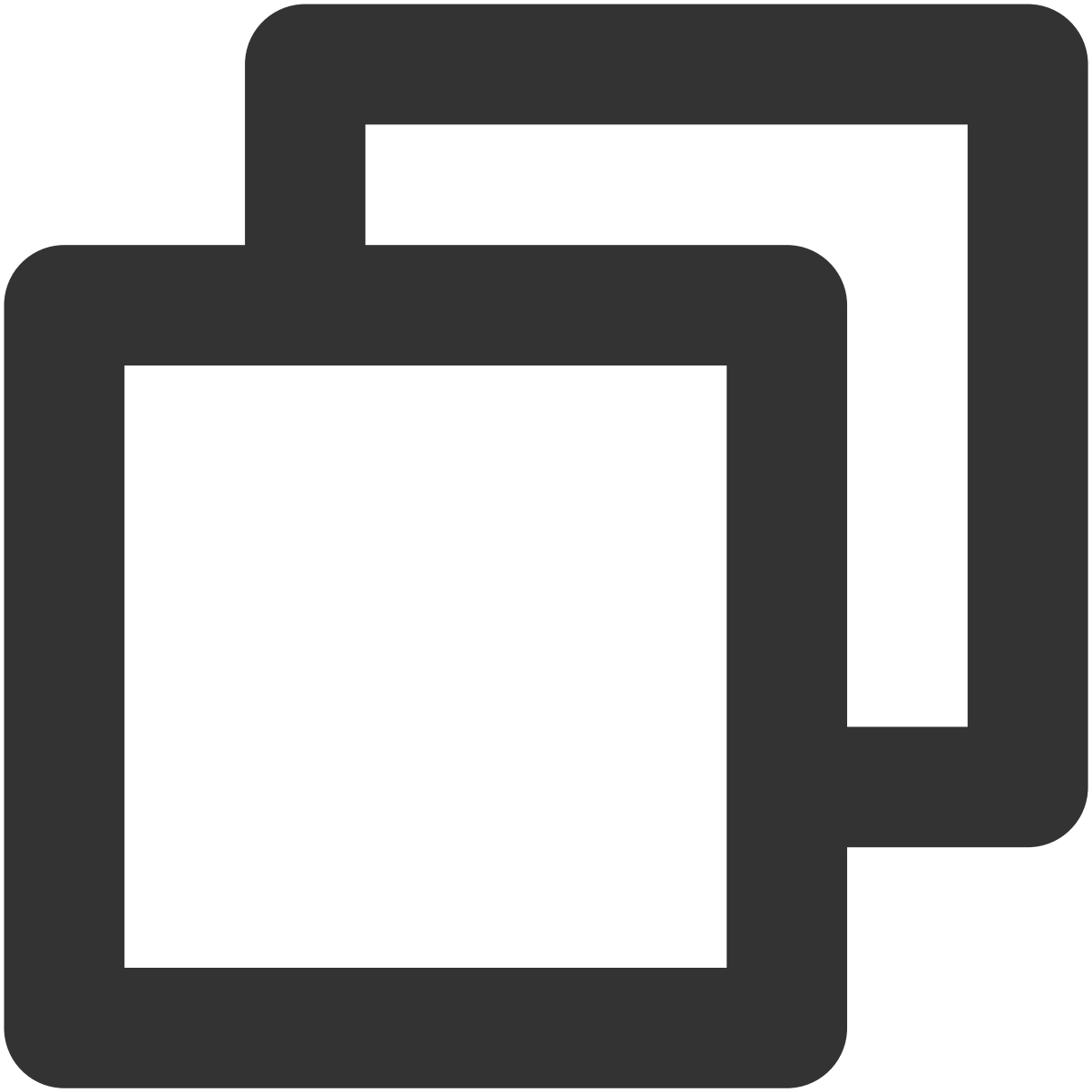


```
include(":tuicallkit-kt")
```

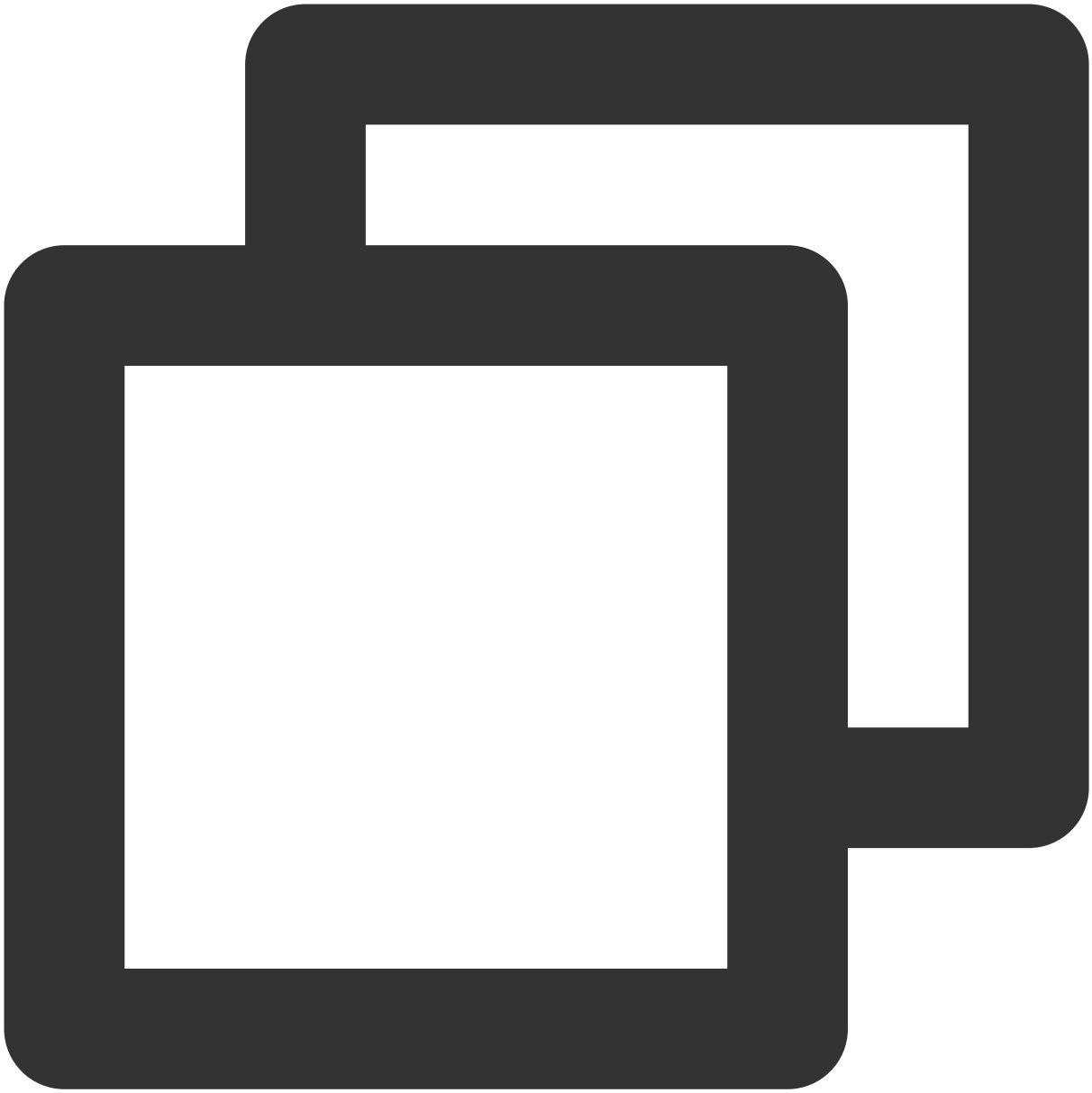
2. 在 app 目录下找到 `build.gradle(或build.gradle.kts)` 文件，增加如下代码，它的作用是声明当前 app 对新加入的组件的依赖：

build.gradle

build.gradle.kts



```
api project(':tuicallkit-kt')
```

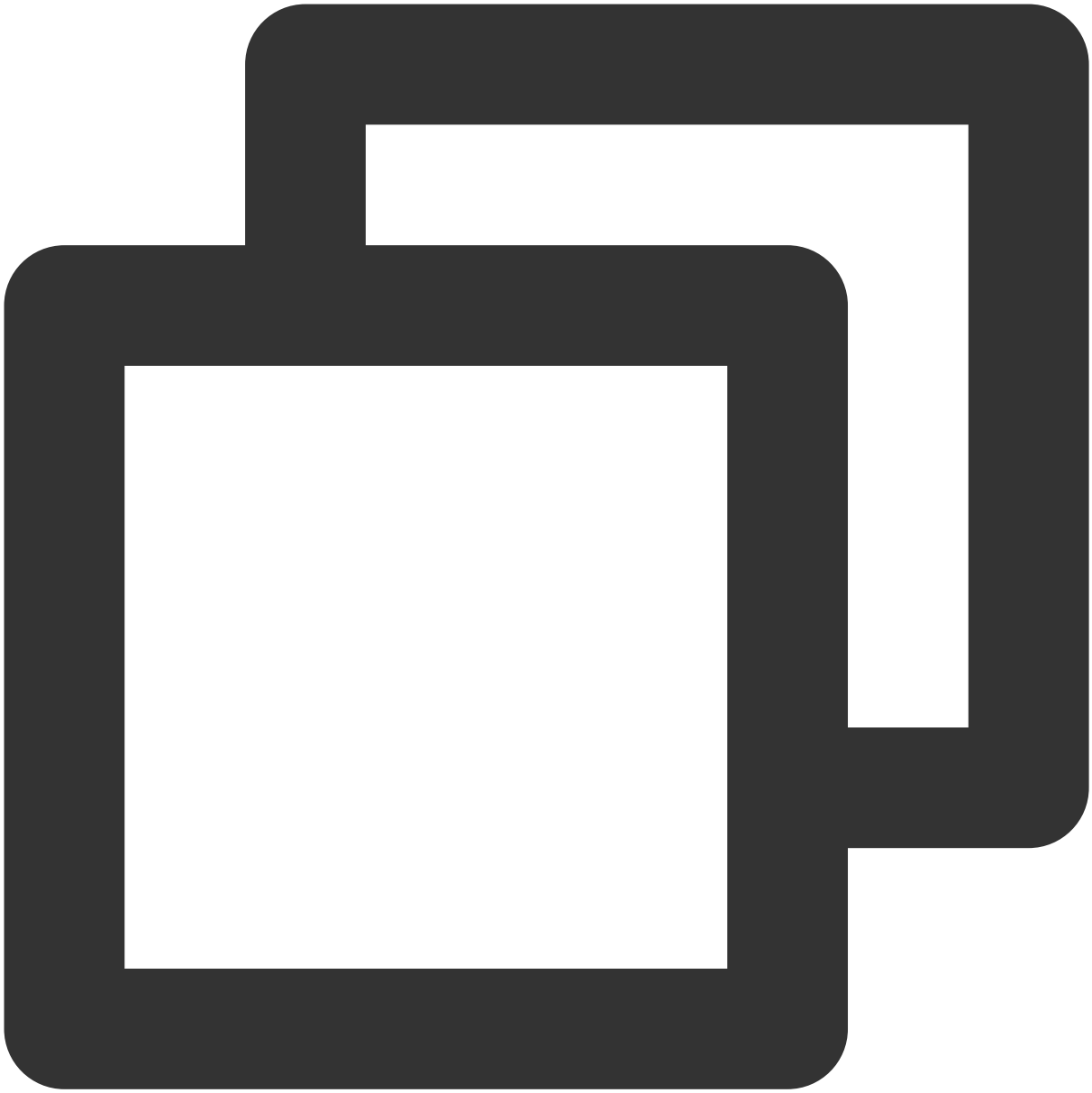


```
api(project(":tuicallkit-kt"))
```

说明

TUICallKit 工程内部已经默认依赖： `TRTC SDK` 、 `IM SDK` 、 `tuicallengine` 以及公共库 `tuicore` ，不需要开发者单独配置。如需进行版本升级，则修改 `tuicallkit-kt/build.gradle` 文件中的版本号即可。

3. 由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单，因此需要您在 `proguard-rules.pro` 文件中添加如下代码：



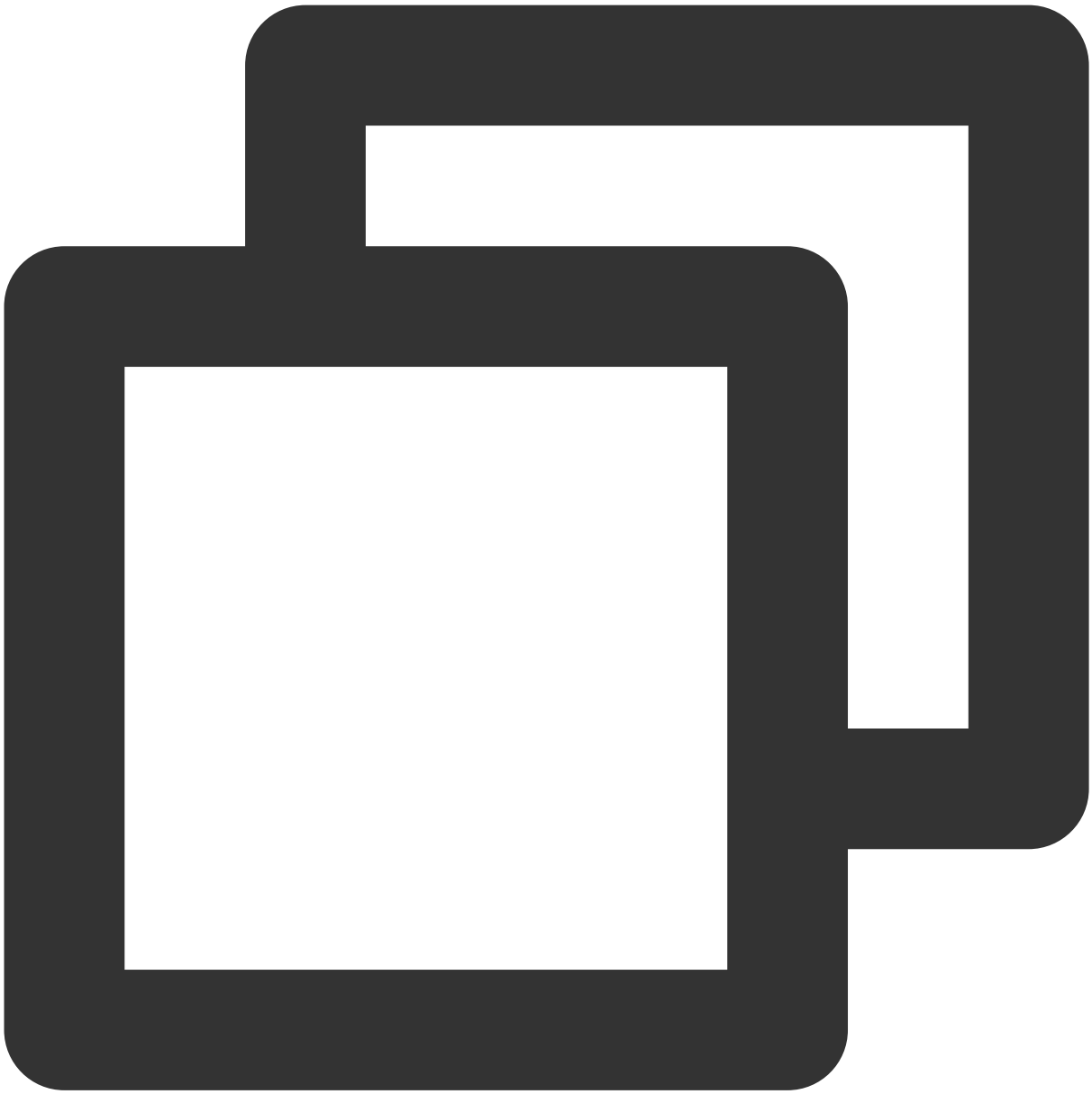
```
-keep class com.tencent.** { *; }
```

注意

TUICallKit 会在内部帮助您动态申请相机、麦克风、读取存储权限等。如果因为您的业务问题需要删减，可以修改组件的 `tuicallkit-kt/src/main/AndroidManifest.xml` 文件。

步骤四：登录 TUI 组件

在您的项目中添加如下代码，它的作用是通过调用 TUICore 中的相关接口完成 TUI 组件的登录。这个步骤异常关键，因为只有在登录成功后才能正常使用 TUICallKit 的各项功能，故请您耐心检查相关参数是否配置正确：



```
TUILogin.login(this,
    1400000001,     // 请替换为步骤一取到的 SDKAppID
    "denny",        // 请替换为您的 UserID
    "xxxxxxxxxxx",  // 您可以在控制台中计算一个 UserSig 并填在这个位置
    object : TUICallback() {
      override fun onSuccess() {
      }
```

```
        override fun onError(errorCode: Int, errorMessage: String) {
        }
    })
}
```

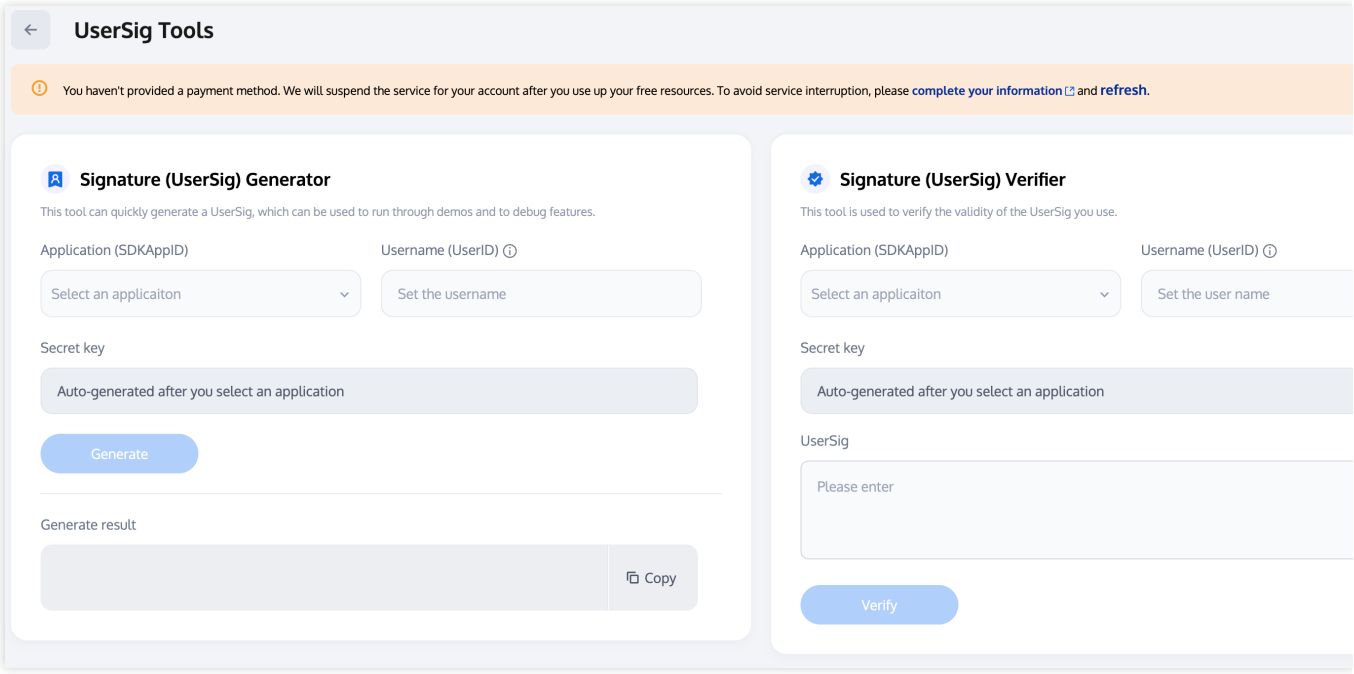
参数说明：

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

SDKAppID：在步骤一中的最后一步中您已经获取到，这里不再赘述。

UserID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

UserSig：使用 `SDKSecretKey` 对 `SDKAppID` 、 `UserID` 等信息进行加密，就可以得到 `UserSig` ，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以访问 实时音视频 TRTC 控制台 -> UserSig Tools，得到一个临时可用的 UserSig。



更多信息请参见 如何计算及使用 UserSig。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下:

SDKAppID 设置错误。

UserSig 被错配成了加密密钥（Secretkey），userSig 是用 SecretKey 把 SDKAppID、UserID 以及过期时间等信息加密得来的，而不是直接把 SecretKey 配置成 UserSig。

UserSig 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

Github 中的示例代码使用了 genTestUserSig 函数在本地计算 UserSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SecretKey，所以我们强烈建

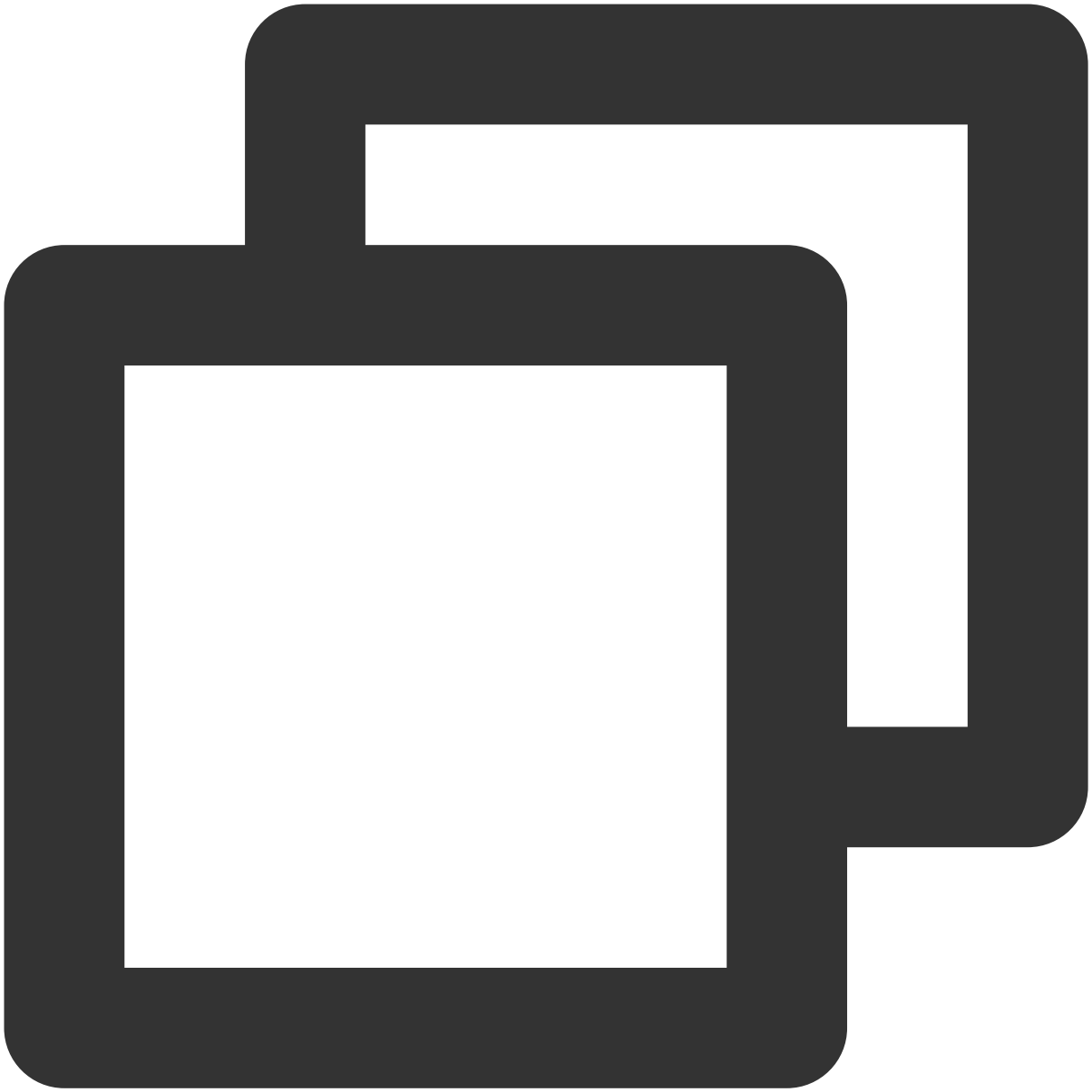
议您将 UserSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUICallKit 组件时向您的服务器请求实时计算出的 UserSig。

步骤五：拨打您的第一通电话

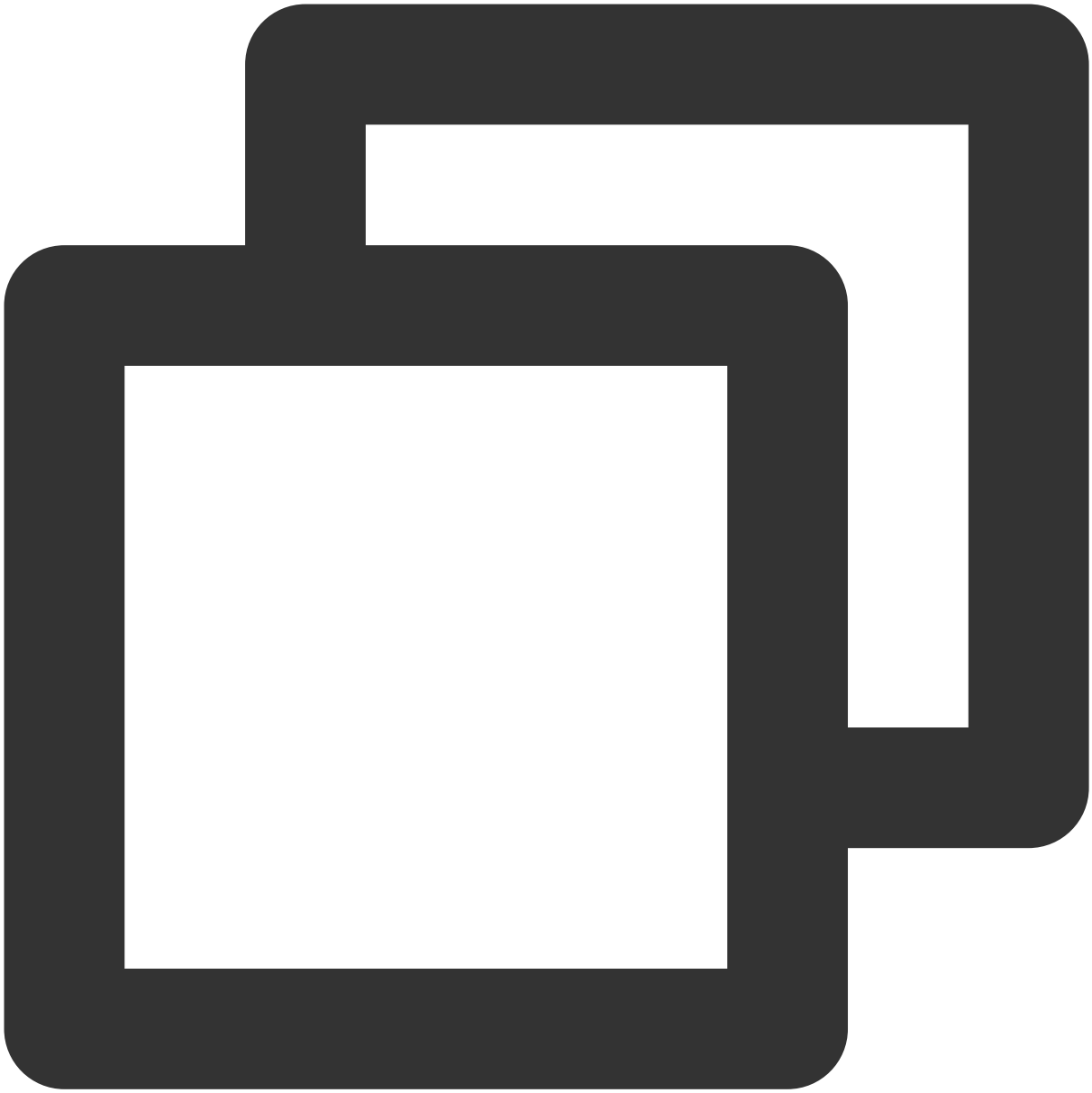
主叫方与被叫方登录成功后，主叫方通过调用 TUICallKit 的 call 方法并指定通话类型和被叫方的 userId，就可以发起语音或者视频通话，被叫方此时就可接受到来电邀请。

Kotlin

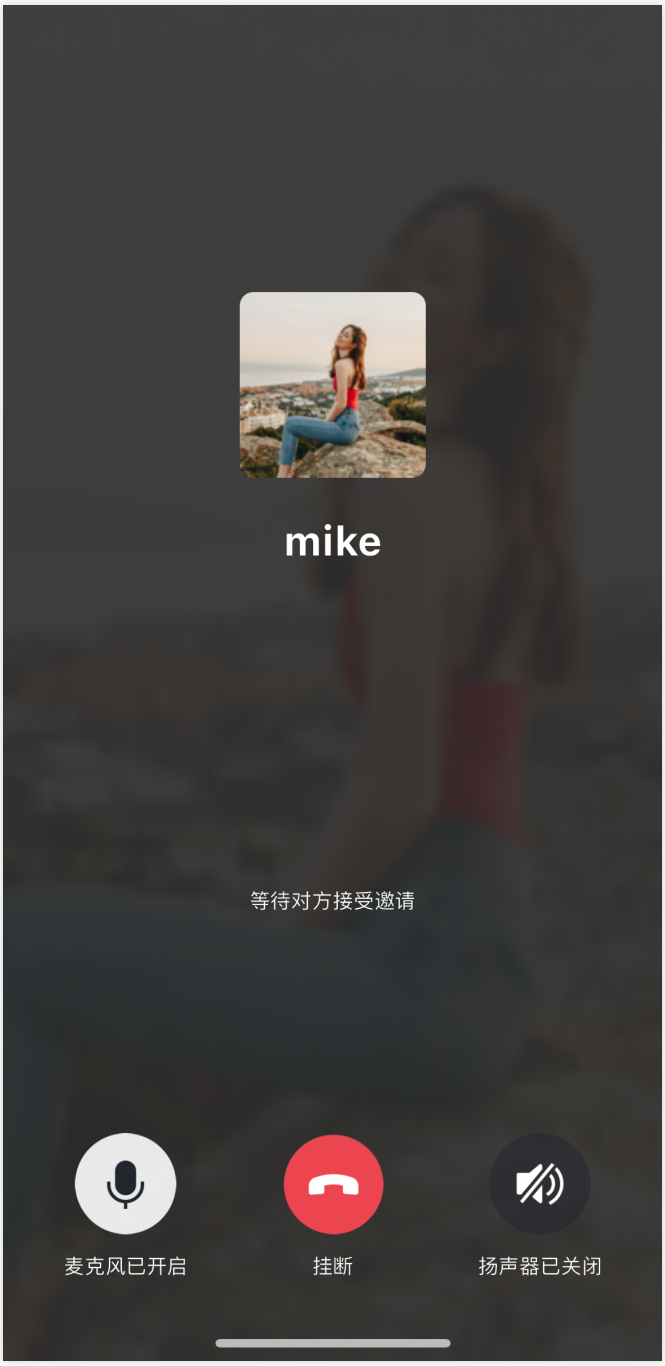
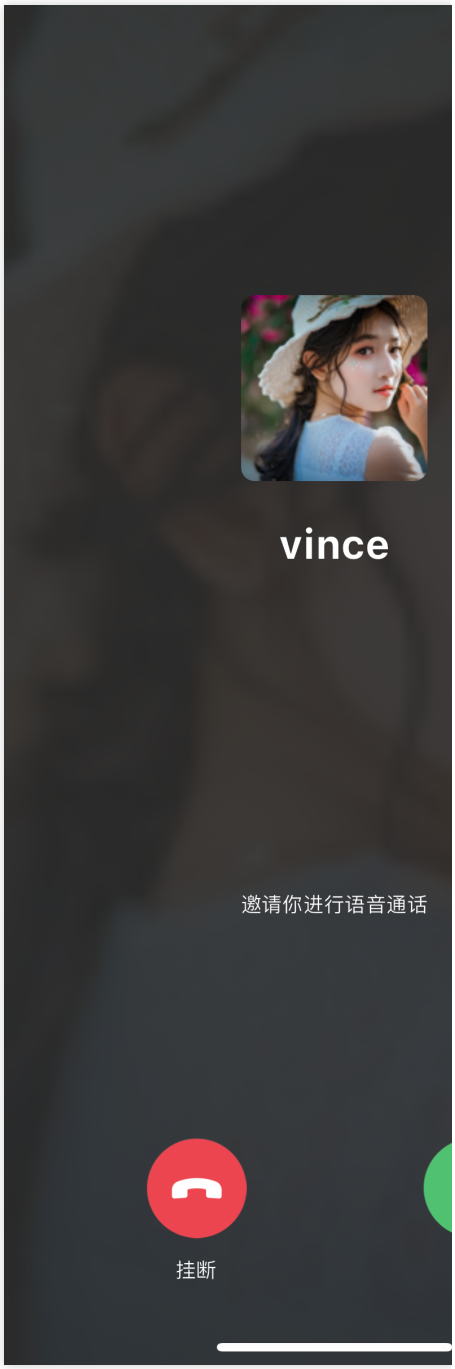
Java



```
// 发起1对1视频通话(假设 UserID 为 mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video)
```



```
// 发起1对1视频通话(假设 UserID 为 mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video);
```

	
主叫方	被叫方

更多特性

界面定制
离线推送
群组通话
悬浮窗

自定义铃声
监听通话状态
云端录制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

iOS

最近更新时间：2024-04-03 17:23:11

本文将介绍如何用最短的时间完成 TUICallKit 组件的接入，跟随本文档，您将在一个小时内完成如下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

环境准备

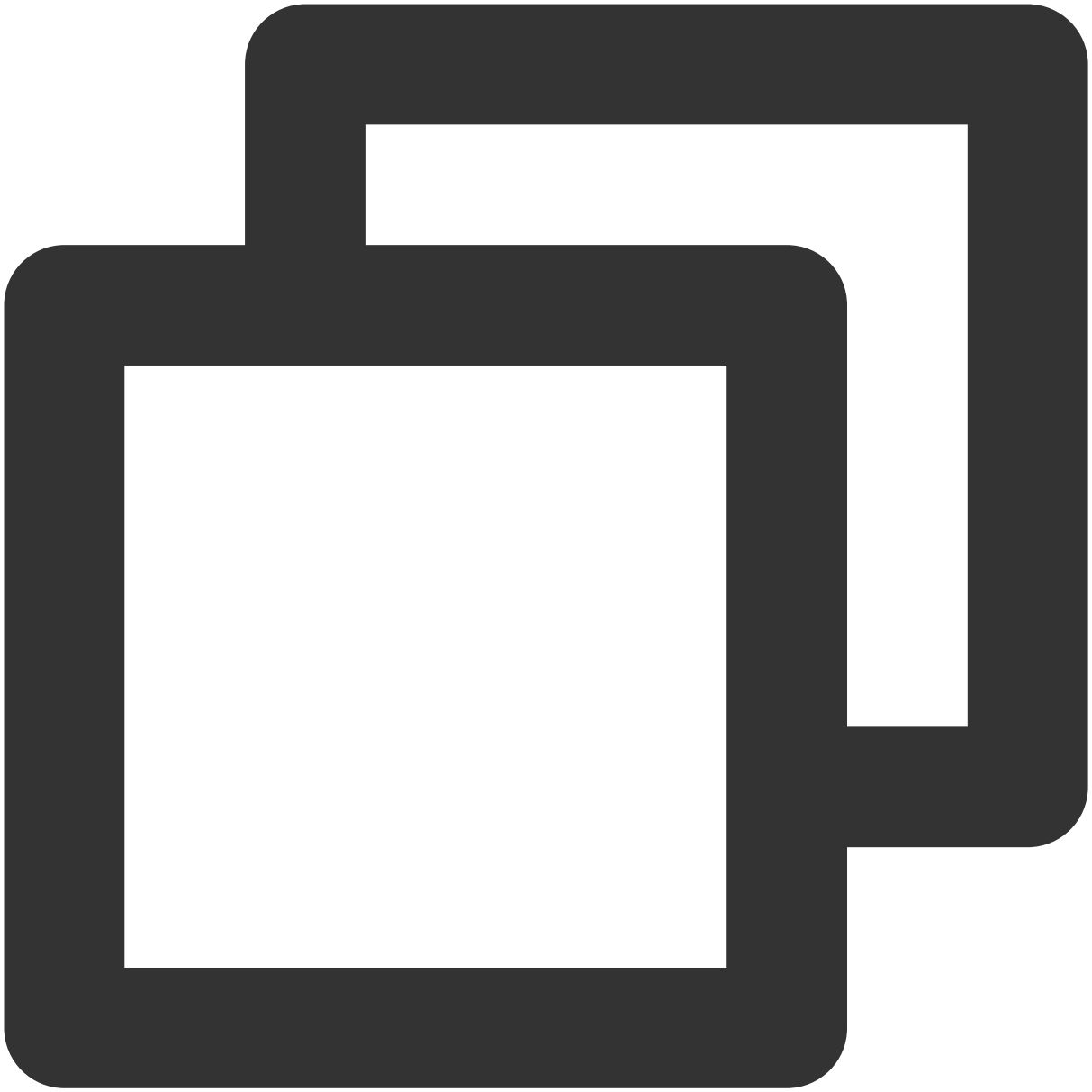
Xcode 13 及以上
iOS 12.0 及以上

步骤一：开通服务

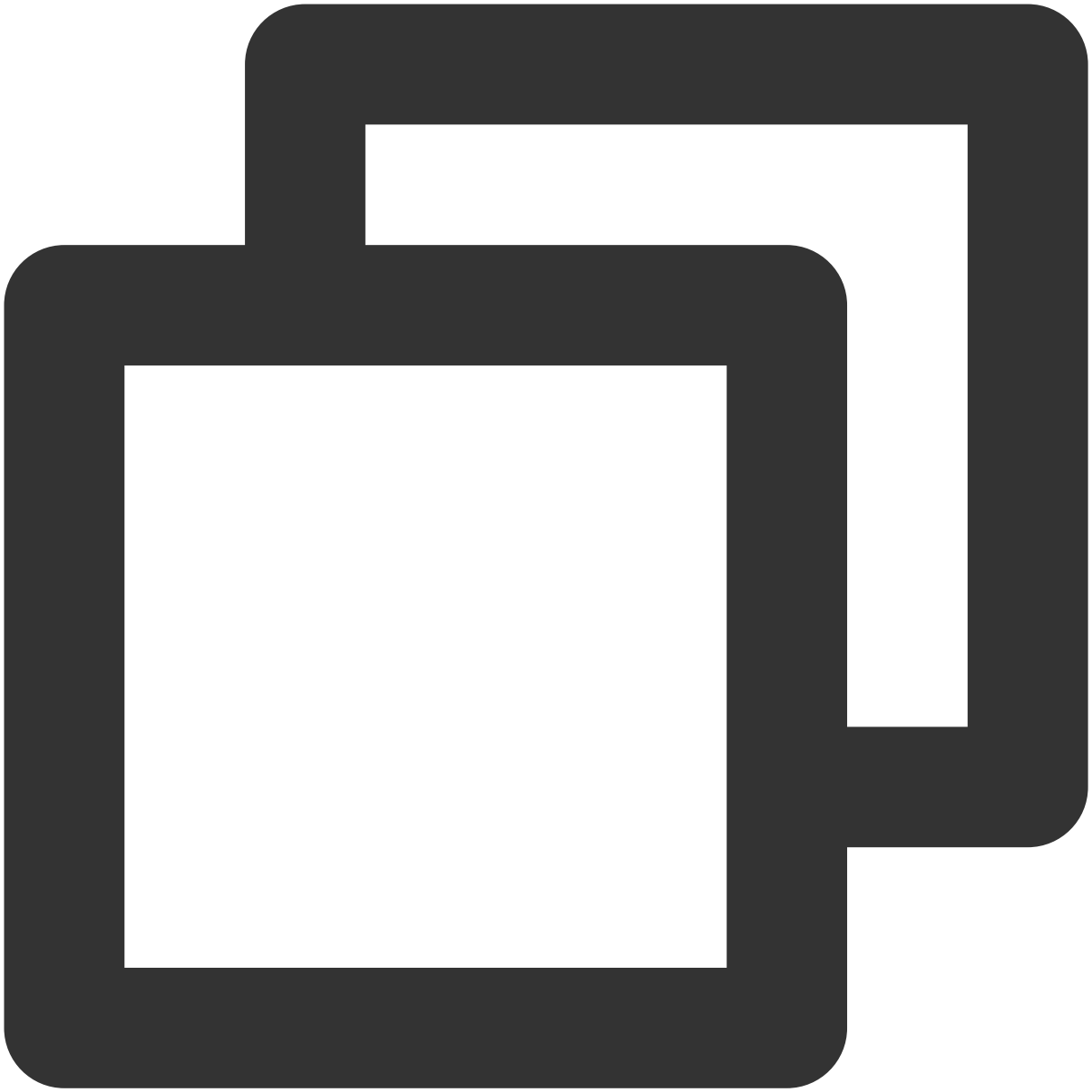
在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：导入组件

推荐使用 CocoaPods 导入组件，具体步骤如下：
1. 在您的 `Podfile` 文件中添加以下依赖。
Swift
Objective-C

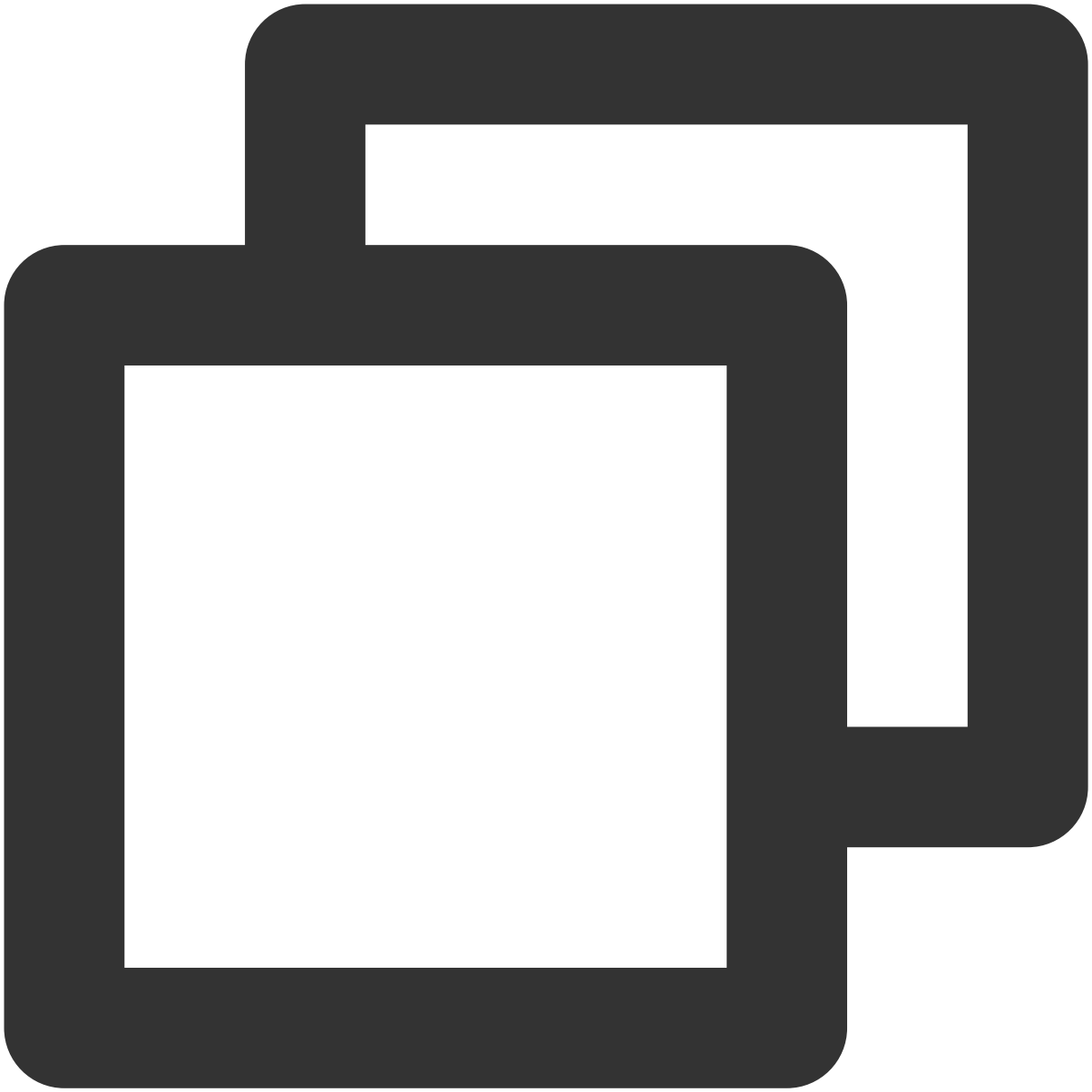


```
pod 'TUICallKit_Swift'
```



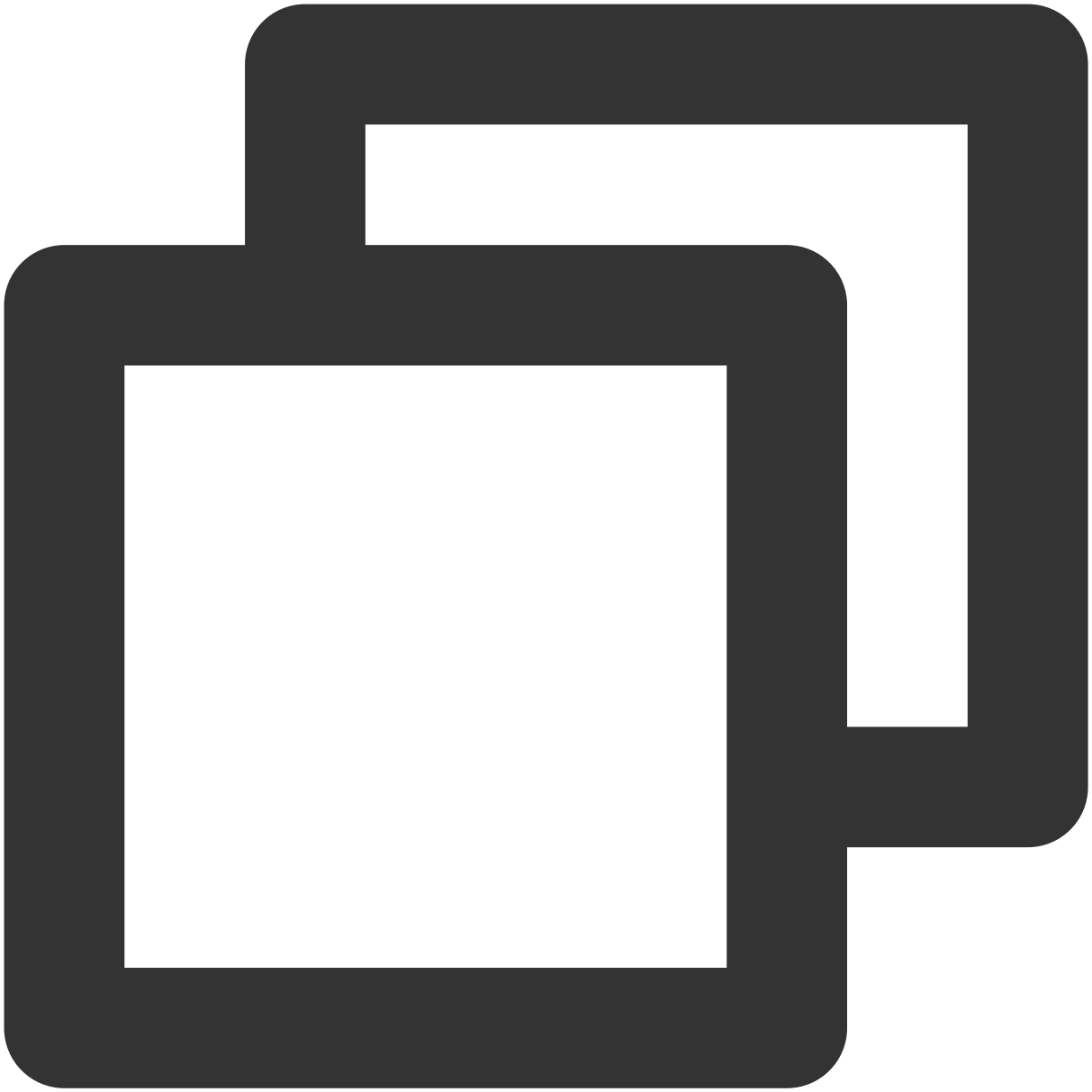
```
pod 'TUICallKit'
```

2. 执行以下命令，安装组件。



```
pod install
```

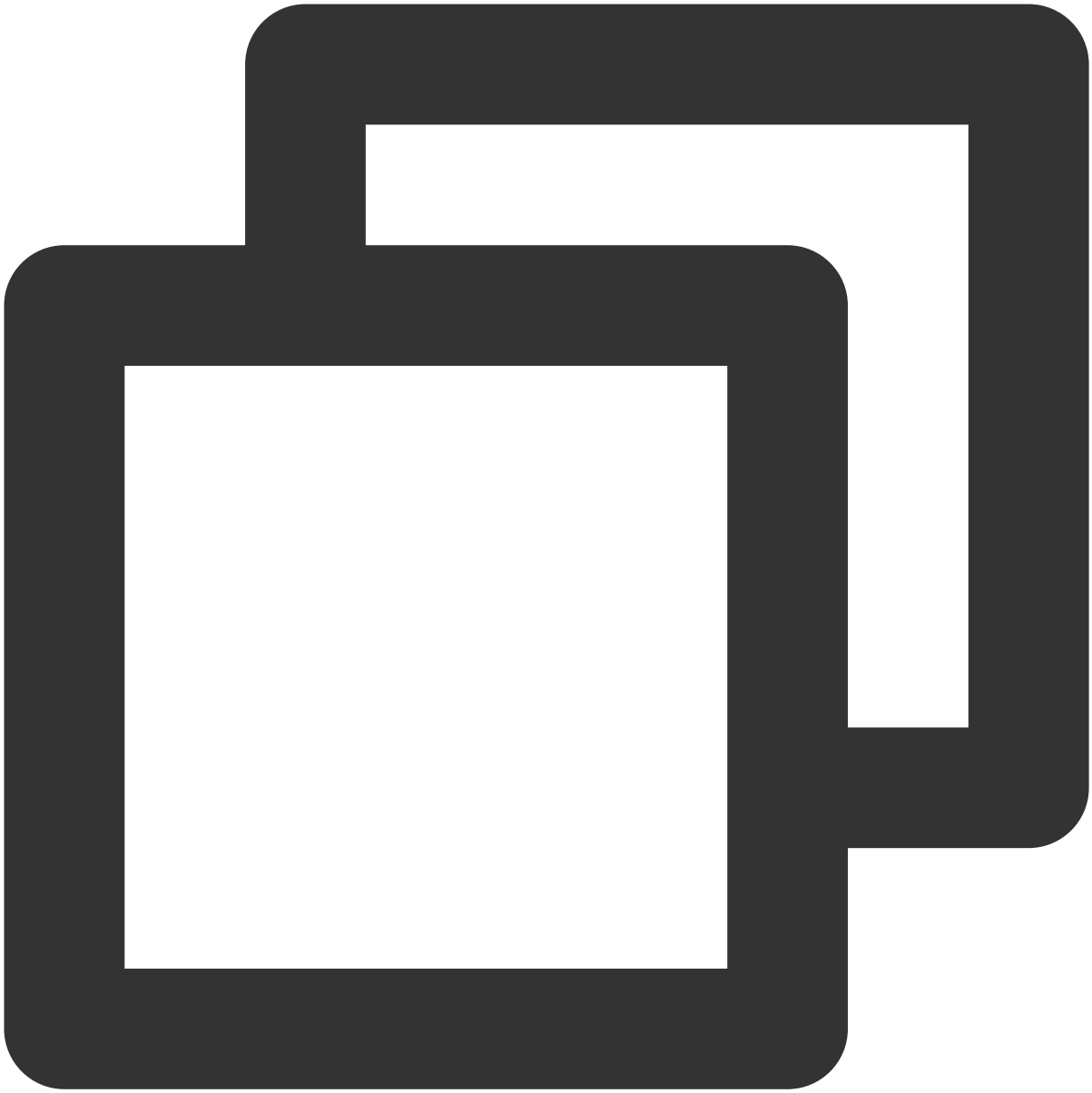
如果无法安装 TUICallKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。



```
pod repo update
```

之后执行以下命令，更新组件库的 Pod 版本。

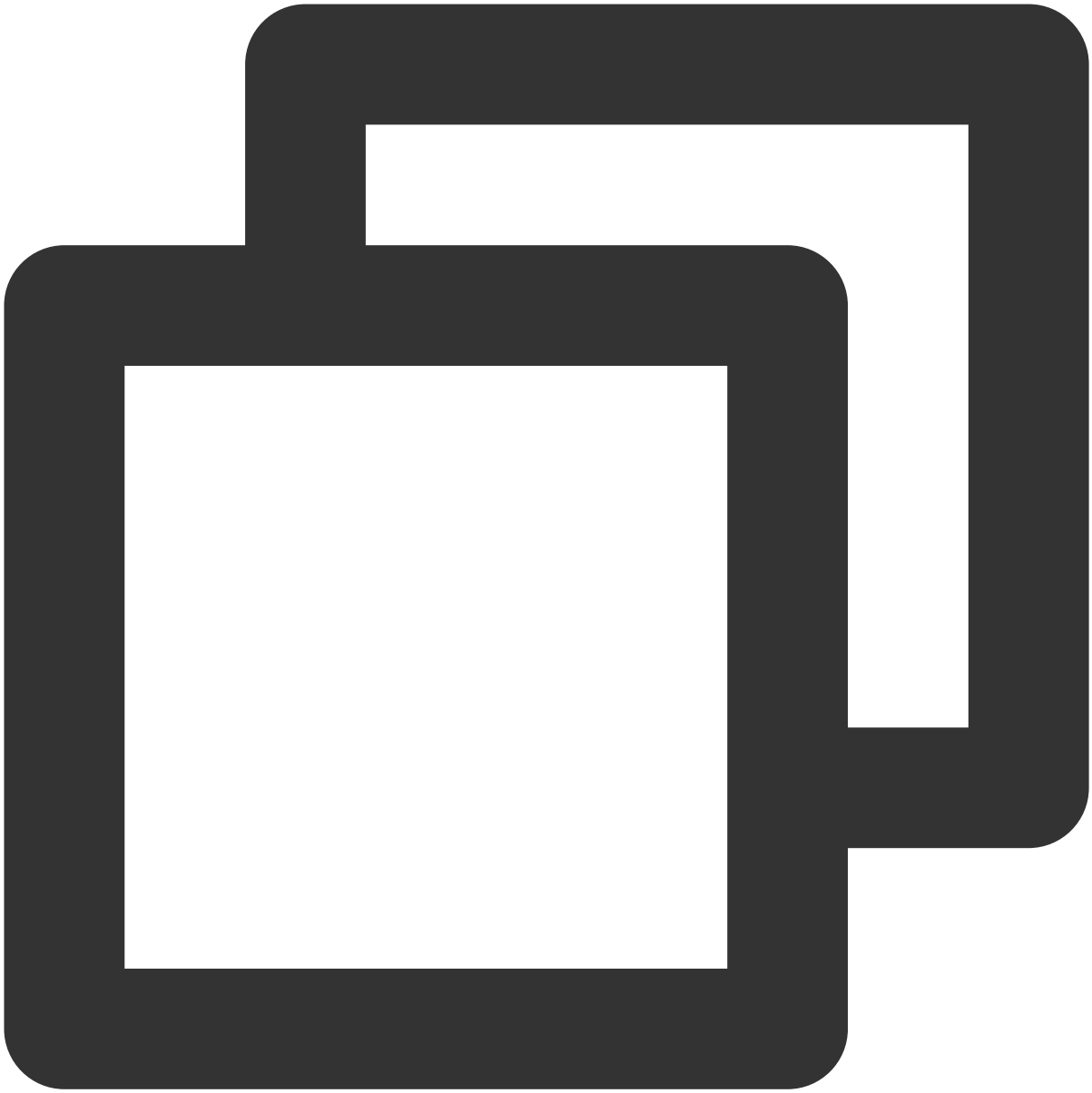
版权所有：腾讯云计算（北京）有限责任公司



```
pod update
```

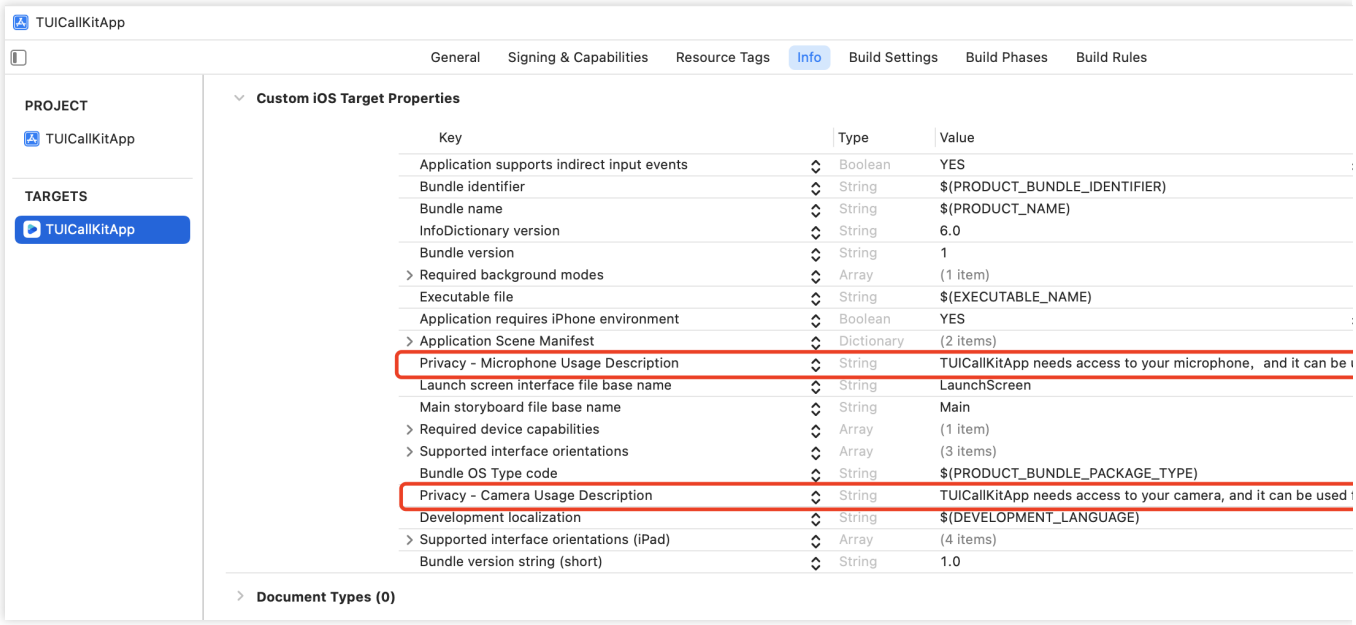
步骤三：完成工程配置

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



```
<key>NSCameraUsageDescription</key>
<string>CallingApp 需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>CallingApp 需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
```

版权所有：腾讯云计算（北京）有限责任公司

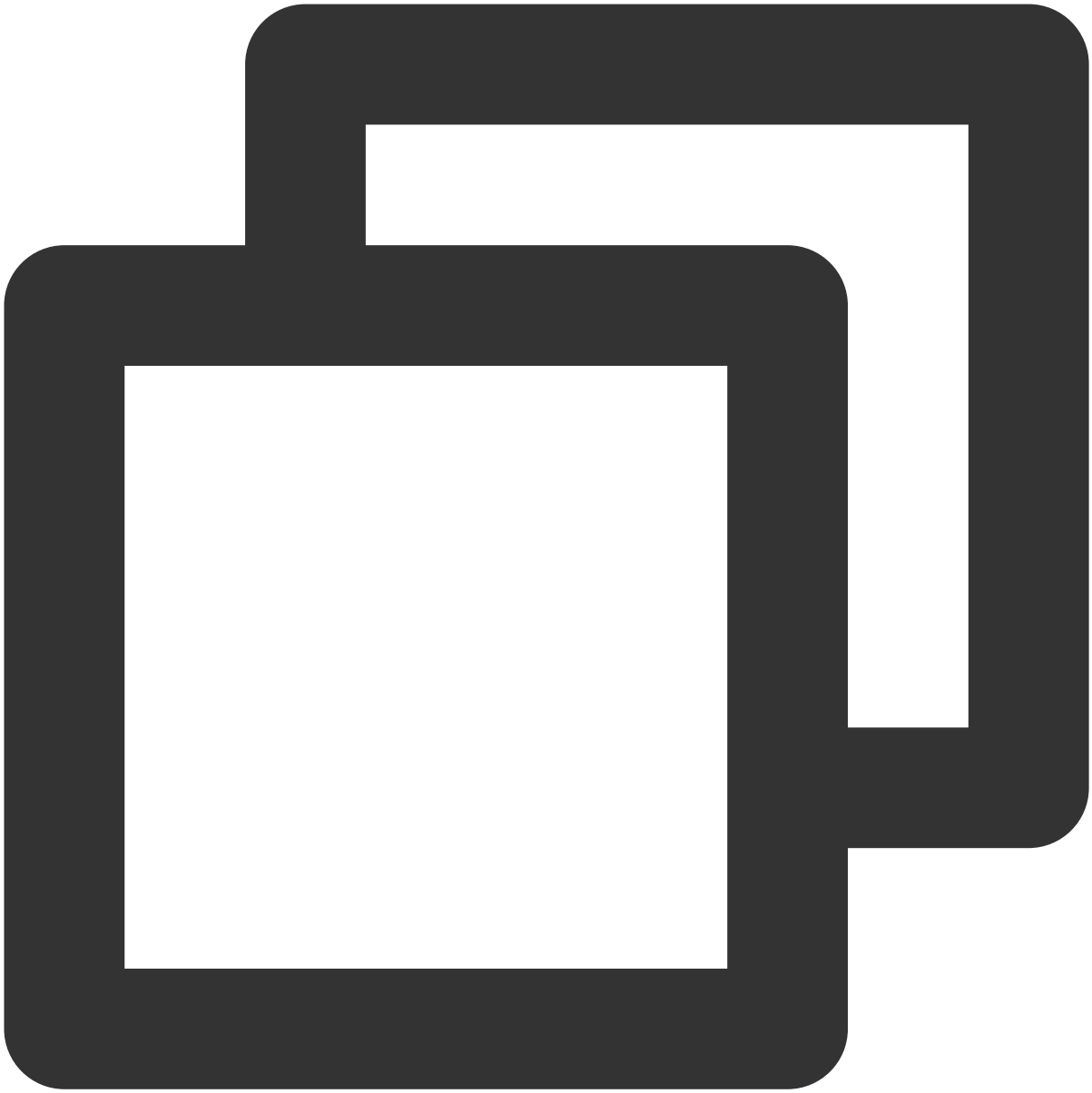


步骤四：登录 TUI 组件

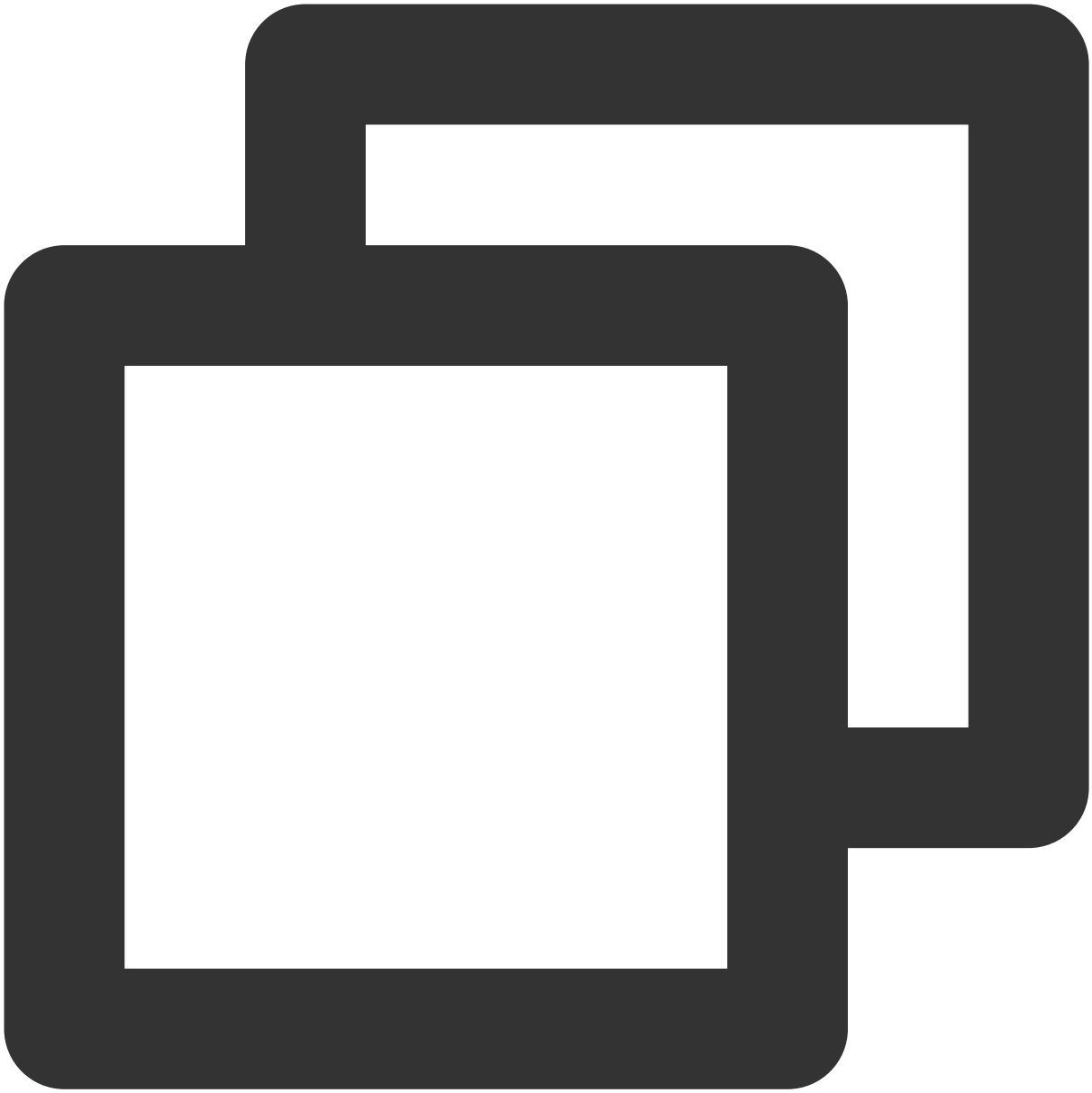
在您的项目中添加如下代码，它的作用是通过调用 TUICore 中的相关接口完成 TUI 组件的登录。这个步骤异常关键，因为只有在登录成功后才能正常使用 TUICallKit 的各项功能，故请您耐心检查相关参数是否配置正确：

Swift

Objective-C



```
import TUICore

TUILogin.login(1400000001,                          // 请替换为步骤一取到的 SDKAppID
            userID: "denny",                        // 请替换为您的 UserID
            userSig: "xxxxxxxxxxx") {               // 您可以在控制台中计算一个 UserSig 并填在这
    print("login success")
} fail: { (code, message) in
    print("login failed, code: \\(code), error: \\(message ?? "nil")")
}
```



```
#import <TUICore/TUILogin.h>

[TUILogin login:1400000001            // 请替换为步骤一取到的 SDKAppID
         userID:@"denny"              // 请替换为您的 UserID
        userSig:@"xxxxxxxxxxx"        // 您可以在控制台中计算一个 UserSig 并填在这个位置
           succ:^{
    NSLog(@"login success");
} fail:^(int code, NSString *msg) {
    NSLog(@"login failed, code: %d, error: %@", code, msg);
}
```

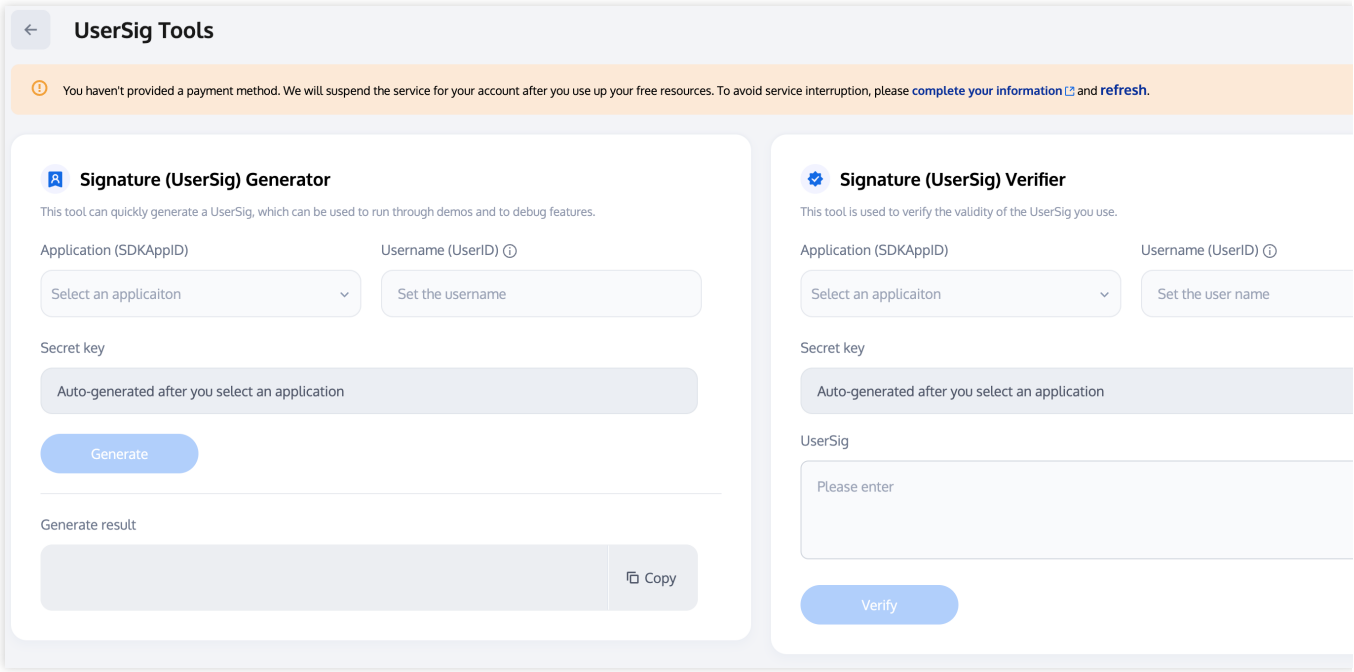
参数说明：

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

SDKAppID：在步骤一中的最后一步中您已经获取到，这里不再赘述。

userID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

userSig：使用 `SDKSecretKey` 对 `SDKAppID` 、 `userID` 等信息进行加密，就可以得到 `userSig` ，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以访问 实时音视频 TRTC 控制台 -> UserSig Tools，得到一个临时可用的 userSig。



更多信息请请参见 如何计算及使用 UserSig。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下:

SDKAppID 设置错误。

userSig 被错配成了加密密钥（Secretkey），userSig 是用 SecretKey 把 SDKAppID、userID 以及过期时间等信息加密得来的，而不是直接把 SecretKey 配置成 userSig。

userSig 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 userID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 userID。

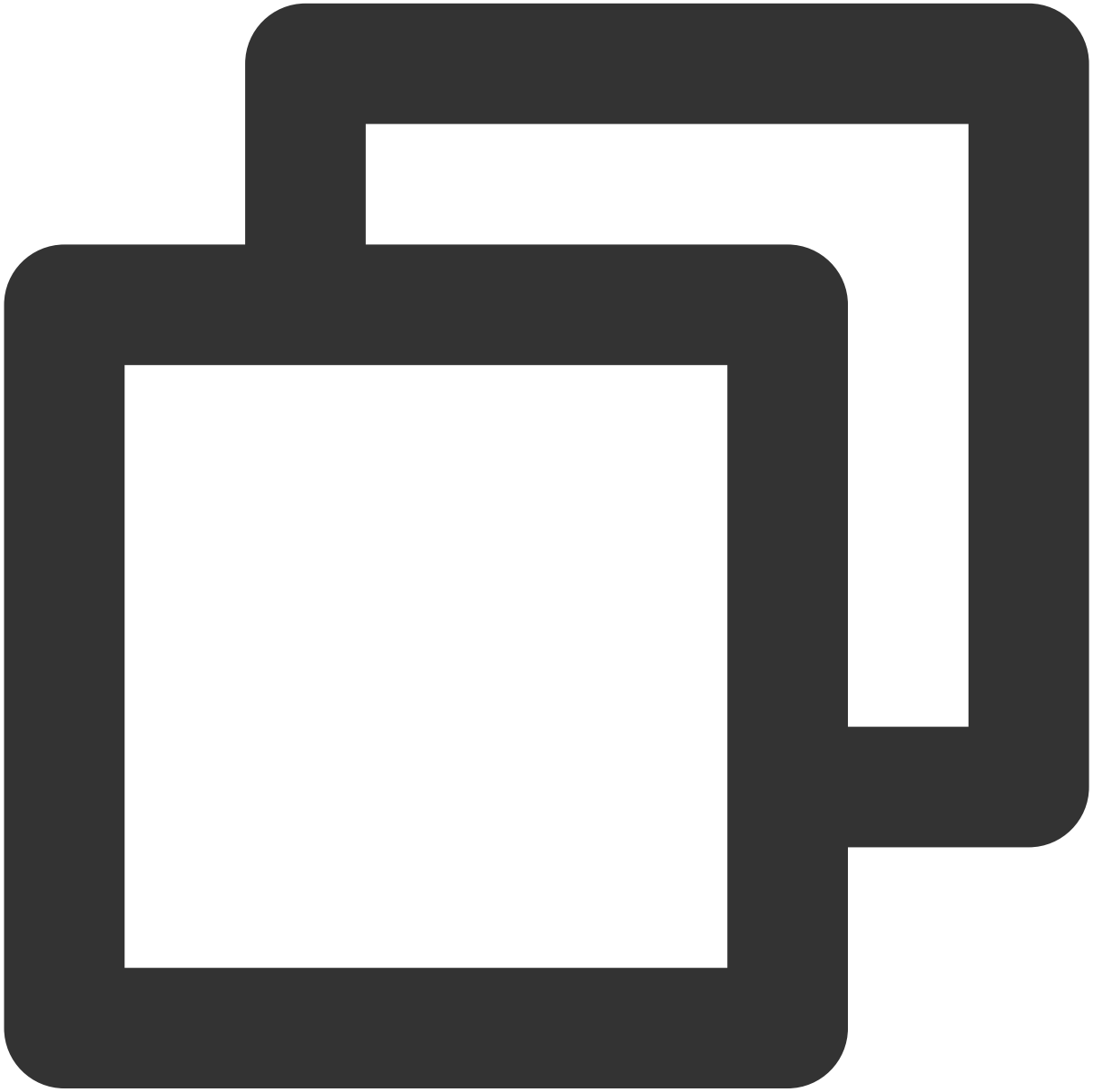
Github 中的示例代码使用了 genTestUserSig 函数在本地计算 userSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SecretKey，所以我们强烈建议您将 userSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUICallKit 组件时向您的服务器请求实时计算出的 userSig。

步骤五：拨打您的第一通电话

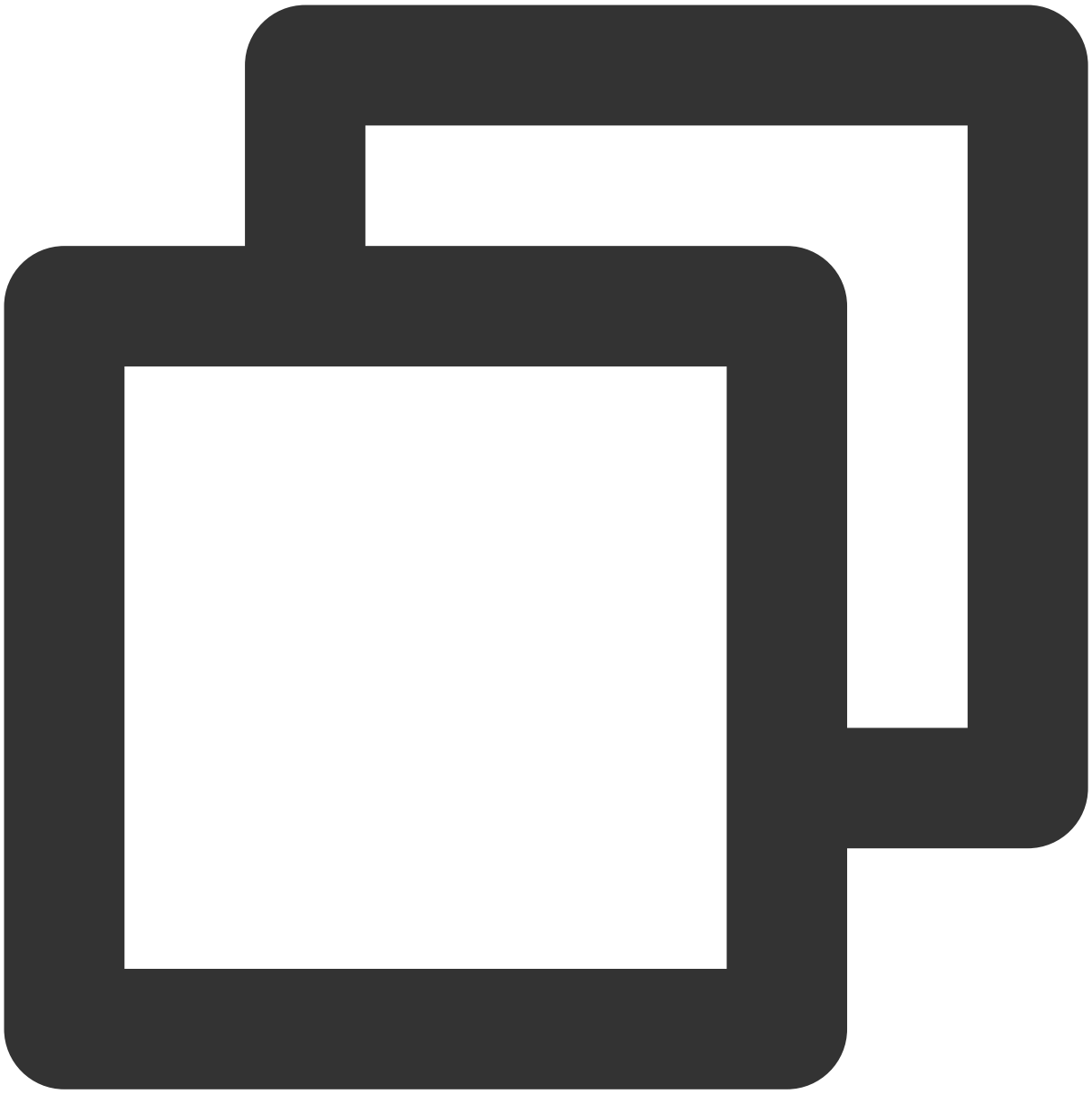
通过调用 TUICallKit 的 call 函数并指定通话类型和被叫方的 userId，就可以发起语音或者视频通话。

Swift

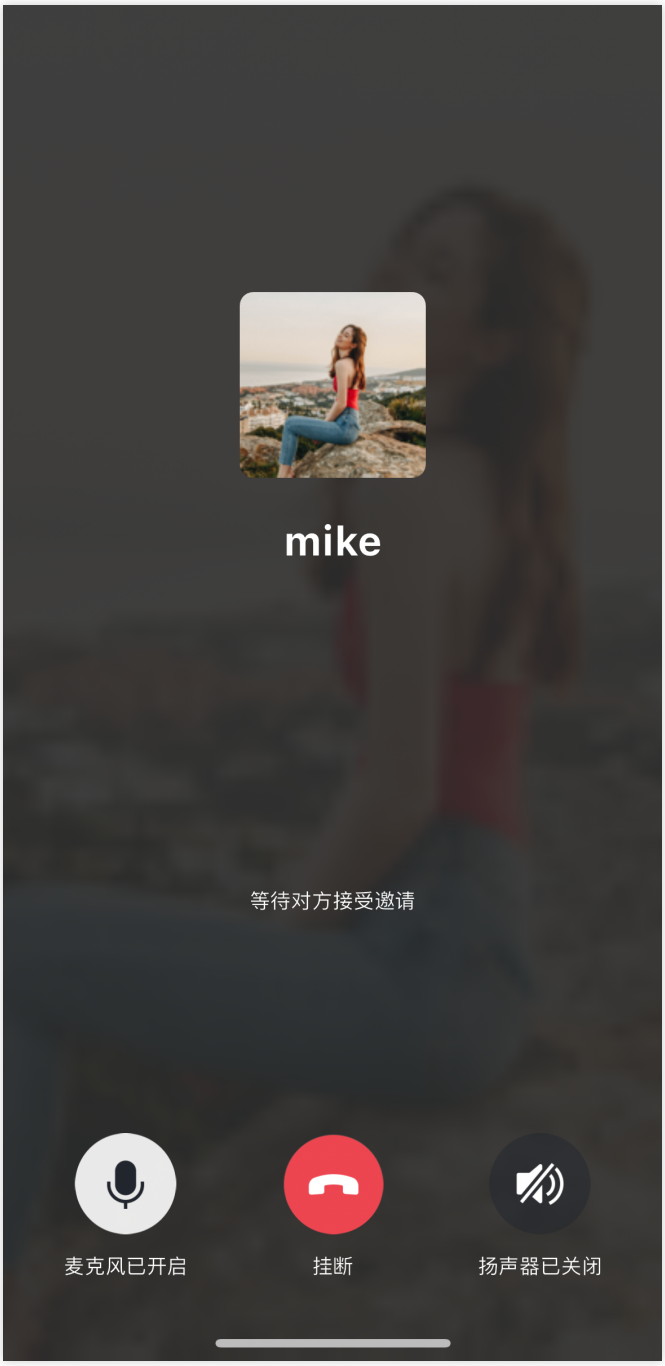
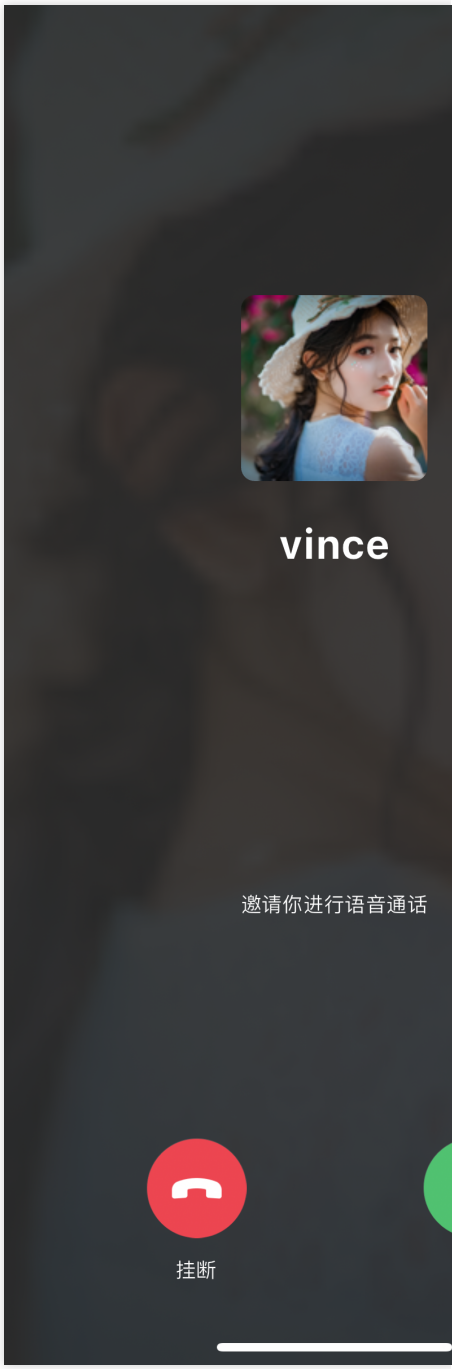
Objective-C



```
import TUICallKit

// 发起1对1视频通话(假设 userId 为 mike)
TUICallKit.createInstance().call(userId: "mike", callMediaType: .video)
```



```
#import <TUICallKit/TUICallKit.h>

// 发起1对1视频通话(假设 userId 为 mike)
[[TUICallKit createInstance] call:@"mike" callMediaType:TUICallMediaTypeVideo];
```

	
主叫方	被叫方

更多特性

界面定制
离线推送
群组通话
悬浮窗

自定义铃声
监听通话状态
云端录制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

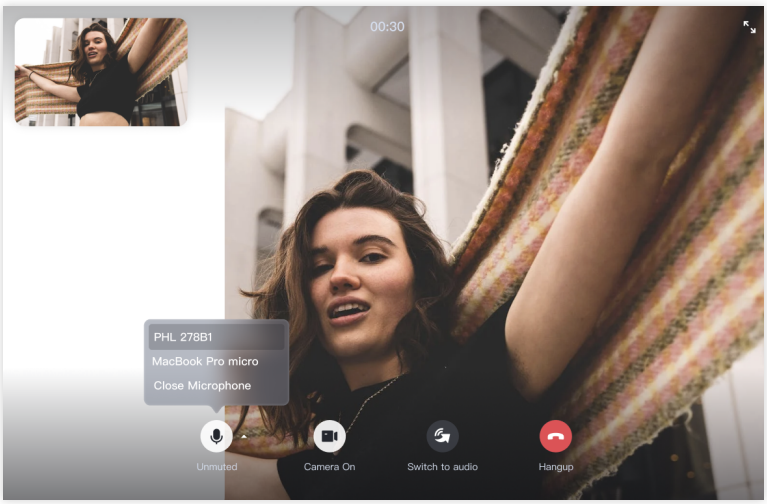
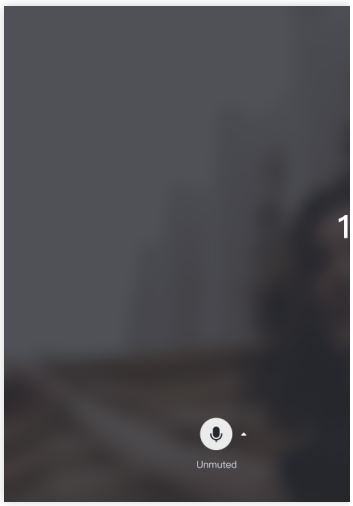
Web&H5 (React)

最近更新时间：2024-04-03 17:23:11

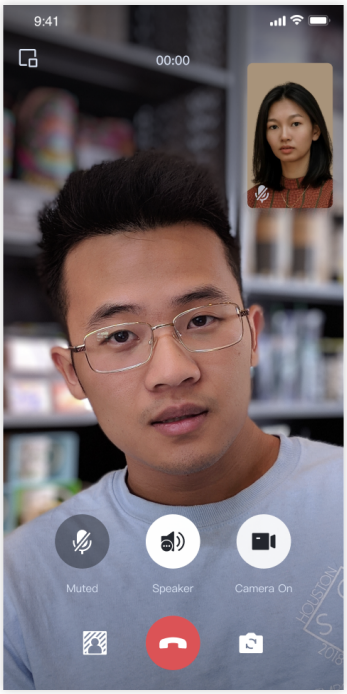
本文将介绍如何快速完成 TUICallKit 组件的接入，您将在10分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

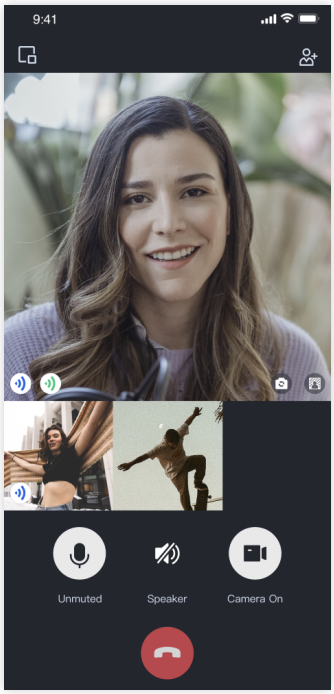
桌面端

移动端

视频通话	语音通话
	

单人通话	群组通话





TUICallKit Demo 体验

如果您想要直接体验音视频通话，进入 Demo 体验页面。
如果您想要直接跑通一个新工程，请直接阅读 Web Demo 快速跑通。

环境准备

React ≥ v18.0。
Node.js 版本：Node.js ≥ 12.13.0（推荐使用官方 LTS 版本，npm 版本请与 node 版本匹配）。
Modern browser, supporting WebRTC APIs.

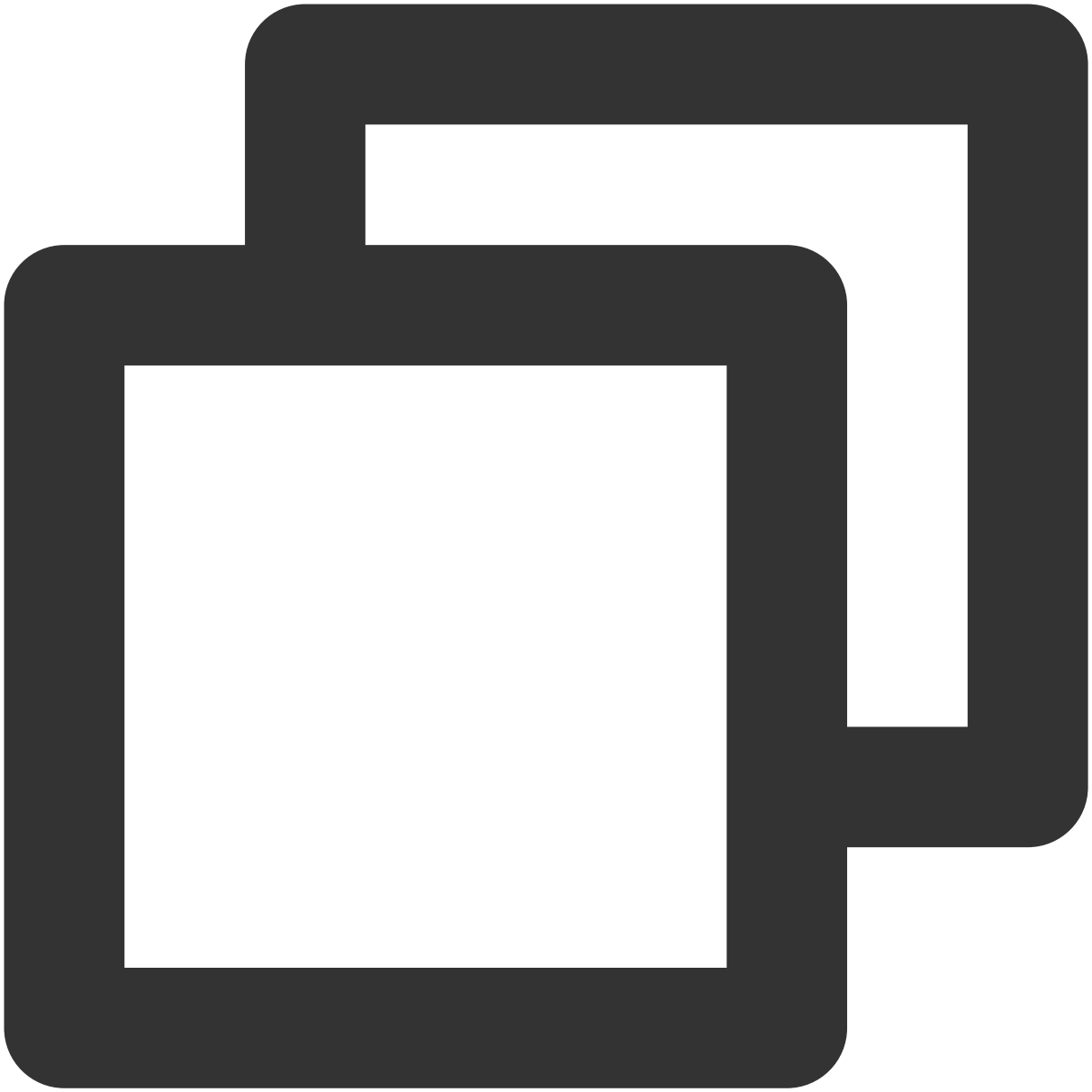
步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：创建 React 项目

1. 如果您尚未安装 create-react-app ，可以在 terminal 或 cmd 中采用如下方式进行安装：

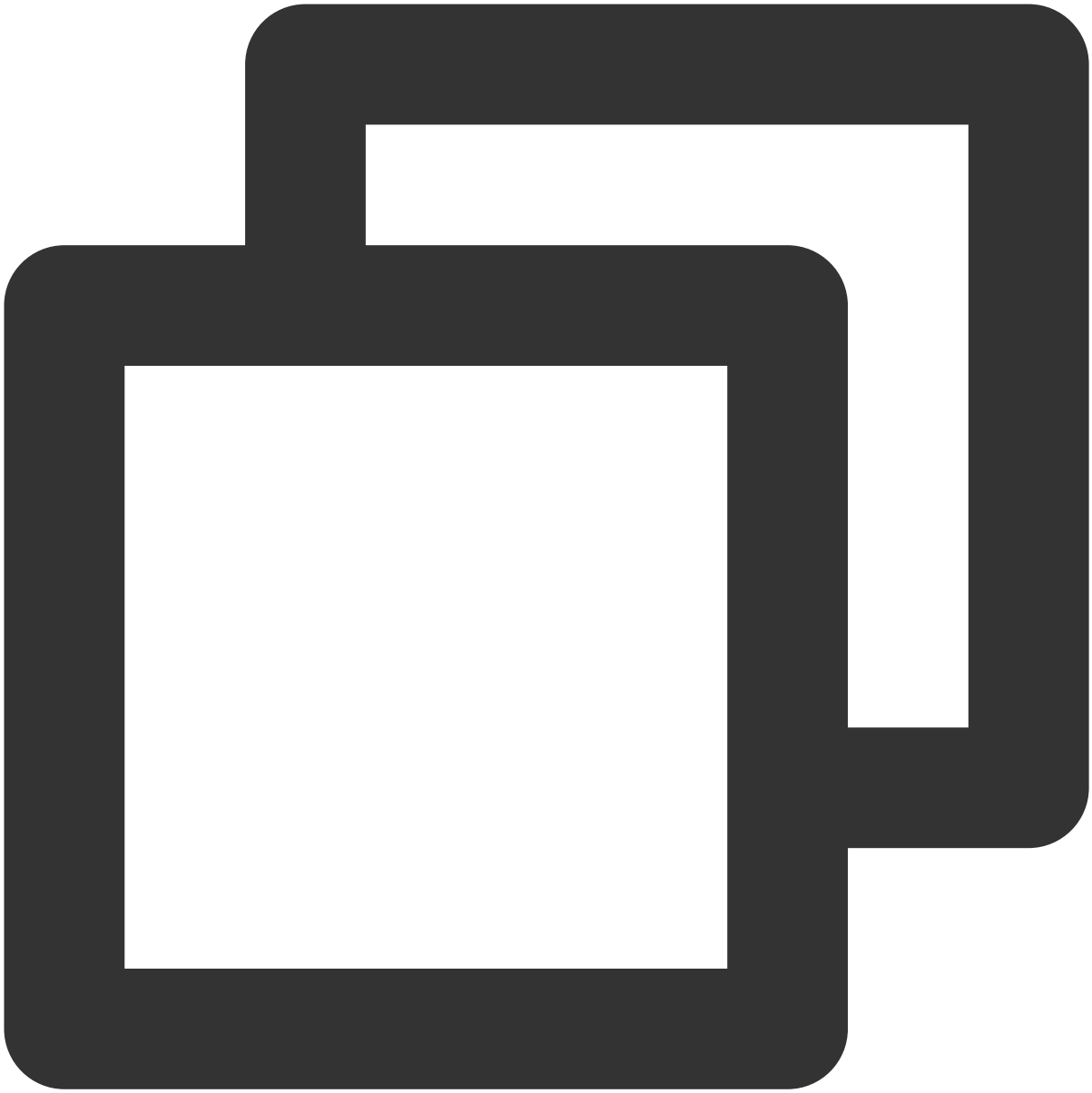
版权所有：腾讯云计算（北京）有限责任公司



```
npm install -g create-react-app
```

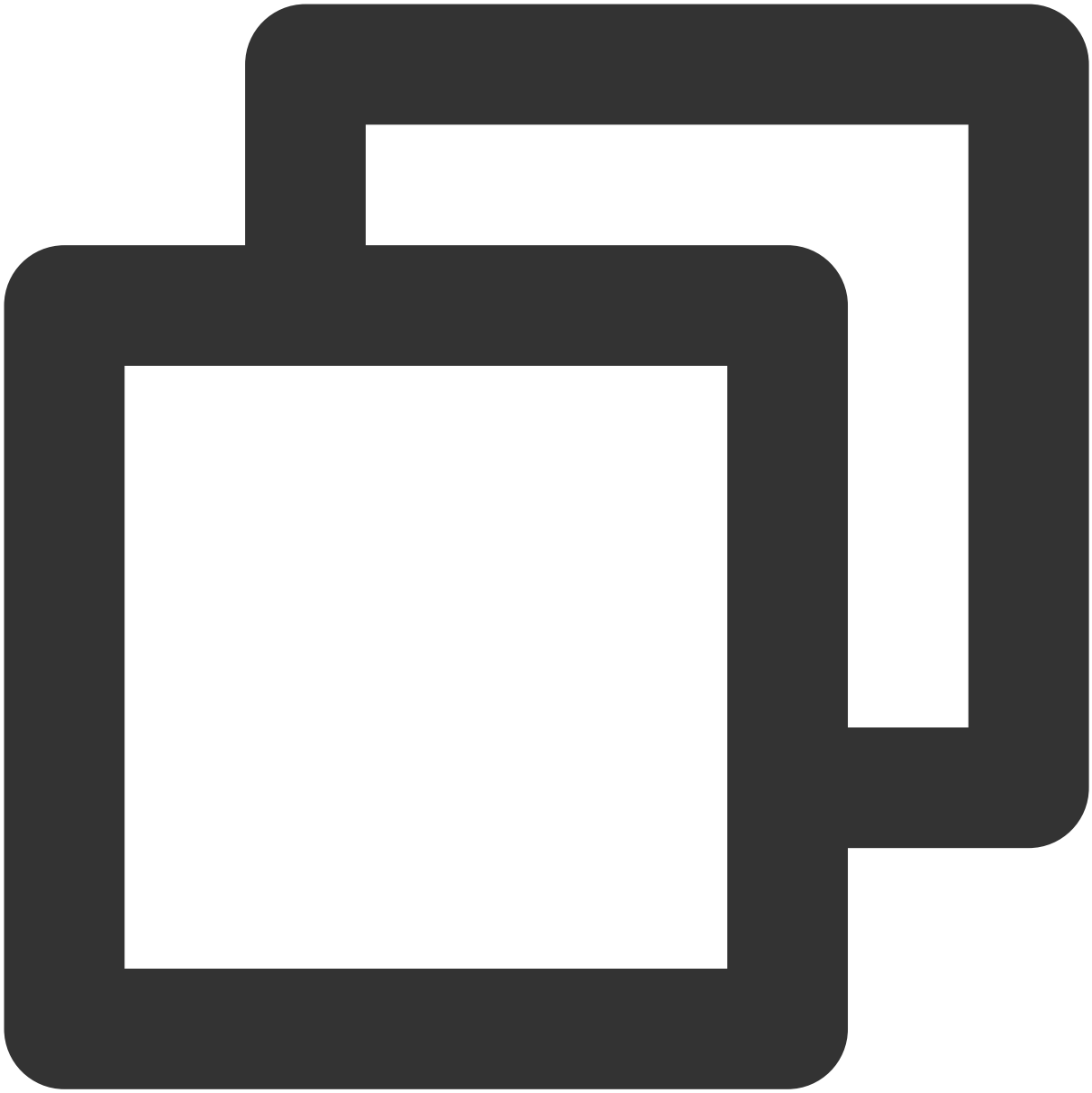
2. 创建一个新的 React 项目，您可自行选择是否需要使用 ts 模板。

版权所有：腾讯云计算（北京）有限责任公司



```
npx create-react-app tuicallkit-demo --template typescript
```

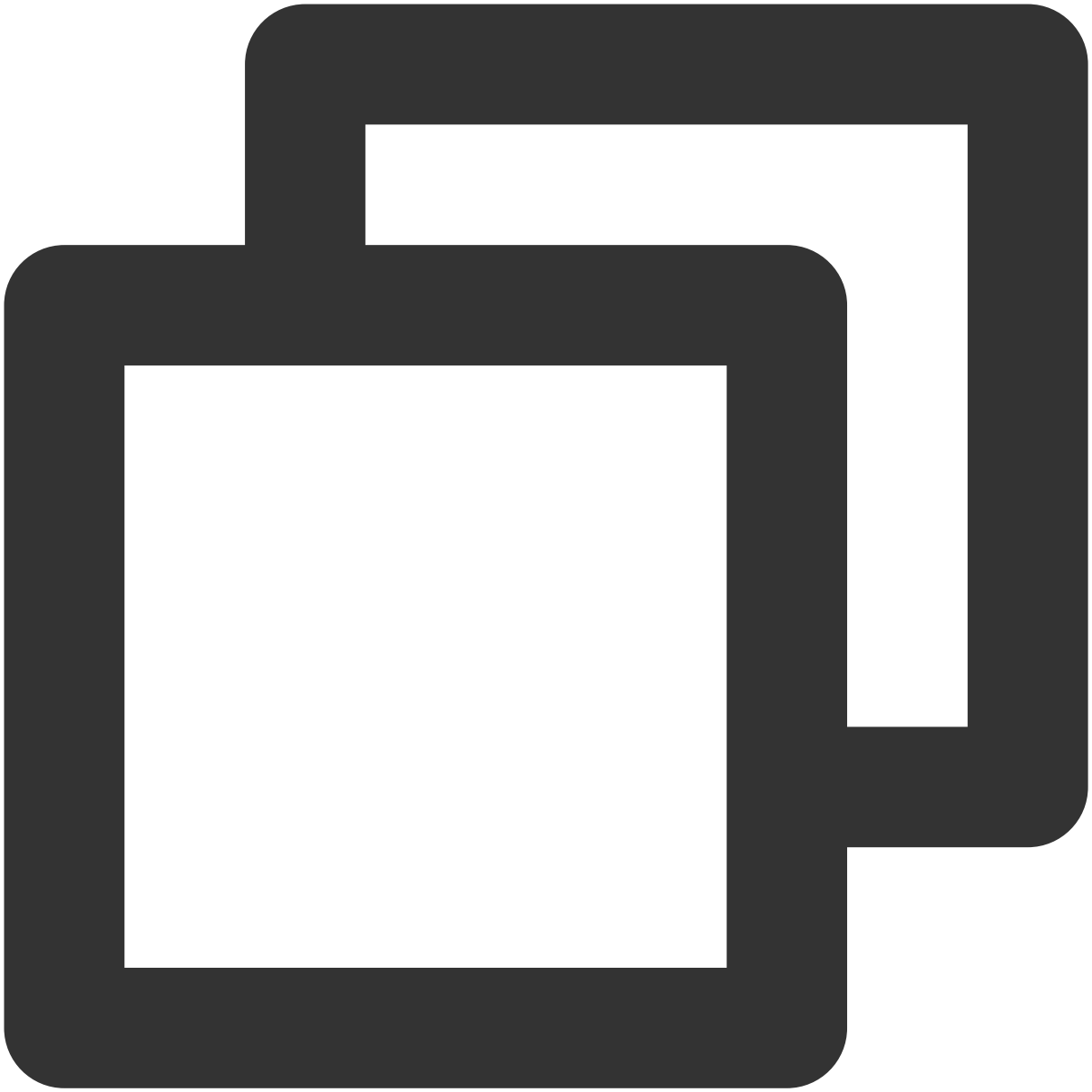
3. 创建项目完成后，切换到项目所在目录。



```
cd tuicallkit-demo
```

步骤三：下载 TUICallKit 组件

1. 通过 npm 方式下载 @tencentcloud/call-uikit-react组件并在项目中使用。

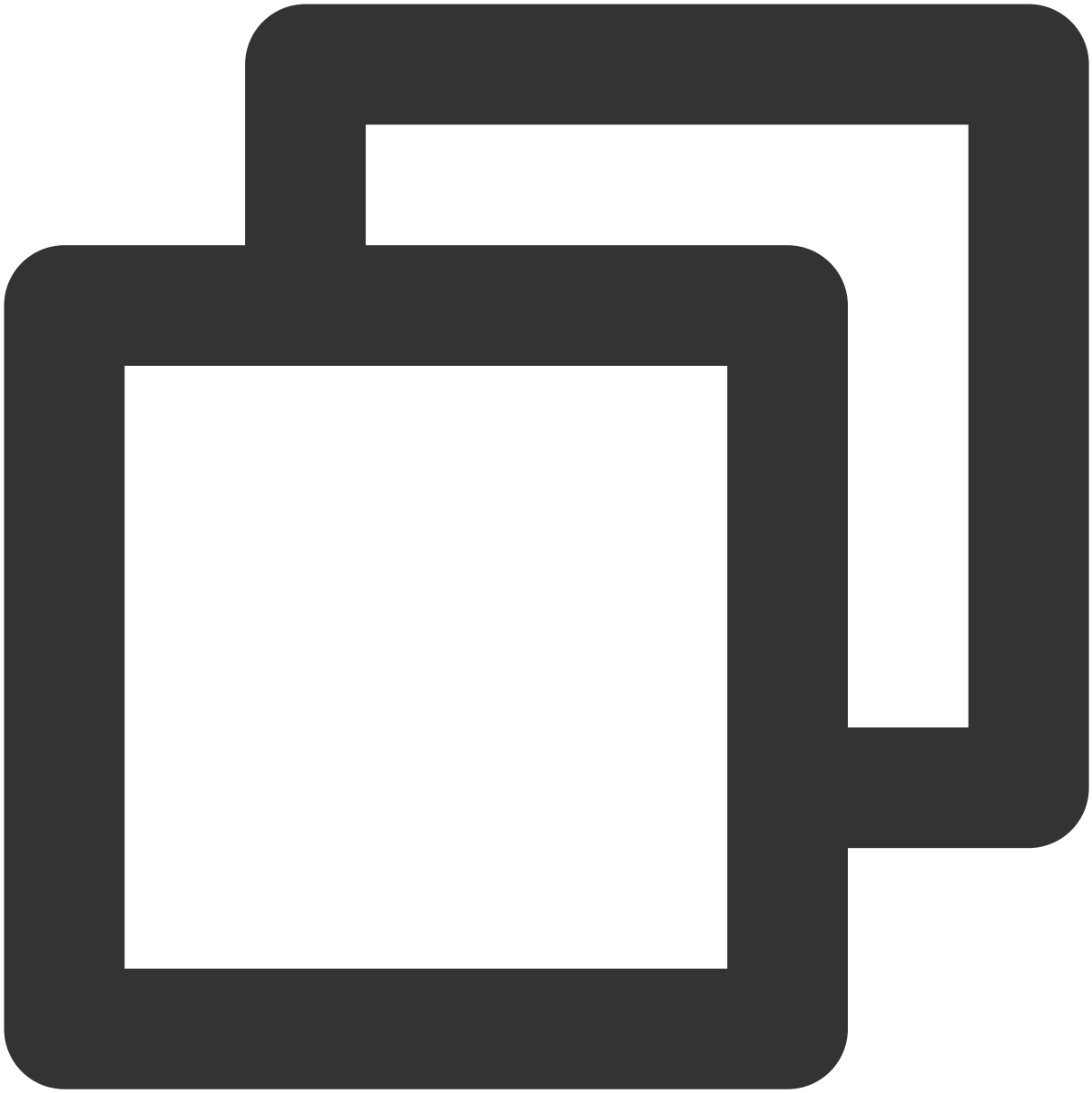


```
npm install @tencentcloud/call-uikit-react
```

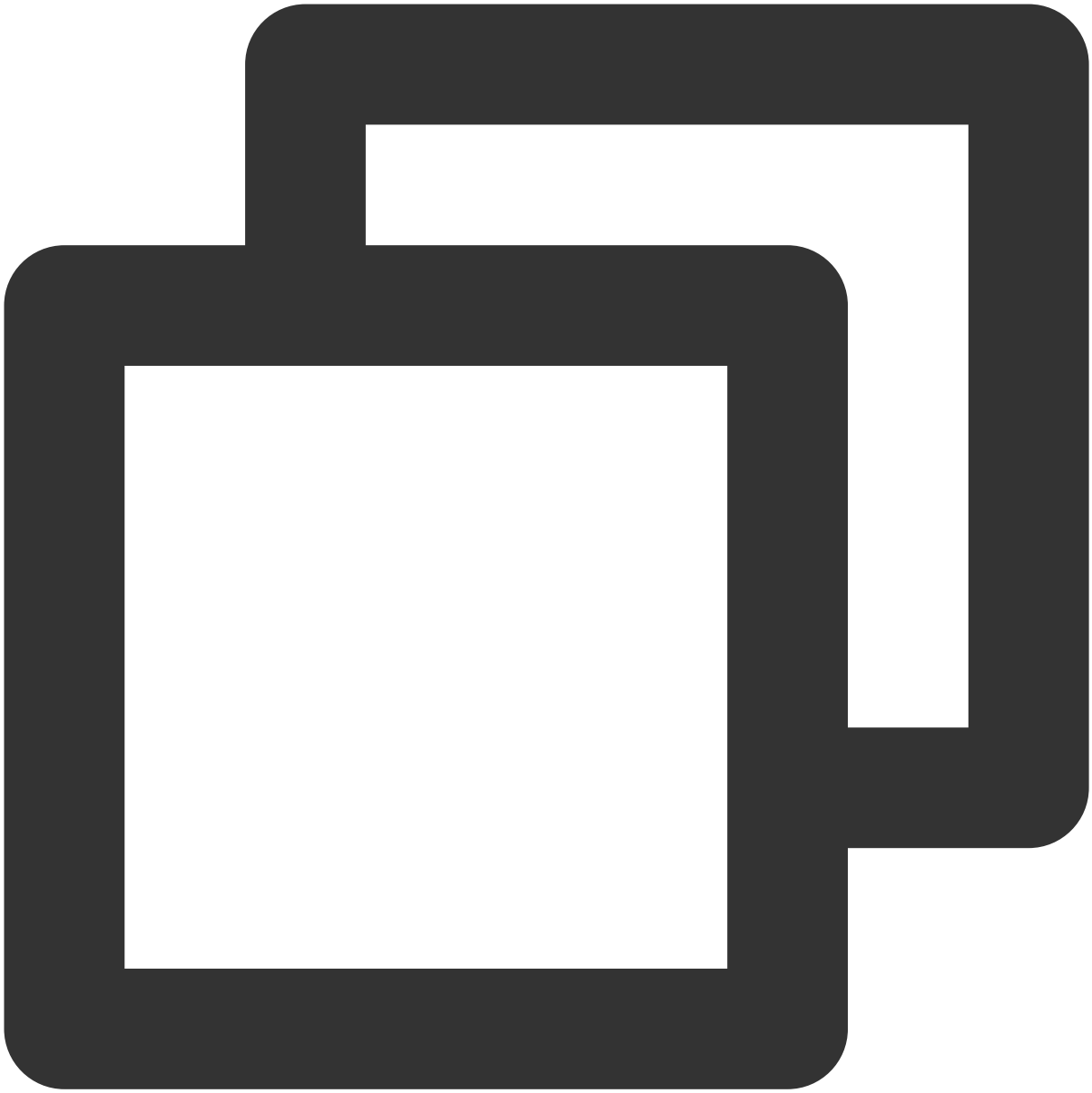
2. 将 debug 目录复制到您的项目目录 `tuicallkit-demo/src` 。

MacOS

Windows



```
cp -r node_modules/@tencentcloud/call-uikit-react/debug ./src
```

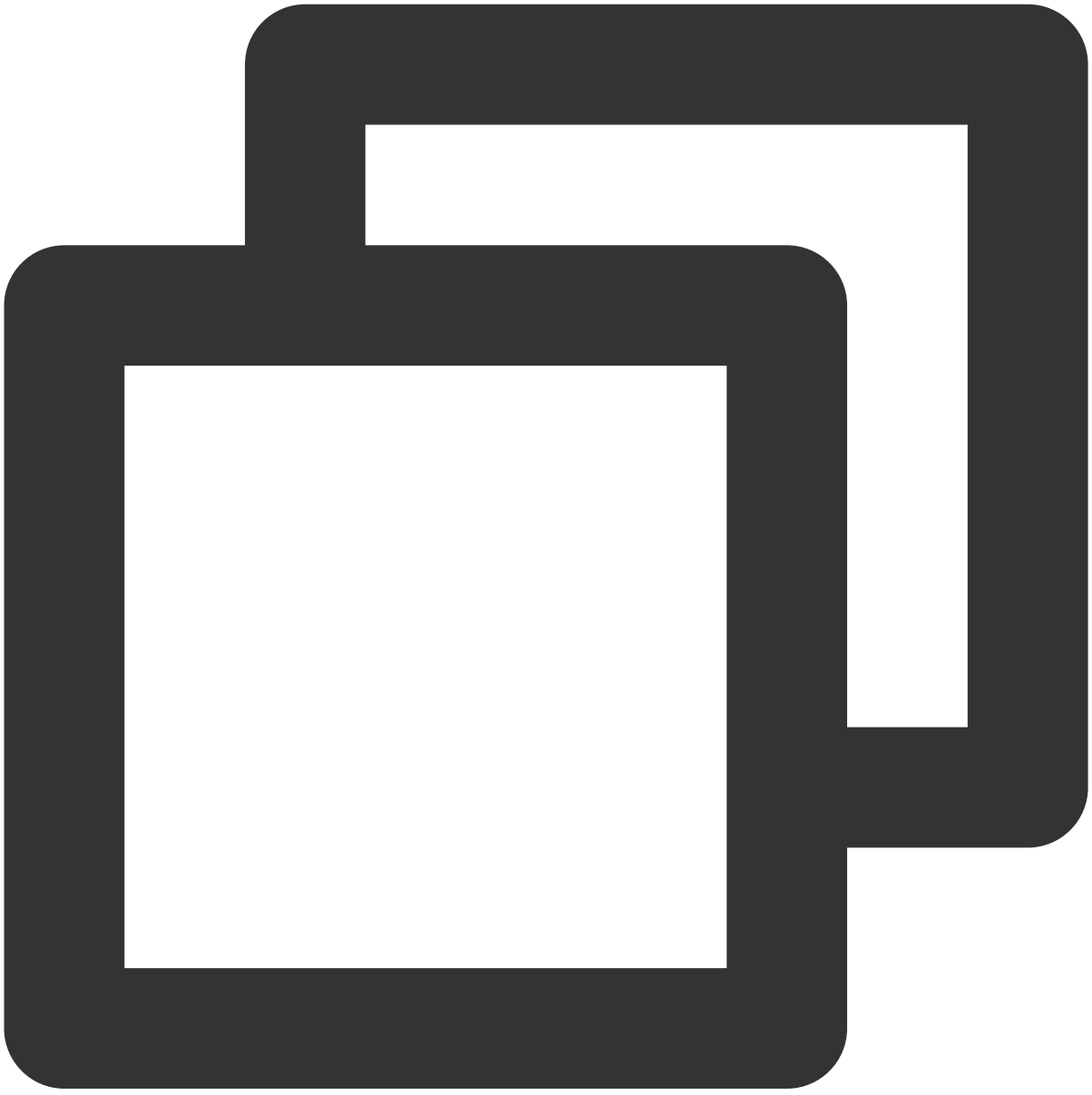


```
xcopy node_modules\\@tencentcloud\\call-uikit-vue\\debug .\\src\\debug /i /e
```

步骤四：引入 TUICallKit 组件

1. 如下代码示例中的 SDKAppID、SDKSecretKey 、userID、callUserID 四个参数。
SDKAppID：在腾讯云创建的音视频应用 SDKAppID，参见 开通服务。
SDKSecretKey：用户签名，参见 开通服务。

userID：主叫的 userID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
callUserID：被叫的 userID，需要已初始化存在。(demo 中的 callUserID，当有被叫时输入，无被叫可以不输入)。
2. 在 `tuicallkit-demo/src/App.tsx` 中引入下面代码。



```
import React, { useState, useMemo } from "react";
// @ts-ignore
import { TUICallKit, TUICallKitServer, TUIGlobal } from "@tencentcloud/call-uikit-r
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es";

function App() {
```

```
  const [userData, setUserData] = useState({
    userID: "",
    callUserID: "",
    SDKAppID: 0,         // Replace with your SDKAppID
    SDKSecretKey: "",    // Replace with your SDKSecretKey
    isLogin: false,
  });
  const init = async () => {
    const { SDKAppID, SDKSecretKey, userID } = userData;
    if (SDKAppID && SDKSecretKey && userID) {
      try {
        await login(SDKAppID, SDKSecretKey, userID);
        setUserData((prevState) => ({
          ...prevState,
          isLogin: true as boolean,
        }));
        alert("[TUICallKit] login success.");
      } catch (error) {
        console.error(error);
      }
    } else {
      alert("[TUICallKit] Please fill in the SDKAppID, userID and SecretKey.");
    }
  };
  const login = async (SDKAppID: any, SecretKey: any, userId: any) => {
    const { userSig } = GenerateTestUserSig.genTestUserSig({
      userID: userId,
      SDKAppID: Number(SDKAppID),
      SecretKey: SecretKey,
    });
    await TUICallKitServer.init({ //【1】Initialize the TUICallKit component
      userID: userId,
      userSig,
      SDKAppID: Number(SDKAppID),
    });
  };
  const call = async () => {
    await TUICallKitServer.call({ userID: userData.callUserID, type: 2 }); //【2】Ma
  };
  const callKitStyle = useMemo(() => {
    if (TUIGlobal.isPC) {
      return { width: '960px', height: '630px', margin: '0 auto' };
    }
    return { width: '100%', height: window.innerHeight };
  }, [TUIGlobal.isPC]);

  return (
```

```
    <>
      <TUICallKit style={callKitStyle}></TUICallKit>
      <div style={{ width: 350, margin: '0 auto' }}>
        <h4> {userData.isLogin ? "Call Panel" : "Login Panel"} </h4>
        <input
          value={userData.isLogin ? userData.callUserID : userData.userID}
          onChange={(value) => {
            const key = userData.isLogin ? "callUserID" : "userID";
            setUserData((prevState) => ({
              ...prevState,
              [key]: value.target.value,
            }));
          }}
          style={{ width: 230, margin: '0 auto' }}
          placeholder={
            userData.isLogin ? "Please enter callee's userID" : "Please enter your
          }
        />
        <button onClick={userData.isLogin ? call : init}> {userData.isLogin ? "call
        <p style={{margin: '10'}}> {userData.isLogin ? `Your userID:  ${userData.us
      </div>
    </>
  );
}

export default App;
```

步骤五：拨打您的第一通电话

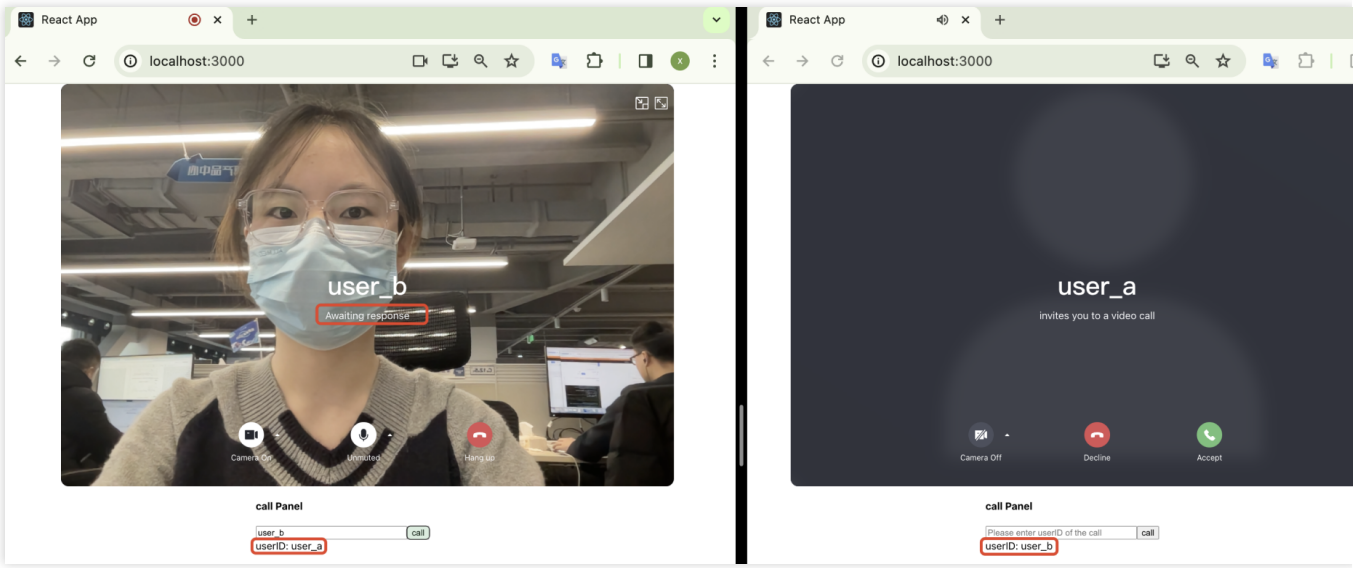
1. 在终端输入 `npm run start` 运行 tuicallkit-demo。

警告：本地环境请在 localhost 协议下访问，具体参见 网络访问协议说明。

2. 输入登录的 userID，点击 login 按钮，弹出 login success 表示初始化成功。



3. 输入被叫的 userID，点击 call 按钮，发起 1v1 视频通话。



更多特性

设置昵称、头像
群组通话
悬浮窗
自定义铃声
监听通话状态、组件回调事件
设置分辨率、填充模式
界面定制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

技术咨询

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com 。

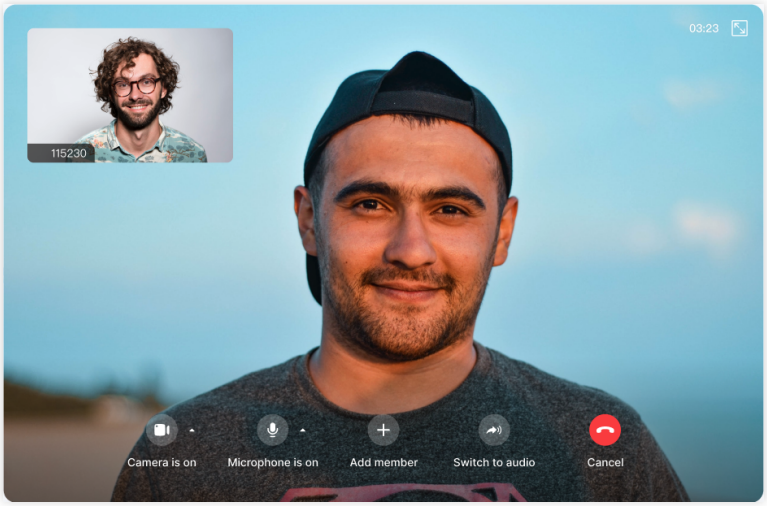
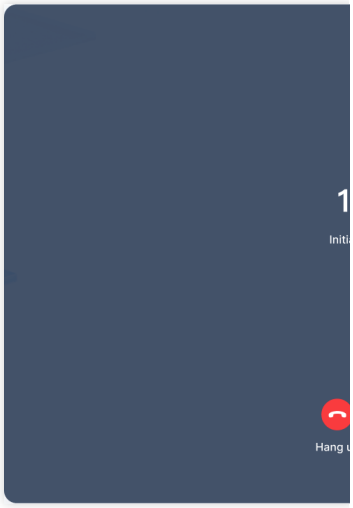
Web&H5 (Vue3)

最近更新时间：2024-04-03 17:23:11

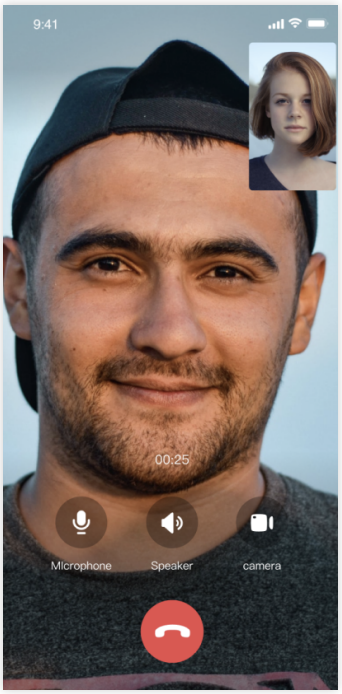
本文将介绍如何快速完成 TUICallKit 组件的接入，您将在 10 分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

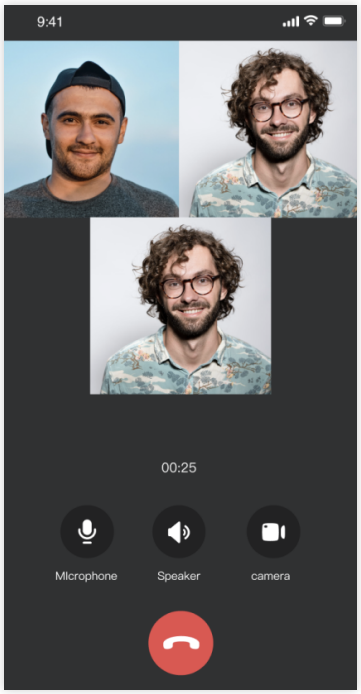
桌面端

移动端

视频通话	语音通话
	

1v1 通话	群组通话





TUICallKit Demo 体验

如果您想要直接体验音视频通话，进入 Demo 体验页面。
如果您想要直接跑通一个新工程，请直接阅读 Web Demo 快速跑通。

环境准备

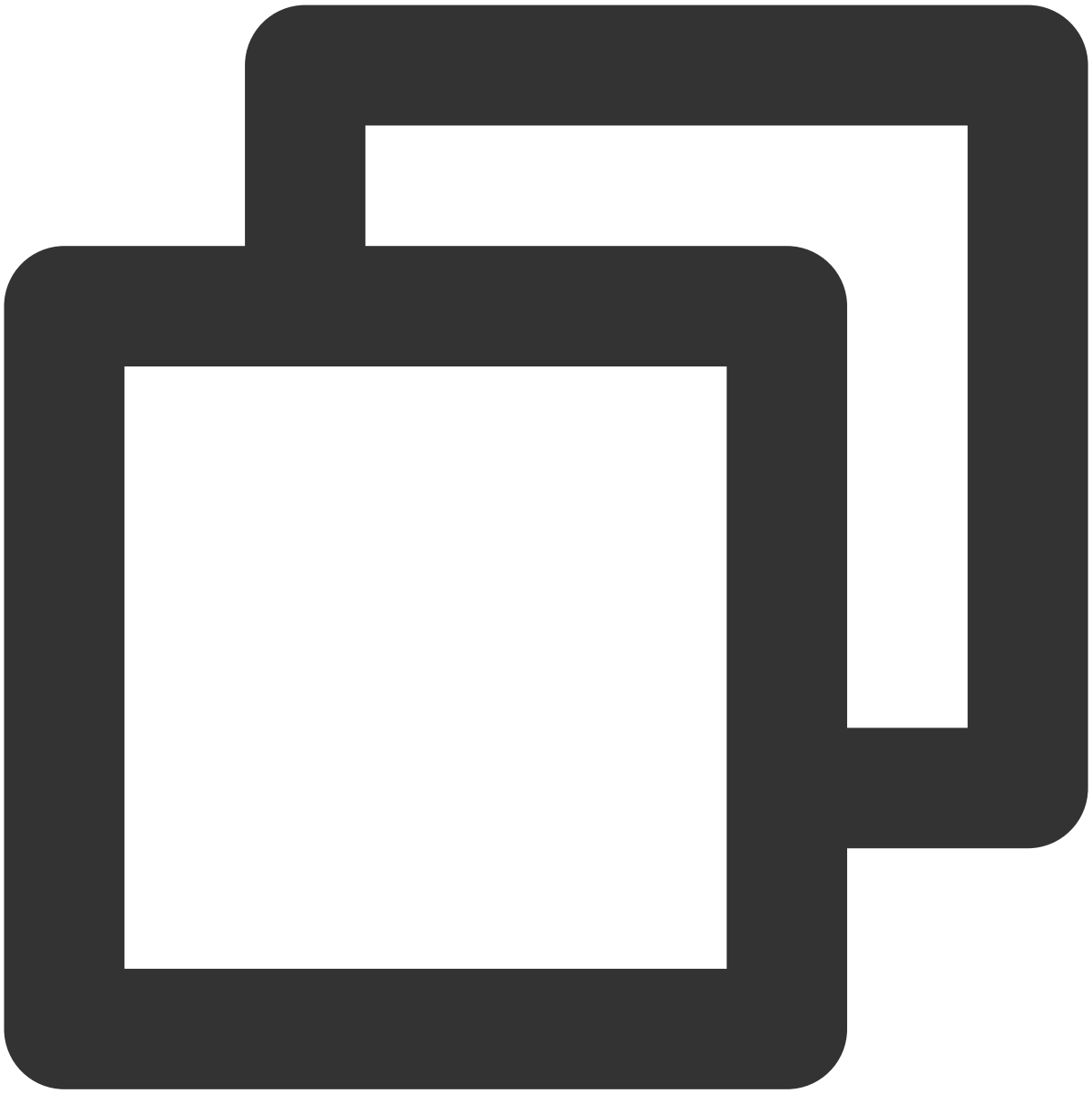
Node.js 版本：Node.js ≥ 12.13.0（推荐使用官方 LTS 版本，npm 版本请与 node 版本匹配）。
Modern browser, supporting WebRTC APIs.

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

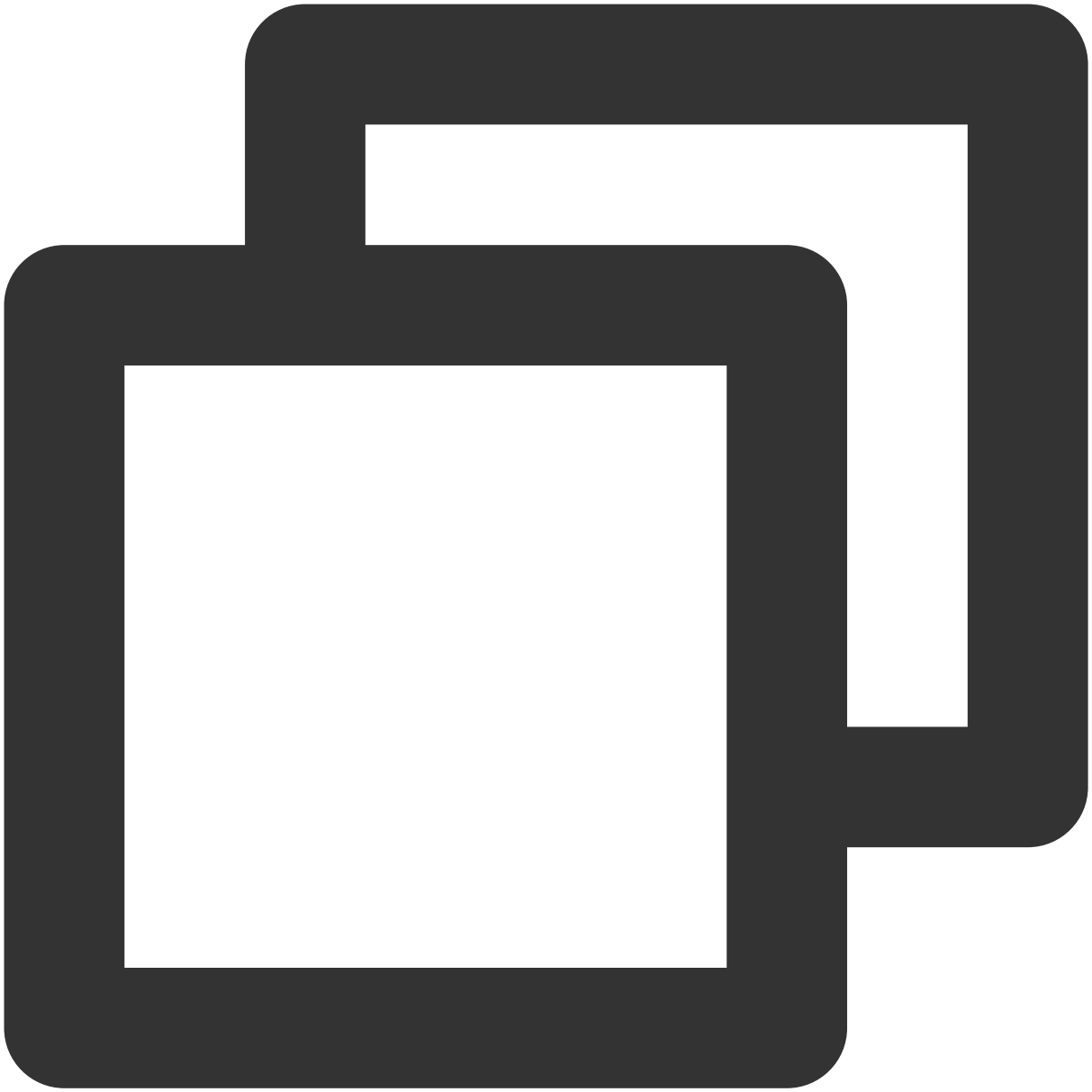
步骤二：创建 Vue3 项目

1. 如果您尚未安装 vue-cli ，可以在 terminal 或 cmd 中采用如下方式进行安装：

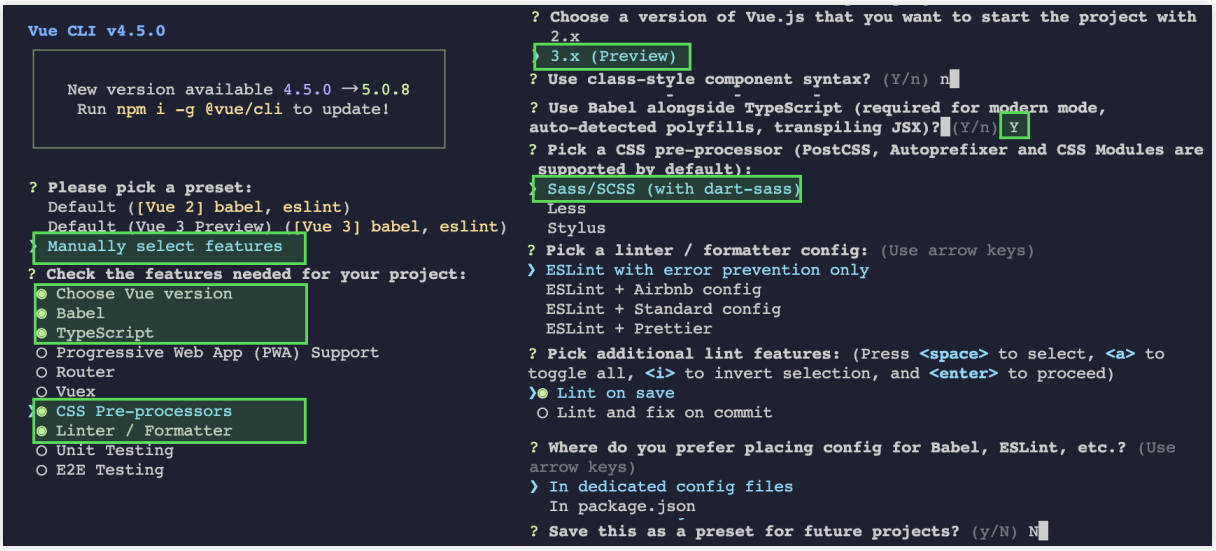


```
npm install -g @vue/cli
```

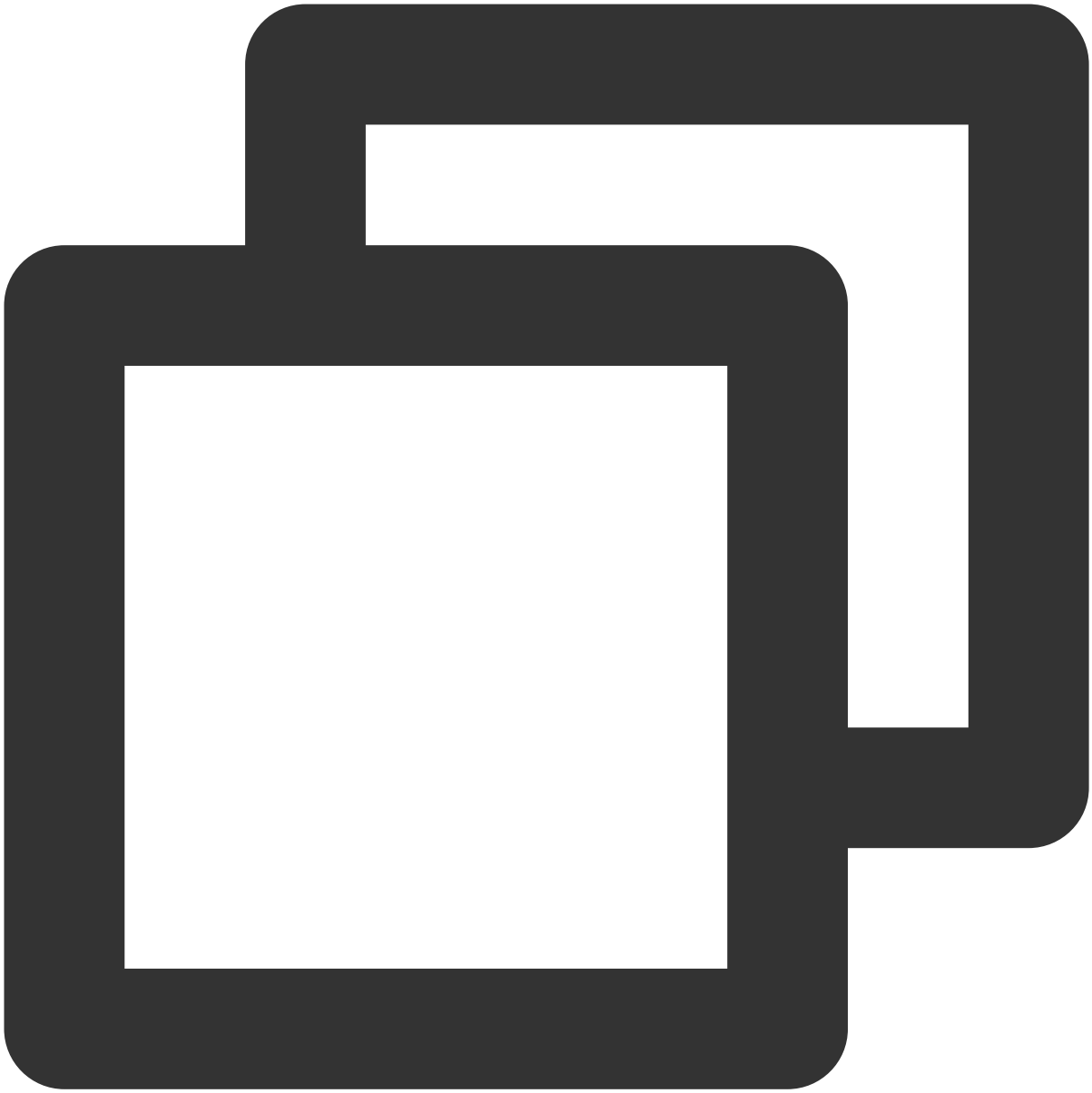
2. 通过 vue-cli 创建项目，并选择下图中所选配置项。



```
vue create tuicallkit-demo
```



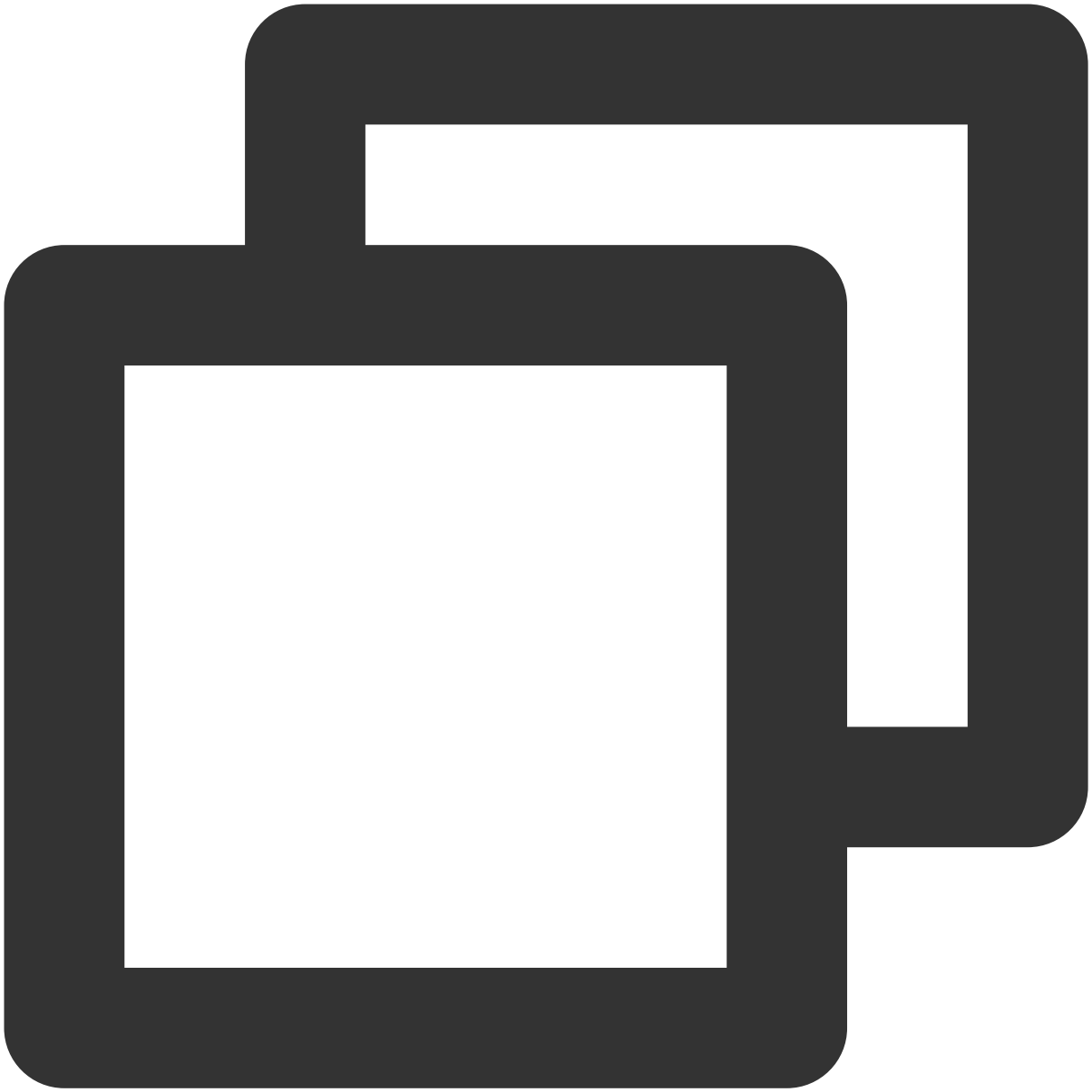
3. 创建完成项目，切换到项目所在目录。



```
cd tuicallkit-demo
```

步骤三：下载 TUICallKit 组件

1. 通过 npm 方式下载 @tencentcloud/call-uikit-vue 组件并在项目中使用。

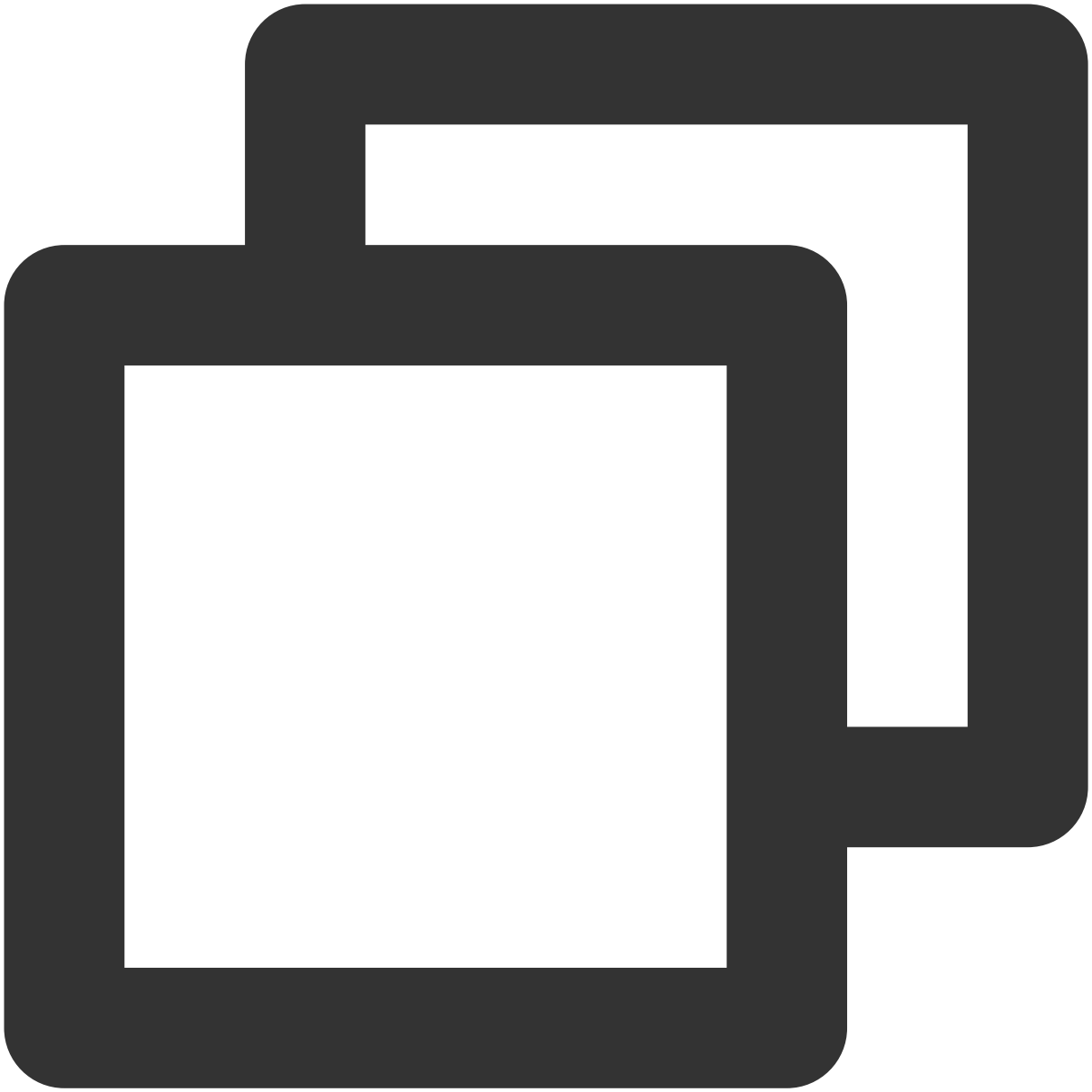


```
npm install @tencentcloud/call-uikit-vue
```

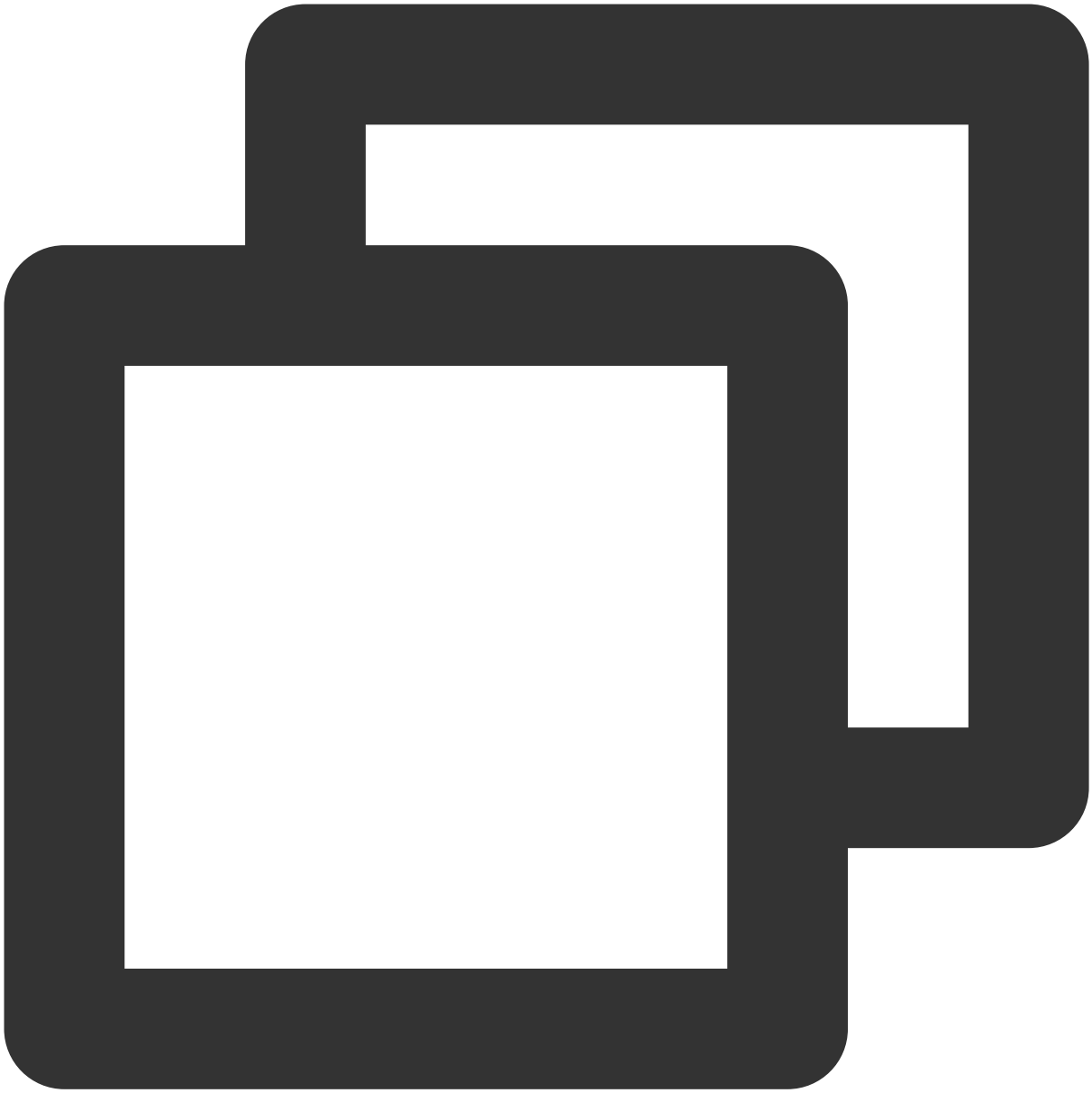
2. 将 debug 目录复制到您的项目。

MacOS

Windows



```
cp -r node_modules/@tencentcloud/call-uikit-vue/debug ./src
```



```
xcopy node_modules\\@tencentcloud\\call-uikit-vue\\debug .\\src\\debug /i /e
```

步骤四：引入 TUICallKit 组件

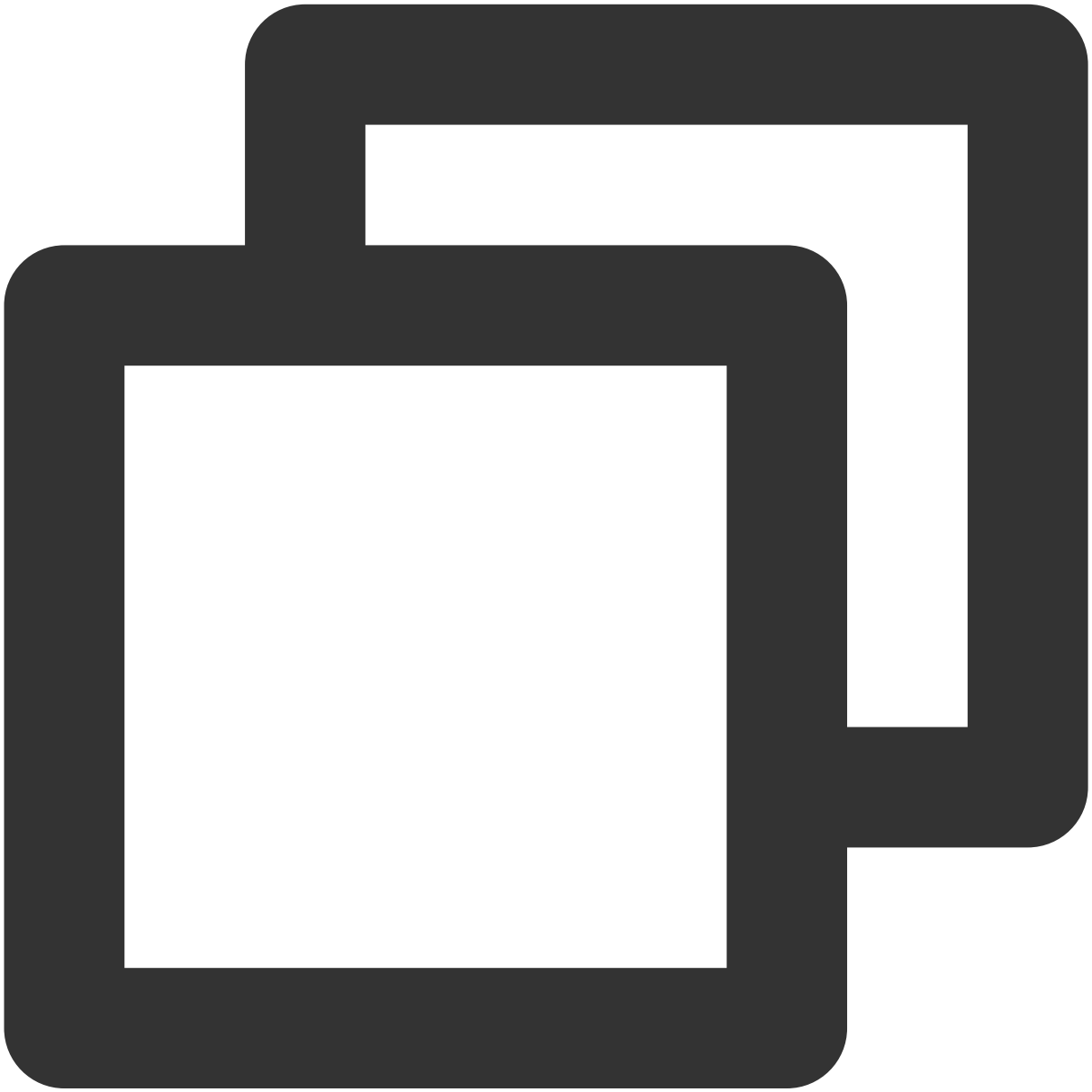
1. 如下代码示例中的 SDKAppID、SDKSecretKey 、userID、callUserID 四个参数。
SDKAppID：在腾讯云创建的音视频应用 SDKAppID，参见 开通服务。
SDKSecretKey：用户签名，参见 开通服务。

userID：主叫的 userID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

callUserID：被叫的 userID，需要已初始化存在。(demo 中的 callUserID，当有被叫时输入，无被叫可以不输入)。

2. 在 `tuicallkit-demo/App.vue` 中引入下面代码。

说明：TUICallKit 组件需要放到一个 dom 节点里，用于显示并控制 TUICallKit 的位置、宽高等样式。



```
<template>
  <div style="padding: 20px 0 0 20px">
    <div><span>userID：</span><input type="text" v-model="userID" /></div>
    <div style="margin-top: 6px;"><span>callUserID：</span><input type="text" v-mode
```

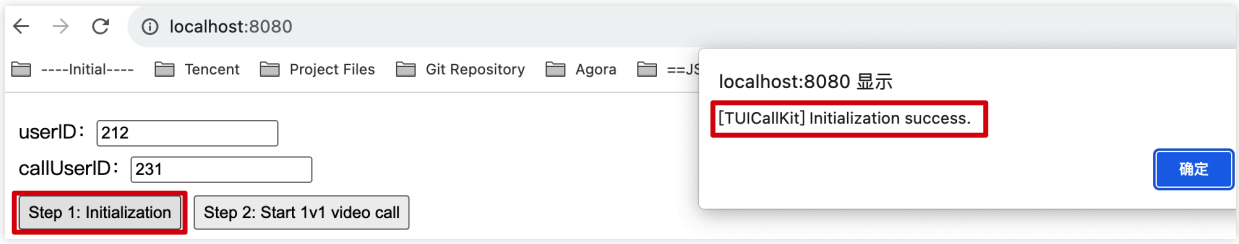
```
    <button @click="init()" style="margin: 10px 10px 10px 0; height: 28px;"> Step 1
    <button @click="call()" style="margin: 10px 10px 10px 0; height: 28px;"> Step 2
    <div style="width: 50rem; height: 35rem; border: 1px solid salmon;">
      <TUICallKit />
    </div>
  </div>
</template>
<script lang="ts" setup>
import { ref } from 'vue';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es";

const userID = ref('');
const callUserID = ref('');
const SDKAppID = 0;       // Replace with your SDKAppID
const SDKSecretKey = '';  // Replace with your SDKSecretKey

async function init() {
  try {
    const { userSig } = GenerateTestUserSig.genTestUserSig({
      userID: userID.value,
      SDKAppID: SDKAppID,
      SecretKey: SDKSecretKey,
    });
    await TUICallKitServer.init({  //【1】Initialize the TUICallKit component
      SDKAppID,
      userID: userID.value,
      userSig,
    });
    alert("[TUICallKit] Initialization success.");
  } catch (error: any) {
    alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
  }
}
async function call() {  //【2】Make a 1v1 video call
  try {
    await TUICallKitServer.call({ userID: callUserID.value, type: TUICallType.VIDEO
  } catch (error: any) {
    alert(`[TUICallKit] Call failed. Reason: ${error}`);
  }
}
</script>
```

步骤五：拨打您的第一通电话

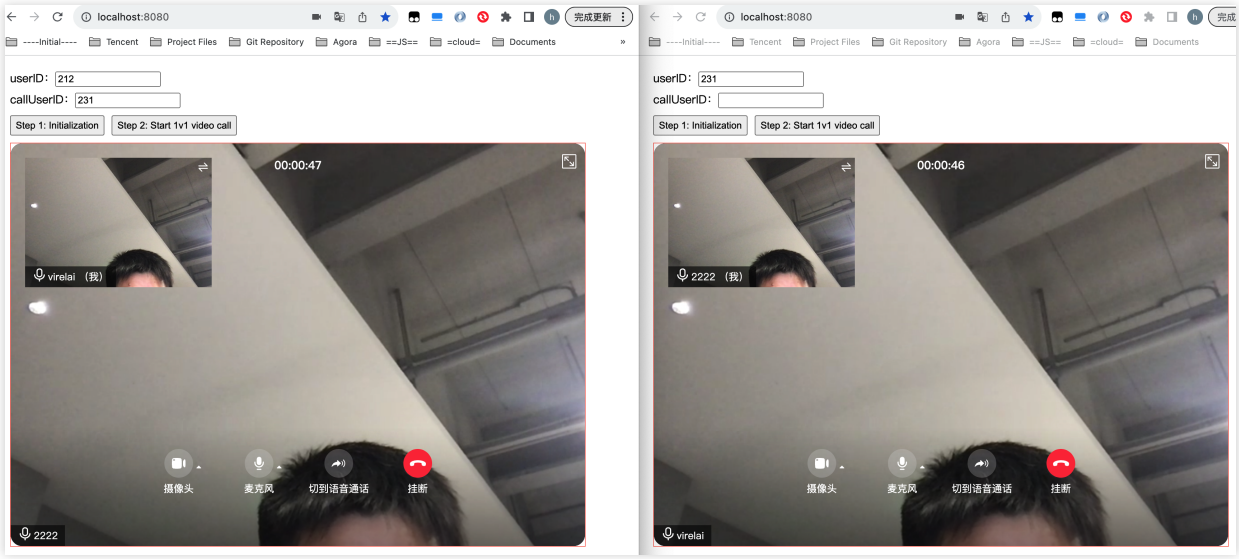
在终端输 `npm run serve` 运行 tuicallkit-demo。

警告：本地环境请在 localhost 协议下访问，具体参见 网络访问协议说明。

点击 `Step 1: Initialization` Button，弹出 **Initialization success** 表示初始化成功。



点击 `Step 2: Start 1v1 video call` 发起 1v1 视频通话，具体效果如下图所示。



更多特性

设置昵称、头像

群组通话

悬浮窗

自定义铃声

监听通话状态、组件回调事件

设置分辨率、填充模式

界面定制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

技术咨询

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com 。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-04-03 17:23:11

本文将介绍如何用最短的时间完成 TUICallKit 组件的接入，跟随本文档，您可以在10分钟内完成接入工作，并最终得到一个包含完备 UI 界面的音视频通话功能。

环境准备

Flutter 3.0及更高版本。

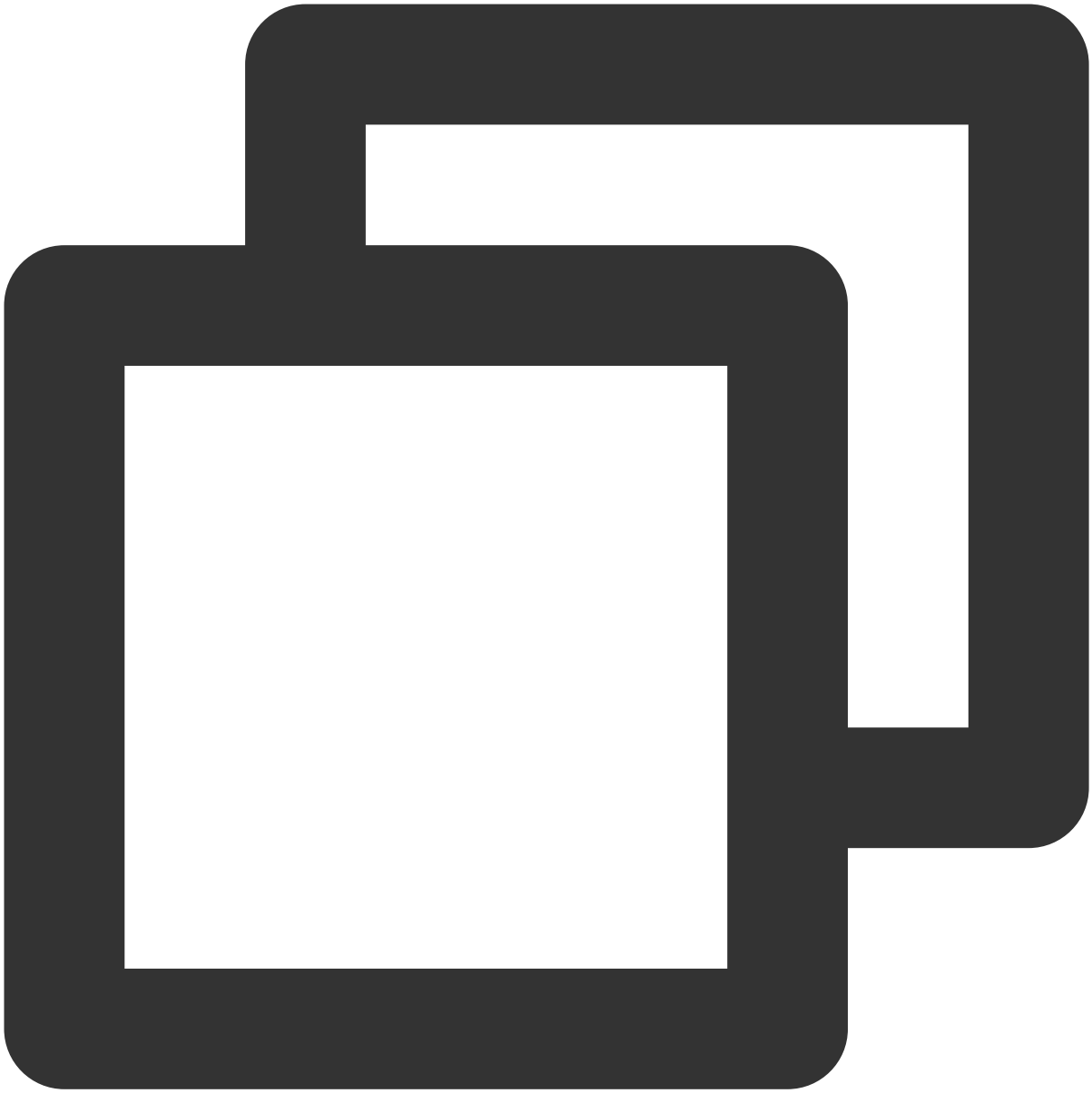
步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：导入TUICallKit组件

在命令行执行以下命令安装组件 tencent_calls_uikit 插件。

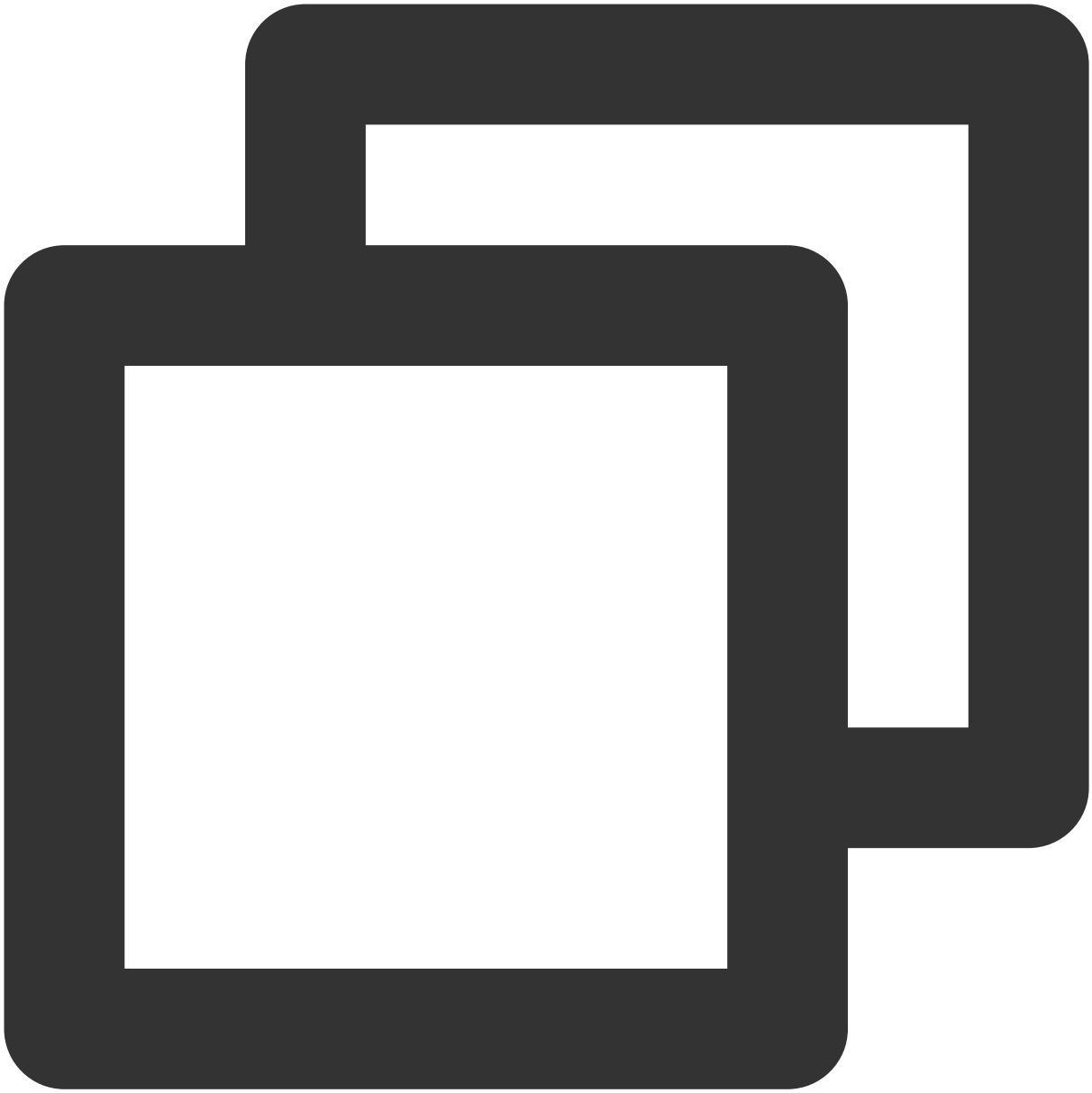
版权所有：腾讯云计算（北京）有限责任公司



```
flutter pub add tencent_calls_uikit
```

步骤三：完成工程配置

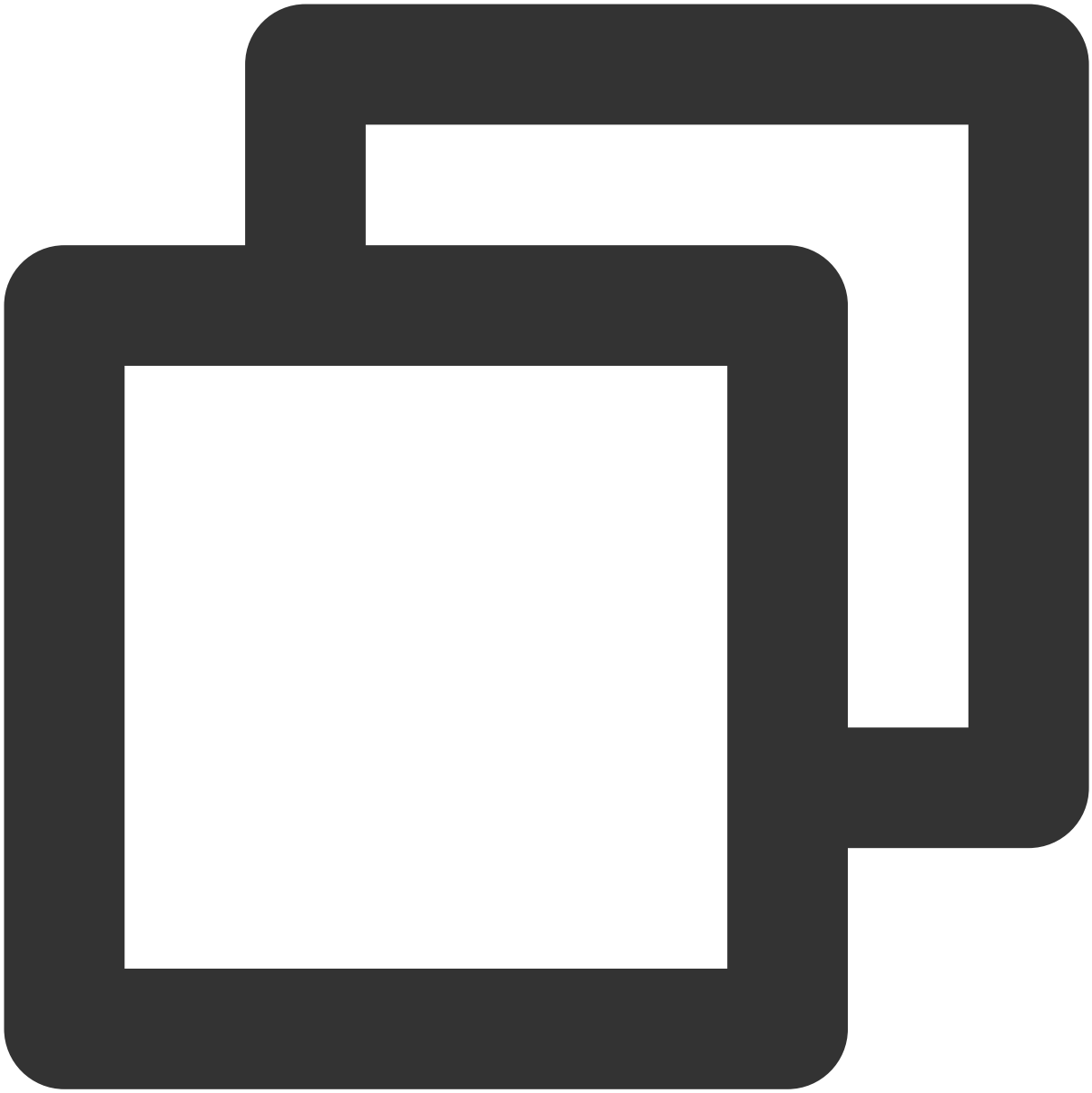
1. 将 TUICallKit的navigatorObserver 添加到 App组件中，以MateriaApp为例，代码如下：



```
import 'package:tencent_calls_uikit/tuicall_kit.dart';

MaterialApp (
      navigatorObservers：[TUICallKit.navigatorObserver],
      ...
)
```

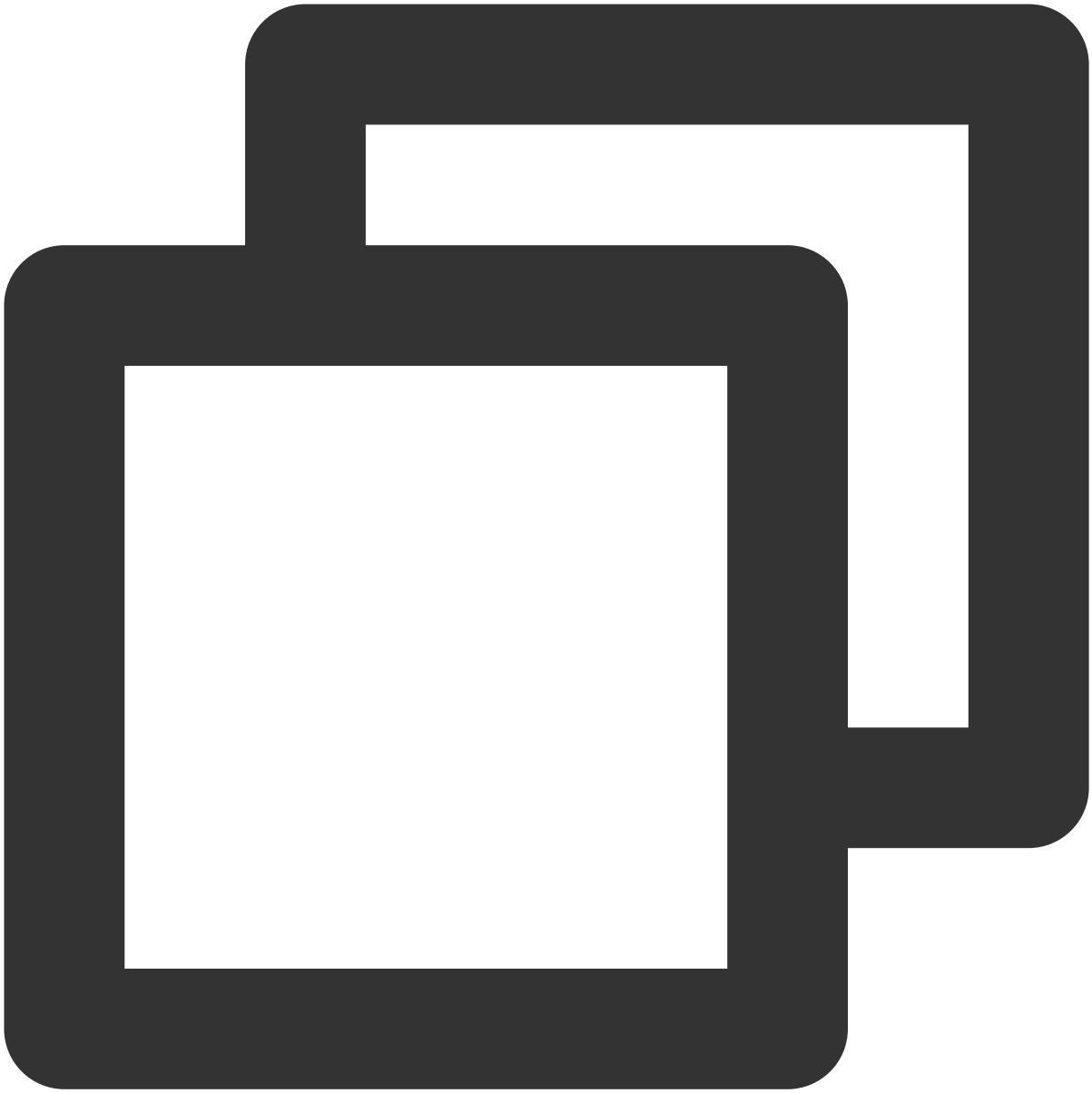
2. 如果您需要编译运行在 Android 平台，由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单。

首先，需要在 app 目录下的 build.gradle 文件中配置并开启混淆规则：



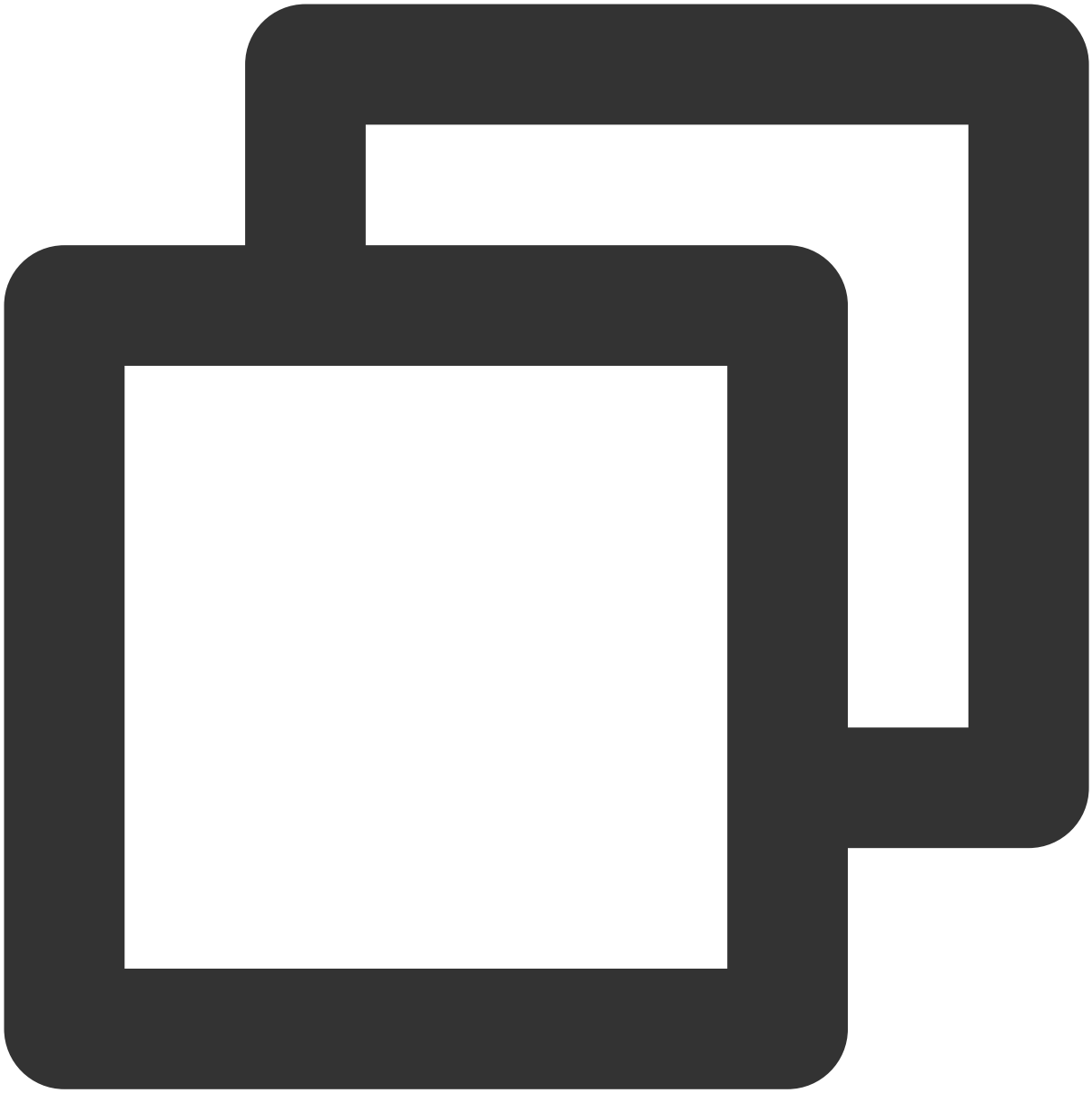
```
  android {
    ...
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard
        }
    }
}
```

再在 proguard-rules.pro 文件中添加如下代码：



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

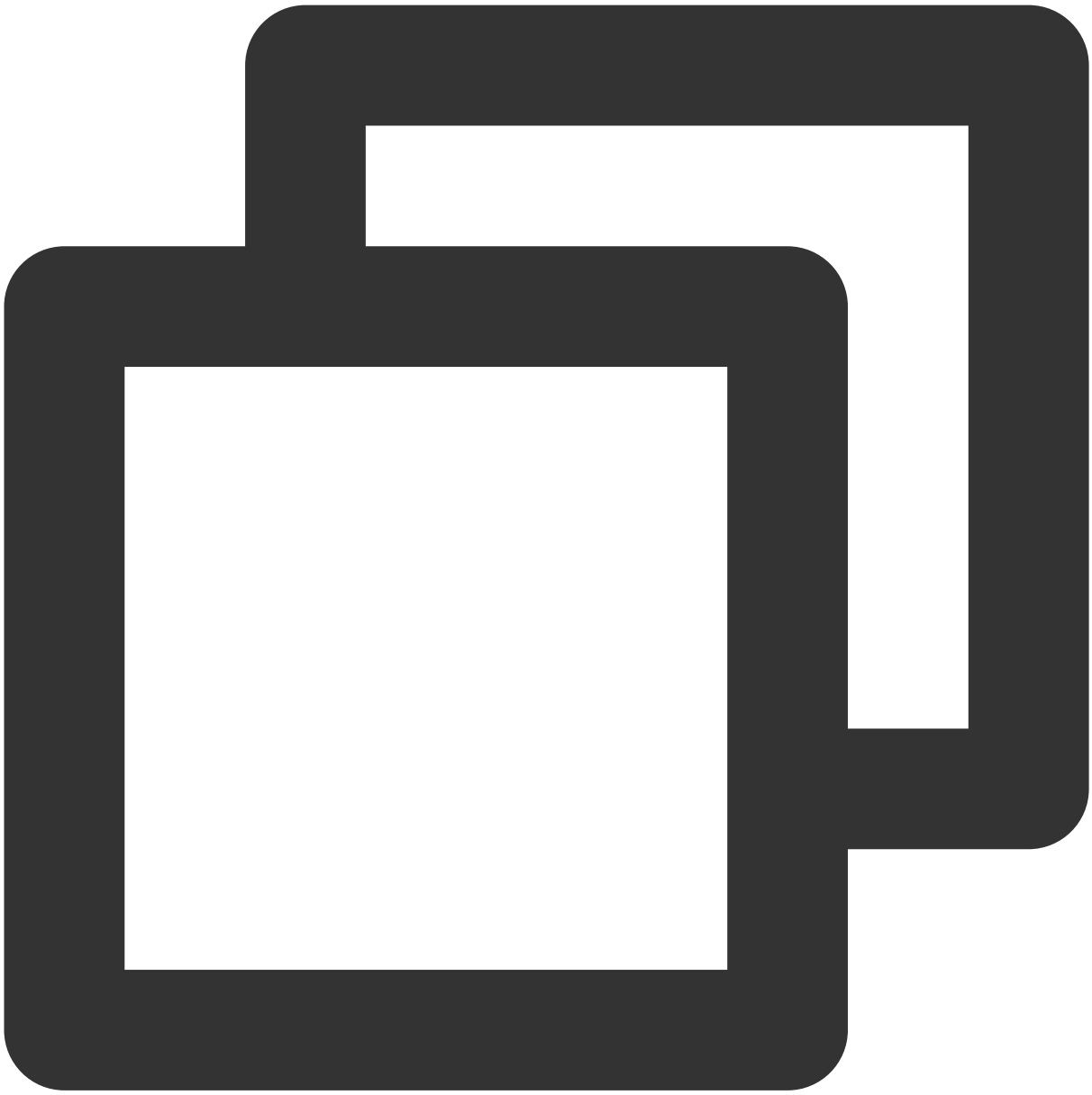
3. 如果您的工程需要在iOS 模拟器上调试，您需要在工程的 `/ios/Podfile` 文件中添加以下代码：



```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
      config.build_settings['VALID_ARCHS'] = 'arm64 arm64e x86_64'
      config.build_settings['VALID_ARCHS[sdk=iphonesimulator*]'] = 'x86_64'
    end
  end
end
```

4. 如果您需要使用 iOS 的音视频功能，您需要授权麦克风和摄像头的使用权限。

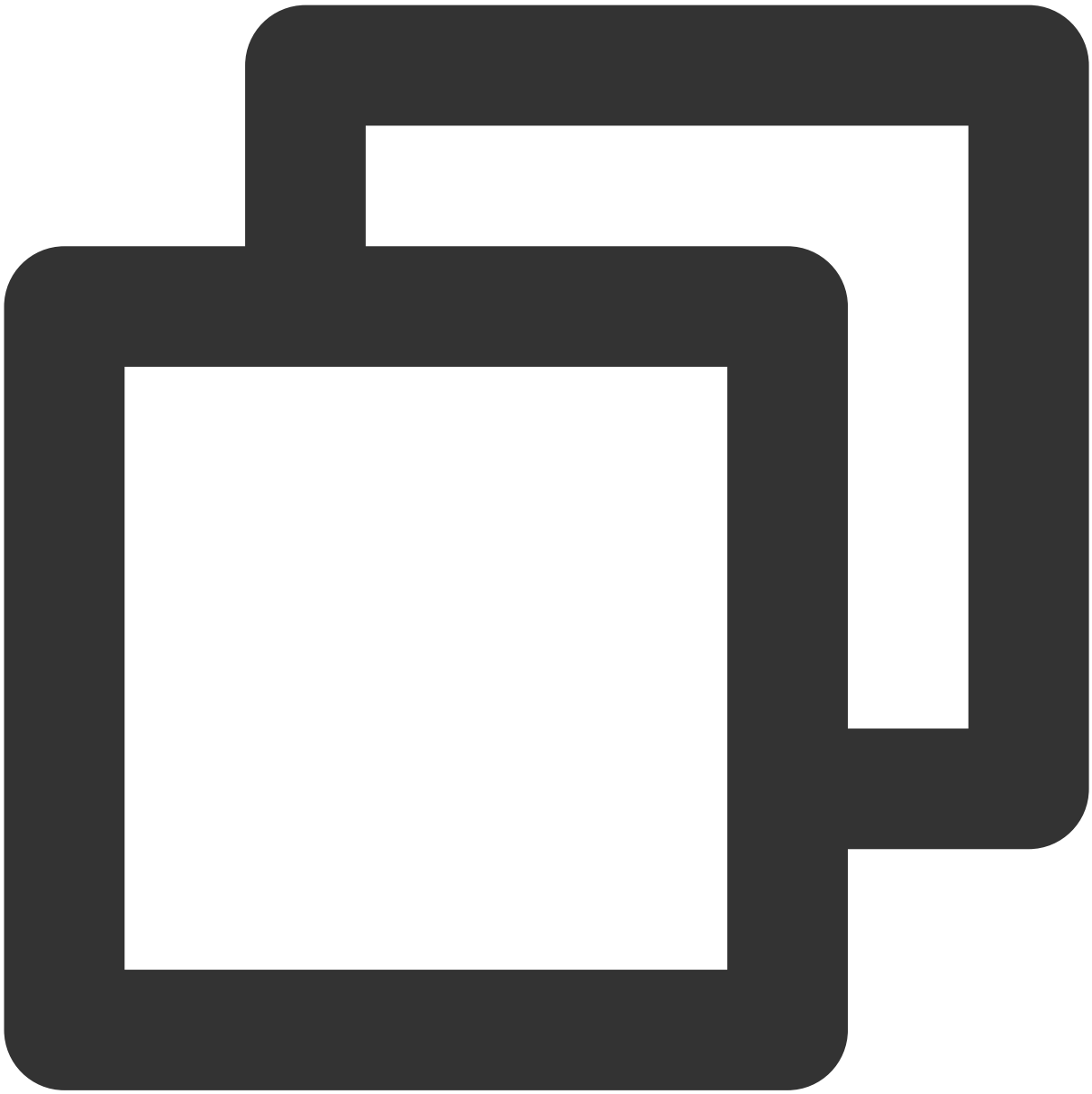
授权操作方法：在您的 iOS 工程的 Info.plist 中添加以下两项，分别对应麦克风和摄像头在系统弹出授权对话框时的提示信息。



```
<key>NSCameraUsageDescription</key>
<string>CallingApp需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>CallingApp需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
```

步骤四：登录 TUICallKit 组件

在您的项目中添加如下代码，这个步骤异常关键，因为只有在登录成功后才能正常使用 TUICallKit 的各项功能，故请您耐心检查相关参数是否配置正确：



```
TUIResult result = TUICallKit.instance.login(SDKAppID,        // 请替换为第一步得到的SDK
                                             'userId',        // 请替换为您的User ID
                                             'userSig');      // 您可以在控制台计算一个u
```

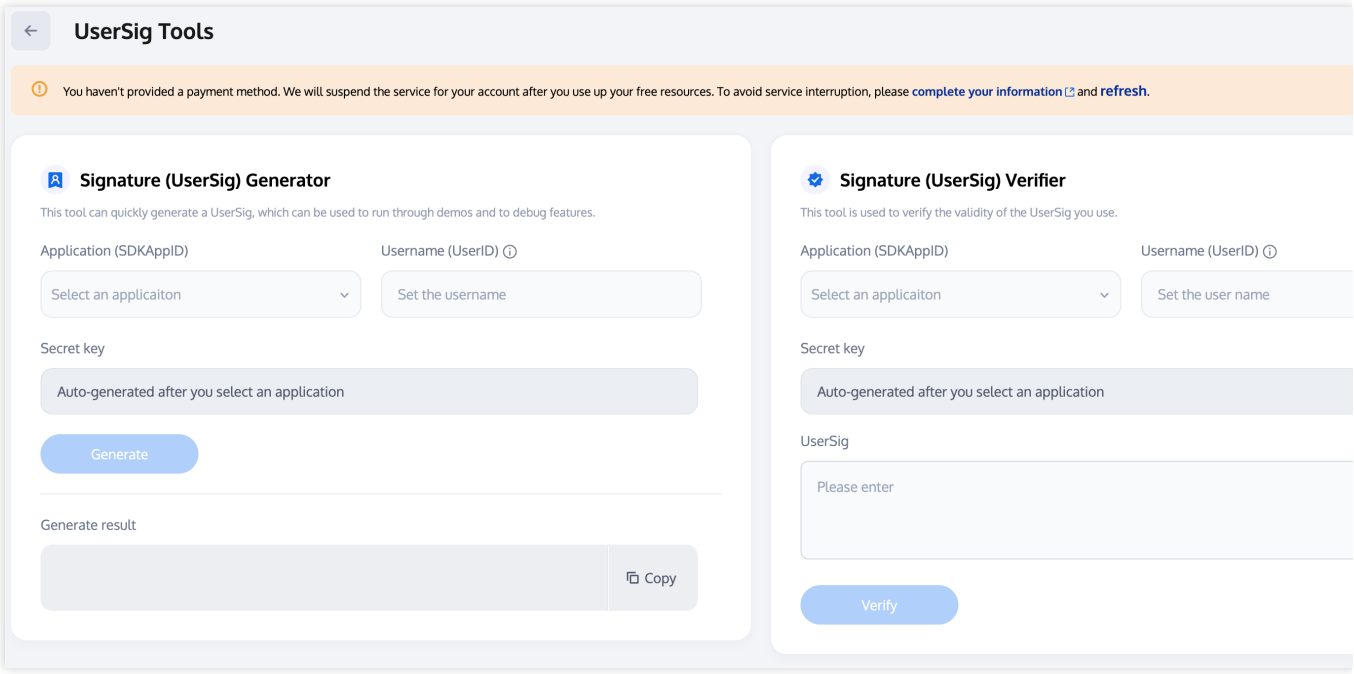
参数说明：

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

sdkAppId：在步骤一中的最后一步中您已经获取到，这里不再赘述。

userId：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

userSig：使用 `SDKSecretKey` 对 `SDKAppID` 、 `UserID` 等信息进行加密，就可以得到 `UserSig` ，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以访问 实时音视频 TRTC 控制台 > UserSig Tools，得到一个临时可用的 UserSig。



更多信息请参见 如何计算及使用 UserSig。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下：

SDKAppID 设置错误。

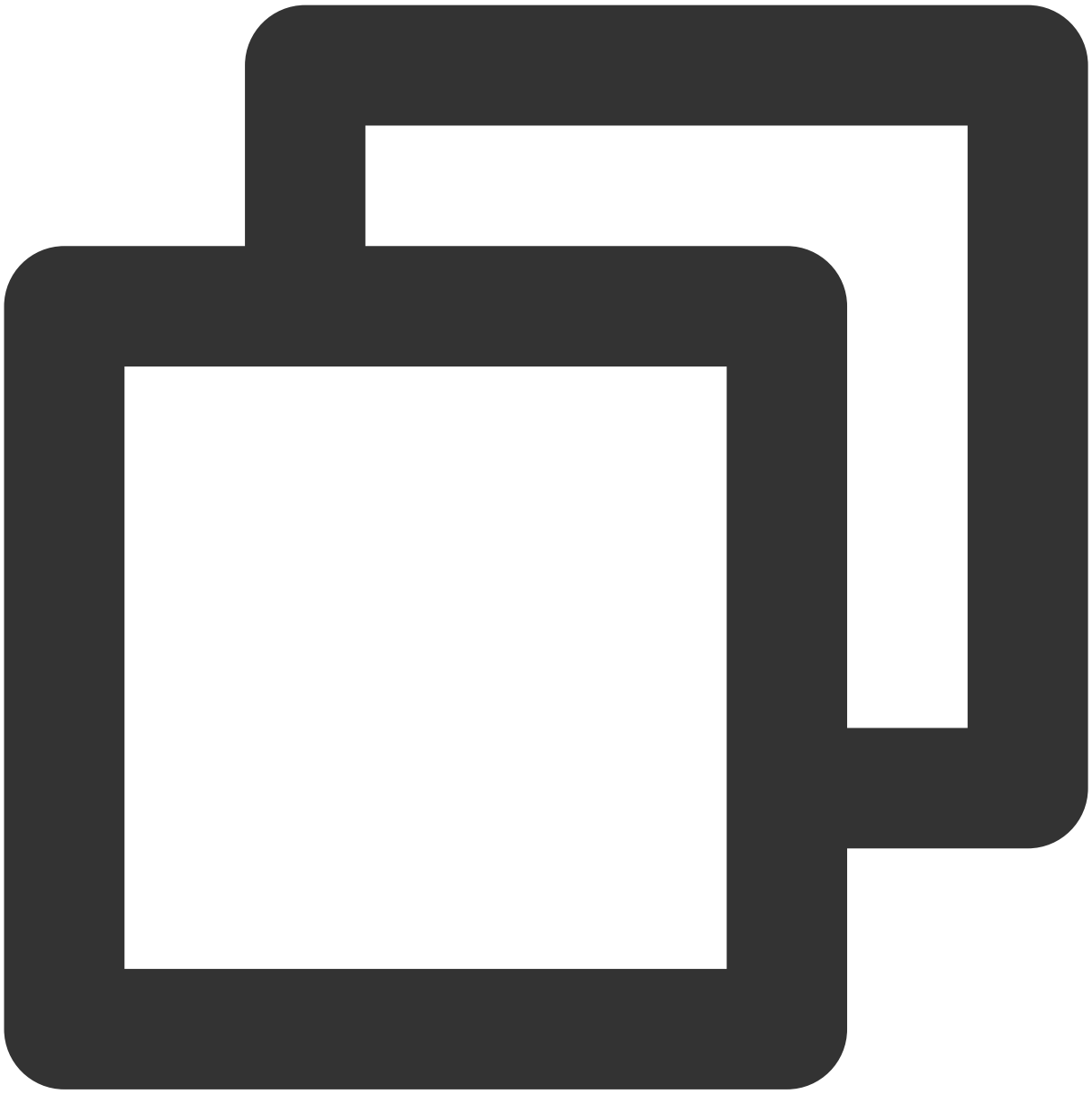
UserSig 被错配成了加密密钥（Secretkey），userSig 是用 SecretKey 把 SDKAppID、UserID 以及过期时间等信息加密得来的，而不是直接把 SecretKey 配置成 UserSig。

UserSig 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

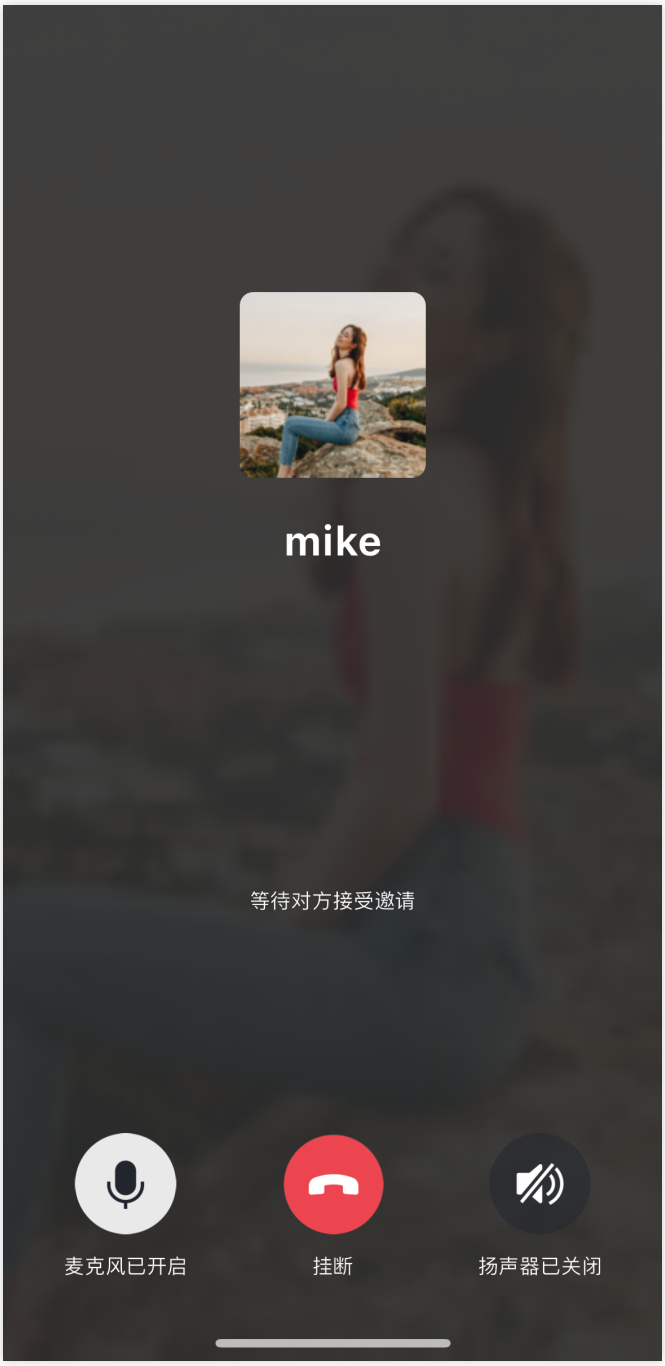
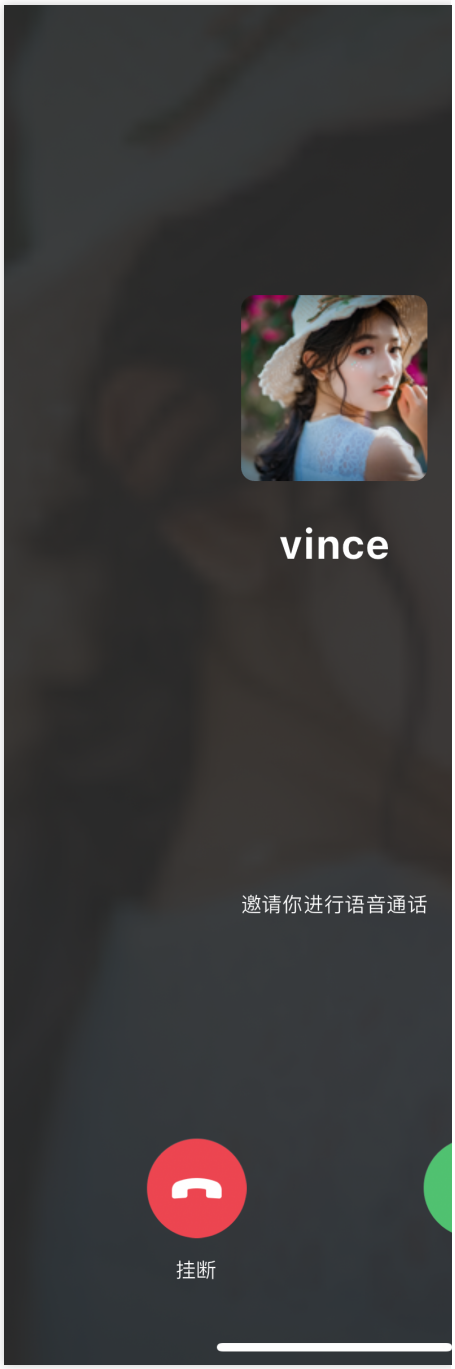
Github 中的示例代码使用了 genTestUserSig 函数在本地计算 UserSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SecretKey，所以我们强烈建议您将 UserSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUICallKit 组件时向您的服务器请求实时计算出的 UserSig。

步骤五：拨打您的第一通电话

主叫方与被叫方登录成功后，主叫方通过调用 TUICallKit 的 call 方法并指定通话类型和被叫方的 userId，就可以发起语音或者视频通话，被叫方此时就可接受到来电邀请。



```
// 假设给mike拨打视频电话
TUICallKit.instance.call('mike', TUICallMediaType.video);
```

	
主叫方	被叫方

更多特性

界面定制
离线推送
群组通话
悬浮窗

自定义铃声
监听通话状态
云端录制
美颜特效

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

uniapp（客户端）

最近更新时间：2024-04-03 17:23:11

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在10分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。



TUICallKit Demo 体验

TUICallKit 插件地址：TUICallKit 插件链接。
如果您想要直接跑通一个新工程，请直接阅读 uni-app Demo 快速跑通。

开发环境要求

开发工具说明：建议使用最新的 HBuilderX 编辑器 。
插件调试说明：原生插件暂不支持模拟器调试。
iOS 设备要求：iOS 系统 ≥ 9.0 ，支持音视频通话的真机设备。
Android 设备要求：Android 系统 ≥ 5.0（SDK API Level 21），支持音视频的真机设备，允许 USB 调试。

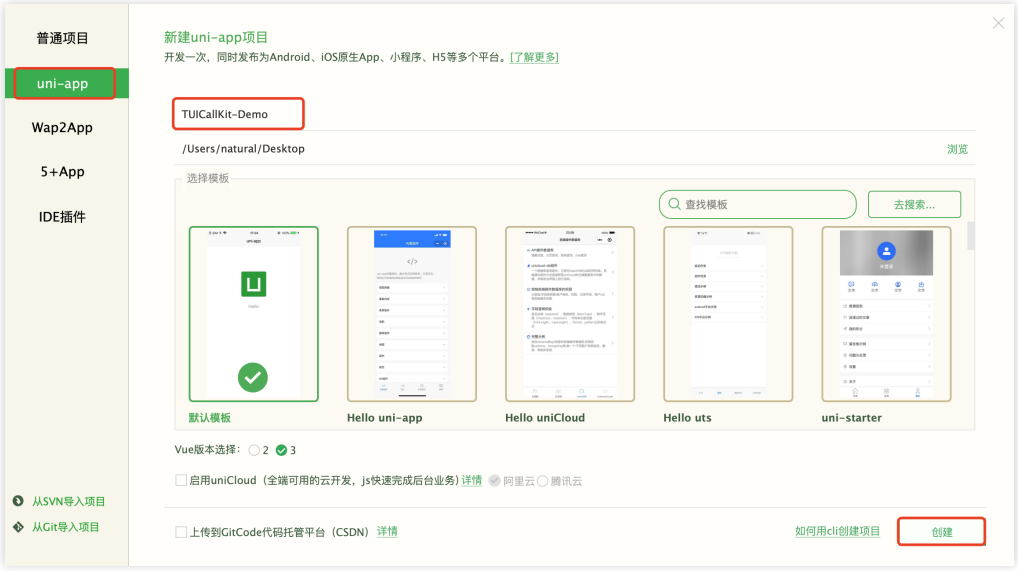
版权所有：腾讯云计算（北京）有限责任公司

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：创建 uni-app 项目

打开 Hbuilderx 开发工具，点击新建 uni-app 项目：项目名称 (TUICallKit-Demo)。



步骤三：下载并导入 TUICallKit 插件

1. 访问 TencentCloud-TUICallKit 插件，在插件详情页中购买插件，购买插件时选择对应的 AppID，绑定正确的包名。

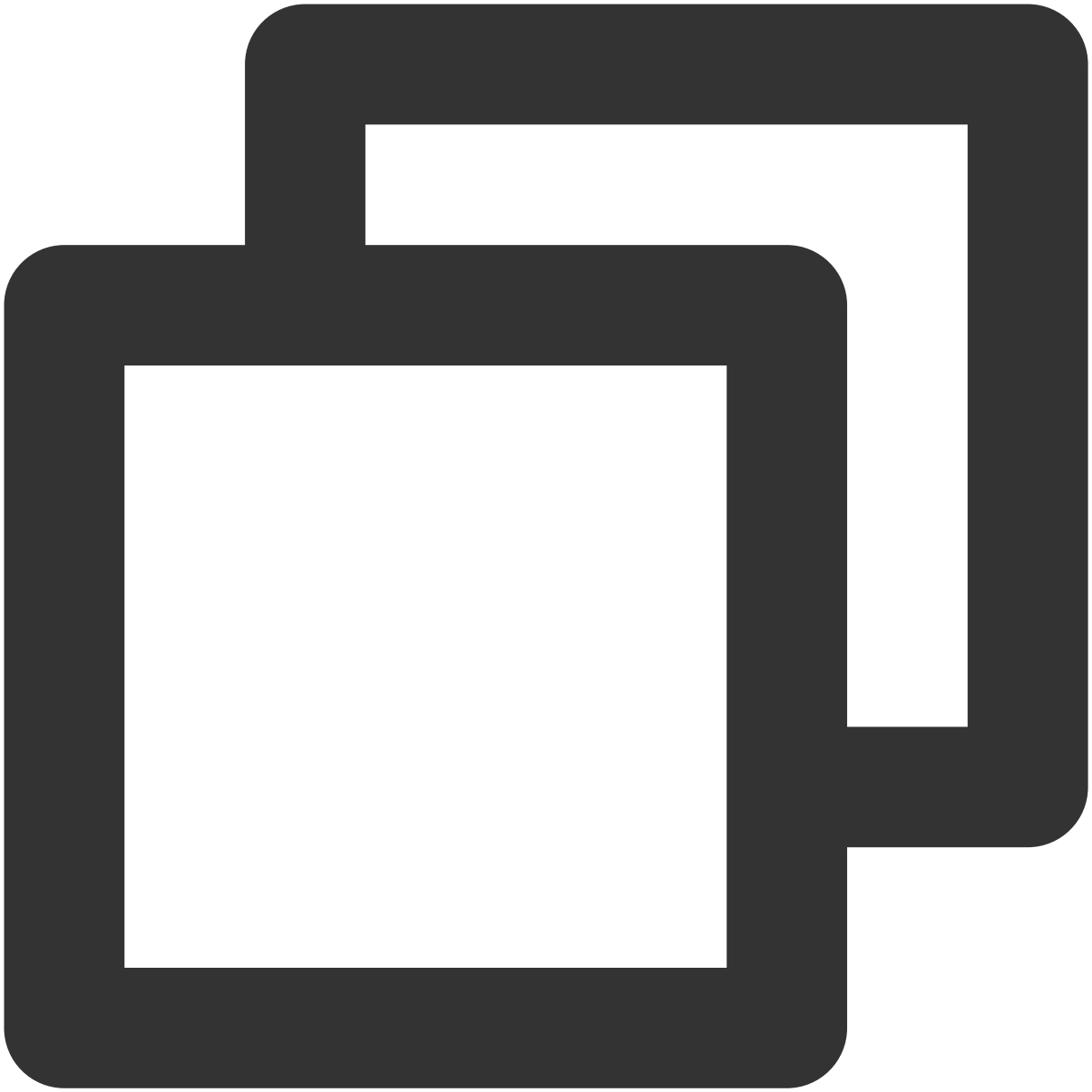


2. 在 `TUICallKit-Demo 项目` 中导入插件。



步骤四：使用 TUICallKit 插件

1. 在 `TUICallKit-Demo/pages/index/index.vue` 中引入下面代码。



```
<template>
        <view class="container">
                <input type="text" v-model="inputID" :placeholder=" isLogin ? 'plea
                <text v-show="isLogin"> your userID: {{ userID }} </text>
                <button v-show="!isLogin" @click="handleLogin"> Login </button>
                <button v-show="isLogin" @click="handleCall"> start call </button>
        </view>
</template>
<script>
        const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit'); // 【
        uni.$TUICallKit = TUICallKit;
```

```
        import { genTestUserSig } from '../../debug/GenerateTestUserSig.js'
        export default {
                data() {
                        return {
                                inputID: '',
                                isLogin: false,
                                userID: '',
                        }
                },
                methods: {
                        handleLogin() {
                                this.userID = this.inputID;
                                const { userSig, sdkAppID: SDKAppID } = genTestUser
                                const loginParams = { SDKAppID, userID: this.userID
                //【2】Login
                                uni.$TUICallKit.login( loginParams, res => {
                                        if (res.code === 0) {
                                                this.isLogin = true;
                                                this.inputID = '';
                                                console.log('[TUICallKit] l
                                        } else {
                                    console.error('[TUICallKit] login failed, failed messag
                                        }
                                    }
                                );
                        },
                        handleCall() {
                                try {
                                        const callParams = {
                                                userID: this.inputID,
                                                callMediaType: 2,   // 1 -- audio c
                                                callParams: { roomID: 234, strRoomI
                                        };
                            //【3】start 1v1 video call
                                        uni.$TUICallKit.call( callParams, res => {
                                                console.log('[TUICallKit] c
                                        }
                                        );
                                        this.inputID = '';
                                } catch (error) {
                                        console.log('[TUICallKit] call error: ', er
                                }
                        }
                }
        }
</script>
<style>
```

```
.container {
        margin: 30px;
}
.container input {
        height: 50px;
        border: 1px solid;
}
.container button {
        margin-top: 30px;
}
</style>
```

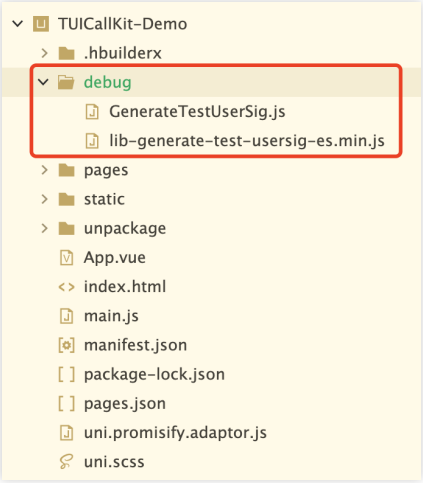
2. 填写 SDKAppID、SDKSecretKey、userSig 参数。

客户端生成 userSig

控制台生成 userSig

由于 UserSig 有时效性，**测试环境下，推荐您采用该方法**。

1. 单击下载 debug 文件夹，将 debug 目录复制到您的项目，如下图所示：

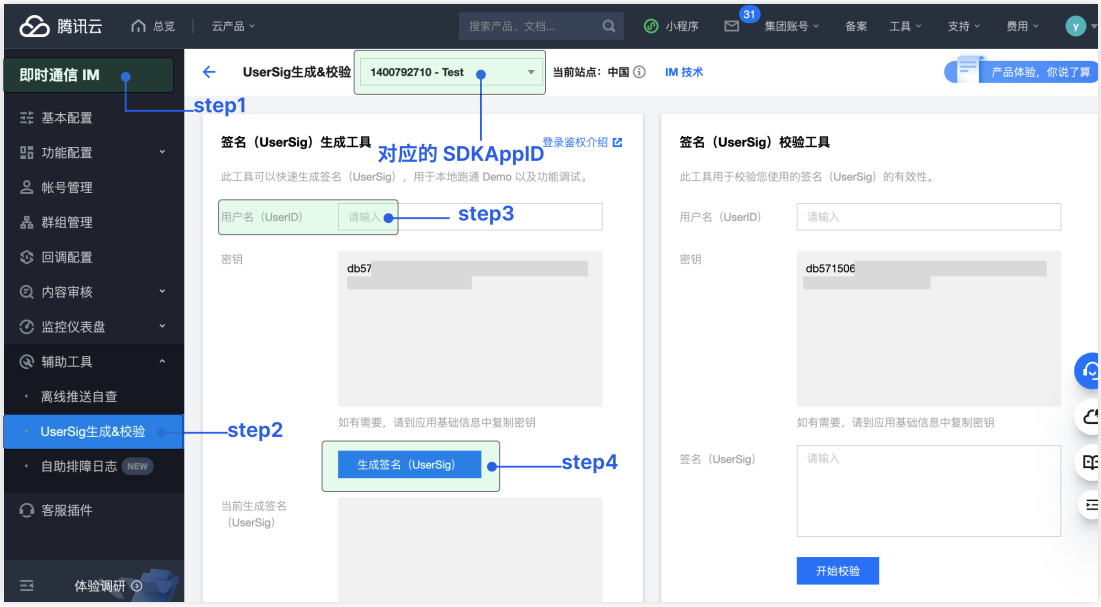


2. 填写 `TUICallKit-Demo/debug/GenerateTestUserSig.js` 文件的 `SDKAppID` 、 `SDKSecretKey` （参考 开通服务）

GenerateTestUserSig.js

```
import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min.js';
/** ... */

const SDKAPPID = 0;
/** ... */

const EXPIRETIME = 604800;
/** ... */

const SECRETKEY = '';
/* ... */

export function genTestUserSig(userID) {
  const generator = new LibGenerateTestUserSig(SDKAPPID, SECRETKEY, EXPIRETIME);
  const userSig = generator.genTestUserSig(userID);

  return {
    sdkAppID: SDKAPPID,
    userSig,
  };
}
```

如果您想要快速体验 TUICallKit，您可以通过控制台中的 **辅助工具** 生成一个临时可用的 UserSig。



如果您采用的是**控制台生成**，那么需要在 `TUICallKit-Demo/pages/index/index.vue` 文件中，赋值这里的 `SDKAppID、userSig` 。



步骤五：拨打您的第一通电话

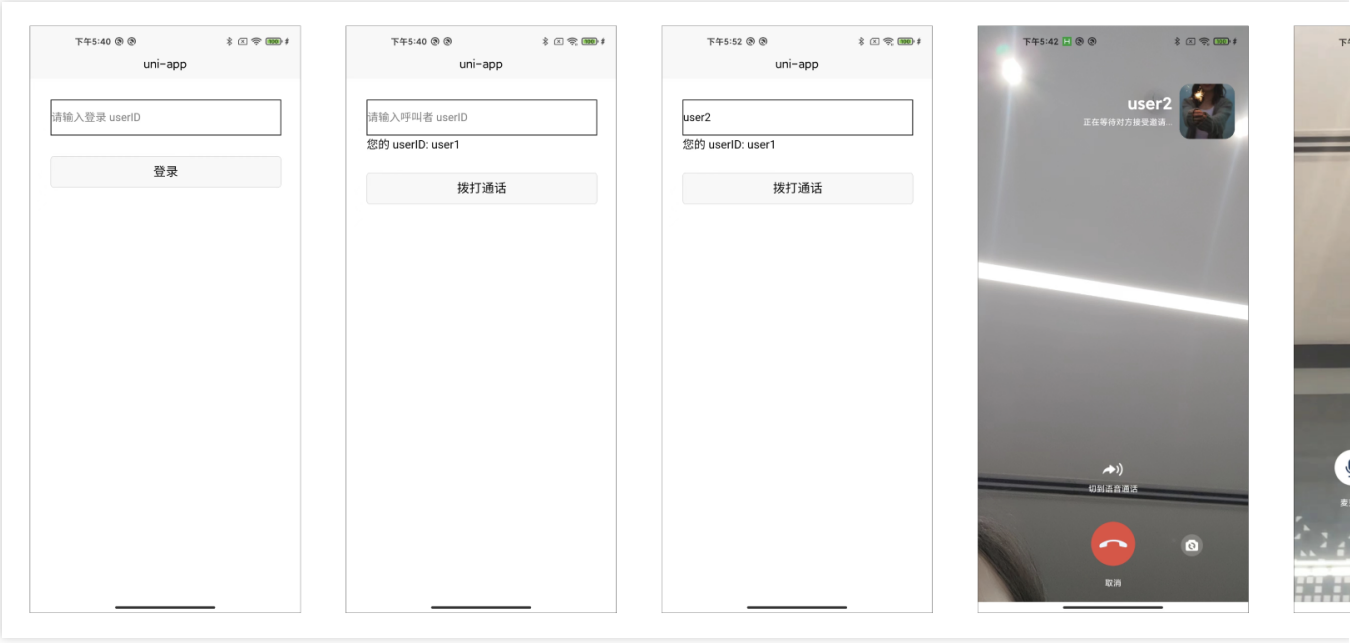
1. 制作自定义调试基座，请选择**传统打包**方式进行打包。



2. 自定义调试基座成功后，**使用自定义基座运行**项目。



3. 拨打 1v1 视频通话具体效果如图所示。



更多特性

设置昵称、头像
群组通话
悬浮窗
自定义铃声
监听通话状态

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。

相关案例-在线客服场景

我们提供了**在线客服场景**的相关源码，建议您 下载在线客服场景 并集成体验。该场景提供了示例客服群 + 示例好友的基础模板，实现功能包括：

支持发送文本消息、图片消息、语音消息、视频消息等常见消息。

支持双人语音、视频通话功能。

支持创建群聊会话、群成员管理等。



技术咨询

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

界面定制（TUICallKit）

Android

最近更新时间：2022-11-21 16:34:34

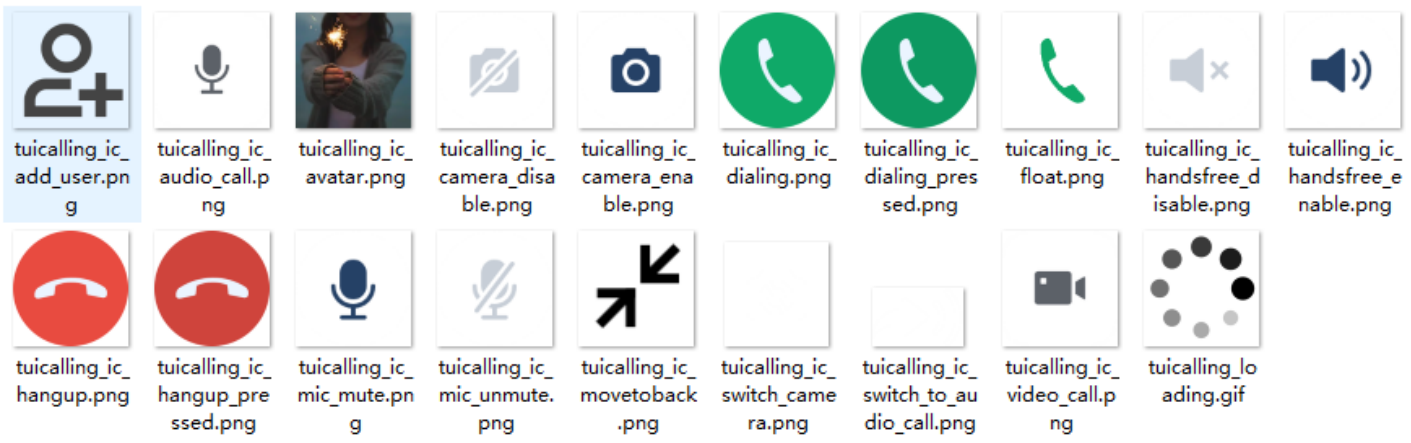
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 Github 中的 `Android/tuicallkit` 文件夹下面：

替换图标

您可以直接替换 `res\drawable-xxhdpi` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 `res\raw` 文件夹下的三个音频文件来达到替换铃声的目的：

文件名	用途
phone_dialing.mp3	发起呼叫时的声音
phone_hangup.mp3	被挂断的声音
phone_ringing.mp3	接到呼叫时的声音

替换文案

您可以通过修改 values-zh 和 values-en 中的 `strings.xml` 文件来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以删掉 tuicallkit 文件夹，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流

版权所有：腾讯云计算（北京）有限责任公司

API	描述
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话的媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调

版权所有：腾讯云计算（北京）有限责任公司

API	描述
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调

关键类型定义

API	描述
TUICallDefine.MediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallDefine.Role	通话的角色，枚举类型：主叫、被叫
TUICallDefine.Status	通话的状态，枚举类型：空闲、待接听、接听中
TUICommonDefine.RoomId	音视频房间 ID，支持数字、字符串两种类型
TUICommonDefine.Camera	摄像头 ID 参数，枚举类型：前摄、后摄
TUICommonDefine.AudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUICommonDefine.NetworkQualityInfo	当前的网络质量信息

iOS

最近更新时间：2023-06-21 16:19:34

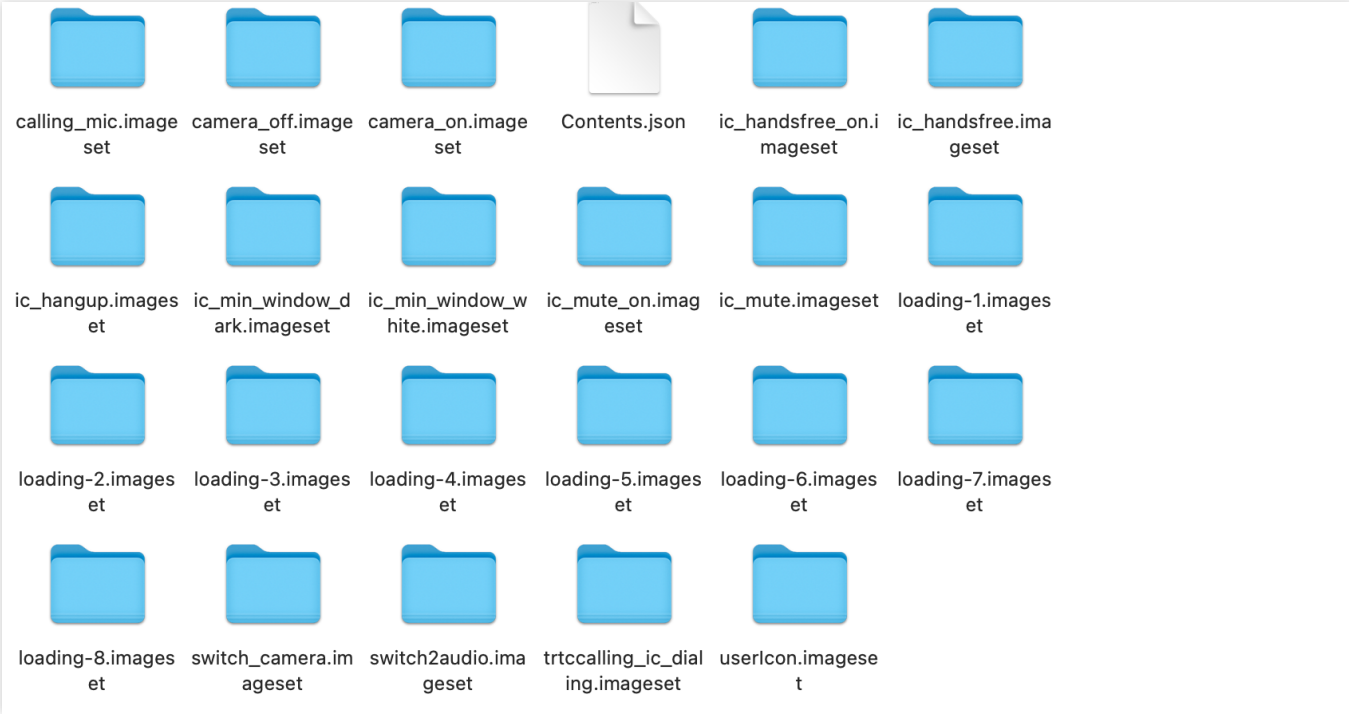
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 Github 中的 `iOS/TUICallKit` 文件夹下面：

替换图标

您可以直接替换 `Resources\\Calling.xcassets` 文件夹下的图标，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 `Resources\\AudioFile` 文件夹下的三个音频文件来达到替换铃声的目的。

文件名	用途
phone_dialing.m4a	发起呼叫时的声音
phone_hangup.mp3	被挂断的声音
phone_ringing.flac	接到呼叫时的声音

版权所有：腾讯云计算（北京）有限责任公司

替换文案

您可以通过修改 zh-Hans.lproj 和 en.lproj 中的 `CallingLocalized.strings` 文件来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以删掉 tuicallkit 文件夹，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流

版权所有：腾讯云计算（北京）有限责任公司

openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）
setSelfInfo	设置用户的头像、昵称
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话的媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调

版权所有：腾讯云计算（北京）有限责任公司

onUserNetworkQualityChanged	所有用户网络质量的反馈回调

关键类型定义

API	描述
TUICallMediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallRole	通话的角色，枚举类型：主叫、被叫
TUICallStatus	通话的状态，枚举类型：空闲、待接听、接听中
TUIRoomId	音视频房间 ID，支持数字、字符串两种类型
TUICamera	摄像头 ID 参数，枚举类型：前摄、后摄
TUIAudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUINetworkQuality	当前的网络质量信息

Web

最近更新时间：2022-12-15 17:13:11

TUICallKit 界面定制指引

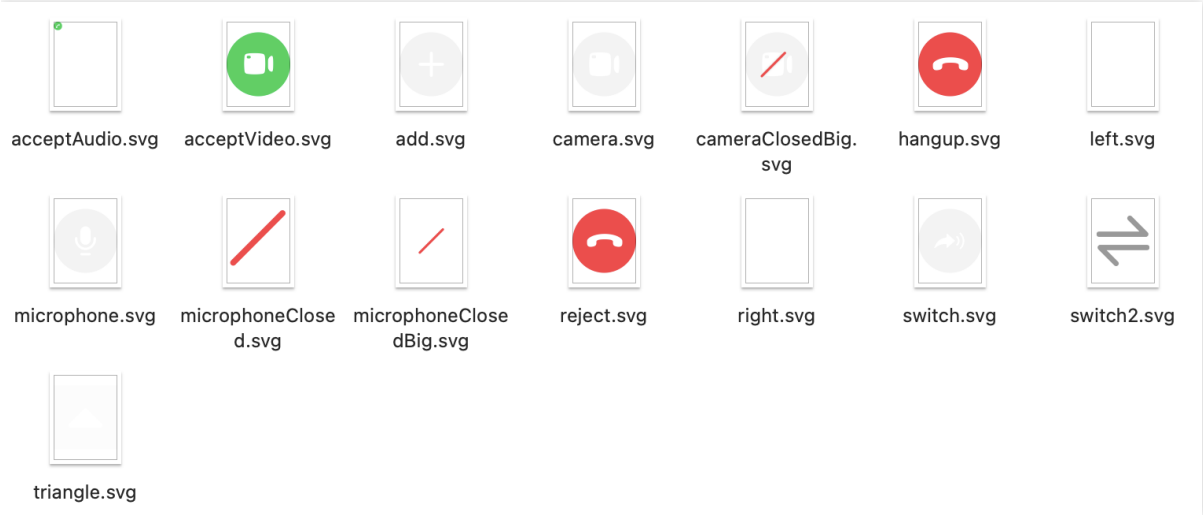
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：界面微调方案和自实现 UI 方案。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 Github 中的 `Web/` 文件夹下面：

替换图标

您可以直接修改 `src/icons` 文件夹下的图标组件，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变，图标预览可参考 `src/assets` 。



调整UI布局

您可以通过修改 src/components/ 文件下的不同页面来修改音视频通话界面

```
- components/
- Calling-C2CVideo.vue 双人视频通话
- Calling-Group.Vue 多人音频、视频通话
- ControlPanel.vue 控制面板
```

```
- ControlPanelItem.vue 控制面板子项
- Dialing.vue 拨打电话页面、来电页面、双人音频通话
- MicrophoneIcon.vue 可显示音量变化的麦克风Icon
- TUICallKit.vue TUICallKit 总组件
```

修改样式

样式文件在 `src/style.css` ，根据不同的 UI 样式，可以进行自行调整

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI SDK实现的，您可以完全基于 TUICallEngine 实现一套自己的 UI 界面。详情可见

- TUICallEngine 接入指引
- TUICallEngine API 接口地址

Flutter

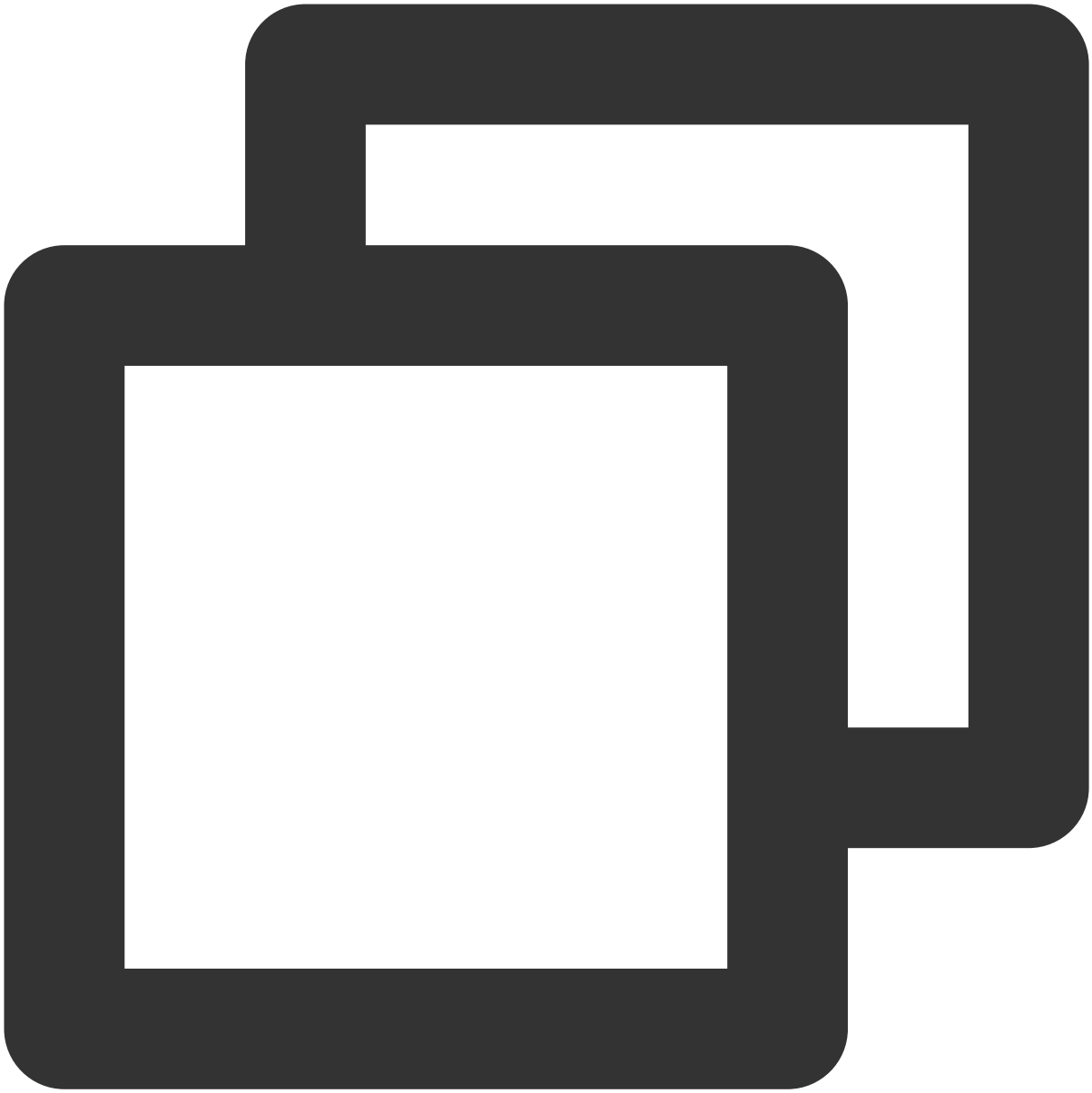
最近更新时间：2024-03-12 14:51:07

本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。
注意：页面定制方案需要使用 tencent_calls_uikit 插件 1.8.0 以上的版本。

方案一：界面微调方案

你可以下载 tencent_calls_uikit 插件的最新版本到本地，之后在你的工程中使用本地依赖的方式接入插件，本地依赖方式如下：
在工程 **pubspec.yaml** 文件中的 **dependencies** 节点下，添加 **tencent_calls_uikit** 插件依赖，示例如下：

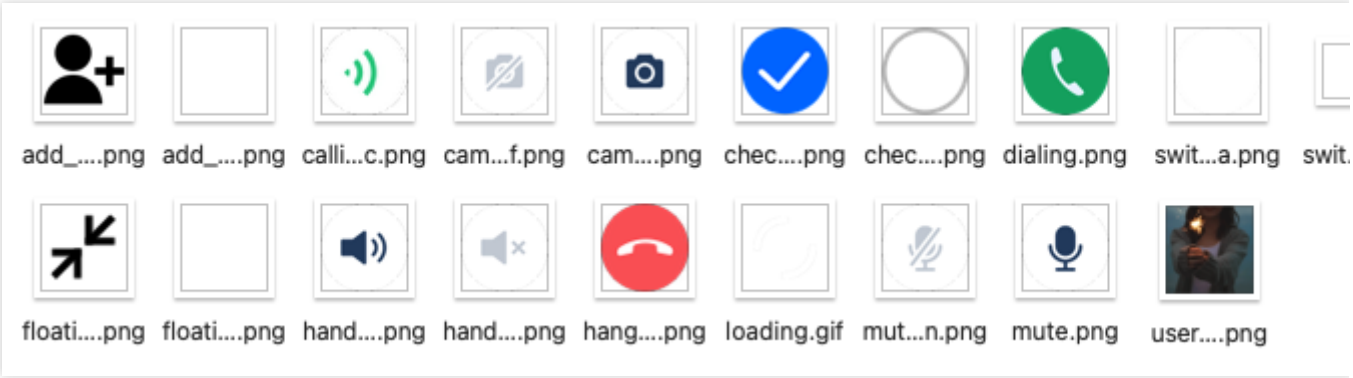
版权所有：腾讯云计算（北京）有限责任公司



```
dependencies:
    tencent_calls_uikit:
      path: 插件目录
```

替换图标

您可以直接替换 `assets\\images` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 **assets\\audios** 文件夹下的三个音频文件来达到替换铃声的目的：

文件名	用途
phone_dialing.mp3	发起呼叫时的声音
phone_hangup.mp3	被挂断的声音
phone_ringing.mp3	接到呼叫时的声音

替换文案

您可以通过修改 **lib\\src\\i18n** 目录下 **strings.g.dart** 文件中的的字符串来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以不依赖 **tencent_calls_uikit** 插件，而直接依赖 tencent_calls_engine 插件，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，收取自己感兴趣的回调事件。

离线唤醒（TUICallKit）

iOS

VoIP

最近更新时间：2024-01-17 11:58:53

VoIP（Voice over IP）Push 是 Apple 提供的用于响应 VoIP calls 的通知机制。结合 Apple 提供的 PushKit.framework 和 CallKit.framework 能够实现系统级的通话效果。目前 VoIP Push 暂时只支持离线状态下推送。

说明：

Apple 要求，VoIP Push 在 iOS 13.0 以后需要搭配 CallKit.framework 使用，否则 App 运行后会 crash。

VoIP Push 无法复用 APNs 普通推送证书，需要单独在苹果开发者网站上 申请 VoIP Push 证书。

配置 VoIP Push

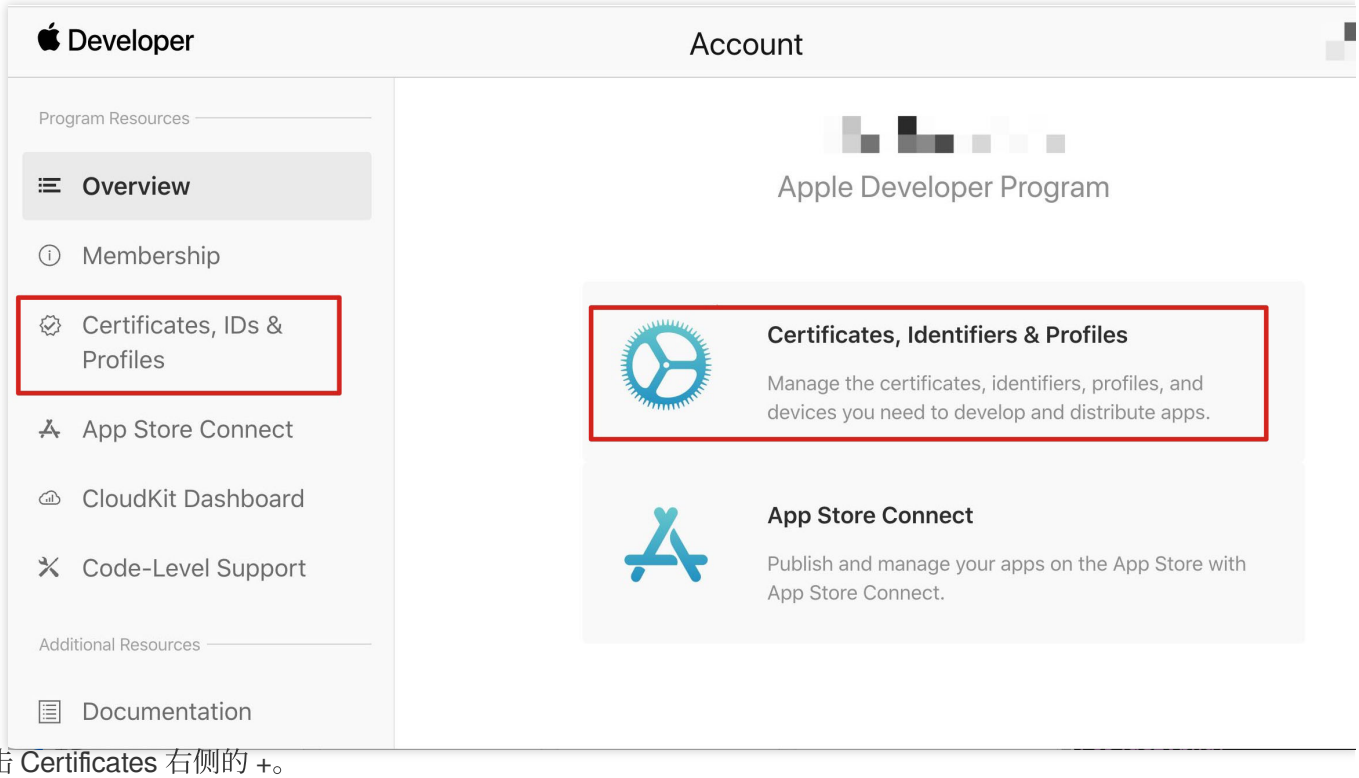
如果想要接收 VoIP Push，需要遵循如下几个步骤：

1. 申请 VoIP Push 证书。
2. 上传证书到 IM 控制台。
3. 完成工程配置。
4. 接入 TUICallKitVoIPExtension 组件。

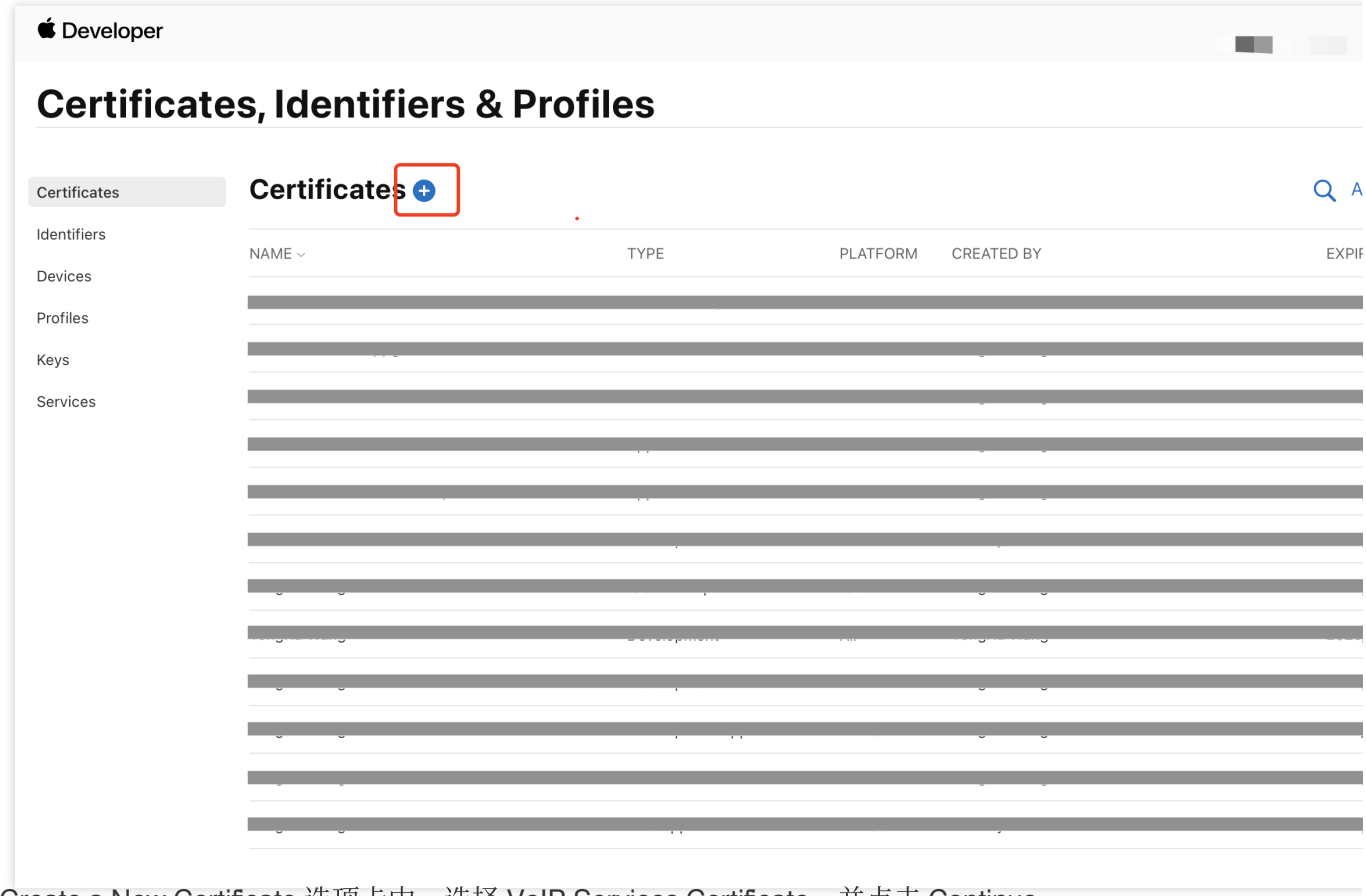
步骤1：申请 VoIP Push 证书

在申请 VoIP Push 证书之前，请先登录 苹果开发者中心 网站， 开启 App 的远程推送功能。当您的 AppID 具备了 Push Notification 能力后，按照如下步骤申请并配置 VoIP Push 证书：

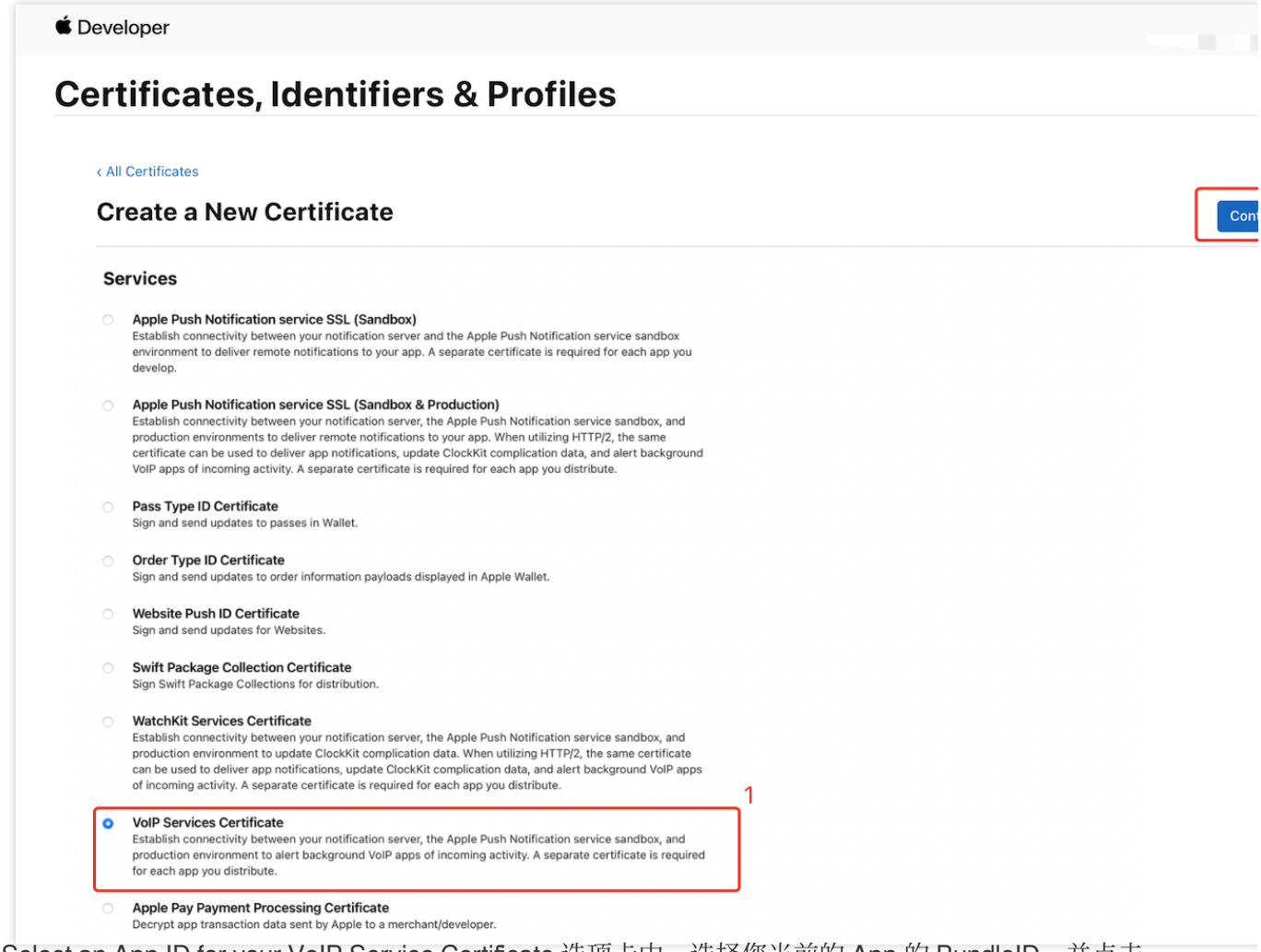
1. 登录 苹果开发者中心 网站，单击 Certificates,Identifiers & Profiles 或者侧栏的 Certificates, IDS & Profiles，进入 Certificates, IDS & Profiles 页面。



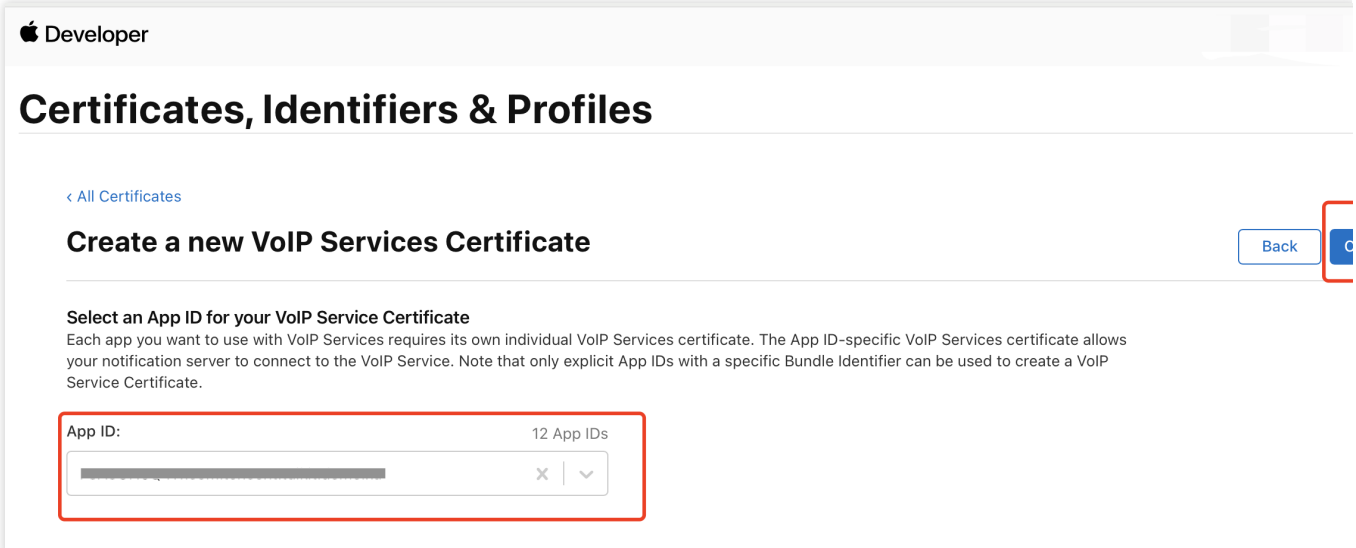
2. 单击 Certificates 右侧的 +。



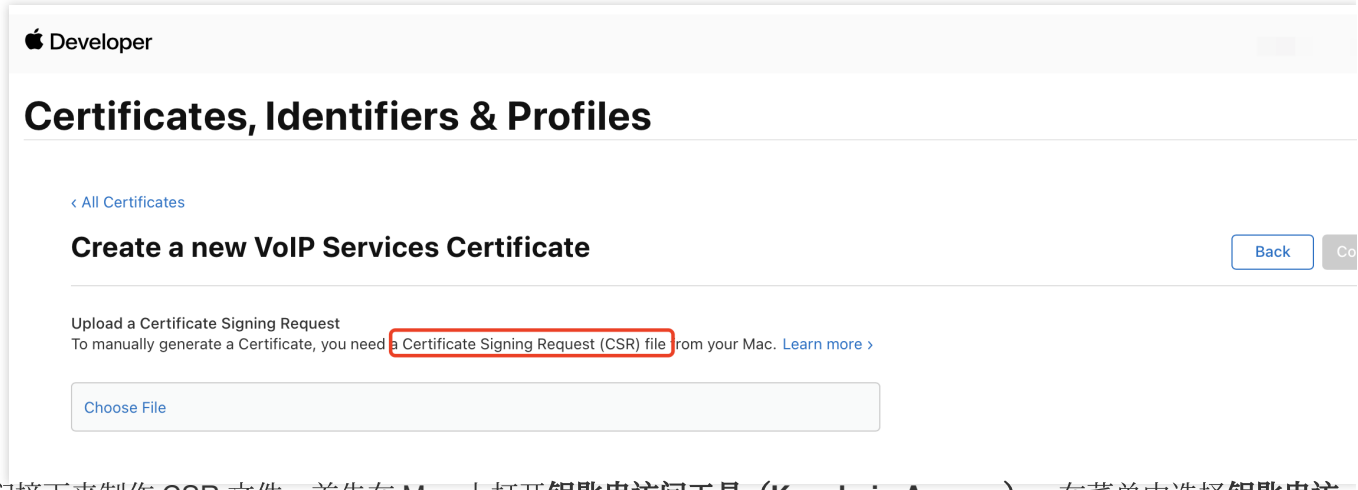
3. 在 Create a New Certificate 选项卡中，选择 VoIP Services Certificate，并点击 Continue。



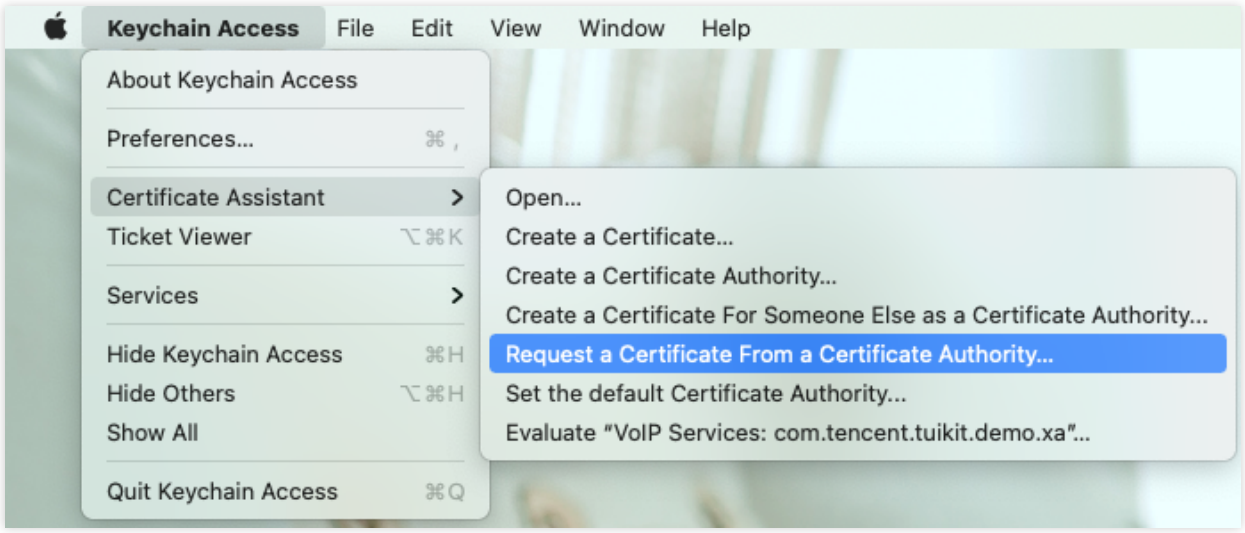
4. 在 Select an App ID for your VoIP Service Certificate 选项卡中，选择您当前的 App 的 BundleID，并点击 Continue。



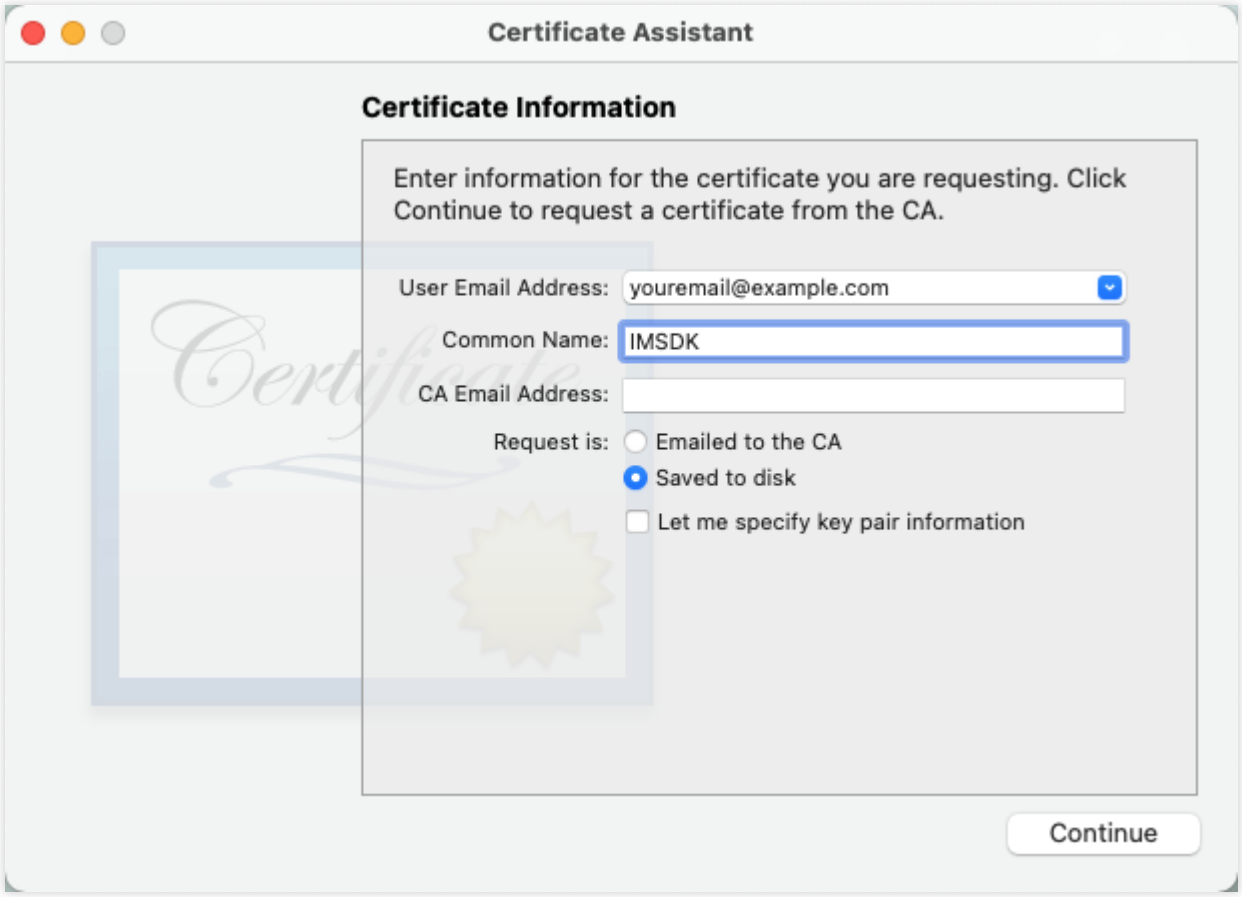
5. 紧接着，系统提示我们需要一个 Certificate Signing Request（CSR）。



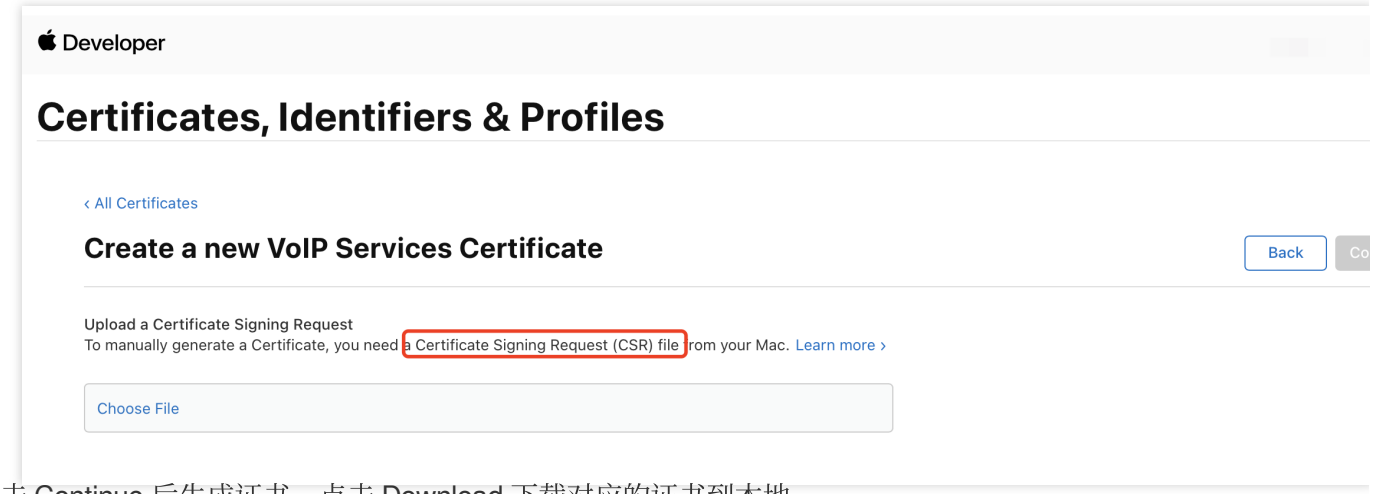
6. 我们接下来制作 CSR 文件。首先在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



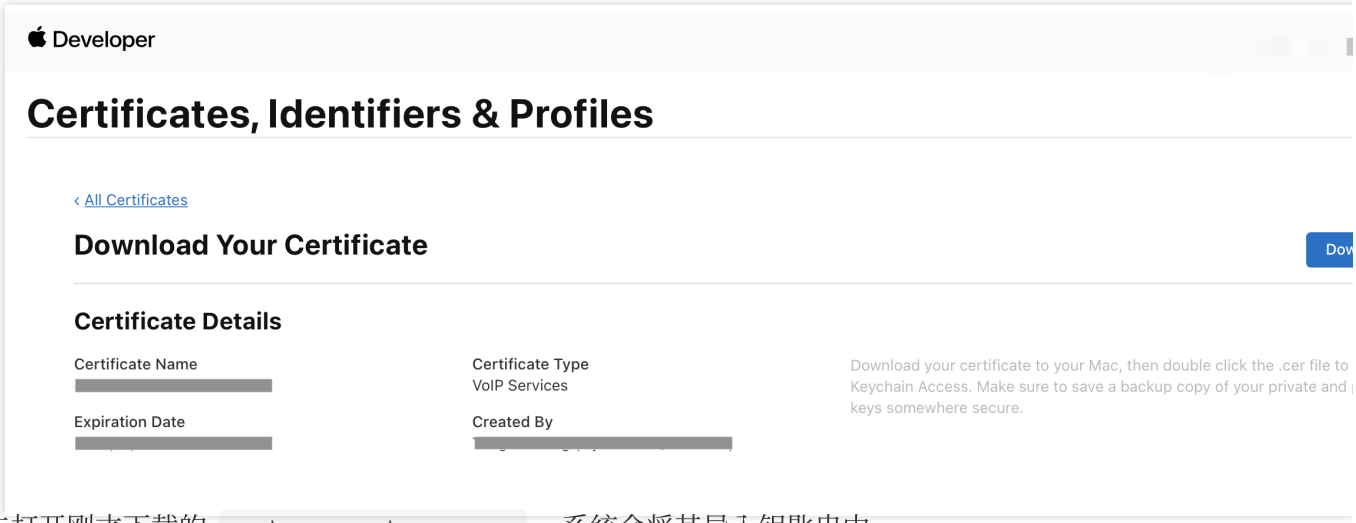
7. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击继续，系统将生成一个 `*.certSigningRequest` 文件。



返回上述**步骤5**中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。



8. 单击 Continue 后生成证书，点击 Download 下载对应的证书到本地。

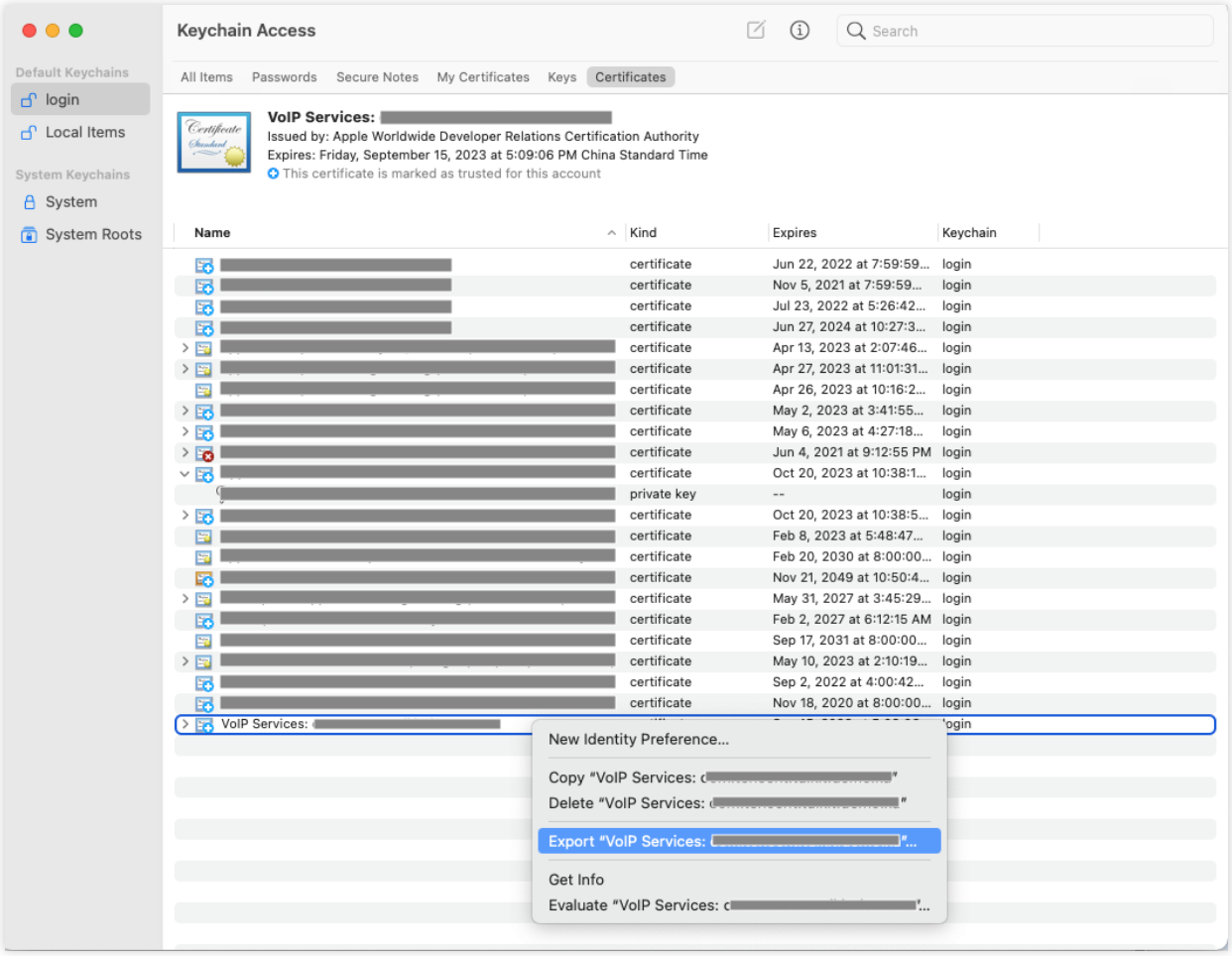


9. 双击打开刚才下载的 `voip_services.cer`，系统会将其导入钥匙串中。

10. 打开钥匙串应用，在**登录** > **我的证书**，右键导出刚创建的 VoIP Services 的 `P12` 文件。

说明：

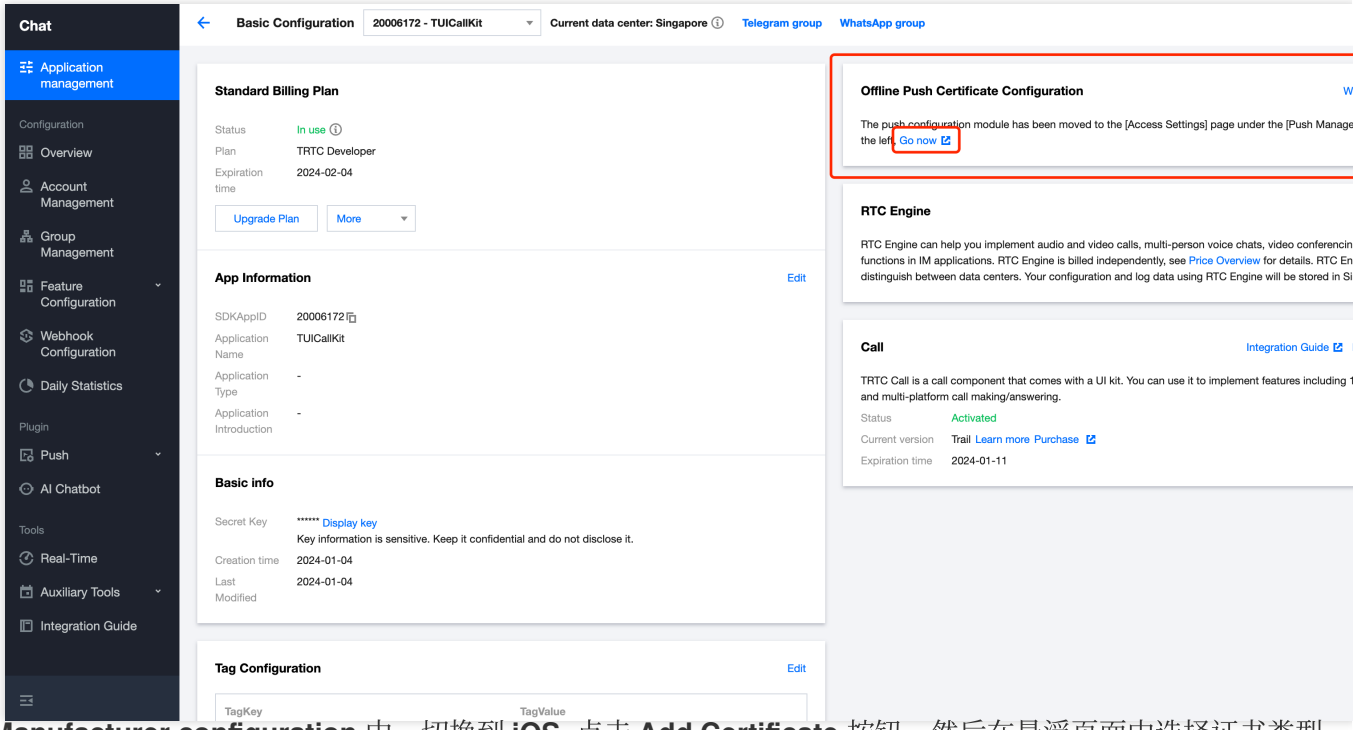
保存 `P12` 文件时，请务必要为其设置密码。



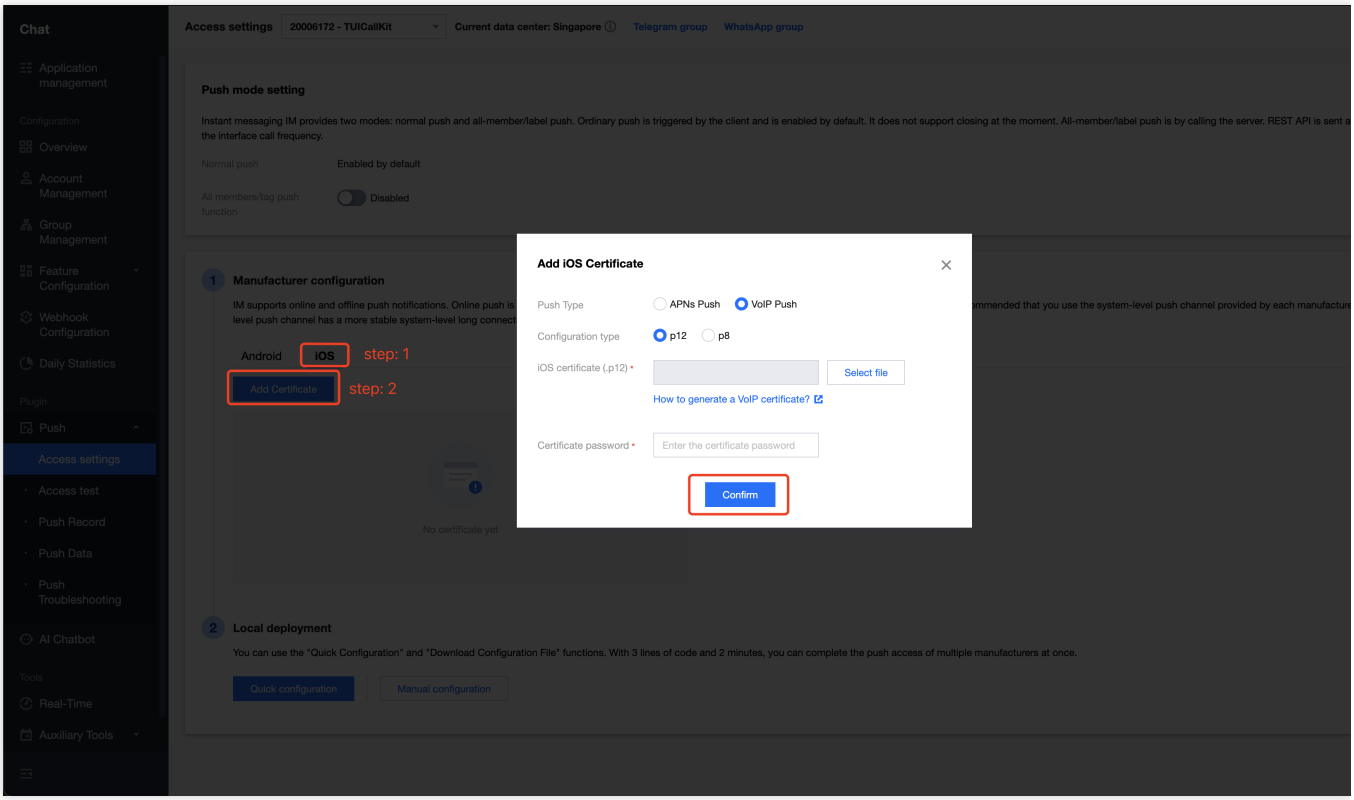
步骤2：上传证书到 IM 控制台

打开 IM 控制台，选择您创建的 IM 应用，并按照如下步骤上传证书：

1. 选择您的 IM 应用，进入应用的基础配置页面，在 **Offline Push Certificate Configuration** 选项卡中，点击 **Go now**。



2. 在 **Manufacturer configuration** 中，切换到 **iOS**，点击 **Add Certificate** 按钮，然后在悬浮页面中选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击 **Confirm**。



说明：

VoIP Push 证书本身不区分生产环境和测试环境，生产环境和开发环境使用的是同一份 VoIP Push 证书。

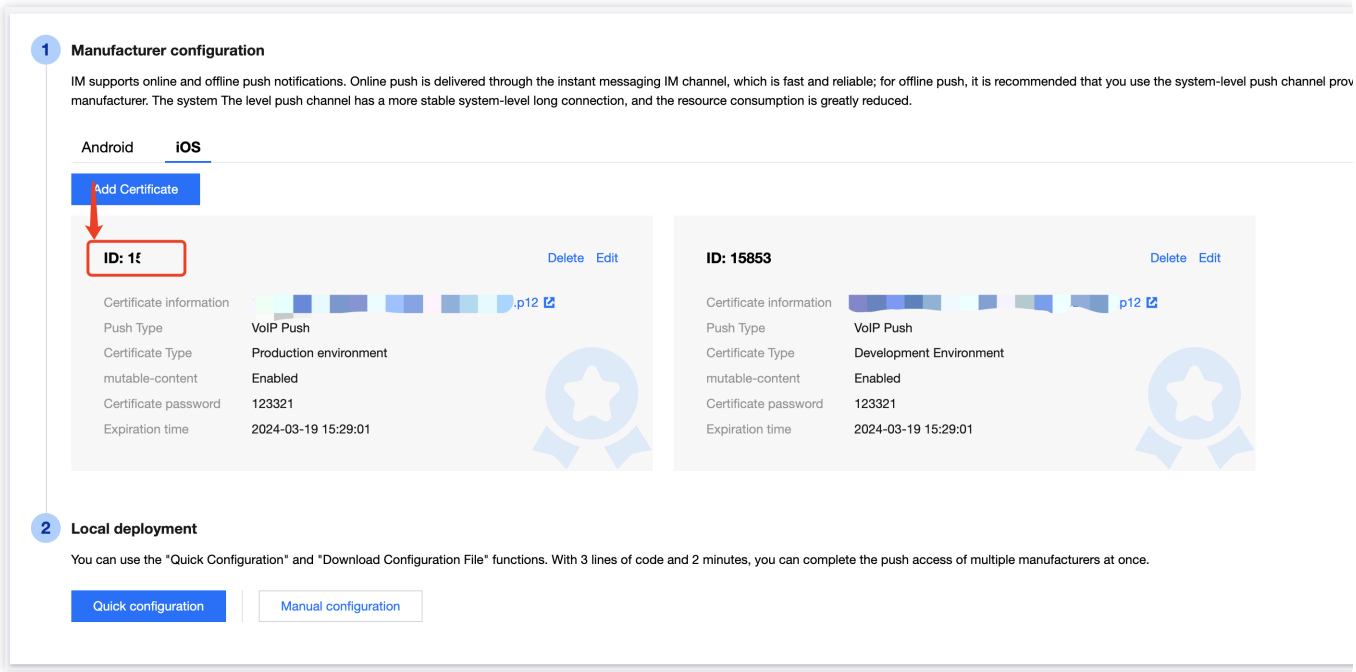
上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

上传证书需要设置密码，无密码收不到推送。

发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。

3. 上传完成后，记录不同环境下的证书 ID。

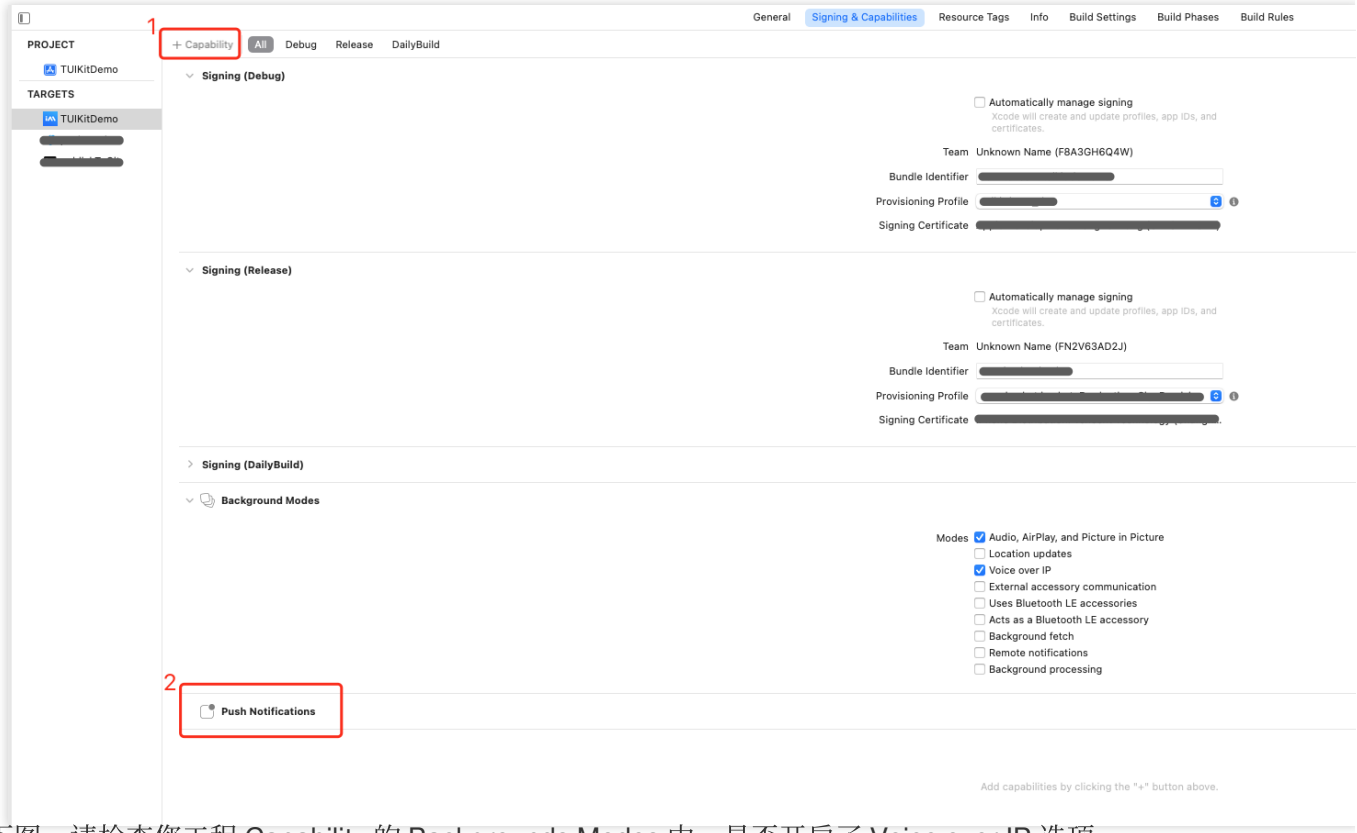


说明：

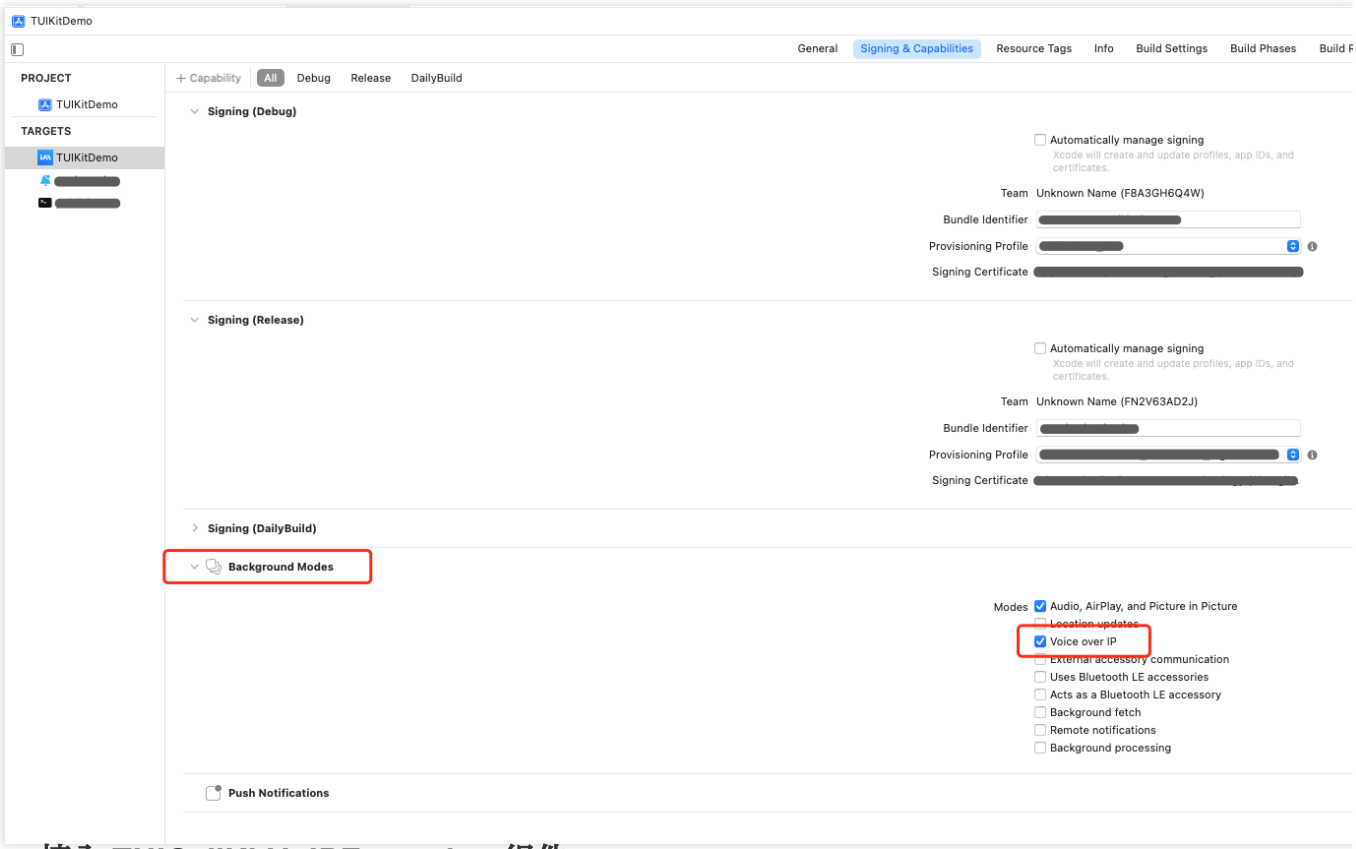
开发环境和生产环境下的证书 ID 要严格区分，并在 步骤4：接入TUICallKitVoIPExtension 组件 时根据实际环境填写。

步骤3：完成工程配置

1. 如下图，确认您工程的 Capability 中是否添加 Push Notification 能力。



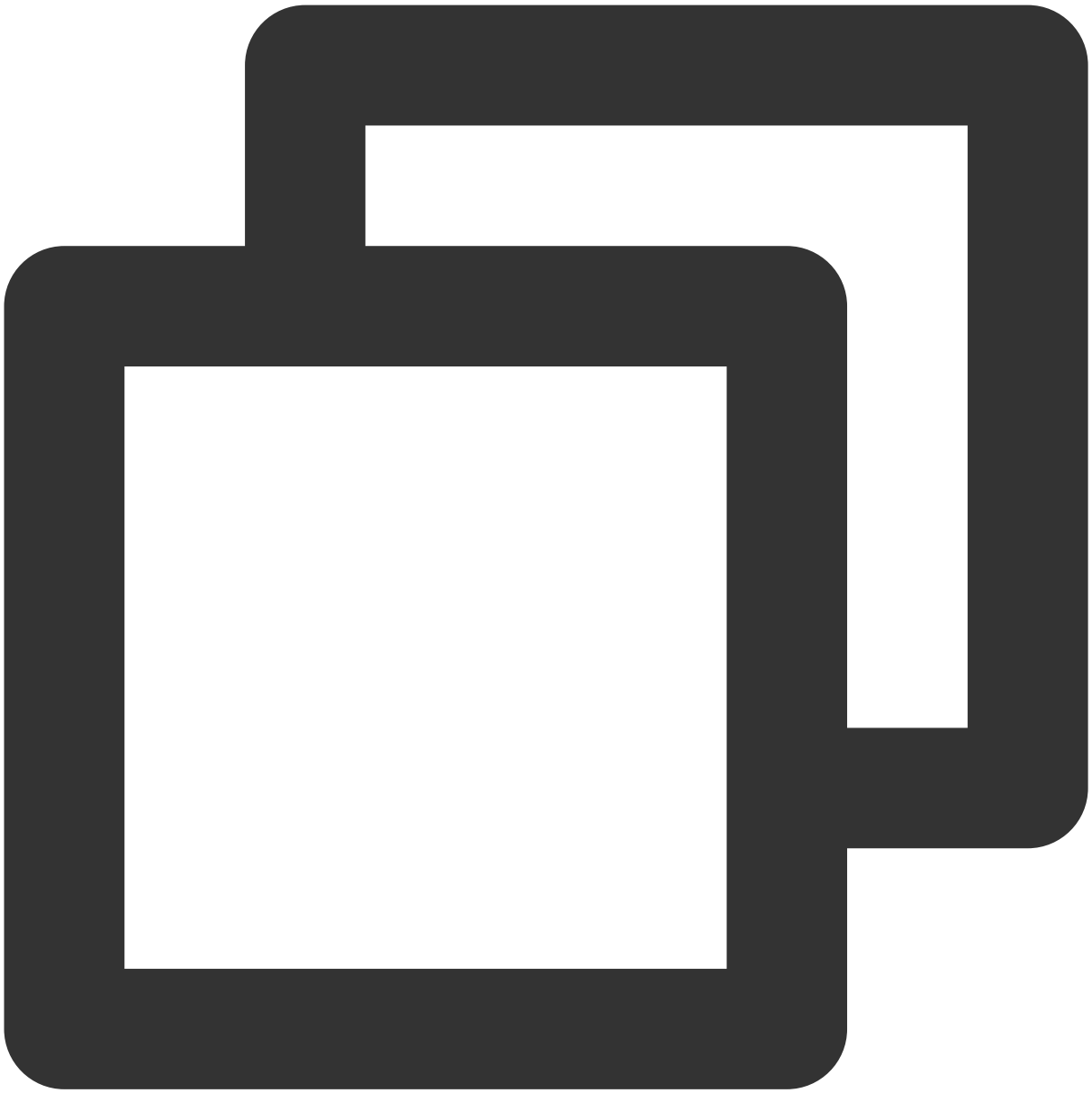
2. 如下图，请检查您工程 Capability 的 Backgrounds Modes 中，是否开启了 Voice over IP 选项。



步骤4：接入 TUICallKitVoIPExtension 组件

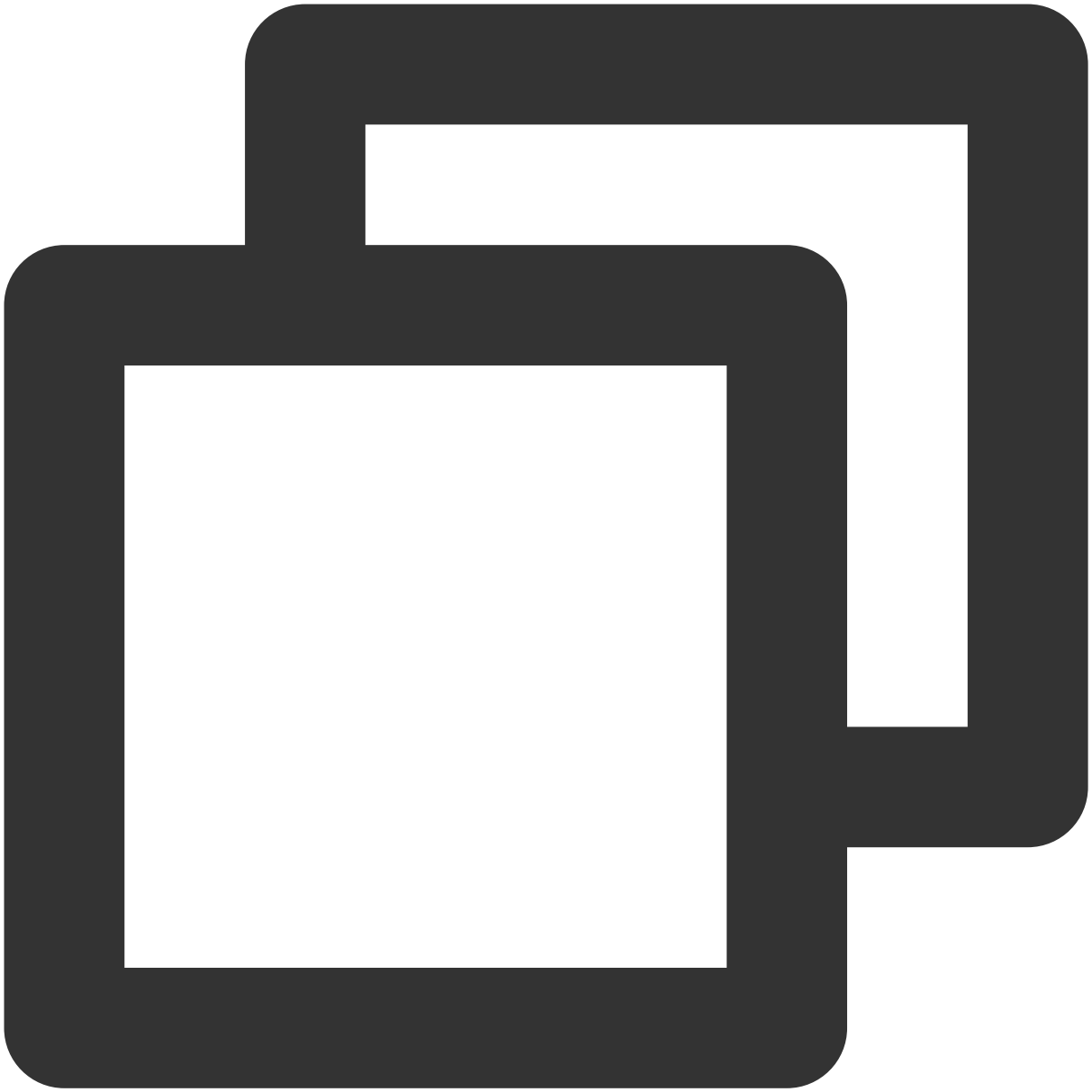
使用 CocoaPods 导入组件，具体步骤如下：

1. 在您的 `Podfile` 文件中添加以下依赖。



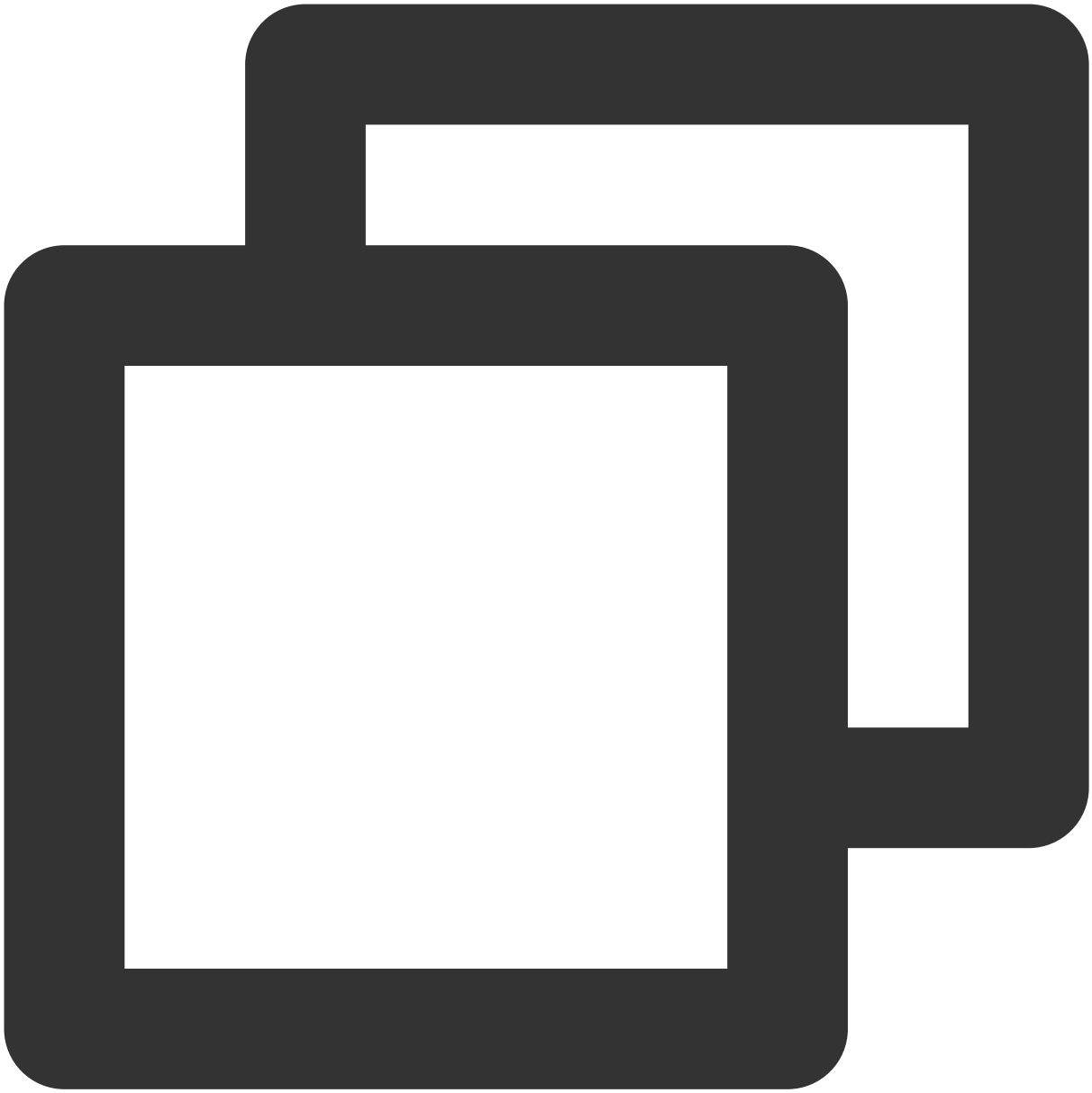
```
pod 'TUICallKitVoIPExtension'
```

2. 执行以下命令，安装组件。



```
pod install
```

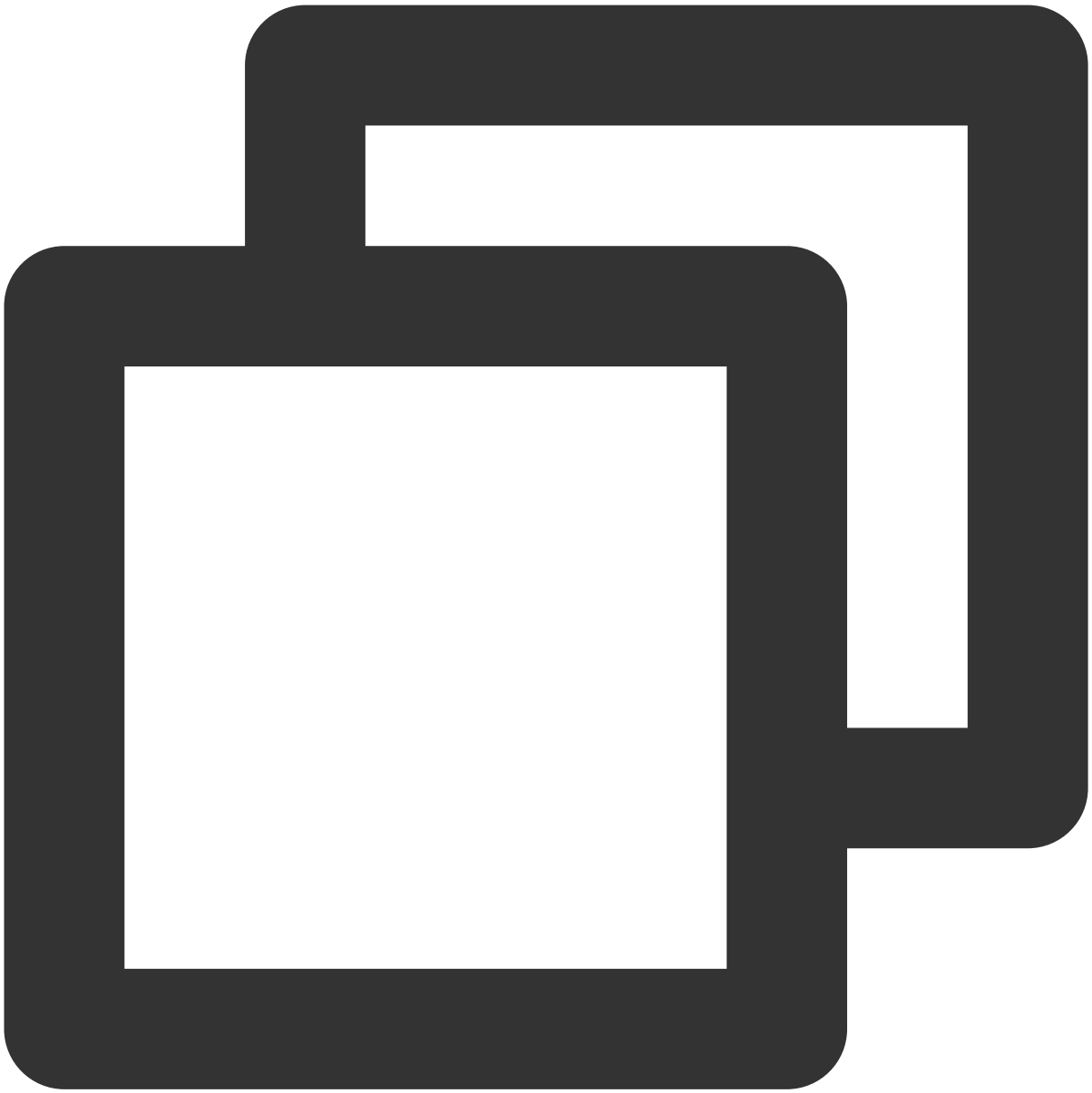
3. 上报 步骤 2：记录的证书 ID 。



```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
        // Override point for customization after application launch.
        // 上报证书 ID
        TUICallKitVoIPExtension.setCertificateID(CertificateID)
        return true
    }
```

说明：

如果无法安装 TUICallKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。



```
pod repo update
```

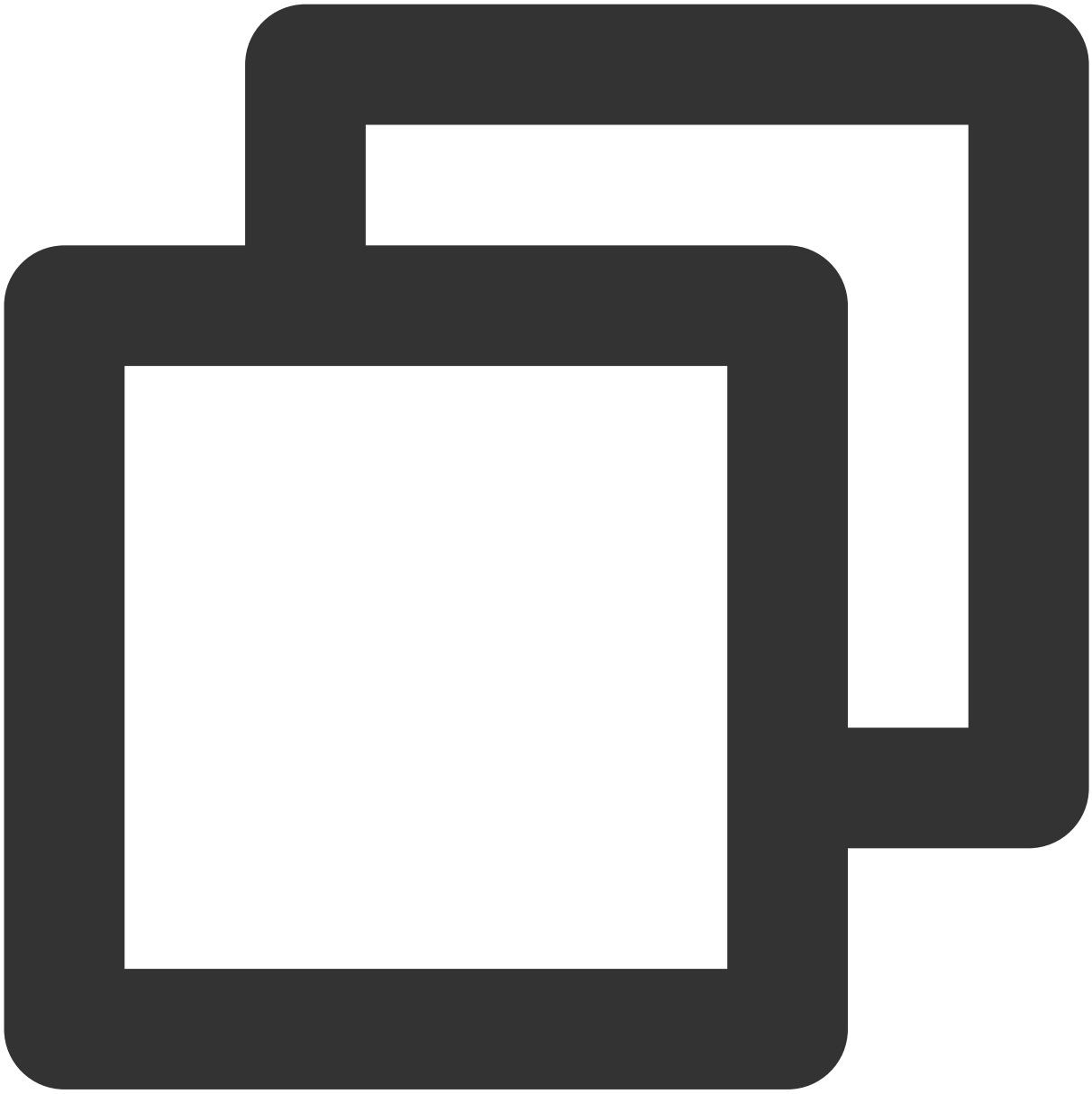
拨打 VoIP 通话

如果您需要拨打 VoIP 通话，需要在调用 call 时候设置 OfflinePushInfo 中的 iOSPushType 字段为 TUICallIOSOfflinePushTypeVoIP，默认为 TUICallIOSOfflinePushTypeAPNs。

Objective-C

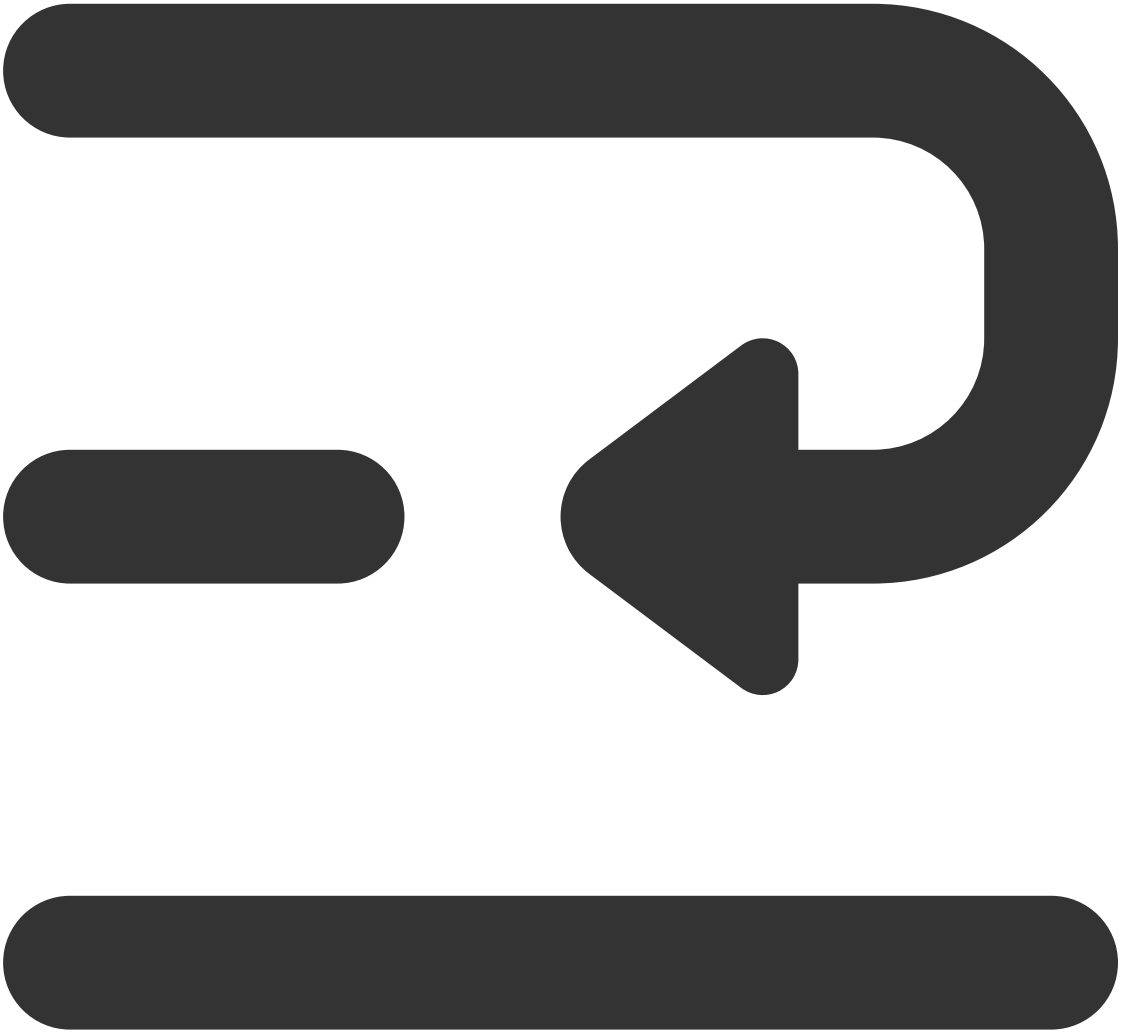
Java

Dart

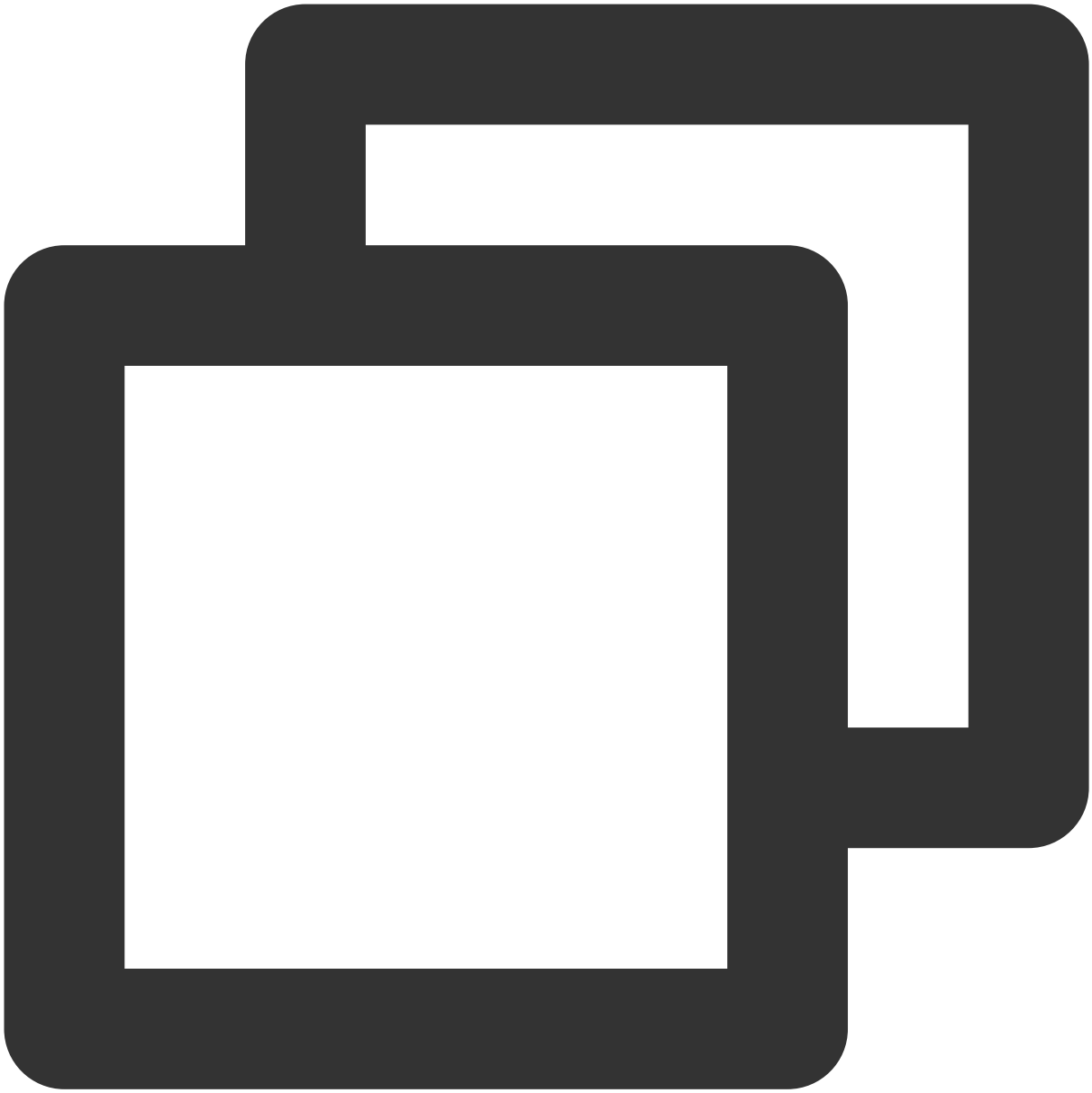


```
- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}
```

```
- (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = @"您有一个新的通话";
    // iOS push type: if you want user VoIP, please modify type to TUICallIOSOfflin
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeVoIP;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChanne
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

[[TUICallKit createInstance] call:@"mike 的 id" params:[self getCallParams] callMedi
```

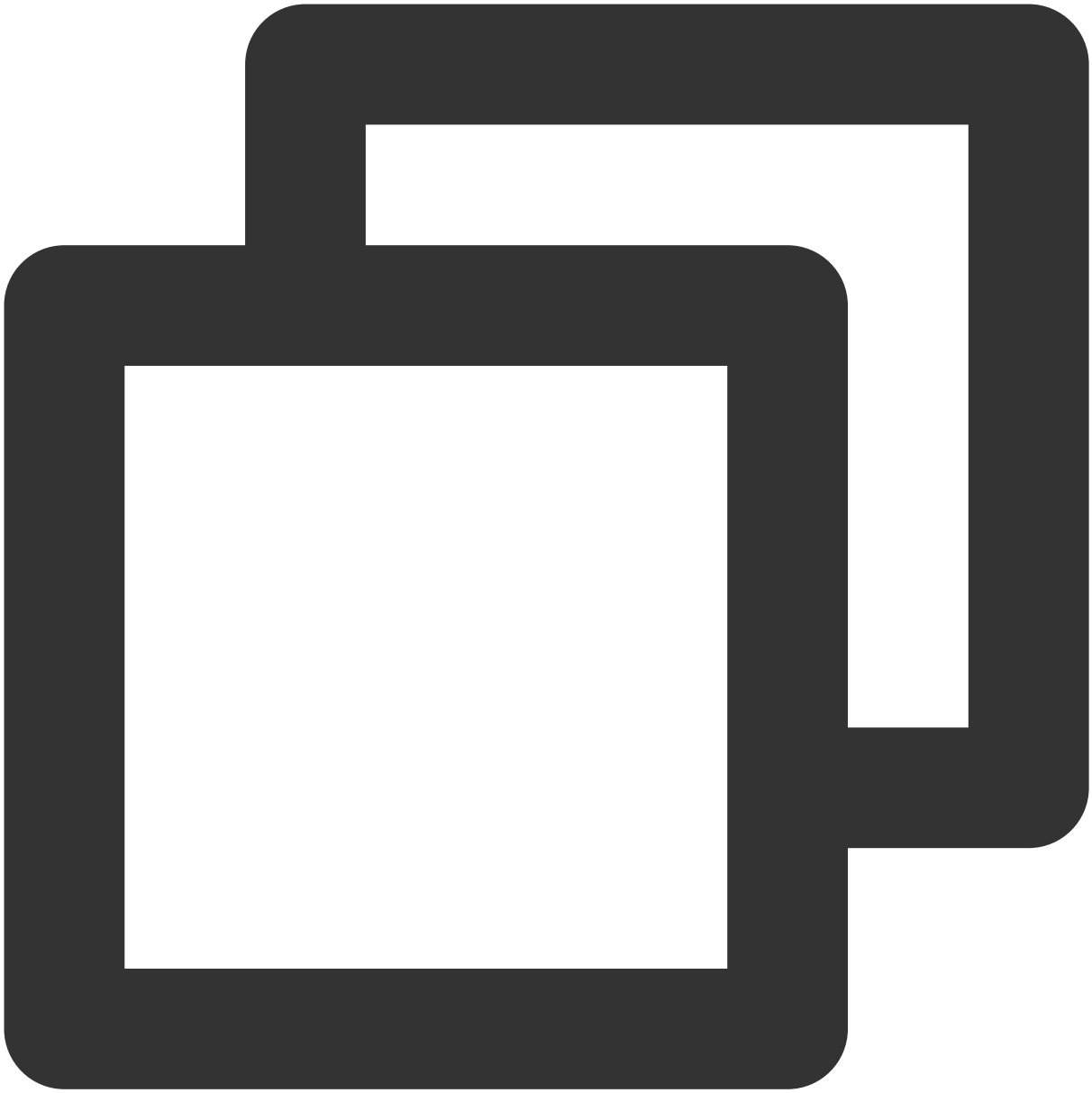


版权所有：腾讯云计算（北京）有限责任公司



```
TUICallDefine.OfflinePushInfo offlinePushInfo = new TUICallDefine.OfflinePushInfo()
offlinePushInfo.setTitle("");
offlinePushInfo.setDesc("You have receive a new call");
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
offlinePushInfo.setAndroidOPPOChannelID("tuikit");
offlinePushInfo.setIgnoreIOSBadge(false);
offlinePushInfo.setIOSSound("phone_ringing.mp3");
offlinePushInfo.setAndroidSound("phone_ringing"); //Note：don't add suffix
//VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
offlinePushInfo.setAndroidVIVOClassification(1);
```

```
//FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
offlinePushInfo.setAndroidFCMChannelID("fcm_push_channel");
//Huawei message type
offlinePushInfo.setAndroidHuaWeiCategory("IM");
//IOS push type: if you want user VoIP, please modify type to TUICallDefine.IOSOffl
offlinePushInfo.setIOSPushType(TUICallDefine.IOSOfflinePushType.VoIP);

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = offlinePushInfo;

TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video, para
```



```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
 offlinePushInfo.title = "Flutter TUICallKit";
 offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
 offlinePushInfo.ignoreIOSBadge = false;
 offlinePushInfo.iOSSound = "phone_ringing.mp3";
 offlinePushInfo.androidSound = "phone_ringing";
 offlinePushInfo.androidOPPOChannelID = "Flutter TUICallKit";
 offlinePushInfo.androidVIVOClassification = 1;
 offlinePushInfo.androidFCMChannelID = "fcm_push_channel";
 offlinePushInfo.androidHuaWeiCategory = "Flutter TUICallKit";
 offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.VoIP;
```

```
TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

TUICallKit.instance.call(callUserId, TUICallMediaType.audio, params);
```

通过系统通话记录拨打通话

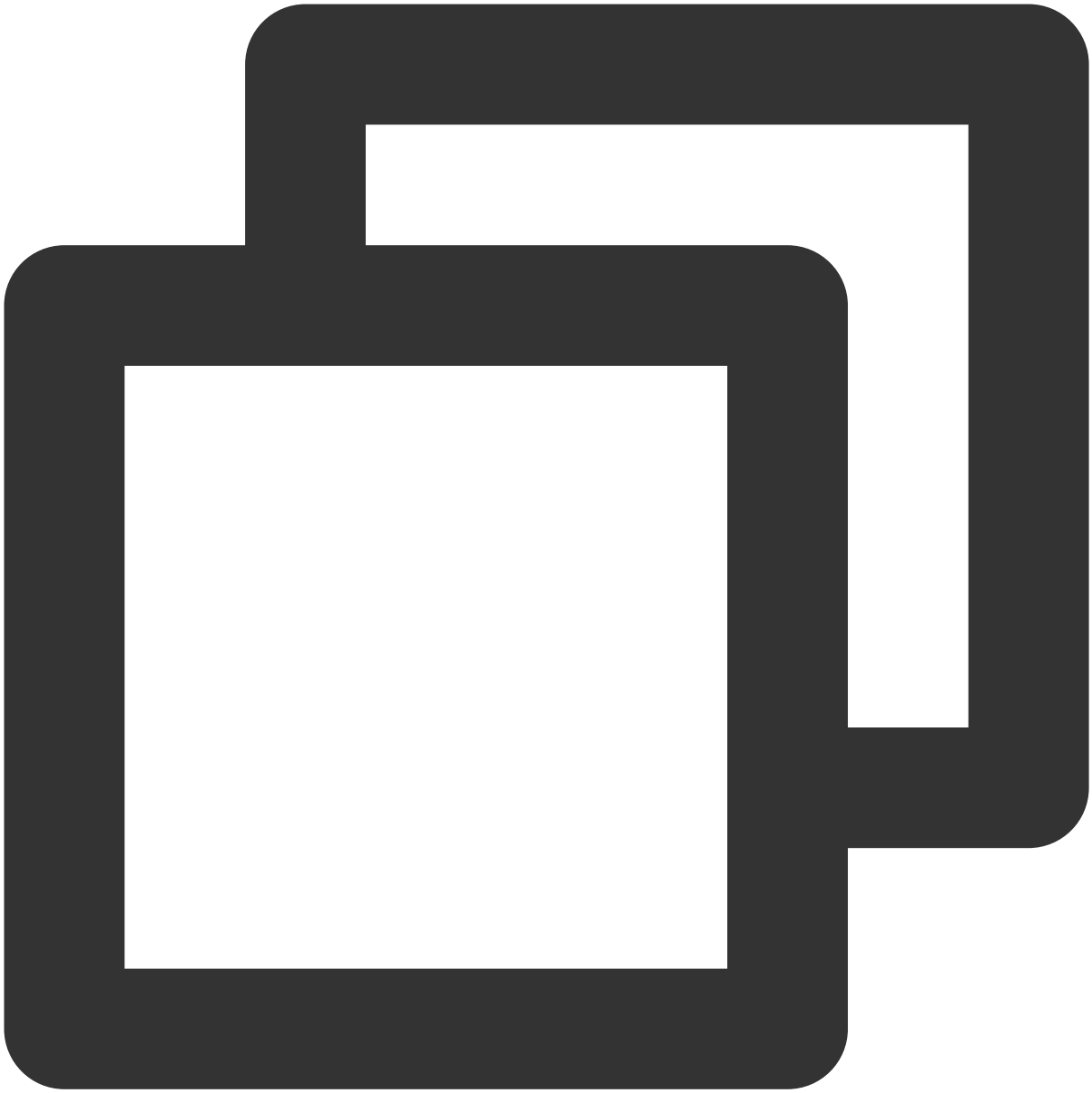
如果需要实现点击**系统电话 - 最近通话**列表中的通话记录，直接发起单人音视频通话，您需要在 Application 的生命周期回调函数使用 **TUICallKitVoIPExtension** 组件中的 **callWith** 接口，具体示例如下：

说明：

仅支持单人音视频通话的直接拨打。

登录账户必须为同一个账户。

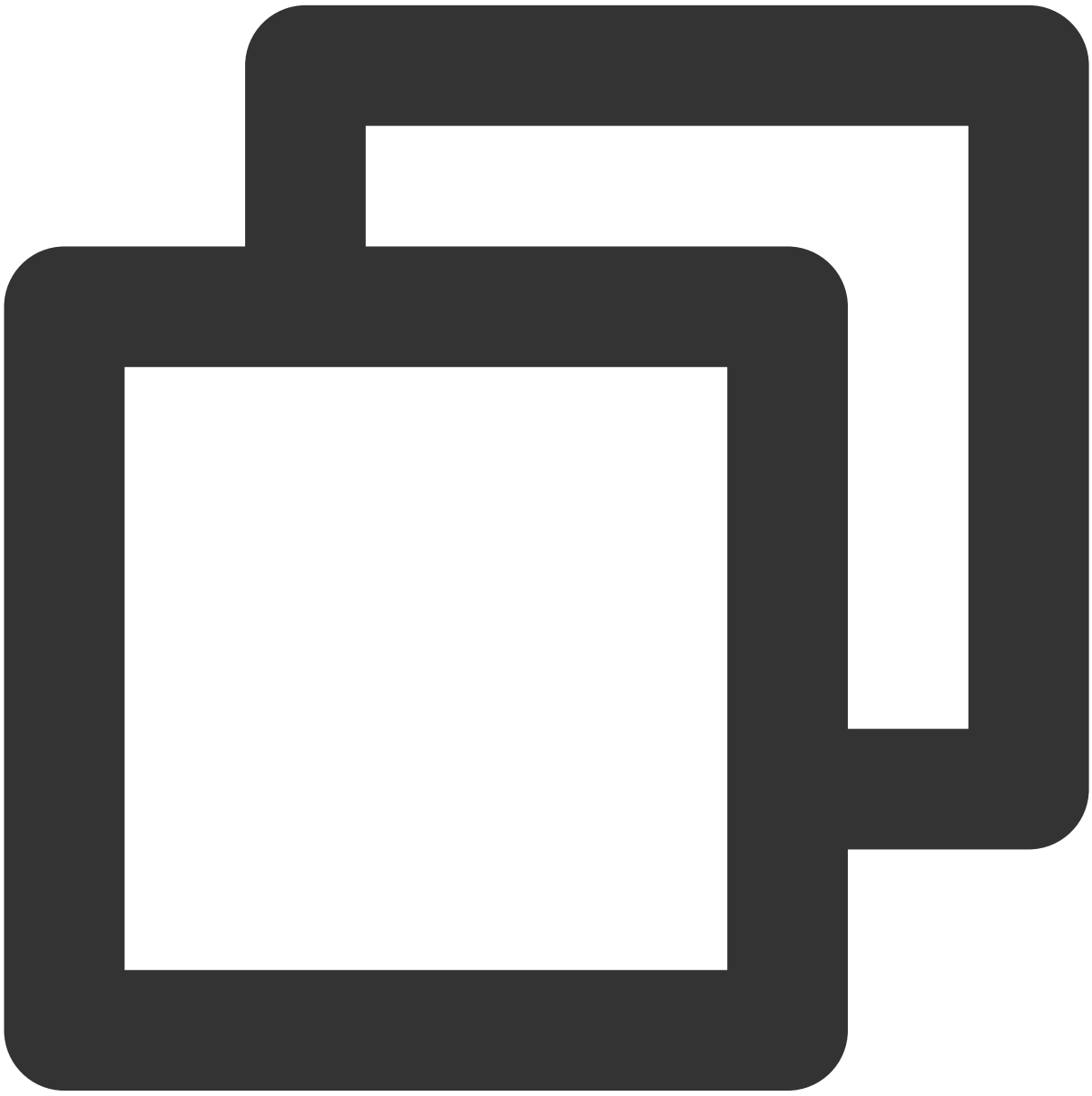
1. 在iOS 13（及以后版本）上，使用了 **SceneDelegate**，且最低兼容版本到iOS 13以下，需分别在 **AppDelegate** 和 **SceneDelegate**中实现如下方法。



```
/// AppDelegate 中的实现，iOS 13 以前版本的处理
- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUs
    [TUICallKitVoIPExtension callWith:userActivity];
    return YES;
}

/// SceneDelegate 中的实现
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session optio
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity *
        [TUICallKitVoIPExtension callWith:userActivity];
    }];
```

```
}

- (void)scene:(UIScene *)scene continueUserActivity:(NSUserActivity *)userActivity
    [TUICallKitVoIPExtension callWith:userActivity];
}
```

2. 在iOS 13（及以后版本）上，未使用**SceneDelegate**，只需要在 **AppDelegate** 实现如下方法。



```
- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUs
    [TUICallKitVoIPExtension callWith:userActivity];
    return YES;
}
```

常见问题

获取不到 VoIP Push

1. 首先检查下 App 的运行环境和证书的环境是否一致，证书 ID 是否匹配，如果不一致，无法收到推送。
2. 请确认当前您登录的账号是否处于离线状态：按 home 键切后台、登录后主动杀进程退出。VoIP Push 目前只支持离线状态下的推送。
3. 检查 步骤3：完成工程配置 是否正确。
4. 尝试重启测试手机来清除系统缓存和内存。

APNs

最近更新时间：2024-01-17 11:58:53

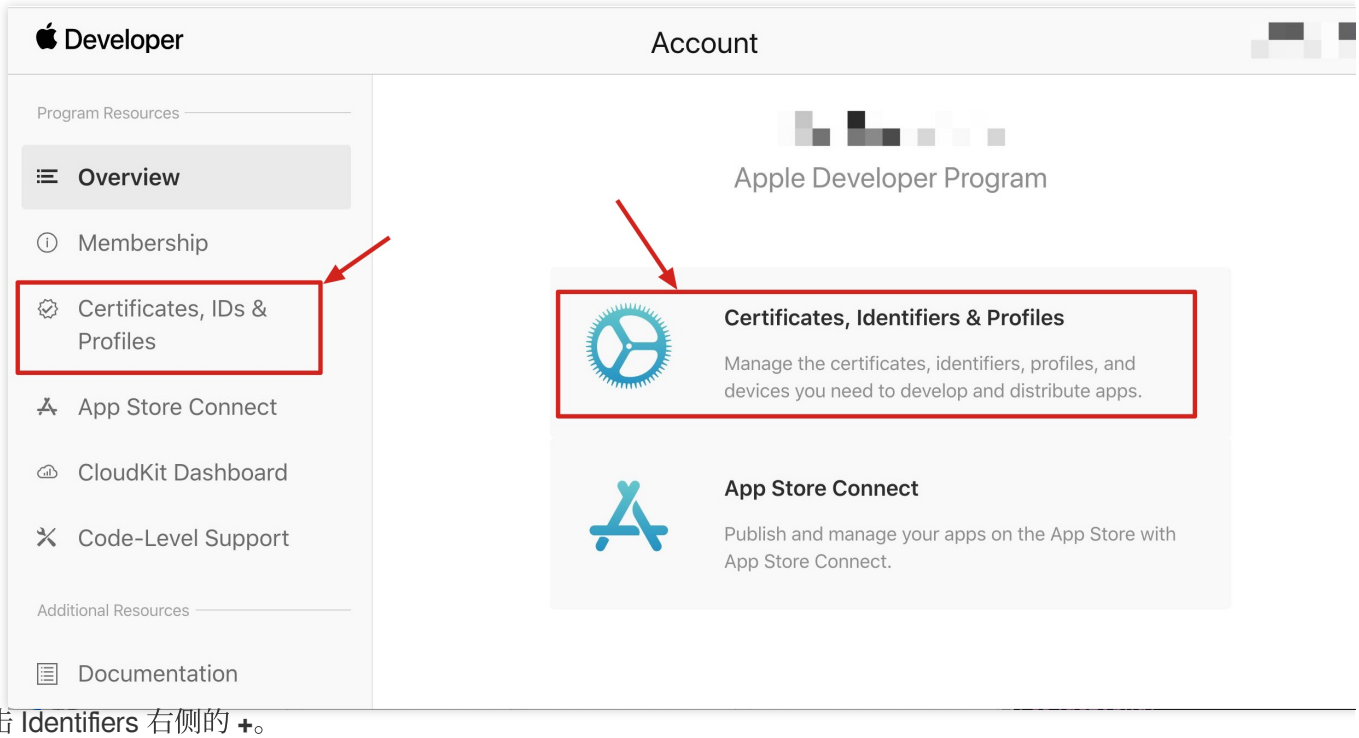
离线唤醒功能，能够让您的 App 在后台运行或者离线状态下依然能够收到音视频通话的响铃呼叫，TUICallKit 使用 Apple 提供的系统级推送通道（APNs）来进行消息通知。

配置离线推送

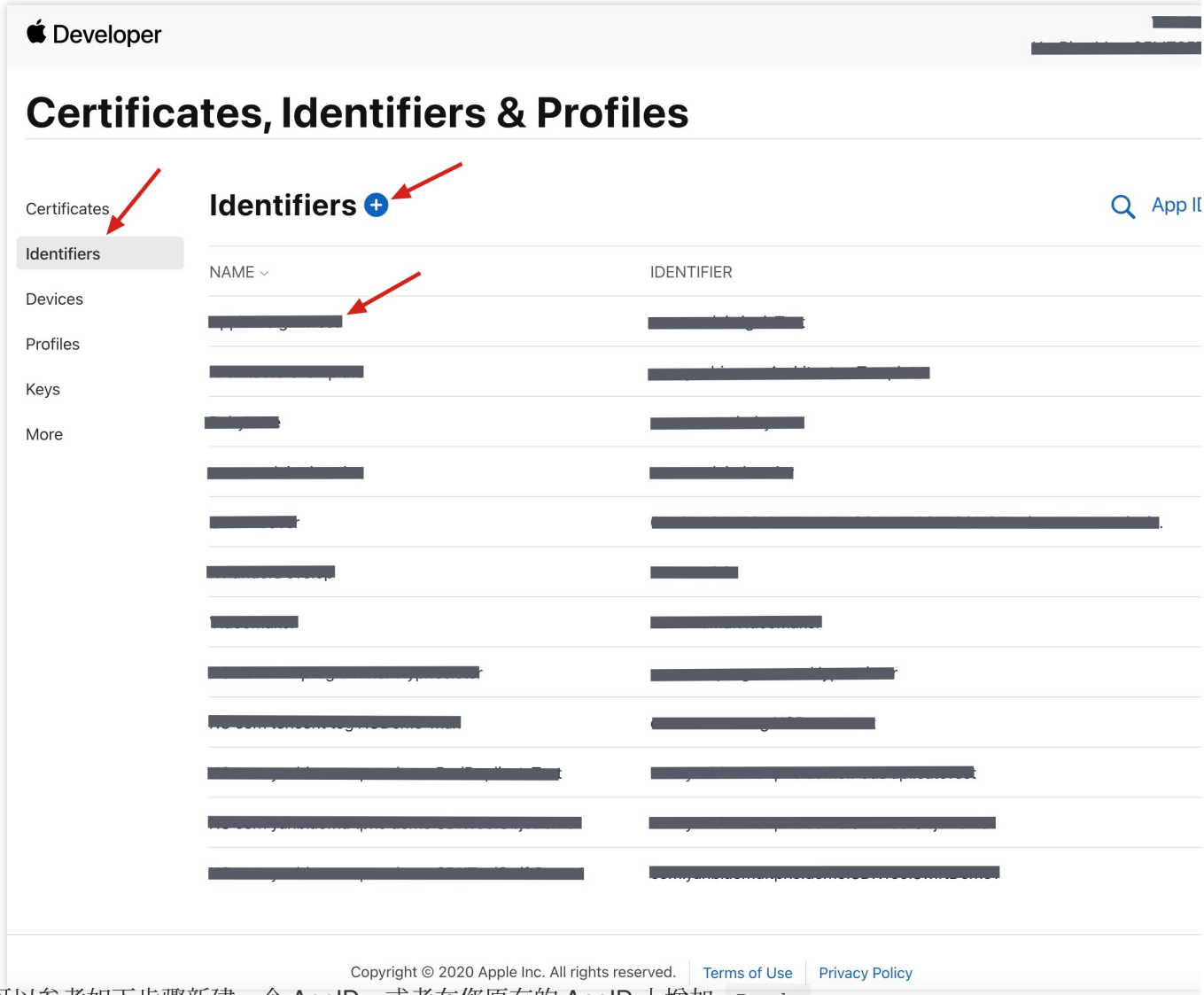
1. 开启 APP 远程推送。
2. 生成推送证书。
3. 上传证书到 IM 控制台。
4. 完成工程配置
5. 在 App 每次登录时，向苹果获取 deviceToken。
6. 调用 setAPNS 接口将其上报到 IM 后台。

步骤1：开启 APP 远程推送

1. 登录 苹果开发者中心 网站，单击 **Certificates,Identifiers & Profiles** 或者侧栏的 **Certificates, IDs & Profiles**，进入 Certificates, IDs & Profiles 页面。



2. 单击 Identifiers 右侧的 +。

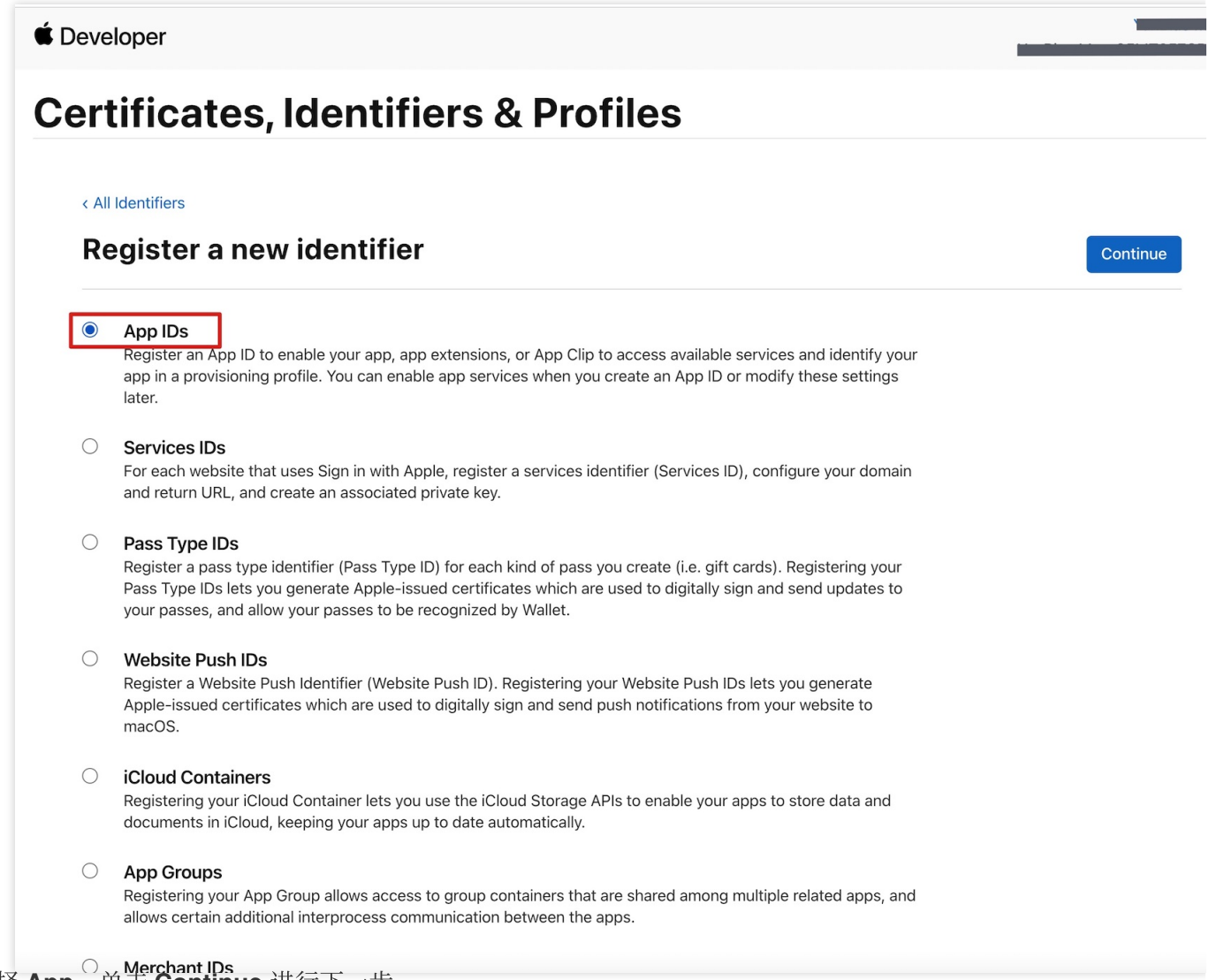


3. 您可以参考如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service`。

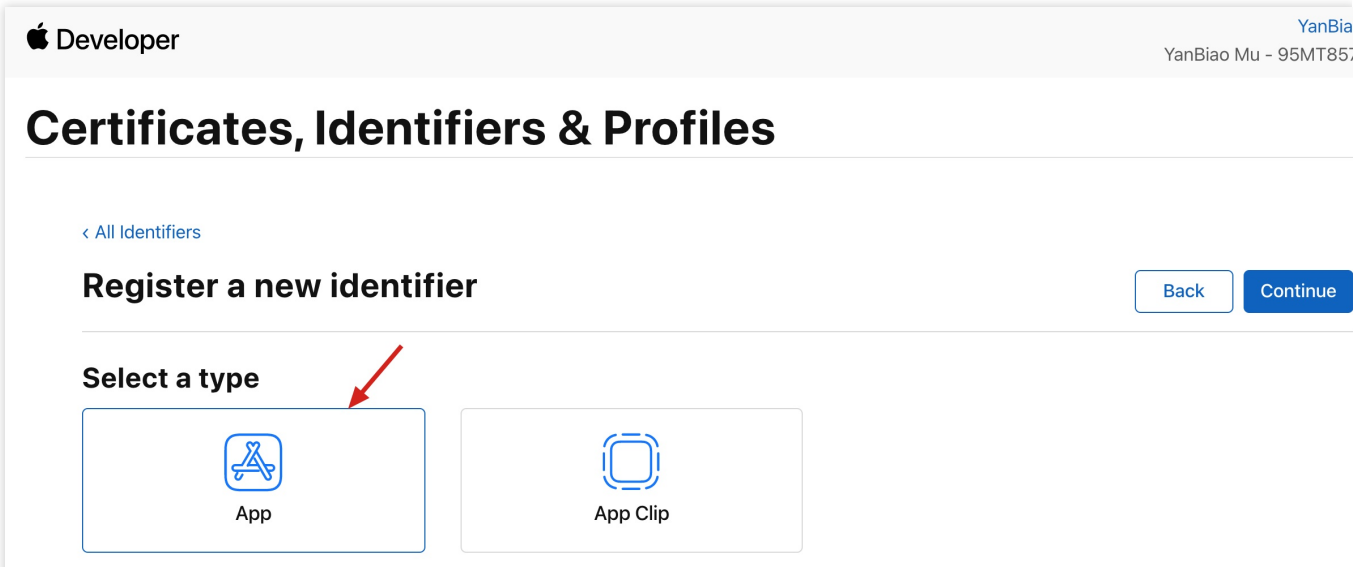
说明：

需要注意的是，您 App 的 `Bundle ID` 不能使用通配符 *，否则将无法使用远程推送服务。

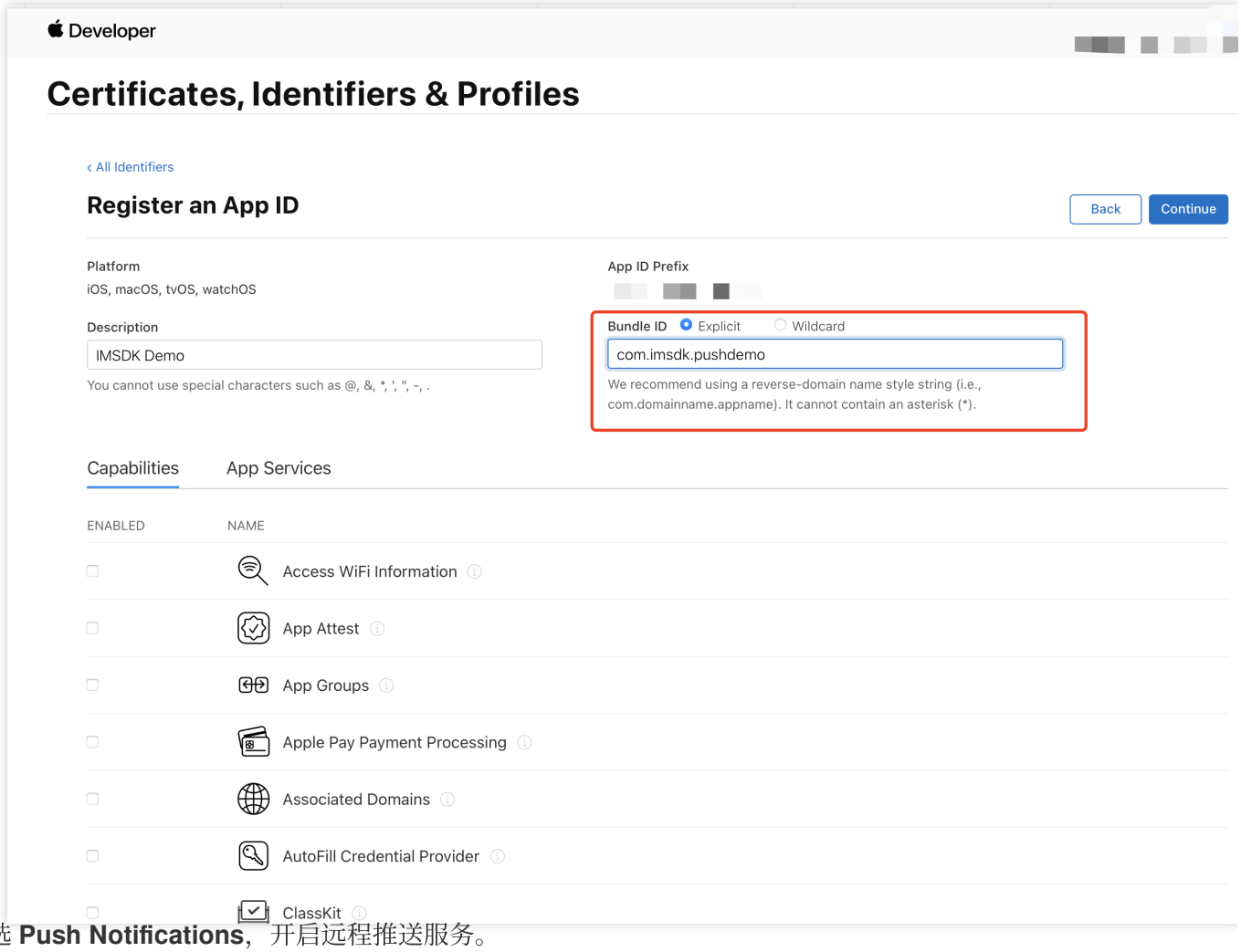
4. 勾选 **App IDs**，单击 **Continue** 进行下一步。



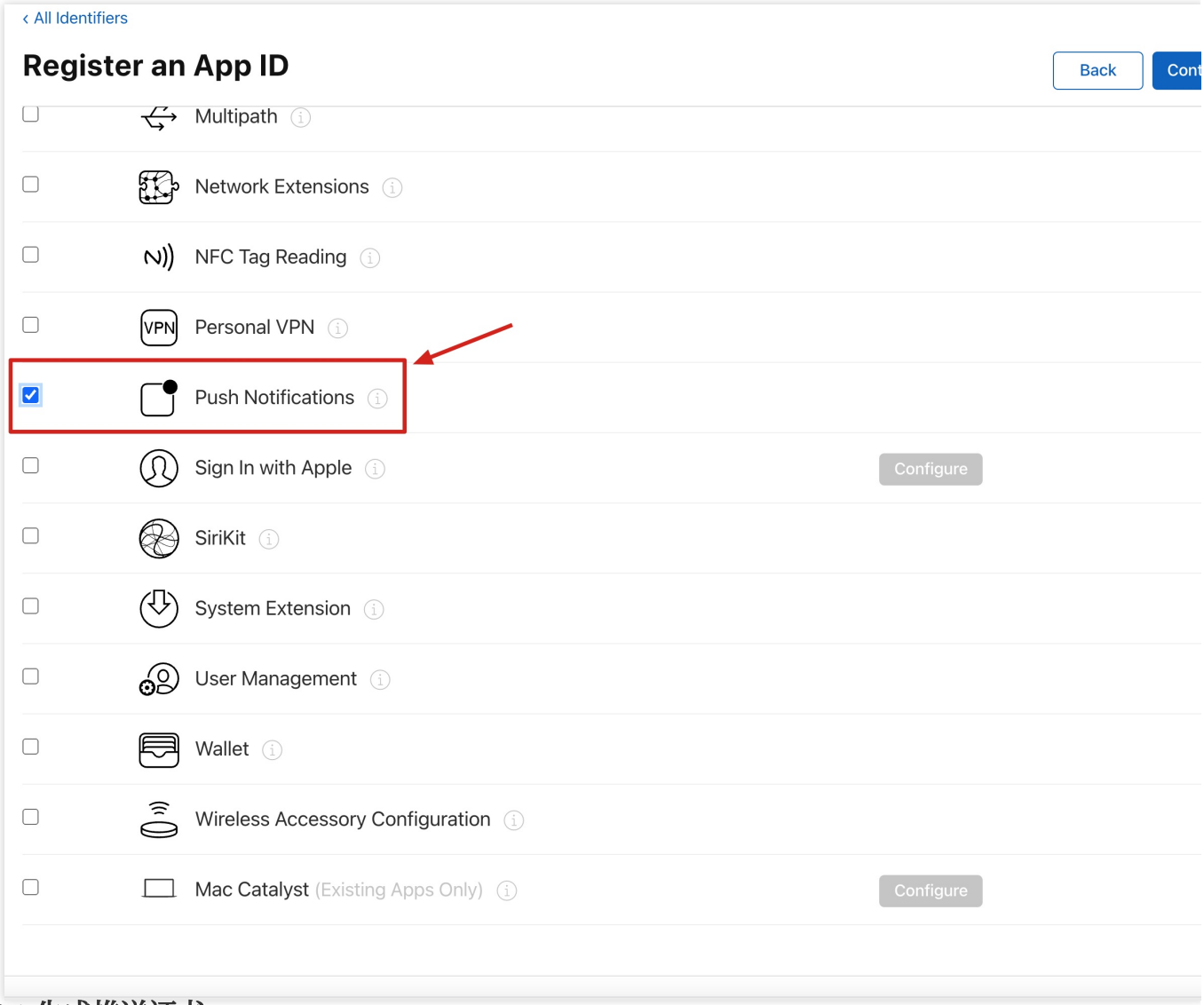
5. 选择 **App**，单击 **Continue** 进行下一步。



6. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。

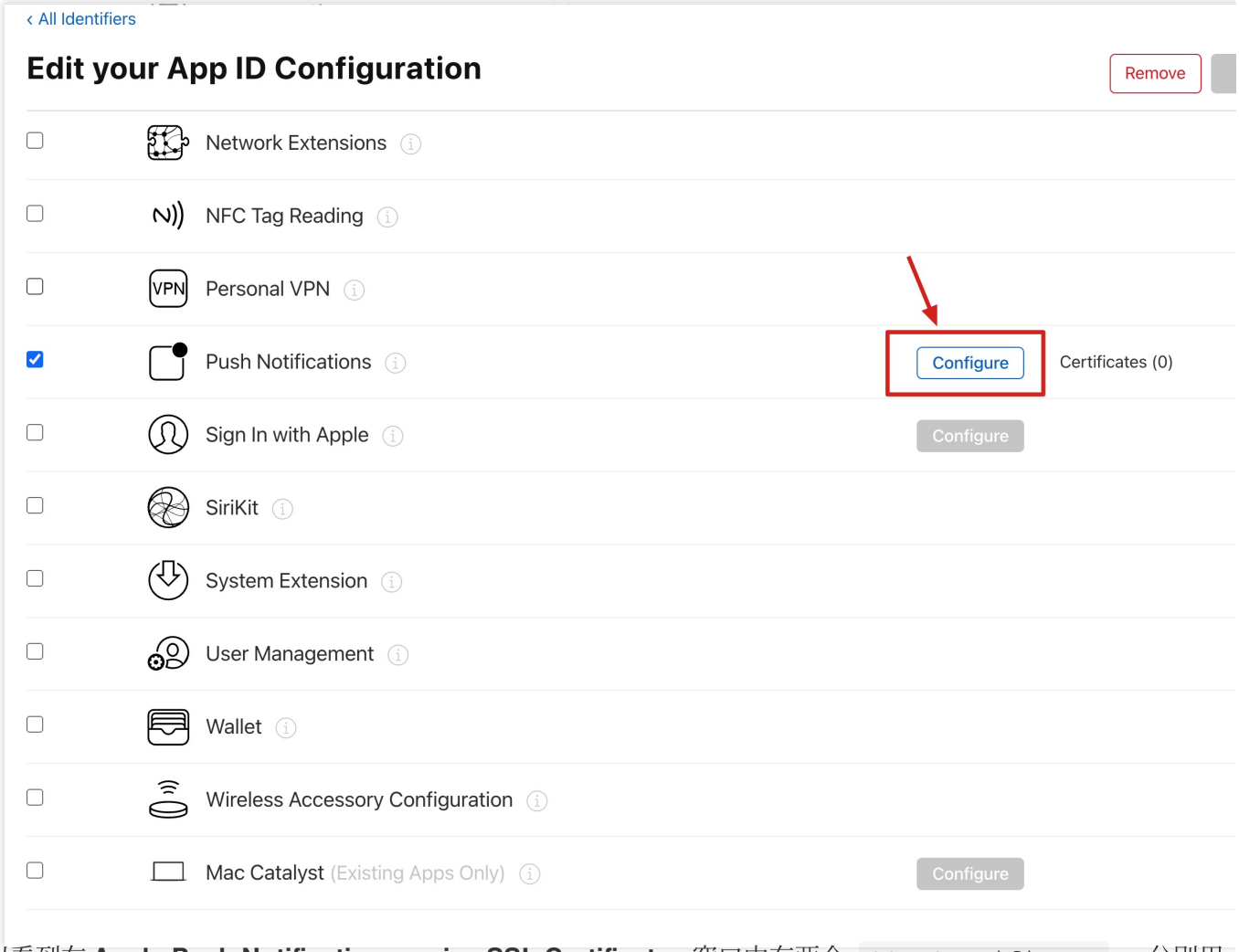


7. 勾选 **Push Notifications**，开启远程推送服务。

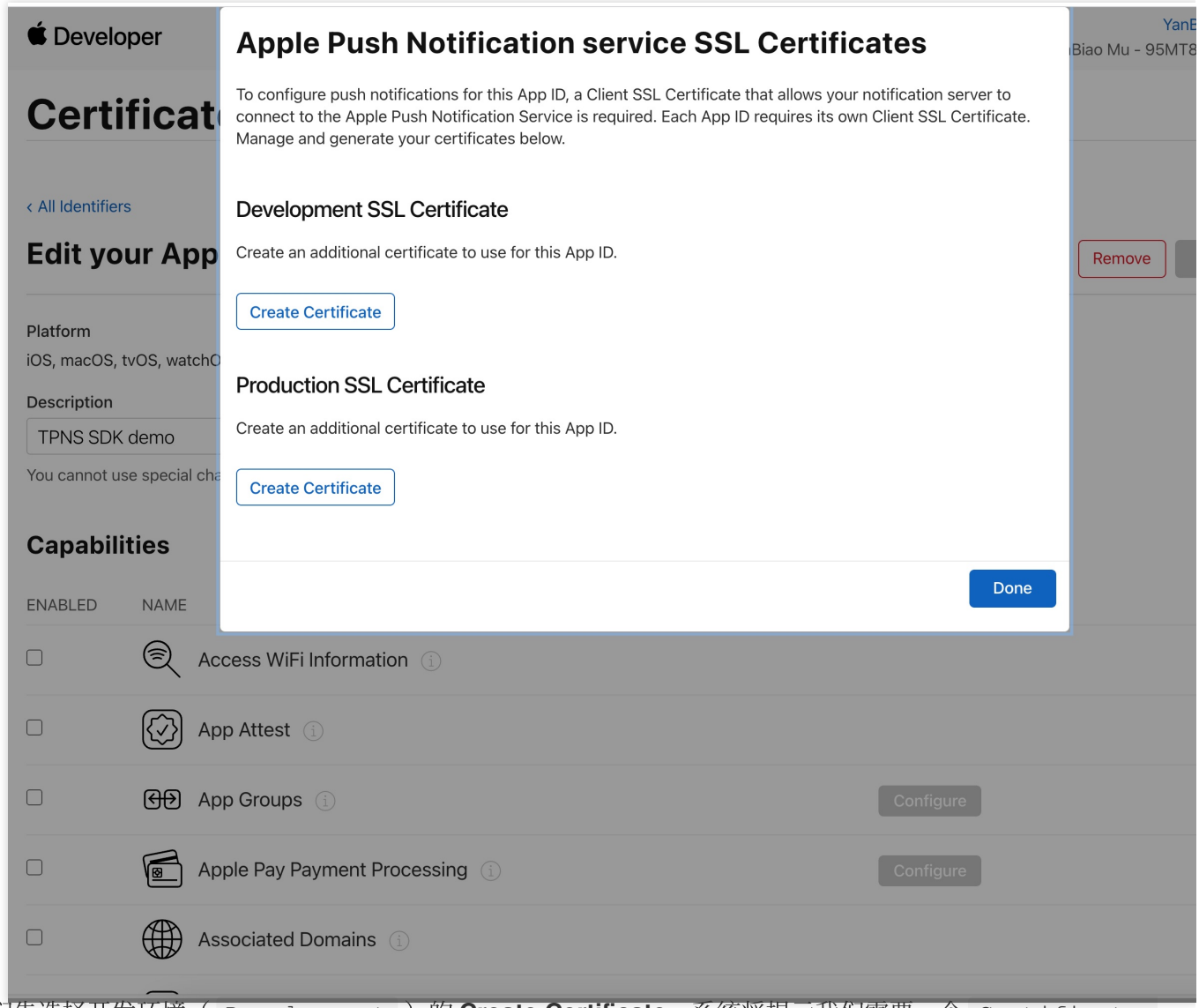


步骤2：生成推送证书

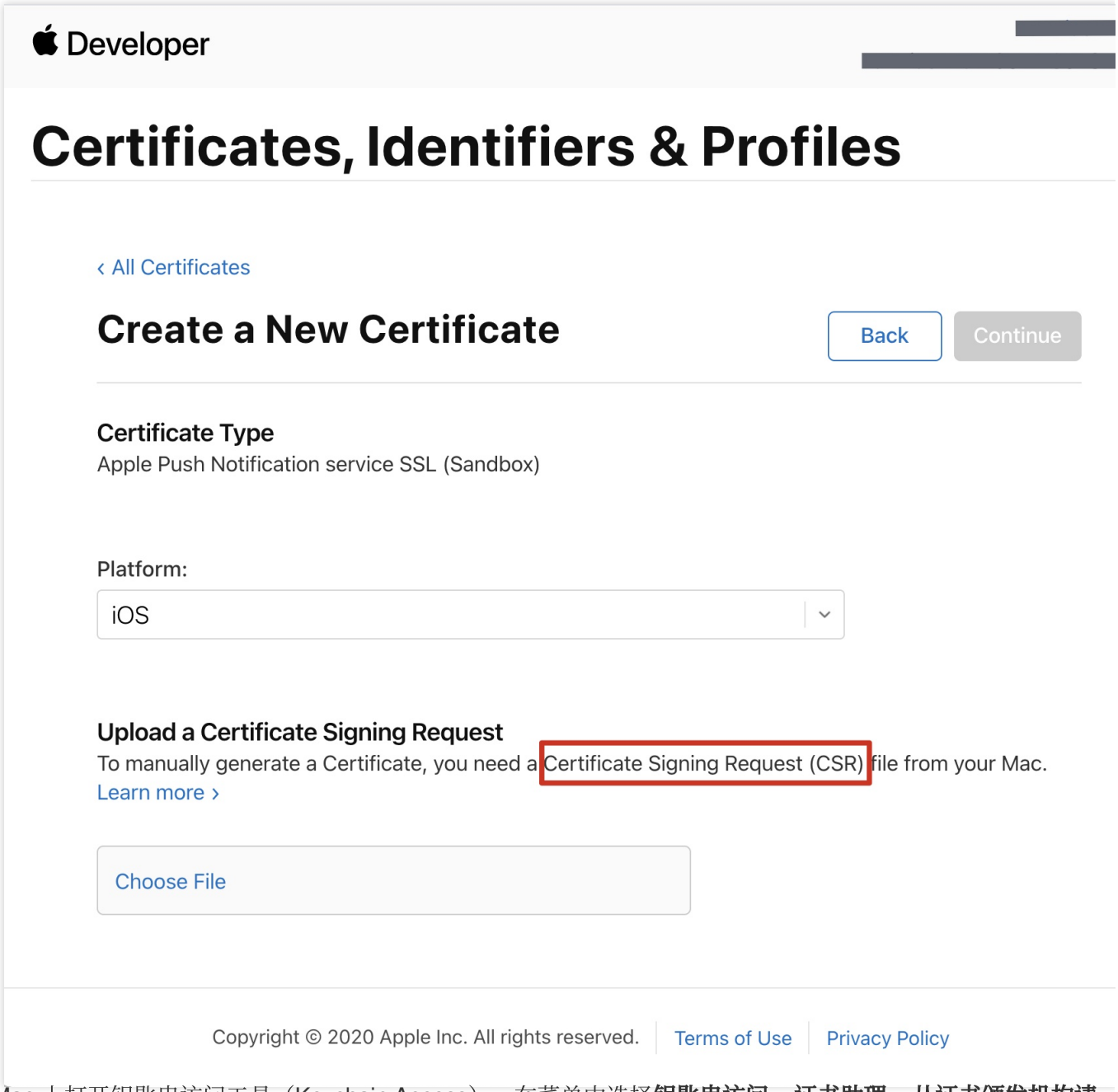
1. 选中您的 AppID，选择 **Configure**。



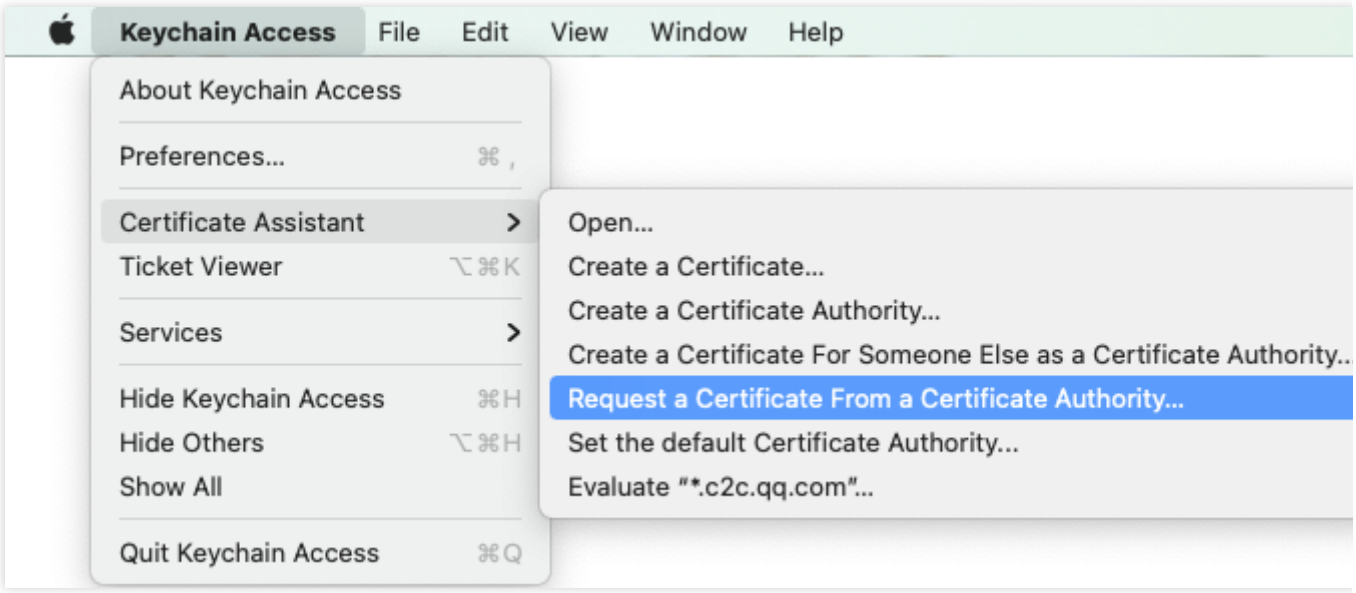
2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate` ，分别用于开发环境（ `Development` ）和生产环境（ `Production` ）的远程推送证书，如下图所示：



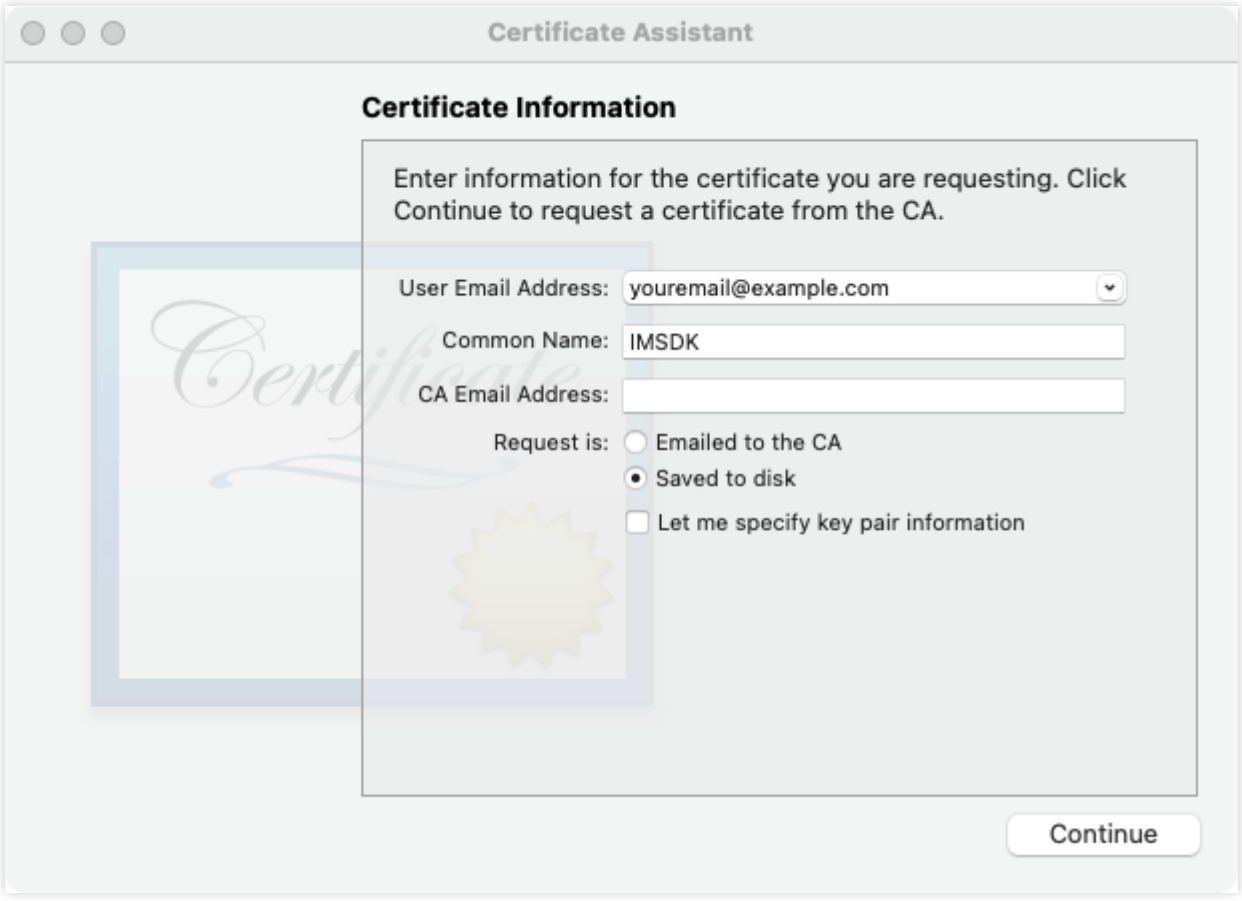
3. 我们先选择开发环境（ `Development` ）的 **Create Certificate**，系统将提示我们需要一个 `Certificate Signing Request (CSR)` 。



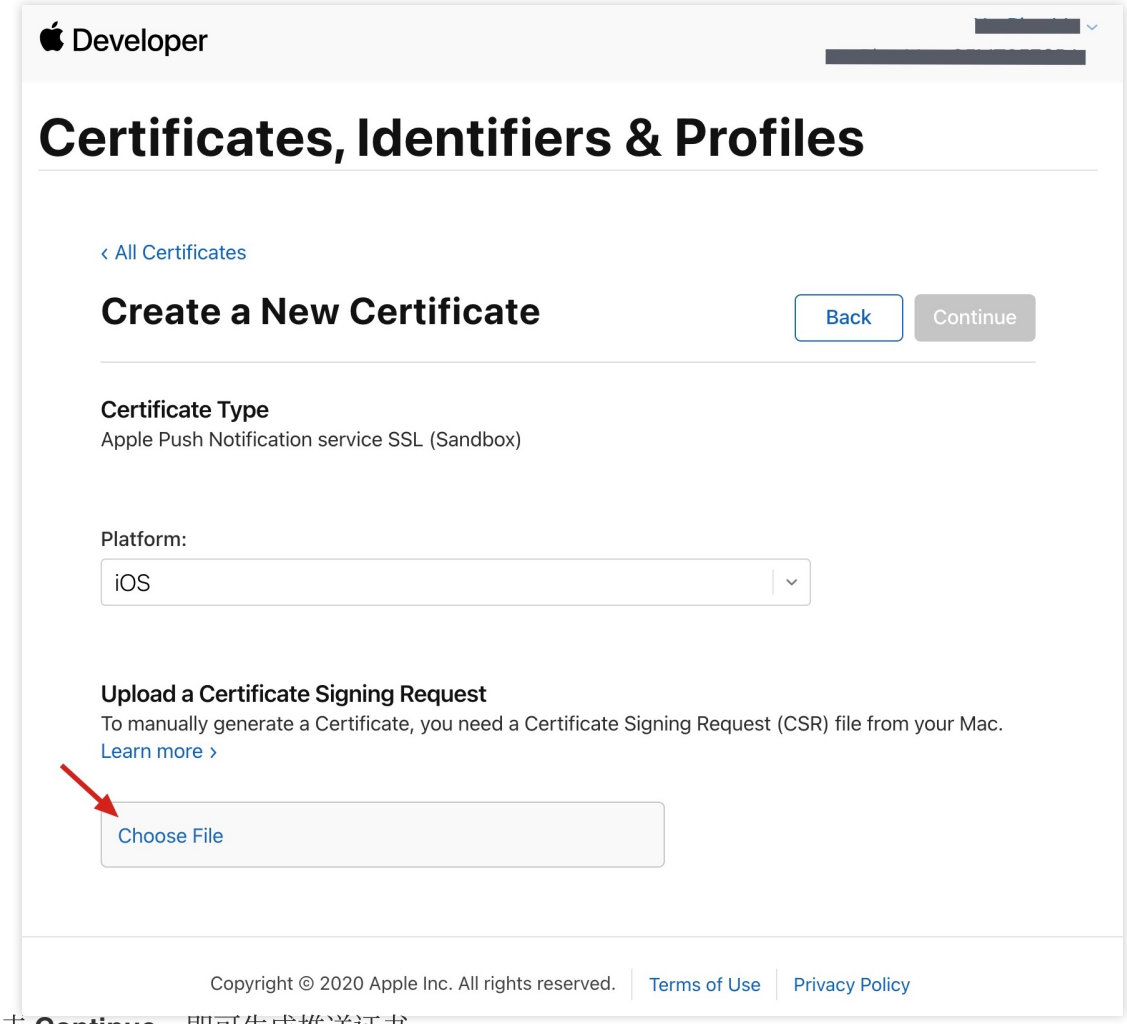
4. 在 Mac 上打开钥匙串访问工具（Keychain Access），在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



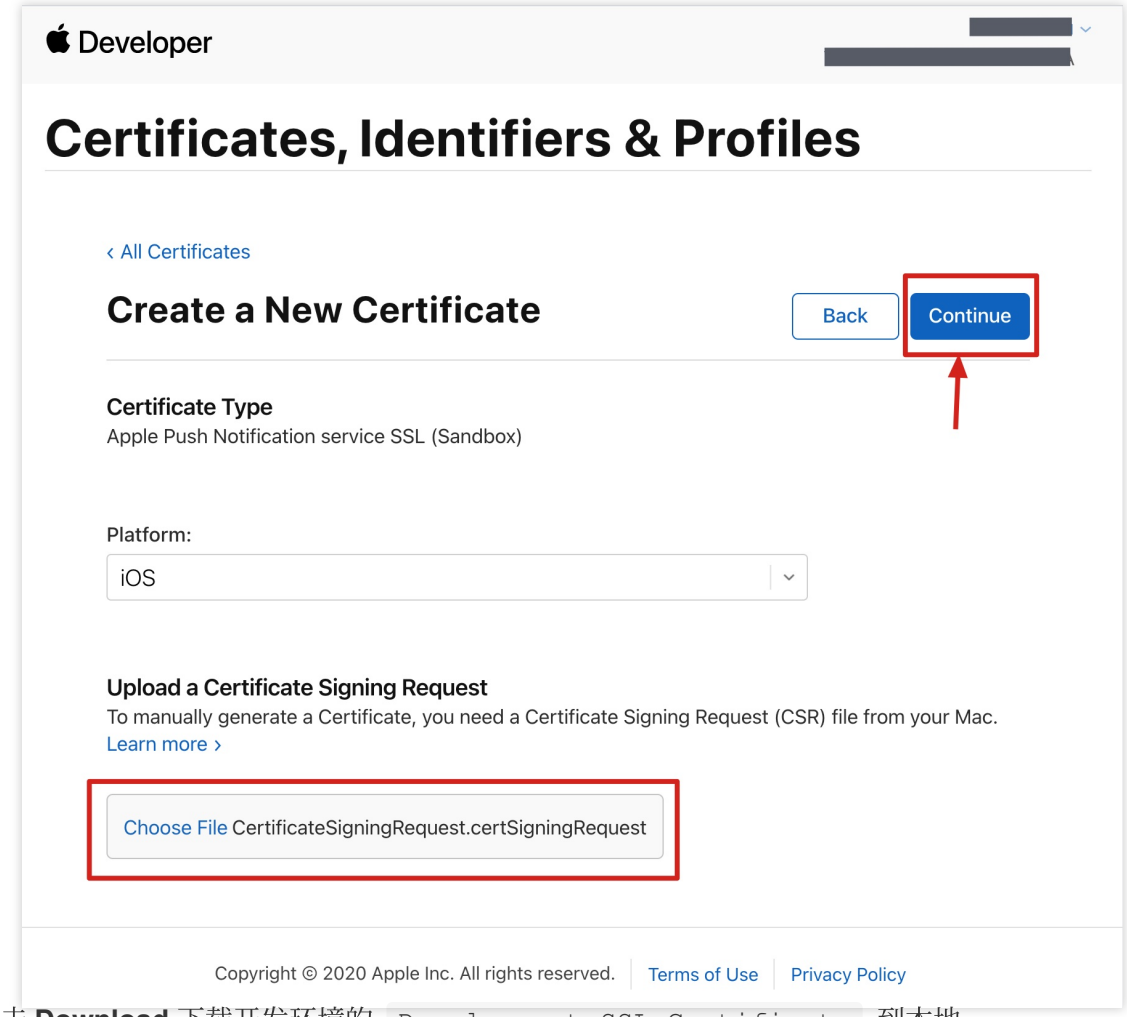
5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击继续，系统将生成一个 `*.certSigningRequest` 文件。



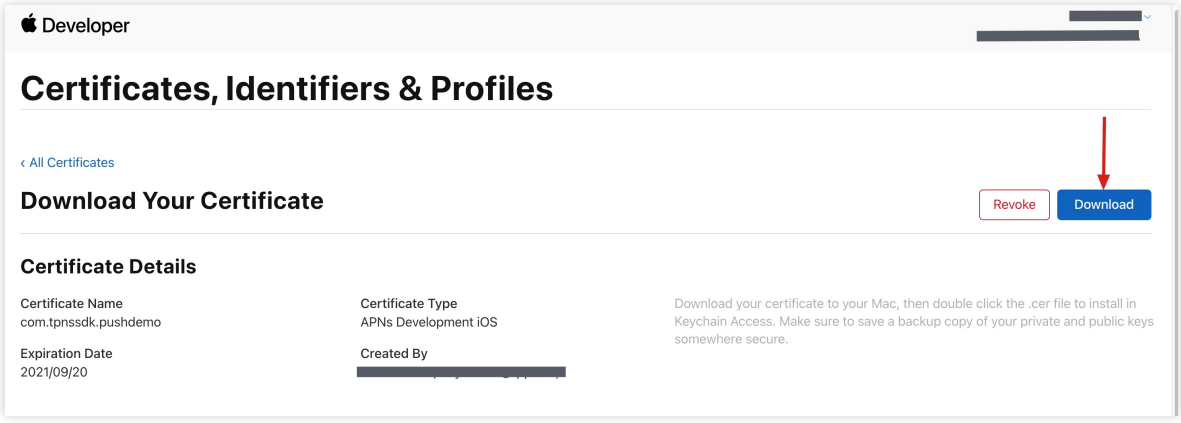
6. 返回上述 `第3步` 中 `Apple Developer` 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。



7. 单击 **Continue**，即可生成推送证书。



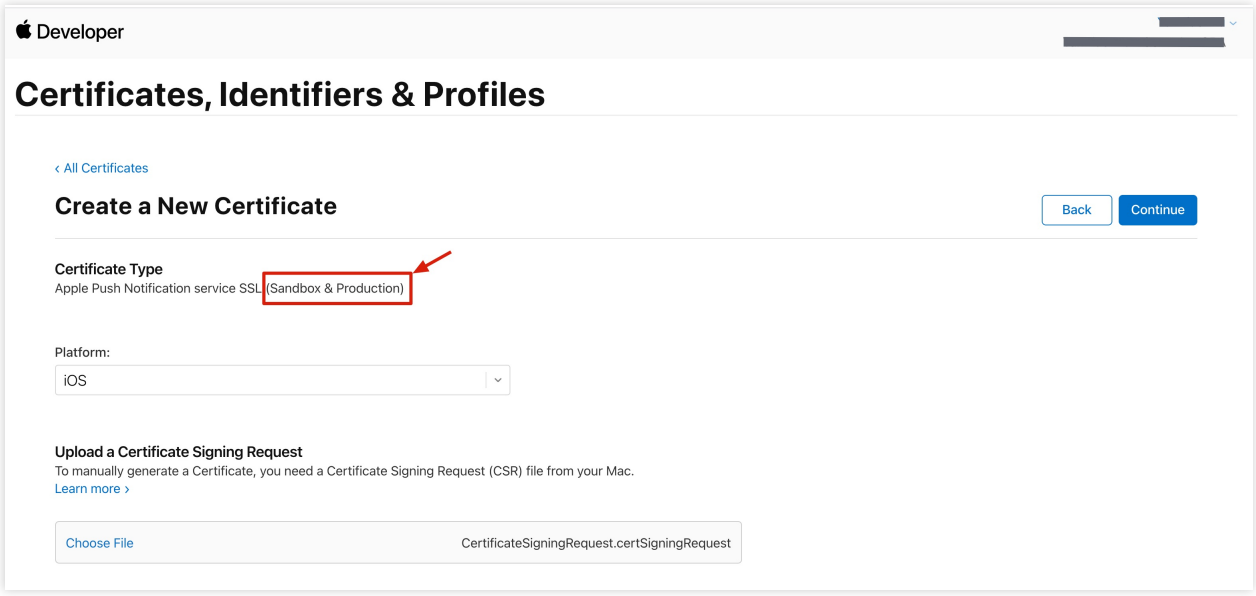
8. 单击 **Download** 下载开发环境的 `Development SSL Certificate` 到本地。

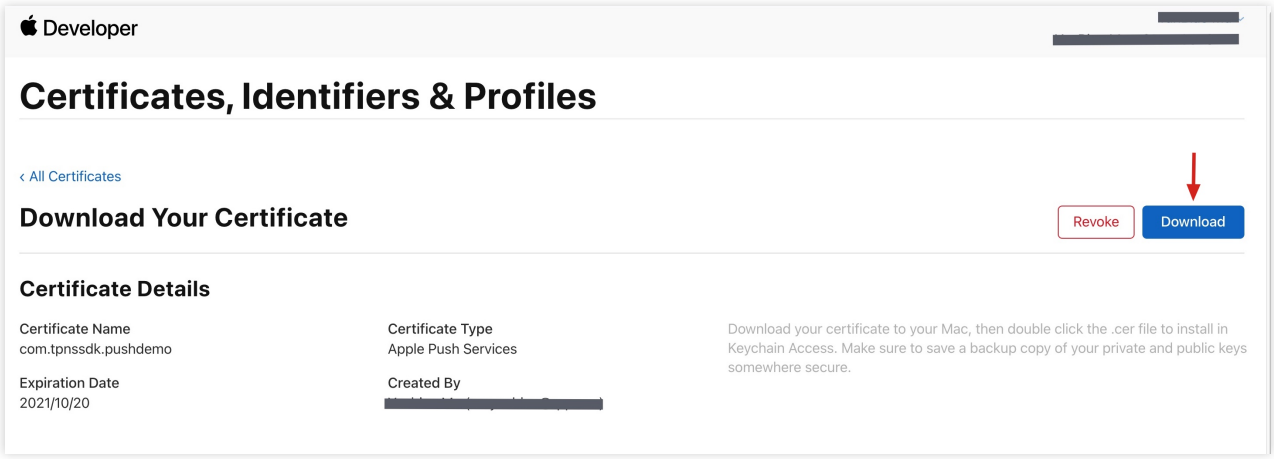


9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

说明：

生产环境的证书实际是开发(`Sandbox`)+生产(`Production`)的合并证书，可以同时作为开发环境和生产环境的证书使用。





10. 双击打开下载的开发环境和生产环境的 `SSL Certificate`，系统会将其导入钥匙串中。

11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（`Apple Development IOS Push Service`）和生产环境（`Apple Push Services`）的 P12 文件。

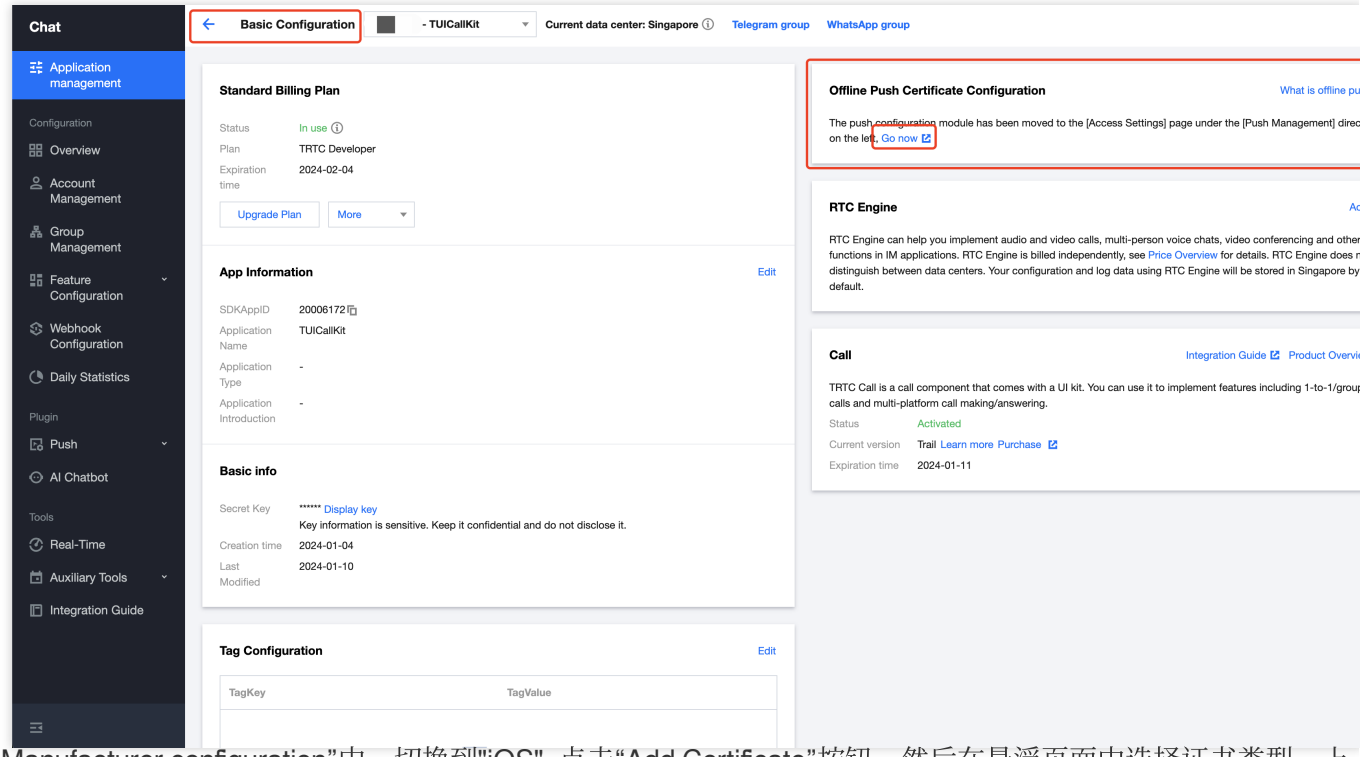
说明：

保存 `P12` 文件时，请务必要为其设置密码。

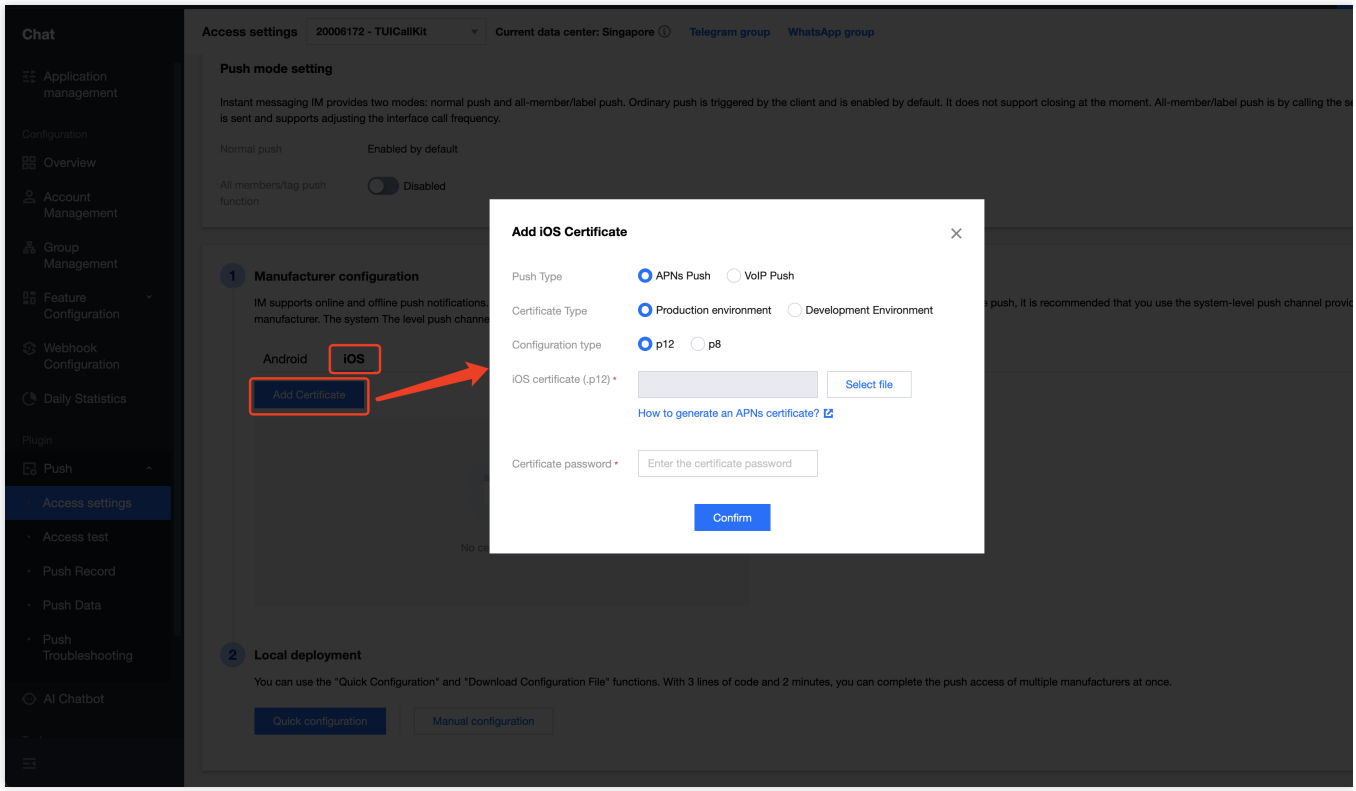
步骤3：上传证书到 IM 控制台

1. 登录 即时通信 IM 控制台。

2. 单击目标应用卡片，进入应用的基础配置页面。单击 "Offline Push Certificate Configuration"选项卡中的 "Go now"。



3. 在“Manufacturer configuration”中，切换到"iOS", 点击“Add Certificate”按钮，然后在悬浮页面中选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击**Confirm**。



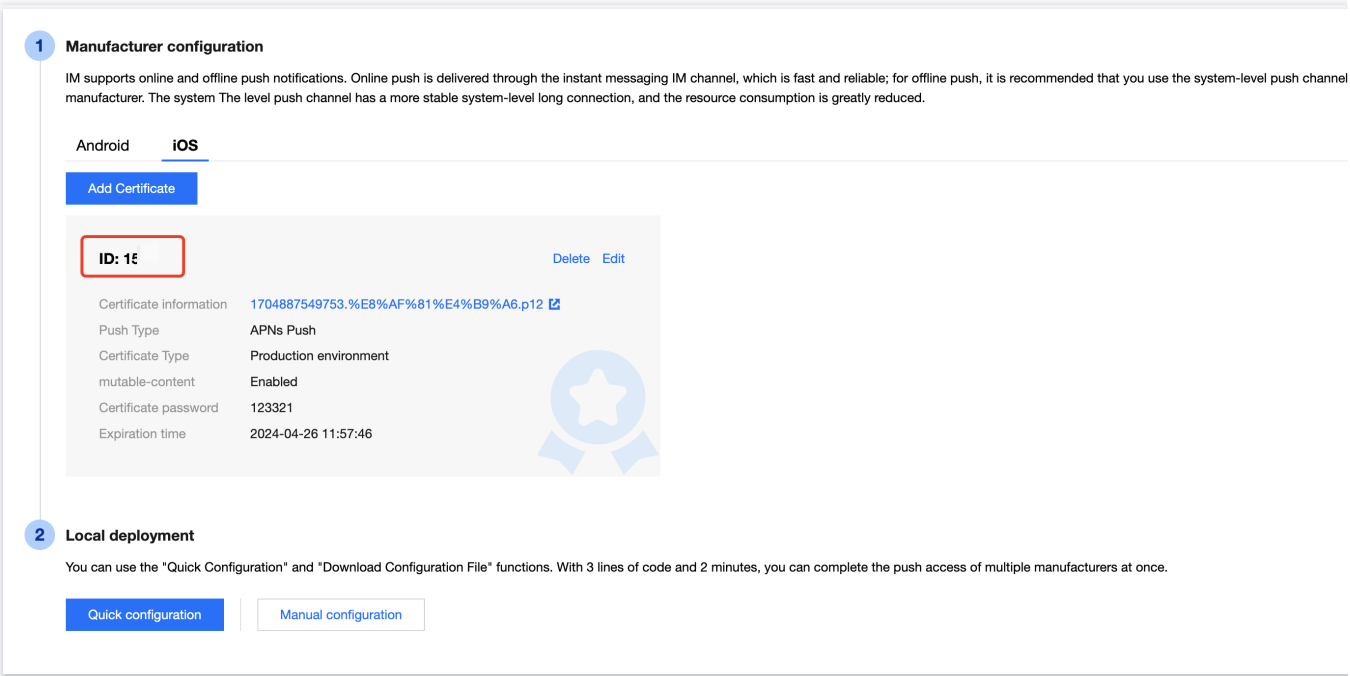
说明：

上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

上传证书需要设置密码，无密码收不到推送。

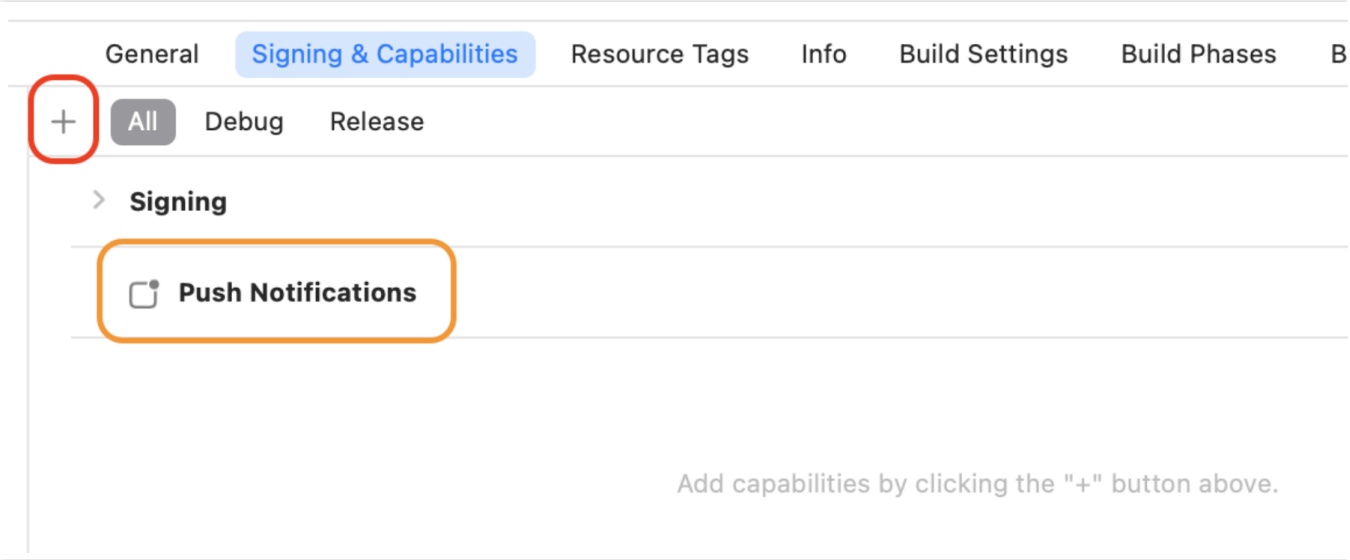
发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。
4. 待推送证书信息生成后，记录证书的 ID。



步骤4：完成工程配置

要在应用程序中添加所需的权限，请在 Xcode 项目中启用推送通知功能。
打开 **Xcode** 项目，在 **Project** > **Target** > **Capabilities** 页面中点击红框中的加号按钮，然后选择并添加 **Push Notifications**，添加后的结果如图中黄框所示：

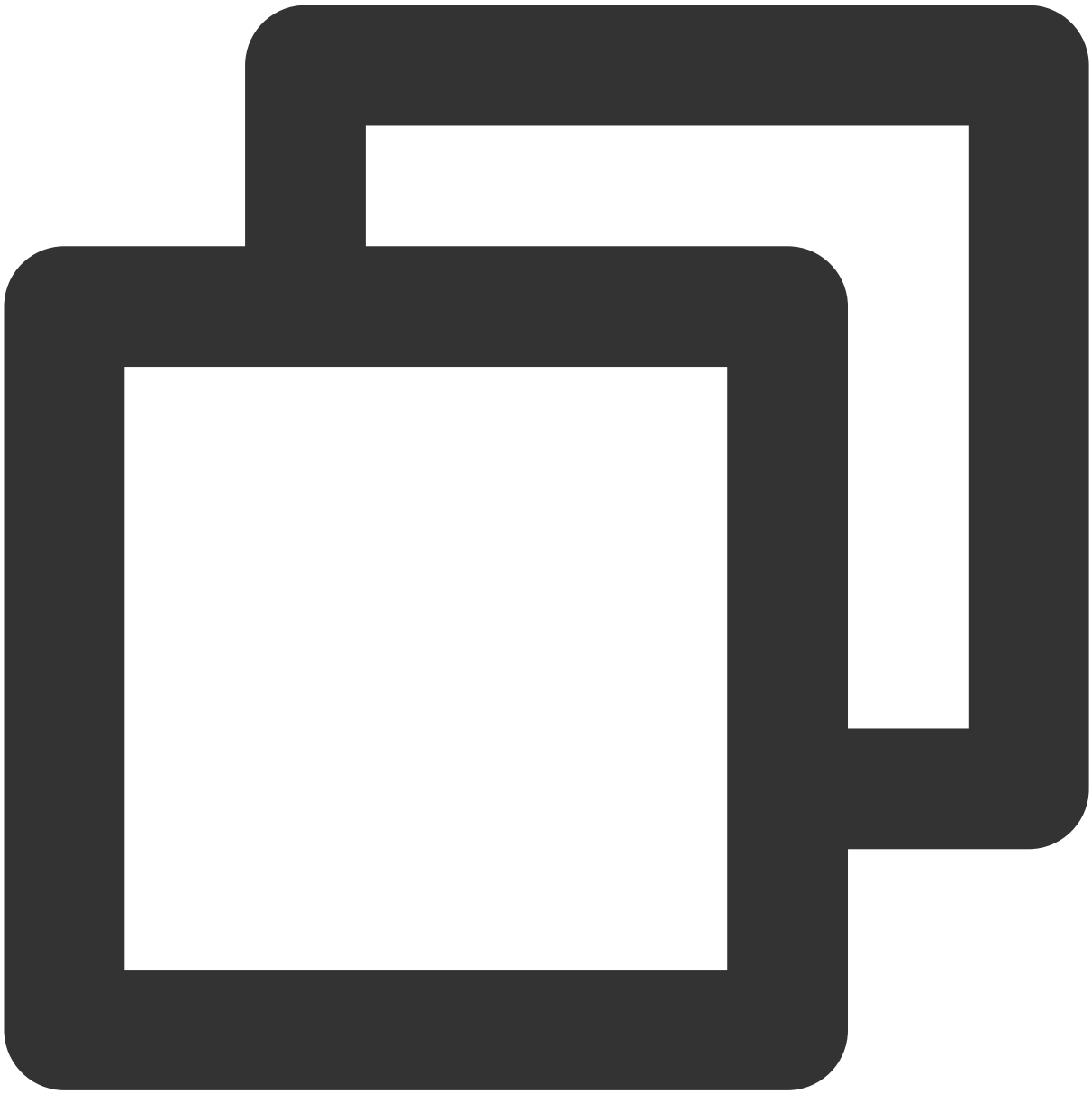


步骤5：在 App 每次登录时，向苹果获取 deviceToken

您可以在您的 App 中添加如下代码，用来向苹果的后台服务器获取 deviceToken：

说明：

考虑到合规，建议您在用户同意隐私协议之后再向苹果请求 deviceToken。

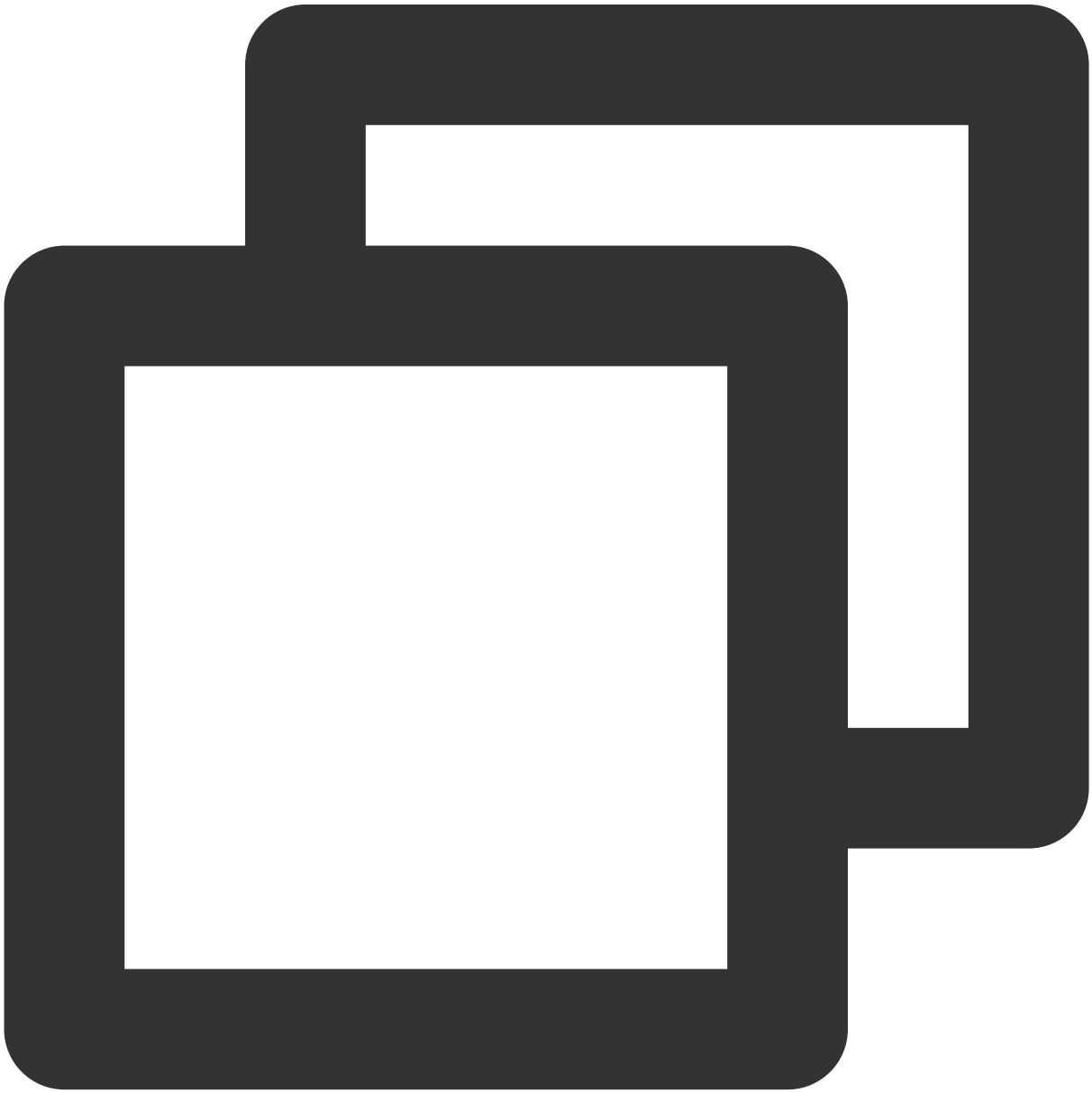


```
// 向苹果后台请求 DeviceToken
- (void)registerNotification {
    [[UIApplication sharedApplication] registerUserNotificationSettings:
              [UIUserNotificationSettings settingsForTypes:
              (UIUserNotificationTypeSound | UIUserNotificationTypeAlert | UIUserNo
              categories:nil]];
    [[UIApplication sharedApplication] registerForRemoteNotifications];
}
```

```
// 在 AppDelegate 的回调中会返回 deviceToken，需要在登录后上报给腾讯云后台
-(void)application:(UIApplication *)app didRegisterForRemoteNotificationsWithDevice
    // 记录下 Apple 返回的 deviceToken
    _deviceToken = deviceToken;
}
```

步骤6：调用 接口将其上报到 IM 后台

在 IM SDK 登录成功后，就可以调用 setAPNS 接口，将 步骤5 中获取的 deviceToken 上传到腾讯云后台，示例代码如下:



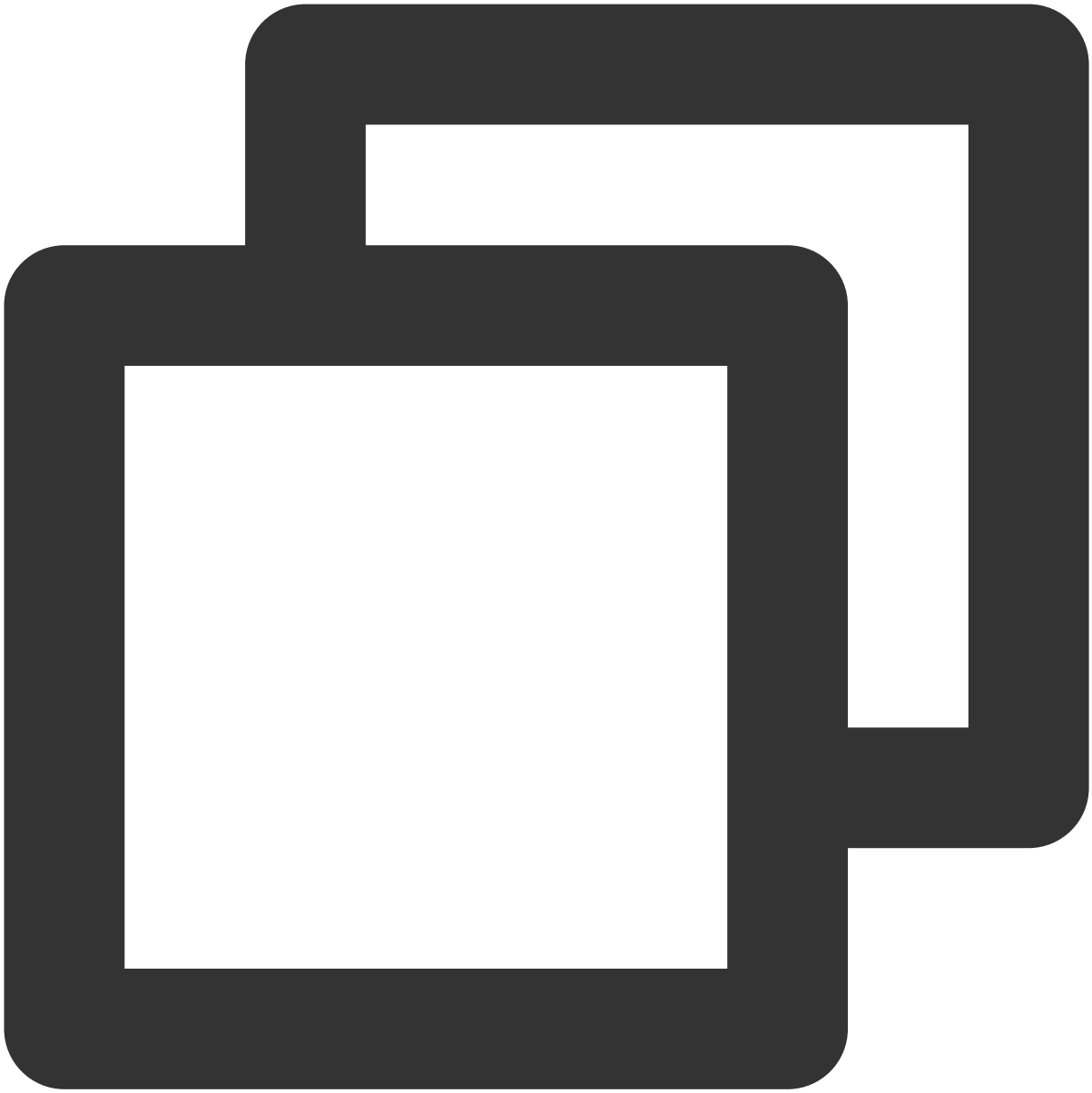
```
- (void)pushRegisterIfLogined {
  if (self.deviceToken) {
      V2TIMAPNSConfig *config = [[V2TIMAPNSConfig alloc] init];
      // 用户自己到苹果注册开发者证书，在开发者帐号中下载并生成证书(p12 文件)，将生成的 p12 文件
      // 控制台会自动生成一个证书 ID，将证书 ID 传入以下 businessID 参数中。
      config.businessID = sdkBusiId;
      // 苹果后台服务器获取的 deviceToken
      config.token = self.deviceToken;
      [[V2TIMManager sharedInstance] setAPNS:config succ:^{
          NSLog(@"%s, succ", __func__");
      } fail:^(int code, NSString *msg) {
          NSLog(@"%s, fail, %d, %@", __func__, code, msg);
      }];
  }
  // ...
}
```

自定义离线铃声

请在调用 call 拨打电话的时候设置 params 的 offlinePushInfo 中的 iOSSound 字段， iOSSound 传语音文件名。

注意：

离线推送声音设置（仅对 iOS 生效），如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀名）设置给 iOSSound。



```
[[TUICallKit createInstance] call:@"mike 的 id" params:[self getCallParams] callMedi

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}
```

```
+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    // iOS push type: if you want user VoIP, please modify type to TUICallIOSOfflin
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChanne
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```

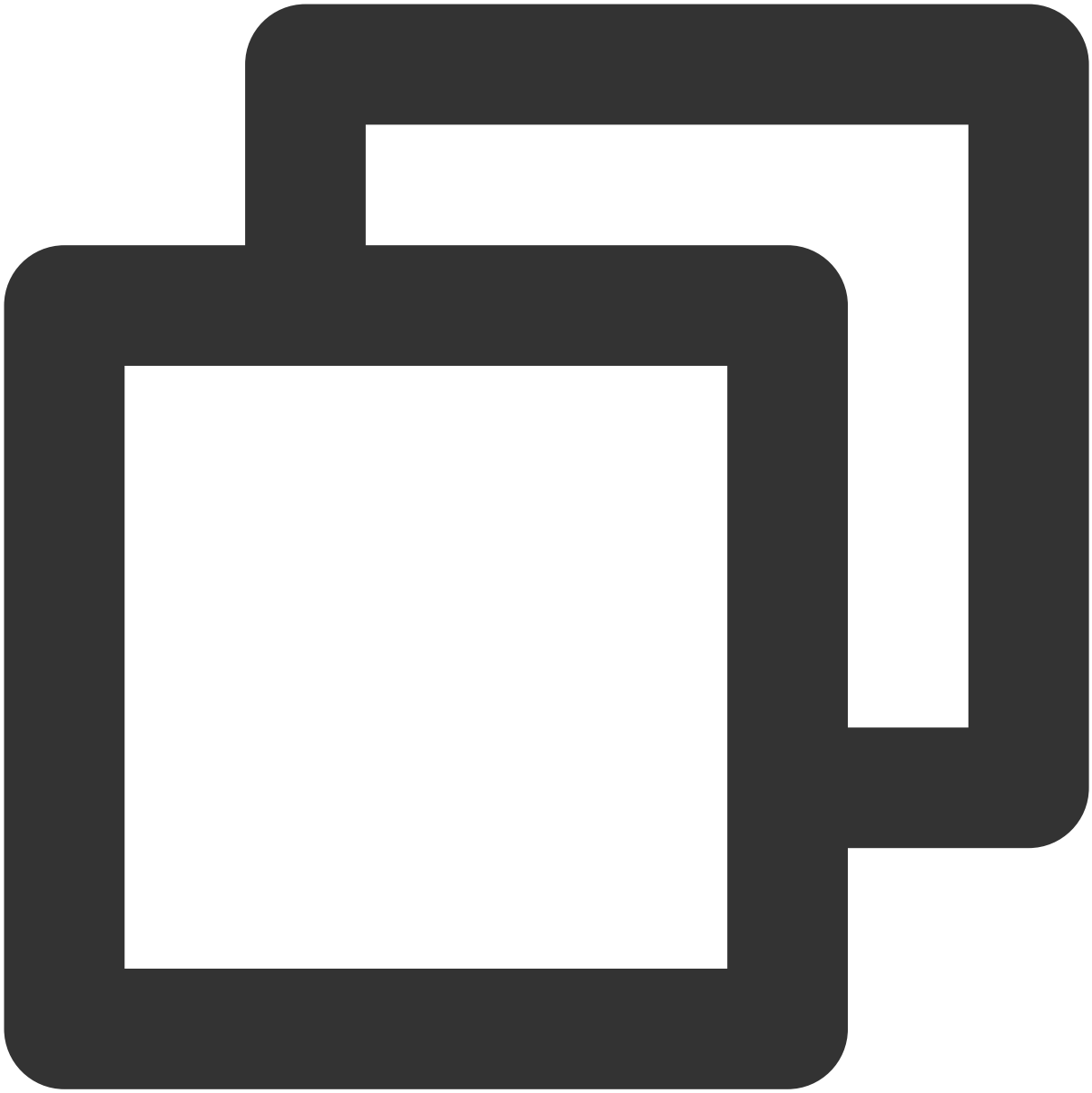
常见问题

1、收不到推送，且后台报错 bad devicetoken？

Apple 的 deviceToken 与当前编译环境有关。如果 登录 IMSDK 后上传 deviceToken 到腾讯云 所使用的证书 ID 和 token 不一致，就会报错。

如果使用的是 Release 环境编译，则 `-application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是发布环境的 token，此时 businessID 需要设置生产环境的 [证书 ID](#businessid :缺少内容)。

如果使用的是 Debug 环境编译，则 `-application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是开发环境的 token，此时 businessID 需要设置开发环境的证书 ID。



```
V2TIMAPNSConfig *confg = [[V2TIMAPNSConfig alloc] init];
/* 用户自己到苹果注册开发者证书，在开发者帐号中下载并生成证书(p12 文件)，将生成的 p12 文件传到腾讯
//推送证书 ID
confg.businessID = sdkBusiId;
confg.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:confg succ:^{

} fail:^(int code, NSString *msg) {

}];
```

2、iOS 开发环境下，注册偶现不返回 deviceToken 或提示 APNs 请求 token 失败？

此问题现象是由于 APNs 服务不稳定导致的，可尝试通过以下方式解决：

1. 给手机插入 SIM 卡后使用4G网络测试。
2. 卸载重装、重启 App、关机重启后测试。
3. 打生产环境的包测试。
4. 更换其它 iOS 系统的手机测试。

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2023-05-23 16:45:17

离线唤醒功能，能够让您的 应用在 `后台运行` 或者 `离线状态` 下依然能够收到音视频通话的响铃呼叫，TUICallKit 集成了 TUIOfflinePush 组件实现离线唤醒功能，本文将介绍如何在音视频通话项目中集成 `TUIOfflinePush` 组件。

前期准备

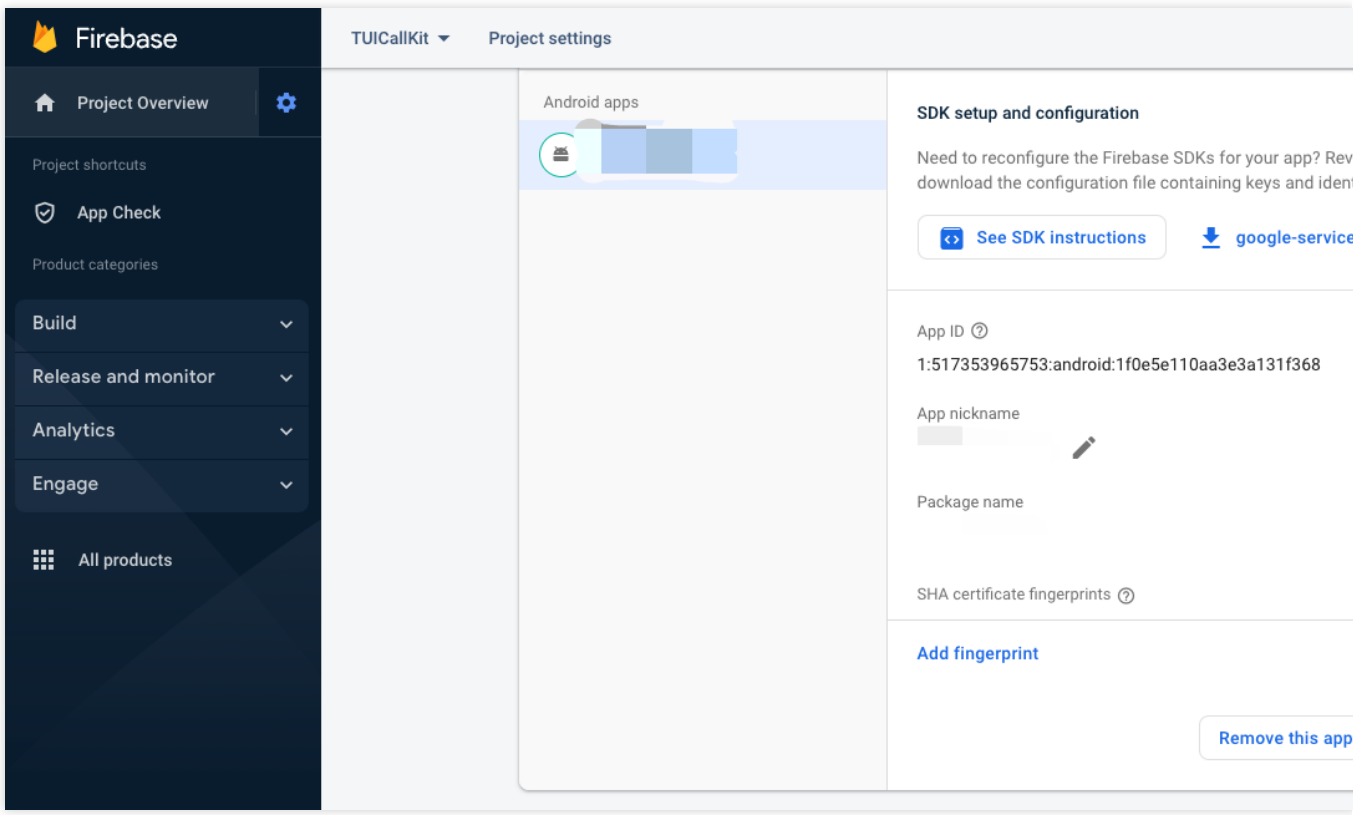
1. 注册应用到厂商推送平台

离线推送功能依赖厂商原始通道，您需要将自己的应用注册到各个厂商的推送平台，得到 APPID 和 APPKEY 等参数。

下述两个文件在后续步骤会用到：

注册华为平台的时候，下载 `agconnect-services.json` 文件，并保存。

注册 Google 平台的时候，下载 `google-services.json` 文件，并保存。

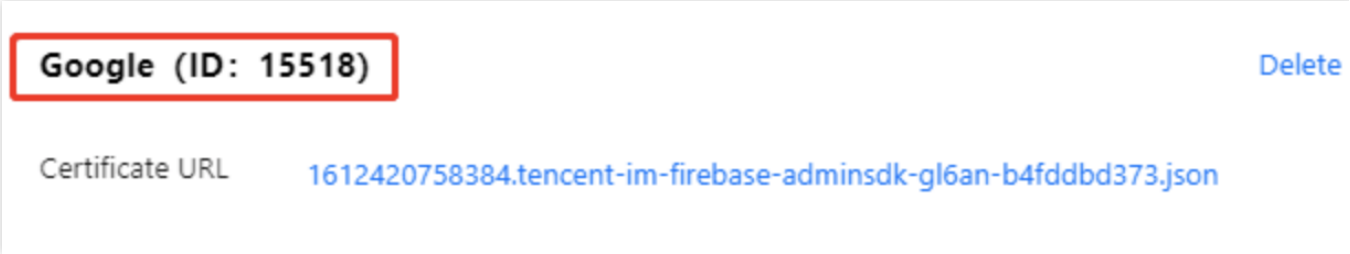


2. 在 IM 控制台进行配置

注册厂商通道需要传入自己的包名，各厂商填入的包名需保持一致，用于消息互通。具体请参见 离线推送(Android)

中的步骤二，记录生成的 ID，APPID 以及 APPKEY，他们会在后续步骤中用到。

以Google为例，推送证书 ID 如下：

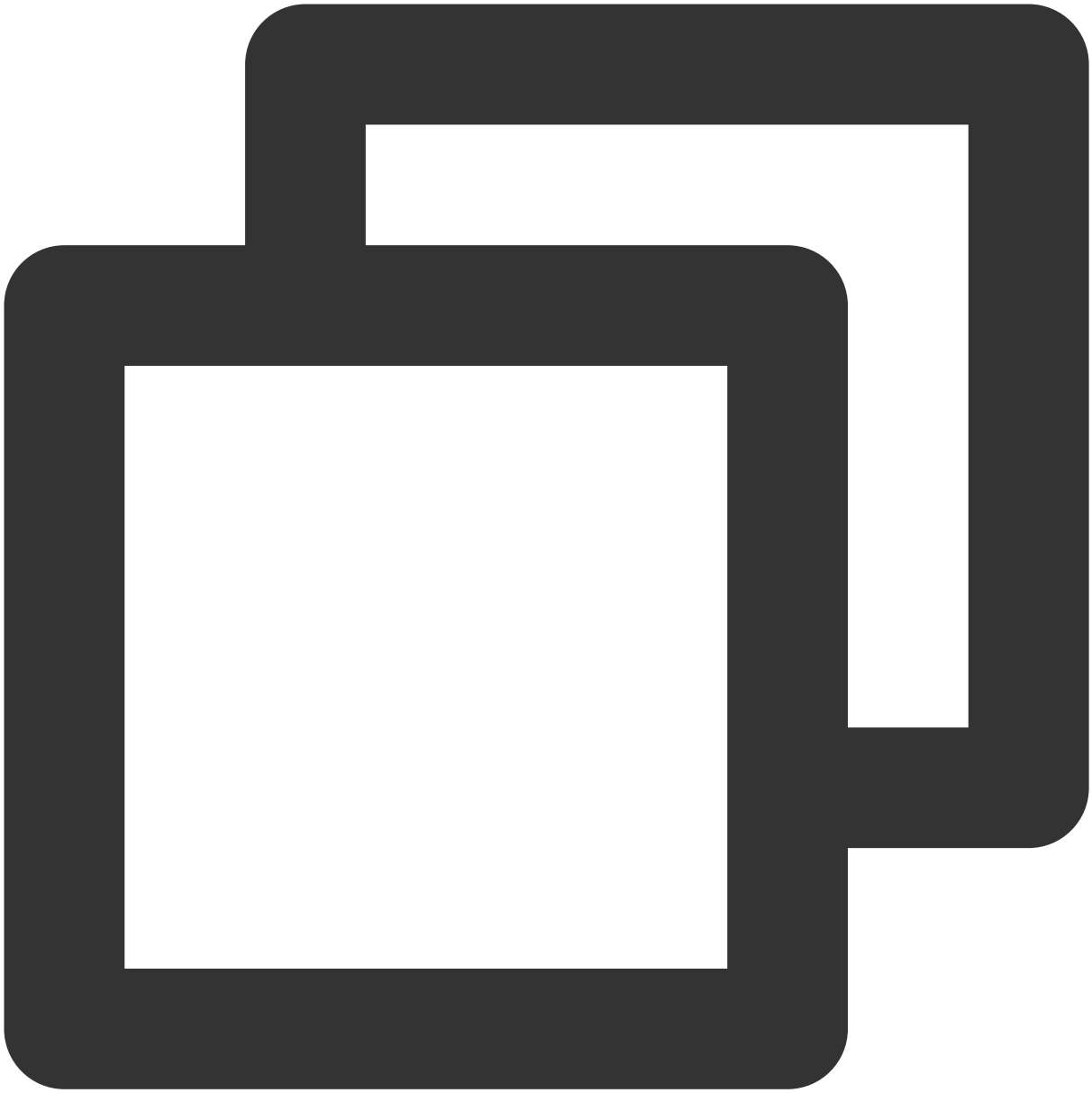


步骤一：下载并导入组件

1. 在 Github 中克隆/下载代码，然后拷贝 Android 目录下的 tuiofflinepush 子目录到您当前工程中的 app 同级目录中，如下图：

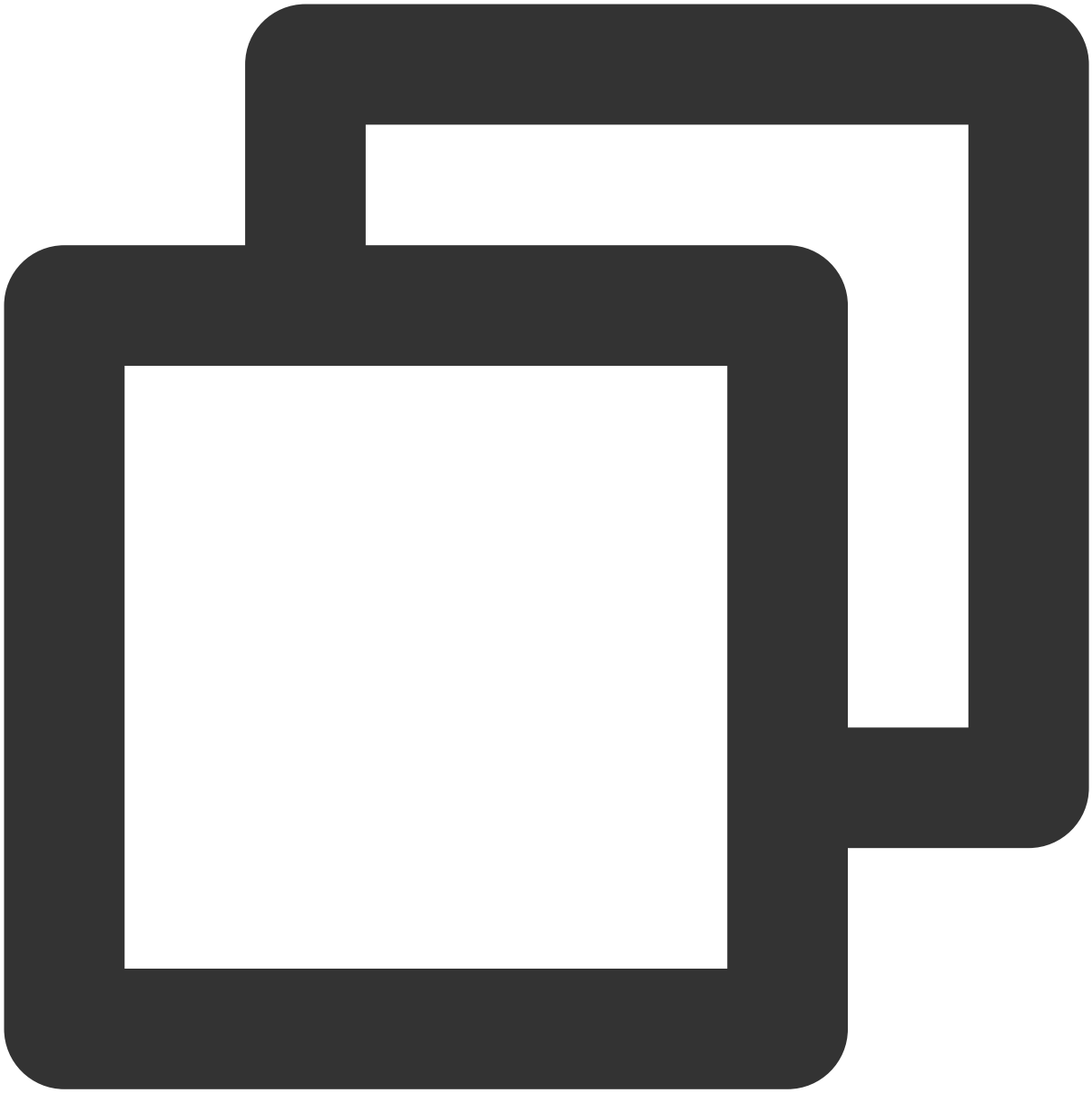


2. 在工程根目录下找到 `settings.gradle` 文件，并在其中增加如下代码：



```
include ':tuiofflinepush'
```

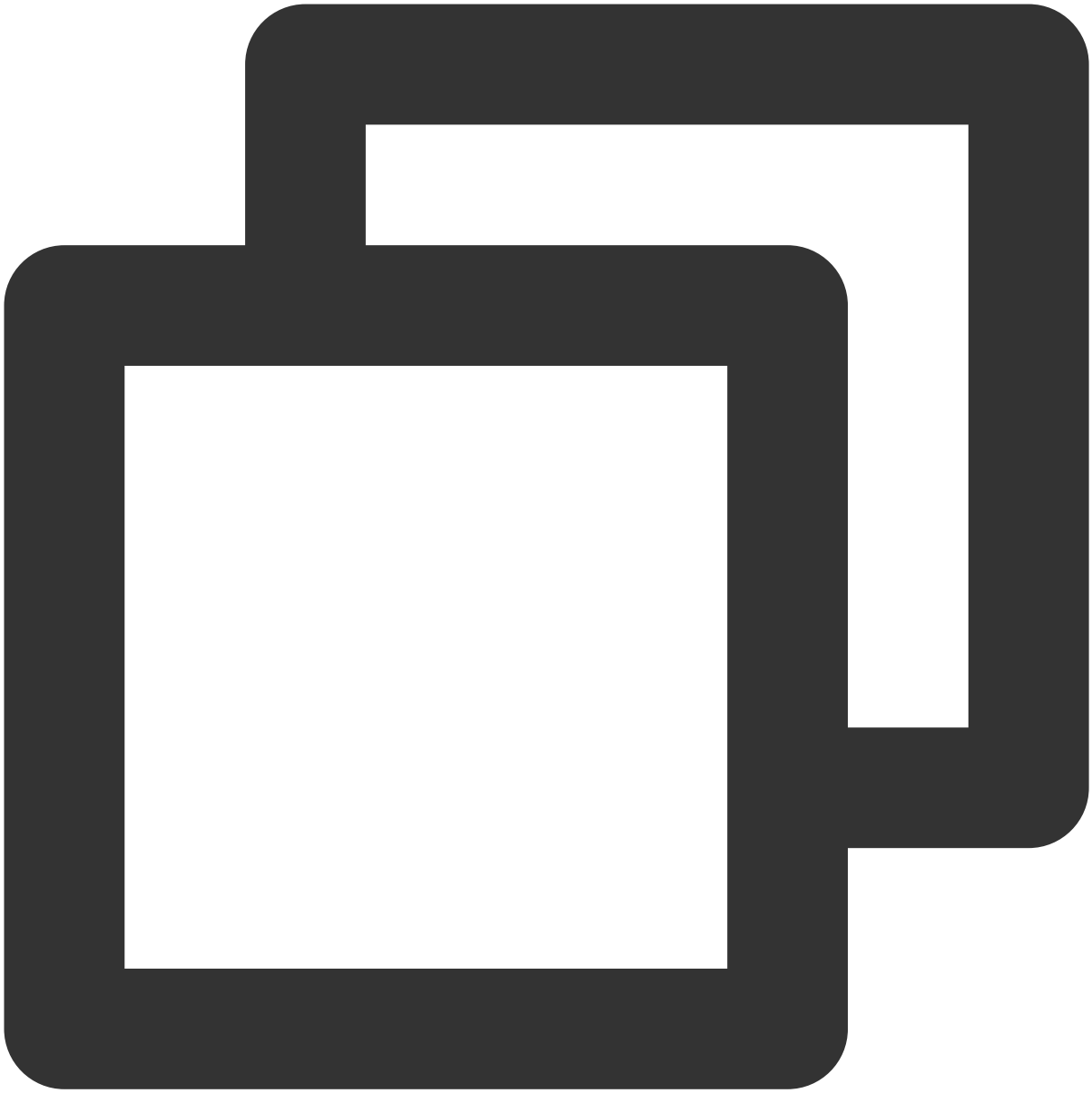
3. 在 app 目录下找到 `build.gradle` 文件，并在其中增加如下代码，它的作用是声明当前 app 对新加入的 tuiofflinepush 组件的依赖：



```
api project(':tuiofflinepush')
```

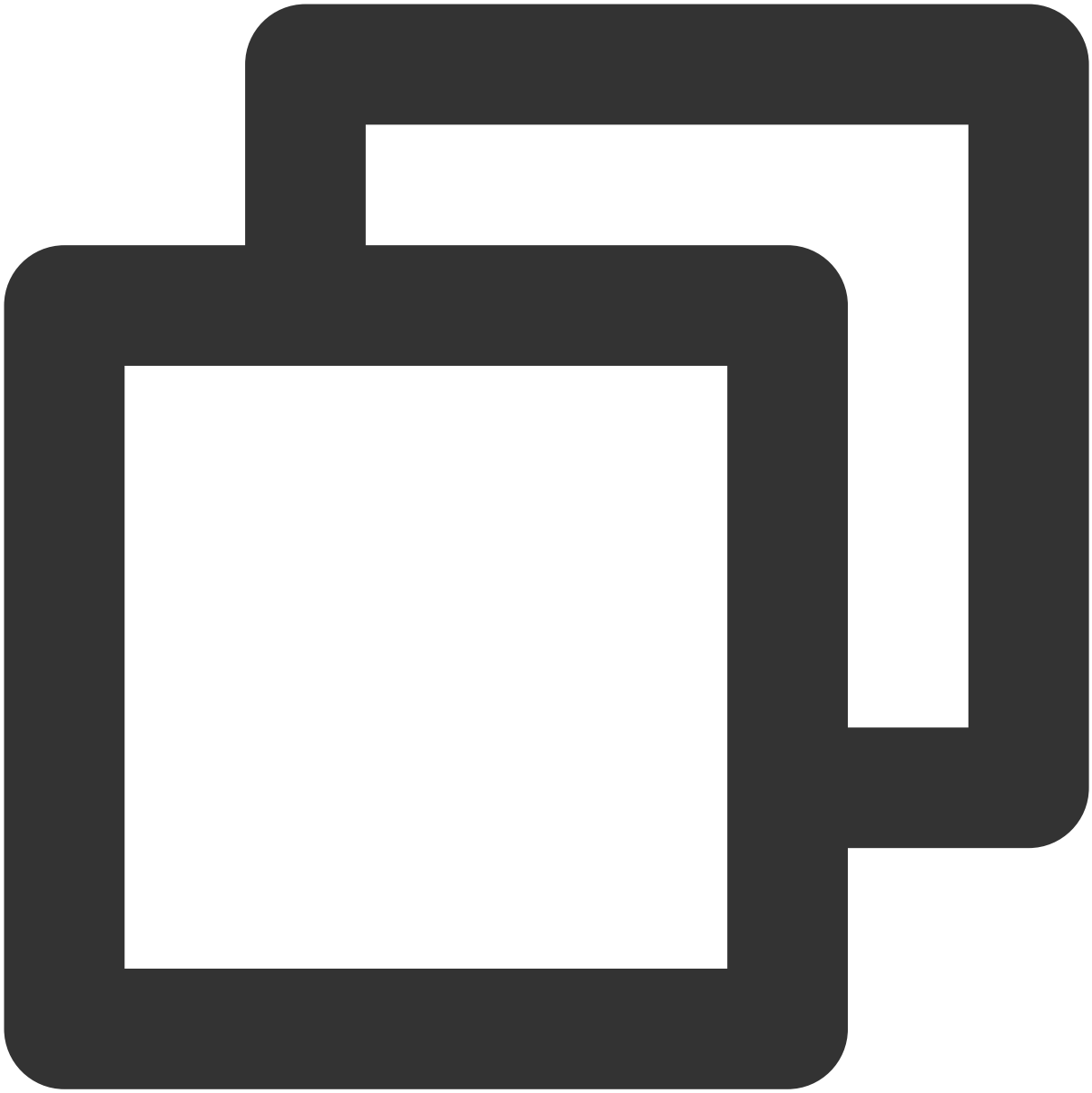
步骤二：完成工程配置

1. 在 app 目录下，找到 `build.gradle` 文件，修改应用包名为自己的包名。



```
applicationId 'com.****.trtc'
```

2. 在 app 目录下，找到 `build.gradle` 文件，设置 `ViVo` 接入参数 `VIVO_APPKEY` 、 `VIVO_APPID` 以及 `HONOR_APPID` ，避免编译或运行报错。



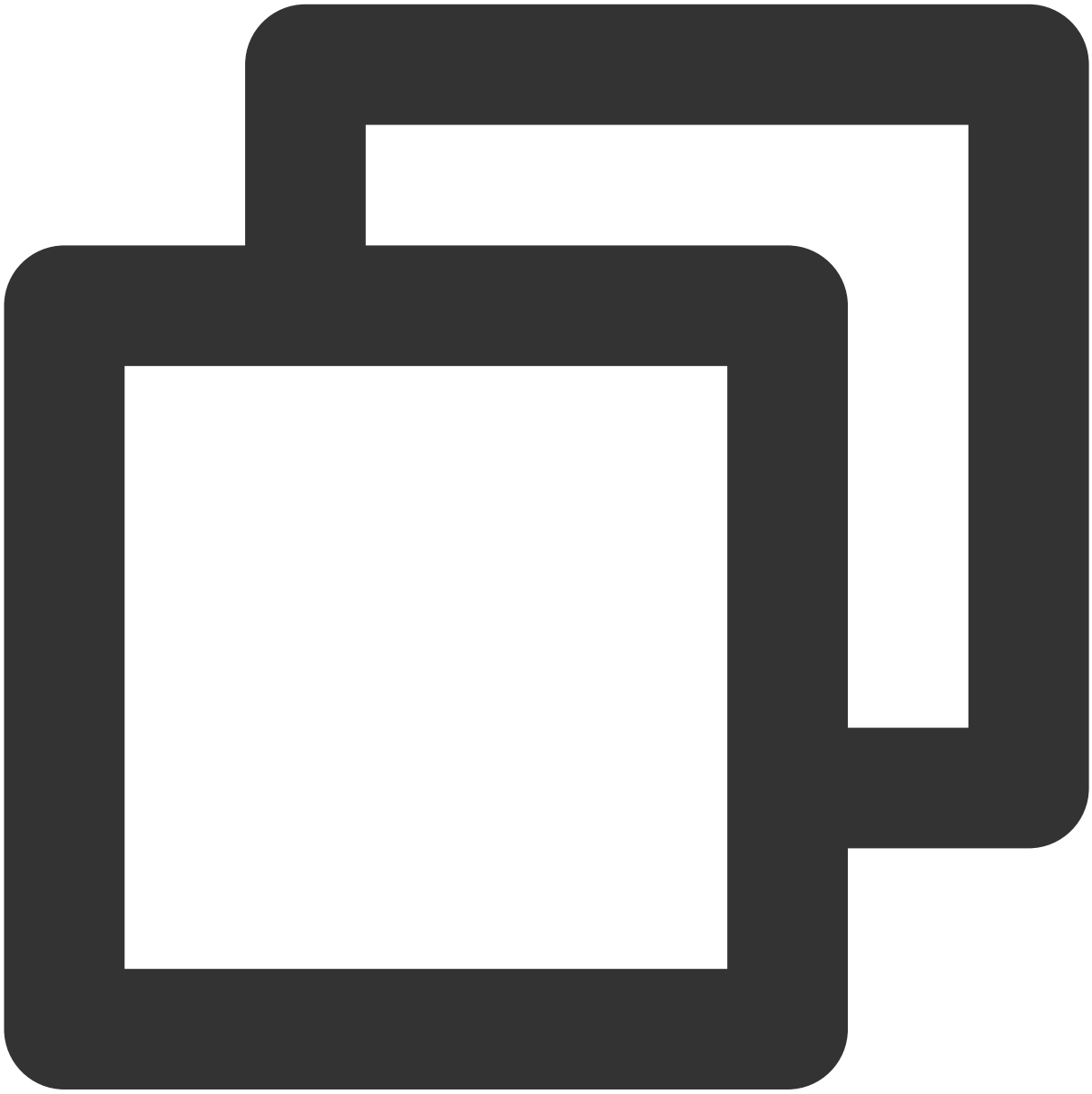
```
manifestPlaceholders = [
    "VIVO_APPKEY": "PLACEHOLDER",
    "VIVO_APPID" : "PLACEHOLDER",
    "HONOR_APPID": "PLACEHOLDER"
]
```

3. 配置 **华为** 和 **谷歌** 文件：

在 app 目录下，替换 `google-services.json` 文件，该文件为您在 前期准备 注册 Google 平台的时候保存的文件。

在 app 目录下，增加 `agconnect-services.json` 文件，该文件为您在 前期准备 注册华为平台的时候下保存的文件。

4. 将 前期准备 记录的 ID、APPID 和 APPKEY 填入 `PrivateConstants` 文件中，并检查参数是否配置正确。**填充的参数如下：**



```
public class PrivateConstants {
    /****** 小米离线推送参数start ******/
    // 在腾讯云控制台上传第三方推送证书后分配的证书 ID
    public static final long XM_PUSH_BUZID = 您应用分配的证书 ID;
    // 小米开放平台分配的应用APPID及APPKEY
    public static final String XM_PUSH_APPID = "您应用分配的 APPID";
```

```
    public static final String XM_PUSH_APPKEY = "您应用分配的 APPKEY";
    /****** 小米离线推送参数end ******/
}
```

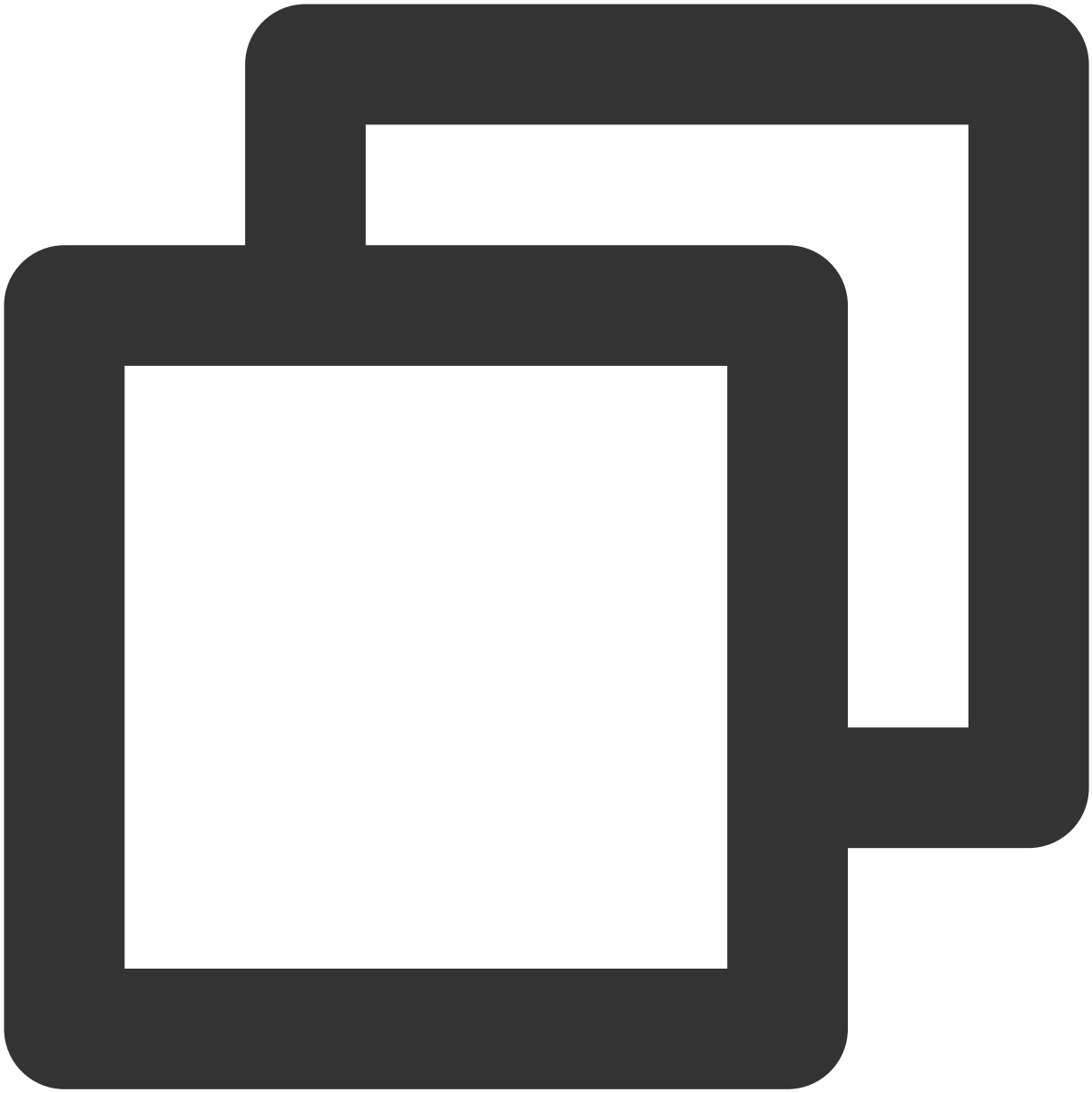
注意

这一步非常重要，请认真检查参数是否配置正确。

完成以上步骤，您的工程就具备了 `TUICallKit` 的离线唤醒功能。

步骤三：自定义离线通知内容

TUICallKit 提供了默认的通知样式，但是如果您需要自定义通知内容，请修改 OfflinePushInfoConfig.java 文件 。



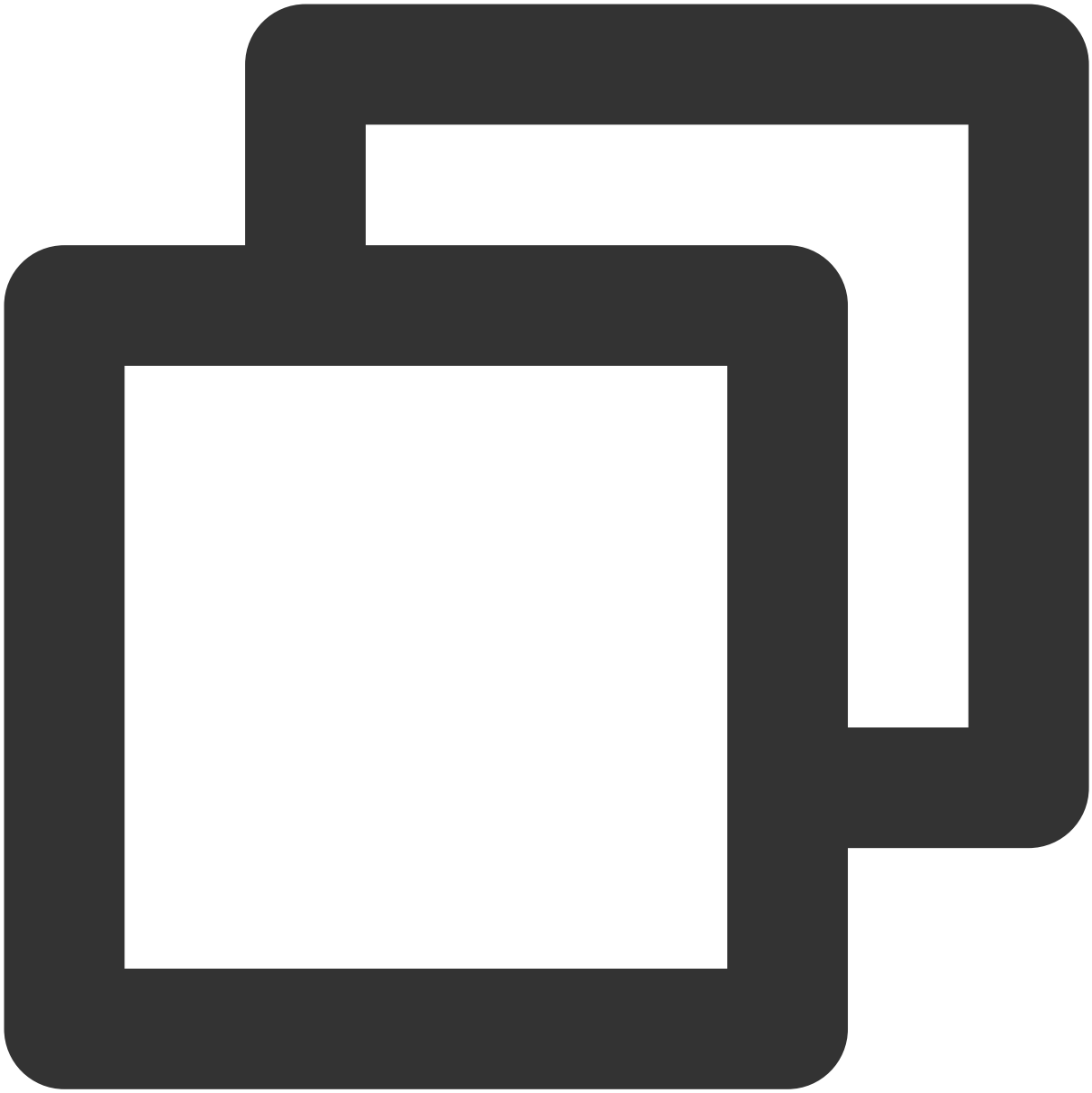
```
public static TUICallDefine.OfflinePushInfo createOfflinePushInfo(Context context)
    TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
    pushInfo.setTitle("mike");
    pushInfo.setDesc("You have receive a new call");
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.setAndroidOPPOChannelID("tuikit");
    pushInfo.setIgnoreIOSBadge(false);
    pushInfo.setIOSSound("phone_ringing.mp3");
    pushInfo.setAndroidSound("phone_ringing"); //Note：don't add suffix
    //VIVO message type: 0-push message, 1-System message(have a higher delivery ra
```

```
    pushInfo.setAndroidVIVOClassification(1);
    //FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannel
    pushInfo.setAndroidFCMChannelID("fcm_push_channel");
    //Huawei message type
    pushInfo.setAndroidHuaWeiCategory("IM");
    //IOS push type: if you want user VoIP, please modify type to TUICallDefine.IOS
    pushInfo.setIOSPushType(TUICallDefine.IOSOfflinePushType.APNs);
    return pushInfo;
}
```

步骤四：自定义离线通知铃音

离线通知铃声仅支持以下厂商自定义：**华为、小米、FCM 和 APNs**，其他厂商 OPPO、VIVO、荣耀等暂不支持。

TUICallKit 引入了 TUIOfflinePushPush 组件，需在应用启动的时，调用以下方法开启离线通知自定义铃音能力。

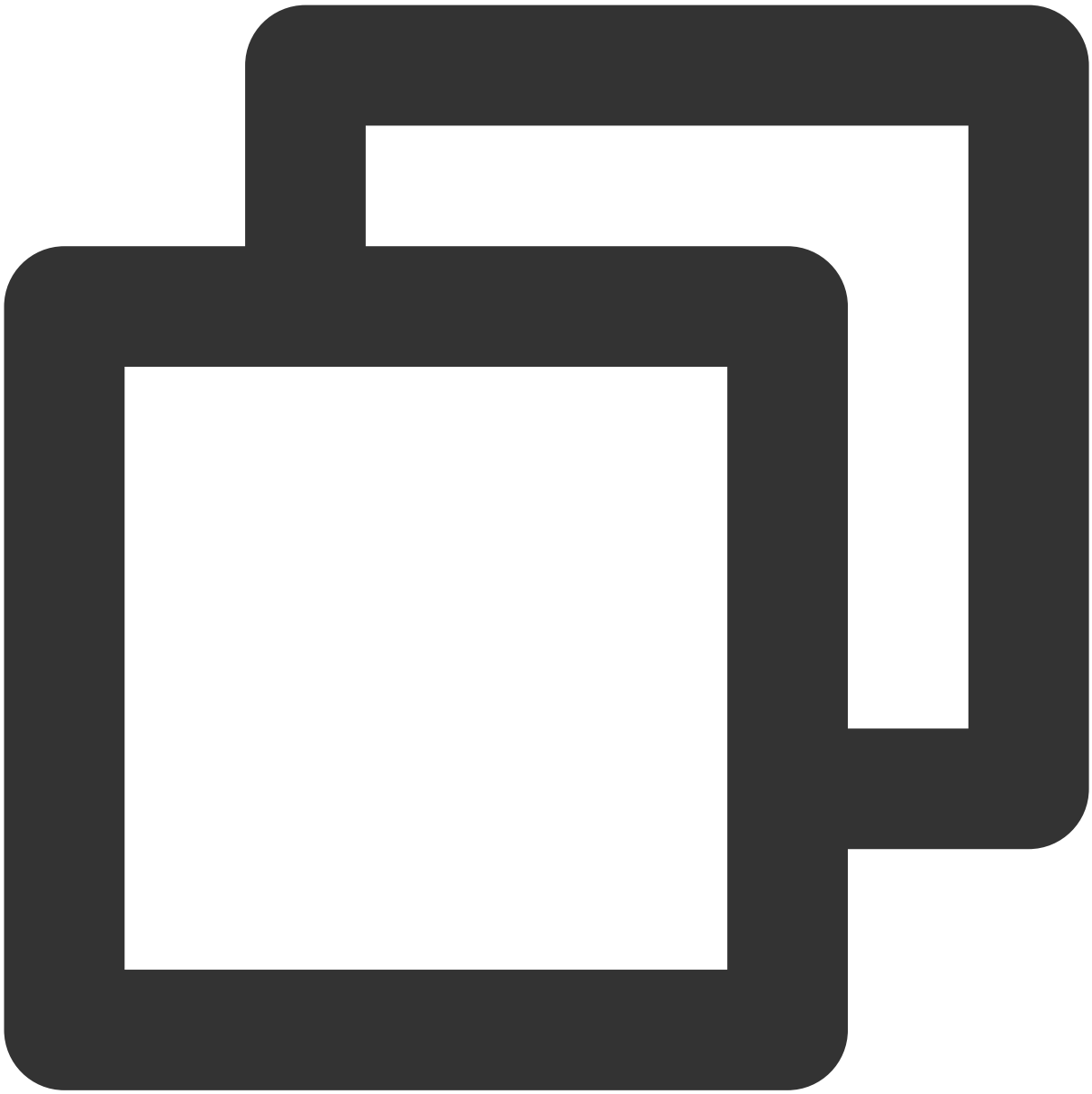


```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        TUIOfflinePushConfig.getInstance().setAndroidPrivateRing(true);
    }
}
```

1、华为 & APNs

调用接口 setAndroidSound() 和 setIOSSound()。

定制铃音资源文件，将铃音文件添加到本地 Android Studio 工程 res/raw 目录下。参考 步骤三 设置铃声。



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
```

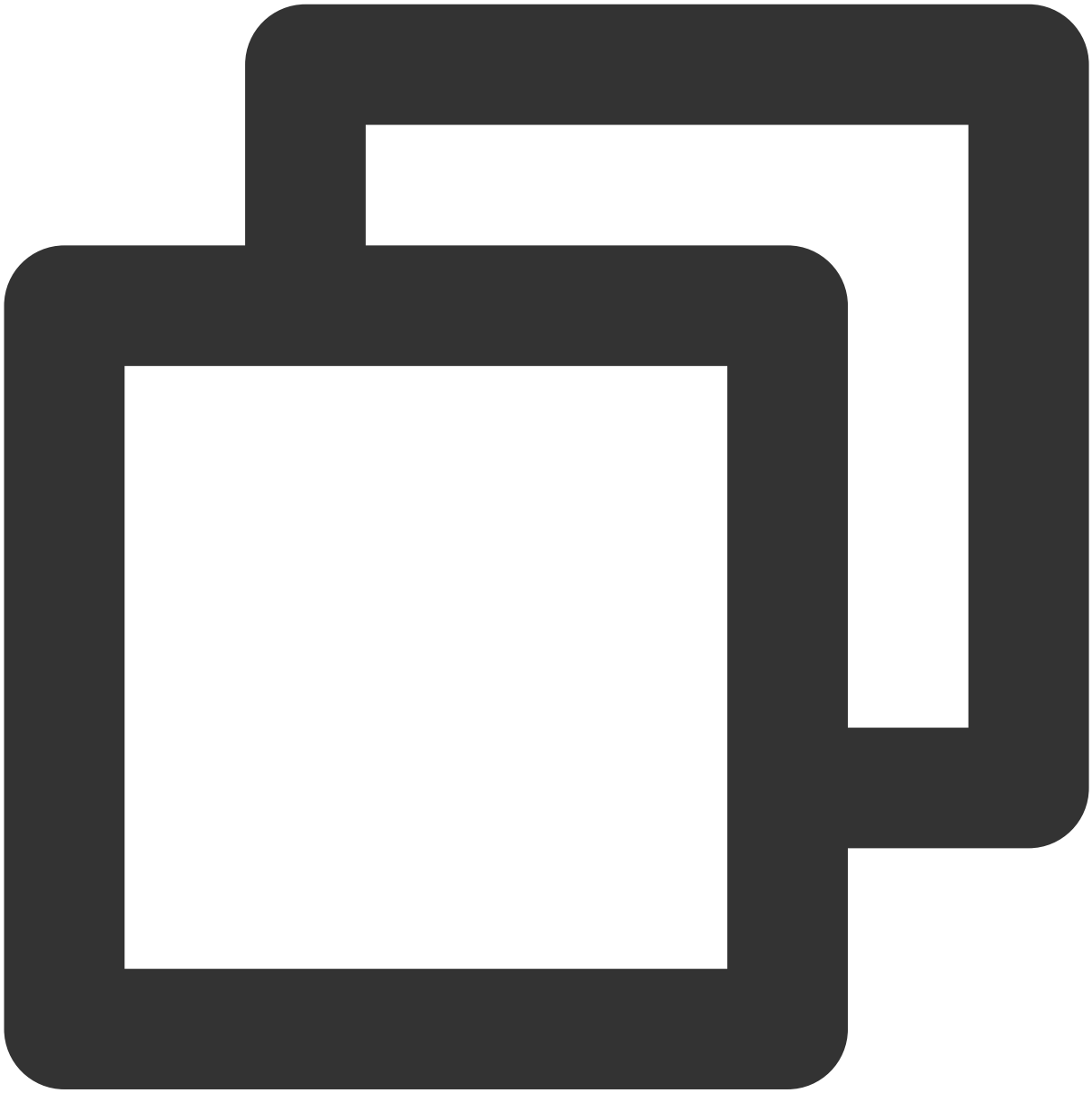
注意

IMSDK 6.1.2155 及以上版本支持。

2、小米

（1）Android 8.0 之前，调用setAndroidSound() 和 setIOSSound()。定制铃音资源文件，将铃音文件添加到工程 res/raw 目录下（参考上述华为调用）。

（2）Android 8.0 之后，还需要登录小米厂商控制台并创建channel并做好配置，其中铃音文件需要添加到本地 Android Studio 工程的 res/raw 目录下。



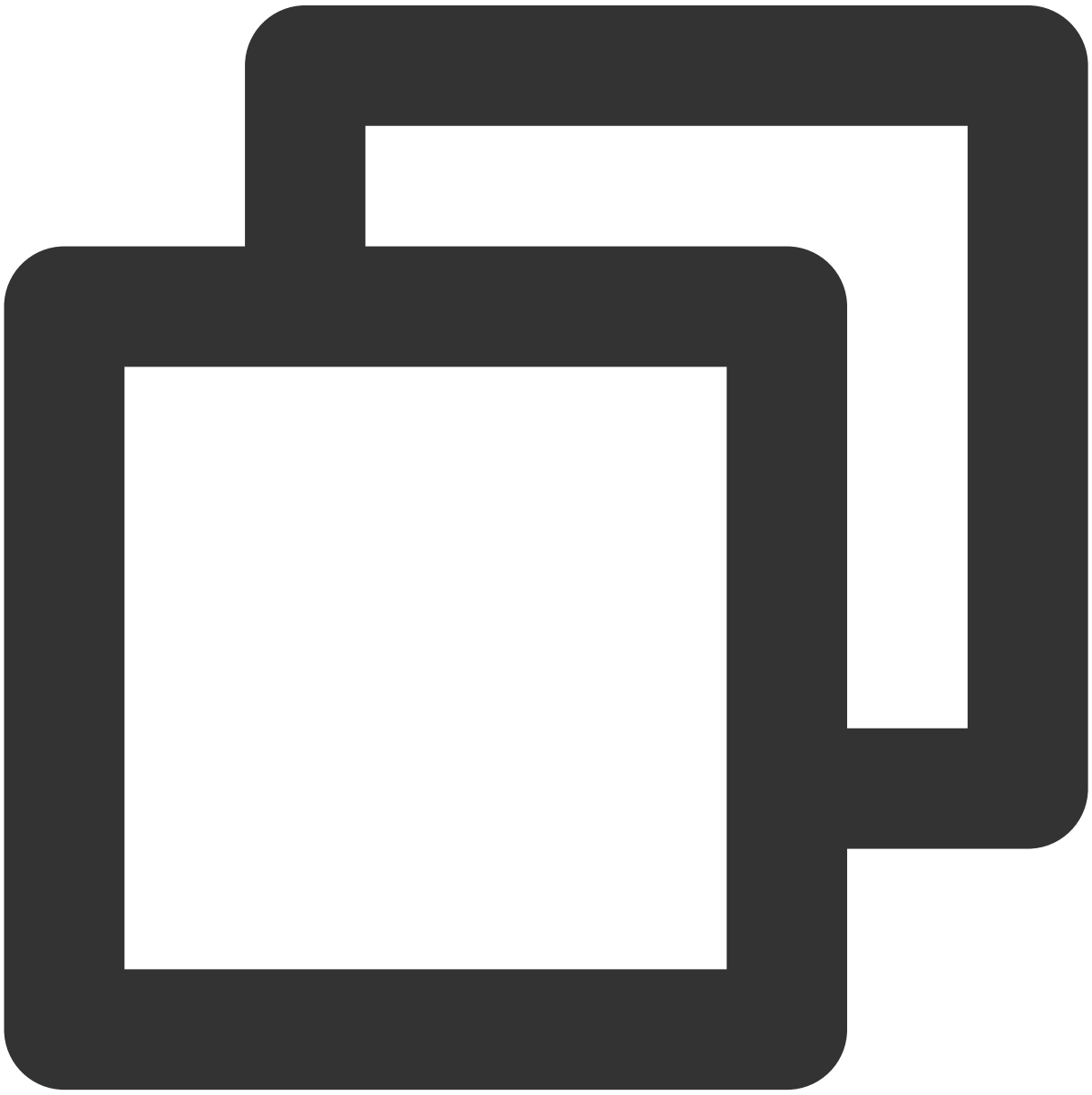
```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
pushInfo.setAndroidXiaoMiChannelID("厂商申请的channelID");
```

注意

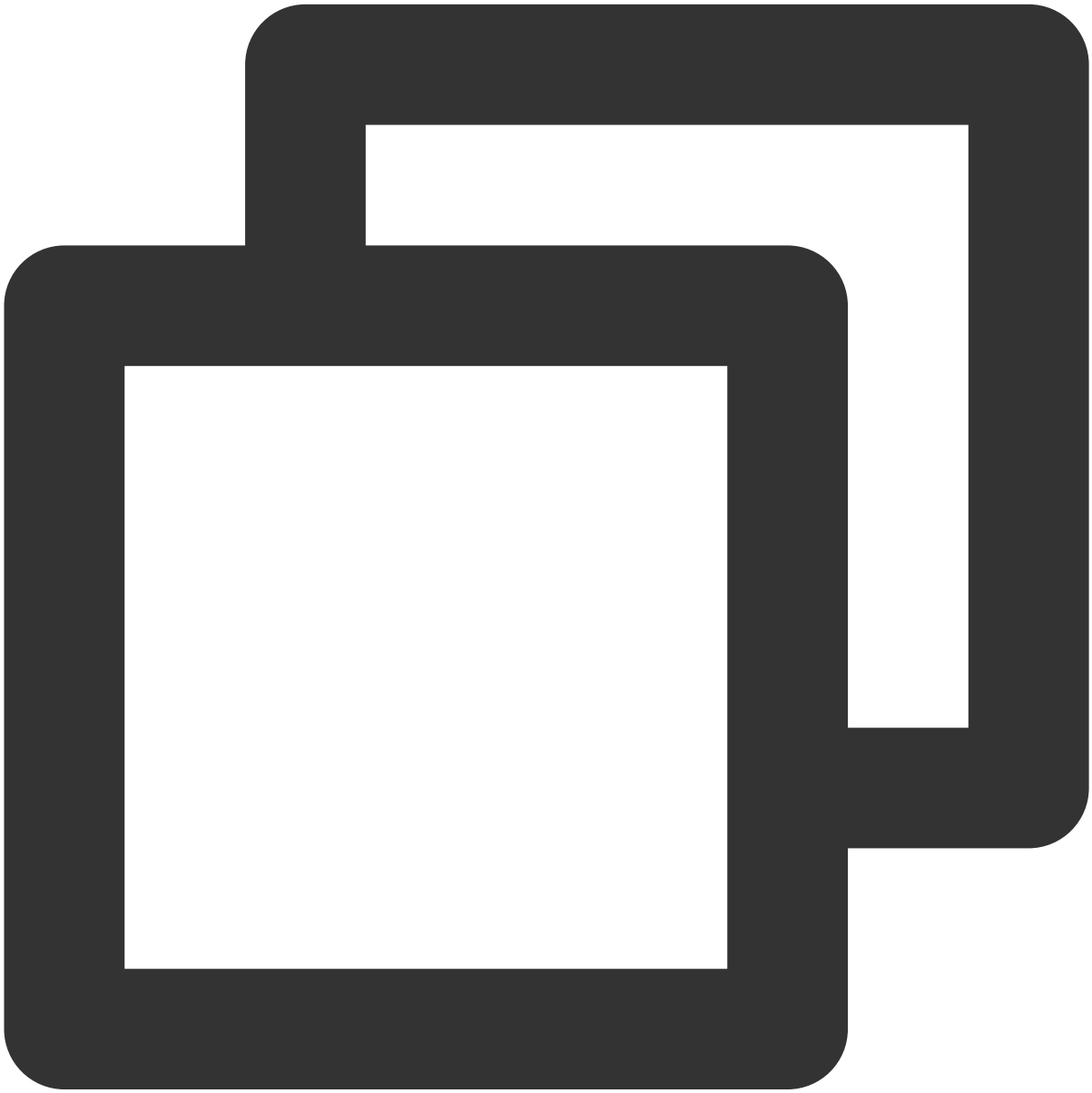
Android 8.0之前的方式，IMSDK 6.1.2155 及以上版本支持。
Android 8.0之后的方式，IMSDK 7.0.3754 及以上版本支持。

3、FCM

（1）Android 8.0 之前，调用 setAndroidSound() 和 setIOSSound()。定制铃音资源文件，将铃音文件添加到工程 res/raw 目录下（参考上述华为调用）。
（2）Android 8.0之后，FCM 需要配置 channelID 以及铃音资源，TUICallKit 引入的 tuiofflinepush 组件已经处理了自定义铃声的播放。铃音文件需要添加到您本地 Android Studio 工程的 res/raw 目录下，并指定铃音名称和 channel ID 的名称，详见 PrivateConstants。



```
public class PrivateConstants {
    // FCM 通道指定 channel ID
    public static String fcmPushChannelId = "FCM ChannelID";
    // FCM 需指定通道的推送铃音名称，与AndroidStudio 工程里 res/raw 目录中的铃音文件名一致，不
    public static String fcmPushChannelSoundName = "铃声名称";
}
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
pushInfo.setAndroidFCMChannelID("厂商申请的channelID");
```

注意

Android 8.0之前的方式，IMSDK 6.1.2155 及以上版本支持。

Android 8.0之后的方式，IMSDK 7.0.3754 及以上版本支持。

FCM 自定义铃声或者设置 channnel id 仅支持证书模式。

常见问题

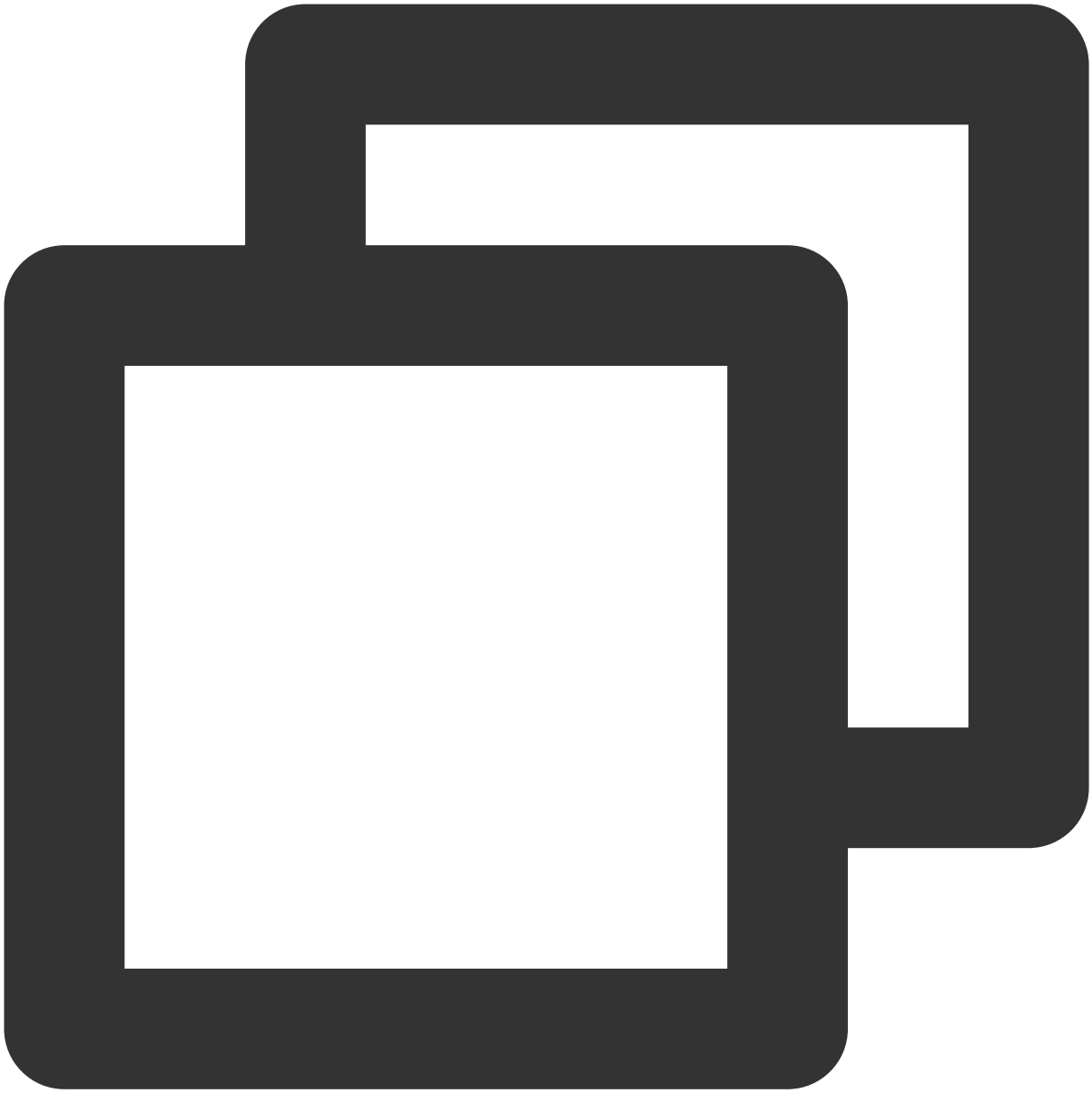
收不到离线推送通知，请 联系我们 进行排查。

1、收不到通知

用厂商控制台进行推送测试，能成功说明厂商通道没有问题。再检查 TUIOfflinePush 控制台厂商参数配置是否正确，按要求进行填写。（经测试：vivo x9必须在控制台配置消息类别）。

部分手机收到通知会放到 `不重要的通知` 中，请下拉状态栏，检查是否归纳到 `不重要的通知` 中。

检查 TUIOfflinePush 注册是否成功，过滤以下日志：



```
TUIOfflinePush
```

2、锁屏时无法点亮屏幕

Android 手机由于厂商和平台的限制，在锁屏情况下需要不同的权限。请按以下情况进行排查。

确认打开厂商锁屏下通知权限

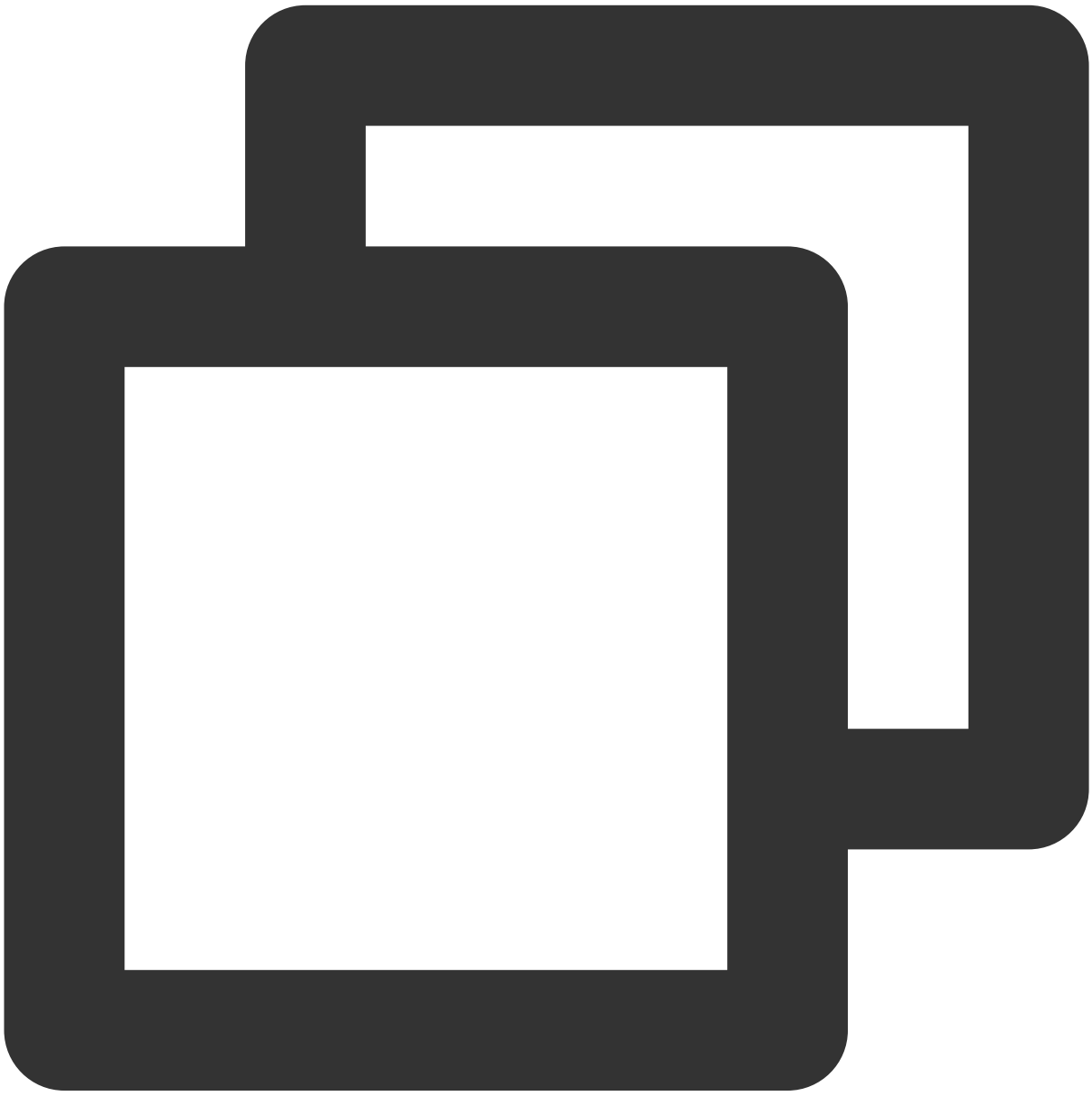
部分厂商统一做了约束，例如小米锁屏下离线通知到达时未亮屏：在**设置 > 锁屏**里，单击开关**锁屏来通知时亮屏**，打开开关。

确认打开应用锁屏通知权限

例如：小米需要锁屏显示权限。

3、单击离线推送通知，拉不起通话界面

需要检查下是否查到了通话请求，可以过滤以下日志：



```
onReceiveNewInvitation
```

4、应用在后台时，不能自动将通话界面拉取到前台

将应用从后台自动拉取到前台，需要检查 App 是否开启了”后台自启动“或”悬浮窗“权限。

值得注意的是，不同厂商、甚至同一厂商不同 Android 版本，其对于应用开放的权限以及权限名称也会存在不一致。

例如，小米6只需要开启**后台弹出界面**权限，而红米需要同时打开**后台弹出界面**和**显示悬浮窗**权限。

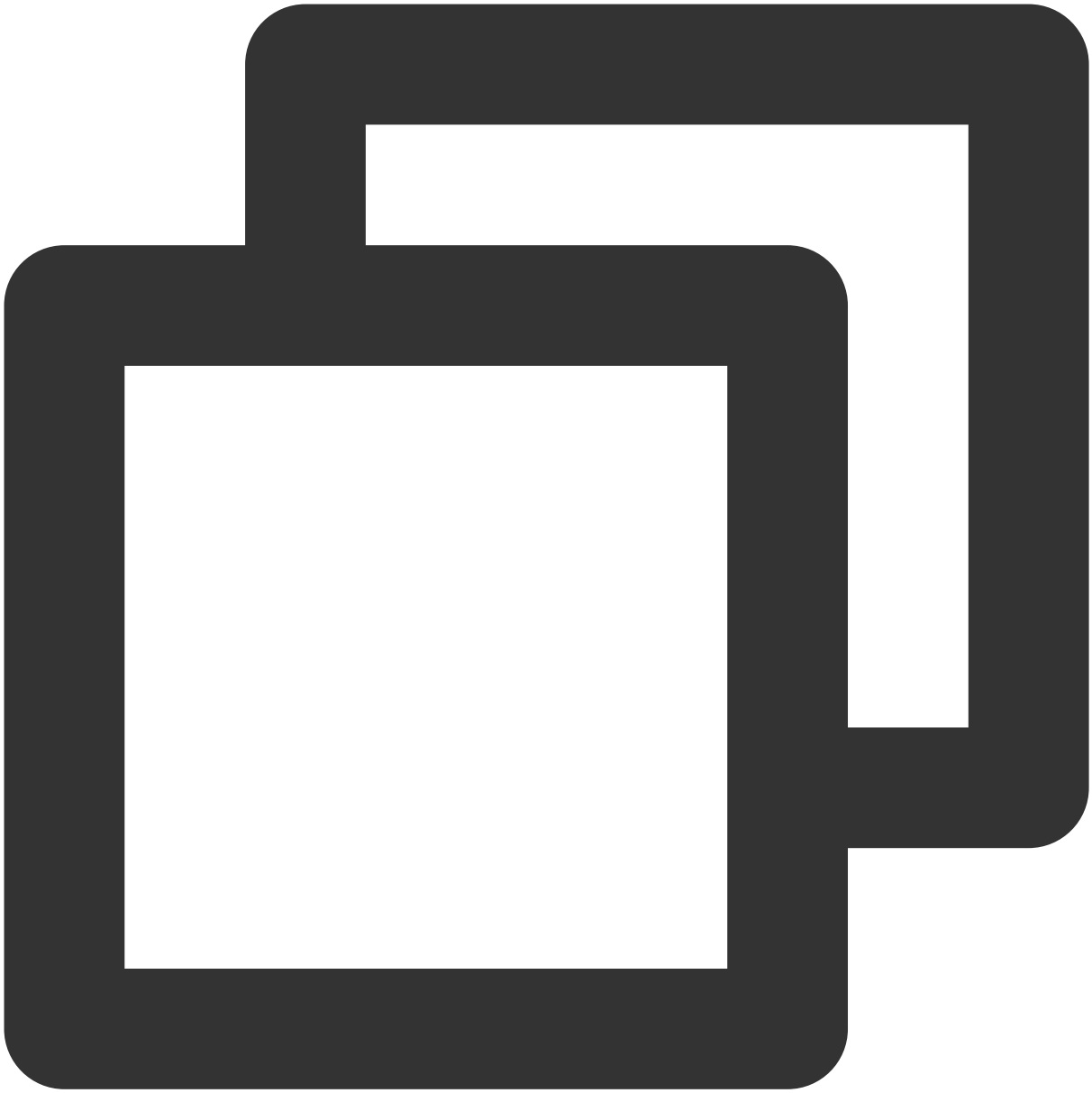
说明：

如果您在测试过程中发现手动开启了所有权限，依然无法自动拉起通话界面到前台，需要做兼容处理。

5、角标问题

目前仅 **华为** 支持收到离线消息时生成角标，其他厂商暂不支持。华为设置角标方法详见其官方文档：华为桌面角标开发指导。

在引入的 TUIOfflinePush 组件中，也调用了该方法进行设置：



```
public void updateBadge(final Context context, final int number) {
    if (BrandUtil.getInstanceType() != TUIOfflinePushConfig.BRAND_HUAWEI) {
        return;
    }
    try {
        Bundle extra = new Bundle();
        extra.putString("package", context.getPackageName());
        extra.putString("class", PrivateConstants.huaweiBadgeClassName); //huaweiBa
        extra.putInt("badgenumber", number);
        context.getContentResolver().call(Uri.parse("content://com.huawei.android.l
    } catch (Exception e) {
```

```
        }
    }
```

如果想要清除角标，可以在适当的时机调用该方法，将角标设置为0。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-03-12 14:51:07

离线唤醒功能，能够让您的 App 在后台运行或者离线状态下依然能够收到音视频通话的响铃呼叫，TUICallKit Flutter 需要依赖厂商： Apple、Google、Vivo、Xiaomi 提供的推送能力来进行消息通知。

配置离线推送

iOS

iOS 平台的离线唤醒支持 `APNs` 和 `VoIP` 具体配置流程请参考：

APNs 离线唤醒配置流程

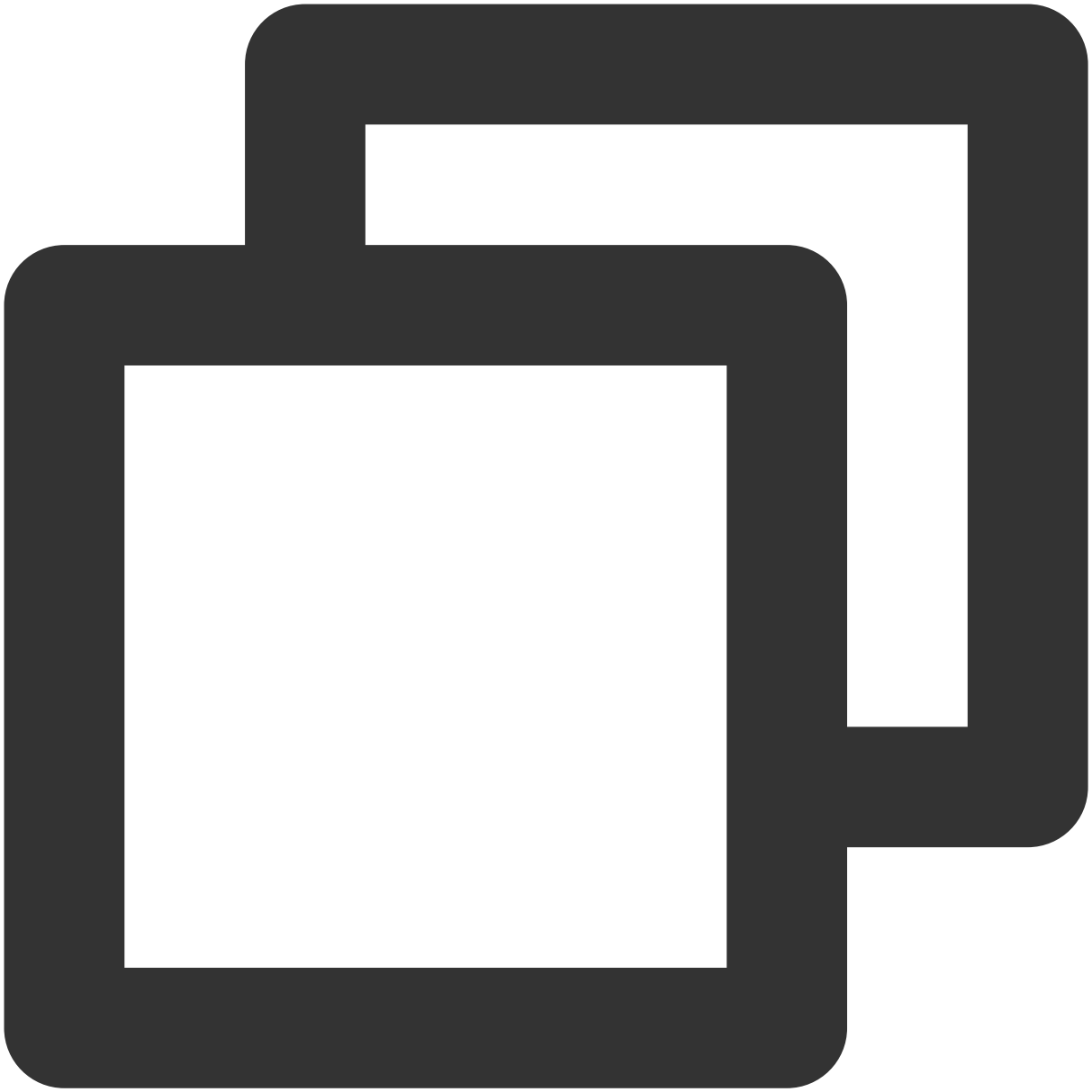
VoIP 离线唤醒配置流程

Android

Android 平台具体配置流程，请参考：离线推送（Flutter）。

自定义离线消息内容

在基础的离线推送配置成功后，您就可以正常收到对来电推送，如果您需要自定义离线消息的内容，可通过 `call` 和 `groupcall` 接口中的参数 `params` 中的 `TUIOfflinePushInfo` 实现自定义通知内容，可以自定义的内容包括**标题**、**内容**、**离线消息铃声**等内容。具体使用如下：



```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa

class TUICallParams {
    TUIOfflinePushInfo offlinePushInfo;
}

class TUIOfflinePushInfo {
    String? title;
    String? desc;
    bool? ignoreIOSBadge;
    String? iOSSound;
```

```
    String? androidSound;
    String? androidOPPOChannelID;
    int? androidVIVOClassification;
    String? androidXiaoMiChannelID;
    String? androidFCMChannelID;
    String? androidHuaWeiCategory;
    bool? isDisablePush;
    TUICallIOSOfflinePushType? iOSPushType;
}}

enum TUICallIOSOfflinePushType {
    APNs,
    VoIP,
}
```

自定义离线消息铃声

iOS

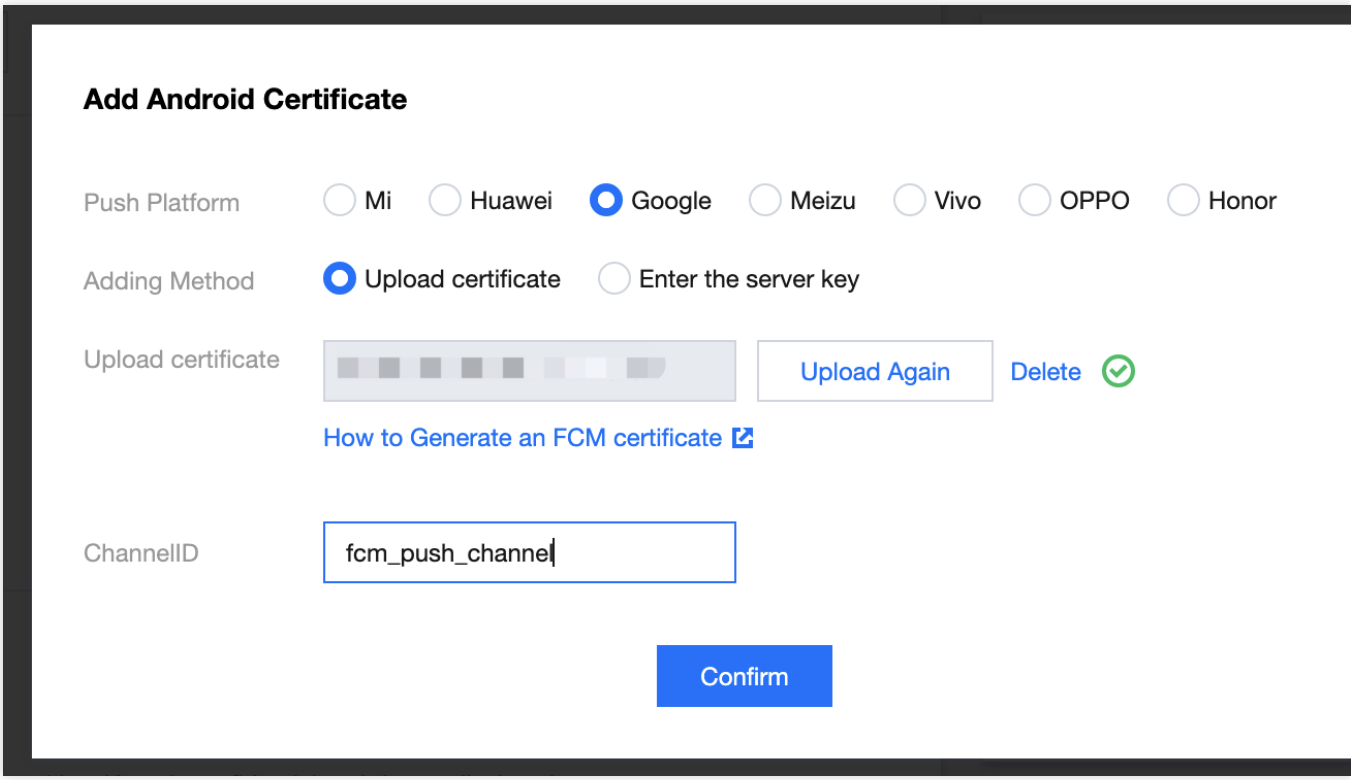
修改 `call` 和 `groupcall` 接口中的参数 `params` 中的 `TUIOfflinePushInfo.iOSSound` 设置 `iOS` 平台上的离线消息铃声。

Android

FCM

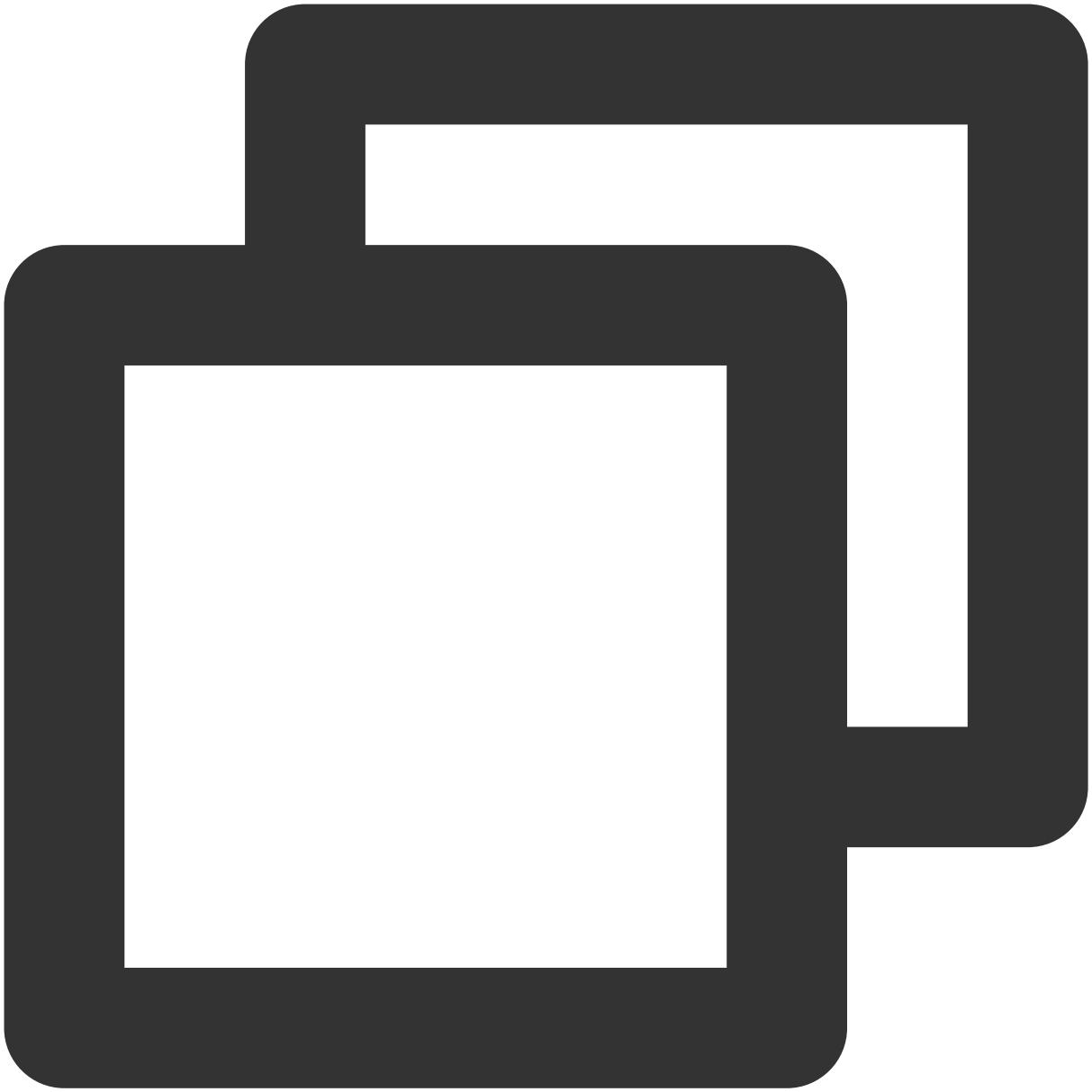
在完成第一步：配置离线推送 之后，即可在离线的情况下收到消息推送，但是是默认的铃声，如果您需要支持自定义铃声，还需要进行如下操作。

1. 在IM 控制台-基础配置，选择 Android 原生离线推送配置 > 选择 Google 的证书进行编辑，添加 ChannelID，如下图所示：



2. 在工程的 `android/app/src/main/res/raw/` 目录下，放入自定义的铃声文件，示例："incoming_call.mp3"。

3. 在登录成功之后，通过 `tim_ui_kit_push_plugin` 插件 向 Android 系统注册播放自定义铃音的 NotificationChannel，示例代码如下：



```
void createNotificationChannel() {
  AndroidNotificationChannelGroup group = new AndroidNotificationChannelGroup("Grou
  AndroidFlutterLocalNotificationsPlugin plugin = AndroidFlutterLocalNotificationsP
  plugin.createNotificationChannelGroup(group);

  AndroidNotificationChannel channel = AndroidNotificationChannel(
      "fcm_push_channel", "ChannelName",
      groupId: "GroupID",
      importance: Importance.high,
      enableLights: true,
      enableVibration: true,
```

```
        sound: RawResourceAndroidNotificationSound("incoming_call")); // Note: Media
    plugin.createNotificationChannel(channel);
  }
```

云端录制（TUICallKit）

最近更新时间：2024-04-03 17:23:11

本文将介绍如何开启 TUICallKit 的云端录制，方便重要通话的存档、审核等，我们提供了两个方案供您选择：自动录制方案 和 REST API 录制方案。

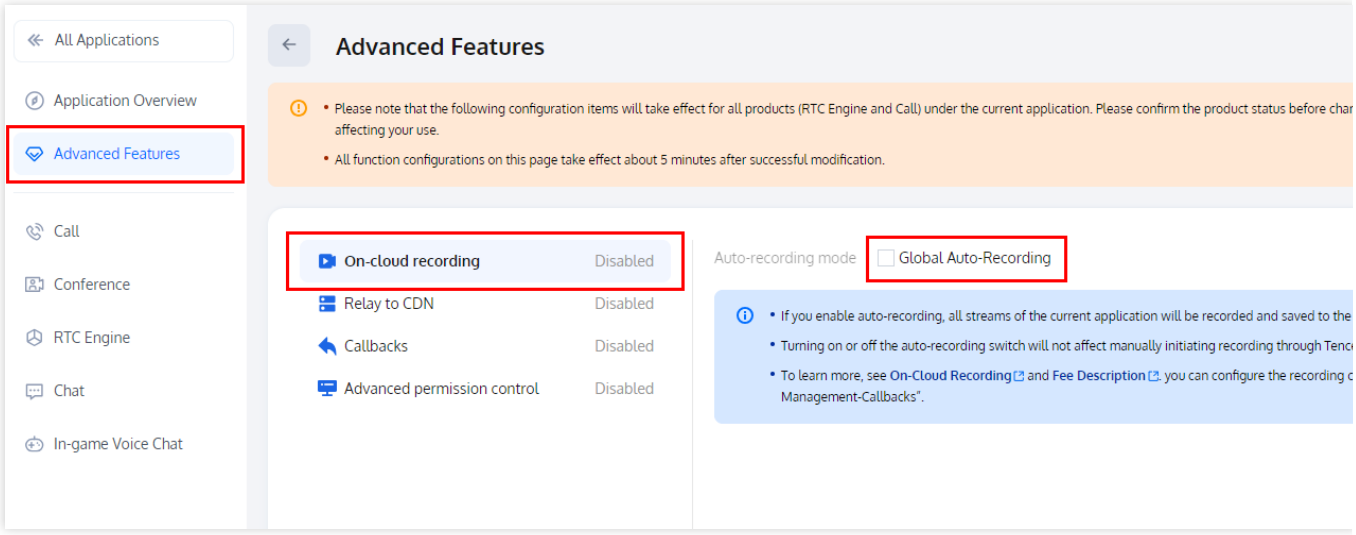
说明：

TUICallKit 整合有多个腾讯云基础的 PaaS 服务，其中音视频相关能力依赖于实时音视频 TRTC，因为 TUICallKit 的云端录制功能需要进入 实时音视频控制台 进行配置。

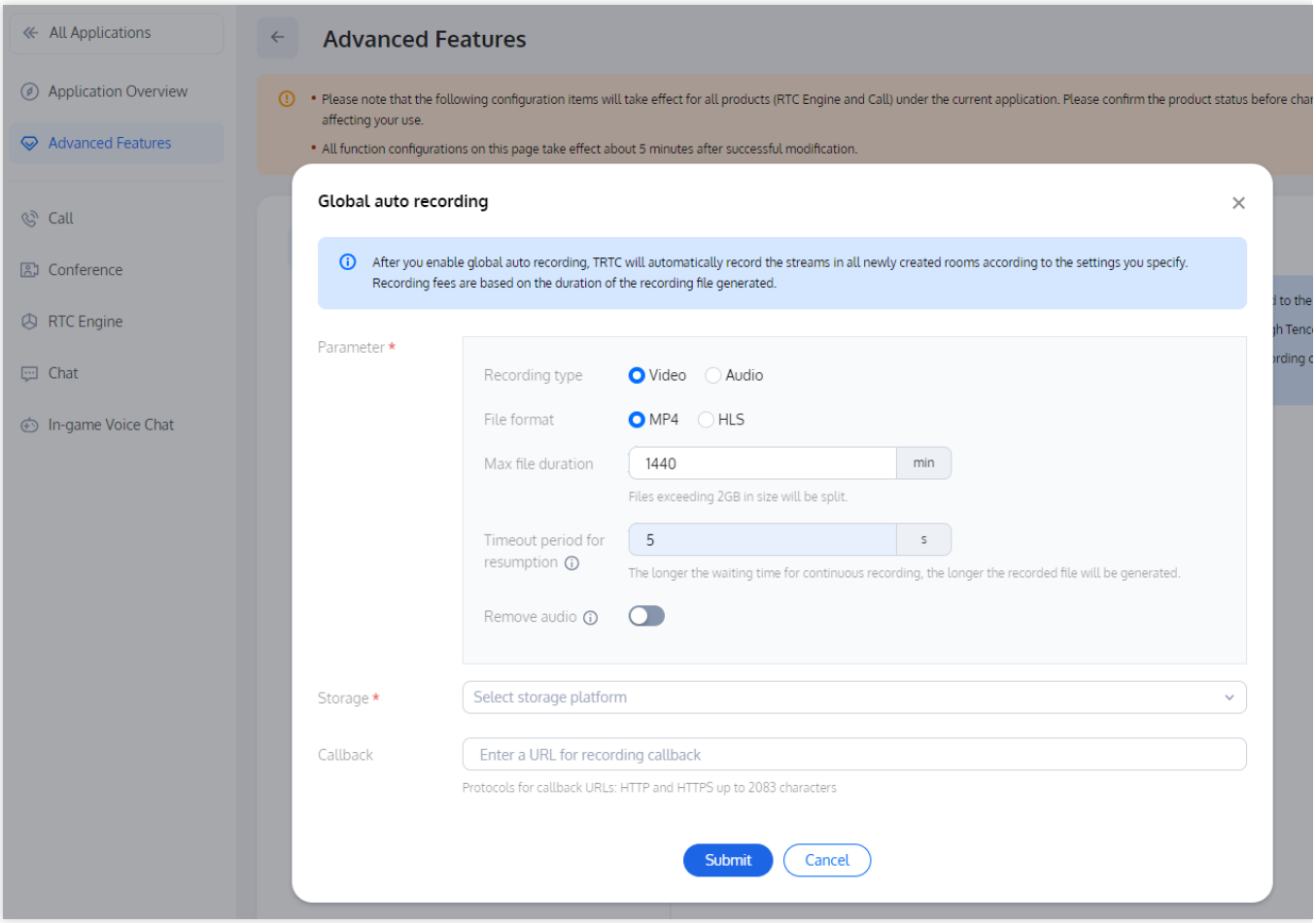
方案一：自动录制方案（推荐）

我们推荐您使用自动录制的方案，录制不需业务方来启动和停止，由腾讯云实时音视频后台来管理录制任务，通话过程中有音视频流上行时就自动录制，**接入过程快速、简单**，您可以通过如下几步完成：

1. 在 实时音视频控制台 > 应用管理 找到对应 SDKAppId 的应用，进入**高级功能**配置页。
2. 在**高级功能**配置页，您可以看到云端录制配置的选项，点击**全局自动录制**进入配置浮窗。



3. 根据音视频通通话的业务场景（1v1通话、群组通话），我们推荐如下配置参数，当然您也可以根据自己的业务需求配置自定义录制模版。



注意：

全局录制支持最大混流人数为 8，若您的通话人数超过8时（包括自己），最后一个用户的流无法录制。

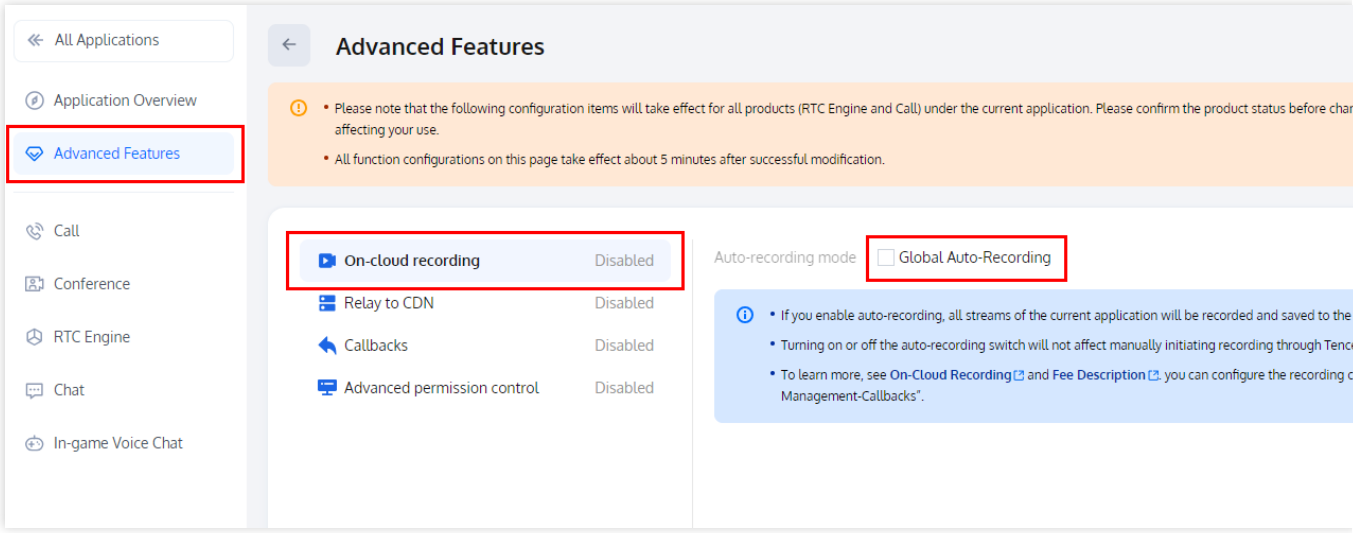
开启全局自动录制功能后，在通话接听后且有音视频上行，就会触发自动启动录制任务，在通话结束后会自动停止录制，如果因为网络或者其他情况异常退房，我们的录制后台会根据您设置的 MaxIdleTime值（空闲等待时间，默认5s）自动停止录制任务，避免造成额外的计费损失。

4. **模版创建完成后，勾选**全局自动录制即可完成。

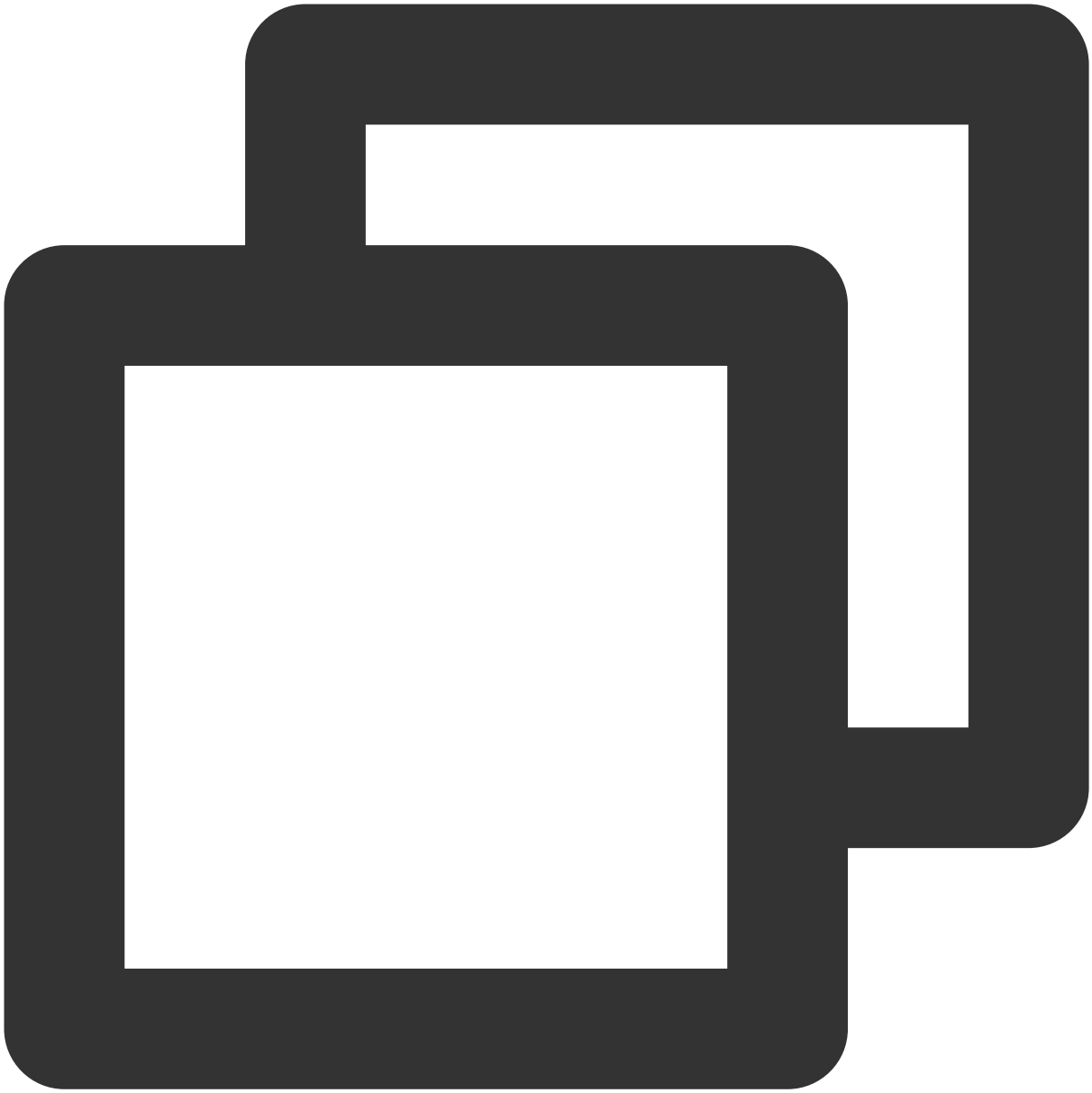
方案二：REST API 录制方案

如果自动录制方案不满足您的业务需求，您也可以采用更加灵活的 REST API 录制方案，可以指定录制订阅房间内的主播，自定义合流布局，录制中途更新布局和订阅等，但是**需要搭配业务后台服务，接入复杂，功能强大，关键步骤如下**：

1. 在 实时音视频控制台 > 应用管理 找到对应 SDKAppId 的应用，进入**高级功能**配置页。
2. 在**高级功能**配置页，您可以看到云端录制配置的选项，**启用云端录制**功能，此时默认勾选**手动自定义录制**，即 REST API 模式。



3. 然后您就可以调用 REST API （CreateCloudRecording）来启动云端的录制，这里建议您可以通过监听 TUICallObserver 的通知事件，在音视频通话开始的时候开启录制，以 Java 代码为例：



```
TUICallEngine.createInstance(context).addObserver(new TUICallObserver() {
    @Override
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType
        // 通知您的业务后台，使用REST API 启动录制任务。
    }
});
```

4. 考虑到客户端有可能会存在网络差、杀进程的情况造成的通话异常挂断，所以如何结束录制，建议您可以通过订阅 TRTC 房间状态的回调（详情请参见 监听服务端事件回调），在收到 TRTC 房间状态解散的回调后，调用 REST API （DeleteCloudRecording）来停止云端的录制任务。

常见问题

1. 如何查看录制时长明细？

您可以在 实时音视频控制台 > 云端录制查看录制的一些时长明细。

2. 如何查看录制的文件？

您可以登录 云点播控制台，在左侧导航栏选择**媒资管理，**单击列表上方的**前缀搜索**，选择**前缀搜索**，在搜索框输入关键词，录制文件命名规则如下：

单流录制 MP4 文件名规则：

`<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Index>.mp4`

合流录制 MP4 文件名规则： `<SdkAppId>_<RoomId>_<Index>.mp4`

版权所有：腾讯云计算（北京）有限责任公司

更多特性（TUICallKit）
设置昵称、头像（全平台）

最近更新时间：2024-04-03 17:23:11

本文介绍如何设置用户的头像和昵称。

设置头像、昵称

如果您需要自定义昵称或头像，可以使用如下接口进行更新：

Android（Kotlin）

Android（Java）

iOS（Objective-C）

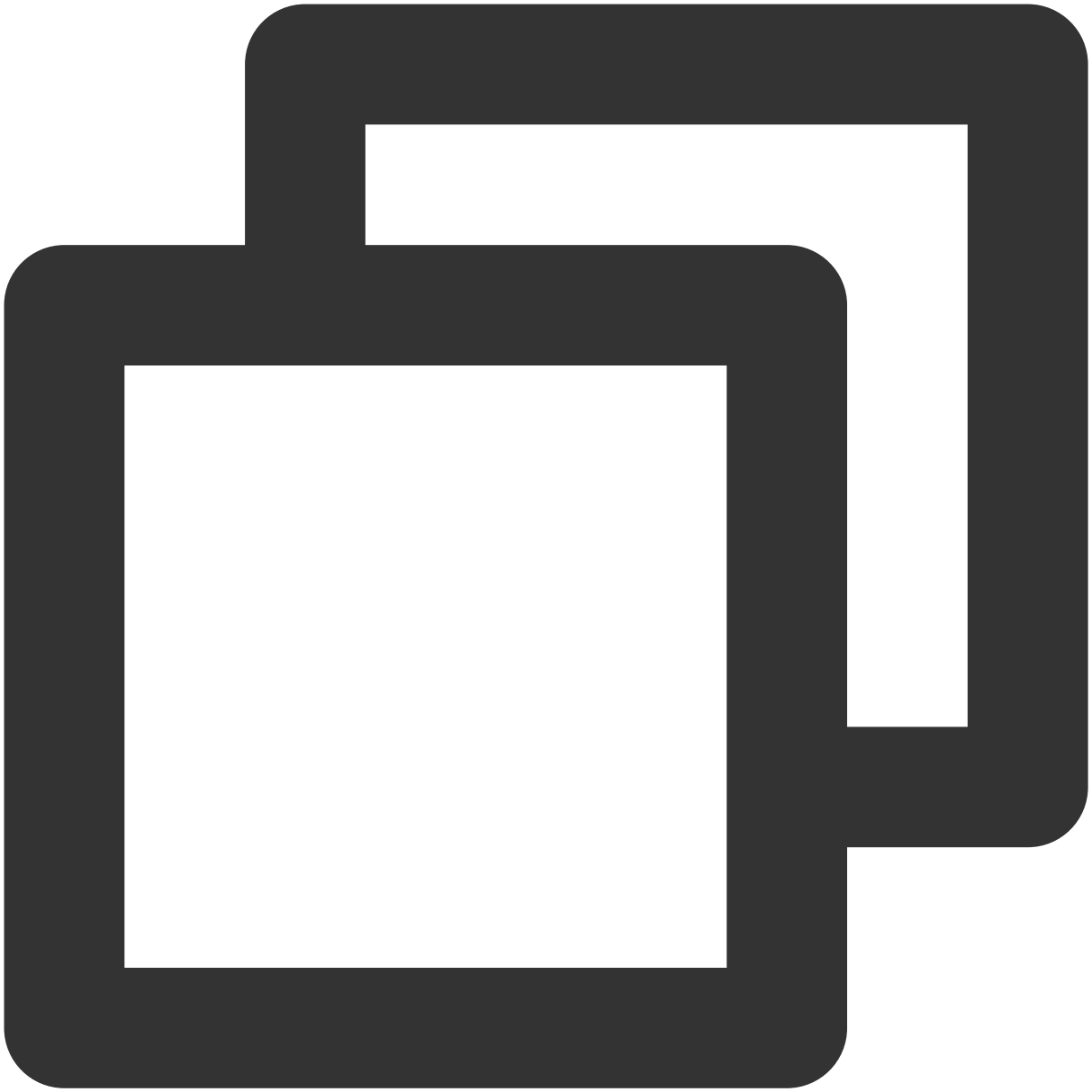
iOS（Swift）

Flutter（Dart）

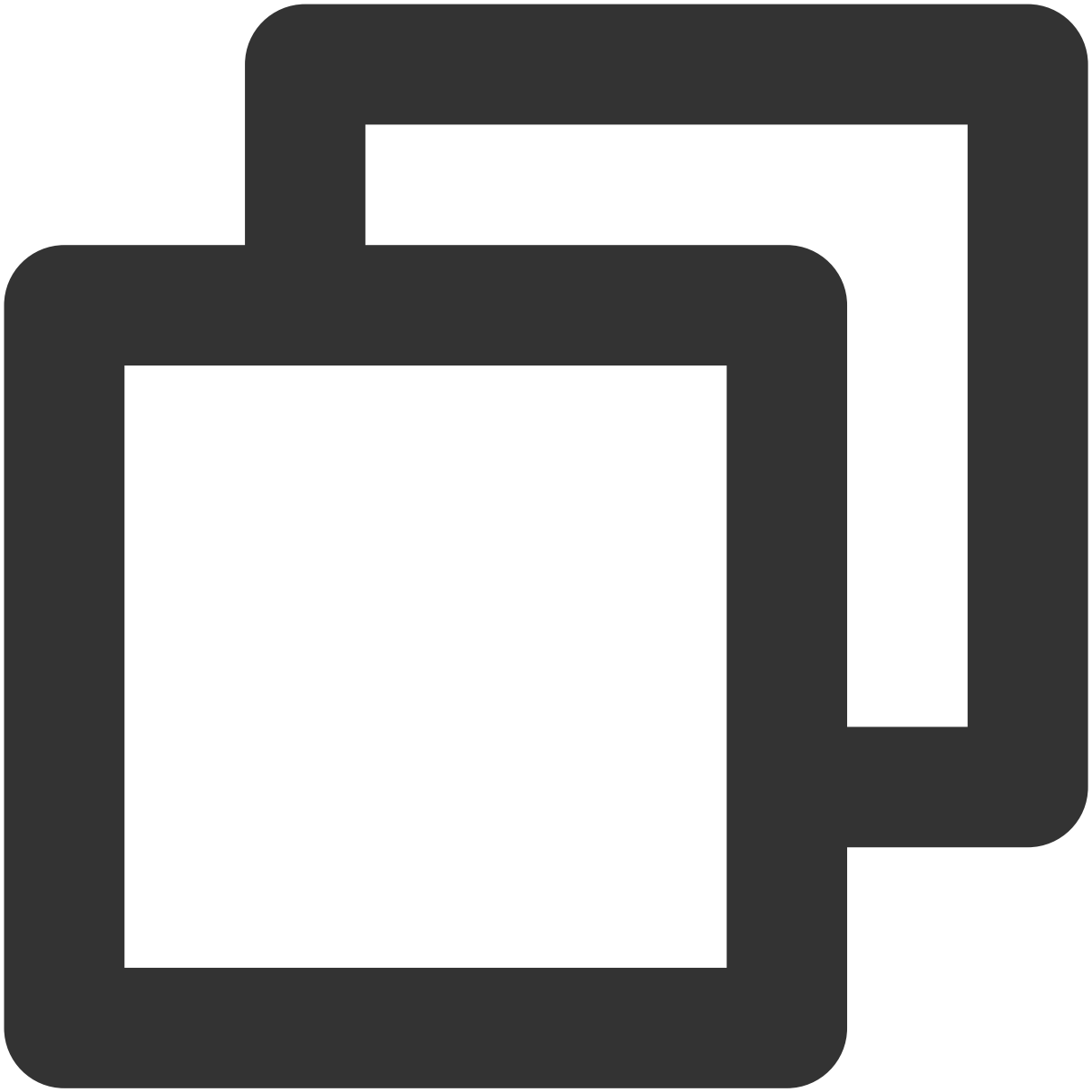
uni-app(Andorid&iOS)

Web&H5

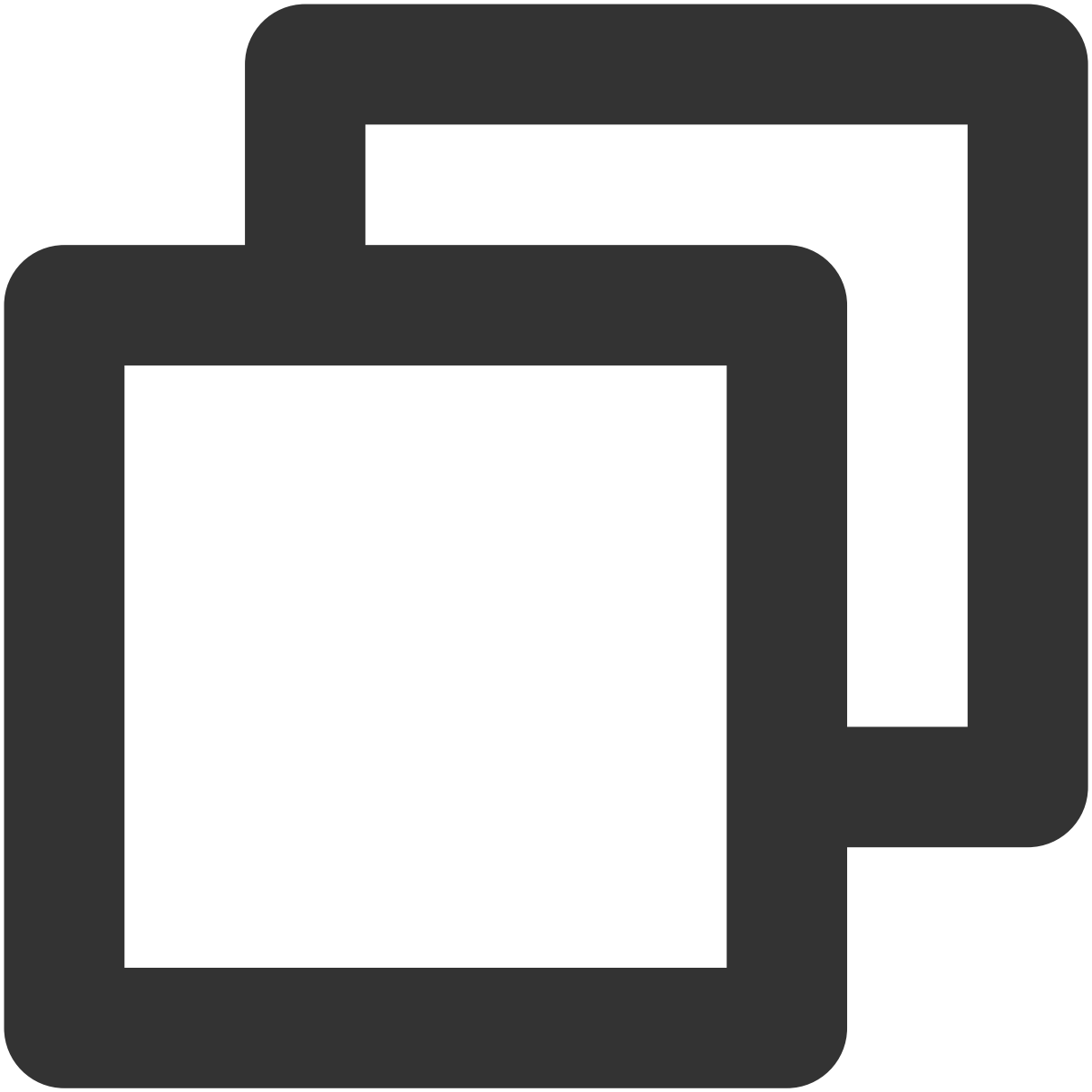
版权所有：腾讯云计算（北京）有限责任公司



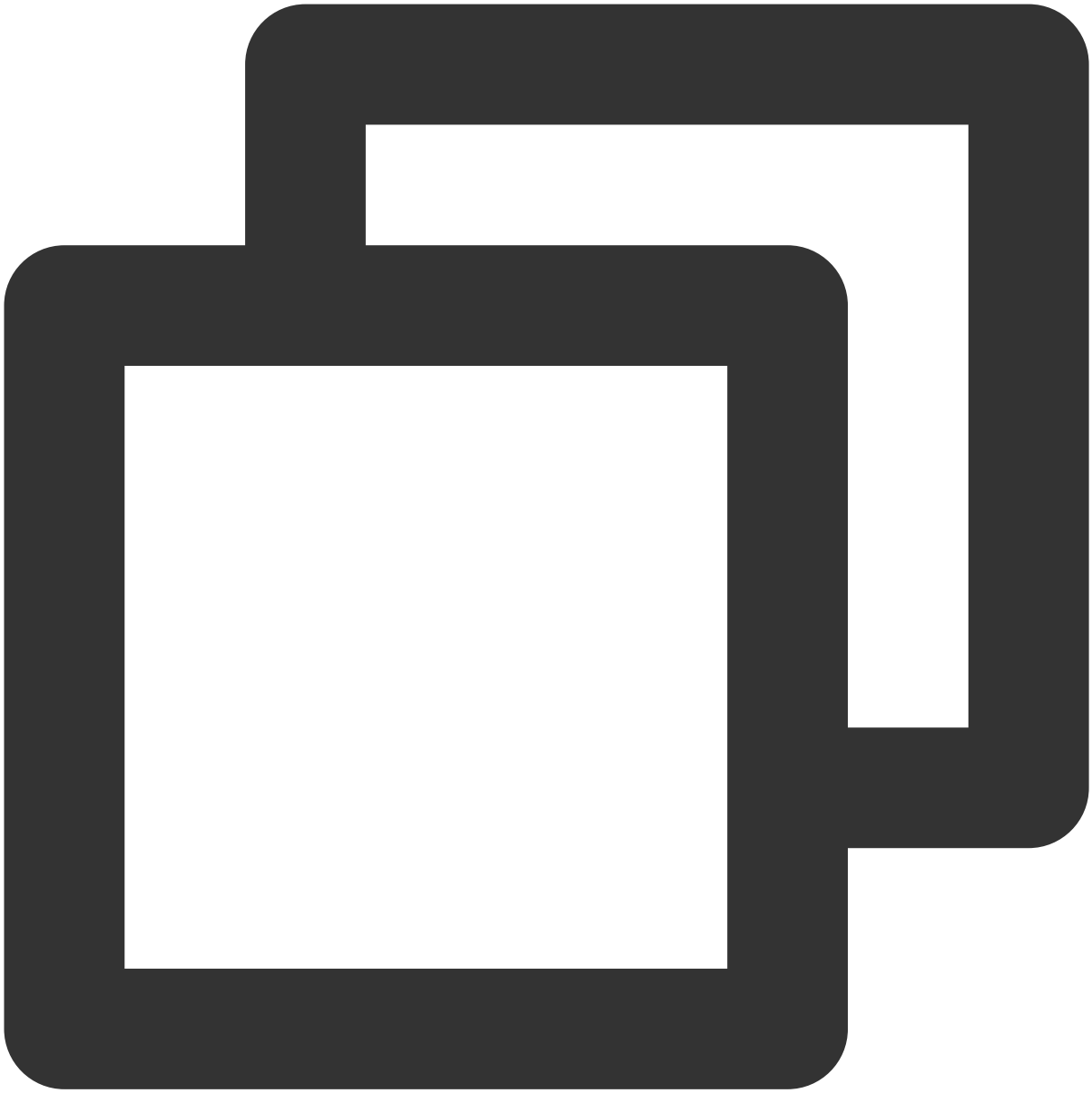
```
TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png
```



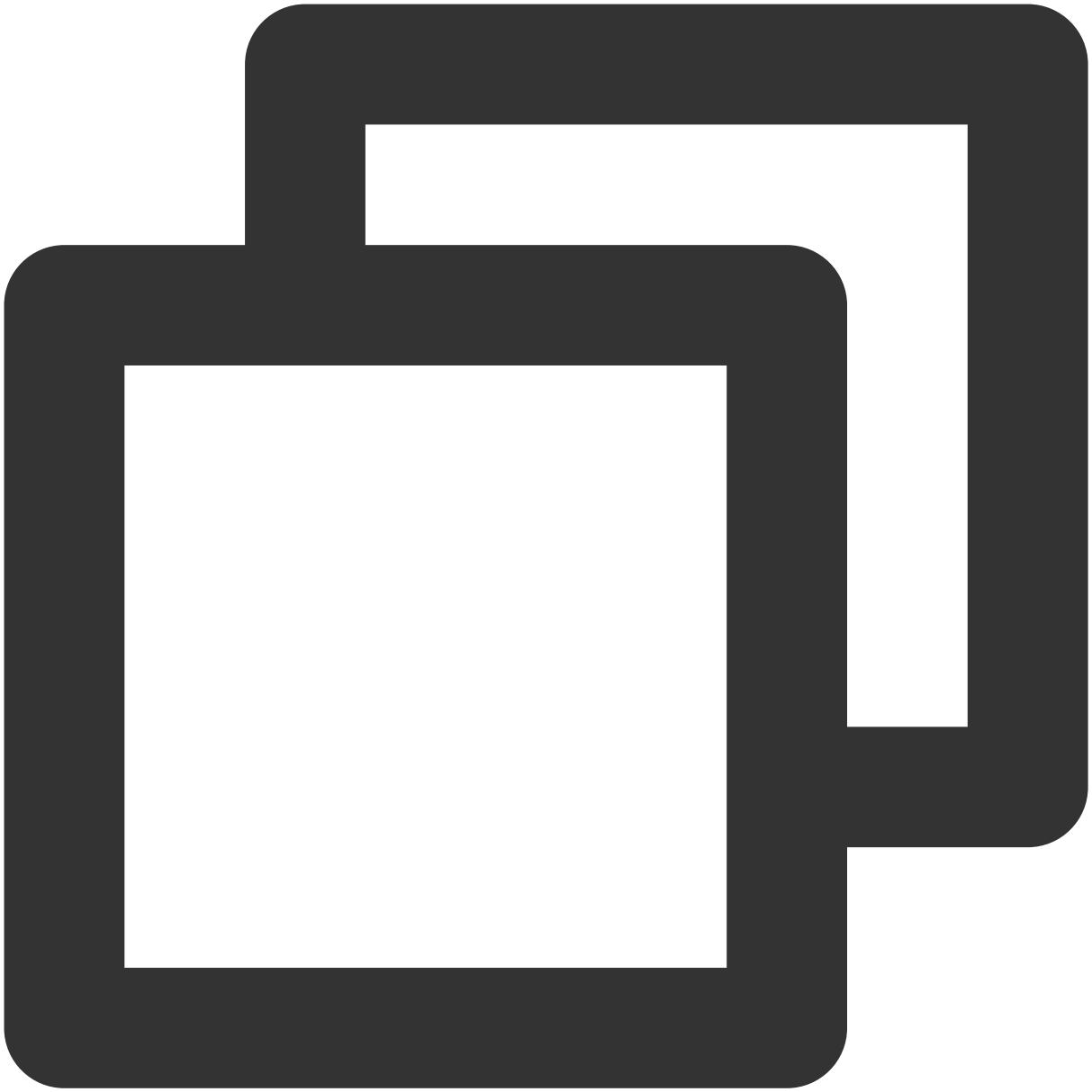
```
iTUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.pn
```



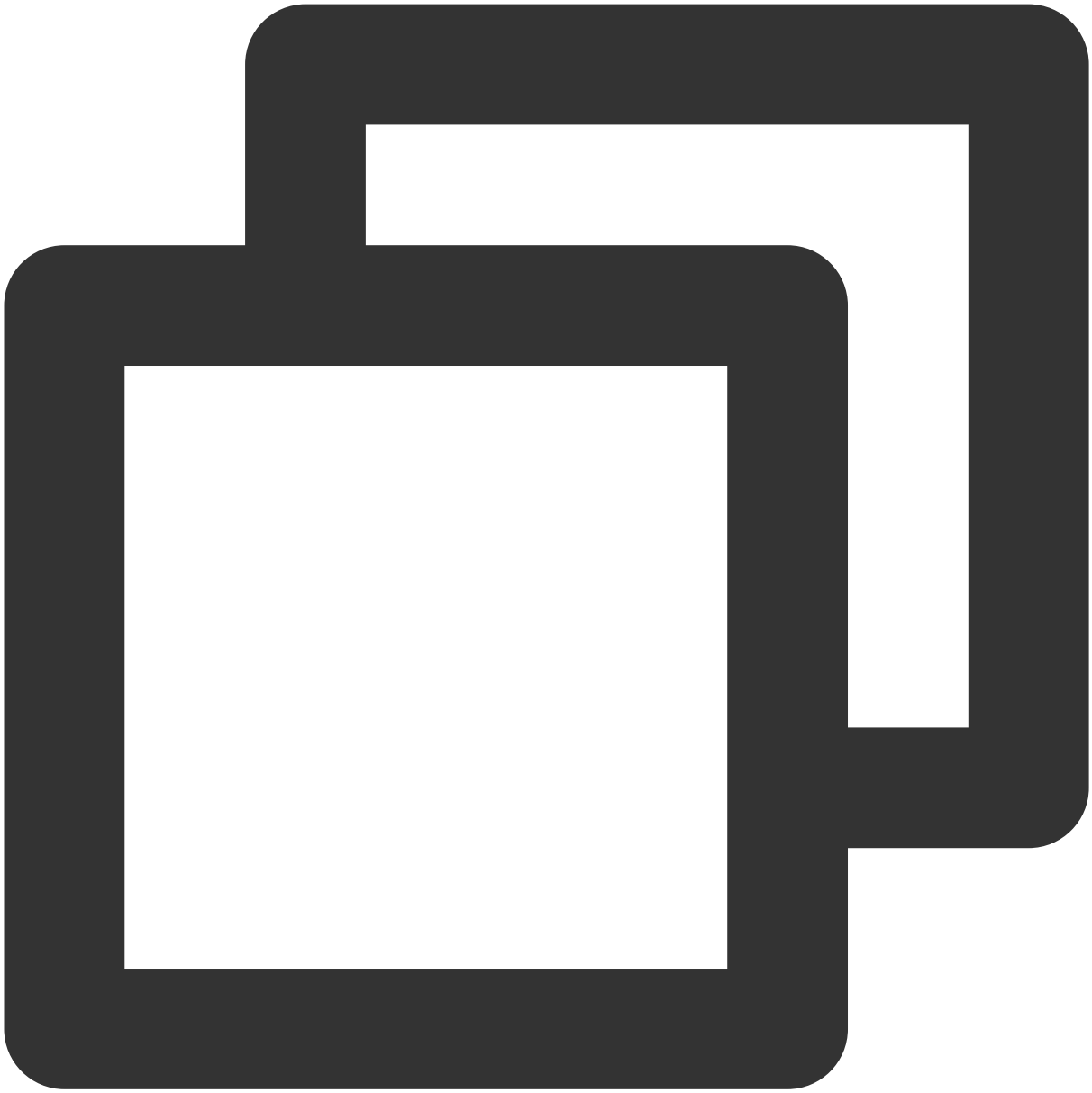
```
[[TUICallKit createInstance] setSelfInfo:@"" avatar:@"" succ:^{

} fail:^(int code, NSString *errMsg) {

}];
```



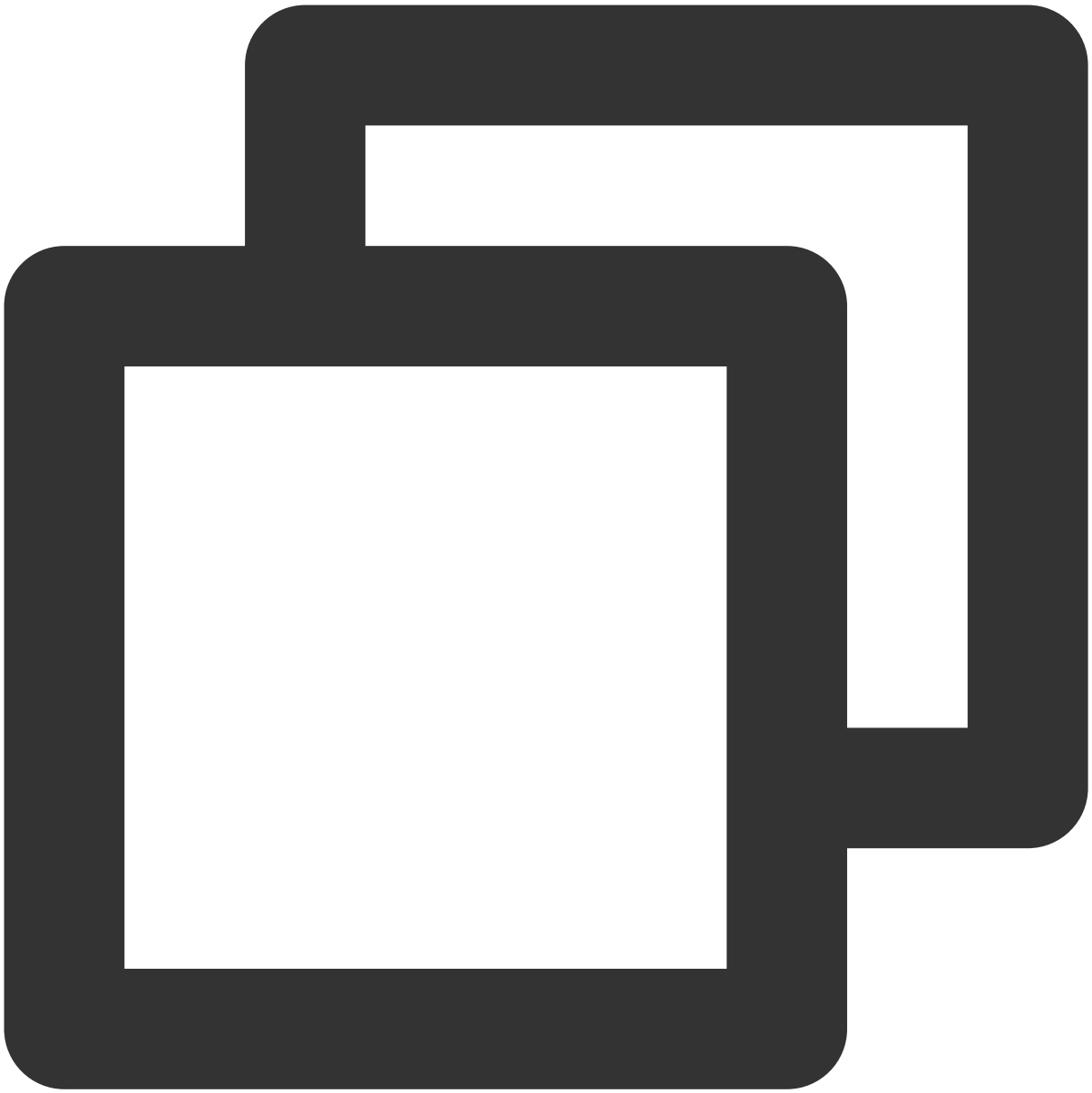
```
TUICallKit.createInstance().setSelfInfo(nickname: "", avatar: "") {

} fail: { code, message in

}
```



```
TUIResult result = TUICallKit.instance.setSelfInfo('userName', 'url:********');
```



```
const options = {
    nickName: 'jack',
    avatar: 'https:/****/user_avatar.png'
}
TUICallKit.setSelfInfo(options, (res) => {
    if (res.code === 0) {
        console.log('setSelfInfo success');
    } else {
        console.log(`setSelfInfo failed, error message = ${res.msg}`);
    }
});
```



```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "jack", avatar: "http://xxx" });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`);
}
```

注意

因为用户隐私限制，非好友之间的通话，被叫的昵称和头像更新可能会有延迟，一次通话成功后就会顺利更新。

设置分辨率、填充模式(Web)

最近更新时间：2024-04-03 17:23:11

本位介绍如何设置分辨率、填充模式。

设置分辨率、填充模式

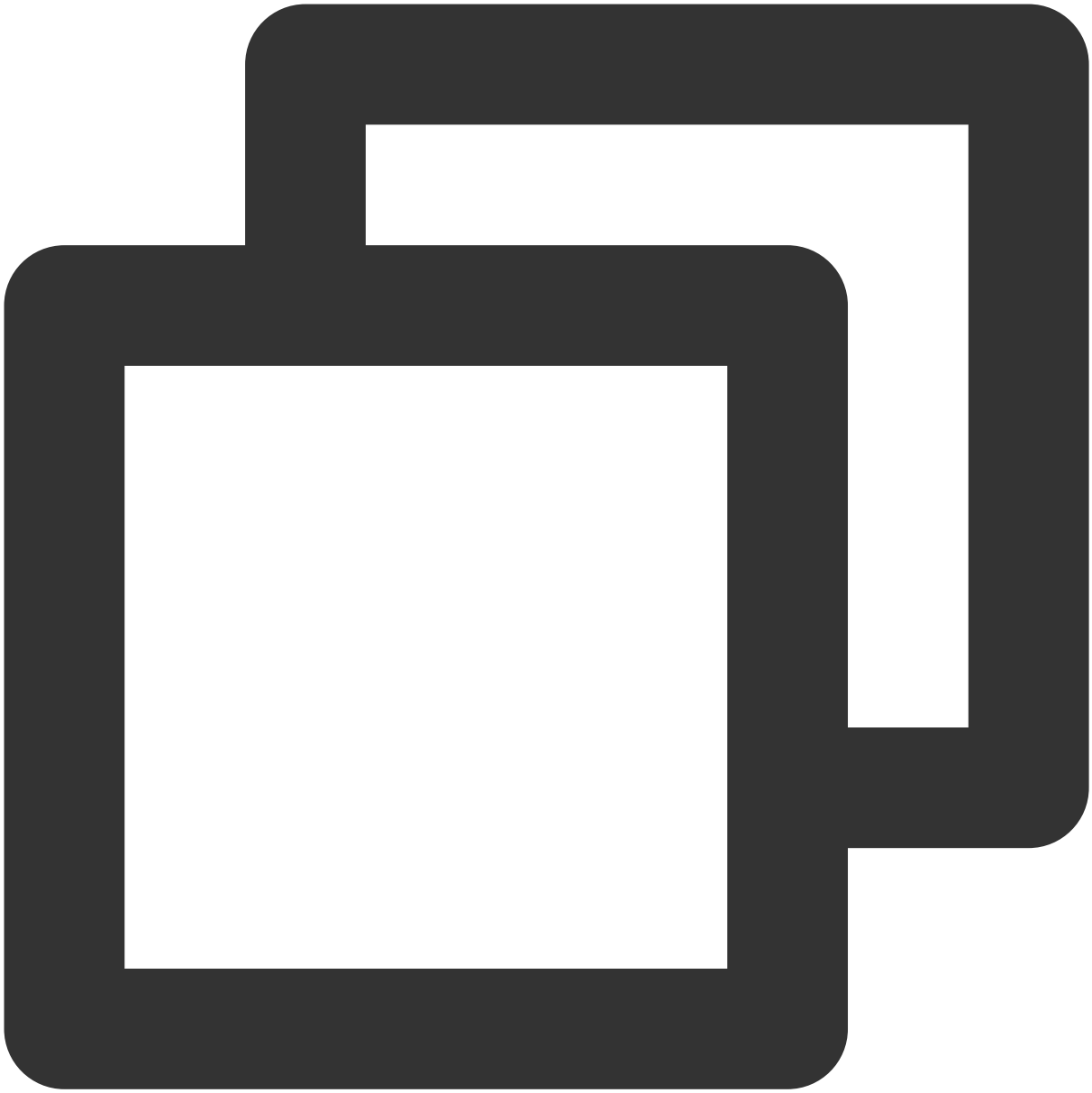
TUICallKit 组件提供了若干个功能开关，可以根据需要选择开启或关闭，具体参见 TUICallKit 属性介绍。

`videoDisplayMode` ：画面显示模式。

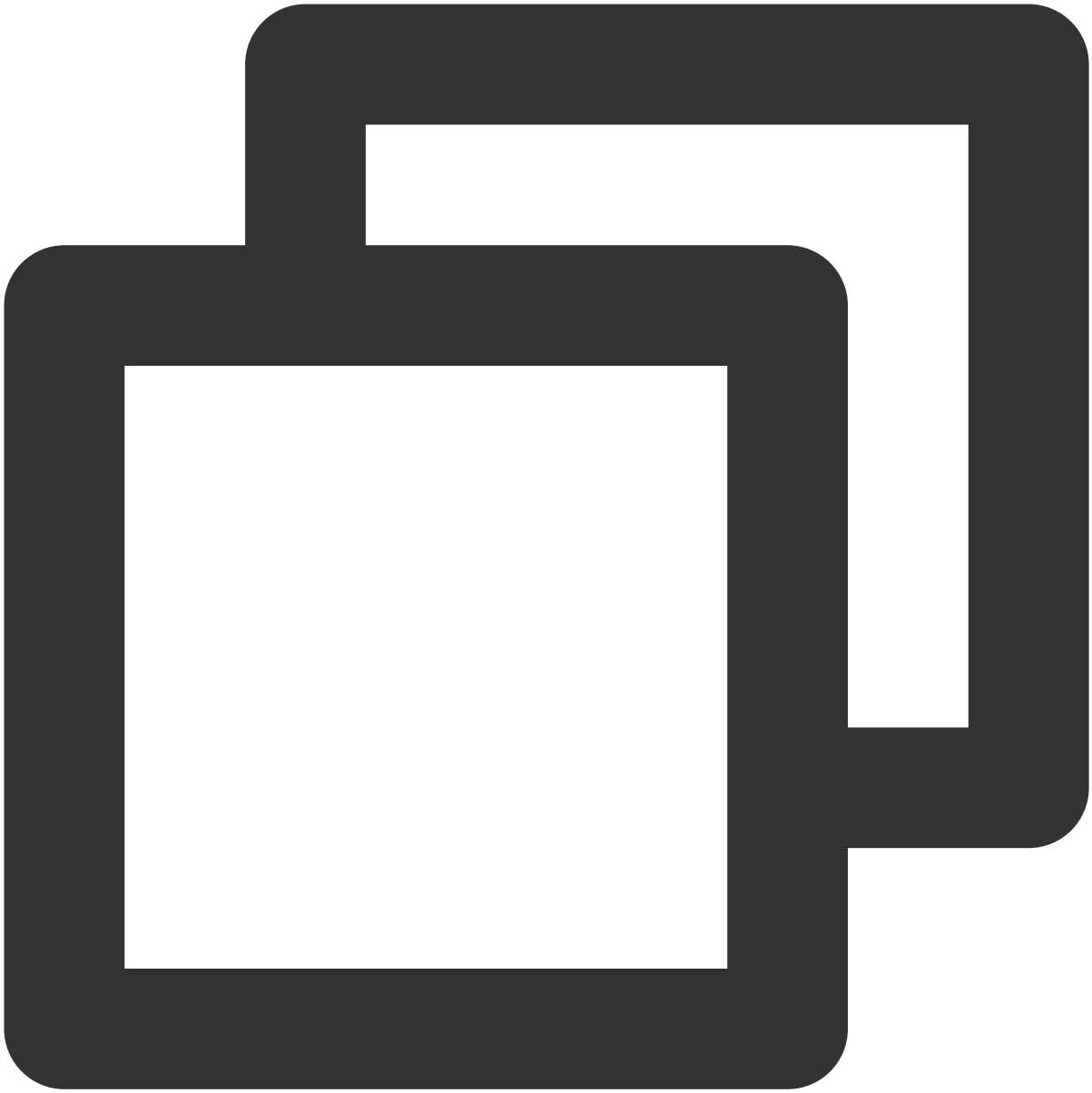
`videoResolution` ：通话分辨率。

React

Vue



```
import { VideoDisplayMode, VideoResolution } from "@tencentcloud/call-uikit-react";

<TUICallKit
    videoDisplayMode={VideoDisplayMode.CONTAIN}
    videoResolution={VideoResolution.RESOLUTION_1080P} />
```



```
import { VideoDisplayMode, VideoResolution } from "@tencentcloud/call-uikit-vue";

<TUICallKit
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P" />
```

videoDisplayMode

画面显示模式 `videoDisplayMode` 属性有三个值：

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER`

`VideoDisplayMode.FILL`，默认值是 `VideoDisplayMode.COVER`。

属性	值	描述
videoDisplayMode	VideoDisplayMode.CONTAIN	优先保证视频内容全部显示。 视频尺寸等比缩放，直至视频窗口的一边与视窗边框对齐。 如果视频尺寸与显示视窗尺寸不一致，在保持长宽比的前提下，将视频进行缩放后填满视窗，缩放后的视频四周会有一圈黑边。
	VideoDisplayMode.COVER	优先保证视窗被填满。 视频尺寸等比缩放，直至整个视窗被视频填满。 如果视频长宽与显示窗口不同，则视频流会按照显示视窗的比例进行周边裁剪或图像拉伸后填满视窗。
	VideoDisplayMode.FILL	保证视窗被填满的同时保证视频内容全部显示，但是不保证视频尺寸比例不变。 视频的宽高会被拉伸至和视窗尺寸一致。

videoResolution

分辨率 `videoResolution` 有三个值：

`VideoResolution.RESOLUTION_480P`

`VideoResolution.RESOLUTION_720P`

`VideoResolution.RESOLUTION_1080P`，默认值是 `VideoResolution.RESOLUTION_480P`。

分辨率说明：

视频 Profile	分辨率（宽 × 高）	帧率（fps）	码率（kbps）
480p	640 × 480	15	900
720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

常见问题：

iOS 13&14不支持编码高于 720P 的视频，建议在这两个系统版本限制最高采集 720P。参见 iOS Safari 已知问题 case 12。

Firefox 不支持自定义视频帧率（默认为 30fps）。

受系统性能占用，摄像头采集能力和浏览器限制等因素的影响，视频分辨率、帧率、码率的实际值不一定能够完全匹配设定值，在这种情况下，浏览器会自动调整 Profile 尽可能匹配设定值。

版权所有：腾讯云计算（北京）有限责任公司

群组通话

Android&iOS&Flutter

最近更新时间：2024-04-03 17:23:11

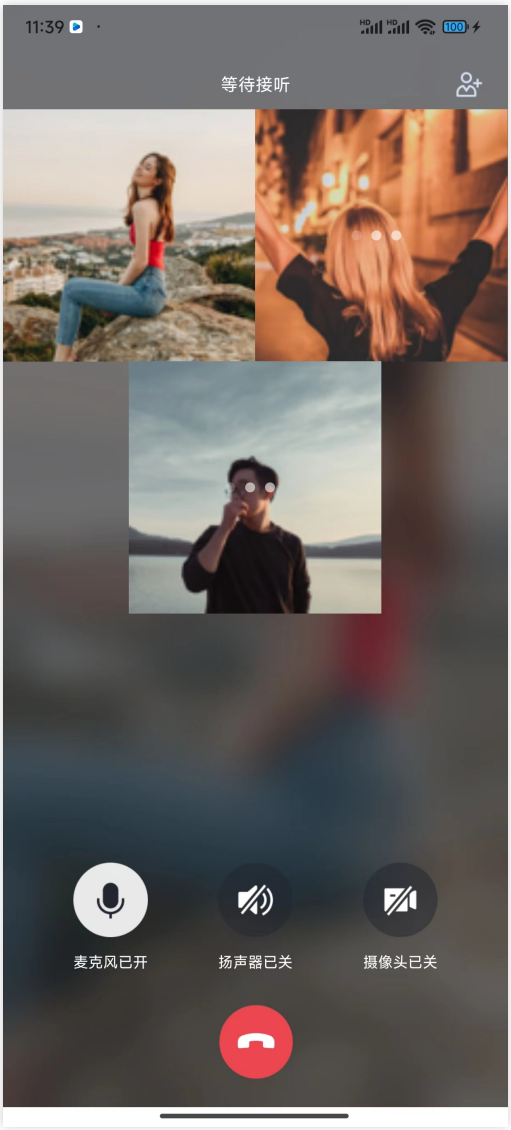
本文介绍群组通话功能的使用，如发起群组通话、加入群组通话。

预期效果

TUICallKit 支持群组通话，预期效果见下图。

发起群组通话	收到群组通话请求

版权所有：腾讯云计算（北京）有限责任公司





创建 groupID

使用群组通话功能前，需要先创建群组，在已存在的群组中发起群组通话。

方式一：通过调用 IM API 创建群，详见 IM 群组管理。

方式二：通过控制台手动创建群组，详见 控制台群组管理。

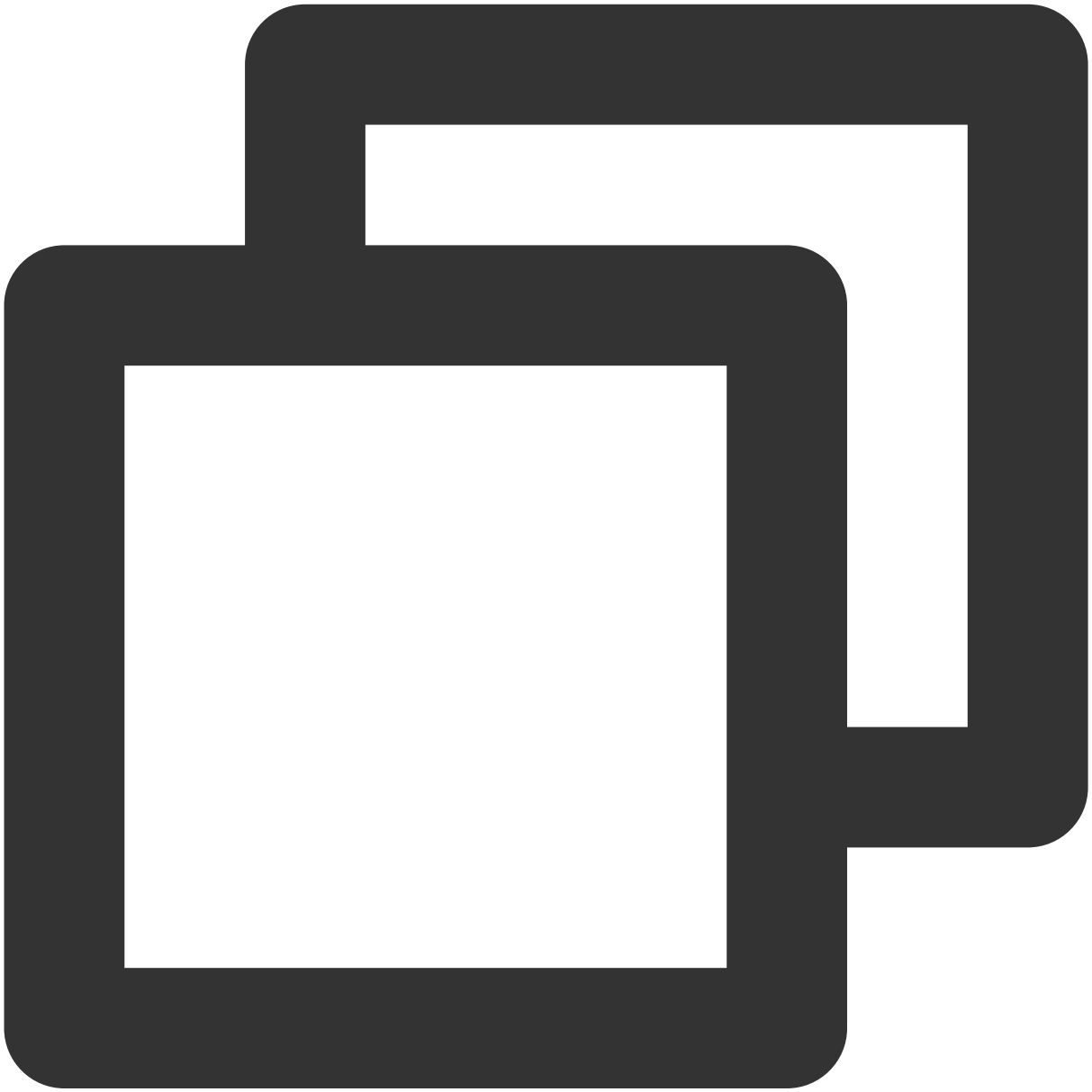
群组通话

发起群组通话

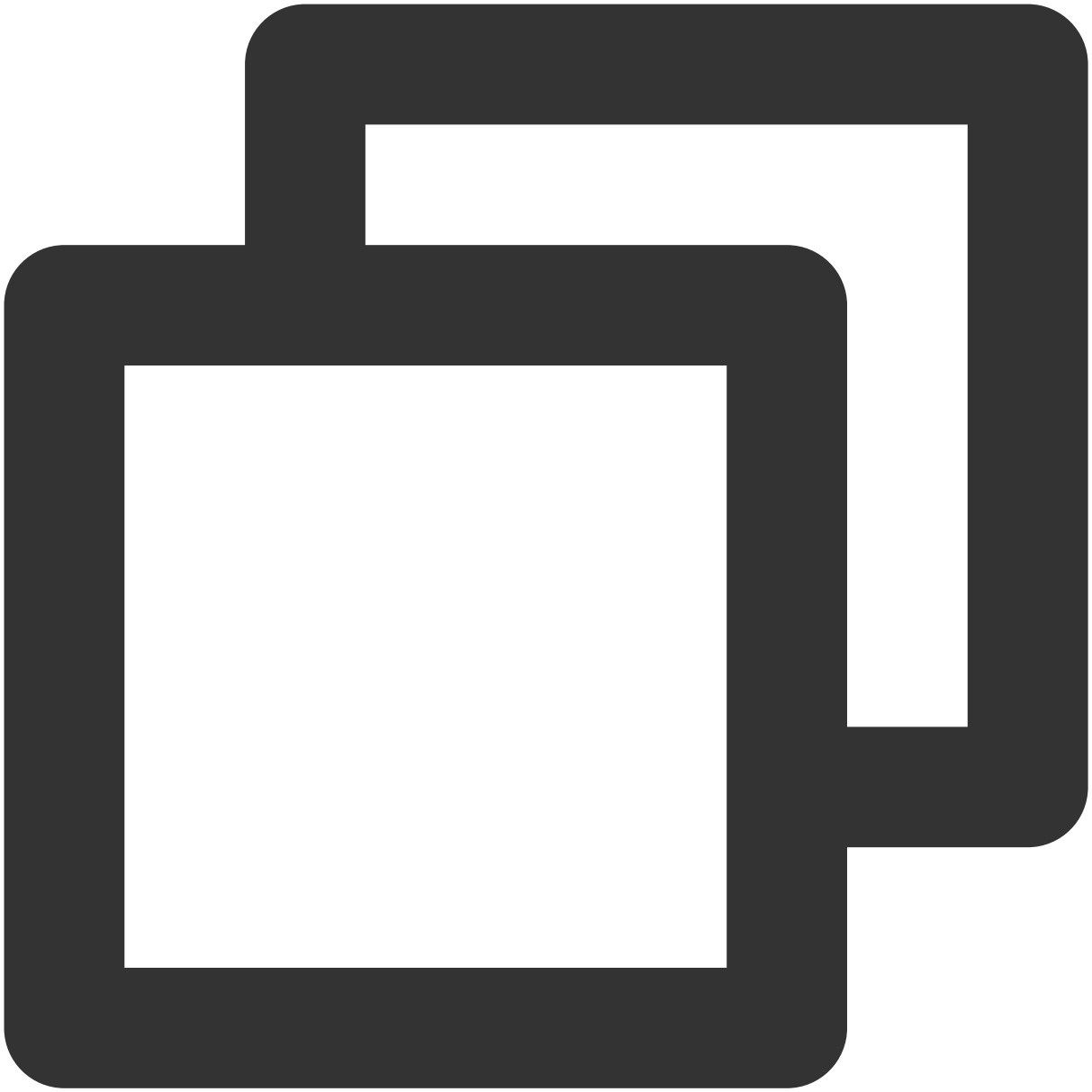
调用 groupCall API 发起群通话。

Android（Kotlin）

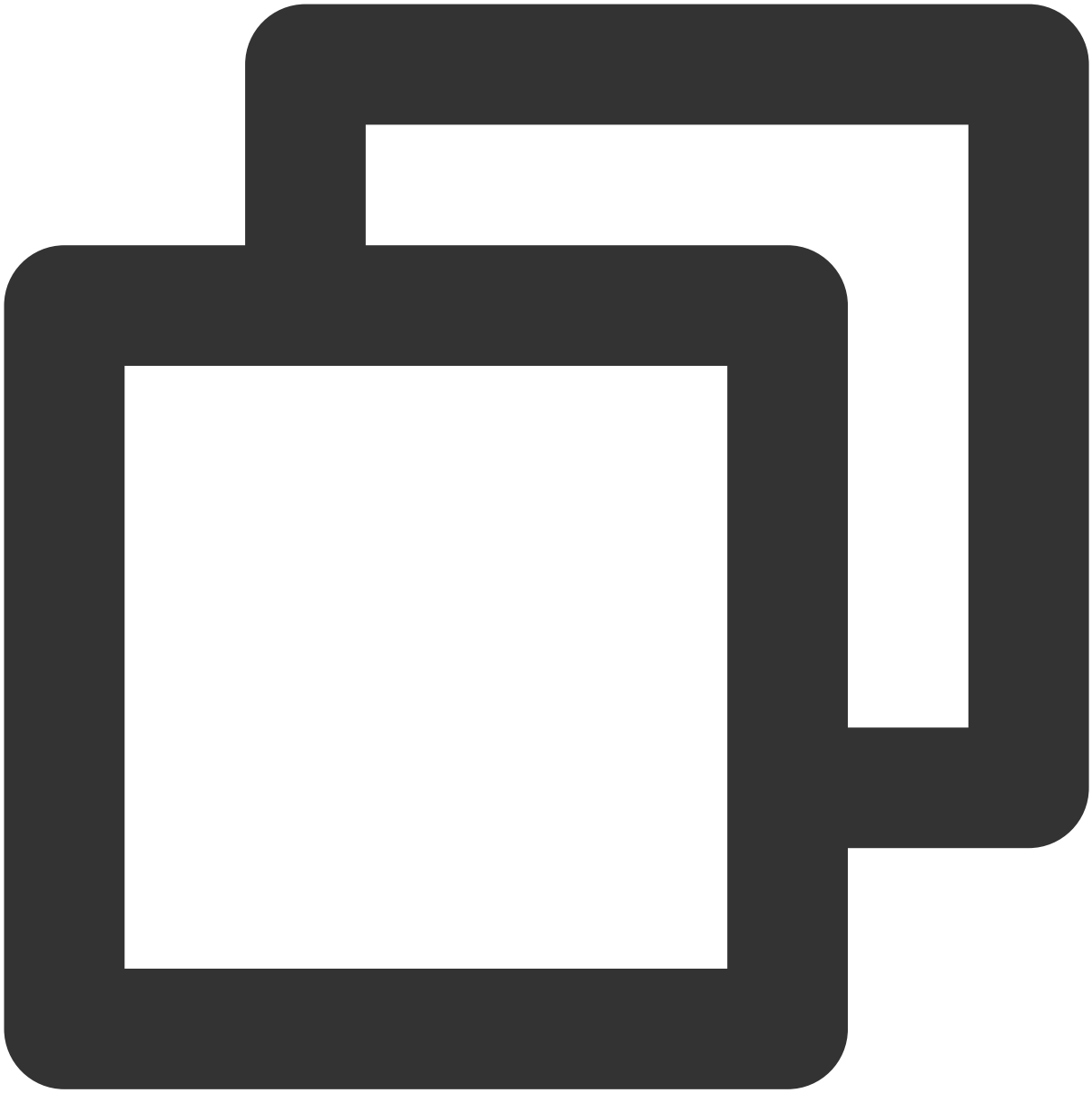
Android（Java）
iOS（Swift）
iOS（Objective-C）
Flutter（Dart）



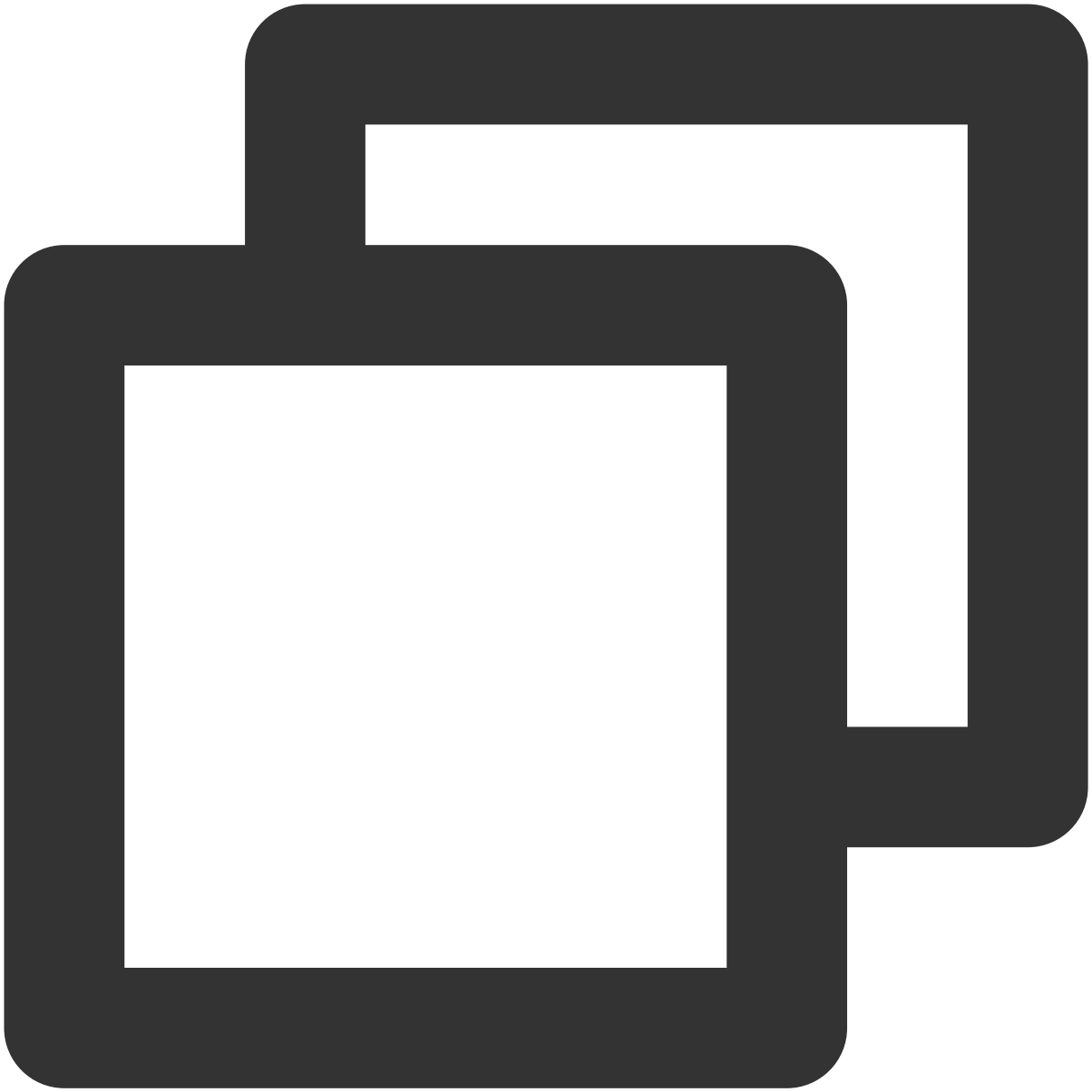
```
TUICallKit.createInstance(context).groupCall("12345678", Arrays.asList("jane", "mik
```



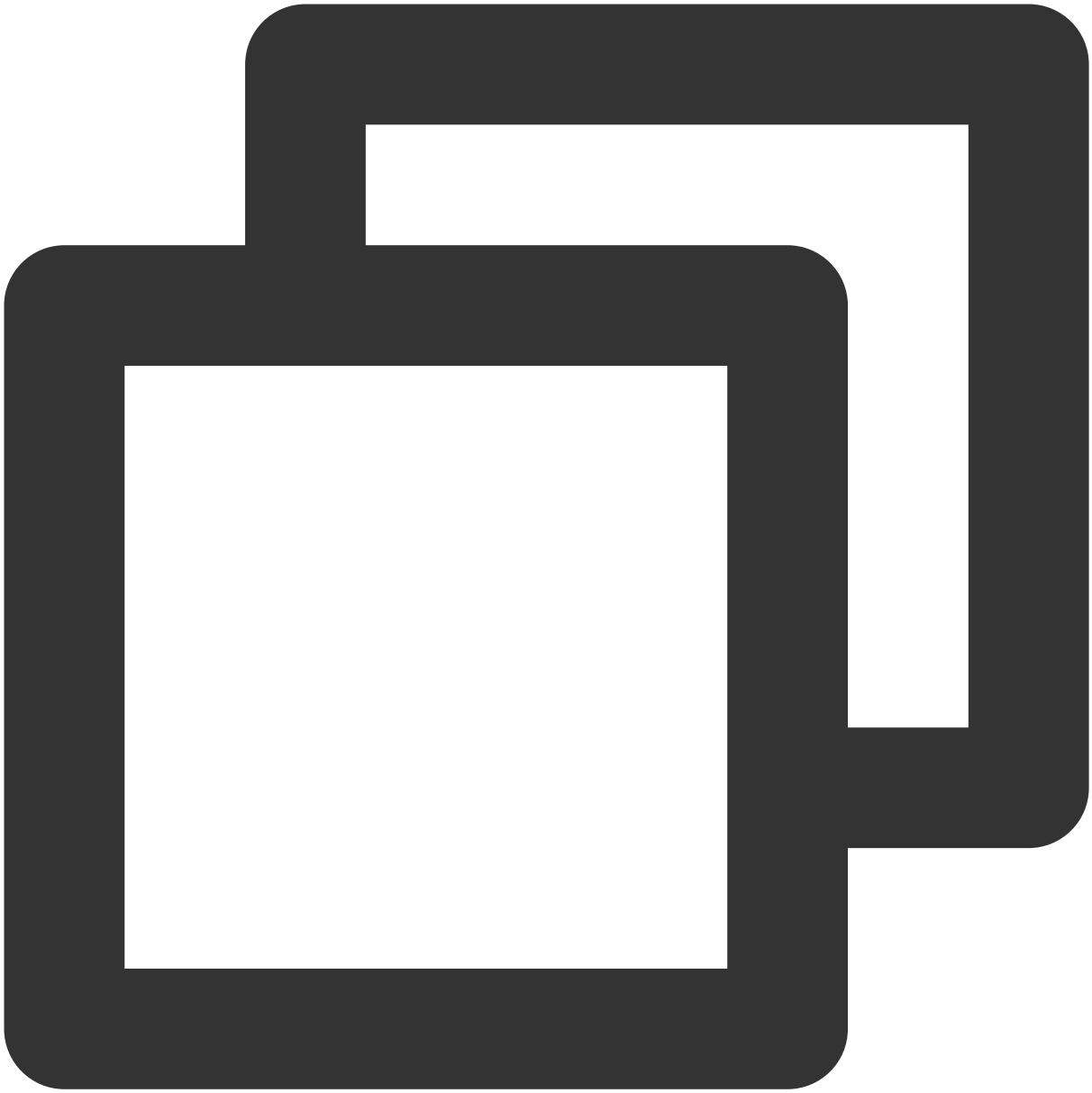
```
TUICallKit.createInstance(context).groupCall("12345678", Arrays.asList("jane", "mik
```



```
import TUICallKit

TUICallKit.createInstance().groupCall(groupId: "12345678",
                                       userIdList: ["denny", "mike", "tommy"],
                                       callMediaType: .video)
```



```
#import <TUICallKit/TUICallKit.h>

[[TUICallKit createInstance] groupCall:@"12345678"
                            userIdList:@[@"denny", @"mike", @"tommy"]
                         callMediaType:TUICallMediaTypeVideo];
```



```
TUICallKit.instance.groupCall('0001', ['denny', 'mike', 'tommy'], TUICallMediaType.
```

加入群组通话

调用 joinInGroupCall API 主动加入群组中已有的音视频通话。

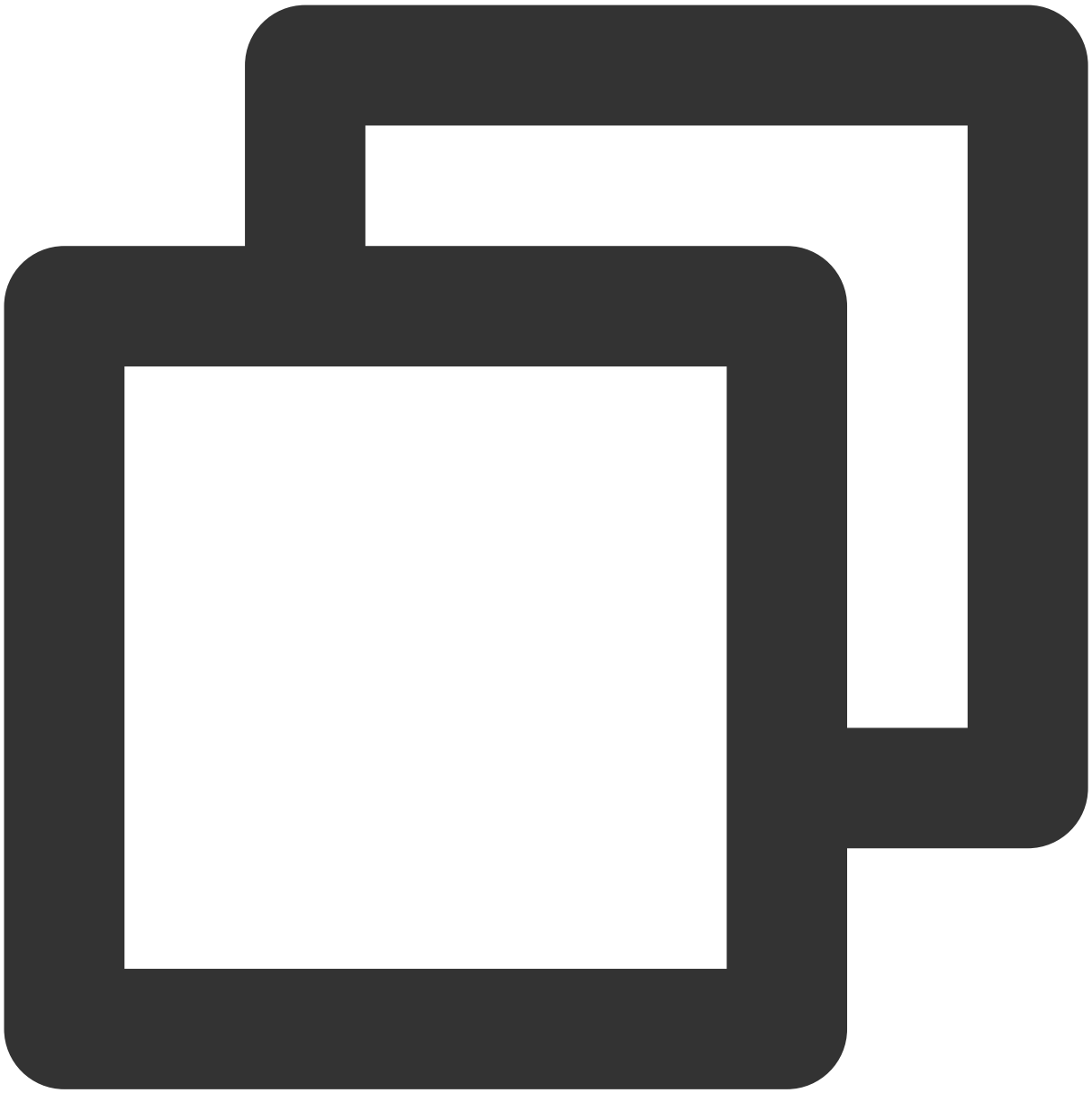
Android（Kotlin）

Android（Java）

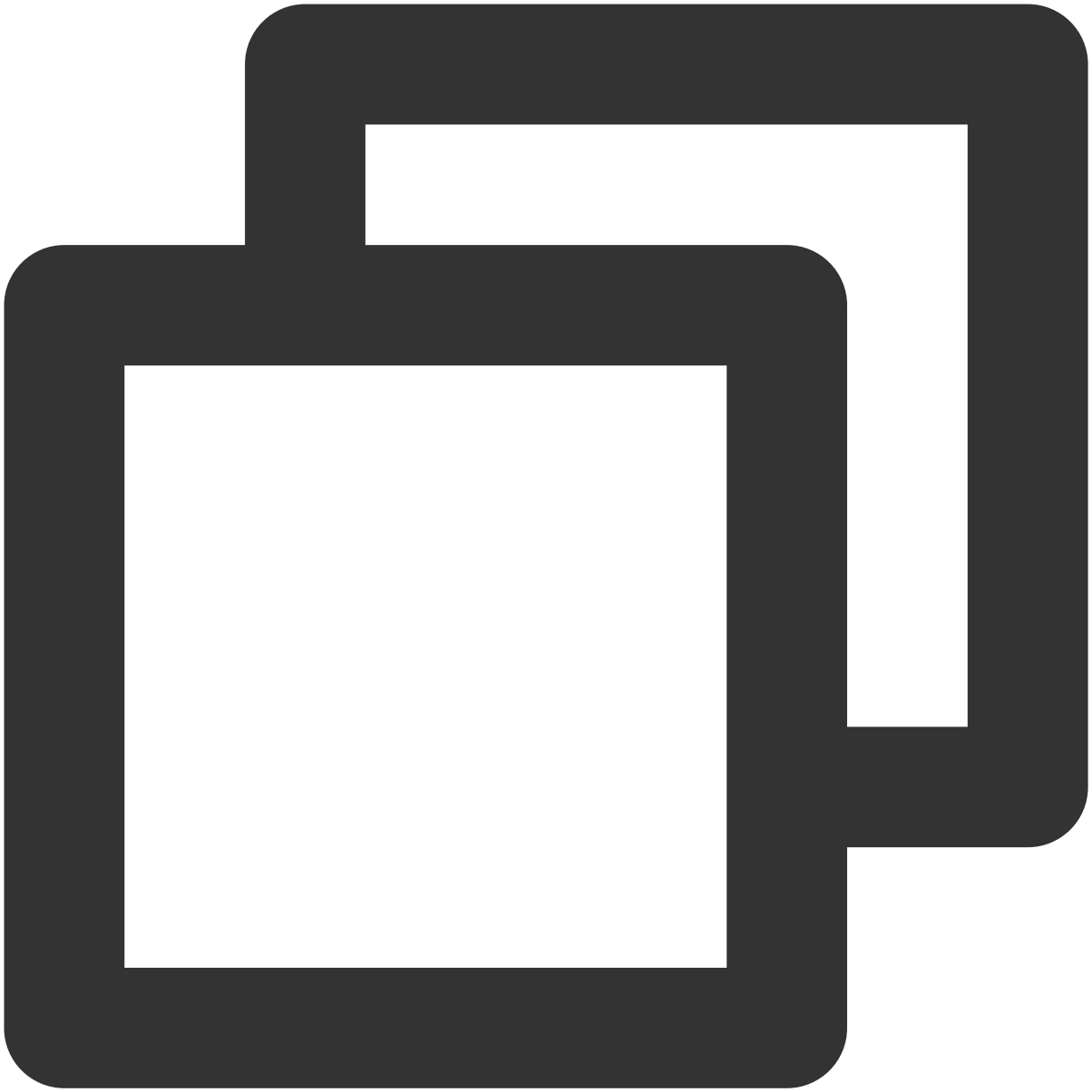
iOS（Swift）

iOS（Objective-C）

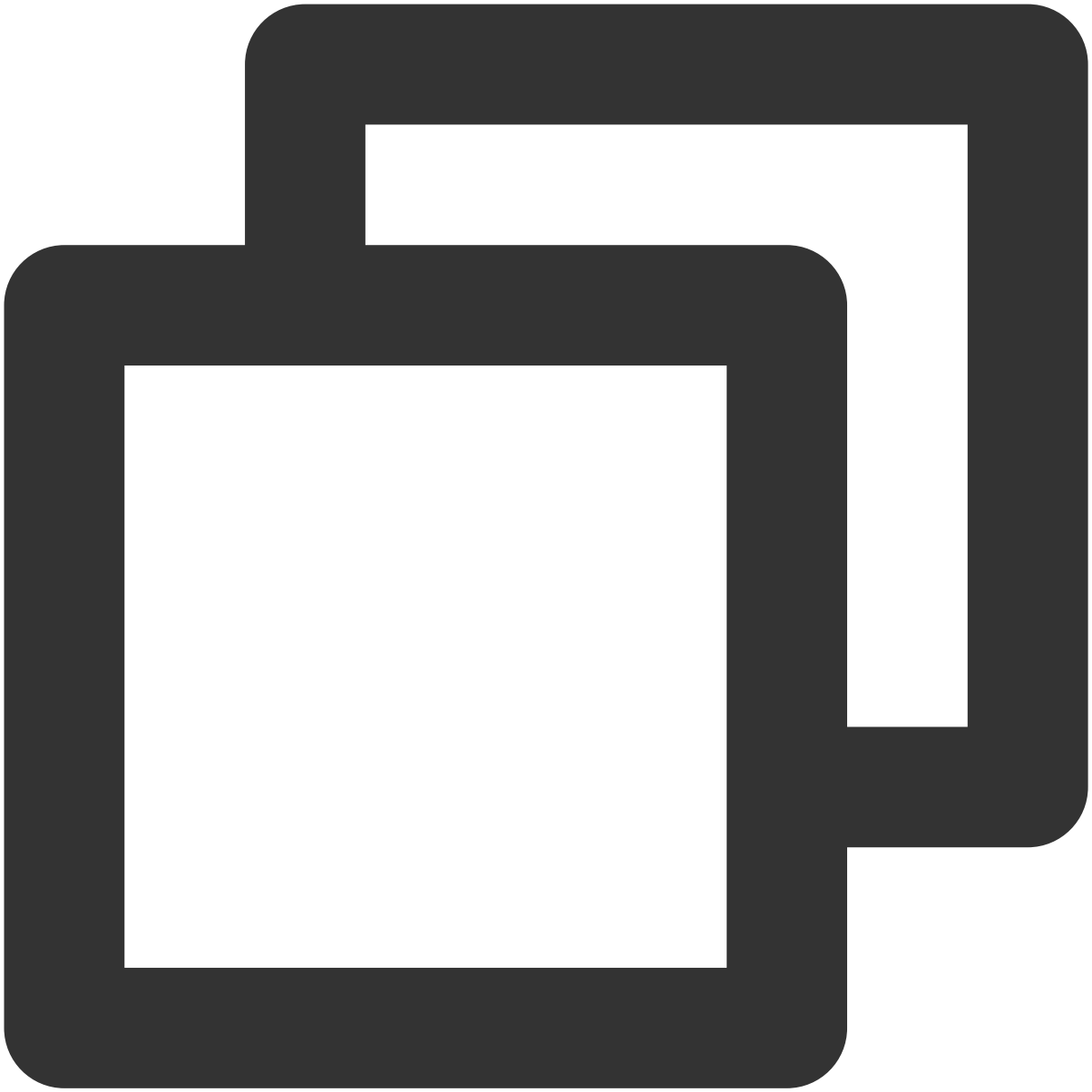
Flutter（Dart）



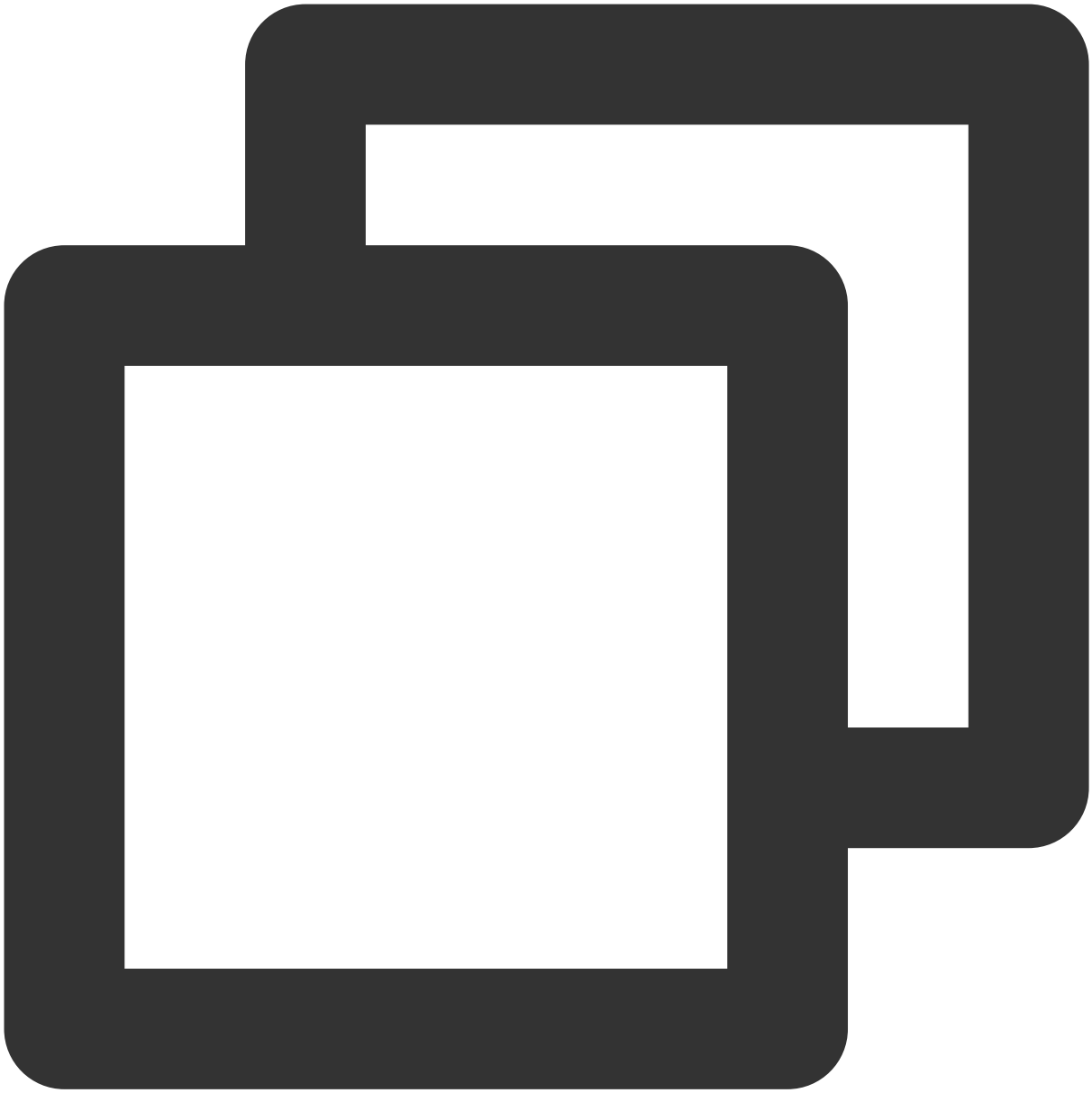
```
RoomId roomId = RoomId();
roomId.intRoomId = 123321;
TUICallKit.createInstance(context).joinInGroupCall(roomId, "12345678", TUICallDefin
```



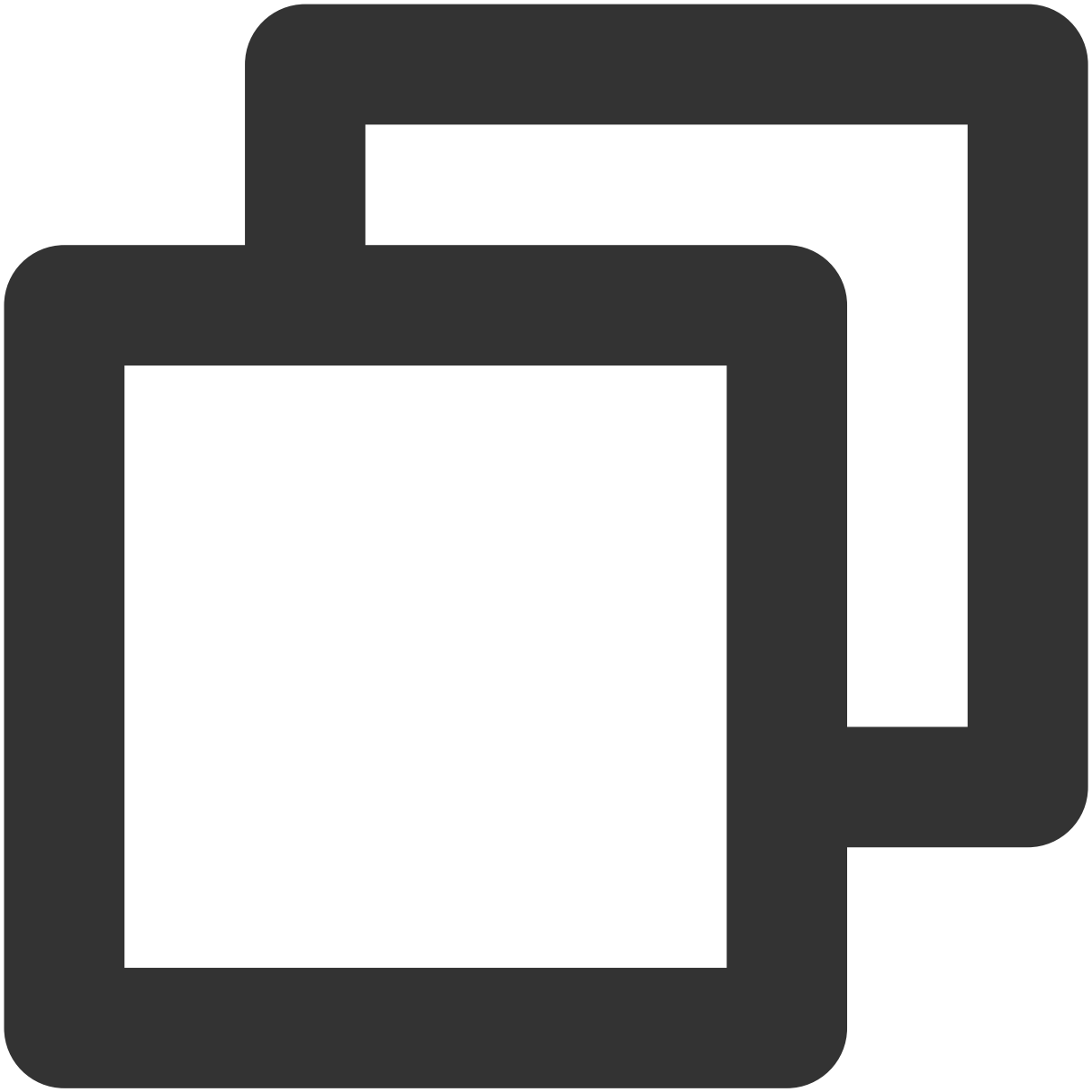
```
RoomId roomId = new RoomId();
roomId.intRoomId = 123321;
TUICallKit.createInstance(context).joinInGroupCall(roomId , "12345678", TUICallDefi
```



```
import TUICallKit
let roomId = TUIRoomId()
roomId.intRoomId = 123321
TUICallKit.createInstance().joinInGroupCall(roomId: roomId,
                                            groupId: "1234567",
                                            callMediaType: .video)
```



```
#import <TUICallKit/TUICallKit.h>

TUIRoomId *roomId = [[TUIRoomId alloc] init];
roomId.intRoomId = 123321;

[[TUICallKit createInstance] roomId: roomId
                            groupId:@"223344"
                      callMediaType:TUICallMediaTypeVideo];
```



```
TUIRoomId roomId = TUIRoomId()
roomId.intRoomId = 123321;
TUICallKit.instance.joinInGroupCall(roomId, '1234567', TUICallMediaType.video);
```

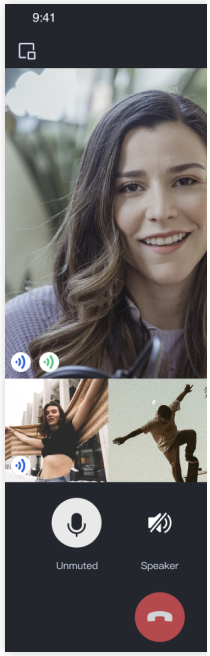
Web&H5

最近更新时间：2024-04-03 17:23:11

本文介绍群组通话功能的使用，如发起群组通话、加入群组通话。

预期效果

TUICallKit 支持群组通话，预期效果见下图。

Web 端	移动端
	

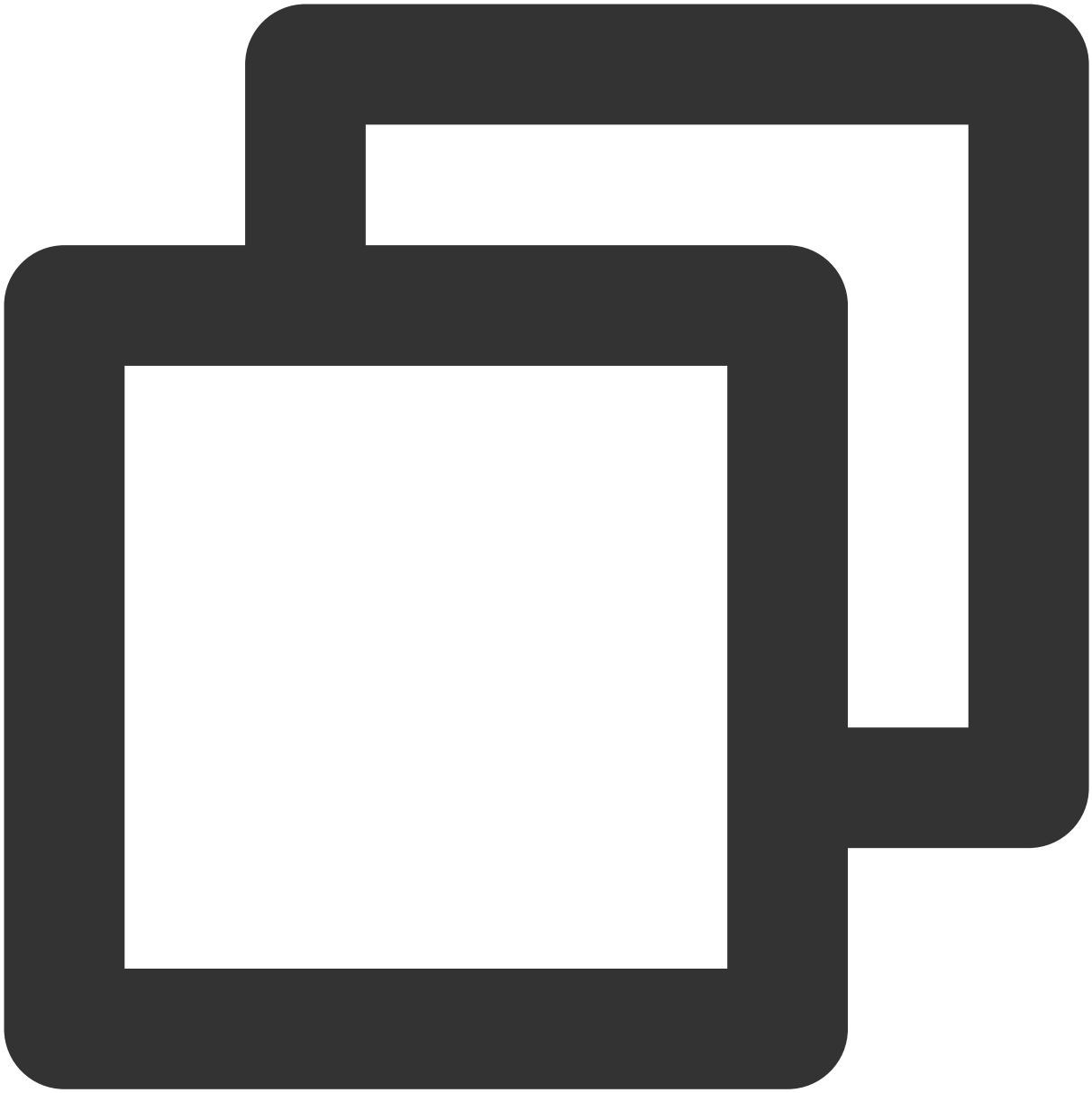
创建 groupID

使用群组通话功能前，需要先创建群组，在已存在的群组中发起群组通话。

方式一：通过调用 IM API 创建群，详见 IM 群组管理。

方式二：通过控制台手动创建群组，详见 控制台群组管理。

版权所有：腾讯云计算（北京）有限责任公司



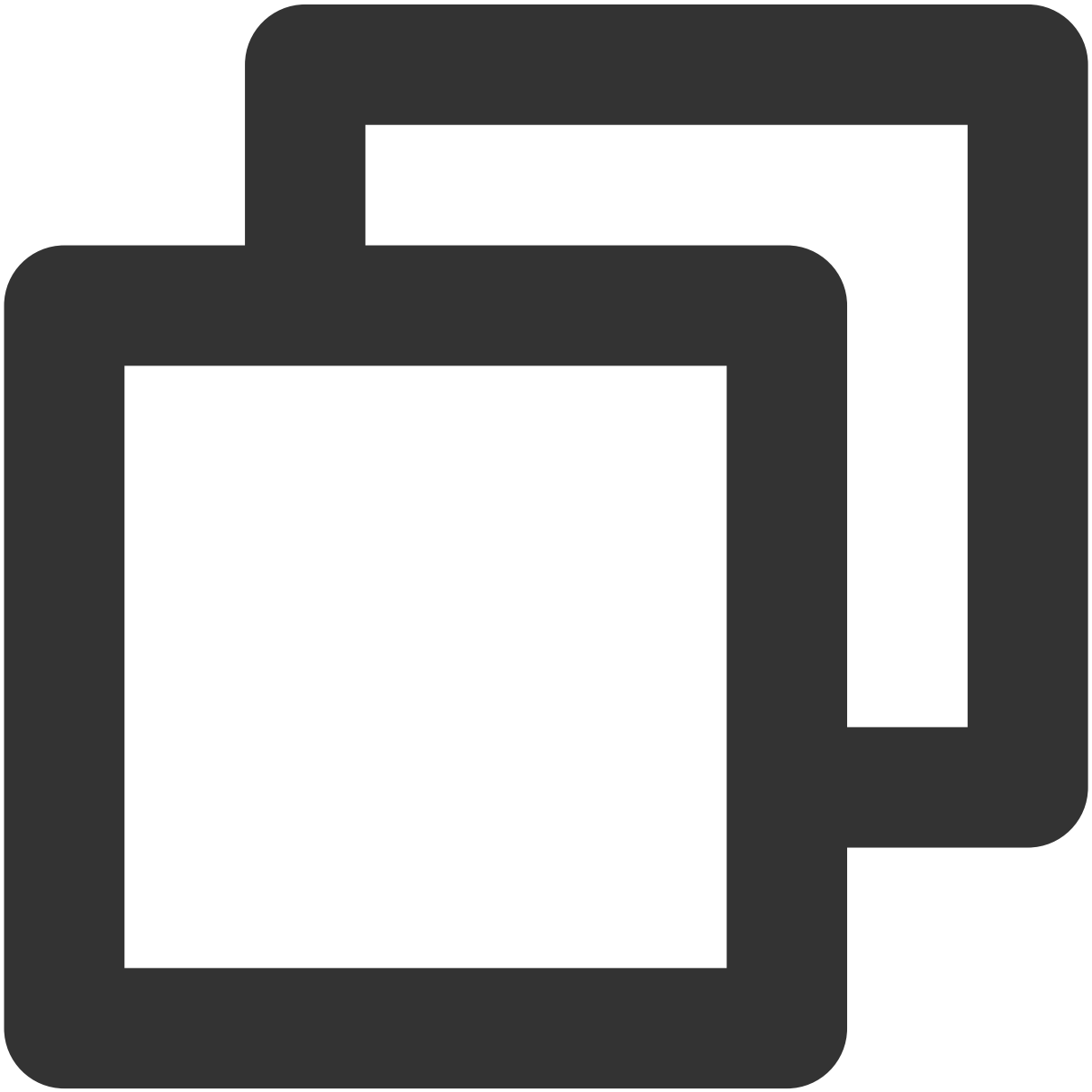
```
import TIM from "@tencentcloud/chat";   // npm i @tencentcloud/chat

const userIDList: string[] = ['user1', 'user2'];
async function createGroupID() {
  const tim = TIM.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userId => ({ userID: userId }));
  const res = await tim.createGroup({
    type: TIM.TYPES.GRP_PUBLIC, // Must be a public group
    name: 'WebSDK',
    memberList
  });
```

```
  return res.data.group.groupID;
}
```

群组通话

发起群组通话

调用 `groupCall` API 发起群通话。

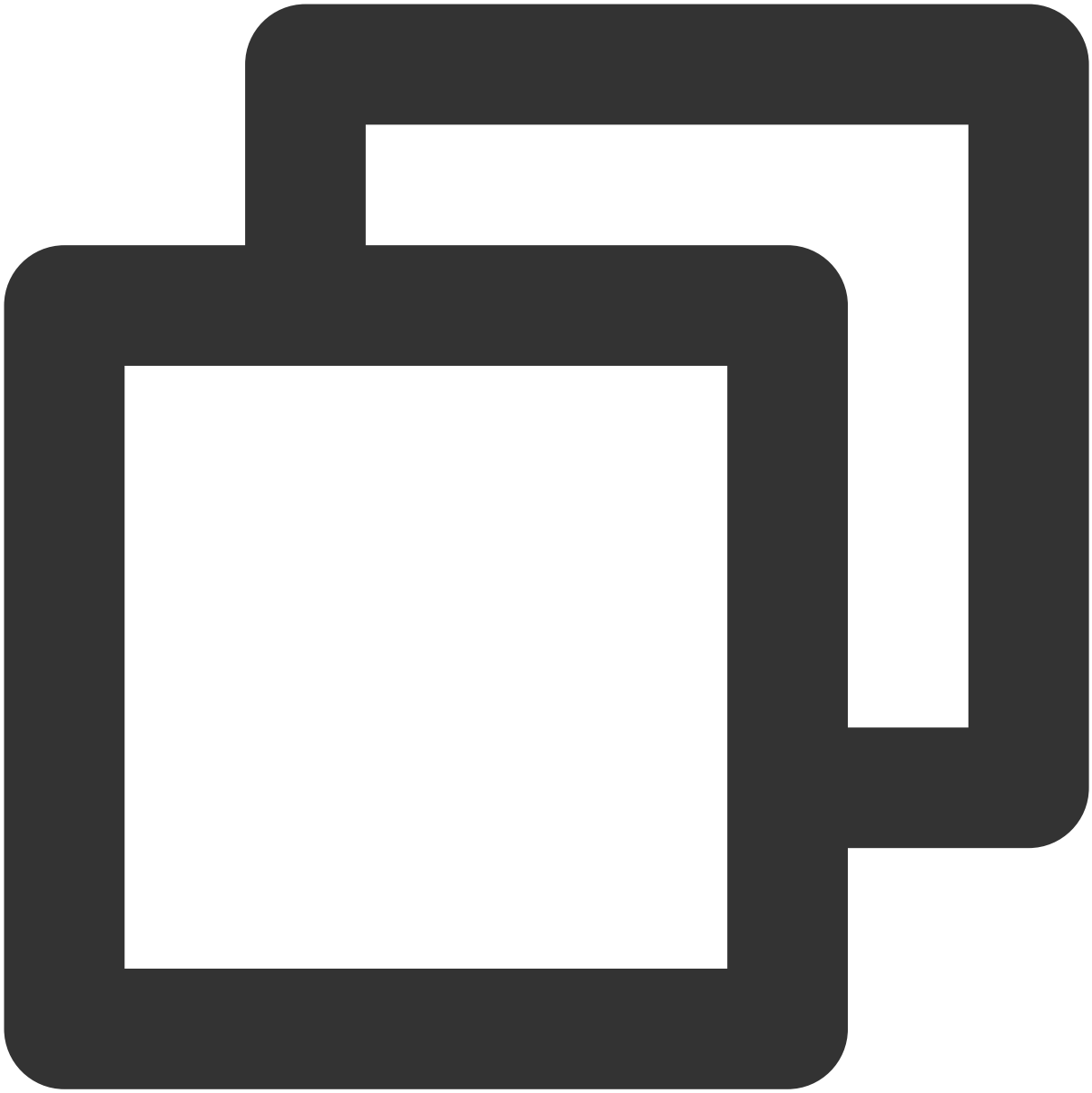


```
import { TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using

try {
   const params = {
     userIDList: ['user1', 'user2'],
     groupID: 'xxx',
     type: TUICallType.VIDEO_CALL,
   }
  await TUICallKitServer.groupCall(params);
} catch (error: any) {
  alert(`[TUICallKit] groupCall failed. Reason:${error}`);
}
```

加入群组通话

调用 `joinInGroupCall` API 主动加入群组中已有的音视频通话。

说明：Vue ≥ v3.1.2 支持



```
import { TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using

try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
```

```
    alert(`[TUICallKit] joinInGroupCall failed. Reason: ${error}`);
}
```

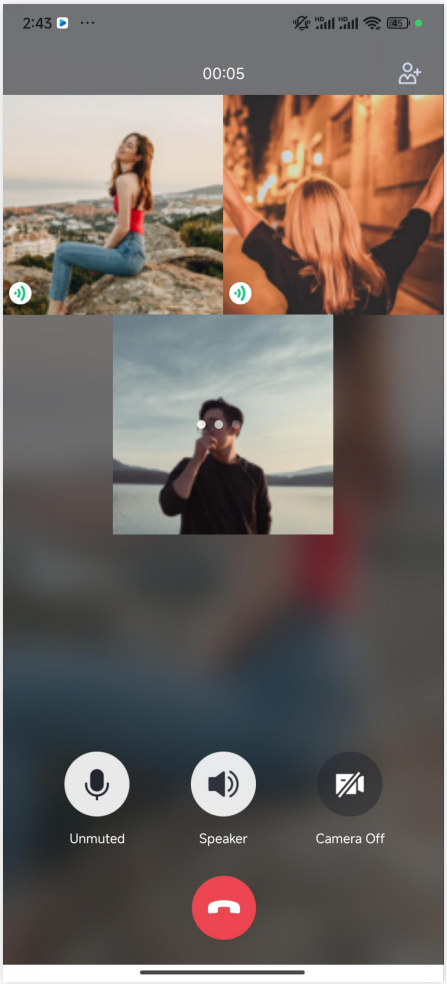
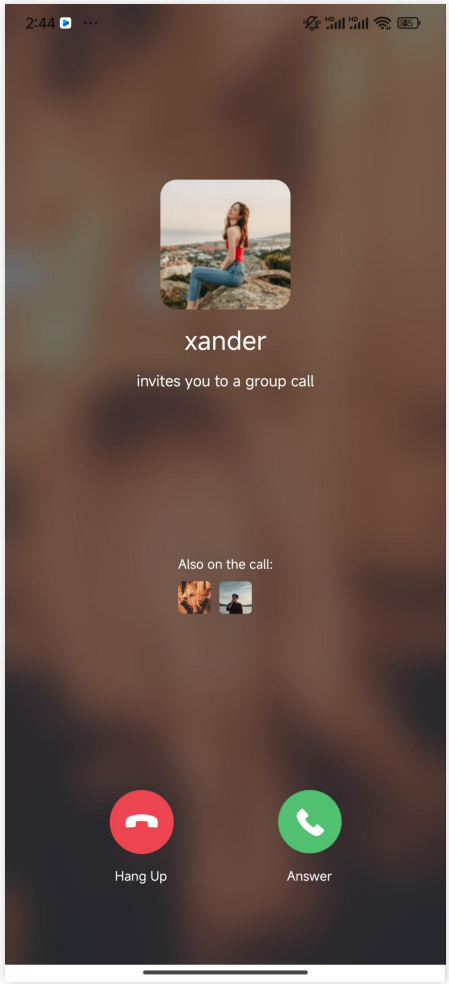
uni-app (Anroid&iOS)

最近更新时间：2024-04-03 17:23:11

本文介绍群组通话功能的使用，如发起群组通话、加入群组通话。

预期效果

TUICallKit 支持群组通话，预期效果见下图。

发起群组通话	收到群组通话邀请	接受群组
		

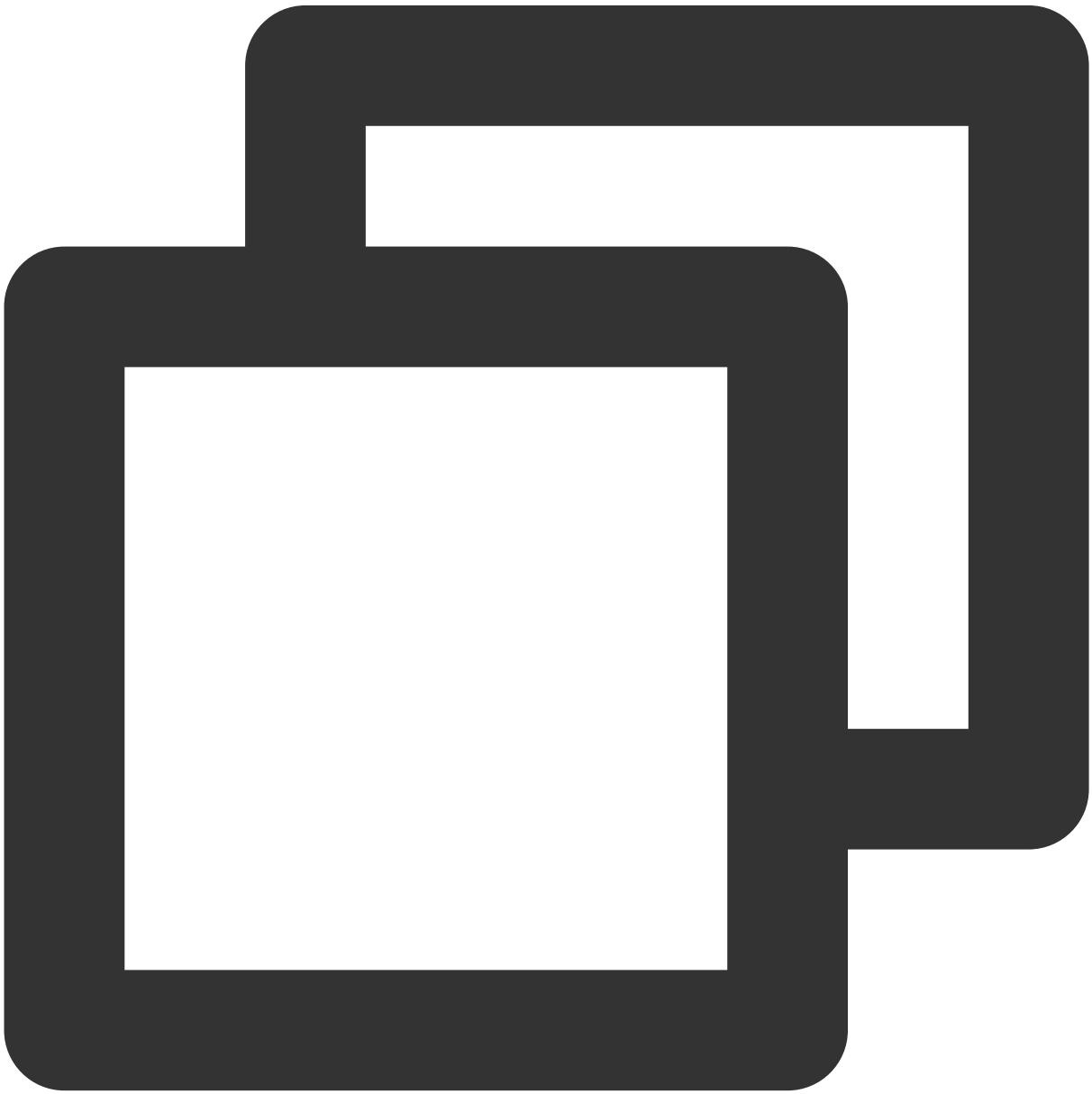
创建 groupID

使用群组通话功能前，需要先创建群组，在已存在的群组中发起群组通话。
方式一：通过调用 IM API 创建群，详见 IM 群组管理。
方式二：通过控制台手动创建群组，详见 控制台群组管理。

群组通话

发起群组通话

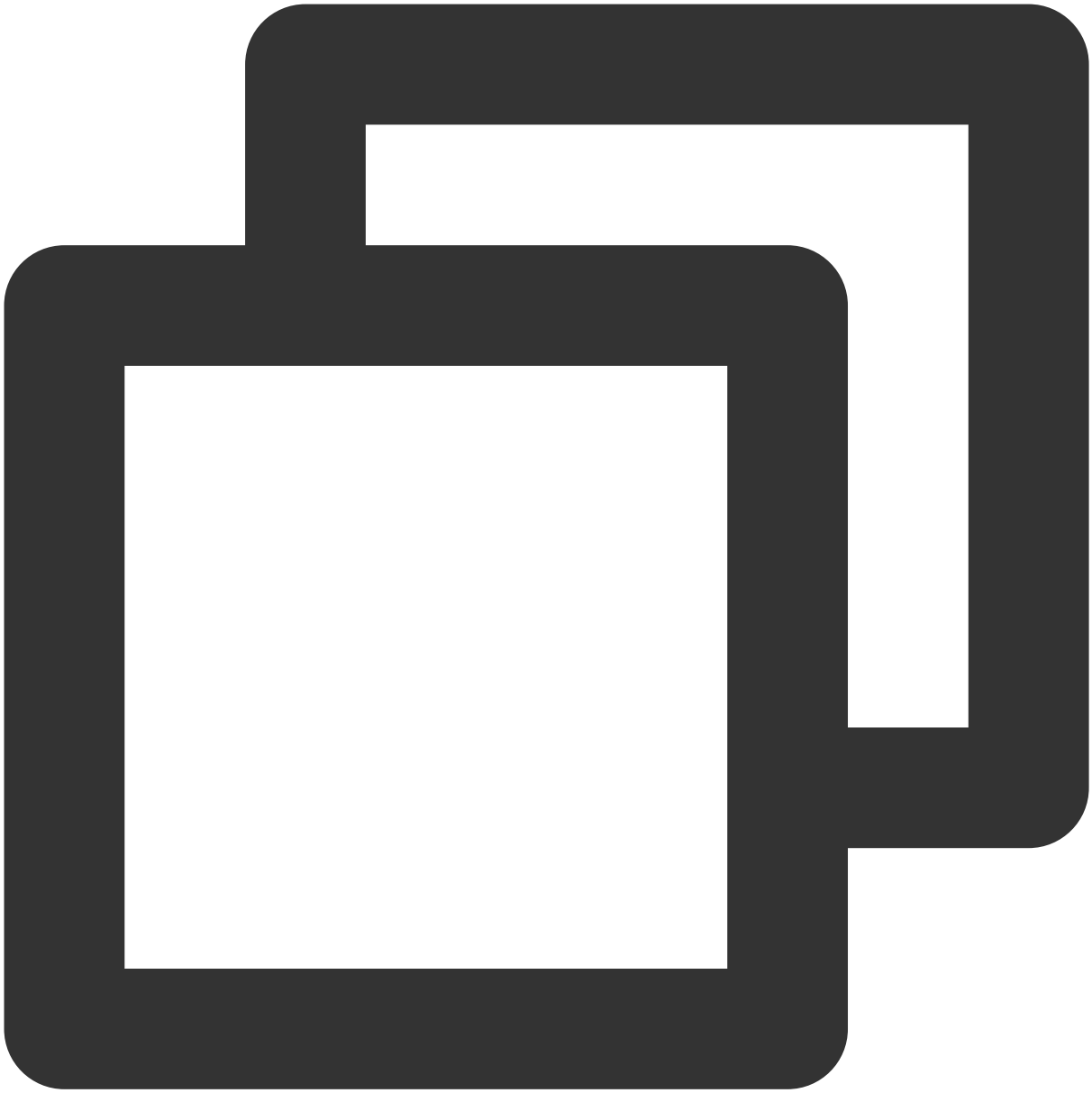
调用 groupCall API 发起群通话。



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // voice call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('groupCall success');
  } else {
    console.log(`groupCall failed, error message = ${res.msg}`);
  }
});
```

加入群组通话

调用 joinInGroupCall API 主动加入群组中已有的音视频通话。



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomID: 9898,
  groupID: 'myGroup',
  callMediaType: 1, // voice call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
```

```
    }
});
```

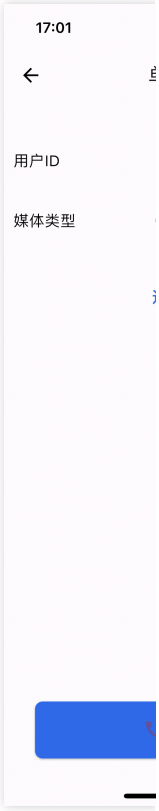
悬浮窗

Android&iOS&Flutter

最近更新时间：2024-04-03 17:23:11

本文介绍如何使用悬浮窗功能。

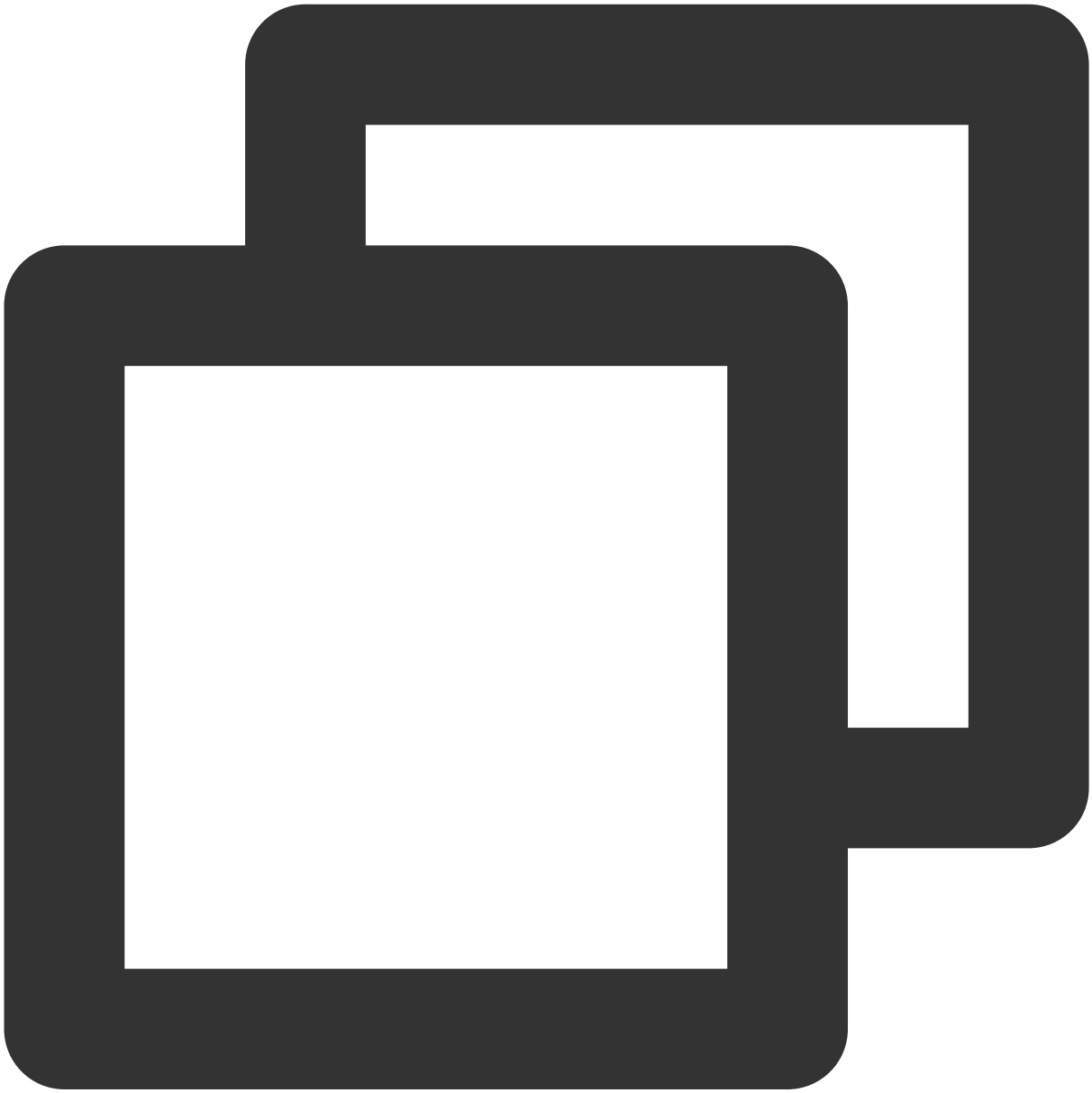
预期效果

开启悬浮窗按钮	语音通话悬浮窗	视频通话悬浮窗
		

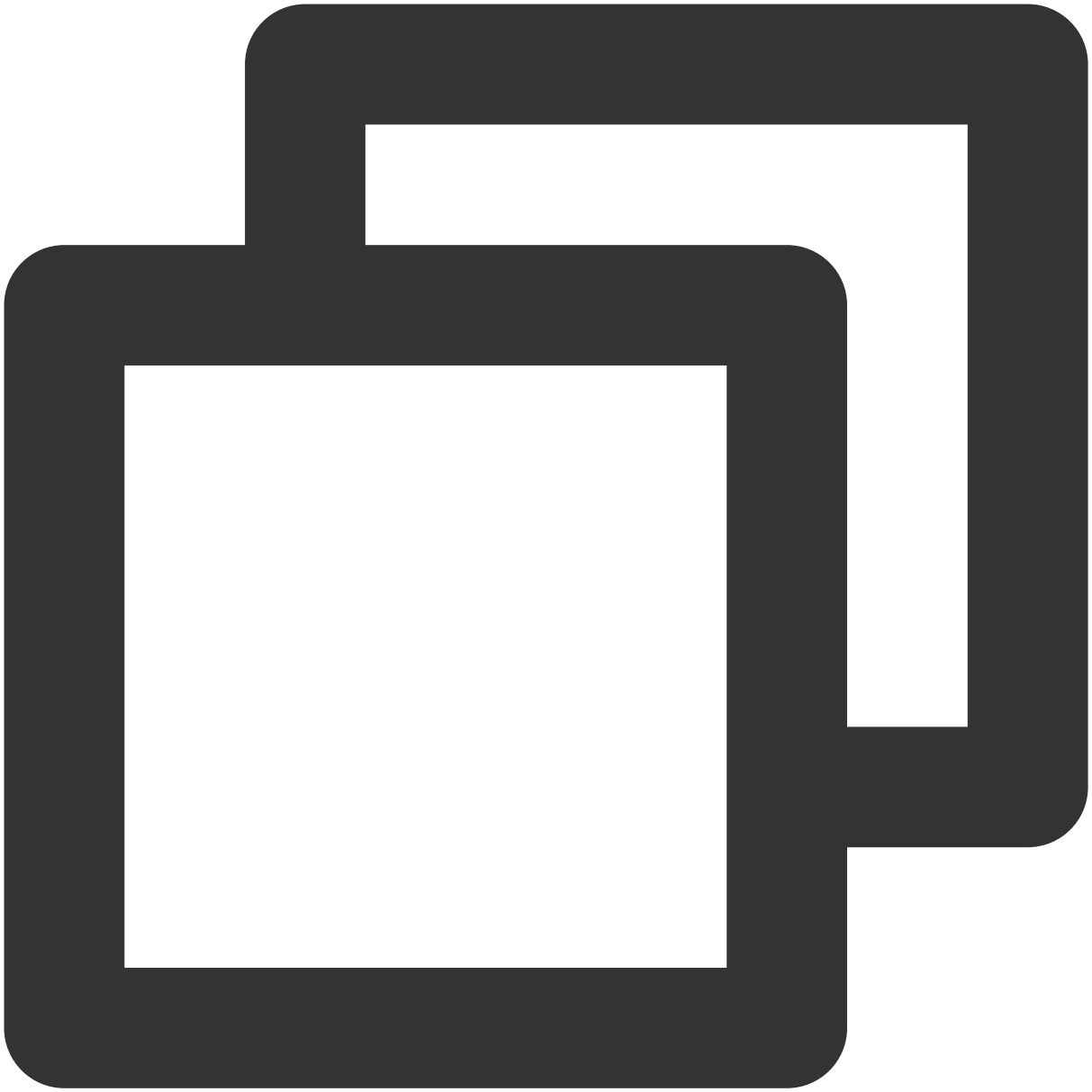
悬浮窗功能

TUICallKit 允许用户在通话时使用通话界面左上角的悬浮窗按钮将通话界面缩小为悬浮窗。
您的业务如果需要启用该功能，可以使用 enableFloatWindow 方法，在 TUICallKit 组件初始化时开启该功能：
Android（Kotlin）

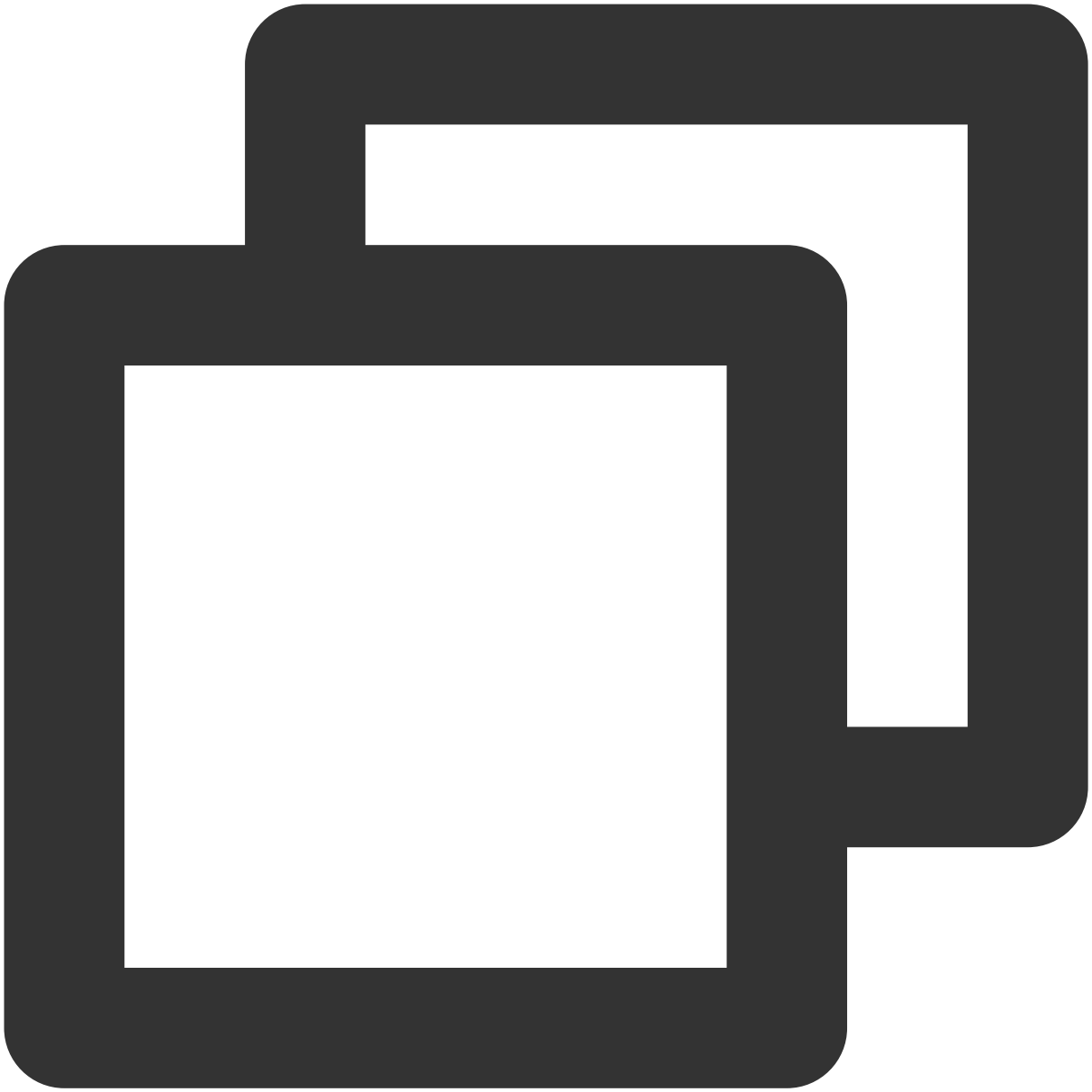
Android（Java）
iOS（Swift）
iOS（Objective-C）
Flutter（Dart）



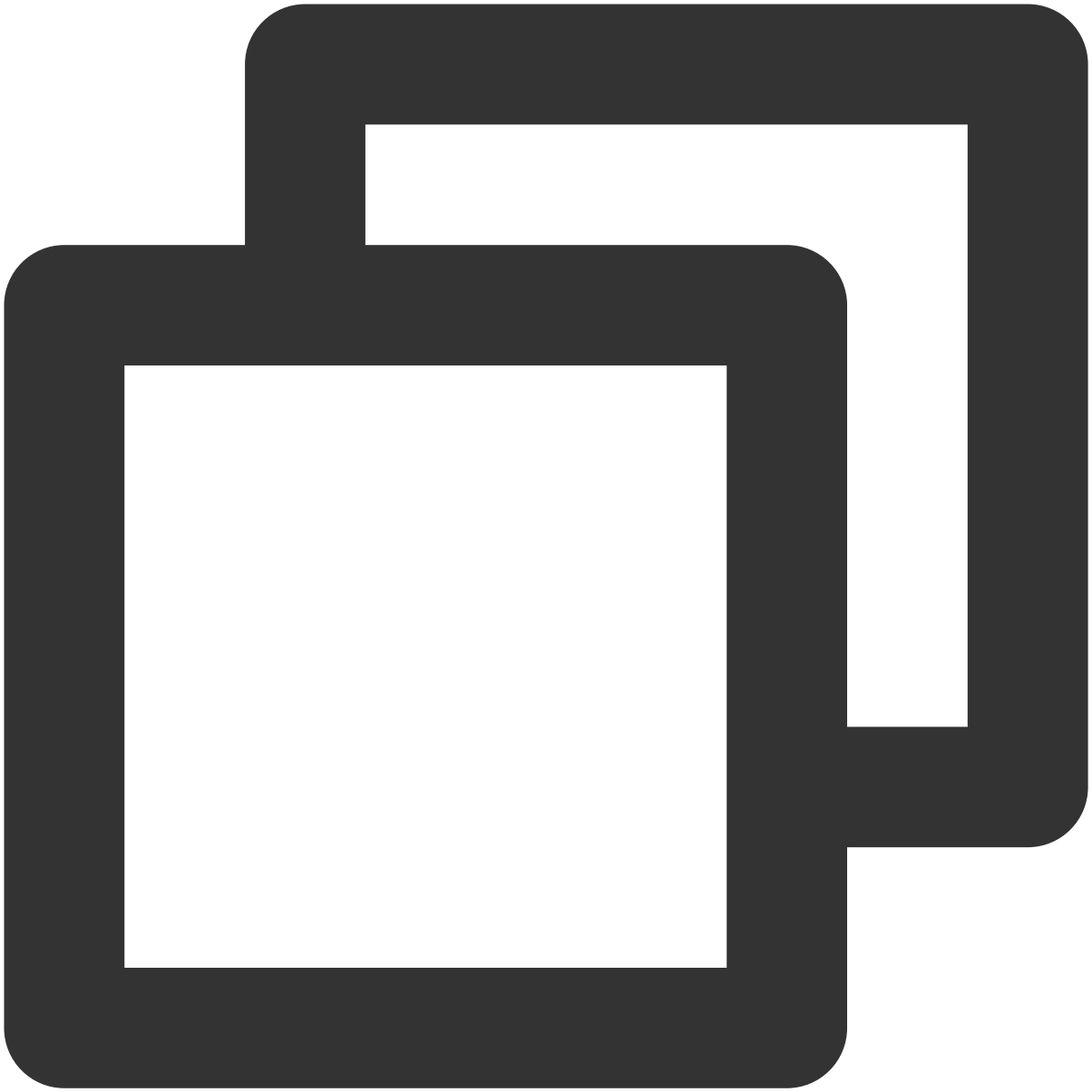
```
TUICallKit.createInstance(context).enableFloatWindow(true)
```



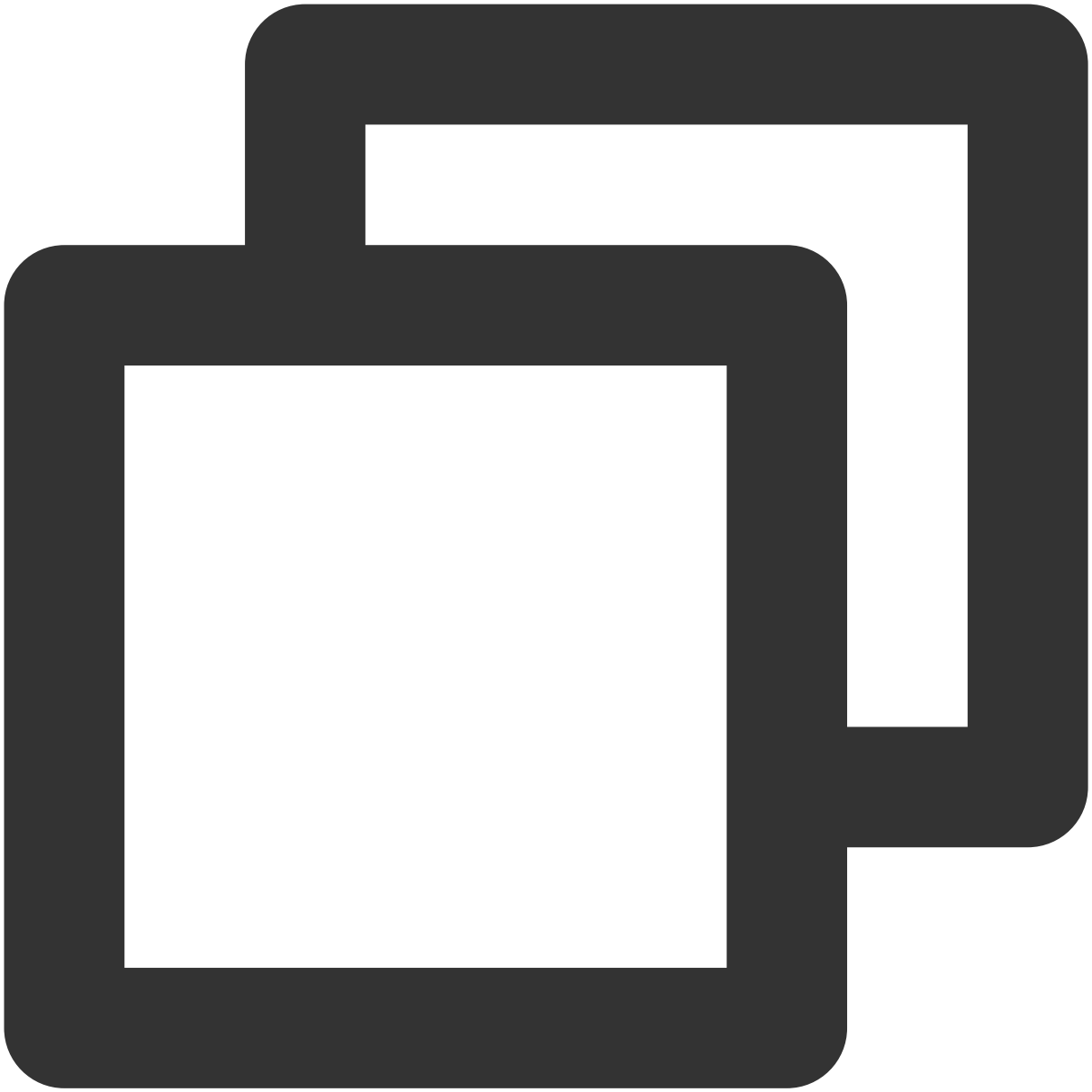
```
TUICallKit.createInstance(context).enableFloatWindow(true);
```



```
TUICallKit.createInstance().enableFloatWindow(true)
```



```
[[TUICallKit createInstance] enableFloatWindow:YES];
```



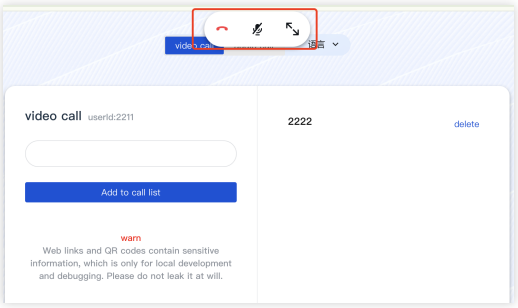
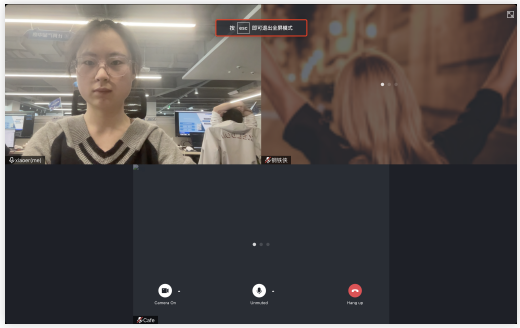
```
TUICallKit.instance.enableFloatWindow(true);
```

Web&H5

最近更新时间：2024-04-03 17:23:11

本文将介绍悬浮窗功能的使用。

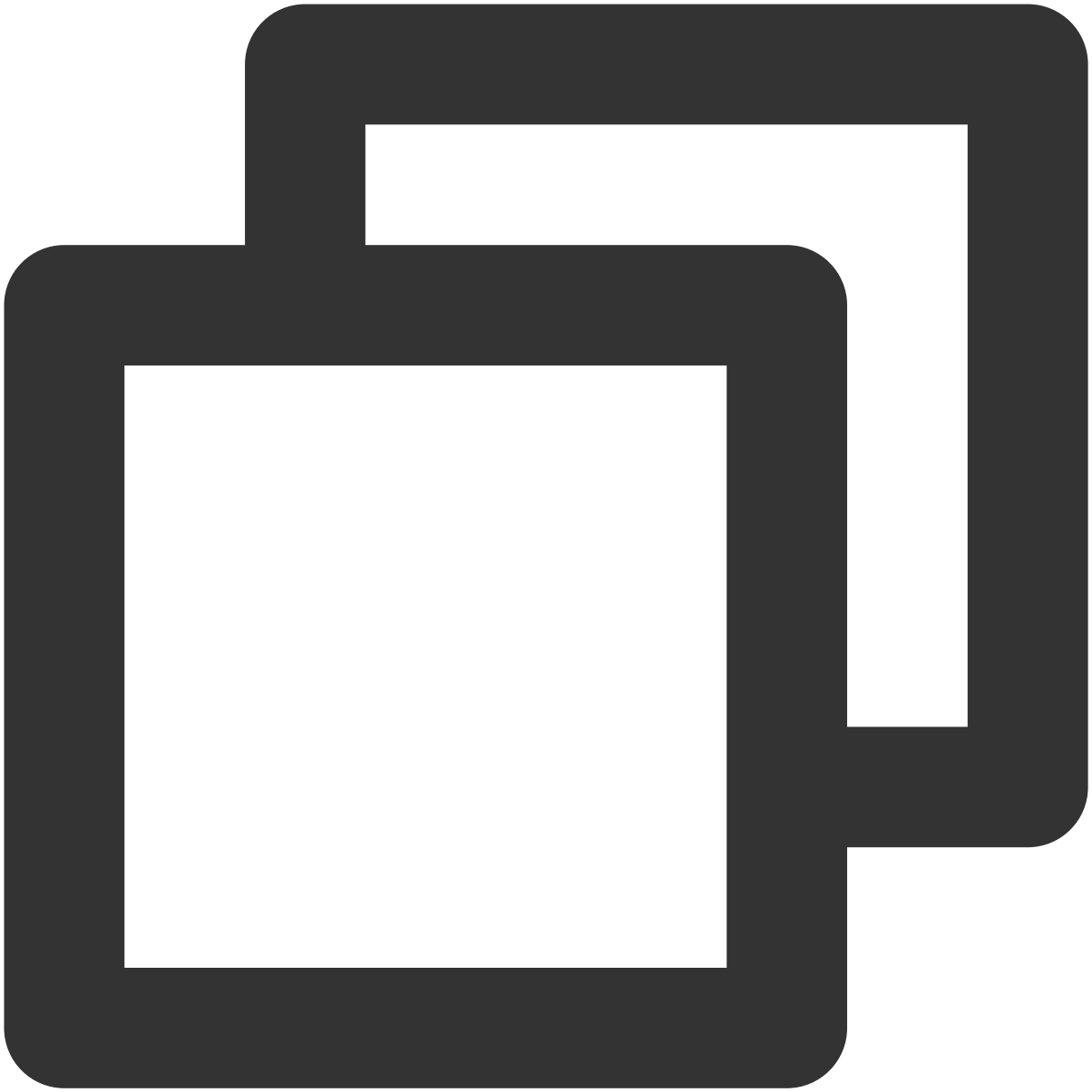
预期效果

Web 端悬浮窗	web 端全屏
	

悬浮窗功能

方式一：调用 `enableFloatWindow(enable: boolean)` API 开启/关闭悬浮窗。

说明：Vue ≥ v3.1.0 支持



```
try {
  await TUICallKitServer.enableFloatWindow(enable: Boolean)
} catch (error: any) {
  alert(`[TUICallKit] enableFloatWindow failed. Reason: ${error}`);
}
```

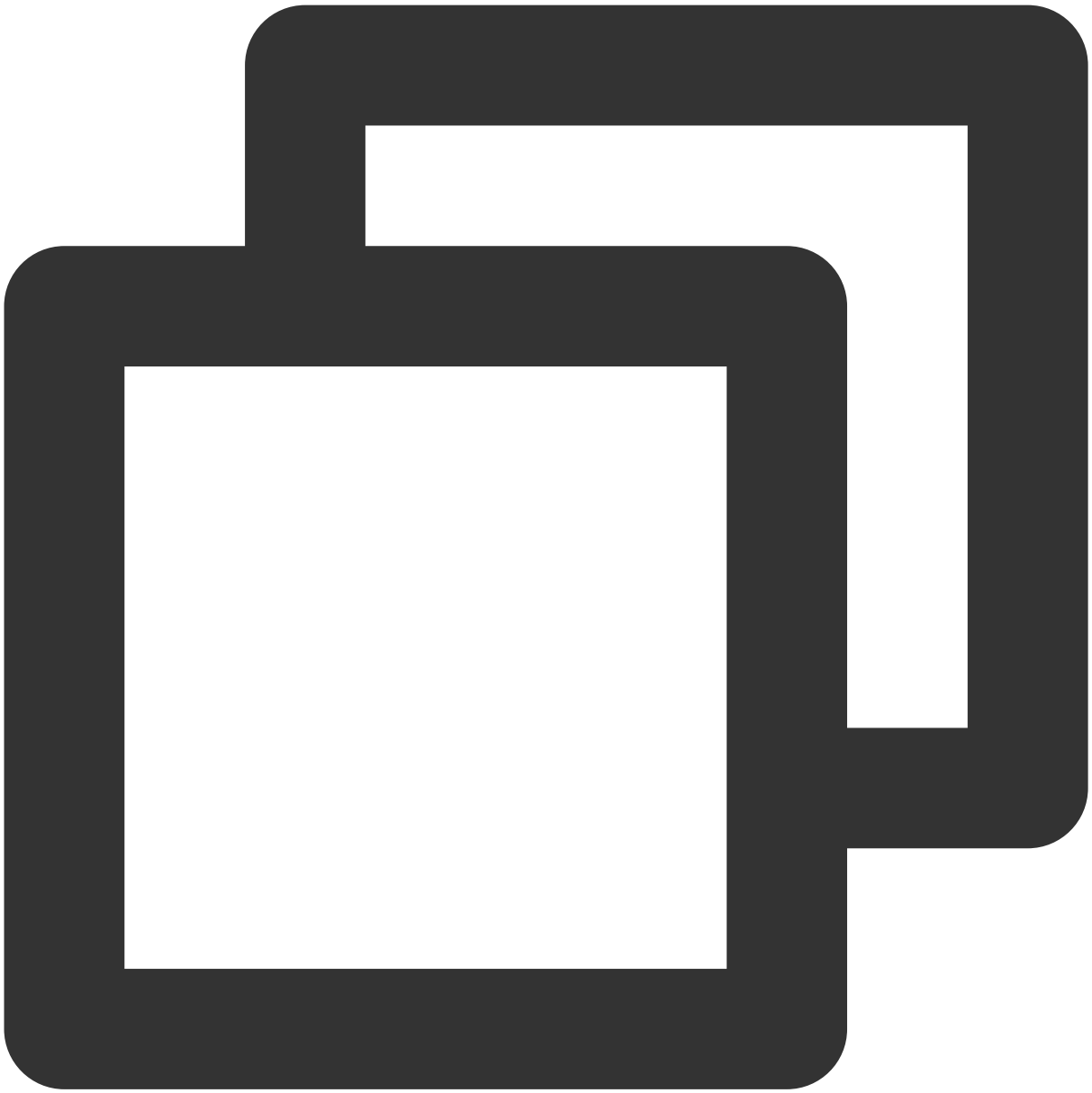
方式二：通过属性控制悬浮窗、全屏的开启/关闭。

`allowedMinimized` 属性控制控制悬浮窗开启/关闭。

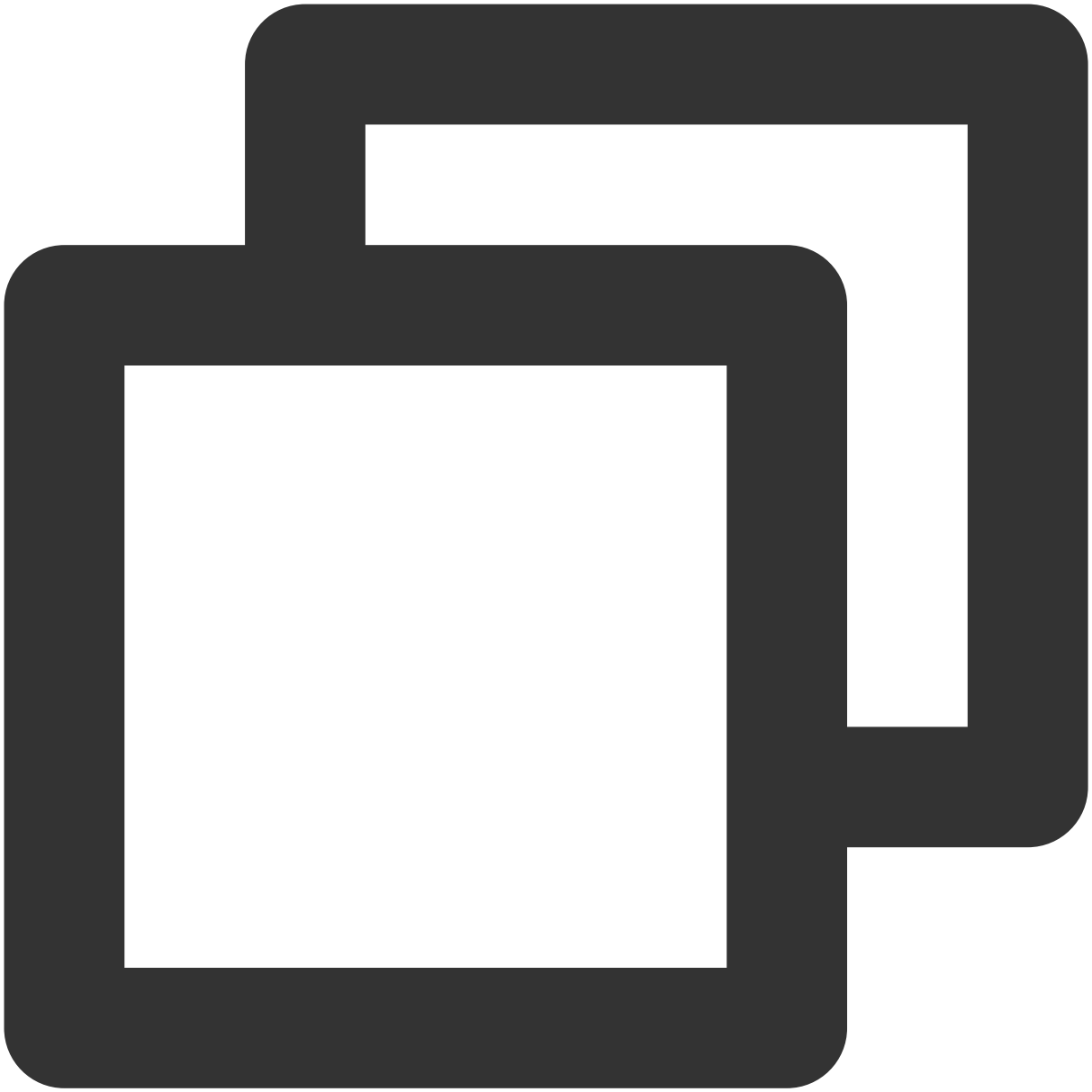
`allowedMinimized` 属性控制全屏的开启/关闭。

React

Vue



```
<TUICallKit
      allowedMinimized={true}
      allowedFullScree={true}
    />
```



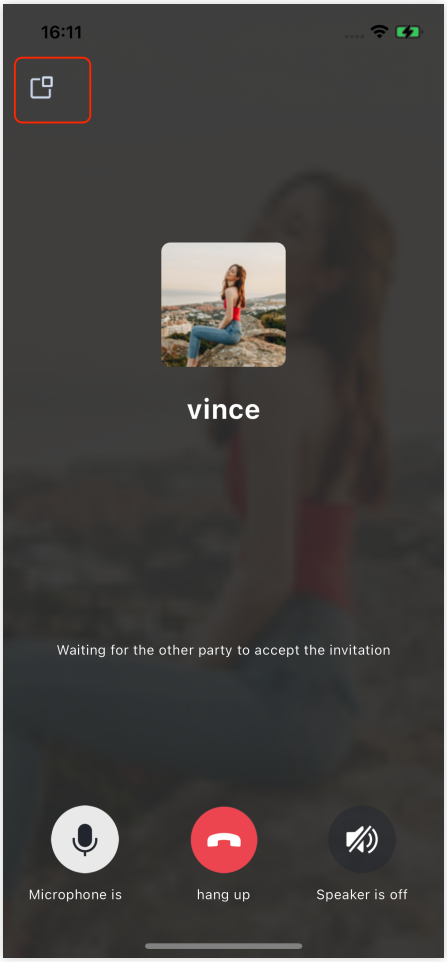
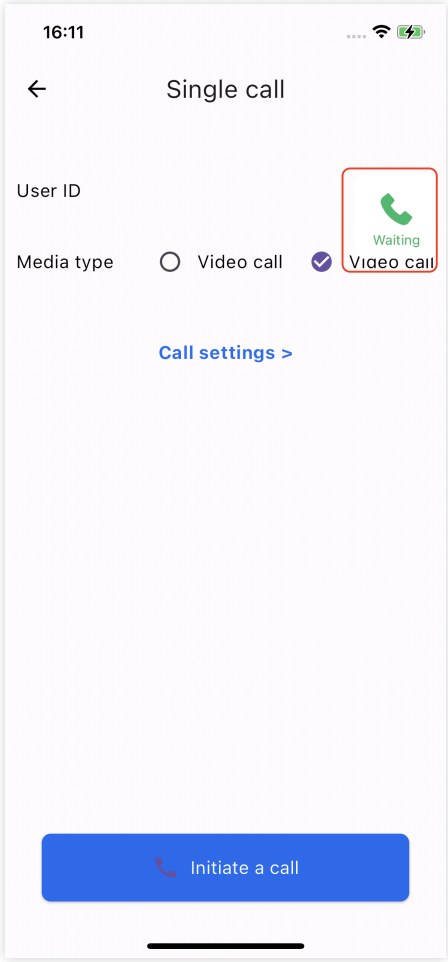
```
<TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
   />
```

uni-app (Anroid&iOS)

最近更新时间：2024-04-03 17:23:11

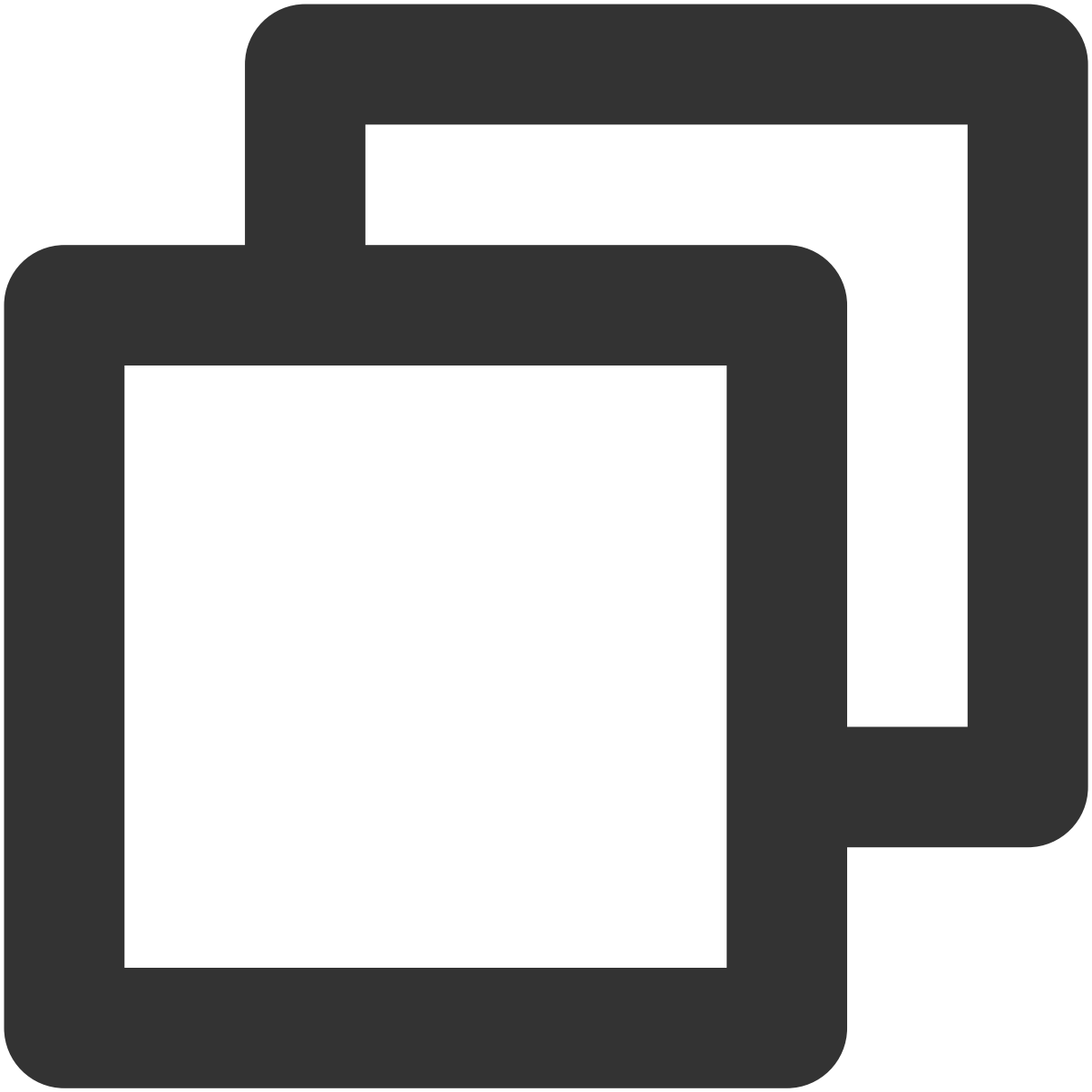
本文介绍如何使用悬浮窗功能。

预期效果

开启悬浮窗按钮	语音通话悬浮窗	视频通
		

悬浮窗功能

调用 `enableFloatWindow(enable: boolean)` API 开启/关闭悬浮窗。



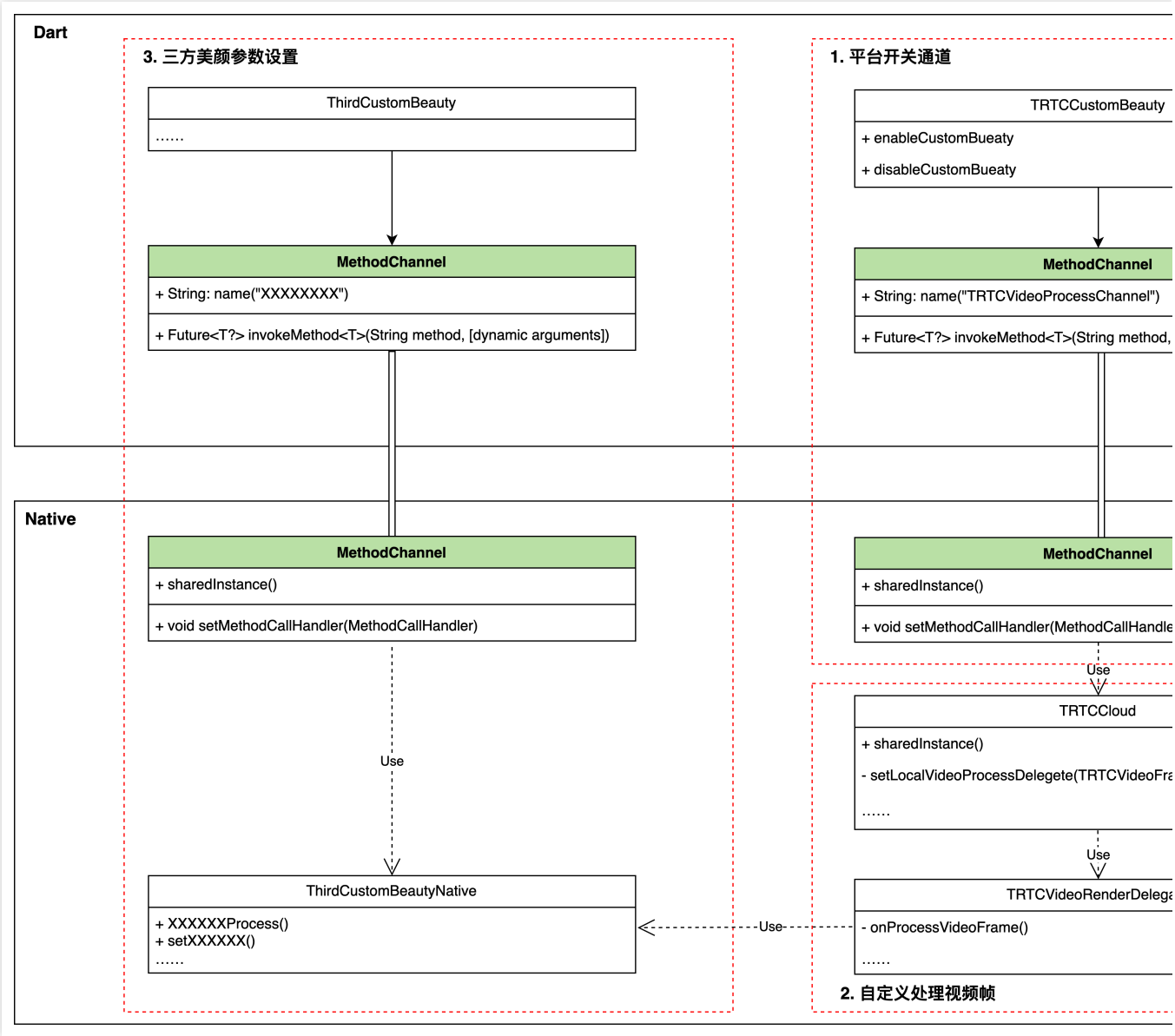
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

美颜特效

Flutter

最近更新时间：2024-04-03 17:23:11

本篇文档主要介绍在 TUICallKit 中接入美颜特效的方法。
在 Flutter 中完成自定义美颜处理，需要通过 TRTC 自定义视频渲染完成。由于 Flutter 不善于进行大量实时数据传输的特性，涉及 TRTC 的自定义视频渲染部分需要在 Native 部分完成。 具体的方案如下：



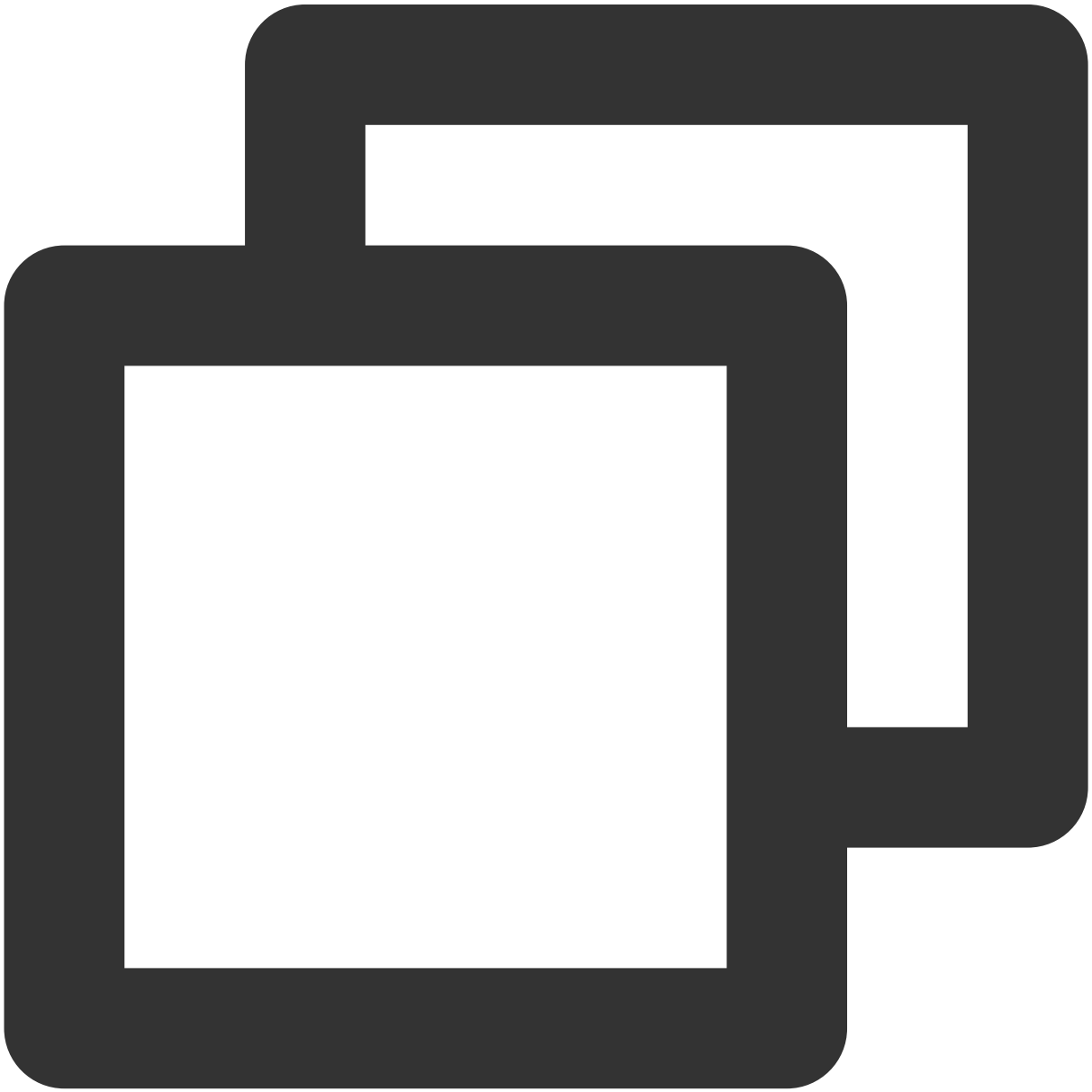
接入方案分为 3 步：
第一步：通过 MethodChannel 通道 开启/关闭 TRTC 自定义渲染逻辑。
第二步：在 TRTC 自定义渲染处理逻辑 onProcessVideoFrame() 中使用 `美颜处理模块` 对原始视频帧进行处理。

第三步：客户的美颜处理模块也需要在 Dart 通过接口设置当前美颜的参数，客户可以通过 MethodChannel 的方式设置美颜参数。这部分客户可根据自己的需求和使用的美颜自行定制。

接入第三方美颜特效

步骤1：实现 Dar t层到 Native 的“开始/结束”美颜的控制接口

实现 Dart 层接口:

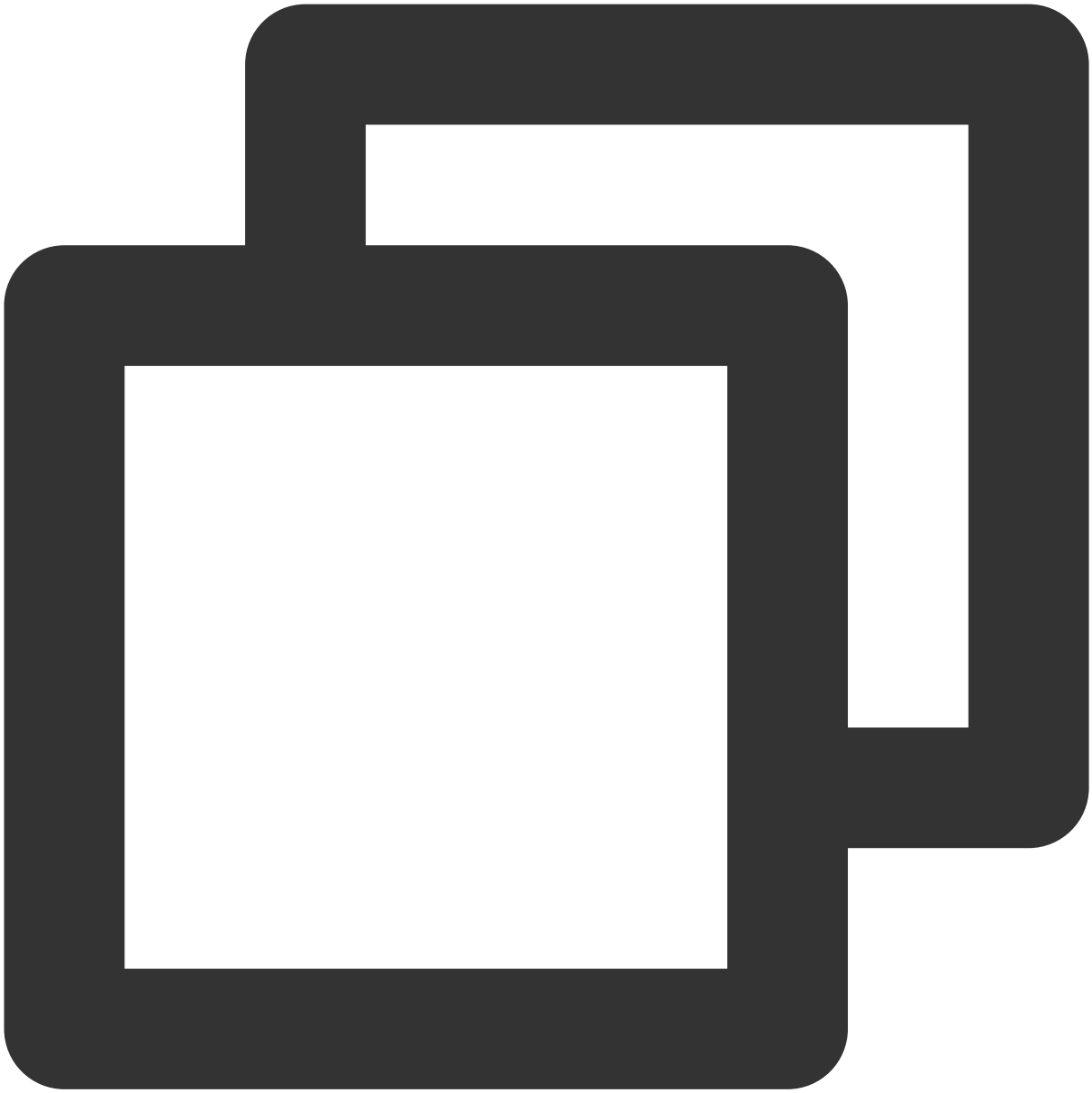


```
final channel = MethodChannel('TUICallKitCustomBeauty');

void enableTUICallKitCustomBeauty() async {
   await channel.invokeMethod('enableTUICallKitCustomBeauty');
}

void disableTUICallKitCustomBeauty() async {
   await channel.invokeMethod('disableTUICallKitCustomBeauty');
}
```

实现对应 Native 层接口:

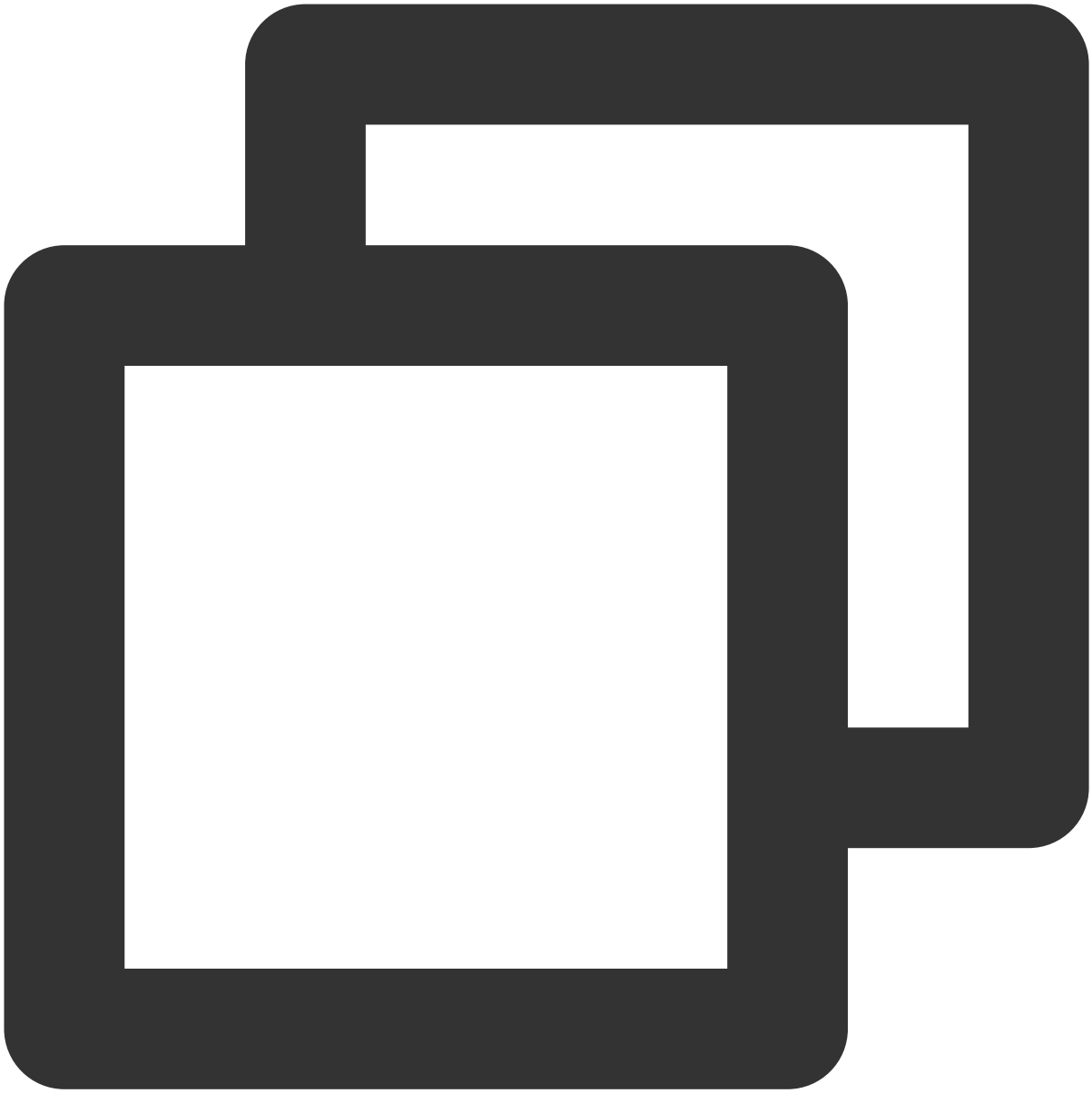
java

swift



```
public class MainActivity extends FlutterActivity {
    private static final String channelName = "TUICallKitCustomBeauty";

    private MethodChannel channel;

    @Override
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        super.configureFlutterEngine(flutterEngine);

        channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessen
        channel.setMethodCallHandler(((call, result) -> {
```

```
            switch (call.method) {
                case "enableTUICallKitCustomBeauty":
                    enableTUICallKitCustomBeauty();
                    break;
                case "disableTUICallKitCustomBeauty":
                    disableTUICallKitCustomBeauty();
                    break;
                default:
                    break;
            }
            result.success("");
        }));
    }

    public void enableTUICallKitCustomBeauty() {

    }

    public void disableTUICallKitCustomBeauty() {

    }
}
```



```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
    var channel: FlutterMethodChannel?

    override func application(_ application: UIApplication,
                             didFinishLaunchingWithOptions launchOptions: [UIAppli
        GeneratedPluginRegistrant.register(with: self)

        guard let controller = window?.rootViewController as? FlutterViewController
            fatalError("Invalid root view controller")
        }
```

```
        channel = FlutterMethodChannel(name: "TUICallKitCustomBeauty", binaryMessen
        channel?.setMethodCallHandler({ [weak self] call, result in
            guard let self = self else { return }
            switch (call.method) {
            case "enableTUICallKitCustomBeauty":
                self.enableTUICallKitCustomBeauty()
                break
            case "disableTUICallKitCustomBeauty":
                self.disableTUICallKitCustomBeauty()
                break
            default:
                break
            }
        })
        result(nil)
        return super.application(application, didFinishLaunchingWithOptions: launch
    }

    func enableTUICallKitCustomBeauty() {

    }

    func disableTUICallKitCustomBeauty() {

    }
}
```

步骤2：在 Native 的 TRTC 自定义渲染逻辑中完成美颜处理

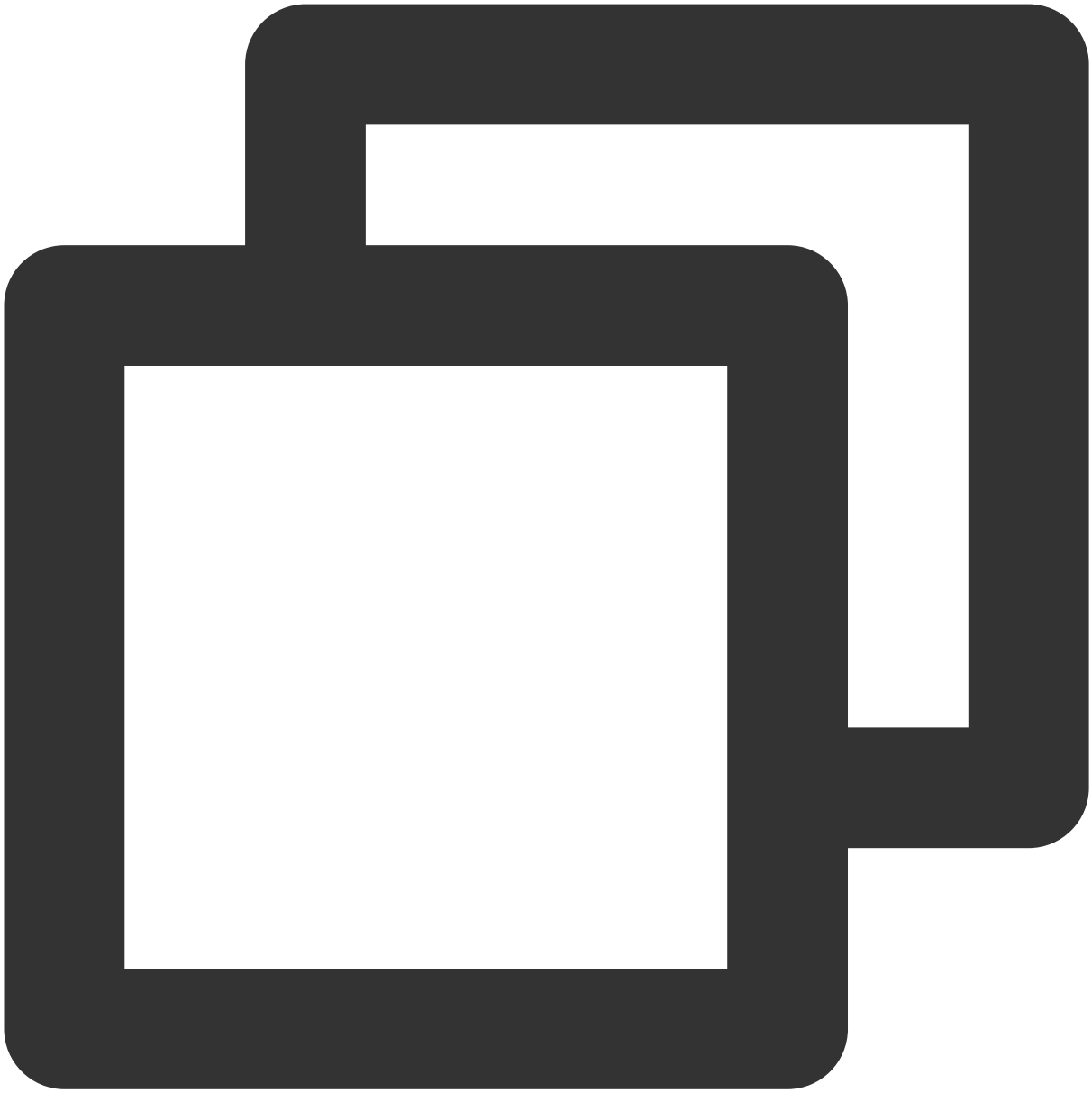
注意：

Android 在接入美颜过程中需要先依赖 **LiteAVSDK_Professional，** **在Android工程的 app/build.gradle中添加一下依赖：**

```
dependencies{
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
```

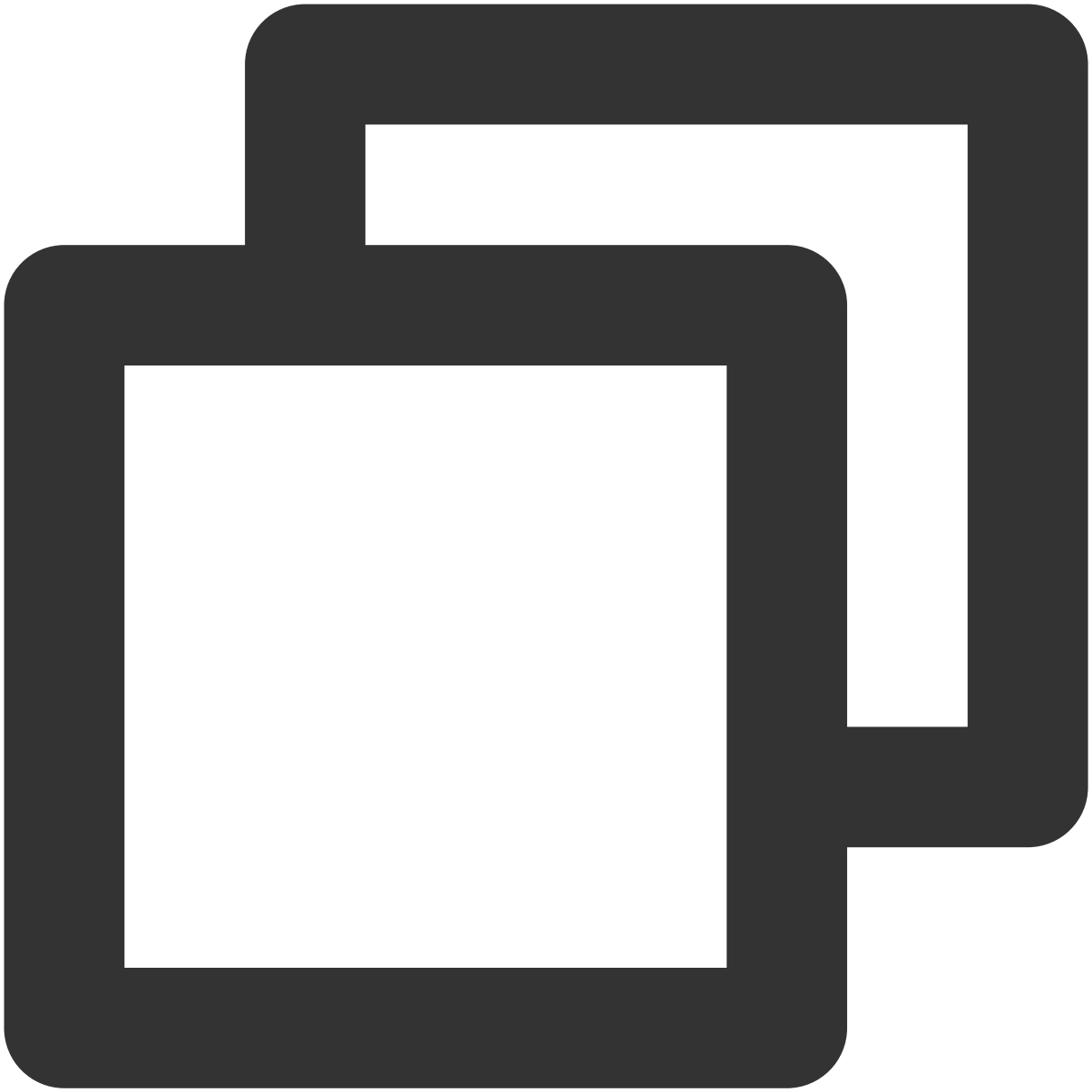
java

swift



```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());

}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
```

```
}

class VideoFrameListerer implements TRTCCloudListener.TRTCVideoFrameListener {
    private XXXBeautyModel  mBeautyModel = XXXBeautyModel.sharedInstance();

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
        // 美颜处理逻辑
        mBeautyModel.process(trtcVideoFrame, trtcVideoFrame1);
        ……

        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

    @Override
    public void onGLContextDestory() {
    }
}
```



```
let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pix
}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Tex
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
```

```
    let bueutyModel = XXXXBeautyModel.shareIntance()
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame)
        // 美颜处理逻辑
        bueutyModel.onProcessVideoFrame(srcFrame, dstFrame)
        ……

        return 0
    }
}
```

步骤3：客户自定义第三方美颜参数控制逻辑

此部分客户可以根据自己的需求和具体使用的美颜模块，参考 步骤1 种实现美颜参数的设置功能，具体实现根据具体使用情况决定。

接入腾讯美颜特效

腾讯美颜特效的接入方法也同样遵循上述方法，现在以腾讯美颜特效为例，详细介绍接入方法：

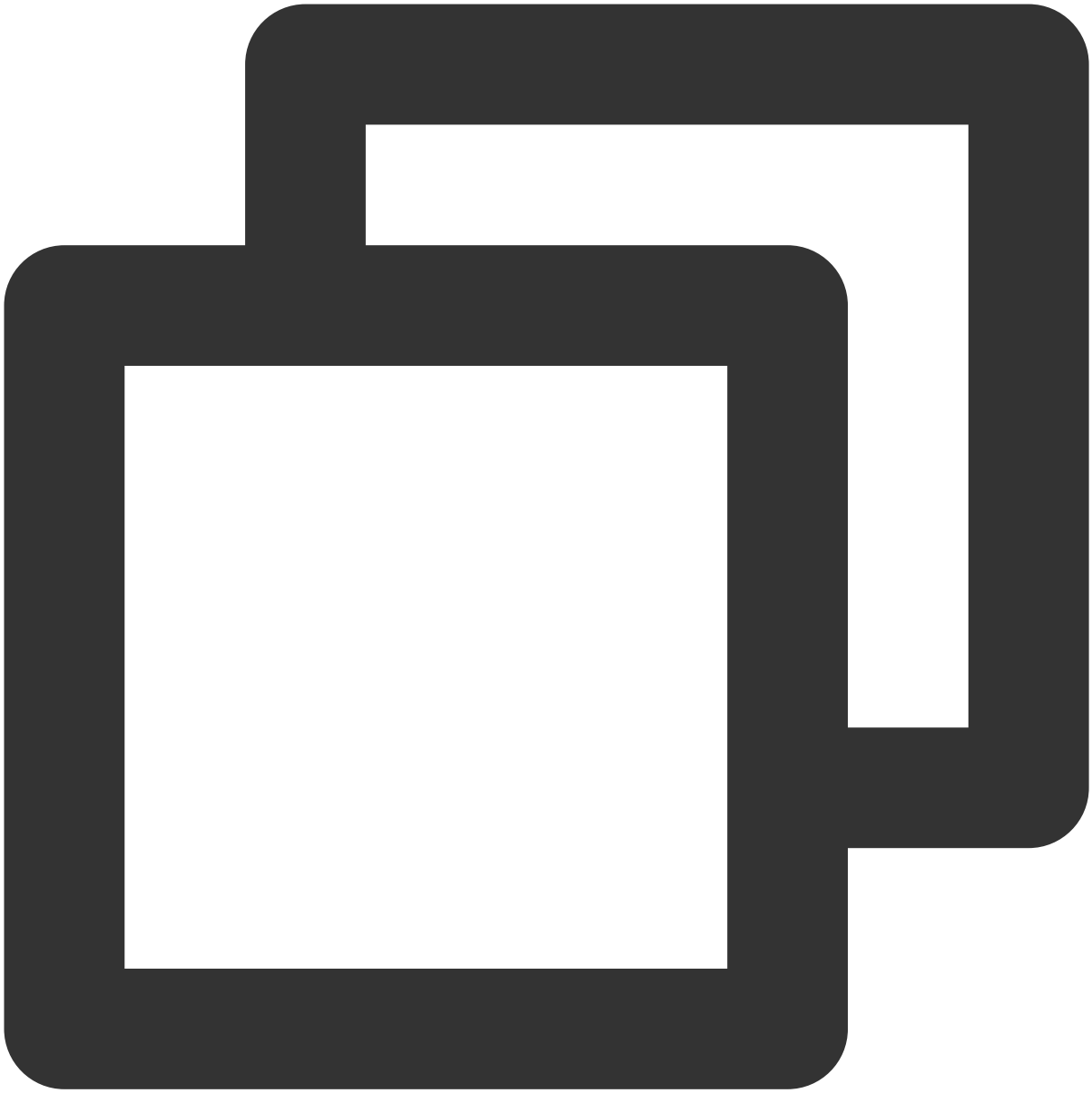
步骤1：美颜资源下载与集成

1. 根据您购买的套餐 下载 SDK。
2. 添加文件到自己的工程中：

Android

iOS

1. 在 app 模块下找到 build.gradle 文件，添加您对应套餐的 maven 引用地址，例如您选择的是S1-04套餐，则添加如下：

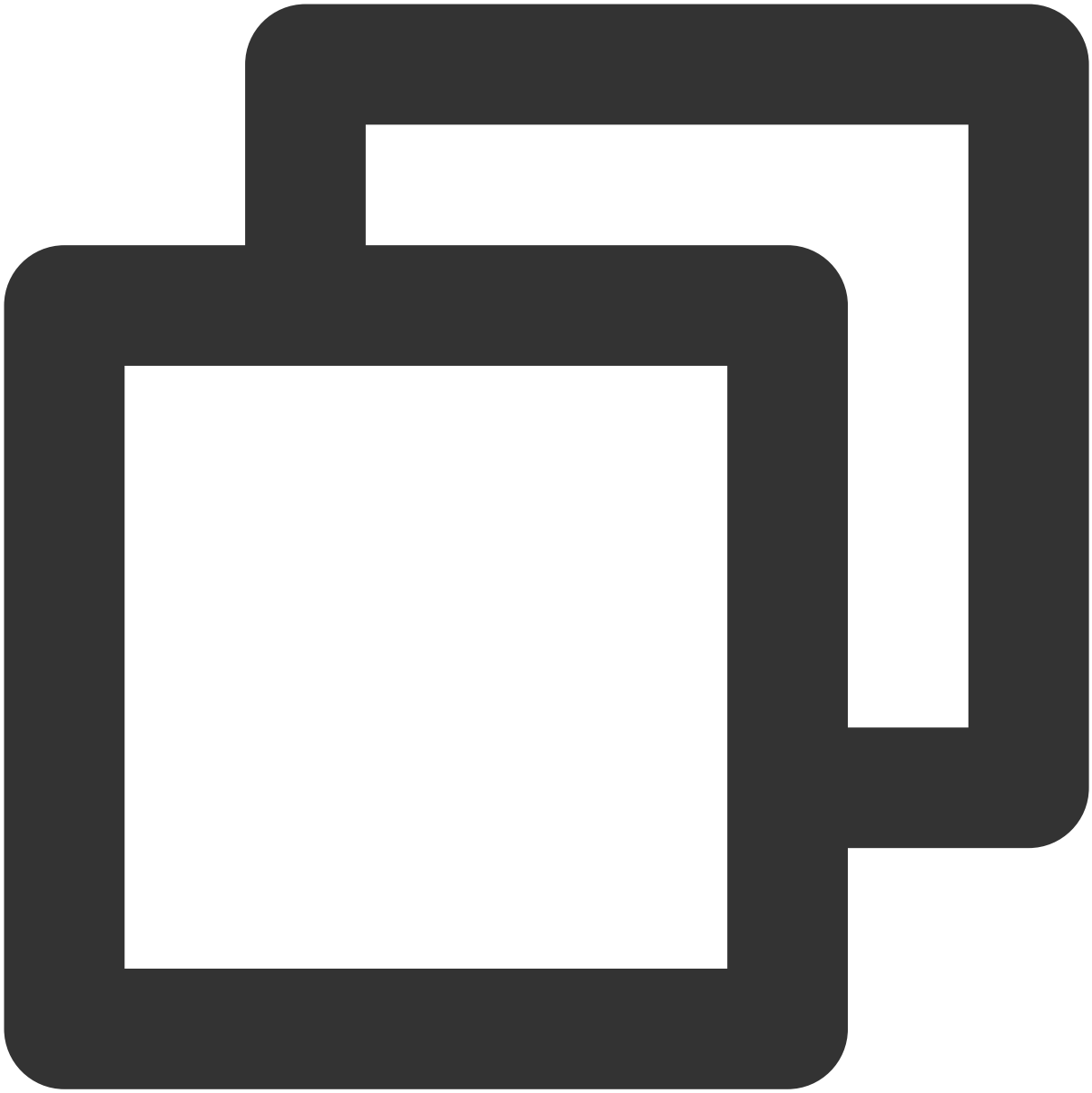


```
dependencies {
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

各套餐对应的 maven 地址，请参见 文档。

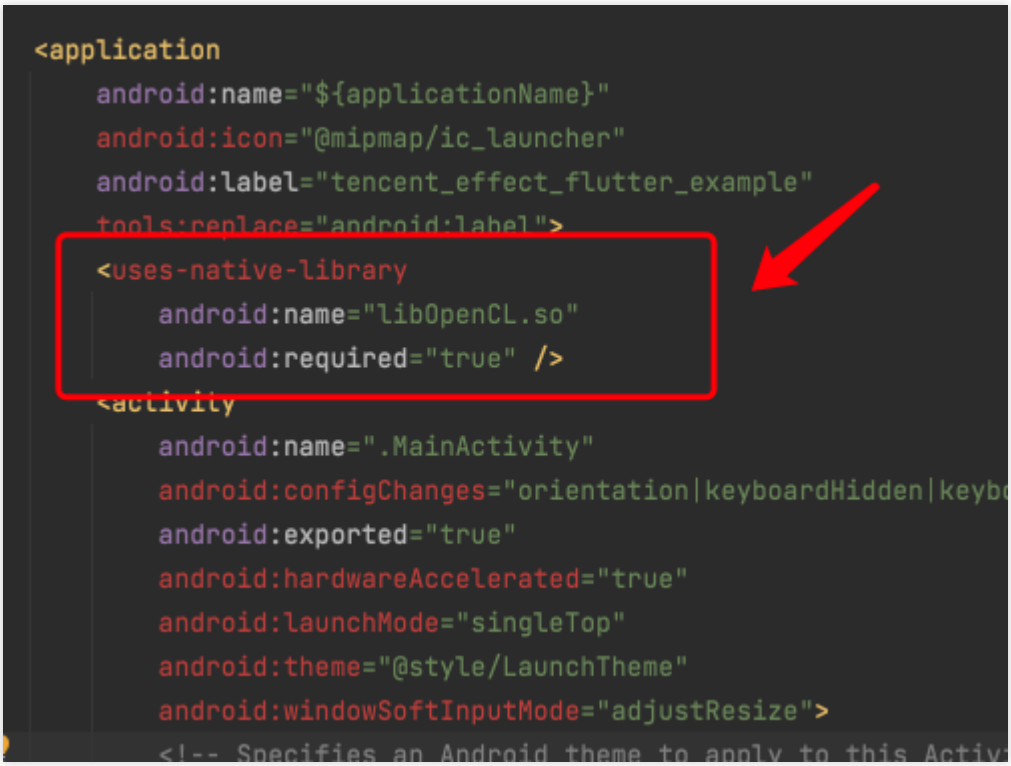
2. 在 app 模块下找到 src/main/assets 文件夹，如果没有则创建，检查下载的 SDK 包中是否有 MotionRes 文件夹，如果有则将此文件夹拷贝到 `../src/main/assets` 目录下。

3. 在 app 模块下找到 AndroidManifest.xml 文件，在 application 表填内添加如下标签。



```
<uses-native-library
       android:name="libOpenCL.so"
       android:required="true" />
       //此处的true 表示如果没有此库，则应用将无法正常运行。系统不允许在没有此库的设备上安装应用
       //false表示应用可以使用此库（如果存在），但专门在没有此库的情况下运行（如果有必要）。系统
       //Android 官网介绍：https://developer.android.com/guide/topics/manifest/uses-
```

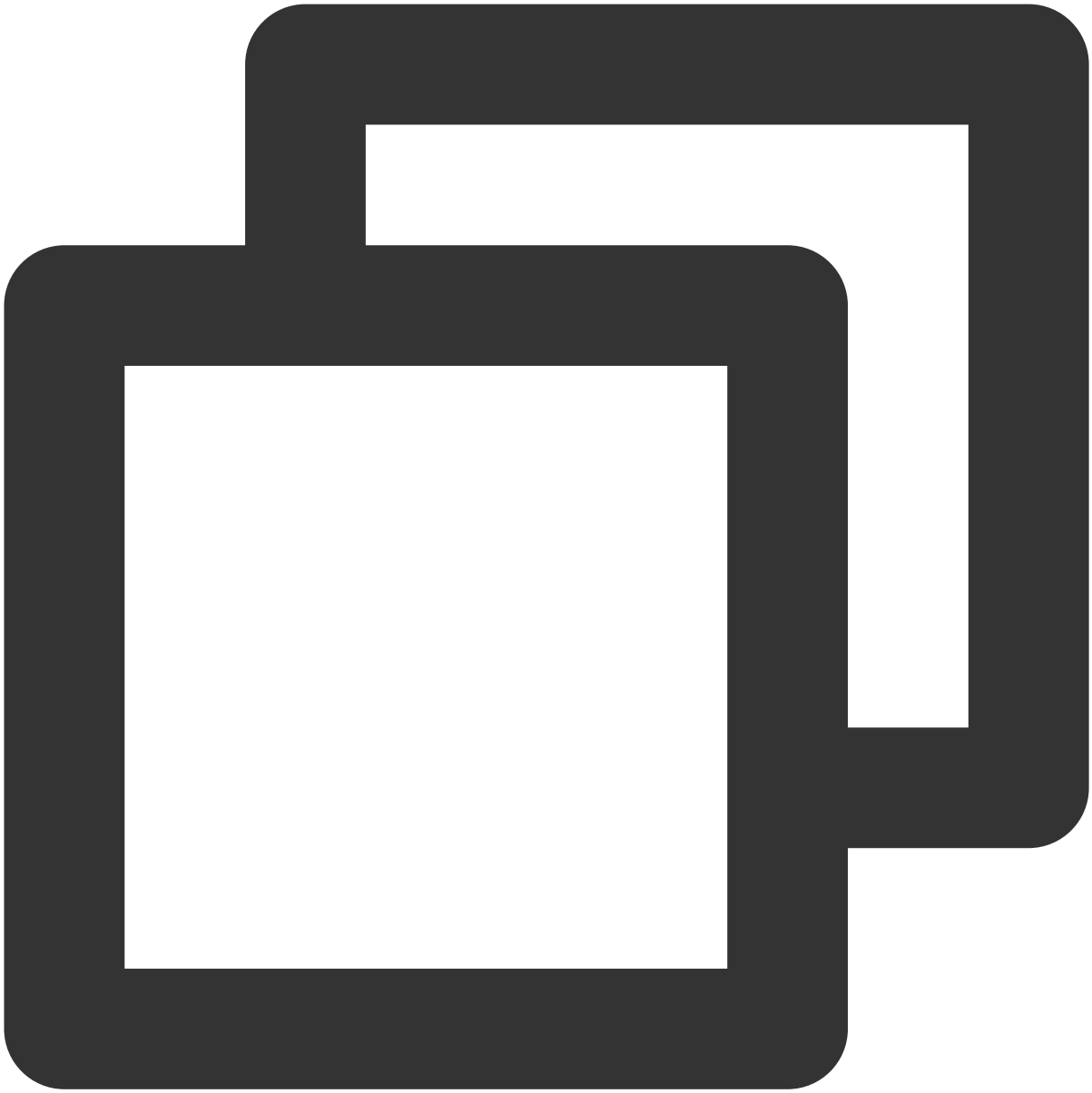
添加后如下图：



4. 混淆配置

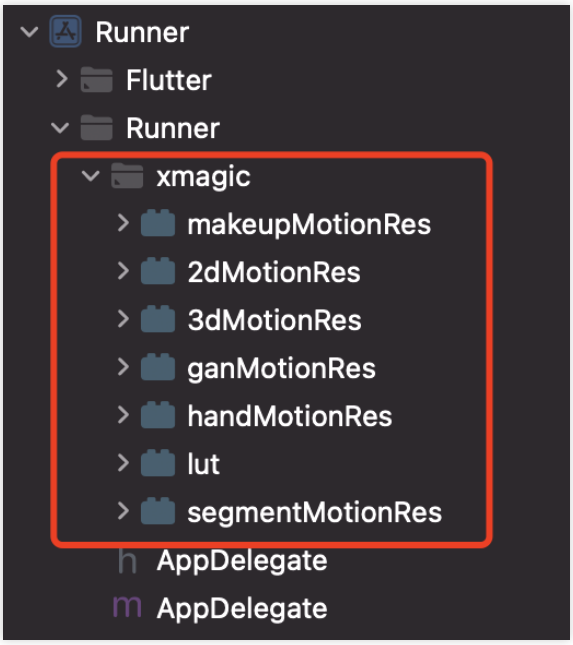
如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。

如果您启用了这样的编译优化，那就要添加这些 keep 规则，防止 xmagic 的代码被裁掉：



```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
-keep class androidx.exifinterface.** { *;}
```

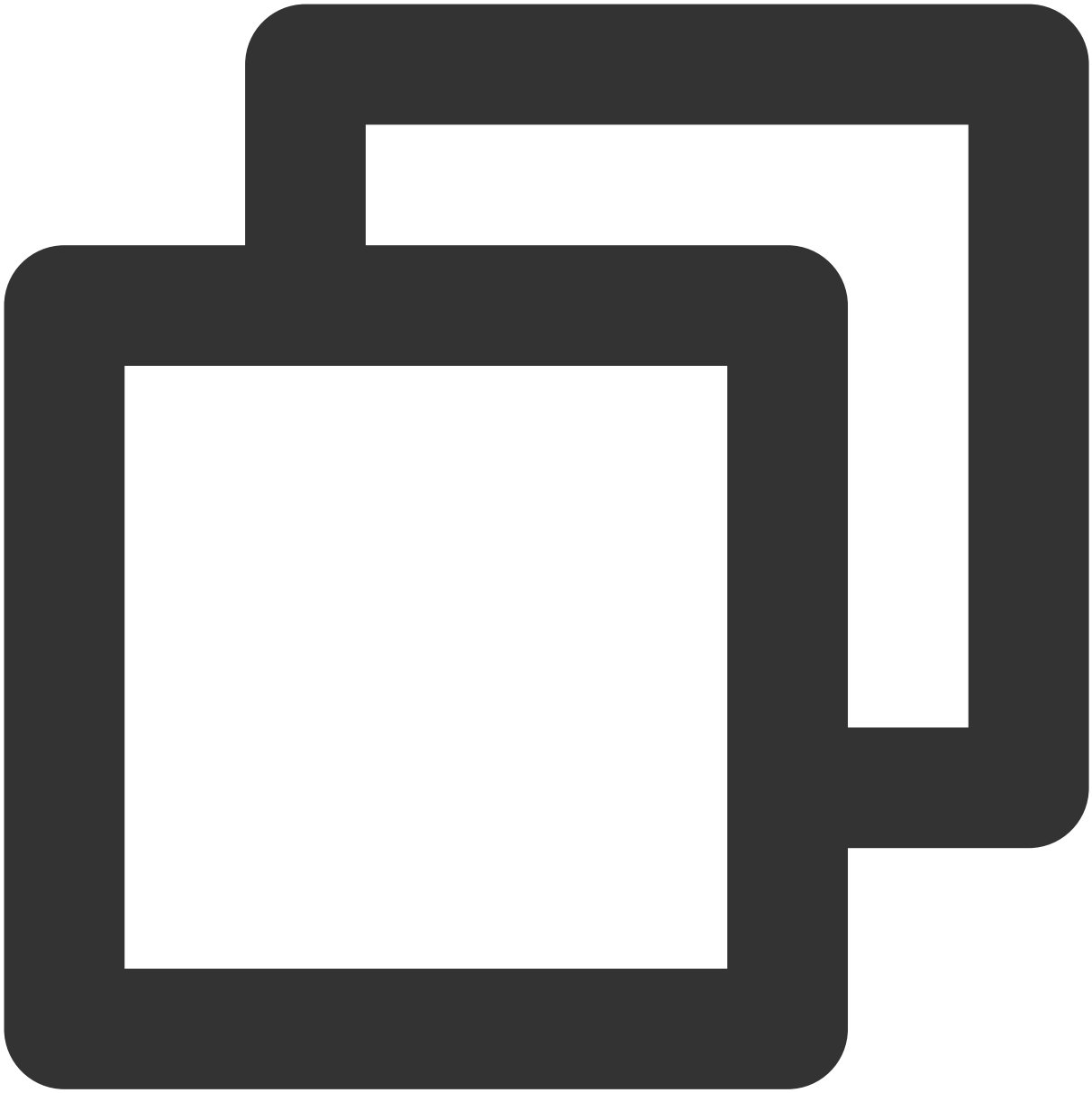
1. 添加美颜资源到您的工程。添加后如下图（您的资源种类跟下图不完全一致）：



2. 在 demo 中把 demo/lib/producer 里面的4个类：BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid 和 BeautyPropertyProducerIOS 复制添加到自己的 Flutter 工程中，这4个类是用来配置美颜资源，把美颜类型展示在美颜面板中。

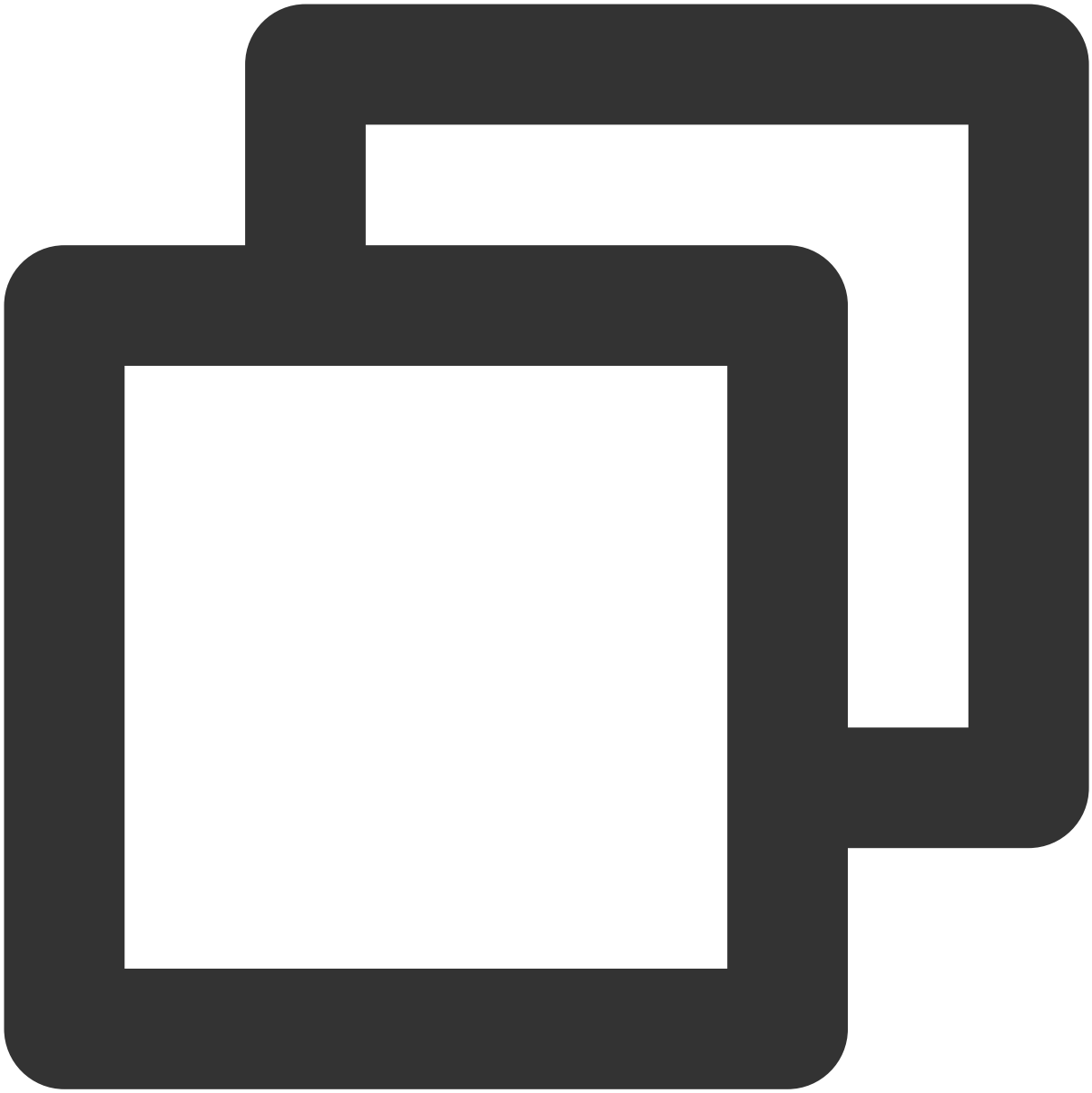
步骤2：引用 Flutter 版本 SDK

GitHub 引用：在工程的 pubspec.yaml 文件中添加如下引用：



```
tencent_effect_flutter:
  git:
    url: https://github.com/TencentCloud/tencenteffect-sdk-flutter
```

本地引用：从 tencent_effect_flutter下载最新版本的 tencent_effect_flutter，然后把文件夹 `android` 、 `ios` 、 `lib` 和文件 `pubspec.yaml` 、 `tencent_effect_flutter.iml` 添加到工程目录下，然后在工程的 pubspec.yaml 文件中添加如下引用：（可参考demo）



```
tencent_effect_flutter:
    path: ../
```

tencent_effect_flutter 只是提供一个桥接，里面依赖的 XMagic 默认是最新版的，真正实现美颜是 XMagic。

如果要使用最新版本的美颜 SDK，您可以通过以下步骤进行 SDK 升级：

Android

iOS

在工程目录下执行命令：`flutter pub upgrade` 或者在 `subspec.yaml` 页面的右上角单击 **Pub upgrade**。

在工程目录下执行命令：flutter pub upgrade，然后在 iOS 目录下执行命令： `pod update` 。

步骤3：实现Dart层到Native的 开始/结束 美颜的控制接口

此部分可以参考 接入第三方美颜特效/步骤1 ，此处不在赘述。

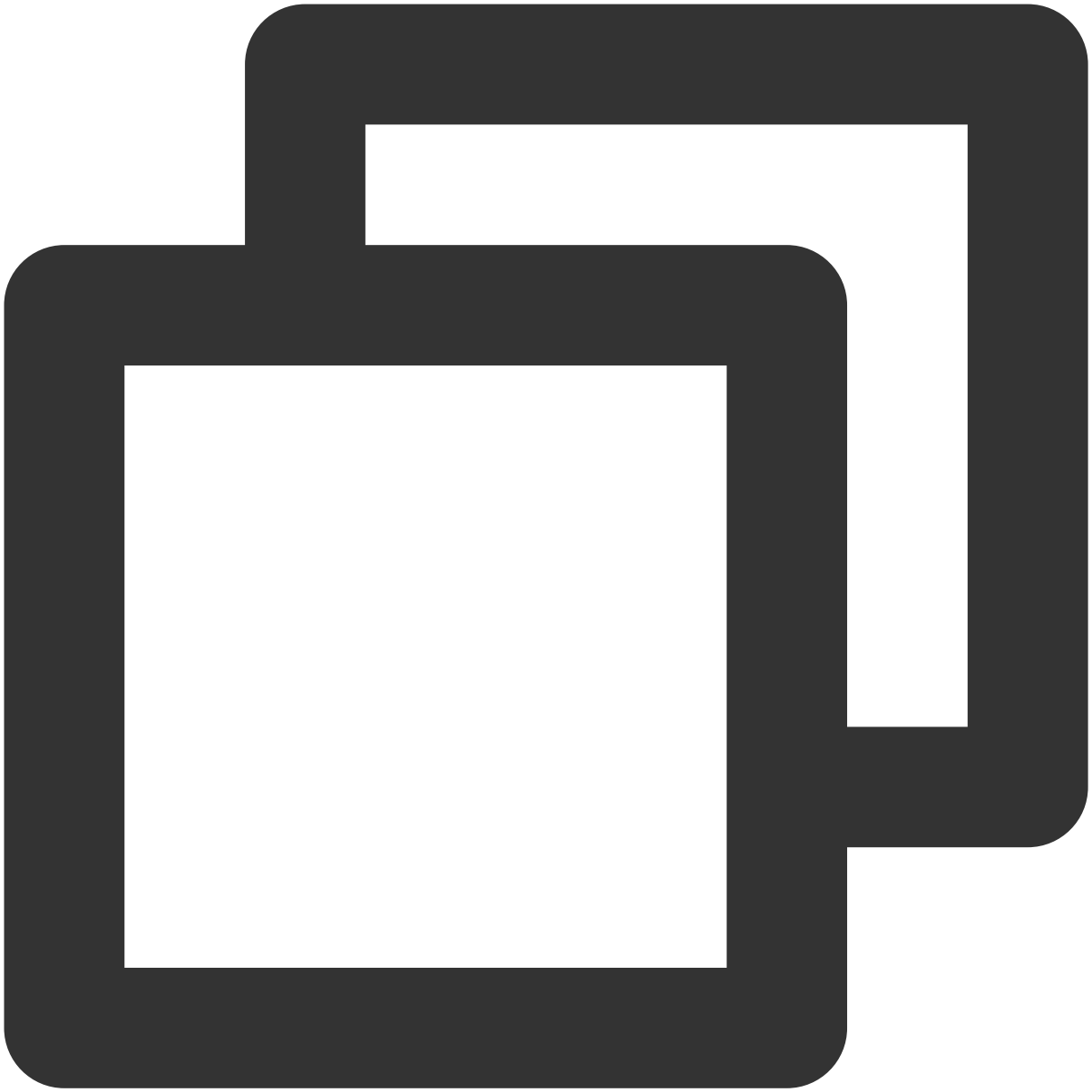
步骤4：在 Native 的 TRTC 自定义渲染逻辑中完成美颜处理

Android

iOS

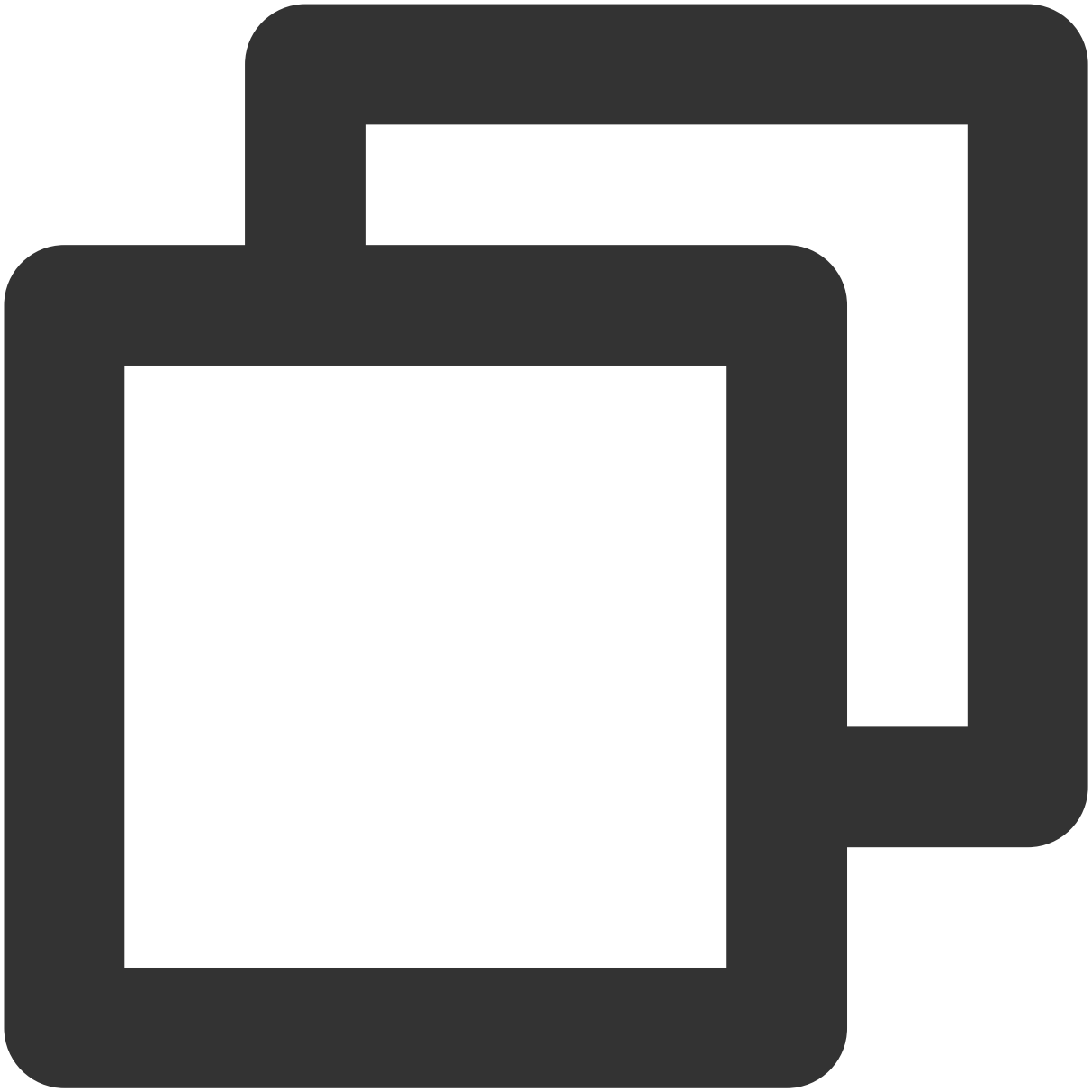
Android 在接入美颜过程中需要先依赖 **LiteAVSDK_Professional，在Android工程的 app/build.gradle中添加一下依赖：**

```
dependencies{
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
```



```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());

}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
```

```
}

class VideoFrameListerer implements TRTCCloudListener.TRTCVideoFrameListener {
    private XXXBeautyModel  mBeautyModel = XXXBeautyModel.sharedInstance();

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
       trtcVideoFrame1.texture.textureId = XmagicApiManager.getInstance()
        .process(trtcVideoFrame.texture.textureId, trtcVideoFrame.width, trtcVideoF
        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

    @Override
    public void onGLContextDestory() {
    }
}
```



```
import TXLiteAVSDK_Professional
import tencent_effect_flutter

let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pix
}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Tex
```

```
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame)
        dstFrame.textureId = GLuint(XmagicApiManager.shareSingleton().getTextureId(
        return 0
    }
}


public class ConvertBeautyFrame: NSObject {
    public static func convertToTRTCPixelFormat(beautyPixelFormat: ITXCustomBeautyP
        switch beautyPixelFormat {
        case .Unknown:
            return ._Unknown
        case .I420:
            return ._I420
        case .Texture2D:
            return ._Texture_2D
        case .BGRA:
            return ._32BGRA
        case .NV12:
            return ._NV12
        }
    }

    public static func convertTRTCVideoFrame(trtcVideoFrame: TRTCVideoFrame) -> ITX
        let beautyVideoFrame = ITXCustomBeautyVideoFrame()
        beautyVideoFrame.data = trtcVideoFrame.data
        beautyVideoFrame.pixelBuffer = trtcVideoFrame.pixelBuffer
        beautyVideoFrame.width = UInt(trtcVideoFrame.width)
        beautyVideoFrame.height = UInt(trtcVideoFrame.height)
        beautyVideoFrame.textureId = trtcVideoFrame.textureId
        switch trtcVideoFrame.rotation {
        case ._0:
            beautyVideoFrame.rotation = .rotation_0
        case ._90:
            beautyVideoFrame.rotation = .rotation_90
        case ._180:
            beautyVideoFrame.rotation = .rotation_180
        case ._270:
            beautyVideoFrame.rotation = .rotation_270
        default:
            beautyVideoFrame.rotation = .rotation_0
        }
        switch trtcVideoFrame.pixelFormat {
        case ._Unknown:
```

```
            beautyVideoFrame.pixelFormat = .Unknown
        case ._I420:
            beautyVideoFrame.pixelFormat = .I420
        case ._Texture_2D:
            beautyVideoFrame.pixelFormat = .Texture2D
        case ._32BGRA:
            beautyVideoFrame.pixelFormat = .BGRA
        case ._NV12:
            beautyVideoFrame.pixelFormat = .NV12
        default:
            beautyVideoFrame.pixelFormat = .Unknown
        }
        beautyVideoFrame.bufferType = ITXCustomBeautyBufferType(rawValue: trtcVideo
        beautyVideoFrame.timestamp = trtcVideoFrame.timestamp
        return beautyVideoFrame
    }
}
```

步骤5：开启美颜并设置美颜参数

完成上述配置后，可以通过 `enableTUICallKitCustomBeauty()/disableTUICallKitCustomBeauty()` 开启/关闭美颜。可以通过 腾讯特效美颜Flutter接口 设置美颜参数。

自定义铃声

Android

最近更新时间：2024-04-03 17:23:11

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

设置应用铃声

设置应用铃声有两种方式：替换铃声音频、调用设置铃声接口。

1、替换音频文件

如果您通过源码依赖 TUICallKit 组件，您可以替换 `res\\raw` 文件夹下的三个音频文件来达到替换铃声的目的：

文件名	用途
phone_dialing.mp3	发起呼叫时的铃音
phone_hangup.mp3	被挂断的铃音
phone_ringing.mp3	接到呼叫时的铃音

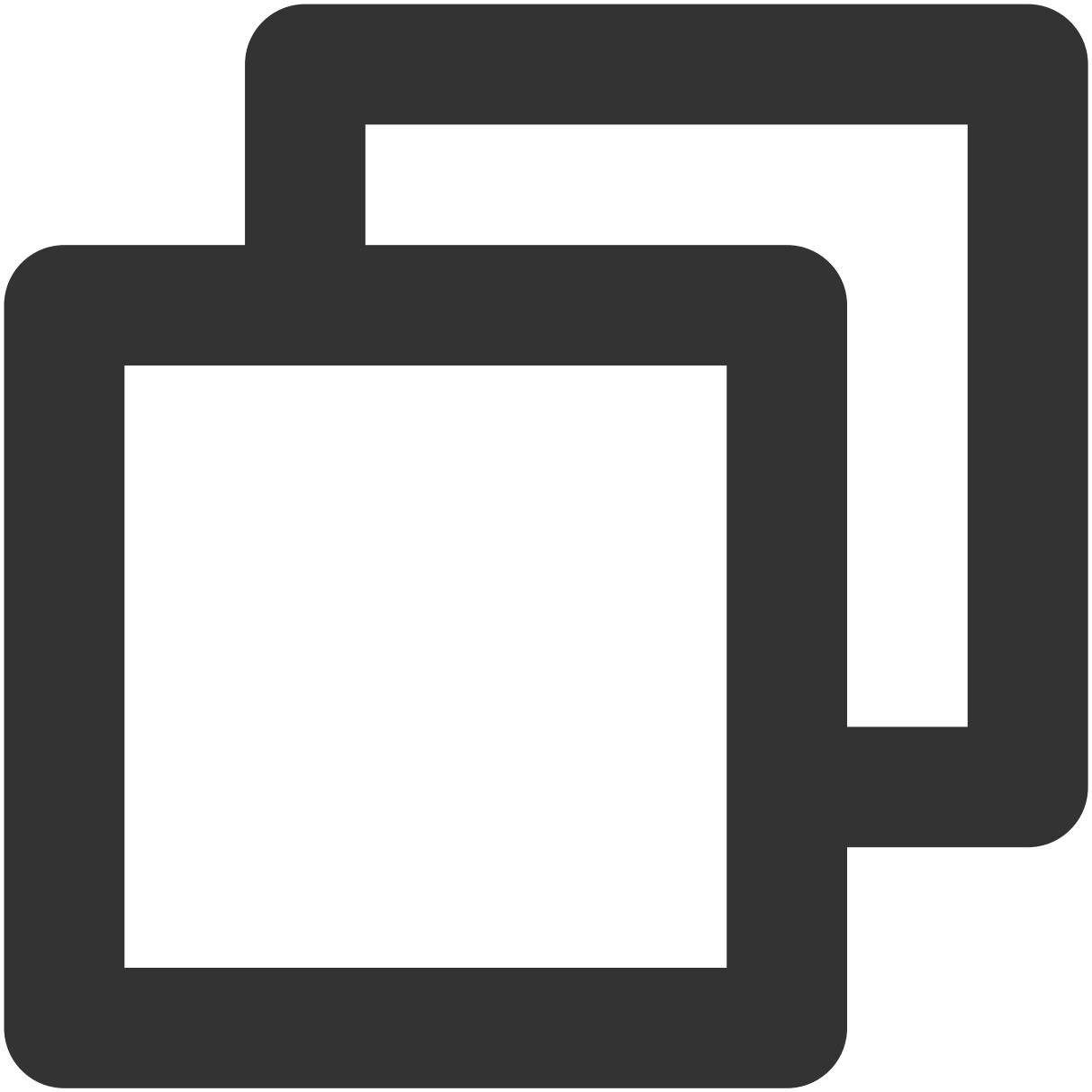
2、设置铃声接口

您也可以通过 setCallingBell 接口设置来电铃声。

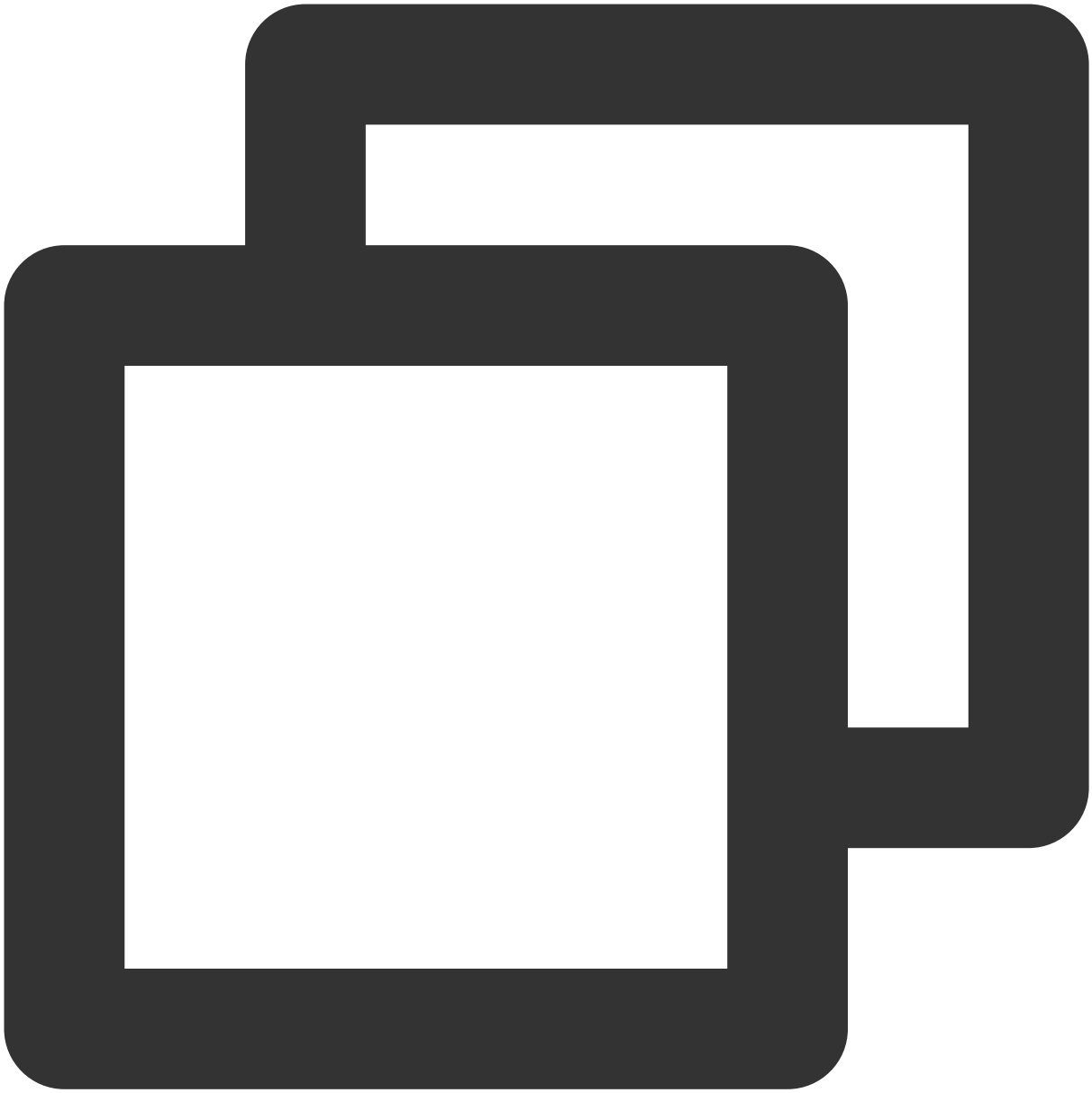
Kotlin

Java

版权所有：腾讯云计算（北京）有限责任公司



```
TUICallKit.createInstance(context).setCallingBell(filePath)
```



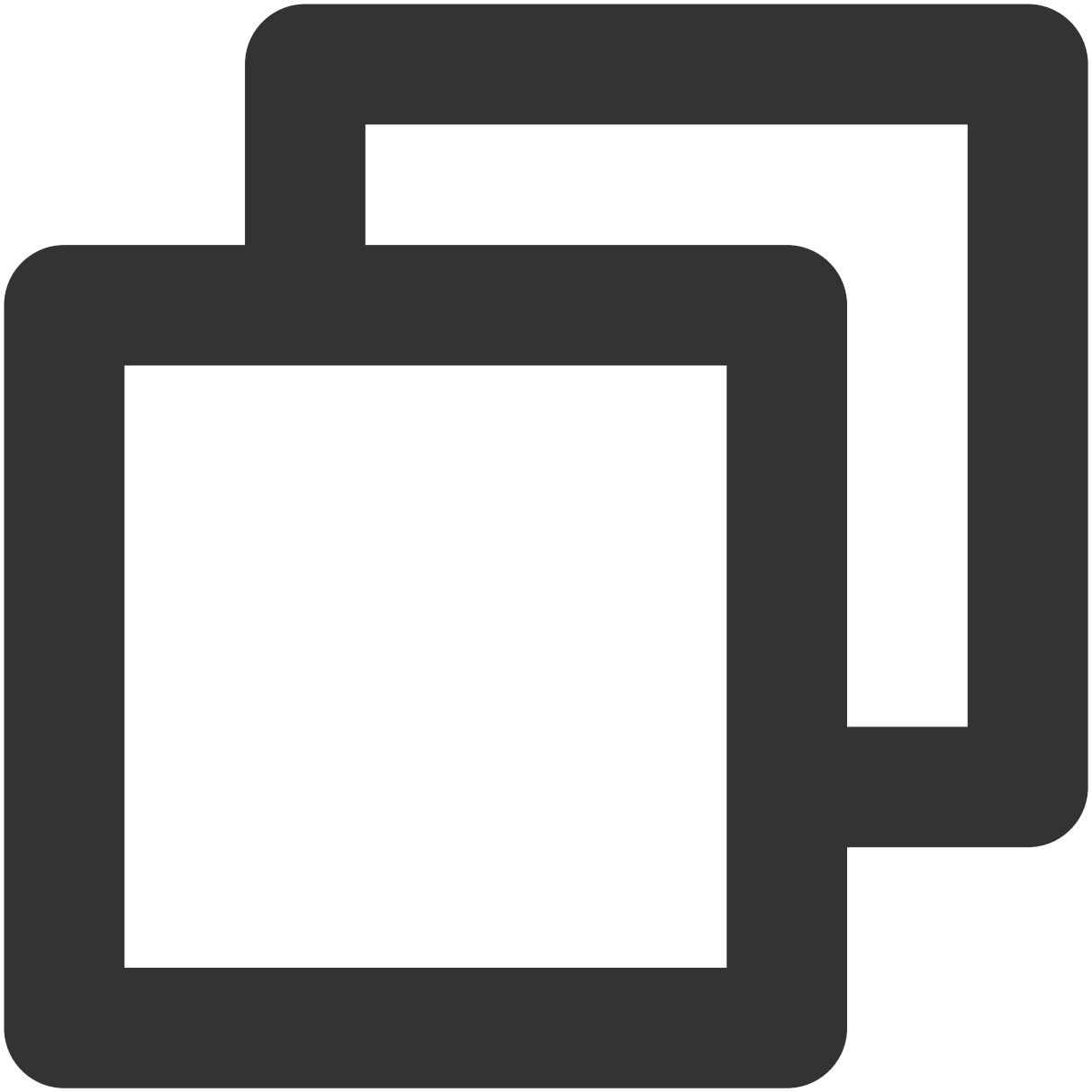
```
TUICallKit.createInstance(context).setCallingBell(filePath);
```

3. 设置静音模式

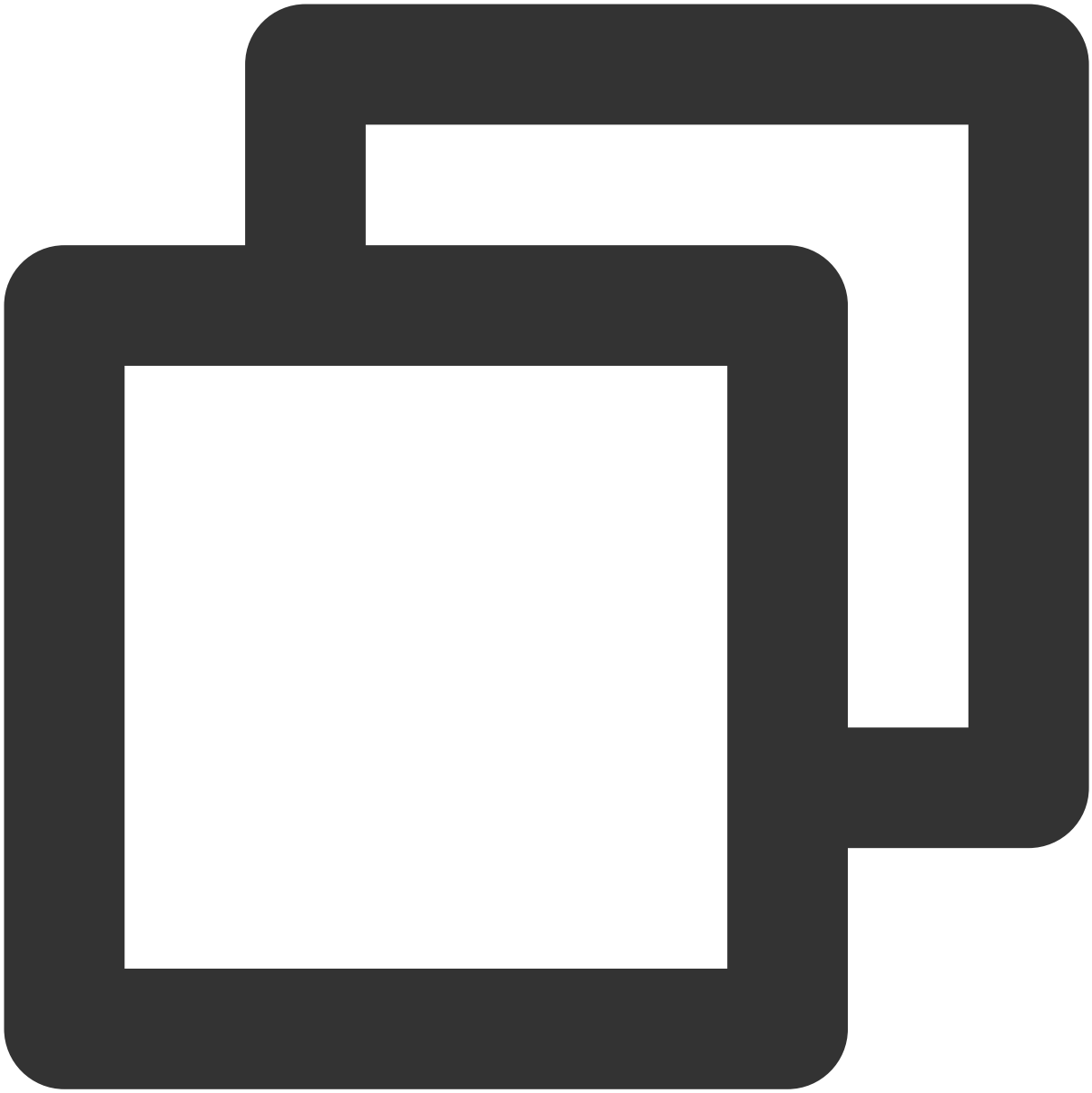
如果您不需要响铃，您可以通过 enableMuteMode 设置静音模式。

Kotlin

Java



```
TUICallKit.createInstance(context).enableMuteMode(true)
```



```
TUICallKit.createInstance(context).enableMuteMode(true);
```

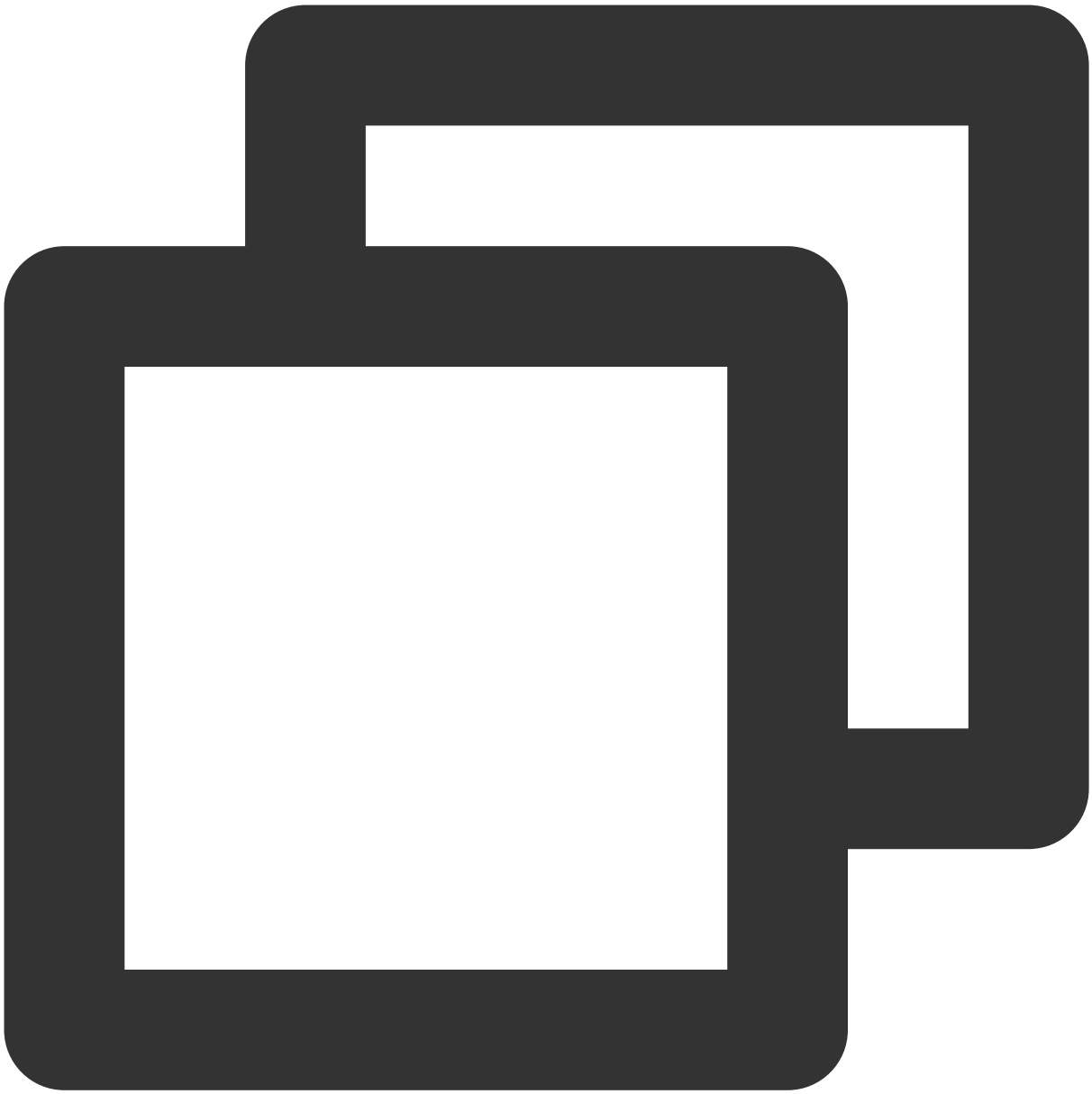
设置离线推送铃音

离线通知铃声仅支持以下厂商自定义：**华为、小米、FCM 和 APNs**，其他厂商 OPPO、VIVO、荣耀等暂不支持。

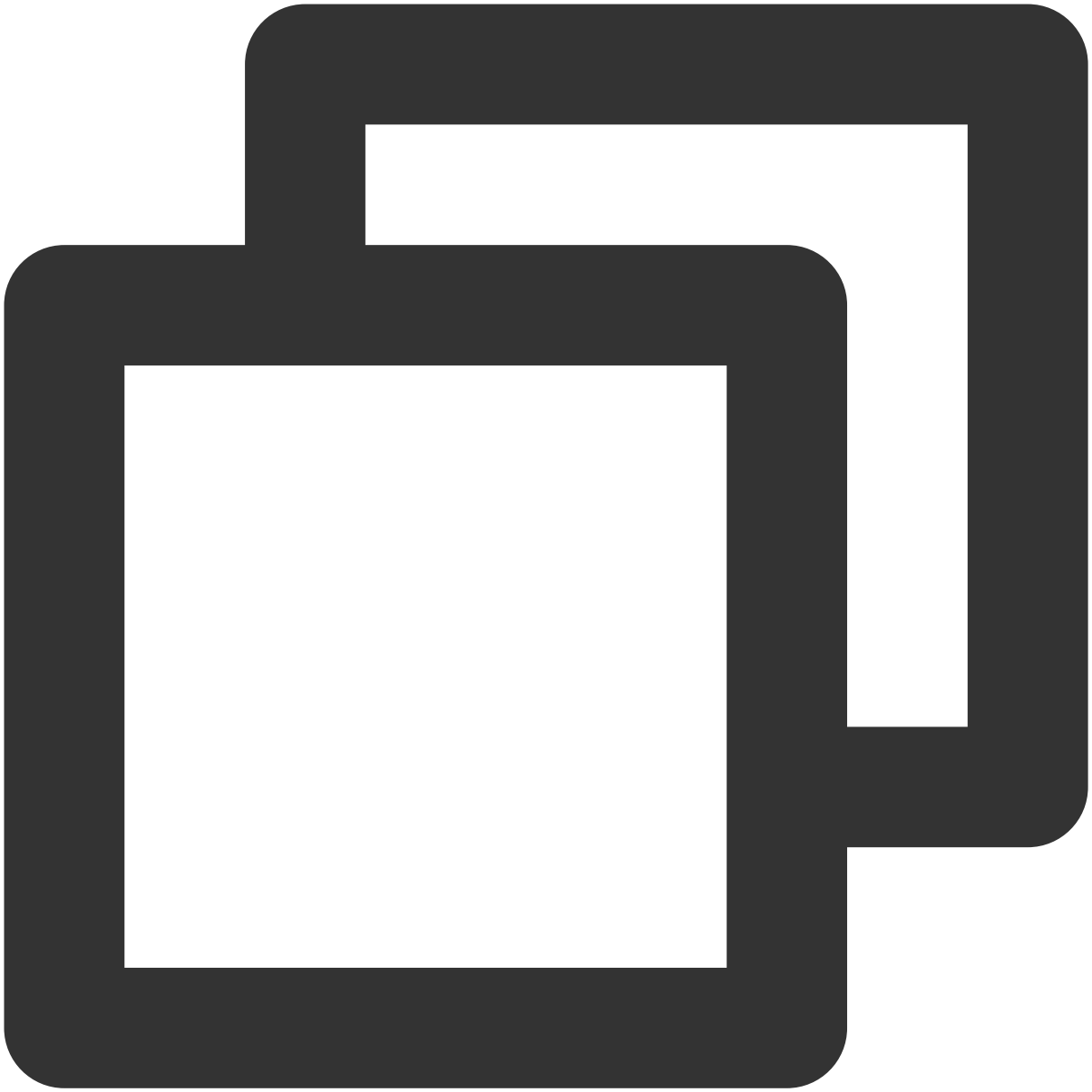
TUICallKit 引入了 TUIOfflinePushPush 组件，需在应用启动时，调用以下方法开启离线通知自定义铃音能力。

Kotlin

Java



```
class DemoApplication : Application() {
    override fun onCreate() {
        TUIOfflinePushConfig.getInstance().isAndroidPrivateRing = true
    }
}
```



```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        TUIOfflinePushConfig.getInstance().setAndroidPrivateRing(true);
    }
}
```

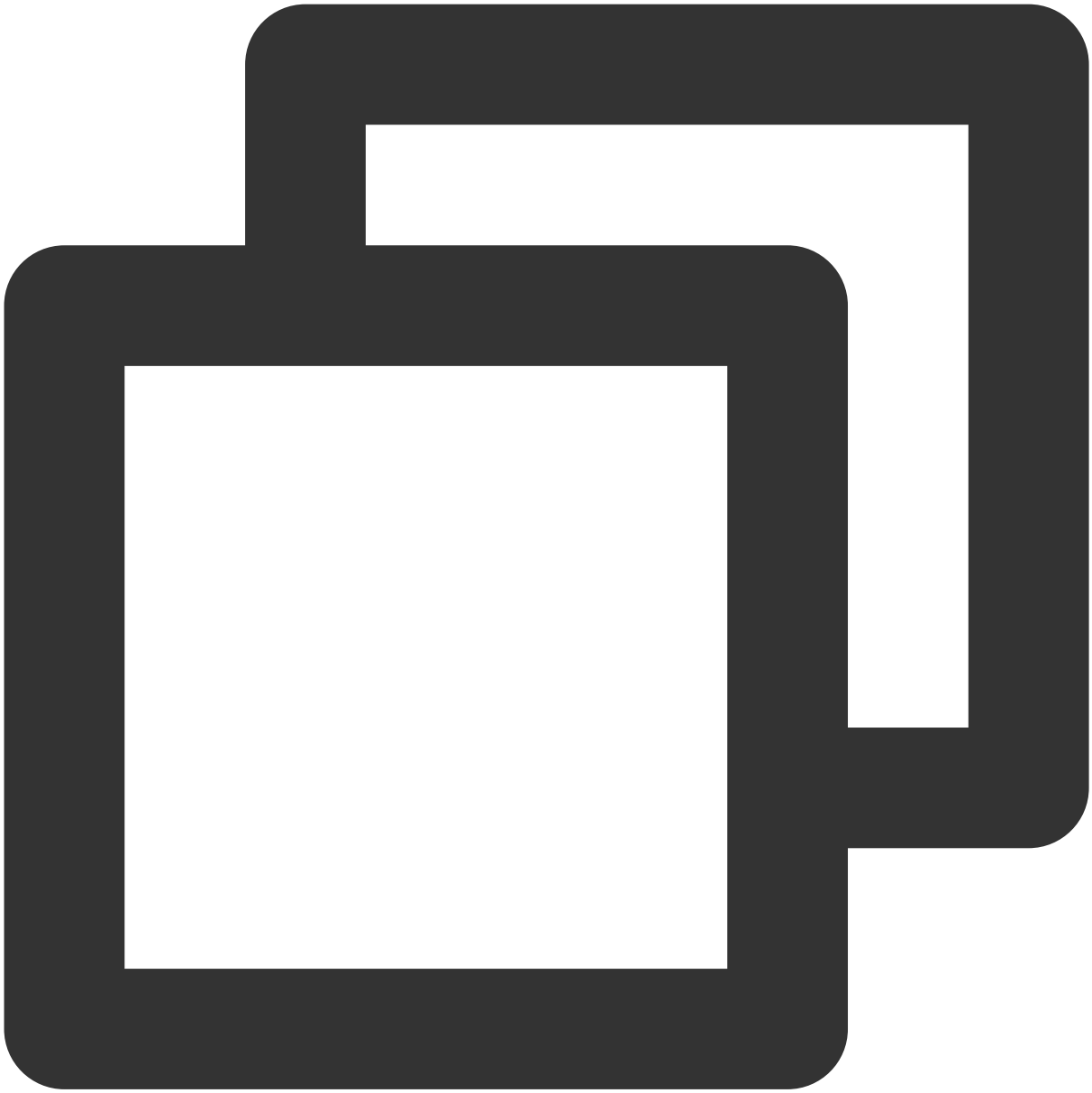
1、华为 & APNs

调用接口 setAndroidSound() 和 setIOSSound()。

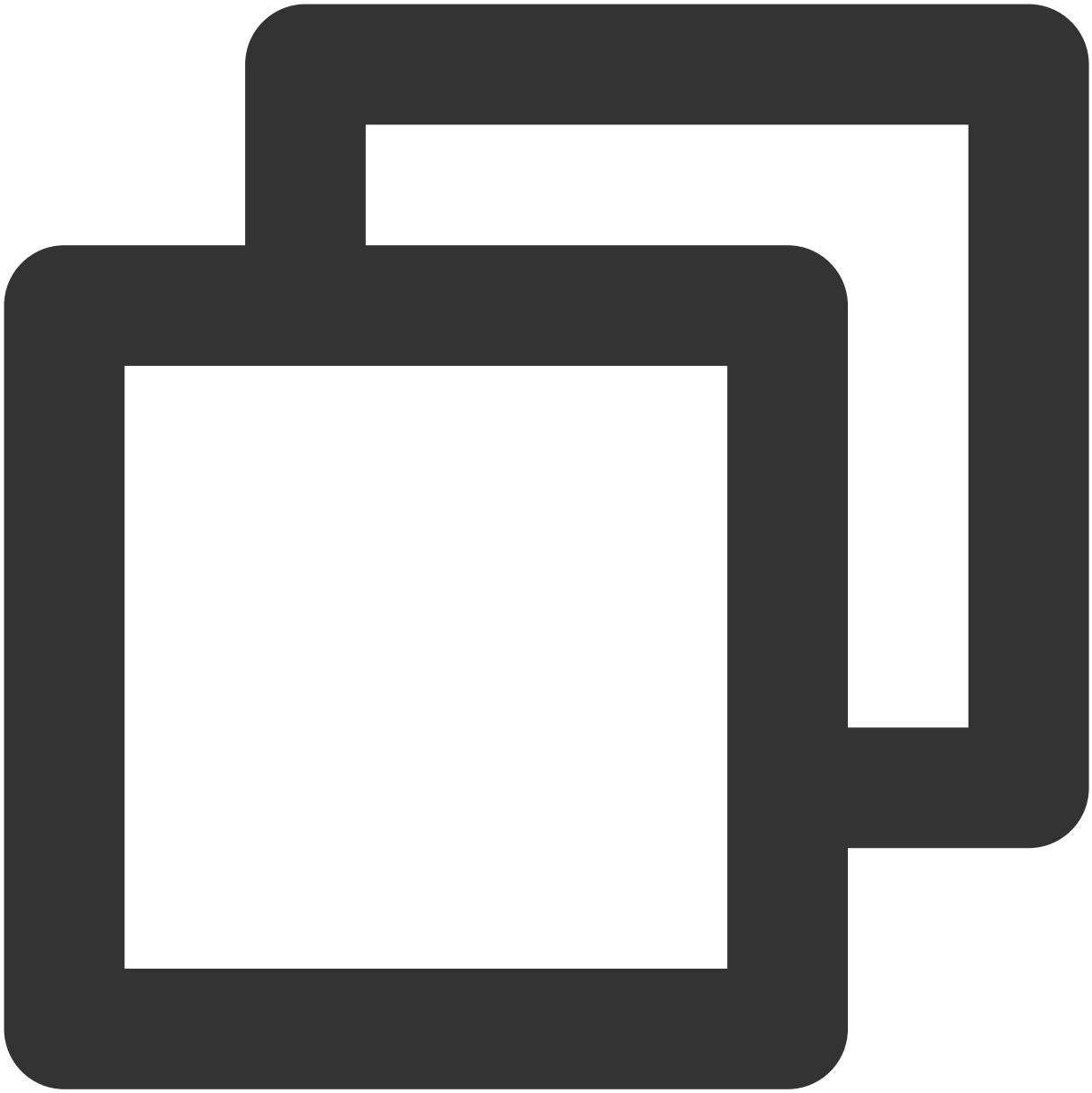
定制铃音资源文件，将铃音文件添加到本地 Android Studio 工程 res/raw 目录下。

Kotlin

Java



```
val pushInfo = OfflinePushInfo()
pushInfo.iosSound = "铃声名称.mp3"
pushInfo.androidSound = "铃声名称"
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
```

注意

IMSDK 6.1.2155 及以上版本支持。

华为推送消息的铃声为首次创建通知渠道时的铃声，之后再次设置其他铃声无效，若想更新自定义通知铃音需卸载重装，详见：华为自定义铃声。

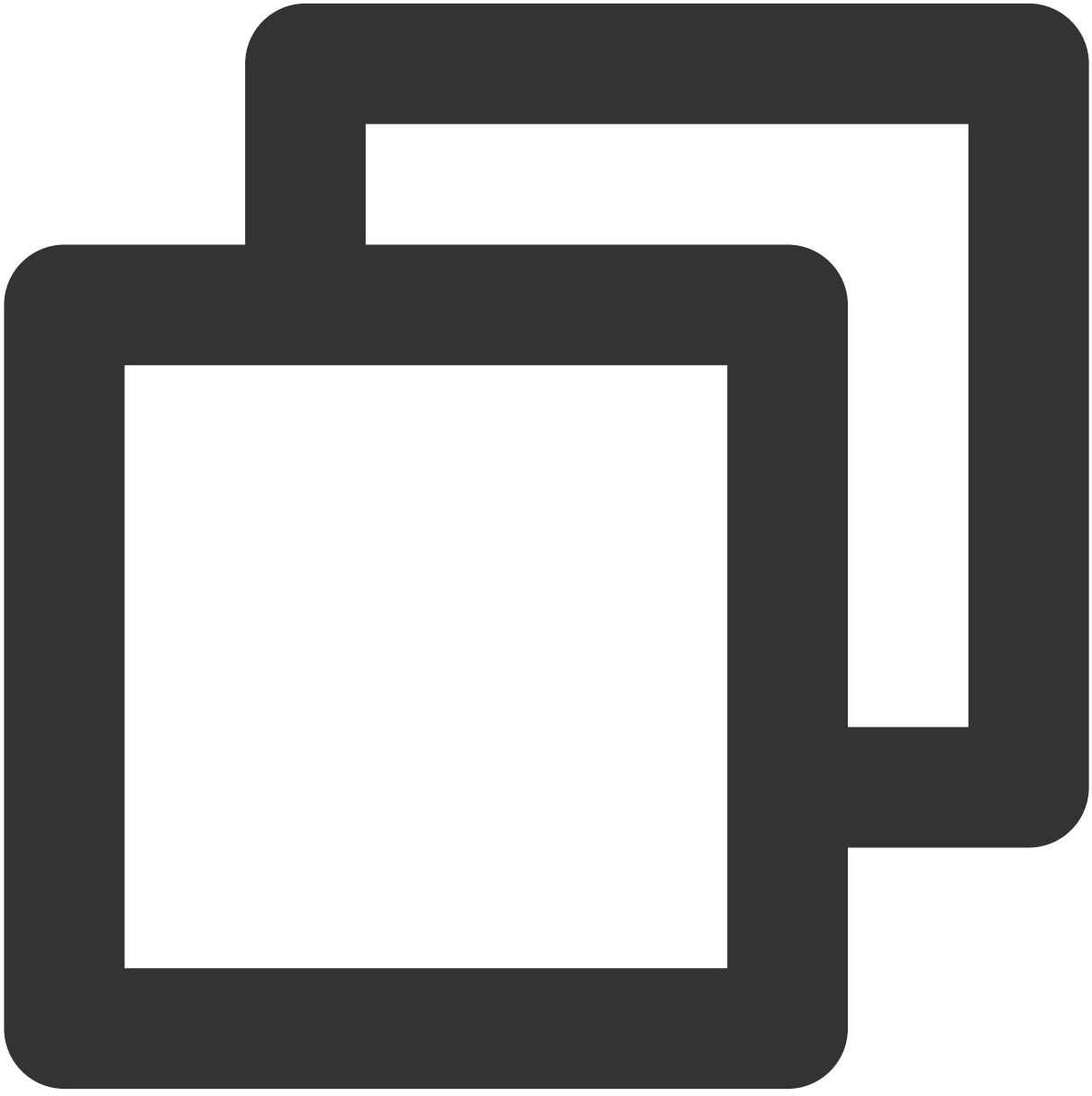
2、小米

（1）Android 8.0 之前，调用setAndroidSound() 和 setIOSSound()。定制铃音资源文件，将铃音文件添加到工程 res/raw 目录下（参考上述华为调用）。

（2）Android 8.0 之后，还需要登录小米厂商控制台并 创建 channel 并做好配置，其中铃音文件需要添加到本地 Android Studio 工程的 res/raw 目录下。

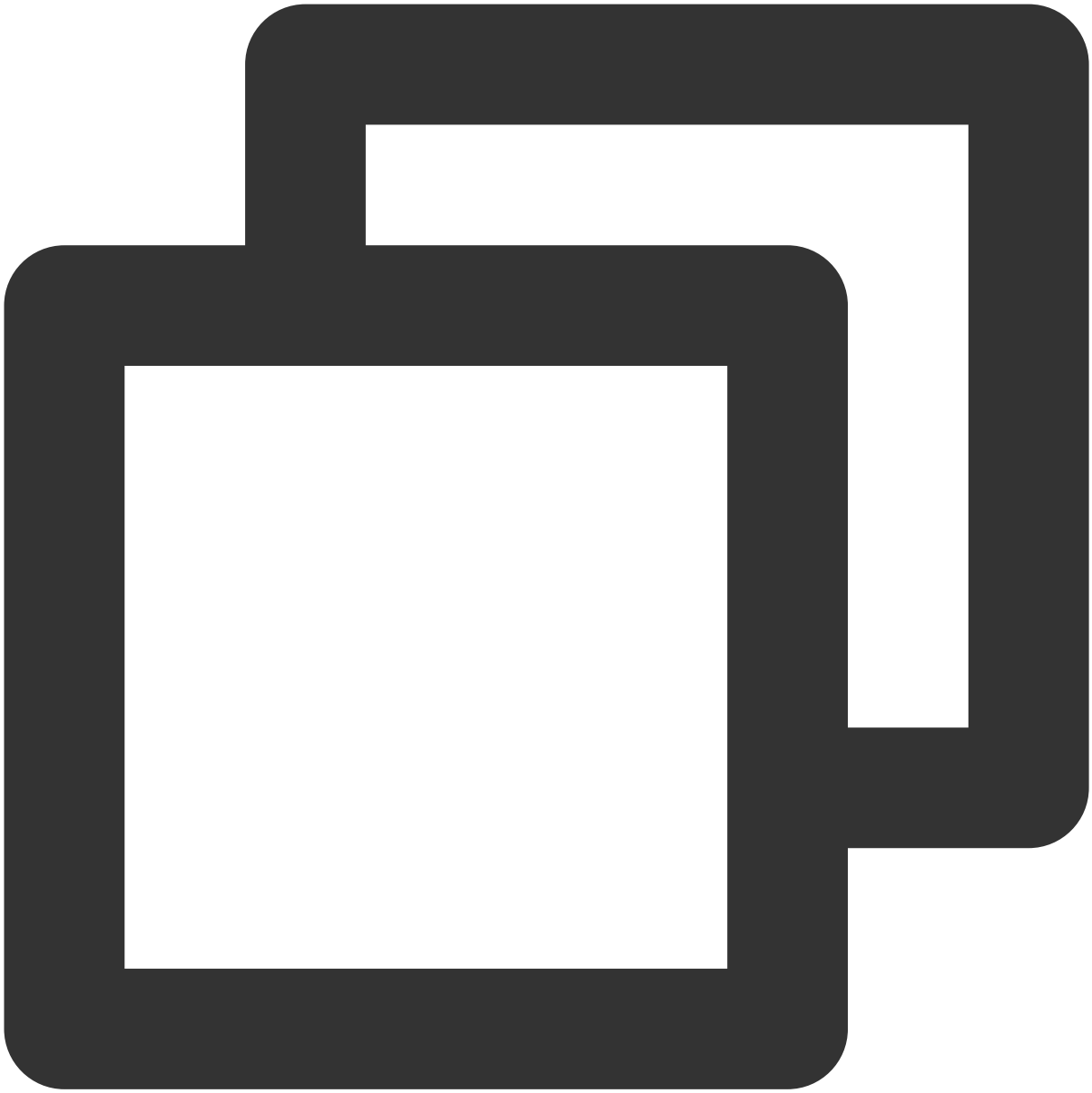
Kotlin

Java



```
val pushInfo = OfflinePushInfo()
pushInfo.iosSound = "铃声名称.mp3"
pushInfo.androidSound = "铃声名称"
```

```
pushInfo.androidXiaoMiChannelID = "厂商申请的channelID"
```



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
pushInfo.setAndroidXiaoMiChannelID("厂商申请的channelID");
```



注意

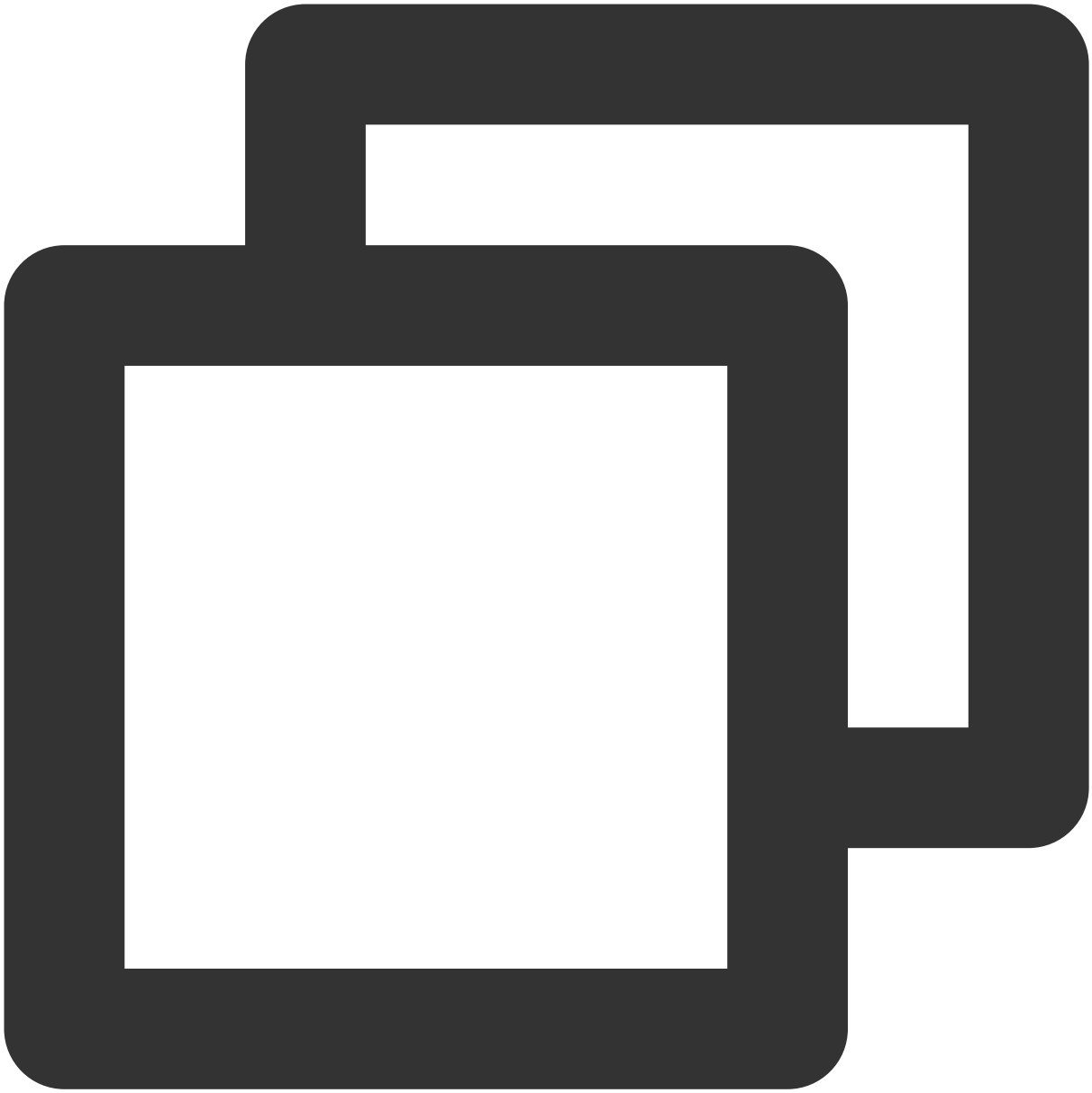
Android 8.0之前的方式，IMSDK 6.1.2155 及以上版本支持。

Android 8.0之后的方式，IMSDK 7.0.3754 及以上版本支持。

3、FCM

（1）Android 8.0 之前，调用 setAndroidSound() 和 setIOSSound()。定制铃音资源文件，将铃音文件添加到工程 res/raw 目录下（参考上述华为调用）。

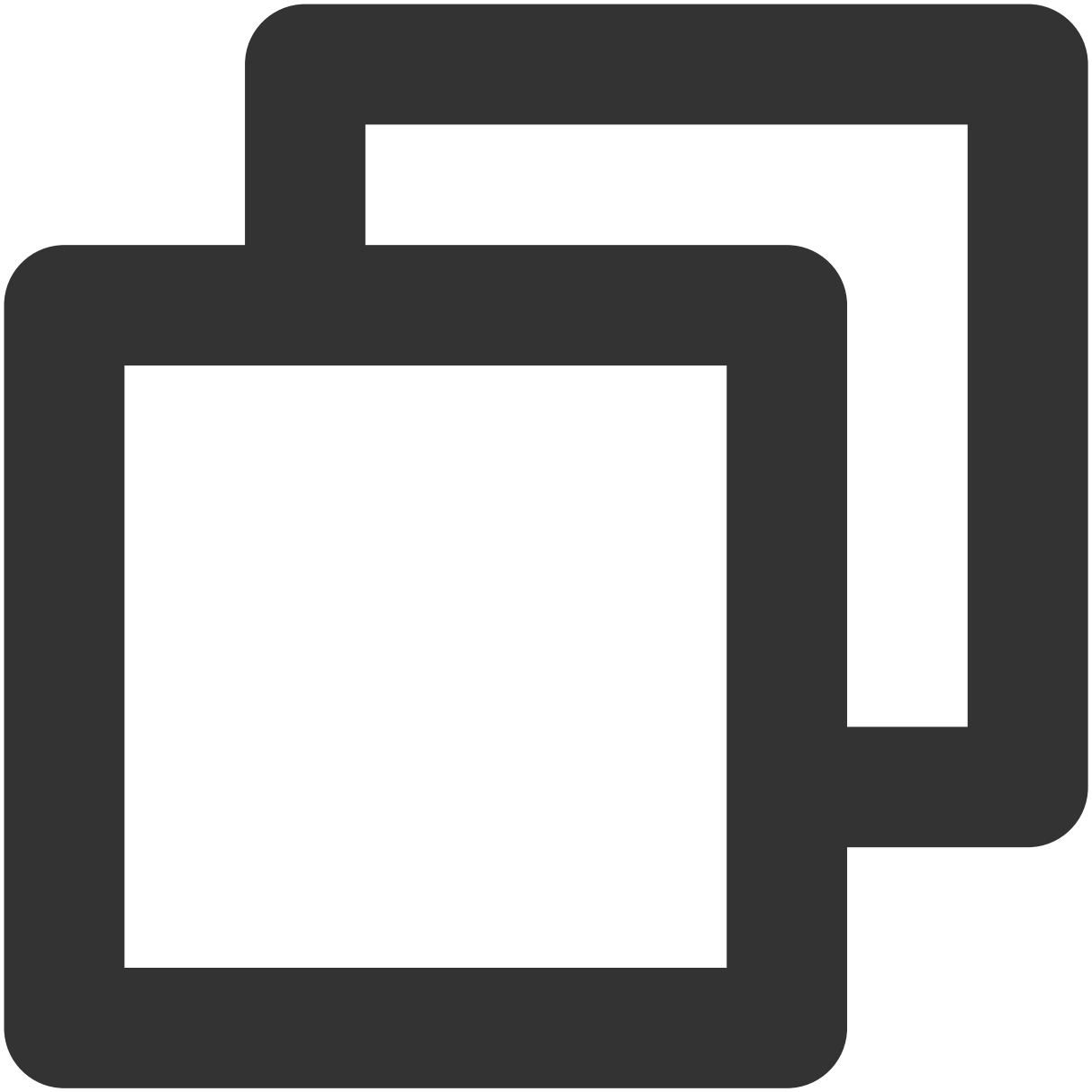
（2）Android 8.0之后，FCM 需要配置 channelID 以及铃音资源，TUICallKit 引入的 tuiofflinepush 组件已经处理了自定义铃声的播放。铃音文件需要添加到您本地 Android Studio 工程的 res/raw 目录下，并指定铃音名称和 channel ID 的名称，详见 PrivateConstants。



```
public class PrivateConstants {
    // FCM 通道指定 channel ID
    public static String fcmPushChannelId = "FCM ChannelID";
    // FCM 需指定通道的推送铃音名称，与AndroidStudio 工程里 res/raw 目录中的铃音文件名一致，不
    public static String fcmPushChannelSoundName = "铃声名称";
}
```

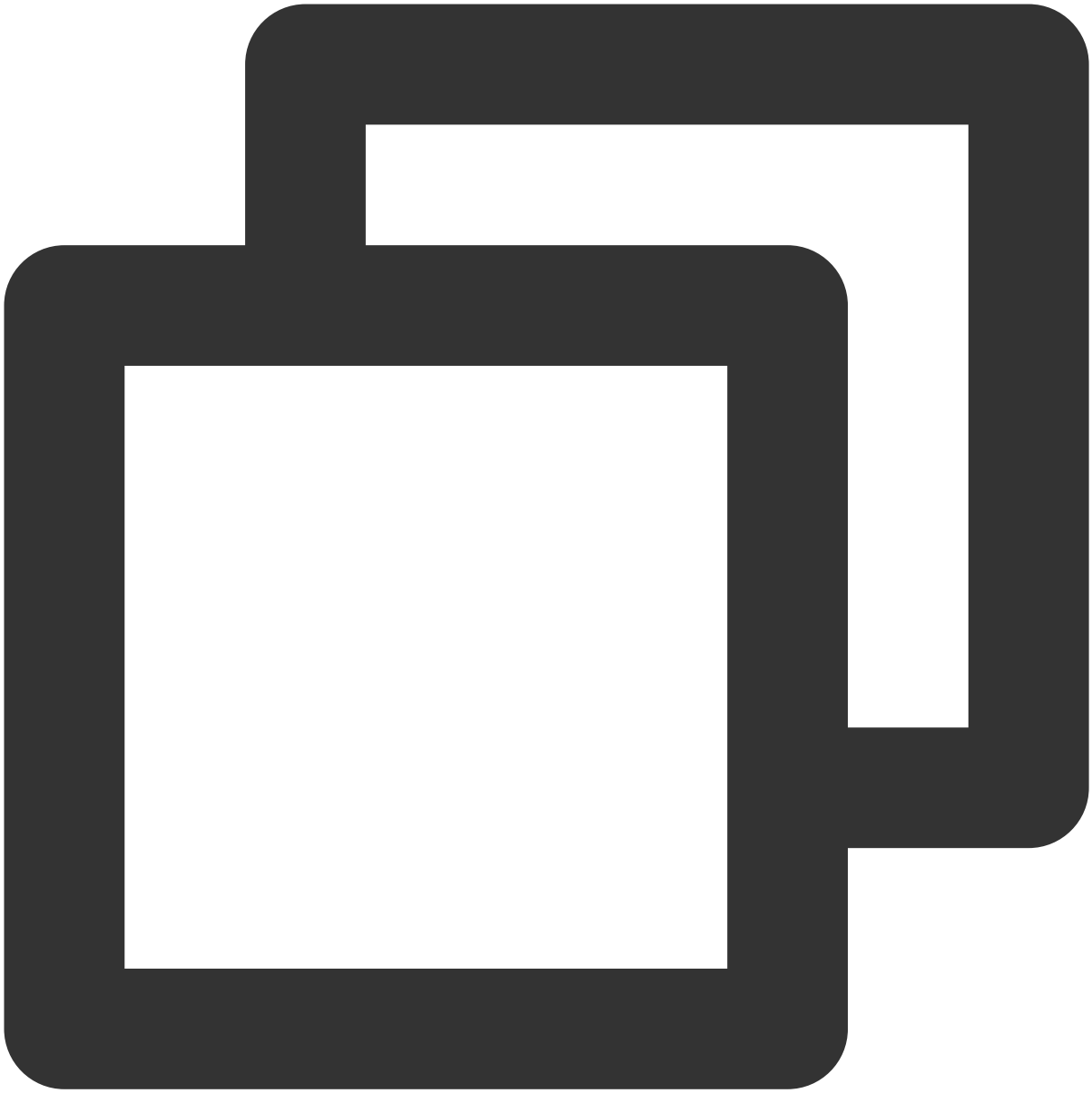
Kotlin

Java



```
val pushInfo = OfflinePushInfo()
pushInfo.iosSound = "铃声名称.mp3"
pushInfo.androidSound = "铃声名称"
pushInfo.androidFCMChannelID = "厂商申请的channelID"
```

版权所有：腾讯云计算（北京）有限责任公司



```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setIOSSound("铃声名称.mp3");
pushInfo.setAndroidSound("铃声名称");
pushInfo.setAndroidFCMChannelID("厂商申请的channelID");
```

注意

Android 8.0之前的方式，IMSDK 6.1.2155 及以上版本支持。

Android 8.0之后的方式，IMSDK 7.0.3754 及以上版本支持。

FCM 自定义铃声或者设置 channel id 仅支持证书模式。


添加Android证书
推送平台
小米
华为
Google
魅族
vivo
OPPO
荣耀
添加方式
填写服务器密钥
上传证书
上传证书
选择文件
如何生成谷歌（FCM）证书?
ChannelID
请输入ChannelID
确定

iOS

最近更新时间：2024-04-03 17:23:11

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

设置应用铃声

设置应用铃声有两种方式：替换铃声音频、调用设置铃声接口。

1、替换音频文件

如果您通过源码依赖 TUICallKit 组件，您可以替换 `Resources\\AudioFile` 文件夹下的三个音频文件来达到替换铃声的目的：

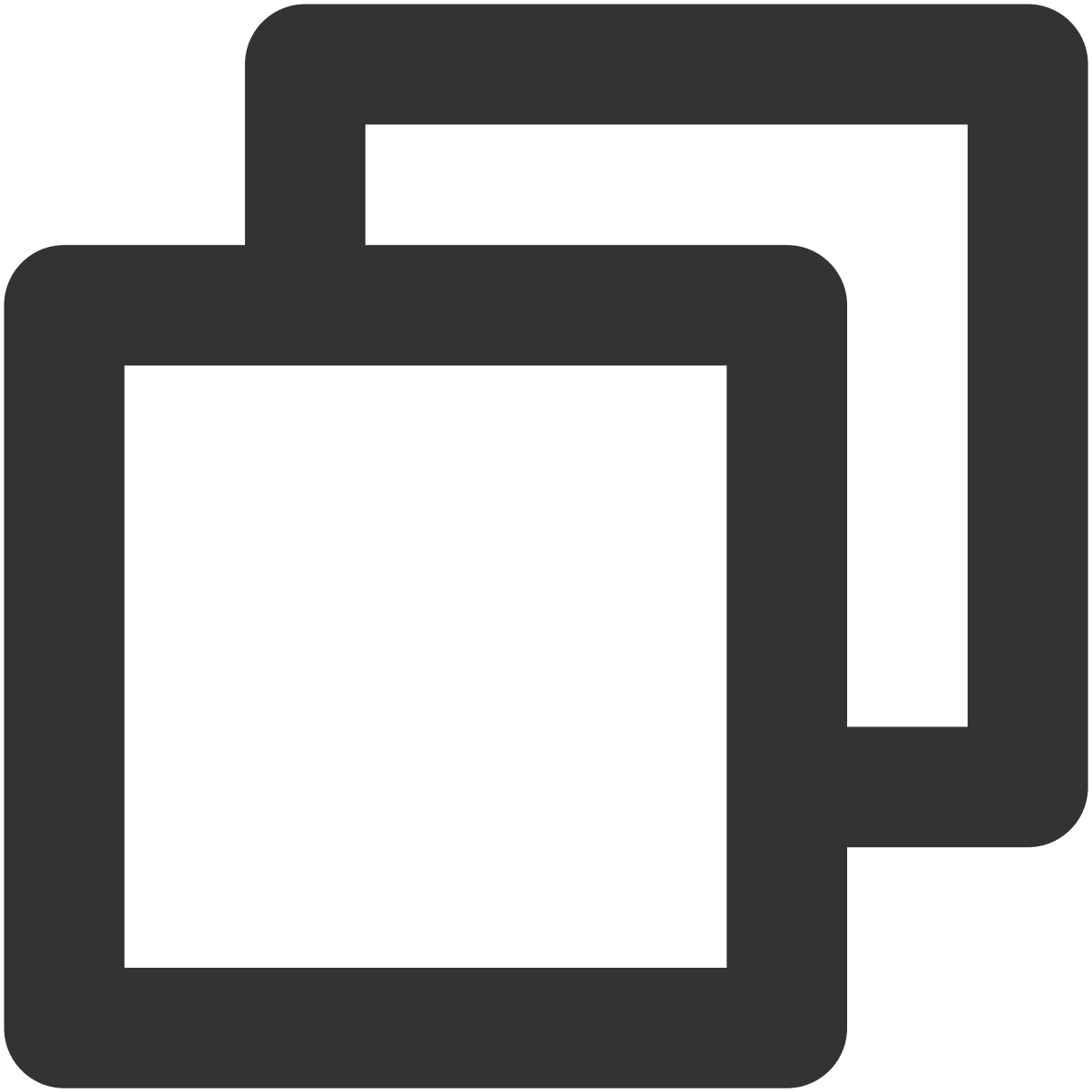
文件名	用途
phone_dialing.mp3	发起呼叫时的铃音
phone_hangup.mp3	被挂断的铃音
phone_ringing.mp3	接到呼叫时的铃音

2、设置铃声接口

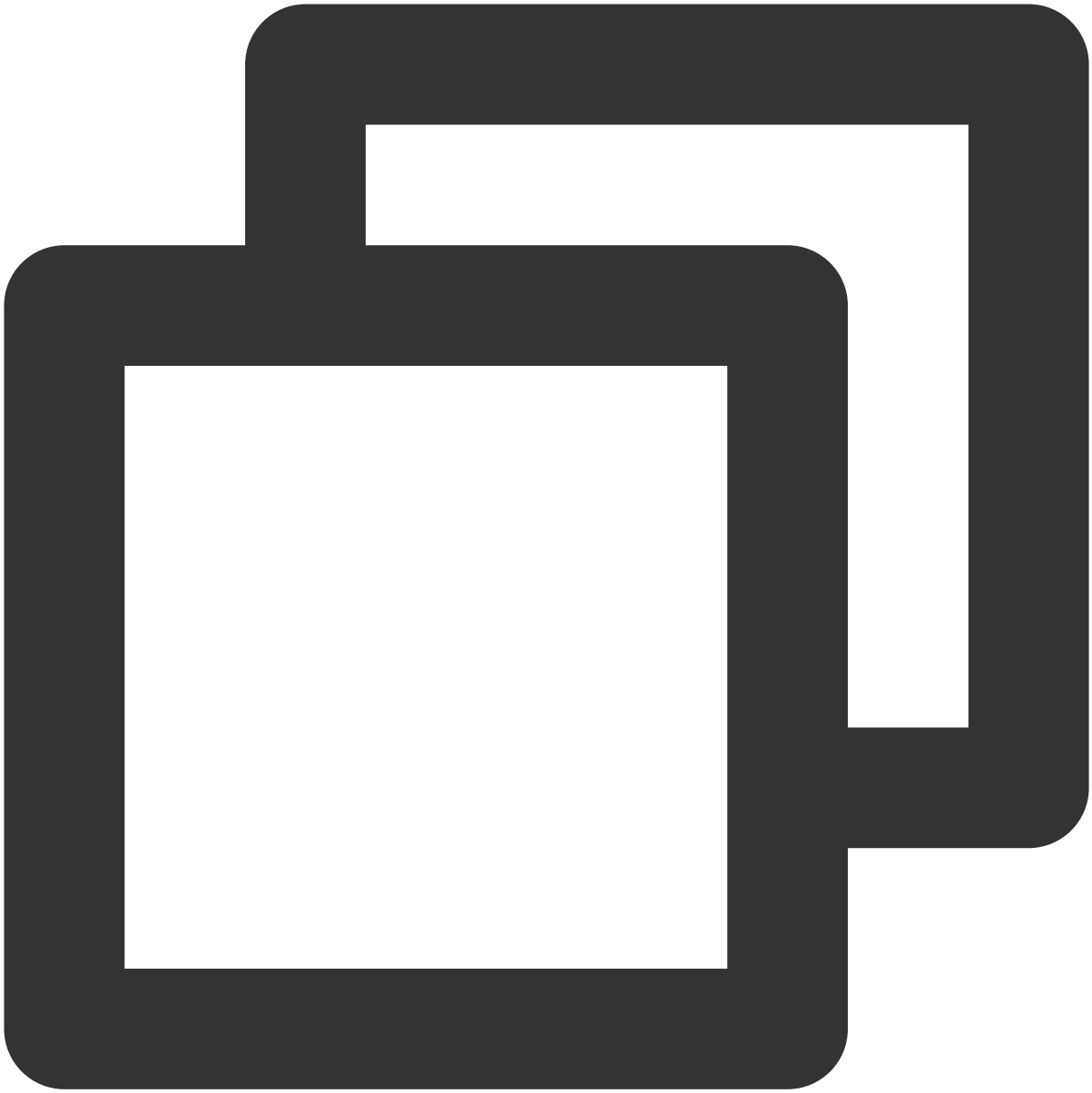
您也可以通过 setCallingBell 接口设置来电铃声。

Swift

Objective-C



```
TUICallKit.createInstance().setCallingBell(filePath: " ")
```



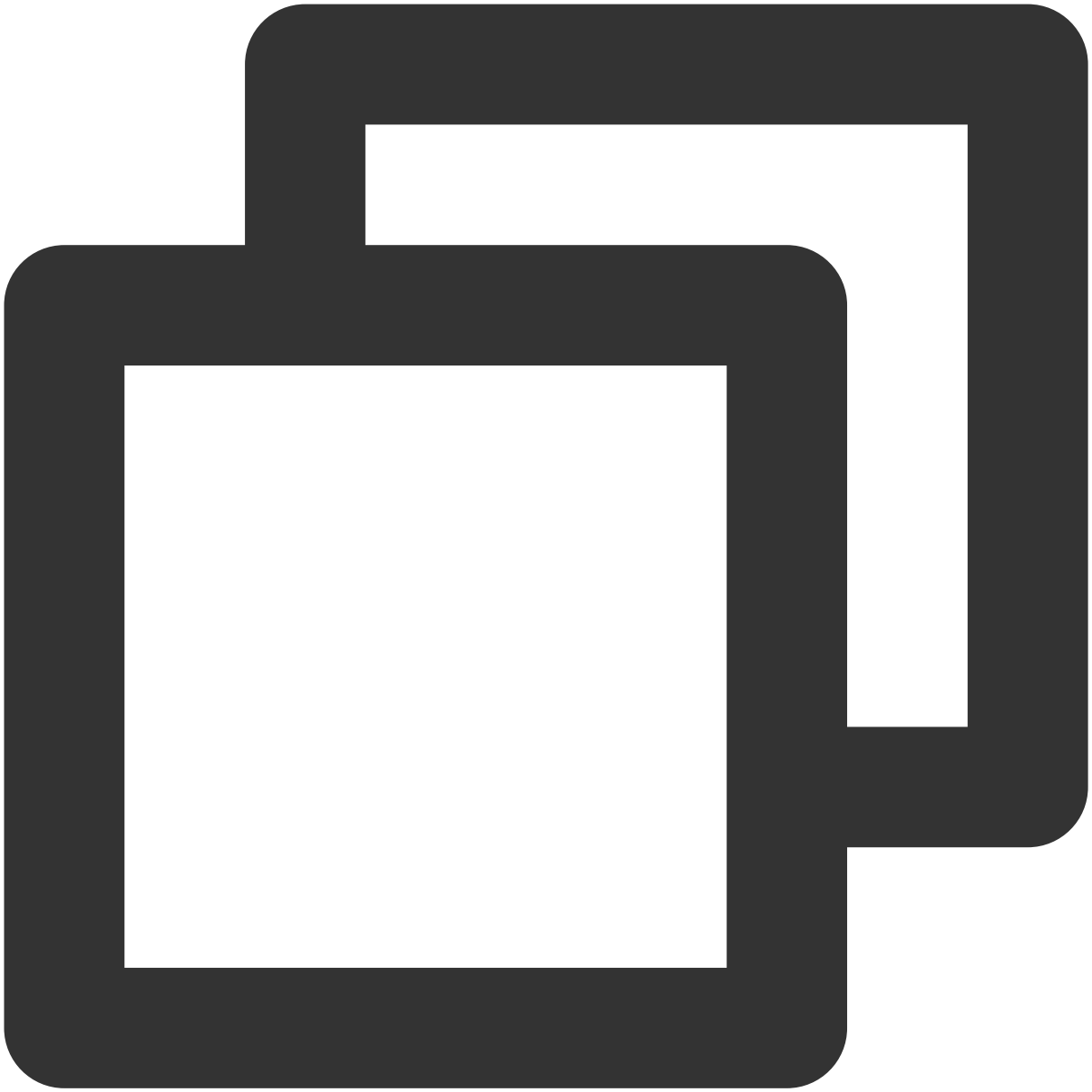
```
[[TUICallKit createInstance] setCallingBell:@" "];
```

3、设置静音模式

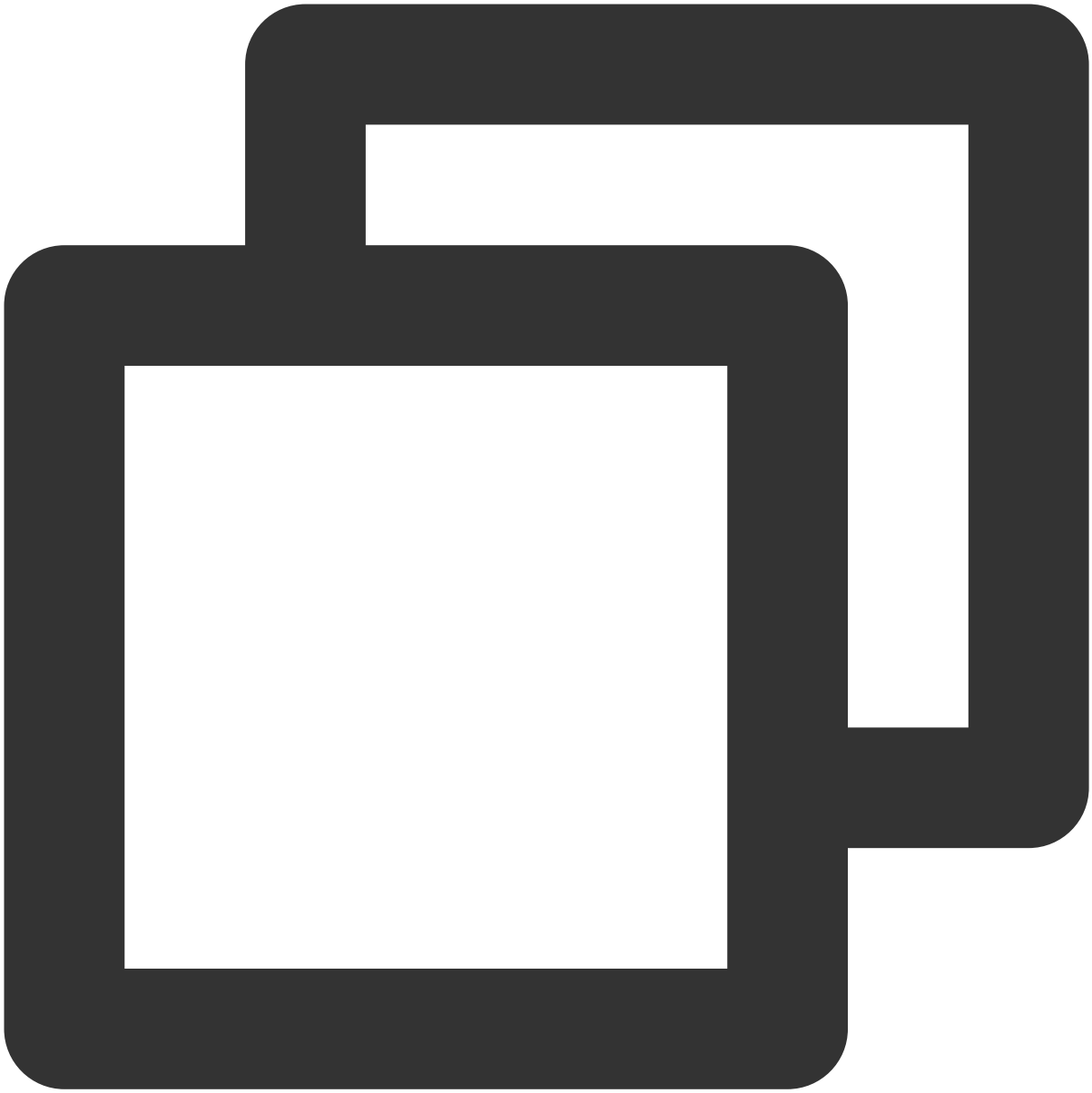
如果您不需要响铃，您可以通过 enableMuteMode 设置静音模式。

Swift

Objective-C



```
TUICallKit.createInstance().setCallingBell(enable: true)
```



```
[[TUICallKit createInstance] enableMuteMode: YES];
```

设置离线推送铃音

Voip 推送不支持自定义推送铃声。 APNs 推送可以在调用 call 接口拨打电话时设置 params 的 offlinePushInfo 中的 iOSSound 字段， iOSSound 传语音文件名。

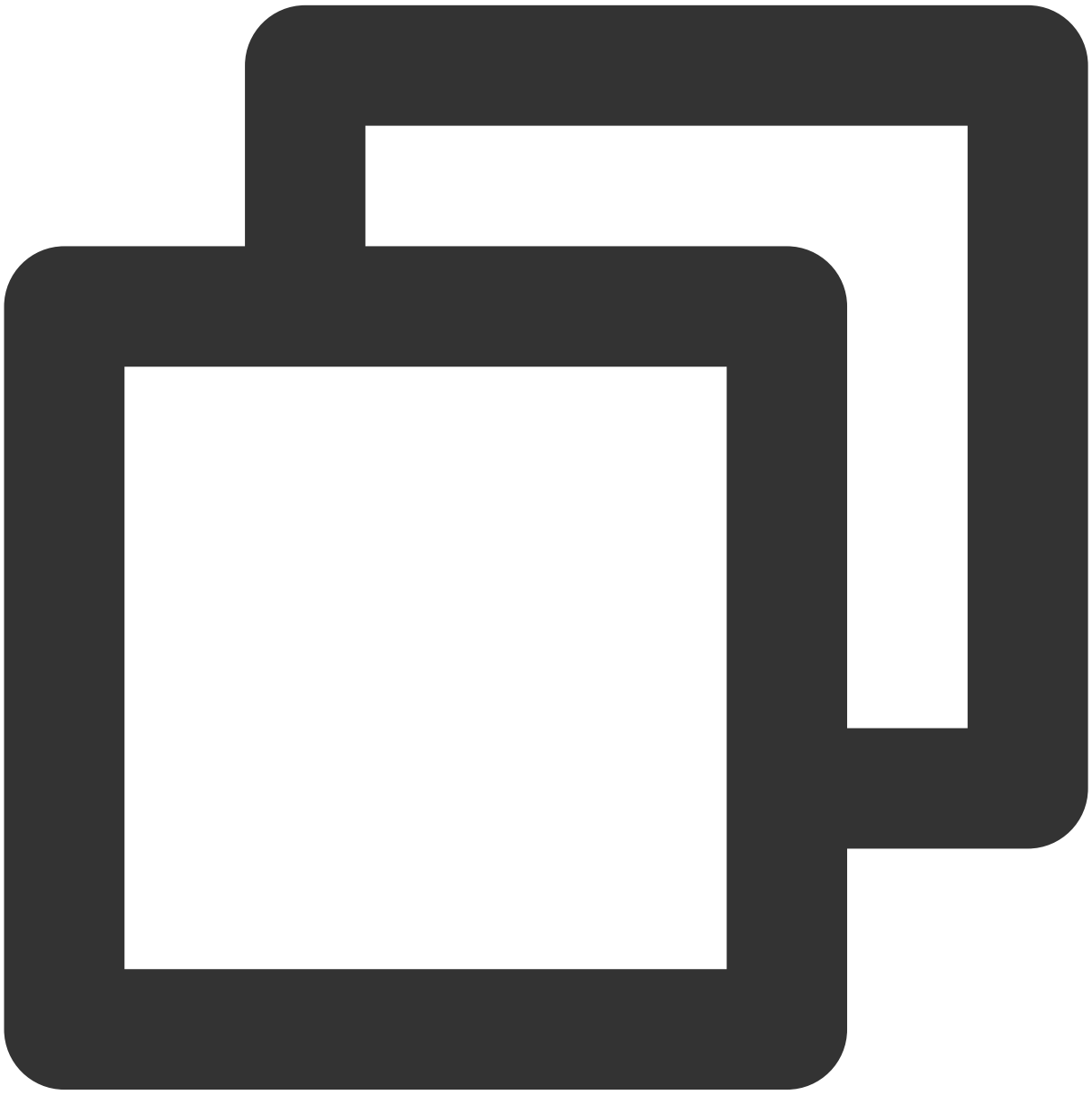
注意：

离线推送声音设置（仅对 iOS 生效），如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀名）设置给 iOSSound。

铃声时长应小于30s。

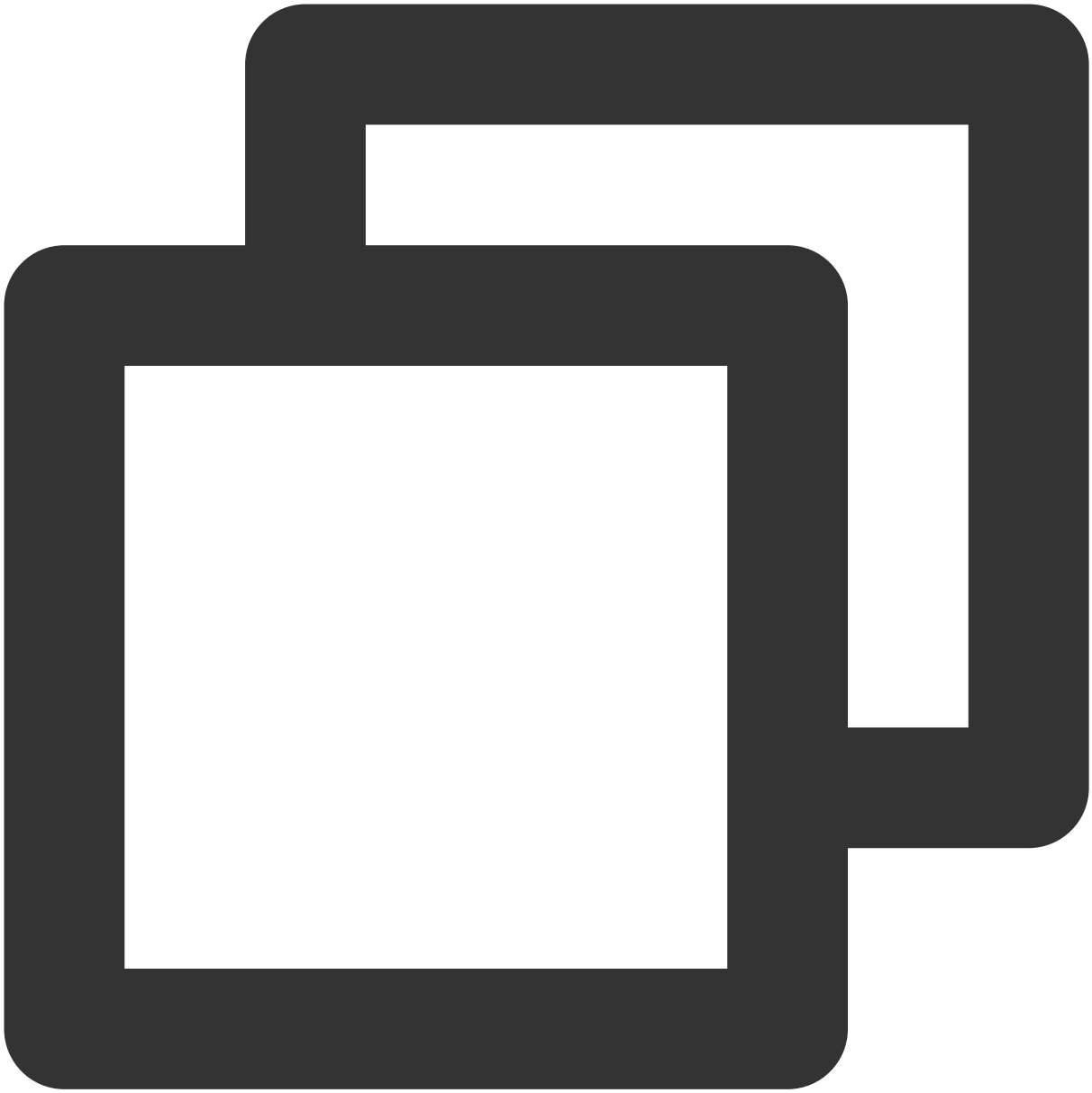
Objective-C

Swift



```
[[TUICallKit createInstance] call:@"mike 的 id" params:[self getCallParams] callMed

- (TUICallParams *)getCallParams {
```

```
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}

+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChanne
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery r
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/developmen
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```



```
let params = TUICallParams()
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = "TUICallKit"
pushInfo.desc = "TUICallKit.have.new.invitation"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
```

```
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HM
pushInfo.androidHuaWeiCategory = "IM"
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: {
    code, message in
}
```

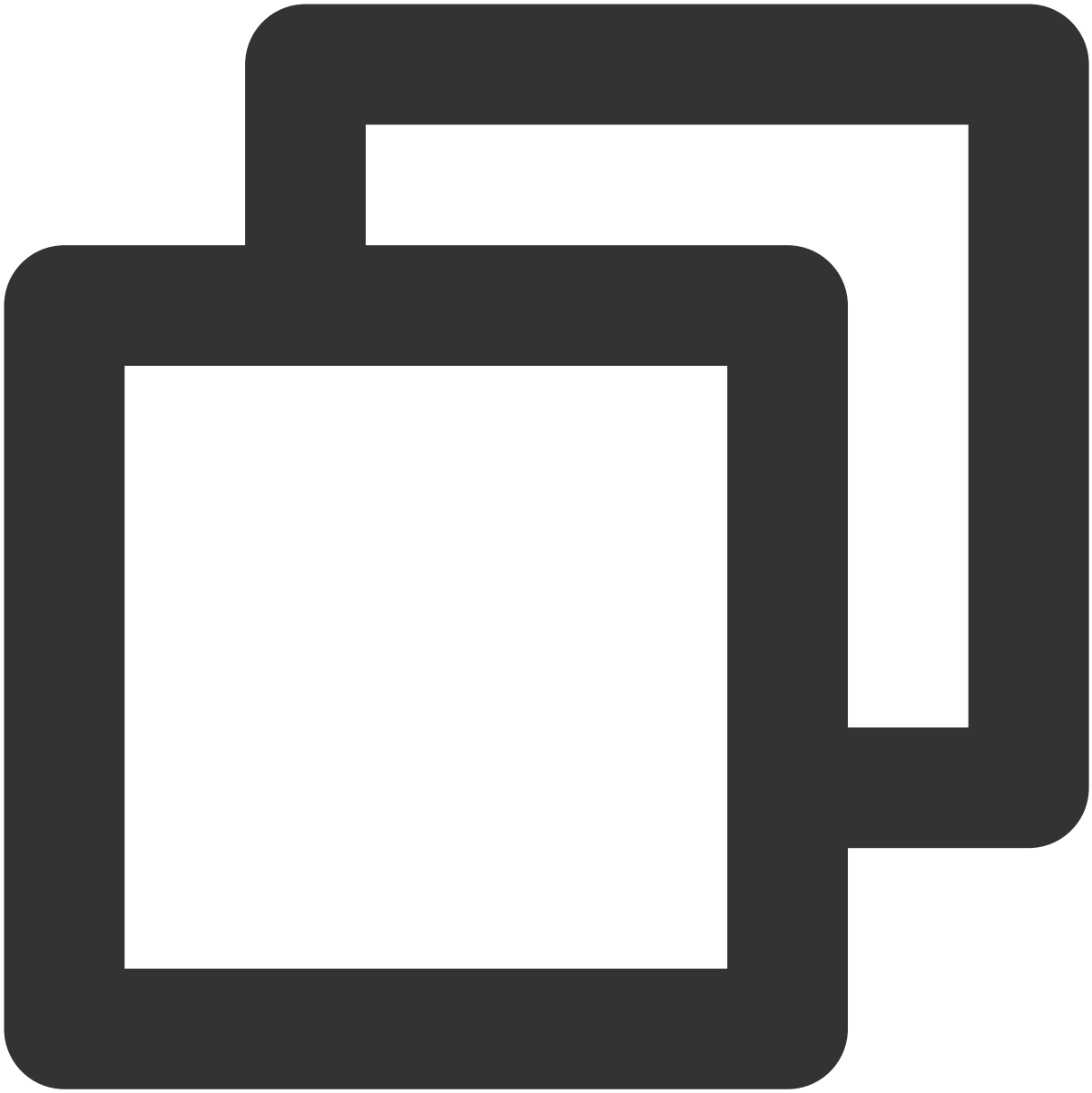
Web&H5

最近更新时间：2024-04-03 17:23:11

本文介绍自定义铃声、静音来电铃声功能的使用。

设置来电铃声

仅限传入本地 MP3 格式的文件地址，确保该文件可以访问的。
如需恢复默认铃声 filePath 传空即可。
使用 ES6 import 方式引入铃声文件。
说明：Vue ≥ v3.0.0 支持



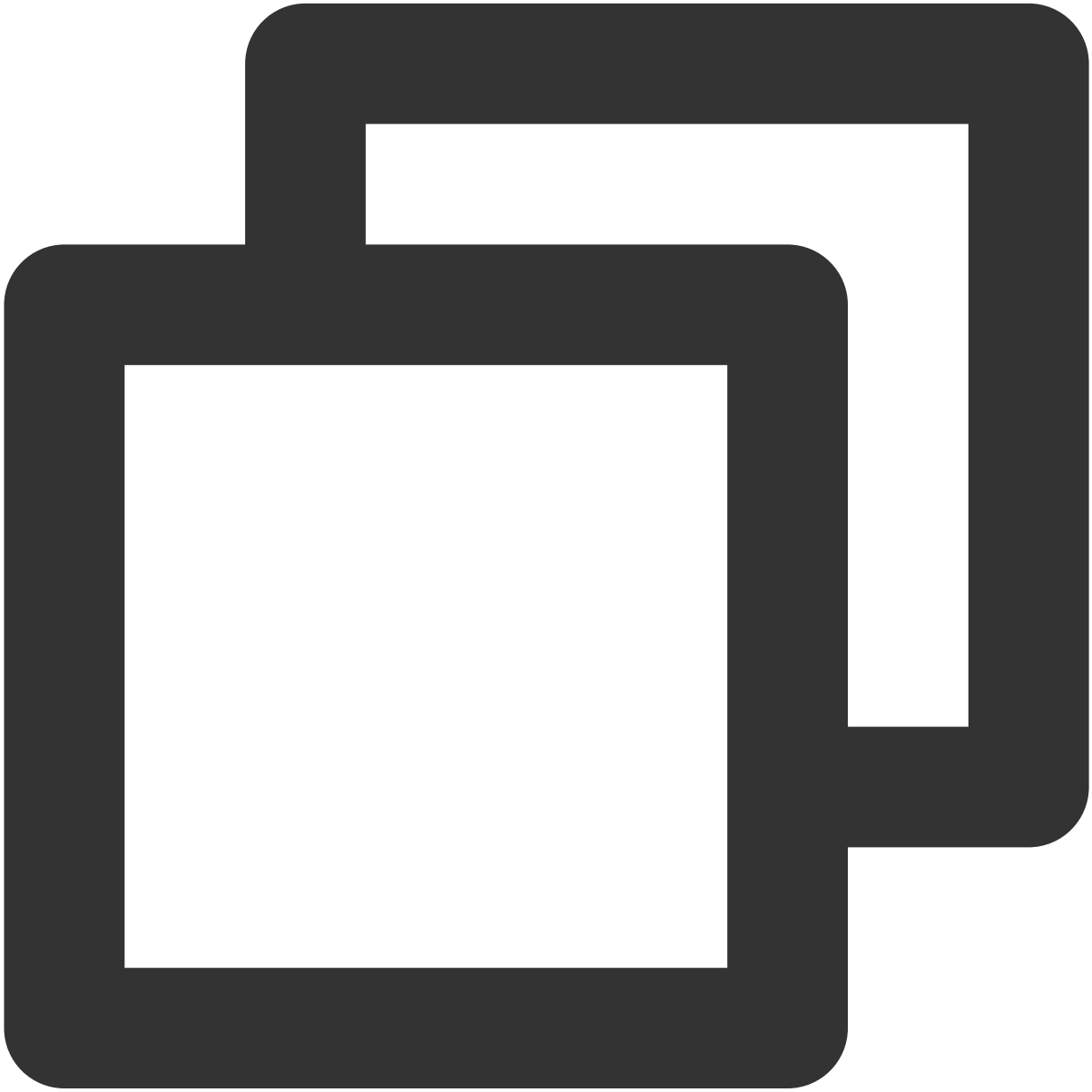
```
import filePath from '../public/ring.mp3';

try {
  await TUICallKitServer.setCallingBell(filePath?: string);
} catch (error: any) {
  alert(`[TUICallKit] setCallingBell API failed. Reason: ${error}`);
}
```

静音来电铃声

开启/关闭来电铃声。

开启后，收到通话请求时，不会播放来电铃声。

说明：Vue ≥ v3.1.2 支持



```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean);
} catch (error: any) {
  alert(`[TUICallKit] enableMuteMode API failed. Reason: ${error}`);
```

```
}
```

uni-app (Anroid&iOS)

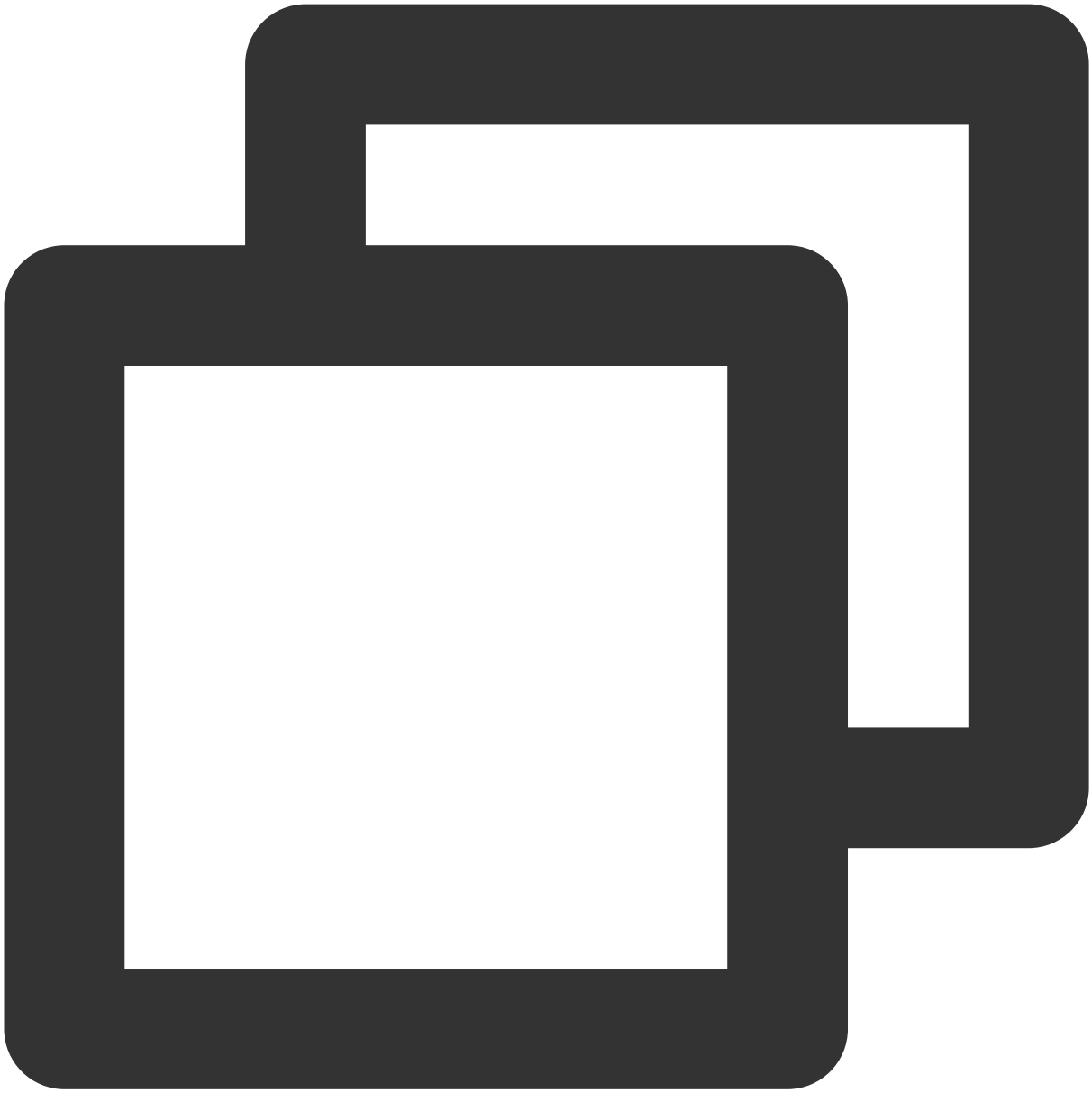
最近更新时间：2024-04-03 17:23:11

本文介绍自定义铃声、静音来电铃声功能的使用。

自定义来电铃声

设置自定义来电铃音，这里**仅限传入本地文件地址，需要确保该文件目录是应用可以访问的**。

版权所有：腾讯云计算（北京）有限责任公司

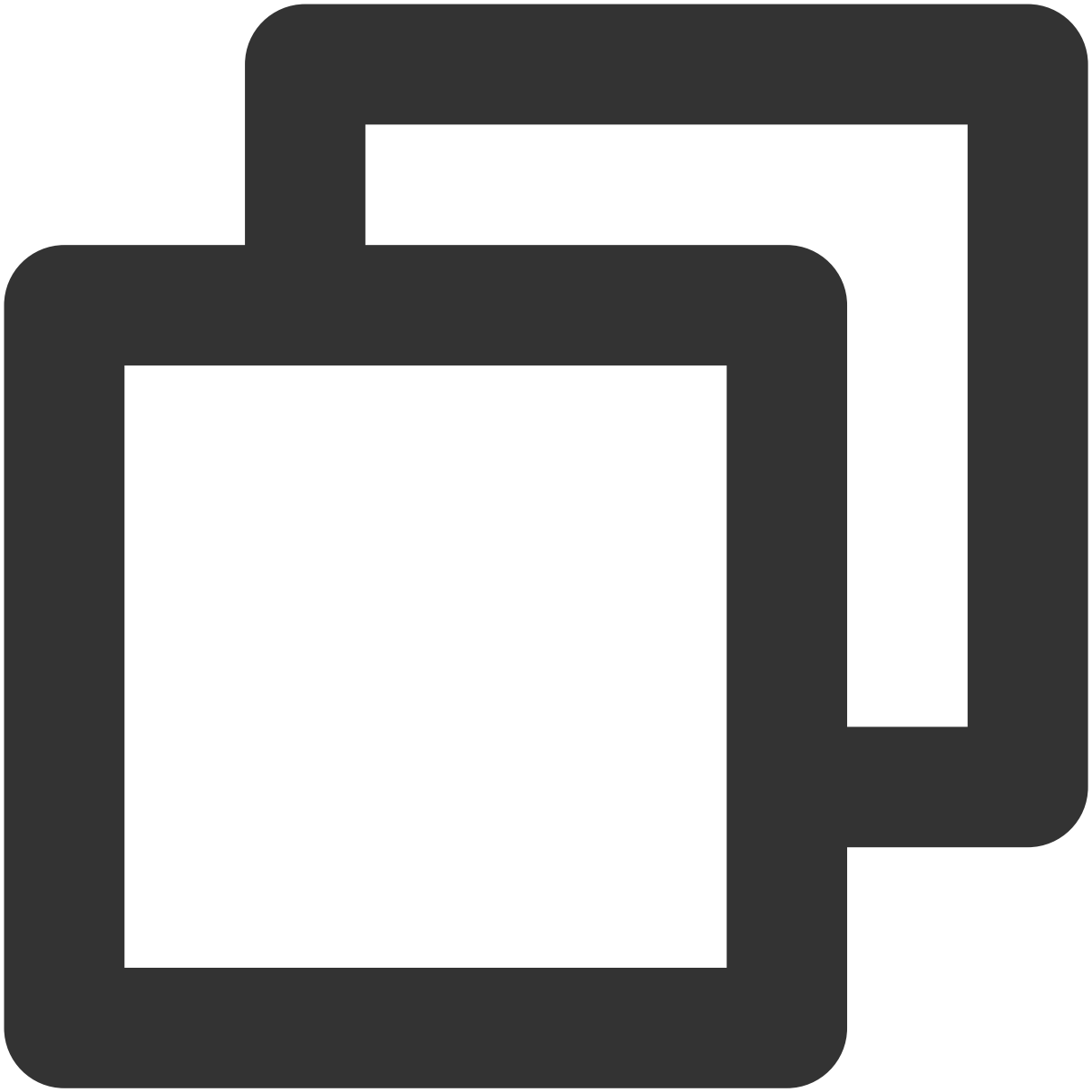


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const tempFilePath = './static/rain.mp3'; // Locally stored audio files
let musicFilePath = '';

uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    console.warn('保存文件成功 = ', JSON.stringify(res));
    musicFilePath =  res.savedFilePath;
    musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);
    // Set ringtone
```

```
TUICallKit.setCallingBell(musicFilePath, (res) => {
  if (res.code === 0) {
    console.log('setCallingBell success');
  } else {
    console.log(`setCallingBell failed, error message = ${res.msg}`);
  }
});
```

静音来电铃声

开启/关闭来电铃声。
开启后，收到通话请求时，不会播放来电铃声。



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableMuteMode(enable);
```

Flutter

最近更新时间：2024-04-03 17:23:11

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

一、 设置应用铃声

设置应用铃声有两种方式：替换铃声音频、调用设置铃声接口。

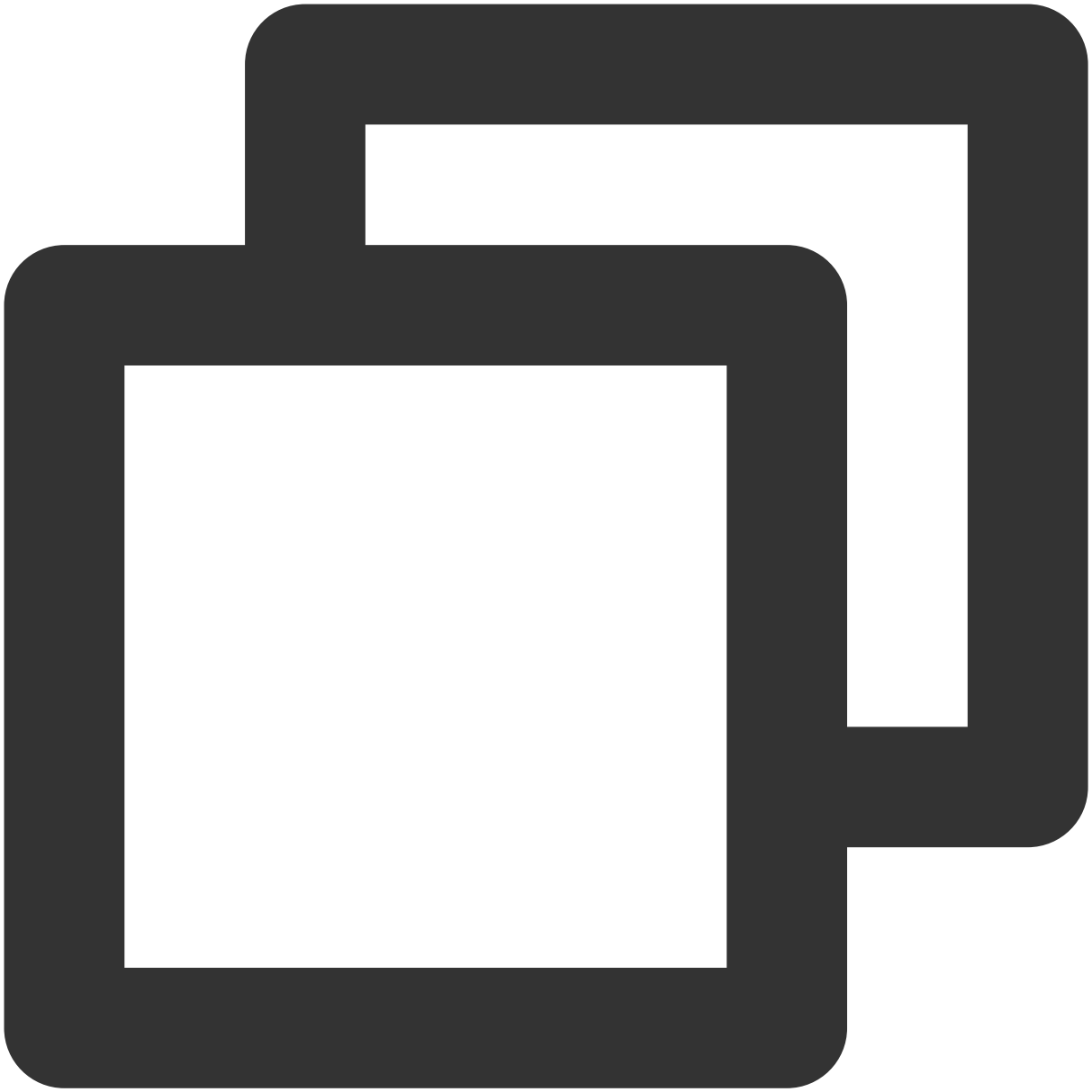
1、替换音频文件

如果您通过源码依赖 TUICallKit 组件，您可以替换 **assets\\audios** 文件夹下的三个音频文件来达到替换铃声的目的：

文件名	用途
phone_dialing.mp3	发起呼叫时的铃音
phone_hangup.mp3	被挂断的铃音
phone_ringing.mp3	接到呼叫时的铃音

2、设置铃声接口

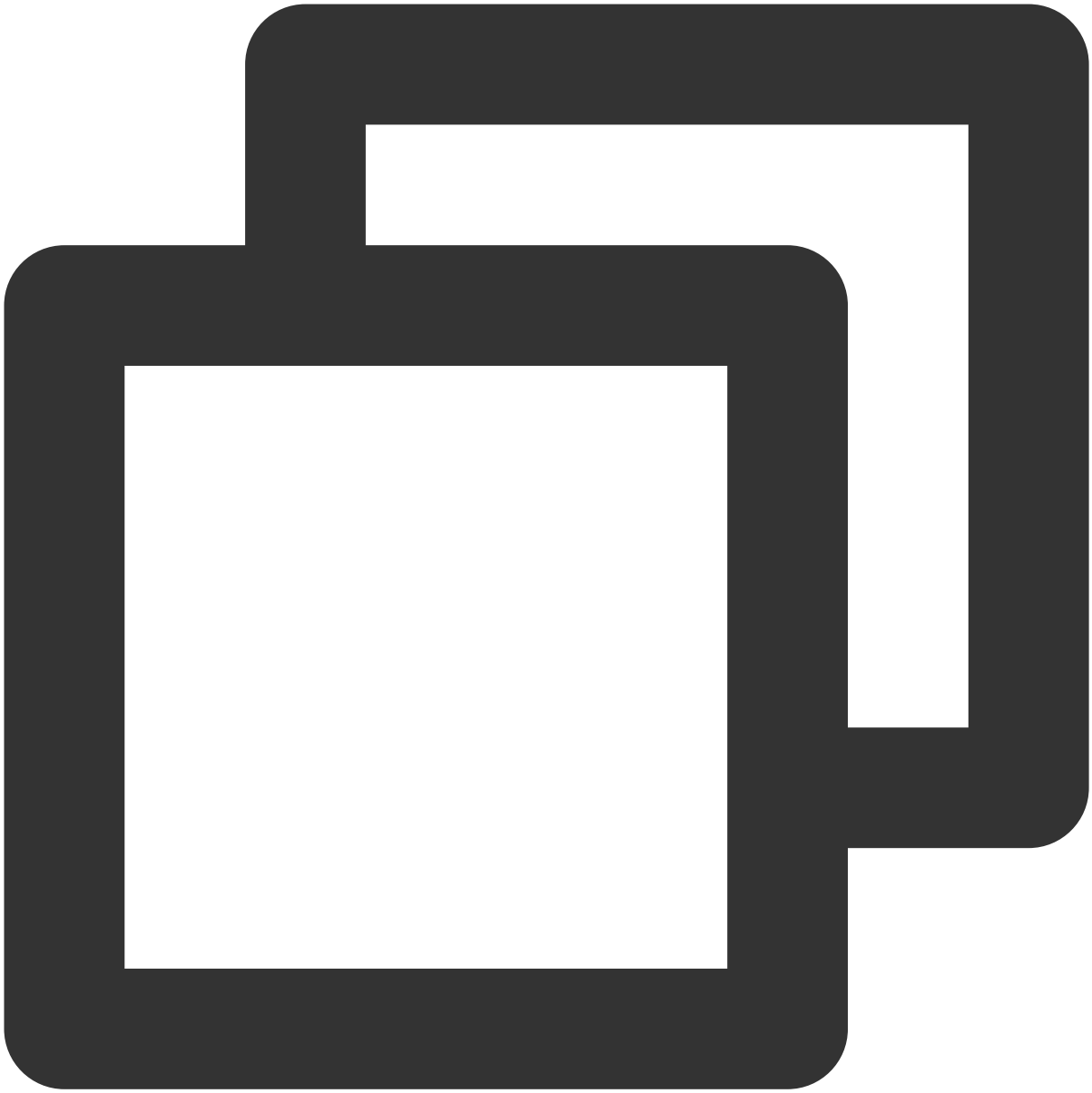
您也可以通过 setCallingBell 接口设置来电铃声。



```
TUICallKit.instance.setCallingBell('flie path');
```

3、 设置静音模式

如果您不需要响铃，您可以通过 enableMuteMode 设置静音模式。



```
TUICallKit.instance.enableMuteMode(true);
```

二、设置离线推送铃音

1、 iOS

Voip 推送不支持自定义推送铃声。 APNs 推送可以修改 `call` 和 `groupcall` 接口中的参数 `params` 中的 `TUIOfflinePushInfo.iOSSound` 设置 `iOS` 平台上的离线消息铃声。

2、 Android

注意：

接口支持华为、小米、FCM 和 APNS。

FCM 的推送铃声为设置的应用铃声。

华为、小米 和 APNS 推送铃声设置请在调用 Call 和 GroupCall 的时候设置 `TUIOfflinePushInfo.iOSSound` 的 `iOSSound` 及 `androidSound` 字段。

监听通话状态

Android&iOS&Flutter

最近更新时间：2024-04-03 17:23:11

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件，可以参考如下代码：

Android（Kotlin）

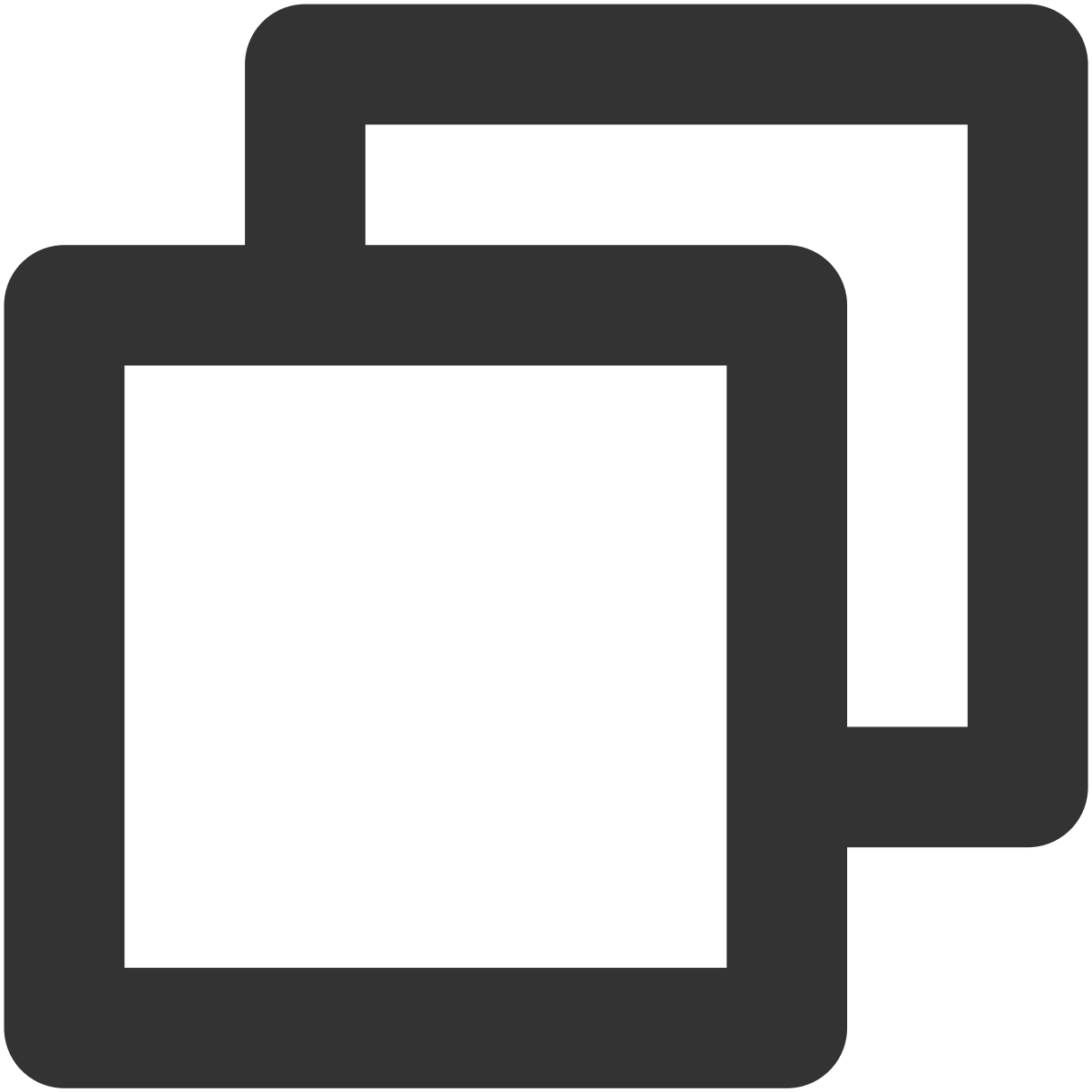
Android（Java）

iOS（Swift）

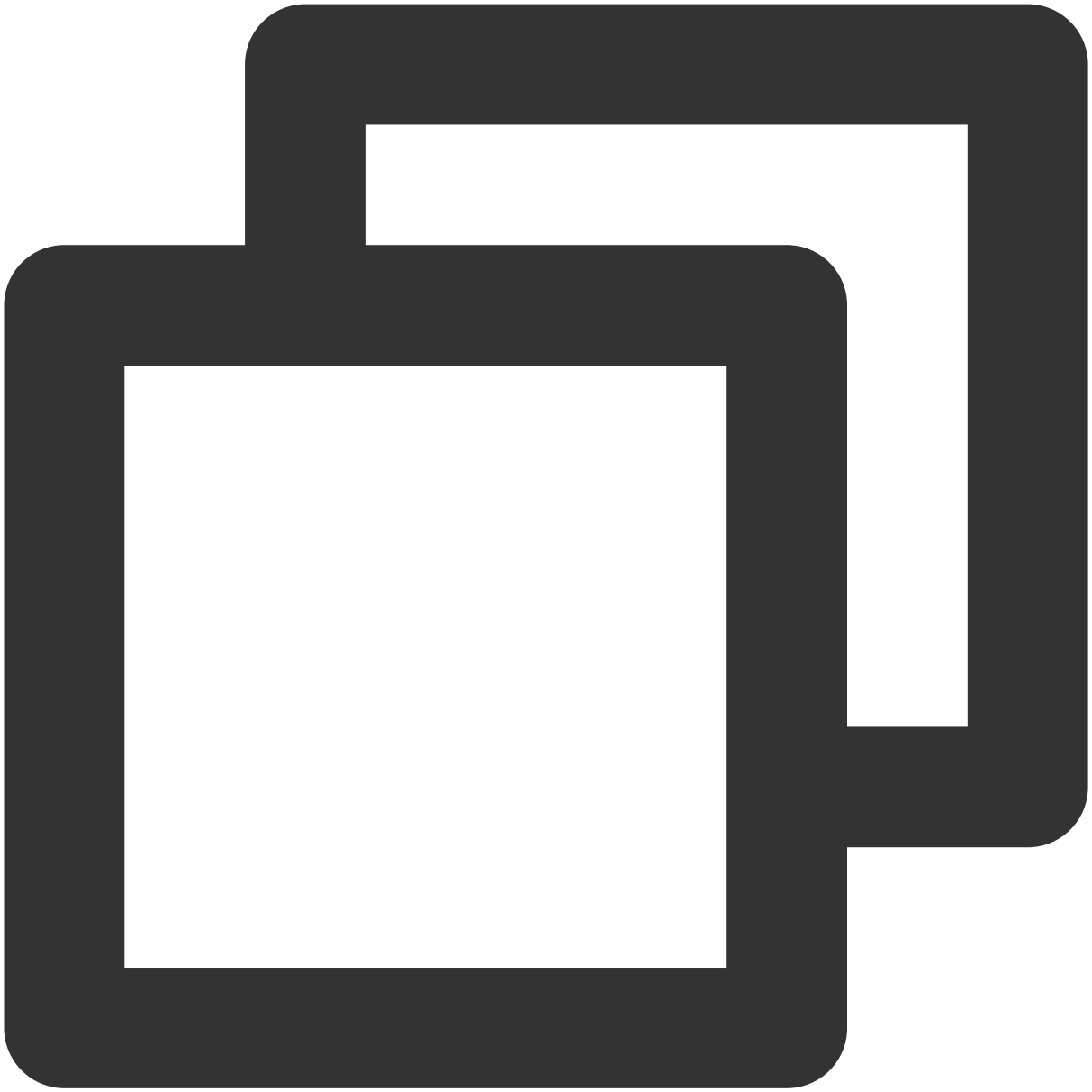
iOS（Objective-C）

Flutter（Dart）

版权所有：腾讯云计算（北京）有限责任公司

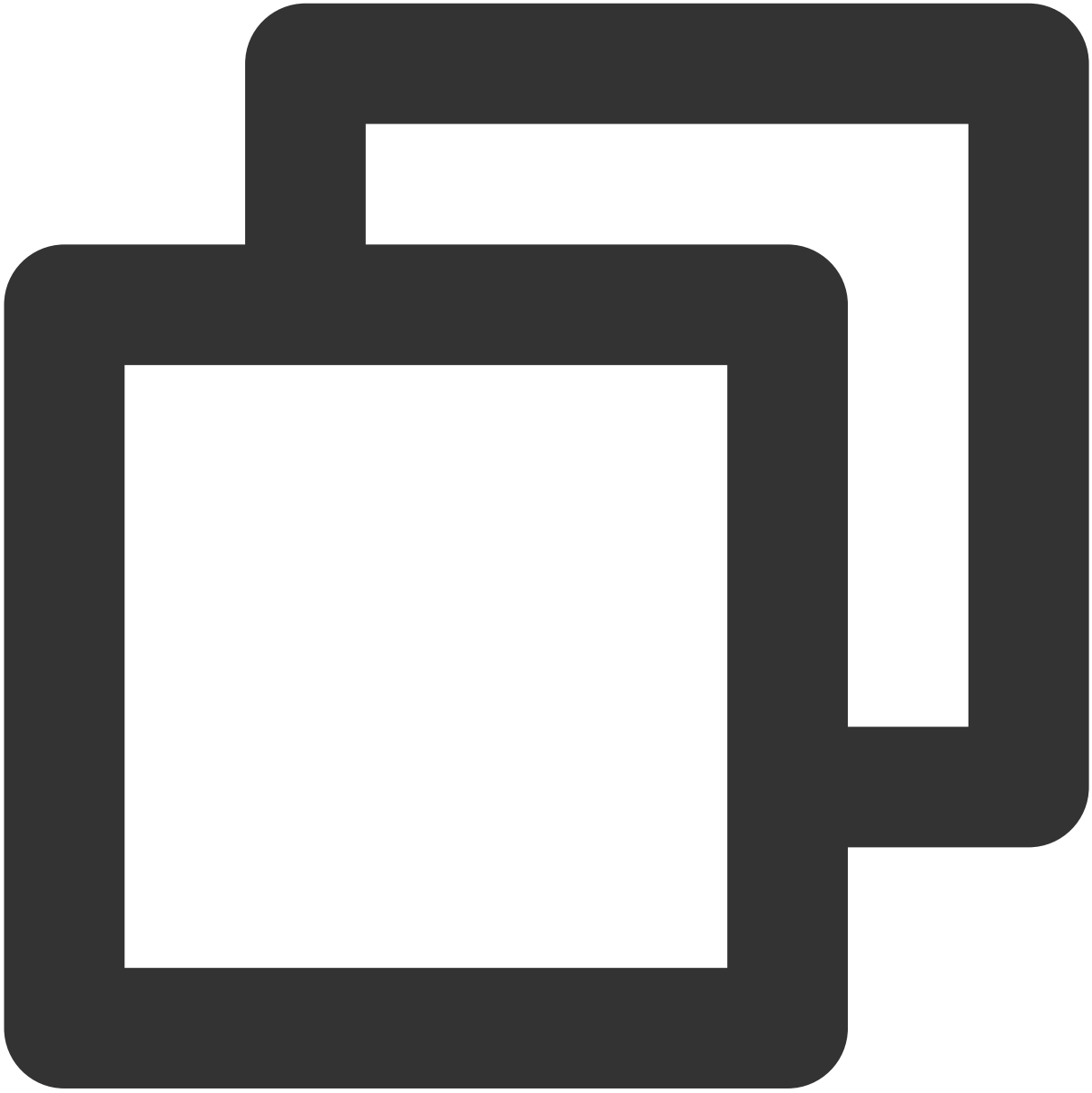


```
private val observer: TUICallObserver = object : TUICallObserver() {
    override fun onCallBegin(roomId: TUICommonDefine.RoomId?, callMediaType: TUICal
    }
    override fun onCallEnd(roomId: TUICommonDefine.RoomId?, callMediaType: TUICallD
    }
    override fun onUserNetworkQualityChanged(networkQualityList: MutableList<TUICom
    }
}
private fun initData() {
    TUICallEngine.createInstance(context).addObserver(observer)
}
```



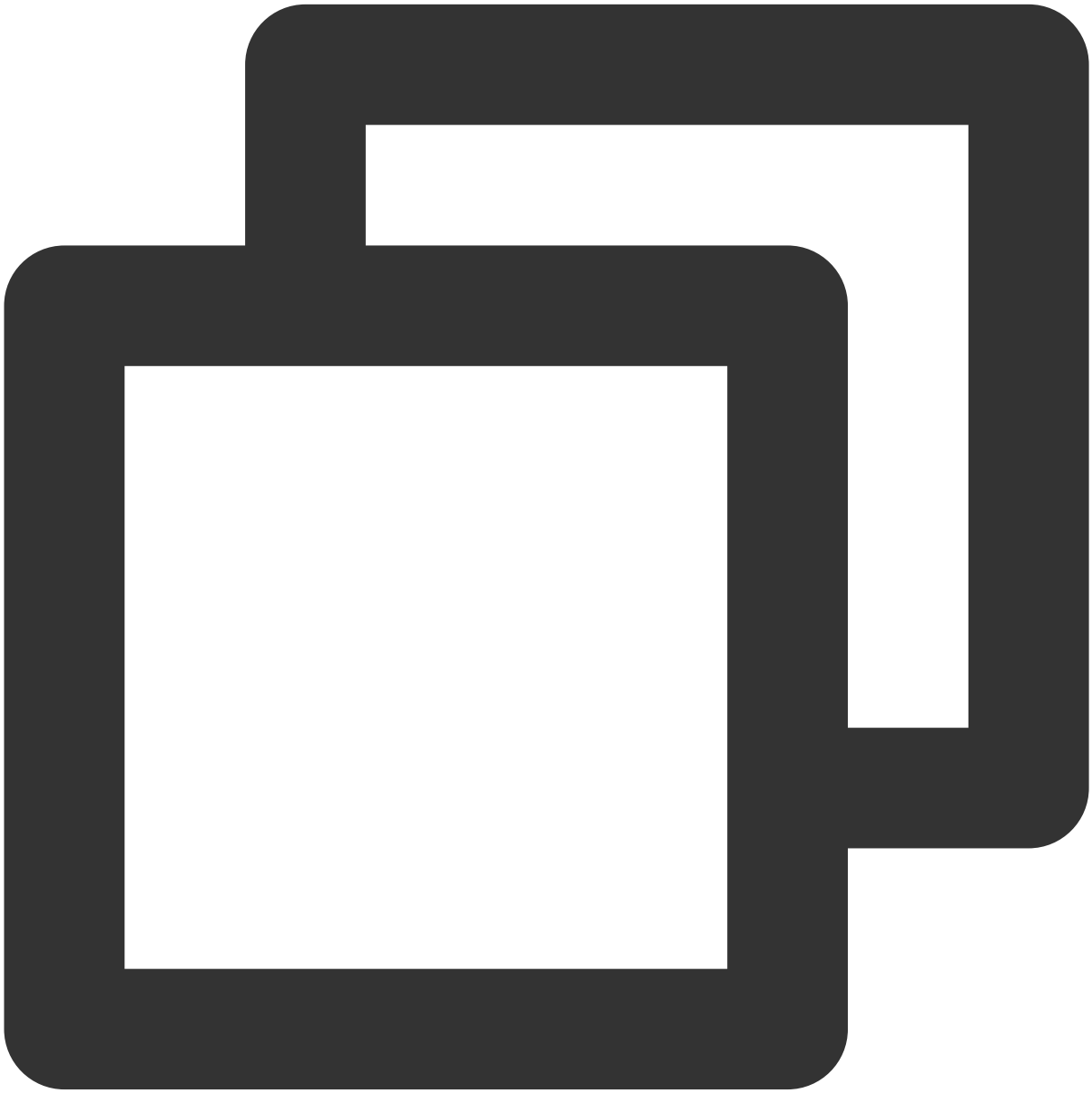
```
private TUICallObserver observer = new TUICallObserver() {
    @Override
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType
    }
    public void onCallEnd(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType ca
    }
    public void onUserNetworkQualityChanged(List<TUICommonDefine.NetworkQualityInfo
    }
};
```

```
private void initData(){
    TUICallEngine.createInstance(context).addObserver(observer);
}
```



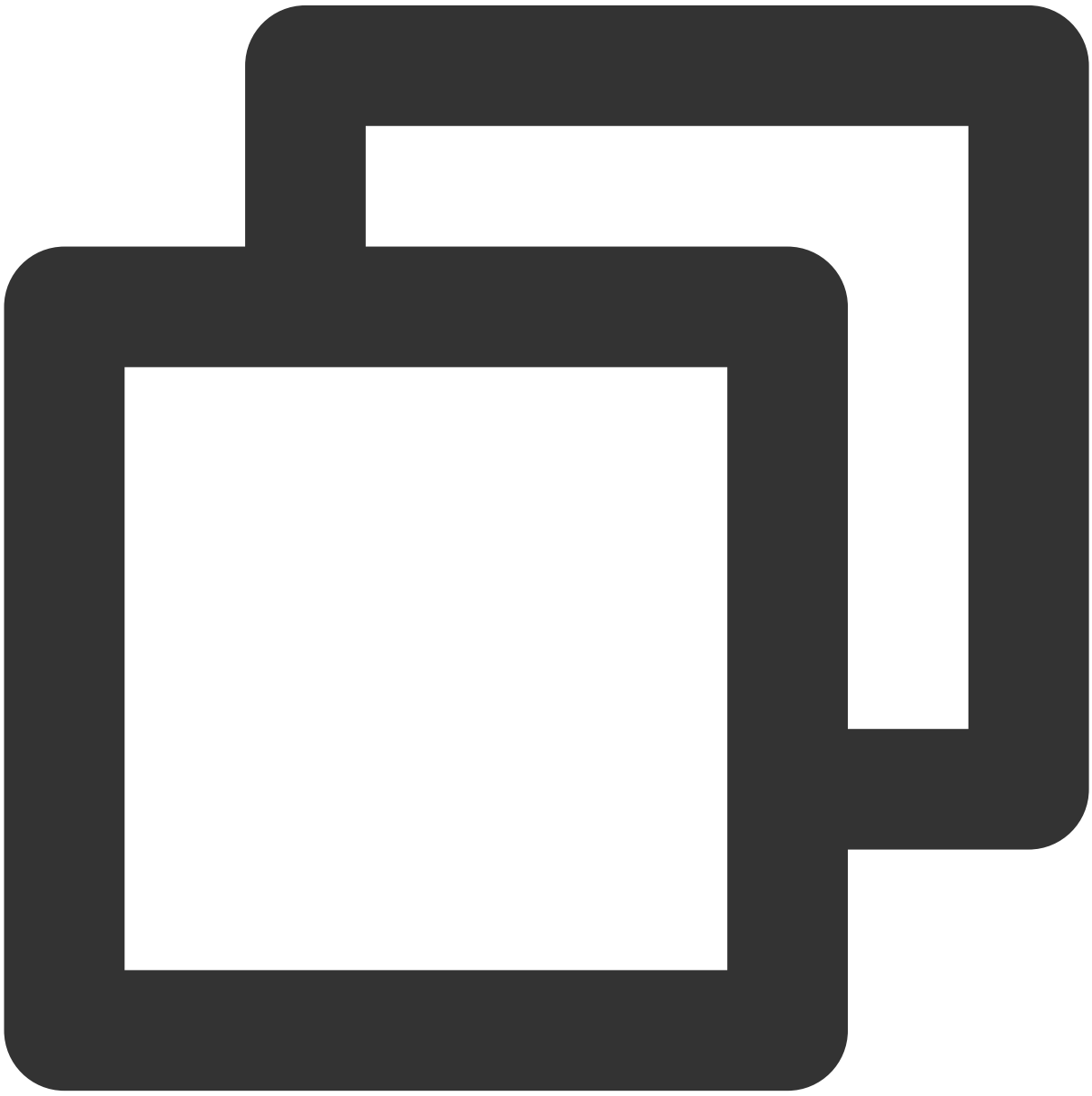
```
import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

public func onCallBegin(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRol

}
```

```
public func onCallEnd(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole:

}
public func onUserNetworkQualityChanged(networkQualityList: [TUINetworkQualityInfo]

}
```



```
#import <TUICallEngine/TUICallEngine.h>

[[TUICallEngine createInstance] addObserver:self];
```

```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy

}
- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType

}
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual

}
```



```
TUICallObserver observer = TUICallObserver(
```

```
    onError: (int code, String message) {
      //您的回调处理逻辑
    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole
      //您的回调处理逻辑
    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca
      //您的回调处理逻辑
    },, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityLis
      //您的回调处理逻辑
    }, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,
      //您的回调处理逻辑
    }
    ……
))

TUICallEngine.instance.addObserver(observer);
```

注意：

在 Android 平台设置 TUICallObserver 监听回调，需保证回调所在类不会被清除，举例：不建议在 LoginActivity 中添加监听，LoginActivity 销毁的时候，回调也会被清除；建议在应用的 Application 中或者应用主界面进行监听。

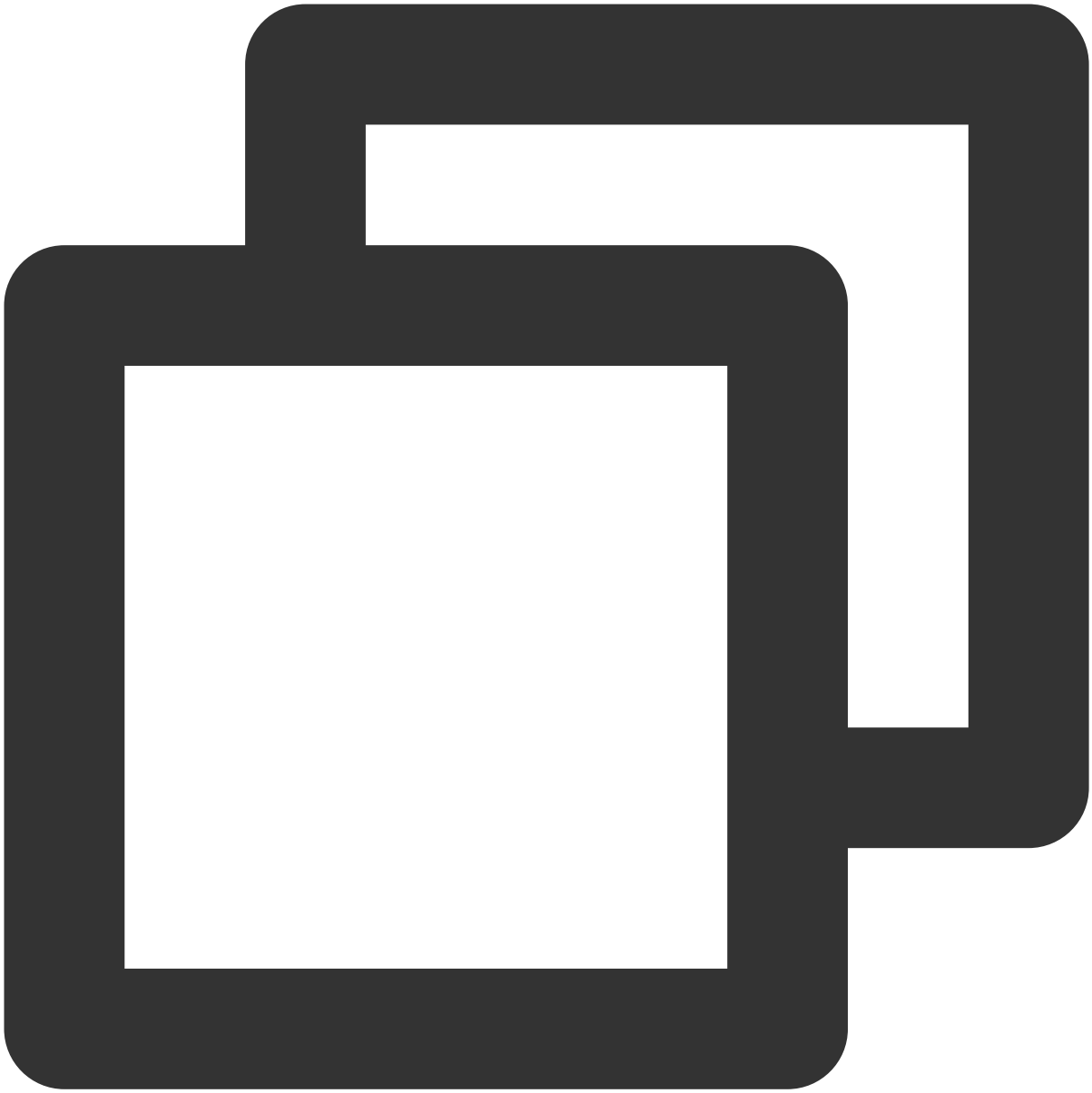
Web&H5

最近更新时间：2024-04-03 17:23:11

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 TUICallEvent），可以参考如下代码：



```
import { TUICallEvent } from 'tuicall-engine-webrtc';

let handleUserEnter = function(event) {
    console.log('TUICallEvent.USER_ENTER: ', event);
};

TUICallKitServer.getTUICallEngineInstance().on(TUICallEvent.USER_ENTER, handleUserE
TUICallKitServer.getTUICallEngineInstance().off(TUICallEvent.USER_ENTER, handleUser
```

版权所有：腾讯云计算（北京）有限责任公司

组件回调事件

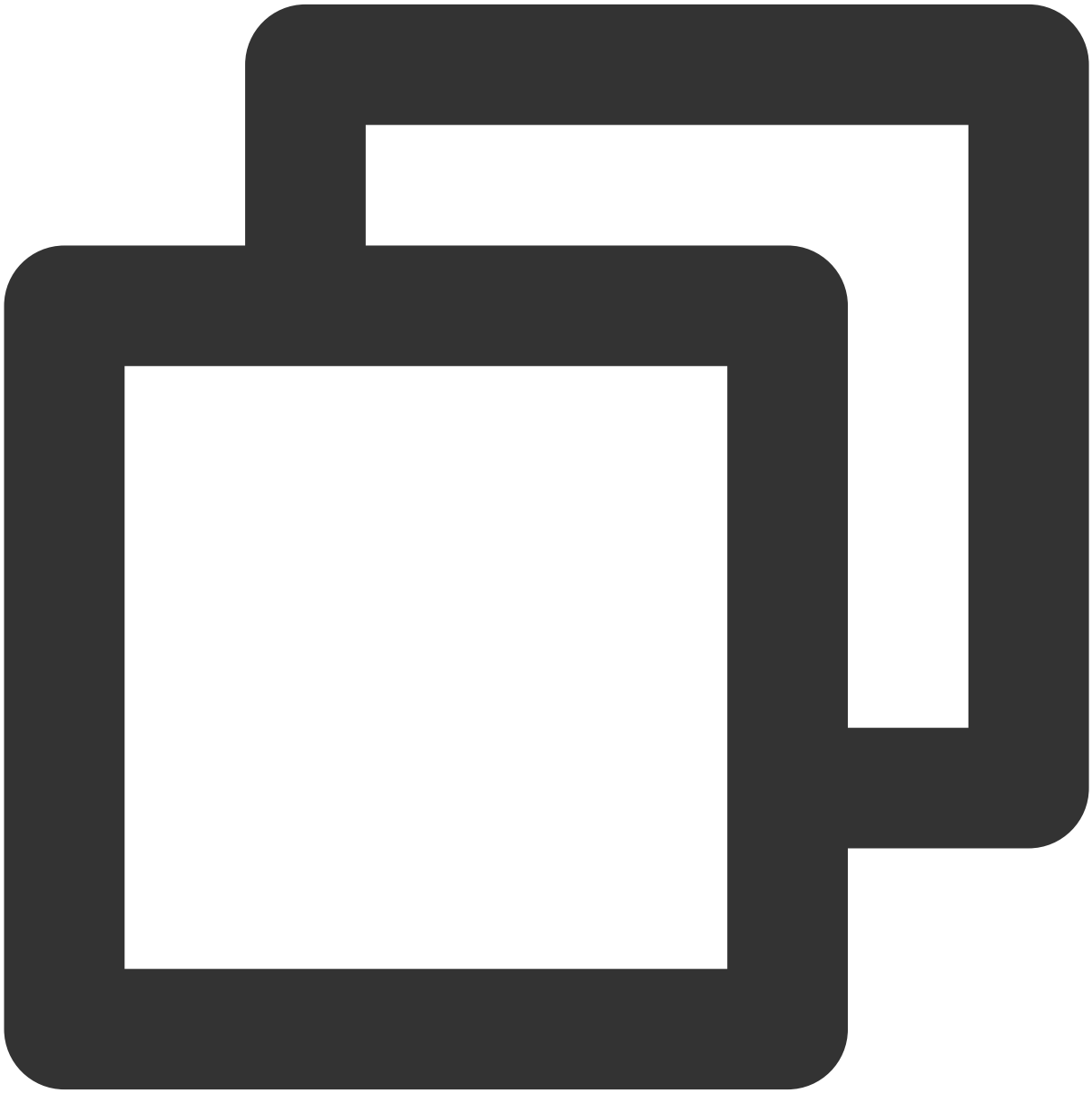
TUICallKit 组件提供了通话状态回调，可以用于业务侧实现更多交互逻辑，具体参考 TUICallKit 属性介绍。

`beforeCalling` ：通话开始前会执行。

`afterCalling` ：通话完成后执行。

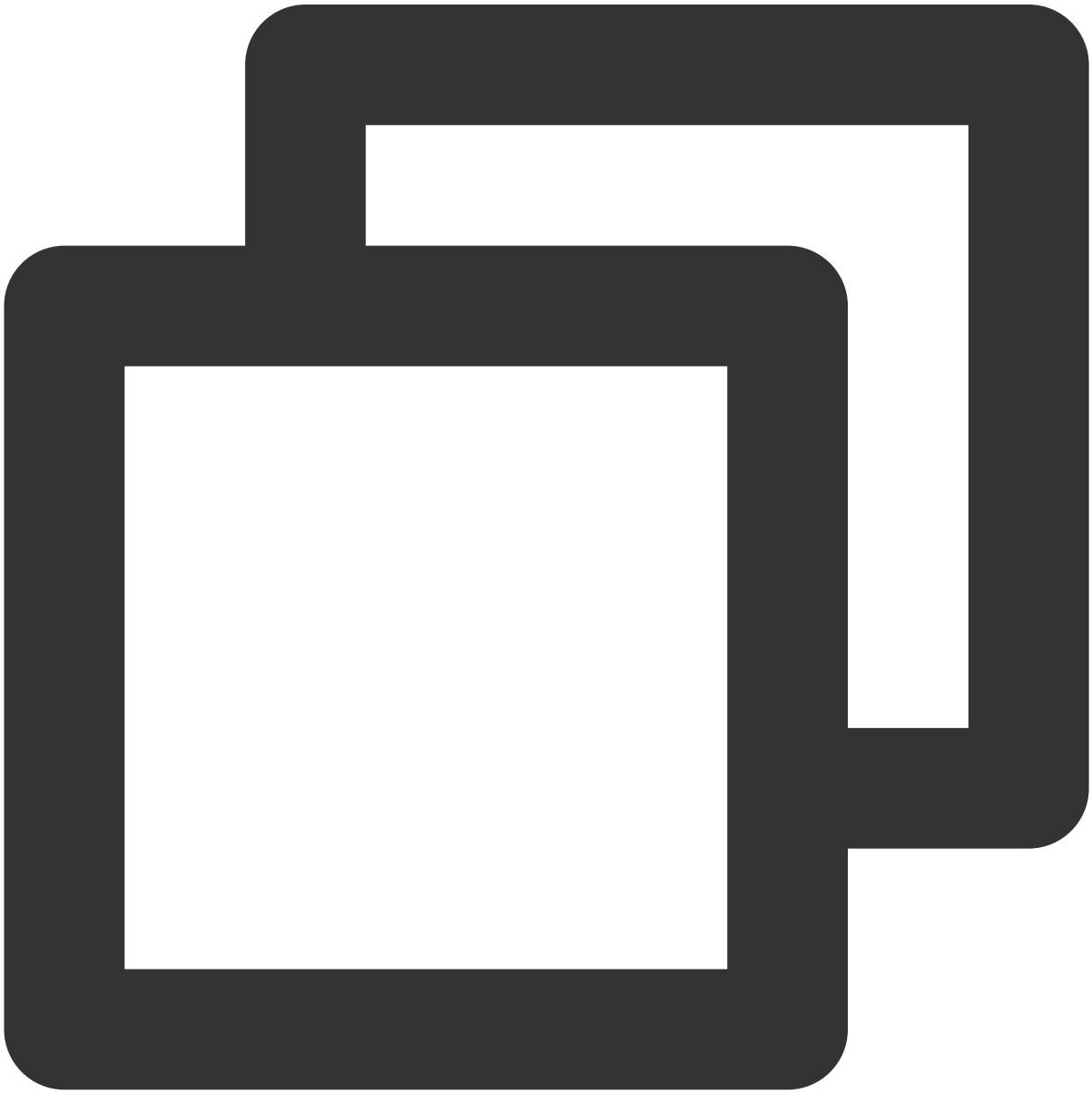
React

Vue



```
function App() {
  const handleBeforeCalling = () => {
```

```
    console.log("[TUICallKit Demo] beforeCalling");
  };
  const handleAfterCalling = () => {
    console.log("[TUICallKit Demo] afterCalling");
  };
  return (
    <TUICallKit
      beforeCalling={handleBeforeCalling}
      afterCalling={handleAfterCalling} />
  )
}
```



```
<template>
  <TUICallKit
    :beforeCalling="handleBeforeCalling"
    :afterCalling="handleAfterCalling" />
</template>
<script setup>
function handleBeforeCalling() {
  console.log("[TUICallKit Demo] beforeCalling");
}
function handleAfterCalling() {
  console.log("[TUICallKit Demo] afterCalling");
}
</script>
```

版权所有：腾讯云计算（北京）有限责任公司

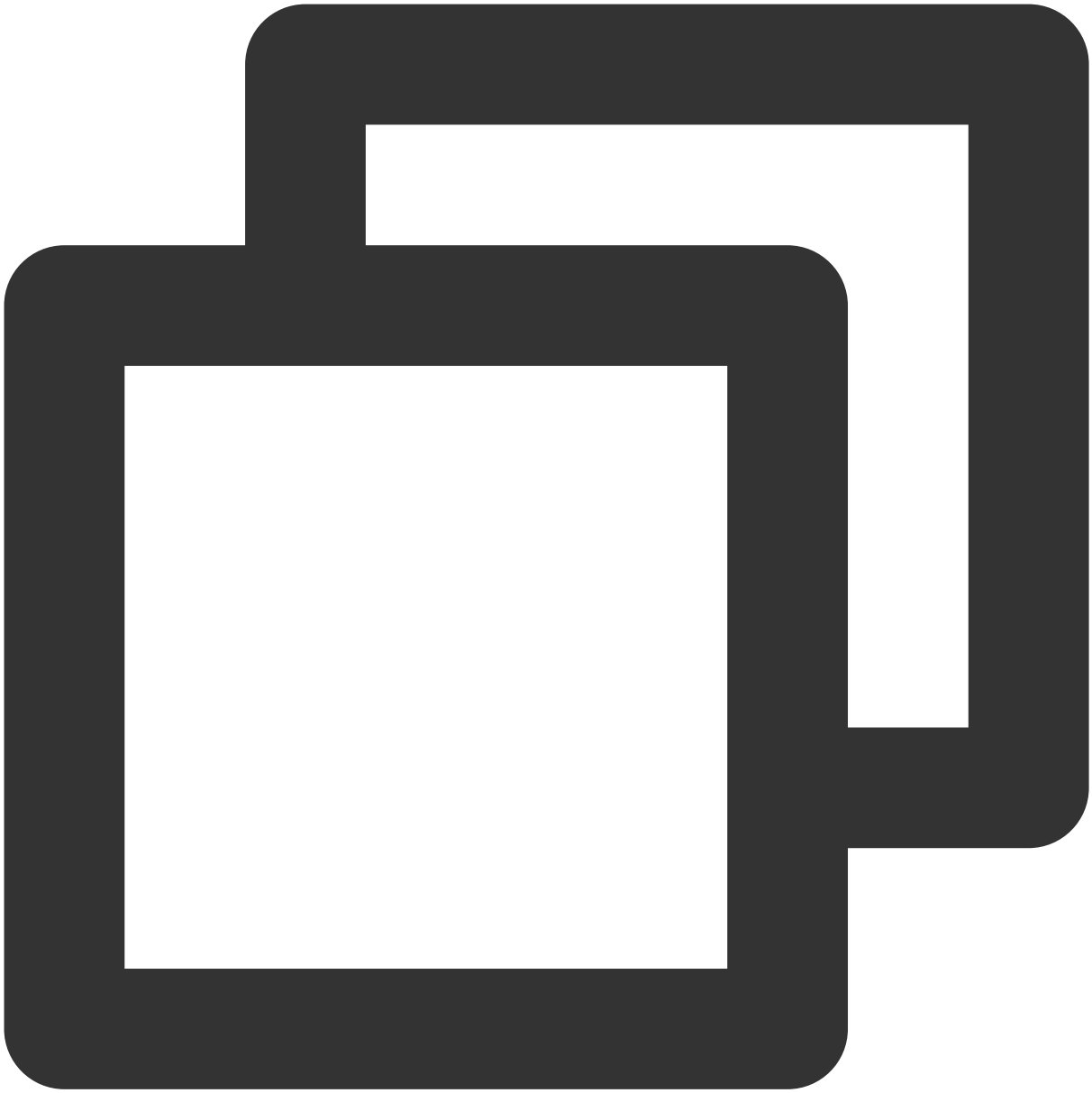
uni-app（Android & iOS）

最近更新时间：2024-04-03 17:23:11

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 事件），请参考如下代码。



```
const TUICallKitEvent = uni.requireNativePlugin('globalEvent');

function handleError(res) {
    console.log('onError', JSON.stringify(res);
}
TUICallKitEvent.addEventListener('onError', handleError);
TUICallKitEvent.removeEventListener('onError', handleError);
```

客户端 API（TUICallKit）

Android

API 概览

最近更新时间：2023-06-21 16:38:57

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的头像、昵称
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权

版权所有：腾讯云计算（北京）有限责任公司

addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

版权所有：腾讯云计算（北京）有限责任公司

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话的媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

关键类型定义

API	描述
TUICallDefine.MediaType	通话的媒体类型，枚举类型：视频通话、语音通话

TUICallDefine.Role	通话的角色，枚举类型：主叫、被叫
TUICallDefine.Status	通话的状态，枚举类型：空闲、待接听、接听中
TUICallDefine.CallParams	通话扩展参数：离线推送消息显示内容
TUICommonDefine.RoomId	音视频房间 ID，支持数字、字符串两种类型
TUICommonDefine.Camera	摄像头Id参数，枚举类型：前摄、后摄
TUICommonDefine.AudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUICommonDefine.NetworkQualityInfo	当前的网络质量信息

TUICallKit

最近更新时间：2023-06-21 16:39:37

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入TUICallKit。

API 概览

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的昵称、头像
call	发起 1v1 通话
call	发起 1v1 通话，支持自定义房间号、通话邀请超时时间，离线推送内容等
groupCall	发起群组通话
groupCall	发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

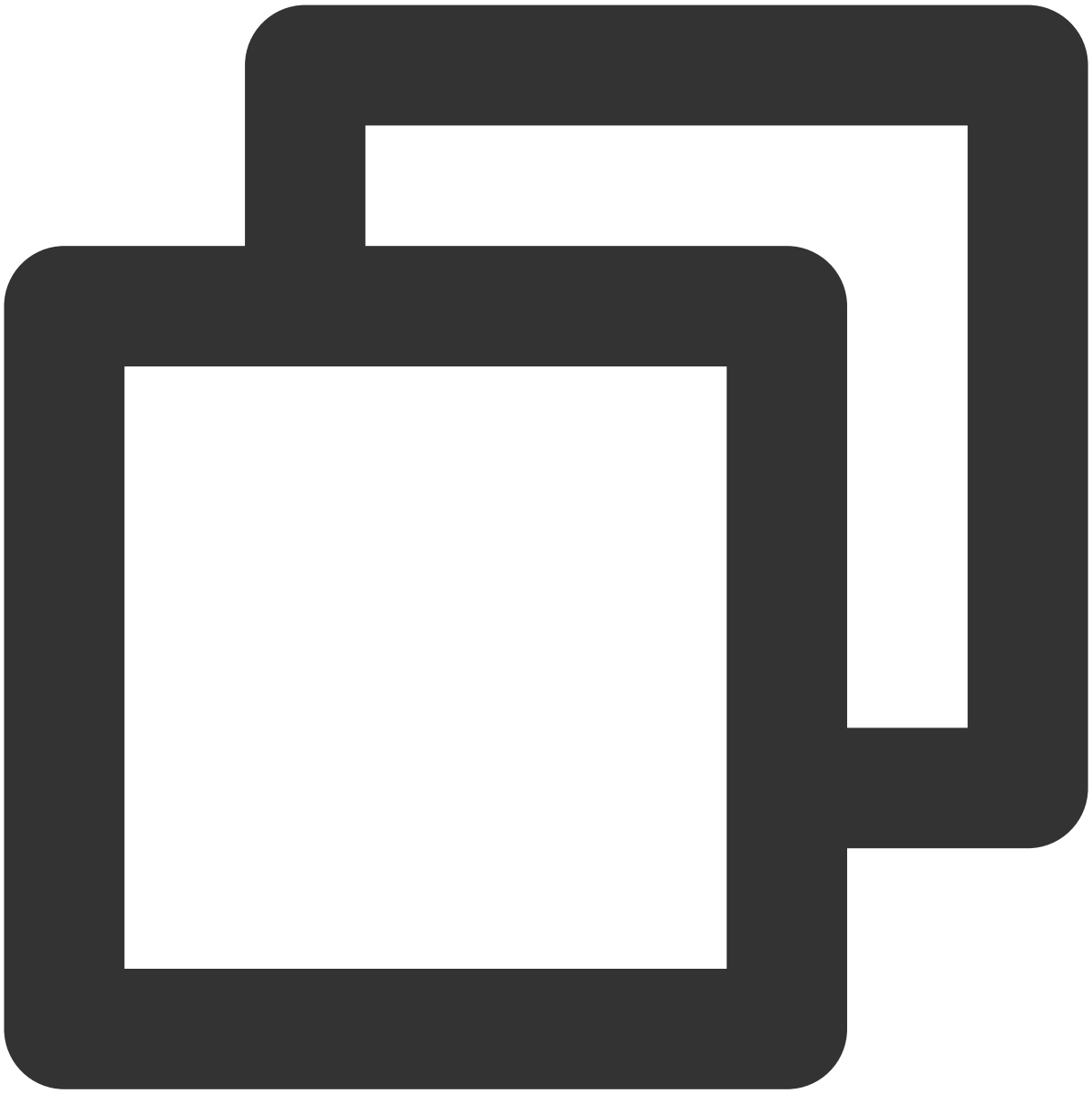
API 详情

createInstance

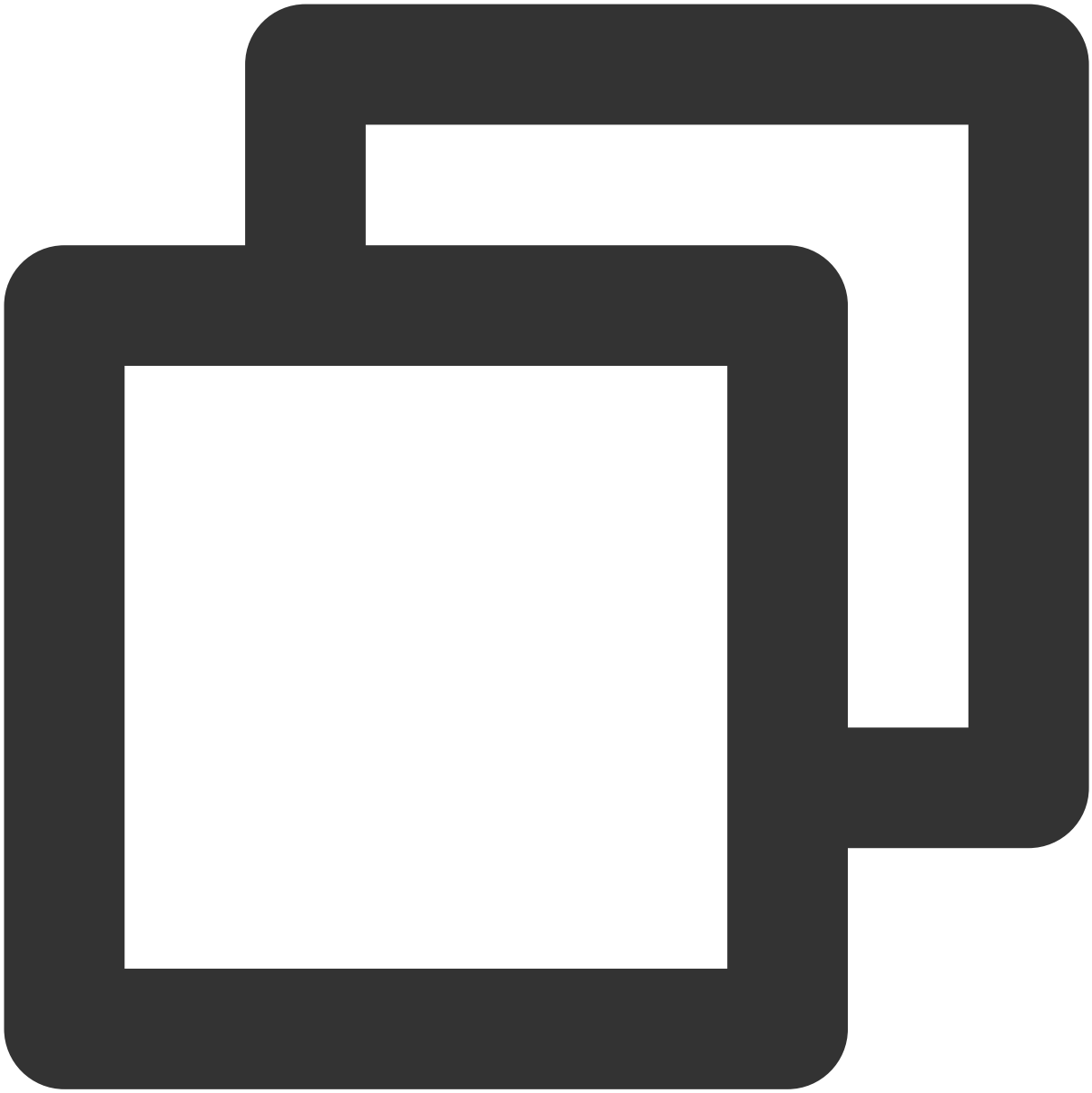
创建 TUICallKit 的单例。

Kotlin

Java



```
fun createInstance(context: Context): TUICallKit
```



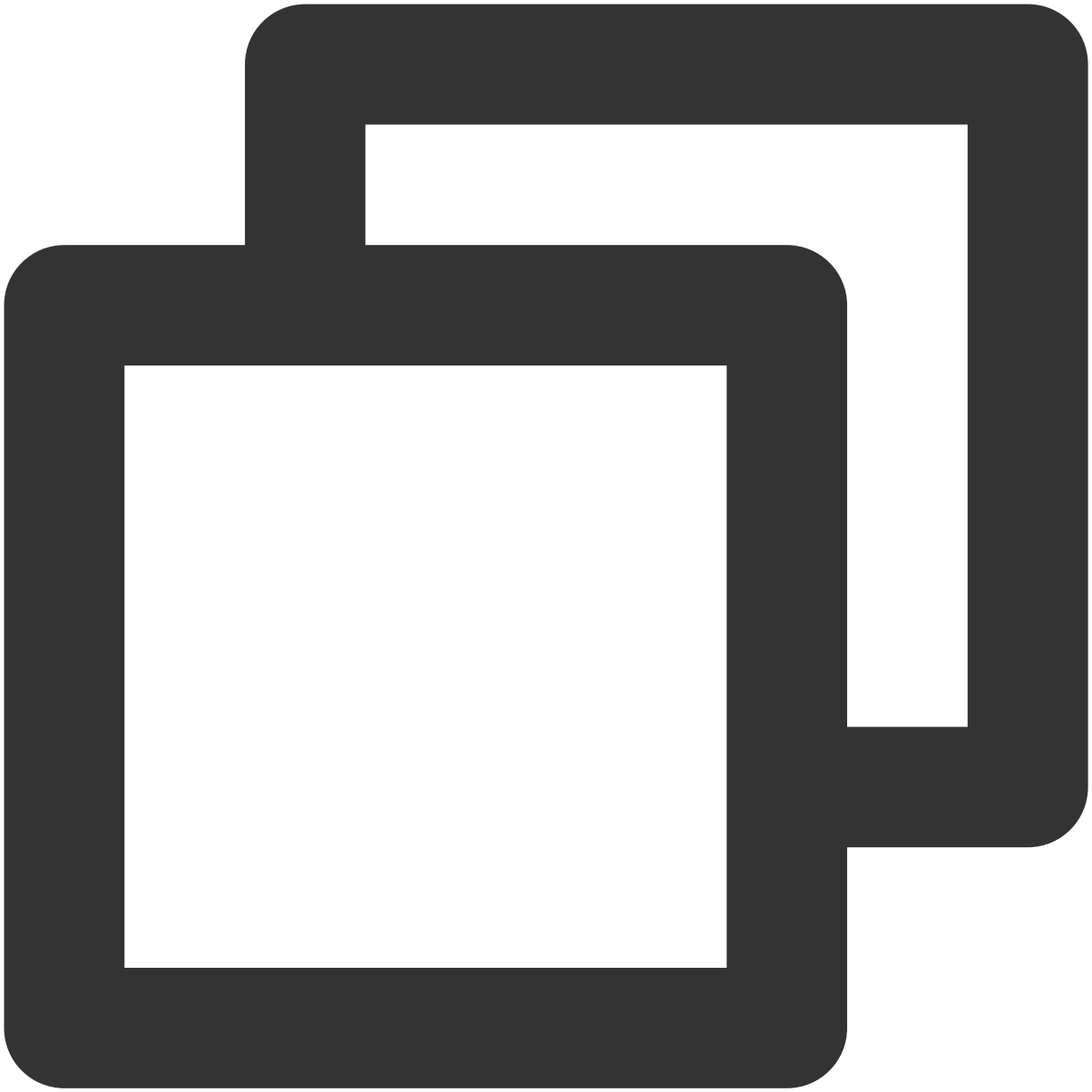
```
TUICallKit createInstance(Context context)
```

setSelfInfo

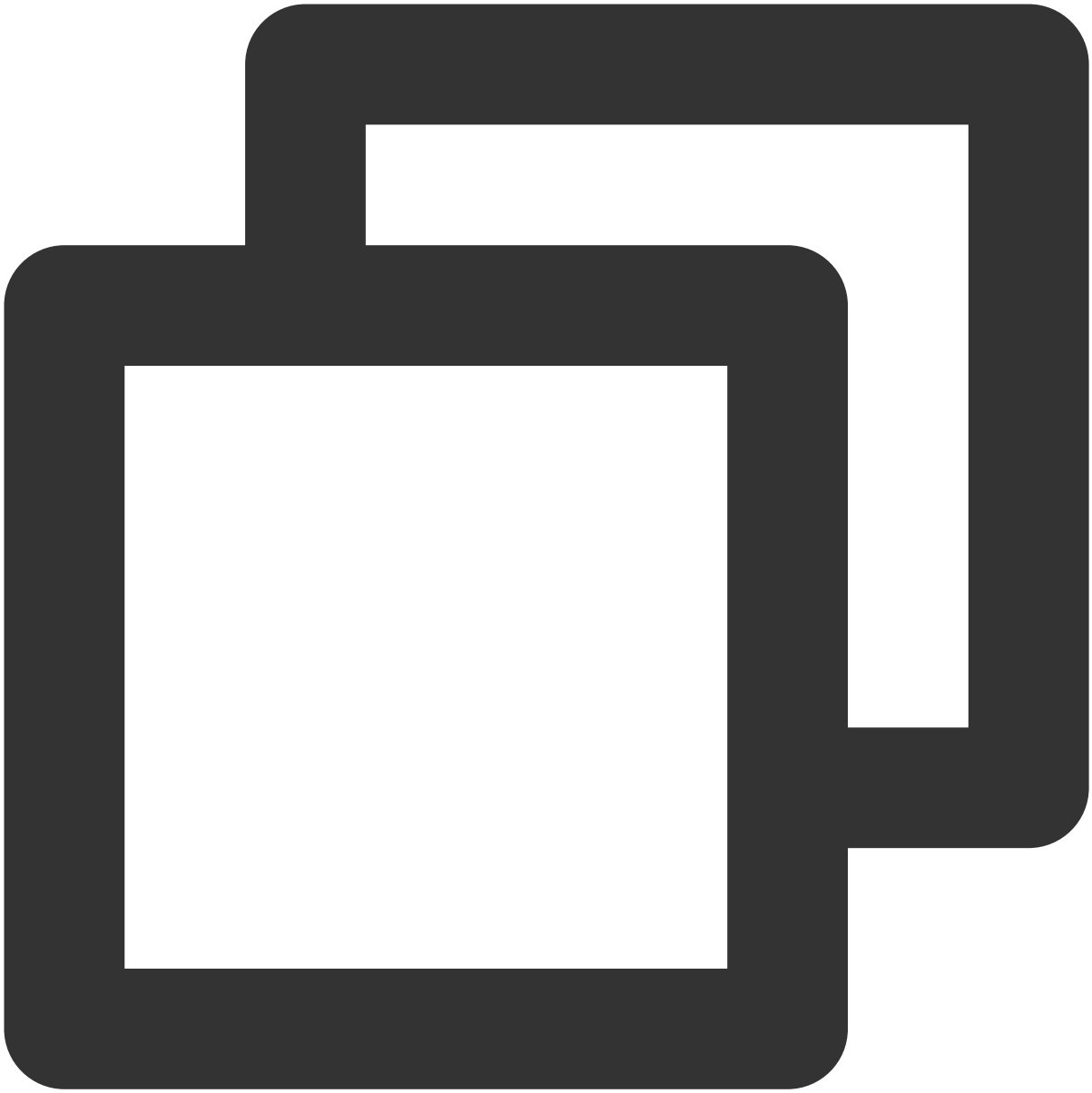
设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

Kotlin

Java



```
fun setSelfInfo(nickname: String?, avatar: String?, callback: TUICommonDefine.Callb
```



```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

参数如下表所示：

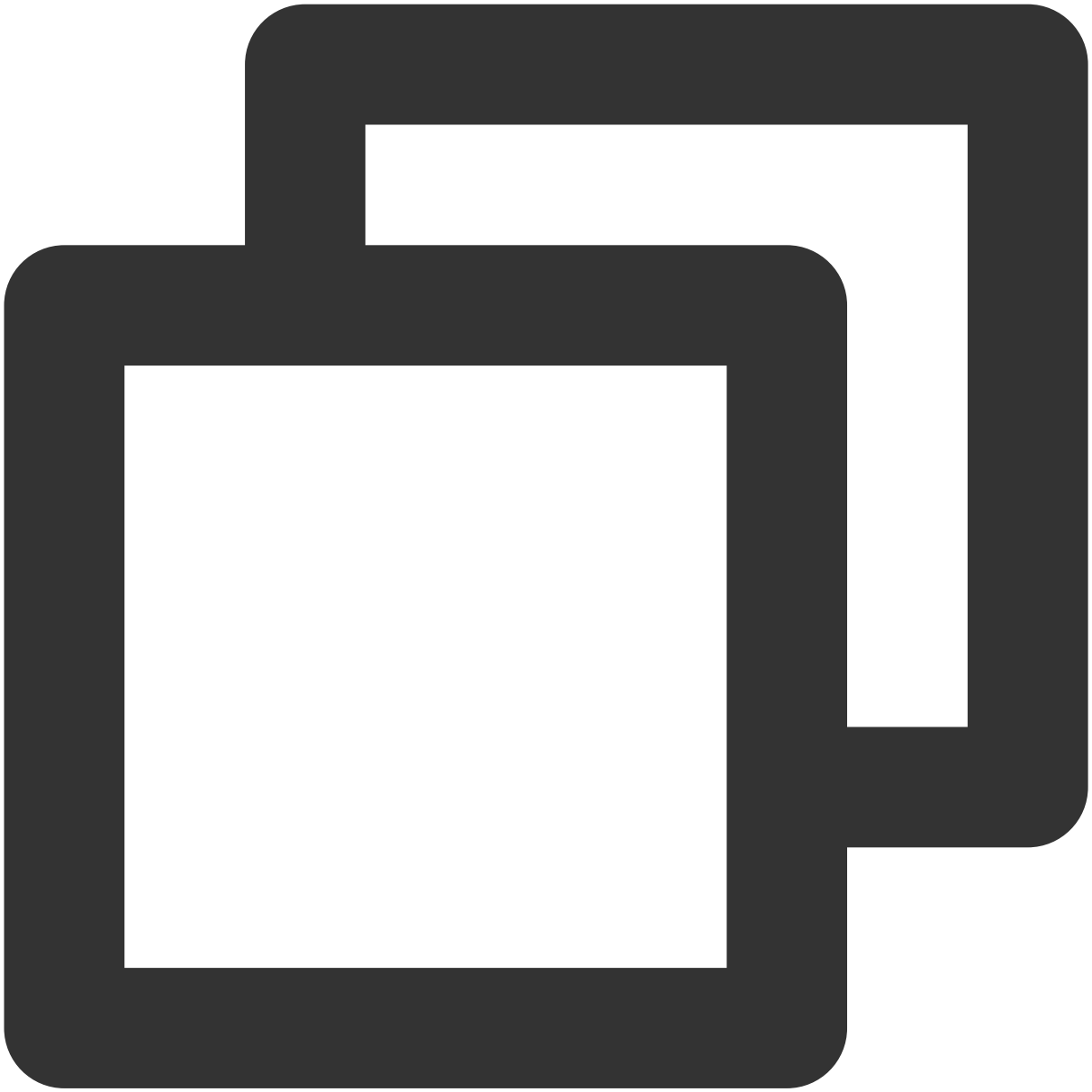
参数	类型	含义
nickname	String	目标用户的昵称
avatar	String	目标用户的头像

call

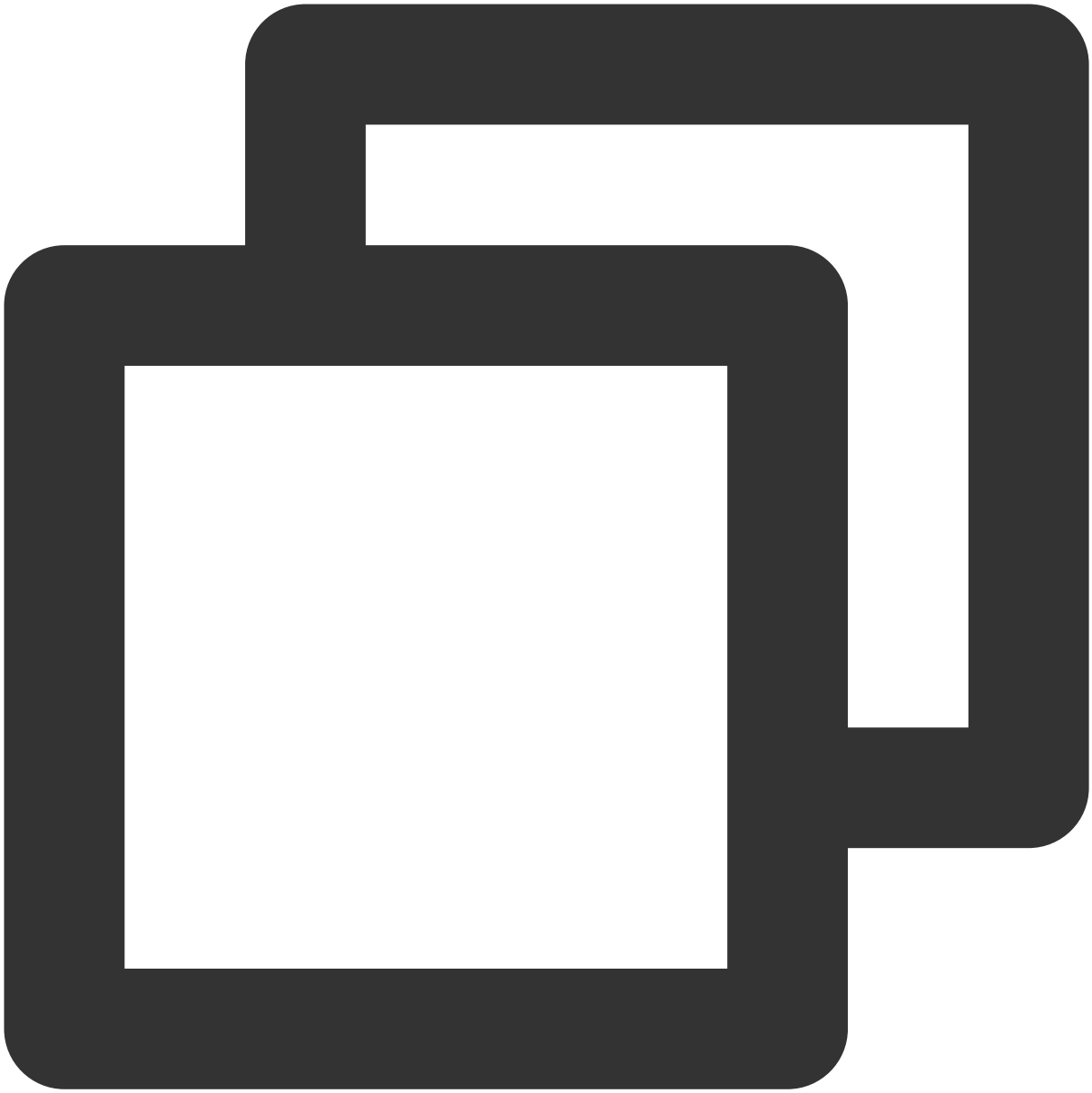
拨打电话（1v1通话）。

Kotlin

Java



```
fun call(userId: String, callMediaType: TUICallDefine.MediaType)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType)
```

参数如下表所示：

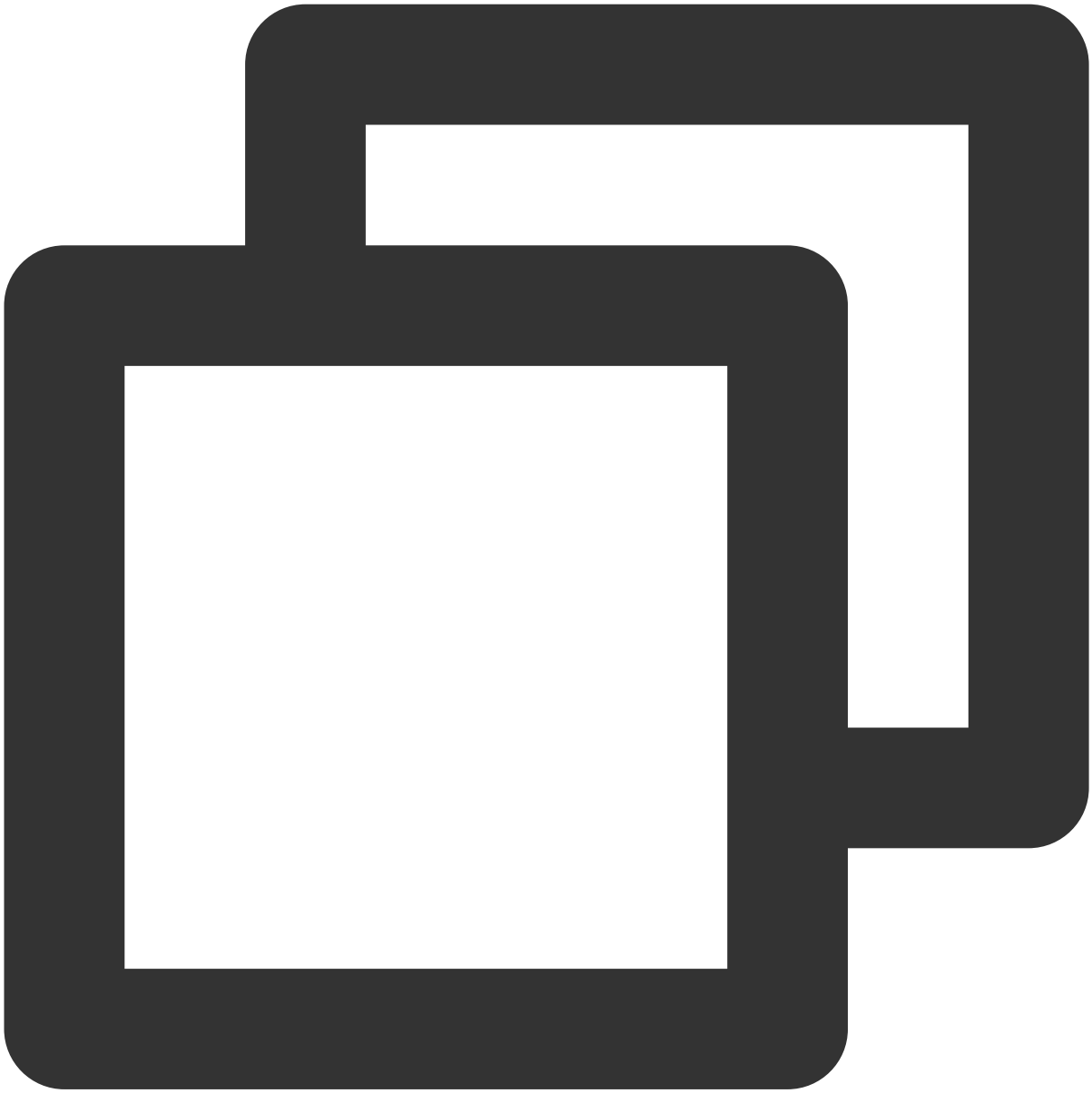
参数	类型	含义
userId	String	目标用户的 userId
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

call

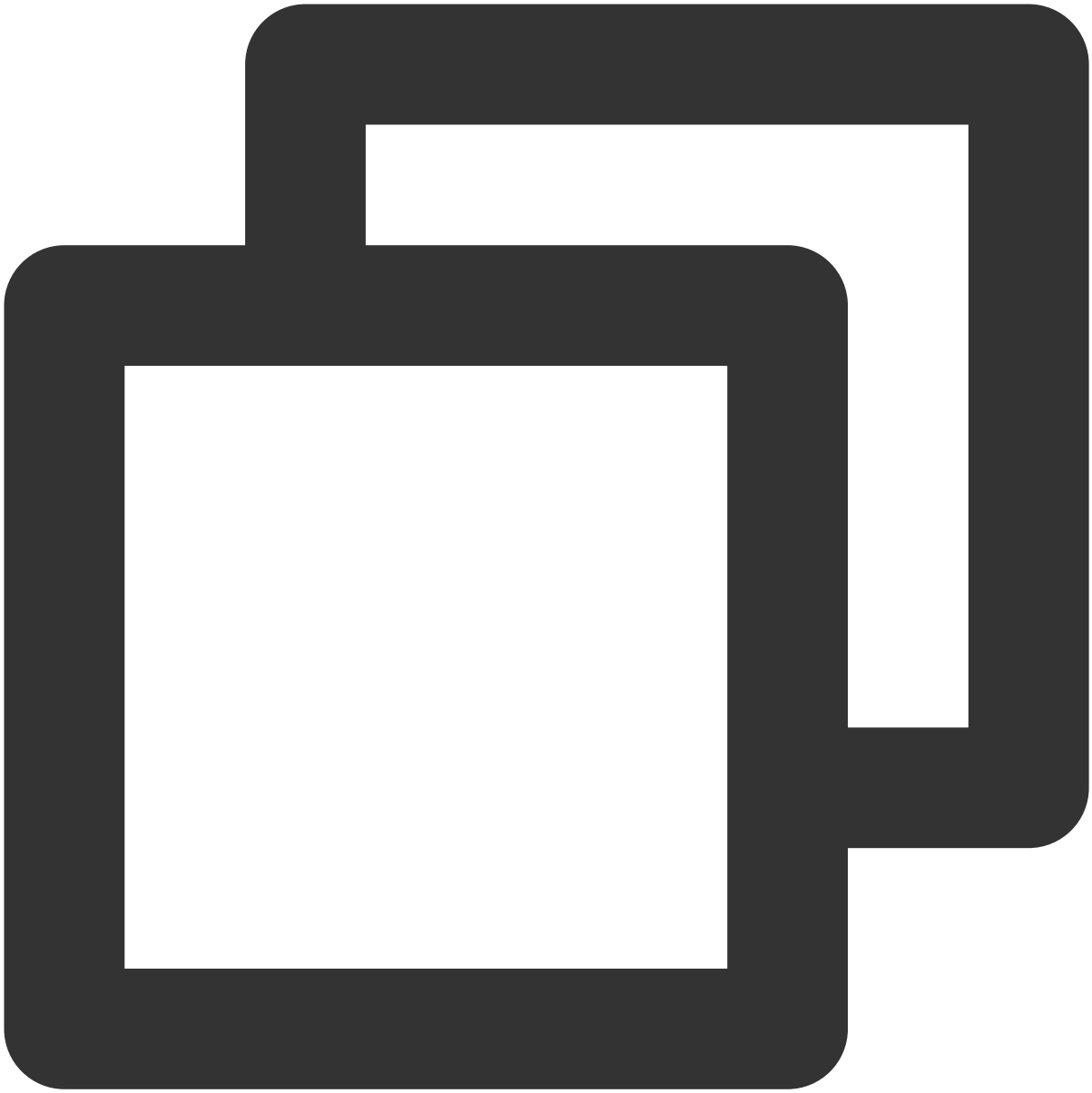
拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。

Kotlin

Java



```
fun call(
    userId: String, callMediaType: TUICallDefine.MediaType,
    params: CallParams?, callback: TUICommonDefine.Callback?
)
```



```
void call(String userId, TUICallDefine.MediaType callMediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

groupCall

发起群组通话。

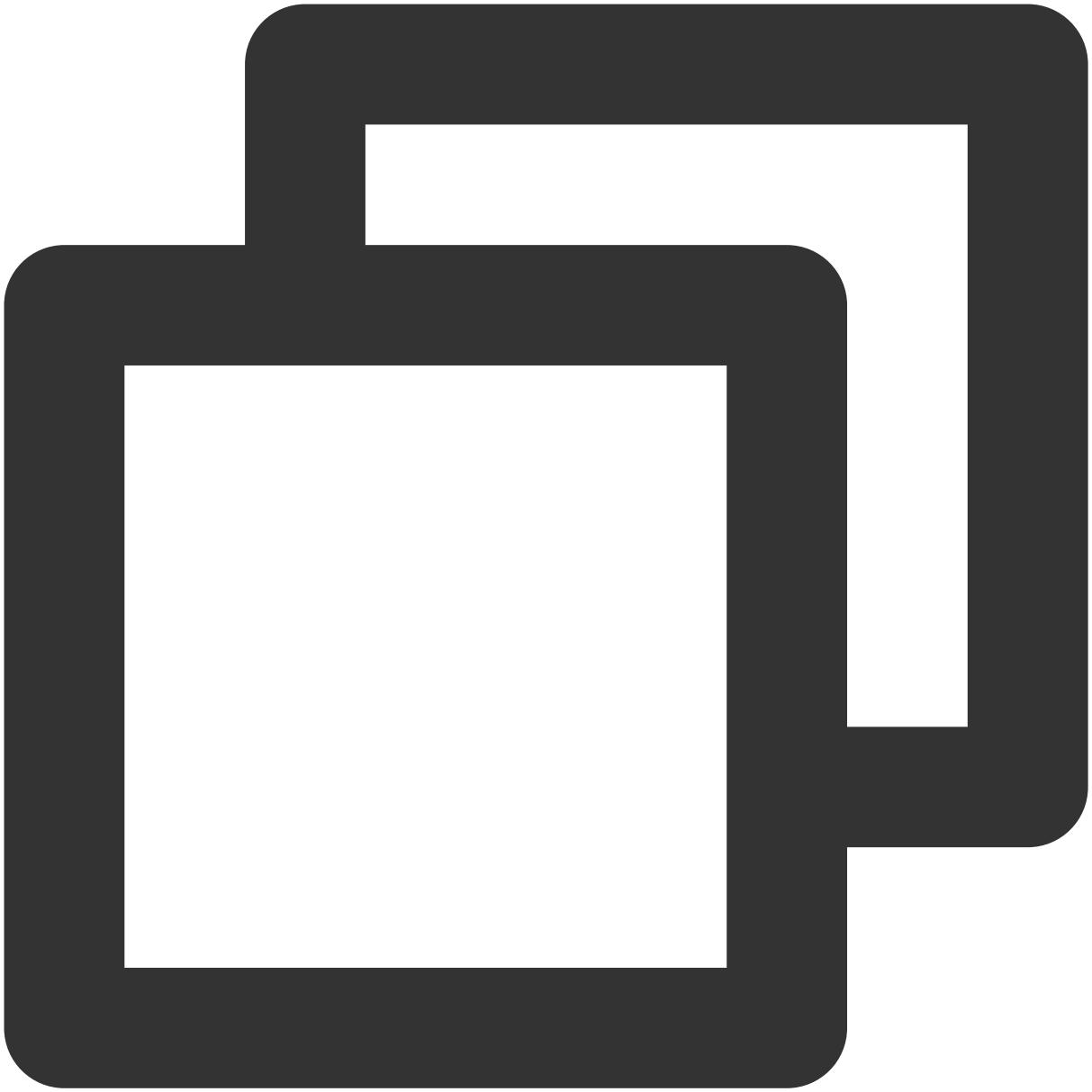
注意

使用群组通话前需要创建IM 群组，如果已经创建，请忽略。

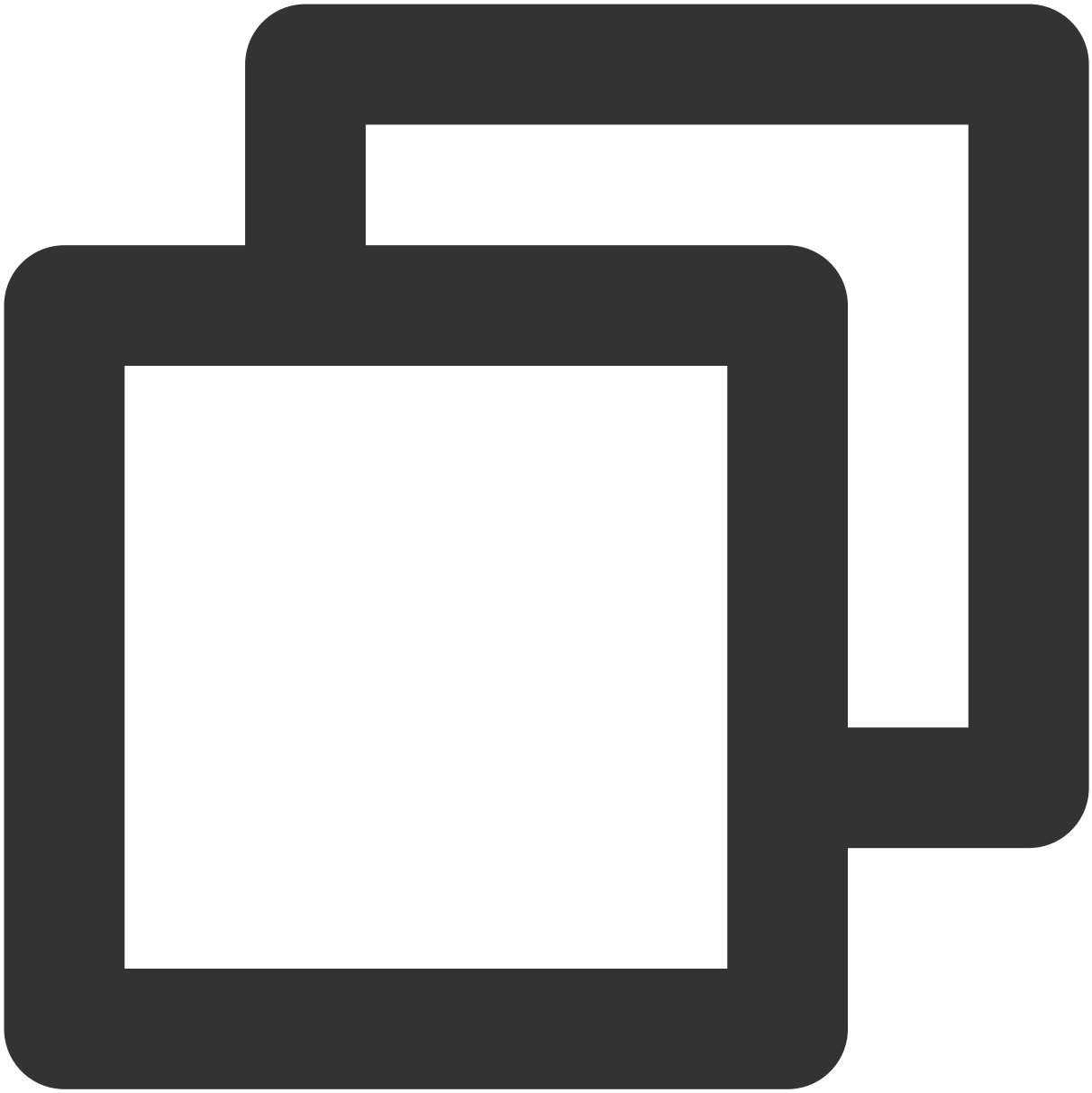
群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Kotlin

Java



```
fun groupCall(groupId: String, userIdList: List<String?>?, callMediaType: TUICallDe
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
```

参数如下表所示：

参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List	目标用户的 userId 列表
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

groupCall

发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等。

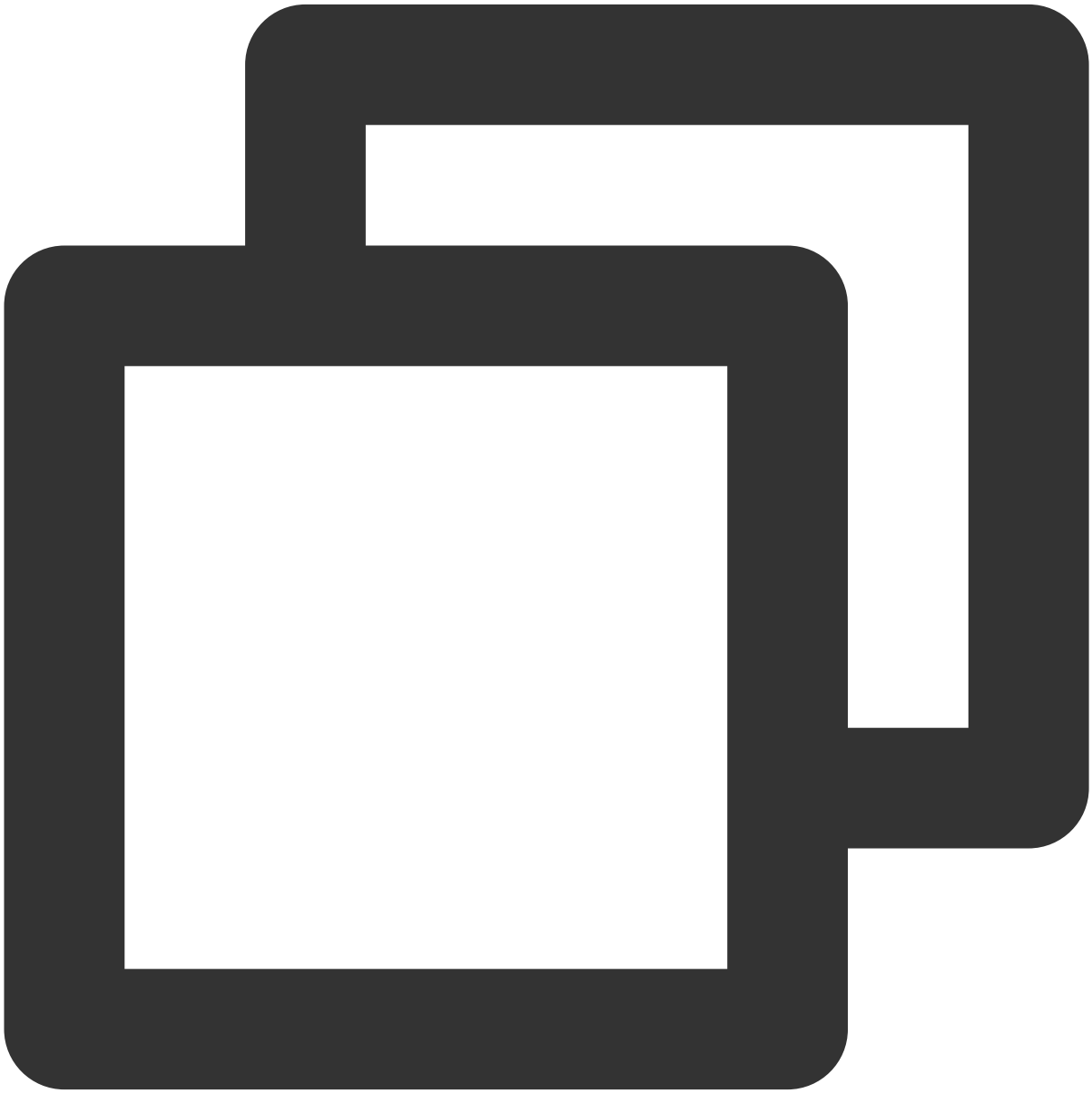
注意

使用群组通话前需要创建IM 群组，如果已经创建，请忽略。

群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

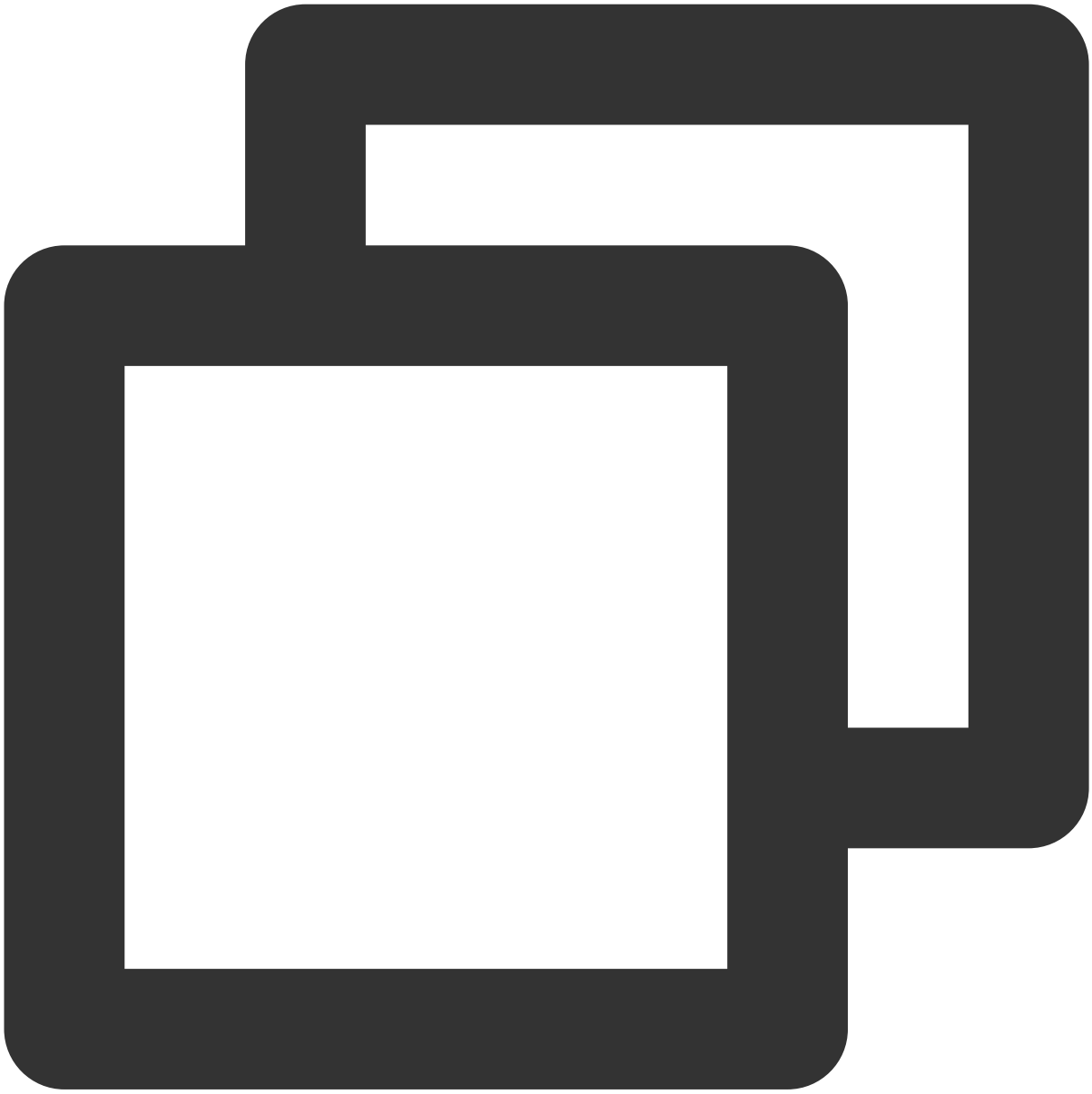
Kotlin

Java



```
fun groupCall(
    groupId: String, userIdList: List<String?>?,
```

```
    callMediaType: TUICallDefine.MediaType, params: CallParams?,
    callback: TUICommonDefine.Callback?
)
```



```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义

groupId	String	此次群组通话的群 ID
userIdList	List	目标用户的 userId 列表
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

joinInGroupCall

加入群组中已有的音视频通话。

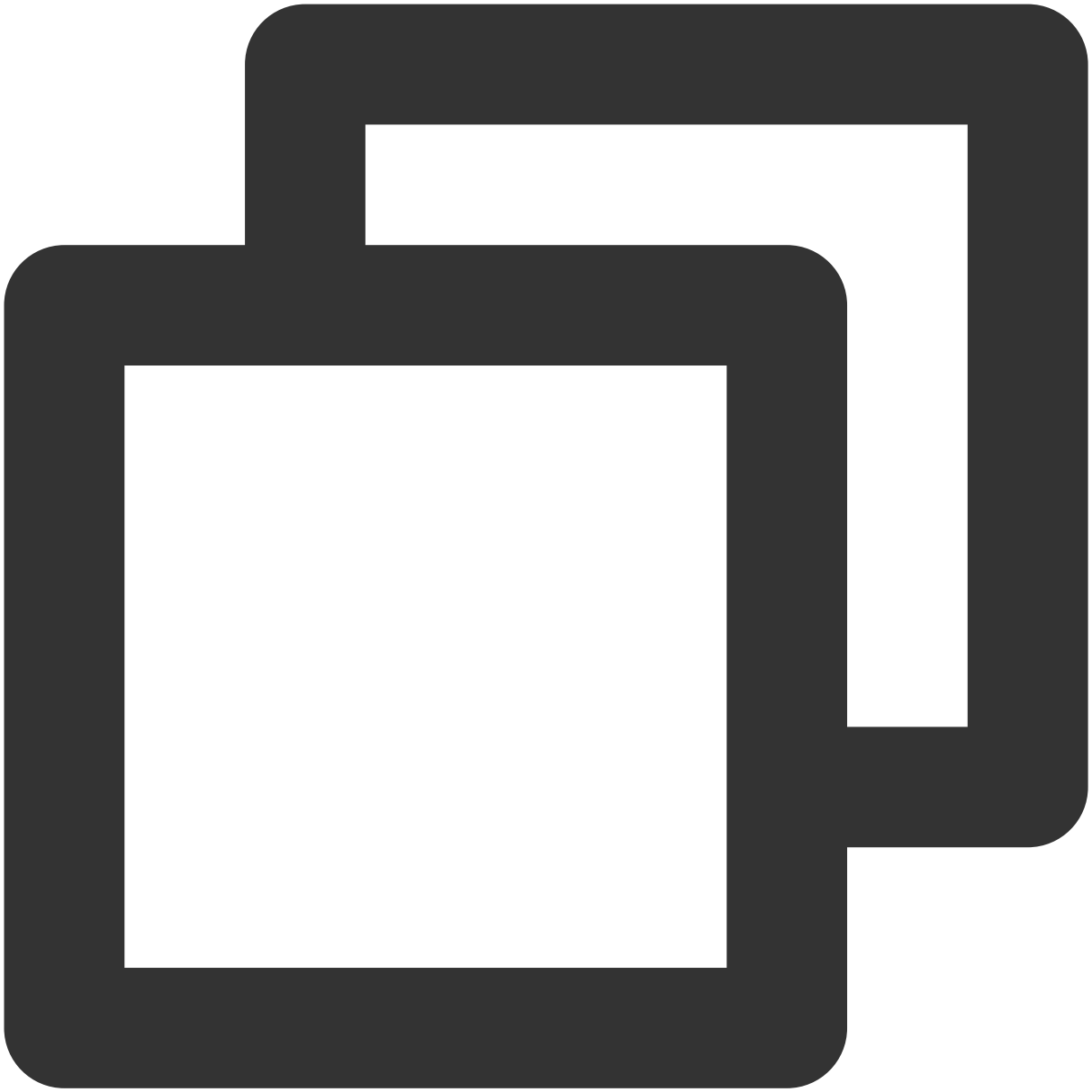
注意

加入群组中已有的音视频通话前，需要提前创建或加入IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。

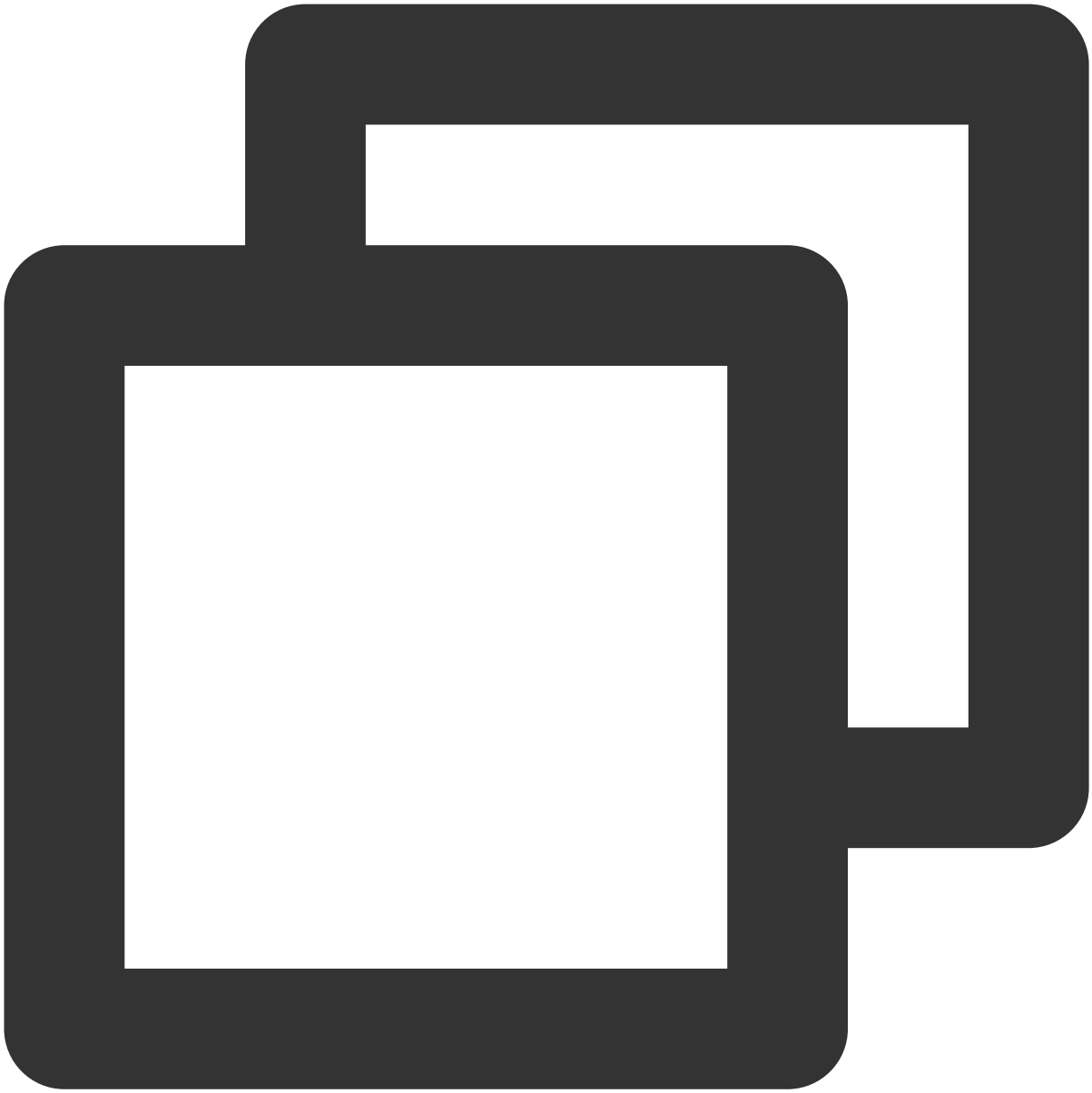
群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Kotlin

Java



```
fun joinInGroupCall(roomId: RoomId?, groupId: String?, callMediaType: TUICallDefine
```



```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId, TUICallDefine.M
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID
groupId	String	此次群组通话的群 ID
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

setCallingBell

设置自定义来电铃音。

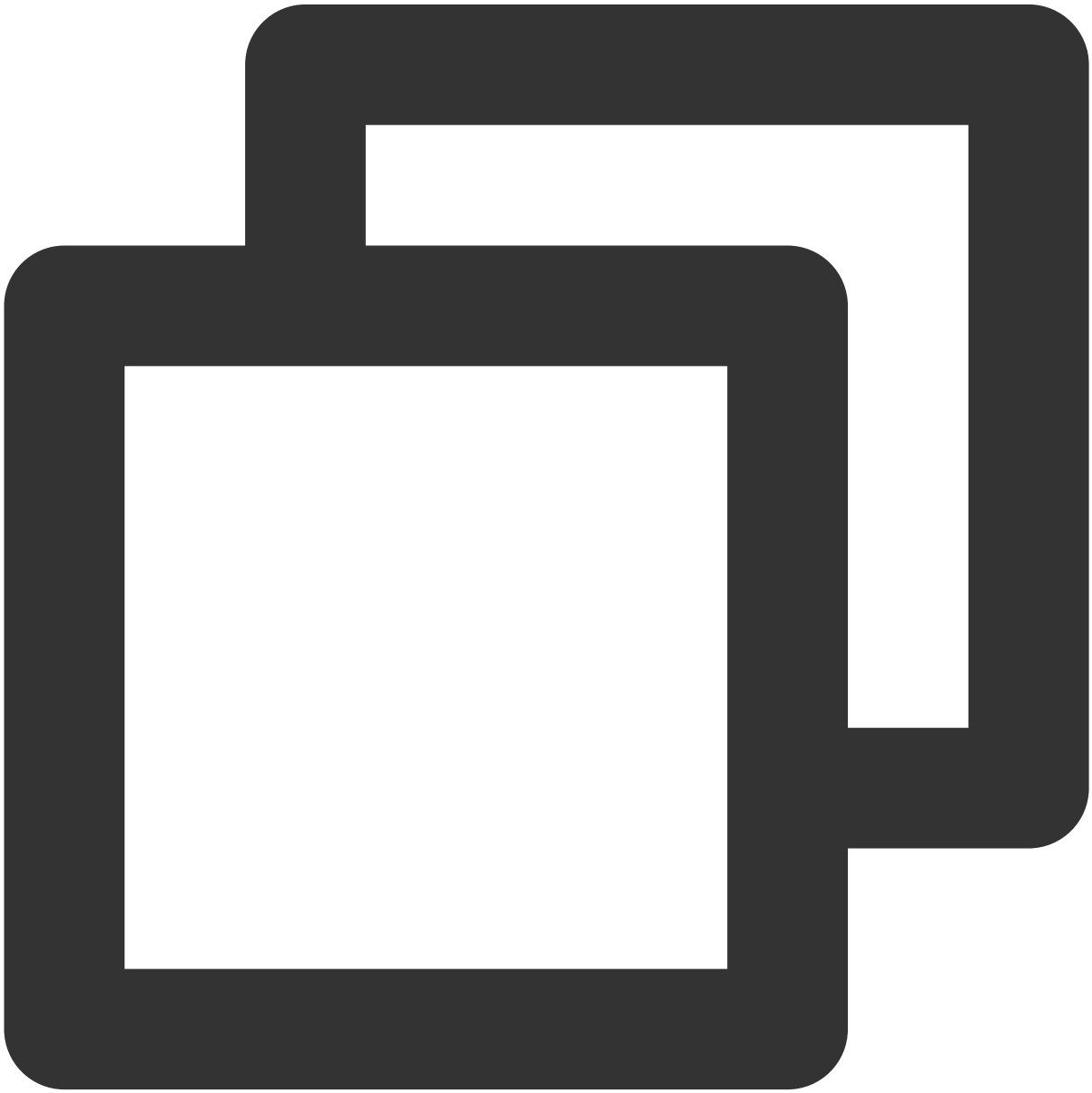
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

铃声设置后与设备绑定，更换用户，铃声依旧会生效。

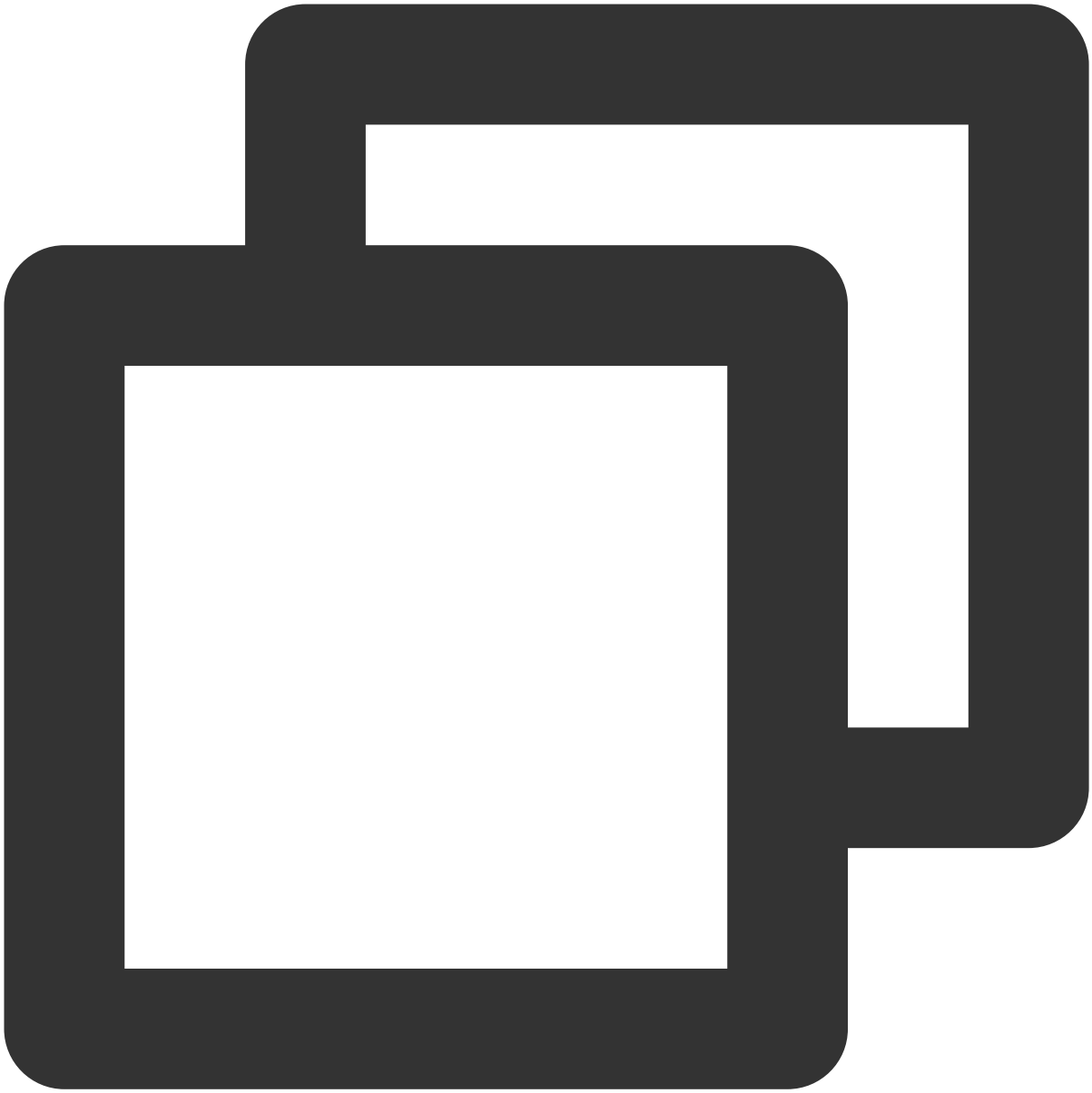
如需恢复默认铃声， `filePath` 传空即可。

Kotlin

Java



```
fun setCallingBell(filePath: String?)
```



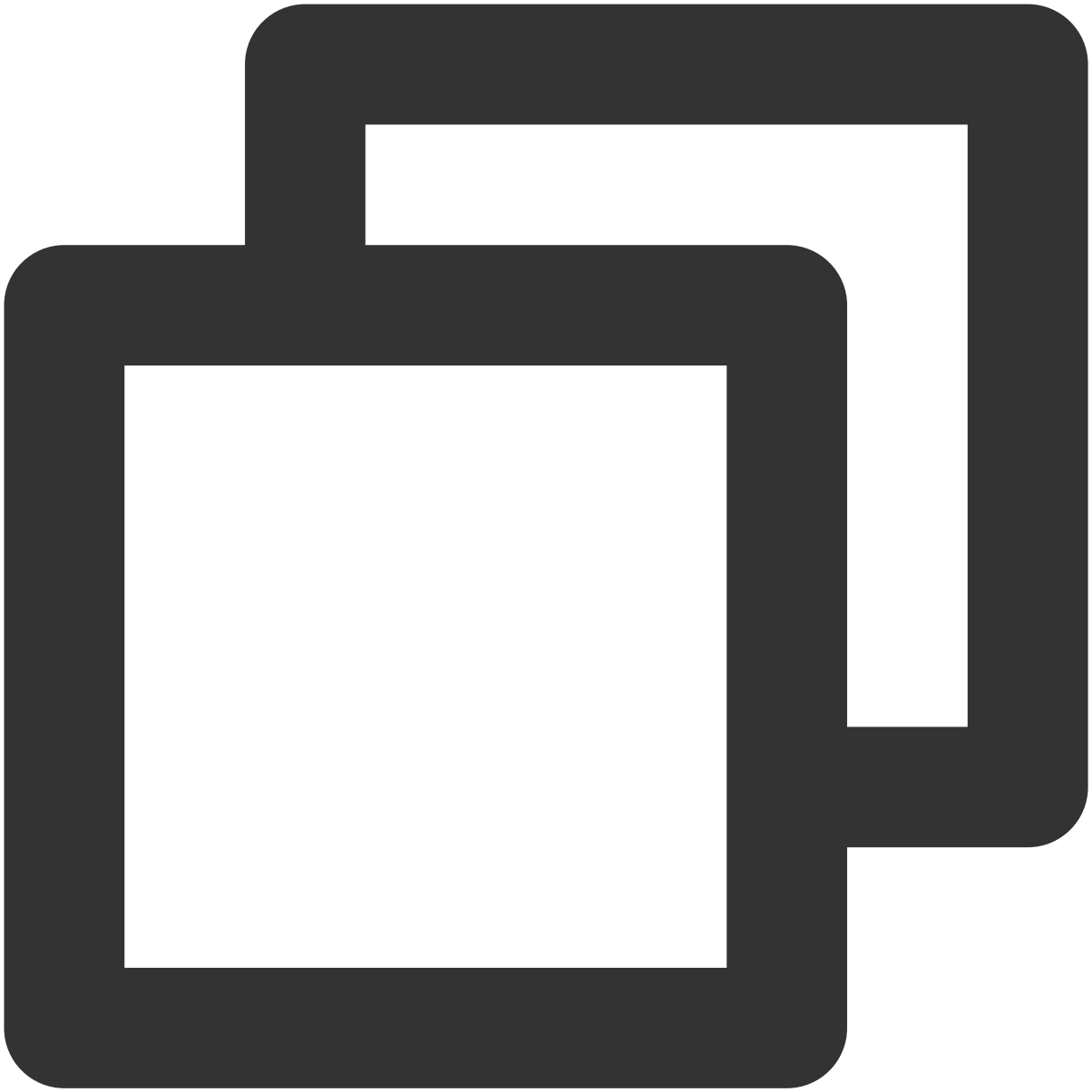
```
void setCallingBell(String filePath);
```

enableMuteMode

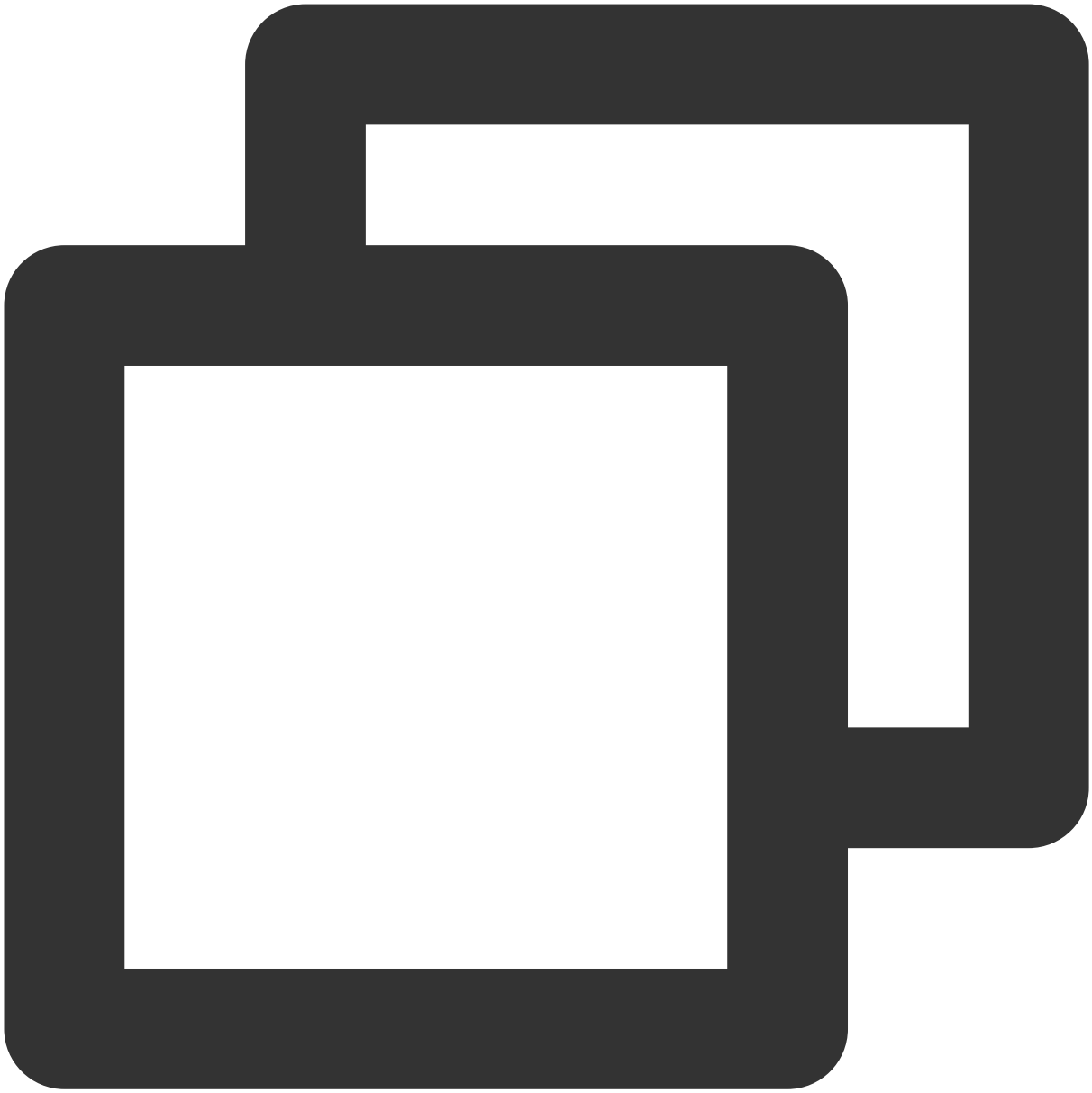
开启/关闭静音模式。

Kotlin

Java



```
fun enableMuteMode(enable: Boolean)
```



```
void enableMuteMode(boolean enable);
```

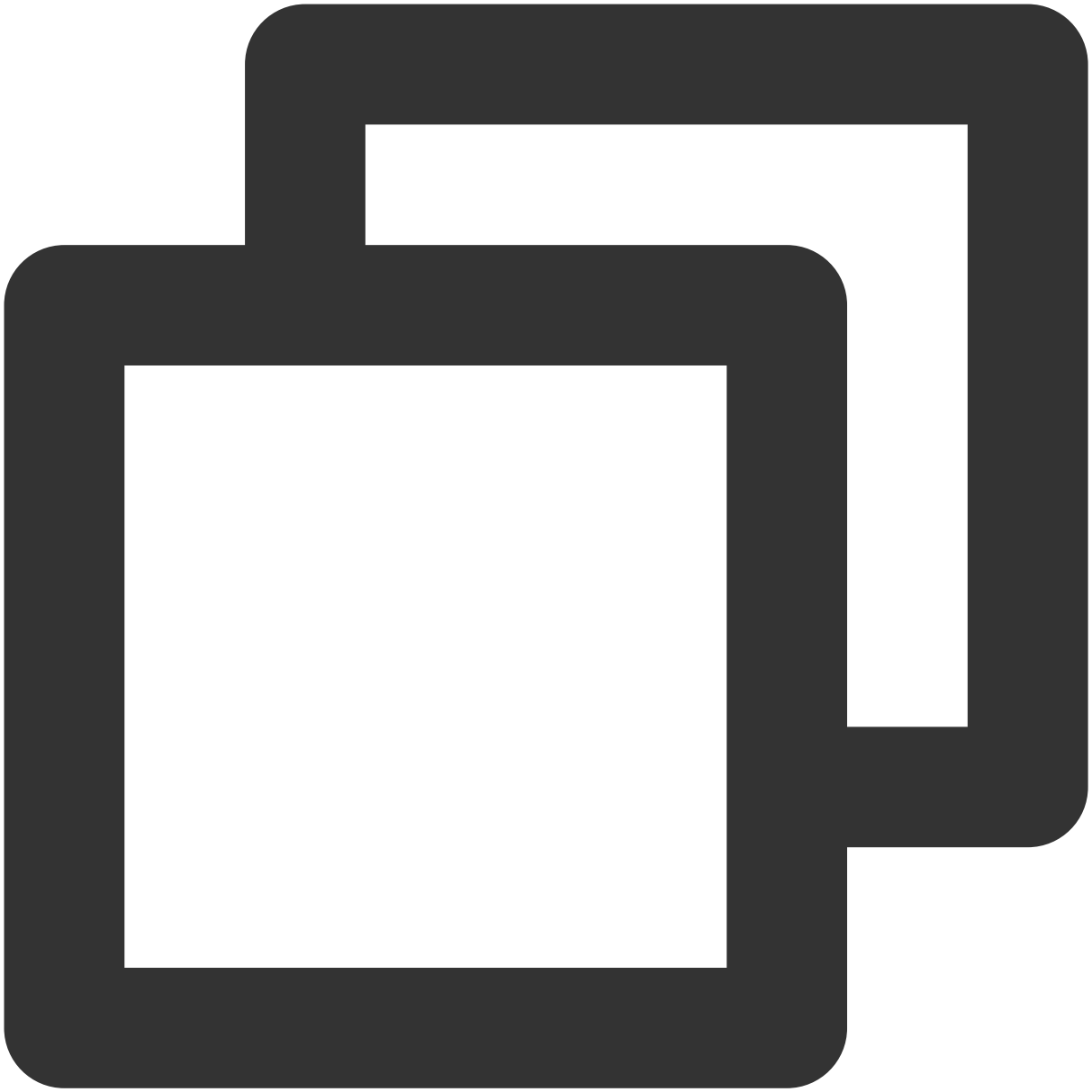
enableFloatWindow

开启/关闭悬浮窗功能。

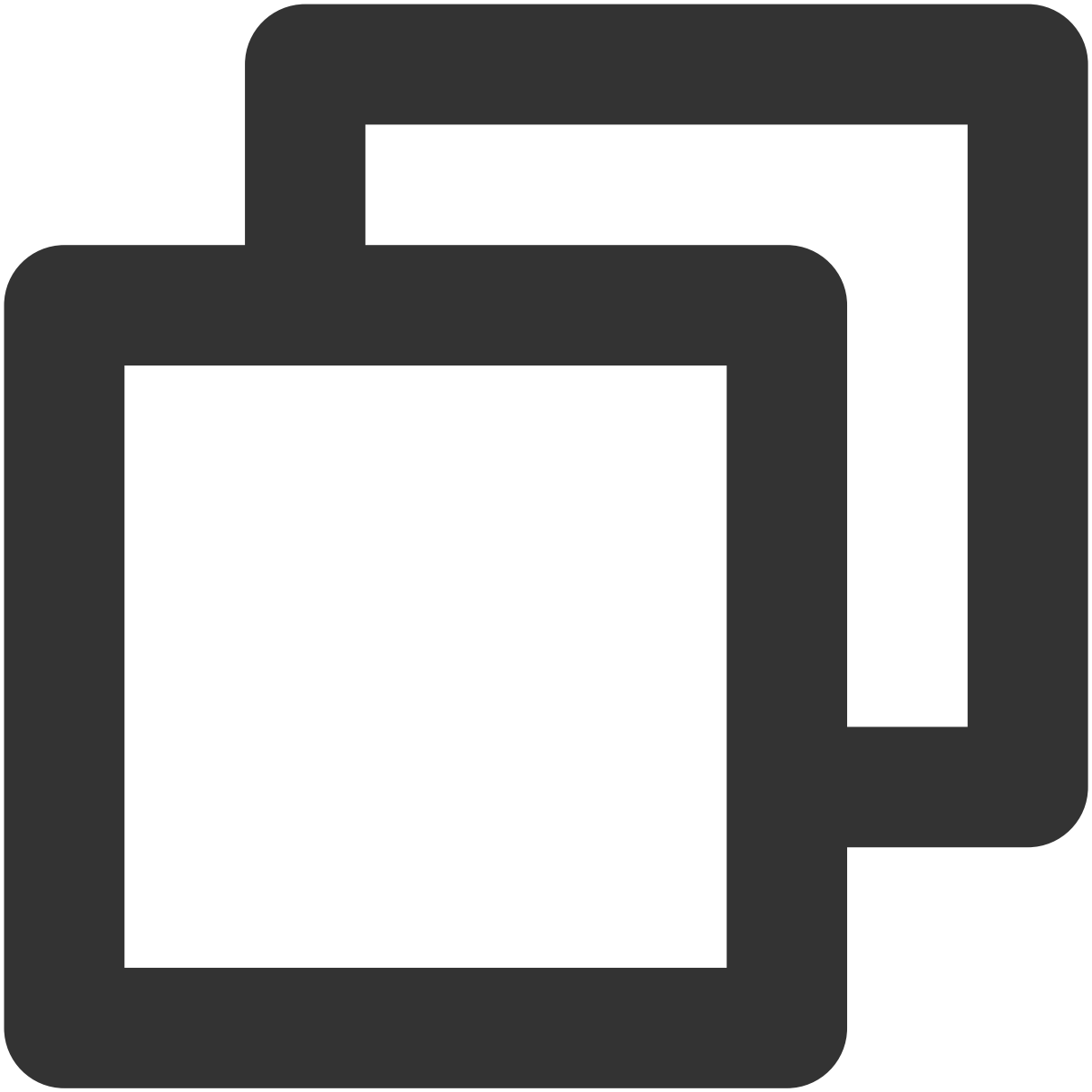
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Kotlin

Java



```
fun enableFloatWindow(enable: Boolean)
```



```
void enableFloatWindow(boolean enable);
```

TUICallEngine

最近更新时间：2023-06-28 11:41:15

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流

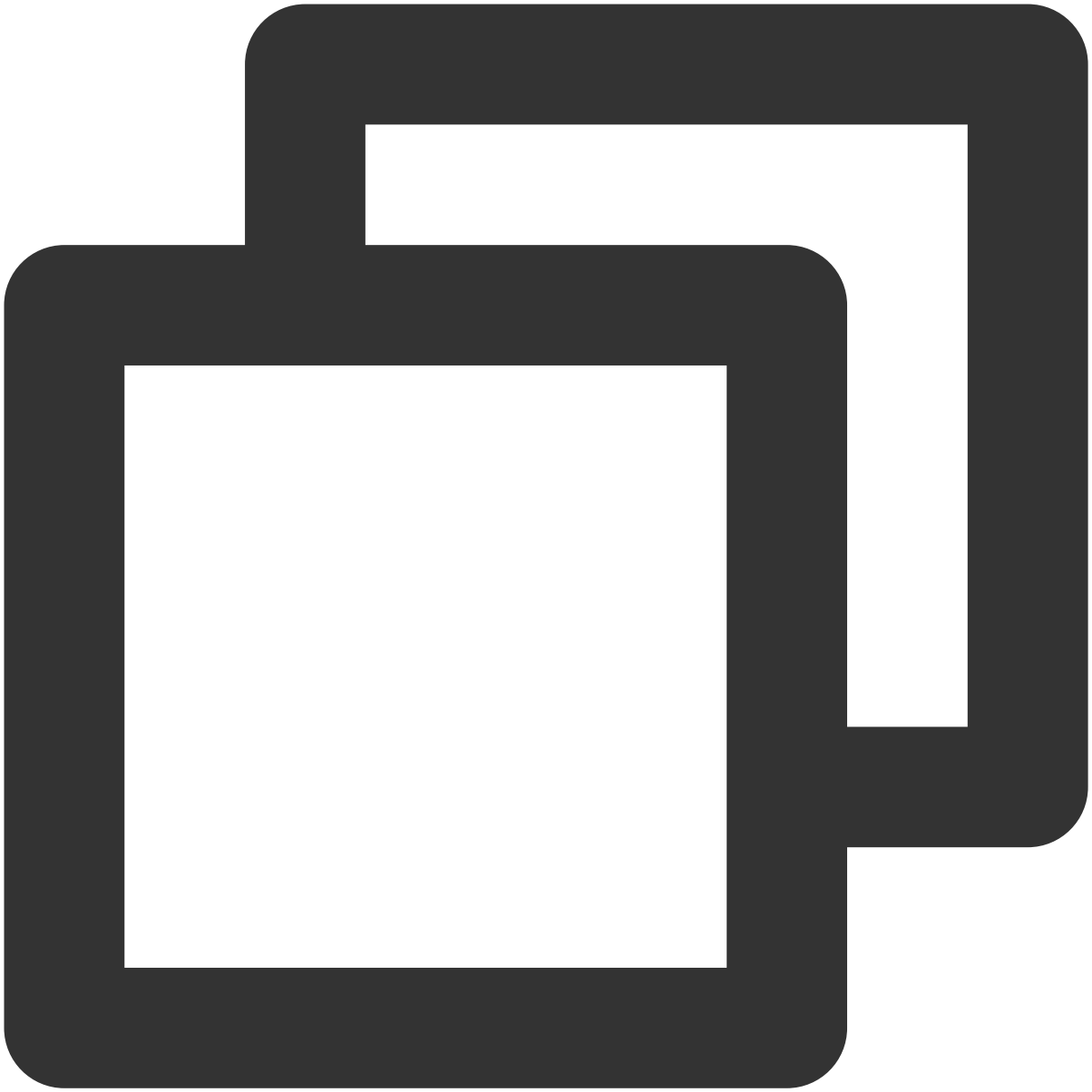
版权所有：腾讯云计算（北京）有限责任公司

openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

API 详情

createInstance

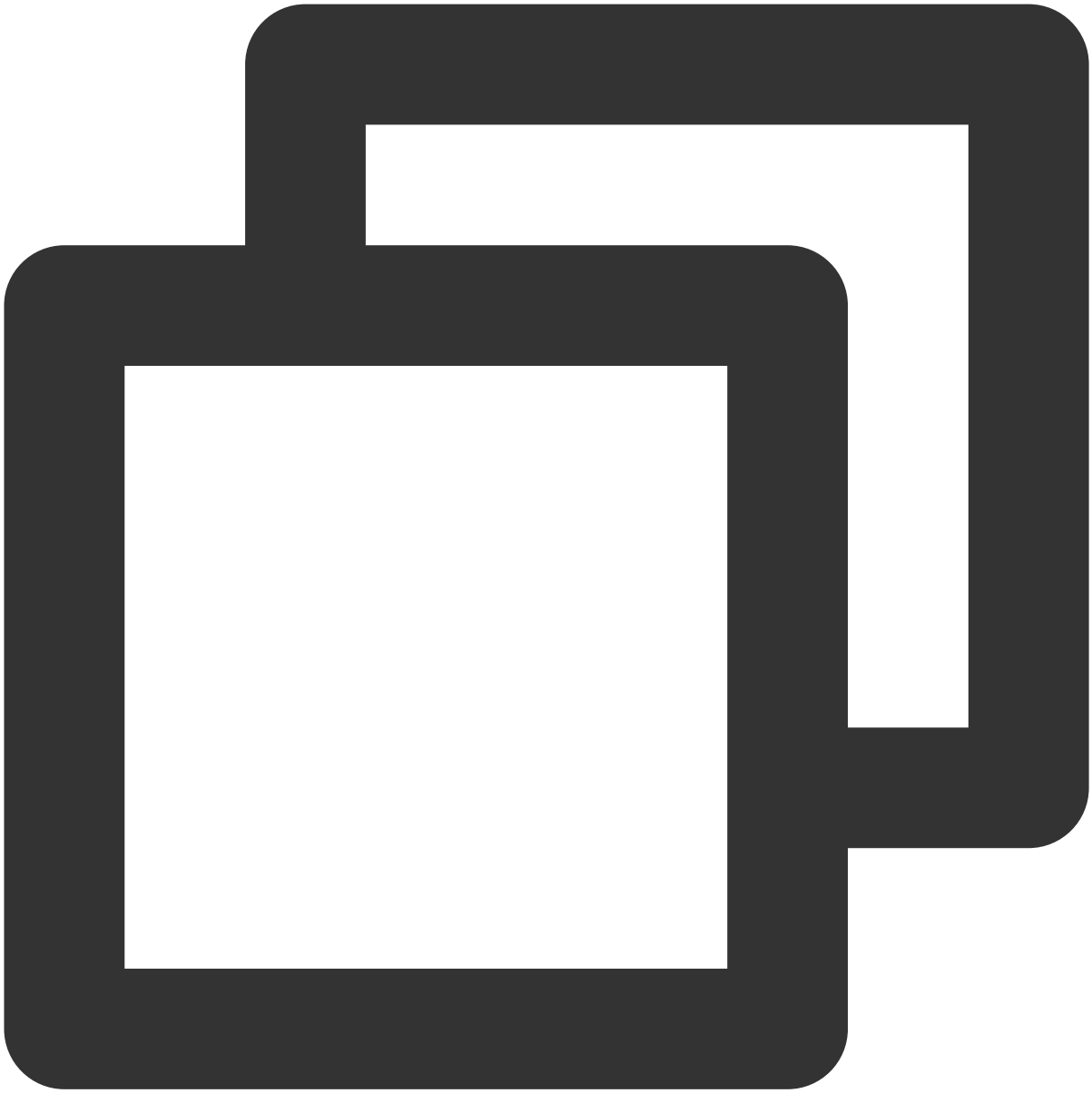
创建 TUICallEngine 的单例。



```
TUICallEngine createInstance(Context context);
```

destroyInstance

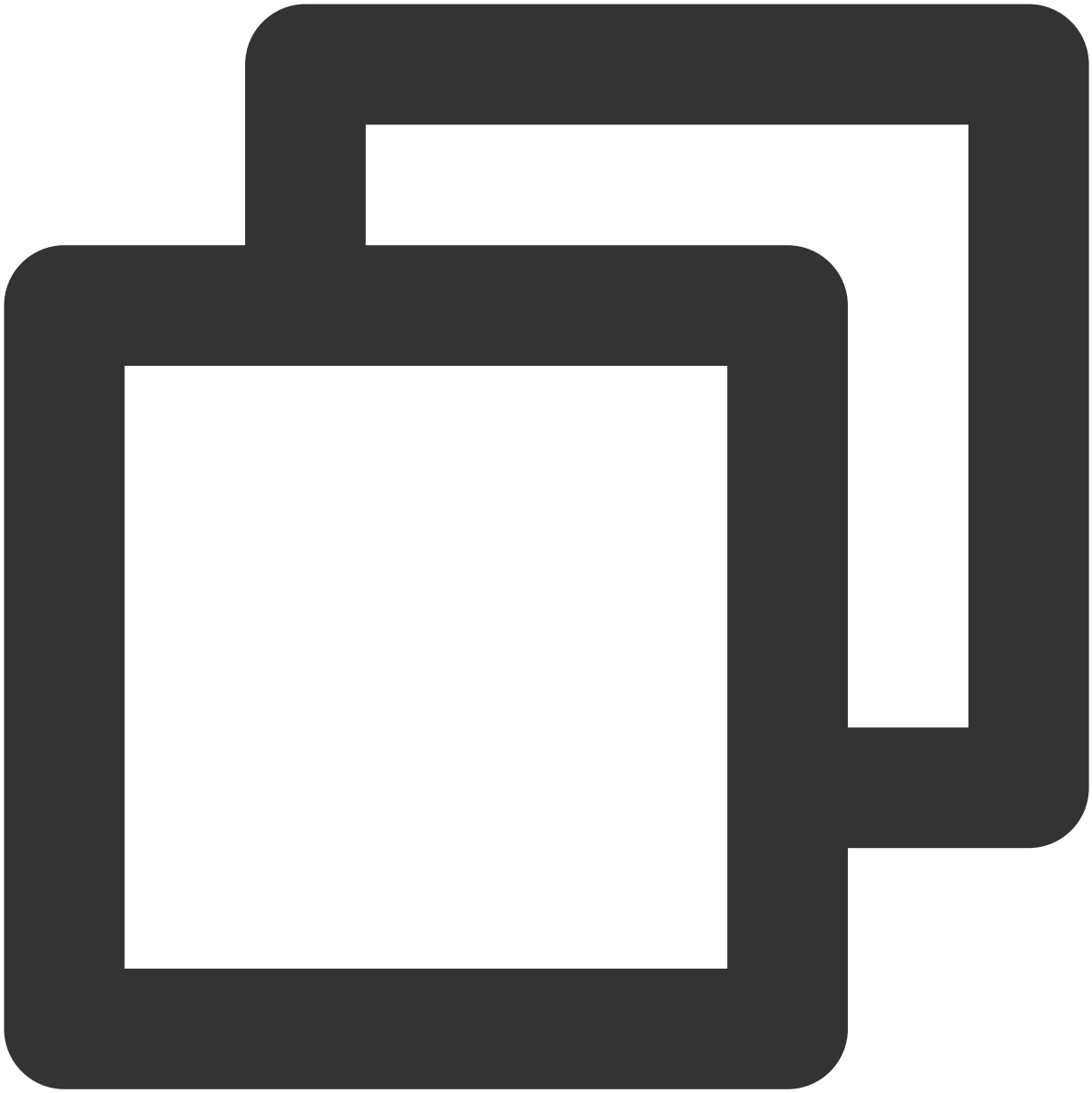
销毁 TUICallEngine 的单例。



```
void destroyInstance();
```

init

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。



```
void init(int sdkAppId, String userId, String userSig, TUICommonDefine.Callback cal
```

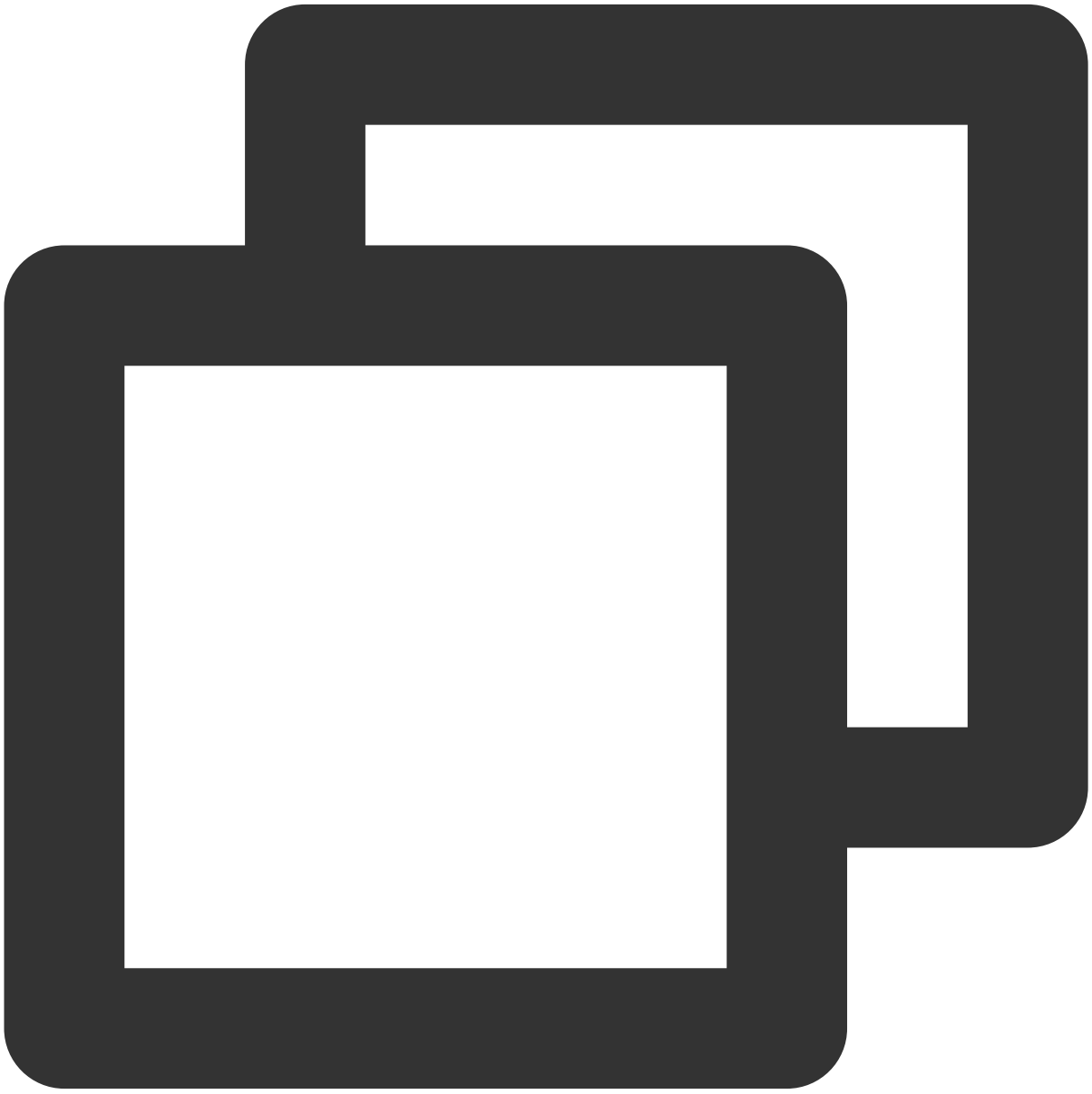
参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在**实时音视频控制台** > **应用管理** > **应用信息**中查看 SDKAppID
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）

userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig
callback	TUICommonDefine.Callback	初始化回调，`onSuccess` 表示初始化成功

addObserver

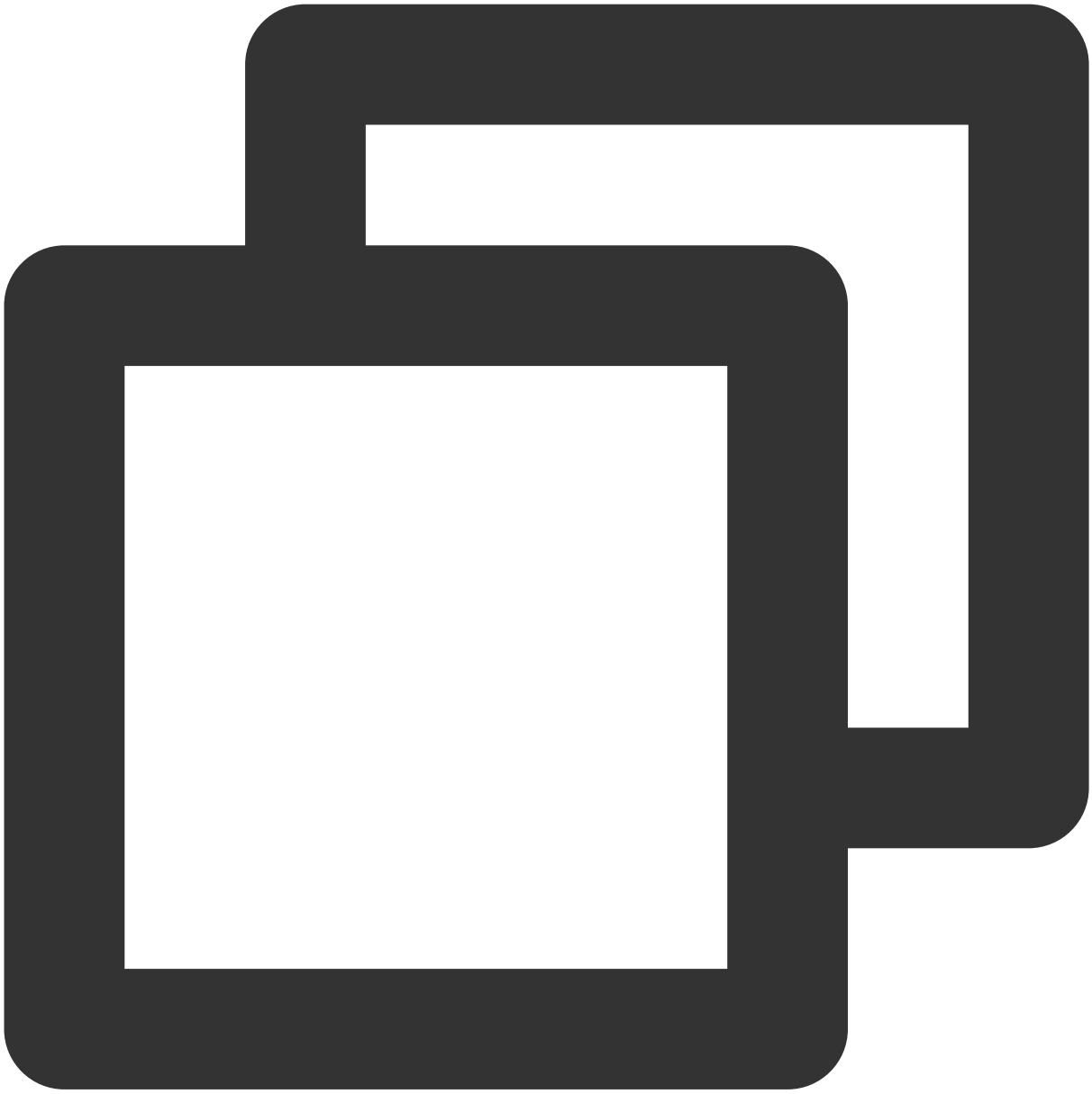
添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。



```
void addObserver(TUICallObserver observer);
```

removeObserver

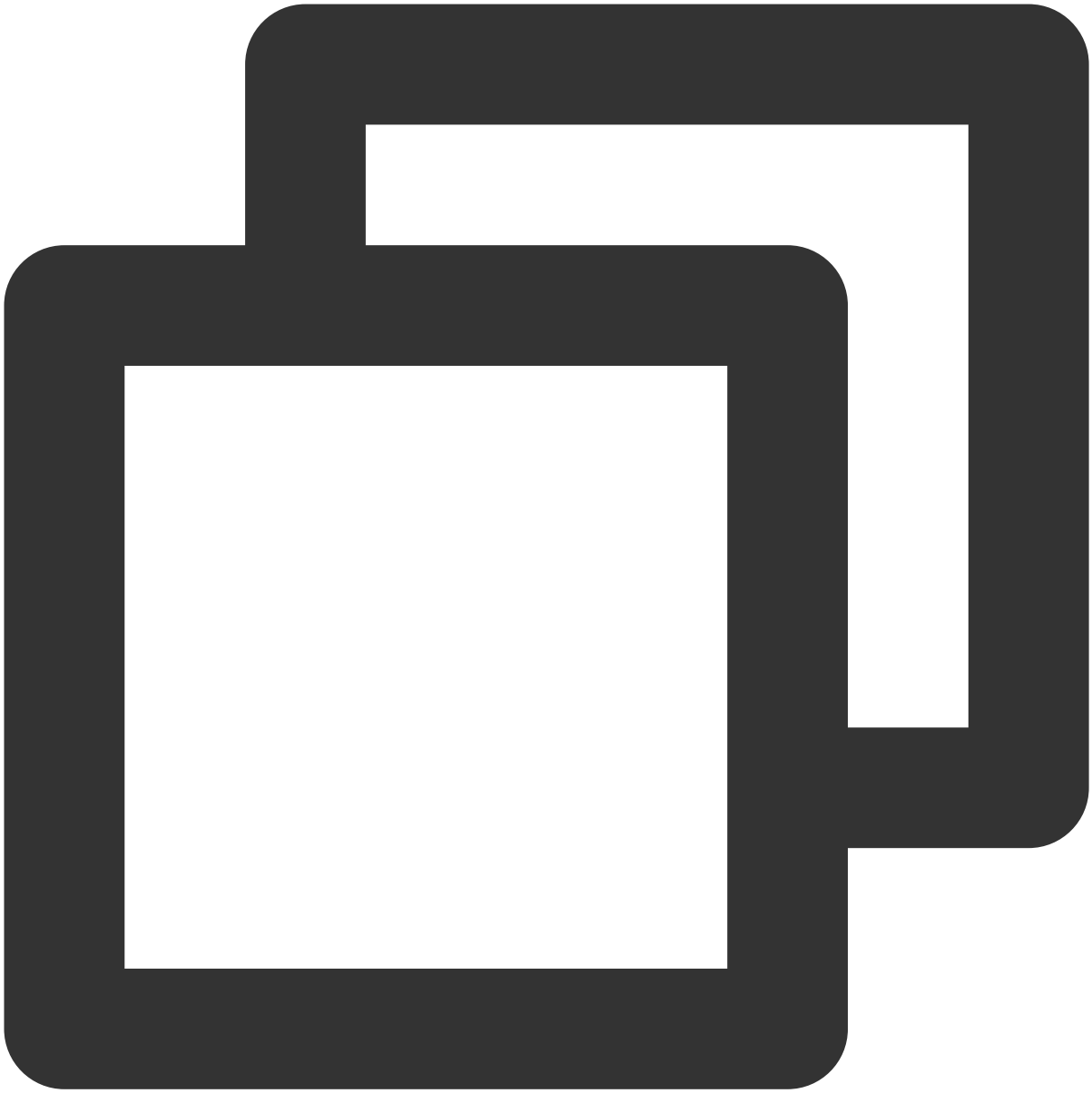
移除回调接口。



```
void removeObserver(TUICallObserver observer);
```

call

拨打电话（1v1通话）。



```
void call(String userId, TUICallDefine.MediaType callMediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
callMediaType	TUICallDefine.MediaType	通话的媒体类型，例如：视频通话、语音通话

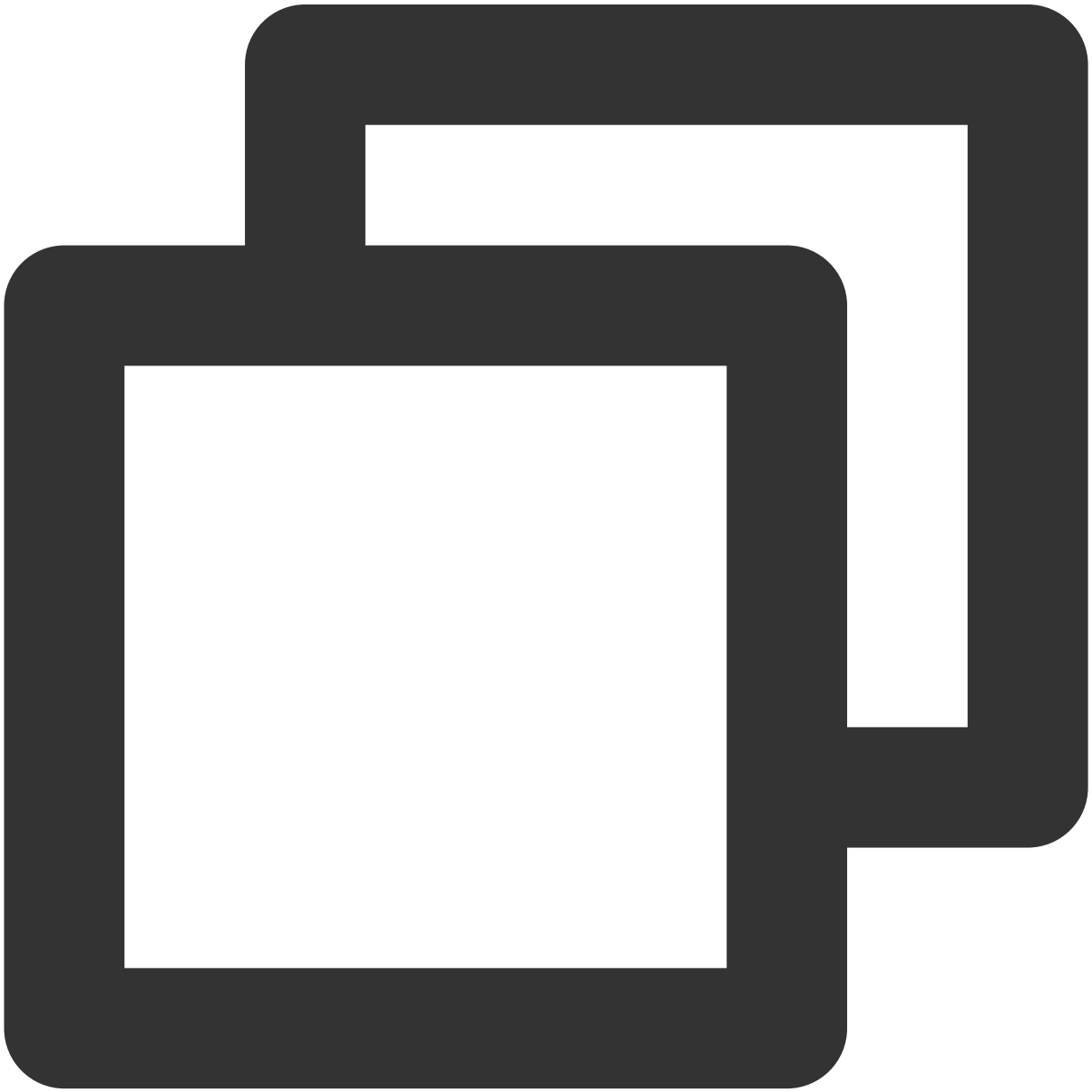
params	TUICallDefine.CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等

groupCall

发起群组通话。

注意：

使用群组通话前需要创建IM 群组，如果已经创建，请忽略。



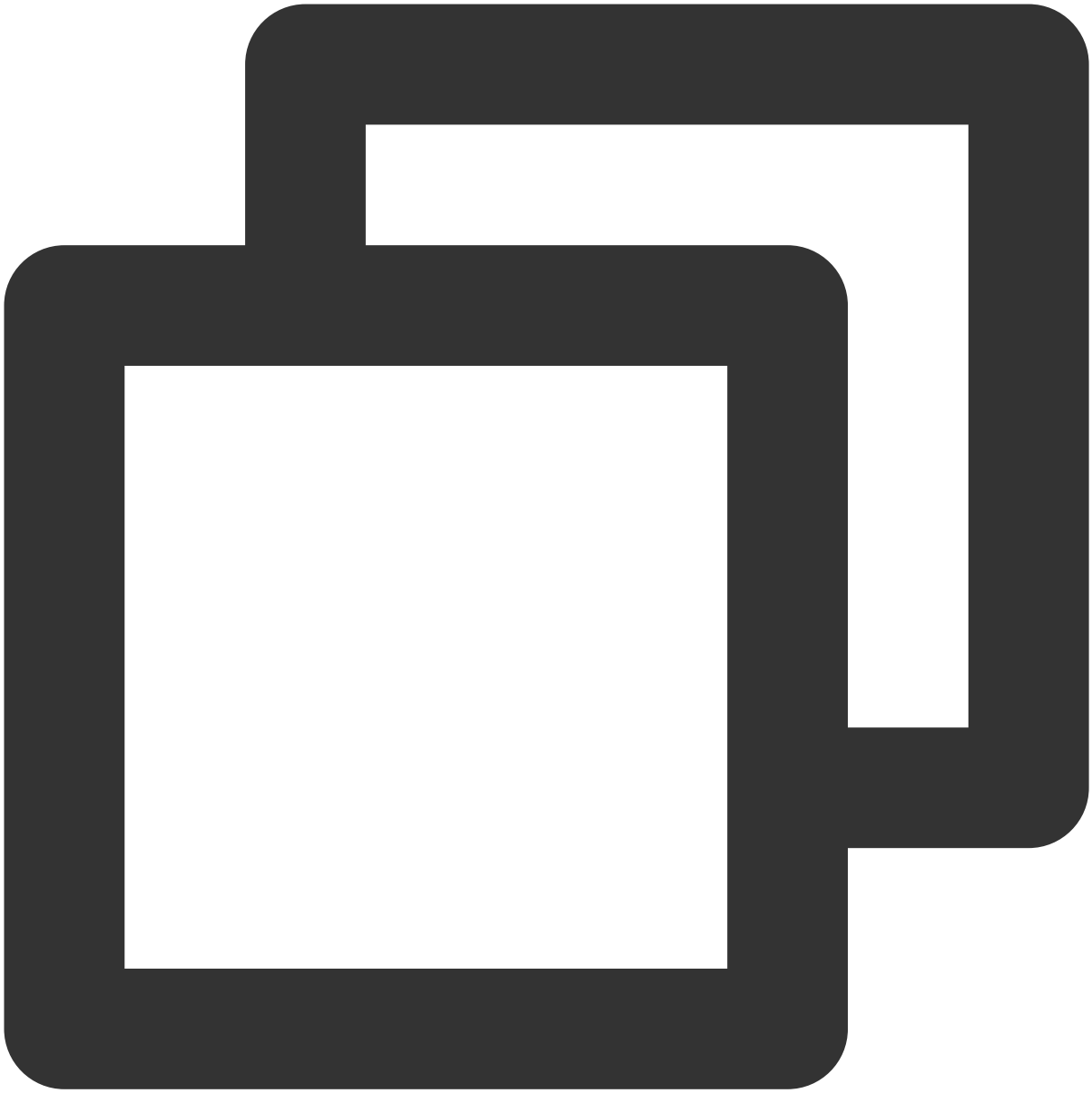
```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List	目标用户的 userId 列表
callMediaType	TUICallDefine.MediaType	通话的媒体类型，例如：视频通话、语音通话
params	TUICallDefine.CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等

accept

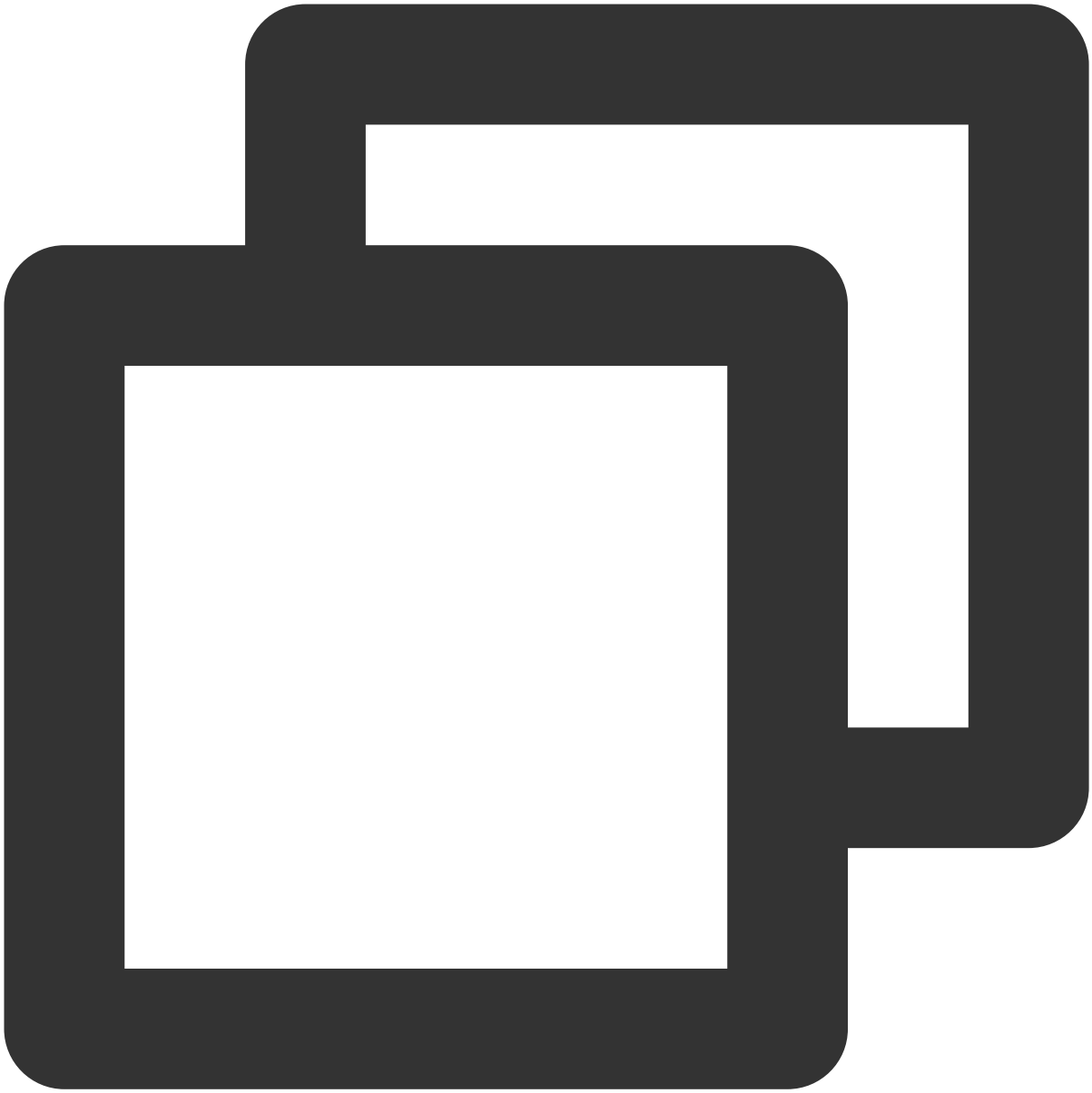
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
void accept(TUICommonDefine.Callback callback);
```

reject

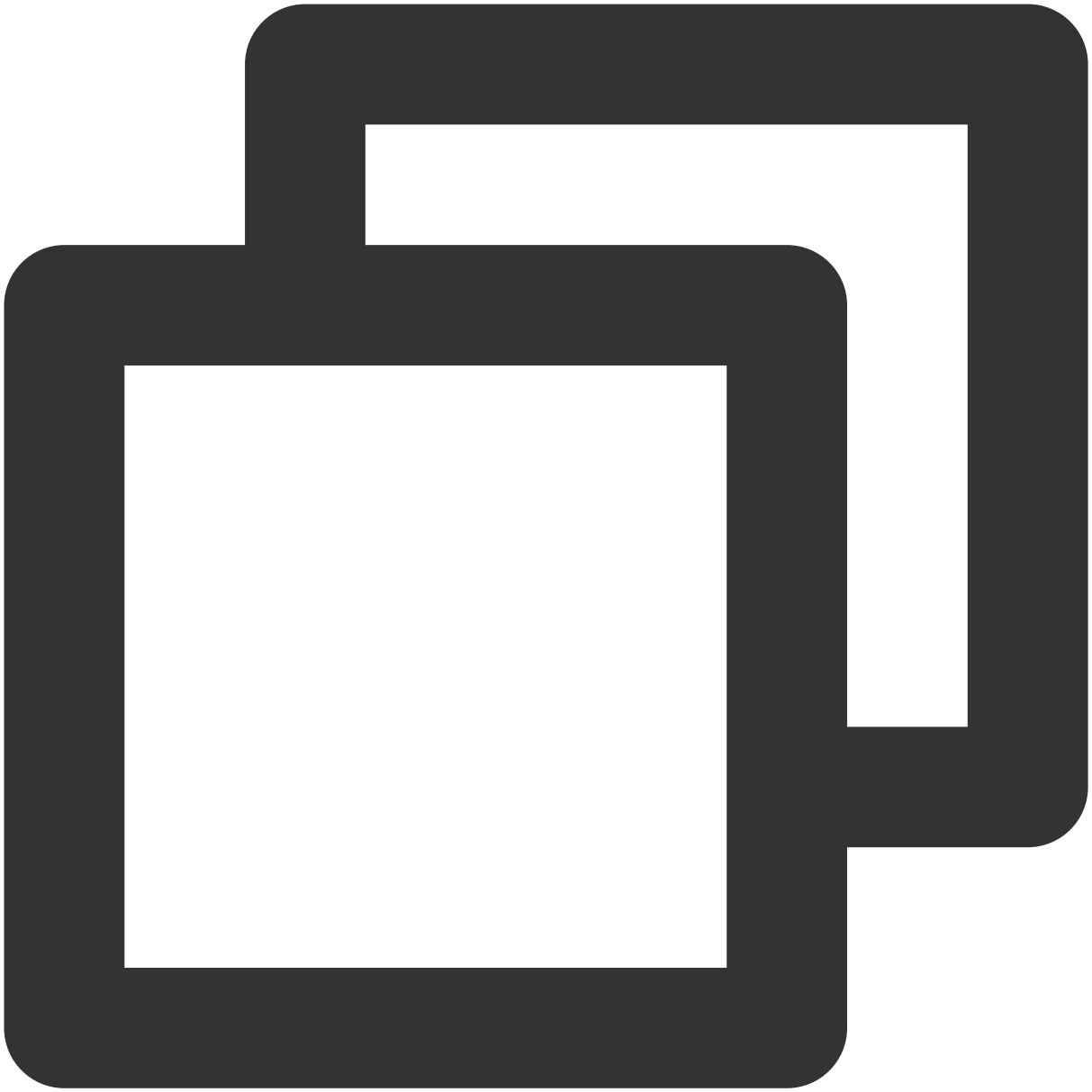
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



```
void reject(TUICommonDefine.Callback callback);
```

ignore

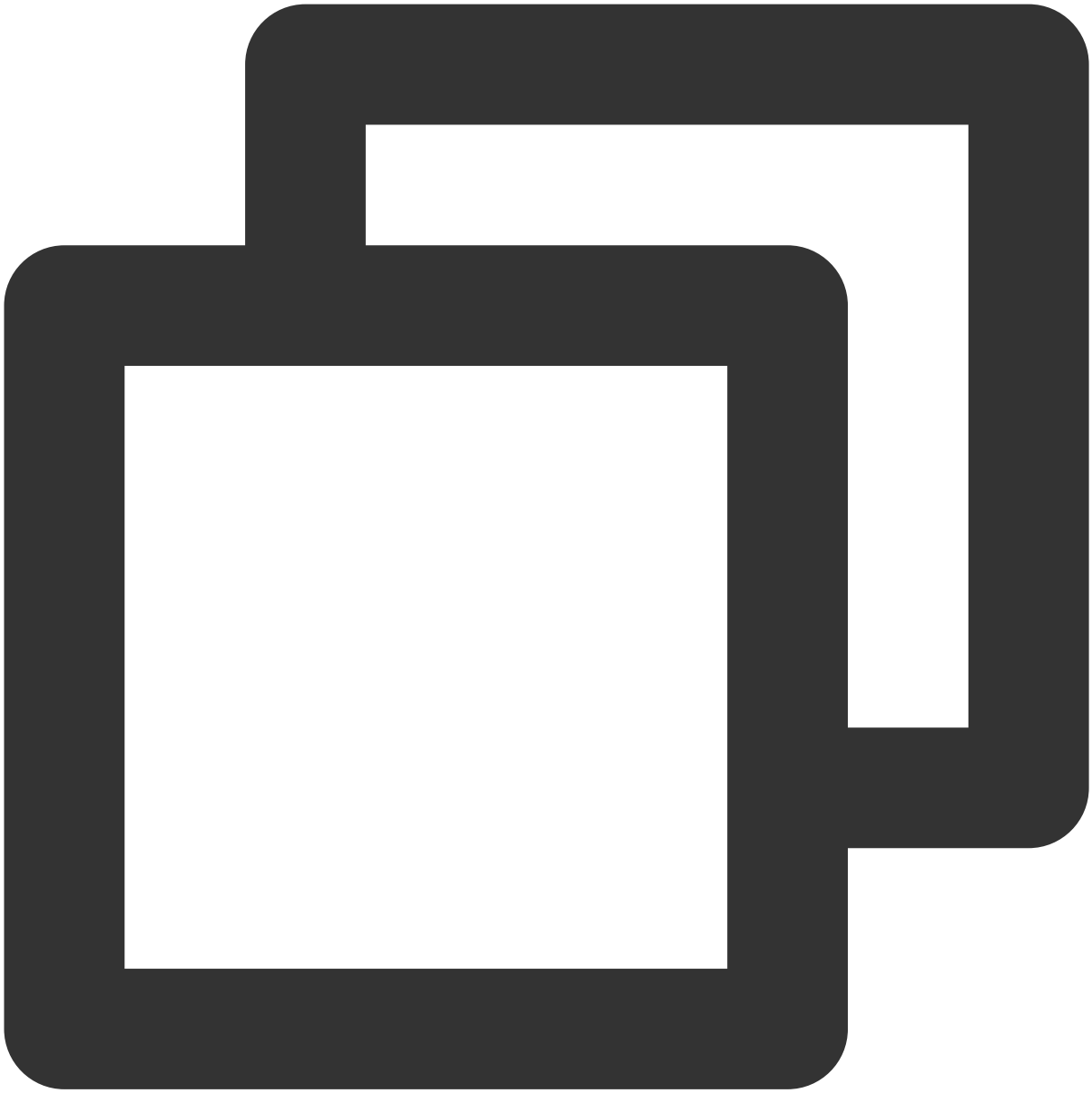
忽略当前通话，当您作为被叫收到 onCallReceived() 的回调时，可以调用该函数忽略来电，此时主叫会收到 onUserLineBusy 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
void ignore(TUICommonDefine.Callback callback);
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

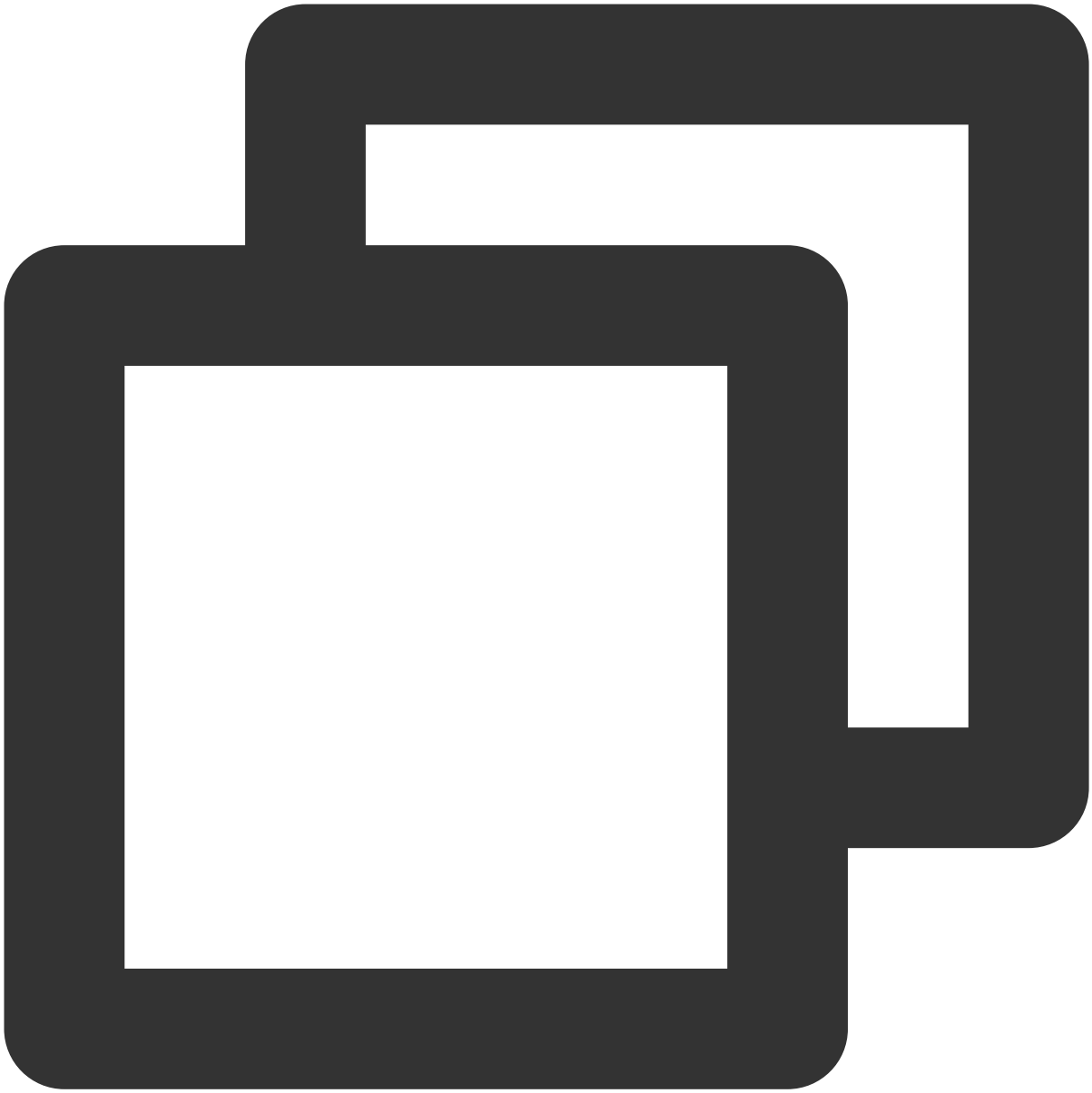


```
void hangup(TUICommonDefine.Callback callback);
```

inviteUser

邀请用户加入此次群组通话。

使用场景：一个群组通话中的用户主动邀请其他人时使用。



```
void inviteUser(List<String> userIdList, TUICallDefine.CallParams params,
                TUICommonDefine.ValueCallback callback);
```

参数如下表所示：

参数	类型	含义
userIdList	List	目标用户的 userId 列表
params	TUICallDefine.CallParams	通话参数扩展字段，例如：通话邀请超时时间，离线推送自定义内容等

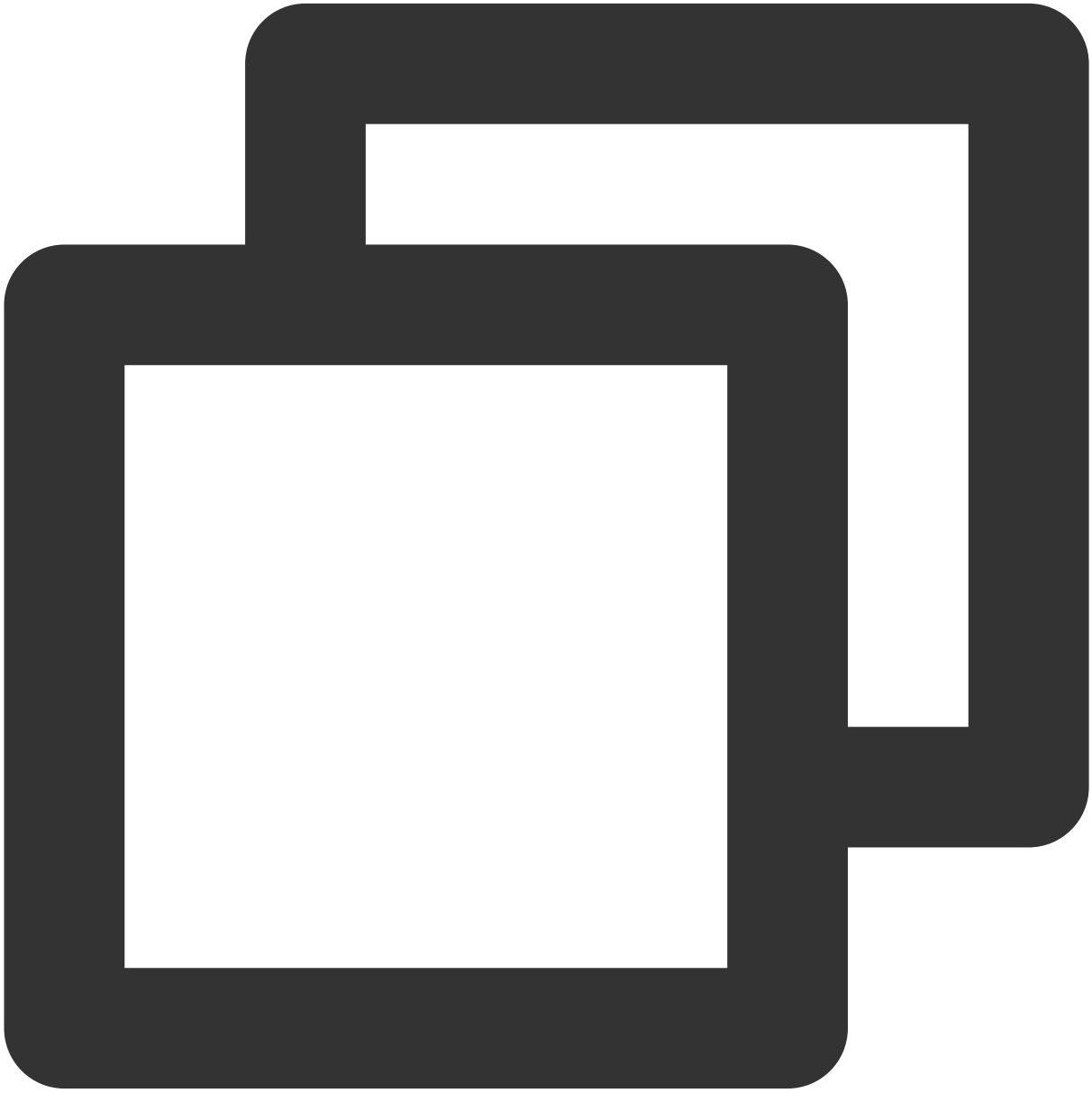
注意：

邀请他人进入当前通话，该场景下，自定义RoomId无效，SDK会邀请他人加入当前邀请者所在的房间。

joinInGroupCall

主动加入此次群组通话。

使用场景：群组内用户主动加入此次群组通话使用。



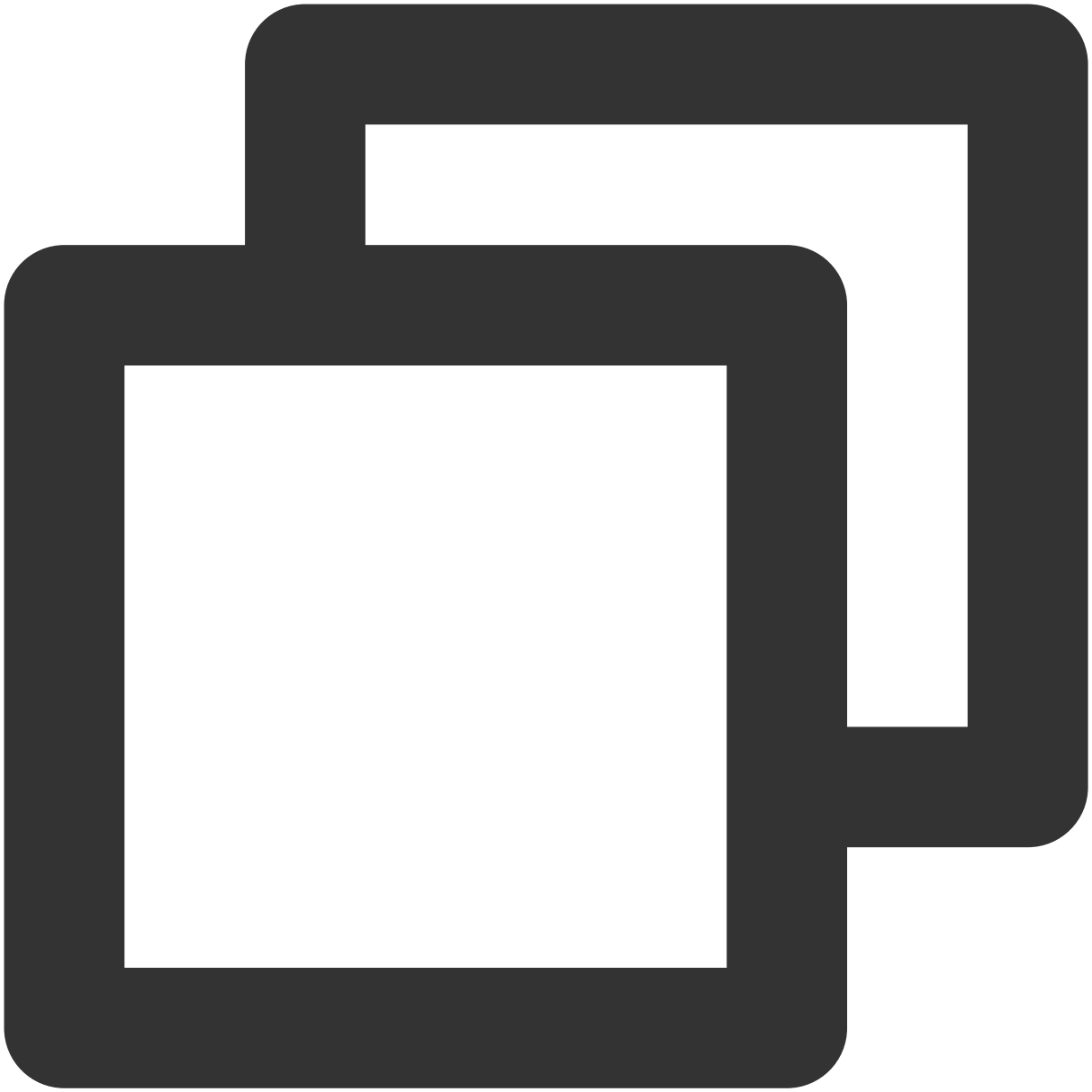
```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId,
                     TUICallDefine.MediaType callMediaType, TUICommonDefine.Callbac
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID
groupId	String	此次群组通话的群 ID
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

switchCallMediaType

切换视频通话到语音通话。



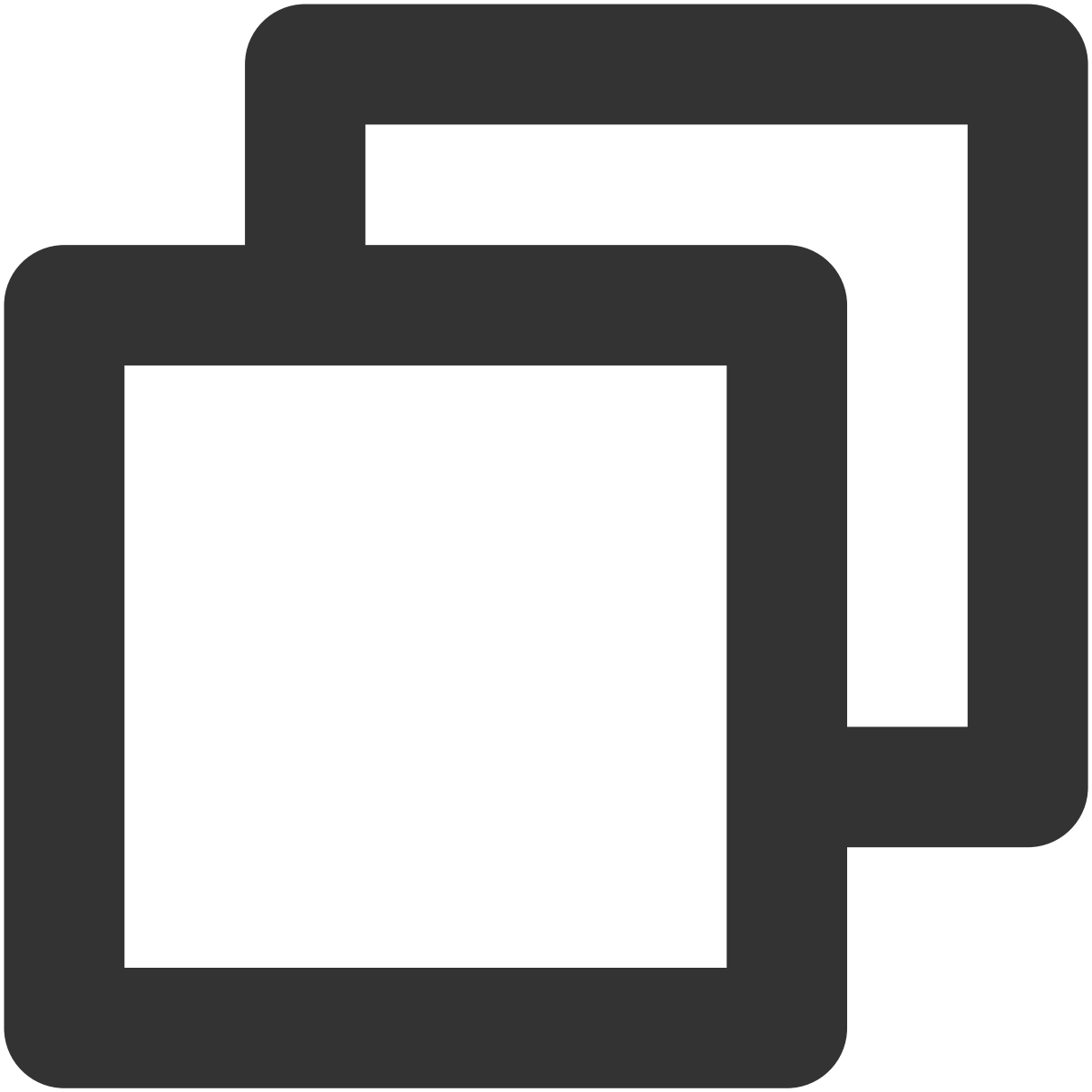
```
void switchCallMediaType(TUICallDefine.MediaType callMediaType);
```

参数如下表所示：

参数	类型	含义
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

startRemoteView

开始订阅远端用户的视频数据。



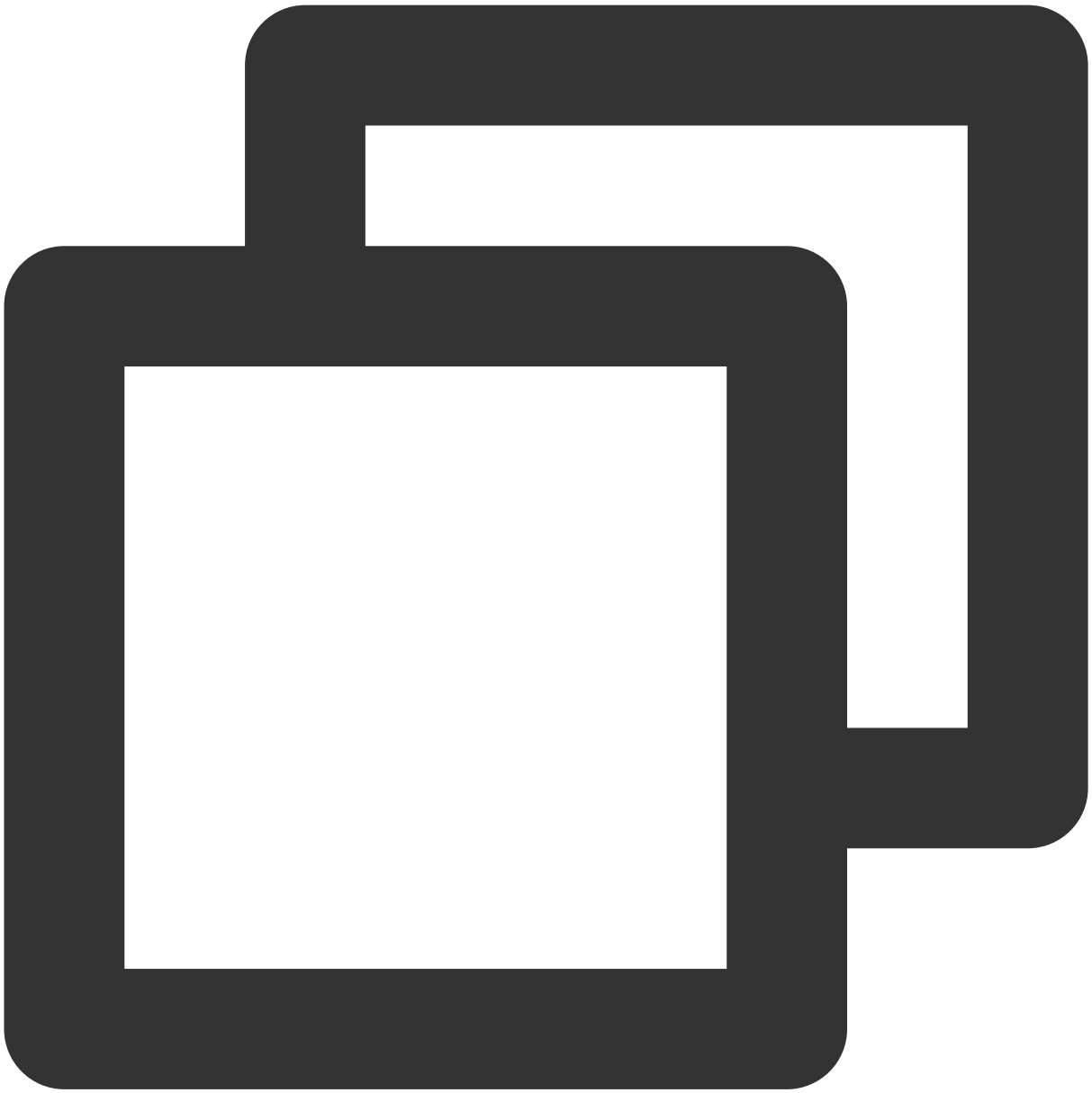
```
void startRemoteView(String userId, TUIVideoView videoView, TUICommonDefine.PlayCal
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
videoView	TUIVideoView	待渲染的视图

stopRemoteView

停止订阅远端用户的视频数据。



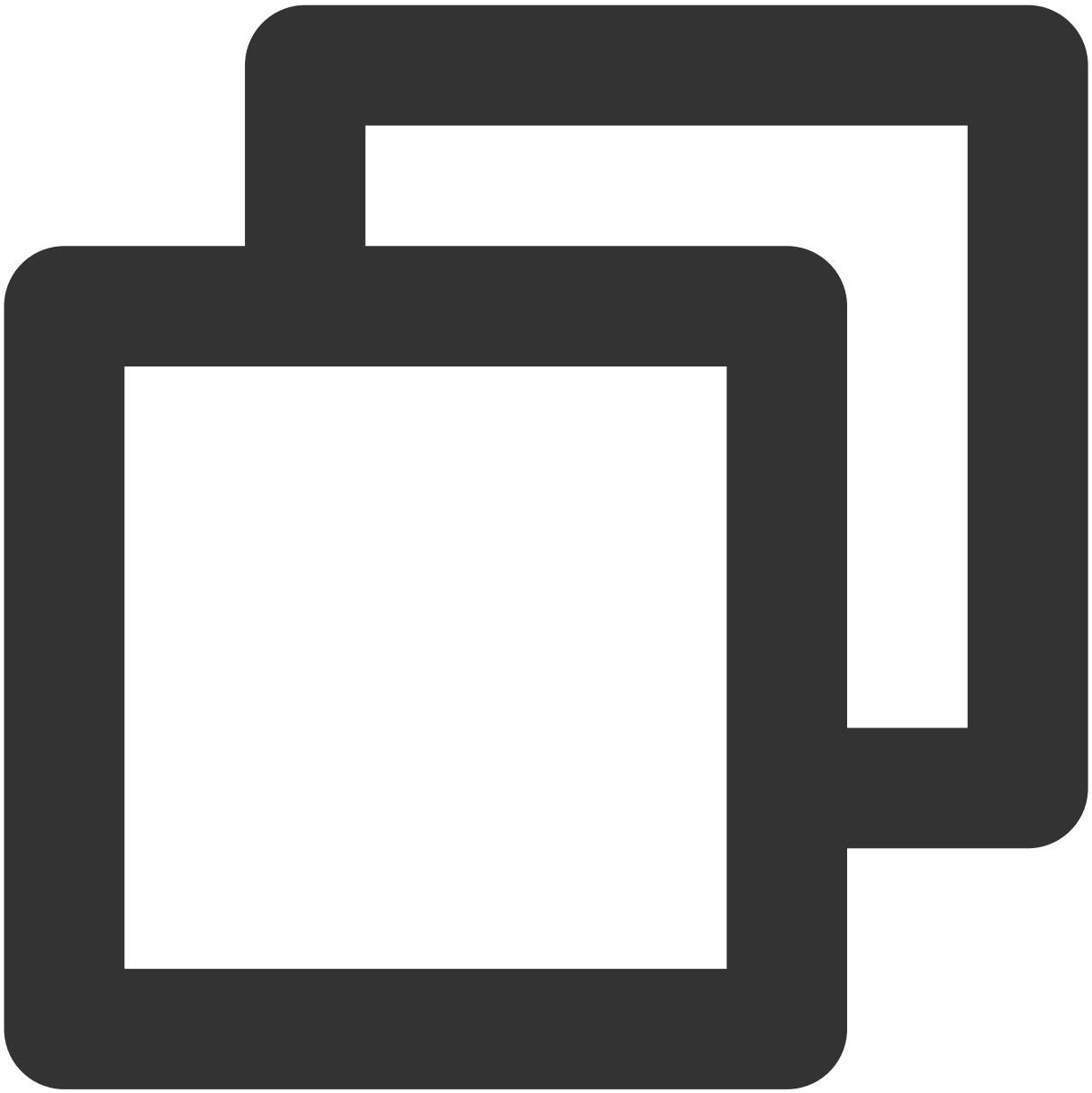
```
void stopRemoteView(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId

openCamera

开启摄像头。



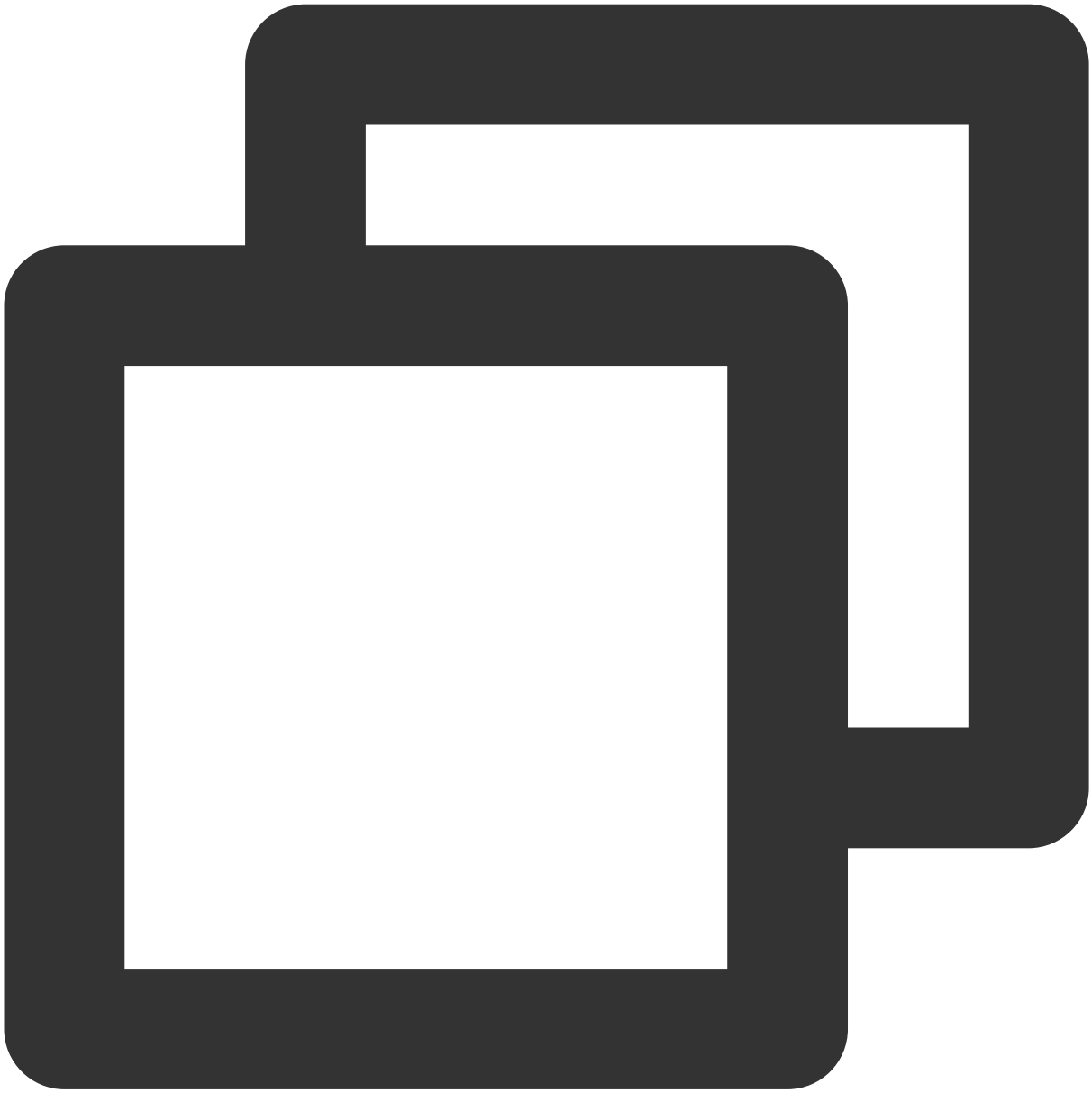
```
void openCamera(TUICommonDefine.Camera camera, TUIVideoView videoView, TUICommonDef
```

参数如下表所示：

参数	类型	含义
camera	TUICommonDefine.Camera	前置/后置 摄像头
videoView	TUIVideoView	待渲染的视图

closeCamera

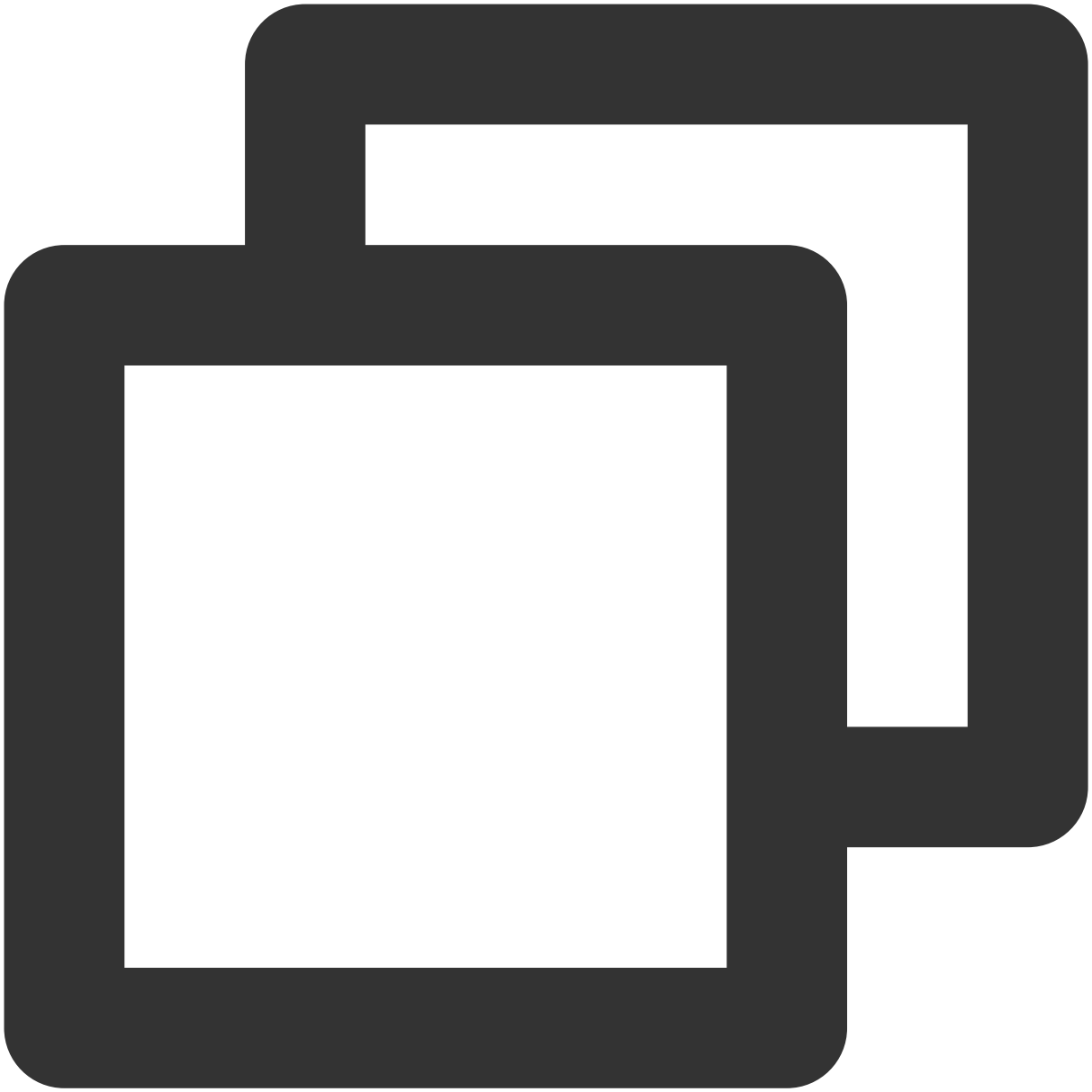
关闭摄像头。



```
void closeCamera();
```

switchCamera

切换前后摄像头。



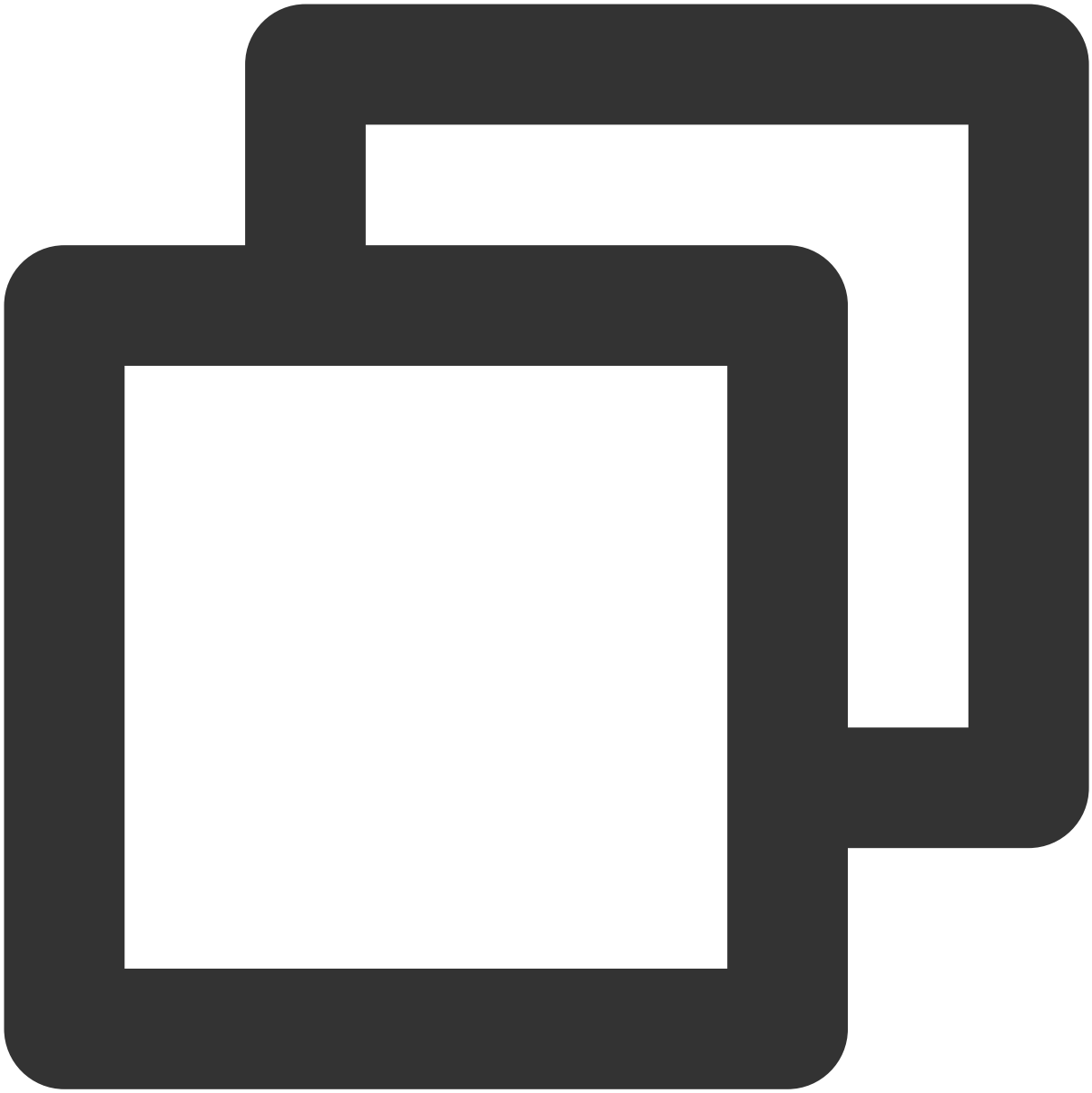
```
void switchCamera(TUICommonDefine.Camera camera);
```

参数如下表所示：

参数	类型	含义
camera	TUICommonDefine.Camera	前置/后置 摄像头

openMicrophone

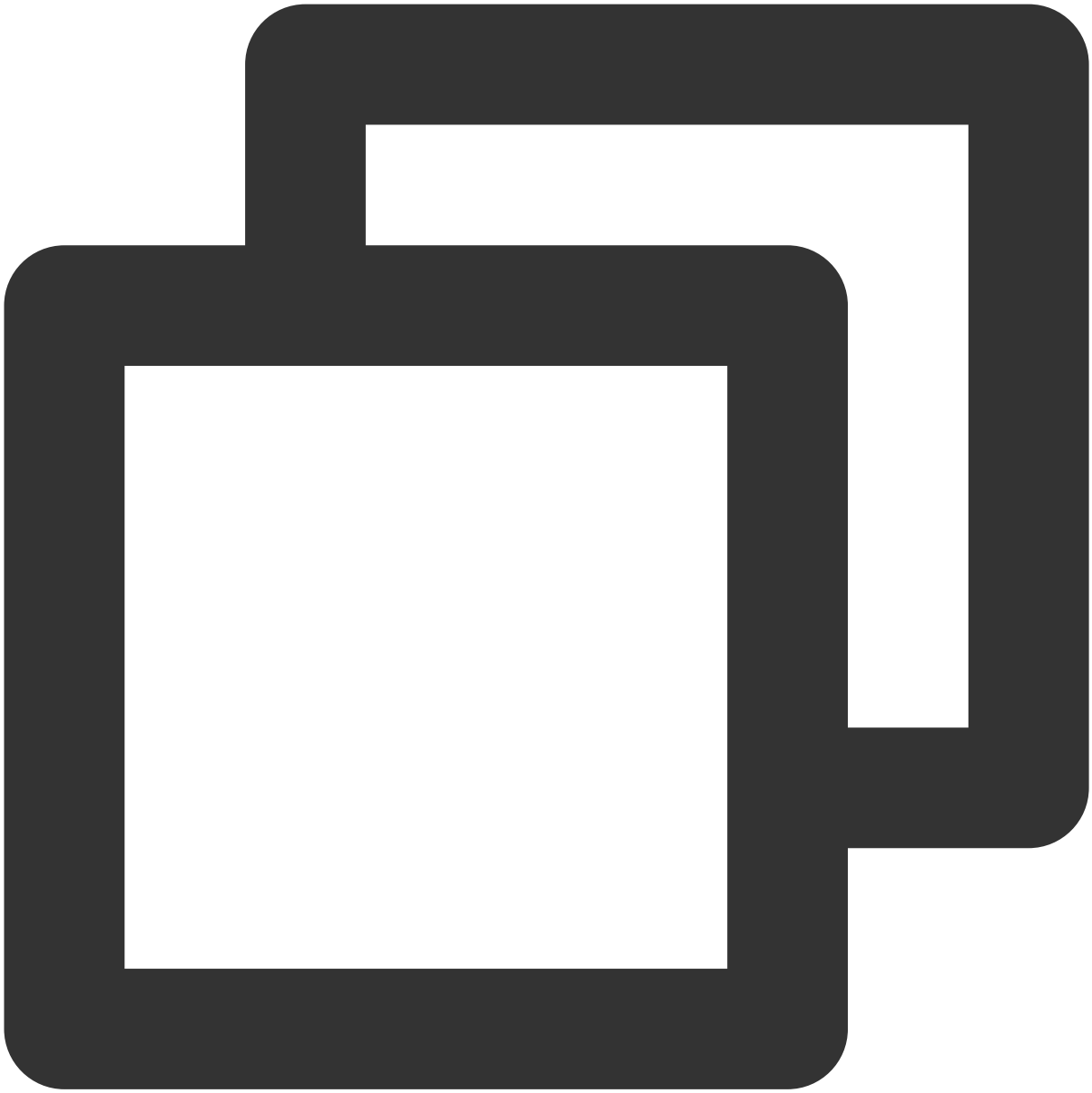
打开麦克风。



```
void openMicrophone(TUICommonDefine.Callback callback);
```

closeMicrophone

关闭麦克风。

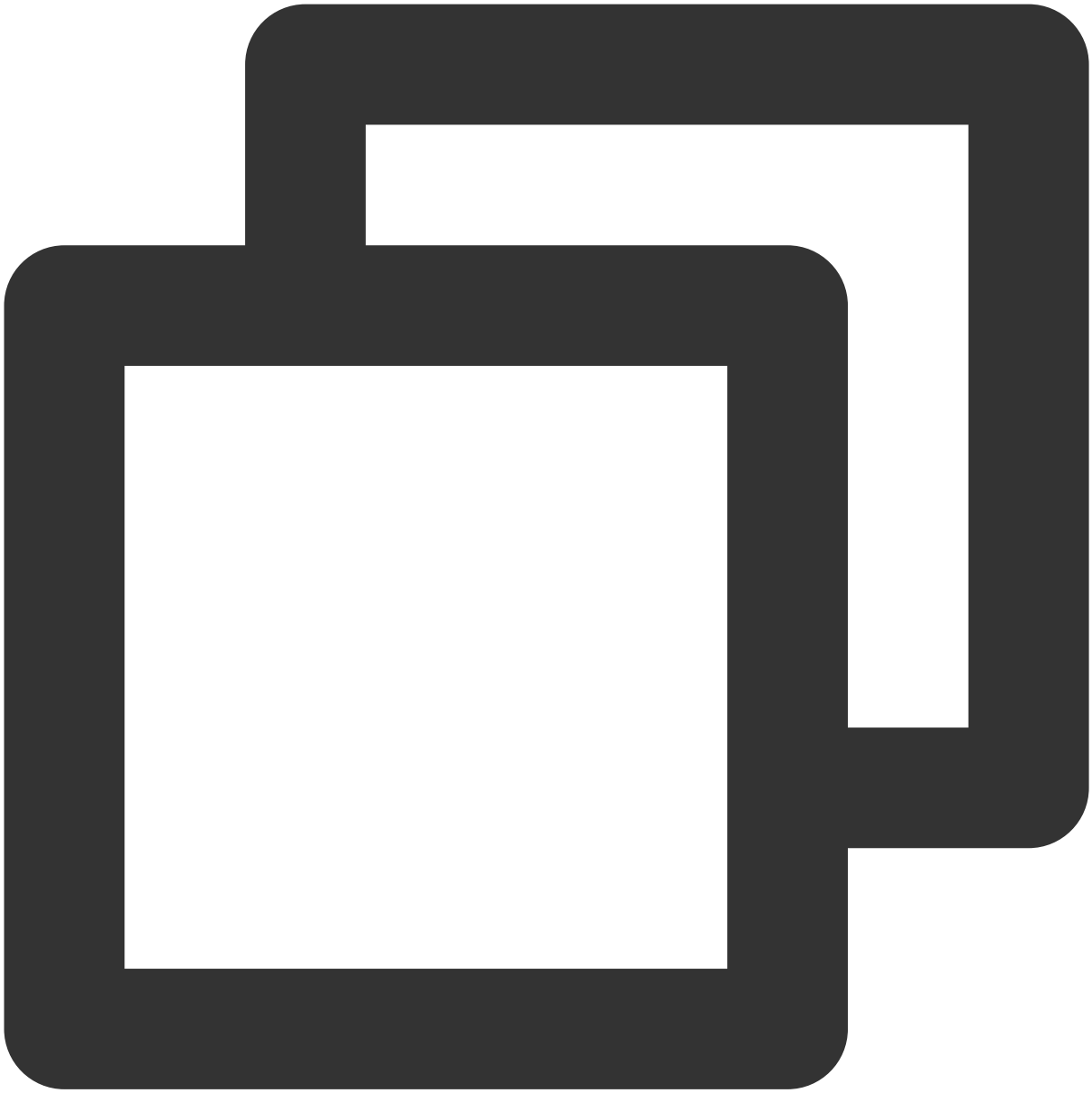


```
void closeMicrophone();
```

selectAudioPlaybackDevice

选择音频播放设备。

目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。



```
void selectAudioPlaybackDevice(TUICommonDefine.AudioPlaybackDevice device);
```

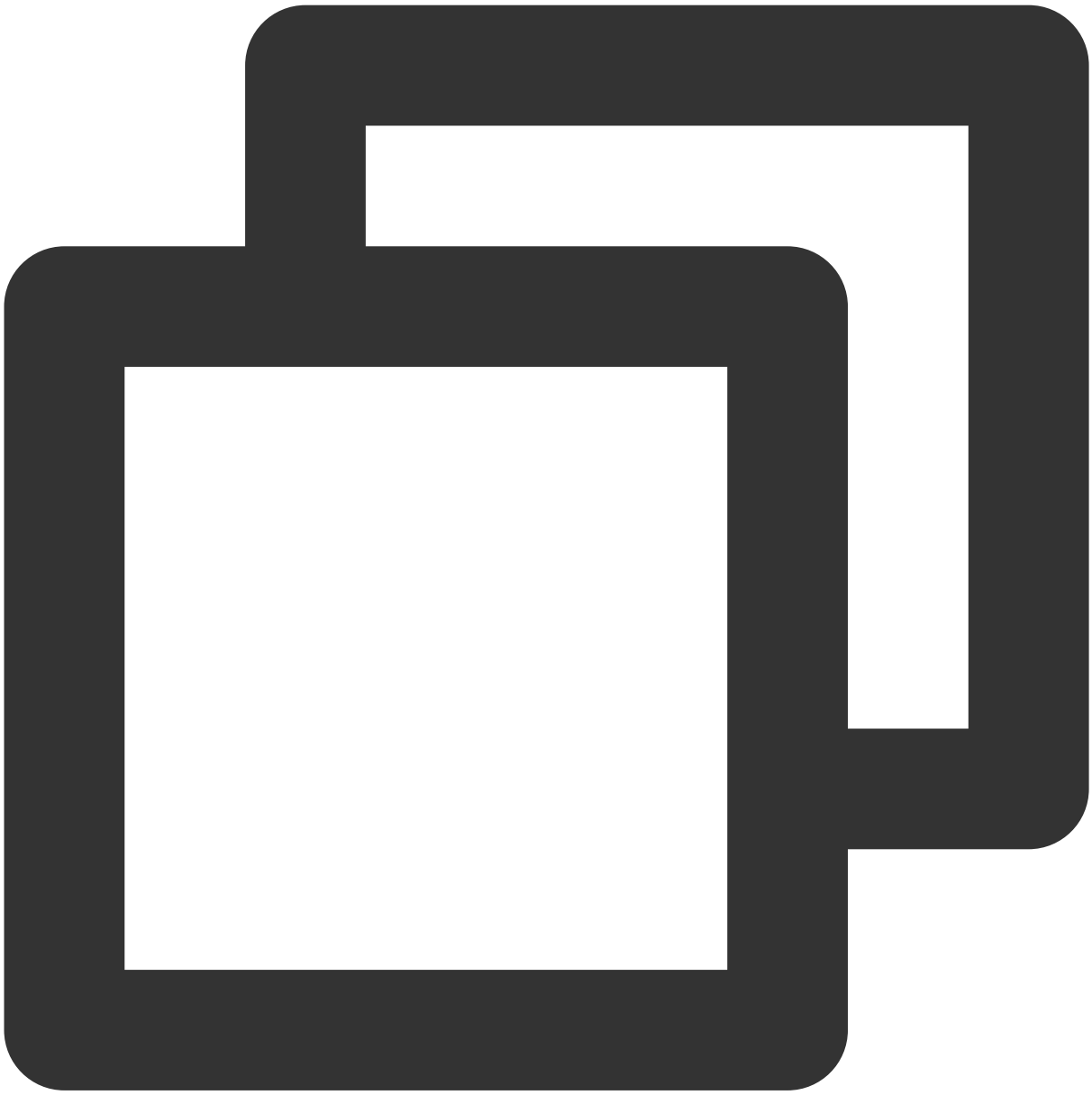
参数如下表所示：

参数	类型	含义
device	TUICommonDefine.AudioPlaybackDevice	听筒/扬声器

setSelfInfo

设置用户昵称、头像。

用户昵称不能超过500字节，用户头像必须是 URL 格式。



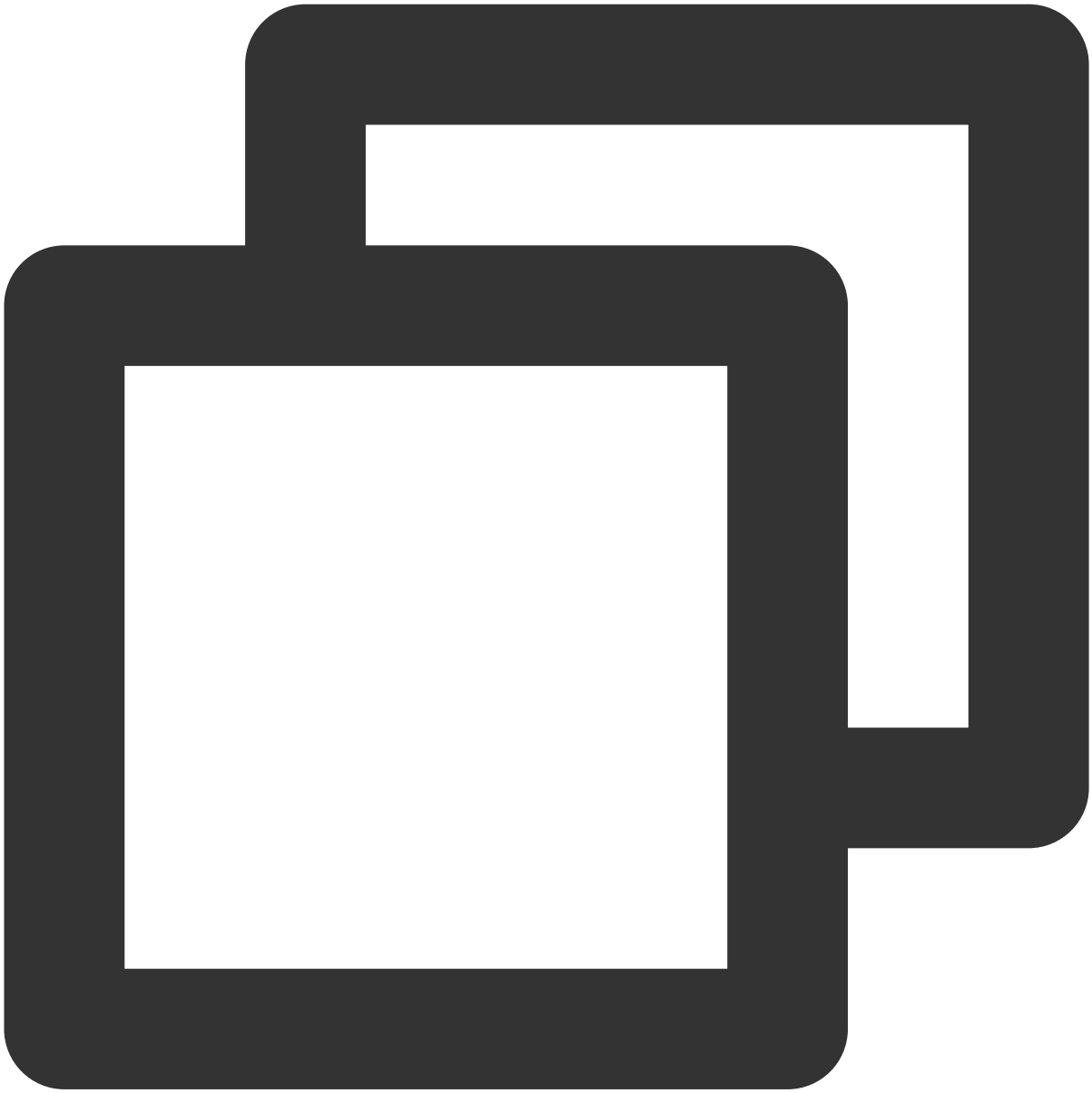
```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
nickname	String	用户昵称
avatar	String	用户头像（格式为 URL）

enableMultiDeviceAbility

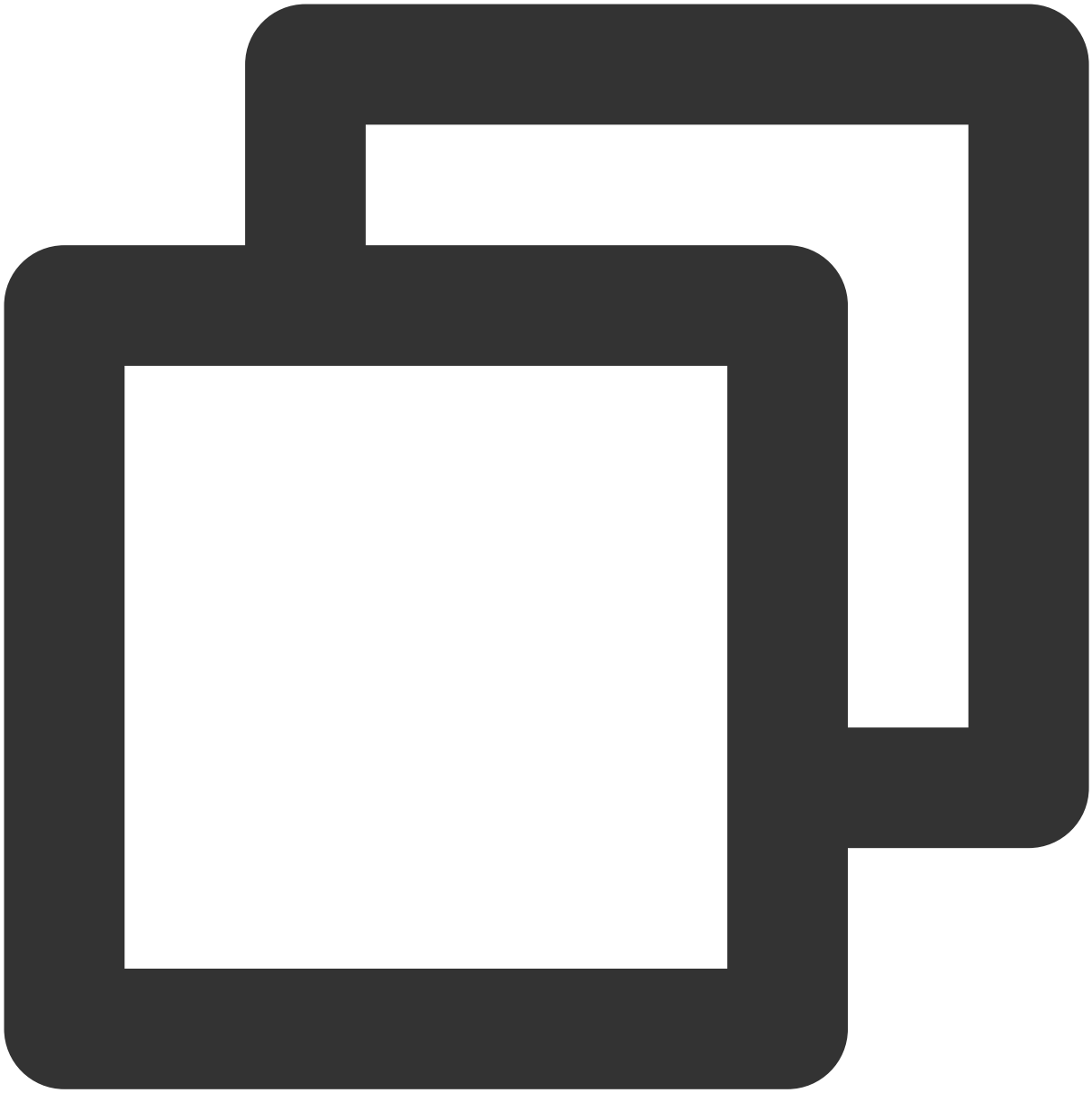
开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）。



```
void enableMultiDeviceAbility(boolean enable, TUICommonDefine.Callback callback);
```

setVideoRenderParams

设置用户视频画面的渲染模式。



```
void setVideoRenderParams(String userId, TUICommonDefine.VideoRenderParams params,
```

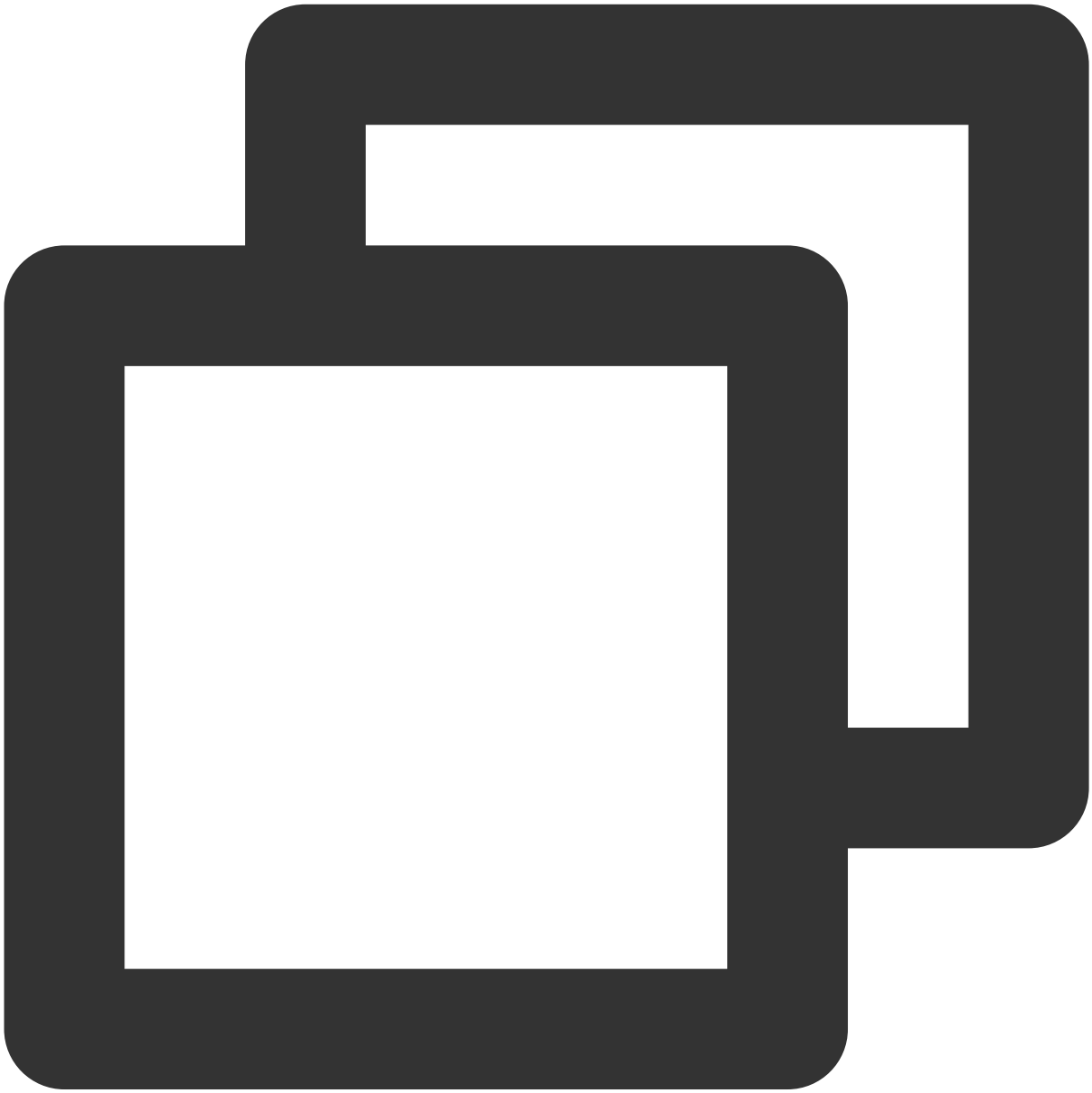
参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
params	TUICommonDefine.VideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式

setVideoEncoderParams

设置视频编码的编码参数。

该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。



```
void setVideoEncoderParams(TUICommonDefine.VideoEncoderParams params, TUICommonDefi
```

参数如下表所示：

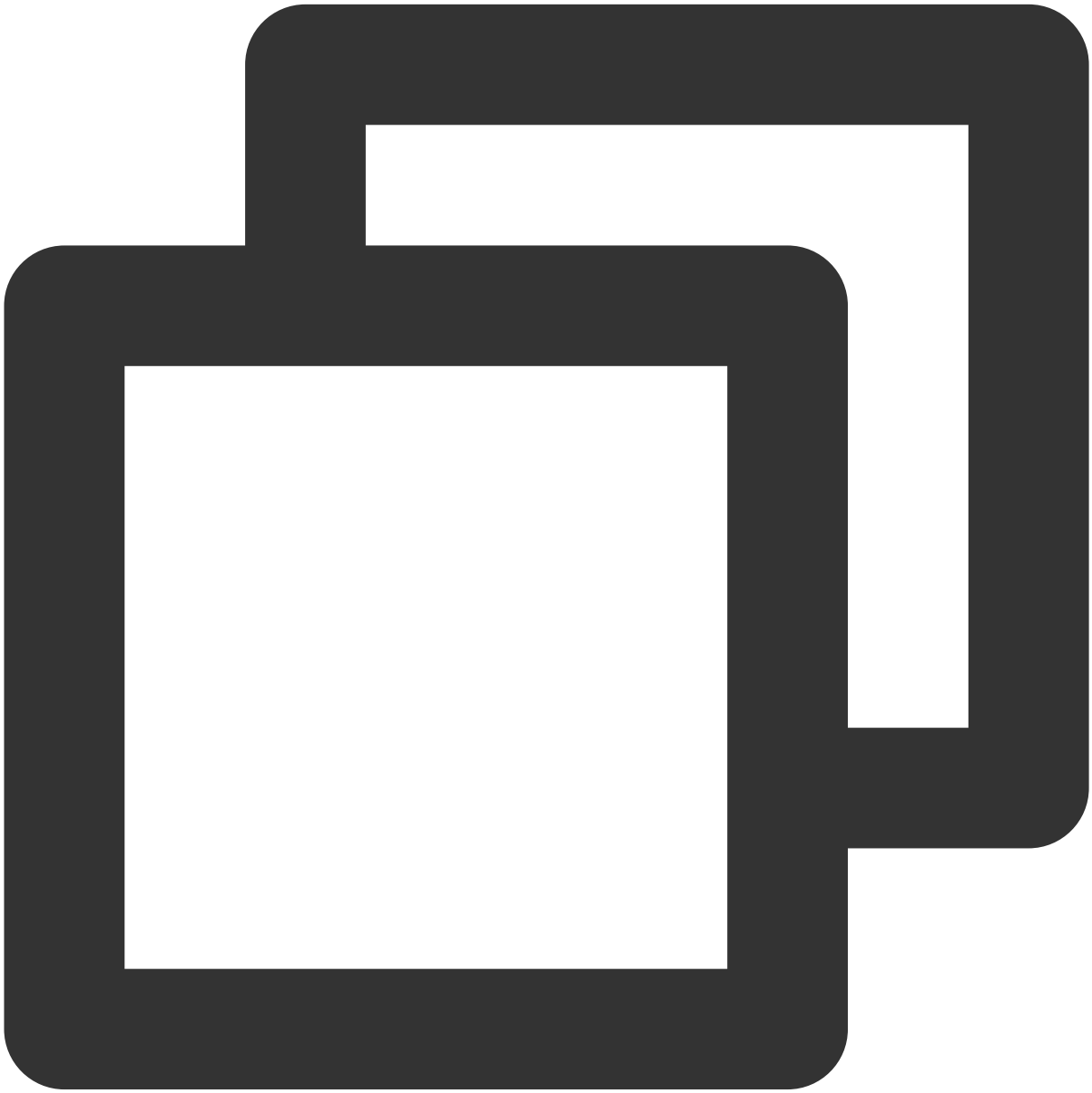
参数	类型	含义
params	TUICommonDefine.VideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高

		比

getTRTCCloudInstance

高级接口。

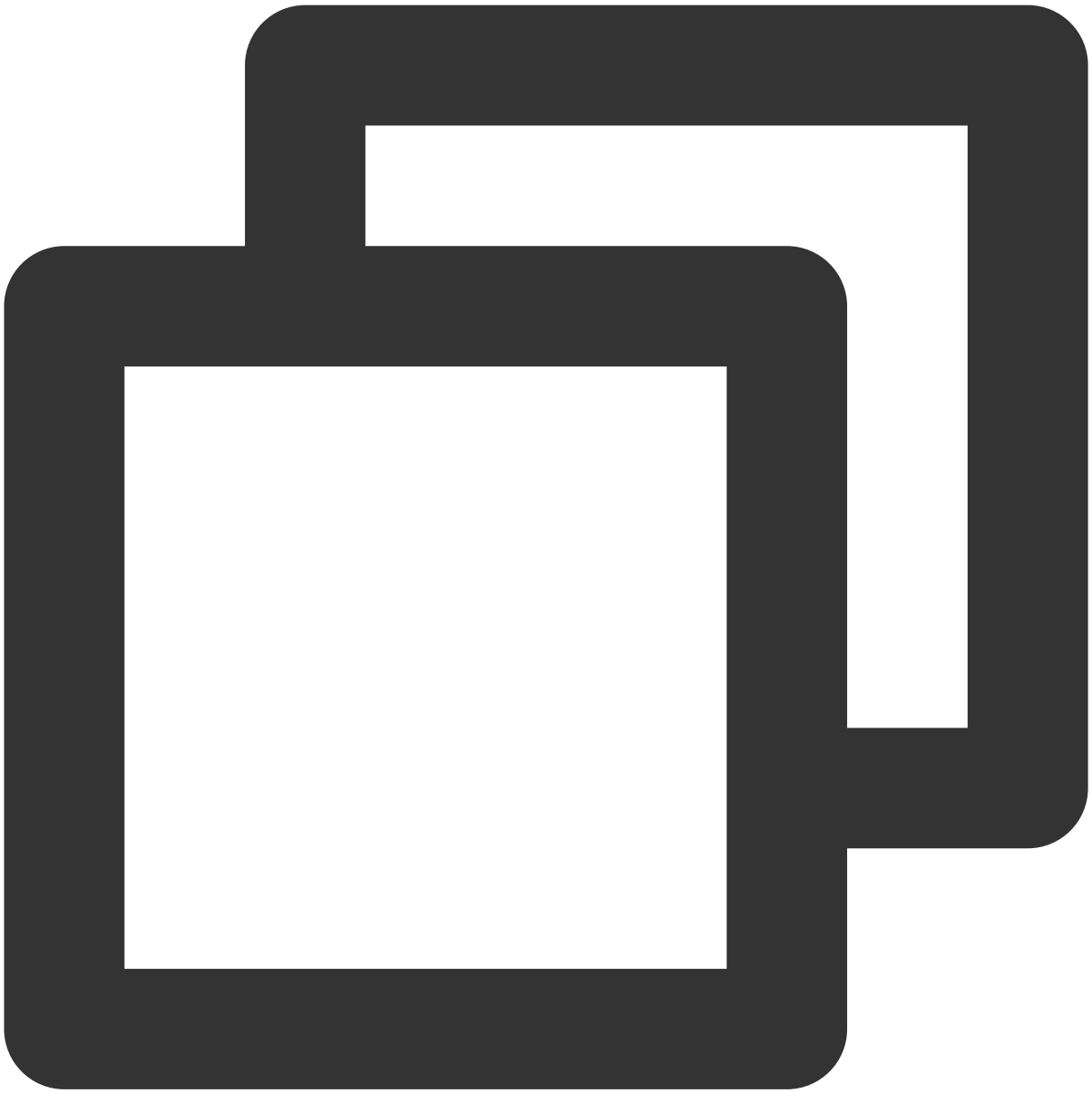
用于接入第三方美颜等。



```
TRTCCloud getTRTCCloudInstance();
```

setBeautyLevel

设置美颜级别，支持关闭默认美颜。



```
void setBeautyLevel(float level, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
level	float	美颜级别，取值范围 0 - 9；0 表示关闭，9 表示效果最明显。

版权所有：腾讯云计算（北京）有限责任公司

TUICallObserver

最近更新时间：2024-01-17 11:58:53

TUICallObserver API 简介

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线

版权所有：腾讯云计算（北京）有限责任公司

onUserSigExpired	在线时票据过期

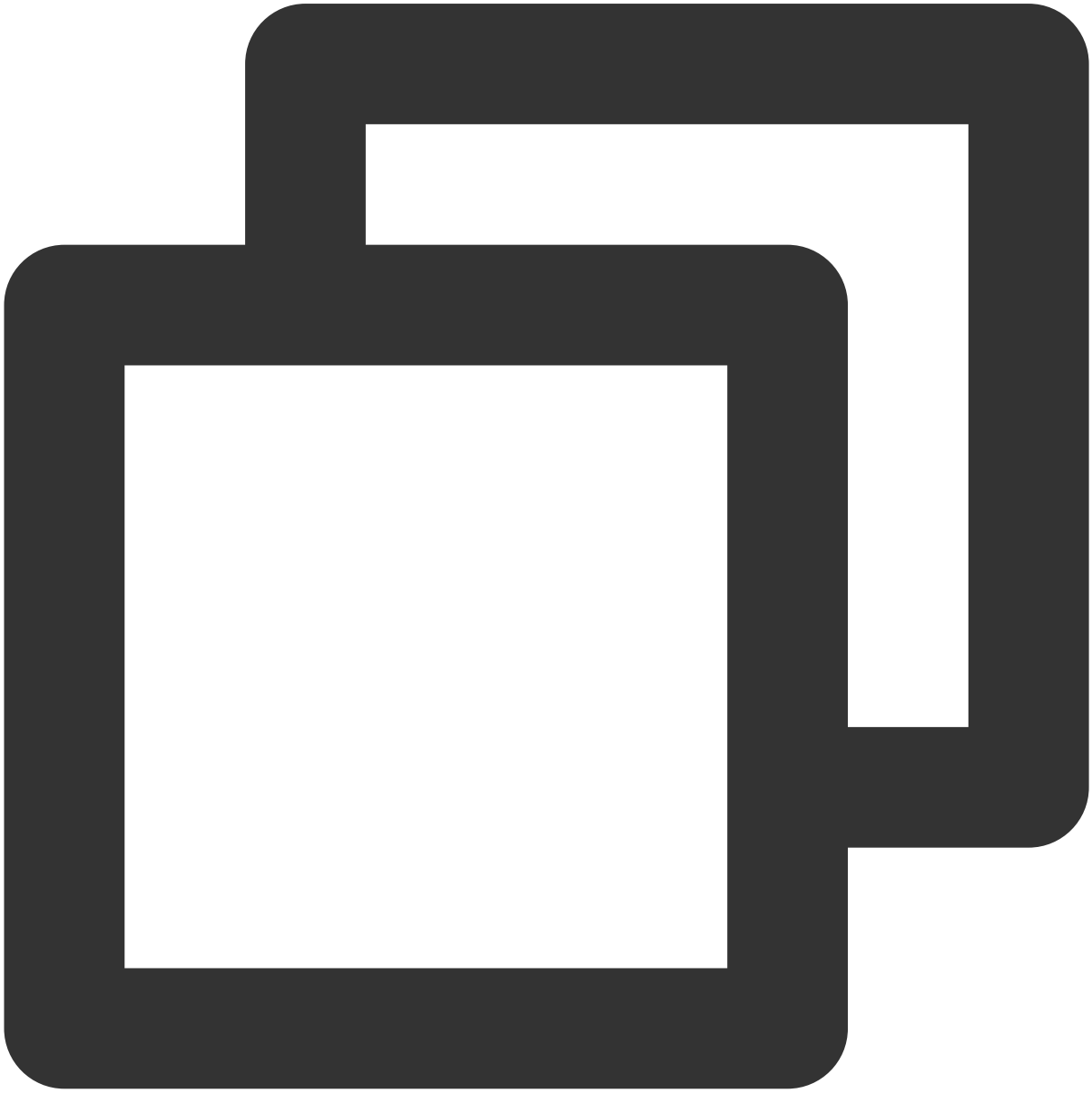
回调事件详情

onError

错误回调。

说明

SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。



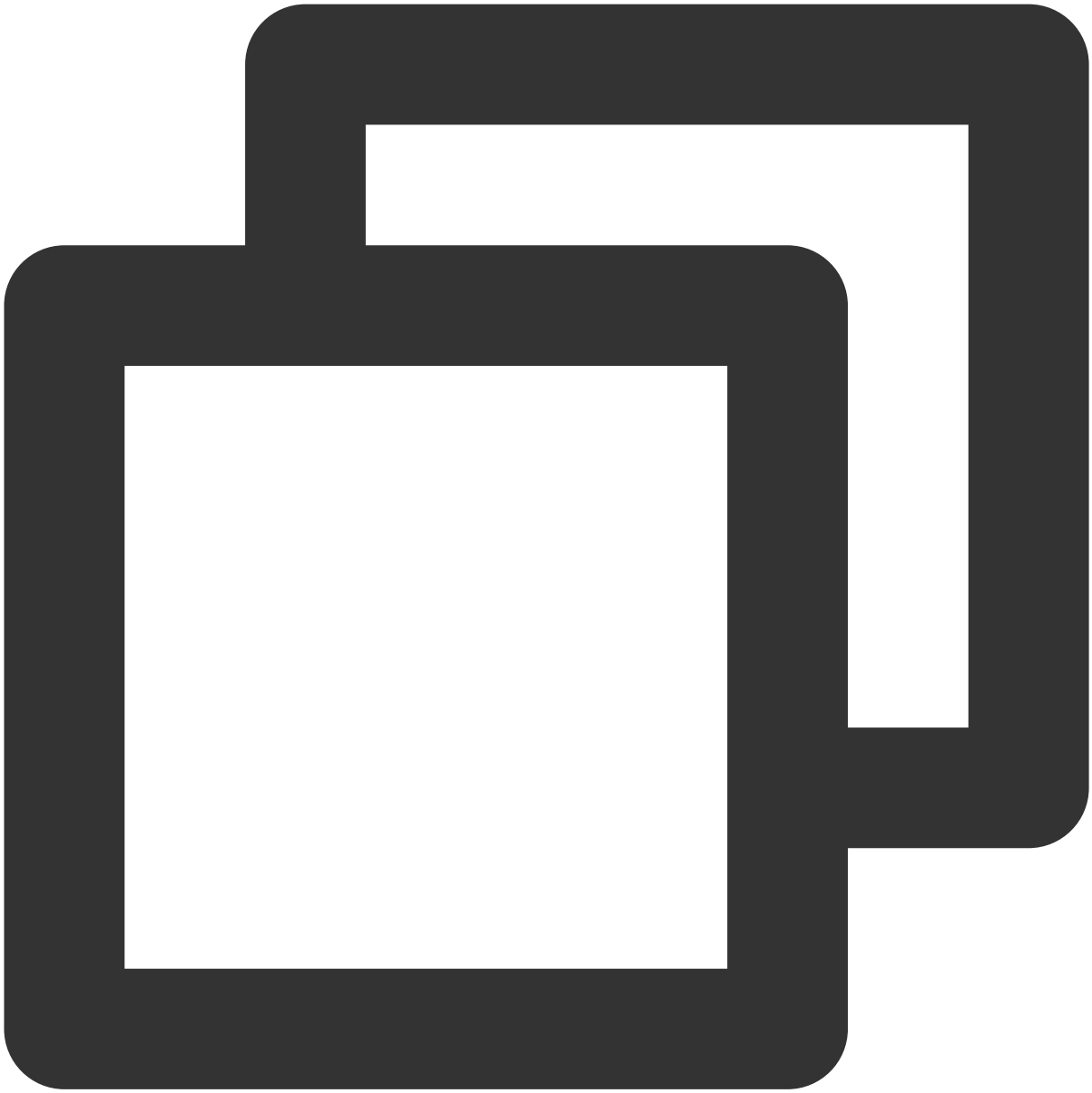
```
void onError(int code, String msg);
```

参数如下表所示：

参数	类型	含义
code	int	错误码
msg	String	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。



```
void onCallReceived(String callerId, List<String> calleeIdList, String groupId,
                    TUICallDefine.MediaType callMediaType, String userData);
```

参数如下表所示：

参数	类型	含义
callerId	String	主叫 ID（邀请方）
calleeIdList	List	被叫 ID 列表（被邀请方）

groupId	String	群组通话 ID
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
userData	String	用户增加的扩展字段，详见 TUICallDefine.CallParams

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

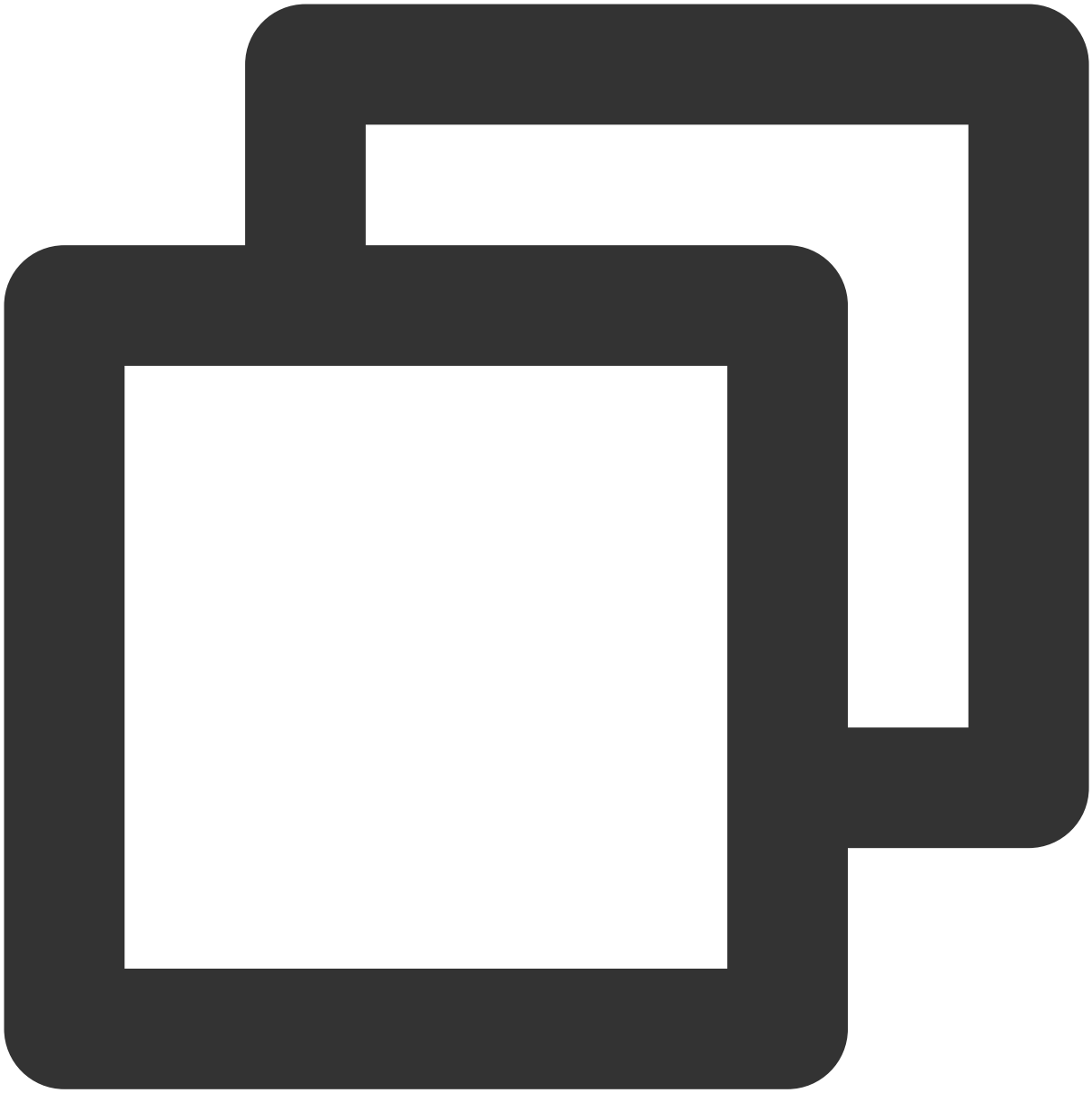
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）

异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）



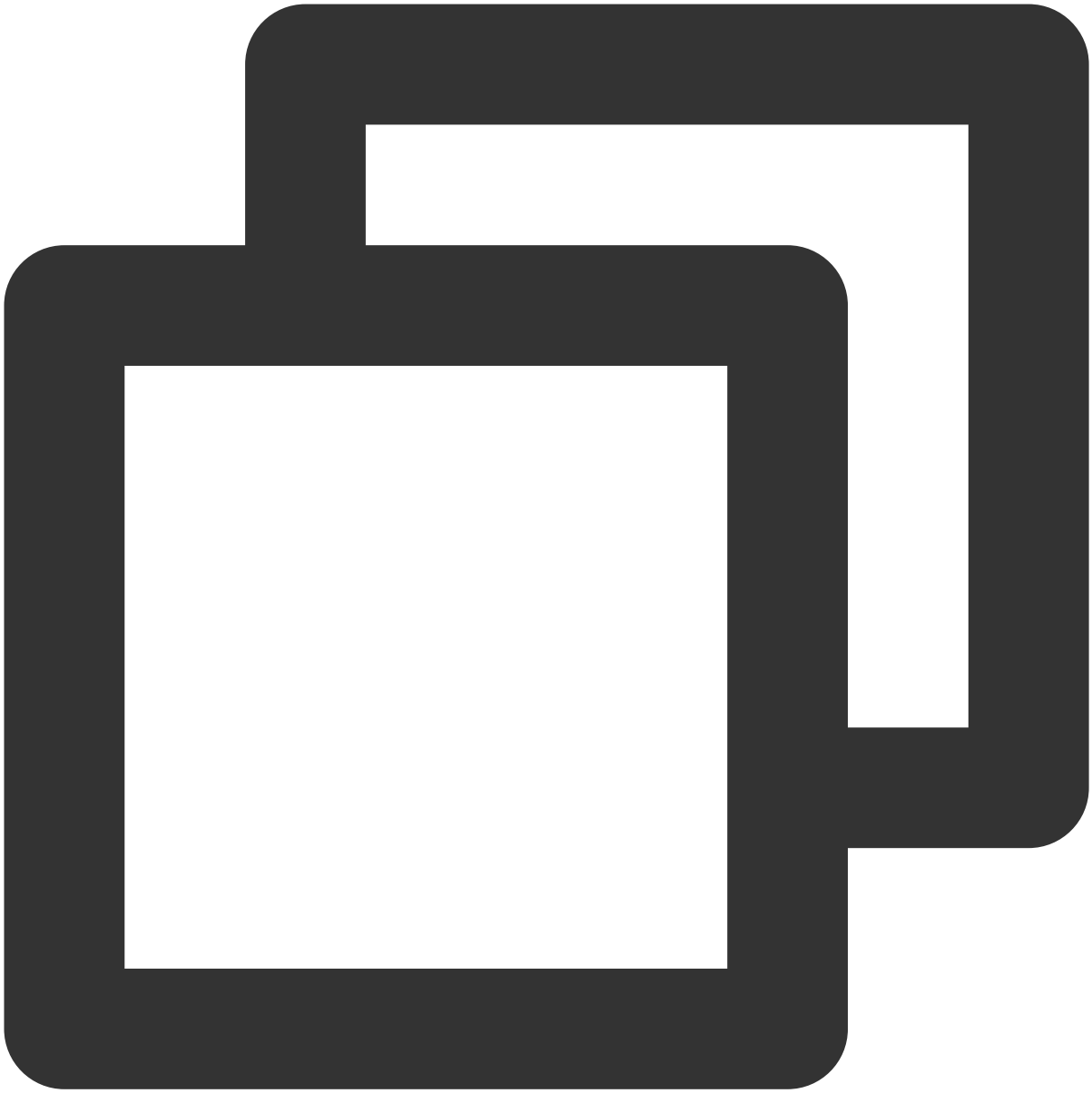
```
void onCallCancelled(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	用户的 ID

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。



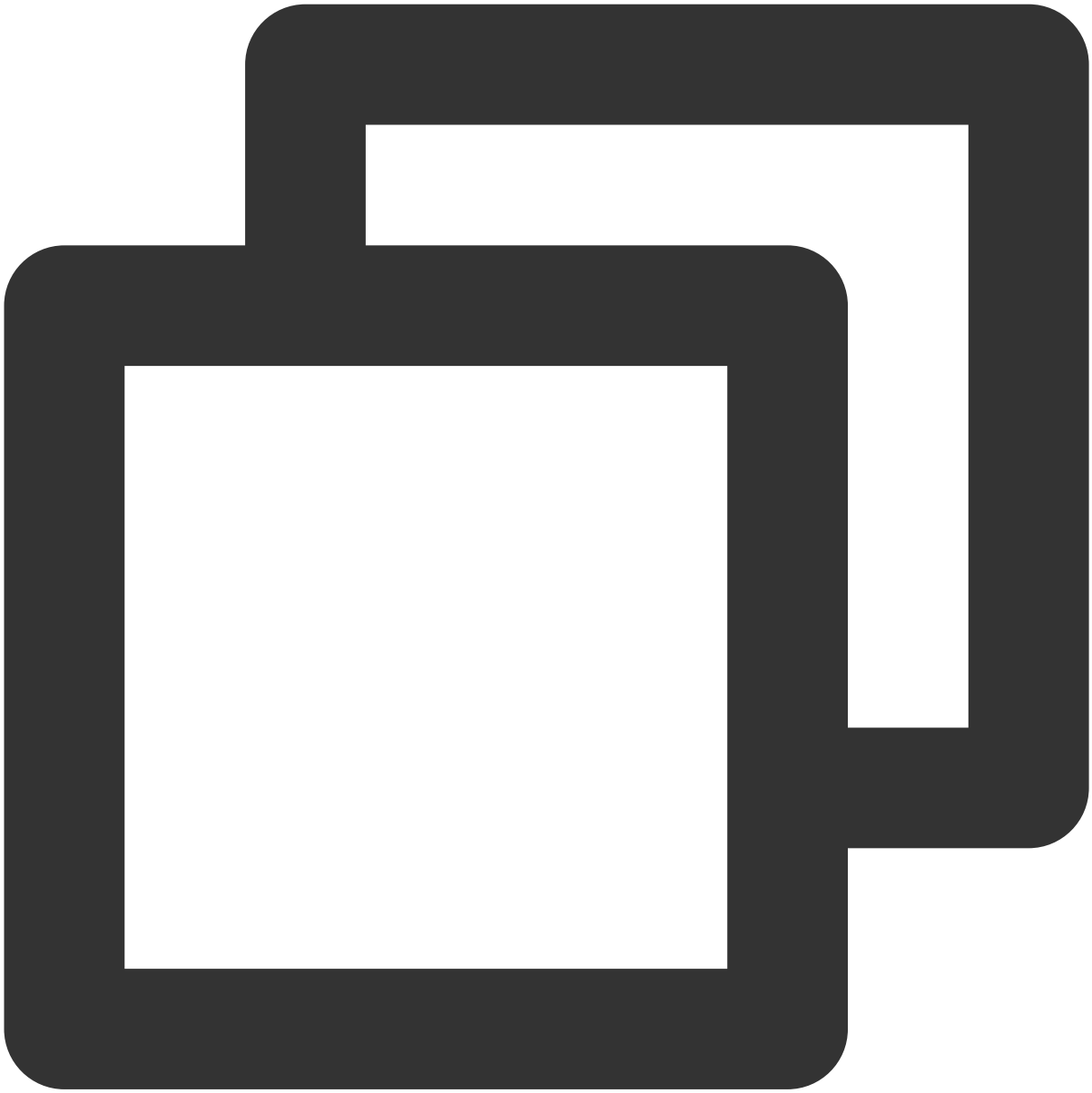
```
void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaTy
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID
callMediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallDefine.Role	角色，枚举类型：主叫、被叫

onCallEnd

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。



```
void onCallEnd(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaType
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID

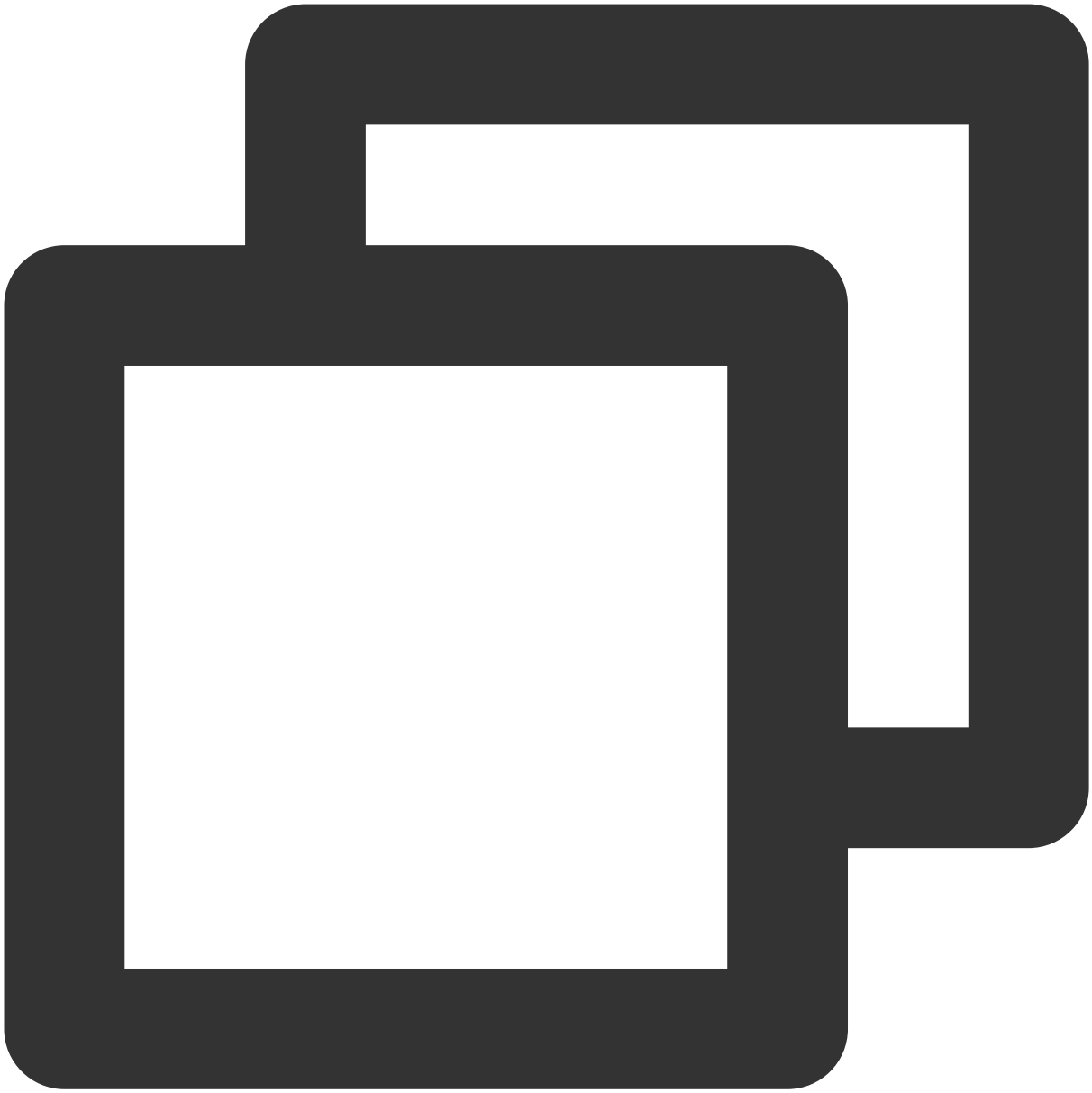
callMediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallDefine.Role	角色，枚举类型：主叫、被叫
totalTime	long	此次通话的时长，单位：秒

注意

客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用REST API来完成这类流程。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。



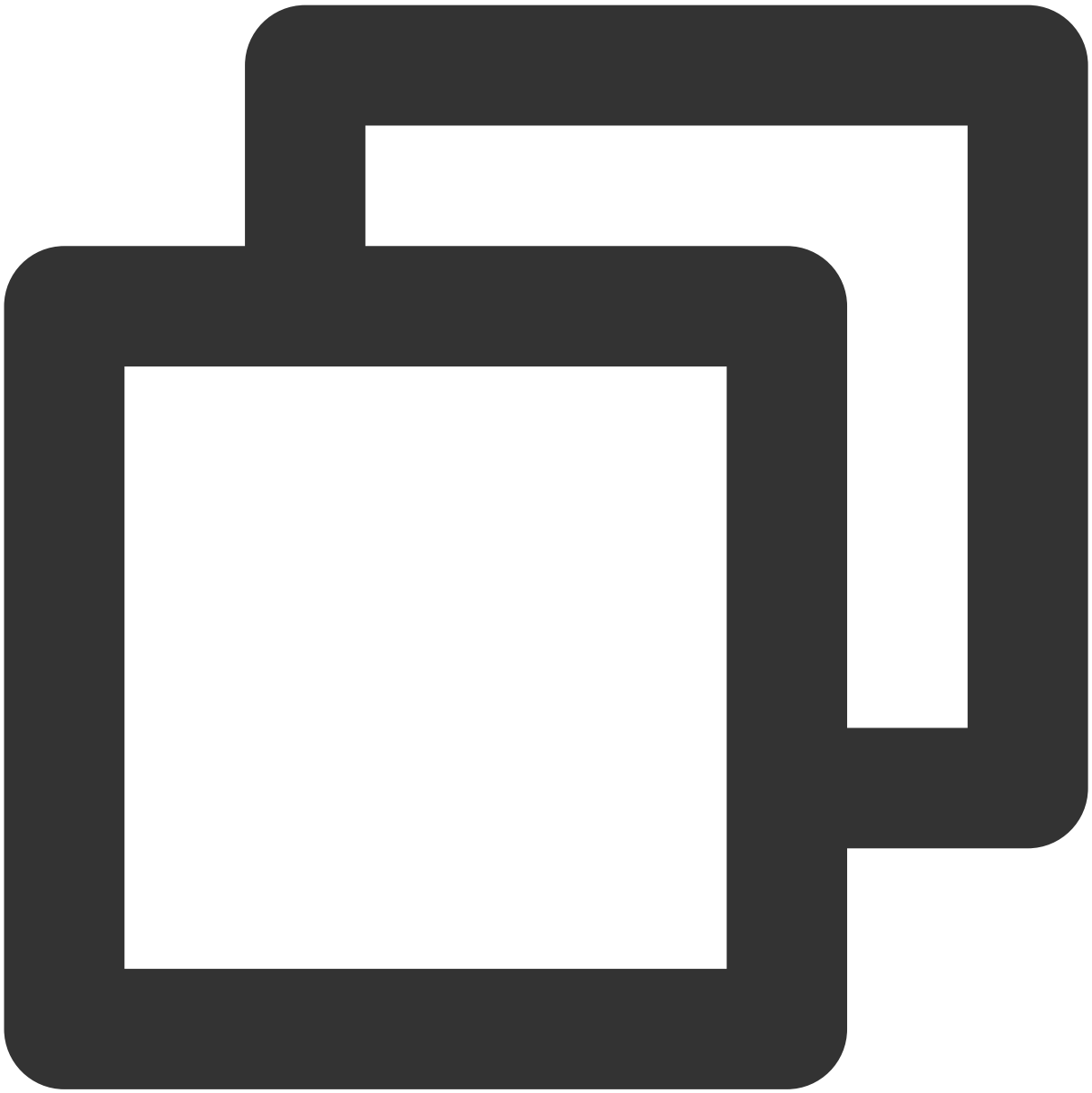
```
void onCallMediaTypeChanged(TUICallDefine.MediaType oldCallMediaType,TUICallDefine.
```

参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallDefine.MediaType	旧的通话类型
newCallMediaType	TUICallDefine.MediaType	新的通话类型

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。



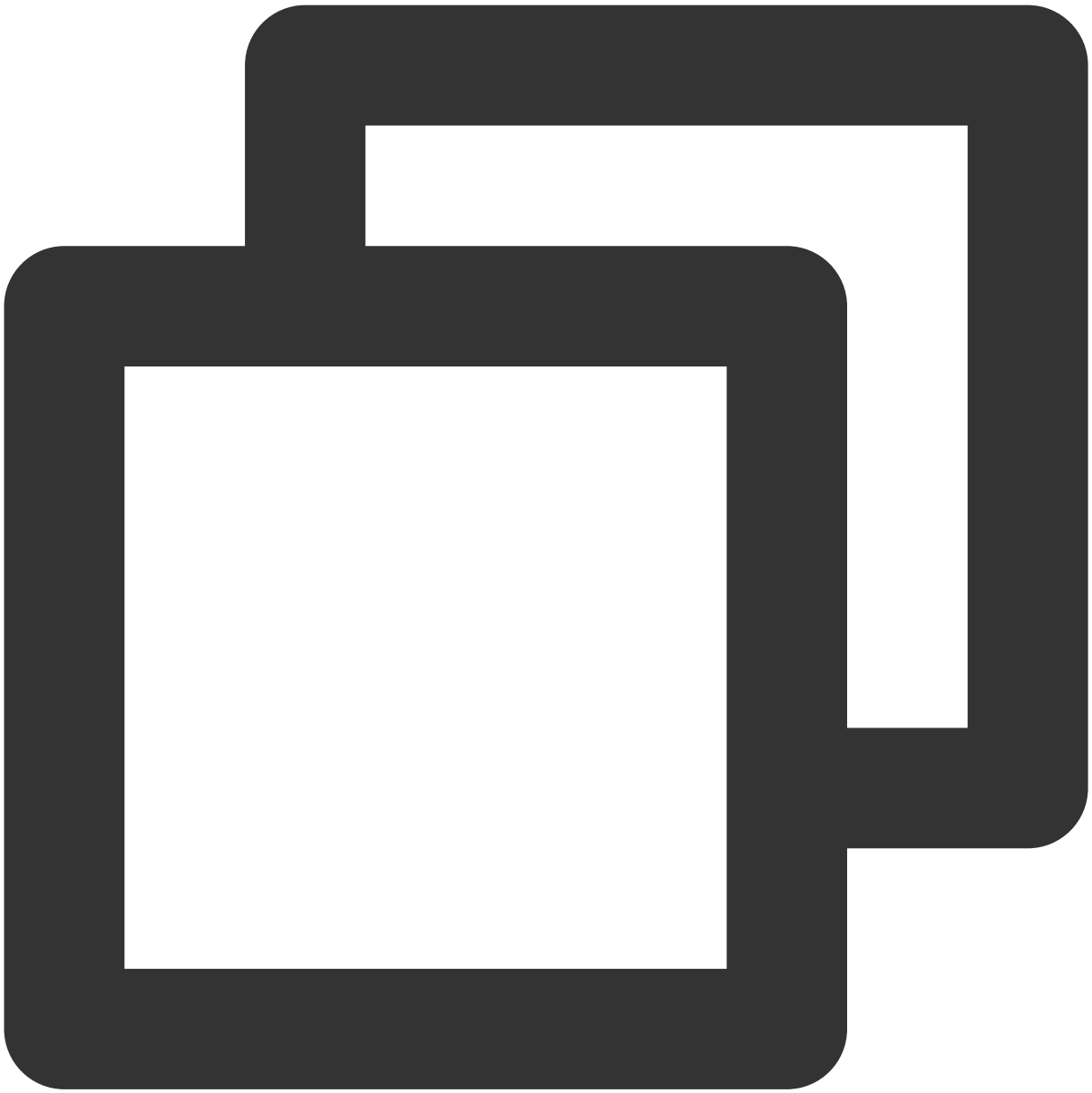
```
void onUserReject(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。



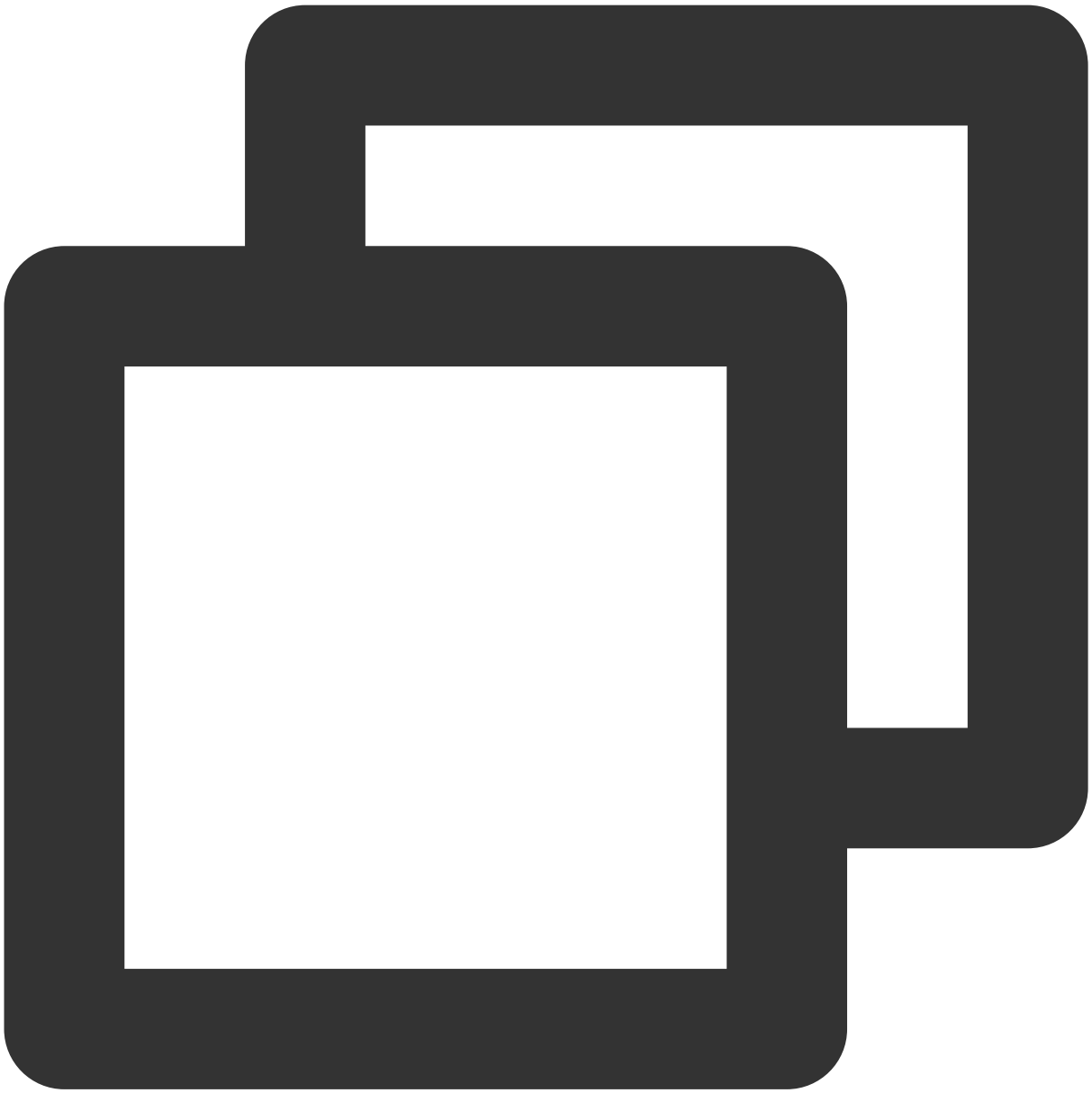
```
void onUserNoResponse(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	无响应用户的 ID

onUserLineBusy

通话忙线回调。



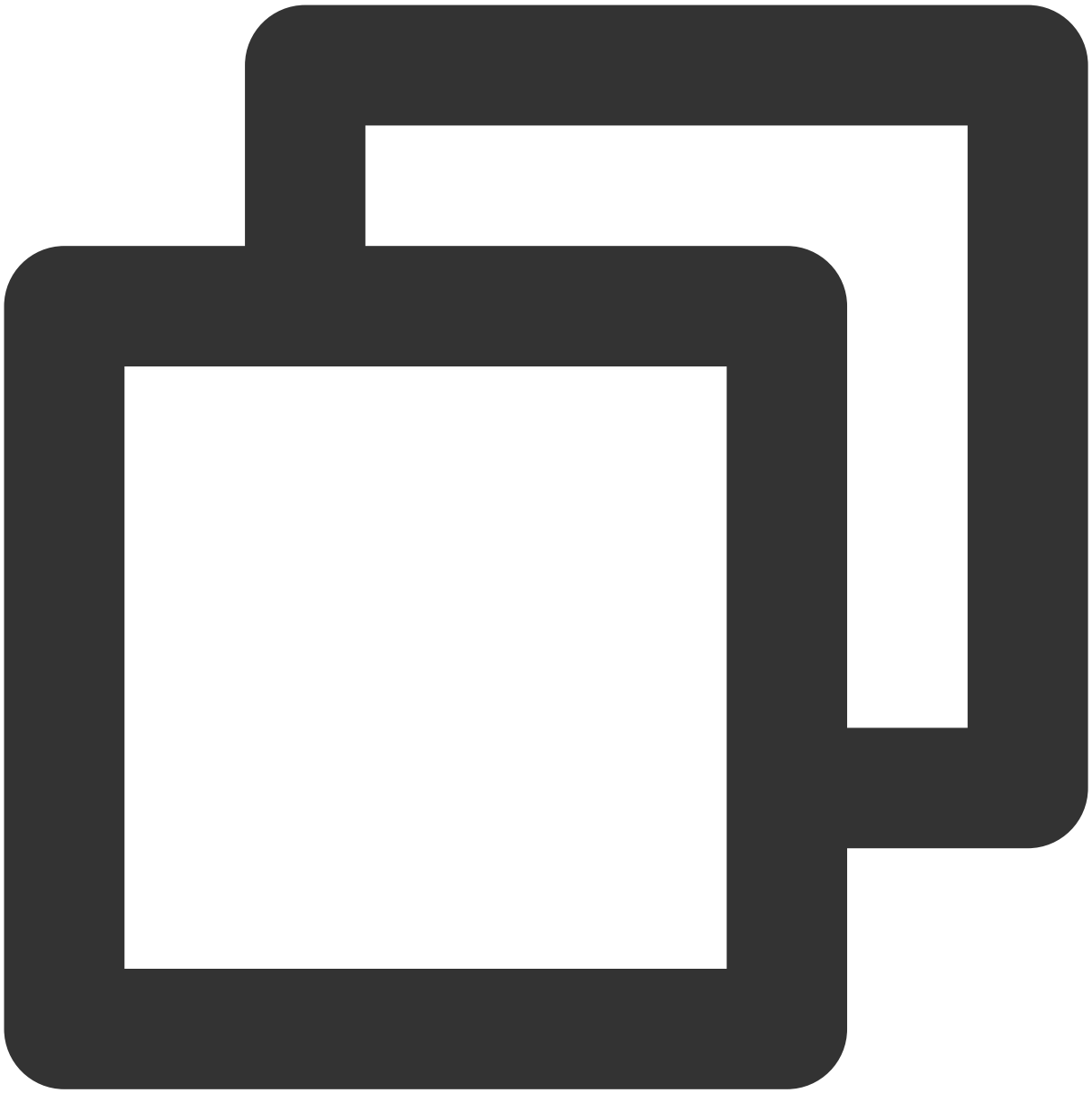
```
void onUserLineBusy(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	忙线用户的 ID

onUserJoin

有用户进入此次通话的回调。



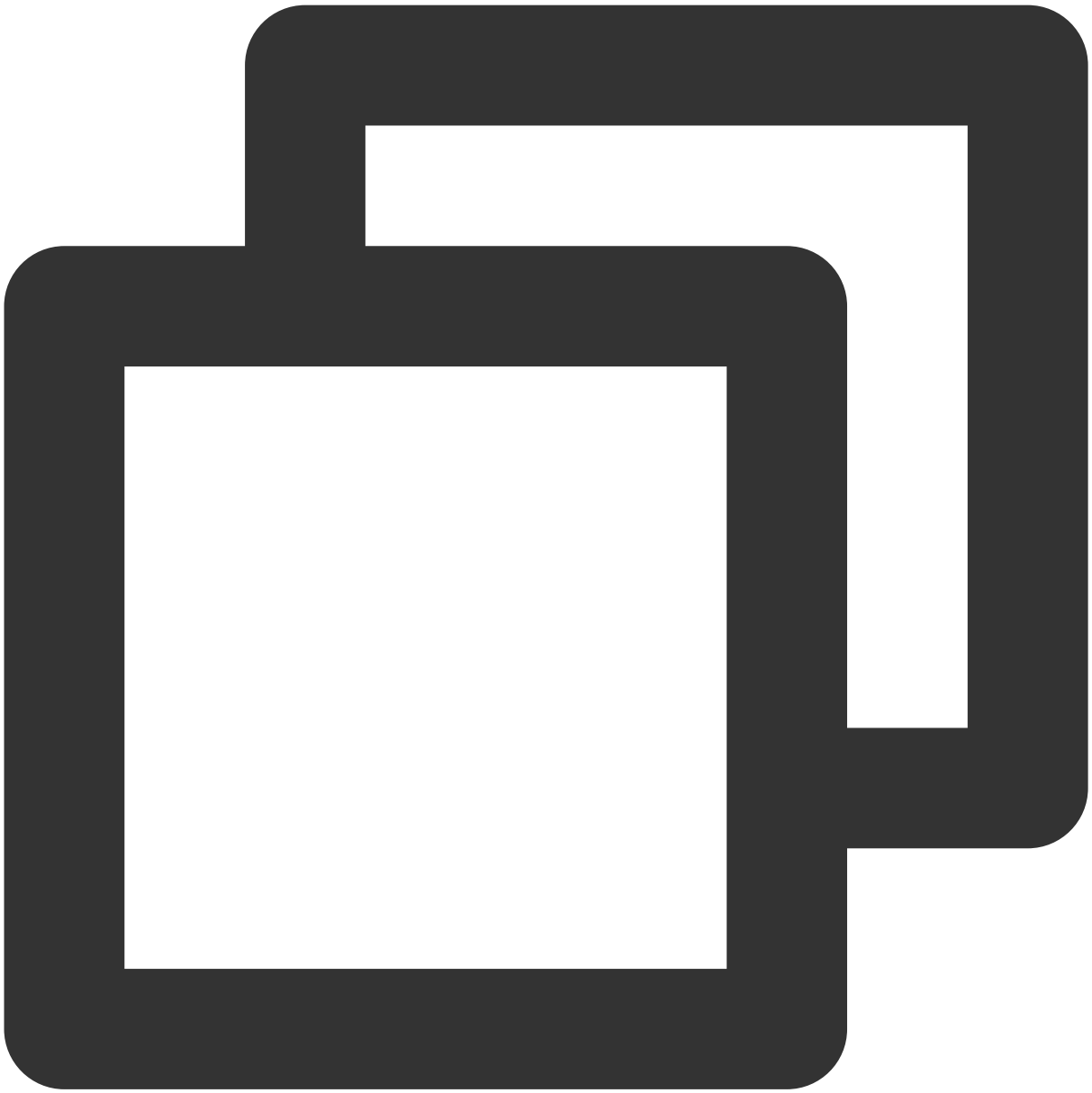
```
void onUserJoin(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。



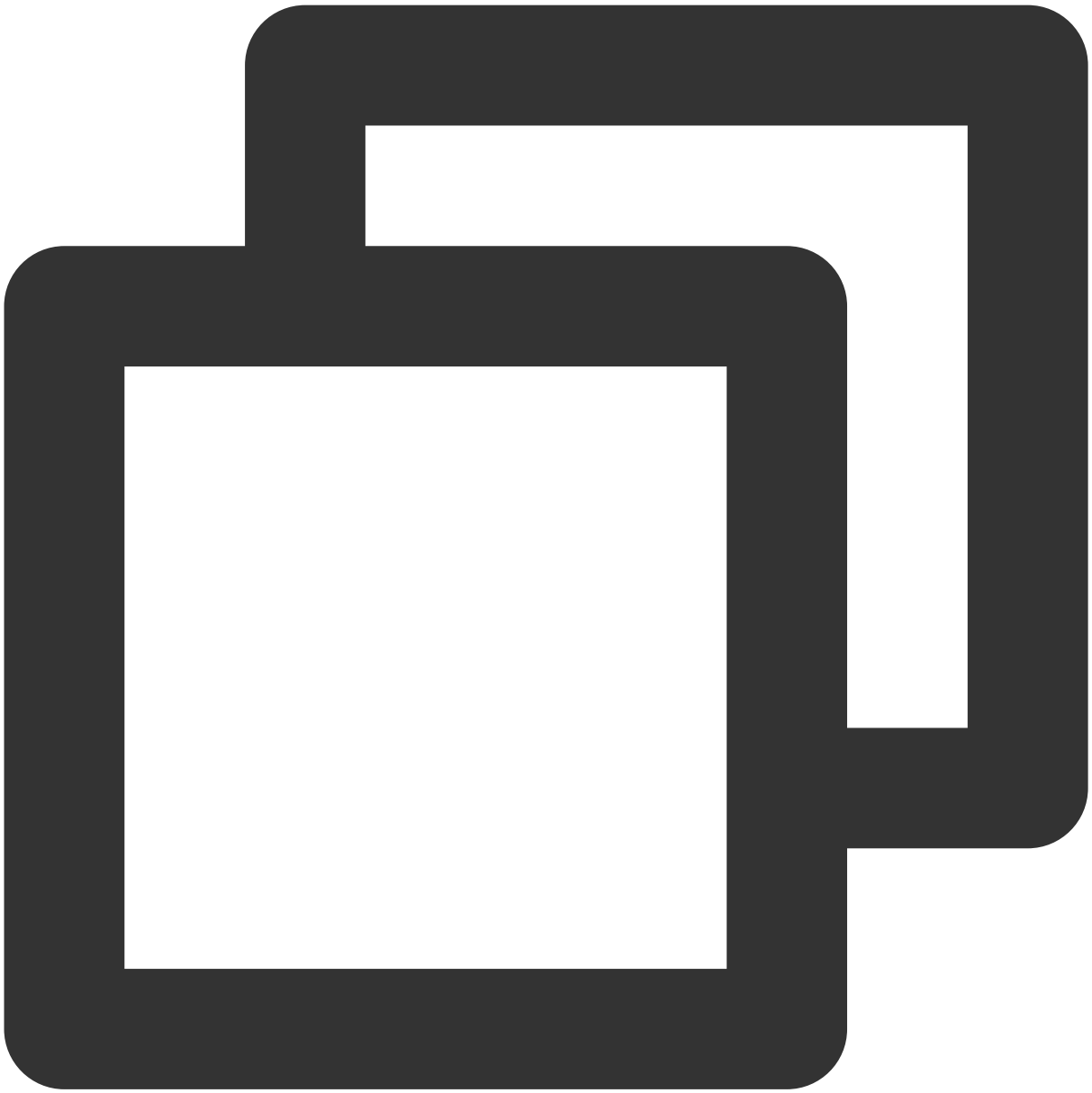
```
void onUserLeave(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。



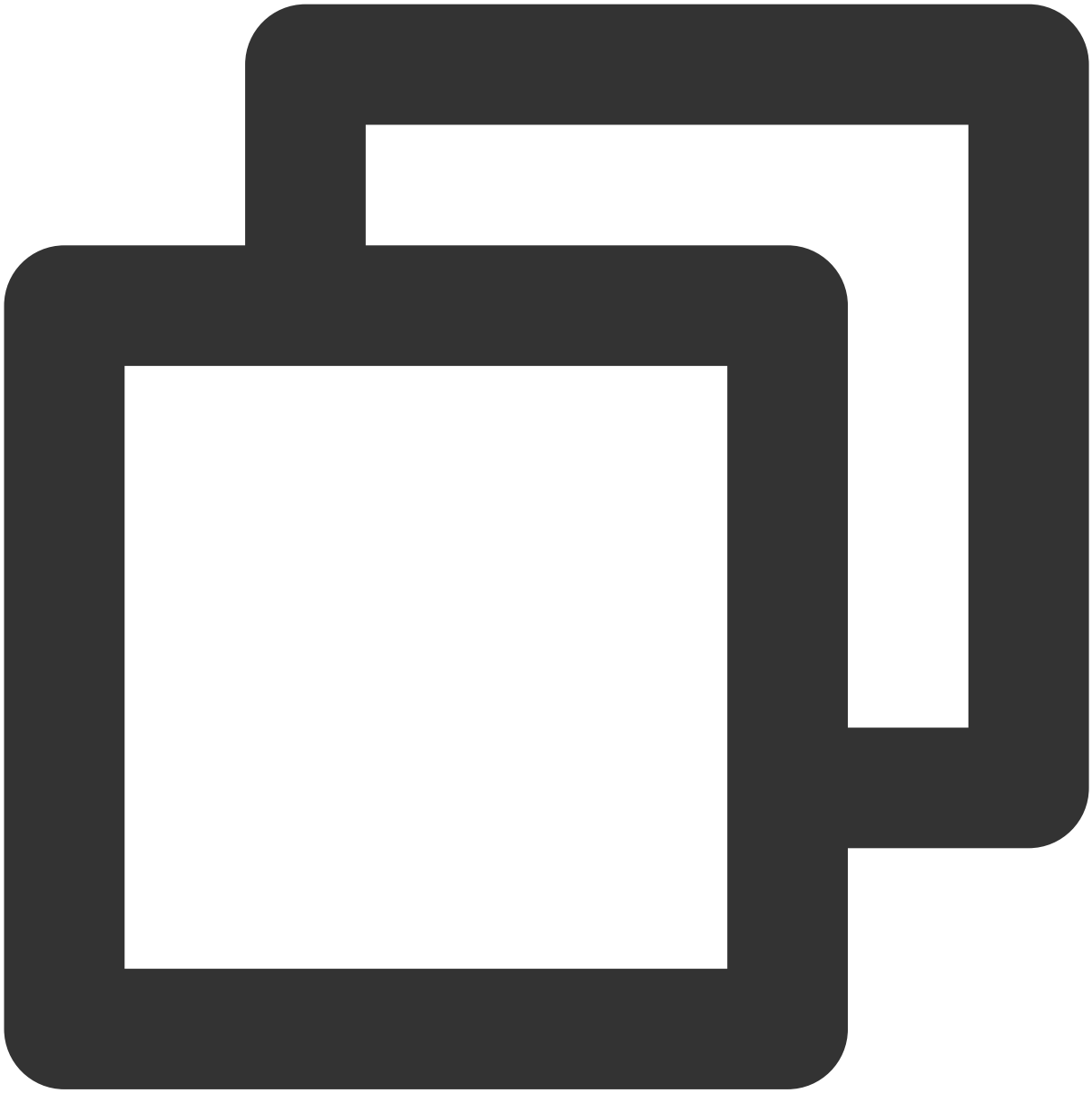
```
void onUserVideoAvailable(String userId, boolean isVideoAvailable);
```

参数如下表所示：

参数	类型	含义
userId	String	通话用户 ID
isVideoAvailable	boolean	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。



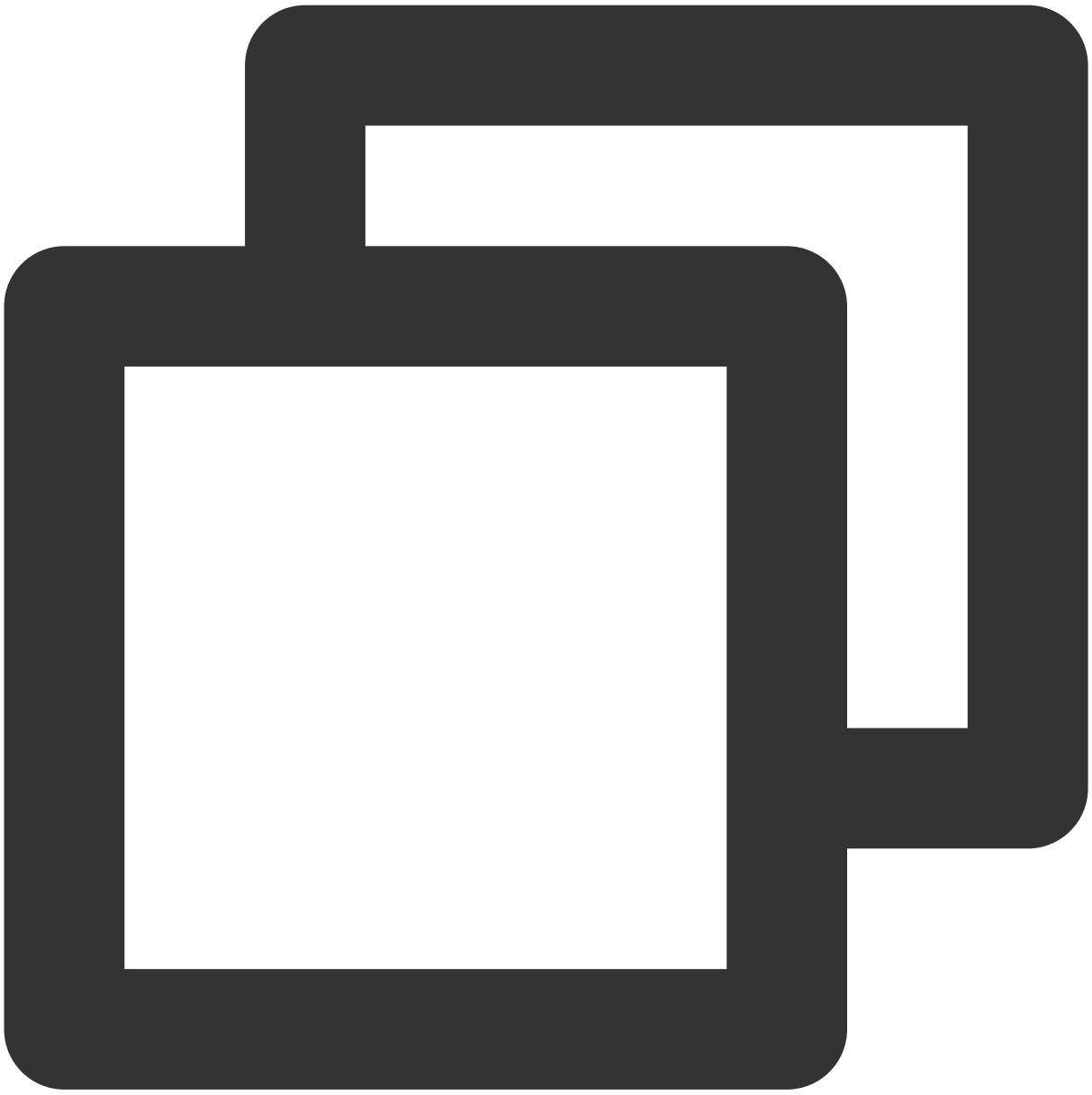
```
void onUserAudioAvailable(String userId, boolean isAudioAvailable);
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID
isAudioAvailable	boolean	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。



```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap);
```

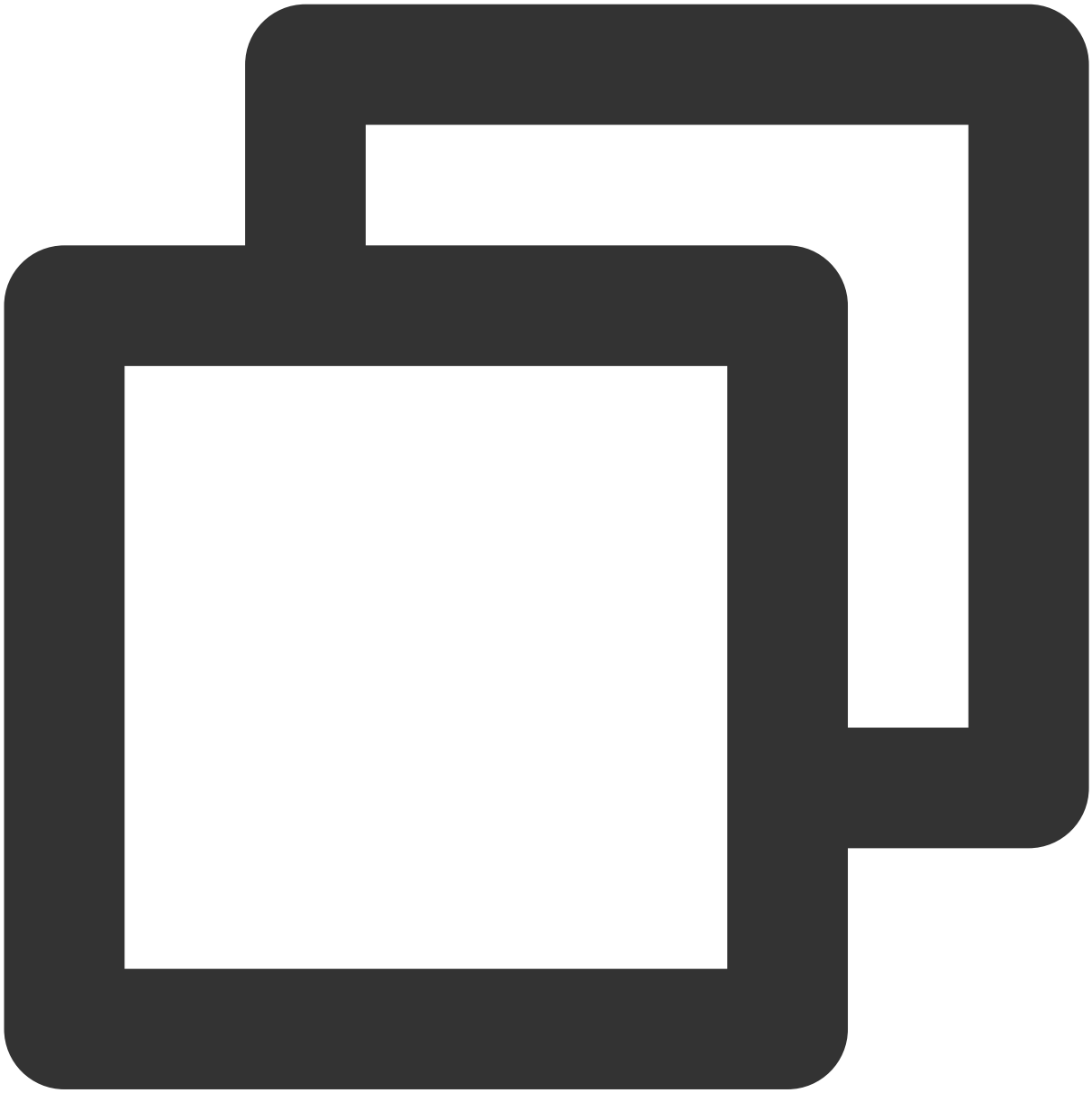
参数如下表所示：

参数	类型	含义
volumeMap	Map<String,	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小

	Integer>	值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。



```
void onUserNetworkQualityChanged(List<TUICallDefine.NetworkQualityInfo> networkQual
```

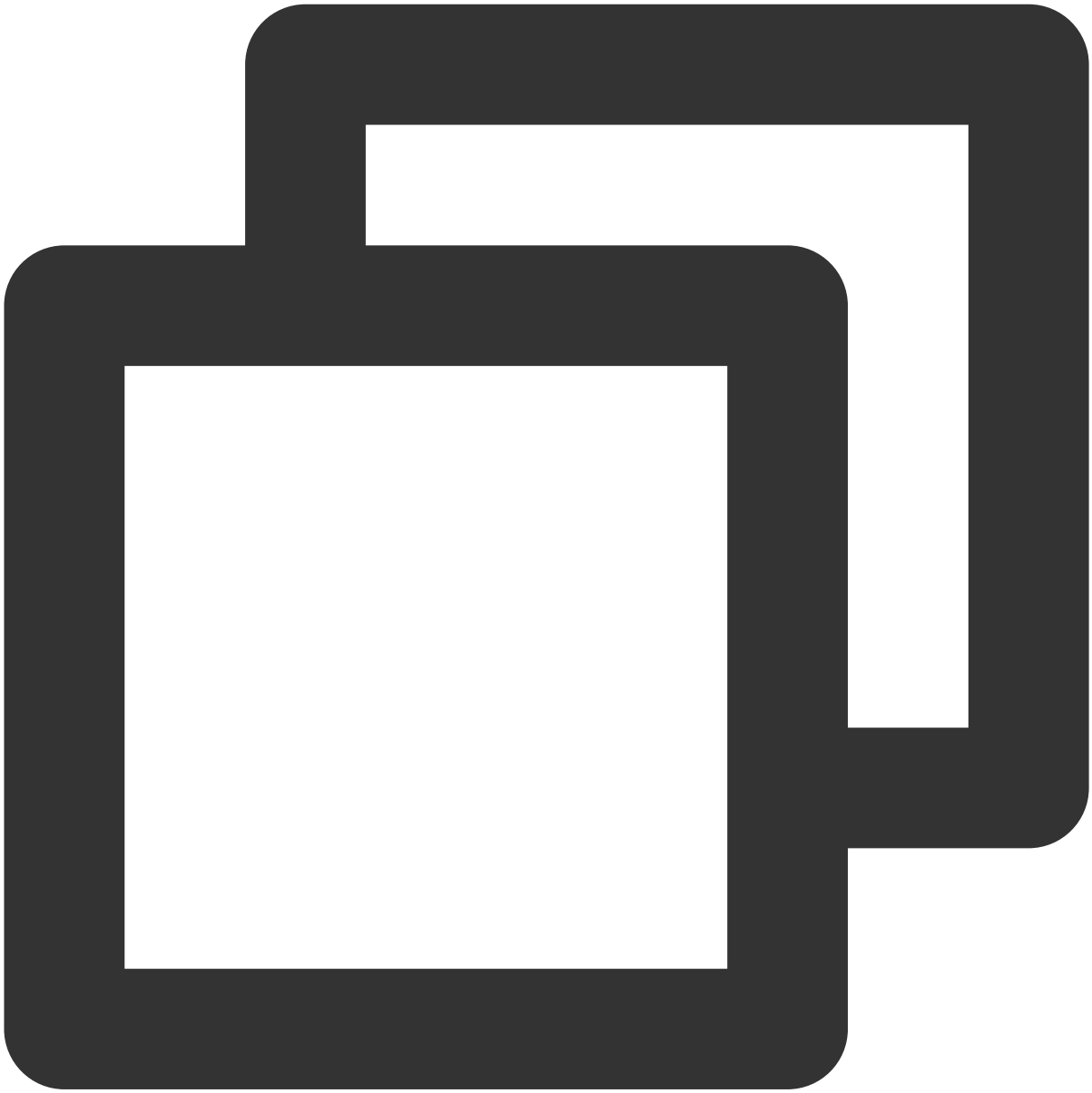
参数如下表所示：

参数	类型	含义

networkQualityList	List	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

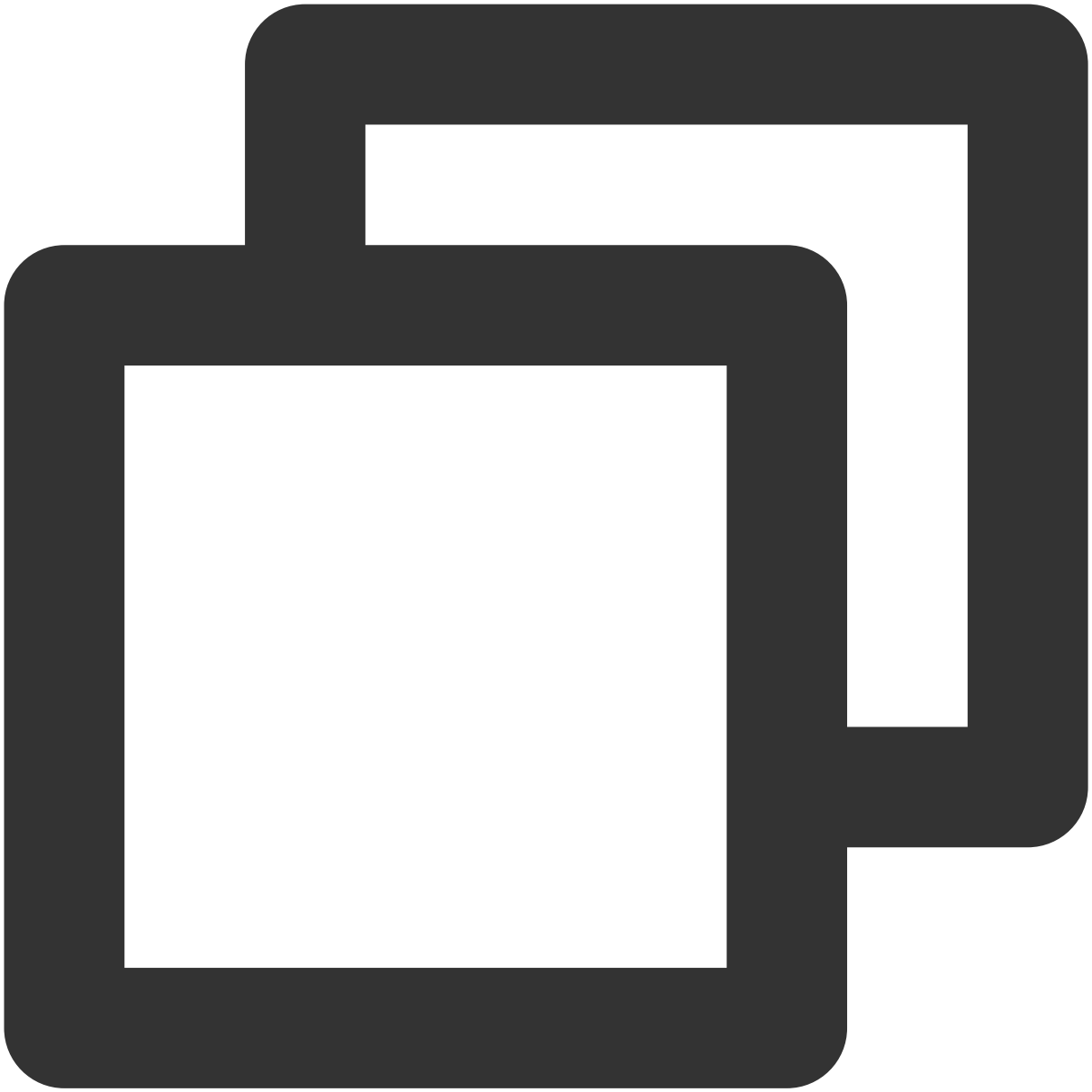
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
void onKickedOffline();
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。



```
void onUserSigExpired();
```

类型定义

最近更新时间：2023-06-16 16:27:14

常用结构

TUICallDefine

类型	描述
CallParams	通话参数
OfflinePushInfo	离线推送厂商配置信息

TUICommonDefine

类型	描述
RoomId	通话中音视频房间 ID
NetworkQualityInfo	网络质量
VideoRenderParams	视频画面的渲染参数
VideoEncoderParams	视频编码参数

枚举定义

TUICallDefine

类型	说明
MediaType	通话类型
Role	通话角色
Status	通话状态
Scene	通话场景
IOSOfflinePushType	iOS 离线推送类型

TUICommonDefine

类型	说明

版权所有：腾讯云计算（北京）有限责任公司

AudioPlaybackDevice	音频设备
Camera	摄像头类型
NetworkQuality	网络质量
FillMode	视频画面填充模式
Rotation	视频画面旋转方向
ResolutionMode	视频宽高比模式
Resolution	视频分辨率

CallParams

通话参数

类型	类型	描述
roomId	TUICommonDefine.RoomId	通话中音视频房间ID
offlinePushInfo	OfflinePushInfo	厂商离线推送配置信息
timeout	int	通话超时时间，默认：30s，单位：秒
userData	String	发起通话时自定义扩展字段，被叫端在 onCallReceived 中回调

OfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参考：离线唤醒

类型	类型	描述
title	String	离线推送展示通知栏标题
desc	String	离线推送展示通知栏内容
ignoreIOSBadge	boolean	离线推送忽略 badge 计数（仅对 iOS 生效）， 如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加。
iOSSound	String	离线推送声音设置（仅对 iOS 生效）。 当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。 当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。 如果要自定义 iOSSound，需要先把

版权所有：腾讯云计算（北京）有限责任公司

		语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	String	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。 只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声。 另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效
androidOPPOChannelID	String	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidVIVOClassification	int	VIVO 推送消息分类 (待废弃接口，VIVO 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1
androidXiaoMiChannelID	String	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	String	FCM 通道手机 8.0 系统及以上的渠道 ID
androidHuaWeiCategory	String	华为推送消息分类，详见：华为消息分类标准
isDisablePush	boolean	是否关闭推送（默认开启推送）
iOSPushType	IOSOfflinePushType	iOS 离线推送类型，默认：APNs

RoomId

通话中音视频房间ID

(1) `intRoomId` 与 `strRoomId` 是互斥的，若您选用 `strRoomId` ，则 `intRoomId` 需要填写为 0。若两者都填，SDK 将优先选用 `intRoomId` 。

(2) 不要混用 `intRoomId` 和 `strRoomId` ，因为它们之间是不互通的，比如数字123和字符串123是两个完全不同的房间。

类型	类型	描述
intRoomId	int	数字房间号 **取值范围** 1 - 2147483647(2^31-1)
strRoomId	String	字符串房间号 **推荐取值** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： 大小写英文字母（a-zA-Z）； 数字（0-9）；

		空格、!、#、$、%、&、(、)、+、-、:、;、<、=、

注意：

字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！

NetworkQualityInfo

用户网络质量

类型	类型	描述
userId	String	用户 ID
quality	NetworkQuality	网络质量

VideoRenderParams

视频画面的渲染参数

类型	类型	描述
fillMode	VideoRenderParams.FillMode	视频画面填充模式
rotation	VideoRenderParams.Rotation	视频画面旋转方向

VideoEncoderParams

视频编码参数

类型	类型	描述
resolution	VideoEncoderParams.Resolution	视频分辨率
resolutionMode	VideoEncoderParams.ResolutionMode	视频宽高比

MediaType

通话类型

类型	取值	描述
Unknown	0	未知类型
Audio	1	语音通话
Video	2	视频通话

版权所有：腾讯云计算（北京）有限责任公司

Role

通话角色

类型	取值	描述
None	0	未知类型
Caller	1	主叫（邀请方）
Called	2	被叫（被邀请方）

Status

通话状态

类型	取值	描述
None	0	未定义
Waiting	1	通话等待中
Accept	2	通话已接听

Scene

通话场景

类型	取值	描述
GROUP_CALL	0	群组通话
MULTI_CALL	1	匿名群组通话（暂不支持，敬请期待）
SINGLE_CALL	2	单人通话

IOSOfflinePushType

iOS离线推送类型

类型	取值	描述
APNs	0	普通推送
VoIP	1	VoIP 推送

AudioPlaybackDevice

音频设备

类型	取值	描述
Earpiece	0	听筒
Speakerphone	1	扬声器

Camera

前置/后置 摄像头

类型	取值	描述
Front	0	前置摄像头
Back	1	后置摄像头

NetworkQuality

网络质量

类型	取值	描述
Unknown	0	未定义
Excellent	1	当前网络非常好
Good	2	当前网络比较好
Poor	3	当前网络一般
Bad	4	当前网络较差
Vbad	5	当前网络很差
Down	6	当前网络不满足通话的最低要求

FillMode

视频画面填充模式

类型	取值	描述
Fill	0	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。
Fit	1	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

版权所有：腾讯云计算（北京）有限责任公司

Rotation

视频画面旋转方向

类型	取值	描述
Rotation_0	0	不旋转
Rotation_90	1	顺时针旋转90度
Rotation_180	2	顺时针旋转180度
Rotation_270	3	顺时针旋转270度

ResolutionMode

视频宽高比模式

类型	取值	描述
Landscape	0	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360。
Portrait	1	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640。

Resolution

视频分辨率

类型	取值	描述
Resolution_640_360	108	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps
Resolution_640_480	62	宽高比 4:3；分辨率 640x480；建议码率（VideoCall）600kbps
Resolution_960_540	110	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps
Resolution_960_720	64	宽高比 4:3；分辨率 960x720；建议码率（VideoCall）1000kbps
Resolution_1280_720	112	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps
Resolution_1920_1080	114	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps

版权所有：腾讯云计算（北京）有限责任公司

iOS
API 概览

最近更新时间：2023-06-21 16:41:14

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的头像、昵称
call	发起 1v1 通话
call	发起 1v1 通话，支持自定义房间号、通话邀请超时时间，离线推送内容等
groupCall	发起群组通话
groupCall	发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）

destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）
setSelfInfo	设置用户的头像、昵称
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数

版权所有：腾讯云计算（北京）有限责任公司

getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话的媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

关键类型定义

API	描述
TUICallMediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallRole	通话的角色，枚举类型：主叫、被叫
TUICallStatus	通话的状态，枚举类型：空闲、待接听、接听中
TUICallParams	通话扩展参数：离线推送消息显示内容
TUIRoomId	音视频房间Id，支持数字、字符串两种类型
TUICamera	摄像头Id参数，枚举类型：前摄、后摄
TUIAudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUINetworkQuality	当前的网络质量信息

TUICallKit

最近更新时间：2023-06-21 16:42:33

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入TUICallKit。

API 概览

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的昵称、头像
call	发起 1v1 通话
call	发起 1v1 通话，支持自定义离线推送消息显示内容
groupCall	发起群组通话
groupCall	发起群组通话，支持自定义离线推送消息显示内容
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

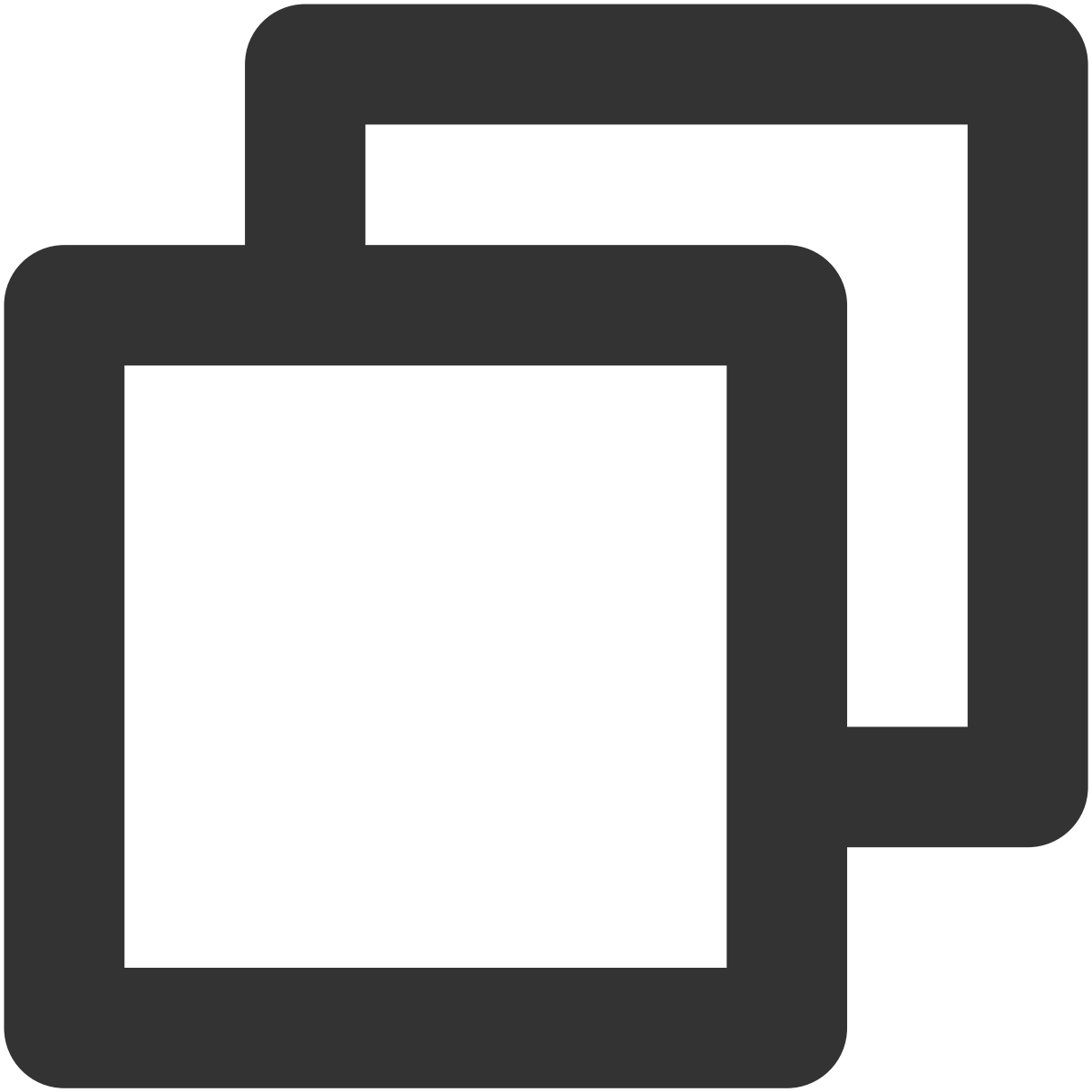
API 详情

createInstance

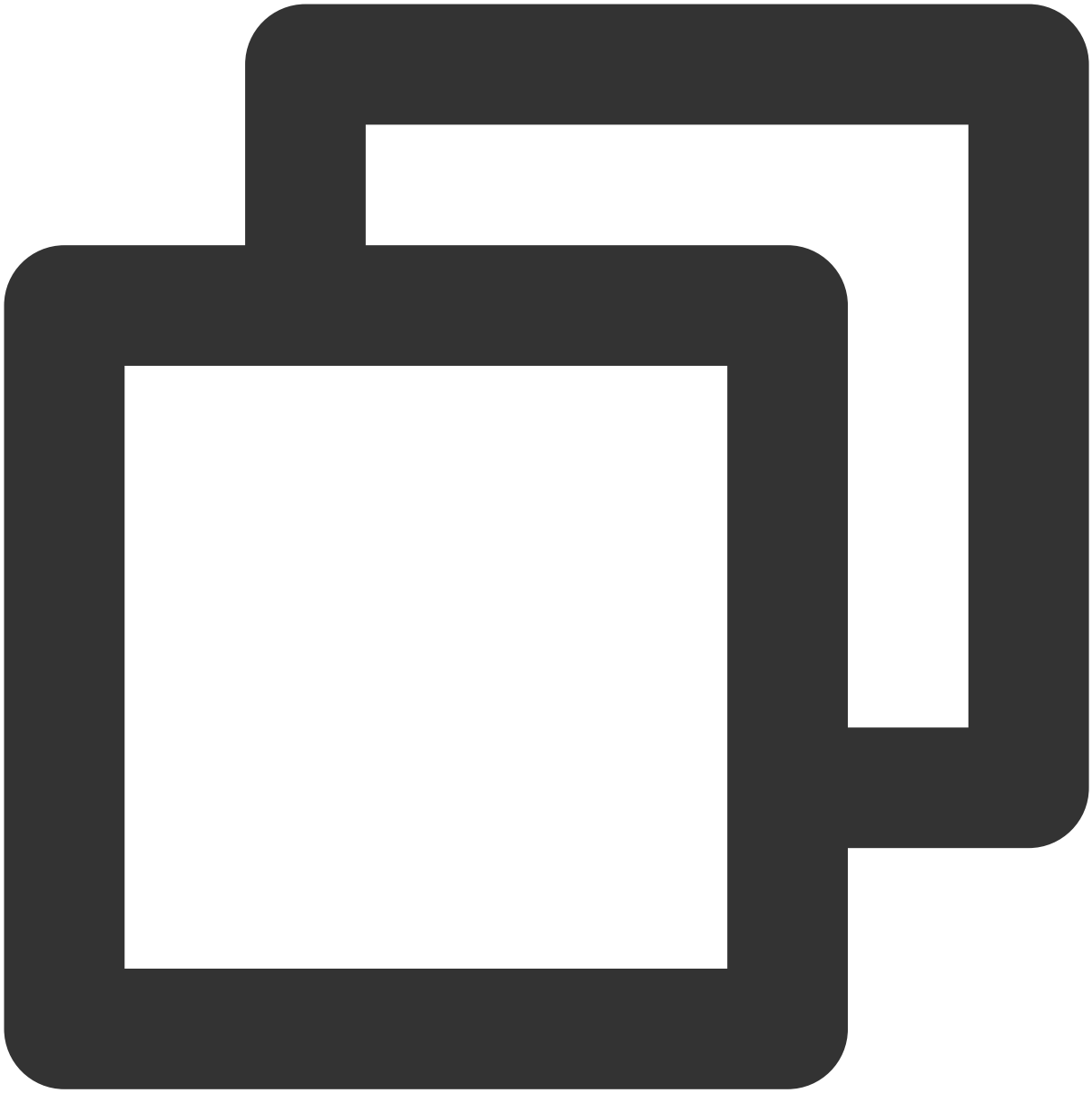
创建 TUICallKit 的单例。

Objective-C

Swift



```
- (instancetype)createInstance;
```



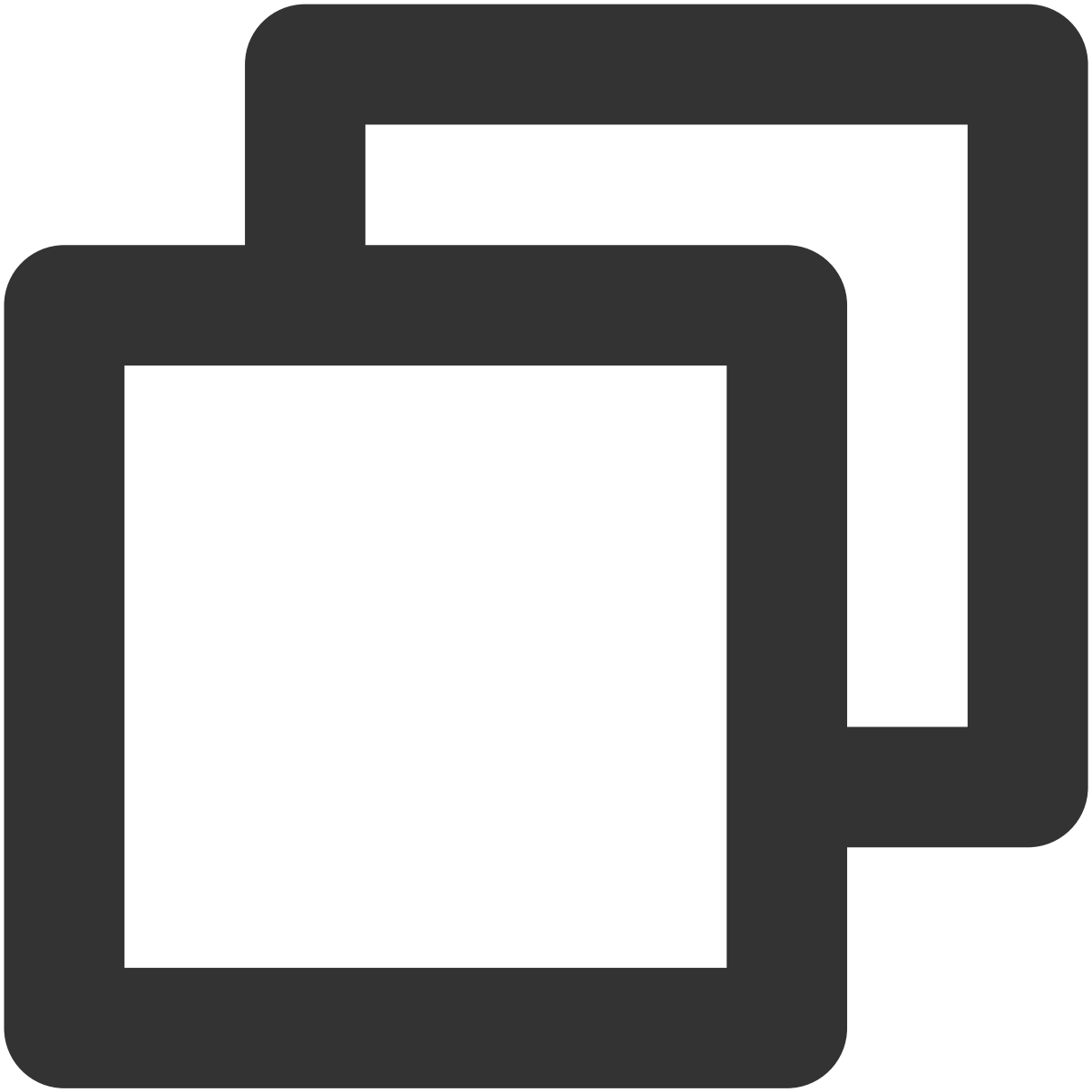
```
public static func createInstance() -> TUICallKit
```

setSelfInfo

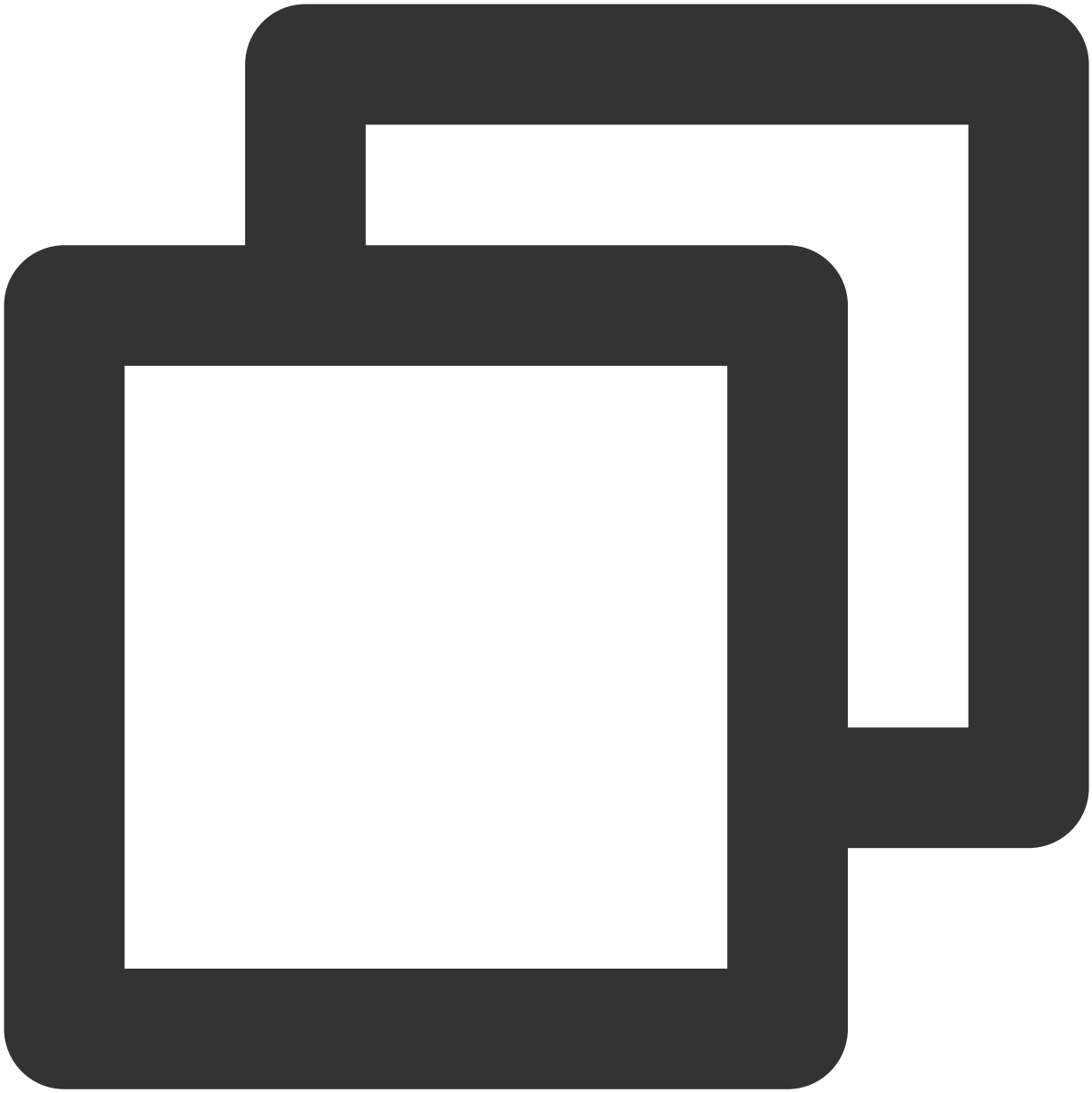
设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

Objective-C

Swift



```
- (void)setSelfInfo:(NSString * _Nullable)nickname avatar:(NSString * _Nullable)ava
```



```
public func setSelfInfo(nickname: String, avatar: String, succ:@escaping TUICallSuc
```

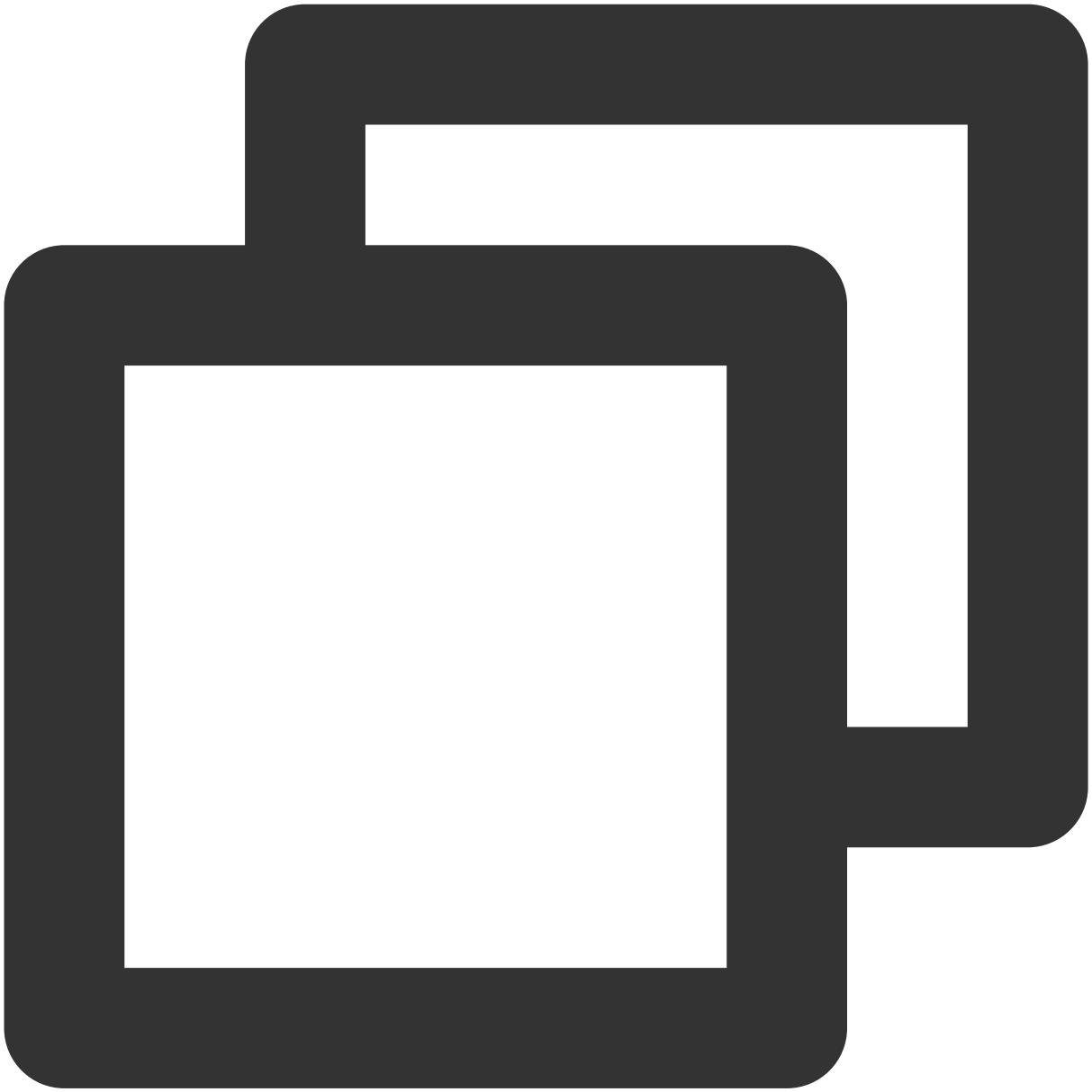
参数	类型	含义
nickName	NSString	目标用户的昵称
avatar	NSString	目标用户的头像

call

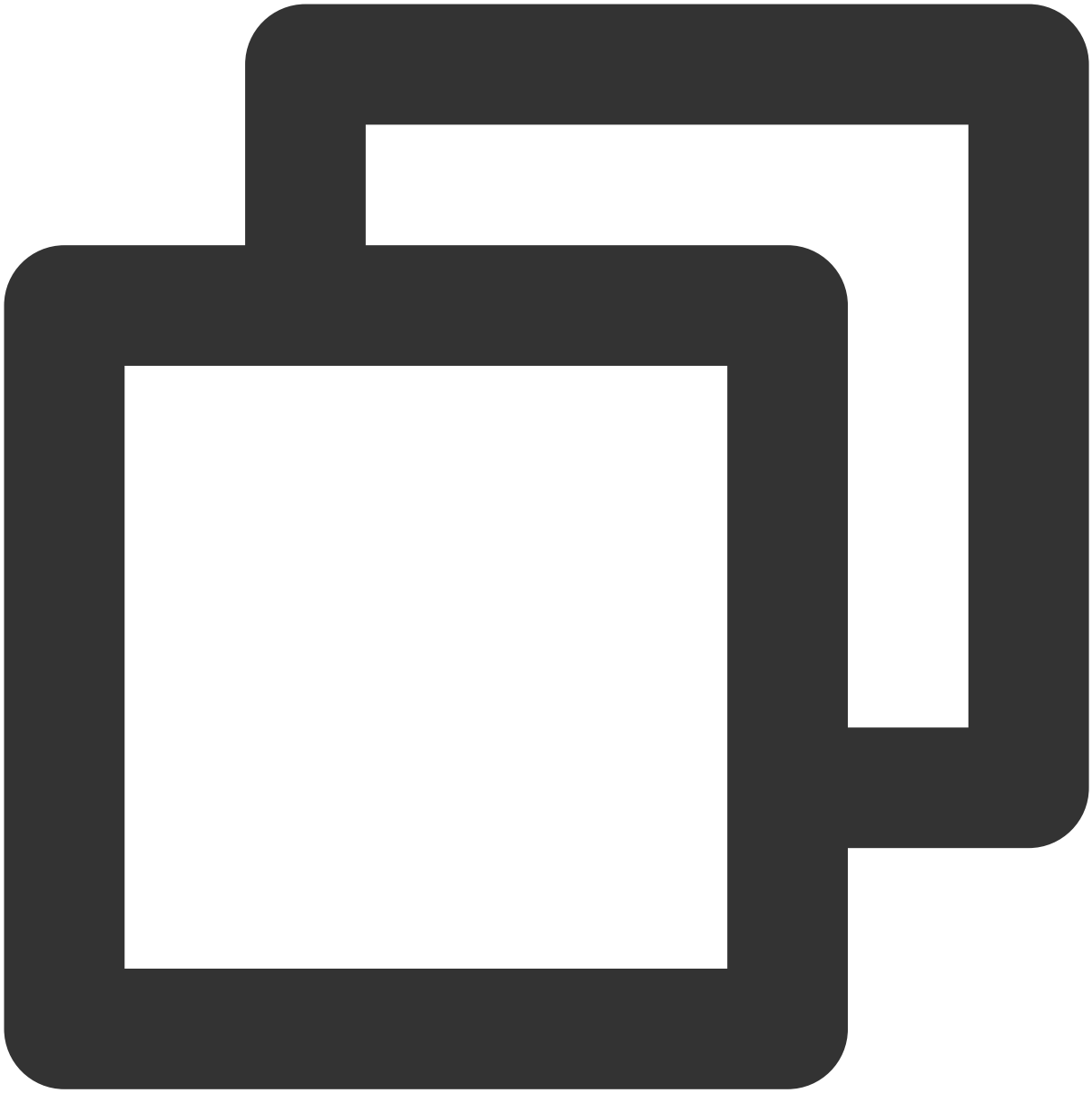
拨打电话（1v1通话）

Objective-C

Swift



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType;
```



```
public func call(userId: String, callMediaType: TUICallMediaType)
```

参数如下表所示：

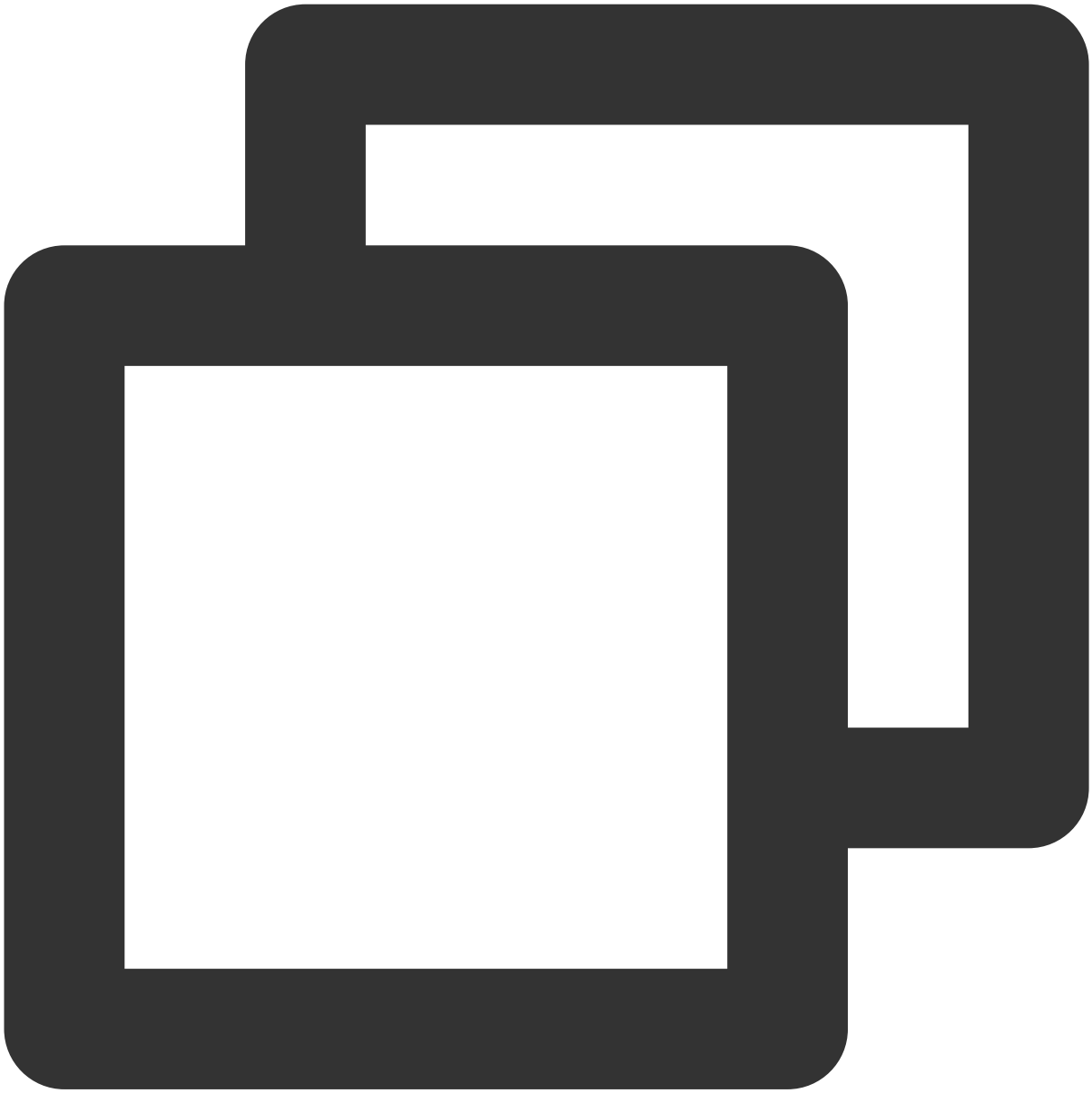
参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

call

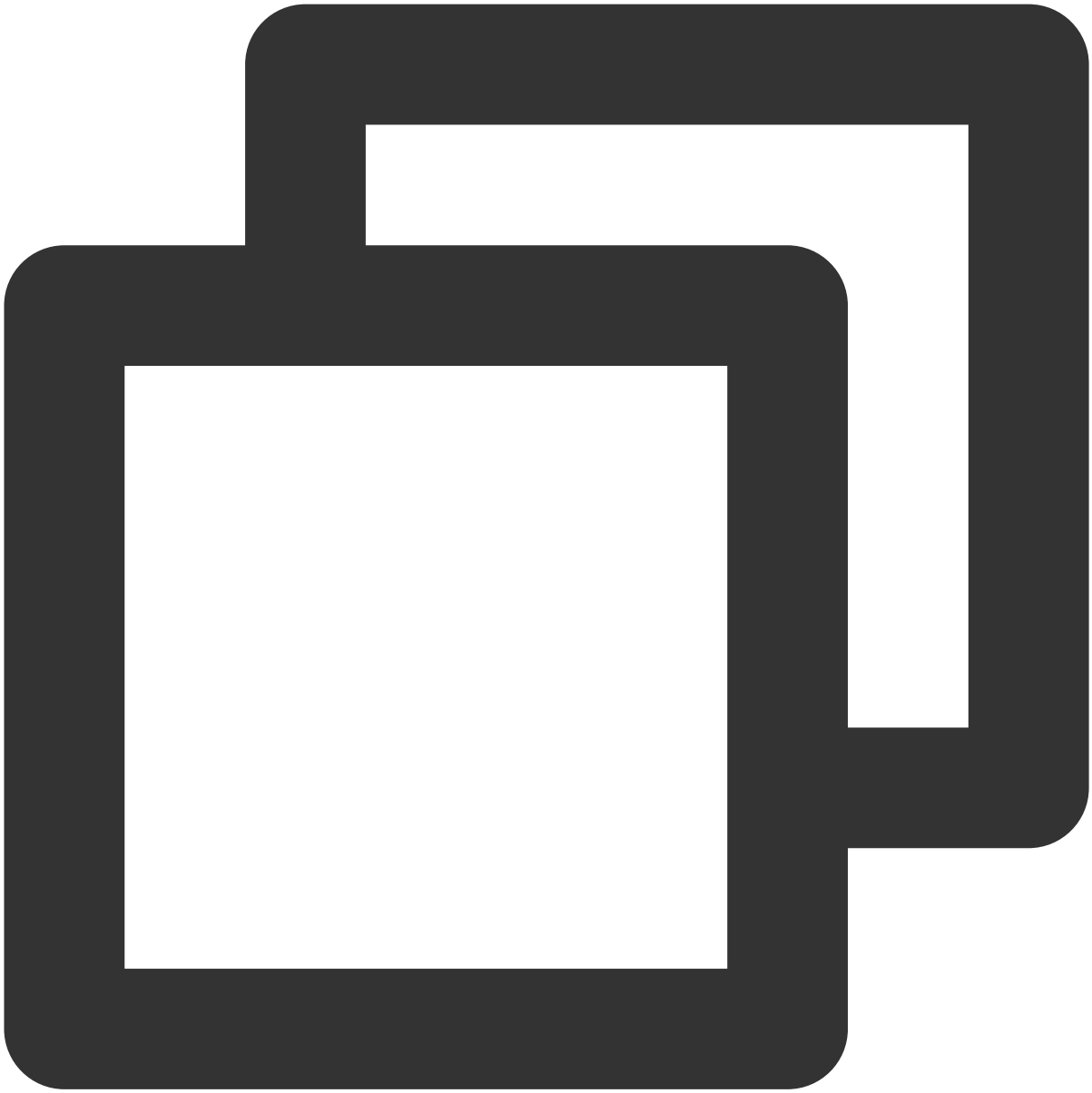
拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。

Objective-C

Swift



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType param
```



```
public func call(userId: String, callMediaType: TUICallMediaType, params: TUICallPa
                 succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

groupCall

发起群组通话

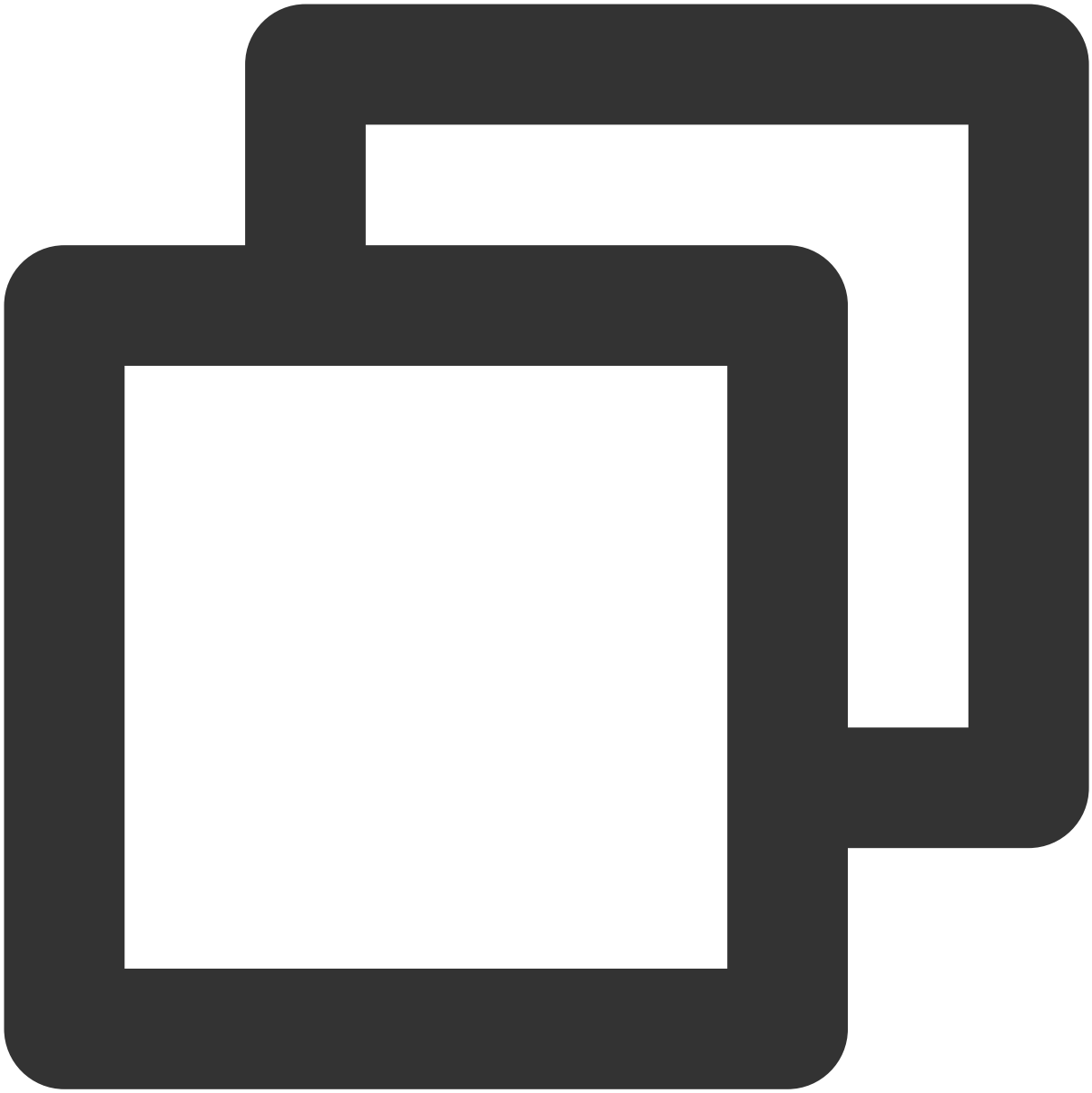
注意：

使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

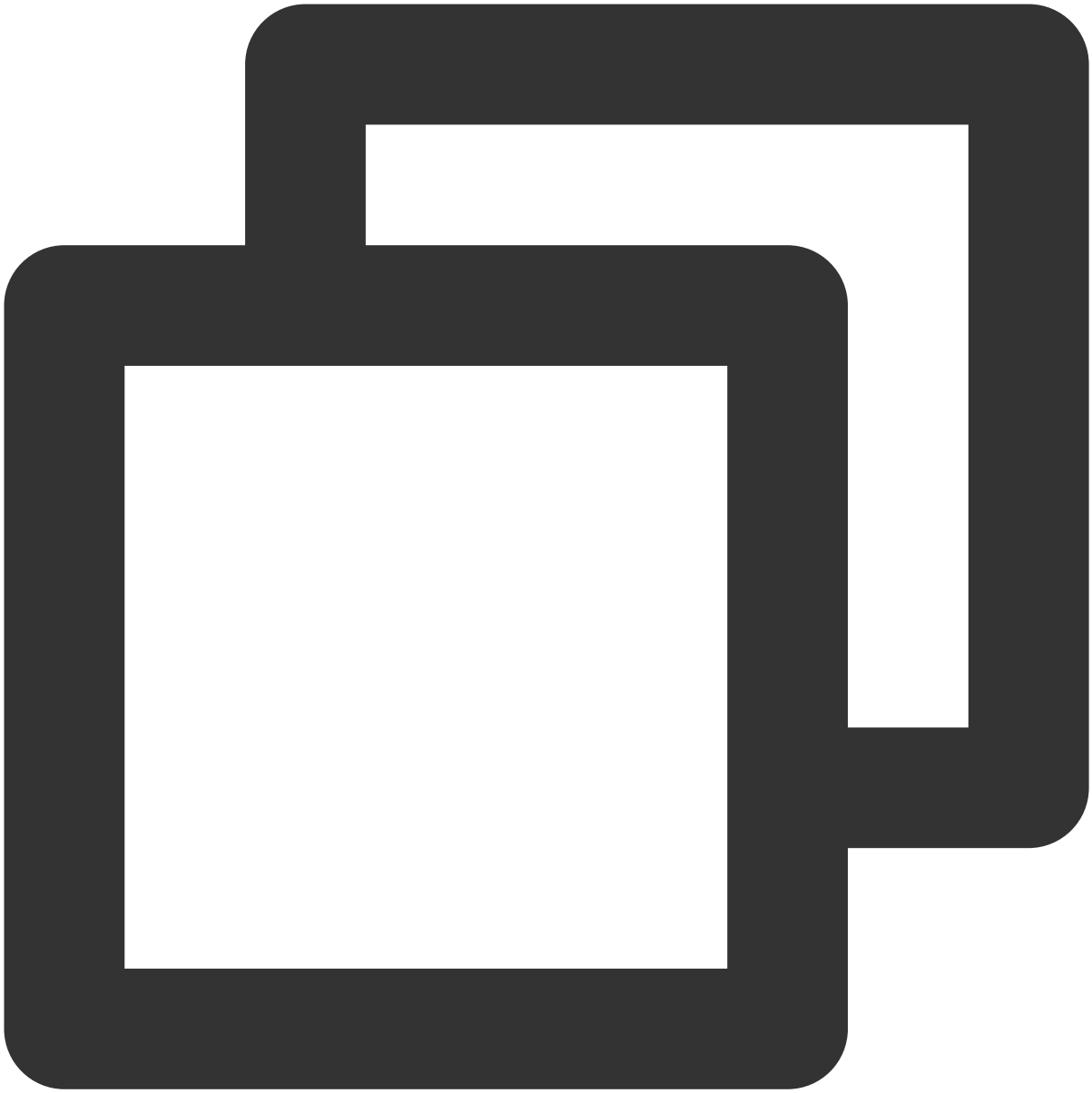
群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

Swift



```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList
```



```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

groupCall

发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等。

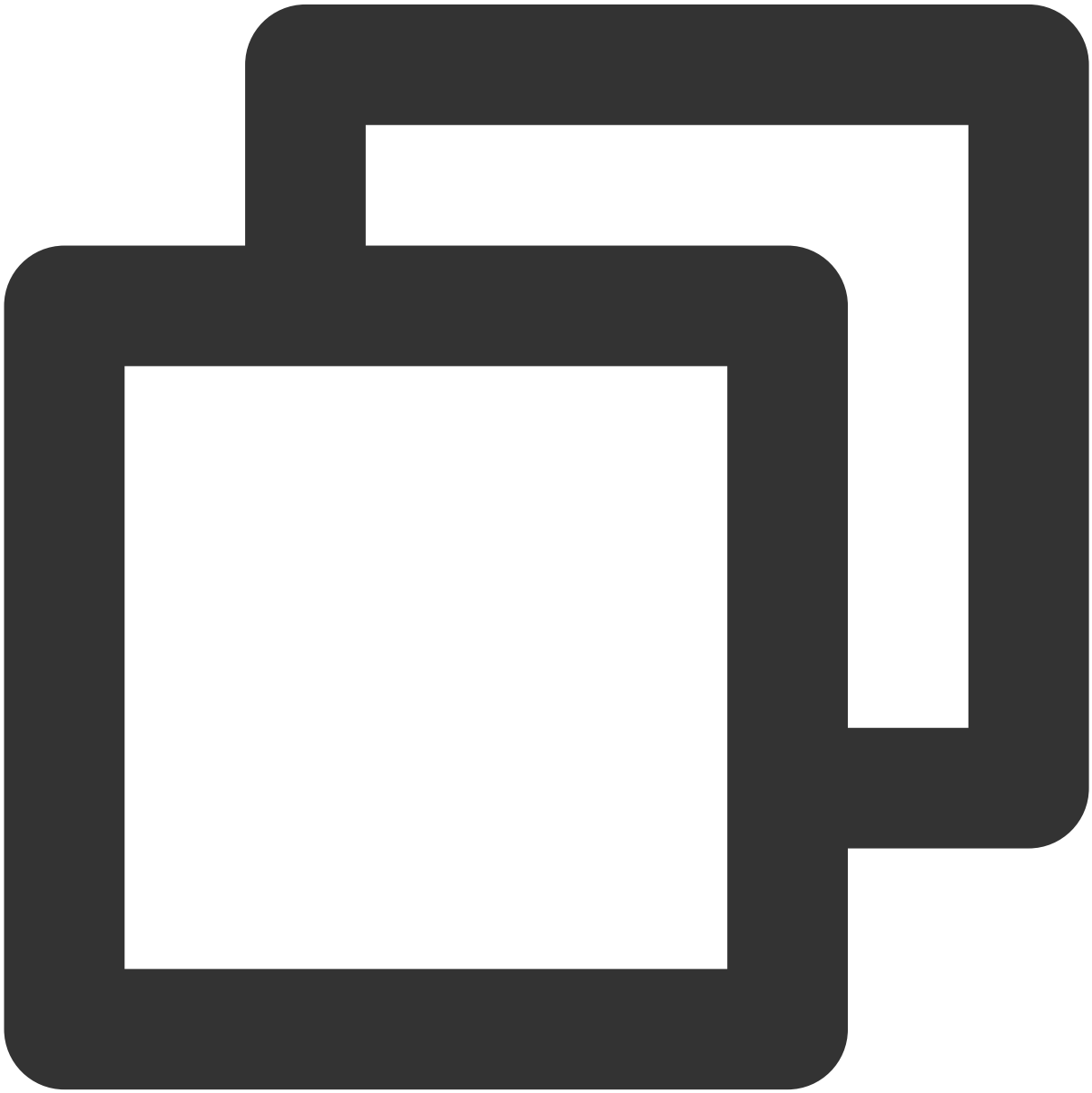
注意：

使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

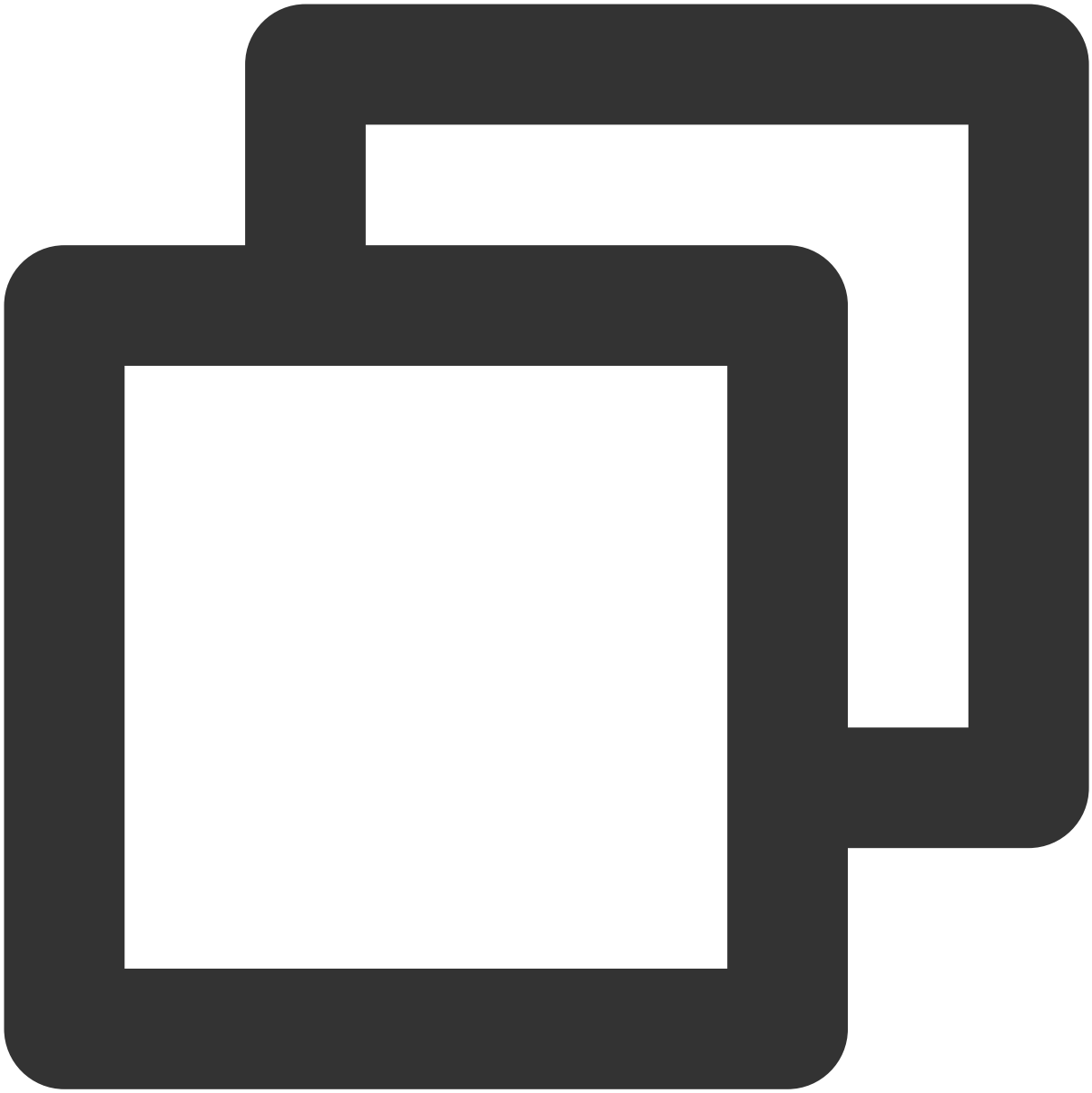
群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

Swift



```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList
```



```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICall
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时

		间，离线推送自定义内容等

joinInGroupCall

发起群组通话

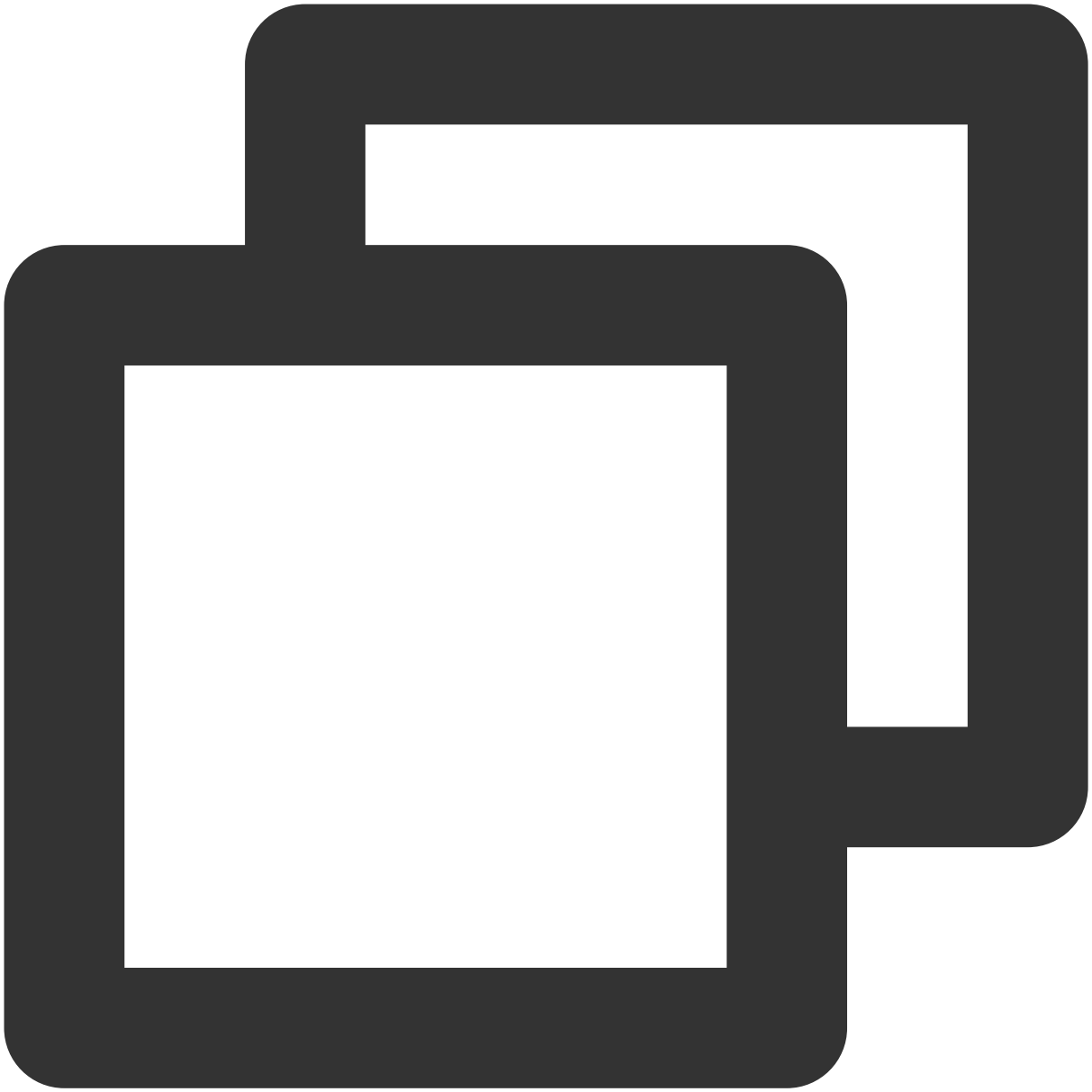
注意：

加入群组中已有的音视频通话前，需要提前创建或加入 IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。

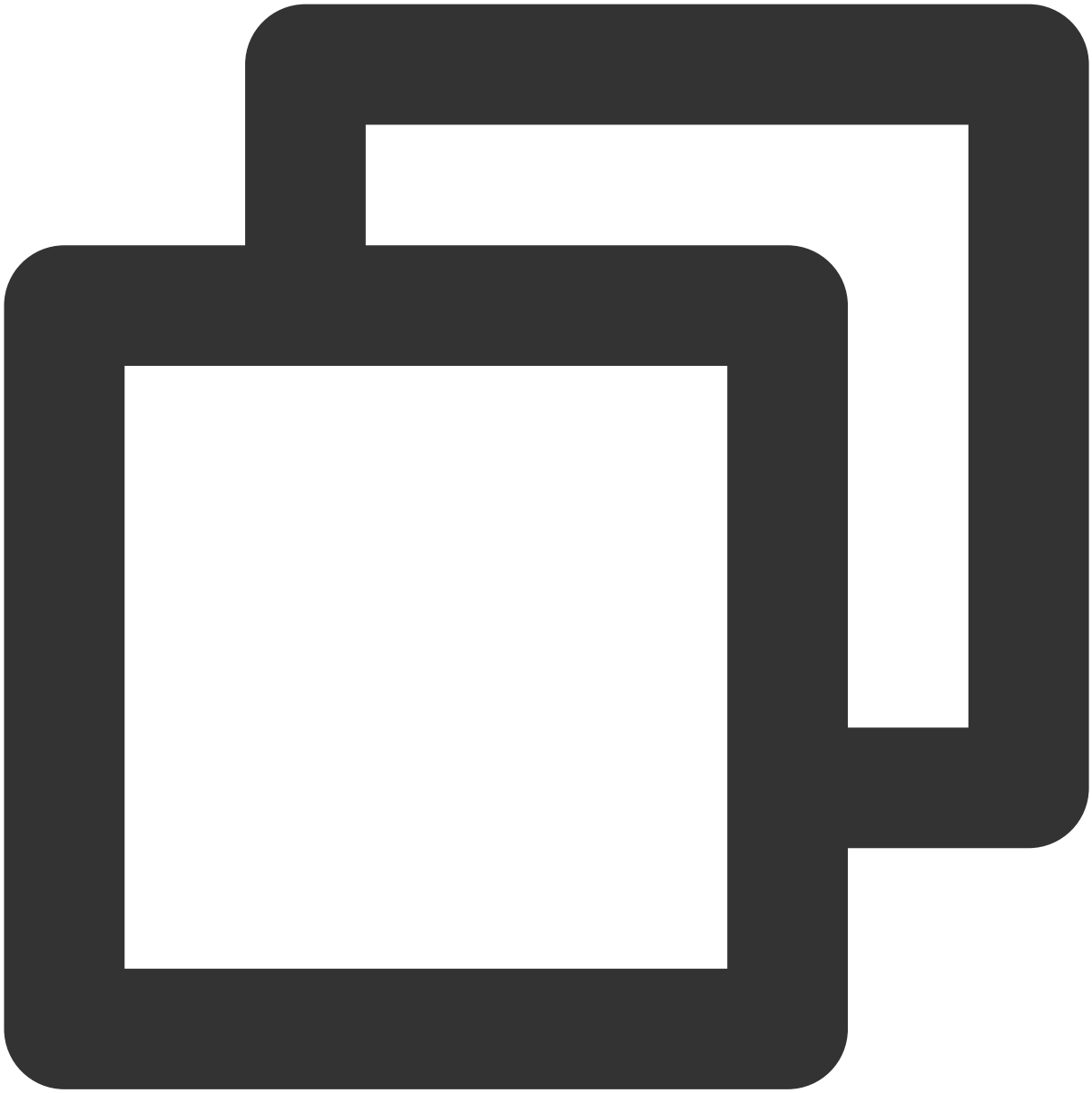
群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

Swift



```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```



```
public func joinInGroupCall(roomId: TUIRoomId, groupId: String, callMediaType: TUIC
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID
groupId	NSString	此次群组通话的群 ID
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

setCallingBell

设置自定义来电铃音。

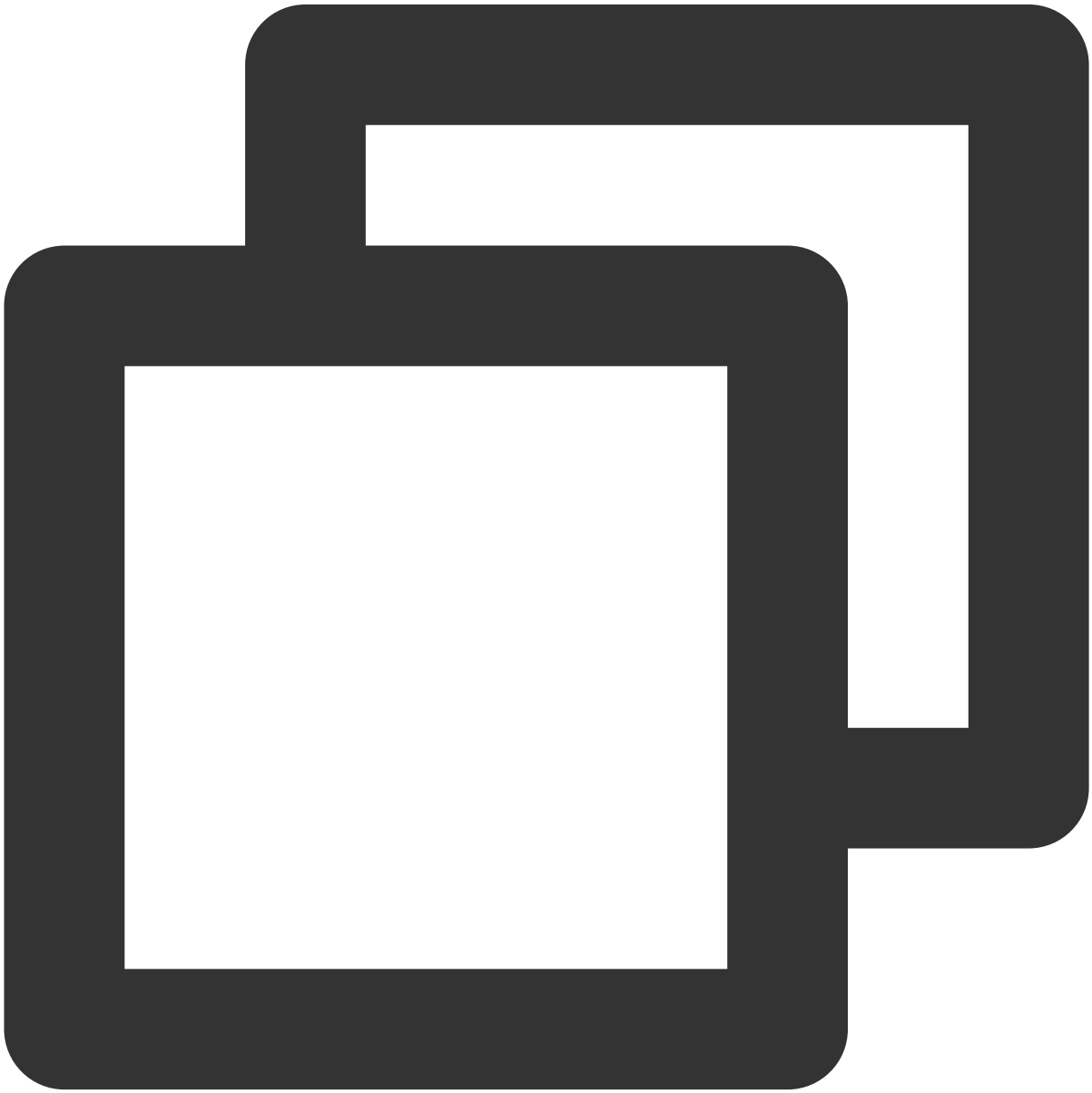
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

铃声设置后与设备绑定，更换用户，铃声依旧会生效。

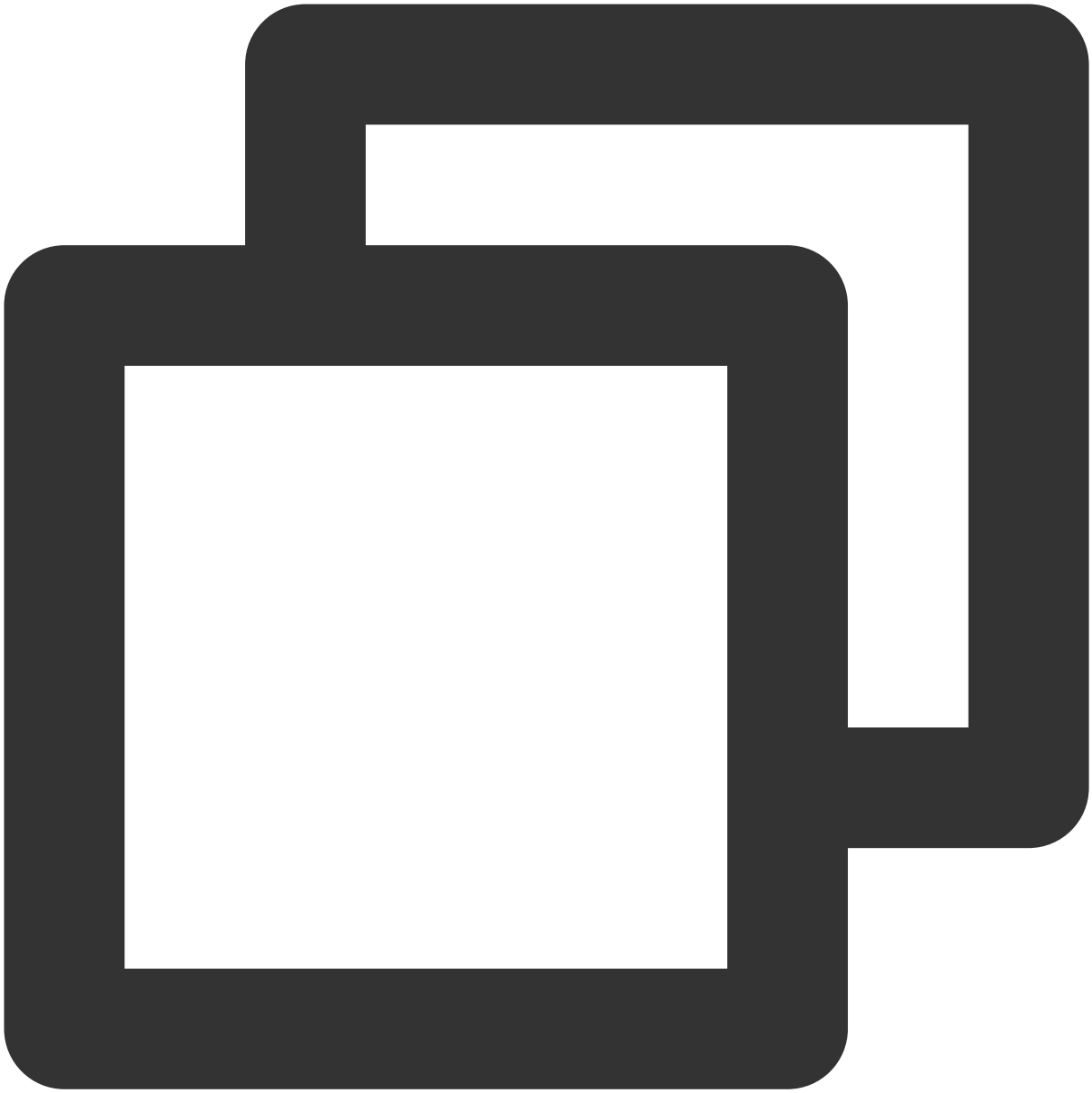
如需恢复默认铃声， `filePath` 传空即可。

Objective-C

Swift



```
- (void)setCallingBell:(NSString *)filePath;
```



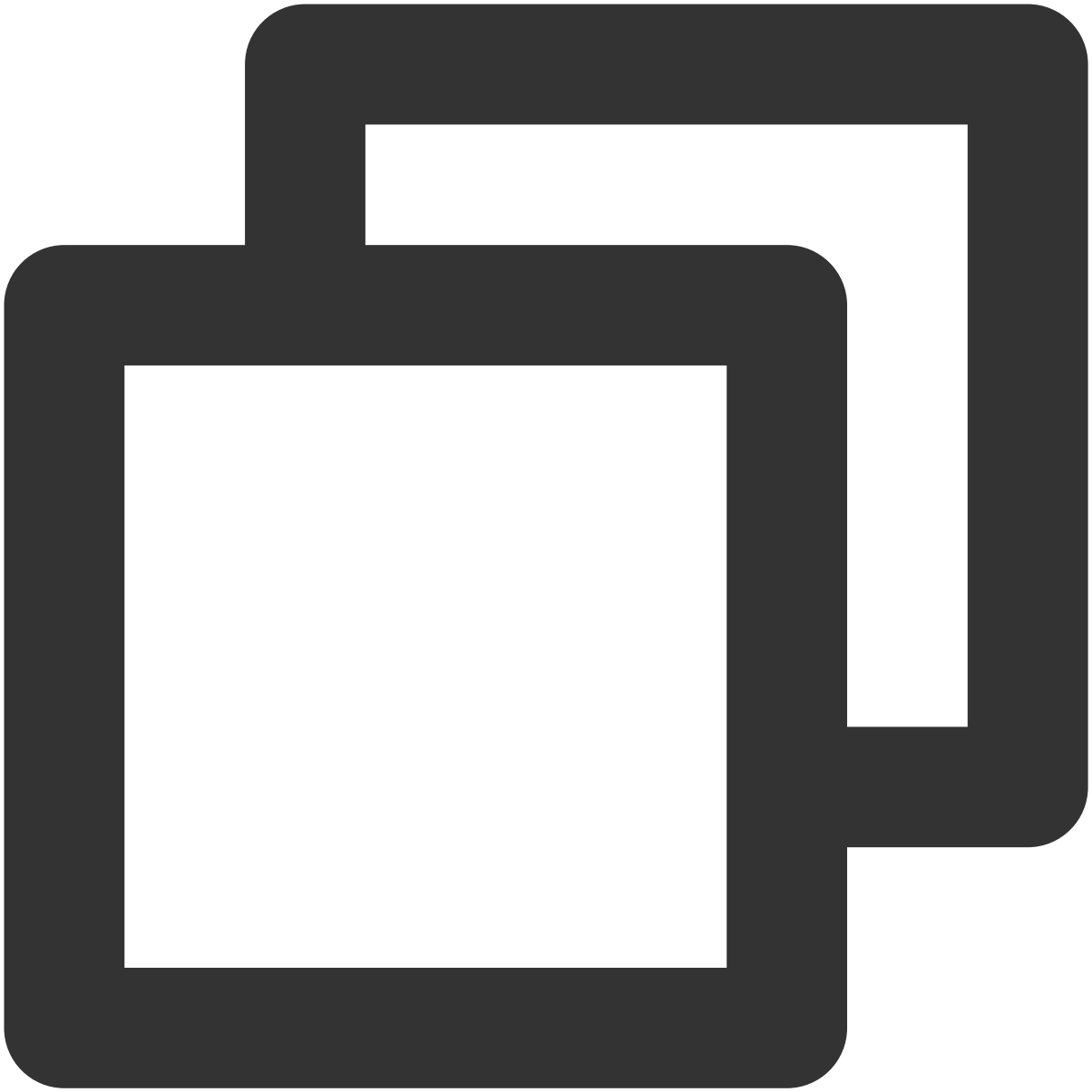
```
public func setCallingBell(filePath: String)
```

enableMuteMode

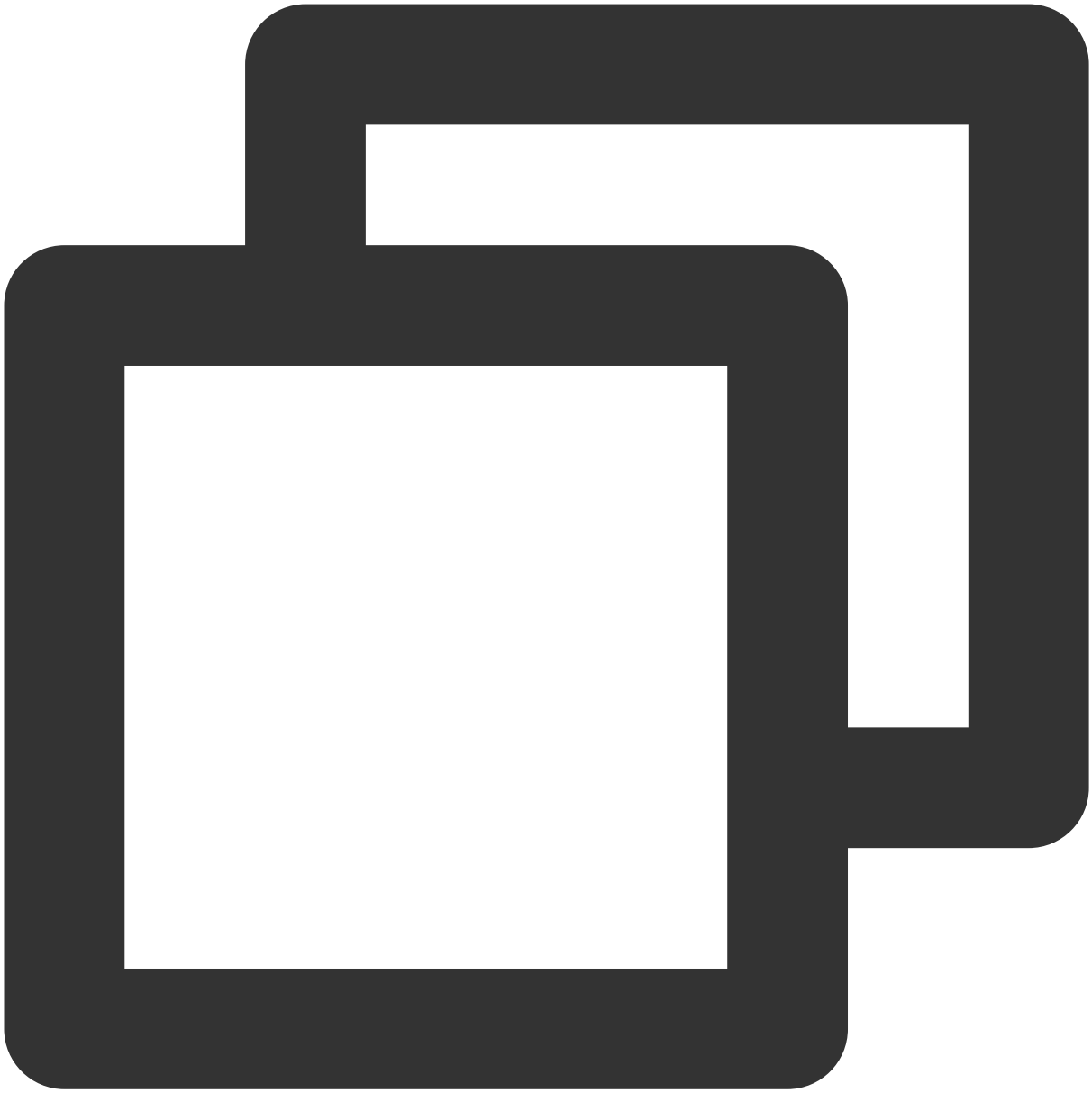
开启/关闭静音模式。

Objective-C

Swift



```
- (void)enableMuteMode:(BOOL)enable;
```



```
public func enableMuteMode(enable: Bool)
```

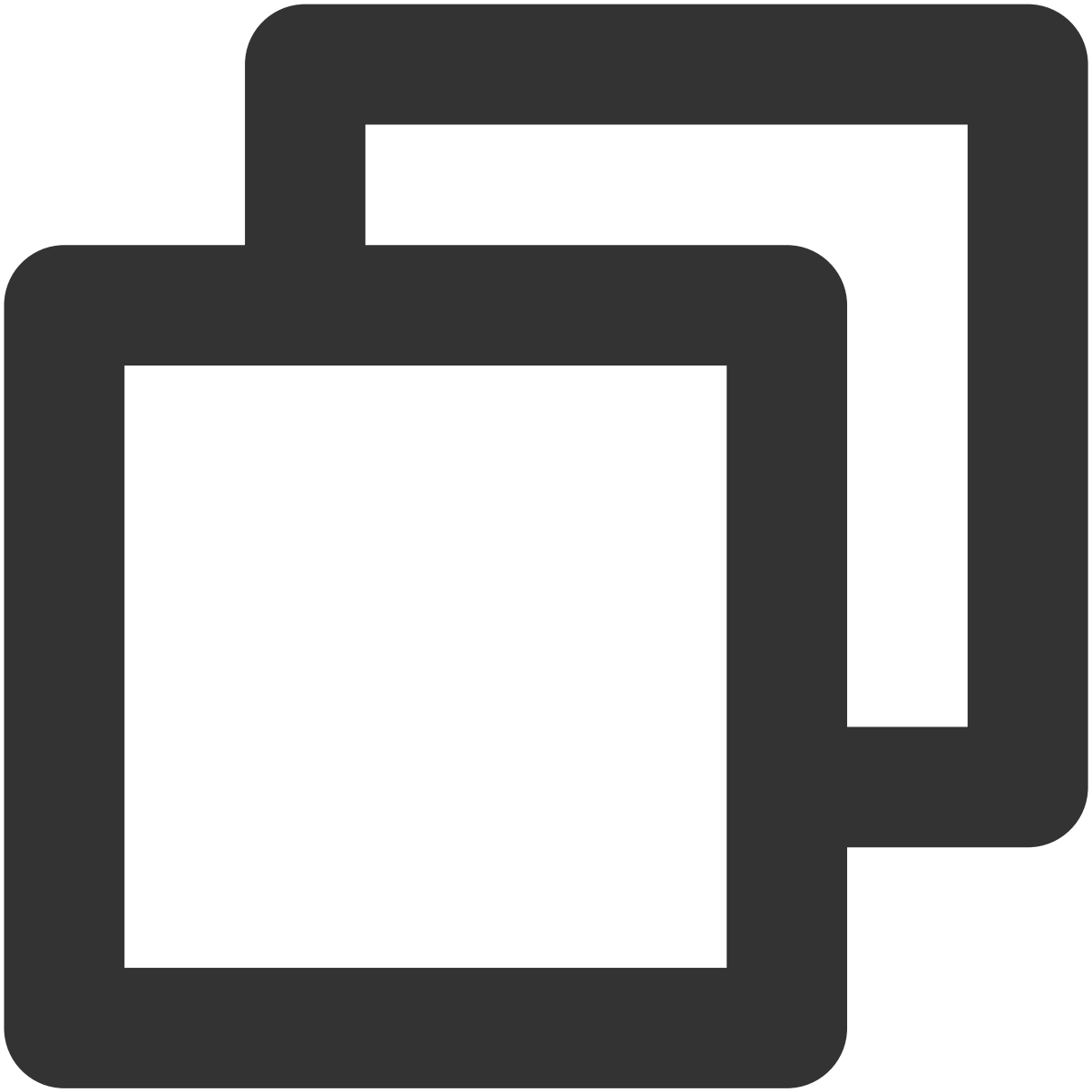
enableFloatWindow

开启/关闭悬浮窗功能。

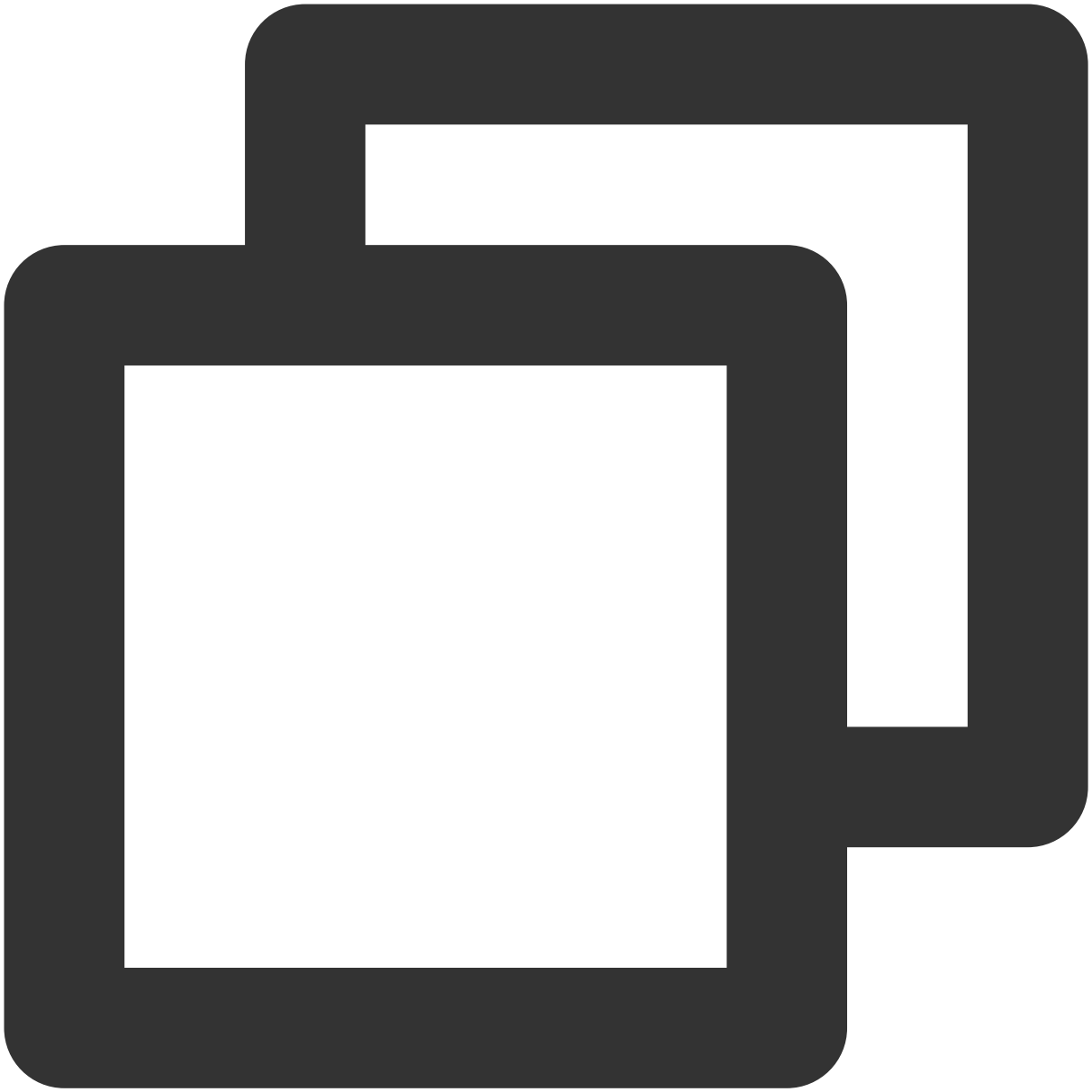
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Objective-C

Swift



```
- (void)enableFloatWindow:(BOOL)enable;
```



```
public func enableFloatWindow(enable: Bool)
```

TUICallEngine

最近更新时间：2024-01-18 11:29:15

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流

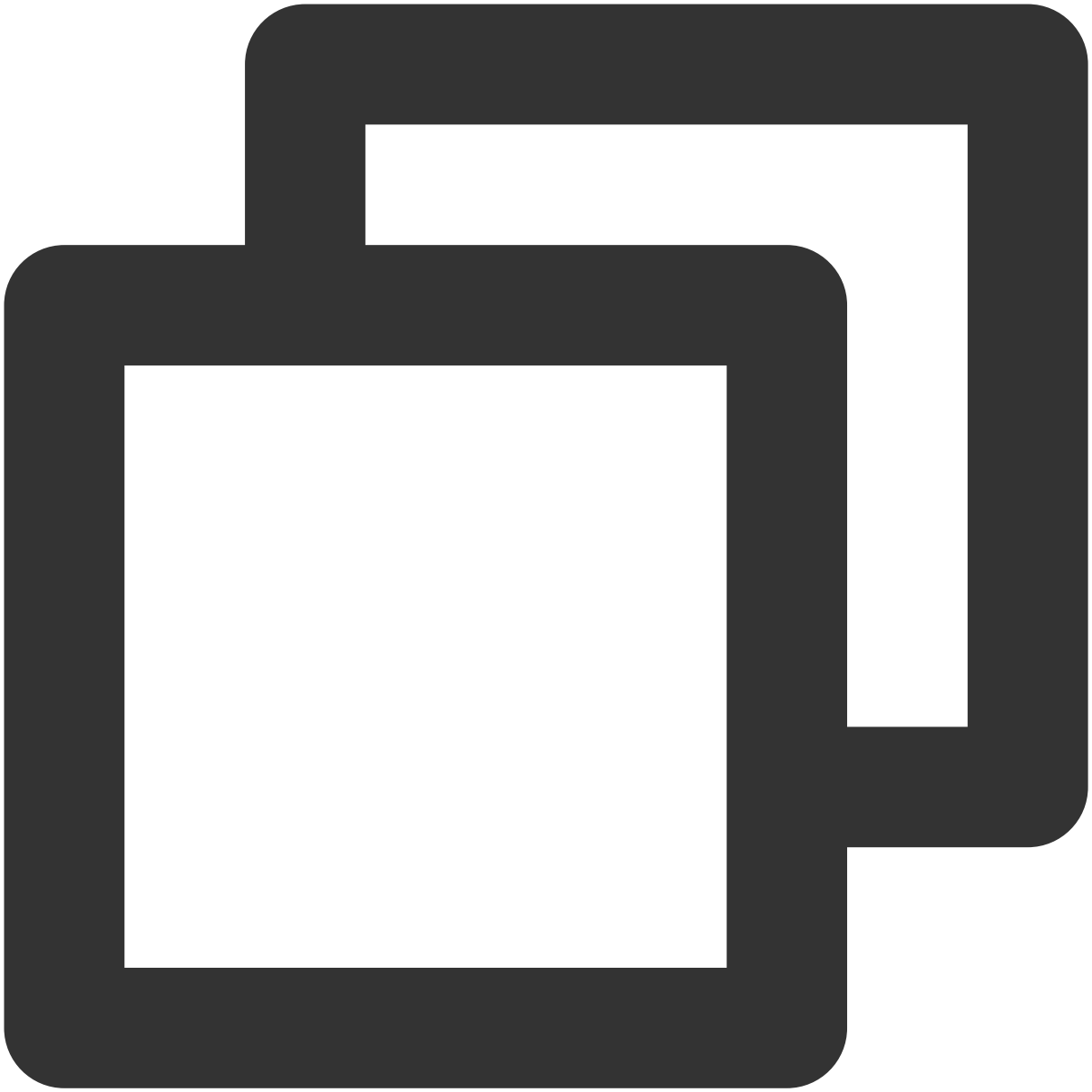
版权所有：腾讯云计算（北京）有限责任公司

openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

API 详情

createInstance

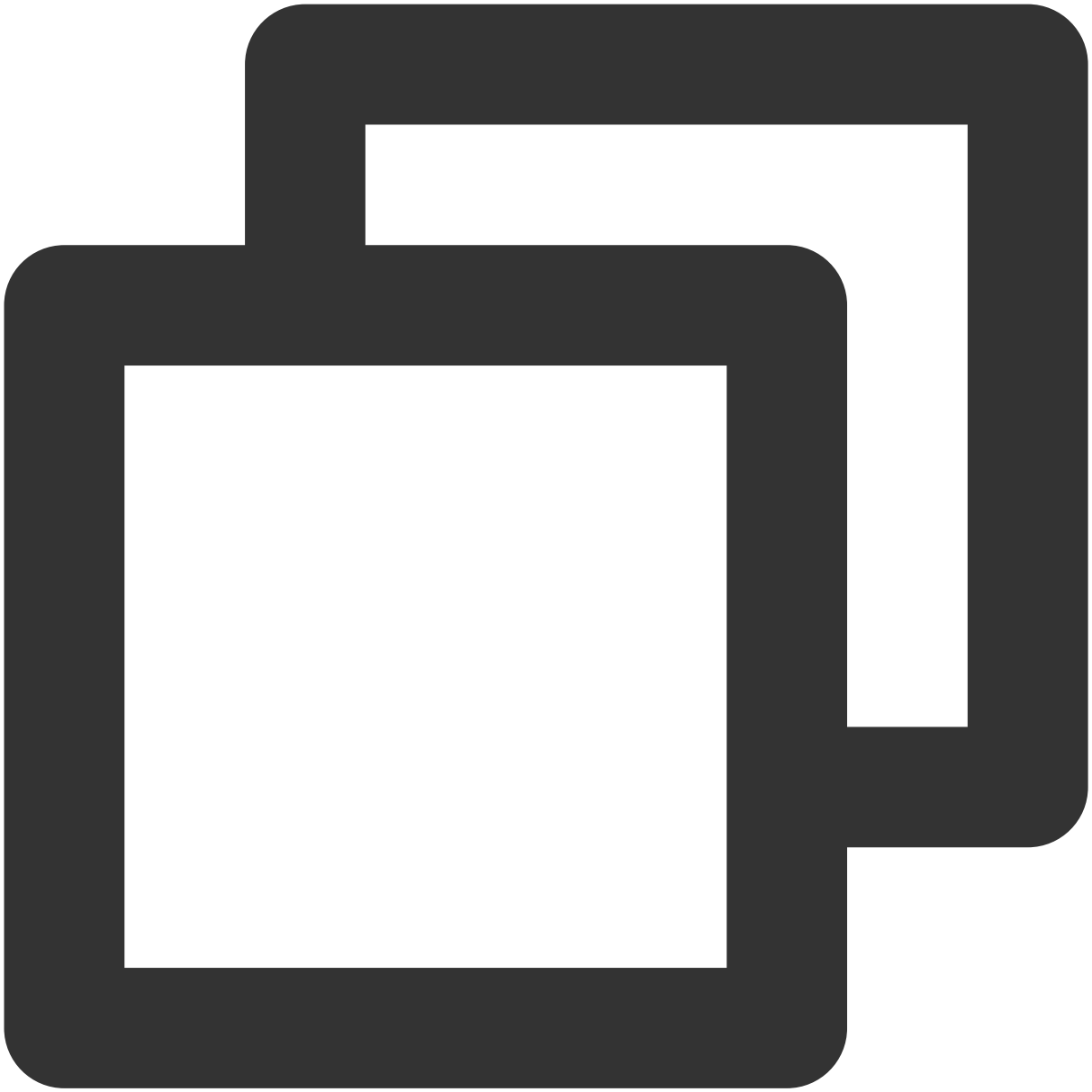
创建 TUICallEngine 的单例。



```
- (TUICallEngine *)createInstance;
```

destroyInstance

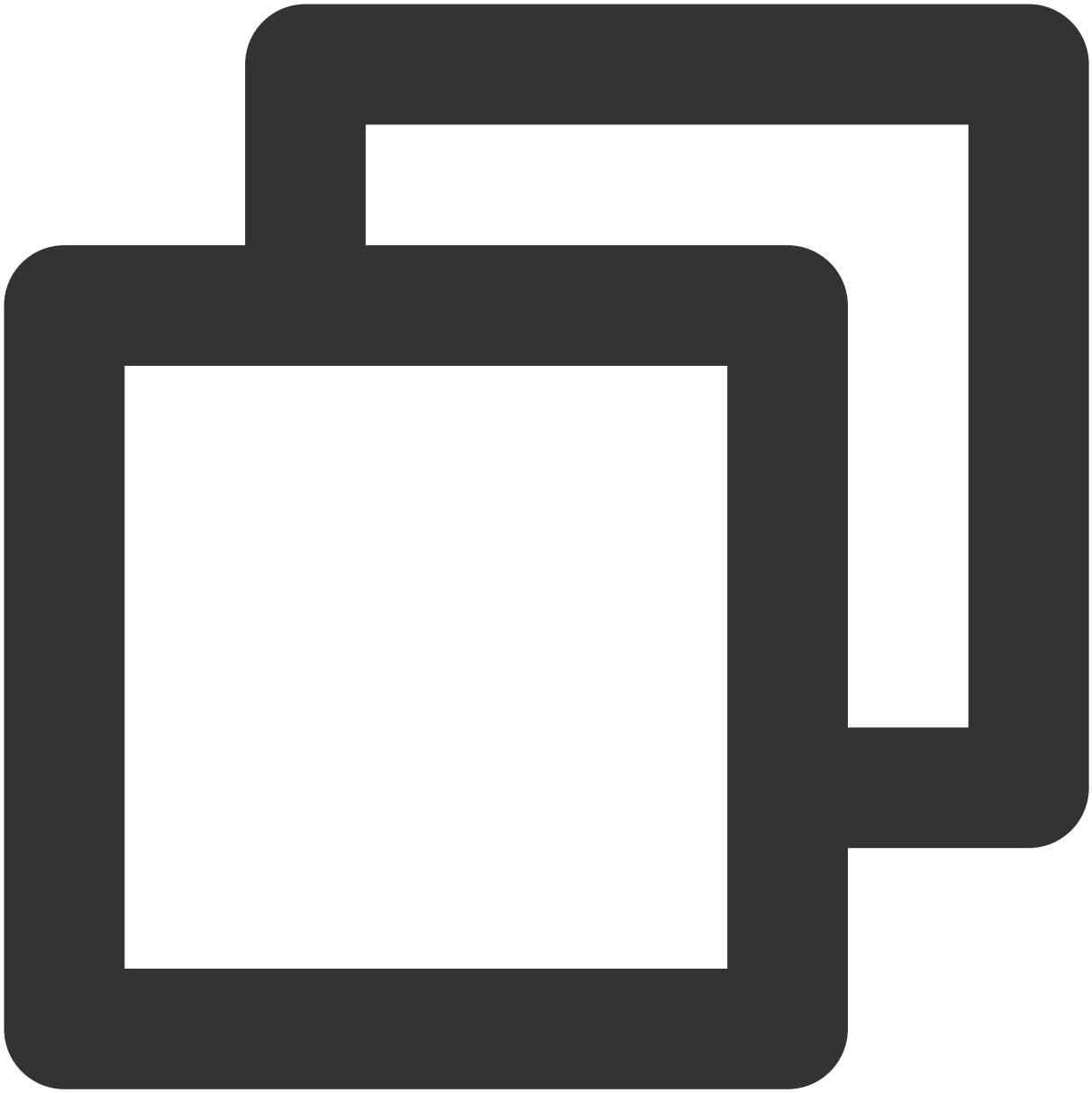
销毁 TUICallEngine 的单例。



```
- (void)destroyInstance;
```

init

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。



```
- (void)init:(NSString *)sdkAppID userId:(NSString *)userId userSig:(NSString *)use
```

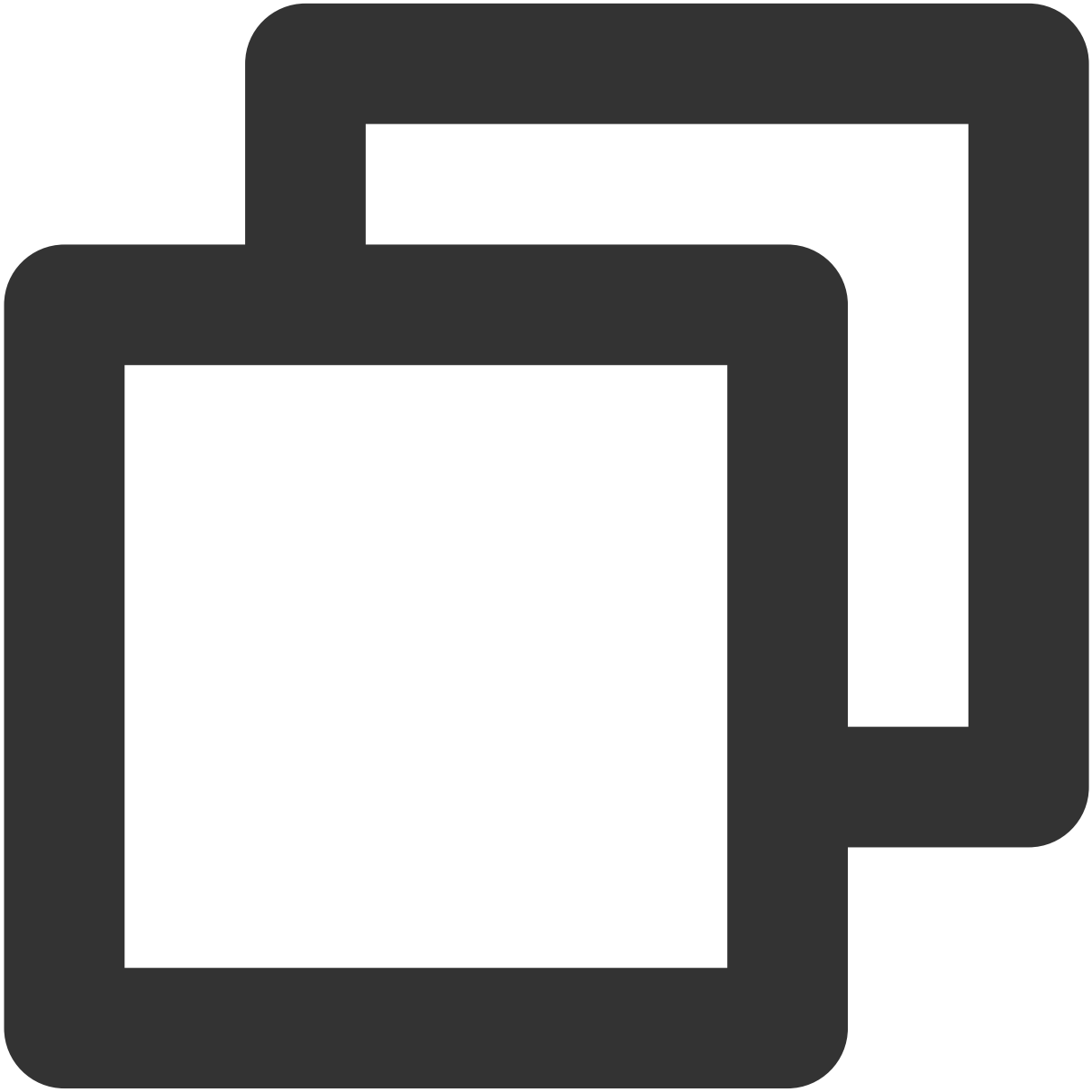
参数如下表所示：

参数	类型	含义
sdkAppID	int	您可以在**实时音视频控制台** >**应用管理** > **应用信息**中查看 SDKAppID
userId	NSString	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）

userSig	NSString	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig

addObserver

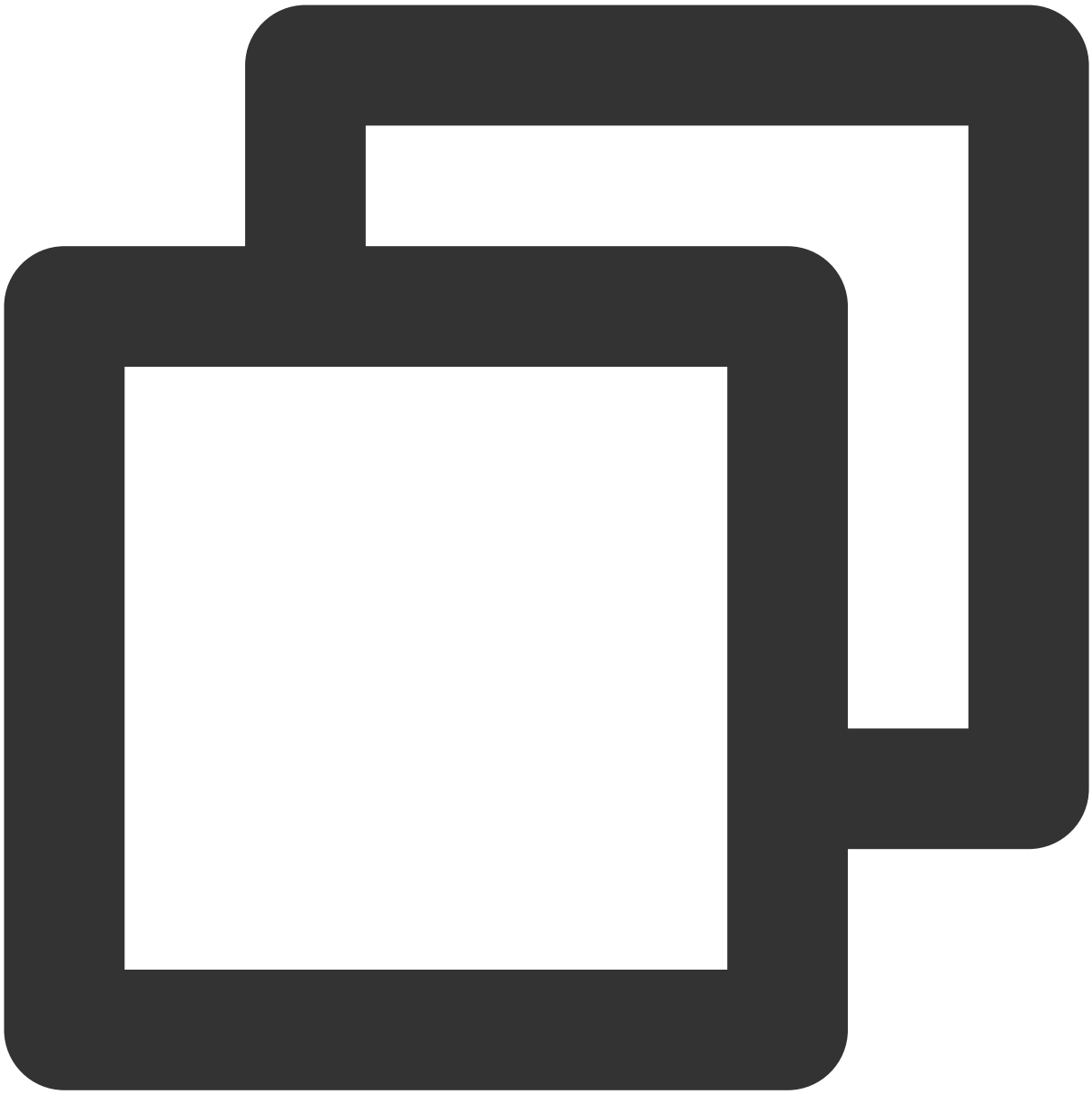
添加回调接口，您可以通过这个接听，监听 TUICallObserver 相关的事件回调。



```
- (void)addObserver:(id<TUICallObserver>)observer;
```

removeObserver

移除回调接口。



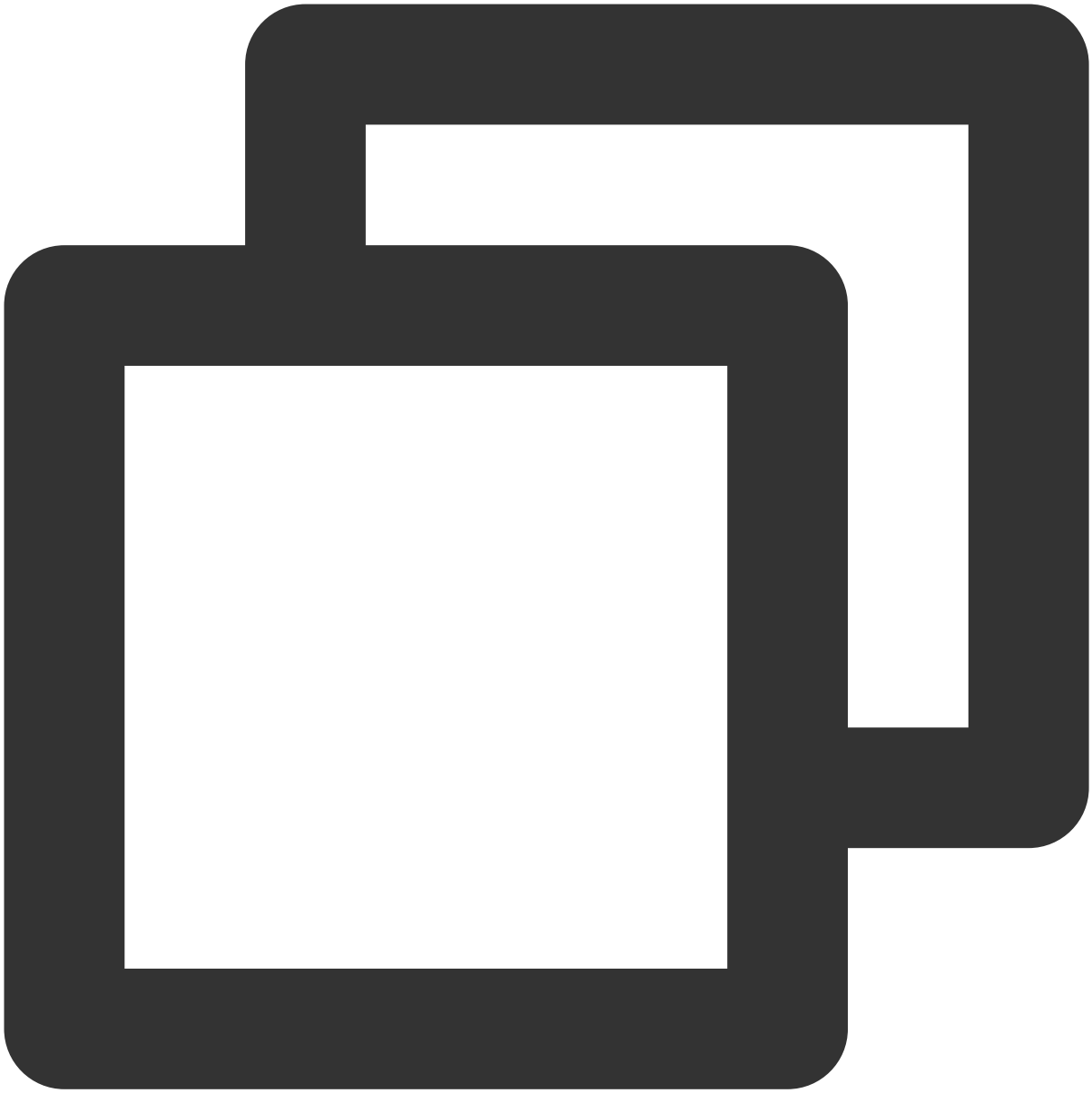
```
- (void)removeObserver:(id<TUICallObserver>)observer;
```

call

拨打电话（1v1通话）。

注意：

字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！



```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType param
```

参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线

		推送自定义内容等

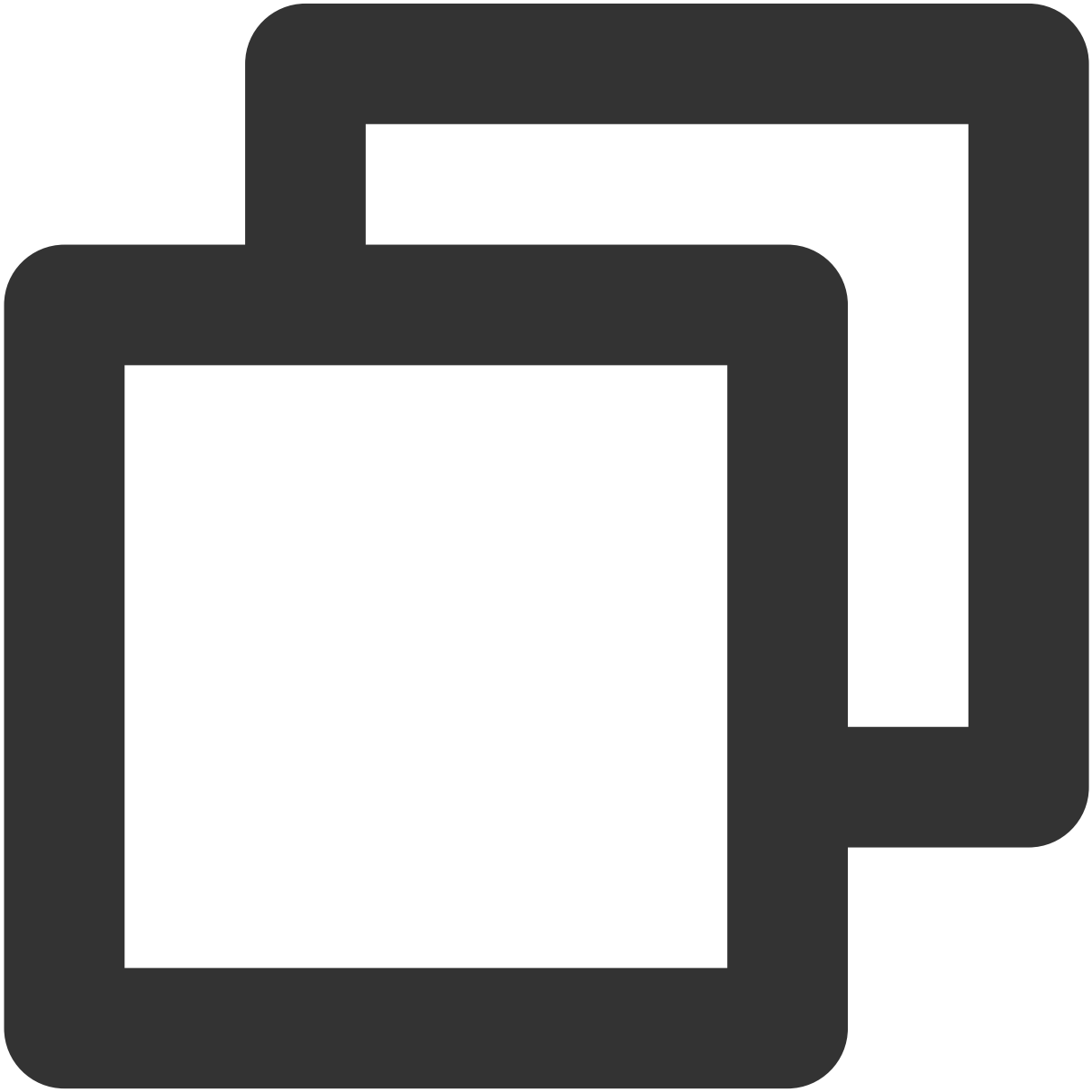
groupCall

发起群组通话。

注意：

使用群组通话前需要创建 IM 群组，如果已经创建，请忽略；

字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！

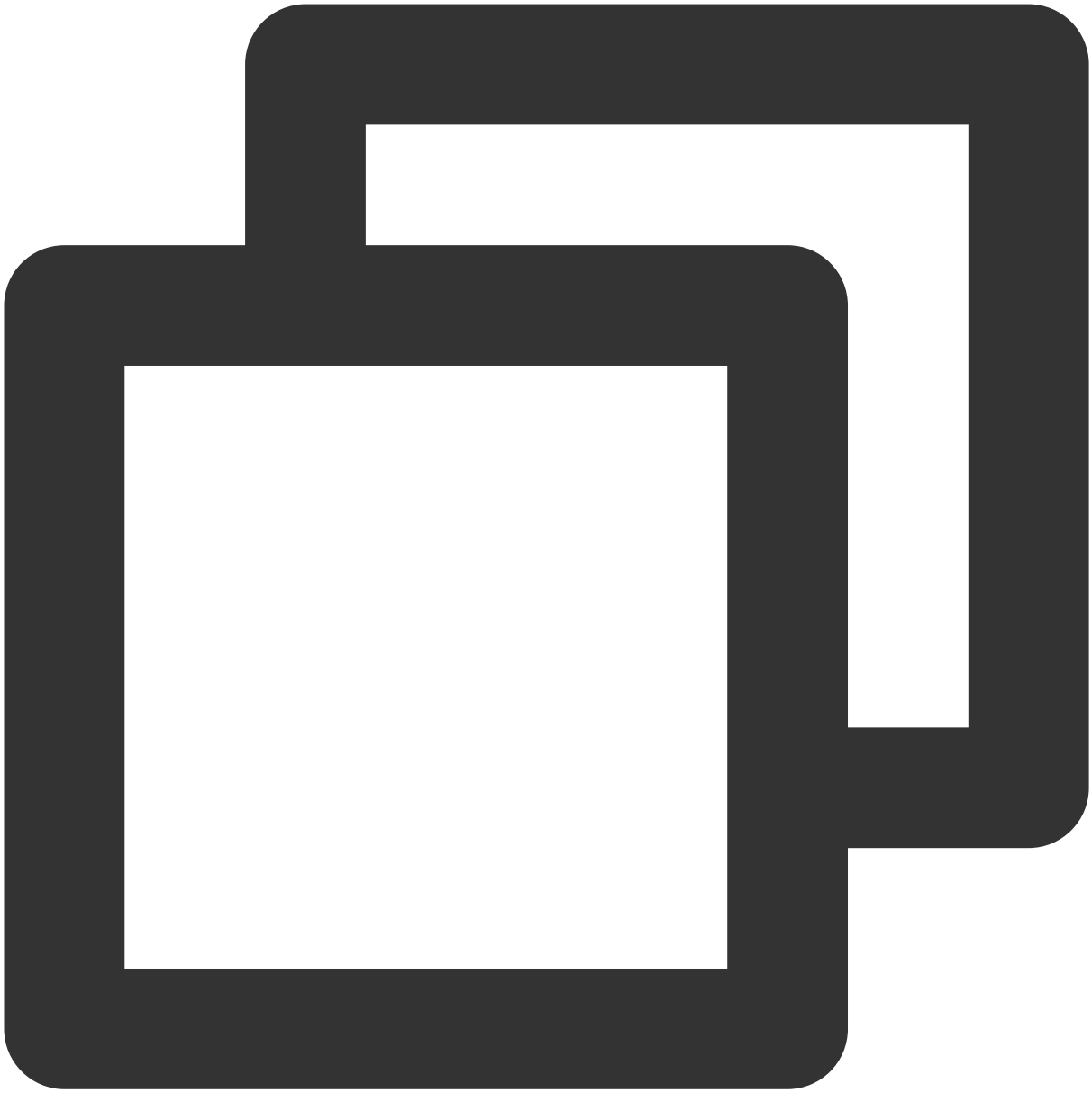


```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray <NSString *> *)userIdList
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等

accept

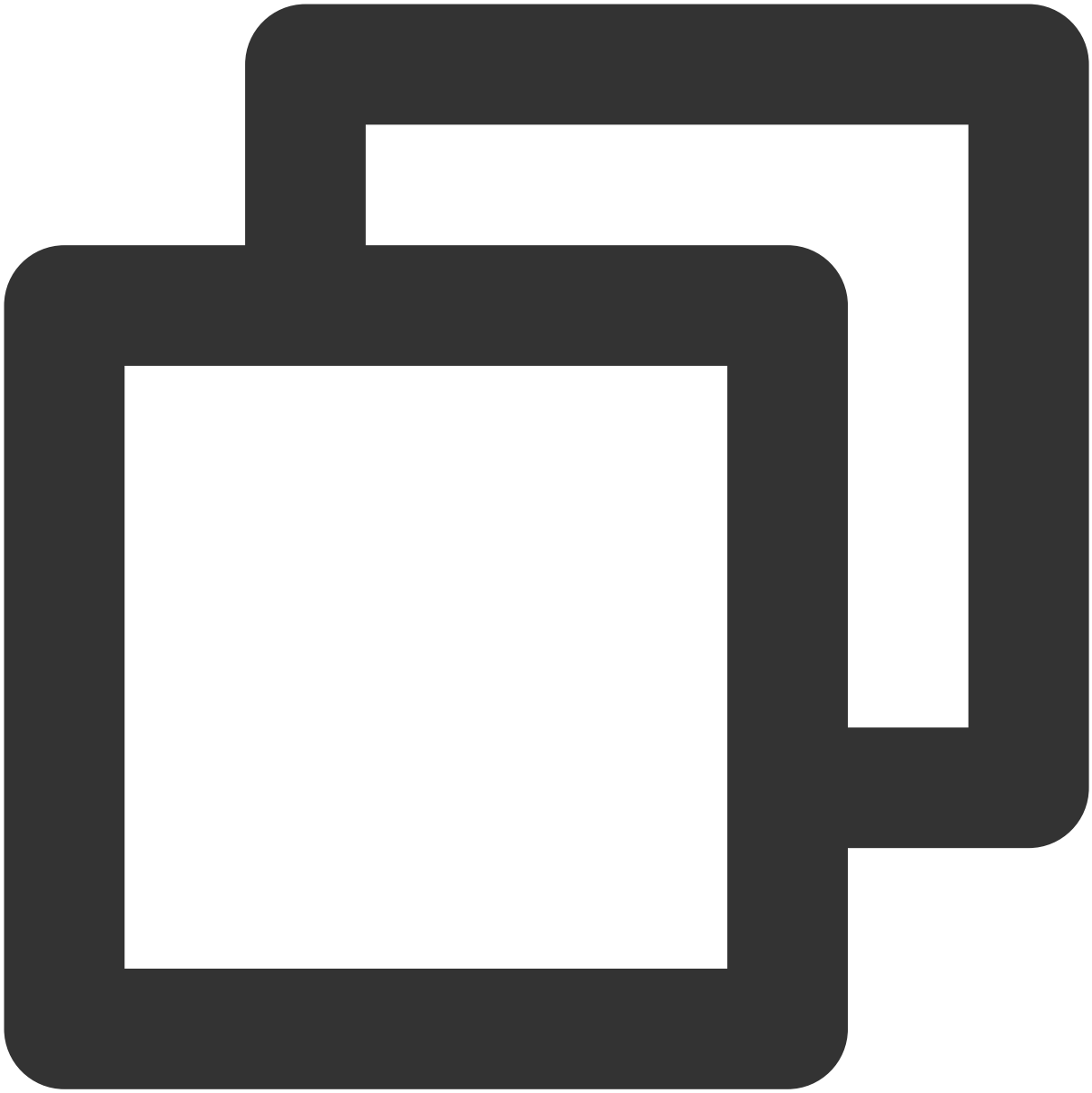
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
- (void)accept:(TUICallSucc)succ fail:(TUICallFail)fail;
```

reject

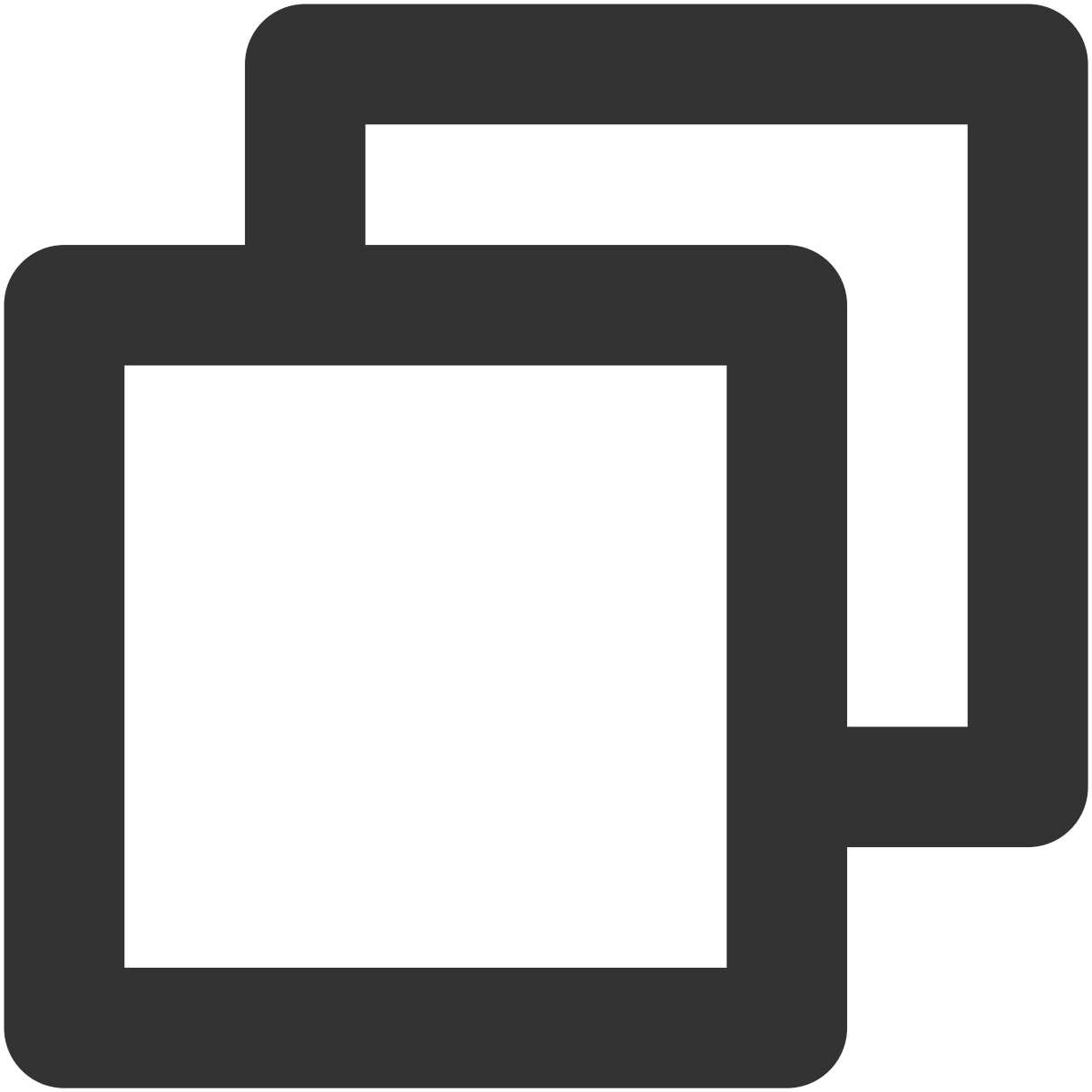
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



```
- (void)reject:(TUICallSucc)succ fail:(TUICallFail)fail;
```

ignore

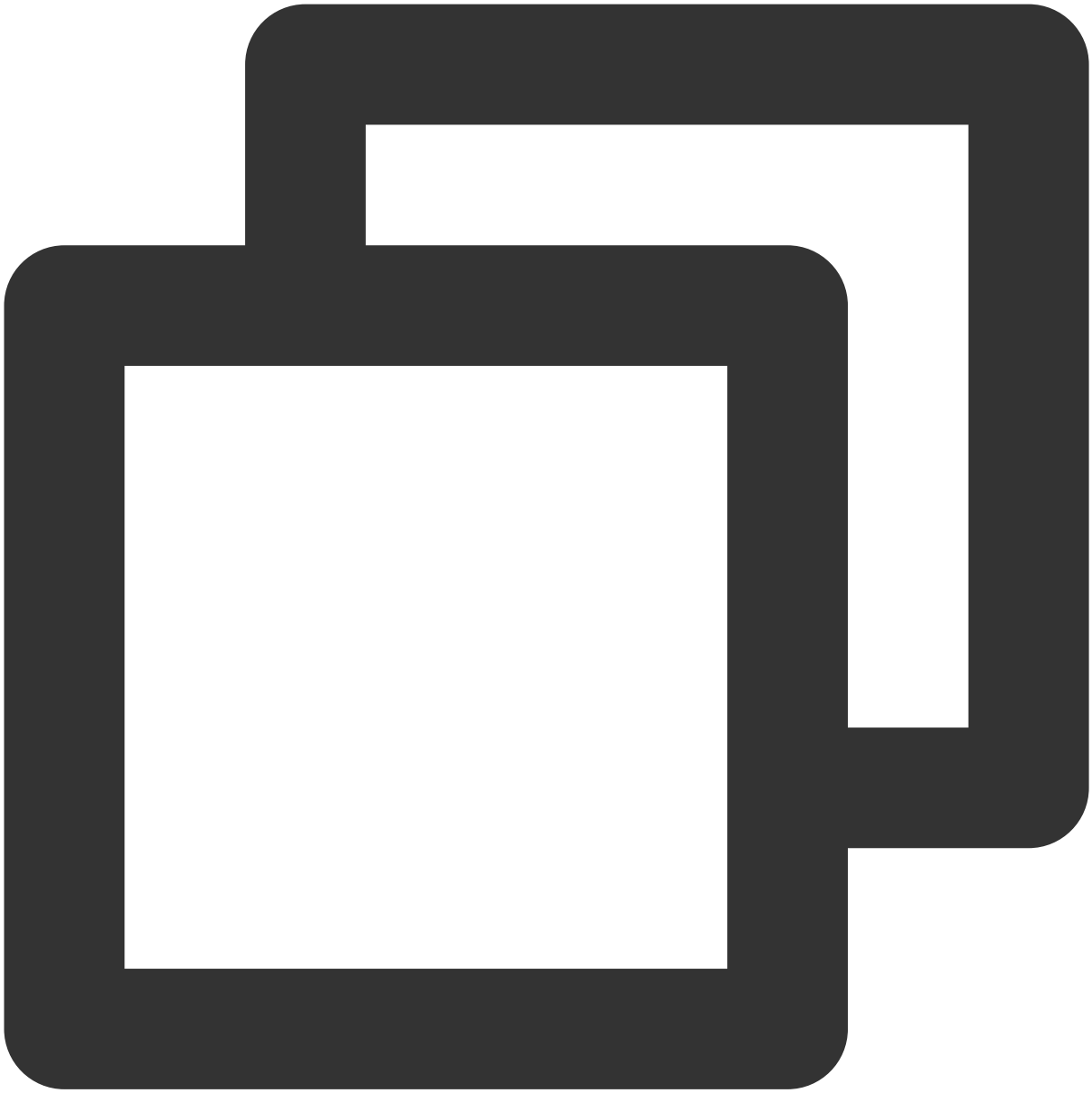
忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
- (void)ignore:(TUICallSucc)succ fail:(TUICallFail)fail;
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

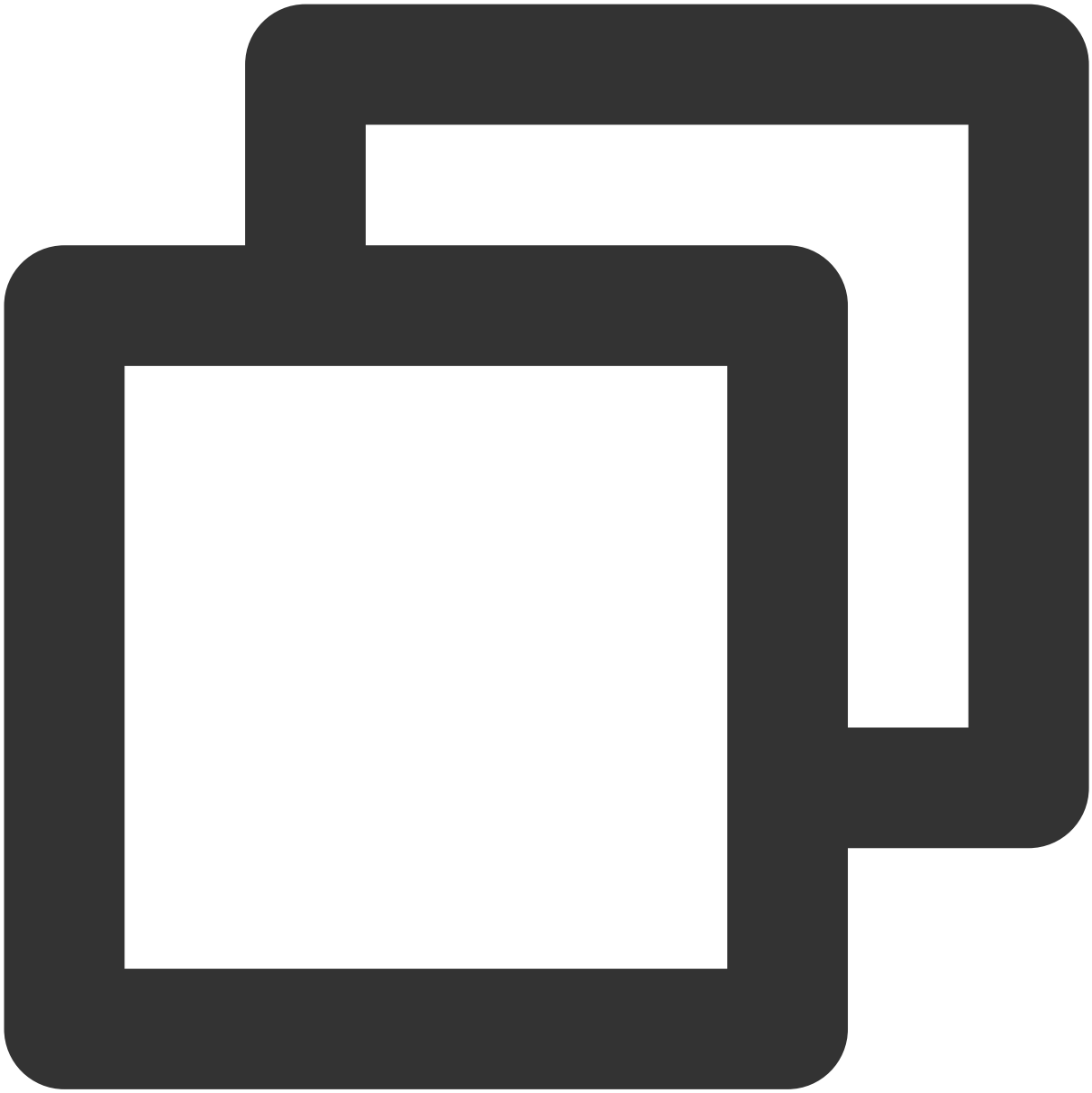


```
- (void)hangup:(TUICallSucc)succ fail:(TUICallFail)fail;
```

inviteUser

邀请用户加入此次群组通话。

使用场景：一个群组通话中的用户主动邀请其他人时使用。



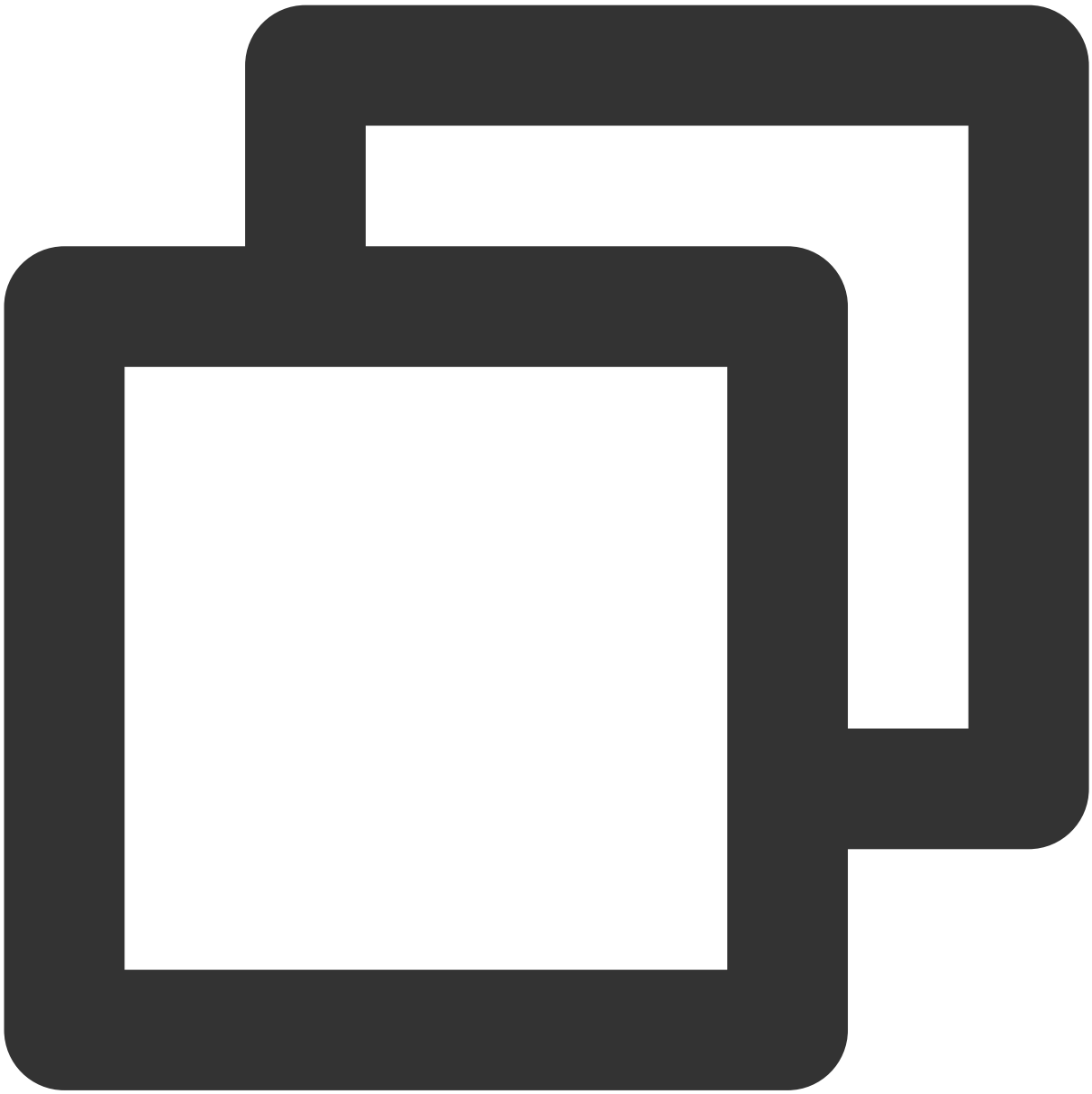
```
- (void)inviteUser:(NSArray<NSString *> *)userIdList params:(TUICallParams *)params
```

参数	类型	含义
userIdList	NSArray	目标用户的 userId 列表
params	TUICallParams	通话参数扩展字段，例如：离线推送自定义内容

joinInGroupCall

主动加入此次群组通话。

使用场景：群组内用户主动加入此次群组通话使用。



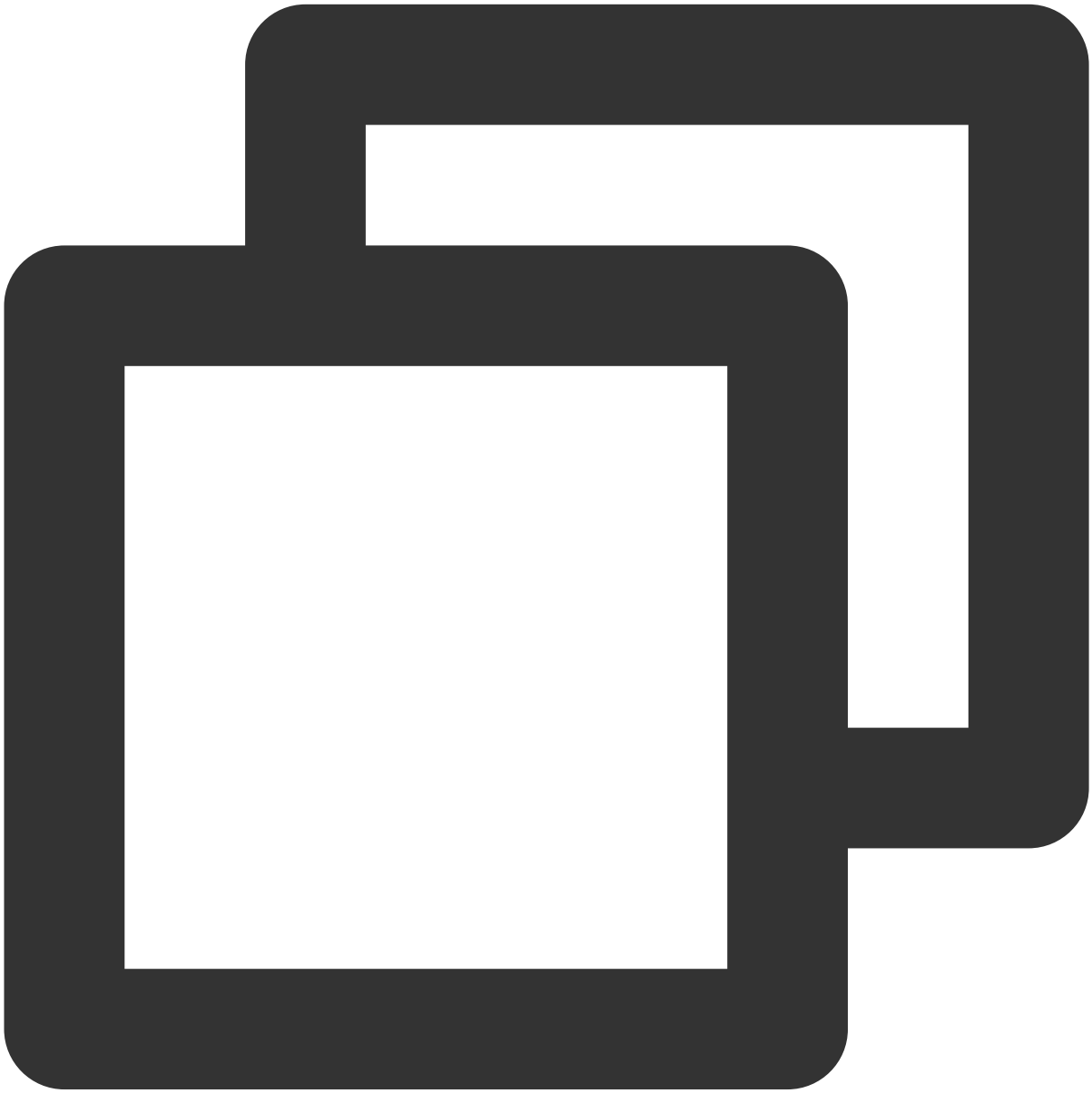
```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID
groupId	NSString	此次群组通话的群 ID

callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

switchCallMediaType

切换视频通话到语音通话。

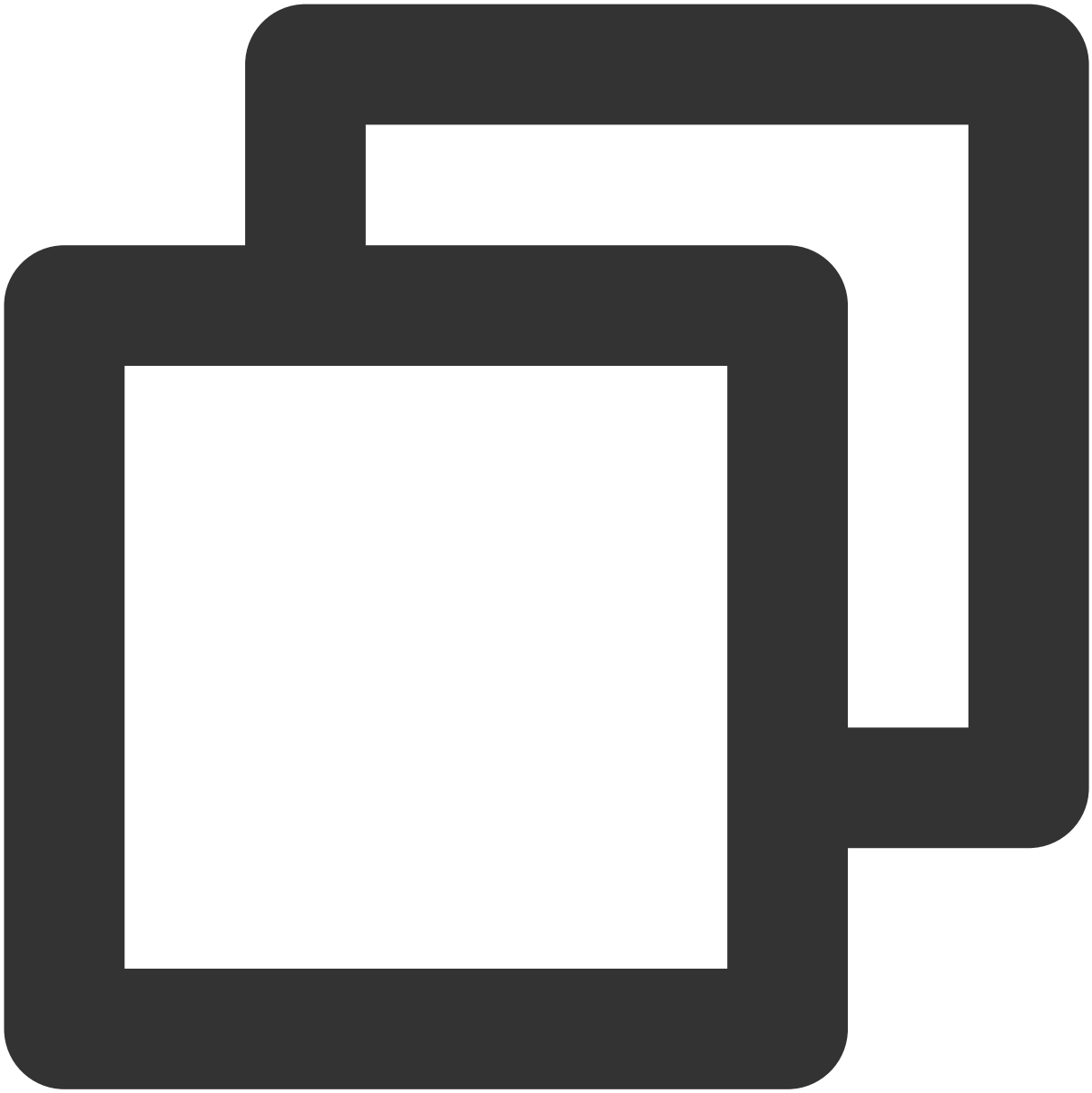


```
- (void)switchCallMediaType:(TUICallMediaType)newType;
```

参数	类型	含义
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

startRemoteView

设置显示视频画面的 View 对象。

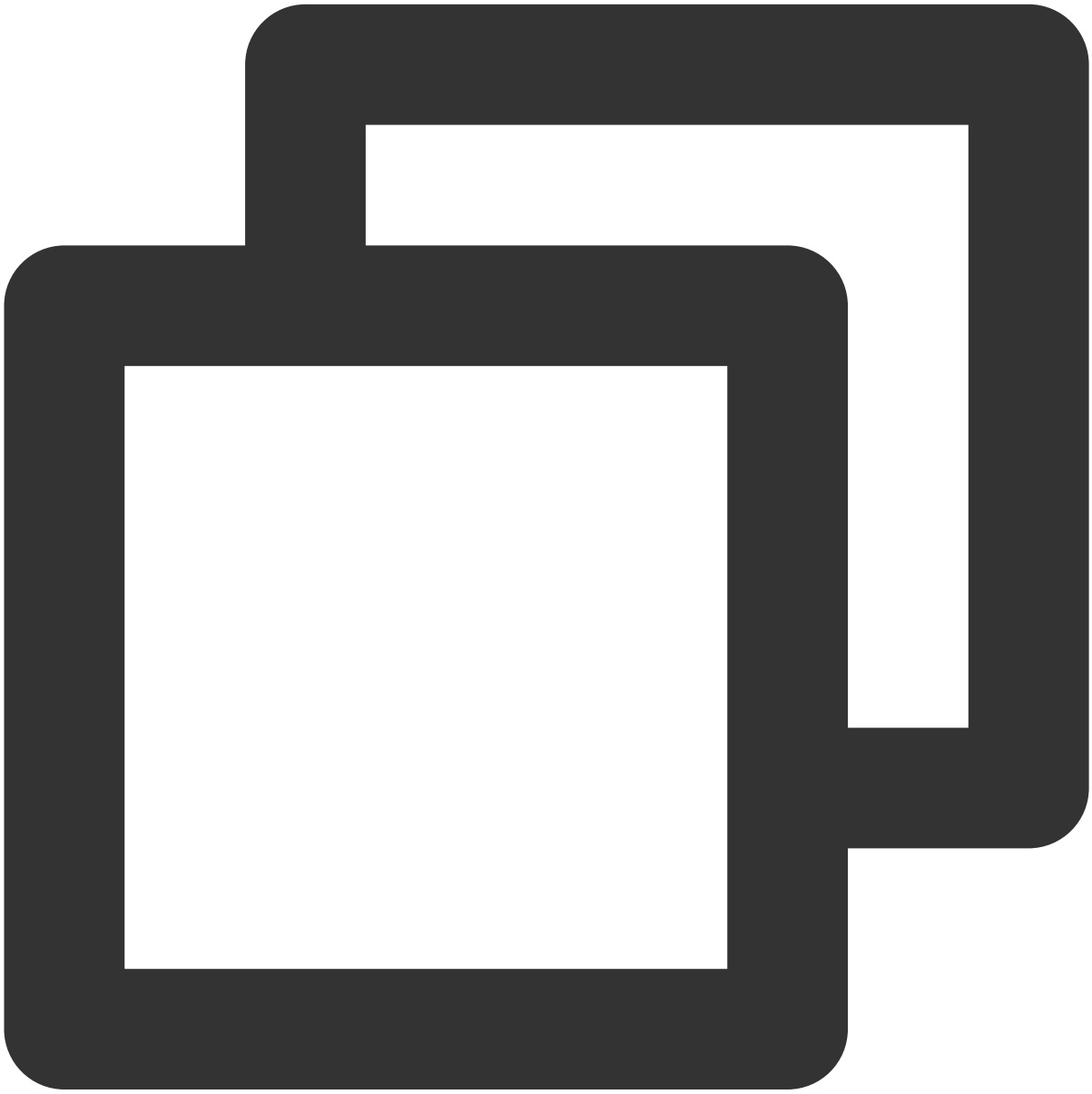


```
- (void)startRemoteView:(NSString *)userId videoView:(TUIVideoView *)videoView onPl
```

参数	类型	含义
userId	NSString	目标用户的 userId
videoView	TUIVideoView	待渲染的视图

stopRemoteview

停止订阅远端用户的视频数据。

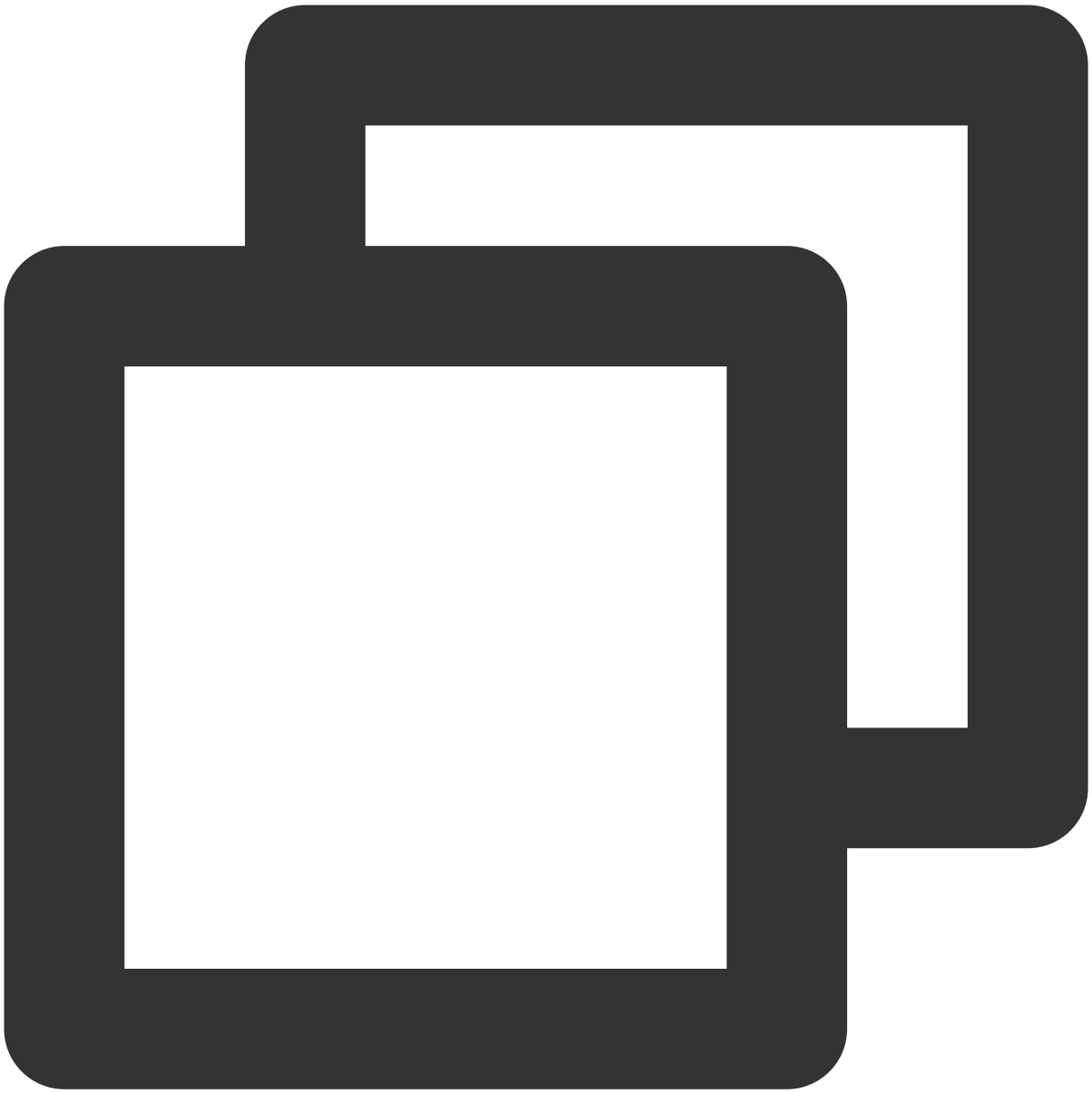


```
- (void)stopRemoteView:(NSString *)userId;
```

参数	类型	含义
userId	NSString	目标用户的 userId

openCamera

开启摄像头。

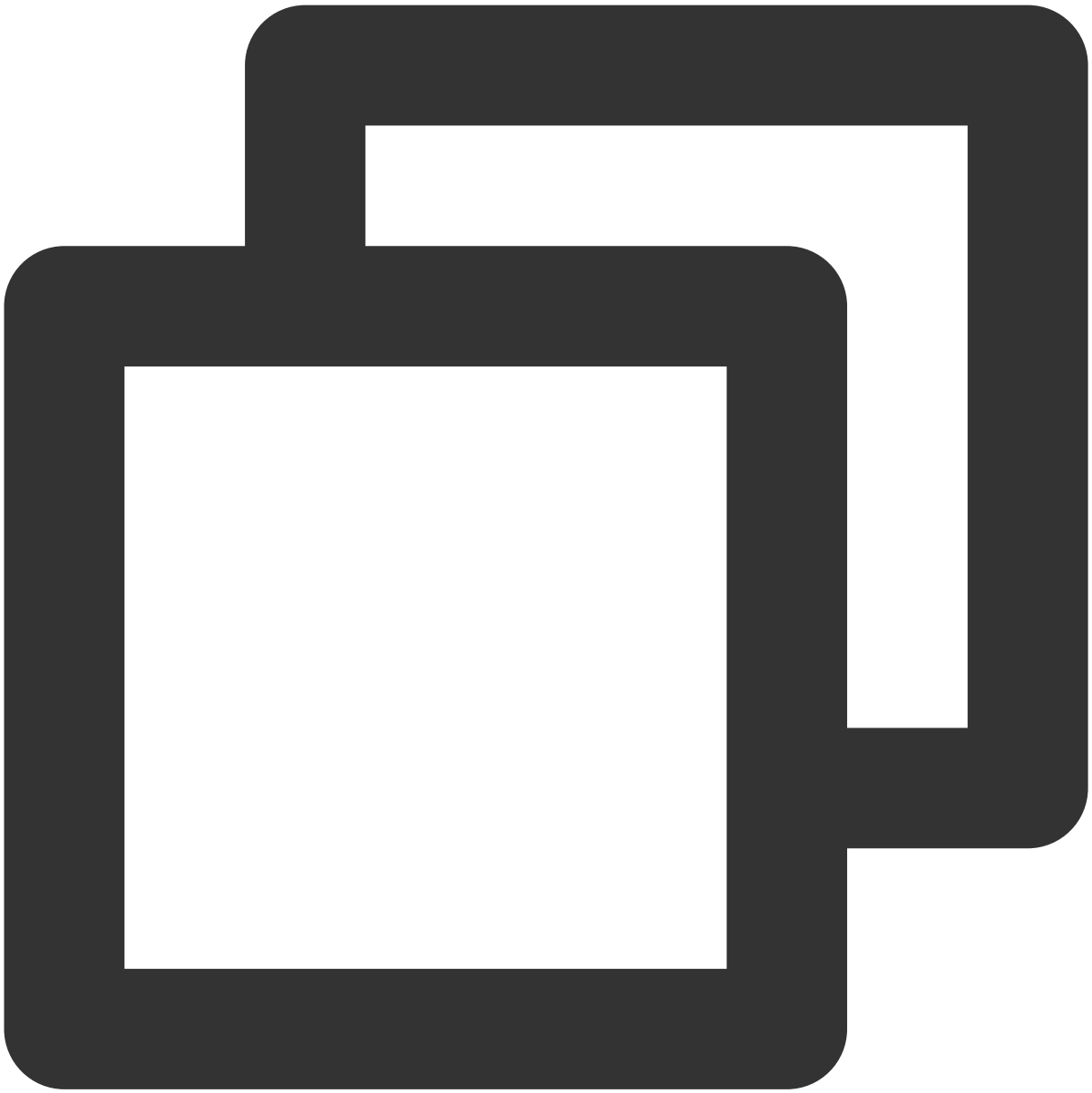


```
- (void)openCamera:(TUICamera)camera videoView:(TUIVideoView *)videoView succ:(TUIC
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头
videoView	TUIVideoView	待渲染的视图

closeCamera

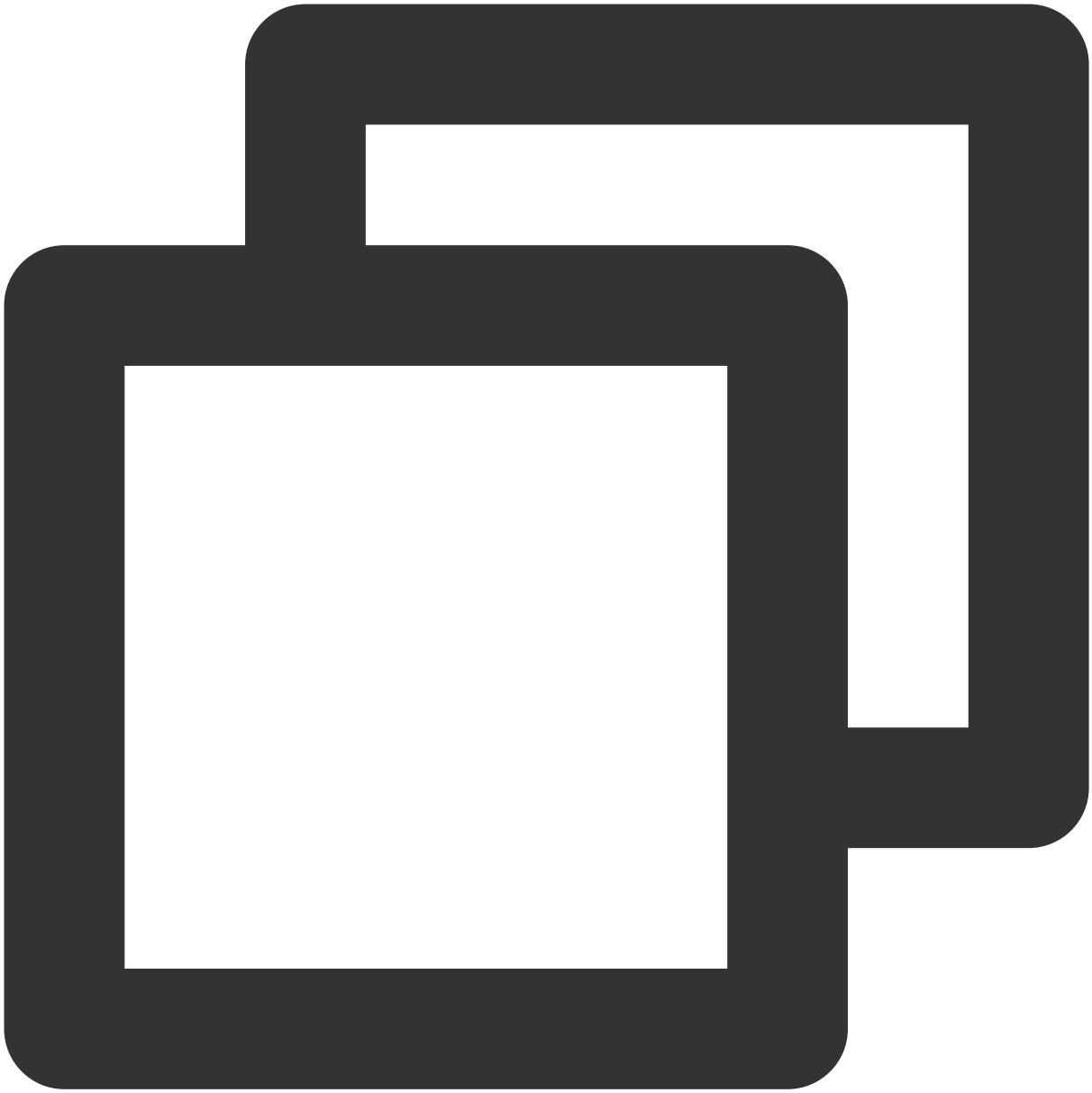
关闭摄像头。



```
- (void)closeCamera;
```

switchCamera

切换前后摄像头。

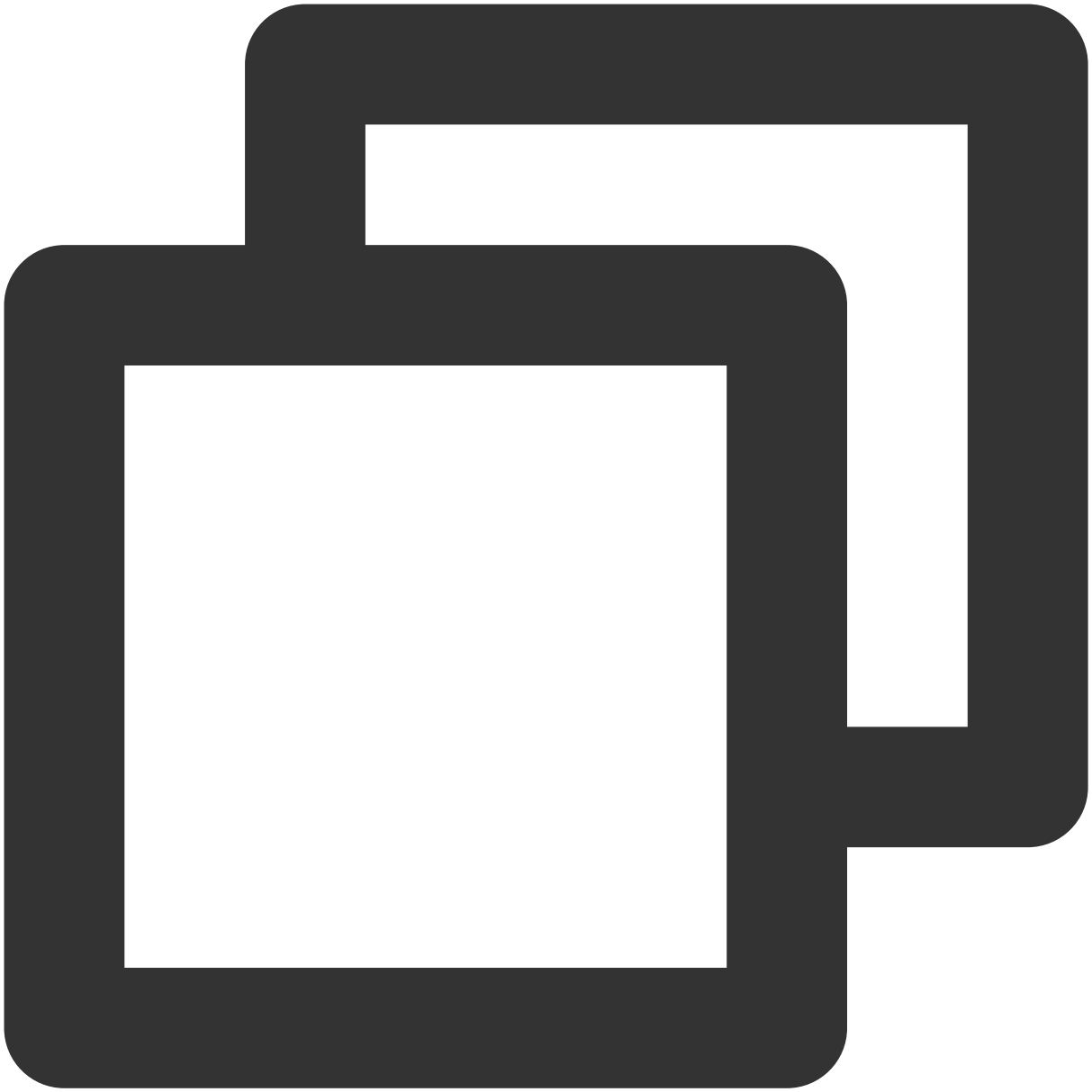


```
- (void)switchCamera:(TUICamera)camera;
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头

openMicrophone

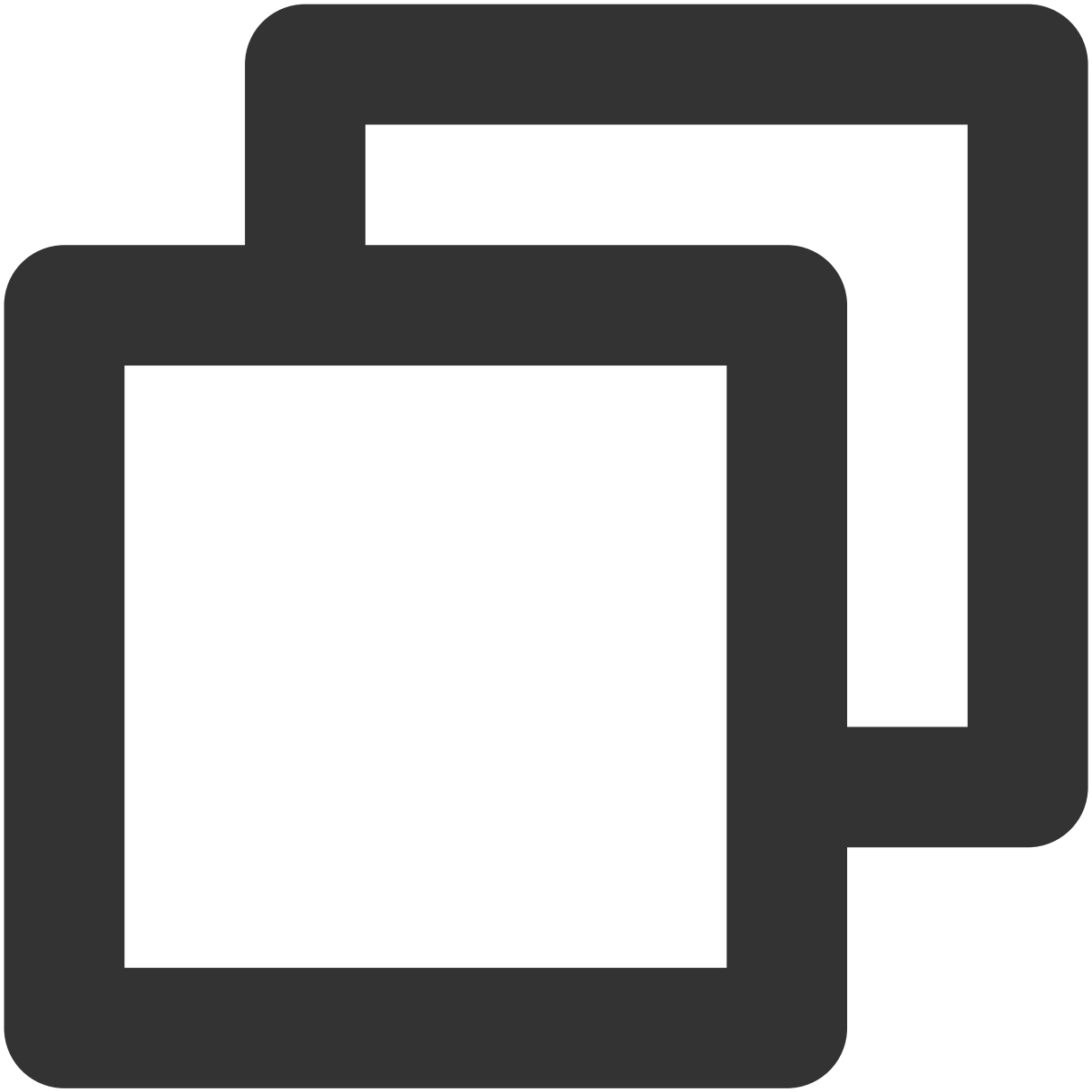
打开麦克风。



```
- (void)openMicrophone:(TUICallSucc)succ fail:(TUICallFail)fail;
```

closeMicrophone

关闭麦克风。

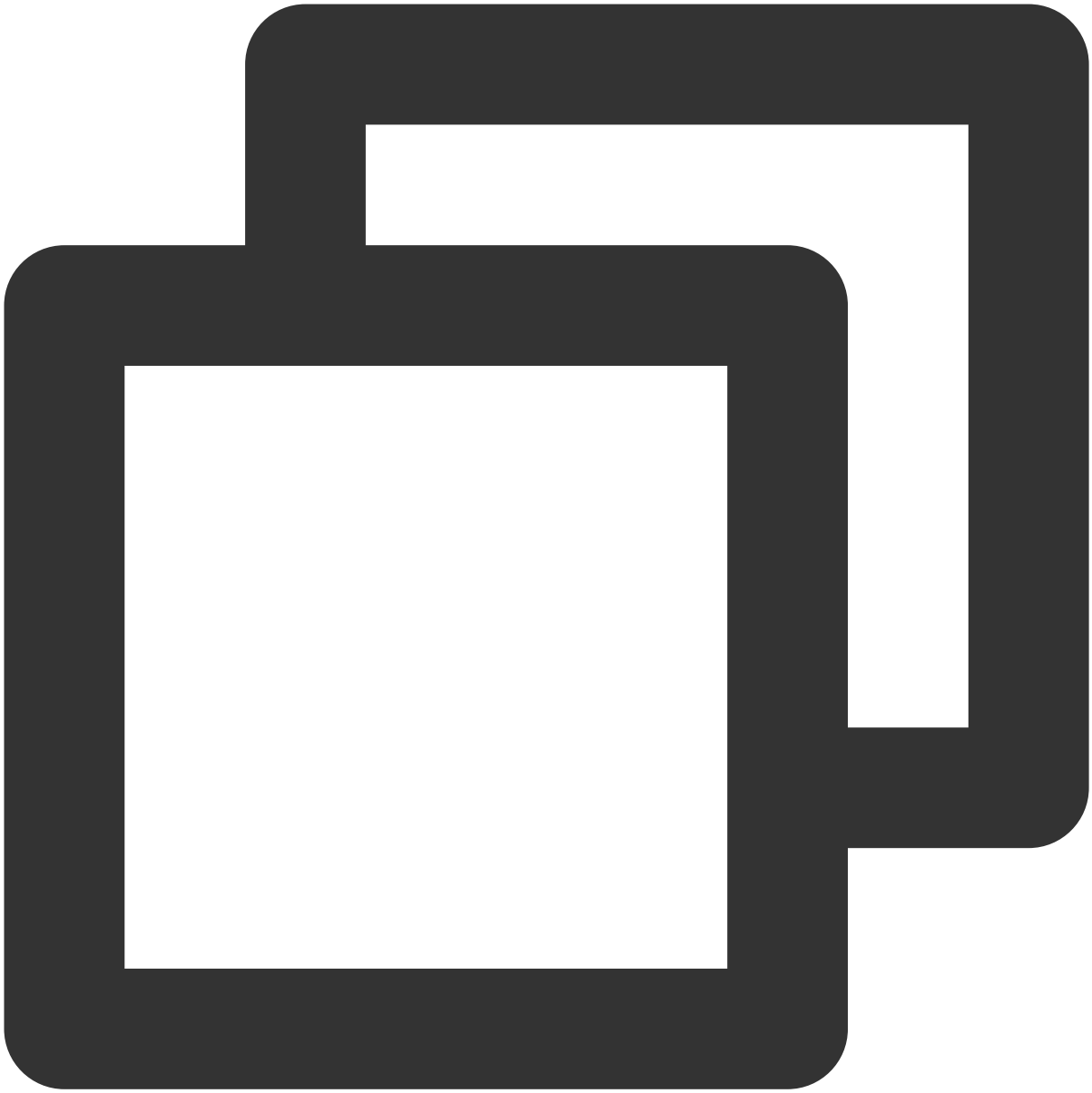


```
- (void)closeMicrophone;
```

selectAudioPlaybackDevice

选择音频播放设备。

目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。



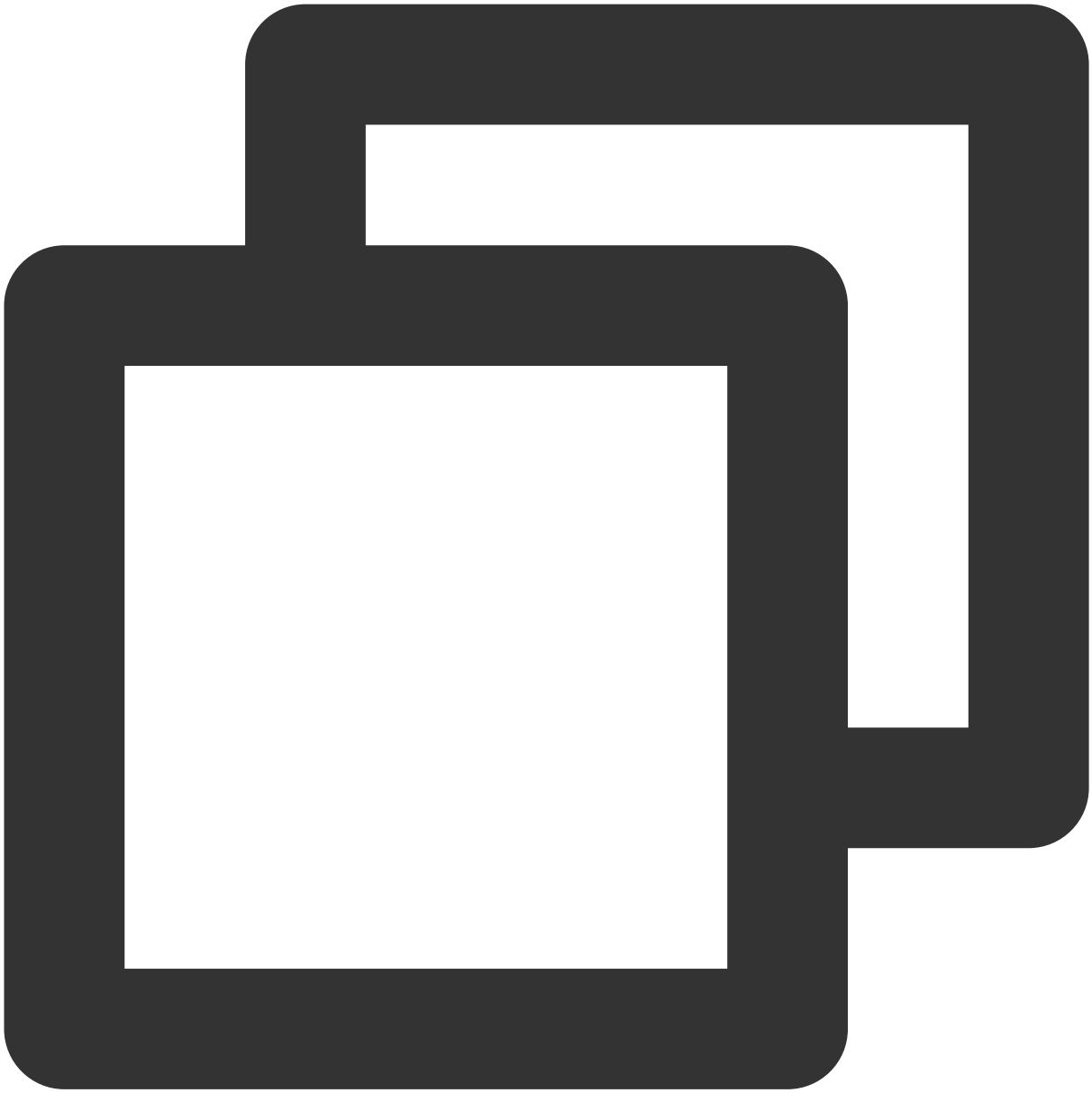
```
- (void)selectAudioPlaybackDevice:(TUIAudioPlaybackDevice)device;
```

参数	类型	含义
device	TUIAudioPlaybackDevice	听筒/扬声器

setSelfInfo

设置用户昵称、头像。

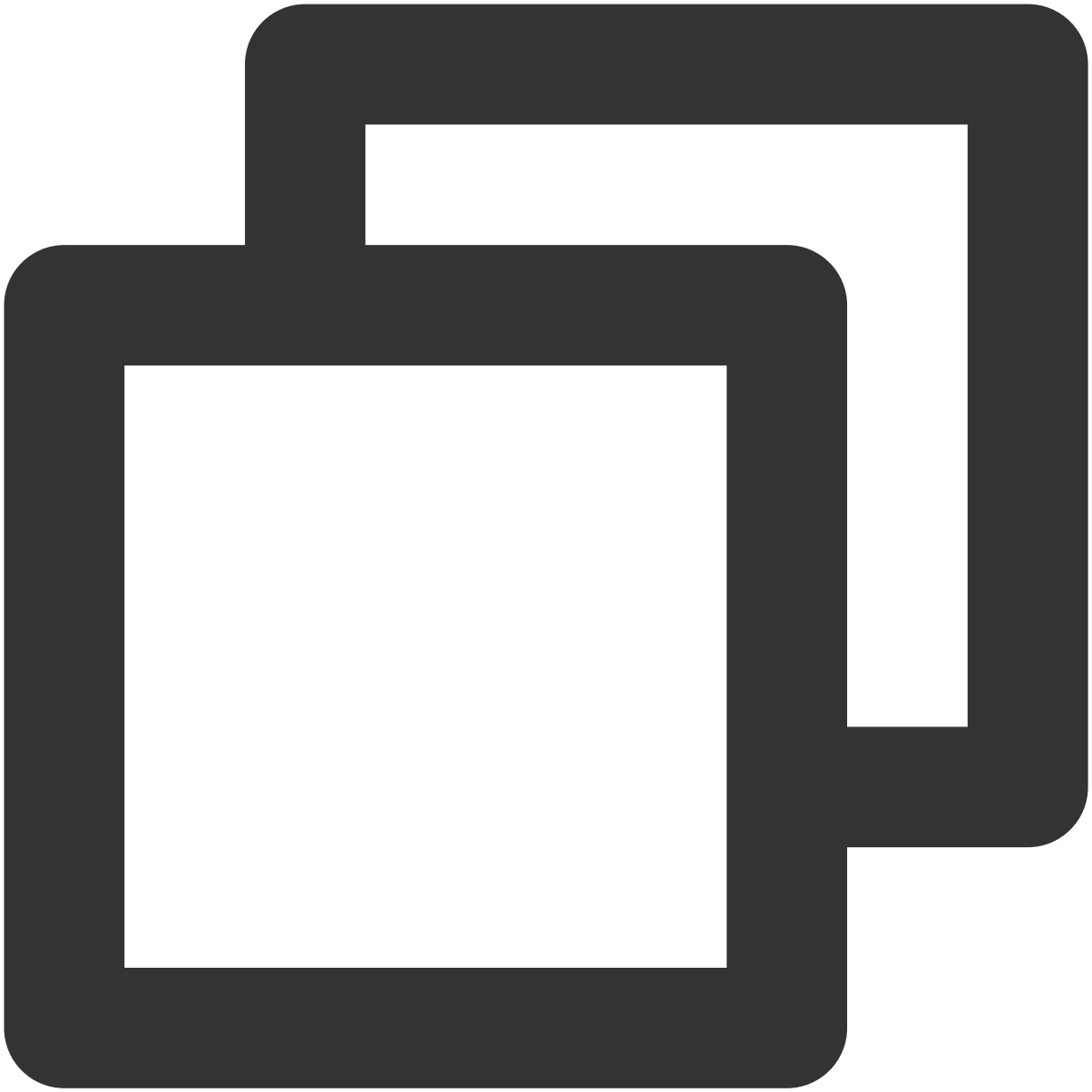
用户昵称不能超过500字节，用户头像必须是 URL 格式。



```
- (void)setSelfInfo:(NSString * _Nullable)nickName avatar:(NSString * _Nullable)ava
```

enableMultiDeviceAbility

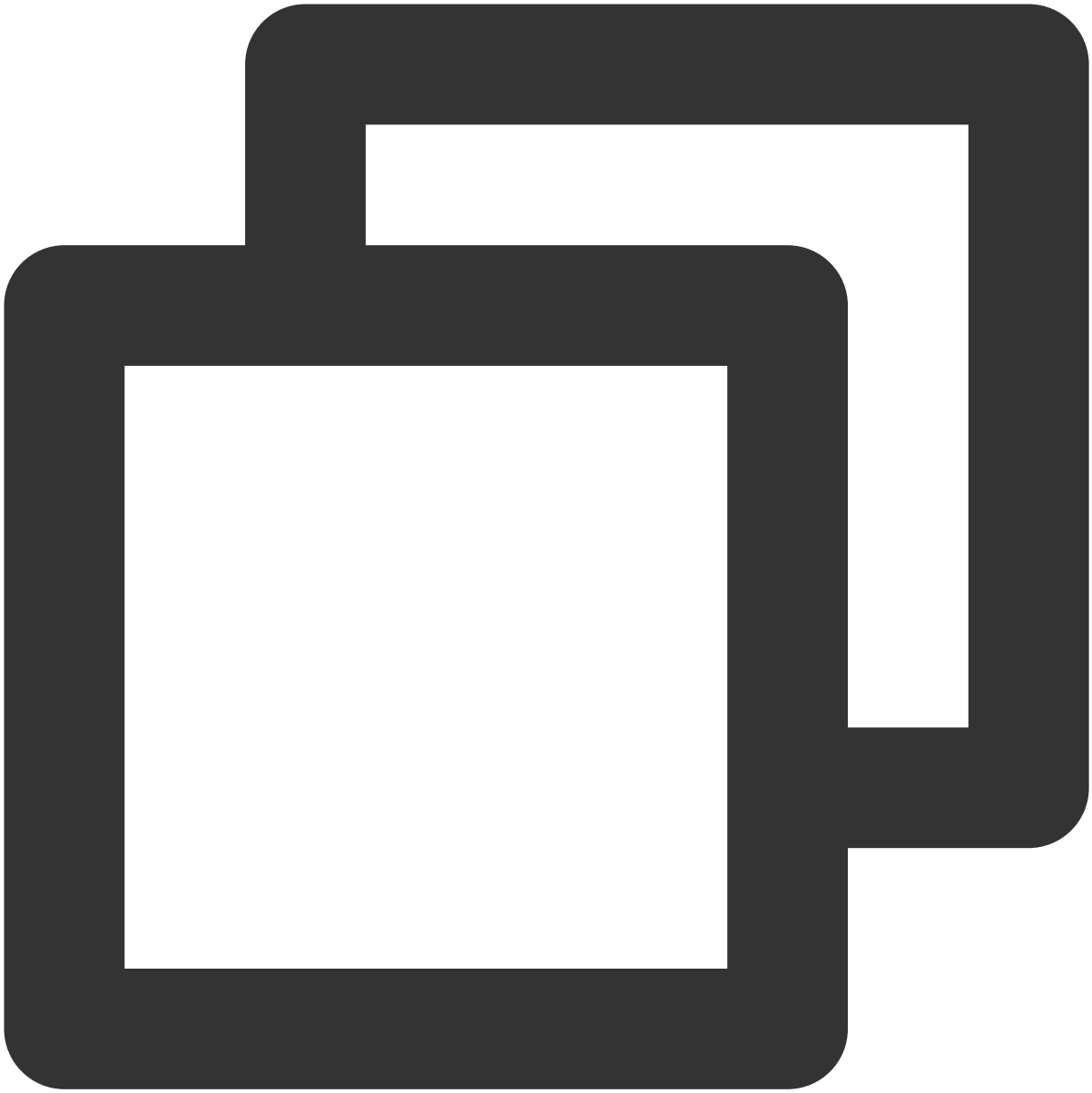
开启/关闭 TUICallEngine 的多设备登录模式（尊享版套餐支持）。



```
- (void)enableMultiDeviceAbility:(BOOL)enable succ:(TUICallSucc)succ fail:(TUICallF
```

setVideoRenderParams

设置用户视频画面的渲染模式。



```
- (void)setVideoRenderParams:(NSString *)userId params:(TUIVideoRenderParams *)para
```

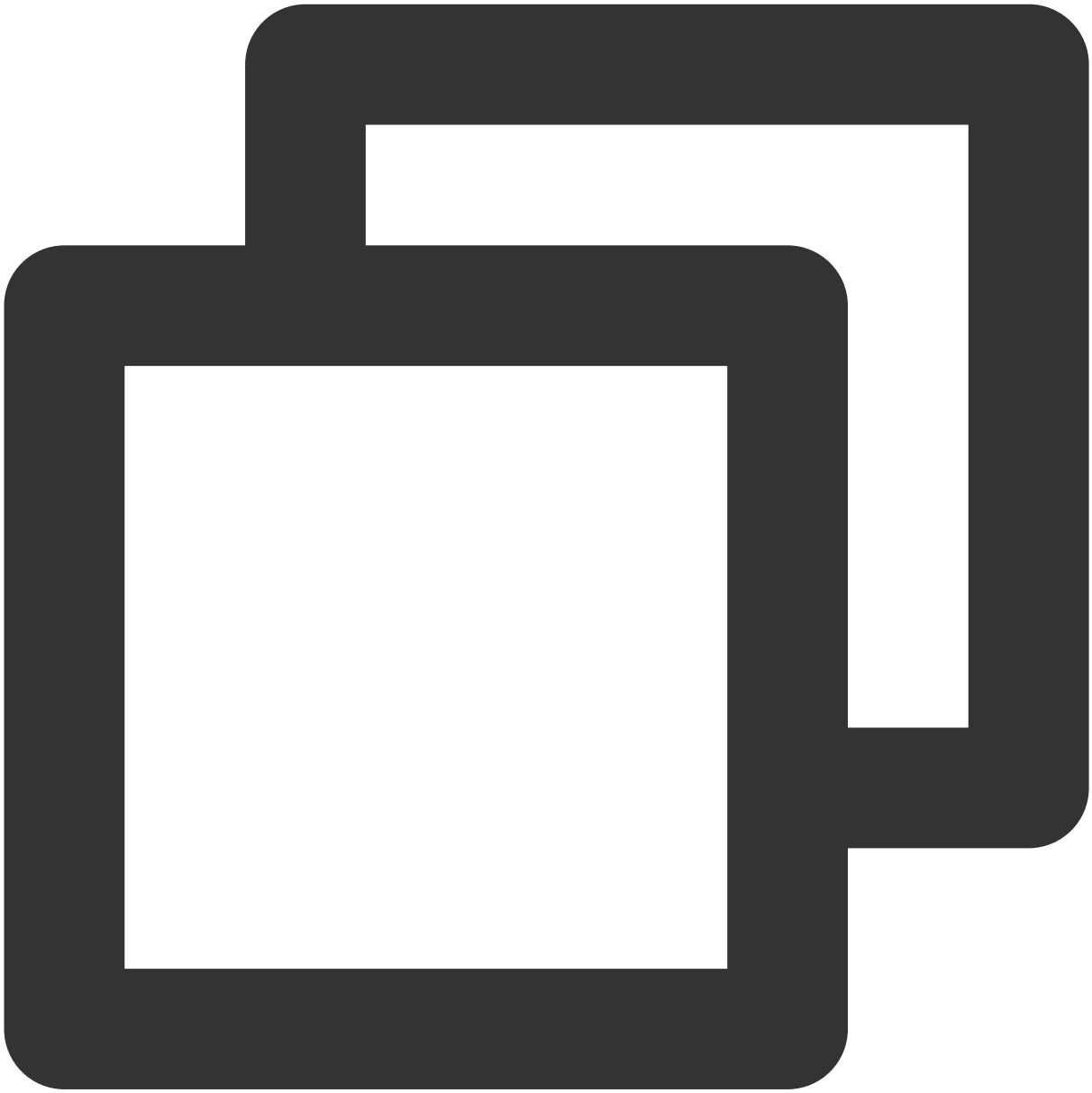
参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
params	TUIVideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式

setVideoEncoderParams

设置视频编码的编码参数。

该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。



```
- (void)setVideoEncoderParams:(TUIVideoEncoderParams *)params succ:(TUICallSucc)suc
```

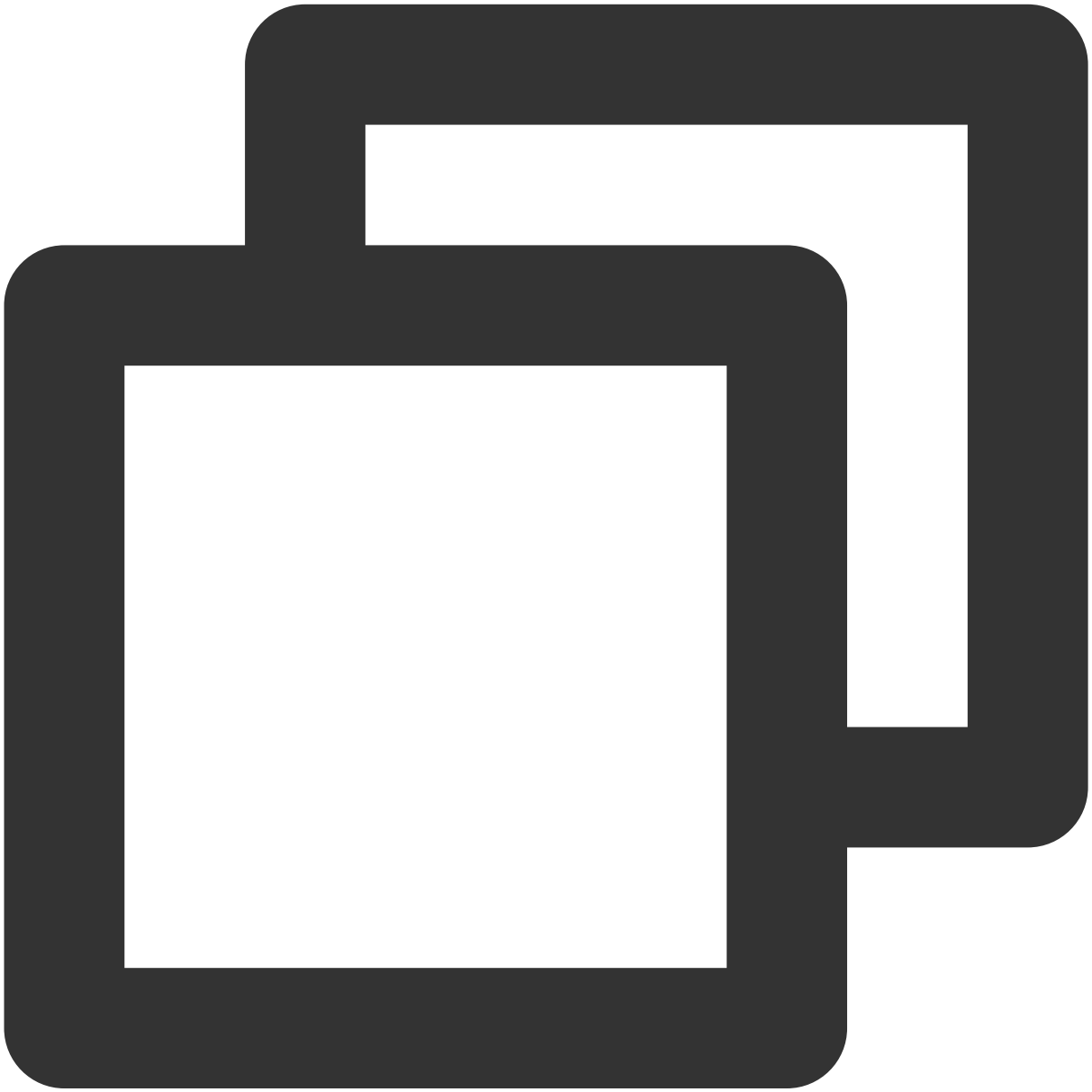
参数如下表所示：

参数	类型	含义
params	TUIVideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高比

getTRTCCloudInstance

高级接口。

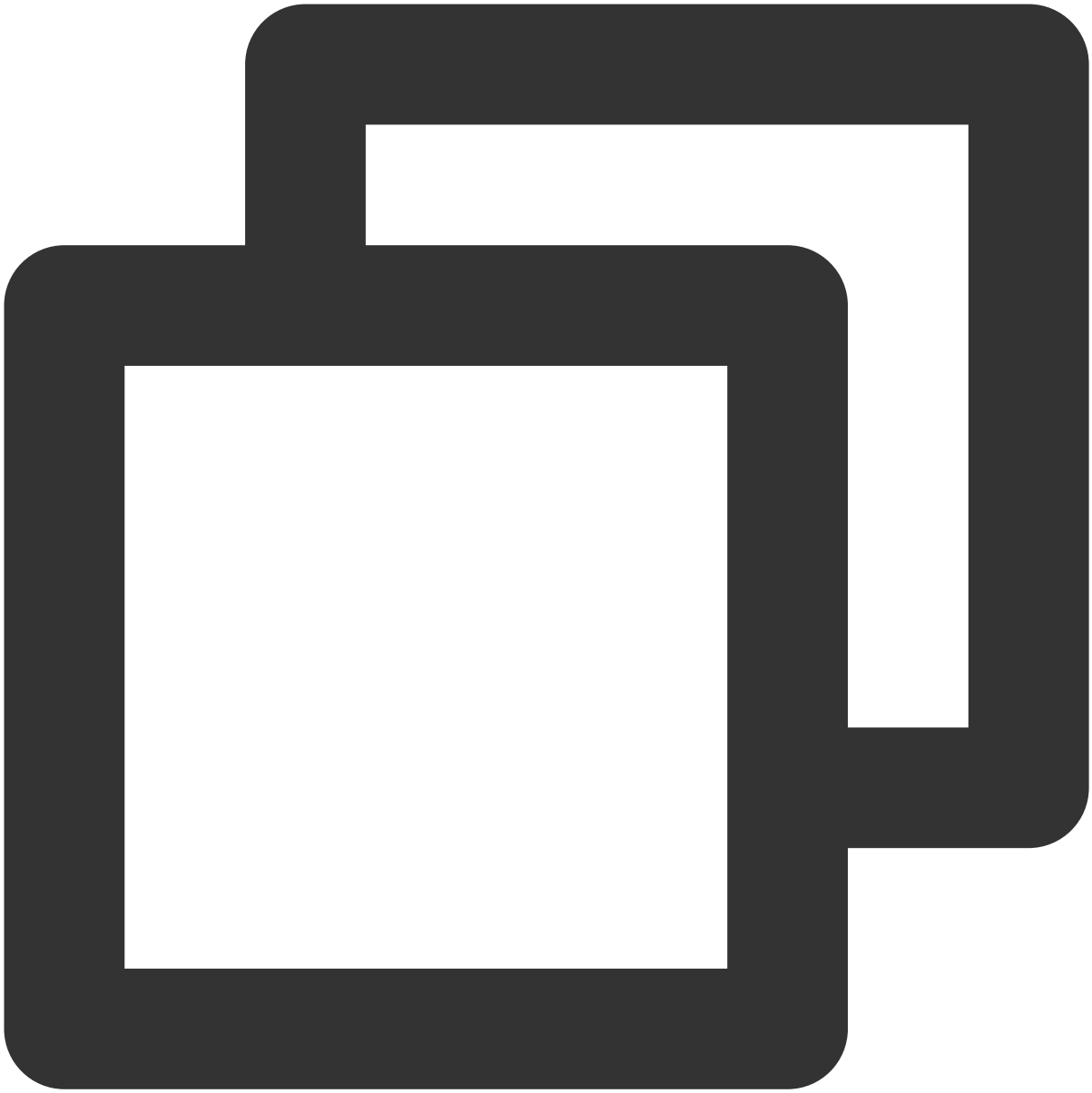
用于接入第三方美颜等。



```
- (TRTCCloud *)getTRTCCloudInstance;
```

setBeautyLevel

设置美颜级别，支持关闭默认美颜。



```
- (void)setBeautyLevel:(CGFloat)level succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

参数如下表所示：

参数	类型	含义
level	CGFloat	美颜级别，美颜级别，取值范围 0 - 9； 0 表示关闭，9 表示效果最明显

TUICallObserver

最近更新时间：2024-01-17 11:58:53

TUICallObserver API 简介

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线

版权所有：腾讯云计算（北京）有限责任公司

onUserSigExpired	在线时票据过期

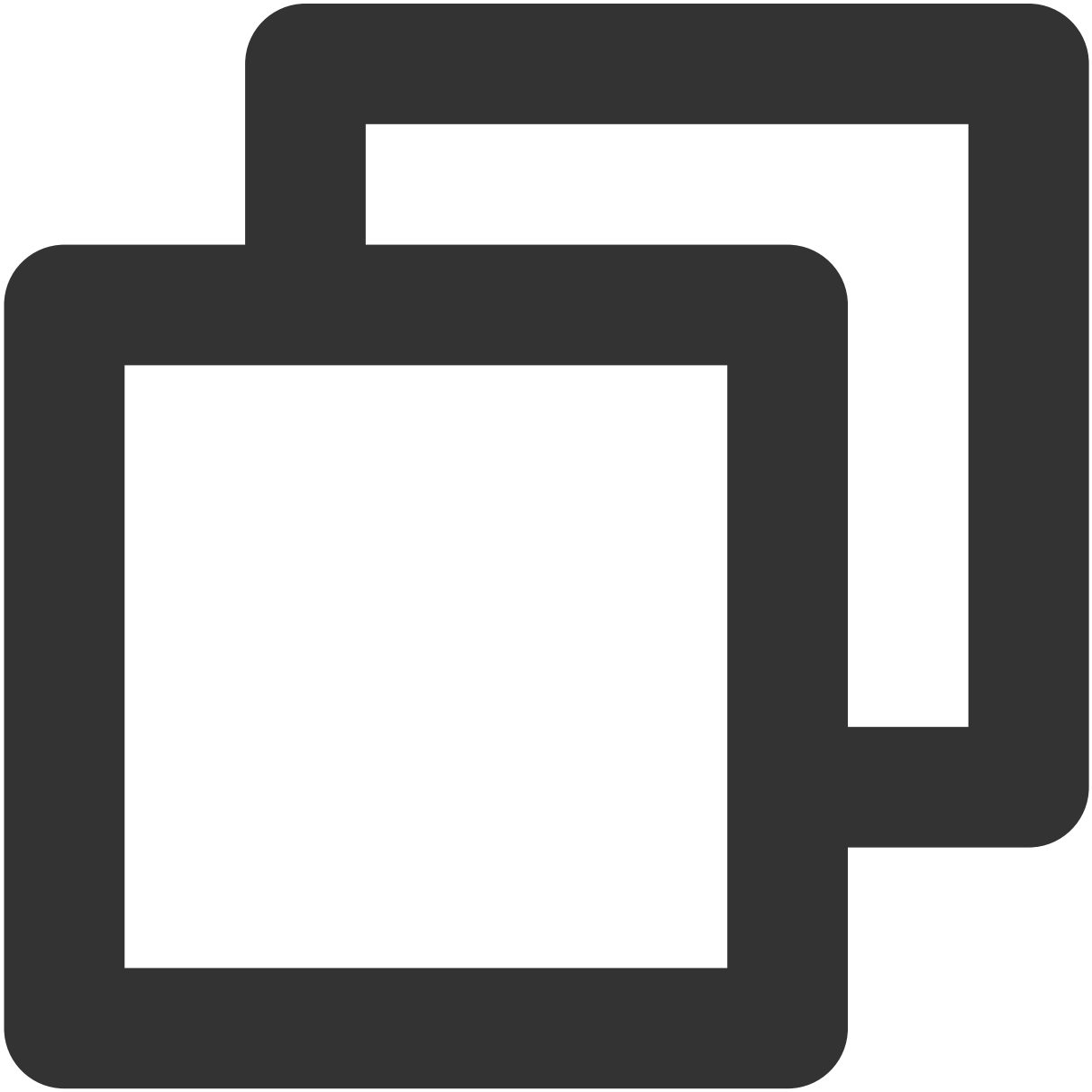
回调事件详情

onError

错误回调。

说明

SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。



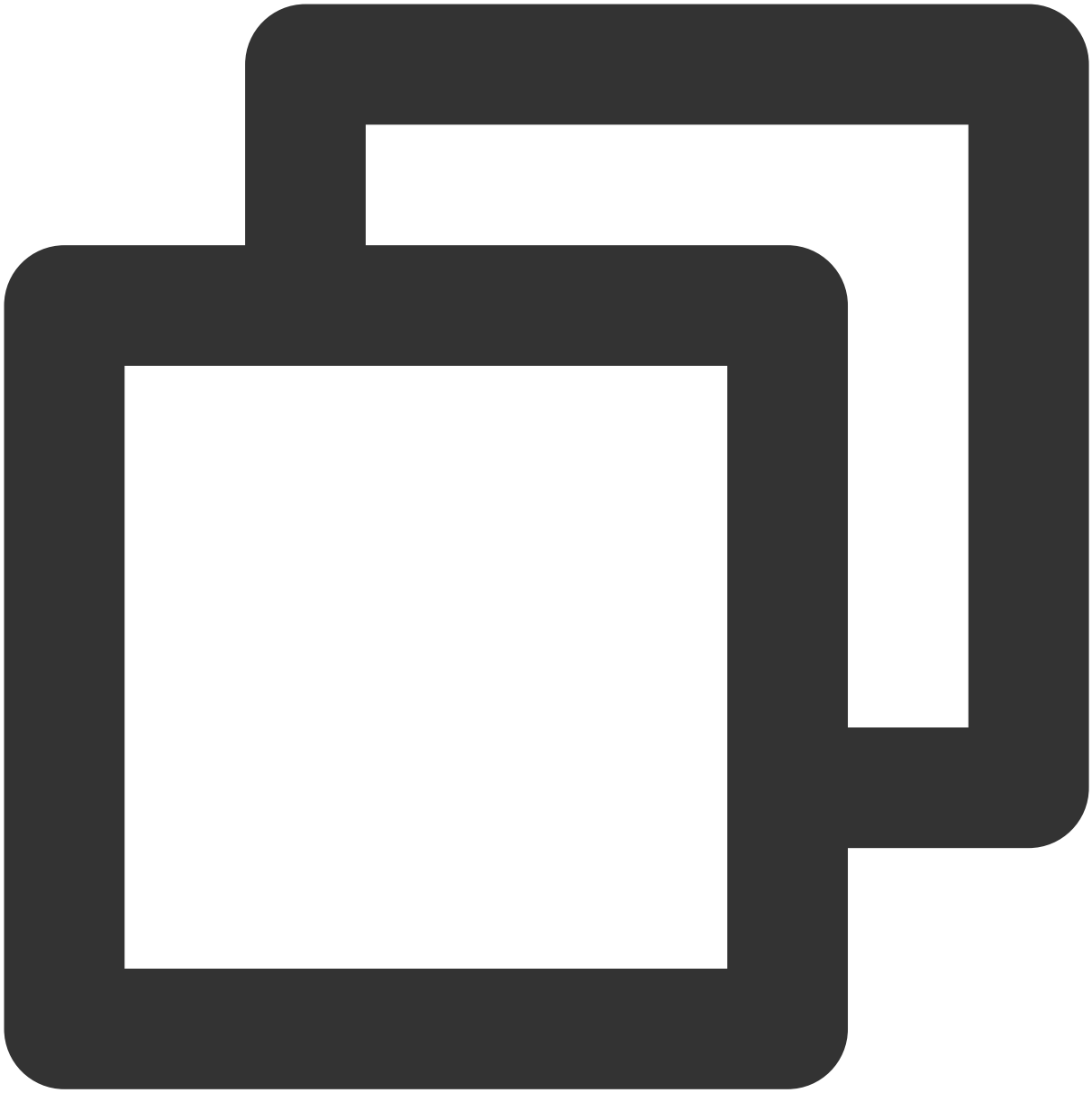
```
- (void)onError:(int)code message:(NSString * _Nullable)message;
```

参数如下表所示：

参数	类型	含义
code	int	错误码
message	NSString	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。



```
- (void)onCallReceived:(NSString *)callerId calleeIdList:(NSArray<NSString *> *)cal
```

参数如下表所示：

参数	类型	含义
callerId	NSString	主叫 ID（邀请方）
calleeIdList	NSArray	被叫 ID 列表（被邀请方）

groupId	NSString	群组通话 ID
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
userData	NSString	用户增加的扩展字段，详见:TUICallParams

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

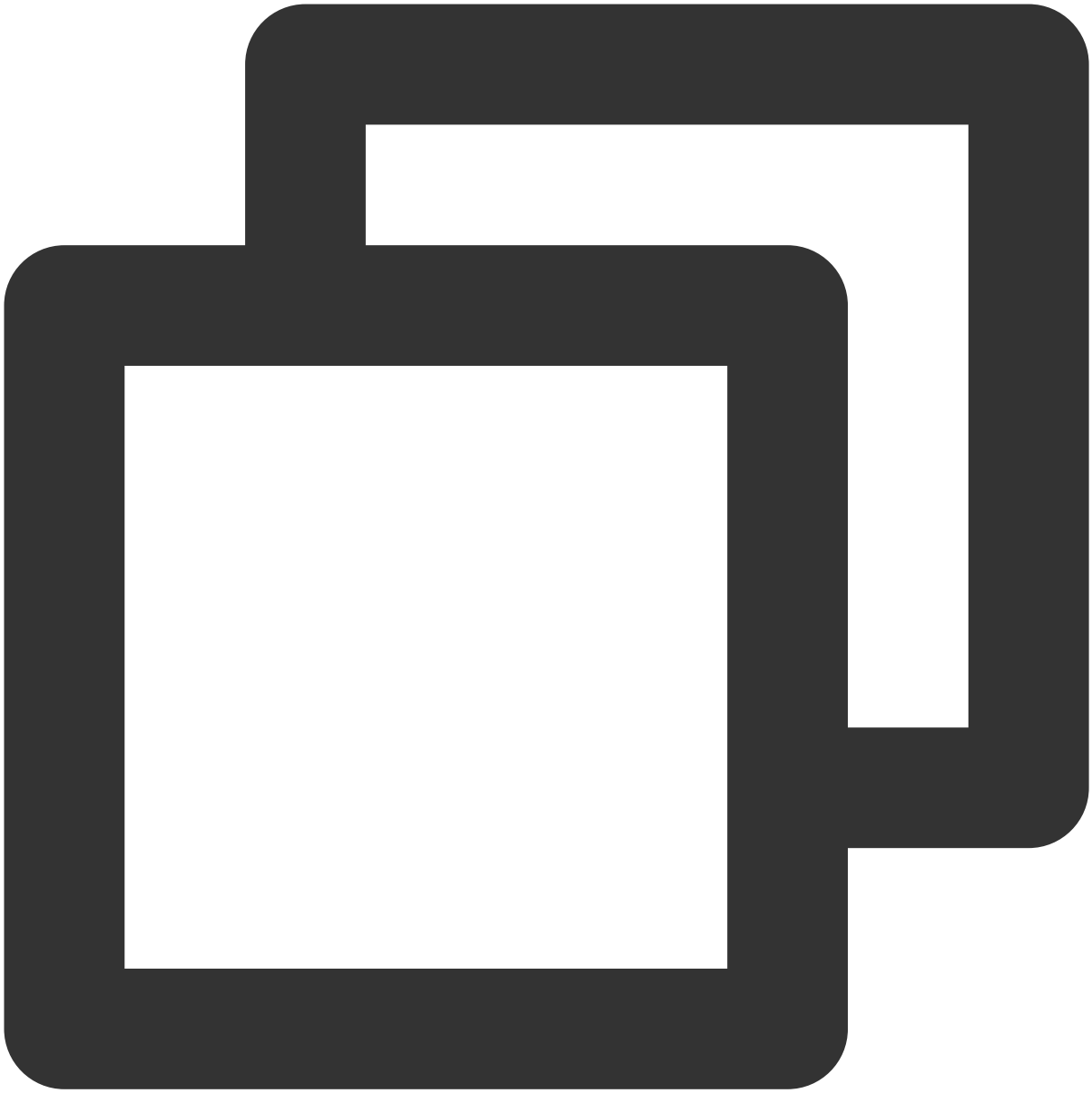
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）

异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）



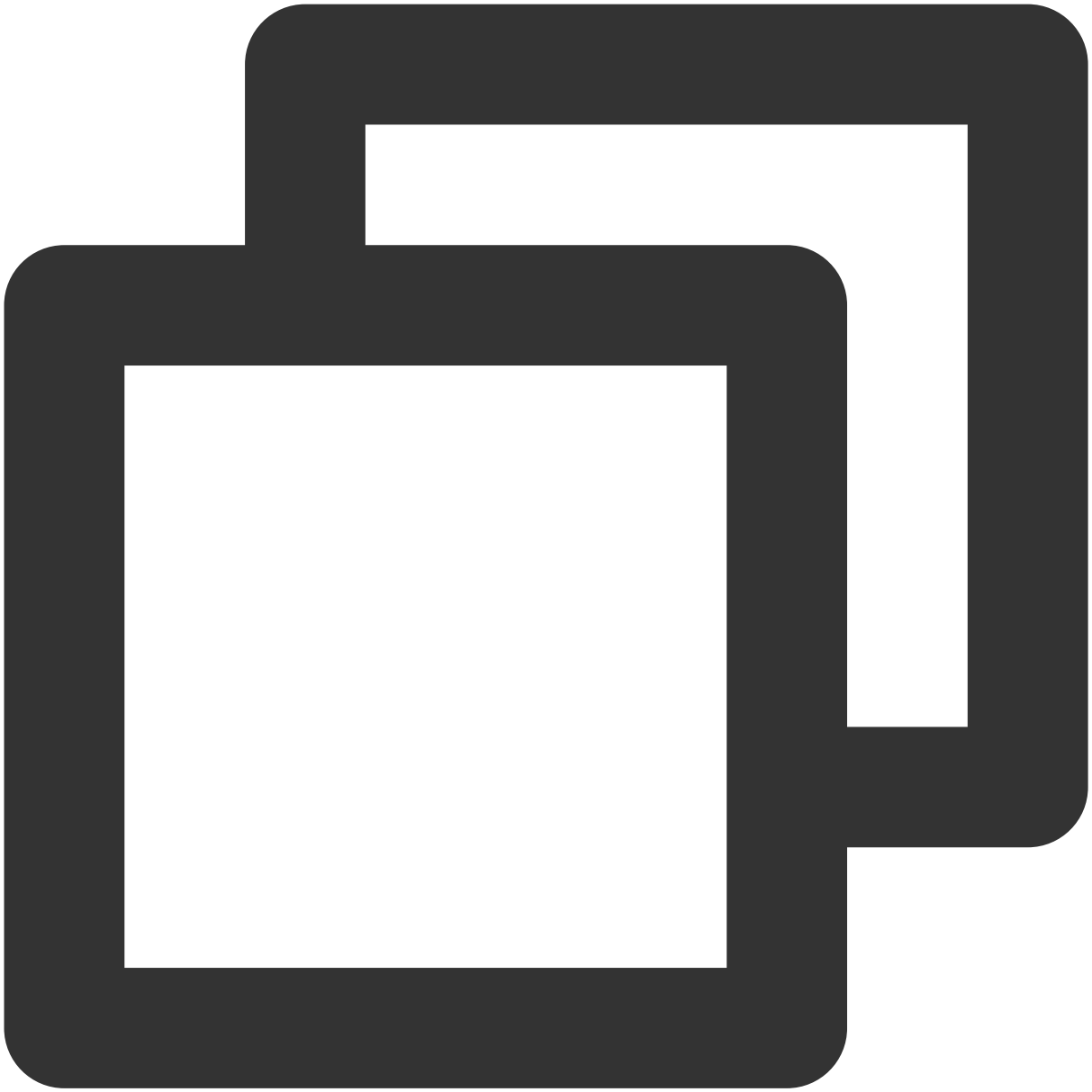
```
- (void)onCallCancelled:(NSString *)callerId;
```

参数如下表所示：

参数	类型	含义
callerId	NSString	用户的 ID

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。



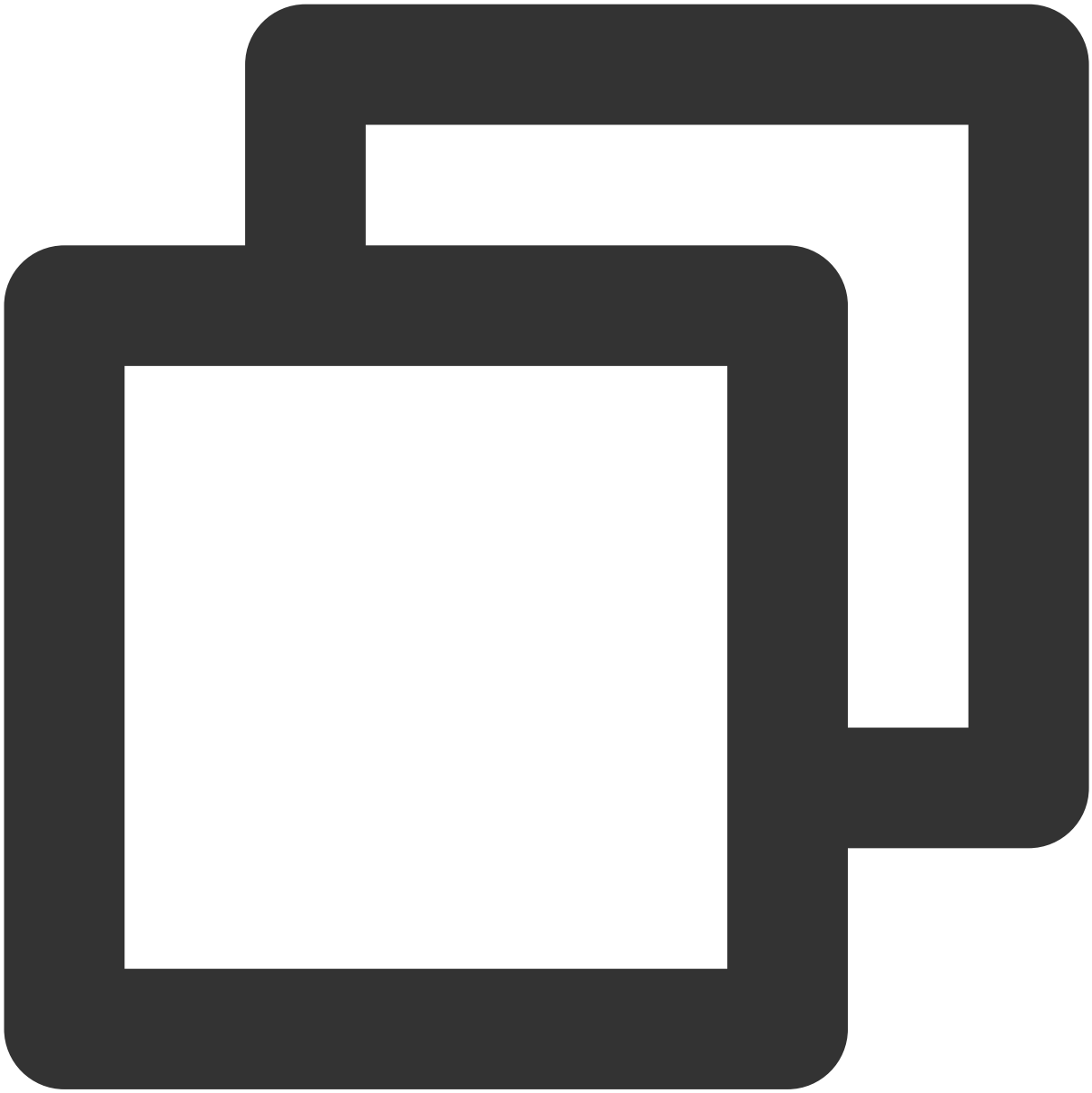
```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy
```

参数如下表所示：

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID
callMediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallRole	角色，枚举类型：主叫、被叫

onCallEnd

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。



```
- (void)onCallEnd:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaType
```

参数如下表所示：

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID

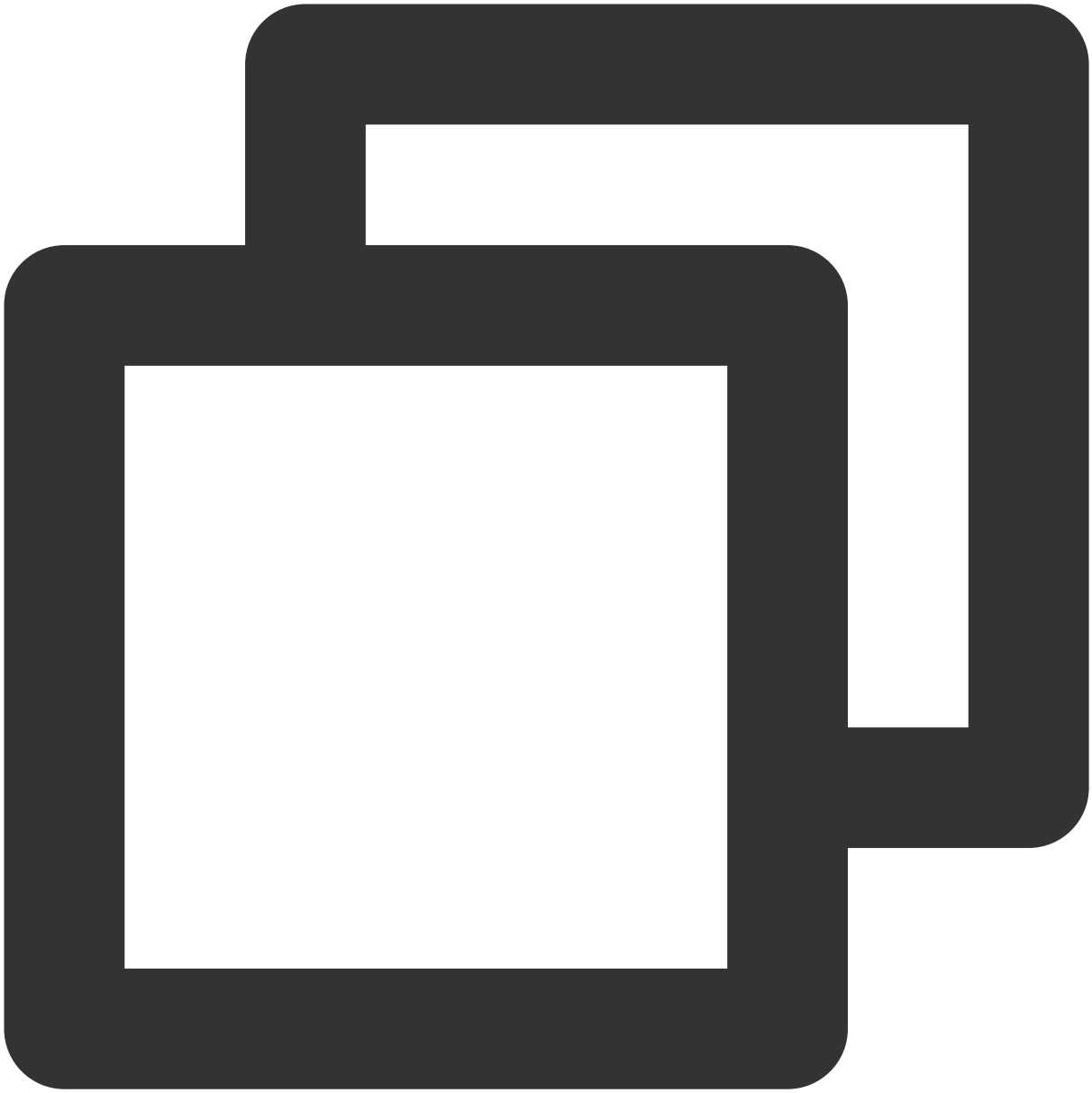
callMediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallRole	角色，枚举类型：主叫、被叫
totalTime	float	此次通话的时长

注意

客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。



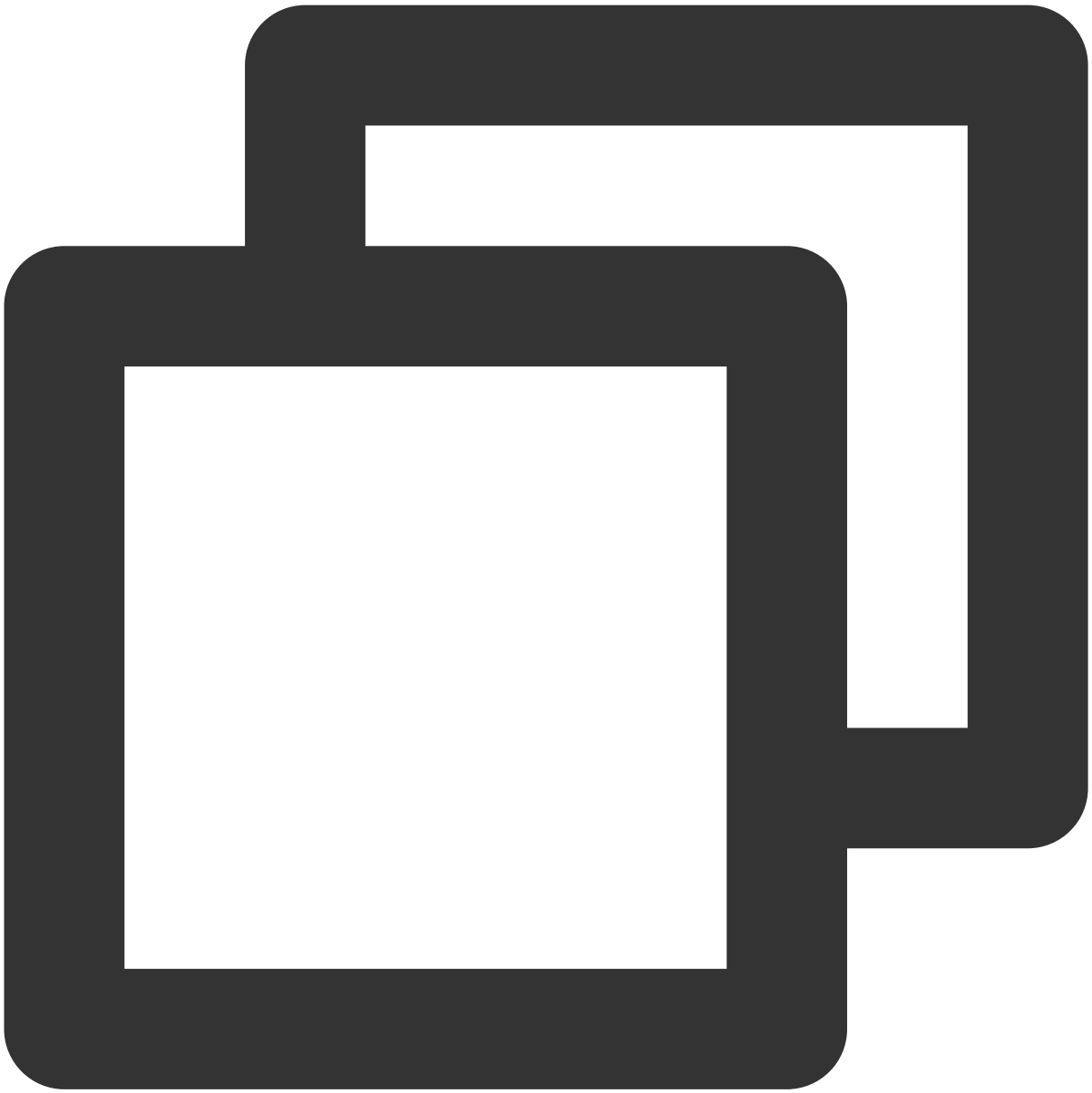
```
- (void)onCallMediaTypeChanged:(TUICallMediaType)oldCallMediaType newCallMediaType:
```

参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallMediaType	旧的通话类型
newCallMediaType	TUICallMediaType	新的通话类型

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。



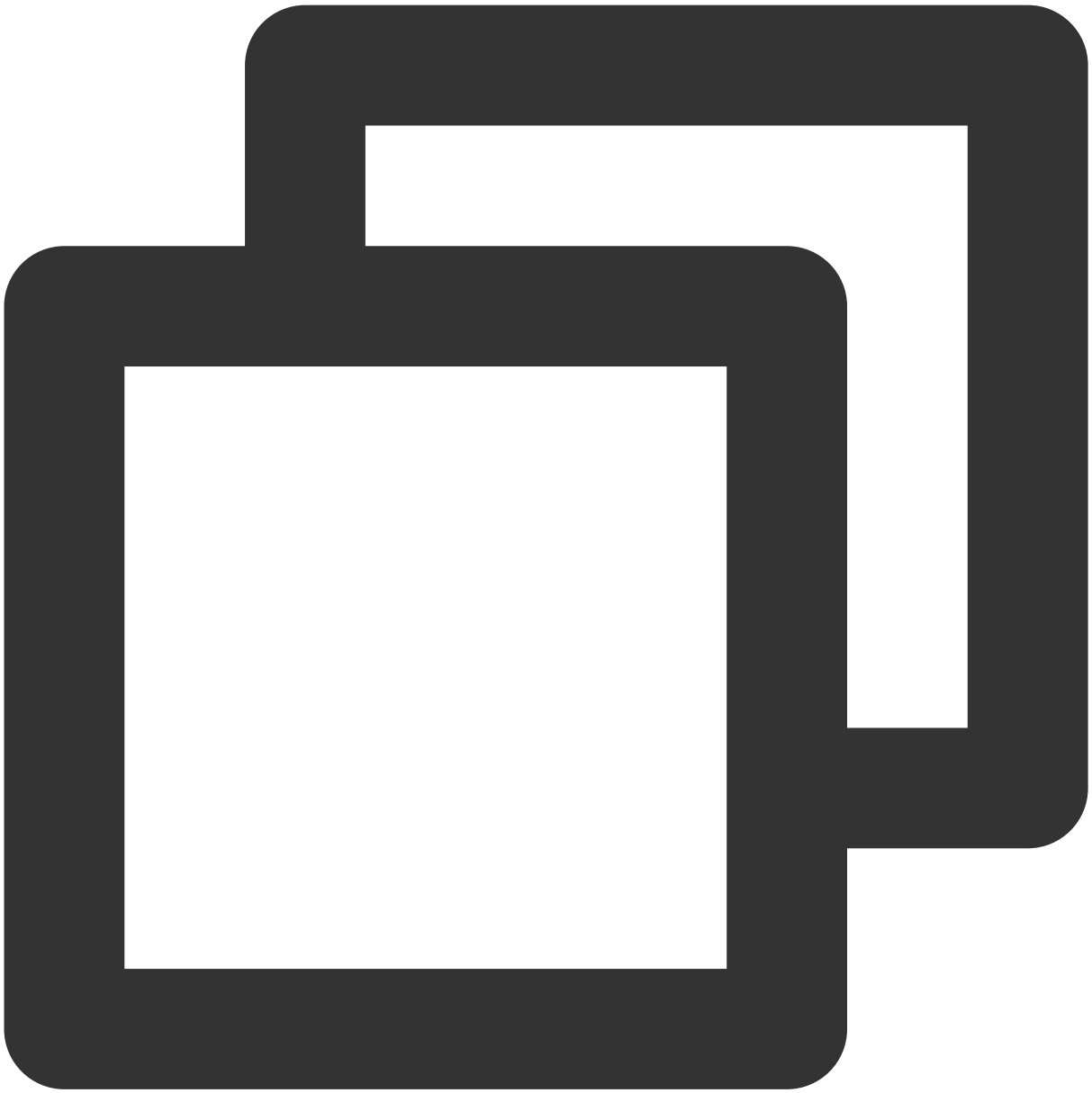
```
- (void)onUserReject:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。



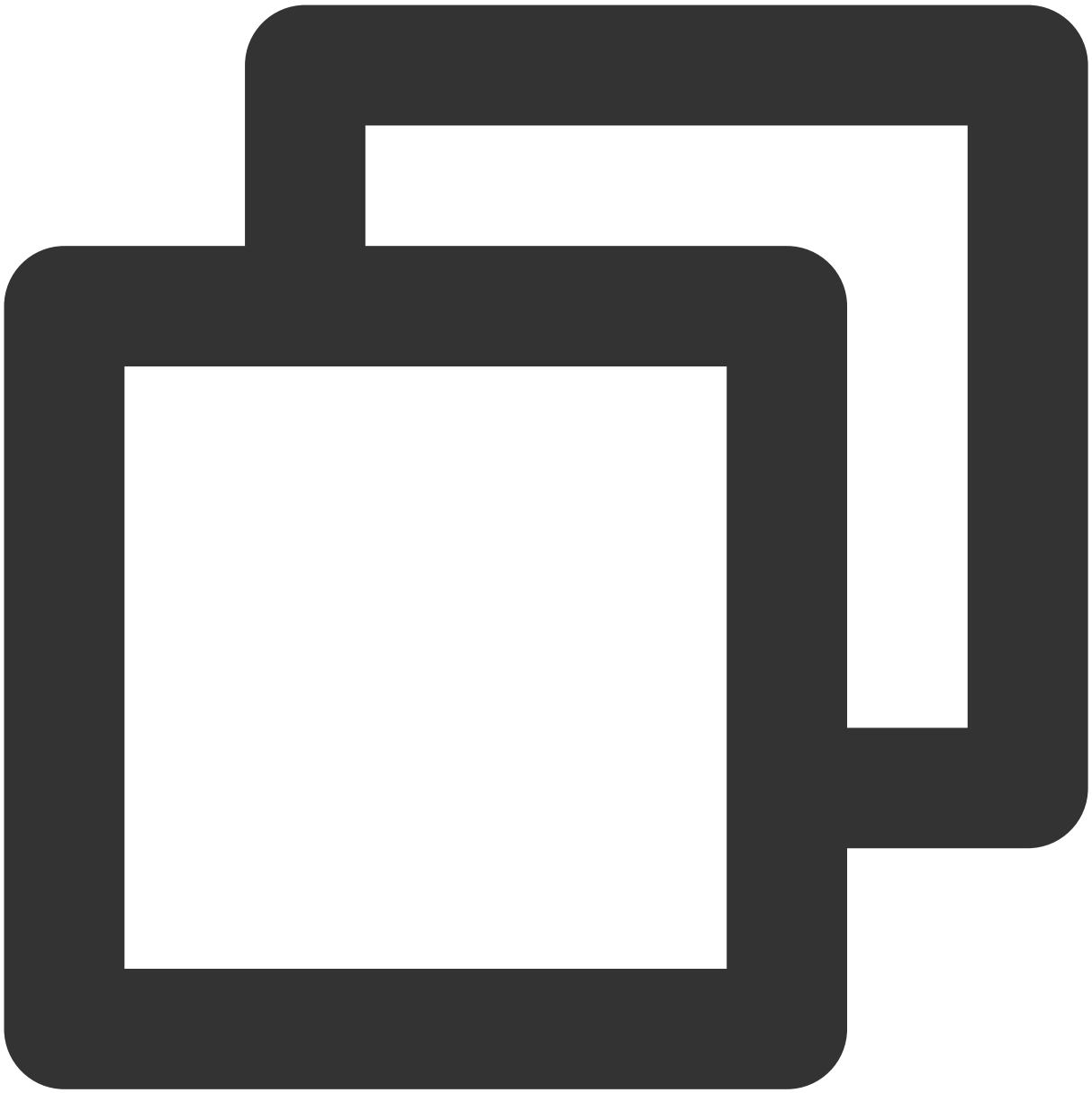
```
- (void)onUserNoResponse:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	无响应用户的 ID

onUserLineBusy

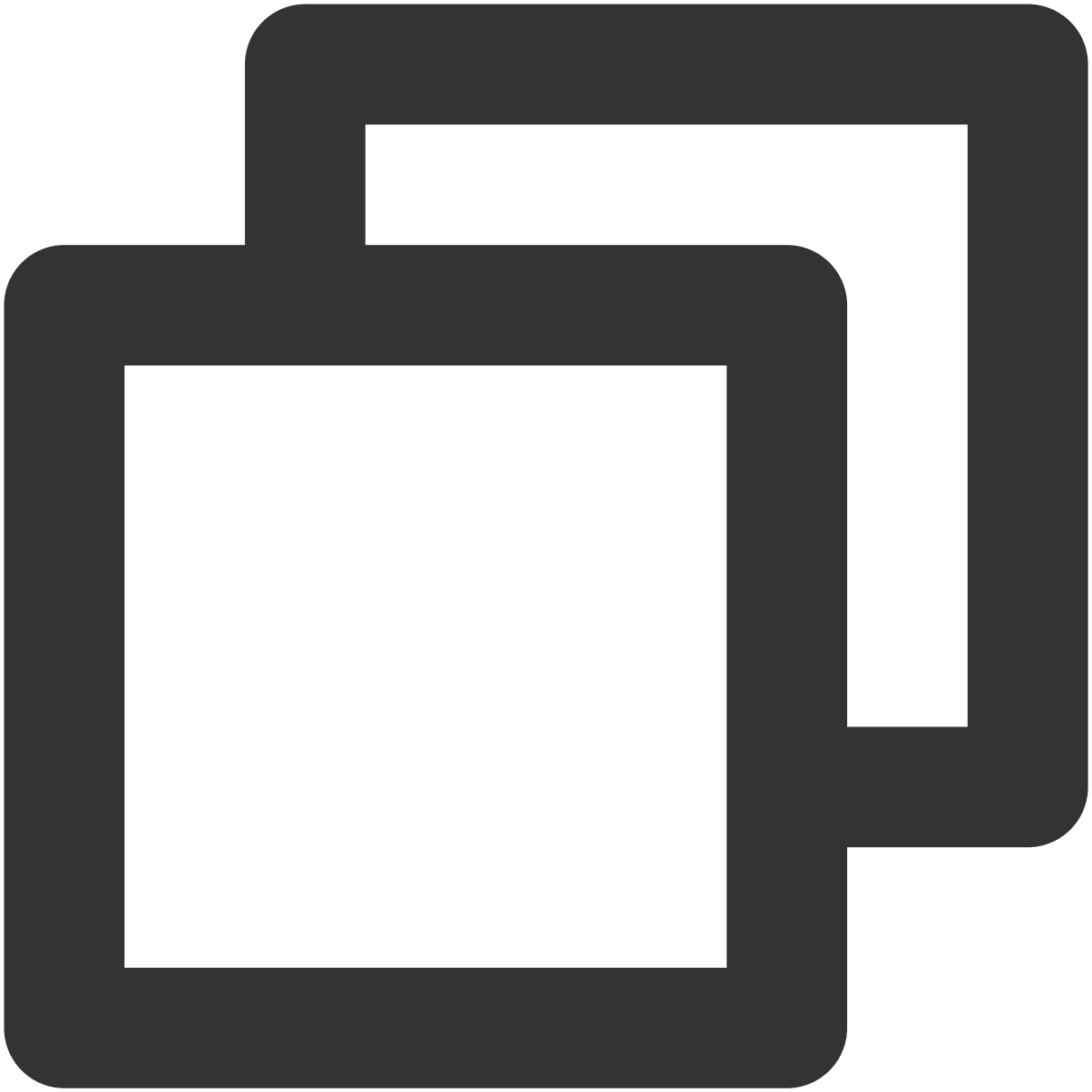
通话忙线回调。



```
- (void)onUserLineBusy:(NSString *)userId;
```

onUserJoin

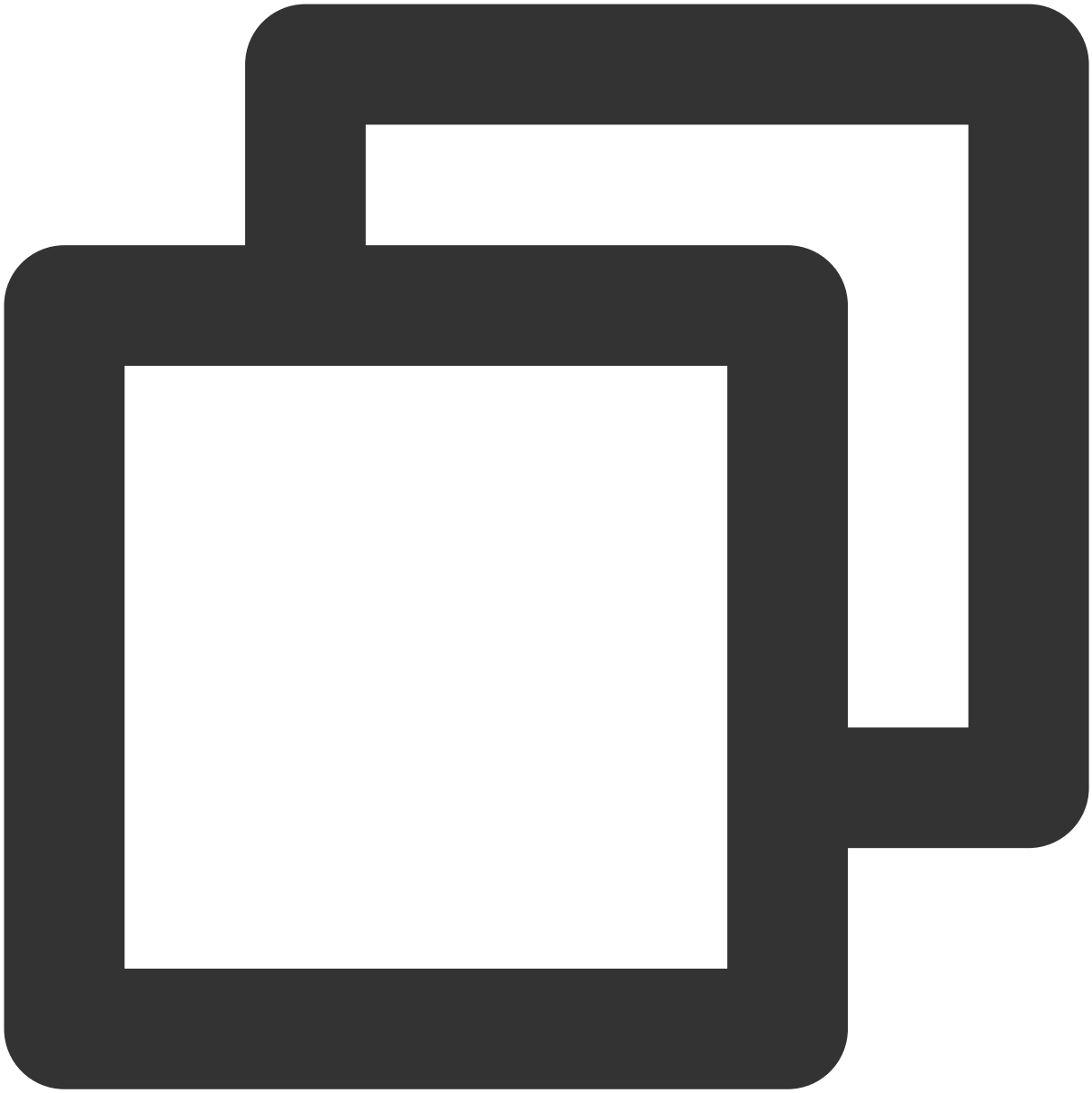
有用户进入此次通话的回调。



```
- (void)onUserJoin:(NSString *)userId;
```

onUserLeave

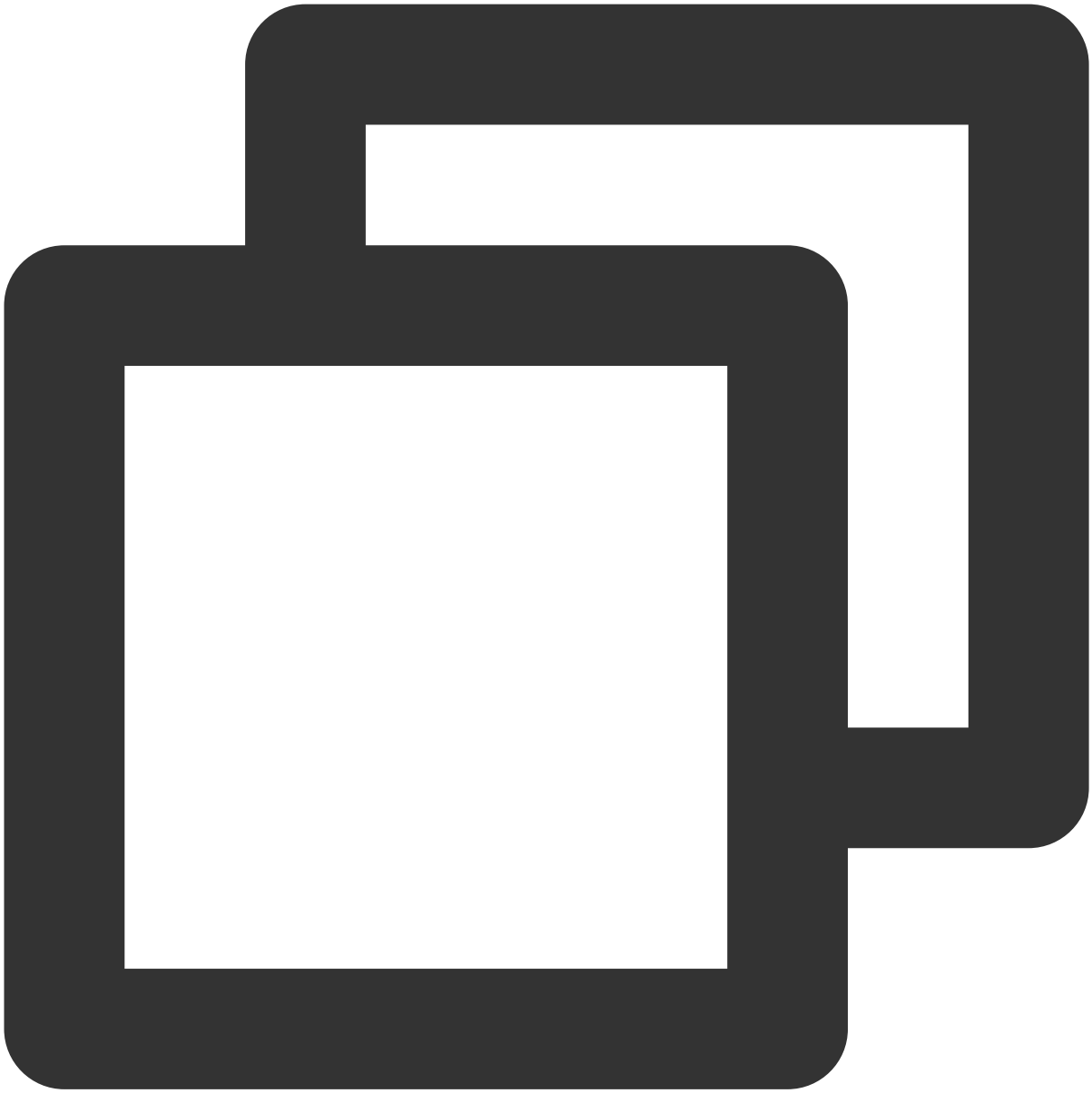
有用户离开此次通话的回调。



```
- (void)onUserLeave:(NSString *)userId;
```

onUserAudioAvailable

用户是否开启音频上行回调。



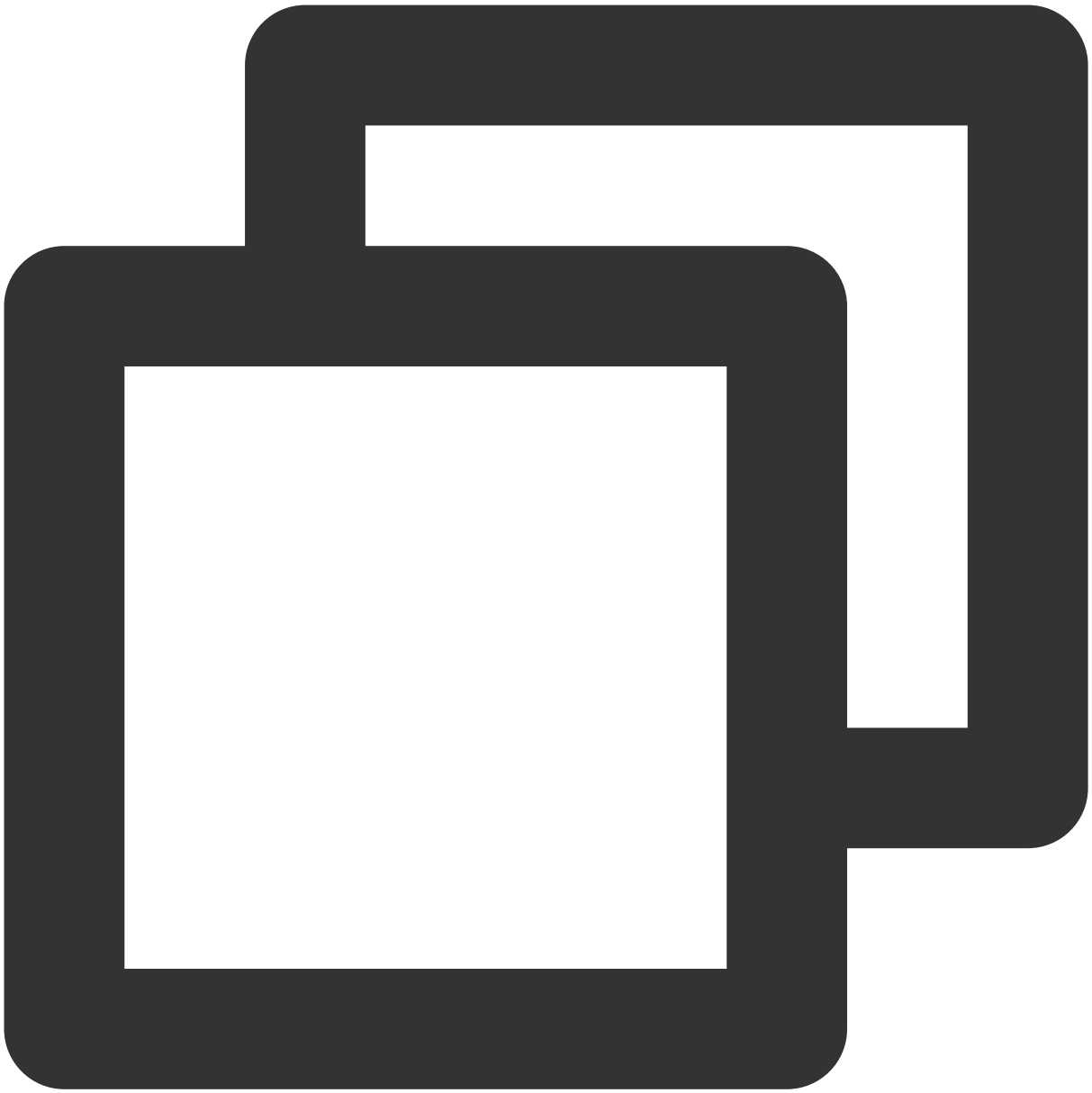
```
- (void)onUserAudioAvailable:(NSString *)userId isAudioAvailable:(BOOL)isAudioAvail
```

参数如下表所示：

参数	类型	含义
userId	NSString	用户 ID
isAudioAvailable	BOOL	用户音频是否可用

onUserVideoAvailable

用户是否开启视频上行回调。



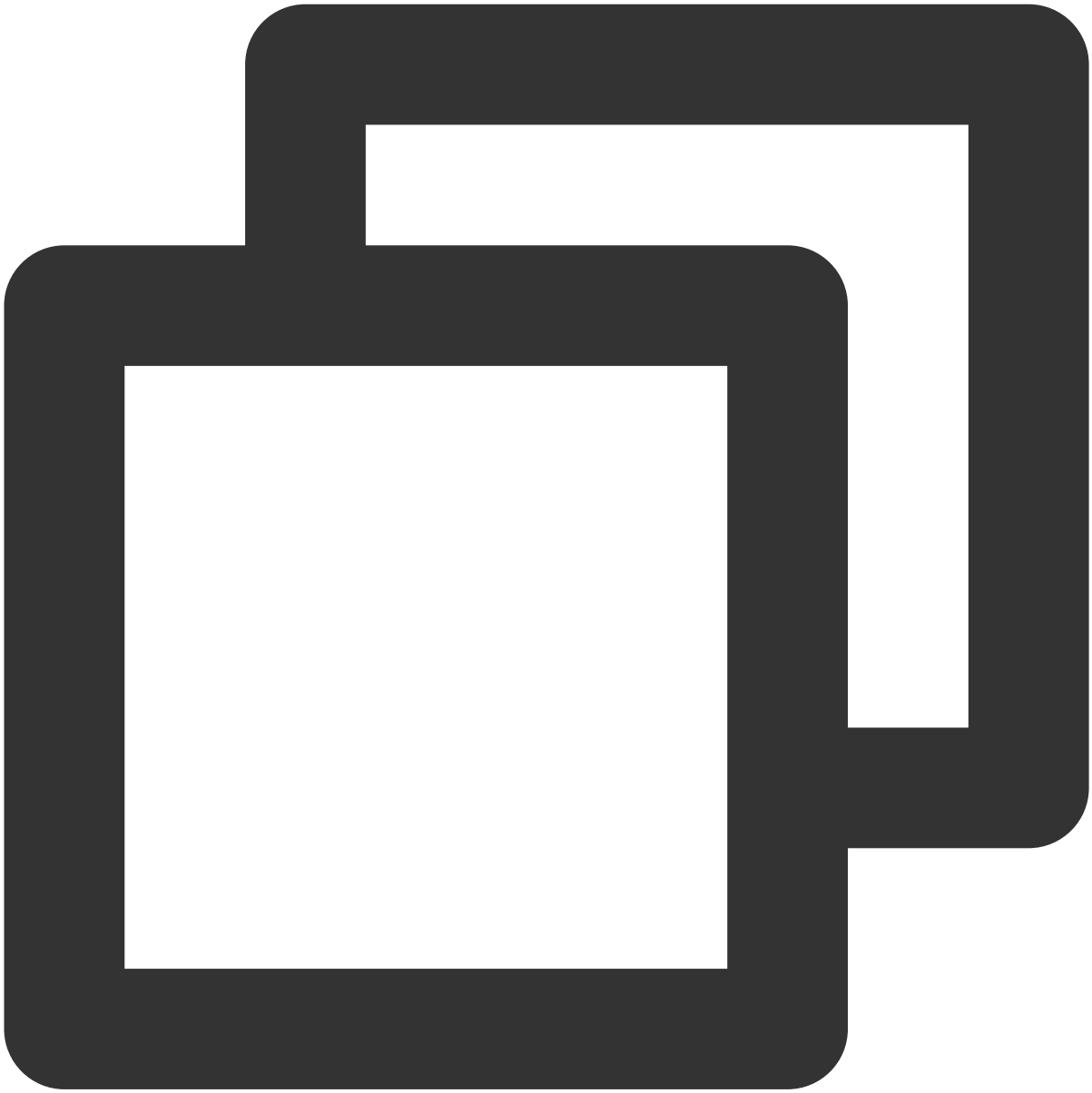
```
- (void)onUserVideoAvailable:(NSString *)userId isVideoAvailable:(BOOL)isVideoAvail
```

参数如下表所示：

参数	类型	含义
userId	NSString	通话用户 ID
isVideoAvailable	BOOL	用户视频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。



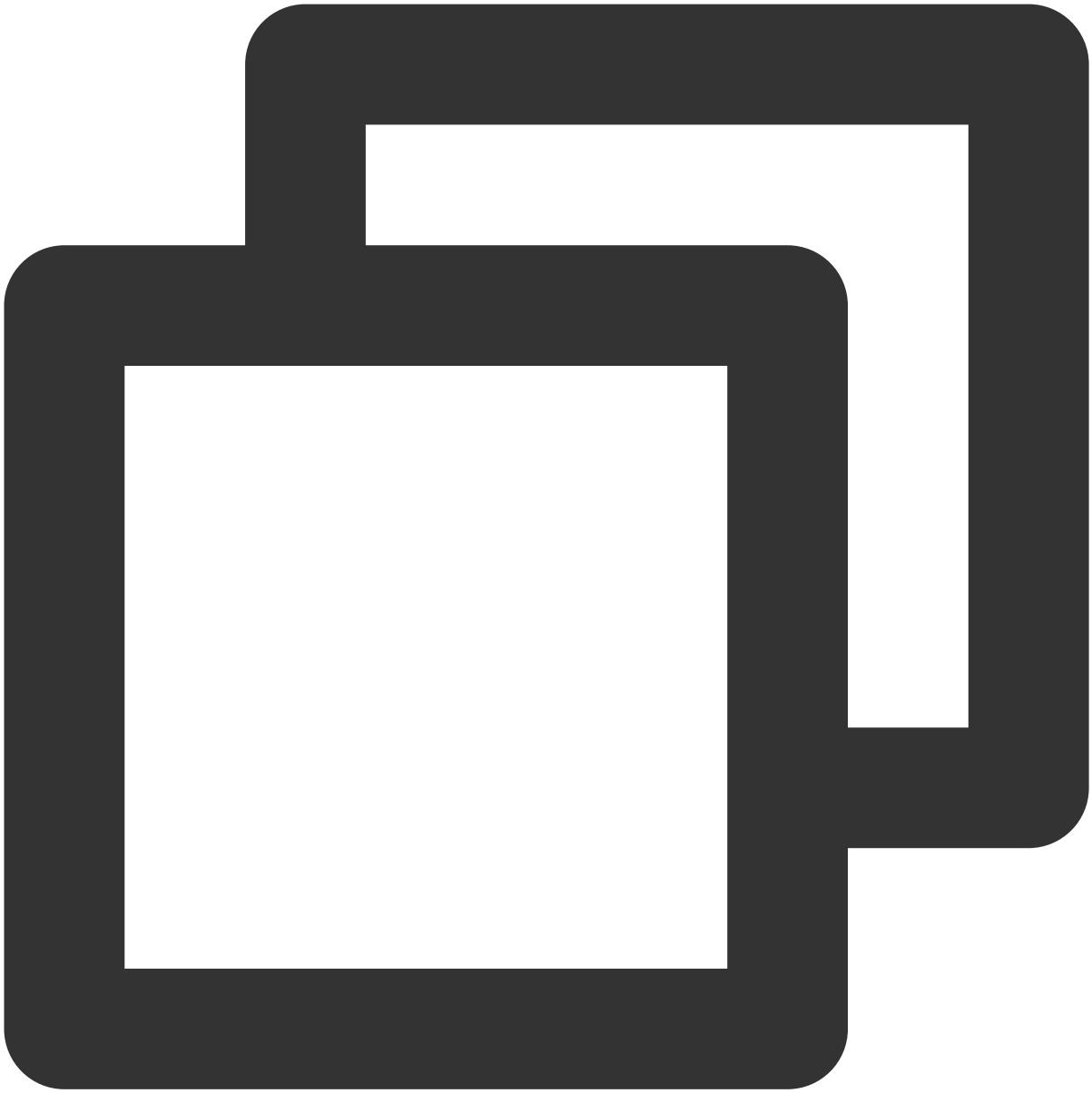
```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap
```

参数如下表所示：

参数	类型	含义
volumeMap	NSDictionary	音量表，根据每个 userId 可以获取对应的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。



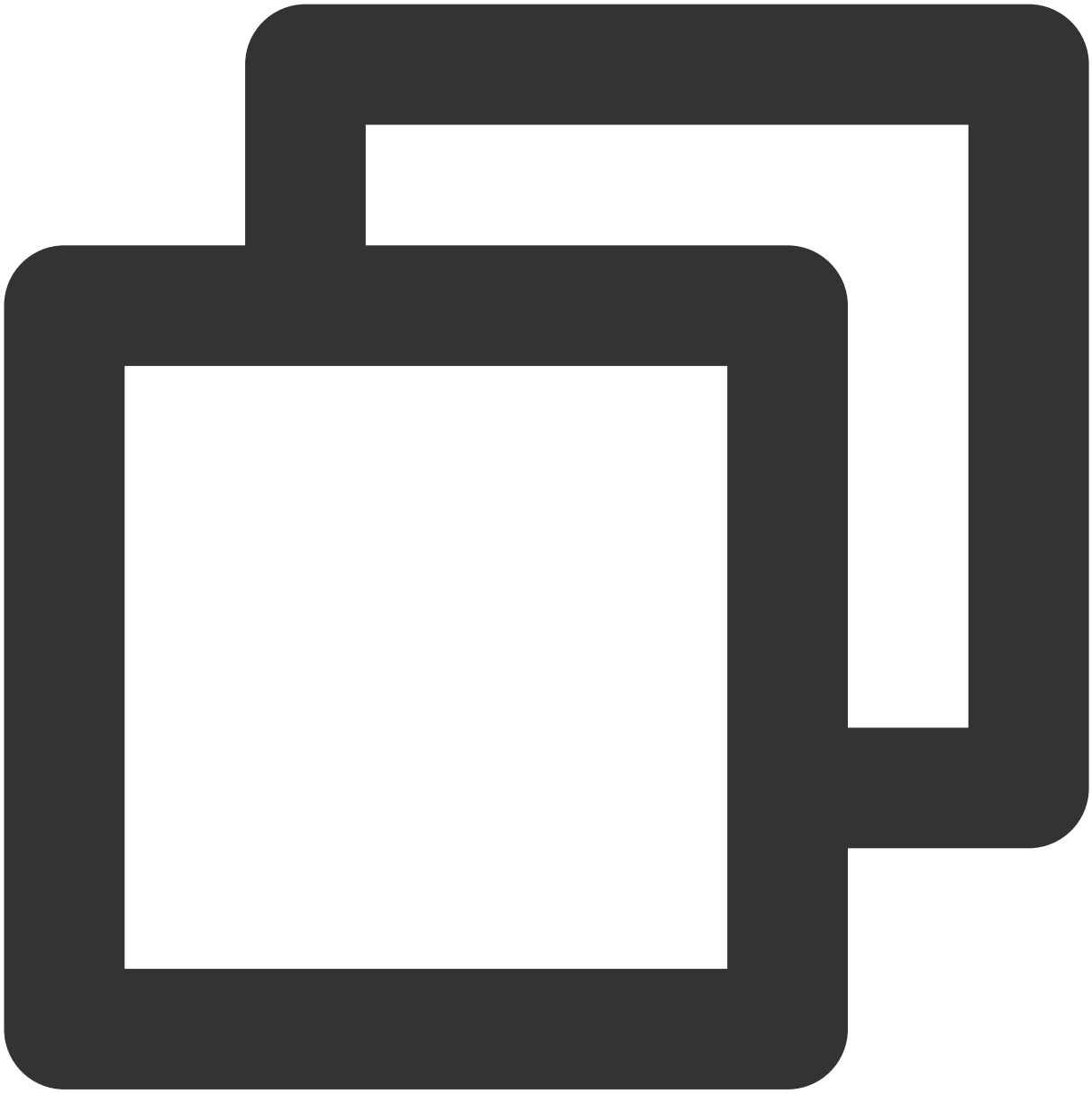
```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual
```

参数如下表所示：

参数	类型	含义
networkQualityList	NSArray	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

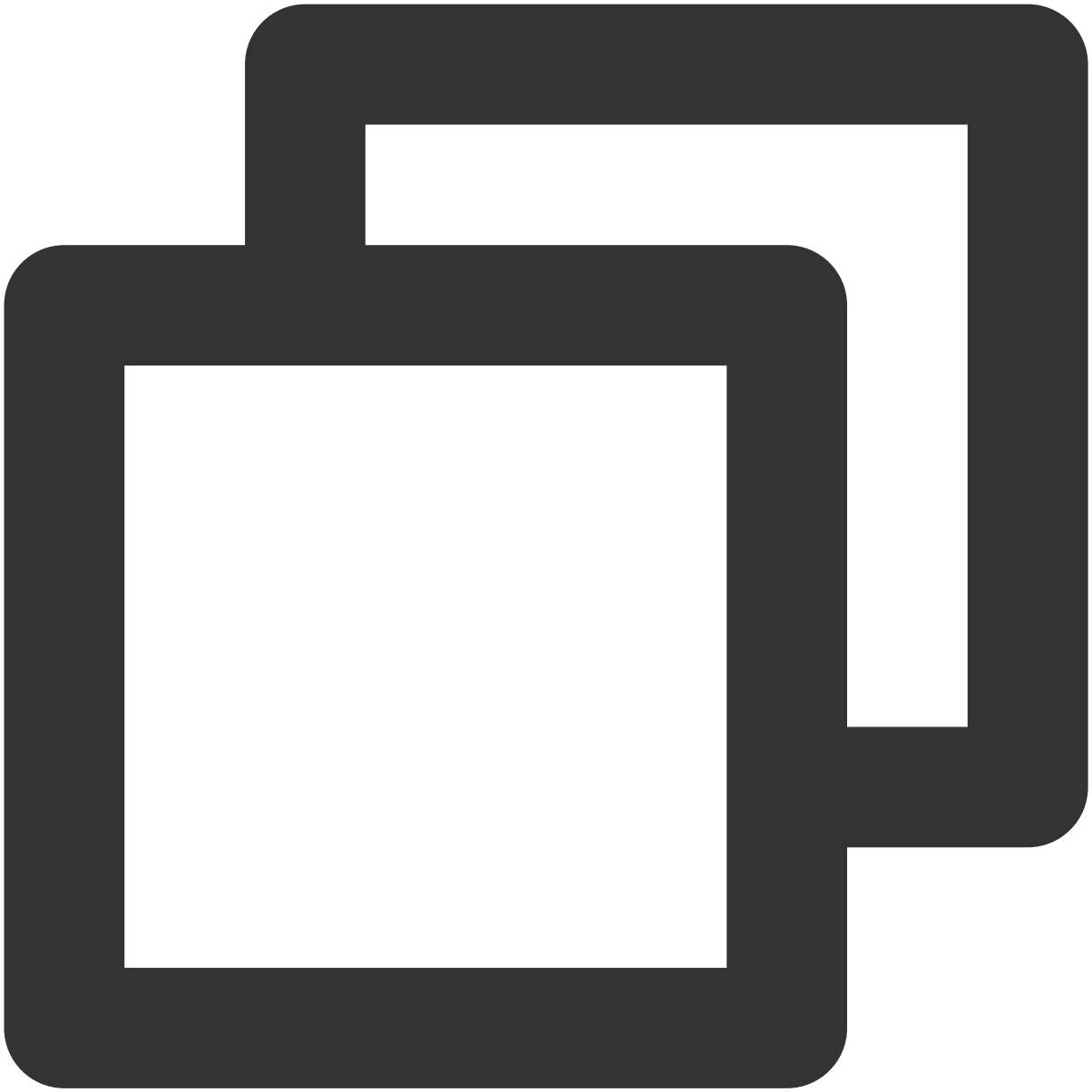
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
- (void)onKickedOffline;
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。



```
- (void)onUserSigExpired;
```

类型定义

最近更新时间：2023-06-21 16:29:42

常用结构

TUICallDefine

类型	描述
TUICallParams	通话参数
TUIOfflinePushInfo	离线推送厂商配置信息

TUICommonDefine

类型	描述
TUIRoomId	通话中音视频房间 ID
TUINetworkQuality	网络质量
TUIVideoRenderParams	视频画面的渲染参数
TUIVideoEncoderParams	视频编码参数

枚举定义

TUICallDefine

类型	说明
TUICallMediaType	通话类型
TUICallRole	通话角色
TUICallStatus	通话状态
TUICallScene	通话场景
TUICallIOSOfflinePushType	iOS 离线推送类型

TUICommonDefine

类型	说明

版权所有：腾讯云计算（北京）有限责任公司

TUIAudioPlaybackDevice	音频设备
TUICamera	摄像头类型
TUINetworkQuality	网络质量
TUIVideoRenderParamsFillMode	视频画面填充模式
TUIVideoRenderParamsRotation	视频画面旋转方向
TUIVideoEncoderParamsResolutionMode	视频宽高比模式
TUIVideoEncoderParamsResolution	视频分辨率

TUICallParams

通话参数

参数	类型	描述
roomId	TUIRoomId	通话中音视频房间ID
offlinePushInfo	TUIOfflinePushInfo	厂商离线推送配置信息
timeout	int	通话超时时间，默认：30s，单位：秒
userData	NSString	发起通话时自定义扩展字段，被叫端在 onCallReceived 中回调

TUIOfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参考：离线唤醒

参数	类型	描述
title	NSString	离线推送展示通知栏标题
desc	NSString	离线推送展示通知栏内容
ignoreIOSBadge	BOOL	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加。
iOSSound	NSString	离线推送声音设置（仅对 iOS 生效）。 当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。 当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。 如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工

		程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	NSString	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。 只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声 另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效
androidOPPOChannelID	NSString	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidVIVOClassification	NSInteger	VIVO 推送消息分类 (待废弃接口，VIVO 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1
androidXiaoMiChannelID	NSString	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	NSString	FCM 通道手机 8.0 系统及以上的渠道 ID
androidHuaWeiCategory	NSString	华为推送消息分类，详见：华为消息分类标准
isDisablePush	BOOL	是否关闭推送（默认开启推送）
iOSPushType	TUICallIOSOfflinePushType	iOS 离线推送类型，默认：APNs

TUIRoomId

通话中音视频房间ID

(1) `intRoomId` 与 `strRoomId` 是互斥的，若您选用 `strRoomId`，则 `intRoomId` 需要填写为 0。若两者都填，SDK 将优先选用 `intRoomId`。

(2) 不要混用 `intRoomId` 和 `strRoomId`，因为它们之间是不互通的，比如数字123和字符串123是两个完全不同的房间。

参数	类型	描述
intRoomId	UInt32	数字房间号 **取值范围** 1 - 2147483647(2^31-1)
strRoomId	NSString	字符串房间号 **推荐取值** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）：

		大小写英文字母（a-zA-Z）； 数字（0-9）； 空 格、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<`、`=`

注意：

字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！

TUIVideoRenderParams

视频画面的渲染参数

参数	类型	描述
fillMode	TUIVideoRenderParamsFillMode	视频画面填充模式
rotation	TUIVideoRenderParamsRotation	视频画面旋转方向

TUIVideoEncoderParams

视频编码参数

参数	类型	描述
resolution	TUIVideoEncoderParamsResolution	视频分辨率
resolutionMode	TUIVideoEncoderParamsResolutionMode	视频宽高比

TUICallMediaType

通话类型

类型	取值	描述
TUICallMediaTypeUnknown	0	未知类型
TUICallMediaTypeAudio	1	语音通话
TUICallMediaTypeVideo	2	视频通话

TUICallRole

通话角色

类型	取值	描述
TUICallRoleNone	0	未知类型

版权所有：腾讯云计算（北京）有限责任公司

TUICallRoleCall	1	主叫（邀请方）
TUICallRoleCalled	2	被叫（被邀请方）

TUICallStatus

通话状态

类型	取值	描述
TUICallStatusNone	0	未定义
TUICallStatusWaiting	1	通话等待中
TUICallStatusAccept	2	通话已接听

TUICallScene

通话场景

类型	取值	描述
TUICallSceneGroup	0	群组通话
TUICallSceneMulti	1	匿名群组通话（暂不支持，敬请期待）
TUICallSceneSingle	2	单人通话

TUICallIOSOfflinePushType

iOS离线推送类型

类型	取值	描述
TUICallIOSOfflinePushTypeAPNs	0	普通推送
TUICallIOSOfflinePushTypeVoIP	1	VoIP 推送

TUIAudioPlaybackDevice

音频设备

类型	取值	描述
TUIAudioPlaybackDeviceSpeakerphone	0	扬声器

版权所有：腾讯云计算（北京）有限责任公司

TUIAudioPlaybackDeviceEarpiece	1	听筒

TUICamera

前置/后置 摄像头

类型	取值	描述
TUICameraFront	0	前置摄像头
TUICameraBack	1	后置摄像头

TUINetworkQuality

网络质量

类型	取值	描述
TUINetworkQualityUnknown	0	未定义
TUINetworkQualityExcellent	1	当前网络非常好
TUINetworkQualityGood	2	当前网络比较好
TUINetworkQualityPoor	3	当前网络一般
TUINetworkQualityBad	4	当前网络较差
TUINetworkQualityVbad	5	当前网络很差
TUINetworkQualityDown	6	当前网络不满足通话的最低要求

TUIVideoRenderParamsFillMode

视频画面填充模式

类型	取值	描述
TUIVideoRenderParamsFillModeFill	0	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整
TUIVideoRenderParamsFillModeFit	1	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边

版权所有：腾讯云计算（北京）有限责任公司

TUIVideoRenderParamsRotation

视频画面旋转方向

类型	取值	描述
TUIVideoRenderParamsRotation_0	0	不旋转
TUIVideoRenderParamsRotation_90	1	顺时针旋转90度
TUIVideoRenderParamsRotation_180	2	顺时针旋转180度
TUIVideoRenderParamsRotation_270	3	顺时针旋转270度

TUIVideoEncoderParamsResolutionMode

视频宽高比模式

类型	取值	描述
TUIVideoEncoderParamsResolutionModeLandscape	0	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360
TUIVideoEncoderParamsResolutionModePortrait	1	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640

TUIVideoEncoderParamsResolution

视频分辨率

类型	取值	描述
TUIVideoEncoderParamsResolution_640_360	108	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps
TUIVideoEncoderParamsResolution_640_480	62	宽高比 4:3；分辨率 640x480；建议码率（VideoCall）600kbps
TUIVideoEncoderParamsResolution_960_540	110	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps
TUIVideoEncoderParamsResolution_960_720	64	宽高比 4:3；分辨率 960x720；建议码率（VideoCall）1000kbps
TUIVideoEncoderParamsResolution_1280_720	112	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps

版权所有：腾讯云计算（北京）有限责任公司

TUIVideoEncoderParamsResolution_1920_1080	114	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps

版权所有：腾讯云计算（北京）有限责任公司

Web
API 概览

最近更新时间：2024-04-03 17:23:11

TUICallKit（含 UI 组件）

TUICallKit 是音视频通话组件的**含 UI 组件**，您可以通过该组件快速实现一个类微信的音视频通话场景。
<TUICallKit/>：UI 通话组件主体。
<TUICallKitMini/>：UI 通话悬浮窗，若开启最小化功能，则 <TUICallKitMini/> 必须被放置在页面中。**注意：v2.3.2+ 已废弃**。
TUICallKitServer 是通话实例，提供的 API 接口如下。

API	描述
init	初始化 TUICallKit。
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
setSelfInfo	设置用户昵称和头像
enableMuteMode	开启/关闭来电铃声
enableFloatWindow	开启/关闭悬浮窗功能
setLanguage	设置 TUICallKit 组件通话语言
destroyed	销毁 TUICallKit

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）

destroyInstance	销毁 TUICallEngine 实例（单例模式）
on	监听事件
off	取消监听事件
login	登录接口
logout	登出接口
setSelfInfo	设置用户昵称和头像
call	发起 1v1 通话
groupCall	群聊邀请通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
switchCallMediaType	音视频通话切换
startRemoteView	启动远端画面渲染
stopRemoteView	停止远端画面渲染
startLocalView	启动本地画面渲染，**注意：后续将废弃，建议使用 openCamera**
stopLocalView	停止本地画面渲染，**注意：后续将废弃，建议使用 closeCamera**
openCamera	开启摄像头
closeCamara	关闭摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
setVideoQuality	设置视频质量
getDeviceList	获取设备列表
switchDevice	切换摄像头或麦克风设备
enableAIVoice	开启/关闭 AI 降噪

版权所有：腾讯云计算（北京）有限责任公司

事件类型定义

TUICallEvent 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

EVENT	描述
TUICallEvent.ERROR	SDK 内部发生了错误
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该回调
TUICallEvent.KICKED_OUT	重复登录，收到该回调说明被踢出房间
TUICallEvent.USER_ACCEPT	如果有用户接听，那么会收到此回调
TUICallEvent.USER_ENTER	如果有用户同意进入通话，那么会收到此回调
TUICallEvent.USER_LEAVE	如果有用户同意离开通话，那么会收到此回调
TUICallEvent.REJECT	用户拒绝通话
TUICallEvent.NO_RESP	邀请用户无应答
TUICallEvent.LINE_BUSY	邀请方忙线
TUICallEvent.CALLING_TIMEOUT	作为被邀请方会收到，收到该回调说明本次通话超时未应答
TUICallEvent.USER_VIDEO_AVAILABLE	远端用户开启/关闭了摄像头, 会收到该回调
TUICallEvent.USER_AUDIO_AVAILABLE	远端用户开启/关闭了麦克风, 会收到该回调
TUICallEvent.USER_VOICE_VOLUME	远端用户说话音量调整, 会收到该回调
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该回调
TUICallEvent.INVITED	被邀请进行通话
TUICallEvent.CALLING_CANCEL	作为被邀请方会收到，收到该回调说明本次通话被取消了
TUICallEvent.CALLING_END	收到该回调说明本次通话结束了
TUICallEvent.DEVICED_UPDATED	设备列表更新收到该回调
TUICallEvent.CALL_TYPE_CHANGED	通话类型切换收到该回调

文档链接

TUICallEngine

TUICallEvent

TUICallKit

最近更新时间：2024-04-03 17:23:11

API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详见：快速接入 TUICallKit。

API 概览

TUICallKit 是音视频通话组件的**含 UI 组件**，您可以通过该组件快速实现一个类微信的音视频通话场景。
<TUICallKit/>：UI 通话组件主体。
TUICallKitServer 是通话实例，提供的 API 接口如下。

API	描述
init	初始化 TUICallKit 组件实例
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
setSelfInfo	设置用户昵称和头像
enableMuteMode	开启/关闭来电铃声
enableFloatWindow	开启/关闭悬浮窗功能
setLanguage	设置 TUICallKit 组件通话语言
destroyed	销毁 TUICallKit 组件实例

`<TUICallKit/>` 属性介绍

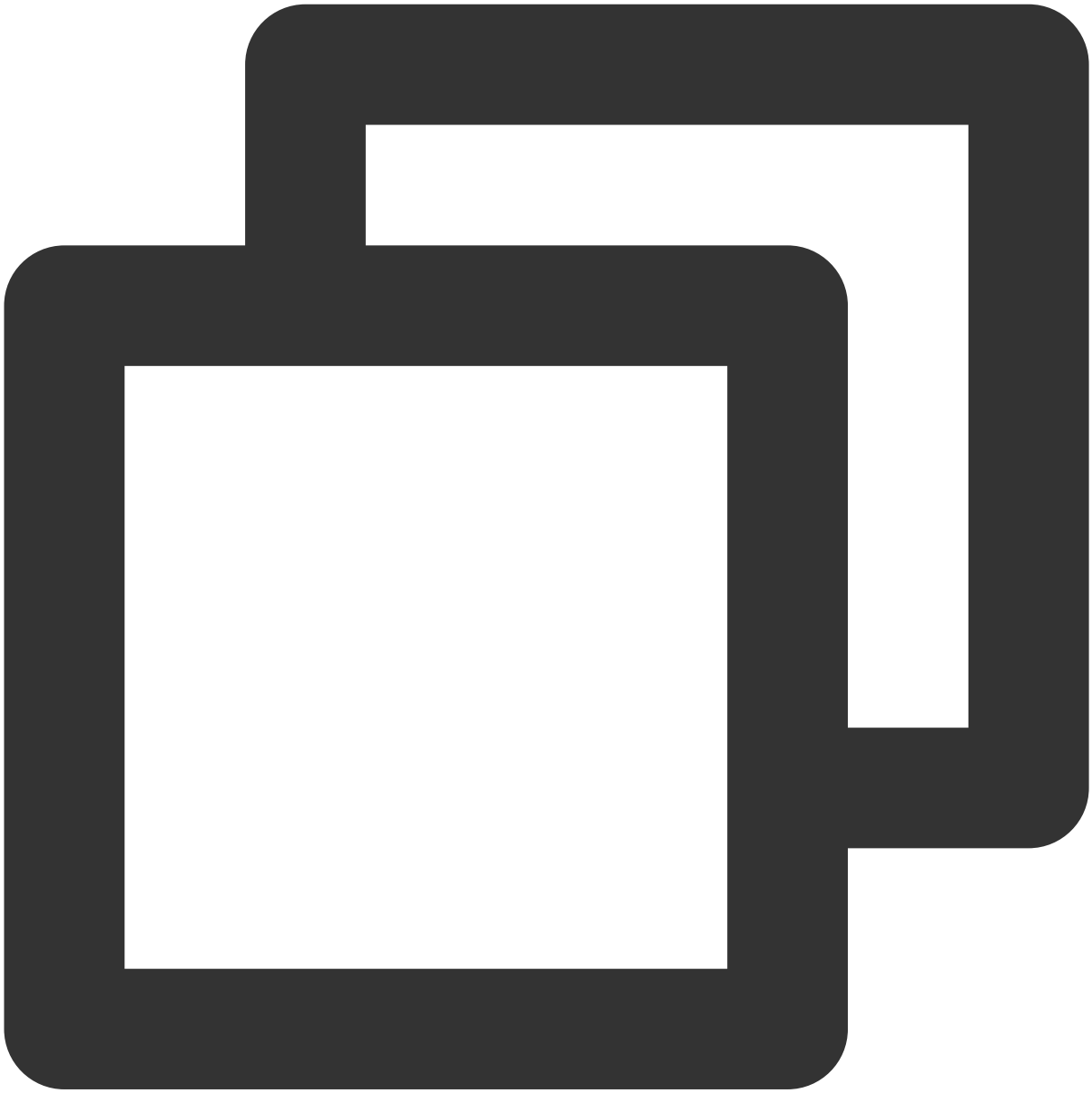
属性概览

属性	描述	类型	是否必填	默认值
allowedMinimized	是否允许悬浮窗	boolean	否	false
allowedFullScreen	是否允许通话界面全屏	boolean	否	true
videoDisplayMode	通话界面显示模式	VideoDisplayMode	否	VideoDisplayMode.COVER
videoResolution	通话分辨率	VideoResolution	否	VideoResolution.RESOLUTION_480P
beforeCalling	拨打电话前与收到通话邀请前会执行此函数	function(type, error)	否	-
afterCalling	结束通话后会执行此函数	function()	否	-
onMinimized	组件切换最小化状态时会执行此函数，STATUS 值说明	function(oldStatus, newStatus)	否	-
kickedOut	组件抛出的事件，当前登录用户被踢出登录时会触发该事件，通话也会自动结束	function()	否	-
statusChanged	组件抛出的事件，当通话状态发生变化时，会触发该事件，通话状态种类详见，STATUS 值说明	function({oldStatus, newStatus})	否	-

示例代码

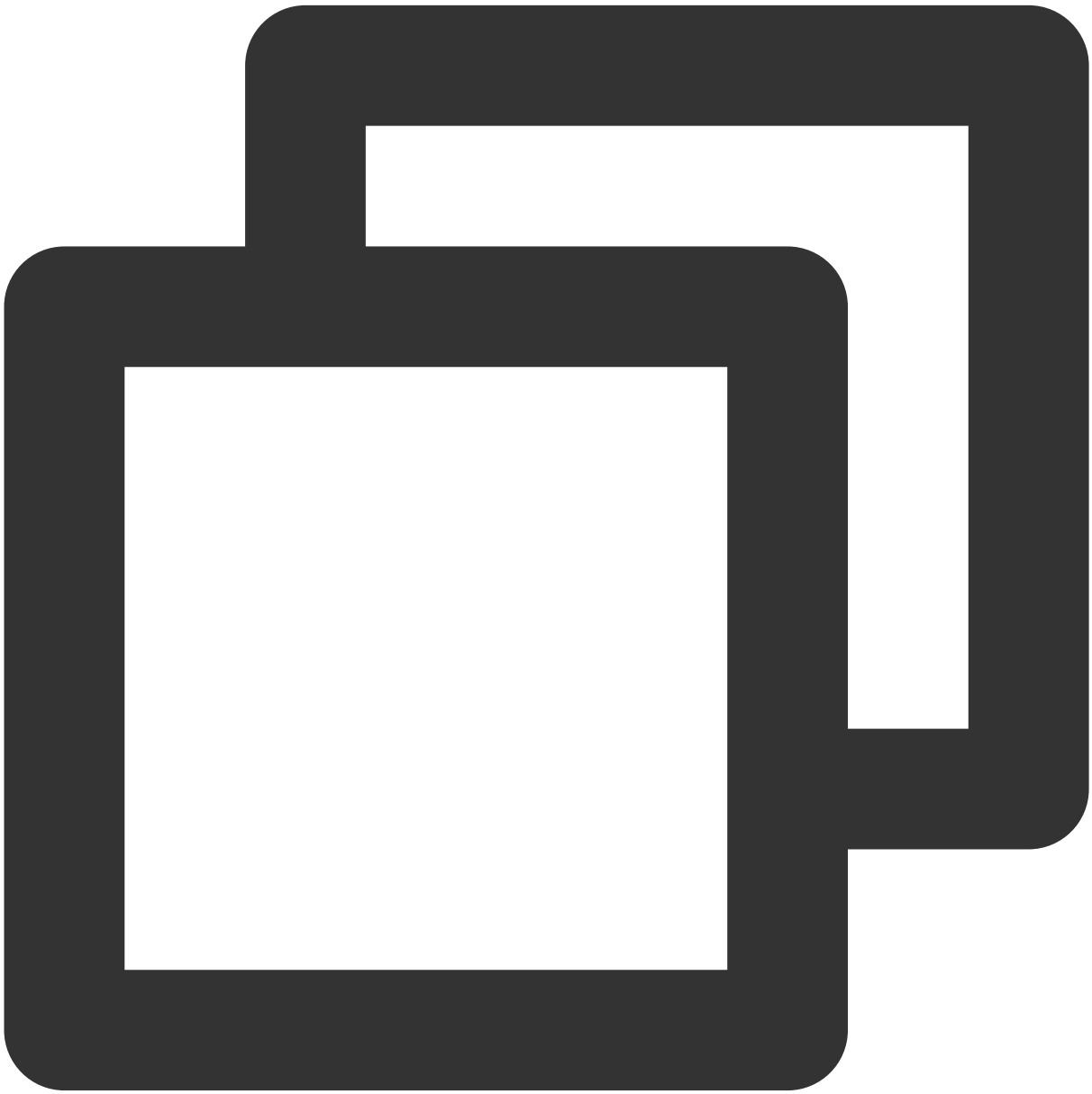
React

Vue



```
import { TUICallKit, VideoDisplayMode, VideoResolution } from "@tencentcloud/call-u

<TUICallKit
    videoDisplayMode={VideoDisplayMode.CONTAIN}
    videoResolution={VideoResolution.RESOLUTION_1080P}
    beforeCalling={handleBeforeCalling}
    afterCalling={handleAfterCalling}
```

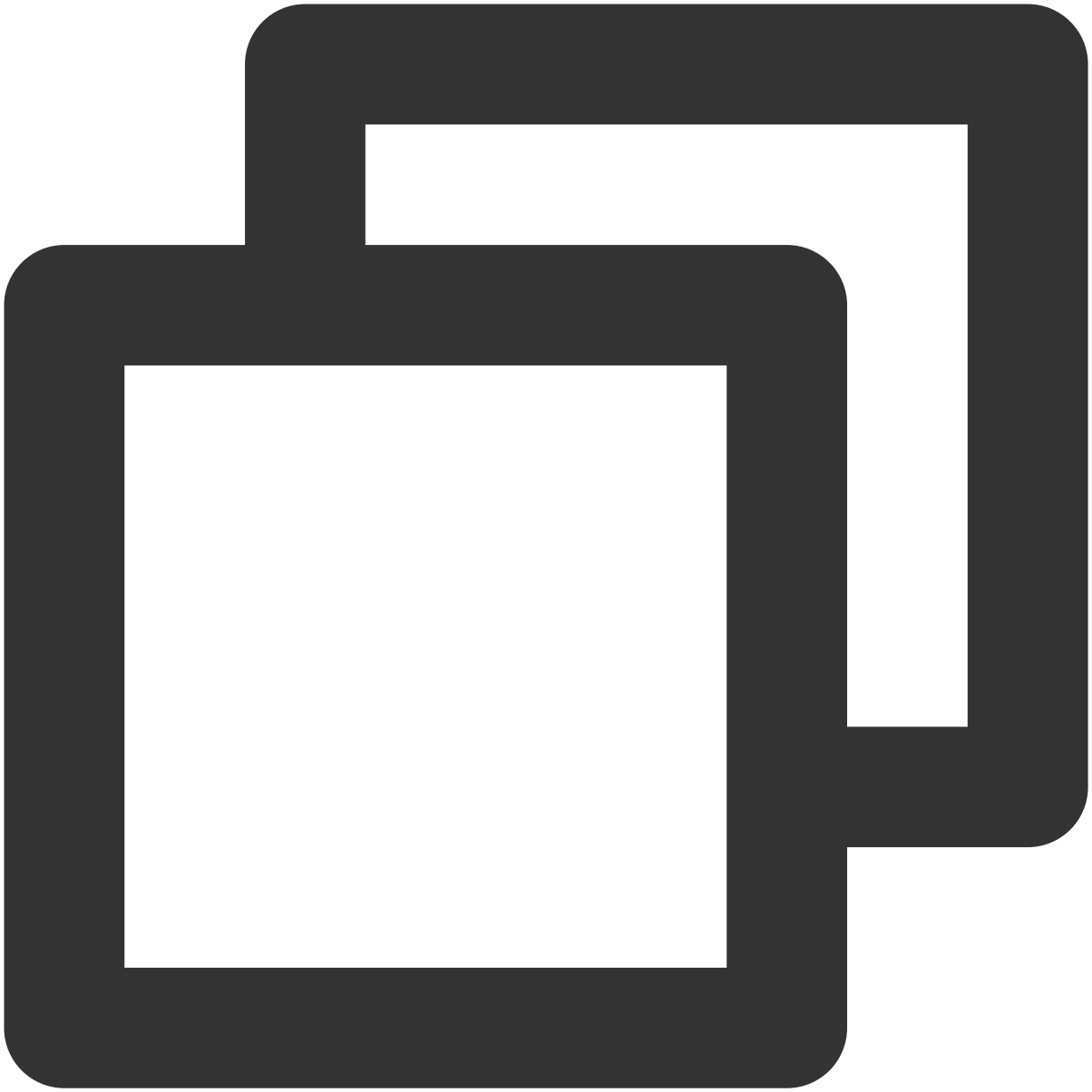
```
/>

function handleBeforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] handleBeforeCalling:", type, error);
}
function handleAfterCalling() {
  console.log("[TUICallkit Demo] handleAfterCalling");
}
```



```
<template>
   <TUICallKit
```

```
      :allowedMinimized="true"
      :allowedFullScreen="true"
      :videoDisplayMode="VideoDisplayMode.CONTAIN"
      :videoResolution="VideoResolution.RESOLUTION_1080P"
      :beforeCalling="beforeCalling"
      :afterCalling="afterCalling"
      :onMinimized="onMinimized"
      :kickedOut="handleKickedOut"
      :statusChanged="handleStatusChanged"
    />
</template>
<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS }

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

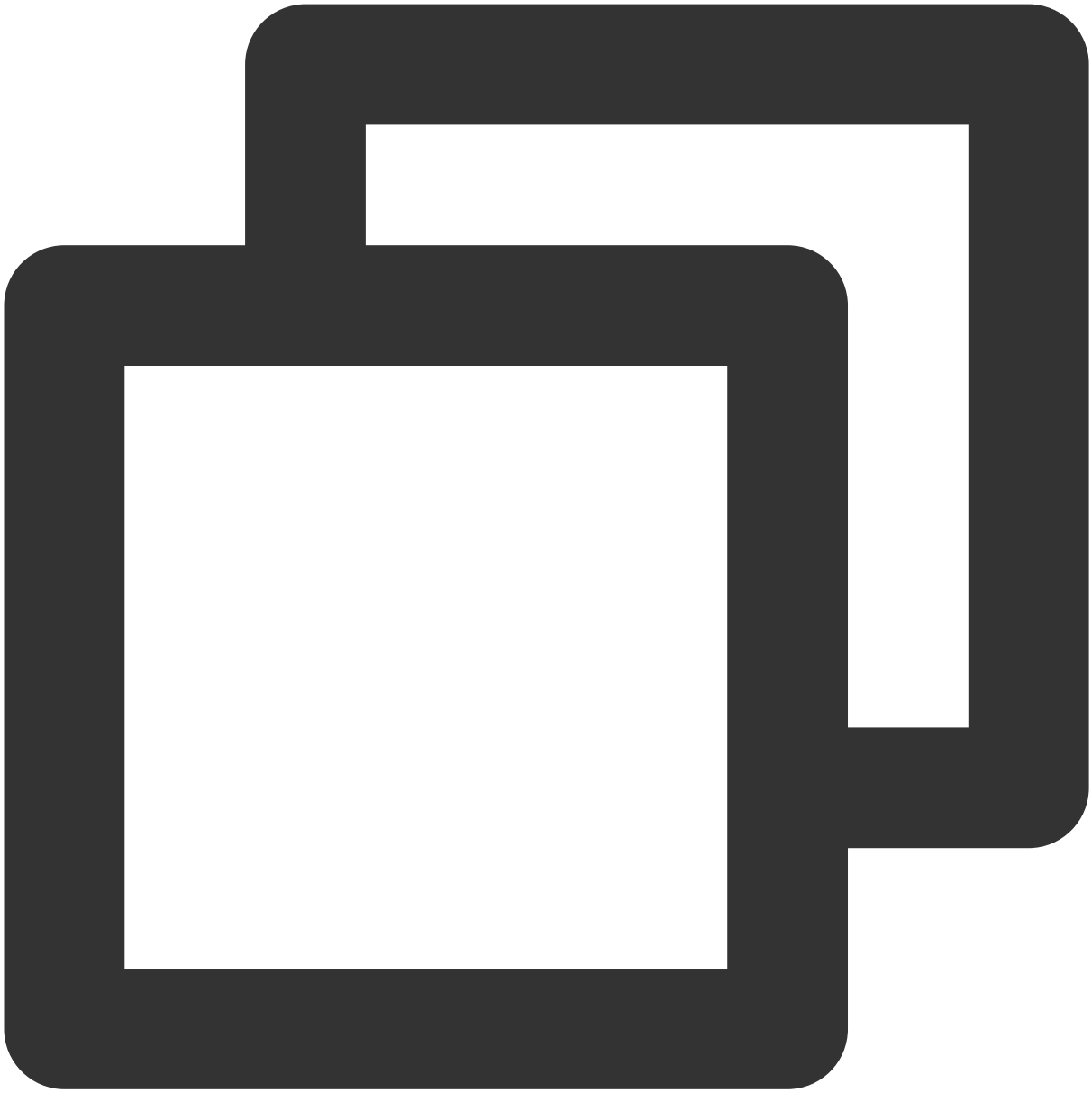
TUICallKitServer API 详情



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using
```

init

初始化 TUICallKit，需在 call, groupCall 之前进行。



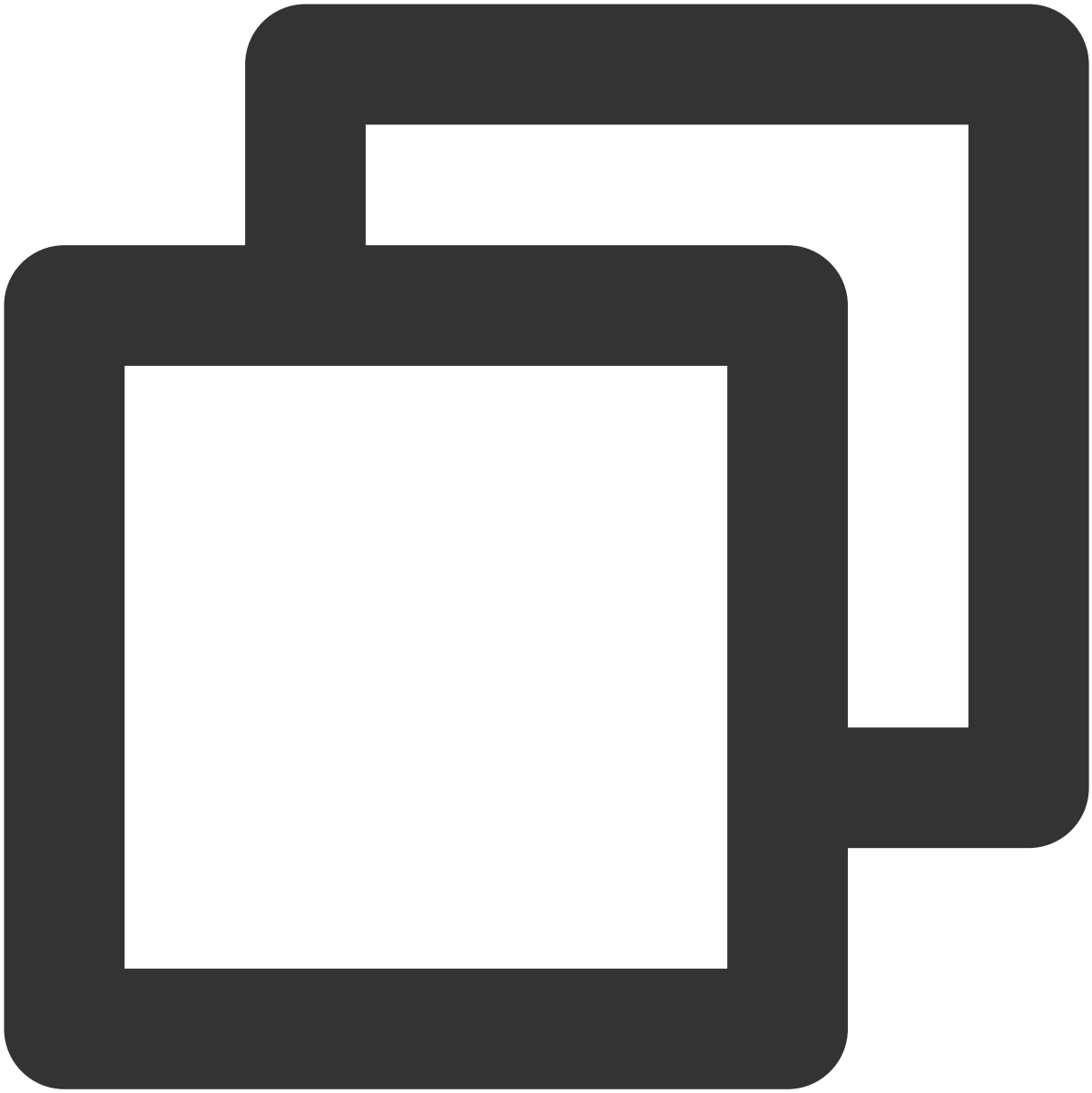
```
try {
  await TUICallKitServer.init({ SDKAppID, userID, userSig });
  // If you already have a tim instance in your project, you need to pass it in her
  // await TUICallKitServer.init({ tim, SDKAppID, userID, userSig});
  alert("[TUICallKit] Initialization succeeds.");
} catch (error: any) {
  alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
SDKAppID	Number	是	云通信应用的 SDKAppID
userID	String	是	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）
userSig	String	是	腾讯云设计的一种安全保护签名，获取方式请参考 如何计算 UserSig
tim	Any	否	tim 参数适用于业务中已存在 TIM 实例，为保证 TIM 实例唯一性

call

拨打 1v1 通话。



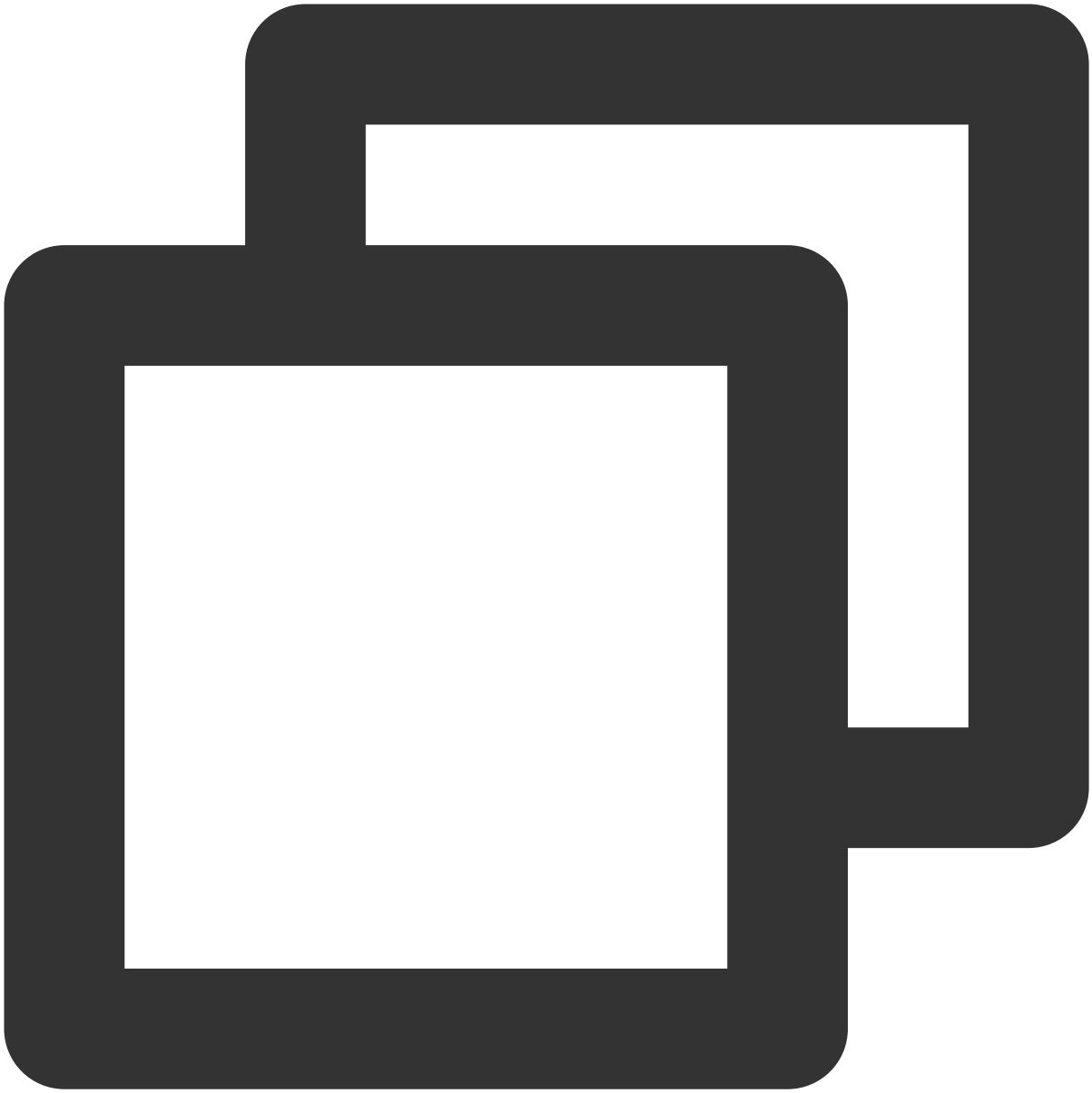
```
try {
  await TUICallKitServer.call({
    userID: callUserID,
    type: TUICallType.VIDEO_CALL,
  });
} catch (error: any) {
  alert(`[TUICallKit] Call failed. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userID	String	是	目标用户的 userId
type	TUICallType	是	通话的媒体类型，参数值说明参考 TUICallType 通话类型
roomID	Number	否	数字房间号，范围 [1, 2147483647]
timeout	Number	否	通话的超时时间，0 为不超时, 单位 s(秒)（选填） - 默认 30s
userData	String	否	扩展字段: 用于在邀请信令中增加扩展信息
offlinePushInfo	Object	否	自定义离线消息推送

groupCall

发起群组通话。



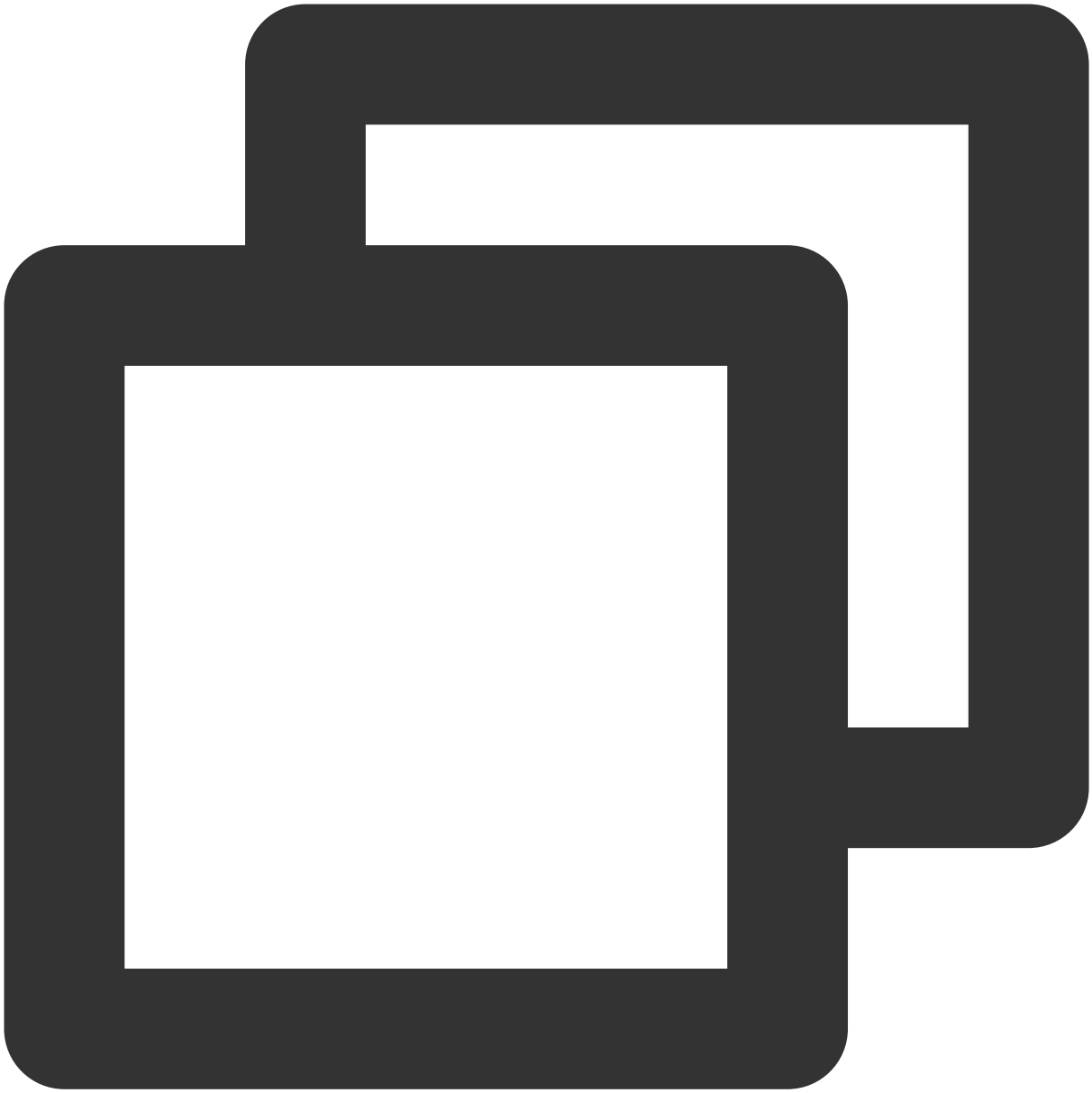
```
try {
  await TUICallKitServer.groupCall({
    userIDList: ['jack', 'tom'],
    groupID: "xxx",
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userIDList	Array<String>	是	邀请列表成员列表
type	TUICallType	是	通话的媒体类型，参数值说明参考TUICallType 通话类型
groupID	String	是	通话群组 ID，groupID 的创建可参考 chat-createGroup API
roomID	Number	否	数字房间号，范围 [1, 2147483647]
timeout	Number	否	通话的超时时间，0 为不超时, 单位 s(秒)（选填） - 默认 30s
userData	String	否	扩展字段：用于在邀请信令中增加扩展信息
offlinePushInfo	Object	否	自定义离线消息推送

setLanguage

设置语言。



```
TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn"
```

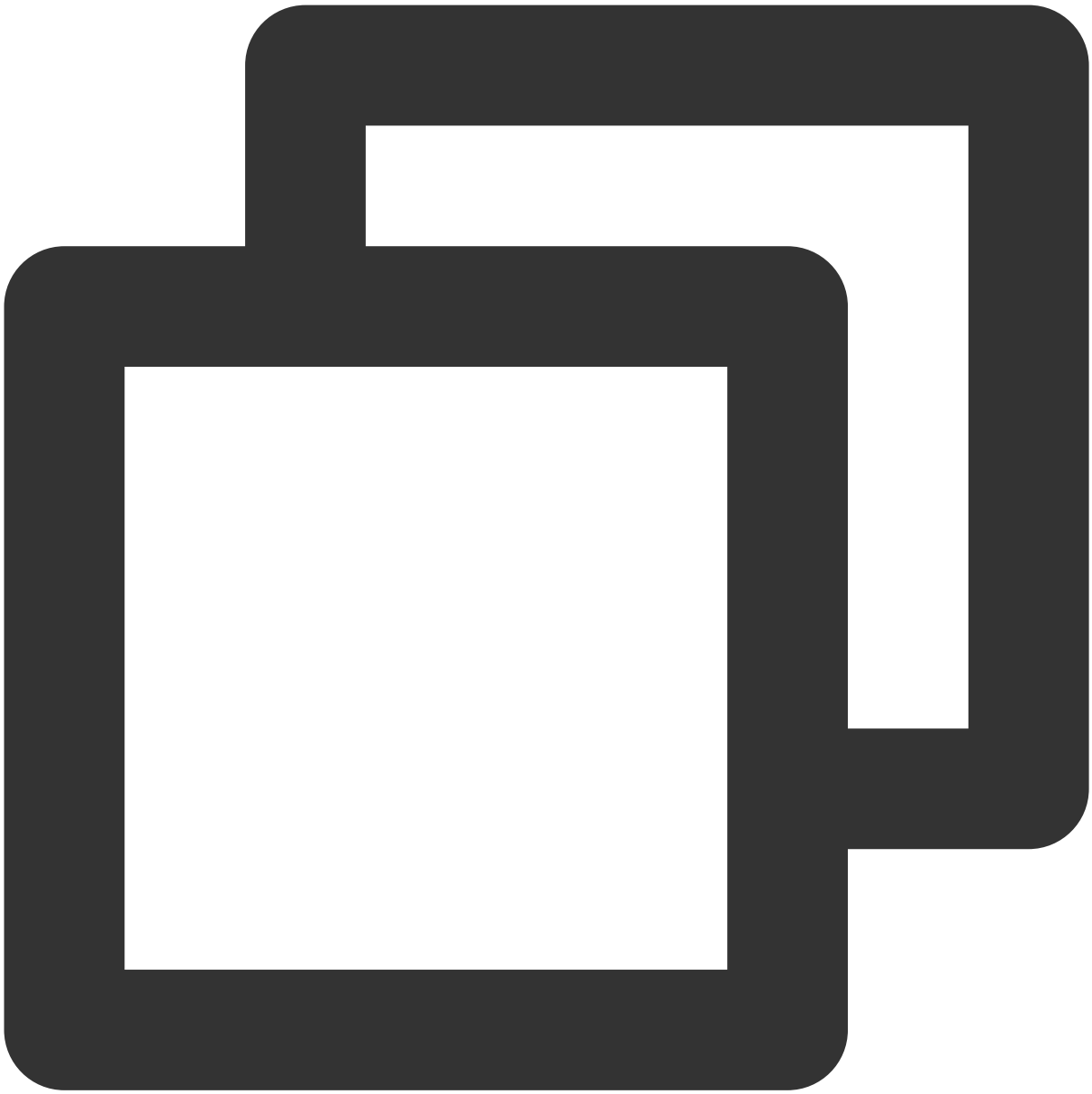
参数如下表所示：

参数	类型	是否必填	含义
lang	String	是	语言类型 `en` 或 `zh-cn`

setSelfInfo

注意：Vue ≥ v2.2.0 支持。通话中使用该接口修改用户信息，UI 不会立即更新，需要等到下次通话才能看到变化。

设置用户昵称和头像。



```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "xxx", avatar: "http://xxx" });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`;
}
```

参数如下表所示：

参数	类型	是否必填	含义

nickName	String	是	用户昵称
avatar	String	是	用户头像地址

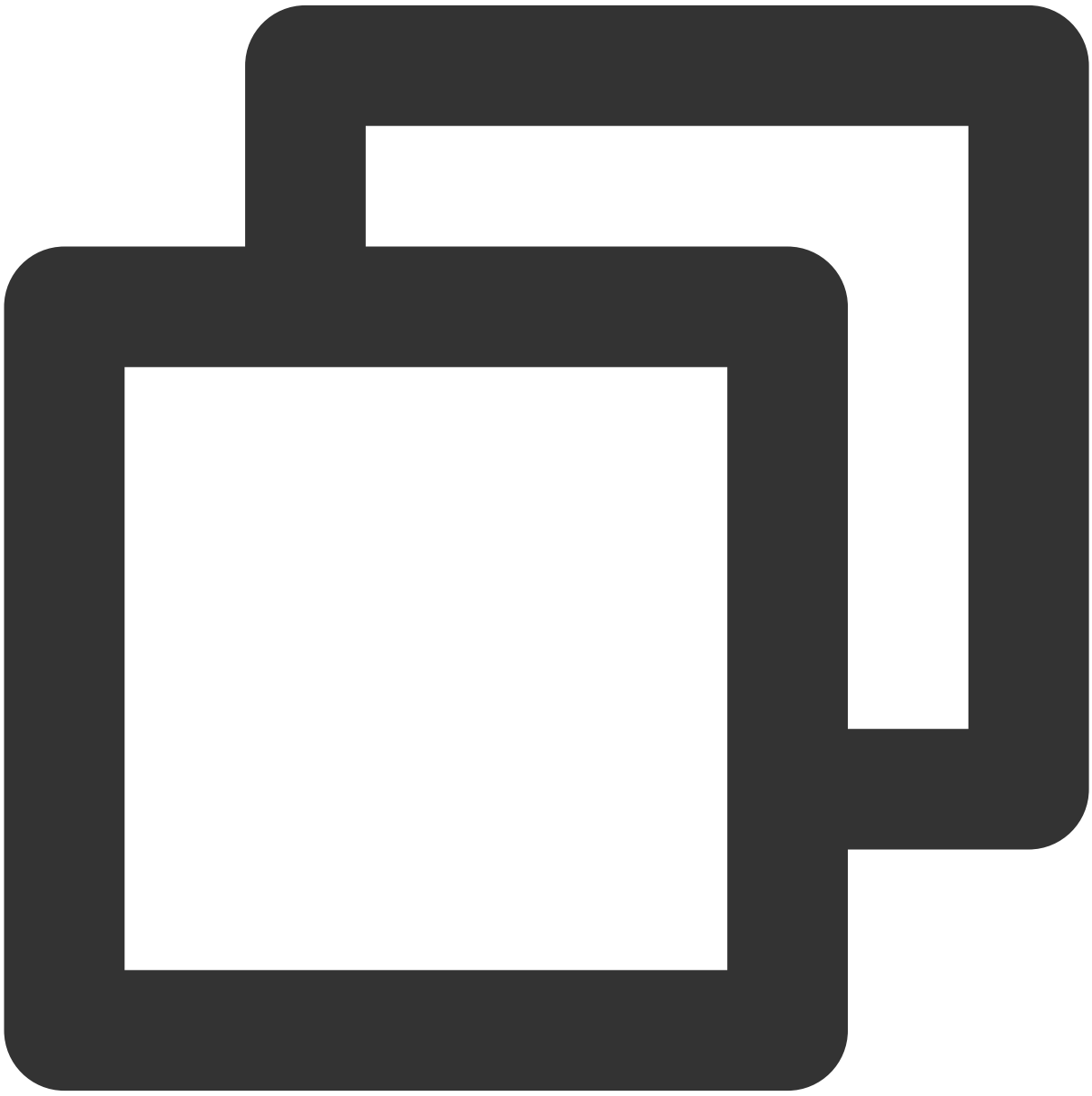
setCallingBell

注意：Vue ≥ v3.0.0 支持

自定义用户的来电铃音。

仅限传入本地 MP3 格式的文件地址，需要确保该文件目录是应用可以访问的。

使用 ES6 import 方式引入铃声文件。



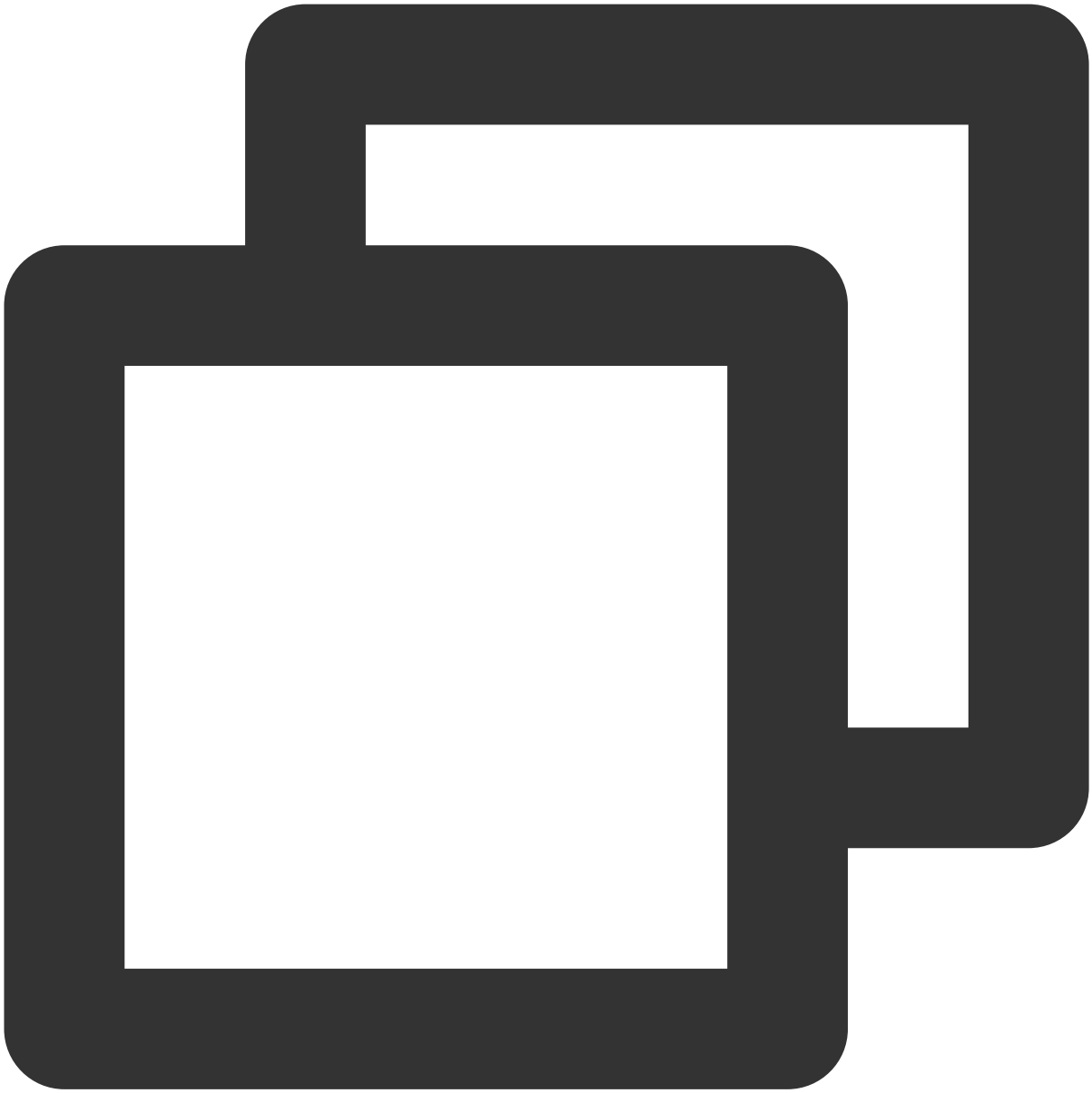
```
try {
  await TUICallKitServer.setCallingBell(filePath?: string)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setCallingBell API. Reason: ${error}`);
}
```

enableFloatWindow

注意：Vue ≥ v3.1.0 支持

开启/关闭悬浮窗功能。

默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。



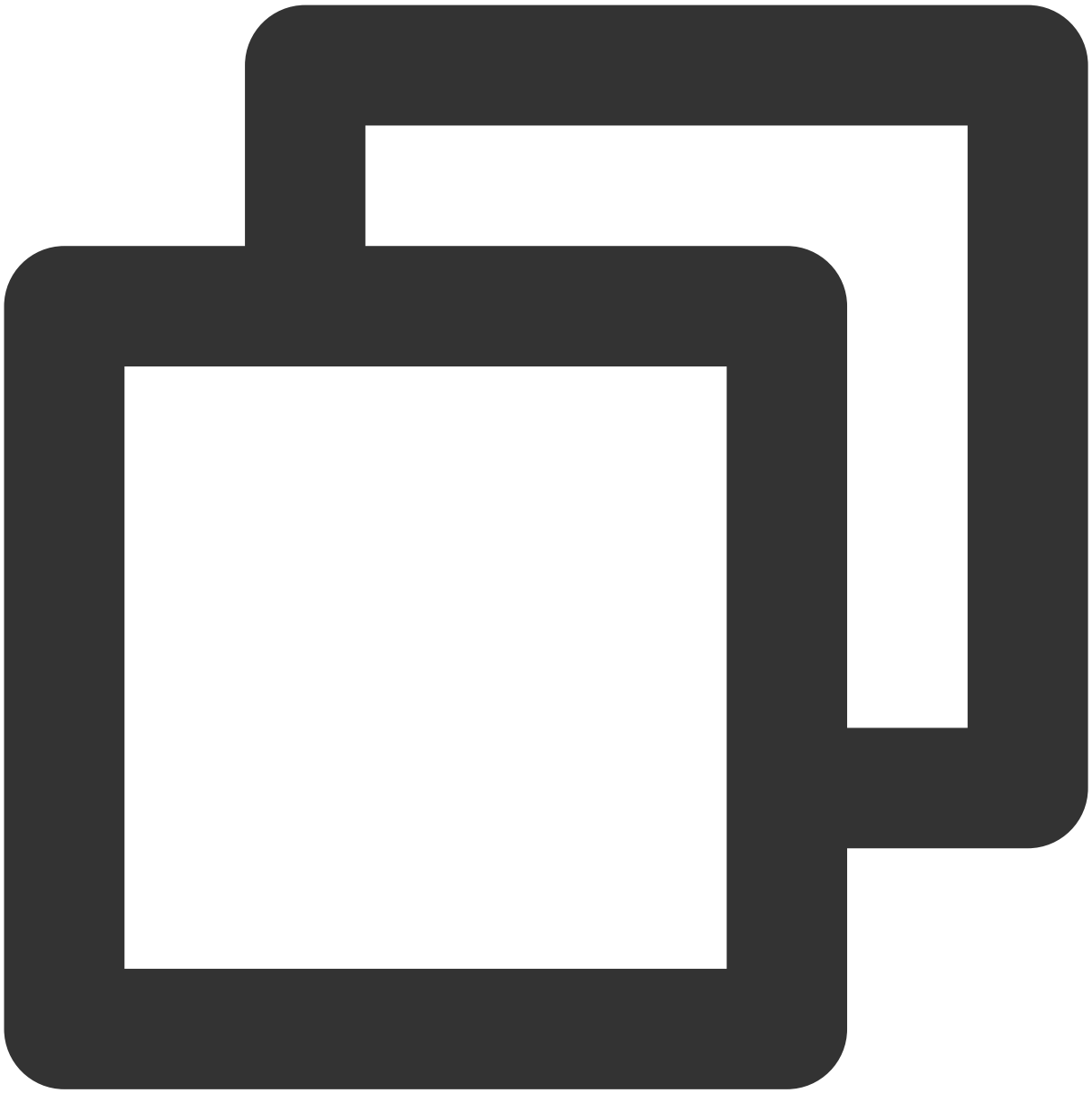
```
try {
  await TUICallKitServer.enableFloatWindow(enable: Boolean)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableFloatWindow API. Reason: ${error}`);
}
```

enableMuteMode

注意：Vue ≥ v3.1.2 支持

开启/关闭来电铃声。

开启后，收到通话请求时，不会播放来电铃声。



```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean)
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

joinInGroupCall

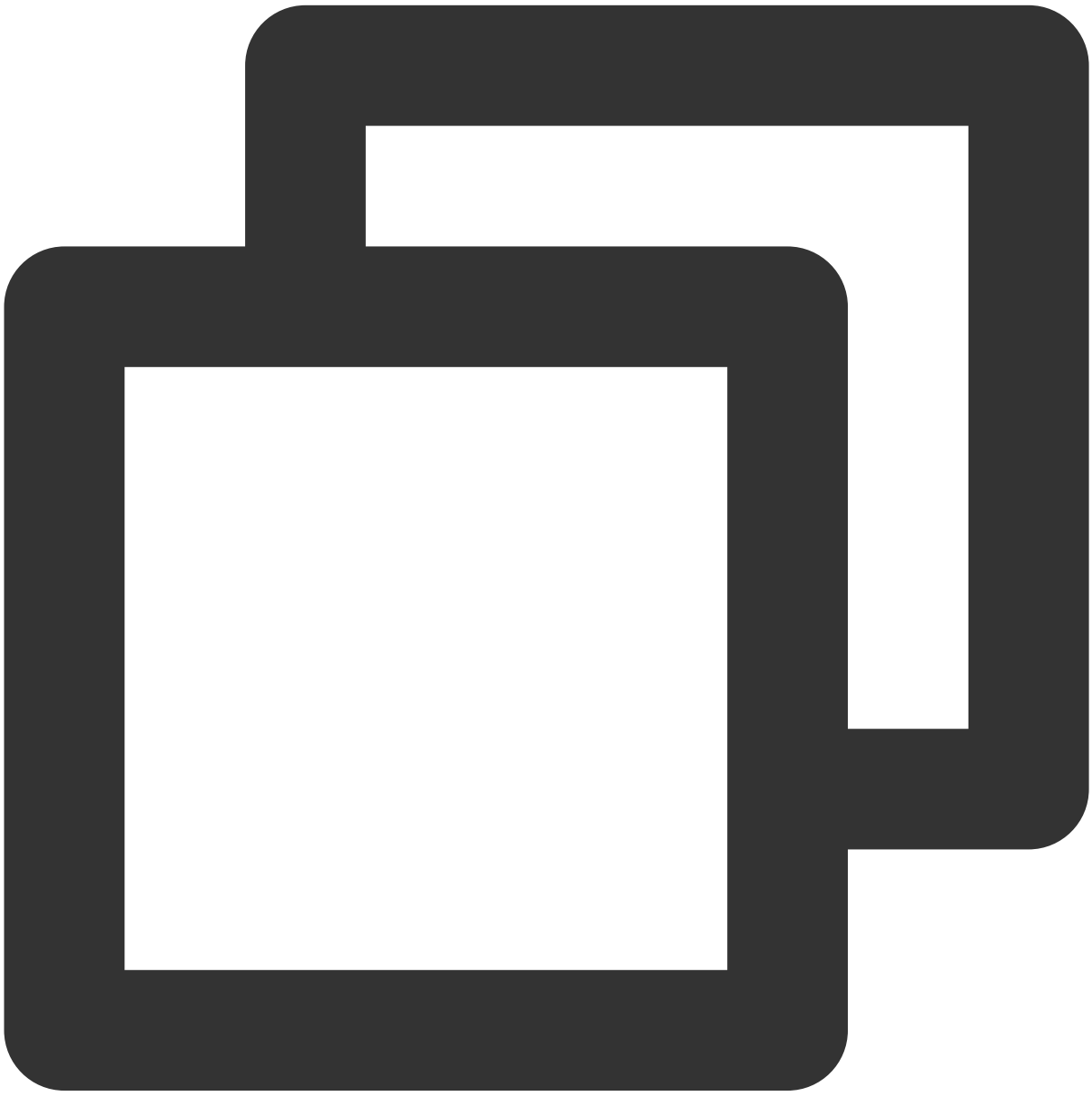
注意：Vue ≥ v3.1.2 支持

加入群组中已有的音视频通话。

说明：

加入群组中已有的音视频通话前，需要提前创建或加入IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。

群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。



```
try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
```

```
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

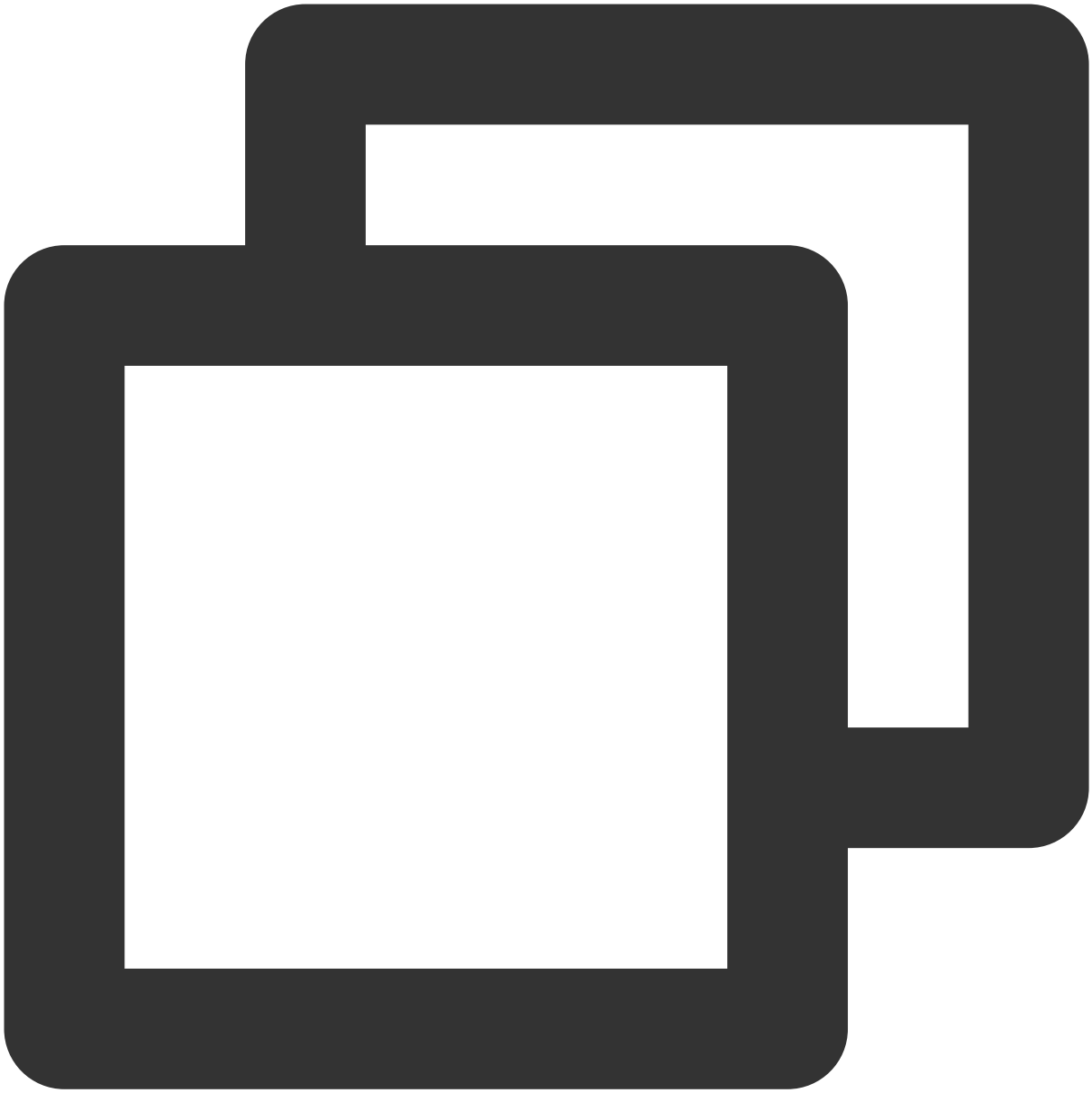
参数列表：

参数	类型	是否必填	含义
type	TUICallType	是	通话的媒体类型，例如视频通话、语音通话
groupID	string	是	此次群组通话的群 ID
roomID	number	是	此次通话的音视频房间 ID

destroyed

销毁 TUICallKit 实例。

该方法不会自动退出 `tim` ，需要手动退出 `tim.logout();` 。



```
try {
  await TUICallKitServer.destroyed();
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the destroyed API. Reason: ${error}`);
}
```

TUICallKit 类型定义

videoDisplayMode

画面显示模式 `videoDisplayMode` 属性有三个值：

`VideoDisplayMode.CONTAIN`

`VideoDisplayMode.COVER`

`VideoDisplayMode.FILL`，默认值是 `VideoDisplayMode.COVER`。

属性	值	描述
videoDisplayMode	VideoDisplayMode.CONTAIN	优先保证视频内容全部显示。 视频尺寸等比缩放，直至视频窗口的一边与视窗边框对齐。 如果视频尺寸与显示视窗尺寸不一致，在保持长宽比的前提下，将视频进行缩放后填满视窗，缩放后的视频四周会有一圈黑边。
	VideoDisplayMode.COVER	优先保证视窗被填满。 视频尺寸等比缩放，直至整个视窗被视频填满。 如果视频长宽与显示窗口不同，则视频流会按照显示视窗的比例进行周边裁剪或图像拉伸后填满视窗。
	VideoDisplayMode.FILL	保证视窗被填满的同时保证视频内容全部显示，但是不保证视频尺寸比例不变。 视频的宽高会被拉伸至和视窗尺寸一致。

videoResolution

分辨率 `videoResolution` 有三个值：

`VideoResolution.RESOLUTION_480P`

`VideoResolution.RESOLUTION_720P`

`VideoResolution.RESOLUTION_1080P`，默认值是 `VideoResolution.RESOLUTION_480P`。

分辨率说明：

视频 Profile	分辨率（宽 × 高）	帧率（fps）	码率（kbps）
480p	640 × 480	15	900
720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

常见问题：

iOS 13&14不支持编码高于 720P 的视频，建议在这两个系统版本限制最高采集 720P。参考 iOS Safari 已知问题 case 12。

Firefox 不支持自定义视频帧率（默认为 30fps）。
受系统性能占用，摄像头采集能力和浏览器限制等因素的影响，视频分辨率、帧率、码率的实际值不一定能够完全匹配设定值，在这种情况下，浏览器会自动调整 Profile 尽可能匹配设定值。

STATUS

STATUS 属性值	描述
STATUS.IDLE	闲置状态
STATUS.BE_INVITED	收到通话邀请
STATUS.DIALING_C2C	正在 1v1 呼叫
STATUS.DIALING_GROUP	正在群组呼叫
STATUS.CALLING_C2C_AUDIO	正在 1v1 语音通话
STATUS.CALLING_C2C_VIDEO	正在 1v1 视频通话
STATUS.CALLING_GROUP_AUDIO	正在群组语音通话
STATUS.CALLING_GROUP_VIDEO	正在群组视频通话

TUICallType

TUICallType 类型	描述
TUICallType.AUDIO_CALL	语音通话
TUICallType.VIDEO_CALL	视频通话

offlinePushInfo

参数	类型	是否必填	含义
offlinePushInfo.title	String	否	离线推送标题（选填）
offlinePushInfo.description	String	否	离线推送内容（选填）
offlinePushInfo.androidOPPOChannelID	String	否	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID（选填）
offlinePushInfo.extension	String	否	离线推送透传内容（选填）（tsignaling 版本 >= 0.9.0）

版权所有：腾讯云计算（北京）有限责任公司

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2024-04-03 17:23:11

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
on	监听事件
off	取消监听事件
login	登录接口
logout	登出接口
setSelfInfo	设置用户昵称和头像
call	C2C邀请通话
groupCall	群聊邀请通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	当前通话类型切换

版权所有：腾讯云计算（北京）有限责任公司

startRemoteView	启动远端画面渲染
stopRemoteView	停止远端画面渲染
startLocalView	启动本地画面渲染，注意：后续将废弃，建议使用 openCamera
stopLocalView	停止本地画面渲染，注意：后续将废弃，建议使用 closeCamera
openCamera	开启摄像头，启动本地画面渲染
closeCamera	关闭摄像头，停止本地画面渲染
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
setVideoQuality	设置视频质量
setVideoRenderParams	设置用户视频画面的显示模式
getDeviceList	获取设备列表
switchDevice	切换摄像头或麦克风设备
enableAIVoice	开启/关闭 AI 降噪
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （**尊享版套餐支持**）

版权所有：腾讯云计算（北京）有限责任公司

TUICallEvent

最近更新时间：2024-04-03 17:23:11

TUICallEvent API 简介

TUICallEvent API 是音视频通话组件的**事件接口**。

事件列表

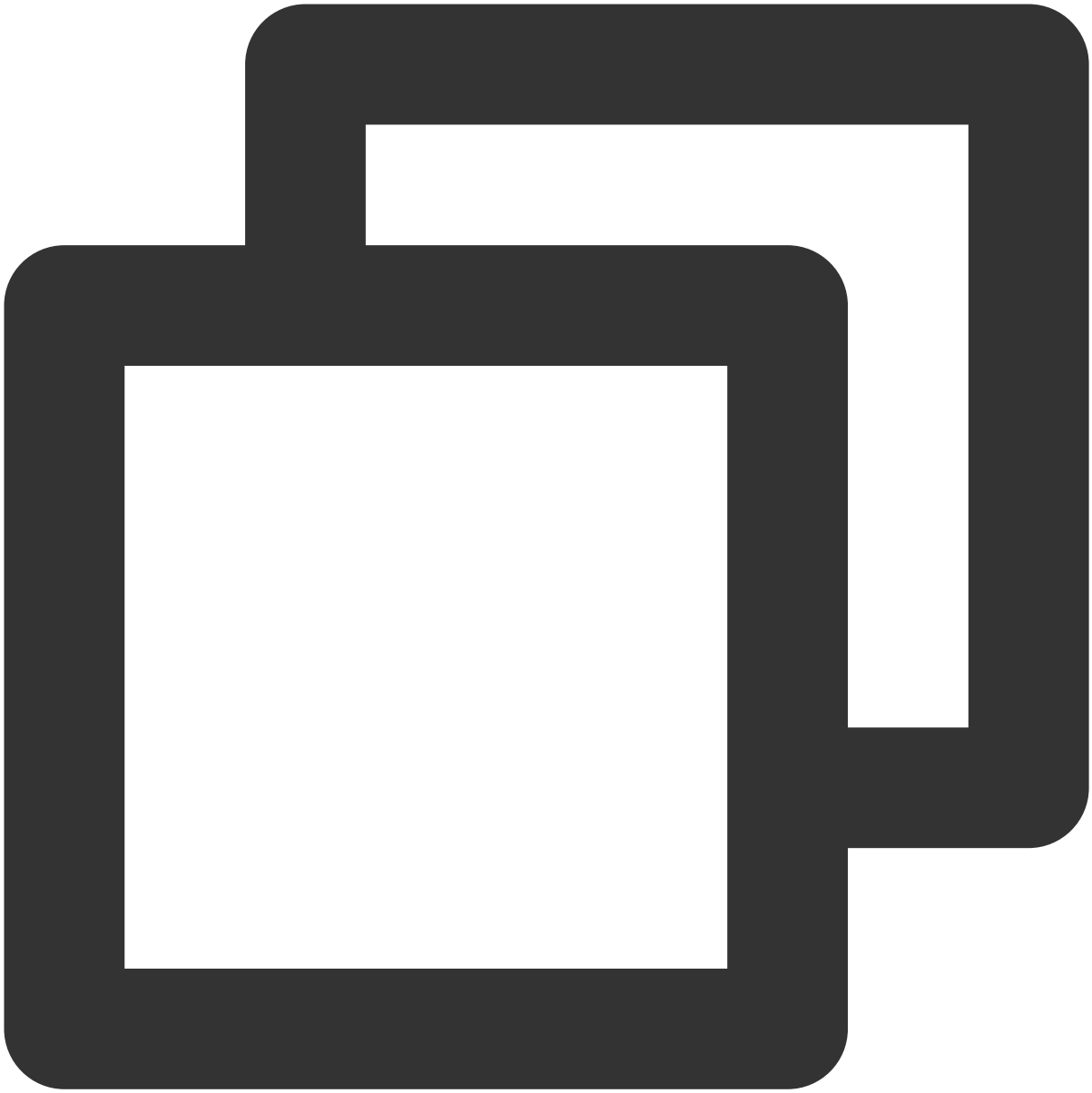
EVENT	描述
TUICallEvent.ERROR	SDK 内部发生了错误
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该事件
TUICallEvent.KICKED_OUT	重复登录，收到该事件说明被移出房间
TUICallEvent.USER_ACCEPT	如果有用户接听，那么会收到此事件
TUICallEvent.USER_ENTER	如果有用户同意进入通话，那么会收到此事件
TUICallEvent.USER_LEAVE	如果有用户同意离开通话，那么会收到此事件
TUICallEvent.REJECT	用户拒绝通话，会收到该事件
TUICallEvent.NO_RESP	邀请用户无应答，会收到该事件
TUICallEvent.LINE_BUSY	邀请方忙线，会收到该事件
TUICallEvent.USER_VIDEO_AVAILABLE	远端用户开启/关闭了摄像头, 会收到该事件
TUICallEvent.USER_AUDIO_AVAILABLE	远端用户开启/关闭了麦克风, 会收到该事件
TUICallEvent.USER_VOICE_VOLUME	远端用户说话音量调整, 会收到该事件
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该事件
TUICallEvent.INVITED	被邀请进行通话，会收到该事件
TUICallEvent.ON_CALL_RECEIVED	通话请求的事件

版权所有：腾讯云计算（北京）有限责任公司

TUICallEvent.CALLING_CANCEL	通话未建立时，通话的各端都会抛出该事件
TUICallEvent.ON_CALL_BEGIN	通话接通时抛出的事件
TUICallEvent.CALLING_END	收到该事件说明本次通话结束了
TUICallEvent.DEVICED_UPDATED	设备列表更新收到该事件
TUICallEvent.CALL_TYPE_CHANGED	通话类型切换收到该事件

ERROR

SDK 内部发生了错误，可以通过监听该事件，捕获底层 SDK 的内部错误。



```
let onError = function(error) {
  console.log(error.code, error.msg);
};
tuiCallEngine.on(TUICallEvent.ERROR, onError);
```

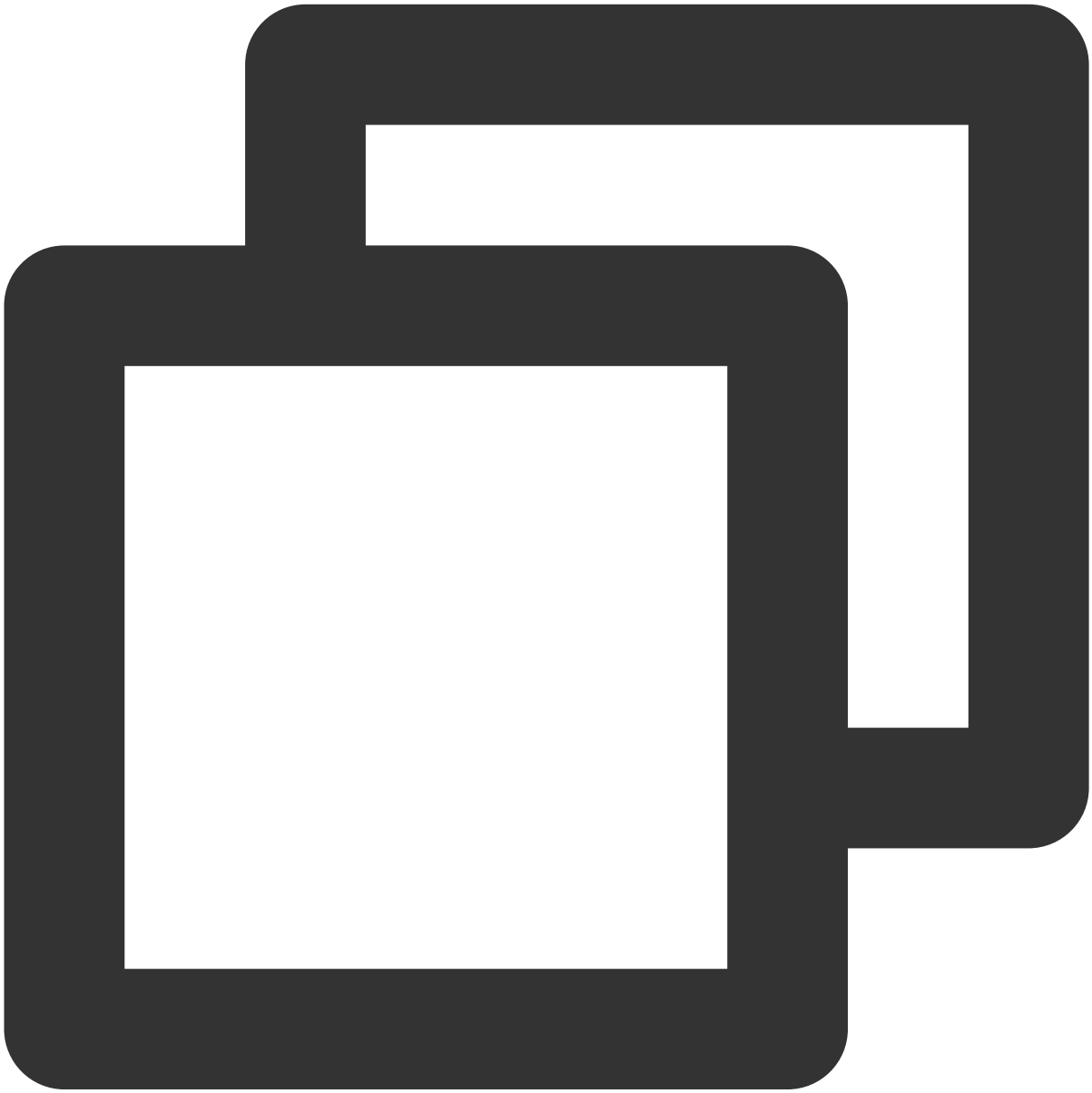
参数如下表所示：

参数	类型	含义
code	int	错误码

msg	String	错误信息

SDK_READY

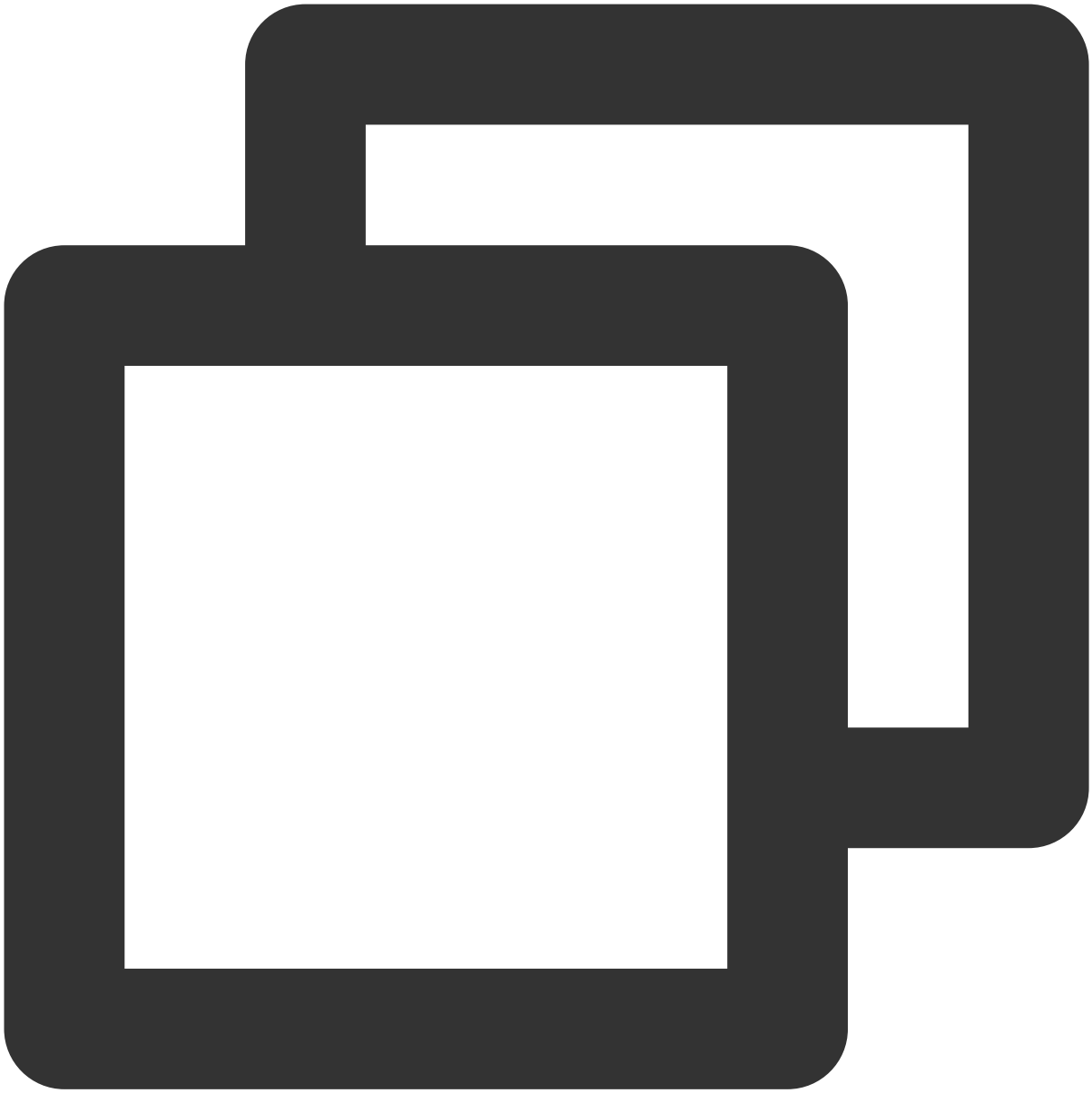
SDK 进入 ready 状态时收到该事件。



```
let onSDKReady = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.SDK_READY, onSDKReady);
```

KICKED_OUT

重复登录，收到该事件说明被踢出房间。



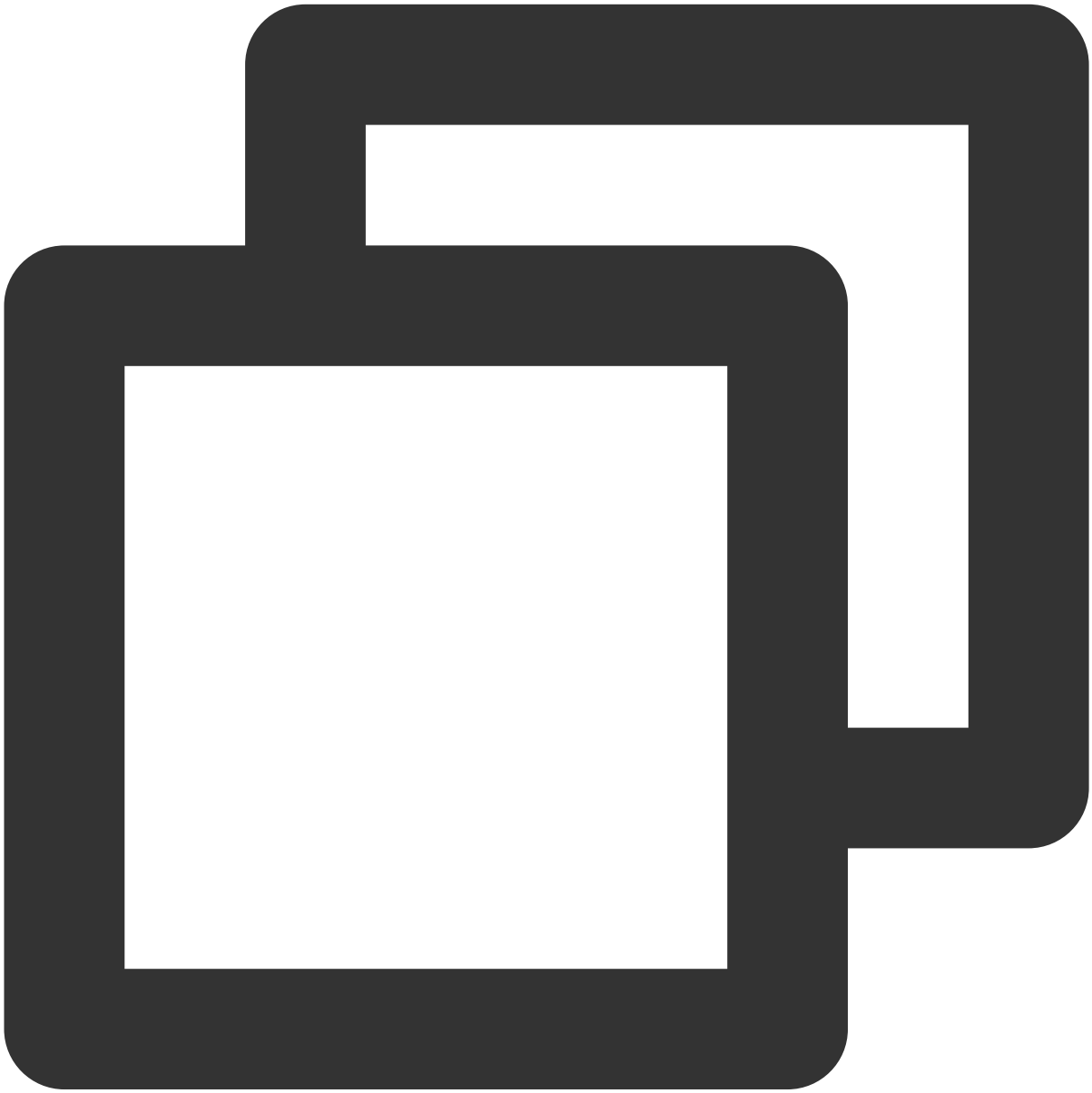
```
let handleOnKickedOut = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.KICKED_OUT, handleOnKickedOut);
```

USER_ACCEPT

如果有用户接听，其他用户都会收到该事件，userID 就是接听的用户。

1. 1v1 通话下：被叫接通，主叫会抛出该事件。

2. 群组通话下：A 呼叫 B、C，B 接通，A、C 都会抛出该事件，事件的 userID 都是 B。同理 C 接通，A、B 都会抛出该事件，事件的 userID 都是 C。



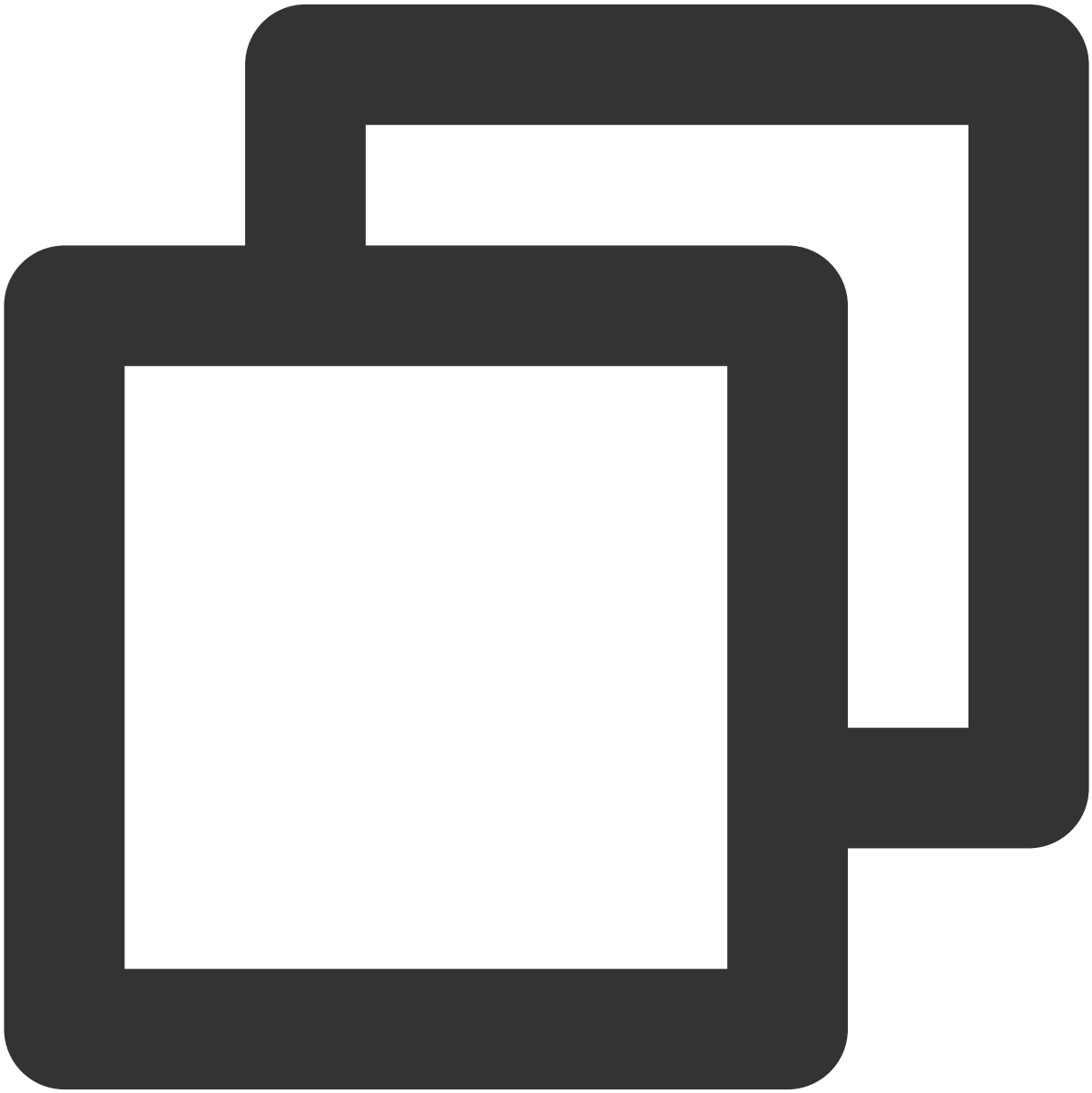
```
let handleUserAccept = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, handleUserAccept);
```

参数如下表所示：

参数	类型	含义
userID	String	接听的用户 ID

USER_ENTER

如果有用户同意进入通话，其他用户都会收到该事件，userID 就是同意进入通话的用户。



```
let handleUserEnter = function(event) {
  console.log(event.userID);
};
```

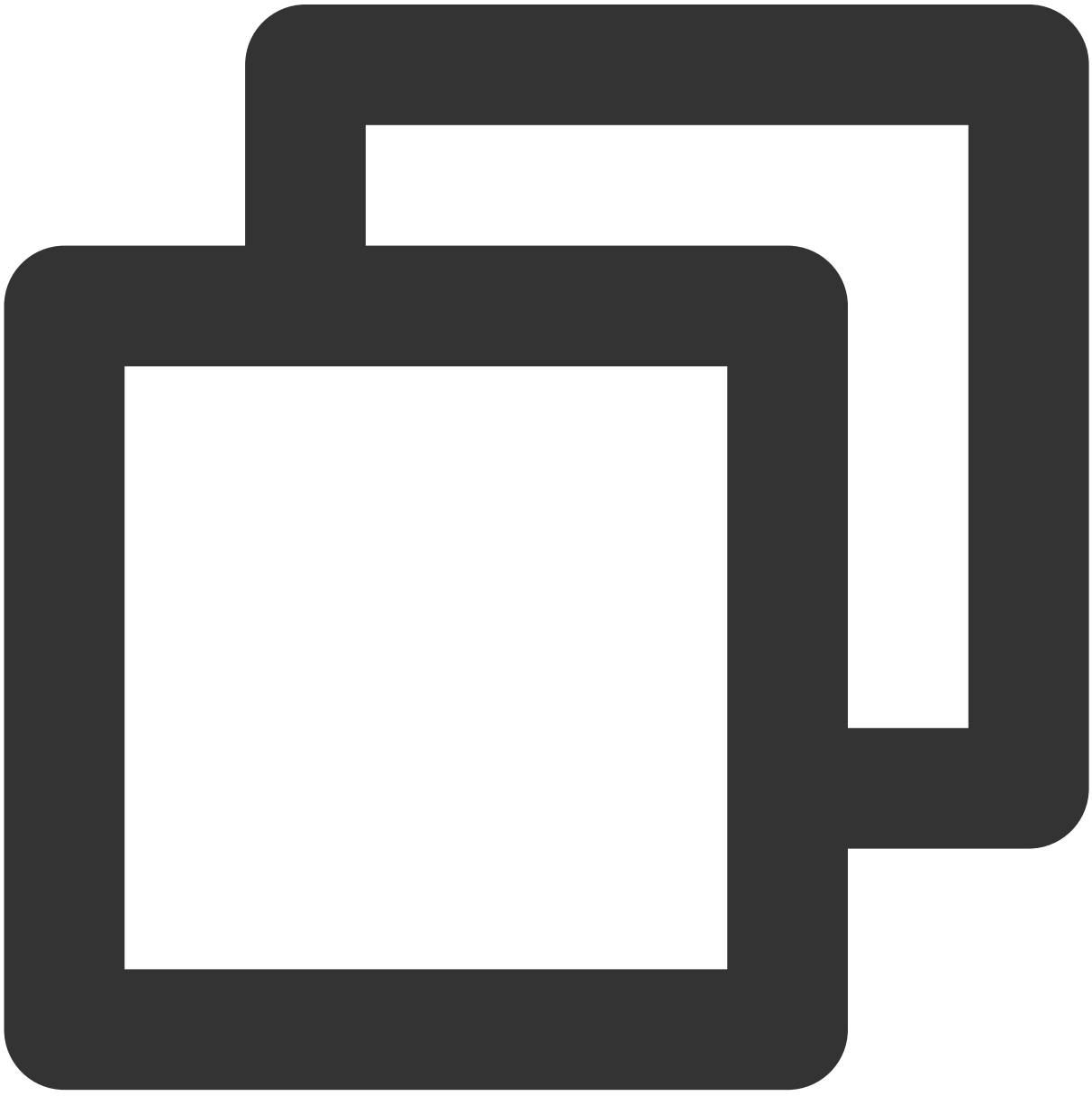
```
tuiCallEngine.on(TUICallEvent.USER_ENTER, handleUserEnter);
```

参数如下表所示：

参数	类型	含义
userID	String	进房的用户 ID

USER_LEAVE

有用户离开此次通话时，**通话中**的其他用户都会收到该事件，userID 就是离开通话的用户。



```
let handleUserLeave = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_LEAVE, handleUserLeave);
```

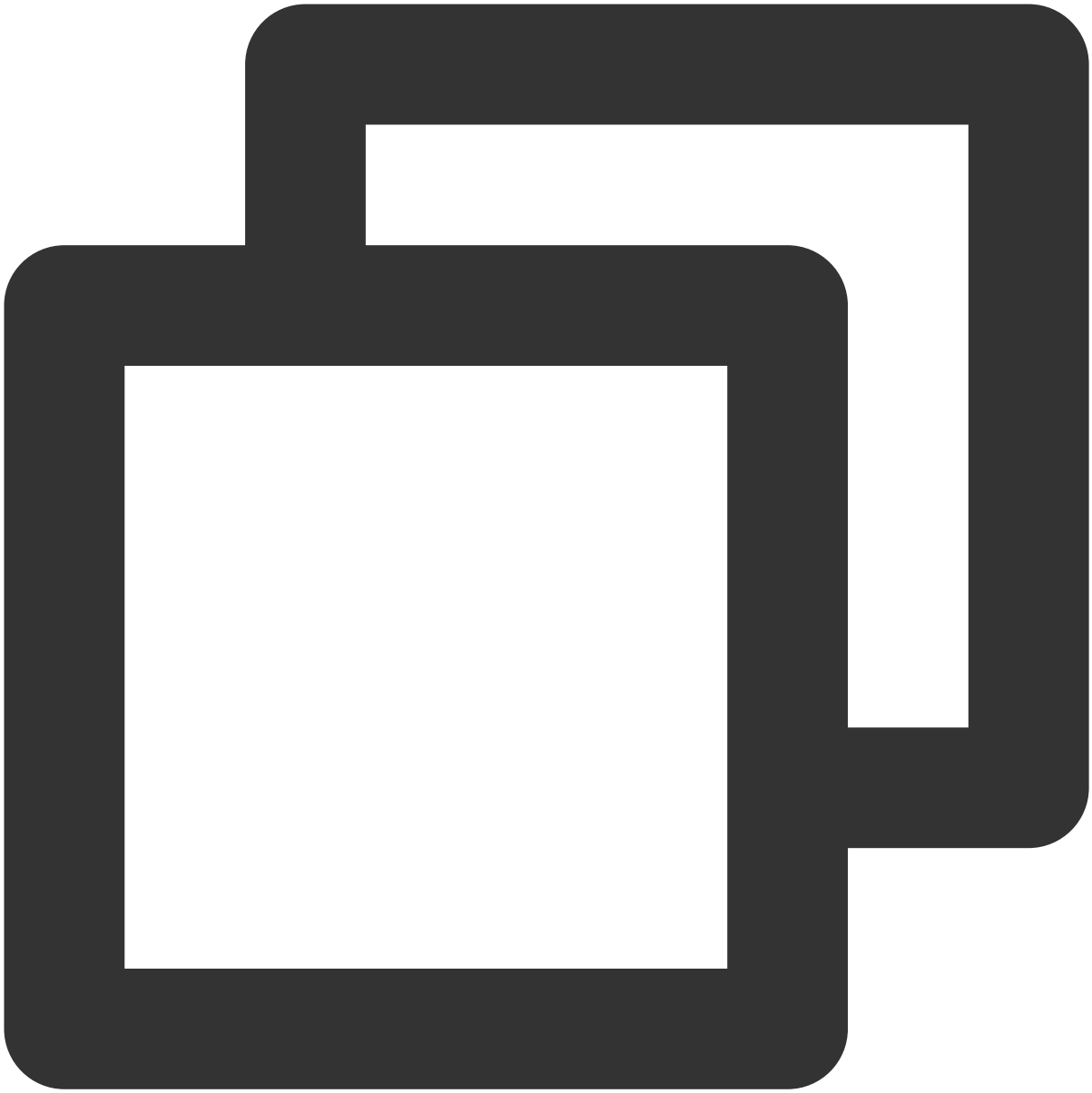
参数如下表所示：

参数	类型	含义
userID	String	退房的用户 ID

REJECT

通话被拒绝事件

1. 在 1v1 通话中，只有主叫方会收到拒绝事件，userID 就是被叫用户。
2. 在群组通话中，所有被邀请者都可以收到该事件，userID 就是拒绝通话的用户。



```
let handleInviteeReject = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.REJECT, handleInviteeReject);
```

参数如下表所示：

参数	类型	含义
userID	String	拒绝通话的用户 ID

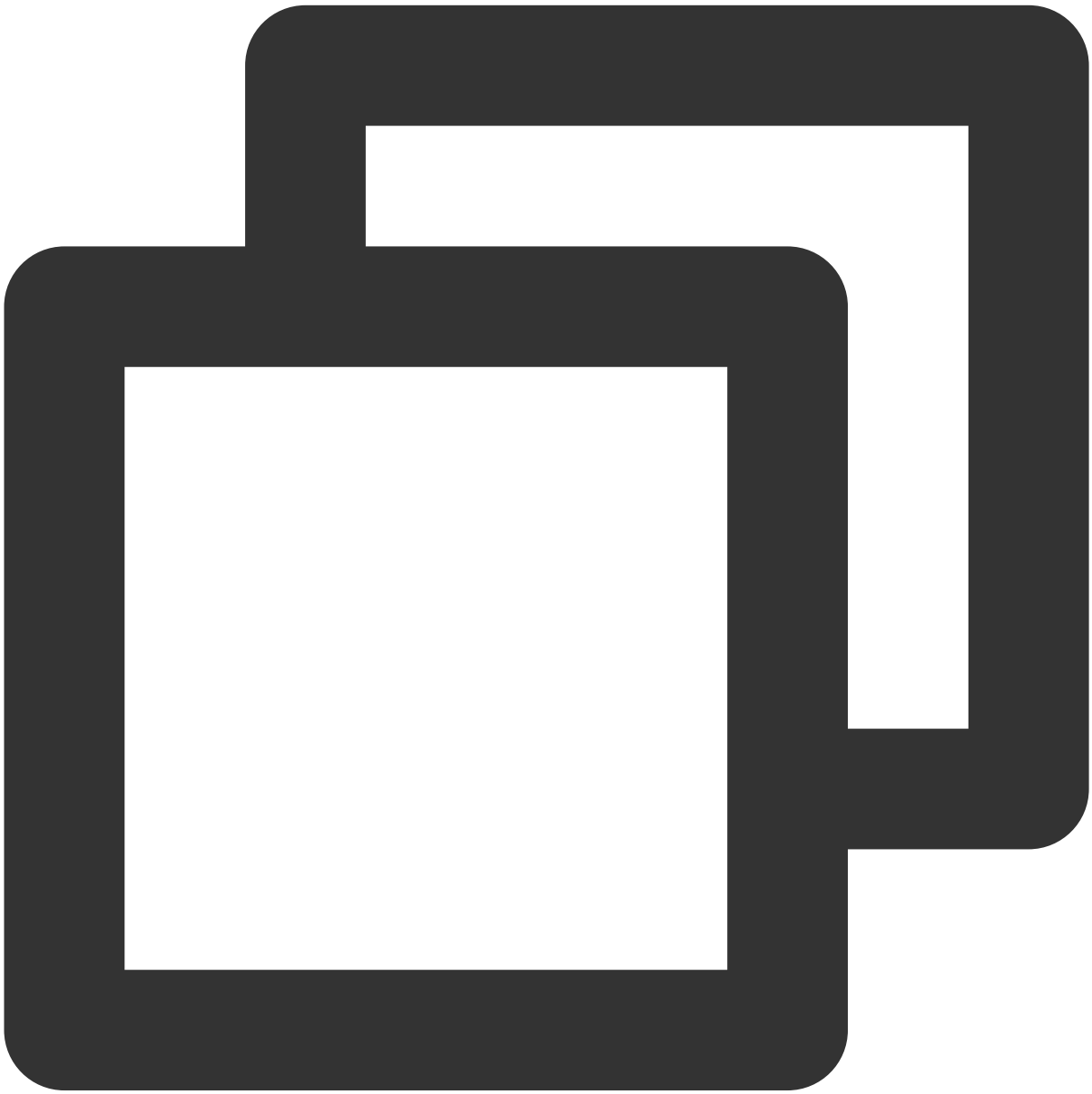
版权所有：腾讯云计算（北京）有限责任公司

NO_RESP

邀请用户无应答事件。

在 1v1 通话中，只有发起方会收到无人应答的事件。例如 A 邀请 B，B 不应答，A 可以收到该事件。

在群组通话中，所有被邀请人均能收到该事件。例如 A 邀请 B、C 进入通话，B 不应答，A、C 均能收到该事件。



```
let handleNoResponse = function(event) {
console.log(event.sponsor, event.userIDList);
};
tuiCallEngine.on(TUICallEvent.NO_RESP, handleNoResponse);
```

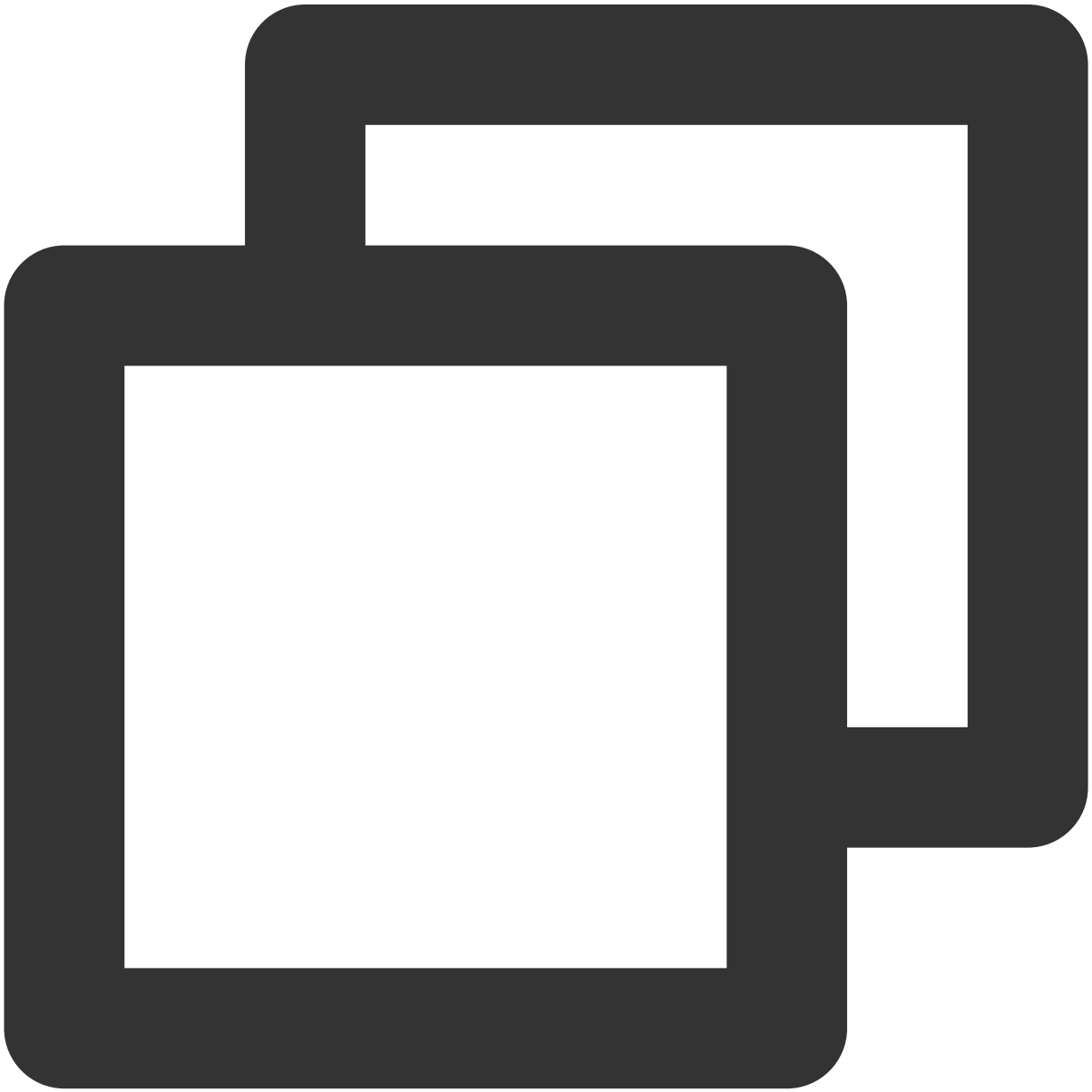
版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
sponsor	String	发起方的用户 ID
userIDList	Array<String>	触发超时无响应的用户列表

LINE_BUSY

通话忙线事件。



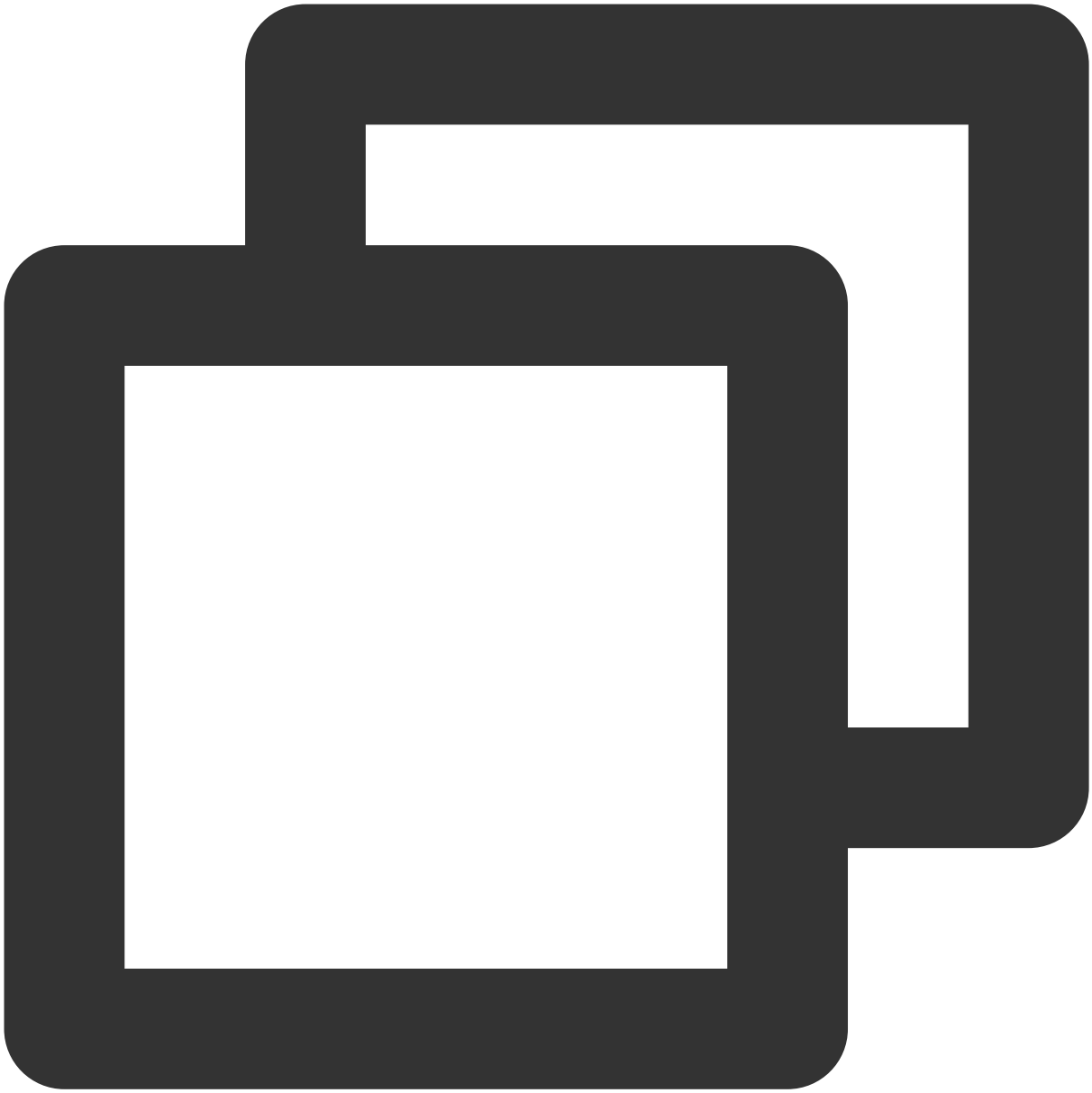
```
let handleLineBusy = function(event) {
  console.log(event)
};
tuiCallEngine.on(TUICallEvent.LINE_BUSY, handleLineBusy);
```

参数如下表所示：

参数	类型	含义
userID	String	忙线用户 ID

USER_VIDEO_AVAILABLE

远端用户开启/关闭了摄像头, 会收到该事件。



```
let handleUserVideoChange = function(event) {
  console.log(event.userID, event.isVideoAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

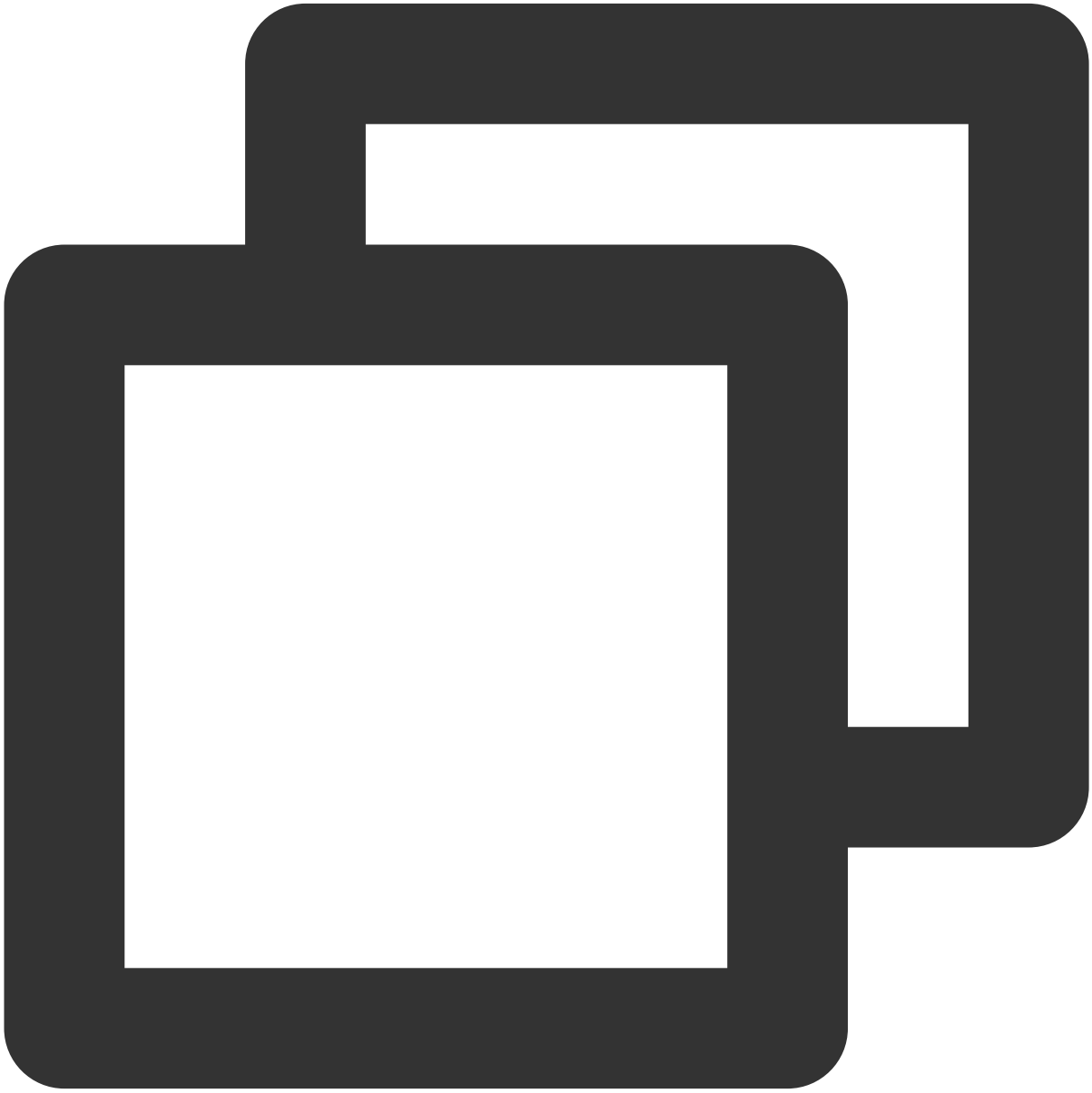
参数如下表所示：

参数	类型	含义
userID	String	远端用户 ID

isVideoAvailable	Boolean	true 远端用户打开摄像头；false 远端用户关闭摄像头

USER_AUDIO_AVAILABLE

远端用户开启/关闭了麦克风, 会收到该事件。



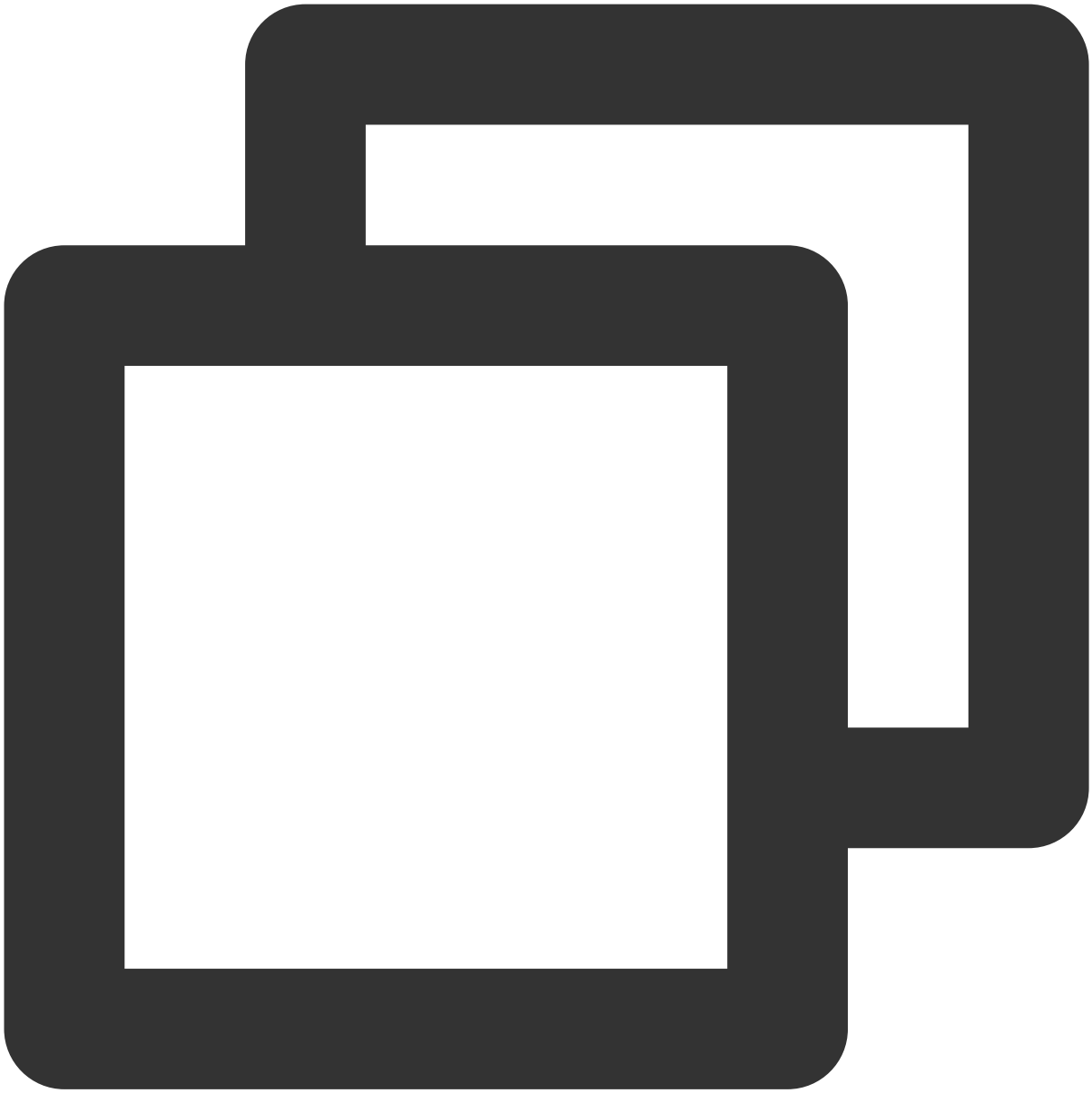
```
let handleUserAudioChange = function(event) {
  console.log(event.userID, event.isAudioAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

参数如下表所示：

参数	类型	含义
userID	String	远端用户 ID
isAudioAvailable	Boolean	true 远端用户打开麦克风；false 远端用户关闭麦克风

USER_VOICE_VOLUME

远端用户说话音量调整, 会收到该事件。



```
let handleUserVoiceVolumeChange = function(event) {
  console.log(event.volumeMap);
};
tuiCallEngine.on(TUICallEvent.USER_VOICE_VOLUME, handleUserVoiceVolumeChange);
```

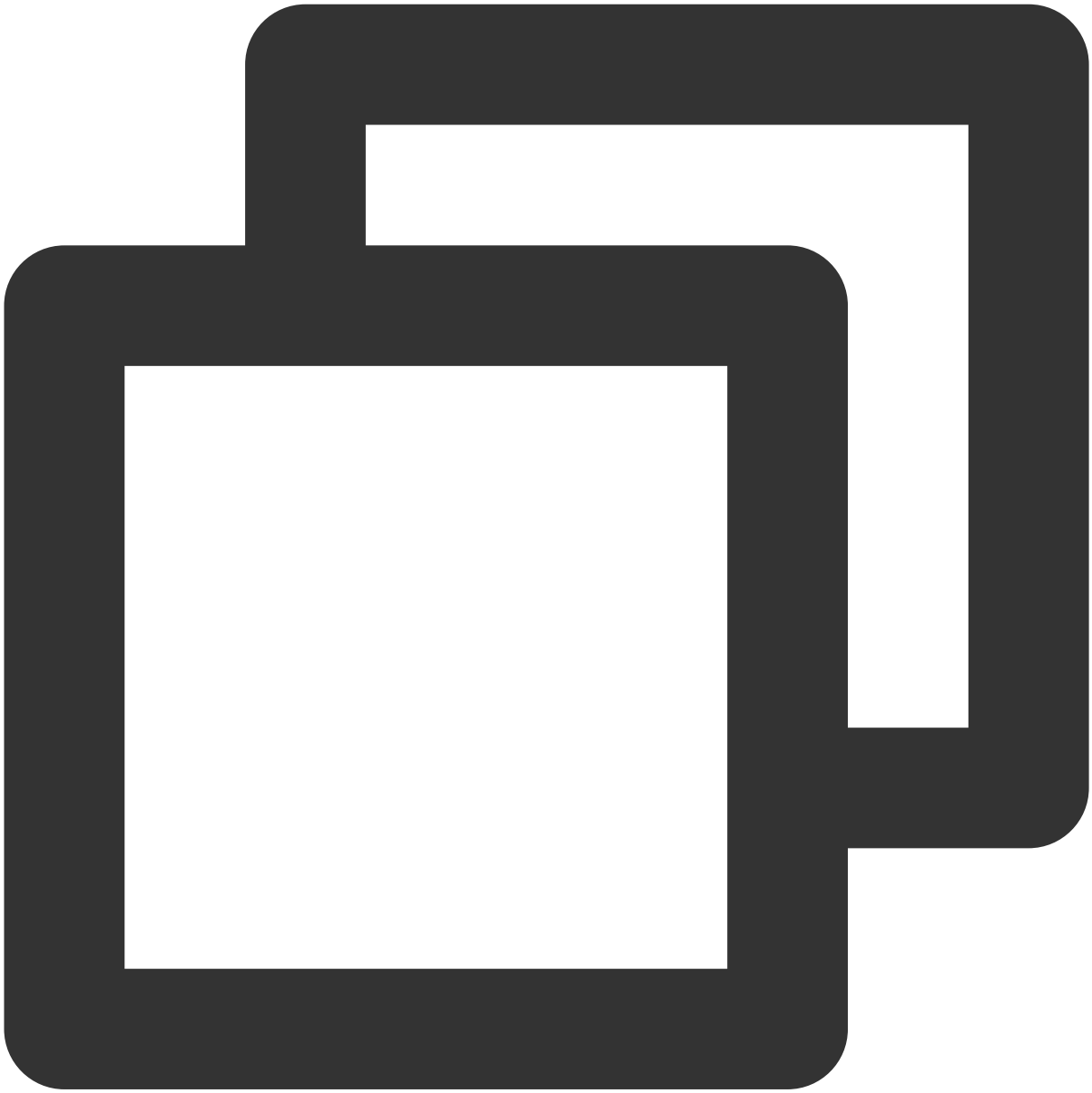
参数如下表所示：

参数	类型	含义
volumeMap	Array<Object>	音量表，根据每个 userid 可以获取对应的音量大小，音量最小值0，音量最大

		值100

GROUP_CALL_INVITEE_LIST_UPDATE

群聊更新邀请列表收到该事件。



```
let handleGroupInviteeListUpdate = function(event) {
  console.log(event.userIDList);
};
tuiCallEngine.on(TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeLis
```

参数如下表所示：

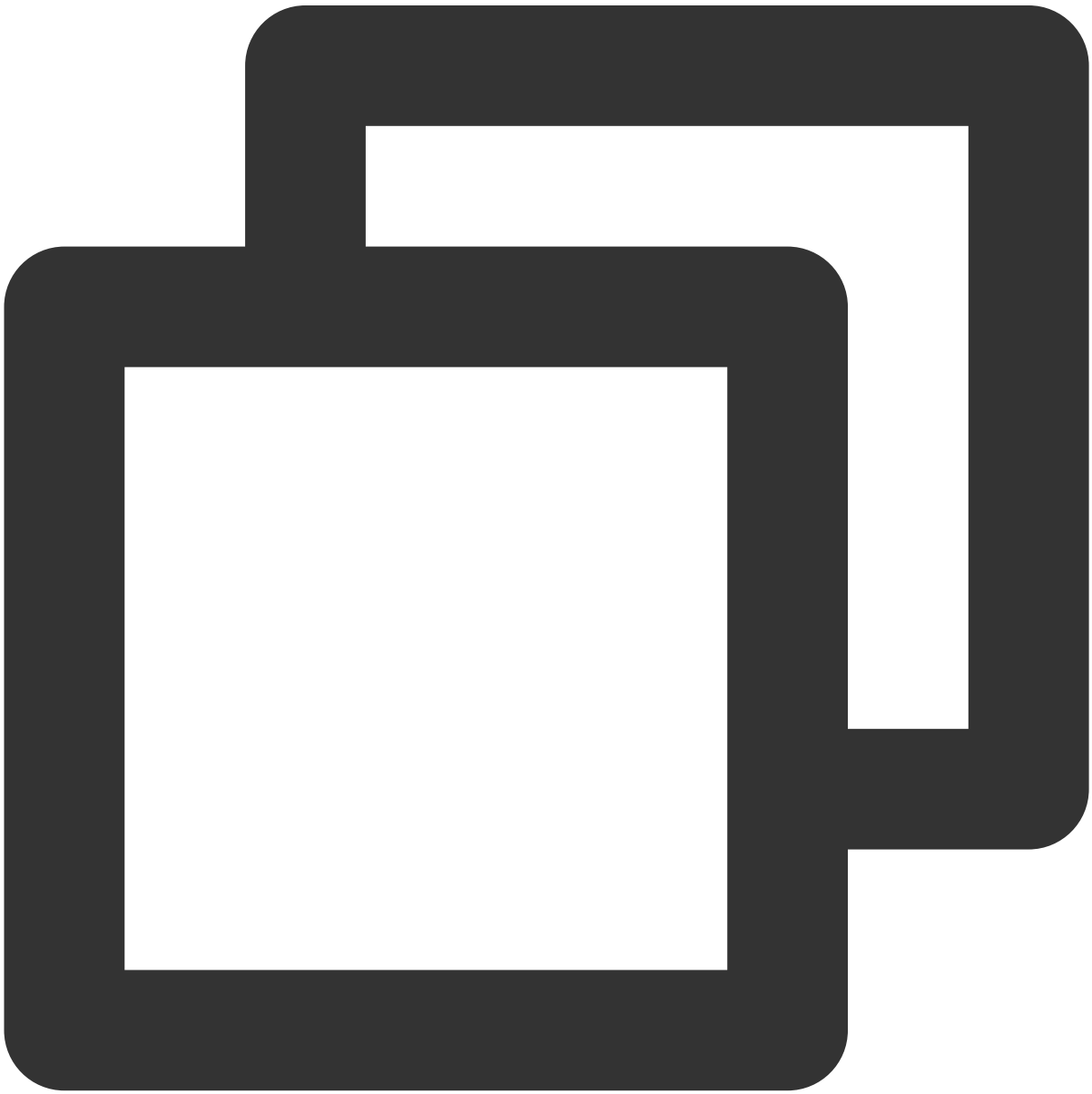
参数	类型	含义
userIDList	Array<String>	群组更新邀请列表

INVITED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

注意：

计划后续版本废弃



```
let handleNewInvitationReceived = function(event) {
  console.log(event.sponsor, event.userIDList, event.isFromGroup, event.inviteData,
};
tuiCallEngine.on(TUICallEvent.INVITED, handleNewInvitationReceived);
```

参数如下表所示：

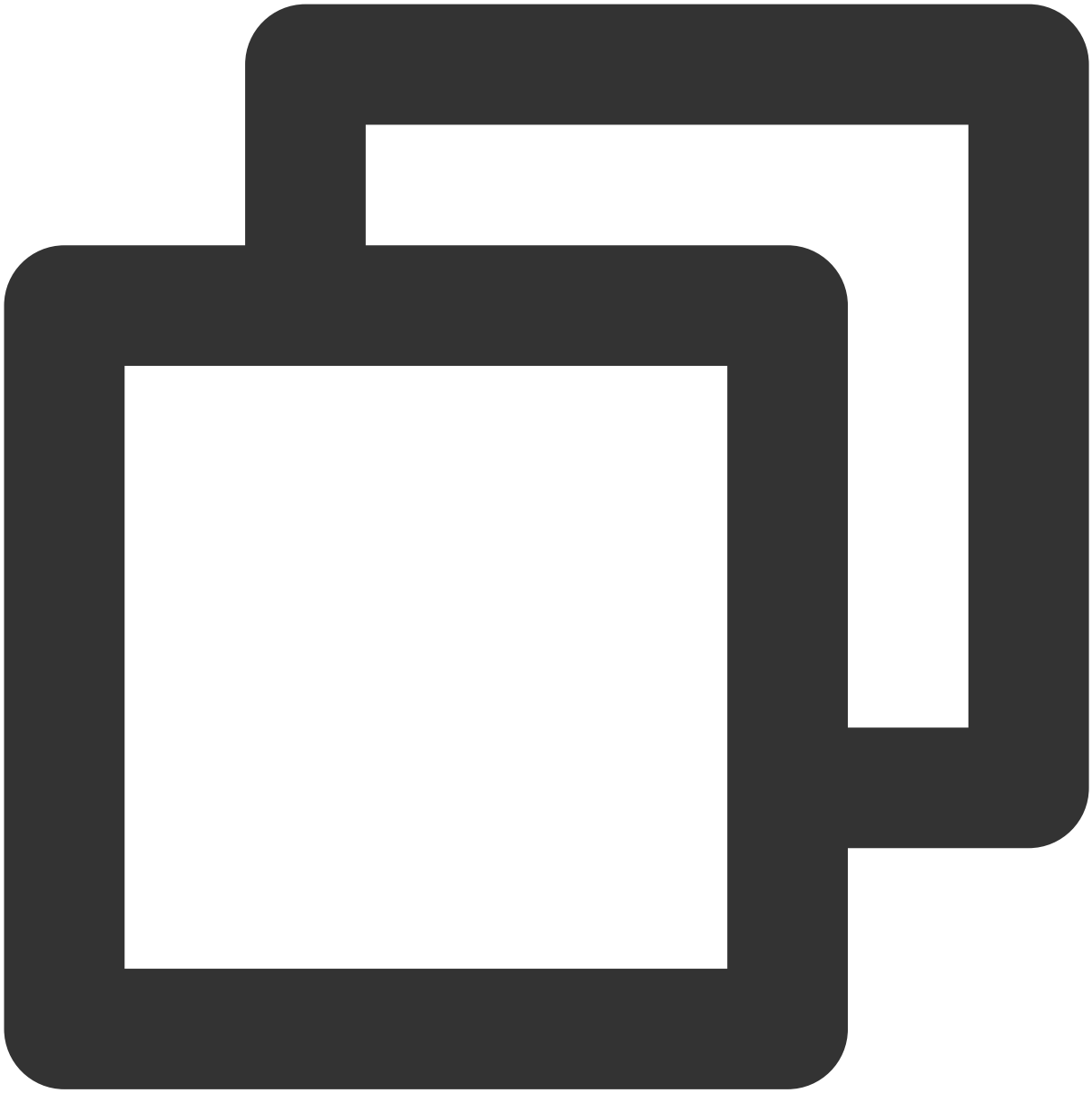
参数	类型	含义
sponsor	String	邀请者
userIDList	Array<String>	同时还被邀请的人
isFromGroup	Boolean	是否是群组通话
inviteData	Object	呼叫数据
inviteID	String	邀请 ID，标识一次邀请
userData	String	扩展字段：用于在邀请信令中增加扩展信息
callId	String	本次通话的唯一 ID。**v1.4.6+ 版本支持**
roomID	Number	此次通话的音视频房间 ID。**v1.4.6+ 版本支持**
callMediaType	Number	通话的媒体类型，视频通话、语音通话。**v1.4.6+ 版本支持**
callRole	String	角色，枚举类型：主叫、被叫。**v1.4.6+ 版本支持**

ON_CALL_RECEIVED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

注意：

v1.4.6+ 版本支持



```
let handleOnCallReceived = function(event) {
    console.log(event)
};
tuiCallEngine.on(TUICallEvent.ON_CALL_RECEIVED, handleOnCallReceived);
```

参数如下表所示：

参数	类型	含义
sponsor	String	邀请者

userIDList	Array<String>	同时还被邀请的人
isFromGroup	Boolean	是否是群组通话
inviteData	Object	呼叫数据
inviteID	String	邀请 ID，标识一次邀请
userData	String	扩展字段：用于在邀请信令中增加扩展信息
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫、被叫

CALLING_CANCEL

如果通话未建立，都会抛出该事件。您可以通过监听该事件来实现类似未接来电、重置 UI 状态等显示逻辑。通话未建立的场景如下：

主叫取消：主叫抛出该事件，userID 为主叫；被叫也会抛出该事件，userID 为被叫；

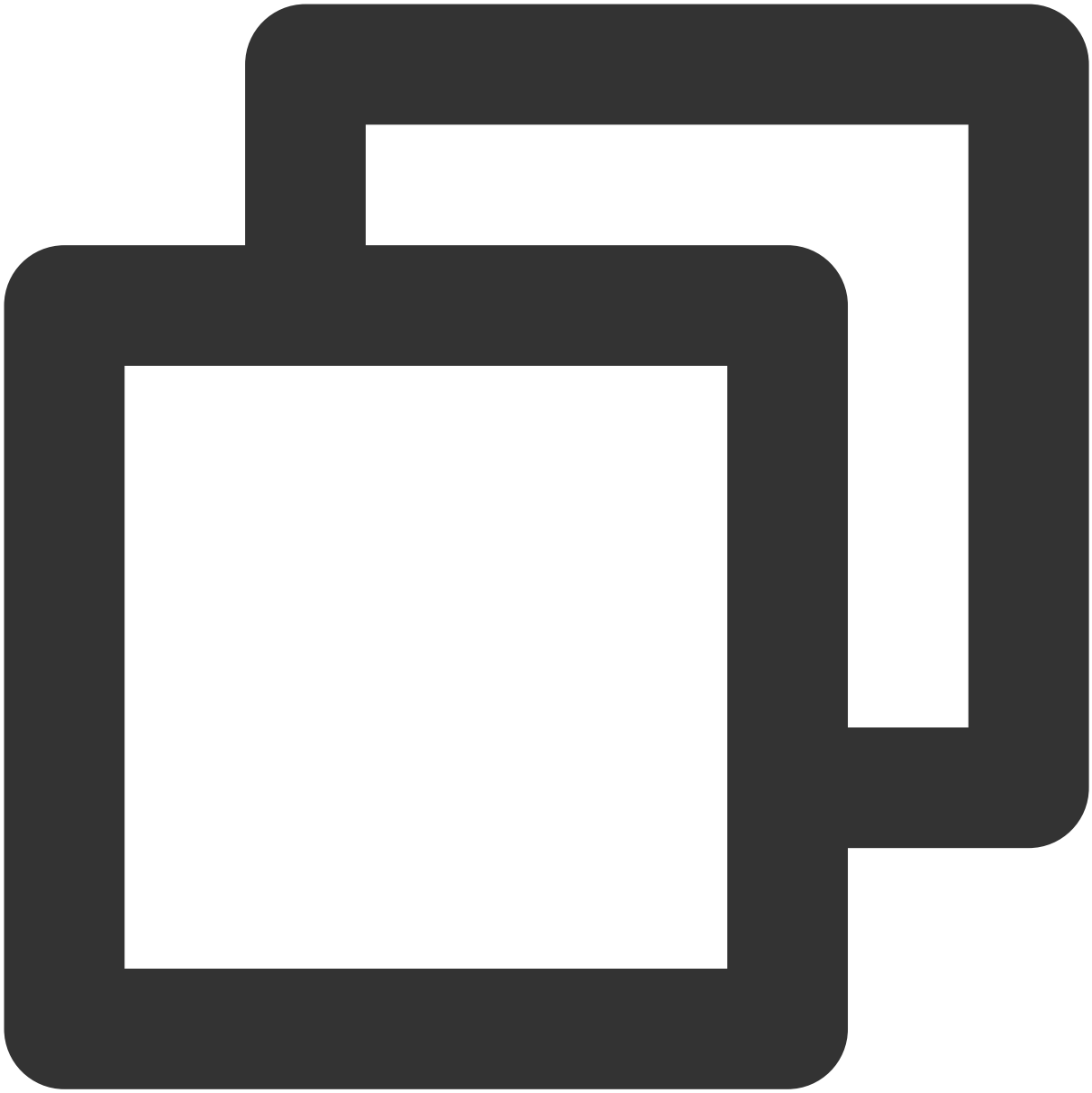
被叫超时：主叫会同时抛出 NO_RESP 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

被叫拒接：主叫会同时抛出 REJECT 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

被叫忙线：主叫会同时抛出 LINE_BUSY 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

注意：

计划后续版本废弃



```
let handleCallingCancel = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.CALLING_CANCEL, handleCallingCancel);
```

参数如下表所示：

参数	类型	含义
userID	String	通话取消的用户 ID

callId	String	本次通话的唯一 ID。**v1.4.6+ 版本支持**
roomID	Number	此次通话的音视频房间 ID。**v1.4.6+ 版本支持**
callMediaType	Number	通话的媒体类型，视频通话、语音通话。**v1.4.6+ 版本支持**
callRole	String	角色，枚举类型：主叫、被叫。**v1.4.6+ 版本支持**

ON_CALL_CANCELED

如果通话未建立，都会抛出该事件。您可以通过监听该事件来实现类似未接来电、重置 UI 状态等显示逻辑。通话未建立的场景如下：

主叫取消：主叫抛出该事件，userID 为主叫；被叫也会抛出该事件，userID 为被叫；

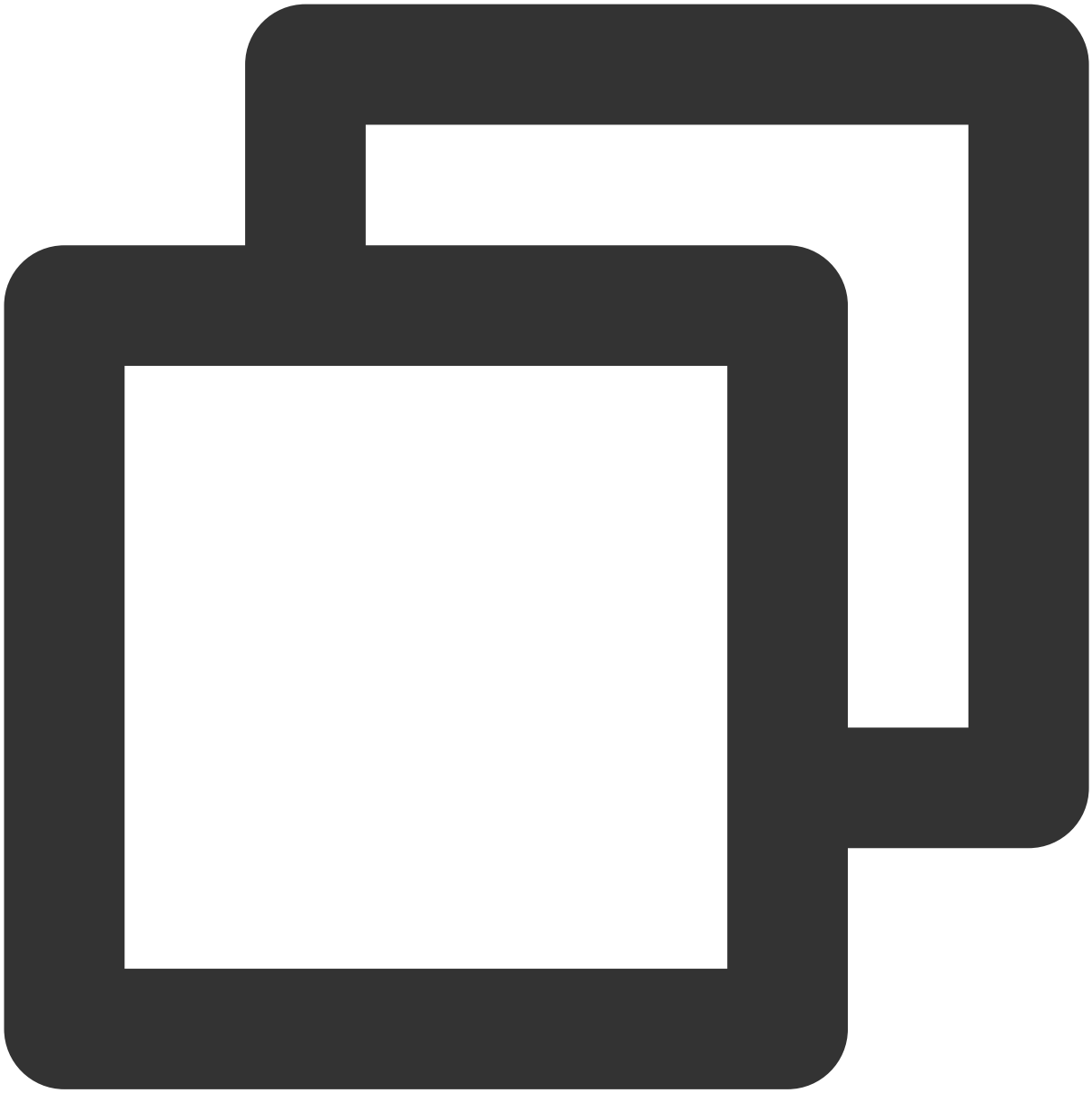
被叫超时：主叫会同时抛出 NO_RESP 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

被叫拒接：主叫会同时抛出 REJECT 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

被叫忙线：主叫会同时抛出 LINE_BUSY 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

注意：

v1.4.6+ 版本支持



```
let handleOnCallCanceled = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_CANCELED, handleOnCallCanceled);
```

参数如下表所示：

参数	类型	含义
userID	String	通话取消的用户 ID

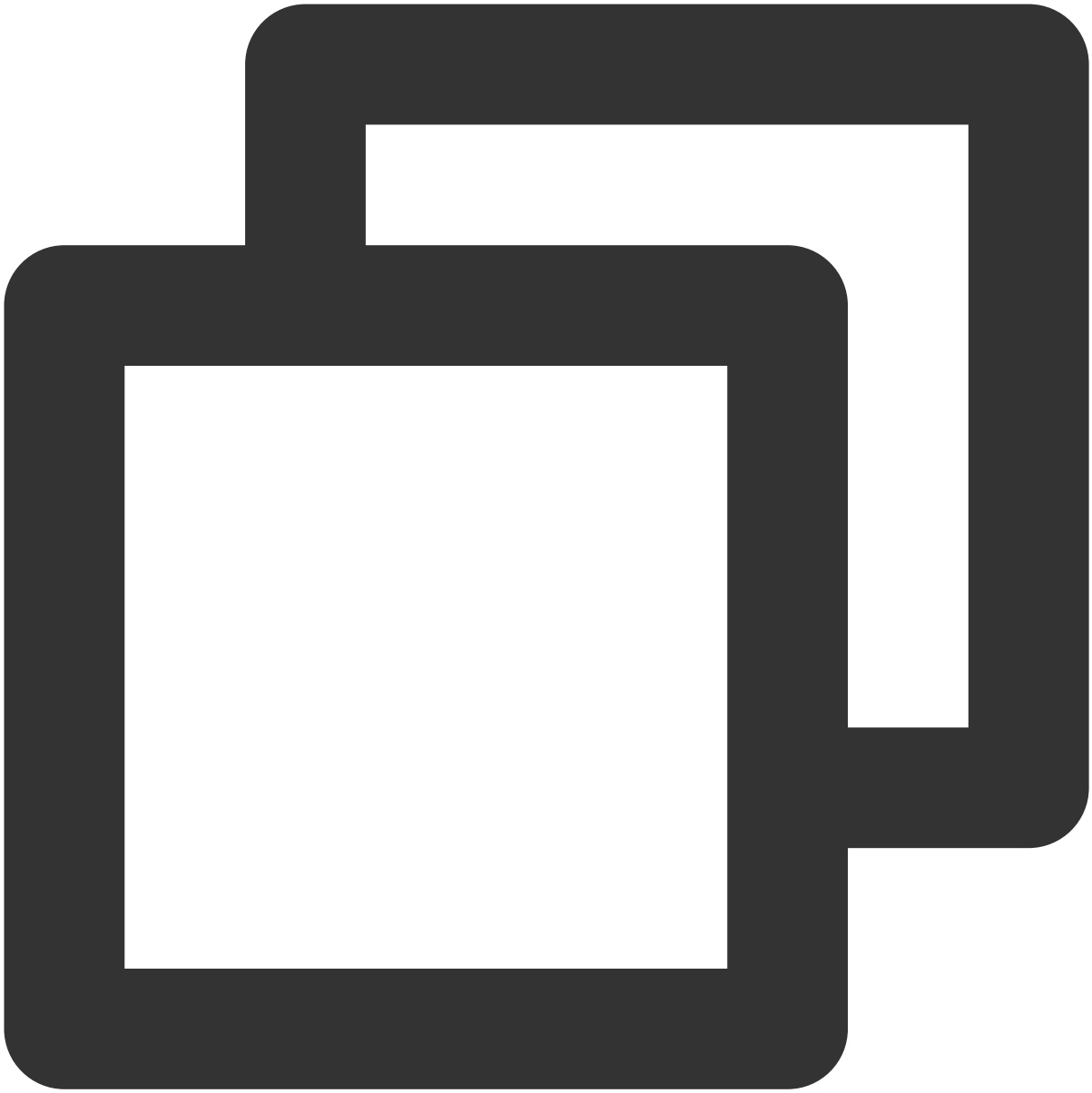
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫、被叫

ON_CALL_BEGIN

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

注意：

v1.4.6+ 版本支持



```
let handleOnCallBegin = function(event) {
    console.log(event)
};
tuiCallEngine.on(TUICallEvent.ON_CALL_BEGIN, handleOnCallBegin);
```

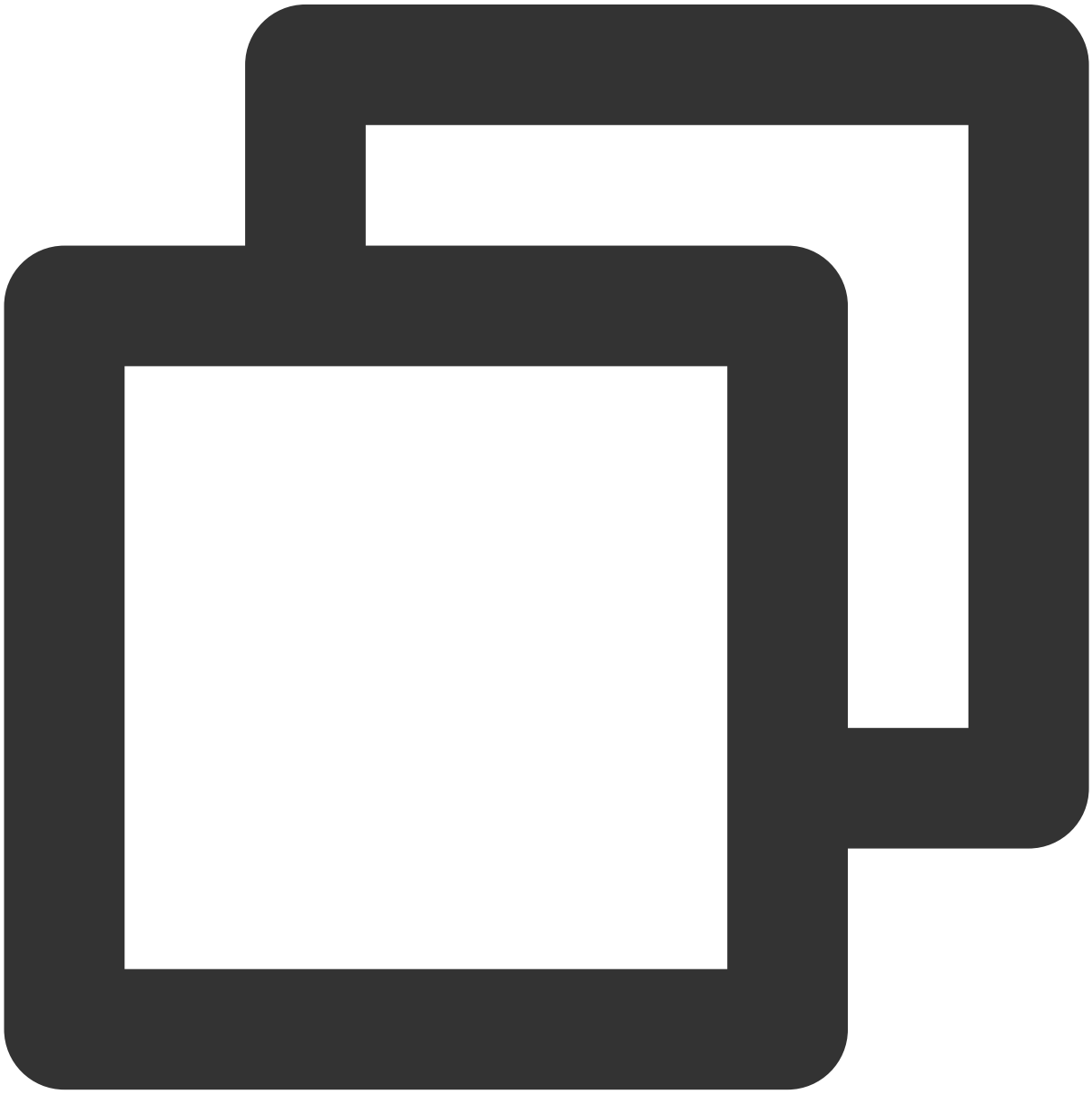
参数如下表所示：

参数	类型	含义
callId	String	本次通话的唯一 ID

roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，类型：主叫、被叫

CALLING_END

表示通话结束，主叫和被叫抛出该事件。您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。



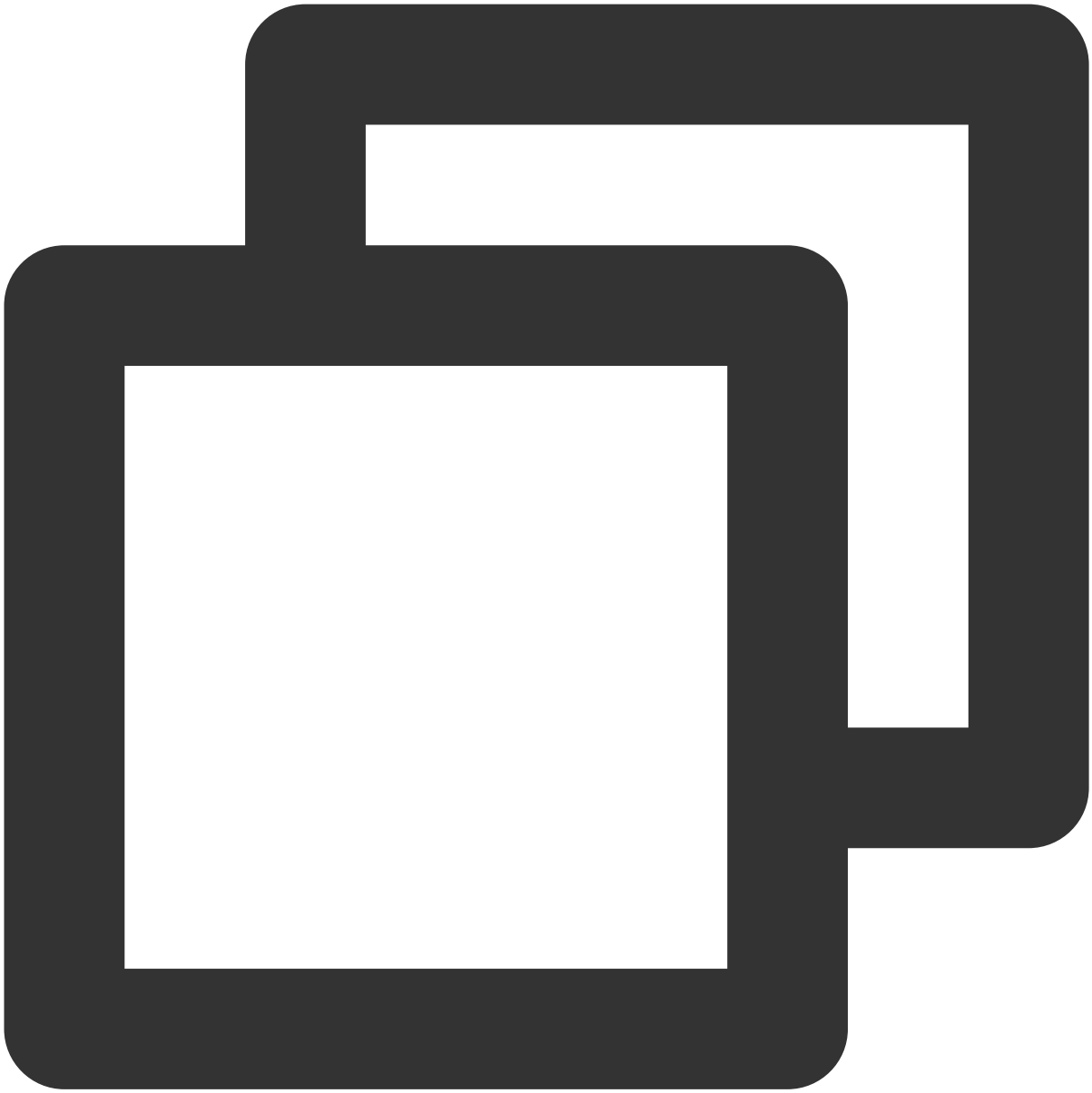
```
let handleCallingEnd = function(event) {
  console.log(event.userID, event.);
};
tuiCallEngine.on(TUICallEvent.CALLING_END, handleCallingEnd);
```

参数如下表所示：

参数	类型	含义
roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫（'inviter'）、被叫（'invitee'）、未知（''）
totalTime	Number	此次通话的时长，单位: 秒
userID	String	通话结束的 userID。
callId	String	本次通话的唯一 ID。**v1.4.6+ 版本支持**
callEnd	Number	此次通话的时长（**后续将废弃**），单位: 秒

DEVICED_UPDATED

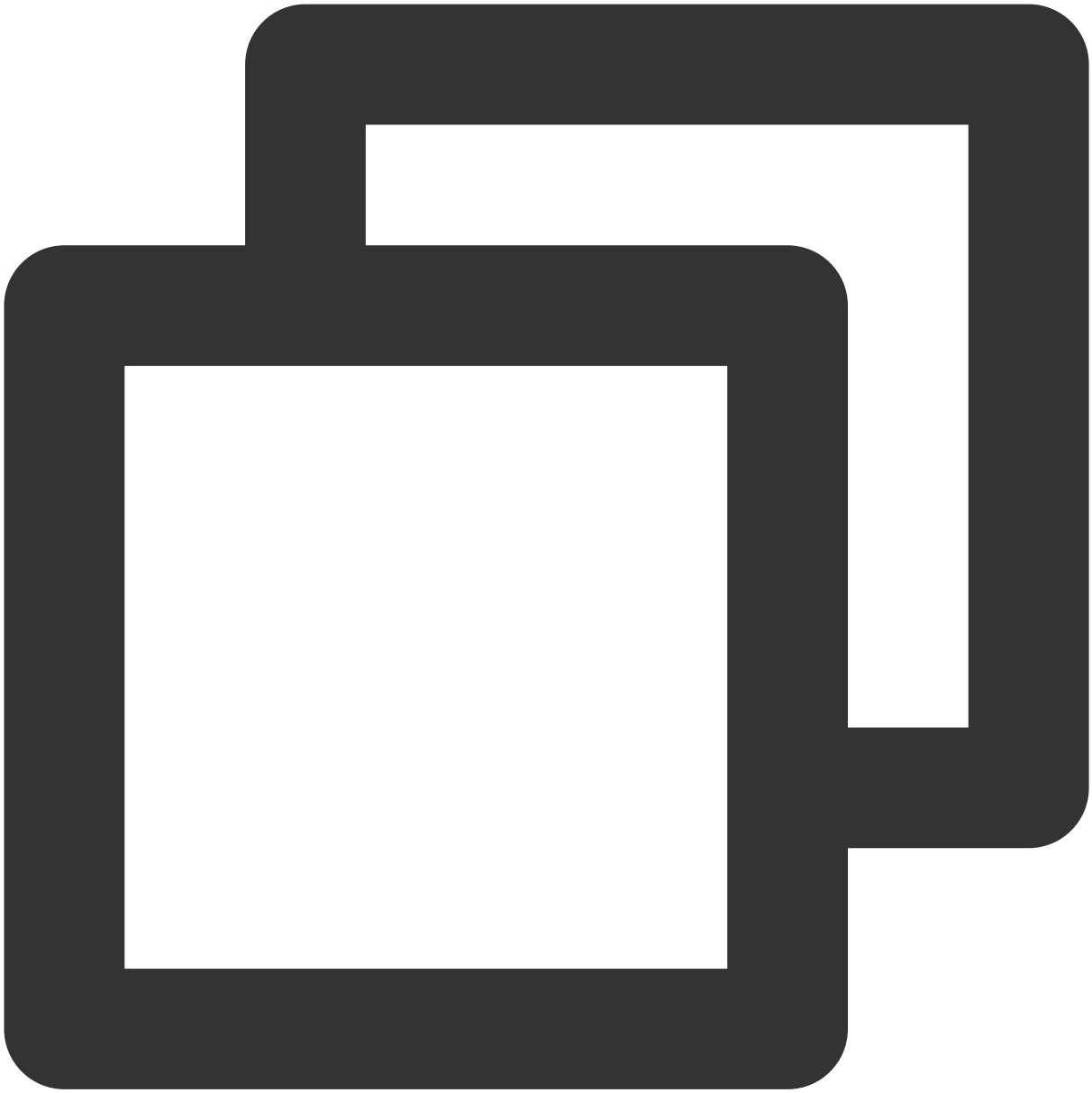
设备列表更新收到该事件。



```
let handleDeviceUpdated = function({ microphoneList, cameraList, currentMicrophoneI
  console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID)
};
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

CALL_TYPE_CHANGED

通话类型切换收到该事件。



```
let handleCallTypeChanged = function({ oldCallType, newCallType }) {
  console.log(oldCallType, newCallType)
};
tuiCallEngine.on(TUICallEvent.CALL_TYPE_CHANGED, handleDeviceUpdated);
```

参数如下表所示：

参数	类型	含义
oldCallType	Number	旧通话类型

newCallType	Number	新通话类型

Flutter
API 概览

最近更新时间：2024-03-12 14:51:07

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的昵称、头像
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
setCallingBell	自定义来电铃声

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
init	初始化SDK函数，完成包含通话服务鉴权在内初始化动作
unInit	反初始化函数，释放TUICallEngine使用到的相关资源。
addObserver	增加事件回调

removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）
setSelfInfo	设置用户的头像、昵称
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
queryRecentCalls	查询通话记录
deleteRecordCalls	删除通话记录
setBeautyLevel	设置美颜级别

版权所有：腾讯云计算（北京）有限责任公司

TUICallObserver

TUICallObserver 是回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2024-03-12 14:51:07

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入（TUICallKit）。

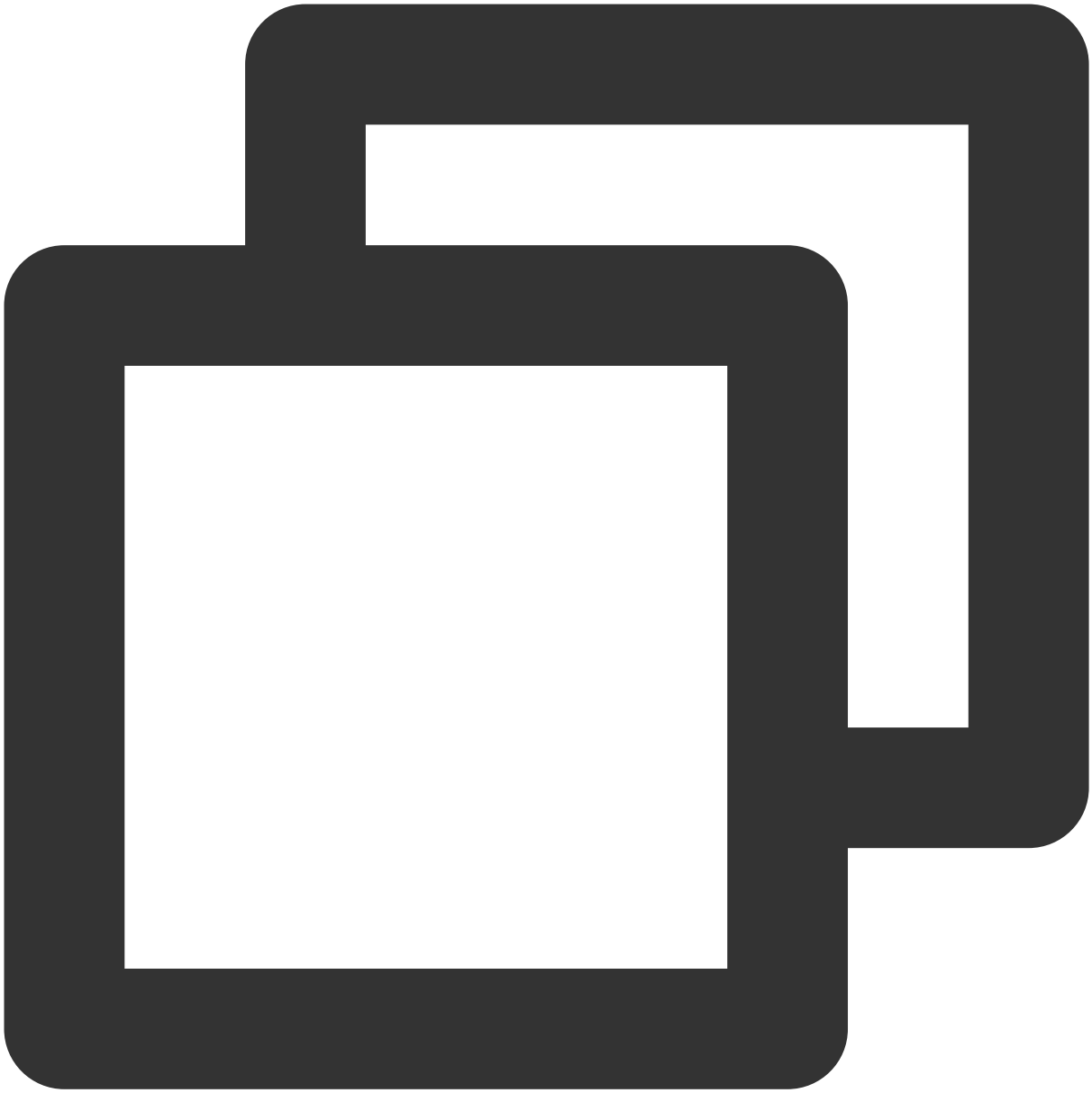
API 概览

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的昵称、头像
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
setCallingBell	自定义来电铃声

API 详情

login

登录。



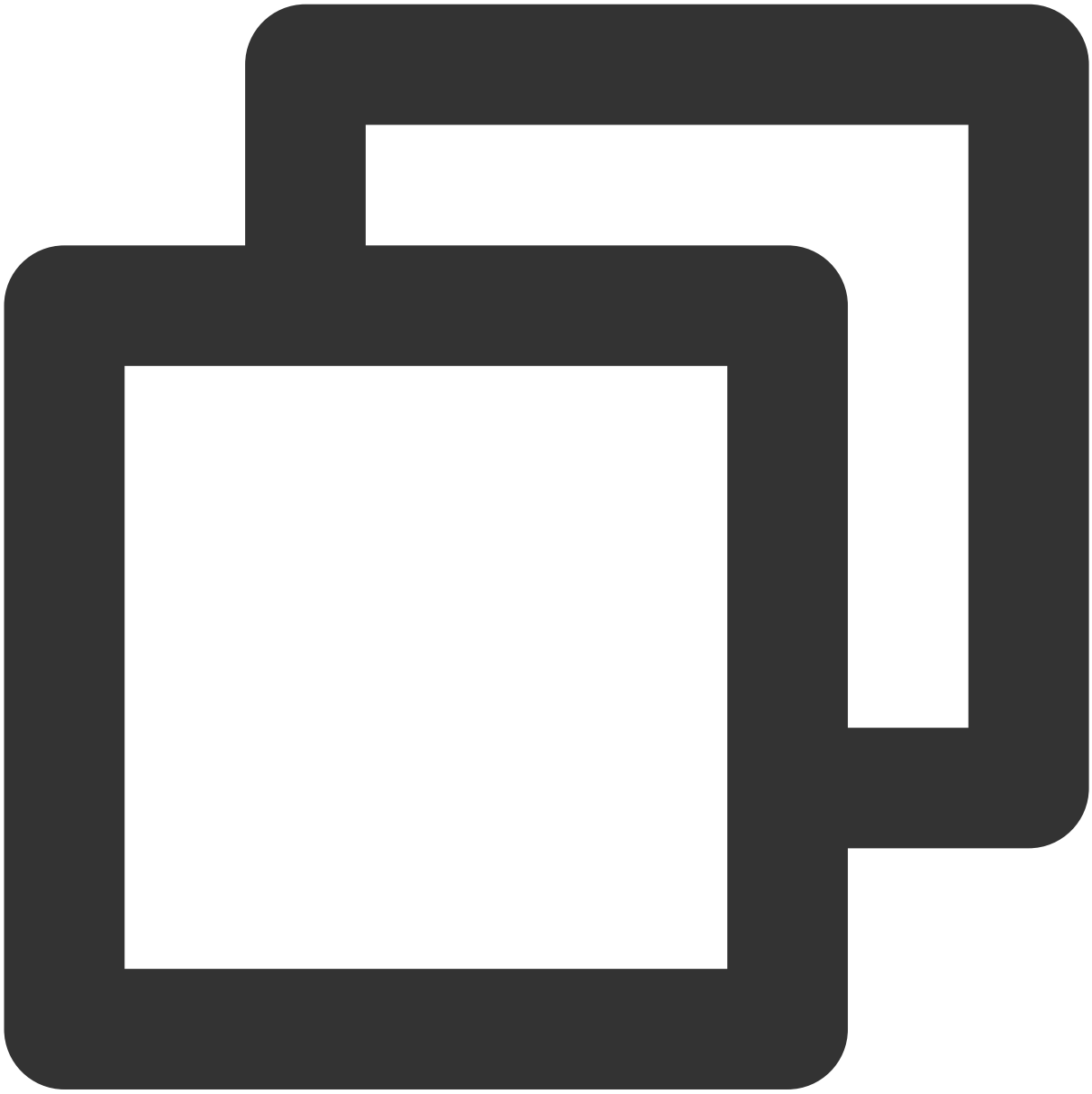
```
Future<TUIResult> login(int sdkAppId, String userId, String userSig);
```

参数	类型	含义
sdkAppId	int	用户 SDKAppID
userId	String	用户 ID
userSig	String	用户签名 userSig（计算方法详见）
返回值	TUIResult	包含code和message信息：code = 0 表示调用成功；code != 0 表示调

		用失败，失败原因见message

logout

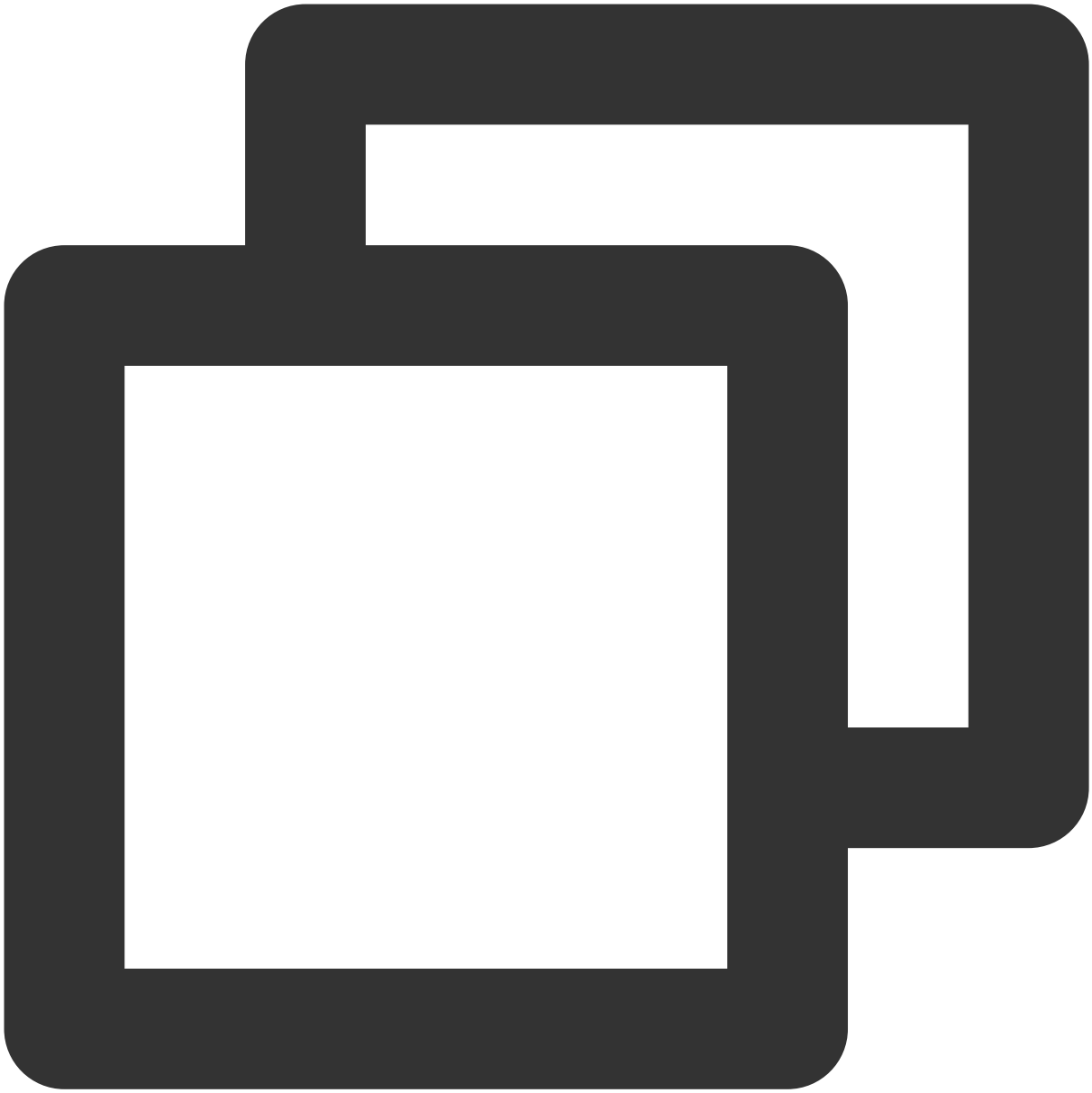
登出。



```
Future<void> logout()
```

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是URL格式。

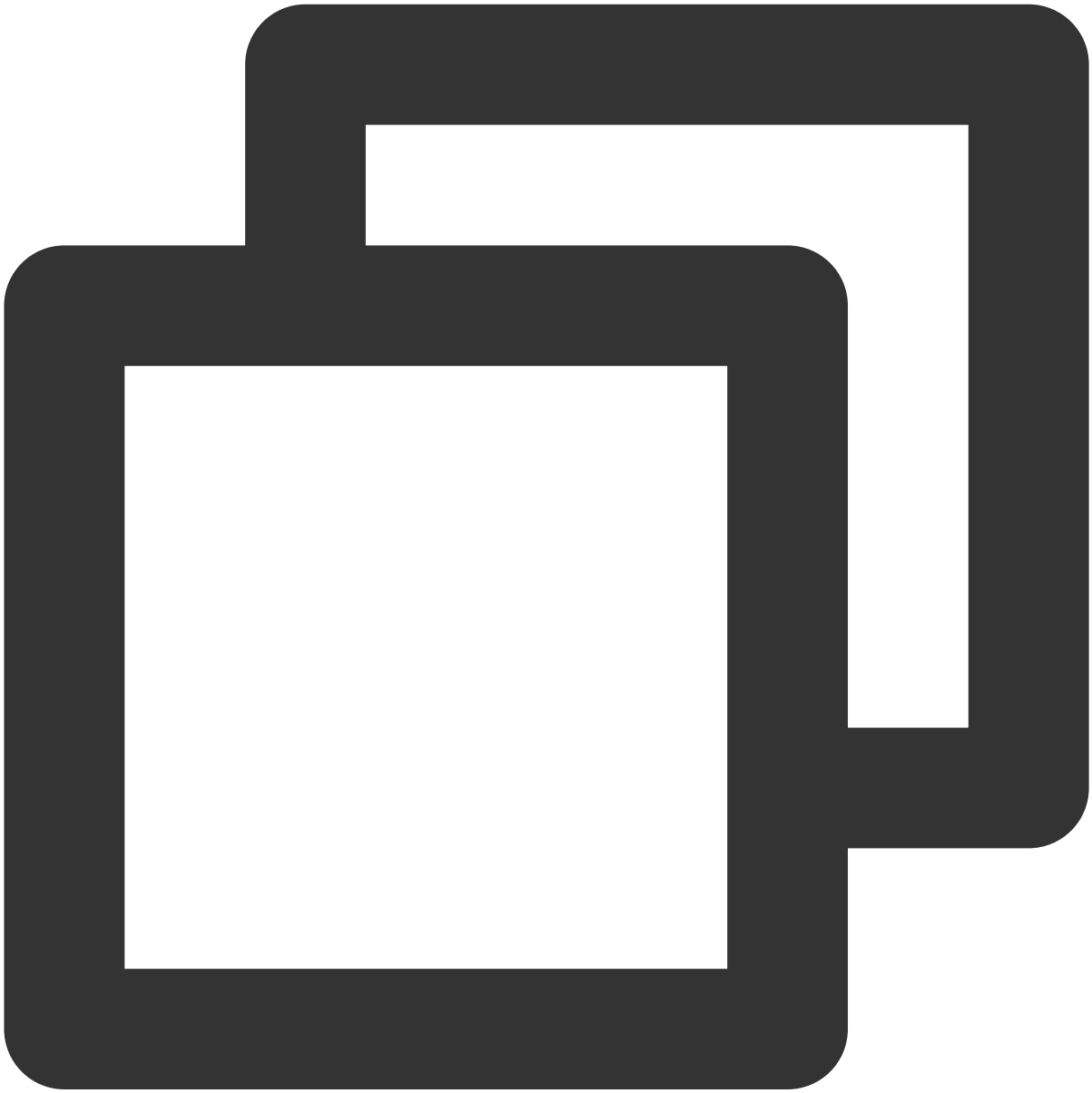


```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

参数	类型	含义
nickName	String	目标用户的昵称，非必填
avatar	String	目标用户的头像，非必填
返回值	TUIResult	包含code和message信息：code = 0 表示调用成功；code != 0 表示调用失败，失败原因见message

call

拨打电话（1v1通话）。



```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa
```

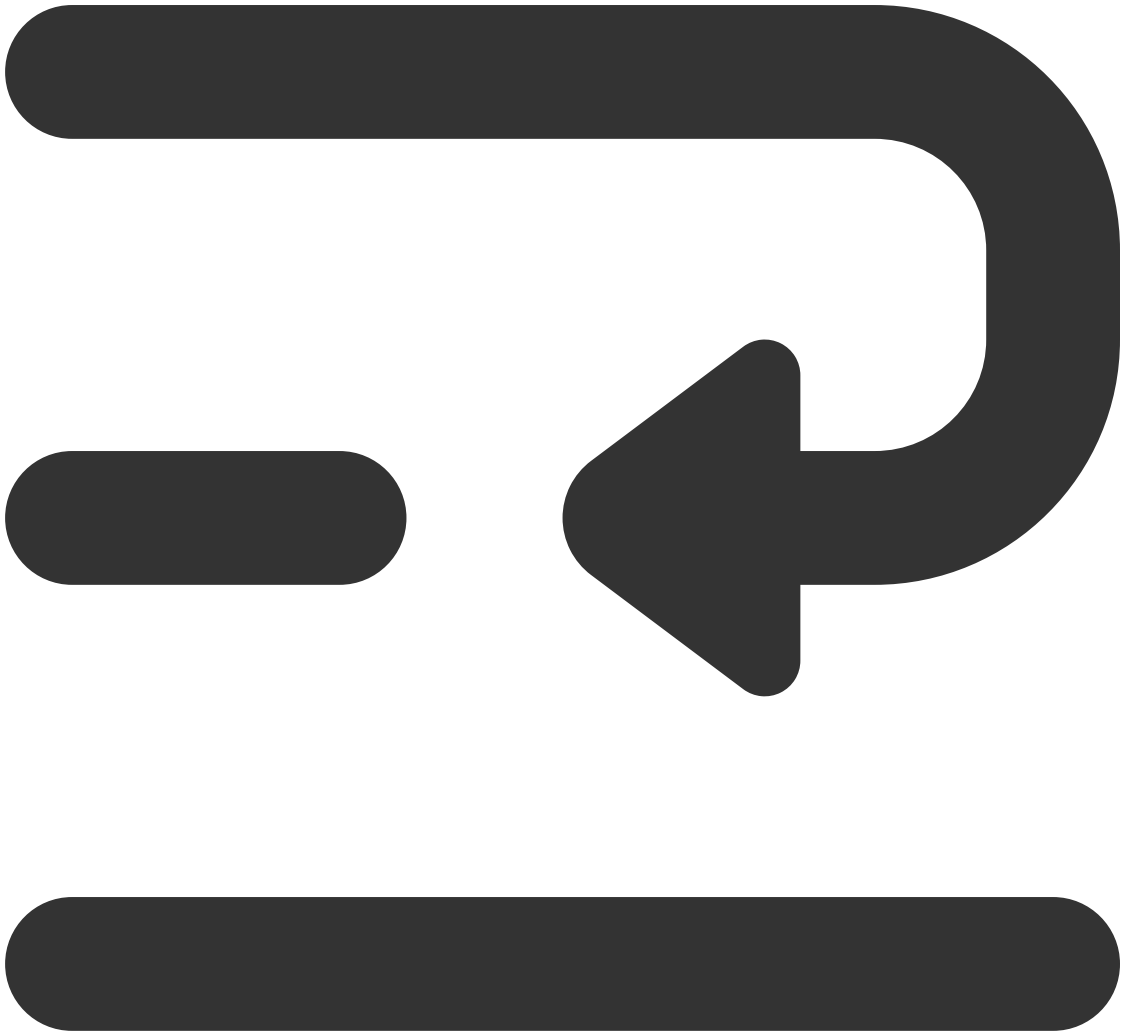
参数如下表所示：

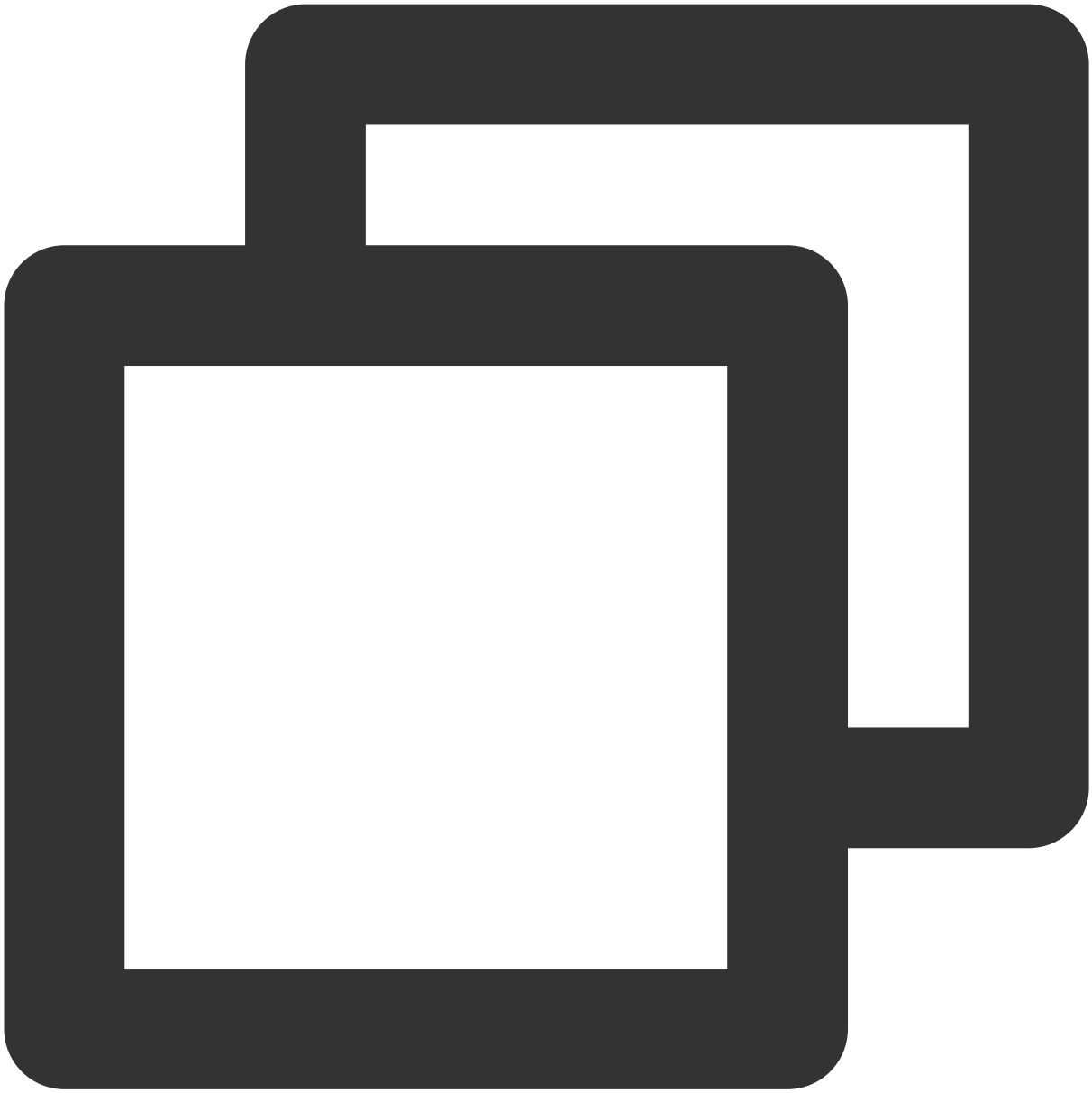
参数	类型	含义
userId	String	目标用户的 userID
callMediaType	TUICallMediaType	

		通话的媒体类型，比如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

groupCall

发起群组通话，注意：使用群组通话前需要创建IM 群组，如果已经创建，请忽略。



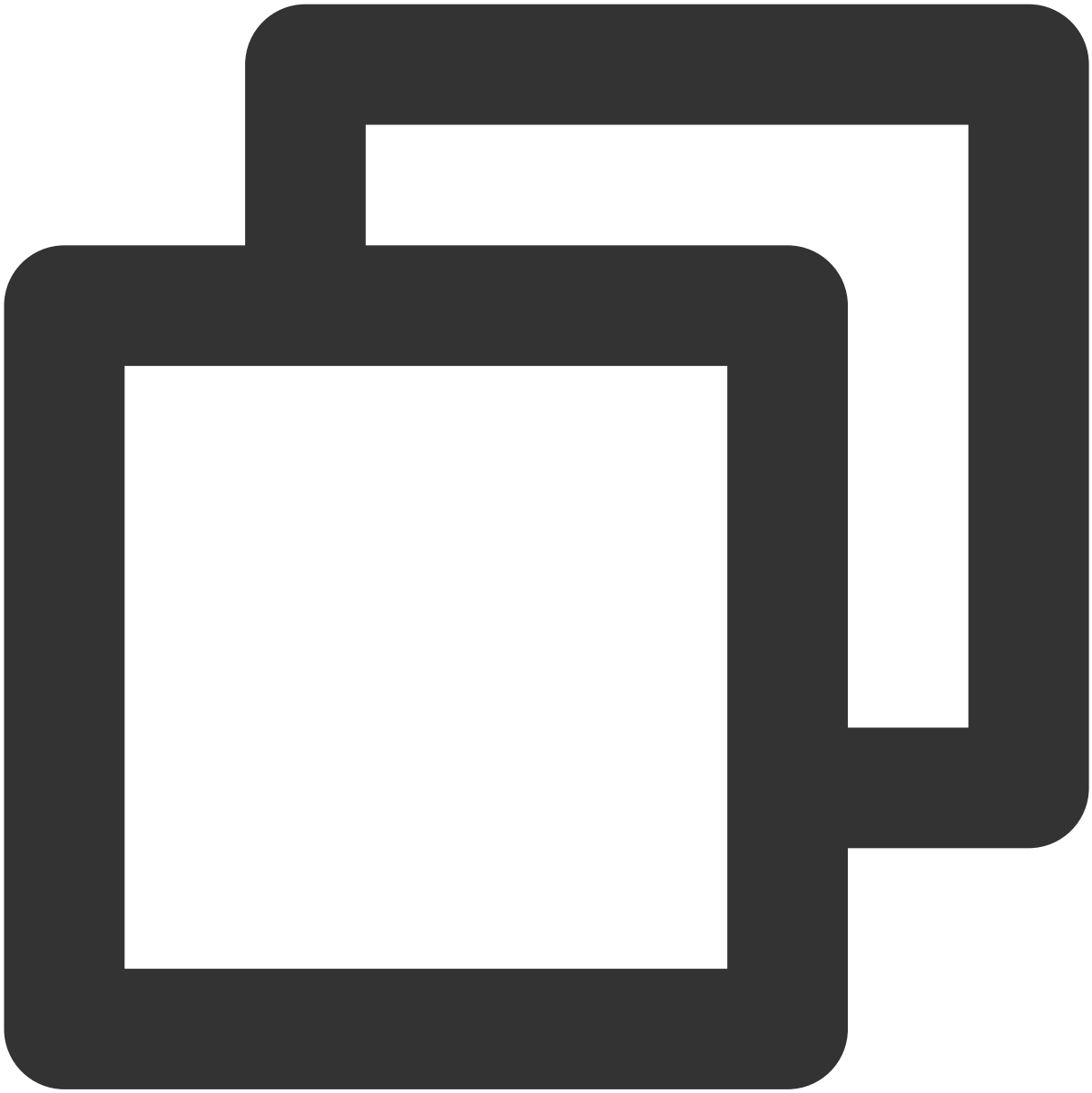


```
Future<void> groupCall(String groupId, List<String> userIdList, TUICallMediaType ca
```

参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List<String>	目标用户的userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

joinInGroupCall

加入群组中已有的音视频通话。



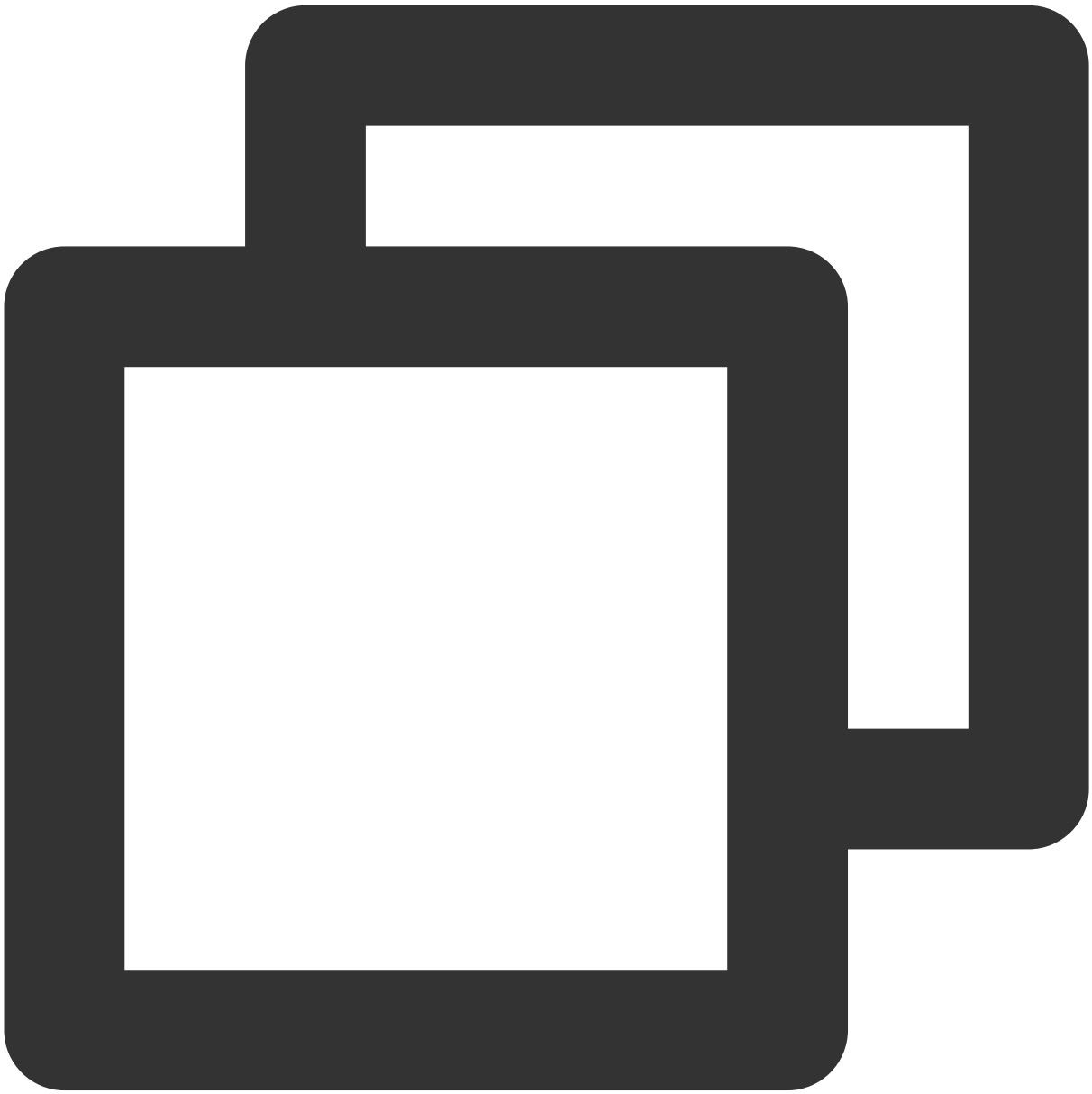
```
Future<void> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType cal
```

参数	类型	含义
roomId	TUIRoomID	此次通话的音视频房间 ID
groupId	String	此次群组通话的群 ID

callMediaType	TUICallMediaType	通话的媒体类型，比如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

enableMuteMode

开启/关闭静音模式。



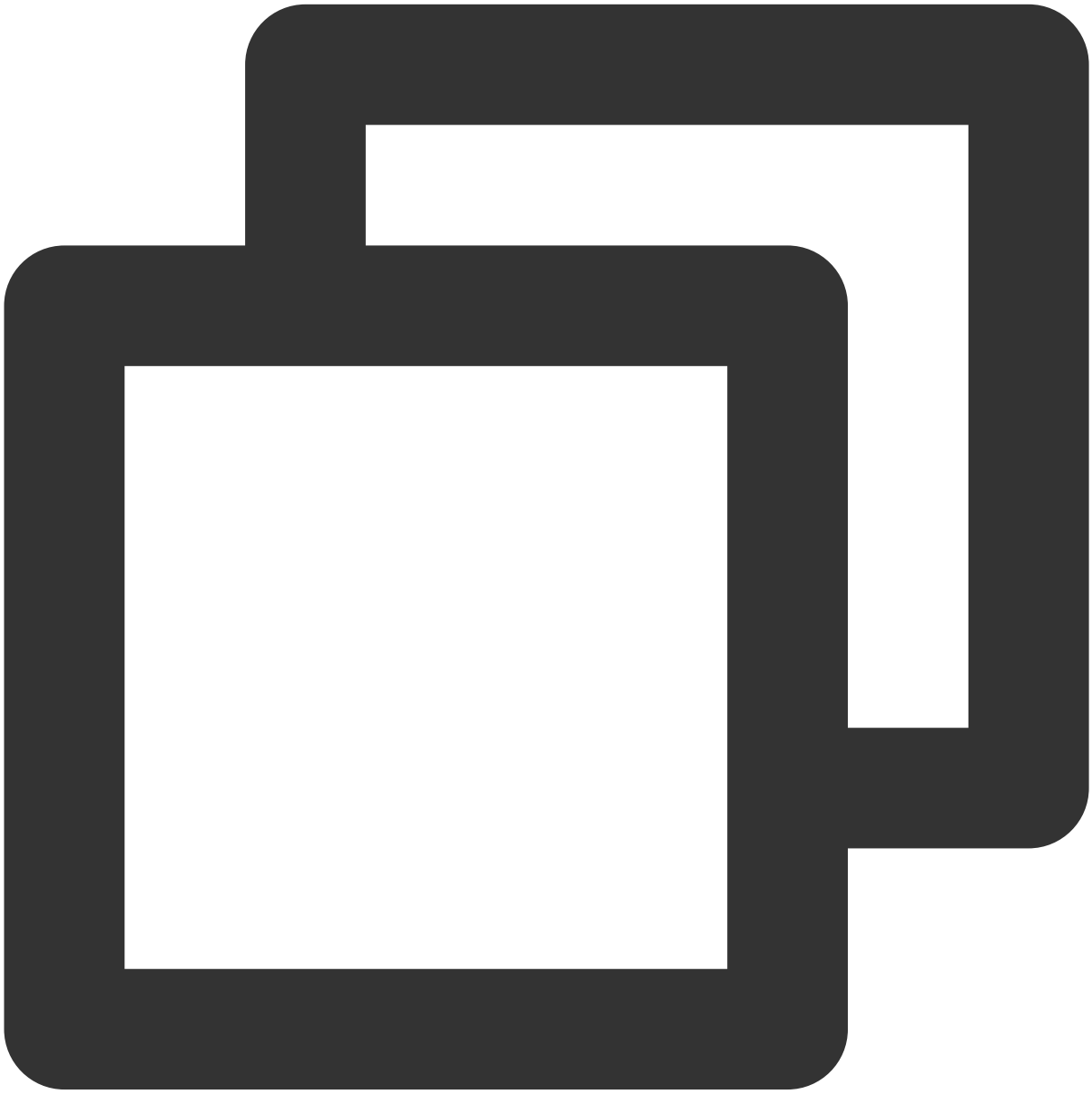
```
Future<void> enableMuteMode(bool enable)
```

参数	类型	含义

enable	Boolean	开启、关闭静音；true 表示开启静音

enableFloatWindow

开启/关闭悬浮窗功能，设置为false后，通话界面左上角的悬浮窗按钮会隐藏。

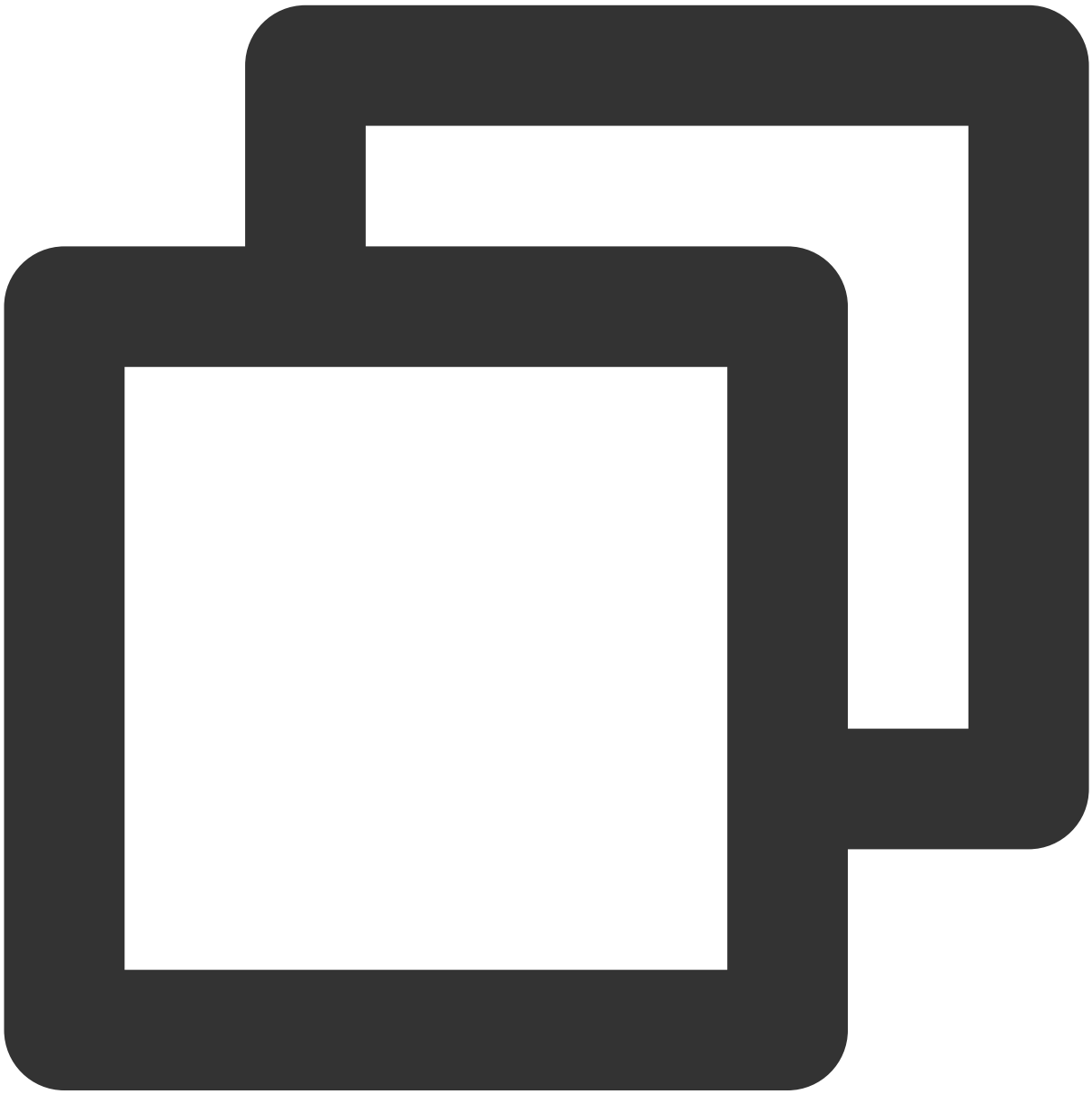


```
Future<void> enableFloatWindow(bool enable)
```

参数	类型	含义
enable	Boolean	开启、关闭悬浮窗功能；true 表示开启浮窗

setCallingBell

自定义来电铃声。



```
Future<void> setCallingBell(String assetName)
```

参数	类型	含义
assetName	String	铃声的路径，需要将铃声文件添加至主工程的 assets 资源中。

TUICallEngine

最近更新时间：2024-01-18 11:30:39

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
init	初始化SDK函数，完成包含通话服务鉴权在内初始化动作
unInit	反初始化函数，释放TUICallEngine使用到的相关资源。
addObserver	增加事件回调
removeObserver	移除回调接口
call	发起 1v1 通话
groupCall	发起群组通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在群组通话中，邀请其他人加入
joinInGroupCall	主动加入当前的群组通话中
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头

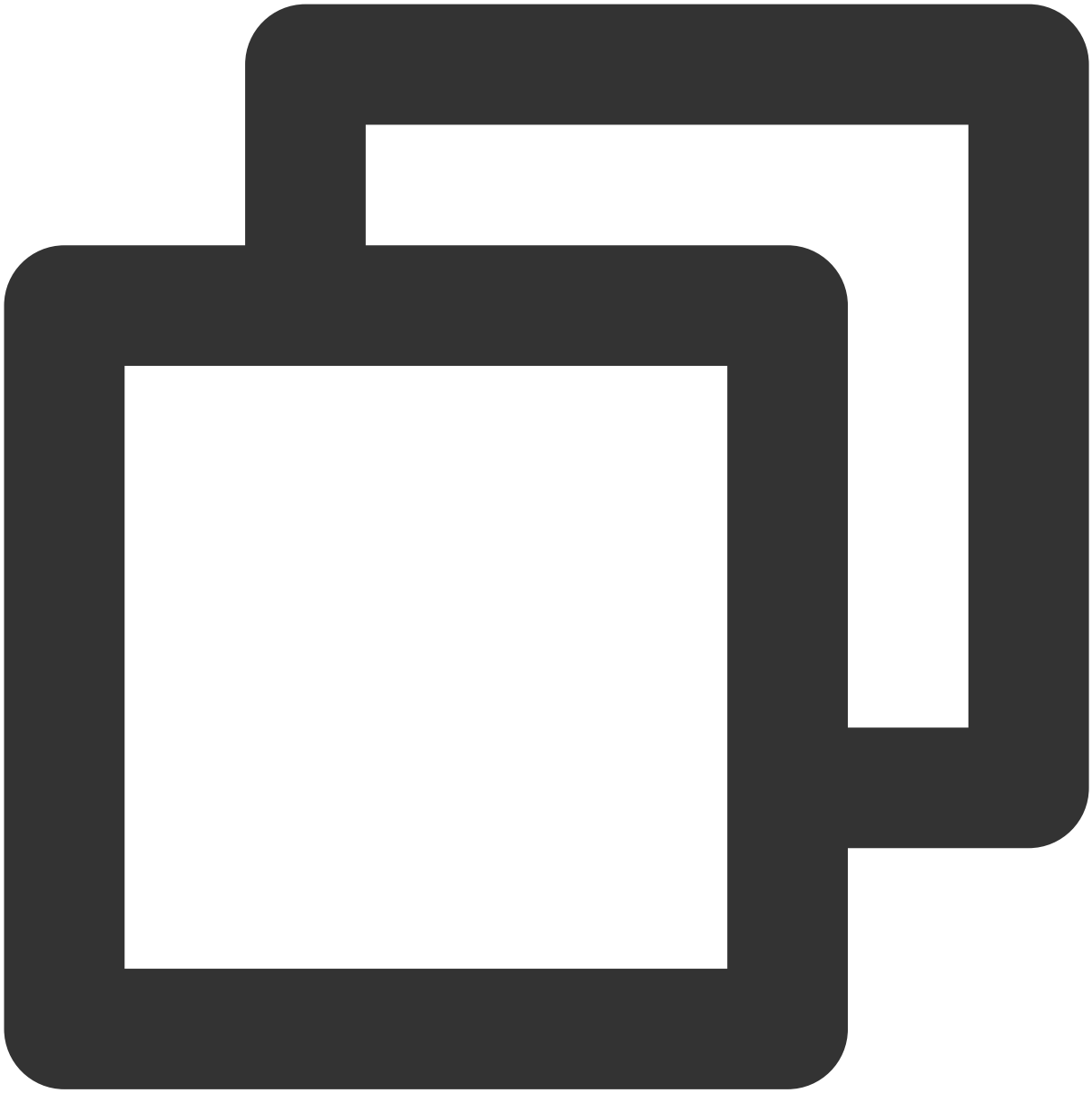
版权所有：腾讯云计算（北京）有限责任公司

closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）
setSelfInfo	设置用户的头像、昵称
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
queryRecentCalls	查询通话记录
deleteRecordCalls	删除通话记录
setBeautyLevel	设置美颜级别

API 详情

init

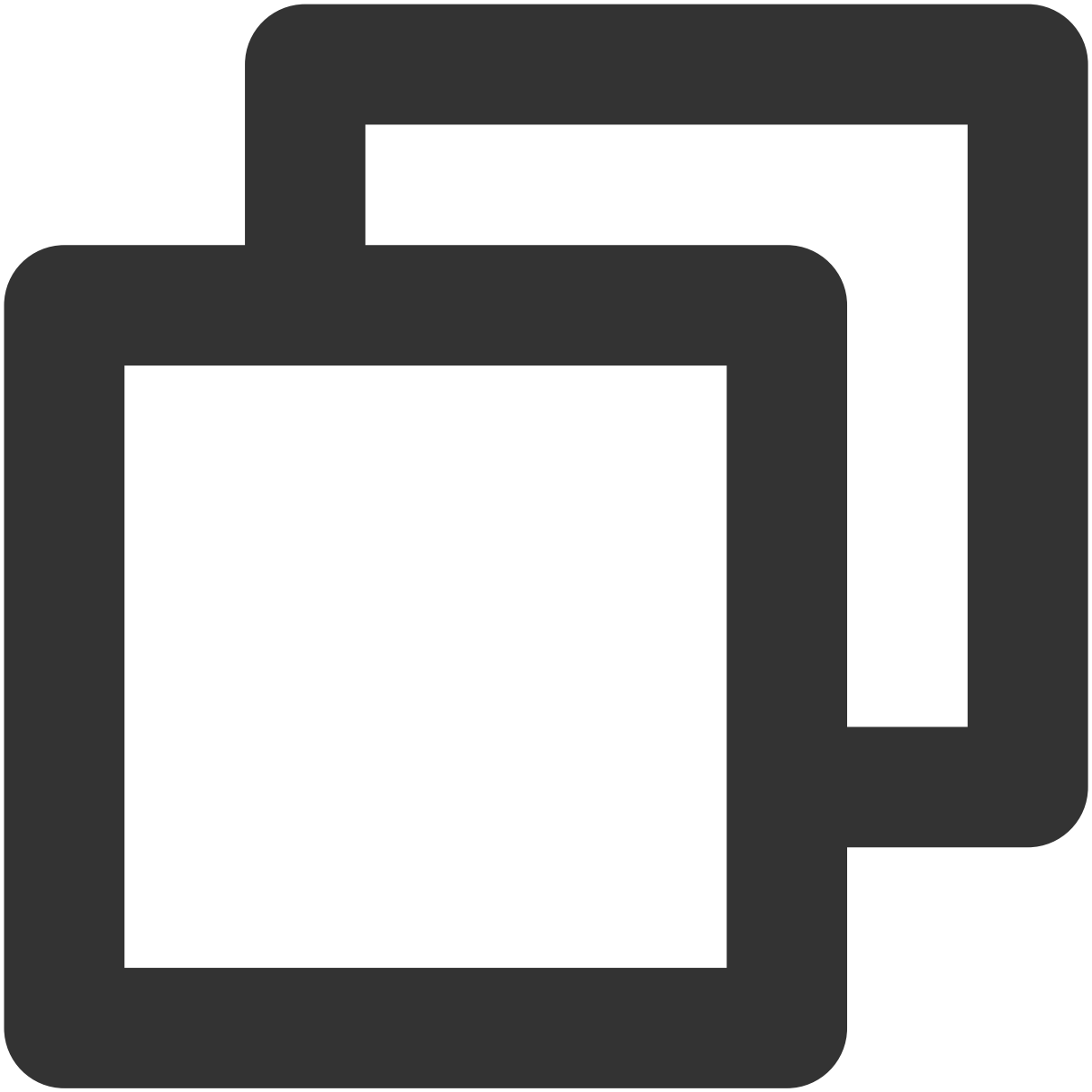
初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。



```
Future<TUIResult> init(int sdkAppID, String userId, String userSig)
```

unInit

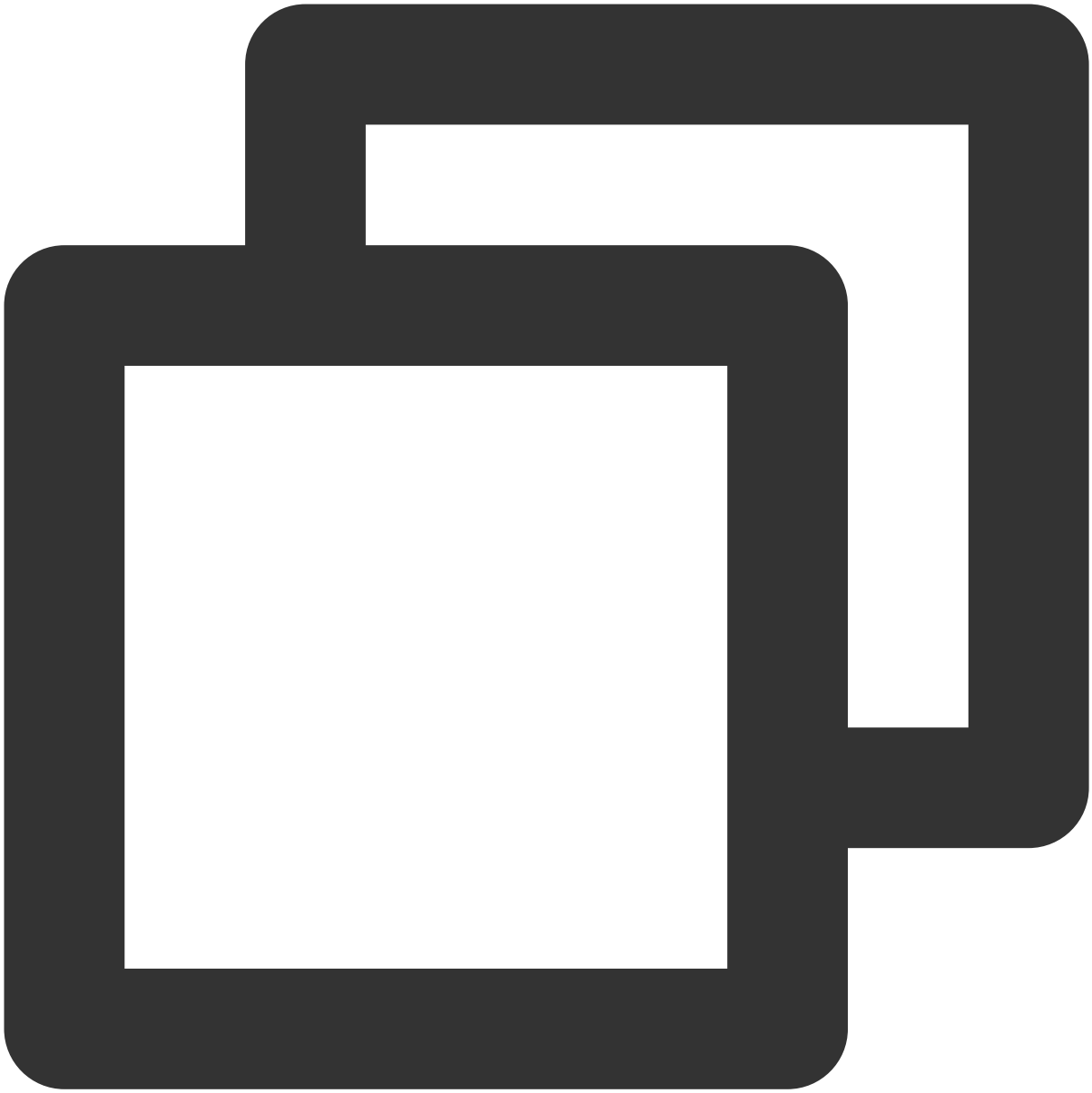
反初始化函数，释放 `TUICallEngine` 使用到的相关资源。



```
Future<TUIResult> unInit()
```

addObserver

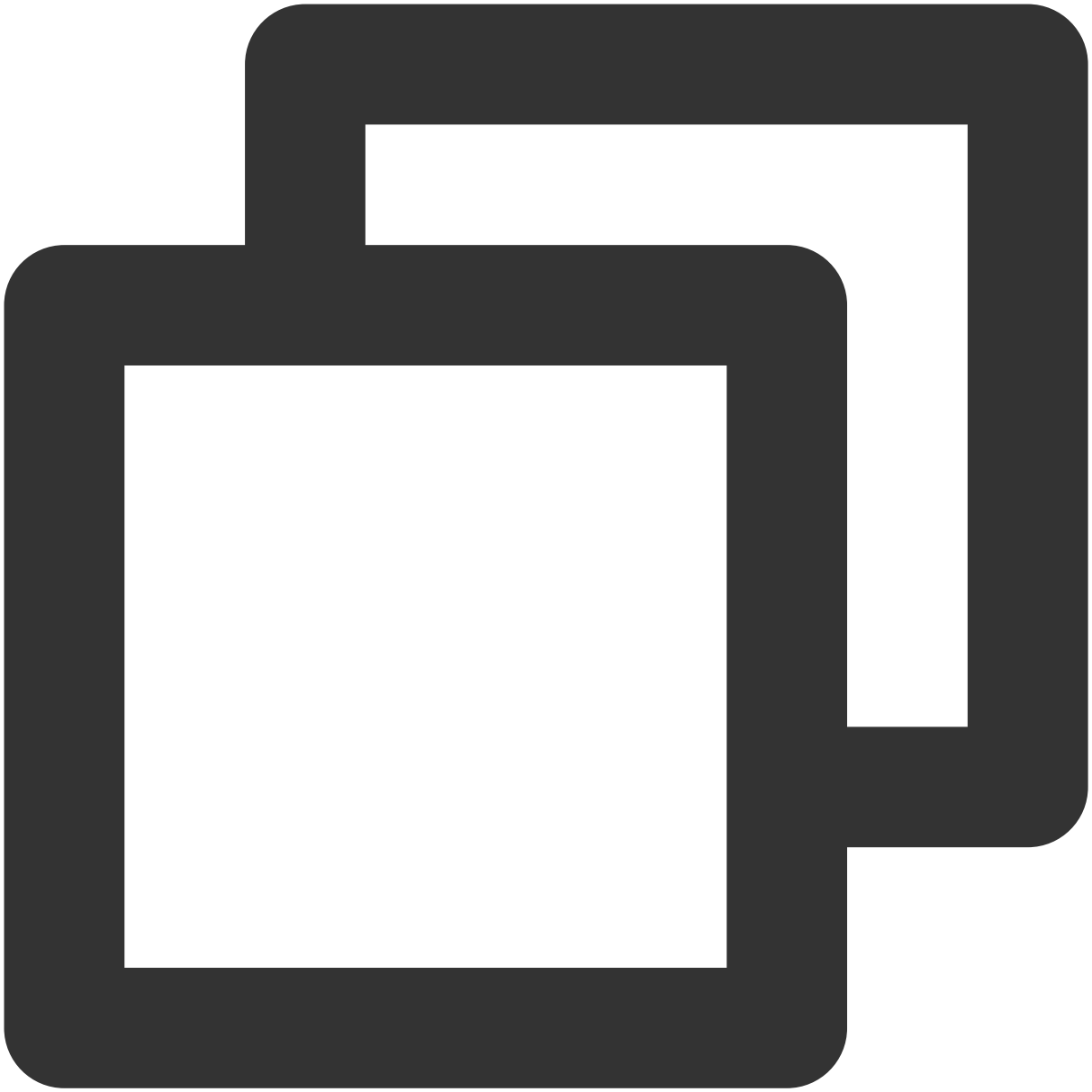
添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。



```
Future<void> addObserver(TUICallObserver observer)
```

removeObserver

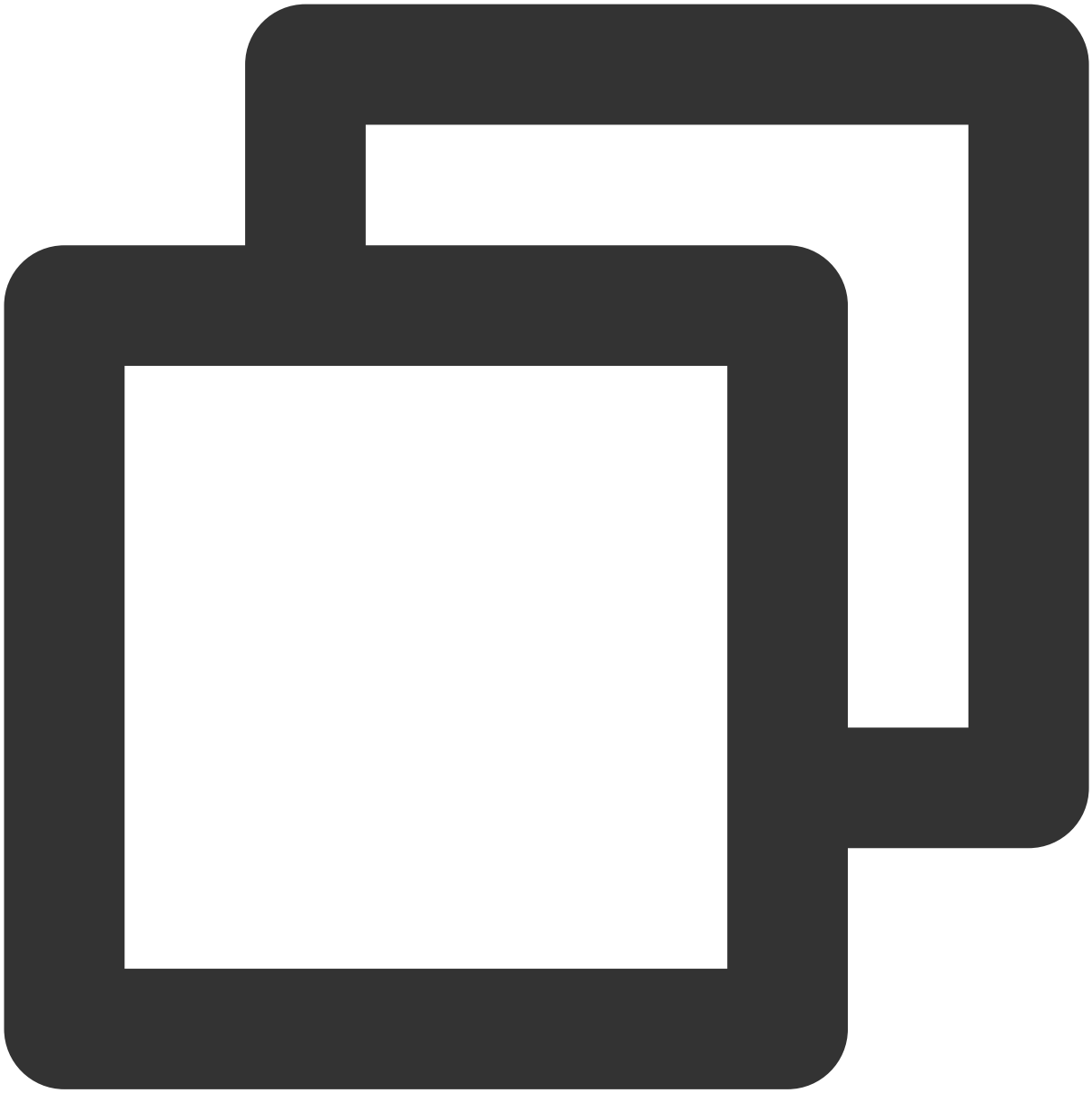
移除回调接口。



```
Future<void> removeObserver(TUICallObserver observer)
```

call

拨打电话（1v1通话）。



```
Future<TUIResult> call(String userId, TUICallMediaType mediaType, TUICallParams par
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
mediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推

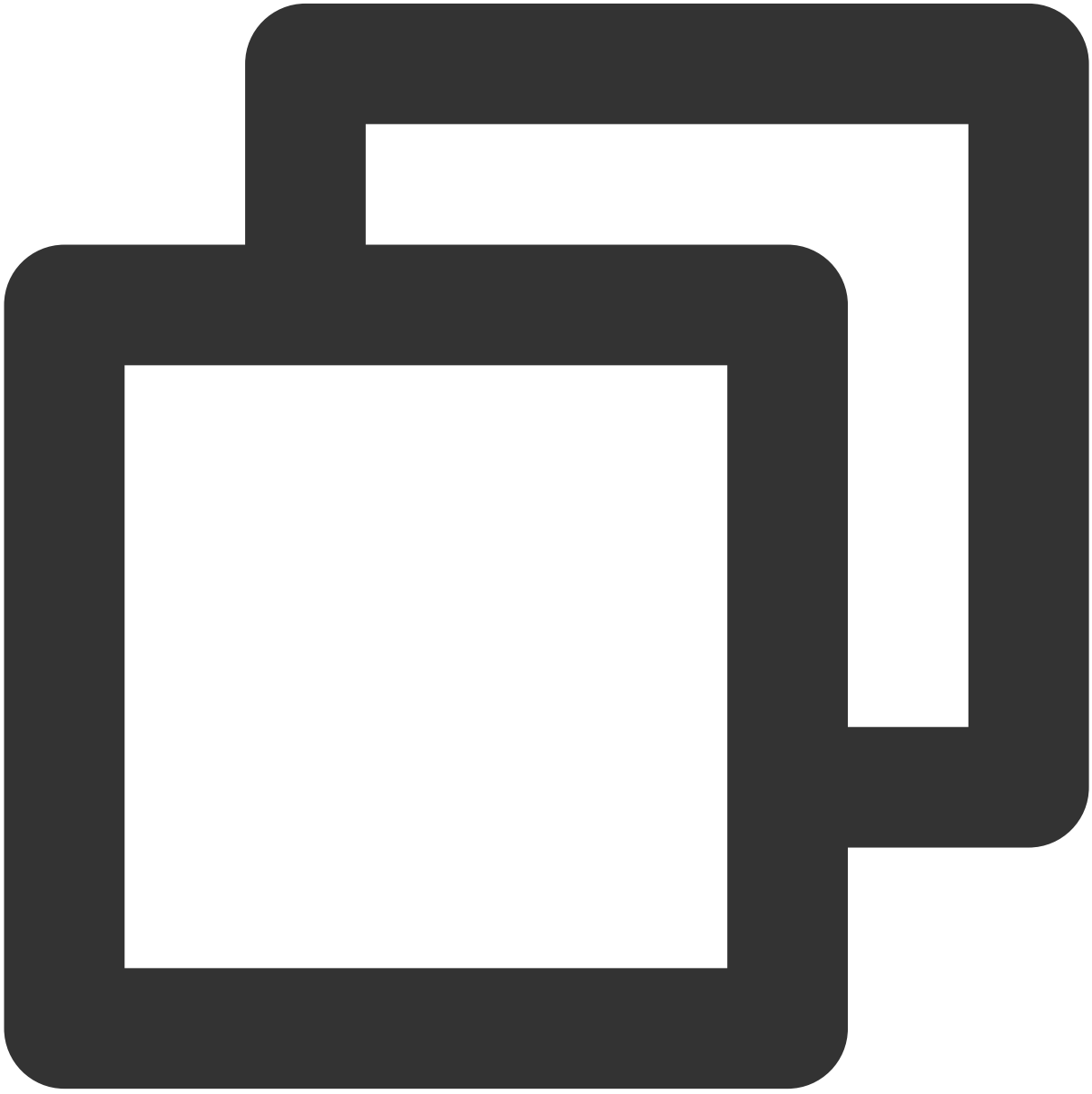
		送自定义内容等

groupCall

发起群组通话。

注意：

使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

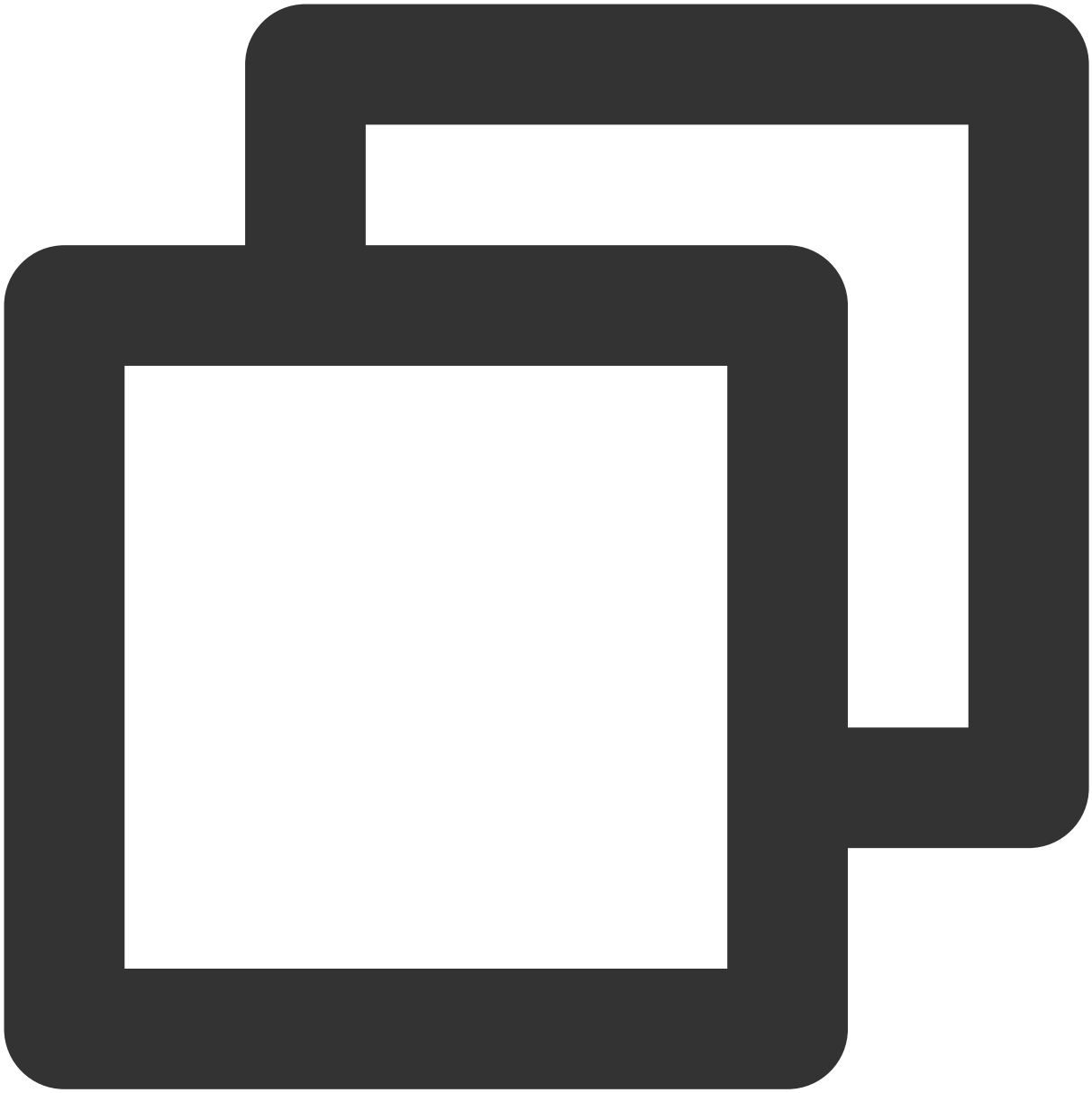


```
Future<TUIResult> groupCall(String groupId, List<String> userIdList, TUICallMediaTy
```


参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List<String>	目标用户的 userId 列表
mediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等

accept

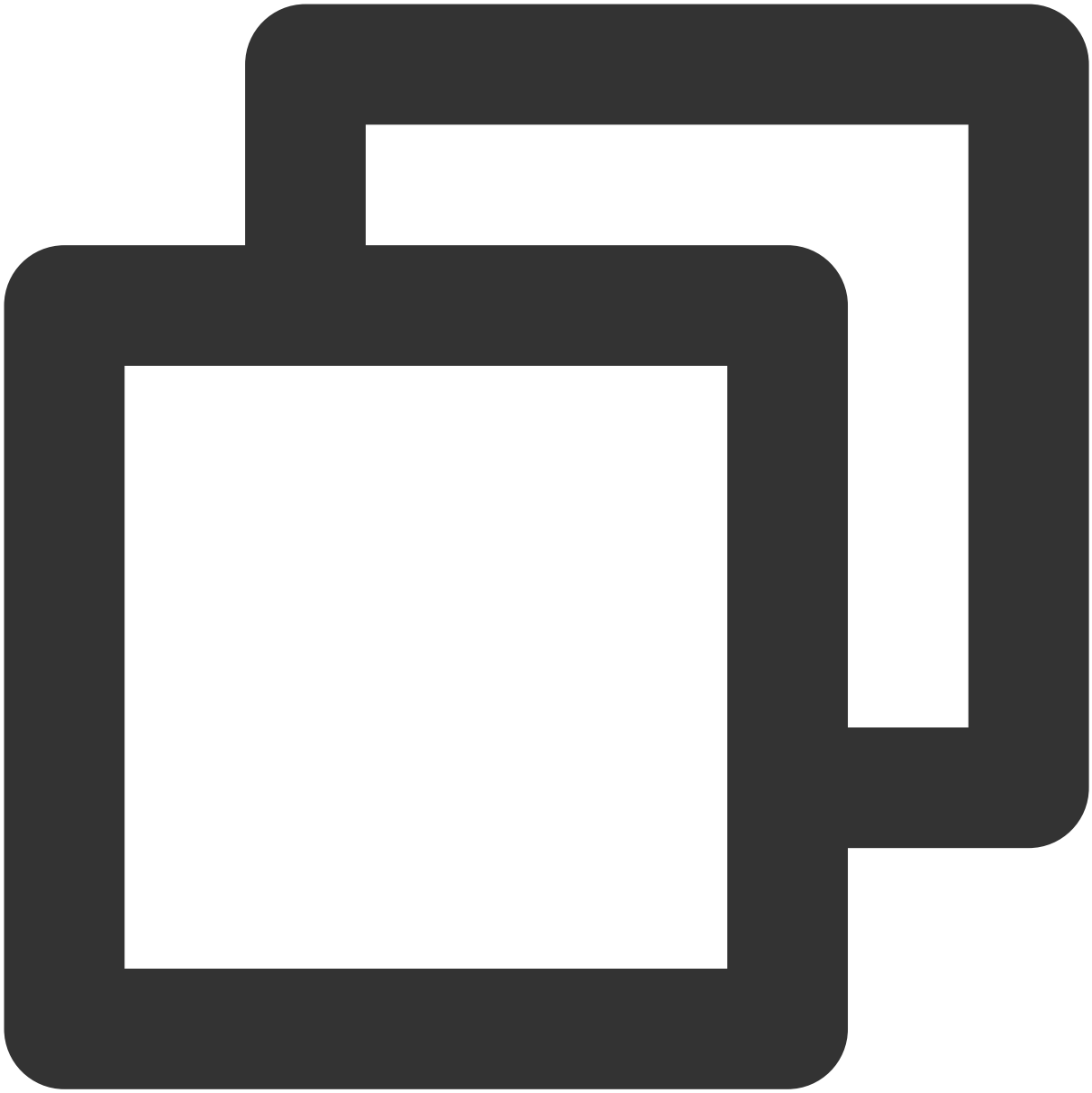
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
Future<TUIResult> accept()
```

reject

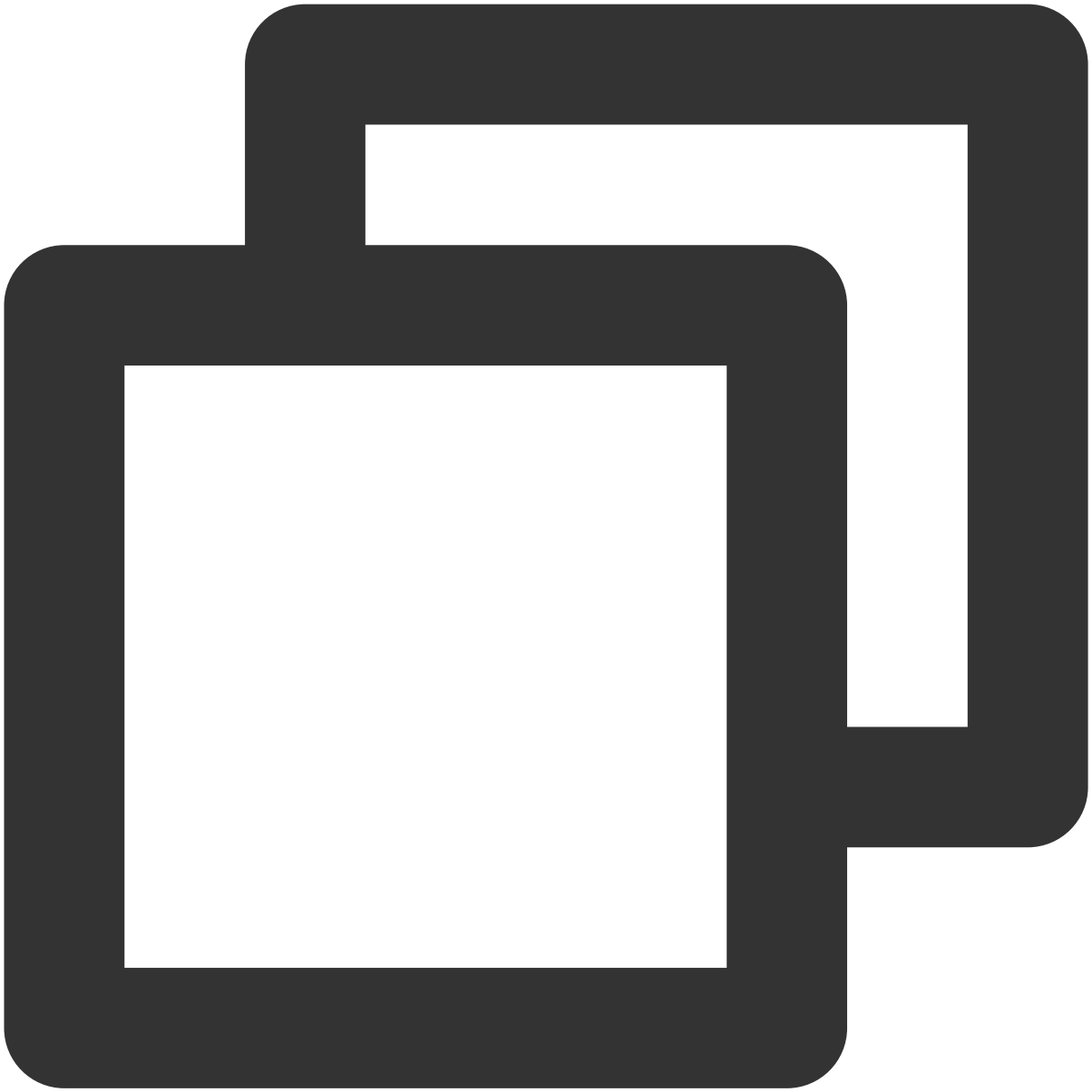
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



```
Future<TUIResult> reject()
```

ignore

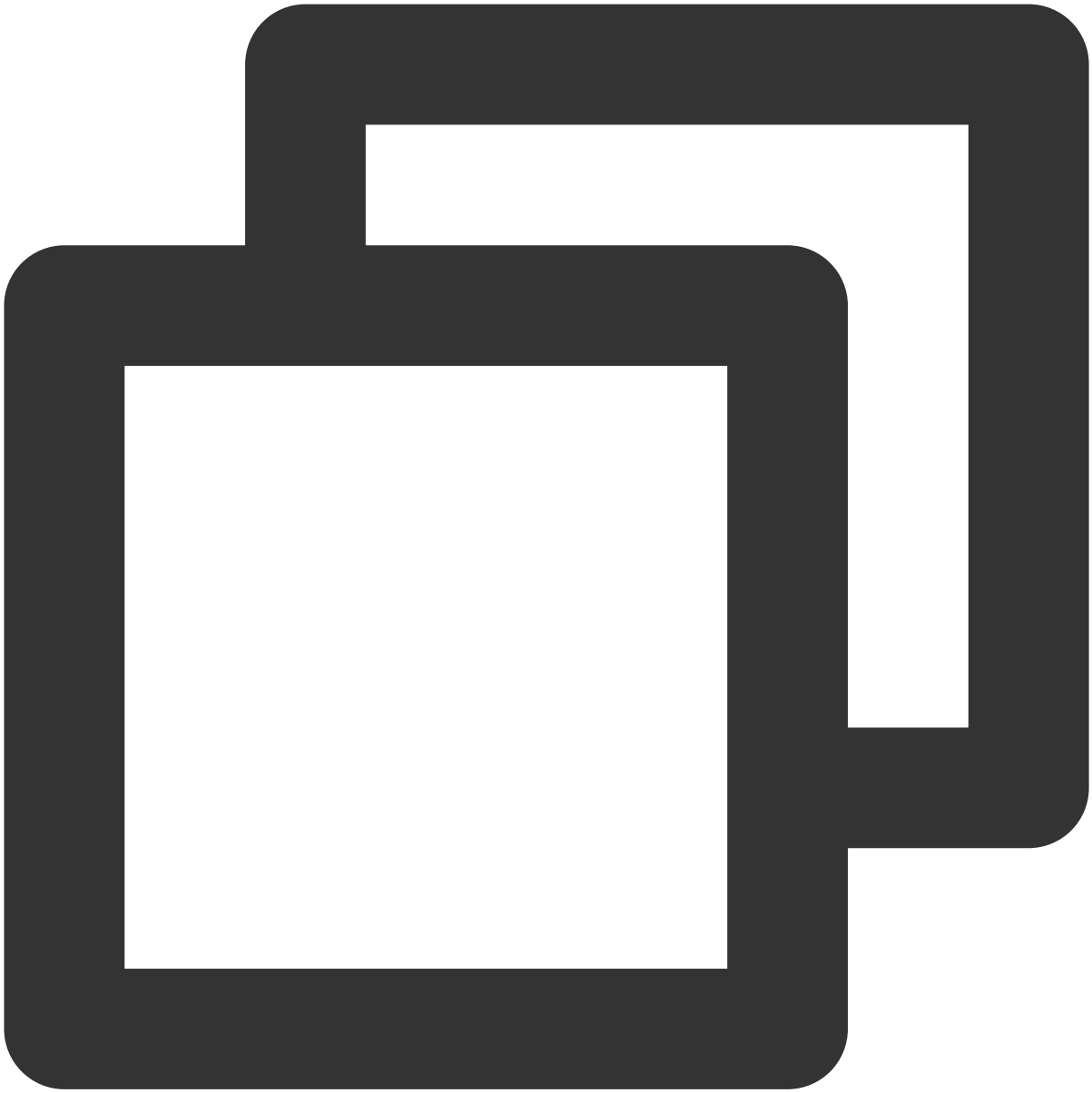
忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
Future<TUIResult> ignore()
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

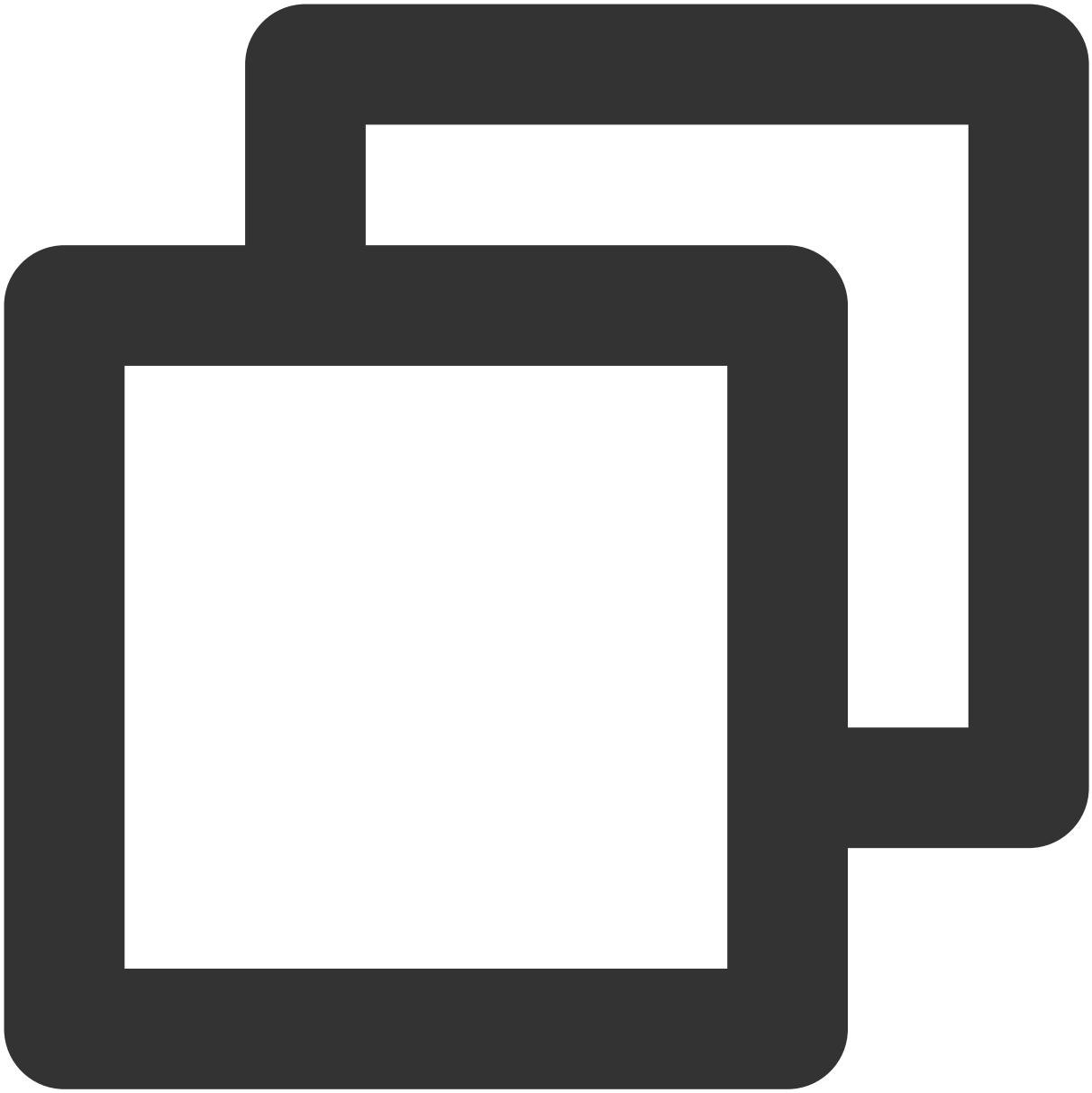


```
Future<TUIResult> hangup()
```

inviteUser

邀请用户加入此次群组通话。

使用场景：一个群组通话中的用户主动邀请其他人时使用。



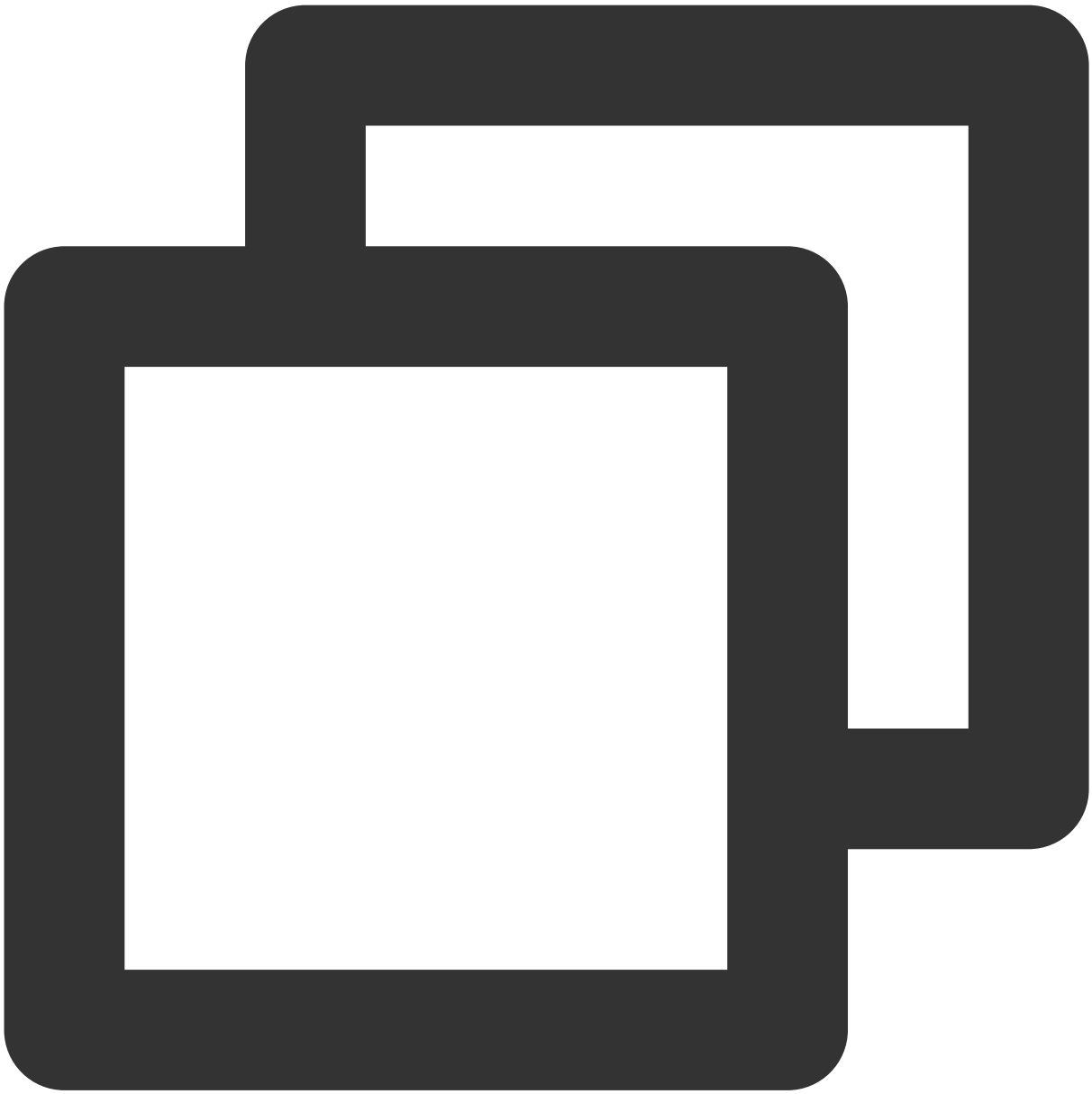
```
Future<void> iniviteUser(List<String> userIdList, TUICallParams params, TUIValueCal
```

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表
params	TUICallParams	通话参数扩展字段，例如：离线推送自定义内容

joinInGroupCall

主动加入此次群组通话。

使用场景：群组内用户主动加入此次群组通话使用。



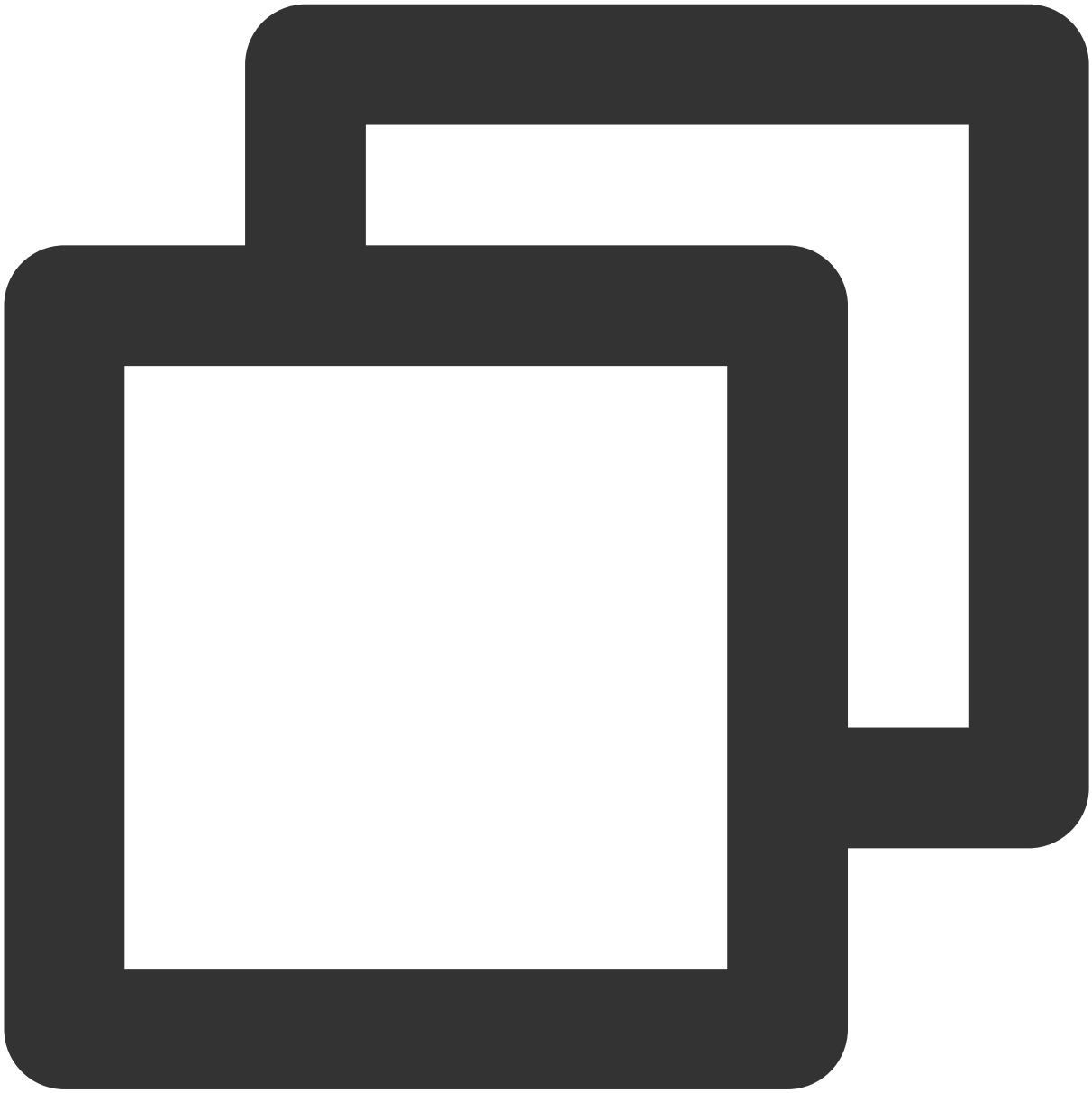
```
Future<TUIResult> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaTyp
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID
groupId	String	此次群组通话的群 ID

mediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

switchCallMediaType

切换视频通话到语音通话。

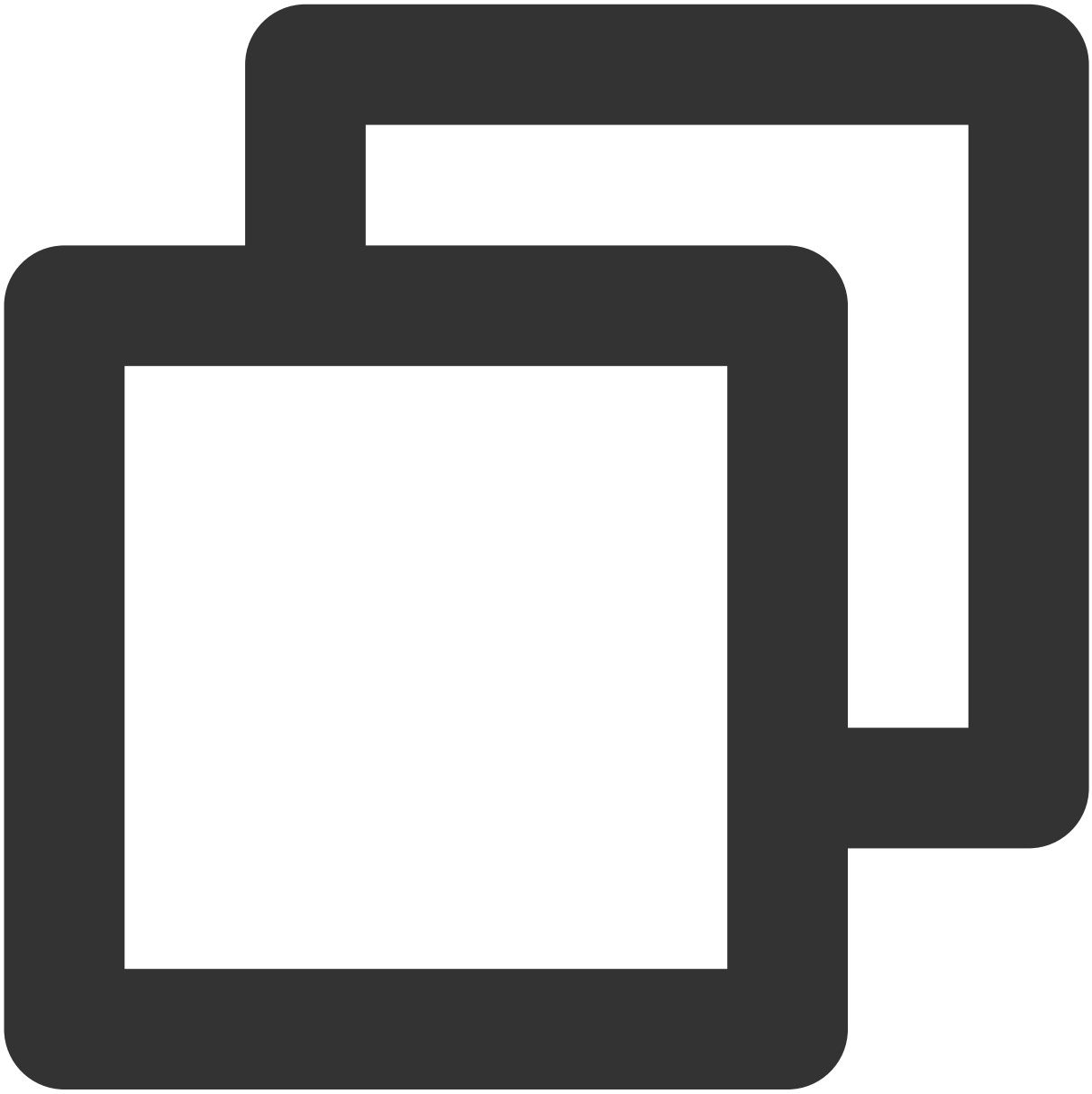


```
Future<void> switchCallMediaType(TUICallMediaType mediaType)
```

参数	类型	含义
mediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

startRemoteView

设置显示视频画面的 View 对象。

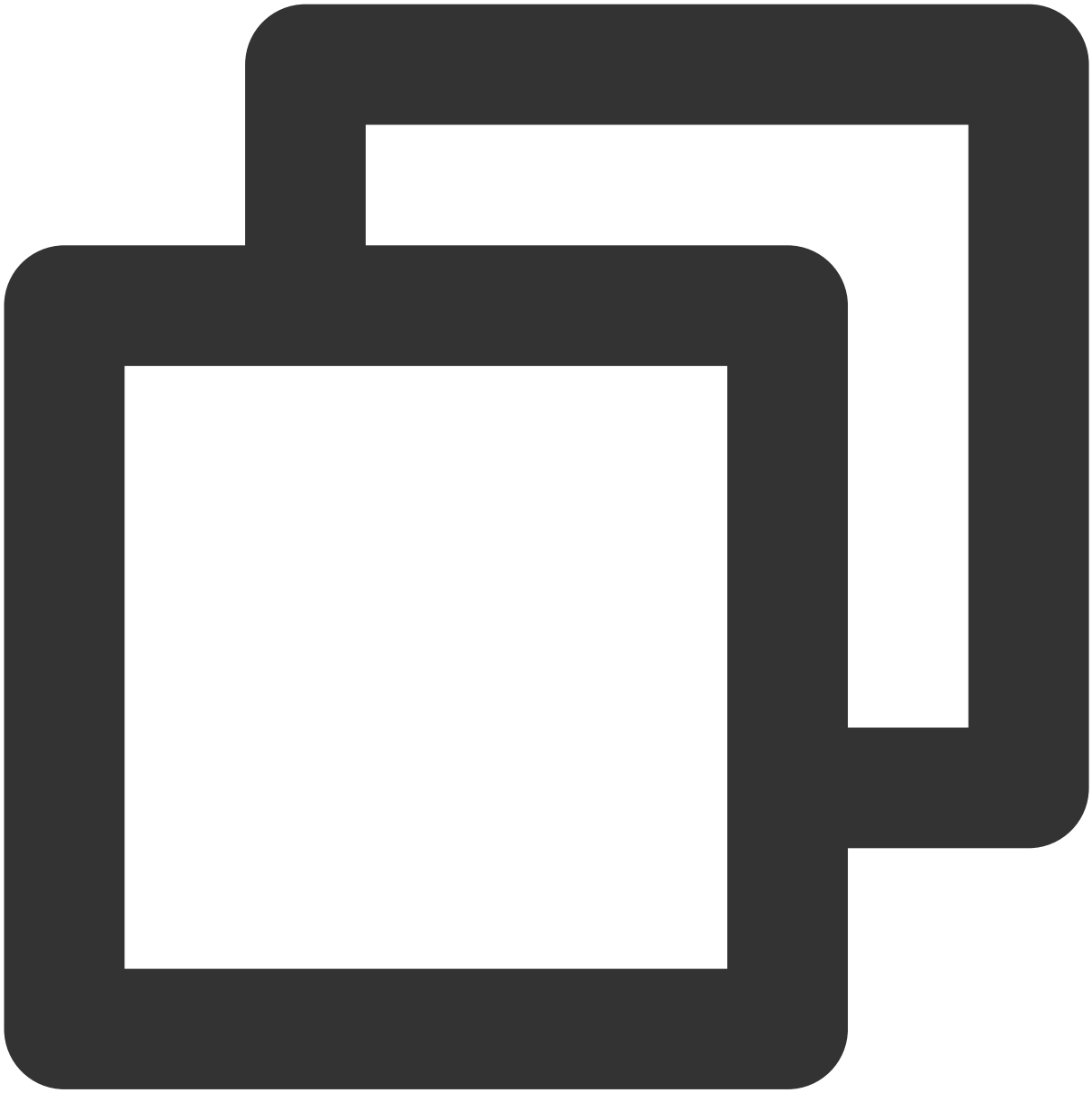


```
Future<void> startRemoteView(String userId, intviewId)
```

参数	类型	含义
userId	String	目标用户的 userId
intviewId	int？	显示视频画面的widget对应的Id

stopRemoteview

停止订阅远端用户的视频数据。

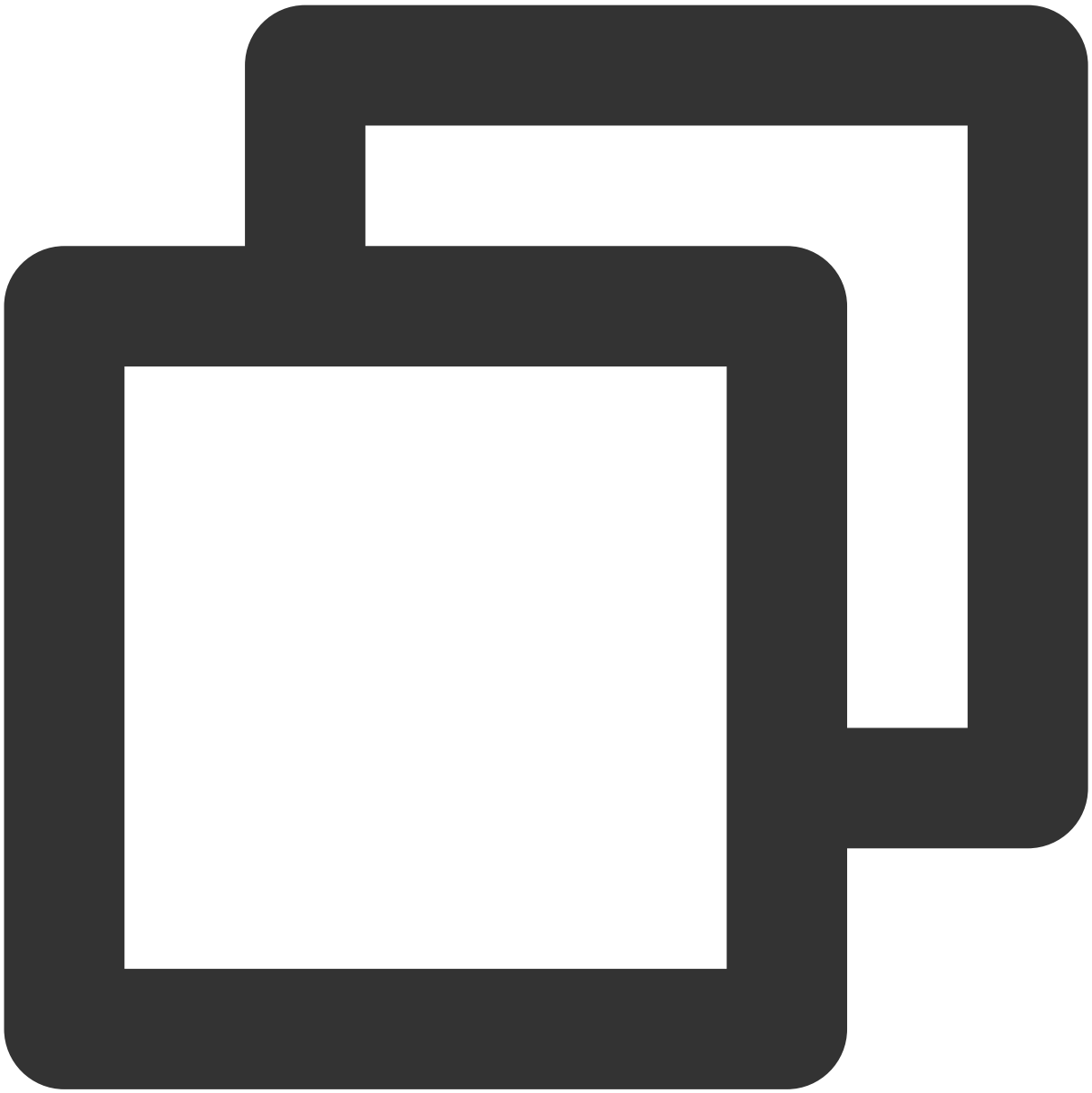


```
Future<void> stopRemoteView(String userId)
```

参数	类型	含义
userId	String	目标用户的 userId

openCamera

开启摄像头。

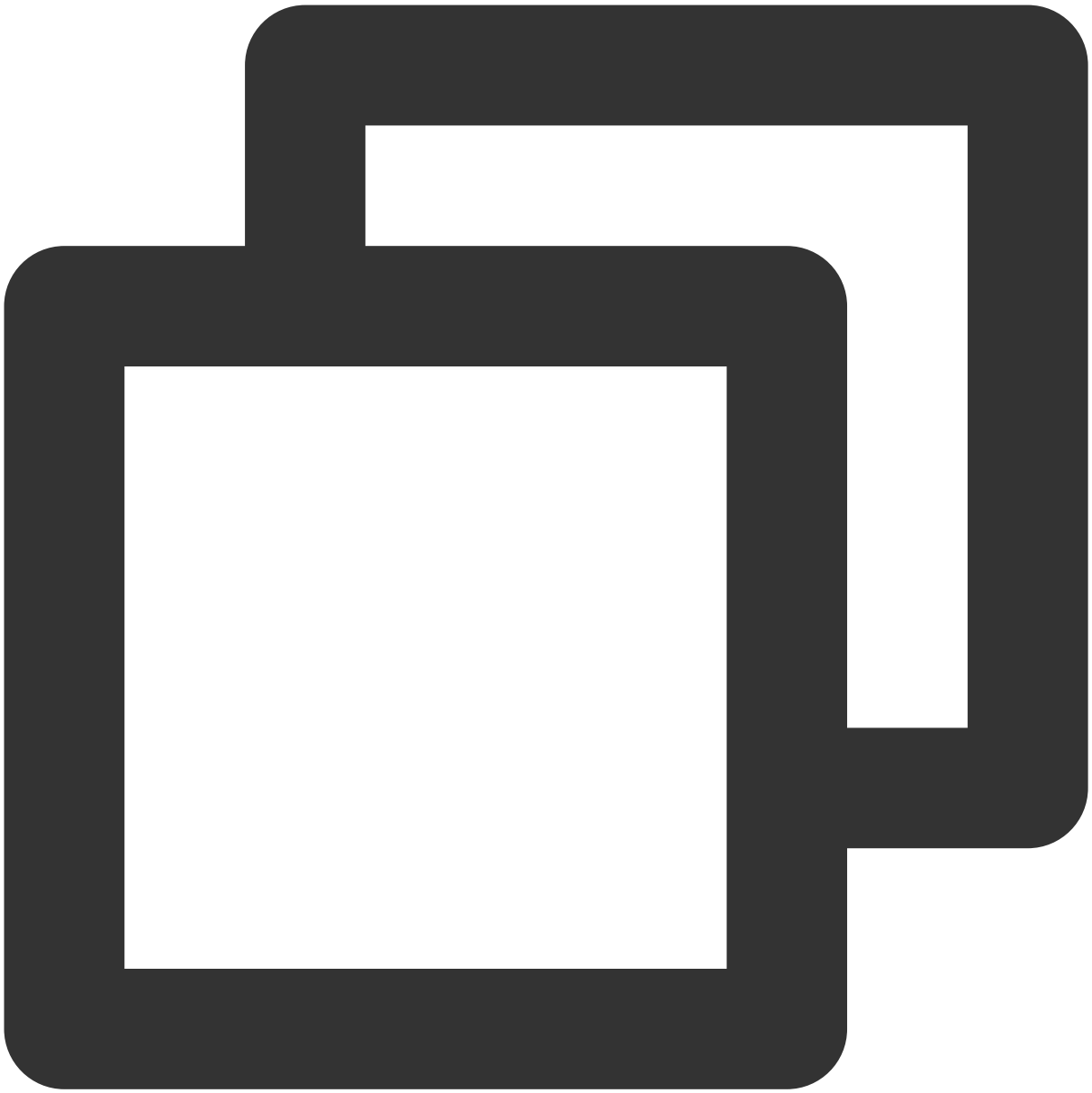


```
Future<TUIResult> openCamera(TUICamera camera, int? viewId)
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头
viewId	int?	显示视频画面的widget对应的Id

closeCamera

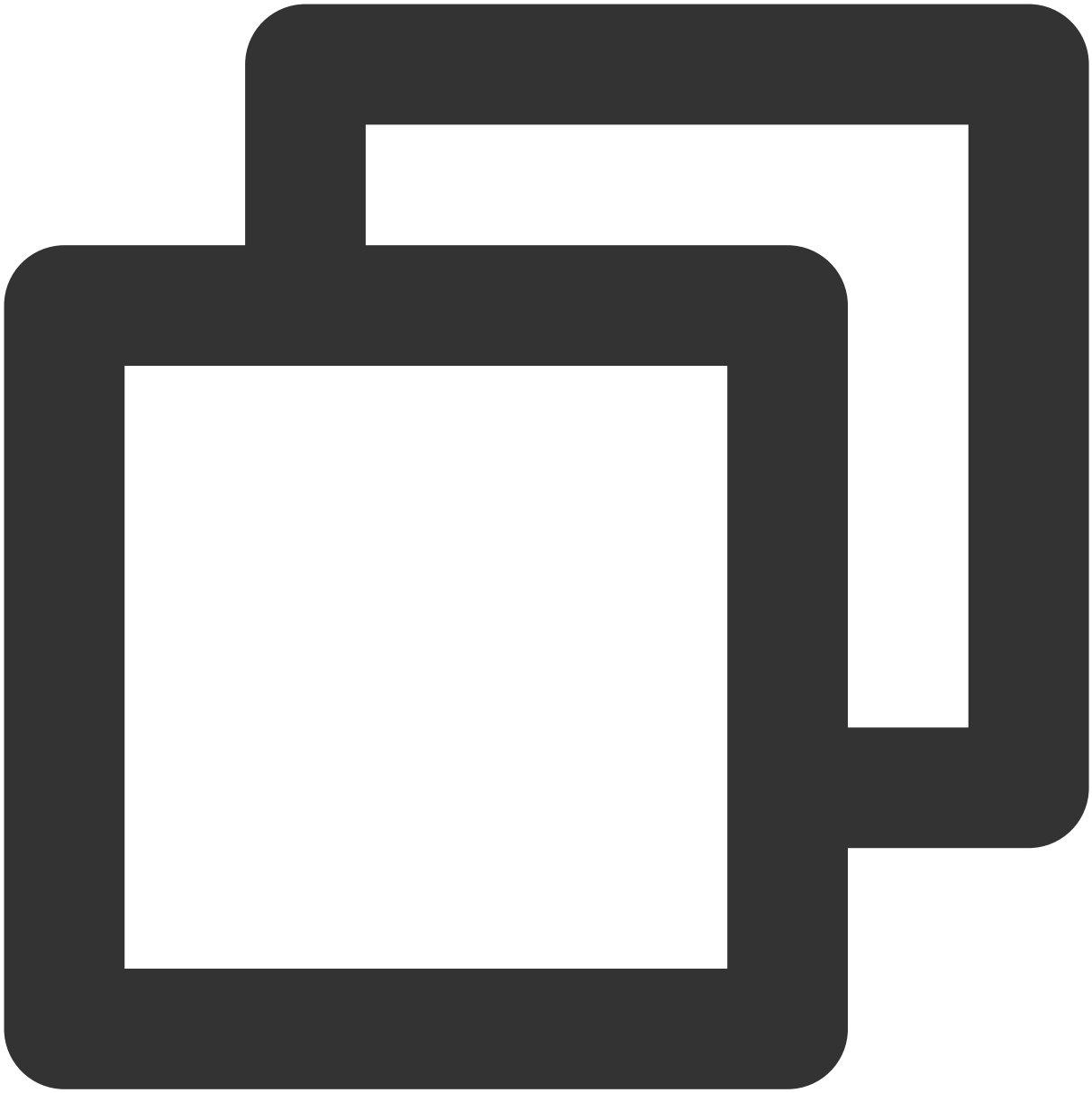
关闭摄像头。



```
Future<void> closeCamera()
```

switchCamera

切换前后摄像头。

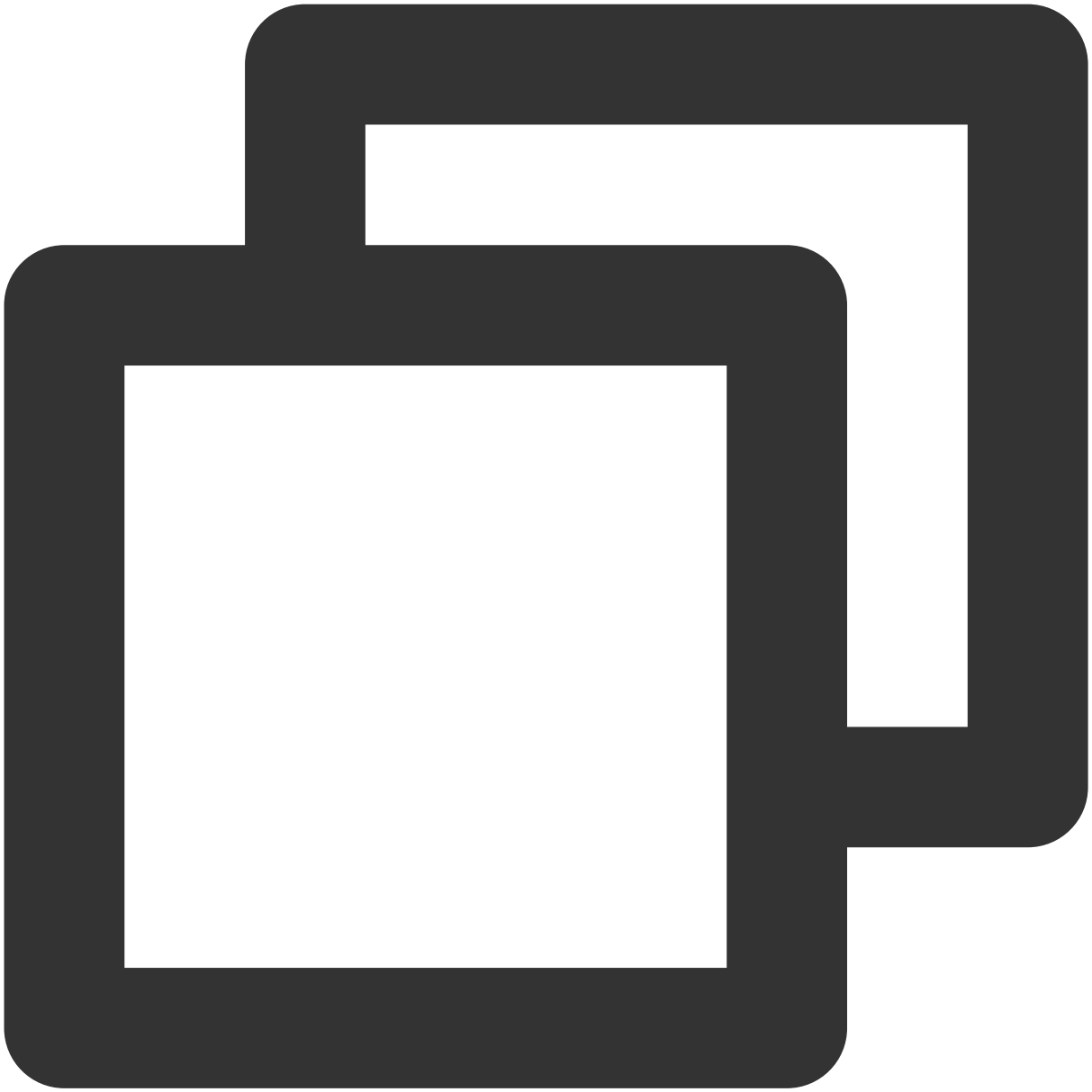


```
Future<void> switchCamera(TUICamera camera)
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头

openMicrophone

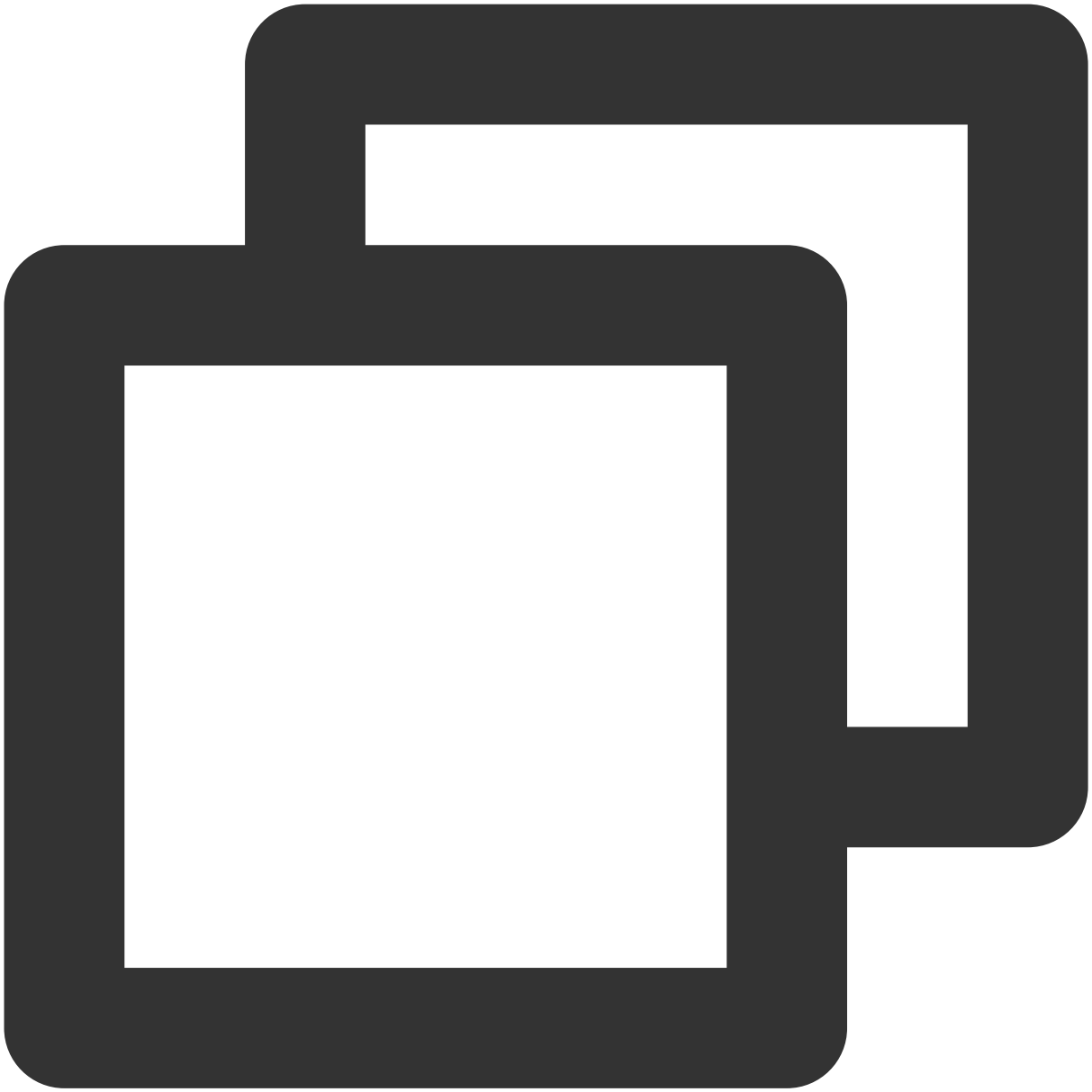
打开麦克风。



```
Future<TUIResult> openMicrophone()
```

closeMicrophone

关闭麦克风。

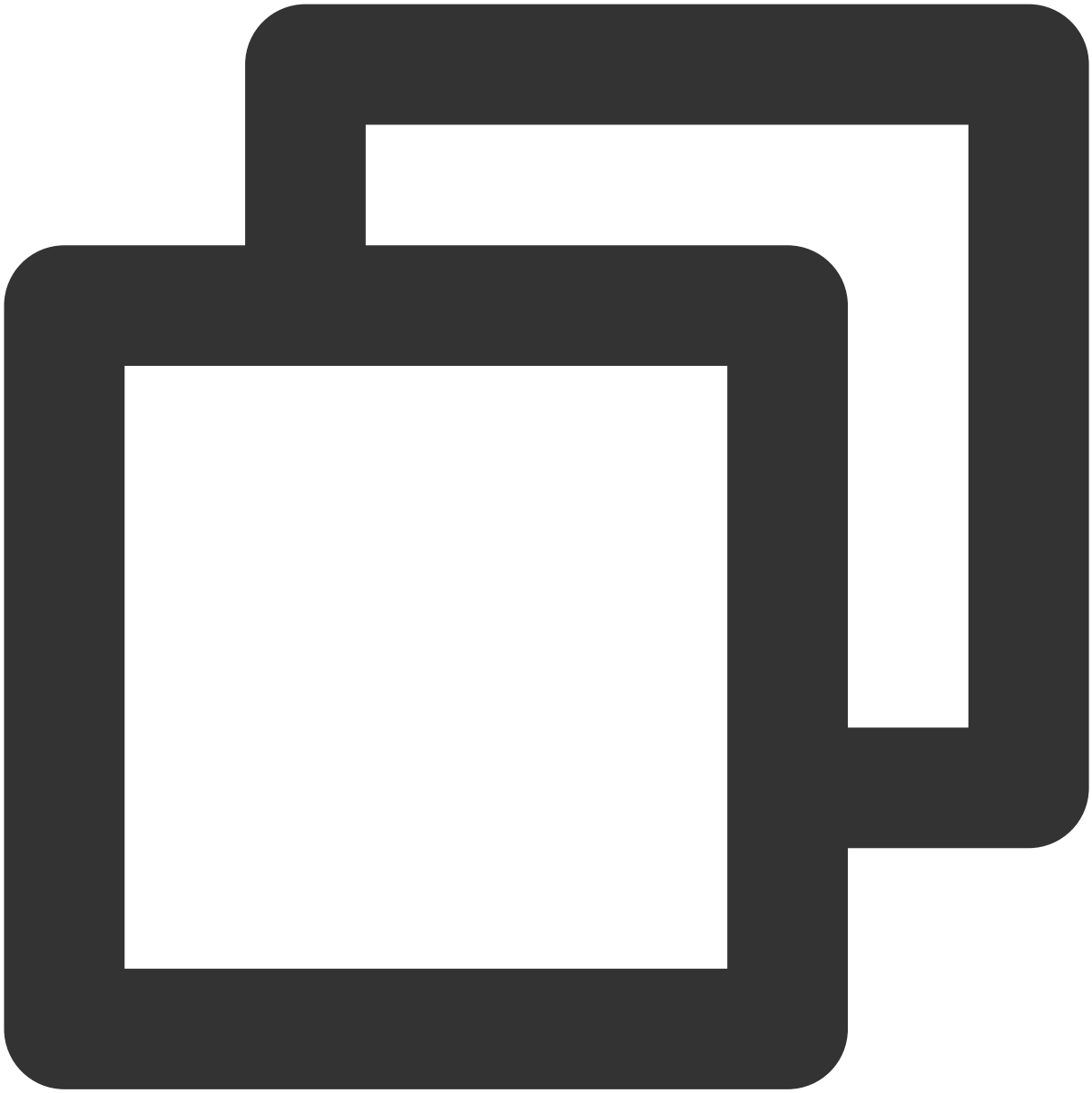


```
Future<void> closeMicrophone()
```

selectAudioPlaybackDevice

选择音频播放设备。

目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。



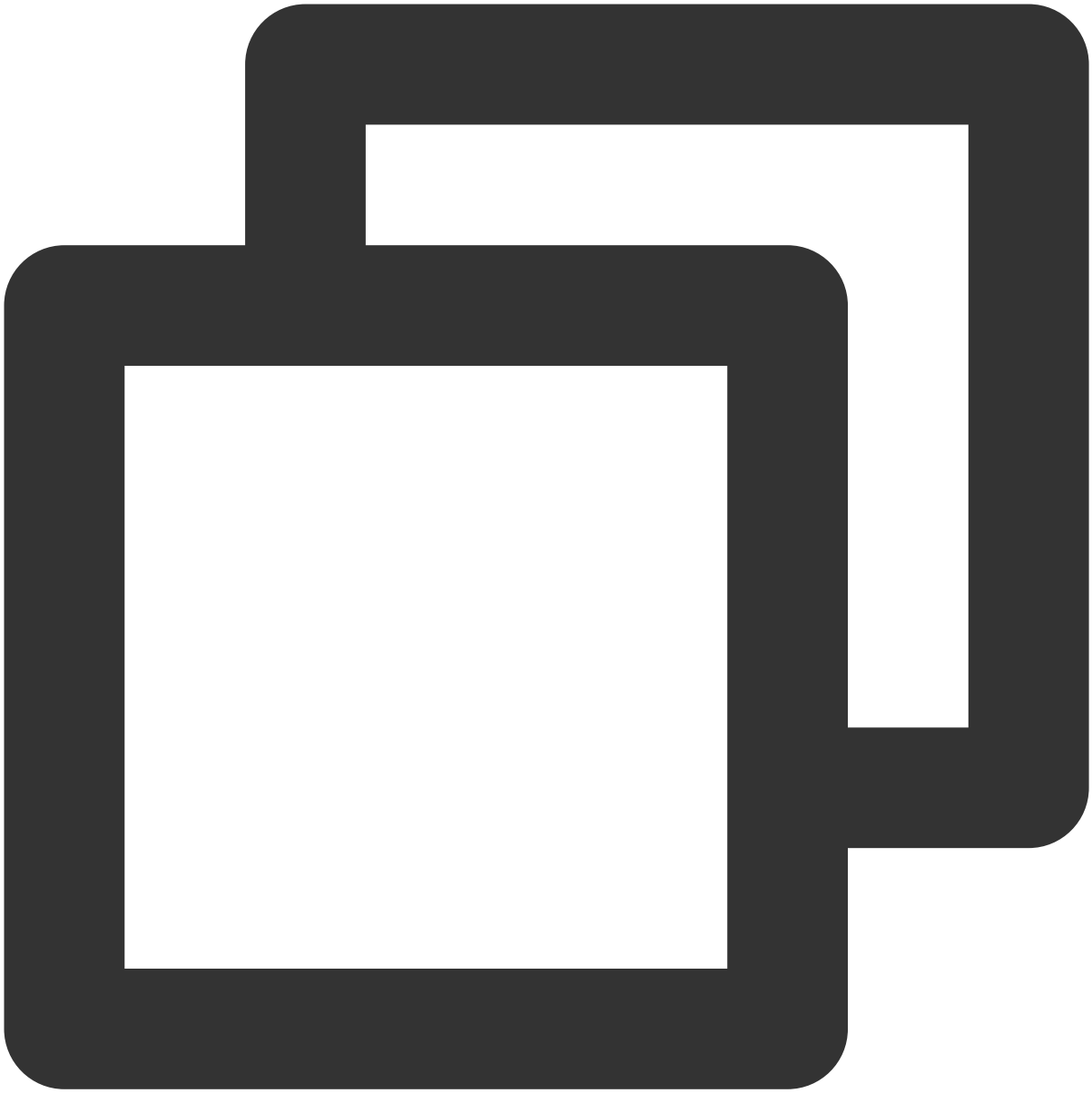
```
Future<void> selectAudioPlaybackDevice(TUIAudioPlaybackDevice device)
```

参数	类型	含义
device	TUIAudioPlaybackDevice	听筒/扬声器

setSelfInfo

设置用户昵称、头像。

用户昵称不能超过500字节，用户头像必须是 URL 格式。

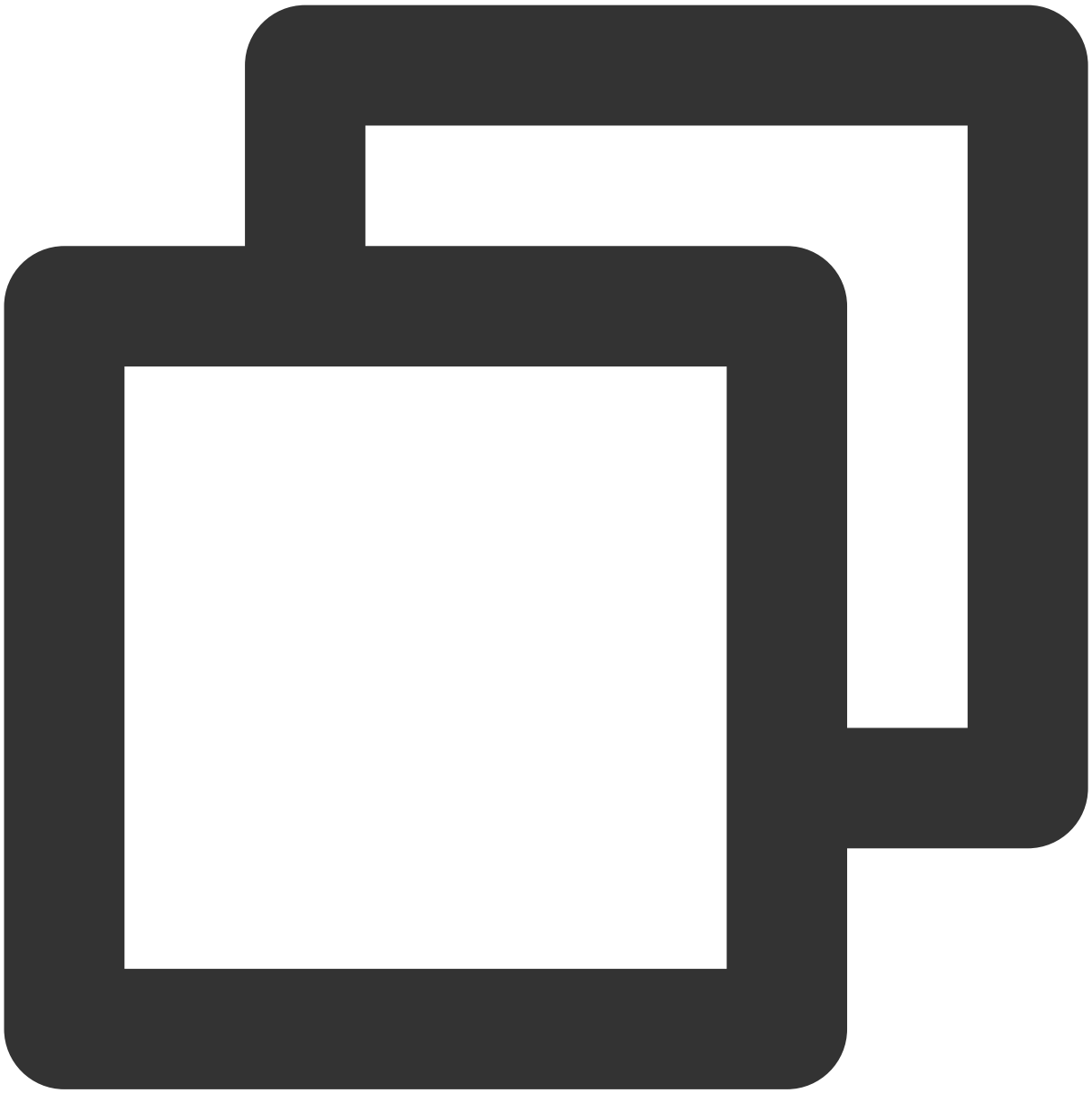


```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

参数	类型	含义
nickname	String	用户昵称
avatar	String	用户头像（格式为 URL）

enableMultiDeviceAbility

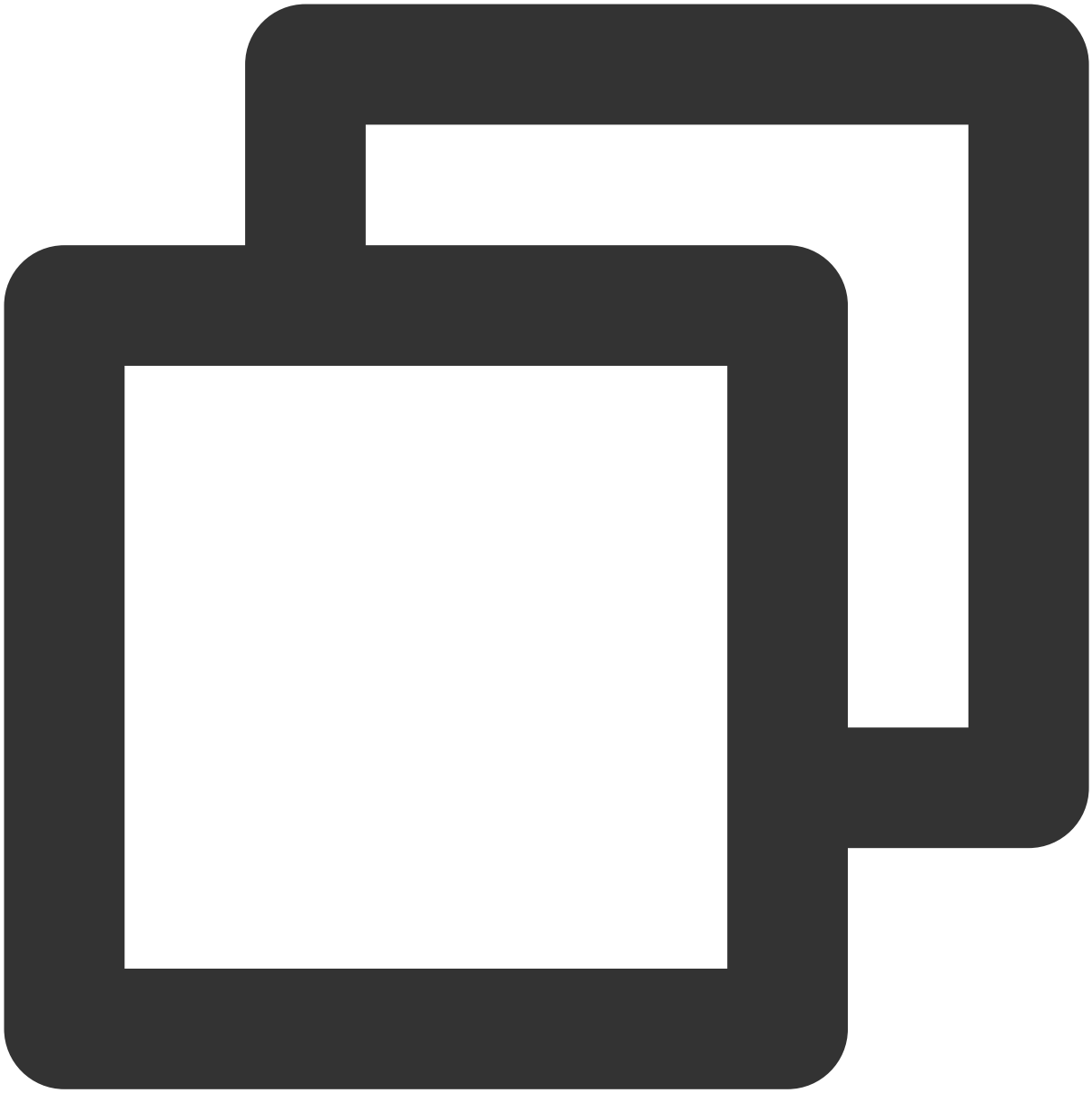
开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）。



```
Future<TUIResult> enableMultiDeviceAbility(bool enable)
```

setVideoRenderParams

设置用户视频画面的渲染模式。



```
Future<TUIResult> setVideoRenderParams(String userId, VideoRenderParams params)
```

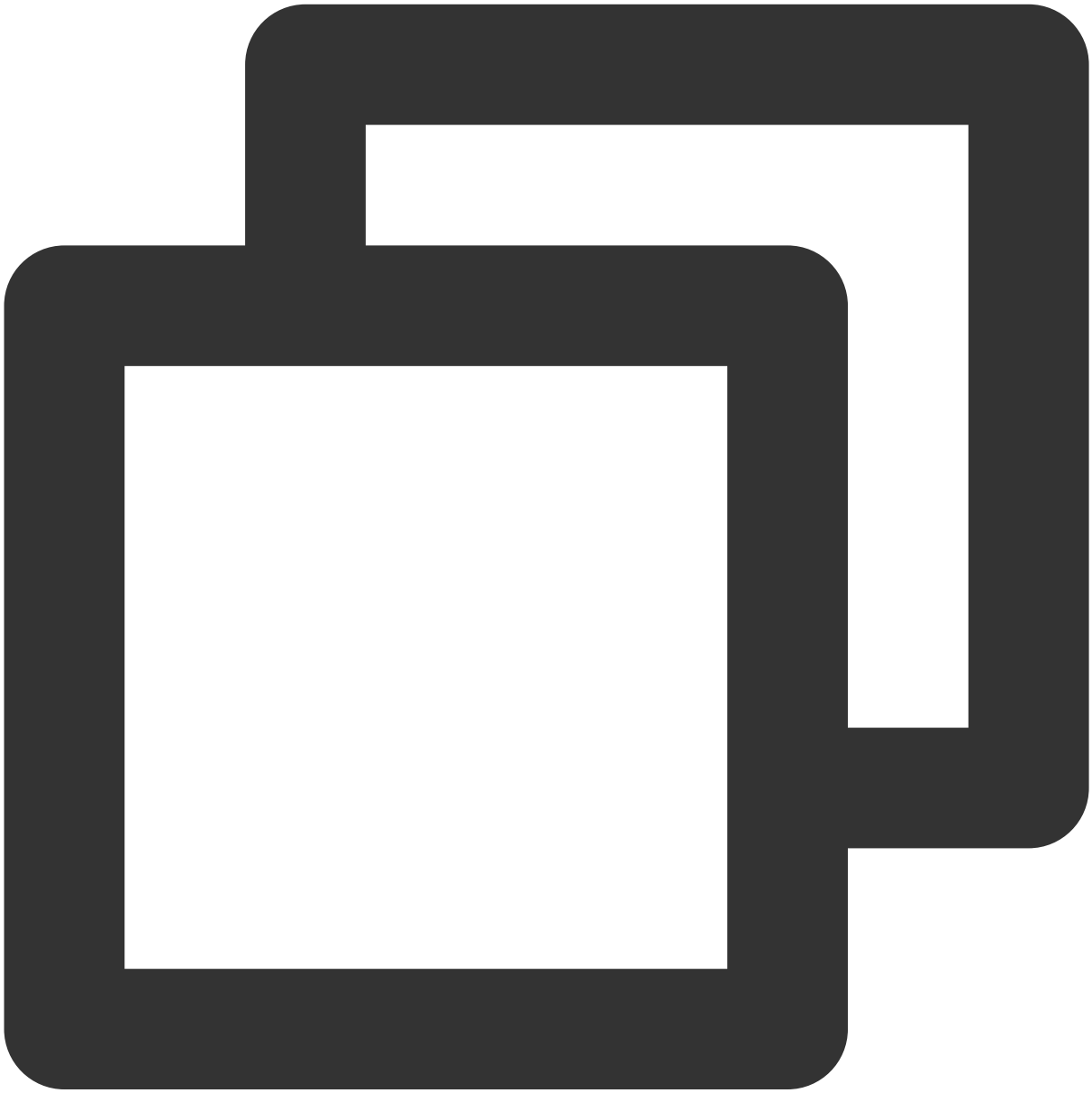
参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
params	VideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式

setVideoEncoderParams

设置视频编码的编码参数。

该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。



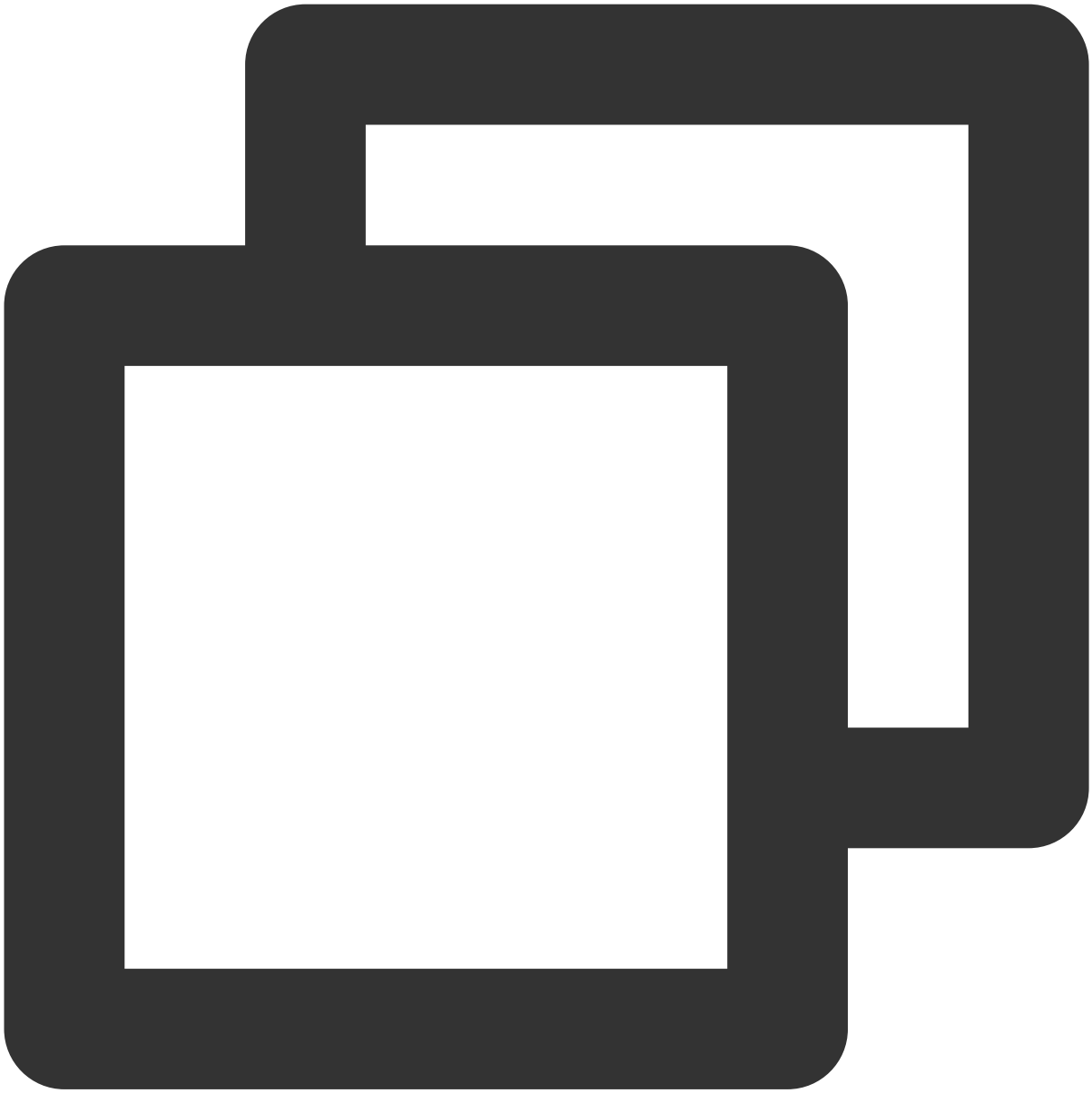
```
Future<TUIResult> setVideoEncoderParams(VideoEncoderParams params)
```

参数如下表所示：

参数	类型	含义
params	VideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高比

queryRecentCalls

查询通话记录。



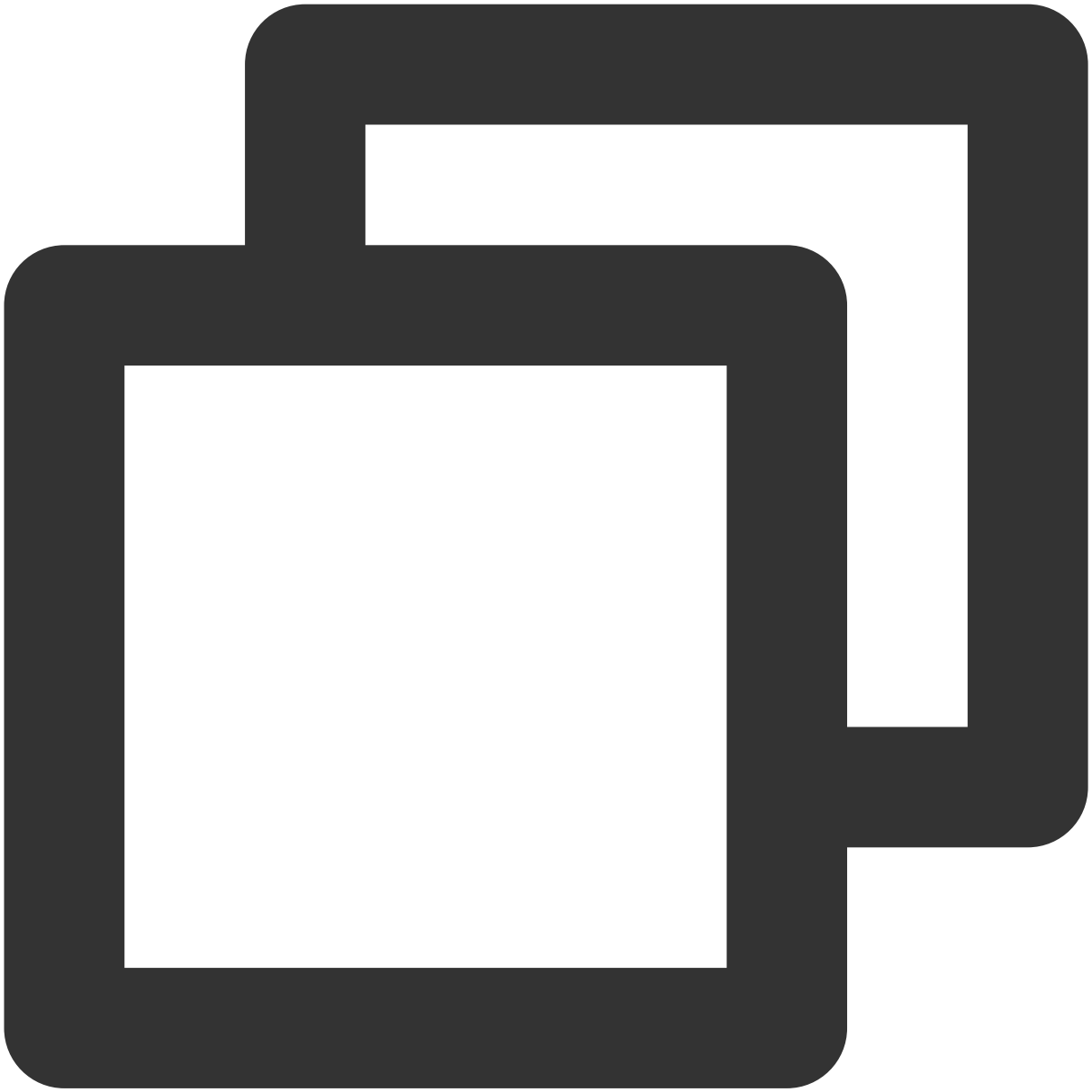
```
Future<void> queryRecentCalls(TUICallRecentCallsFilter filter, TUIValueCallback cal
```

参数如下表所示：

参数	类型	含义
filter	TUICallRecentCallsFilter	查询过滤条件

deleteRecordCalls

删除通话记录。



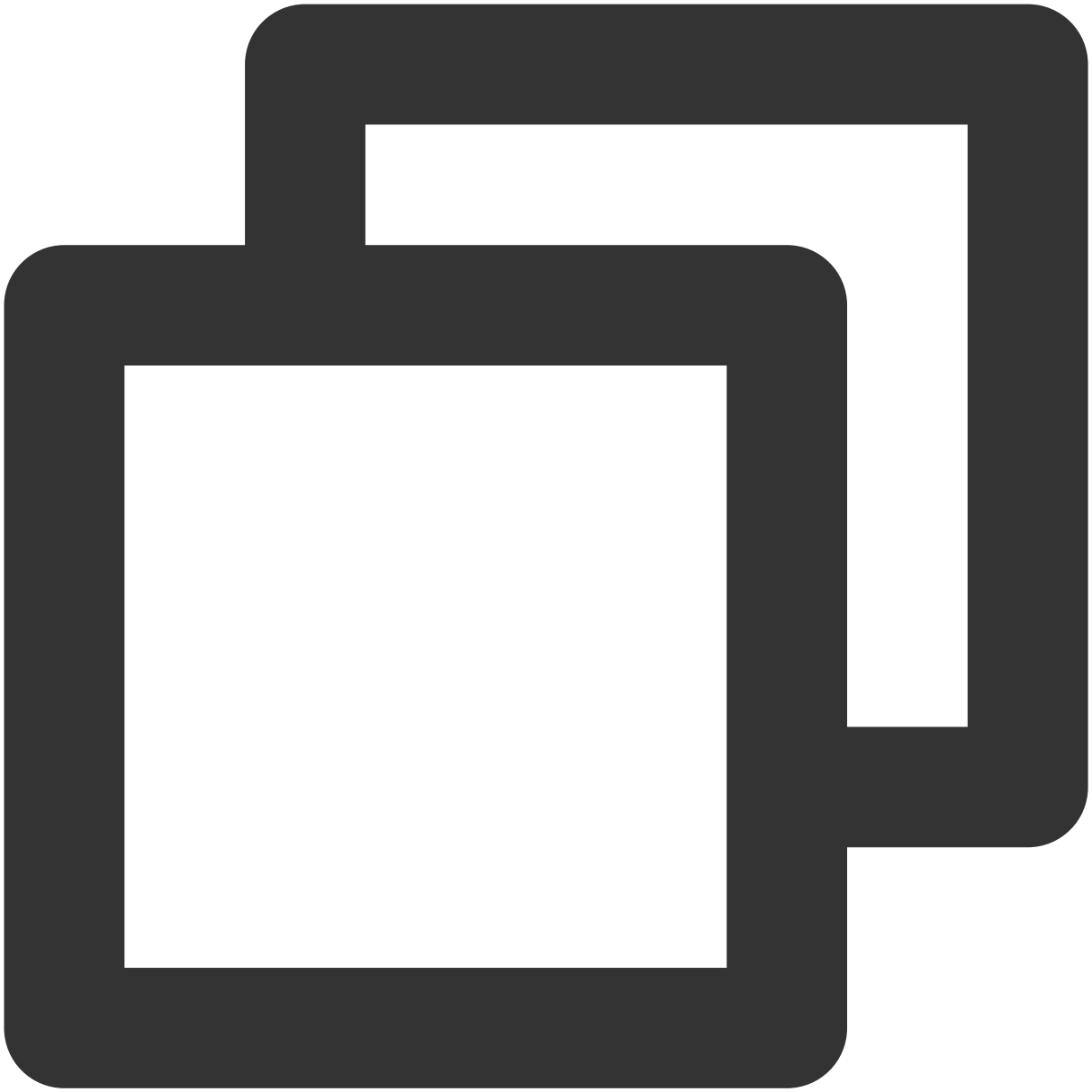
```
Future<void> deleteRecordCalls(List<String> callIdList, TUIValueCallback callback)
```

参数如下表所示：

参数	类型	含义
callIdList	List<String>	待删除记录的Id列表

setBeautyLevel

设置美颜级别。



```
Future<TUIResult> setBeautyLevel(double level)
```

参数如下表所示：

参数	类型	含义
level	double	美颜等级，使用范围0.0～10.0

TUICallObserver

最近更新时间：2024-01-17 11:58:53

TUICallObserver API 简介

TUICallObserver 是 TUICallKit 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

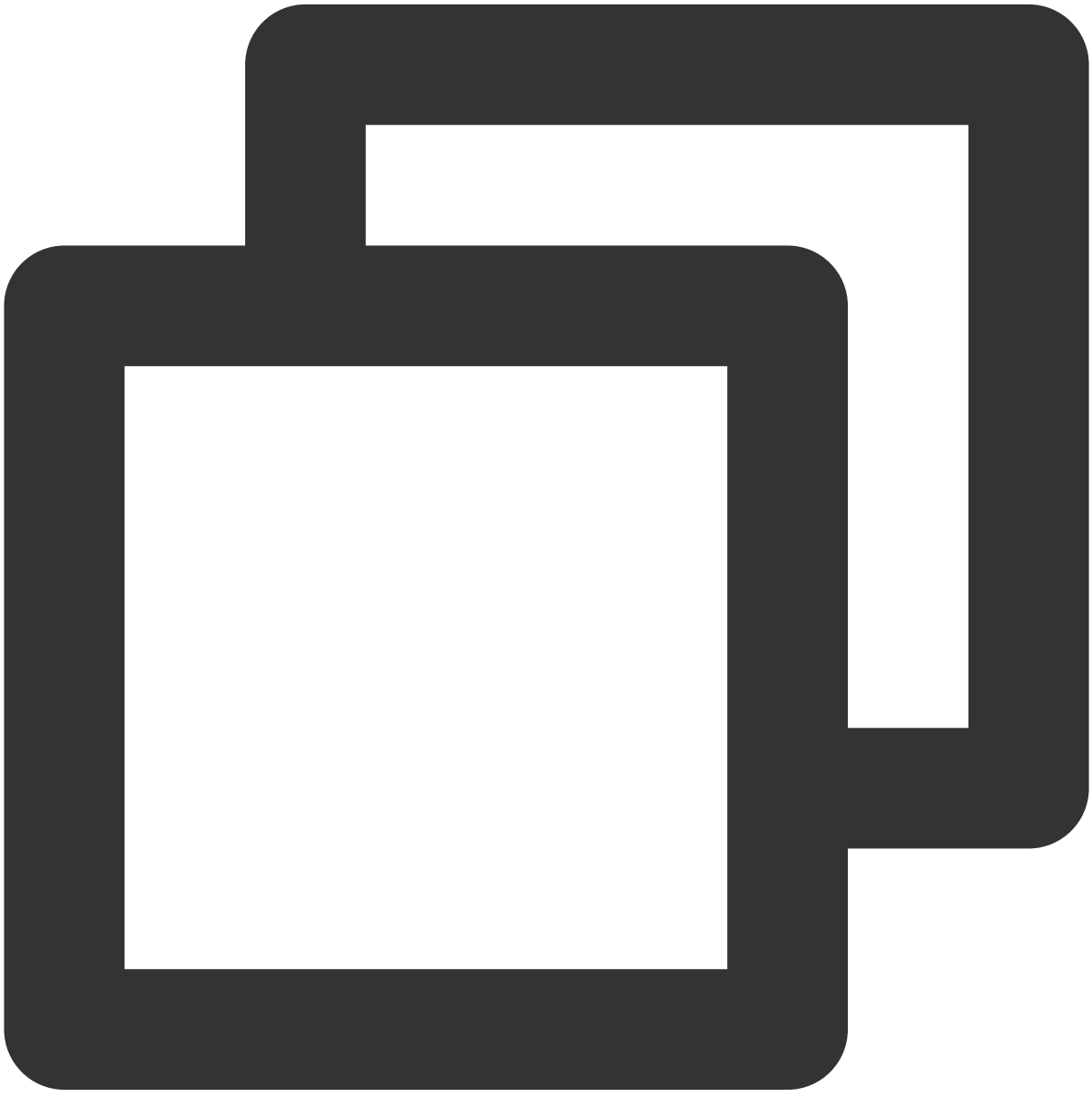
API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线

版权所有：腾讯云计算（北京）有限责任公司

onUserSigExpired	在线时票据过期

回调事件详情

通过 addObserver 监听Flutter插件抛出的事件。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {
      //您的回调处理逻辑
```

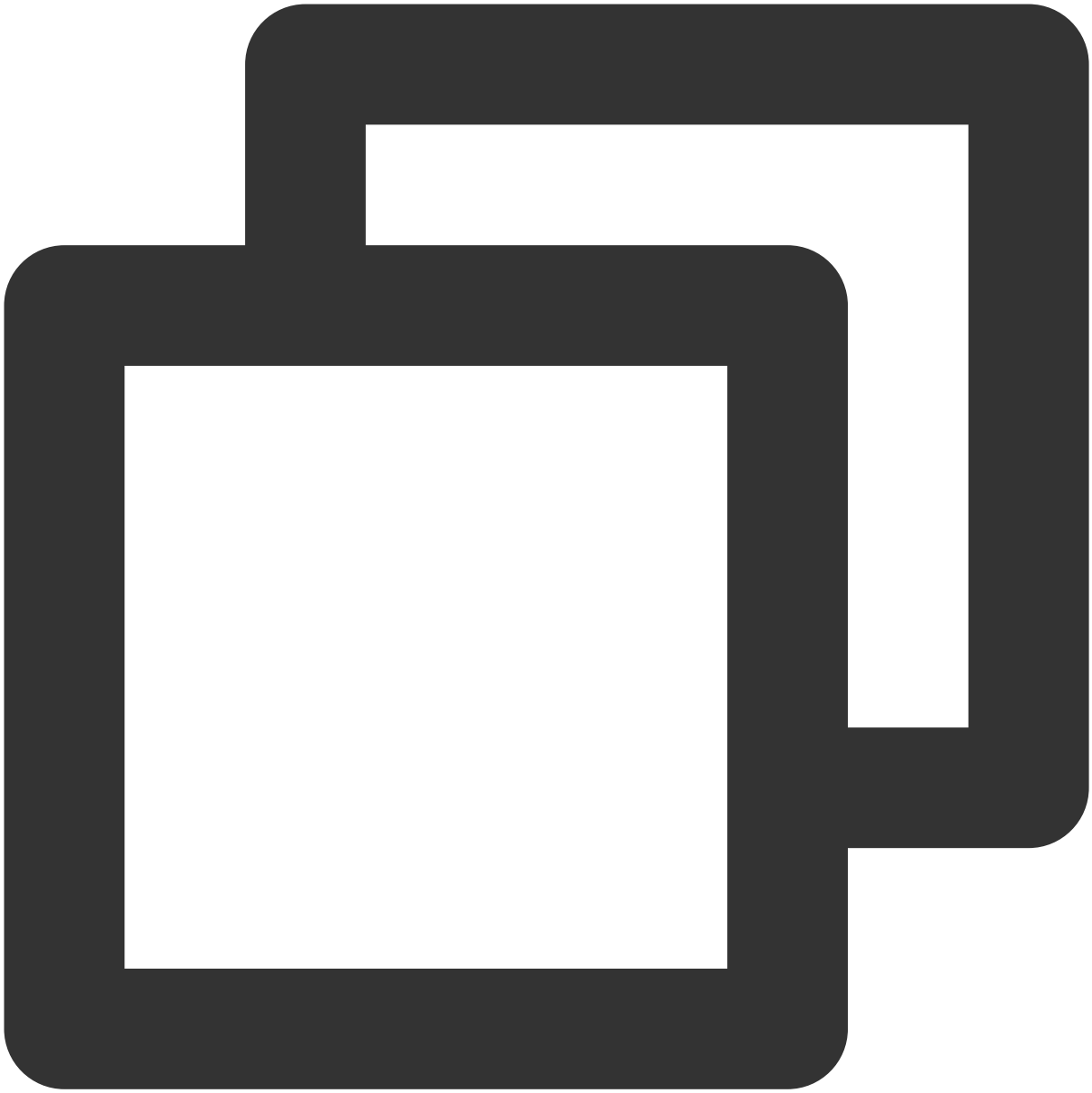
```
    },onCallCancelled: (String callerId) {
      //您的回调处理逻辑
    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole
      //您的回调处理逻辑
    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca
      //您的回调处理逻辑
    }, onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType
      //您的回调处理逻辑
    }, onUserReject: (String userId) {
      //您的回调处理逻辑
    }, onUserNoResponse: (String userId) {
      //您的回调处理逻辑
    }, onUserLineBusy: (String onUserLineBusy) {
      //您的回调处理逻辑
    }, onUserJoin: (String userId) {
      //您的回调处理逻辑
    }, onUserLeave: (String userId) {
      //您的回调处理逻辑
    }, onUserVideoAvailable: (String userId, bool isVideoAvailable) {
      //您的回调处理逻辑
    }, onUserAudioAvailable: (String userId, bool isAudioAvailable) {
      //您的回调处理逻辑
    }, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList
      //您的回调处理逻辑
    }, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,
      //您的回调处理逻辑
    }, onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {
      //您的回调处理逻辑
    }, onKickedOffline: () {
      //您的回调处理逻辑
    }, onUserSigExpired: () {
      //您的回调处理逻辑
    }
));
```

onError

错误事件回调。

说明

SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {
      //您的回调处理逻辑
    }
));
```

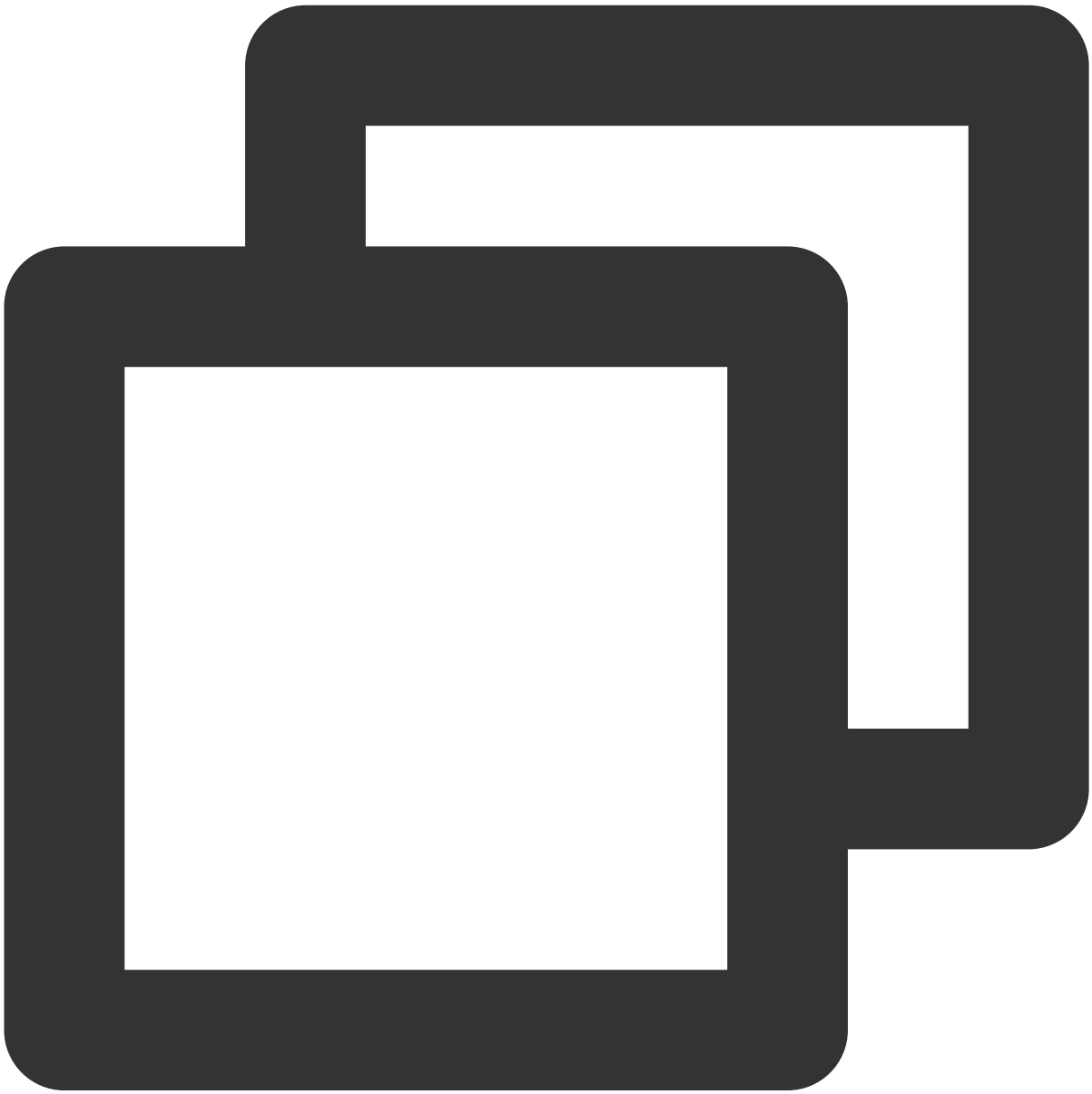
参数如下表所示：

参数	类型	含义
code	int	错误码

message	String	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallReceived: (String callerId, List<String> calleeIdList, String groupId, TU
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
callerId	String	主叫 ID（邀请方）
calleeIdList	List<String>	被叫 ID 列表（被邀请方）
groupId	String	群组ID
callMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

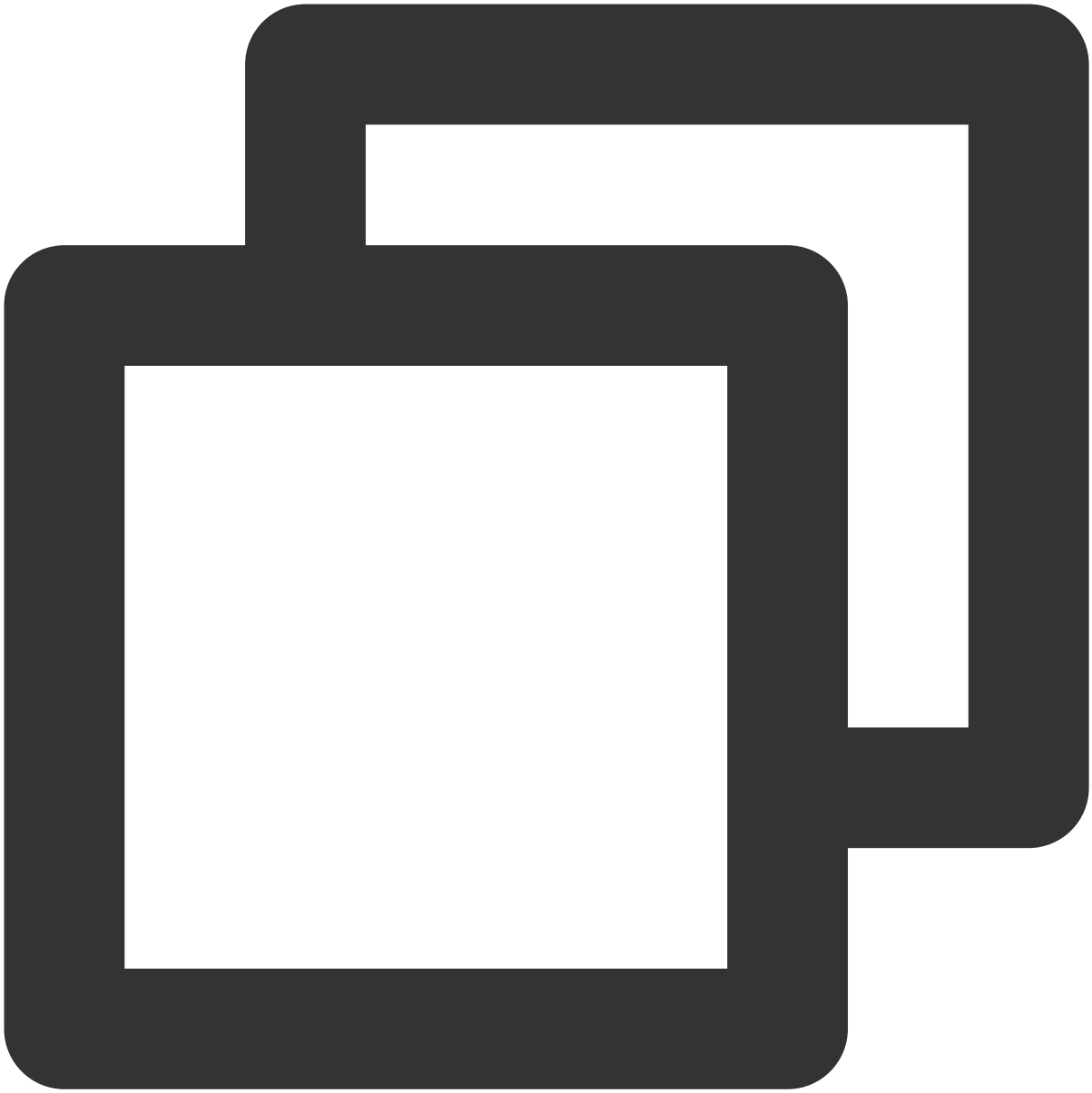
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）

被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）

异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）



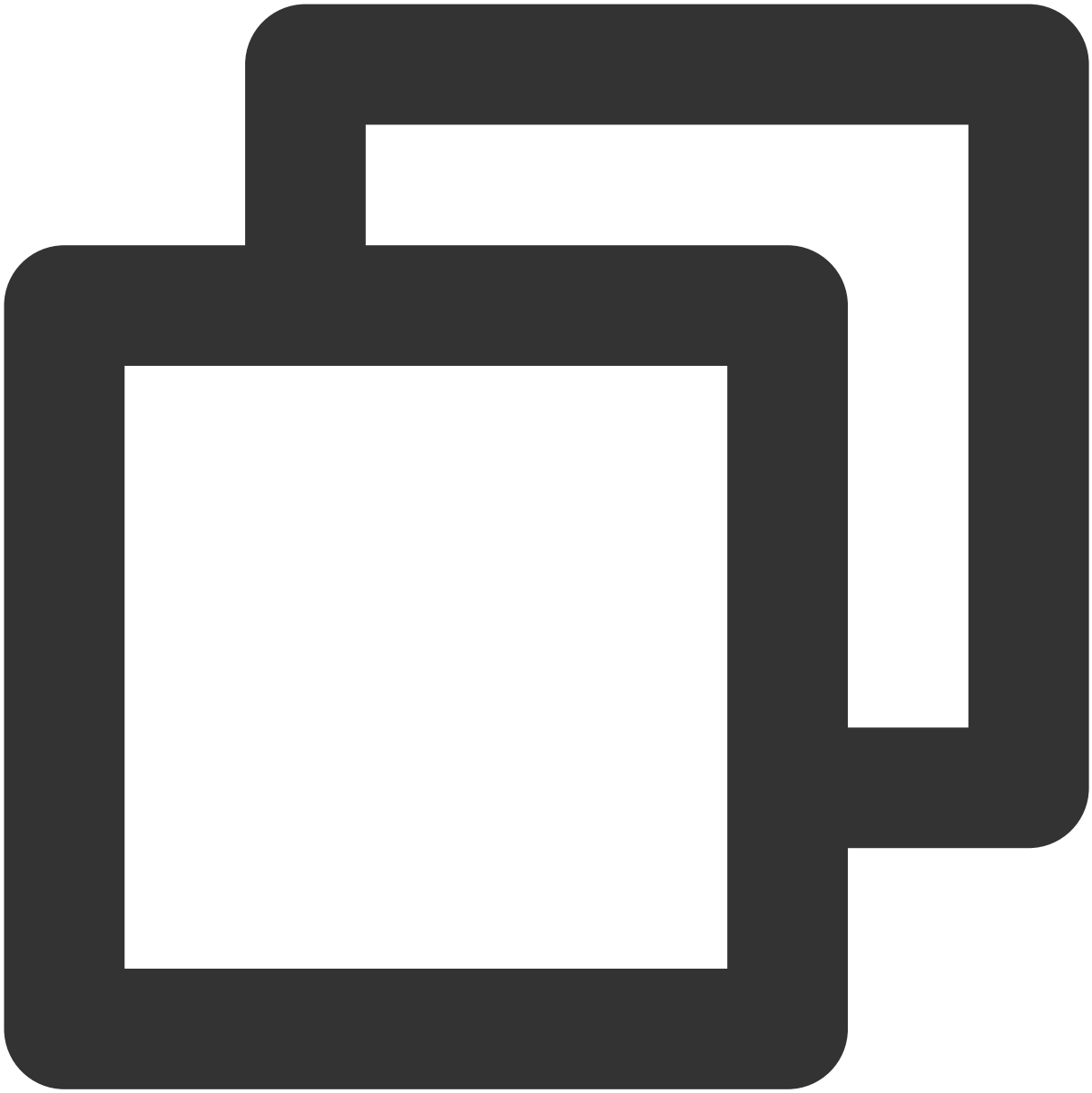
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallCancelled: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	用户的 ID

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。



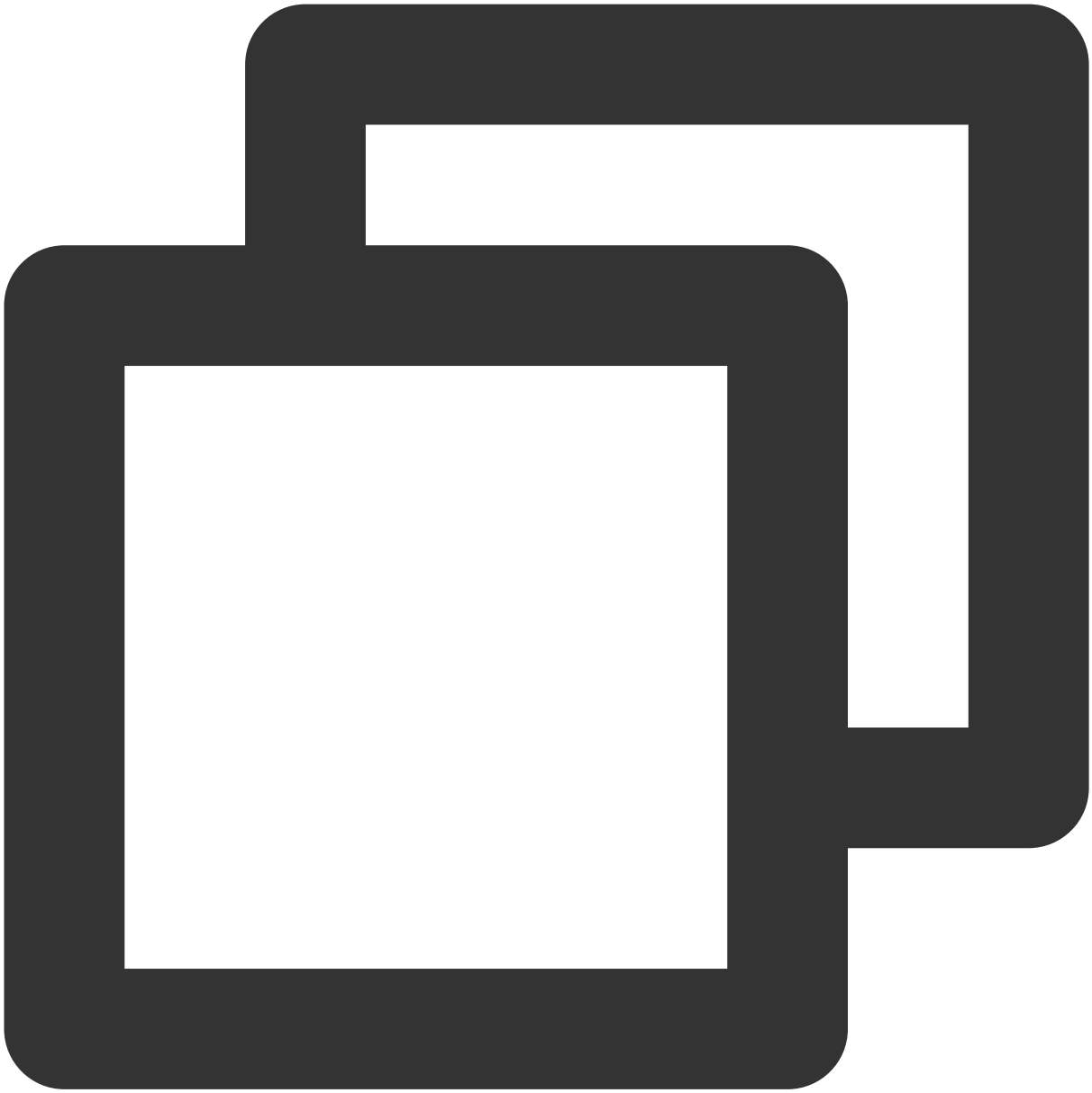
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole cal
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 Id
callMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`
callRole	TUICallRole	角色，枚举类型：主叫(`TUICallRole.caller`)、被叫(`TUICallRole.called`)

onCallEnd

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole callR
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 Id

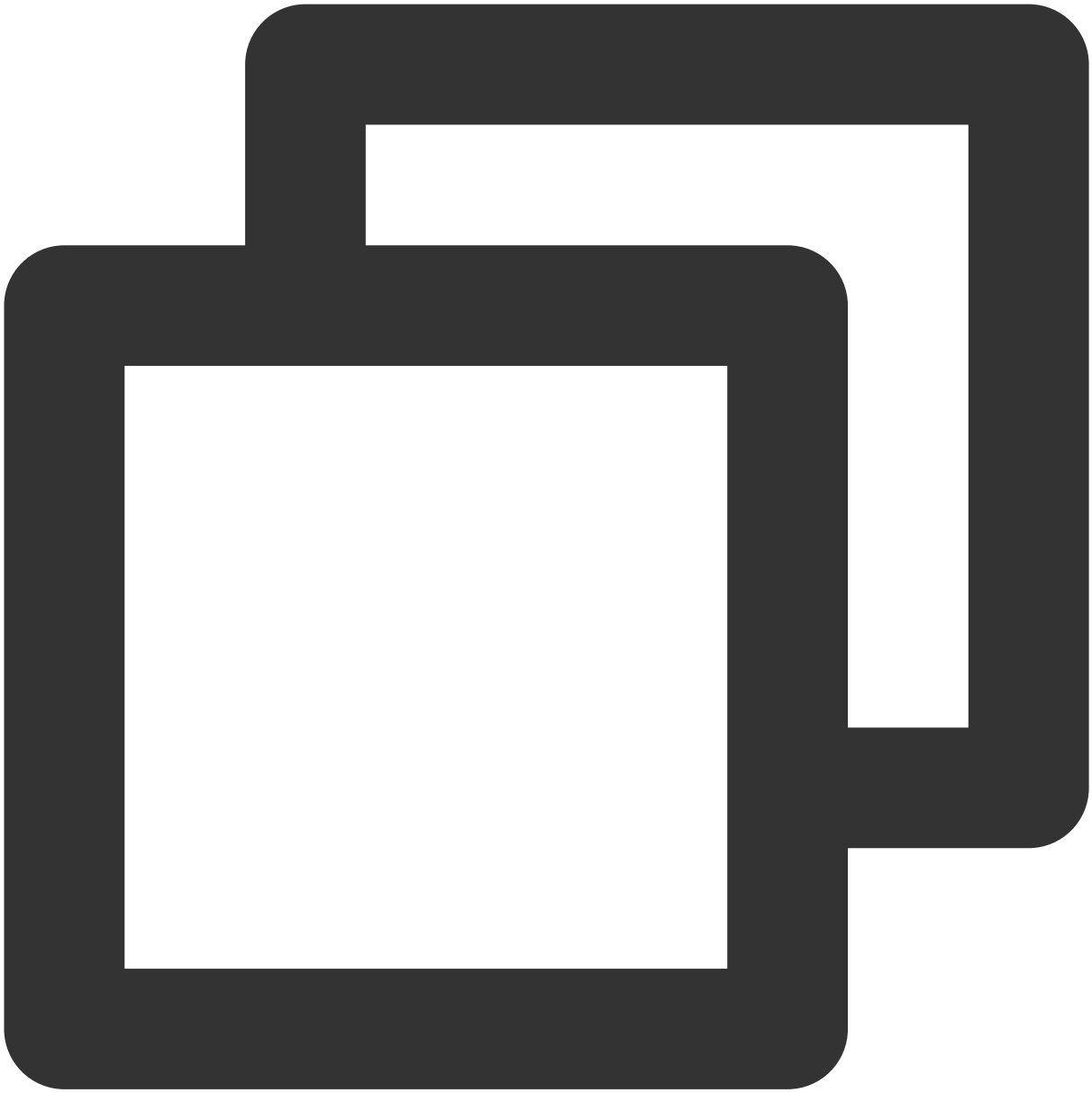
callMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`
callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)
totalTime	double	此次通话的时长，单位 ms

注意

客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType ne
      //您的回调处理逻辑
    }
));
```

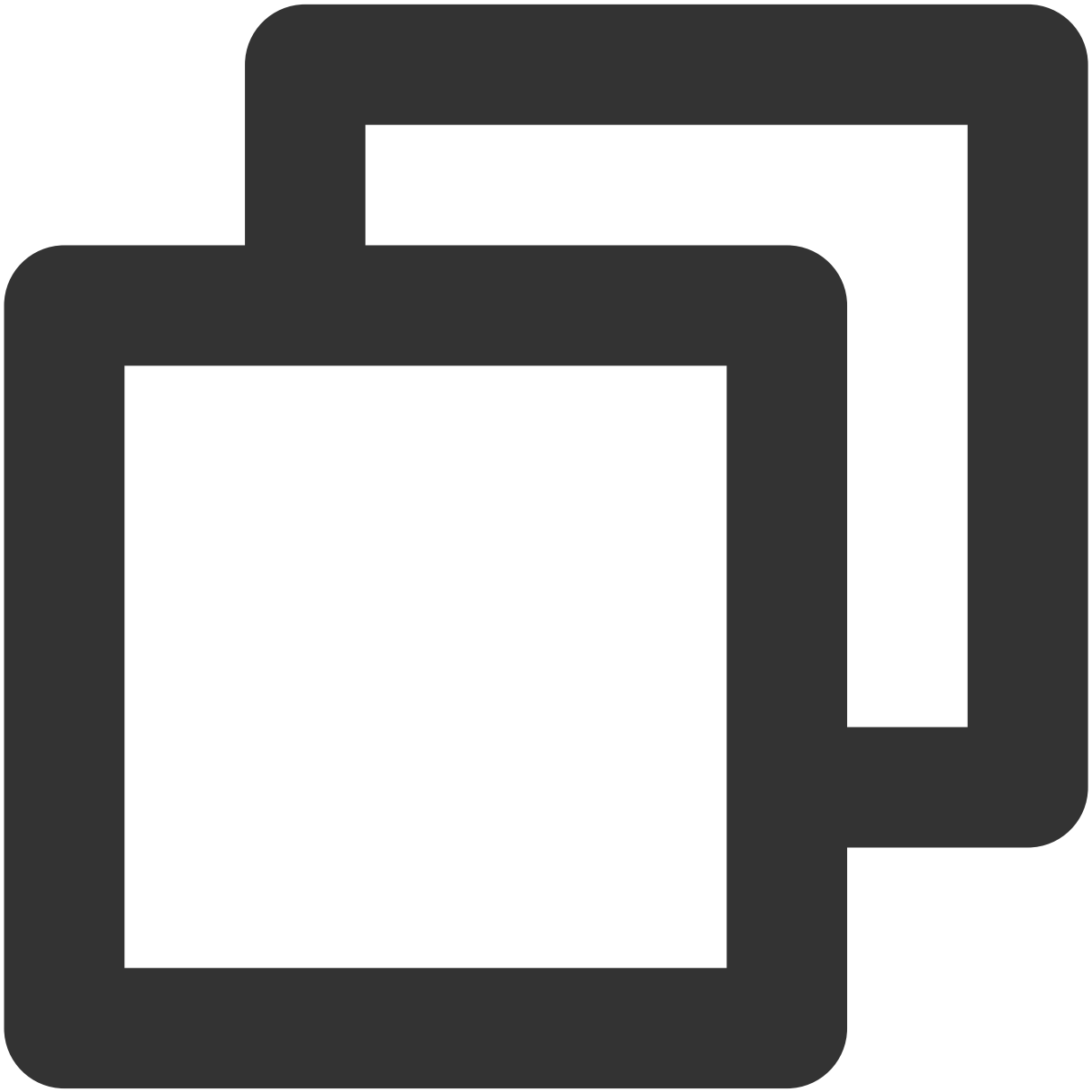
参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或

		`TUICallMediaType.audio`
newCallMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserReject: (String userId) {
```

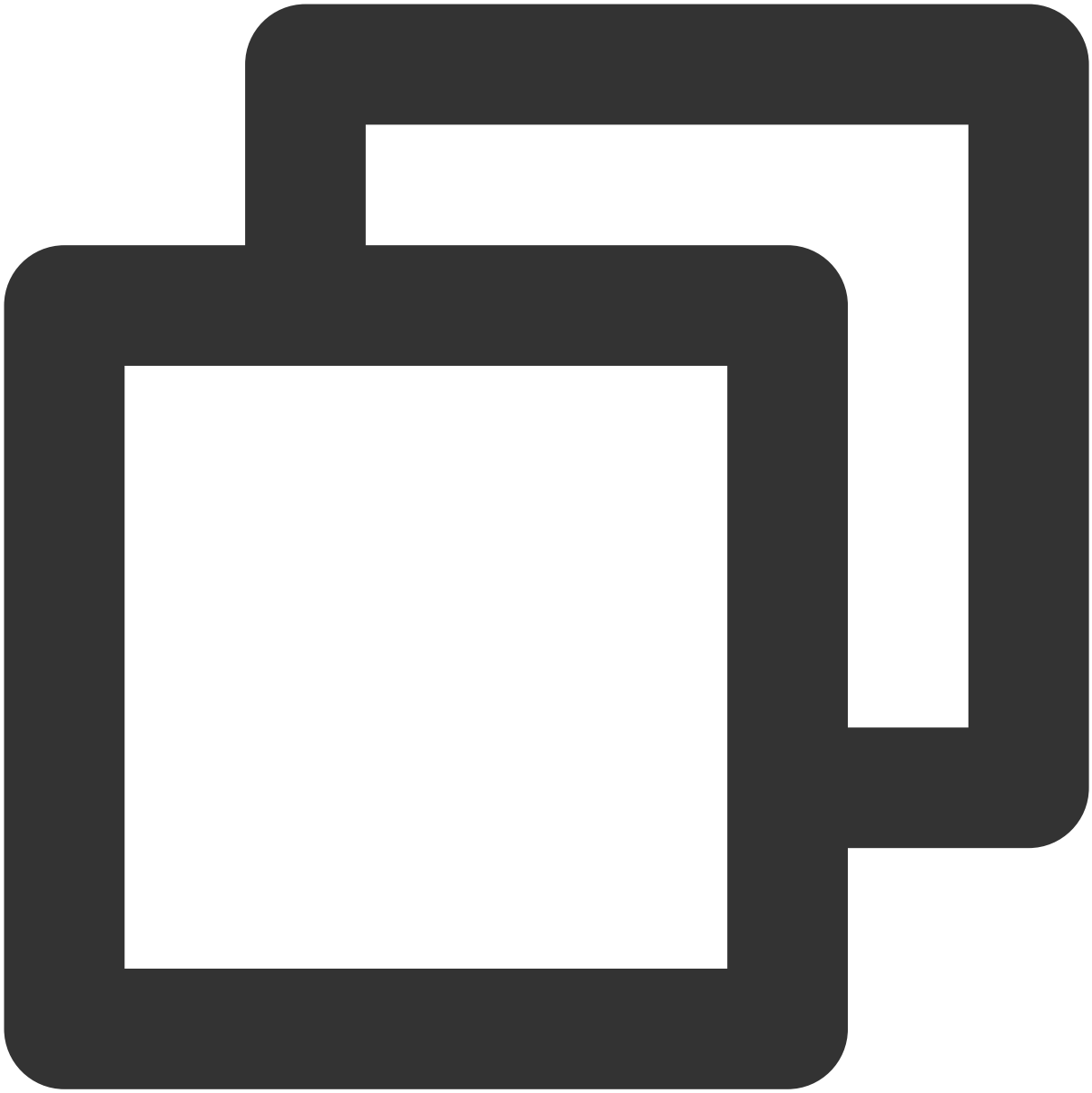
```
        //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
res.userId	String	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。



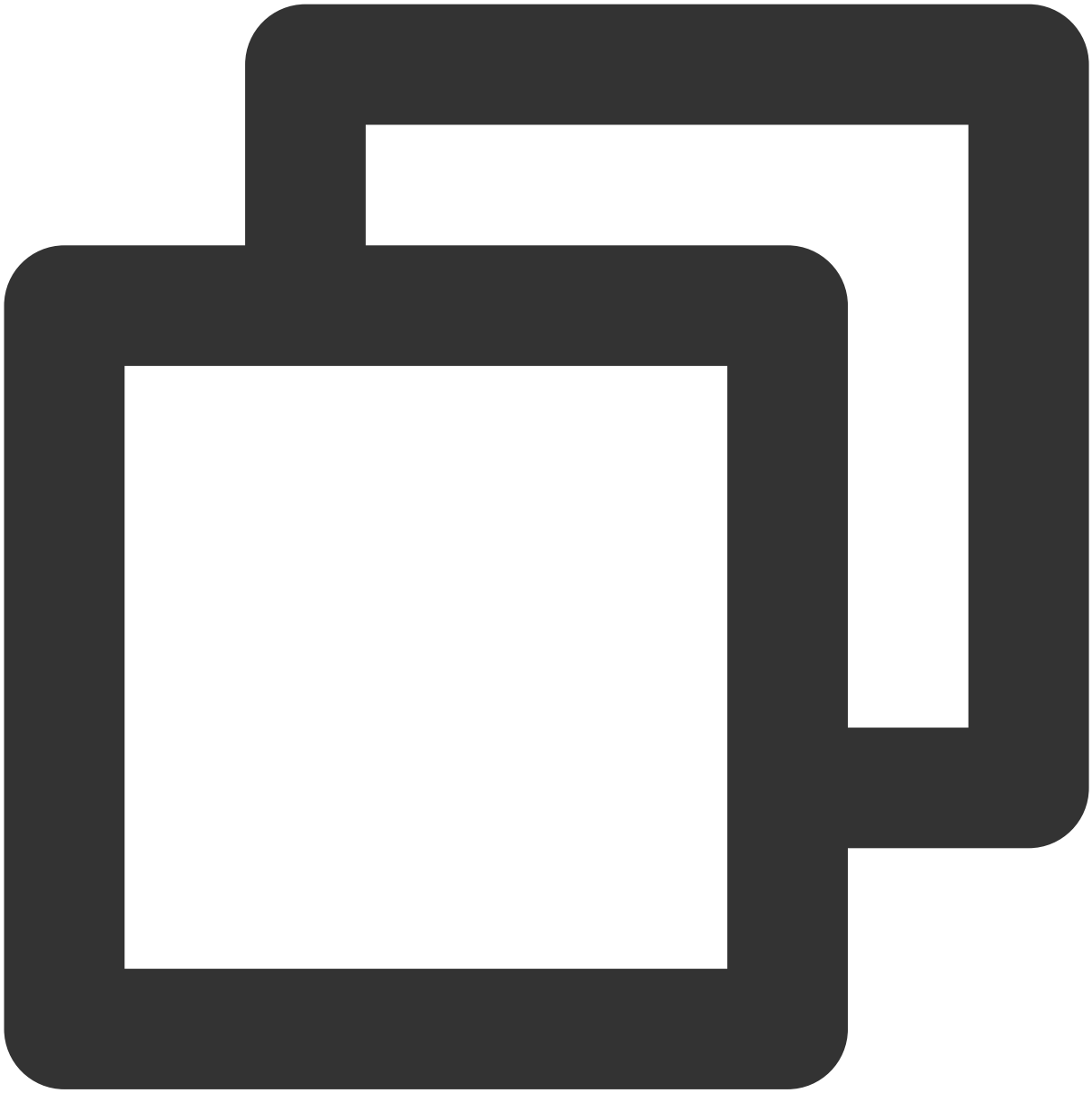
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserNoResponse: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	无响应用户的 ID

onUserLineBusy

通话忙线回调。



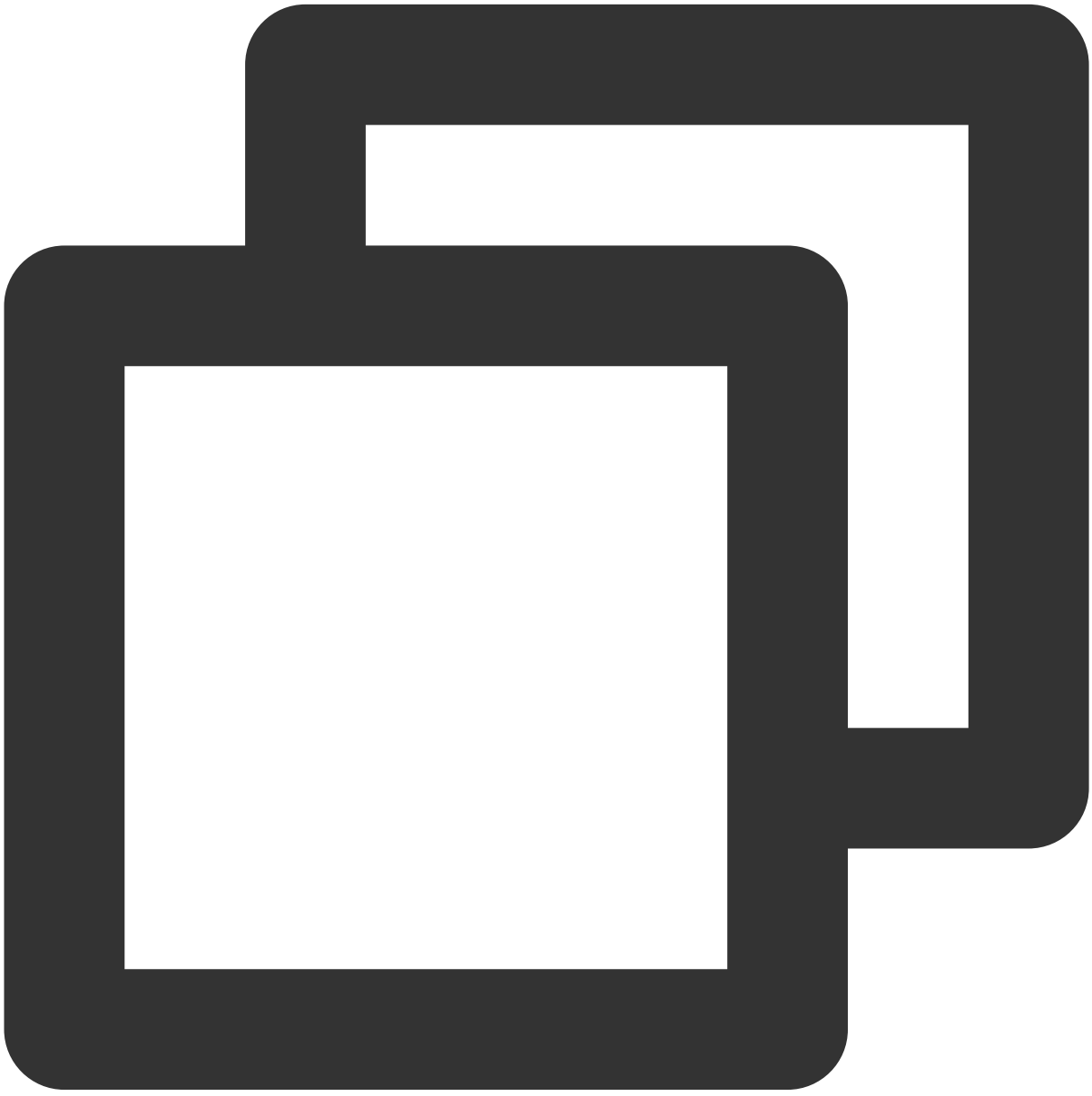
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLineBusy: (String onUserLineBusy) {
      //您的回调处理逻辑
    },
));
```

参数如下表所示：

参数	类型	含义
userId	String	忙线用户的 ID

onUserJoin

有用户进入此次通话的回调。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserJoin: (String userId) {
      //您的回调处理逻辑
```

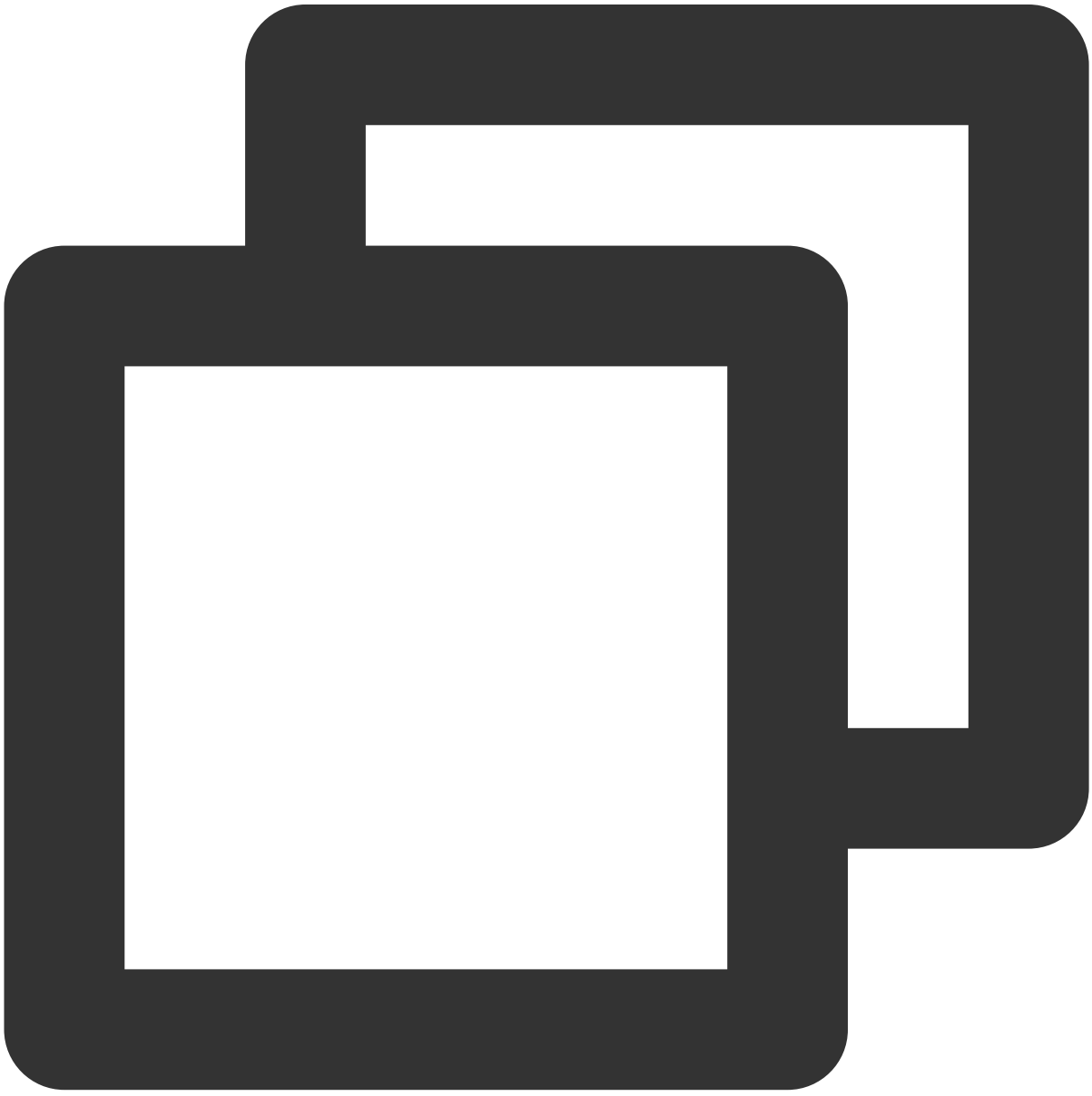
```
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。



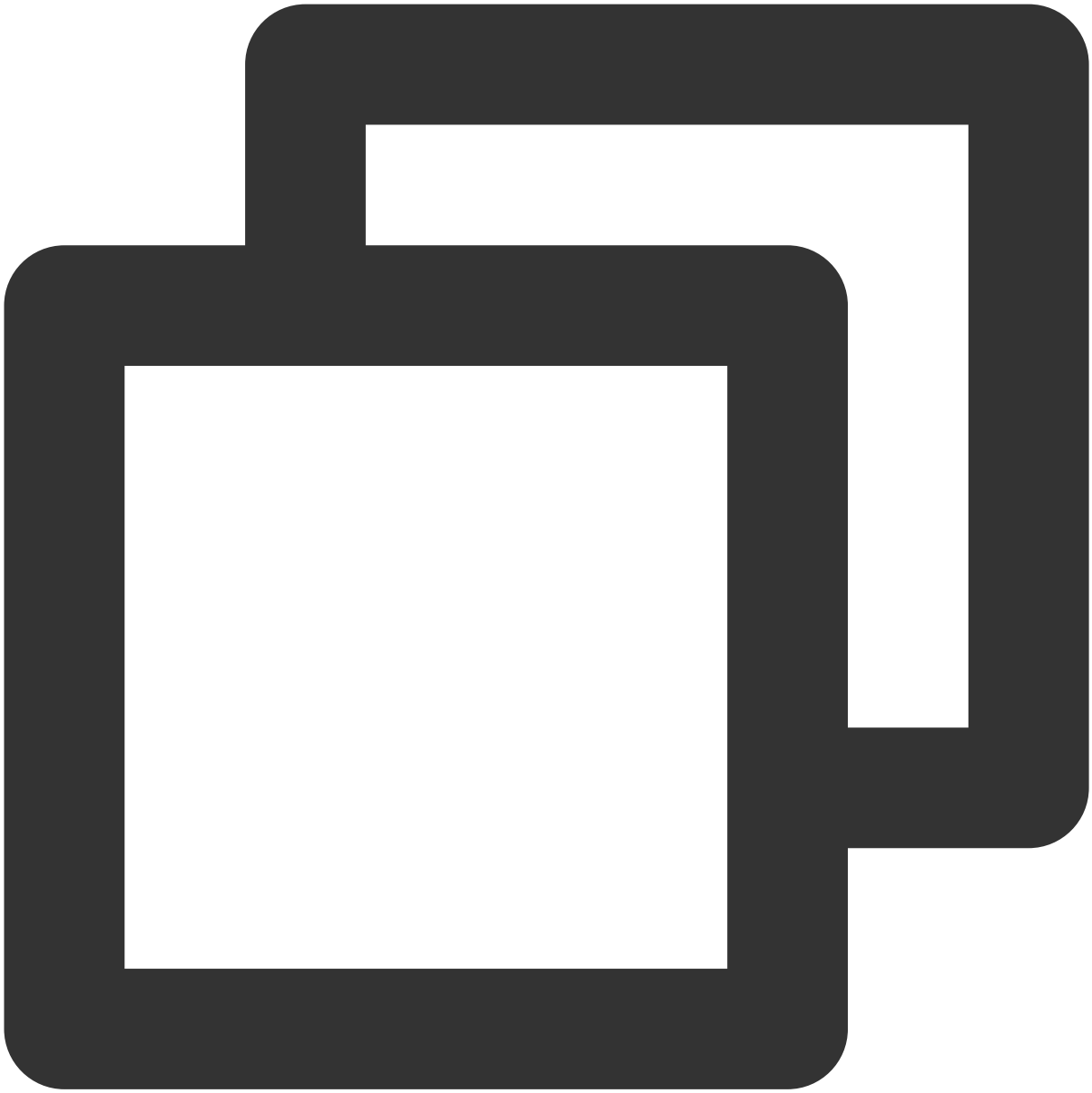
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLeave: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVideoAvailable: (String userId, bool isVideoAvailable) {
      //您的回调处理逻辑
    }
));
```

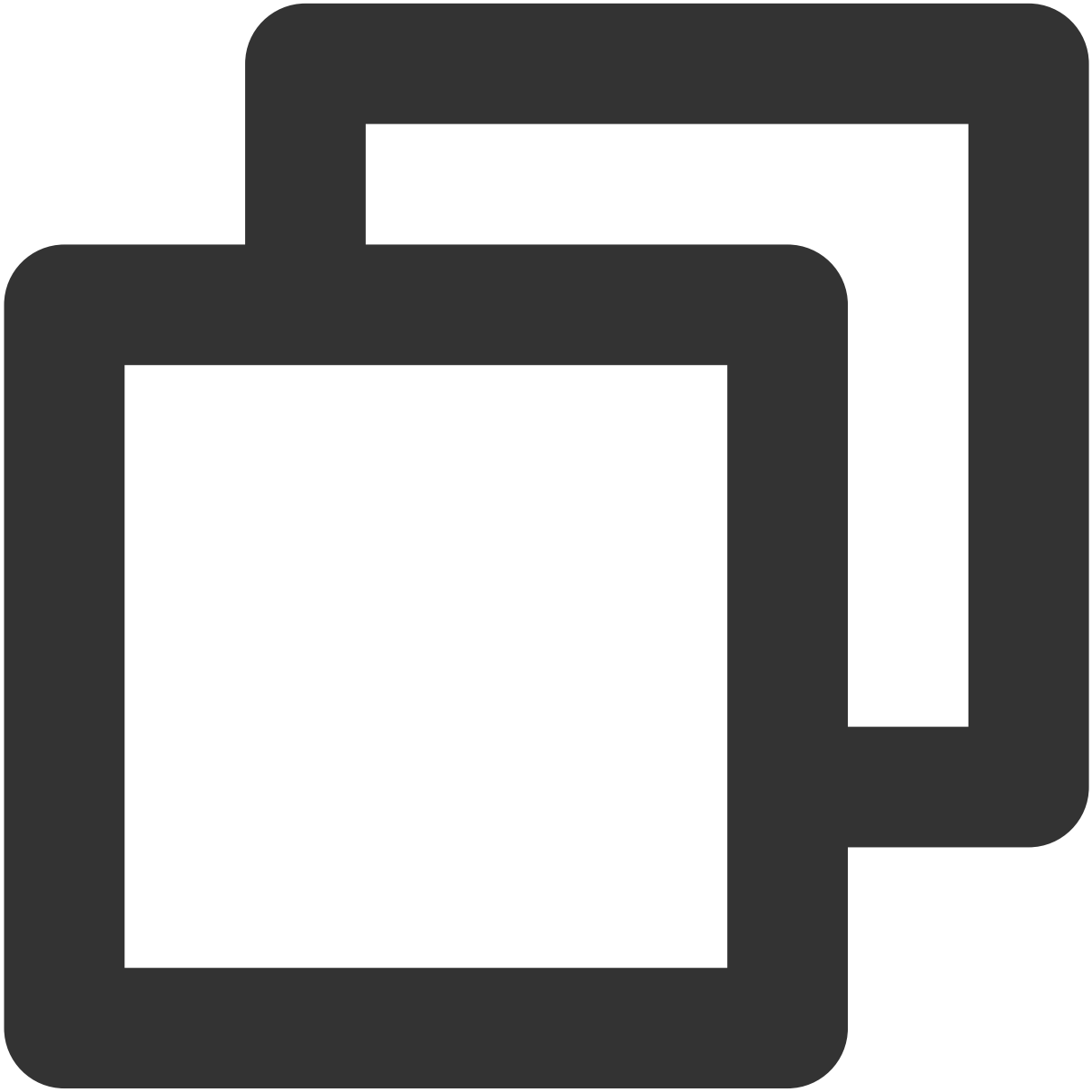
参数如下表所示：

参数	类型	含义
userId	String	通话用户 ID

isVideoAvailable	bool	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。



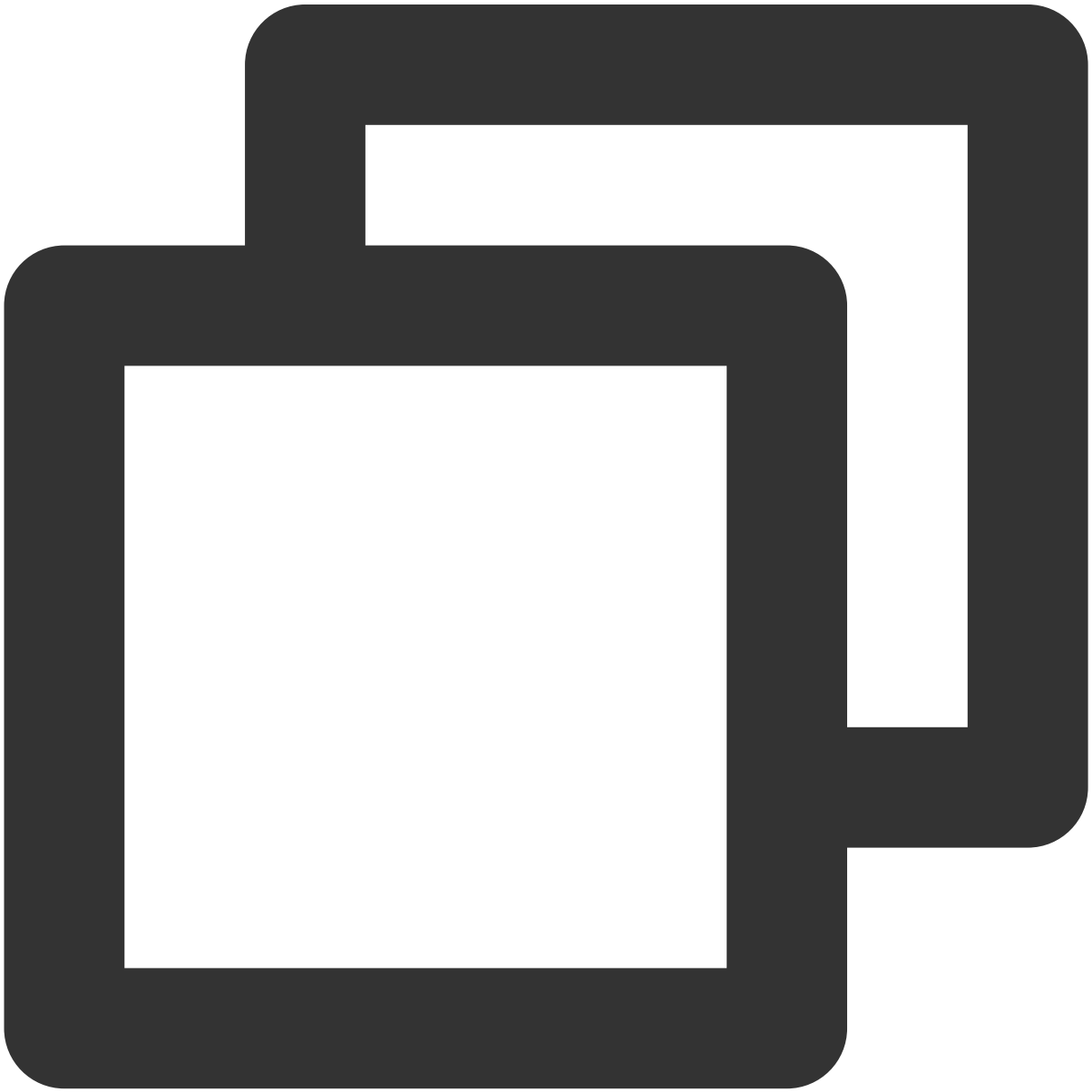
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID
isAudioAvailable	bool	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。



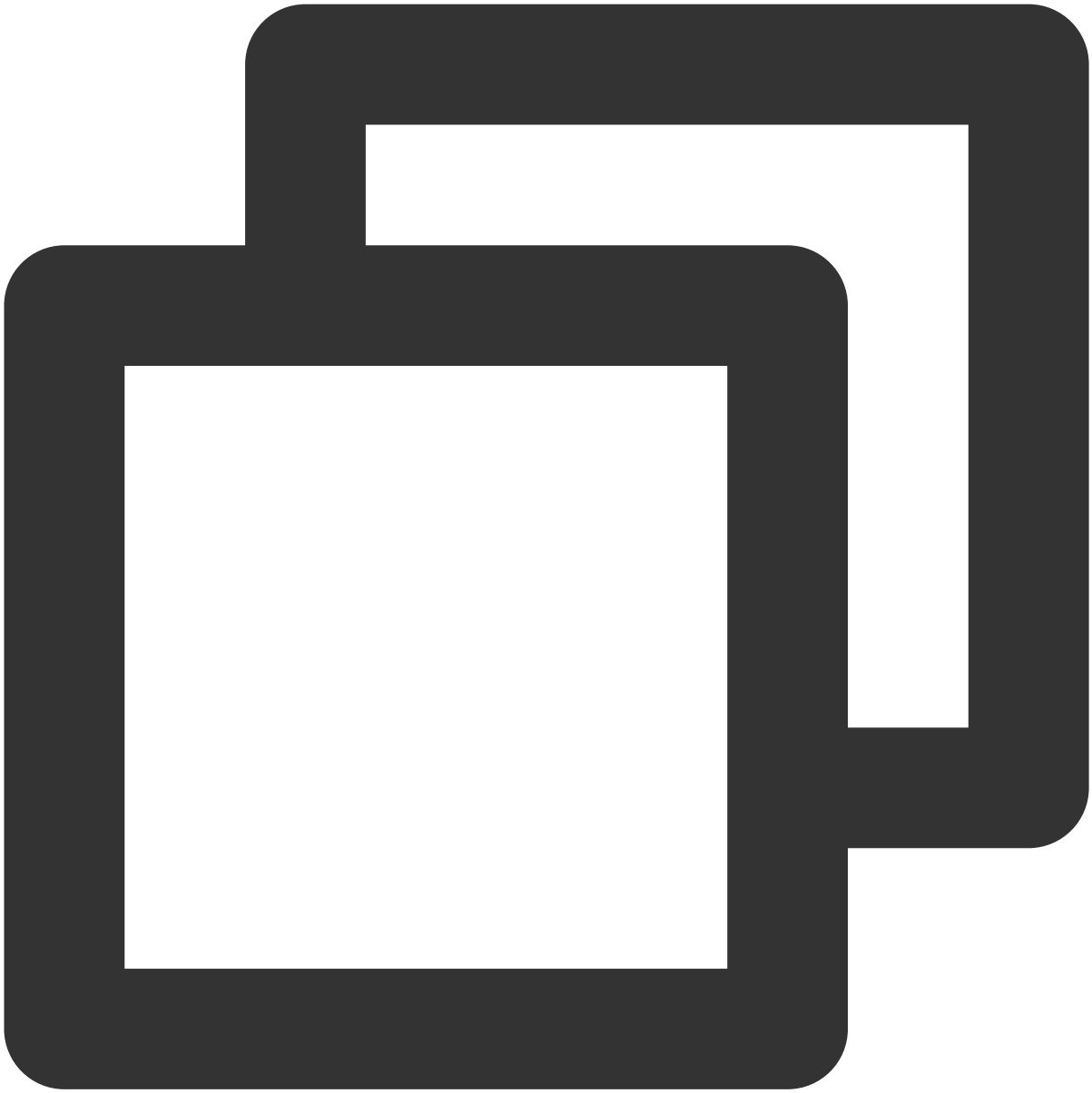
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
volumeMap	Map<String, int>	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList) {
      //您的回调处理逻辑
    }
));

//TUINetworkQualityInfo的定义如下：
class TUINetworkQualityInfo {
    String userId;
    TUINetworkQuality quality;
    TUINetworkQualityInfo({required this.userId, required this.quality});
```

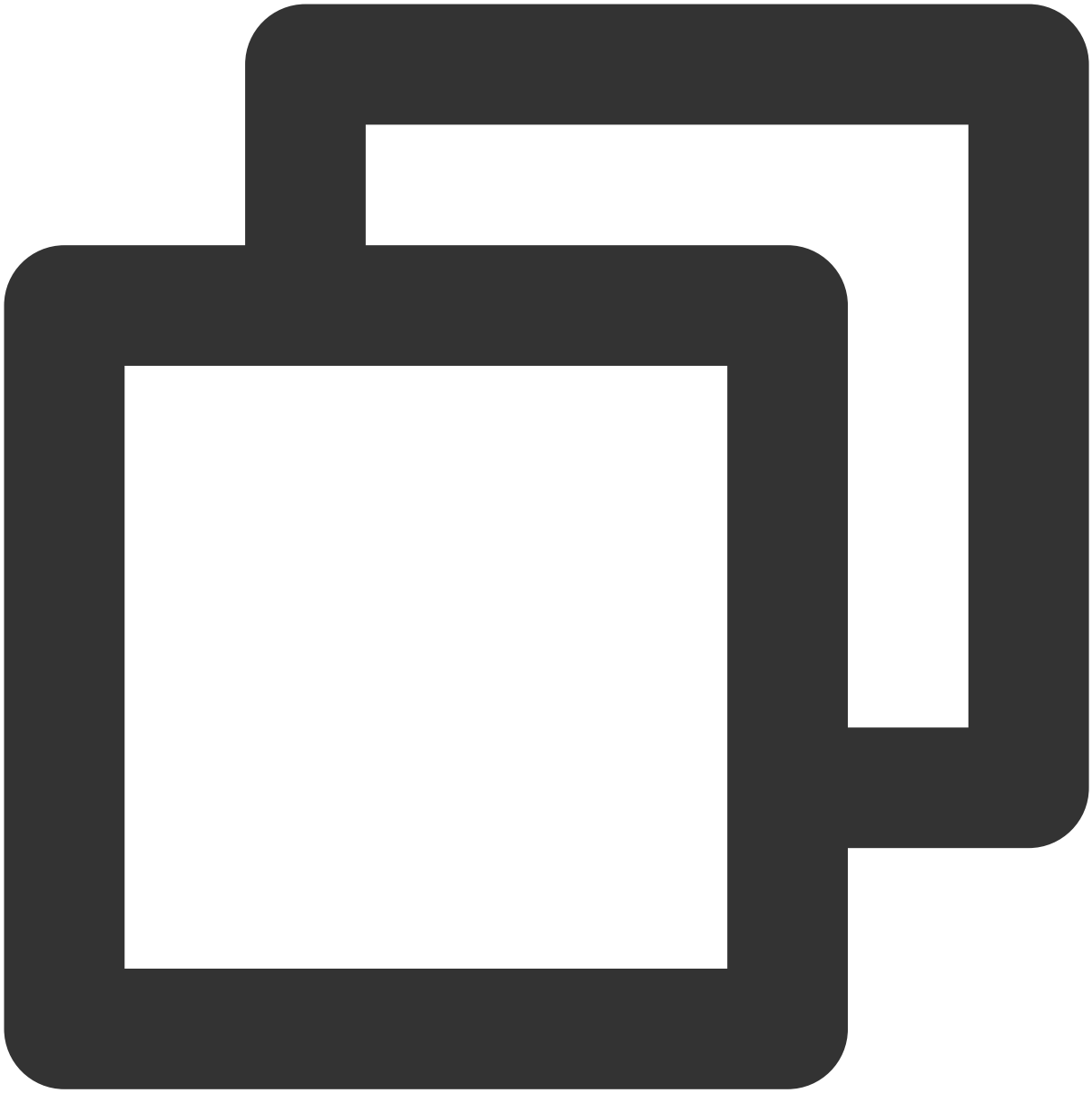
```
}
// TUINetworkQuality的定义如下：
enum TUINetworkQuality {
  unknown,
  excellent,
  good,
  poor,
  bad,
  vBad,
  down
}
```

参数如下表所示：

参数	类型	含义
networkQualityList	List<TUINetworkQualityInfo>	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

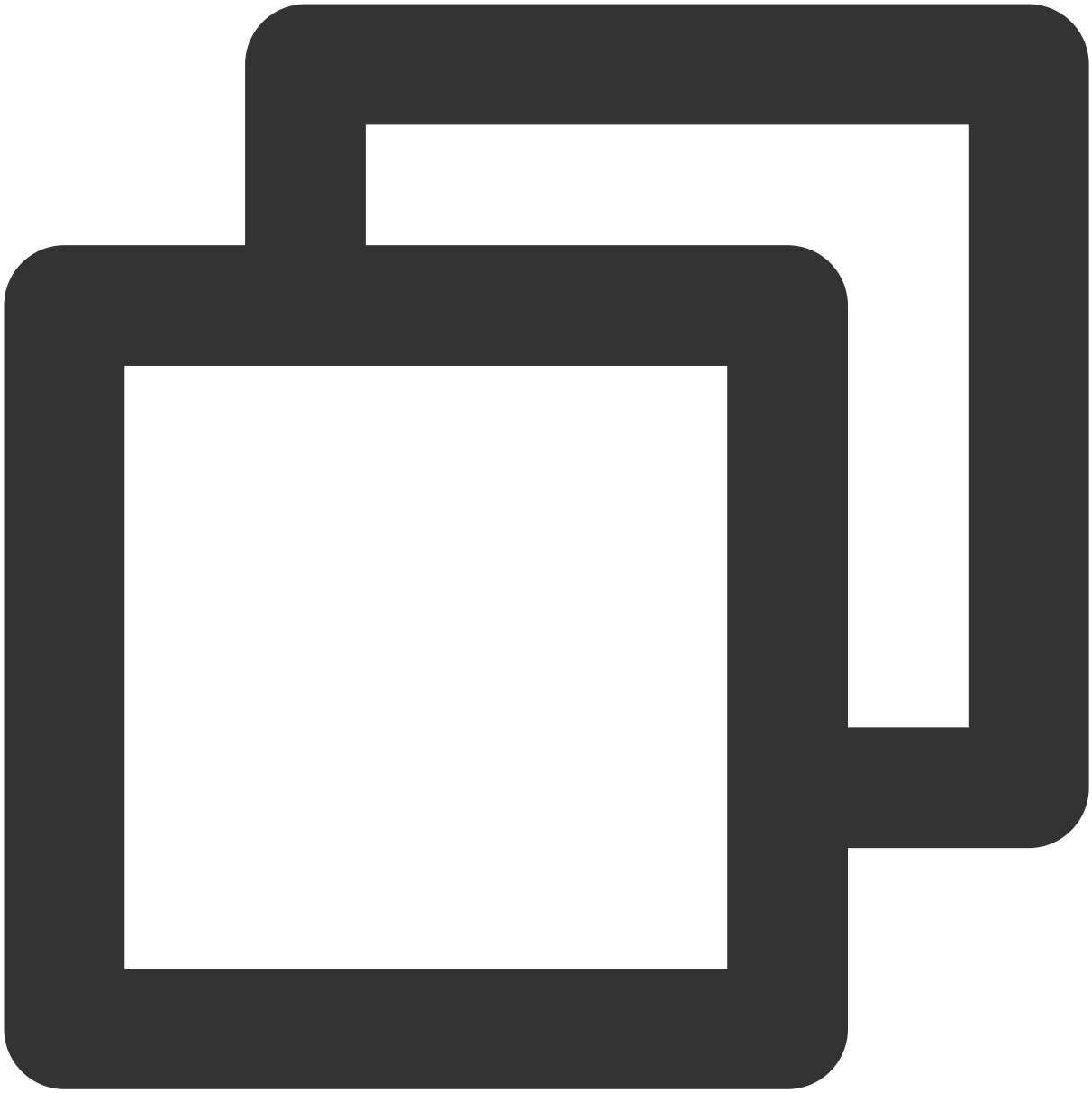
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onKickedOffline: () {
      //您的回调处理逻辑
    }
));
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。



```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserSigExpired: () {
      //您的回调处理逻辑
    }
));
```

类型定义

最近更新时间：2023-08-18 14:59:58

常用结构

TUIResult

调用 API 的返回值

类型	类型	描述
code	String	code 为空 "" 表示调用成功,code 不为空 "" 表示调用失败
message	String?	失败原因

TUIRoomId

房间 ID

类型	类型	描述
intRoomId	int	数字房间号
strRoomId	String	字符串房间号

VideoRenderParams

视频画面的渲染参数

类型	类型	描述
fillMode	FillMode	视频画面填充模式
rotation	Rotation	视频画面旋转方向

VideoEncoderParams

视频编码参数

类型	类型	描述
resolution	Resolution	视频分辨率
resolutionMode	ResolutionMode	视频宽高比模式

版权所有：腾讯云计算（北京）有限责任公司

TUICallParams

通话参数

类型	类型	描述
roomId	TUIRoomId	房间 ID
offlinePushInfo	TUIOfflinePushInfo	厂商离线推送配置信息
timeout	String	超时时常
userData	String	用户自定义数据

TUIOfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参考：离线唤醒

类型	类型	描述
title	String	离线推送展示通知栏标题
desc	String	离线推送展示通知栏内容
ignoreIOSBadge	bool	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加
iOSSound	String	离线推送声音设置（仅对 iOS 生效）。 当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。 当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。 如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	String	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。 只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声。 另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效
androidOPPOChannelID	String	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidVIVOClassification	int	VIVO 推送消息分类 (待废弃接口，VIVO 推送服

		务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1
androidXiaoMiChannelID	String	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	String	FCM 通道手机 8.0 系统及以上的渠道 ID
androidHuaWeiCategory	String	华为推送消息分类，详见：华为消息分类标准
isDisablePush	bool	是否关闭推送（默认开启推送）
iOSPushType	TUICallIOSOfflinePushType	iOS 离线推送类型，默认：APNs

TUICallRecords

通话记录信息

类型	类型	描述
callId	String	通话记录ID
inviter	String	邀请者ID
inviteList	List<String>	被邀请用户ID列表
scene	TUICallScene	通话场景
mediaType	TUICallMediaType	通话媒体类型
groupId	String	群组ID
role	TUICallRole	通话角色
result	TUICallResultType	通话结果类型
beginTime	int	开始时间
totalTime	int	总时间

TUICallRecentCallsFilter

通话记录过滤条件

类型	类型	描述
begin	double	开始时间
end	double	结束时间

版权所有：腾讯云计算（北京）有限责任公司

resultType	TUICallResultType	通话记录类型

枚举定义

TUICallMediaType

通话类型

类型	取值	描述
none	0	未知类型
audio	1	语音通话
video	2	视频通话

TUICallRole

通话角色

类型	取值	描述
none	0	未知类型
caller	1	主叫（邀请方）
called	2	被叫（被邀请方）

TUICallStatus

通话状态

类型	取值	描述
none	0	未知类型
waiting	1	通话等待中
accept	2	通话已接听

TUICallScene

通话场景

类型	取值	描述
groupCall	0	群组通话

版权所有：腾讯云计算（北京）有限责任公司

multiCall	1	匿名群组通话（暂不支持，敬请期待）
singleCall	2	单人通话

TUINetworkQuality

网络质量

类型	取值	描述
unknown	0	未定义
excellent	1	当前网络非常好
good	2	当前网络比较好
poor	3	当前网络一般
bad	4	当前网络较差
vBad	5	当前网络很差
down	6	当前网络不满足通话的最低要求

FillMode

视频画面填充模式

类型	取值	描述
fill	0	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。
fit	1	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

Rotation

视频画面旋转方向

类型	取值	描述
rotation_0	0	不旋转
rotation_90	1	顺时针旋转90度
rotation_180	2	顺时针旋转180度

版权所有：腾讯云计算（北京）有限责任公司

rotation_270	3	顺时针旋转270度

ResolutionMode

视频宽高比模式

类型	取值	描述
landscape	0	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360。
portrait	1	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640。

Resolution

视频分辨率

类型	取值	描述
resolution_640_360	0	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps
resolution_640_480	1	宽高比 4:3；分辨率 640x480；建议码率（VideoCall）600kbps
resolution_960_540	2	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps
resolution_960_720	3	宽高比 4:3；分辨率 960x720；建议码率（VideoCall）1000kbps
resolution_1280_720	4	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps
resolution_1920_1080	5	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps

TUICallIOSOfflinePushType

iOS离线推送类型

类型	取值	描述
APNs	0	普通推送
VoIP	1	VoIP 推送

版权所有：腾讯云计算（北京）有限责任公司

TUICamera

摄像头类型类型

类型	取值	描述
front	0	前置摄像头
back	1	后置摄像头

TUIAudioPlaybackDevice

音频播放路由

类型	取值	描述
speakerphone	0	扬声器
earpiece	1	耳麦

TUICallResultType

通话记录类型

类型	取值	描述
unknown	0	未知
missed	1	未接
incoming	2	接入
outgoing	3	拨出

版权所有：腾讯云计算（北京）有限责任公司

uniapp（Android&iOS）
API 概览

最近更新时间：2024-03-07 10:47:48

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的头像、昵称
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

事件

目前 TUICallKit 抛出下述事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调

版权所有：腾讯云计算（北京）有限责任公司

onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，使用 TUICallEngine API，您可以做一些额外的操作（目前仅支持以下接口）。

API	描述
hangup	结束通话
accept	接听通话
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2024-03-07 18:03:53

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入 TUICallKit。

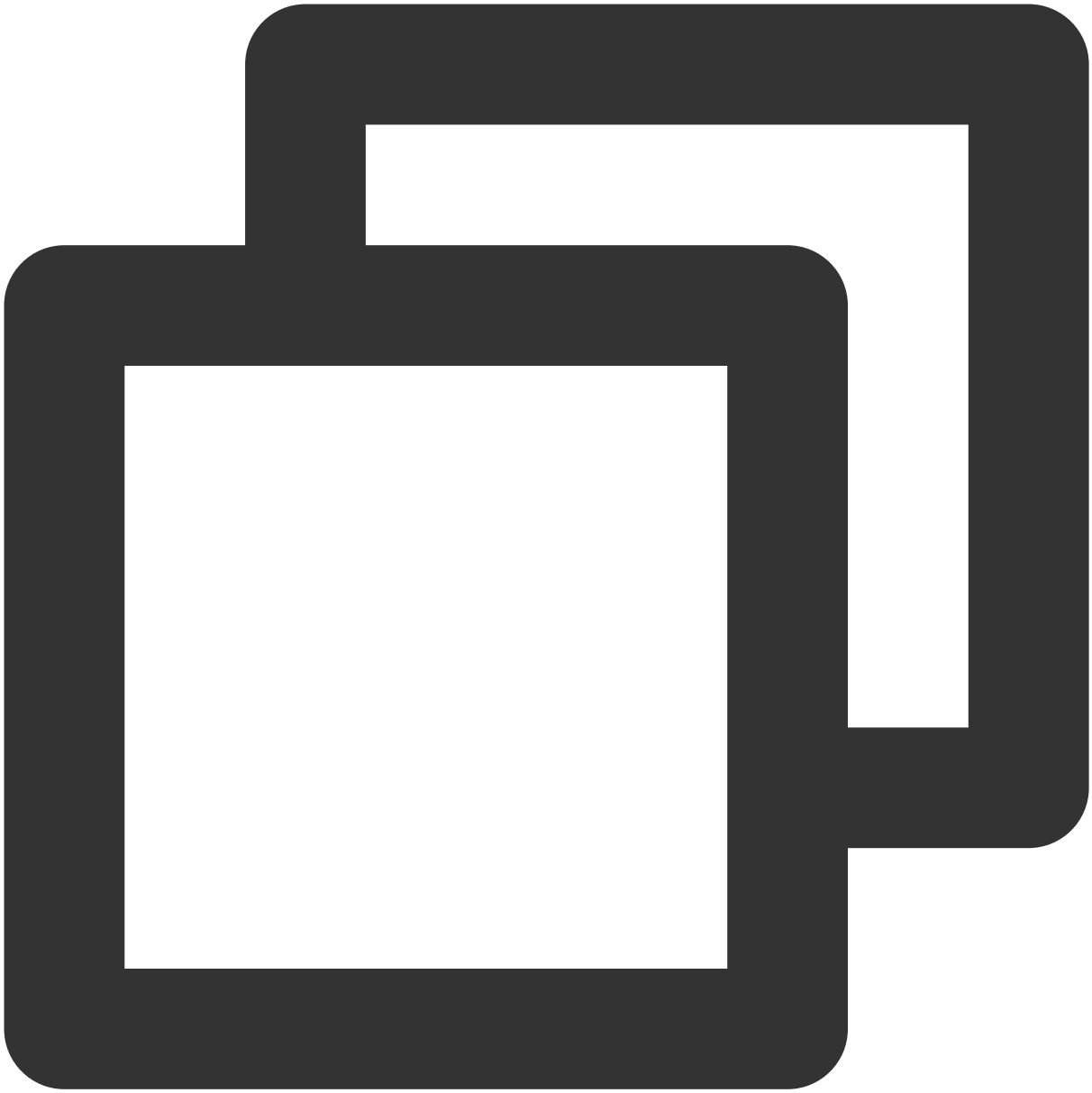
API 概览

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的昵称、头像
call	发起 1v1 通话
groupCall	发起群组通话
joinInGroupCall	主动加入当前的群组通话中
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能

API 详情

login

登录



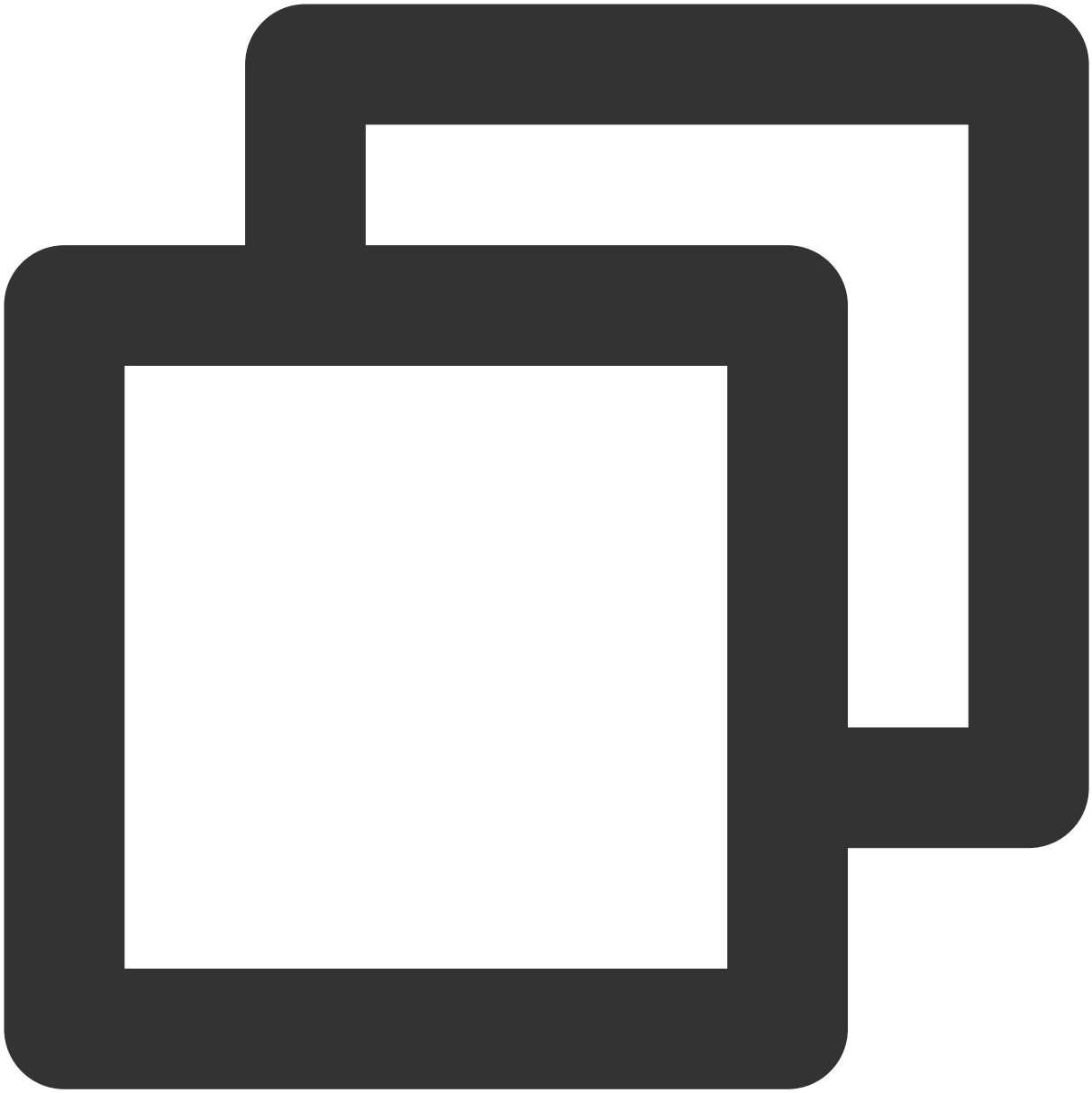
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  SDKAppID: 0,
  userID: 'mike',
  userSig: '',
};
TUICallKit.login(options, (res) => {
  if (res.code === 0) {
    console.log('login success');
  } else {
    console.log(`login failed, error message = ${res.msg}`);
```

```
    }
});
```

参数	类型	含义
options	Object	初始化参数
options.SDKAppID	Number	用户 SDKAppID
options.userID	String	用户 ID
options.userSig	String	用户签名 userSig
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

logout

登出



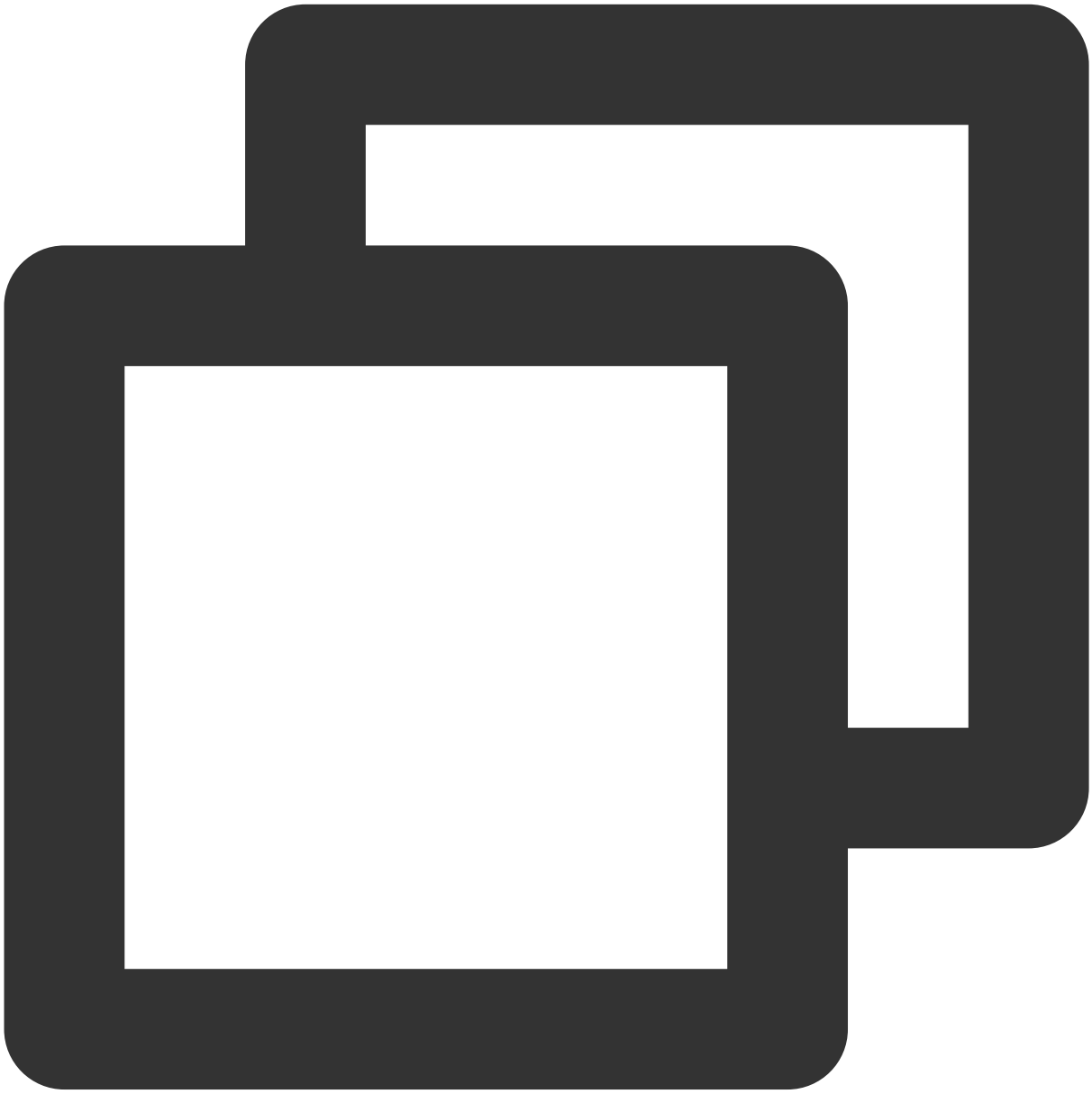
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.logout((res) => {
  if (res.code === 0) {
    console.log('logout success');
  } else {
    console.log(`logout failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义

callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是URL格式。



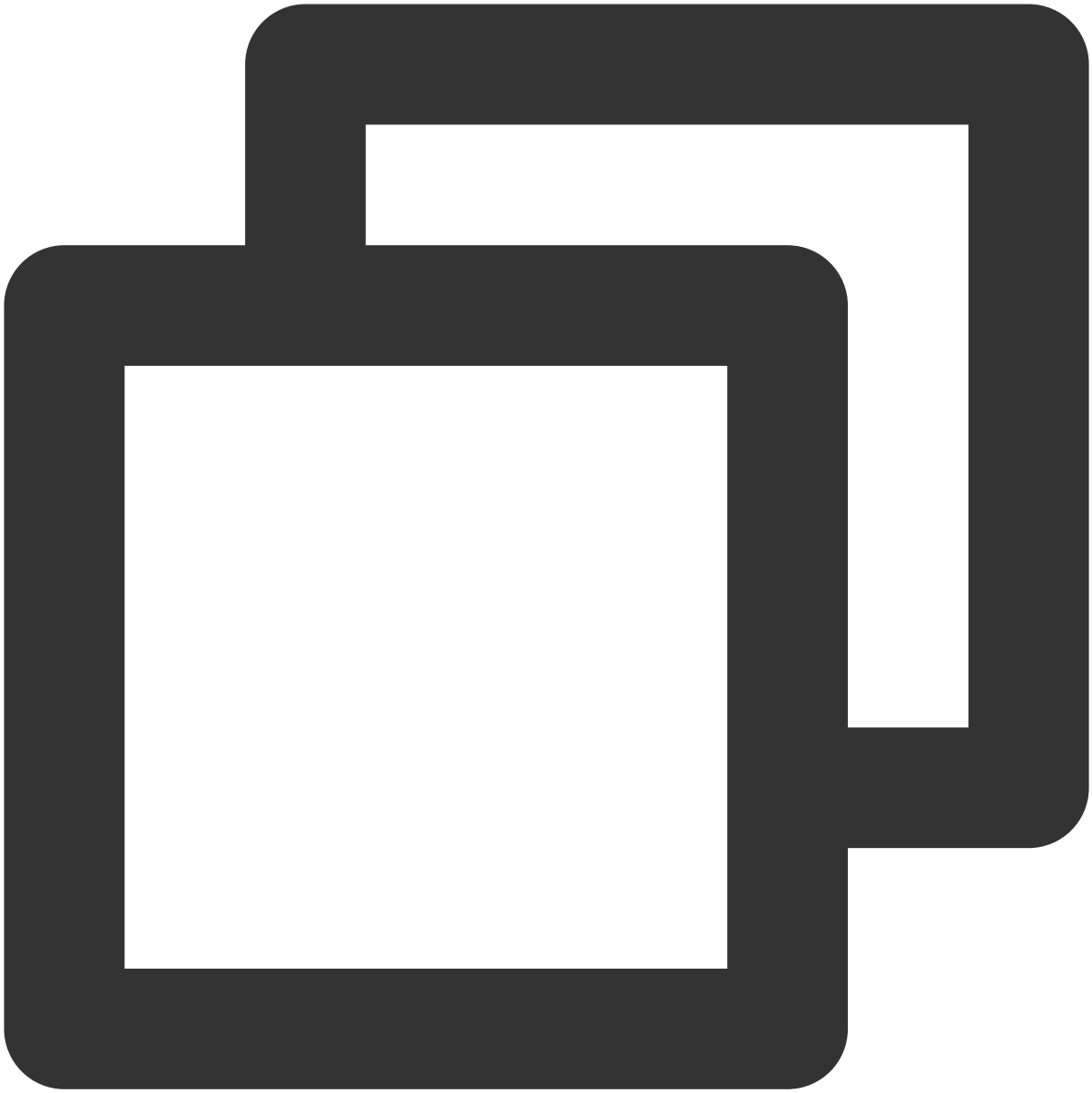
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  nickName: 'jack',
  avatar: 'https:/****/user_avatar.png'
}
```

```
TUICallKit.setSelfInfo(options, (res) => {
  if (res.code === 0) {
    console.log('setSelfInfo success');
  } else {
    console.log(`setSelfInfo failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
options	Object	初始化参数
options.nickName	String	目标用户的昵称，非必填
options.avatar	String	目标用户的头像，非必填
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

call

拨打电话（1v1通话）



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userID: 'mike',
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
  roomID: 0,
  strRoomID: '1223', // 使用字符串房间号
};
TUICallKit.call(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
```

```
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

参数如下表所示：

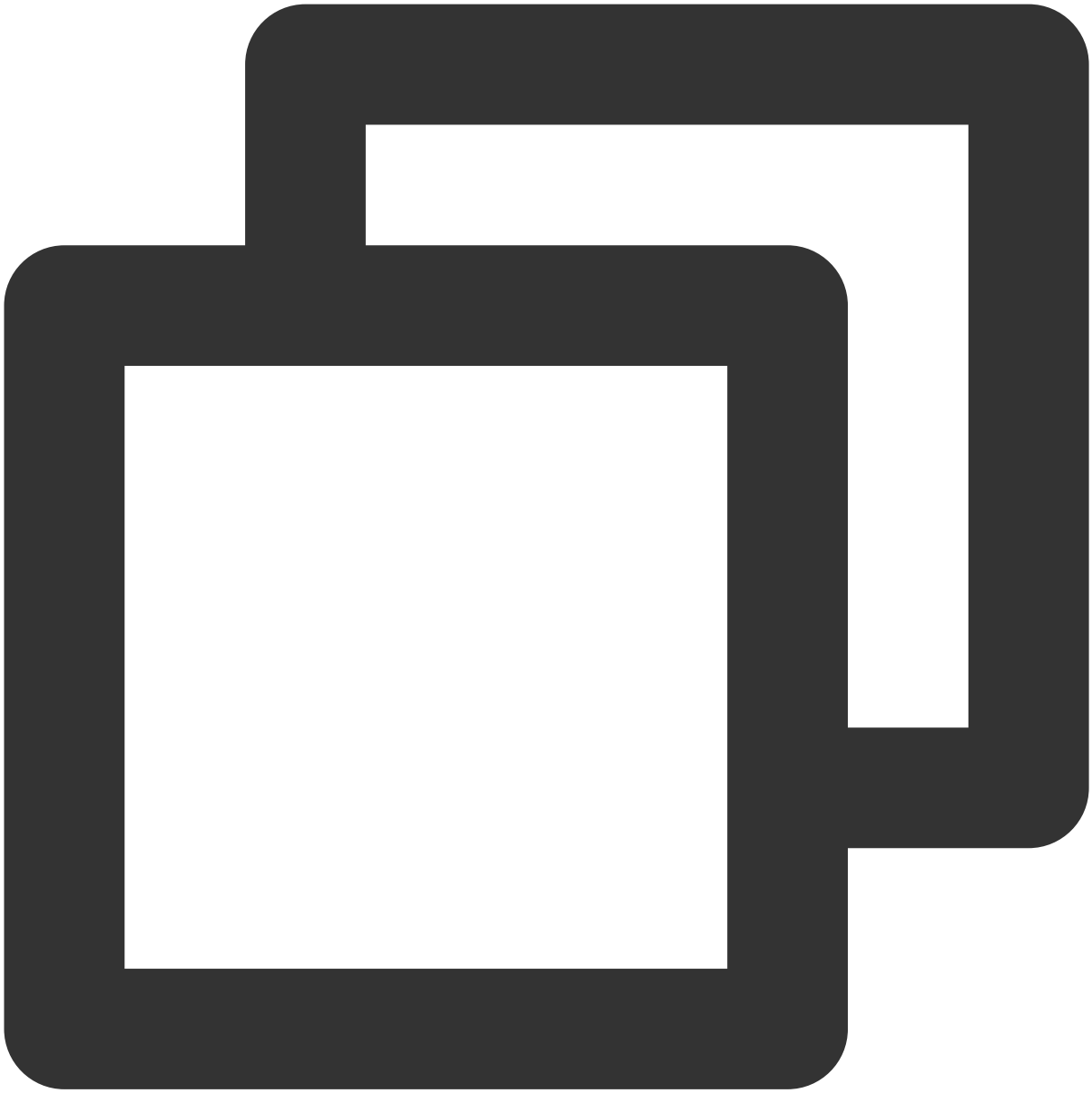
参数	类型	含义
options	Object	初始化参数
options.userID	String	目标用户的 userID
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.roomID	Number	自定义数字房间号。**只要有 roomID，就用的是数字房间号，即使 strRoomID 存在**。
options.strRoomID	String	自定义字符串房间号。**如果想使用字符串房间号，设置 strRoomID 后需要将 roomID = 0**。
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

groupCall

发起群组通话。

注意

使用群组通话前需要创建IM 群组，如果已经创建，请忽略。



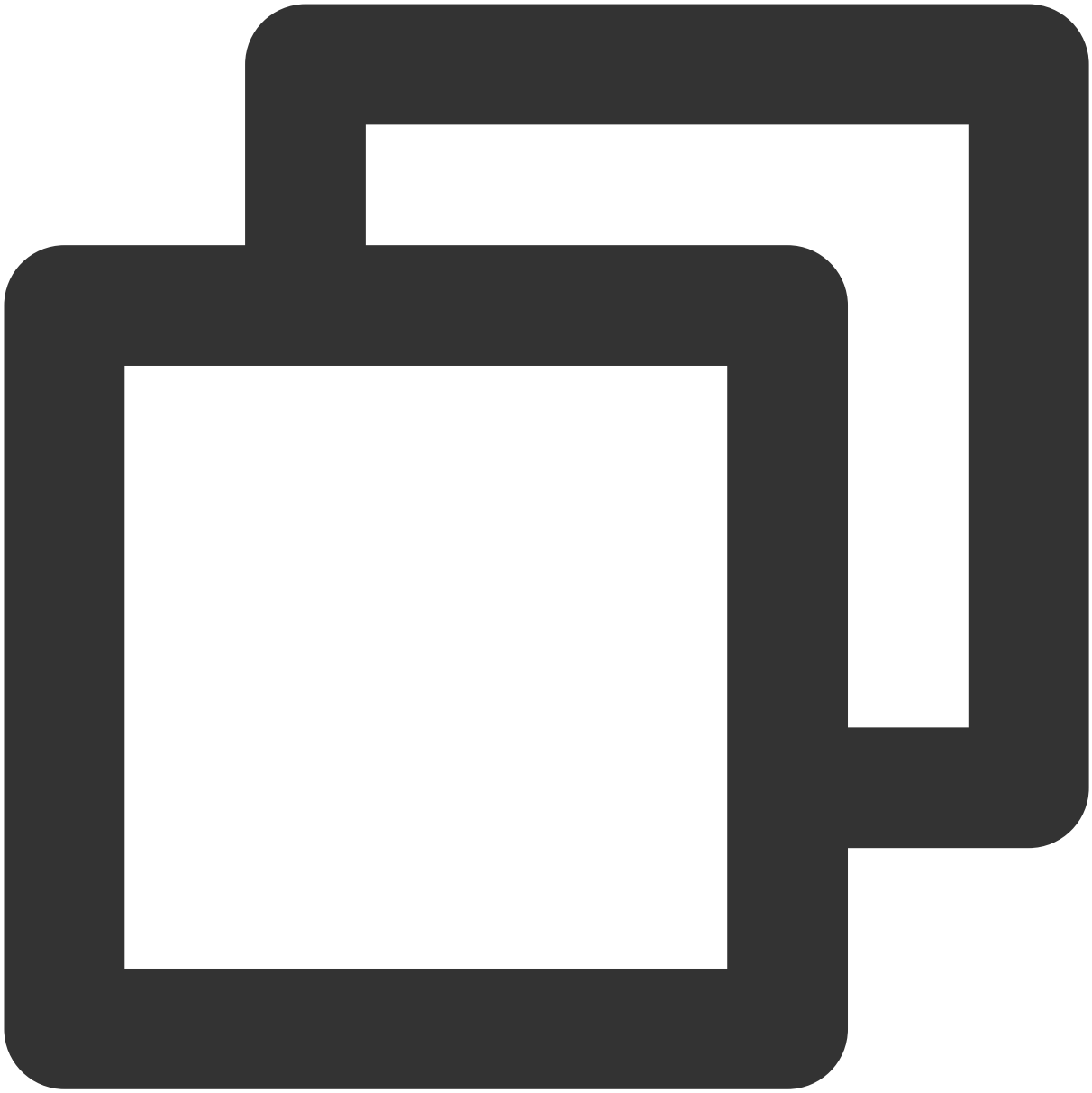
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
```

```
    }
});
```

参数	类型	含义
options	Object	初始化参数
options.groupID	String	此次群组通话的群 ID
options.userIDList	List	目标用户的userId 列表
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.roomID	Number	自定义数字房间号。**只要有 roomID，就用的是数字房间号，即使 strRoomID 存在**。
options.strRoomID	String	自定义字符串房间号。**如果想使用字符串房间号，设置 strRoomID 后需要将 roomID = 0**。

joinInGroupCall

加入群组中已有的音视频通话。



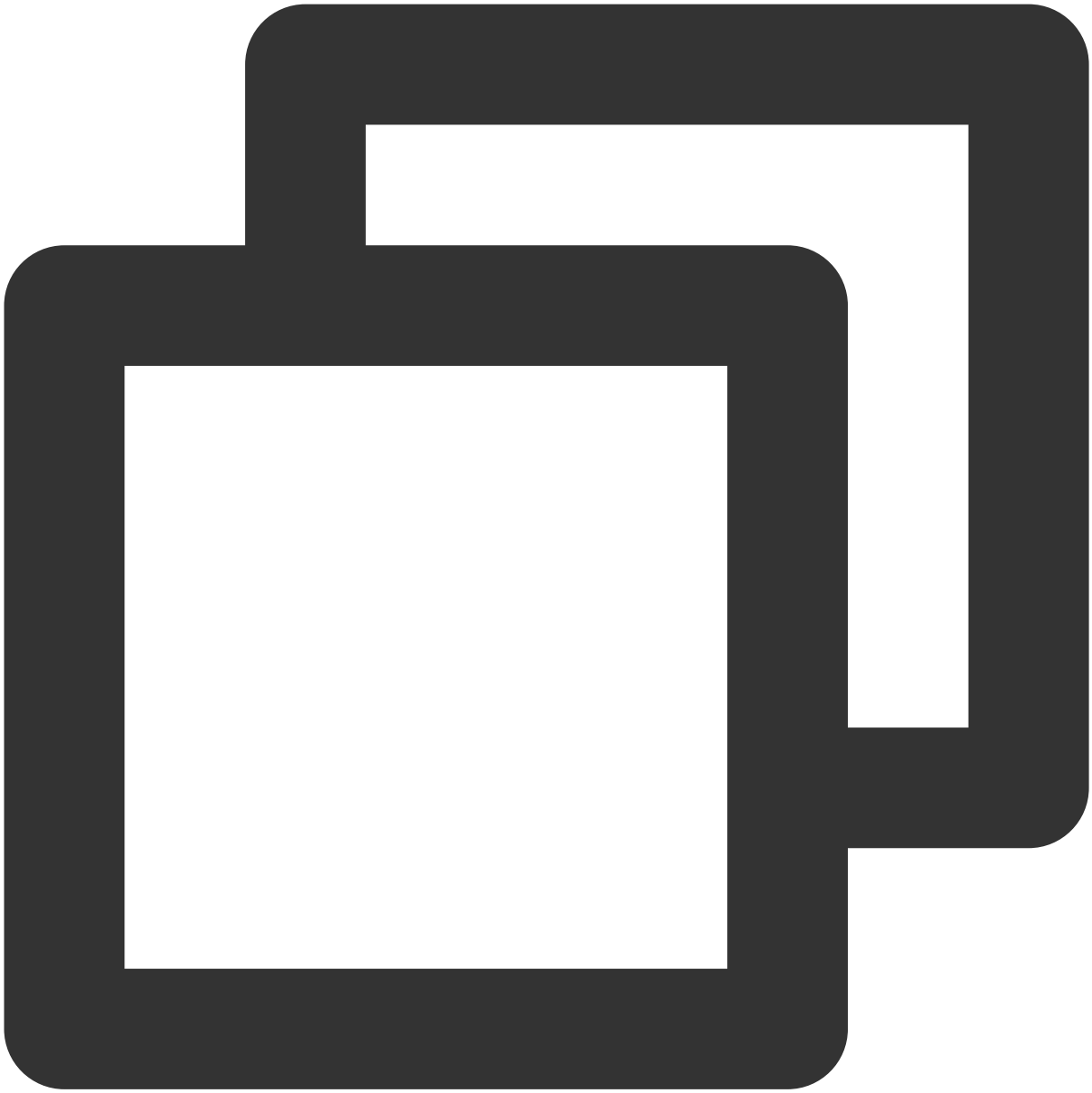
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomID: 9898,
  groupID: 'myGroup',
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
```

```
    }
});
```

参数	类型	含义
options	Object	初始化参数
options.roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
options.groupID	String	此次群组通话的群 ID
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

setCallingBell

设置自定义来电铃音，这里**仅限传入本地文件地址，需要确保该文件目录是应用可以访问的**。



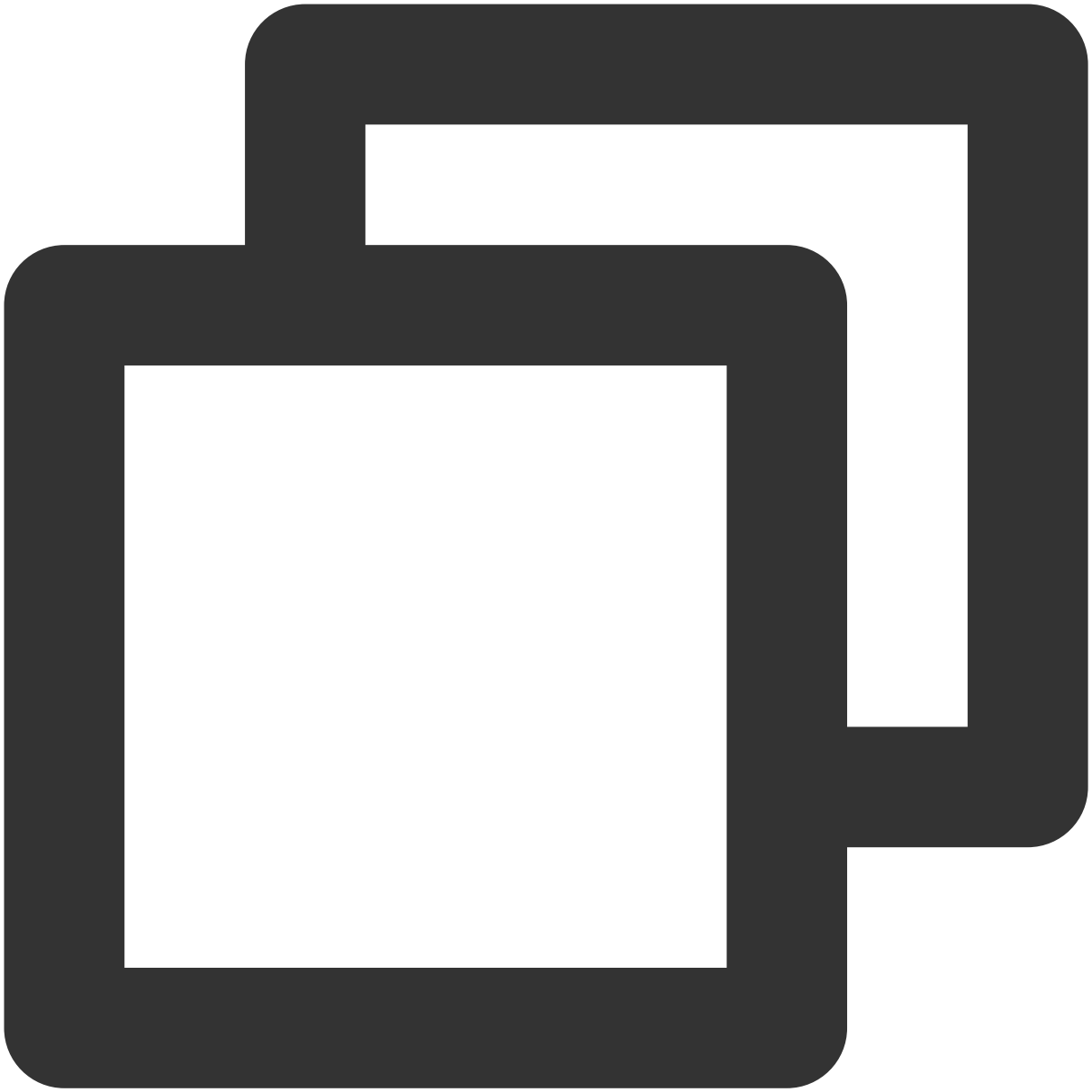
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');

// 【1】通过 uni.saveFile 保存音频文件到本地，具体参考 saveFile 接口: https://zh.uniapp.dc
const tempFilePath = './static/rain.mp3';
let musicFilePath = '';
uni.saveFile({
    tempFilePath: tempFilePath,
    success: (res) => {
        console.warn('保存文件成功 = ', JSON.stringify(res));
        musicFilePath =  res.savedFilePath;
```

```
        // 【2】相对路径转绝对路径，否则访问不到
        musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);

        // 【3】设置铃声
        TUICallKit.setCallingBell(musicFilePath, (res) => {
          if (res.code === 0) {
            console.log('setCallingBell success');
          } else {
            console.log(`setCallingBell failed, error message = ${res.msg}`);
          }
        });
    },
    fail: (err) => {
      console.error('save failed');
    },
});
```

参数	类型	含义
filePath	String	来电铃音本地文件地址
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

enableMuteMode

开启/关闭静音模式。
开启后，收到通话请求，不会播放来电铃声。

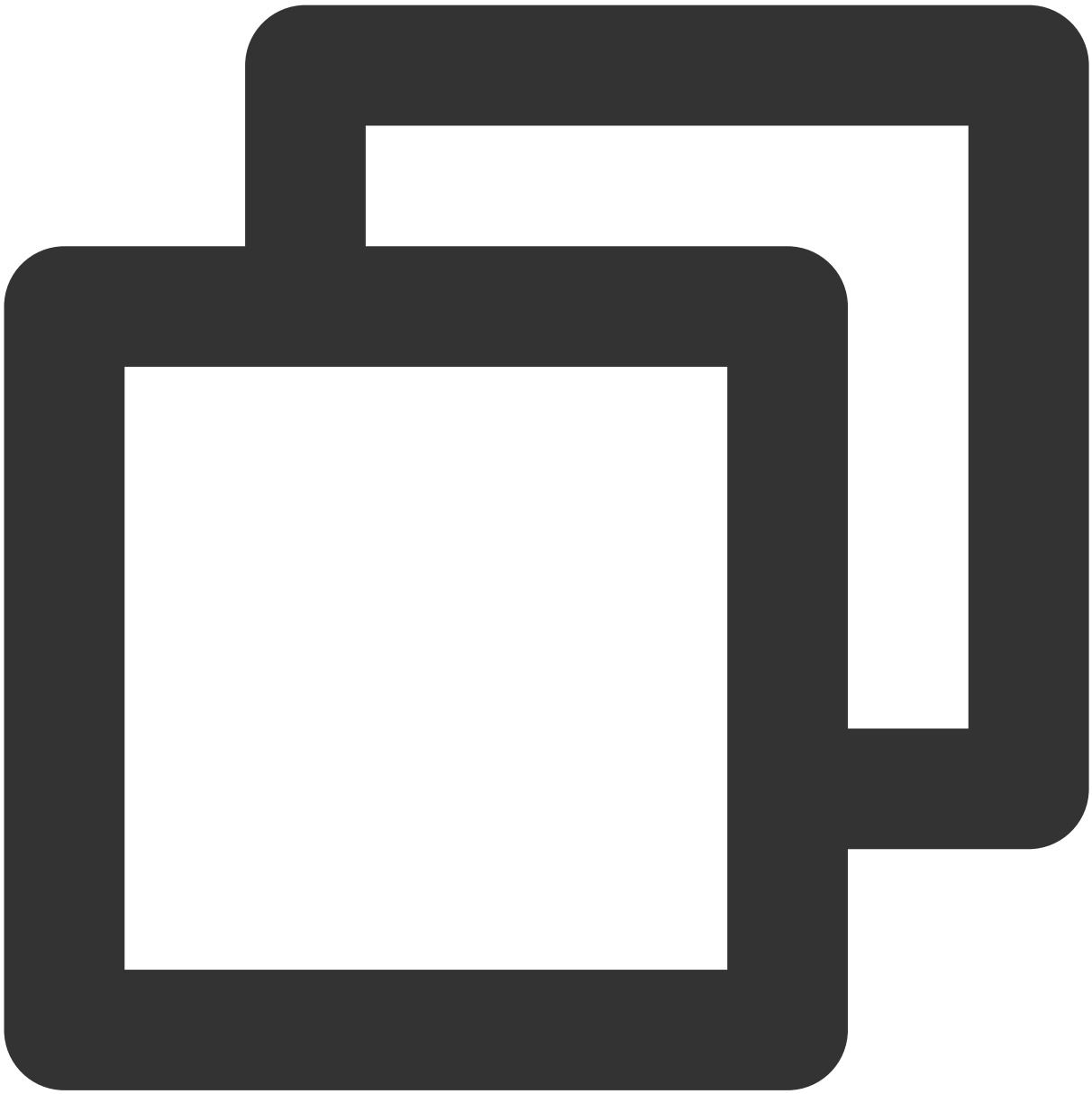


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableMuteMode(enable);
```

参数	类型	含义
enable	Boolean	开启、关闭静音；true 表示开启静音

enableFloatWindow

开启/关闭悬浮窗功能，设置为 false 后，通话界面左上角的悬浮窗按钮会隐藏。



```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

参数	类型	含义
enable	Boolean	开启、关闭悬浮窗功能；true 表示开启浮窗

TUICallEngine

最近更新时间：2024-03-07 10:47:19

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口实现交互。

注意：

需要结合 TencentCloud-TUICallKit 插件使用，不能单独使用。下述接口也是结合客户有自动挂断、设置渲染模式的需求下，才支持的。

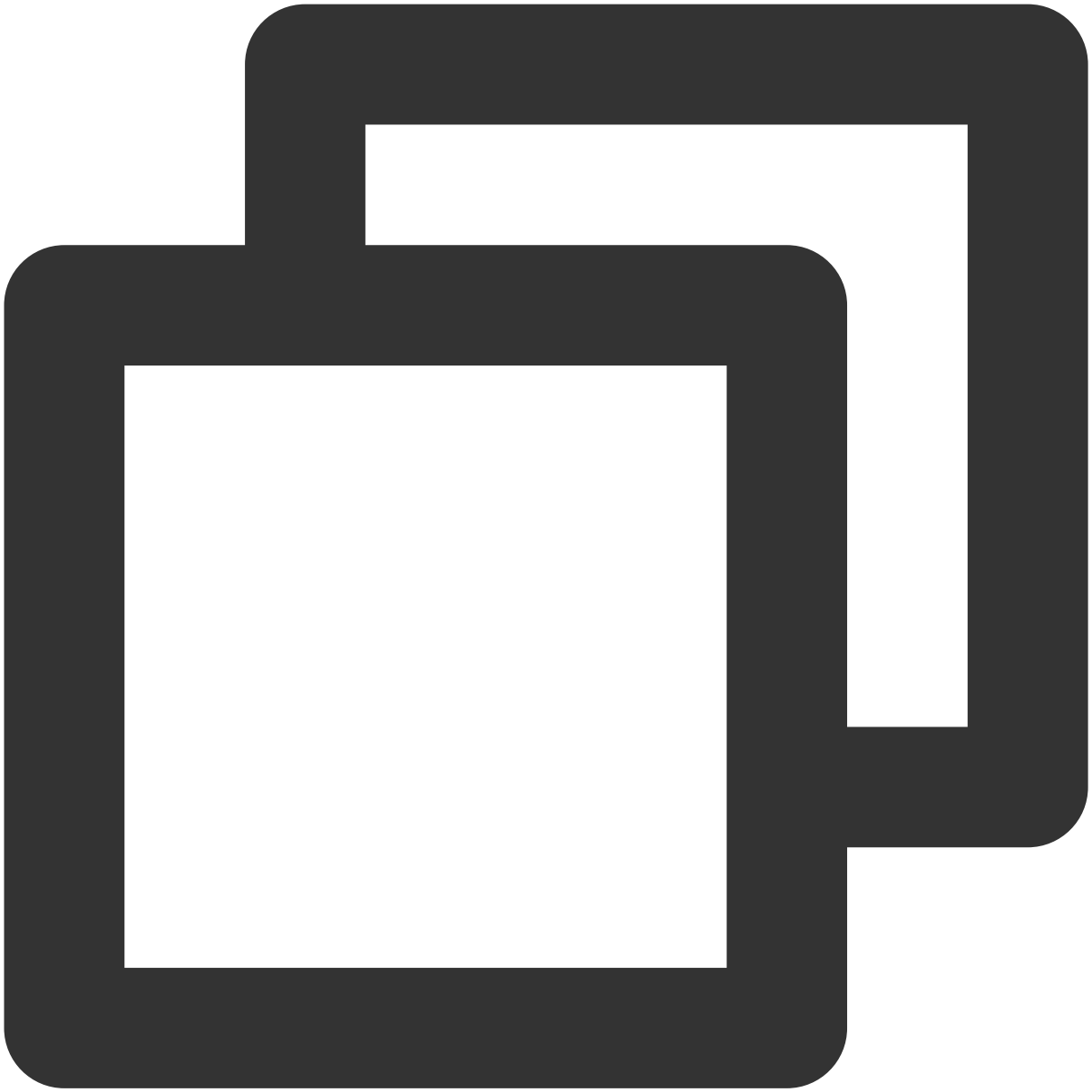
API 概览

API	描述
hangup	结束通话
accept	接听通话
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数

API 详情

hangup

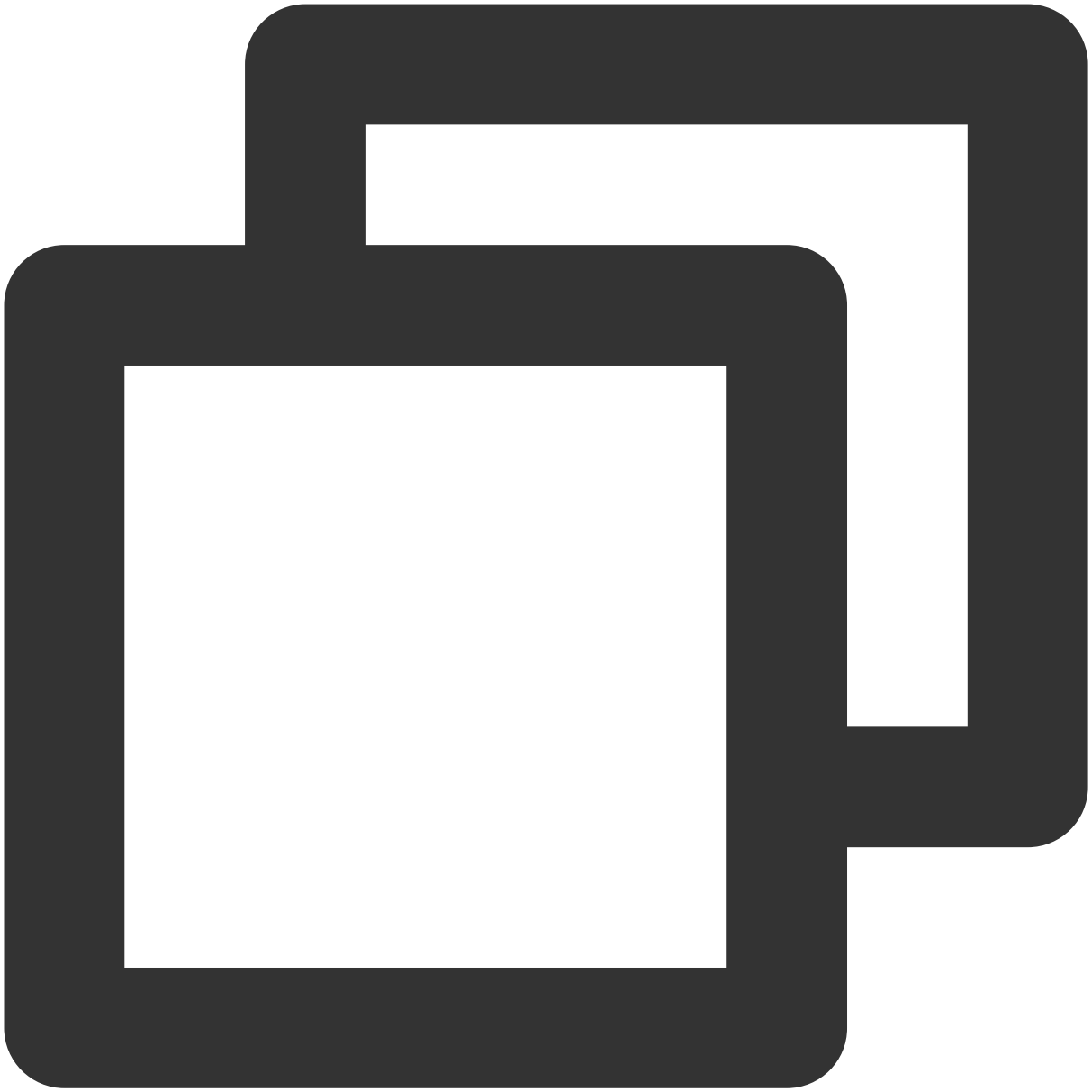
挂断当前通话，当您处于通话中，可以调用该函数结束通话。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.hangup();
```

accept

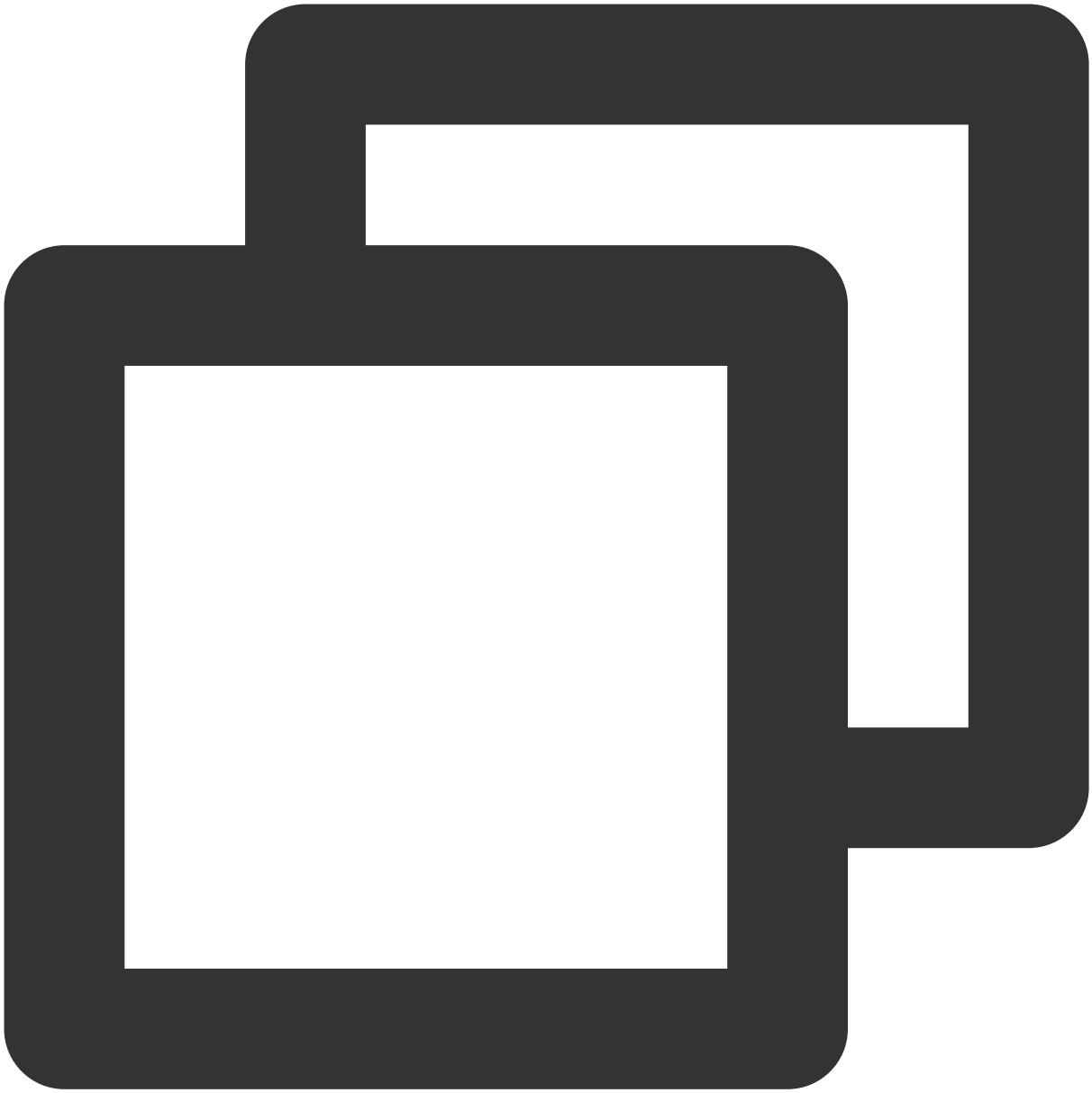
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.accept();
```

setVideoRenderParams

设置用户视频画面的渲染模式。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
const params = {
        userID: '234',
        fillMode: 0, // 0-填充模式，1一适应模式
        rotation: 1, // 0~3分别对应值：Rotation_0, Rotation_90, Rotation_180, Rotatio
};
TUICallEngine.setVideoRenderParams(params, (res) => {
        console.warn('res = ', JSON.stringify(res));
});
```

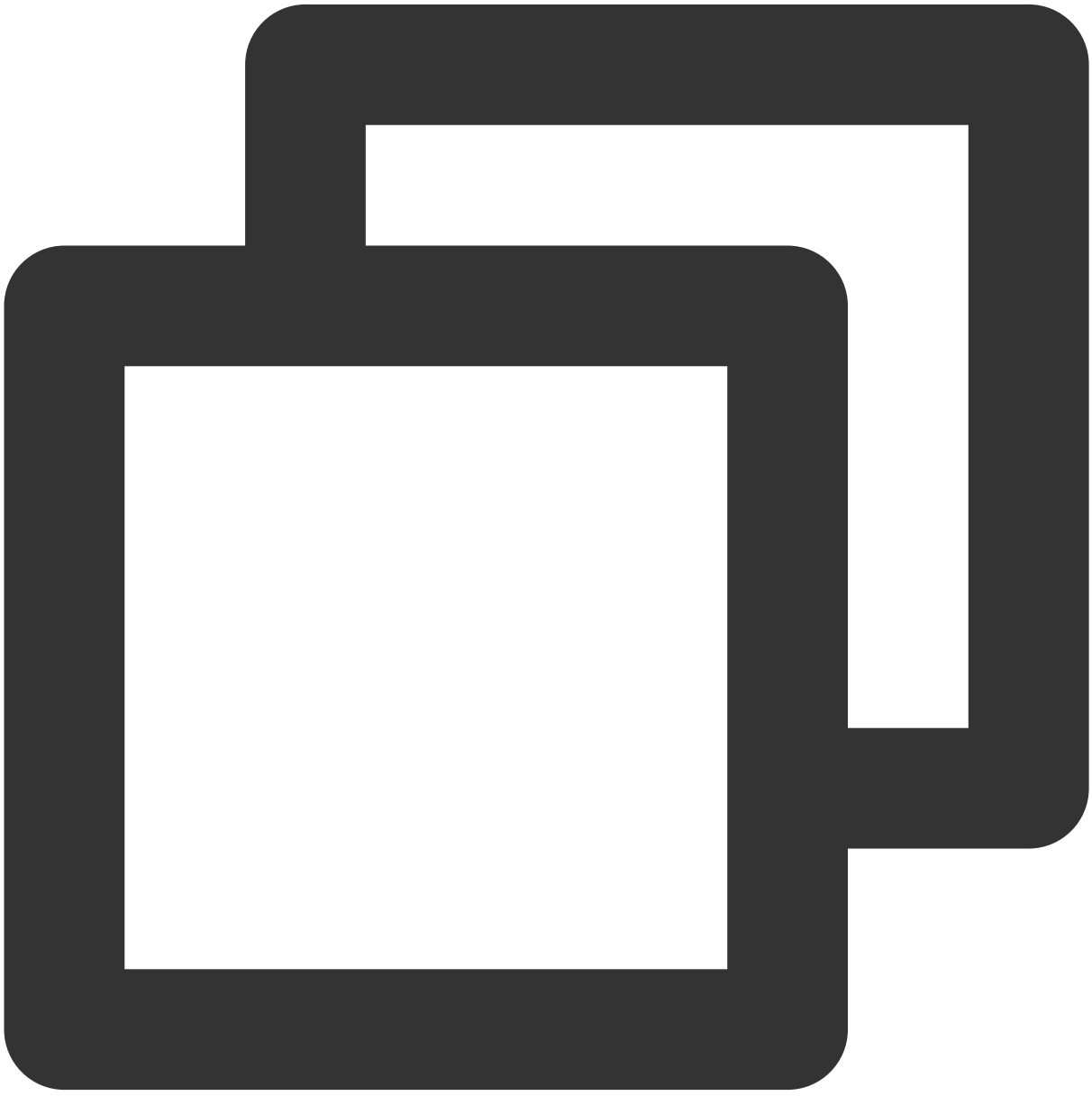
参数如下表所示：

参数	类型	含义
userID	String	目标用户的 userID
params	Object	视频画面渲染参数，例如：画面旋转角度、填充模式

setVideoEncoderParams

设置视频编码的编码参数。

该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
const params = {
        resolution: 108,
        resolutionMode: 0, // 0--横屏, 1--竖屏
};
TUICallEngine.setVideoEncoderParams(params, (res) => {
        console.warn('res = ', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
resolution	Number	视频分辨率 62：宽高比 16:9；分辨率 640x360； 64：宽高比 4:3；分辨率 960x720； 108：宽高比 16:9；分辨率 640x360； 110：宽高比 16:9；分辨率 960x540； 112：宽高比 16:9；分辨率 1280x720； 114：宽高比 16:9；分辨率 1920x1080；
resolutionMode	Number	分辨率模式（横屏分辨率 or 竖屏分辨率） 0：Landscape（横屏） 1：Portrait（竖屏）

事件

最近更新时间：2024-03-07 10:41:58

事件简介

可以监听 TUICallKit 对应的事件，进行提示或其它交互。

回调事件概览

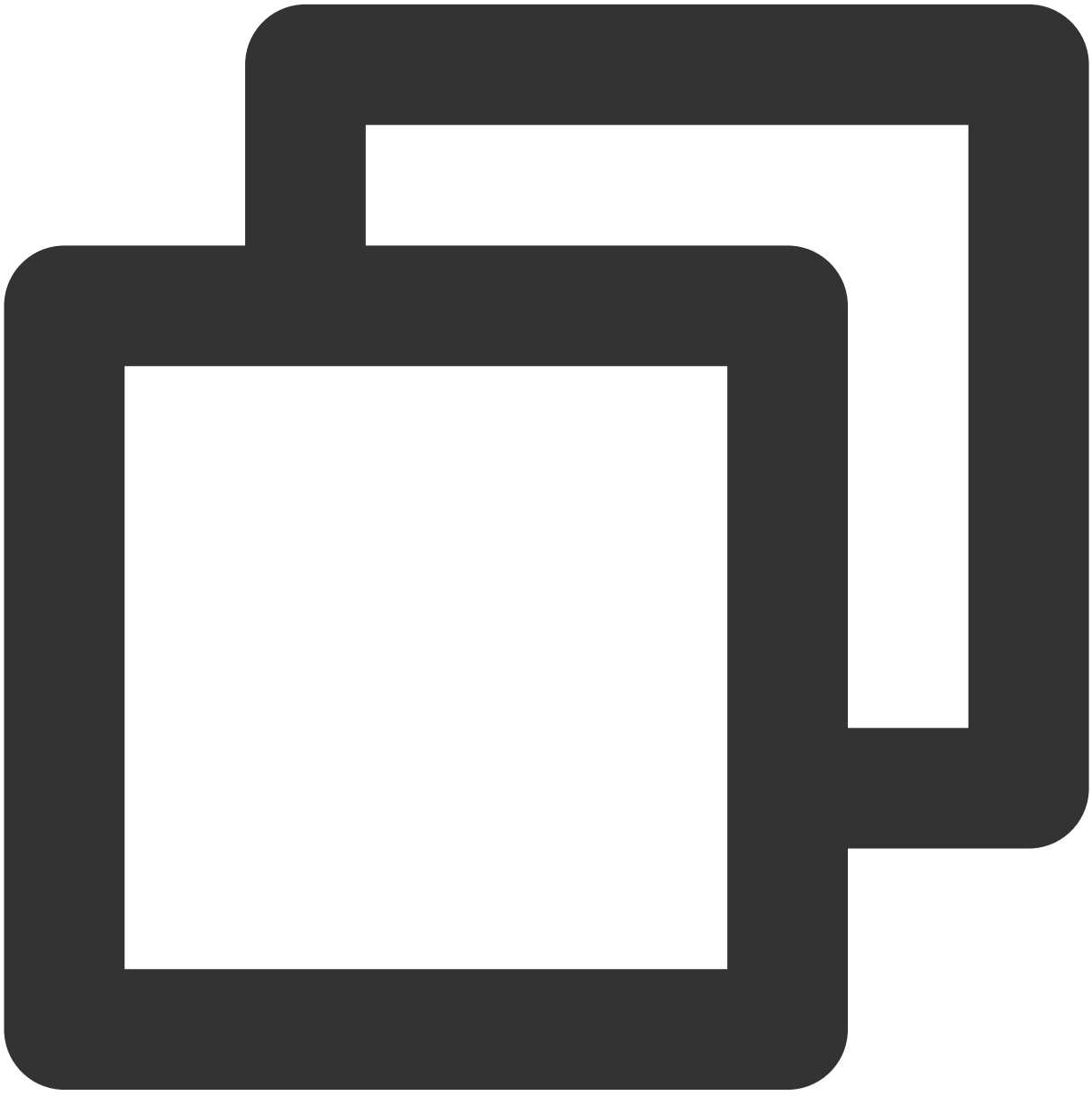
API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线

版权所有：腾讯云计算（北京）有限责任公司

onUserSigExpired	在线时票据过期

回调事件详情

通过 globalEvent 监听原生插件抛出的事件。



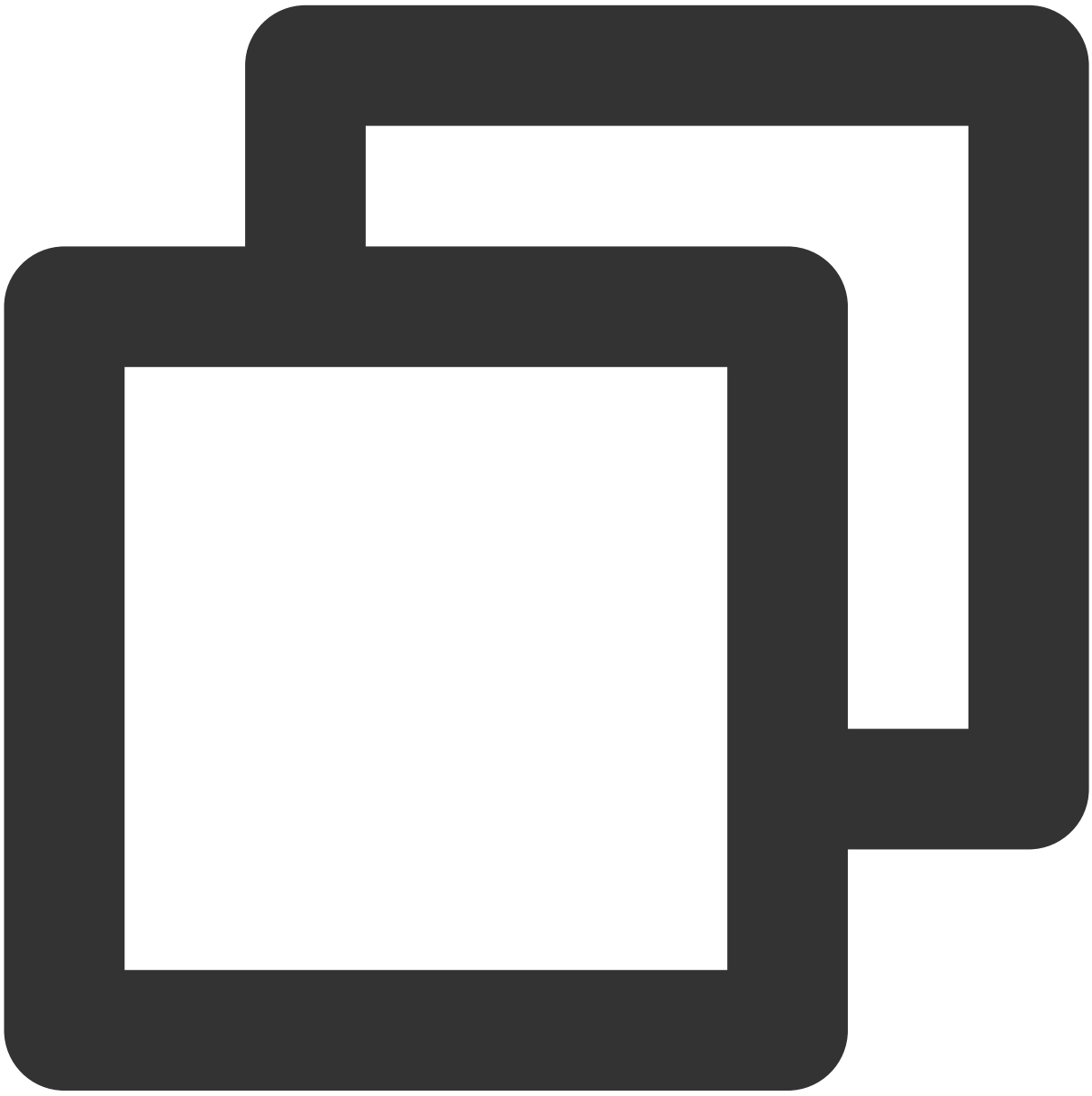
```
const TUICallingEvent = uni.requireNativePlugin('globalEvent');
```

onError

错误回调。

说明

SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。



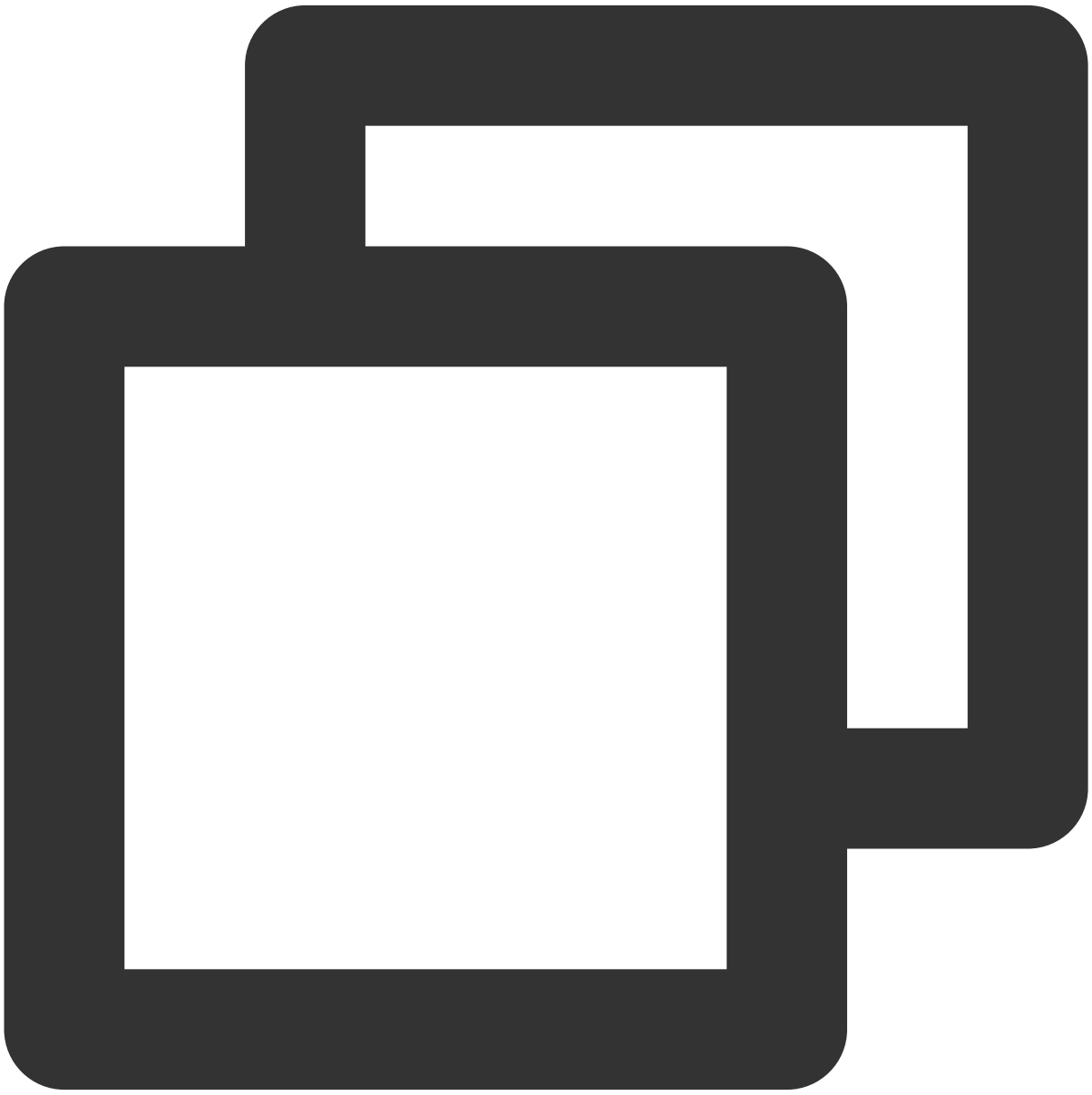
```
TUICallingEvent.addEventListener('onError', (res) => {
  console.log('onError', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	错误回调参数
res.code	Number	错误码
res.msg	String	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。



```
TUICallingEvent.addEventListener('onCallReceived', (res) => {
  console.log('onCallReceived', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.callerId	String	主叫 ID（邀请方）
res.calleeIdList	Array<String>	被叫 ID 列表（被邀请方）
res.groupId	String	群通话 ID。在群组通话时返回，1v1 返回 groupId 为空字符串
res.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.userData	String	用户增加的扩展字段

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

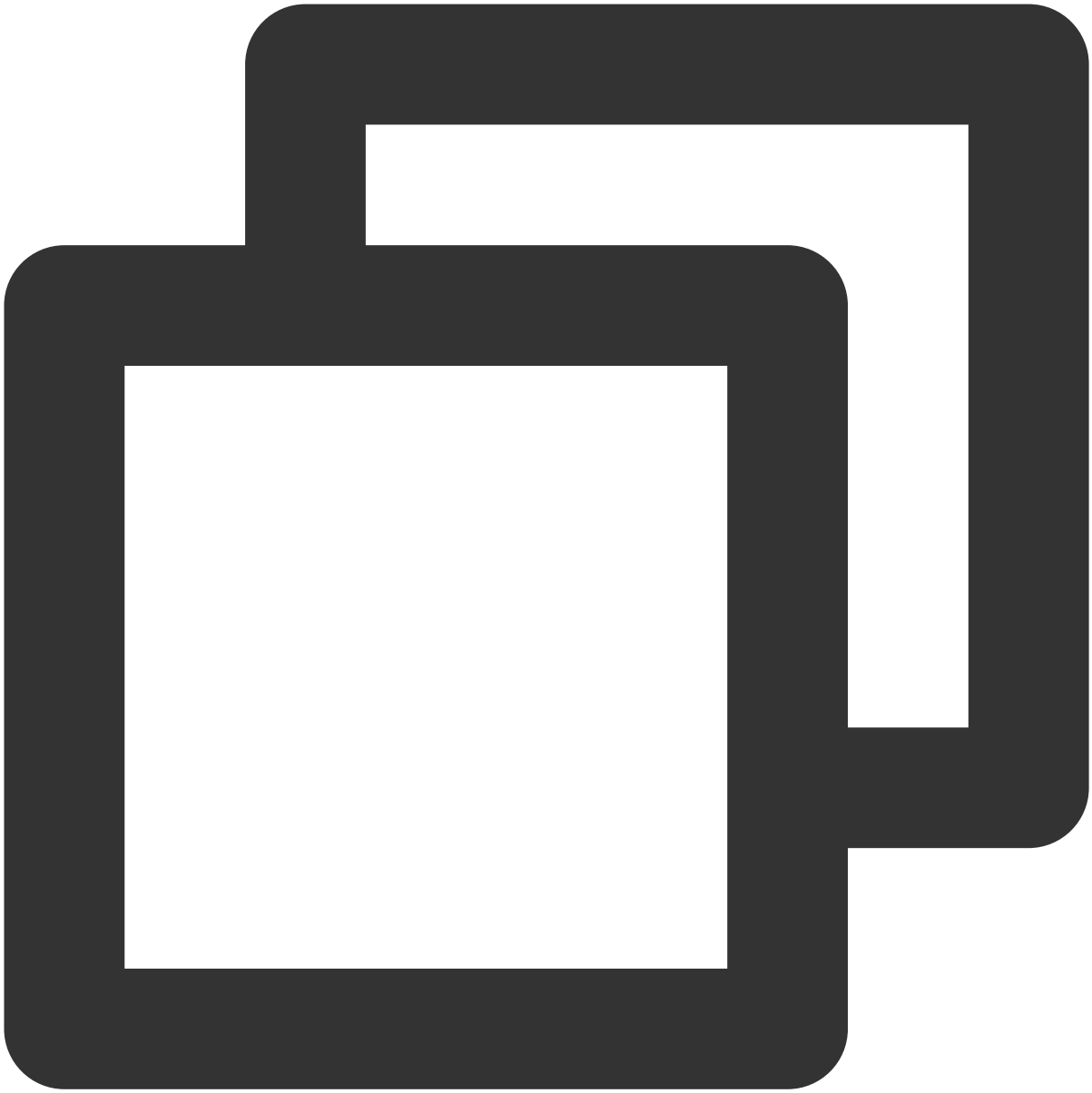
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。

被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）。

被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。

被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）。

异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。



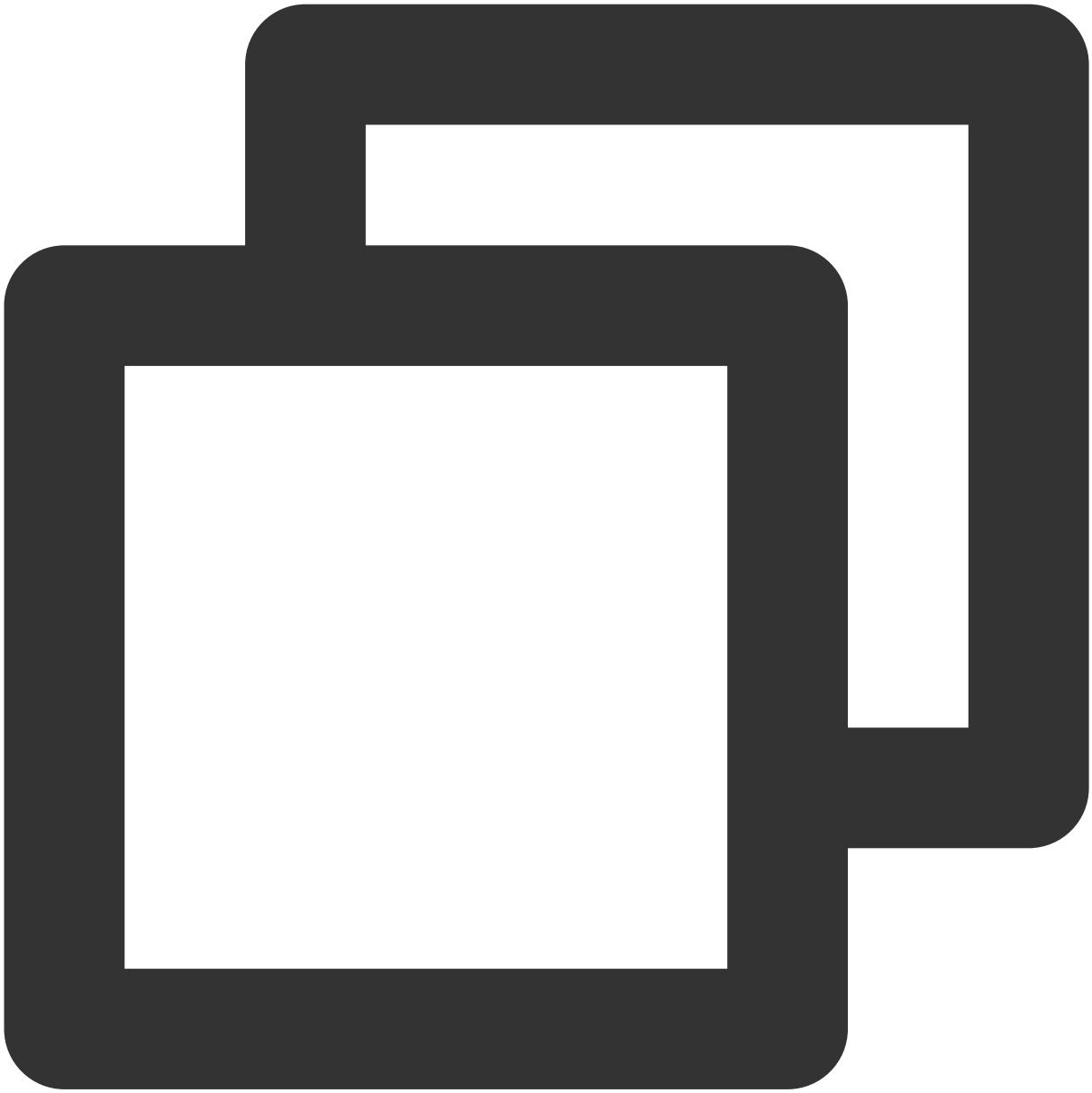
```
TUICallingEvent.addEventListener('onCallCancelled', (res) => {
  console.log('onCallCancelled', res);
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	用户的 ID

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。



```
TUICallingEvent.addEventListener('onCallBegin', (res) => {
  console.log('onCallBegin', JSON.stringify(res));
});
```

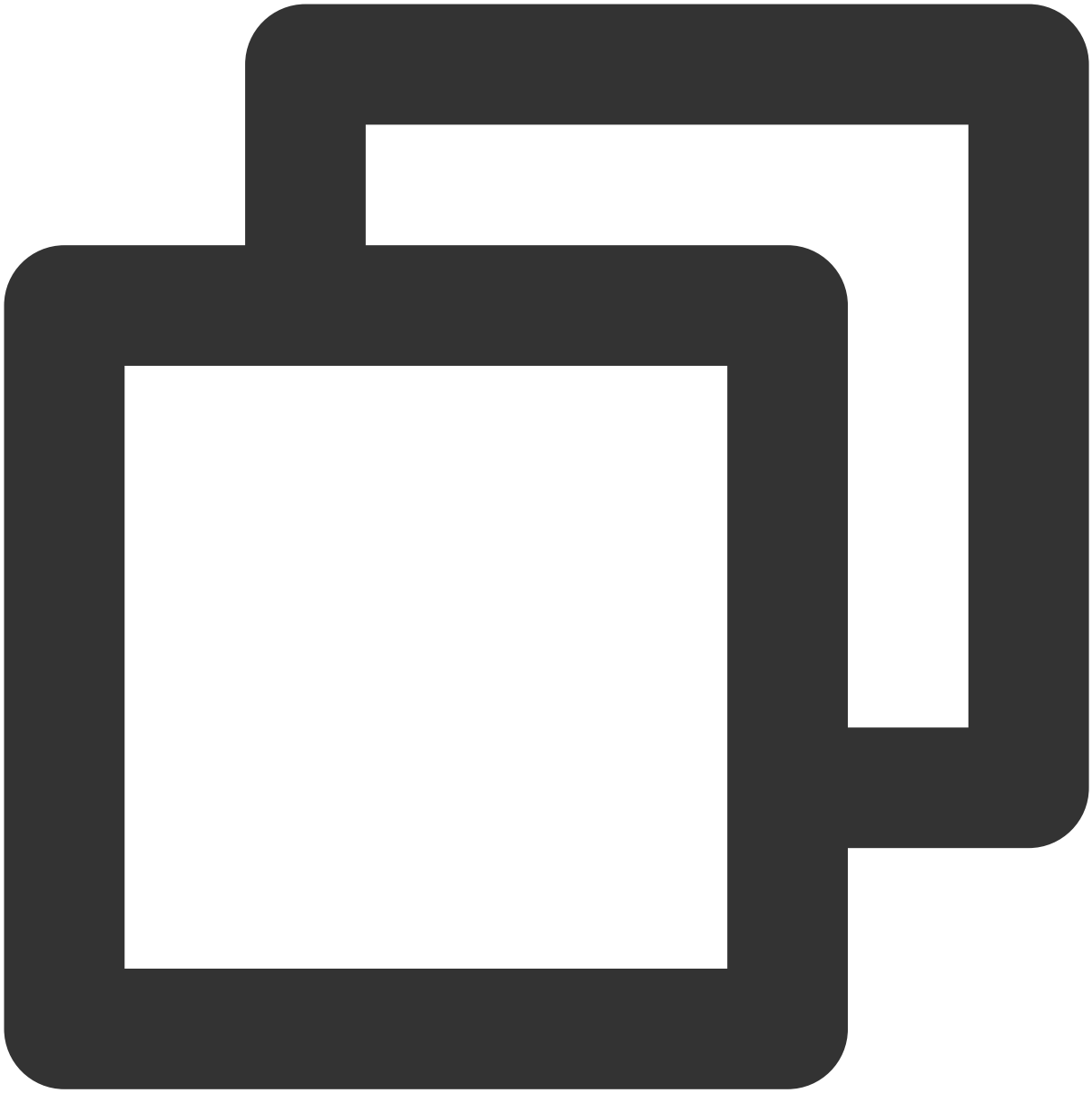
参数如下表所示：

参数	类型	含义

res	Object	回调参数
res.roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
res.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)

onCallEnd

表示通话结束，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。



```
TUICallingEvent.addEventListener('onCallEnd', (res) => {
  console.log('onCallEnd', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串

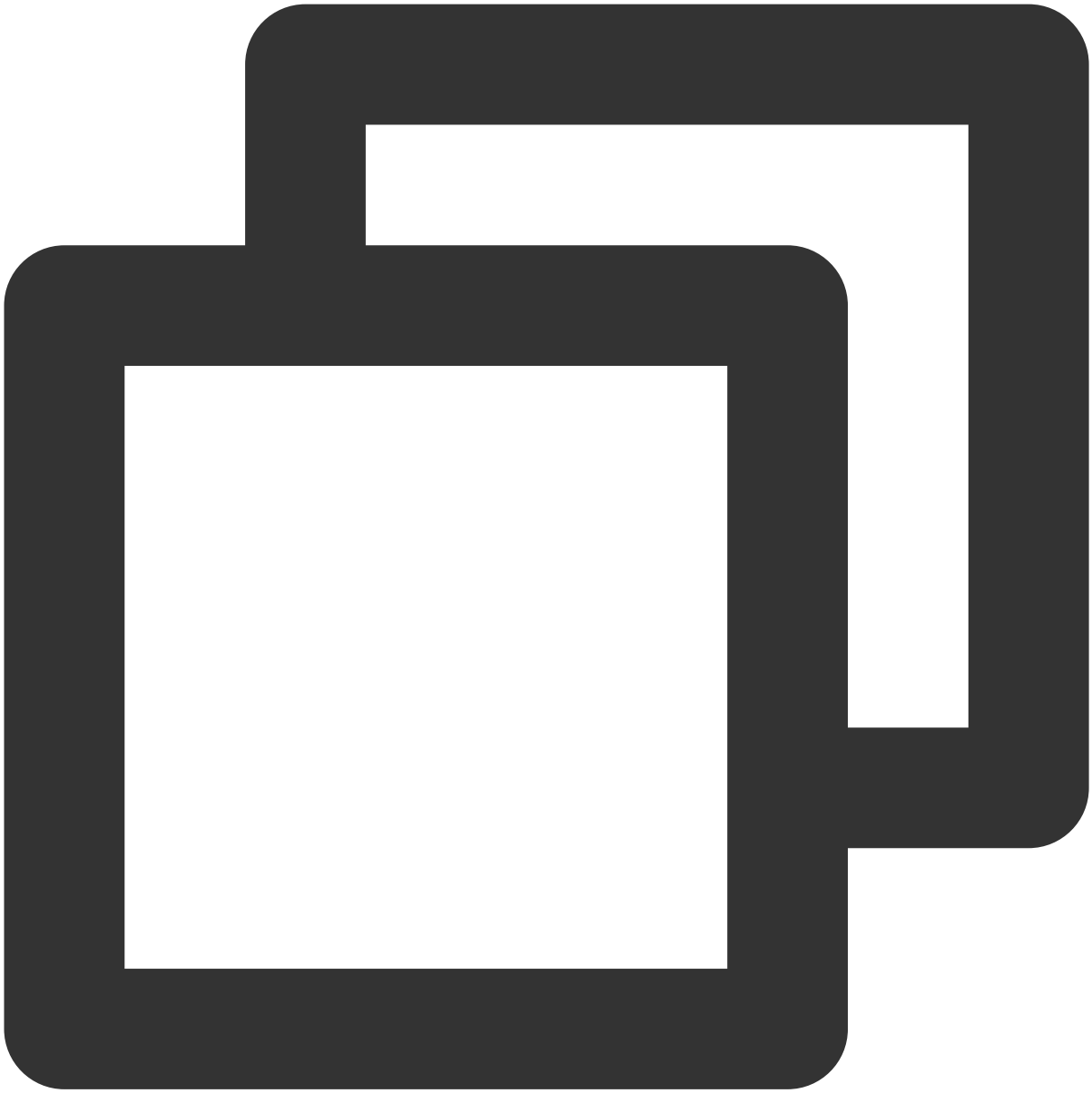
		房间号
res.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)
res.totalTime	Number	此次通话的时长，单位：秒

注意

客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。



```
TUICallingEvent.addEventListener('onCallMediaTypeChanged', (res) => {
  console.log('onCallMediaTypeChanged', JSON.stringify(res));
});
```

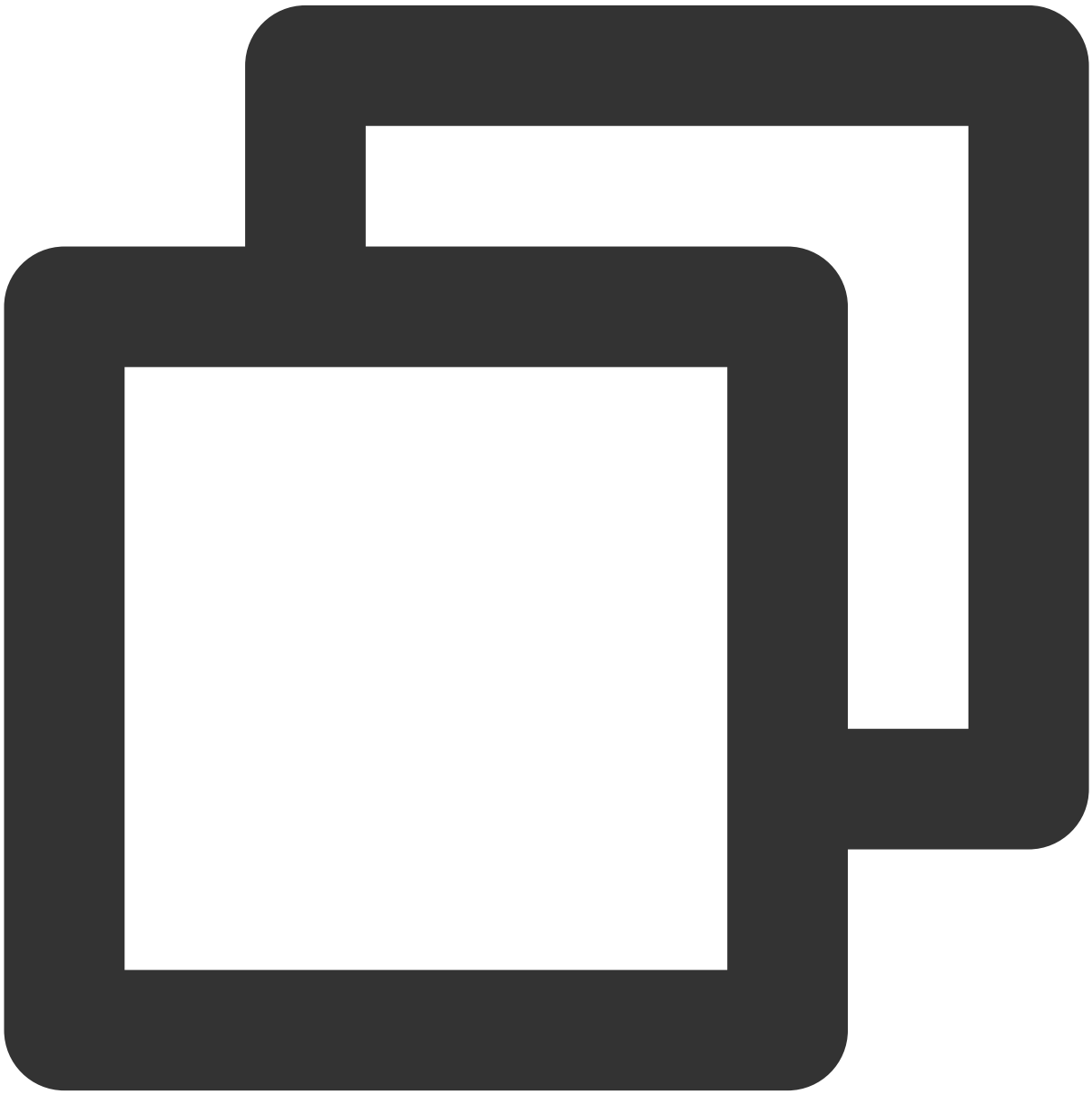
参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.oldCallMediaType	Number	旧的通话类型，比如：语音通话(callMediaType = 1)、视频通话

		(callMediaType = 2)
res.newCallMediaType	Number	新的通话类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。



```
TUICallingEvent.addEventListener('onUserReject', (res) => {
  console.log('onUserReject', JSON.stringify(res));
```

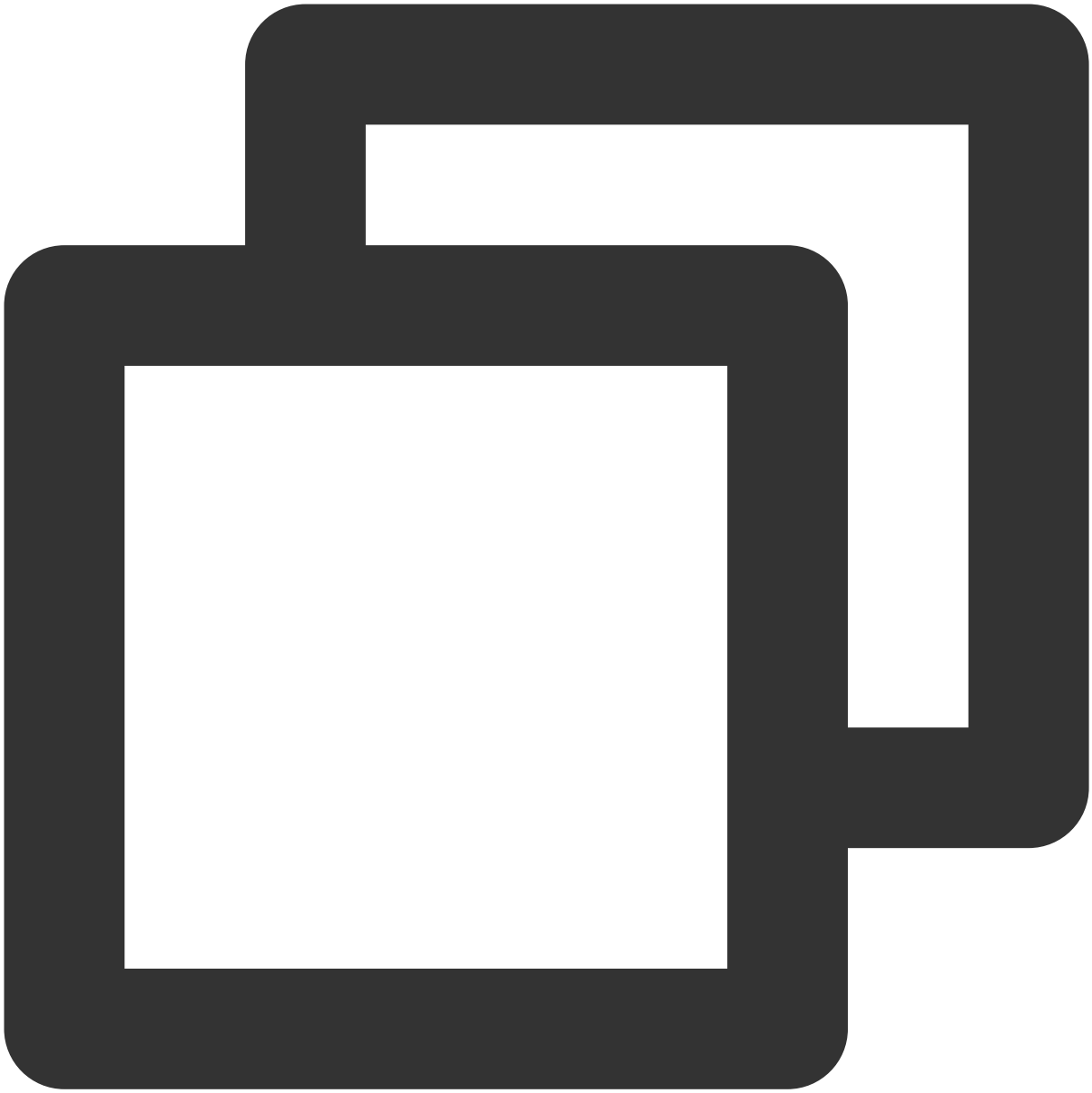
```
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。



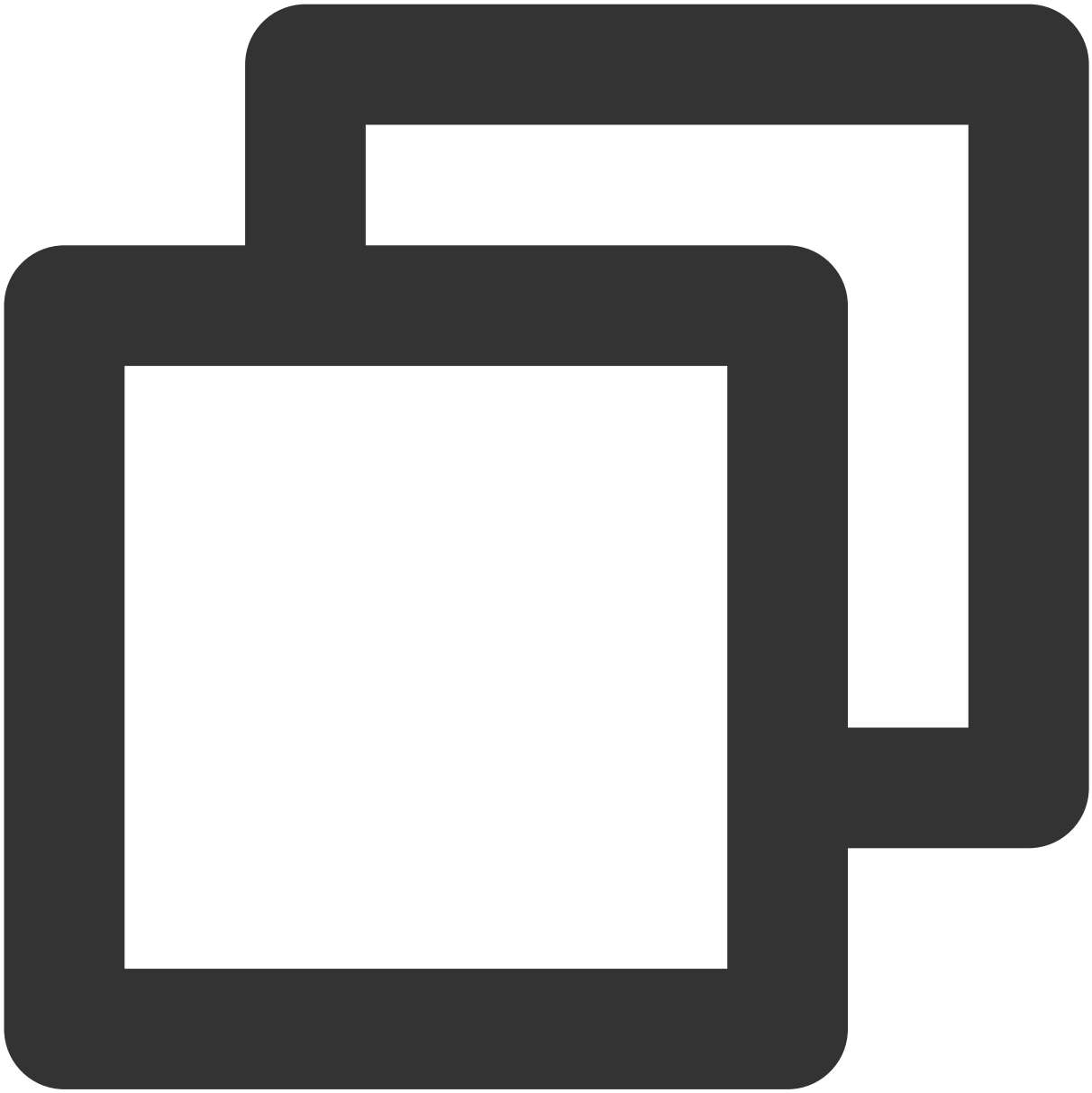
```
TUICallingEvent.addEventListener('onUserNoResponse', (res) => {
  console.log('onUserNoResponse', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	无响应用户的 ID

onUserLineBusy

通话忙线回调。



```
TUICallingEvent.addEventListener('onUserLineBusy', (res) => {
  console.log('onUserLineBusy', JSON.stringify(res));
});
```

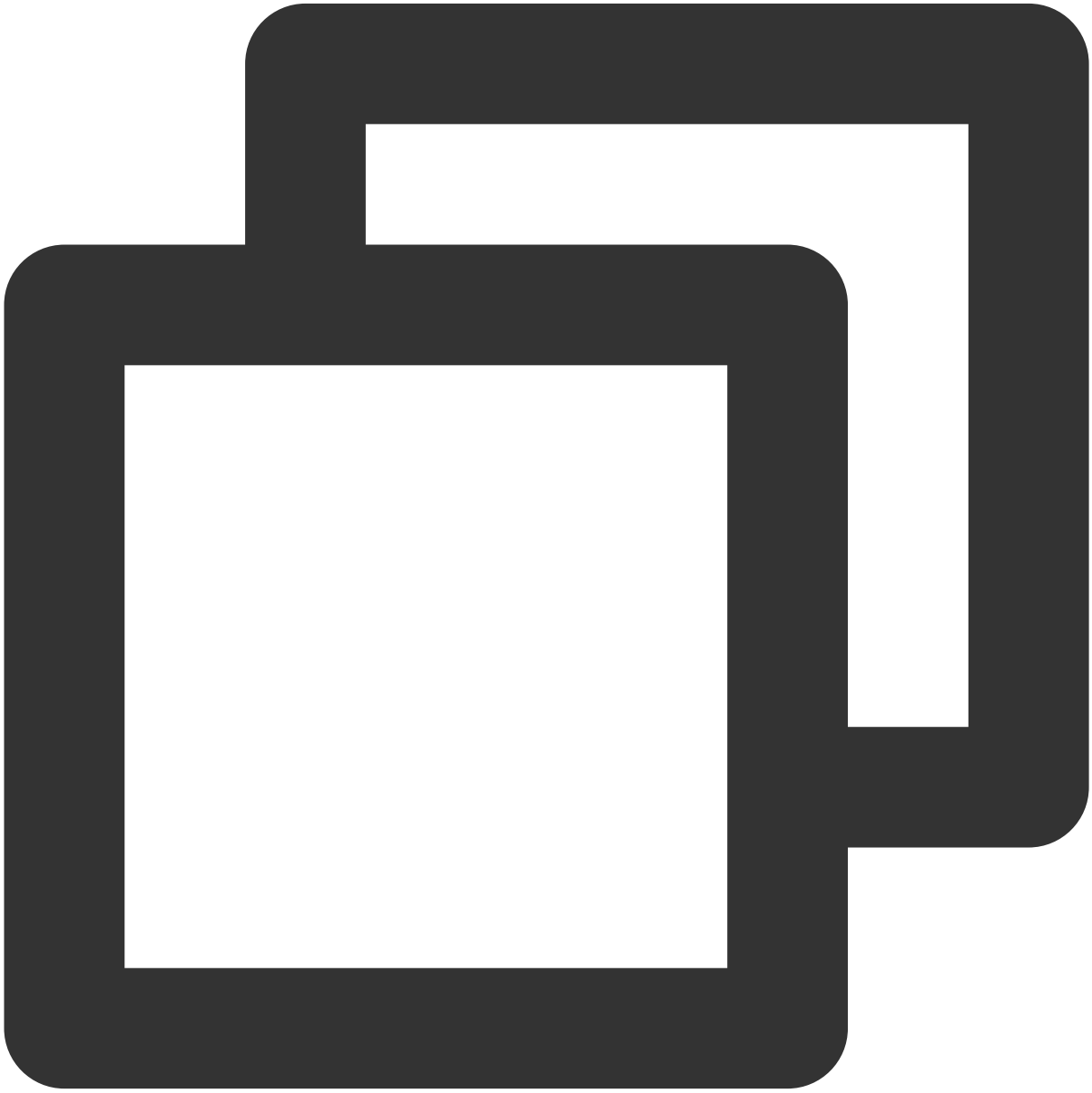
参数如下表所示：

参数	类型	含义

res	Object	回调参数
res.userId	String	忙线用户的 ID

onUserJoin

有用户进入此次通话的回调。



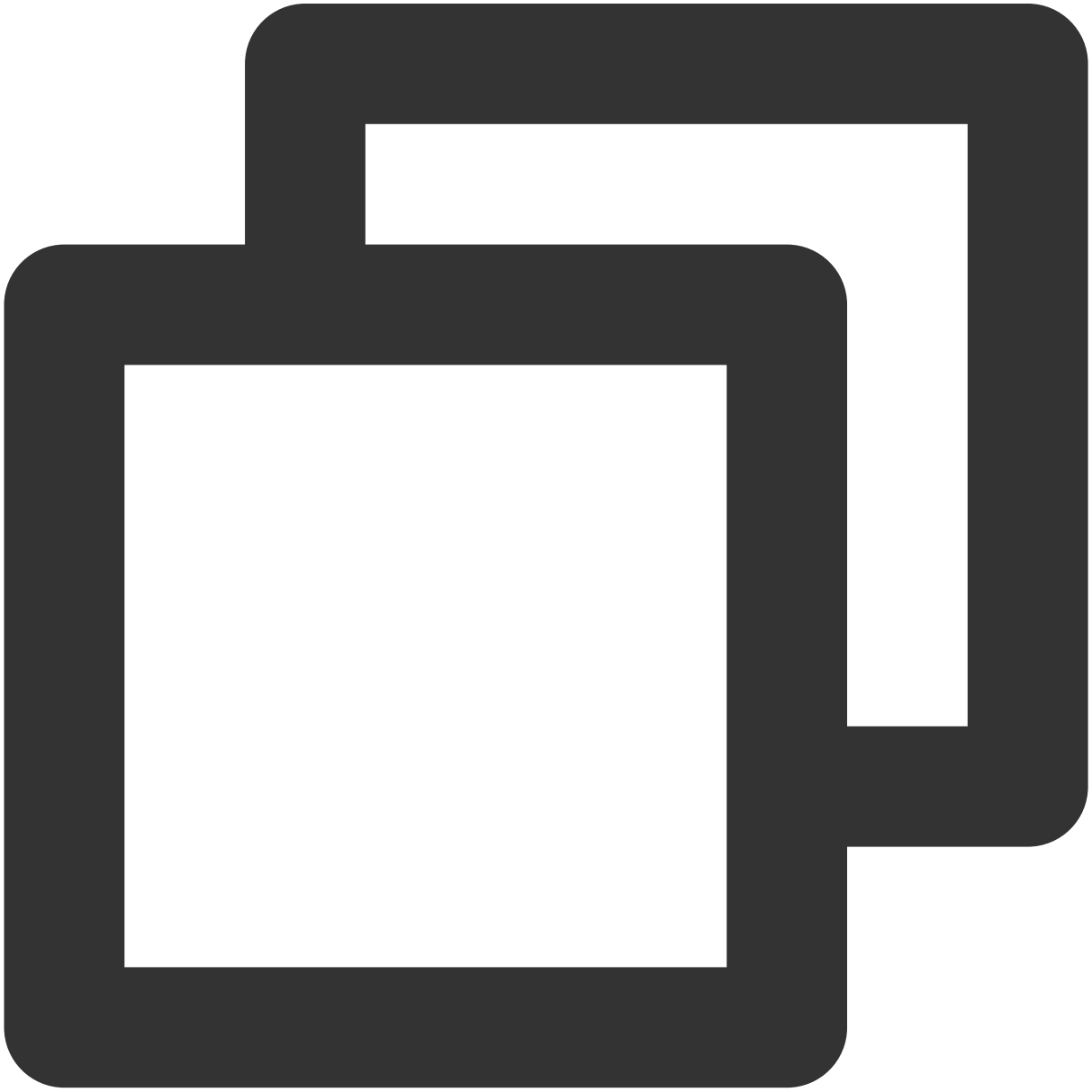
```
TUICallingEvent.addEventListener('onUserJoin', (res) => {
  console.log('onUserJoin', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。



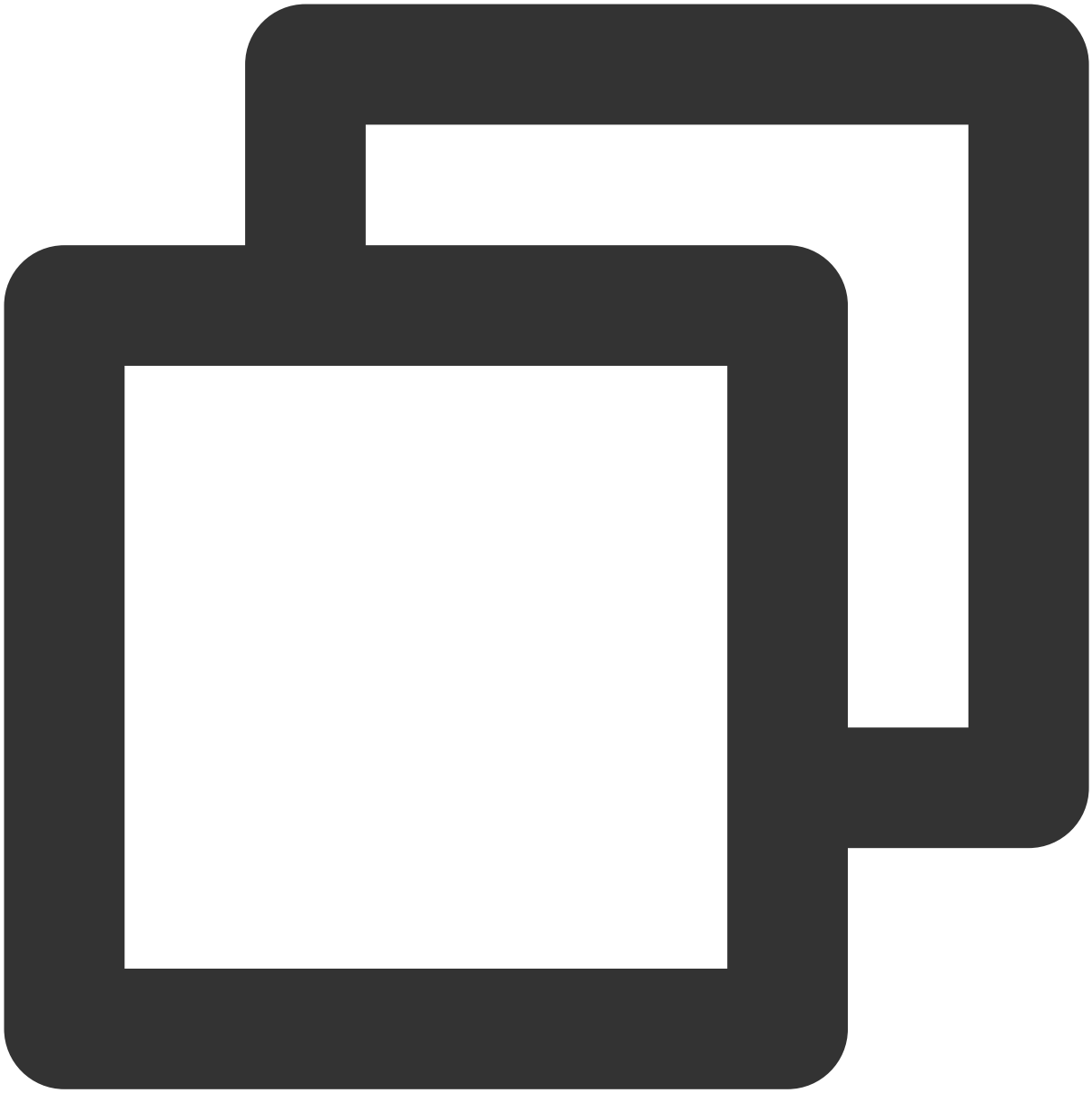
```
TUICallingEvent.addEventListener('onUserLeave', (res) => {
  console.log('onUserLeave', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。



```
TUICallingEvent.addEventListener('onUserVideoAvailable', (res) => {
  console.log('onUserVideoAvailable', JSON.stringify(res));
});
```

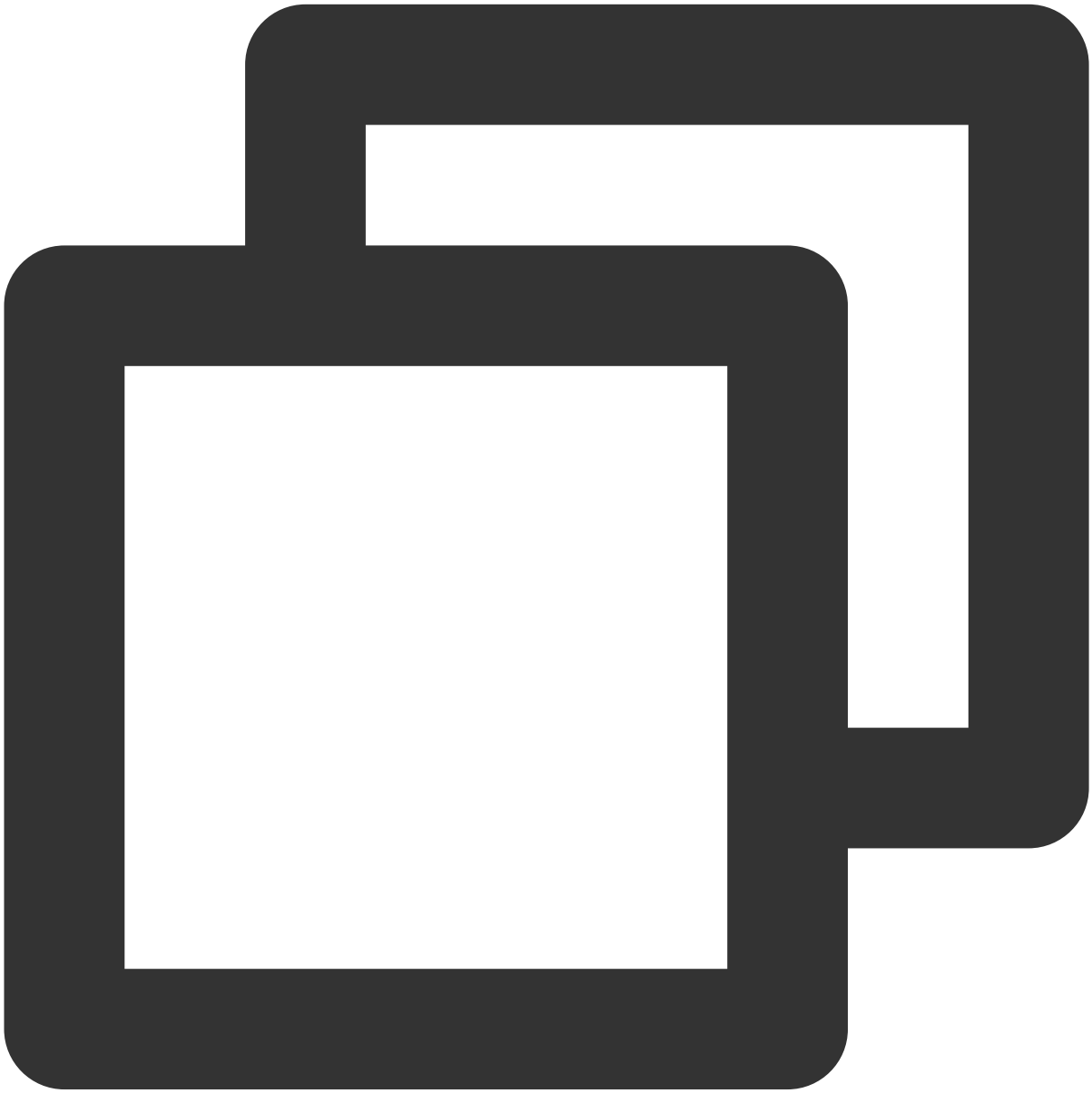
参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	通话用户 ID

res.isVideoAvailable	boolean	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。



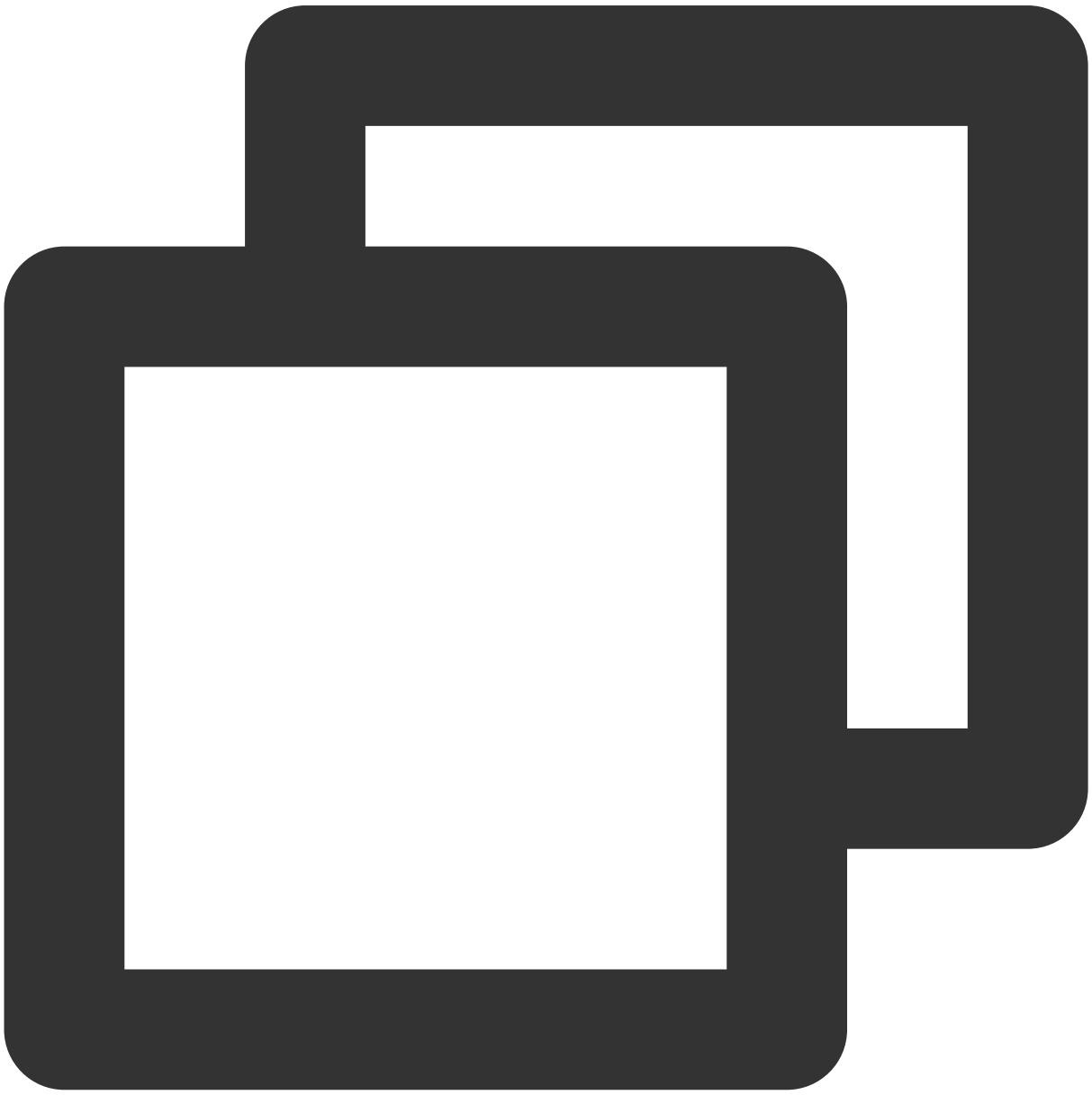
```
TUICallingEvent.addEventListener('onUserAudioAvailable', (res) => {
  console.log('onUserAudioAvailable', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	用户 ID
res.isAudioAvailable	boolean	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。



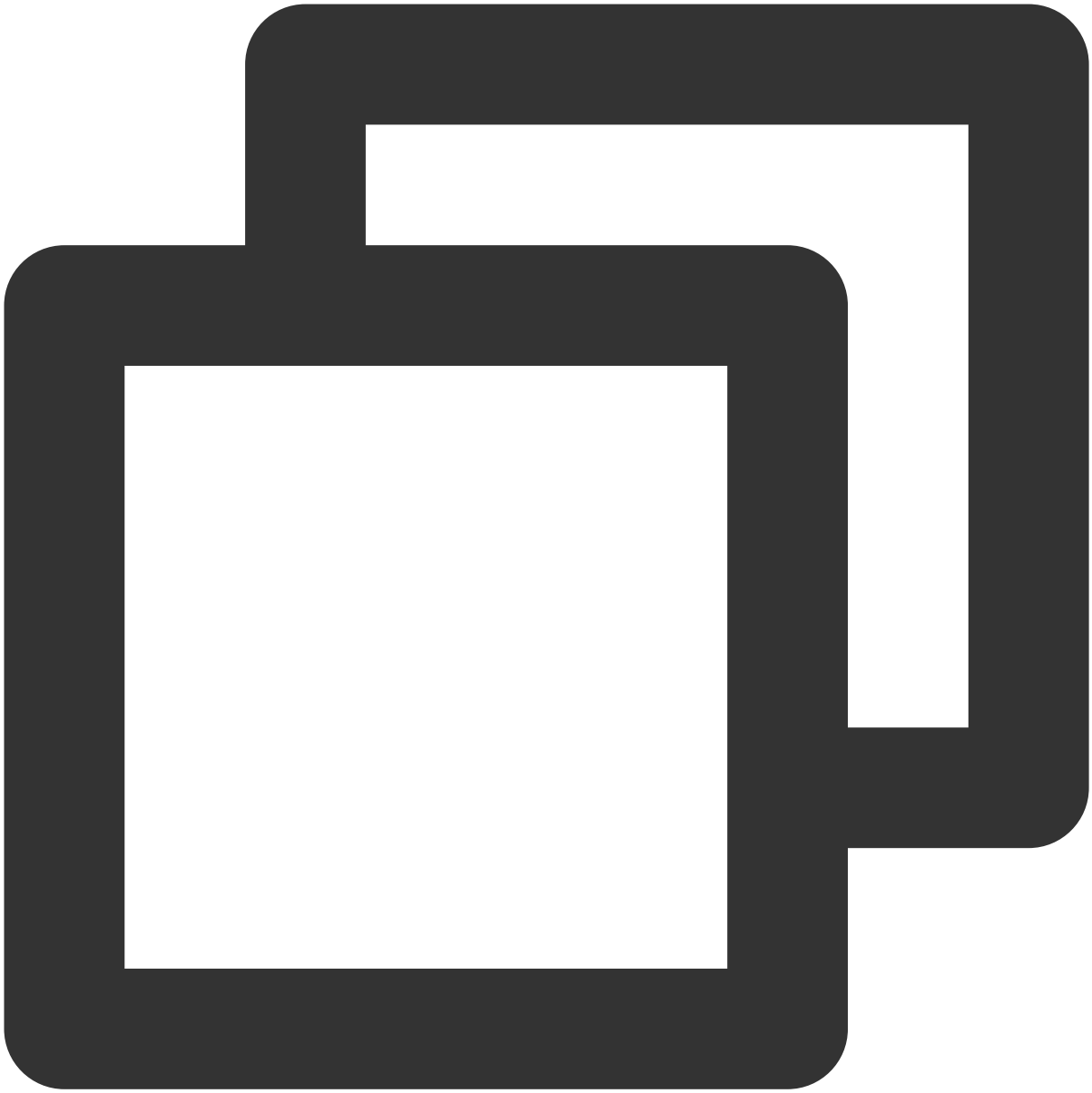
```
TUICallingEvent.addEventListener('onUserVoiceVolumeChanged', (res) => {
  console.log('onUserVoiceVolumeChanged', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.volumeMap	Map<String, Integer>	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。



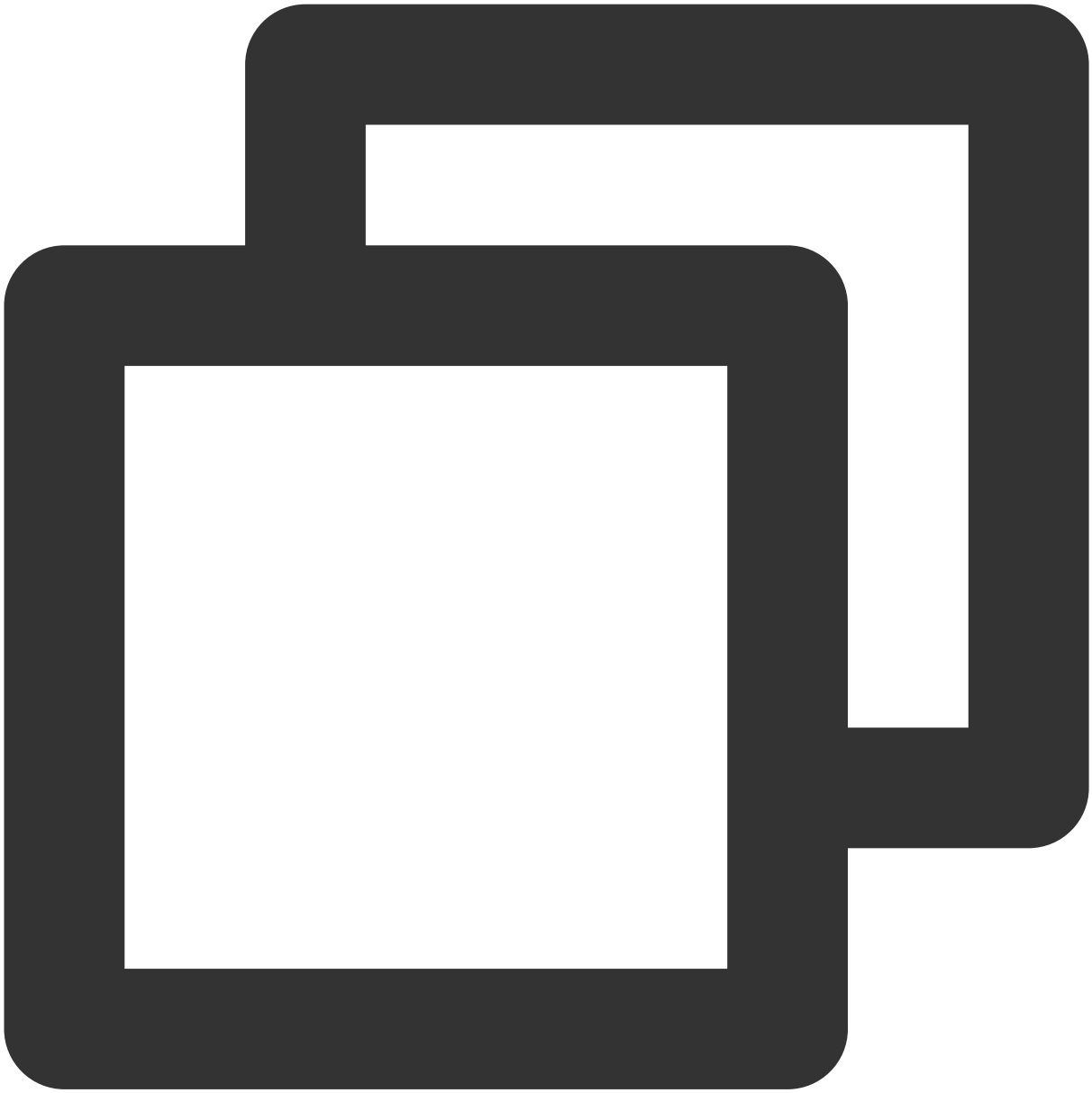
```
TUICallingEvent.addEventListener('onUserNetworkQualityChanged', (res) => {
  console.log('onUserNetworkQualityChanged', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.networkQualityList	List	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

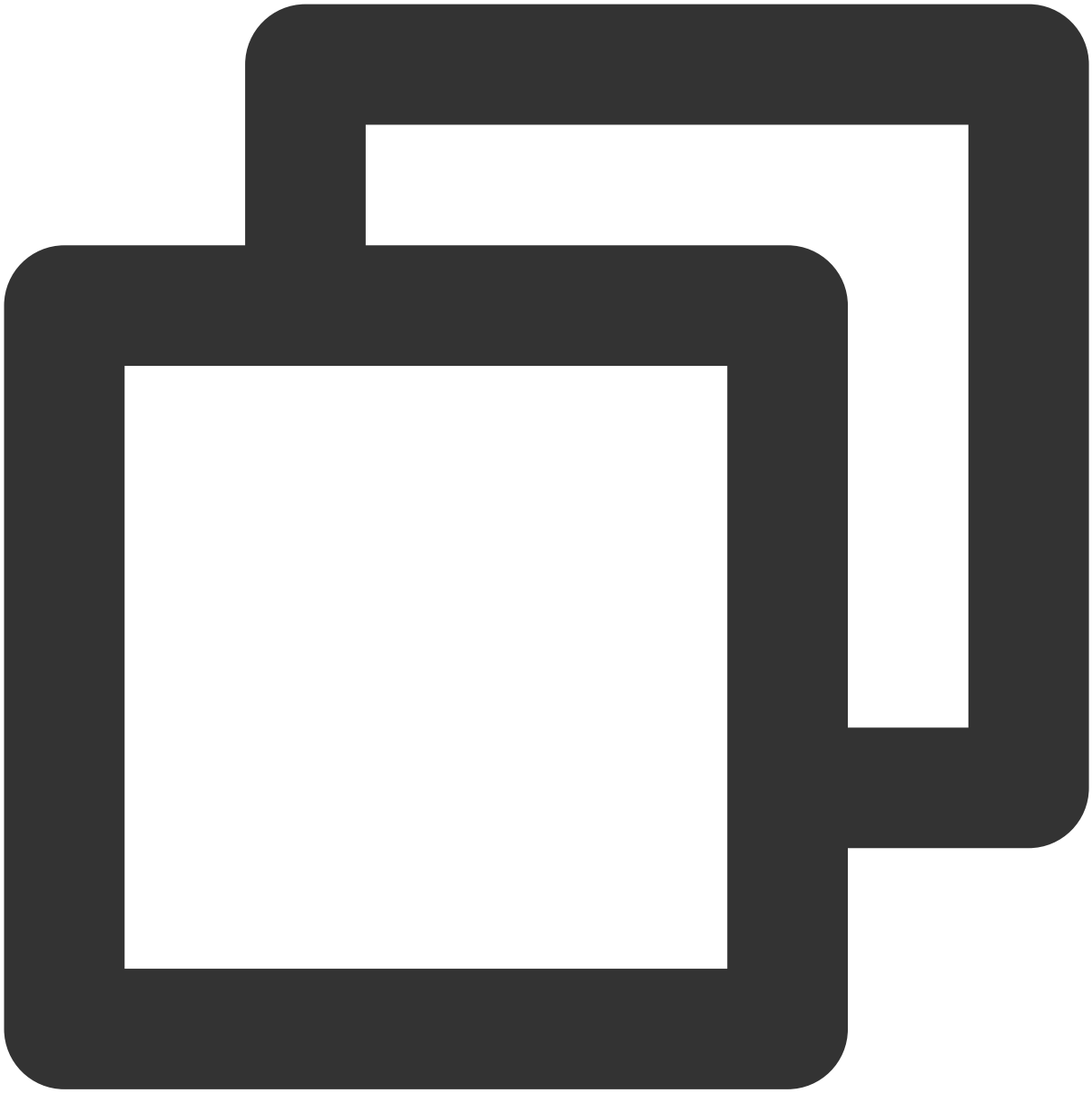
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
TUICallingEvent.addEventListener('onKickedOffline', (res) => {
  console.log('onKickedOffline', JSON.stringify(res));
});
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。



```
TUICallingEvent.addEventListener('onUserSigExpired', (res) => {
  console.log('onUserSigExpired', JSON.stringify(res));
});
```

错误码表

最近更新时间：2023-06-16 16:30:17

通知用户在使用音视频通话过程中出现的警告和错误。

TUICallDefine 错误码

定义	取值	描述
ERROR_PACKAGE_NOT_PURCHASED	-1001	您当前未购买音视频通话能力套餐，请前往IM控制台 开通免费体验 或 加购正式版
ERROR_PACKAGE_NOT_SUPPORTED	-1002	您当前购买的音视频通话能力套餐不支持该功能，建议您前往IM控制台 升级套餐类型
ERROR_TIM_VERSION_OUTDATED	-1003	IM SDK 版本过低，请升级 IM SDK 版本 >= 6.6；在build.gradle文件中查找并修改："com.tencent.imsdk:imsdk-plus:7.1.3925"
ERROR_PERMISSION_DENIED	-1101	获取权限失败，当前未授权音/ 视频权限，请查看是否开启设备权限
ERROR_GET_DEVICE_LIST_FAIL	-1102	获取设备列表失败（仅 web 平台支持）
ERROR_INIT_FAIL	-1201	未调用 init 方法进行初始化，TUICallEngine API 需在初始化之后使用
ERROR_PARAM_INVALID	-1202	参数错误
ERROR_REQUEST_REFUSED	-1203	当前状态不支持调用
ERROR_REQUEST_REPEATED	-1204	当前方法正在执行中，请勿重复调用
ERROR_SCENE_NOT_SUPPORTED	-1205	当前通话场景，不支持该功能
ERROR_SIGNALING_SEND_FAIL	-1406	信令发送失败，具体失败原因可查看具体方法回调中的 errorMsg

IM 错误码

音视频通话使用了腾讯云 IM SDK 作为通信的基础服务，比如通话拨打信令、通话忙线信令等核心逻辑。常见错误码如下：

错误码	描述
6014	IM SDK 未登录，请先登录，成功回调之后重试，或者已被踢下线，可使用 TIMManager getLoginUser 检查当前是否在线。
6017	参数无效，请检查参数是否符合要求，具体可查看错误信息进一步定义哪个字段。
6206	UserSig 过期，请重新获取有效的 UserSig 后再重新登录，获取方法请参见 生成 UserSig。
7013	套餐包不支持该接口的使用，请升级到旗舰版套餐。
8010	信令请求 ID 无效或已经被处理过。

说明：

更多 IM SDK 错误码详见：IM错误码

TRTC 错误码

音视频通话使用了腾讯云 TRTC SDK 作为音频和视频的基础服务，比如开关摄像头、开关麦克风等核心逻辑。常见错误码如下：

枚举	取值	描述
ERR_CAMERA_START_FAIL	-1301	打开摄像头失败，例如在 Windows 或 Mac 设备，摄像头的配置程序（驱动程序）异常，禁用后重新启用设备，或者重启机器，或者更新配置程序
ERR_CAMERA_NOT_AUTHORIZED	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了
ERR_CAMERA_SET_PARAM_FAIL	-1315	摄像头参数设置出错（参数不支持或其它）
ERR_CAMERA_OCCUPY	-1316	摄像头正在被占用中，可尝试打开其他摄像头
ERR_MIC_START_FAIL	-1302	打开麦克风失败，例如在 Windows 或 Mac 设备，麦克风的配置程序（驱动程序）异常，禁用后重新启用设备，或者重启机器，或者更新配置程序
ERR_MIC_NOT_AUTHORIZED	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了

版权所有：腾讯云计算（北京）有限责任公司

ERR_MIC_SET_PARAM_FAIL	-1318	麦克风设置参数失败
ERR_MIC_OCCUPY	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败
ERR_TRTC_ENTER_ROOM_FAILED	-3301	进入房间失败，请查看 onError 中的 -3301 对应的 msg 提示确认失败原因

说明：

更多 TRTC SDK 错误码详见：TRTC错误码

发布日志（TUICallKit）

Web

最近更新时间：2024-04-03 18:02:21

说明：

TUICallKit Vue3 Github 版本地址：Github TUICallKit Web。

Version 3.2.1 @2024.03.08

新增

language 日志上报。

Version 3.2.0 @2024.02.23

新增

增加默认的离线推送参数。

修复

修复群组通话没有昵称问题。

Version 3.1.9 @2024.01.30

修复

修复群组通话没有展示用户信息问题。
修复选人组件在没有可选成员时，“确认”按钮仍然可点击的问题。
修复通话中关闭麦克风，再次通话不上行音频流问题（升级 trtc-cloud-js-sdk 到 v2.2.7+）。

Version 3.1.8 @2024.01.19

修复

修复选人组件样式对页面的影响。

Version 3.1.7 @2024.01.12

修复

增加接口的重试机制，修复播放时因找不到 dom 节点导致播放失败问题。
修复 PC 上设备列表选择样式问题。

Version 3.1.6 @2023.12.29

优化

优化群组通话下的提示信息。
优化昵称过长时的显示问题。

修复

修复语音通话请求摄像头权限问题。
修复 destroyed 存在的问题。
修复不同呼叫场景下悬浮窗里的挂断问题。
修复主叫呼叫状态下显示远端问题。
修复 pc 上填充不全的样式问题。

Version 3.1.5 @2023.12.15

新增

优化获取设备权限的时机，初始化时不再获取设备权限，使用 call 时才获取。

修复

修复 @tencentcloud/call-uikit-vue2、@tencentcloud/call-uikit-vue2.6 组件没有声明文件问题。

Version 3.1.4 @2023.12.01

新增

融合到 Chat 中增加 isFromChat 上报。

修复

修复按钮 loading 下可点击的问题。

Version 3.1.3 @2023.11.17

新增

接口增加参数校验。

Version 3.1.2 @2023.11.03

新增

新增邀请他人 inviteUser 功能。
新增中途加人 joinInGroupCall 功能。
新增被叫来电铃声静音接口 enableMuteMode功能。

修复

修复远端流麦克风状态展示不对问题。

Version 3.1.0 @2023.10.20

新增

增加悬浮窗功能。
增加 enableFloatWindow 接口，用于开启/关闭悬浮窗功能。
桌面端支持摄像头、麦克风切换设备。
增加呼叫黑名单用户失败提示信息。
增加对日语的支持。

优化

视频通话下，大屏默认显示远端用户。

Version 3.0.8 @2023.10.10

新增

增加版本号、框架等信息上报。

Version 3.0.7 @2023.10.08

版权所有：腾讯云计算（北京）有限责任公司

新增

增加桌面端视频通话下，通话时长展示。

优化

优化桌面端上视频流预览圆角黑边问题。
优化远端流用户信息显示优先级：备注 > 昵称 > userId。
优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 3.0.6 @2023.09.19

修复

修复融合到 TUIKit 中的消息上屏问题。

Version 3.0.5 @2023.09.15

优化

优化 TUICallKit 之间的相互引用，避免在 uniapp 打包小程序出现的栈溢出问题。

新增

新增桌面端设备无权限时的提示，引导客户如何对设备进行授权。

修复

修复 setCallingBell 被叫铃声被主叫铃声覆盖，导致铃声重复问题。
修复移动端上的样式问题。

Version 3.0.4 @2023.09.01

修复

修复 setCallingBell 作用对象为来电铃声（被叫铃声）。
修复 destroyed 报错问题。
修复错误弹框提示没有中英文。
修复 1v1 通话关闭摄像头后, 无法切换大小屏。

Version 3.0.3 @2023.8.25

新增

新增 @tencentcloud/call-uikit-vue2.6 适配 Vue2.6 版本。

优化

优化组件默认语言，调整为系统默认的语言。
优化打印的日志信息。
优化 tuicall-engine-webrtc 抛出错误的信息提示。
优化组件销毁后，对所用资源的清理。

修复

修复 videoDisplayMode、videoResolution 挂断后再次呼叫时，未生效问题。
修复通话中，statusChanged 未抛出问题。
修复 init 多次调用问题。
修复通话中，关闭摄像头无法切换大小屏问题。

Version 3.0.2 @2023.8.14

修复

修复 H5 端被叫组件的样式问题。
修复切换小窗后，再次通话出现的样式问题。

Version 3.0.1 @2023.8.8

修复

修复群通话主叫方本地预览失败的问题，修改组件层对数据层默认读取方式。

Version 3.0.0 @2023.8.4

Breaking Change

升级底层依赖 tuicall-engine-webrtc 至 ^2.0.0，不再支持 tim-js-sdk 创建 tim 实例，如需创建 tim 实例，请使用 @tencentcloud/chat 创建。

新增

新增自定义铃声功能 setCallingBell。

Version 2.4.2 @2023.11.03

新增

新增邀请他人 inviteUser 功能。
新增中途加人 joinInGroupCall 功能。
新增被叫来电铃声静音接口 enableMuteMode功能。

修复

修复远端流麦克风状态展示不对问题。

Version 2.4.0 @2023.10.20

新增

增加悬浮窗功能。
增加 enableFloatWindow 接口，用于开启/关闭悬浮窗功能。
桌面端支持摄像头、麦克风切换设备。
增加呼叫黑名单用户失败提示信息。
增加对日语的支持。

优化

视频通话下，大屏默认显示远端用户。

Version 2.3.9 @2023.10.10

新增

增加版本号、框架等信息上报。

Version 2.3.8 @2023.10.08

新增

增加桌面端视频通话下，通话时长展示。

优化

优化桌面端上视频流预览圆角黑边问题。

版权所有：腾讯云计算（北京）有限责任公司

优化远端流用户信息显示优先级：备注 > 昵称 > userId。
优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 2.3.6 @2023.09.15

优化

优化 TUICallKit 之间的相互引用，避免在 uniapp 打包小程序出现的栈溢出问题。

新增

新增桌面端设备无权限时的提示，引导客户如何对设备进行授权。

修复

修复 setCallingBell 被叫铃声被主叫铃声覆盖，导致铃声重复问题。
修复移动端上的样式问题。

Version 2.3.5 @2023.9.5

修复

修复进房前摄像头、麦克风按钮默认打开的问题。

Version 2.3.4 @2023.9.1

新增

新增视频通话接听前可以关闭或者开启摄像头功能。

修复

修复 1v1 通话关闭摄像头后, 无法切换大小屏问题。
修复从视频通话切换到语音通话 statusChanged 未抛出问题。

Version 2.3.3 @2023.8.22

修复

修复 videoDisplayMode、videoResolution 挂断后再次呼叫时，未生效问题。
修复通话中，statusChanged 未抛出问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 2.3.2 @2023.7.26

Breaking Change

移除悬浮窗 TUICallKitMini 组件，合并到 TUICallKit 组件中。
抛出的 @kicked-out 事件调整为绑定回调 `:kickedOut` 的方式。
抛出的 @status-changed 事件调整为绑定回调 `:statusChanged` 的方式。

新增

新增通话页面出现时的动画效果。
新增 H5 上群组通话布局。

优化

优化通话中的问题提示信息、提示方式。
优化在 H5 页面的支持、交互。
优化通话界面唤起的时间。
优化 @tencentcloud/call-uikit-vue 包目录结构。

修复

修复接通后立刻挂断等边界操作下的通话问题。
修复 H5 在部分机型、浏览器上的样式问题。
修复重复点击引起的通话异常问题。

Version 2.2.1 @2023.7.7

新增

@tencentcloud/call-uikit-vue2 新增 Vue 版本的检测及提示。

修复

修复来电页面重复点击“接听”按钮导致接听失败。

Version 2.2.0 @2023.6.30

新增

call、groupCall 支持自定义数字房间号 roomID 参数。
call、groupCall 支持自定义扩展字段 userData 参数（用于在邀请信令中增加扩展信息）。

新增 setSelfInfo 接口，支持用户设置昵称和头像。

Version 2.1.0 @2023.4.14

新增

在 H5 模式语音通话下，呼叫时支持显示对方昵称。
发起通话失败时，会在呼叫页面显示“发起通话失败”。
接听通话失败时，会在来电页面显示“接听失败”。
支持监听当前用户是否被踢出（如被挤出登录），详见 TUICallKit 方法 - @kicked-out。
支持监听 TUICallKit 通话状态，详见 TUICallKit 方法 - @status-changed。
支持业务侧代码控制通话的接听、取消和挂断，详见 更多特性 - 通过接口设置自动接听。
Vue2 版本新增 TypeScript 类型声明文件，在 TypeScript 项目中可以正常编译类型。

修复

修复在初始化组件时，控制台出现的更新个人资料接口的警告。
修复在 H5 模式下，被叫接听按钮出现背景图片错位的问题。

接口变更

`TUICallKitServer.destroy()` 增加调用限制，仅支持在未通话状态时调用。

Version 2.0.1 @2023.03.31

新增

优化 1v1 通话与多人通话视频的渲染逻辑，提升性能与稳定性。
优化 UI 展示，支持在 `TUICallKitServer.call()` 执行过程中显示对应 UI，即可以实现点击通话按钮立刻展示 `<TUICallKit/>` UI 组件。

修复

修复多人通话中昵称显示错误的问题。
修复 CSS 未限制生效范围导致污染全局样式的问题。

Version 2.0.0 @2023.03.21

新增

支持从 npm 引入已打包的 CallKit 文件。

支持 JavaScript 版本的 Vue 项目。
支持 Vue2 所有版本的 Vue 项目，适用于 Vue2 的 npm 包：call-uikit-vue2。

修复

修复无摄像头设备或权限导致通话发起异常的问题。

Version 1.4.2 @2023.03.03

新增

支持设置通话分辨率，详见 API 文档。
支持修改画面显示模式，详见 API 文档。
优化了接入步骤。
优化了错误抛出。

Version 1.4.1 @2023.02.13

新增

优化了本地摄像头的预览逻辑。
优化了远端视频流的渲染逻辑。

Version 1.4.0 @2023.01.06

新增

支持 Vue2.7+ 项目引入。
通话界面默认显示昵称，设置昵称请参考 TIM#updateMyProfile。

Version 1.3.3 @2022.12.27

新增

在 Basic Demo 中拨打电话时，新增了通话列表的空值检测。
在拨打电话时，Basic demo 中增加了 loading 图标。
优化了 Basic Demo 中设备检测的逻辑，手动跳过后不再主动弹出。
优化了组件图标的引用方式。
修改了默认的包管理工具为 npm。

版权所有：腾讯云计算（北京）有限责任公司

优化了视频的渲染方式，减少了重复渲染的次数。

修复

修复了由于 vue-CLI 过时依赖导致的 Basic Demo 报错。

Version 1.3.2 @2022.12.07

新增

支持语言切换，接口详情可见 setLanguage。
优化 Basic Demo 设备检测逻辑，手动跳过后不会再主动弹出。

修复

修复引入 `defineProps` 导致的警告。

Version 1.3.1 @2022.11.29

注意：
本版本依赖 SDK tuicall-engine-webrtc@1.2.1 版本，请及时更新。

新增

优化样式细节。
支持未接听对方通话时，监听对方修改通话类型。
basic demo 增加设备检测功能。

修复

修复在挂断电话时内部逻辑导致的报错。

Version 1.3.0 @2022.11.14

新增

支持手机 H5 使用时，自动切换为竖屏样式。
支持在拨打电话时预览本地摄像头。
basic demo 增加拨打电话前的设备检测。

修复

修复调用 TUICallKitServer.destroyed() 之后 tim 实例未完全退出的问题。

修复对方忙线时显示无应答的问题。
修复 vite 环境下 typescript 类型未成功打包的问题。

接口变更

主动调用 TUICallKitServer.call() 或 TUICallKitServer.groupCall() 时，如果遇到报错，不会再调用 beforeCalling 回调。请直接使用 try catch 捕获错误。

Version 1.2.0 @2022.11.03

新增

适配新版本 TUICallEngine SDK。

Version 1.1.0 @2022.10.21

新增

通话过程中，通话页面可以全屏。
通话过程中，可以使用 `<TUICallKitMini/>` 进行最小化。

修复

修复已知问题，提升稳定性。

Version 1.0.3 @2022.10.14

新增

basic demo 增加快捷复制 UserID、一键打开新窗口。

Version 1.0.2 @2022.09.30

新增

优化接入文档，增加演示图片和详细指引。

修复

修复首次进入房间时，设备状态位失效的问题。
修复在 webpack 工具打包时，图标偶现加载失败的问题。
修复已知样式问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 1.0.1 @2022.09.26

新增

在拨打电话时，隐藏对方麦克风图标。

修复

修复 basic demo SDKAppID 输入框应为数字的问题。

Version 1.0.0 @2022.09.23

TUICallKit demo 快速跑通

TUICallKit 快速接入

TUICallKit API

TUICallKit 界面定制指引

TUICallKit (Web) 常见问题

版权所有：腾讯云计算（北京）有限责任公司

Android & iOS

最近更新时间：2023-11-29 18:20:03

Version 2.0.0.750 @ 2023.11.03

功能优化

Android&iOS：UIKit 支持日语。
Android&iOS：优化通话昵称显示。
Android&iOS：调整铃声的默认音量，由60%调整到100%。

缺陷修复

iOS：修复 Swift 版本加载图片过慢问题。
iOS：修复主叫端发起通话后退后台，通话邀请自动取消问题。

Version 1.9.0.680 @ 2023.09.27

功能优化

Android&iOS：支持阿拉伯语言。
Android&iOS：优化套餐购买提示，支持根据链接跳转到对应的套餐购买页。
Android&iOS：优化不同分辨率下的默认码率，高分辨率下画面更清晰，详见 Resolution。
Android&iOS：调整视频通话的默认码率为600kbps，美颜等级为4级。

缺陷修复

Android&iOS：对黑名单用户发起通话，通话拒绝提示与发送单聊消息拒绝提示不一致。
iOS：4人群视频通话界面，一名成员拒绝接听，发起人视频位置画面异常。
iOS：修复登录成功后立即设置美颜，分辨率会被重置问题。

Version 1.8.0.620 @ 2023.08.14

功能优化

Android&iOS：通话消息默认不计入未读数。
Android：优化小米手机悬浮窗权限跳转页面。

缺陷修复

版权所有：腾讯云计算（北京）有限责任公司

iOS：修复被移下线后回调接口 onKickOffline 无效问题。
iOS： 修复通话记录未接界面清除通话后，回到所有通话界面，列表为空问题。

Version 1.7.2.570 @ 2023.07.20

功能优化

Android：默认关闭重力感应，优化在大屏、定制设备的通话体验。

缺陷修复

Android&iOS：A 呼叫 B（离线）后取消，A 再次呼叫 B，B 登录上线，B 的云端通话记录异常问题。
Android： 升级 TRTC SDK 版本到11.3后，TUICallKit 的崩溃问题。

Version 1.7.0.460 @ 2023.06.25

功能优化

Android&iOS：含 UI 集成方案，优化示例工程，完善通话设置项。
Android：降低通话时系统保活的级别，只在状态栏显示保活提示，去掉通知和震动。

Version 1.6.1.410 @ 2023.05.22

新特性

Android&iOS：含 UI 接口 call、groupCall 支持自定义房间号。
Android&iOS：支持发起通话时传入字符串型房间号，详见 CallParams。

缺陷修复

Android：修复通过 Arrays.asList 方式生成列表参数时，groupCall 报错问题。
Android：修复视频通话显示异常问题。
iOS：修复与 TUIRoom 组件冲突问题。
iOS：修复登录成功回调中，立刻发起通话崩溃问题。
iOS：修复点击通知消息进入 APP，偶现不显示邀请页面问题。

Version 1.6.0.360 @ 2023.04.27

新特性

版权所有：腾讯云计算（北京）有限责任公司

Android：TUICallKit 新增 Kotlin 语言版本；
iOS：TUICallKit 新增 Swift 语言版本；
Android&iOS：新增通话记录（本地）的显示页面。

功能优化

Android：优化单人视频通话头像显示；
Android&iOS：群组通话中，默认支持邀请其他群成员加入通话。

缺陷修复

Android：修复 Android 12及其以上 设备链接蓝牙后无声的问题；
Android：修复被叫端偶现静音设置不生效问题；
iOS：修复重新登录后偶现设备接收不到来电邀请的问题；
IOS：修复 TUICallKit 中 enableCustomViewRoute 接口失效问题；
IOS：修复 VoIP 推送页面昵称显示错误的问题。

Version 1.5.1.310 @ 2023.04.17

新特性

Android&iOS：新增 VoIP 消息推送功能，提供更好的通话接听体验。
Android&iOS：支持发起通话时自定义扩展字段，详见 call 方法的 TUICallDefine.CallParams 参数。

功能优化

Android&iOS：优化华为、小米、FCM 等厂商的离线推送能力，增加厂商消息分类及 channelID 设置。

缺陷修复

Android&iOS：修复 IM 自定义属性发起通话后被覆盖的问题。
Android：修复回调onCallEnd中 totalTime 的时间单位错误的问题；

Version 1.5.0.305 @ 2023.03.09

功能优化

Android&iOS：优化上屏消息显示。
Android：贴耳息屏功能默认关闭。
Android：升级 gradle 插件及版本。
Android：优化铃声播放类，支持循环播放。

版权所有：腾讯云计算（北京）有限责任公司

缺陷修复

Android&iOS：修复被叫接听通话失败，没有 onCallCancel 回调问题。
Android：修复被叫接听通话失败，主叫异常问题。
Android：修复首次通话检查权限时，主叫端取消通话，被叫端再次拉起界面问题。
Android：修复回调上层的网络质量 userId 为空问题。

Version 1.4.0.255 @ 2023.01.06

新特性

Android&iOS：支持自定义来电超时时间，详见 call方法 TUICallDefine.CallParams 参数。

缺陷修复

Android&iOS：修复主动加入房间（joinInGroupCall）时通话异常结束问题。
Android：修复在音视频通话接听中退后台，再回到前台时，通话状态异常问题。

Version 1.3.0.205 @ 2022.11.30

新特性：

Android&iOS：新增美颜设置接口 setBeautyLevel，支持关闭默认美颜。

功能优化：

iOS: 优化 TUICallKit 的 Framework 体积。

缺陷修复：

Android&iOS：修复服务端解散房间或踢出用户时，通话界面不消失问题。
Android：修复A呼叫离线用户B，然后取消；A 再次呼叫 B，B 上线后通话界面不显示问题。

Version 1.2.0.153 @ 2022.11.14

新特性：

Android&iOS：支持视频自定义编码分辨率。
Android&iOS：支持设置视频的渲染参数：渲染方向、填充模式。
Android&iOS：支持第三方美颜的接入。
Android&iOS：TUICallKit 新增重载接口 call()、groupCall()，支持自定义离线消息（详情请参见 API文档）。

功能优化：

Android&iOS：优化部分 TUICallObserver 回调异常问题。
Android&iOS：优化视频切语音功能，支持离线状态下切换。
Android&iOS：完善 TUICallKit 的错误码及错误提示。
iOS：规范 TUICallEngine、TUICallKit Swift API 名称。

缺陷修复：

Android&iOS：修复重新登录账号后，仍会收到之前已拒绝的来电问题。
Android：修复单聊中收到群聊邀请，通话异常挂断问题。
Android：多场景退出直播间无法发起通话异常问题。
Android：修复 A 呼叫 B，同时 B 呼叫 C，C 概率性进入 A 的房间问题。

Version 1.1.0.103 @ 2022.09.30

Android&iOS：优化群组通话过程中，邀请群成员加入功能。
Android&iOS：优化通话流程，避免通话接听前产生录制、审核等费用。
Android&iOS：支持自定义音视频通话的离线推送消息。
Android&iOS：部分 **TUICallEngine** 接口参数变更，详见 call()、groupCall()、inviteUser()、以及回调接口 onCallReceived()。
Android&iOS：修复群组通话过程中，偶现的少量回调异常问题。
Android&iOS：修复重复登录以及UserSig过期造成的状态异常问题。
iOS：修复 TUCallKit Objective-C 和 Swift 混合编译时，`init` 接口报错的问题。
Android：修复 Kotlin 集成悬浮窗功能时编译报错问题。

Version 1.0.0.53 @ 2022.08.15

全新发布：

Android&iOS：支持1v1 音视频通话、群组音视频通话。
Android&iOS：支持主流厂商的离线唤醒。
Android&iOS：支持自定义头像、自定义昵称。
Android&iOS：支持通话过程中开启悬浮窗。
Android&iOS：支持设置自定义铃音。
Android&iOS：支持多平台登录状态下的来电服务。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-01-10 17:10:40

Version 2.0.3 @2023.12.04

问题修复

Android&iOS：修复了点击来电通知后有时不显示来电页面的偶现问题。

Version 2.0.2 @2023.11.27

问题修复

Android：修复工程中有多个 Flutter Engine 实例时导致 Method Channel 通信异常的问题。

Version 2.0.1 @2023.11.15

问题修复

Android&iOS：修复了在使用其它 TRTC 组件时导致 TRTC Observer 不兼容的问题。
Android：修复了与 TUIRoom 结合使用时的不兼容问题。

Version 2.0.0 @2023.11.06

依赖描述

升级依赖客户端 SDK 版本：
Android&iOS TUICore：7.6.5011。
Android&iOS TUICallEngine：2.0.0.750。

Version 1.9.1 @2023.10.21

新特性

iOS：支持 Voip。

问题修复

iOS：修复后台接听电话时通话页面异常的问题。

Version 1.9.0 @2023.10.09

新特性

Android&iOS：添加设置铃声的接口。

功能优化

Android&iOS：优化购买包的提示。
Android&iOS：优化不同分辨率的默认比特率，参见。

问题修复

iOS：修复了同一个 Observer 对象可以被注册两次的问题。

依赖描述

升级依赖客户端 SDK 版本：
Android&iOS TUICore：7.5.4852。
Android&iOS TUICallEngine：1.9.0.680。

Version 1.8.3 @2023.08.25

问题修复

Android&iOS：修复了使用 tencent_cloud_chat_uikit 时没有通话消息显示的问题。
Android&iOS：修复了在单人通话过程中偶尔拉起群通话页面的问题。
Android&iOS：修复了在通话过程中偶尔拉起通话页面两次的问题。
Android&iOS：修复了通话时间显示异常的问题。

功能优化

Android：优化了在后台接收到通话时无法拉起界面的问题。

依赖描述

升级 tencent_cloud_uikit_core 到版本 1.1.1。

Version 1.8.2 @2023.08.19

问题修复

iOS：修复了某些编译器由于使用了已弃用的 Swift 接口而无法编译的问题。

Version 1.8.1 @2023.08.18

缺陷修复

Android&iOS：修复了在进行群通话语音通话时，另一方的视频流显示的问题。

Version 1.8.0 @ 2023.08.17

新特性

Android&iOS：基于 Dart 语言构建全新的 TUICallkit，可以更方便的定制自己的 UI 风格。
Android&iOS：TUICallEngine 新增诸如 hangup、accept、reject 等多个业务接口。

Version 1.7.4 @ 2023.07.20

功能优化

Android：默认关闭重力感应，优化在大屏、定制设备的通话体验。

缺陷修复

Android&iOS：A呼叫B(离线)后取消，A再次呼叫B，B登录上线，B的云端通话记录异常问题。

Version 1.7.3 @ 2023.07.19

功能优化

Android: 支持使用模拟器进行开发&调试。

依赖说明

升级依赖的客户端 SDK 版本：Android LiteAVSDK_Professional: 11.3.0.13176。

Version 1.7.2 @ 2023.07.09

缺陷修复

iOS: 升级客户端 SDK版本，修复因为 Non-public API usage 问题导致的 AppStore 上架失败的问题。

Version 1.7.1 @ 2023.07.03

新特性

Android&iOS：新增云端通话记录，您可以在控制台开通服务进行体验查询。

功能优化

Android：降低通话时系统保活的级别，只在状态栏显示保活提示，去掉通知和震动。

Version 1.6.3 @ 2023.06.07

缺陷修复

iOS：修复调用 joinInGroupCall 后中途加人页面为空的问题。
iOS：修复调用 joinInGroupCall 后用户画面遮挡的问题。

Version 1.6.2 @ 2023.05.24

缺陷修复

Android：修复调用 joinInGroupCall api引起Crash问题 。

Version 1.6.1 @ 2023.05.14

缺陷修复

Android：修复在 Vivo 特定机型上申请悬浮窗权限时偶现的崩溃问题。
依赖说明
升级依赖的客户端 SDK 版本：Android LiteAVSDK_Professional: 11.1.0.13111、iOS TXLiteAVSDK_Professional: 11.1.14143。

Version 1.6.0 @ 2023.05.07

新特性

Android&iOS：增加hangup挂断接口。
Android&iOS：增加用户自定义字段、增加用户自定义通话超时时长。
Android&iOS：群组通话增加中途加人页面。

功能优化

Android：优化单人视频通话头像显示。
Android&iOS：群组通话中，默认支持邀请其他群成员加入通话。

缺陷修复

Android：修复 Android 12及其以上 设备连接蓝牙后无声的问题。
Android：修复被叫端偶现静音设置不生效问题。
iOS：修复重新登录后偶现设备接收不到来电邀请的问题。
iOS：修复 VoIP 推送页面昵称显示错误的问题。

依赖说明

升级依赖的客户端 SDK 版本：Android LiteAVSDK_Professional: 11.1.0.13111、iOS TXLiteAVSDK_Professional: 11.1.14143。

Version 1.5.4 @ 2023.04.14

新特性

Android&iOS：增加小米、华为、VIVO 离线推送参数。
iOS：支持 Voip 消息推送功能。
Android&iOS：增加分辨率设置功能。

功能优化

Android：优化 onCallEnd 回调中 totaltime 参数单位为毫秒。

缺陷修复

Android&iOS：修复 onCallReceived 回调异常问题。
iOS：修复屏幕旋转时通话页面显示不全的问题

Version 1.5.3 @ 2023.03.17

缺陷修复

Android：修复打包失败问题。

Android&iOS：修复回调方法未实现时抛出异常的问题。

Version 1.5.2 @ 2023.03.13

缺陷修复

Android：修复 `TUICallDefine.OfflinePushInfo` API 变更导致的编译报错。

Version 1.5.1 @ 2023.03.13

缺陷修复

Android&iOS：修复 `tencent_calls_engine` 版本依赖错误的问题。

Version 1.5.0 @ 2023.03.13

功能优化

Android：贴耳息屏功能默认关闭。
Android：升级 gradle 插件及版本。
Android：优化铃声播放类，支持循环播放。

缺陷修复

Android&iOS：修复被叫接听通话失败，没有 onCallCancel 回调问题。
Android：修复被叫接听通话失败，主叫异常问题。
Android：修复首次通话检查权限时，主叫端取消通话，被叫端再次拉起界面问题。
Android：修复回调上层的网络质量 userId 为空问题。
iOS：修复 Example 中 Observer 注册时机错误的问题。

Version 1.4.2 @ 2023.02.24

缺陷修复

Android&iOS：修复 Example 中错误的 Observer 注册时机导致的通话异常问题。
iOS：修复 `removeObserver` API 偶现设置无效的问题。

Version 1.4.1 @ 2023.02.20

版权所有：腾讯云计算（北京）有限责任公司

缺陷修复

Android：修复通话结束后，`OnCallEnd`事件丢失的问题。

Version 1.4.0 @ 2023.01.16

缺陷修复：

Android&iOS：修复主动加入房间（joinInGroupCall）时通话异常结束问题。
Android：修复在音视频通话接听中退后台，再回到前台时，通话状态异常问题。
Android：修复同时集成 `tencent_cloud_chat_uikit` 插件时，登录状态导致的通话发起失败问题。
Android：修复发起群组通话中因为参数检查问题导致的通话发起失败问题。

Version 1.3.1 @ 2022.12.27

新特性：

Android&iOS: 支持通话时自定义离线推送消息。

缺陷修复：

Android：修复同时集成 tencent_cloud_chat_uikit 插件时，提示sdkappid is invalid的问题。

Version 1.3.0 @ 2022.12.02

功能优化：

iOS：优化 TUICallKit 的 Framework 体积。

缺陷修复：

Android&iOS：修复服务端解散房间或踢出用户时，通话界面不消失问题。
Android：修复A呼叫离线用户B，然后取消；A 再次呼叫 B，B 上线后通话界面不显示问题。

Version 1.2.2 @ 2022.11.17

缺陷修复：

iOS：修复静态库链接导致的编译问题。

Version 1.2.0 @ 2022.11.17

新特性：

支持1v1 音视频通话、群组音视频通话。
支持自定义头像、自定义昵称。
支持设置自定义铃音。
支持通话过程中开启悬浮窗。
支持多平台登录状态下的来电服务。

常见问题（TUICallKit）

Web

最近更新时间：2024-04-03 17:23:11

错误提示“The package you purchased does not support this ability”？

如遇以上错误提示，是由于您当前应用的音视频通话能力包过期或未开通，请参见 开通服务，领取或者开通音视频通话能力，进而继续使用 TUICallKit 组件。

如何购买套餐？

请参考购买链接 购买正式版。

如何生成 UserSig？

UserSig 是腾讯云为其云服务设计的一种安全保护签名，是一种登录凭证，由 SDKAppID 与 SecretKey 等信息组合加密得到。

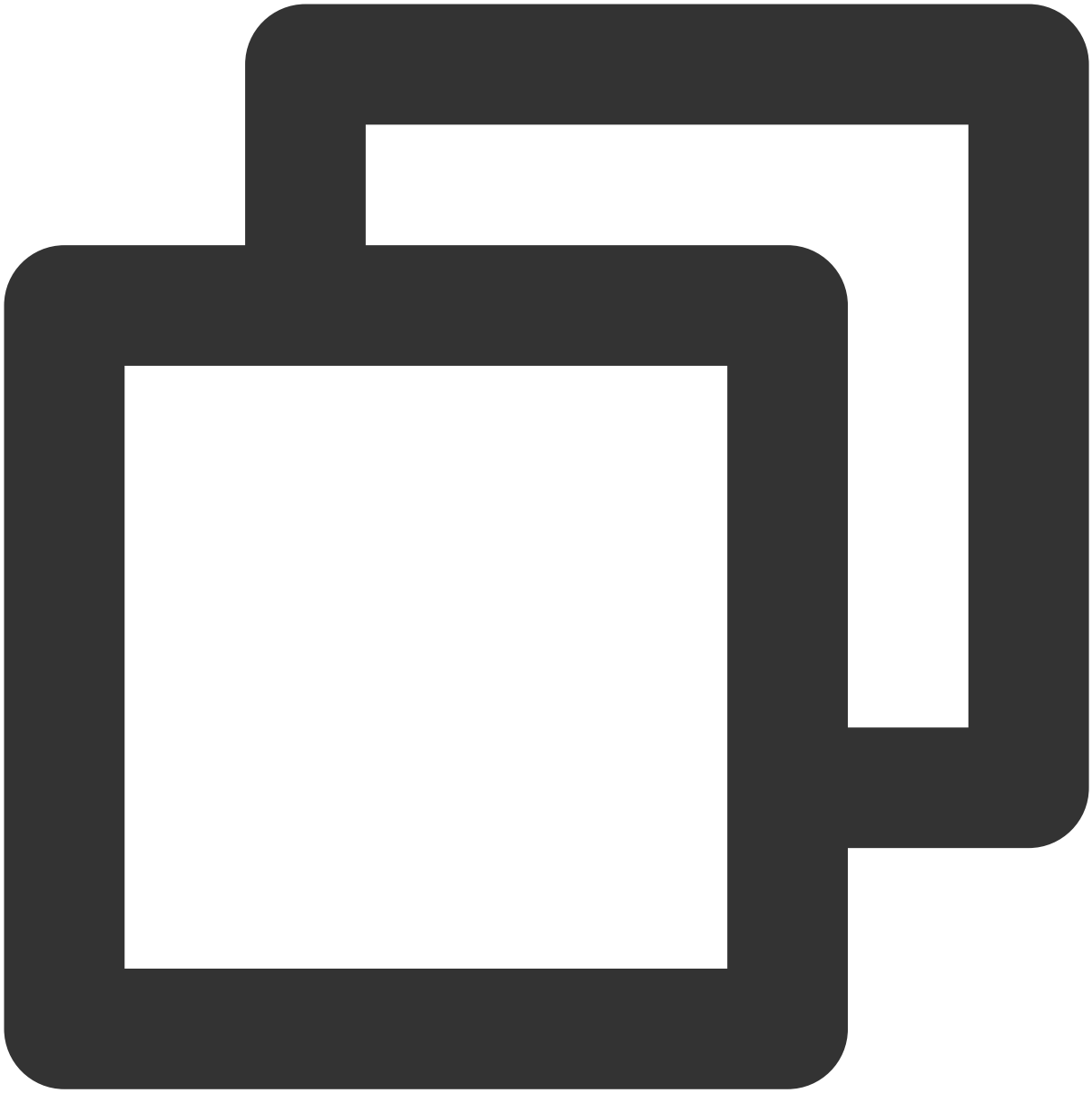
方式一：控制台获取，参考 获取临时 userSig。

方式二：部署临时生成脚本。

警告：

此方式是在前端代码中配置 SecretKey，该方法中 SecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**，生产环境请看方式三。

为方便初期调试，userSig 可临时使用 `GenerateTestUserSig-es.js` 中 `genTestUserSig(params)` 函数来计算，例如：



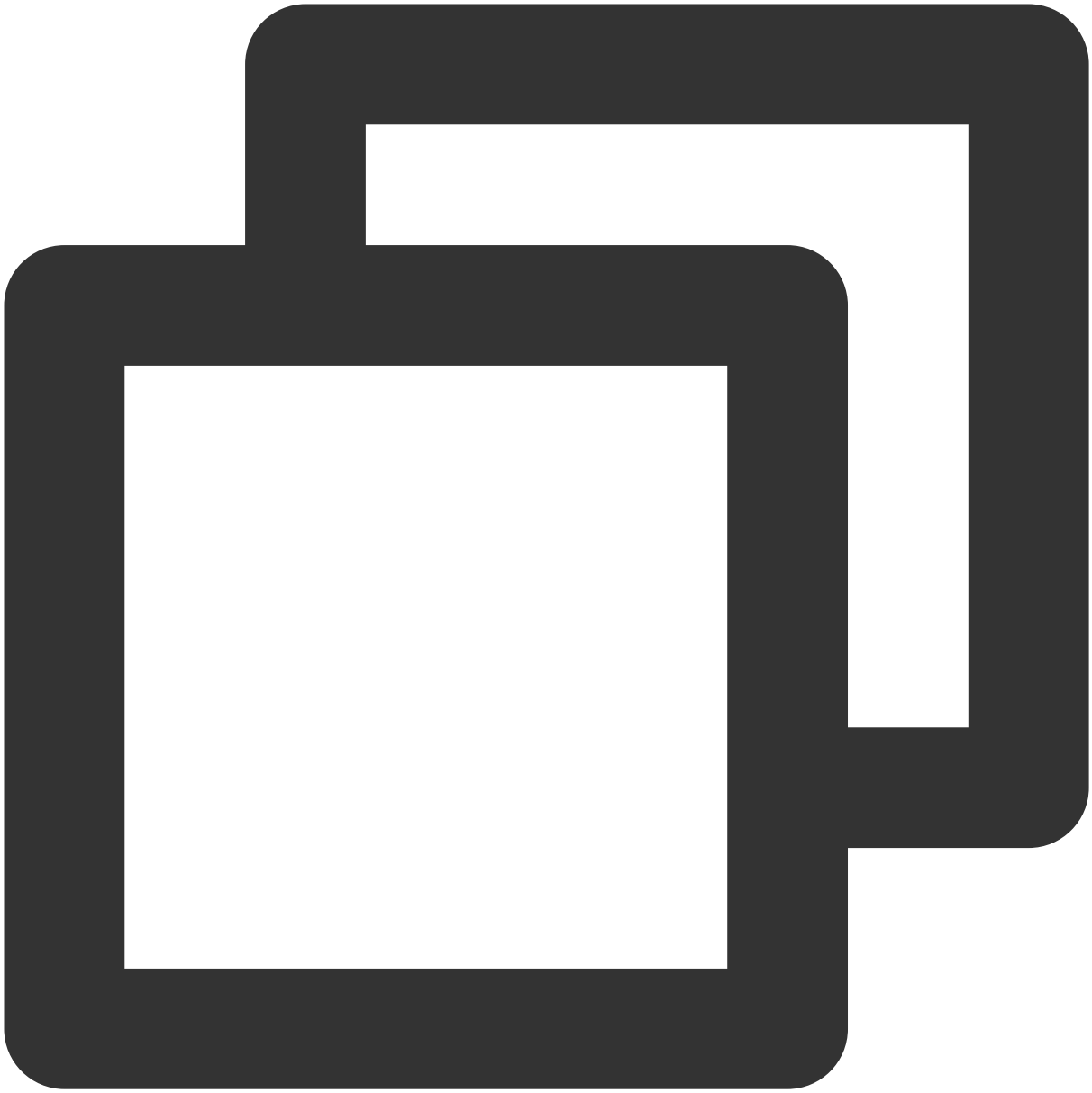
```
import { genTestUserSig } from "@tencentcloud/call-uikit-vue/debug/GenerateTestUser
const { userSig } = genTestUserSig({ userID: "Alice", SDKAppID: 0, SecretKey: "YOUT
```

方式三：正式环境使用。

正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

群组通话中的 groupID 如何生成？

groupID 的生成需要集成 @tencentcloud/chat 包，具体参考 createGroup API；如下为生成 groupID 的示例代码。



```
import TIM from "@tencentcloud/chat"; // npm i @tencentcloud/chat

const userIDList: string[] = ['user1', 'user2', 'xxx'];  // group member
async function createGroupID() {
  const tim = TIM.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userId => ({ userID: userId }));
  const res = await tim.createGroup({
    type: TIM.TYPES.GRP_PUBLIC,  // must be a public group
    name: 'WebSDK',
    memberList
  });
```

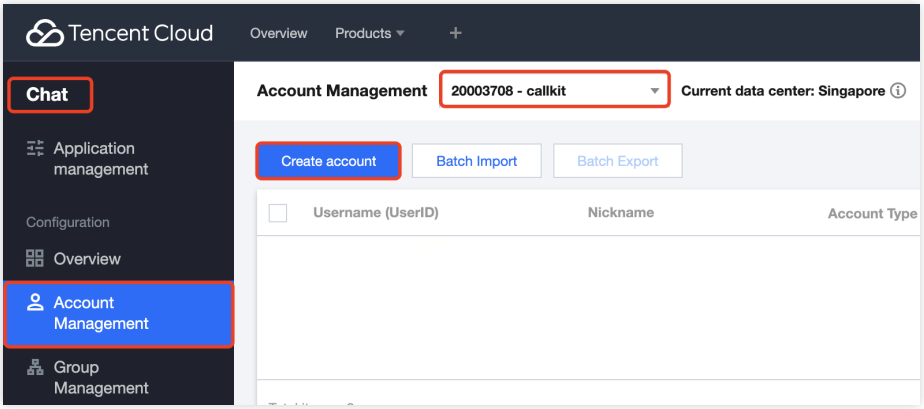
版权所有：腾讯云计算（北京）有限责任公司

```
    return res.data.group.groupID;
}
```

如何创建 userID？

通过 userID 与 userSig 登录过一次，会默认创建该用户。

通过 即时通信 IM 控制台 进行创建和获取。



全平台

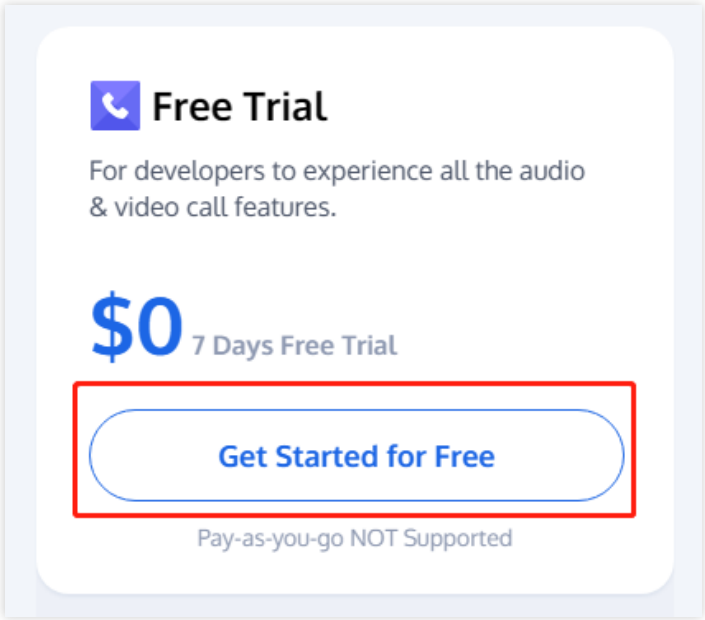
最近更新时间：2023-09-11 17:08:55

音视频通话功能出现了未开通的错误提示？

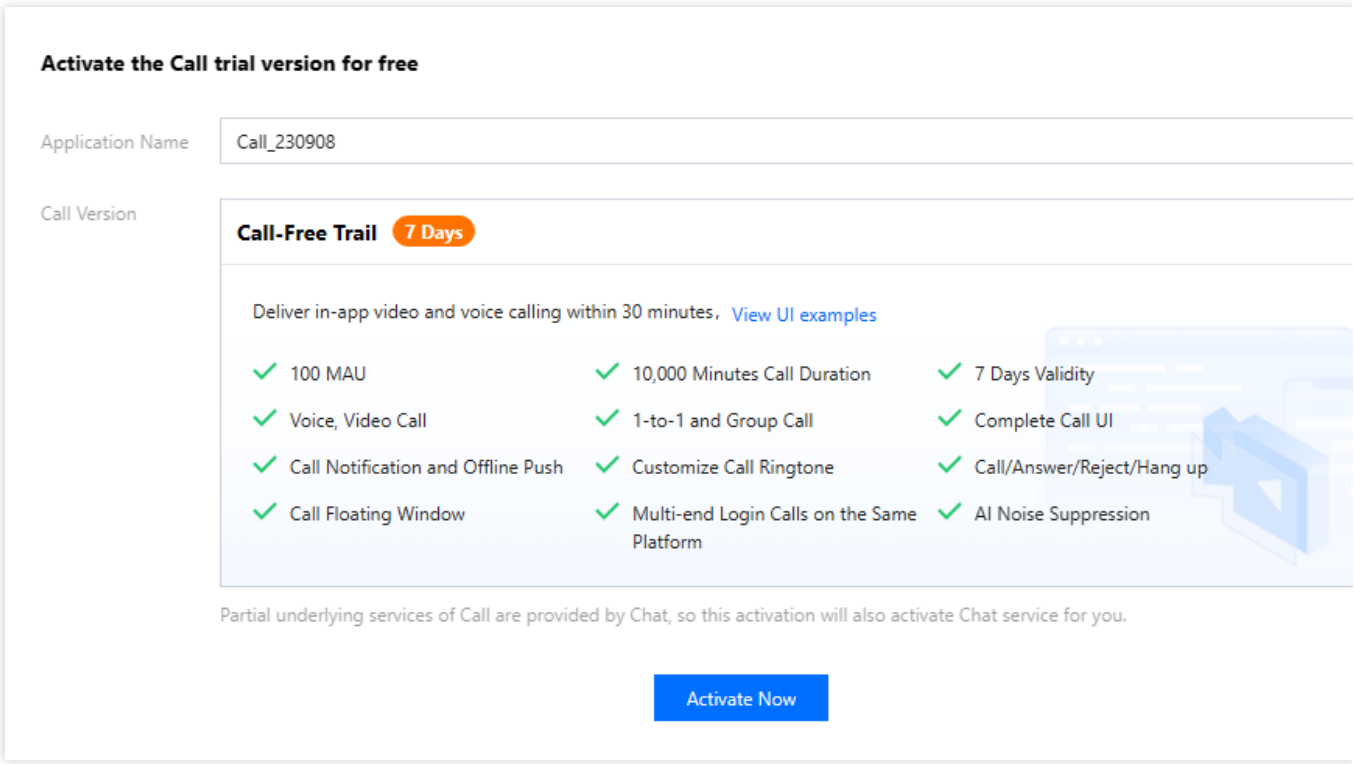
您可以选择体验版（7 天试用）或者购买正式版。

对于体验版

1. 进入价格页，阅读体验版套餐包含的具体服务，单击**免费体验**，根据文档提示进入控制台。



2. 在控制台弹窗中阅读**Call-Free Trail**套餐信息，确认无误后单击**现在激活**即可开通 TUICallKit 的 7 天免费试用服务，成为体验版。

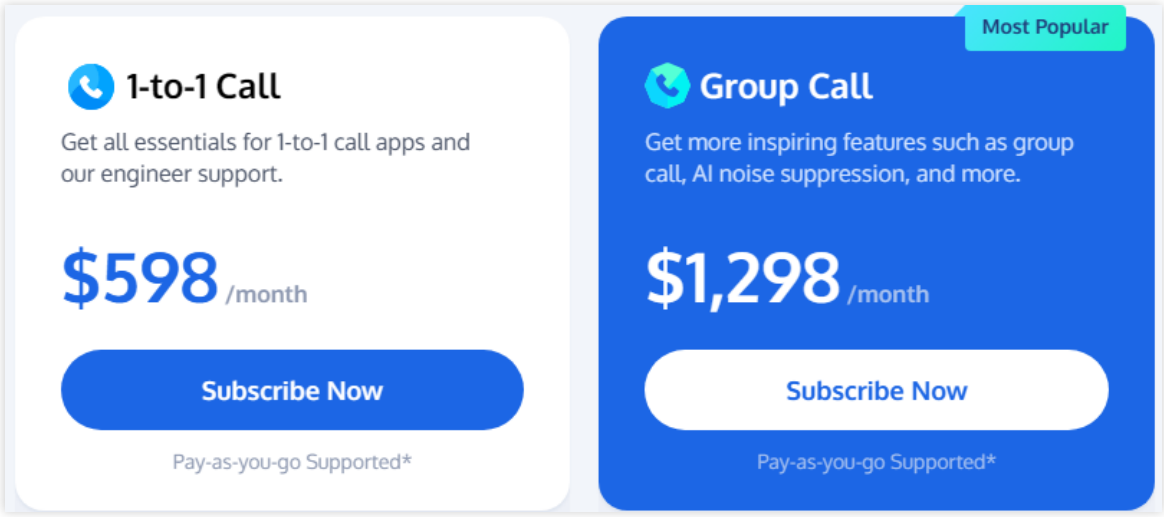


注意：

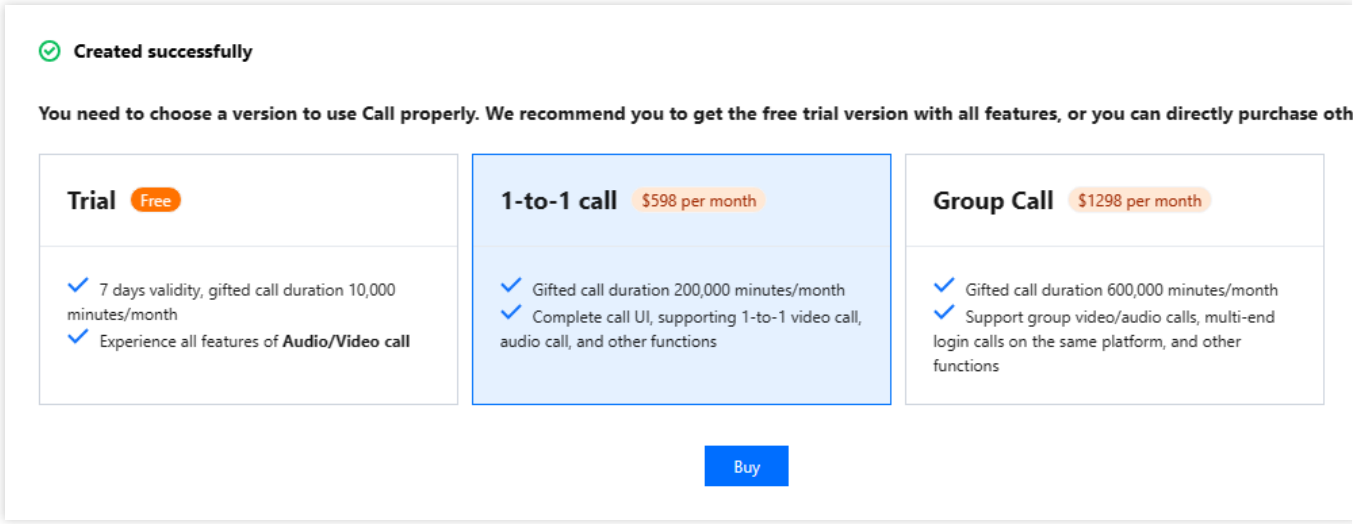
Call的部分底层服务是由Chat提供的，因此此激活也将为您激活Chat服务。

对于正式版

1. 进入价格页，阅读体验版套餐包含的具体服务，单击**现在订阅**，根据文档提示进入控制台。



2. 在控制台中单击**创建应用**，在创建成功后的弹窗中根据需要选择**1-to-1 call**或**Group Call**即可开通 TUICallKit 的正式版服务。



报错 inviteID 无效或邀请已处理

该 inviteID 已经被主叫方取消，或者该邀请已经被接受。由于信令发送受网络带宽等诸多因素影响，在极端情况下(accept 与 reject 接口重复调用)可能会导致 inviteID 无效的错误。

消息发送方或接收方 UserID 无效或不存在

您呼叫的用户还没有登录过 TIM，导致呼叫失败，请尝试呼叫已经登录成功过的用户。

TUICallKit 是否可以不引入 IM SDK，只使用 TRTC？

不可以，TUIKit 全系组件都使用了腾讯云 IM SDK 做为通信的基础服务，比如通话拨打信令、通话忙线信令等核心逻辑，如果您已经购买有其他 IM 产品，也可以参照 TUICallKit 逻辑进行适配。

如何购买音视频通话套餐？

请参考购买链接 音视频通话 SDK 价格总览，如有其他问题，请点击页面顶部Talk to us，进行售前套餐咨询。

如何获取通话的房间号 Roomid？

通话接通后，您可以通过 onCallBegin 返回 roomid 字段进行获取。

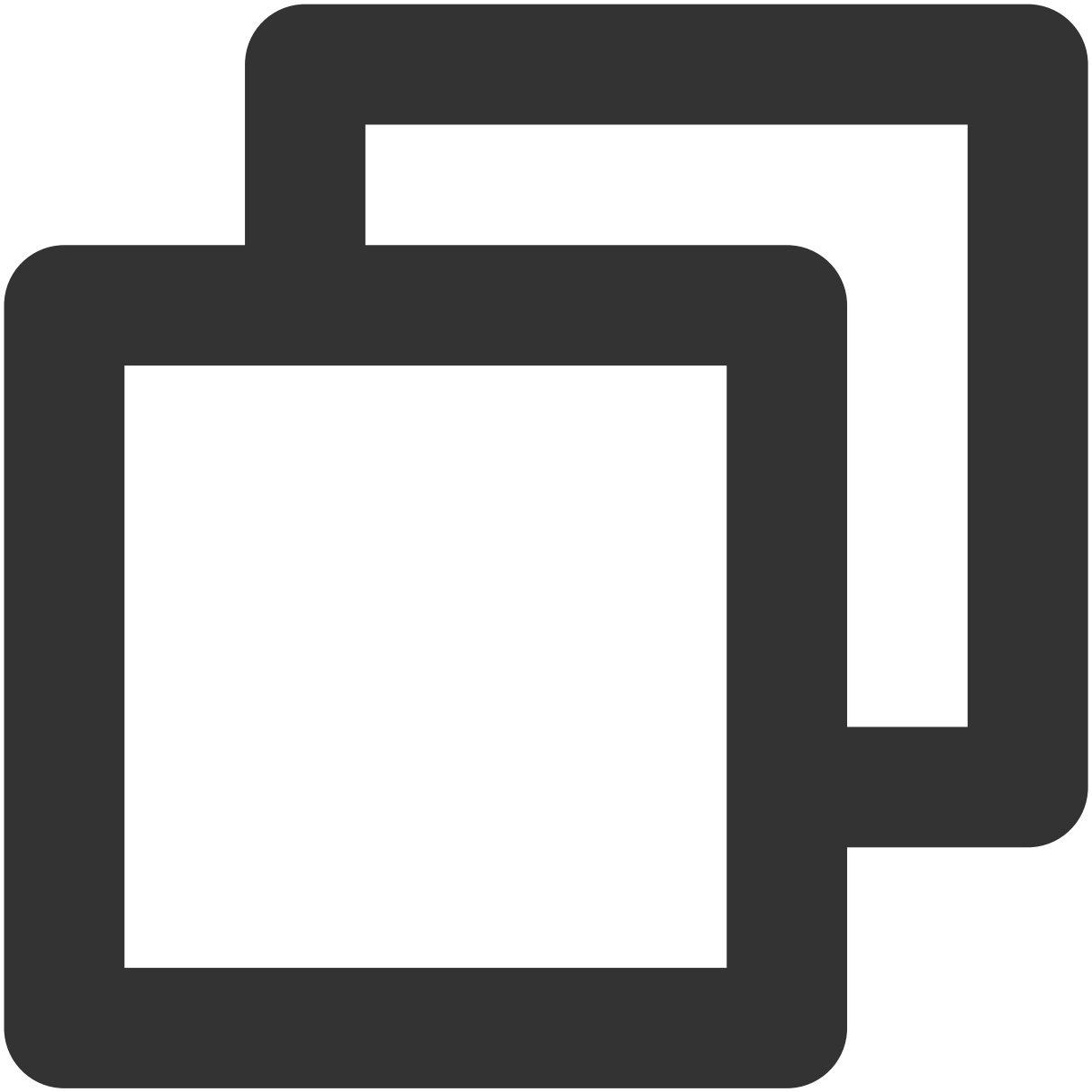
版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2023-09-11 17:46:52

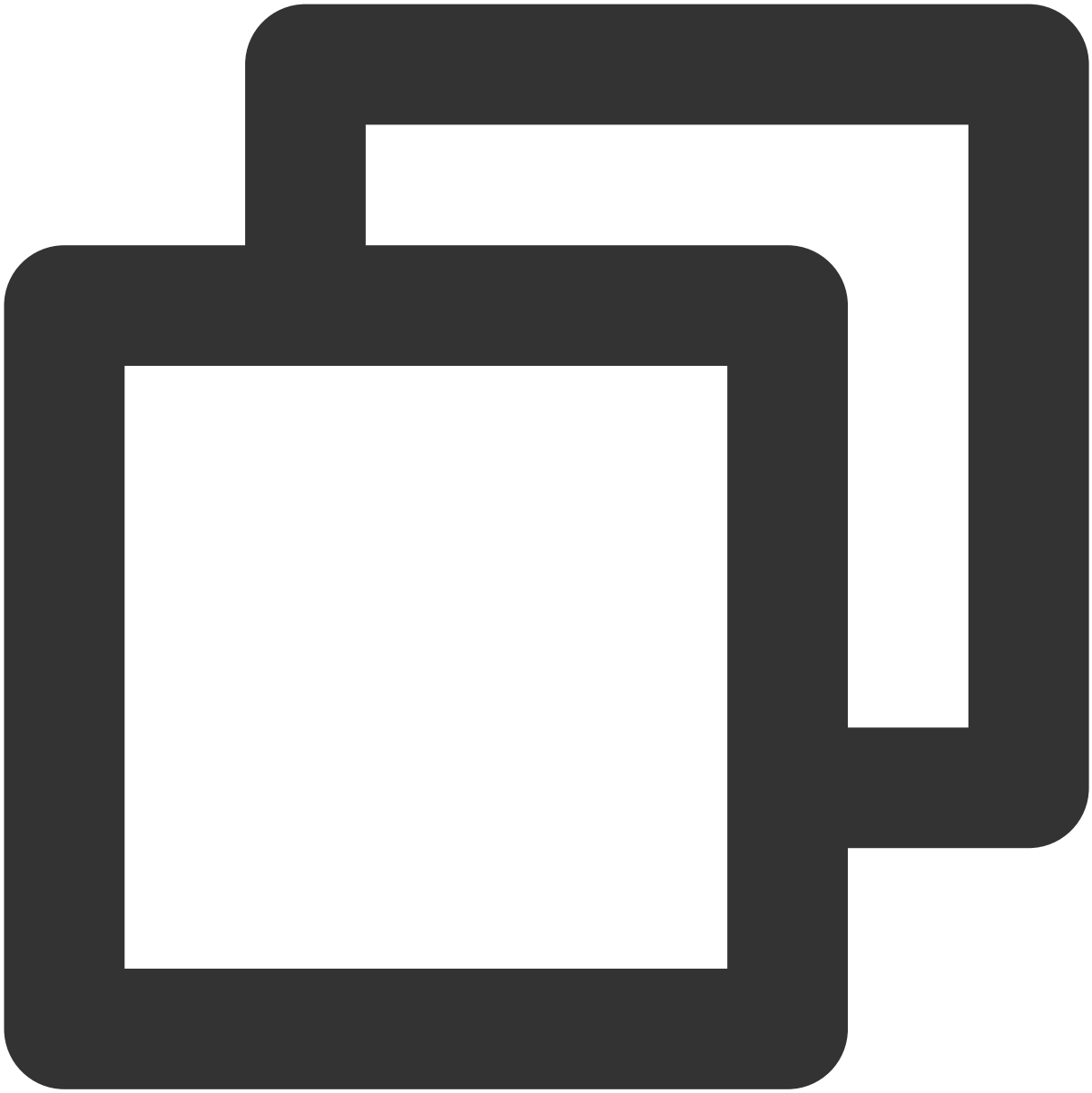
1、同时集成 tencent_calls_uikit 和 tencent_trtc_cloud，或同时集成 tencent_calls_uikit 和 live_flutter_plugin，出现符号冲突报错，怎么解决？

问题详情：当引入 flutter "tencent_calls_uikit" 进我们现有的 project 后，Android 构建 APK 时出现如下报错：



```
Duplicate class com.tencent.liteav.LiveSettingJni found in modules jetified-LiteAVS
(com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053) and jetified-LiteAVSDK_TRT
(com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225)
```

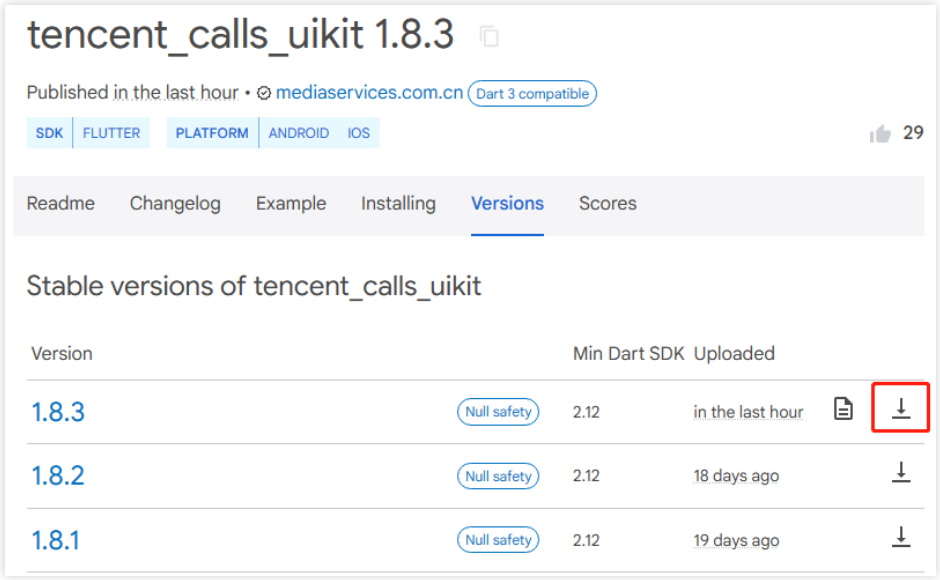
iOS 在执行 `pod install` 出现如下报错：

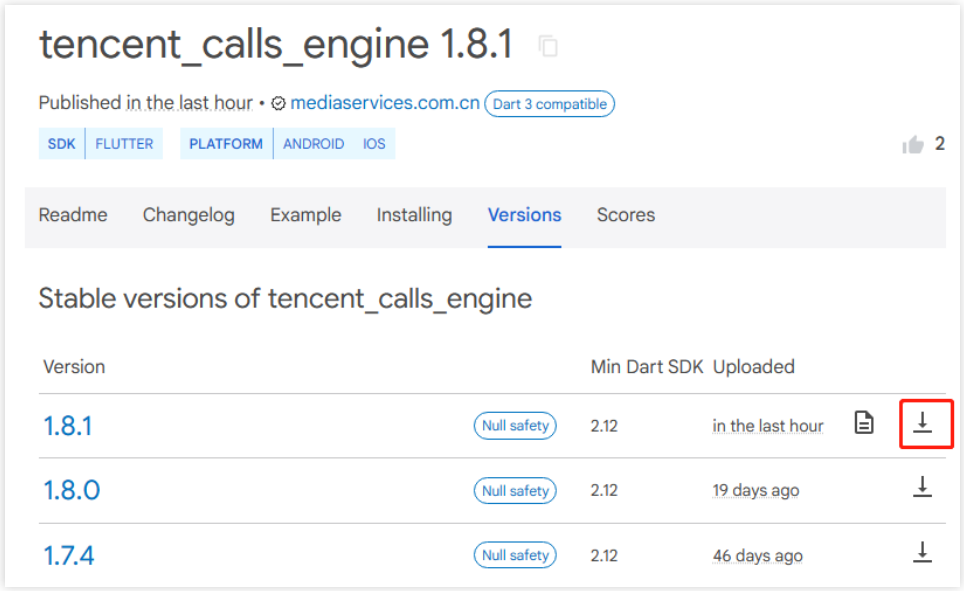


```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xc
```

这个问题原因是 tencent_calls_uikit 和 tencent_trtc_cloud 分别依赖于 TRTC Android SDK 的专业版和精简版（这个问题我们会在后续版本逐步统一），目前可以通过本地集成的方式来修复这个问题，具体步骤如下：

1. 分别进入 tencent_calls_uikit 和 tencent_calls_engine 的版本管理页面（tencent_calls_uikit依赖于tencent_calls_engine），单击下载按钮，等待下载完成后，解压文件。





2. 修改 tencent_calls_engine 插件中 TRTC SDK 依赖为专业版本。

Android：进入 `tencent_calls_engine` 解压后的目录，修改 `android/build.gradle` 文件中的依赖为报错信息中呈现的：`com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053` 。

```
}

dependencies{
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    api 'com.google.code.gson:gson:2.9.1'
    api "com.tencent.liteav.tuikit:tuicore:1.1.16"
    api 'com.tencent.imsdk:imsdk-plus:6.8.3374'
    api "com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225"
    api 'com.tencent.liteav.tuikit:tuicallengine:1.3.0.205'
}
```

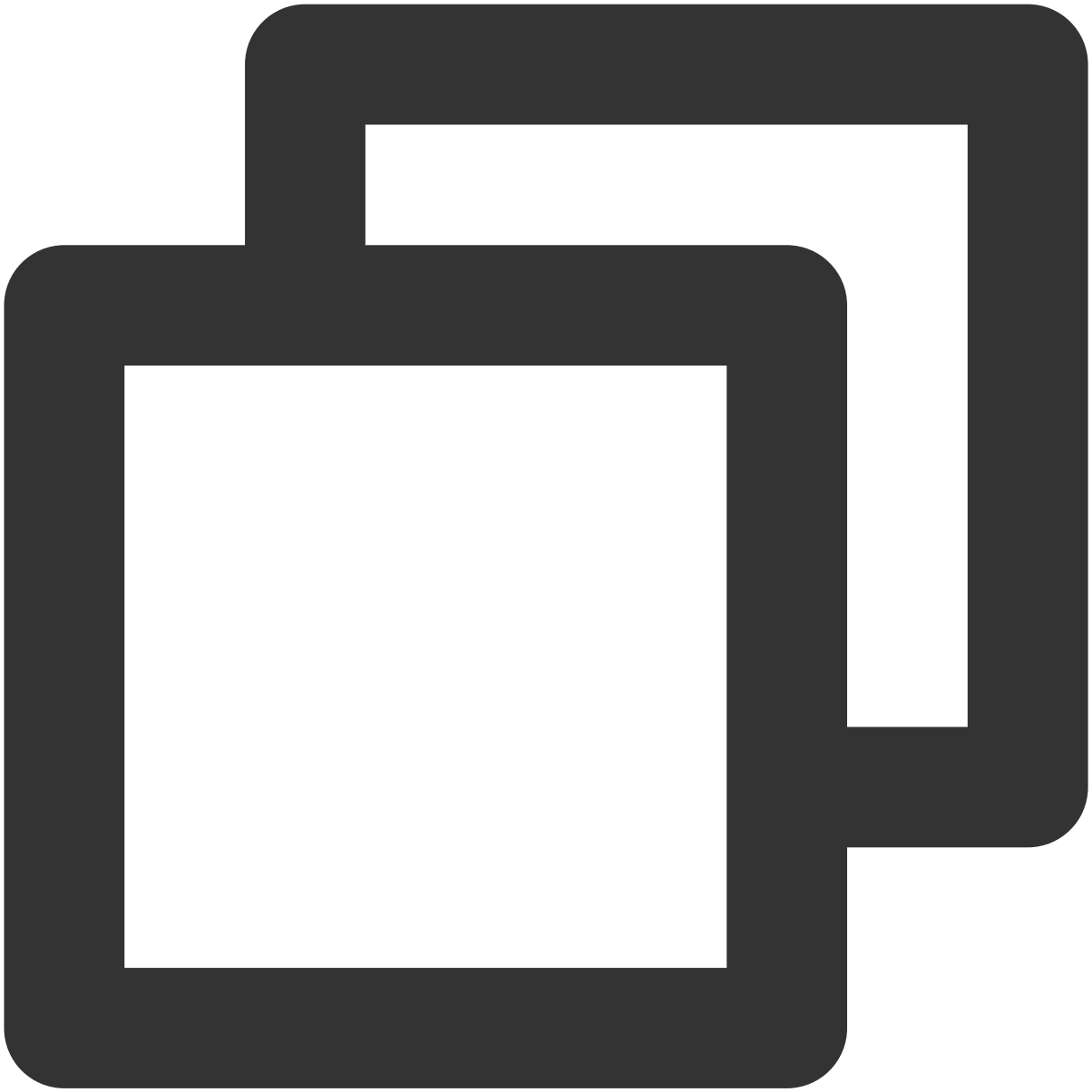
iOS：首先进入 `tencent_calls_engine` 解压后的目录，修改 `ios/tencent_calls_engine.podspec` 文件中的依赖为：`s.dependency 'TUICallEngine/Professional', '1.5.1.310'` 。

```
A new Flutter plugin project.
                    DESC
  s.homepage         = 'http://example.com'
  s.license          = { :file => '../LICENSE' }
  s.author           = { 'Your Company' => 'email@example.com' }
  s.source           = { :path => '.' }
  s.source_files = 'Classes/**/*'
  s.dependency 'Flutter'
  s.dependency 'TUICallEngine/Professional', '1.5.1.310'
  s.platform = :ios, '9.0'
  s.static_framework = true
  # Flutter.framework does not contain a i386 slice.
  s.pod_target_xcconfig = { 'DEFINES_MODULE' => 'YES', 'EXCLUDED_ARCHS[sdk=iphonesimulator*]' => 'i38
  s.swift_version = '5.0'
```

再进入 `tencent_calls_uikit` 解压后的目录，修改 `ios/tencent_calls_uikit.podspec` 文件中的依赖为：`s.dependency 'TUICallKit/Professional', '1.5.1.310'` 。

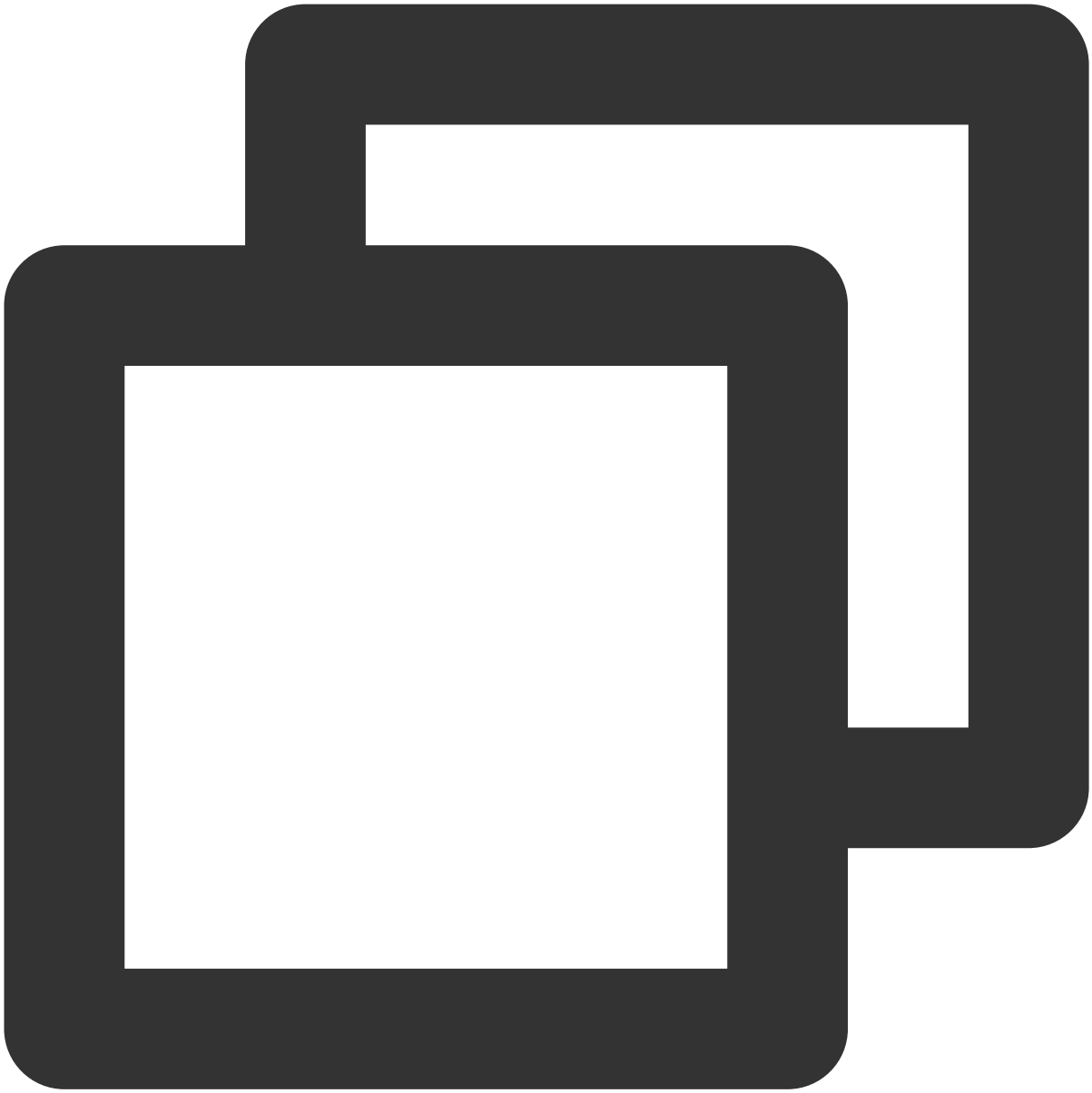
```
A new Flutter plugin project.
                      DESC
  s.homepage         = 'http://example.com'
  s.license          = { :file => '../LICENSE' }
  s.author           = { 'Your Company' => 'email@example.com' }
  s.source           = { :path => '.' }
  s.source_files = 'Classes/**/*'
  s.dependency 'Flutter'
  s.dependency 'TUICallKit/Professional' '1.5.1.310'
  s.platform = :ios, '9.0'
  s.static_framework = true
  # Flutter.framework does not contain a i386 slice.
  s.pod_target_xcconfig = { 'DEFINES_MODULE' => 'YES', 'EXCLUDED_ARCHS[sdk=iphonesimulator*]' => 'i386
  s.swift_version = '5.0'
end
```

3. 修改 tencent_calls_uikit 工程中的 `pubspec.yaml`，将 `tencent_calls_engine` 的依赖调整为本地依赖，示例如下：



```
    tencent_calls_engine:
    path: tencent_calls_engine/   # 这里的目录对应您刚才下载的插件目录
```

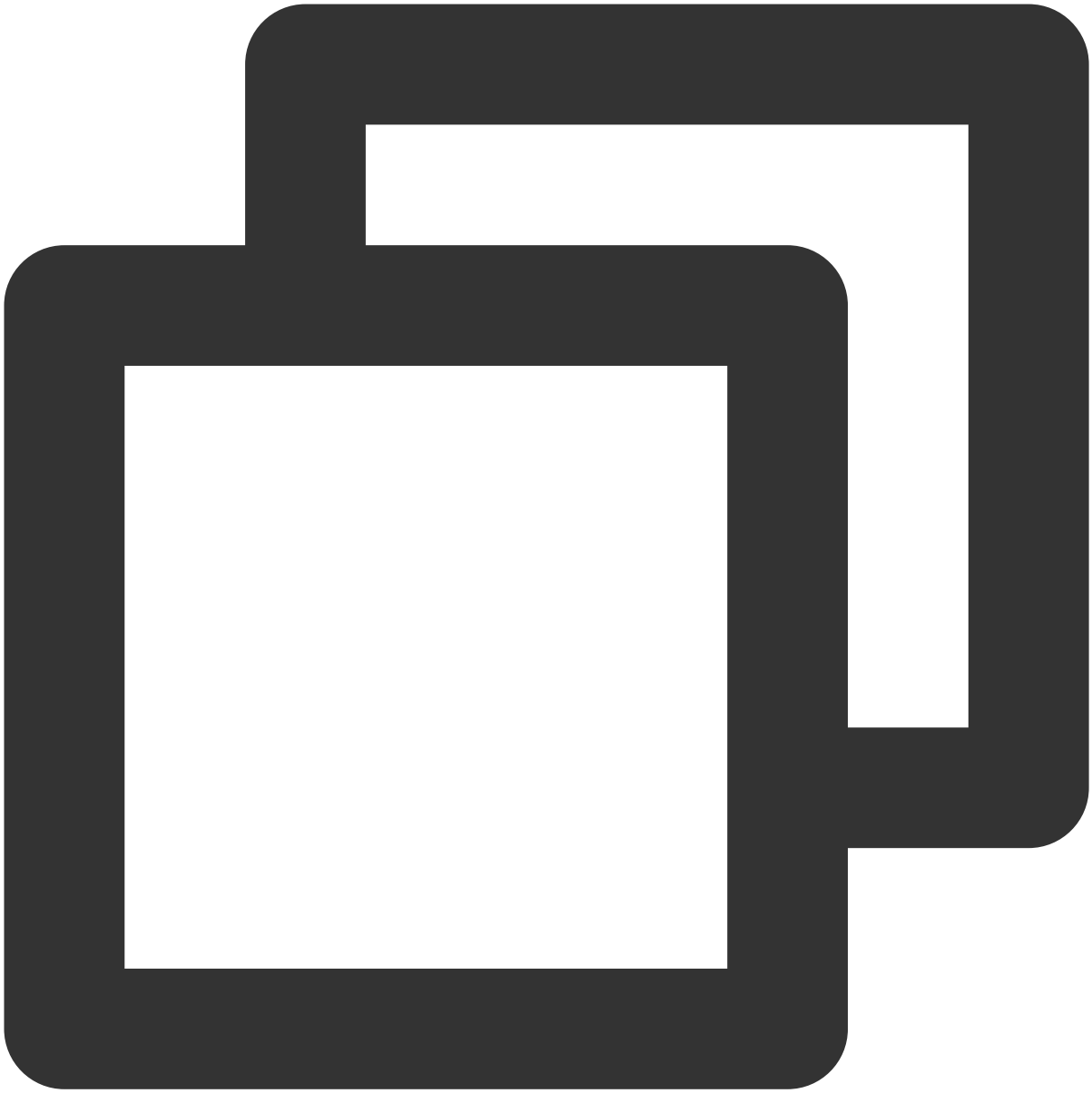
4. 修改您工程中的 `pubspec.yaml` ，将 `tencent_calls_uikit` 的依赖修改为本地依赖，示例如下：



```
tencent_calls_uikit:
  path: tencent_calls_uikit/   # 这里的目录对应您刚才下载的插件目录
```

2、Flutter Android未添加混淆设置，怎么设置？

如果您需要编译运行在 Android 平台，由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单，因此需要您在 proguard-rules.pro 文件中添加如下代码：

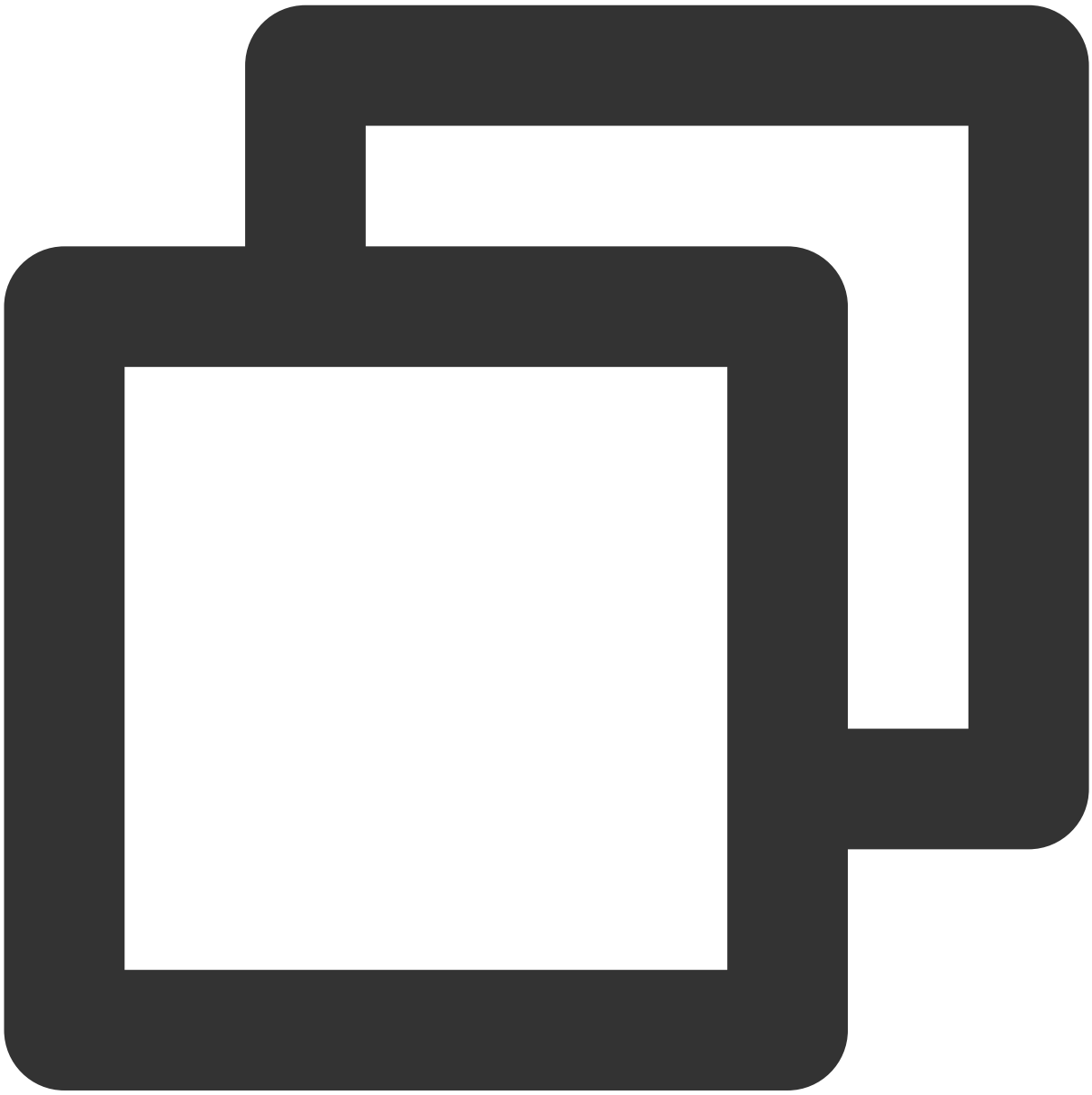


```
-keep class com.tencent.** { *; }
```

3、从1.8.0以下的版本升级到1.8.0及以上的版本，导致编译报错或者拉不起页面问题如何修复？

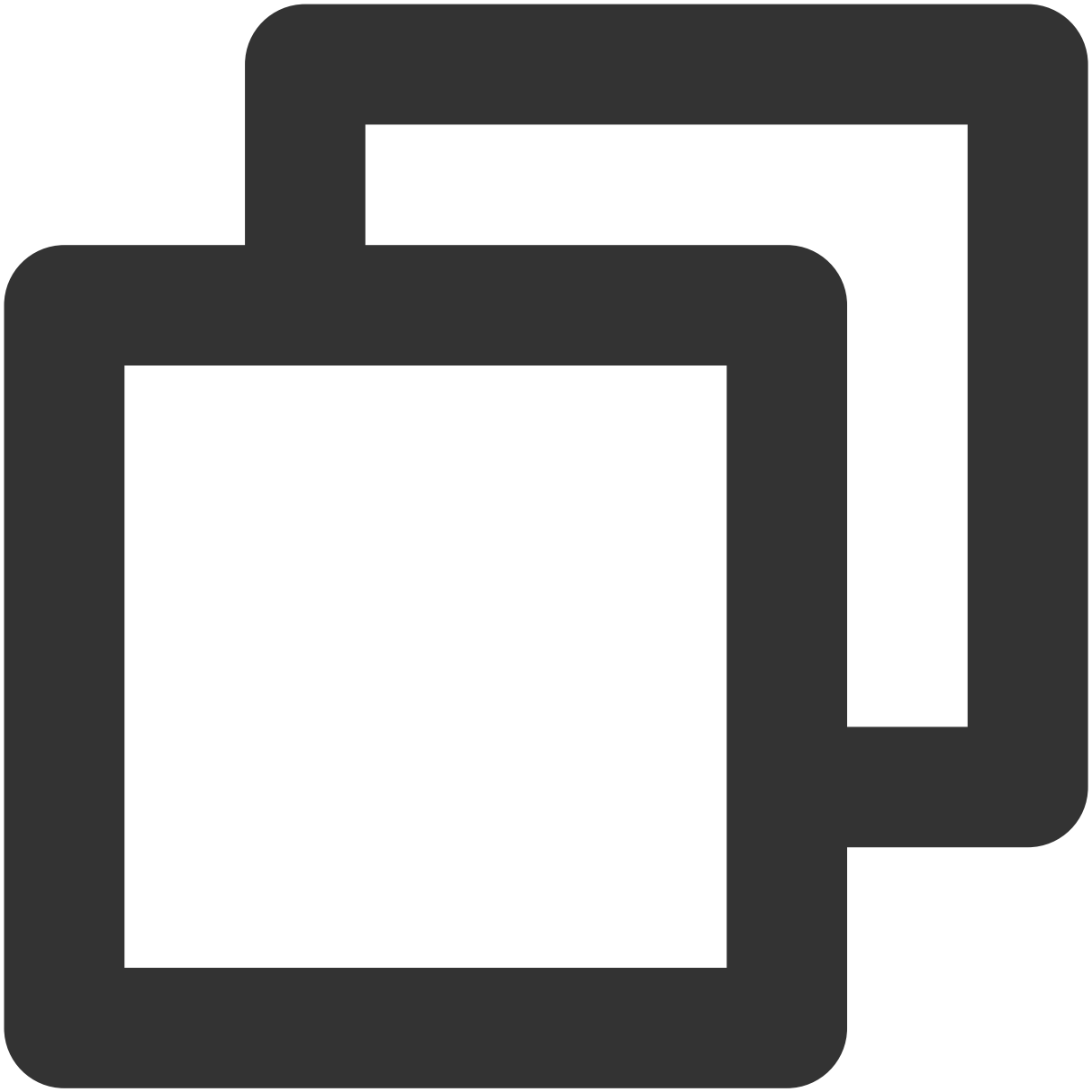
如果是从1.8.0以下升级到1.8.0及以上的版本，需要您检查下面几个步骤是否正常：

1. 将 `navigatorObservers` 添加到 `MateriaApp` 。 目的是在收到呼叫邀请时导航到 TUICallKit 页面。示例代码如下：



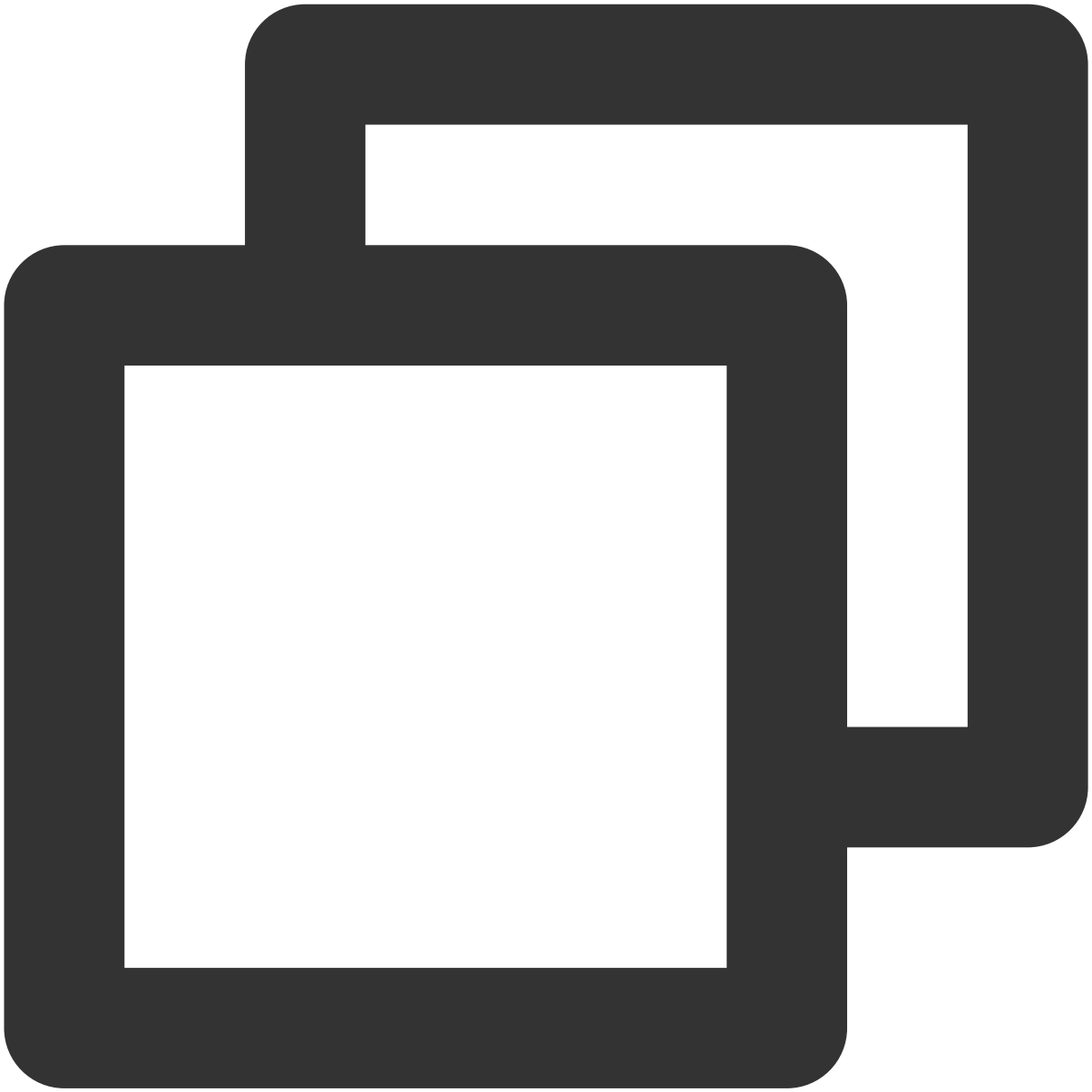
```
import 'package:tencent_calls_uikit/tuicall_kit.dart';

MaterialApp (
      navigatorObservers：[TUICallKit.navigatorObserver],
      ...
)
```

2. `tencent_calls_engine` 插件中的导入文件统一替换为新的。



```
import 'package:tencent_calls_engine/tuicall_engine.dart';
import 'package:tencent_calls_engine/tuicall_observer.dart';
import 'package:tencent_calls_engine/tuicall_define.dart';
```

以上代码块内容替换为下方所示：



```
import 'package:tencent_calls_engine/tencent_calls_engine.dart';
```

3. 登录 API 调整更规范，不需要再指定参数。



4. 离线推送参数构造优化。

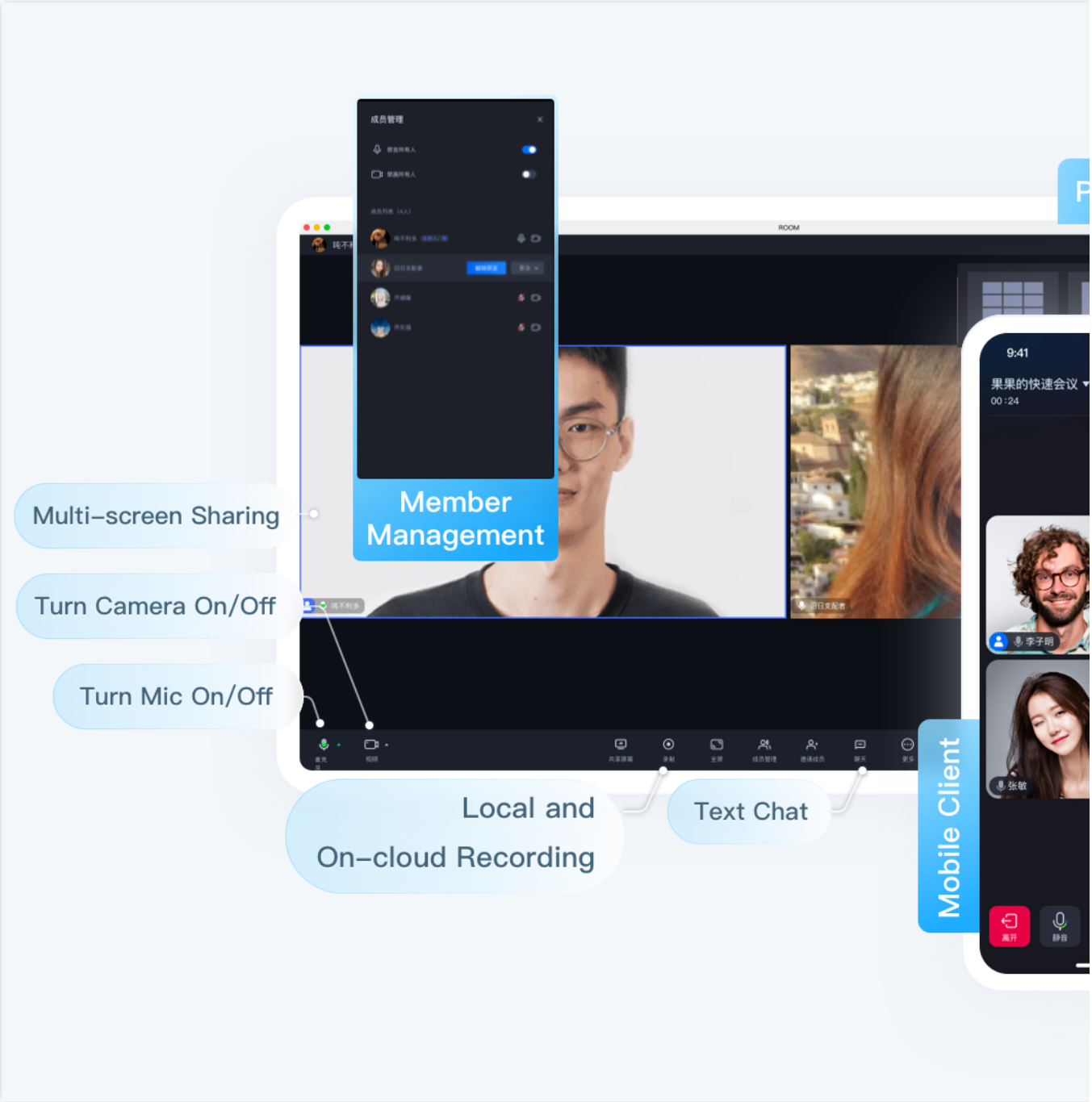
多人音视频
组件介绍 (TUIRoomKit)

最近更新时间：2024-04-12 11:19:05

组件介绍

Conference（TUIRoomKit）是一款适用于**企业会议、网络研讨会、在线教育**等多人音视频会话场景的产品，通过集成该产品，仅需三步一天内就可以为您的 App 添加房间管理、成员管理、屏幕共享等功能，快速上线业务。基本功能展示如下图：

版权所有：腾讯云计算（北京）有限责任公司



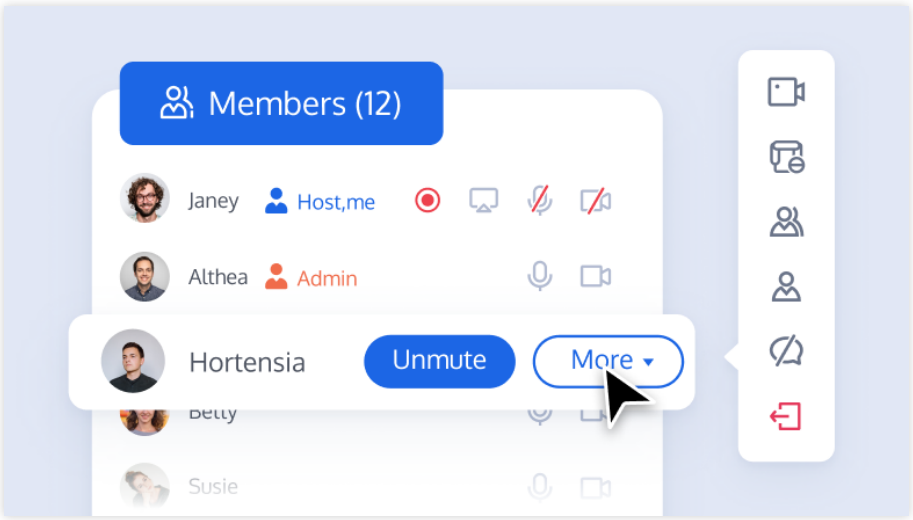
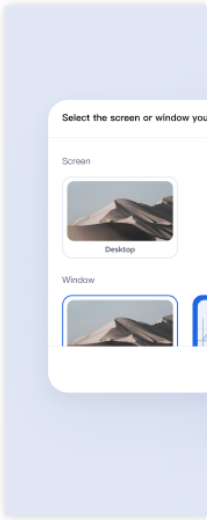
支持平台

平台	Android	iOS	Web	Flutter	Electron
是否支持					
支持语言/框架	Java	Swift	Vue3 Vue2	Dart	Vue3 Vue2

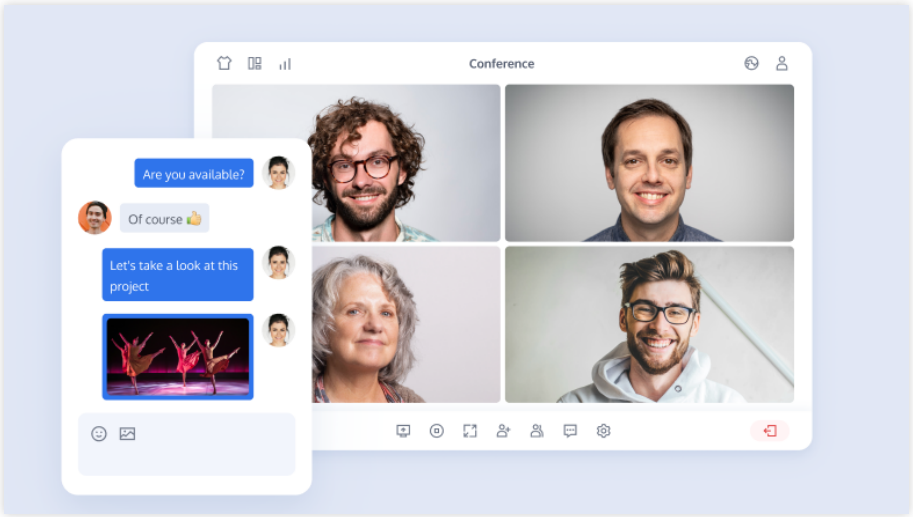
版权所有：腾讯云计算（北京）有限责任公司

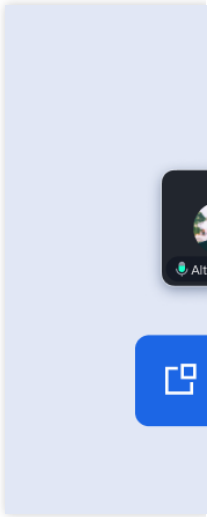
功能介绍

基本功能	高级功能	功能优势
多人视频会话 丰富的会议控制功能 支持上台发言模式 开启/关闭悬浮窗 音视频相关设置 屏幕共享	会中聊天 云端录制 AI 降噪	**快速集成** **完善的 UI 交互** **支持多平台互联互通** **支持多设备登录**

成员管理	屏幕分享
	

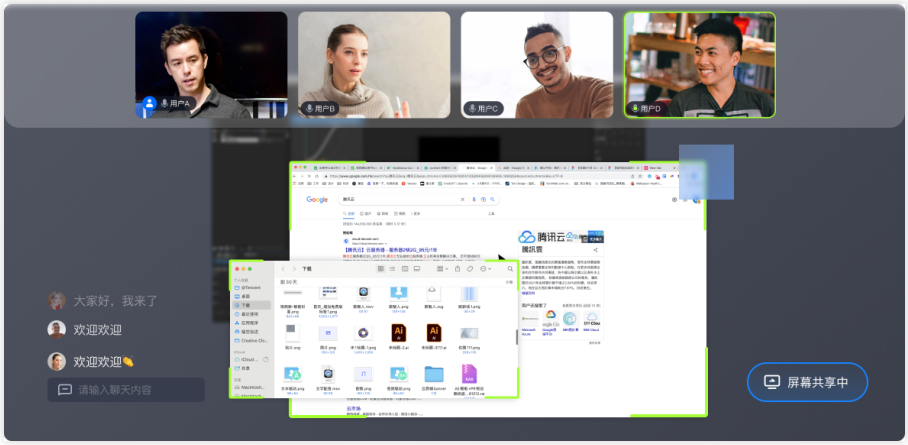
会中聊天	悬浮窗



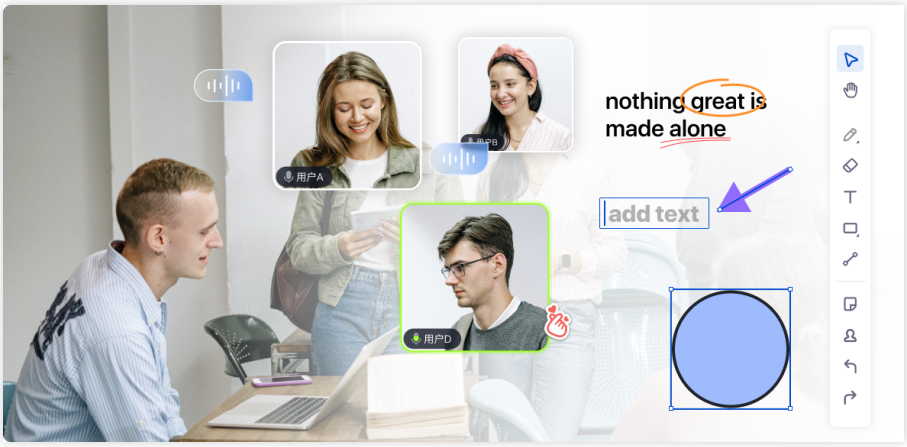


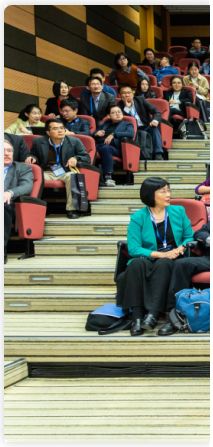
适用场景

多人进行音视频会话互动，覆盖企业会议、网络研讨会、在线教育、在线招聘、语聊房等场景。

音视频会议	在线教育
	

网络研讨会	企业培训





在线体验

平台	Web	Android	iOS	其他平台
在线体验				/
示例代码	Github: web	Github: Andorid	Github: iOS	Github

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

开通服务（TUIRoomKit）

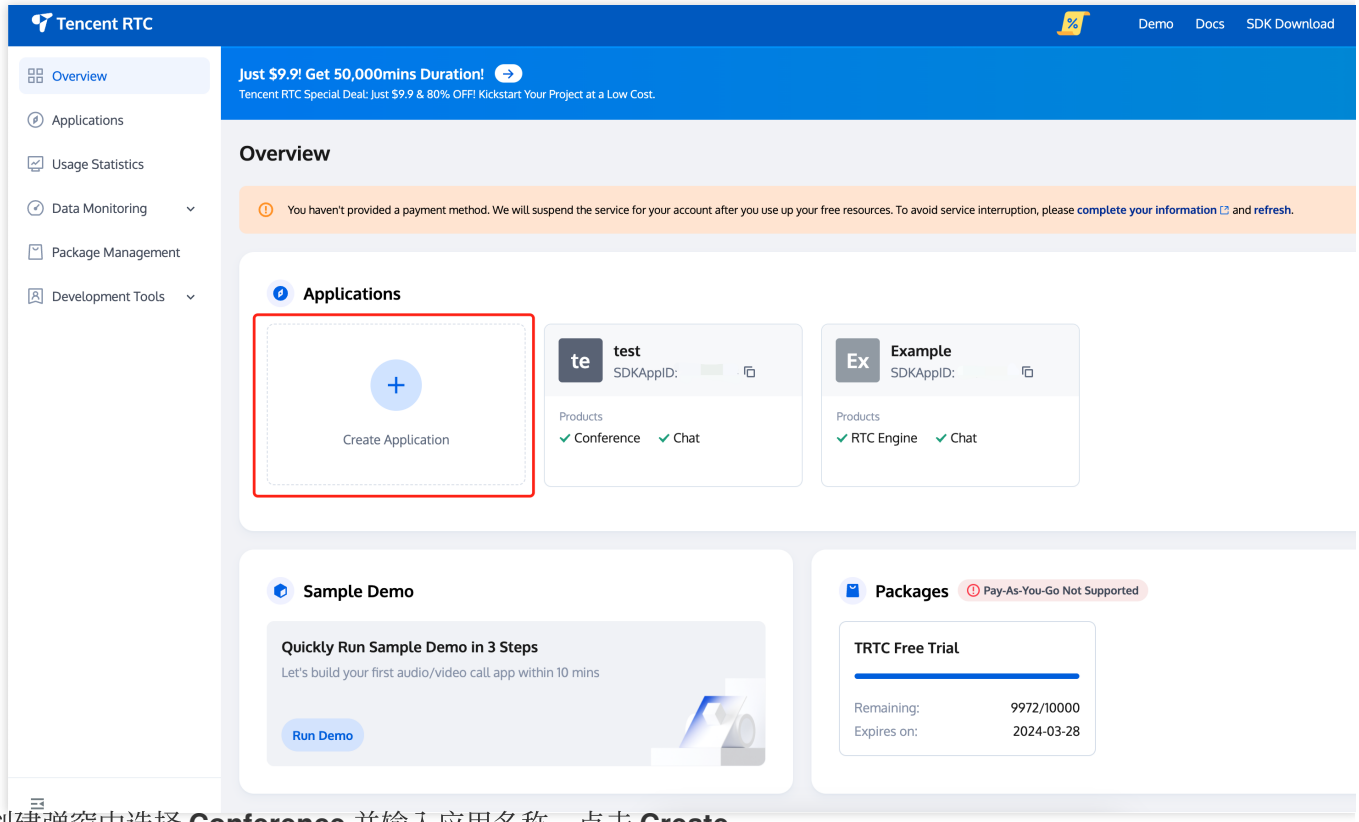
最近更新时间：2024-04-12 11:17:40

开通和购买

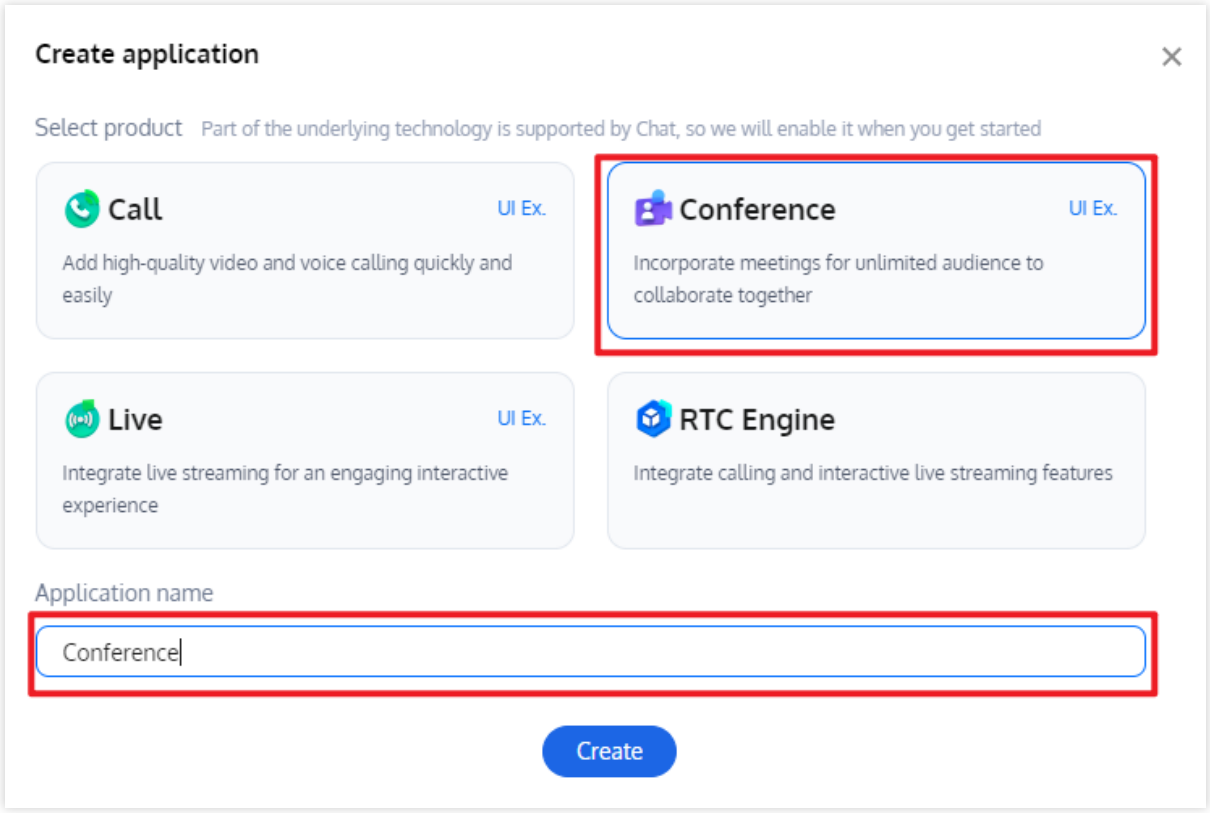
开通体验服务

您可以按照如下步骤开通 TRTC Conference 产品服务，并免费领取14天的体验版。

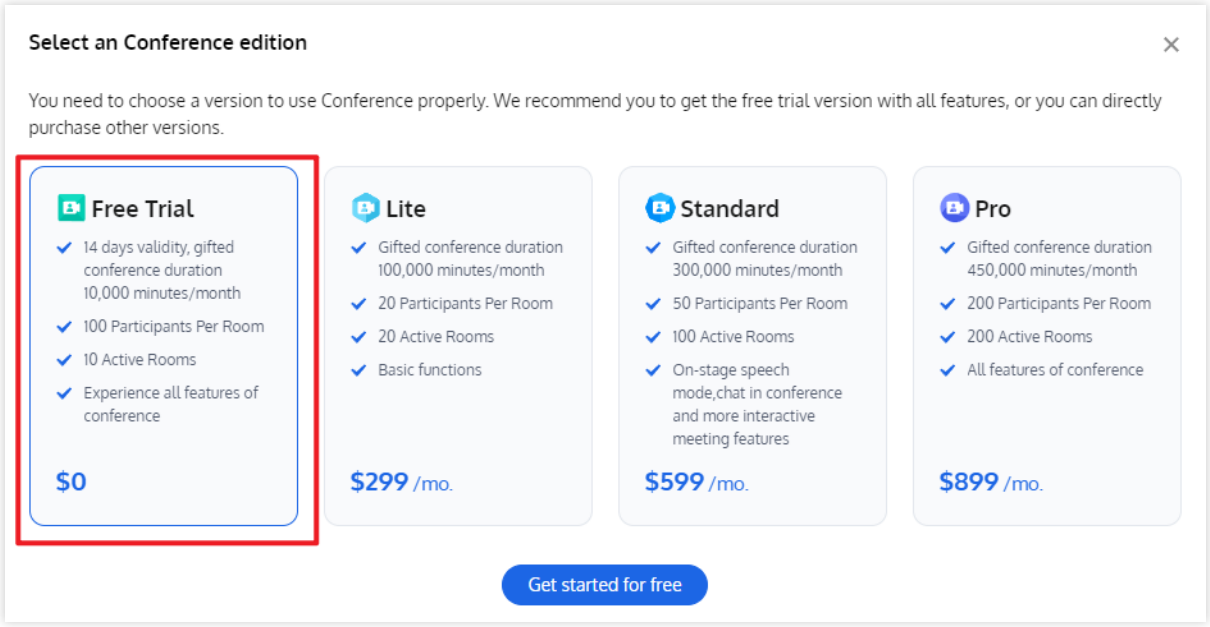
1. 登录到 实时音视频 TRTC 控制台，点击 **Create Application**。



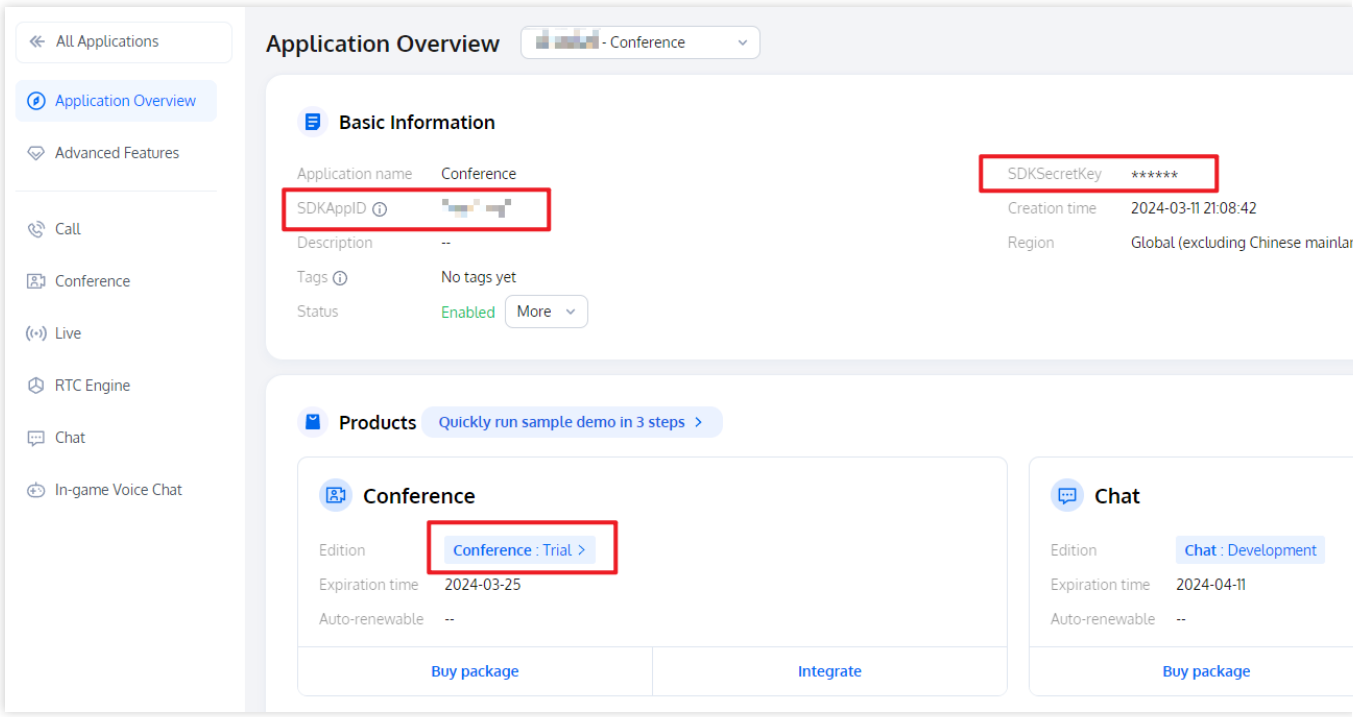
2. 在创建弹窗中选择 **Conference** 并输入应用名称，点击 **Create**。



3. 完成应用创建后将默认进入应用详情页，在推荐版本浮窗中选择 **Free Trial**，并点击 **Get started for free**。

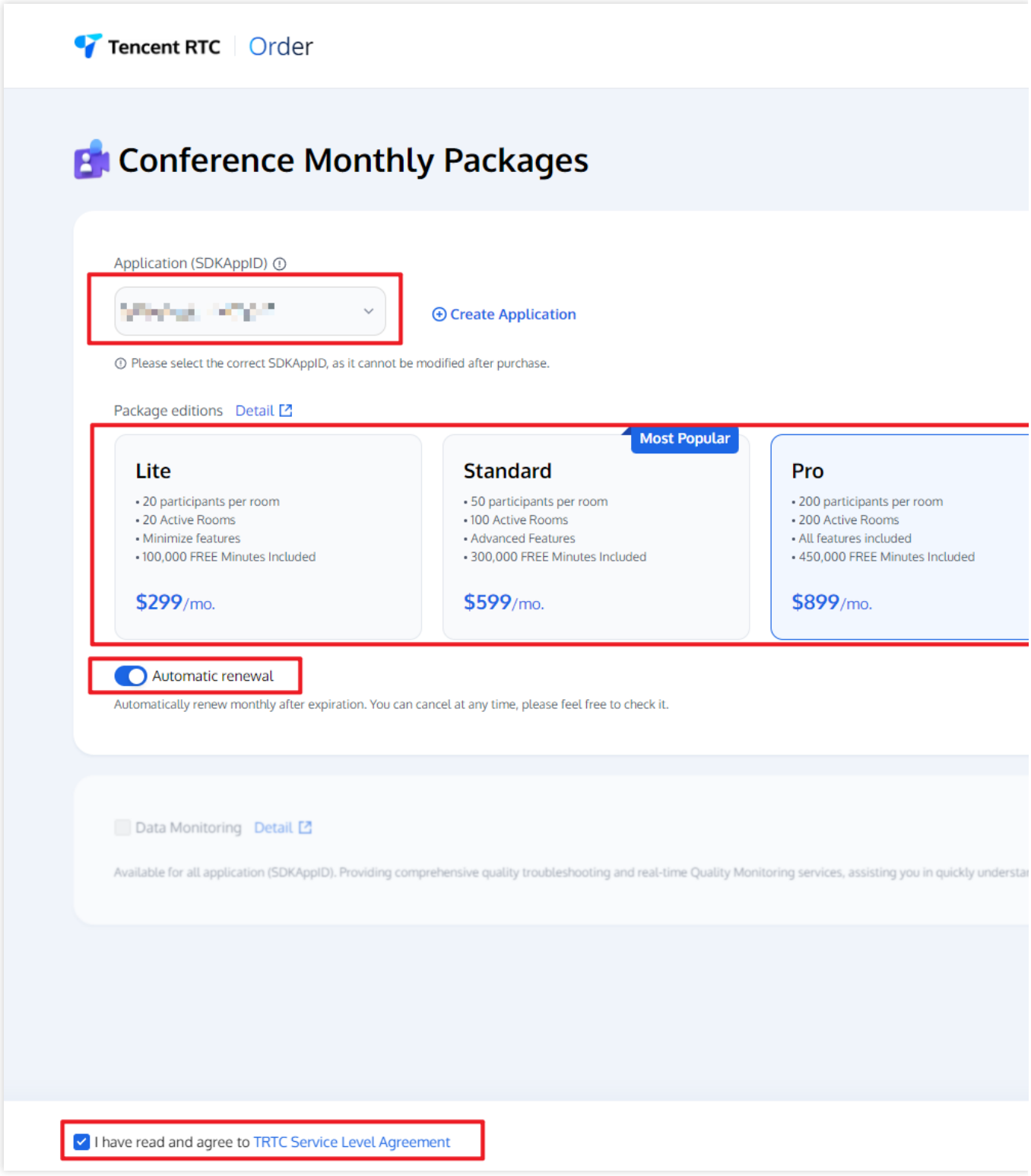


4. 体验版开通完成后可在当前页面查看信息，这里的 `SDKAppID` 、 `SDKSecretKey` 会在后续步骤中使用到。

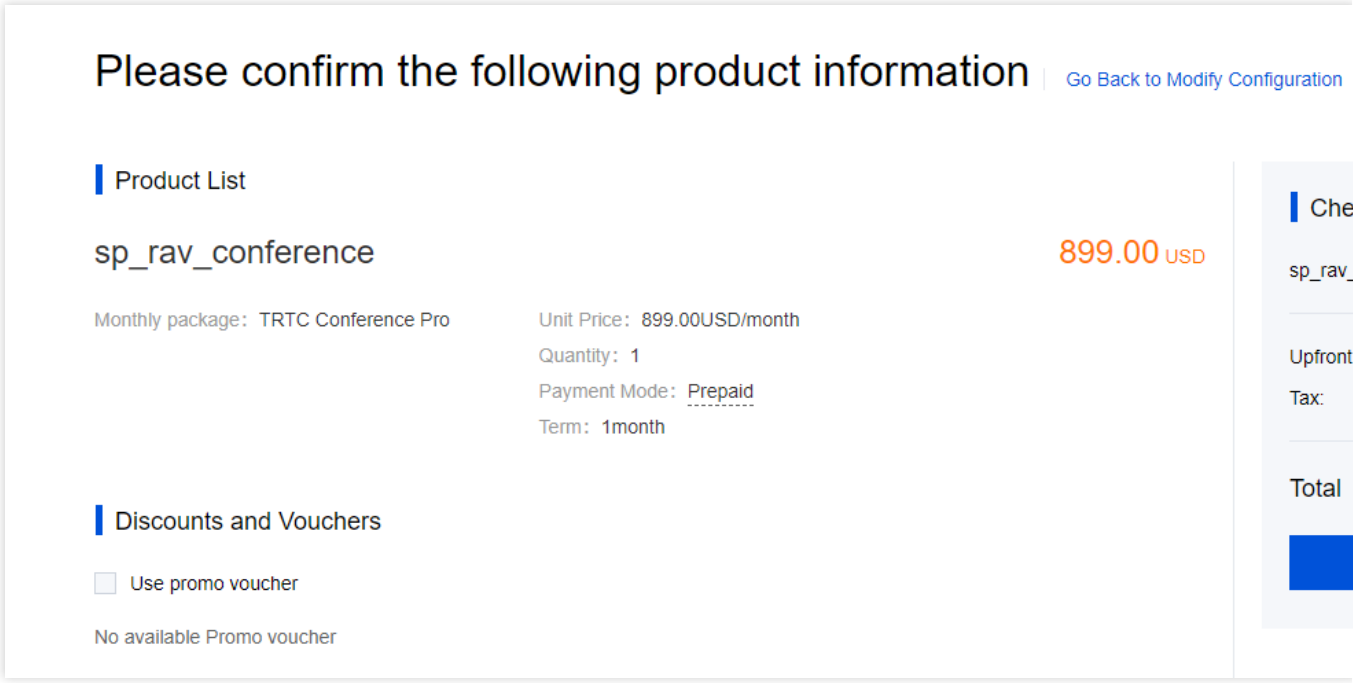


购买正式版

1. 访问 Conference 购买页面，选择需要购买的应用（SDKAppID）及版本，**同时建议您开启自动续期以避免影响业务使用**。确认购买信息后并同意相关协议后，勾选协议内容并点击 **Subscribe now**。



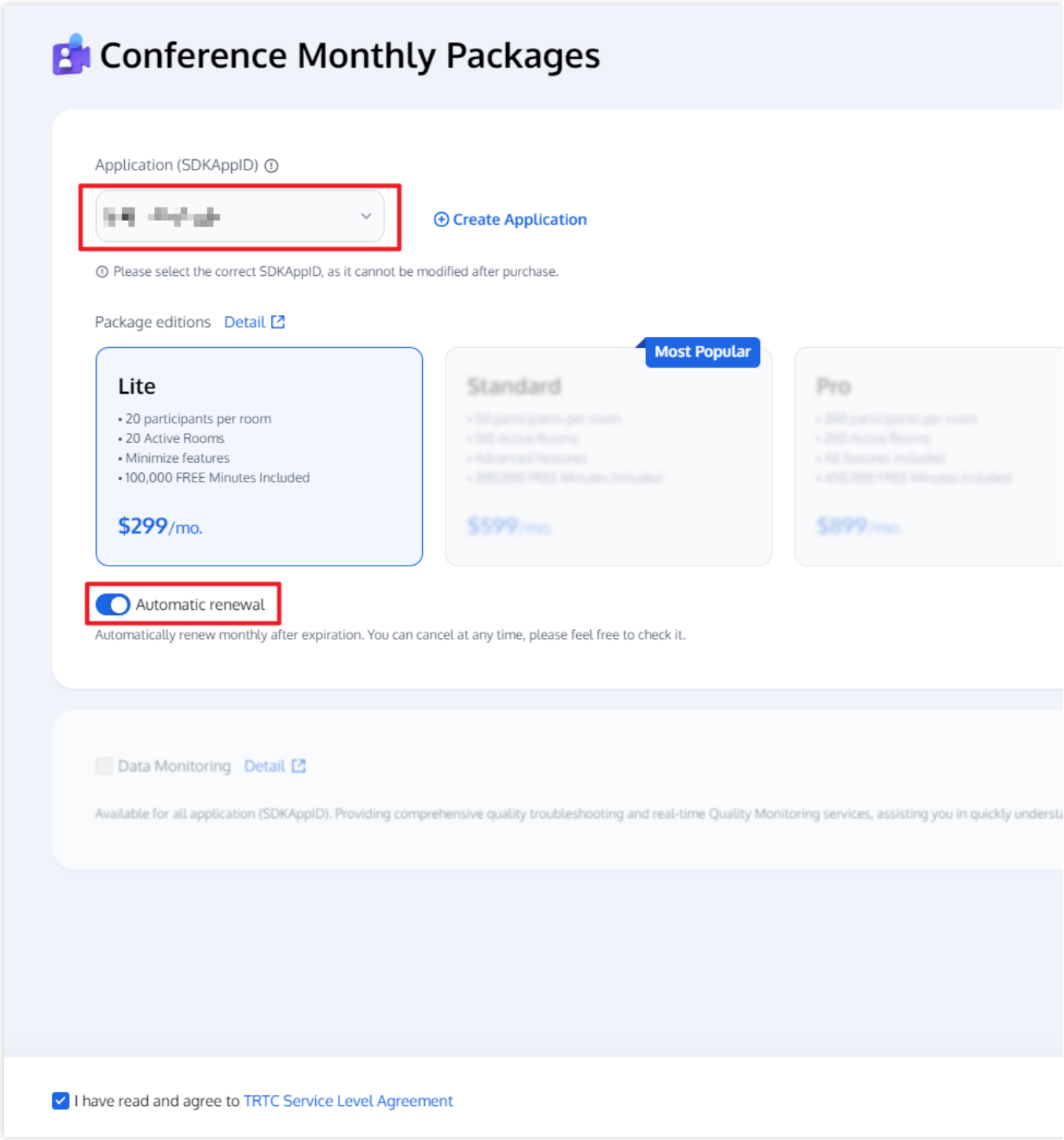
2. 前往订单确认页确认商品信息。



3. 前往支付页完成支付。购买完成后，您可前往 TRTC 控制台 查看应用版本信息，并可参见 集成指引 进行集成。

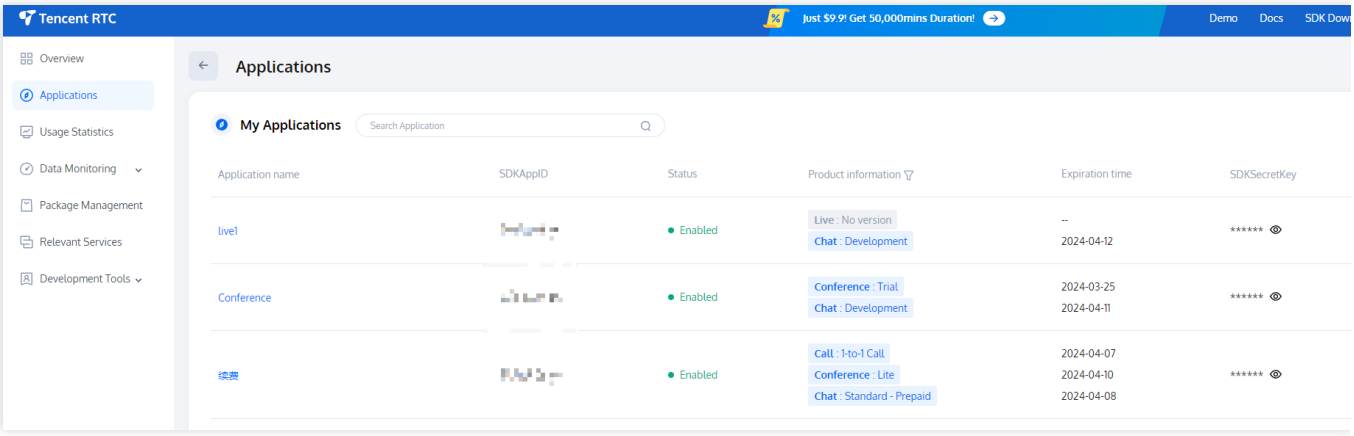
续费正式版

参考 购买正式版 的流程再次购买相同版本的 Conference 包月套餐，即可完成续费。建议您通过开启自动续期的方式为 Conference 续费，开启后将账户**余额足够时，到期后自动按月续费。**

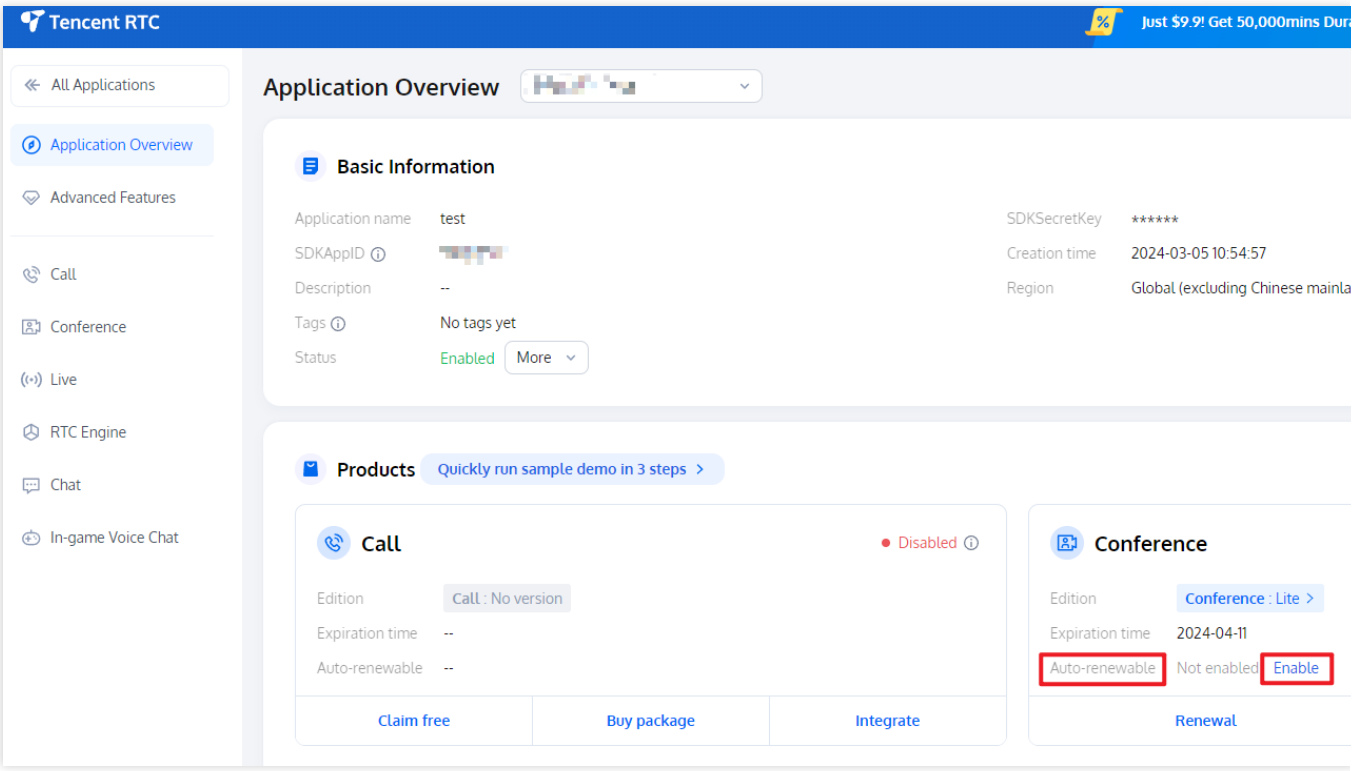


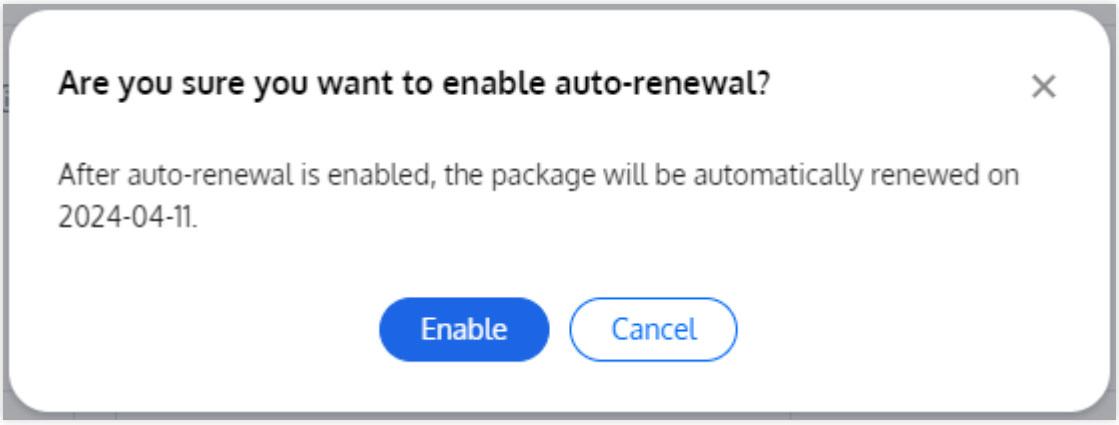
控制台开启自动续期的具体步骤如下：

1. 访问 TRTC 控制台 > Applications，选择需要开启自动续费的应用并点击 **Manage** 按钮进入应用详情页。



2. 在 Conference 产品信息中，点击 **Enable Auto-renewable** 按钮，将出现确认弹窗，点击 **Enable**。





功能及计费说明

您需要购买 Conference 包月套餐方可使用 TUIRoomKit，Conference 包月套餐的价格和功能对比参见表格，计费详细说明可参考Conference 包月套餐计费说明。

对比项		体验版	轻量版	标准版	专业版
价格		0 美元/14天起	立即购买	立即购买	立即购买
套餐时长	免费时长	10,000 分钟/月	10,000 分钟/月	10,000 分钟/月	10,000 分钟/月
	套餐赠送音视频时长	-	100,000 分钟/月	300,000 分钟/月	450,000 分钟/月
	超量后付费（套餐有效期内）	超量后服务不可用	✓	✓	✓
Conference 专属功能	单房间最大人数	100	20	50	200
	活跃房间数	10	20	100	200
	免费月活跃用户数（MAU）	100/月	10,000/月	10,000/月	10,000/月
	视频分辨率	HD, FHD, 2K, 4K	HD, FHD	HD, FHD	HD, FHD, 2K, 4K
	完整UI	✓	✓	✓	✓
	定制化UI	✓	✓	✓	✓

	会议悬浮窗	✓	✓	✓	✓
	屏幕分享	✓	✓	✓	✓
	成员管理	✓	✓	✓	✓
	房间管理	✓	✓	✓	✓
	自由发言模式	✓	✓	✓	✓
	音频默认混流	✓	✓	✓	✓
	上台发言模式	✓	-	✓	✓
	麦位管理	✓	-	✓	✓
	麦位数量	20	-	9	20
	房间中聊天	✓	-	✓	✓
	AI 降噪	✓	-	✓	✓
	弱网通话卡顿优化	✓	-	✓	✓
	多端登录	✓	-	-	✓
TRTC 通用功能	云端录制	✓	✓	✓	✓
	云端混流转码	✓	✓	✓	✓
	旁路转推云直播	✓	✓	✓	✓
支持平台		iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows	iOS, Android, Web, Flutter, Electron, Windows
技术支持		-	需求响应时间：（5*12小时） P1-2小时 P2-6小时 P3-12小时	需求响应时间：（5*12小时） P1-2小时 P2-6小时 P3-12小时	需求响应时间（7*24小时） P1-1小时 P2-4小时 P3-8小时
		-	工单及邮件支持	工单及邮件支持	工单及邮件支持

	-	Telegram群聊支持	Telegram群聊支持	Telegram群聊支持

说明：

1. 套餐时长：可用于抵扣使用 Conference 过程中产生的时长用量。Conference 体验版用量超出套餐时长后将无法继续使用；轻量版、标准版、专业版在套餐有效期内产生超出套餐时长部分的用量将默认启用后付费模式计费。
2. 免费时长：所有开通 Conference 的腾讯云账号（UIN）每月可获赠一个10,000分钟的免费时长包，可用于抵扣使用 Conference 专属功能时在房间内产生的音视频时长，以及使用 TRTC 通用功能产生的云端录制和混流转码时长，更多说明见 免费时长说明。
3. 套餐赠送音视频时长：可用于抵扣使用 Conference 专属功能时在房间内产生的音视频时长，有效期一个月，到期余量失效。
4. TRTC 通用功能：您在使用 Conference 专属功能的同时，也可使用 TRTC 提供的通用功能，使用过程中将产生对应的用量，计费标准可参见 云端录制、混流转推。

快速接入 (TUIRoomKit)
iOS

最近更新时间：2024-04-12 11:29:00

本文将介绍如何用最短的时间完成 `TUIRoomKit` 组件的接入，跟随本文档，您将在一个小时的时间内完成如下几个关键步骤，并最终得到一个包含完备 UI 界面的音视频会议功能。

环境准备

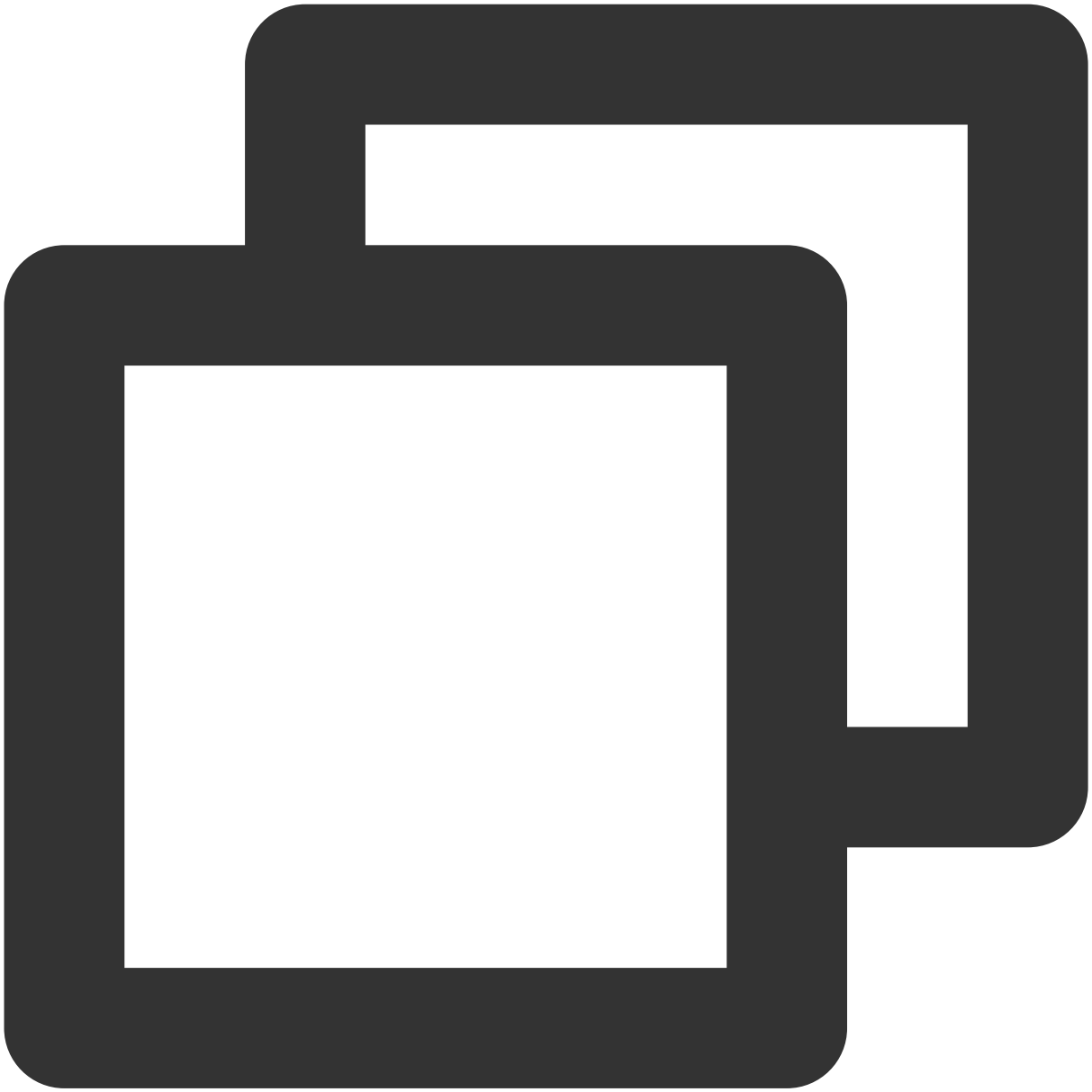
iOS 13.0 及更高。
Xcode 12.0 及更高。
Swift 4.2 及更高。

开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

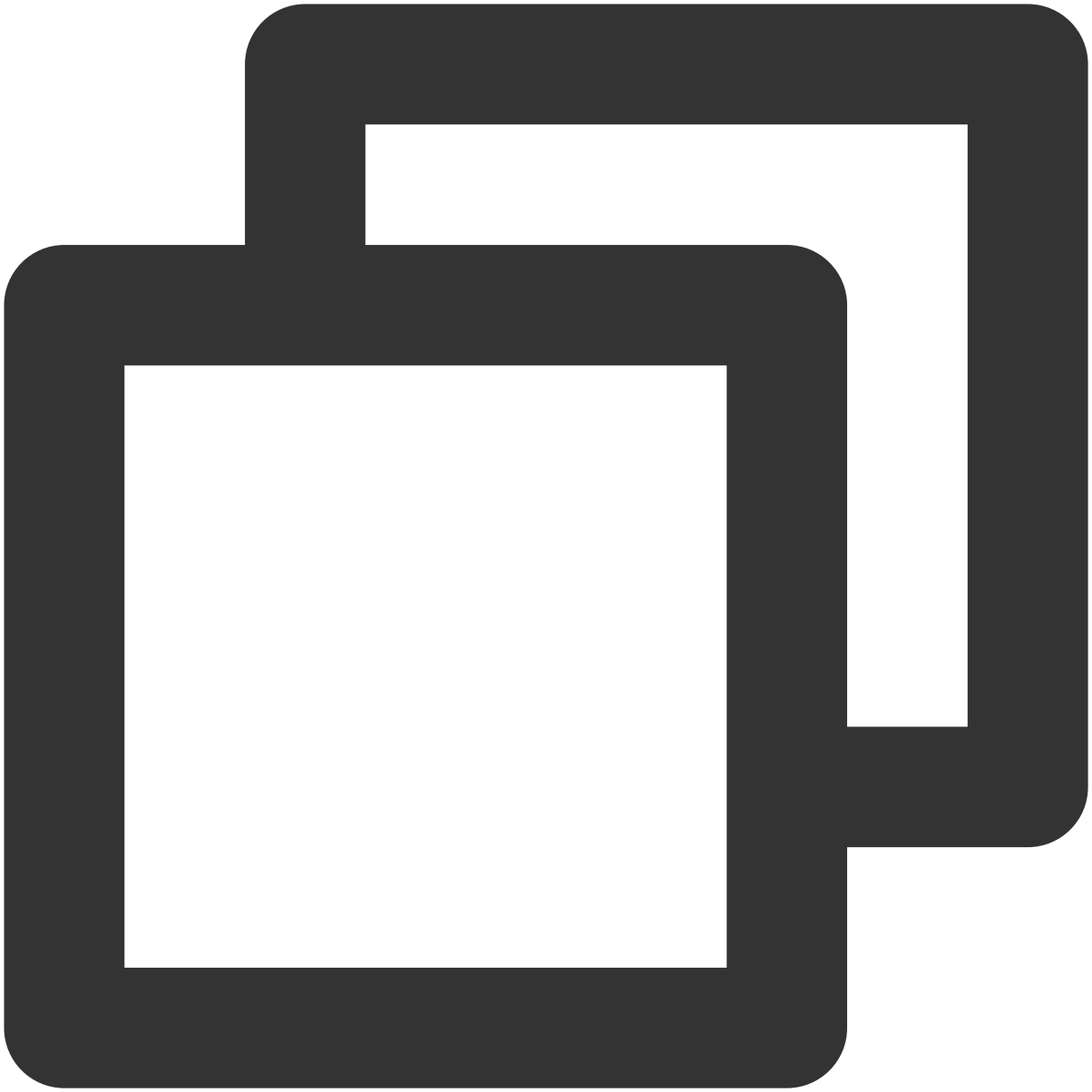
方式一：拉取远程 CocoaPods 集成（无源码修改时推荐）

1. 在您的 `Podfile` 文件中添加以下依赖。



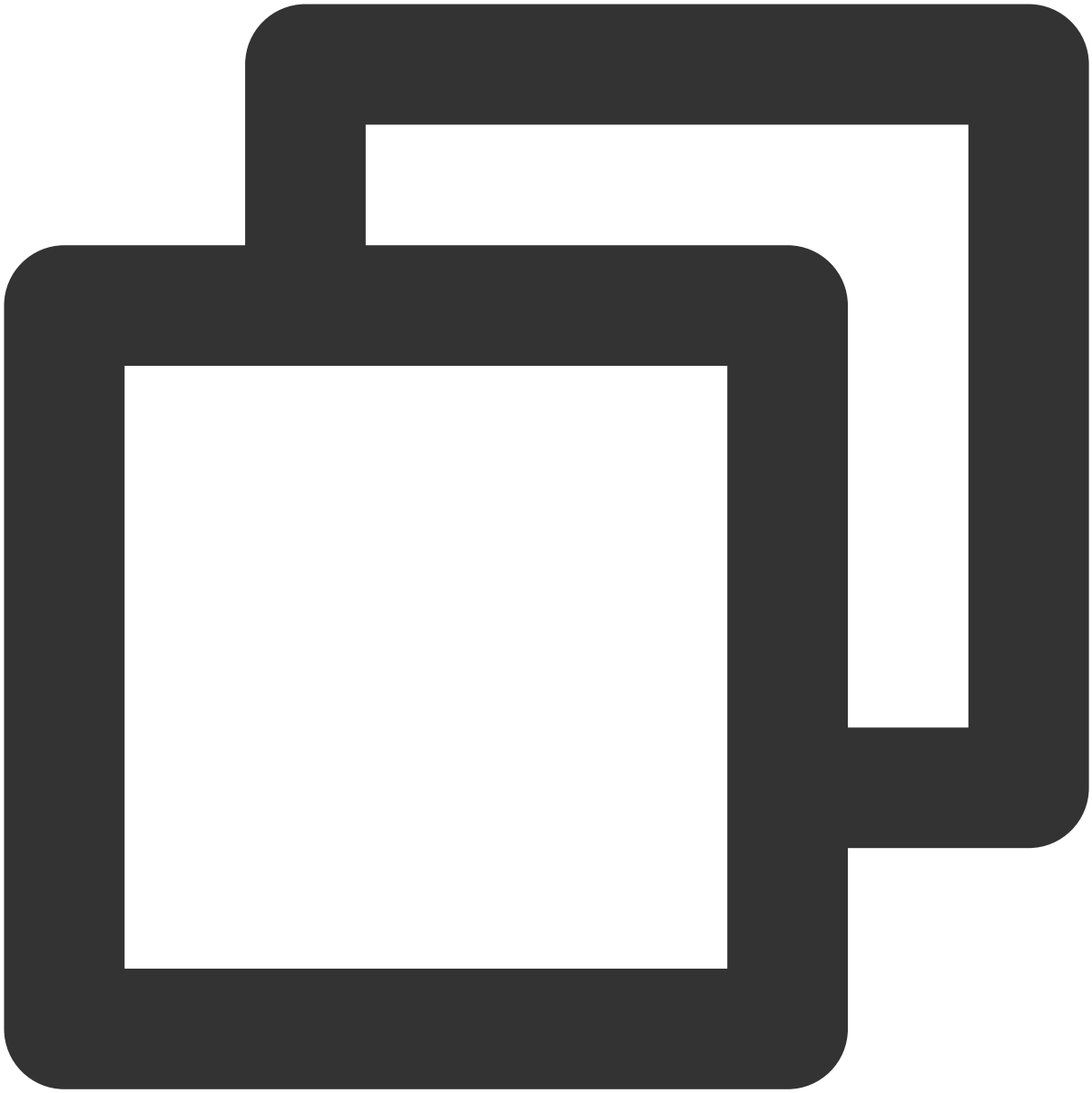
```
pod 'TUIRoomKit'
```

2. 执行以下命令，安装组件。



```
pod install
```

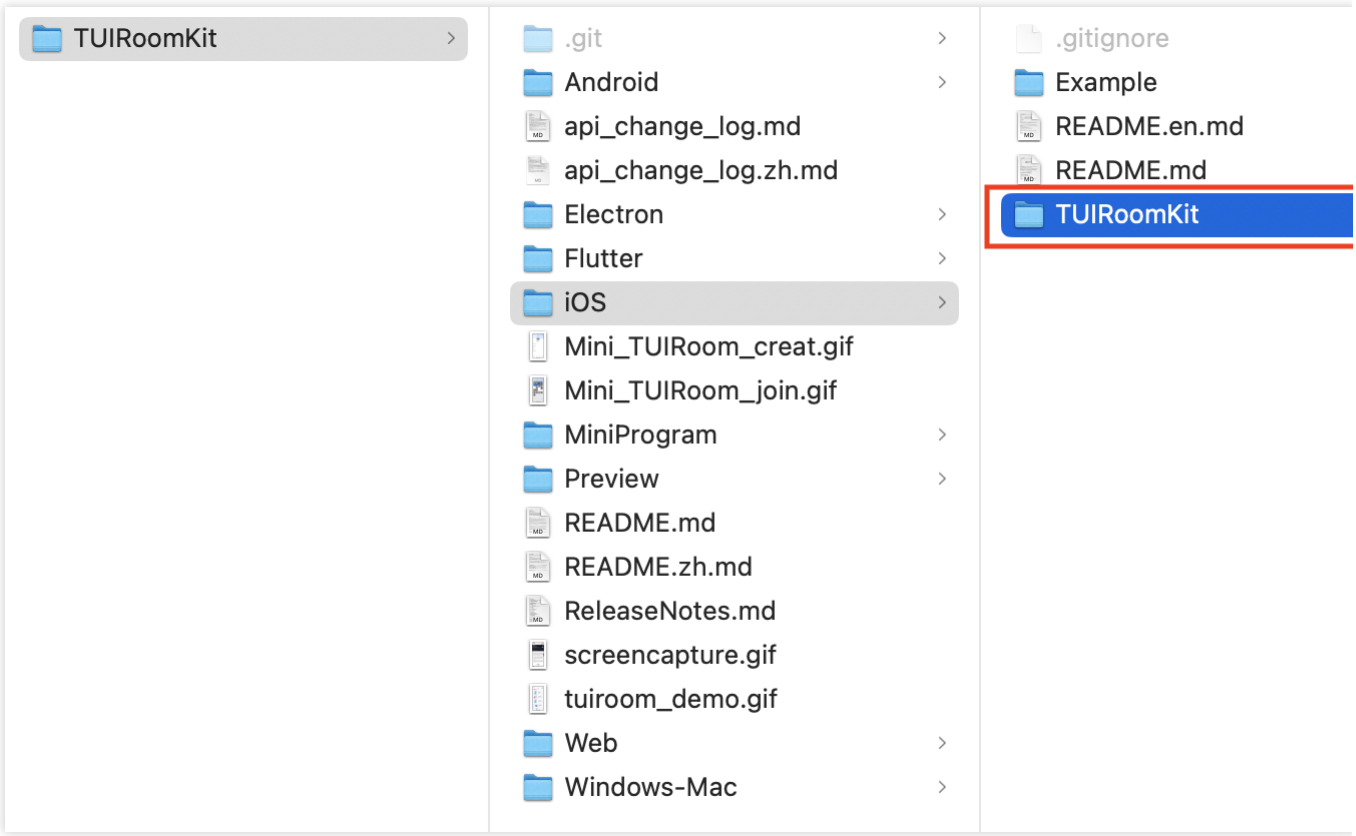
3. 如果无法安装 TUIRoomKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。



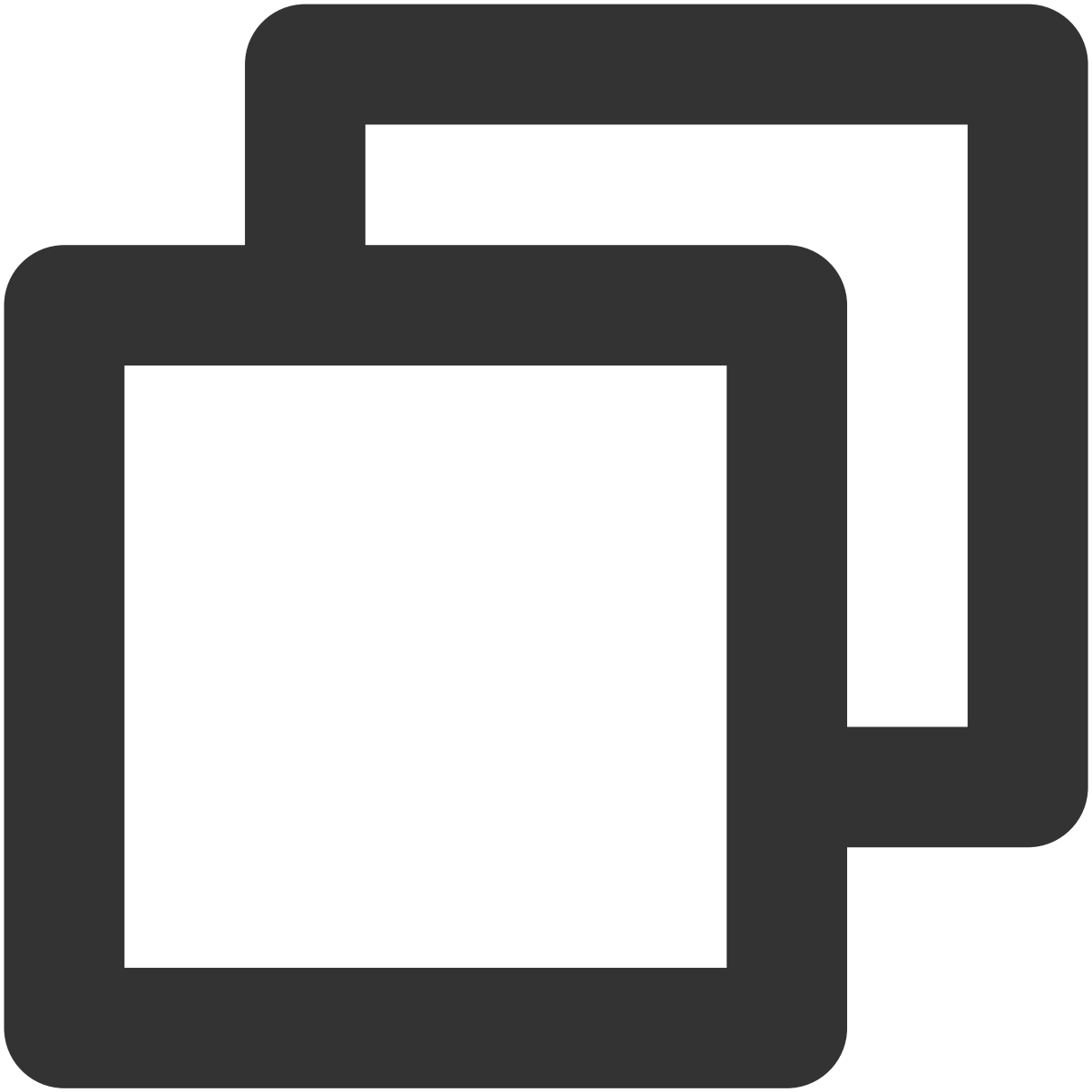
```
pod repo update
```

方式二：源码集成（有源码修改时推荐）

1. 从 GitHub 下载 TUIRoomKit 源码。直接拖入您的工程目录下: TUIRoomKit/iOS/TUIRoomKit。

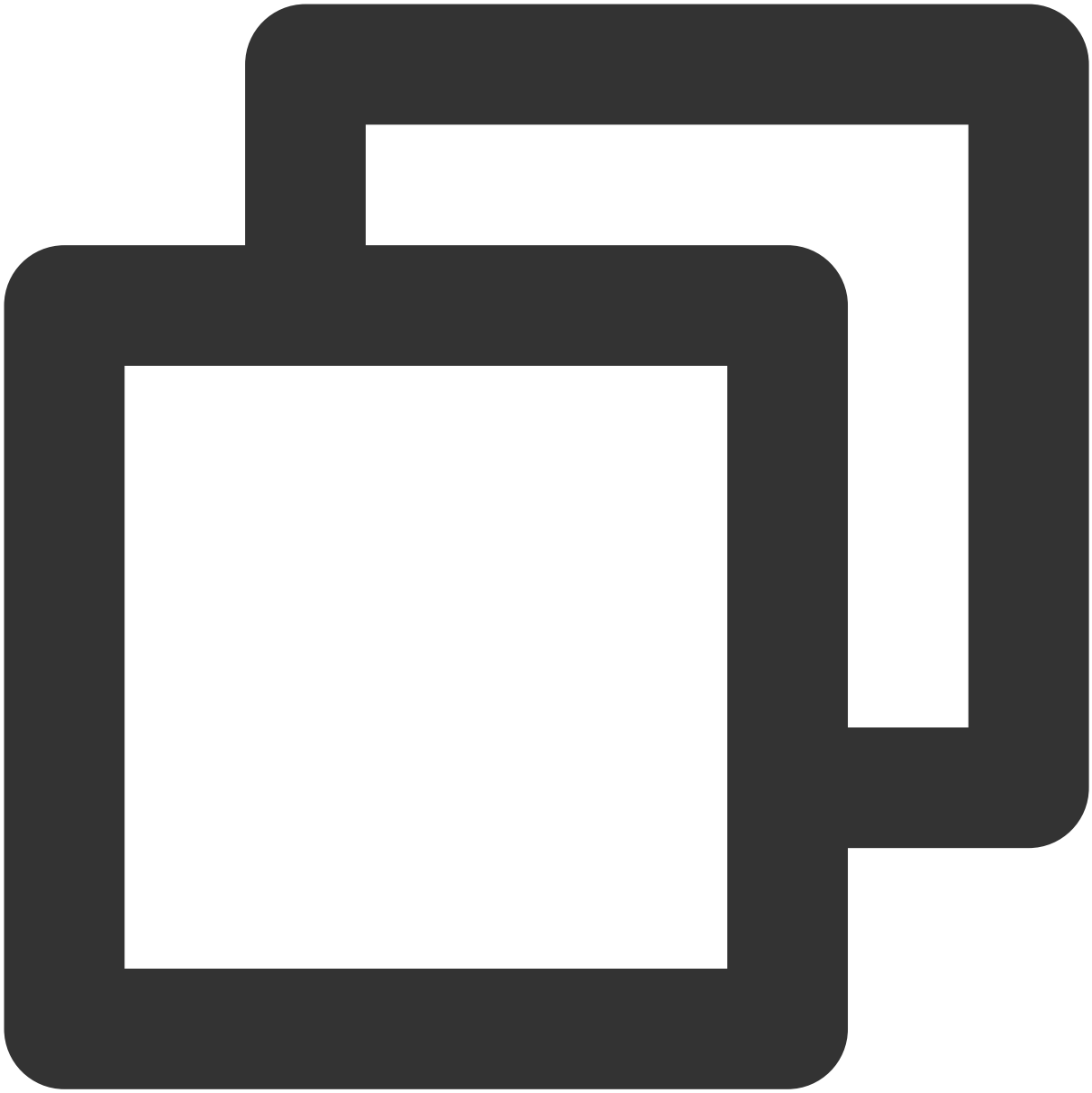


2. 修改您Podfile中TUIRoomKit组件的路径，例如：



```
pod 'TUIRoomKit', :path => "../TUIRoomKit/"
```

3. 执行安装命令

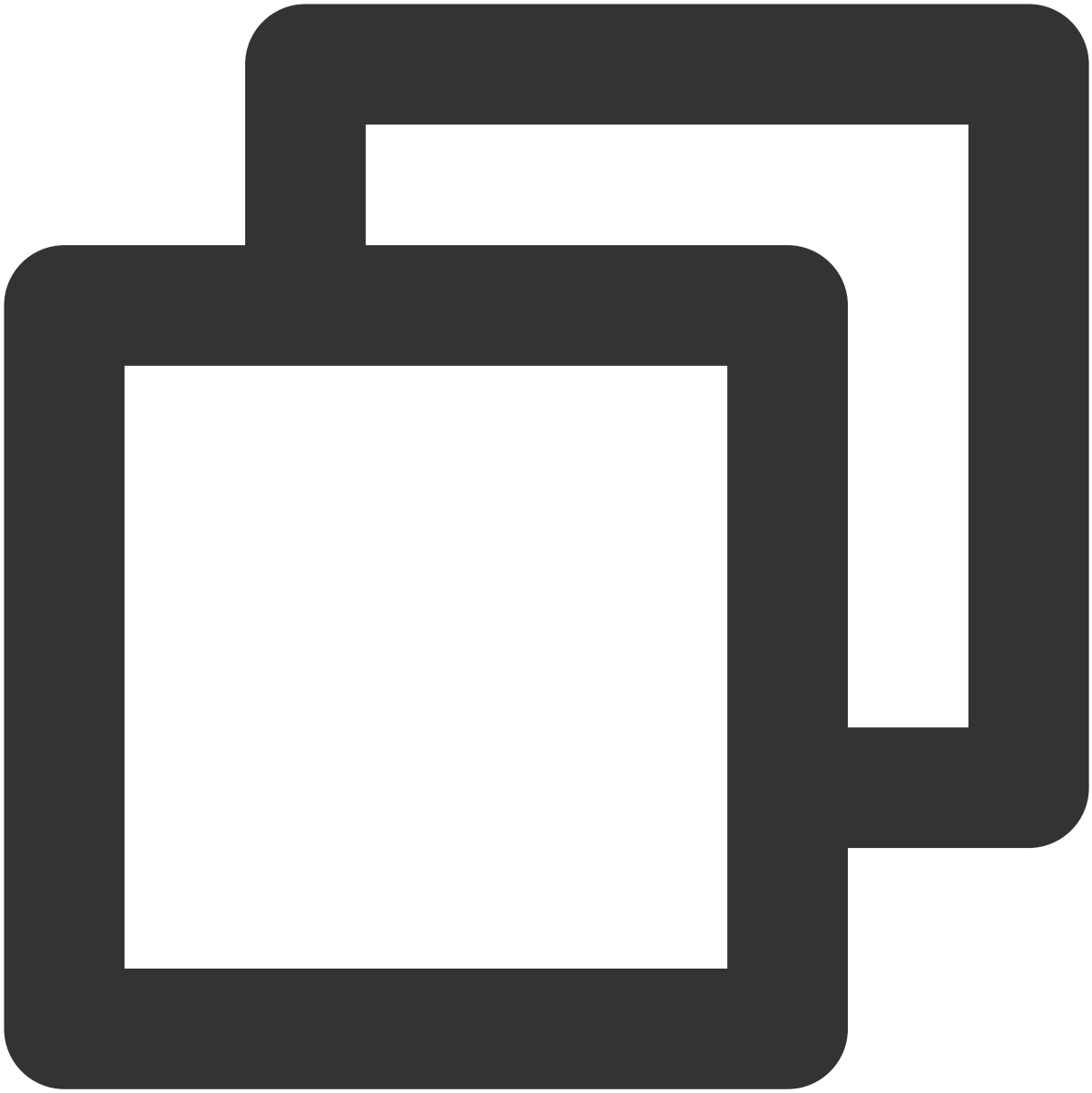


```
pod install
```

快速搭建

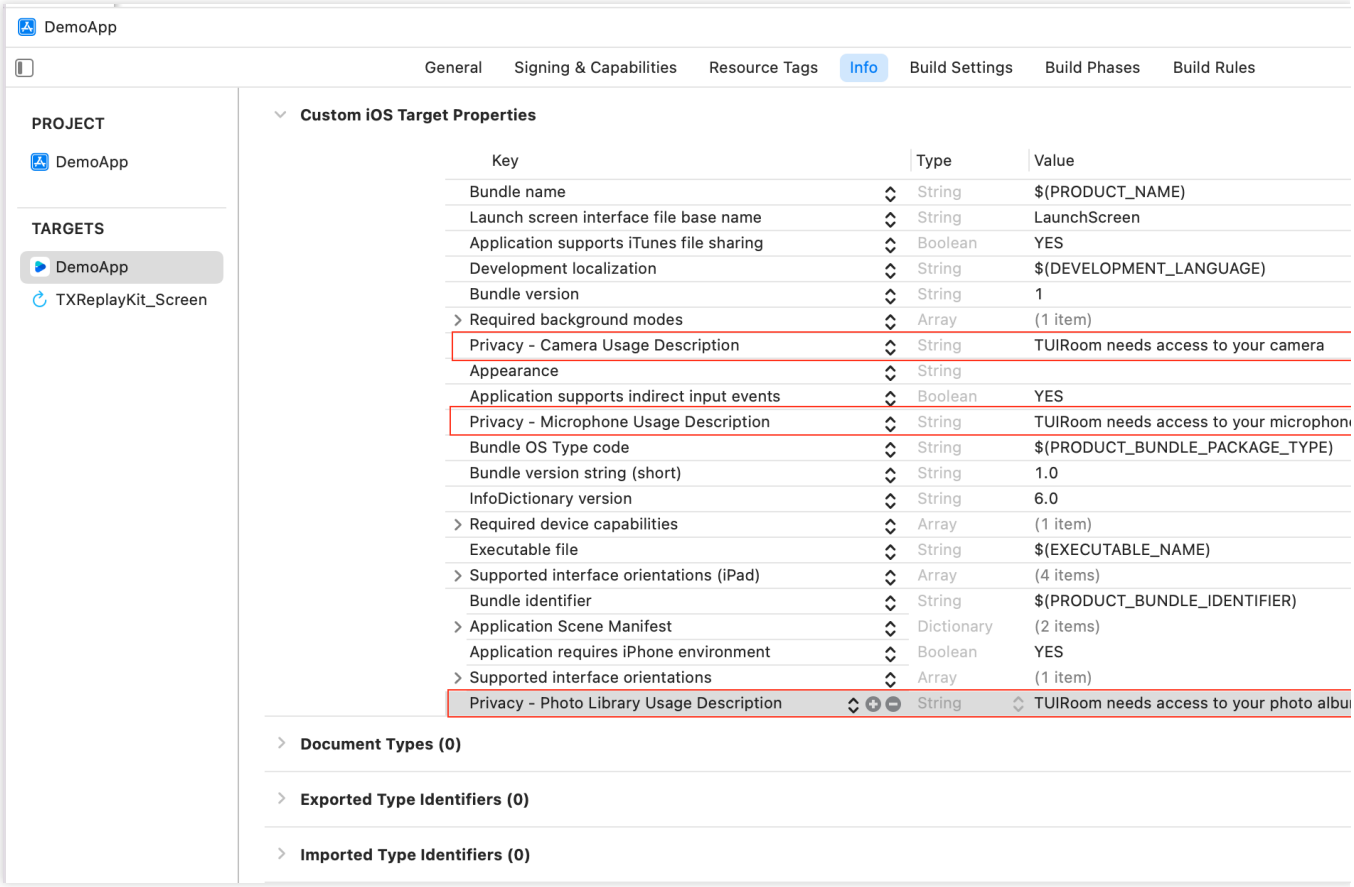
步骤1：工程配置

使用音视频功能，需要授权麦克风、摄像头和相册的使用权限。在 App 的 Info.plist 中添加以下几项，分别对应麦克风、摄像头和相册在系统弹出授权对话框时的提示信息。



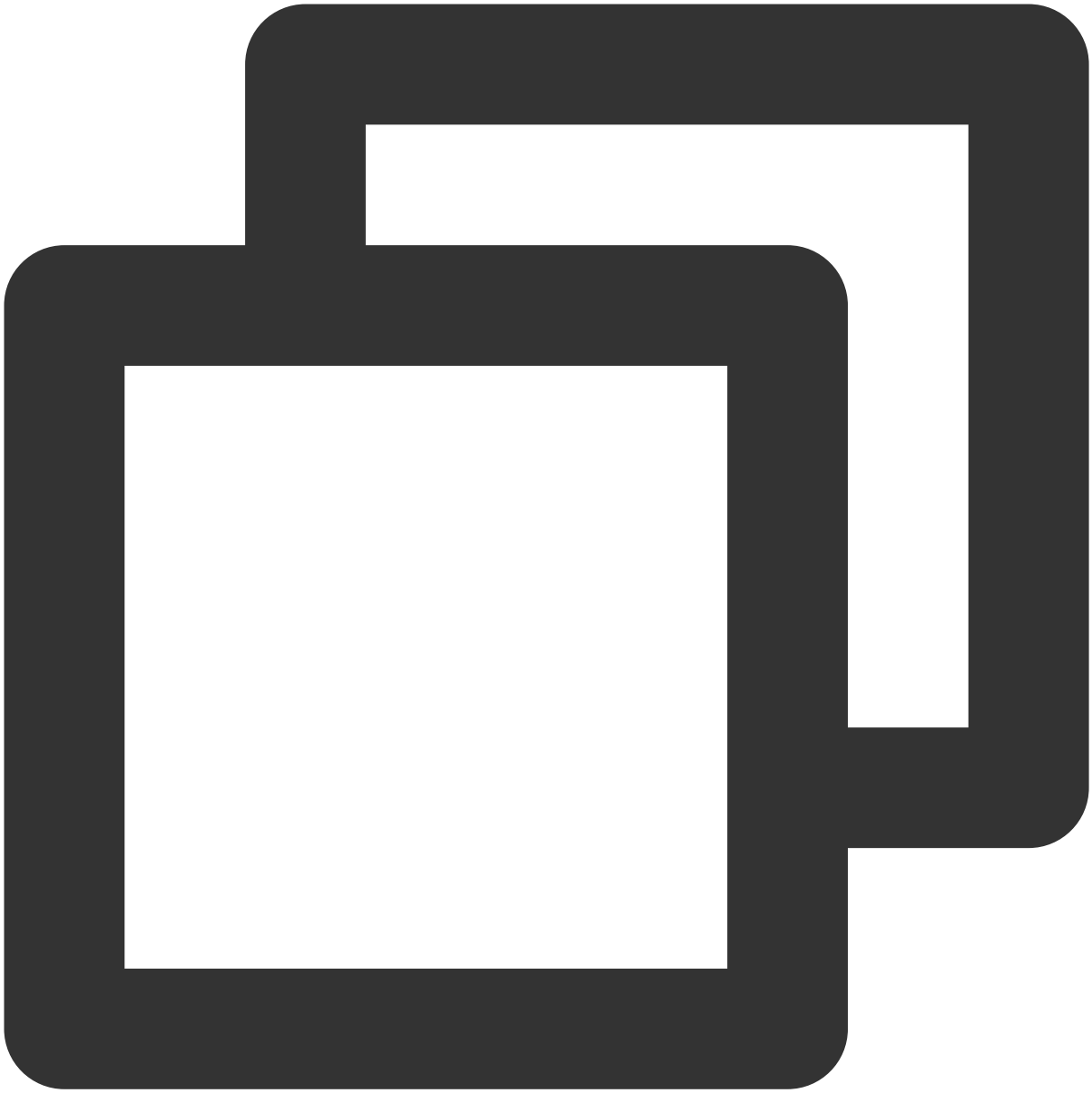
```
<key>NSCameraUsageDescription</key>
<string>TUIRoom needs permission to access your camera</string>
<key>NSMicrophoneUsageDescription</key>
<string>TUIRoom needs access to your photo album</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>TUIRoom needs access to your photo album</string>
```

版权所有：腾讯云计算（北京）有限责任公司



步骤2：组件登录

在您的项目中添加如下代码，它的作用是通过调用TUICore 中的相关接口完成TUI组件的初始化。这个步骤非常关键，因为只有在登录成功后才能正常使用 TUIRoomKit 的各项功能，故请您耐心检查相关参数是否配置正确：



```
import TUICore

TUILogin.login(1400000001,                                  // 请替换为步骤一取到的 SDKAppID
            userID: "998",                                  // 请替换为您的 UserID
           userSig: "xxxxxxxxxx") {                         // 您可以在控制台中计算一个 UserSig 并
    print("login success")
} fail: { (code, message) in
    print("login failed, code: \\(code), error: \\(message ?? "nil")")
}
```

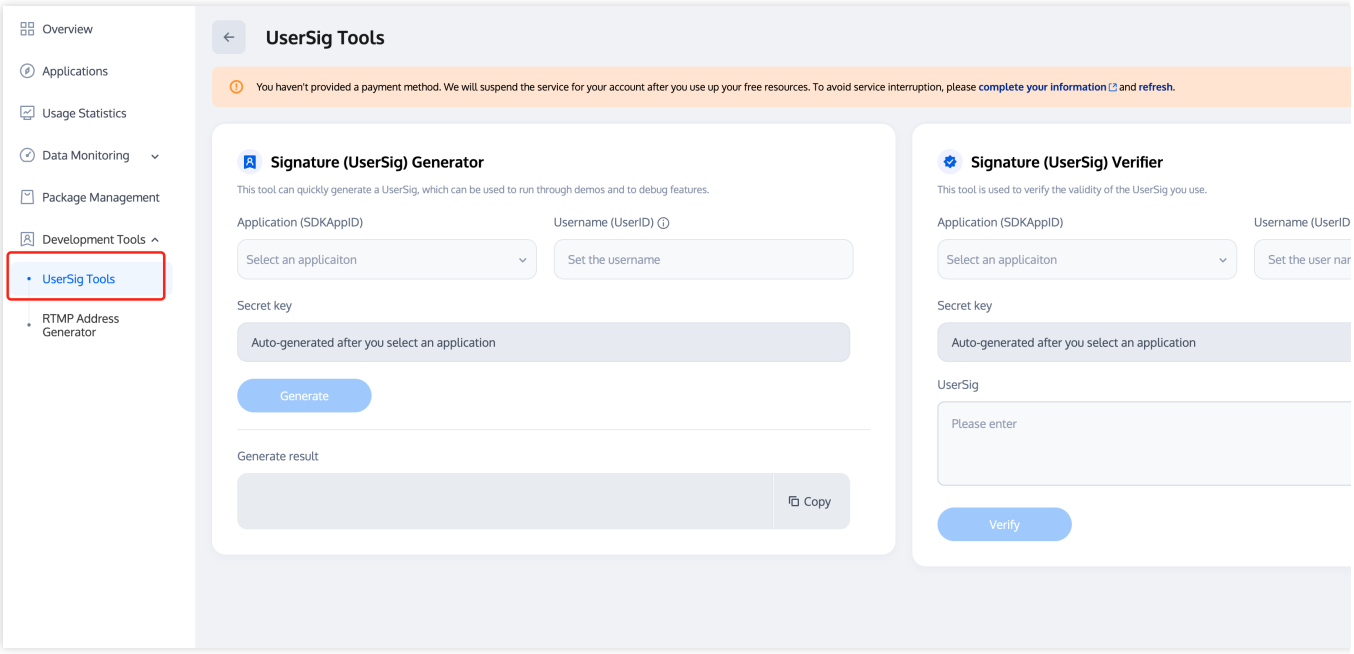
参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

SDKAppID：在开通服务中您已经获取到，这里不再赘述。

UserID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

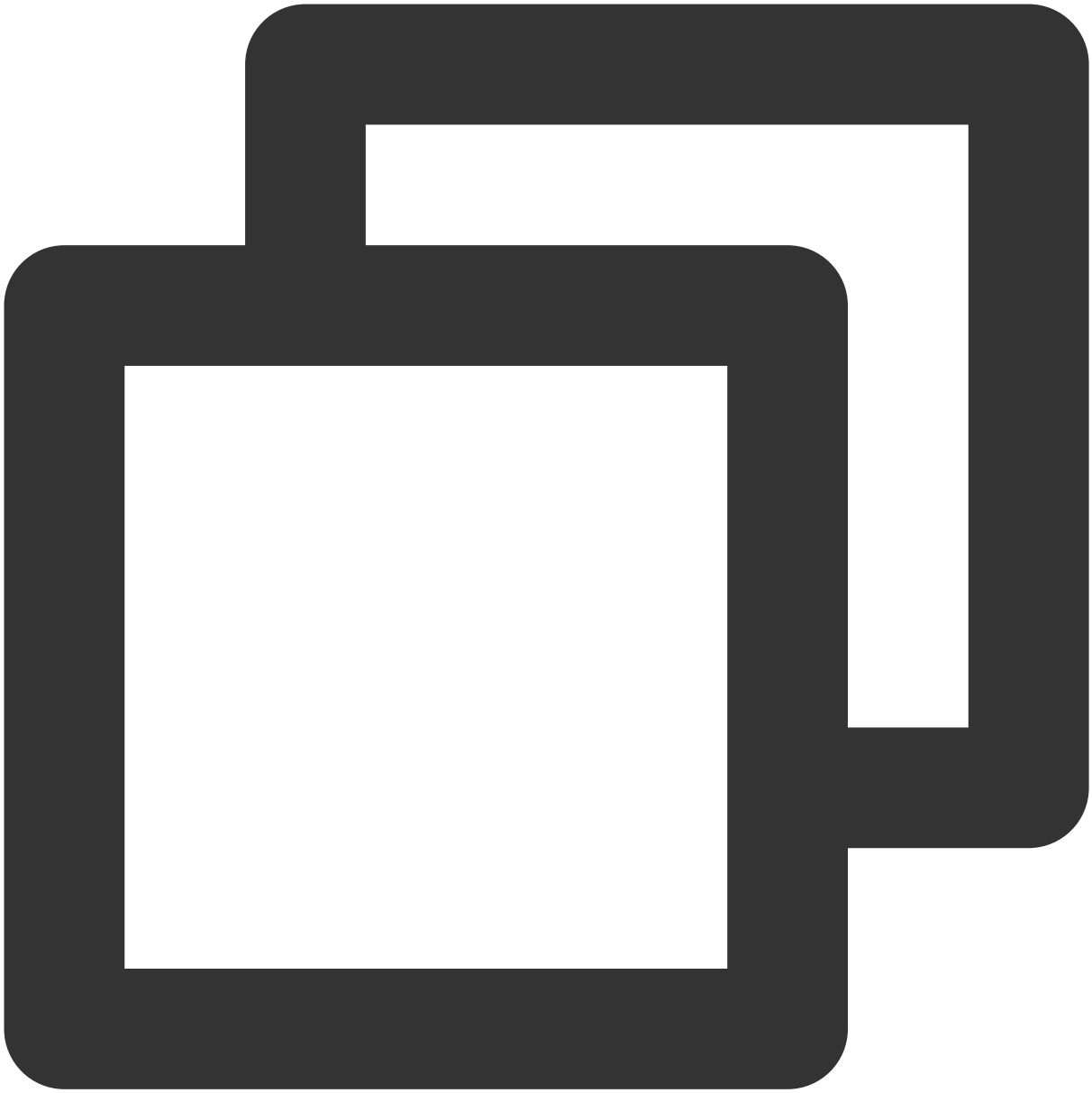
UserSig：使用开通服务获取的 SDKSecretKey 对 SDKAppID、UserID 等信息进行加密，就可以得到 UserSig，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以通过控制台左侧项目栏中的 UserSig 工具，创建一个临时可用的 UserSig。



更多信息请参见 UserSig 相关。

步骤3：设置用户昵称和头像

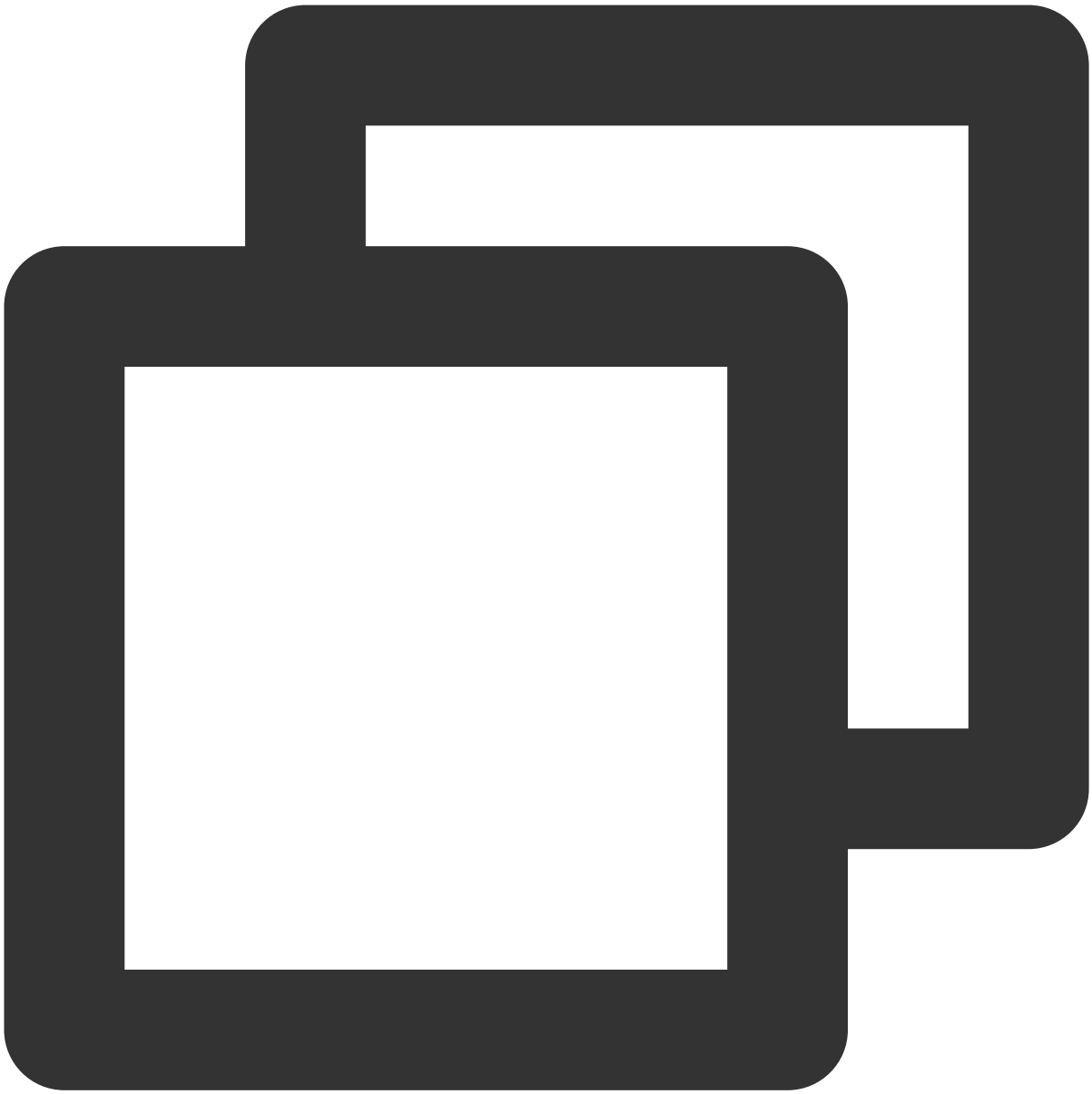
调用 `TUIRoomEngine` 的 `setSelfInfo` 即可设置用户的用户名、头像，可以在登录成功的回调中设置。



```
import TUIRoomEngine

TUIRoomEngine.setSelfInfo(userName: "xxx", avatarUrl: "xxx") {
    print("setSelfInfo success")
} onError: { code, message in
    print("setSelfInfo failed, code:\\(code),message:\\(message)")
}
```

步骤4：主持人发起快速会议

会议主界面为ConferenceMainViewController，只需调用 quickStartConference 发起快速会议，并在onConferenceStarted回调中跳转到会议主界面，即可发起快速会议。

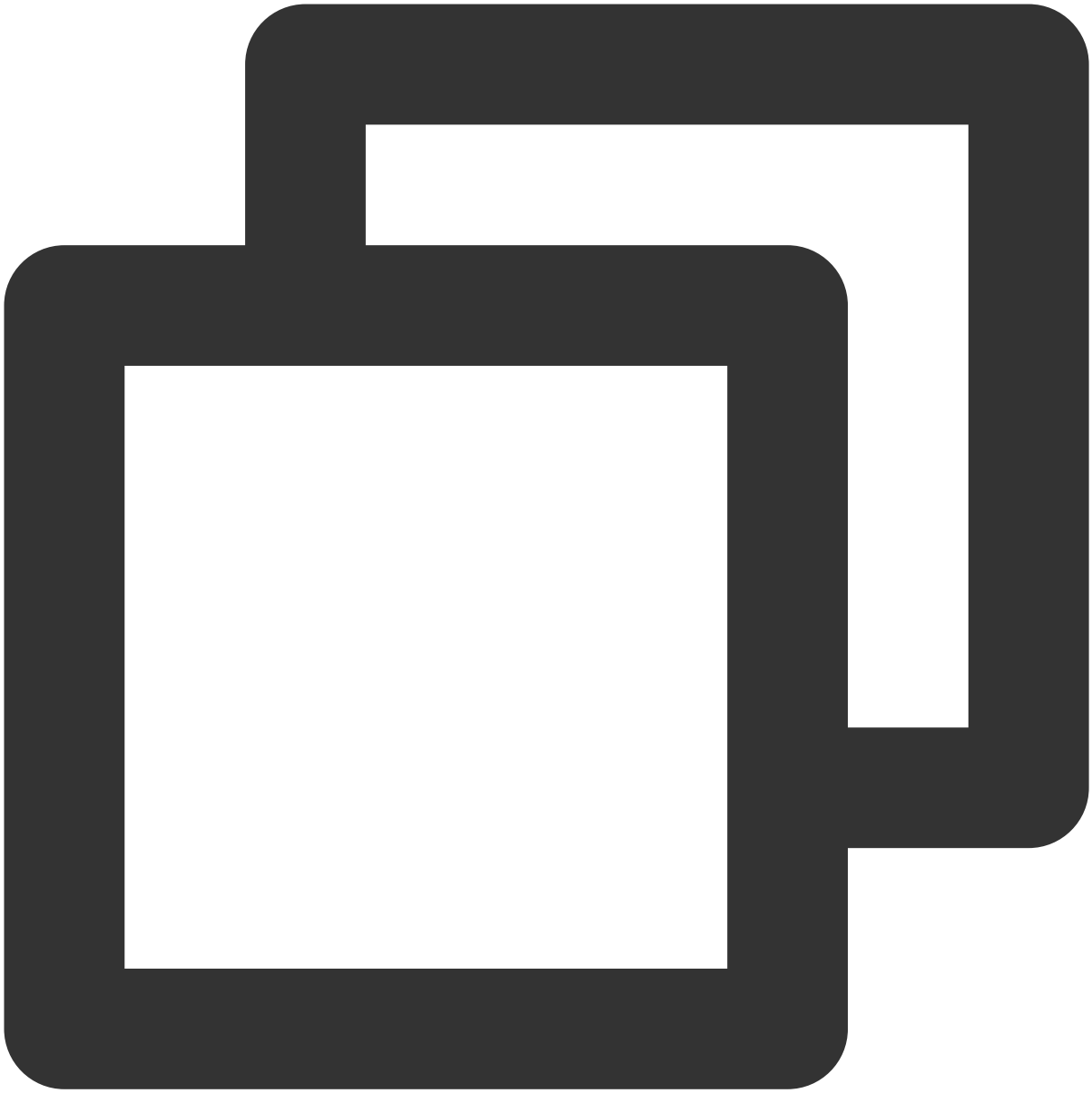


```
import TUIRoomKit

// CreateRoomViewController 为您自己的 ViewController
class CreateConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    func quickStartConferenceAction() {
        conferenceViewController = ConferenceMainViewController()
        conferenceViewController?.setConferenceObserver(observer: self)
```

```
            conferenceViewController?.quickStartConference(conferenceId: "123456")
        }
    }

extension CreateConferenceViewController: ConferenceObserver {
    func onConferenceStarted(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```

步骤5：普通成员加入会议

会议主界面为ConferenceMainViewController，只需调用 joinConference 加入会议，并在onConferenceJoined回调中跳转到会议主界面，即可参与当前会议。



```
import TUIRoomKit

// EnterRoomViewController 为您自己的 ViewController
class EnterConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    private func joinConferenceAction() {
        conferenceViewController = ConferenceMainViewController()
        conferenceViewController?.setConferenceObserver(observer: self)
        conferenceViewController?.joinConference(conferenceId: "123456")
    }
 }
```

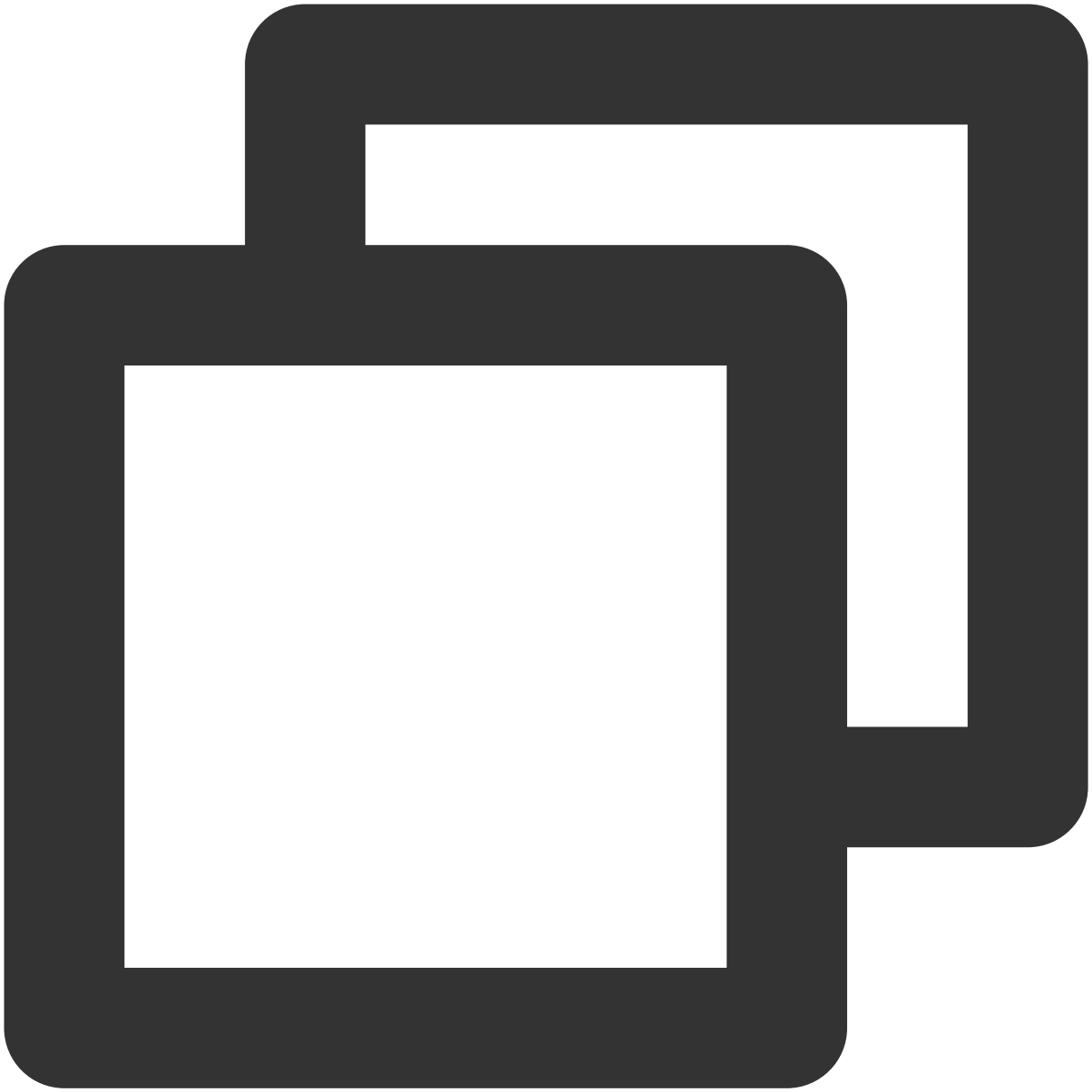
```
extension EnterConferenceViewController: ConferenceObserver {
    func onConferenceJoined(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```

更多特性

接入聊天挂件

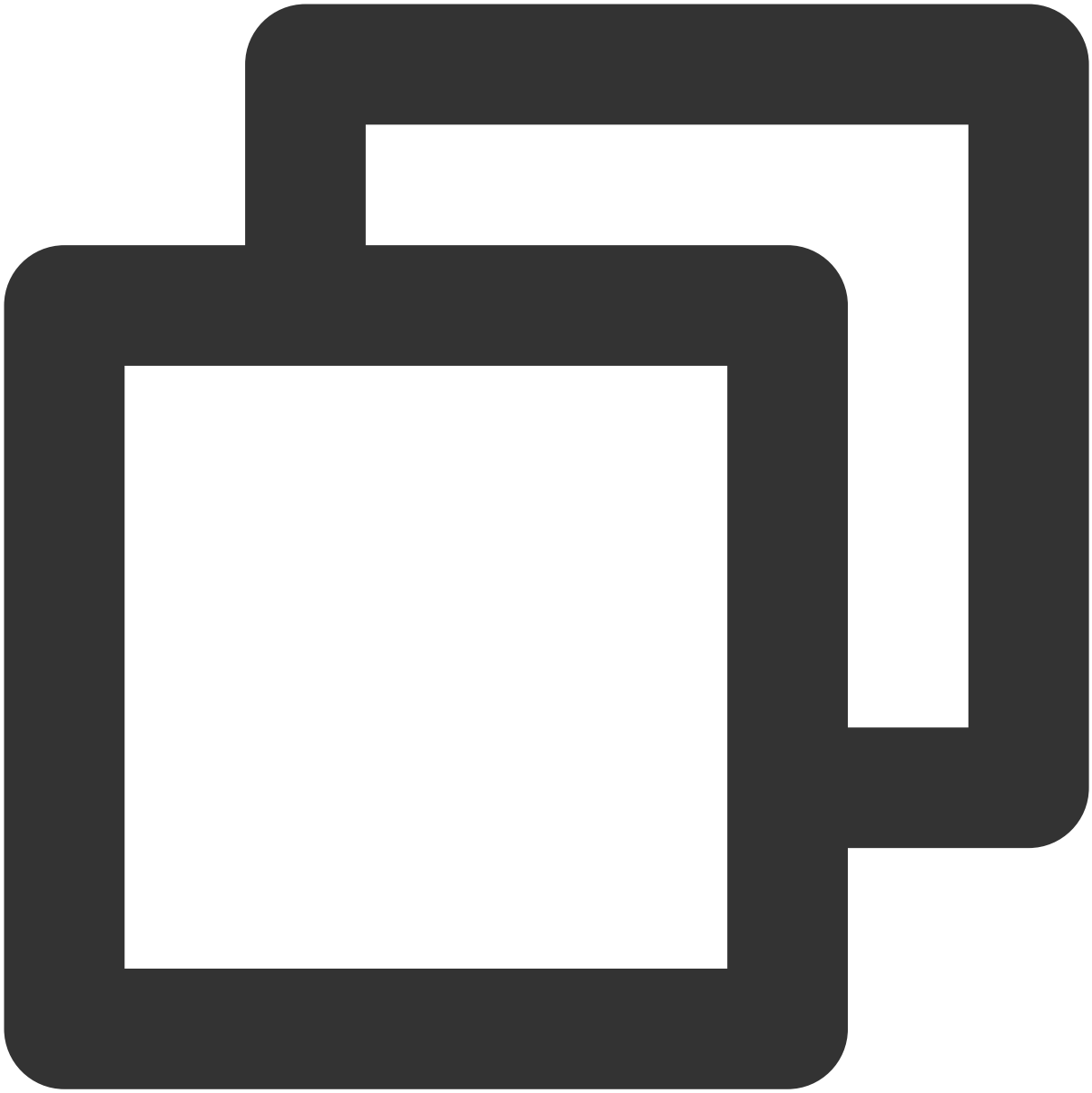
使用 CocoaPods 导入聊天挂件，具体步骤如下：

1. 在您的 `Podfile` 文件中添加以下依赖。



```
pod 'TUIChat'           # [可选] 聊天挂件
```

2.执行以下命令，安装组件。



```
pod install
```

常见问题

如果在使用TUIRoomKit时遇见了问题，可以先行查看 常见问题 文档。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2024-04-12 11:29:03

本文将介绍如何用最短的时间完成 `TUIRoomKit` 组件的接入，跟随本文档，您将在一个小时的时间内完成如下几个关键步骤，并最终得到一个包含完备 UI 界面的音视频会议功能。

环境准备

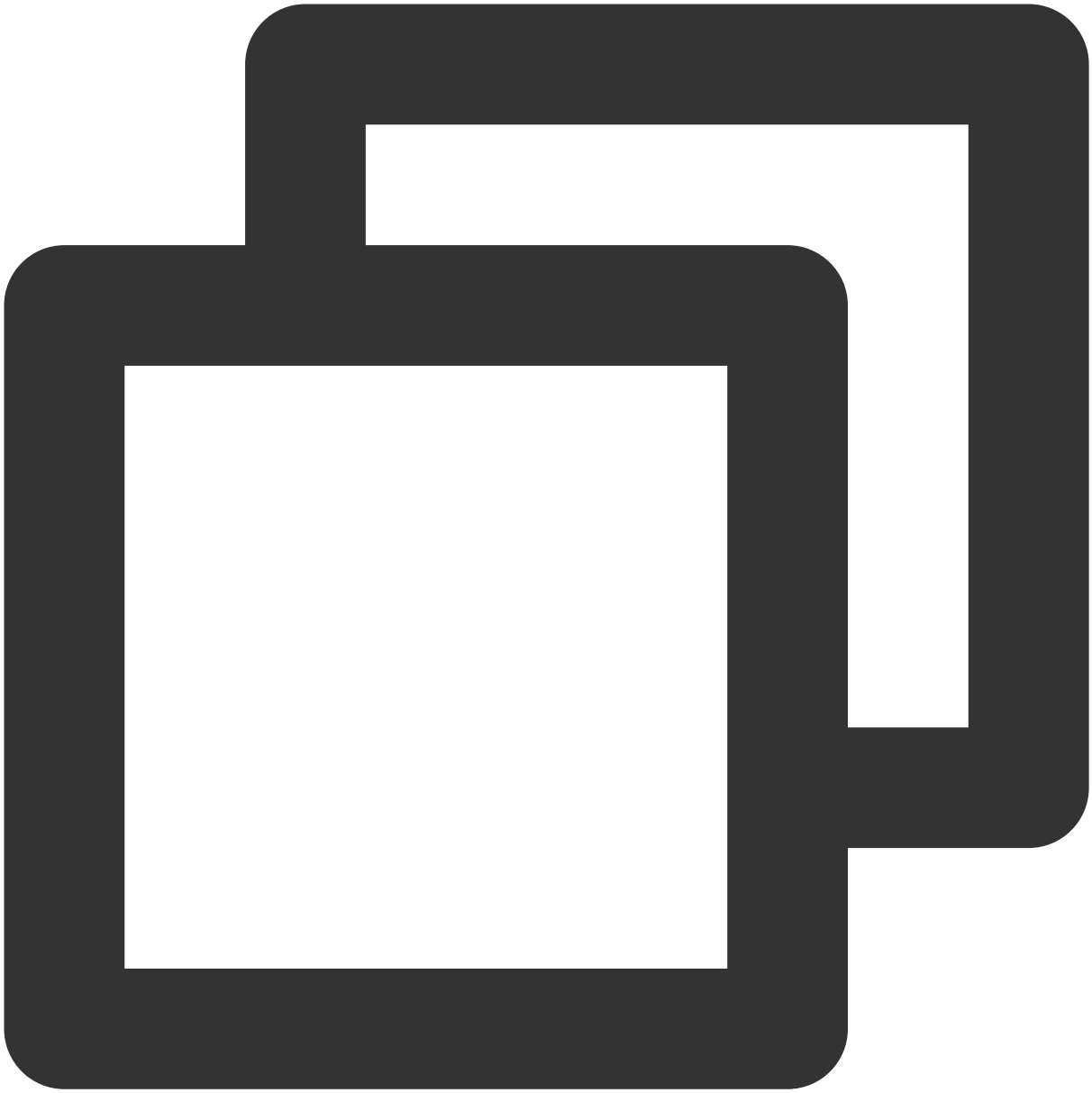
最低兼容 Android 4.4（SDK API Level 19），建议使用 Android 5.0 （SDK API Level 21）及以上版本。
Android Studio 3.5 及以上的版本（Gradle 3.5.4 及以上的版本）。
Android 4.4 及以上的手机设备。

开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

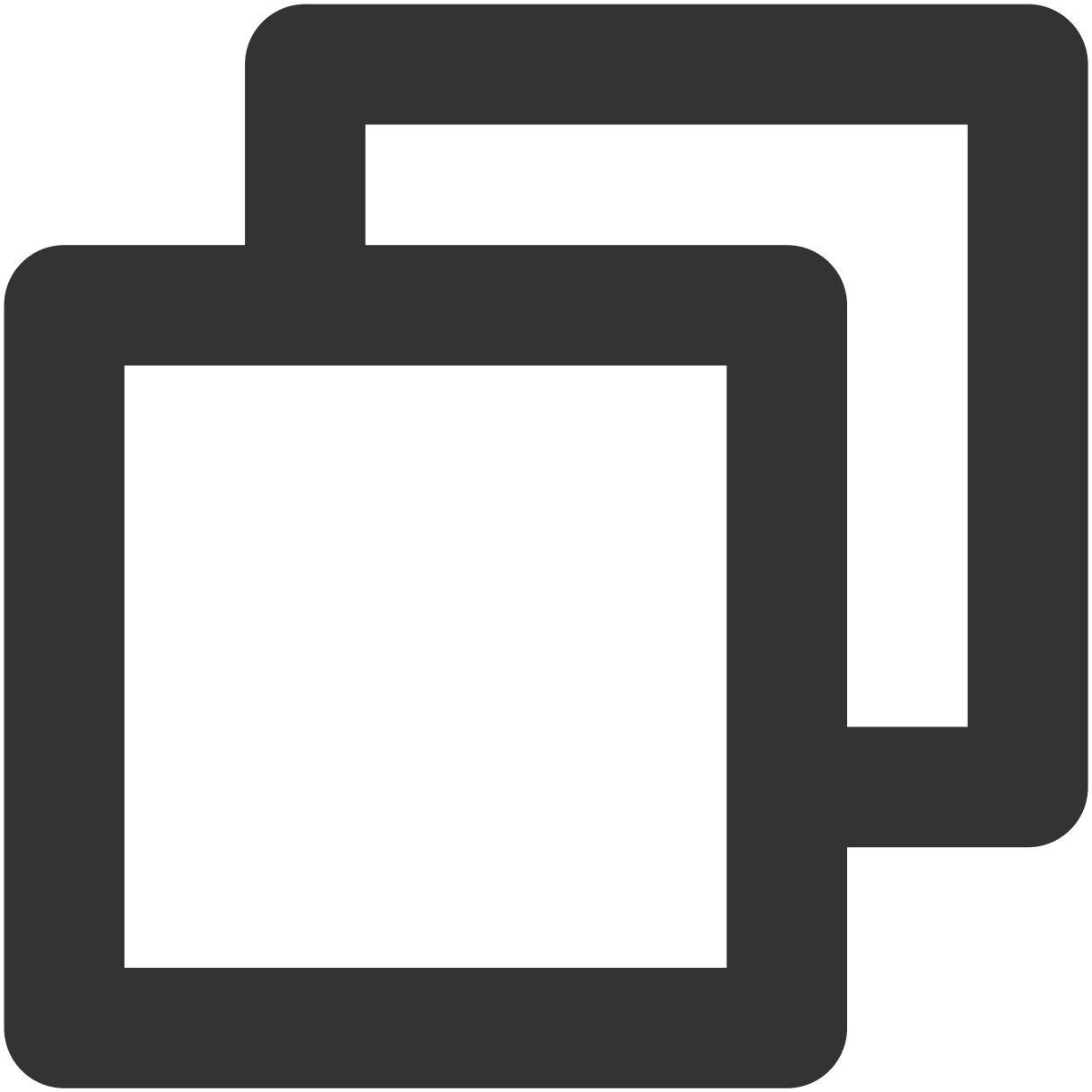
module 源码集成

1. 在 Github 中克隆/下载代码，然后拷贝 `Android` 目录下的 `debug` 、 `timcommon` 、 `tuichat` 、 `tuiroomkit` 子目录到您当前工程中的 app 同级目录中。
2. 工程根目录下找到 `setting.gradle` 文件，并在其中增加如下代码，它的作用是将 `tuiroomkit` 作为本地模块导入到您当前的项目中。



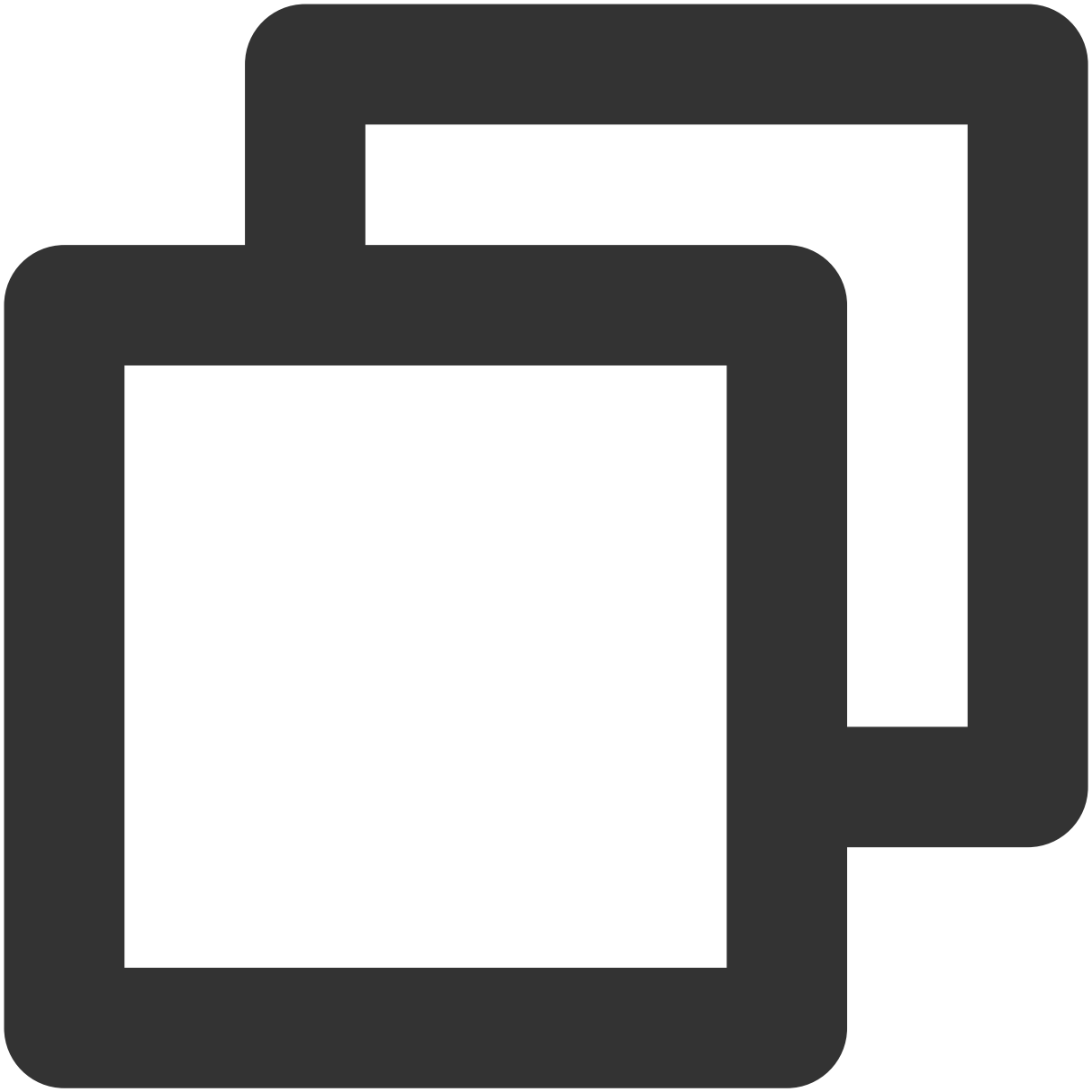
```
include ':debug'
include ':timcommon'
include ':tuichat'
include ':tuiroomkit'
```

3. 在 app 目录下找到 `build.gradle` 文件，并在其中增加如下代码，它的作用是声明当前 `app` 对新加入的 `tuiroomkit` 组件的依赖。



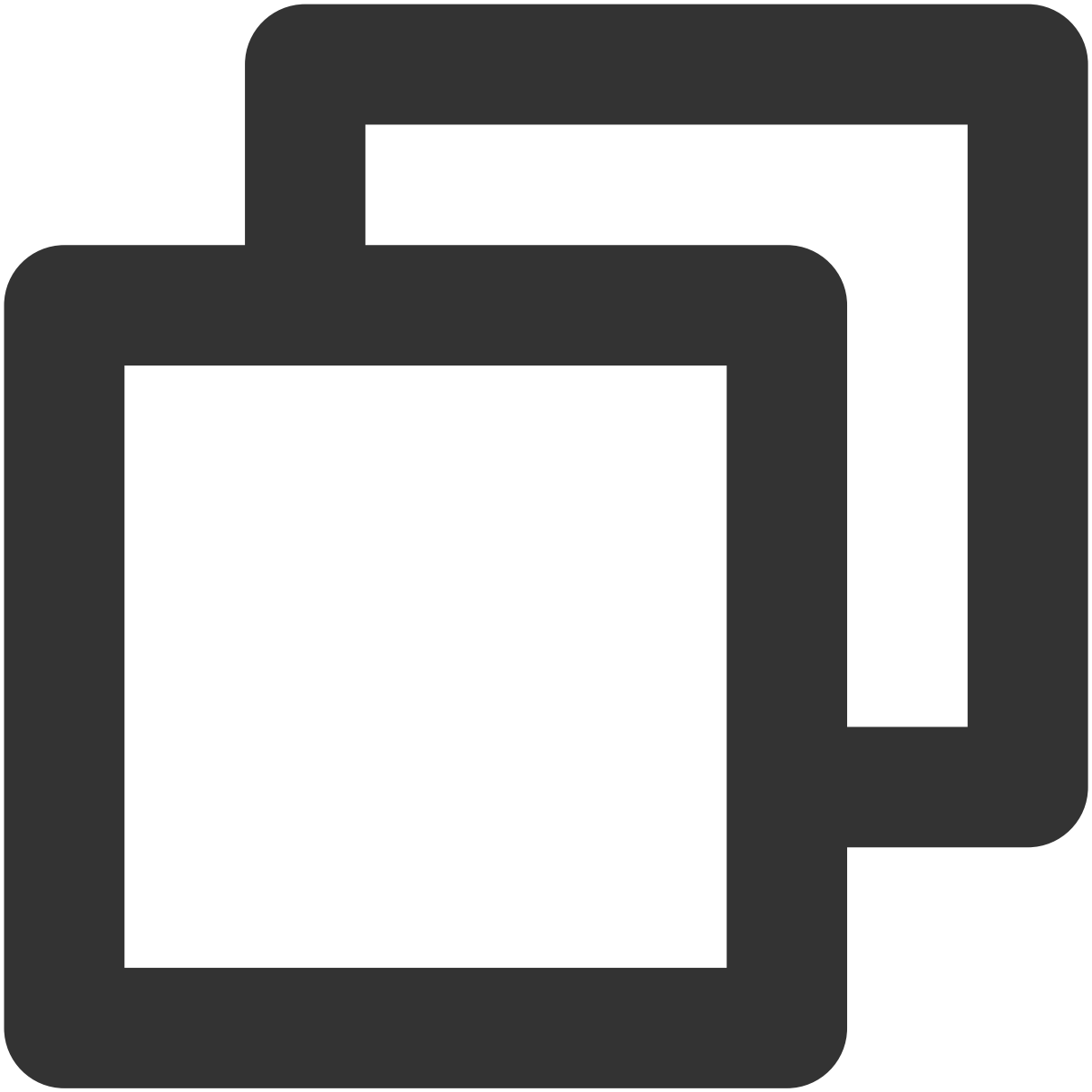
```
api project(':tuiroomkit')
```

4. 在 gradle.properties 文件中加入下行，表示自动转换三方库以兼容 AndroidX：



```
android.enableJetifier=true
```

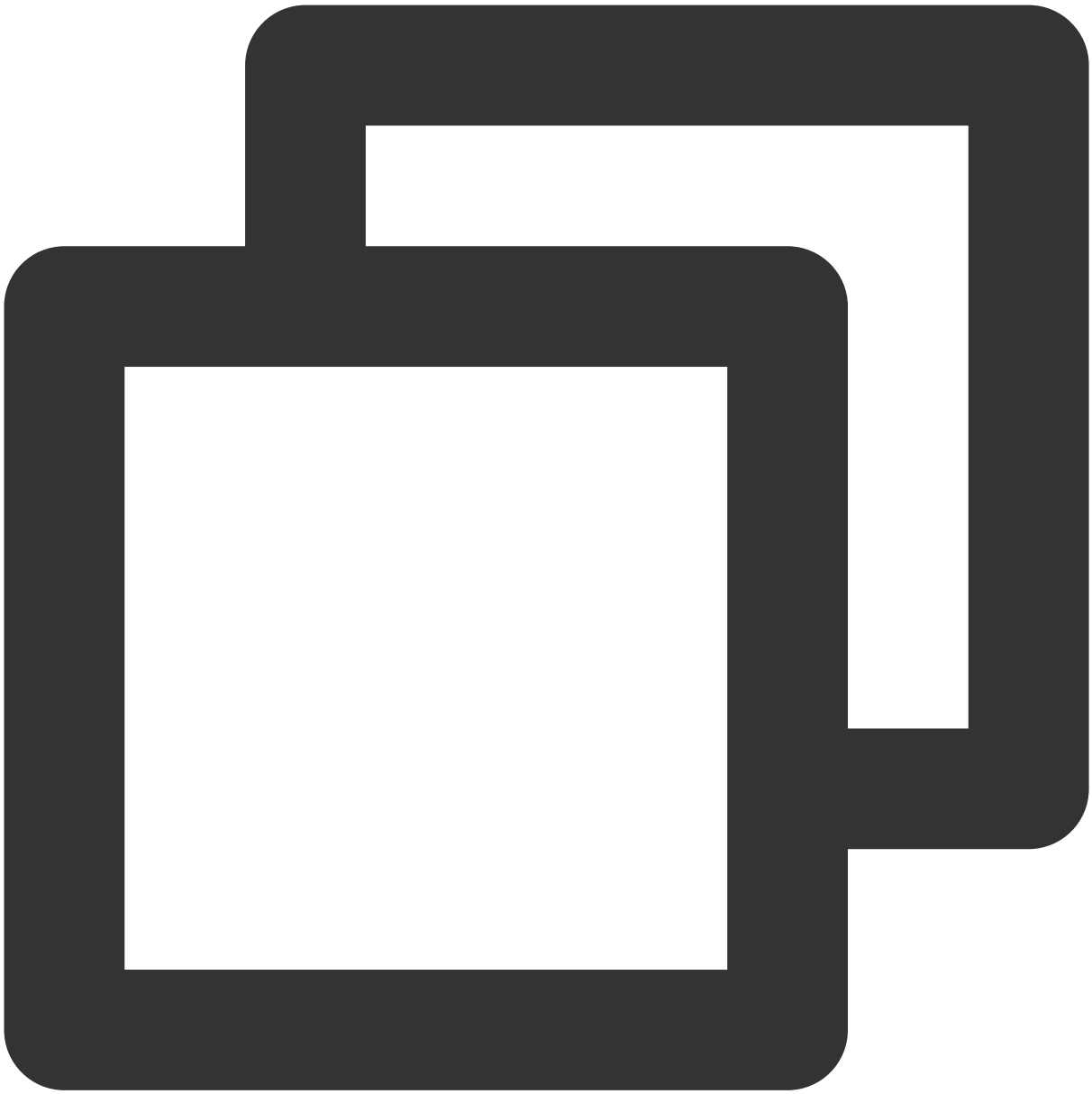
5. 添加 maven 仓库 和 Kotlin 支持，在 root 工程的 build.gradle 文件（与 settings.gradle 同级）中添加：



```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.0.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.31"
    }
}
```

版权所有：腾讯云计算（北京）有限责任公司

如果您使用 Gradle 8.x，则需要添加以下代码。



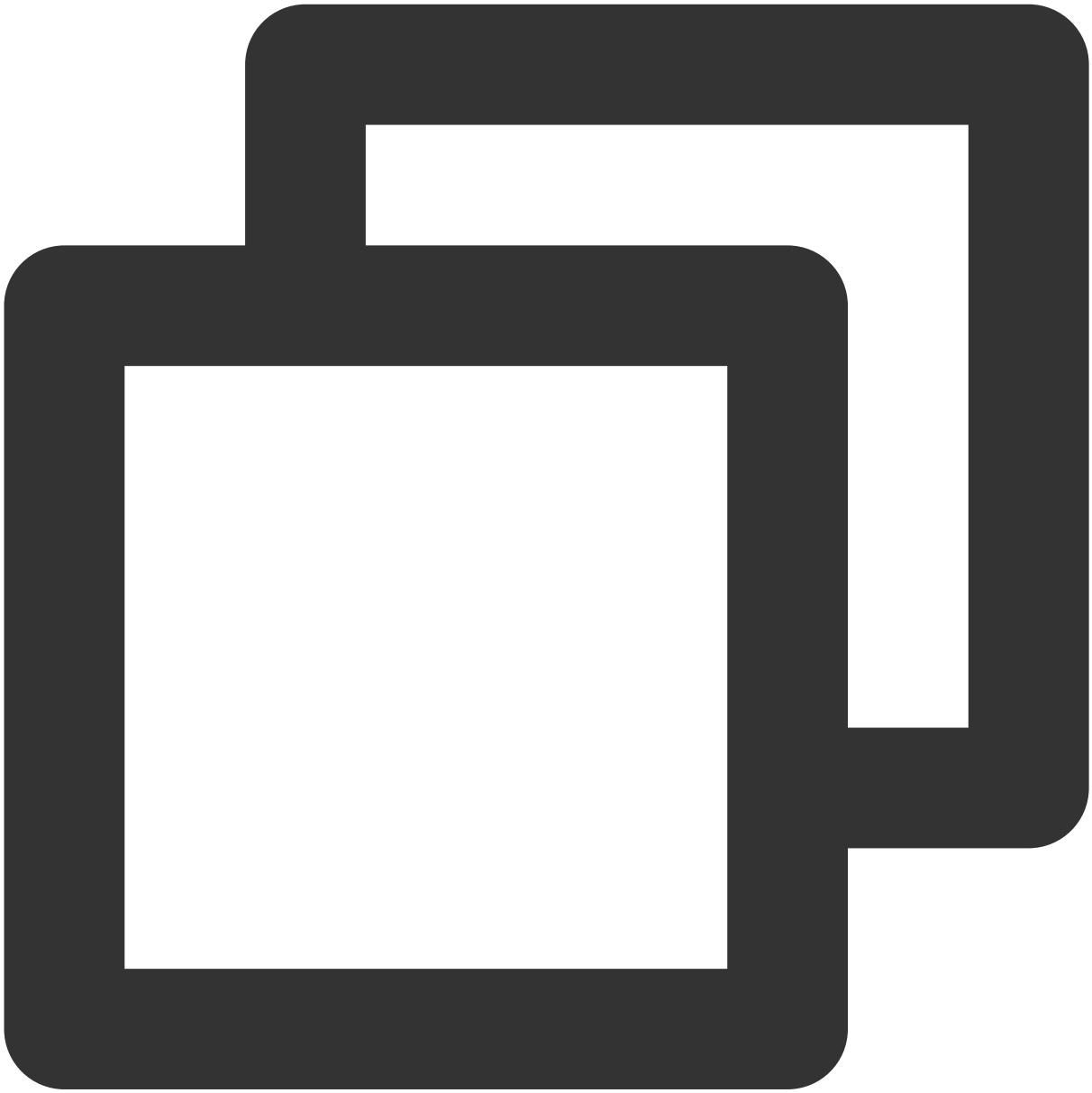
```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:8.0.2'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.0"
    }
```

```
}
```

说明：

Kotlin 、Gradle 和 AGP 的版本对应关系可以 在此查看。

6. 由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单，因此需要您在 proguard-rules.pro 文件中添加如下代码：



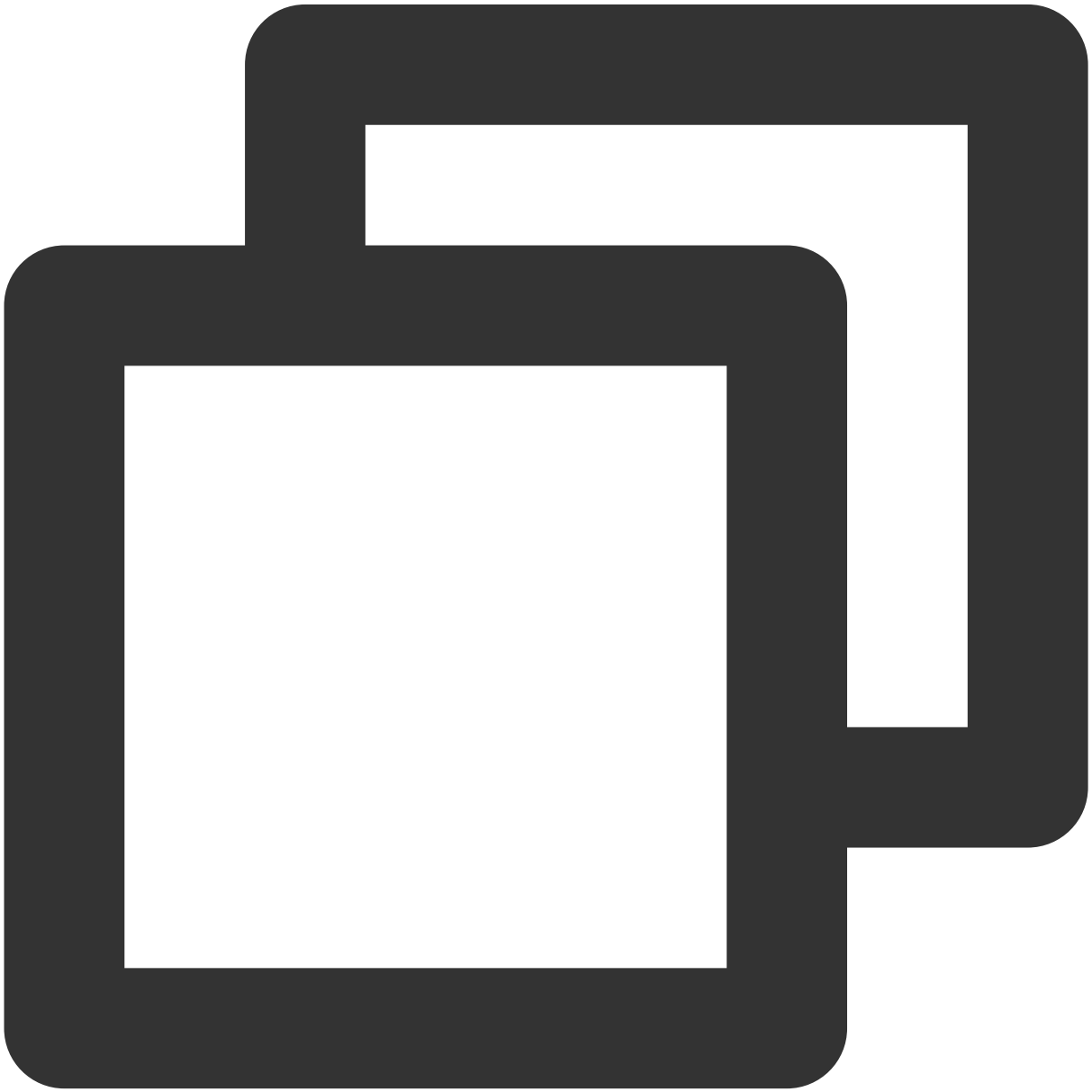
```
-keep class com.tencent.** { *; }
```

快速搭建

参考下面的步骤，您可在项目中快速搭建会议的主界面。

步骤1：组件登录

在您的项目中添加如下代码，它的作用是通过调用 `TUILogin` 中的相关接口完成组件的登录。这个步骤异常关键，因为只有在登录后才能正常使用 `TUIRoomKit` 的各项功能，故请您耐心检查相关参数是否配置正确：



```
TUILogin.login(context,
               1400000001,     // 请替换为开通服务取到的 SDKAppID
```

```
                "998",          // 请替换为您的 UserID
                "xxxxxxxxxx",   // 您可以在控制台中计算一个 UserSig 并填在这个位置
                callback);      // 登录是否成功的回调
```

参数说明

这里详细介绍一下 `login` 函数中所需要用到的几个关键参数：

SDKAppID：在 开通服务 中的最后一步中获取。

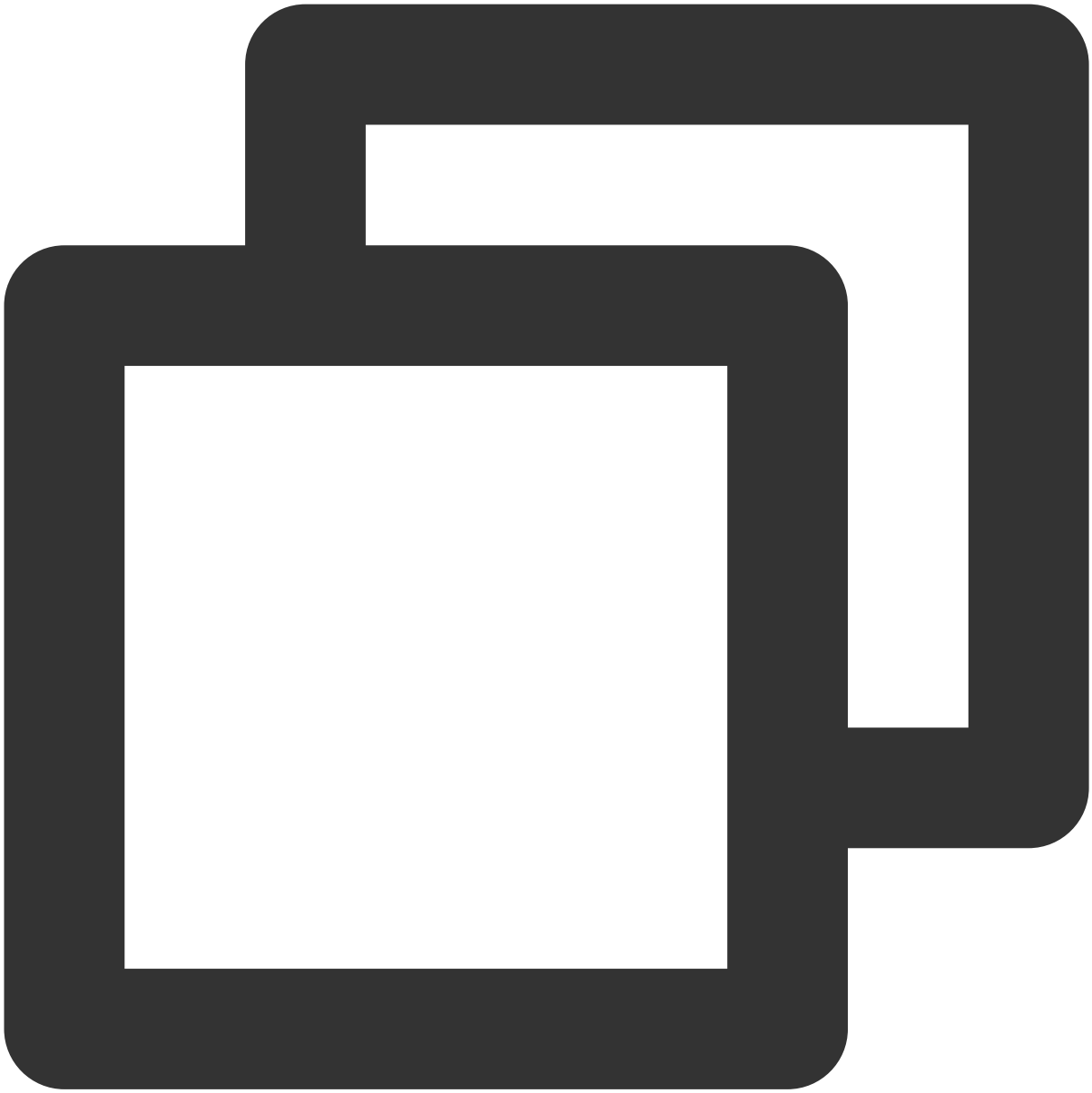
UserID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

UserSig：使用 开通服务 的第3步中获取的 `SDKSecretKey` 对 `SDKAppID` 、 `UserID` 等信息进行加密，就可以得到 `UserSig` ，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 `TRTC` 的服务。您可以通过控制台中的 **辅助工具** 生成一个临时可用的 `UserSig` 。

更多信息请参见 如何计算及使用 UserSig。

步骤2：设置用户昵称和头像

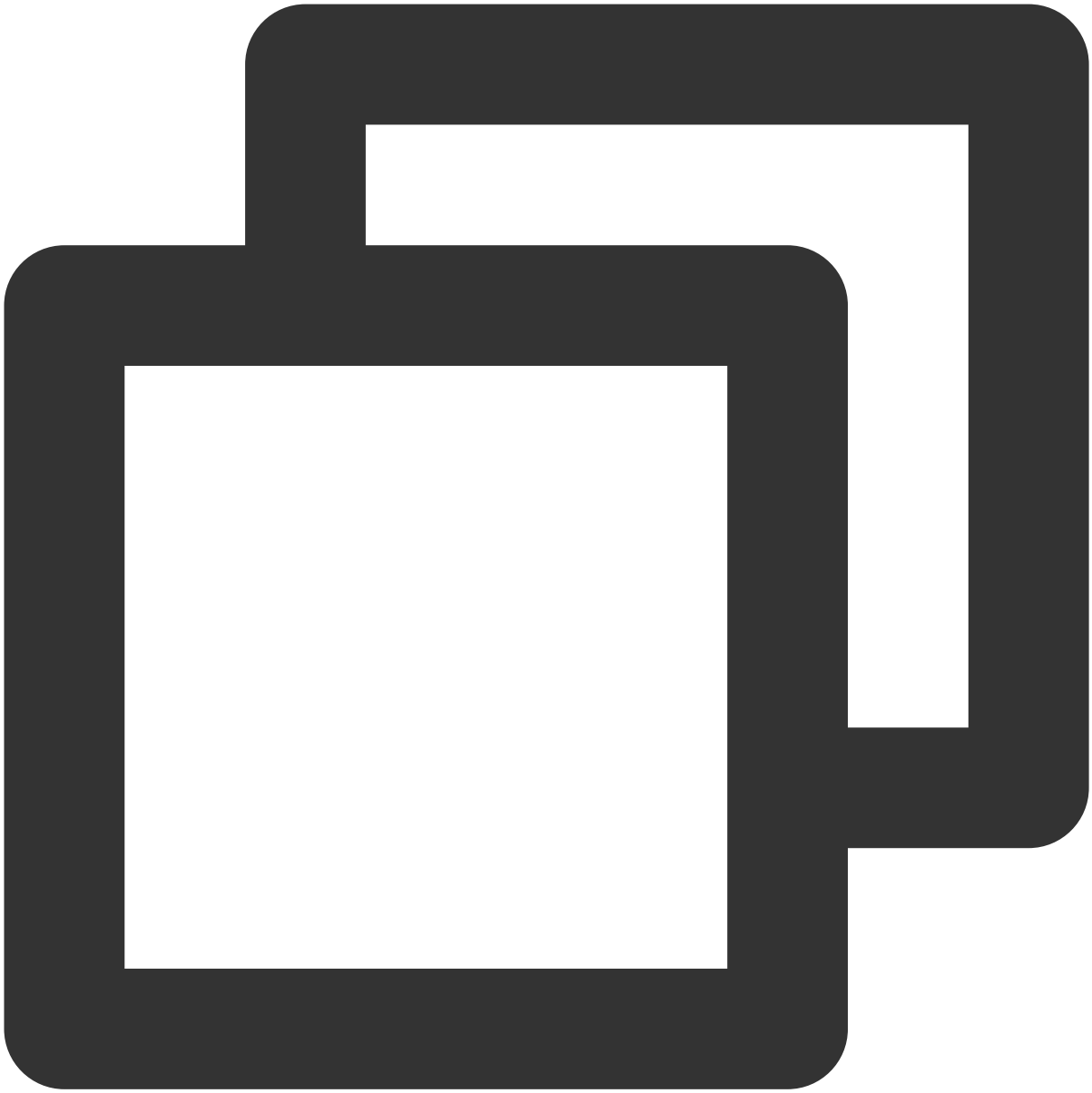
调用 `TUIRoomEngine` 的 `setSelfInfo` 即可设置用户的用户名、头像，可以在登录成功的回调中设置。



```
TUIRoomEngine.setSelfInfo("nickName", "faceUrl", null);
```

步骤3：主持人发起快速会议

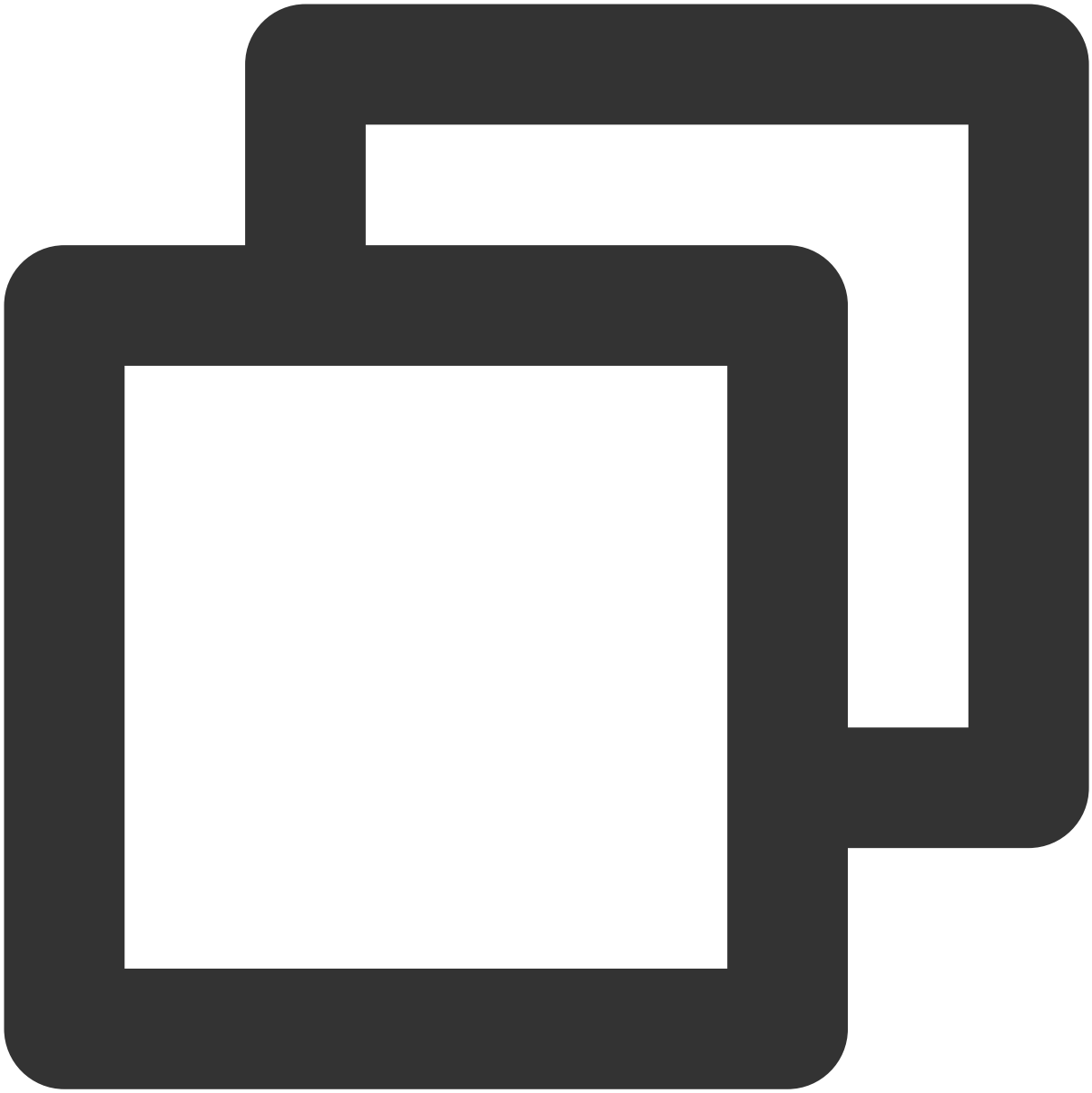
1. 在 activity_conference_owner.xml 中添加界面布局：



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/conference_owner_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. ConferenceMainFragment 是会议的主界面，只需调用 quickStartConference 发起快速会议，并在快速会议的回调 onConferenceStarted 中将 ConferenceMainFragment 添加到当前 Activity 中，即可发起快速会议。

版权所有：腾讯云计算（北京）有限责任公司



```
public class ConferenceOwnerActivity extends AppCompatActivity {
    private static final String TAG = "ConferenceOwnerActivity";

    private ConferenceObserver mConferenceObserver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_conference_owner);
```

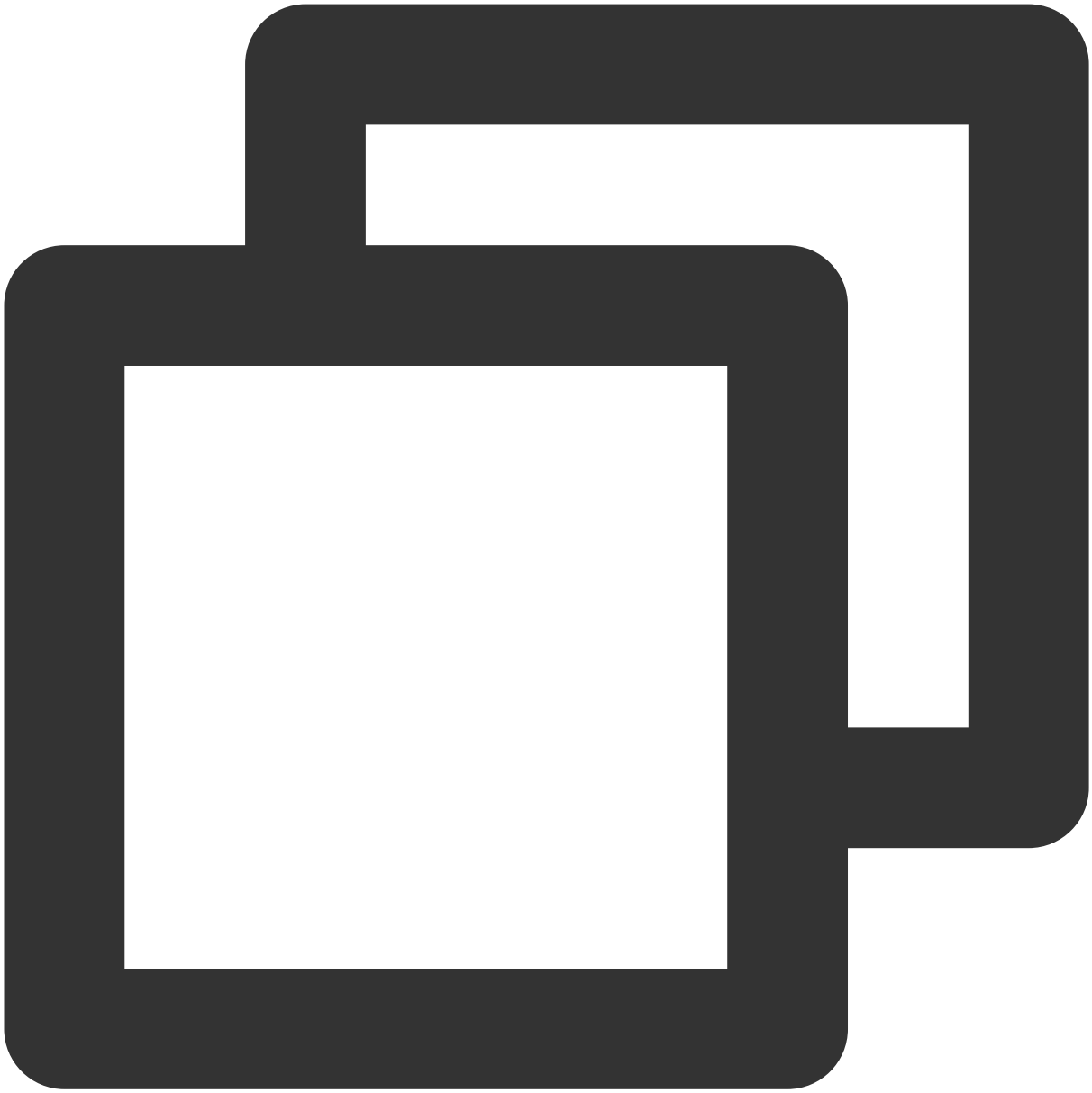
```
        ConferenceMainFragment fragment = new ConferenceMainFragment();
        setConferenceObserver(fragment);
        //"123456" 替换为对应的会议号
        fragment.quickStartConference("123456");
    }

    private void setConferenceObserver(ConferenceMainFragment fragment) {
        mConferenceObserver = new ConferenceObserver() {
            @Override
            public void onConferenceStarted(String conferenceId, ConferenceError er
                super.onConferenceStarted(conferenceId, error);
                if (error != ConferenceError.SUCCESS) {
                    Log.e(TAG, "Error : " + error);
                    return;
                }
                FragmentManager manager = getSupportFragmentManager();
                FragmentTransaction transaction = manager.beginTransaction();
                transaction.add(R.id.conference_owner_container, fragment);
                transaction.commitAllowingStateLoss();
            }
        };
        fragment.setConferenceObserver(mConferenceObserver);
    }
}
```

注意

ConferenceOwnerActivity 必须继承 FragmentActivity 或其子类，否则界面将无法显示。

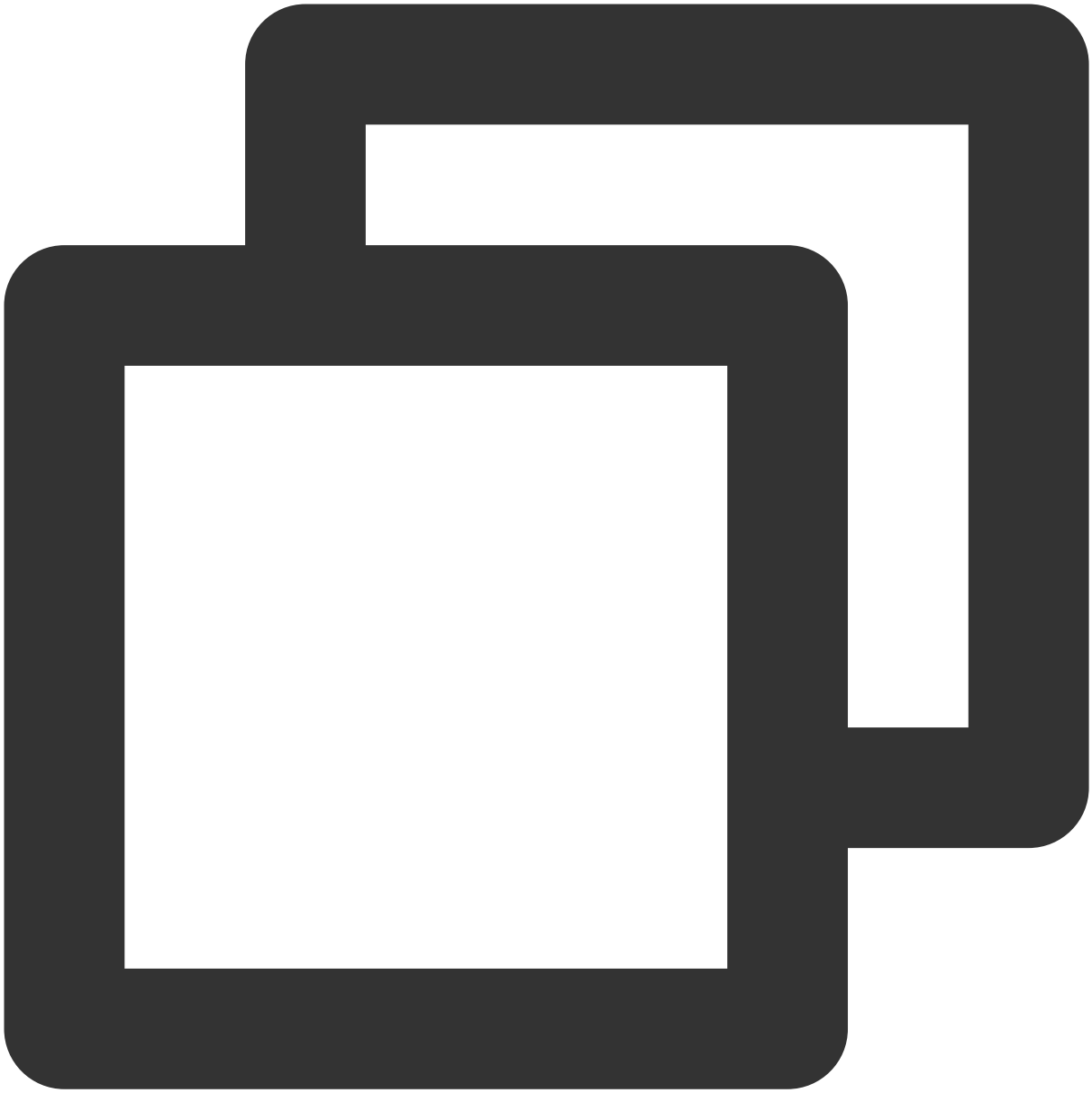
步骤4：普通成员加入会议

1. 在 activity_conference_general.xml 中添加界面布局：



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/conference_general_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. ConferenceMainFragment 是会议的主界面，只需调用 joinConference 加入会议，并在加入会议的回调 onConferenceJoined 中将 ConferenceMainFragment 添加到当前 Activity 中，即可参与当前会议。



```
public class ConferenceGeneralActivity extends AppCompatActivity {
    private static final String TAG = "ConferenceGeneralActivity";

    private ConferenceObserver mConferenceObserver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_conference_general);
```

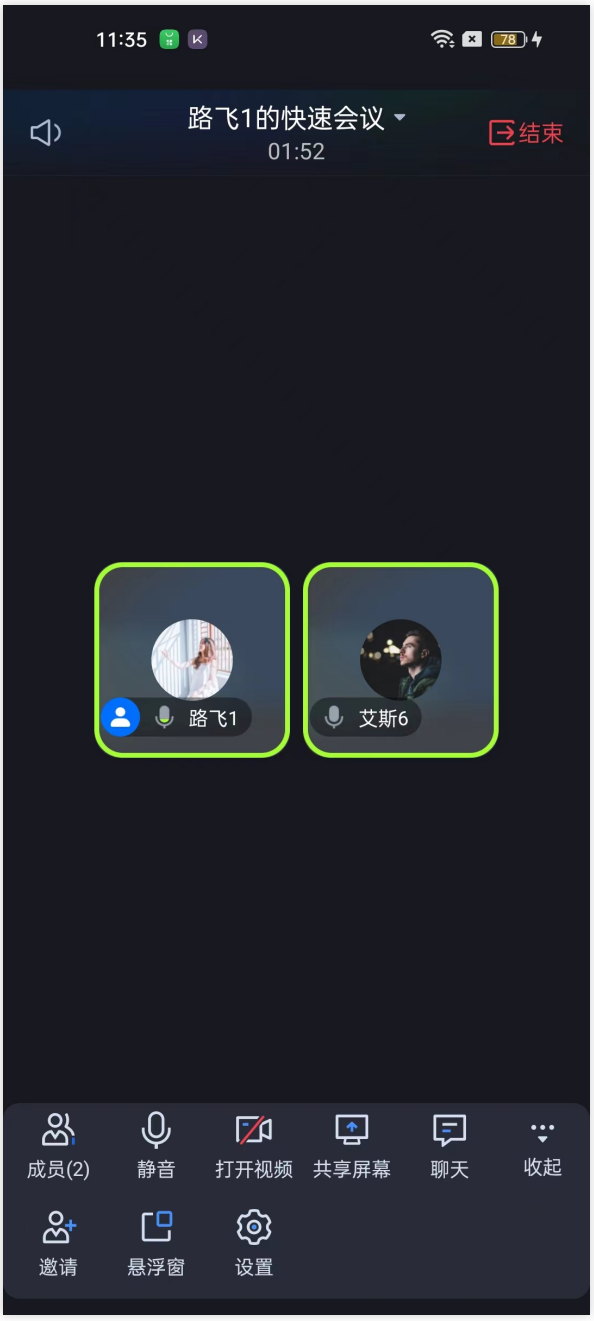
```
        ConferenceMainFragment fragment = new ConferenceMainFragment();
        setConferenceObserver(fragment);
        //"123456" 替换为对应的会议号
        fragment.joinConference("123456");
    }

    private void setConferenceObserver(ConferenceMainFragment fragment) {
        mConferenceObserver = new ConferenceObserver() {
            @Override
            public void onConferenceJoined(String conferenceId, ConferenceError err
                super.onConferenceJoined(conferenceId, error);
                if (error != ConferenceError.SUCCESS) {
                    Log.e(TAG, "Error : " + error);
                    return;
                }
                FragmentManager manager = getSupportFragmentManager();
                FragmentTransaction transaction = manager.beginTransaction();
                transaction.add(R.id.conference_general_container, fragment);
                transaction.commitAllowingStateLoss();
            }
        };
        fragment.setConferenceObserver(mConferenceObserver);
    }
}
```

注意

ConferenceGeneralActivity 必须继承 FragmentActivity 或其子类，否则界面将无法显示。

步骤5：运行

运行代码后，您将看到如下的会议界面。


11:35
78
路飞1的快速会议
01:52
结束
路飞1
艾斯6
成员(2)
静音
打开视频
共享屏幕
聊天
收起
邀请
悬浮窗
设置



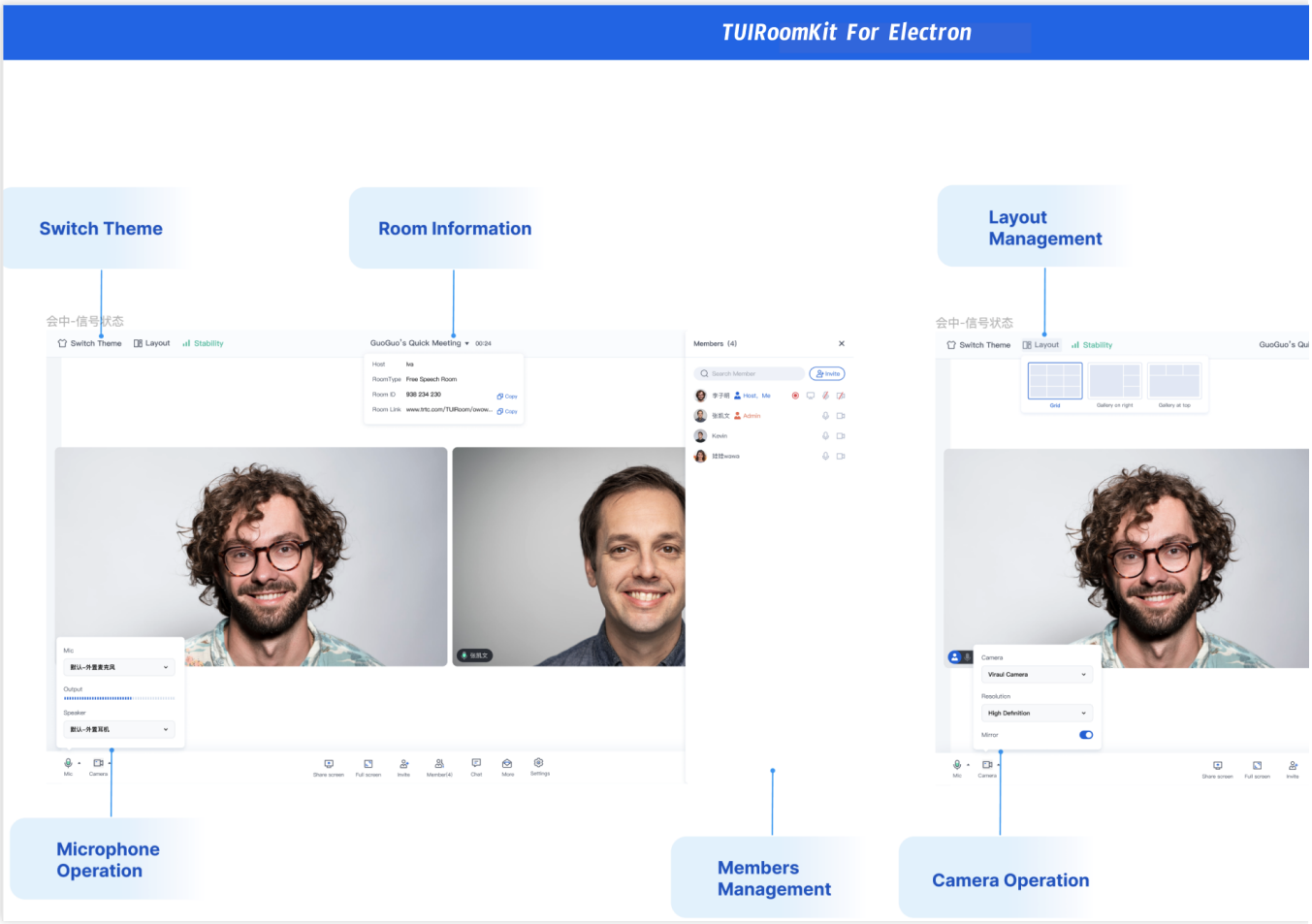
交流与反馈

您在接入或使用过程有任何疑问或者建议，欢迎联系：info_rtc@tencent.com 。

Electron

最近更新时间：2024-04-12 11:29:03

TUIRoomKit 是腾讯云推出的多人音视频 UI 组件。组件提供房间管理，音视频控制，屏幕共享，成员管理，麦位管理，即时聊天，自定义布局切换等丰富的功能交互，同时支持中英文切换，一键换肤等能力。



您可以单击在线体验链接： Mac OS版 及 Windows版 下载体验 TUIRoomKit Electron 更多功能。
您也可以单击 Github 下载 TUIRoomKit 代码，并参考 TUIRoomKit Electron 示例工程快速跑通 文档跑通 TUIRoomKit Electron 示例工程。
如需在现有业务中集成 Electron 端 TUIRoomKit 组件，请参考本文档。

支持的平台

Windows（PC）
Mac

版权所有：腾讯云计算（北京）有限责任公司

步骤一：开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

步骤二：准备 Vue 工程代码

Vue3 + Vite + TS

Vue2 + Webpack +TS

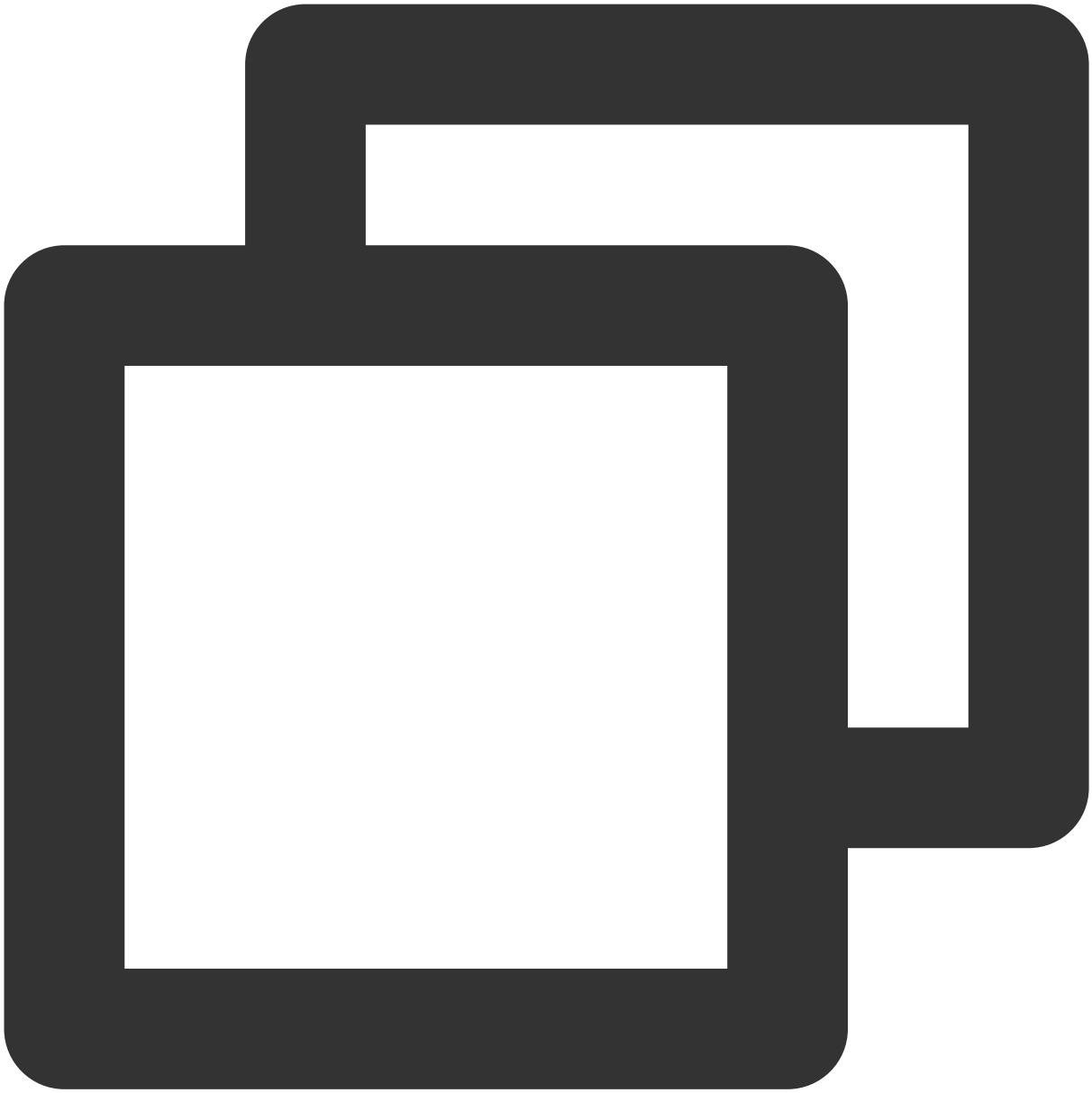
1. 打开业务侧已有 Electron + Vue3 + TS 项目，如果无 Electron + Vue3 + TS 项目,可通过此模板 Github 生成 Electron + Vue3 + TS 的模板工程, nodejs 版本大于 14.17.0。

注意：

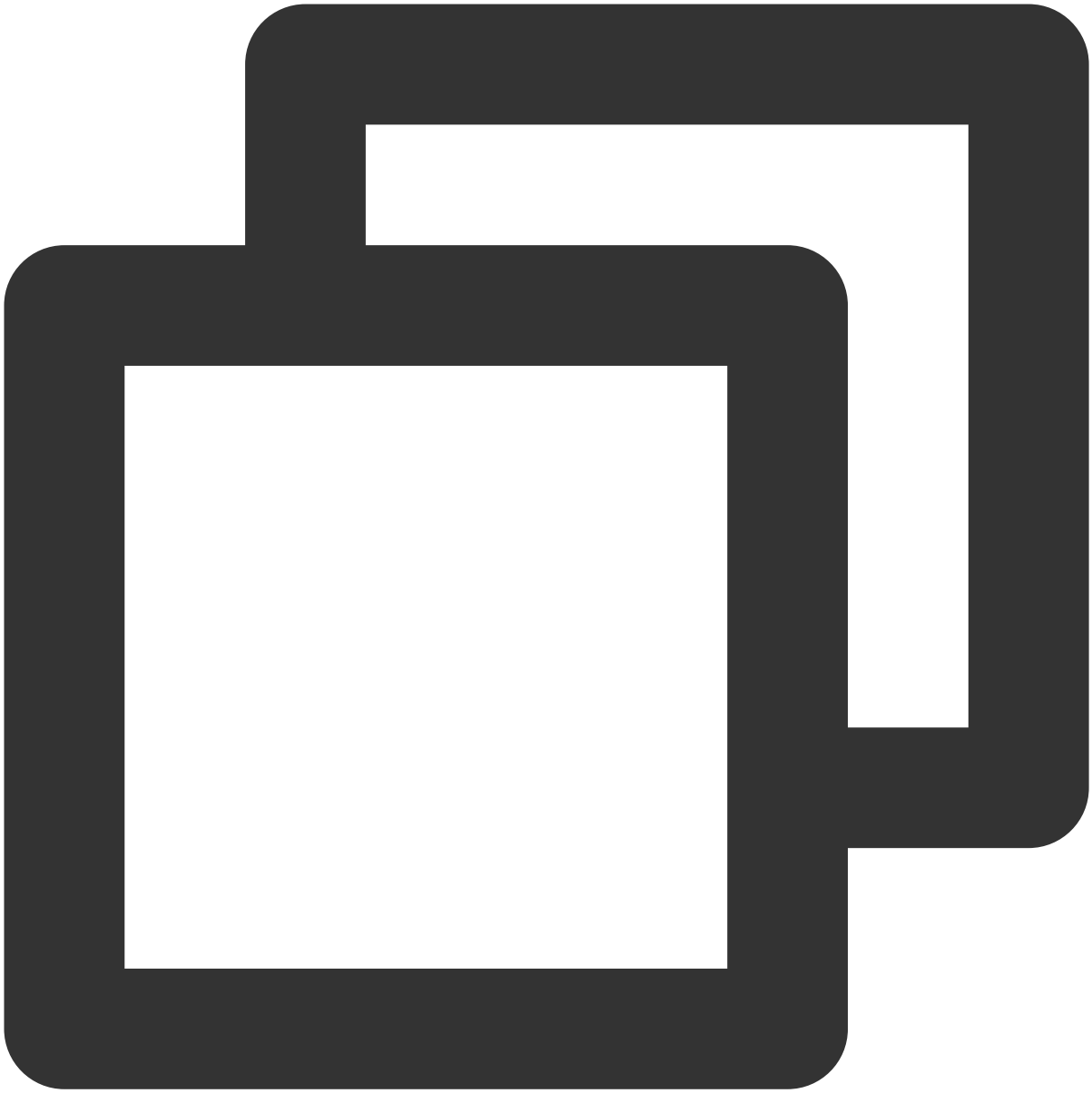
本文档介绍的集成步骤基于 electron-vite-vue 模版工程1.0.0版本。

electron-vite-vue 模版工程最新版本目录结构有调整，如需使用最新版本，可参照本文档自行调整目录和配置。

2. 克隆生成模板工程后，执行以下脚本：



```
git clone https://github.com/electron-vite/electron-vite-vue.git
cd electron-vite-vue
git checkout v1.0.0
git switch -c TUIRoomKit-quick-start
npm install
npm run dev
```



```
// 安装 vue-cli，注意 Vue CLI 4.x 要求 Node.js 为 v10 以上版本
npm install -g @vue/cli
// 创建 Vue2 + Webpack + TS 模版工程
vue create tuiroomkit-demo
```

注意：

执行生成模板工程脚本的过程中，生成模版的方式选择 Manually select features，其余配置选项参考图片。

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected
polyfills, transpiling JSX)? Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated
config files
```

成功生成 Vue2 + Webpack + TS 模版工程后， 参考 Electron 无UI集成文档，往模版工程中集成 Electron。

注意：

由于 TUIRoom 组件内已经引入了 trtc-electron-sdk ， 在参考 Electron 无UI集成文档 时，可以忽略：

2. 在项目脚本里引入模块并使用：

```
const TRTCCloud = require('trtc-electron-sdk').default;
// import TRTCCloud from 'trtc-electron-sdk';
this.rtcCloud = new TRTCCloud();
// 获取 SDK 版本号
this.rtcCloud.getSDKVersion();
```

从v7.9.348起，TRTC Electron SDK 增加了 trtc.d.ts 文件，方便使用 TypeScript 的开发者：

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud: TRTCCloud = new TRTCCloud();
// 获取 SDK 版本号
rtcCloud.getSDKVersion();
```

这一步骤。

步骤三：下载并引用 TUIRoomkit 组件

1. **下载 TUIRoom 组件代码**

单击 Github , 克隆或下载 TUIRoomKit 仓库代码。

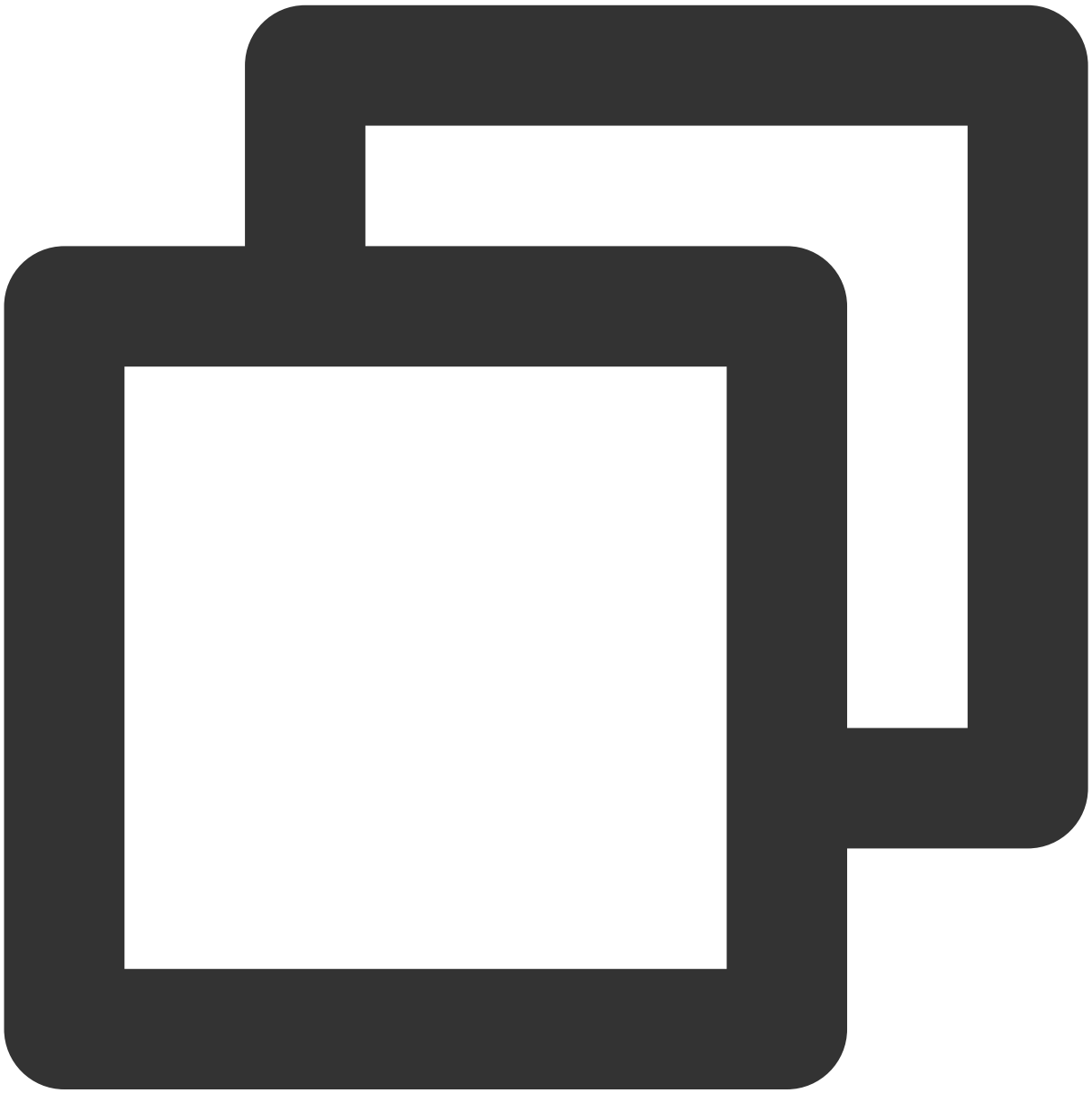
2. **引用 TUIRoom 组件**

Vue3 项目中引入 TUIRoom 组件

Vue2 项目中引入 TUIRoom 组件

1. 复制 `TUIRoomKit/Electron/vue3/packages/renderer/src/TUIRoom` 文件夹到已有项目 `packages/renderer/src/` 目录下, 复制 `TUIRoomKit/Electron/vue3/packages/renderer/index.html` 文件夹到已有项目 `packages/renderer/` 目录下。
2. 在页面中引用 TUIRoom 组件。例如：在 `App.vue` 组件中引入 TUIRoom 组件。

TUIRoom 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。主持人及普通成员可通过 init 方法向 TUIRoom 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// 引入 TUIRoom 组件，注意确认引入路径是否正确
import Room from './TUIRoom/index.vue';
// 获取 TUIRoom 组件元素，用于调用 TUIRoom 组件的方法
const TUIRoomRef = ref();

onMounted(async () => {
  // 初始化 TUIRoom 组件
  // 主持人在创建房间前需要先初始化 TUIRoom 组件
  // 普通成员在进入房间前需要先初始化 TUIRoom 组件
  await TUIRoomRef.value.init({
    // 获取 sdkAppId 请您参考 步骤一
    sdkAppId: 0,
    // 用户在您业务中的唯一标示 Id
    userId: '',
    // 本地开发调试可在 https://console.trtc.io/usersig 页面快速生成 userSig，注意 userSi
    userSig: '',
    // 用户在您业务中使用的昵称
    userName: '',
    // 用户在您业务中使用的头像链接
    avatarUrl: '',
    // 用户在您业务中需要的皮肤主题颜色及是否支持切换皮肤主题
    theme: {
      defaultTheme:'black',
      isSupportSwitchTheme: true
     }
  })
  // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
  await handleCreateRoom();
})

// 主持人创建房间，该方法只在创建房间时调用
async function handleCreateRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模式) 两种
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
```

```
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomPar
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
async function handleEnterRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
  const roomId = '123456';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.enterRoom({ roomId, roomParam });
  } catch (error: any) {
    alert('TUIRoomKit.enterRoom error: ' + error.message);
  }
}
</script>

<style lang="scss">
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

1. 复制 `TUIRoomKit/Electron/vue2/src/TUIRoom` 文件夹到已有项目 src/ 目录下。

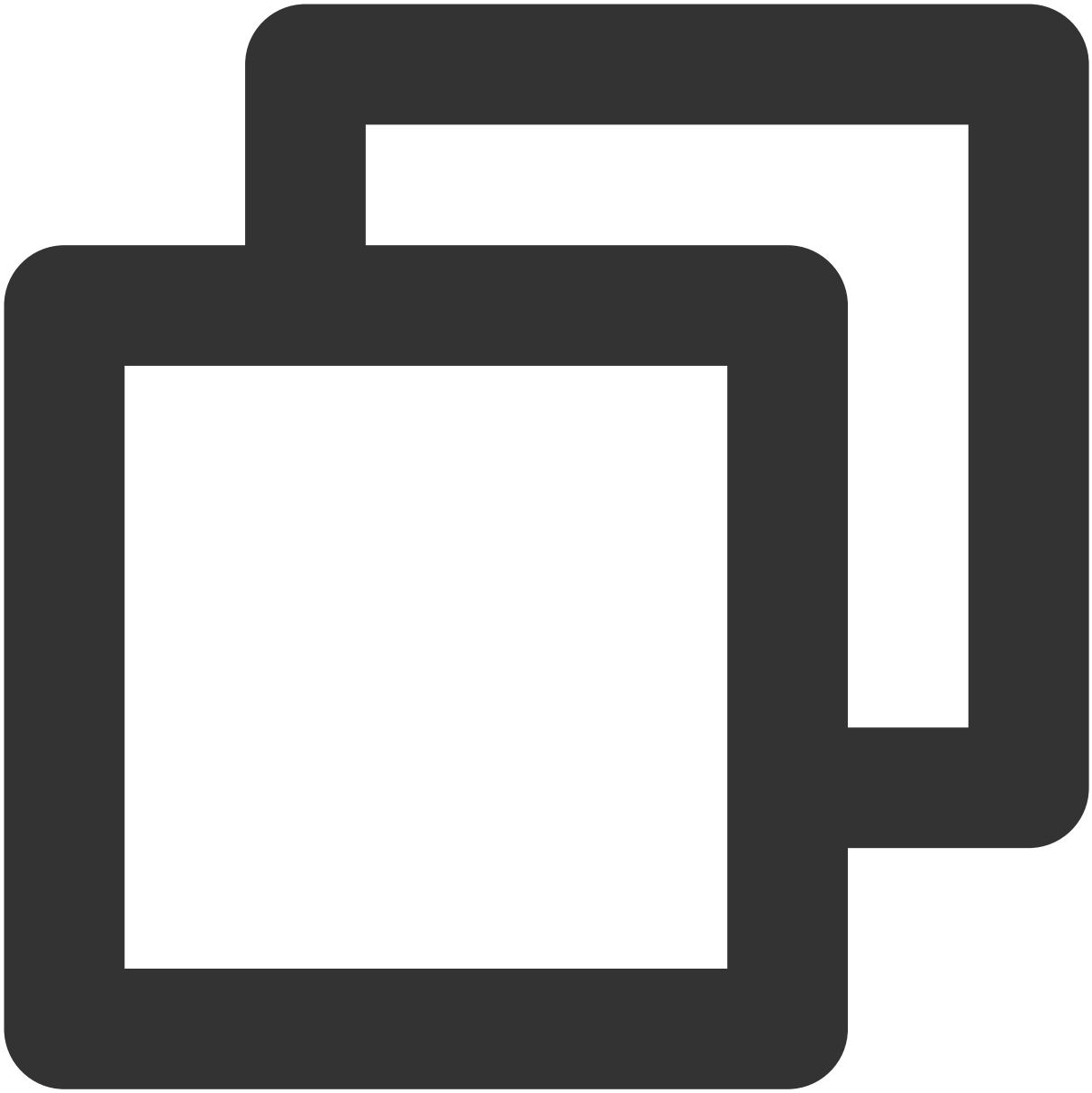
2. 在页面中引用 TUIRoom 组件。例如：在 `App.vue` 组件中引入 TUIRoom 组件。

TUIRoom 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。

主持人及普通成员可通过 init 方法向 TUIRoom 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。

注意：

在页面中复制以下代码之后，需要修改 TUIRoom 接口的参数为实际数据。



```
<template>
  <div id="app">
    <room-container ref="TUIRoomRef"></room-container>
  </div>
</template>

<script>
import RoomContainer from '@/TUIRoom/index.vue';
export default {
  name: 'App',
  components: { RoomContainer },
```

```
  data() {
    return {};
  },
  async mounted() {
    // 初始化 TUIRoom 组件
    // 主持人在创建房间前需要先初始化 TUIRoom 组件
    // 普通成员在进入房间前需要先初始化 TUIRoom 组件
    await this.$refs.TUIRoomRef.init({
      // 获取 sdkAppId 请您参考 步骤一
      sdkAppId: 0,
      // 用户在您业务中的唯一标示 Id
      userId: '',
      // 本地开发调试可在 https://console.trtc.io/usersig 页面快速生成 userSig，注意 user
      userSig: '',
      // 用户在您业务中使用的昵称
      userName: '',
      // 用户在您业务中使用的头像链接
      avatarUrl: '',
    });
    // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
    await this.handleCreateRoom();
  },
  methods: {
    // 主持人创建房间，该方法只在创建房间时调用
    async handleCreateRoom() {
      // roomId 为用户进入的房间号，要求 roomId 类型为 string
      // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模式)
      // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒
      const roomId = '123456';
      const roomMode = 'FreeToSpeak';
      const roomParam = {
          isOpenCamera: true,
          isOpenMicrophone: true,
      }
      try {
        await this.$refs.TUIRoomRef.createRoom({ roomId, roomName: roomId, roomMode
      } catch (error) {
        alert('TUIRoomKit.createRoom error: ' + error.message);
      }
    },
    // 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
    async handleEnterRoom() {
      // roomId 为用户进入的房间号，要求 roomId 类型为 string
      // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒
      const roomId = '123456';
      const roomParam = {
          isOpenCamera: true,
```

```
            isOpenMicrophone: true,
        }
        try {
          await this.$refs.TUIRoomRef.enterRoom({ roomId, roomParam });
        } catch (error) {
          alert('TUIRoomKit.enterRoom error: ' + error.message);
        }
      }
    },
};

</script>

<style lang="scss">
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
}
</style>
```

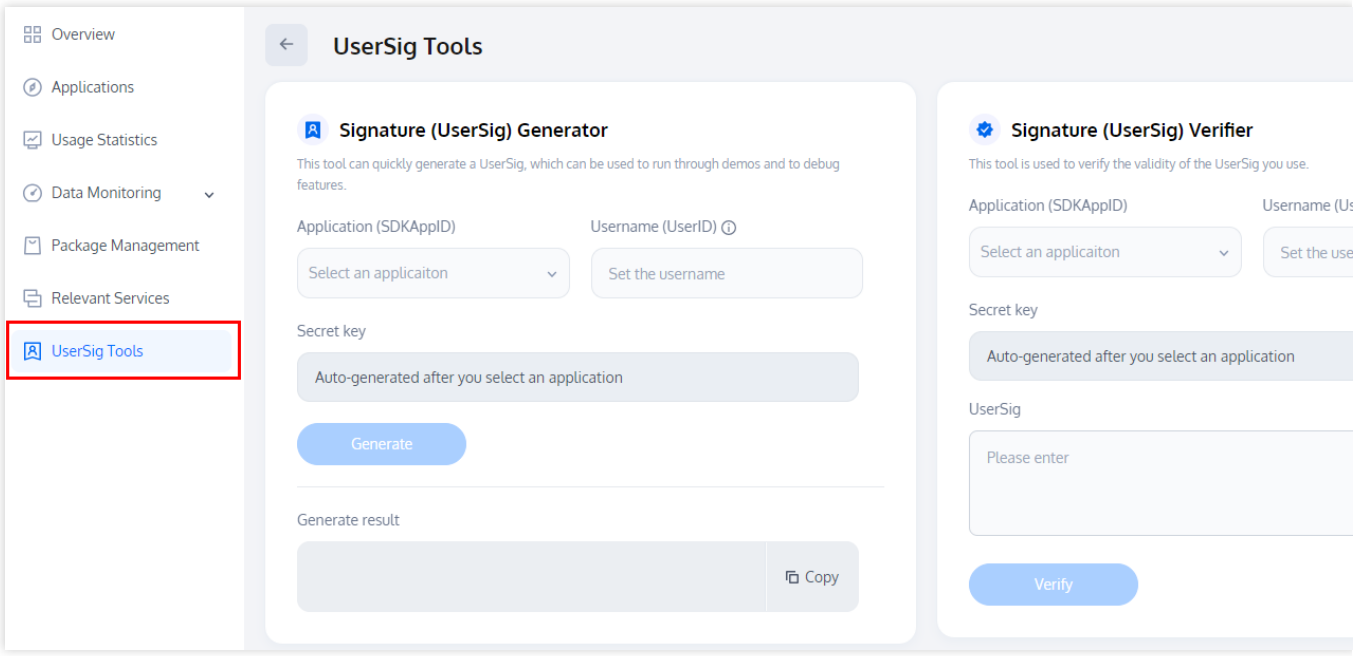
参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

SDKAppID：在 步骤一 中您已经获取到，这里不再赘述。

UserID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

UserSig：使用 步骤一 获取的 SDKSecretKey 对 SDKAppID、UserID 等信息进行加密，就可以得到 UserSig，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以通过控制台左侧项目栏中的 UserSig 工具，创建一个临时可用的 UserSig。



更多信息请参见 UserSig 相关。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下：

SDKAppID 设置错误，请正确使用国际站点的 SDKAppID，否则会无法接入。

UserSig 被错配成了加密密钥（SDKSecretKey），UserSig 是用 SDKSecretKey 把 SDKAppID、UserID 以及过期时间等信息加密得来的，而不是直接把 SDKSecretKey 配置成 UserSig。

UserID 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

Github 中的示例代码使用了 genTestUserSig 函数在本地计算 UserSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SDKSecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SDKSecretKey，所以我们强烈建议您将 UserSig 的计算逻辑放在服务端进行，并由 app 在每次使用 TUIRoomKit 组件时向您的服务器请求实时计算出的 UserSig。

步骤四：配置开发环境

TUIRoom 组件引入之后，为了确保项目可以正常运行，需要进行以下配置：

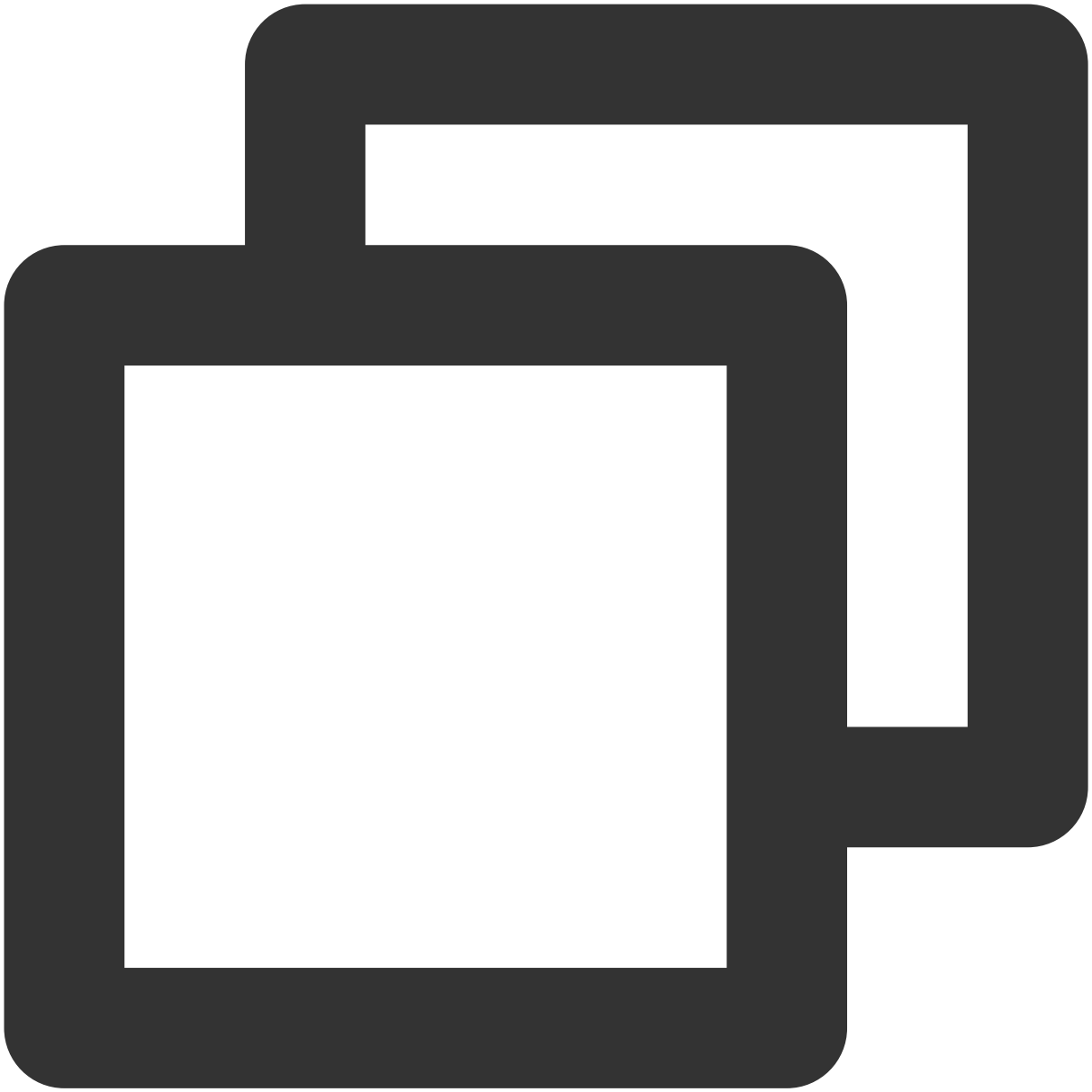
配置 Vue3 + Vite + TS 开发环境

配置 Vue2 + Webpack + TS 开发环境

配置 Vue2 + Webpack + JS 开发环境

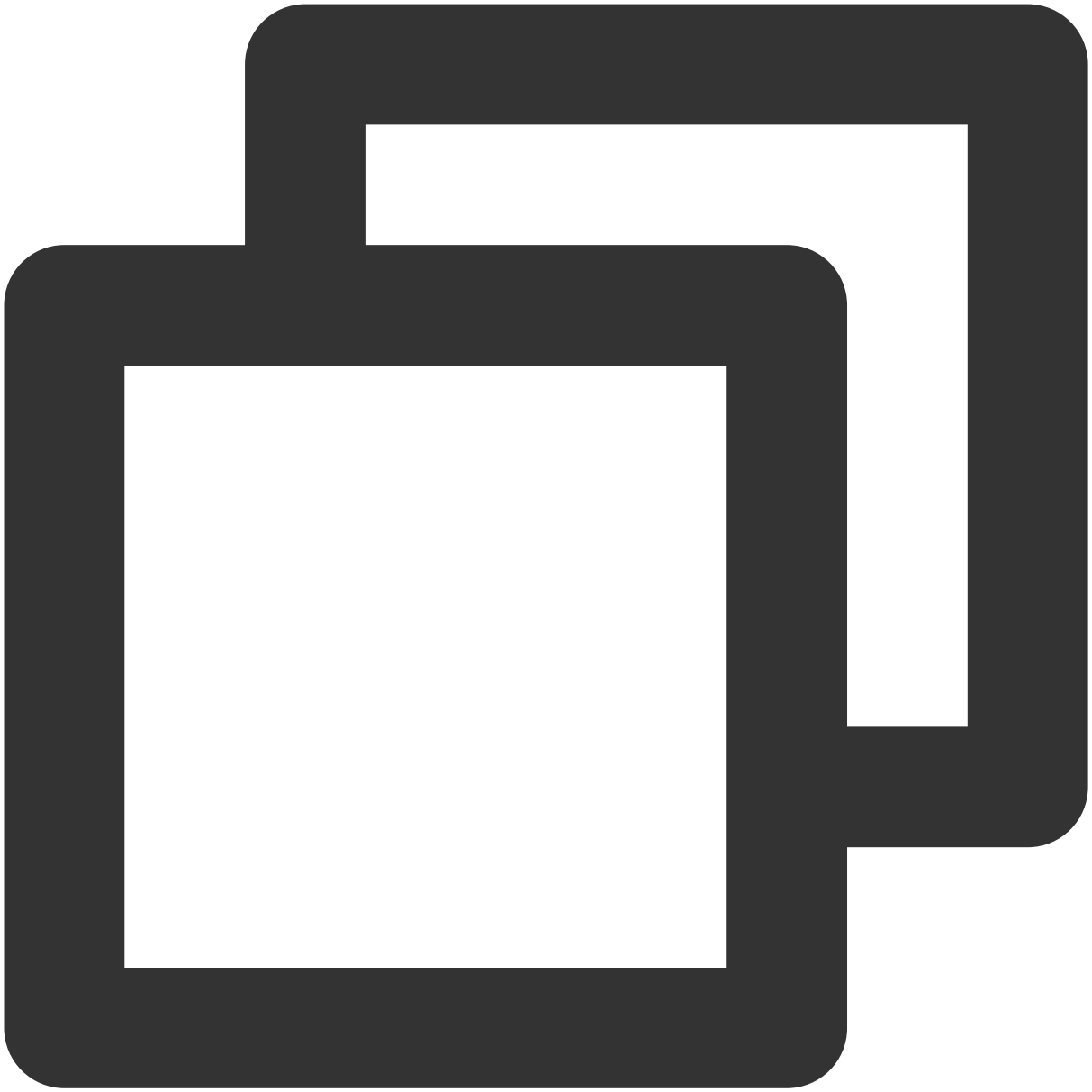
1. 安装依赖

安装开发环境依赖



```
npm install sass typescript unplugin-auto-import unplugin-vue-components --save-dev
```

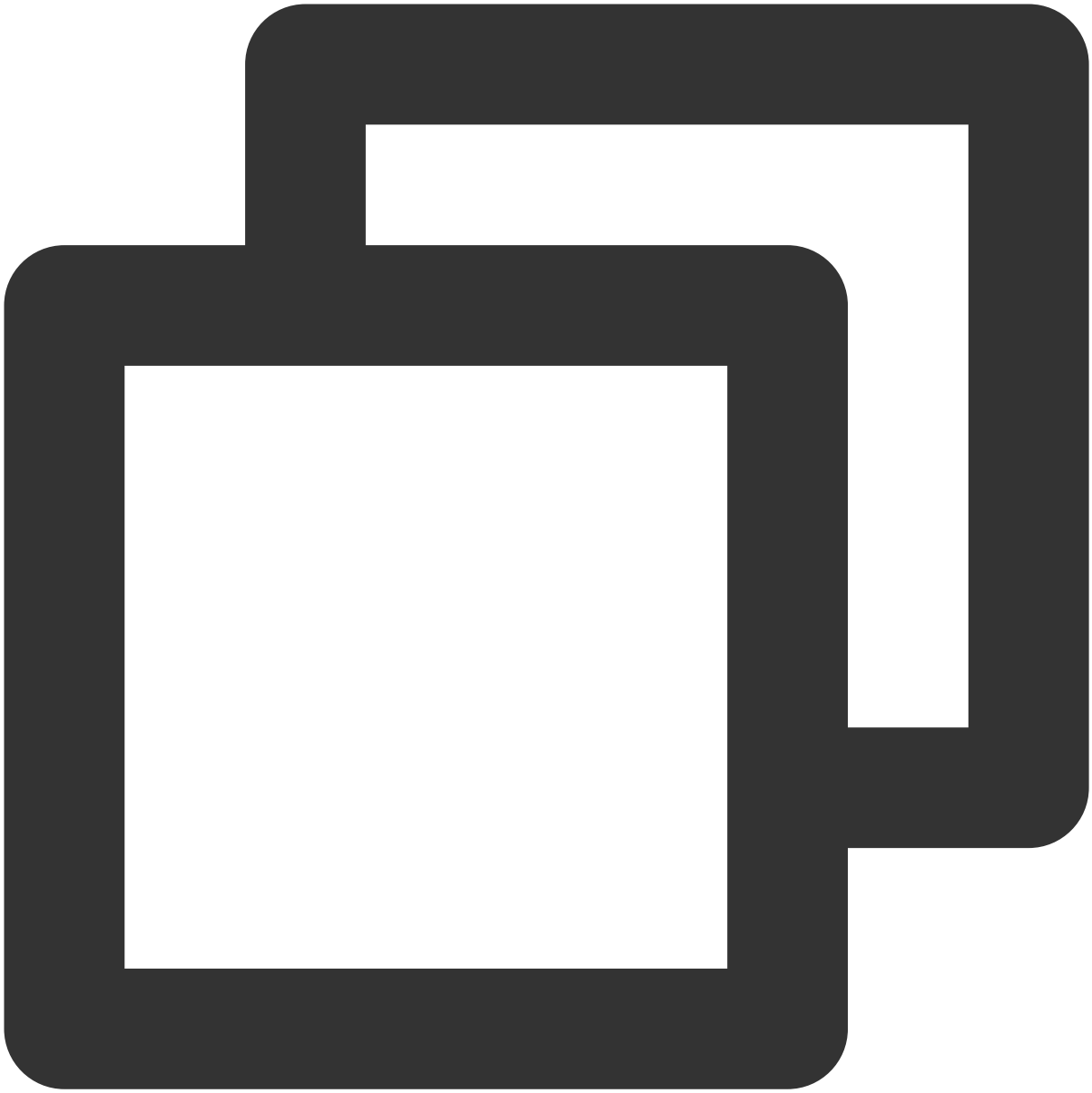
安装生产环境依赖



```
npm install events mitt pinia trtc-electron-sdk tim-js-sdk tsignaling vue-i18n @ten
```

2. 注册 Pinia

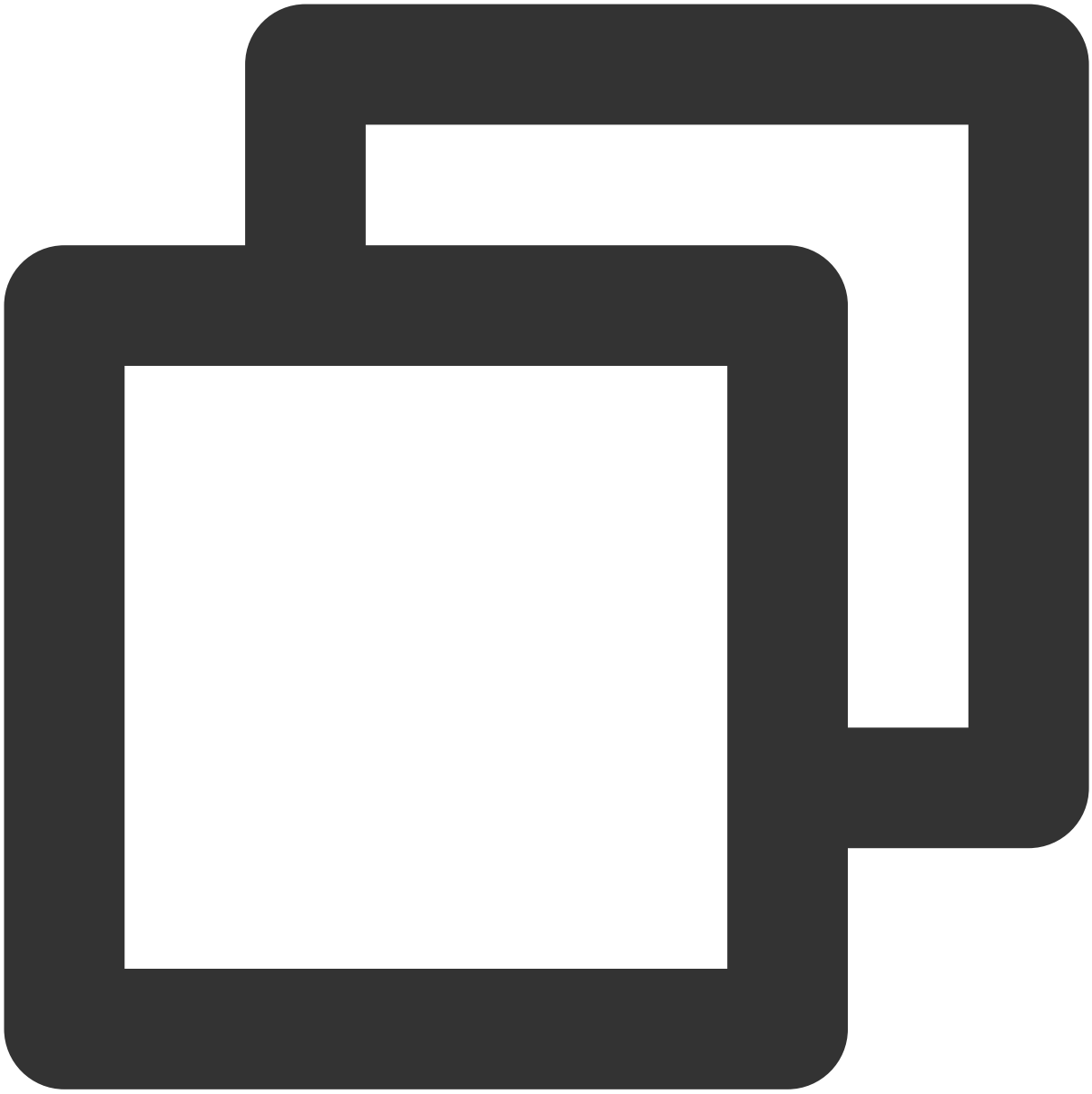
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `packages/renderer/src/main.ts` 文件。



```
// src/main.ts 文件
import { createPinia } from 'pinia';

const app = createApp(App);
// 注册Pinia
createApp(App)
  .use(createPinia())
  .mount('#app')
  .$nextTick(window.removeLoading)
```

3. 引入 trtc-electron-sdk

为了在 UI 层以 import 方式引入 trtc-electron-sdk，统一代码风格，否则必须要以 require 的方式引入，需要您在 packages/renderer/vite.config.ts 中配置。以下配置项将 resolve 中的内容替换掉, 可参见 `packages/renderer/vite.config.ts` 文件。



```
// vite.config.ts

export default defineConfig({
 // ...
 plugins: [
 resolve(
   {
```

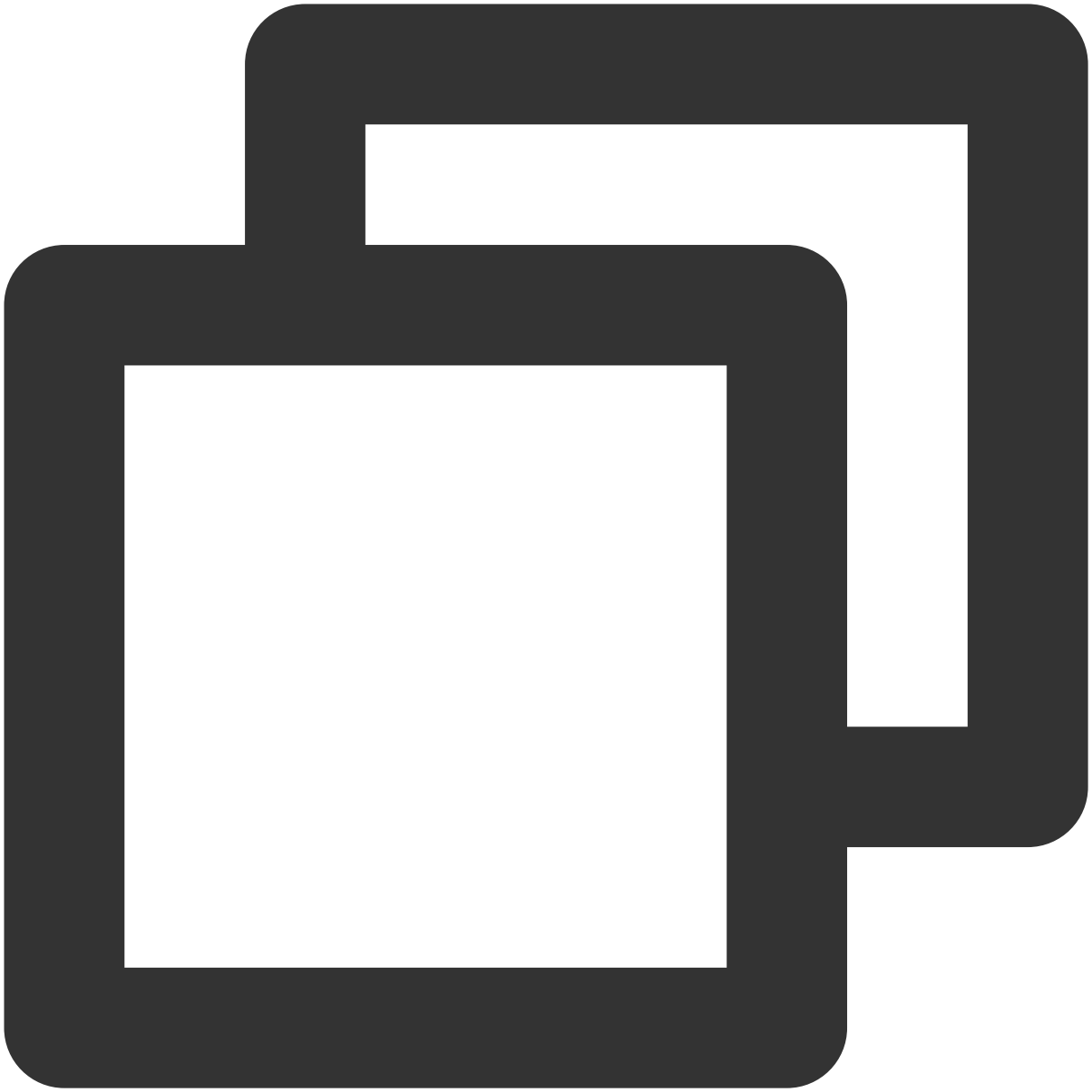
```
"trtc-electron-sdk": `
  const TRTCCloud = require("trtc-electron-sdk");
  const TRTCParams = TRTCCloud.TRTCParams;
  const TRTCAppScene = TRTCCloud.TRTCAppScene;
  const TRTCVideoStreamType = TRTCCloud.TRTCVideoStreamType;
  const TRTCScreenCaptureSourceType = TRTCCloud.TRTCScreenCaptureSourceType;
  const TRTCVideoEncParam = TRTCCloud.TRTCVideoEncParam;
  const Rect = TRTCCloud.Rect;
  const TRTCAudioQuality = TRTCCloud.TRTCAudioQuality;
  const TRTCScreenCaptureSourceInfo = TRTCCloud.TRTCScreenCaptureSourceInfo;
  const TRTCDeviceInfo = TRTCCloud.TRTCDeviceInfo;
  const TRTCVideoQosPreference = TRTCCloud.TRTCVideoQosPreference;
  const TRTCQualityInfo = TRTCCloud.TRTCQualityInfo;
  const TRTCQuality = TRTCCloud.TRTCQuality;
  const TRTCStatistics = TRTCCloud.TRTCStatistics;
  const TRTCVolumeInfo = TRTCCloud.TRTCVolumeInfo;
  const TRTCDeviceType = TRTCCloud.TRTCDeviceType;
  const TRTCDeviceState = TRTCCloud.TRTCDeviceState;
  const TRTCBeautyStyle = TRTCCloud.TRTCBeautyStyle;
  const TRTCVideoResolution = TRTCCloud.TRTCVideoResolution;
  const TRTCVideoResolutionMode = TRTCCloud.TRTCVideoResolutionMode;
  const TRTCVideoMirrorType = TRTCCloud.TRTCVideoMirrorType;
  const TRTCVideoRotation = TRTCCloud.TRTCVideoRotation;
  const TRTCVideoFillMode = TRTCCloud.TRTCVideoFillMode;
  const TRTCRoleType = TRTCCloud.TRTCRoleType;
  const TRTCScreenCaptureProperty = TRTCCloud.TRTCScreenCaptureProperty;
  export {
    TRTCParams,
    TRTCAppScene,
    TRTCVideoStreamType,
    TRTCScreenCaptureSourceType,
    TRTCVideoEncParam,
    Rect,
    TRTCAudioQuality,
    TRTCScreenCaptureSourceInfo,
    TRTCDeviceInfo,
    TRTCVideoQosPreference,
    TRTCQualityInfo,
    TRTCStatistics,
    TRTCVolumeInfo,
    TRTCDeviceType,
    TRTCDeviceState,
    TRTCBeautyStyle,
    TRTCVideoResolution,
    TRTCVideoResolutionMode,
    TRTCVideoMirrorType,
    TRTCVideoRotation,
```

版权所有：腾讯云计算（北京）有限责任公司

```
          TRTCVideoFillMode,
          TRTCRoleType,
          TRTCQuality,
          TRTCScreenCaptureProperty,
        };
        export default TRTCCloud.default;
      `
    }
  ),
  // ...
  ]
  // ...
});
```

4. env.d.ts 文件配置

env.d.ts 文件配置需要您在 `packages/renderer/src/env.d.ts` 中配置。以下配置项为增量配置，不要删除已经存在的 `env.d.ts` 文件配置。

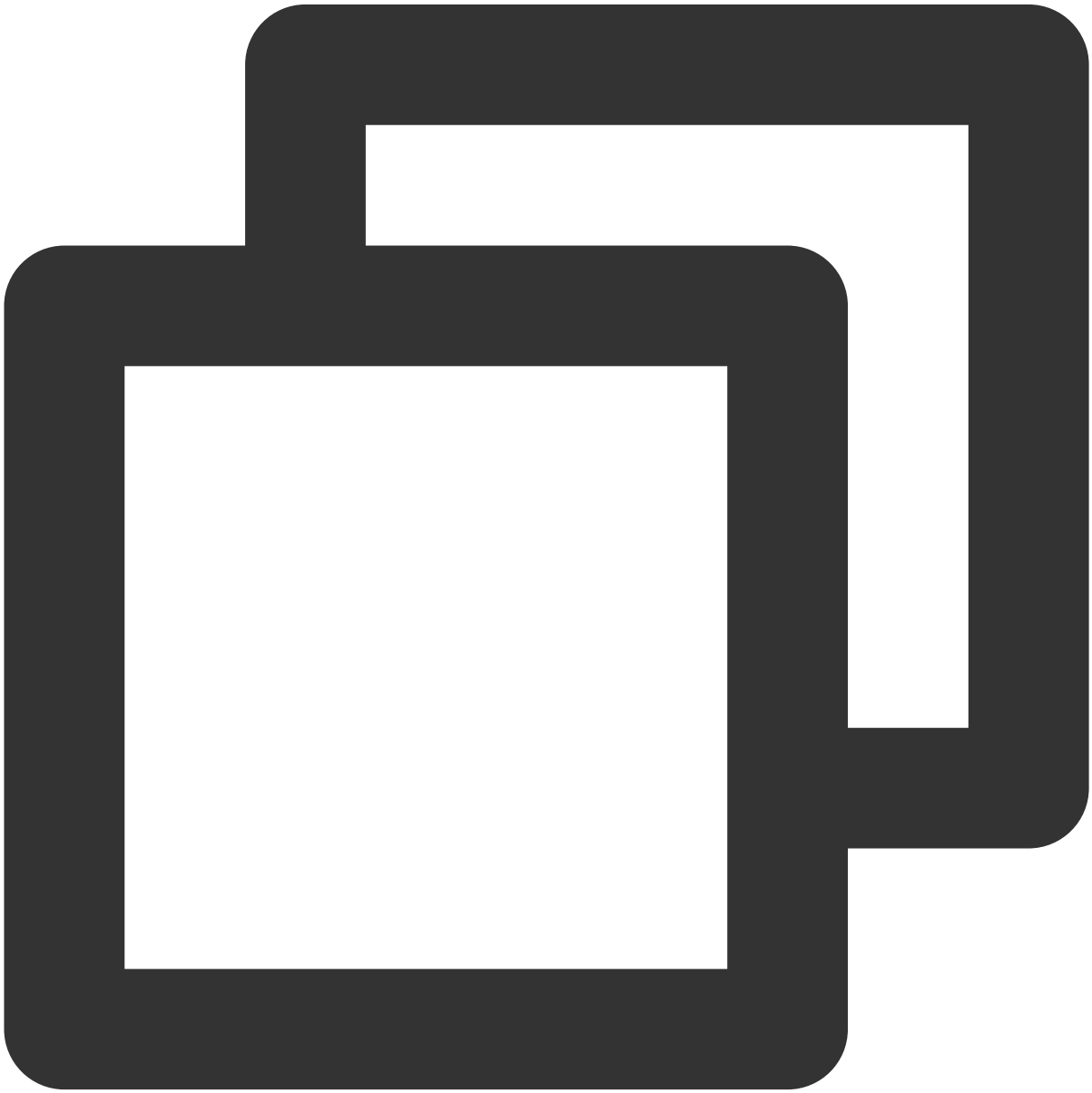


```
// env.d.ts
declare module '*.vue' {
  import { DefineComponent } from 'vue'
  // eslint-disable-next-line @typescript-eslint/no-explicit-any, @typescript-eslin
  const component: DefineComponent<{}, {}, any>
  export default component
}

declare module 'tsignaling/tsignaling-js' {
  import TSignaling from 'tsignaling/tsignaling-js';
  export default TSignaling;
```

```
}

declare module 'tim-js-sdk' {
  import TIM from 'tim-js-sdk';
  export default TIM;
}

declare const Aegis: any;
```

5. 配置中英文切换

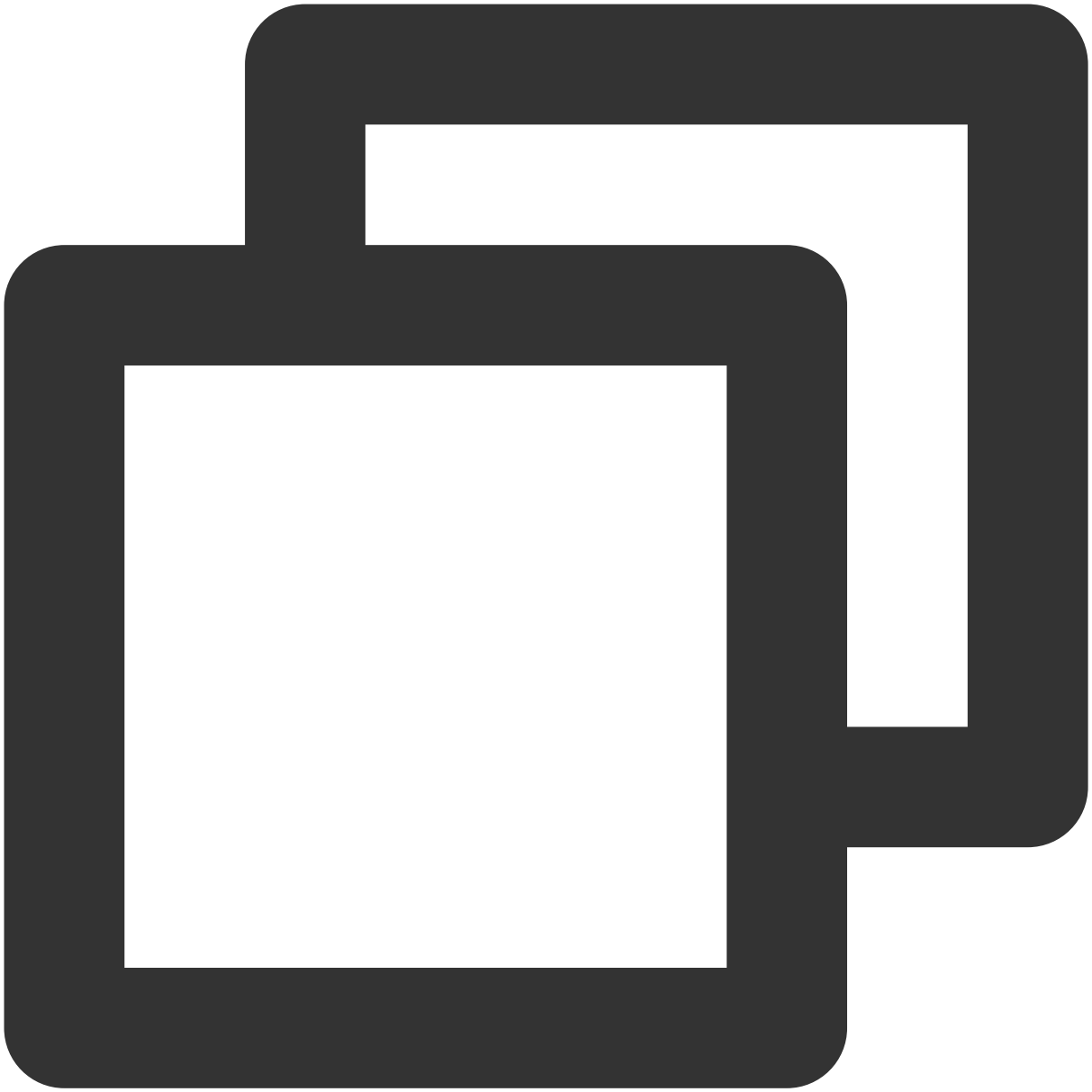
TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



```
// src/main.ts
// 引入 locales/index.ts 文件，请确认引入路径是否正确
import VueI18n from './TUIRoom/locales/index';

createApp(App)
  .use(createPinia())
  .use(VueI18n)
  .mount('#app')
  .$nextTick(window.removeLoading);
```

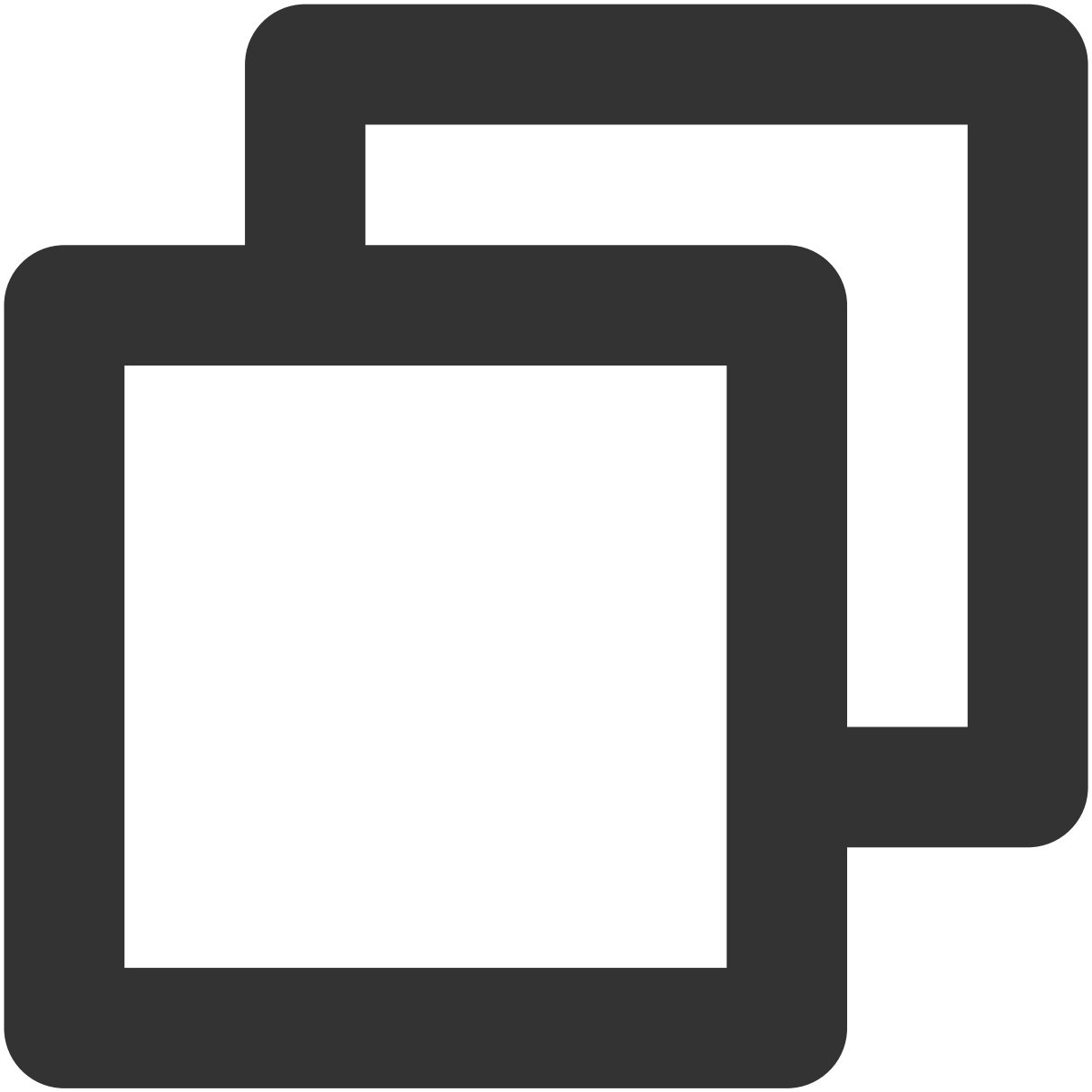
1. 安装依赖

安装开发环境依赖



```
npm install sass sass-loader -S -D
```

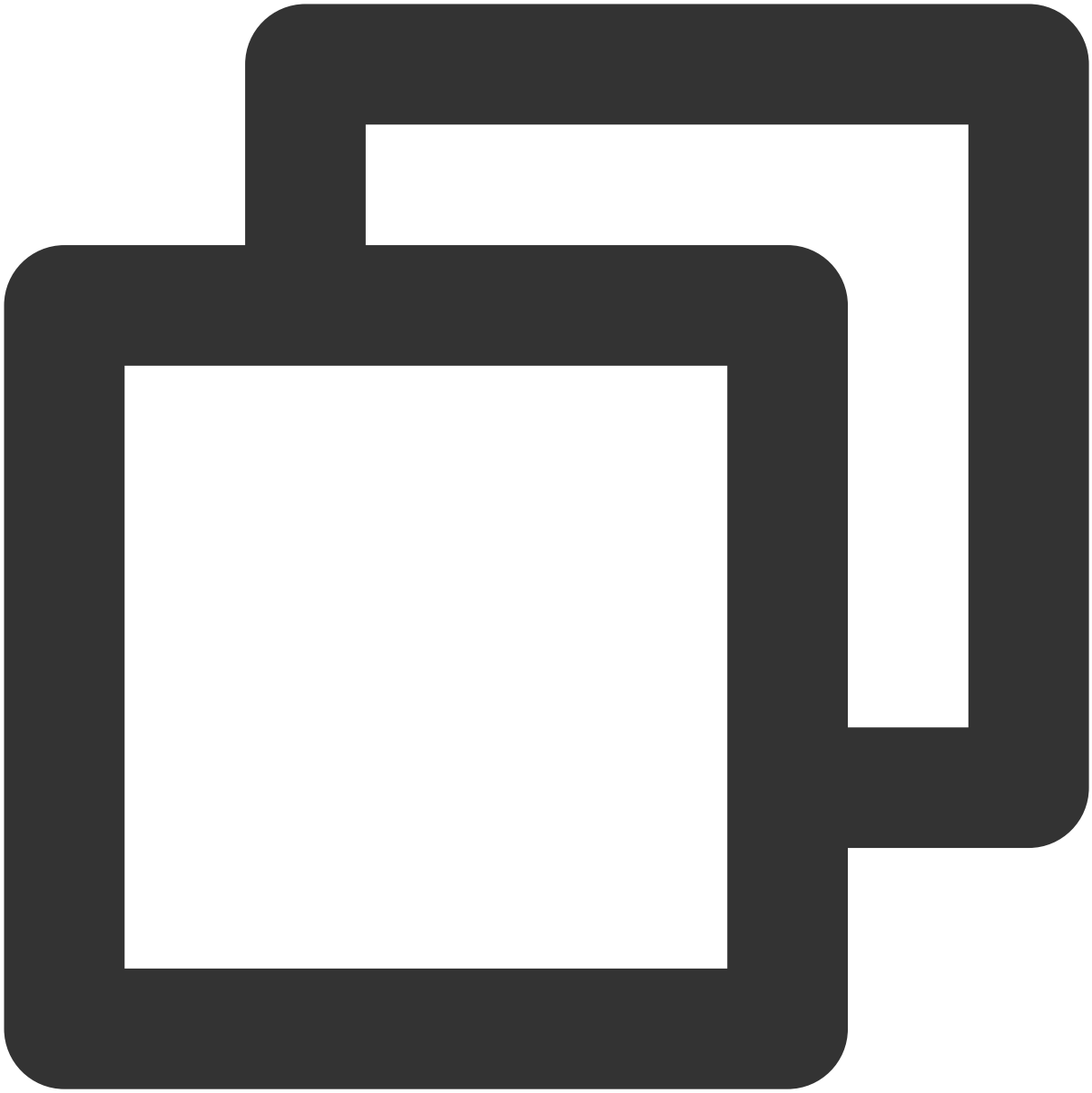
安装生产环境依赖



```
npm install vue@^2.7 pinia trtc-electron-sdk tim-js-sdk tsignaling @tencentcloud/tu
```

2. 注册 Pinia

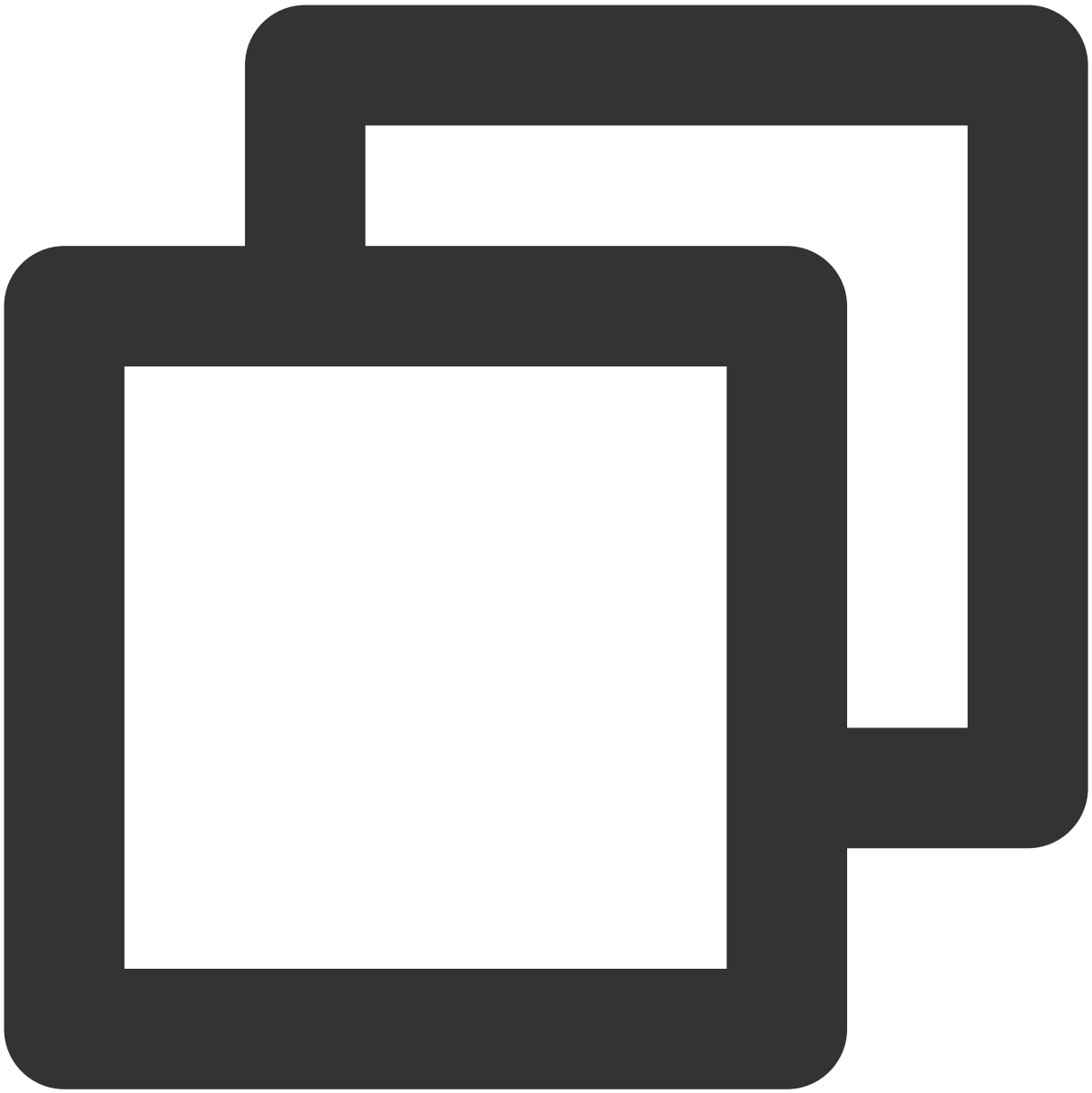
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 文件。



```
import { createPinia, PiniaVuePlugin } from 'pinia';

Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

3. 配置中英文切换

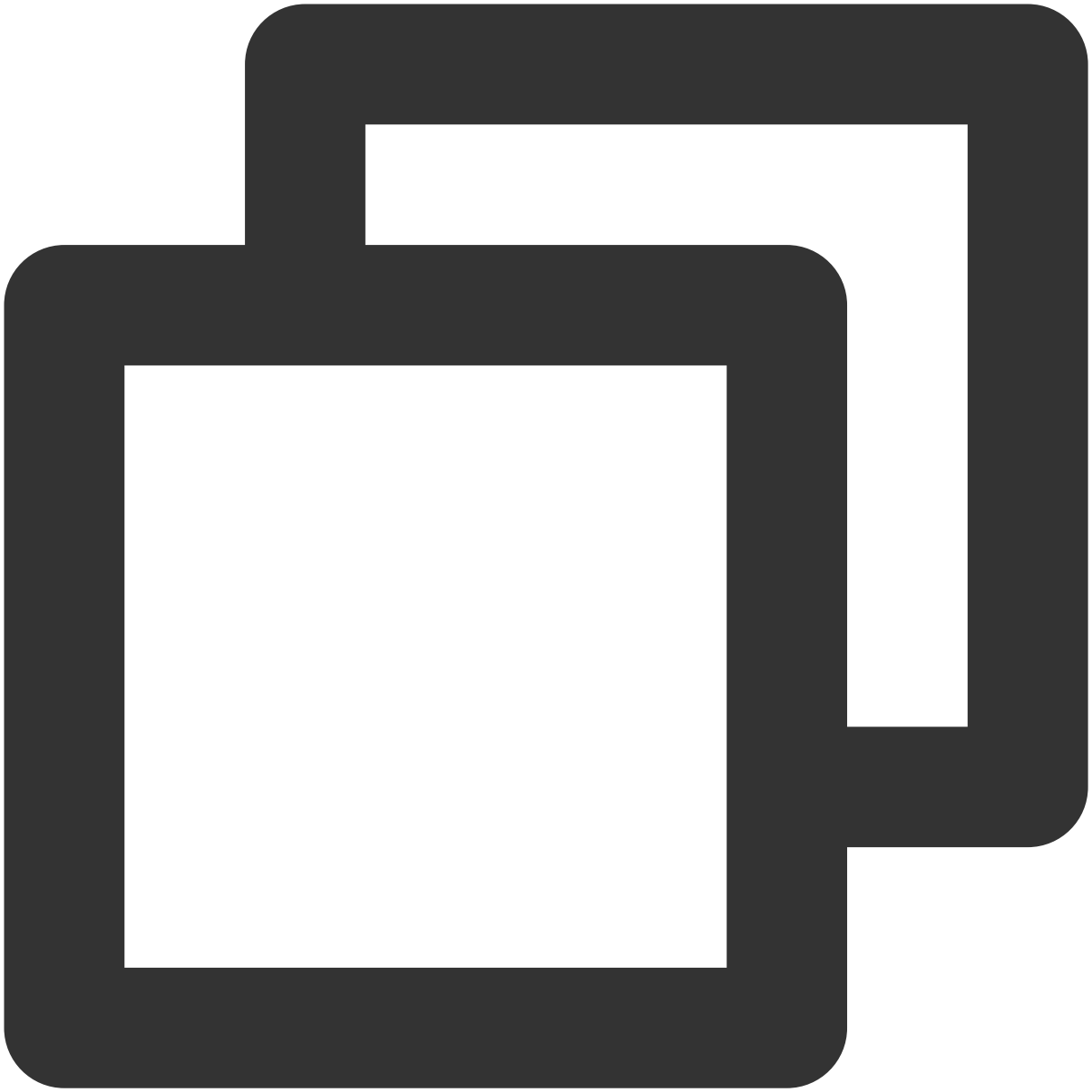
TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



```
import i18n from './TUIRoom/locales/';

Vue.use(i18n);
```

TUIRoom 默认支持环境为 Vue2 + Webpack + TS 环境，如果您想在Vu2 + Webpack + JS 环境中集成 TUIRoom 组件，您需要做如下配置。

1. 配置 typescript 支持 TUIRoom 组件加载。



```
vue add typescript
```

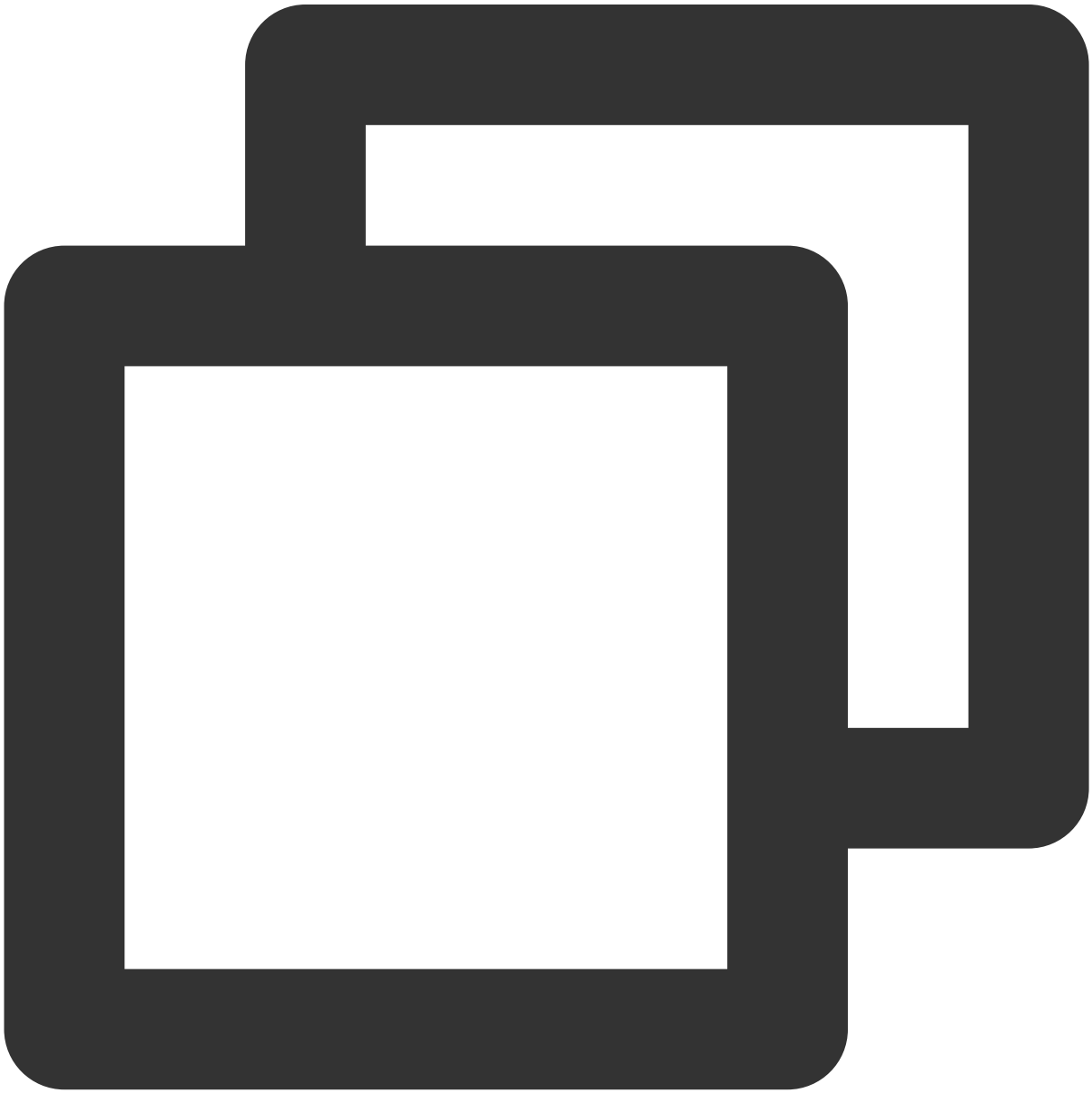
配置 TS 开发环境的选项可参考图片

版权所有：腾讯云计算（北京）有限责任公司

```
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected
polyfills, transpiling JSX)? Yes
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? No
```

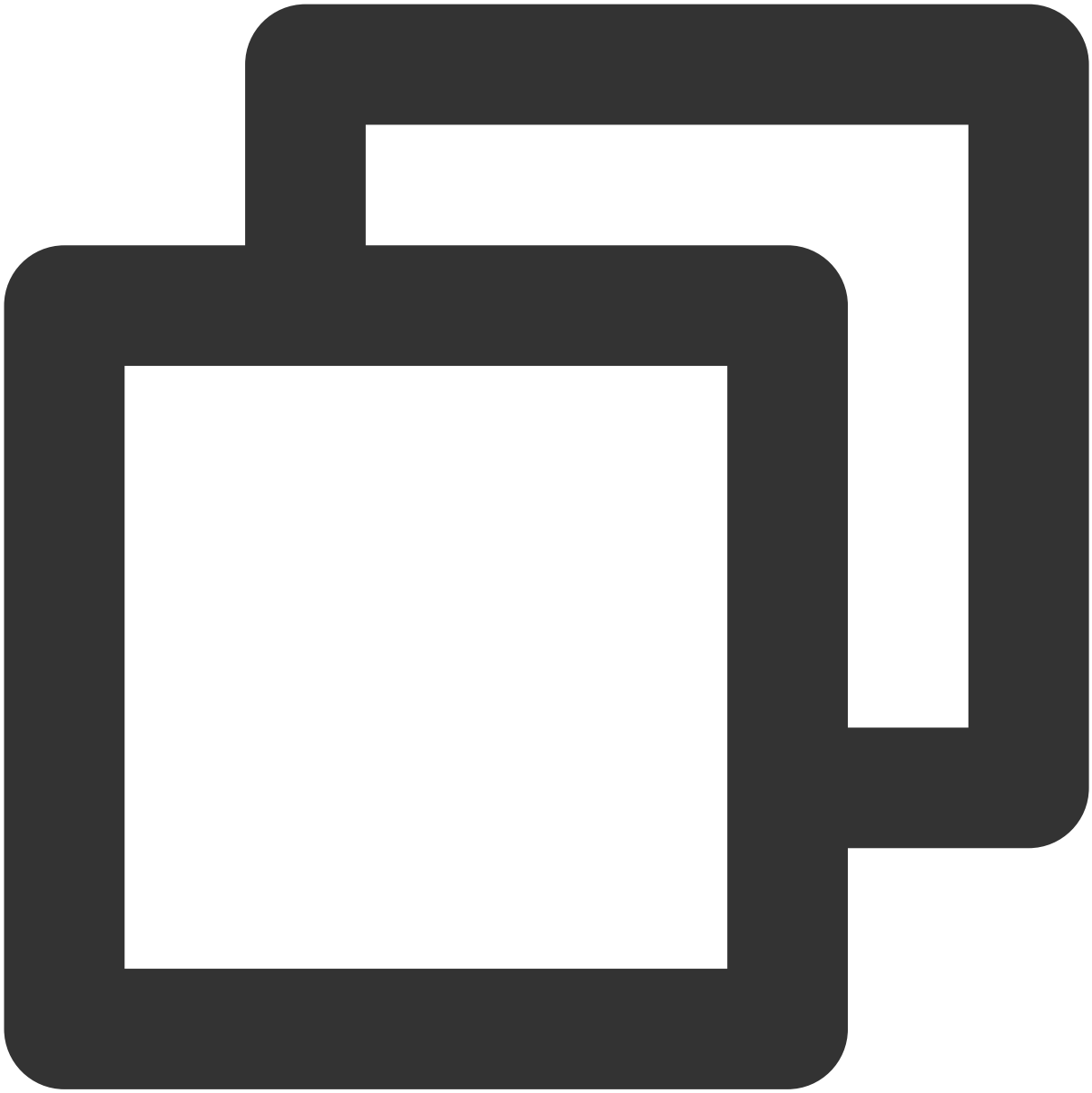
2. 安装依赖

安装开发环境依赖



```
npm install sass sass-loader -S -D
```

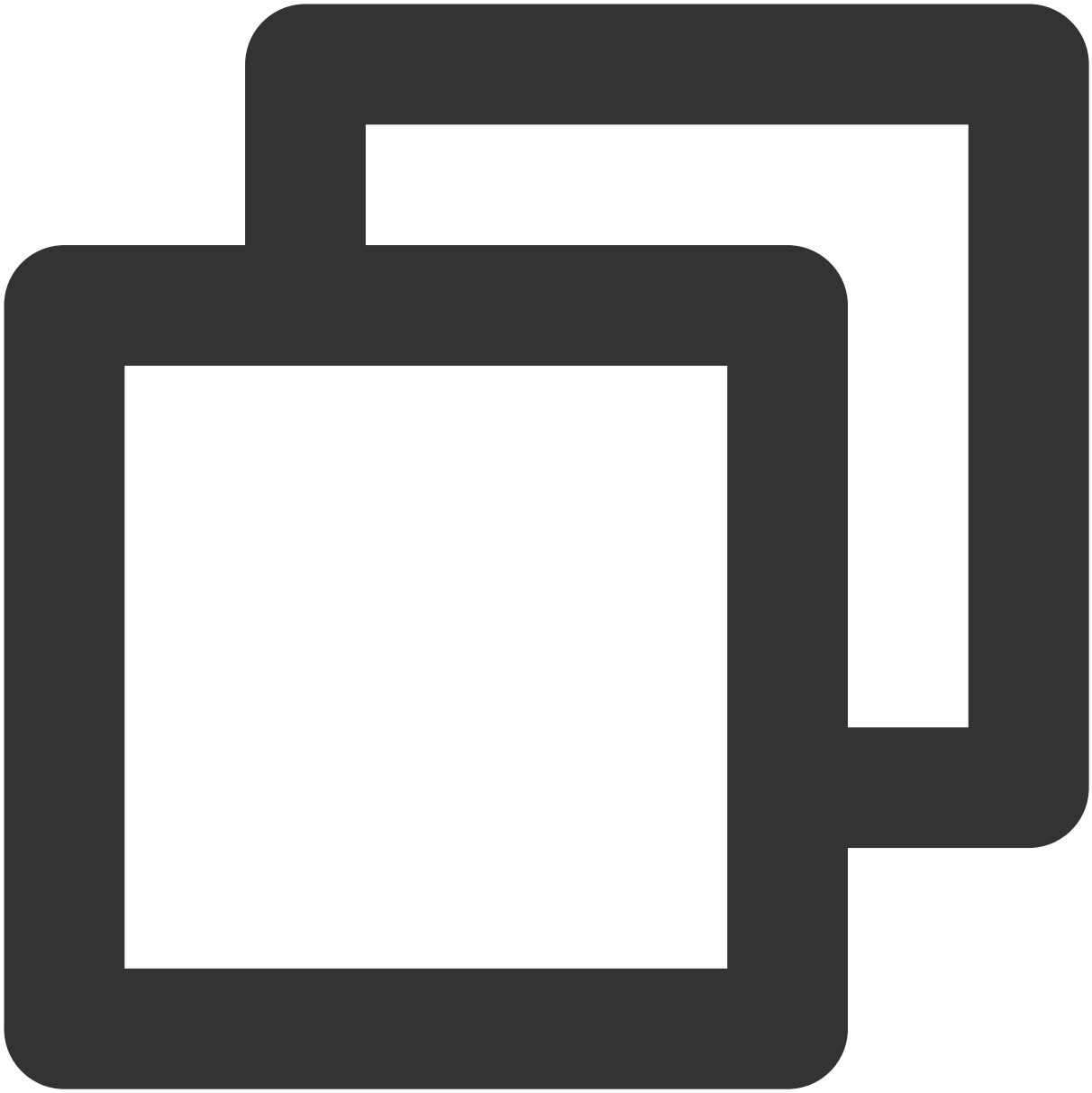
安装生产环境依赖



```
npm install vue@^2.7 pinia trtc-electron-sdk tim-js-sdk tsignaling @tencentcloud/tu
```

3. 注册 Pinia

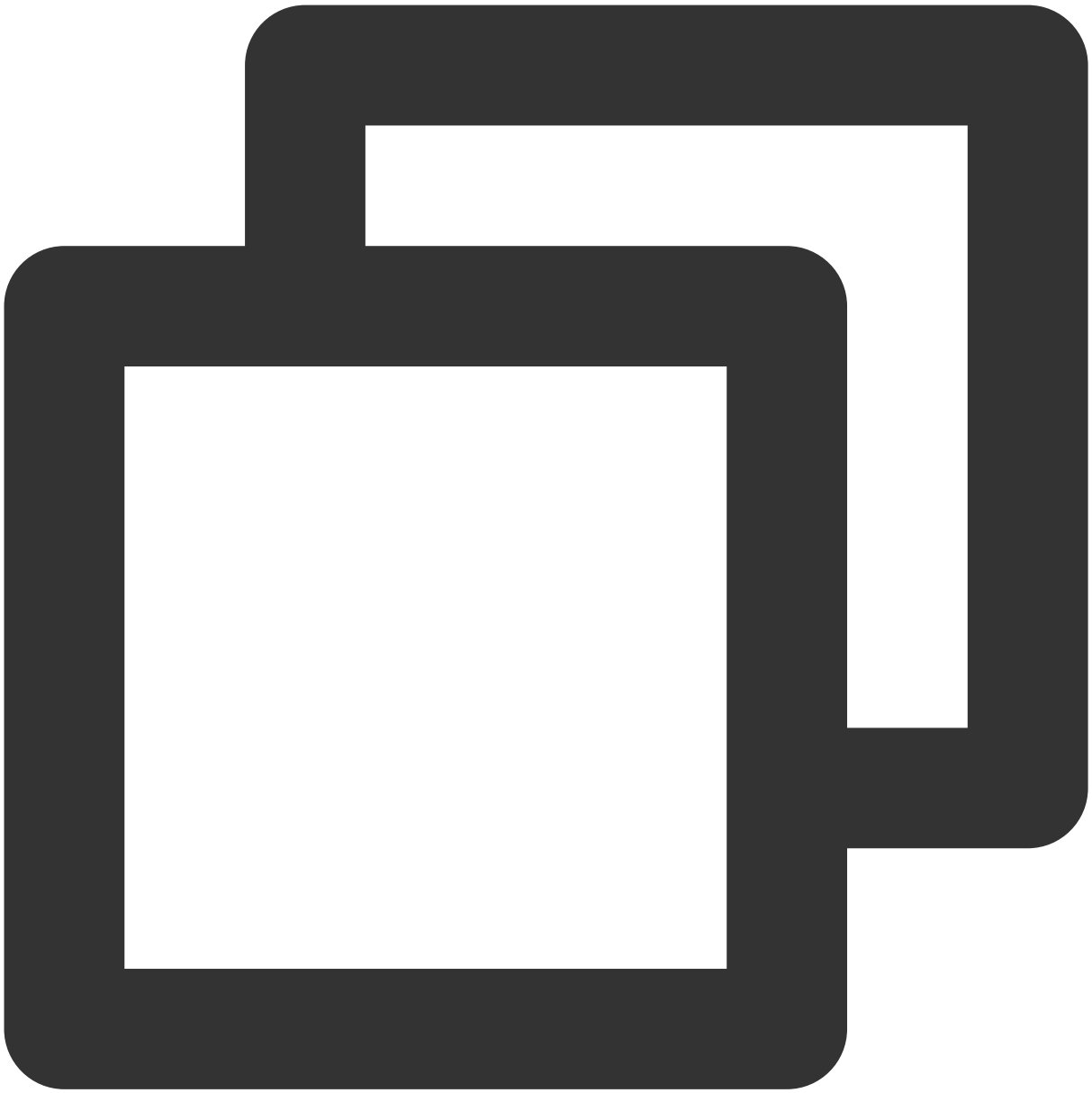
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 文件。



```
import { createPinia, PiniaVuePlugin } from 'pinia';

Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

4. 配置中英文切换

TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



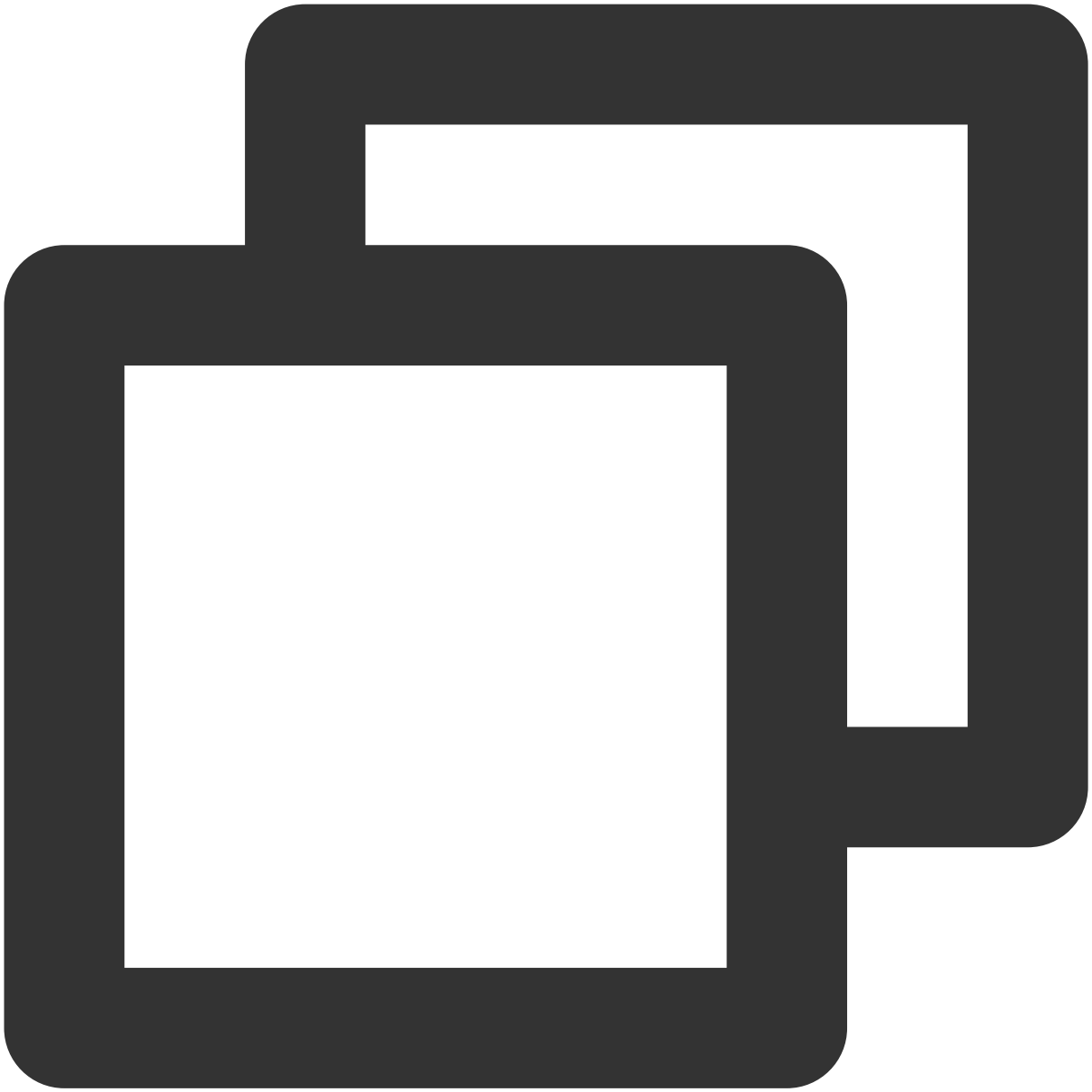
```
import i18n from './TUIRoom/locales/';

Vue.use(i18n);
```

步骤五：开发环境运行

Vue3 + Vite + TS 项目

Vue2 + Webpack + TS 项目

1. 执行开发环境命令



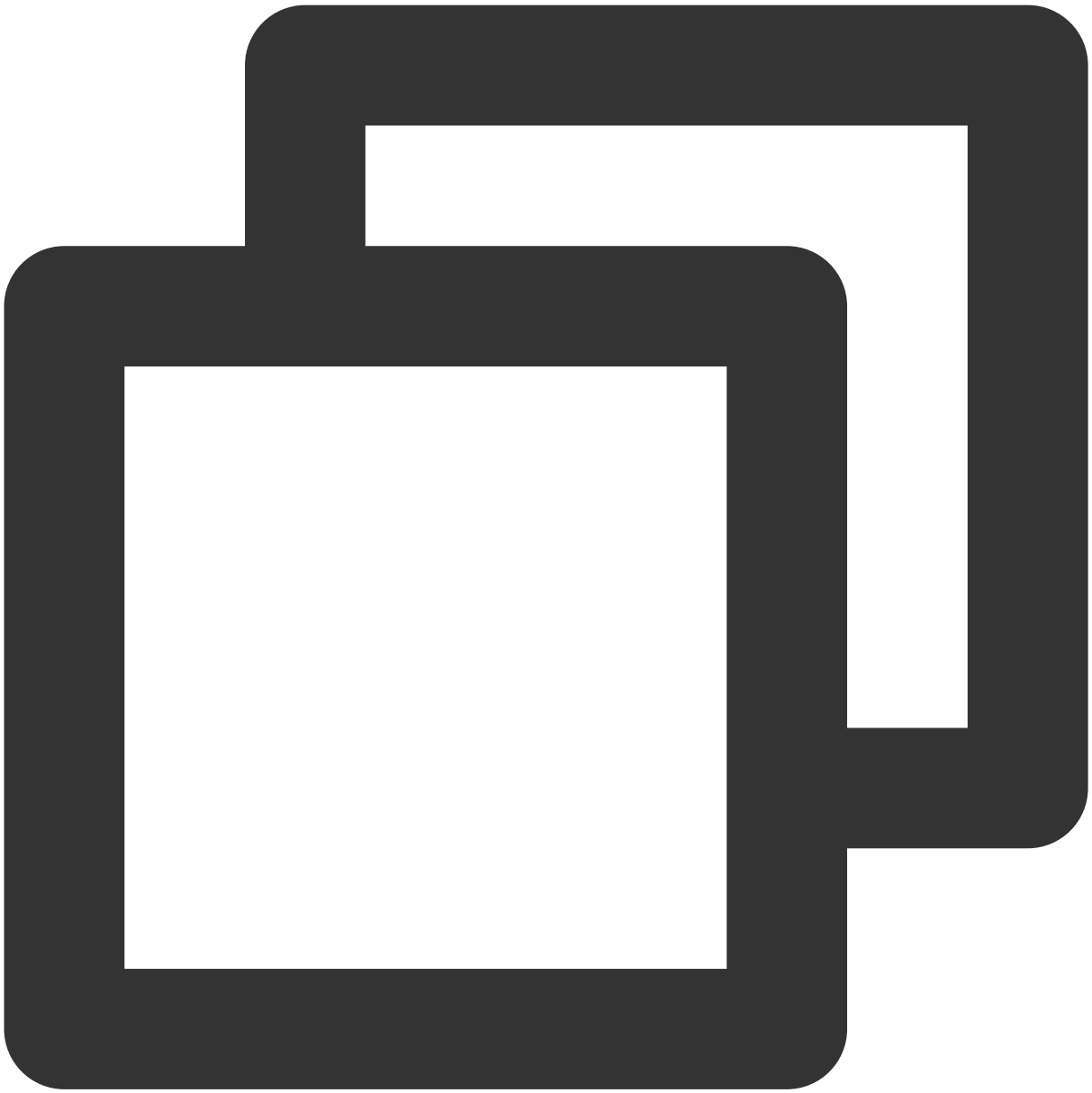
```
npm run dev
```

2. 体验 TUIRoom 组件功能

注意：

运行过程中若 src/TUIRoom 目录中有 eslint 报错，可在 .eslintignore 文件中添加 /src/TUIRoom/ 路径屏蔽 eslint 检查。

1. 执行开发环境命令



```
npm run start
```

注意：

Electron 默认监听端口号为 8080，若出现打包完成 UI 没有渲染的情况，可能是默认端口被占用，在 `main.electron.js` 中修改对应端口号即可。

2. 体验 TUIRoom 组件功能

注意：

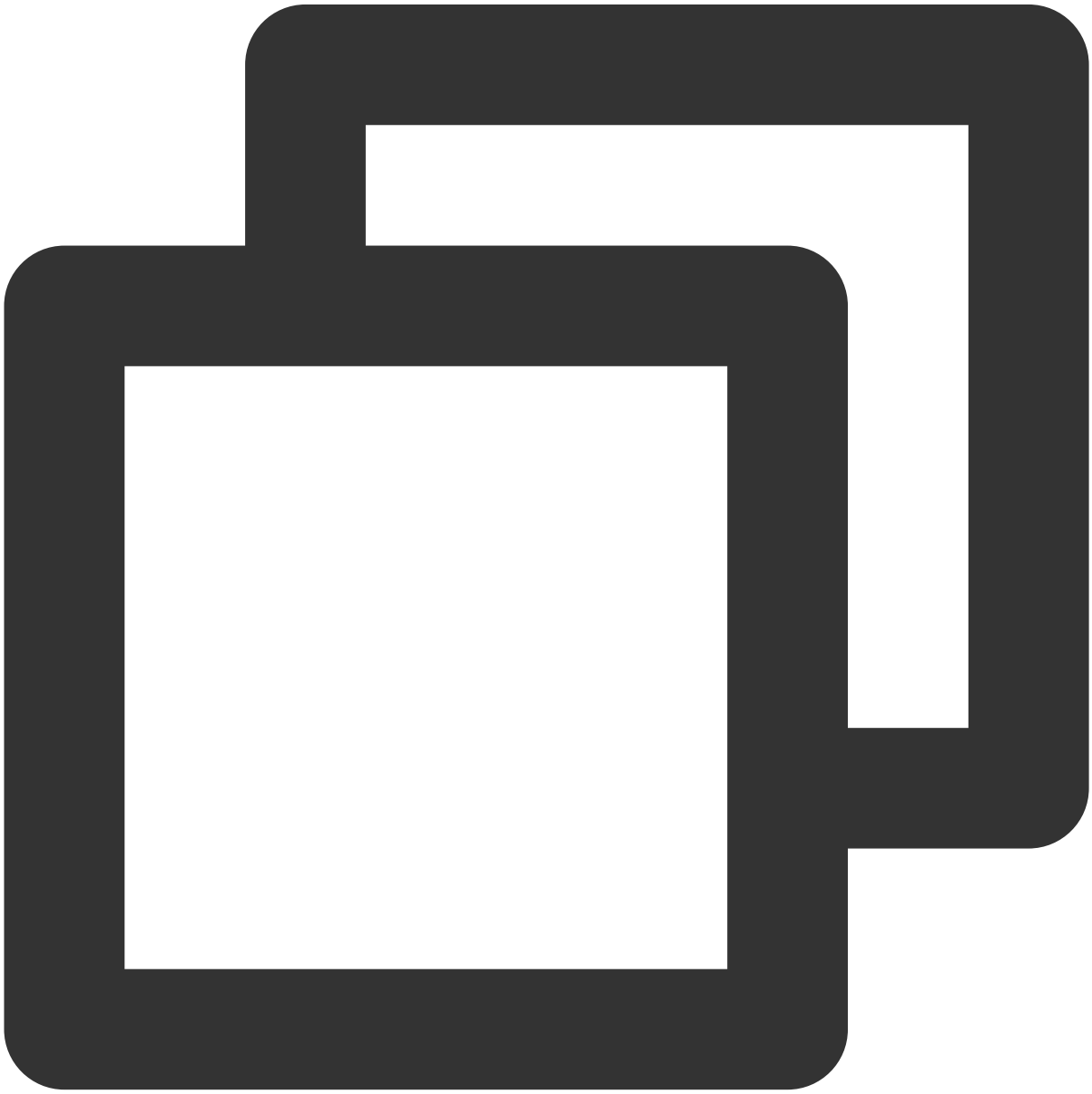
运行过程中若 src/TUIRoom 目录中有 eslint 报错，可在 .eslintignore 文件中添加 /src/TUIRoom/ 路径屏蔽 eslint 检查。

步骤六：构建安装包、运行

Vue3 + Vite + TS 项目

Vue2 + Webpack + TS 项目

在命令行终端中，执行以下命令构建安装包，构建好的安装包位于 release 目录下，可以安装运行。



```
npm run build
```

在命令行终端中，执行 package.json 中对应的构建命令，构建好的安装包位于 bin 目录下，可以安装运行。

注意：

只能使用 Mac 电脑构建 Mac 安装包，使用 Windows 电脑构建 Windows 安装包, 暂不支持交叉编译。

常见问题

1. 运行时报错："does not provide an export named 'createBlock' "？

若按照上述步骤接入后，运行时出现以下错误，请您务必将 vite 版本固定在 3.0.0 以下，将 vue 版本固定在 3.4.9 以下。

```
/node_modules/.vite/deps/vue.js?v=b083d236' does not provide an export named 'create
```

原因是项目提供的 electron 工程模板支持的 vite 版本指定在3.0.0以下，但是 vue 的 3.4.9 及以上版本中的属性需要升级 vite 的版本来获取支持，这里已经向 vue 官方提了 issue。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

Windows

最近更新时间：2024-04-12 11:29:03

本文将介绍如何用最短的时间完成 `TUIRoomKit` 组件的接入，跟随本文档，您将在一个小时内完成如下几个关键步骤，并最终得到一个包含完备 UI 界面的多人音视频房间功能。包括成员管理、屏幕分享、聊天、自由发言申请发言模式等功能。

步骤一：开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

步骤二：下载 TUIRoomKit 工程代码

1. **下载 TUIRoomKit 组件代码**

单击 Github , 克隆或下载 TUIRoomKit 仓库代码，Windows-Mac 目录为 Windows 工程所在目录。

2. **修改用户配置**

进入 Windows-Mac 目录后，打开 `utils\\usersig\\win\\GenerateTestUserSig.h` 文件，修改 SDKAppID 和 密钥。

步骤三：配置开发环境

Windows 环境 ：

Visual Studio 2015及以上集成开发环境。

QT5.9.1及以上版本的 Qt 开发库。

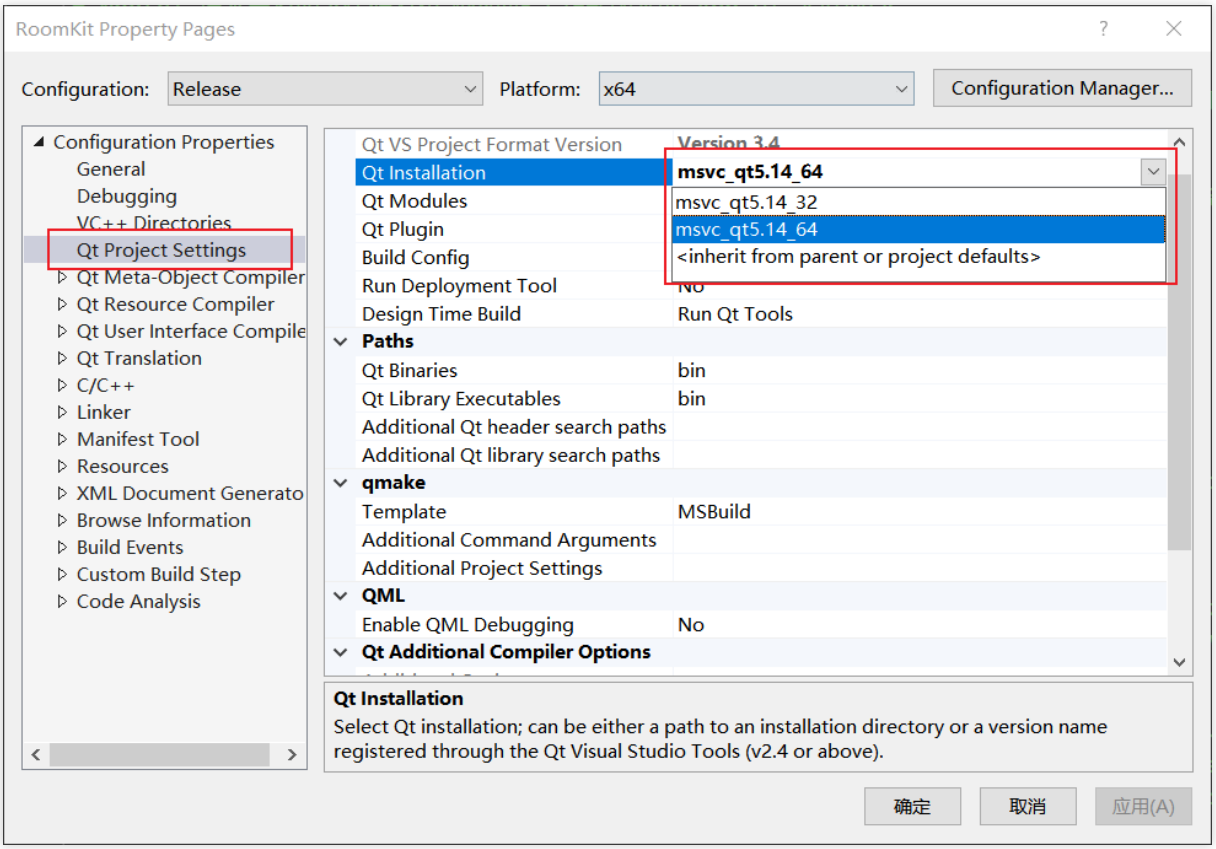
VS 下的 QT 开发插件 Qt Visual Studio Tools 2.2.0及以上。

最低支持系统：Windows 8.1。

请确保您的集成开发环境能够正常开发。

步骤四：配置 TUIRoomKit 工程

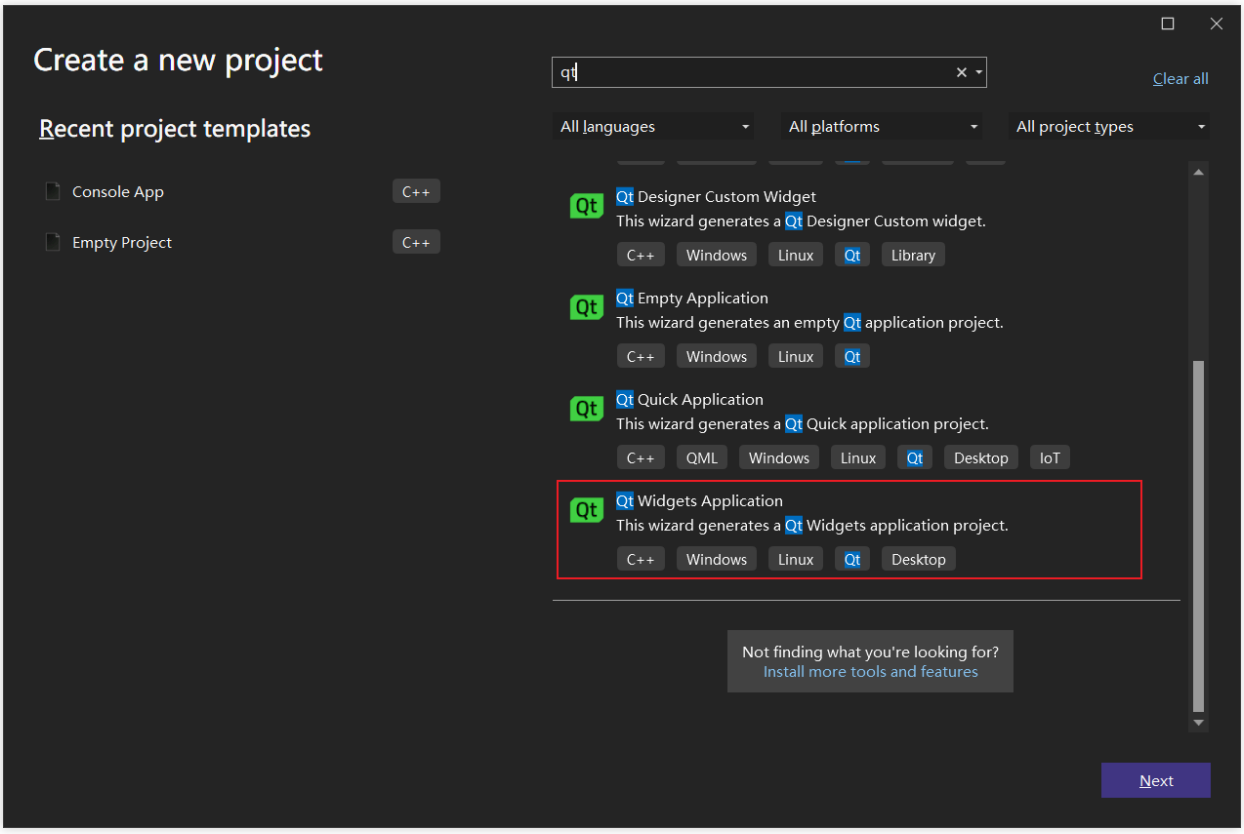
1. 进入 RoomKit 目录，使用 Visual Studio 打开 `RoomKit.vcxproj` 文件，打开 TUIRoomKit 工程。
2. 配置 QT 开发设置：

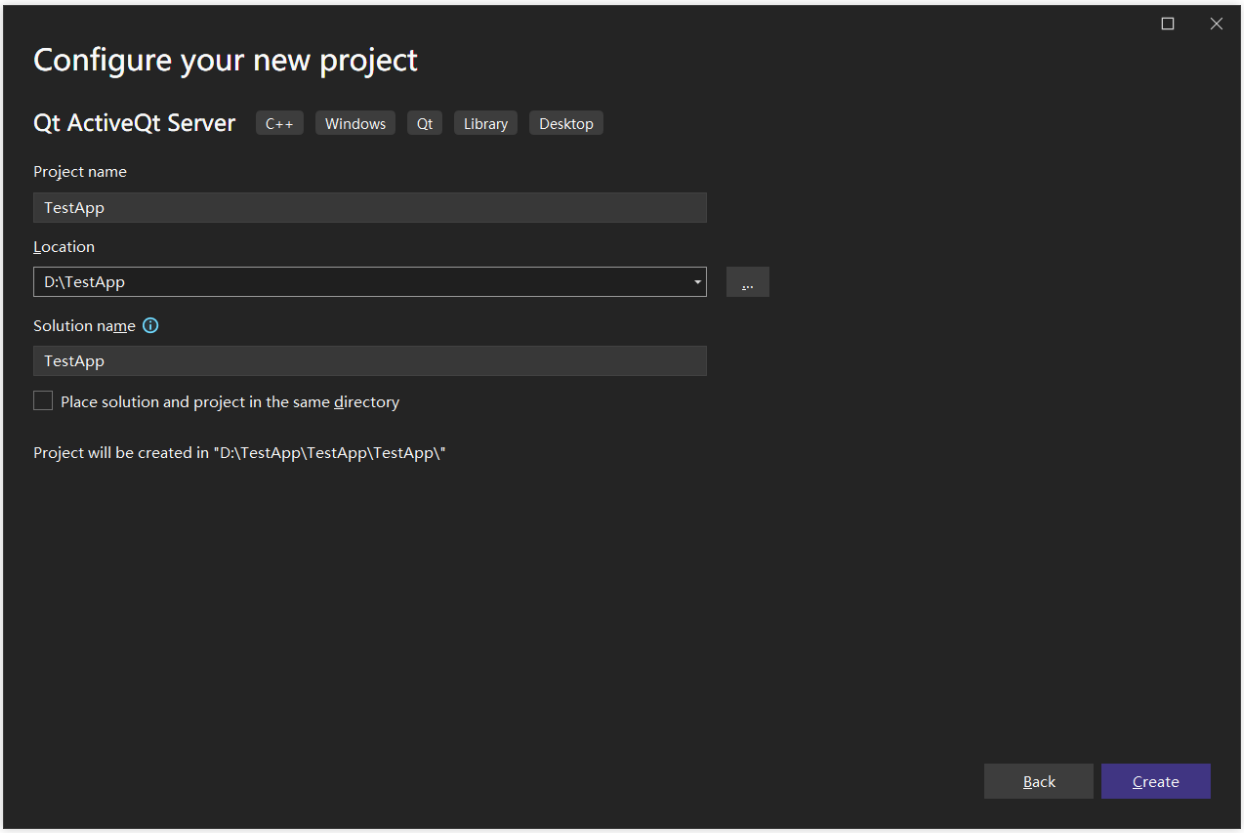


3. 编译 TUIRoomKit 工程，生成 RoomKit.exe 程序。

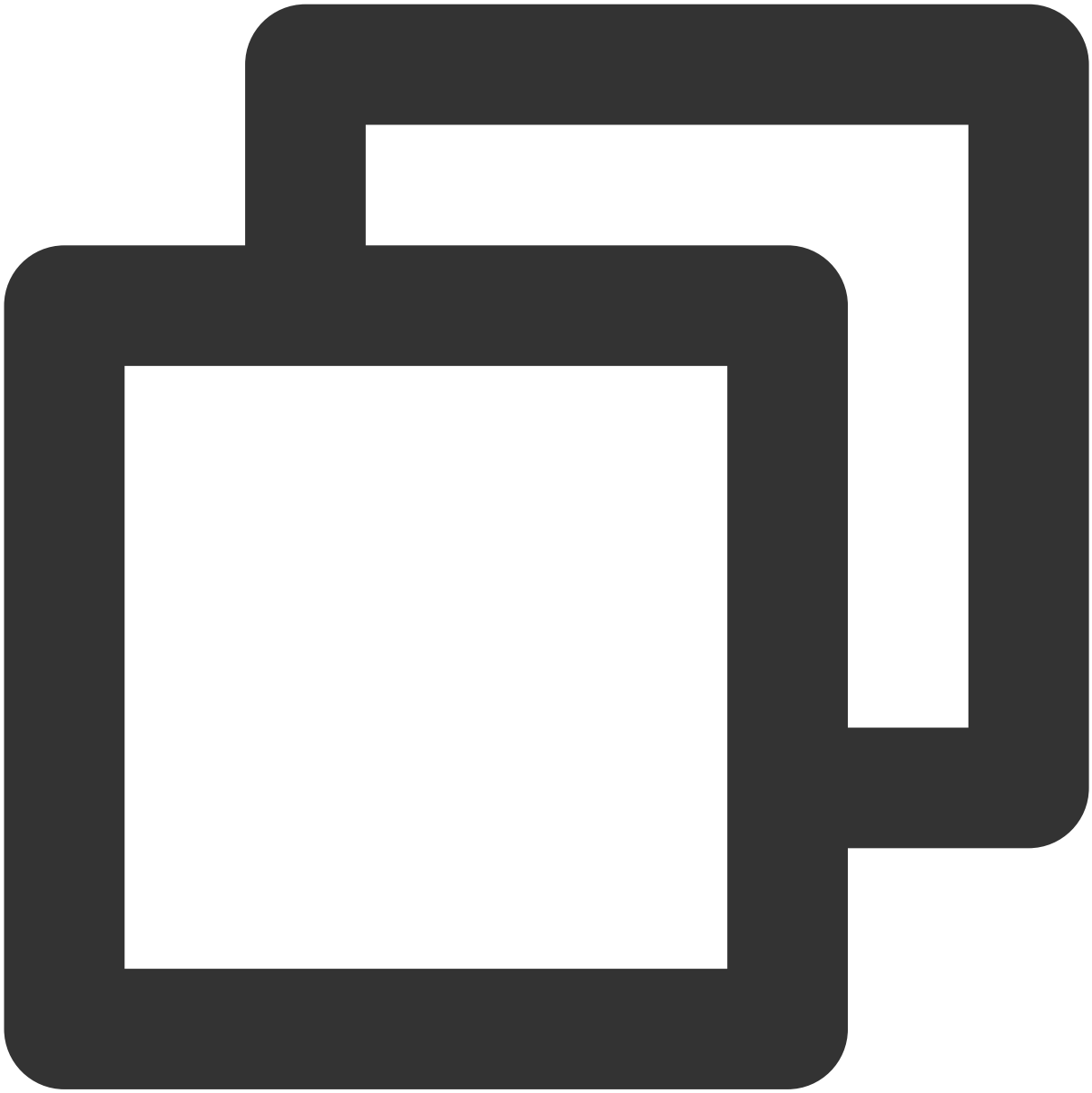
步骤五：外部启动 TUIRoomKit 程序

1. 打开 VS，选择 Qt Widgets Application 工程类型，创建 TestApp 工程。

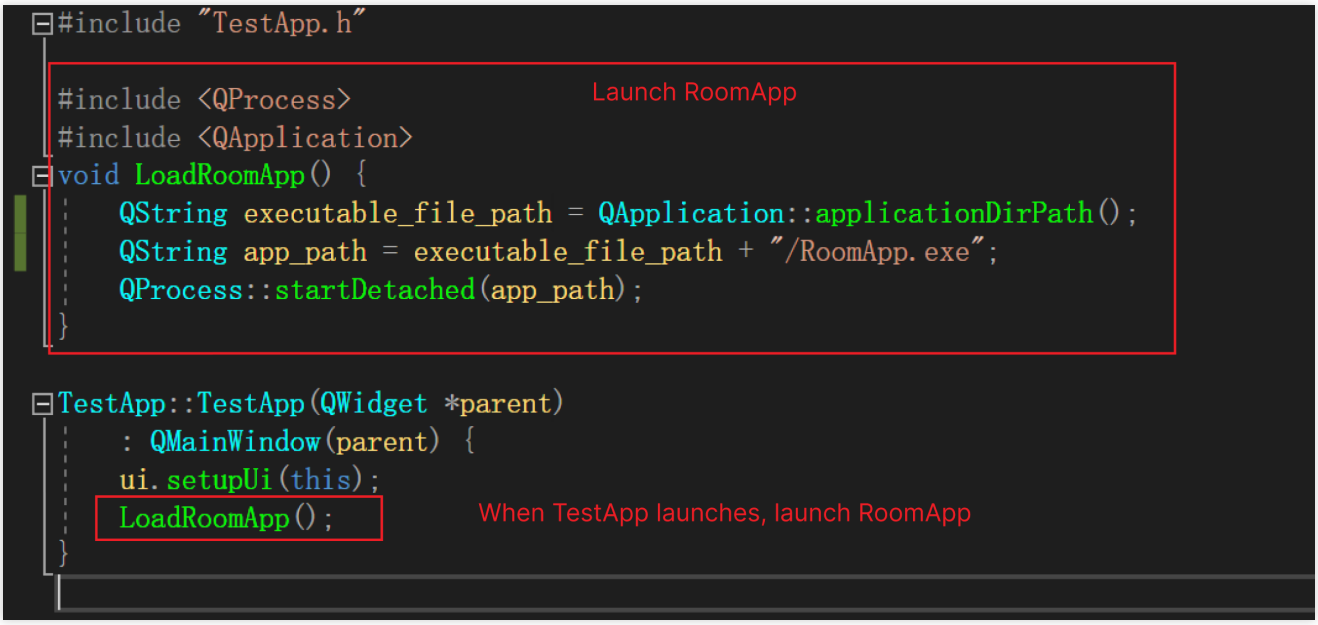




2. 编写启动进程的程序，并在合适的位置调用 LoadRoomApp 函数。



```
#include <QProcess>
#include <QApplication>
void LoadRoomApp() {
  QString executable_file_path = QApplication::applicationDirPath();
  QString app_path = executable_file_path + "/RoomKit.exe";
  QProcess::startDetached(app_path);
}
```



3. 编译项目，并将 RoomKit 编译的成果物复制到当前可执行程序目录，以 release x86 程序为例：
复制 `TUIRoomKit\\Windows-Mac\\RoomKit\\bin\\Win32\\Release` 目录下所有文件到当前程序目录下。
4. 执行程序，启动 TestApp 的同时启动 RoomKit。

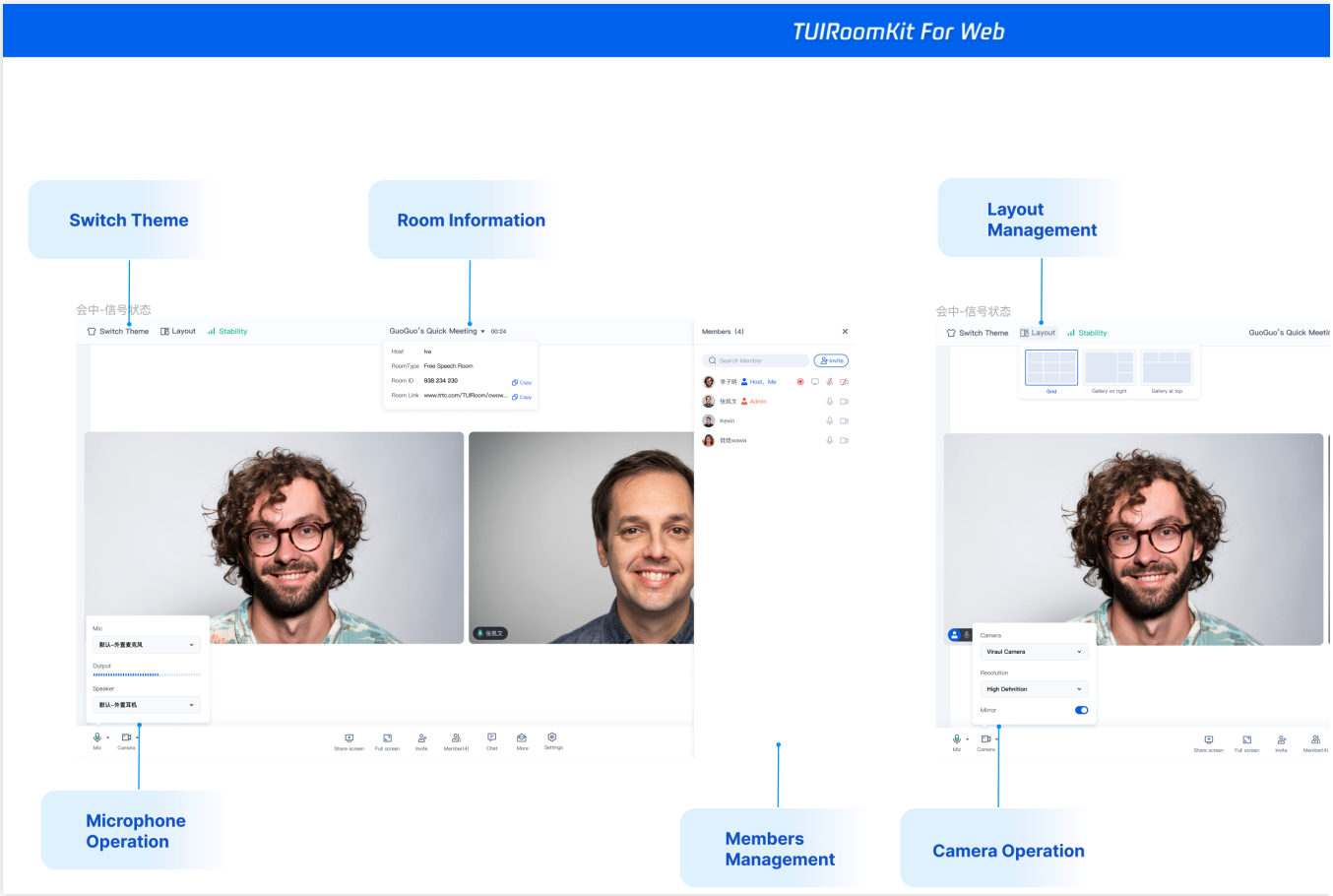
交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

Web

最近更新时间：2024-04-12 11:29:00

TUIRoomKit 是腾讯云推出的多人音视频 UI 组件。组件提供房间管理，音视频控制，屏幕共享，成员管理，麦位管理，即时聊天，自定义布局切换等丰富的功能交互，同时支持中英文切换，一键换肤等能力。



本文介绍 TUIRoomKit(Web) 的接入指引，助力您快速上线企业会议、在线教育、医疗问诊、在线巡视、远程定损等业务场景。

您可以单击 TUIRoomKit 在线体验链接 体验 TUIRoomKit 更多功能。

您可以单击 Github 下载 TUIRoomKit 代码，并参考代码仓库 README.md 文档跑通 TUIRoomKit Web 示例工程。

步骤一：开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

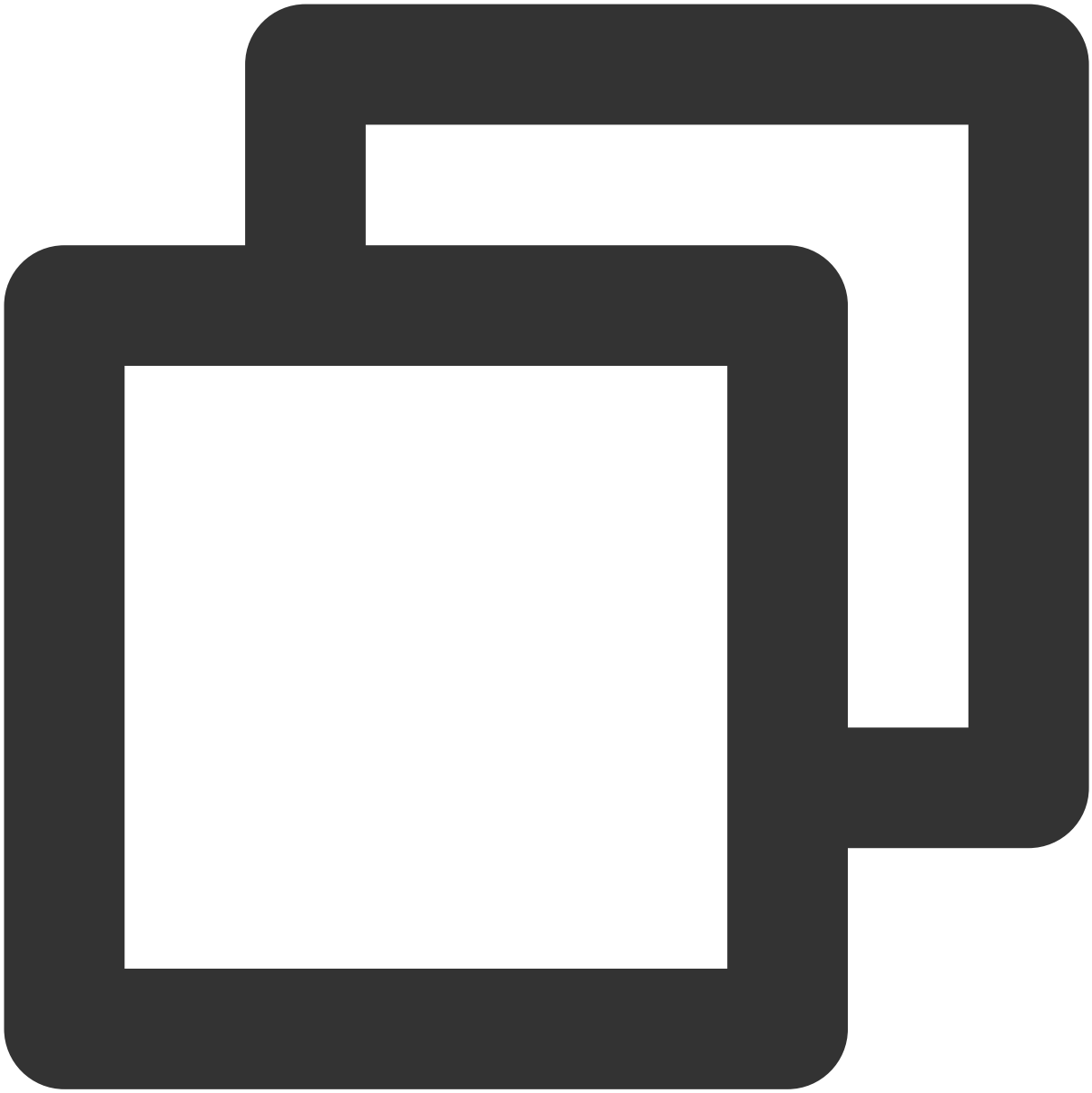
版权所有：腾讯云计算（北京）有限责任公司

步骤二：准备 Vue 工程代码

打开您已有 Vue 工程，可跳过该步骤。
如果无 Vue 项目，可选择以下方式生成模版工程。
生成 Vue3 + Vite + TS 模版工程
生成 Vue3 + Webpack + TS 模版工程
生成 Vue2 + Webpack + JS 模版工程

注意：

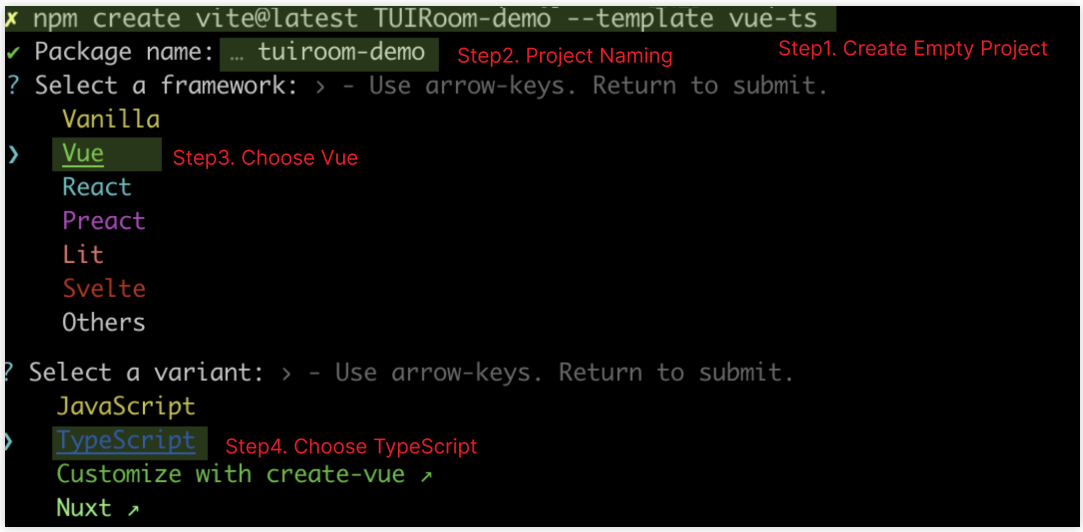
Vite 需要 Node.js 版本 14.18+，16+。然而，有些模板需要依赖更高的 Node 版本才能正常运行，当您的包管理器发出警告时，请注意升级您的 Node 版本。



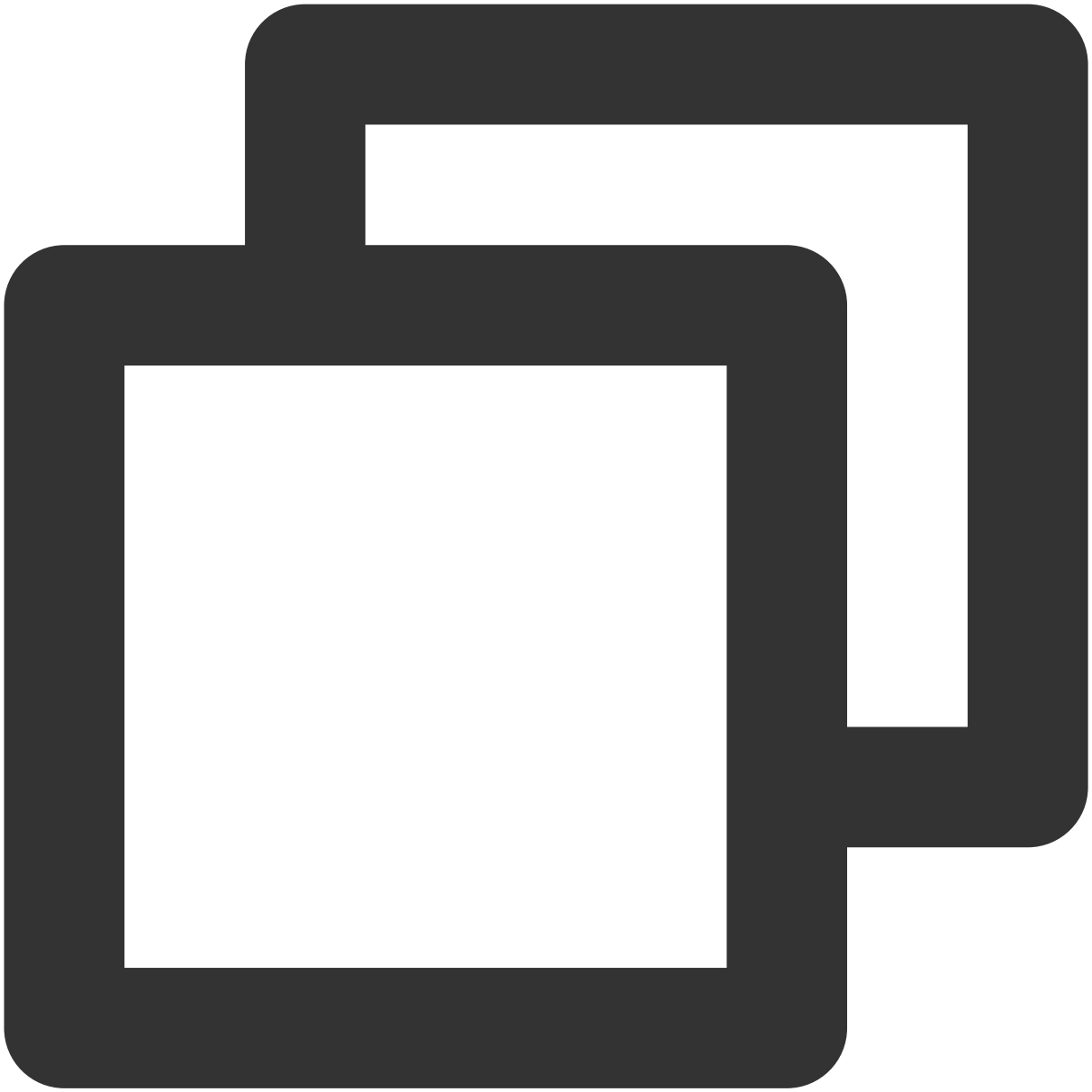
```
# npm 6.x
npm create vite@latest TUIRoom-demo --template vue-ts

# npm 7+, extra double-dash is needed:
npm create vite@latest TUIRoom-demo -- --template vue-ts

# yarn
yarn create vite TUIRoom-demo --template vue-ts

# pnpm
pnpm create vite TUIRoom-demo --template vue-ts
```

注意：

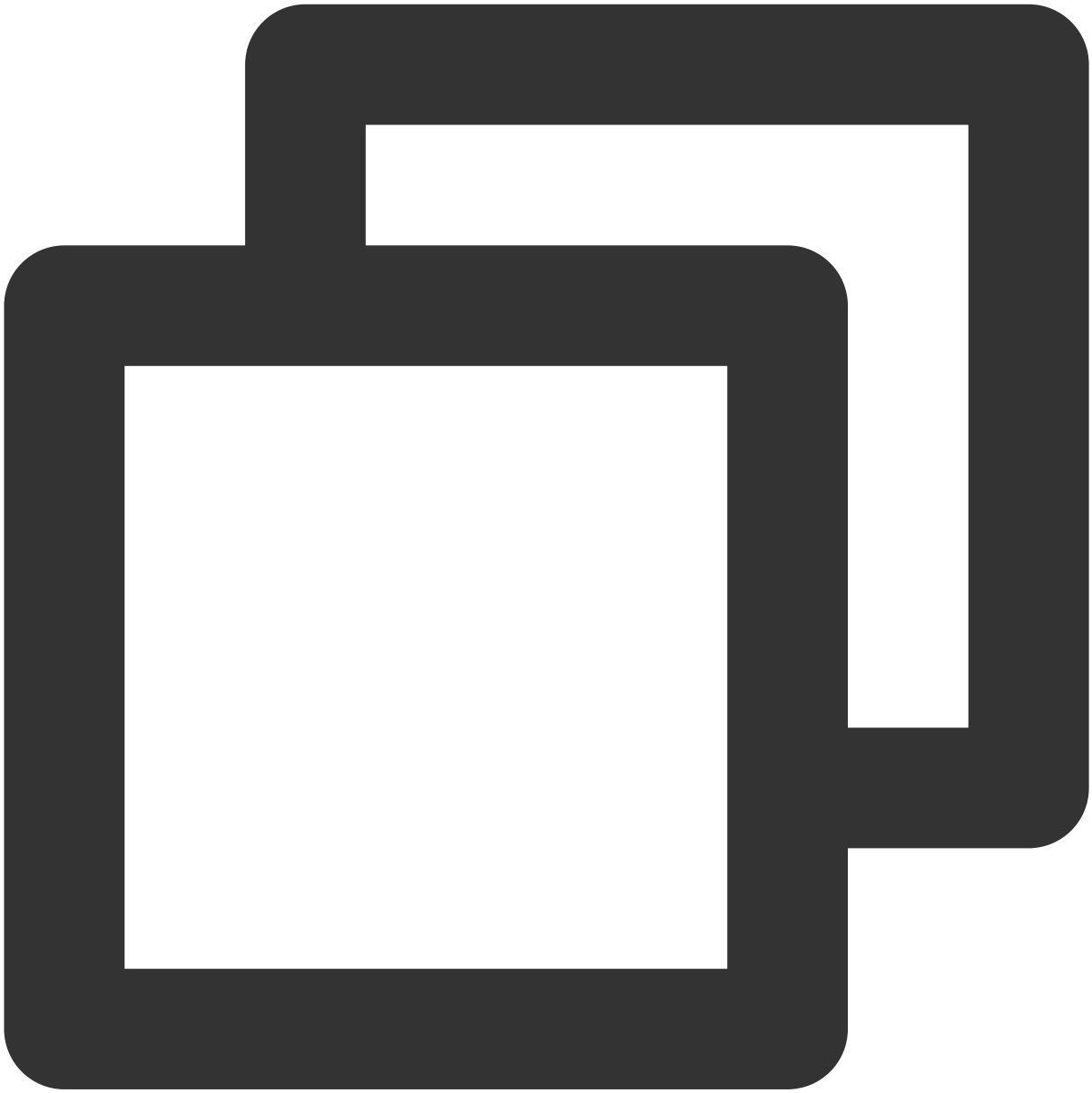
以 npm6.x 的创建方式为例，生成模板工程操作流程图如下：（其中标记高亮区域为需要输入或选择）



成功生成 Vue3 + Vite + TS 模板工程后，执行以下脚本：



```
cd TUIRoom-demo
npm install
npm run dev
```



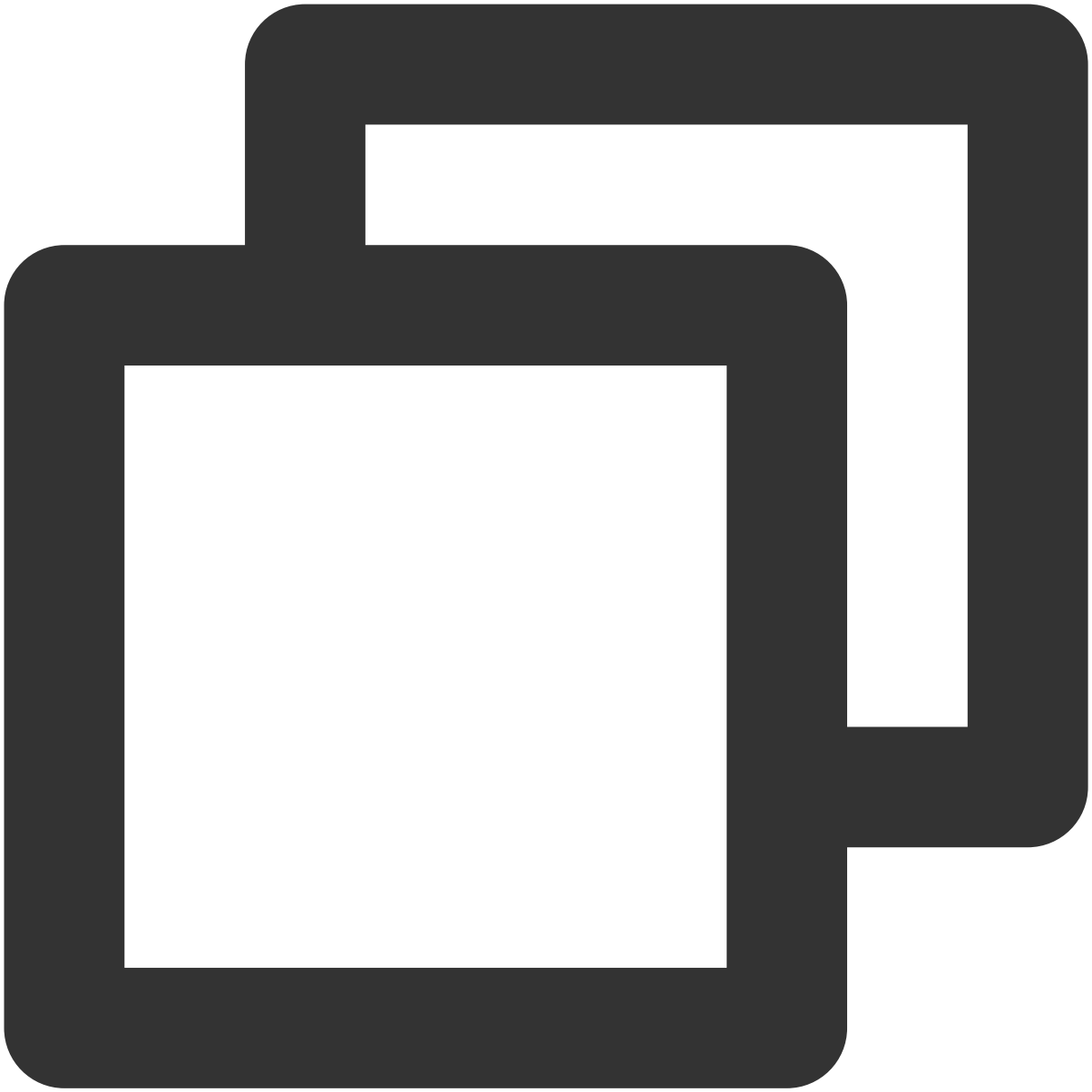
```
// 安装 vue-cli，注意 Vue CLI 4.x 要求 Node.js 为 v10 以上版本
npm install -g @vue/cli
// 创建 Vue3 + Webpack + TS 模版工程
vue create tuiroomkit-demo
```

注意：

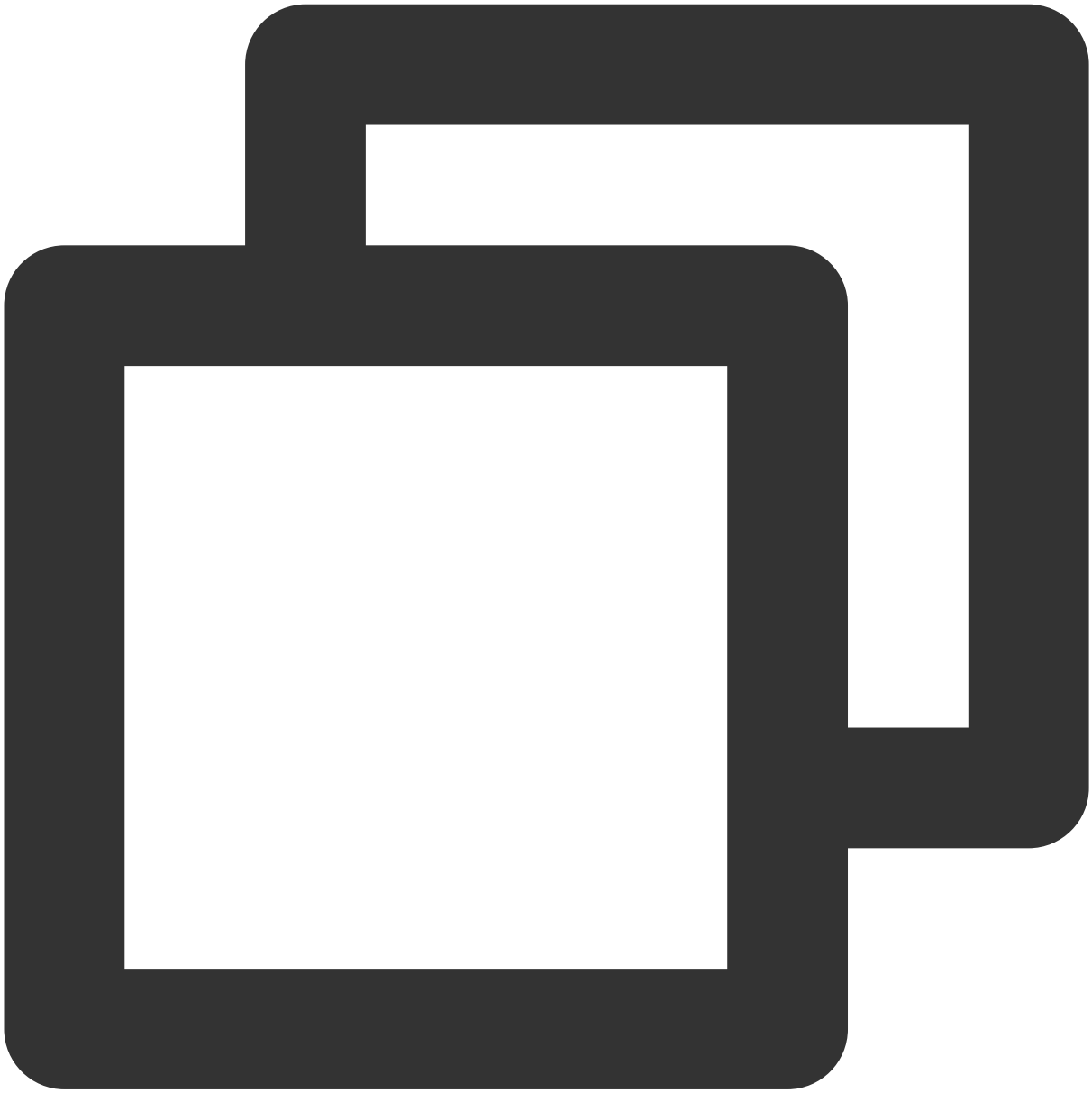
执行生成模板工程脚本的过程中，生成模版的方式选择 Manually select features，其余配置选项参考图片。

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected
polyfills, transpiling JSX)? Yes
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated
config files
? Save this as a preset for future projects? No
```

成功生成 Vue3 + Webpack + TS 模板工程后，执行以下脚本：



```
cd tuiroomkit-demo
npm run serve
```



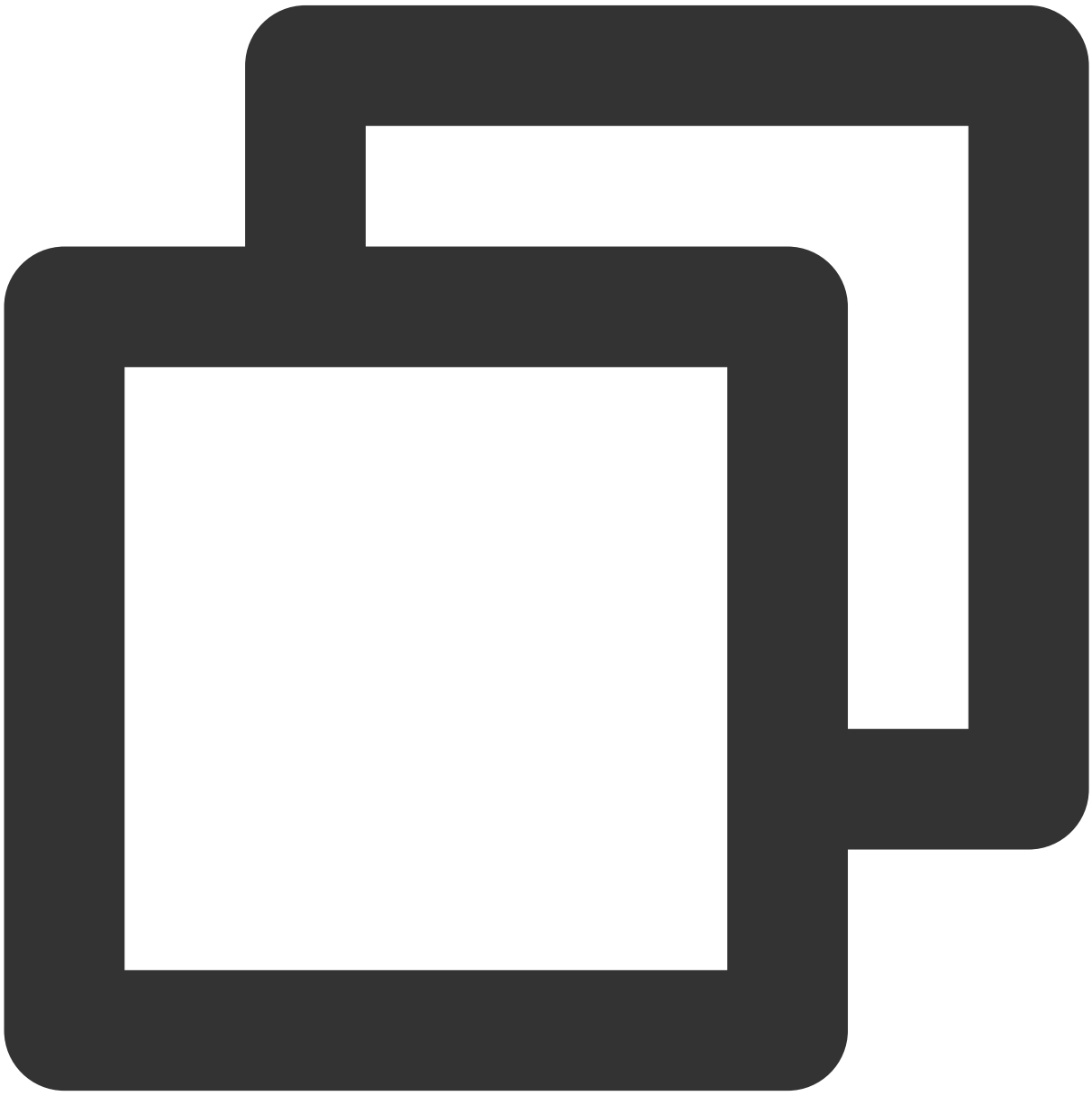
```
// 安装 vue-cli，注意 Vue CLI 4.x 要求 Node.js 为 v10 以上版本
npm install -g @vue/cli
// 创建 Vue2 + Webpack + Js 模版工程
vue create tuiroomkit-demo
```

注意：

执行生成模版工程脚本的过程中，生成模版的方式选择 `Default ([Vue 2] babel, eslint)` 。

```
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
❯ Default ([Vue 2] babel, eslint)
  Manually select features
```

成功生成 Vue2 + Webpack + JS 模版工程后，执行以下脚本：



```
cd tuiroomkit-demo
npm run serve
```

步骤三：下载并引用 TUIRoom 组件

1. 下载 TUIRoom 组件代码

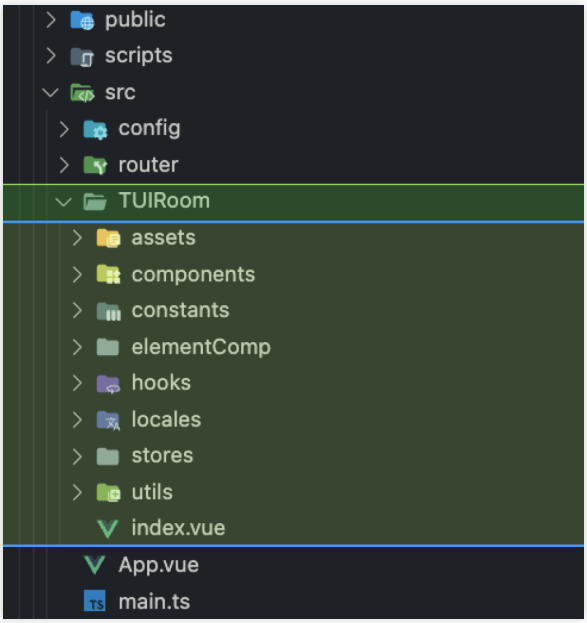
单击 Github , 克隆或下载 TUIRoomKit 仓库代码。

2. 引用 TUIRoom 组件

Vue3 项目引入 TUIRoom 组件

Vue2 项目引入 TUIRoom 组件

复制 `TUIRoomKit/Web/vue3/src/TUIRoom` 文件夹到已有项目 src/ 目录下，引用成功后目录结果如图所示。



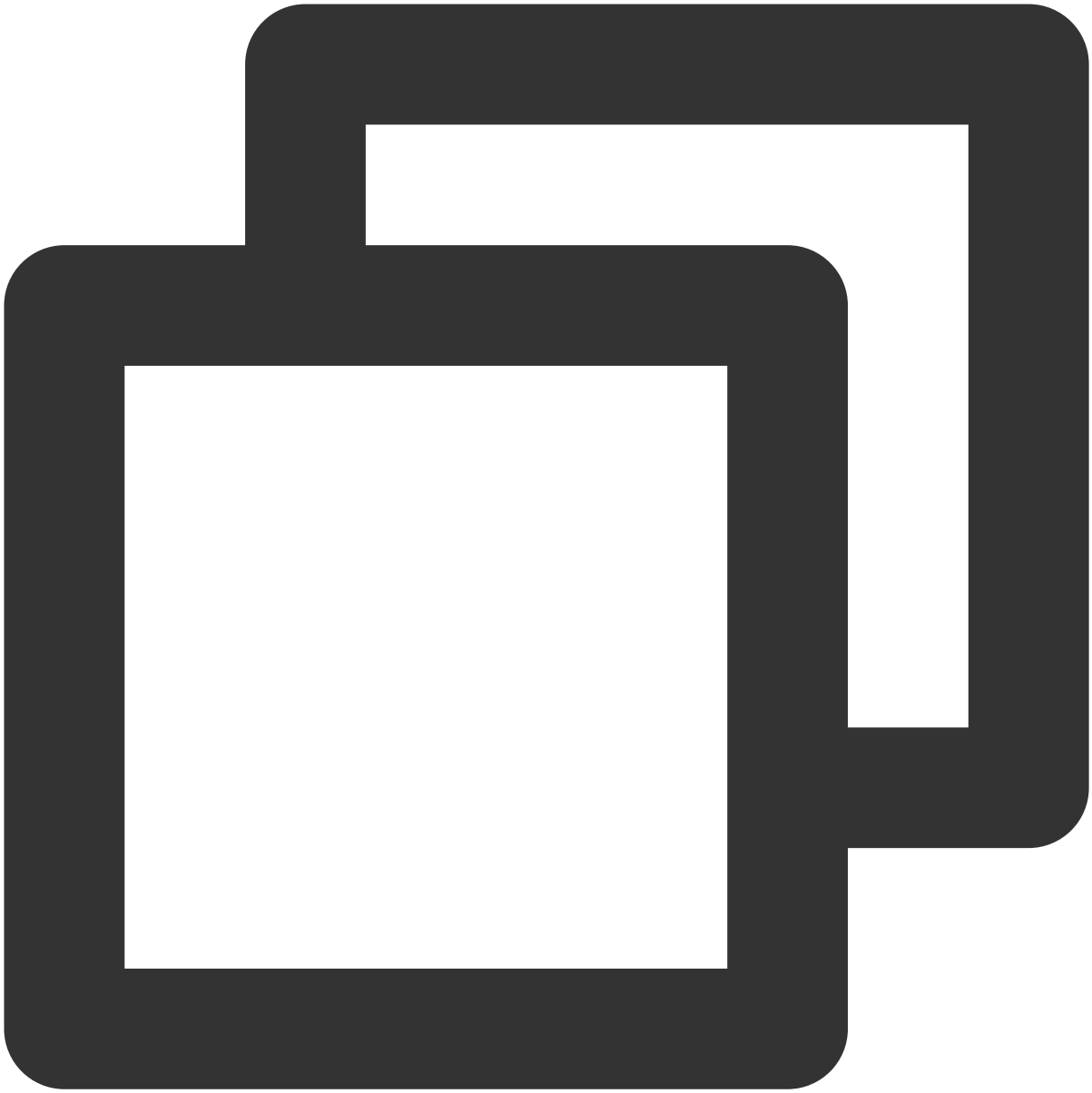
在页面中引用 TUIRoom 组件。例如：在 `App.vue` 组件中引入 TUIRoom 组件。

TUIRoom 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。

主持人及普通成员可通过 init 方法向 TUIRoom 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。

注意：

在页面中复制以下代码之后，需要修改 TUIRoom 接口的参数为实际数据。



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// 引入 TUIRoom 组件，注意确认引入路径是否正确
import Room from './TUIRoom/index.vue';
// 获取 TUIRoom 组件元素，用于调用 TUIRoom 组件的方法
const TUIRoomRef = ref();
```

```
onMounted(async () => {
  // 初始化 TUIRoom 组件
  // 主持人在创建房间前需要先初始化 TUIRoom 组件
  // 普通成员在进入房间前需要先初始化 TUIRoom 组件
  await TUIRoomRef.value.init({
    // 获取 sdkAppId 请您参考 步骤一
    sdkAppId: 0,
    // 用户在您业务中的唯一标示 Id
    userId: '',
    // 本地开发调试可在 https://console.trtc.io/usersig 页面快速生成 userSig，注意 userSi
    userSig: '',
    // 用户在您业务中使用的昵称
    userName: '',
    // 用户在您业务中使用的头像链接
    avatarUrl: '',
    // 用户在您业务中需要的皮肤主题颜色及是否支持切换主题皮肤
    theme: {
      defaultTheme: 'black',
      isSupportSwitchTheme: true
     }
  })
  // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
  await handleCreateRoom();
})

// 主持人创建房间，该方法只在创建房间时调用
async function handleCreateRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模式) 两种
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomPar
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
async function handleEnterRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
```

```
    const roomId = '123456';
    const roomParam = {
      isOpenCamera: true,
      isOpenMicrophone: true,
    }
    try {
      await TUIRoomRef.value.enterRoom({ roomId, roomParam });
    } catch (error: any) {
      alert('TUIRoomKit.enterRoom error: ' + error.message);
    }
  }
  </script>

  <style lang="scss">
  html, body {
      width: 100%;
      height: 100%;
      margin: 0;
  }

  #app {
    width: 100%;
    height: 100%;
  }
  </style>
```

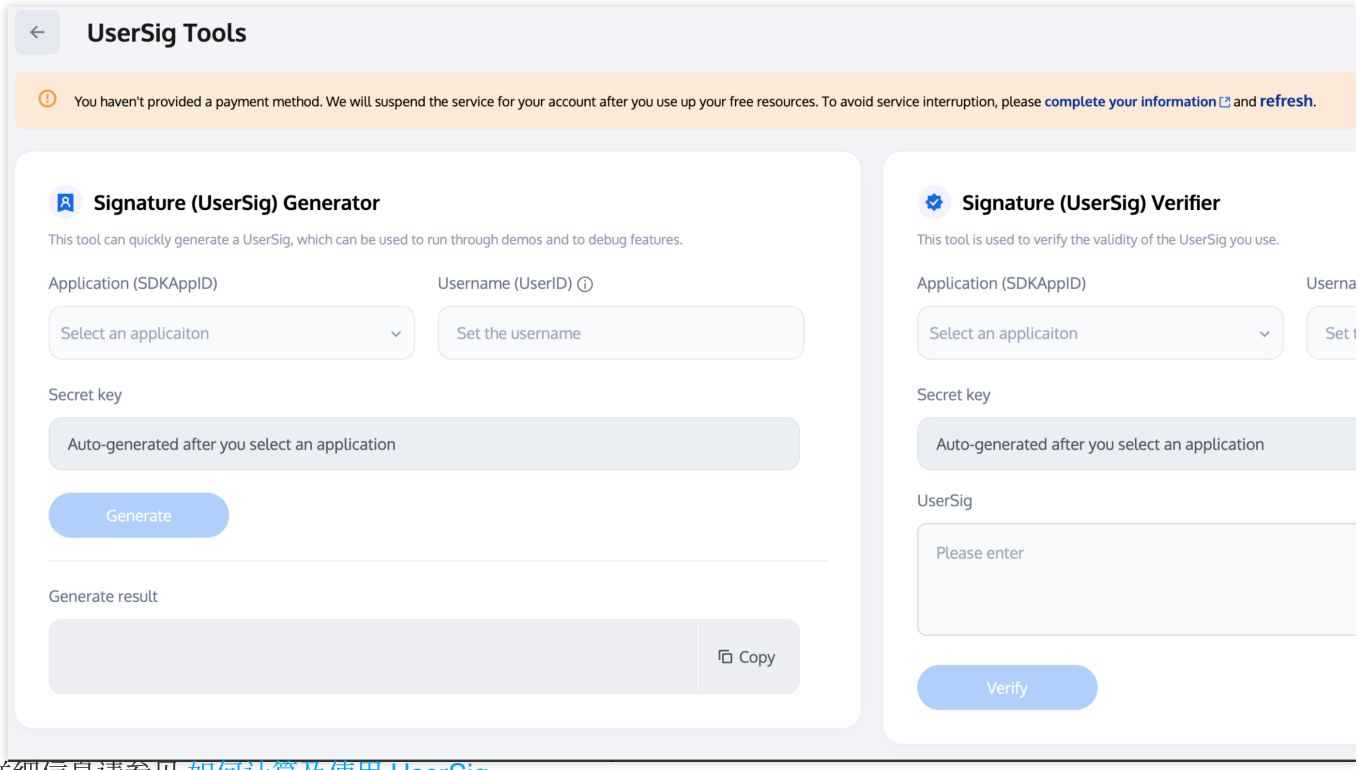
参数说明：

这里详细介绍一下 init 中所需要用到的几个关键参数：

sdkAppId：在步骤一中的最后一步中您已经获取到，这里不再赘述。

userId：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

userSig：使用 `SDKSecretKey` 对 `sdkAppId`、`userId` 等信息进行加密，就可以得到 `userSig`，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以访问实时音视频 TRTC 控制台 -> UserSig Tools, 得到一个临时可用的 userSig。



更多详细信息请参见 如何计算及使用 UserSig。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下:

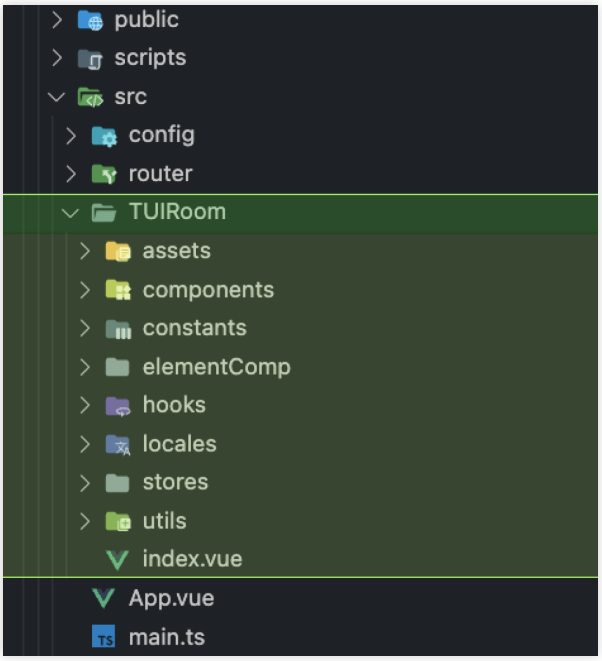
sdkAppId 设置错误。

userSig 被错配成了加密密钥（SDKSecretkey），userSig 是用 SDKSecretkey 把 sdkAppId、userId 以及过期时间等信息加密得来的，而不是直接把 SDKSecretkey 配置成 userSig。

UserSig 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 userId 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 userId。

Github 中的示例代码使用了 genTestUserSig 函数在本地计算 userSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SDKSecretkey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SDKSecretkey，所以我们强烈建议您将 UserSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUIRoomKit 组件时向您的服务器请求实时计算出的 userSig。

复制 `TUIRoomKit/Web/vue2/src/TUIRoom` 文件夹到已有项目 src/ 目录下，引用成功后目录结果如图所示：



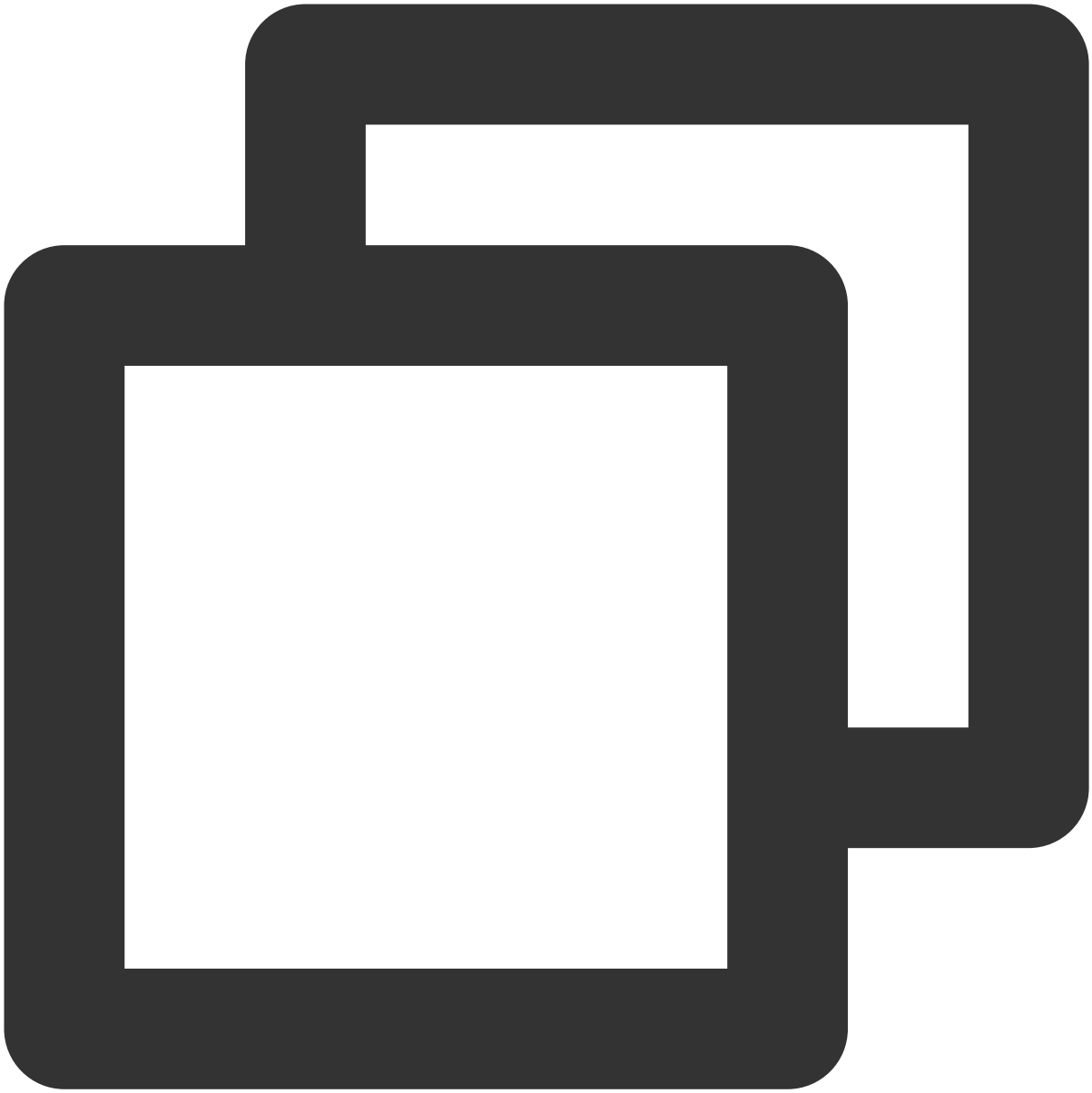
在页面中引用 TUIRoom 组件。例如：在 `App.vue` 组件中引入 TUIRoom 组件。

TUIRoom 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。

主持人及普通成员可通过 init 方法向 TUIRoom 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。

注意：

在页面中复制以下代码之后，需要修改 TUIRoom 接口的参数为实际数据。



```
<template>
  <div id="app">
    <room-container ref="TUIRoomRef"></room-container>
  </div>
</template>

<script>
import RoomContainer from '@/TUIRoom/index.vue';
export default {
  name: 'App',
  components: { RoomContainer },
```

```
  data() {
    return {};
  },
  async mounted() {
    // 初始化 TUIRoom 组件
    // 主持人在创建房间前需要先初始化 TUIRoom 组件
    // 普通成员在进入房间前需要先初始化 TUIRoom 组件
    await this.$refs.TUIRoomRef.init({
      // 获取 sdkAppId 请您参考 步骤一
      sdkAppId: 0,
      // 用户在您业务中的唯一标示 Id
      userId: '',
      // 本地开发调试可在 https://console.trtc.io/usersig 页面快速生成 userSig，注意 user
      userSig: '',
      // 用户在您业务中使用的昵称
      userName: '',
      // 用户在您业务中使用的头像链接
      avatarUrl: '',
    });
    // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
    await this.handleCreateRoom();
  },
  methods: {
    // 主持人创建房间，该方法只在创建房间时调用
    async handleCreateRoom() {
      // roomId 为用户进入的房间号，要求 roomId 类型为 string
      // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模式)
      // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒
      const roomId = '123456';
      const roomMode = 'FreeToSpeak';
      const roomParam = {
          isOpenCamera: true,
          isOpenMicrophone: true,
      }
      try {
        await this.$refs.TUIRoomRef.createRoom({ roomId, roomName: roomId, roomMode
      } catch (error) {
        alert('TUIRoomKit.createRoom error: ' + error.message);
      }
    },
    // 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
    async handleEnterRoom() {
      // roomId 为用户进入的房间号，要求 roomId 类型为 string
      // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒
      const roomId = '123456';
      const roomParam = {
          isOpenCamera: true,
```

```
            isOpenMicrophone: true,
        }
        try {
          await this.$refs.TUIRoomRef.enterRoom({ roomId, roomParam });
        } catch (error) {
          alert('TUIRoomKit.enterRoom error: ' + error.message);
        }
      }
    },
};

</script>

<style lang="scss">
html, body {
  width: 100%;
  height: 100%;
  margin: 0;
}

#app {
  width: 100%;
  height: 100%;
  * {
    box-sizing: border-box;
  }
}
</style>
```

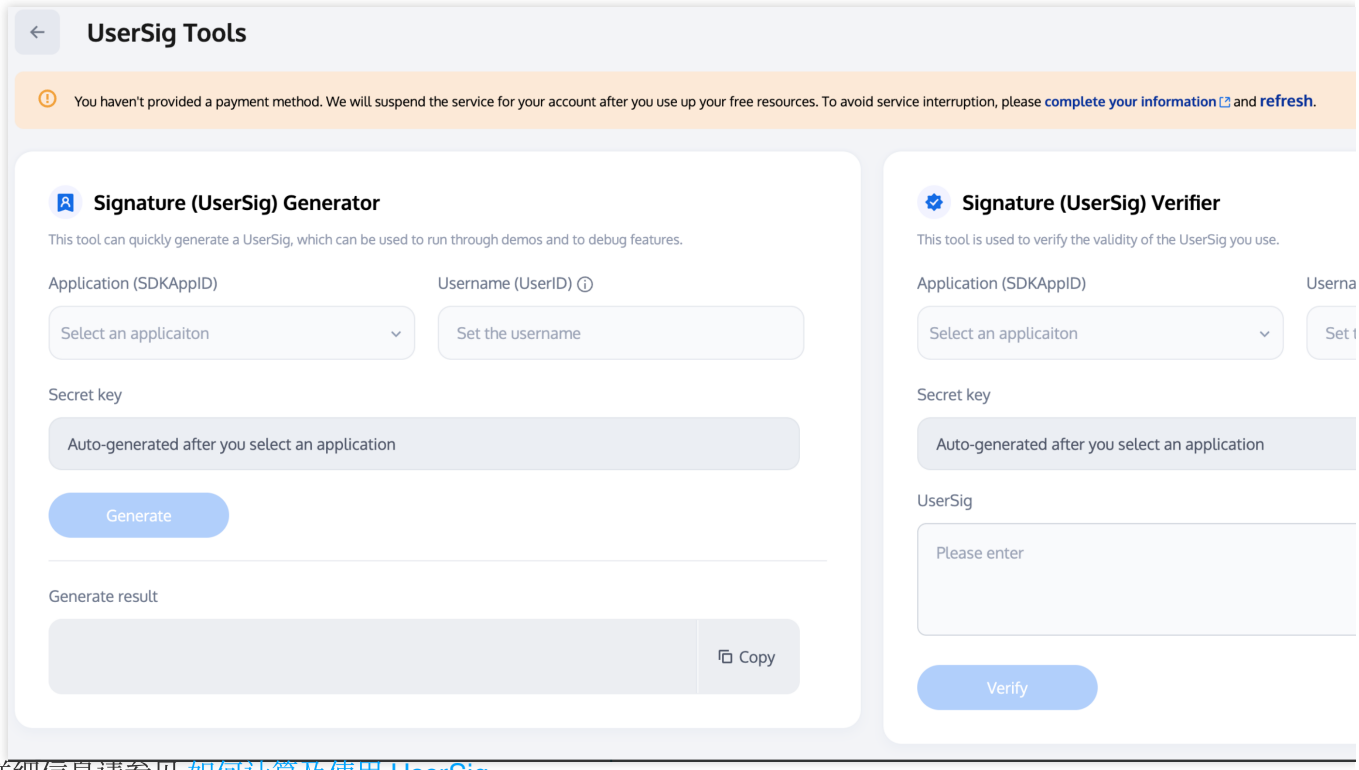
参数说明：

这里详细介绍一下 init 中所需要用到的几个关键参数：

sdkAppId：在步骤一中的最后一步中您已经获取到，这里不再赘述。

userId：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

userSig：使用 `SDKSecretKey` 对 `sdkAppId`、`userId` 等信息进行加密，就可以得到 `userSig`，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以访问实时音视频 TRTC 控制台 -> UserSig Tools, 得到一个临时可用的 userSig。



更多详细信息请参见 如何计算及使用 UserSig。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下:

sdkAppId 设置错误。

userSig 被错配成了加密密钥（Secretkey），userSig 是用 SecretKey 把 sdkAppId、userId 以及过期时间等信息加密得来的，而不是直接把 SecretKey 配置成 userSig。

UserSig 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 userId 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 userId。

Github 中的示例代码使用了 genTestUserSig 函数在本地计算 userSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SecretKey，所以我们强烈建议您将 UserSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUIRoomKit 组件时向您的服务器请求实时计算出的 userSig。

步骤四：配置开发环境

TUIRoom 组件引入之后，为了确保项目可以正常运行，需要进行以下配置：

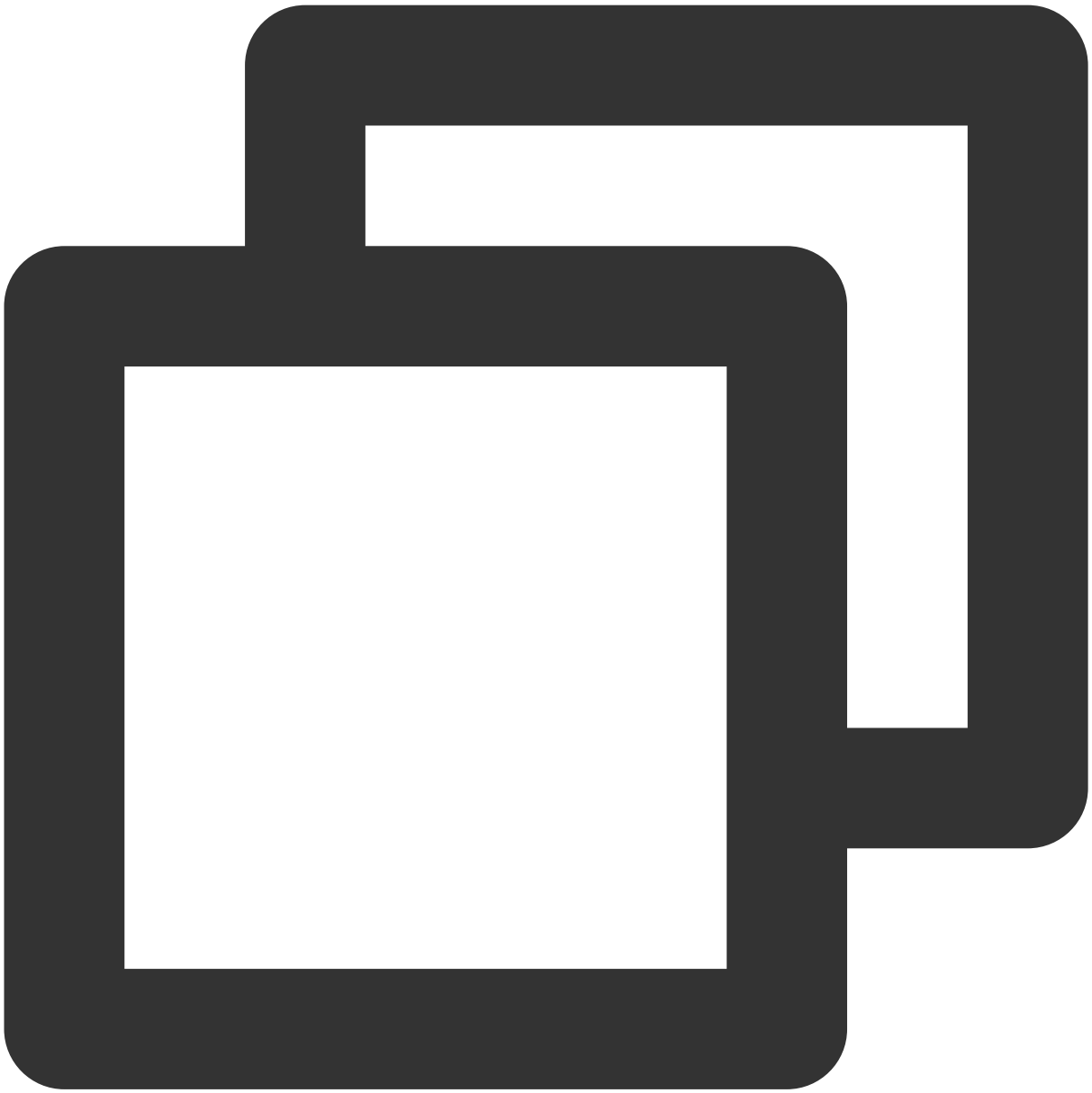
配置 Vue3 + Vite + TS 开发环境

配置 Vue3 + Webpack + TS 开发环境

配置 Vue2 + Webpack + TS 开发环境

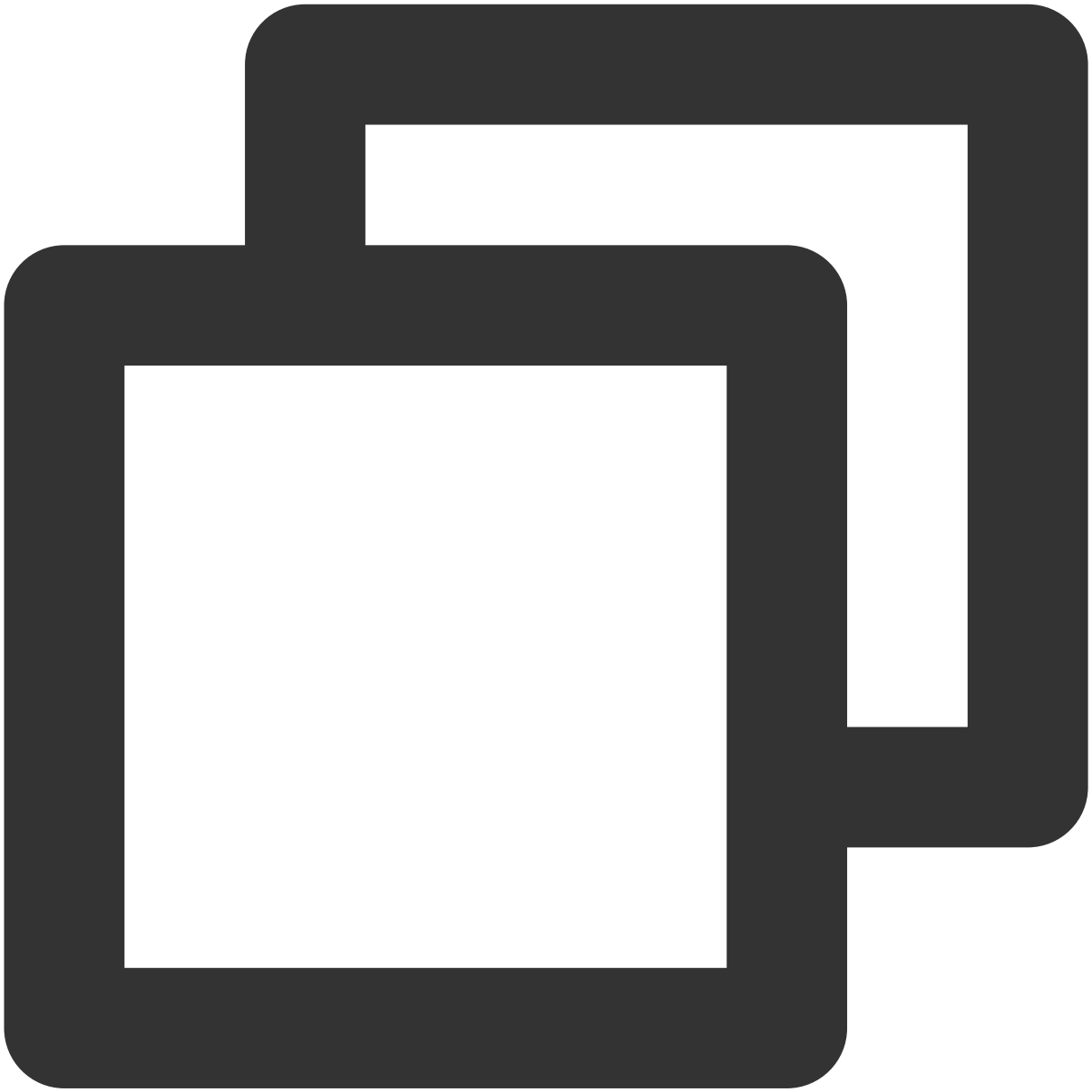
1. **安装依赖**

安装开发环境依赖：



```
npm install sass typescript -S -D
```

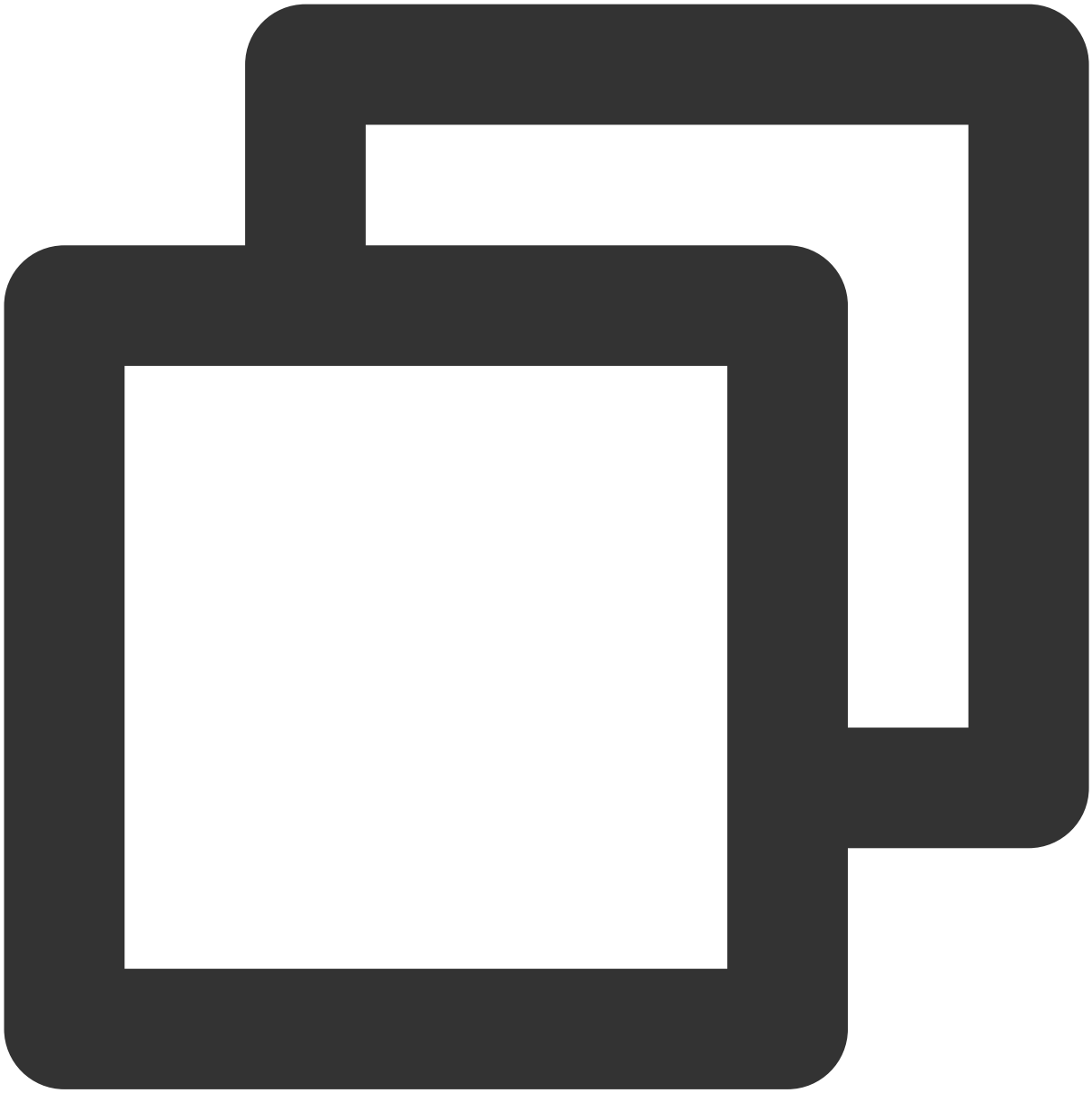
安装生产环境依赖：



```
npm install mitt pinia @tencentcloud/tuiroom-engine-js vue-i18n rtc-detect -S
```

2. **注册 Pinia**

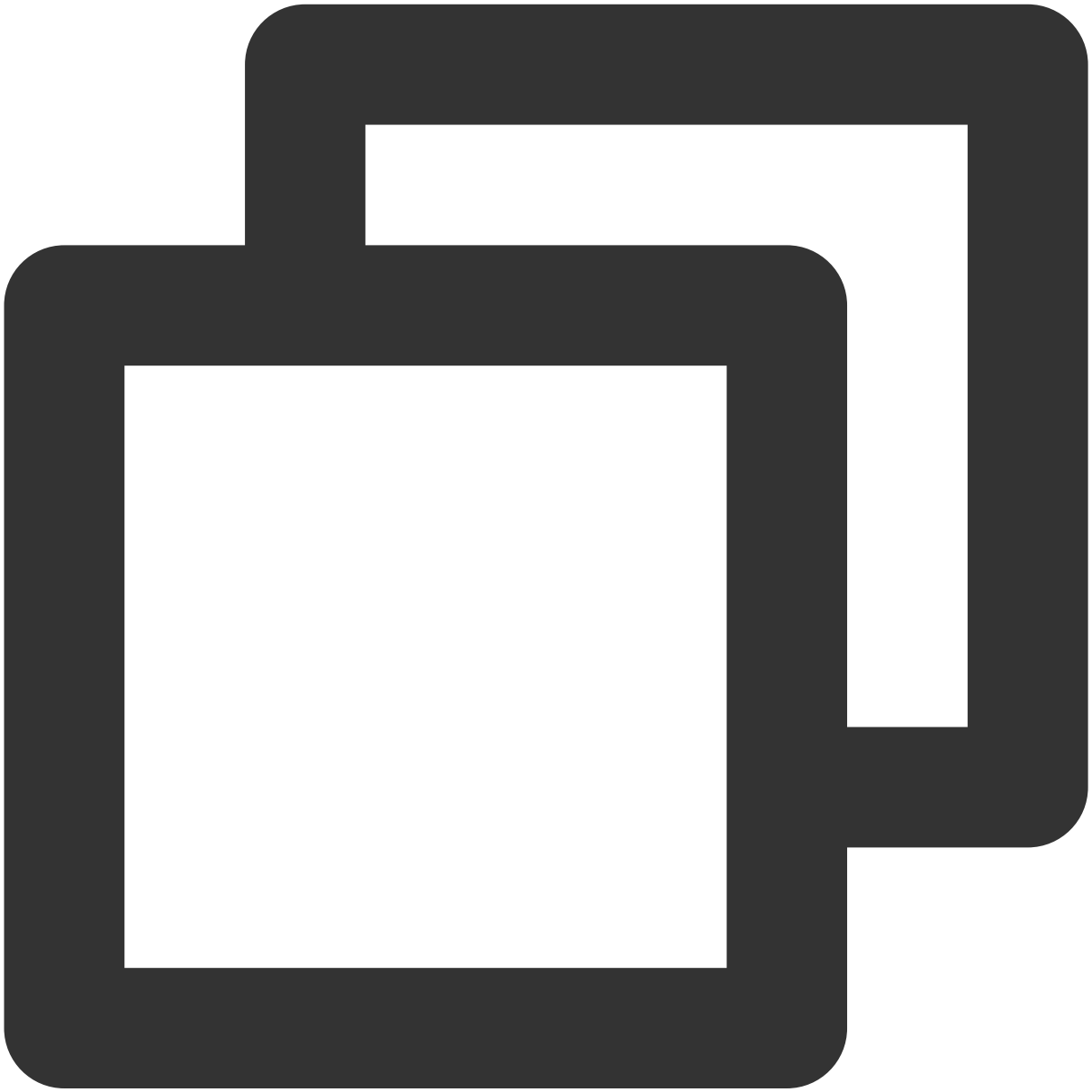
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 文件。



```
// src/main.ts 文件
import { createPinia } from 'pinia';

const app = createApp(App);
// 注册 pinia
app.use(createPinia());
app.mount('#app');
```

3. 配置中英文切换

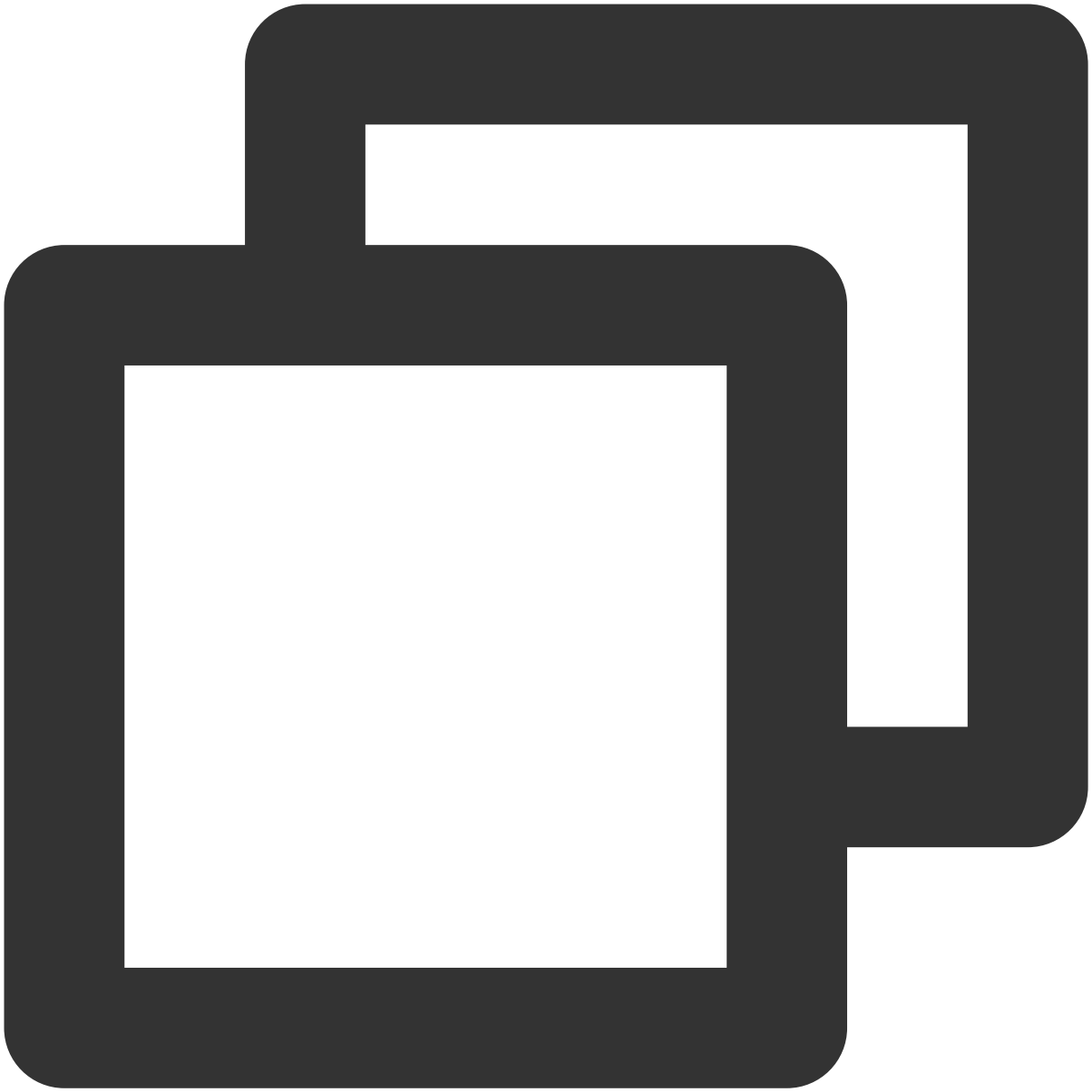
TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



```
// src/main.ts
// 引入 locales/index.ts 文件，请确认引入路径是否正确
import VueI18n from './TUIRoom/locales/index';

app.use(VueI18n);
```

4. **配置 esLint 校验**

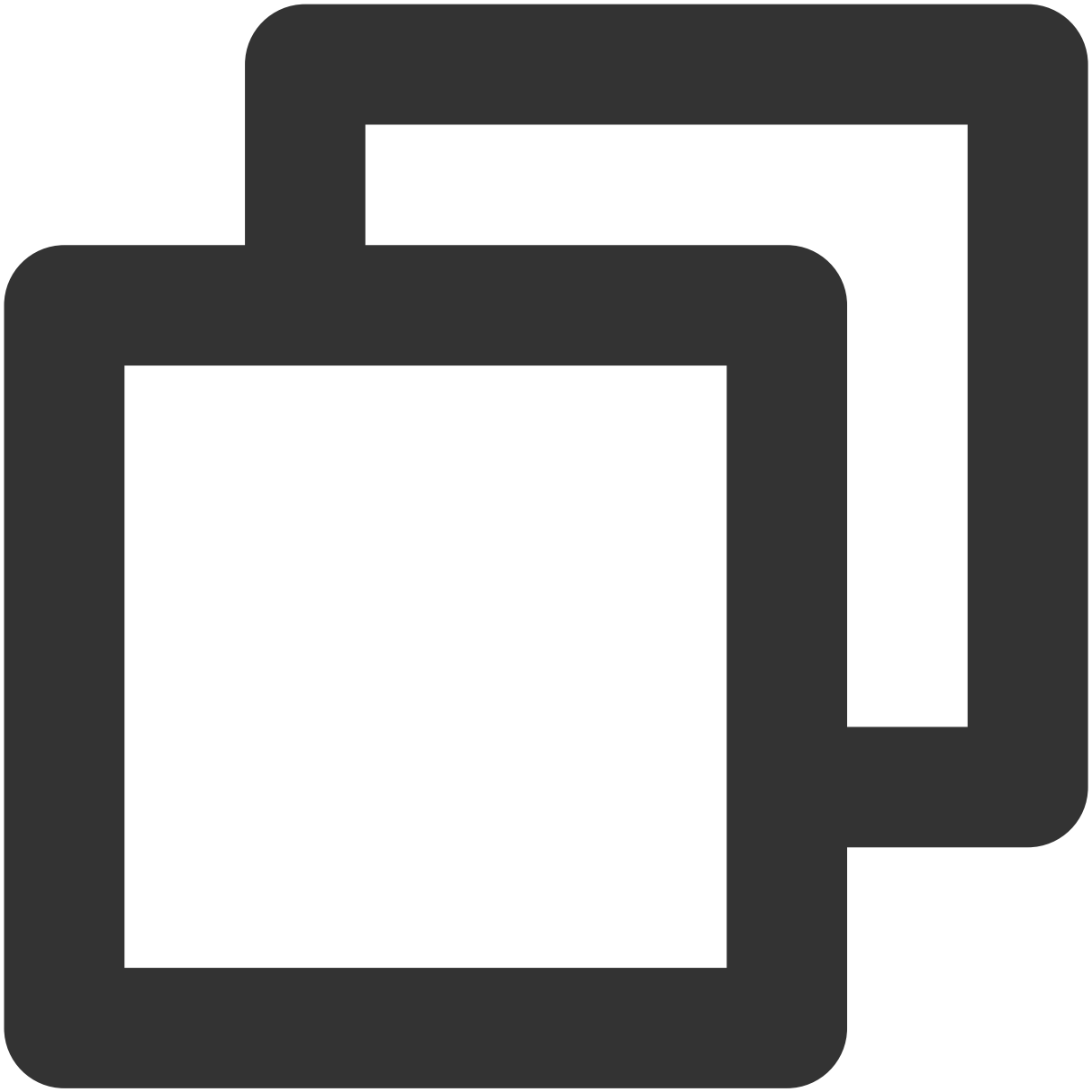
如果您不希望 TUIRoomKit 组件的 esLint 规则和您本地的规则出现冲突，您可以在 `.eslintignore` 中添加忽略 TUIRoom 文件夹。



```
src/TUIRoom
```

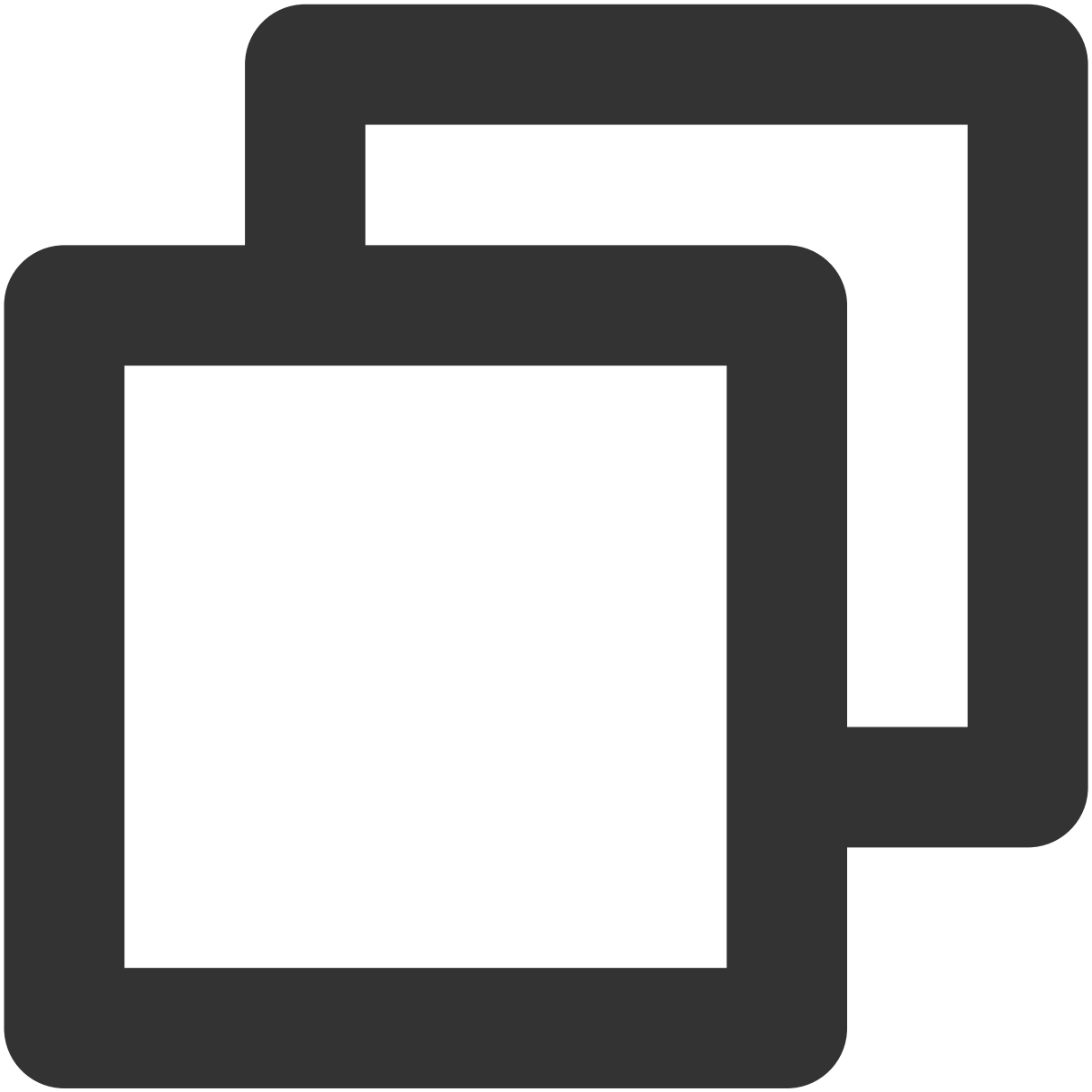
1. 安装依赖

安装开发环境依赖：



```
npm install sass sass-loader typescript -S -D
```

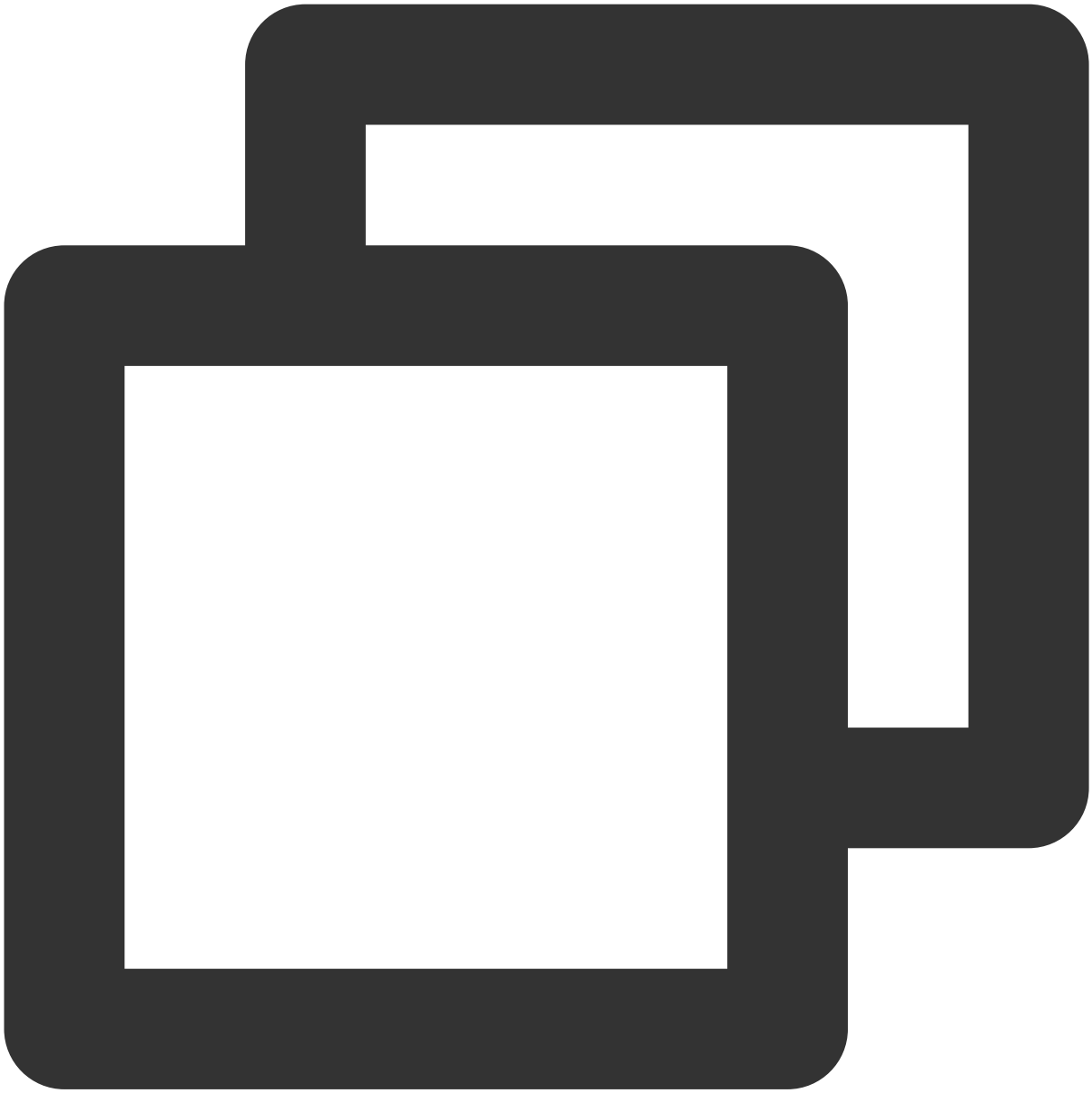
安装生产环境依赖：



```
npm install mitt pinia @tencentcloud/tuiroom-engine-js vue-i18n rtc-detect -S
```

2. **注册 Pinia**

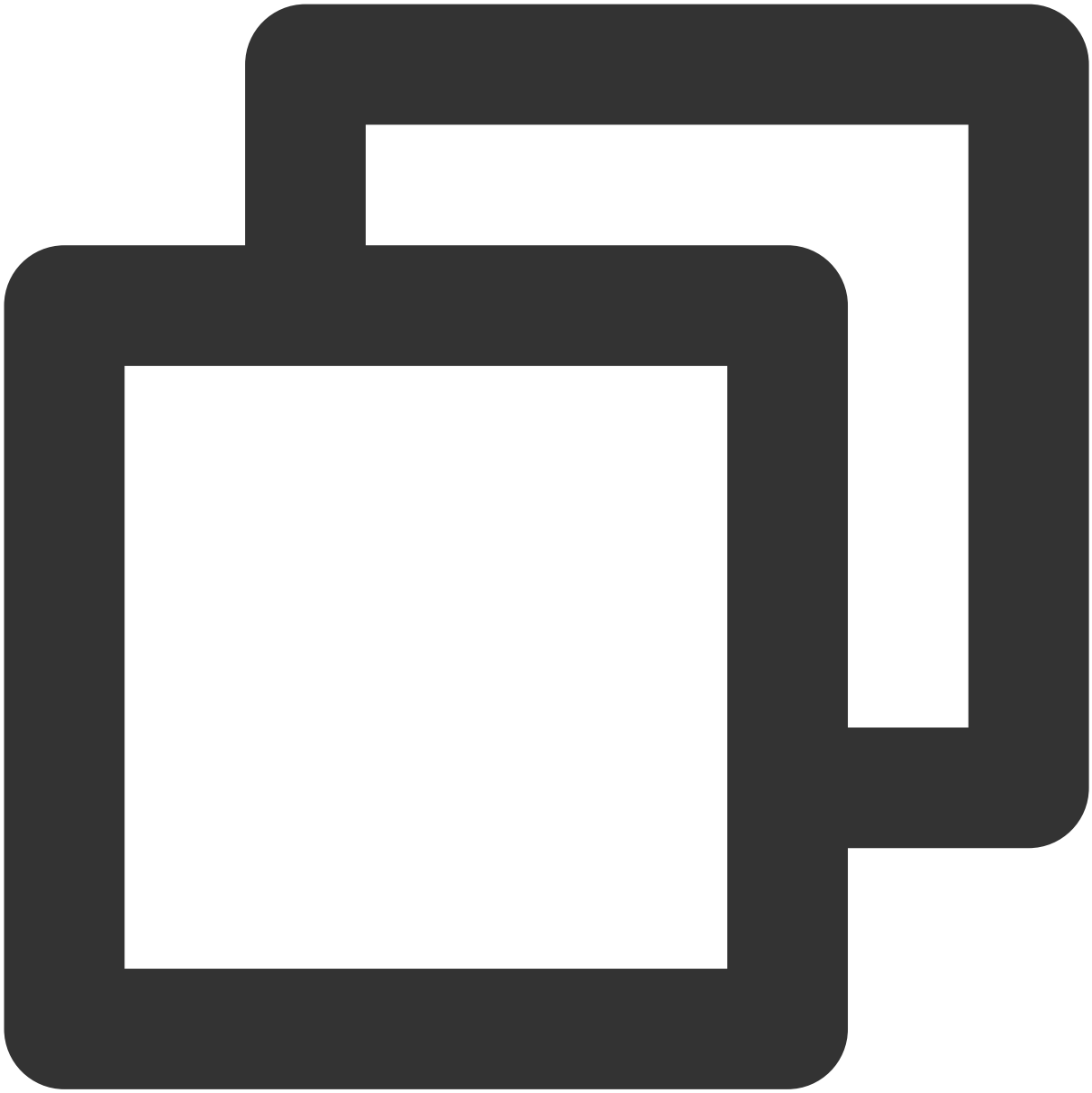
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 文件。



```
// src/main.ts 文件
import { createPinia } from 'pinia';

const app = createApp(App);
// 注册 pinia
app.use(createPinia());
app.mount('#app');
```

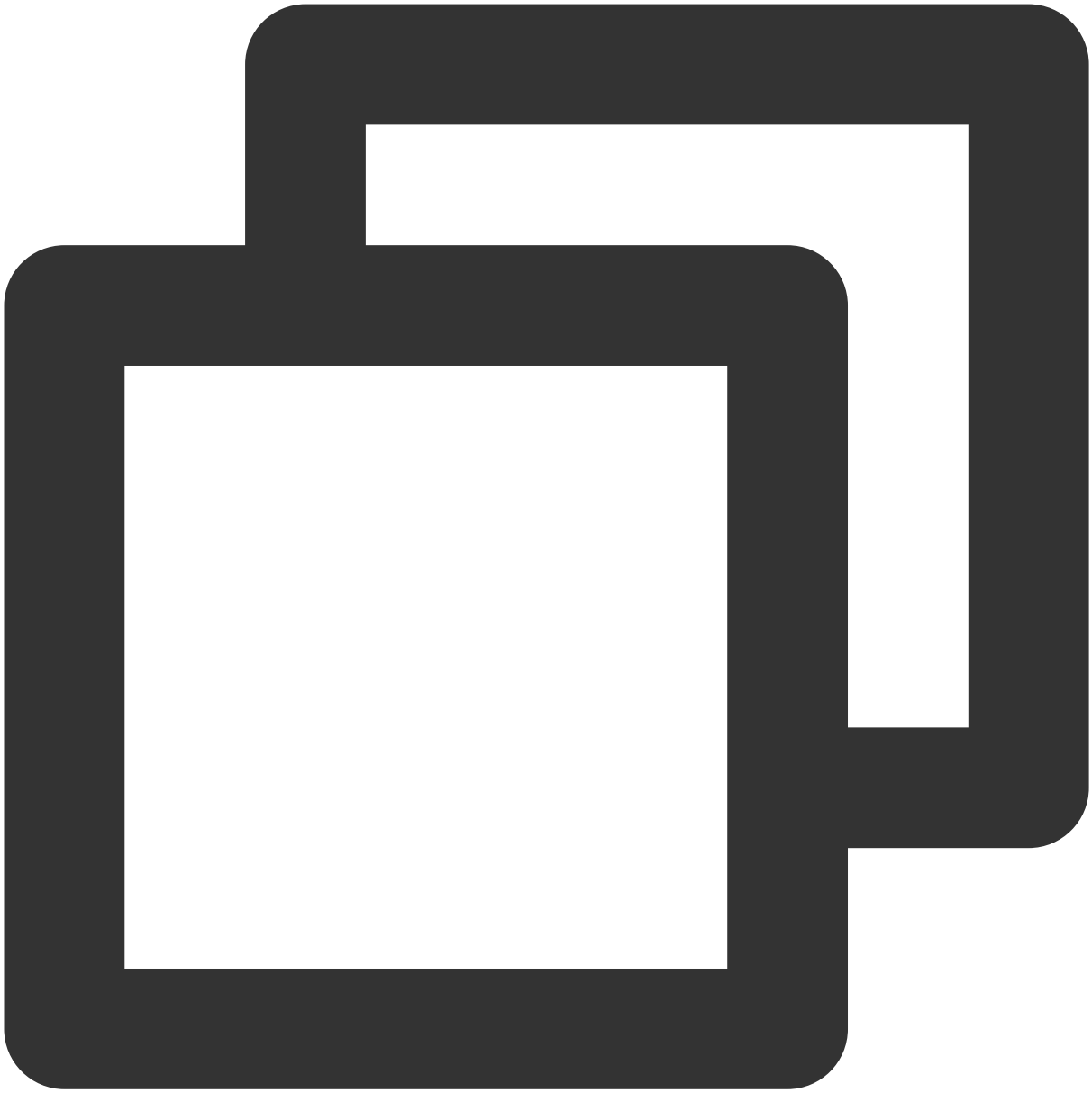
3. 配置中英文切换

TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



```
// src/main.ts
// 引入 locales/index.ts 文件，请确认引入路径是否正确
import VueI18n from './TUIRoom/locales/index';

app.use(VueI18n);
```

4. **配置 esLint 校验**



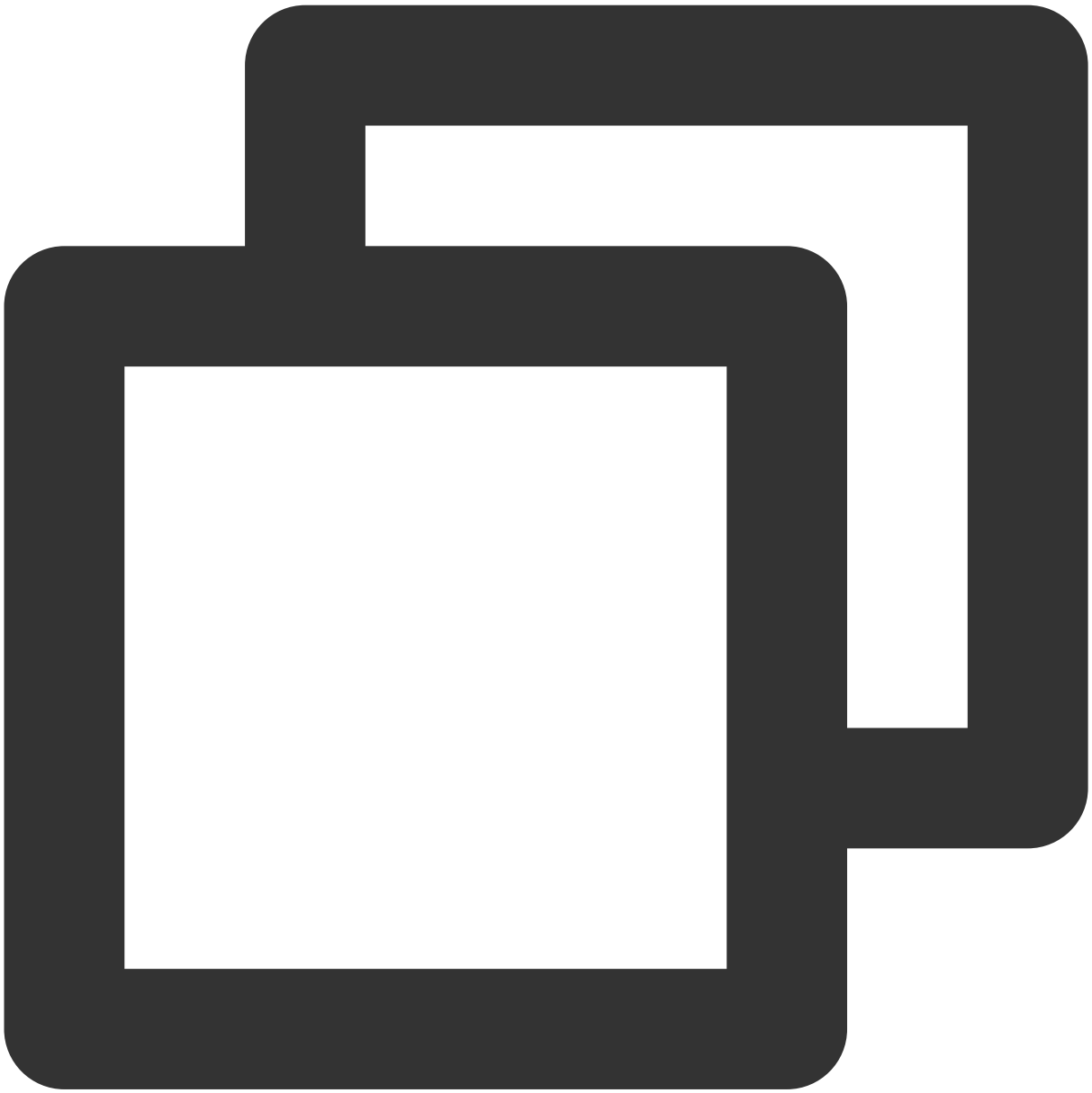
```
src/TUIRoom
src/App.vue
src/main.ts
```

说明：

TUIRoomKit 组件要求 vue2 项目工程安装 vue2.7 版本并支持 typescript 环境。如您当前项目为 vue2.6 + js 环境，按照以下步骤执行，可升级为 vue2.7 + ts 环境。

vue2.6 升级到 vue2.7 是平滑升级的，对已有代码无影响。配置 ts 环境之后，对已有 js 代码无影响。请放心升级。

1. **配置 typescript, 支持 TUIRoom 组件加载**



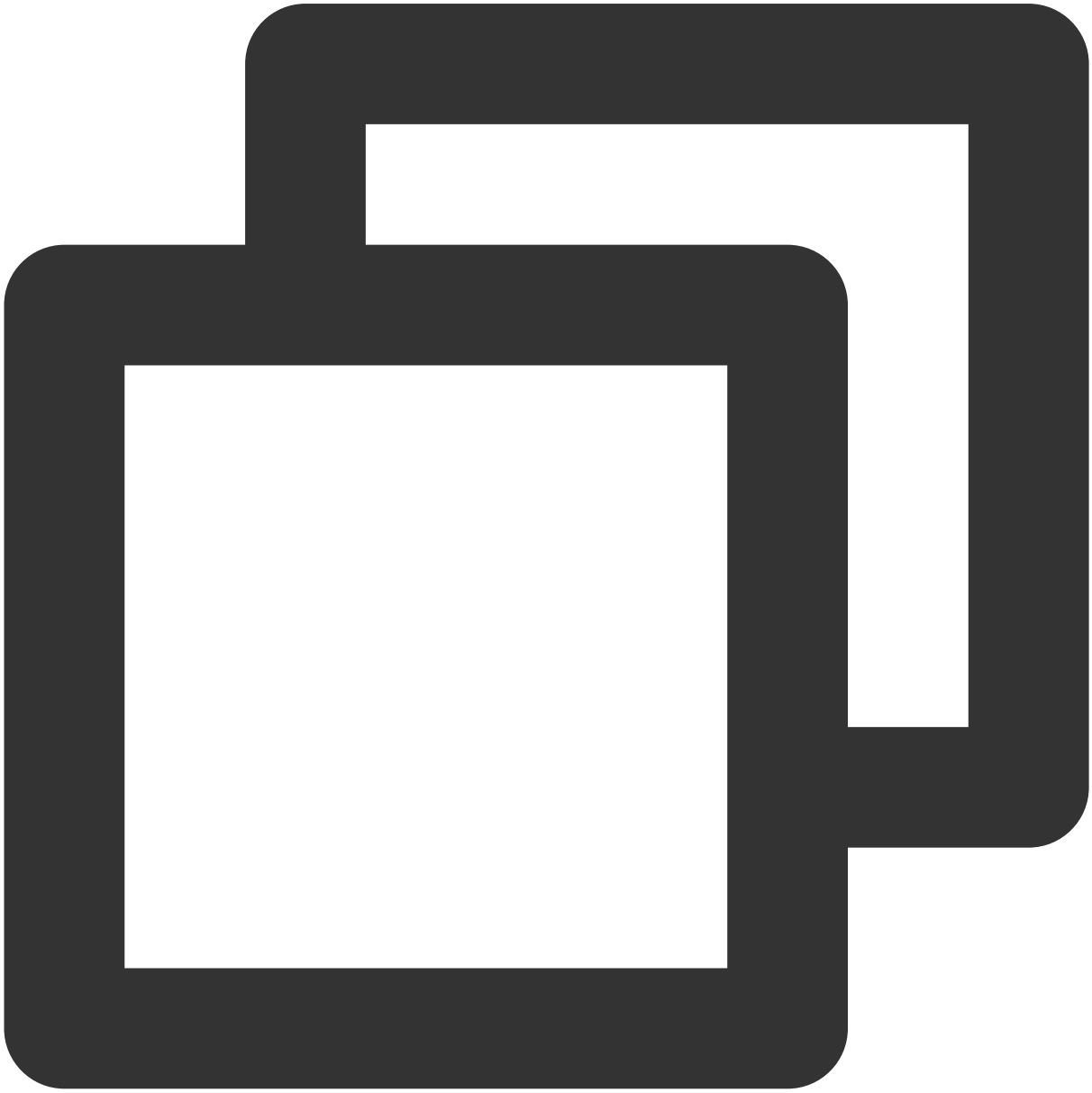
```
vue add typescript
```

配置 TS 开发环境的选项可参考图片：

```
? Use class-style component syntax? No
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? No
```

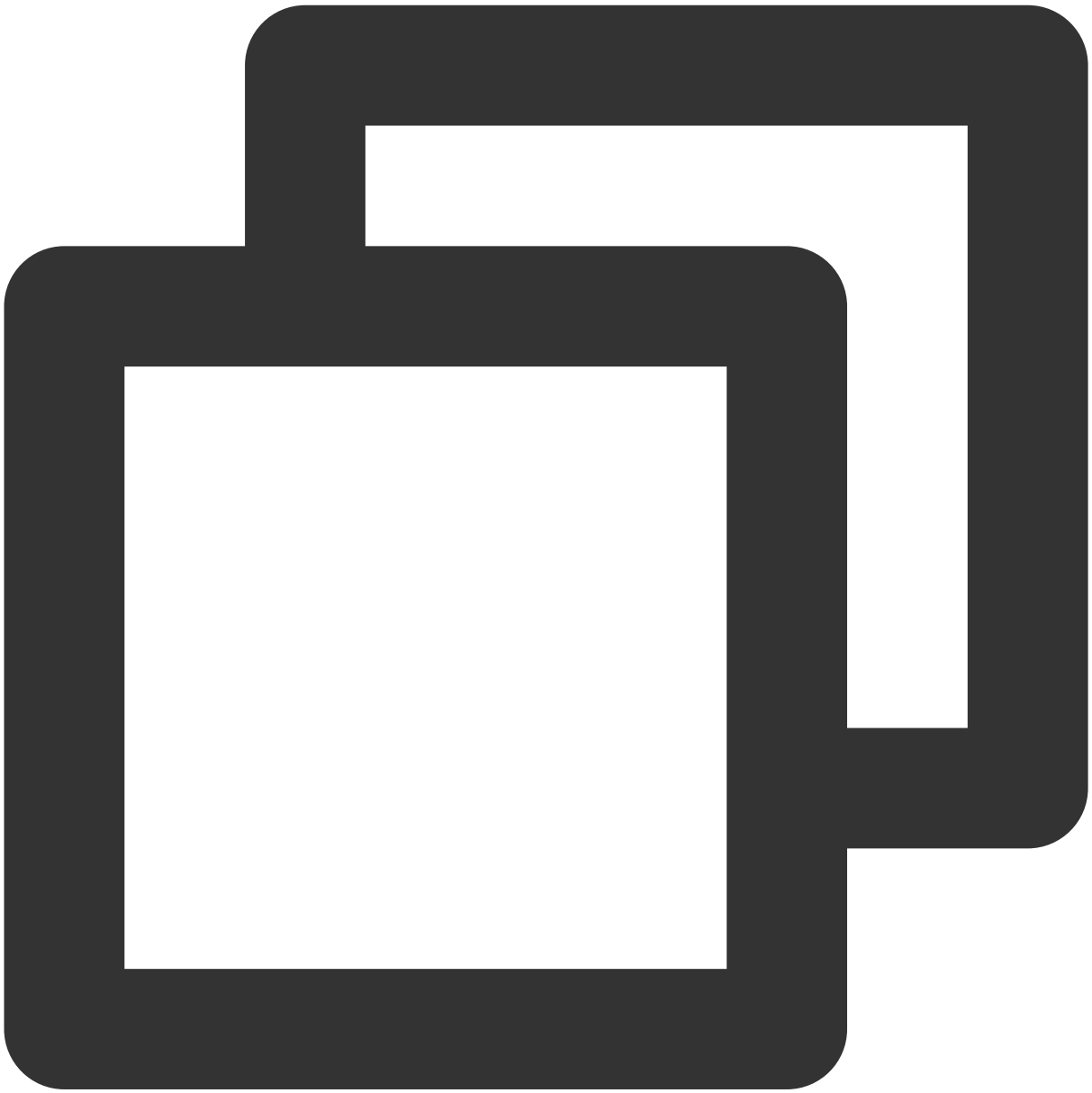
2. **安装依赖**

安装开发环境依赖：



```
npm install sass sass-loader -S -D
```

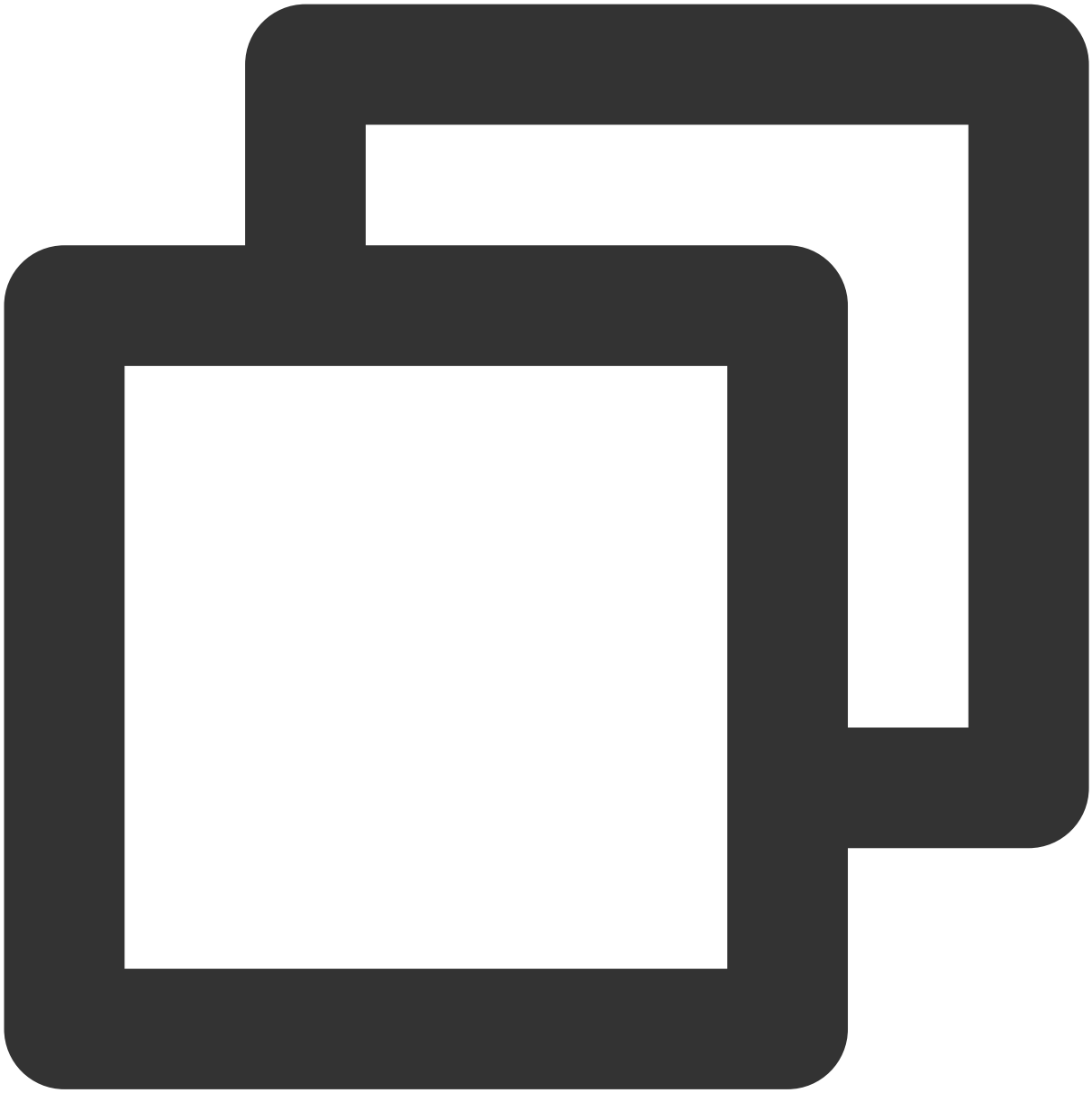
安装生产环境依赖：



```
npm install vue@^2.7 pinia @tencentcloud/tuiroom-engine-js vue-i18n@^8 vue-i18n-bri
```

3. 注册 Pinia

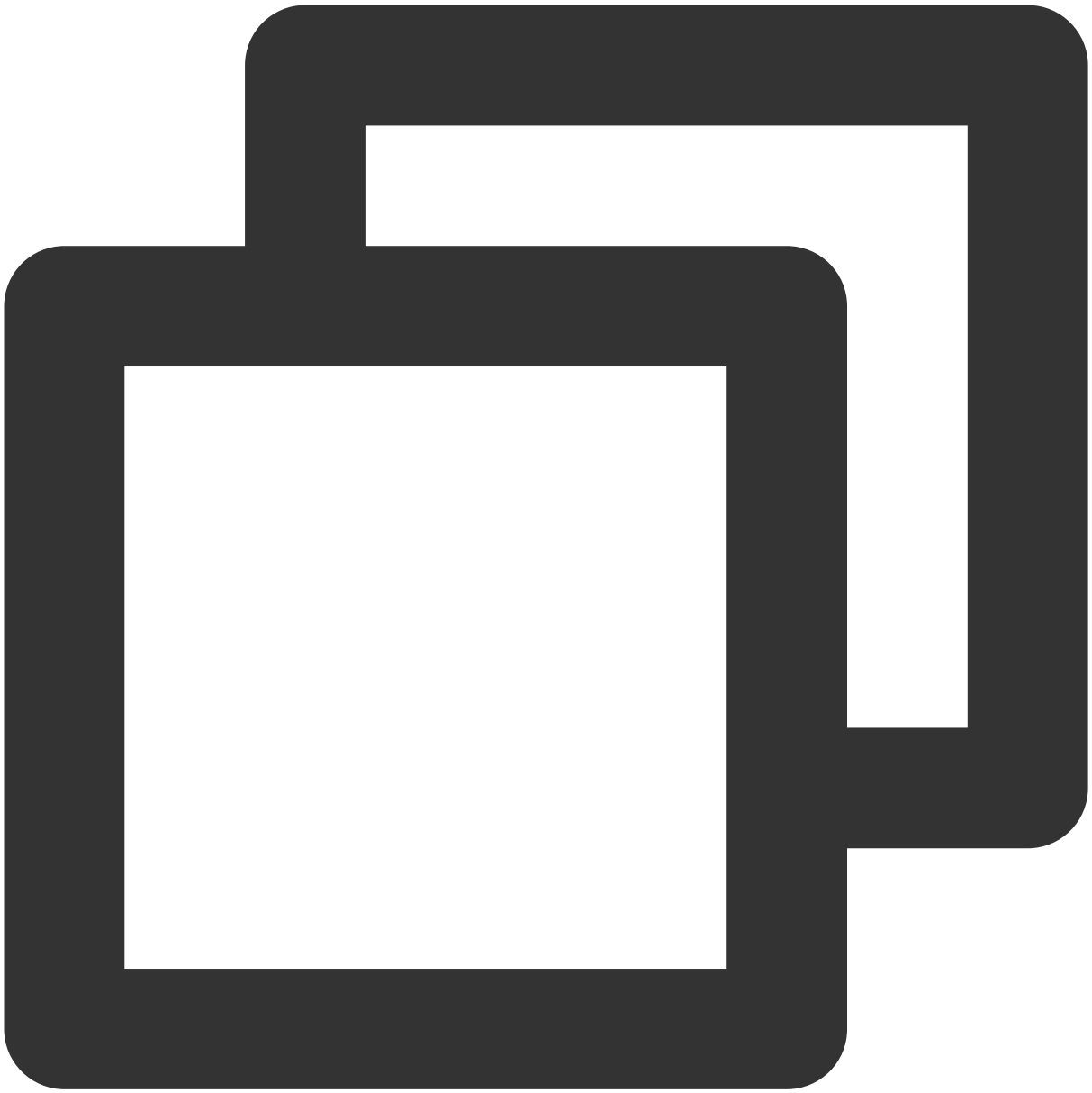
TUIRoom 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 或者 `src/main.js` 文件。



```
import { createPinia, PiniaVuePlugin } from 'pinia';

Vue.use(PiniaVuePlugin);
const pinia = createPinia();

new Vue({
  pinia,
  render: h => h(App),
}).$mount('#app');
```

4. 配置中英文切换

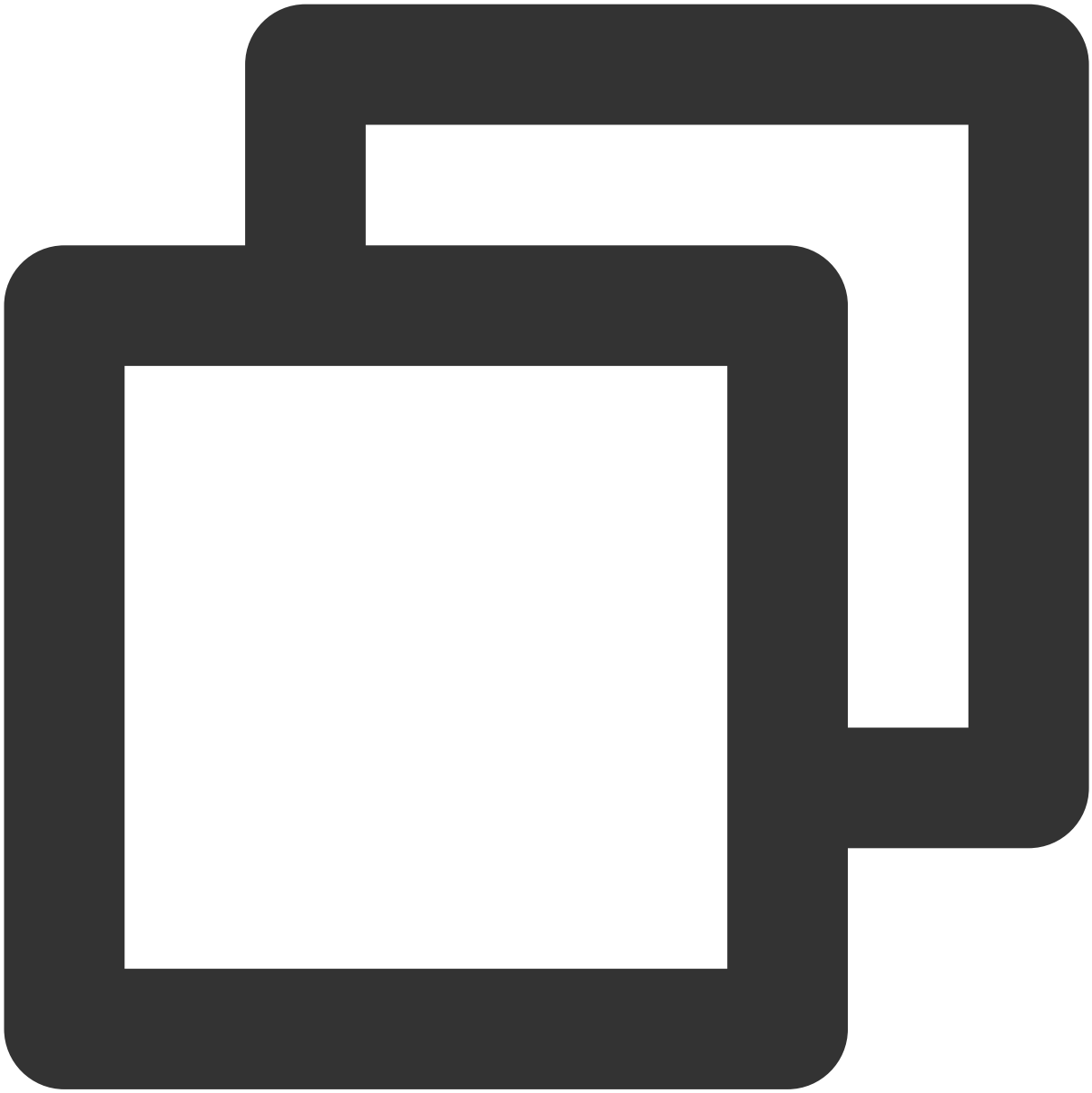
TUIRoom 目前支持中英文语言切换，需要您在 main.ts 文件中注册 i18n 实例。



```
import i18n from './TUIRoom/locales/';

Vue.use(i18n);
```

5. **配置 esLint 校验**

如果您不希望 TUIRoomKit 组件的 esLint 规则和您本地的规则出现冲突，您可以在 `.eslintignore` 中配置忽略 TUIRoom 相关文件。



```
src/TUIRoom
```

步骤五：开发环境运行

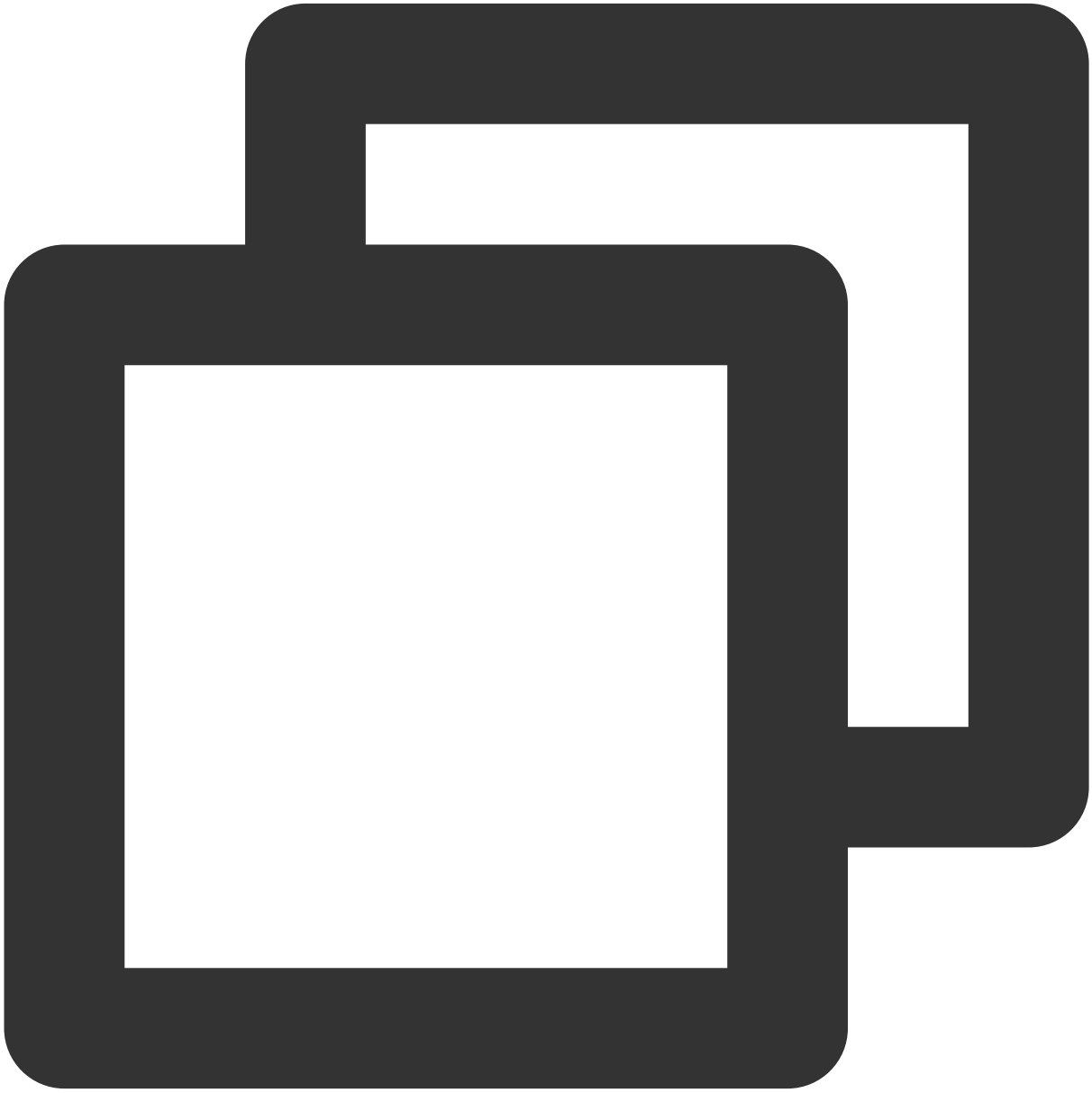
在控制台执行开发环境运行脚本，使用浏览器打开包含 TUIRoomKit 的页面，即可在页面中使用 TUIRoomKit 组件。

Vue3 + Vite + TS 项目

Vue3 + Webpack + TS 项目

Vue2 + Webapck + TS 项目

1. 执行开发环境命令。



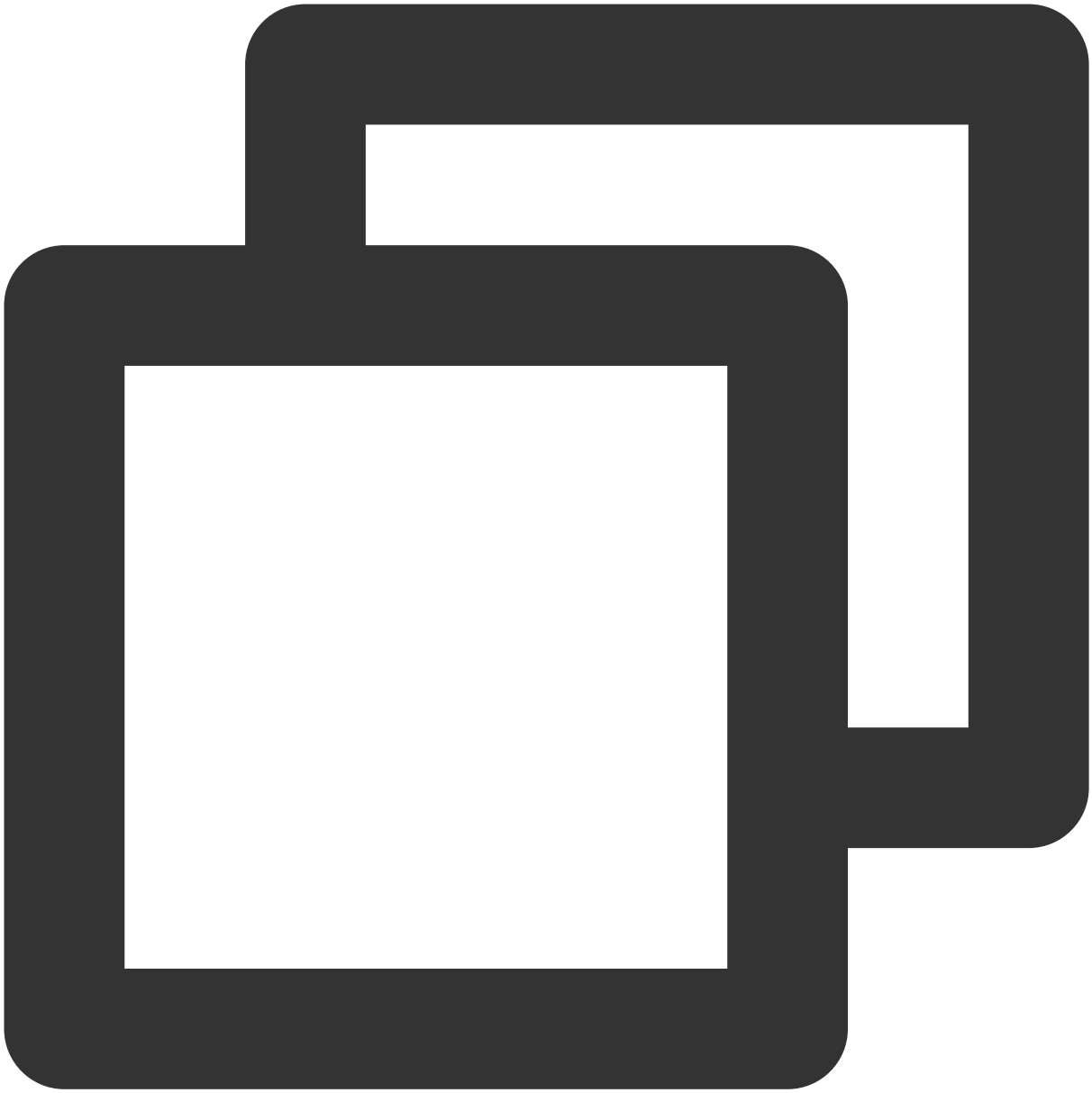
```
npm run dev
```

2. 在浏览器中打开页面 `http://localhost:3000/` 。

注意：

因 TUIRoom 按需引入 element-plus 组件，会导致开发环境路由页面第一次加载时反应较慢，等待 element-plus 按需加载完成即可正常使用。element-plus 按需加载不会影响打包之后的页面加载。

1. 执行开发环境命令



```
npm run serve
```

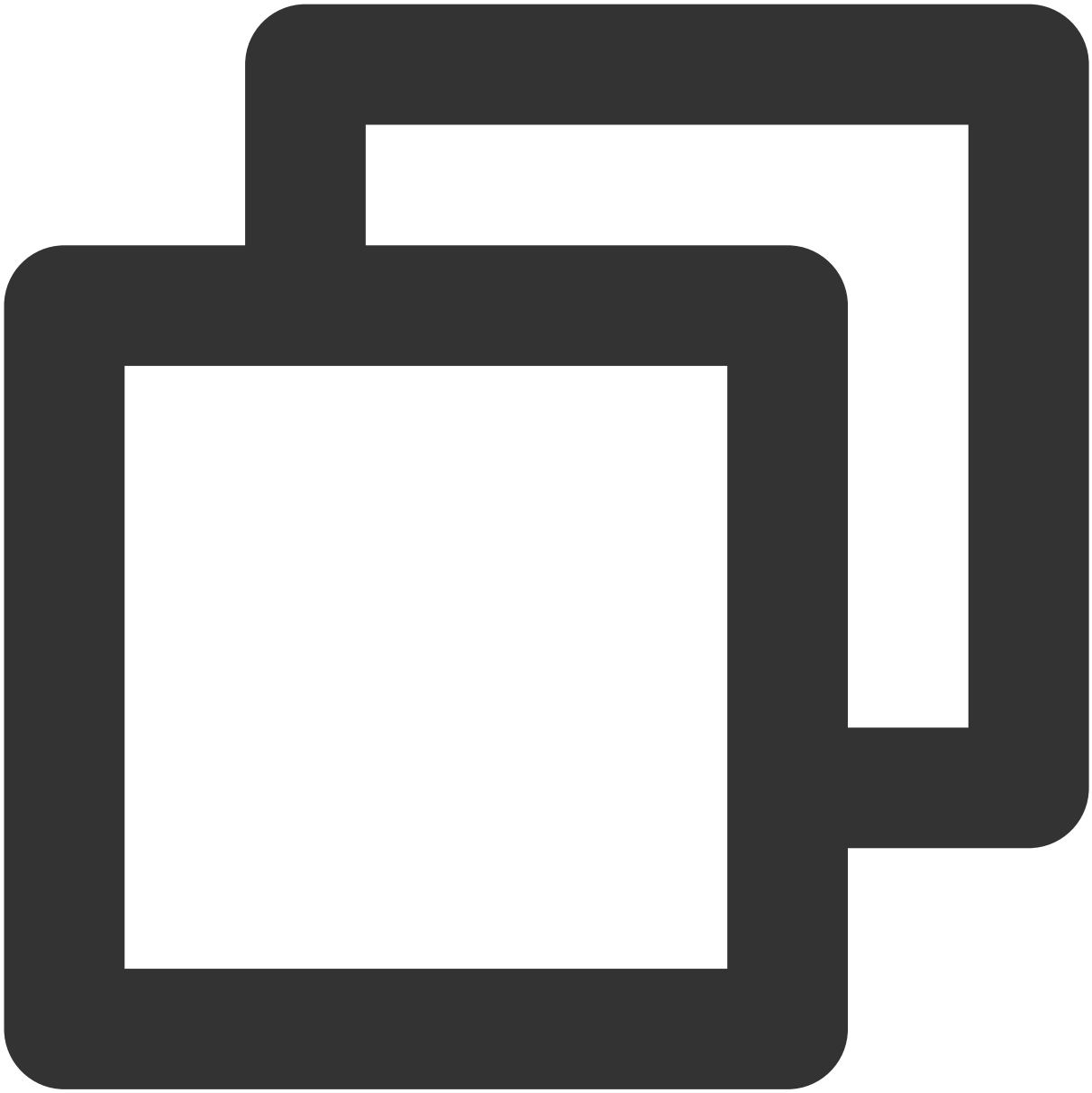
2. 在浏览器中打开页面 `http://localhost:8080/`

注意

运行过程中若 src/TUIRoom 目录中有 eslint 报错，可在 .eslintignore 文件中添加 /src/TUIRoom/ 路径屏蔽 eslint 检查。

3. 体验 TUIRoom 组件功能

1. 执行开发环境命令



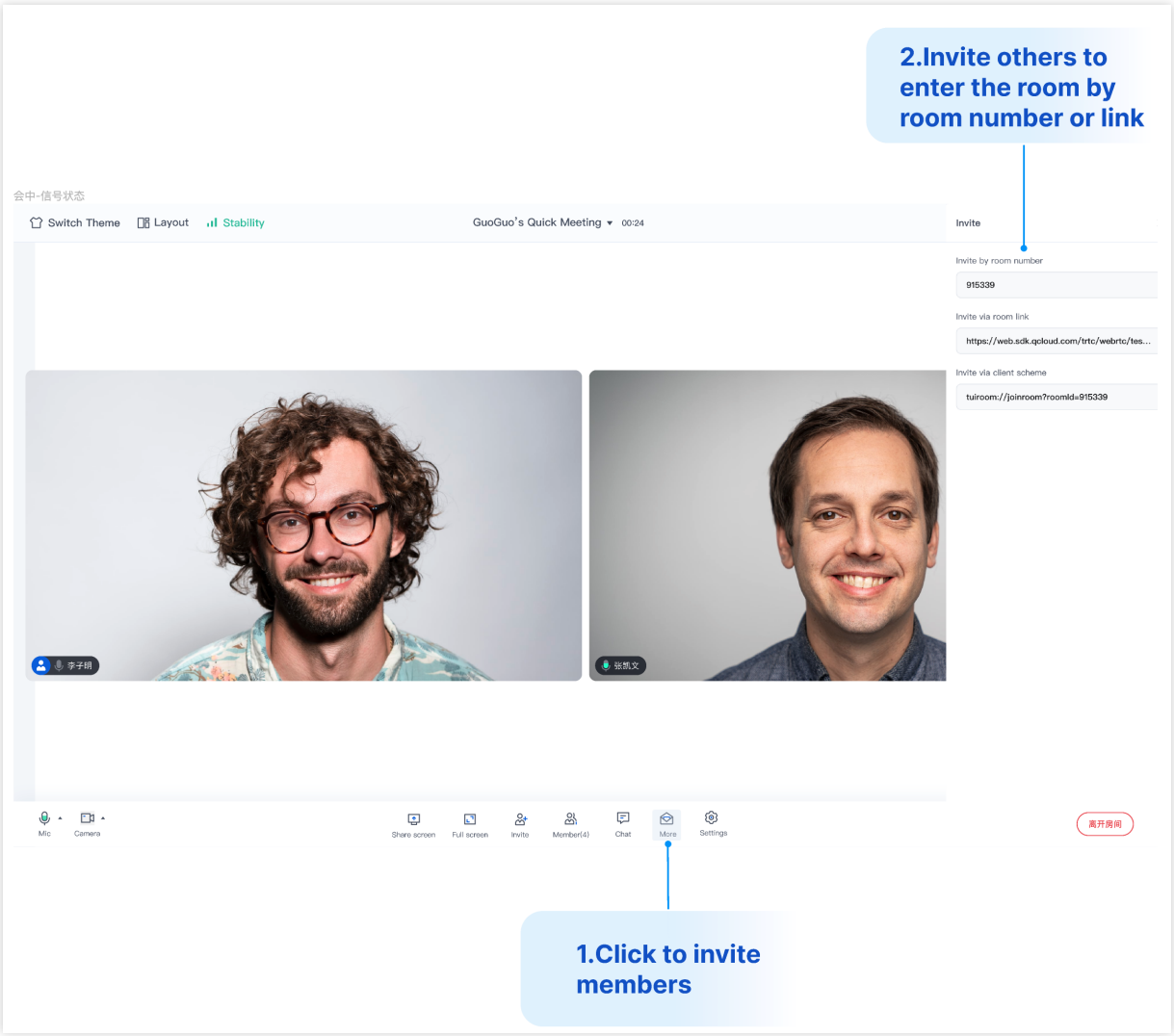
```
npm run serve
```

2. 在浏览器中打开页面 `http://localhost:8080/`

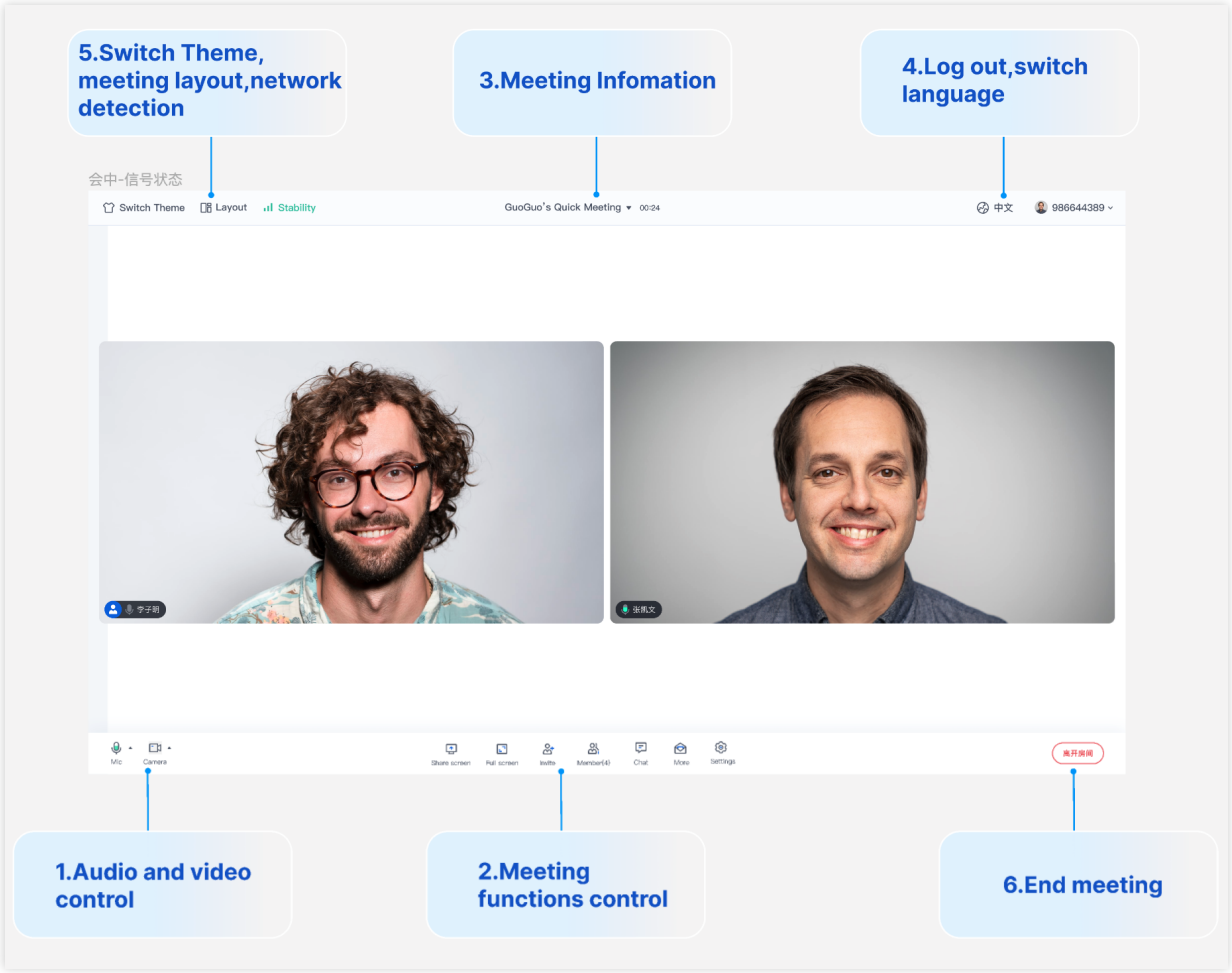
注意：

运行过程中若 src/TUIRoom 目录中有 eslint 报错，可在 .eslintignore 文件中添加 /src/TUIRoom/ 路径屏蔽 eslint 检查。

项目启动之后单击邀请成员，通过房间链接和客户端 scheme 邀请的方式在本地体验多用户进房交互。

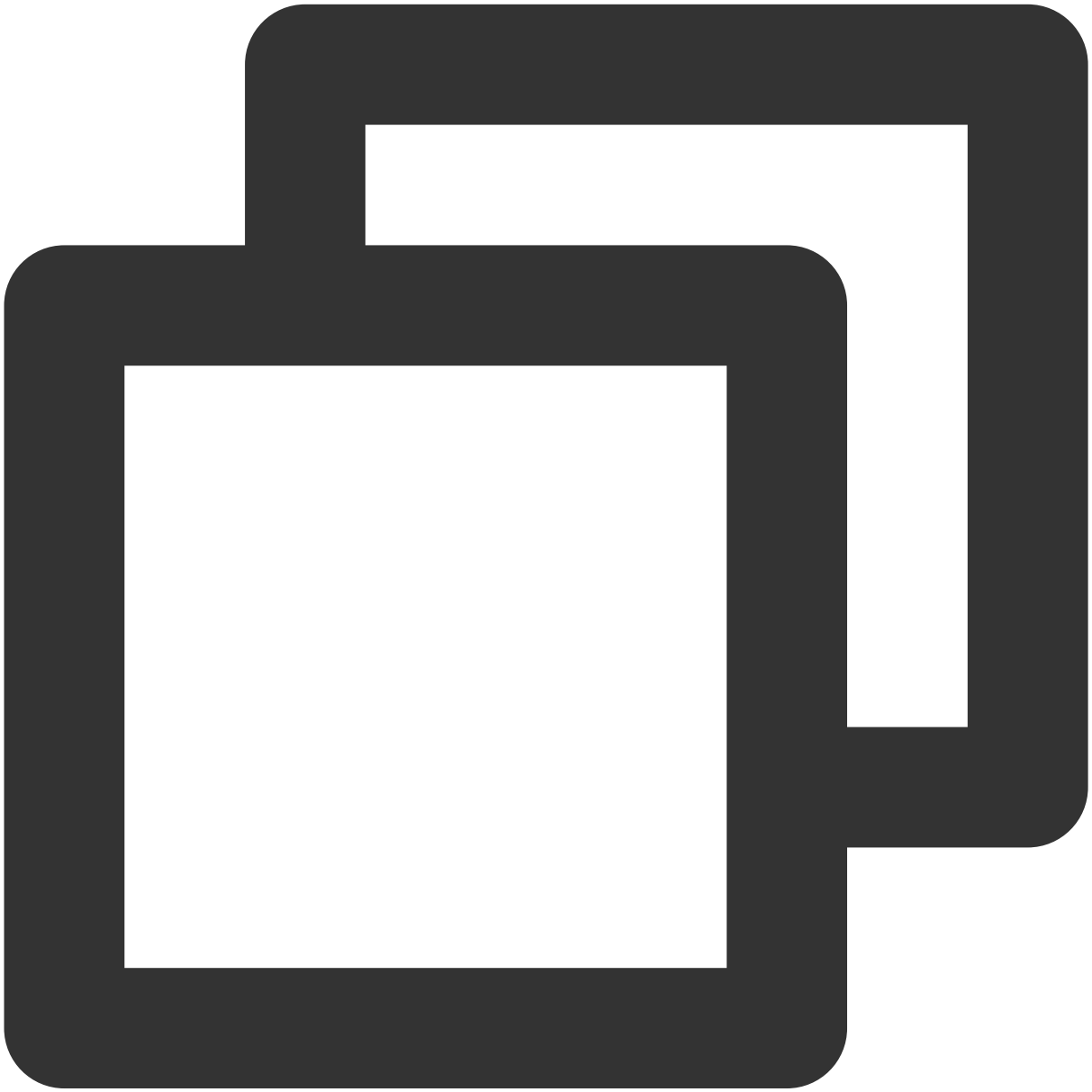

2.Invite others to enter the room by room number or link
会中-信号状态
Switch Theme
Layout
Stability
GuoGuo's Quick Meeting
00:24
Invite
Invite by room number
915339
Invite via room link
https://web.sdk.qcloud.com/trtc/webrtc/tes...
Invite via client scheme
tuiroom://joinroom?roomId=915339
李子明
张凯文
Mic
Camera
Share screen
Full screen
Invite
Member(4)
Chat
More
Settings
离开房间
1.Click to invite members

开始您的第一场会议。



步骤六：生产环境部署

1. 打包 dist 文件



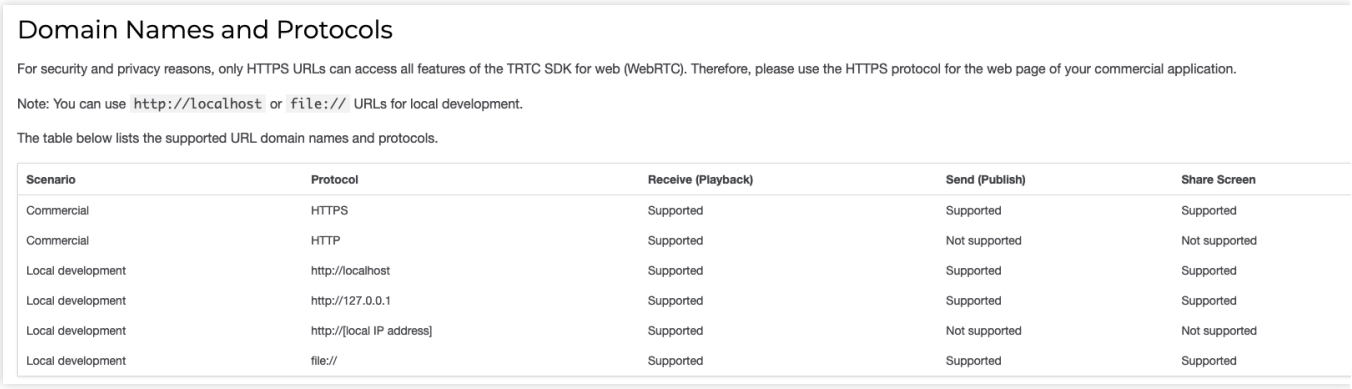
```
npm run build
```

说明：

实际打包命令请查看 package.json 文件

2. 部署 dist 文件到服务器上

生产环境要求使用 HTTPS 域名：



Domain Names and Protocols

For security and privacy reasons, only HTTPS URLs can access all features of the TRTC SDK for web (WebRTC). Therefore, please use the HTTPS protocol for the web page of your commercial application.

Note: You can use `http://localhost` or `file://` URLs for local development.

The table below lists the supported URL domain names and protocols.

Scenario	Protocol	Receive (Playback)	Send (Publish)	Share Screen
Commercial	HTTPS	Supported	Supported	Supported
Commercial	HTTP	Supported	Not supported	Not supported
Local development	http://localhost	Supported	Supported	Supported
Local development	http://127.0.0.1	Supported	Supported	Supported
Local development	http://[local IP address]	Supported	Not supported	Not supported
Local development	file://	Supported	Supported	Supported

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

Flutter

最近更新时间：2024-04-12 11:29:00

本文将介绍如何用最短的时间完成 `TUIRoomKit` 组件的接入，跟随本文档，您将在一个小时的时间内完成如下几个关键步骤，并最终得到一个包含完备 UI 界面的音视频会议功能。

环境准备

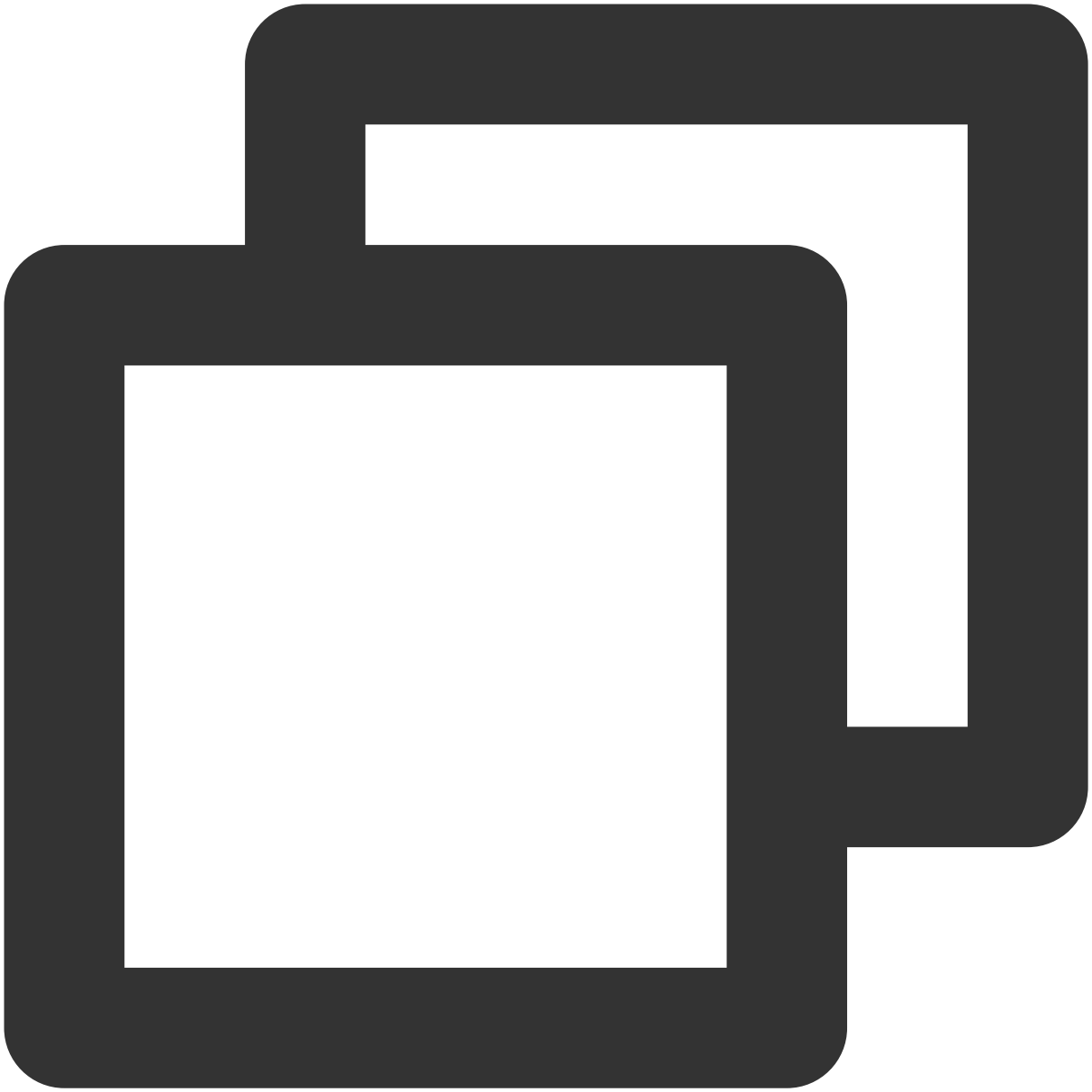
平台	版本
Flutter	3.0.0 及以上版本。
Android	最低兼容 Android 4.1（SDK API Level 16），建议使用 Android 5.0 （SDK API Level 21）及以上版本。 Android Studio 3.5 及以上的版本（Gradle 3.5.4 及以上的版本）。 Android 4.1 及以上的手机设备。
iOS	iOS 12.0 及更高。

步骤一：开通服务

在使用 TUIRoomKit 发起会议前，您需要前往控制台开通 TUIRoomKit 专属的多人音视频互动服务，具体步骤请参见开通服务。

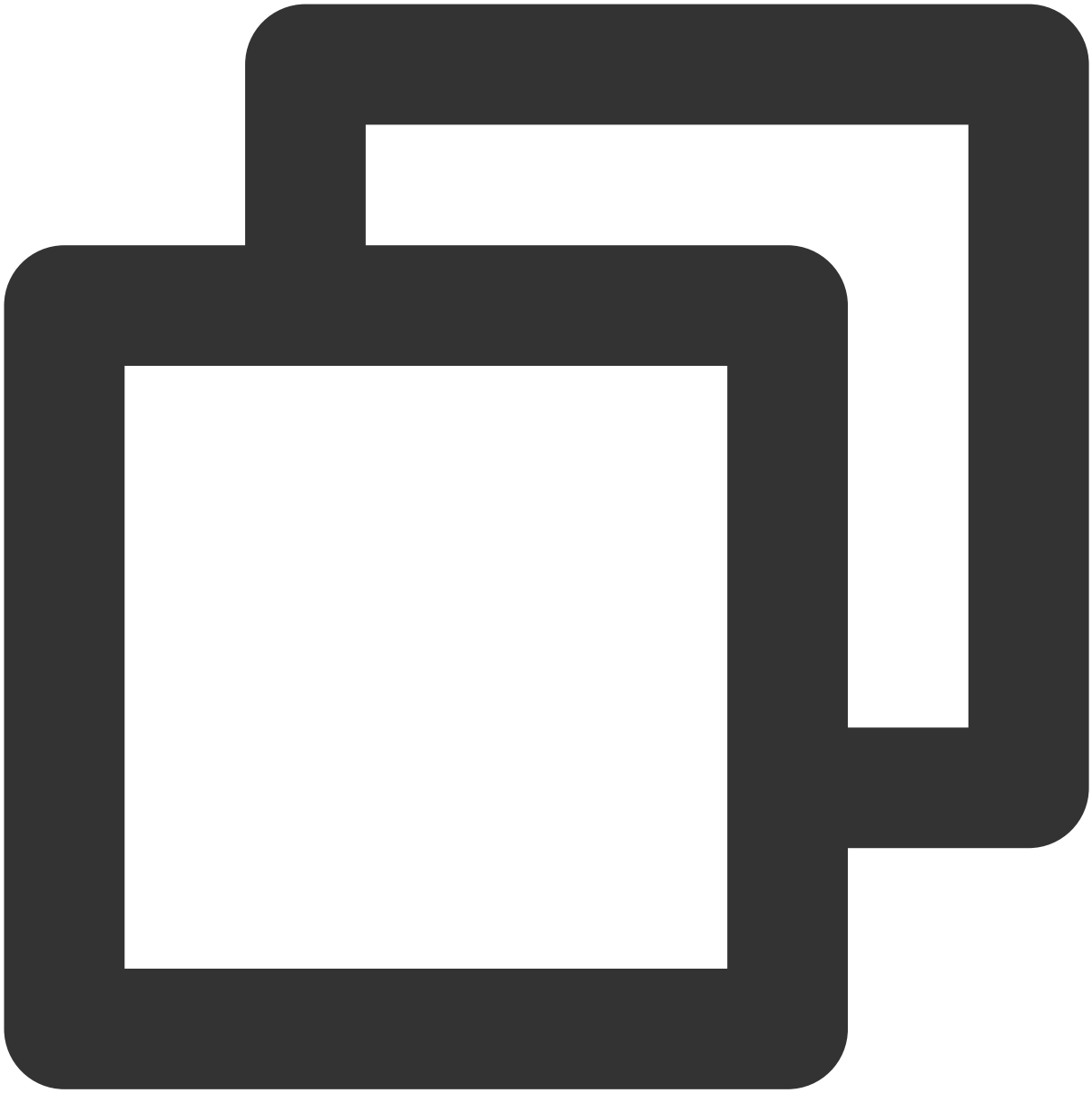
步骤二：集成 TUIRoomKit 组件

在您的工程 `pubspec.yaml` 文件中，添加 rtc_conference_tui_kit 插件依赖，或进入你的工程目录下执行如下命令添加依赖。



```
dependencies:
    rtc_conference_tui_kit: 最新版本
```

执行以下命令安装组件：

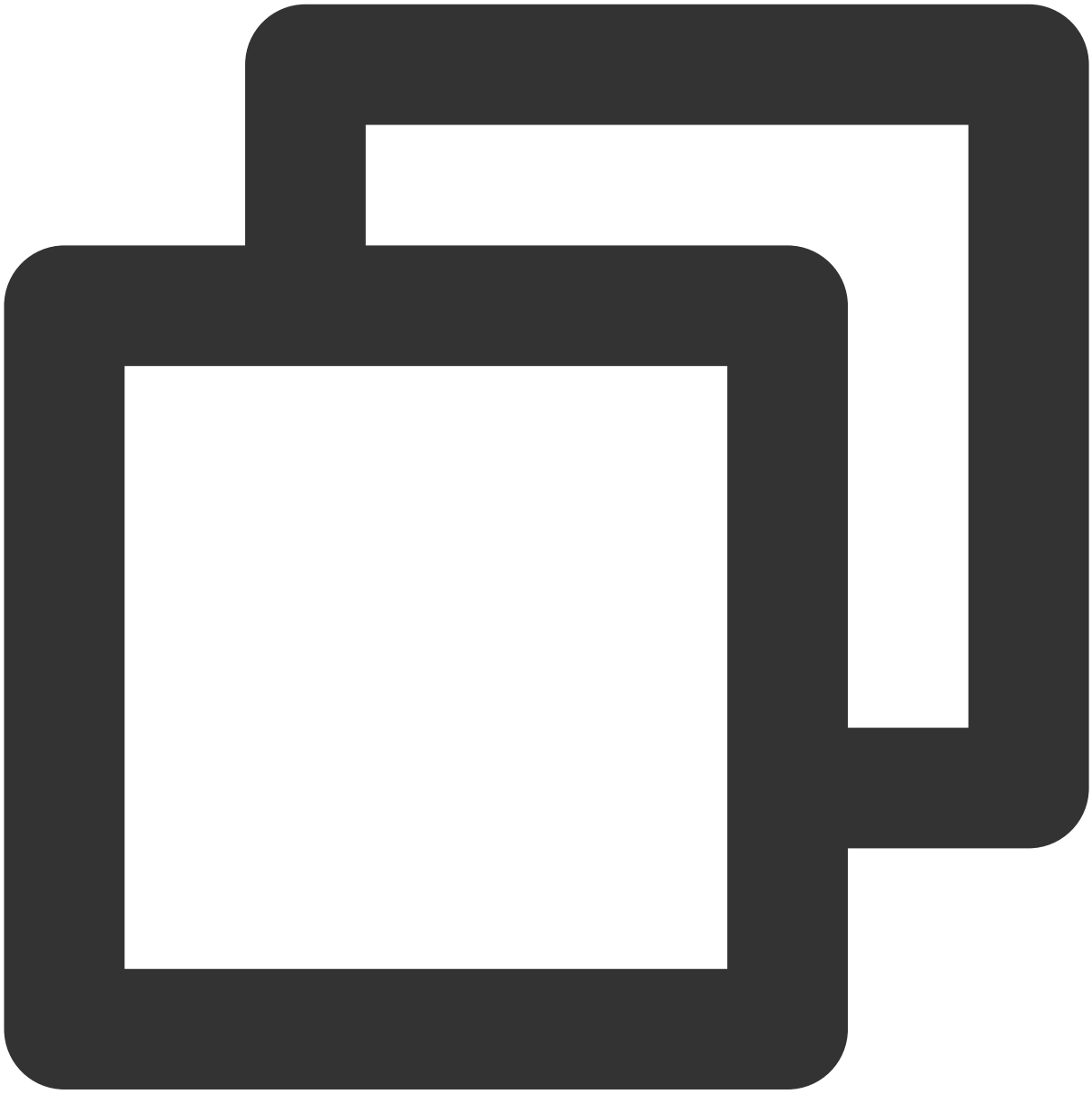


```
flutter pub get
```

步骤三：完成工程配置

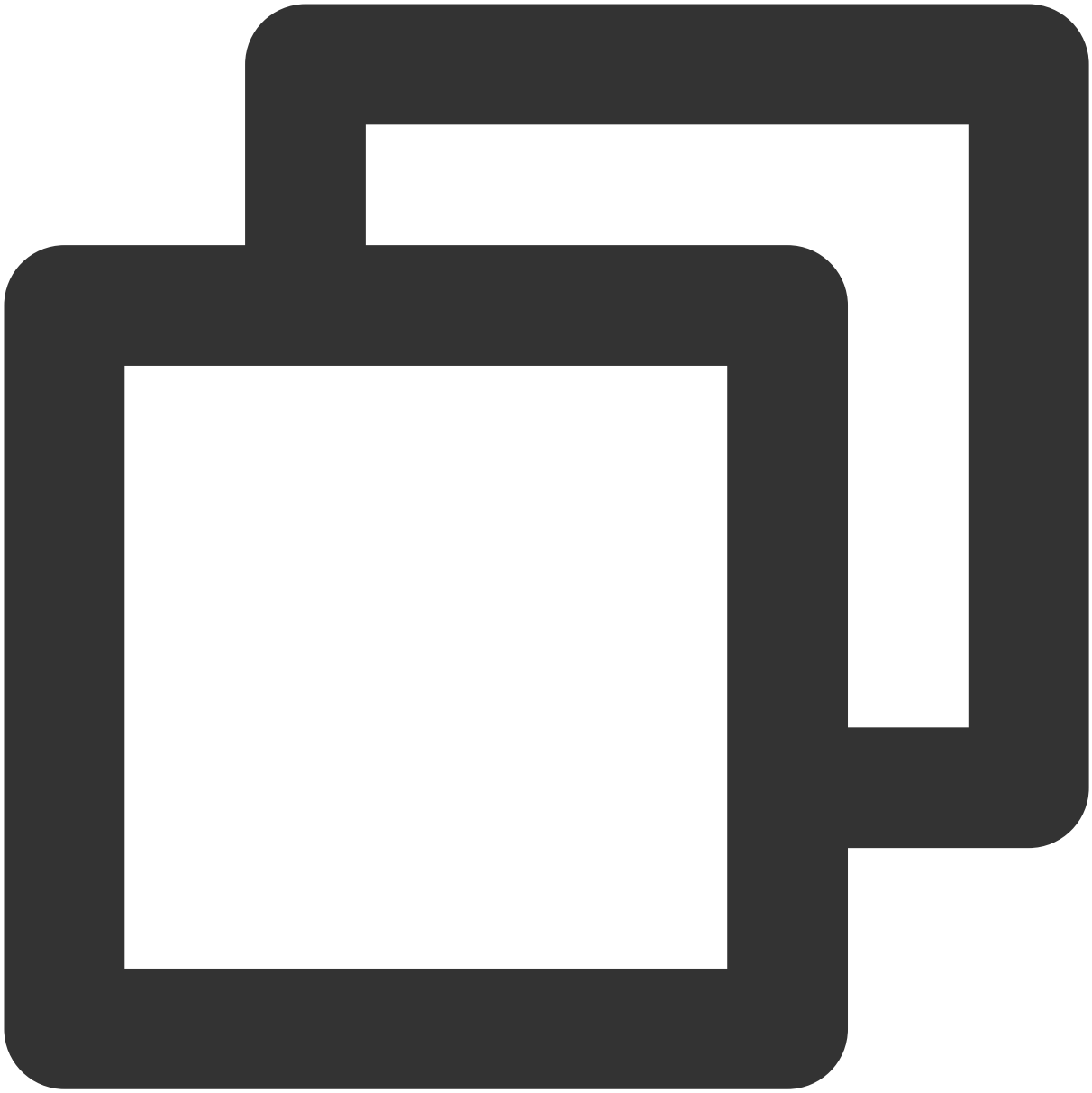
由于 **rtc_conference_tui_kit** 组件使用了 `GetX` 状态管理库进行导航，您需要在您的应用程序中使用 `GetMaterialApp` 来代替 `MaterialApp` 。或者您也可以将您 `MaterialApp` 中的 `navigatorKey` 属性设置为 `Get.key` 以实现相同的效果。

使用 **Xcode** 打开您的工程，选择**项目** > **Building Settings** > **Deployment**，将其下的 **Strip Style** 设置为 `Non-Global Symbols` ，以保留所需要的全局符号信息。
如您需要在 `iOS` 端使用音视频功能，需要授权麦克风和摄像头的使用权限（ `Android` 端已在 SDK 中声明相关权限，您无需手动进行相关配置）。
在 App 的 `Info.plist` 中添加以下两项，分别对应麦克风和摄像头在系统弹出授权对话框时的提示信息。



```
<key>NSCameraUsageDescription</key>
<string>TUIRoom需要访问您的相机权限</string>
<key>NSMicrophoneUsageDescription</key>
<string>TUIRoom需要访问您的麦克风权限</string>
```

完成以上添加后，在您的 **ios/Podfile** 中添加以下预处理器定义，用于启用相机与麦克风权限。

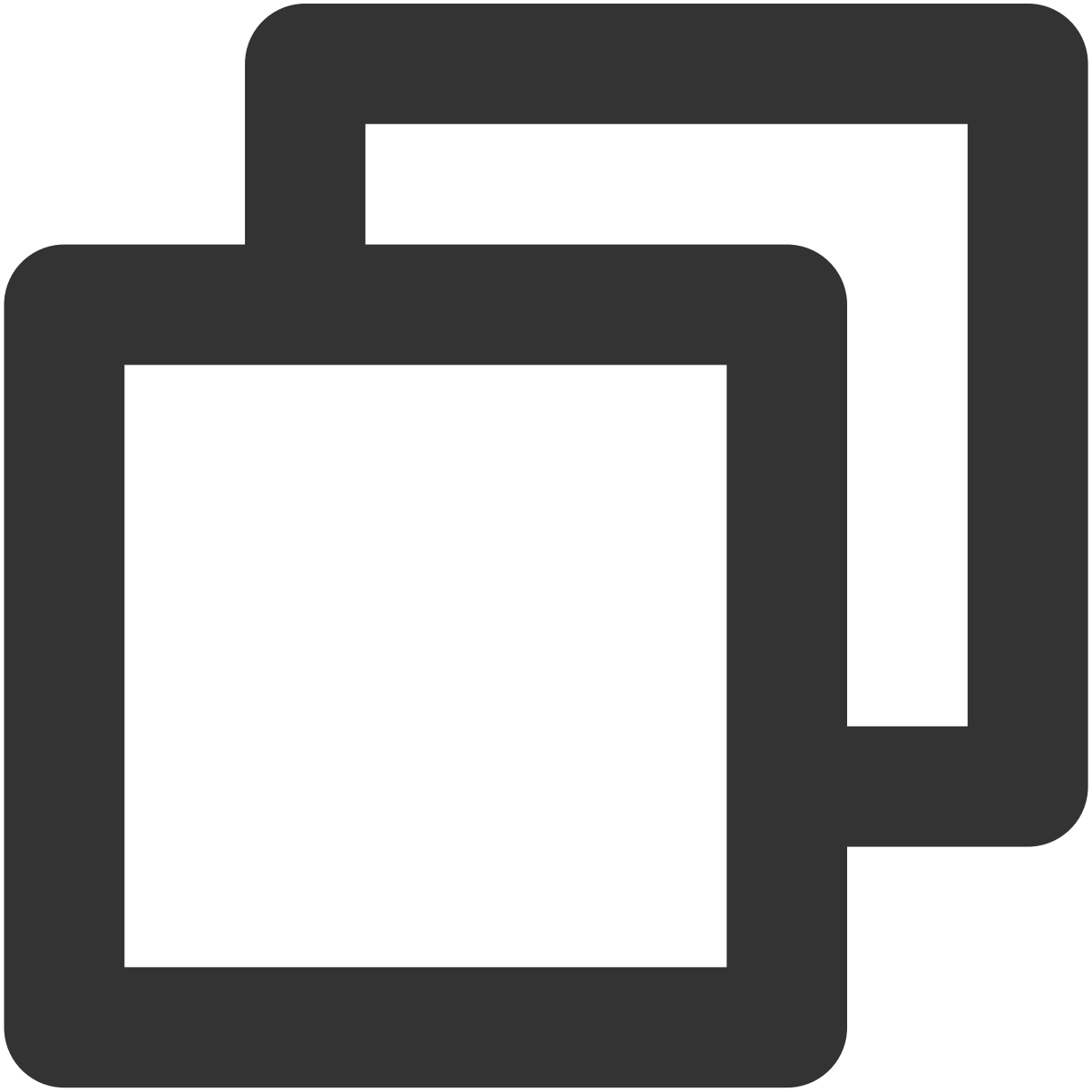


```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
      target.build_configurations.each do |config|
        config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
      '$(inherited)',
      'PERMISSION_MICROPHONE=1',
      'PERMISSION_CAMERA=1',
      ]
```

```
    end
  end
end
```

步骤四：登录组件

在您的项目中添加如下代码，它的作用是通过调用 `TUIRoomKit` 中的相关接口完成组件的初始化。这个步骤非常关键，因为只有在初始化后才能正常使用 `TUIRoomKit` 的各项功能，故请您耐心检查相关参数是否配置正确：



```
import 'package:rtc_room_engine/rtc_room_engine.dart';

var result = await TUIRoomEngine.login(
    SDKAPPID, // 请替换为您的SDKAPPID
    'userId', // 请替换为您的User ID
    'userSig',// 请替换为您的userSig
    );

if (result.code == TUIError.success) {
    // login success
} else {
    // login error
}
```

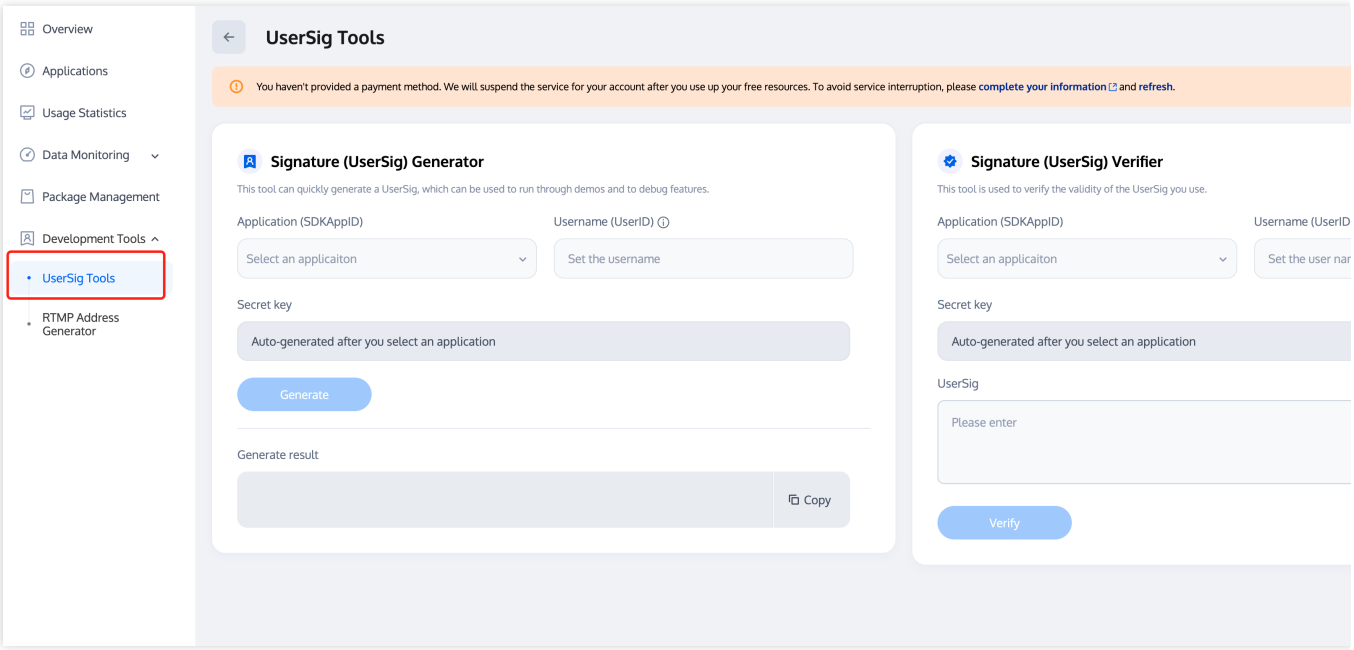
参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

SDKAppID：在 步骤一 中的第4步中您已经获取到，这里不再赘述。

UserID：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。

UserSig：使用 步骤一 的第4步中获取的 SDKSecretKey 对 SDKAppID、UserID 等信息进行加密，就可以得到 UserSig，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以通过控制台左侧项目栏中的 UserSig 工具，创建一个临时可用的 UserSig。



更多信息请参见 UserSig相关。

注意：

这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下：

`SDKAppID` 设置错误，国内站的 `SDKAppID` 一般是以140开头的10位整数。

`UserSig` 被错配成了加密密钥（ `SDKSecretKey` ）， `UserSig` 是用 `SDKSecretKey` 把 `SDKAppID` 、 `UserID` 以及过期时间等信息加密得来的，而不是直接把 `SDKSecretKey` 配置成 `UserSig` 。

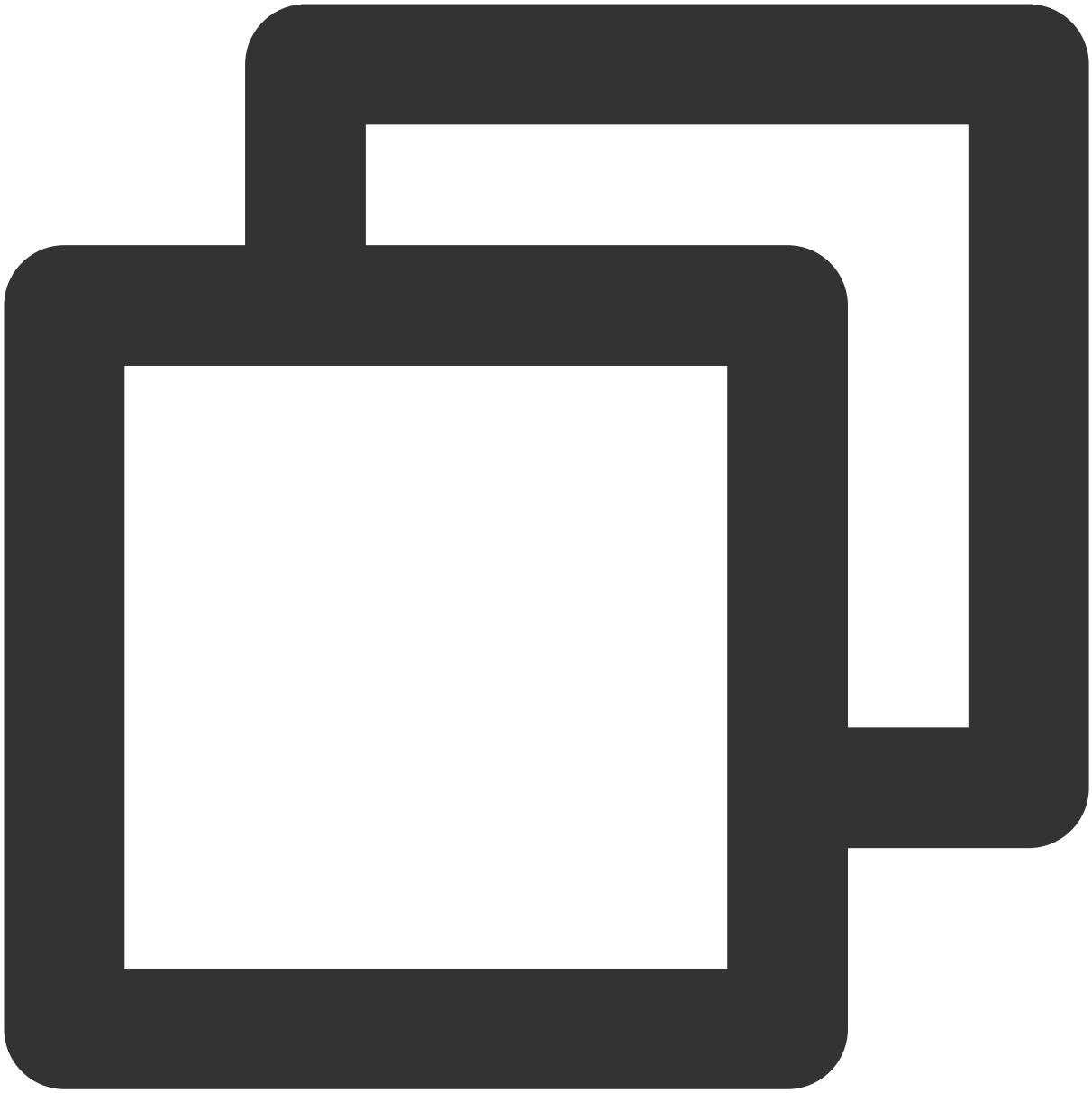
`UserID` 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 `UserID` 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 `UserID` 。

`Github` 中的示例代码使用了 `genTestUserSig` 函数在本地计算 UserSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 `SDKSecretKey` 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SDKSecretKey，所以我们强烈建议您将 `UserSig` 的计算逻辑放在服务端进行，并由 App 在每次使用 `TUIRoomKit` 组件时向您的服务器请求实时计算出的 UserSig。

步骤五：使用组件

设置用户信息（可选）

通过调用 `TUIRoomEngine` 的 `setSelfInfo` 即可设置当前用户的用户名、头像。

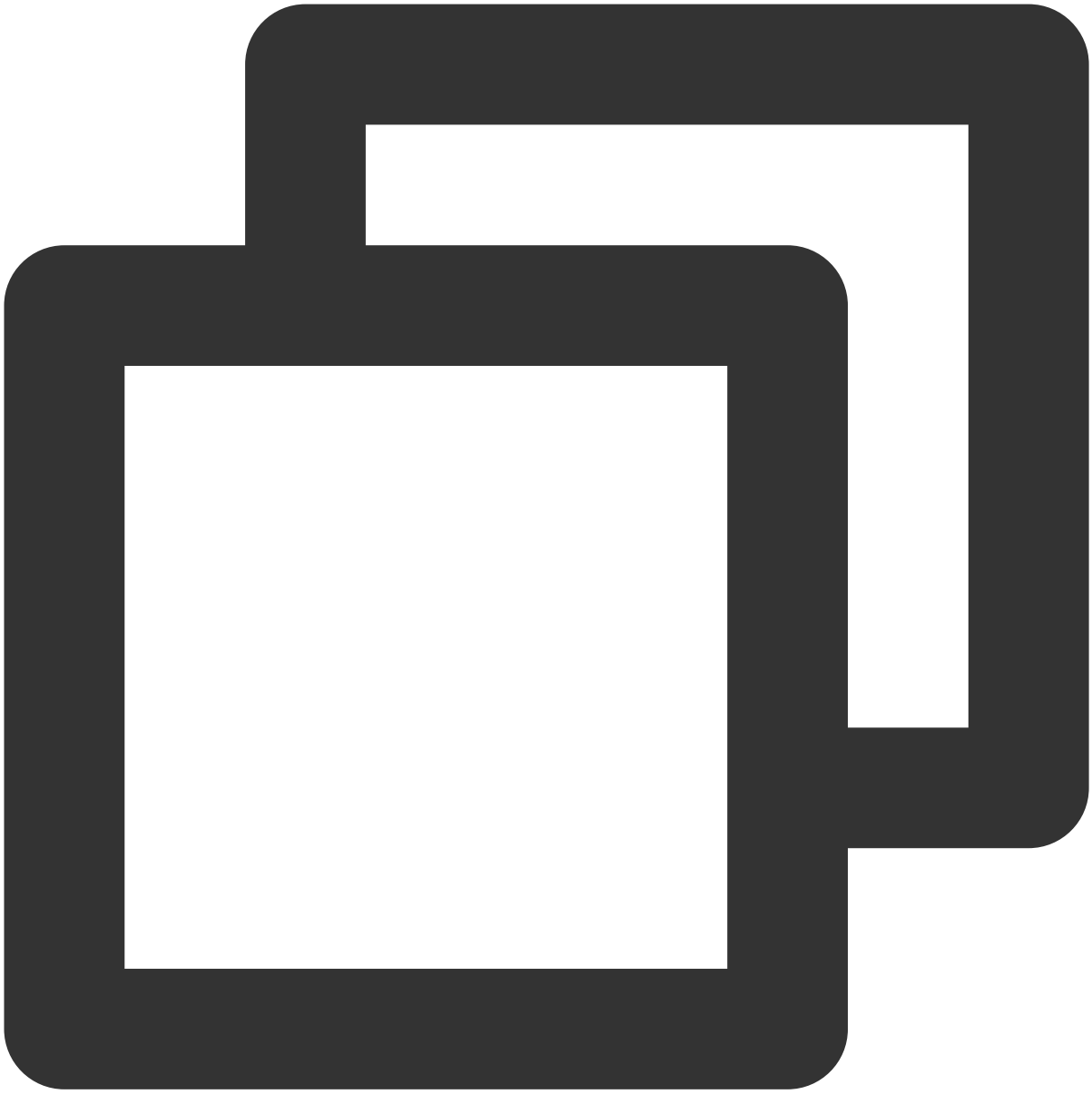


```
import 'package:rtc_room_engine/rtc_room_engine.dart';

TUIRoomEngine.setSelfInfo(userName, avatarURL);
```

参数	类型	含义
userName	String	用户名
avatarURL	String	用户头像 URL

开始快速会议

通过调用 `ConferenceSession` 的 `quickStart` 方法，便可以开始快速会议。

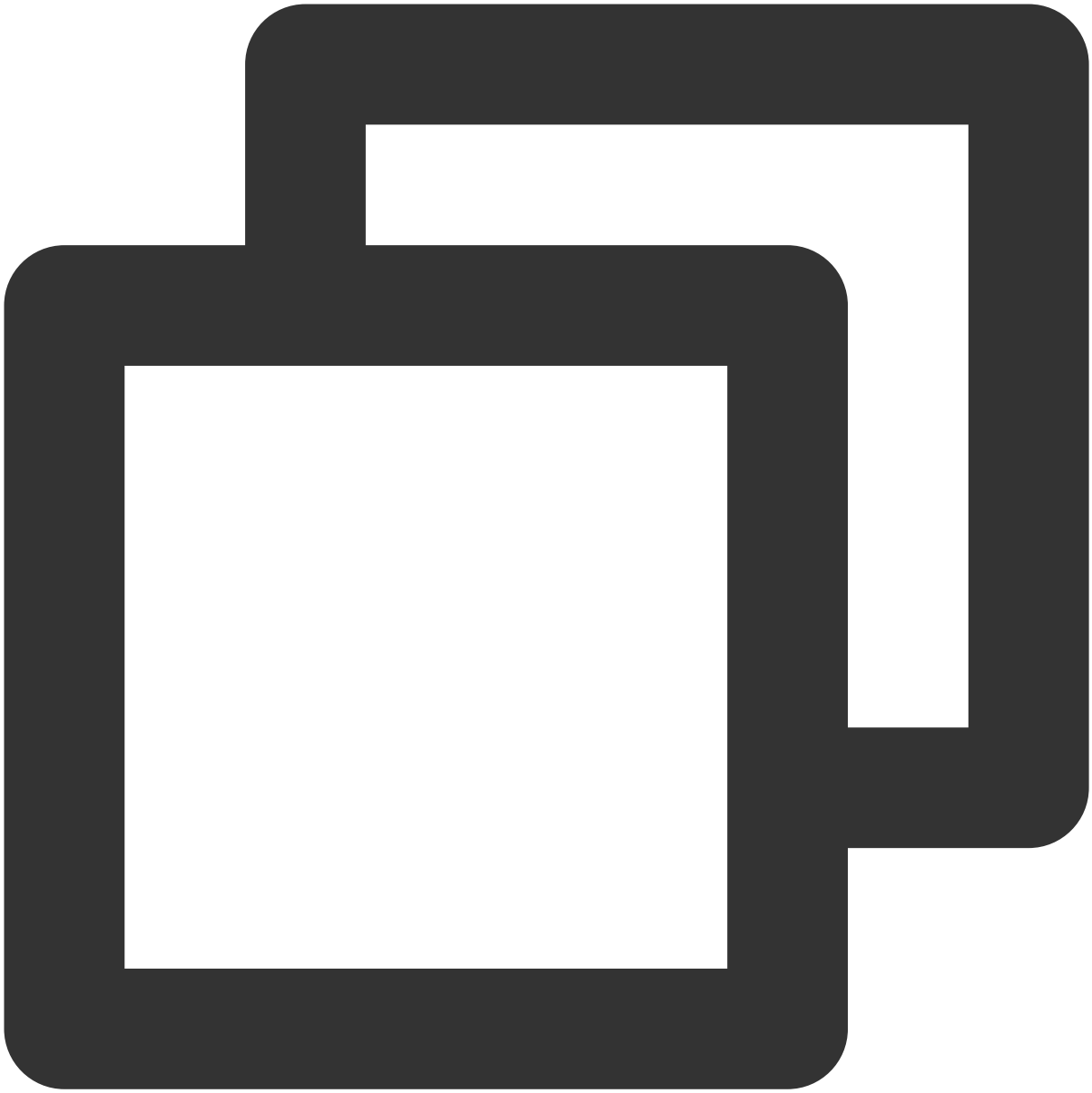


```
import 'package:rtc_conference_tui_kit/rtc_conference_tui_kit.dart';

ConferenceSession.newInstance('roomId')    //您的room id
  ..onActionSuccess = _quickStartSuccess
  ..onActionError = _quickStartError
  ..quickStart();

void _quickStartSuccess() {
```

```
  //您可以在开始快速会议的成功回调中,自行导航至会议页面。
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => ConferenceMainPage(),
    ),
  );
}

void _quickStartError(ConferenceError error, String message) {
  debugPrint("code: $error message: $message");
}
```

加入会议

通过调用 `ConferenceSession` 的 `join` 方法，便可以加入指定会议。



```
import 'package:rtc_conference_tui_kit/rtc_conference_tui_kit.dart';

ConferenceSession.newInstance('roomId')    //您的room id
  ..onActionSuccess = _joinSuccess
  ..onActionError = _joinError
  ..join();

void _joinSuccess() {
  //您可以在加入会议的成功回调中,自行导航至会议页面。
  Navigator.push(
    context,
```

```
      MaterialPageRoute(
        builder: (context) => ConferenceMainPage(),
      ),
    );
}

void _joinError(ConferenceError error, String message) {
  debugPrint("code: $error message: $message");
}
```

更多功能

TUIRoomEngine SDK 提供了丰富音视频房间功能，具体内容请参考 API 概览。

交流与反馈

如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

界面定制 (TUIRoomKit)

Electron

最近更新时间：2023-09-22 09:58:34

本文将介绍如何定制 TUIRoomkit 的用户界面，我们提供了两个方案供您选择：换肤方案和自定义 UI 方案。

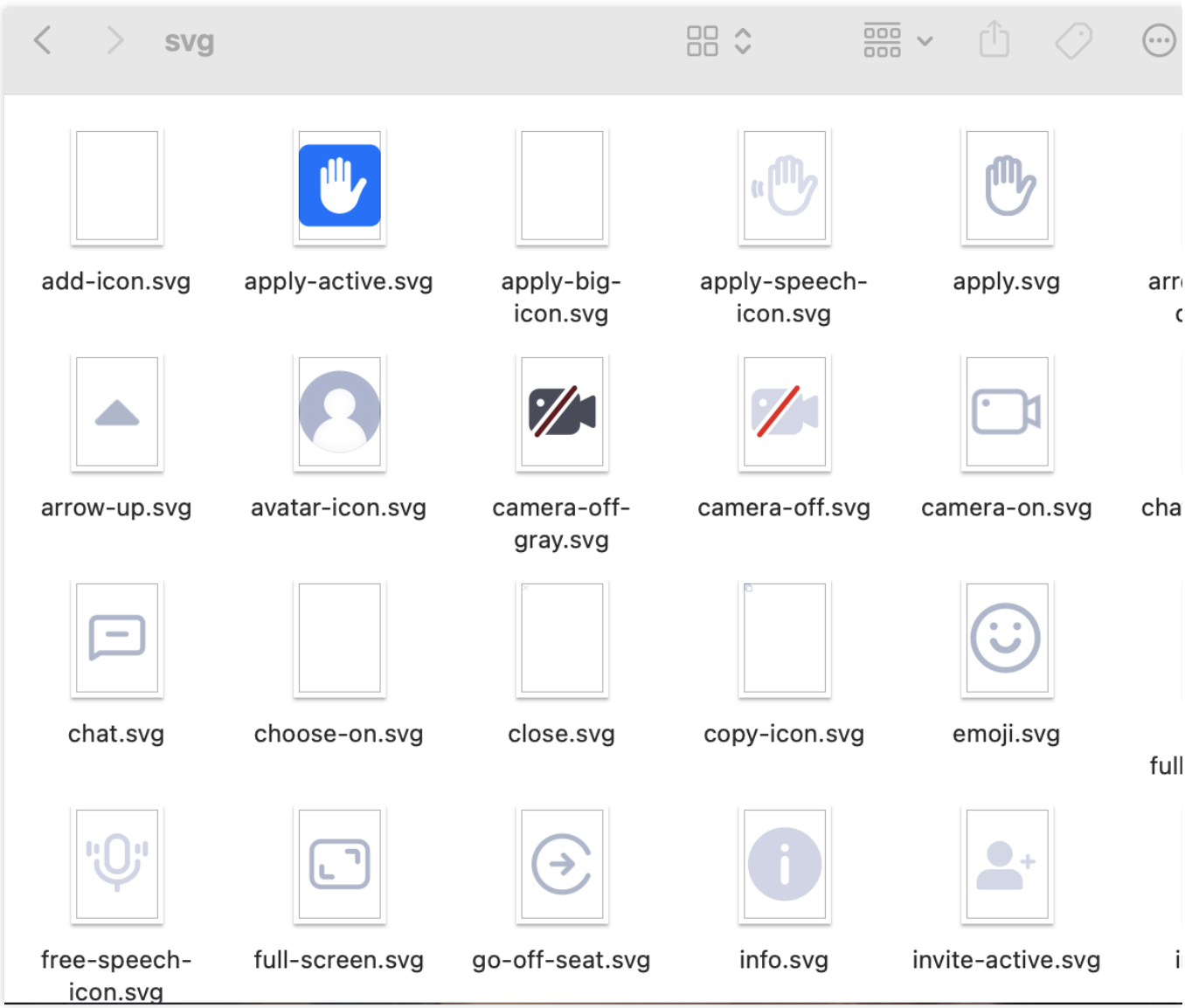
方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUIRoomKit 的用户界面进行调整，TUIRoomKit 的界面源代码位于 Github 中的 Electron/ 文件夹下面：

替换图标

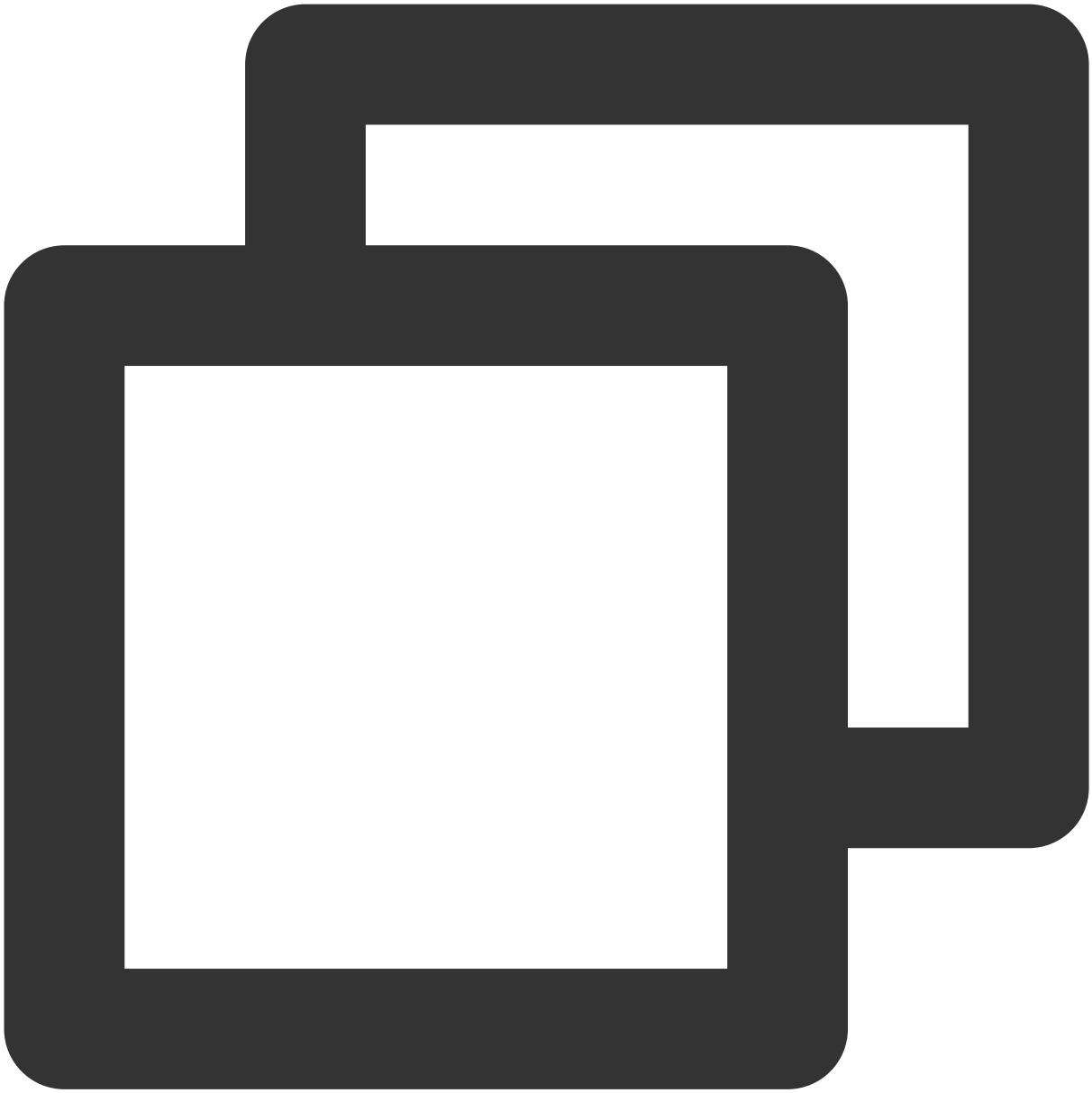
您可以直接修改 `packages/renderer/src/TUIRoom/assests/icons/svg` 文件夹下的图标组件，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。

版权所有：腾讯云计算（北京）有限责任公司



调整 UI 布局

您可以通过修改 `packages/renderer/src/TUIRoom/components/` 文件下的不同页面来调整多人视频会议界面的UI布局



```
- components/
    - base
    - Chat                  聊天
    - common                公用图标组件
    - ManageMember          成员管理
    - RoomContent           Room视频
    - RoomFooter            Room页面Footer部分
    - RoomHeader            Room页面Header部分
    - RoomHome              Home页面
    - RoomInvite            邀请成员
    - RoomLogin             登录页面
```

```
- RoomMore          更多
- RoomSetting       设置
- RoomSidebar       抽屉组件
```

方案二：自定义 UI 方案

TUIRoom 的整体功能是基于 TUIRoomEngine 这个无 UI SDK实现的，您可以完全基于 TUIRoomEngine 实现一套自己的 UI 界面。详情可见
TUIRoomEngine 接入指引
TUIRoomEngine API 接口地址

Android

最近更新时间：2023-09-22 10:11:05

本文将介绍如何定制 TUIRoomKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

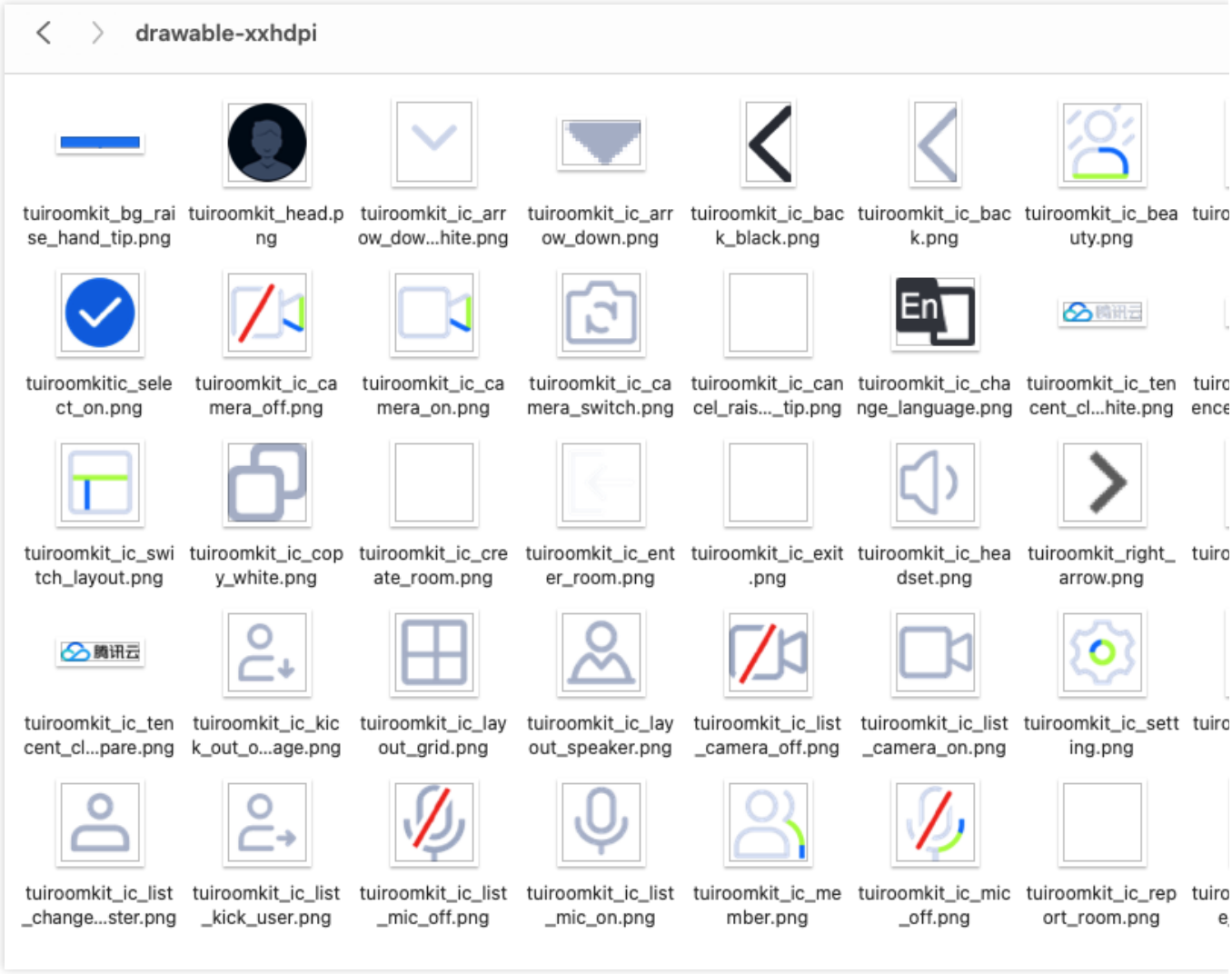
方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUIRoomKit 的用户界面进行调整，TUIRoomKit 的界面源代码位于 Github 中的 `Android/tuiroomkit` 文件夹下面：

替换图标

您可以直接替换 `src/res/drawable-xxhdpi` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。

版权所有：腾讯云计算（北京）有限责任公司



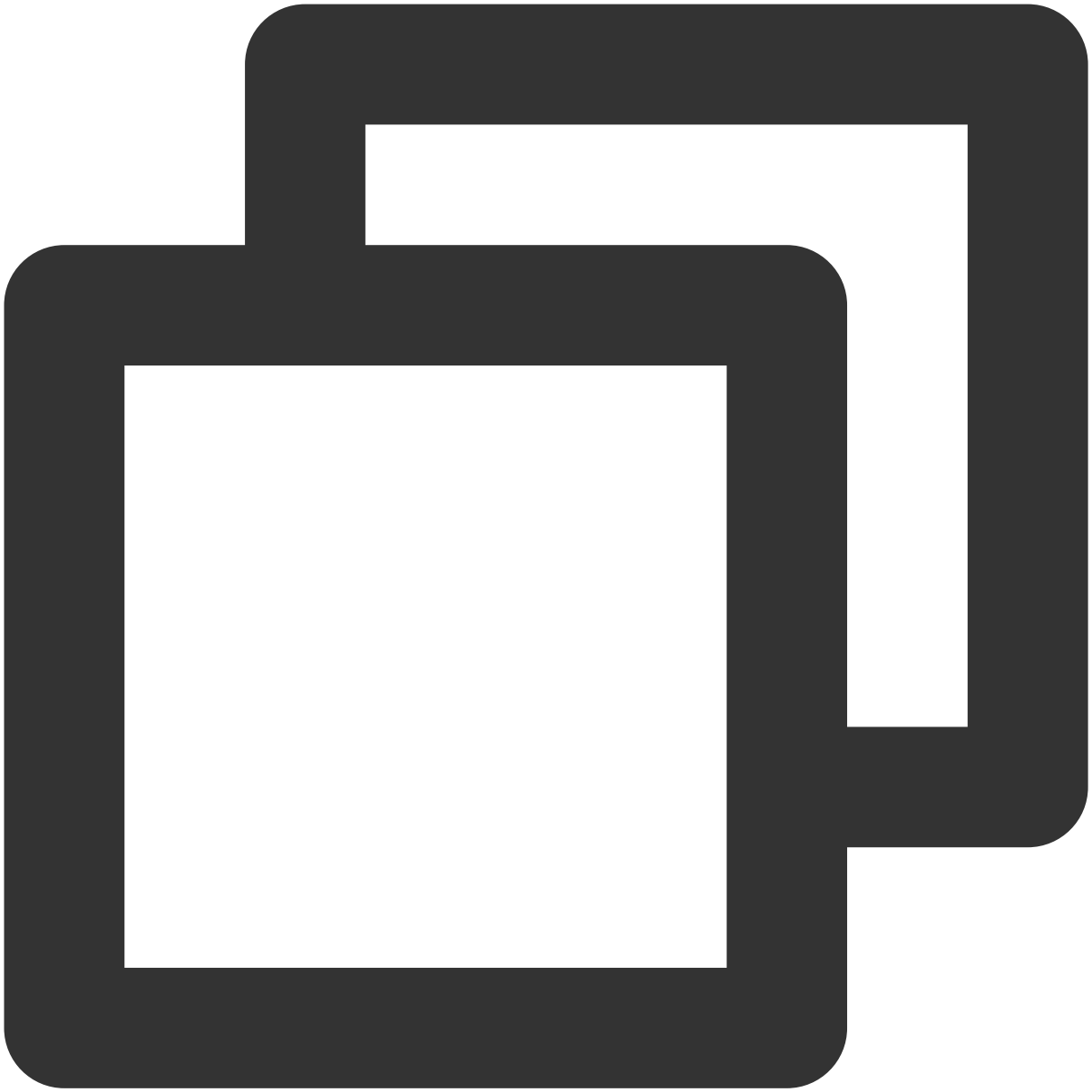
替换文案

您可以通过修改 values-zh 和 values-en 中的 `strings.xml` 文件来修改视频会议界面的字符串内容。

方案二：自定义部分 UI 方案

TUIRoomKit 的 UI 代码位于 `src/main/java/com/tencent/cloud/tuikit/roomkit/view` 目录下，视屏 view 位于 TUIVideoSeat 组件中

TUIRoomKit 的 View 关键文件如下，您可以按照您的需求更改对应的view，调整您的UI

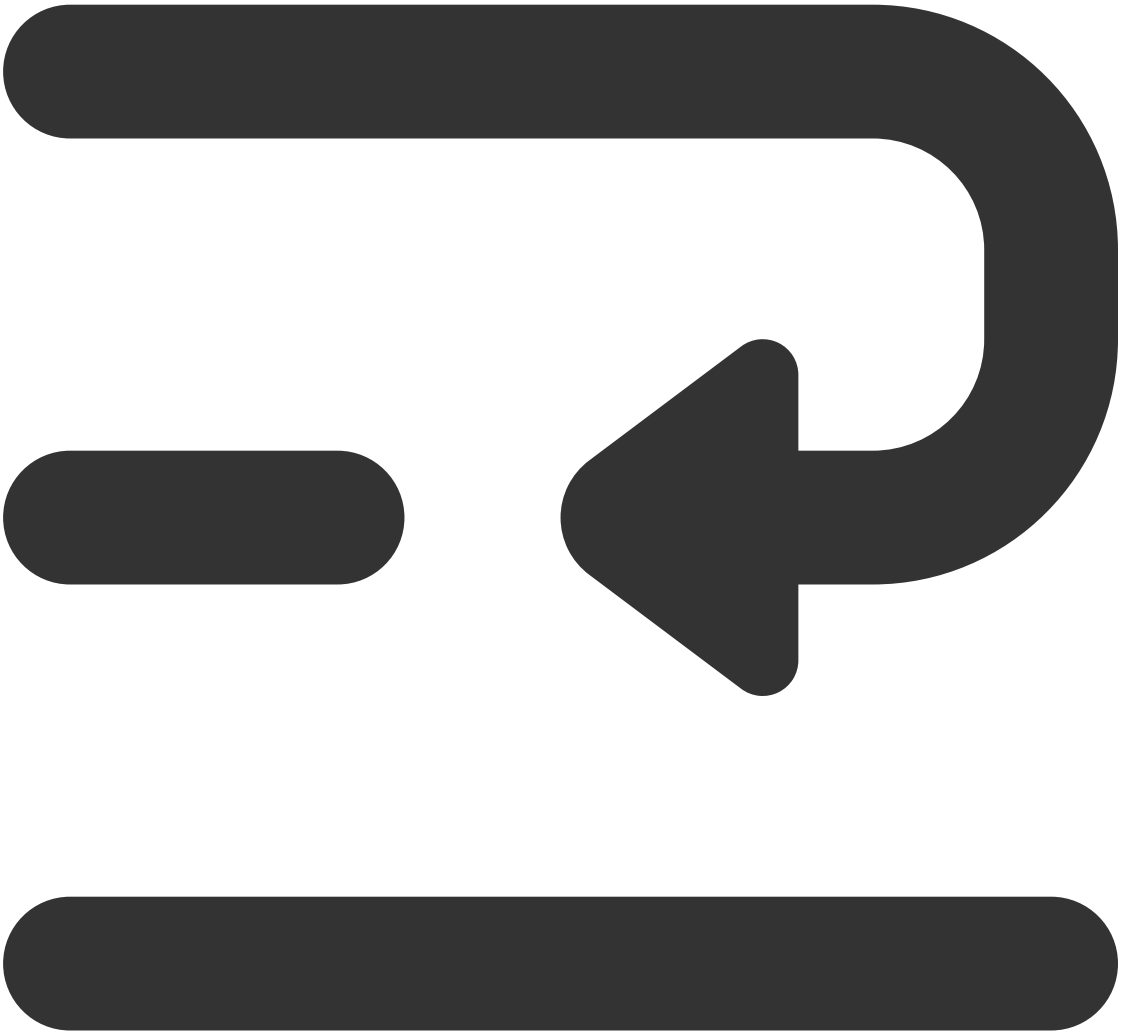


```
                ├── PrepareActivity     // 准备页面Activity
                │
                ├── CreateRoomActivity  // 创建房间Activity
                │
                ├── EnterRoomActivity   // 进入房间Activity
                │
    activity ── ├── RoomMainActivity    // 房间主页面Activity
        │
view ── │
        │
    component── ├── PrepareView         // 准备页面View
```

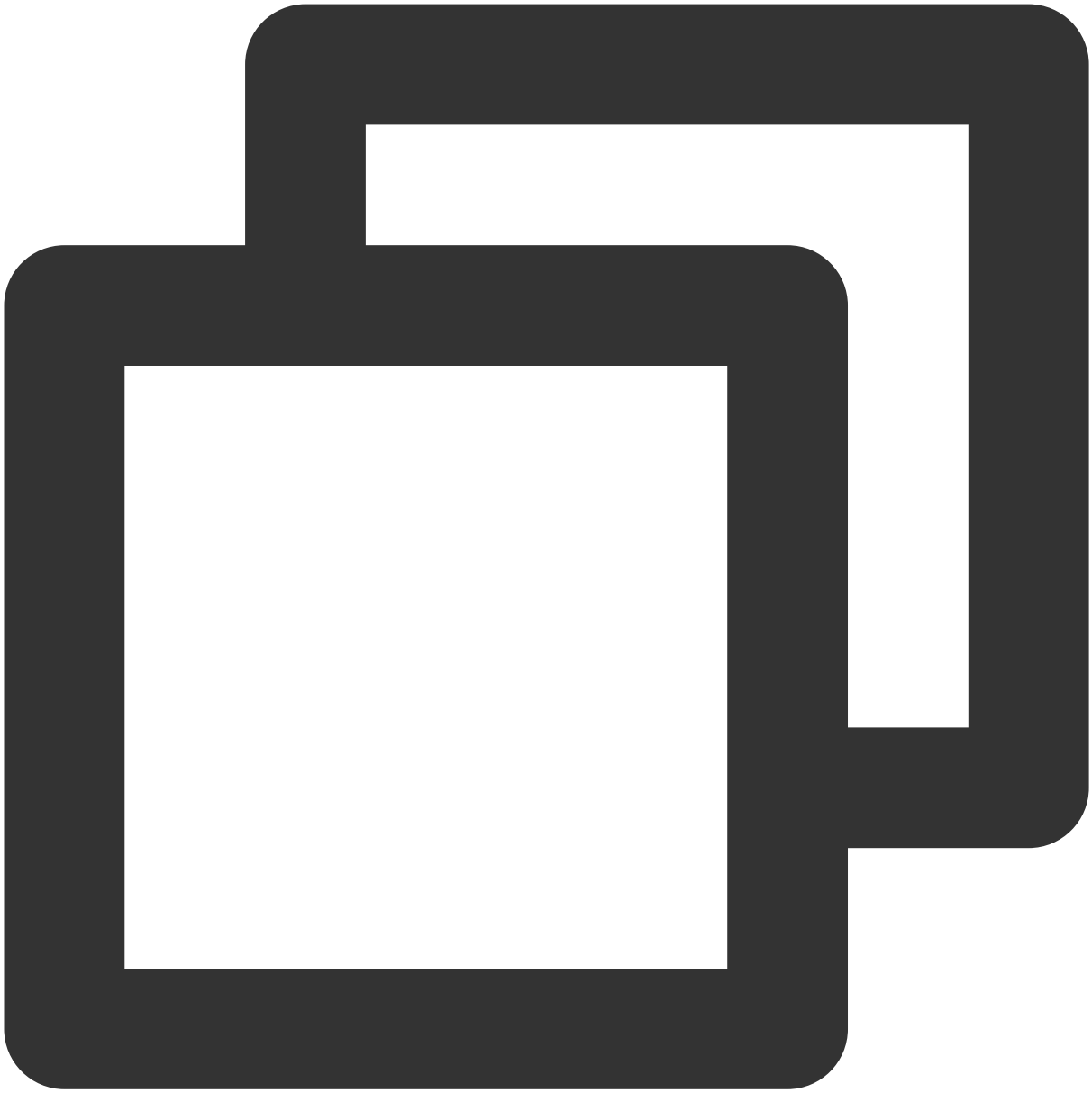
```
|
|── CreateRoomView      // 创建房间View
|
|── EnterRoomView       // 进入房间View
|
|── RoomMainView        // 音视频会议页面主页面View
|
|── TopView             // 顶部按钮所在view，顶部按钮包括：扬声器/听筒切换，
|
|── BottomView          // 底部按钮所在view，底部按钮包括：摄像头、麦克风、
|
|── UseListView         // 用户列表所在view
|
|── RaiseHandApplicationListView      // 举手发言模式，举手申请列表页面
|
|── TransferMasterView // 房主转让页面
|
|── MoreFunctionView    // “更多“功能所在view，更多功能有聊天、美颜以及设置
|
|── MeetingActivity     // 音视频会议界面主activity
```

底部按钮 BottomView 的修改

为了方便您自定义底部的功能按钮，我们的 BottomView 是通过读取 list 来自动构造，以开关视频按钮为例子，代码如下



版权所有：腾讯云计算（北京）有限责任公司



```
private BottomItemData createCameraItem() {
    BottomItemData cameraItemData = new BottomItemData();
    //设置按钮类型用于区分不同按钮
    cameraItemData.setType(BottomItemData.Type.VIDEO);
    //设置按钮是否可点击
    if (isOwner()) {
      cameraItemData.setEnable(true);
    } else if (mRoomStore.roomInfo.enableSeatControl) {
      cameraItemData.setEnable(false);
    } else {
      cameraItemData.setEnable(mRoomStore.roomInfo.enableVideo);
```

```
    }
    //设置按钮默认图标
    cameraItemData.setIconId(R.drawable.tuiroomkit_ic_camera_off);
    //设置按钮背景图片
    cameraItemData.setBackground(R.drawable.tuiroomkit_bg_bottom_item_black);
    //设置按钮不可点击时的图标
    cameraItemData.setDisableIconId(R.drawable.tuiroomkit_ic_camera_off);
    //设置按钮默认图标
    cameraItemData.setName(mContext.getString(R.string.tuiroomkit_item_open_camera))

    //按钮点击效果，如果您的按钮点击时需要切换图片/名称等，需要构造此数据
    BottomSelectItemData camaraSelectItemData = new BottomSelectItemData();
    //设置按钮选中时的名称
    camaraSelectItemData.setSelectedName(mContext.getString(R.string.tuiroomkit_item
    //设置按钮未选中时的名称
    camaraSelectItemData.setUnSelectedName(mContext.getString(R.string.tuiroomkit_it
    //设置按钮是否选中
    camaraSelectItemData.setSelected(false);
    //设置按钮选中时的图标
    camaraSelectItemData.setSelectedIconId(R.drawable.tuiroomkit_ic_camera_on);
    //设置按钮未选中时的图标
    camaraSelectItemData.setUnSelectedIconId(R.drawable.tuiroomkit_ic_camera_off);
    //设置按钮选中/未选中时的点击事件
    camaraSelectItemData.setOnItemSelectListener(new BottomSelectItemData.OnItemSele
          @Override
          public void onItemSelected(boolean isSelected) {
             enableCamera(isSelected);
          }
    });
    cameraItemData.setSelectItemData(camaraSelectItemData);
    return cameraItemData;
    }
```

方案三：自定义全部 UI 方案

TUIRoomKit 的整体功能是基于 TUIRoomEngine 这个无 UI SDK实现的，您可以完全基于 TUIRoomEngine 实现一套自己的 UI 界面。详情可见TUIRoomEngine API 接口地址。

iOS

最近更新时间：2024-03-15 16:07:22

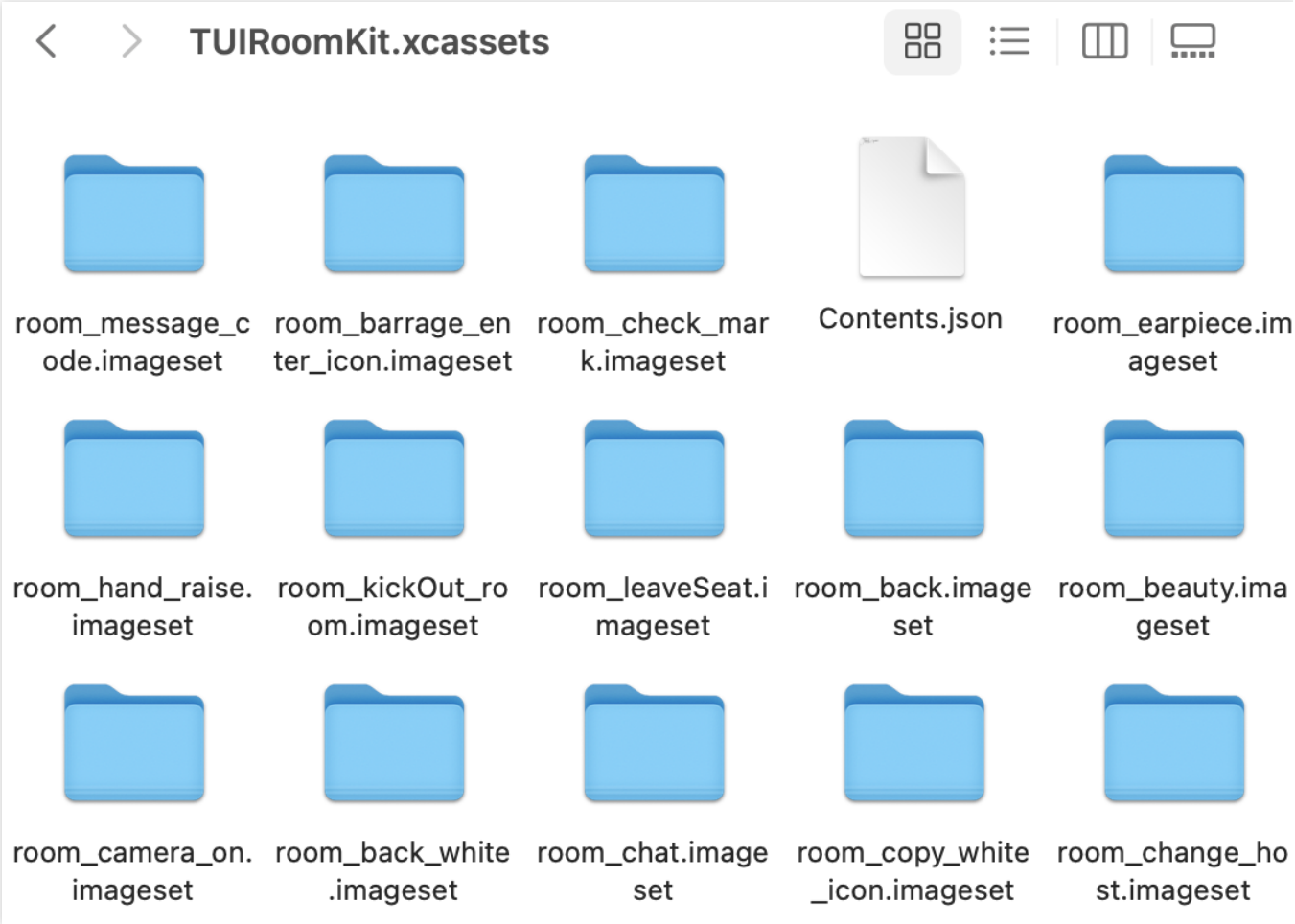
本文将介绍如何定制 TUIRoomKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自定义 UI 方案**。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUIRoomKit 的用户界面进行调整，TUIRoomKit 的界面源代码位于 Github 中的 iOS/ 文件夹下面：

替换图标

您可以直接修改 **TUIRoomKit/Resources/TUIRoomKit.xcassets** 文件夹下的图标组件，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



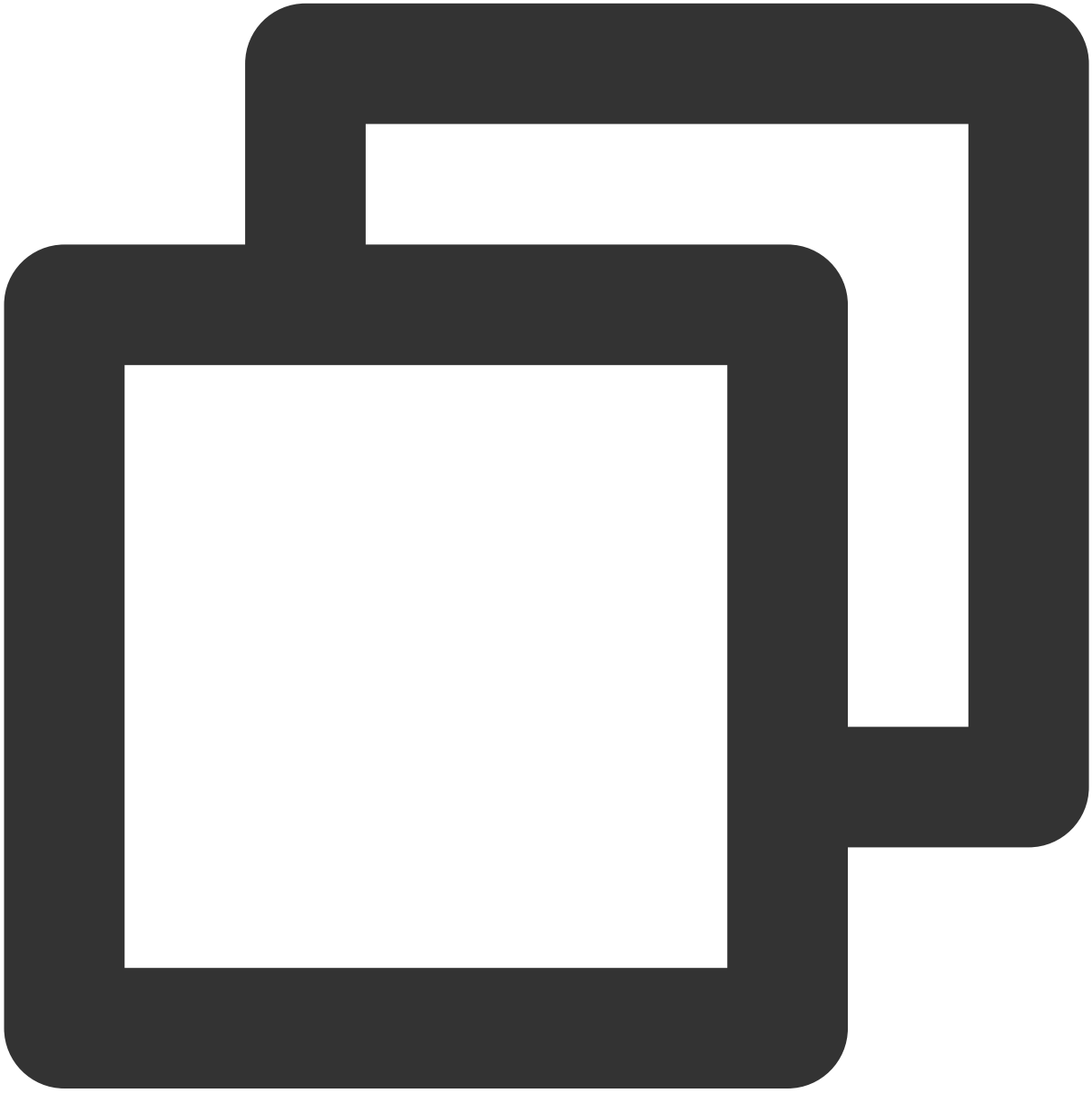
版权所有：腾讯云计算（北京）有限责任公司

替换文案

您可以通过修改 `TUIRoomKit/Resources/Localized` 文件中 values-zh 和 values-en 的 `strings.xml` 文件来修改视频会议界面的字符串内容。

调整 UI 布局

您可以通过修改 `TUIRoomKit/Source/View` 文件下的不同页面来调整多人视频会议界面的 UI 布局：

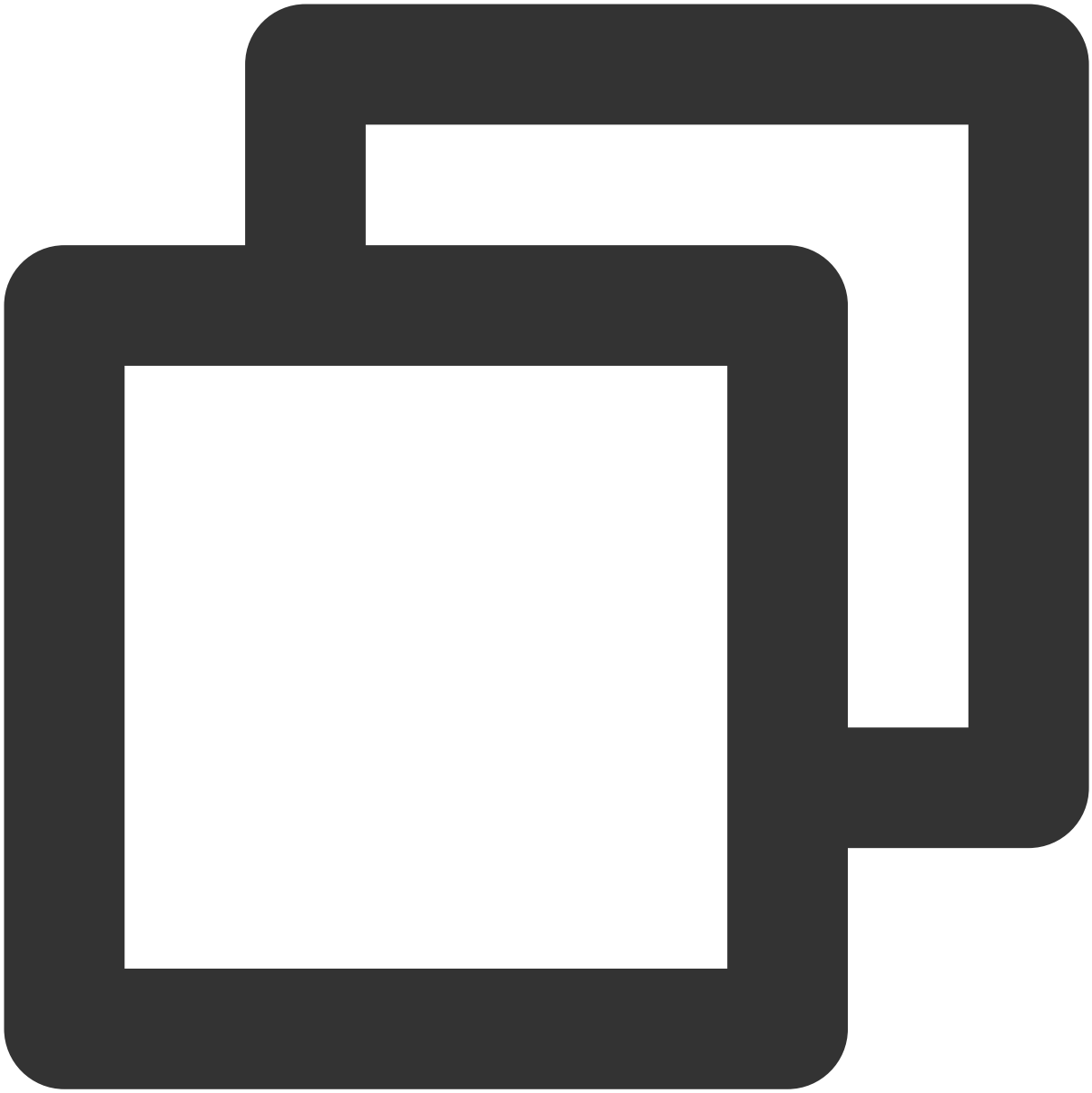


```
├── Component
│   ├── ButtonItemView.swift                    //通用View
│   └── ListCellItemView.swift                  //通用View
```

```
└── Page
    ├── RoomMainView.swift                          //会议主界面View
    ├── RoomMainViewController.swift                //会议主界面ViewController
    ├── RoomRouter.swift                            //路由
    └── Widget
        ├── BottomNavigationBar                     //底部栏
        │   ├── BottomItemView.swift
        │   └── BottomView.swift
        ├── FloatWindow                             //悬浮窗
        │   ├── RoomUserStatusView.swift
        │   └── RoomVideoFloatView.swift
        ├── MediaSettings                           //设置
        │   ├── MediaSettingView.swift
        │   └── VideoChoicePanel.swift
        ├── TopNavigationBar                        //顶部栏
        │   ├── TopItemView.swift
        │   └── TopView.swift
        ├── UserControlPanel                        //用户列表
        │   ├── UserListManagerView.swift
        │   └── UserListView.swift
        ├── RaiseHandControlPanel                   //举手用户管理列表
        │   └── RaiseHandApplicationListView.swift
        ├── TransferOwnerControlPanel               //房间转让列表
        │   └── TransferMasterView.swift
        ├── PopUpControlPanel                       //通用弹框
        │   ├── PopUpView.swift
        │   └── PopUpViewController.swift
        └── VideoSeat                               //视频显示
            ├── ScreenCaptureMaskView.swift
            ├── TUIVideoSeatCell.swift
            ├── TUIVideoSeatLayout.swift
            ├── TUIVideoSeatUserStatusView.swift
            └── TUIVideoSeatView.swift
```

底部按钮 BottomView 的修改

为了方便您自定义底部的功能按钮，我们的 BottomView 是通过读取 list 来自动构造，以开关视频按钮为例子，代码如下:



```
func createBottomData() {
        let muteVideoItem = ButtonItemData()
        //设置按钮默认标题
        muteVideoItem.normalTitle = .unMuteVideoText
        //设置按钮点击后标题
        muteVideoItem.selectedTitle = .muteVideoText
        //设置按钮默认图标
        muteVideoItem.normalIcon = "room_camera_on"
        //设置按钮点击后图标
        muteVideoItem.selectedIcon = "room_camera_off"
```

```
        //设置按钮图片资源获取处
        muteVideoItem.resourceBundle = tuiRoomKitBundle()
        //设置按钮是否点击
        muteVideoItem.isSelect = !(roomInfo.isOpenCamera)
        //设置按钮类型用于区分不同按钮
        muteVideoItem.buttonType = .muteVideoItemType
        //设置按钮点击事件
        muteVideoItem.action = { [weak self] sender in
            guard let self = self, let button = sender as? UIButton else { return }
            self.muteVideoAction(sender: button)
        }
}
```

方案二：自定义 UI 方案

TUIRoomKit 的整体功能是基于 TUIRoomEngine 这个无 UI SDK 实现的，您可以完全基于 TUIRoomEngine 实现一套自己的 UI 界面。详情可见：
TUIRoomEngine API 接口地址。

Windows

最近更新时间：2023-09-22 10:13:05

本文将介绍如何定制 TUIRoomKit 的用户界面，我们提供了两个方案供您选择：换肤方案和自定义 UI 方案。

方案一：界面微调方案

您可以直接基于我们提供的 App 进行修改适配
App 目录包含了 UI 相关的设计与逻辑，您可以根据需求，修改 RoomApp 源码进行二次开发，主要功能点如下：

功能点	文件目录
首页登录	Windows-Mac\\RoomApp\\App\\LoginViewController.cpp
设备检测	Windows-Mac\\RoomApp\\App\\PresetDeviceController.cpp
主页面	Windows-Mac\\RoomApp\\App\\MainWindow.cpp
麦上列表	Windows-Mac\\RoomApp\\App\\StageListController.cpp
成员列表	Windows-Mac\\RoomApp\\App\\MemberListViewController.cpp
设置页面	Windows-Mac\\RoomApp\\App\\SettingViewController.cpp
聊天室	Windows-Mac\\RoomApp\\App\\ChatRoomViewController.cpp
屏幕分享	Windows-Mac\\RoomApp\\App\\ScreenShareWindow.cpp
底部工具栏	Windows-Mac\\RoomApp\\App\\BottomBarController.cpp

方案二：自定义 UI 方案

TUIRoomKit 的整体功能是基于 TUIRoomEngine 这个无 UI SDK实现的，您可以完全基于 TUIRoomEngine 实现一套自己的 UI 界面。详情可见TUIRoomEngine API 接口地址。

版权所有：腾讯云计算（北京）有限责任公司

Web

最近更新时间：2023-09-22 10:14:29

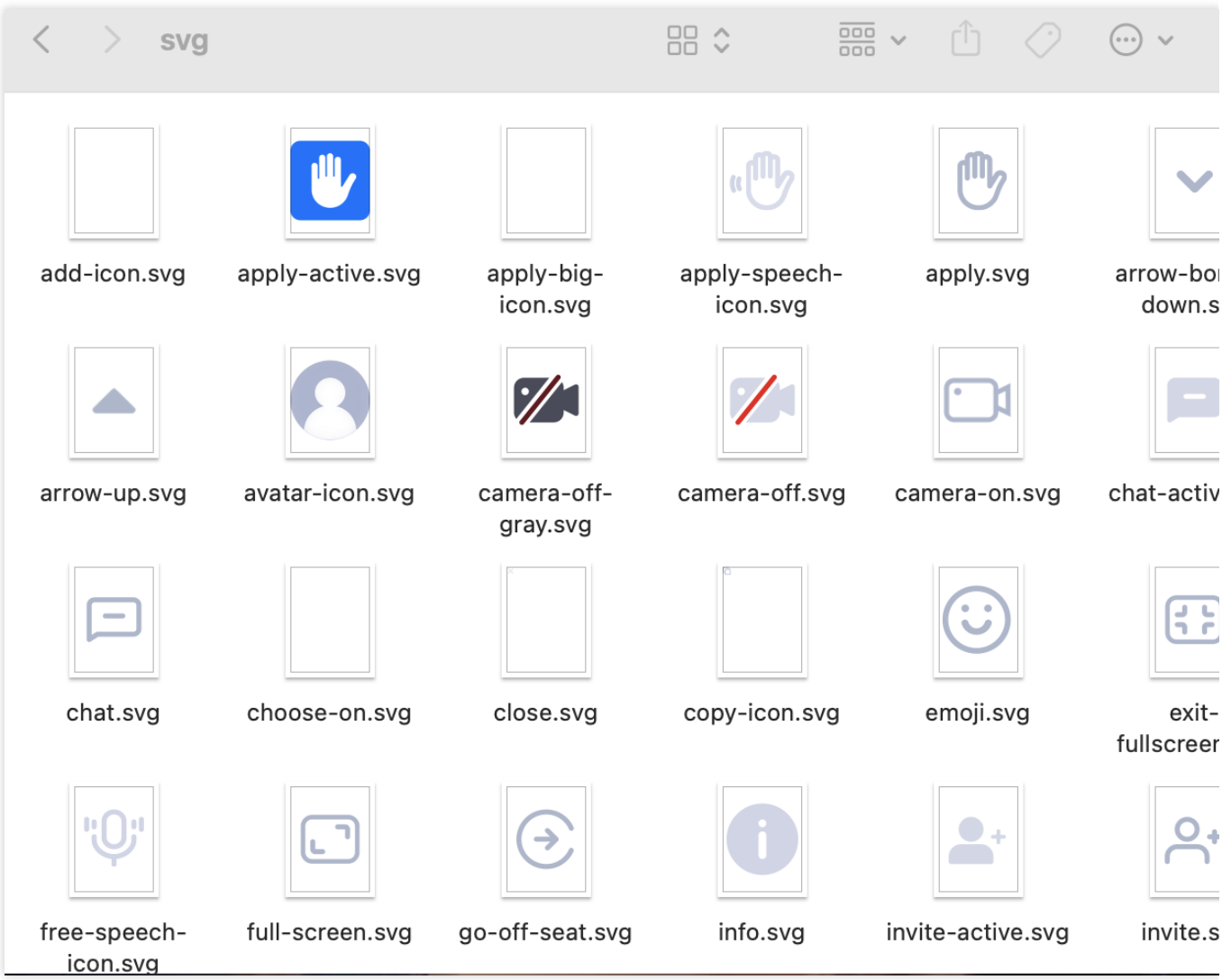
本文将介绍如何定制 TUIRoomkit 的用户界面，我们提供了两个方案供您选择：换肤方案和自定义 UI 方案。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUIRoom 的用户界面进行调整，TUIRoom 的界面源代码位于 Github 中的 Web/ 文件夹下面：

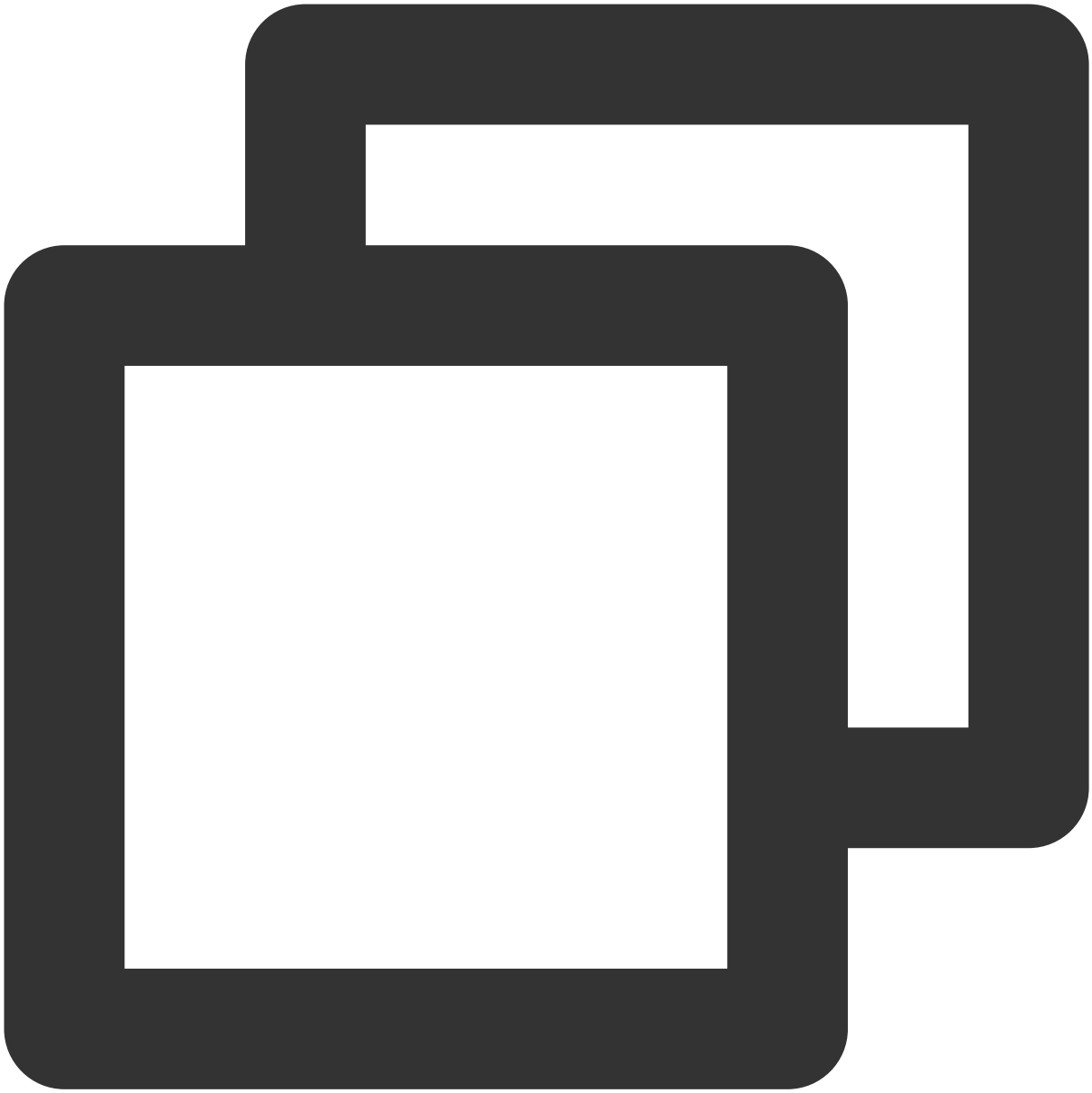
替换图标

您可以直接修改 **web/vue3/src/TUIRoom/assests/icons/svg** 文件夹下的图标组件，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



调整 UI 布局

您可以通过修改 src/components/ 文件下的不同页面来调整多人视频会议界面的UI布局



```
- components/
    - base
    - Chat                  聊天
    - common                公用图标组件
    - ManageMember          成员管理
    - RoomContent           Room 视频
    - RoomFooter            Room 页面 Footer部分
    - RoomHeader            Room 页面 Header部分
    - RoomHome              Home 页面
    - RoomInvite            邀请成员
    - RoomLogin             登录页面
```

```
- RoomMore              更多
- RoomSetting           设置
- RoomSidebar           抽屉组件
```

方案二：自定义 UI 方案

TUIRoom 的整体功能是基于 TUIRoomEngine 这个无 UI SDK实现的，您可以完全基于 TUIRoomEngine 实现一套自己的 UI 界面。详情可见

TUIRoomEngine 接入指引

TUIRoomEngine API 接口地址

Flutter

最近更新时间：2024-04-12 11:22:28

本文将介绍如何定制 `TUIRoomKit` 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

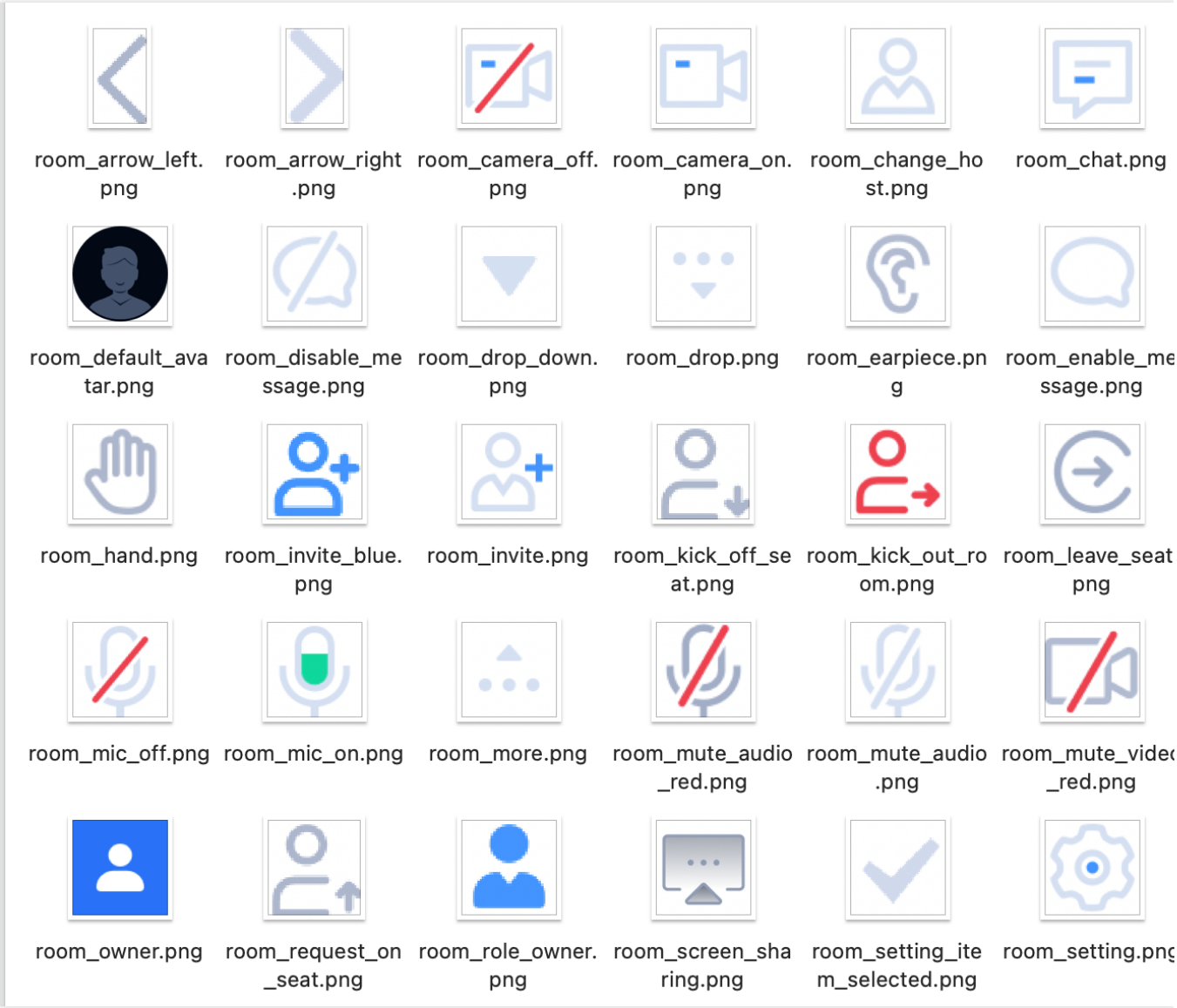
方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 `TUIRoomKit` 的用户界面进行调整。

替换图标

您可以直接替换 `rtc_conference_tui_kit/assets/images` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。

版权所有：腾讯云计算（北京）有限责任公司

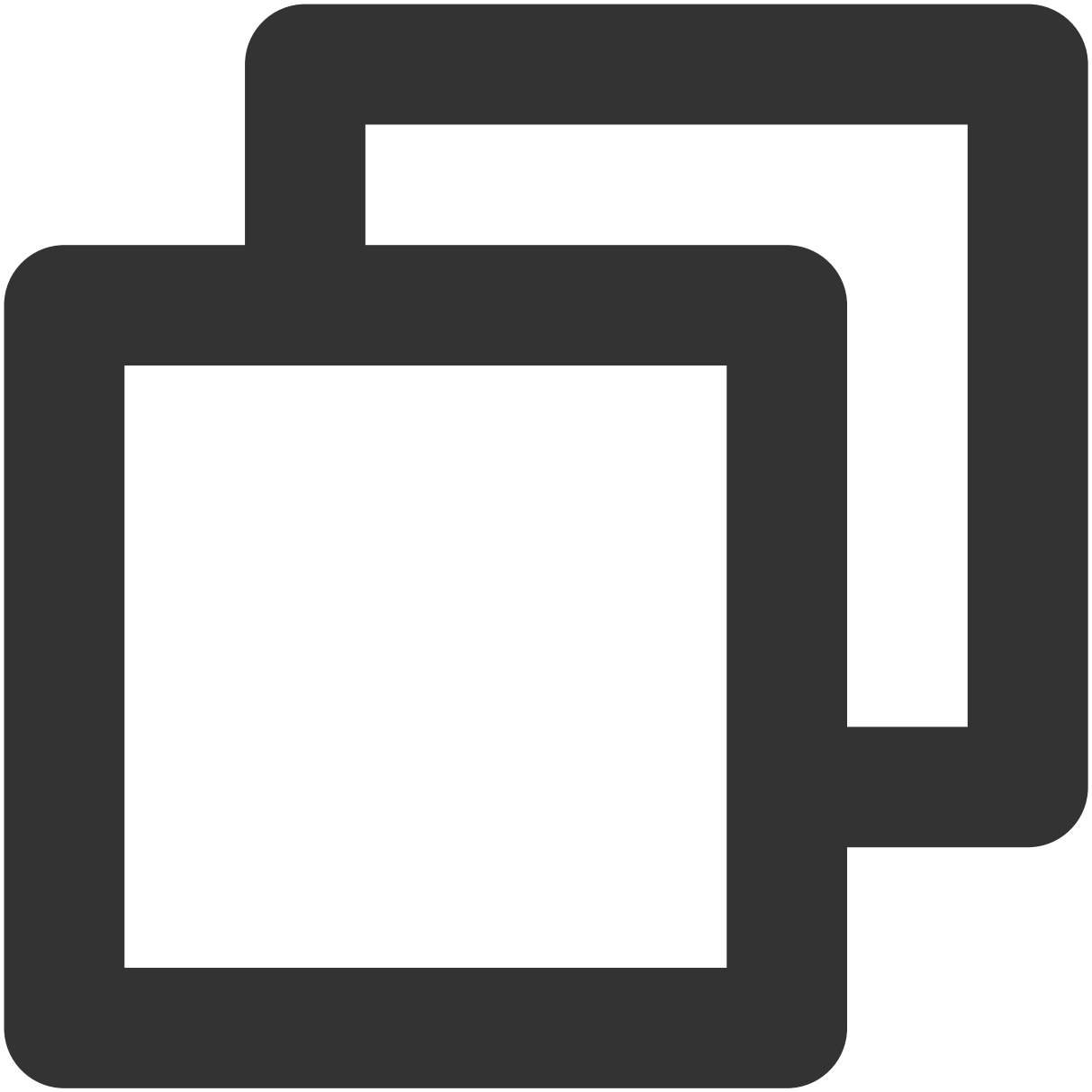


替换文案

您可以通过修改 `rtc_conference_tui_kit/lib/common/languages` 目录下的 `en_us.dart` 和 `zh_cn.dart` 文件来修改视频会议界面的字符串内容。

方案二：自定义部分 UI 方案

`TUIRoomKit` 的各 `UI` 组件文件夹如下，您可根据自身的实际业务需求来修改相应文件夹下的 `view.dart` 来自定义 `UI` 。



```
├── lib
├── common
│   └── widgets
│       ├── bottom_sheet.dart (1 KB)                    // 通用底部弹窗
│       ├── copy_text_button.dart (1 KB)                // 通用文本复制按钮
│       ├── dialog.dart (4 KB)                          // 通用对话框弹窗
│       ├── drop_down_button.dart (<1 KB)               // 通用下拉按钮
│       ├── info_list.dart (1 KB)                       // 通用信息展示item
│       ├── rounded_container.dart (<1 KB)              // 通用圆角container
│       ├── search_bar.dart (1 KB)                      // 通用搜索框
│       ├── single_select_list.dart (1 KB)              // 通用单选列表
```

```
│       ├── slider.dart (1 KB)                                  // 通用滑动条
│       ├── switch.dart (<1 KB)                                  // 通用开关按钮
│       ├── toast.dart (<1 KB)                                   // 通用toast提示
│       ├── user_info.dart (3 KB)                                // 通用用户信息item组件
│       └── volume_bar.dart (2 KB)                               // 通用动态麦克风音量条
├── pages
│   └── conference_main
│       ├── view.dart (3 KB)
│       └── widgets
│           ├── bottom_view
│           │   ├── view.dart (1 KB)                             // 底部功能按钮文件
│           │   └── widgets
│           │       ├── base_button.dart (10 KB)                 // 未展开的基础底部功能按钮
│           │       ├── bottom_button_item.dart (2 KB)           // 封装的通用底部单个item按钮
│           │       ├── mic_button.dart (1 KB)                   // 底部栏隐藏后弹出的麦克风按钮
│           │       └── more_button.dart (3 KB)                  // 展开后的所有底部功能按钮
│           ├── exit
│           │   └── view.dart (2 KB)                             // 退出房间底部弹窗
│           ├── invite_sheet
│           │   └── invite_sheet.dart (1 KB)                     // 邀请弹窗
│           ├── local_screen_sharing
│           │   └── local_screen_sharing.dart (1 KB)             // 本地开屏幕共享后屏幕中间屏幕
│           ├── raise_hand_list
│           │   ├── view.dart (3 KB)                             // 举手申请列表（尽在举手发言模
│           │   └── widgets
│           │       ├── title.dart (<1 KB)                       // 列表标题栏
│           │       ├── user_item.dart (1 KB)                    // 封装的列表组件的单个item
│           │       └── user_table.dart (1 KB)                   // 列表组件
│           ├── setting
│           │   ├── view.dart (1 KB)                             // 设置面板
│           │   └── widgets
│           │       ├── audio_setting.dart (1 KB)。              // 音频设置组件
│           │       ├── setting_info_select.dart (1 KB)。        // 单选值设置组件，如分辨率设置
│           │       ├── setting_item.dart (<1 KB)。              // 设置面板单个设置项item
│           │       └── video_setting.dart (4 KB)                // 视频设置
│           ├── top_view
│           │   ├── view.dart (3 KB)                             // 顶部功能按钮
│           │   └── widgets
│           │       ├── meeting_title.dart (2 KB)                // 顶部会议信息显示组件
│           │       ├── room_info_sheet.dart (2 KB)。            // 详细会议信息底部弹窗
│           │       └── top_button_item.dart (1 KB)              // 顶部功能单个通用item按钮
│           ├── transfer_host
│           │   ├── view.dart (1 KB)                             // 转让房主面板
│           │   └── widgets
│           │       ├── title.dart (<1 KB)                       // 转让房主面板标题栏
│           │       ├── user_item.dart (1 KB)                    // 可转让的房主用户列表单个ite
│           │       └── user_table.dart (1 KB)                   // 可转让的房主用户列表
```

```
│           ├── user_list
│           │   ├── view.dart (3 KB)                          // 成员列表
│           │   └── widgets
│           │       ├── button_item.dart (1 KB)               // 成员列表底部按钮item
│           │       ├── user_control.dart (7 KB)              // 点击单个成员后弹出的成员管理
│           │       ├── user_control_item.dart (1 KB)         // 成员管理页面item
│           │       ├── user_item.dart (3 KB)                 // 成员列表item
│           │       └── user_table.dart (3 KB)                // 成员列表页面列表
│           ├── video_seat
│           │   ├── video_layout
│           │   │   ├── view.dart (3 KB)                      // 单页视频画面中视频item的布局
│           │   │   └── widgets
│           │   │       ├── video_item                        // 单个视频画面item文件夹
│           │   │       │   ├── view.dart (4 KB)              // 单个视频画面item
│           │   │       │   └── widgets
│           │   │       │       ├── video_user_info.dart (2 KB)       // 视频画面item
│           │   │       │       └── volume_bar.dart (<1 KB)           // 用户信息中的表
│           │   │       └── with_draggable_window_widget.dart (1 KB) // 两人视频画面大
│           │   └── video_page_turning
│           │       ├── view.dart (2 KB)                      // 视频画面翻页管
│           │       └── widgets
│           │           └── page_indicator.dart (1 KB)        // 页面指示器小圆
```

方案三：自定义全部 UI 方案

`TUIRoomKit` 的整体功能是基于 `TUIRoomEngine` 这个无 UI SDK 实现的，您可以完全基于 `TUIRoomEngine` 实现一套自己的 UI 界面。详情可见TUIRoomEngine API 接口地址。

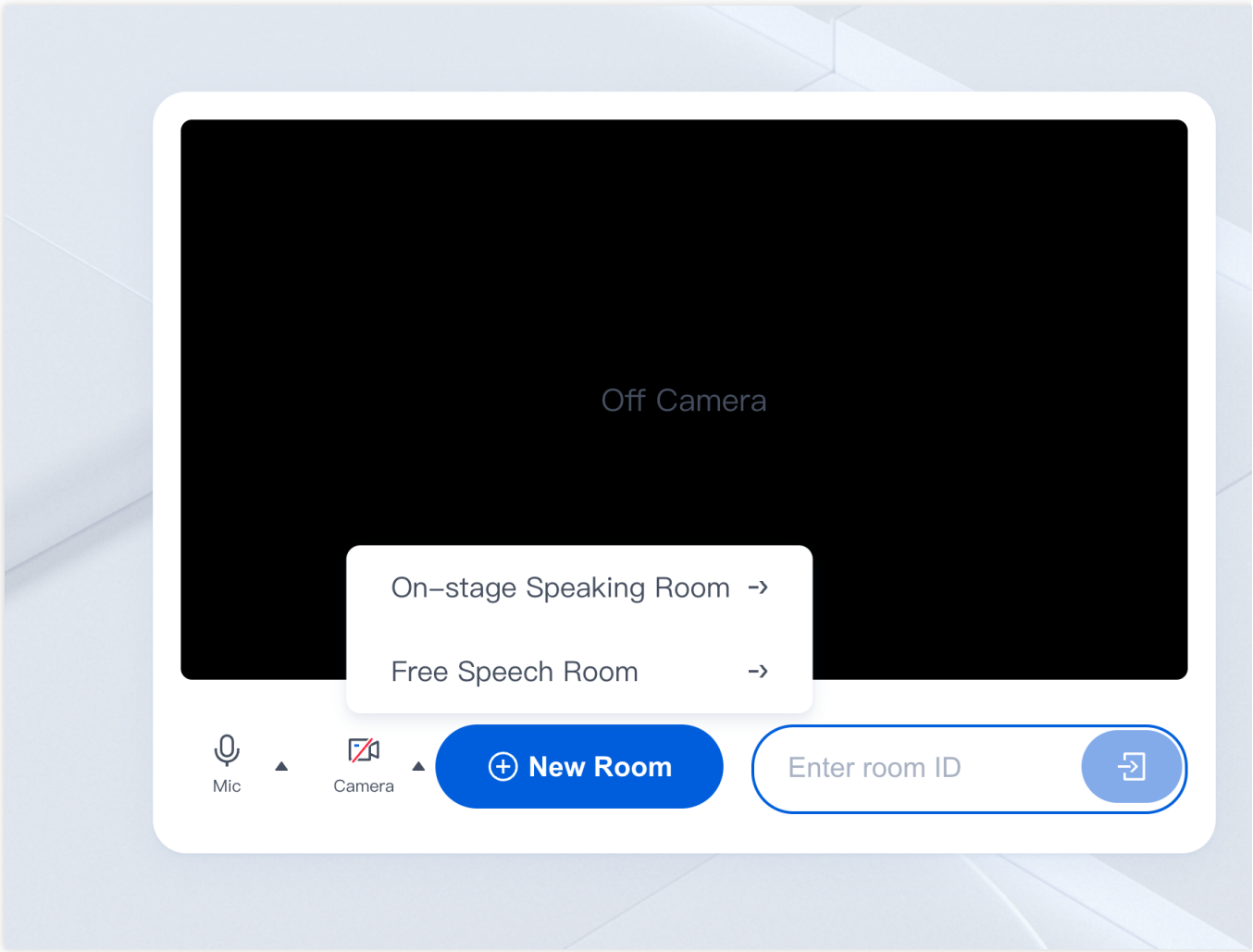
会议控制 (TUIRoomKit)

最近更新时间：2024-04-12 11:21:34

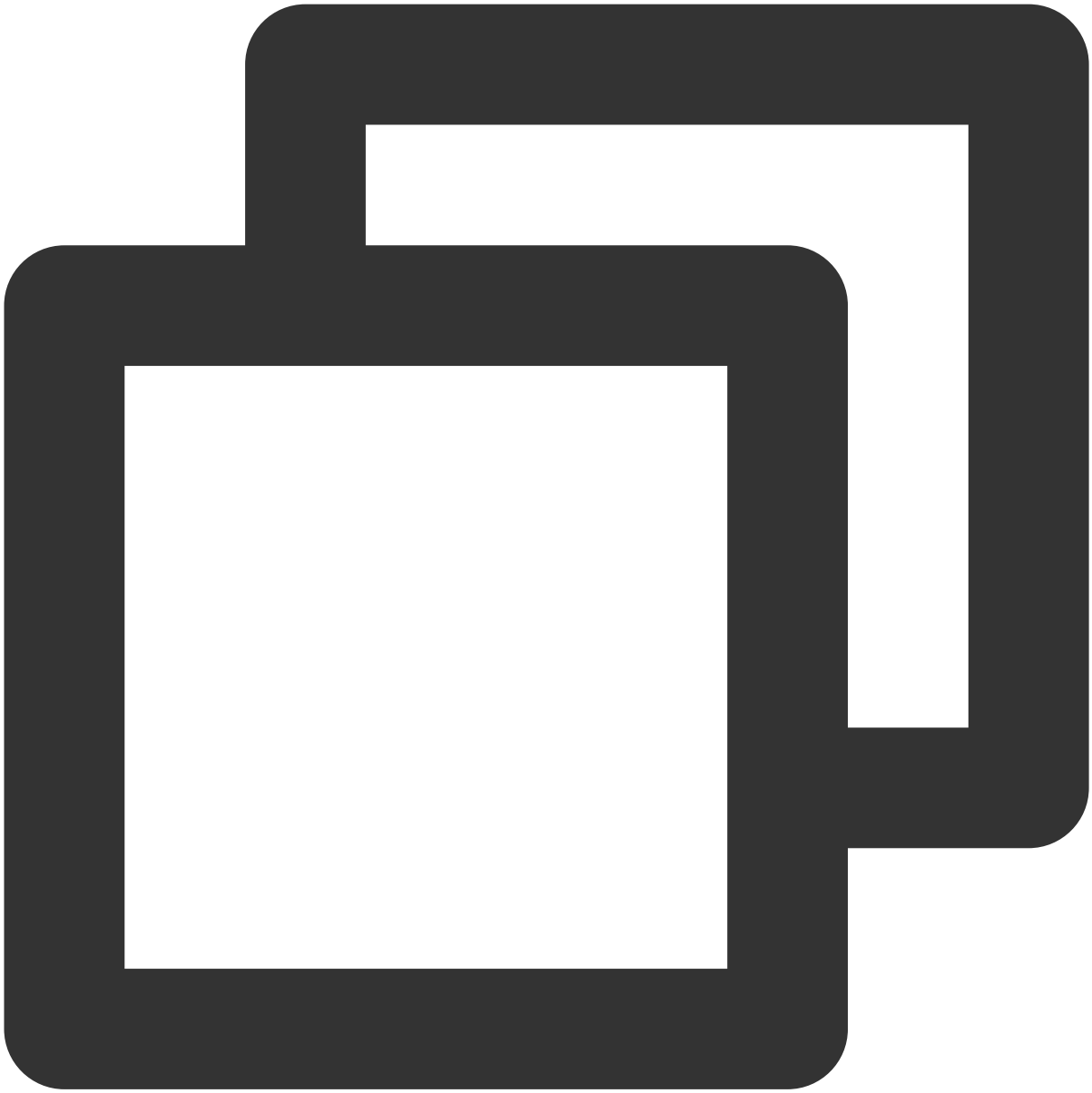
本文档将详细介绍 `TUIRoomKit` 的会前控制、会中控制等方面的内容，以帮助您更好地掌握 `TUIRoomKit` 的会议控制相关功能。通过本文档，您能够充分利用 `TUIRoomKit` 的功能，实现高质量的音视频会议。

会前控制

在进入会议之前，通过 `TUIRoomKit` 会前控制的相关功能，您可以预先设定会议的相关参数，确保会议顺利进行。

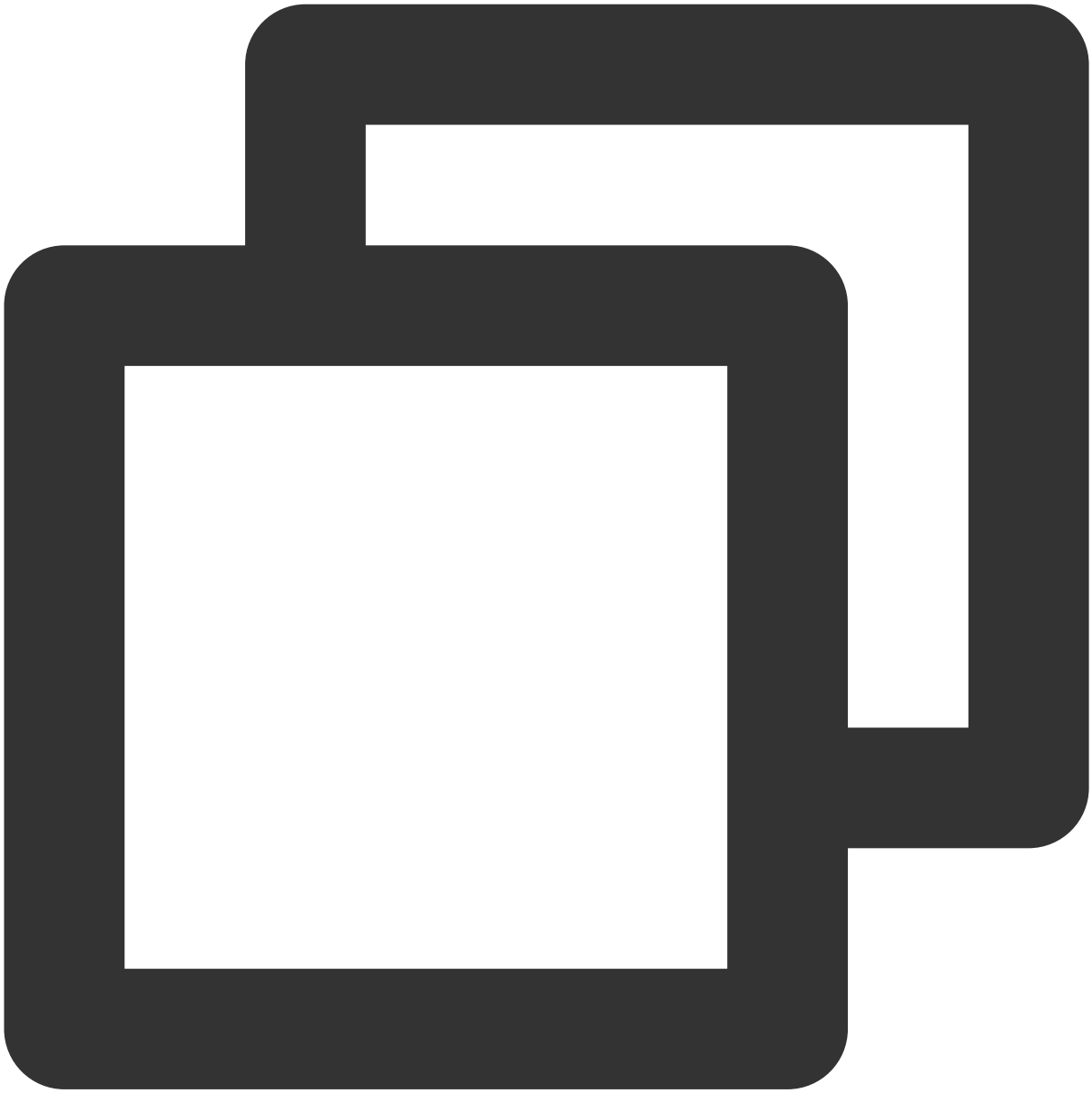


iOS (Swift)
Android（Java）
Flutter (Dart)



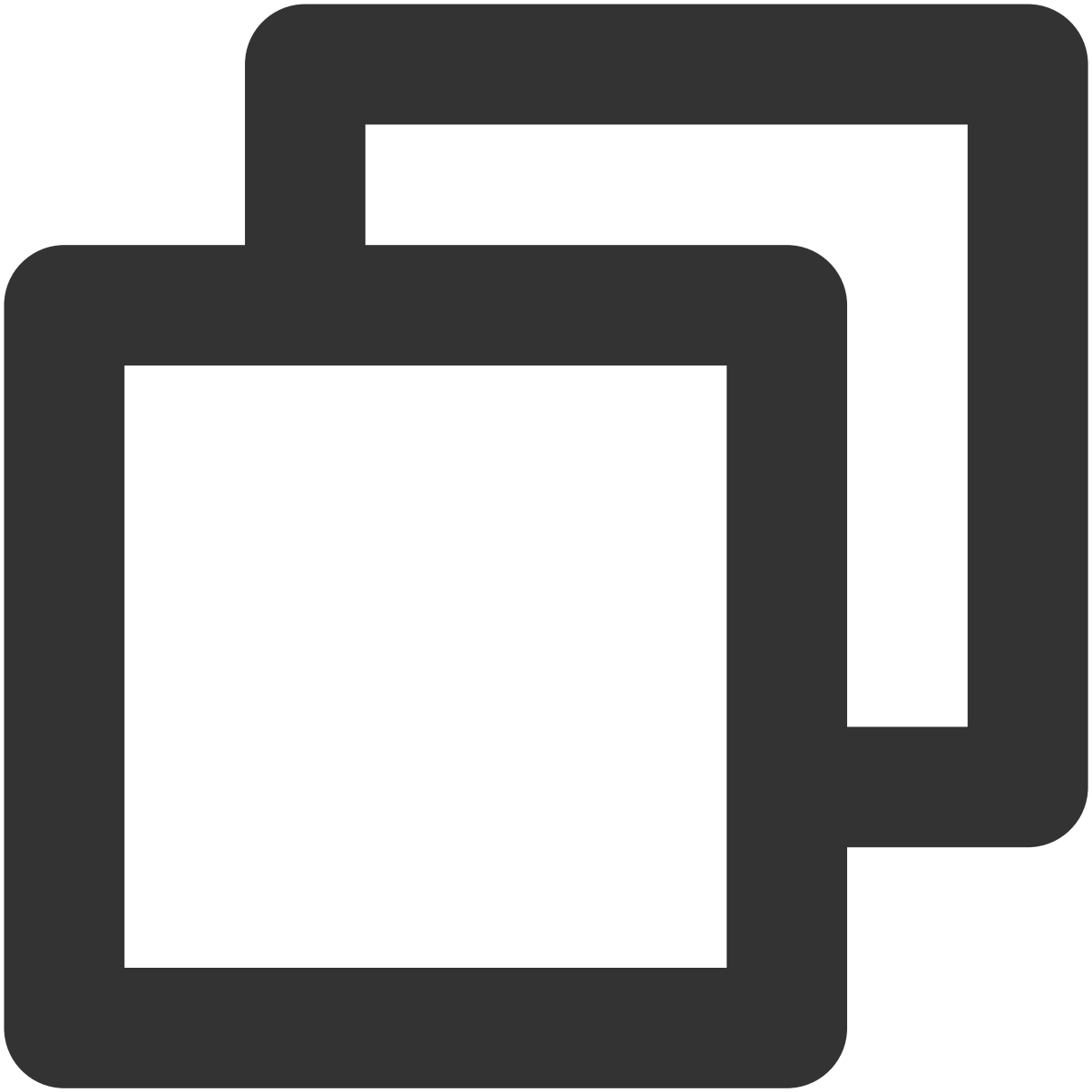
```
// CreateRoomViewController 为您自己的 ViewController
class CreateConferenceViewController: UIViewController {
    private var conferenceViewController: ConferenceMainViewController?
    func quickStartConferenceAction() {
      conferenceViewController = ConferenceMainViewController()
      // 通过设置ConferenceParams中的参数来实现会前控制的相关功能
      let params = ConferenceParams()
      params.isMuteMicrophone = false
      params.isOpenCamera = false
      params.isSoundOnSpeaker = true
      params.name = "您的会议名称"
```

```
        params.enableMicrophoneForAllUser = true
        params.enableCameraForAllUser = true
        params.enableMessageForAllUser = true
        params.enableSeatControl = false
        conferenceViewController?.setConferenceParams(params: params)
        conferenceViewController?.setConferenceObserver(observer: self)
        //设置完成后再调用开始会议或加入会议的接口,这里以开始会议为例
        conferenceViewController?.quickStartConference(conferenceId: "您的conferenceId
    }
}

extension CreateConferenceViewController: ConferenceObserver {
    func onConferenceStarted(conferenceId: String, error: ConferenceError) {
        if error == .success, let vc = conferenceViewController {
            navigationController?.pushViewController(vc, animated: true)
        }
        conferenceViewController = nil
    }
}
```



```
public class ConferenceOwnerActivity extends AppCompatActivity {
    private static final String TAG = "ConferenceOwnerActivity";

    private ConferenceObserver mConferenceObserver;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_conference_main);
        // 通过设置ConferenceParams中的参数来实现会前控制的相关功能
        ConferenceParams params = new ConferenceParams();
        params.setMuteMicrophone(false);
```

```
        params.setOpenCamera(false);
        params.setSoundOnSpeaker(true);
        params.setName("您的会议名称");
        params.setEnableMicrophoneForAllUser(true);
        params.setEnableCameraForAllUser(true);
        params.setEnableMessageForAllUser(true);
        params.setEnableSeatControl(false);
        ConferenceMainFragment fragment = new ConferenceMainFragment();
        fragment.setConferenceParams(params);
        setConferenceObserver(fragment);
        fragment.quickStartConference("您的conferenceId");  //设置完成后再调用开始会议或
    }

    private void setConferenceObserver(ConferenceMainFragment fragment) {
        mConferenceObserver = new ConferenceObserver() {
            @Override
            public void onConferenceStarted(String conferenceId, ConferenceError er
                super.onConferenceStarted(conferenceId, error);
                if (error != ConferenceError.SUCCESS) {
                    Log.e(TAG, "Error : " + error);
                    return;
                }
                FragmentManager manager = getSupportFragmentManager();
                FragmentTransaction transaction = manager.beginTransaction();
                transaction.add(R.id.conference_owner_container, fragment);
                transaction.commitAllowingStateLoss();
            }
        };
        fragment.setConferenceObserver(mConferenceObserver);
    }
}
```



```
var conferenceSession = ConferenceSession.newInstance("您的conferenceId")
      ..isMuteMicrophone = false
      ..isOpenCamera = false
      ..isSoundOnSpeaker = true
      ..name = "您的会议名称"
      ..enableMicrophoneForAllUser = true
      ..enableCameraForAllUser = true
      ..enableMessageForAllUser = true
      ..enableSeatControl = false
      ..onActionSuccess = () {     //操作成功的回调,您可以在这里导航至会议界面
        Navigator.push(
```

```
            context,
            MaterialPageRoute(
            builder: (context) => ConferenceMainPage(),
            ),
          );
        }
        ..onActionError = (ConferenceError error, String message) {}  //操作失败的回调
        ..quickStart();                                  //设置完成后再调用开始会议或加入会议的接口,这
```

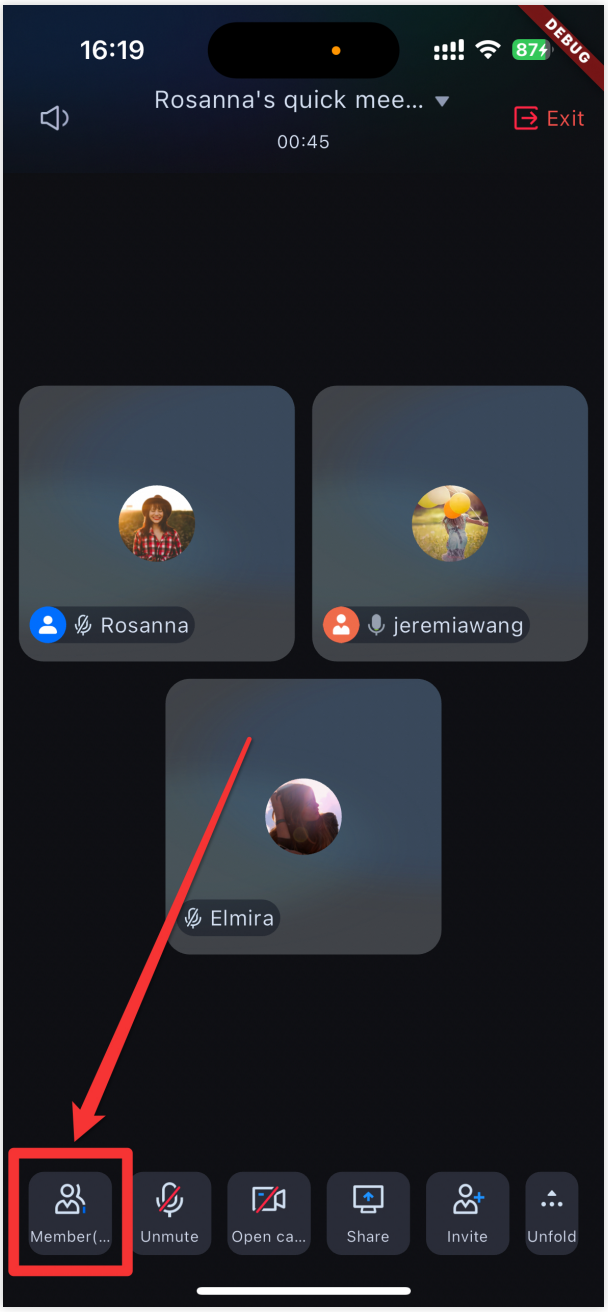
以下是对上述代码中参数的详细介绍。

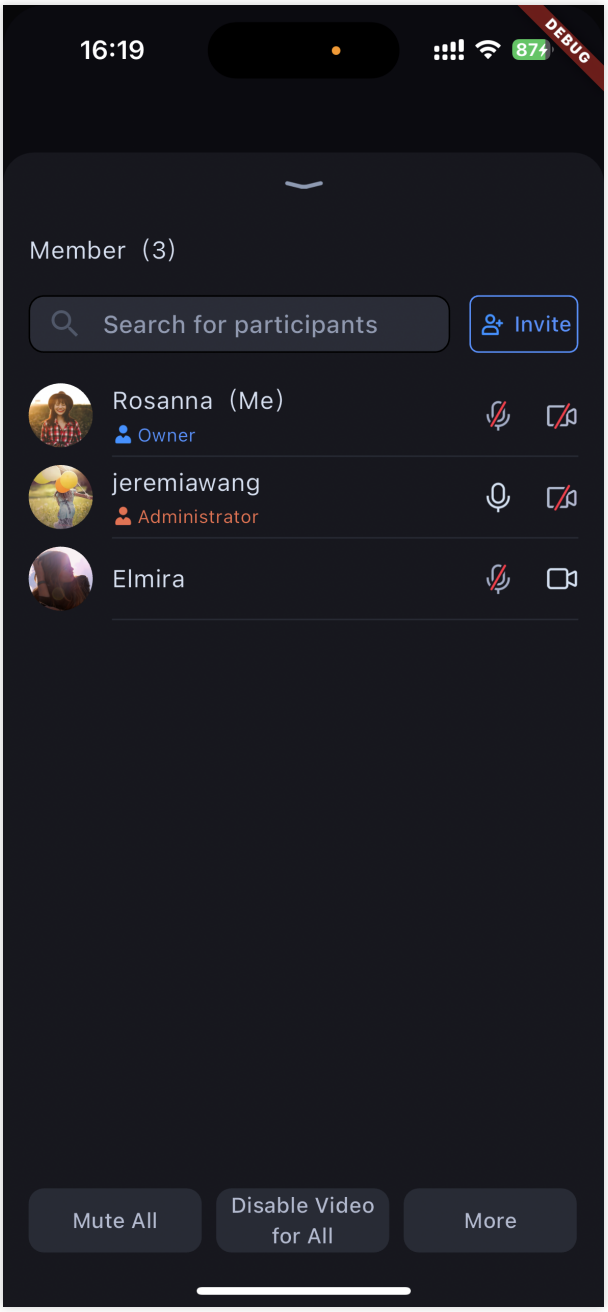
字段	类型	含义
isMuteMicrophone	bool	是否静音麦克风（默认为false）
isOpenCamera	bool	是否开启摄像头（默认为false）
isSoundOnSpeaker	bool	是否使用扬声器（默认为true）
name	String	会议名称（默认为您的conferenceId）
enableMicrophoneForAllUser	bool	是否开启全员的麦克风权限（默认为true）
enableCameraForAllUser	bool	是否开启全员的摄像头权限（默认为true）
enableMessageForAllUser	bool	是否开启全员的发言权限（默认为true）
enableSeatControl	bool	是否开启上台发言模式（默认为false）

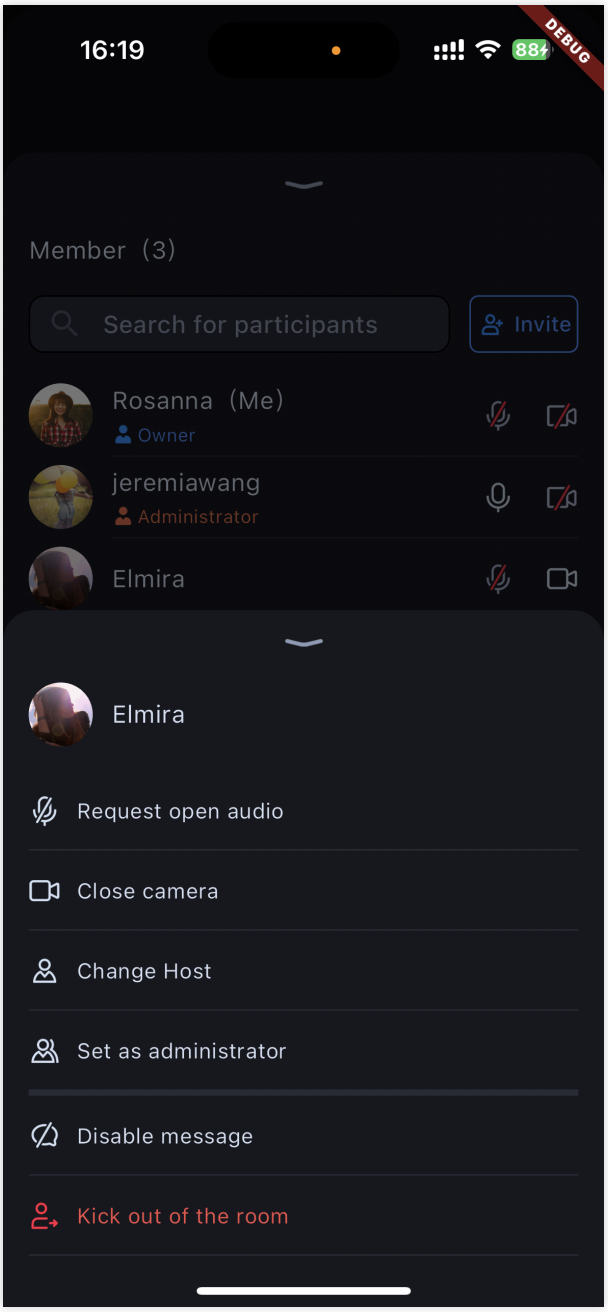
会中控制

在会议进行的过程中，您可能需要随时对会议进行控制以满足不同场景下的需求。`TUIRoomKit` 提供了丰富的会中控制功能，让您可以轻松应对各种会议情况。通过本节的介绍，您将直观地了解到如何在会议进行中进行有效的会议控制。

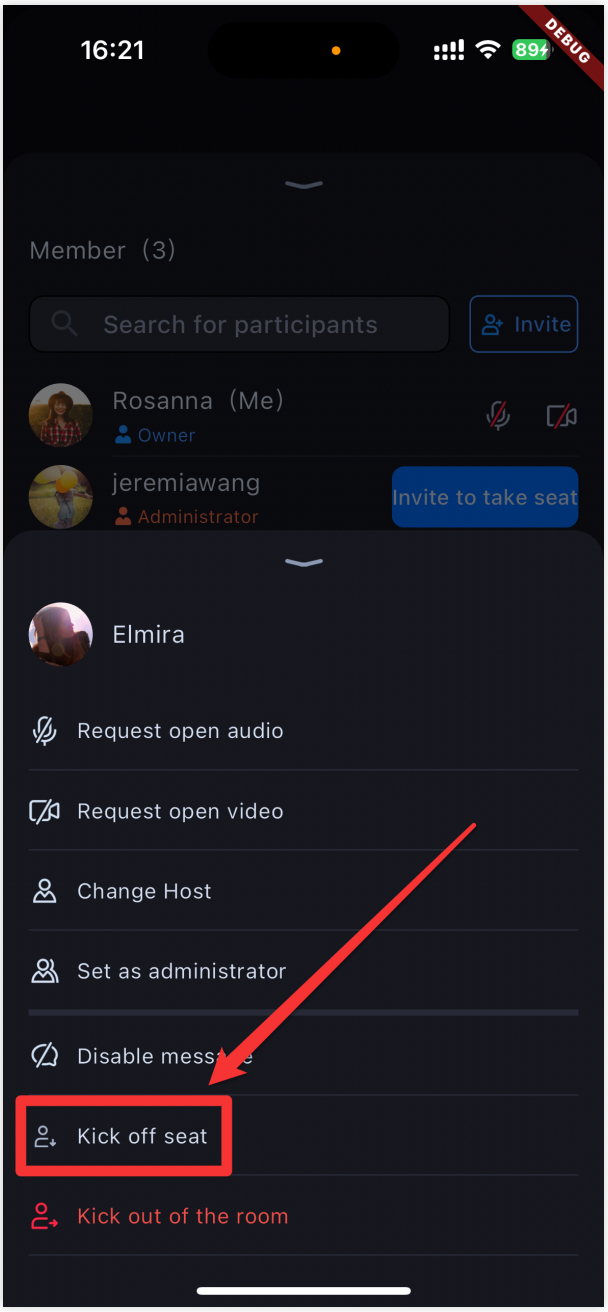
当您为房主或管理员时，点击左下角 `成员` 按钮，打开成员列表即可进行相关会议控制，如下图（左二）所示。在成员列表中，点击需要操作的用户，会出现如下图（右一）所示界面。

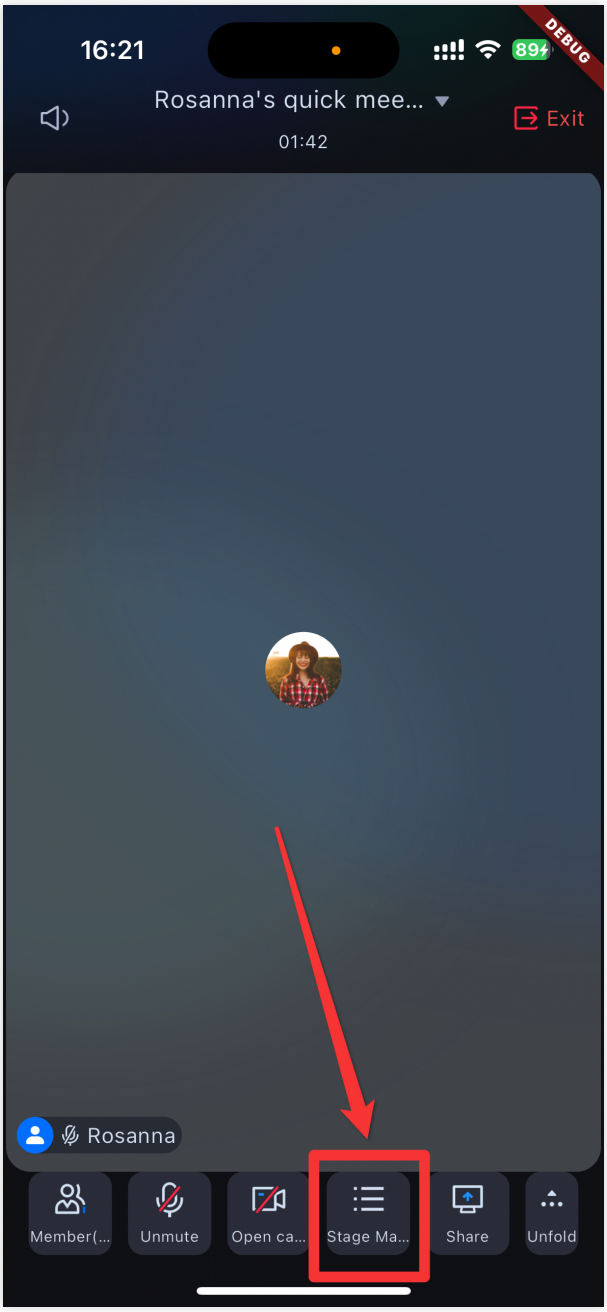

16:19
DEBUG
Rosanna's quick mee...
00:45
Exit
Rosanna
jeremiawang
Elmira
Member(...
Unmute
Open ca...
Share
Invite
Unfold

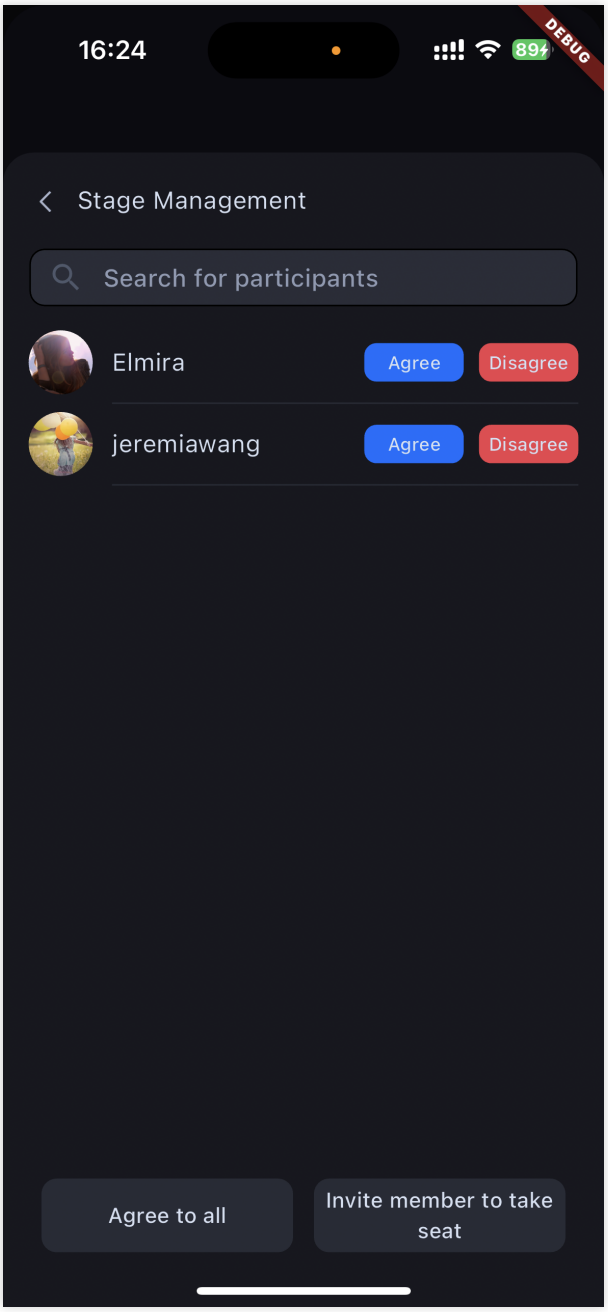

16:19
DEBUG
Member（3）
Search for participants
Invite
Rosanna（Me）
Owner
jeremiawang
Administrator
Elmira
Mute All
Disable Video for All
More



在上台发言模式下，点击成员列表中需要操作的用户，出现的界面如下图（左一）所示。点击会议底部工具栏的 `上台管理` 按钮来打开上台管理界面，如下图（右一）所示。


16:21
DEBUG
Member（3）
Search for participants
Invite
Rosanna（Me）
Owner
jeremiawang
Administrator
Invite to take seat
Elmira
Request open audio
Request open video
Change Host
Set as administrator
Disable mess
Kick off seat
Kick out of the room


16:21
89
DEBUG
Rosanna's quick mee...
Exit
01:42
Rosanna
Member(...
Unmute
Open ca...
Stage Ma...
Share
Unfold



功能	对应接口（以 Android 为例）
全员静音/禁画	disableDeviceForAllUserByAdmin
关闭/请求开启用户的音频/视频	closeRemoteDeviceByAdmin / openRemoteDeviceByAdmin
转交房主	changeUserRole
设置/取消管理员	changeUserRole
禁言/解除禁言	disableSendingMessageByAdmin

踢出房间	kickRemoteUserOutOfRoom
邀请上台/请下台	takeUserOnSeatByAdmin / kickUserOffSeatByAdmin
同意/拒绝用户的上台申请	responseRemoteRequest

API 参考 (TUIRoomKit)

iOS&Mac

API 概览

最近更新时间：2023-10-20 10:14:15

TUIRoomEngine API 列表

TUIRoomEngine API 是音视频通话组件的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。

TUIRoomEngine

TUIRoomEngine 核心方法

API	描述
init	创建 TUIRoomEngine 实例。
login	登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。
logout	退出登录接口，会有主动离开房间操作、销毁资源。
setSelfInfo	设置本地用户名称和头像。
getSelfInfo	获取本地用户登录的基本信息。
addObserver	设置事件回调。
removeObserver	移除事件回调。

房间相关主动接口

API	描述
createRoom	创建房间。
destroyRoom	解散房间。
enterRoom	进入房间。
exitRoom	离开房间。
connectOtherRoom	连接其他房间。

disconnectOtherRoom	断开与其他房间的连接。
fetchRoomInfo	获取房间信息。
updateRoomNameByAdmin	更新房间名称（只有管理员或群主能够调用）。
updateRoomSpeechModeByAdmin	设置房间麦控模式（只有管理员或群主能够调用）。

本地用户视图渲染、视频管理

API	描述
setLocalVideoView	设置本地用户视频渲染的控件。
openLocalCamera	打开本地摄像头。
closeLocalCamera	关闭本地摄像头。
updateVideoQuality	更新本地视频编码质量设置。
startPushLocalVideo	开始推送本地视频。
stopPushLocalVideo	停止推送本地视频。
startScreenCapture	开始屏幕分享（该接口仅支持移动端）。
startScreenCapture	开始屏幕分享（该接口仅支持桌面端 Mac OS 系统）。
stopScreenCapture	结束屏幕分享。
getScreenCaptureSources	枚举可分享的屏幕和窗口（该接口仅支持 Mac OS 系统）。
selectScreenCaptureTarget	选取要分享的屏幕或窗口（该接口仅支持 Mac OS 系统）。

本地用户音频管理

API	描述
openLocalMicrophone	打开本地麦克风。
closeLocalMicrophone	关闭本地麦克风。
updateAudioQuality	更新本地音频编码质量设置。
startPushLocalAudio	开始推送本地音频。

版权所有：腾讯云计算（北京）有限责任公司

stopPushLocalAudio	停止推送本地音频。

远端用户视图渲染、视频管理

API	描述
setRemoteVideoView	设置远端用户视频渲染的控件。
startPlayRemoteVideo	开始播放远端用户视频。
stopPlayRemoteVideo	停止播放远端用户视频。
muteRemoteAudioStream	将远端用户禁音。

房间内用户信息

API	描述
getUserList	获取房间内的成员列表。
getUserInfo	获取成员信息。

房间内用户管理

API	描述
changeUserRole	修改用户角色（只有管理员或群主能够调用）。
kickRemoteUserOutOfRoom	将远端用户踢出房间（只有管理员或群主能够调用）。

房间内用户发言管理

API	描述
disableDeviceForAllUserByAdmin	控制当前房间内所有用户是否可打开音频流、视频流采集设备的权限状态，例如：全员禁止打开麦克风、全员禁止打开摄像头、全员禁止打开屏幕分享（目前仅会议场景下可用，并且只有管理员或群主能够调用）。
openRemoteDeviceByAdmin	请求远端用户打开媒体设备（只有管理员或群主能够调用）。
closeRemoteDeviceByAdmin	关闭远端用户媒体设备（只有管理员或群主能够调

版权所有：腾讯云计算（北京）有限责任公司

	用）。
applyToAdminToOpenLocalDevice	请求打开本地媒体设备（普通用户可用）。

房间内麦位管理

API	描述
setMaxSeatCount	设置最大麦位数（仅支持进房前和创建房间时设置）。
getSeatList	获取麦位列表。
lockSeatByAdmin	锁定麦位（只有管理员或群主能够调用，包括位置锁定、音频状态锁定和视频状态锁定）。
takeSeat	申请上麦（自由发言模式下，不需要申请）。
leaveSeat	申请下麦（自由发言模式下，不需要申请）。
takeUserOnSeatByAdmin	主持人/管理员 邀请用户上麦。
kickUserOffSeatByAdmin	主持人/管理员 将用户踢下麦。

信令管理

API	描述
cancelRequest	取消请求。
responseRemoteRequest	回复请求。

发送消息

API	描述
sendTextMessage	发送本文消息。
sendCustomMessage	发送自定义消息。
disableSendingMessageByAdmin	禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）。
disableSendingMessageForAllUser	禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）。

高级功能：获取 TRTC 实例

版权所有：腾讯云计算（北京）有限责任公司

API	描述
getDeviceManager	获取原生 TRTC 设备管理类。
getAudioEffectManager	获取原生 TRTC 音效类。
getBeautyManager	获取原生 TRTC 美颜类。
getTRTCCloud	获取原生 TRTC 实例类。

TUIRoomObserver 回调事件

TUIRoomObserver 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己需要的回调事件。
TUIRoomObserver

TUIRoomObserver

错误回调

API	描述
onError	错误事件回调。

登录状态事件回调

API	描述
onKickedOffLine	其他终端登录被踢下线。
onUserSigExpired	用户凭证超时事件。

房间内事件回调

API	描述
onRoomNameChanged	房间名称更改事件。
onAllUserMicrophoneDisableChanged	房间内所有用户麦克风被禁用事件。
onAllUserCameraDisableChanged	房间内所有用户摄像头被禁用事件。

onSendMessageForAllUserDisableChanged	房间内所有用户发送文本消息被禁用事件。
onKickedOutOfRoom	被踢出房间事件。
onRoomDismissed	房间被解散事件。
onRoomSpeechModeChanged	房间麦控模式发生变化

房间内用户事件回调

API	描述
onRemoteUserEnterRoom	远端用户进房事件。
onRemoteUserLeaveRoom	远端用户离开房间事件。
onUserRoleChanged	用户角色发生变化事件。
onUserVideoStateChanged	用户视频状态发生变化事件。
onUserAudioStateChanged	用户音频状态发生变化事件。
onUserScreenCaptureStopped	用户屏幕采集停止事件。
onUserVoiceVolumeChanged	用户音量变化事件。
onSendMessageForUserDisableChanged	用户文本消息发送能力发生变化事件
onUserNetworkQualityChanged	用户网络状态变化事件。

房间内麦位事件回调

API	描述
onRoomMaxSeatCountChanged	房间内最大麦位数发生变化事件（仅在会议类型房间生效）
onSeatListChanged	麦位列表发生变化事件。
onKickedOffSeat	收到用户被踢下麦事件。

请求信令事件回调

API	描述
onRequestReceived	收到请求消息事件。

版权所有：腾讯云计算（北京）有限责任公司

onRequestCancelled	收到请求被取消的事件。

房间内消息事件回调

API	描述
onReceiveTextMessage	收到普通文本消息事件。
onReceiveCustomMessage	收到自定义消息事件。

TUIRoomKit

最近更新时间：2024-03-15 16:08:25

简介

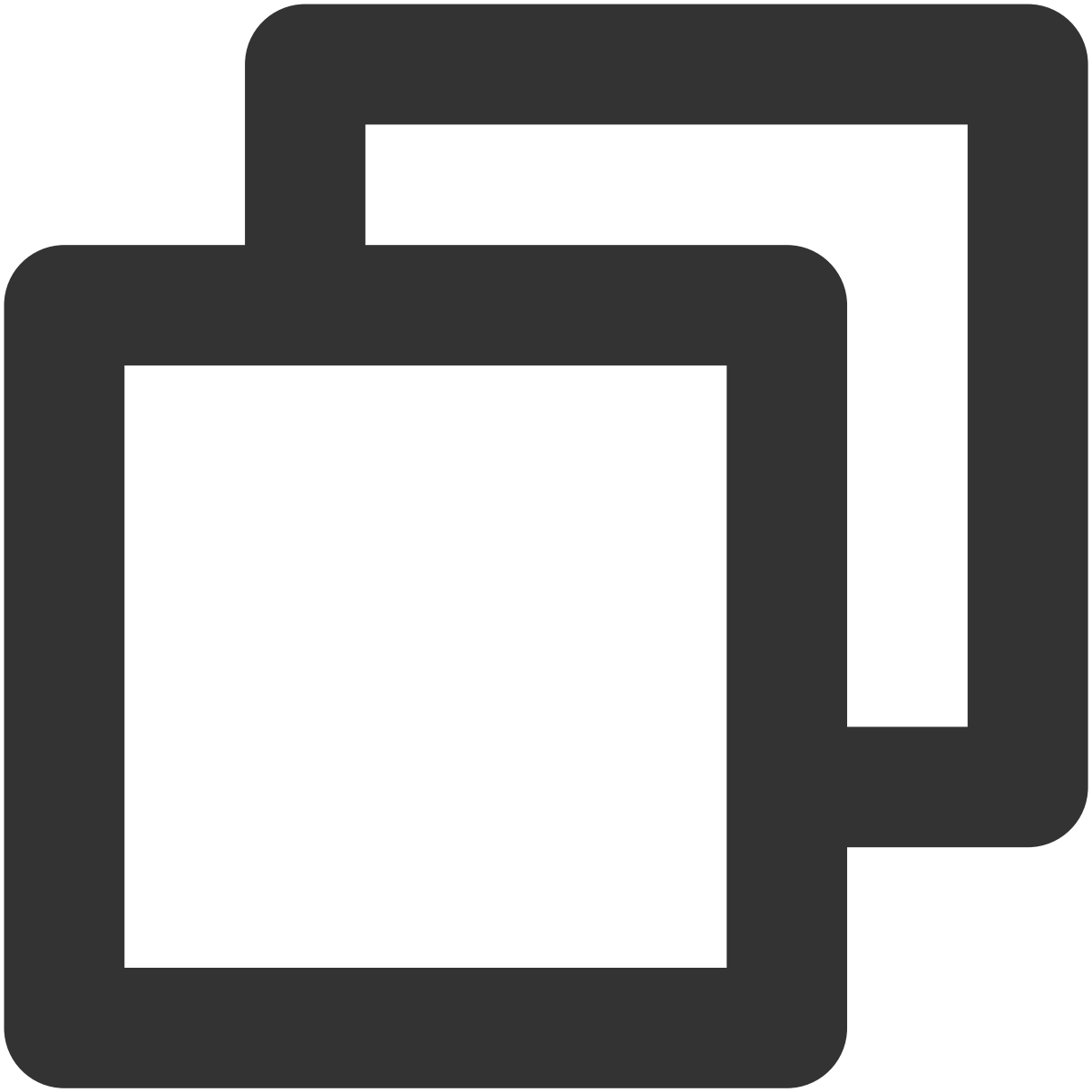
TUIRoomKit 是多人视频会议 SDK 提供的上层 UI 开源套件，目前 iOS 平台仅支持 Swift 语言，通过简单 API 调用即可唤起会议 UI。

TUIRoomKit 接口

API	描述
createInstance	初始化 TUIRoomKit 单例对象
destroyInstance	销毁 TUIRoomKit 单例对象
setSelfInfo	设置用户信息（头像、昵称）（可选）
createRoom	创建房间
enterRoom	进入房间

createInstance

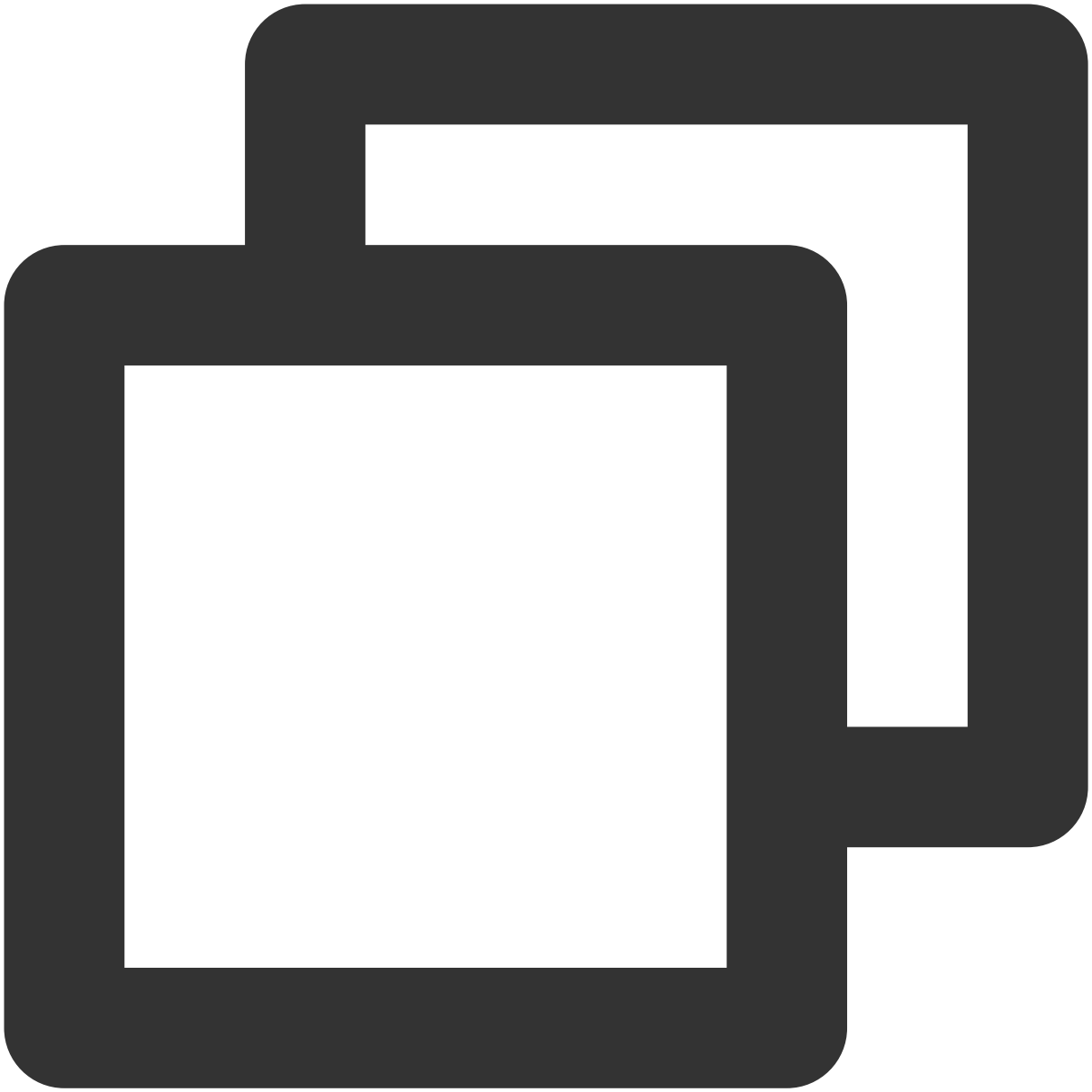
初始化 TUIRoomKit 单例对象。



```
public class func createInstance() -> TUIRoomKit
```

destroyInstance

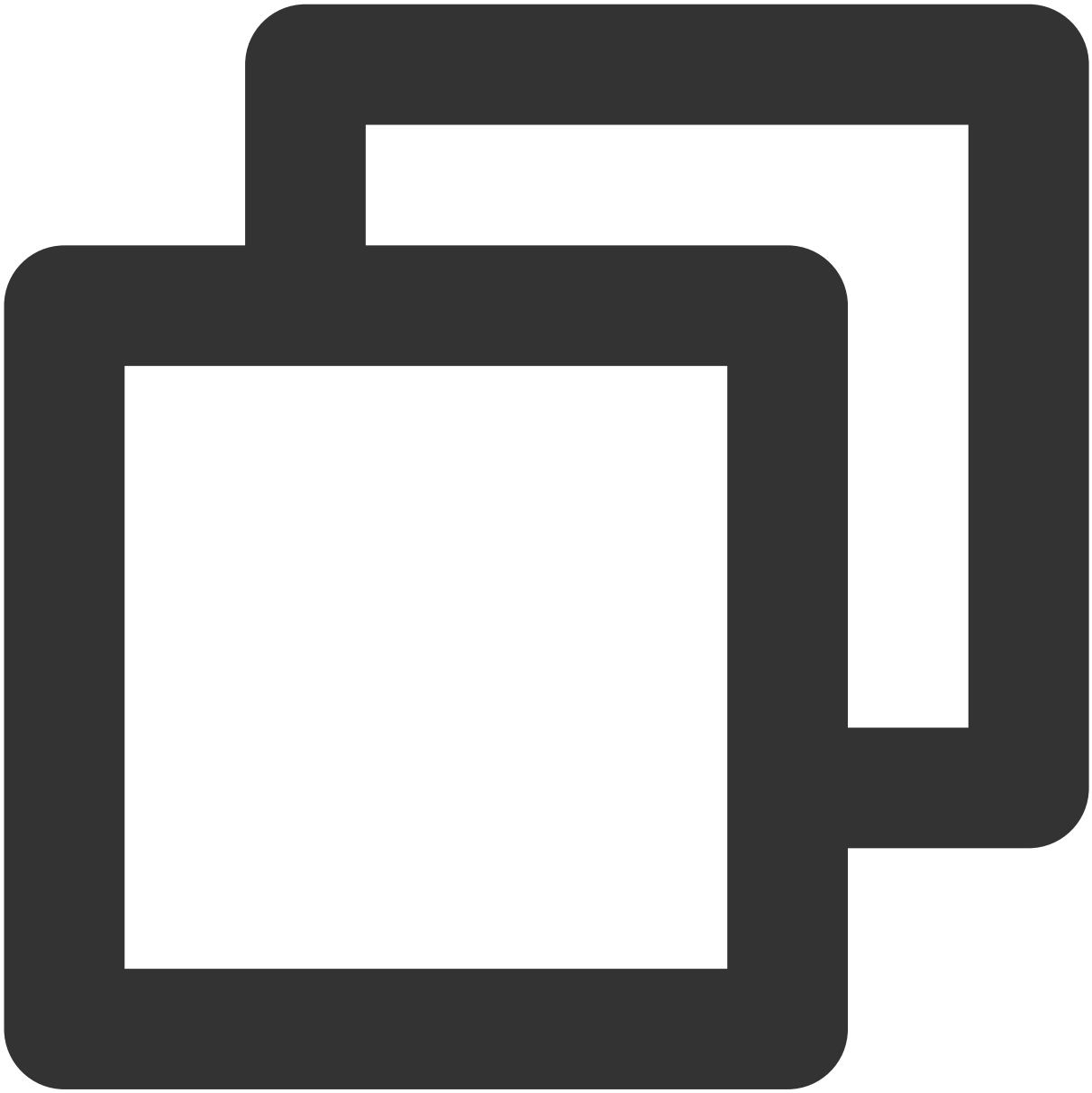
销毁 TUIRoomKit 单例对象。



```
public class func destroyInstance() -> Void
```

setSelfInfo（可选）

设置用户信息（头像、昵称）。



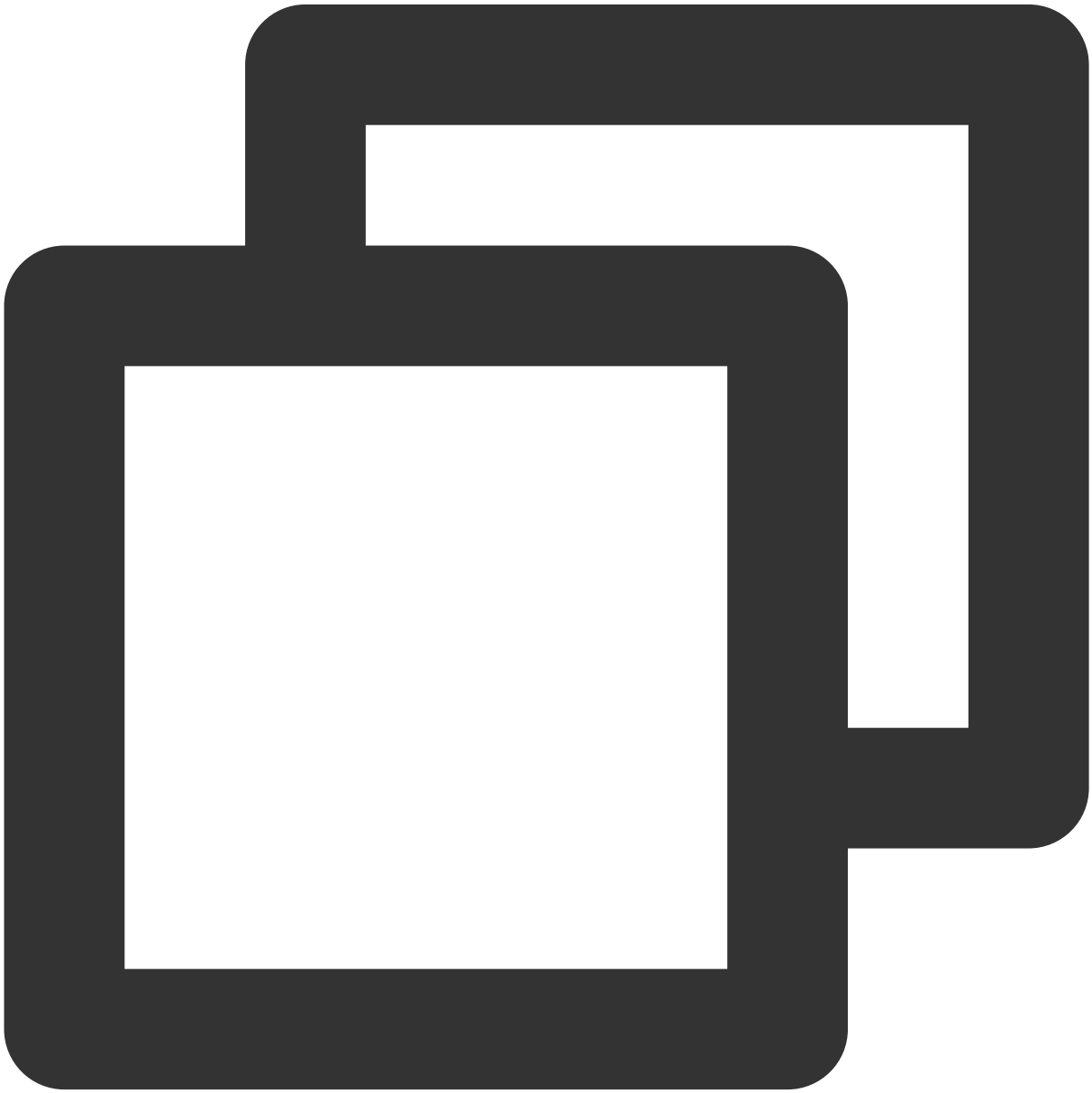
```
public func setSelfInfo(userName: String,
                        avatarURL: String,
                        onSuccess: @escaping TUISuccessBlock,
                          onError: @escaping TUIErrorBlock) -> Void
```

参数	类型	含义
userName	String	用户名
avatarURL	String	用户头像链接

onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

createRoom

创建房间。



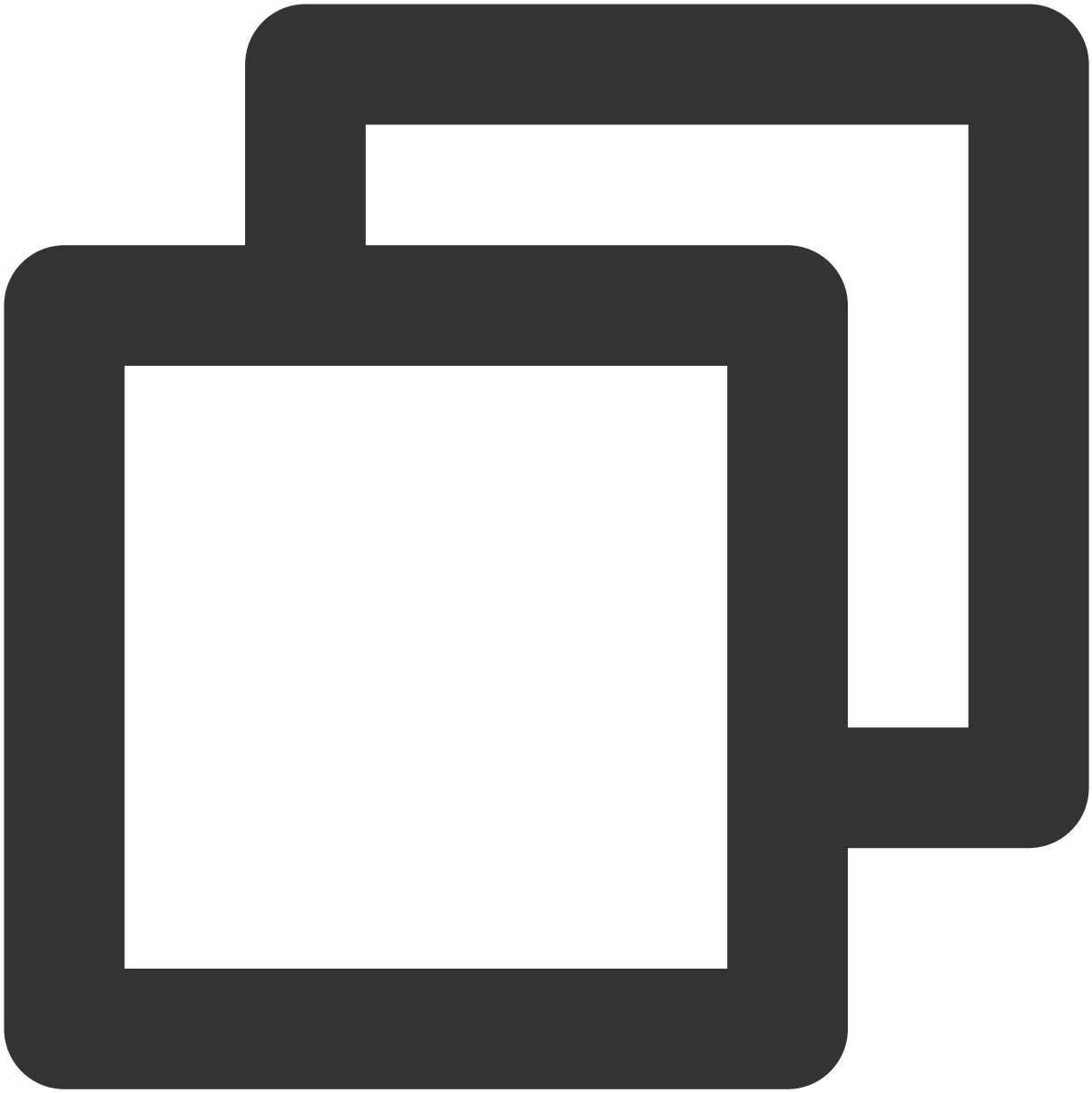
```
public func createRoom(roomInfo: TUIRoomInfo,
                       onSuccess: @escaping TUISuccessBlock,
                         onError: @escaping TUIErrorBlock) -> Void
```

参数含义如下表所示：

参数	类型	含义
roomInfo	TUIRoomInfo	房间基本信息
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

enterRoom

进入房间。



```
public func enterRoom(roomId: String,
                      enableAudio: Bool,
                      enableVideo: Bool,
                isSoundOnSpeaker: Bool,
                       onSuccess: @escaping TUISuccessBlock,
                         onError: @escaping TUIErrorBlock) -> Void
```

参数含义如下表所示：

参数	类型	含义

roomId	String	房间号字符串
enableAudio	Bool	进入房间是否开启音频
enableVideo	Bool	进入房间是否开启视频
isSoundOnSpeaker	Bool	进入房间是否开启扬声器
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEngine

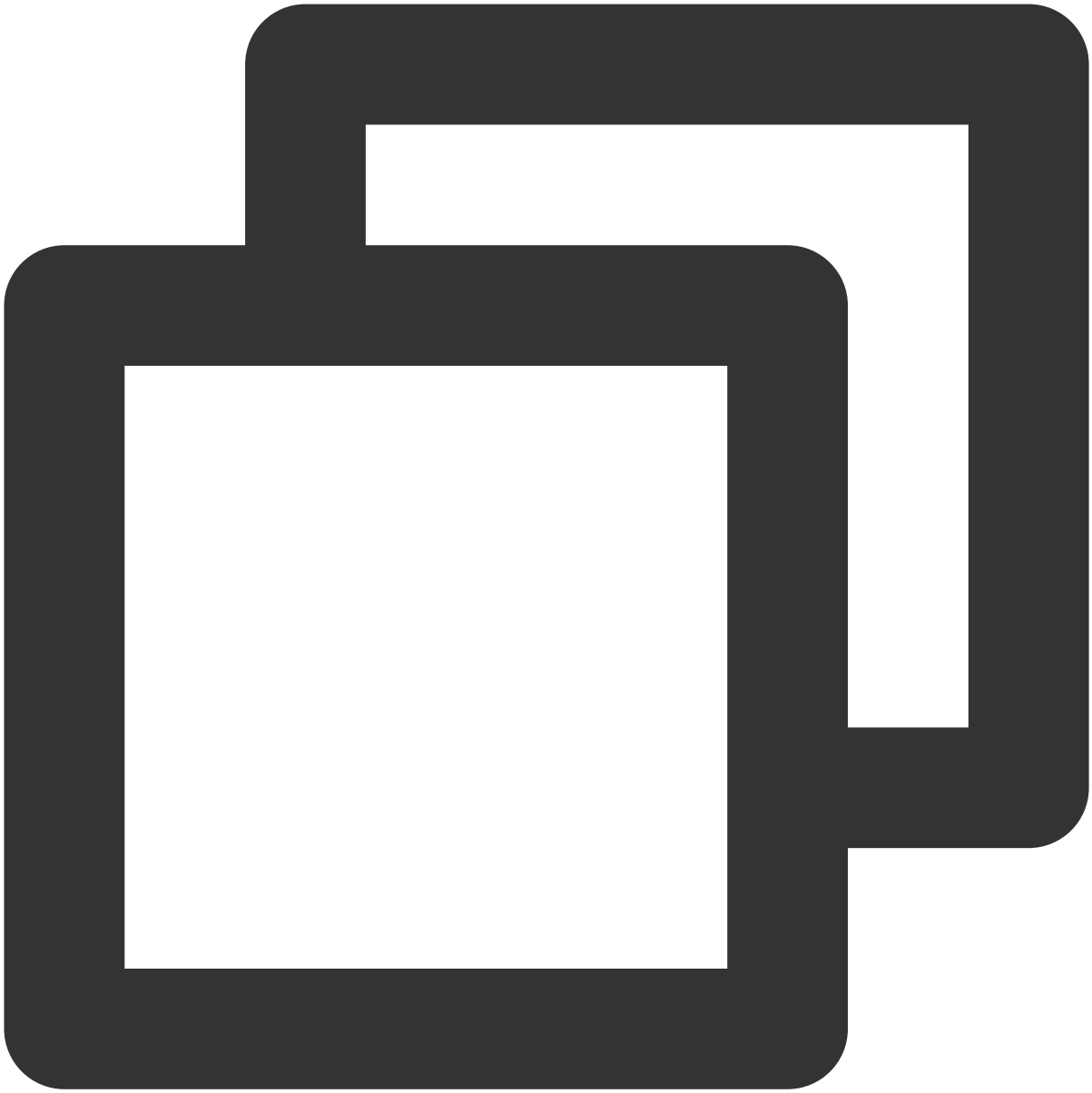
最近更新时间：2023-10-20 10:13:17

TUIRoomEngine API 简介

TUIRoomEngine API 是音视频通话组件的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。

init

创建 TUIRoomEngine 实例

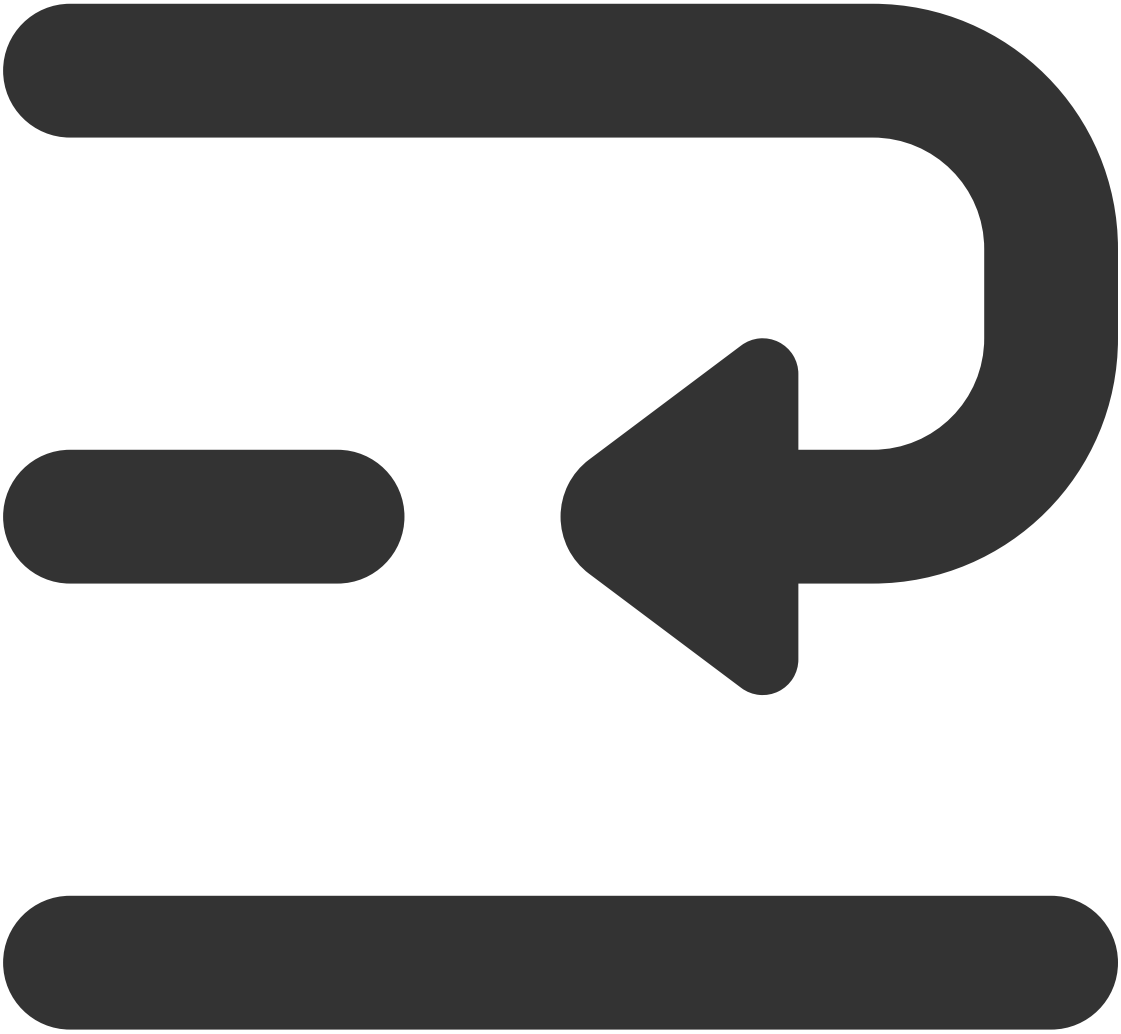


```
TUIRoomEngine *roomEngine = [[TUIRoomEngine alloc] init];
```

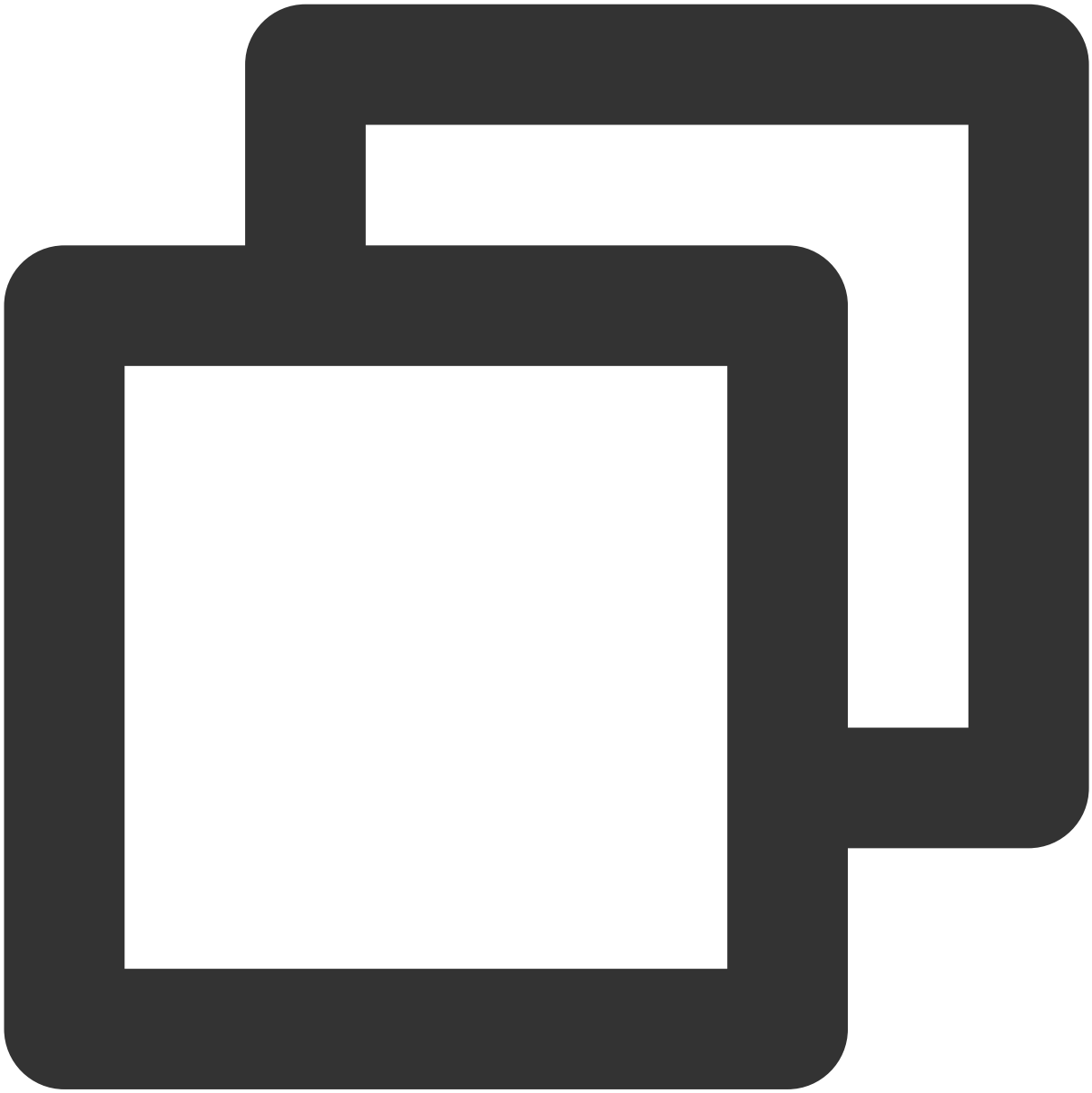
login

登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。

说明：v1.0.0 该接口命名为 setup，v1.0.1 及其以上版本请使用 TUIRoomEnine.login 登录 TUIRoomEngine。



版权所有：腾讯云计算（北京）有限责任公司



```
+ (void)loginWithSDKAppId:(NSInteger)sdkAppId
                   userId:(NSString *)userId
                  userSig:(NSString *)userSig
                onSuccess:(TUISuccessBlock)onSuccess
                  onError:(TUIErrorBlock)onError;
```

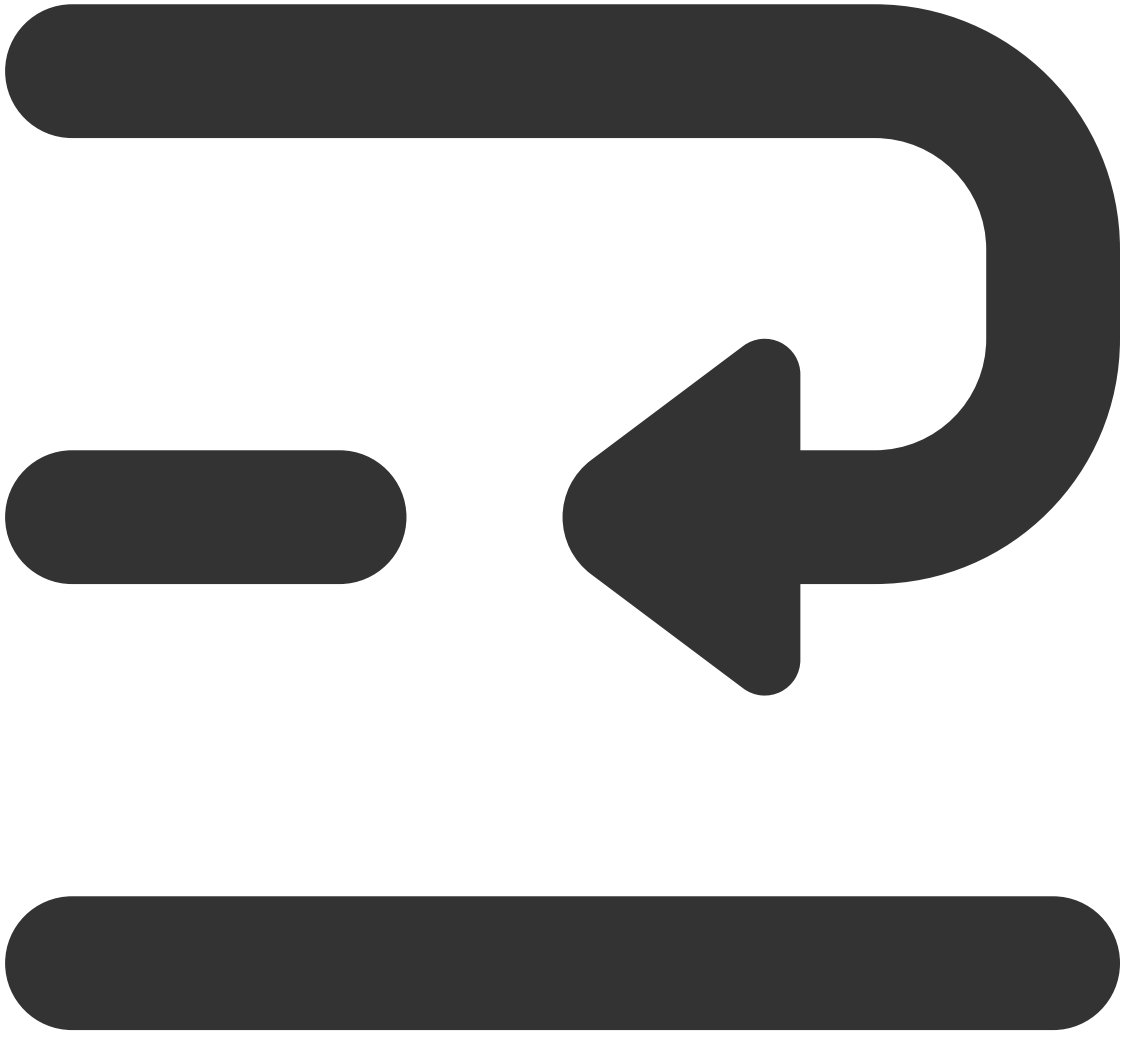
参数如下表所示：

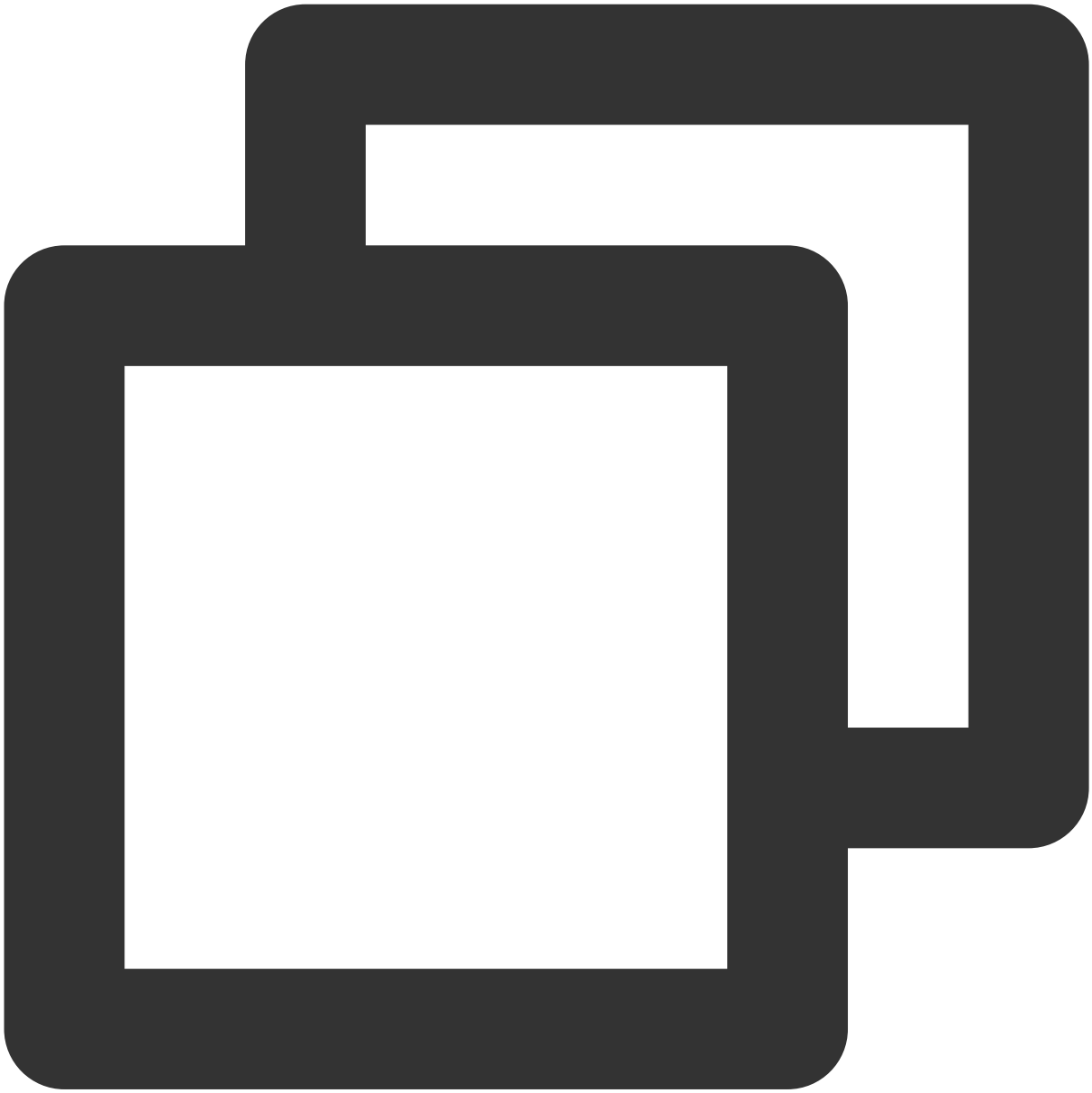
参数	类型	含义
sdkAppId	NSInteger	云通信应用的 SDKAppID

userId	NSString *	用户 ID，用于区分不同用户
userSig	NSString *	用户签名，用于腾讯云流量的认证
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

logout

退出登录接口，会有主动离开房间操作、销毁资源





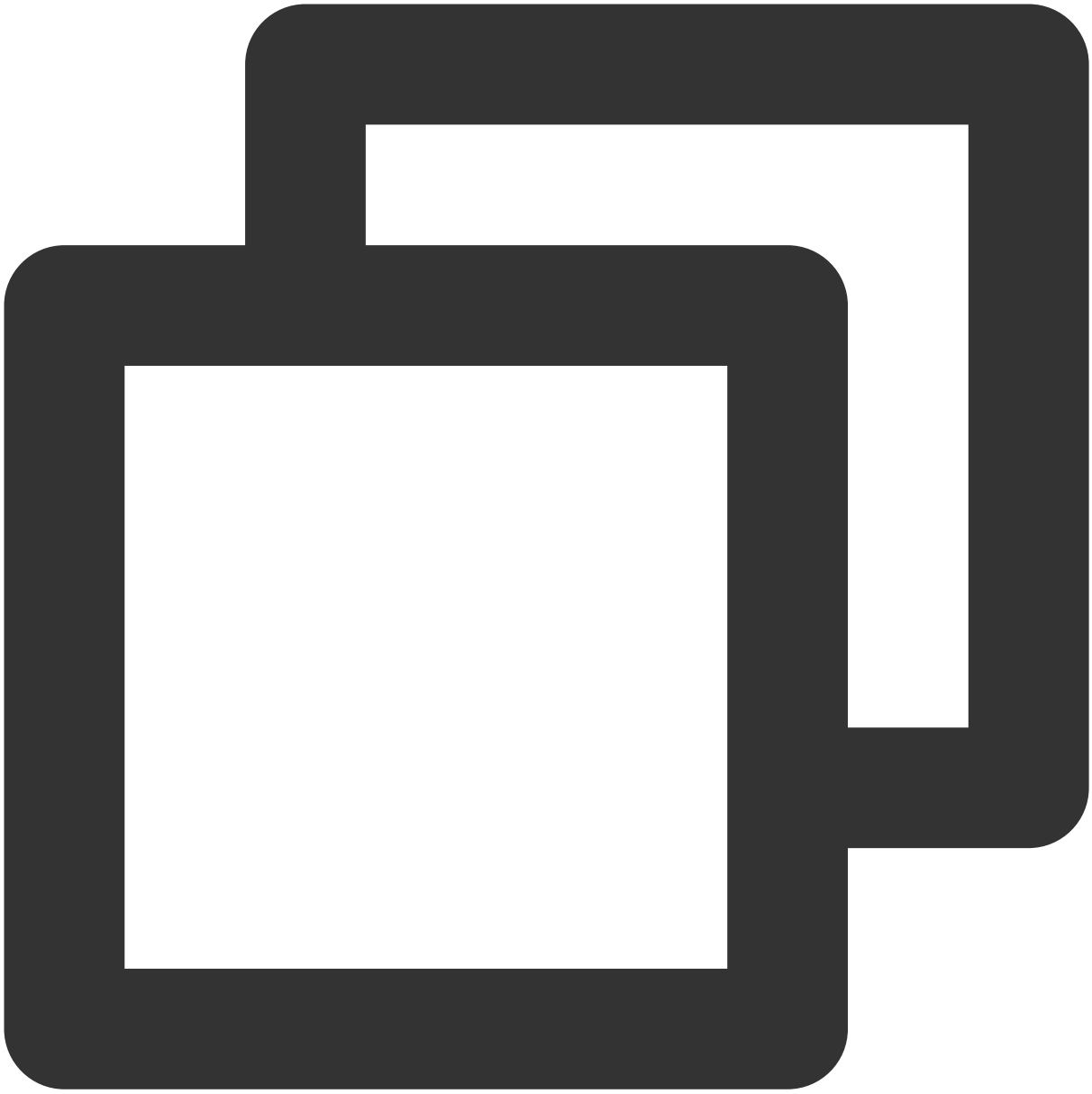
```
+ (void)logout:(TUISuccessBlock)onSuccess
       onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

setSelfInfo

设置本地用户名称和头像。



```
+ (void)setSelfInfoWithUserName:(NSString *)userName
                      avatarUrl:(NSString *)avatarUrl
                      onSuccess:(TUISuccessBlock)onSuccess
                        onError:(TUIErrorBlock)onError;
```

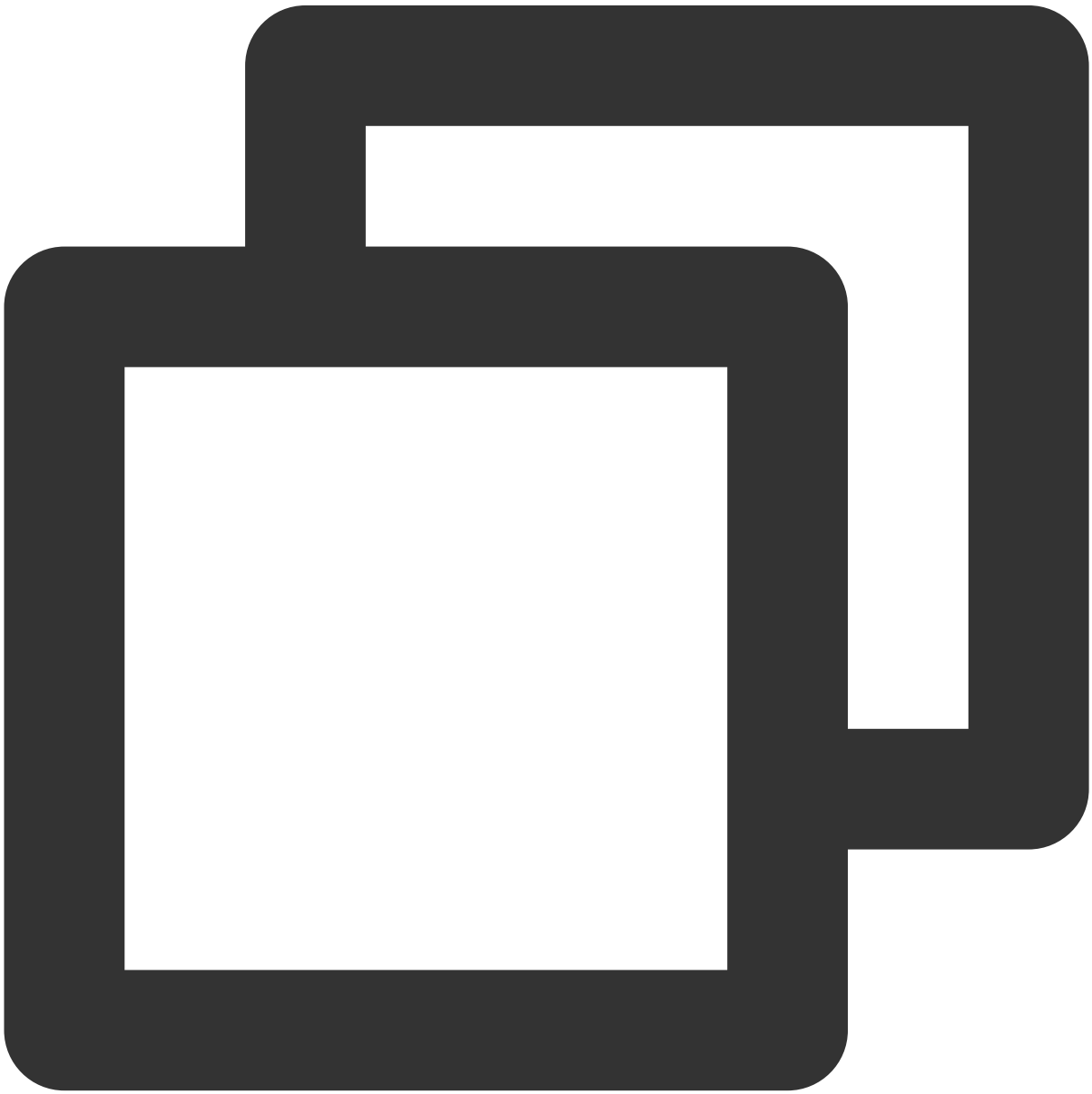
参数如下表所示：

参数	类型	含义

userName	NSString *	用户名称
avatarUrl	NSString *	用户头像 URL 地址
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

getSelfInfo

获取本地用户登录的基本信息。



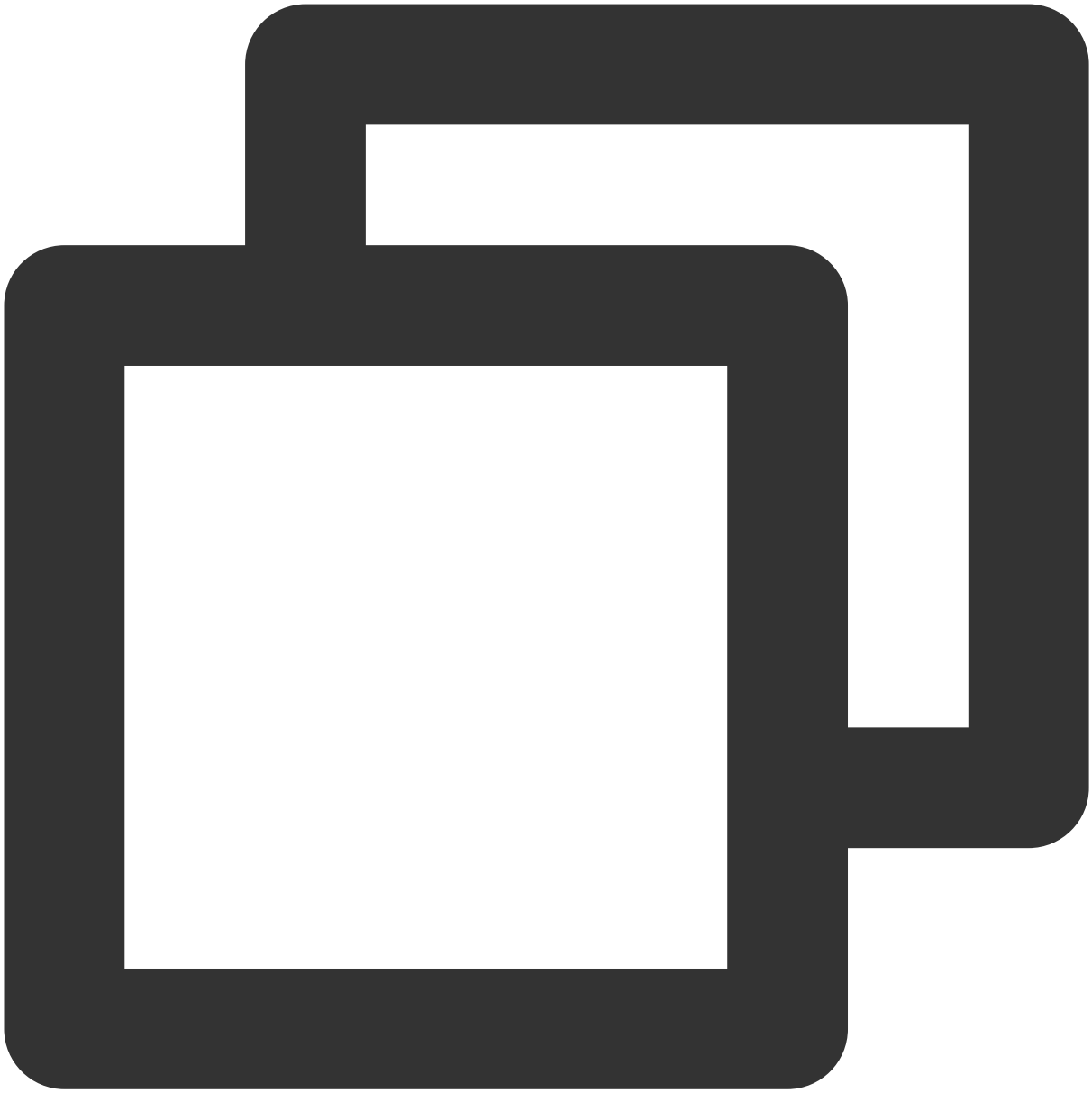
```
+ (TUILoginUserInfo *)getSelfInfo;
```

参数如下表所示：

返回值	类型	含义
selfInfo	TUILoginUserInfo *	本地用户登录的基本信息，详细定义可以参考 TUIRoomDefine.h 中的TUILoginUserInfo。

addObserver

设置事件回调。



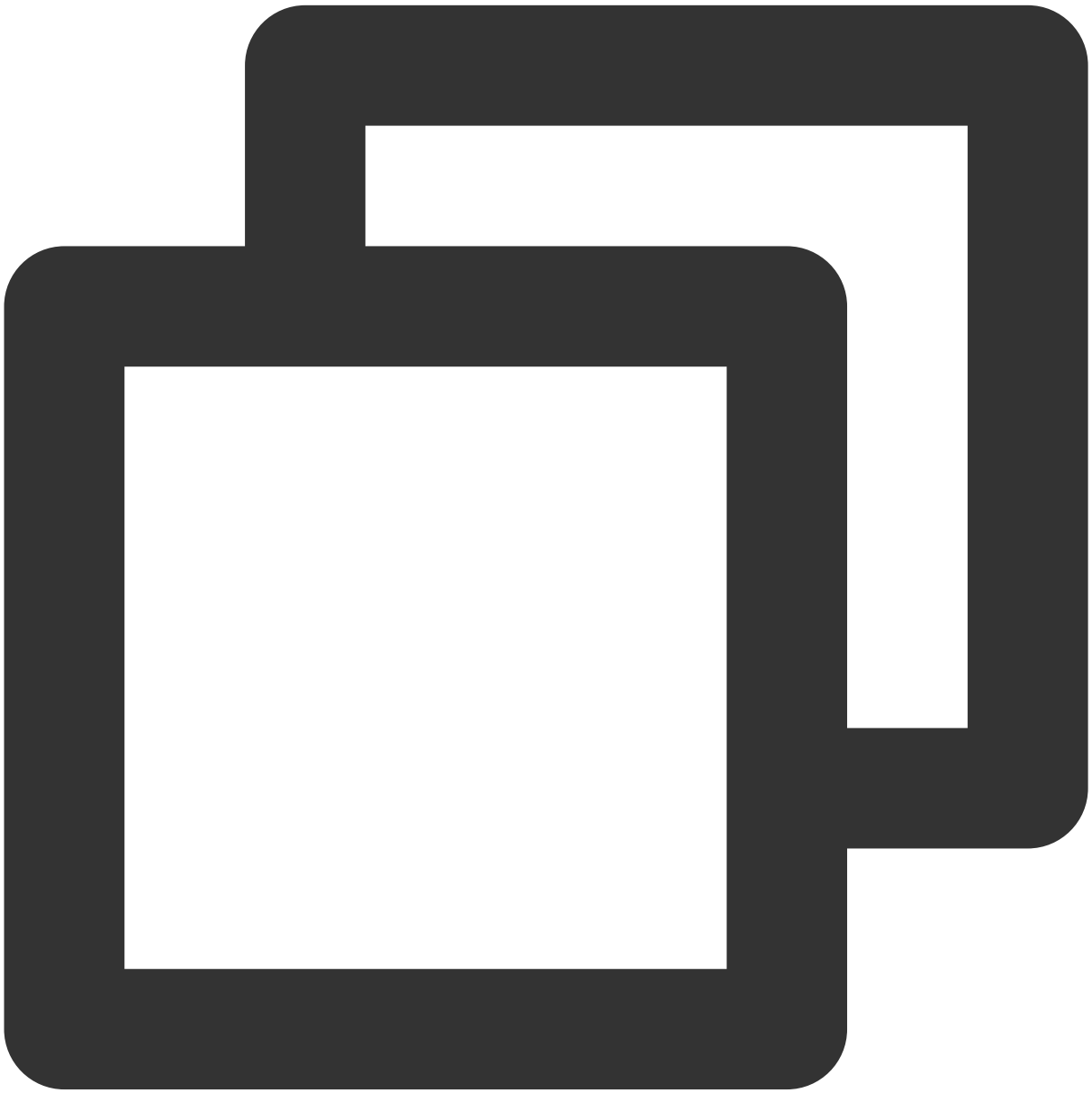
```
- (void)addObserver:(id<TUIRoomObserver>)observer;
```

参数如下表所示：

参数	类型	含义
observer	TUIRoomObserver *	回调实例的指针，您可以通过 TUIRoomObserver 获得各类事件通知（比如：错误码，远端用户进房，音视频状态参数等）

removeObserver

移除事件回调。



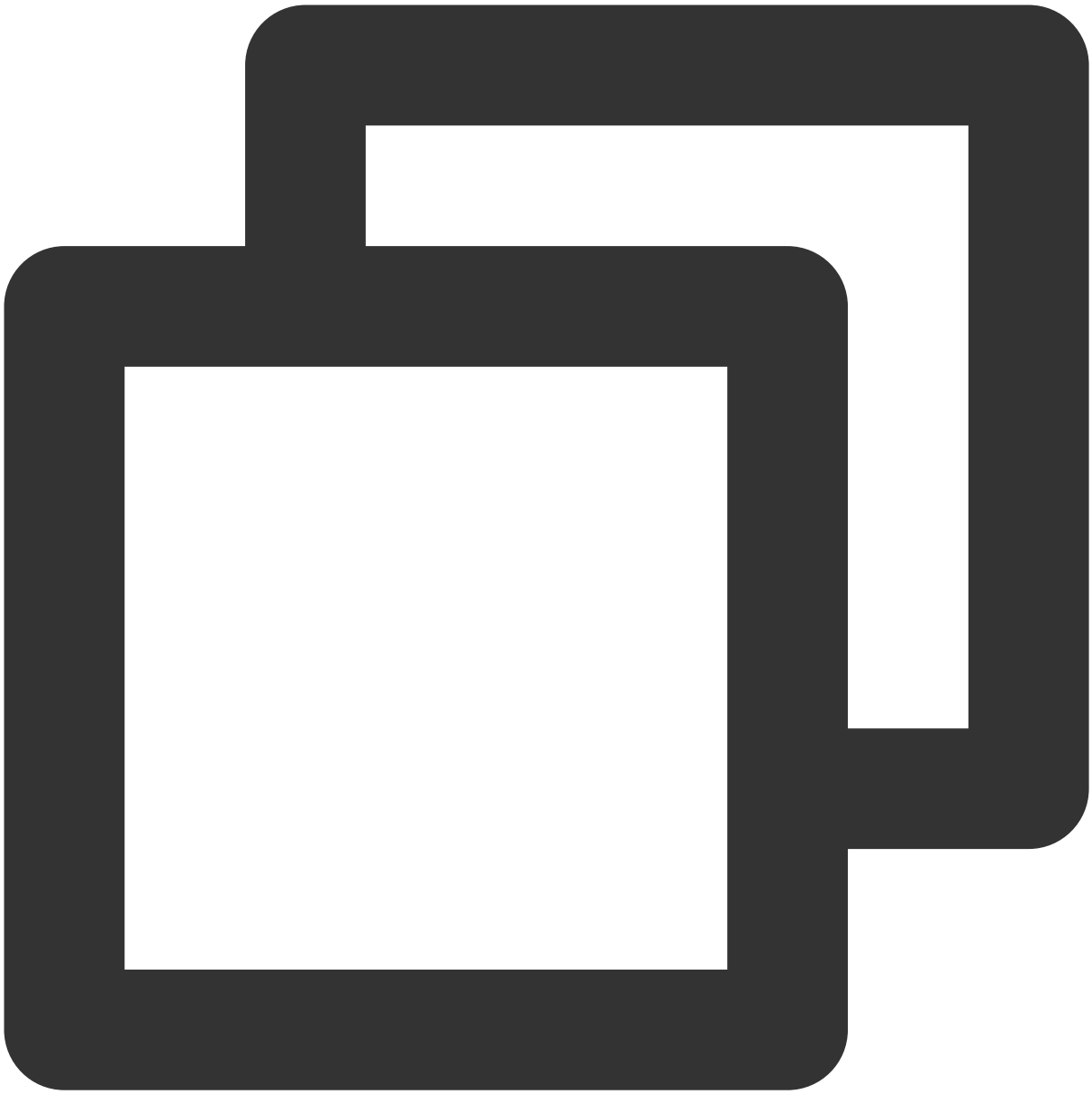
```
- (void)removeObserver:(id<TUIRoomObserver>)observer;
```

参数如下表所示：

参数	类型	含义
observer	TUIRoomObserver *	回调实例的指针

createRoom

创建房间。



```
- (void)createRoom:(TUIRoomInfo *)roomInfo
         onSuccess:(TUISuccessBlock)onSuccess
           onError:(TUIErrorBlock)onError;
```

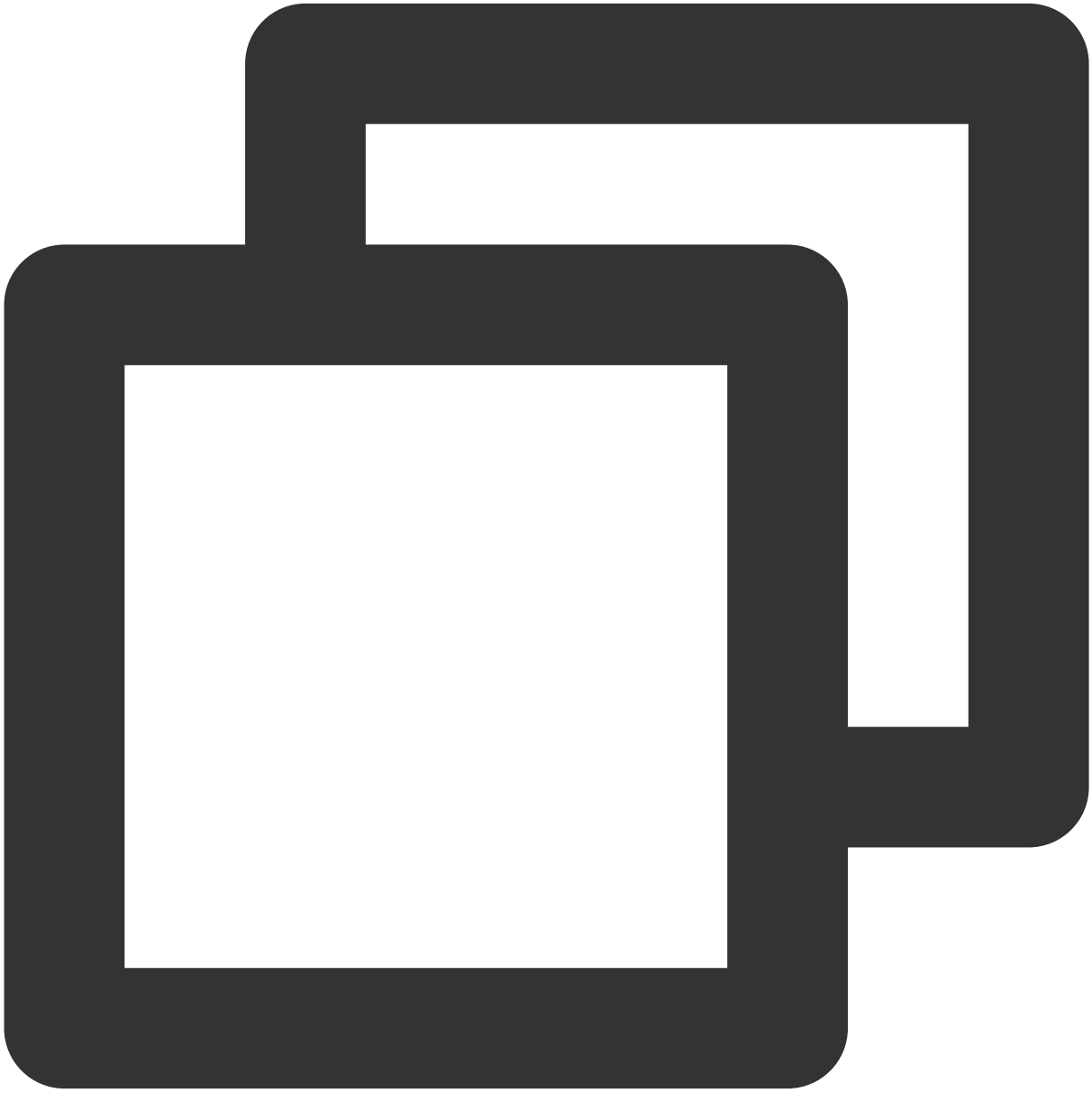
参数如下表所示：

参数	类型	含义
roomInfo	TUIRoomInfo *	房间信息，可以初始化房间的一些设置

onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

destroyRoom

解散房间。



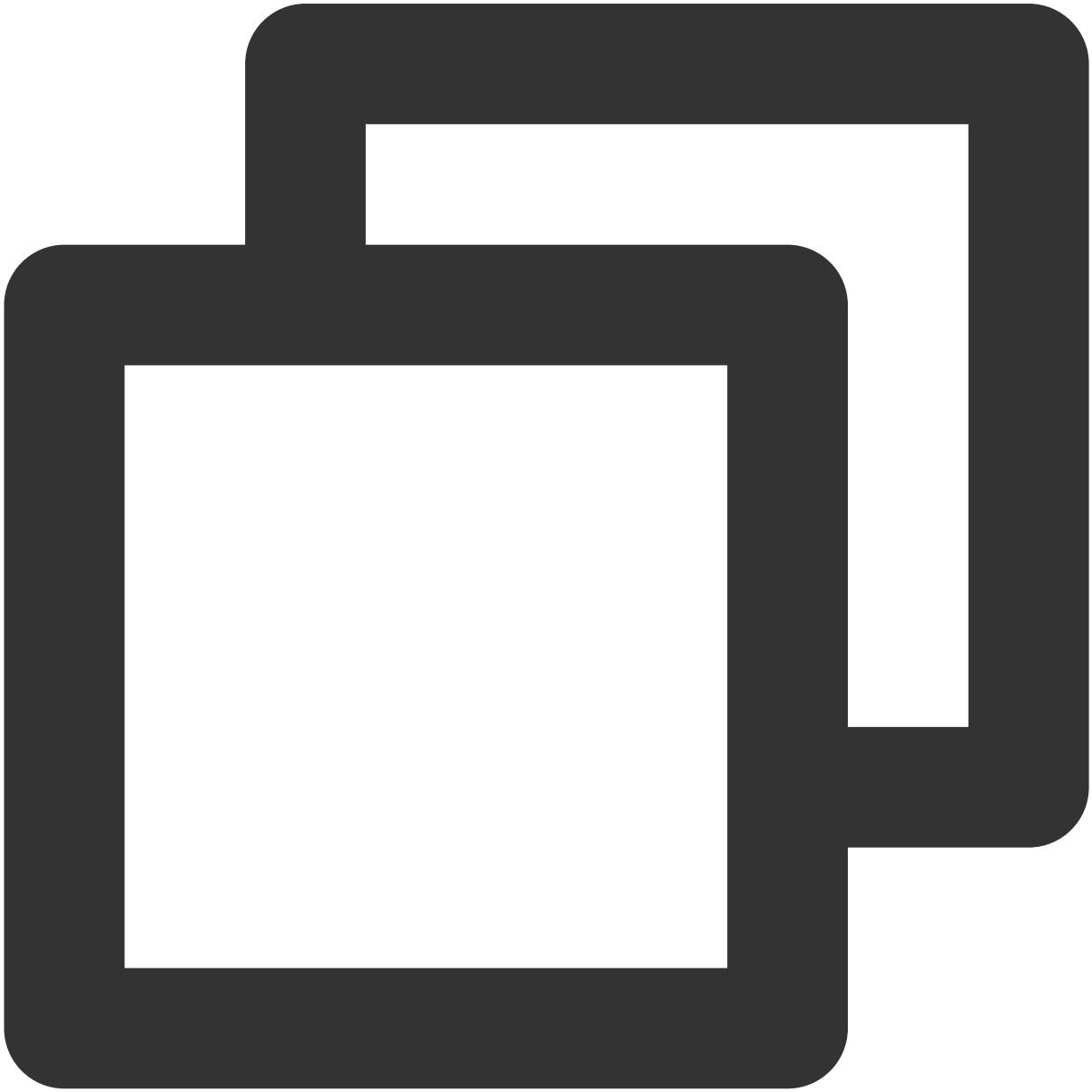
```
- (void)destroyRoom:(TUISuccessBlock)onSuccess onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

enterRoom

进入房间。



```
- (void)enterRoom:(NSString *)roomId
```

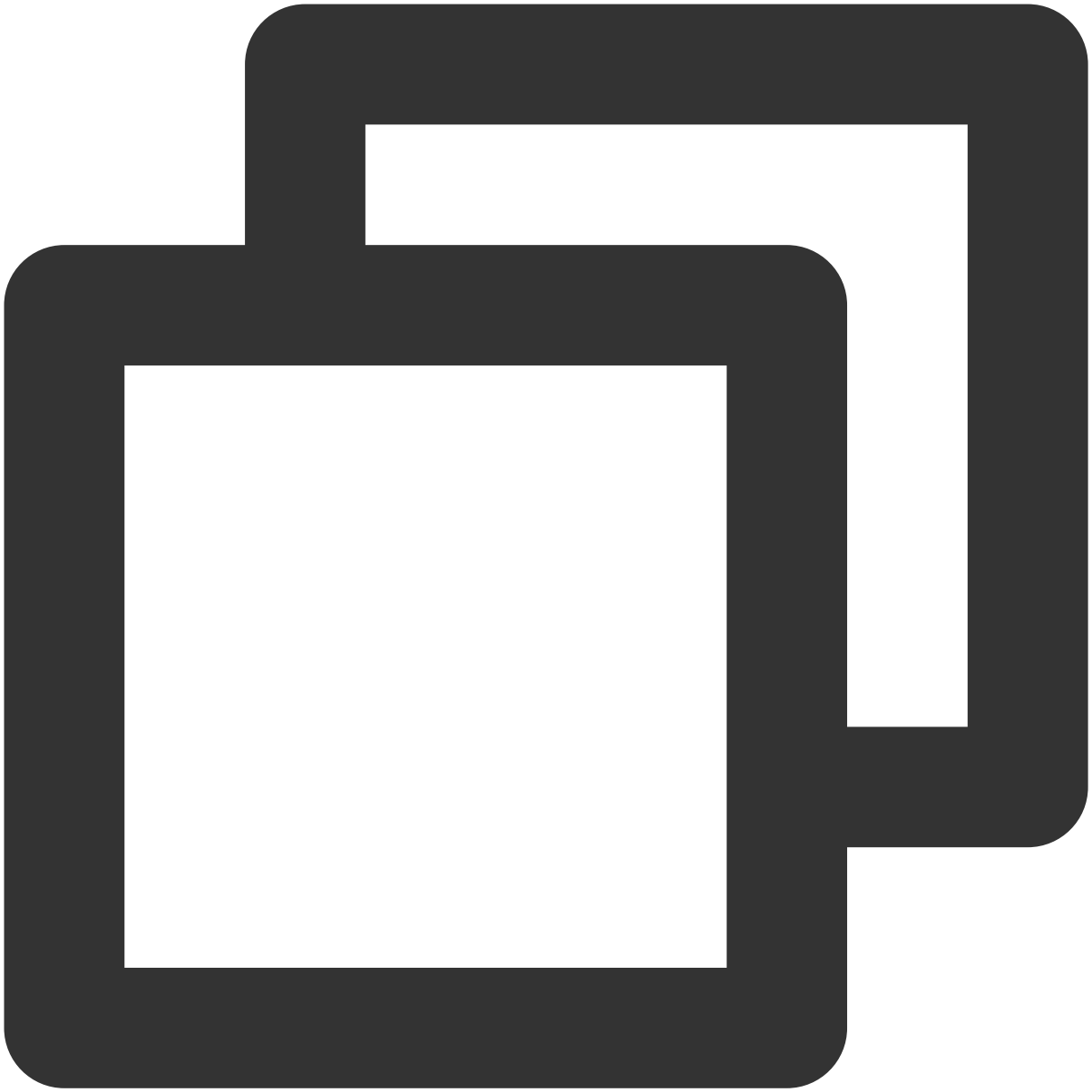
```
        onSuccess:(TUIRoomInfoBlock)onSuccess
          onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID
onSuccess	TUIRoomInfoBlock	成功回调
onError	TUIErrorBlock	失败回调

exitRoom

离开房间。



```
- (void)exitRoom:(BOOL)syncWaiting
       onSuccess:(TUISuccessBlock)onSuccess
         onError:(TUIErrorBlock)onError;
```

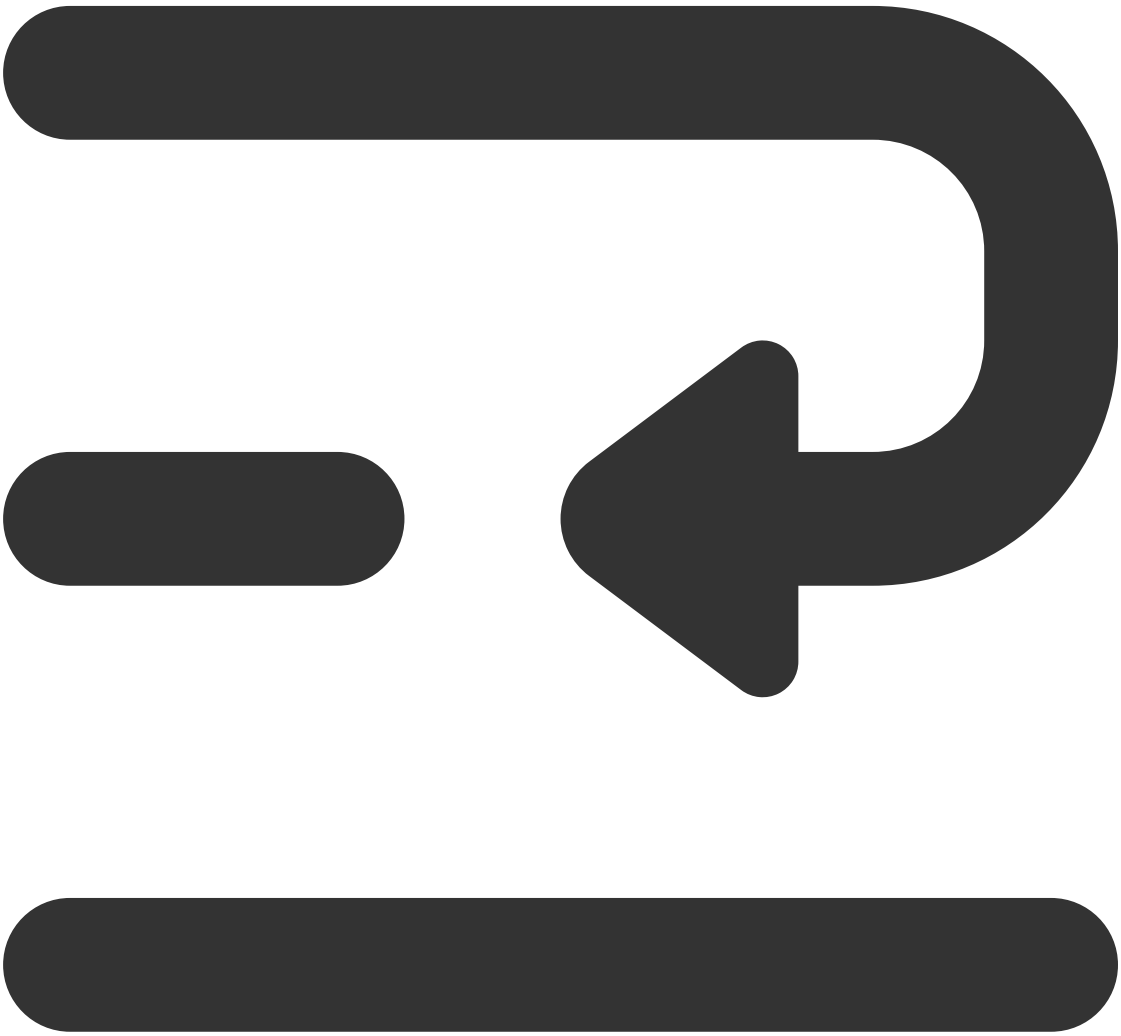
参数如下表所示：

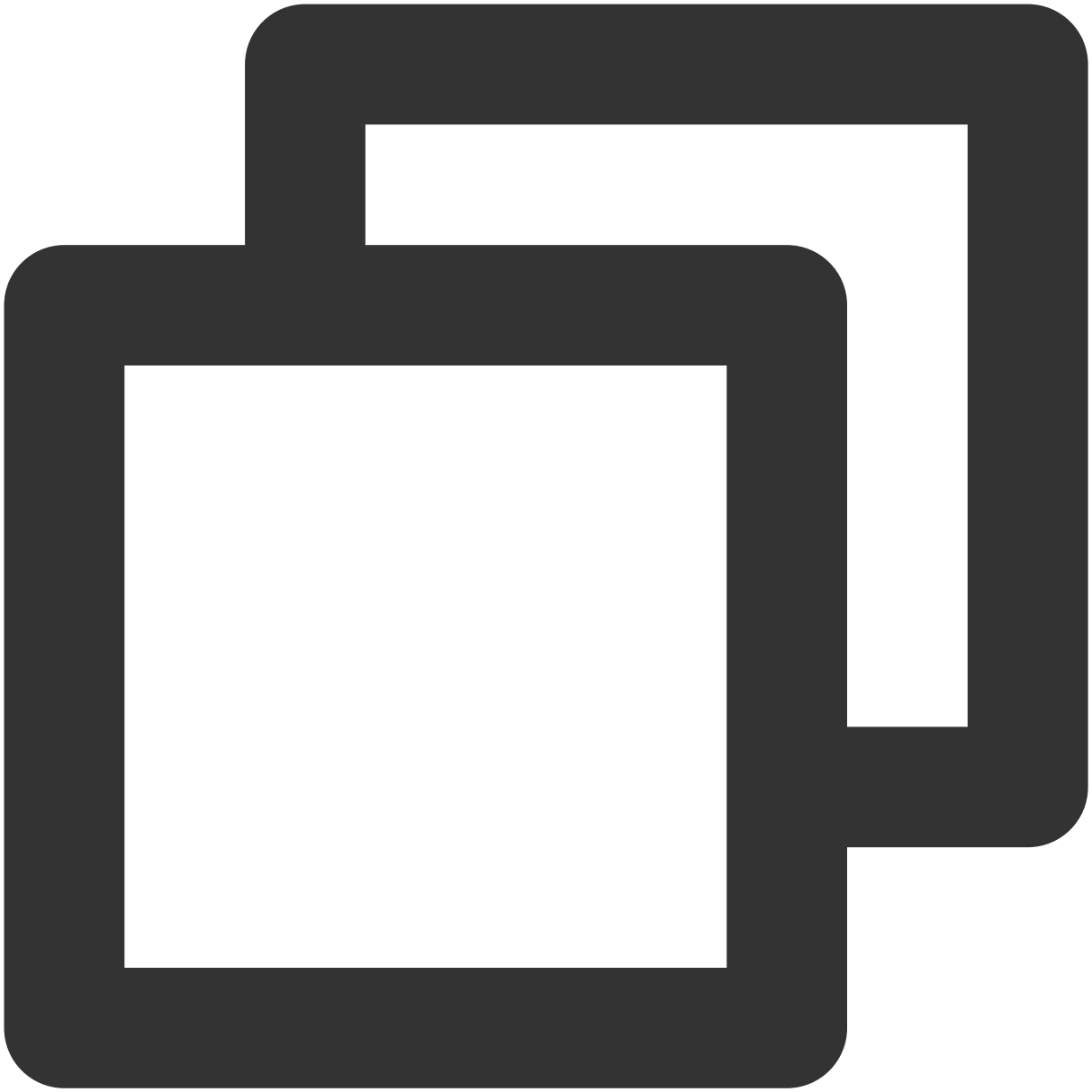
参数	类型	含义
syncWaiting	BOOL	是否同步等待接口返回
onSuccess	TUISuccessBlock	成功回调

onError	TUIErrorBlock	失败回调

connectOtherRoom

连接其他房间（用于直播场景下的申请跨房连麦）。





```
- (TUIRequest *)connectOtherRoom:(NSString *)roomId
                          userId:(NSString *)userId
                         timeout:(NSTimeInterval)timeout
                      onAccepted:(TUIRequestAcceptedBlock)onAccepted
                      onRejected:(TUIRequestRejectedBlock)onRejected
                     onCancelled:(TUIRequestCancelledBlock)onCancelled
                       onTimeout:(TUIRequestTimeoutBlock)onTimeout
                         onError:(TUIRequestErrorBlock)onError;
```

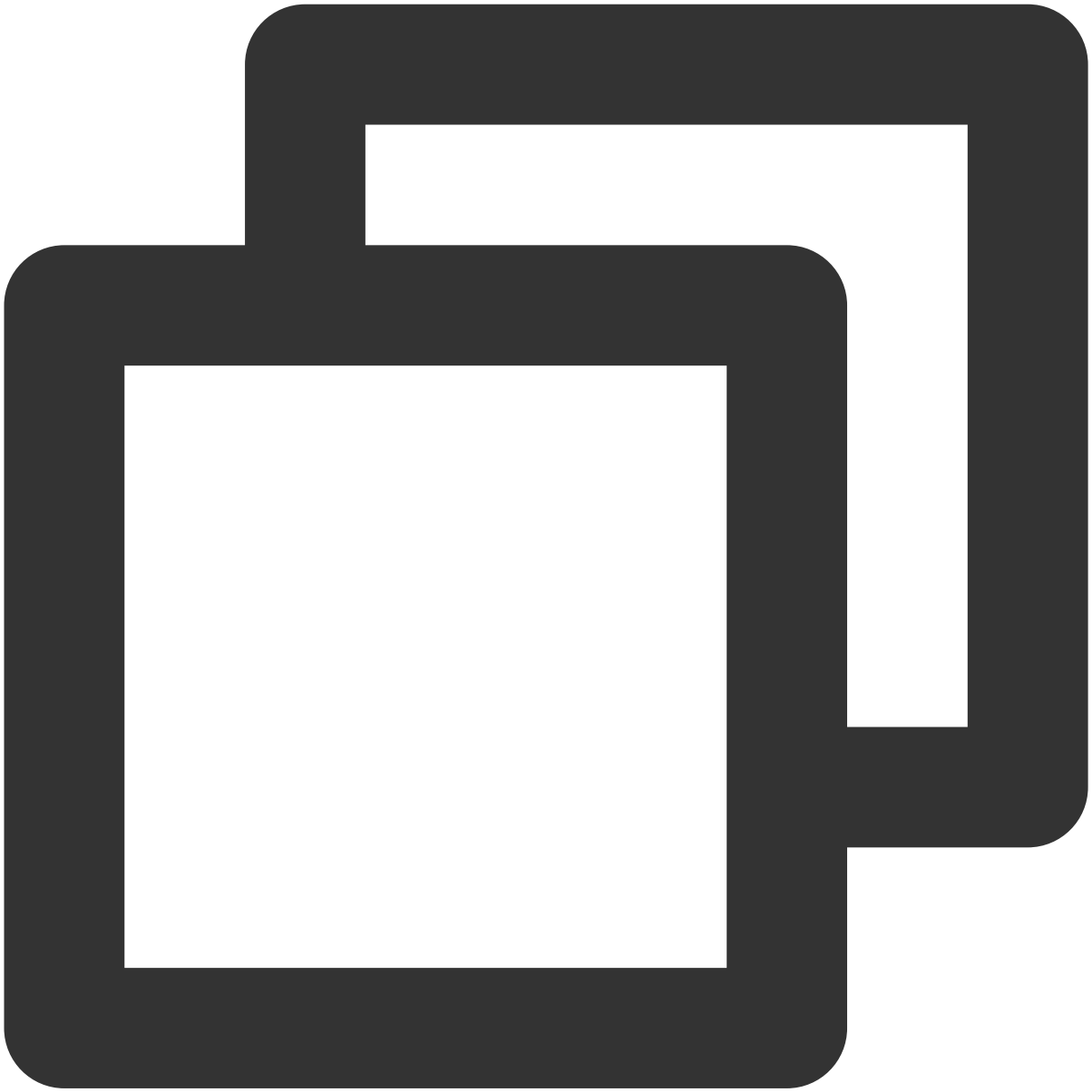
参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID
userId	NSString *	用户 ID
timeout	NSTimeInterval	超时时间（单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调）
onAccepted	TUIRequestAcceptedBlock	邀请被接受的回调
onRejected	TUIRequestRejectedBlock	邀请被拒绝的回调
onCancelled	TUIRequestCancelledBlock	邀请被取消的回调
onTimeout	TUIRequestTimeoutBlock	邀请超时未处理的回调
onError	TUIRequestErrorBlock	邀请发生错误的回调

返回值	类型	含义
request	TUIRequest	请求体

disconnectOtherRoom

断开与其他房间的连接（用于直播场景下的断开跨房连麦）。



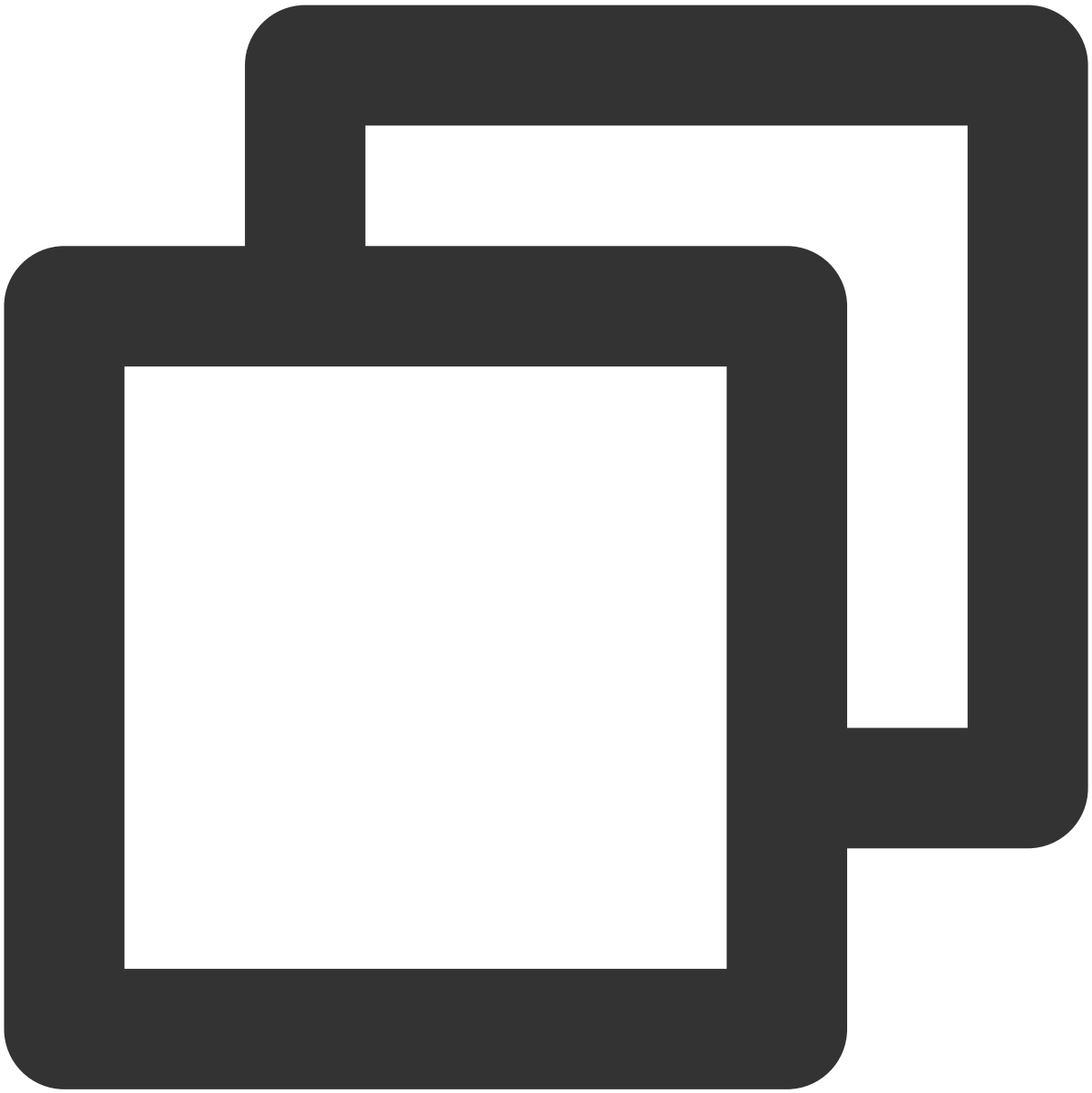
```
- (void)disconnectOtherRoom:(TUISuccessBlock)onSuccess onError:(TUIErrorBlock)onErr
```

参数如下表所示：

参数	类型	含义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

fetchRoomInfo

获取房间信息。



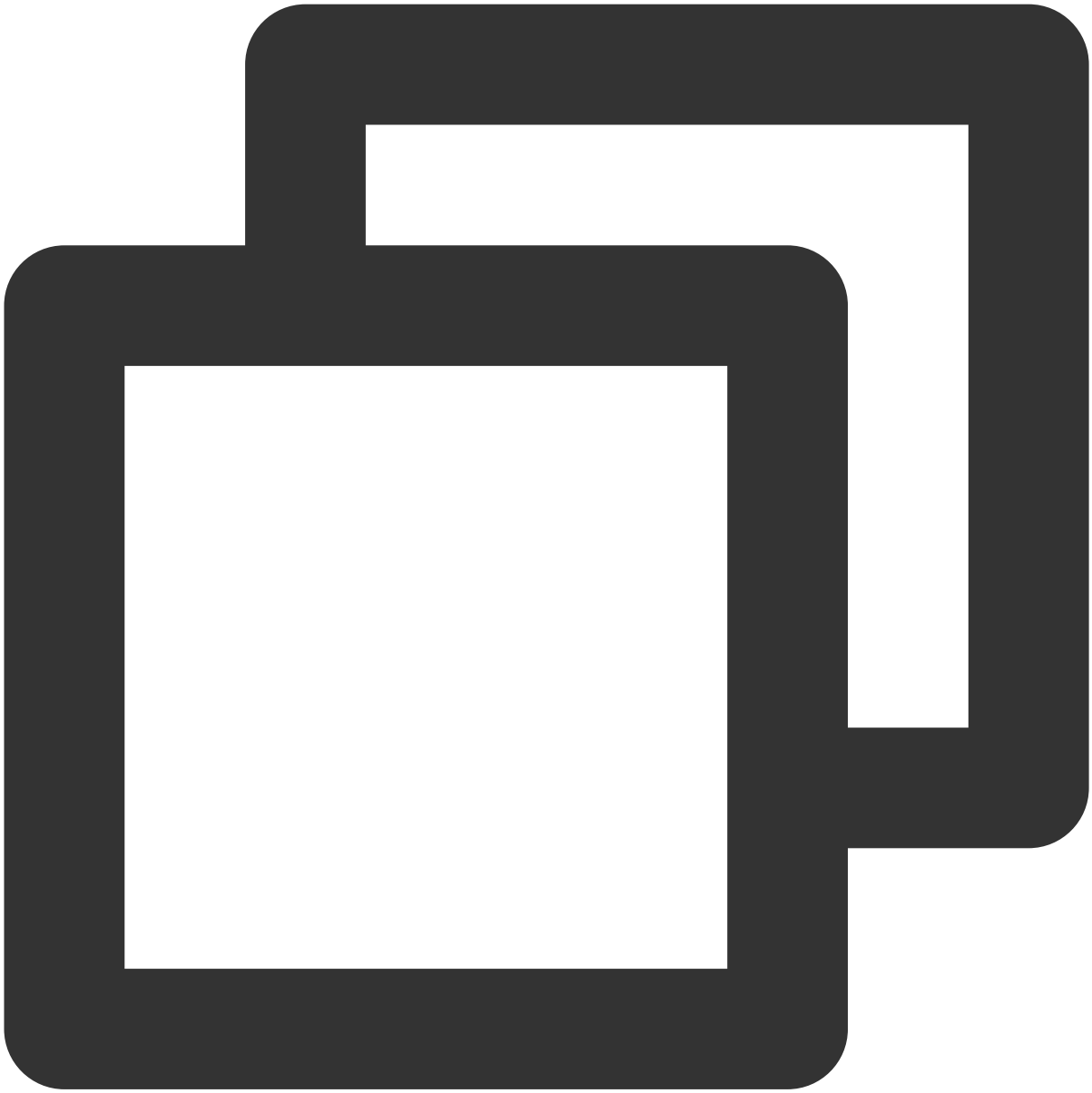
```
- (void)fetchRoomInfo:(TUIRoomInfoBlock)onSuccess onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
onSuccess	TUIRoomInfoBlock	成功获取到房间信息回调，回调会包含 TUIRoomInfo 房间信息
onError	TUIErrorBlock	失败回调

updateRoomNameByAdmin

更新房间名称（只有管理员或群主能够调用）。



```
- (void)updateRoomNameByAdmin:(NSString *)roomName
                    onSuccess:(TUISuccessBlock)onSuccess
                      onError:(TUIErrorBlock)onError;
```

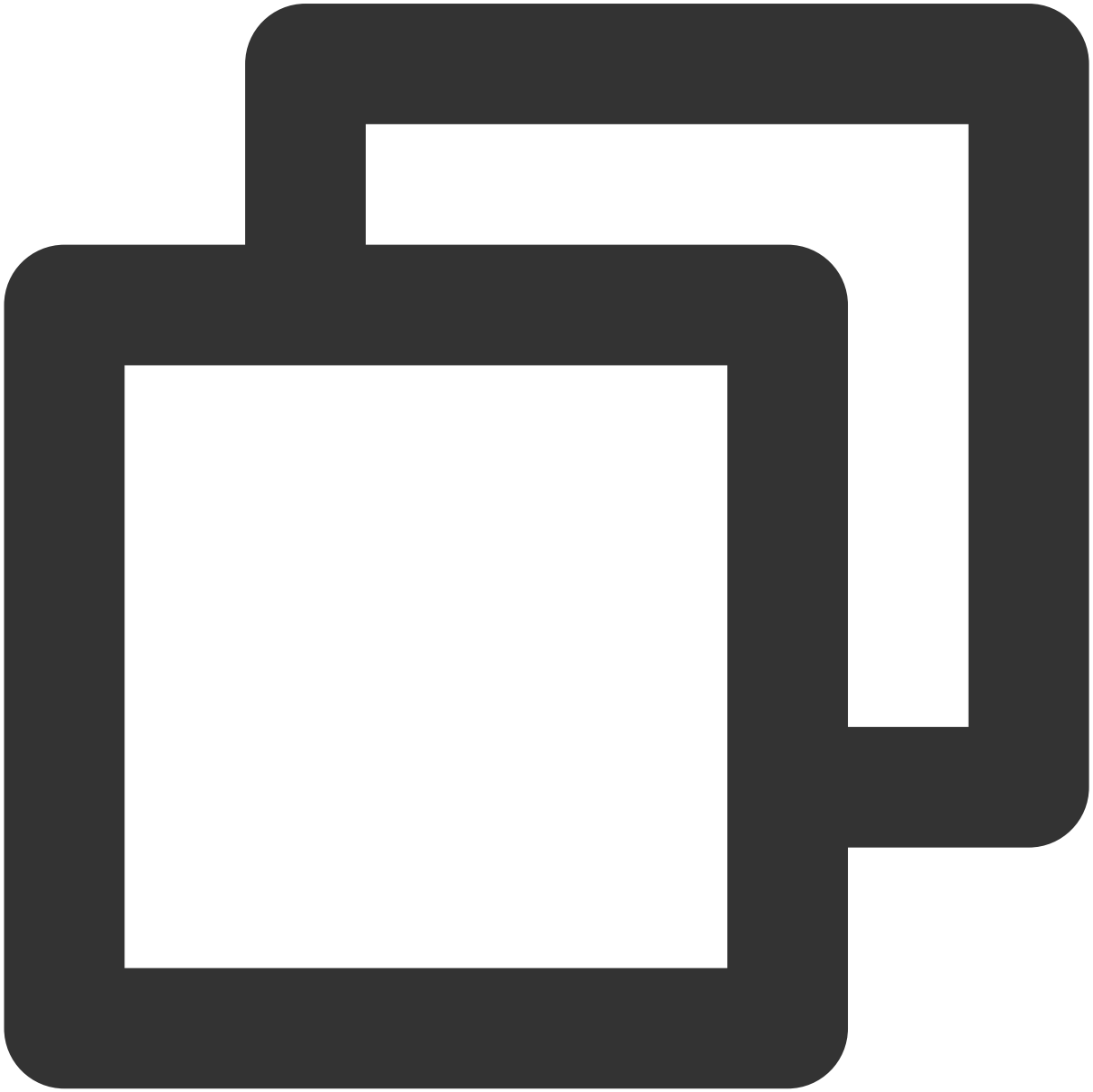
参数如下表所示：

参数	类型	含义

roomName	NSString *	房间名称
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

updateRoomSpeechModeByAdmin

设置房间麦控模式（只有管理员或群主能够调用）。



```
- (void)updateRoomSpeechModeByAdmin:(TUISpeechMode)mode
```

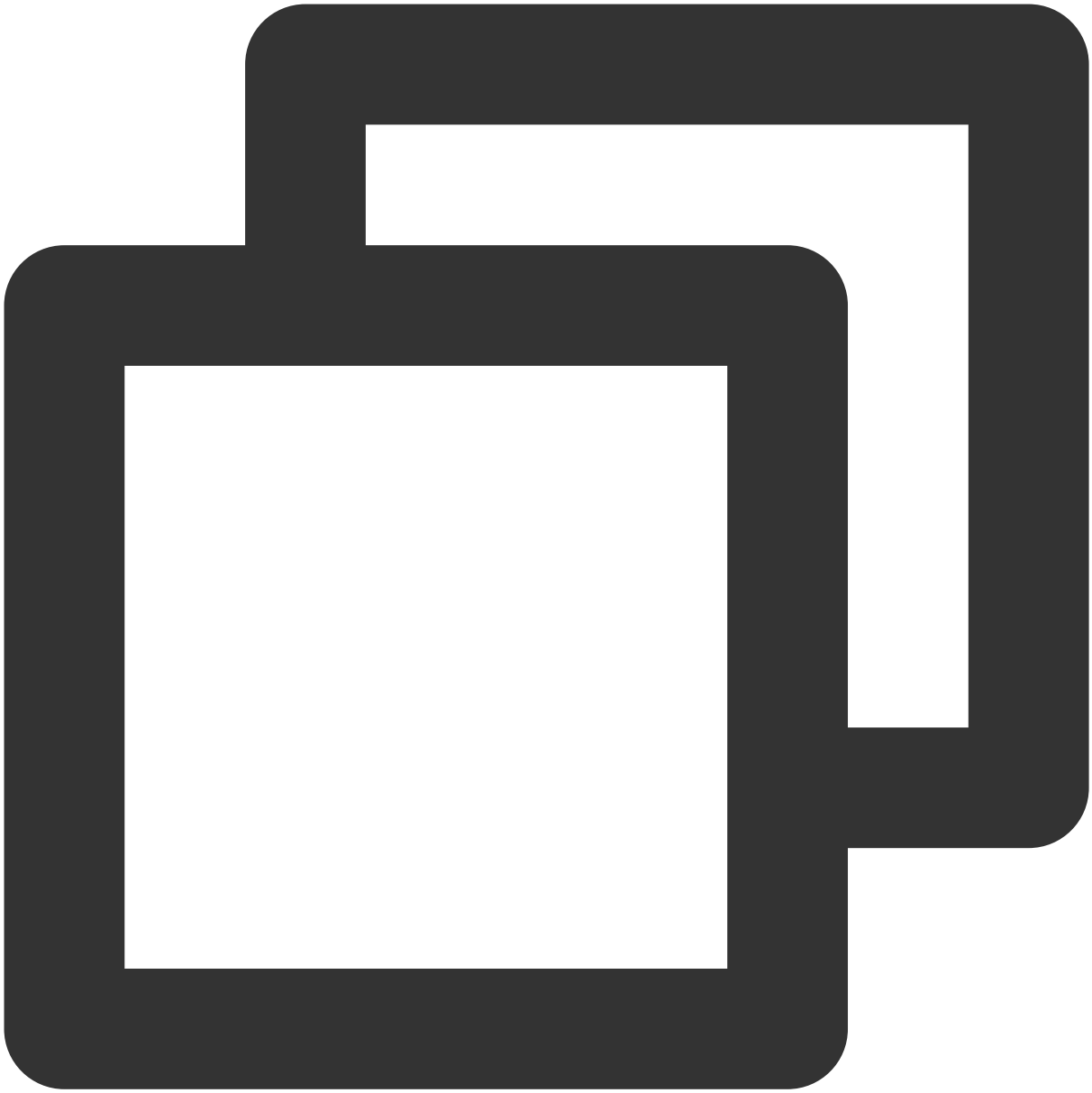
```
                                onSuccess:(TUISuccessBlock)onSuccess
                                  onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
mode	TUISpeechMode	麦控模式,详细定义可以参考 TUIRoomDefine.h 文件的 TUISpeechMode。
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

setLocalVideoView

设置本地用户视频渲染的控件。



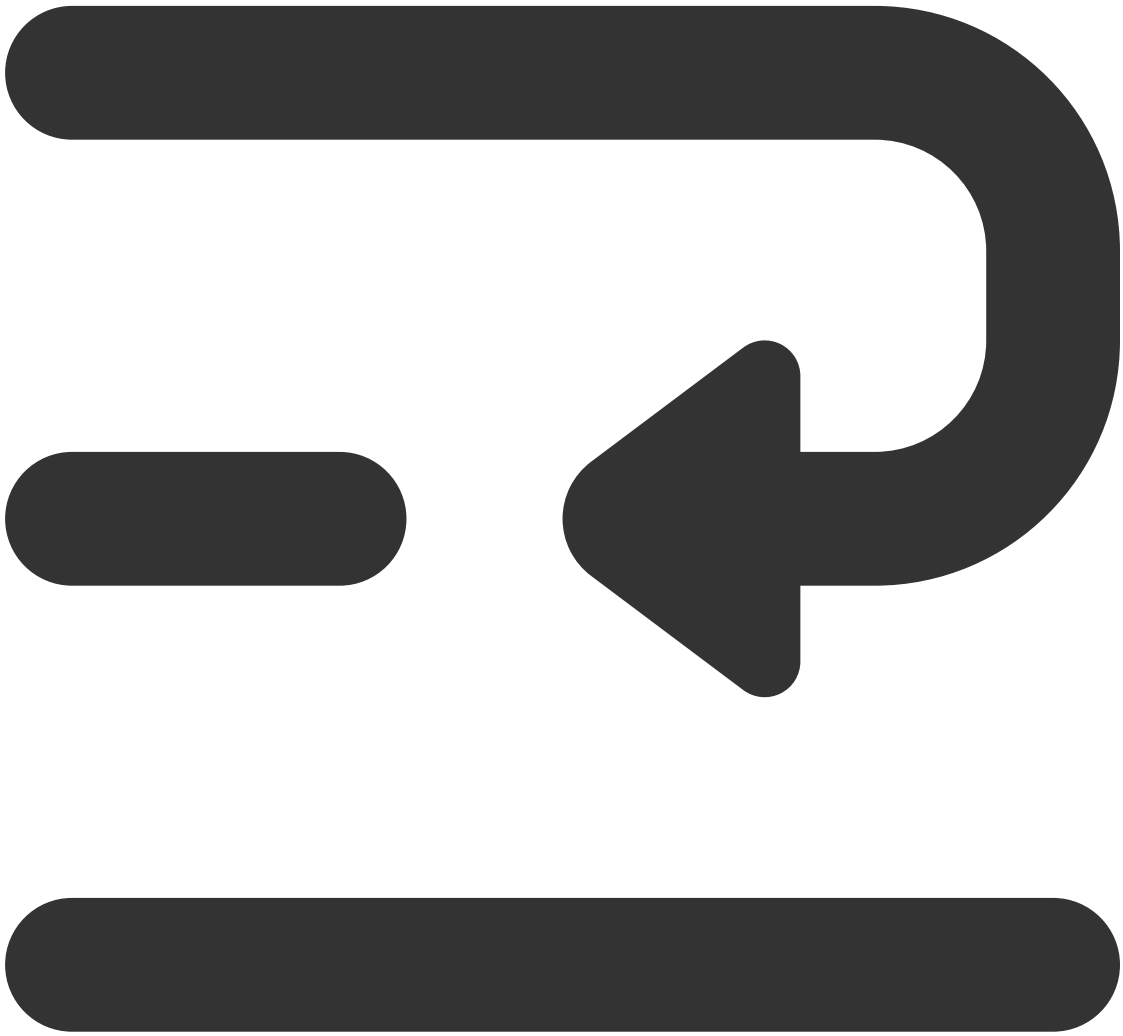
```
- (void)setLocalVideoView:(TUIVideoStreamType)streamType view:(TUIVideoView *)view;
```

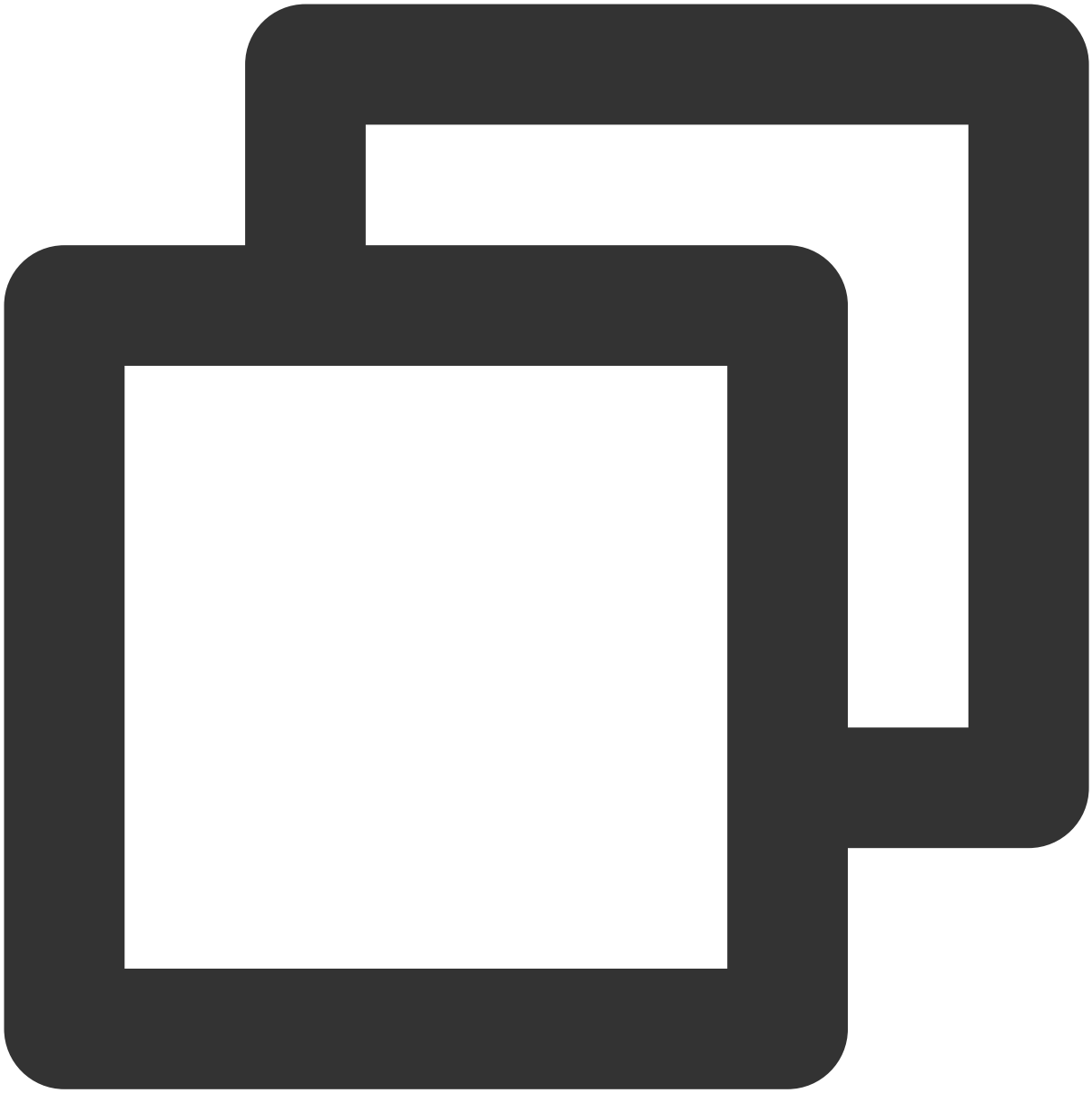
参数如下表所示：

参数	类型	含义
streamType	TUIVideoStreamType	视频流类型，详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义
view	TUIVideoView *	视频渲染视图

setRemoteVideoView

设置远端用户视频渲染的控件。





```
- (void)setRemoteVideoView:(NSString *)userId
                streamType:(TUIVideoStreamType)streamType
                      view:(TUIVideoView * __nullable)view;
```

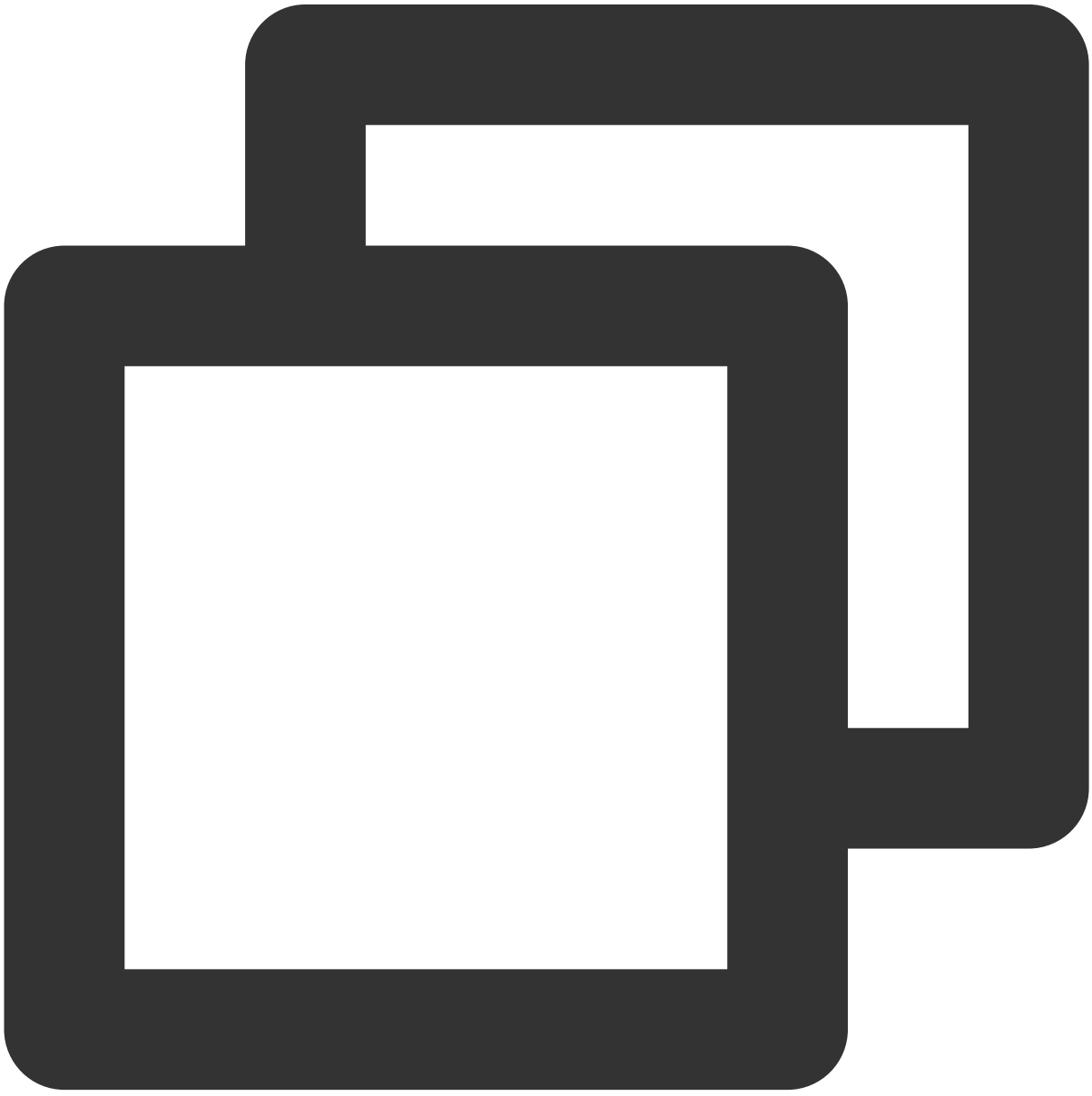
参数如下表所示：

参数	类型	含义
userId	NSString	远端用户ID
streamType	TUIVideoStreamType	视频流类型，详细定义可以参考 TUIRoomDefine.h 中

		TUIVideoStreamType的定义
view	TUIVideoView *	视频渲染视图

openLocalCamera

打开本地摄像头。



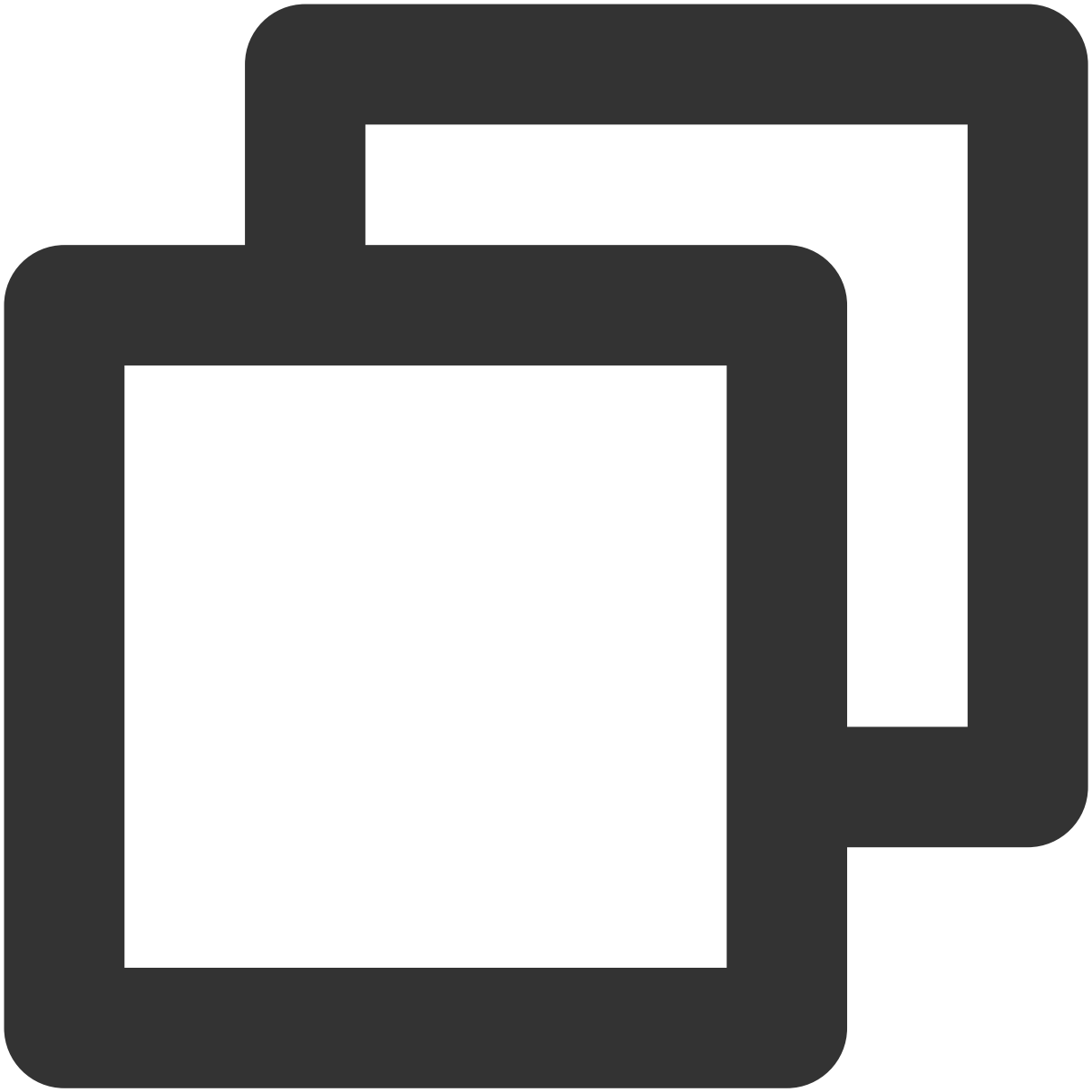
```
- (void)openLocalCamera:(BOOL)isFront
                quality:(TUIVideoQuality)quality
              onSuccess:(TUISuccessBlock)onSuccess
```

```
                onError:(TUIErrorBlock)onError;
```

参数	类型	含义
isFront	BOOL	是否为前置
quality	TUIVideoQuality	视频质量，详细定义可以参考TUIRoomDefine.h 中 TUIVideoQuality 的定义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

closeLocalCamera

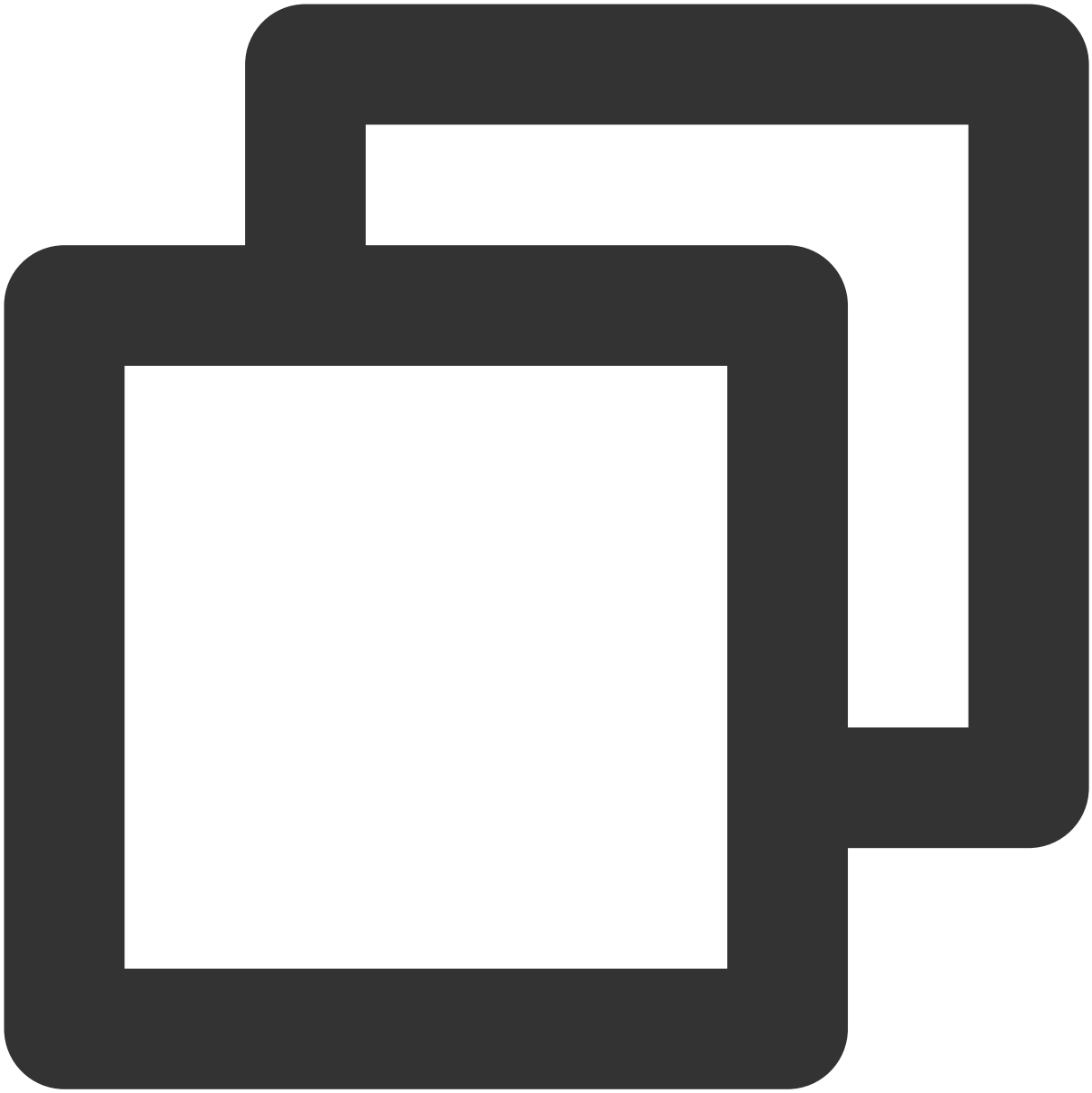
关闭本地摄像头。



```
- (void)closeLocalCamera;
```

updateVideoQuality

更新本地视频编码质量设置。

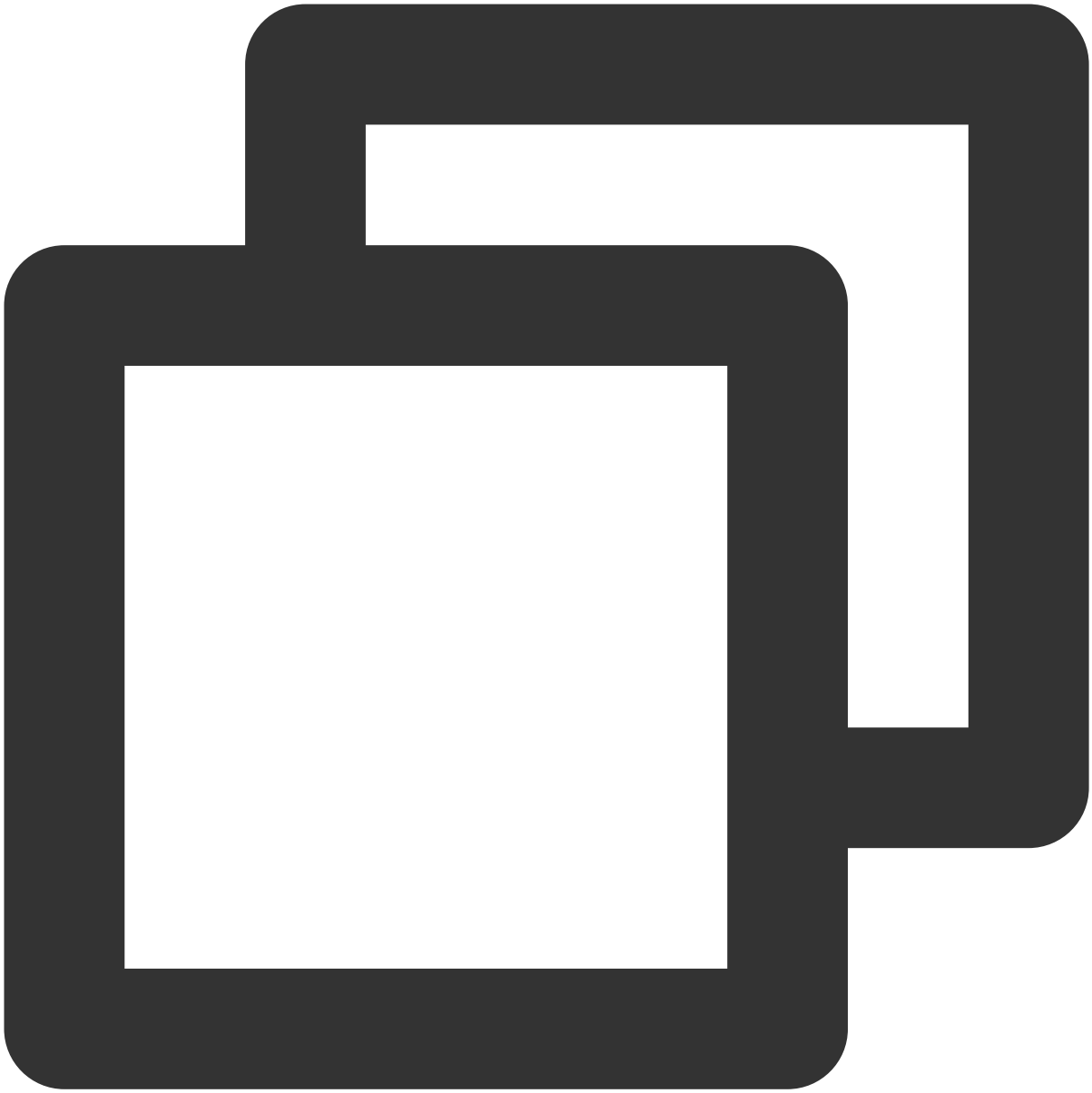


```
- (void)updateVideoQuality:(TUIVideoQuality)quality;
```

参数	类型	含义
quality	TUIVideoQuality	视频质量

openLocalMicrophone

打开本地麦克风。



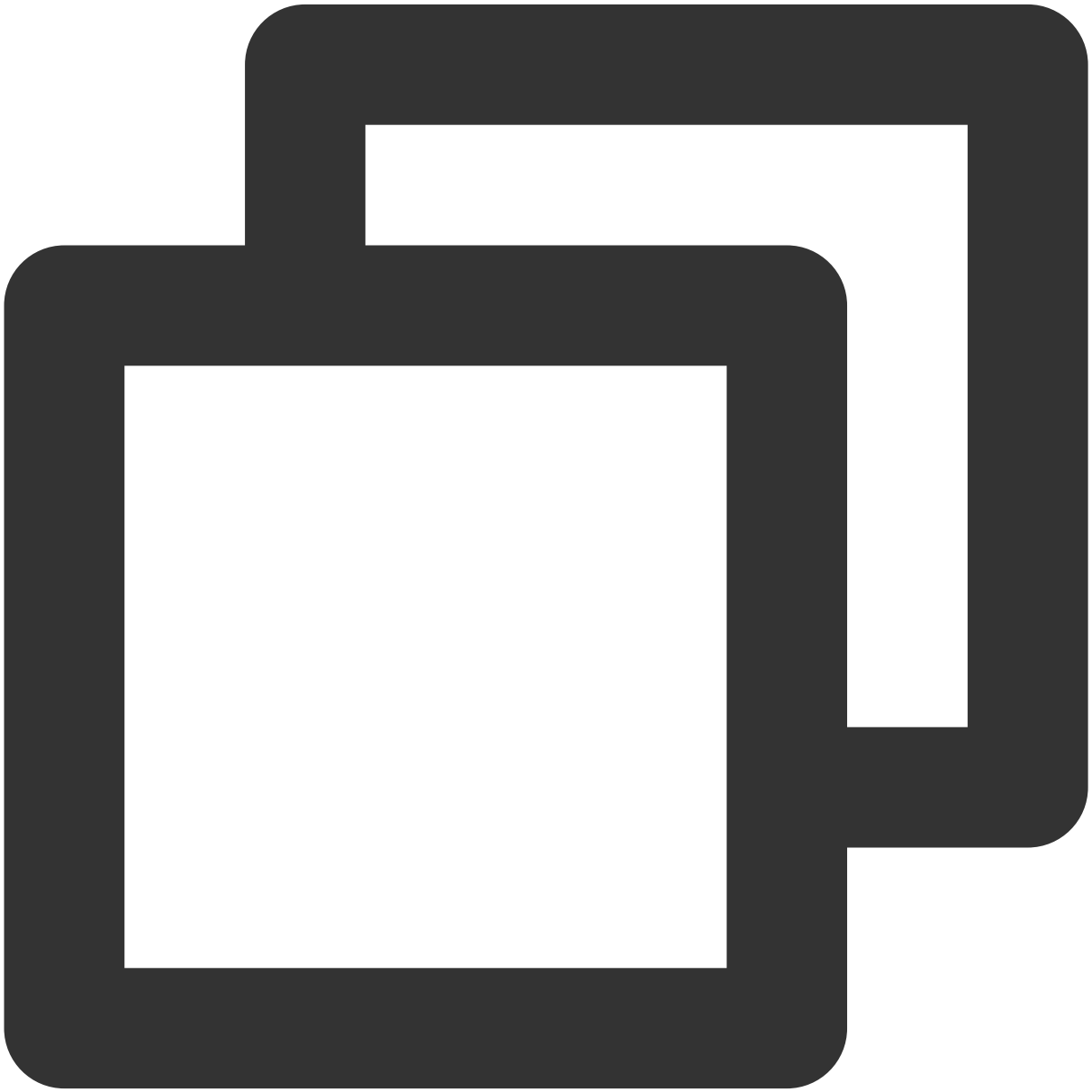
```
- (void)openLocalMicrophone:(TUIAudioQuality)quality
                  onSuccess:(TUISuccessBlock)onSuccess
                    onError:(TUIErrorBlock)onError;
```

参数	类型	含义
quality	TUIAudioQuality	音频质量，详细定义可以参考 TUIRoomDefine.h 中 TUIAudioQuality 的定义
onSuccess	TUISuccessBlock	成功回调

onError	TUIErrorBlock	失败回调

closeLocalMicrophone

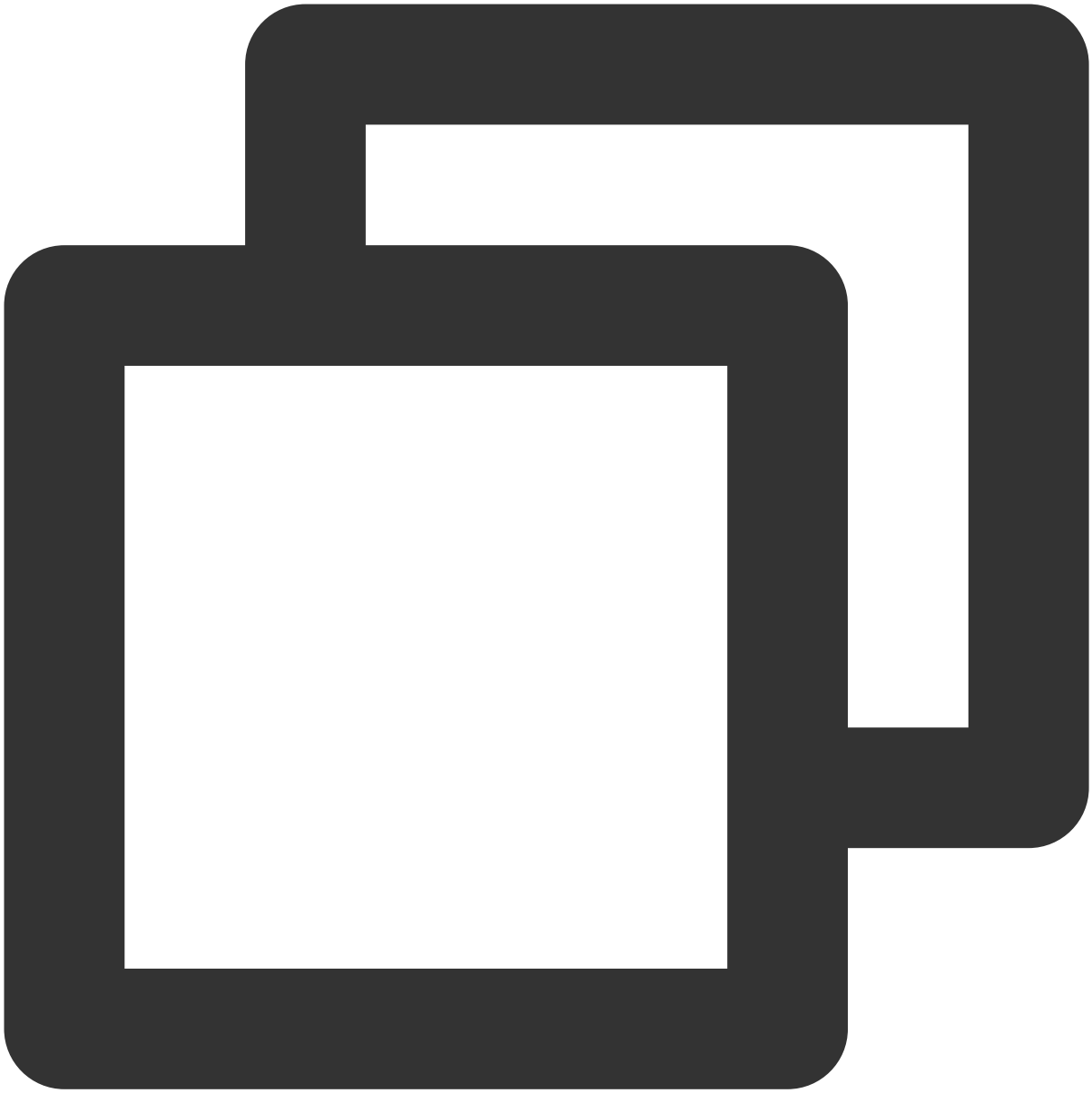
关闭本地麦克风。



```
- (void)closeLocalMicrophone;
```

updateAudioQuality

更新本地音频编码质量设置。



```
- (void)updateAudioQuality:(TUIAudioQuality)quality;
```

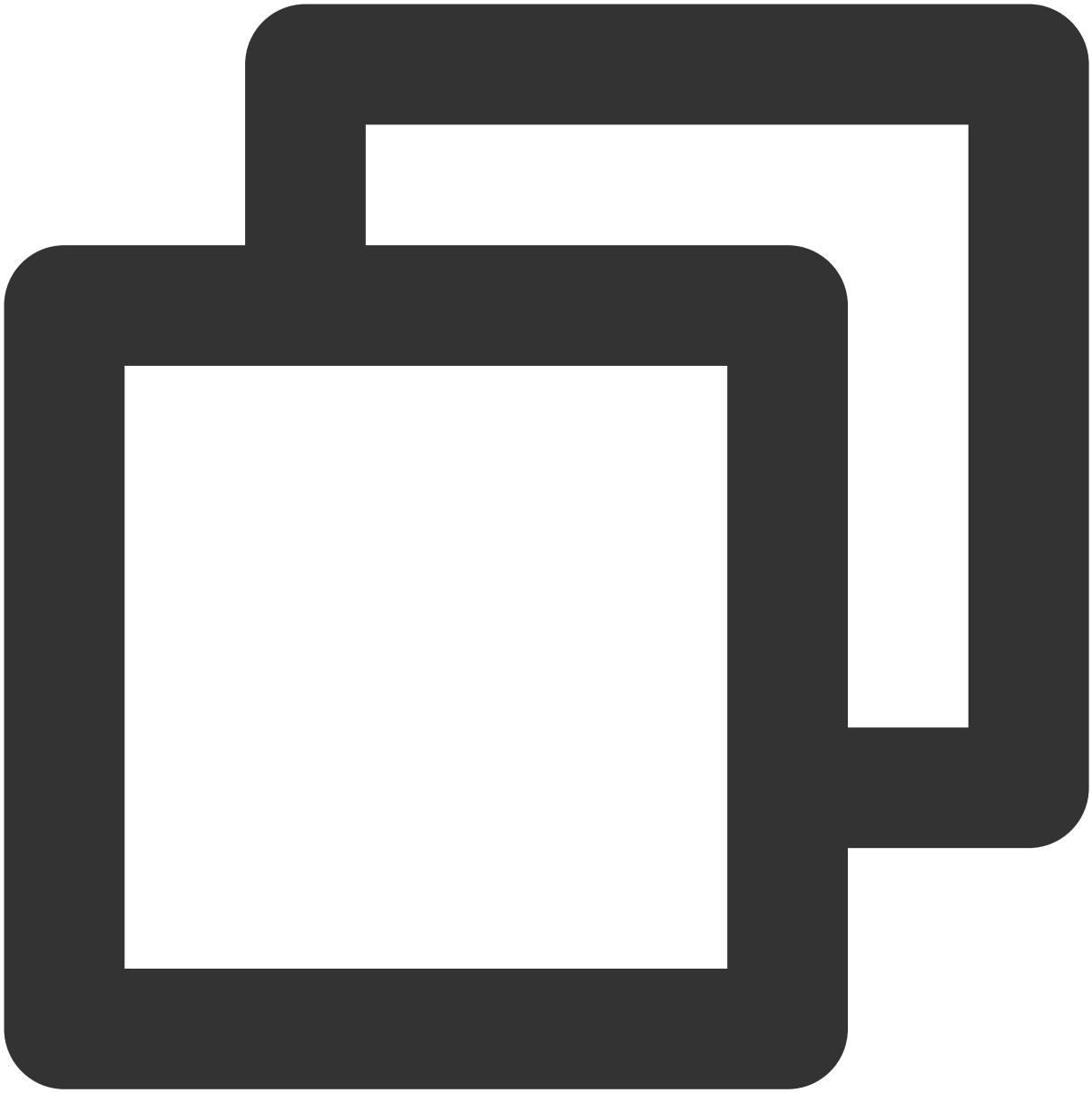
参数	类型	含义
quality	TUIAudioQuality	音频质量

startScreenCapture

开始屏幕分享（仅支持移动端 iOS 11.0 及以上系统）

该接口支持抓取整个 iOS 系统的屏幕，可以实现类似腾讯会议的全系统级的屏幕分享。

版权所有：腾讯云计算（北京）有限责任公司



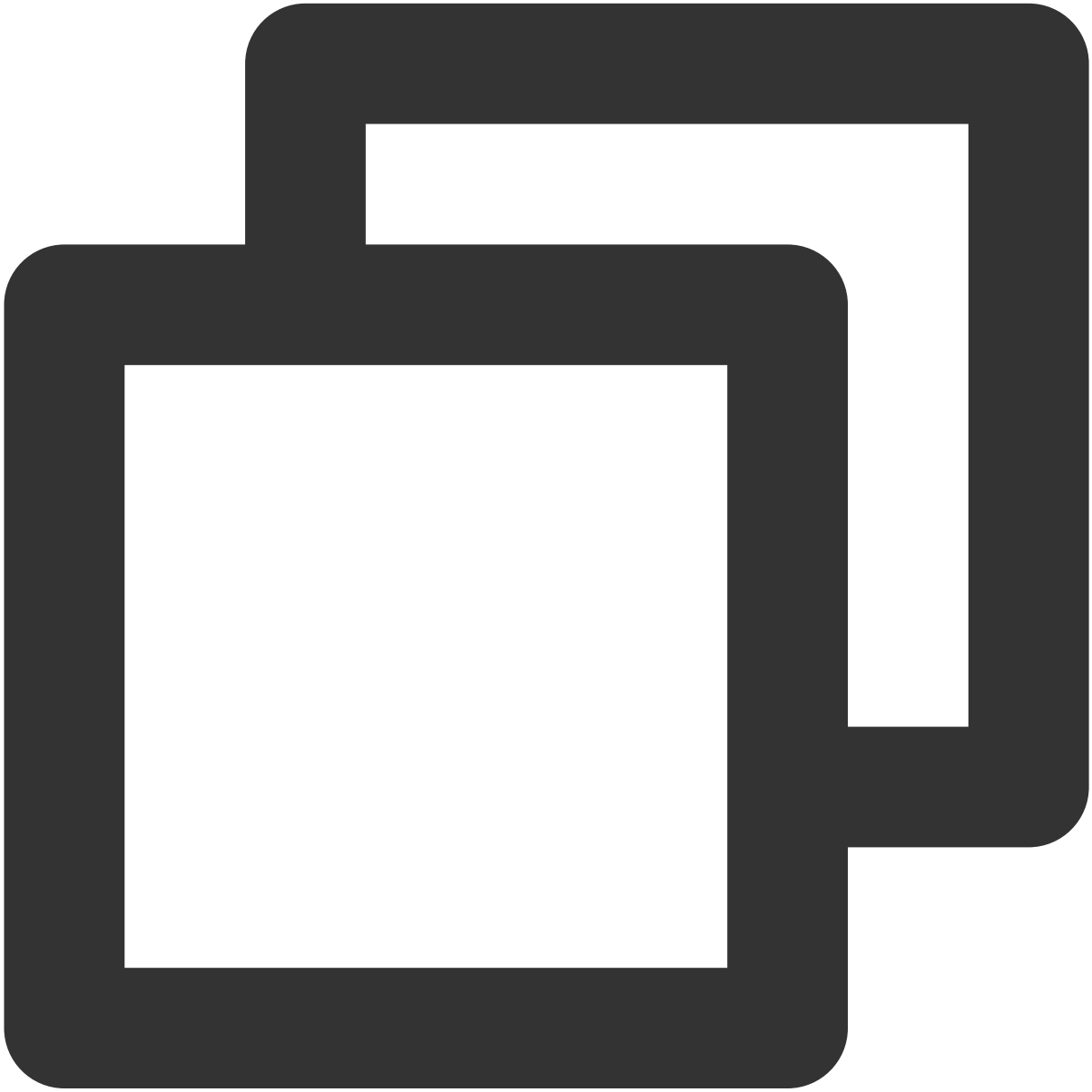
```
- (void)startScreenCaptureByReplaykit:(NSString *)appGroup API_AVAILABLE(ios(11.0))
```

参数如下表所示：

参数	类型	含义
appGroup	NSString *	用于指定您的应用与录屏进程共享的 Application Group Identifier，您可以指定该参数为 nil，但推荐您按照文档指示进行设置，从而获得更好的可靠性。

startScreenCapture

开始屏幕分享（该接口仅支持桌面端 Mac OS系统）
该接口可以抓取整个 Mac OS 系统的屏幕内容，或抓取您指定的某个应用的窗口内容，并将其分享给同房间中的其他用户。



```
- (void)startScreenCapture:(TUIVideoView *)view
                 onSuccess:(TUISuccessBlock)onSuccess
                   onError:(TUIErrorBlock)onError;
```

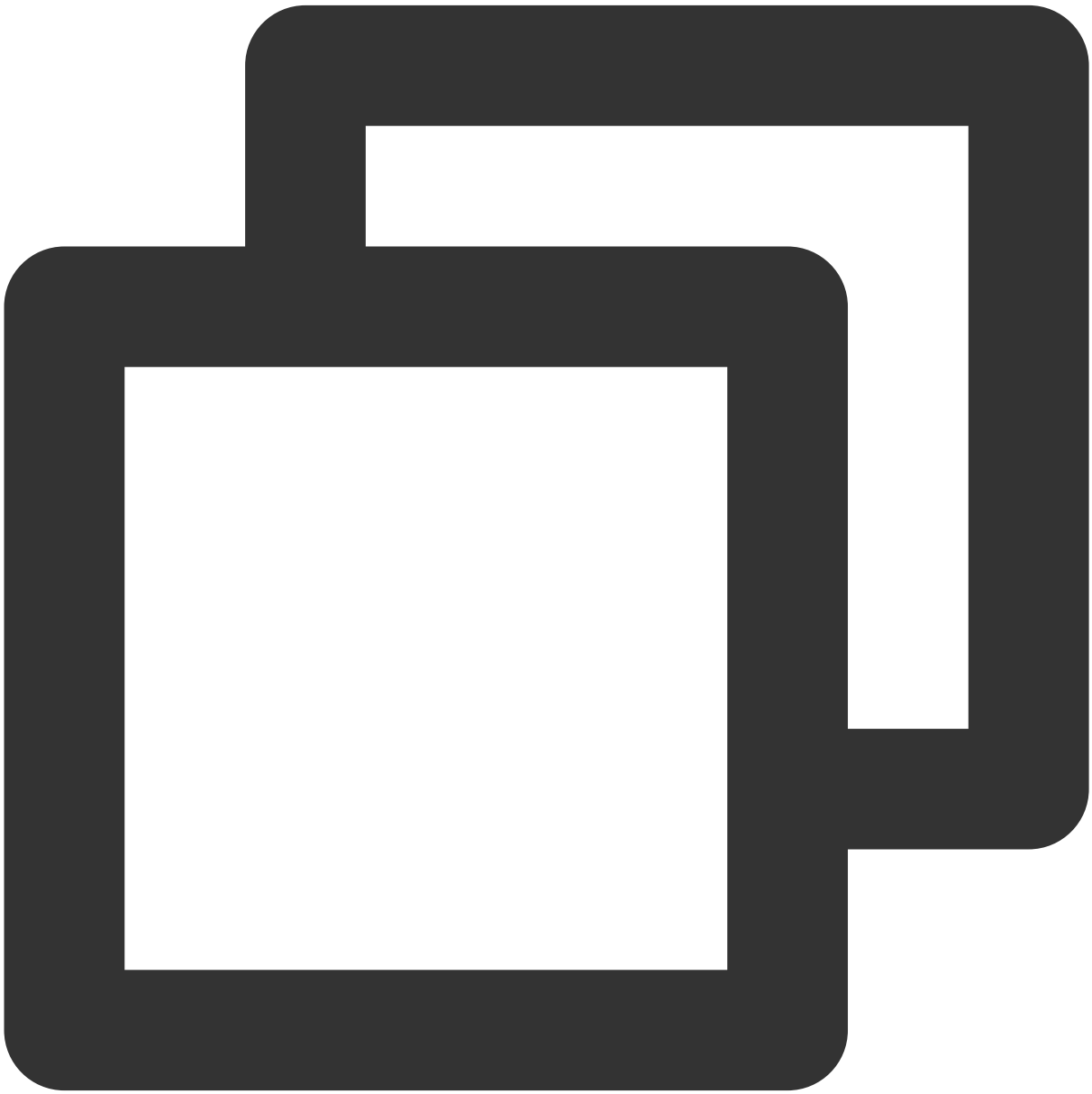
参数如下表所示：

参数	类型	含义

view	TUIVideoView *	渲染控件所在的父控件，可以设置为空值，表示不显示屏幕分享的预览效果。
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

stopScreenCapture

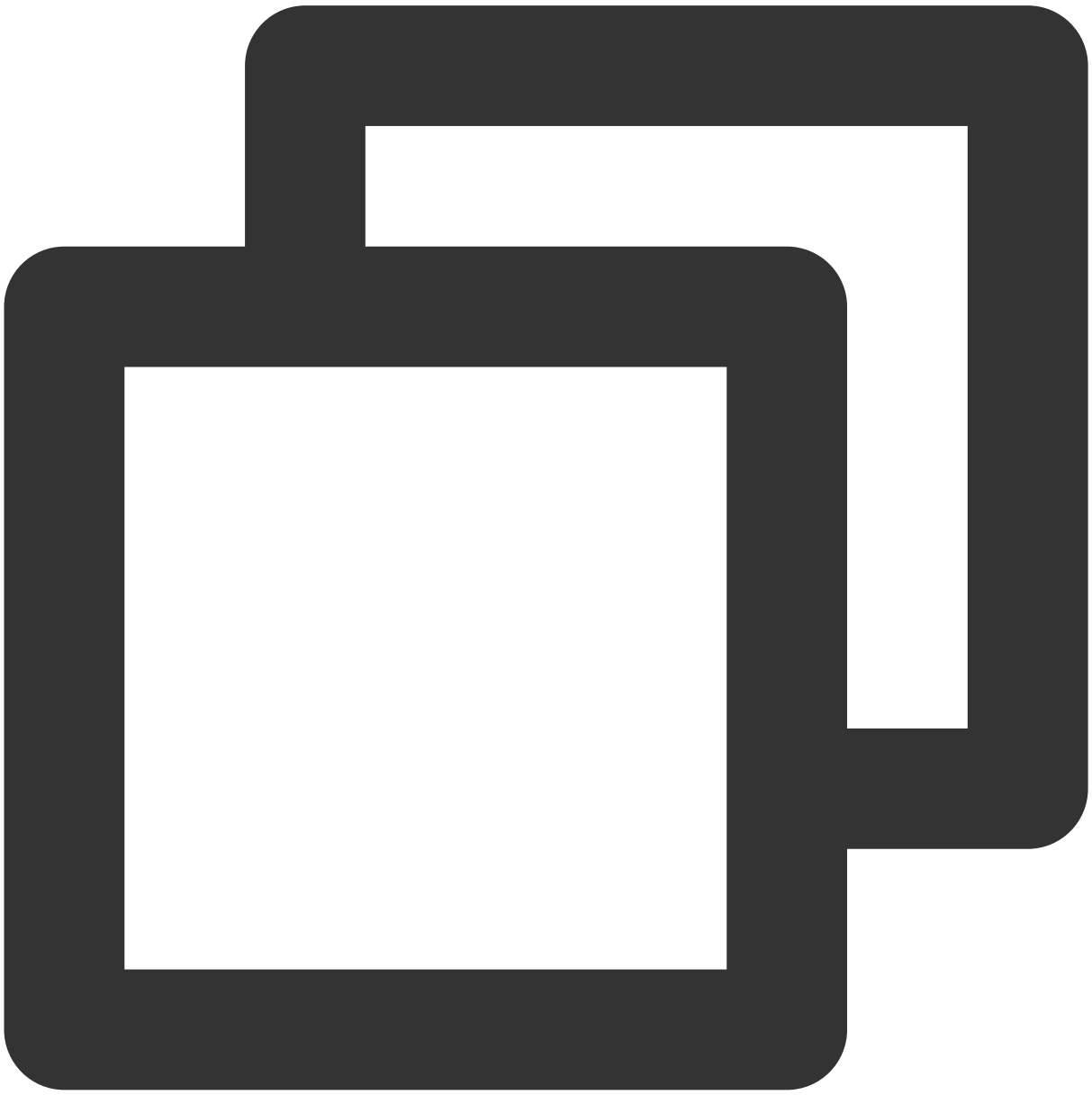
结束屏幕分享。



```
- (void)stopScreenCapture;
```

getScreenCaptureSources

枚举可分享的屏幕和窗口（该接口仅支持 Mac OS 系统）
当您在对接桌面端系统的屏幕分享功能时，一般都需要展示一个选择分享目标的界面，这样用户能够使用这个界面选择是分享整个屏幕还是某个窗口。
通过本接口，您就可以查询到当前系统中可用于分享的窗口的 ID、名称以及缩略图。我们在 Demo 中提供了一份默认的界面实现供您参考。

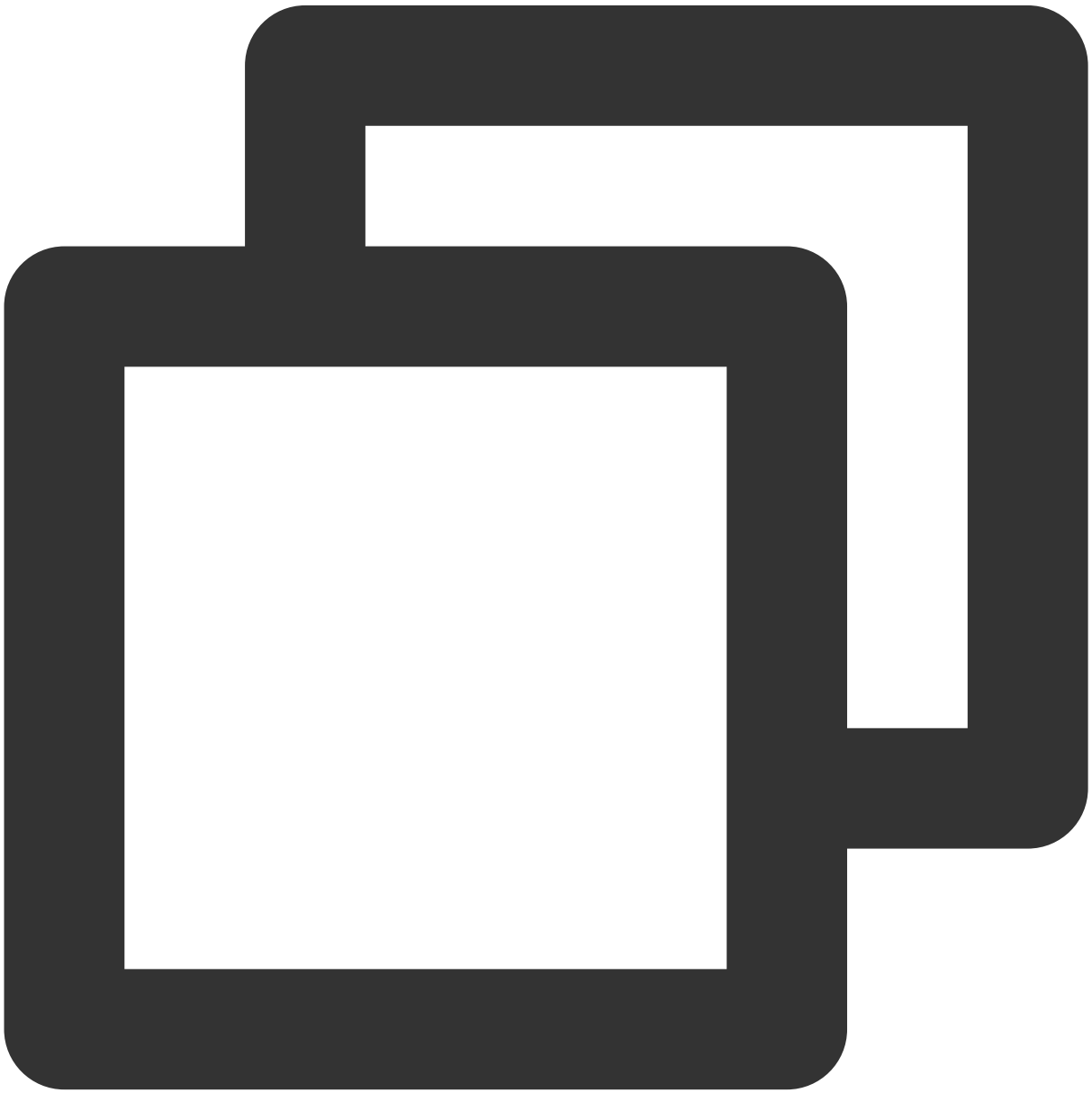


```
- (NSArray<TUIShareTarget *> *)getScreenCaptureSources;
```

返回值	类型	含义
screenCaptureSources	NSArray<TUIShareTarget *> *	窗口列表包括屏幕

selectScreenCaptureTarget

选择屏幕分享对象。



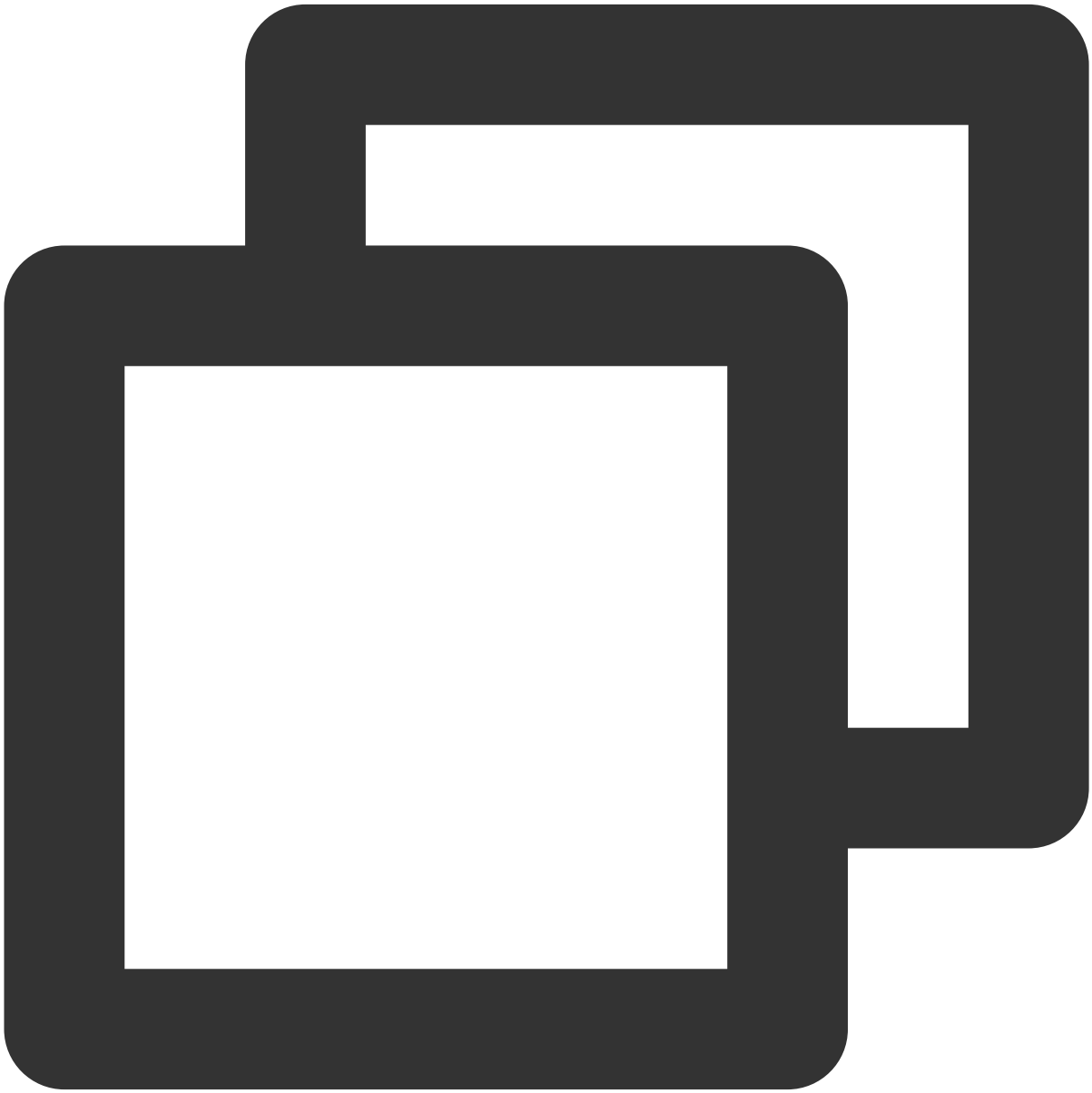
```
- (void)selectScreenCaptureTarget:(NSString *)targetId;
```

参数如下表所示：

参数	类型	含义
targetId	NSString *	指定分享源

startPushLocalVideo

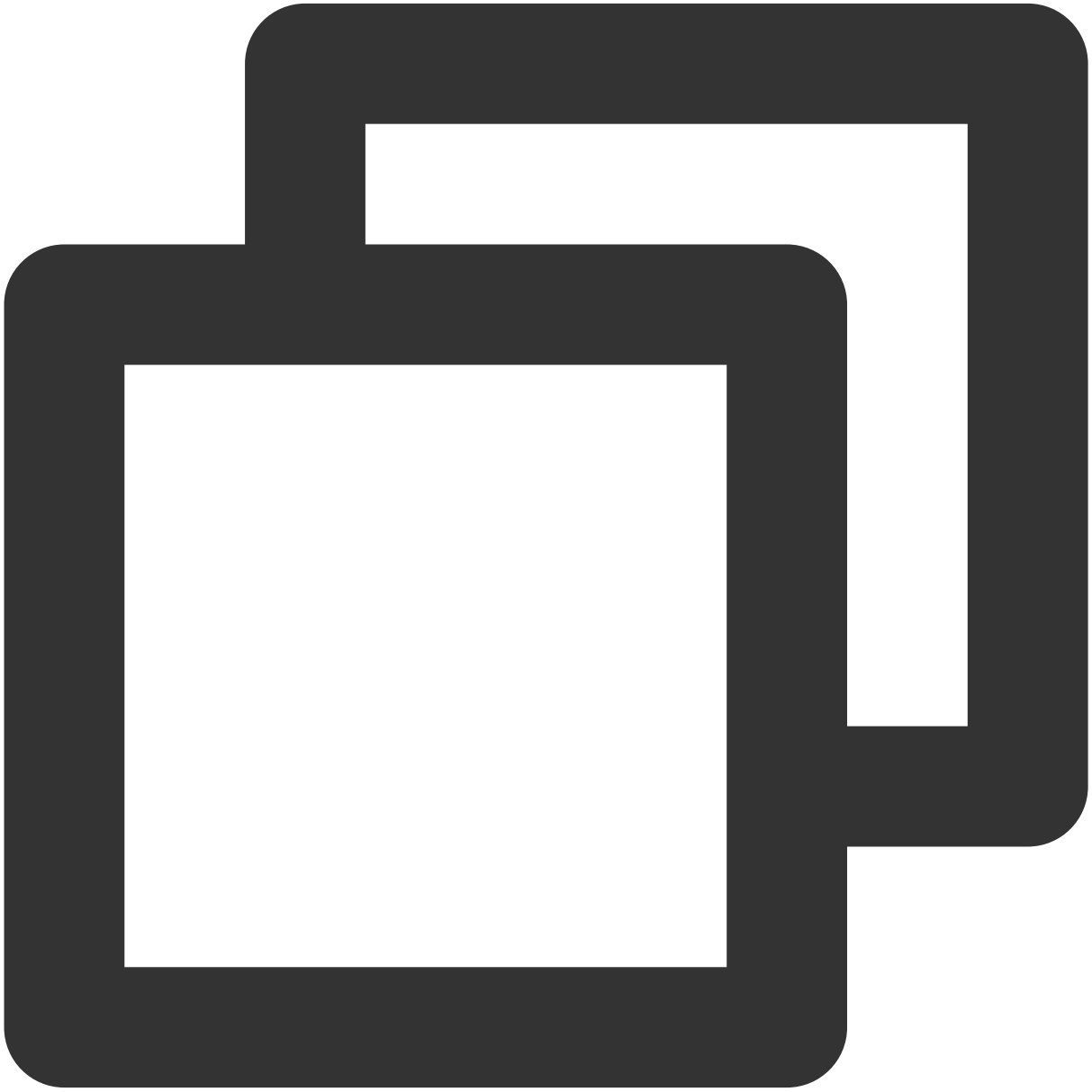
开始推送本地视频。



```
- (void)startPushLocalVideo;
```

stopPushLocalVideo

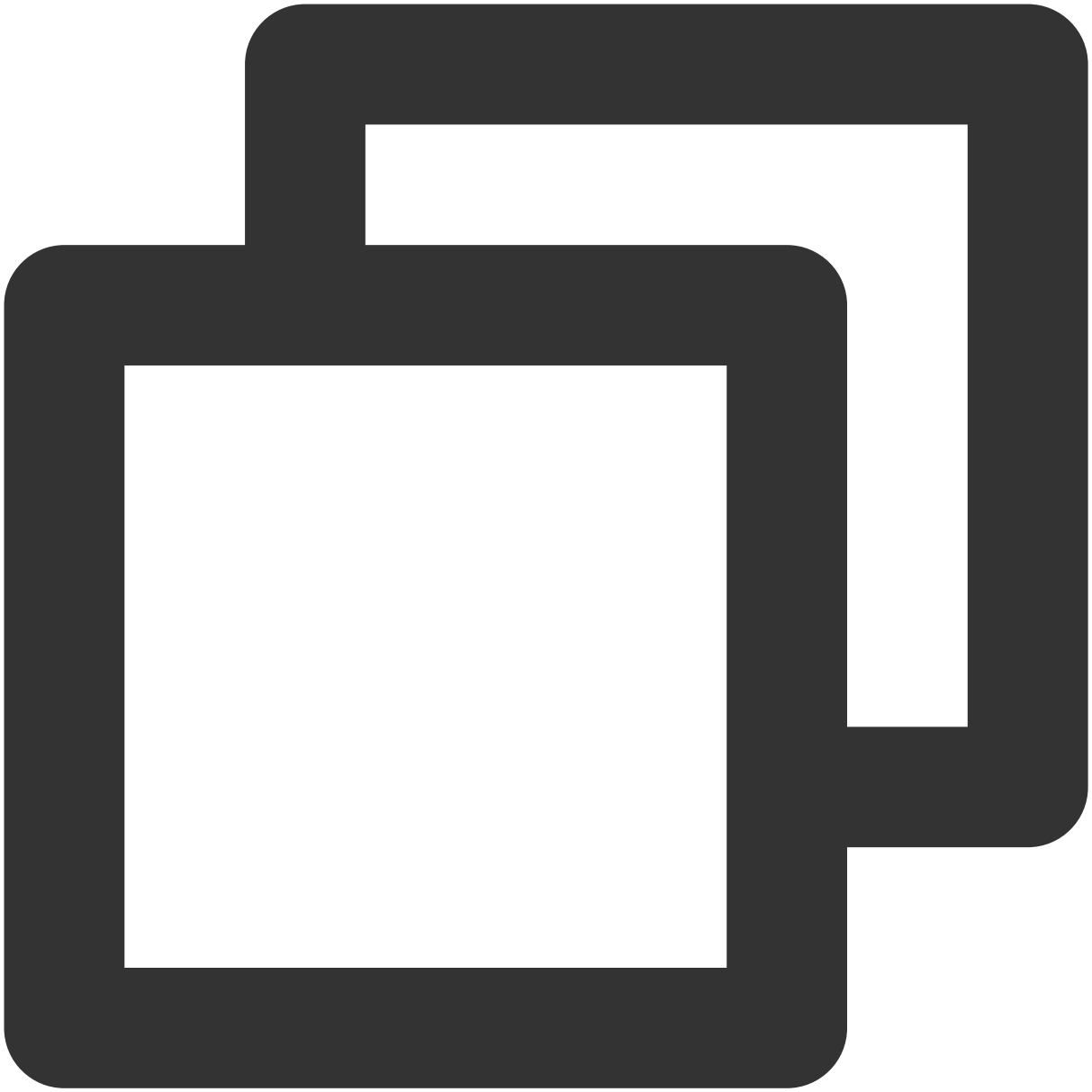
停止推送本地视频。



```
- (void)stopPushLocalVideo;
```

startPushLocalAudio

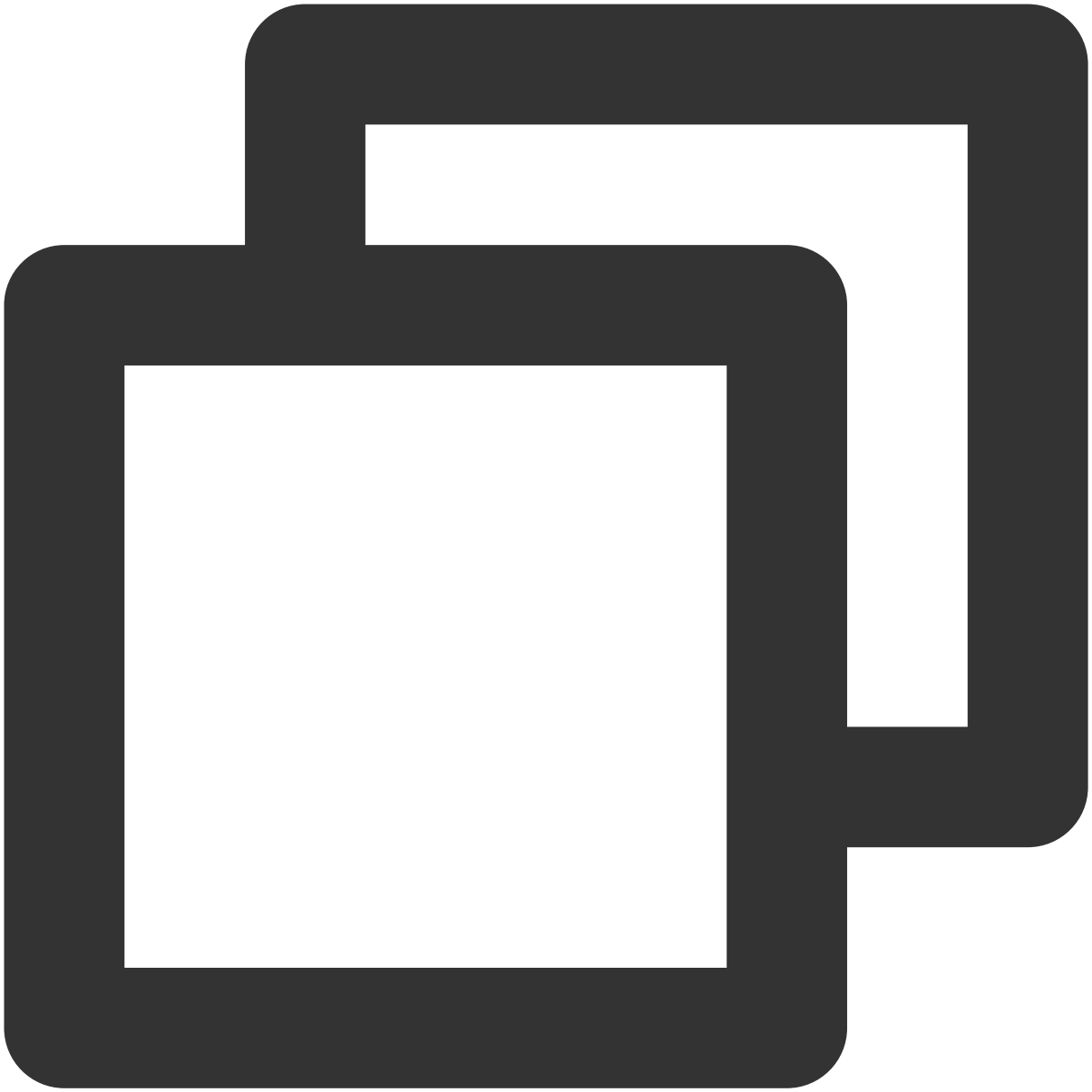
开始推送本地音频。



```
- (void)startPushLocalAudio;
```

stopPushLocalAudio

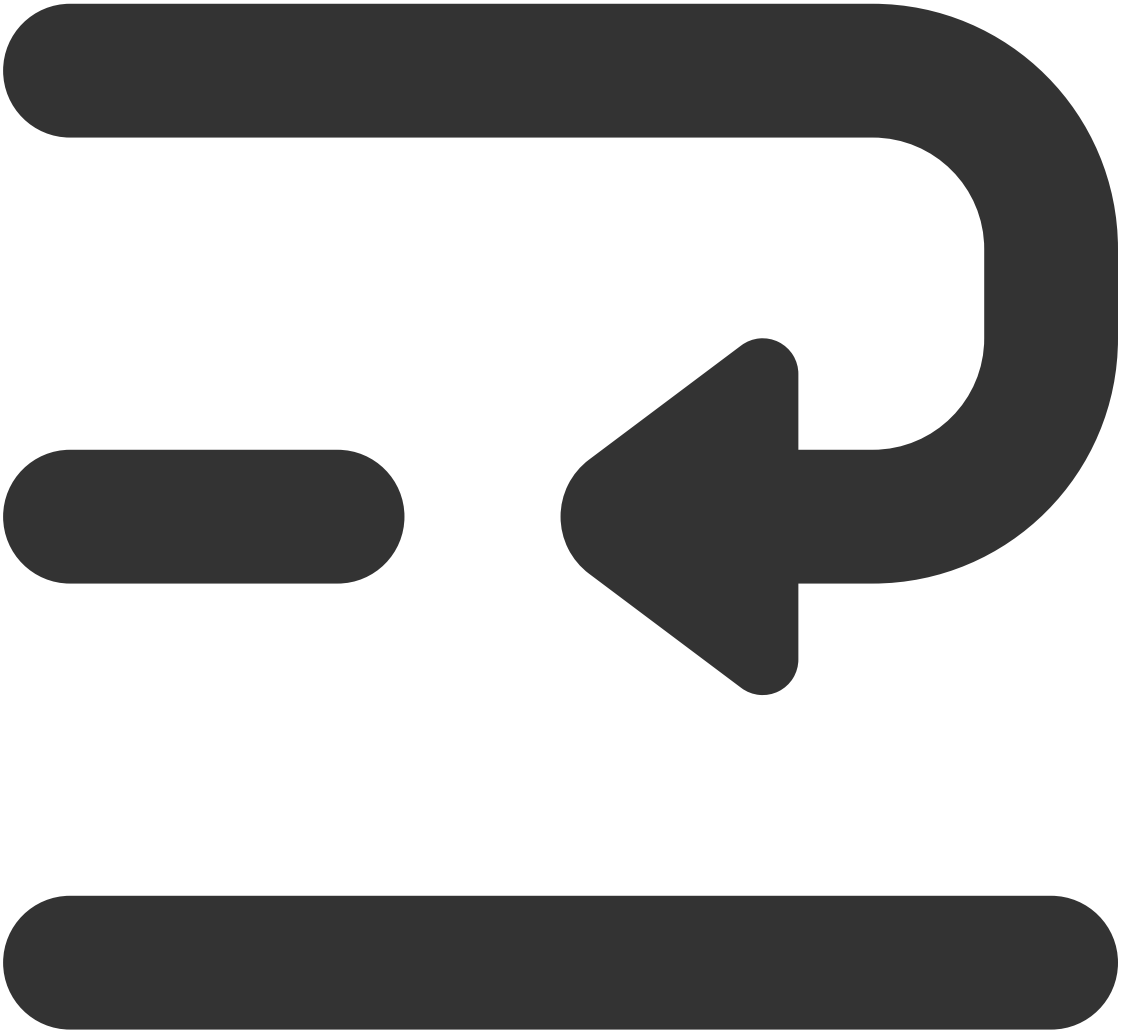
停止推送本地音频。



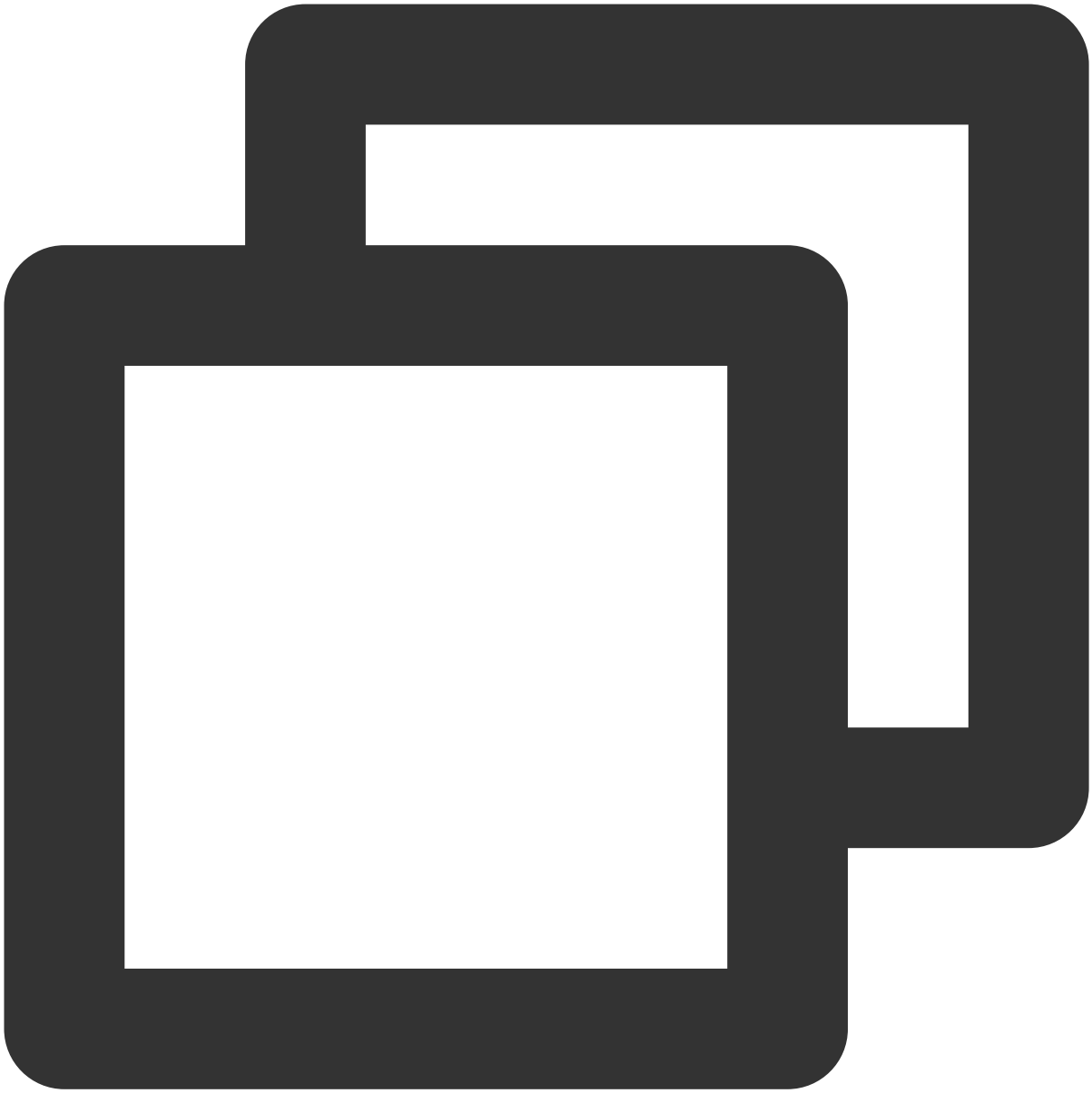
```
- (void)stopPushLocalAudio;
```

startPlayRemoteVideo

开始播放远端用户视频。



版权所有：腾讯云计算（北京）有限责任公司



```
- (void)startPlayRemoteVideo:(NSString *)userId
                  streamType:(TUIVideoStreamType)streamType
                   onPlaying:(TUIPlayOnPlayingBlock)onPlaying
                   onLoading:(TUIPlayOnLoadingBlock)onLoading
                     onError:(TUIPlayOnErrorBlock)onError;
```

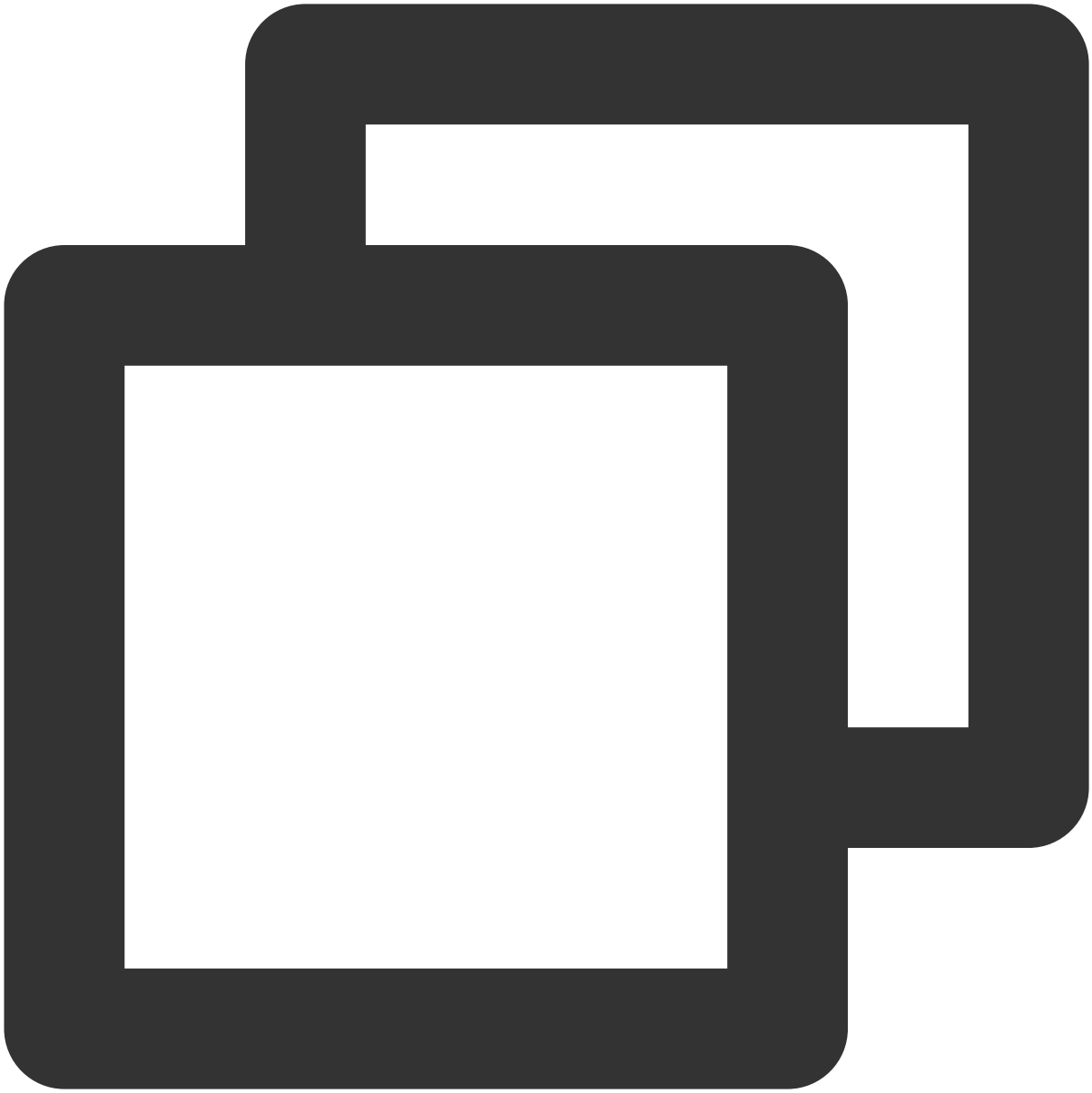
参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID

streamType	TUIVideoStreamType	视频流的类型。详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义
onPlaying	TUIPlayOnPlayingBlock	播放回调
onLoading	TUIPlayOnLoadingBlock	加载回调
onError	TUIPlayOnErrorBlock	错误回调

stopPlayRemoteVideo

停止播放远端用户视频。



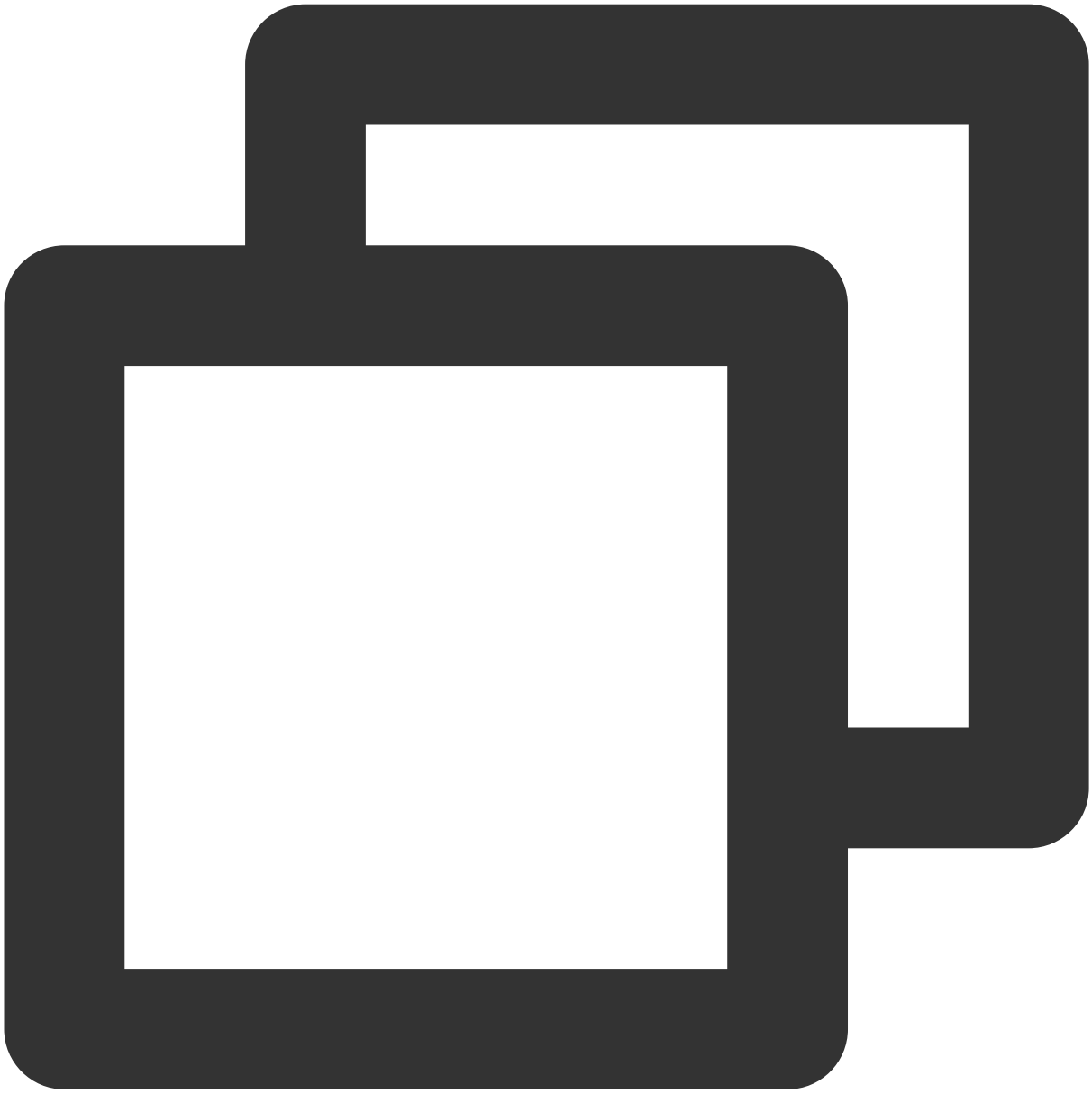
```
- (void)stopPlayRemoteVideo:(NSString *)userId streamType:(TUIVideoStreamType)strea
```

参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID
streamType	TUIVideoStreamType	视频流的类型。详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义

muteRemoteAudioStream

将远端用户禁音

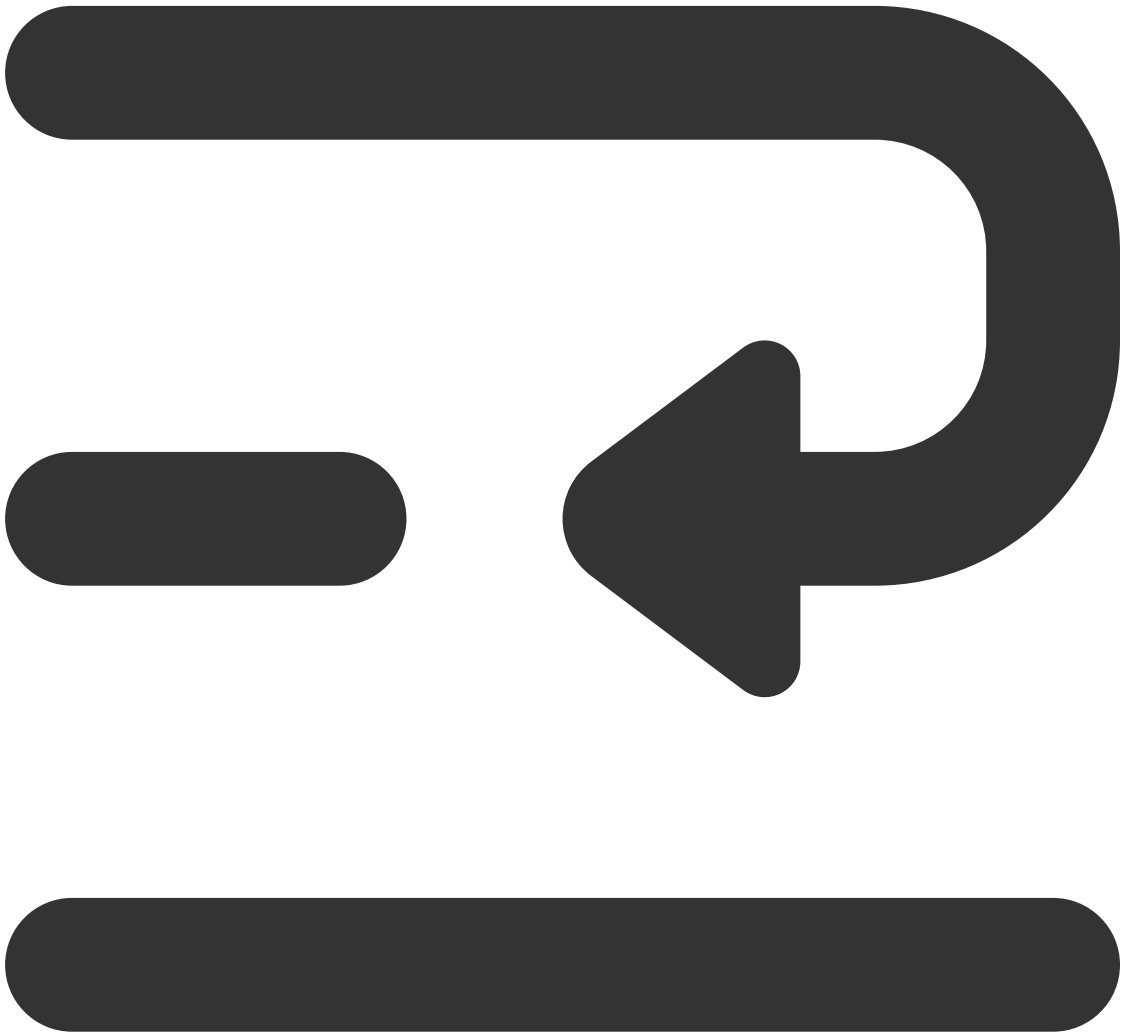


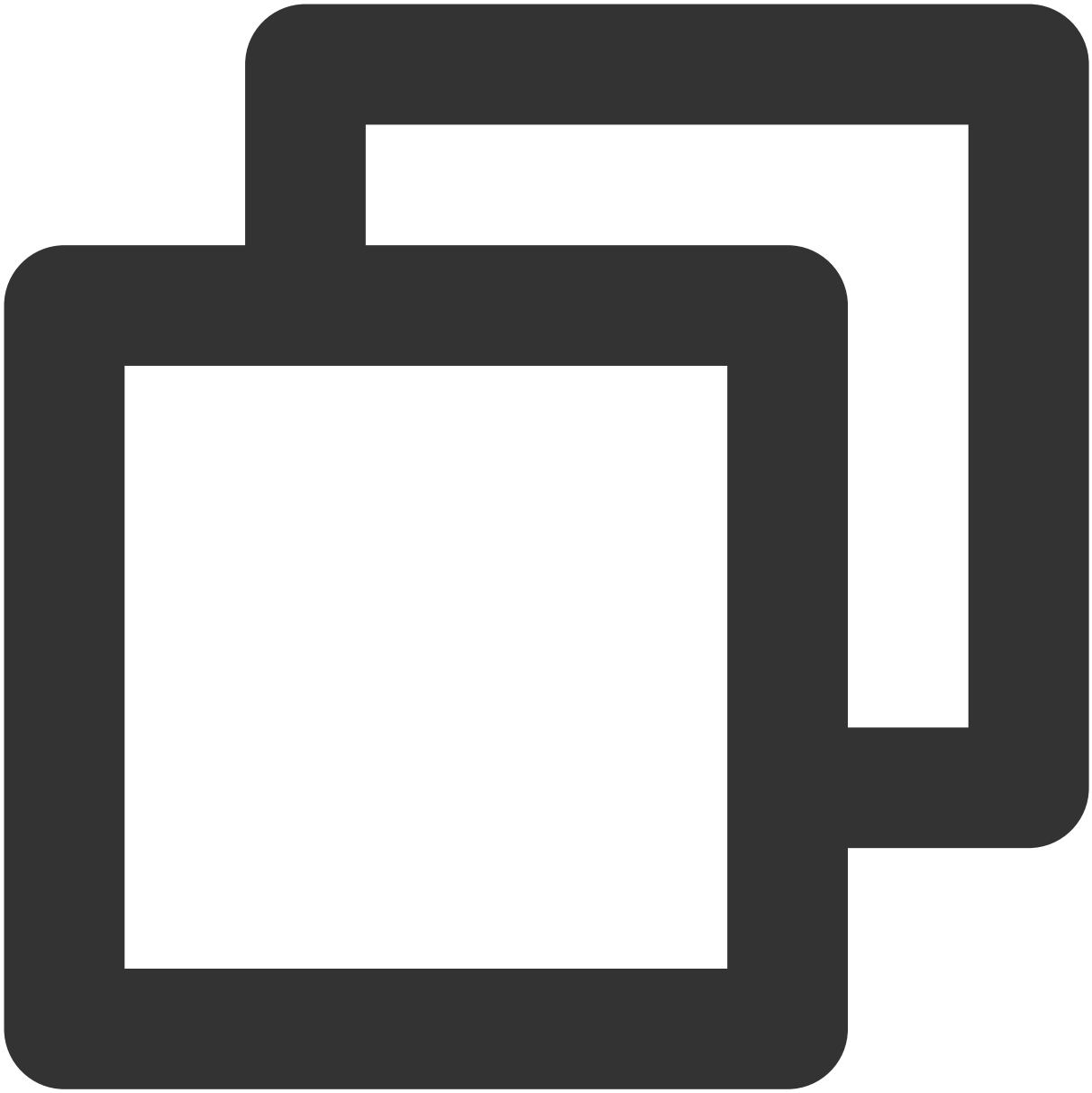
```
- (void)muteRemoteAudioStream:(NSString *)userId isMute:(BOOL)isMute;
```

参数	类型	含义
userId	NSString *	用户 ID
isMute	BOOL	是否禁音

getUserList

获取房间内的成员列表。





```
- (void)getUserList:(NSInteger)nextSequence
          onSuccess:(TUIUserListResponseBlock)onSuccess
            onError:(TUIErrorBlock)onError;
```

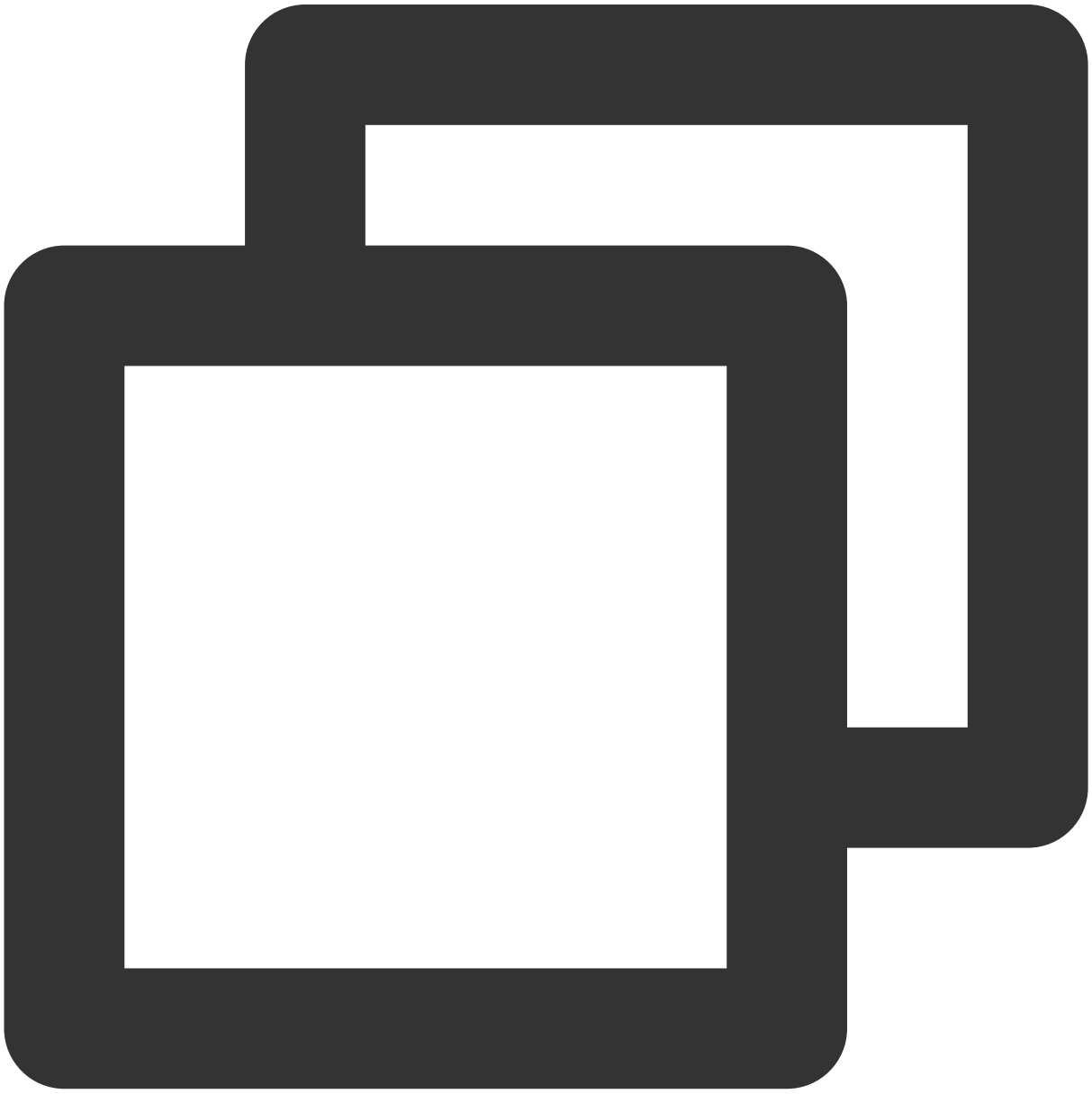
参数如下表所示：

参数	类型	含义
nextSequence	NSInteger	分页拉取标志，第一次拉取填0，回调成功如果 callback 返回 TUIUserListResult 中 next_sequence 不为零，需要分页，传入再次拉取，直至为0

onSuccess	TUIUserListResponseBlock	成功回调。详细定义可以参考 TUIRoomDefine.h 中 TUIUserListResponseBlock 的定义
onError	TUIErrorBlock	错误回调

getUserInfo

获取成员信息。



```
- (void)getUserInfo:(NSString *)userId
          onSuccess:(TUIUserInfoBlock)onSuccess
```

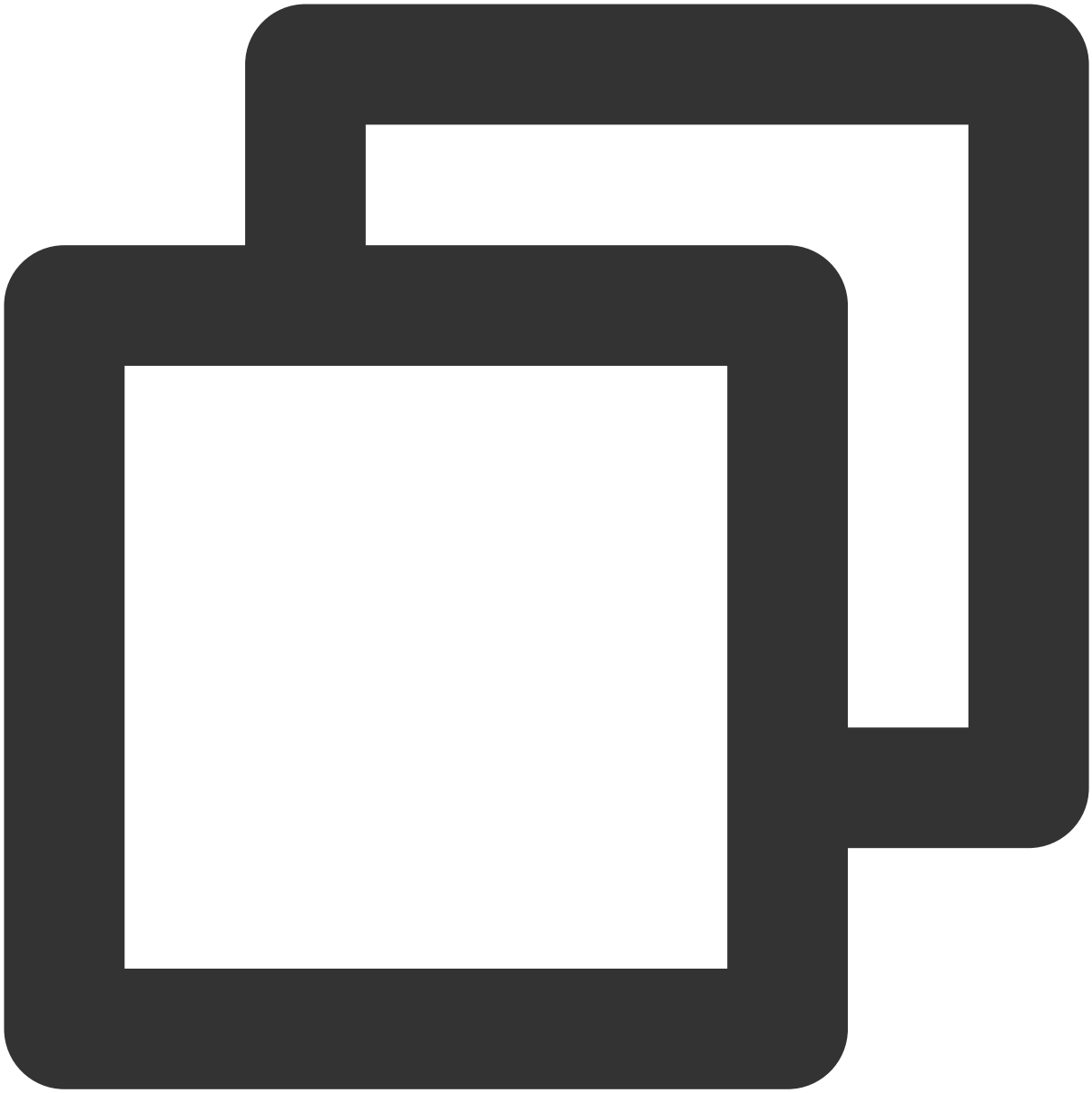
```
            onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID
onSuccess	TUIUserInfoBlock	成功回调。详细定义可以参考 TUIRoomDefine.h 中 TUIUserInfoBlock 定义
onError	TUIErrorBlock	错误回调

changeUserRole

修改用户角色（只有管理员或群主能够调用）。



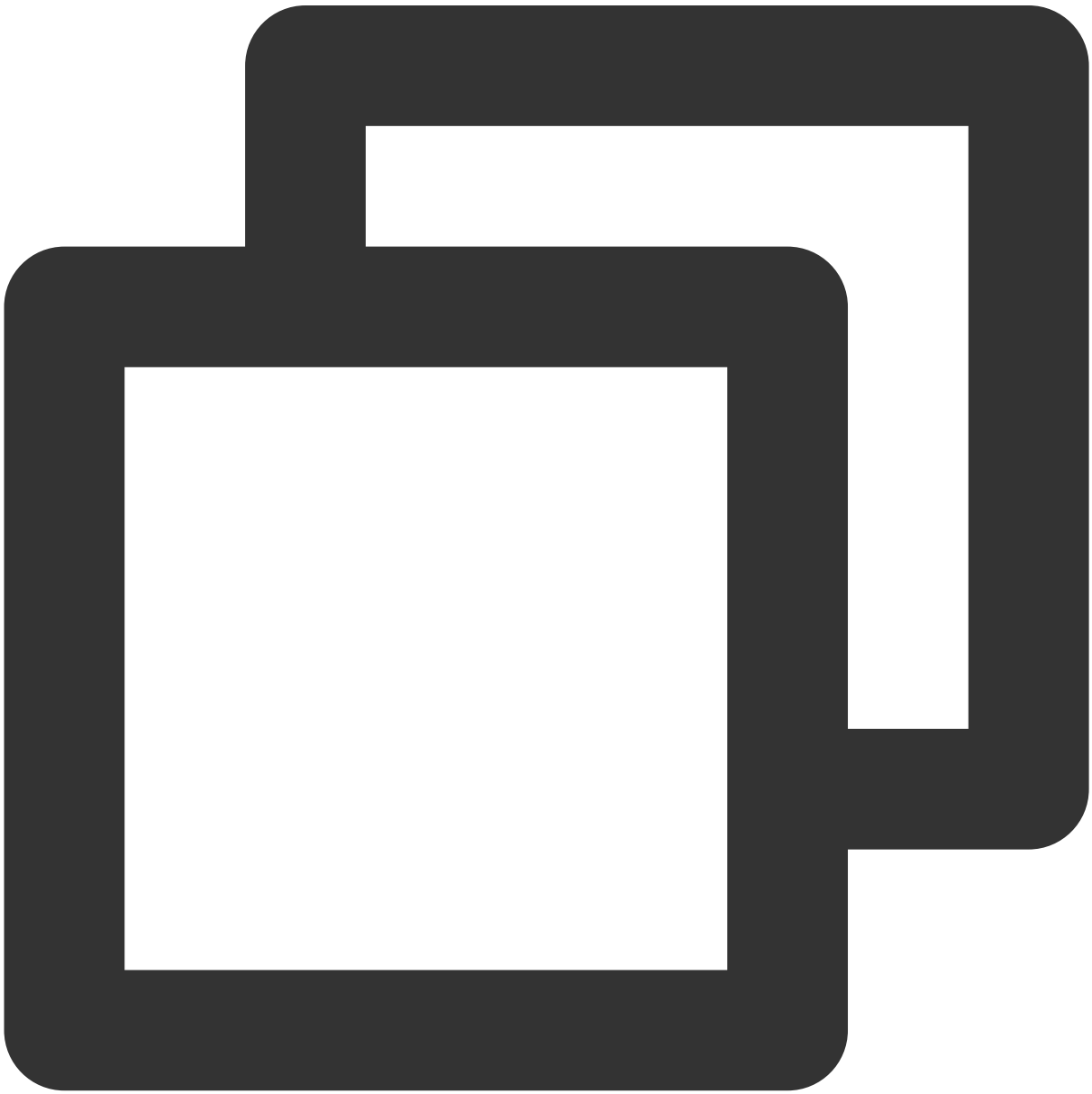
```
- (void)changeUserRoleWithUserId:(NSString *)userId
                            role:(TUIRole)role
                       onSuccess:(TUISuccessBlock)onSuccess
                         onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID

role	TUIRole	需要切换的用户角色
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

kickRemoteUserOutOfRoom



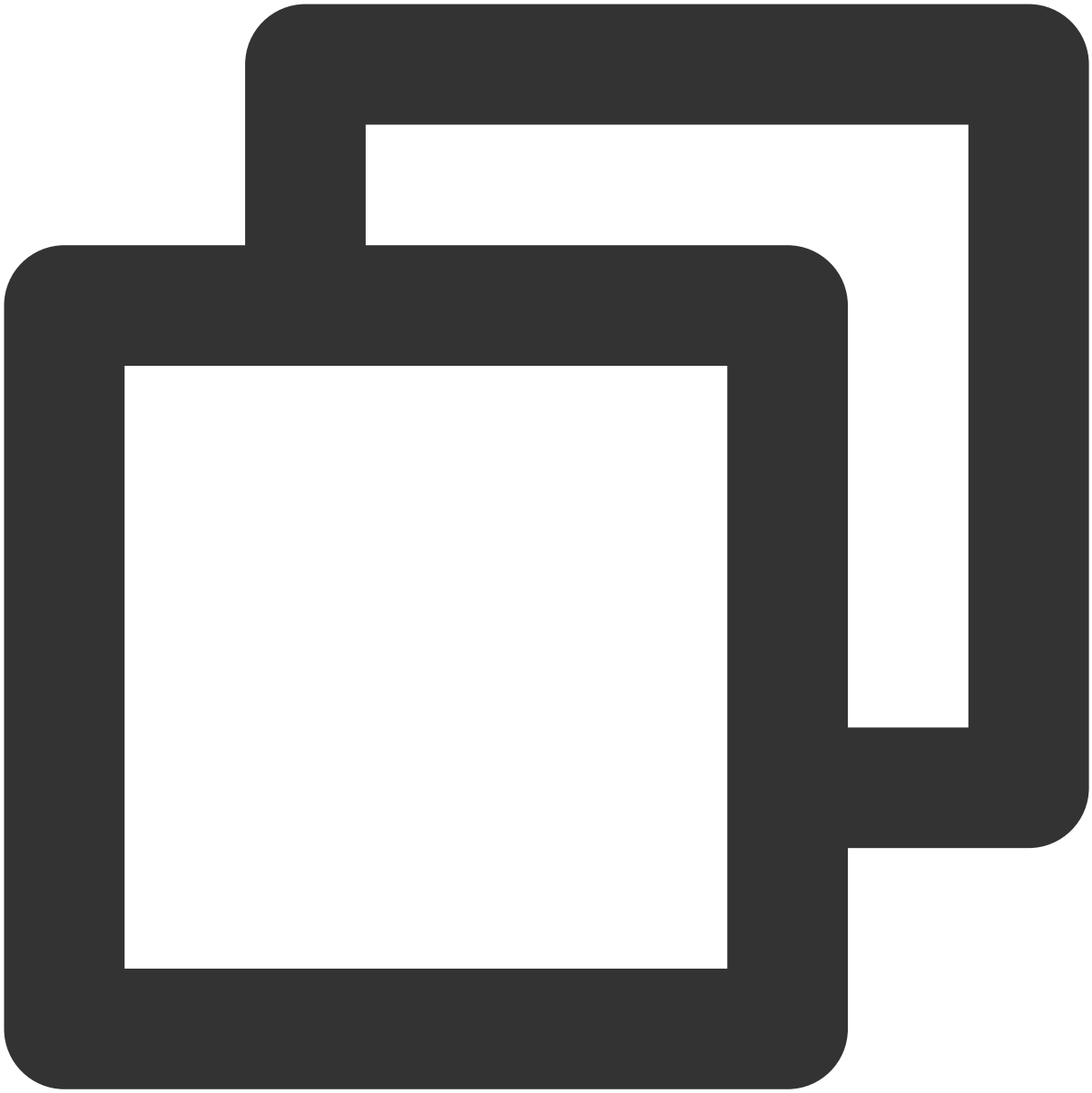
```
- (void)kickRemoteUserOutOfRoom:(NSString *)userId
                        onSuccess:(TUISuccessBlock)onSuccess
                          onError:(TUIErrorBlock)onError;
```

将远端用户踢出房间（只有管理员或群主能够调用）。

参数	类型	含义
userId	NSString *	用户 ID
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

disableDeviceForAllUserByAdmin

控制当前房间内所有用户是否可打开音频流、视频流采集设备的权限状态，例如：全员禁止打开麦克风、全员禁止打开摄像头、全员禁止打开屏幕分享
（目前仅会议场景下可用，并且只有管理员或群主能够调用）。



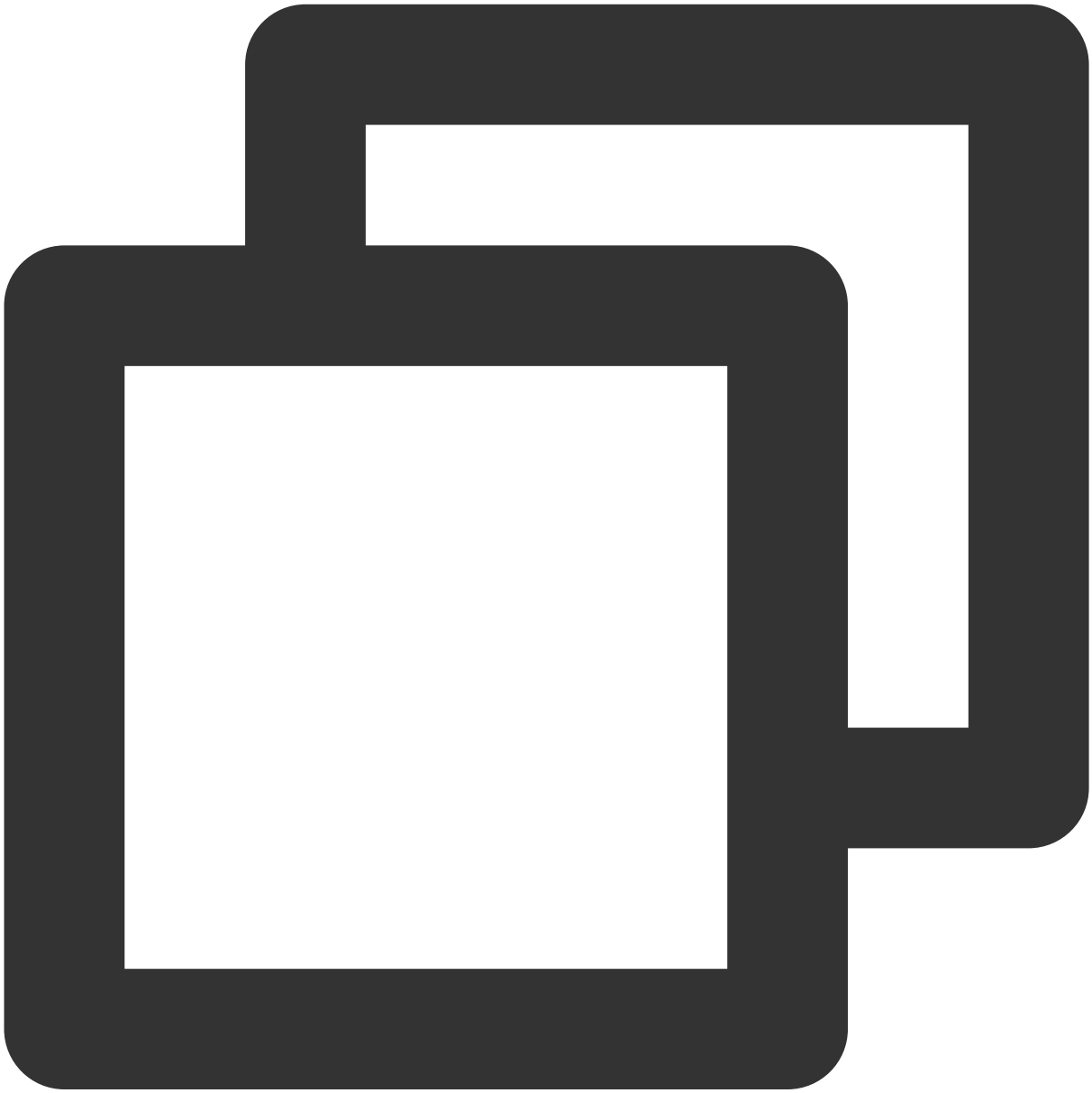
```
- (void)disableDeviceForAllUserByAdmin:(TUIMediaDevice)device
                             isDisable:(BOOL)isDisable
                             onSuccess:(TUISuccessBlock)onSuccess
                               onError:(TUIErrorBlock)onError;
```

参数	类型	含义
device	TUIMediaDevice	设备类型
isDisable	BOOL	是否禁用

onSuccess	TUISuccessBlock	操作成功回调
onError	TUIErrorBlock	操作失败回调

openRemoteDeviceByAdmin

请求远端用户打开媒体设备（只有管理员或群主能够调用）。



```
- (TUIRequest *)openRemoteDeviceByAdmin:(NSString *)userId
```

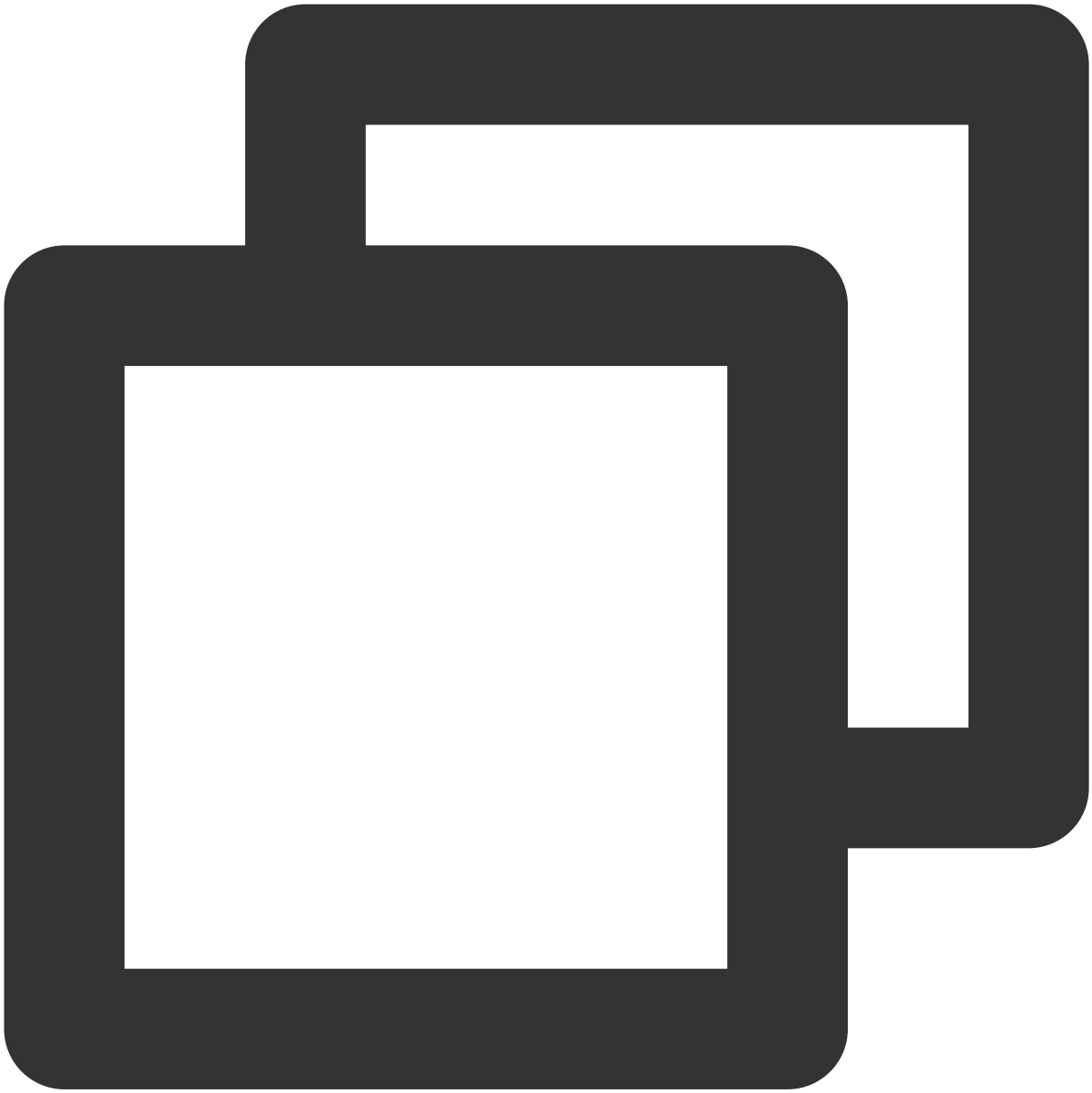
```
                                device:(TUIMediaDevice)device
                               timeout:(NSTimeInterval)timeout
                            onAccepted:(nullable TUIRequestAcceptedBlock)onAccepte
                            onRejected:(nullable TUIRequestRejectedBlock)onRejecte
                           onCancelled:(nullable TUIRequestCancelledBlock)onCancel
                             onTimeout:(nullable TUIRequestTimeoutBlock)onTimeout
                               onError:(nullable TUIRequestErrorBlock)onError;
```

参数	类型	含义
userId	NSString *	用户 ID
device	TUIMediaDevice	媒体设备
timeout	NSTimeInterval	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
onAccepted	nullable TUIRequestAcceptedBlock	邀请被接受的回调
onRejected	nullable TUIRequestRejectedBlock	邀请被拒绝的回调
onCancelled	nullable TUIRequestCancelledBlock	邀请被取消的回调
onTimeout	nullable TUIRequestTimeoutBlock	邀请超时未处理的回调
onError	nullable TUIRequestErrorBlock	邀请发生错误的回调

返回值	类型	含义
request	TUIRequest	请求体

closeRemoteDeviceByAdmin

关闭远端用户媒体设备（只有管理员或群主能够调用）。



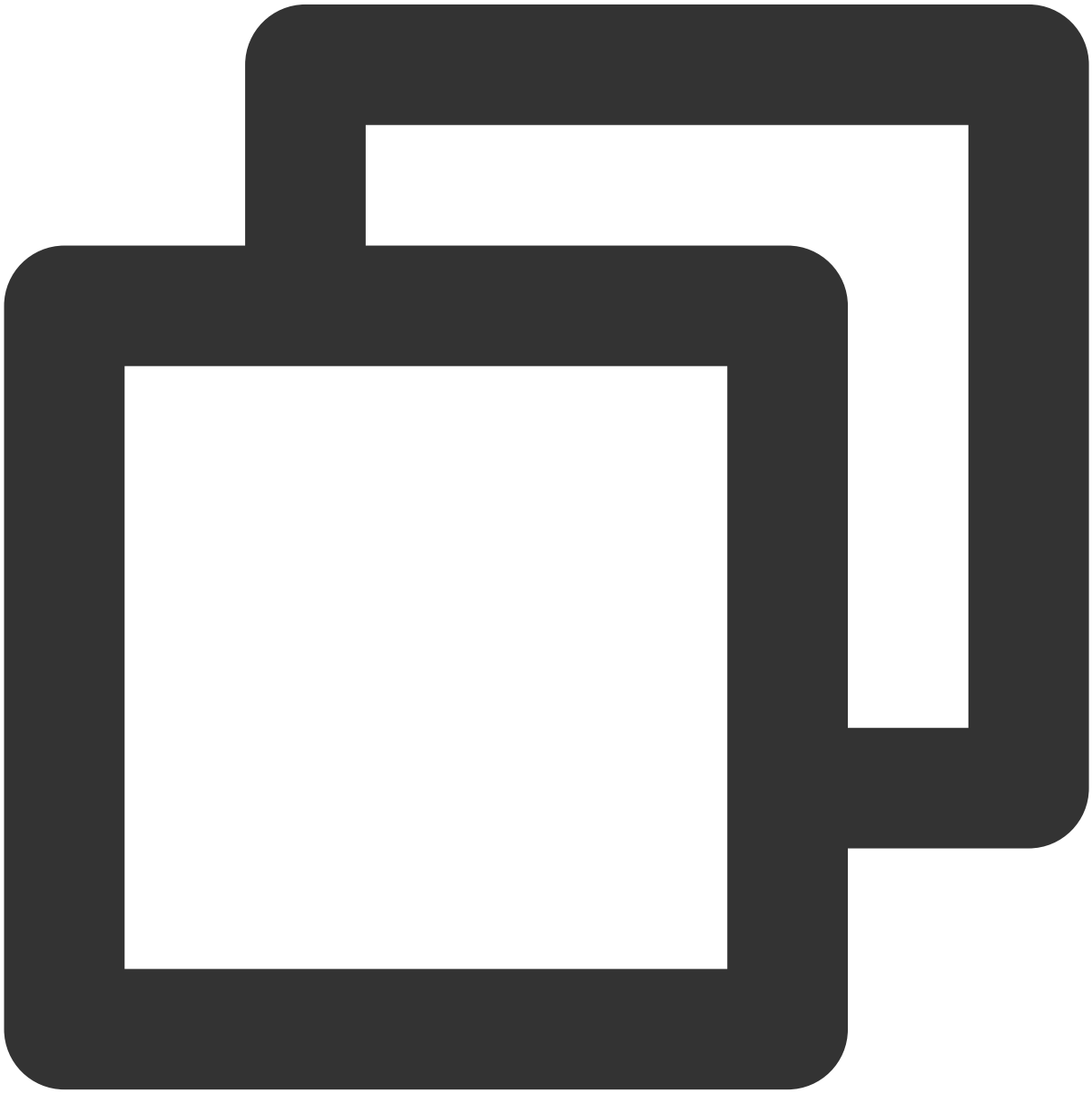
```
- (void)closeRemoteDeviceByAdmin:(NSString *)userId
                          device:(TUIMediaDevice)device
                       onSuccess:(TUISuccessBlock)onSuccess
                         onError:(TUIErrorBlock)onError;
```

参数	类型	含义
userId	NSString *	用户 ID
device	TUIMediaDevice	媒体设备

onSuccess	TUISuccessBlock	调用成功回调
onError	TUIErrorBlock	调用失败回调

applyToAdminToOpenLocalDevice

请求打开本地媒体设备（普通用户可用）。



```
- (TUIRequest *)applyToAdminToOpenLocalDevice:(TUIMediaDevice)device
```

```
                                    timeout:(NSTimeInterval)timeout
                                 onAccepted:(nullable TUIRequestAcceptedBlock)onA
                                 onRejected:(nullable TUIRequestRejectedBlock)onR
                                onCancelled:(nullable TUIRequestCancelledBlock)on
                                  onTimeout:(nullable TUIRequestTimeoutBlock)onTi
                                    onError:(nullable TUIRequestErrorBlock)onErro
```

参数	类型	含义
device	TUIMediaDevice	媒体设备
timeout	NSTimeInterval	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
onAccepted	nullable TUIRequestAcceptedBlock	邀请被接受的回调
onRejected	nullable TUIRequestRejectedBlock	邀请被拒绝的回调
onCancelled	nullable TUIRequestCancelledBlock	邀请被取消的回调
onTimeout	nullable TUIRequestTimeoutBlock	邀请超时未处理的回调
onError	nullable TUIRequestErrorBlock	邀请发生错误的回调

返回值	类型	含义
request	TUIRequest	请求体

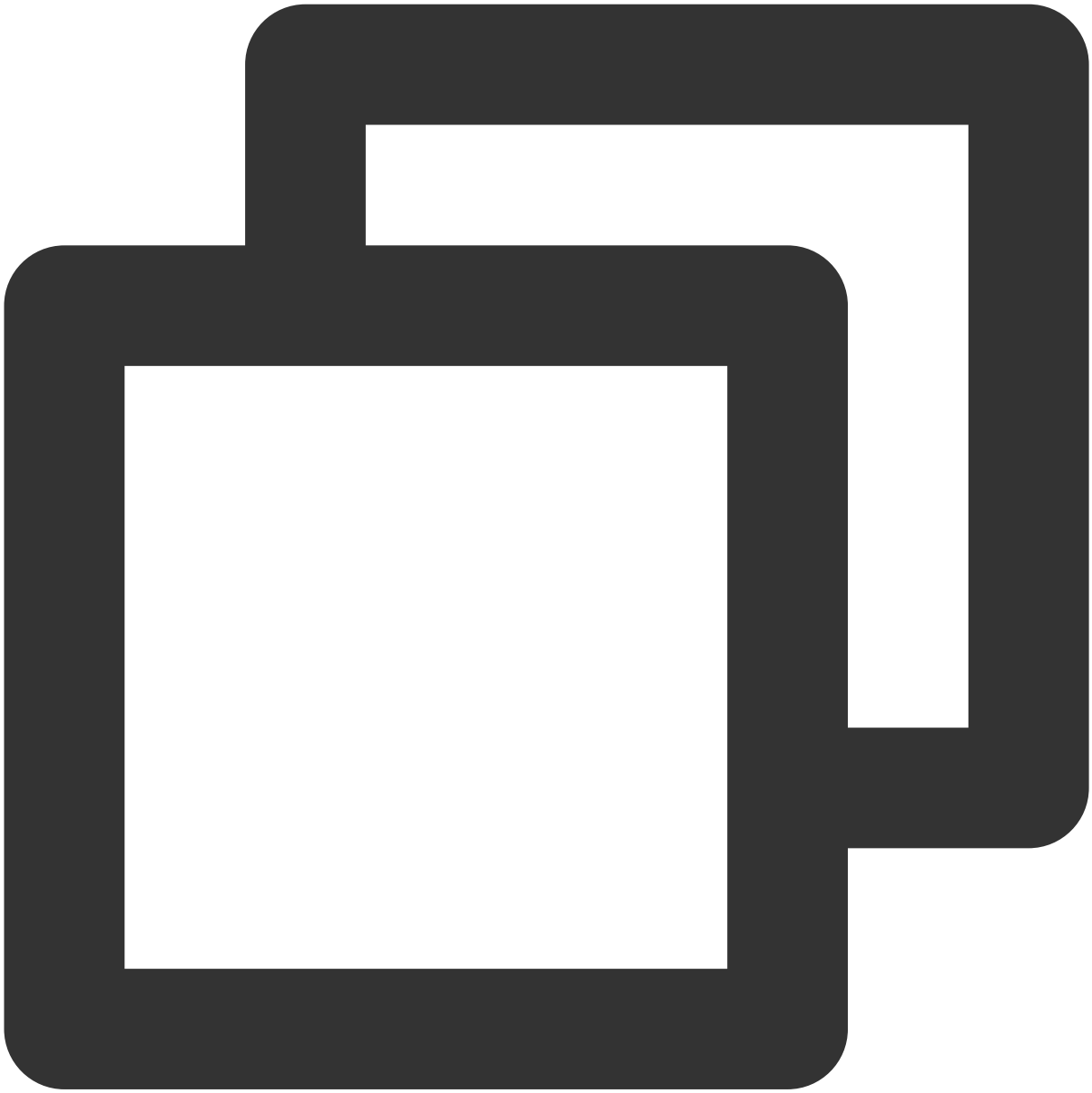
setMaxSeatCount

设置最大麦位数（仅支持进房前和创建房间时设置）。

roomType 为 TUIRoomTypeConference (教育及会议场景) 时，maxSeatCount 值不做限制；

roomType 为 TUIRoomTypeLivingRoom (直播场景) 时，maxSeatCount 最大限制为 16；

版权所有：腾讯云计算（北京）有限责任公司



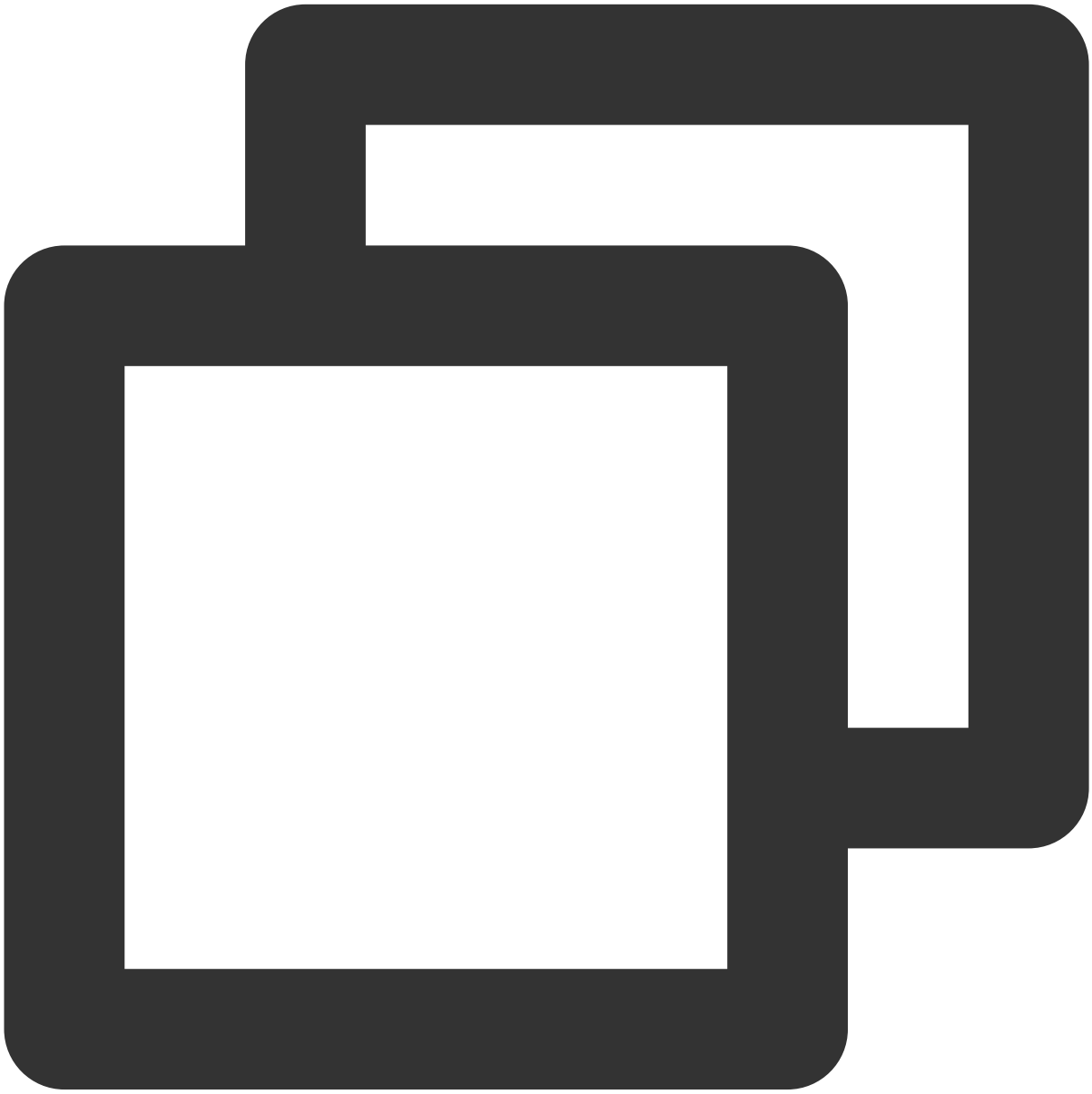
```
- (void)setMaxSeatCount:(NSUInteger)maxSeatCount
              onSuccess:(TUISuccessBlock)onSuccess
                onError:(TUIErrorBlock)onError;
```

参数	类型	含义
maxSeatCount	NSUInteger	最大麦位数
onSuccess	TUISuccessBlock	成功回调

onError	TUIErrorBlock	错误回调

getSeatList

获取麦位列表。



```
- (void)getSeatList:(TUISeatListResponseBlock)onSuccess
            onError:(TUIErrorBlock)onError;
```

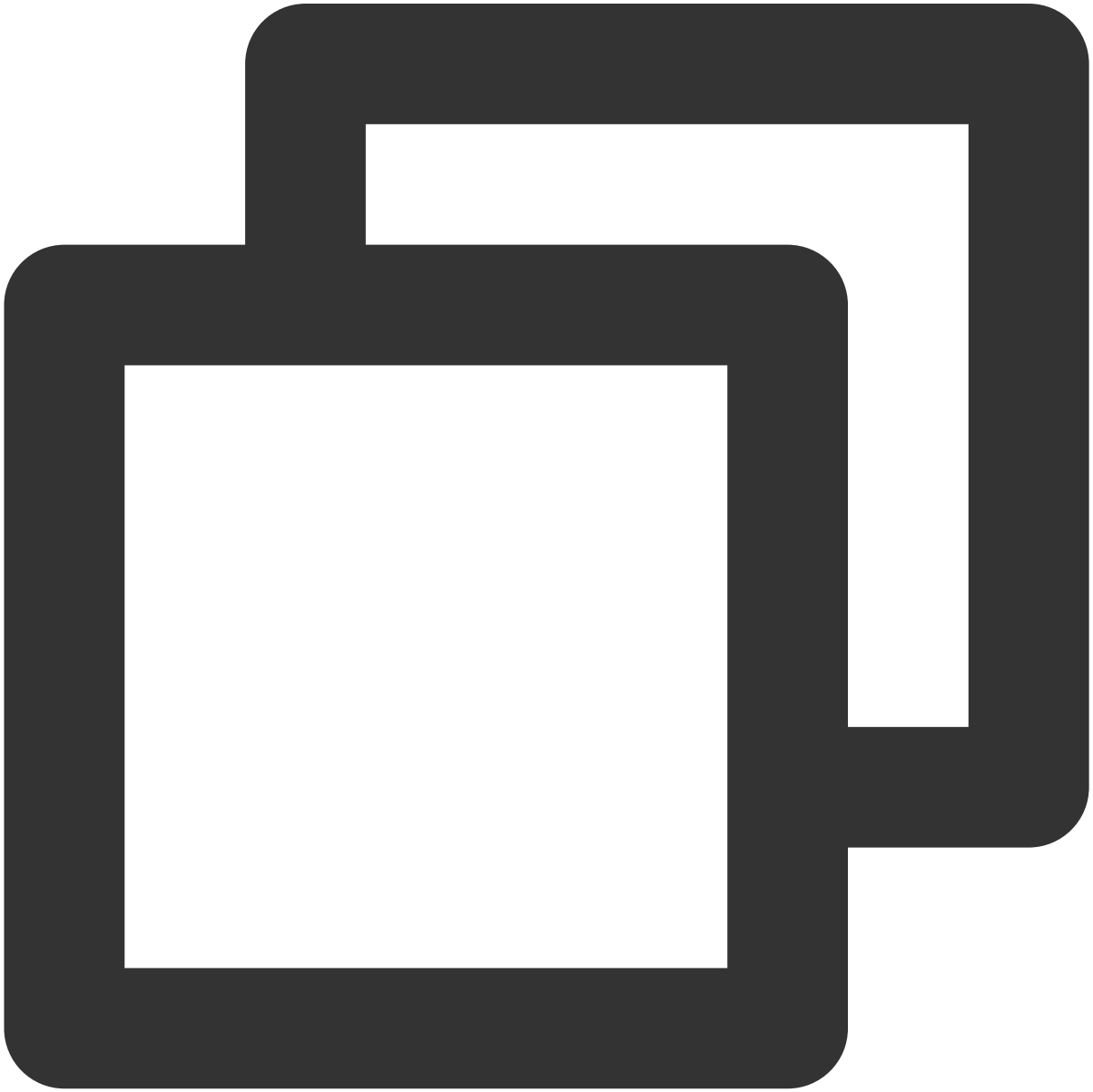
参数如下表所示：

参数	类型	含义

onSuccess	TUISeatListResponseBlock	成功回调。详细定义可以参考 TUIRoomDefine.h 中 TUISeatListResponseBlock 和 TUISeatInfo 的定义
onError	TUIErrorBlock	失败回调

lockSeatByAdmin

锁定麦位（只有管理员或群主能够调用，包括位置锁定、音频状态锁定和视频状态锁定）。



```
- (void)lockSeatByAdmin:(NSInteger)seatIndex
```

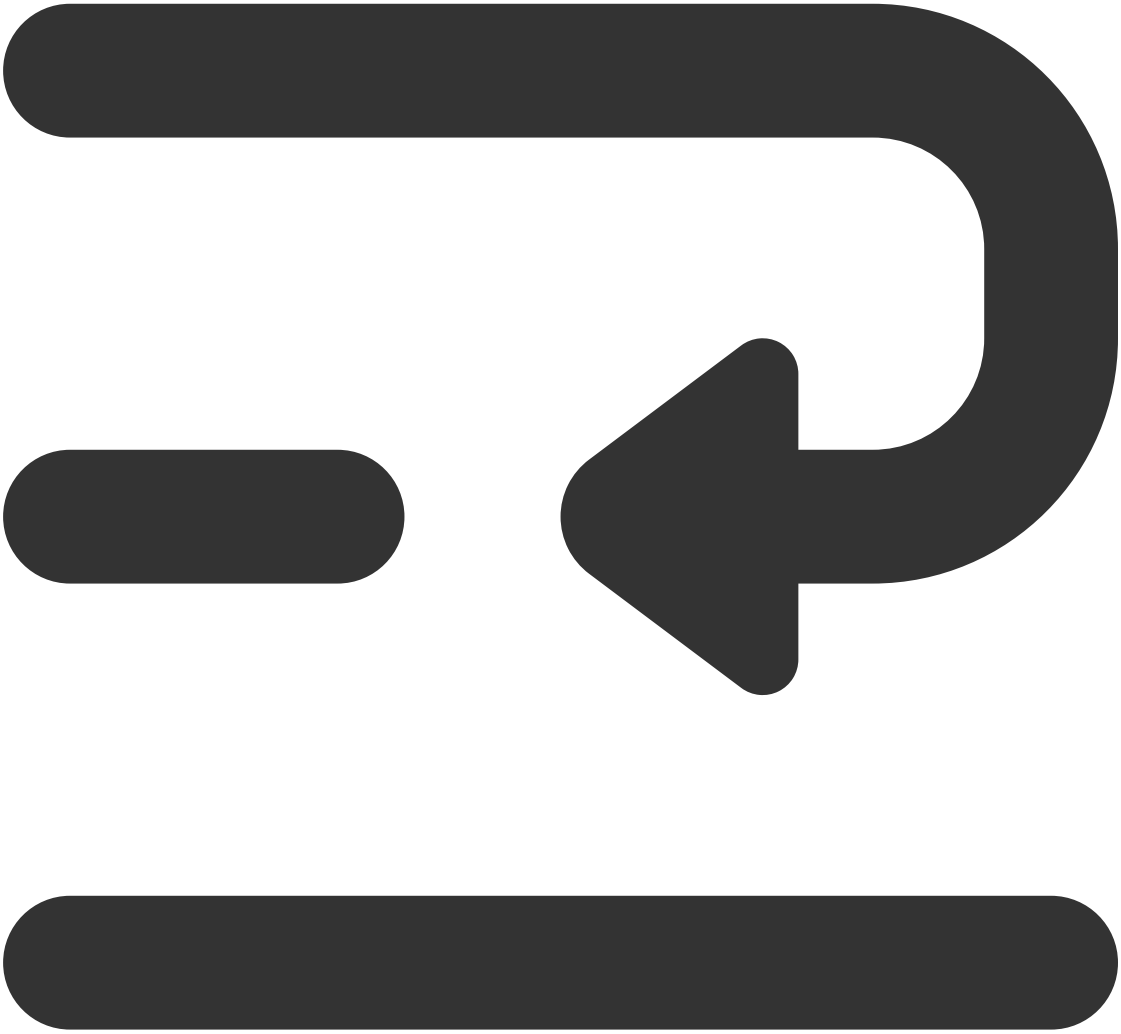
```
                lockMode:(TUISeatLockParams *)lockParams
               onSuccess:(TUISuccessBlock)onSuccess
                 onError:(TUIErrorBlock)onError;
```

参数如下表所示：

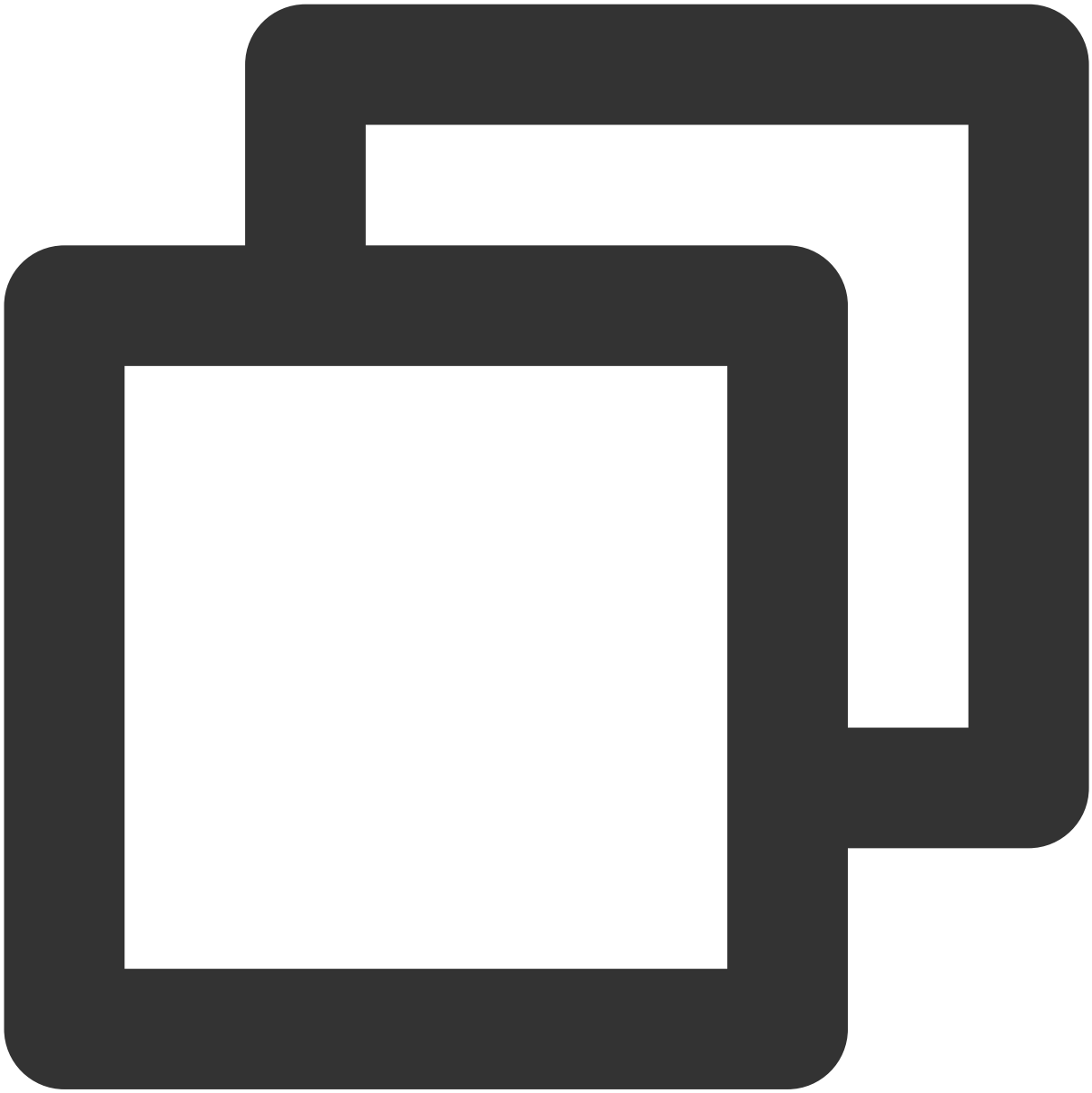
参数	类型	含义
seatIndex	NSInteger	麦位编号
lockParams	TUISeatLockParams *	锁麦参数，详细定义可以参考TUIRoomDefine.h 中TUISeatLockParams的定义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

takeSeat

本地用户上麦（自由发言模式下，不需要申请，开启麦控模式时，需要向主持人或管理员发起申请才允许上麦开启自由发言模式，直播场景可以自由上麦，上麦后开麦发言，会议场景无需调用该接口，即可开麦发言）。



版权所有：腾讯云计算（北京）有限责任公司



```
- (TUIRequest *)takeSeat:(NSInteger)seatIndex
                 timeout:(NSTimeInterval)timeout
              onAccepted:(TUIRequestAcceptedBlock)onAccepted
              onRejected:(TUIRequestRejectedBlock)onRejected
             onCancelled:(TUIRequestCancelledBlock)onCancelled
               onTimeout:(TUIRequestTimeoutBlock)onTimeout
                 onError:(TUIRequestErrorBlock)onError;
```

参数如下表所示：

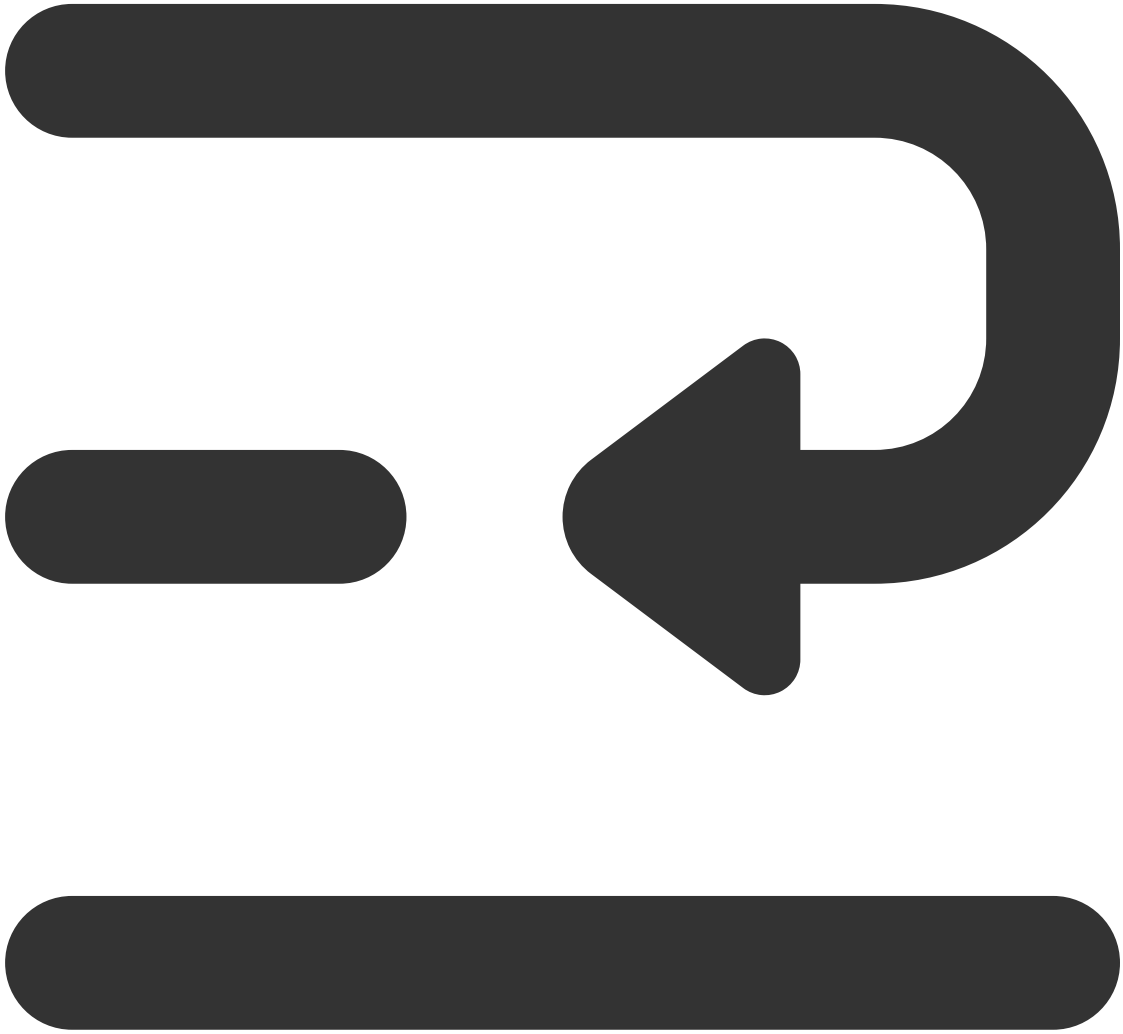
参数	类型	含义

seatIndex	NSInteger	麦位编号
timeout	NSTimeInterval	超时时间
onAccepted	TUIRequestAcceptedBlock	信令被接受回调
onRejected	TUIRequestRejectedBlock	信令被拒绝回调
onCancelled	TUIRequestCancelledBlock	信令被取消回调
onTimeout	TUIRequestTimeoutBlock	信令超时回调
onError	TUIRequestErrorBlock	错误回调

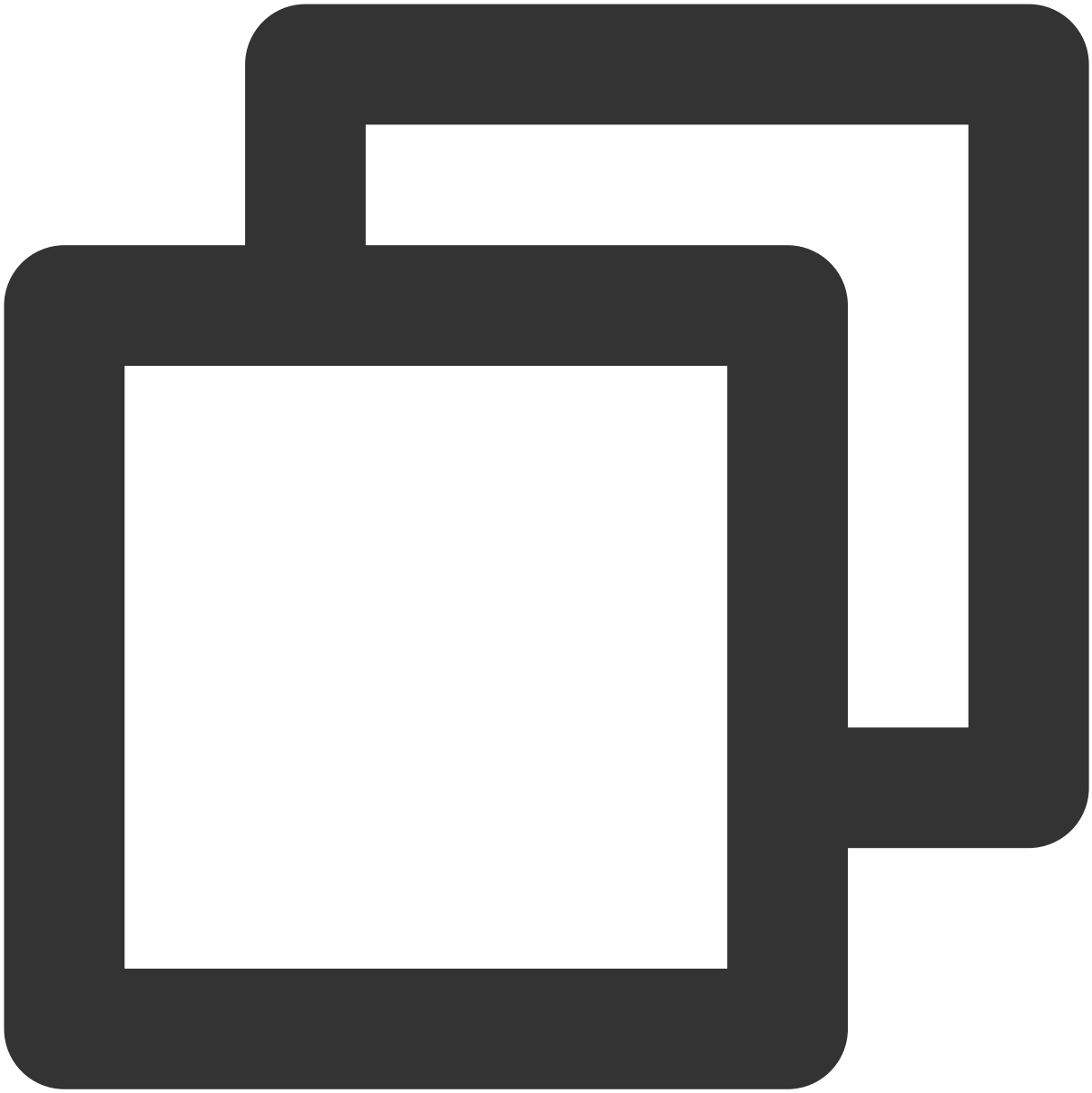
返回值	类型	含义
request	TUIRequest	请求体

leaveSeat

本地用户下麦（自由发言模式下，不需要申请）。



版权所有：腾讯云计算（北京）有限责任公司



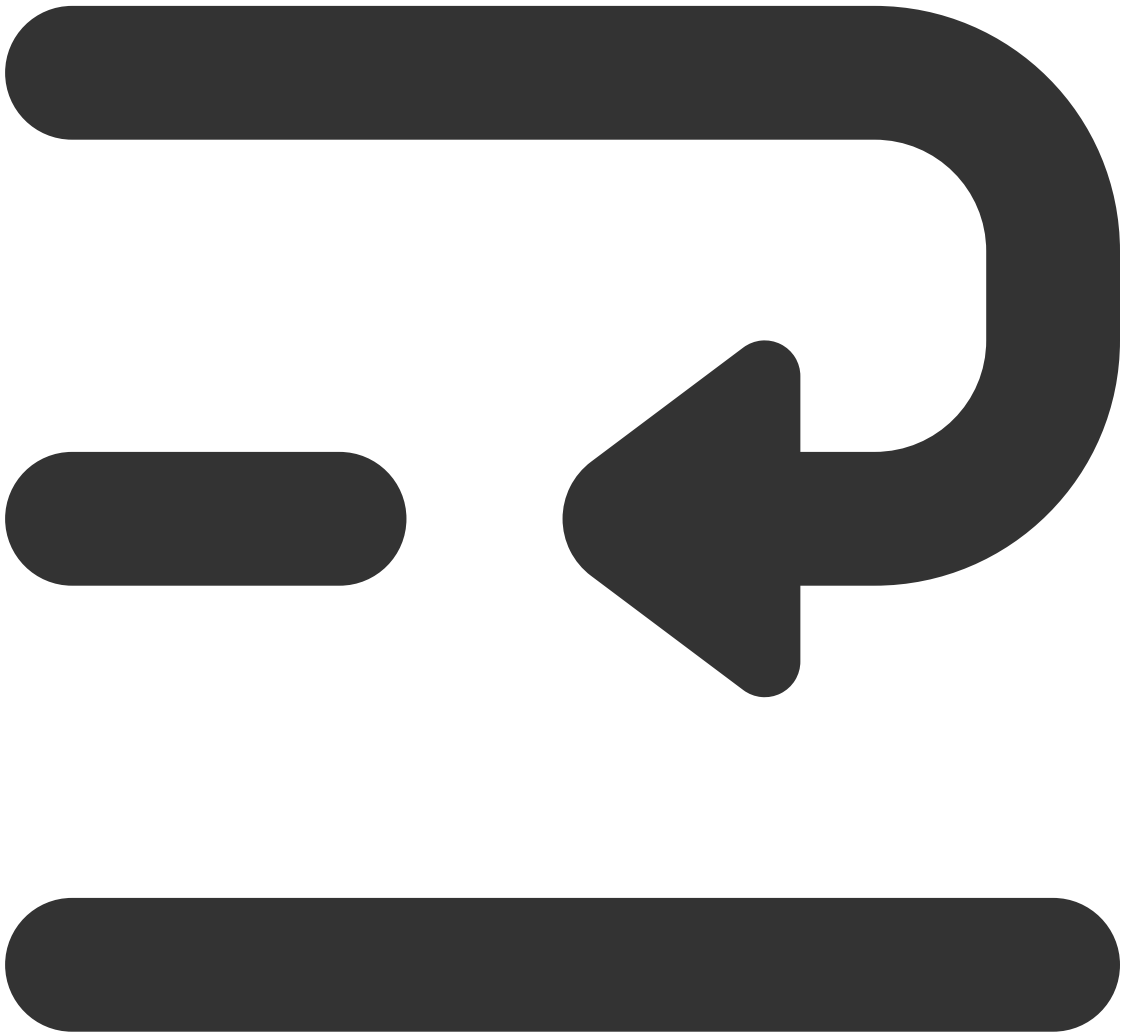
```
- (void)leaveSeat:(TUISuccessBlock)onSuccess
          onError:(TUIErrorBlock)onError;
```

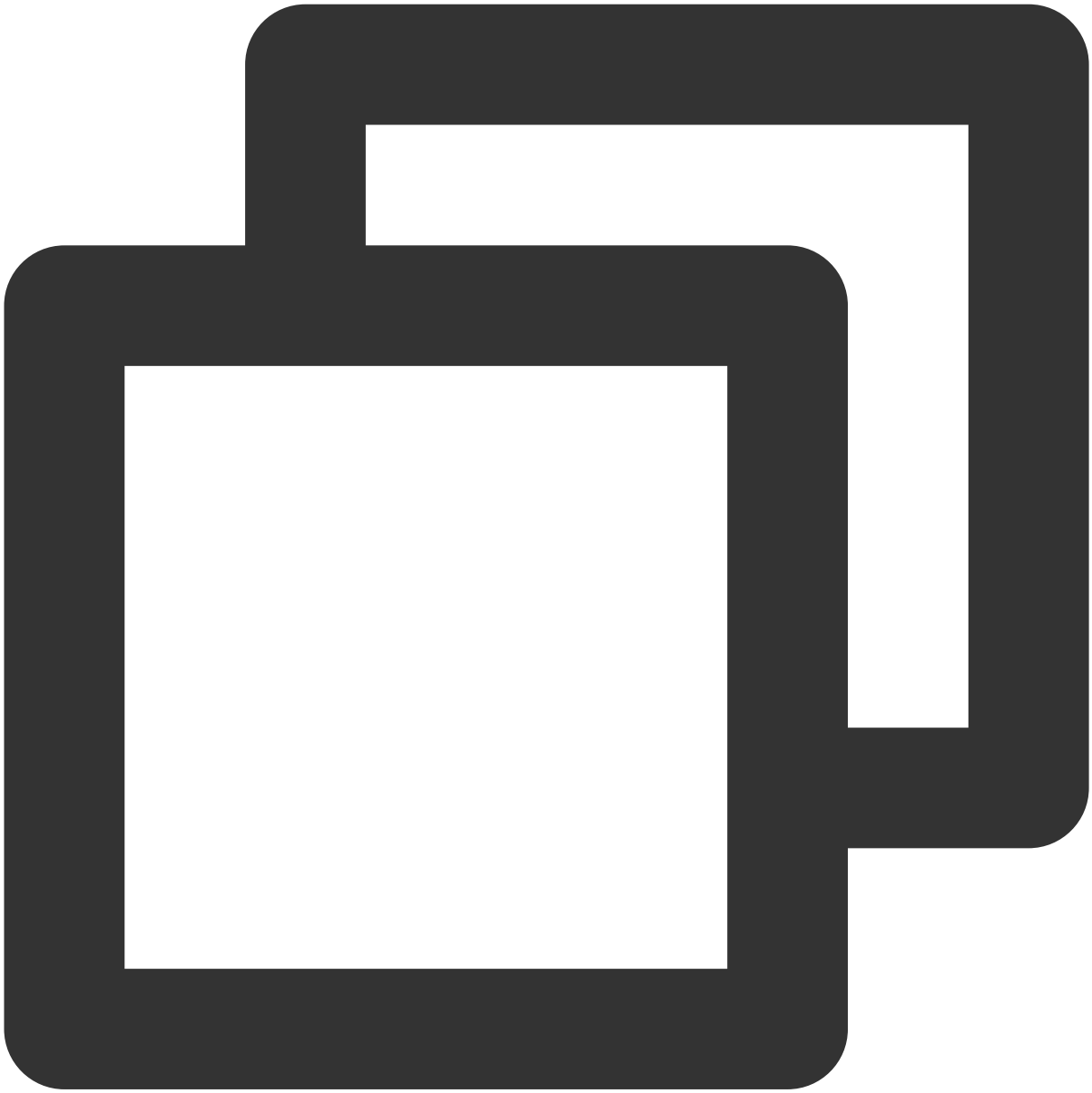
参数如下表所示：

参数	类型	含义
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

takeUserOnSeatByAdmin

主持人/管理员 邀请用户上麦。





```
- (TUIRequest *)takeUserOnSeatByAdmin:(NSInteger)seatIndex
                               userId:(NSString *)userId
                              timeout:(NSTimeInterval)timeout
                           onAccepted:(TUIRequestAcceptedBlock)onAccepted
                           onRejected:(TUIRequestRejectedBlock)onRejected
                          onCancelled:(TUIRequestCancelledBlock)onCancelled
                            onTimeout:(TUIRequestTimeoutBlock)onTimeout
                              onError:(TUIRequestErrorBlock)onError;
```

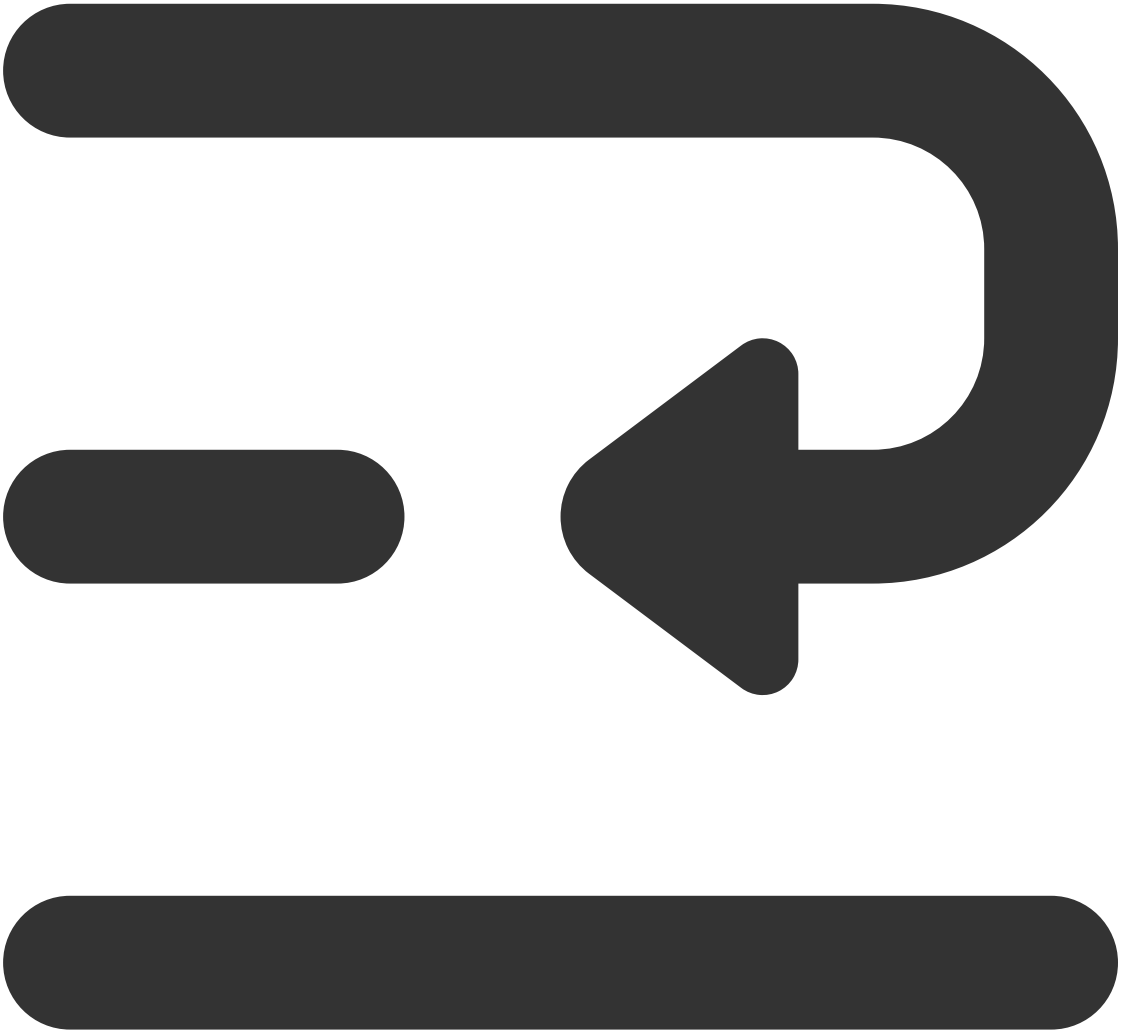
参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	麦位编号
userId	NSString *	用户 ID
timeout	NSTimeInterval	超时时间（单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调）
onAccepted	TUIRequestAcceptedBlock	邀请被接受的回调
onRejected	TUIRequestRejectedBlock	邀请被拒绝的回调
onCancelled	TUIRequestCancelledBlock	邀请被取消的回调
onTimeout	TUIRequestTimeoutBlock	邀请超时未处理的回调
onError	TUIRequestErrorBlock	邀请发生错误的回调

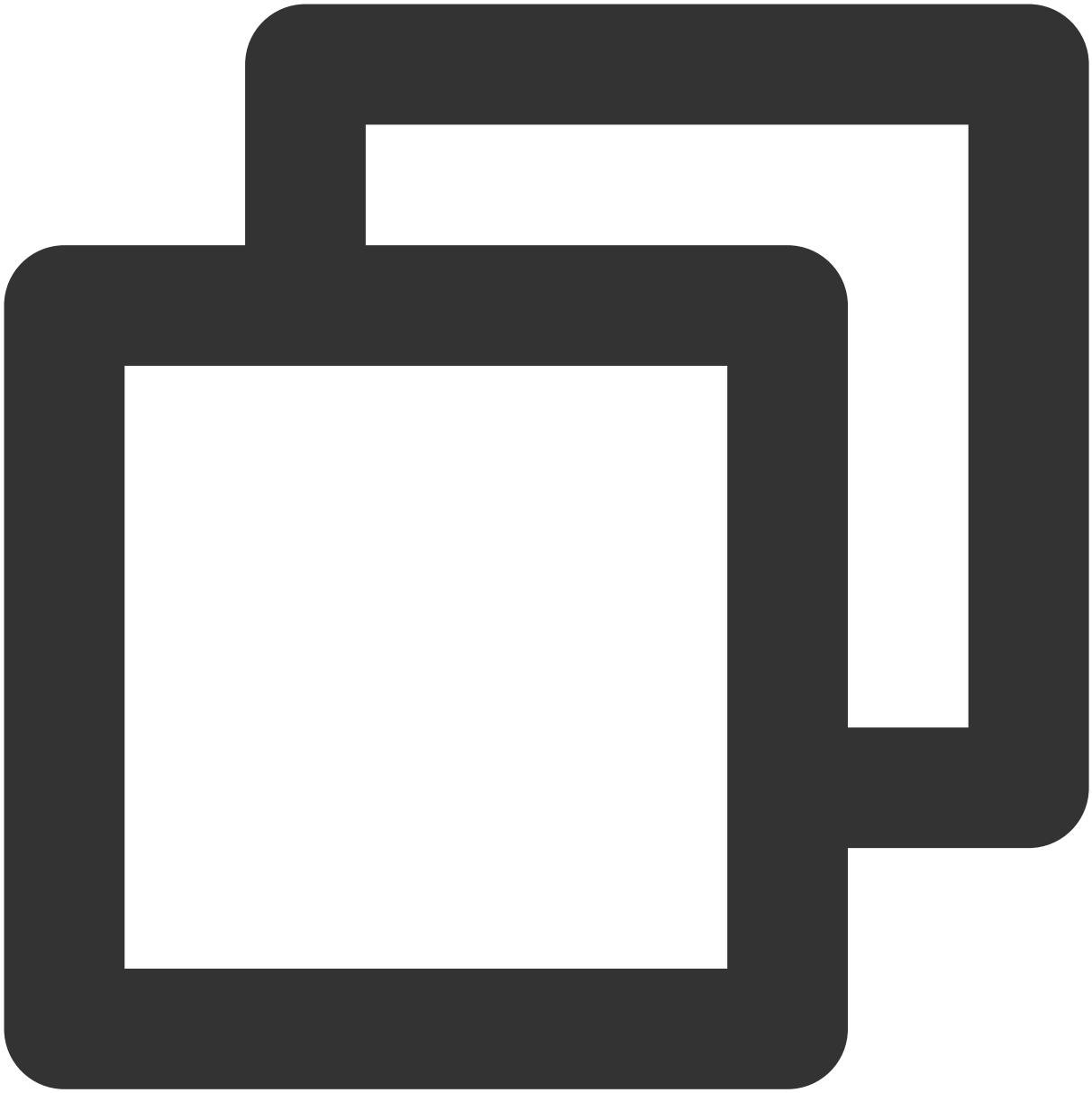
返回值	类型	含义
request	TUIRequest	请求体

kickUserOffSeatByAdmin

主持人/管理员 将用户踢下麦。



版权所有：腾讯云计算（北京）有限责任公司



```
- (void)kickUserOffSeatByAdmin:(NSInteger)seatIndex
                        userId:(NSString *)userId
                     onSuccess:(TUISuccessBlock)onSuccess
                       onError:(TUIErrorBlock)onError;
```

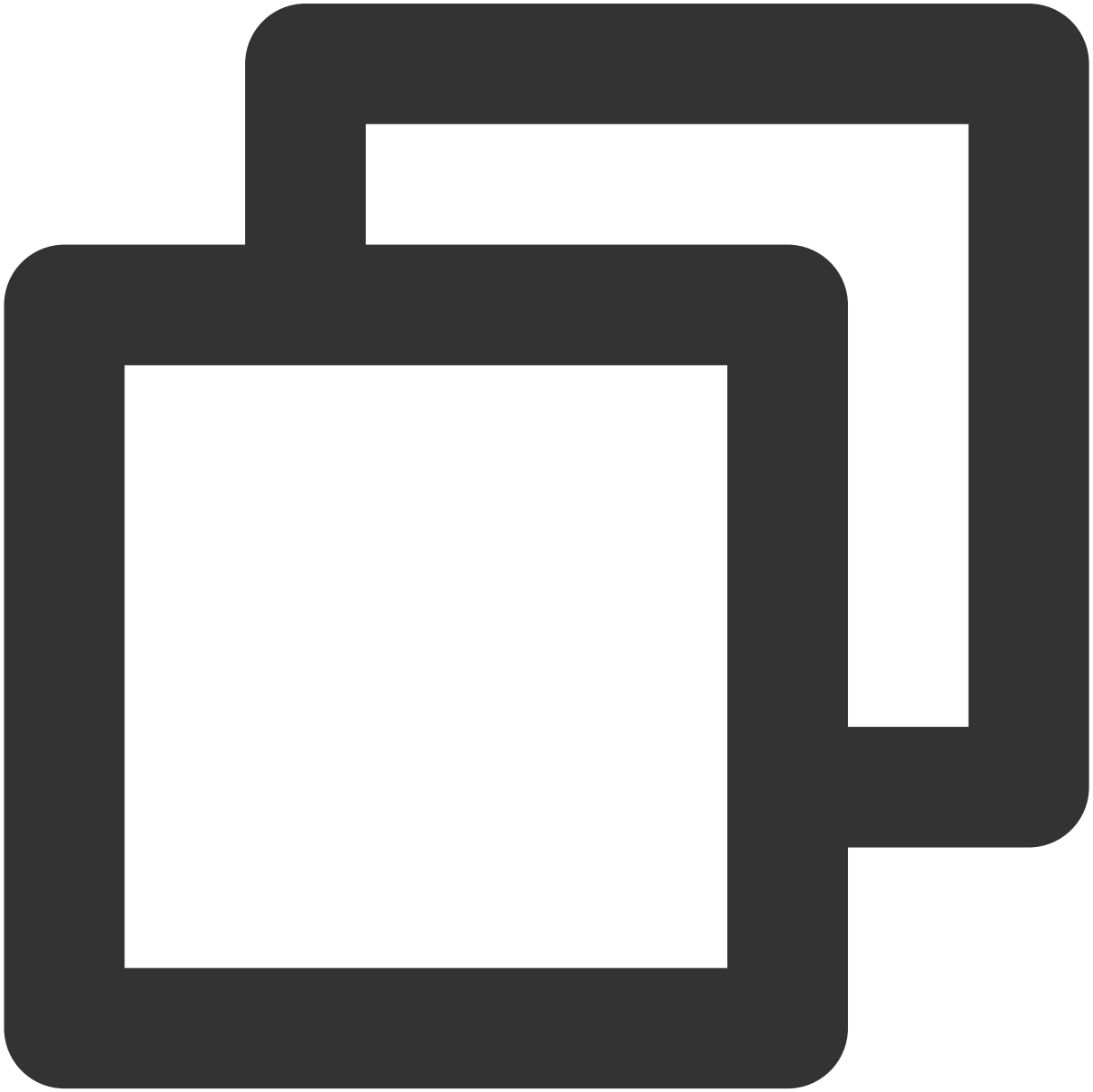
参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	麦位编号

userId	NSString *	用户 ID
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	错误回调

cancelRequest

取消请求。



```
- (void)cancelRequest:(NSInteger)requestId
```

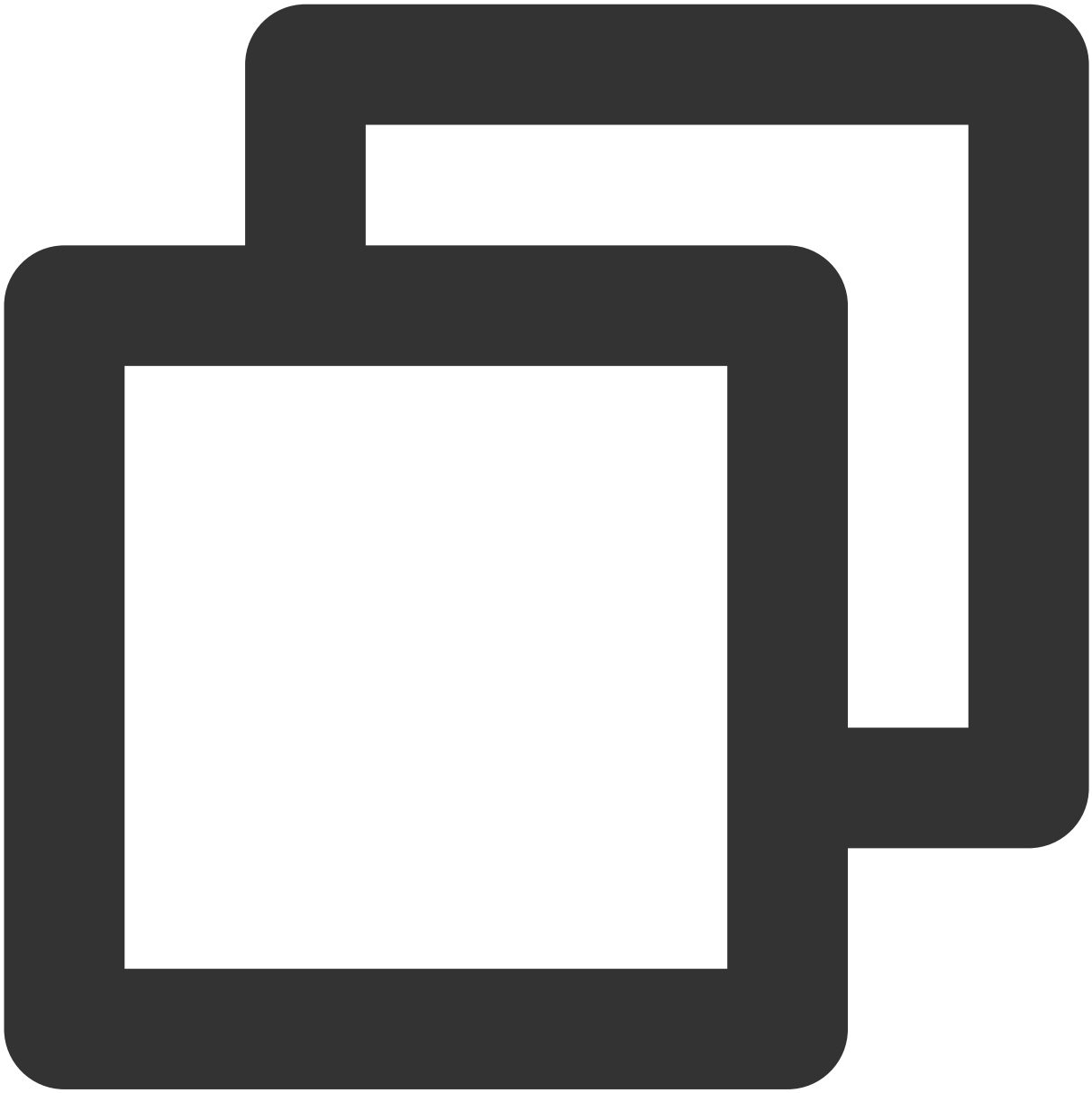
```
                onSuccess:(TUISuccessBlock)onSuccess
                  onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
requestId	NSString *	请求 ID。发送请求的接口返回
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

responseRemoteRequest

回复请求。



```
- (void)responseRemoteRequest:(NSString *)requestId
                        agree:(BOOL)agree
                    onSuccess:(TUISuccessBlock)onSuccess
                      onError:(TUIErrorBlock)onError;
```

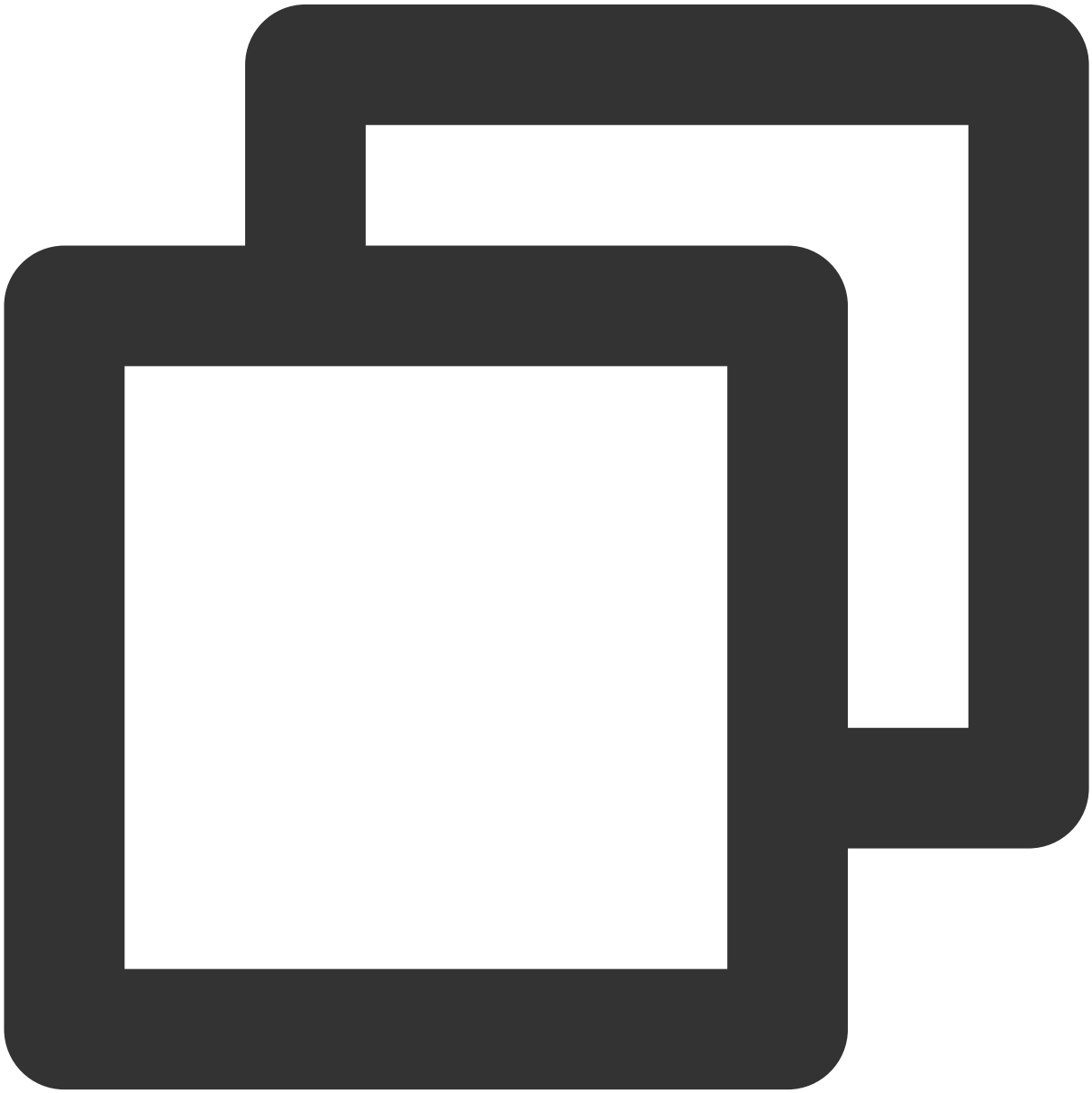
参数如下表所示：

参数	类型	含义
requestId	NSString *	请求ID。发送请求的接口返回或者 OnRequestReceived 事件通知

agree	BOOL	是否同意。YES：同意，NO：拒绝
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	失败回调

sendTextMessage

发送本文消息。



```
- (void)sendTextMessage:(NSString *)message
```

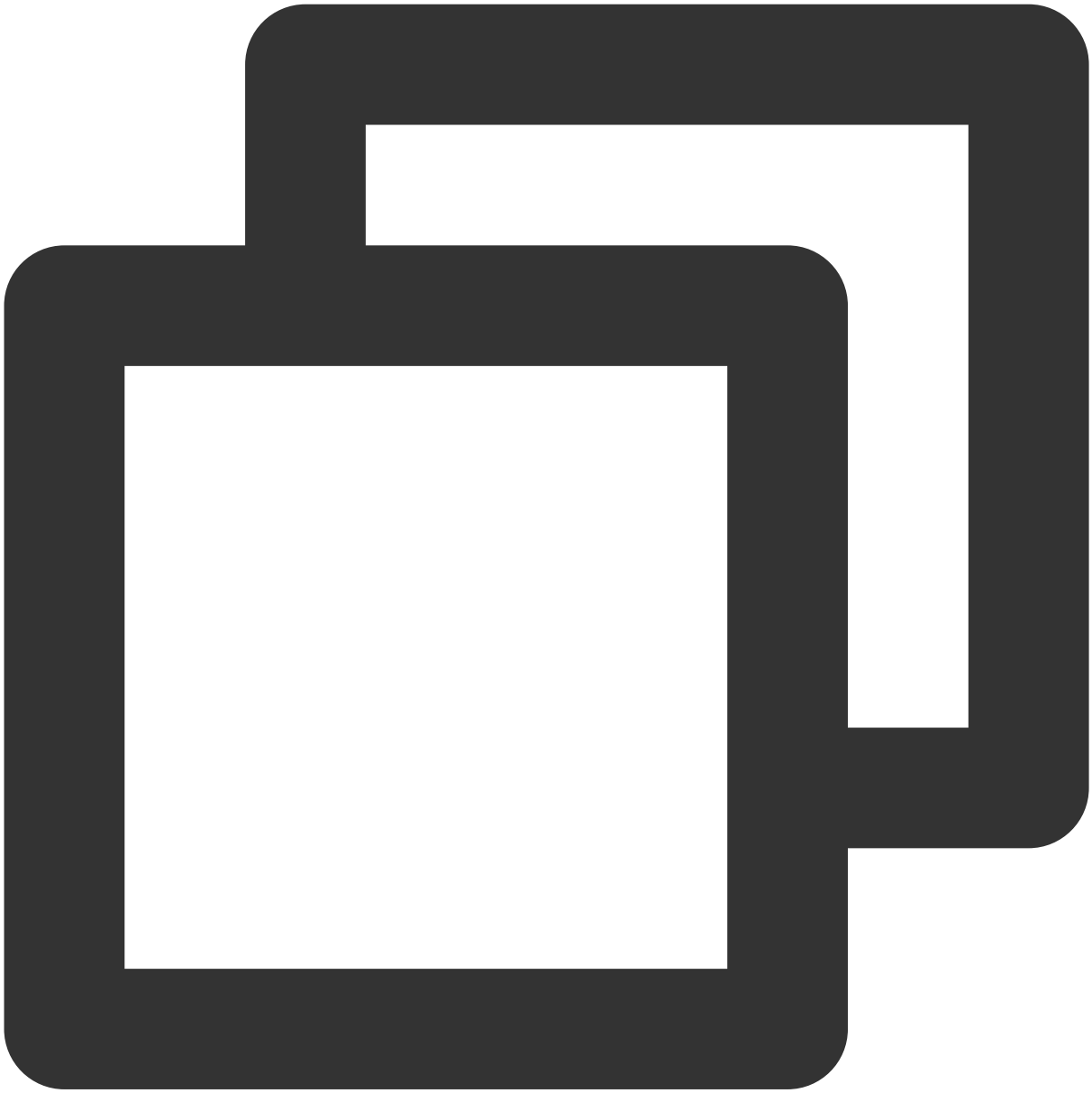
```
                onSuccess:(TUISuccessBlock)onSuccess
                  onError:(TUIErrorBlock)onError;
```

参数如下表所示：

参数	类型	含义
message	NSString *	消息内容
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	错误回调

sendCustomMessage

发送自定义消息。



```
- (void)sendCustomMessage:(NSString *)message
                onSuccess:(TUISuccessBlock)onSuccess
                  onError:(TUIErrorBlock)onError;
```

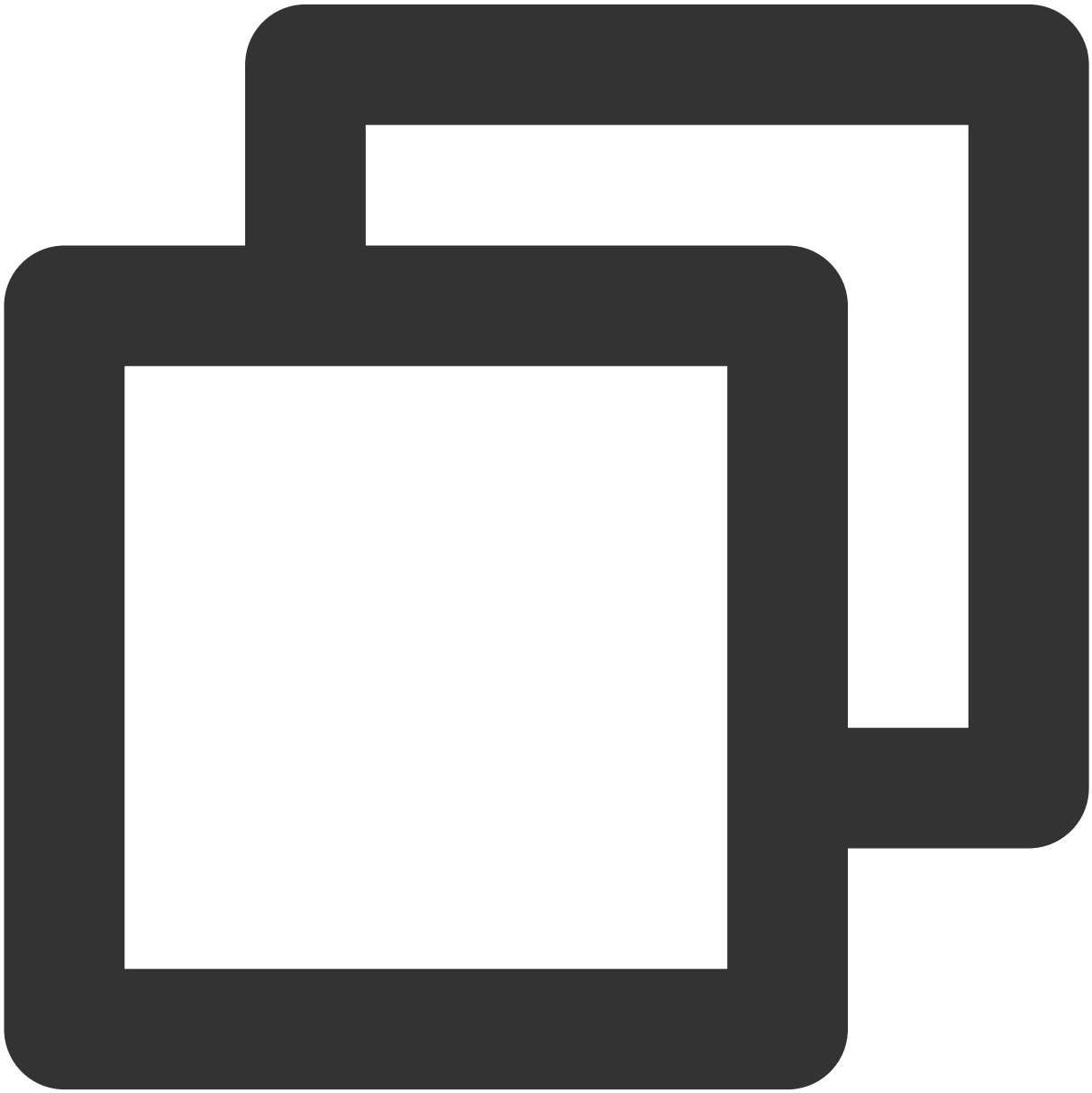
参数如下表所示：

参数	类型	含义
message	NSString *	消息内容
onSuccess	TUISuccessBlock	成功回调

onError	TUIErrorBlock	错误回调

disableSendingMessageByAdmin

禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）。

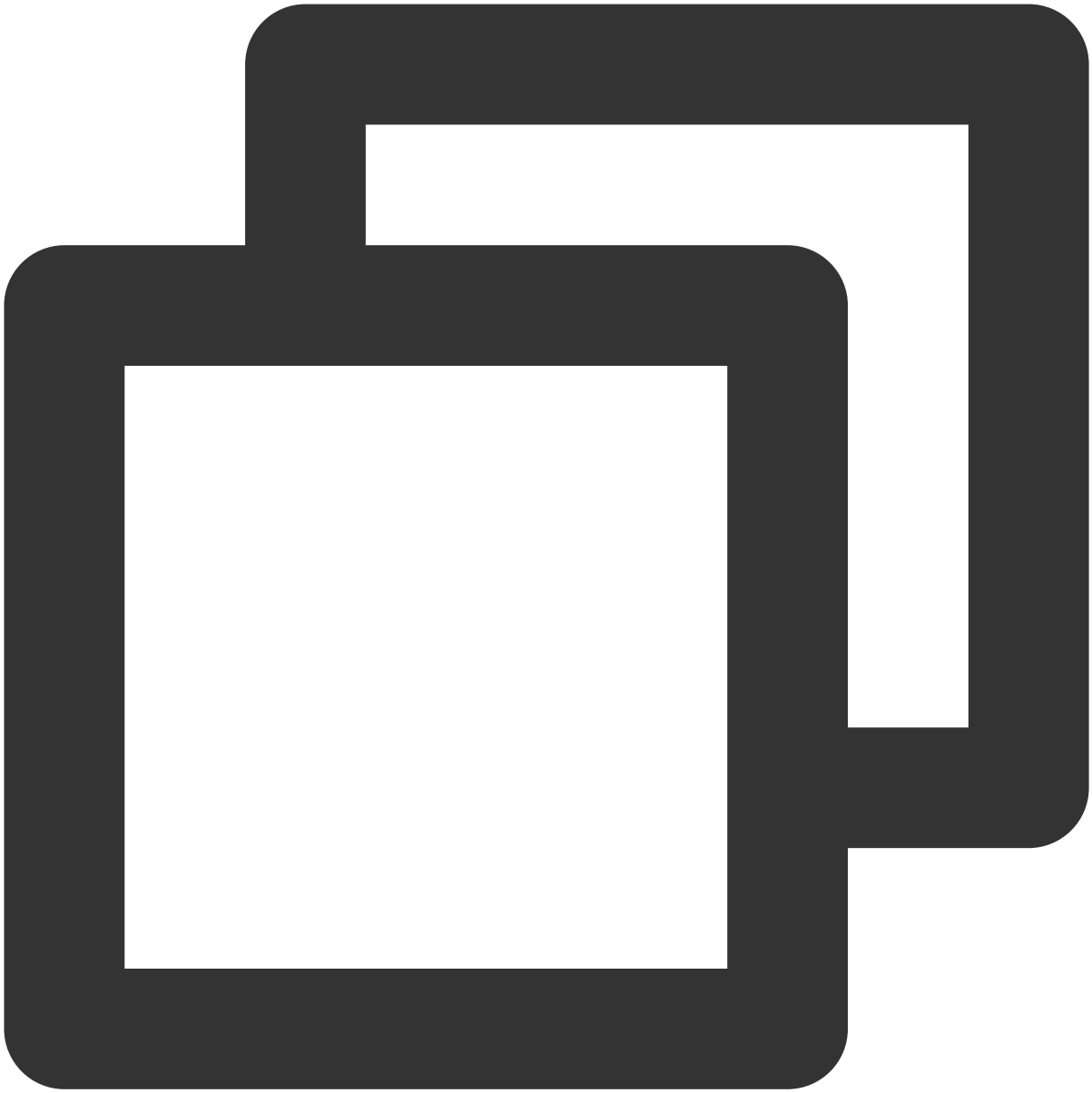


```
- (void)disableSendingMessageByAdmin:(NSString *)userId
                          isDisable:(BOOL)isDisable
                          onSuccess:(TUISuccessBlock)onSuccess
                            onError:(TUIErrorBlock)onError;
```


参数	类型	含义
userId	NSString *	用户 ID
isDisable	BOOL	是否禁用
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	错误回调

disableSendingMessageForAllUser

禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）。

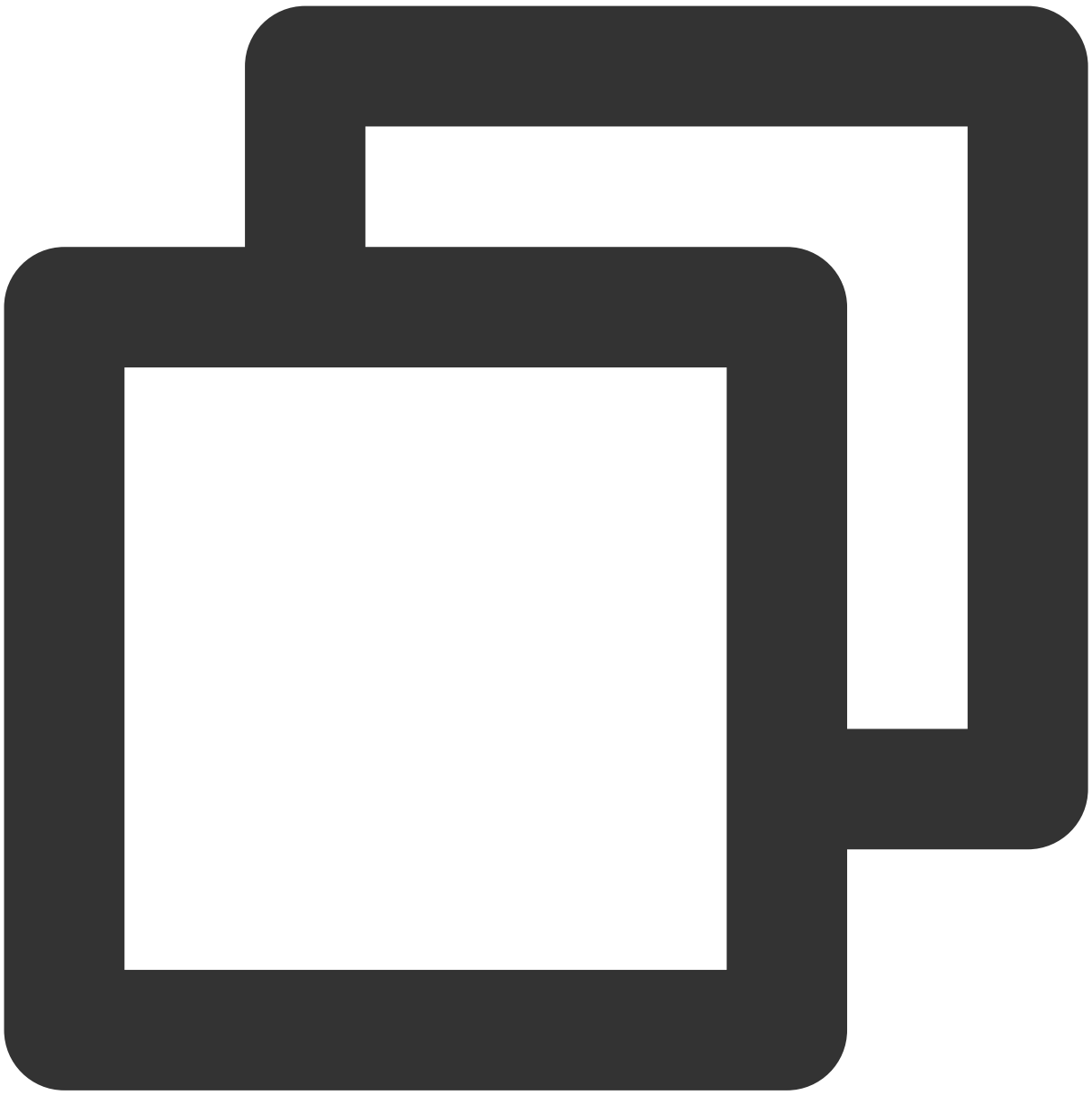


```
- (void)disableSendingMessageForAllUser:(BOOL)isDisable
                              onSuccess:(TUISuccessBlock)onSuccess
                                onError:(TUIErrorBlock)onError;
```

参数	类型	含义
isDisable	BOOL	是否禁用
onSuccess	TUISuccessBlock	成功回调
onError	TUIErrorBlock	错误回调

getDeviceManager

获得设备管理对象。

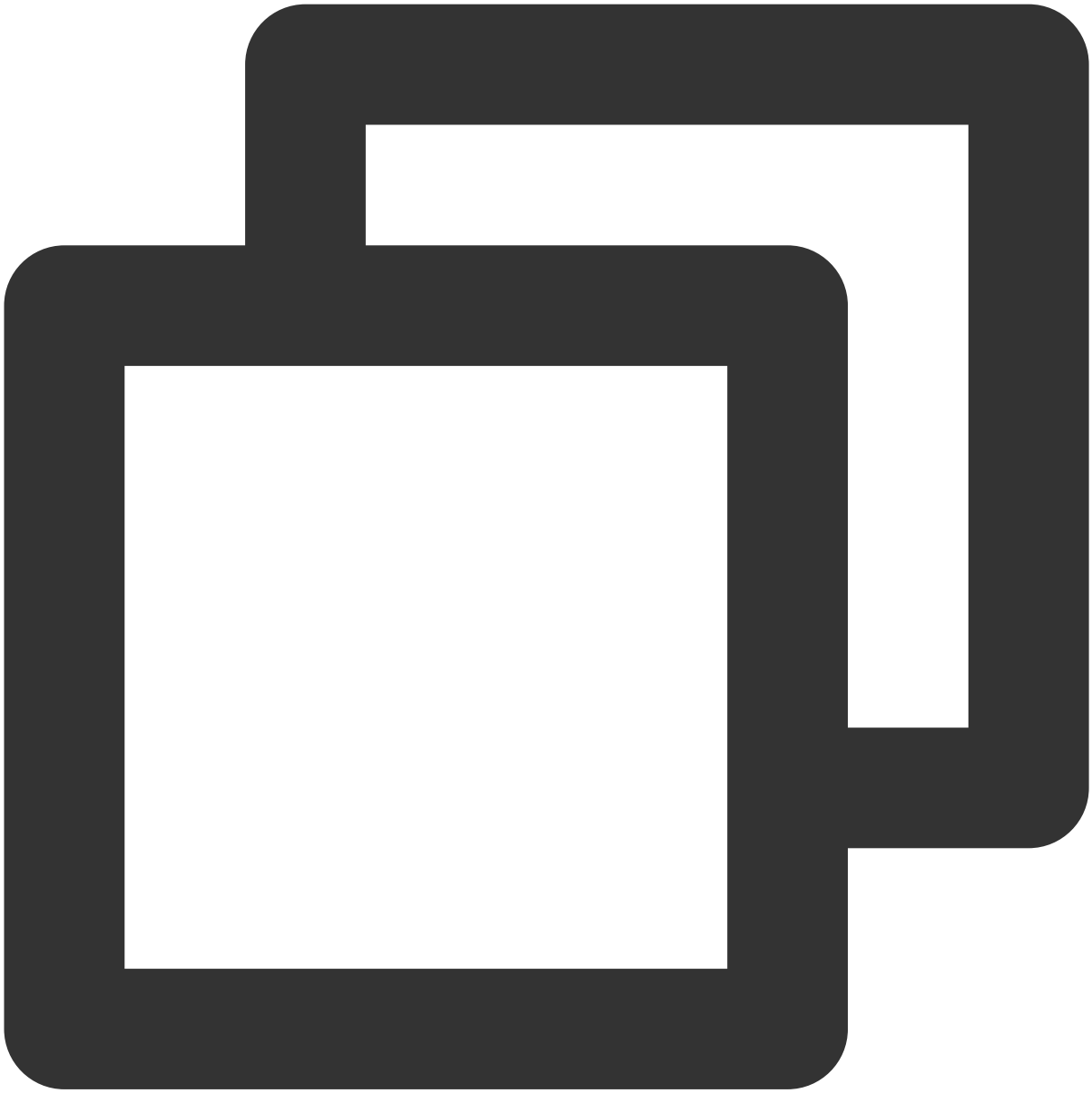


```
- (TXDeviceManager *)getDeviceManager;
```

返回值	类型	含义
manager	TXDeviceManager *	TXDeviceManager 对象

getBeautyManager

获得音效管理对象。

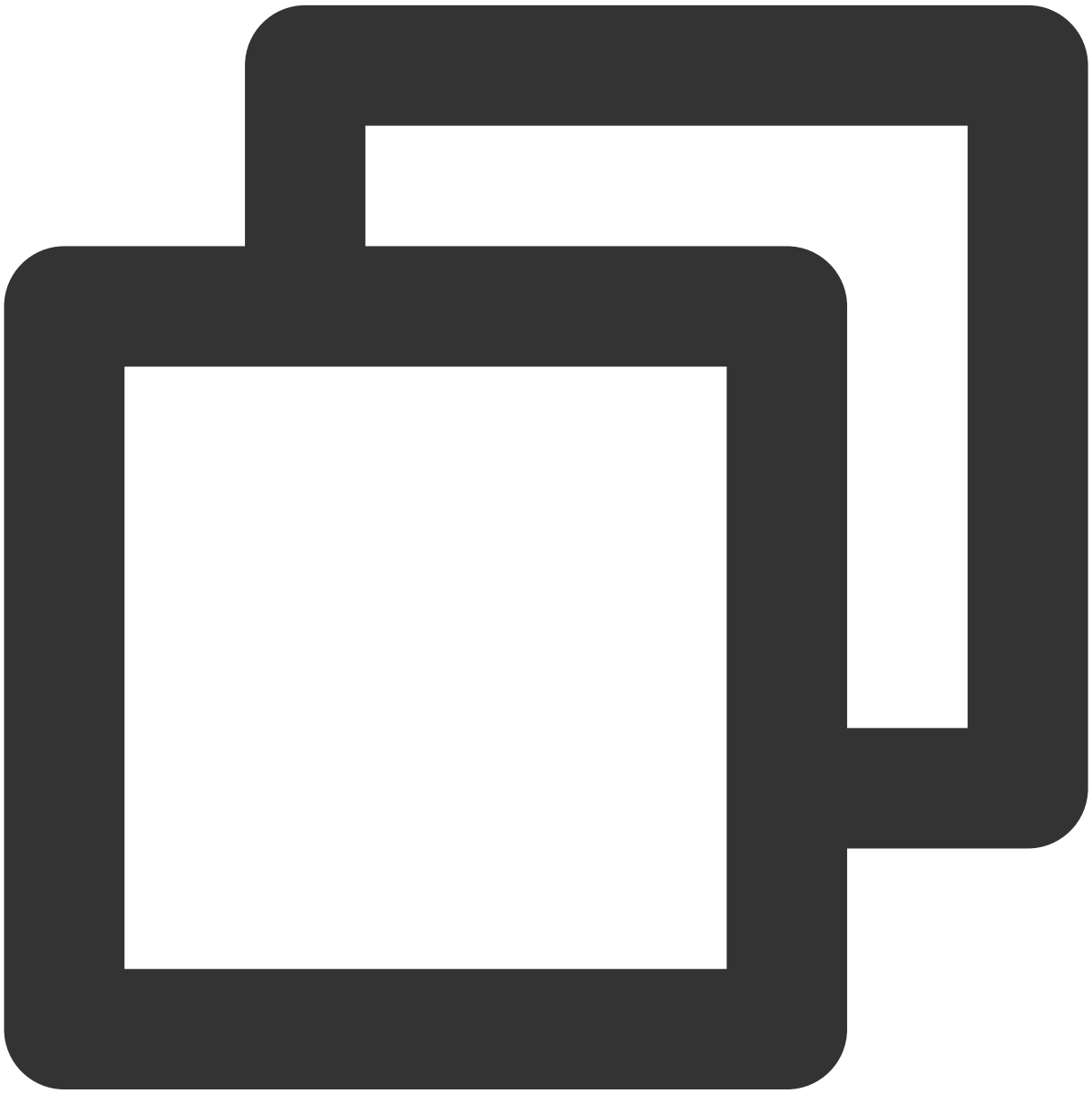


```
- (TXBeautyManager *)getBeautyManager;
```

返回值	类型	含义
manager	TXBeautyManager *	TXBeautyManager 对象

getAudioEffectManager

获得音效管理对象。

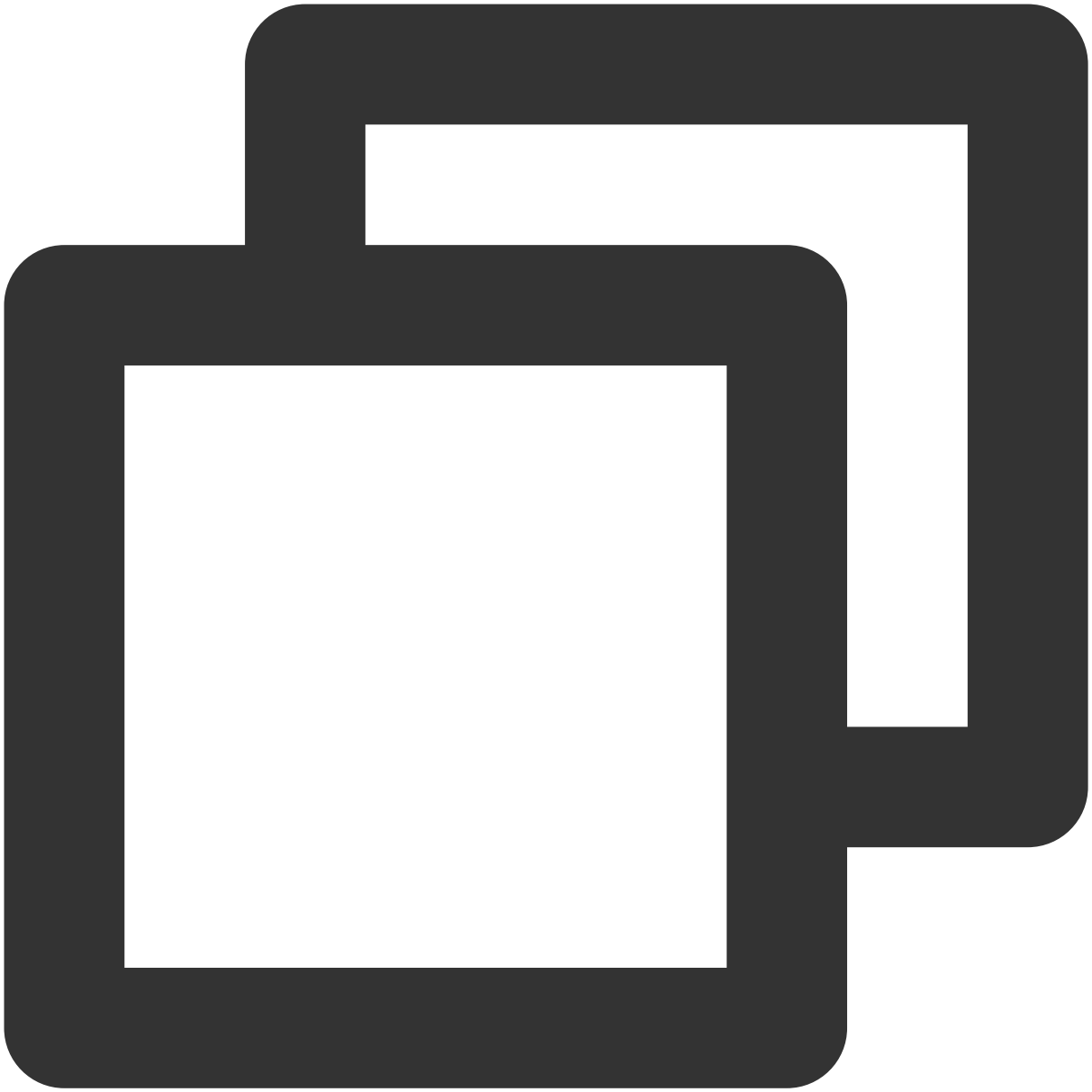


```
- (TXAudioEffectManager *)getAudioEffectManager;
```

返回值	类型	含义
manager	TXAudioEffectManager *	TXAudioEffectManager 对象

getTRTCCloud

获得 TRTC 实例对象。



```
- (TRTCCloud *)getTRTCCloud;
```

返回值	类型	含义
manager	TRTCCloud *	TRTCCloud 对象

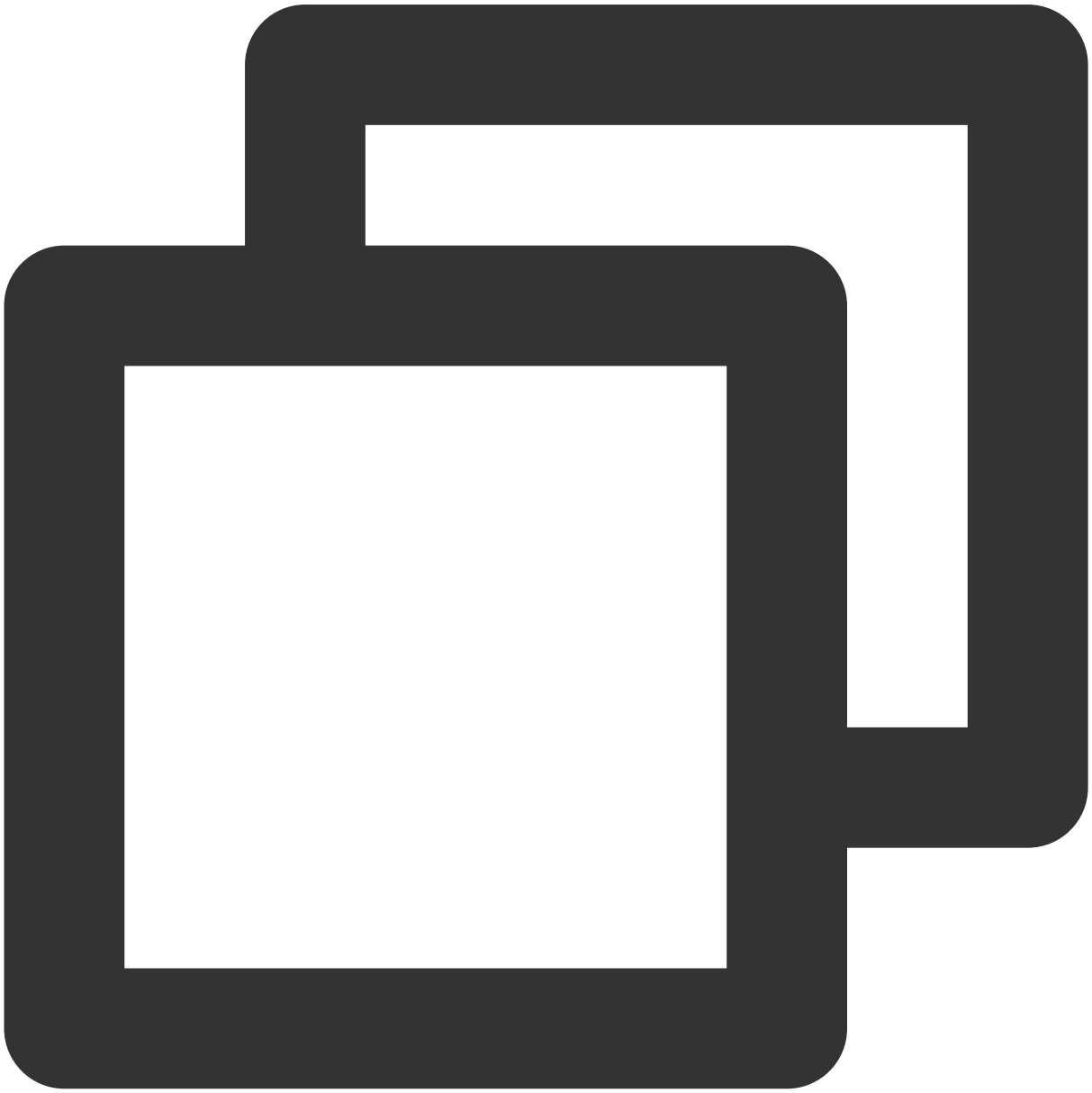
TUIRoomObserver

最近更新时间：2023-10-19 17:35:51

TUIRoomObserver 类是 TUIRoomEngine 对应的回调事件类，您可以通过此回调接口，监听自己感兴趣的回调事件。

onError

错误事件回调。



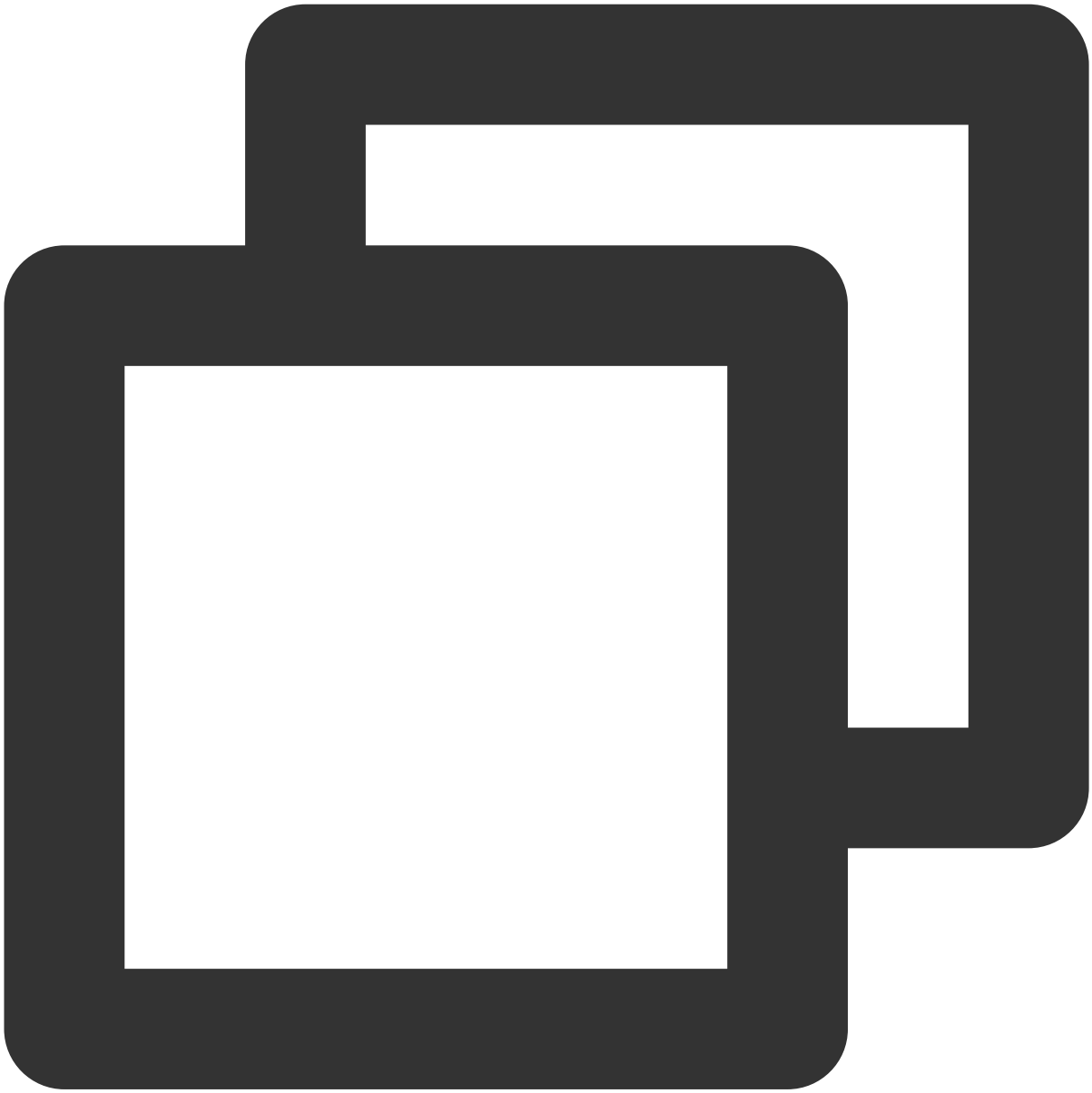
```
- (void)onError:(TUIError)error message:(NSString *)message;
```

参数如下表所示：

参数	类型	含义
error	TUIError	错误码
message	NSString *	错误信息

onKickedOffLine

其他终端登录被踢下线事件。



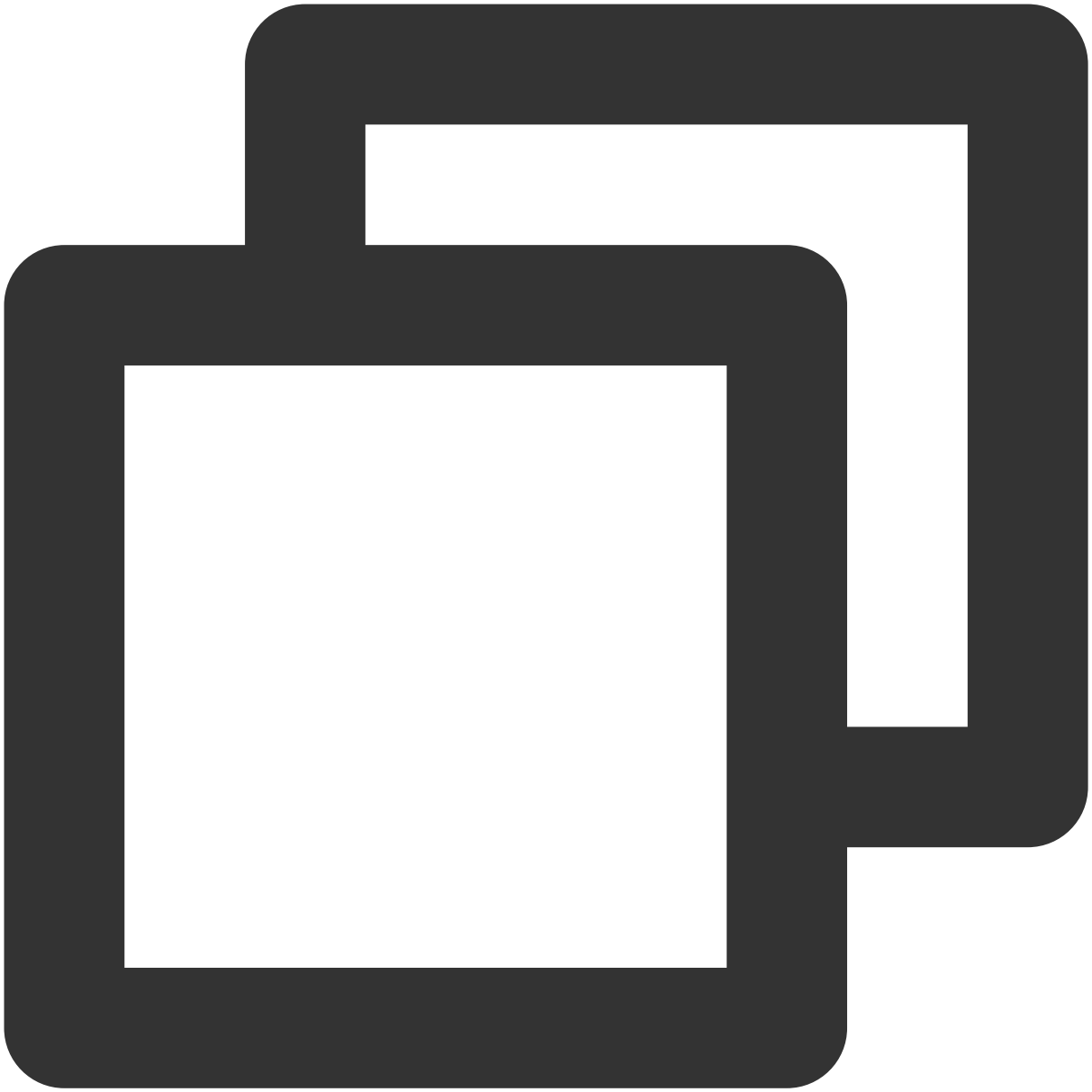
```
- (void)onKickedOffLine:(NSString *)message;
```

参数如下表所示：

参数	类型	含义
message	NSString *	被踢出的描述

onUserSigExpired

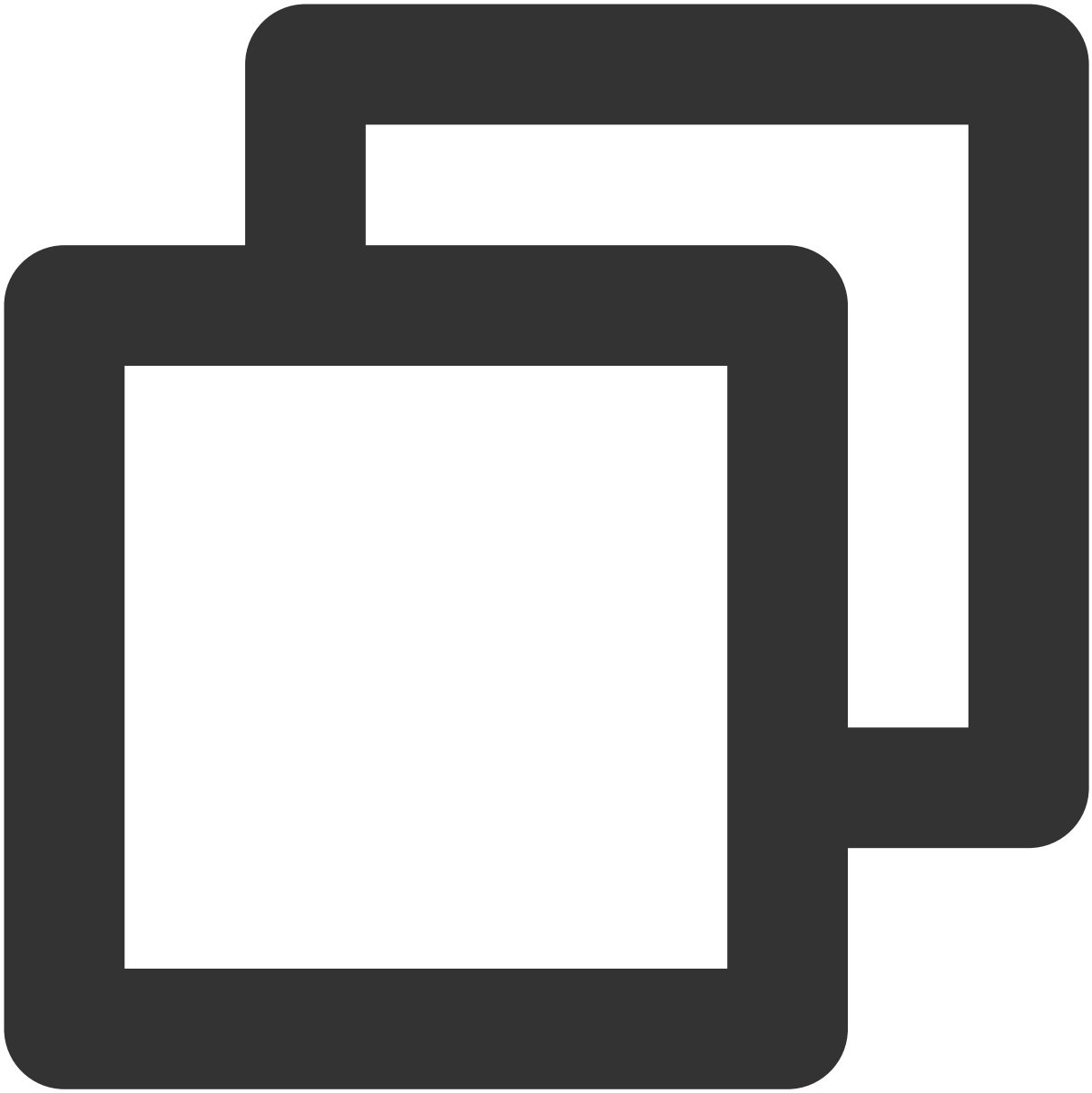
用户凭证超时事件。



```
- (void)onUserSigExpired;
```

onRoomNameChanged

房间名称更改事件。

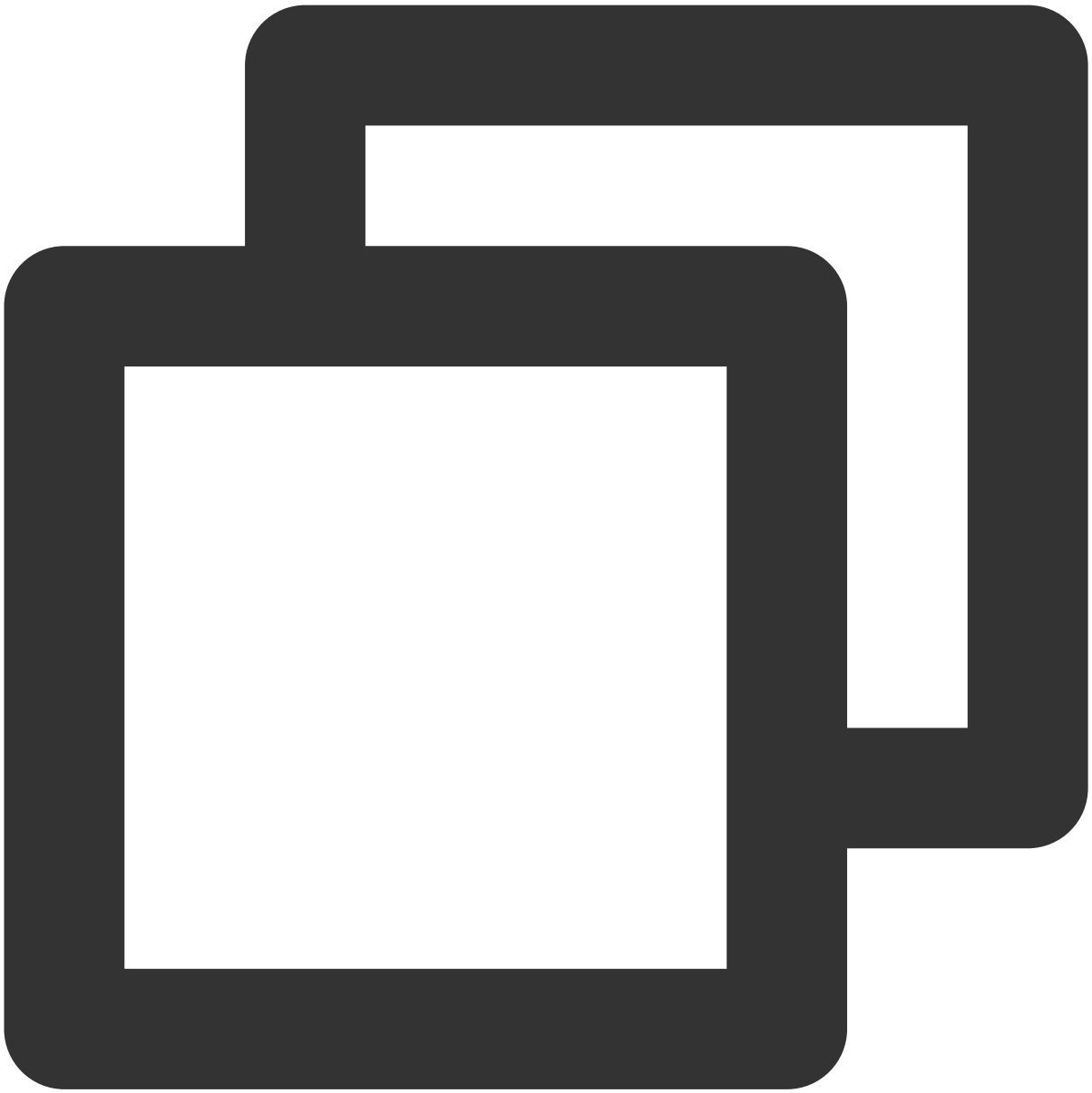


```
- (void)onRoomNameChanged:(NSString *)roomId roomName:(NSString *)roomName;
```

参数	类型	含义
roomId	NSString *	房间 ID
roomName	NSString *	房间名称

onAllUserMicrophoneDisableChanged

房间内所有用户麦克风被禁用事件。

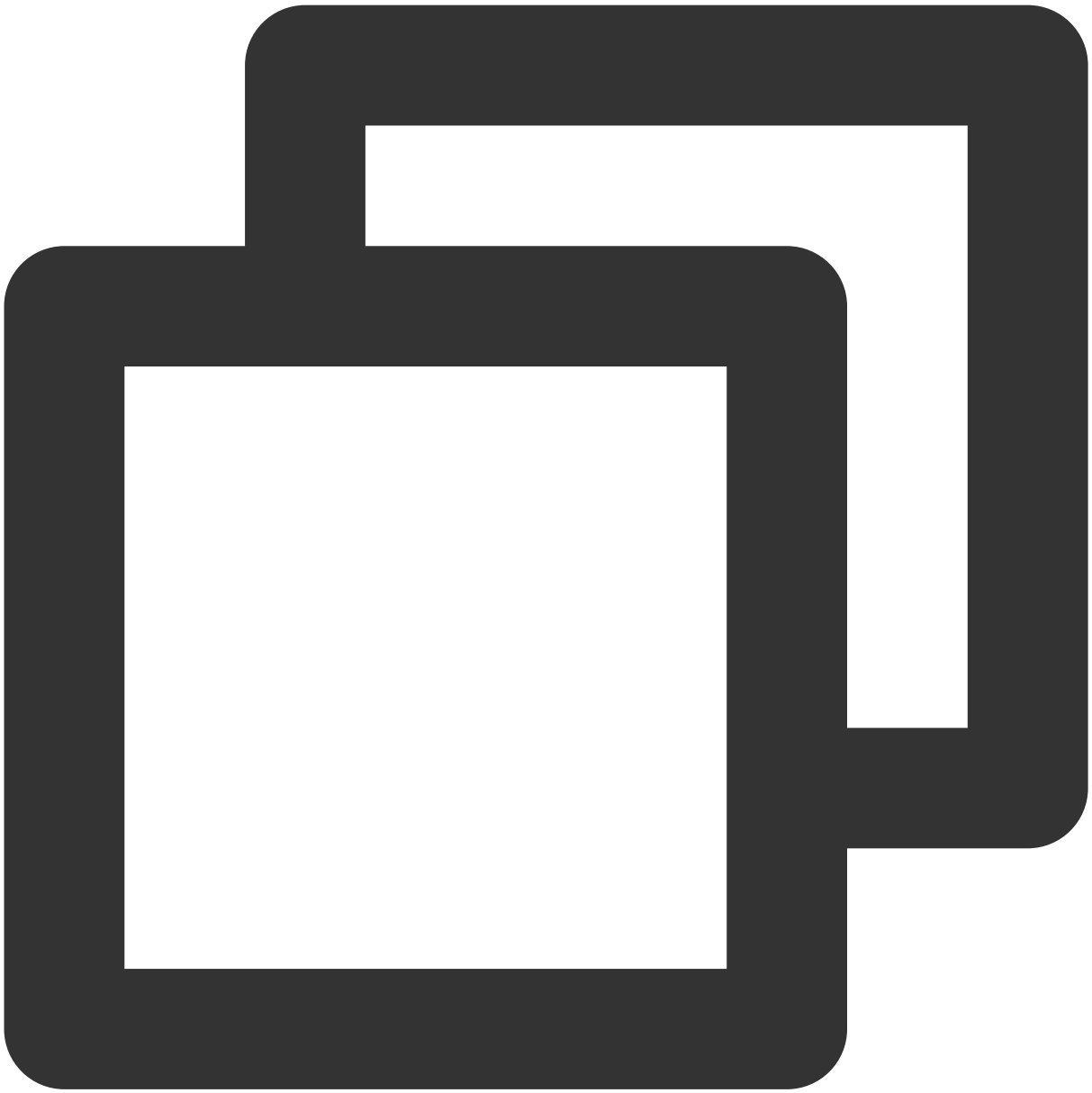


```
- (void)onAllUserMicrophoneDisableChanged:(NSString *)roomId isDisable:(BOOL)isDisa
```

参数	类型	含义
roomId	NSString *	房间 ID
isDisable	BOOL	是否被禁用

onAllUserCameraDisableChanged

房间内所有用户摄像头被禁用事件。

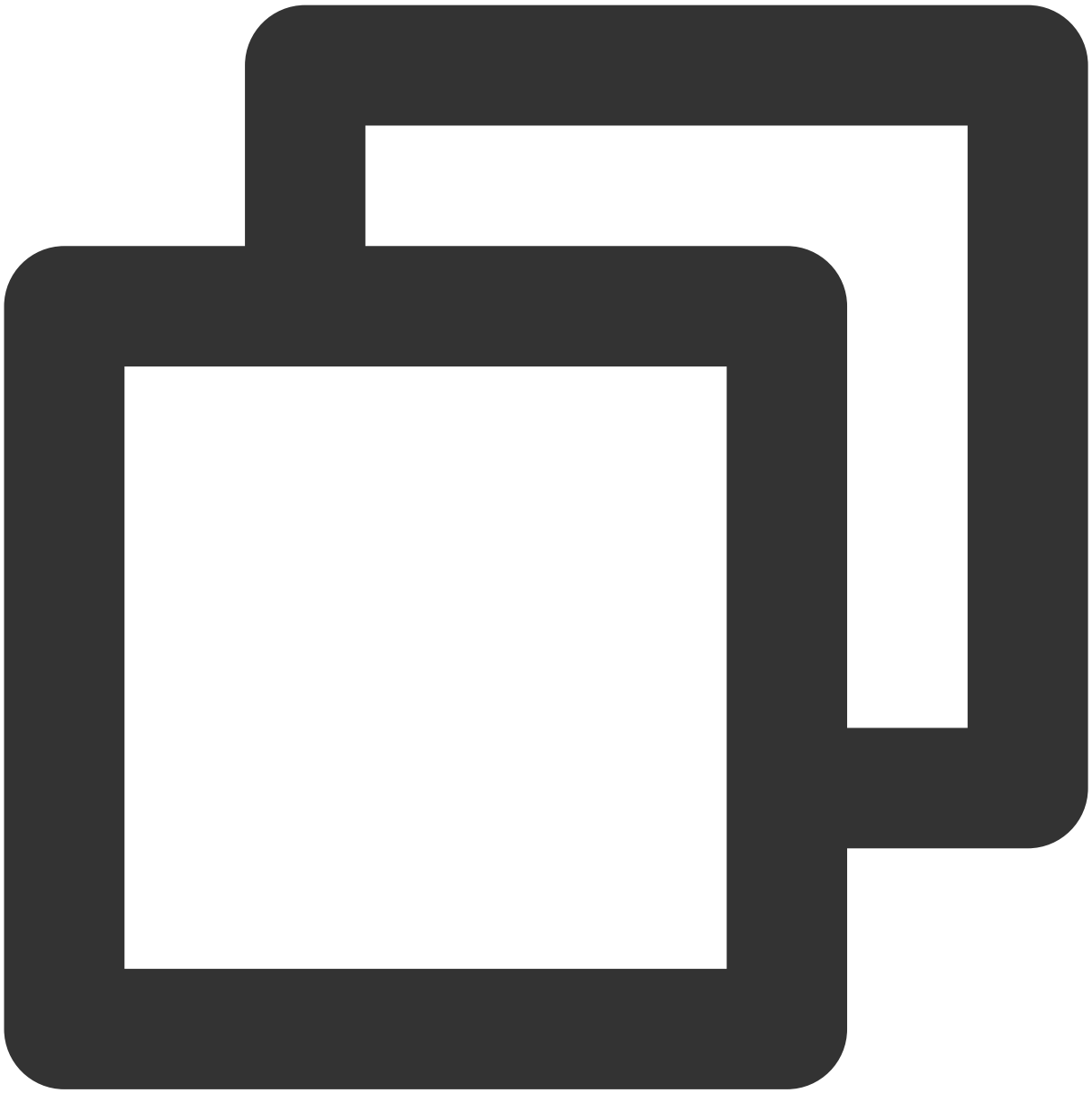


```
- (void)onAllUserCameraDisableChanged:(NSString *)roomId isDisable:(BOOL)isDisable;
```

参数	类型	含义
roomId	NSString *	房间 ID
isDisable	BOOL	是否被禁用

onSendMessageForAllUserDisableChanged

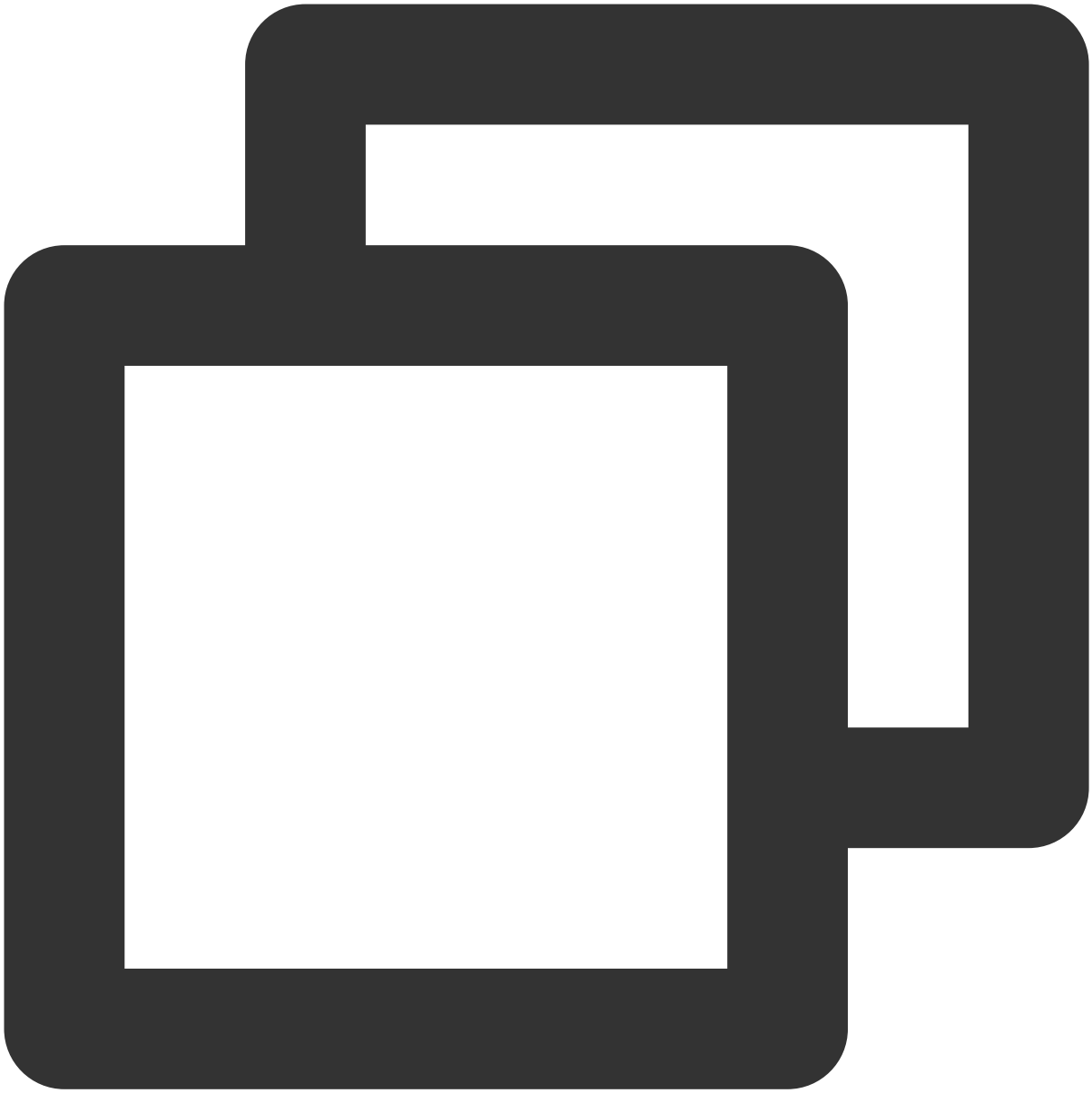
房间内所有用户发送文本消息被禁用事件。



```
- (void)onSendMessageForAllUserDisableChanged:(NSString *)roomId isDisable:(BOOL)is
```

onKickedOutOfRoom

被踢出房间事件。



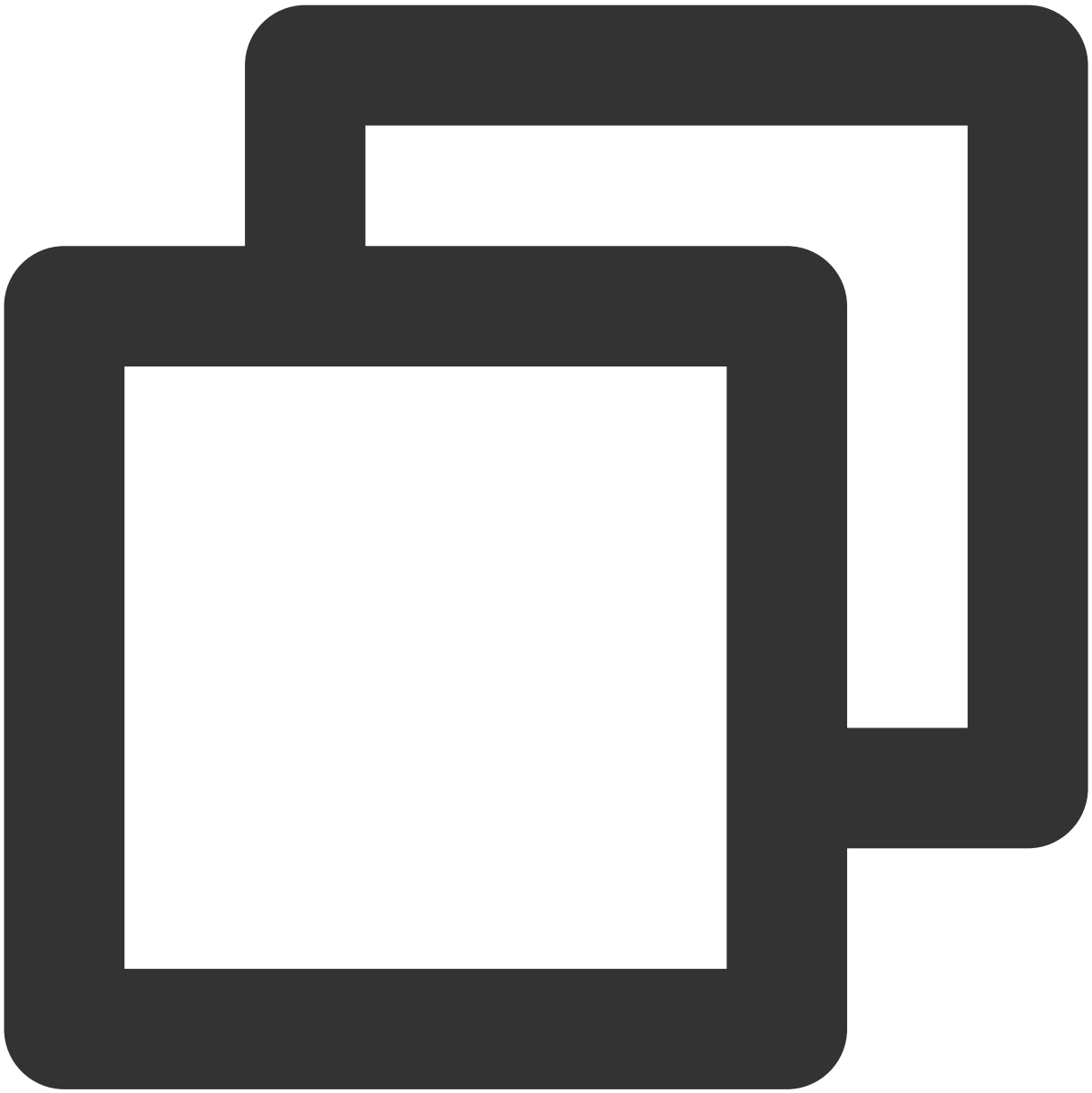
```
- (void)onKickedOutOfRoom:(NSString *)roomId message:(NSString *)message;
```

参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID
message	NSString *	被踢出的描述

onRoomDismissed

房间被解散事件。



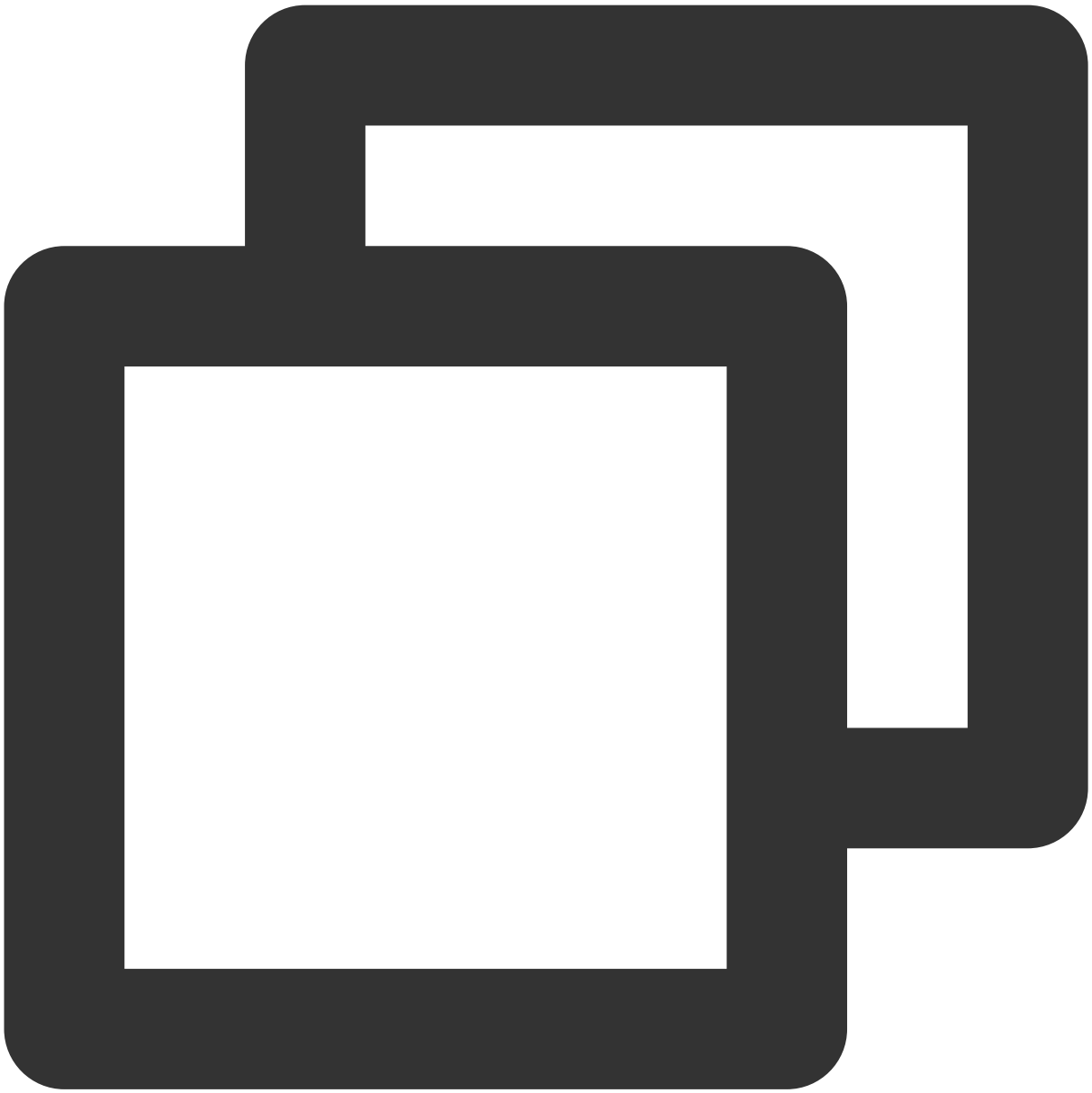
```
- (void)onRoomDismissed:(NSString *)roomId;
```

参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID

onRoomSpeechModeChanged

房间麦控模式发生变化。

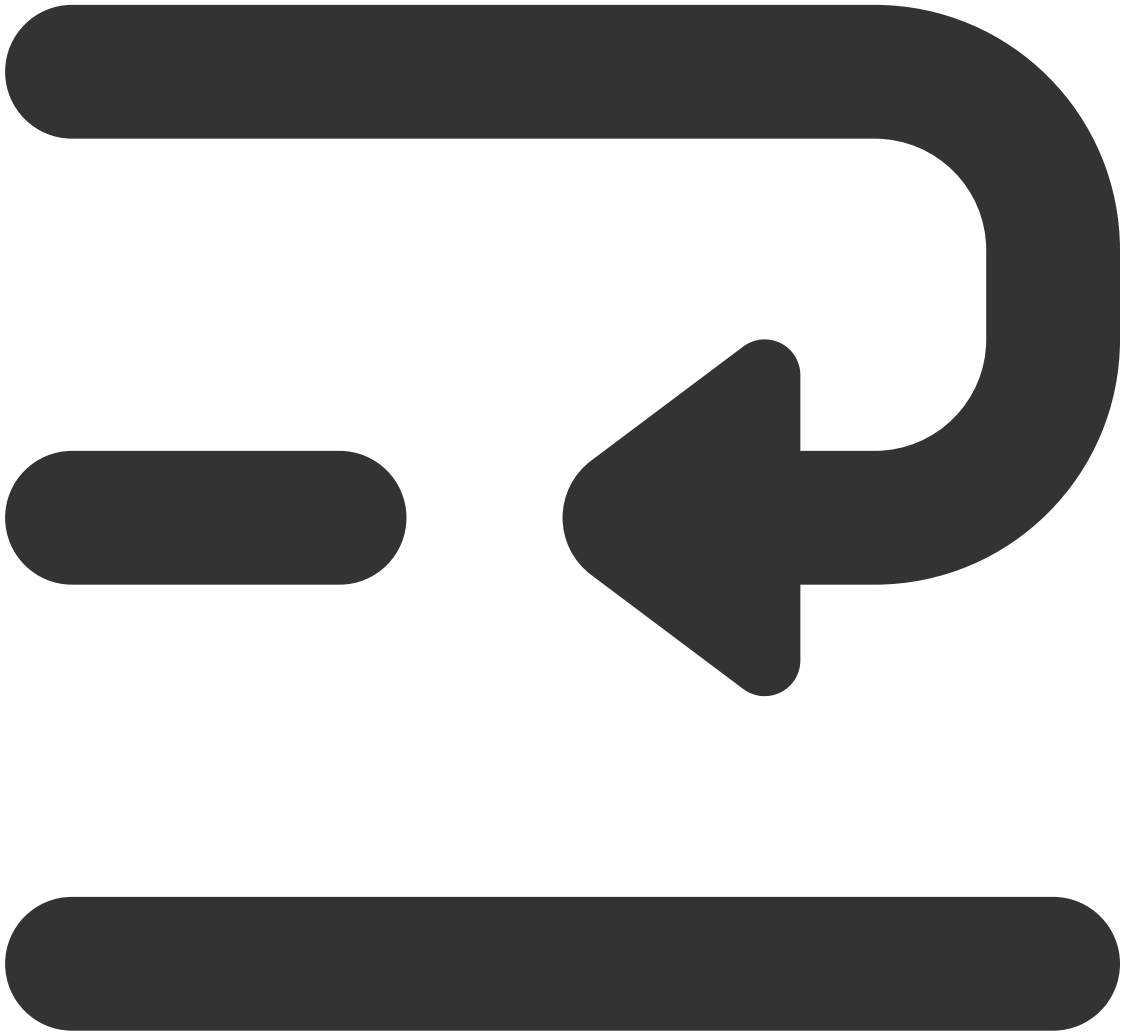


```
- (void)onRoomSpeechModeChanged:(NSString *)roomId speechMode:(TUISpeechMode)mode;
```

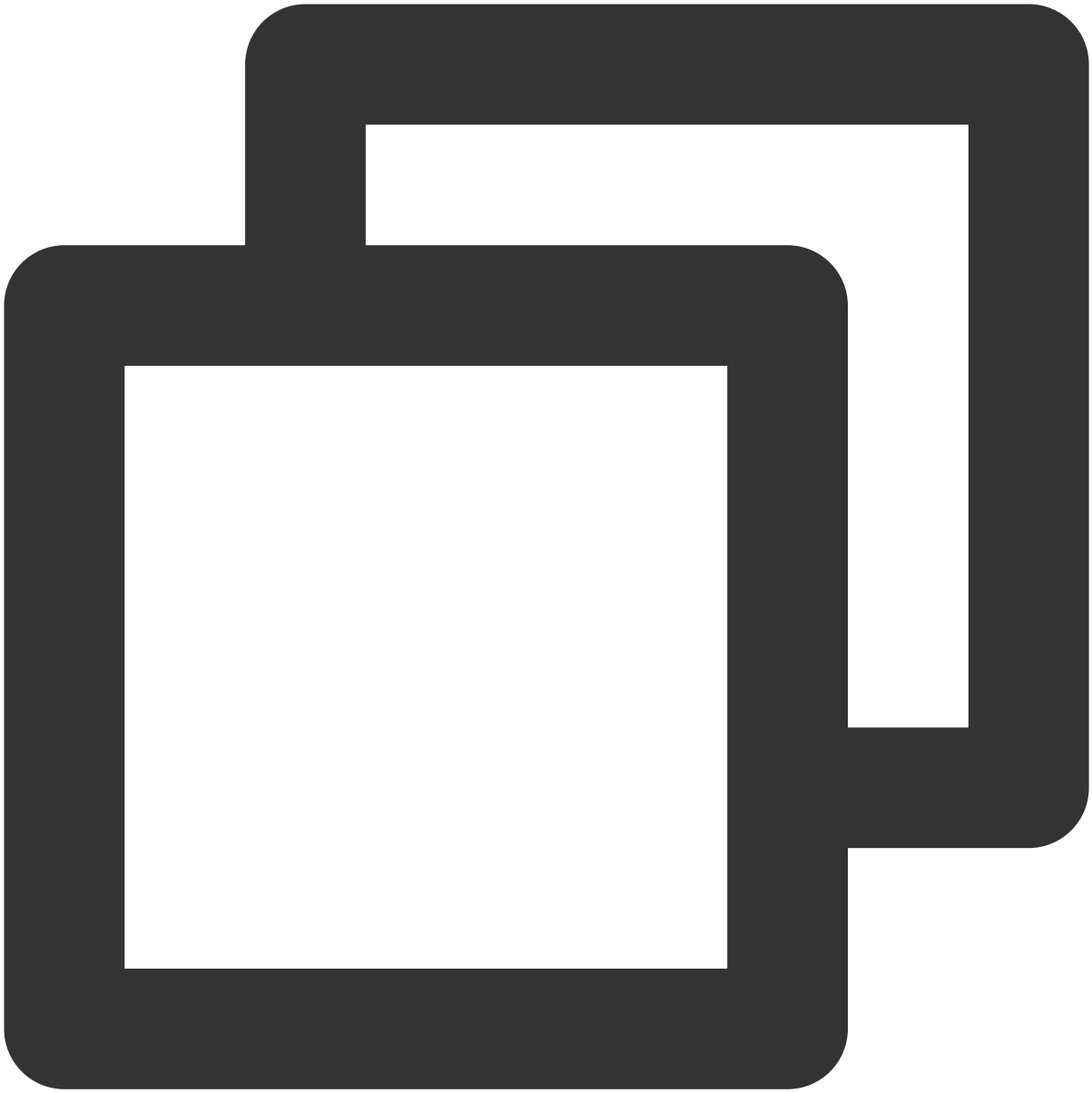
参数	类型	含义
roomId	NSString *	房间 ID
mode	TUISpeechMode	麦控模式

onRemoteUserEnterRoom

远端用户进房事件。



版权所有：腾讯云计算（北京）有限责任公司



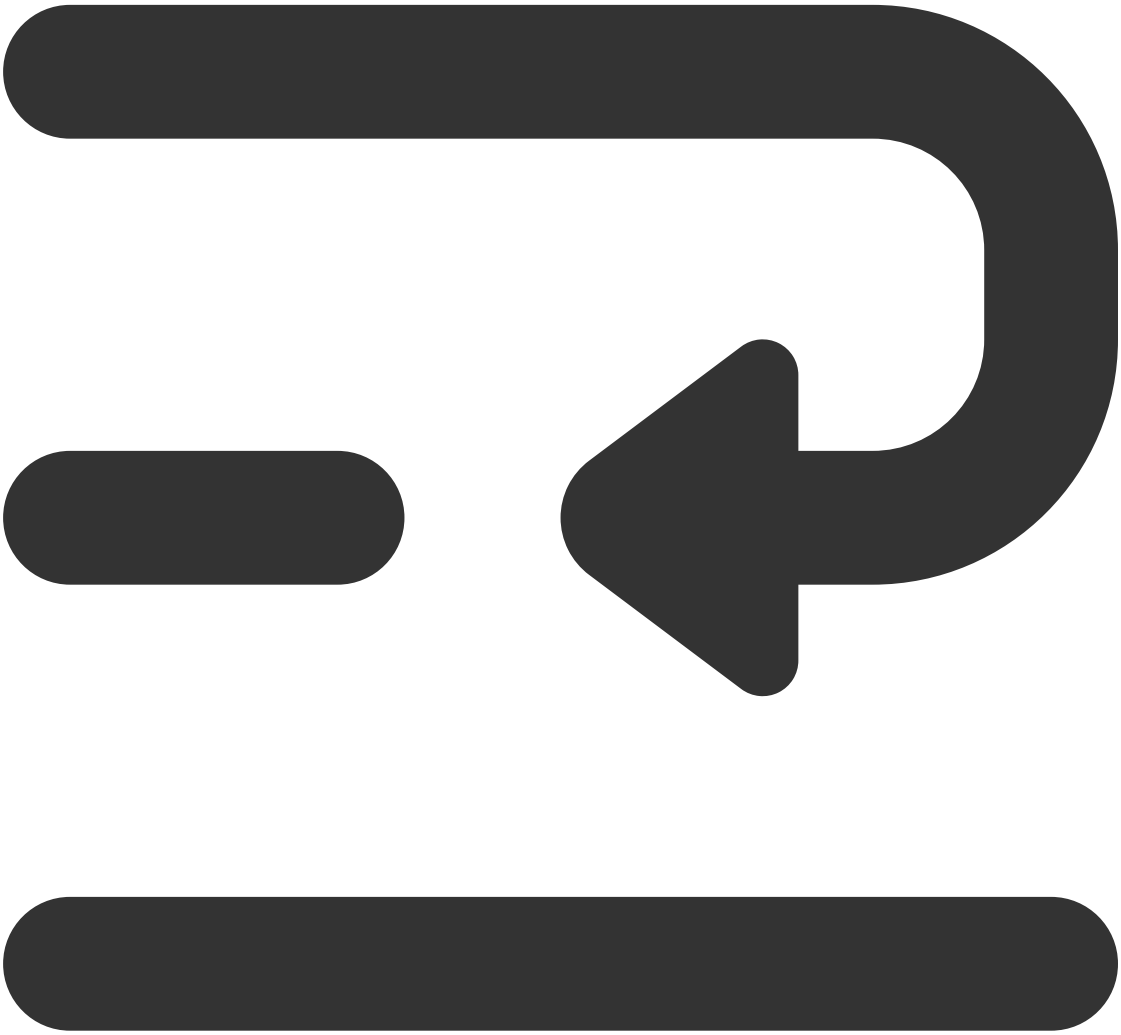
```
- (void)onRemoteUserEnterRoom:(NSString *)roomId userInfo:(TUIUserInfo *)userInfo;
```

参数如下表所示：

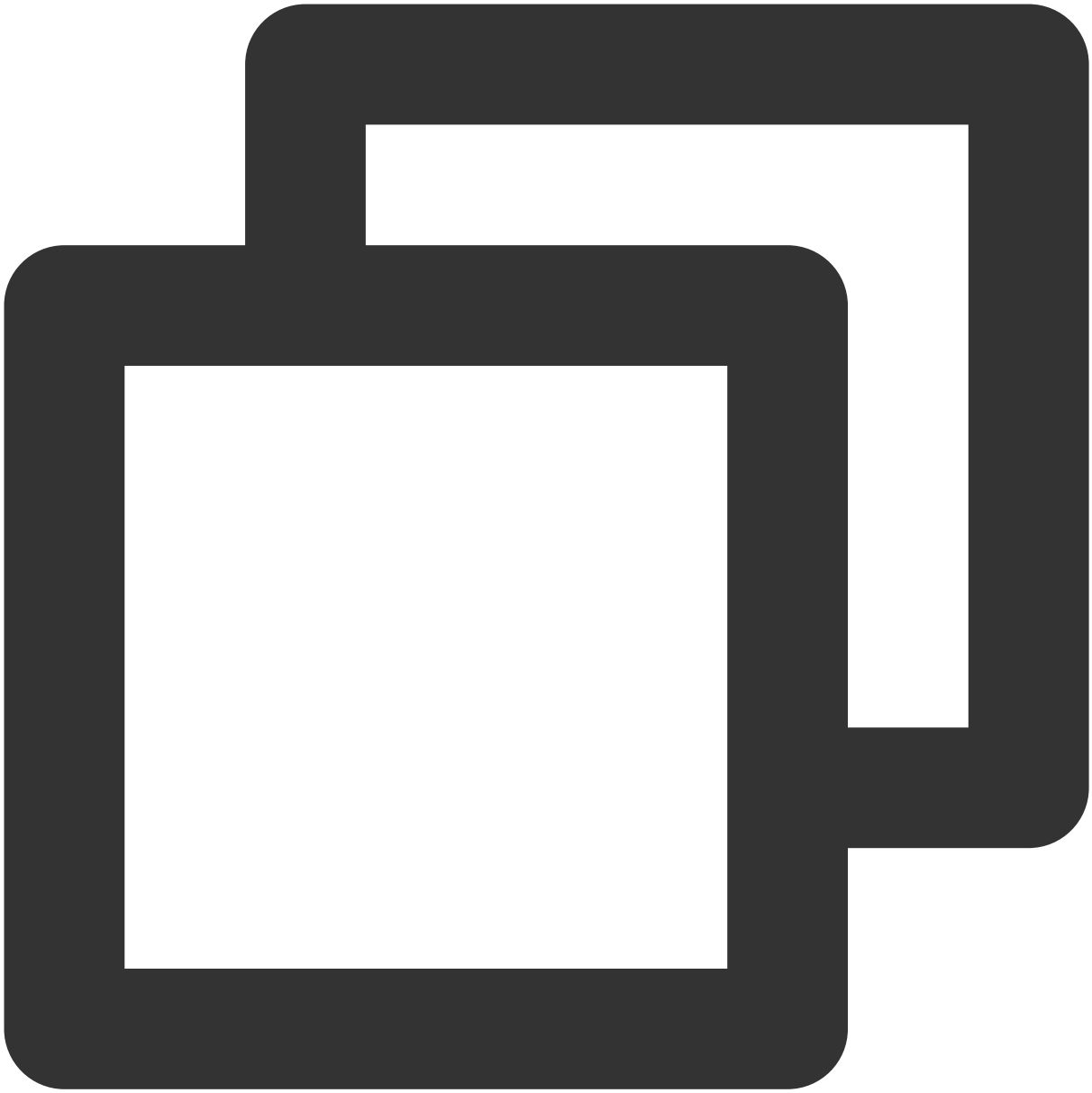
参数	类型	含义
roomId	NSString *	房间 ID
userInfo	TUIUserInfo *	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件。



版权所有：腾讯云计算（北京）有限责任公司



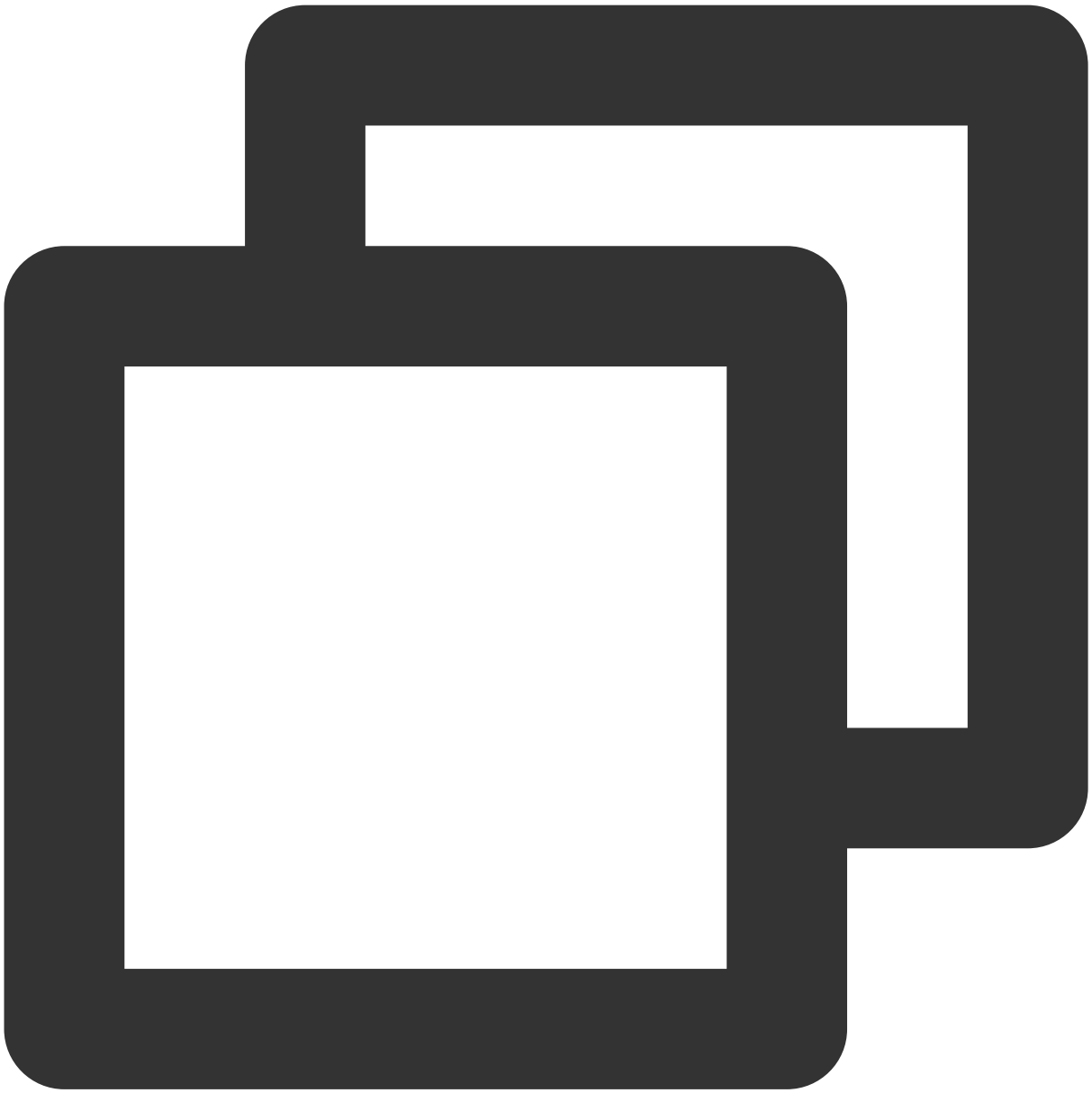
```
- (void)onRemoteUserLeaveRoom:(NSString *)roomId userInfo:(TUIUserInfo *)userInfo;
```

参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID
userInfo	TUIUserInfo *	用户信息

onUserRoleChanged

用户角色发生变化事件。



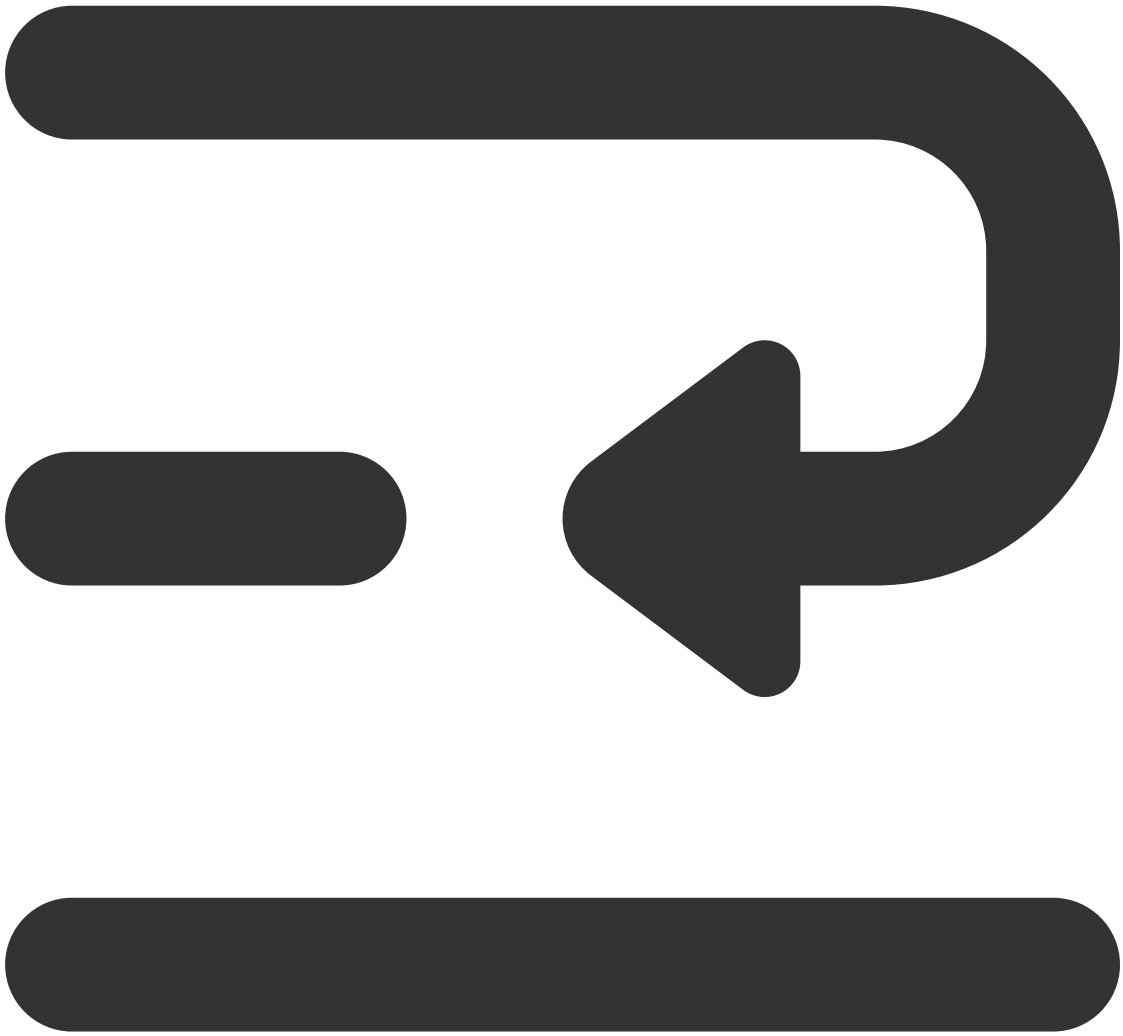
```
- (void)onUserRoleChanged:(NSString *)userId userRole:(TUIRole)userRole;
```

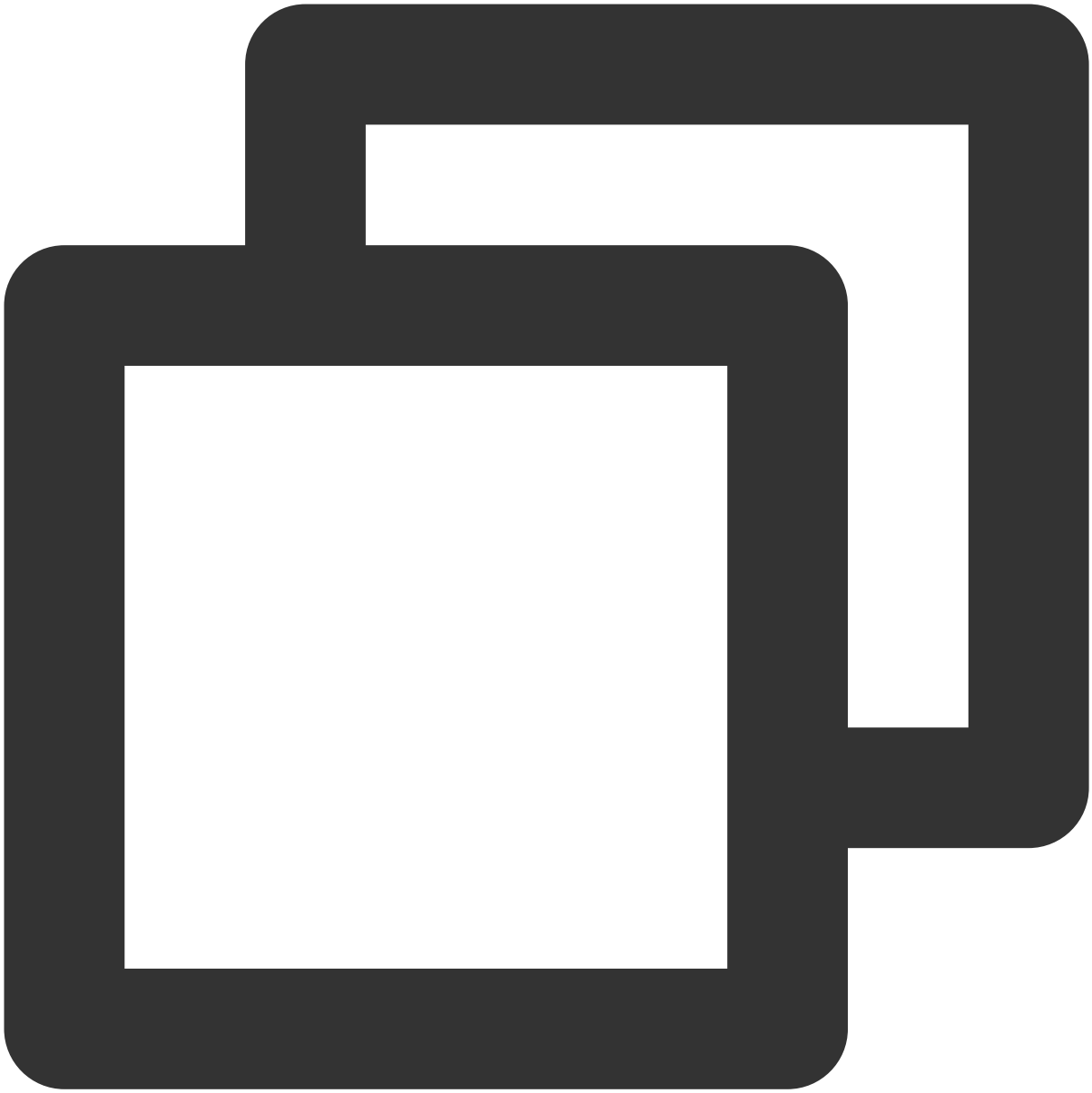
参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID
userRole	TUIRole	用户角色

onUserVideoStateChanged

用户视频状态发生变化事件。





```
- (void)onUserVideoStateChanged:(NSString *)userId
                     streamType:(TUIVideoStreamType)streamType
                       hasVideo:(BOOL)hasVideo
                         reason:(TUIChangeReason)reason;
```

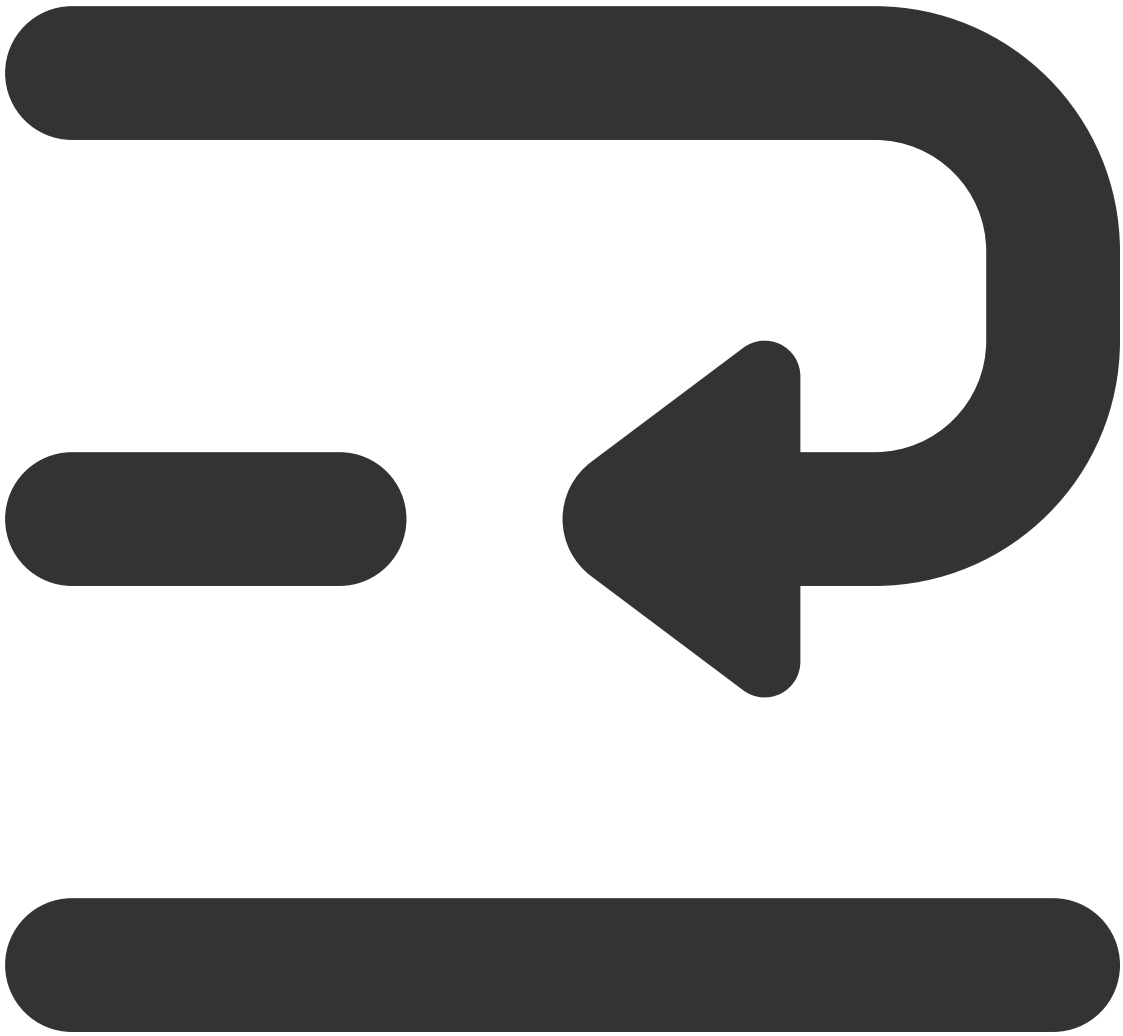
参数如下表所示：

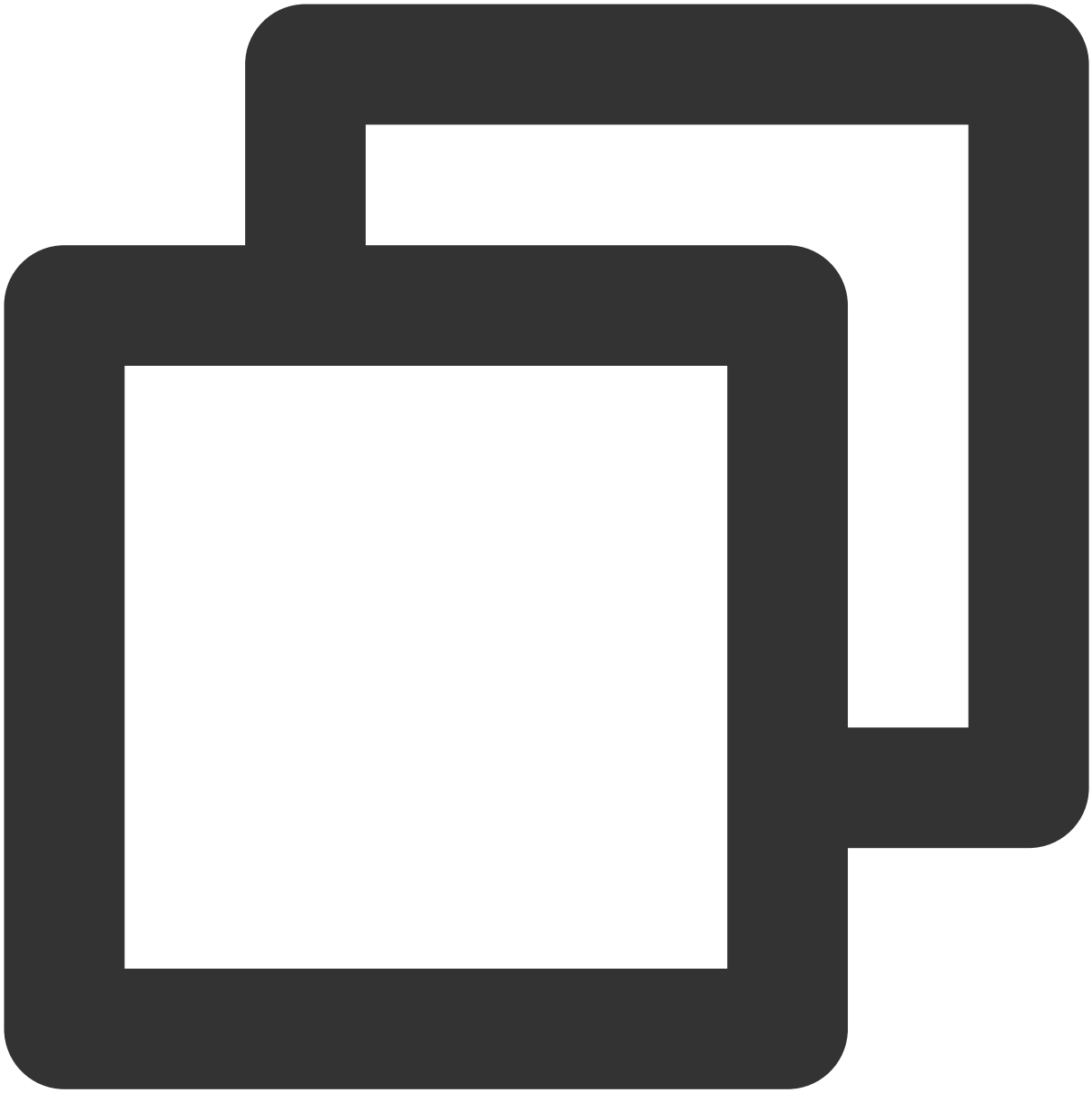
参数	类型	含义
userId	NSString *	用户 ID

streamType	TUIVideoStreamType	视频流类型
hasVideo	BOOL	是否有视频流
reason	TUIChangeReason	视频流发生变化原因

onUserAudioStateChanged

用户音频状态发生变化事件。





```
- (void)onUserAudioStateChanged:(NSString *)userId
                        hasAudio:(BOOL)hasAudio
                          reason:(TUIChangeReason)reason;
```

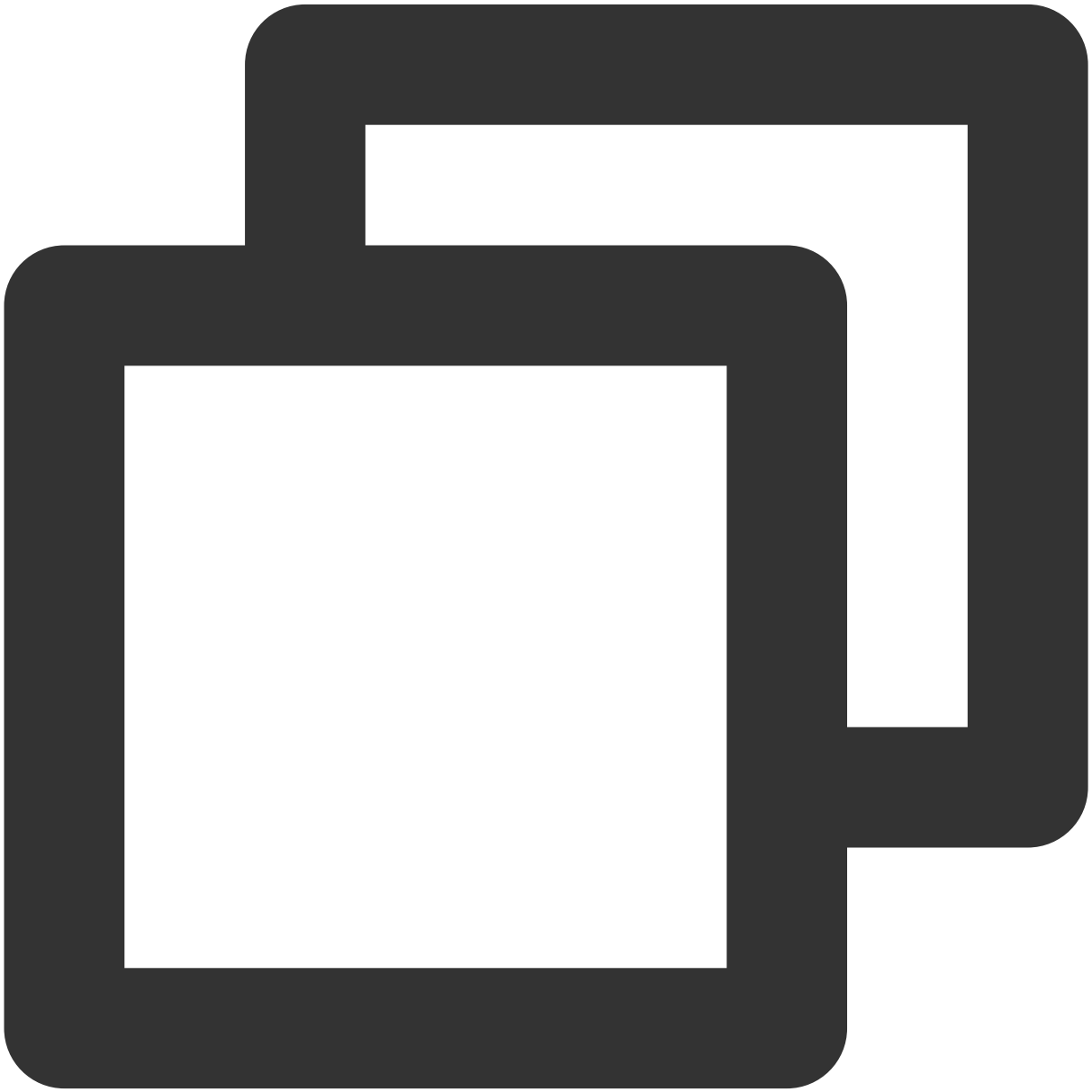
参数如下表所示：

参数	类型	含义
userId	NSString *	用户 ID
hasAudio	BOOL	是否有音频流

reason	TUIChangeReason	音频流发生变化原因

onUserScreenCaptureStopped

屏幕分享停止的事件回调。



```
- (void)onUserScreenCaptureStopped:(NSInteger)reason;
```

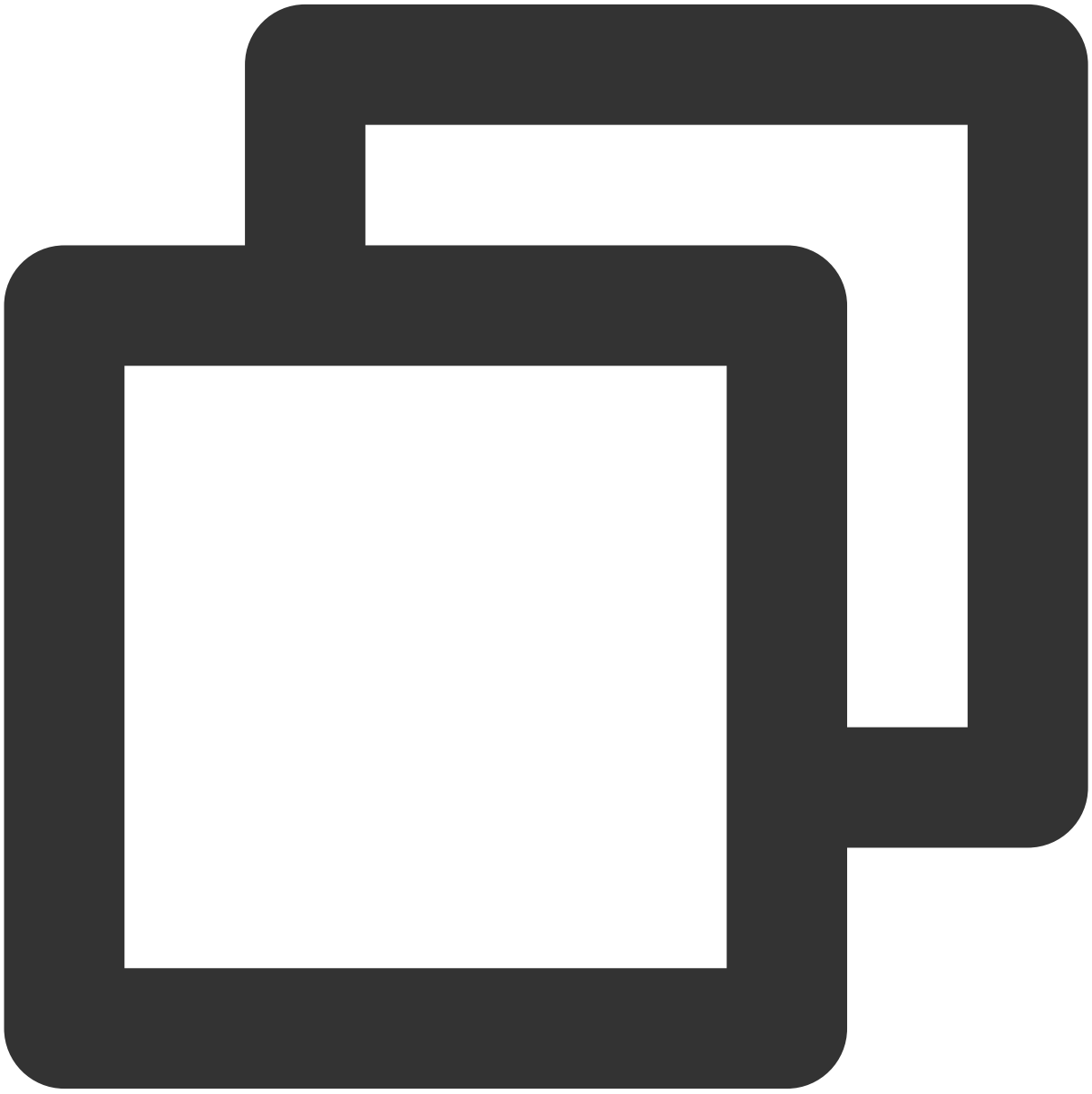
参数如下表所示：

参数	类型	含义

reason	NSInteger	停止原因： 0：用户主动停止 1：屏幕窗口关闭导致停止 2：表示屏幕分享的显示屏状态变更（如接口被拔出、投影模式变更等）

onRoomMaxSeatCountChanged

房间内最大麦位数发生变化事件（仅在会议类型房间生效）。

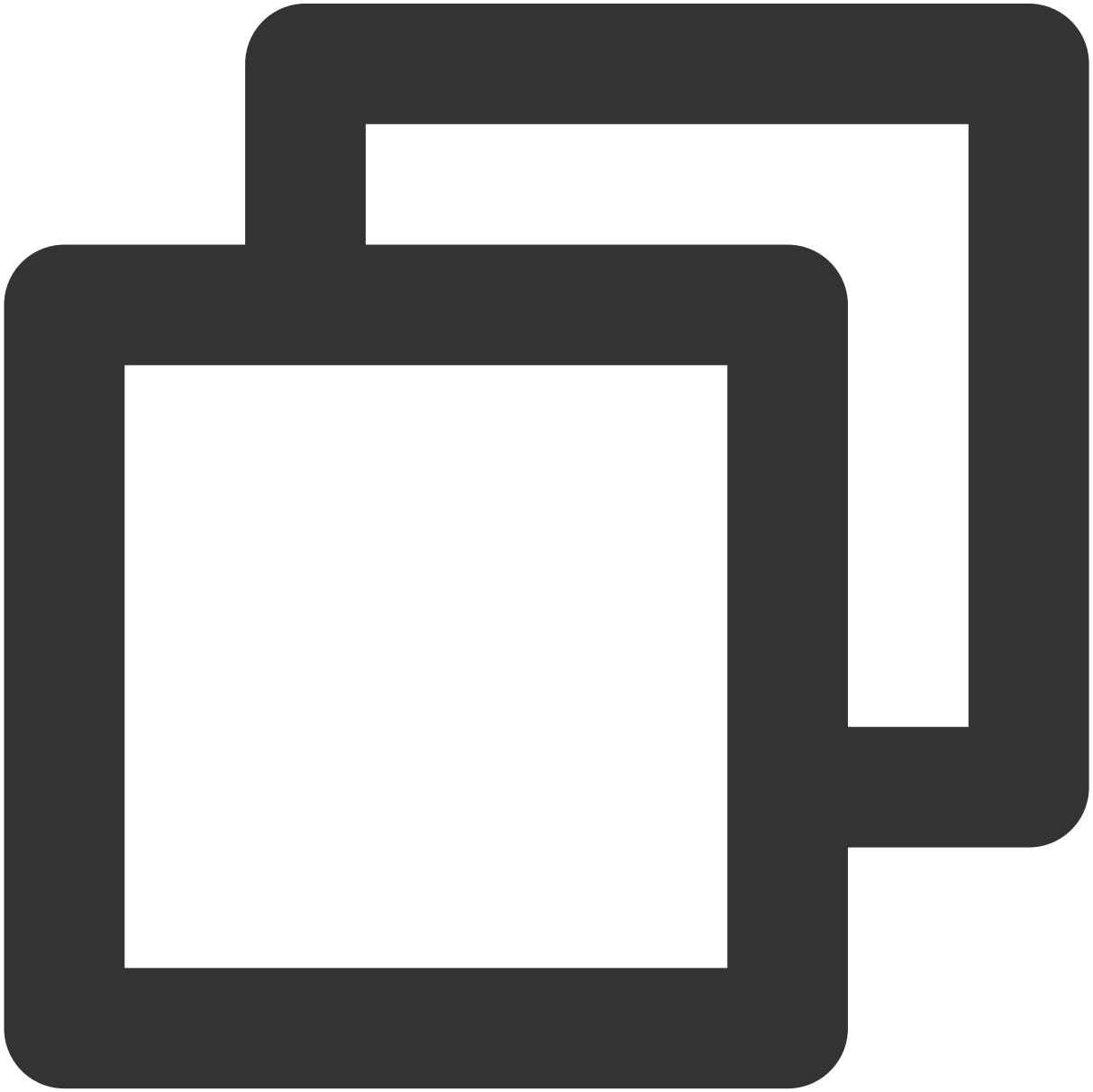


```
- (void)onRoomMaxSeatCountChanged:(NSString *)roomId maxSeatNumber:(NSInteger)maxSe
```


参数	类型	含义
roomId	NSString *	房间 ID
maxSeatNumber	NSInteger	房间内最大麦位数量

onUserVoiceVolumeChanged

用户音量变化事件。



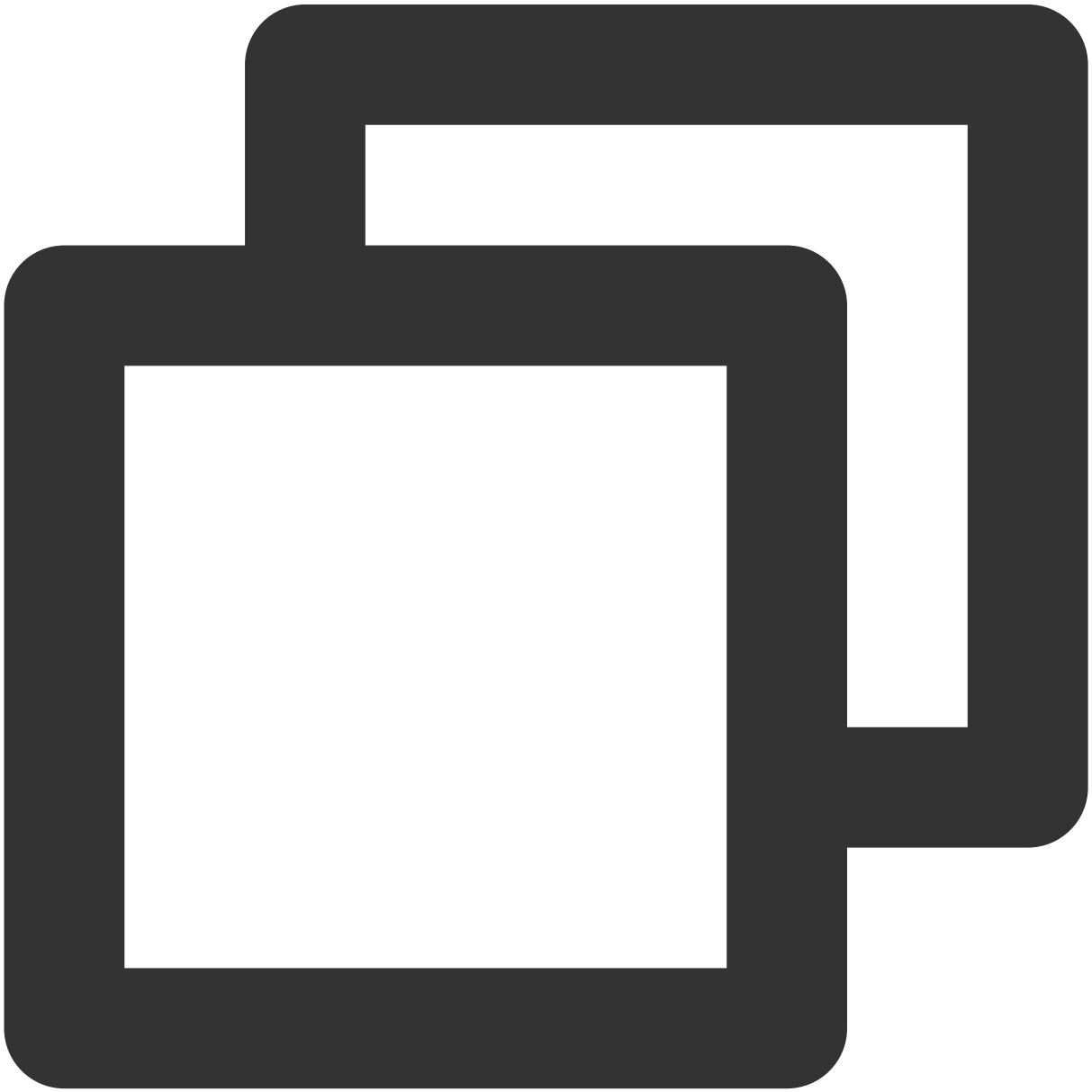
```
- (void)onUserVoiceVolumeChanged:(NSDictionary<NSString *, NSNumber *> *)volumeMap;
```

参数如下表所示：

参数	类型	含义
volumeMap	NSDictionary<NSString *, NSNumber *> *	用户音量 map

onSendMessageForUserDisableChanged

用户文本消息发送能力发生变化事件。



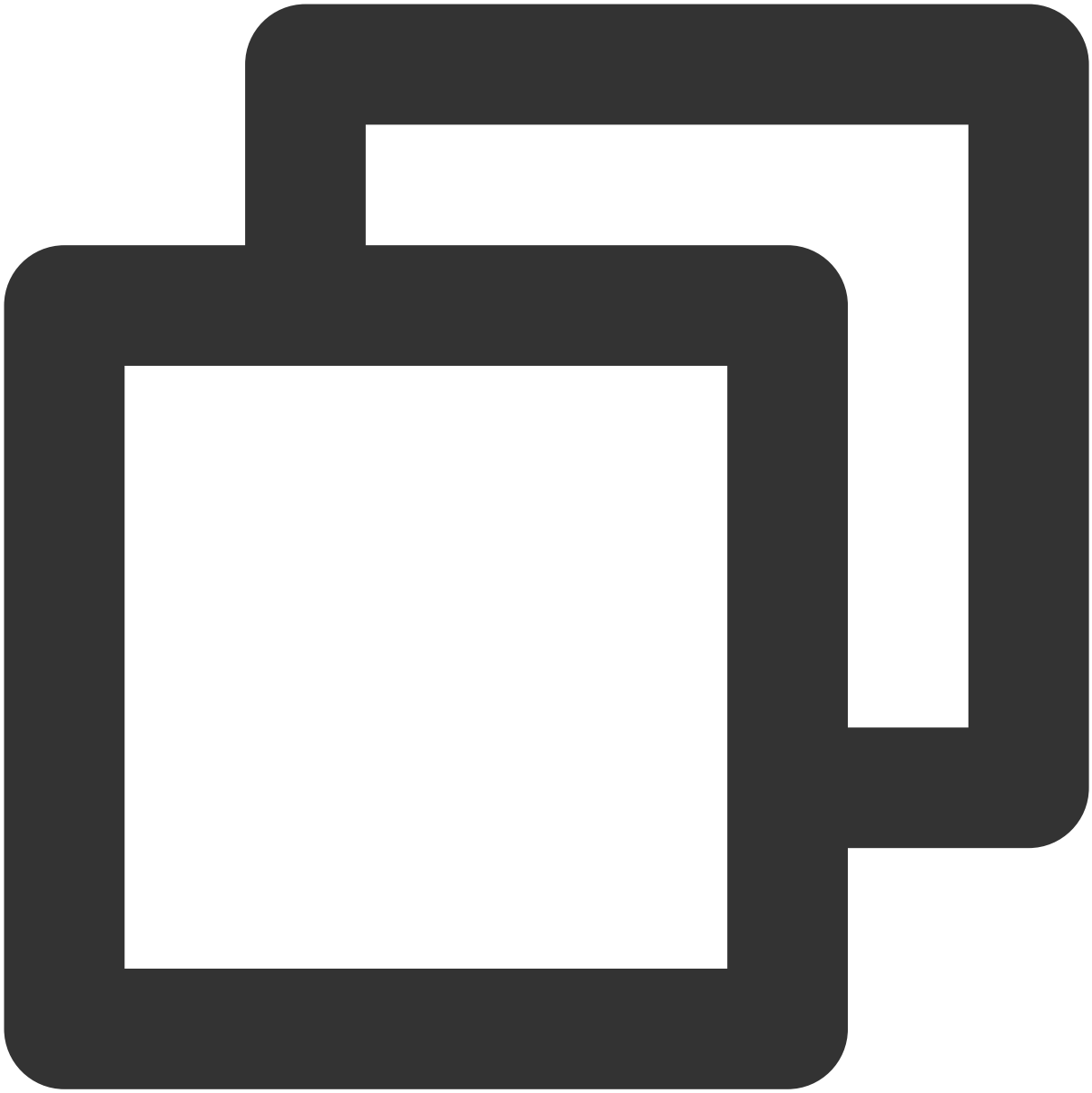
```
- (void)onSendMessageForUserDisableChanged:(NSString *)roomId
                                    userId:(NSString *)userId
```

```
isDisable:(BOOL)muted;
```

参数	类型	含义
roomId	NSString *	房间 ID
userId	NSString *	用户 ID
muted	BOOL	是否被禁止发送文本消息

onUserNetworkQualityChanged

用户网络状态变化事件。



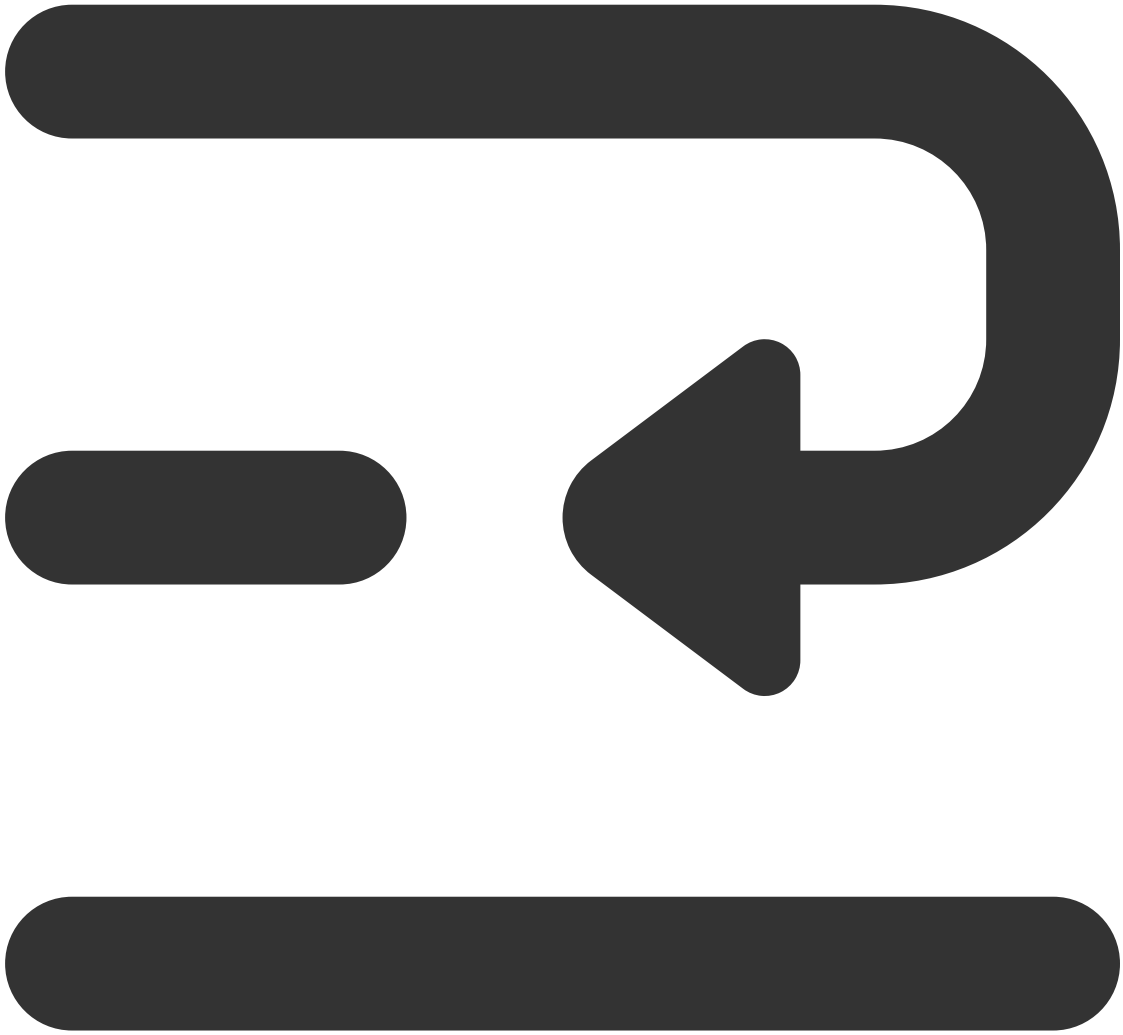
```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkInfo *> *)networkList;
```

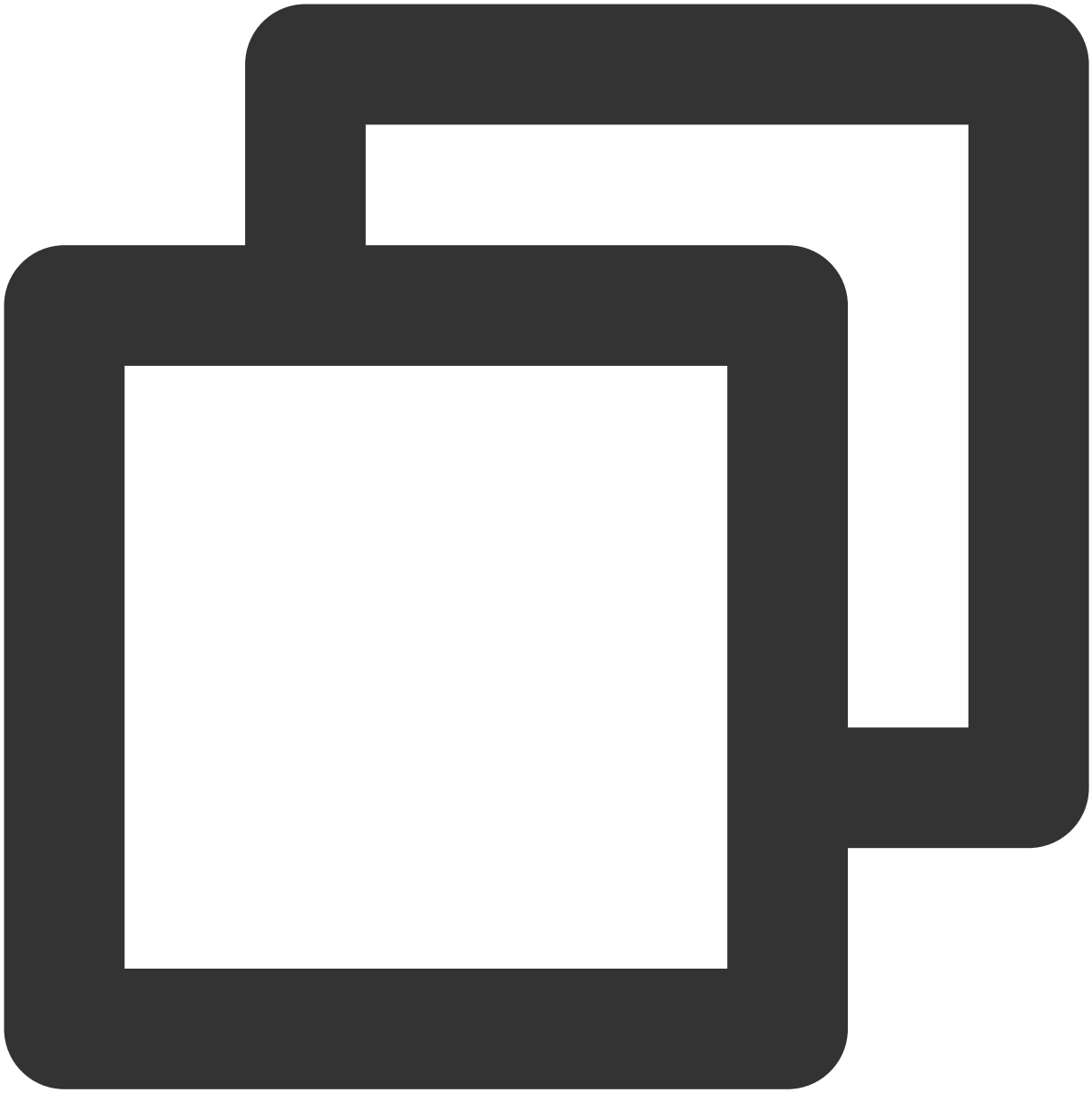
参数如下表所示：

参数	类型	含义
networkList	NSArray<TUINetworkInfo>*	用户网络状态数组，可以参考TUINetworkInfo对象

onSeatListChanged

麦位列表发生变化事件。





```
- (void)onSeatListChanged:(NSArray<TUISeatInfo *> *)seatList
                   seated:(NSArray<TUISeatInfo *> *)seatedList
                     left:(NSArray<TUISeatInfo *> *)leftList;
```

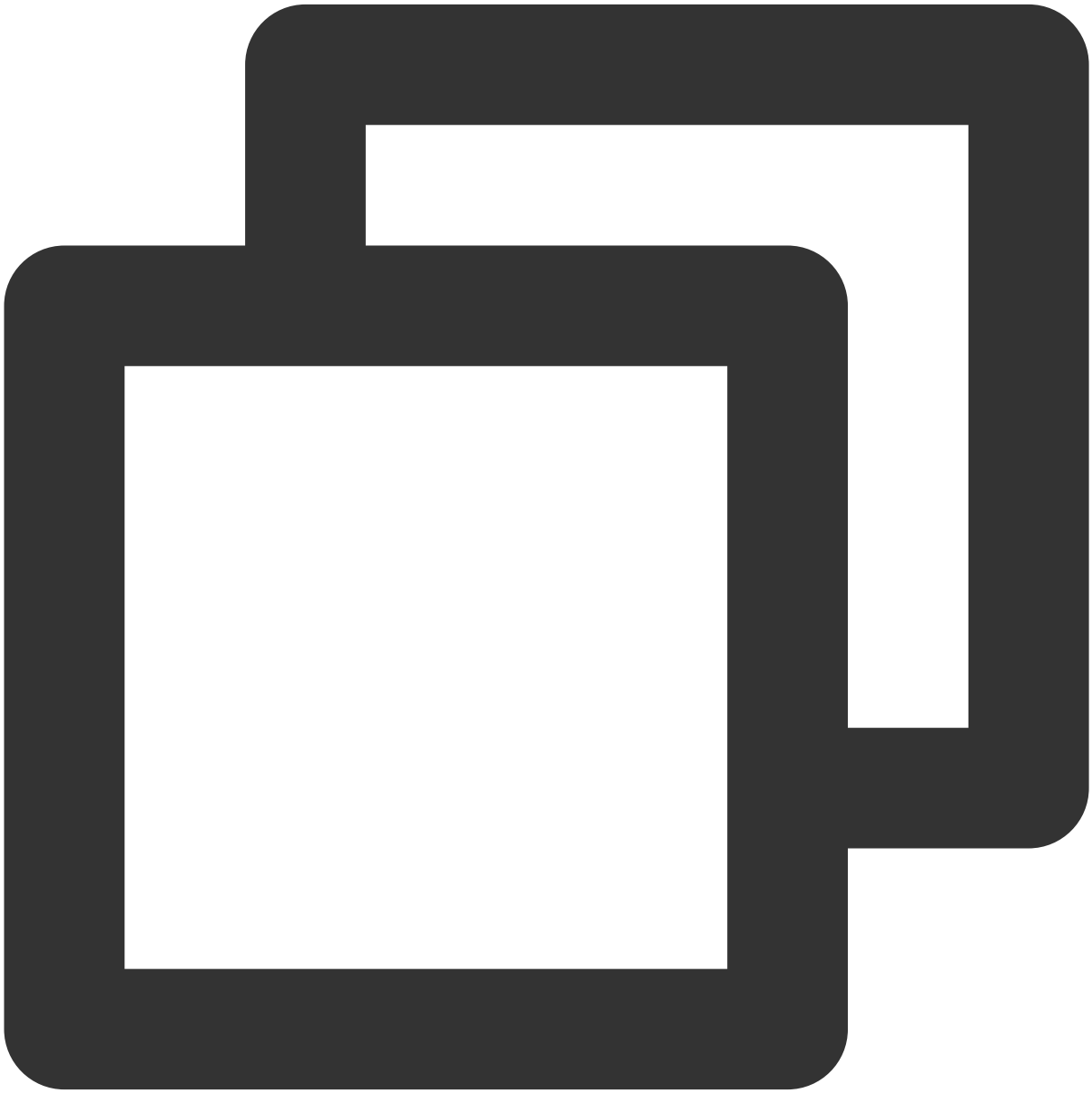
参数如下表所示：

参数	类型	含义
seatList	NSArray<TUISeatInfo *> *	目前麦上最新的用户列表，包含新上麦的用户
seatedList	NSArray<TUISeatInfo *> *	新上麦的用户列表

leftList	NSArray<TUISeatInfo *> *	新下麦的用户列表

onKickedOffSeat

收到用户被踢下麦事件。

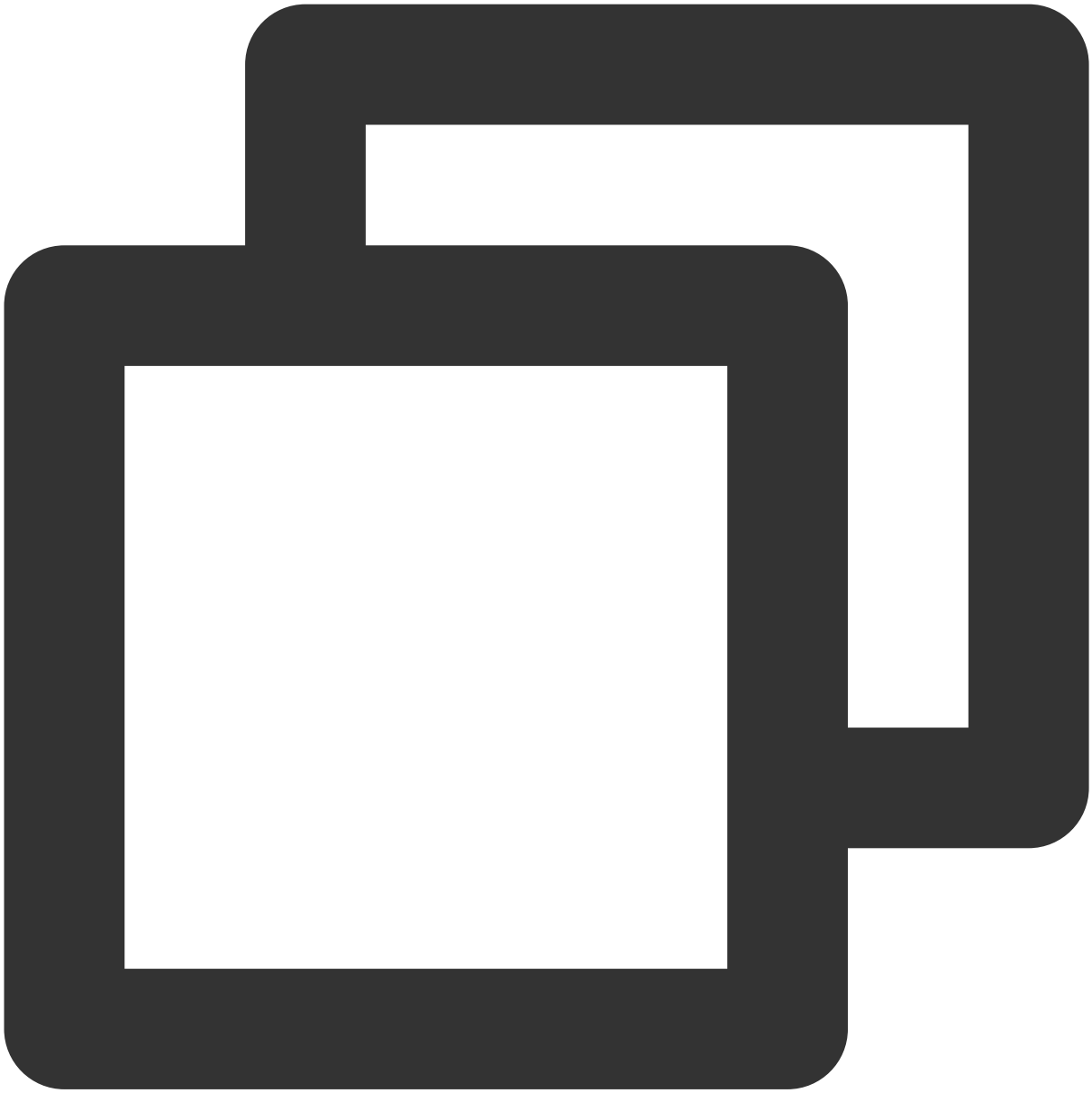


```
- (void)onKickedOffSeat:(NSString *)userId;
```

参数	类型	含义
userId	NSString *	用户 ID

onRequestReceived

收到请求消息事件。



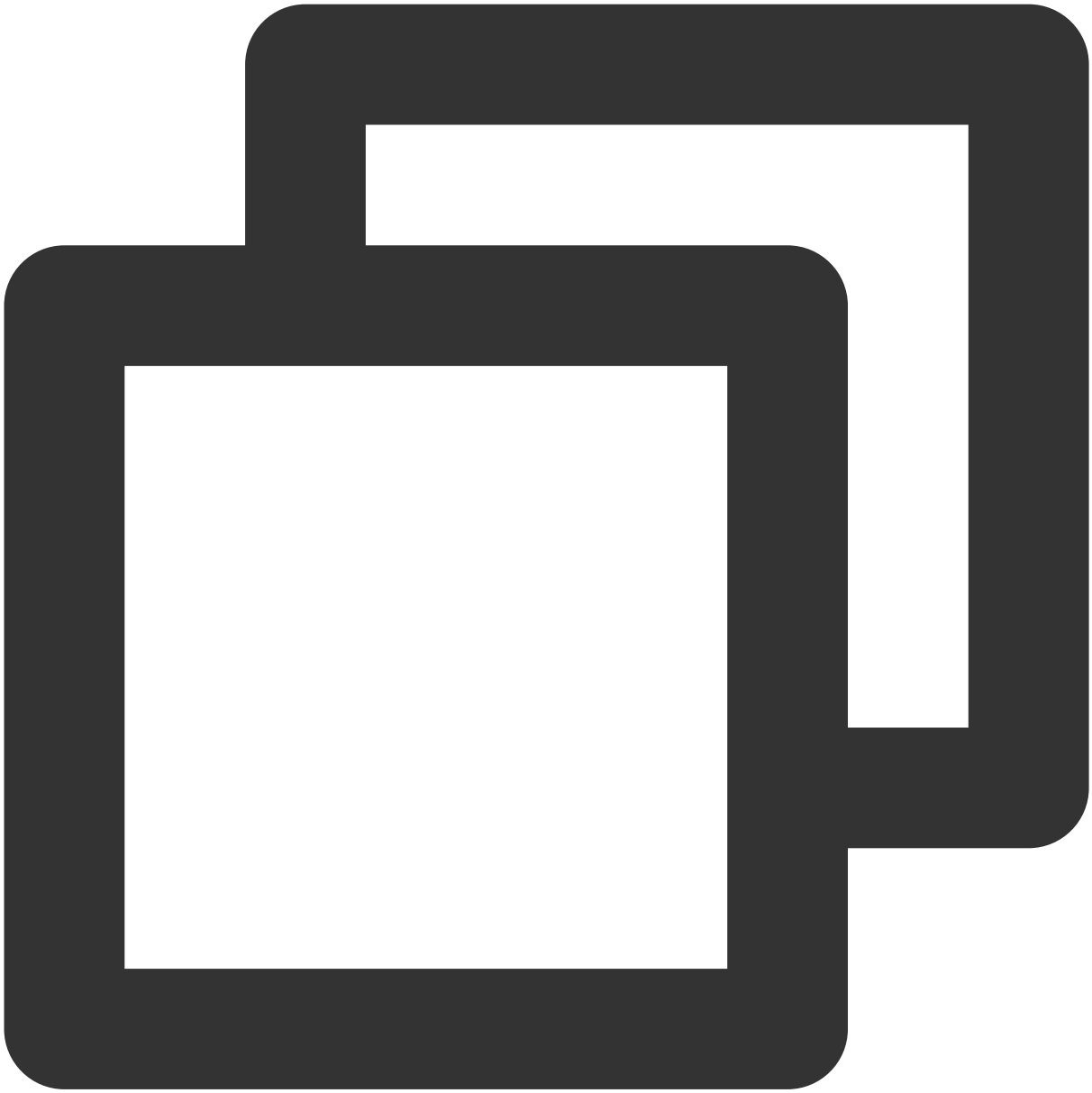
```
- (void)onRequestReceived:(TUIRequest *)request;
```

参数如下表所示：

参数	类型	含义
request	TUIRequest *	请求内容

onRequestCancelled

收到请求被取消的事件。



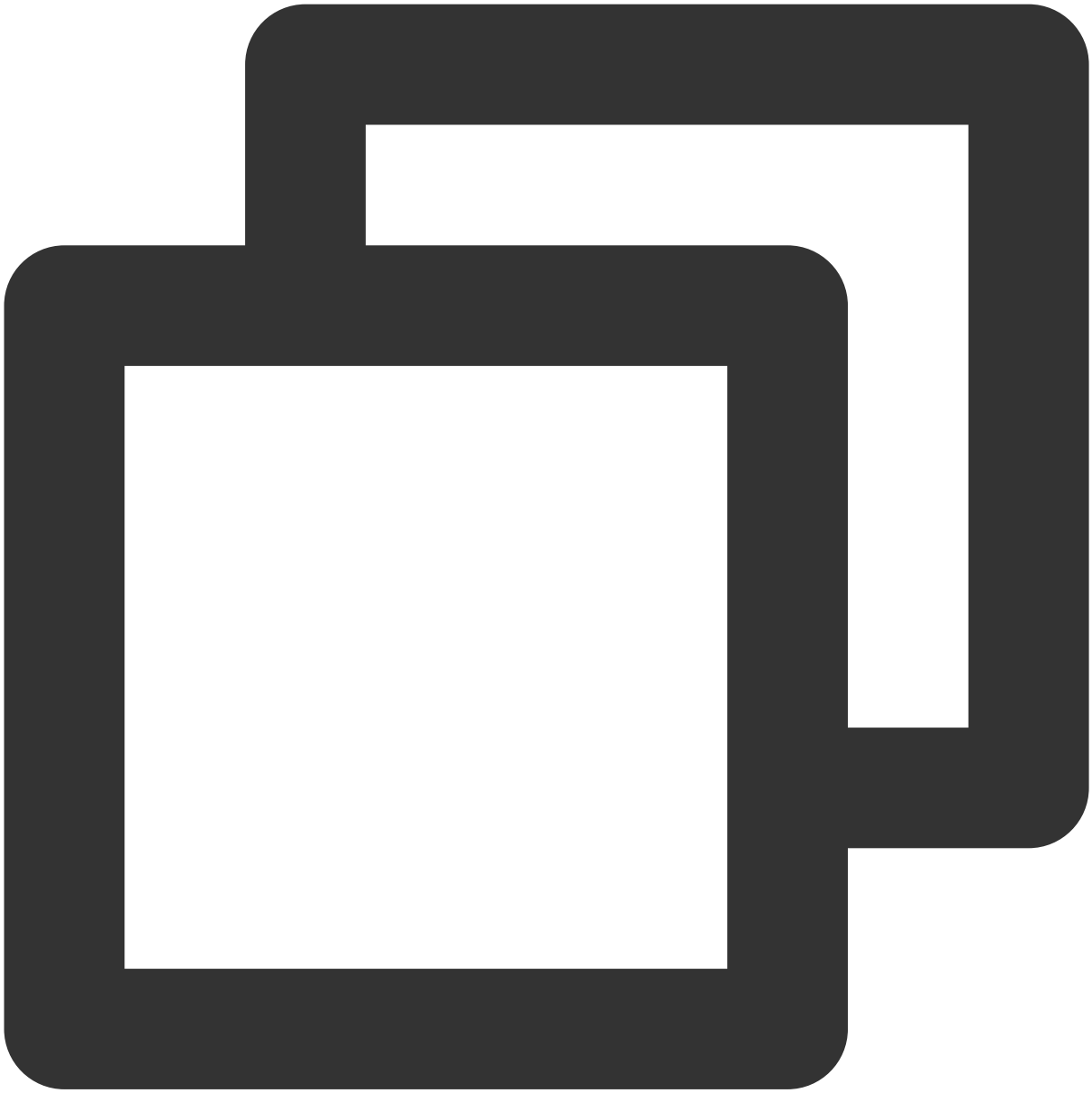
```
- (void)onRequestCancelled:(NSString *)requestId;
```

参数如下表所示：

参数	类型	含义
requestId	NSString *	请求 ID

onReceiveTextMessage

收到普通文本消息事件。



```
- (void)onReceiveTextMessage:(NSString *)roomId
                     message:(TUIMessage *)message;
```

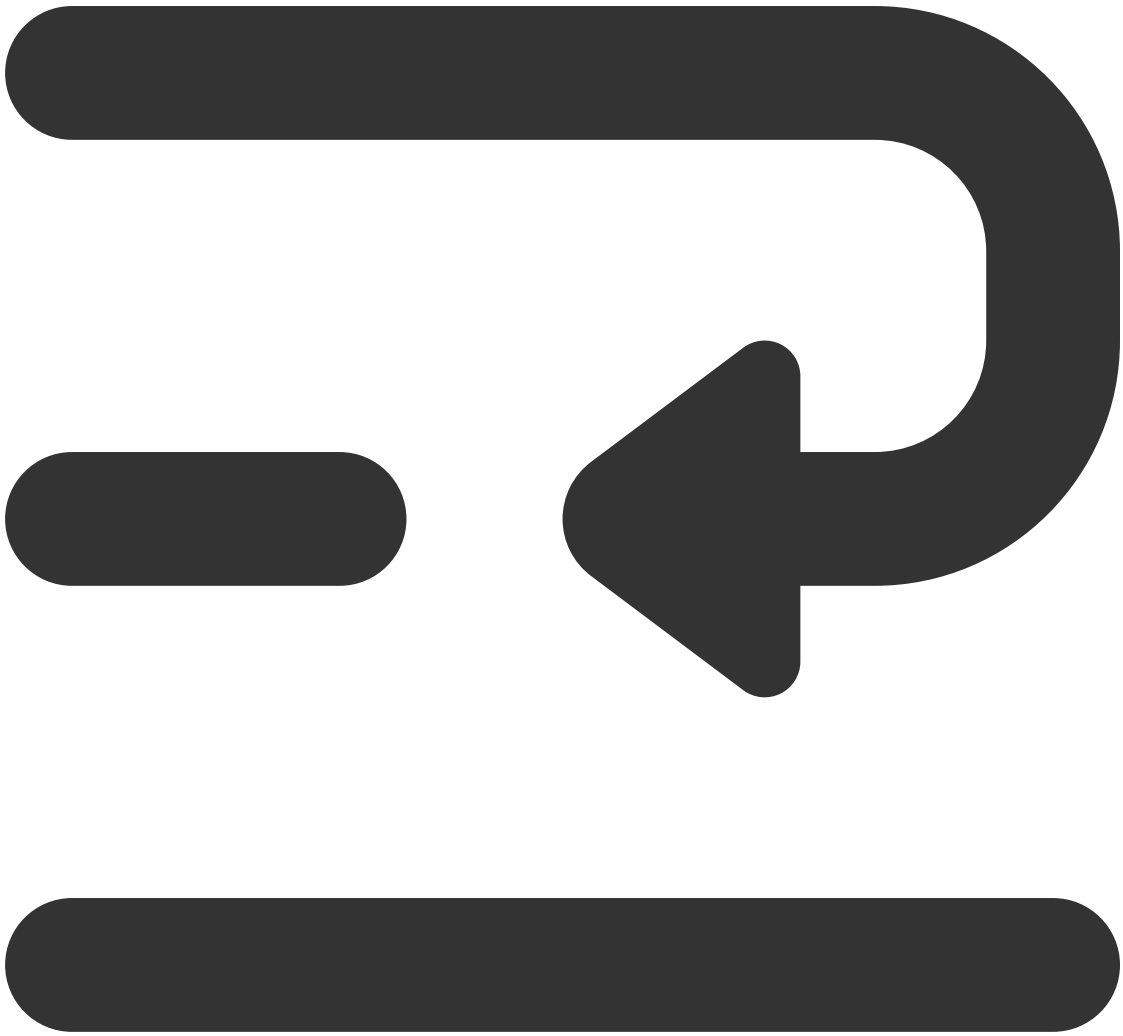
参数如下表所示：

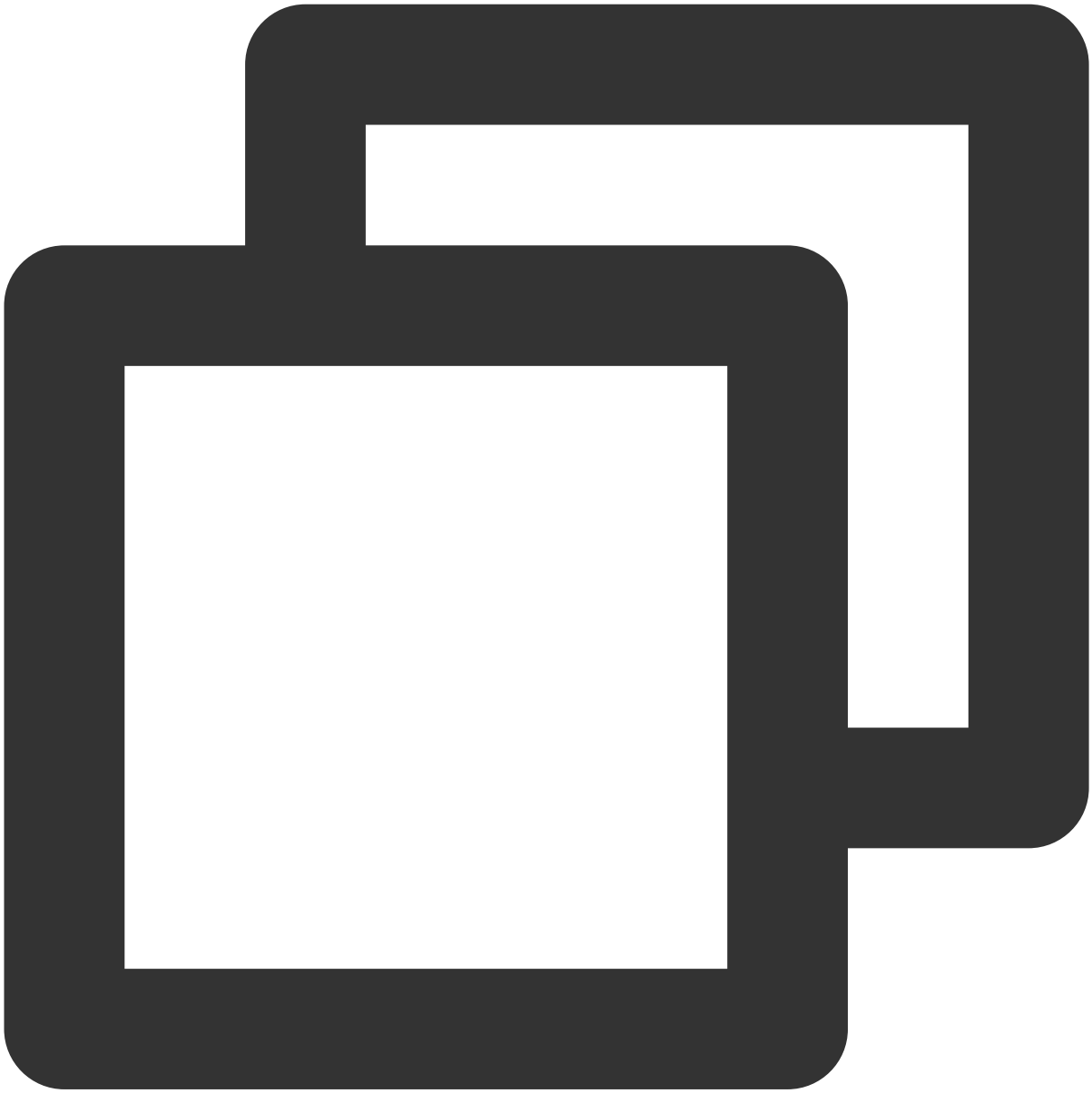
参数	类型	含义
roomId	NSString *	房间 ID

message	TUIMessage	消息内容，详细定义可以参考TUIRoomDefine.h文件中TUIMessage的定义

onReceiveCustomMessage

收到自定义消息事件。





```
- (void)onReceiveCustomMessage:(NSString *)roomId
                       message:(TUIMessage *)message;
```

参数如下表所示：

参数	类型	含义
roomId	NSString *	房间 ID
message	TUIMessage	消息内容

版权所有：腾讯云计算（北京）有限责任公司

类型定义

最近更新时间：2023-10-19 17:27:38

枚举定义

TUIRoomDefine

类型	说明
TUIRoomType	房间类型
TUISpeechMode	麦控模式
TUIMediaDevice	房间内媒体设备类型
TUIRole	房间内角色类型
TUIVideoQuality	视频质量
TUIAudioQuality	音频质量
TUIVideoStreamType	视频流类型
TUIChangeReason	更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）
TUICaptureSourceType	屏幕分享捕获源类型
TUIRequestAction	请求类型

TUICommonDefine

类型	说明
TUINetworkQuality	网络质量

常用结构

TUIRoomDefine

类型	说明
TUIRoomInfo	房间信息
TUILoginUserInfo	用户登录信息

版权所有：腾讯云计算（北京）有限责任公司

TUIUserInfo	房间内用户信息
TUISeatInfo	房间内座位信息
TUISeatLockParams	锁定麦位操作参数
TUIUserVoiceVolume	房间内用户音量
TUIRequest	信令请求
TUIShareTarget	屏幕分享采集源信息

TUICommonDefine

类型	说明
TUINetworkInfo	网络质量信息
TUIMessage	消息

TUIRoomType

房间类型

枚举	取值	描述
TUIRoomTypeConference	1	会议类型房间，适用于会议，教育场景，该房间中可以开启自由发言，申请发言、上麦发言等不同模式。
TUIRoomTypeLivingRoom	2	直播类型房间，适用于直播场景，该房间可以开启自由发言，麦位控制模式，该房间中麦位是有编号的。

TUISpeechMode

麦控模式

枚举	取值	描述
TUISpeechModeFreeToSpeak	1	自由发言模式。
TUISpeechModeApplyToSpeak	2	申请发言模式。（仅在会议类型房间下生效）
TUISpeechModeApplySpeakAfterTakingSeat	3	上麦发言模式。

版权所有：腾讯云计算（北京）有限责任公司

TUIMediaDevice

房间内媒体设备类型

枚举	取值	描述
TUIMediaDeviceMicrophone	1	麦克风
TUIMediaDeviceCamera	2	摄像头
TUIMediaDeviceApplyScreenSharing	3	屏幕共享

TUIRole

房间内角色类型

枚举	取值	描述
TUIRoleRoomOwner	0	房主，一般指房间的创建者，房间内最高权限拥有者
TUIRoleAdministrator	1	房间管理员
TUIRoleGeneralUser	2	房间内普通成员

TUIVideoQuality

视频质量

枚举	取值	描述
TUIVideoQuality360P	1	低清360P
TUIVideoQuality540P	2	标清540P
TUIVideoQuality720P	3	高清720P
TUIVideoQuality1080P	4	超清1080P

TUIAudioQuality

音频质量

枚举	取值	描述
TUIAudioQualitySpeech	0	人声模式
TUIAudioQualityDefault	1	默认模式

版权所有：腾讯云计算（北京）有限责任公司

TUIAudioQualityMusic	2	音乐模式

TUIVideoStreamType

视频流类型

枚举	取值	描述
TUIVideoStreamTypeCameraStream	0	高清摄像头视频流
TUIVideoStreamTypeScreenStream	1	屏幕分享视频流
TUIVideoStreamTypeCameraStreamLow	2	低清摄像头视频流

TUIChangeReason

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

枚举	取值	描述
TUIChangeReasonBySelf	0	自己操作
TUIChangeReasonByAdmin	1	房主或管理员操作

TUICaptureSourceType

屏幕分享捕获源类型

枚举	取值	描述
TUICaptureSourceTypeUnknown	-1	未定义
TUICaptureSourceTypeWindow	0	窗口
TUICaptureSourceTypeScreen	1	屏幕

TUIRequestAction

请求类型

枚举	取值	描述
TUIRequestActionInvalidAction	0	无效请求
TUIRequestActionOpenRemoteCamera	1	请求远端用户打开摄像头
TUIRequestActionOpenRemoteMicrophone	2	请求远端用户打开麦克风
TUIRequestActionConnectOtherRoom	3	请求连接到其他房间

版权所有：腾讯云计算（北京）有限责任公司

TUIRequestActionTakeSeat	4	请求上麦
TUIRequestActionRemoteUserOnSeat	5	请求远端用户上麦
TUIRequestActionApplyToAdminToOpenLocalCamera	6	向管理员请求打开本地摄像头
TUIRequestActionApplyToAdminToOpenLocalMicrophone	7	向管理员请求打开本地麦克风

TUINetworkQuality

网络质量

枚举	取值	描述
TUINetworkQualityUnknown	0	未定义
TUINetworkQualityExcellent	1	当前网络非常好
TUINetworkQualityGood	2	当前网络比较好
TUINetworkQualityPoor	3	当前网络一般
TUINetworkQualityBad	4	当前网络较差
TUINetworkQualityVeryBad	5	当前网络很差
TUINetworkQualityDown	6	当前网络不满足 TRTC 的最低要求

TUIRoomInfo

房间信息

字段	类型	描述
roomId	NSString *	房间 ID
roomType	TUIRoomType	房间类型
ownerId	NSString *	主持人 ID，默认为房间创建者（只读）
name	NSString *	房间名称,默认为房间ID
speechMode	TUISpeechMode	麦控模式

版权所有：腾讯云计算（北京）有限责任公司

createTime	NSUInteger	房间创建时间（只读）
memberCount	NSInteger	房间内成员数量（只读）
maxSeatCount	NSUInteger	最大麦位数量（仅支持进房前和创建房间时设置）
isCameraDisableForAllUser	BOOL	是否禁止打开摄像头（创建房间可选参数），默认值：false
isMicrophoneDisableForAllUser	BOOL	是否禁止打开麦克风（创建房间可选参数），默认值：false
isMessageDisableForAllUser	BOOL	是否禁止发送消息（创建房间可选参数），默认值：false
enableCDNStreaming	BOOL	是否开启CDN直播（创建房间可选参数，直播房间使用），默认值：false
cdnStreamDomain	NSString*	直播推流域名（创建房间可选参数，直播房间使用），默认值：空

TUILoginUserInfo

用户登录信息

字段	类型	含义
userId	NSString *	用户 ID
userName	NSString *	用户名称
avatarUrl	NSString *	用户头像URL

TUIUserInfo

房间内用户信息

字段	类型	描述
userId	NSString *	用户 ID
userName	NSString *	用户名称
avatarUrl	NSString *	用户头像URL
userRole	TUIRole	用户角色类型

版权所有：腾讯云计算（北京）有限责任公司

hasAudioStream	BOOL	是否有音频流，默认值：false
hasVideoStream	BOOL	是否有视频流，默认值：false
hasScreenStream	BOOL	是否有屏幕分享流，默认值：false

TUISeatInfo

房间内座位信息

字段	类型	描述
index	NSInteger	麦位序号
userId	NSString *	用户 ID
isLocked	BOOL	麦位是否被锁定，默认 false
isVideoLocked	BOOL	麦位是否被禁止打开摄像头，默认 false
isAudioLocked	BOOL	麦位是否被禁止打开麦克风，默认 false

TUISeatLockParams

锁定麦位操作参数

字段	类型	含义
lockSeat	BOOL	锁定麦位，默认 false
lockVideo	BOOL	锁定麦位摄像头，默认 false
lockAudio	BOOL	锁定麦位麦克风，默认 false

TUIUserVoiceVolume

房间内用户音量

字段	类型	描述
userId	NSString *	用户 ID
volume	NSUInteger	音量 用于承载所有正在说话的用户的音量大小，取值范围 0 - 100

版权所有：腾讯云计算（北京）有限责任公司

TUIRequest

信令请求

字段	类型	描述
requestId	NSString *	请求 ID
requestAction	TUIRequestAction	请求类型
userId	NSString *	用户 ID
content	NSString *	信令内容
timestamp	NSUInteger	时间戳

TUIShareTarget

屏幕分享采集源信息

字段	类型	描述
targetId	NSString *	采集源的 ID，对于窗口，该字段代表窗口的 ID；对于屏幕，该字段代表显示器的 ID
sourceType	TUICaptureSourceType	采集源类型
sourceName	NSString *	采集源名称
thumbnailImage	TUIImage *	缩略图
iconImage	TUIImage *	图标
extInfo	NSDictionary *	窗口的扩展信息

TUINetworkInfo

网络质量信息

字段	类型	描述
userId	NSString *	用户 ID
quality	TUINetworkQuality	网络质量
upLoss	uint32_t	上行丢包率
downLoss	uint32_t	下行丢包率

版权所有：腾讯云计算（北京）有限责任公司

delay	uint32_t	网络延迟

TUIMessage

消息

字段	类型	描述
messageId	NSString *	消息 ID
message	NSString *	消息文本
timestamp	uint64_t	消息时间
userId	NSString *	消息发送者
userName	NSString *	消息发送者昵称
avatarUrl	NSString *	消息发送者头像

Android
API 概览

最近更新时间：2023-11-16 16:14:59

TUIRoomEngine (无 UI 接口)

TUIRoomEngine API 是音视频通话组件的**无 UI 接口,**您可以使用这套 API 根据您的业务需求自定义封装。
TUIRoomEngine

TUIRoomEngine 核心方法

API	描述
createInstance	创建 TUIRoomEngine 实例
destroyInstance	销毁 TUIRoomEngine 实例
login	登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。
logout	退出登录接口，会有主动离开房间操作、销毁资源
setSelfInfo	设置本地用户名称和头像
getSelfInfo	获取本地用户基本信息
addObserver	设置事件回调
removeObserver	移除事件回调

房间相关主动接口

API	描述
createRoom	创建房间
destroyRoom	解散房间
enterRoom	进入房间
exitRoom	离开房间
connectOtherRoom	连接其他房间

版权所有：腾讯云计算（北京）有限责任公司

disconnectOtherRoom	断开与其他房间的连接
fetchRoomInfo	获取房间信息
updateRoomNameByAdmin	更新房间名称
updateRoomSpeechModeByAdmin	设置房间管理模式（只有管理员或群主能够调用）

本地用户视图渲染、视频管理

API	描述
setLocalVideoView	设置本地用户视频渲染的视图控件
openLocalCamera	打开本地摄像头
closeLocalCamera	关闭本地摄像头
updateVideoQuality	更新本地视频编码质量设置
startScreenSharing	开始屏幕分享
stopScreenSharing	结束屏幕分享
startPushLocalVideo	开始推送本地视频
stopPushLocalVideo	停止推送本地视频

本地用户音频管理

API	描述
openLocalMicrophone	打开本地麦克风
closeLocalMicrophone	关闭本地麦克风
updateAudioQuality	更新本地音频编码质量设置
startPushLocalAudio	开始推送本地音频
stopPushLocalAudio	停止推送本地音频

远端用户视图渲染、视频管理

API	描述
setRemoteVideoView	设置远端用户视频渲染的视图控件

版权所有：腾讯云计算（北京）有限责任公司

startPlayRemoteVideo	开始播放远端用户视频
stopPlayRemoteVideo	停止播放远端用户视频
muteRemoteAudioStream	将远端用户禁音

房间内用户信息

API	描述
getUserList	获取房间内的成员列表
getUserInfo	获取成员信息

房间内用户管理

API	描述
changeUserRole	修改用户角色（只有管理员或群主能够调用）
kickRemoteUserOutOfRoom	将远端用户踢出房间（只有管理员或群主能够调用）

房间内用户发言管理

API	描述
disableDeviceForAllUserByAdmin	全体用户媒体设备管理（只有管理员或群主能够调用）
openRemoteDeviceByAdmin	请求远端用户打开媒体设备（只有管理员或群主能够调用）
closeRemoteDeviceByAdmin	关闭远端用户媒体设备（只有管理员或群主能够调用）
applyToAdminToOpenLocalDevice	请求打开本地媒体设备（普通用户可用）

房间内麦位管理

API	描述
setMaxSeatCount	设置最大麦位数（仅支持进房前和创建房间时设置）
getSeatList	获取麦位列表
lockSeatByAdmin	锁定麦位（包括位置锁定，音频状态锁定，视频状态锁定）
takeSeat	本地上麦

版权所有：腾讯云计算（北京）有限责任公司

	会议场景：SPEAK_AFTER_TAKING_SEAT 模式需要向主持人或管理员发起申请才允许上麦，其他模式不支持上麦。 直播场景：FREE_TO_SPEAK 模式可以自由上麦，上麦后开麦发言；SPEAK_AFTER_TAKING_SEAT 模式需要向主持人或管理员发起申请才允许上麦；其他模式不支持上麦
leaveSeat	本地下麦
takeUserOnSeatByAdmin	主持人/管理员 邀请用户上麦
kickUserOffSeatByAdmin	主持人/管理员 将用户下麦

信令管理

API	描述
cancelRequest	取消请求
responseRemoteRequest	回复请求

发送消息

API	描述
sendTextMessage	发送本文消息
sendCustomMessage	发送自定义消息
disableSendingMessageByAdmin	禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）
disableSendingMessageForAllUser	禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）

高级功能：获取 TRTC 实例

API	描述
getTRTCCloud	获得 TRTC 实例对象
getDeviceManager	获得设备管理对象
getAudioEffectManager	获得音效管理对象
getBeautyManager	获得美颜管理对象

版权所有：腾讯云计算（北京）有限责任公司

事件类型定义

TUIRoomObserver 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己需要的回调事件。
TUIRoomObserver

错误回调

事件	描述
onError	错误回调事件

登录状态事件回调

API	描述
onKickedOffLine	用户被踢下线事件
onUserSigExpired	用户凭证超时事件

房间内事件回调

API	描述
onRoomNameChanged	房间名称更改事件
onAllUserMicrophoneDisableChanged	房间内所有用户麦克风被禁用事件
onAllUserCameraDisableChanged	房间内所有用户摄像头被禁用事件
onSendMessageForAllUserDisableChanged	房间内所有用户发送文本消息被禁用事件
onRoomDismissed	房间被解散事件
onKickedOutOfRoom	被踢出房间事件
onRoomSpeechModeChanged	房间麦控模式发生变化

房间内用户事件回调

API	描述
onRemoteUserEnterRoom	远端用户进房事件
onRemoteUserLeaveRoom	远端用户离开房间事件

版权所有：腾讯云计算（北京）有限责任公司

onUserRoleChanged	用户角色发生变化事件
onUserVideoStateChanged	用户视频状态发生变化事件
onUserAudioStateChanged	用户音频状态发生变化事件
onUserVoiceVolumeChanged	用户音量变化事件
onSendMessageForUserDisableChanged	用户文本消息发送能力发生变化事件
onUserNetworkQualityChanged	用户网络状态变化事件
onUserScreenCaptureStopped	屏幕分享结束事件

房间内麦位事件回调

API	描述
onRoomMaxSeatCountChanged	房间内最大麦位数发生变化事件（仅在会议类型房间生效）
onSeatListChanged	麦位列表发生变化事件
onKickedOffSeat	收到用户被踢下麦事件

请求信令事件回调

API	描述
onRequestReceived	收到请求消息事件
onRequestCancelled	收到请求被取消的事件

房间内消息事件回调

API	描述
onReceiveTextMessage	收到普通文本消息事件
onReceiveCustomMessage	收到自定义消息事件

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomKit

最近更新时间：2024-03-29 17:49:35

Module: TUIRoomKit

Function: 多人音视频主功能接口

Version: 2.1.0

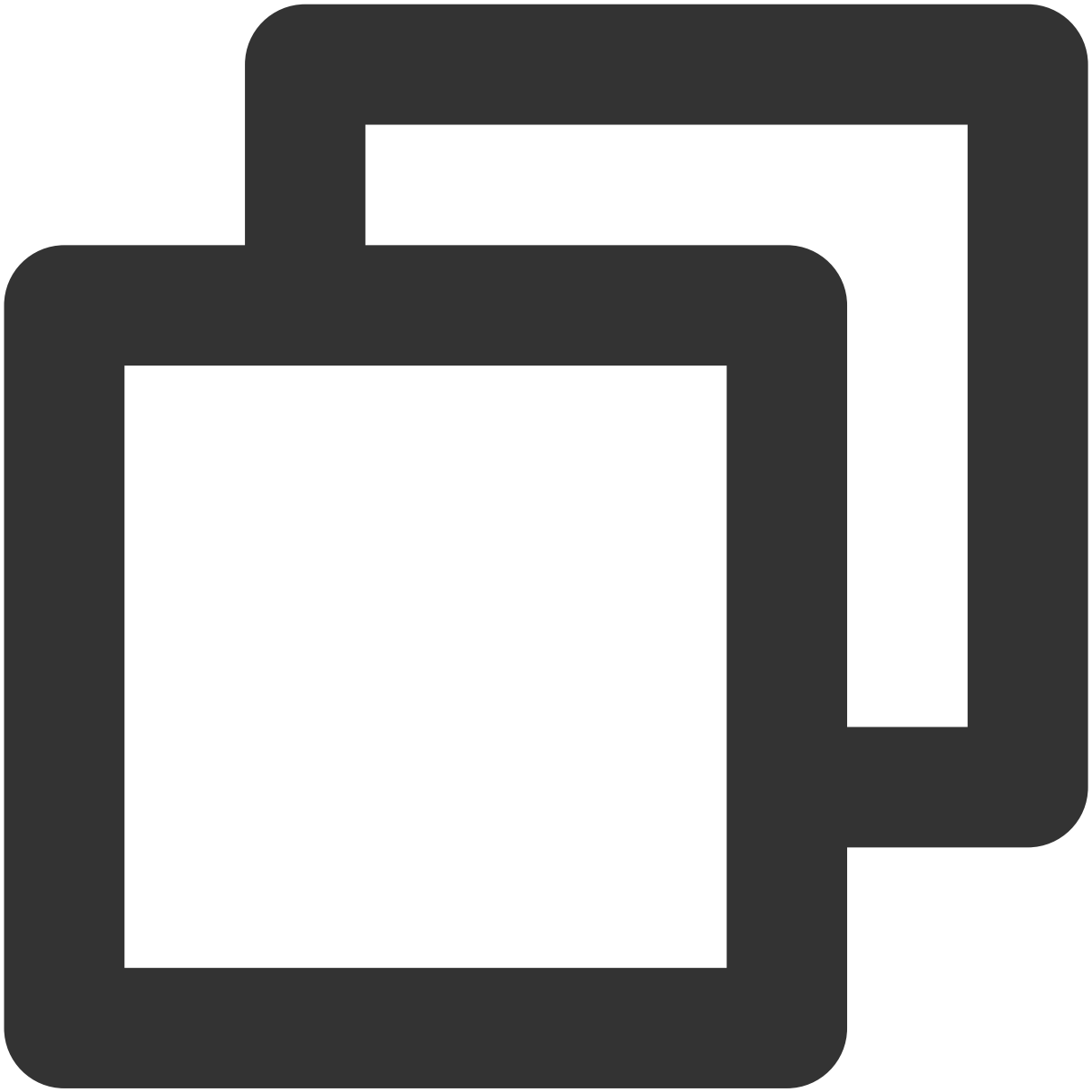
TUIRoomKit

TUIRoomKit

函数列表	描述
createInstance	创建 TUIRoomKit 实例（单例模式）。
destroyInstance	销毁 TUIRoomKit 实例。
setSelfInfo	设置个人信息，包括用户名和头像。
createRoom	创建房间。
enterRoom	进入房间。

createInstance

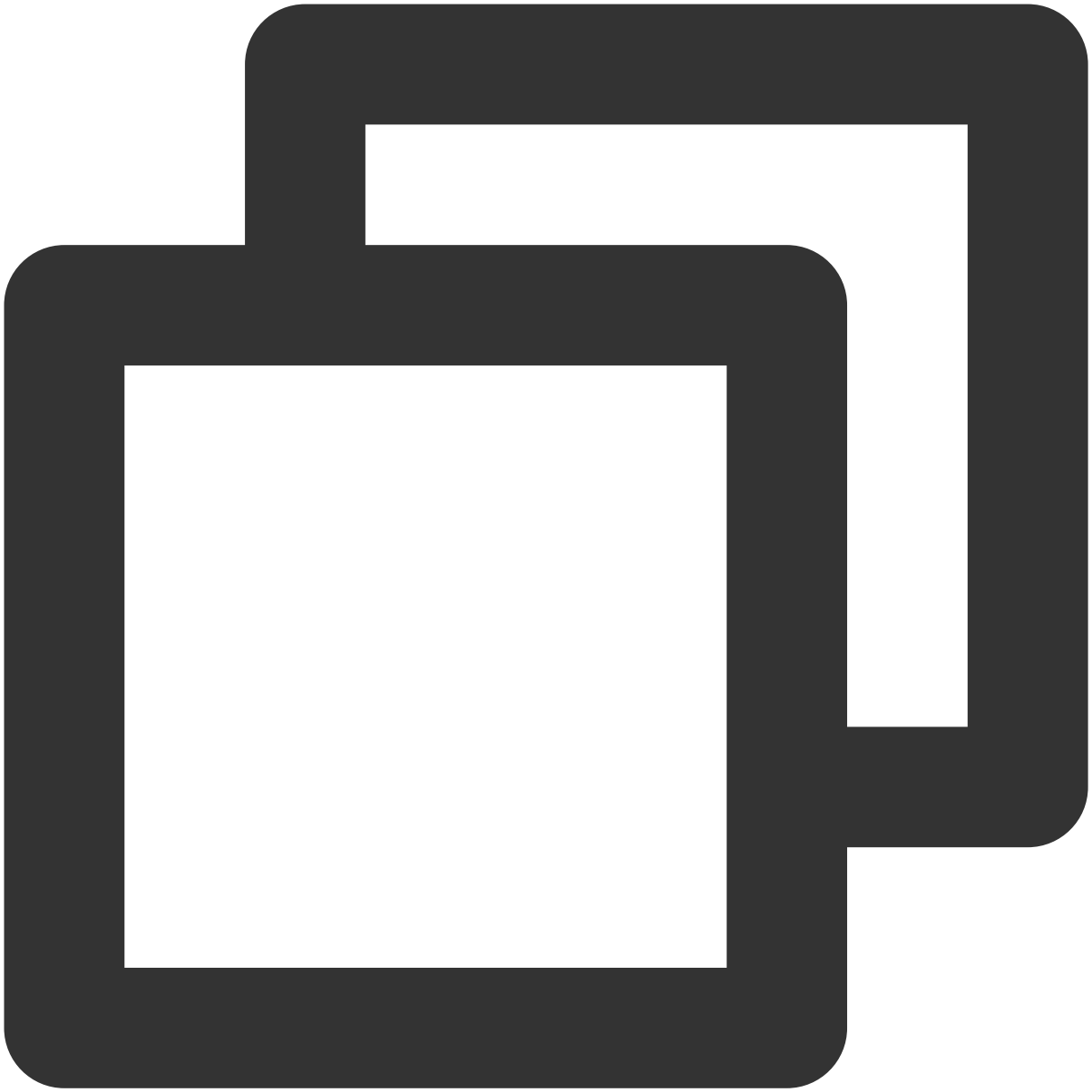
初始化 TUIRoomKit 单例对象。



```
public static TUIRoomKit createInstance();
```

destroyInstance

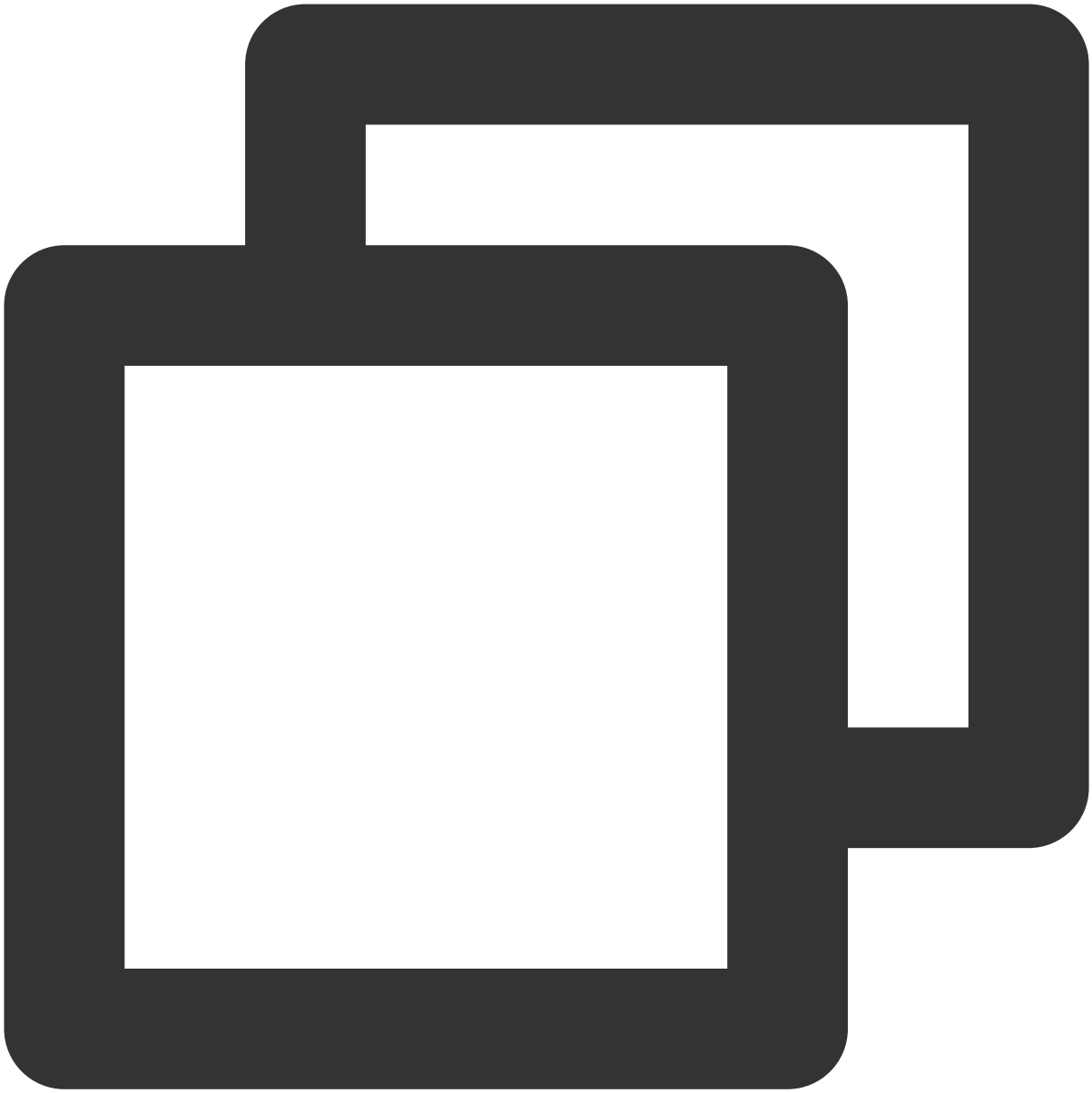
销毁 TUIRoomKit 实例。



```
public static void destroyInstance();
```

setSelfInfo

设置个人信息，包括用户名和头像。

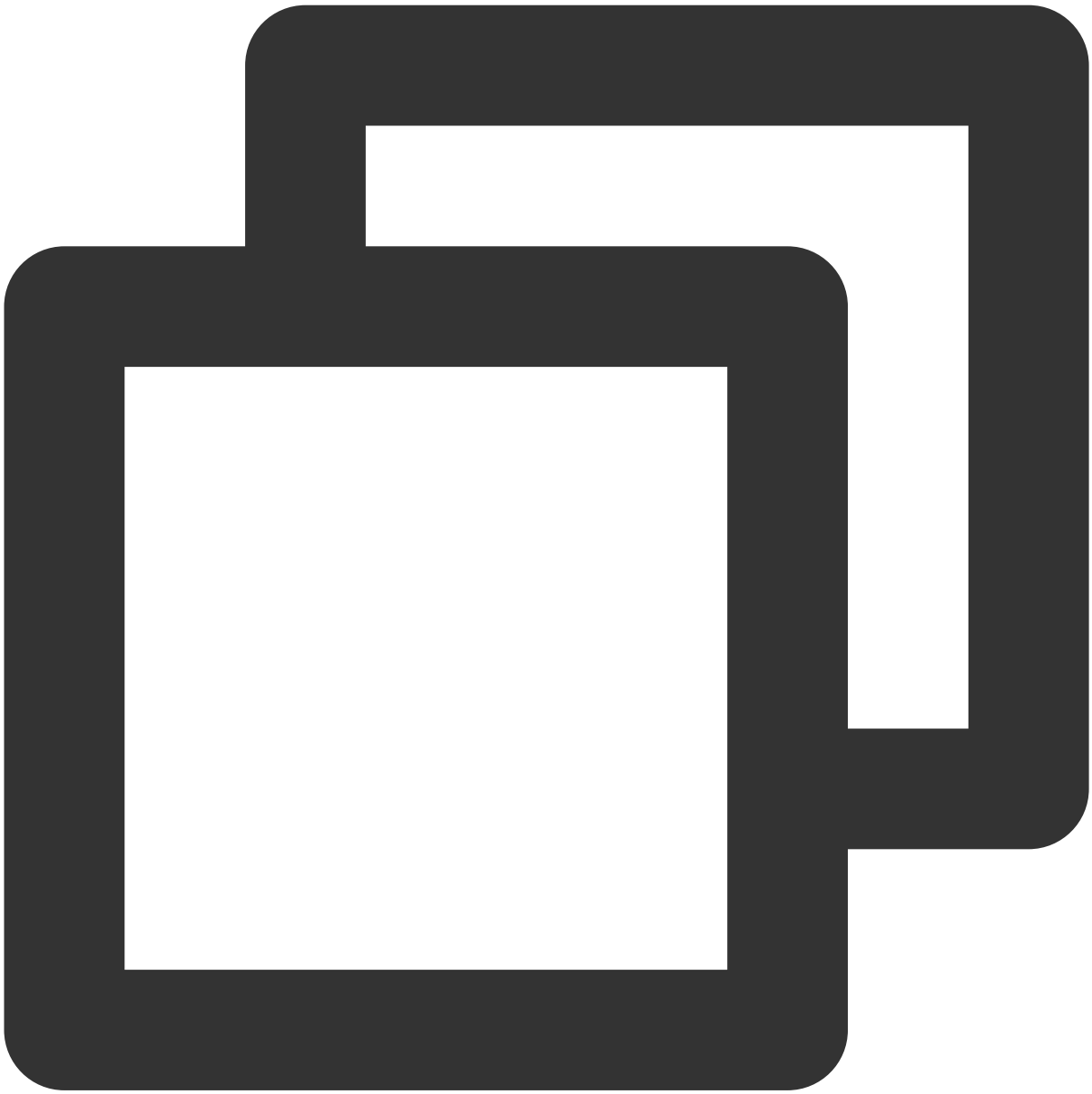


```
public abstract void setSelfInfo(String userName, String avatarURL,
                                 TUIRoomDefine.ActionCallback callback);
```

参数	描述
userName	个人的用户名。
avatarURL	个人的头像链接。
callback	设置个人信息是否成功的回调。

createRoom

创建房间。

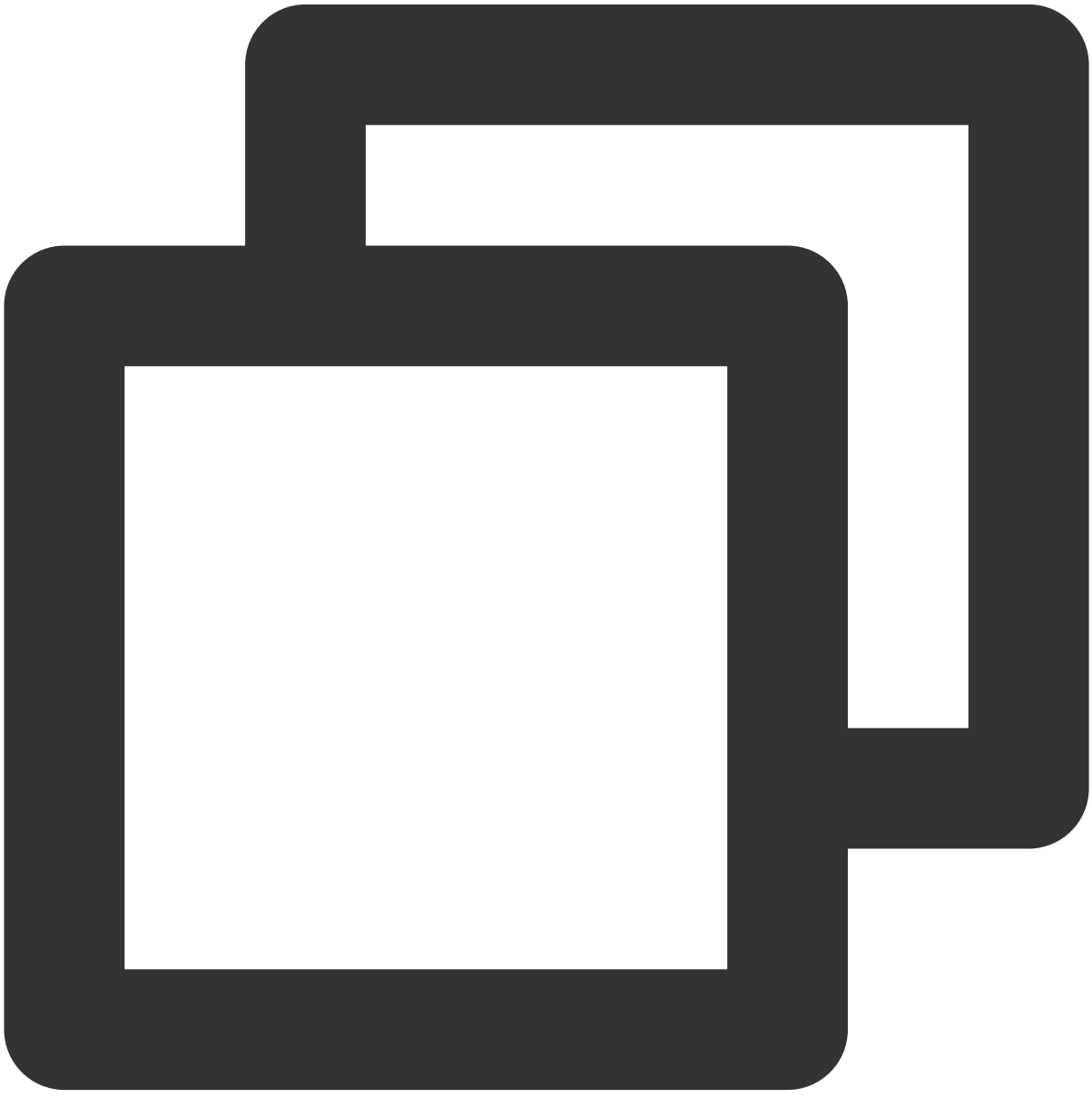


```
public abstract void createRoom(TUIRoomDefine.RoomInfo roomInfo, TUIRoomDefine.Acti
```

参数	描述
roomInfo	创建房间的参数，包含房间号、房间名称等，其中 roomId 是必填项，其余可为默认值。
callback	创建房间是否成功的回调。

enterRoom

进入房间。



```
public abstract void enterRoom(String roomId,
                               boolean enableAudio,
                               boolean enableVideo,
```

```
                                boolean isSoundOnSpeaker,
                                TUIRoomDefine.GetRoomInfoCallback callback);
```

参数	描述
roomId	进入房间的房间号。
enableAudio	true 进房打开麦克风，并推送本地音频数据到远端，其他成员可以正常听到本地声音； false 进房只打开麦克风，不推送本地音频数据到远端，其他成员无法听到本地声音。
enableVideo	true 进房打开摄像头，并推送本地视频数据到远端，其他成员可以正常看到本地画面； false 进房不打开摄像头，也不推送本地视频数据到远端，其他成员无法看到本地画面。
isSoundOnSpeaker	是否使用扬声器播放声音，true 则使用扬声器，false 则使用听筒。
callback	进入房间是否成功的回调。

TUIRoomEngine

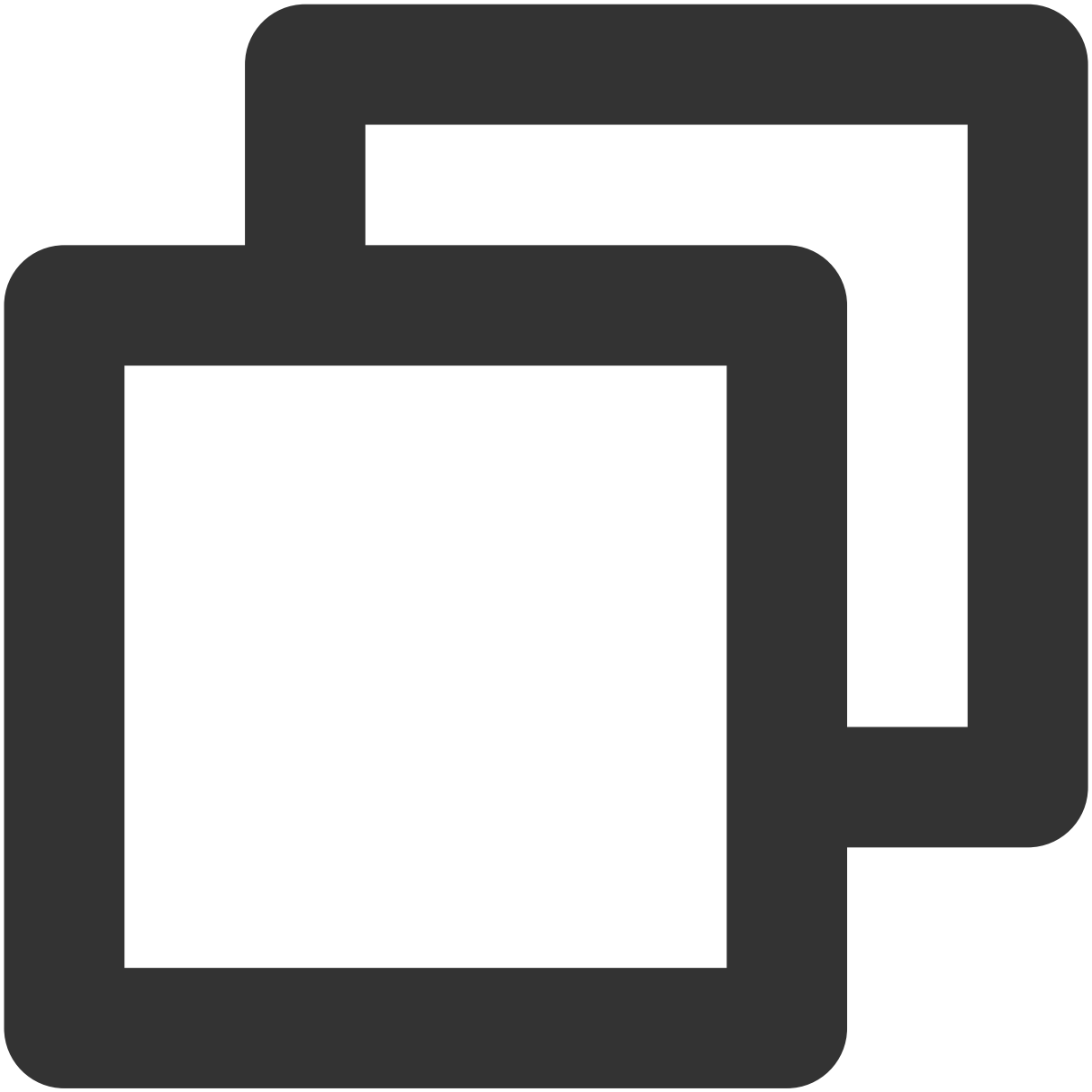
最近更新时间：2023-11-14 09:50:56

TUIRoomEngine (无 UI 接口)

TUIRoomEngine API 是音视频通话组件的**无 UI 接口,**您可以使用这套 API 根据您的业务需求自定义封装。

createInstance

创建 TUIRoomEngine 实例

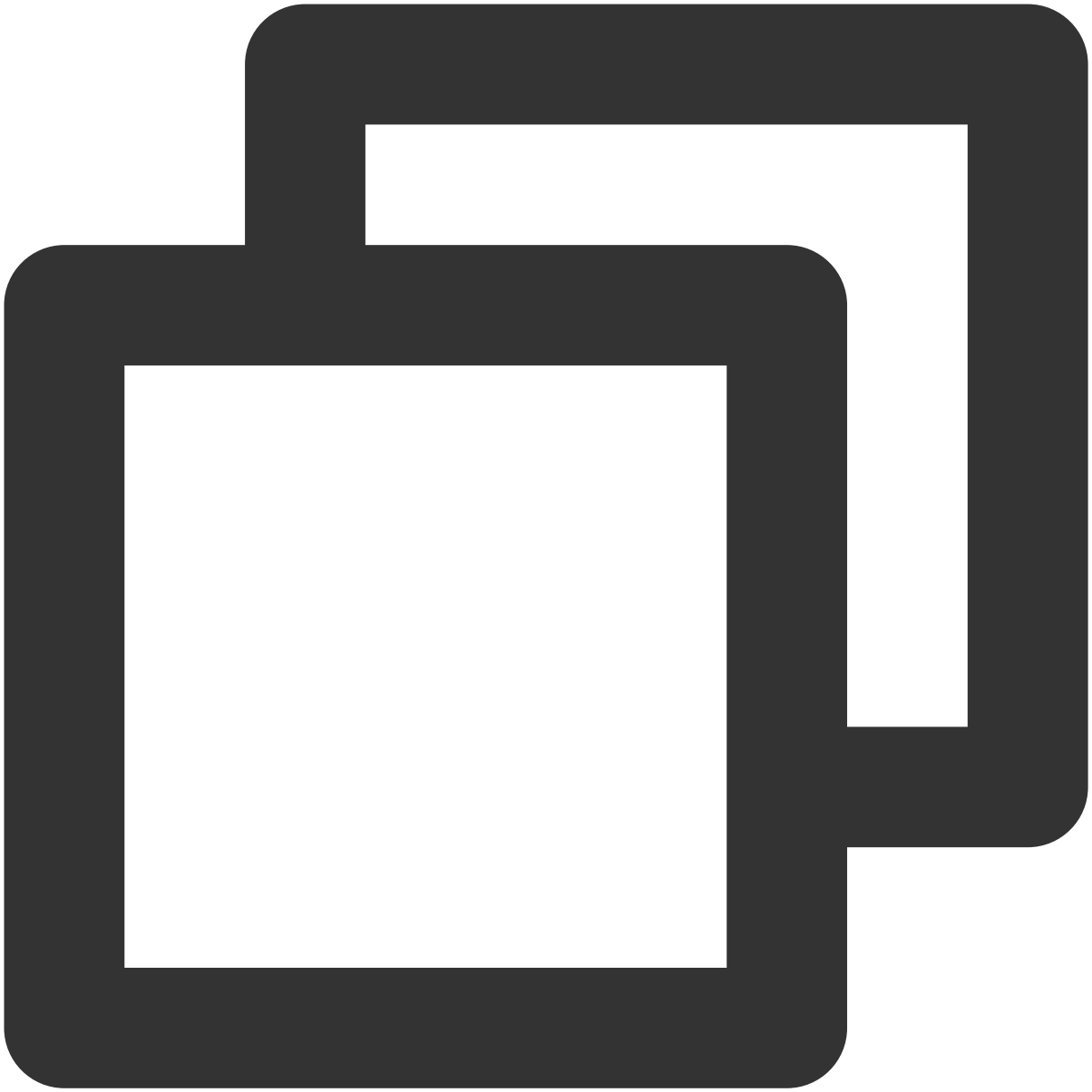


```
static TUIRoomEngine createInstance()
```

return:TUIRoomEngine 实例

destroyInstance

销毁TUIRoomEngine实例

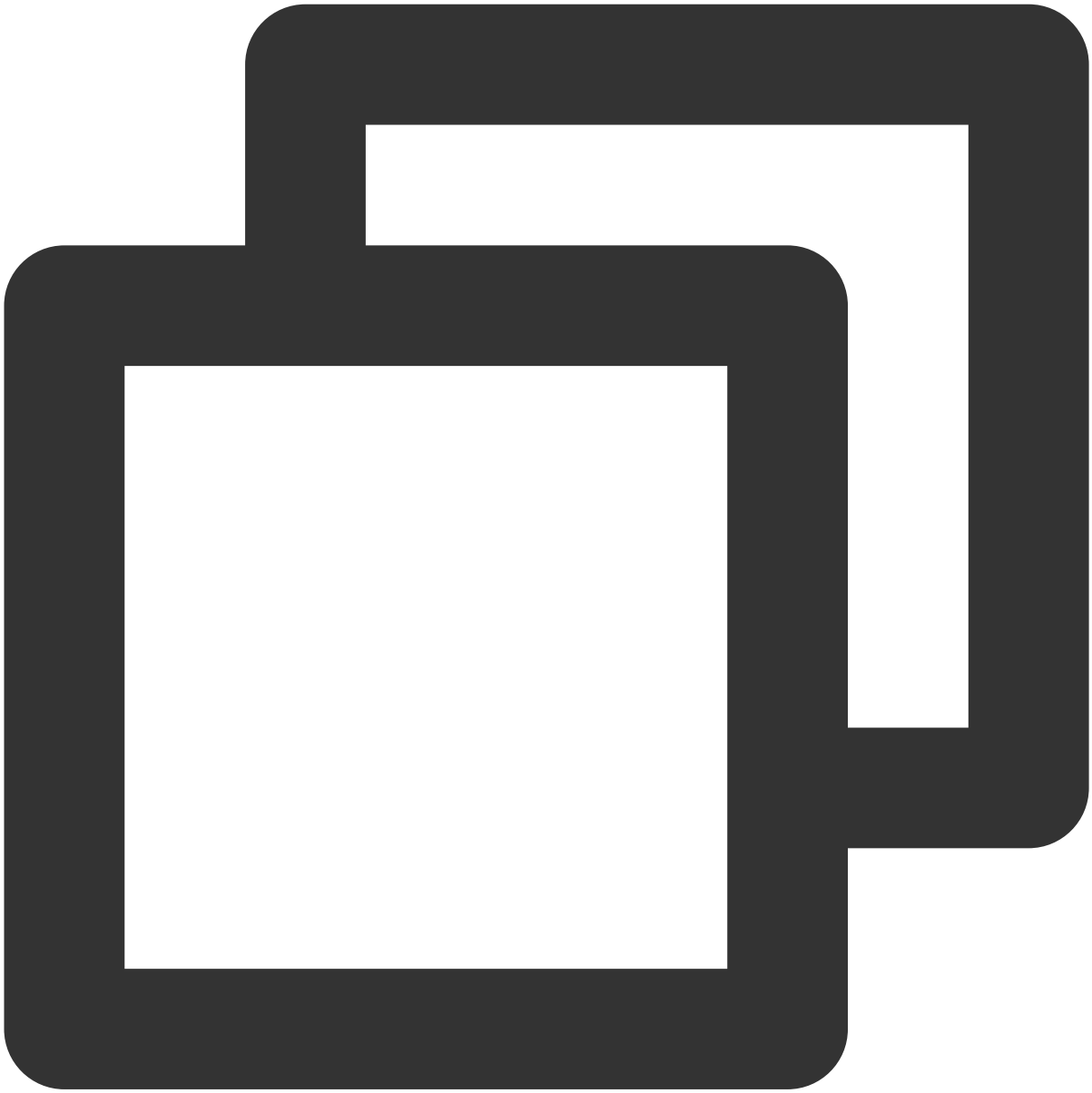


```
void destroyInstance()
```

login

登录 roomEngine 接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作

说明：v1.0.0 该接口命名为 setup，v1.0.1 及其以上版本请使用 TUIRoomEnine.login 登录 TUIRoomEngine。



```
void static login(Context context,
                  int sdkAppId,
                  String userId,
                  String userSig,
                  TUIRoomDefine.ActionCallback callback)
```

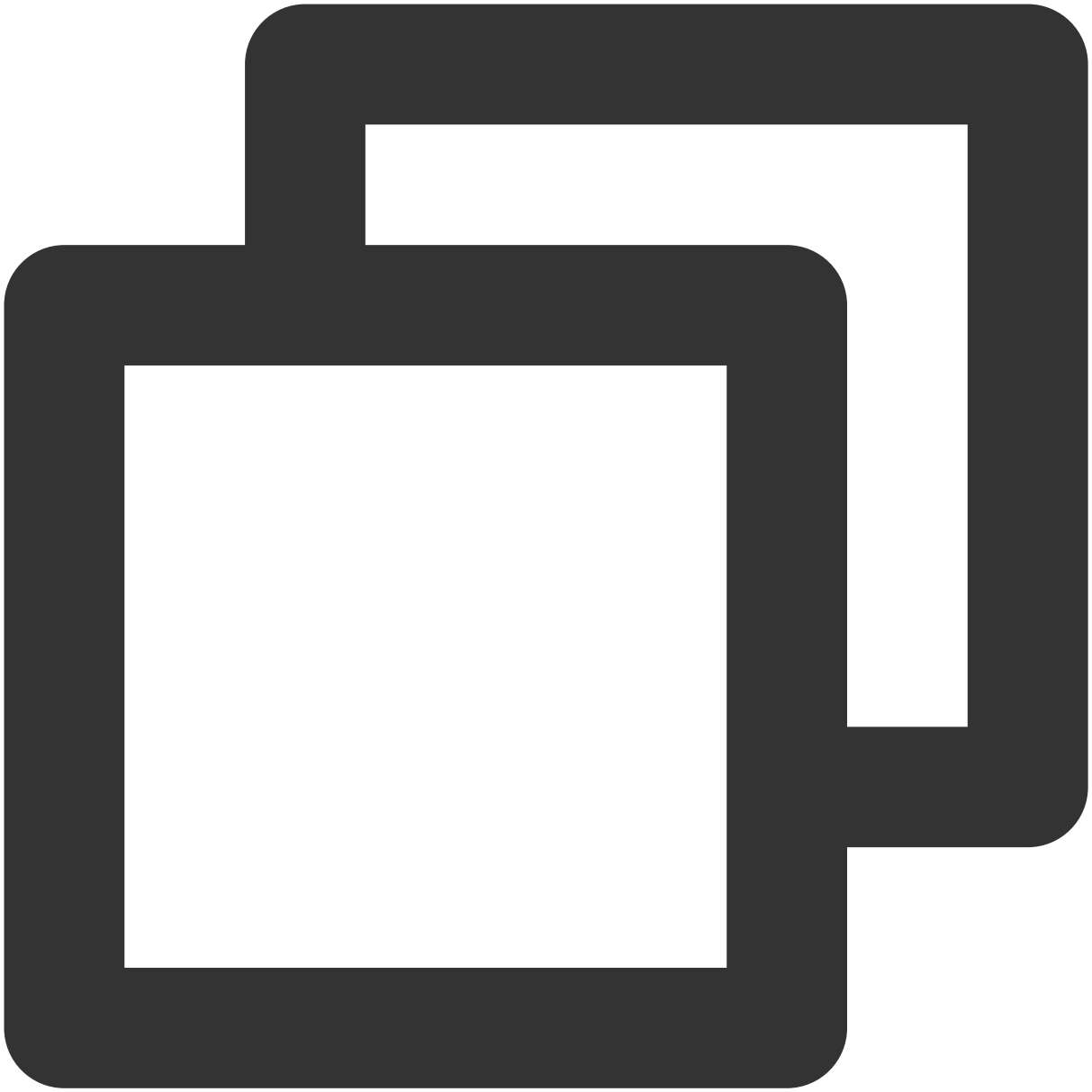
参数：

参数	类型	含义
context	Context	Android上下文

sdkAppId	int	应用信息中获取 sdkAppId 信息
userId	String	用户 ID
userSig	String	userSig 签名 计算 userSig 的方式请参考UserSig 相关
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

logout

退出登录接口，会有主动离开房间操作、销毁资源



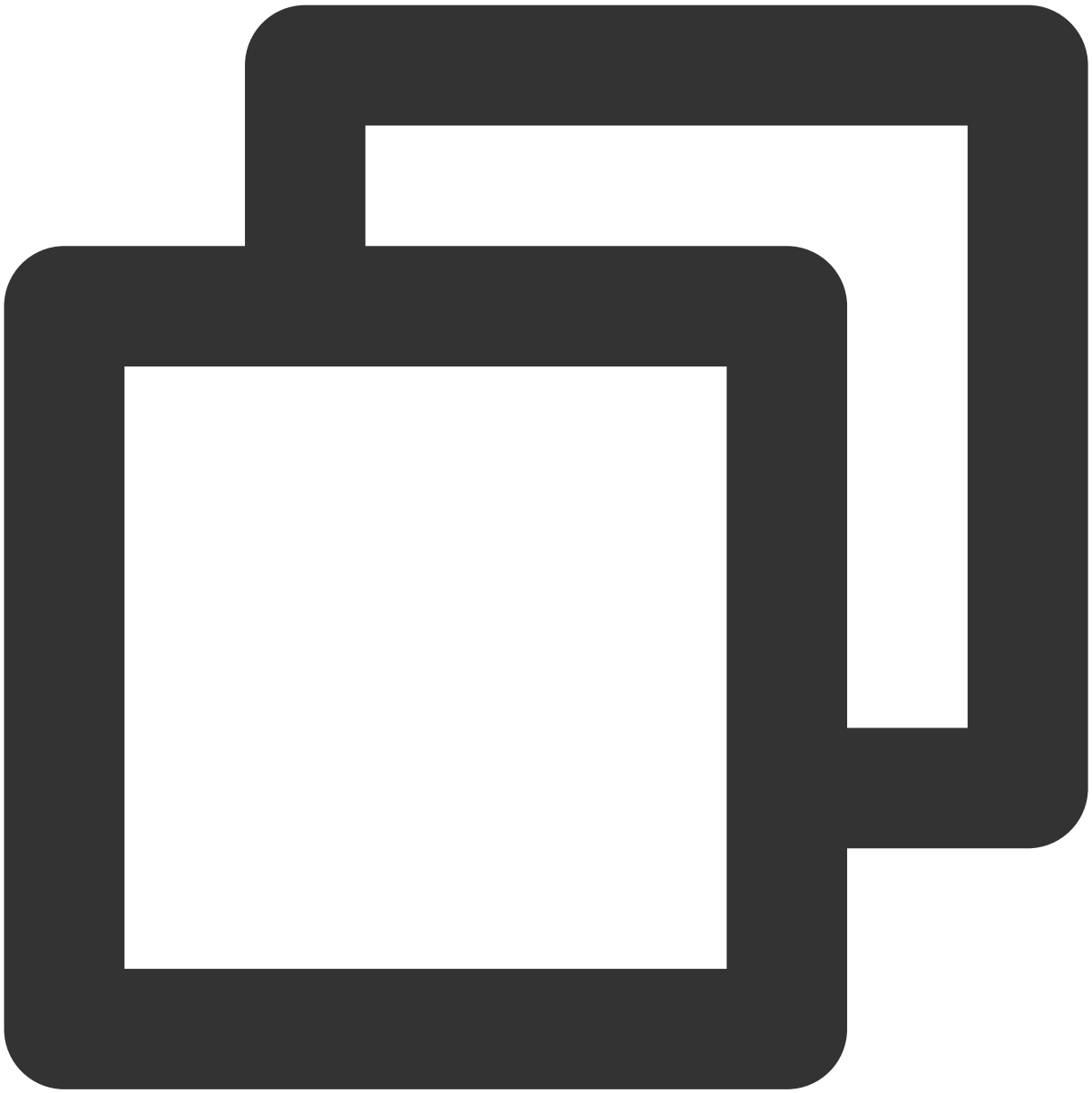
```
static void logout(TUIRoomDefine.ActionCallback callback)
```

参数：

参数	类型	含义
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

setSelfInfo

设置本地用户名称和头像



```
static void setSelfInfo(String userName, String avatarURL, TUIRoomObserver.ActionCa
```

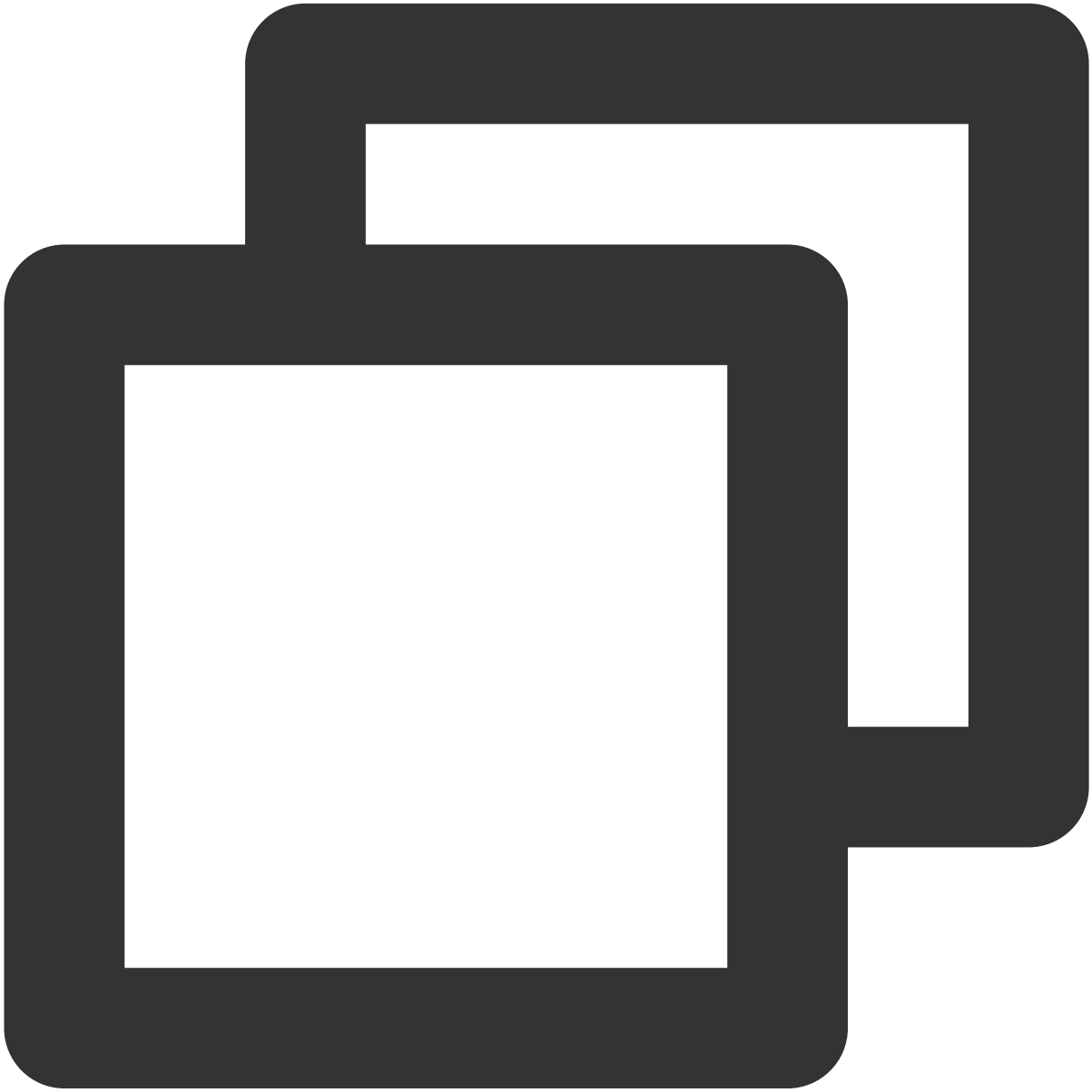
参数：

参数	类型	含义
userName	int	用户名
avatarUrl	String	用户头像

callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

getSelfInfo

获取本地用户登录的基本信息

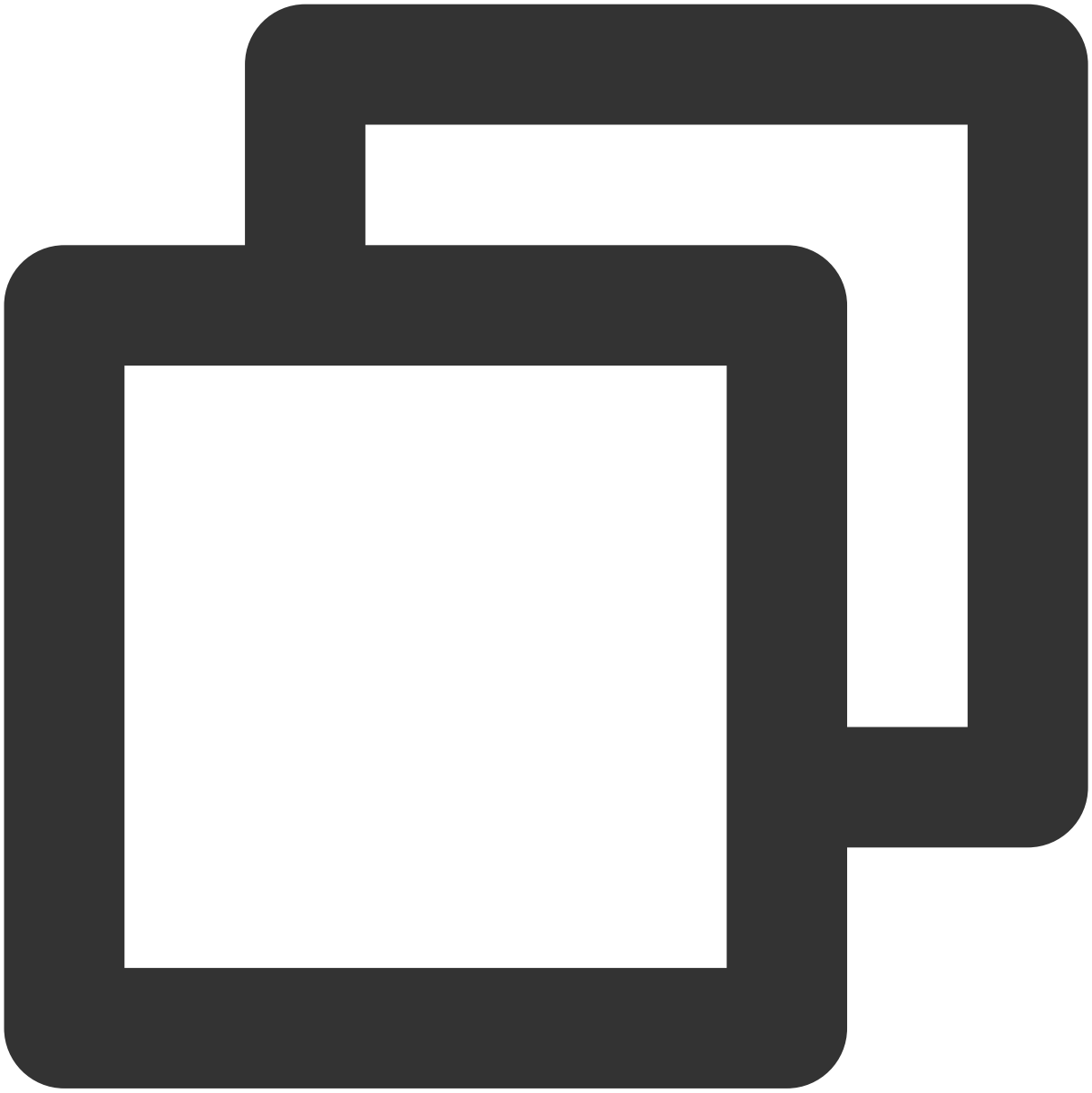


```
static TUIRoomDefine.LoginUserInfo getSelfInfo()
```

return:用户登录信息

addObserver

添加TUIRoomEngine事件回调



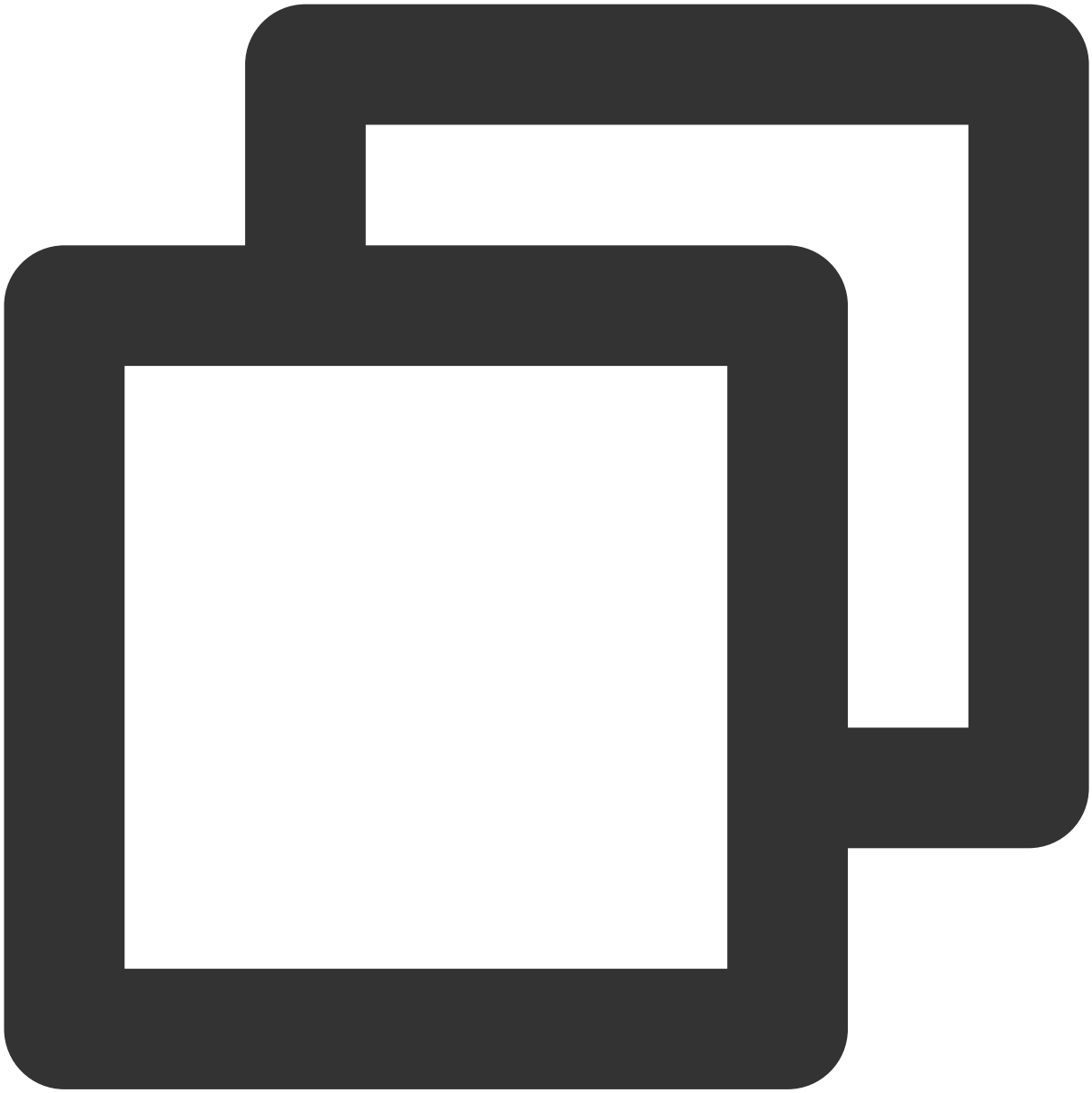
```
void addObserver(TUIRoomObserver observer)
```

参数：

参数	类型	含义
observer	TUIRoomObserver	TUIRoomEngine 事件回调

removeObserver

移除TUIRoomEngine事件回调

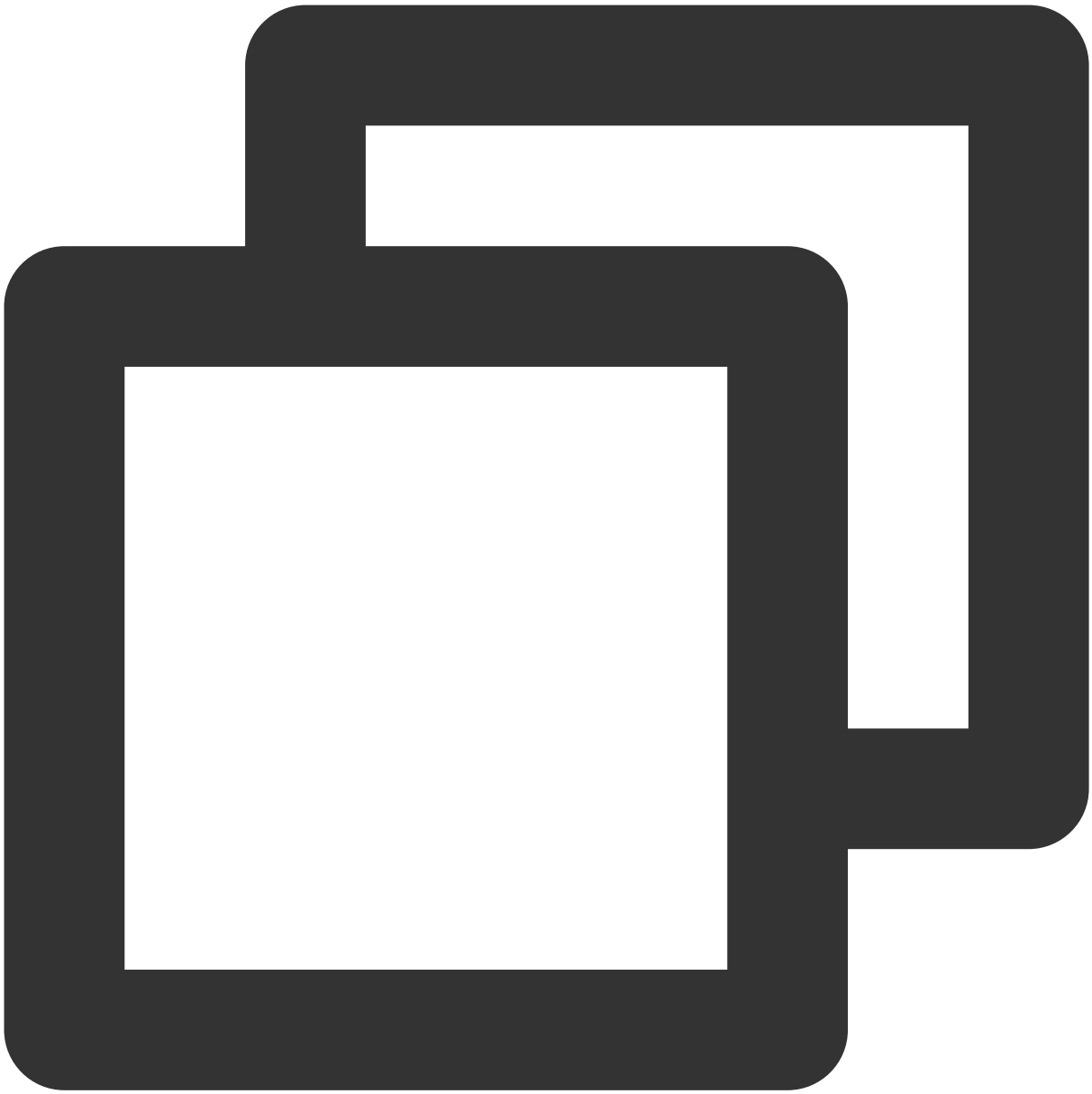


```
void removeObserver(TUIRoomObserver observer)
```

参数	类型	含义
observer	TUIRoomObserver	TUIRoomEngine 事件回调

createRoom

主持人创建房间，调用 createRoom 的用户为房间的所有者。创建房间时可设置房间 ID，房间名称，以及房间是否允许加入用户开启音视频，发送消息等功能。



```
void createRoom(TUIRoomDefine.RoomInfo roomInfo, TUIRoomDefine.ActionCallback callb
```

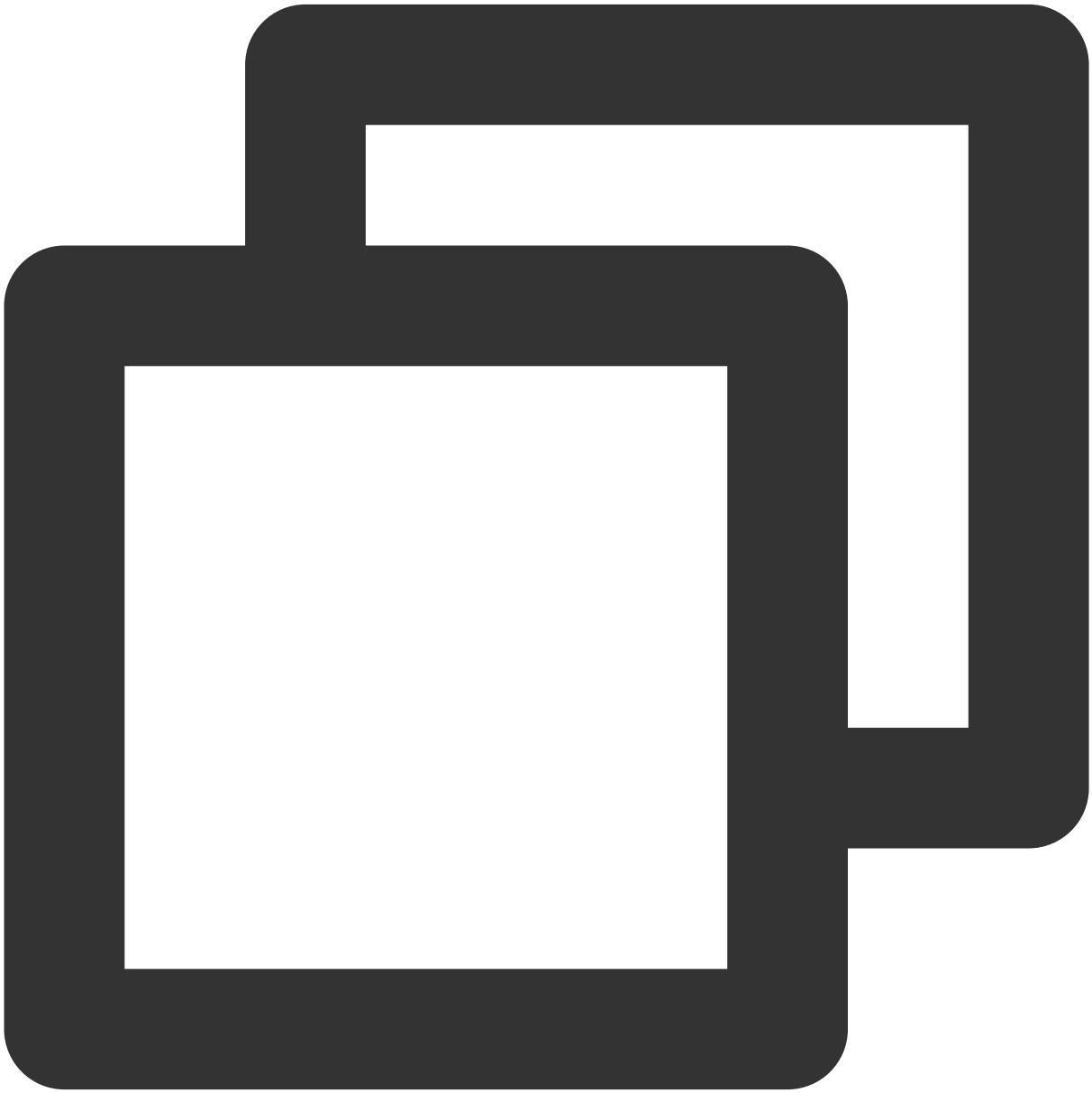
参数：

参数	类型	含义
roomInfo	TUIRoomDefine.RoomInfo	房间基本信息
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或

		者失败

destroyRoom

销毁房间接口，销毁房间必须由房间所有者发起，销毁房间之后房间不可进入。



```
void destroyRoom(TUIRoomDefine.ActionCallback callback)
```

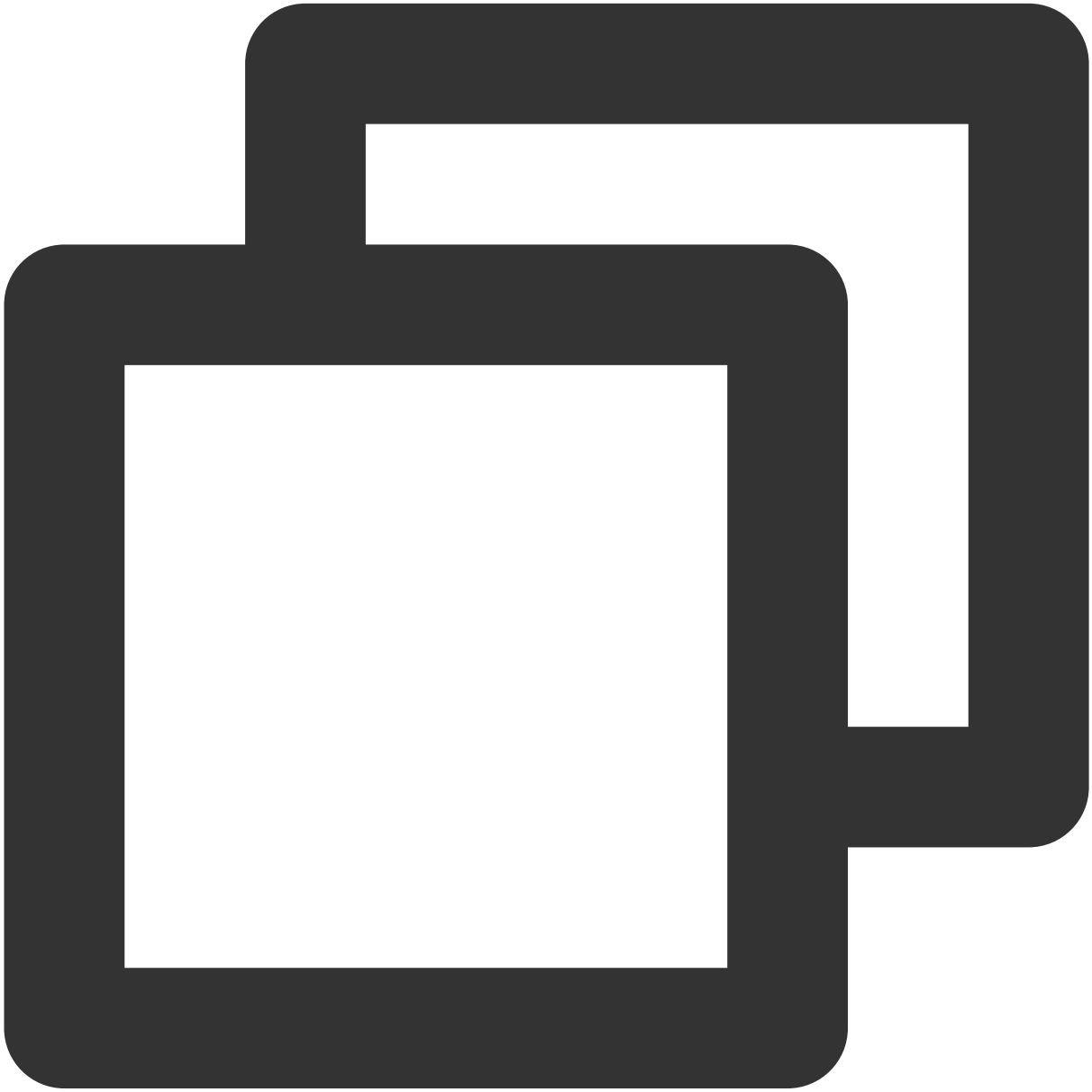
参数：

参数	类型	含义

callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

enterRoom

进入房间接口



```
void enterRoom(String roomId, TUIRoomDefine.GetRoomInfoCallback callback)
```

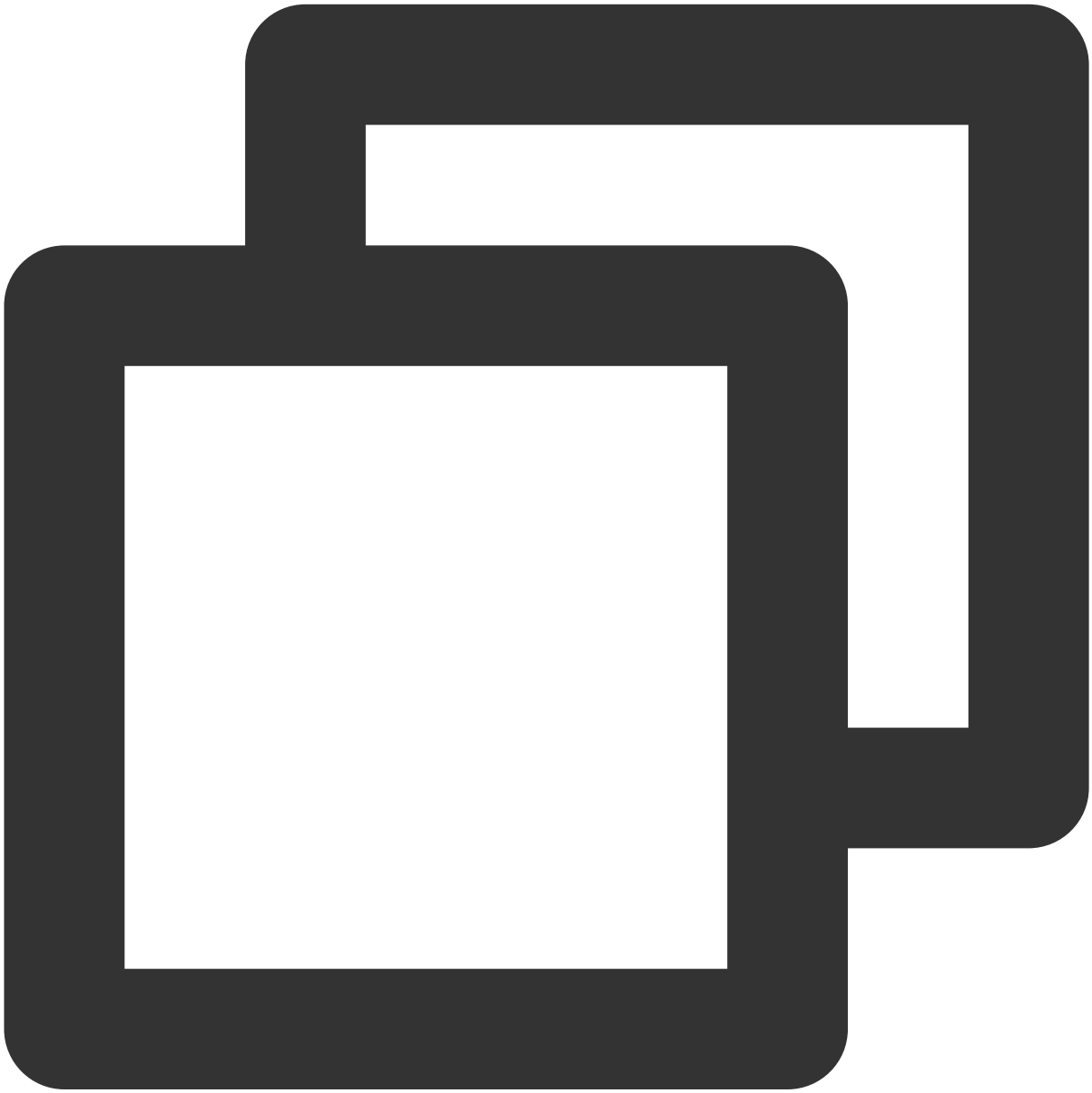
参数：

参数	类型	含义

roomId	String	房间号，字符串类型
callback	TUIRoomDefine.GetRoomInfoCallback	获取所进入的房间信息回调

exitRoom

离开房间接口，用户在执行 enterRoom 之后可通过 exitRoom 离开房间。



```
void exitRoom(boolean isSyncWaiting, TUIRoomDefine.ActionCallback callback);
```

参数：

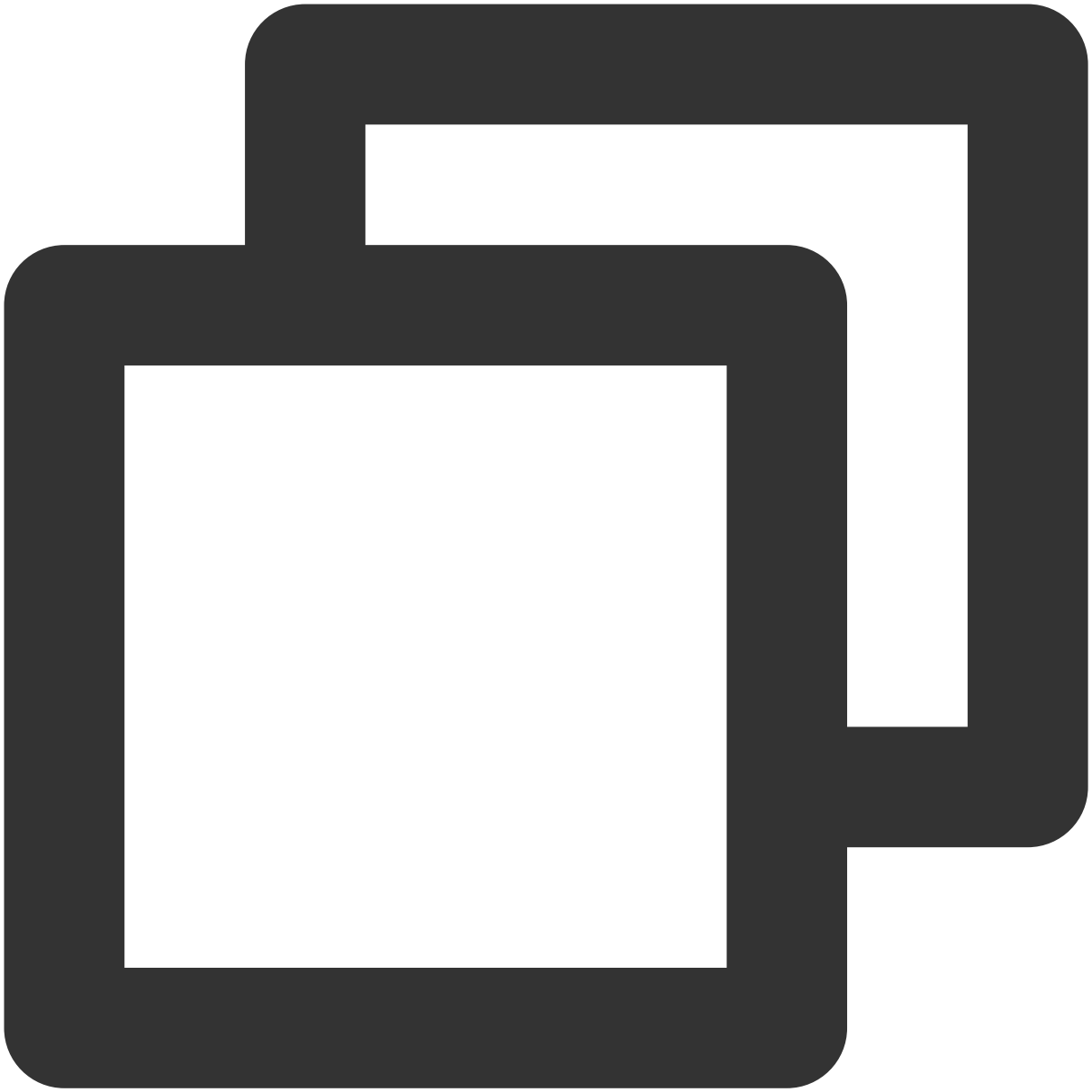
参数	类型	含义
isSyncWaiting	boolean	是否同步退出房间
callback	TUIRoomDefine.ActionCallback	退出房间结果回调

connectOtherRoom

连接其他房间

说明：

用于直播场景下的申请跨房连麦



```
TUIRoomDefine.Request connectOtherRoom(String roomId,
                                       String userId,
                                       int timeout,
                                       TUIRoomDefine.RequestCallback callback);
```

参数：

参数	类型	含义
roomId	String	房间 ID

userId	String	用户 ID
timeout	int	时间
callback	TUIRoomDefine.RequestCallback	连接其他房间请求回调

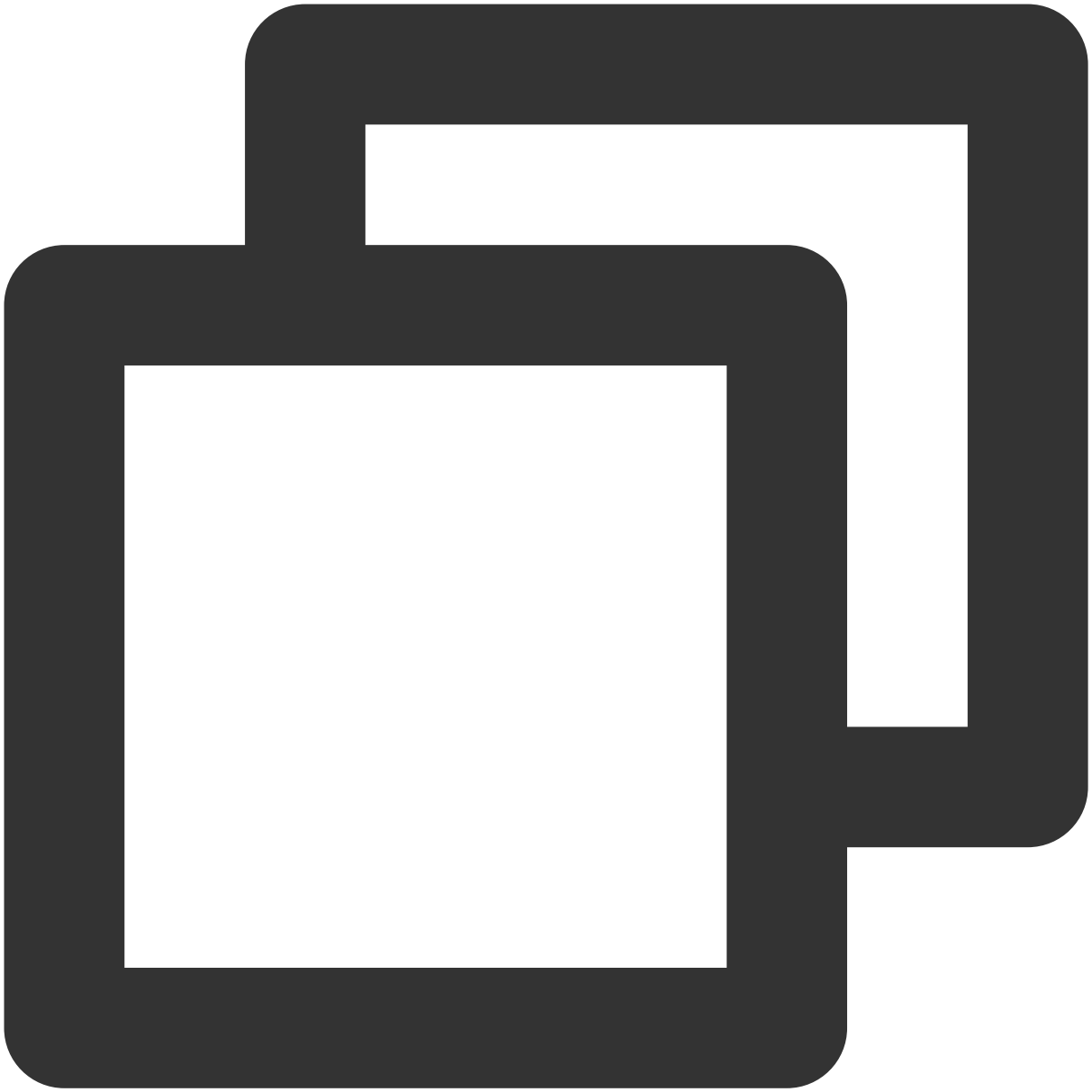
Return：请求体

disconnectOtherRoom

断开与其他房间的连接

说明：

用于直播场景下的断开跨房连麦



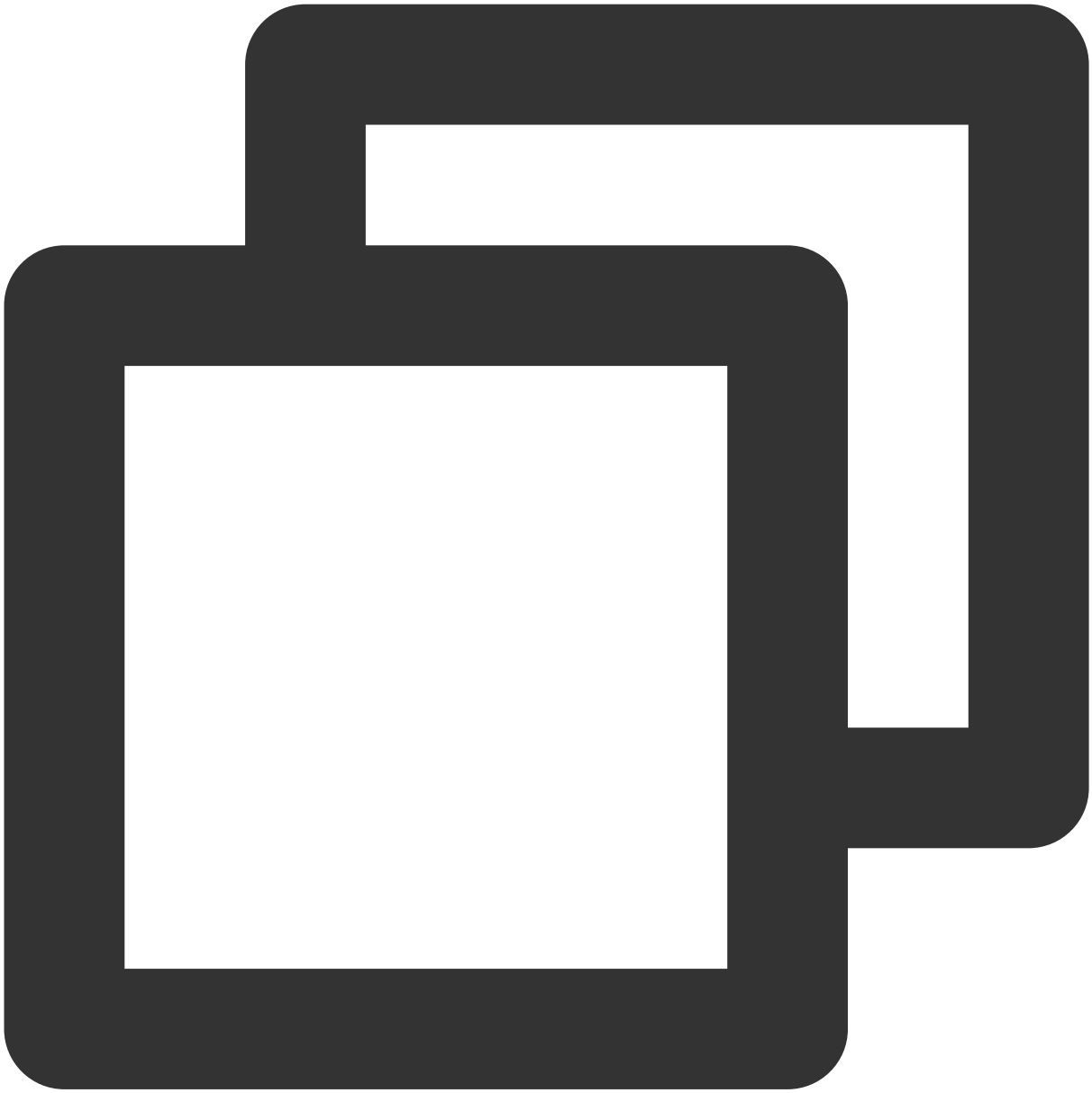
```
void disconnectOtherRoom(TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
callback	TUIRoomDefine.ActionCallback	断开与其他房间的连接结果回调

fetchRoomInfo

获取房间信息



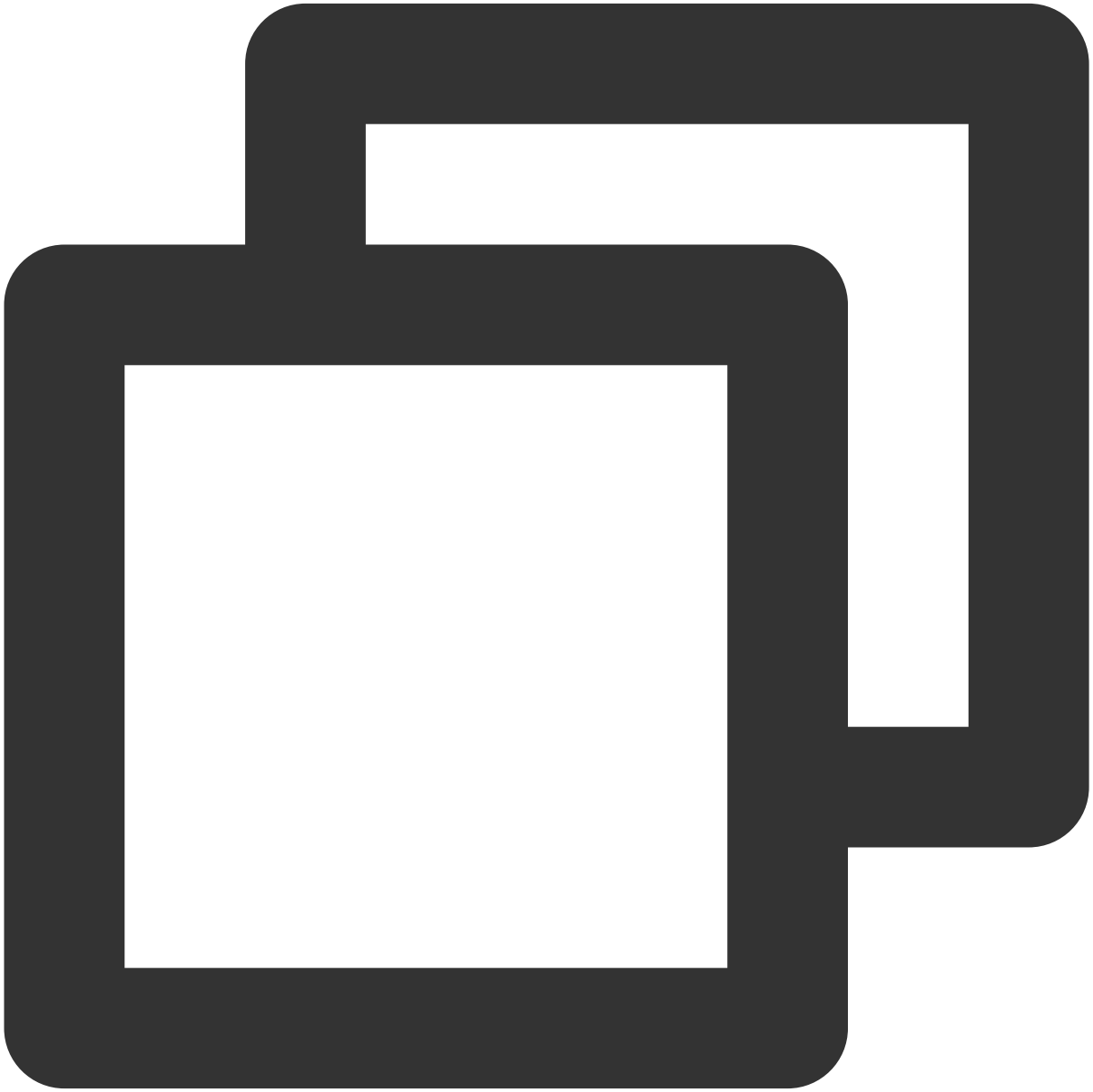
```
void fetchRoomInfo(TUIRoomDefine.GetRoomInfoCallback callback);
```

参数：

参数	类型	含义
callback	TUIRoomDefine.GetRoomInfoCallback	获取房间信息回调

updateRoomNameByAdmin

更新房间名称



```
void updateRoomNameByAdmin(String roomName, TUIRoomDefine.ActionCallback callback);
```

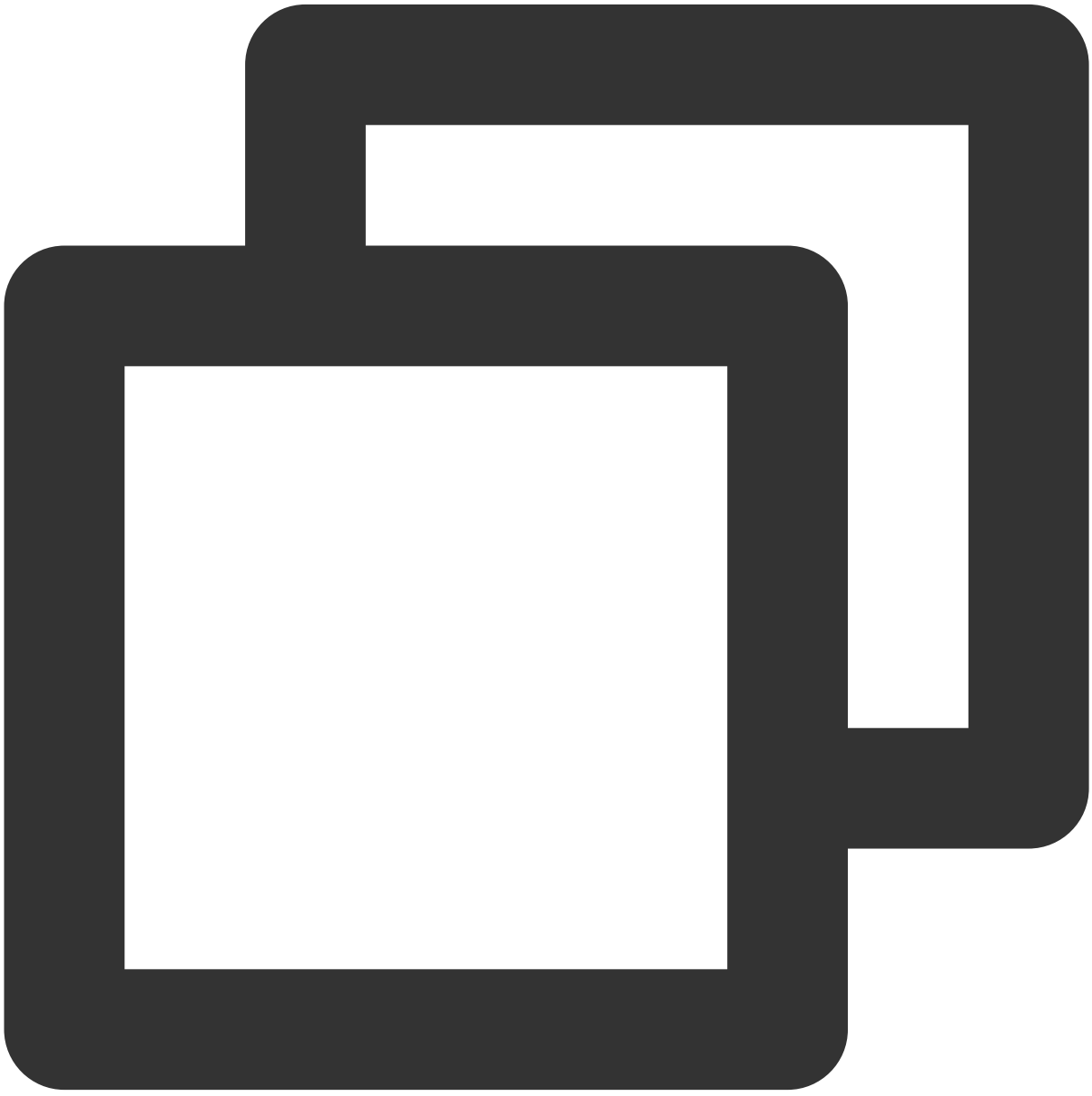
参数：

参数	类型	含义

roomName	String	房间名称
callback	TUIRoomDefine.ActionCallback	更新操作结果回调

updateRoomSpeechModeByAdmin

设置房间管理模式（只有管理员或群主能够调用）



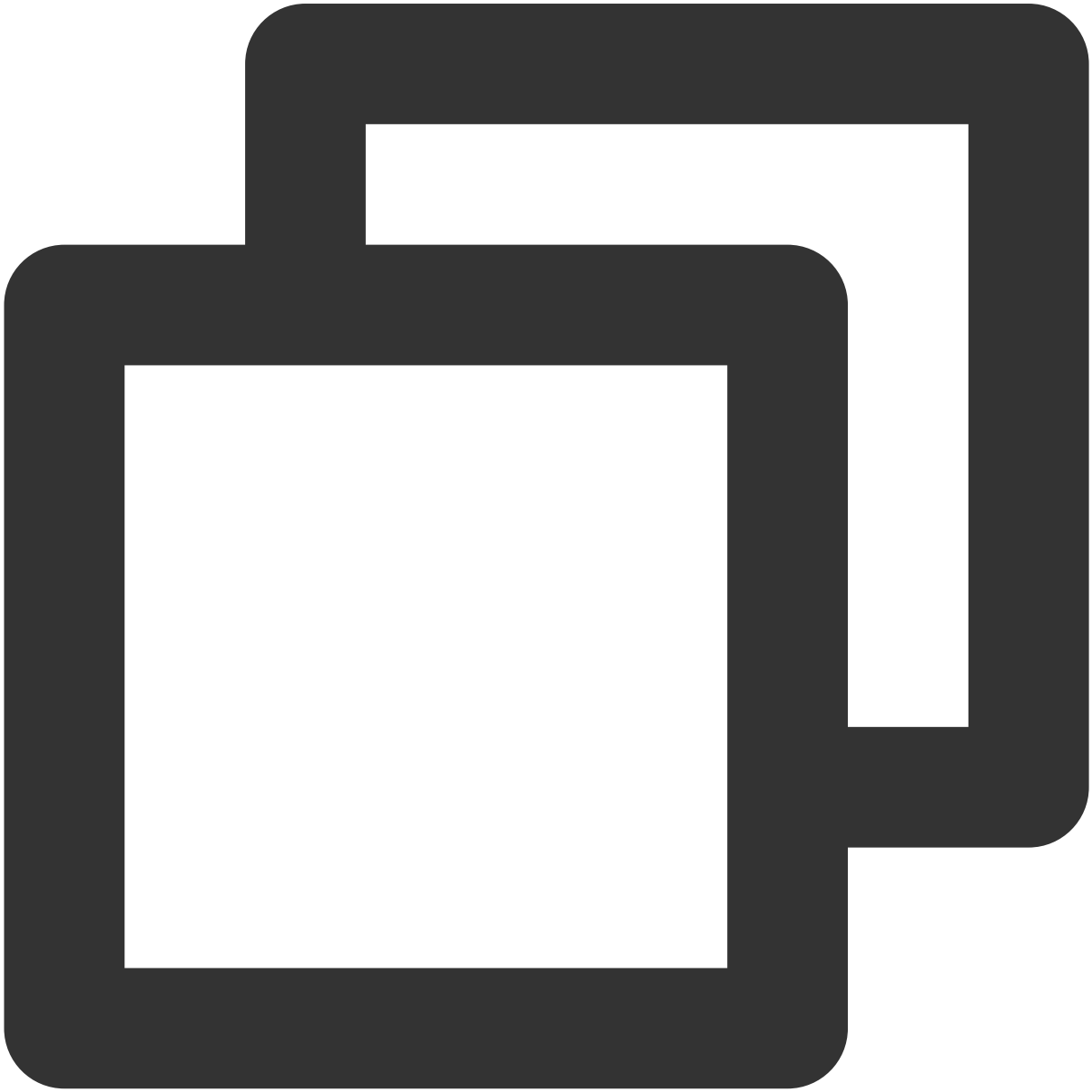
```
void updateRoomSpeechModeByAdmin(TUIRoomDefine.SpeechMode mode, TUIRoomDefine.Actio
```

参数	类型	含义

mode	TUIRoomDefine.SpeechMode	房间模式
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

setLocalVideoView

设置本地用户视频渲染的视图控件



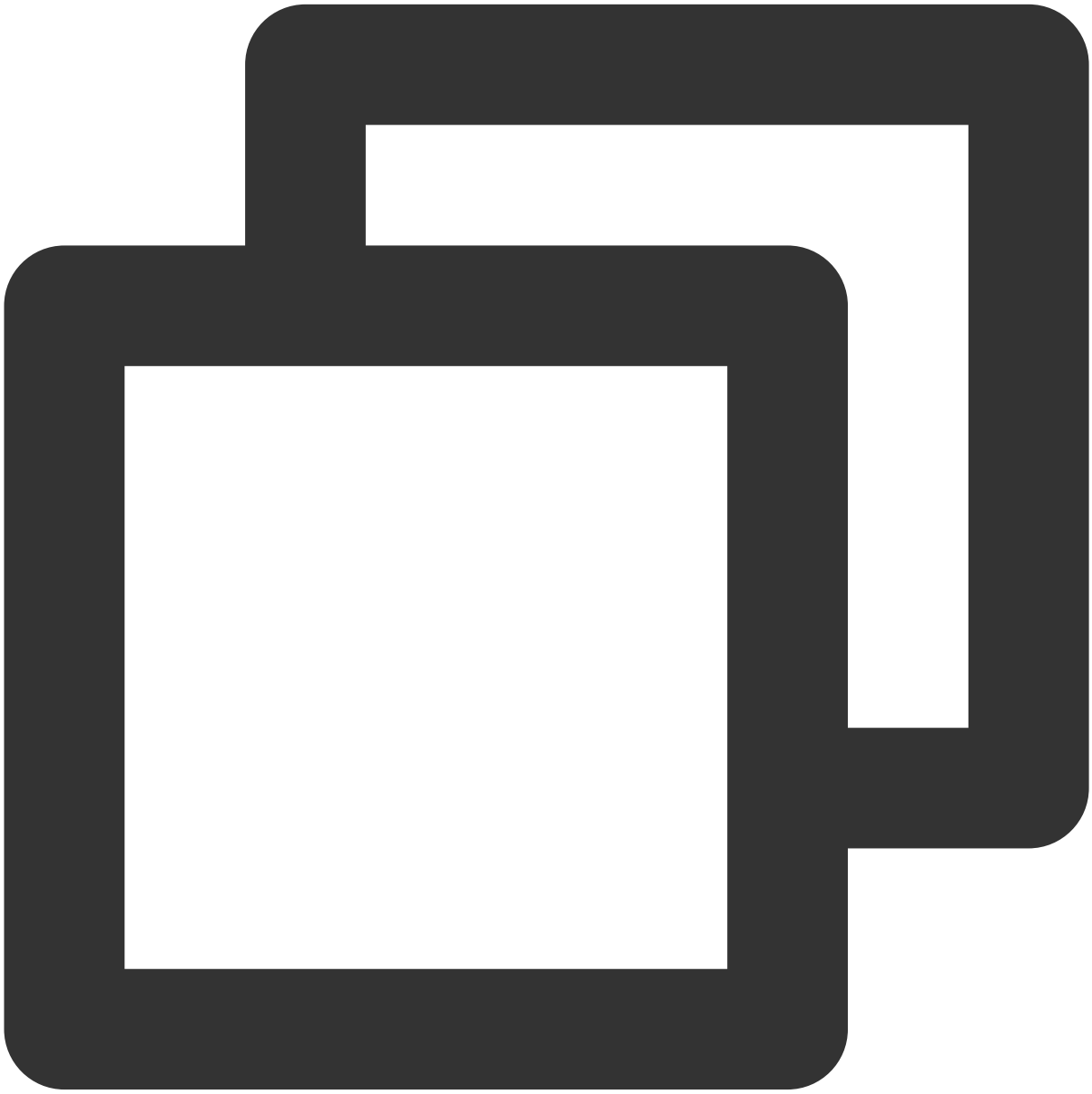
```
void setLocalVideoView(TUIRoomDefine.VideoStreamType streamType,
                       TUIVideoView view);
```

参数：

参数	类型	含义
streamType	TUIRoomDefine.VideoStreamType	本地流类型
view	TUIVideoView	待渲染 view，视频画面将渲染于此 view

openLocalCamera

打开本地摄像头，开始视频流采集



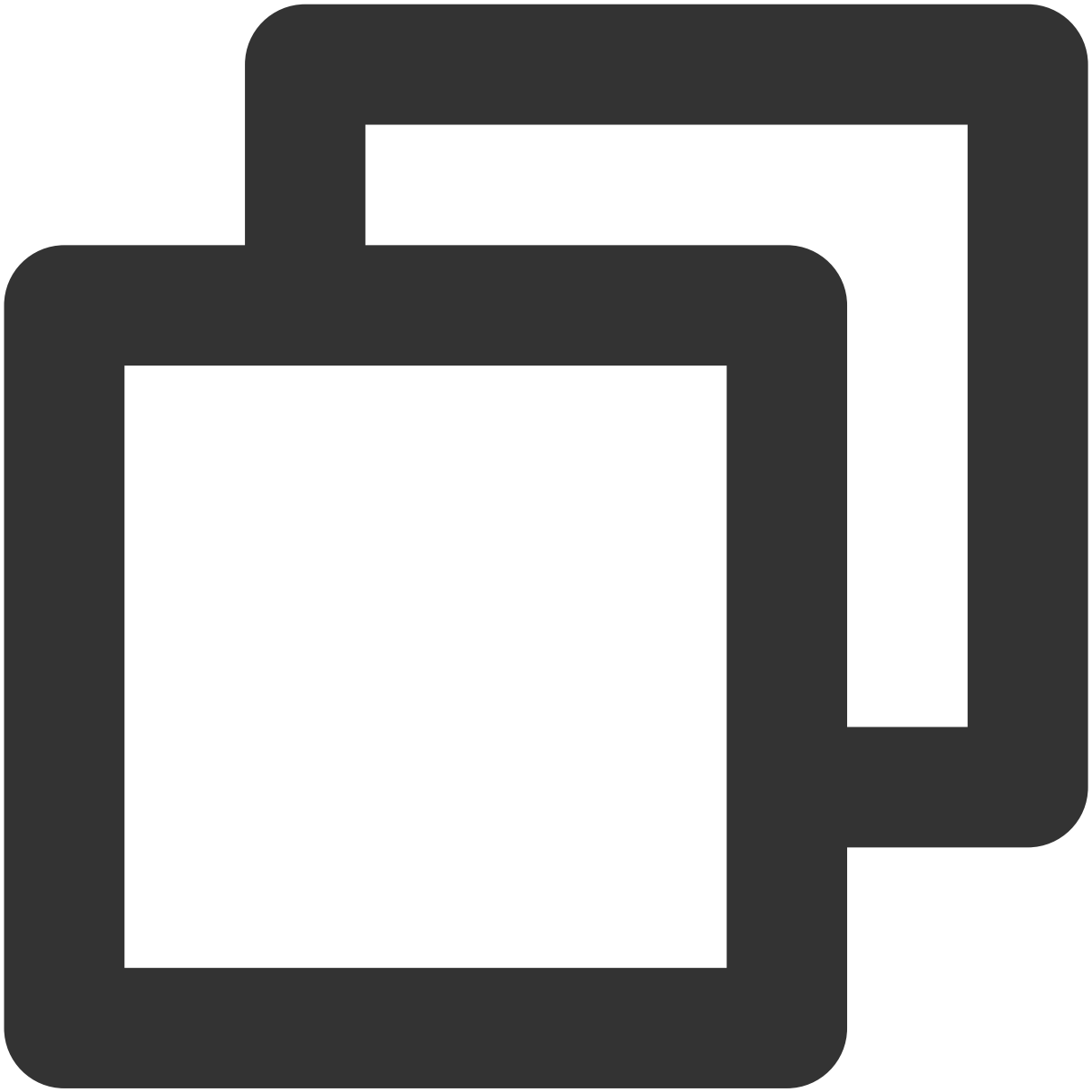
```
void openLocalCamera(boolean isFront,
                     TUIRoomDefine.VideoQuality quality,
                     TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
isFront	boolean	是否使用前置摄像头
quality	TUIRoomDefine.VideoQuality	视频质量
callback	TUIRoomDefine.ActionCallback	打开摄像头结果回调

closeLocalCamera

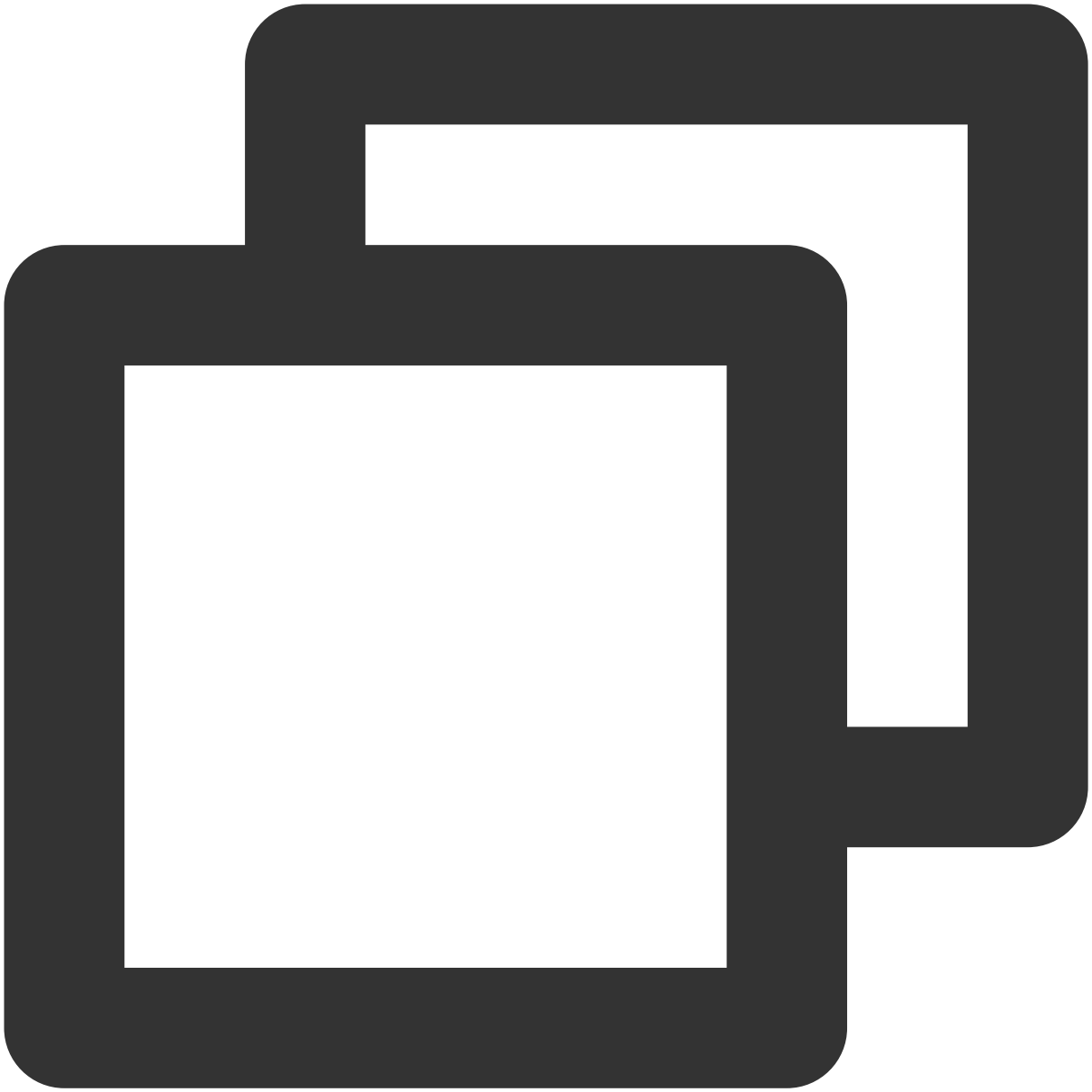
关闭本地摄像头



```
void closeLocalCamera();
```

updateVideoQuality

设置本地视频的参数



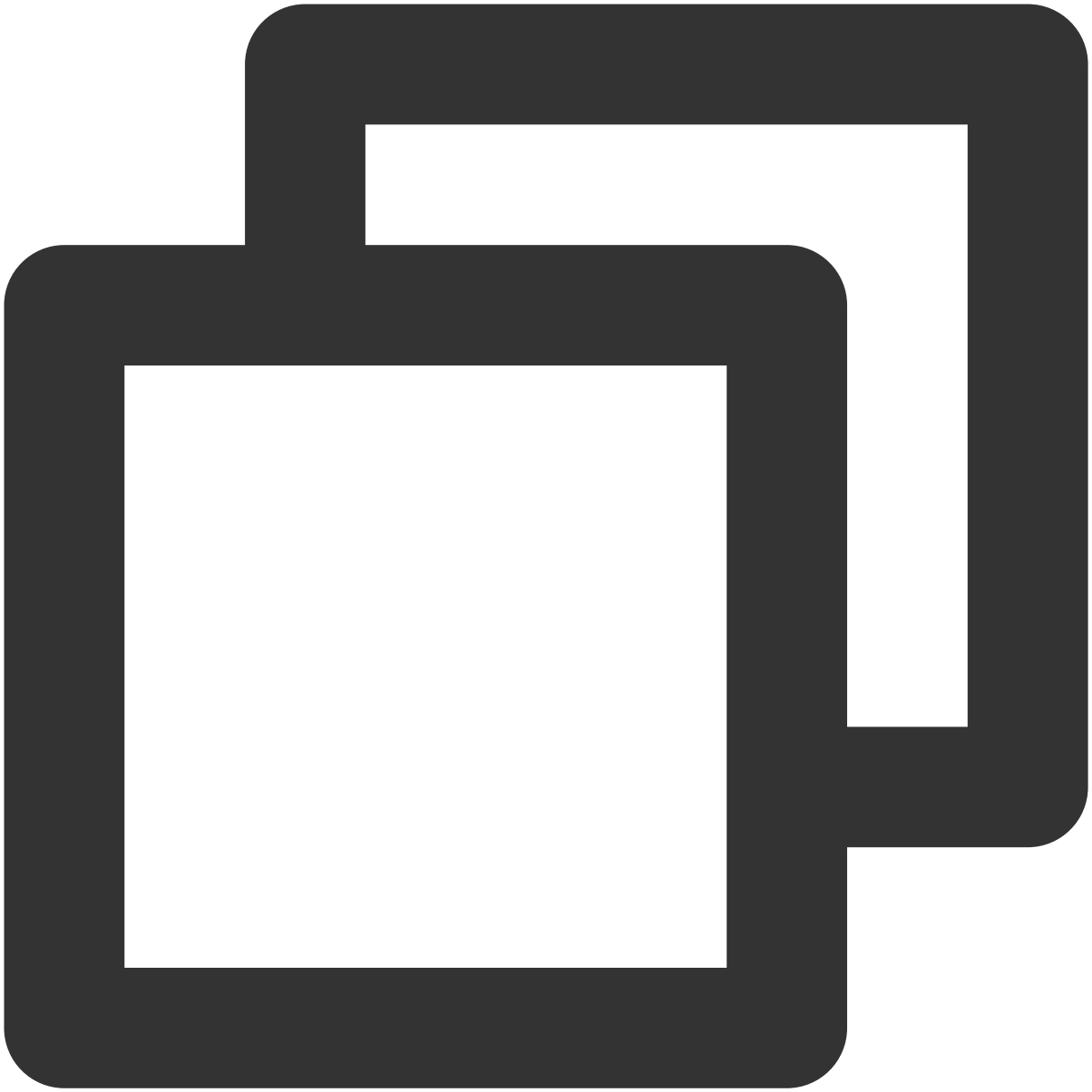
```
void updateVideoQuality(TUIRoomDefine.VideoQuality quality);
```

参数：

参数	类型	含义
quality	TUIRoomDefine.VideoQuality	视频质量

startScreenSharing

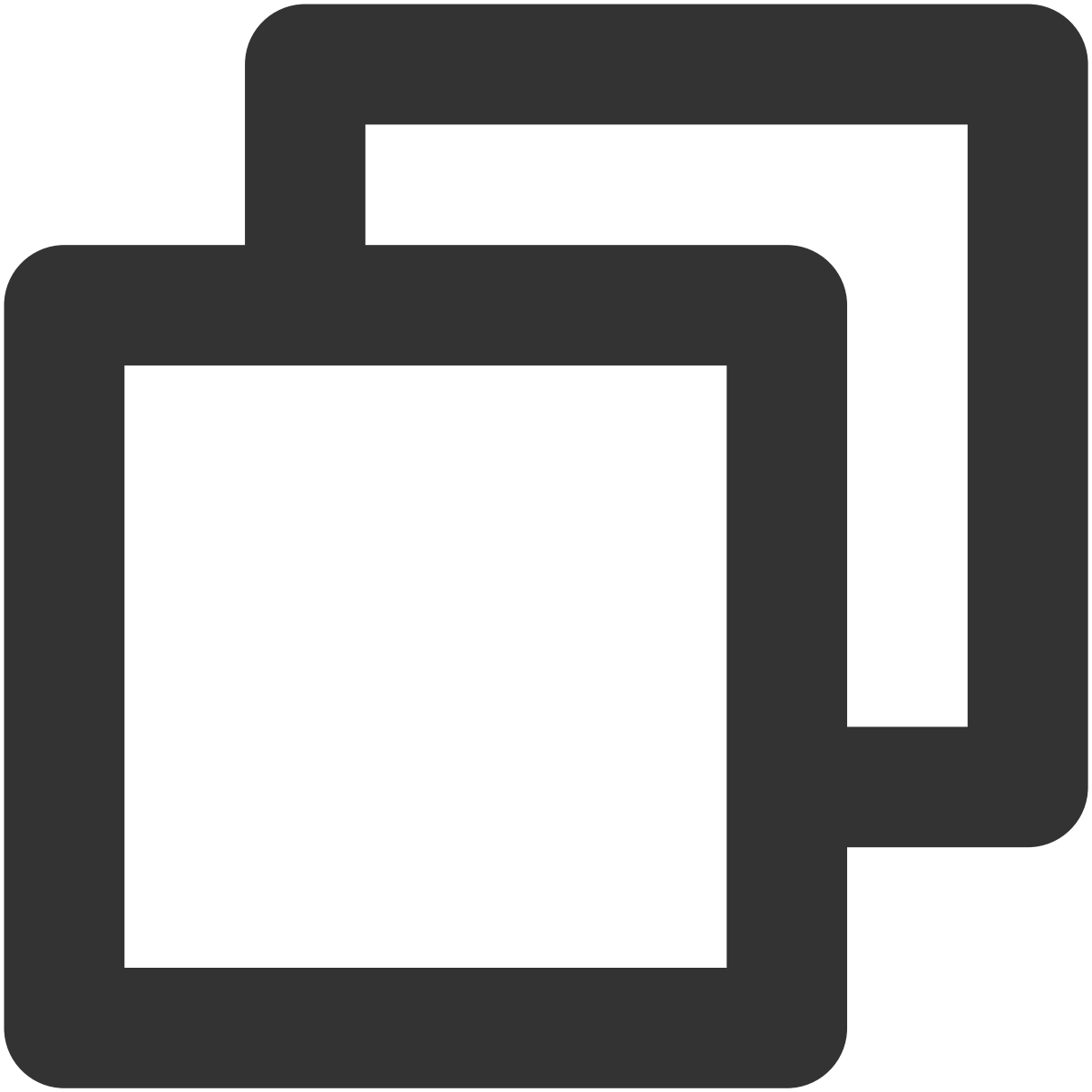
开始屏幕共享



```
void startScreenSharing();
```

stopScreenSharing

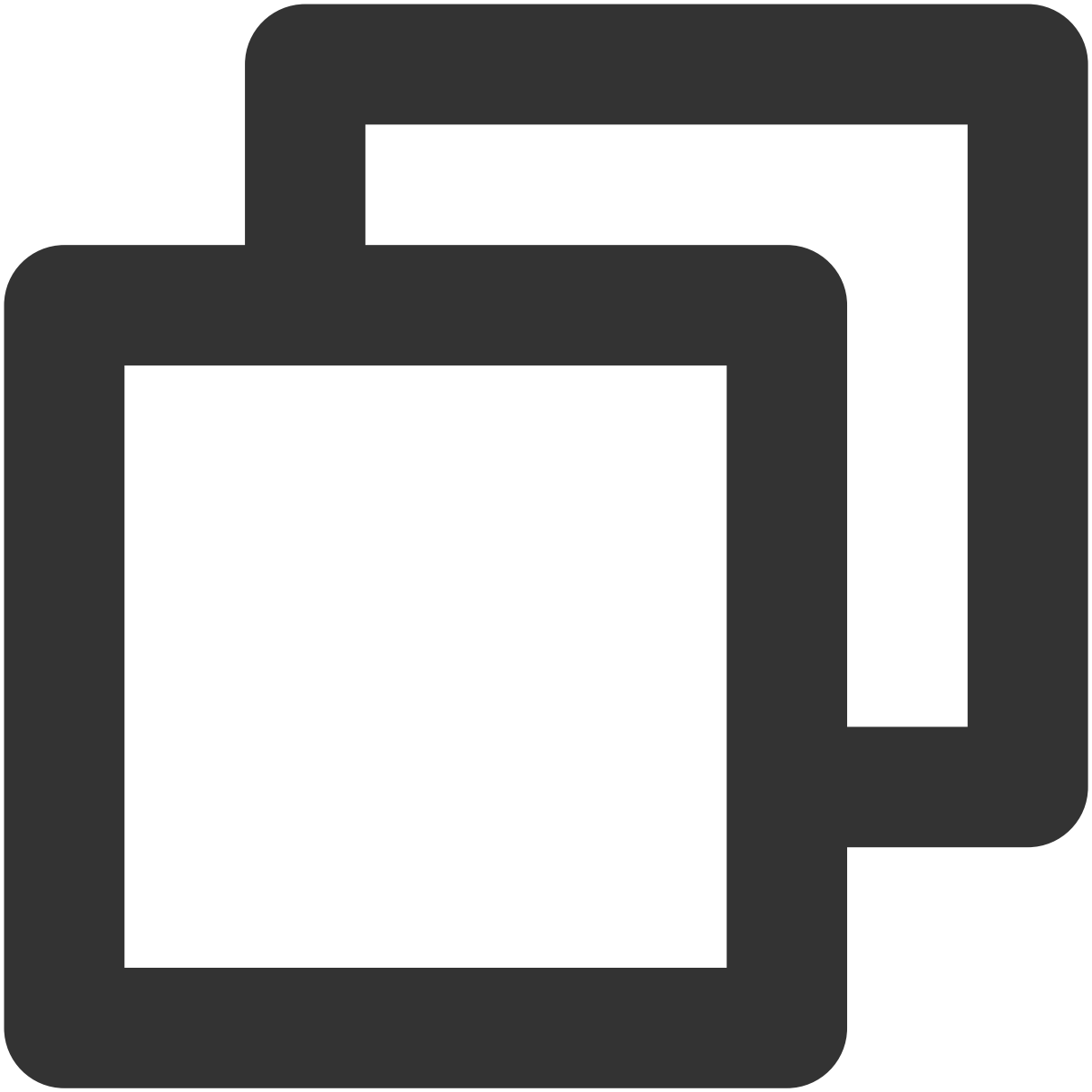
停止屏幕共享



```
void stopScreenSharing();
```

startPushLocalVideo

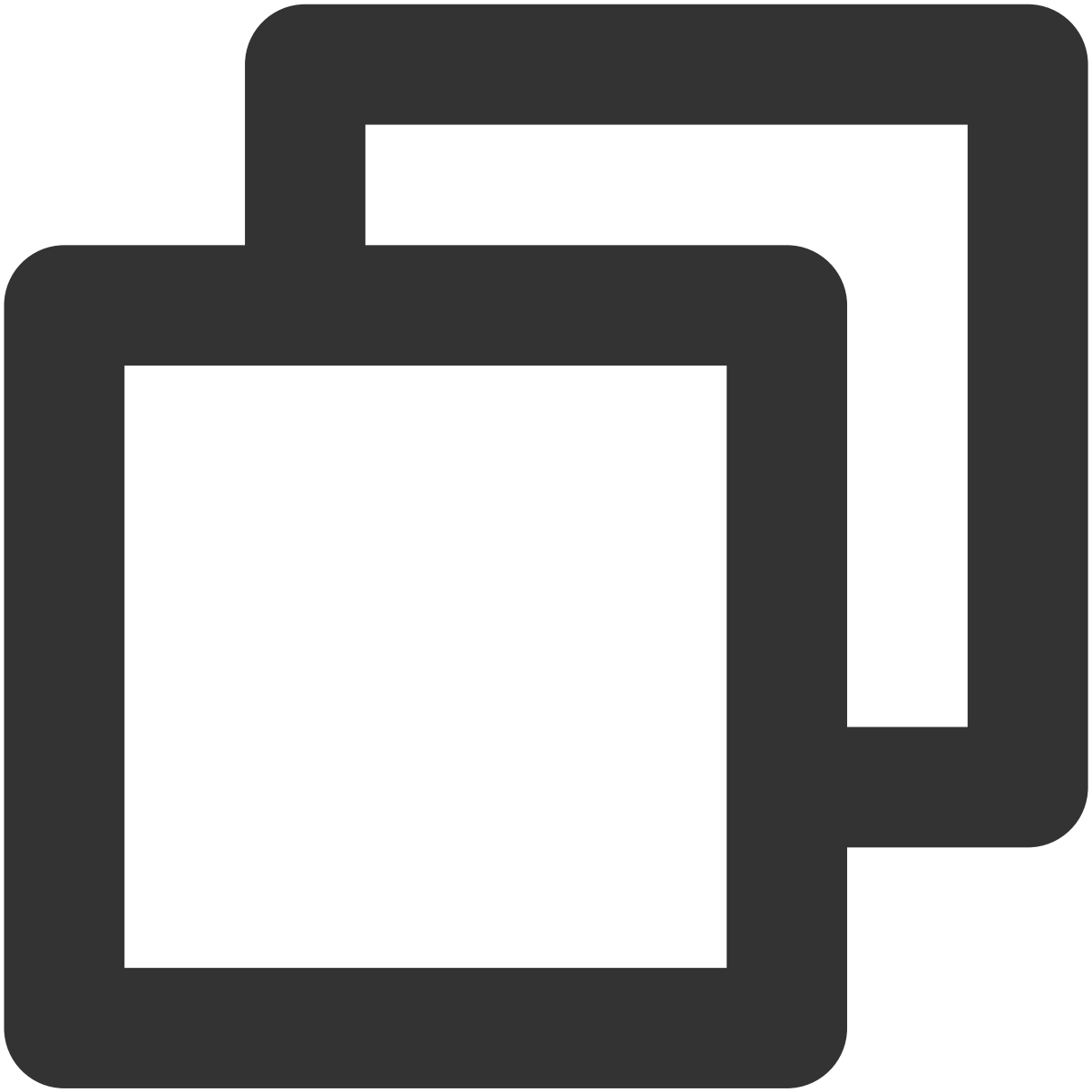
开始向远端推本地视频流



```
void startPushLocalVideo();
```

stopPushLocalVideo

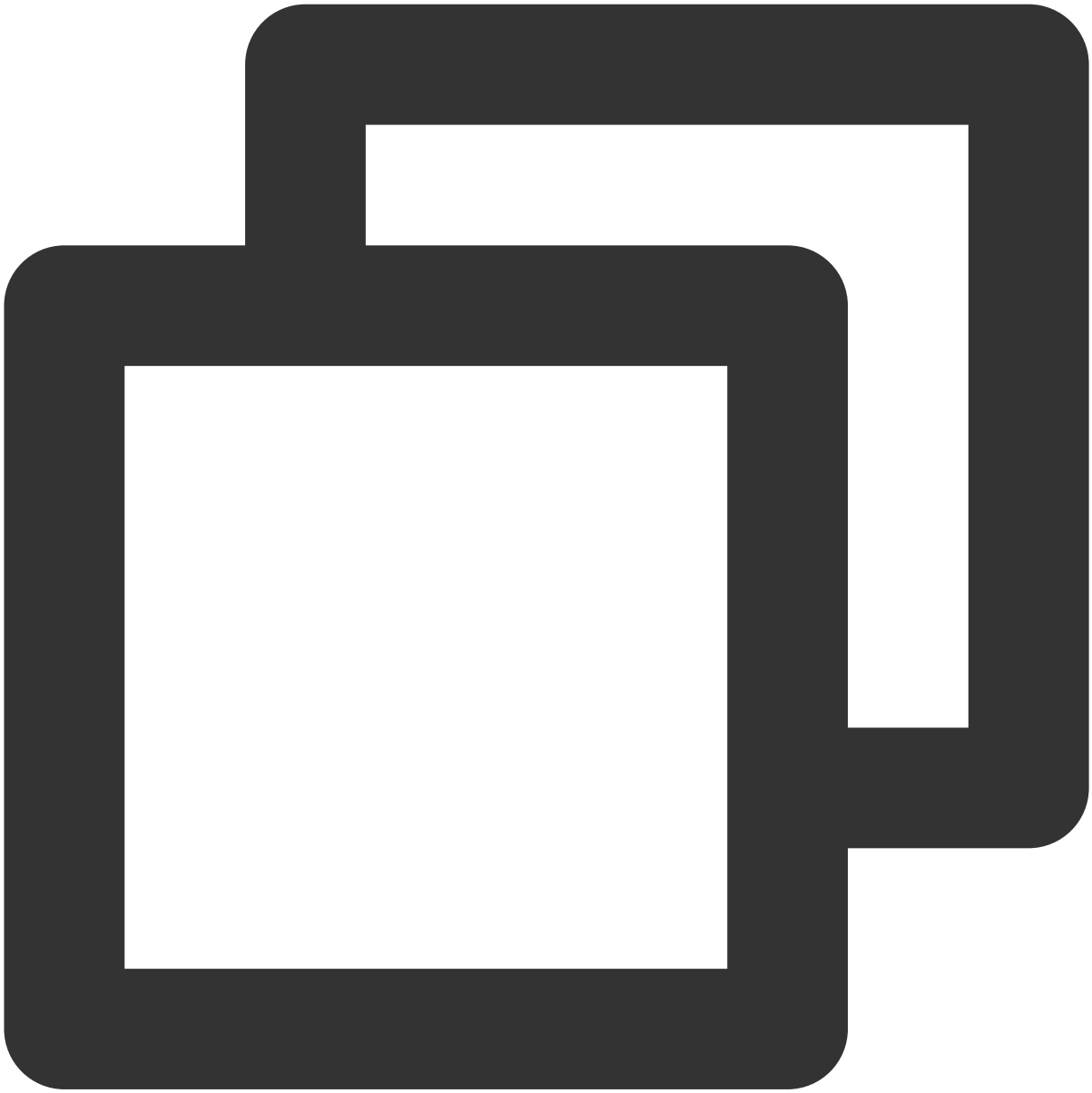
停止向远端推本地视频流



```
void stopPushLocalVideo();
```

openLocalMicrophone

打开本地麦克风

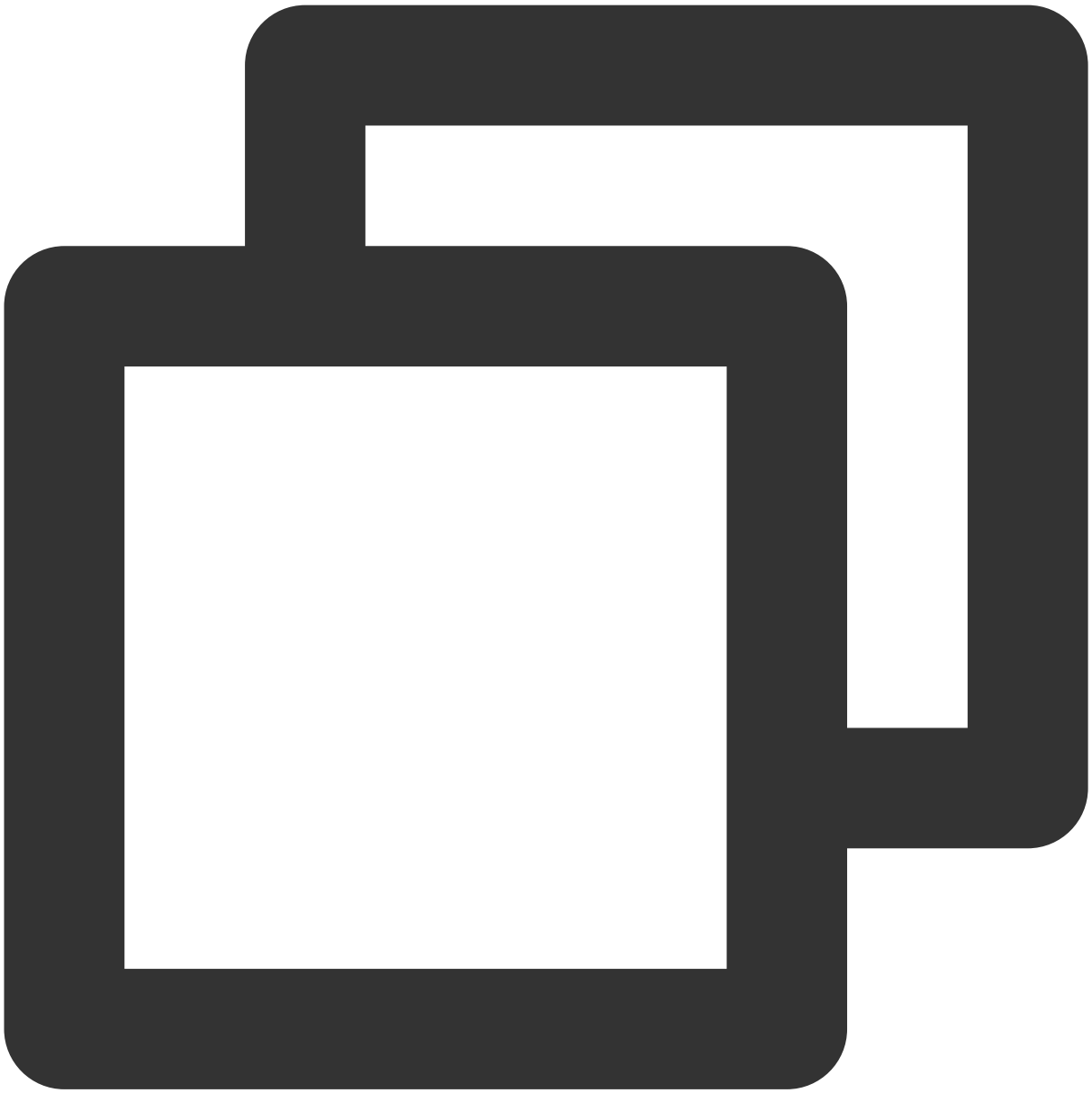


```
void openLocalMicrophone(TUIRoomDefine.AudioQuality quality, TUIRoomDefine.ActionCa
```

参数	类型	含义
quality	TUIRoomDefine.AudioQuality	音频质量
callback	TUIRoomDefine.ActionCallback	打开麦克风结果回调

closeLocalMicrophone

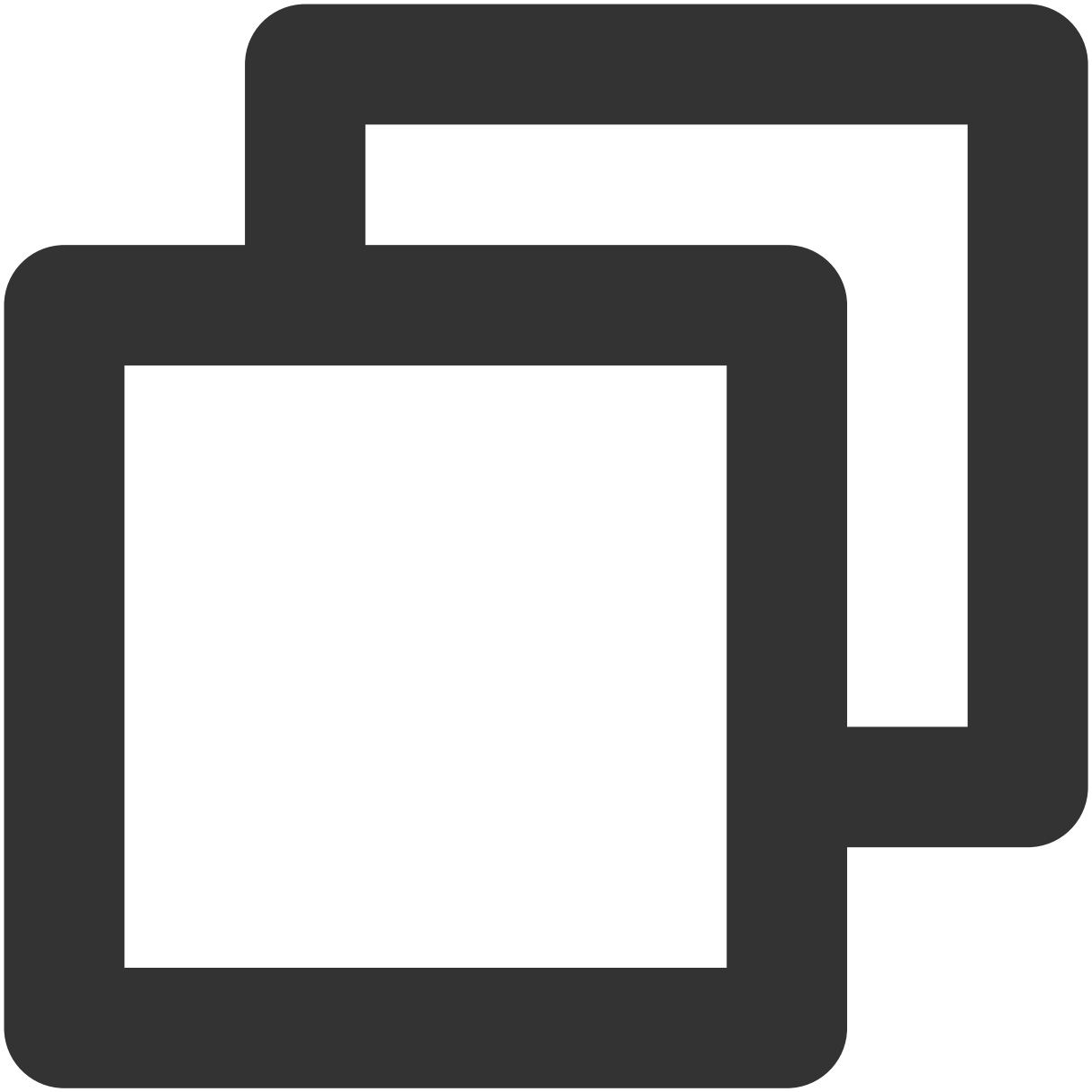
关闭本地麦克风



```
void closeLocalMicrophone();
```

updateAudioQuality

更新本地音频编码质量设置



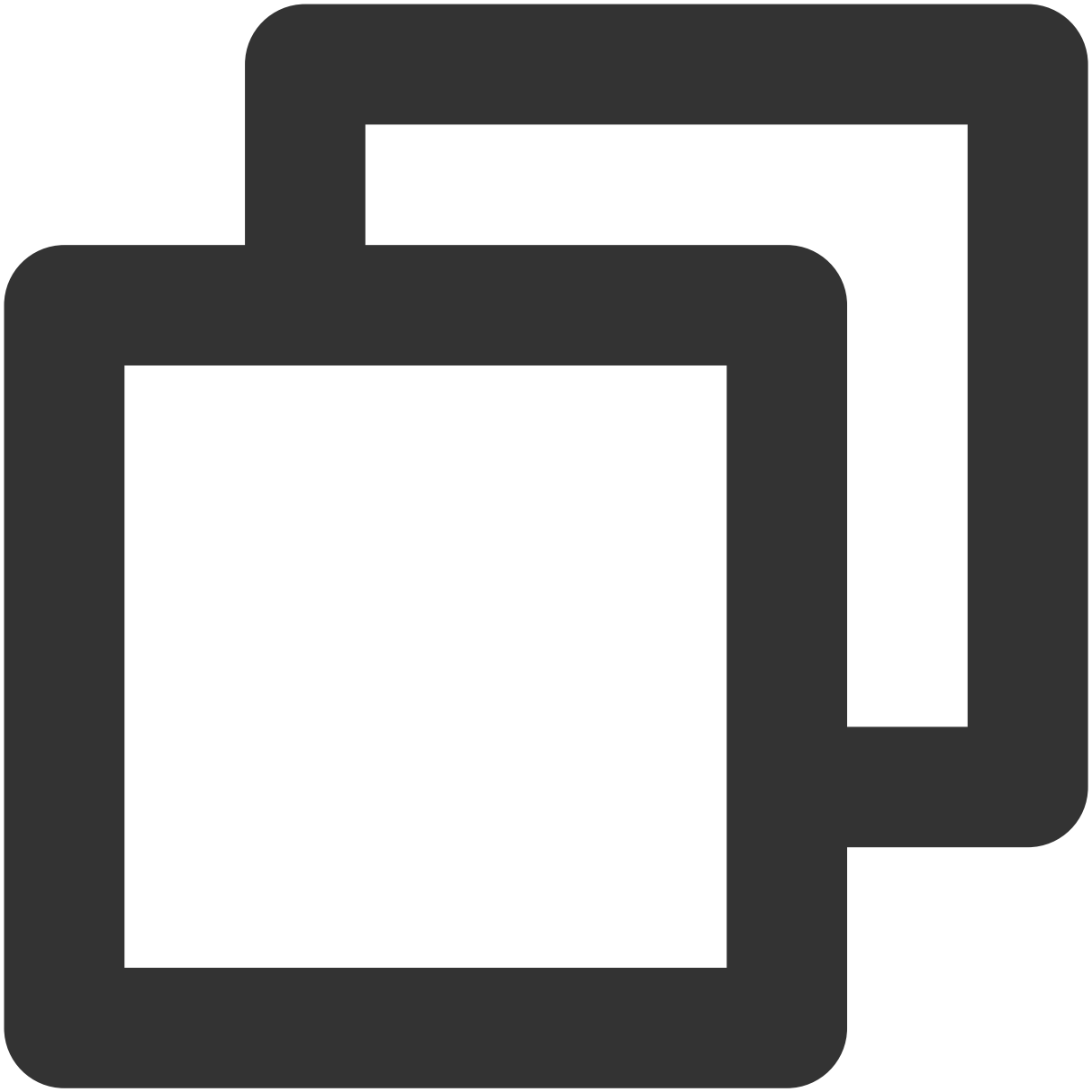
```
void updateAudioQuality(TUIRoomDefine.AudioQuality quality);
```

参数：

参数	类型	含义
quality	TUIRoomDefine.AudioQuality	音频质量

startPushLocalAudio

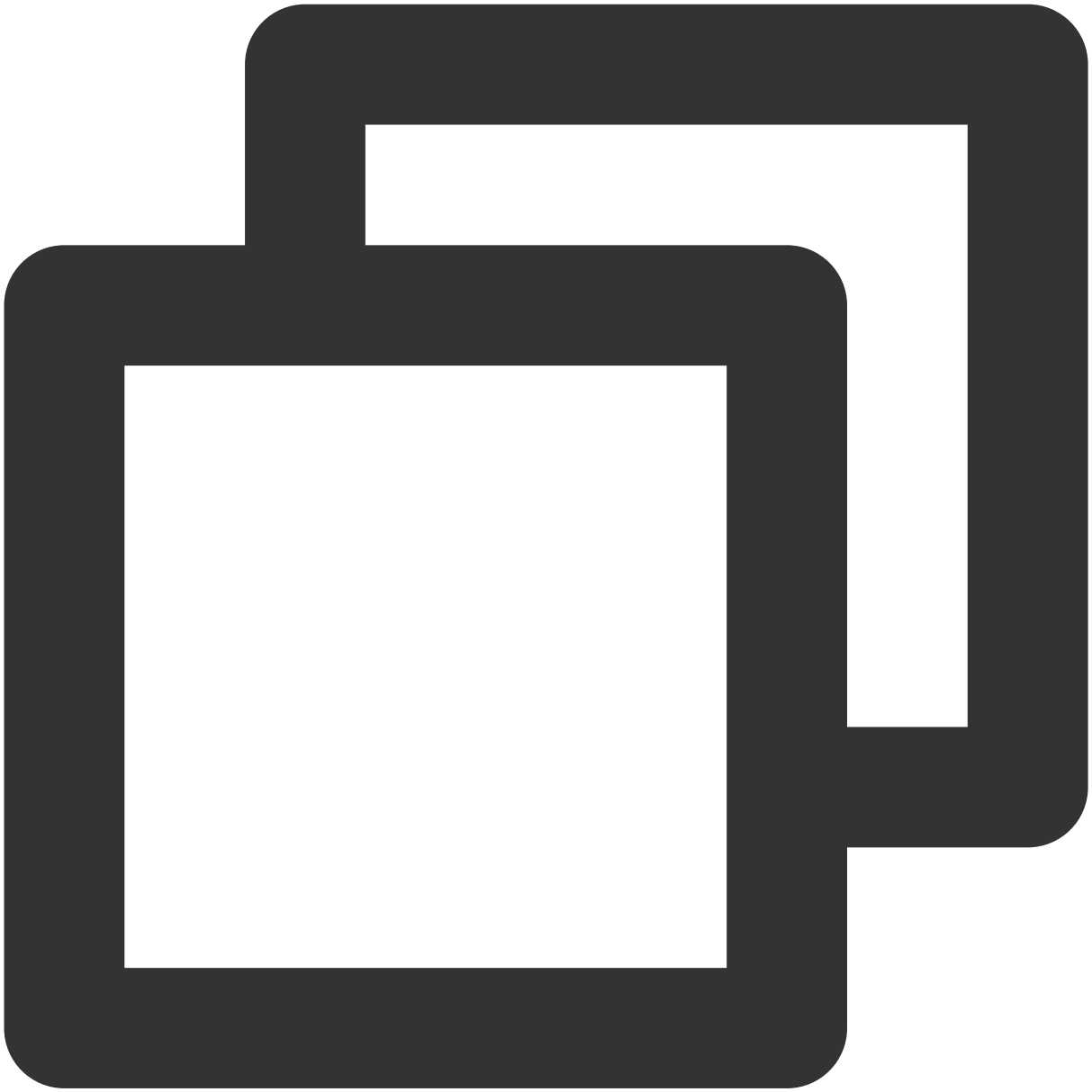
开始向远端推本地音频流



```
void startPushLocalAudio();
```

stopPushLocalAudio

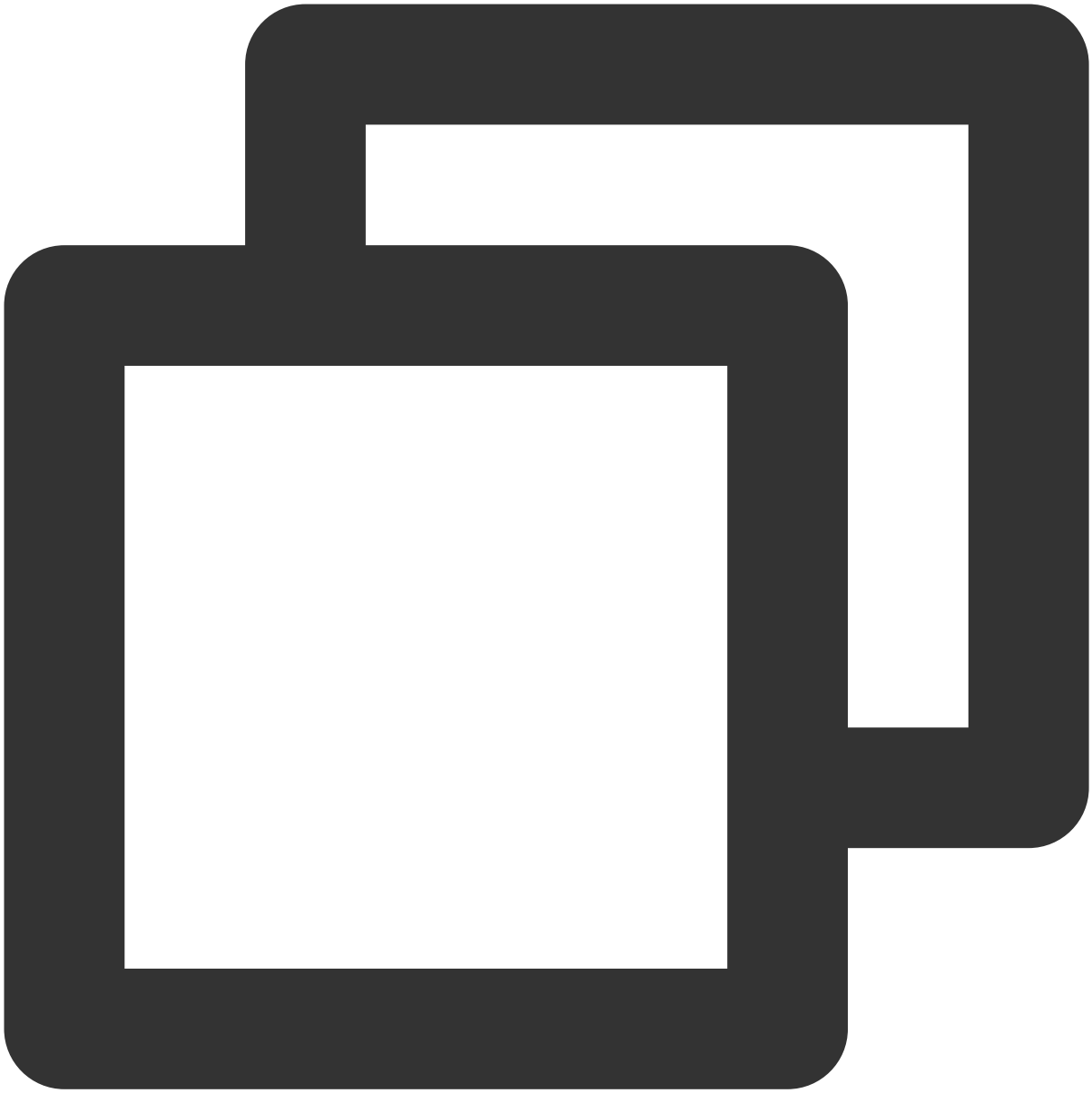
停止向远端推本地音频流



```
void stopPushLocalAudio();
```

setRemoteVideoView

设置远端用户视频渲染的视图控件



```
void setRemoteVideoView(String userId,
                        TUIRoomDefine.VideoStreamType streamType,
                        TUIVideoView view);
```

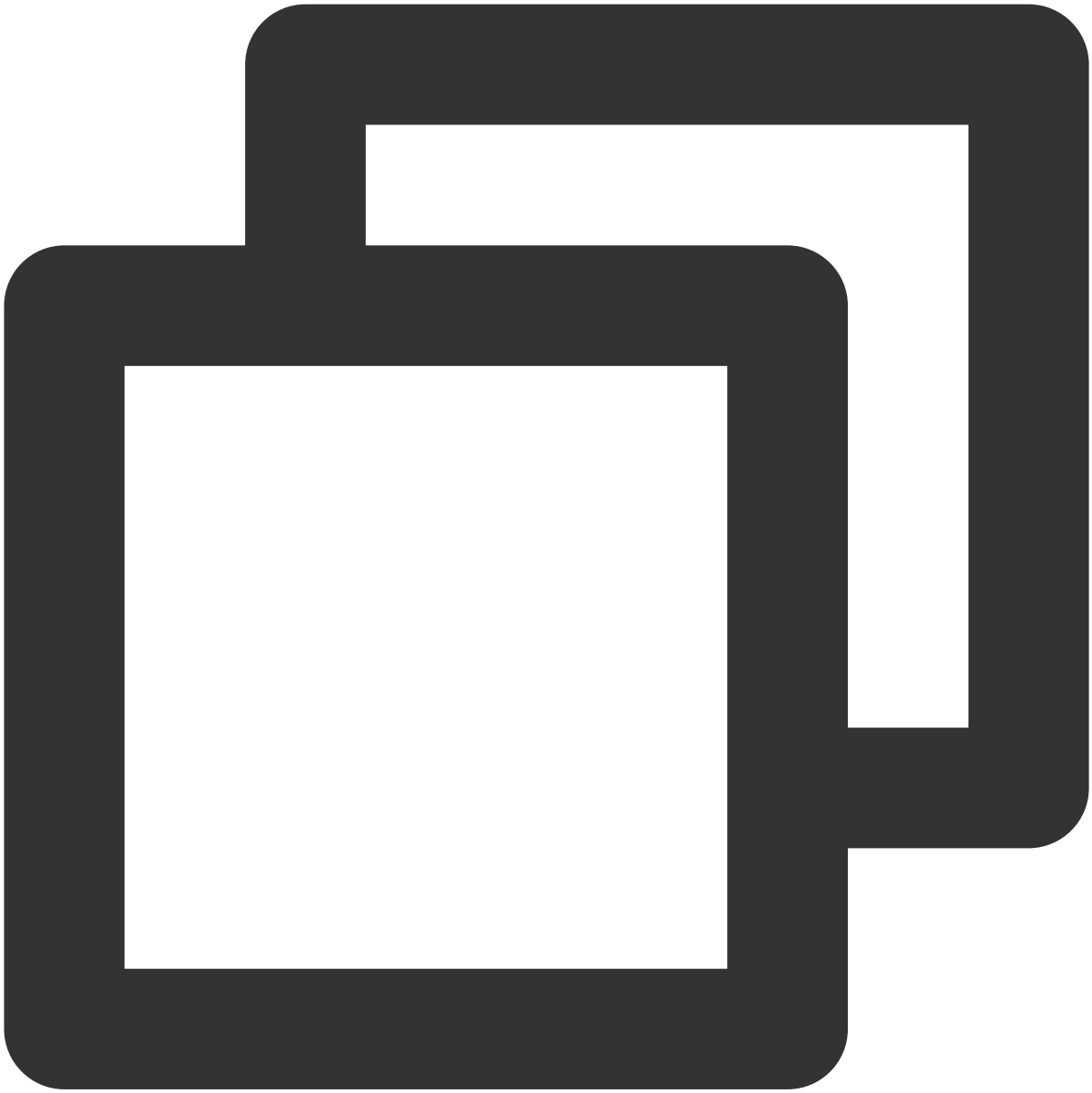
参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIRoomDefine.VideoStreamType	用户流类型

view	TUIVideoView	播放远端用户流的 view

startPlayRemoteVideo

开始播放远端用户视频流



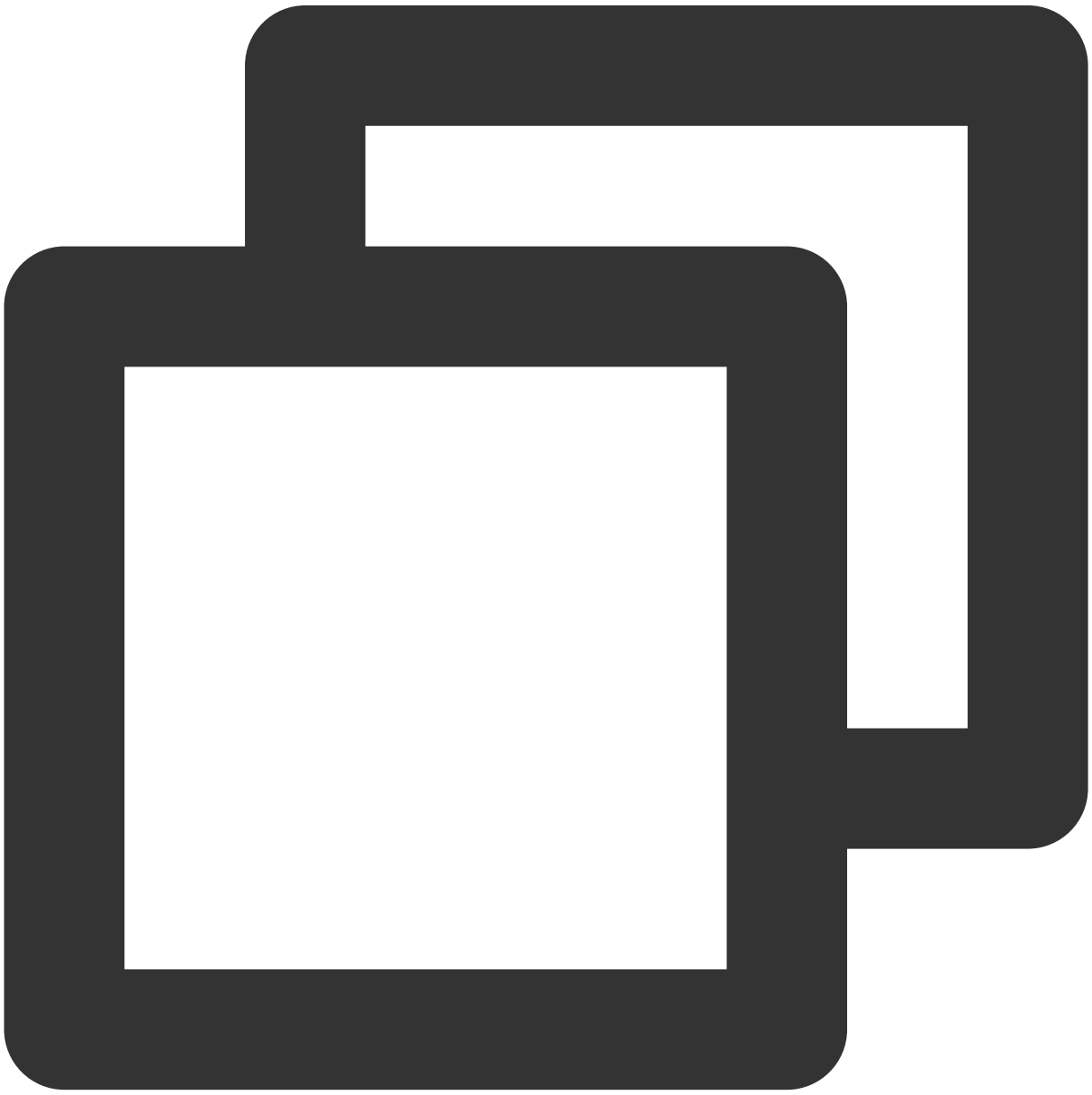
```
void startPlayRemoteVideo(String userId,
                          TUIRoomDefine.VideoStreamType streamType,
                          TUIRoomDefine.PlayCallback callback);
```

参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIRoomDefine.VideoStreamType	用户流类型
callback	TUIRoomDefine.PlayCallback	播放结果回调

stopPlayRemoteVideo

停止播放远端用户视频流



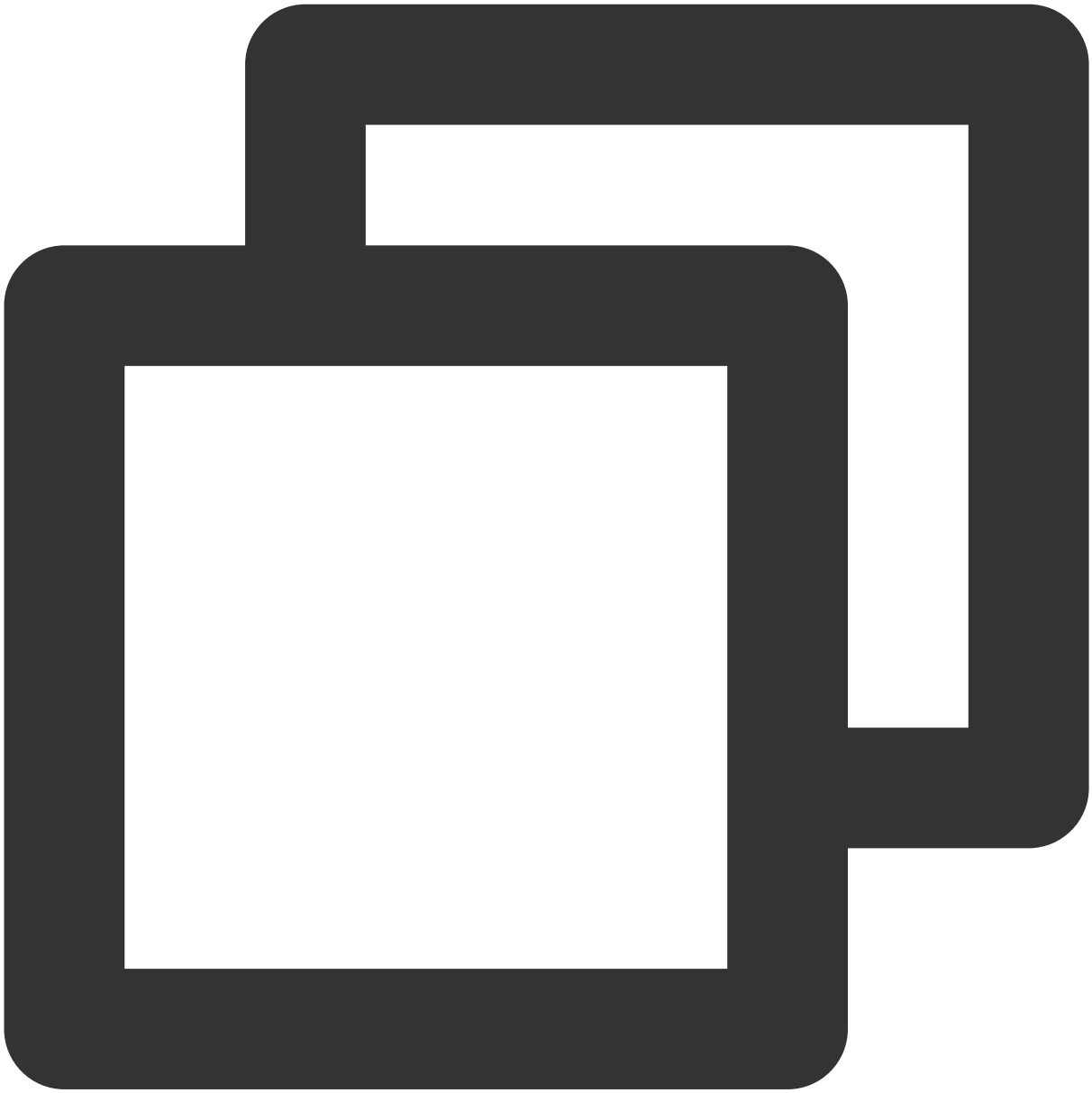
```
void stopPlayRemoteVideo(String userId,TUIRoomDefine.VideoStreamType streamType);
```

参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIRoomDefine.VideoStreamType	用户流类型

muteRemoteAudioStream

将远端用户禁音



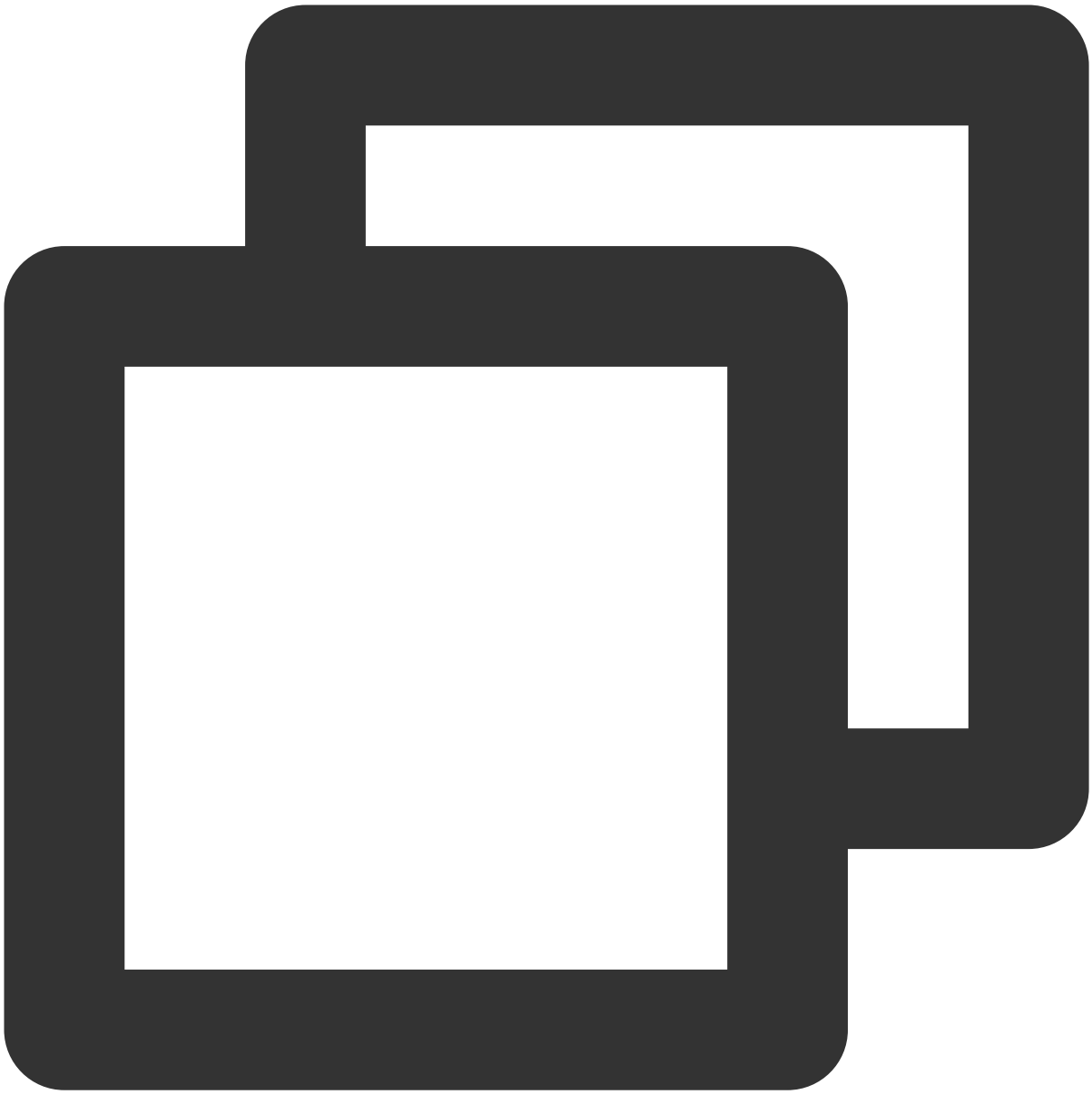
```
void muteRemoteAudioStream(String userId, boolean isMute);
```

参数：

参数	类型	含义
userId	String	用户 ID
isMute	boolean	是否禁音

getUserList

获取当前房间用户列表，注意该接口一次拉取的用户列表量最大为 100 个



```
void getUserList(long nextSequence, TUIRoomDefine.GetUserListCallback callback);
```

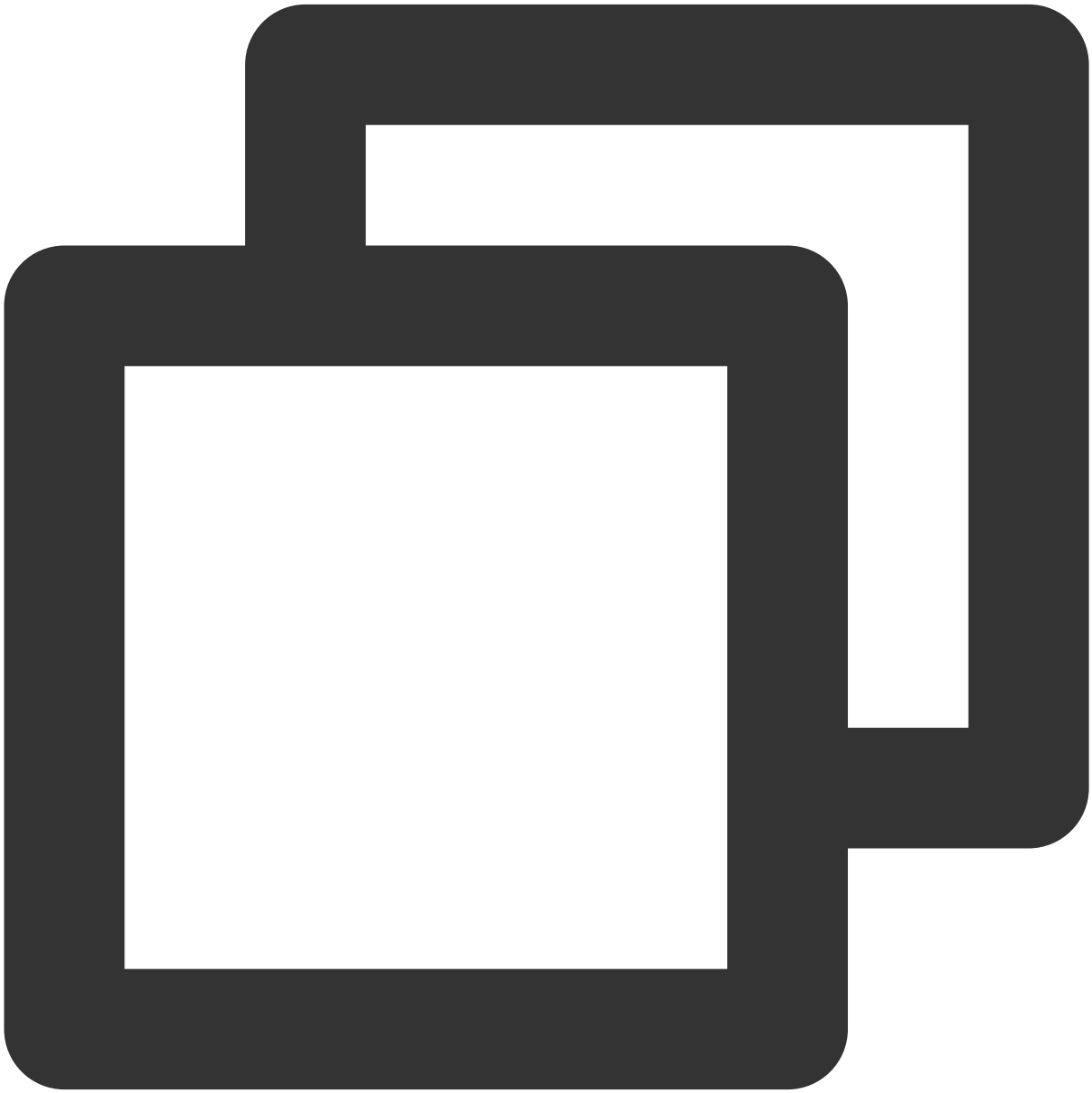
参数：

参数	类型	含义
nextSequence	long	分页拉取标志，第一次拉取填0，回调成功如果 nextSeq 不为零，需要分页，传入 nextSeq 再次拉取，直至 callback 中 nextSeq 为0

callback	TUIRoomDefine.GetUserListCallback	获取房间用户列表回调

getUserInfo

获取用户的详细信息



```
void getUserInfo(String userId, TUIRoomEngineDef.GetUserInfoCallback callback);
```

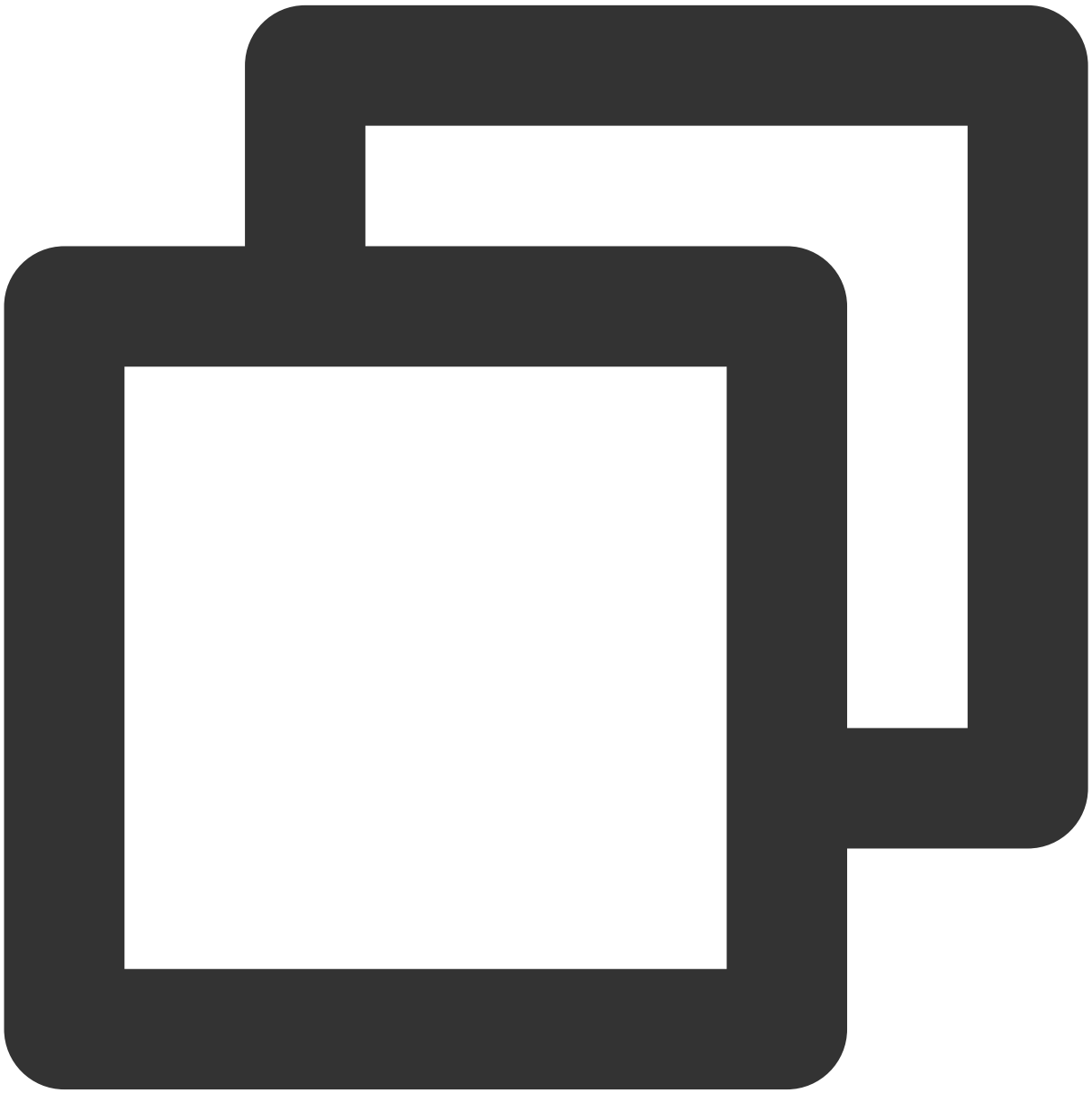
参数：

参数	类型	含义

userId	String	根据 userId 获取该用户的详细信息
callback	TUIRoomDefine.GetUserInfoCallback	获取用户信息回调

changeUserRole

改变用户的角色，只有管理员或群主能够调用



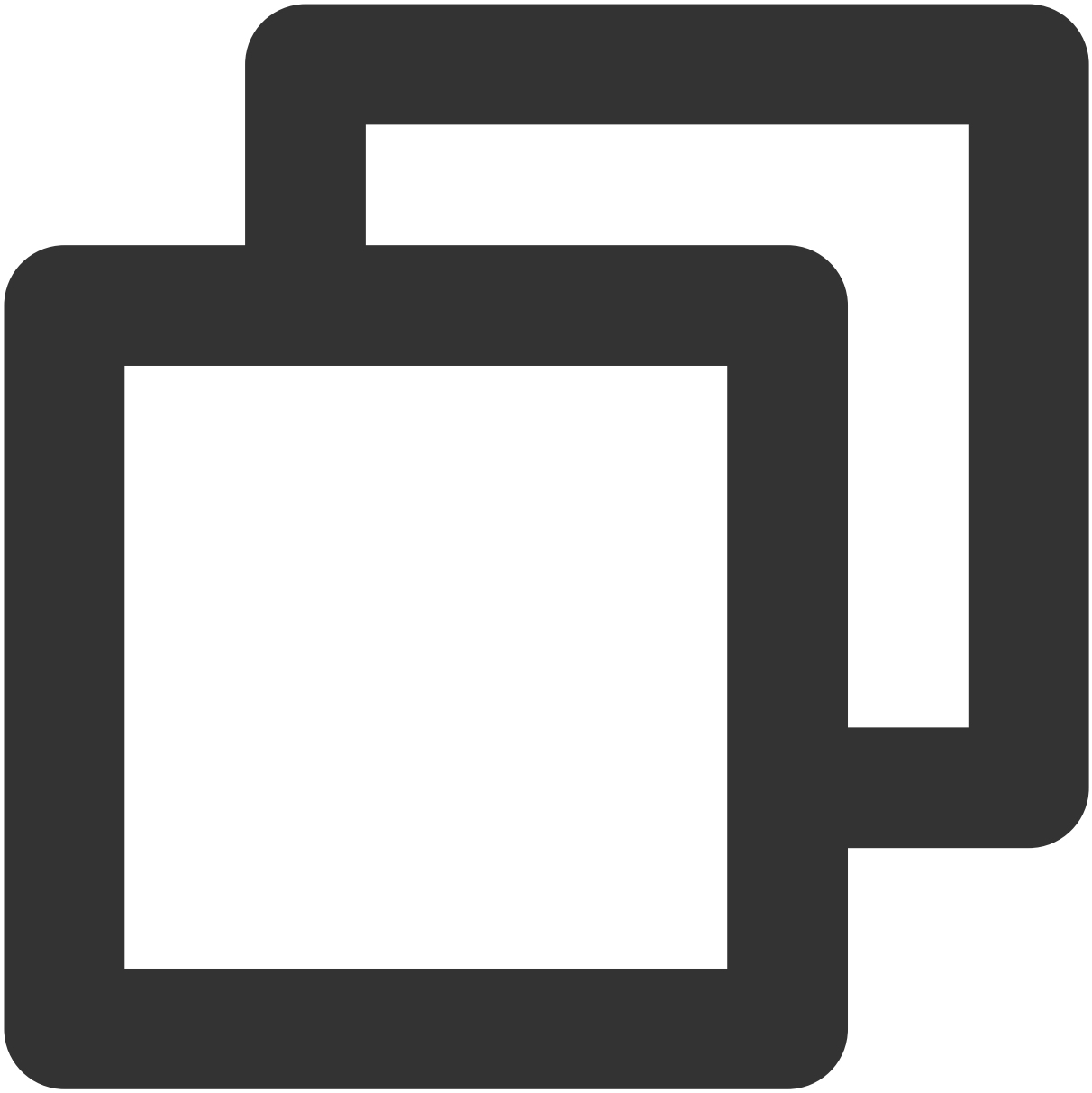
```
void changeUserRole(String userId,
                    TUIRoomDefine.Role role,
                    TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
userId	String	用户 ID
role	TUIRoomDefine.Role	用户角色
callback	TUIRoomDefine.ActionCallback	改变角色结果回调

kickRemoteUserOutOfRoom

将用户踢出房间，只有管理员或群主能够调用



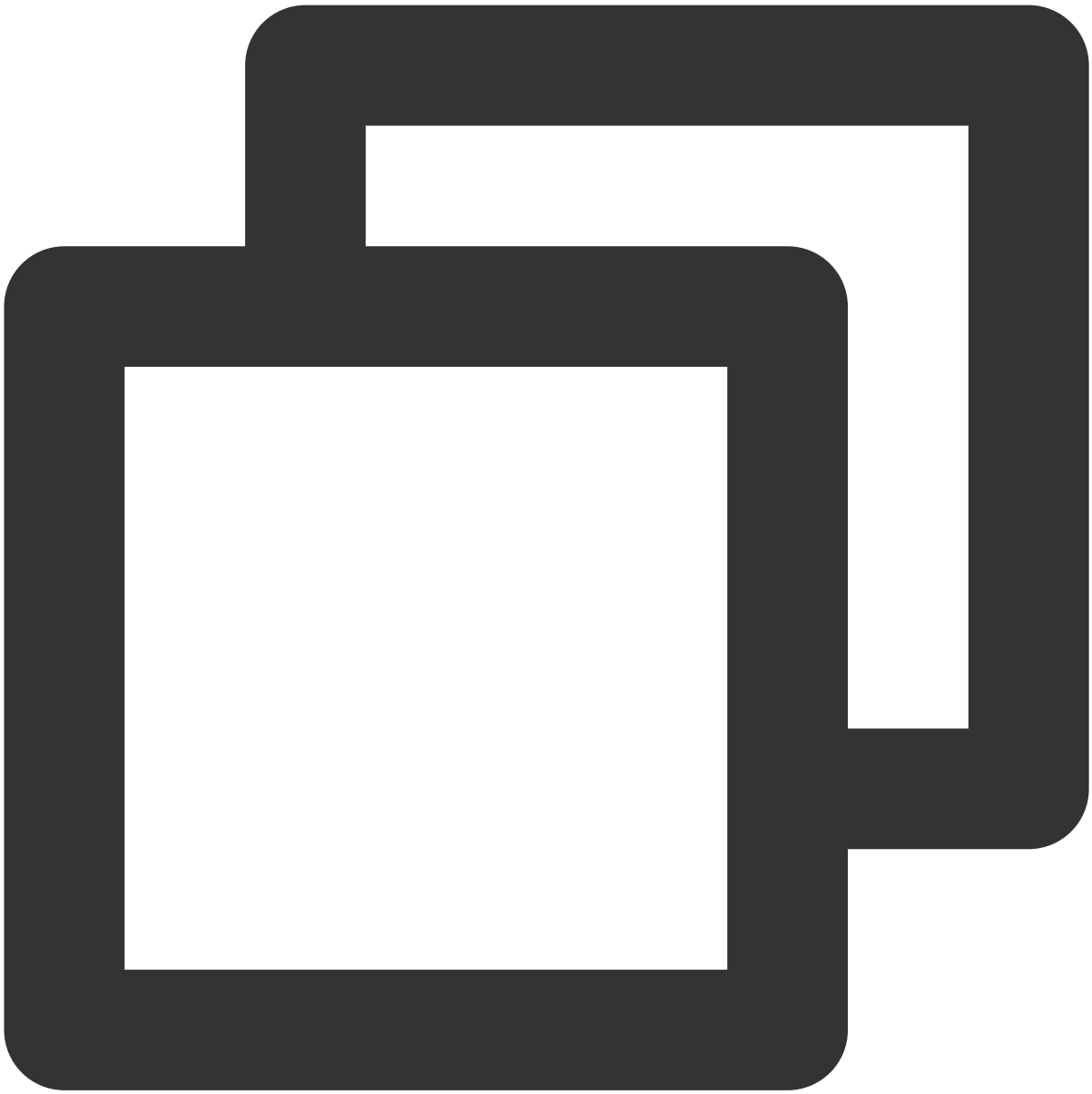
```
void kickRemoteUserOutOfRoom(String userId, TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
userId	String	用户 ID
callback	TUIRoomDefine.ActionCallback	操作结果回调

disableDeviceForAllUserByAdmin

全体用户媒体设备管理，只有管理员或群主能够调用



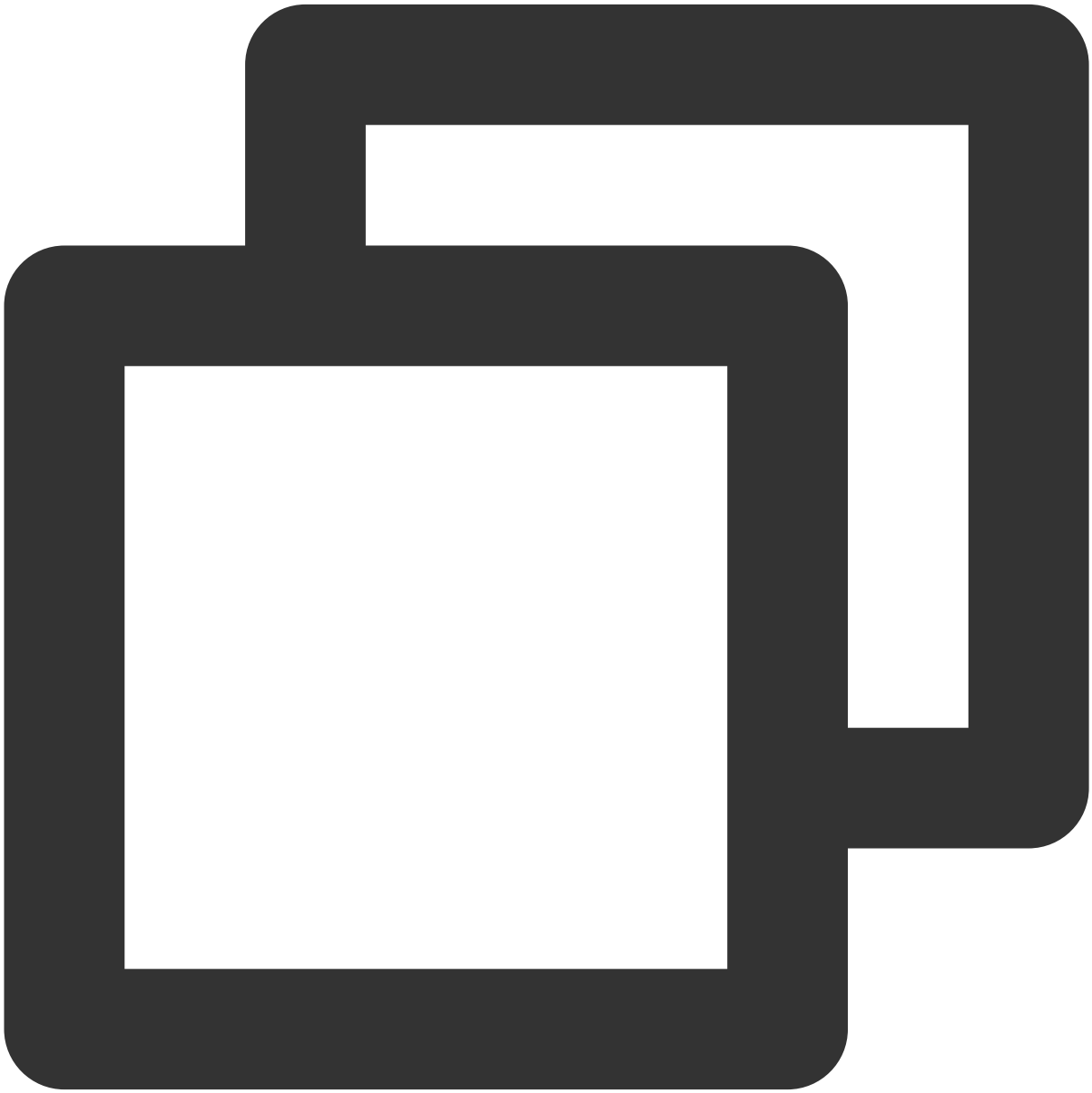
```
void disableDeviceForAllUserByAdmin(TUIRoomDefine.MediaDevice device,
                                    boolean isDisable,
                                    TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
device	TUIRoomDefine.MediaDevice	设备
isDisable	boolean	是否禁用

callback	TUIRoomDefine.ActionCallback	操作结果回调

openRemoteDeviceByAdmin

请求远端用户打开媒体设备，只有管理员或群主能够调用

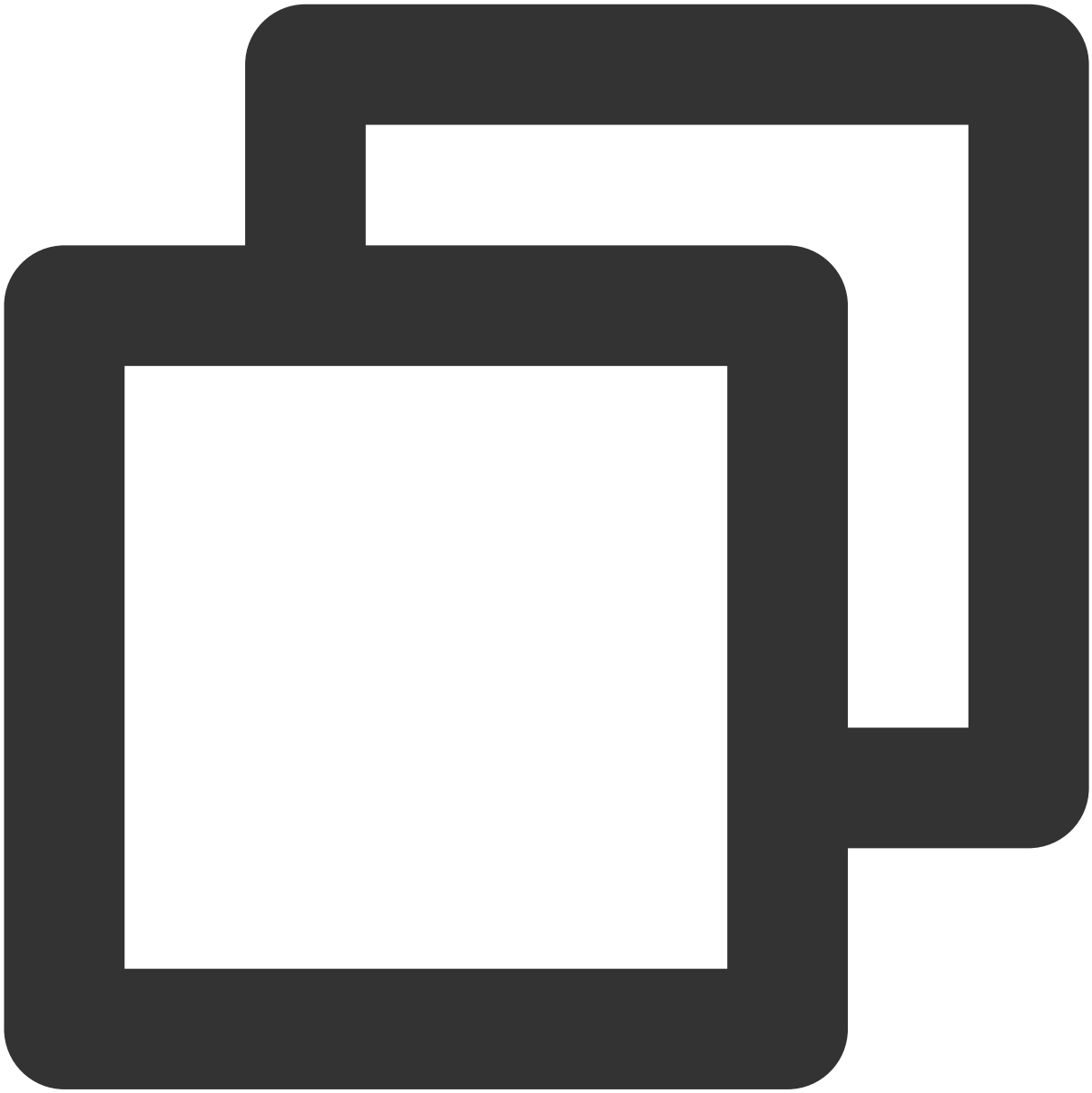


```
TUIRoomDefine.Request openRemoteDeviceByAdmin(String userId,
                                              TUIRoomDefine.MediaDevice device,
                                              int timeout,
                                              TUIRoomDefine.RequestCallback callbac
```


参数	类型	含义
userId	String	用户 ID
device	TUIRoomDefine.MediaDevice	设备
timeout	int	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
callback	TUIRoomDefine.ActionCallback	操作结果回调

closeRemoteDeviceByAdmin

关闭远端用户媒体设备，只有管理员或群主能够调用

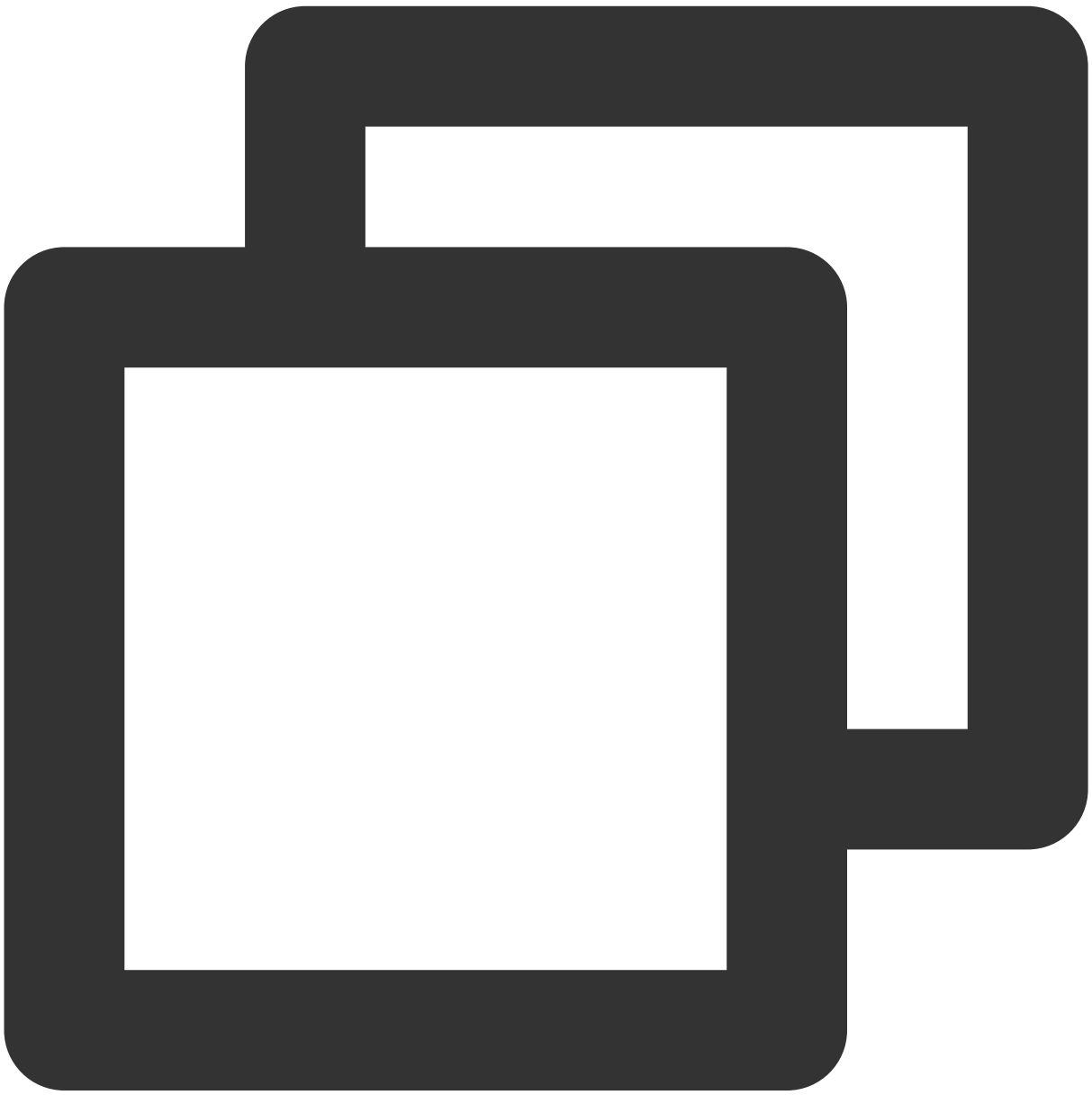


```
void closeRemoteDeviceByAdmin(String userId,
                              TUIRoomDefine.MediaDevice device,
                              TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
device	TUIRoomDefine.MediaDevice	设备
isDisable	boolean	是否禁用
callback	TUIRoomDefine.ActionCallback	操作结果回调

applyToAdminToOpenLocalDevice

全体用户媒体设备管理lock



```
TUIRoomDefine.Request applyToAdminToOpenLocalDevice(TUIRoomDefine.MediaDevice devic
                                                    int timeout,
                                                    TUIRoomDefine.RequestCallback c
```

参数	类型	含义

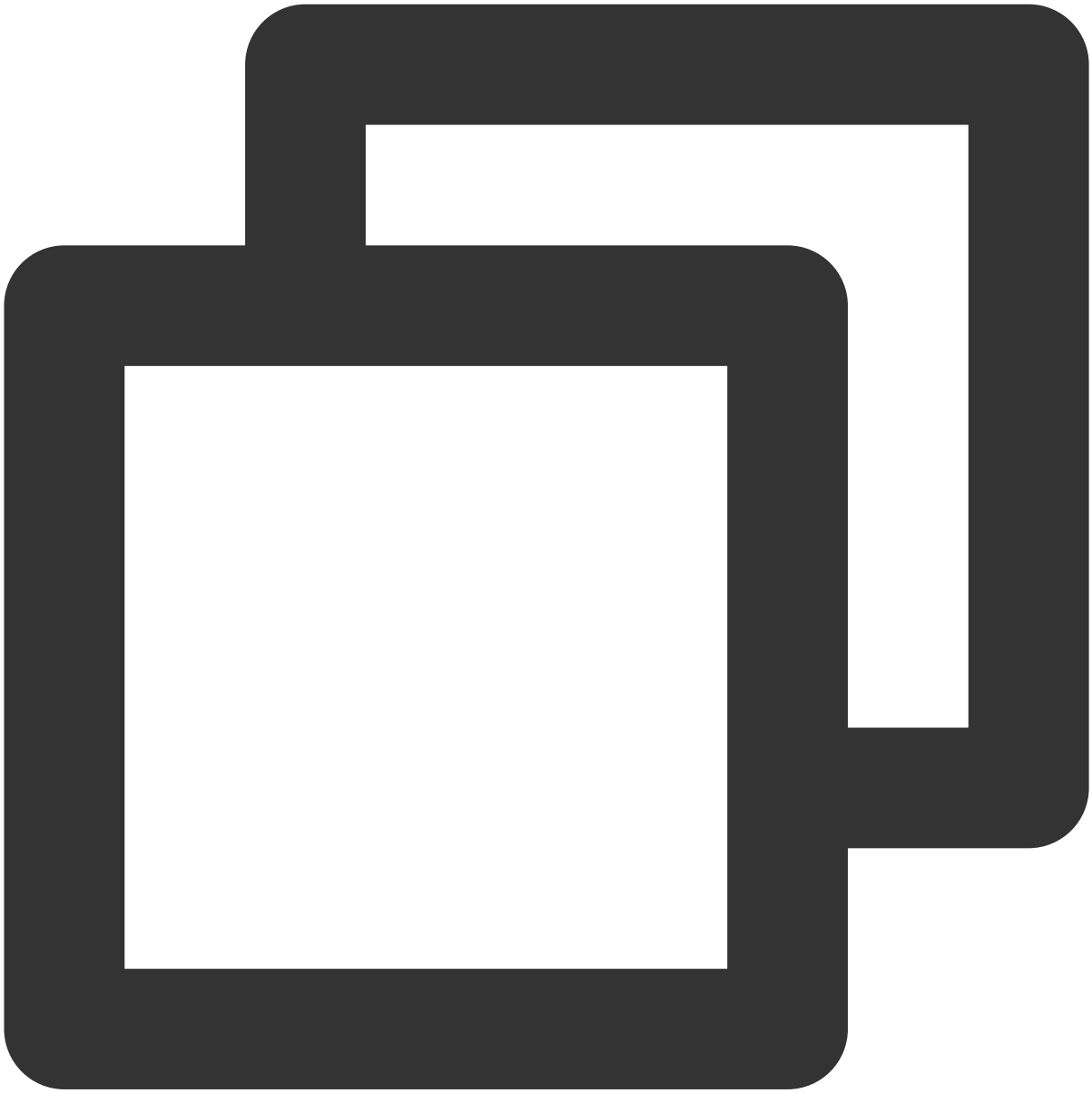
device	TUIRoomDefine.MediaDevice	设备
timeout	int	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
callback	TUIRoomDefine.ActionCallback	操作结果回调

setMaxSeatCount

设置最大麦位数，仅支持进房前和创建房间时设置

roomType 为 RoomType.CONFERENCE (教育及会议场景) 时，maxSeatCount 值不做限制；

roomType 为 RoomType.LIVE_ROOM (直播场景) 时，maxSeatCount 最大限制为 16；

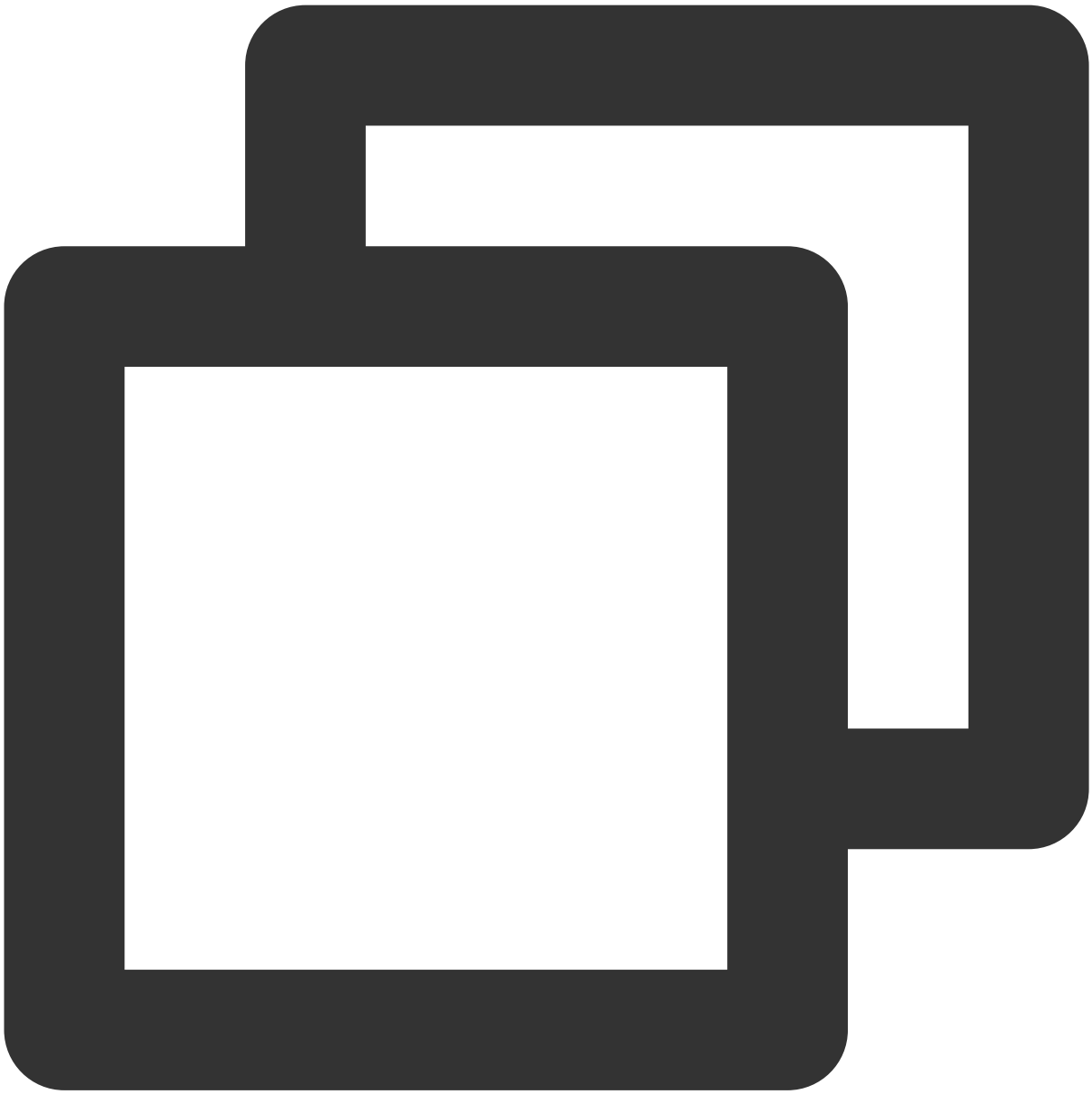


```
void setMaxSeatCount(int maxSeatCount, TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
maxSeatCount	int	最大麦位数
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

lockSeatByAdmin

锁定麦位（包括位置锁定，音频状态锁定，视频状态锁定）



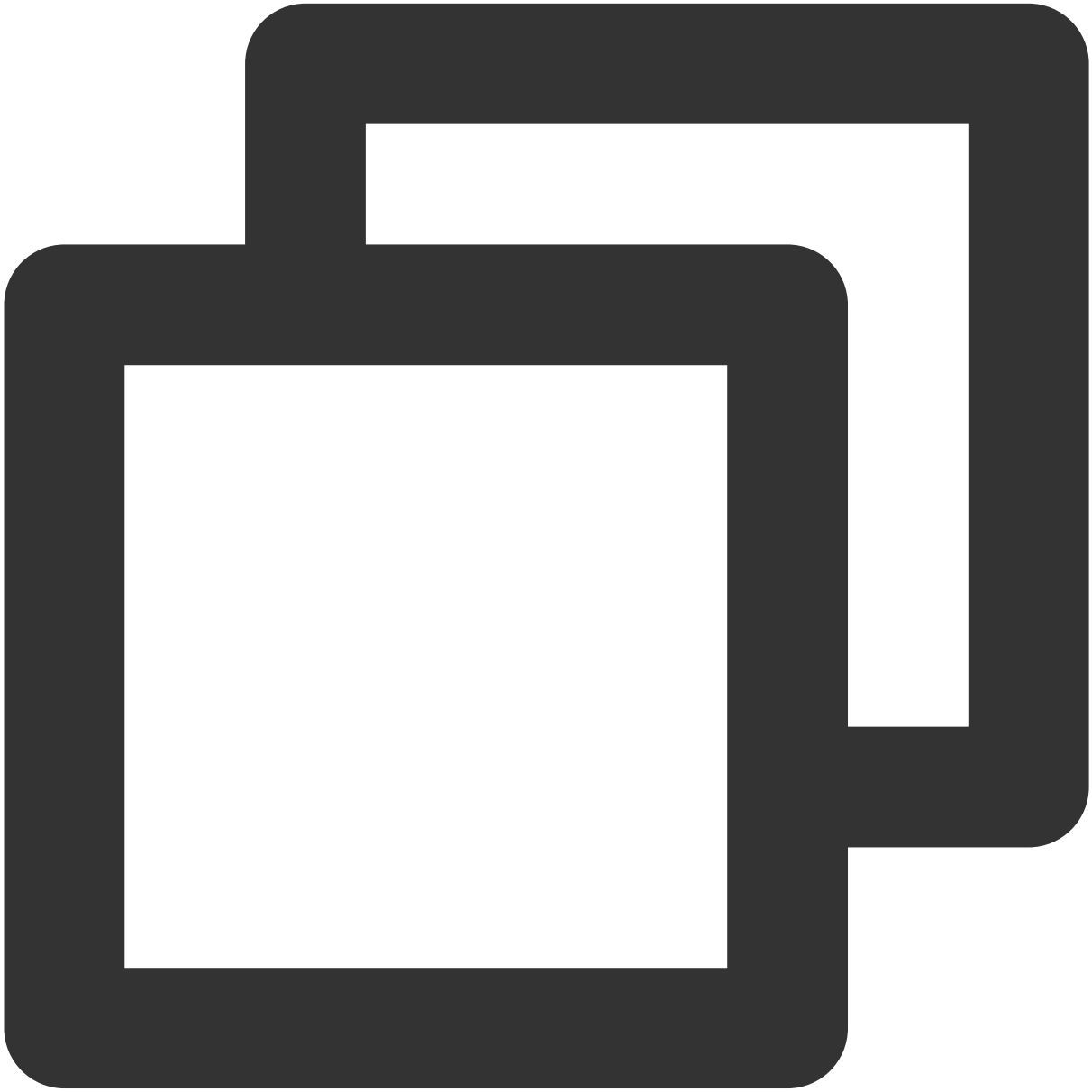
```
void lockSeatByAdmin(int seatIndex,
                     TUIRoomDefine.SeatLockParams lockParams,
                     TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
seatIndex	int	麦位编号
lockParams	TUIRoomDefine.SeatLockParams	锁麦参数

callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

getSeatList

获取麦位列表



```
void getSeatList(TUIRoomDefine.GetSeatListCallback callback);
```

参数	类型	含义

callback	TUIRoomDefine.GetSeatListCallback	获取麦位列表回调

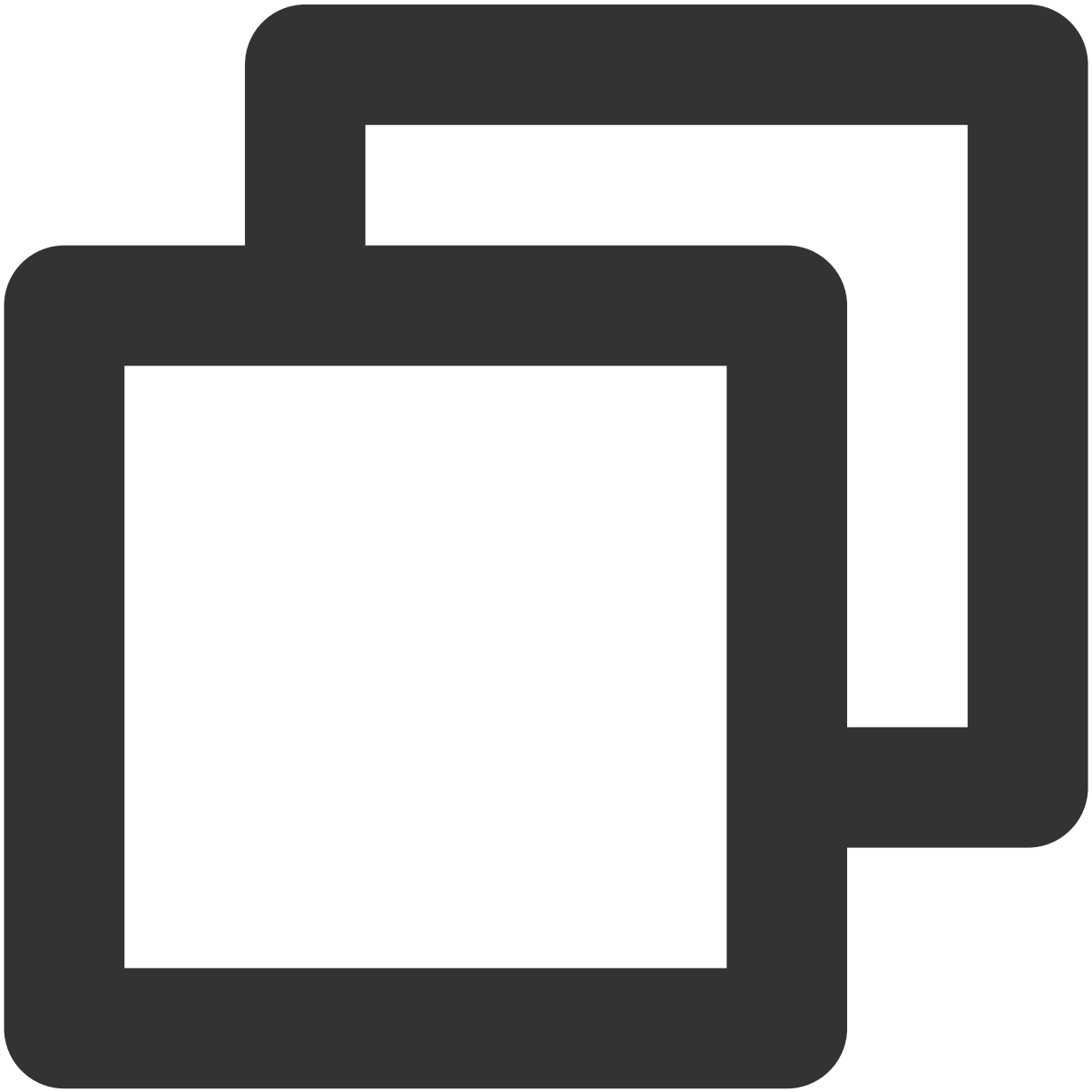
takeSeat

本地上麦

说明：

会议场景：SPEAK_AFTER_TAKING_SEAT模式需要向主持人或管理员发起申请才允许上麦，其他模式不支持上麦。

直播场景：FREE_TO_SPEAK模式可以自由上麦，上麦后开麦发言；SPEAK_AFTER_TAKING_SEAT模式需要向主持人或管理员发起申请才允许上麦；其他模式不支持上麦



```
TUIRoomDefine.Request takeSeat(int seatIndex,
                               int timeout,
                               TUIRoomDefine.RequestCallback callback);
```

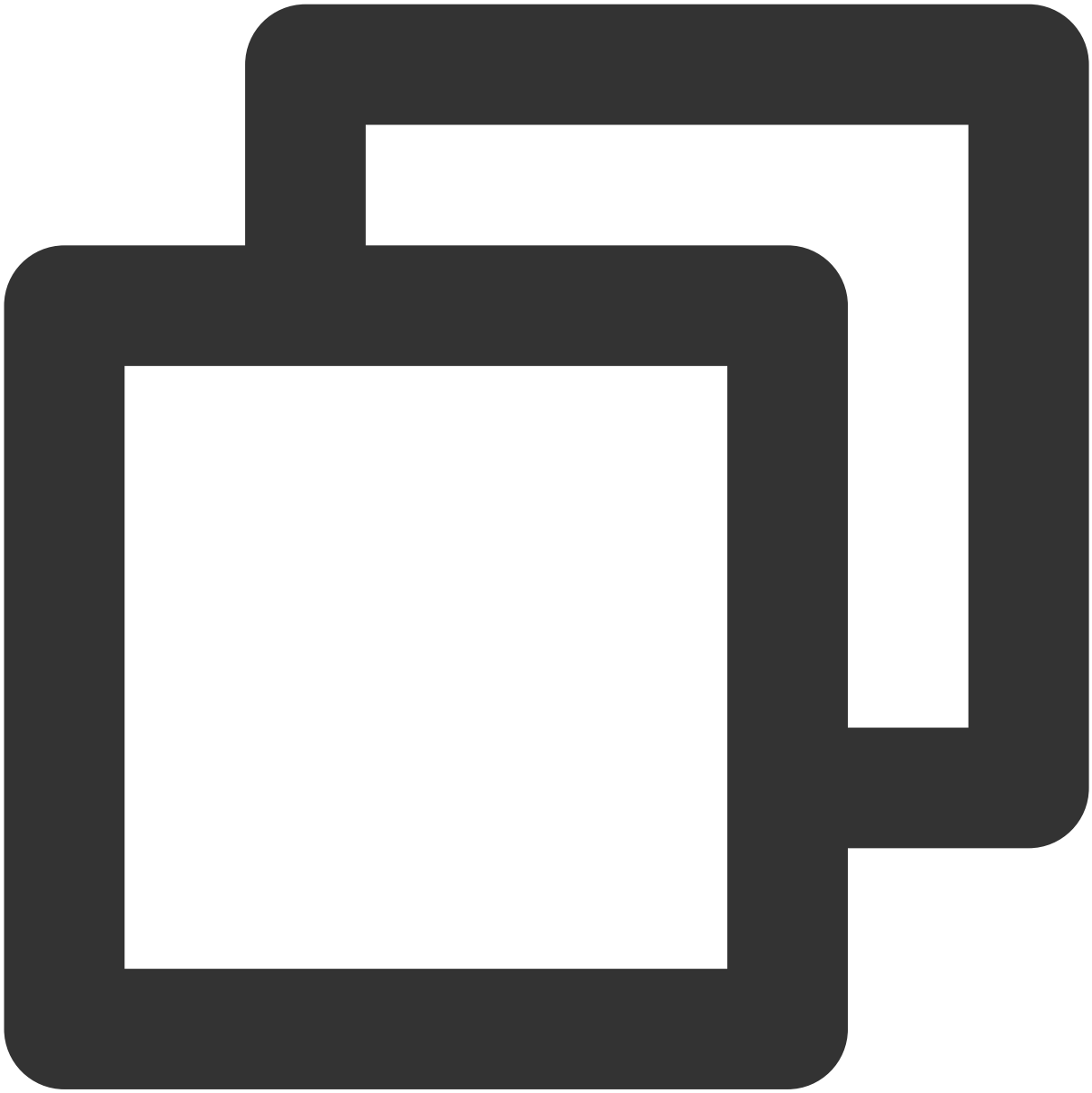
参数：

参数	类型	含义
seatIndex	int	麦位编号
timeout	int	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
callback	TUIRoomDefine.RequestCallback	调用接口的回调，用于通知请求的回调状态

Return: 请求体

leaveSeat

本地下麦



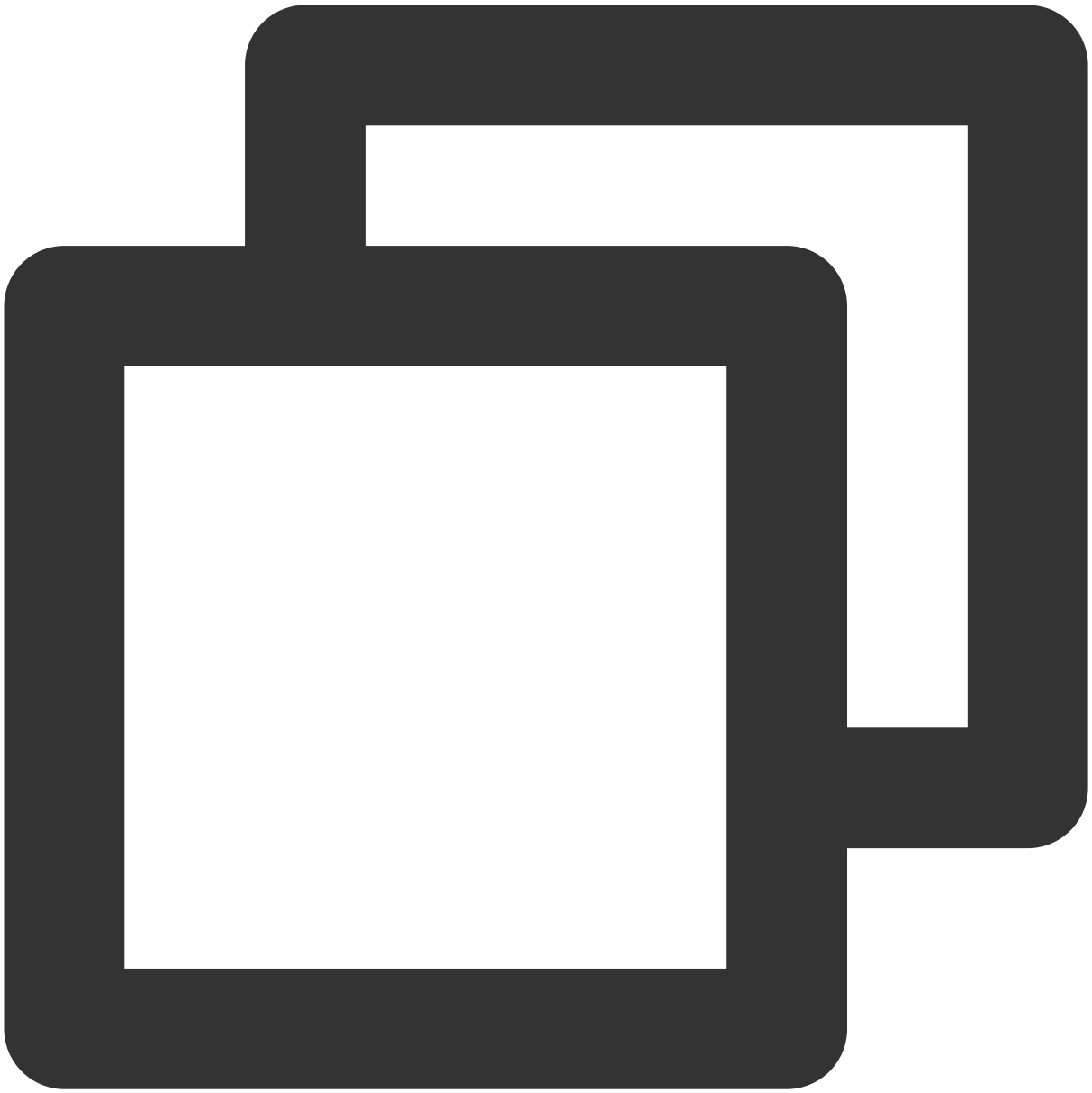
```
void leaveSeat(TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

takeUserOnSeatByAdmin

主持人/管理员 邀请用户上麦



```
TUIRoomDefine.Request takeUserOnSeatByAdmin(int seatIndex,
                                            String userId,
                                            int timeout,
                                            TUIRoomDefine.RequestCallback callback)
```

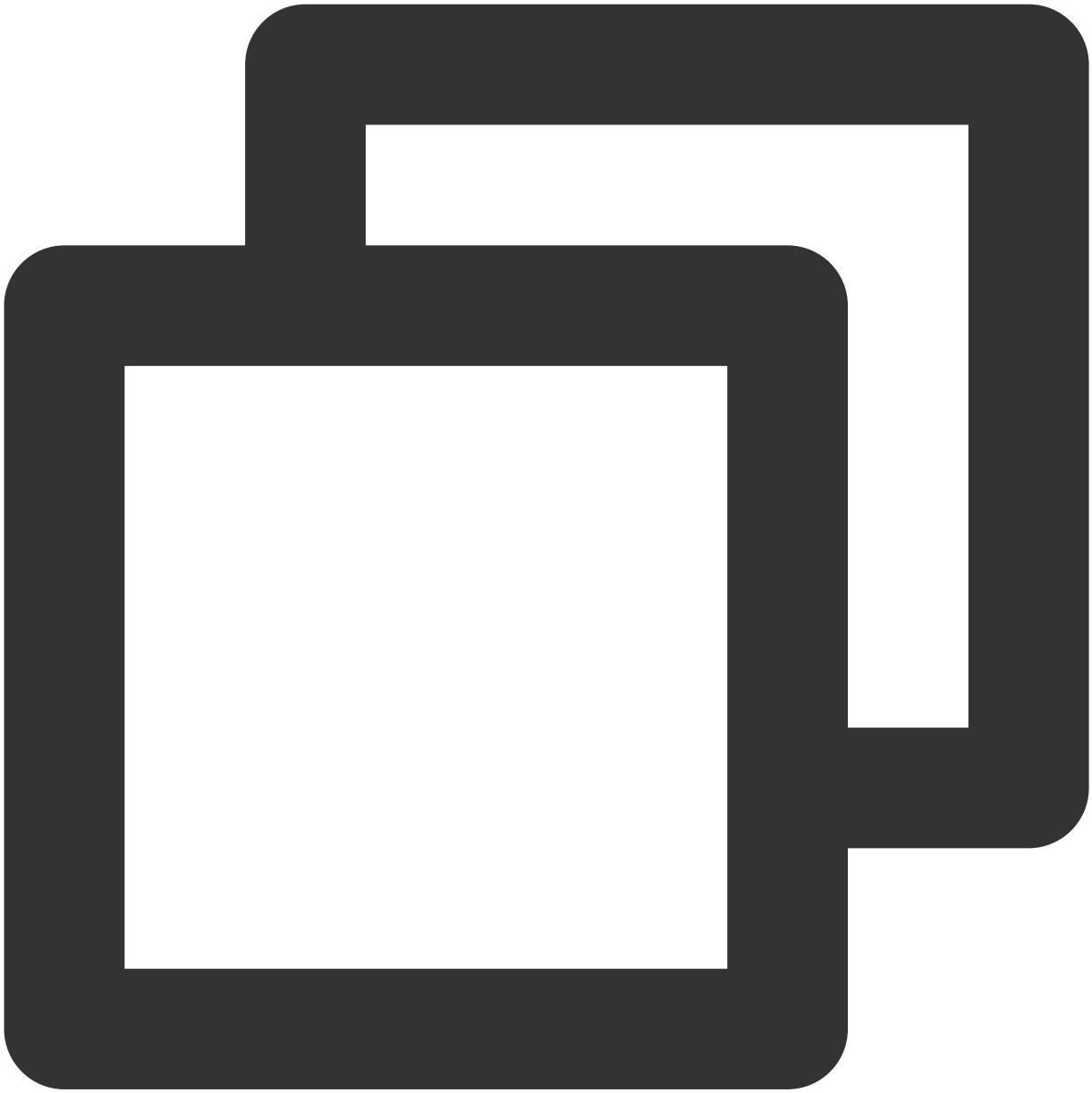
参数	类型	含义
seatIndex	int	麦位编号

userId	String	用户 ID
timeout	int	超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
callback	TUIRoomDefine.RequestCallback	调用接口的回调，用于通知请求的回调状态

Return: 请求体

kickUserOffSeatByAdmin

主持人/管理员 将用户下麦

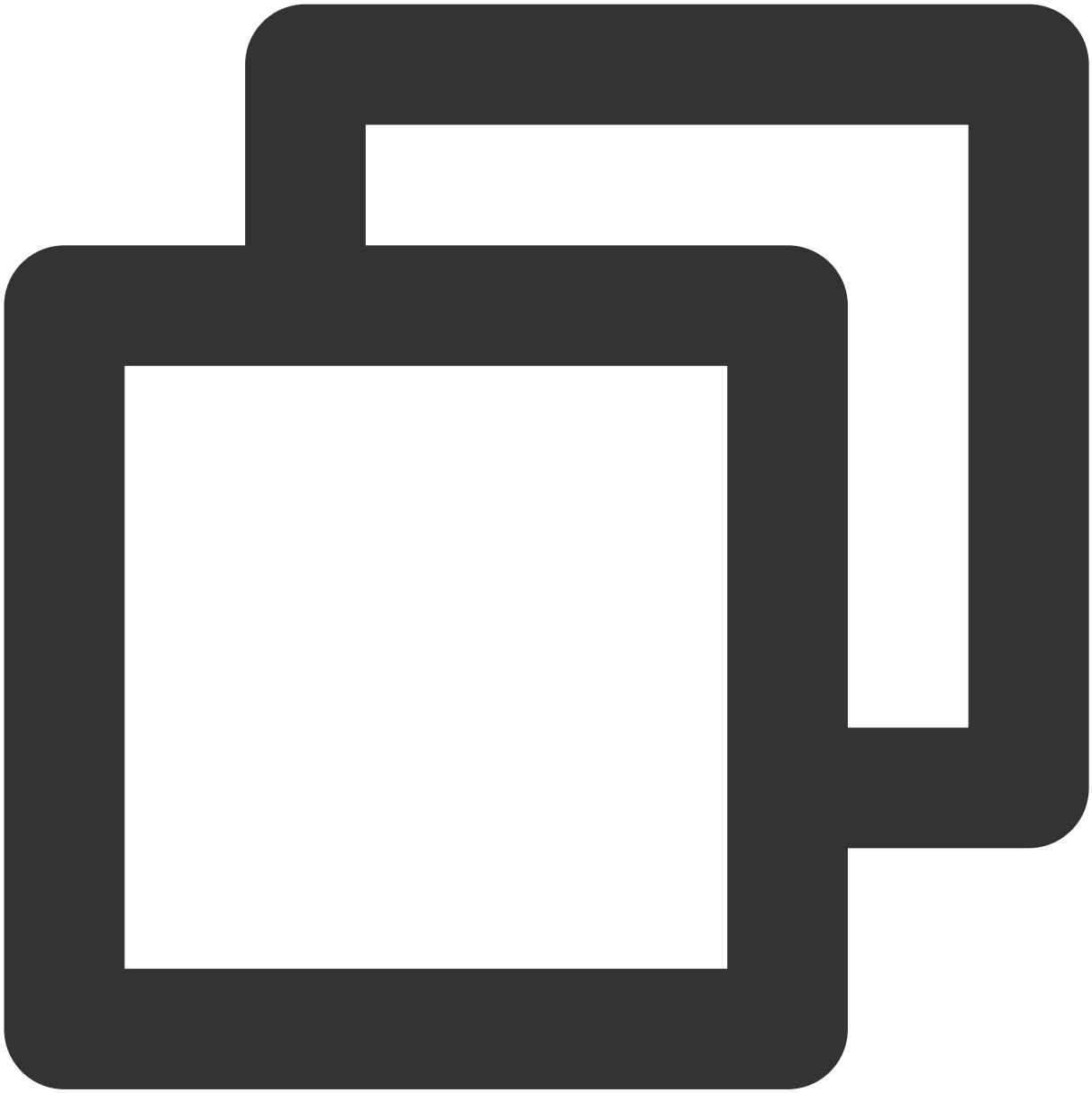


```
void kickUserOffSeatByAdmin(int seatIndex, String userId, TUIRoomDefine.ActionCallb
```

参数	类型	含义
seatIndex	int	麦位编号
userId	String	用户 ID
callback	TUIRoomDefine.RequestCallback	调用接口的回调，用于通知请求的回调状态

sendTextMessage

发送文本消息



```
void sendTextMessage(String message,
                     TUIRoomDefine.ActionCallback callback);
```

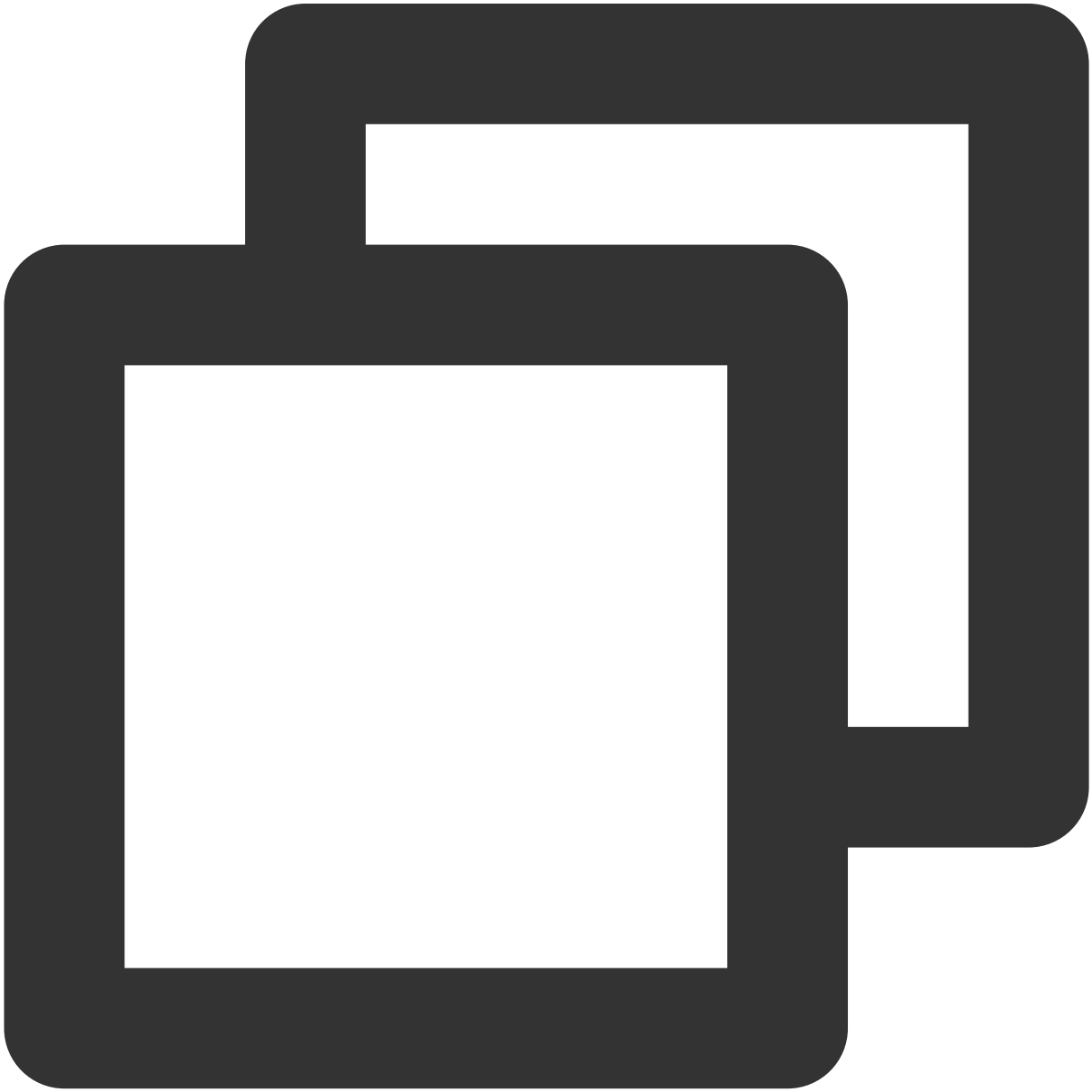
参数：

参数	类型	含义
message	String	文本消息内容

callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

sendCustomMessage

发送自定义消息



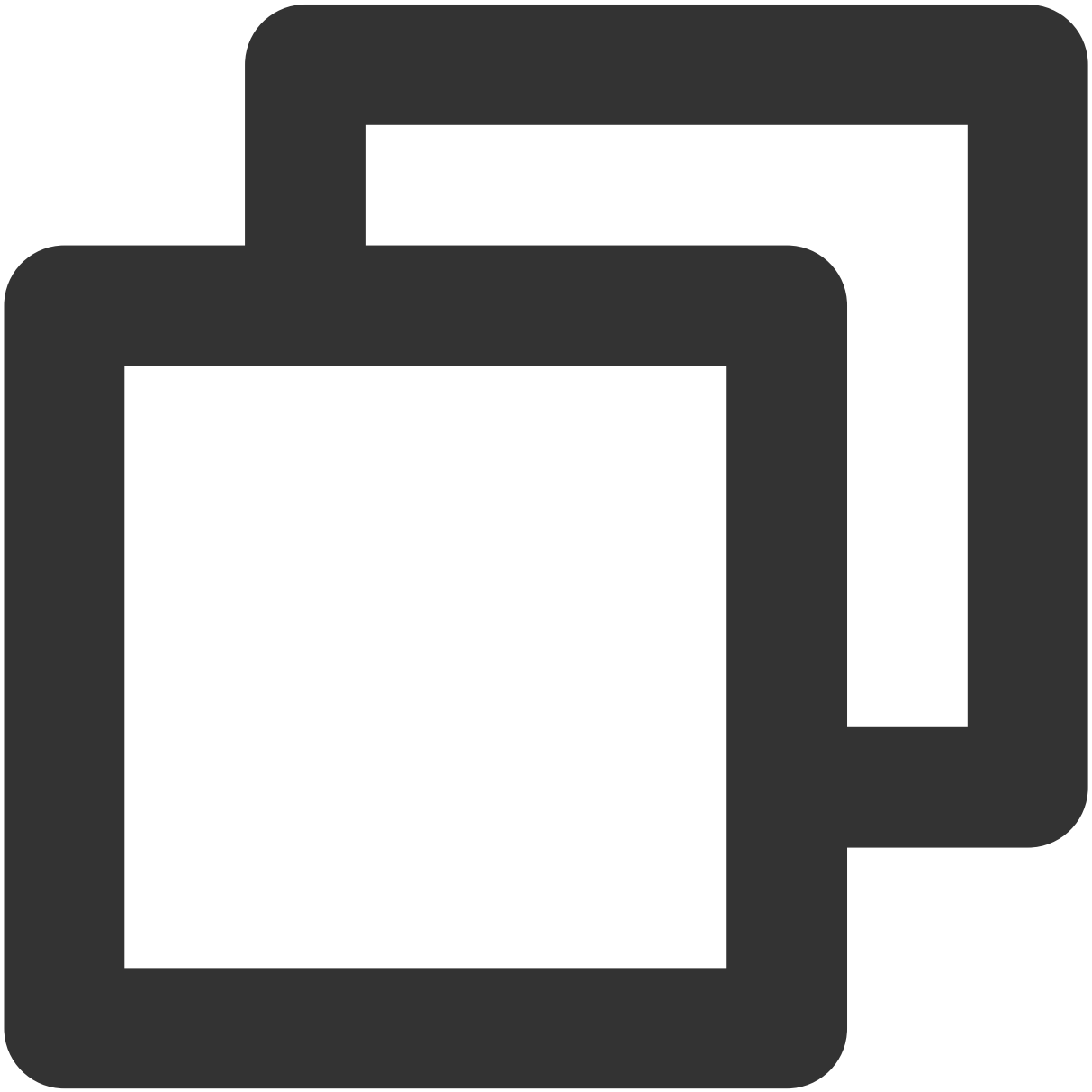
```
void sendCustomMessage(String message,
                       TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
message	String	自定义消息内容
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

disableSendingMessageByAdmin

禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）

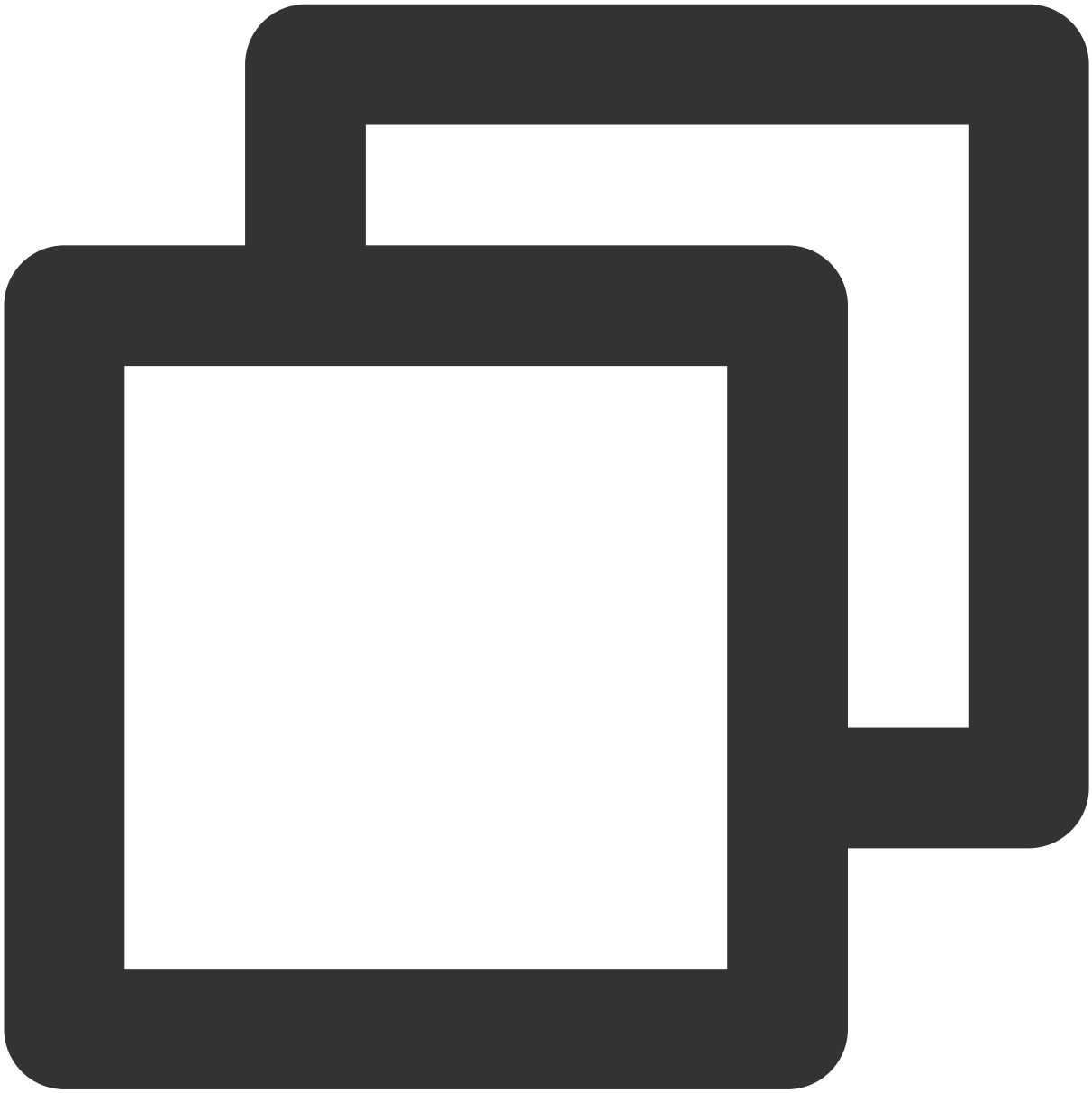


```
void disableSendingMessageByAdmin(String userId,
                                  boolean isDisable,
                                  TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
userId	String	用户 ID
isDisable	boolean	是否禁用
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

disableSendingMessageForAllUser

禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）

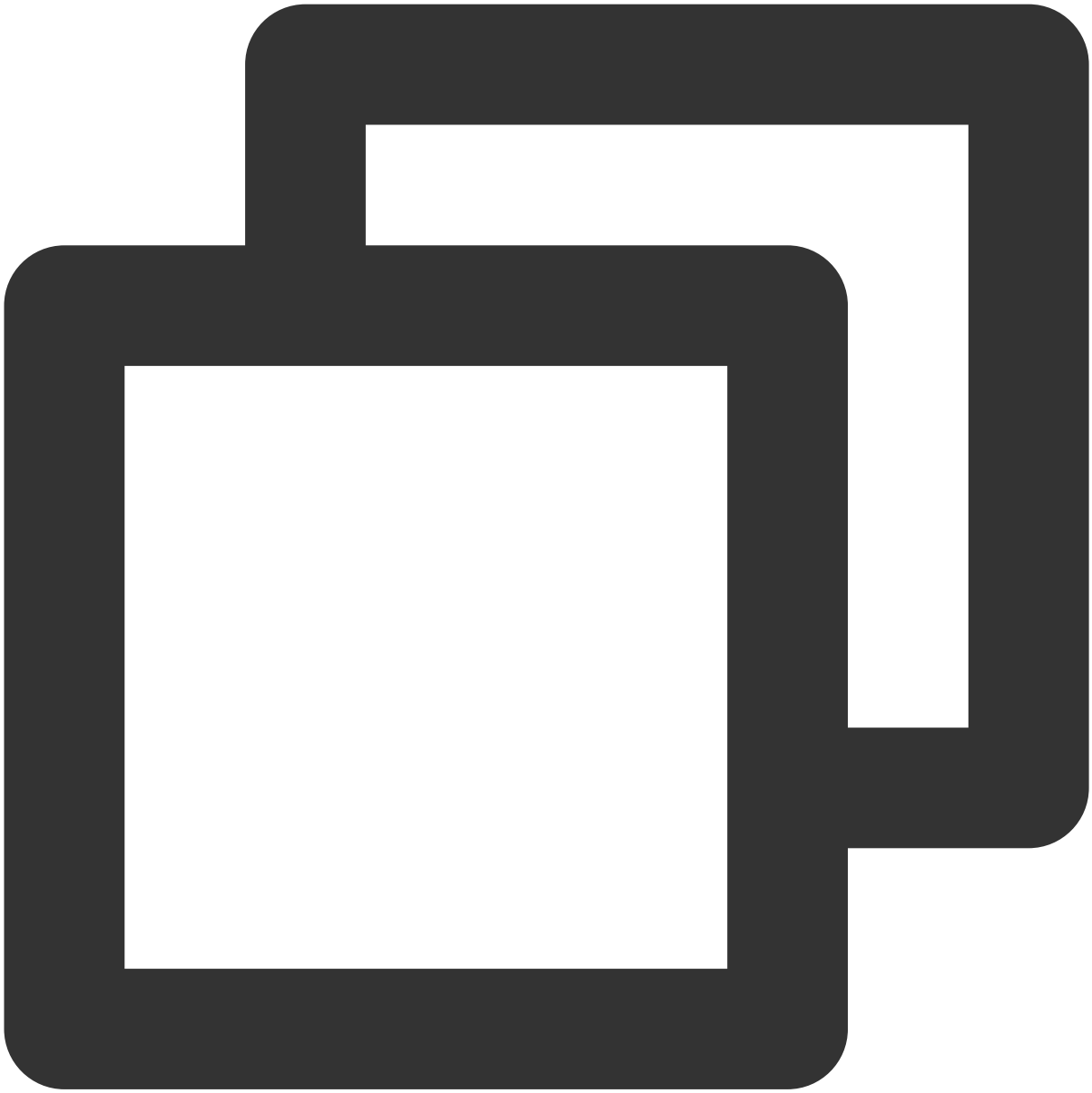


```
void disableSendingMessageForAllUser(boolean isDisable,
                                     TUIRoomDefine.ActionCallback callback);
```

参数	类型	含义
isDisable	boolean	是否禁用
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

cancelRequest

取消已经发出的请求



```
void cancelRequest(String requestId, TUIRoomDefine.ActionCallback callback);
```

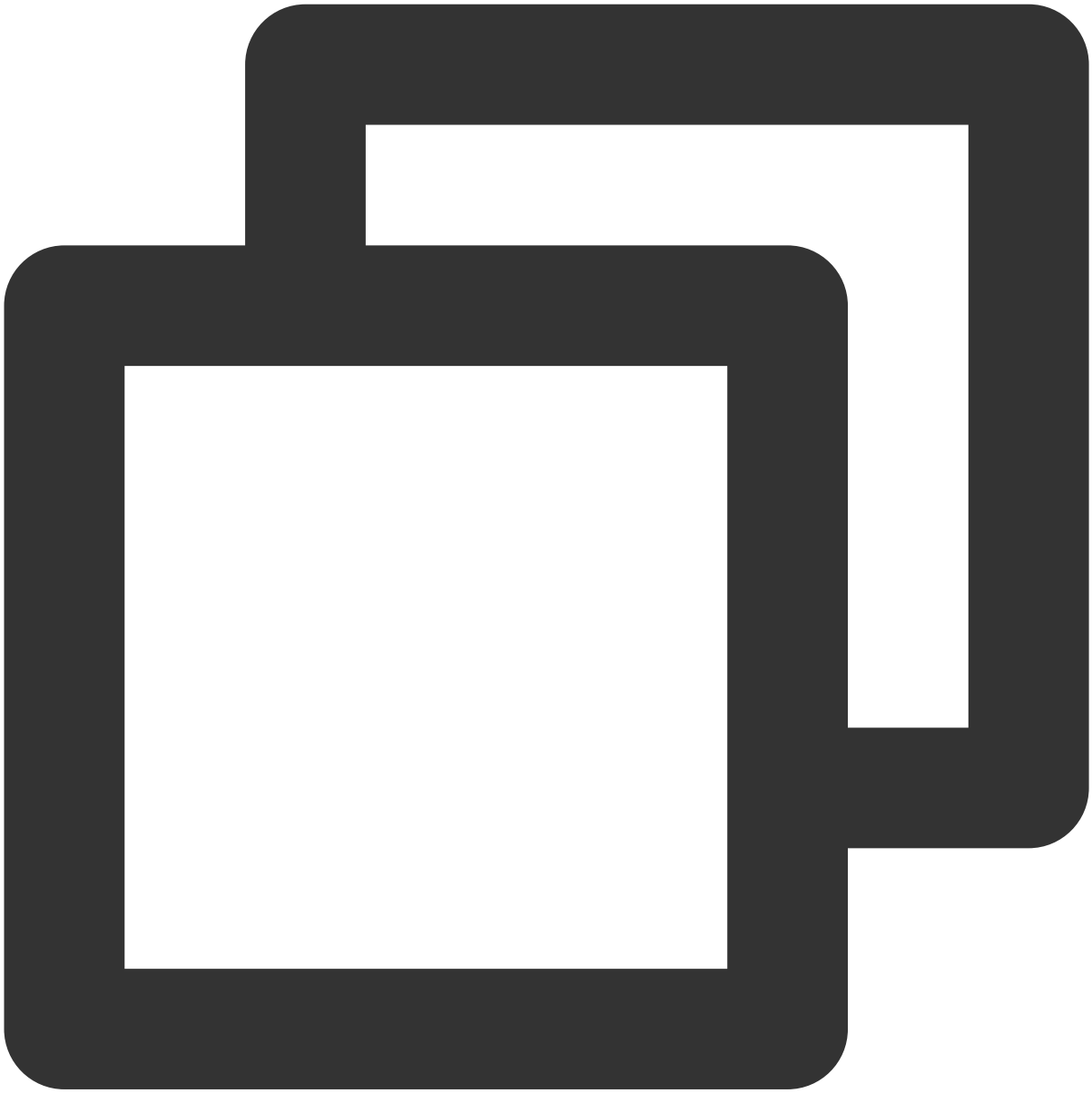
参数：

参数	类型	含义
requestId	String	请求 ID
callback	TUIRoomDefine.ActionCallback	调用接口的回调，用于通知接口调用

		的成功或者失败

responseRemoteRequest

回复远端用户的请求



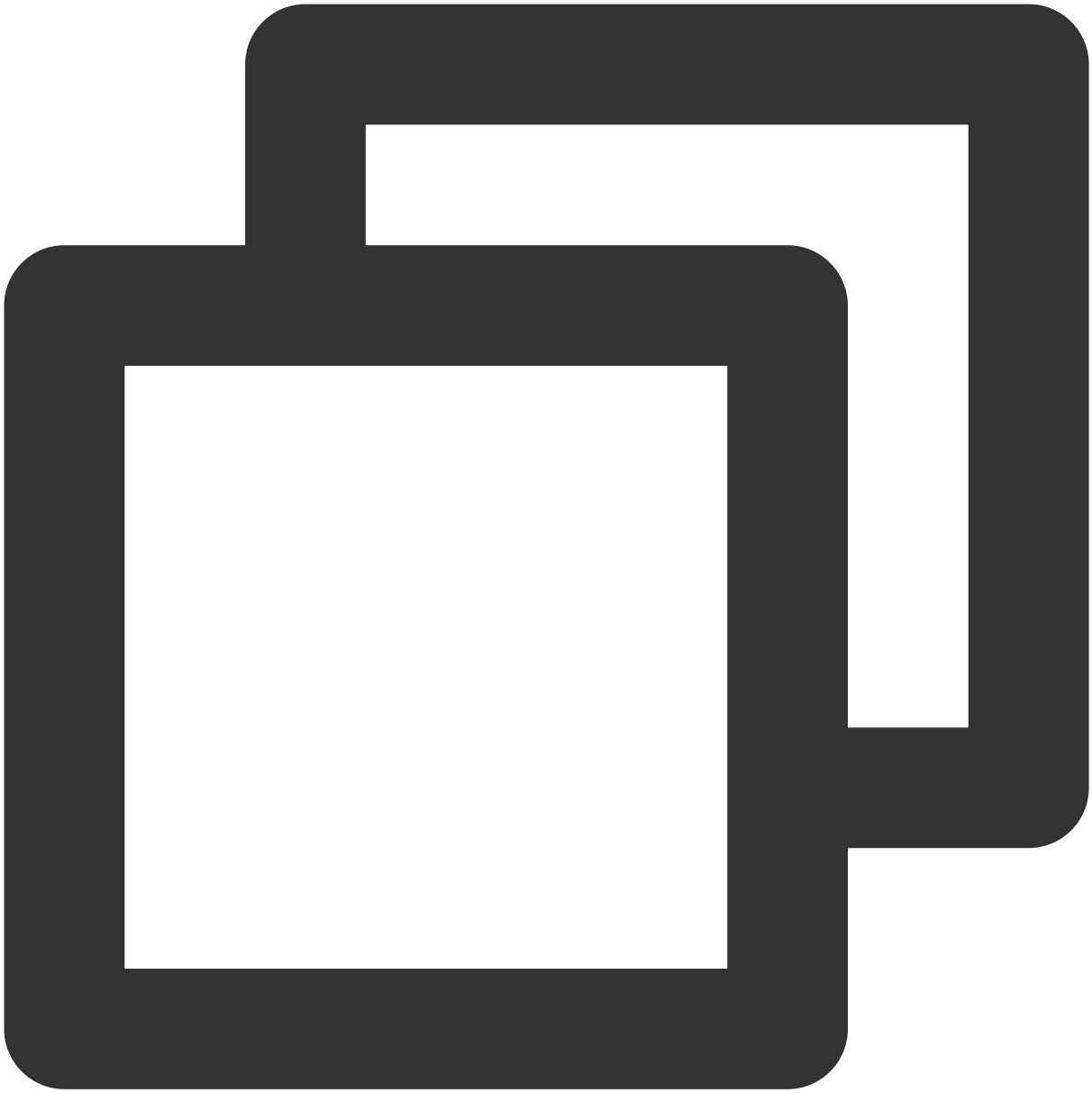
```
void responseRemoteRequest(String requestId,
                           boolean agree,
                           TUIRoomDefine.ActionCallback callback);
```

参数：

参数	类型	含义
requestId	String	请求 ID
agree	boolean	是否同意
callback	TUIRoomEngineDef.ActionCallback	调用接口的回调，用于通知接口调用的成功或者失败

getTRTCCloud

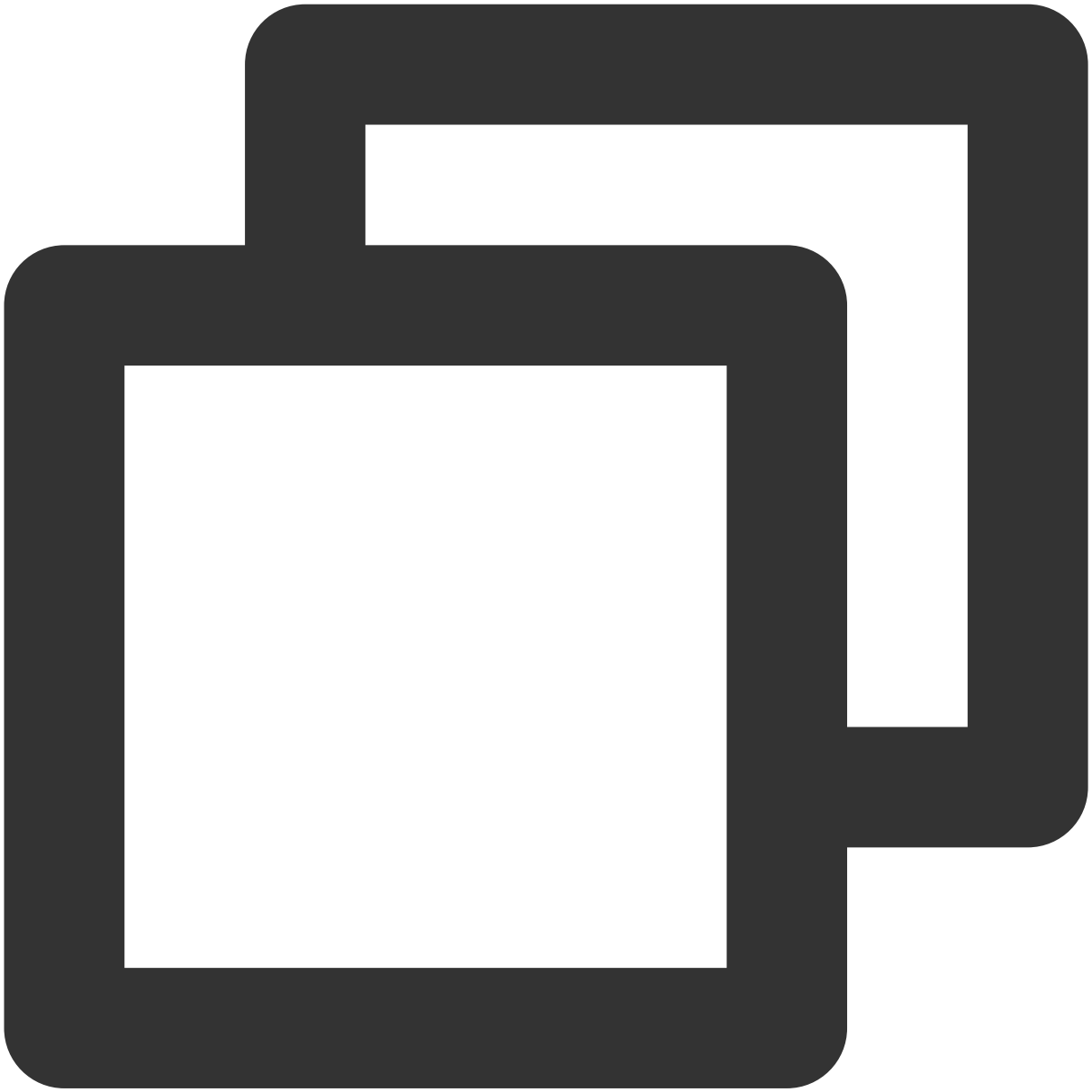
获取TRTCCloud实例



```
TRTCCloud getTRTCCloud();
```

getDeviceManager

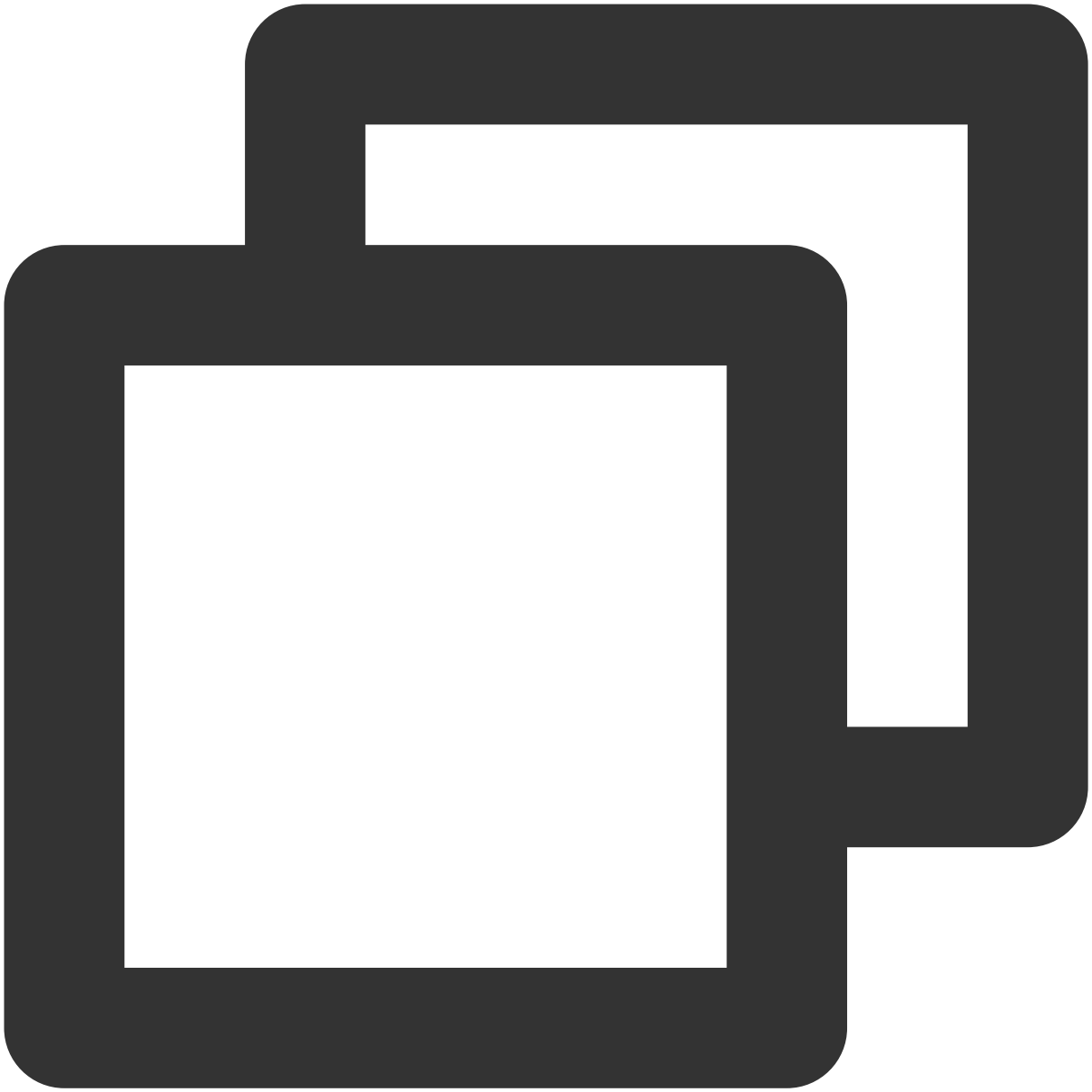
获取 deviceManager, 可使用 deviceManager 的方法获取设备列表，甚至设备，切换设备等功能。



```
TXDeviceManager getDeviceManager();
```

getAudioEffectManager

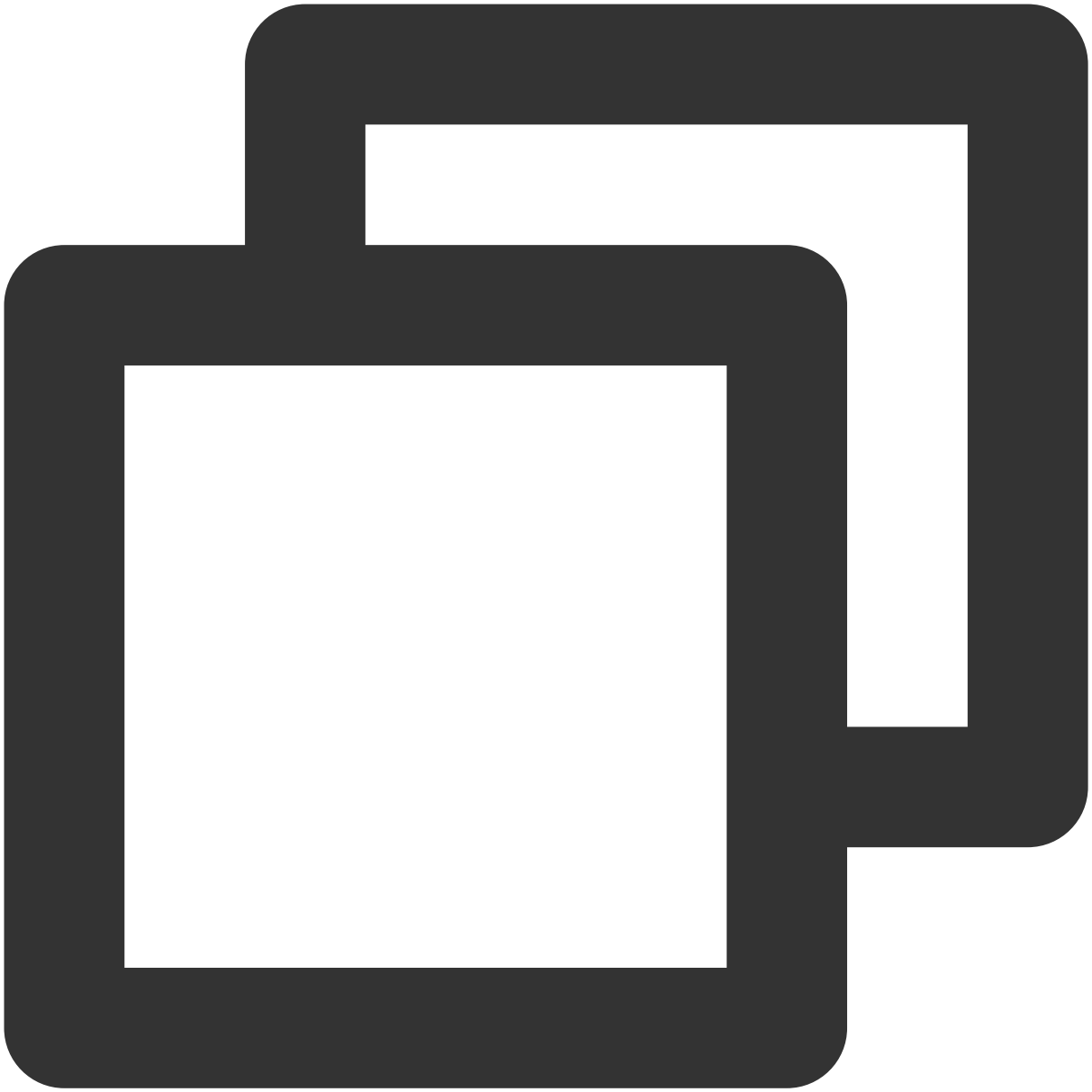
获取音频管理



```
TXAudioEffectManager getAudioEffectManager();
```

getBeautyManager

获取美颜管理对象



```
TXBeautyManager getBeautyManager();
```

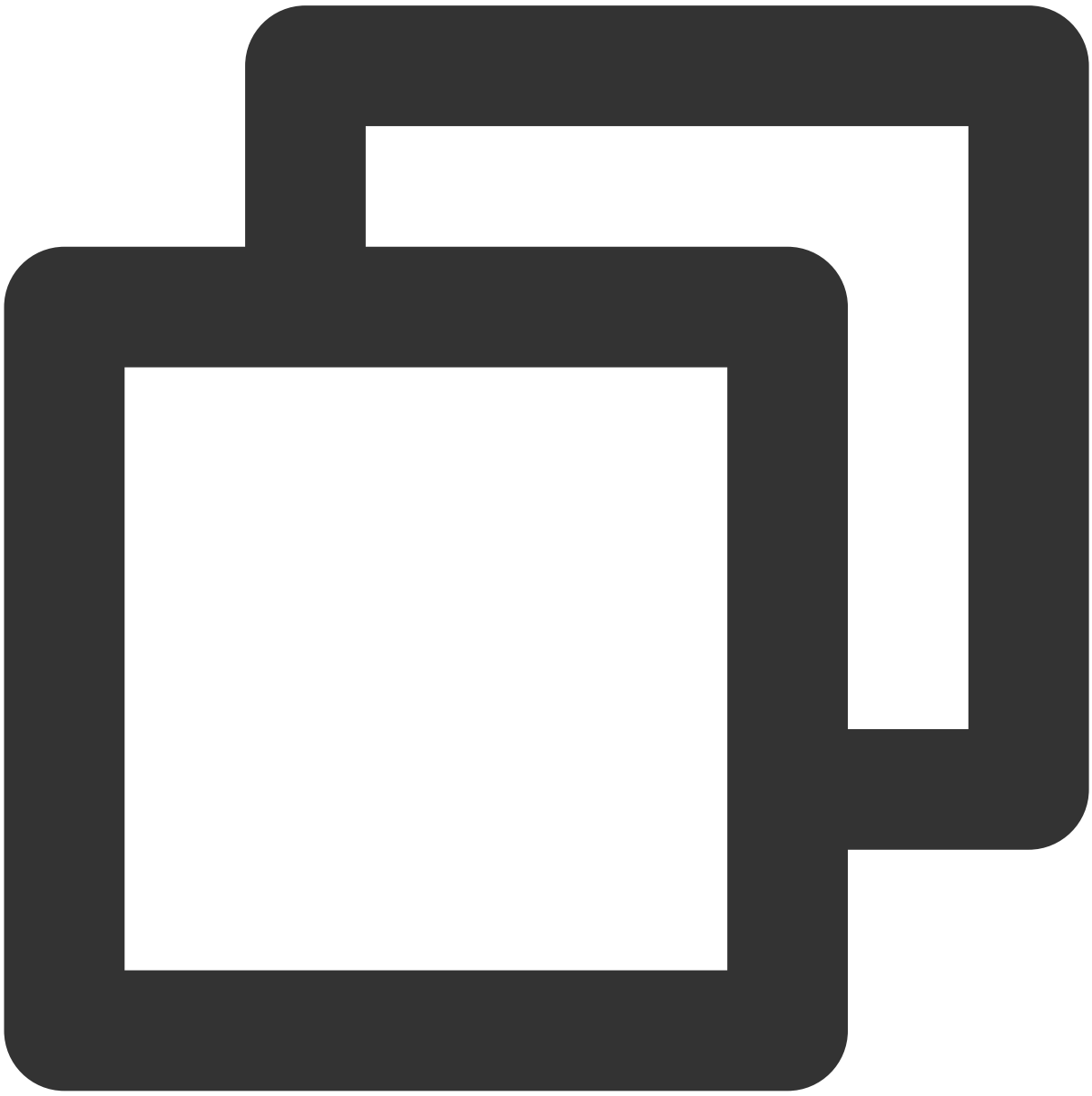
TUIRoomObserver

最近更新时间：2023-10-24 17:07:05

TUIRoomEngine 事件回调

onError

错误事件

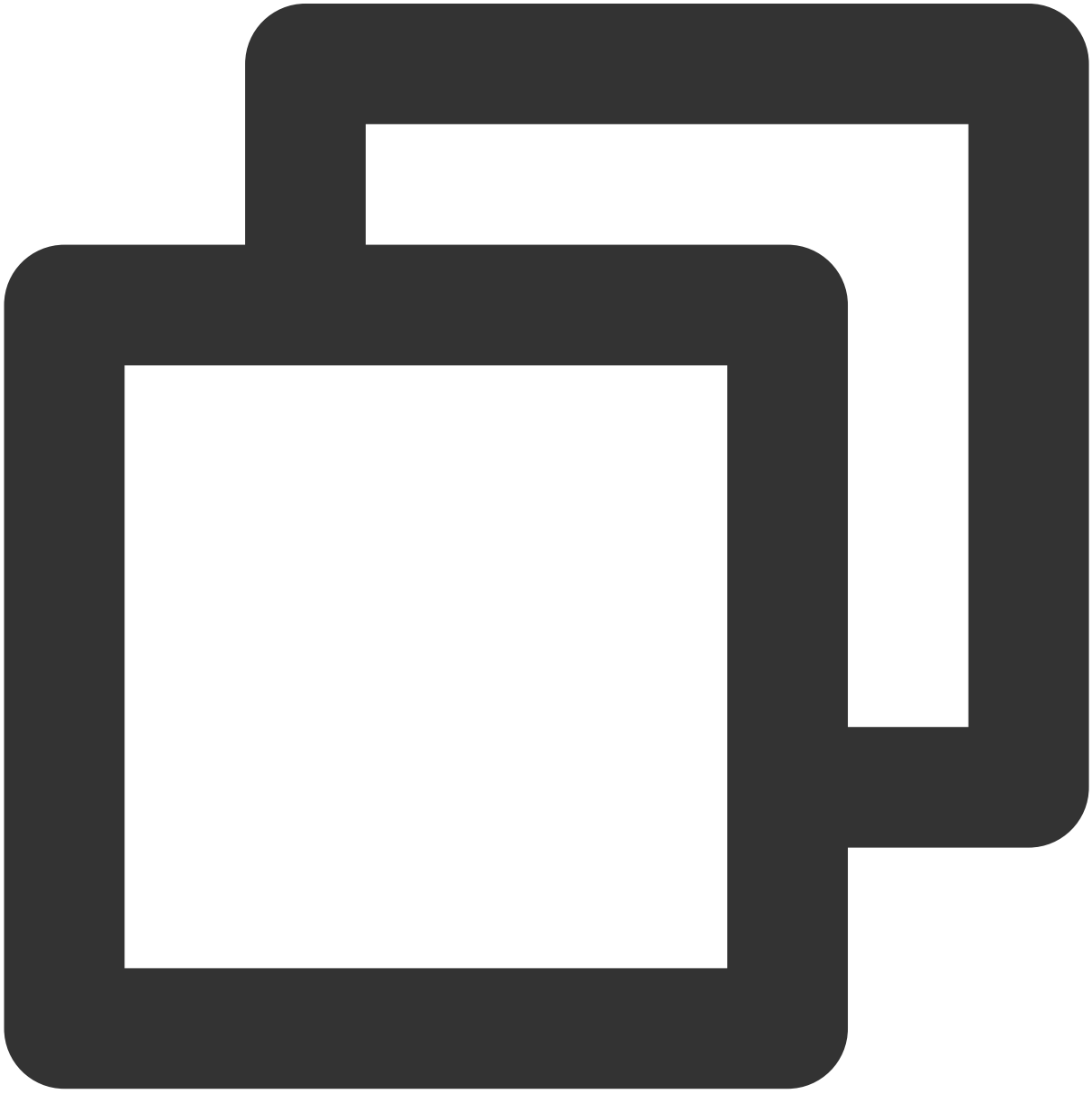


```
void onError(TUICommonDefine.Error errorCode, String message)
```

参数	类型	描述
errorCode	TUICommonDefine.Error	错误码
message	String	错误信息

onKickedOffLine

用户被踢下线事件

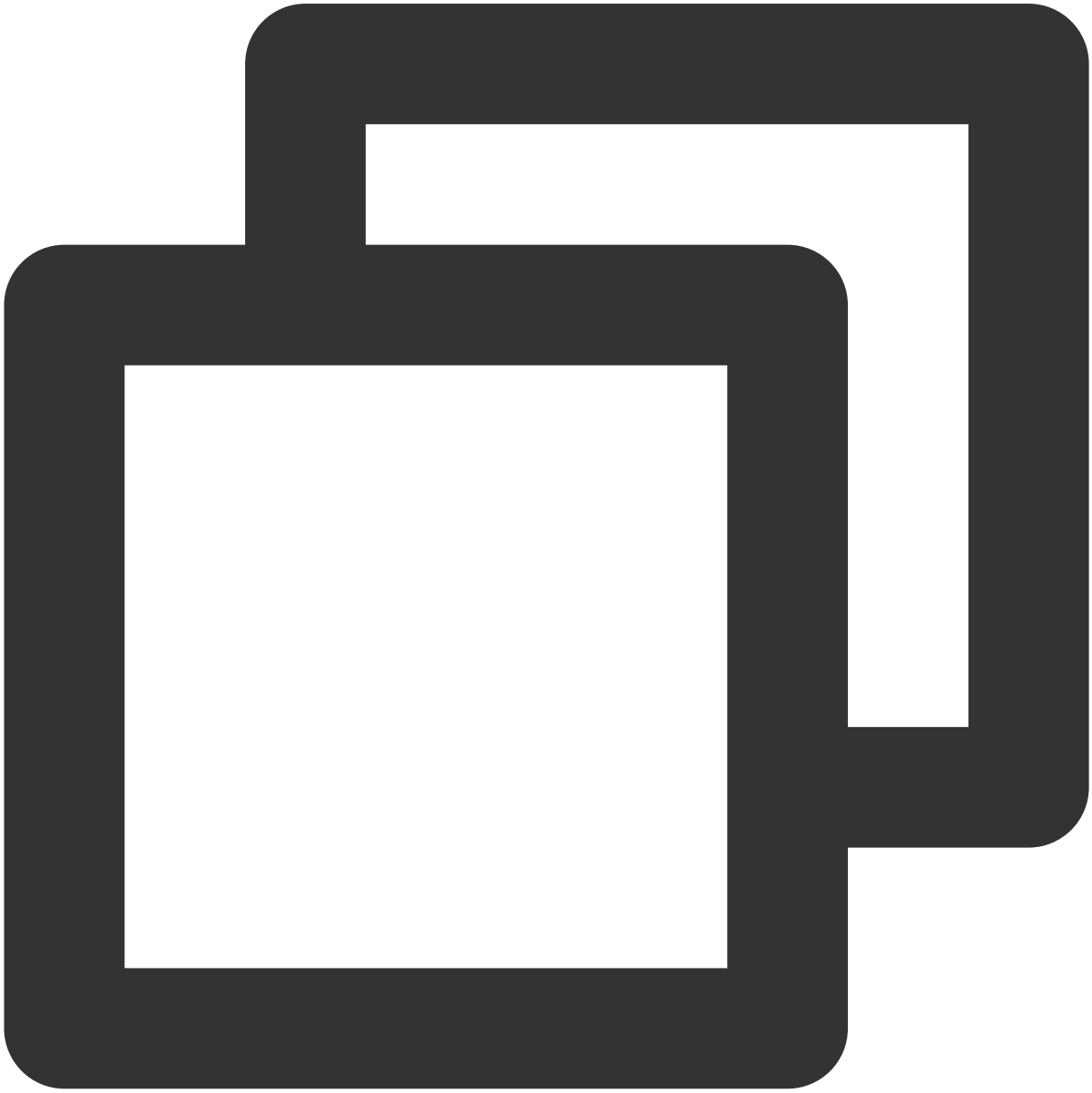


```
void onKickedOffLine(String message)
```

参数	类型	描述
message	String	被踢下线的描述

onUserSigExpired

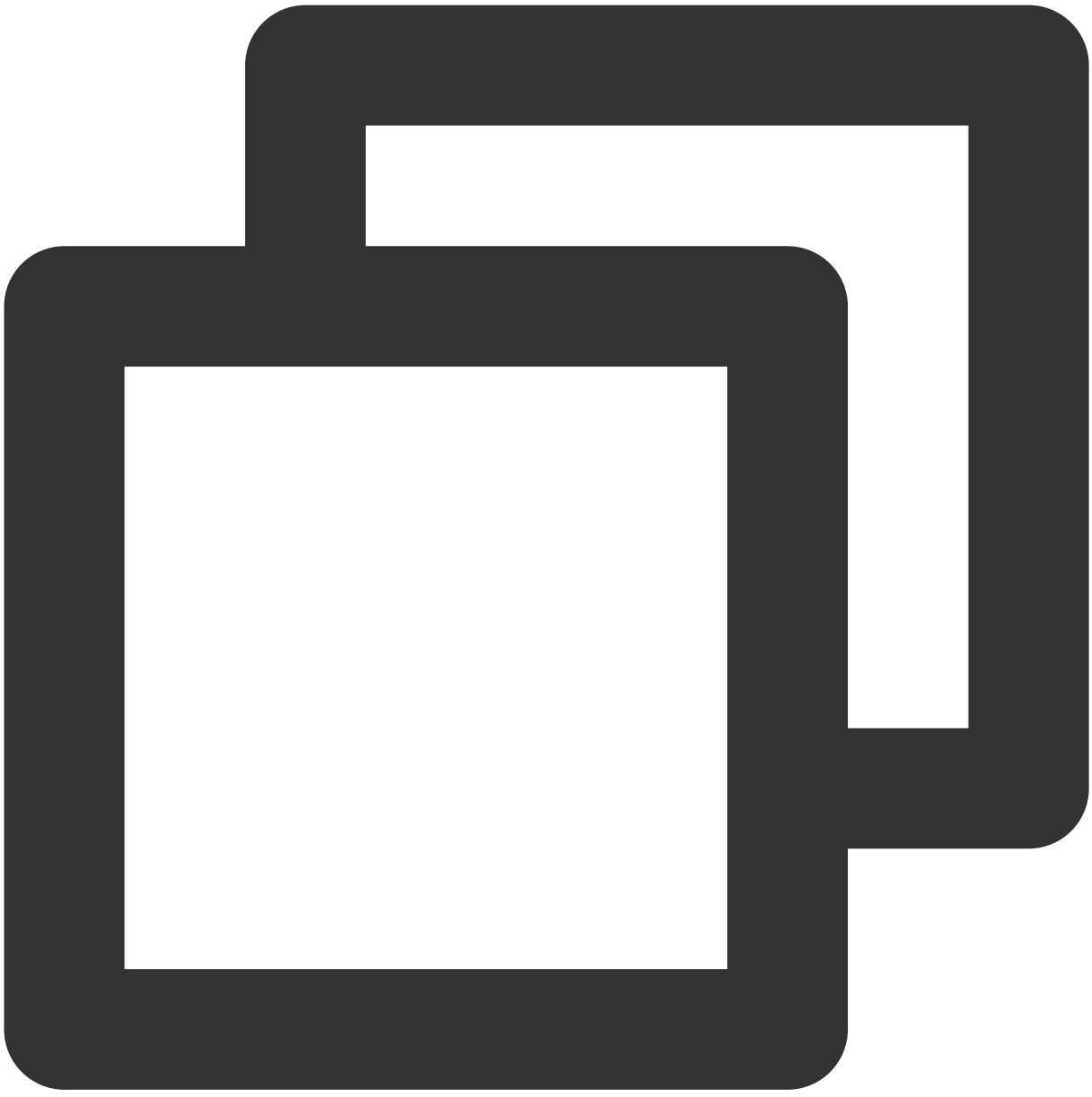
`userSig` 过期事件



```
void onUserSigExpired()
```

onRoomNameChanged

房间名称更改事件



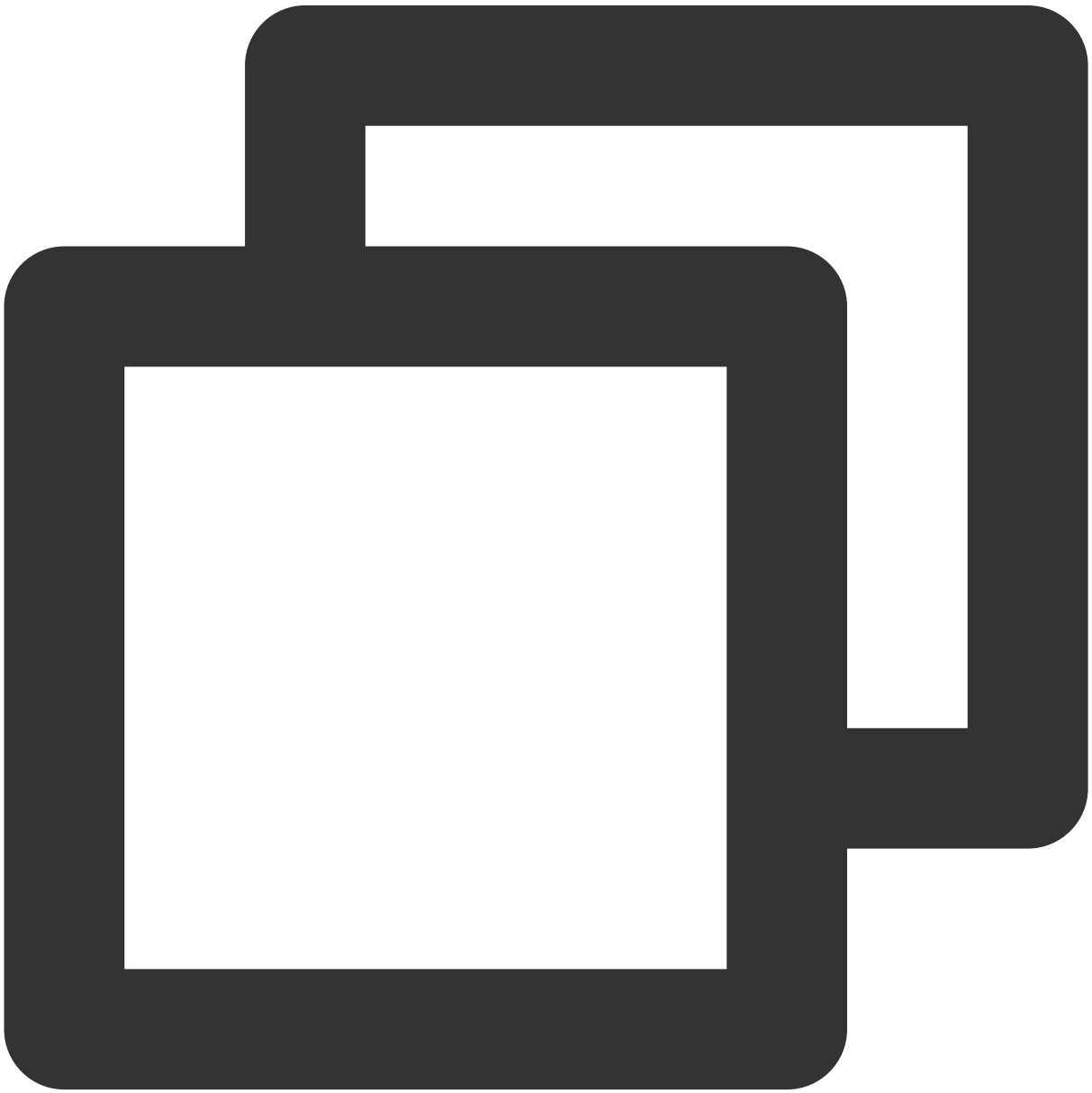
```
void onRoomNameChanged(String roomId, String roomName)
```

参数	类型	描述

roomId	String	房间 ID
roomName	String	房间名称

onAllUserMicrophoneDisableChanged

房间内所有用户麦克风被禁用事件

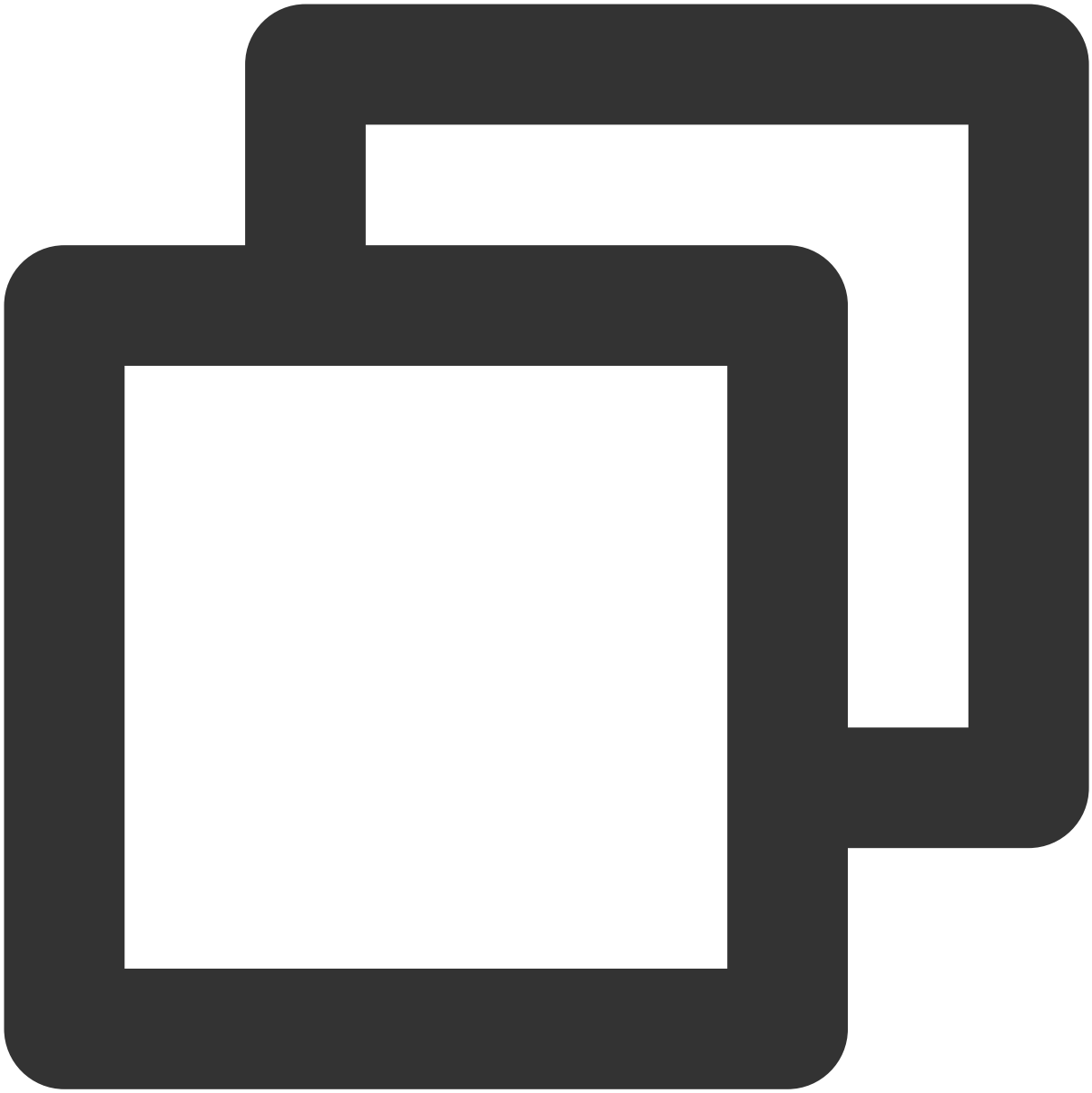


```
void onAllUserMicrophoneDisableChanged(String roomId, boolean isDisable)
```

参数	类型	描述
roomId	String	房间 ID
isDisable	boolean	是否被禁用

onAllUserCameraDisableChanged

房间内所有用户摄像头被禁用事件

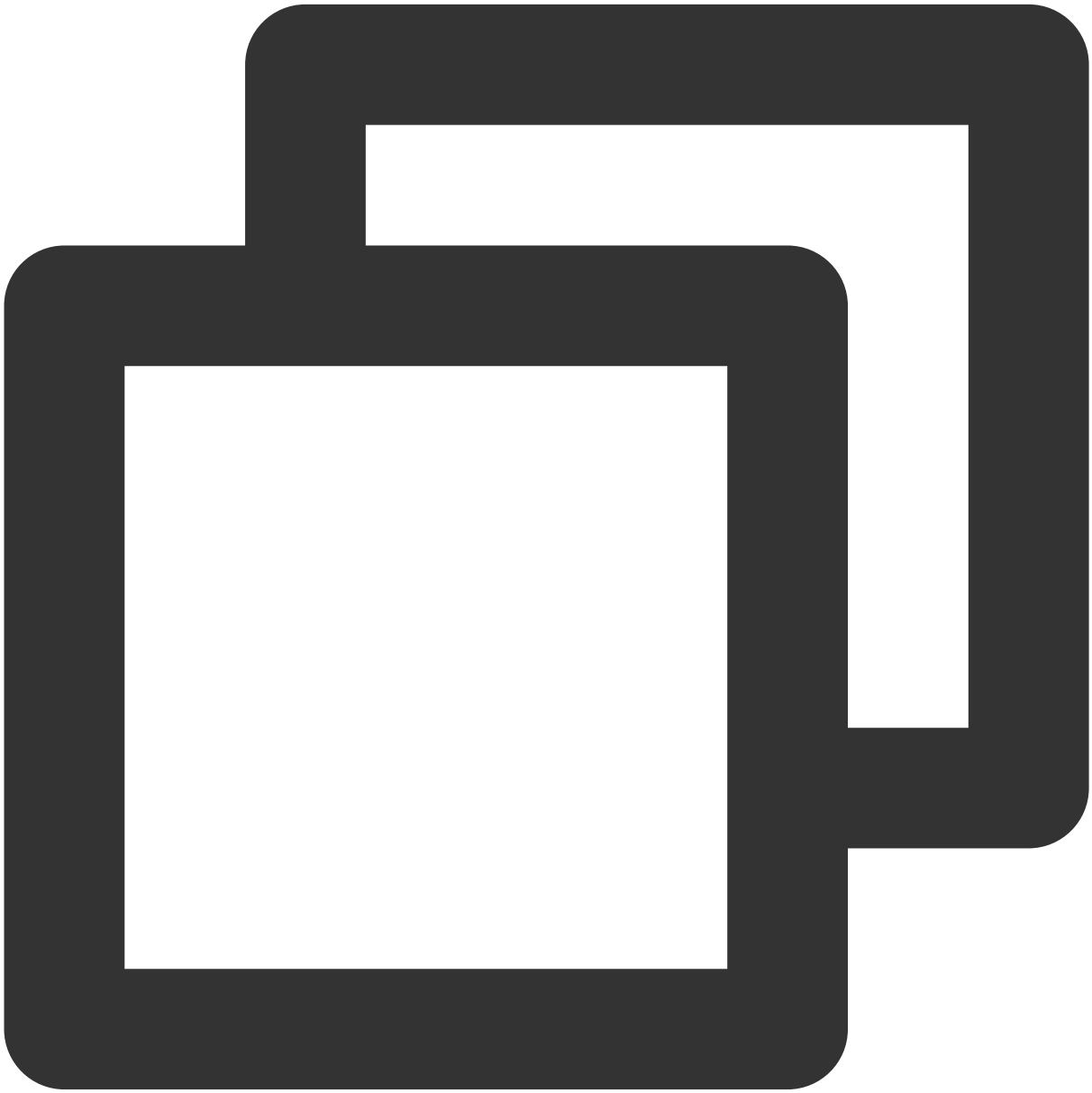


```
void onAllUserCameraDisableChanged(String roomId, boolean isDisable)
```

参数	类型	描述
roomId	String	房间 ID
isDisable	boolean	是否被禁用

onSendMessageForAllUserDisableChanged

房间内所有用户发送文本消息被禁用事件

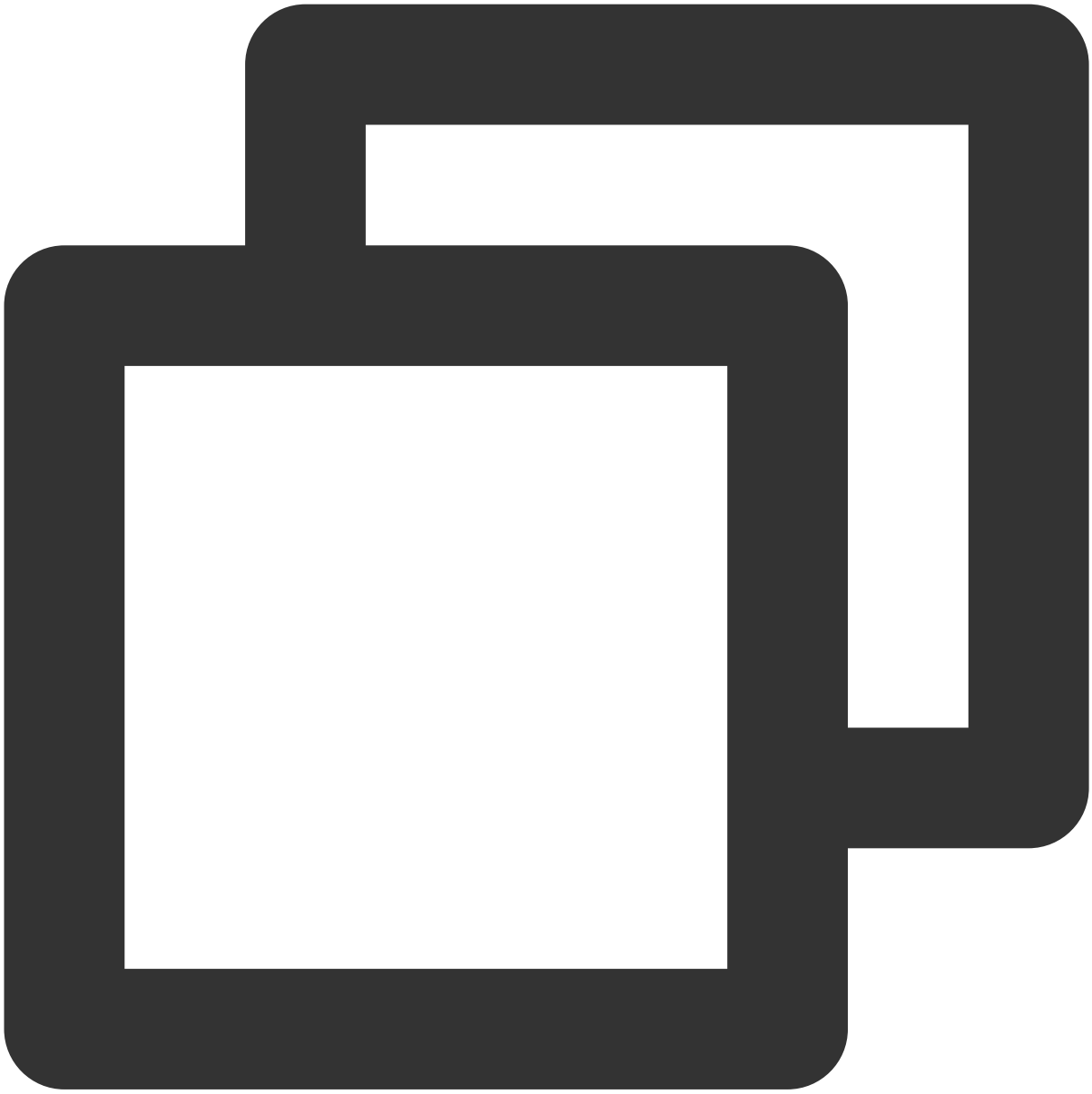


```
void onSendMessageForAllUserDisableChanged(String roomId, boolean isDisable)
```

参数	类型	描述
roomId	String	房间 ID
isDisable	boolean	是否被禁用

onRoomDismissed

房间被解散事件

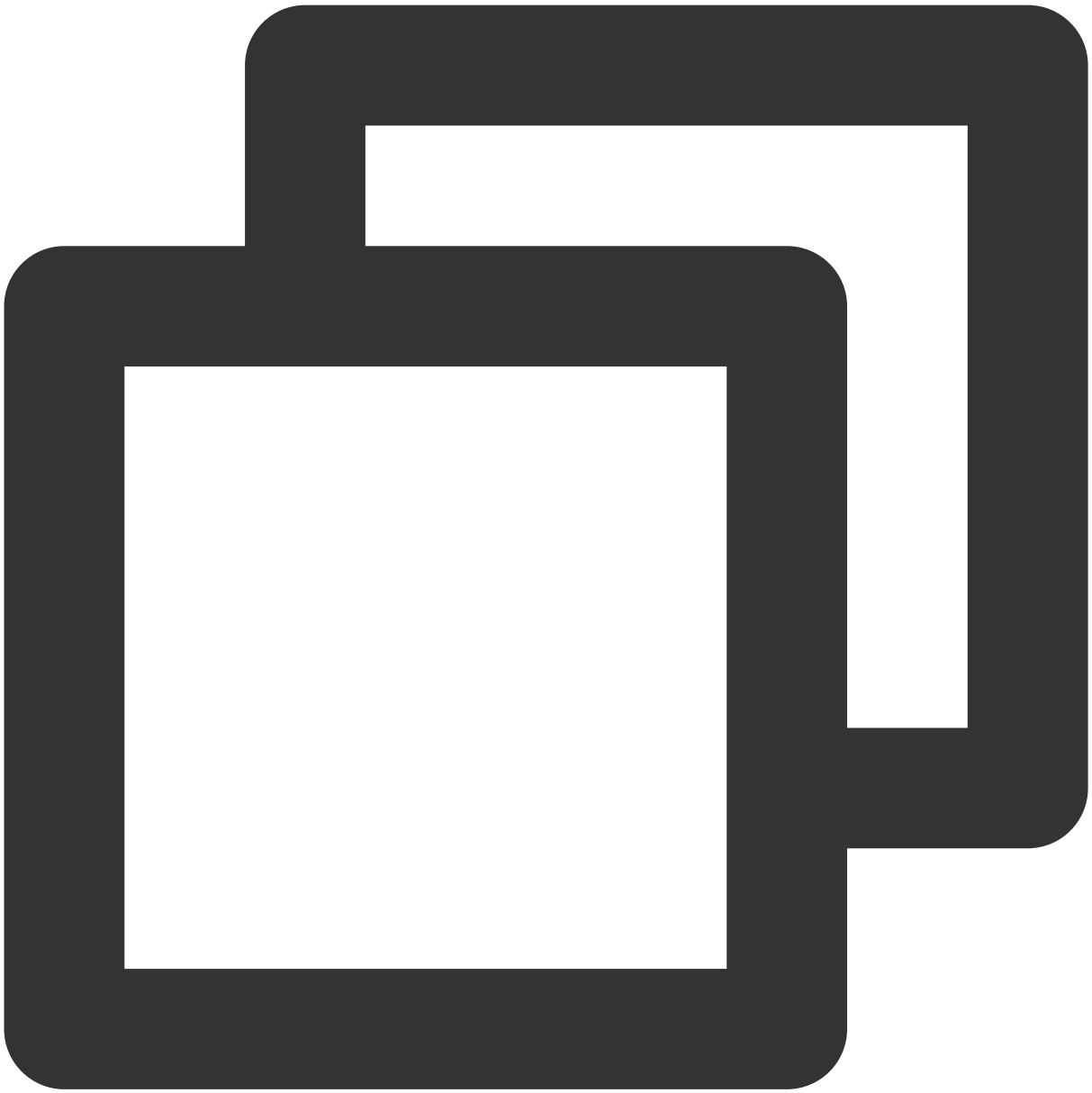


```
void onRoomDismissed(String roomId)
```

参数	类型	描述
roomId	String	房间 ID

onKickedOutOfRoom

踢出房间事件

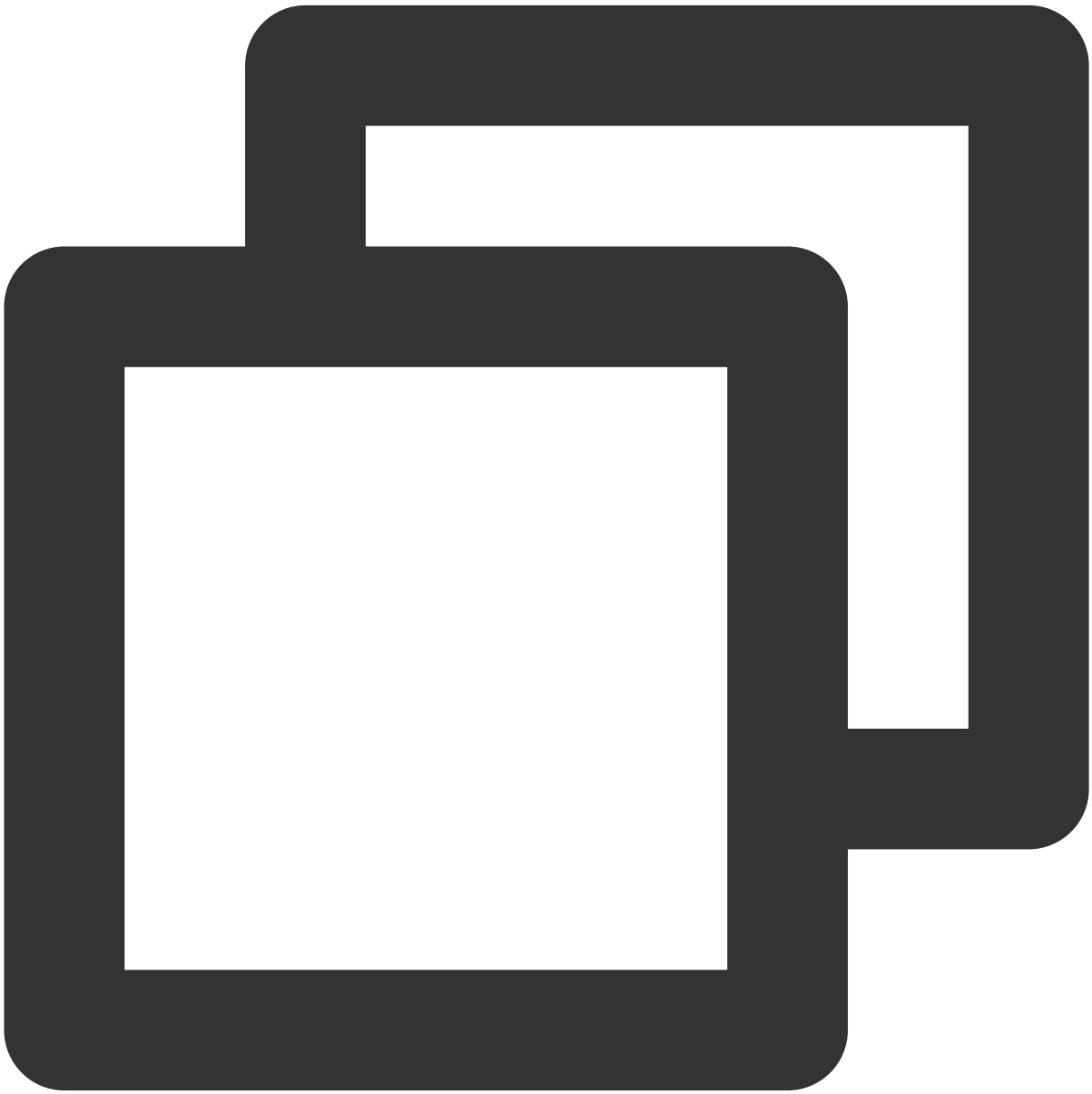


```
void onKickedOutOfRoom(String roomId, String message)
```

参数	类型	描述
roomId	String	房间 ID
message	String	被踢出的描述

onRoomSpeechModeChanged

房间麦控模式发生变化

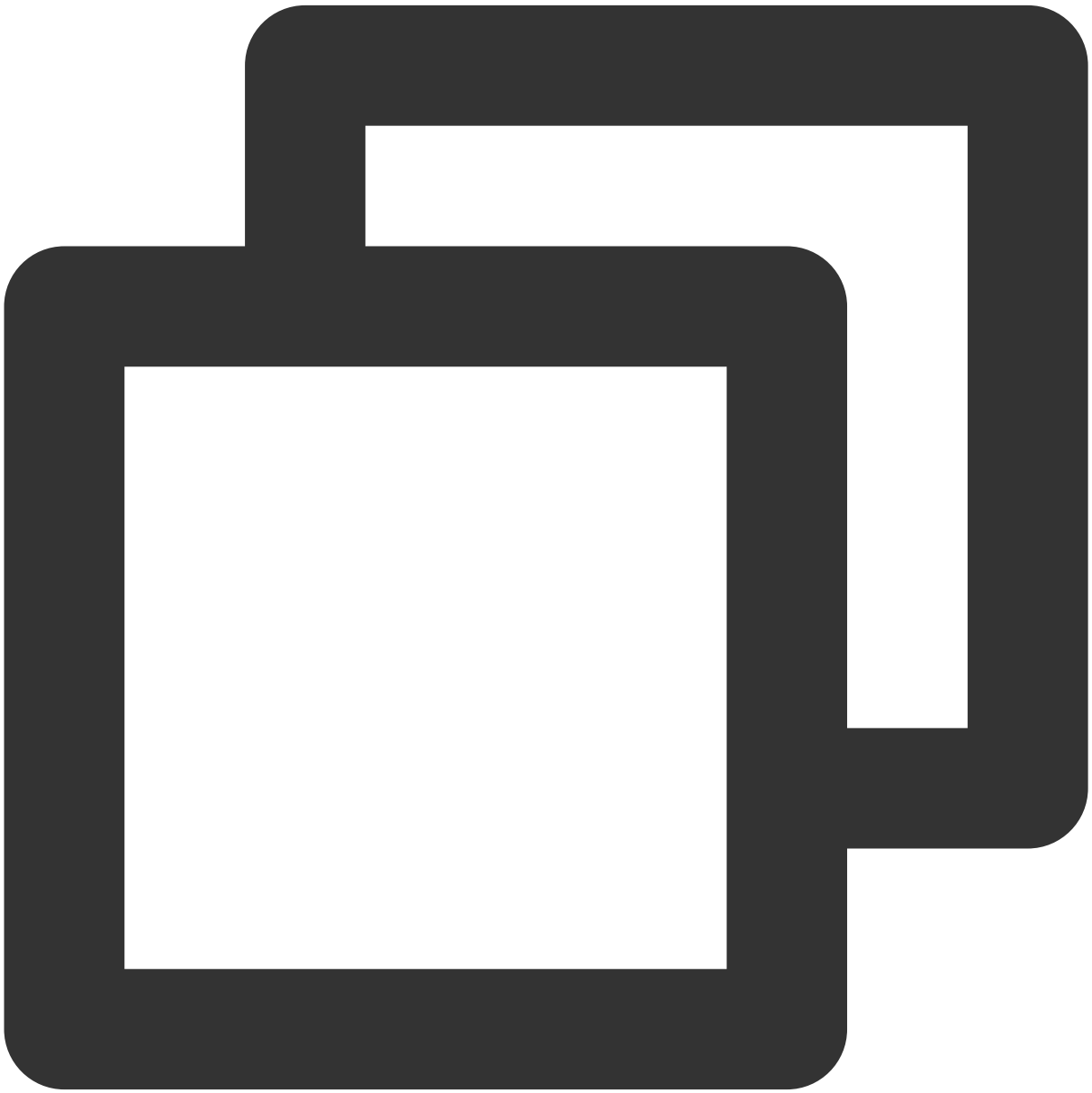


```
void onRoomSpeechModeChanged(String roomId, TUIRoomDefine.SpeechMode speechMode)
```

参数	类型	描述
roomId	String	房间 ID
speechMode	TUIRoomDefine.SpeechMode	房间麦控模式

onRemoteUserEnterRoom

远端用户进入房间事件



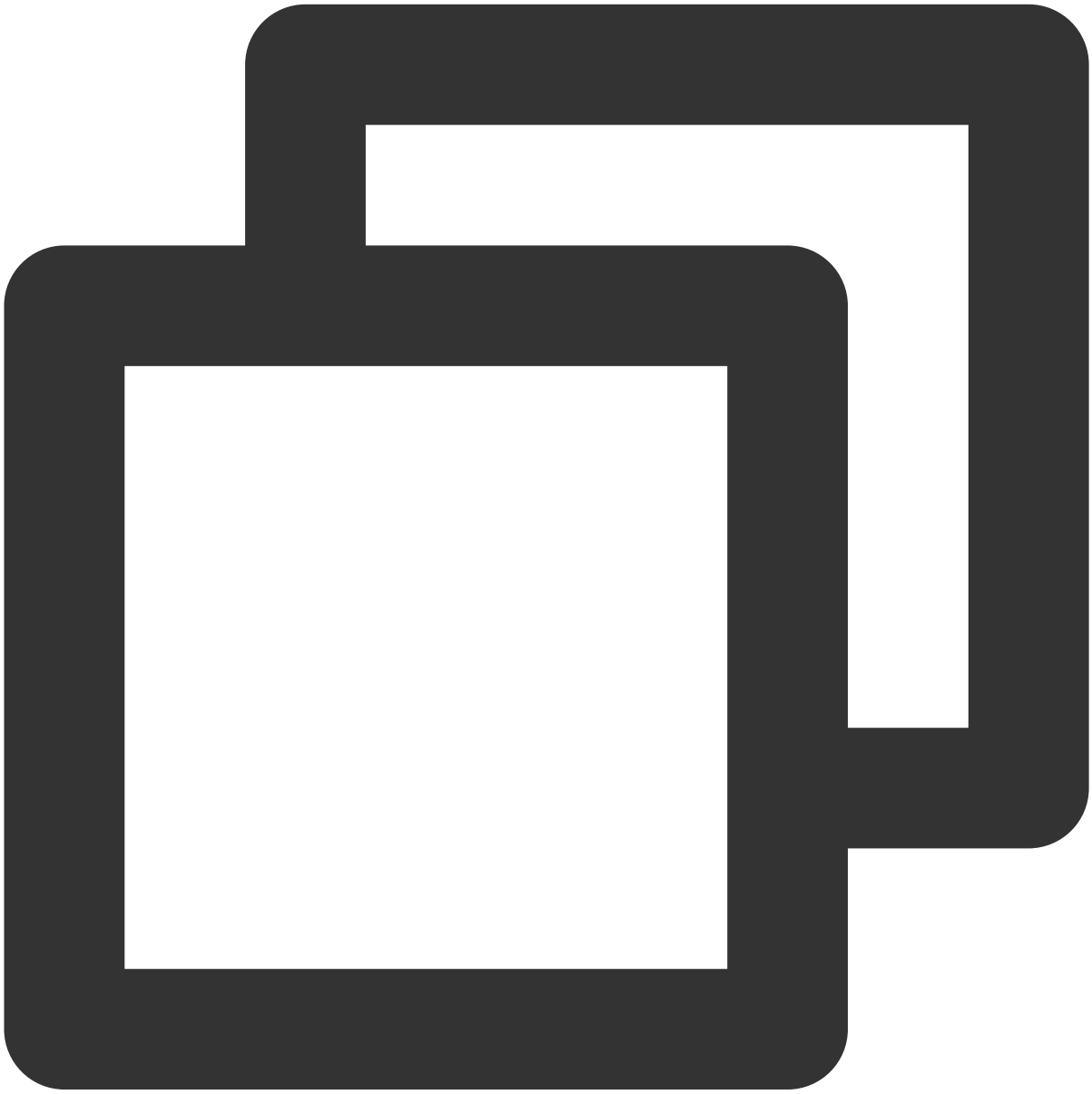
```
void onRemoteUserEnterRoom(String roomId, TUIRoomDefine.UserInfo userInfo)
```

参数	类型	描述
roomId	String	房间 ID

userInfo	TUIRoomDefine.UserInfo	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件



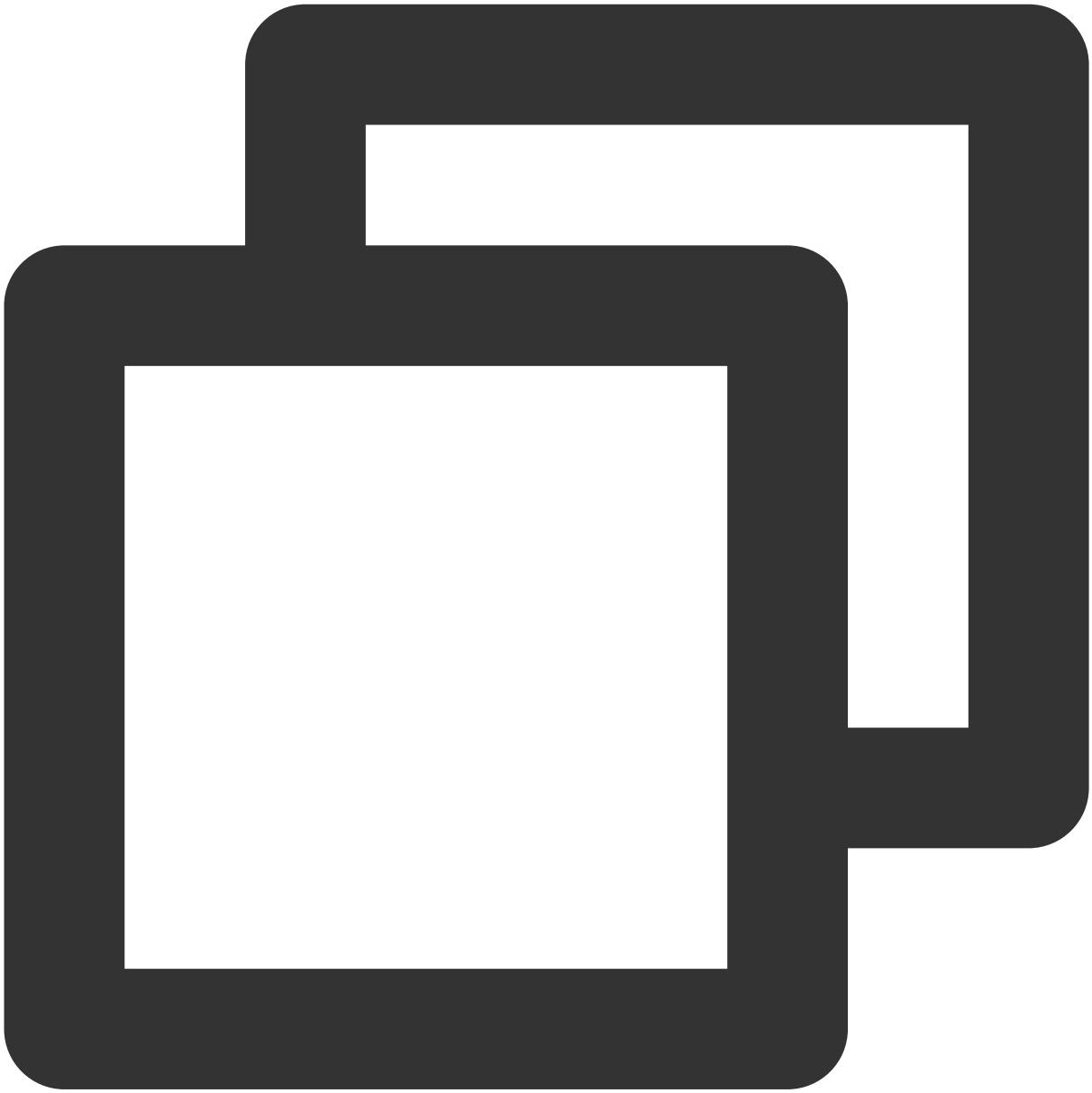
```
void onRemoteUserLeaveRoom(String roomId, TUIRoomDefine.UserInfo userInfo)
```

参数	类型	描述
roomId	String	房间 ID

userInfo	TUIRoomDefine.UserInfo	用户信息

onUserRoleChanged

用户角色改变事件



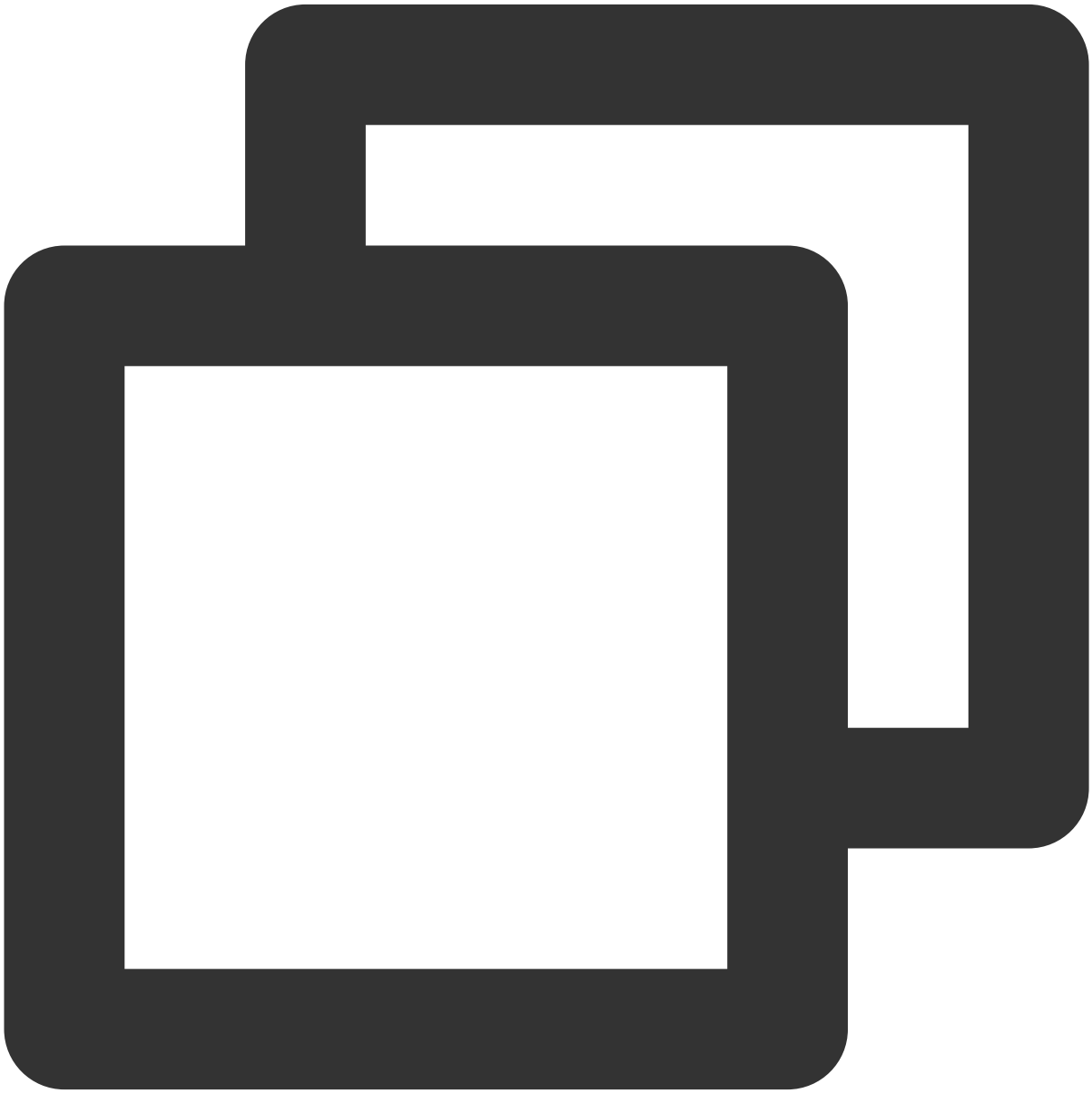
```
void onUserRoleChanged(String userId, TUIRoomDefine.Role role)
```

参数	类型	描述
userId	String	用户 ID

role	TUIRoomDefine.Role	用户角色

onUserVideoStateChanged

用户视频状态改变事件

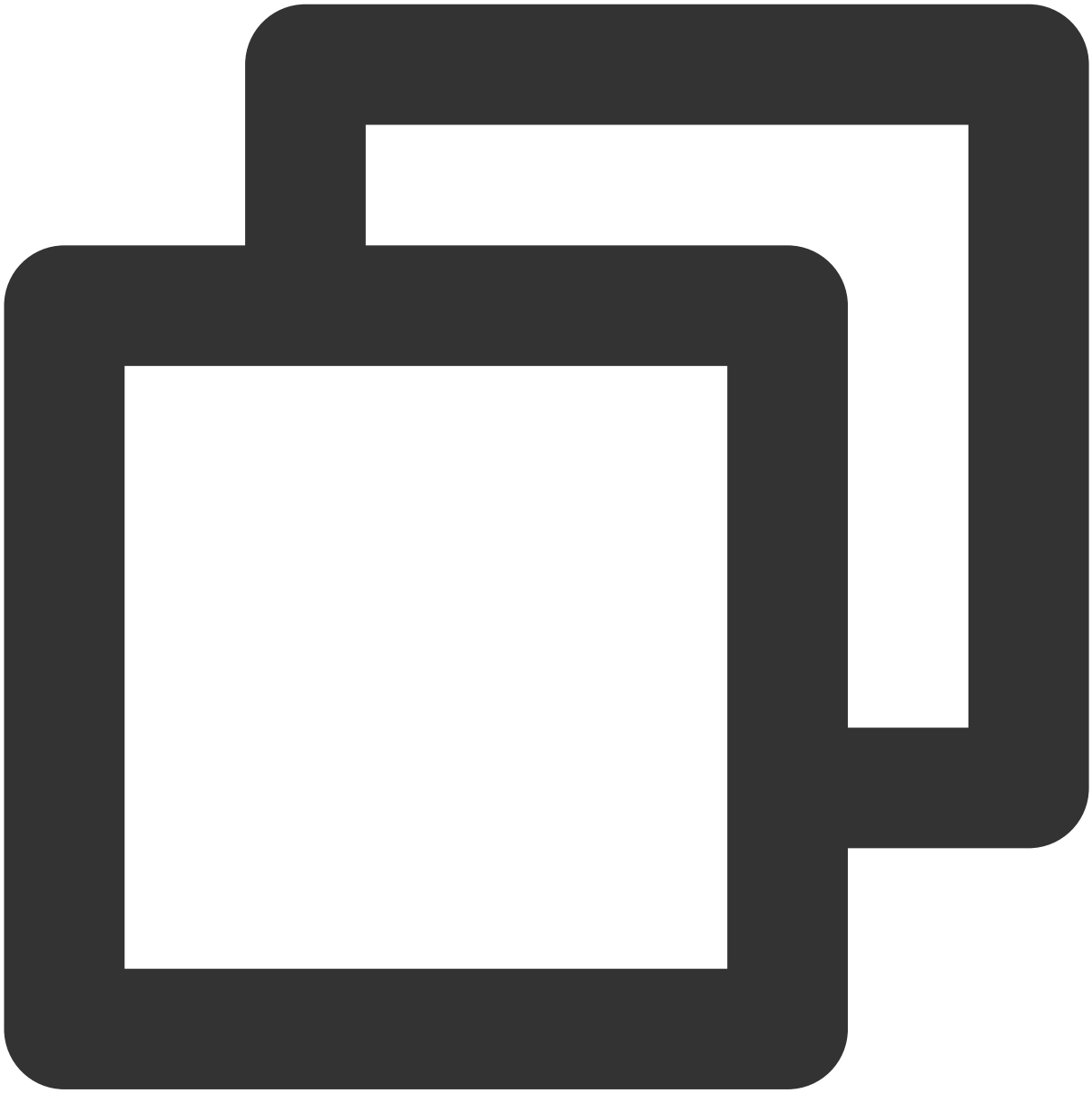


```
void onUserVideoStateChanged(String userId,
                             TUIRoomDefine.VideoStreamType streamType,
                             boolean hasVideo,
                             TUIRoomDefine.ChangeReason reason)
```

参数	类型	描述
userId	String	用户 ID
streamType	TUIRoomDefine.VideoStreamType	视频流类型
hasVideo	boolean	是否有视频流
reason	TUIRoomDefine.ChangeReason	视频流发生变化原因

onUserAudioStateChanged

用户音频状态改变事件

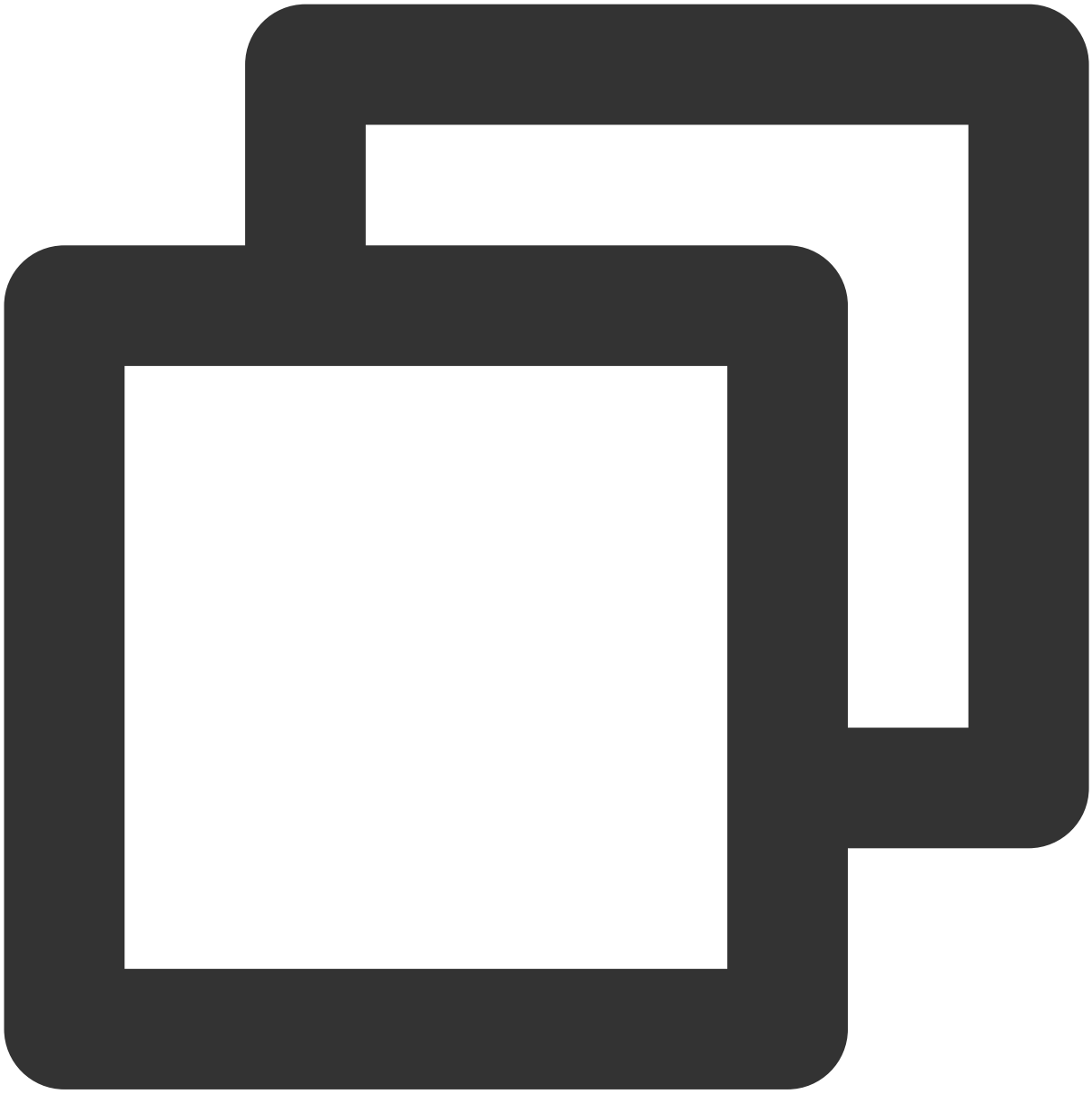


```
void onUserAudioStateChanged(String userId, boolean hasAudio, TUIRoomDefine.ChangeR
```

参数	类型	描述
userId	String	用户 ID
hasAudio	boolean	是否有音频流
reason	TUIRoomDefine.ChangeReason	视频流发生变化原因

onUserVoiceVolumeChanged

用户音量改变事件

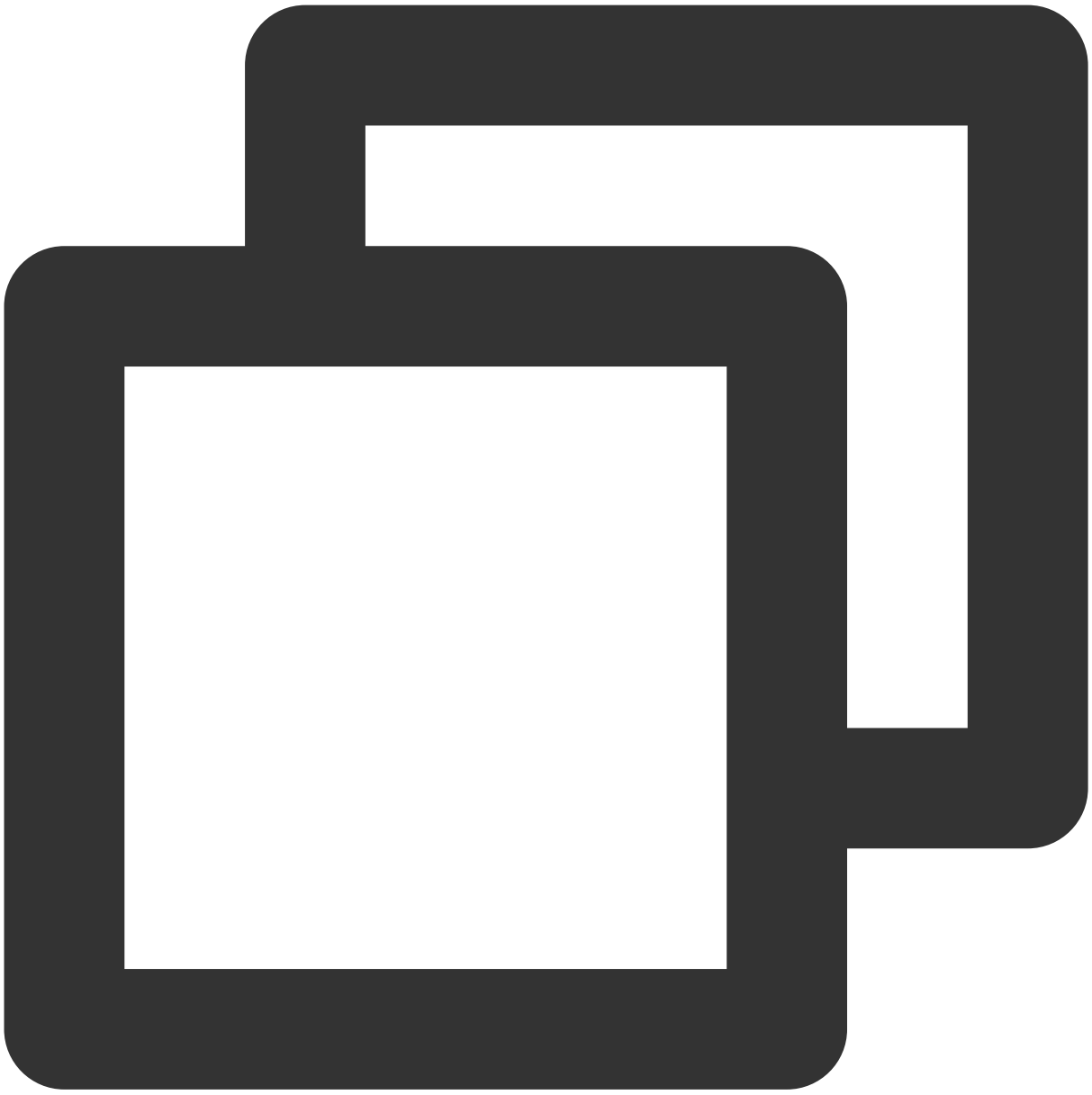


```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap)
```

参数	类型	描述
volumeMap	Map	用户音量 Map key：userId value：用于承载所有正在说话的用户的音量大小，取值范围 0 - 100

onSendMessageForUserDisableChanged

用户文本消息发送能力发生变化事件



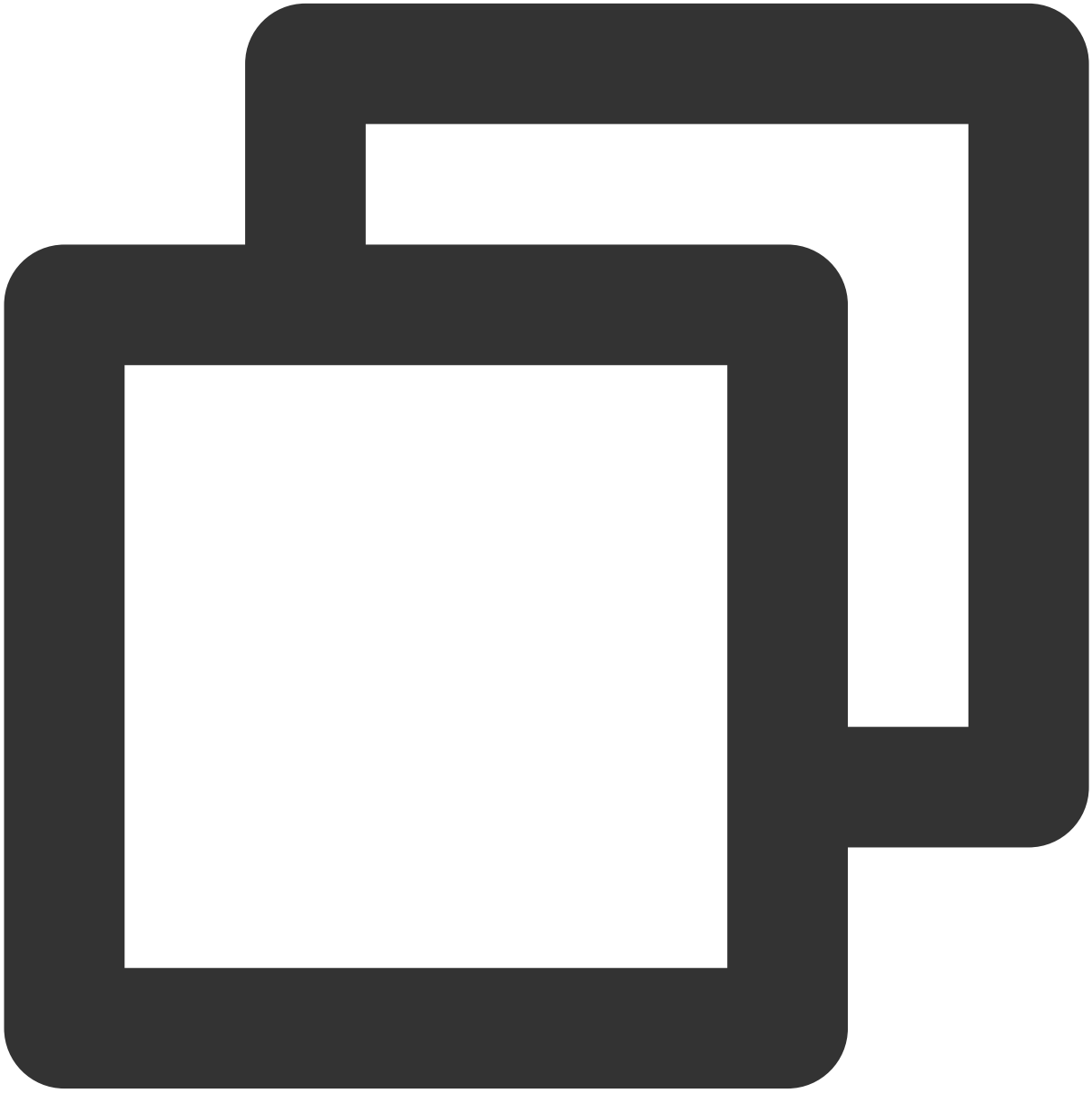
```
void onSendMessageForUserDisableChanged(String roomId, String userId, boolean isDis
```

参数	类型	描述
roomId	String	房间 ID

userId	String	用户 ID
isDisable	boolean	是否被禁止发送文本消息

onUserNetworkQualityChanged

用户网络质量改变事件



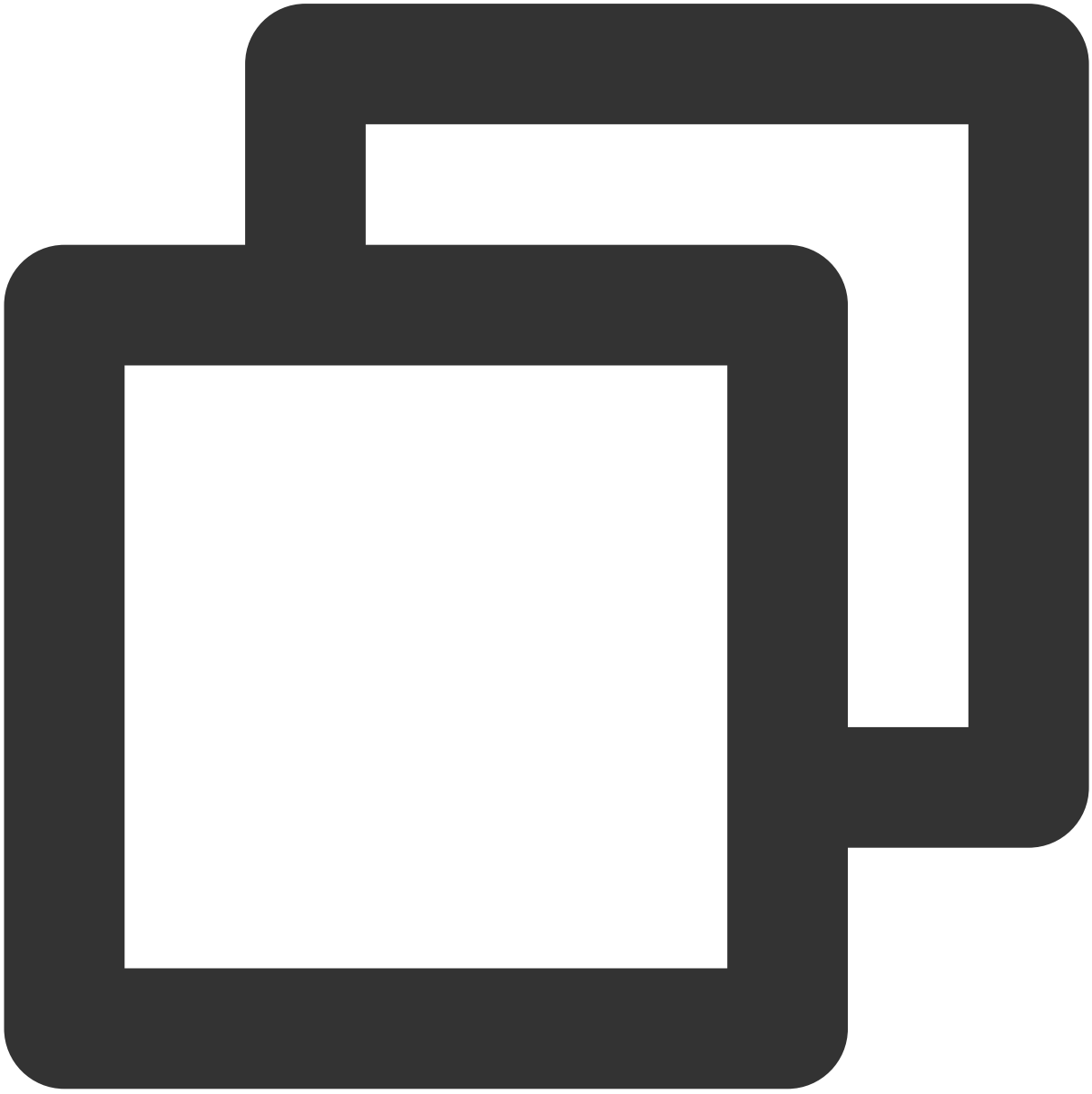
```
void onUserNetworkQualityChanged(Map<String, TUICommonDefine.NetworkInfo> networkMa
```

参数	类型	描述

networkMap	Map	用户网络状态 Map key：userId value：网络状况

onUserScreenCaptureStopped

屏幕分享结束事件

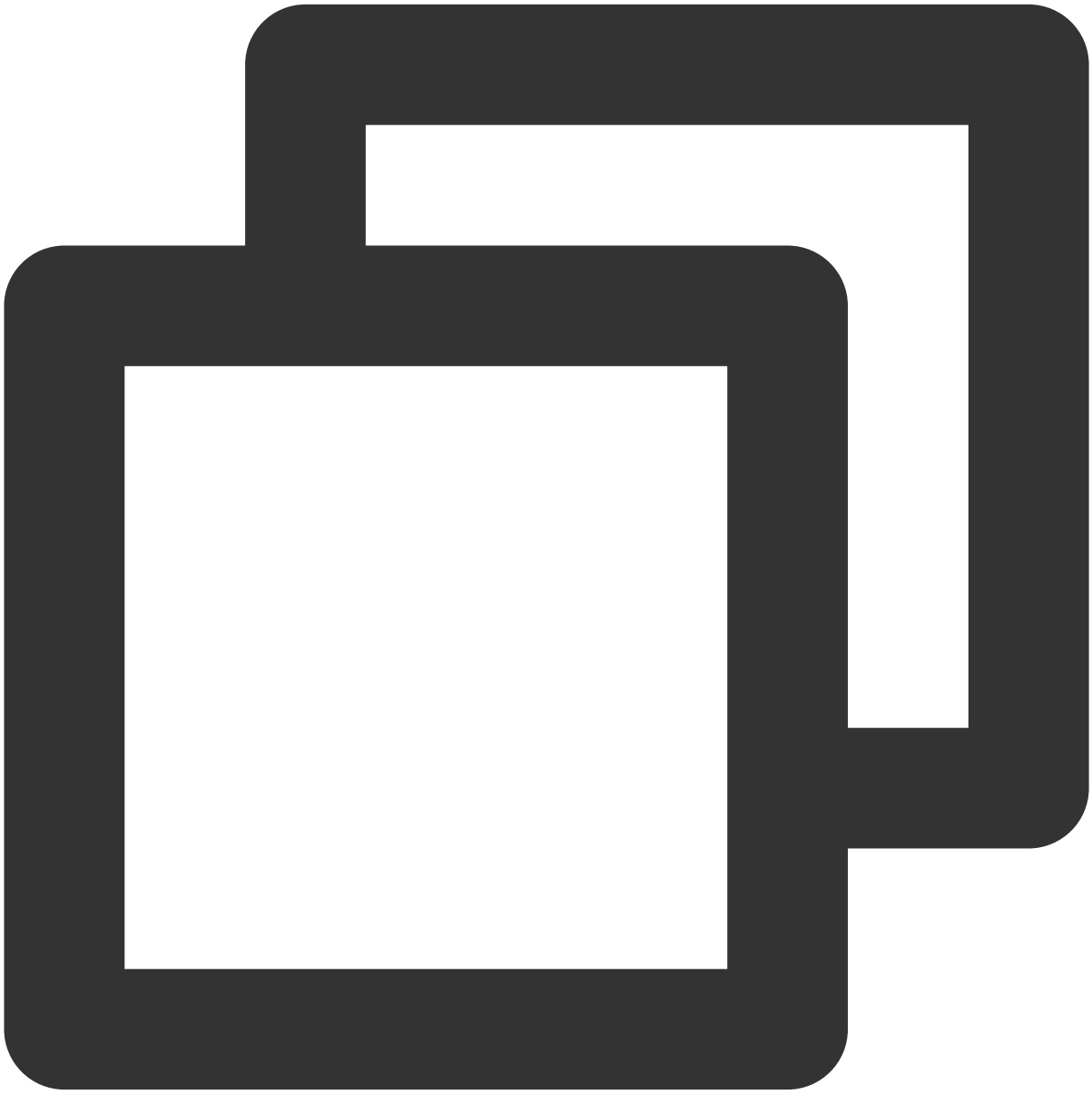


```
void onUserScreenCaptureStopped(int reason)
```


参数	类型	描述
reason	int	停止原因 0：用户主动停止 1：屏幕窗口关闭导致停止、 2：表示屏幕分享的显示屏状态变更（如接口被拔出、投影模式变更等）

onRoomMaxSeatCountChanged

房间内最大麦位数发生变化事件（仅在会议类型房间生效）

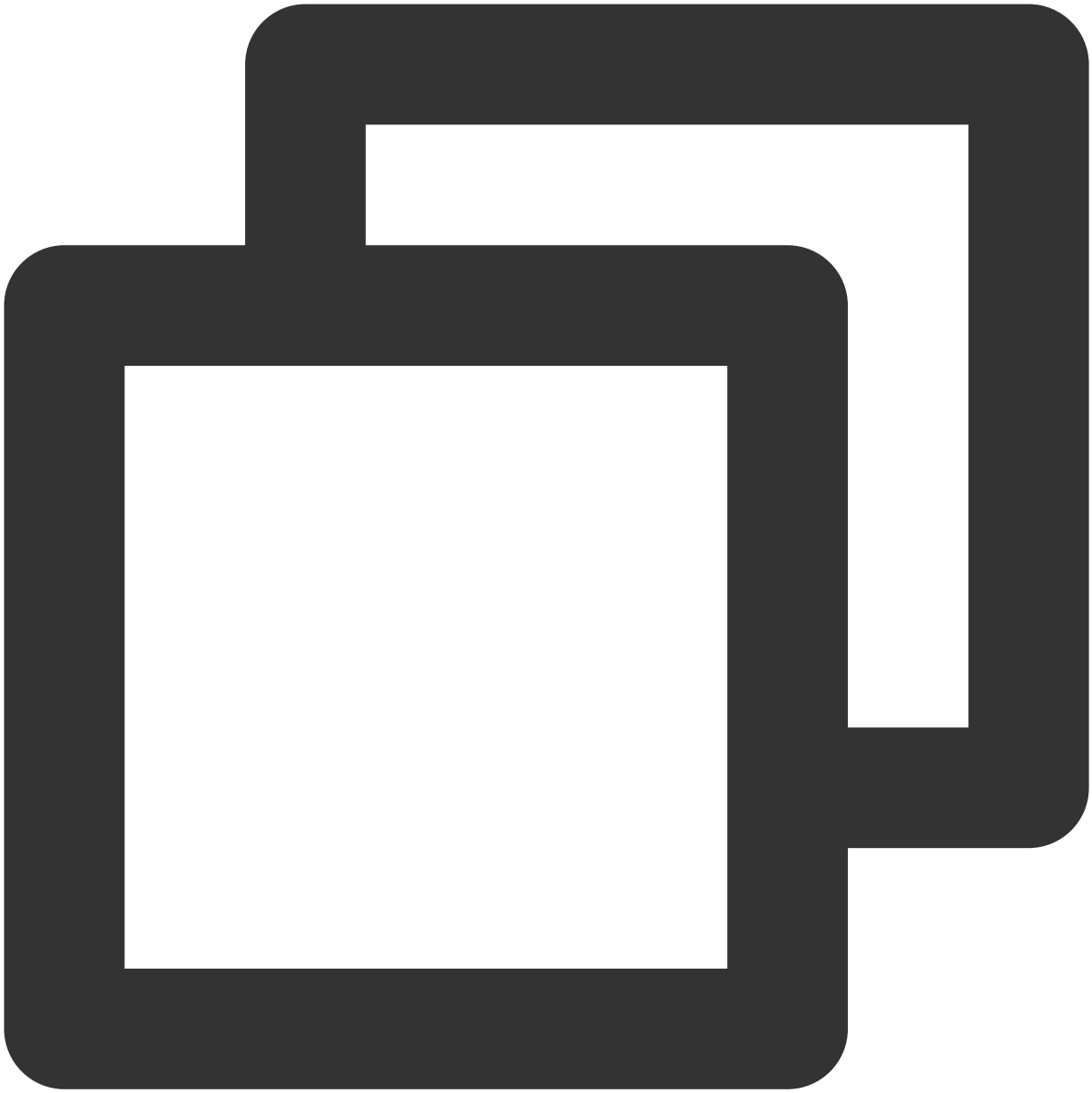


```
void onRoomMaxSeatCountChanged(String roomId, int maxSeatCount)
```

参数	类型	描述
roomId	String	房间 ID
maxSeatCount	int	房间内最大麦位数量

onSeatListChanged

麦位列表改变事件



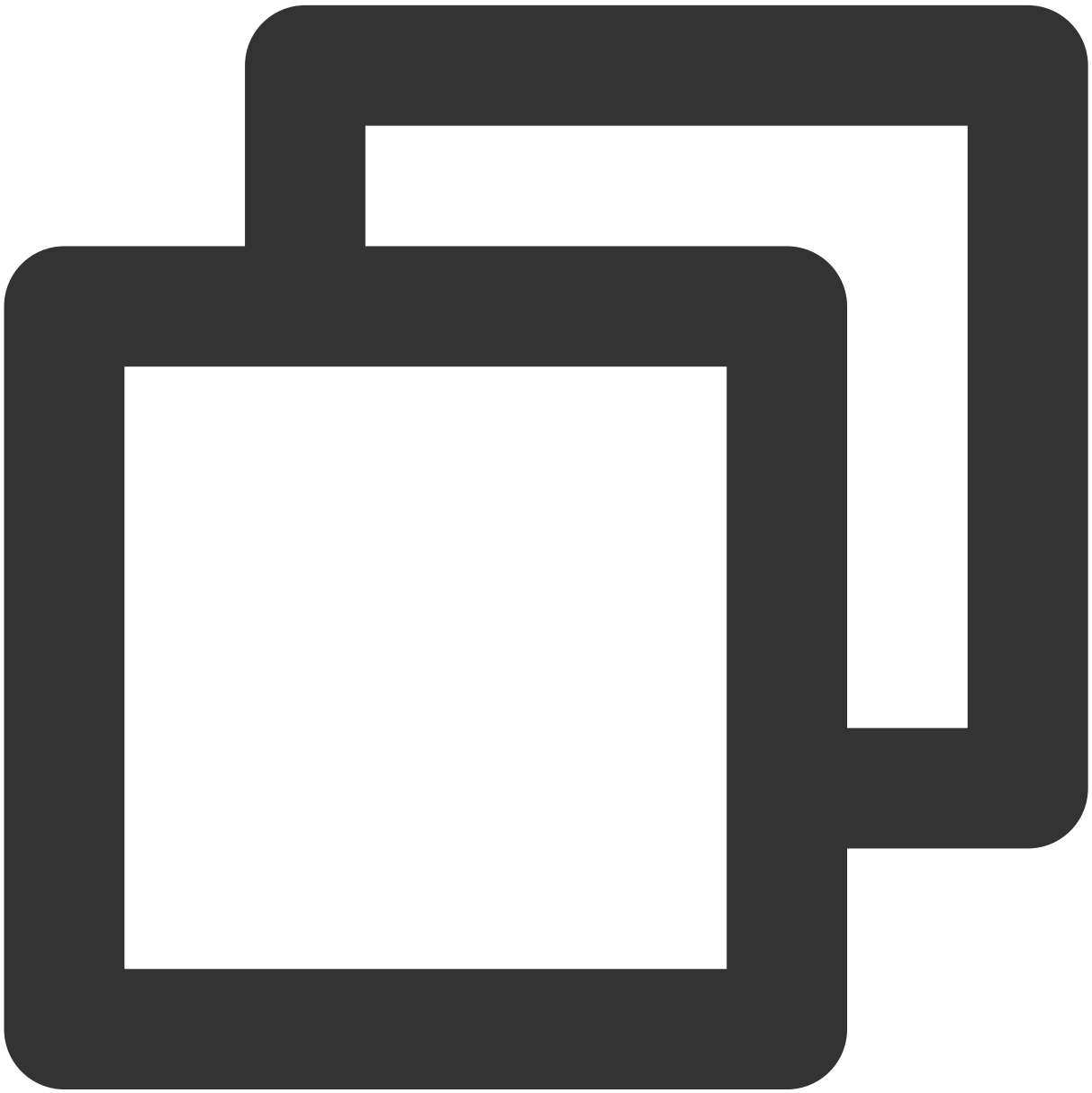
```
void onSeatListChanged(List<TUIRoomDefine.SeatInfo> seatList,
                       List<TUIRoomDefine.SeatInfo> seatedList,
                       List<TUIRoomDefine.SeatInfo> leftList)
```

参数	类型	描述
seatList	List<TUIRoomDefine.SeatInfo>	目前麦上最新的用户列表，包含新上麦的用户
seatedList	List<TUIRoomDefine.SeatInfo>	新上麦的用户列表
leftList	List<TUIRoomDefine.SeatInfo>	新下麦的用户列表

版权所有：腾讯云计算（北京）有限责任公司

onKickedOffSeat

收到用户被踢下麦事件



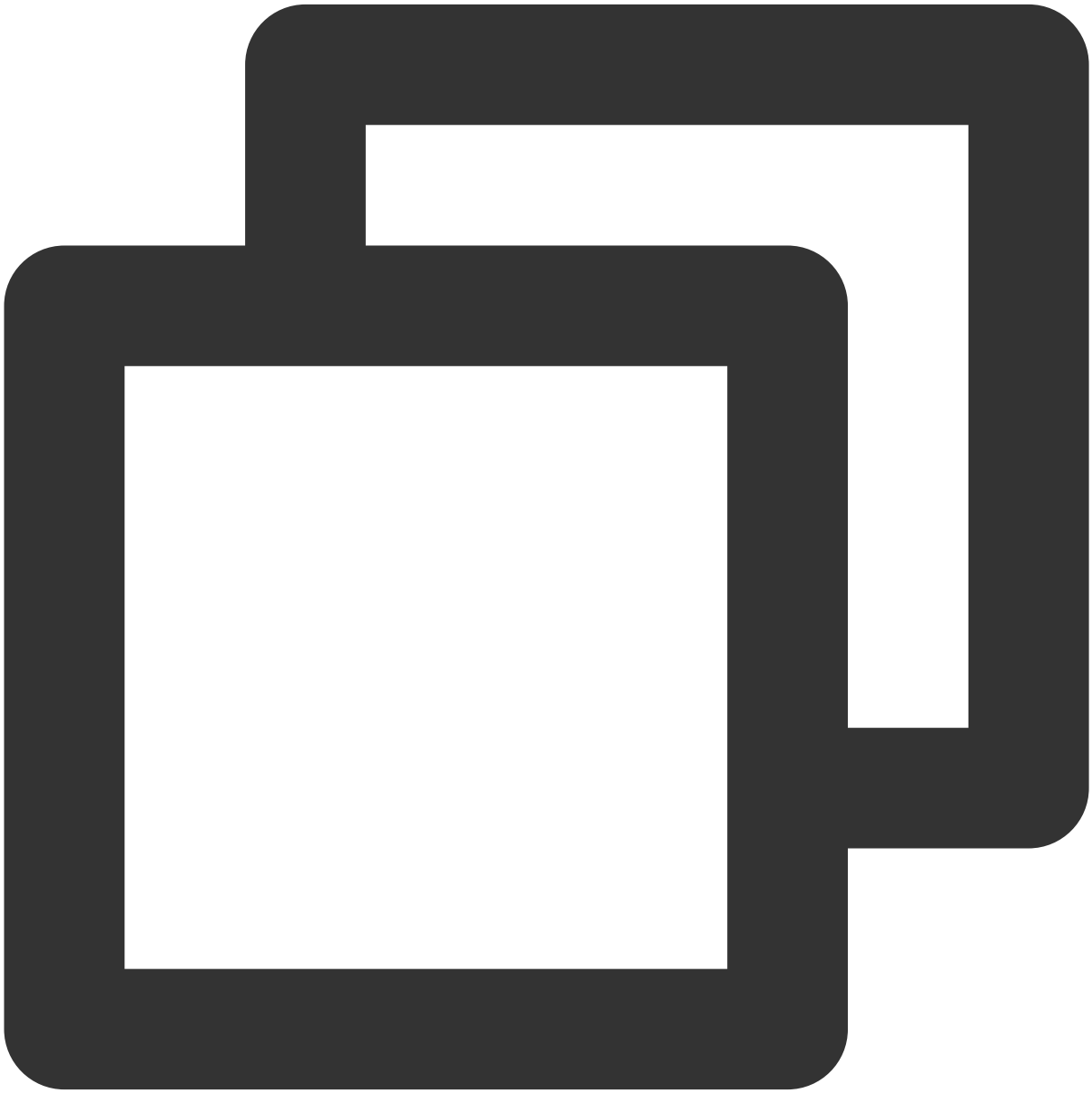
```
void onKickedOffSeat(String userId)
```

参数	类型	描述

userId	String	操作踢人的（主持人/管理员）用户 ID

onRequestReceived

请求接收事件

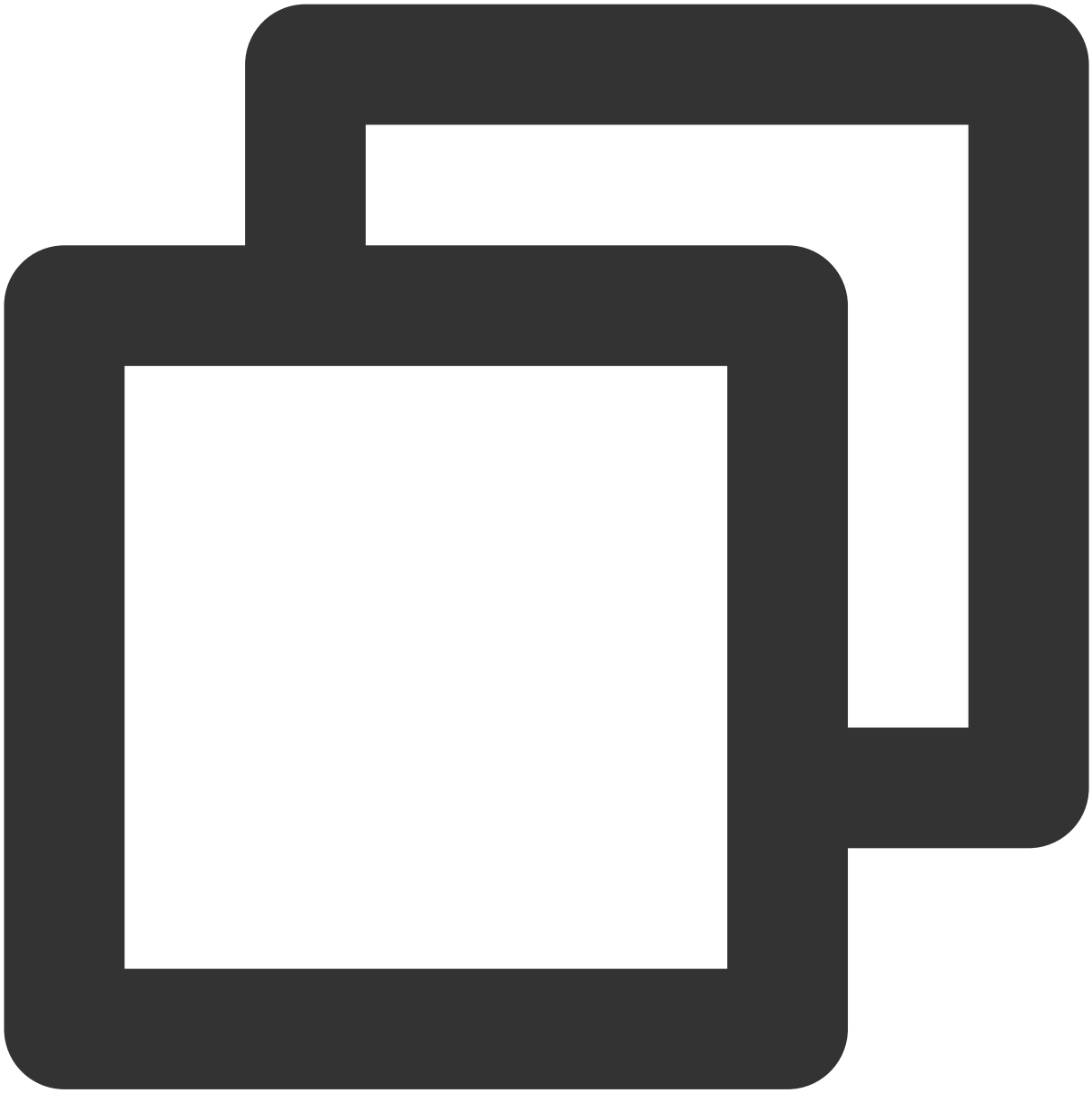


```
void onRequestReceived(TUIRoomDefine.Request request)
```

参数	类型	描述
request	TUIRoomDefine.Request	请求内容

onRequestCancelled

请求取消事件

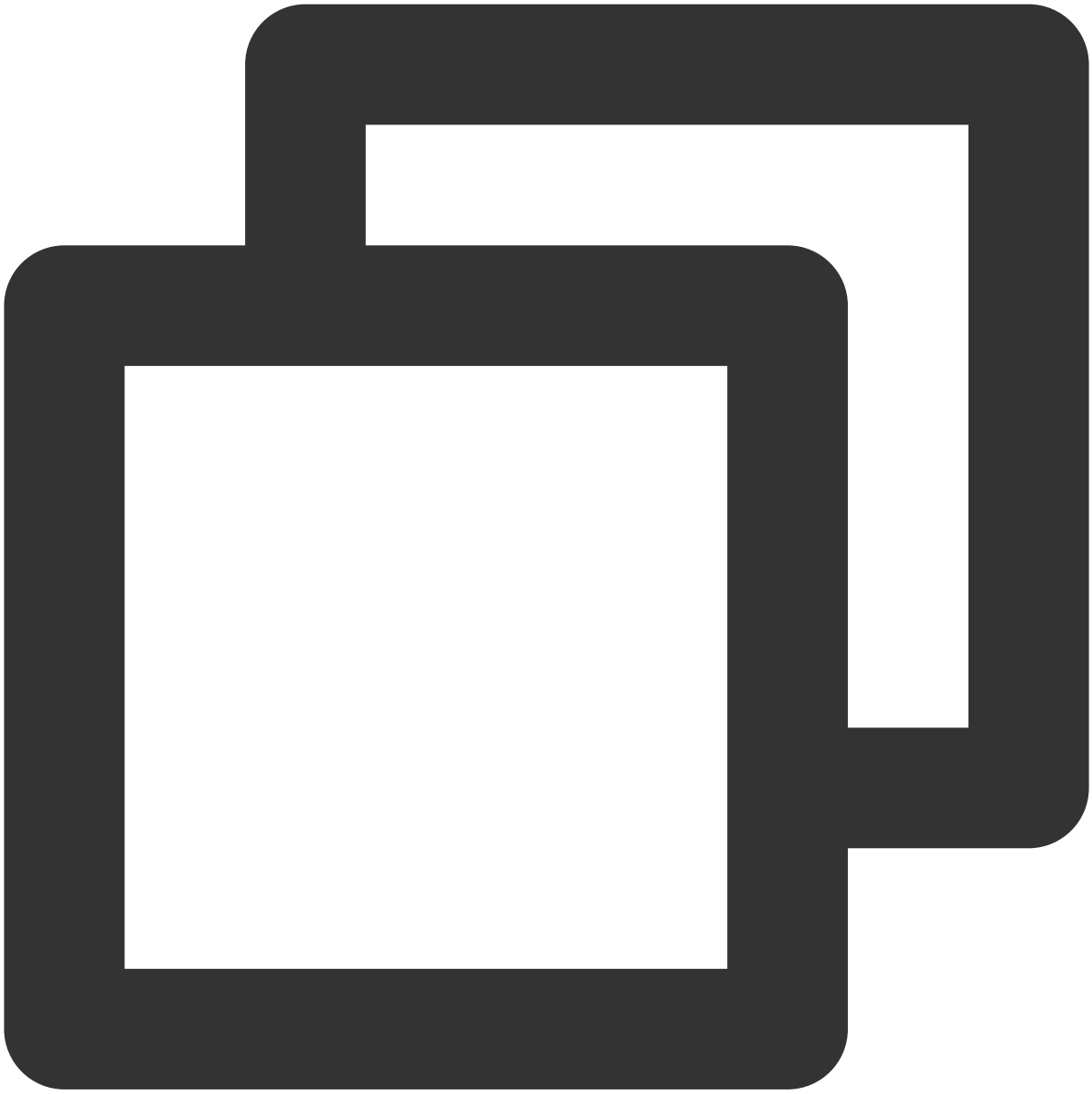


```
void onRequestCancelled(String requestId, String userId)
```

参数	类型	描述
requestId	String	请求 ID
userId	String	取消信令的用户 ID

onReceiveTextMessage

接收文本消息事件

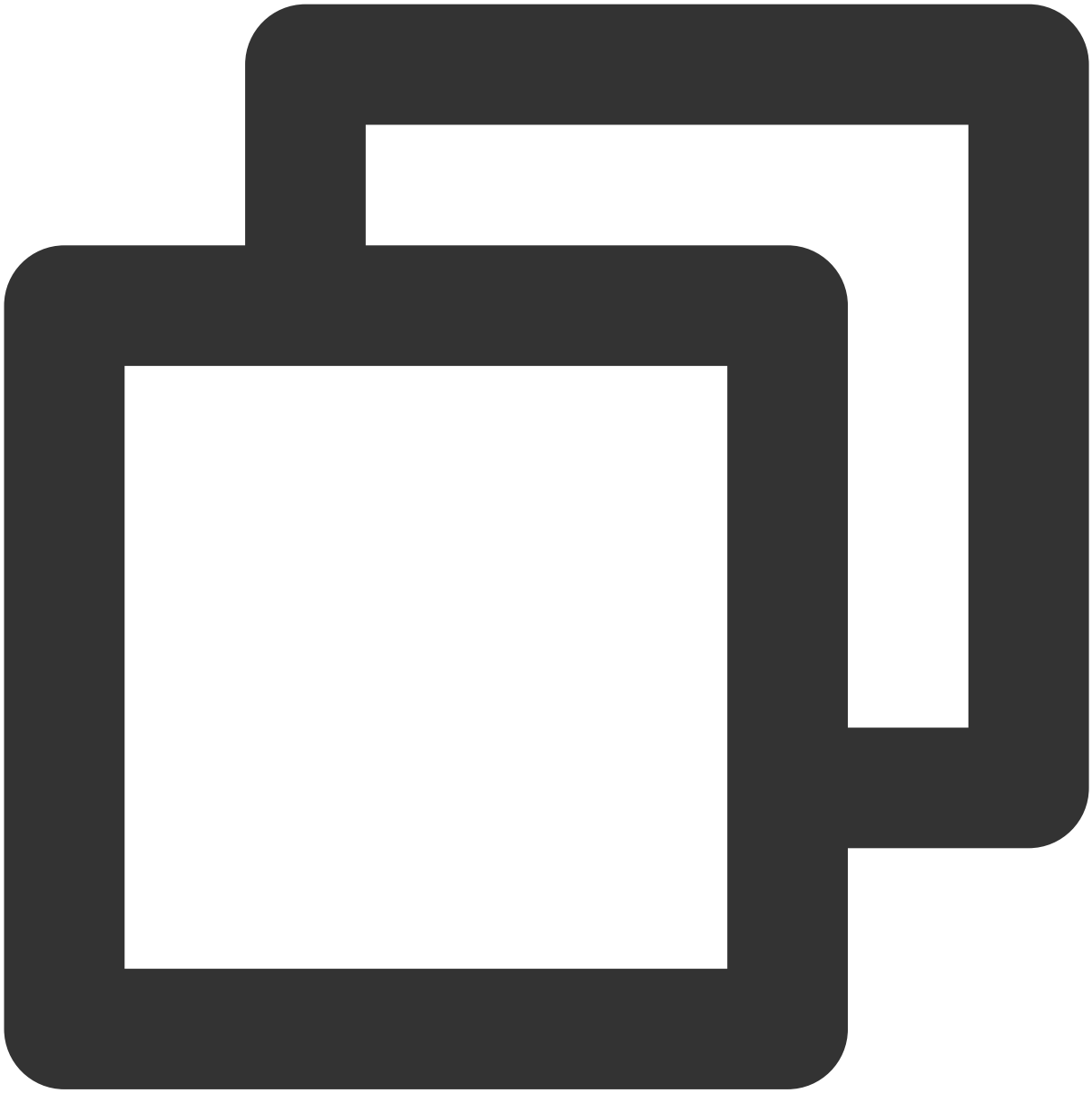


```
void onReceiveTextMessage(String roomId, TUICommonDefine.Message message)
```

参数	类型	描述
roomId	String	房间 ID
message	TUICommonDefine.Message	消息内容

onReceiveCustomMessage

接收自定义消息事件



```
void onReceiveCustomMessage(String roomId, TUICommonDefine.Message message)
```

参数	类型	描述
roomId	String	房间 ID
message	TUICommonDefine.Message	消息内容

版权所有：腾讯云计算（北京）有限责任公司

类型定义

最近更新时间：2023-10-23 11:30:14

枚举定义

TUIRoomDefine

类型	说明
RoomType	房间类型
SpeechMode	房间模式
MediaDevice	房间内媒体设备类型
Role	房间内角色类型
VideoQuality	视频质量
AudioQuality	音频质量
VideoStreamType	视频流类型
ChangeReason	更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）
RequestAction	请求类型

TUICommonDefine

类型	说明
NetworkQuality	网络质量

常用结构

TUIRoomDefine

类型	说明
RoomInfo	房间信息
LoginUserInfo	登录用户信息
UserInfo	房间内用户信息

版权所有：腾讯云计算（北京）有限责任公司

SeatInfo	房间内座位信息
SeatLockParams	锁定麦位操作参数
UserVoiceVolume	房间内用户音量
Request	信令请求

TUICommonDefine

类型	说明
NetworkInfo	网络质量信息
Message	消息

RoomType

房间类型

枚举	取值	描述
CONFERENCE	1	会议类型房间，适用于会议，教育场景，该房间中可以开启自由发言，申请发言、麦控等不同模式，麦位是无序的
LIVE_ROOM	2	直播类型房间，适用于直播场景，该房间可以开启自由发言，麦位控制模式，该房间中麦位是有编号的

SpeechMode

房间内麦控模式

枚举	取值	描述
FREE_TO_SPEAK	1	自由发言模式：普通用户可以自由发言
APPLY_TO_SPEAK	2	申请发言模式（仅在会议类型房间下生效）：普通用户每次发言均需要先申请，待同意后才可以发言
SPEAK_AFTER_TAKING_SEAT	3	上麦发言模式：普通用户需要申请上麦，待同意上麦之后才能发言

MediaDevice

房间内媒体设备类型

枚举	取值	描述
MICROPHONE	1	麦克风
CAMERA	2	摄像头
SCREEN_SHARING	3	屏幕共享

Role

房间内角色类型

枚举	取值	描述
ROOM_OWNER	0	房主，一般指房间的创建者，房间内最高权限拥有者
MANAGER	1	房间管理员
GENERAL_USER	2	房间内普通成员

VideoQuality

视频质量

枚举	取值	描述
Q_360P	1	低清360P
Q_540P	2	标清540P
Q_720P	3	高清720P
Q_1080P	4	超清1080P

AudioQuality

音频质量

枚举	取值	描述
SPEECH	0	人声模式
DEFAULT	1	默认模式
MUSIC	2	音乐模式

版权所有：腾讯云计算（北京）有限责任公司

VideoStreamType

视频流类型

枚举	取值	描述
CAMERA_STREAM	0	高清摄像头视频流
SCREEN_STREAM	1	屏幕分享视频流
CAMERA_STREAM_LOW	2	低清摄像头视频流

ChangeReason

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

枚举	取值	描述
BY_SELF	0	自己操作
BY_ADMIN	1	房主或管理员操作

RequestAction

请求类型

枚举	取值	描述
INVALID_ACTION	0	无效请求
REQUEST_TO_OPEN_REMOTE_CAMERA	1	请求远端用户打开摄像头
REQUEST_TO_OPEN_REMOTE_MICROPHONE	2	请求远端用户打开麦克风
REQUEST_TO_CONNECT_OTHER_ROOM	3	请求连接到其他房间
REQUEST_TO_TAKE_SEAT	4	请求上麦
REQUEST_REMOTE_USER_ON_SEAT	5	请求远端用户上麦
REQUEST_APPLY_TO_ADMIN_TO_OPEN_LOCAL_CAMERA	6	向管理员请求打开本地摄像头
REQUEST_APPLY_TO_ADMIN_TO_OPEN_LOCAL_MICROPHONE	7	向管理员请求打开本地麦克风

版权所有：腾讯云计算（北京）有限责任公司

NetworkQuality

网络质量

枚举	取值	描述
UNKNOWN	0	未定义
EXCELLENT	1	当前网络非常好
GOOD	2	当前网络比较好
POOR	3	当前网络一般
BAD	4	当前网络较差
VERY_BAD	5	当前网络很差
DOWN	6	当前网络不满足 TRTC 的最低要求

RoomInfo

房间信息

字段	类型	描述
roomId	String	房间 ID(创建房间必填参数)
ownerId	String	主持人 ID: 默认为房间创建者（只读）
roomType	RoomType	房间类型（创建房间可选参数，默认 CONFERENCE 类型）
name	String	房间名称（创建房间可选参数，默认房间 ID）
speechMode	SpeechMode	房间麦控模式,默认 FREE_TO_SPEAK
isCameraDisableForAllUser	boolean	是否禁止打开摄像头（创建房间可选参数），默认值 false
isMicrophoneDisableForAllUser	boolean	是否禁止打开麦克风（创建房间可选参数），默认值 false
isMessageDisableForAllUser	boolean	是否禁止发送消息（创建房间可选参数），默认值 false
maxSeatCount	int	最大麦位数
enableCDNStreaming	boolean	是否开启CDN直播（创建房间可选参数，直播房间使用），默认值 false

版权所有：腾讯云计算（北京）有限责任公司

cdnStreamDomain	String	直播推流域名（创建房间可选参数，直播房间使用），默认值空。
createTime	long	房间创建时间（只读）
memberCount	int	房间内成员数量（只读）

LoginUserInfo

登录用户信息

字段	类型	描述
userId	String	用户 ID
userName	String	用户名称
avatarUrl	String	用户头像 URL

UserInfo

房间内用户信息

字段	类型	描述
userId	String	用户 ID
userName	String	用户名称
avatarUrl	String	用户头像 URL
userRole	Role	用户角色类型
hasAudioStream	boolean	是否有音频流
hasVideoStream	boolean	是否有视频流
hasScreenStream	boolean	是否有屏幕分享流

SeatInfo

房间内座位信息

字段	类型	描述
index	int	麦位序号

版权所有：腾讯云计算（北京）有限责任公司

userId	String	用户 ID
isLocked	boolean	麦位是否被锁定，默认 false
isVideoLocked	boolean	麦位是否被禁止打开摄像头，默认 false
isAudioLocked	boolean	麦位是否被禁止打开麦克风，默认 false

SeatLockParams

锁定麦位操作参数

字段	类型	描述
lockSeat	boolean	锁定麦位，默认值 false
lockVideo	boolean	锁定麦位摄像头，默认值 false
lockAudio	boolean	锁定麦位麦克风，默认值 false

UserVoiceVolume

房间内用户音量

字段	类型	描述
userId	String	用户 ID
volume	int	音量 用于承载所有正在说话的用户的音量大小，取值范围 0 - 100

Request

信令请求

字段	类型	描述
requestId	String	请求 ID
requestAction	RequestAction	请求类型
userId	String	用户 ID
content	String	信令内容

timestamp	int	时间戳

NetworkInfo

网络质量信息

字段	类型	描述
userId	String	用户 ID
quality	NetworkQuality	网络质量
upLoss	int	上行丢包率
downLoss	int	下行丢包率
delay	int	网络延迟

Message

消息

字段	类型	描述
messageId	String	消息 ID
message	String	消息文本
timestamp	long	消息时间
userId	String	消息发送者
userName	String	消息发送者昵称
avatarUrl	String	消息发送者头像

版权所有：腾讯云计算（北京）有限责任公司

Windows API 概览

最近更新时间：2023-10-30 11:01:22

TUIRoomEngine API 列表

TUIRoomEngine API 是音视频通话组件的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。

TUIRoomEngine

TUIRoomEngine 核心方法

API	描述
createTUIRoomEngine	创建 TUIRoomEngine 实例
destroyTUIRoomEngine	销毁 TUIRoomEngine 实例
login	登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。
logout	退出登录接口，会有主动离开房间操作、销毁资源。
setSelfInfo	设置本地用户名称和头像。
getSelfInfo	获取本地用户登录的基本信息。
addObserver	设置事件回调。
removeObserver	移除事件回调。

房间相关主动接口

API	描述
createRoom	创建房间。
destroyRoom	解散房间。
enterRoom	进入房间。
exitRoom	离开房间。
connectOtherRoom	连接其他房间。

版权所有：腾讯云计算（北京）有限责任公司

disConnectOtherRoom	断开与其他房间的连接。
fetchRoomInfo	获取房间信息。
updateRoomNameByAdmin	更新房间名称（只有管理员或群主能够调用）。
updateRoomSpeechModeByAdmin	设置房间麦控模式（只有管理员或群主能够调用）。

本地用户视图渲染、视频管理

API	描述
setLocalVideoView	设置本地用户视频渲染的控件。
openLocalCamera	打开本地摄像头。
closeLocalCamera	关闭本地摄像头。
updateVideoQuality	更新本地视频编码质量设置。
startScreenSharing	开始屏幕分享。
stopScreenSharing	结束屏幕分享。
getScreenSharingTargetList	枚举可分享的屏幕和窗口。
selectScreenSharingTarget	选取要分享的屏幕或窗口。
startPushLocalVideo	开始推送本地视频。
stopPushLocalVideo	停止推送本地视频。

本地用户音频管理

API	描述
openLocalMicrophone	打开本地麦克风。
closeLocalMicrophone	关闭本地麦克风。
updateAudioQuality	更新本地音频编码质量设置。
startPushLocalAudio	开始推送本地音频。
stopPushLocalAudio	停止推送本地音频。

远端用户视图渲染、视频管理

版权所有：腾讯云计算（北京）有限责任公司

API	描述
setRemoteVideoView	设置远端用户视频渲染的控件。
startPlayRemoteVideo	开始播放远端用户视频。
stopPlayRemoteVideo	停止播放远端用户视频。
muteRemoteAudioStream	将远端用户禁音。

房间内用户信息

API	描述
getUserList	获取房间内的成员列表。
getUserInfo	获取成员信息。

房间内用户管理

API	描述
changeUserRole	修改用户角色（只有管理员或群主能够调用）。
kickRemoteUserOutOfRoom	将远端用户踢出房间（只有管理员或群主能够调用）。

房间内用户发言管理

API	描述
disableDeviceForAllUserByAdmin	控制当前房间内所有用户是否可打开音频流、视频流采集设备的权限状态，例如：全员禁止打开麦克风、全员禁止打开摄像头、全员禁止打开屏幕分享（目前仅会议场景下可用，并且只有管理员或群主能够调用）。
openRemoteDeviceByAdmin	请求远端用户打开媒体设备（只有管理员或群主能够调用）。
closeRemoteDeviceByAdmin	关闭远端用户媒体设备（只有管理员或群主能够调用）。
applyToAdminToOpenLocalDevice	请求打开本地媒体设备（普通用户可用）。

房间内麦位管理

版权所有：腾讯云计算（北京）有限责任公司

API	描述
setMaxSeatCount	设置最大麦位数（仅支持进房前和创建房间时设置）。
getSeatList	获取麦位列表。
lockSeatByAdmin	锁定麦位（只有管理员或群主能够调用，包括位置锁定、音频状态锁定和视频状态锁定）。
takeSeat	申请上麦（自由发言模式下，不需要申请）。
leaveSeat	申请下麦（自由发言模式下，不需要申请）。
takeUserOnSeatByAdmin	主持人/管理员 邀请用户上麦。
kickUserOffSeatByAdmin	主持人/管理员 将用户踢下麦。

发送消息

API	描述
sendTextMessage	发送本文消息。
sendCustomMessage	发送自定义消息。
disableSendingMessageByAdmin	禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）。
disableSendingMessageForAllUser	禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）。

信令管理

API	描述
cancelRequest	取消请求。
responseRemoteRequest	回复请求。

高级功能：获取 TRTC 实例

API	描述
getTRTCCloud	获取原生 TRTC 实例类。
getDeviceManager	获取原生 TRTC 设备管理类。

版权所有：腾讯云计算（北京）有限责任公司

getAudioEffectManager	获取原生 TRTC 音效类。

TUIRoomObserver 回调事件

TUIRoomObserver 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己需要的回调事件。
TUIRoomObserver

TUIRoomObserver

错误回调

API	描述
onError	错误事件回调。

登录状态事件回调

API	描述
onKickedOffLine	其他终端登录被踢下线。
onUserSigExpired	用户凭证超时事件。

房间内事件回调

API	描述
onRoomNameChanged	房间名称更改事件。
onAllUserMicrophoneDisableChanged	房间内所有用户麦克风被禁用事件。
onAllUserCameraDisableChanged	房间内所有用户摄像头被禁用事件。
onSendMessageForAllUserDisableChanged	房间内所有用户发送文本消息被禁用事件。
onKickedOutOfRoom	被踢出房间事件。
onRoomDismissed	房间被解散事件。
onRoomSpeechModeChanged	房间麦控模式发生变化

版权所有：腾讯云计算（北京）有限责任公司

房间内用户事件回调

API	描述
onRemoteUserEnterRoom	远端用户进房事件。
onRemoteUserLeaveRoom	远端用户离开房间事件。
onUserRoleChanged	用户角色发生变化事件。
onUserVideoStateChanged	用户视频状态发生变化事件。
onUserAudioStateChanged	用户音频状态发生变化事件。
onUserVoiceVolumeChanged	用户音量变化事件。
onSendMessageForUserDisableChanged	用户文本消息发送能力发生变化事件
onUserNetworkQualityChanged	用户网络状态变化事件。
onUserScreenCaptureStopped	用户屏幕采集停止事件。

房间内麦位事件回调

API	描述
onRoomMaxSeatCountChanged	房间内最大麦位数发生变化事件（仅在会议类型房间生效）
onSeatListChanged	麦位列表发生变化事件。
onKickedOffSeat	收到用户被踢下麦事件。

请求信令事件回调

API	描述
onRequestReceived	收到请求消息事件。
onRequestCancelled	收到请求被取消的事件。

房间内消息事件回调

API	描述
onReceiveTextMessage	收到普通文本消息事件。

版权所有：腾讯云计算（北京）有限责任公司

onReceiveCustomMessage	收到自定义消息事件。

TUIRoomEngine

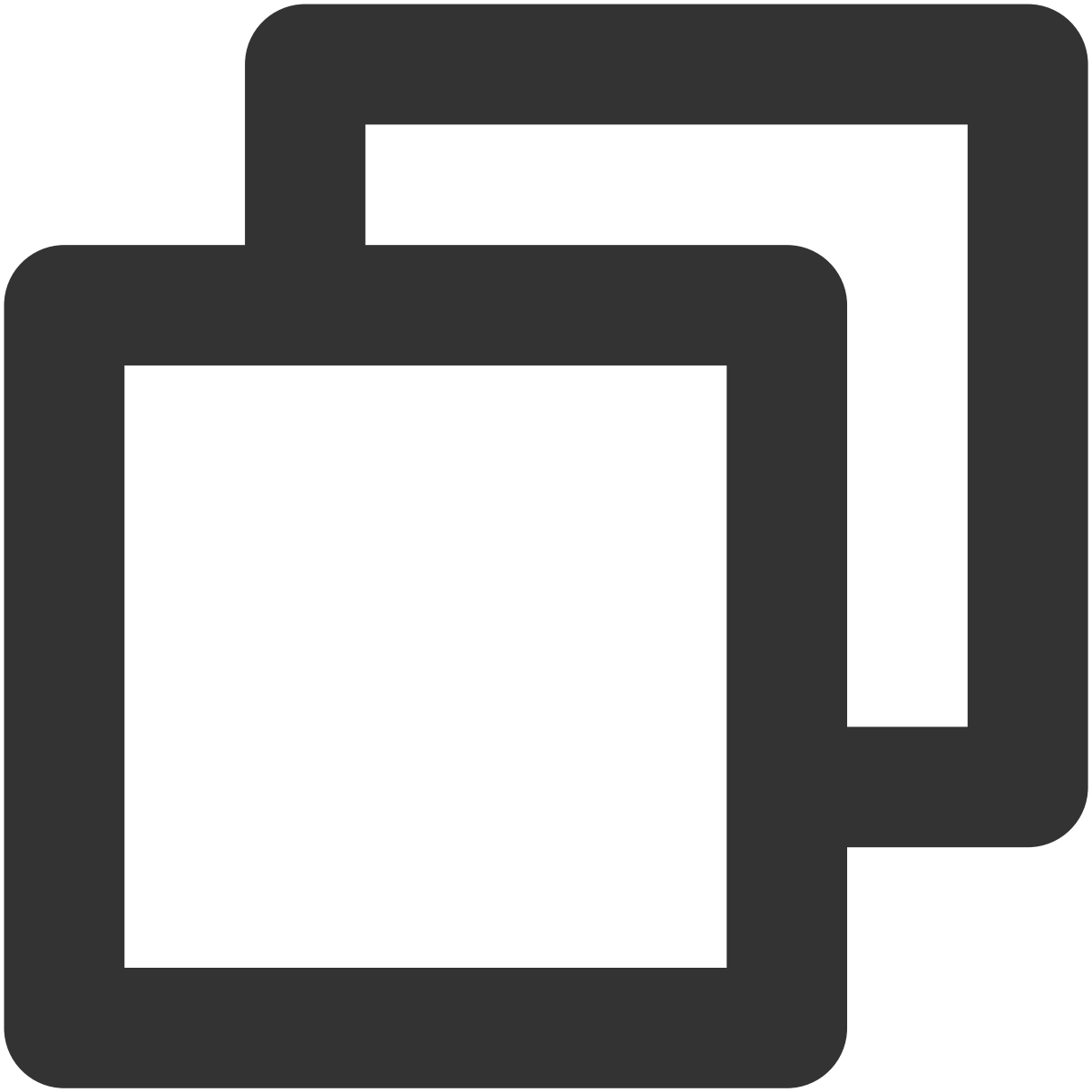
最近更新时间：2023-10-30 10:57:25

TUIRoomEngine API 简介

TUIRoomEngine API 是音视频通话组件的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。

createTUIRoomEngine

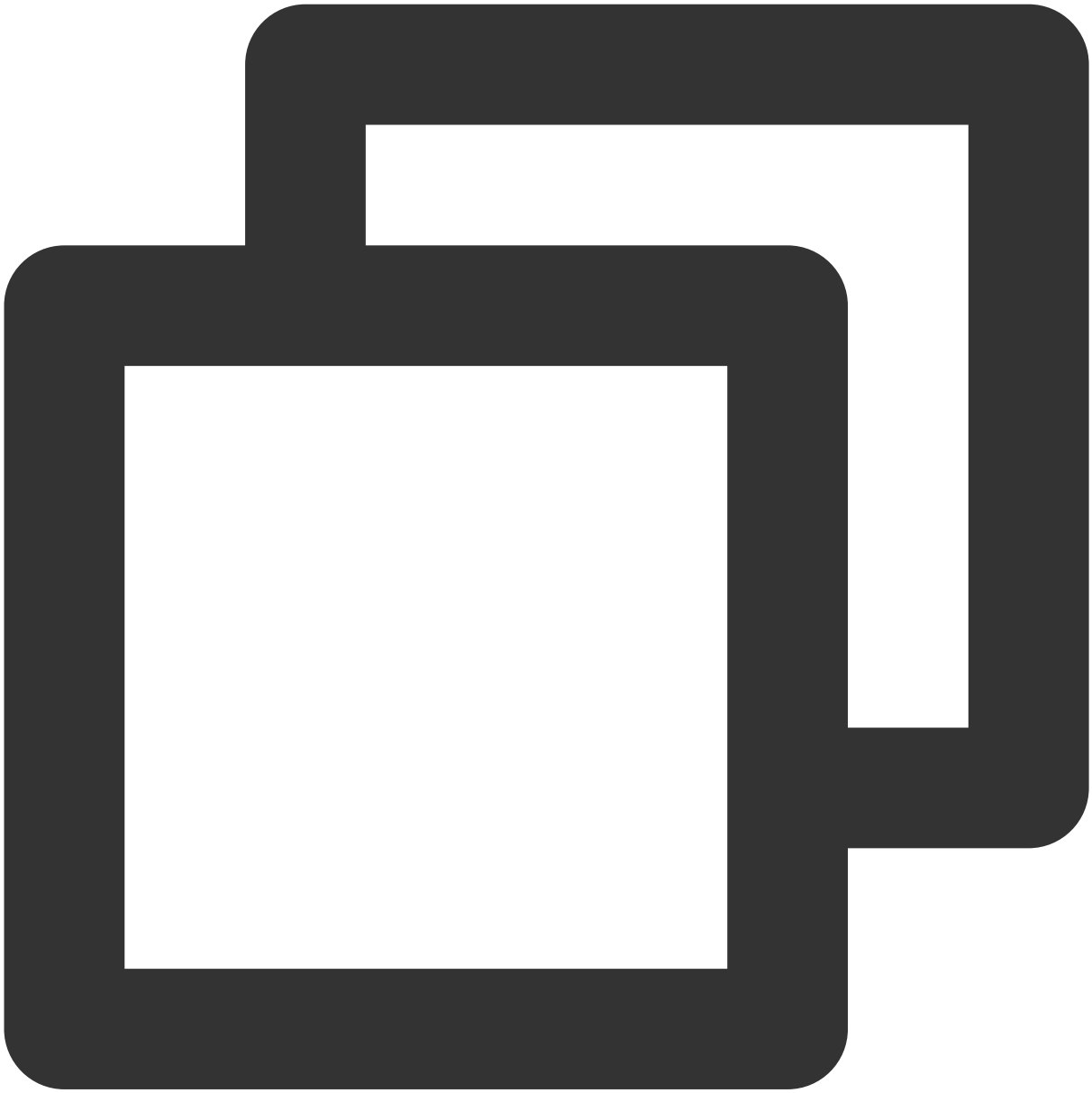
创建 TUIRoomEngine 实例



```
tuikit::TUIRoomEngine* createTUIRoomEngine();
```

destroyTUIRoomEngine

销毁 TUIRoomEngine 实例

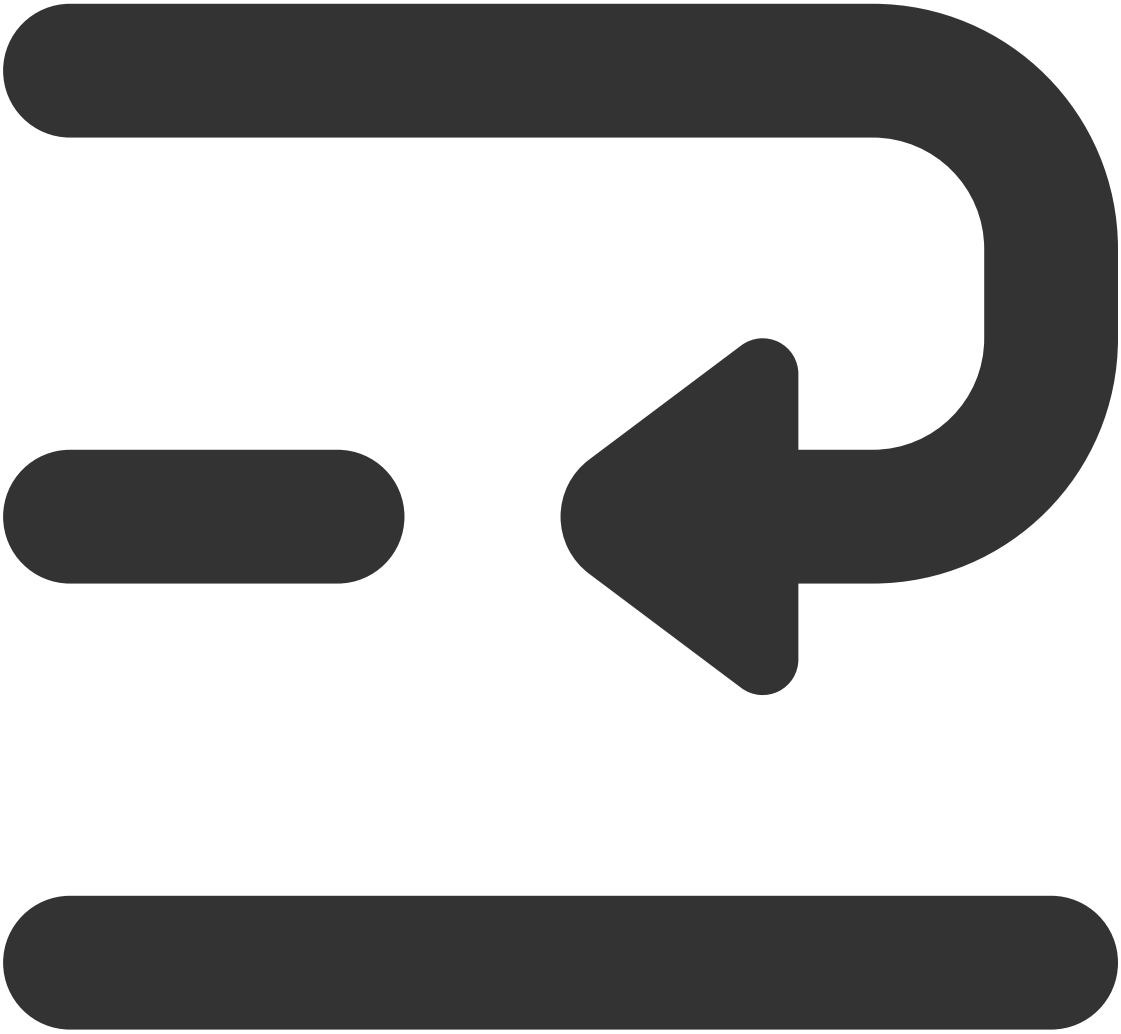


```
void destroyTUIRoomEngine(tuikit::TUIRoomEngine* roomEngine);
```

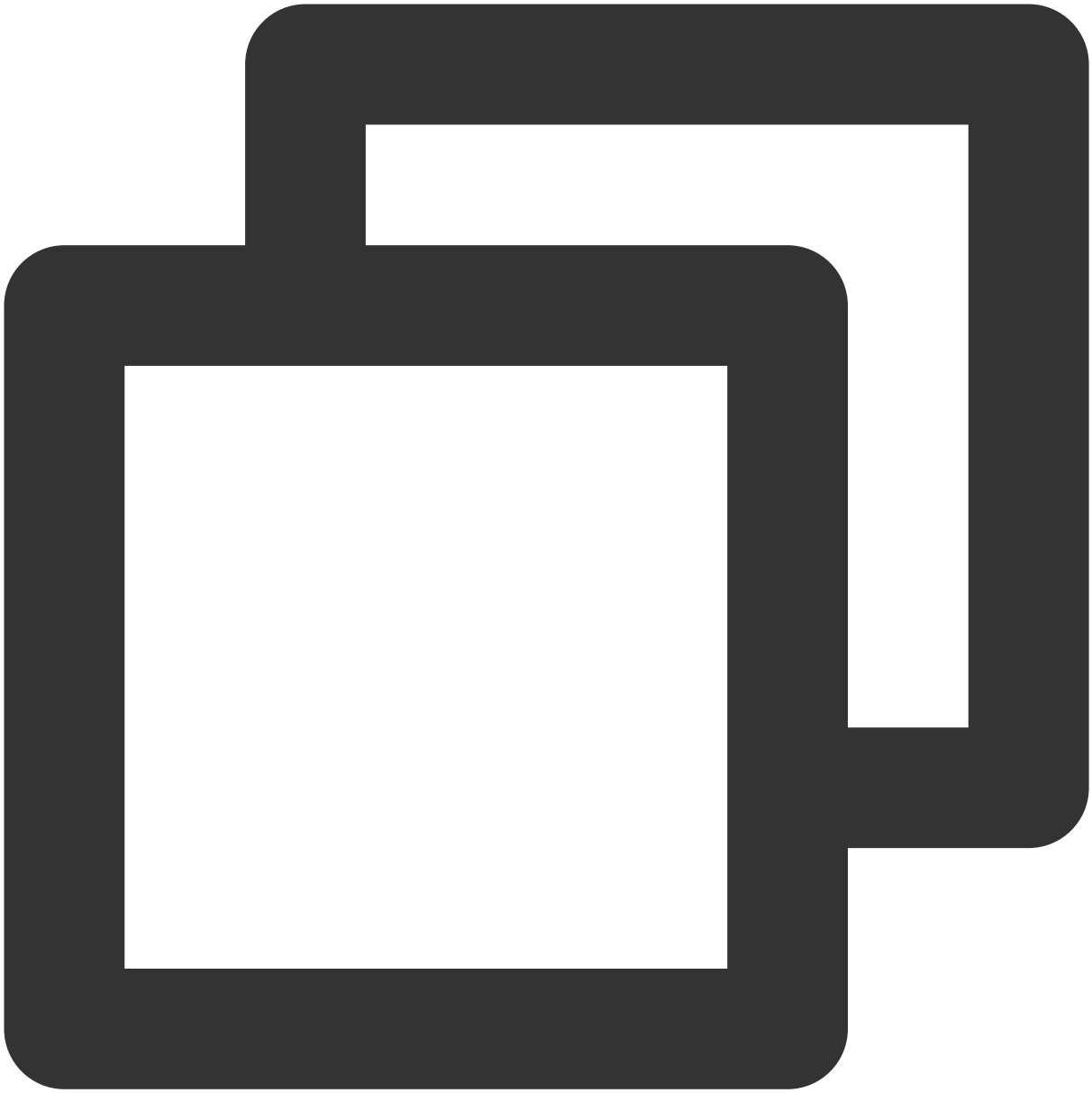
参数	类型	含义
roomEngine	tuikit::TUIRoomEngine*	TUIRoomEngine 实例指针，该指针只能通过 CreateTUIRoomEngine 接口获取。

login

登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。



版权所有：腾讯云计算（北京）有限责任公司



```
static void login(int sdkAppId, const char* userId, const char* userSig, TUICallbac
```

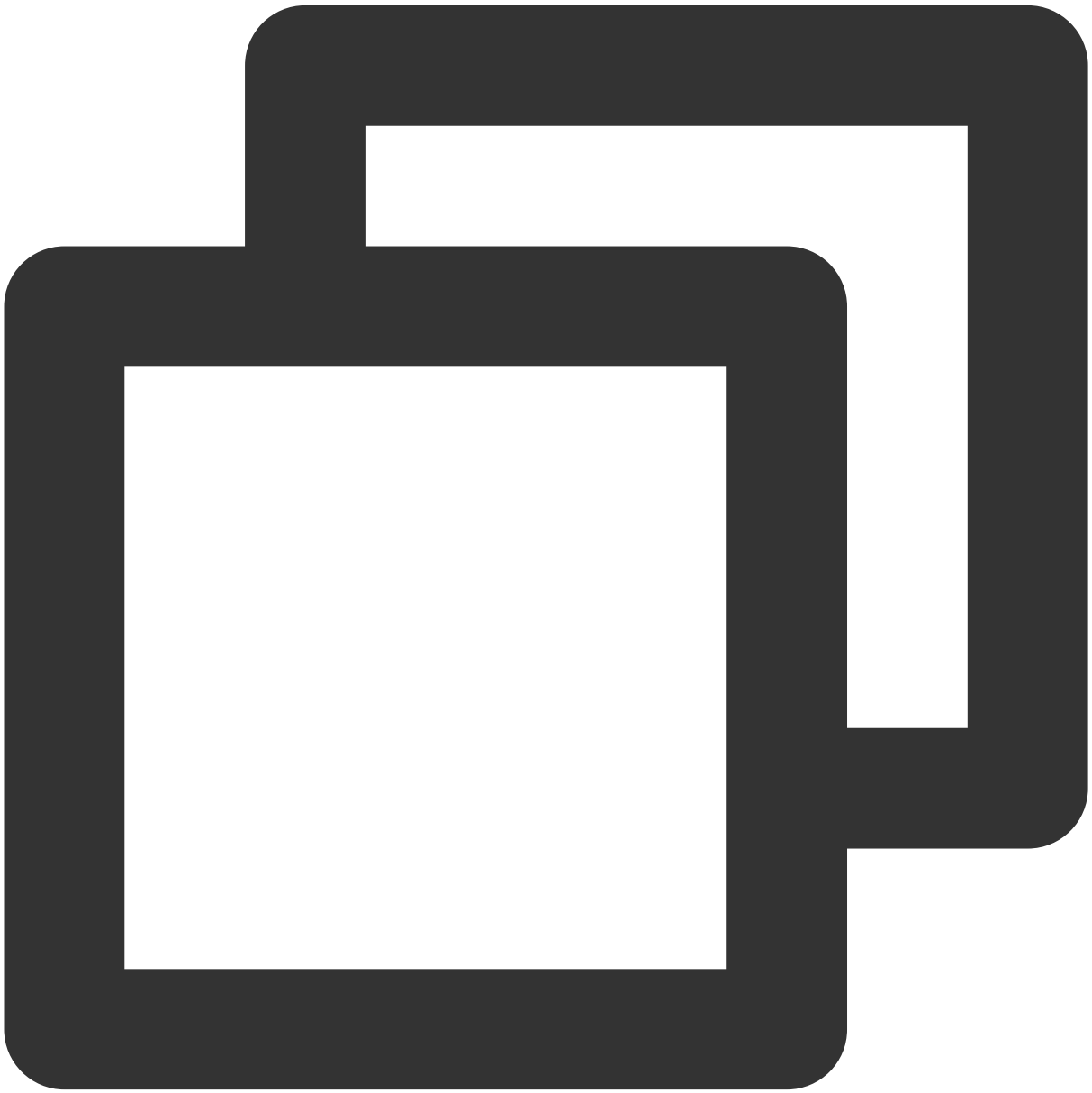
参数如下表所示：

参数	类型	含义
sdkAppId	int	云通信应用的 SDKAppID
userId	const char*	用户ID，用于区分不同用户
userSig	const char*	用户签名，用于腾讯云流量的认证

callback	TUICallback*	调用接口的回调，同于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

logout

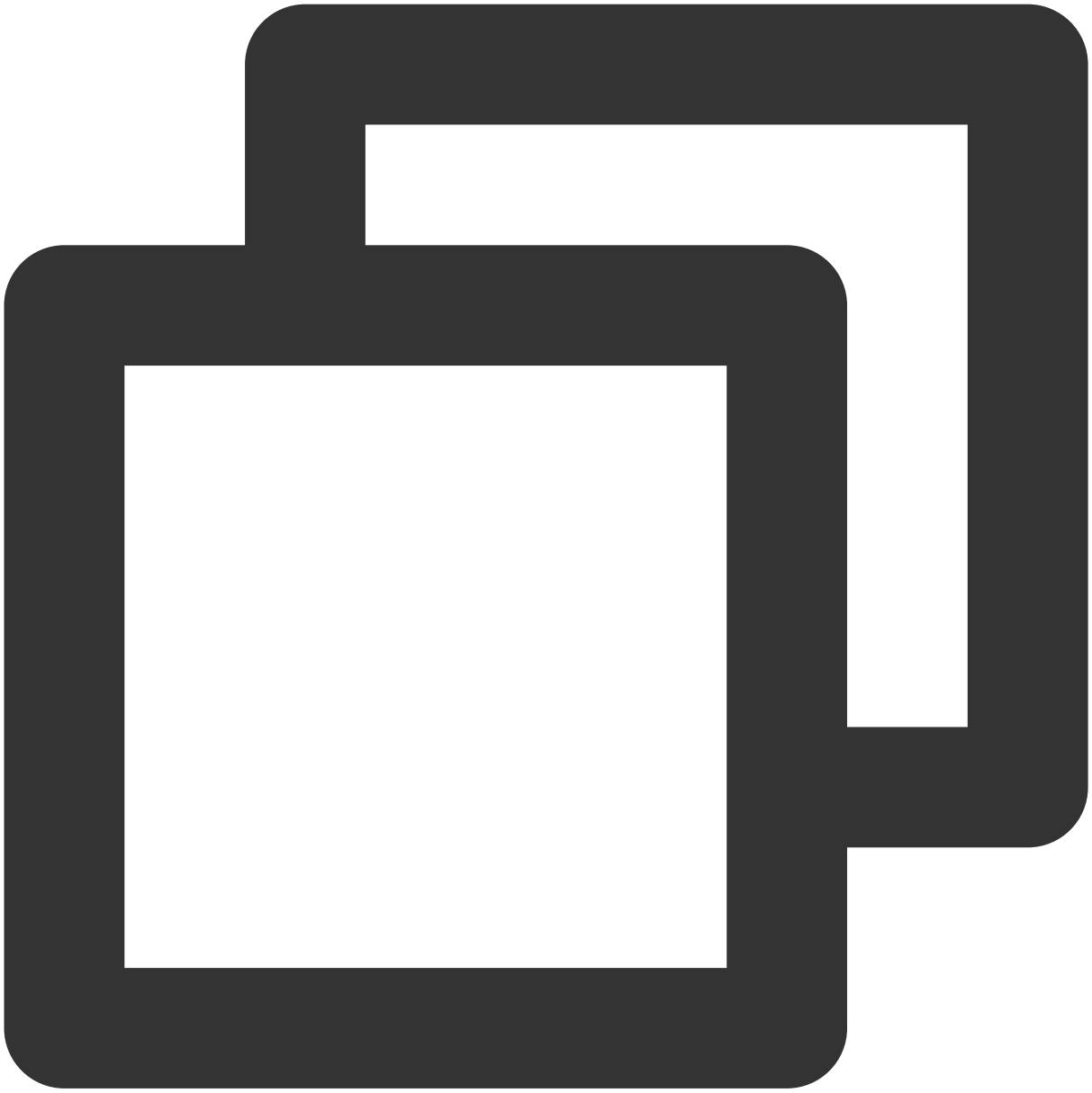
退出登录接口，会有主动离开房间操作、销毁资源。



```
static void logout(TUICallback* callback);
```

setSelfInfo

设置本地用户名称和头像。



```
static void SetSelfInfo(const char* userName, const char* avatarUrl, TUICallback* c
```

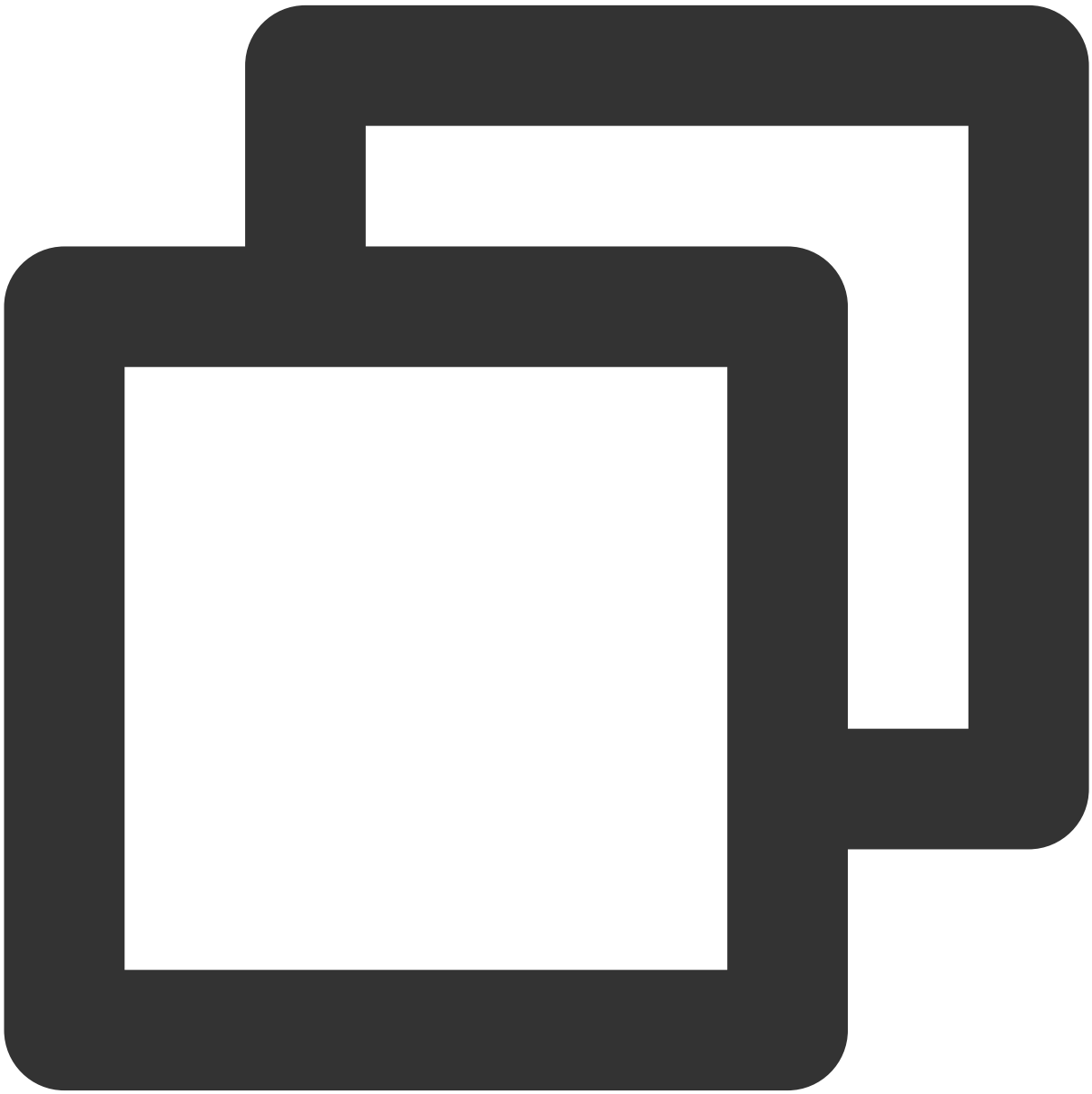
参数如下表所示：

参数	类型	含义
userName	const char*	用户名称
avatarUrl	const char*	用户头像 URL 地址

callback	TUICallback*	调用接口的回调，同于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUICallback 的定义

getSelfInfo

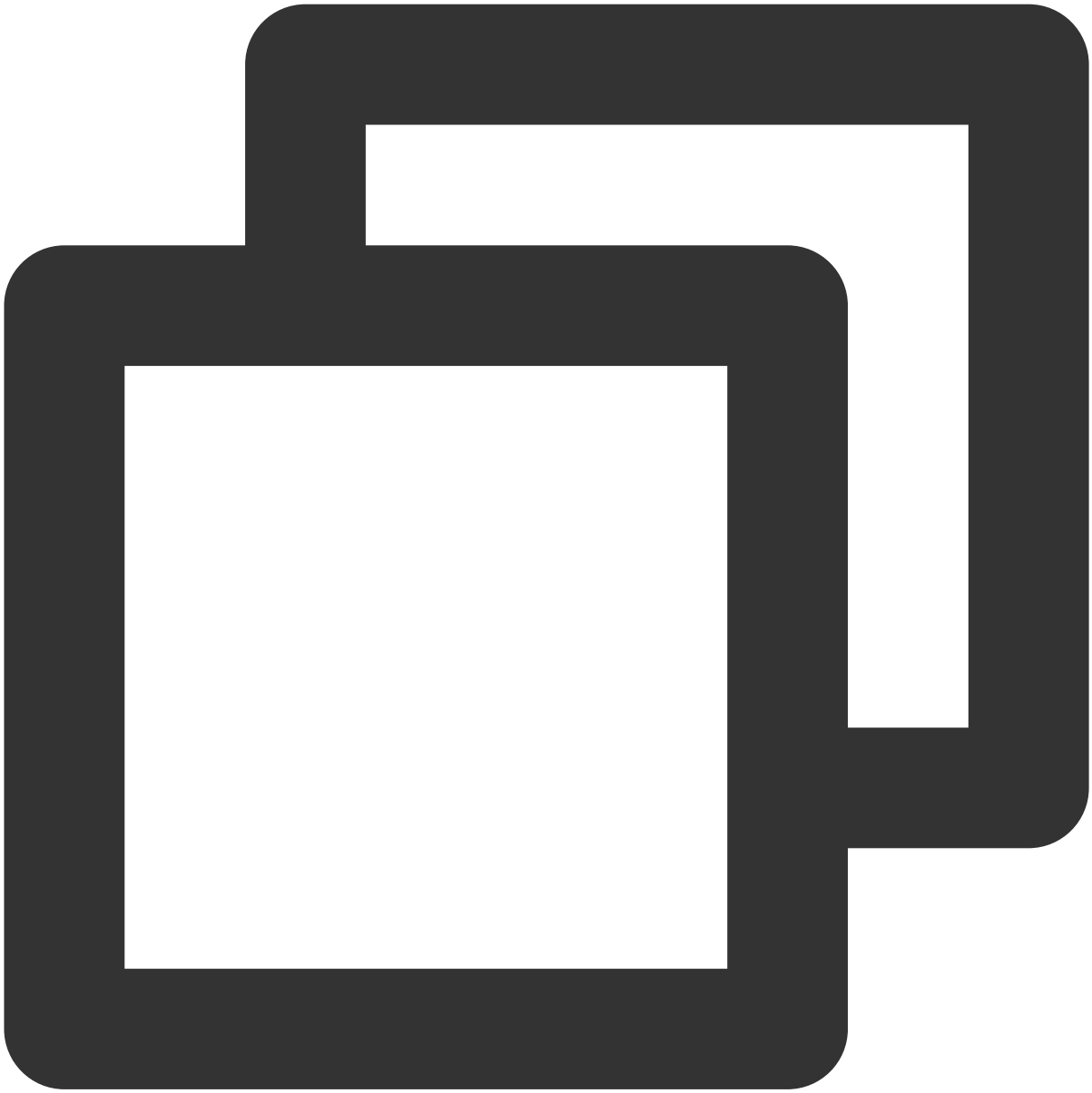
获取本地用户基本信息。



```
static TUILoginUserInfo GetSelfInfo();
```

addObserver

设置事件回调。



```
virtual void AddObserver(TUIRoomObserver* observer) = 0;
```

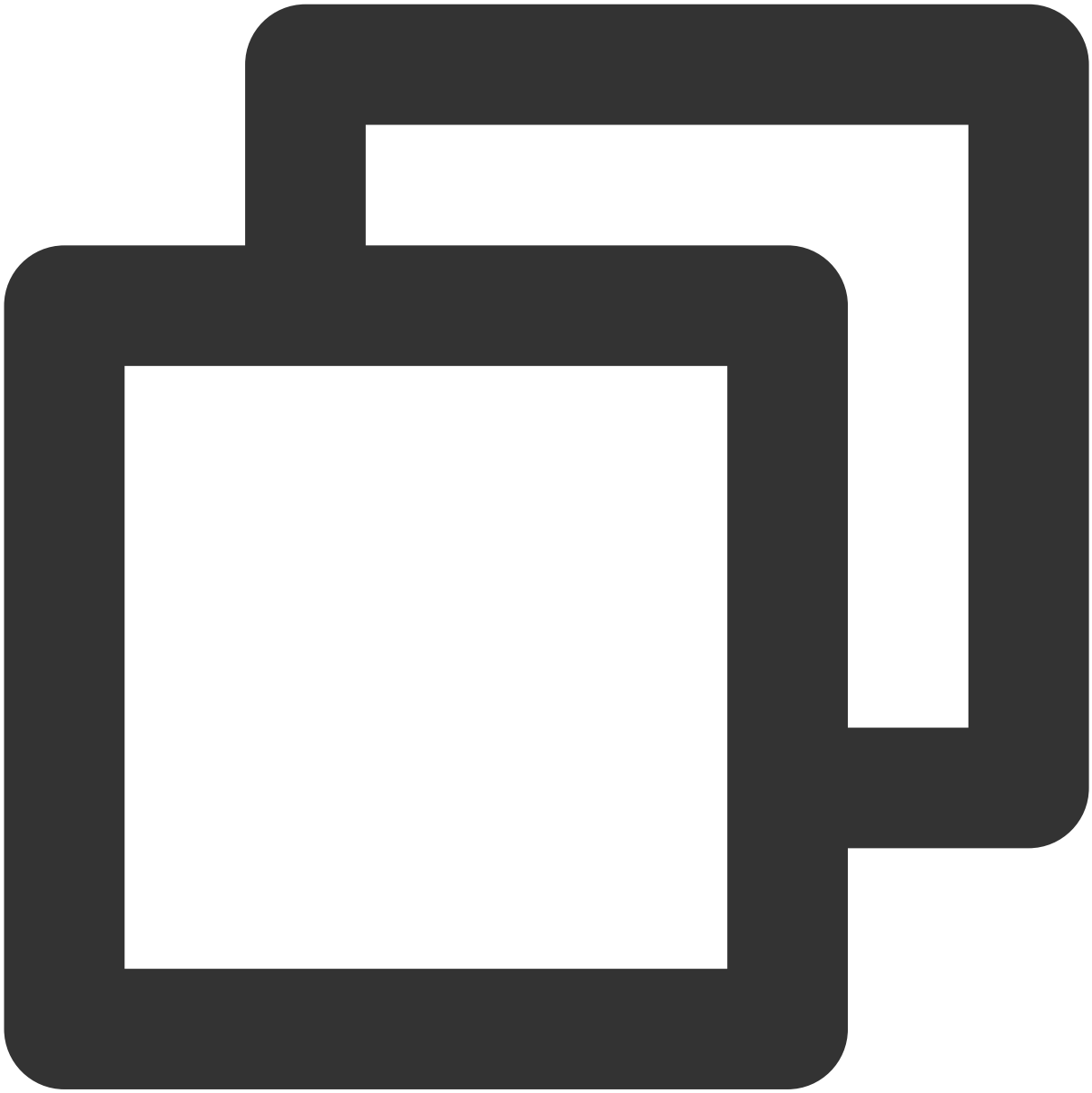
参数如下表所示：

参数	类型	含义
observer	TUIRoomObserver*	回调实例的指针，您可以通过 TUIRoomObserver 获得各类事件通知（比如：错误码，远端用户进

		房，音视频状态参数等）

removeObserver

移除事件回调。



```
virtual void removeObserver(TUIRoomObserver* observer) = 0;
```

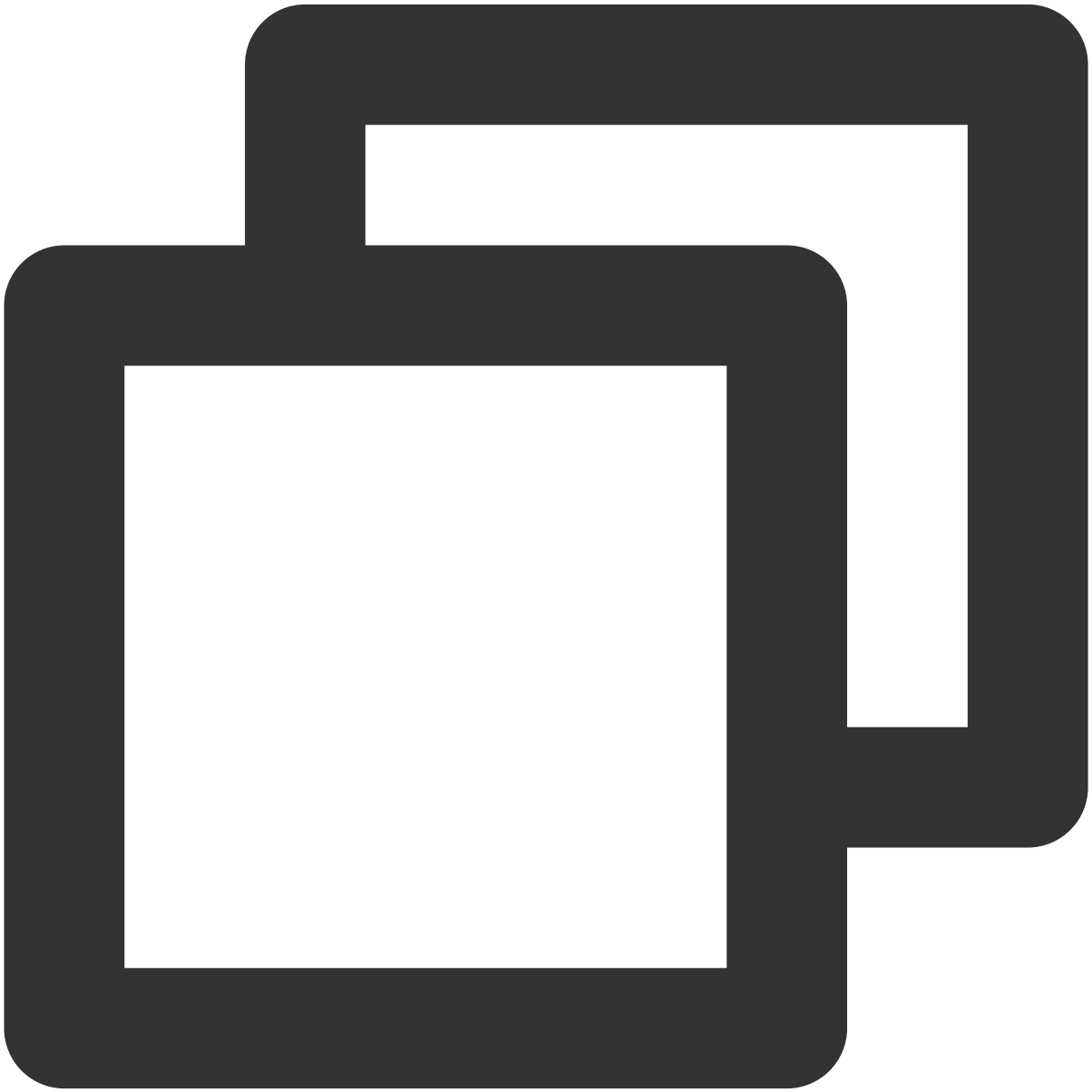
参数如下表所示：

参数	类型	含义

observer	TUIRoomObserver*	回调实例的指针

createRoom

创建房间。



```
virtual void createRoom(const TUIRoomInfo& roomInfo, TUICallback* callback) = 0;
```

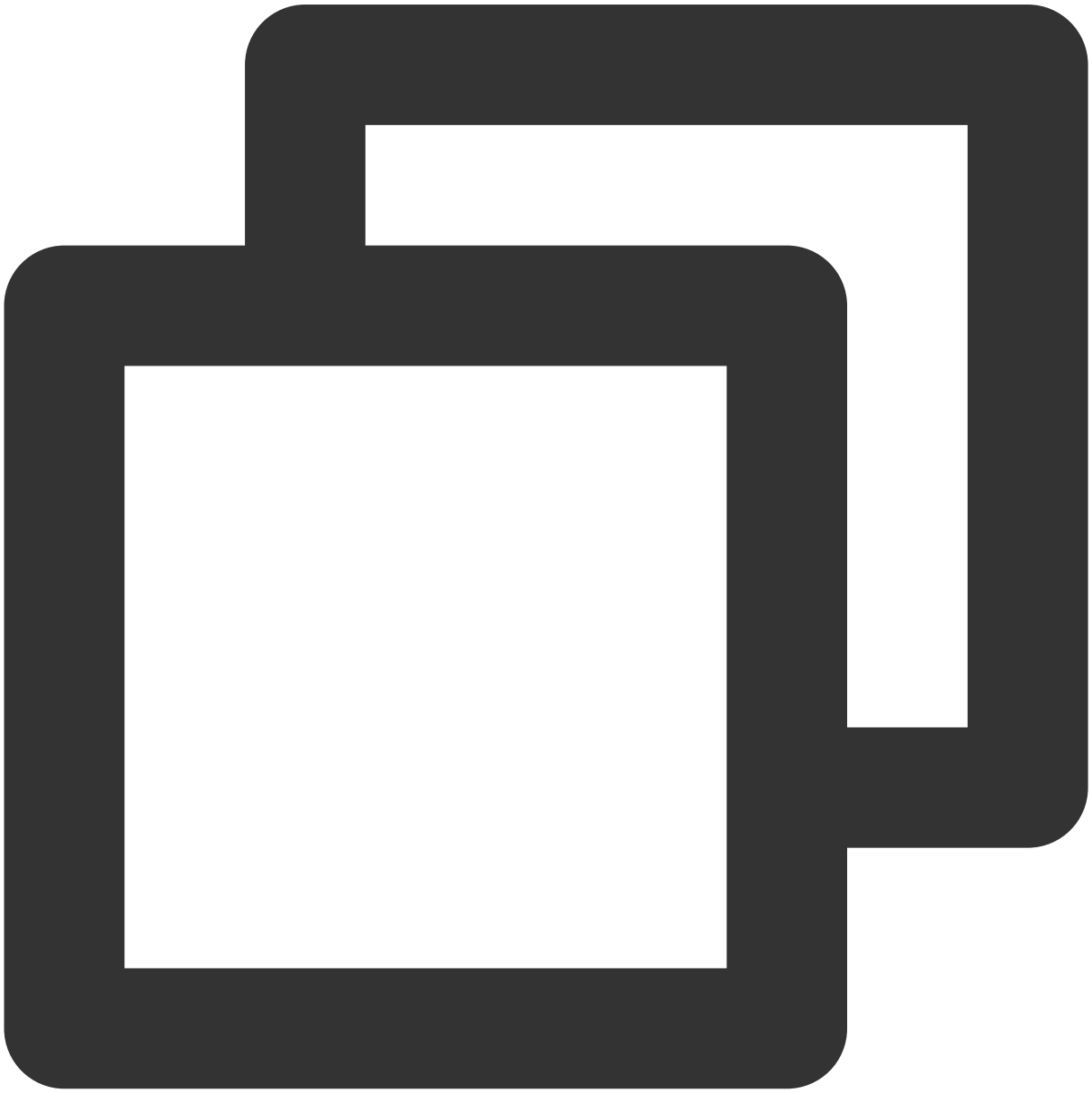
参数如下表所示：

参数	类型	含义

roomInfo	const TUIRoomInfo&	房间信息，可以初始化房间的一些设置，定义参考 TUIRoomDefine.h 中TUIRoomInfo
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUICallback 的定义

destroyRoom

解散房间。



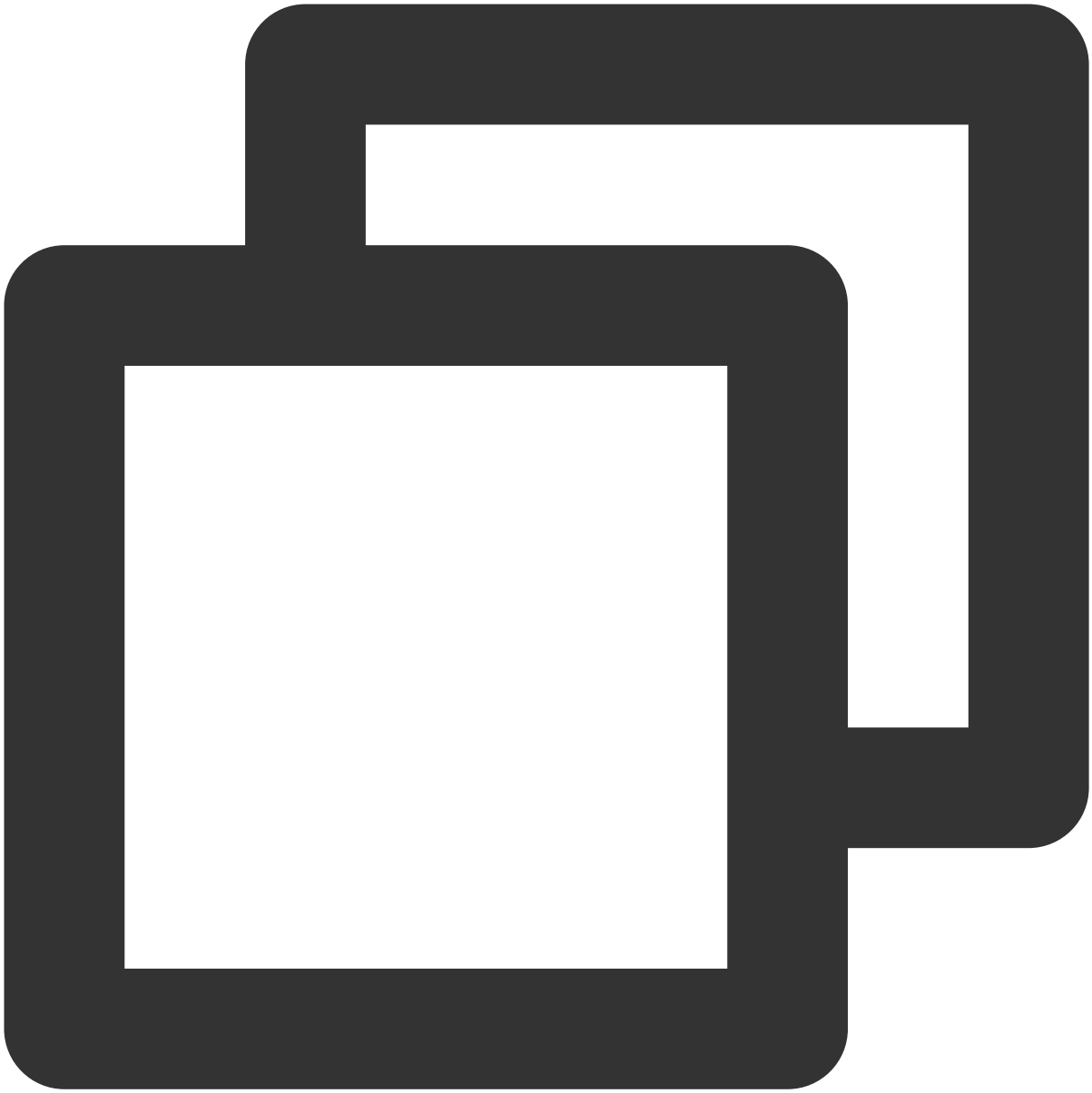
```
virtual void destroyRoom(TUICallback* callback) = 0;
```

参数如下表所示：

参数	类型	含义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUICallback 的定义

enterRoom

进入房间。



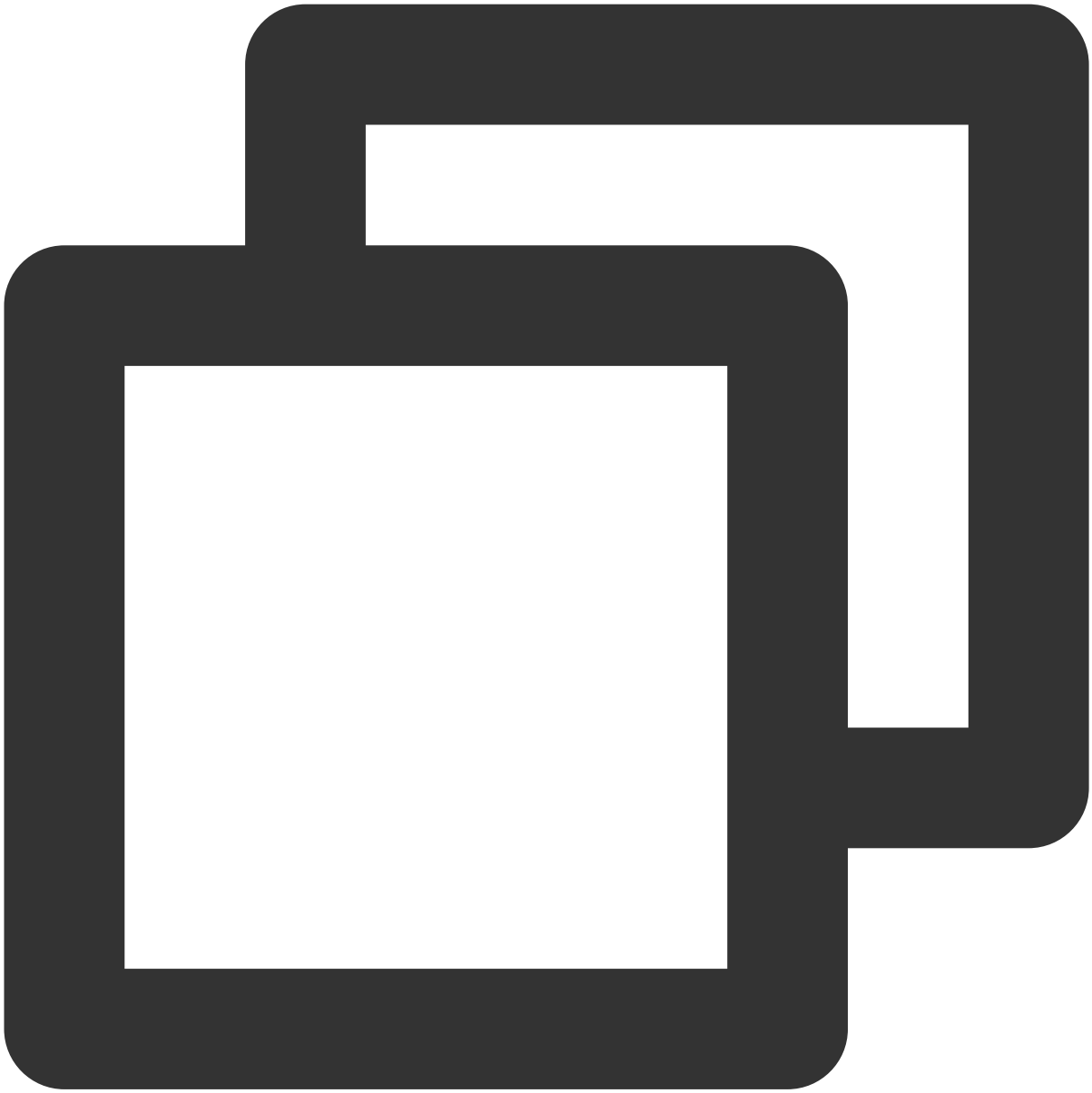
```
virtual void enterRoom(const char* roomId, TUIValueCallback<TUIRoomInfo>* callback)
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
callback	TUIValueCallback<TUIRoomInfo>*	调用接口的回调，携带房间信息。详细定义可以参考TUIRoomDefine.h 中 TUIValueCallback 和 TUIRoomInfo 的定义

exitRoom

离开房间。



```
virtual void exitRoom(bool syncWaiting, TUICallback* callback) = 0;
```

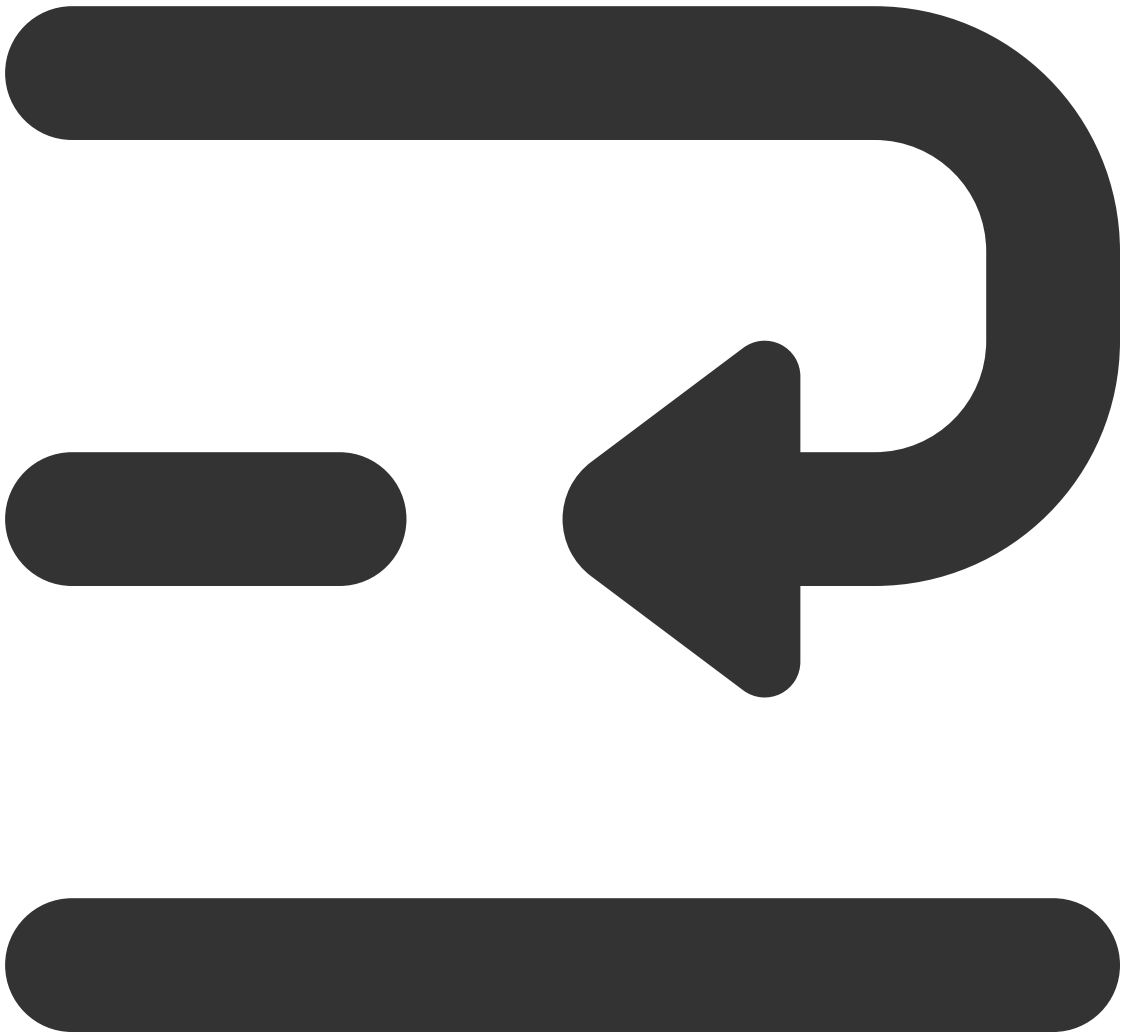
参数如下表所示：

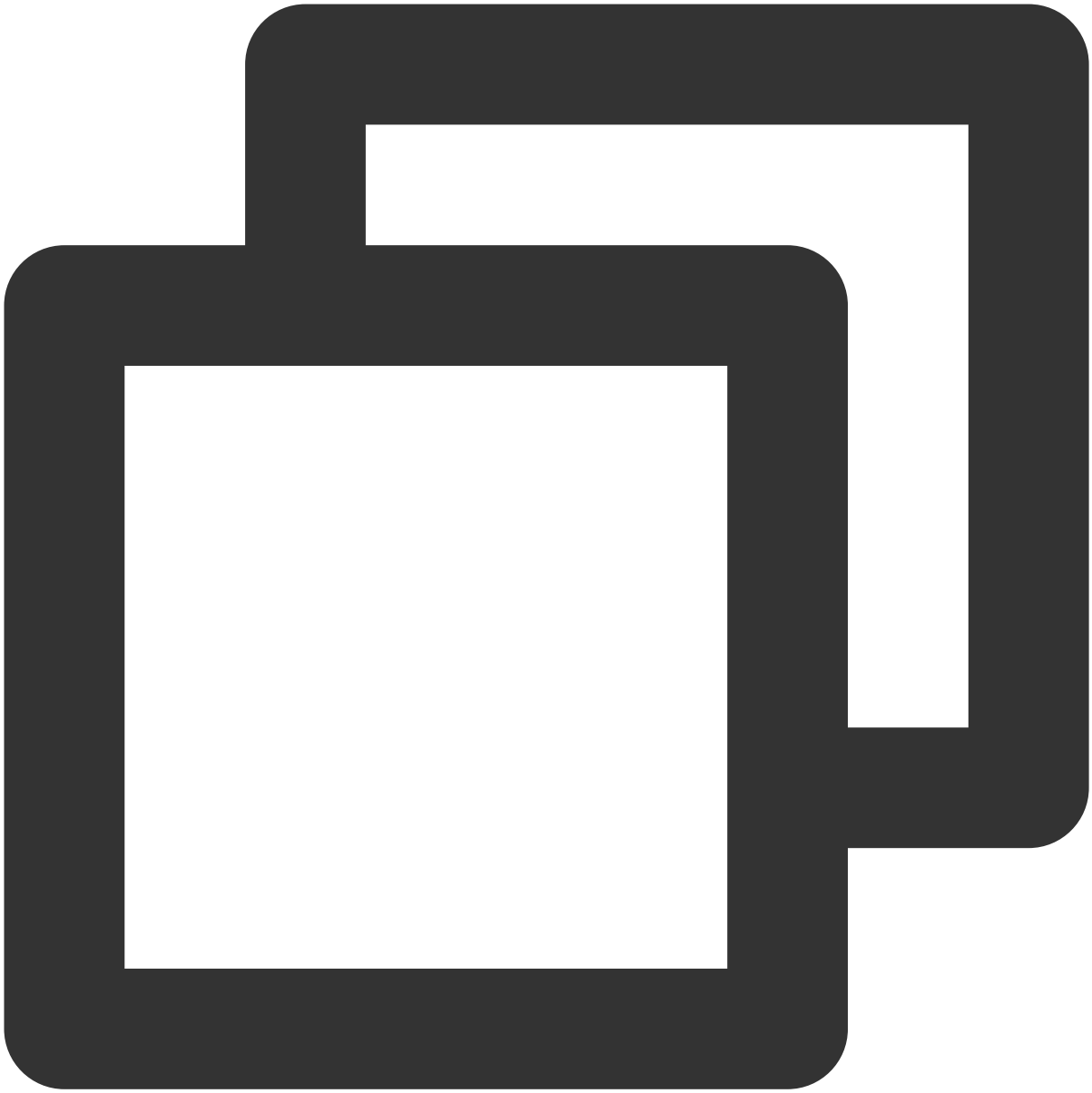
参数	类型	含义
syncWaiting	bool	是否同步等待接口返回
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

connectOtherRoom

连接其他房间。





```
virtual TUIRequest connectOtherRoom(const char* roomId, const char* userId, int tim
```

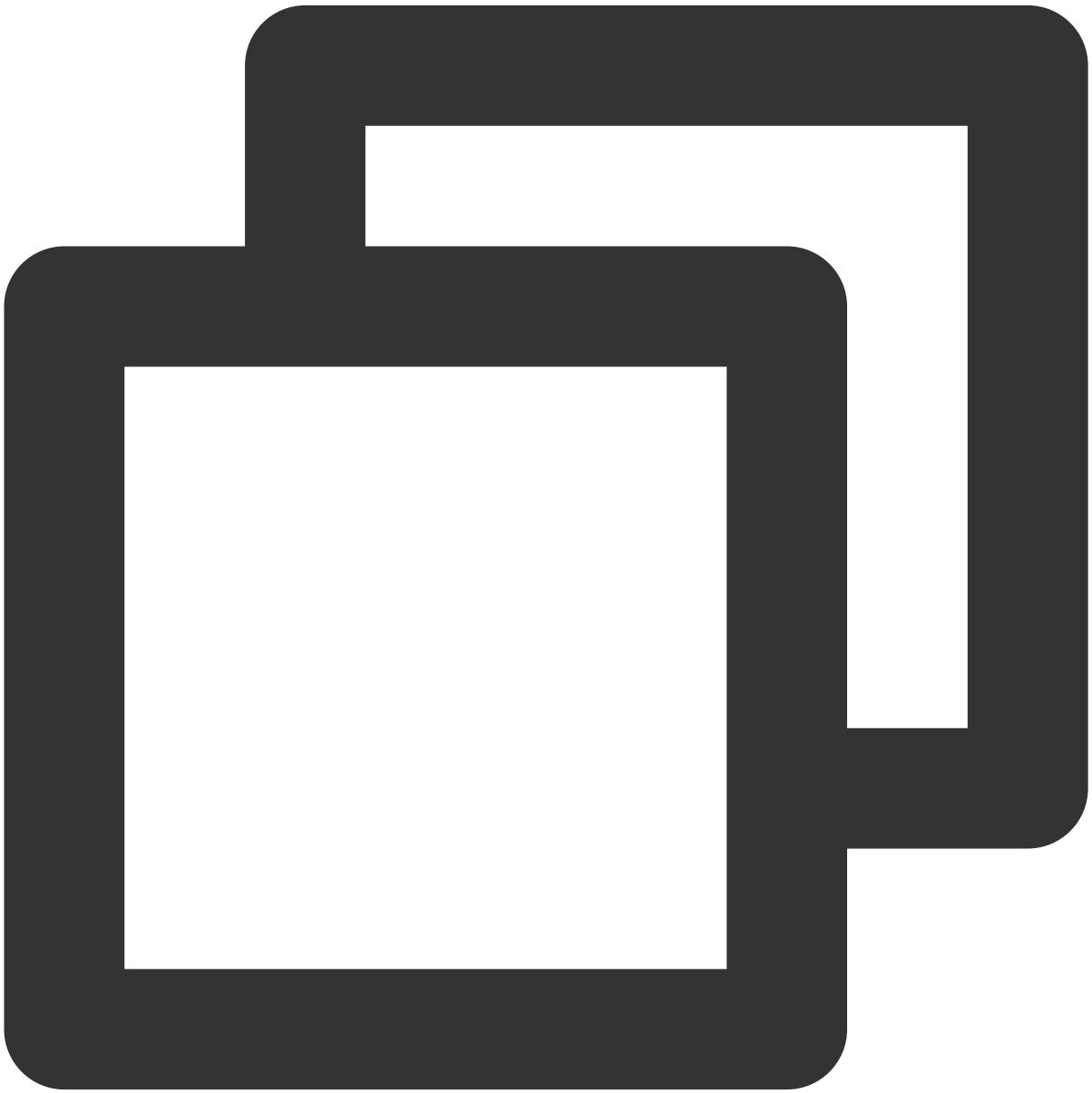
参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
userId	const char*	用户 ID
timeout	int	超时时间

callback	TUIRequestCallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIRequestCallback 的定义

disConnectOtherRoom

断开与其他房间的连接。



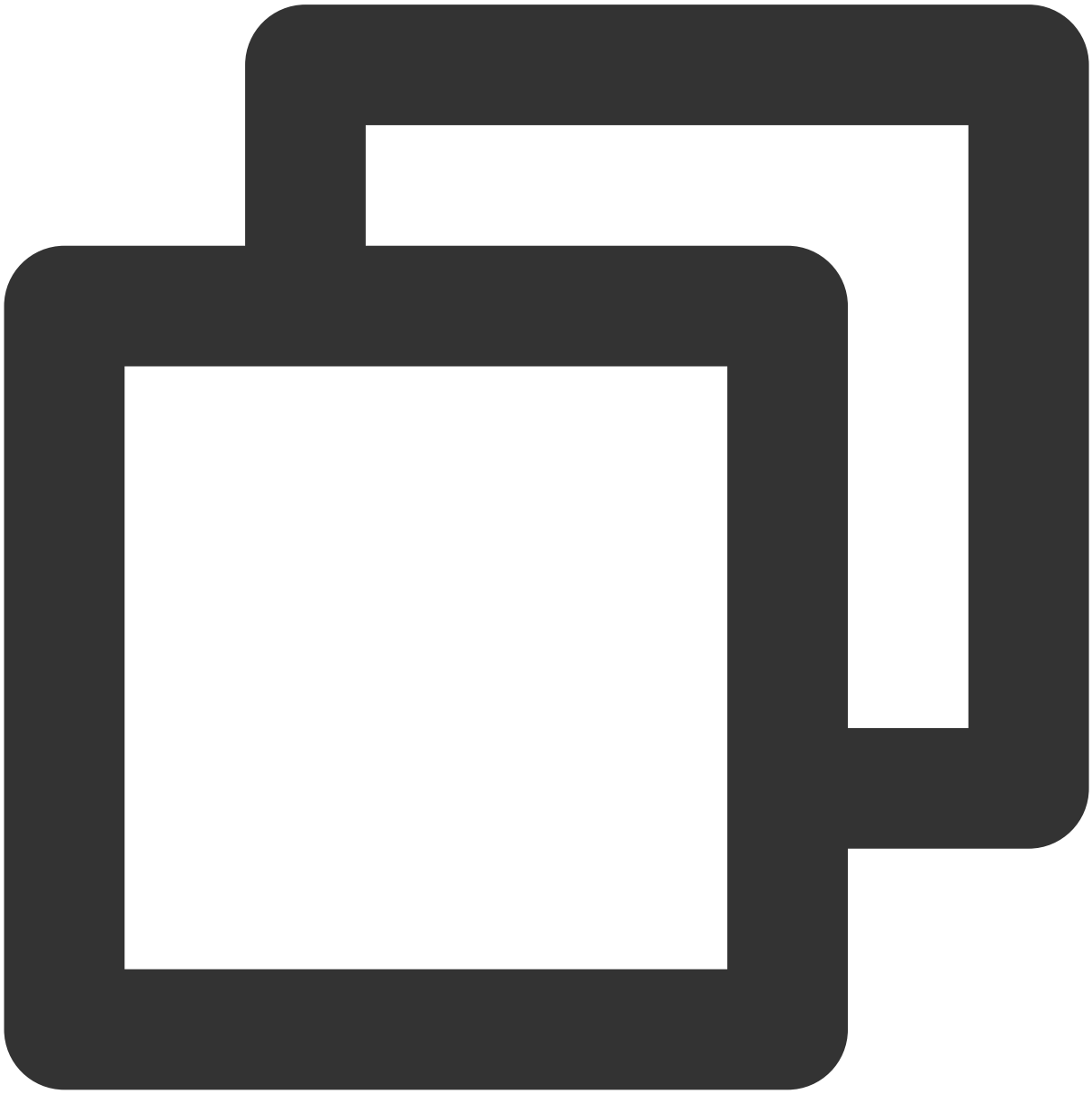
```
virtual void disConnectOtherRoom(TUICallback* callback) = 0;
```

参数如下表所示：

参数	类型	含义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

fetchRoomInfo

获取房间信息。



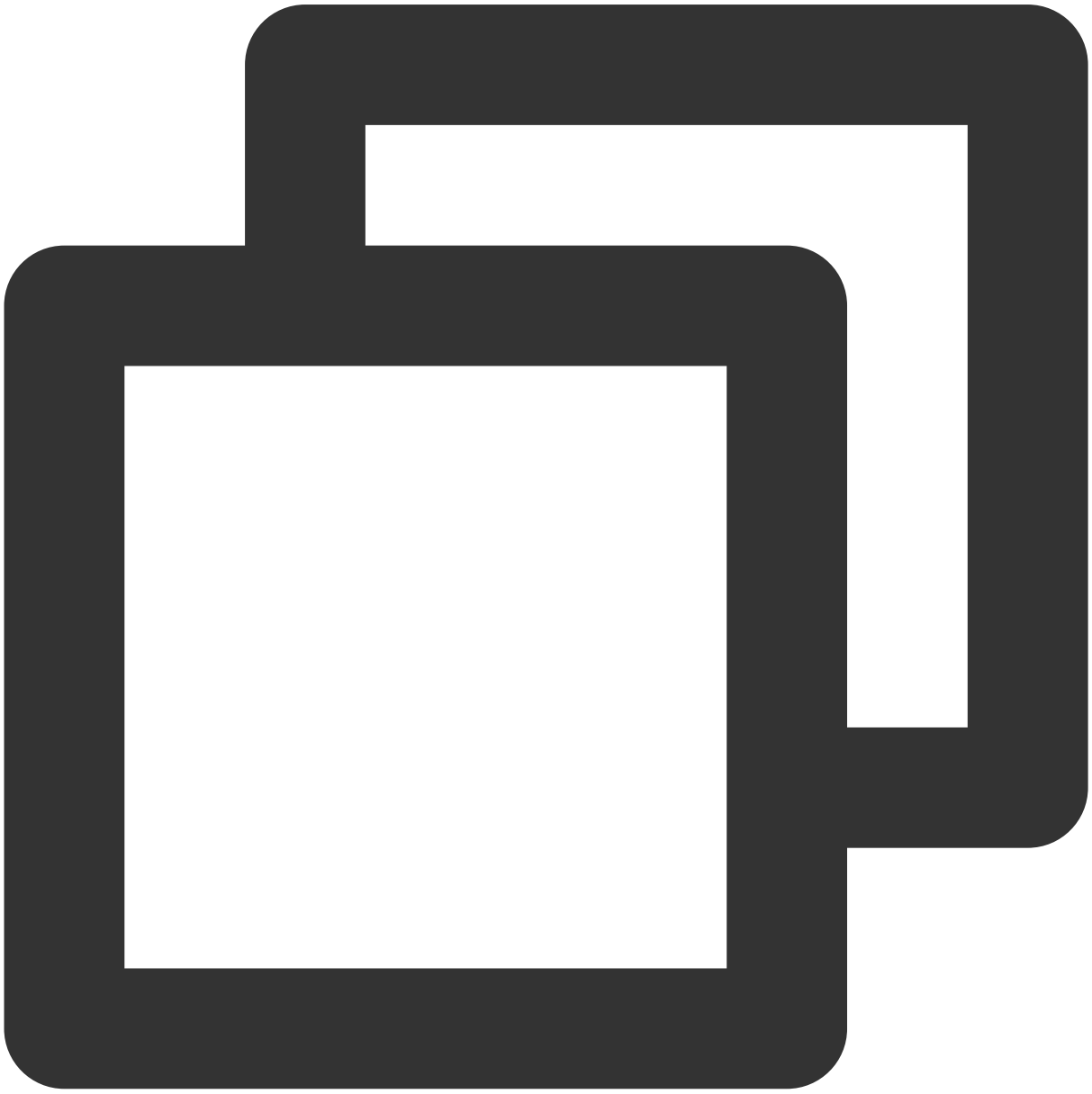
```
virtual void fetchRoomInfo(TUIValueCallback<TUIRoomInfo>* callback) = 0;
```

参数如下表所示：

参数	类型	含义
callback	TUIValueCallback<TUIRoomInfo>*	调用接口的回调，携带房间信息。详细定义可以参考TUIRoomDefine.h 中 TUIValueCallback 和 TUIRoomInfo 的定义

updateRoomNameByAdmin

更新房间名称。



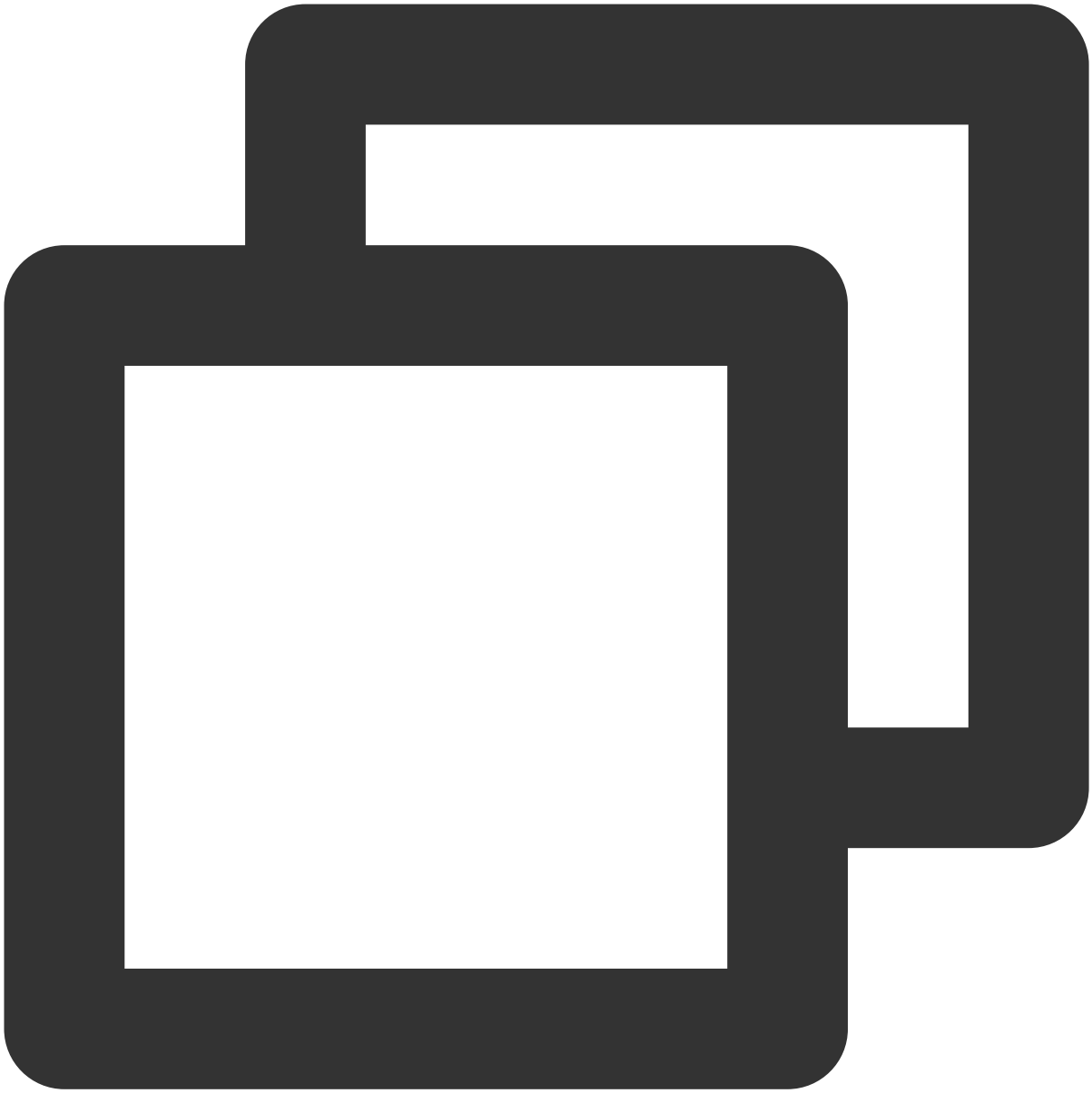
```
virtual void updateRoomNameByAdmin(const char* roomName, TUICallback* callback) = 0
```

参数如下表所示：

参数	类型	含义
roomName	const char*	房间名称
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

updateRoomSpeechModeByAdmin

更新房间麦控模式。



```
virtual void updateRoomSpeechModeByAdmin(TUISpeechMode mode, TUICallback* callback)
```

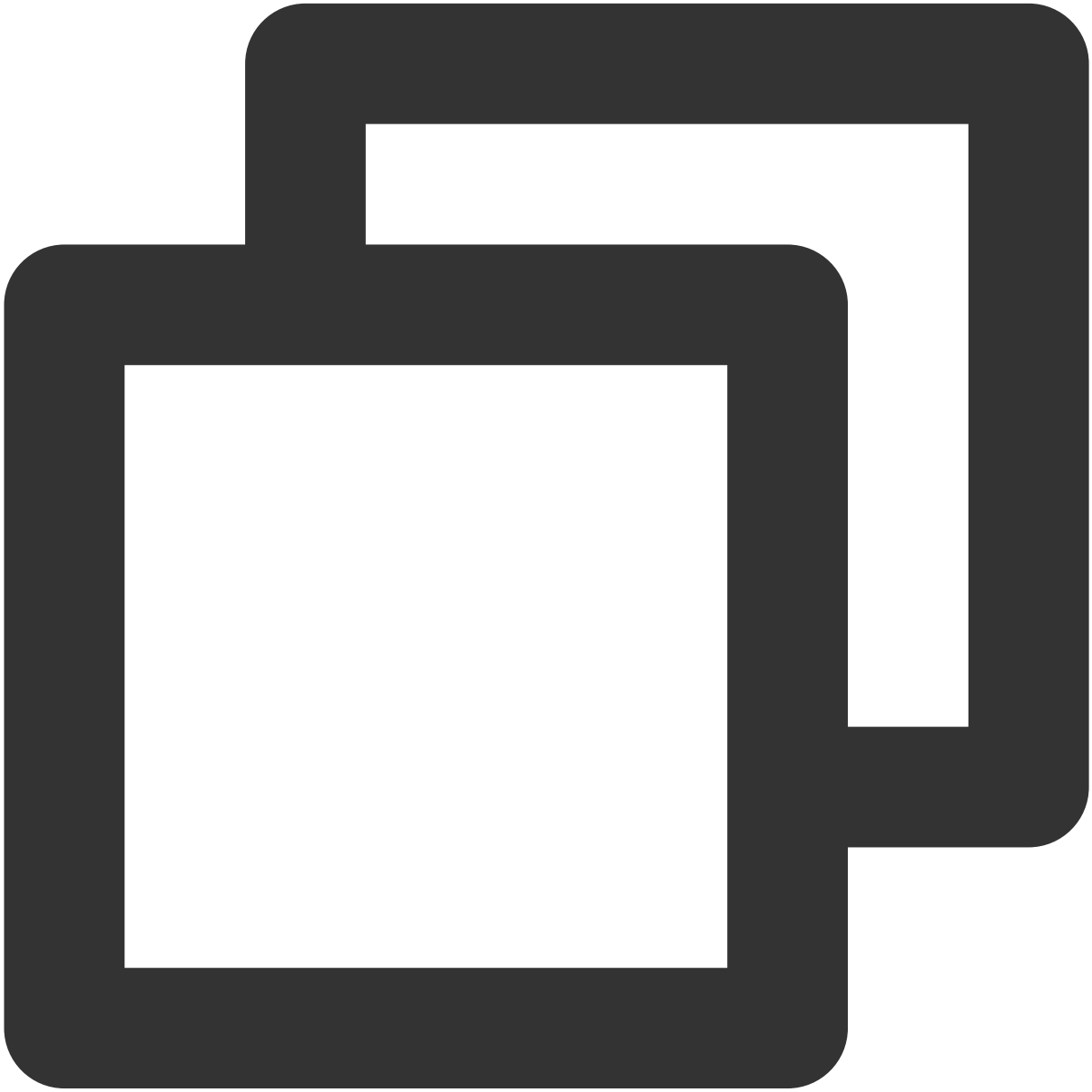
参数如下表所示：

参数	类型	含义
mode	TUISpeechMode	麦控模式,详细定义可以参考 TUIRoomDefine.h 文件的 TUISpeechMode
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

setLocalVideoView

设置本地用户视频渲染的控件。



```
virtual void setLocalVideoView(TUIVideoStreamType streamType, const TUIVideoView& v
```

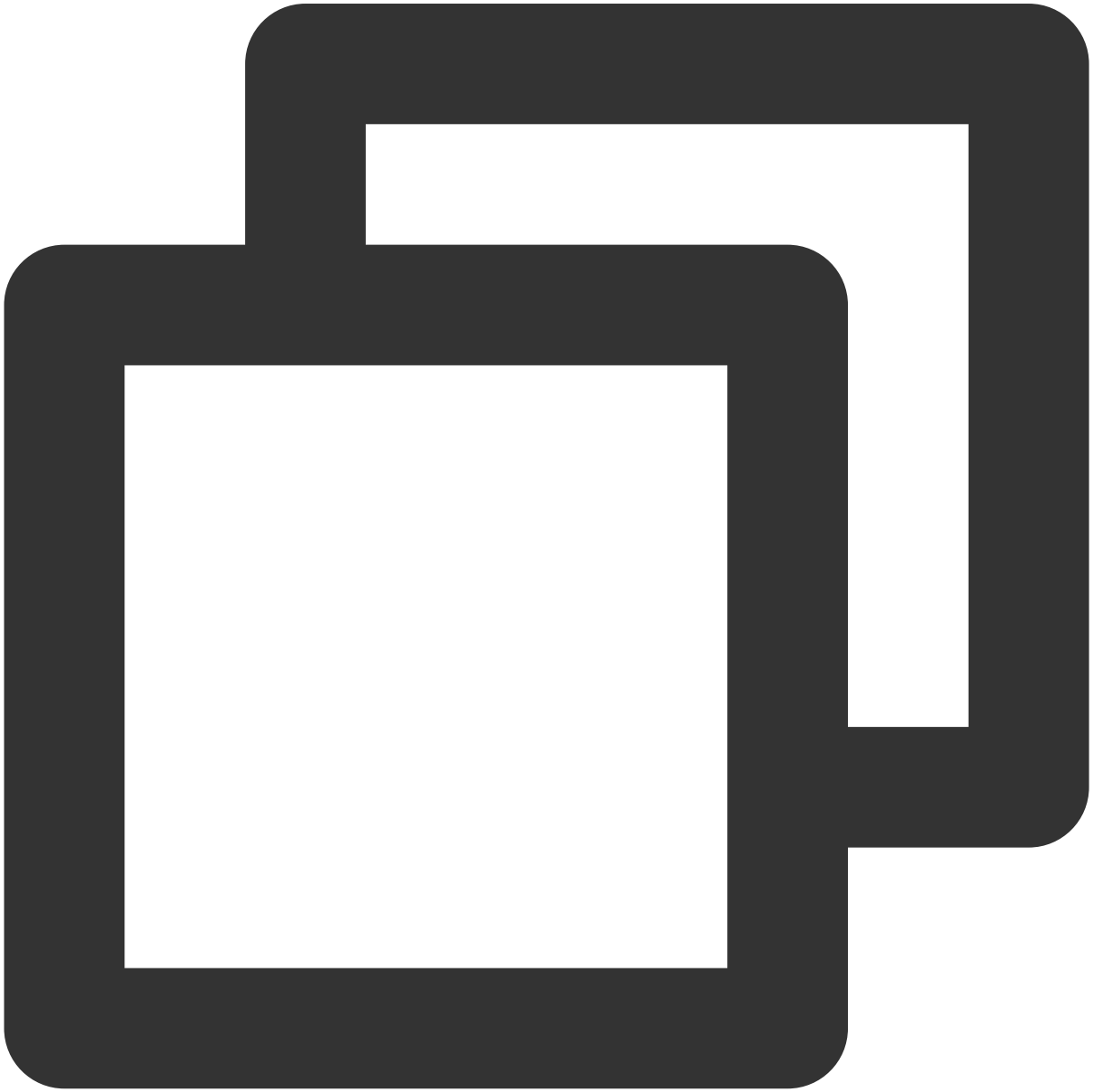
参数如下表所示：

参数	类型	含义

streamType	TUIVideoStreamType	流类型，详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义
view	const TUIVideoView&	渲染窗口的句柄

openLocalCamera

打开本地摄像头。



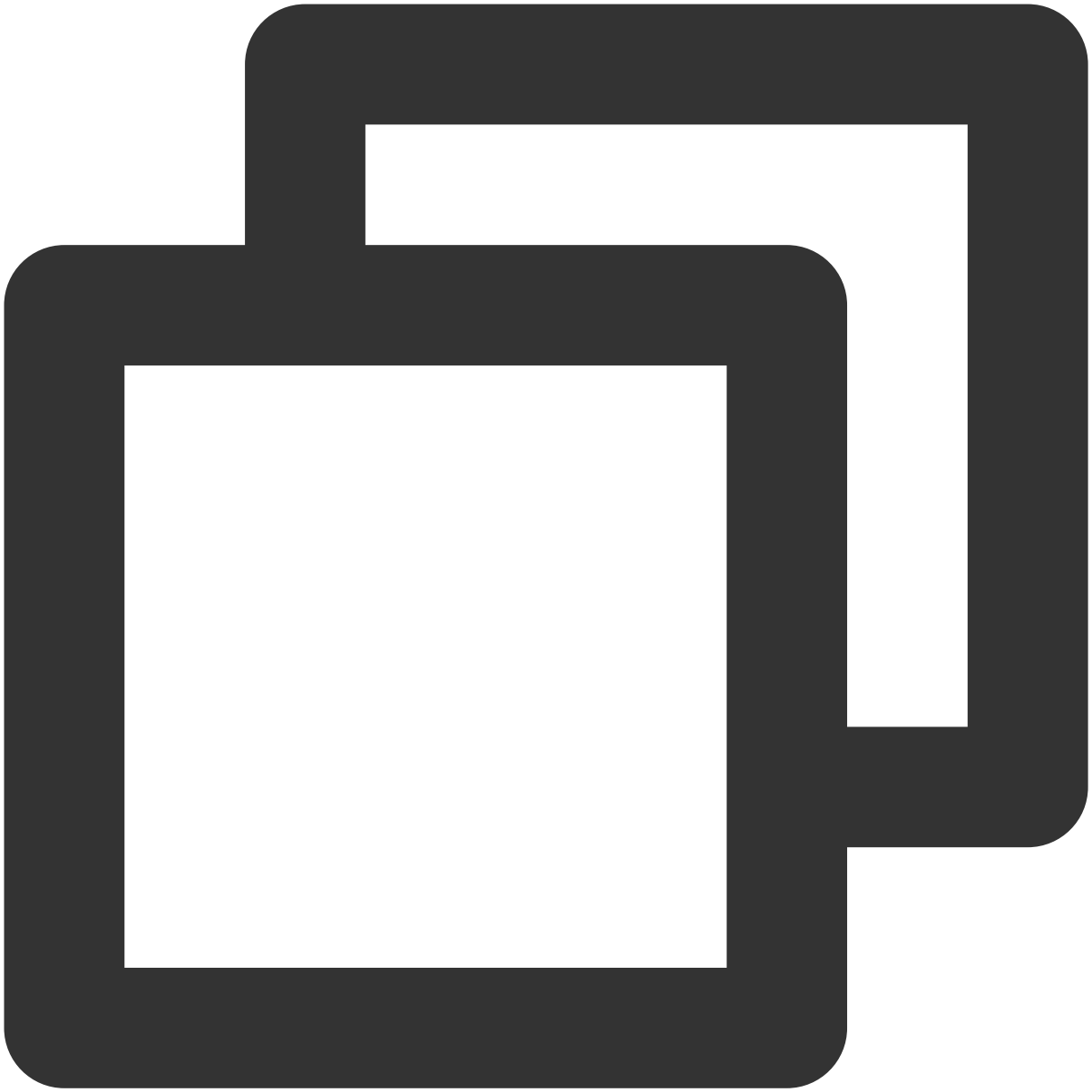
```
virtual void openLocalCamera(bool isFront, TUIVideoQuality quality, TUICallback* ca
```

参数如下表所示：

参数	类型	含义
isFront	bool	是否是前置摄像头（Windows下无效）
quality	TUIVideoQuality	视频质量，详细定义可以参考TUIRoomDefine.h 中 TUIVideoQuality 的定义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

closeLocalCamera

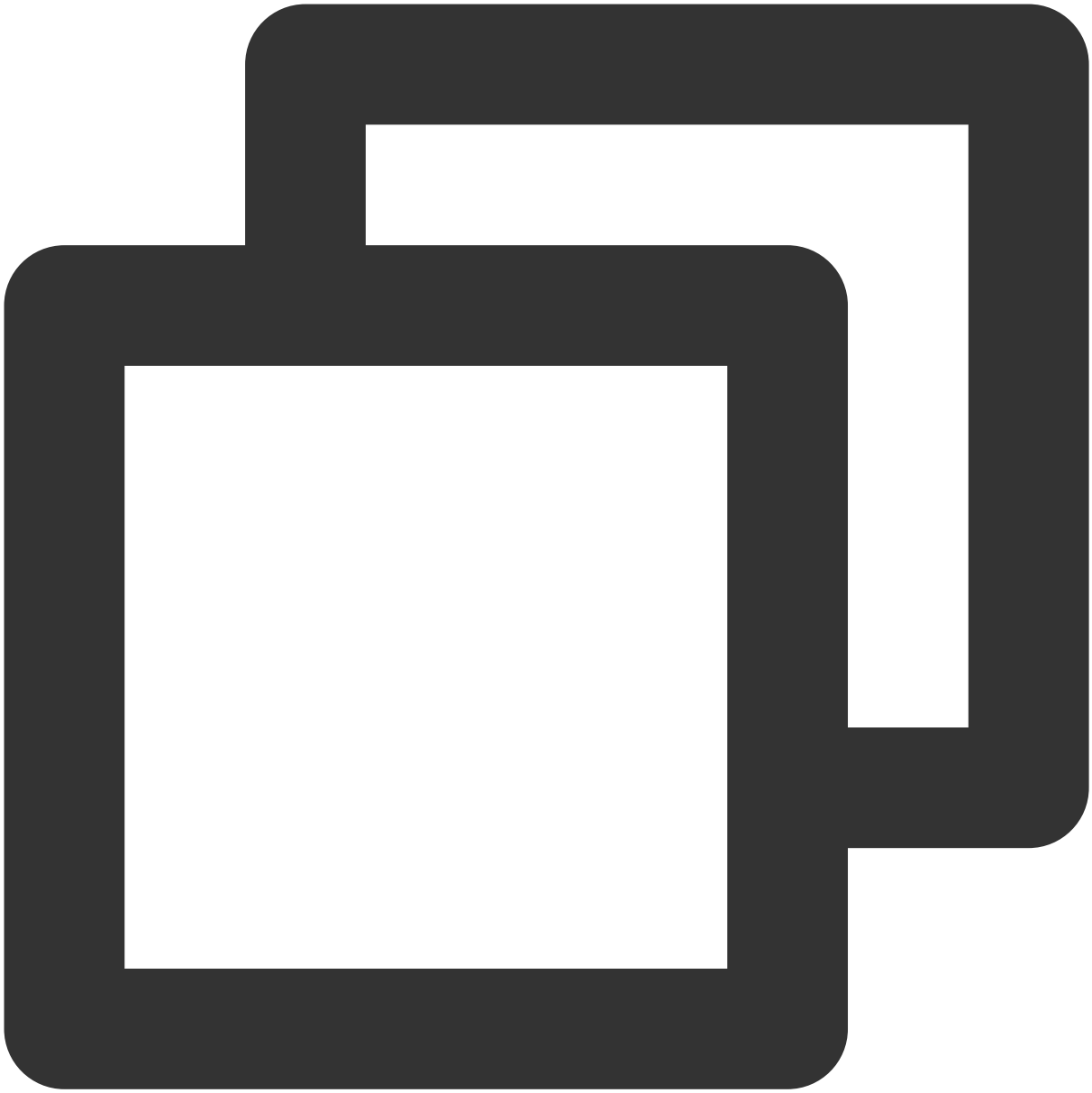
关闭本地摄像头。



```
virtual void closeLocalCamera() = 0;
```

updateVideoQuality

设置本地视频质量。



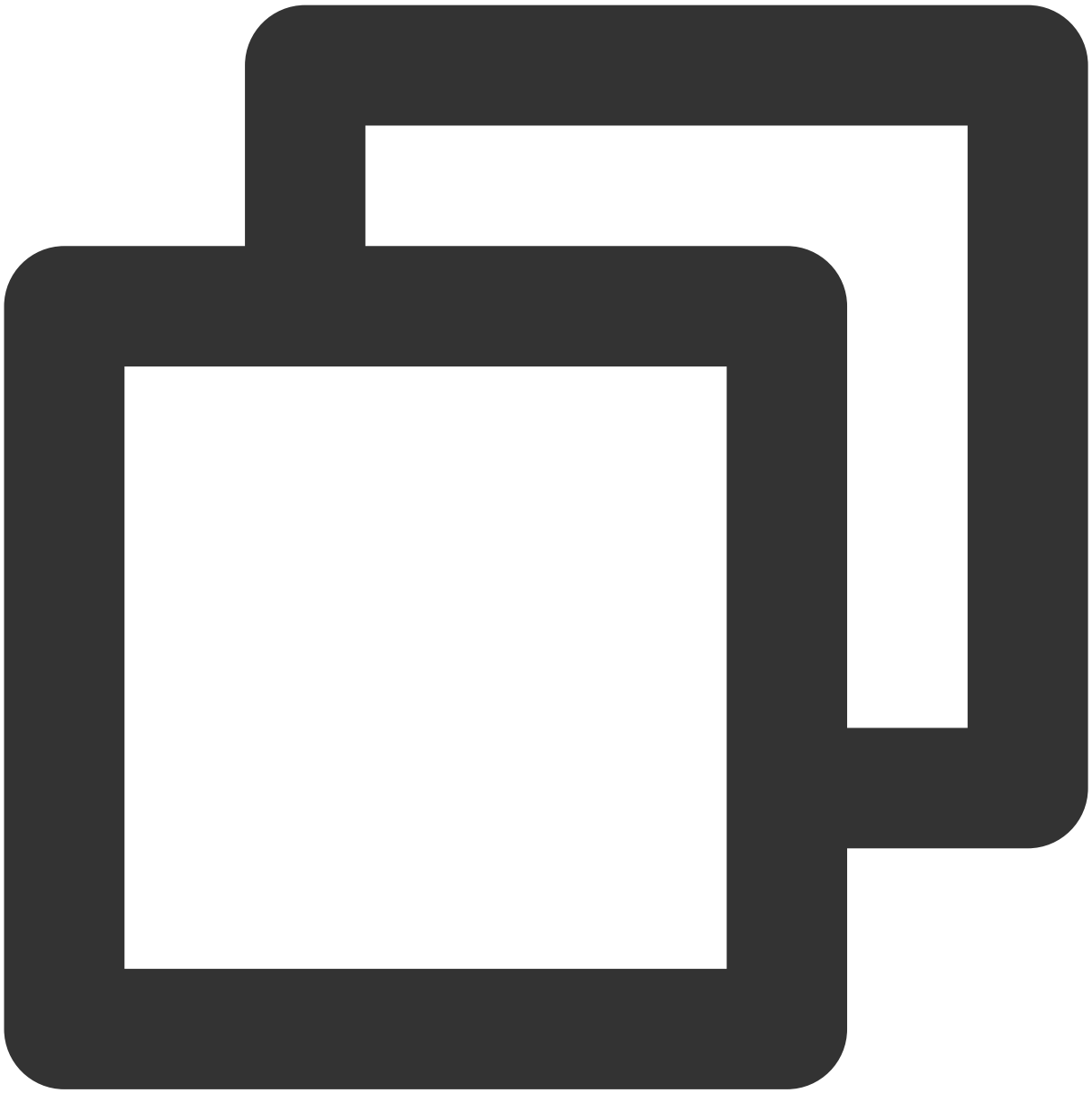
```
virtual void updateVideoQuality(TUIVideoQuality quality) = 0;
```

参数如下表所示：

参数	类型	含义
quality	TUIVideoQuality	视频质量，详细定义可以参考TUIRoomDefine.h 中 TUIVideoQuality 的定义

startScreenSharing

开始屏幕分享。



```
virtual void startScreenSharing(const TUISourceId& sourceId, TUICallback* callback)
```

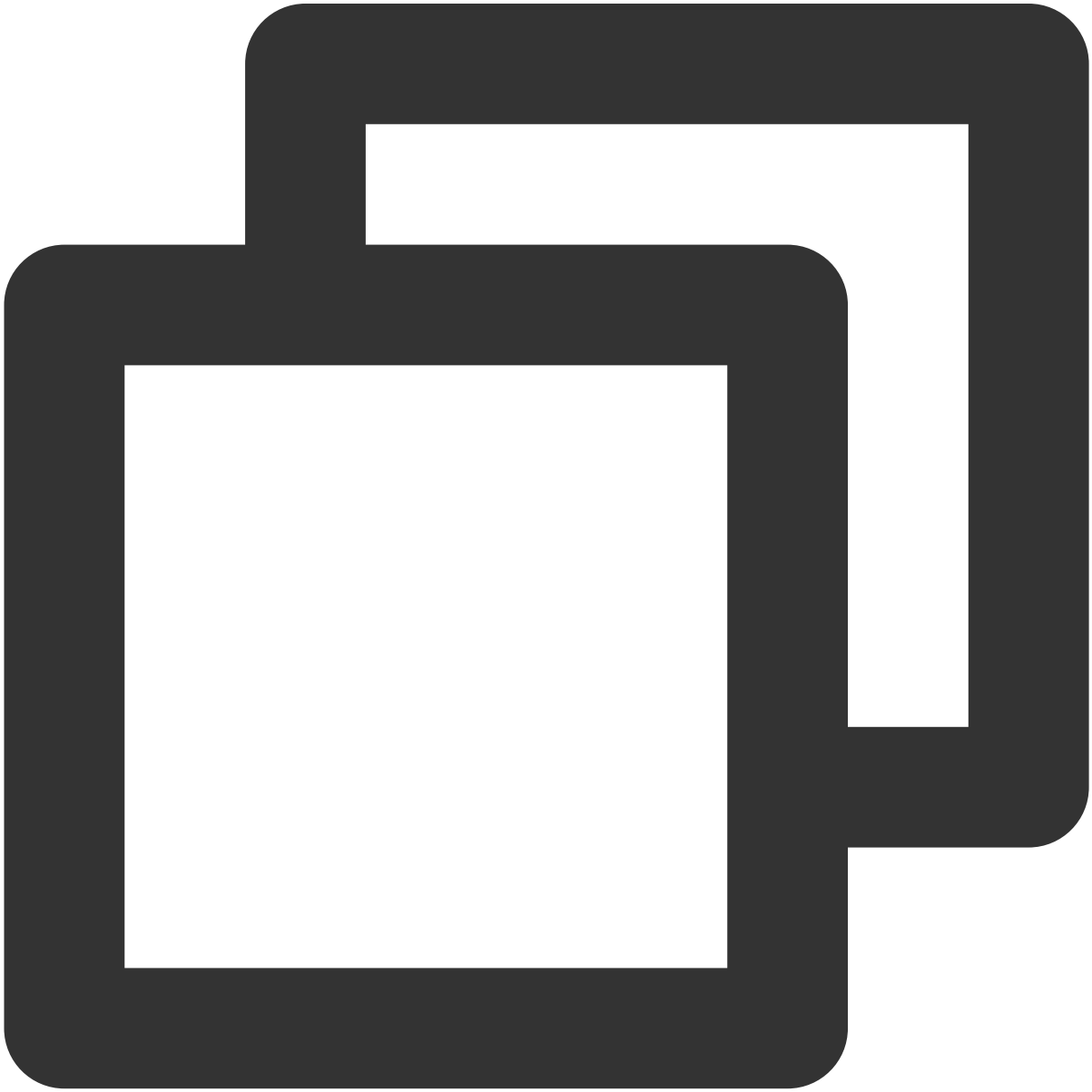
参数如下表所示：

参数	类型	含义
sourceId	const TUISourceId&	屏幕分享窗口或屏幕的句柄，可以调用 GetScreenSharingTargetList 获取
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考 TUIRoomDefine.h 中 TUICallback 的定义

stopScreenSharing

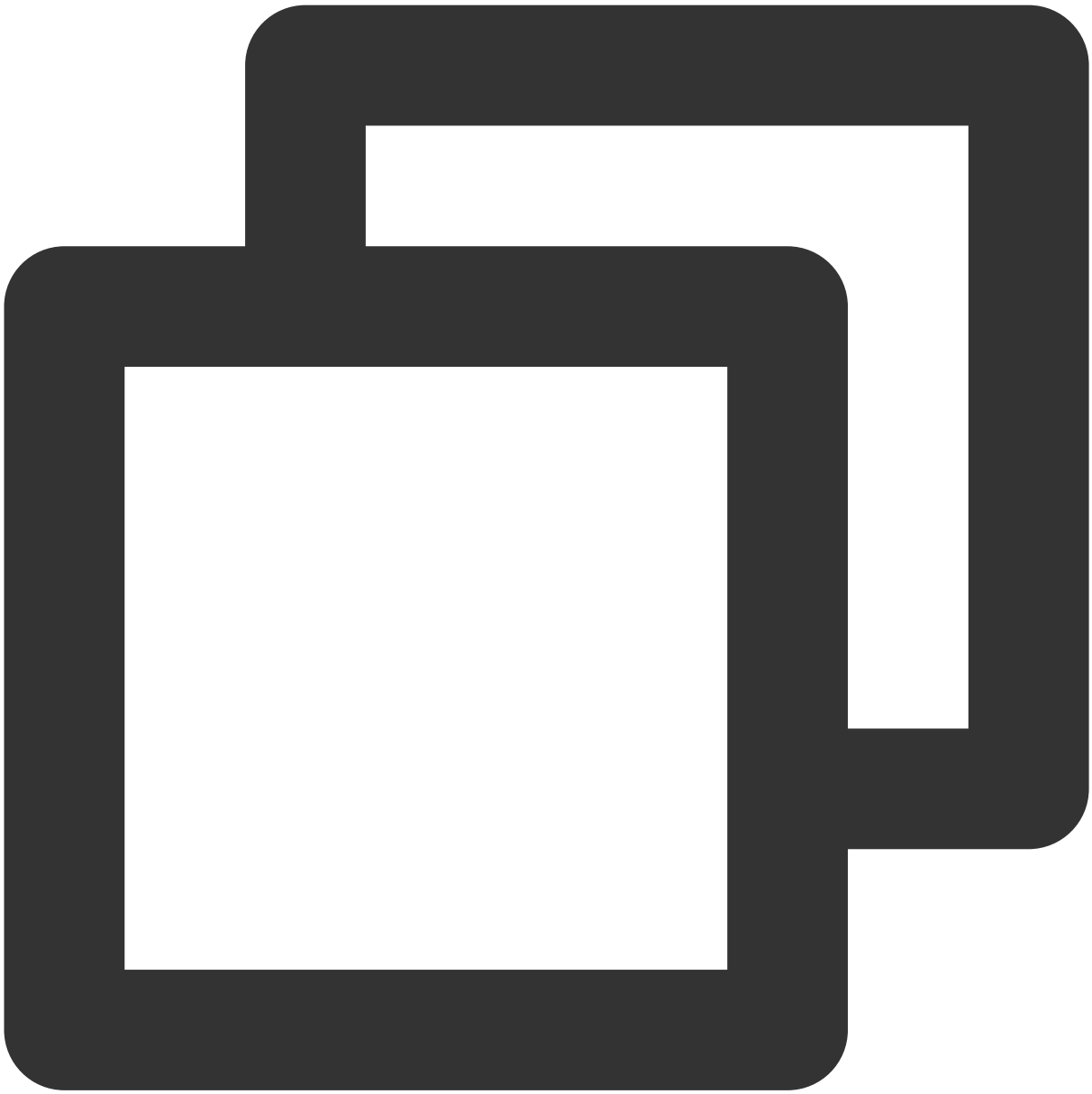
结束屏幕分享。



```
virtual void stopScreenSharing() = 0;
```

getScreenSharingTargetList

获取分享对象列表。



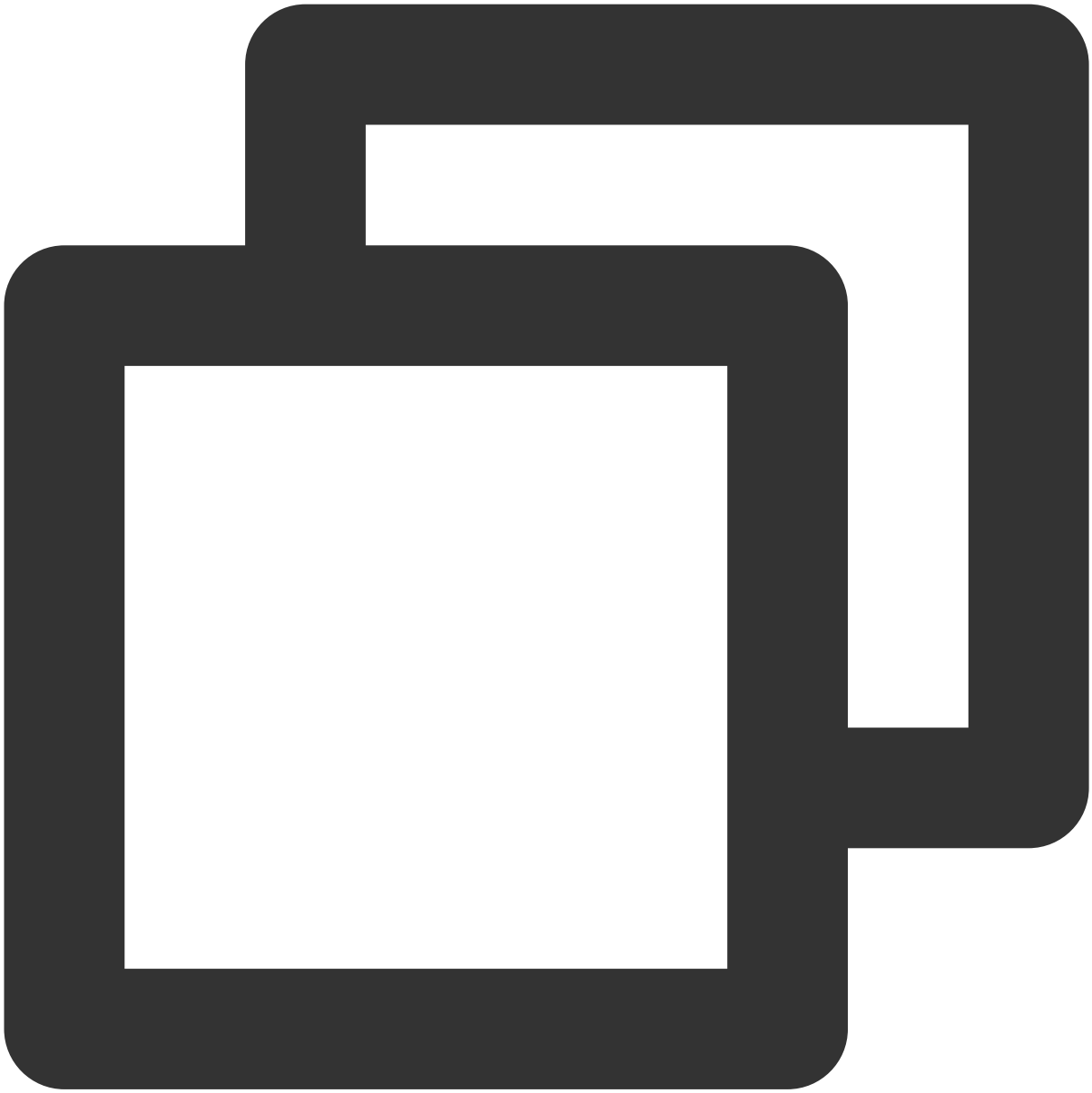
```
virtual void getScreenSharingTargetList(TUIListCallback<TUIShareTarget>* list) = 0;
```

参数如下表所示：

参数	类型	含义
list	TUIListCallback<TUIShareTarget>*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUIListCallback 与 TUIShareTarget 的定义

selectScreenSharingTarget

选择屏幕分享对象。



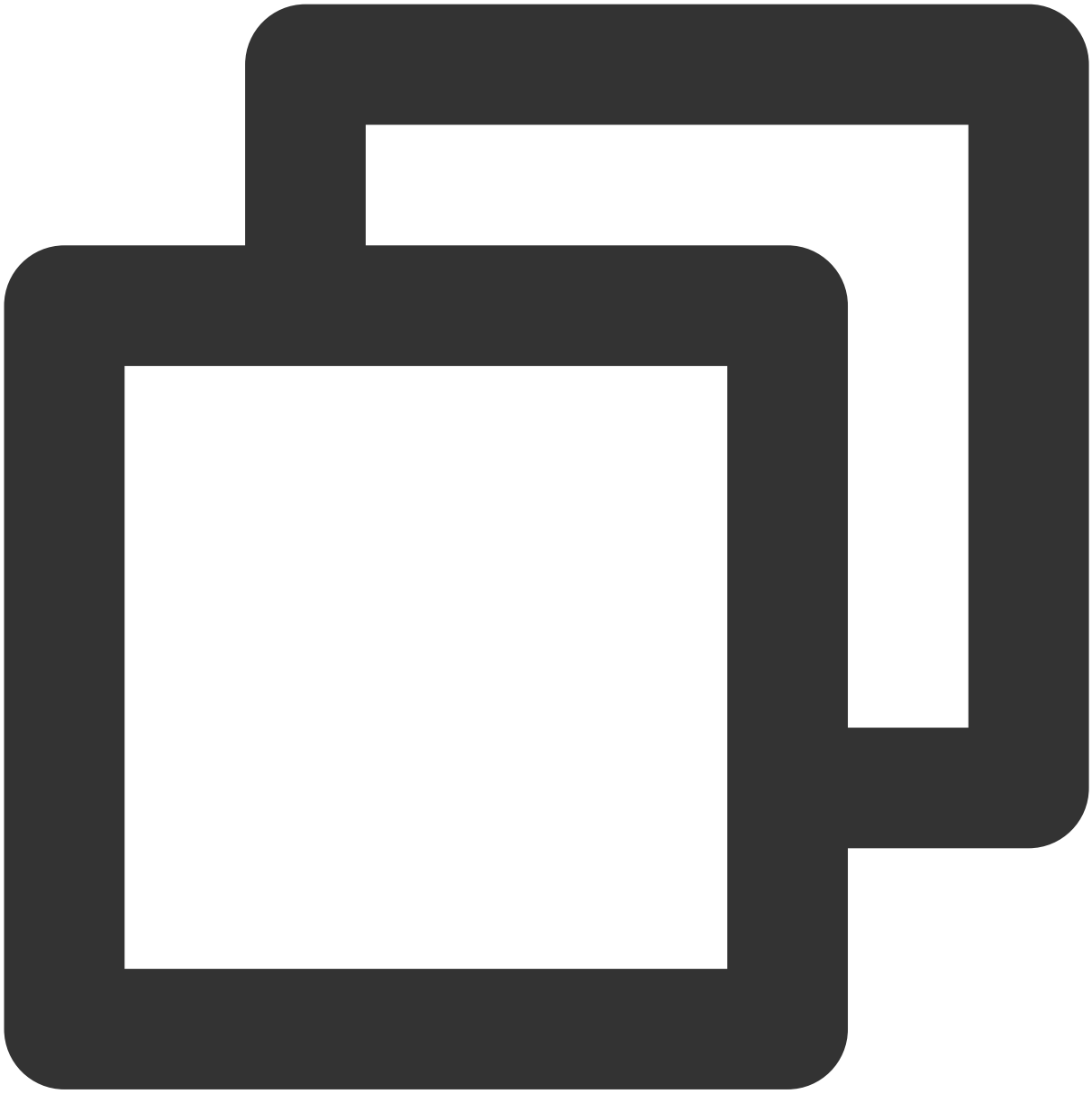
```
virtual void selectScreenSharingTarget(const TUISourceId& sourceId) = 0;
```

参数如下表所示：

参数	类型	含义
sourceId	const TUISourceId&	屏幕分享窗口或屏幕的句柄，可以调用 GetScreenSharingTargetList 获取

startPushLocalVideo

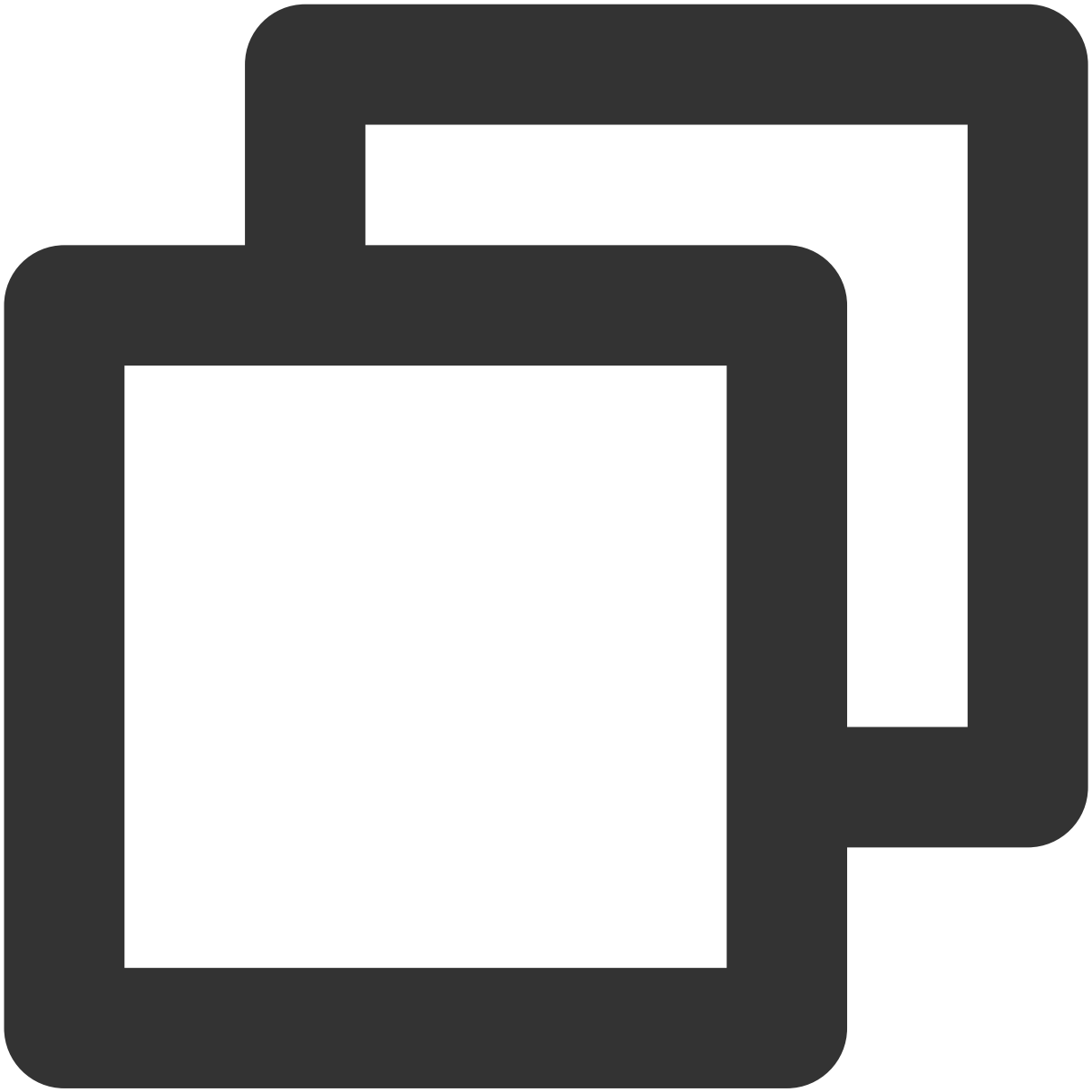
开始推送本地视频。



```
virtual void startPushLocalVideo() = 0;
```

stopPushLocalVideo

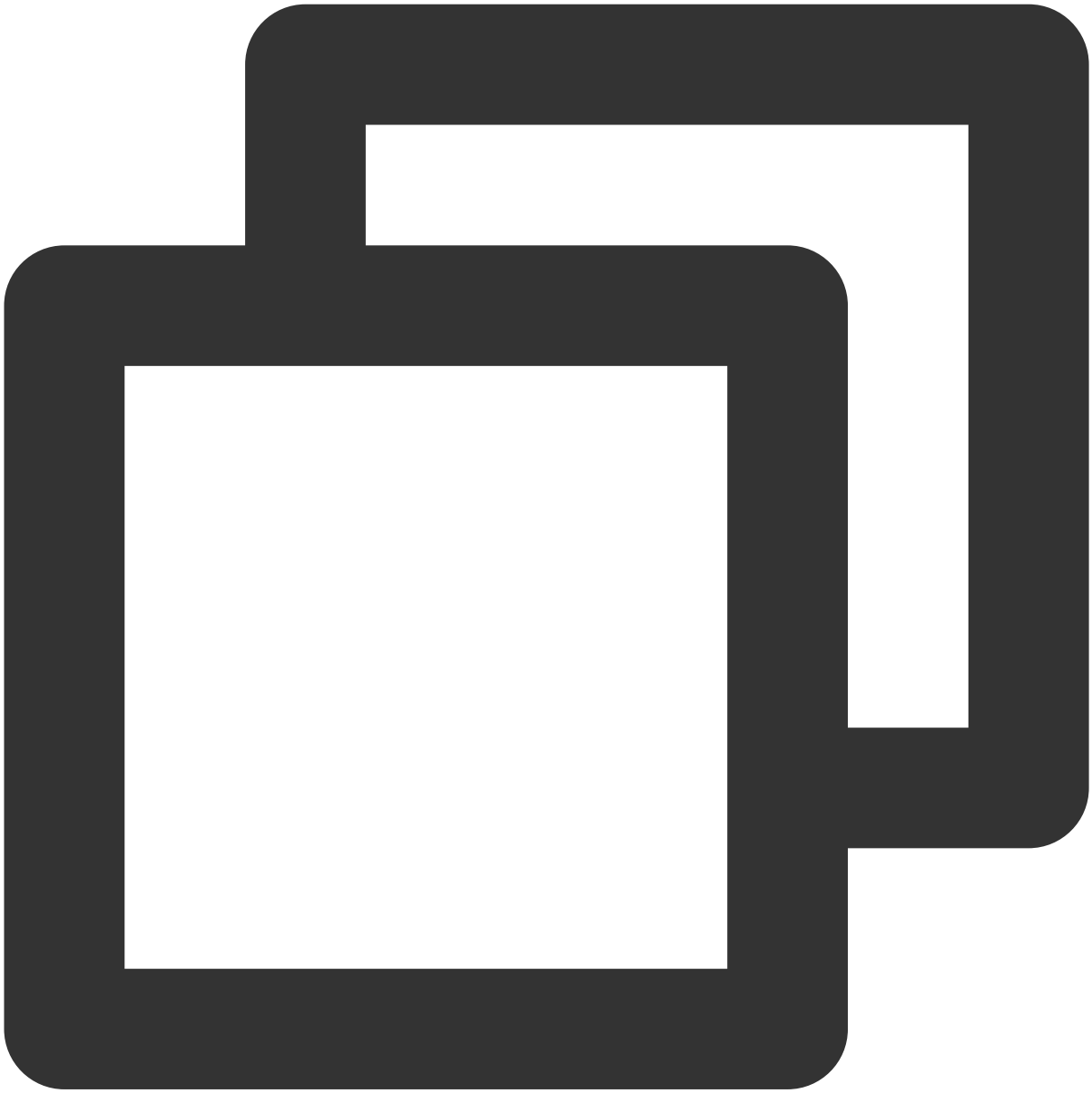
停止推送本地视频。



```
virtual void stopPushLocalVideo() = 0;
```

openLocalMicrophone

打开本地麦克风。



```
virtual void openLocalMicrophone(TUIAudioQuality quality, TUICallback* callback) =
```

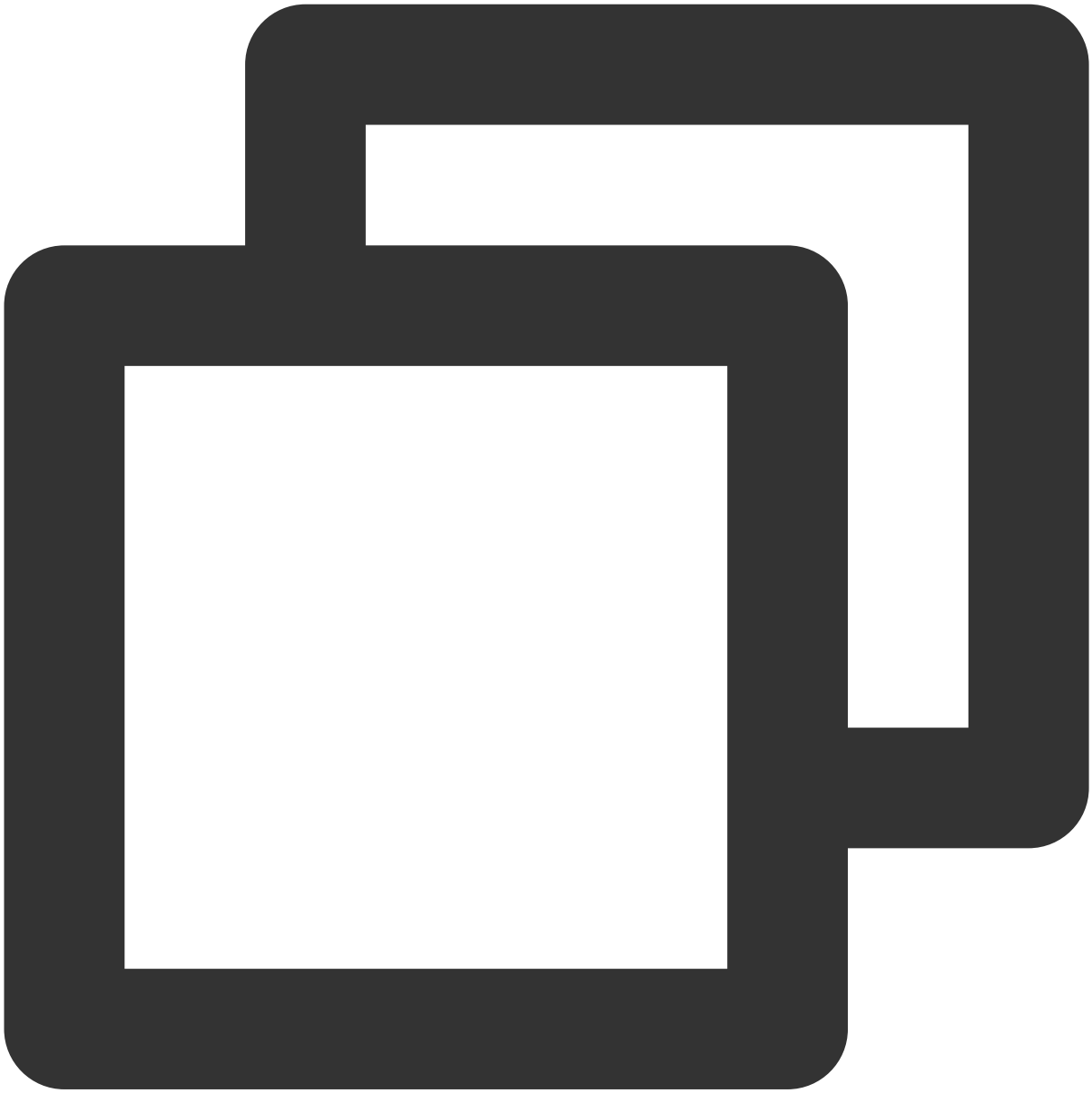
参数如下表所示：

参数	类型	含义
quality	TUIAudioQuality	音频质量。详细定义可以参考 TUIRoomDefine.h 中 TUIAudioQuality 的定义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中

		TUICallback 的定义

closeLocalMicrophone

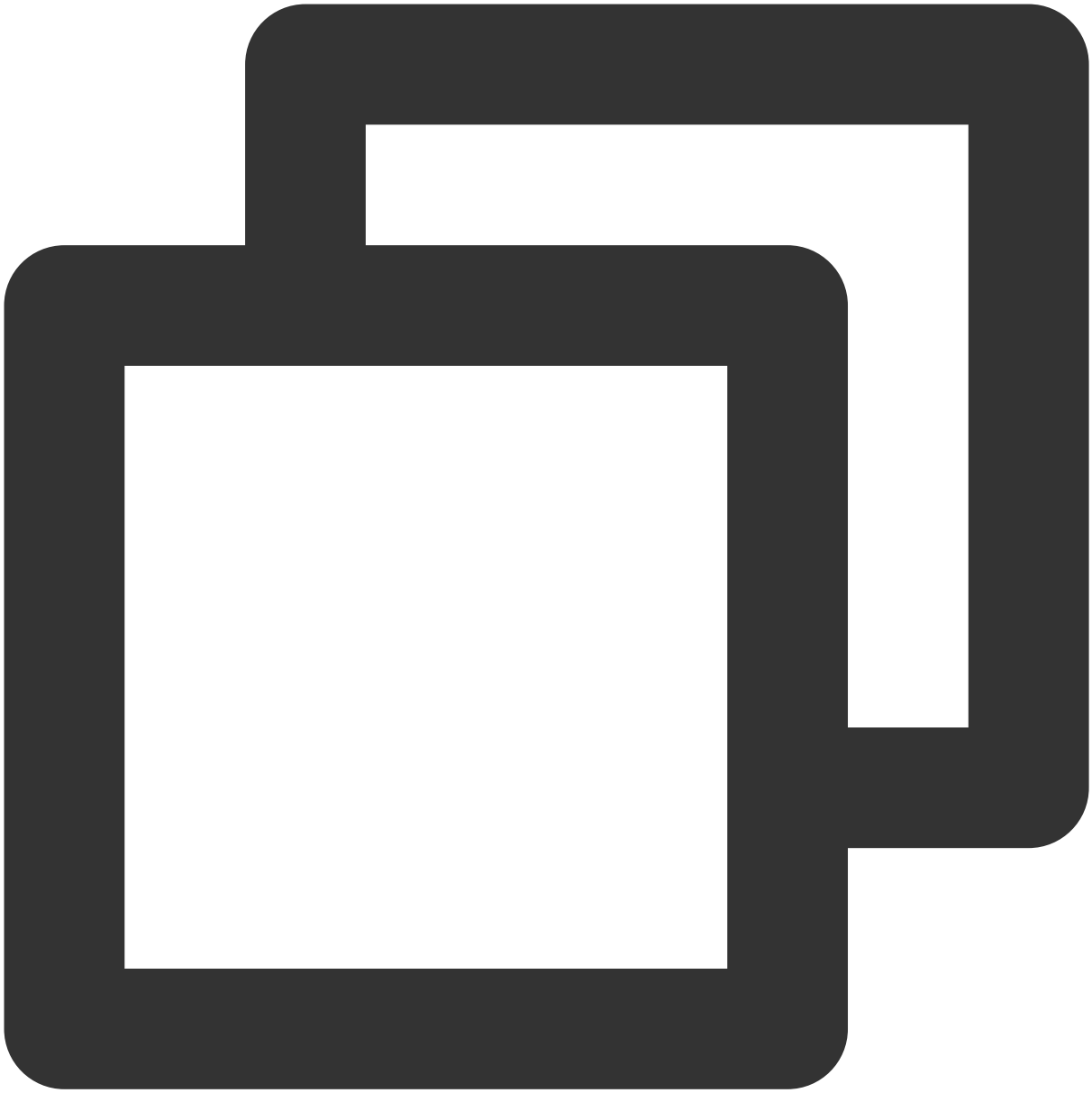
关闭本地麦克风。



```
virtual void closeLocalMicrophone() = 0;
```

updateAudioQuality

设置本地音频质量。



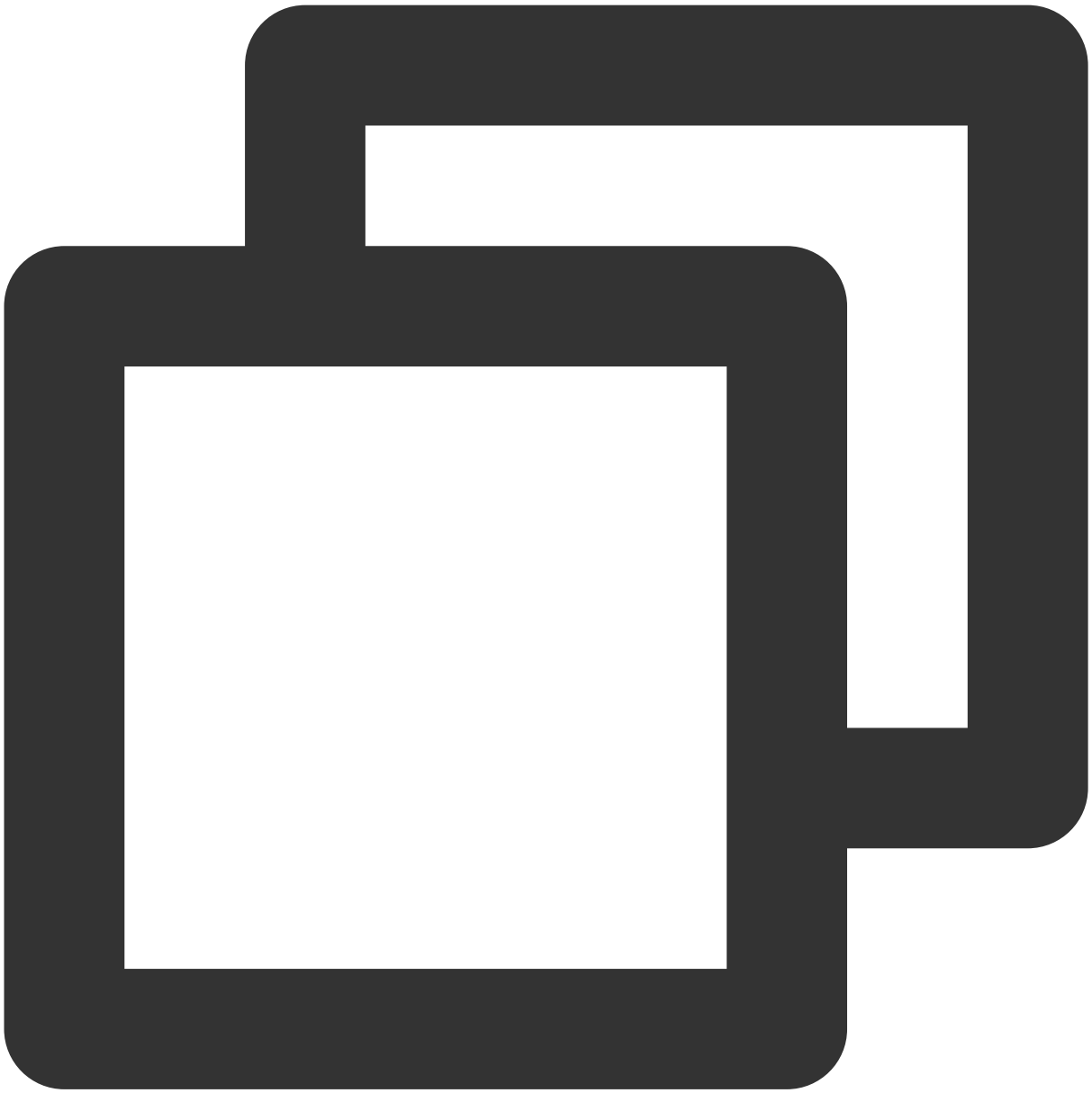
```
virtual void updateAudioQuality(TUIAudioQuality quality) = 0;
```

参数如下表所示：

参数	类型	含义
quality	TUIAudioQuality	音频质量。详细定义可以参考 TUIRoomDefine.h 中 TUIAudioQuality 的定义

startPushLocalAudio

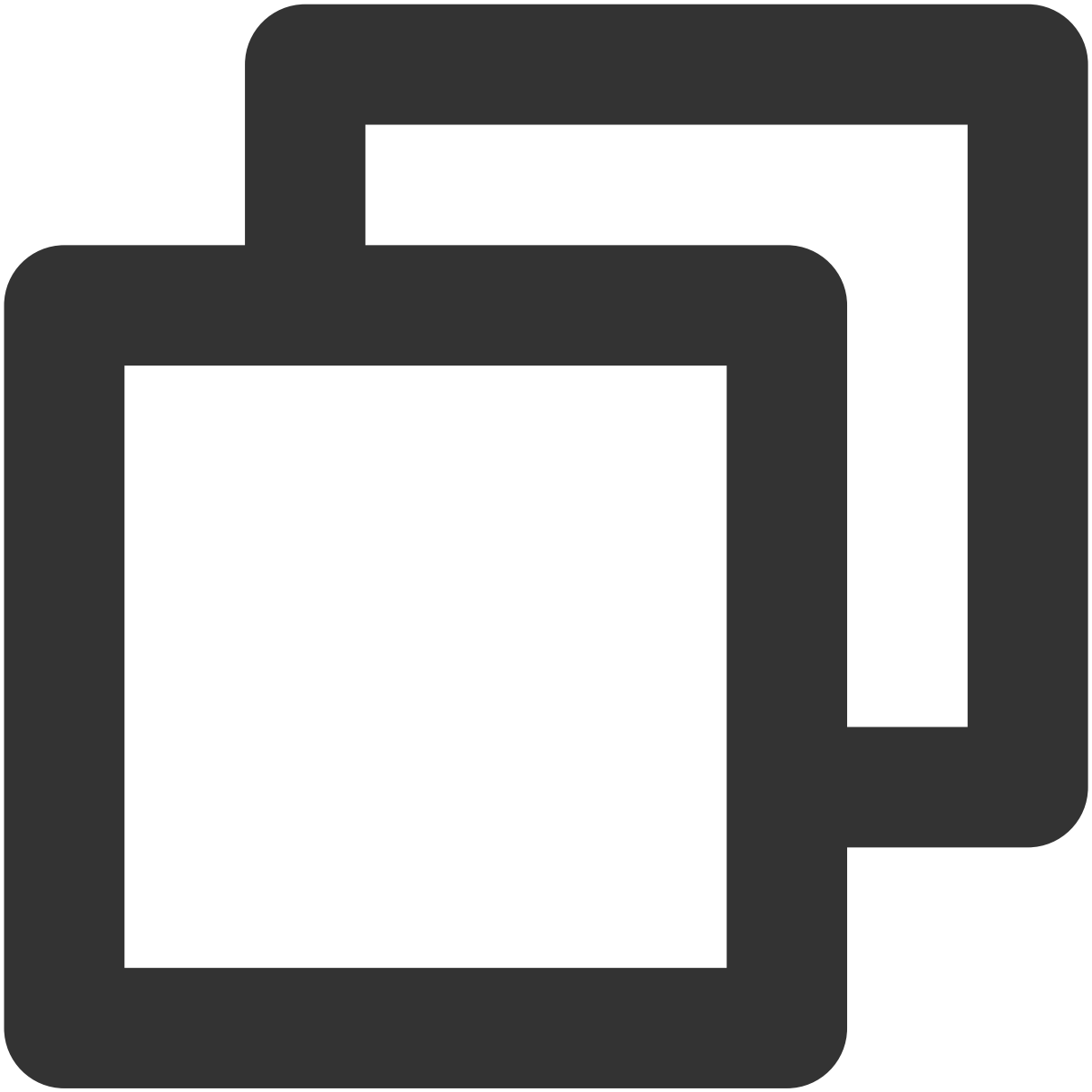
开始推送本地音频。



```
virtual void startPushLocalAudio() = 0;
```

stopPushLocalAudio

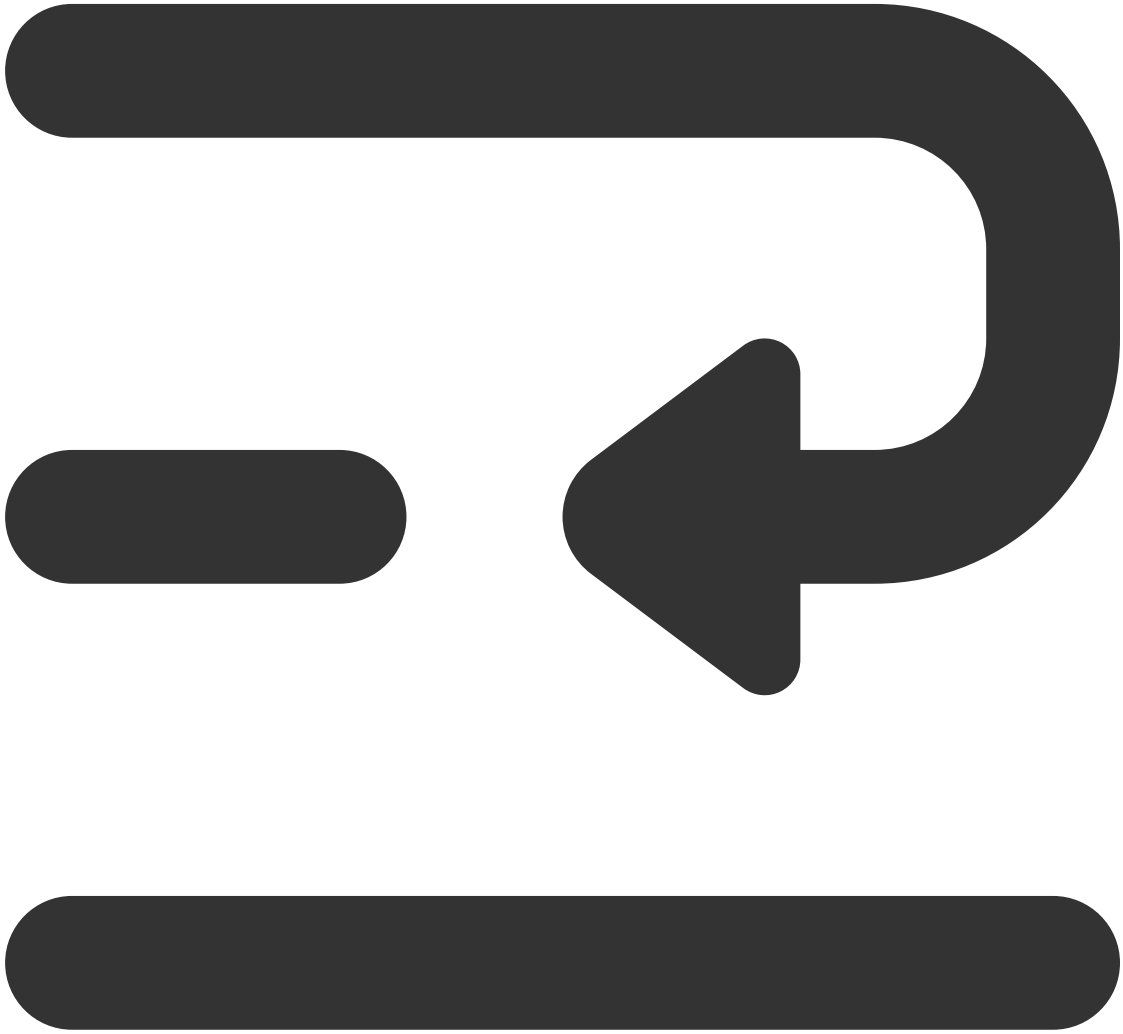
停止推送本地音频。



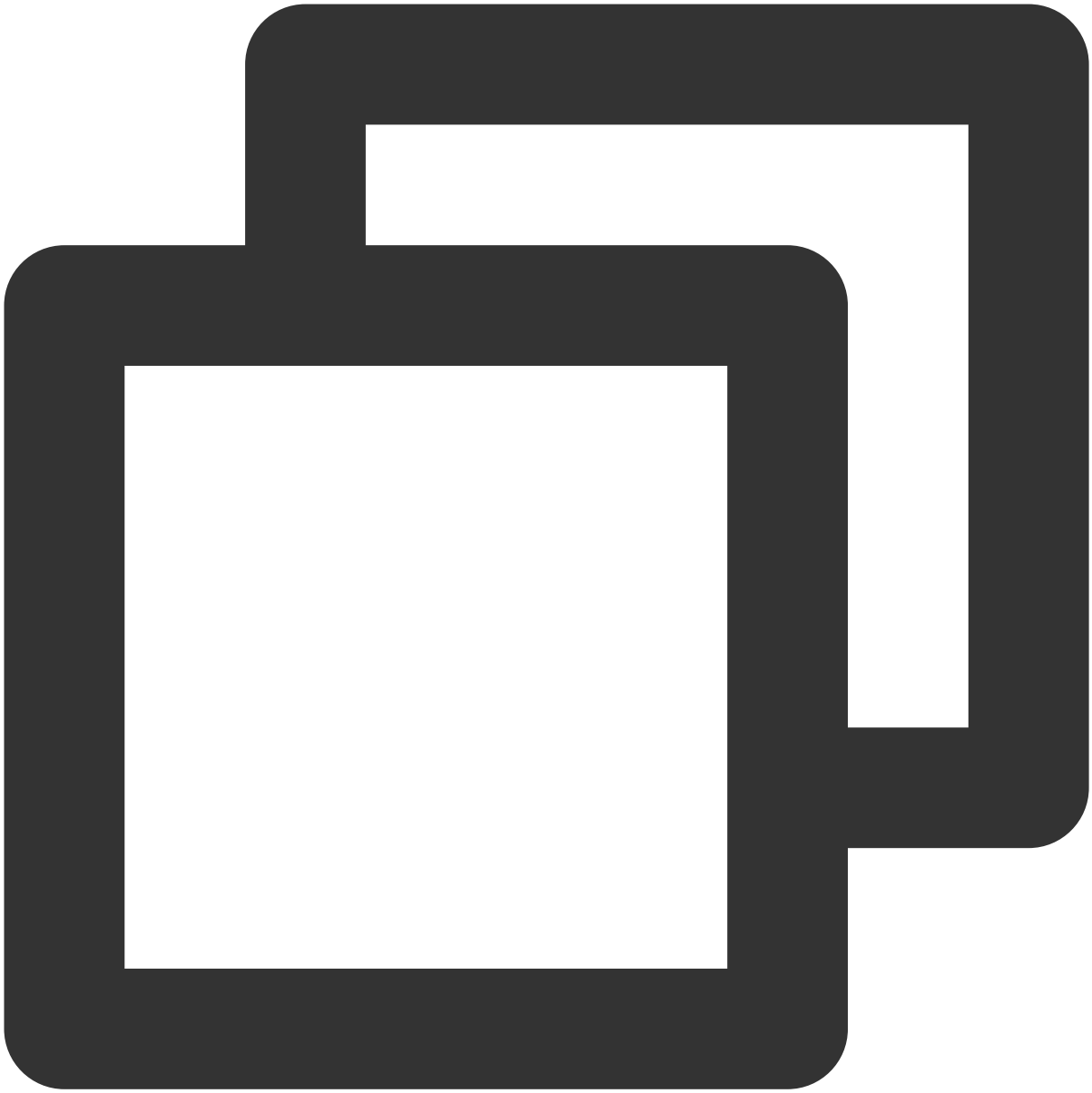
```
virtual void stopPushLocalAudio() = 0;
```

setRemoteVideoView

设置远端用户视频渲染的控件。



版权所有：腾讯云计算（北京）有限责任公司



```
virtual void setRemoteVideoView(const char* userId, TUIVideoStreamType streamType,
```

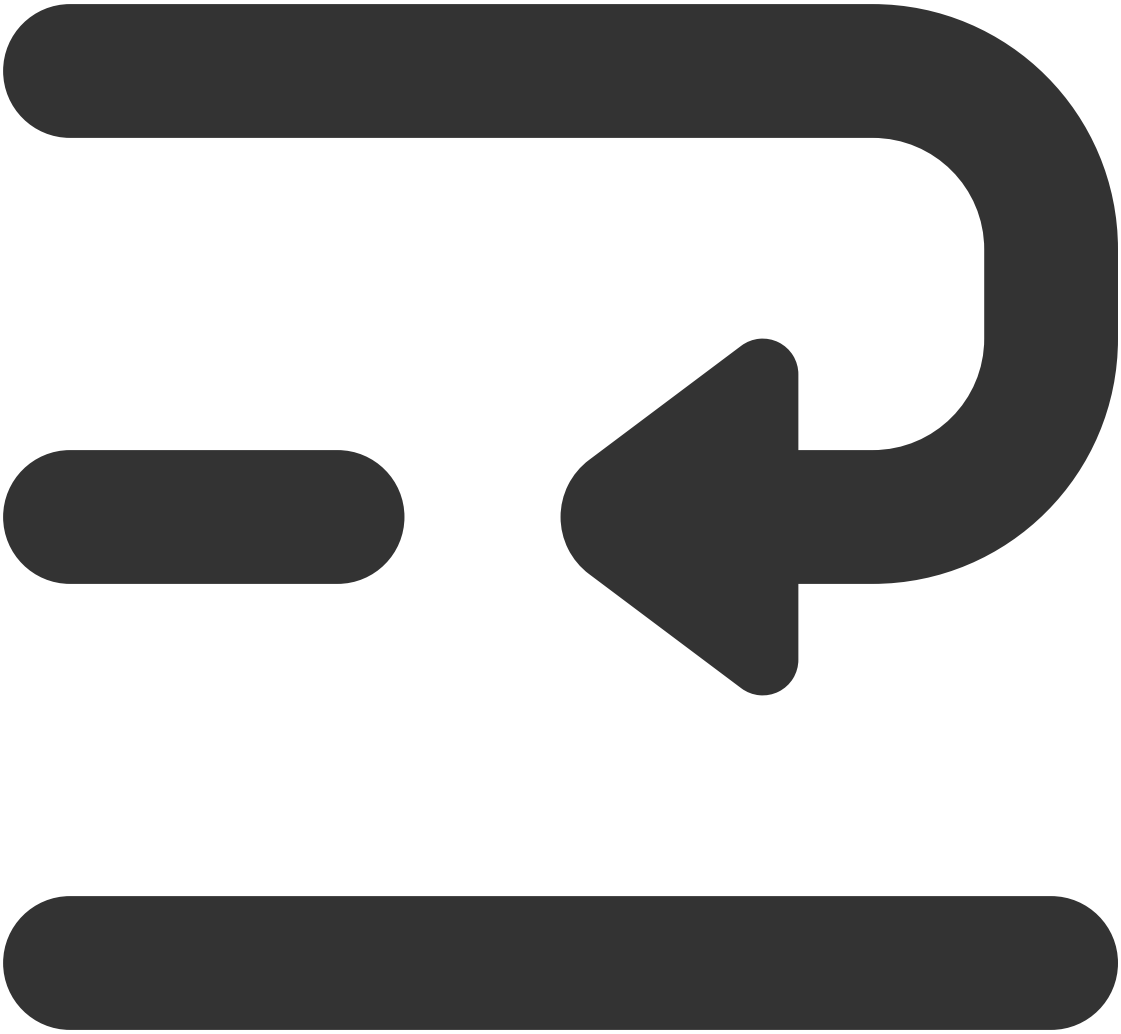
参数如下表所示：

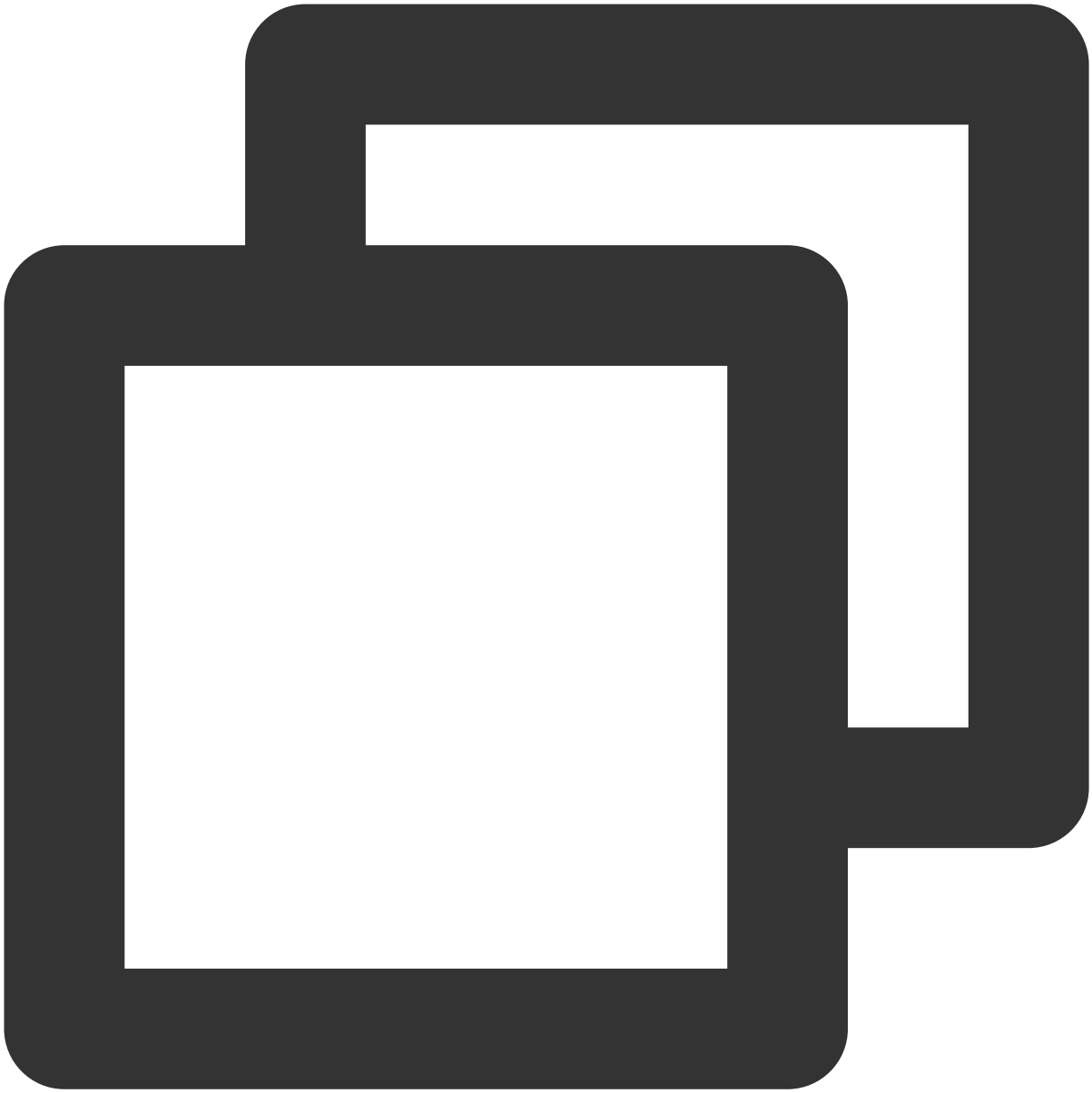
参数	类型	含义
userId	const char*	远端用户 ID
streamType	TUIVideoStreamType	流类型，详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义

view	const TUIVideoView&	渲染窗口的句柄

startPlayRemoteVideo

开始播放远端用户视频。





```
virtual void startPlayRemoteVideo(const char* userId, TUIVideoStreamType streamType
```

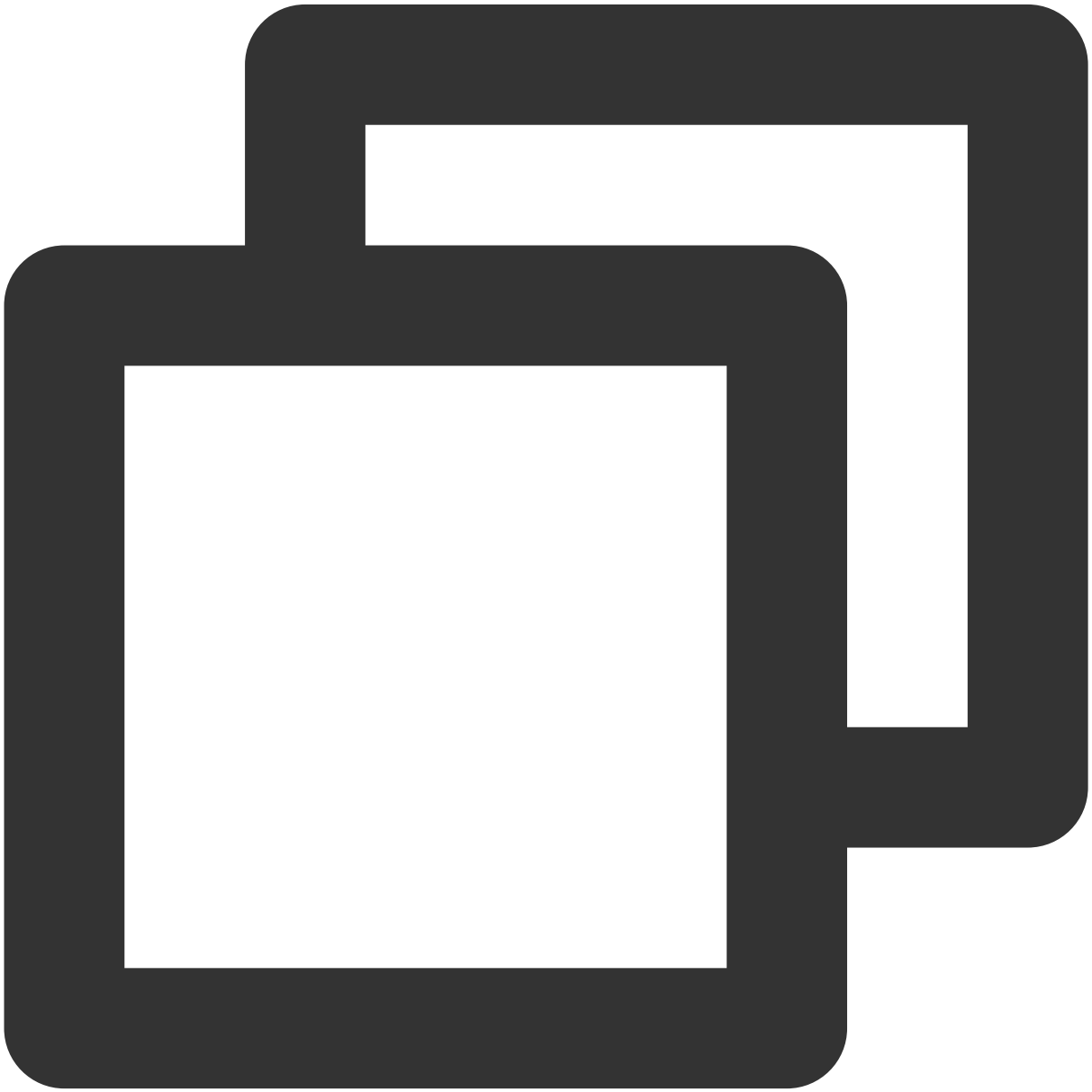
参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
streamType	TUIVideoStreamType	视频流的类型。详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义

callback	TUIPlayCallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUIPlayCallback 的定义

stopPlayRemoteVideo

停止播放远端用户视频。



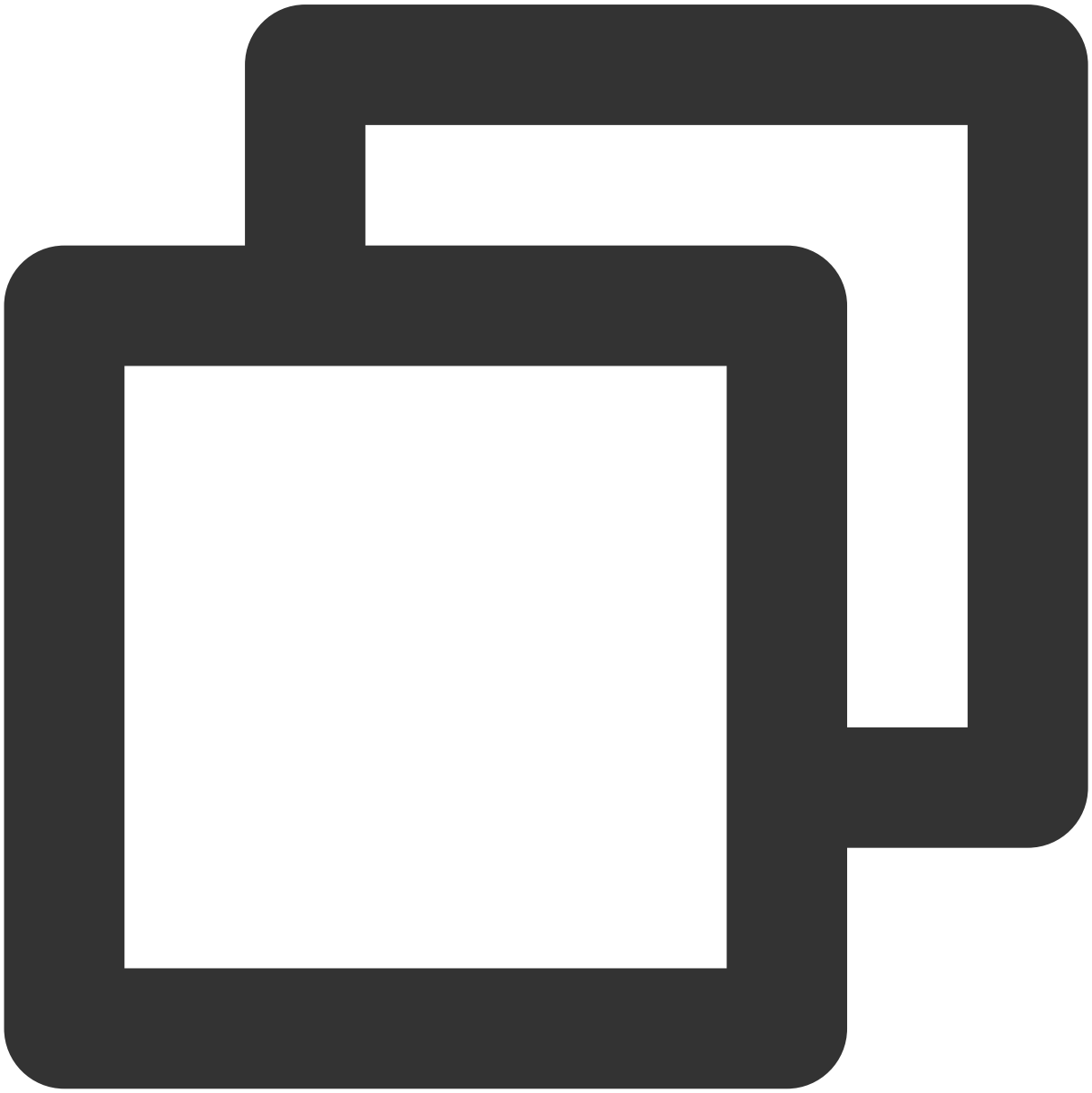
```
virtual void stopPlayRemoteVideo(const char* userId, TUIVideoStreamType streamType)
```

参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
streamType	TUIVideoStreamType	视频流的类型。详细定义可以参考 TUIRoomDefine.h 中 TUIVideoStreamType 的定义

muteRemoteAudioStream

停止订阅远端用户音频。



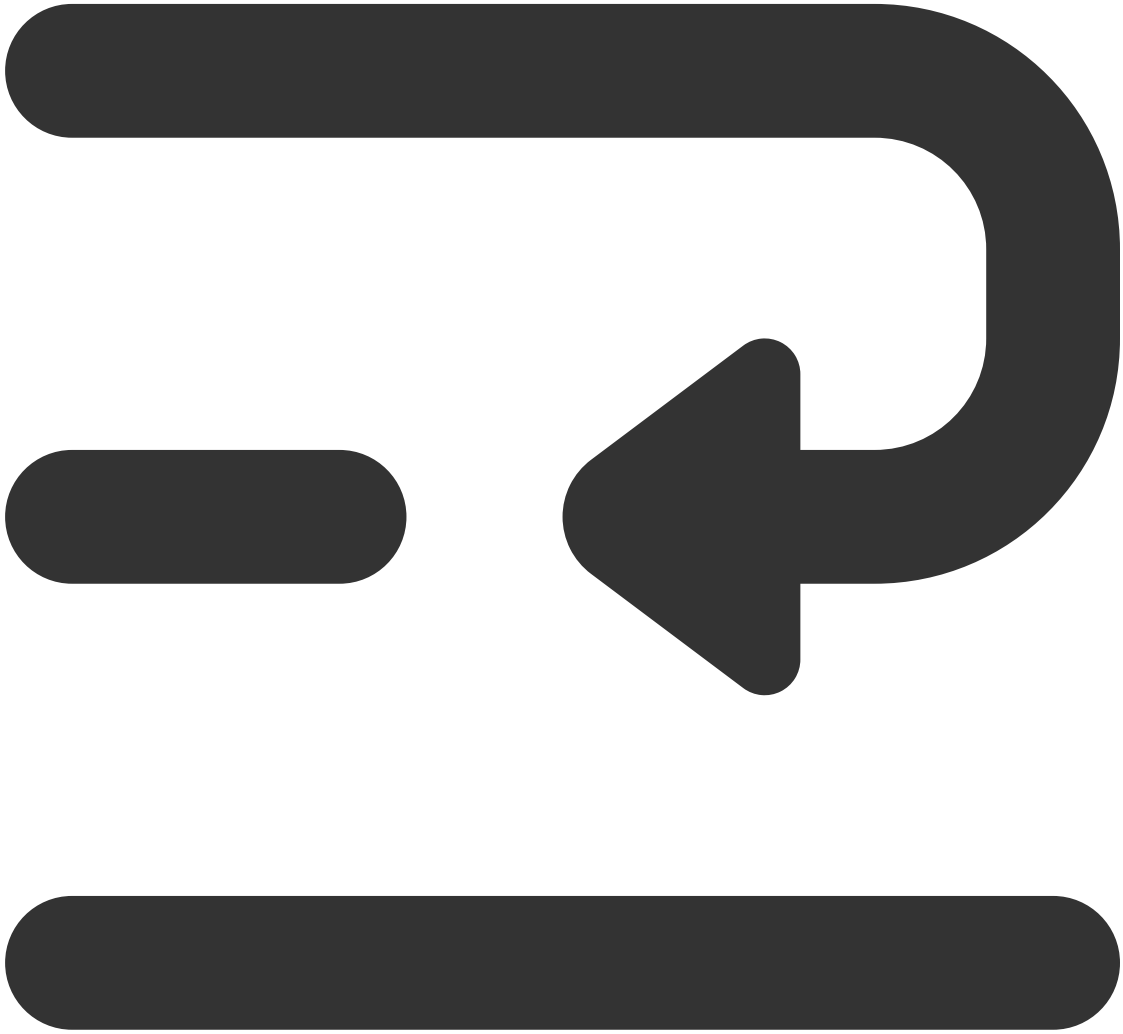
```
virtual void muteRemoteAudioStream(const char* userId, bool isMute) = 0;
```

参数如下表所示：

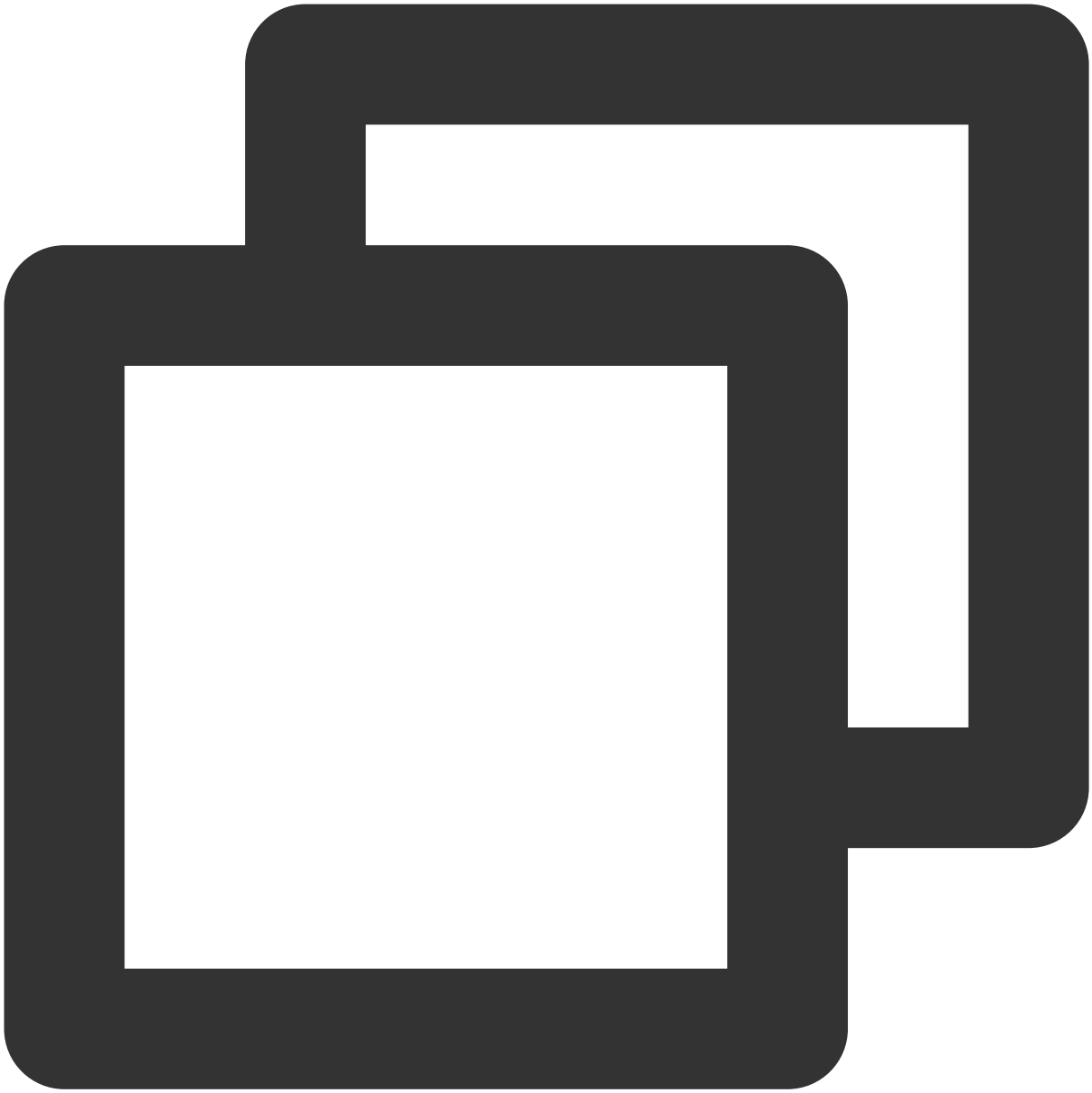
参数	类型	含义
userId	const char*	用户 ID
isMute	bool	是否订阅远端用户音频流

getUserList

获取房间内的成员列表。



版权所有：腾讯云计算（北京）有限责任公司



```
virtual void getUserList(uint64_t nextSequence, TUIValueCallback<TUIUserListResult>
```

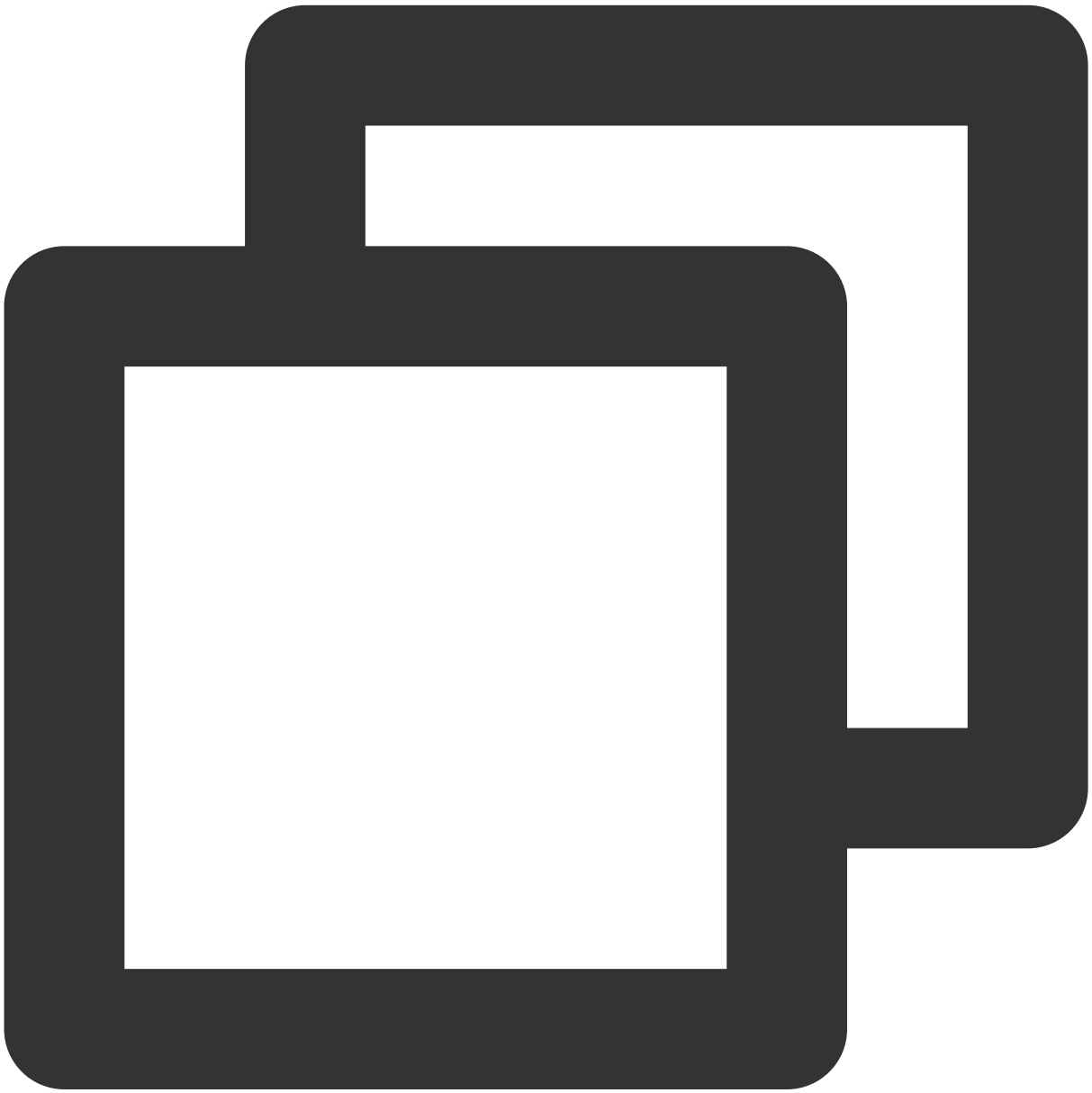
参数如下表所示：

参数	类型	含义
nextSequence	uint64_t	分页拉取标志，第一次拉取填0，回调成功如果 callback 返回TUIUserListResult中 nextSequence 不为零，需要分页，传入再次拉取，直至为0

callback	TUIValueCallback<TUIUserListResult>*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考 TUIRoomDefine.h 中 TUICallback 的定义

getUserInfo

获取成员信息。



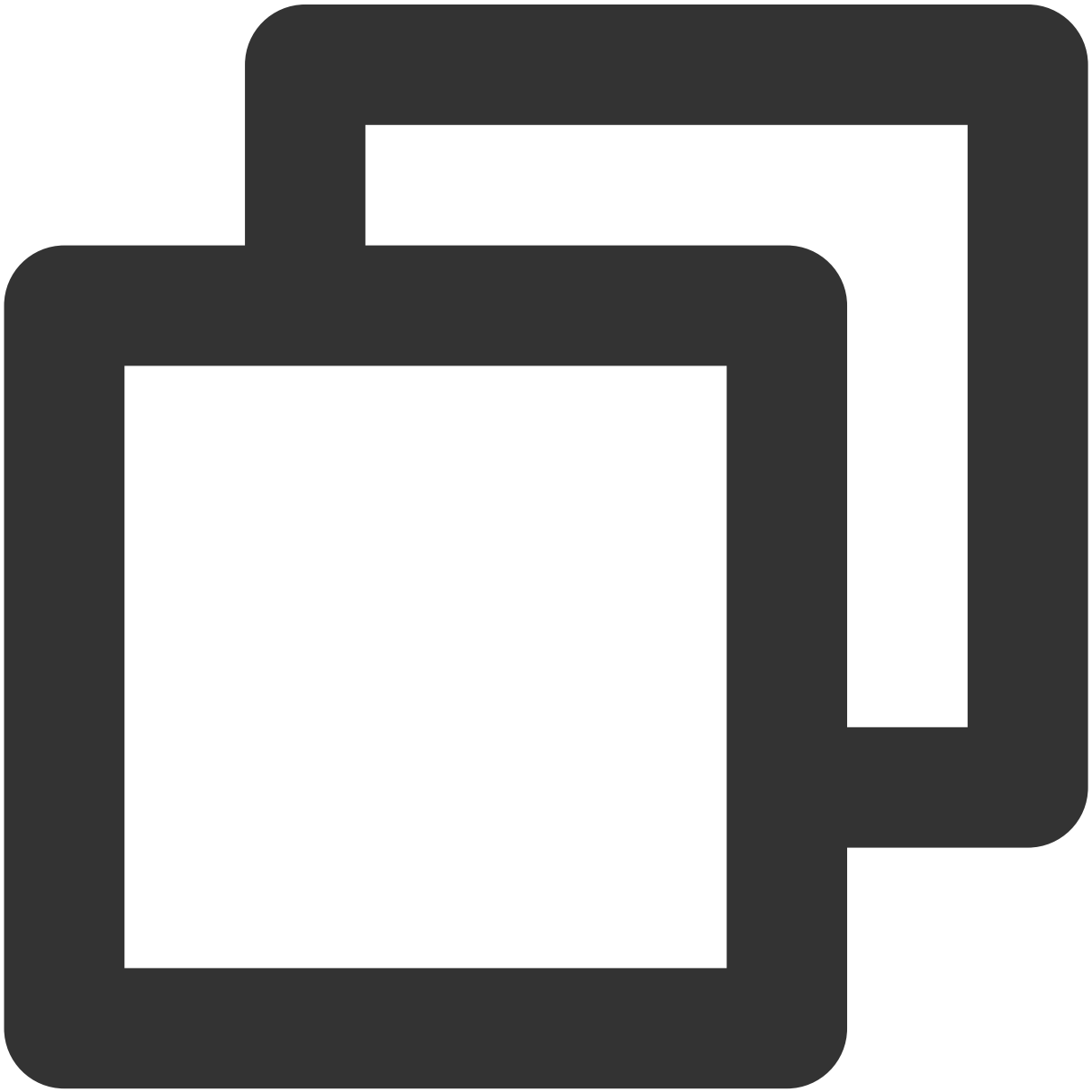
```
virtual void getUserInfo(const char* userId, TUIValueCallback<TUIUserInfo>* callbac
```

参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
callback	TUIValueCallback<TUIUserInfo>*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIValueCallback 和 TUIUserInfo 的定义

changeUserRole

修改用户角色。



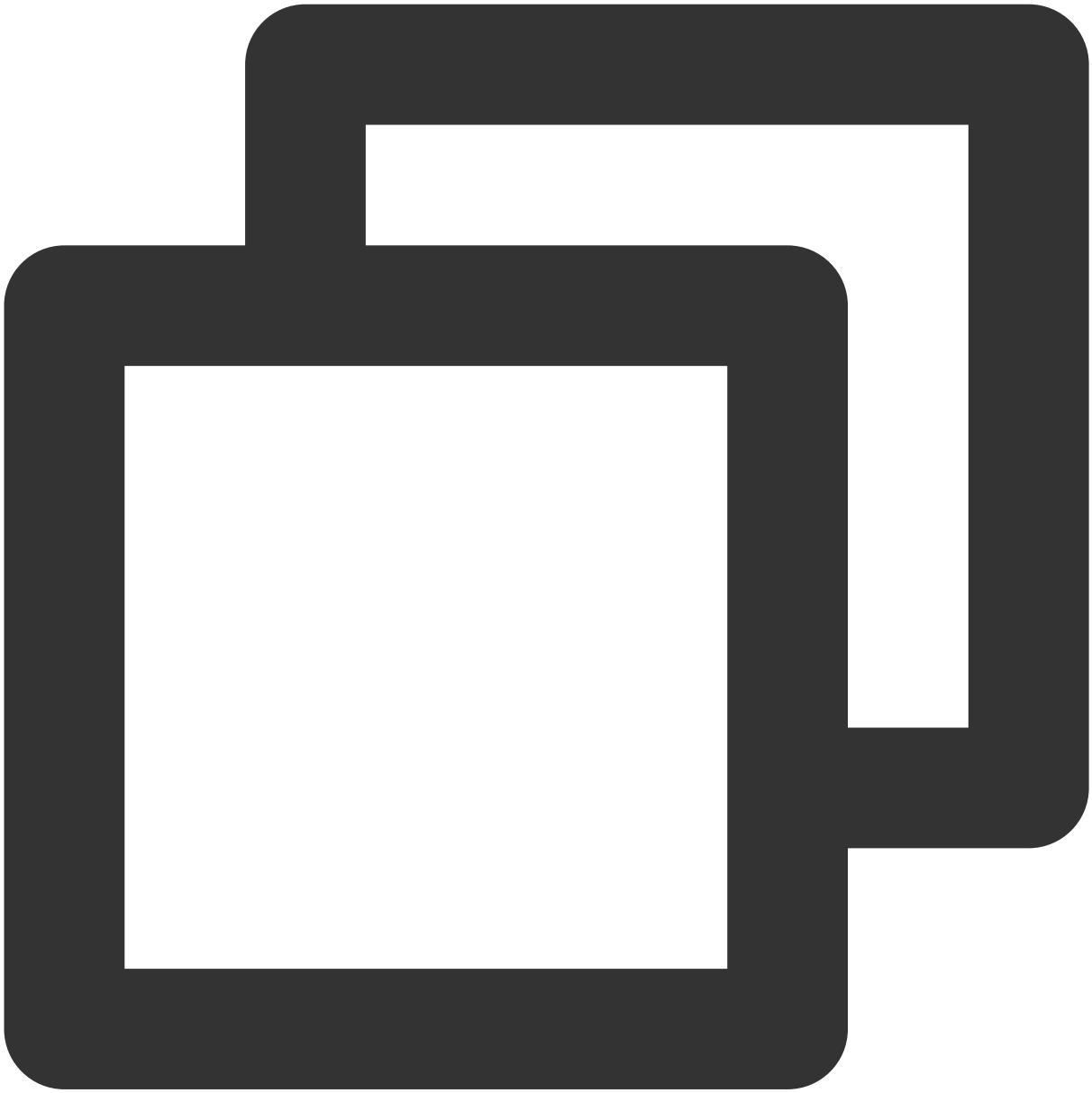
```
virtual void changeUserRole(const char* userId, TUIRole role, TUICallback* callback
```

参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
role	TUIRole	角色类型，详细定义可以参考TUIRoomDefine.h 中TUIRole的定义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中TUICallback 的定义

kickRemoteUserOutOfRoom

将远端用户踢出房间。



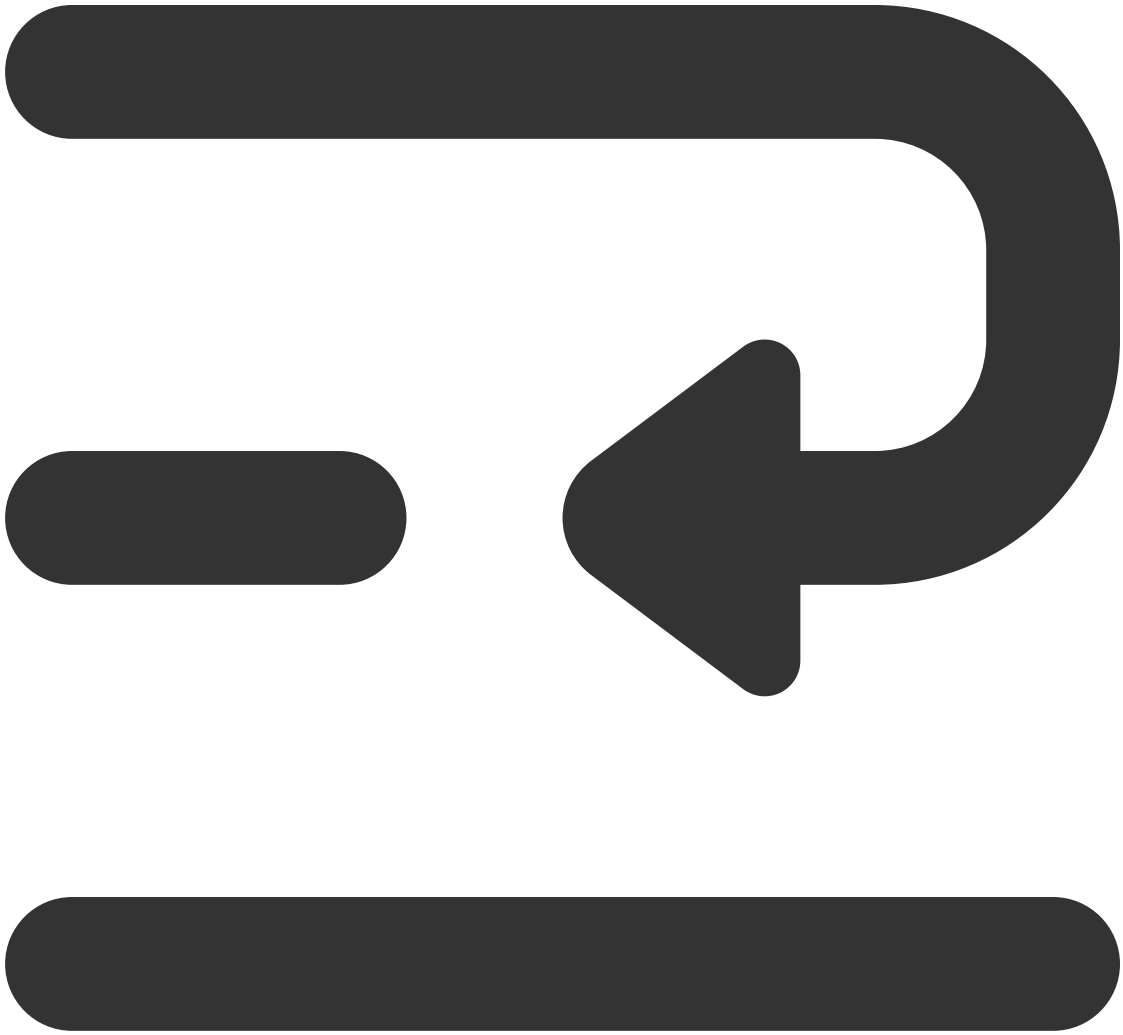
```
virtual void kickRemoteUserOutOfRoom(const char* userId, TUICallback* callback) = 0
```

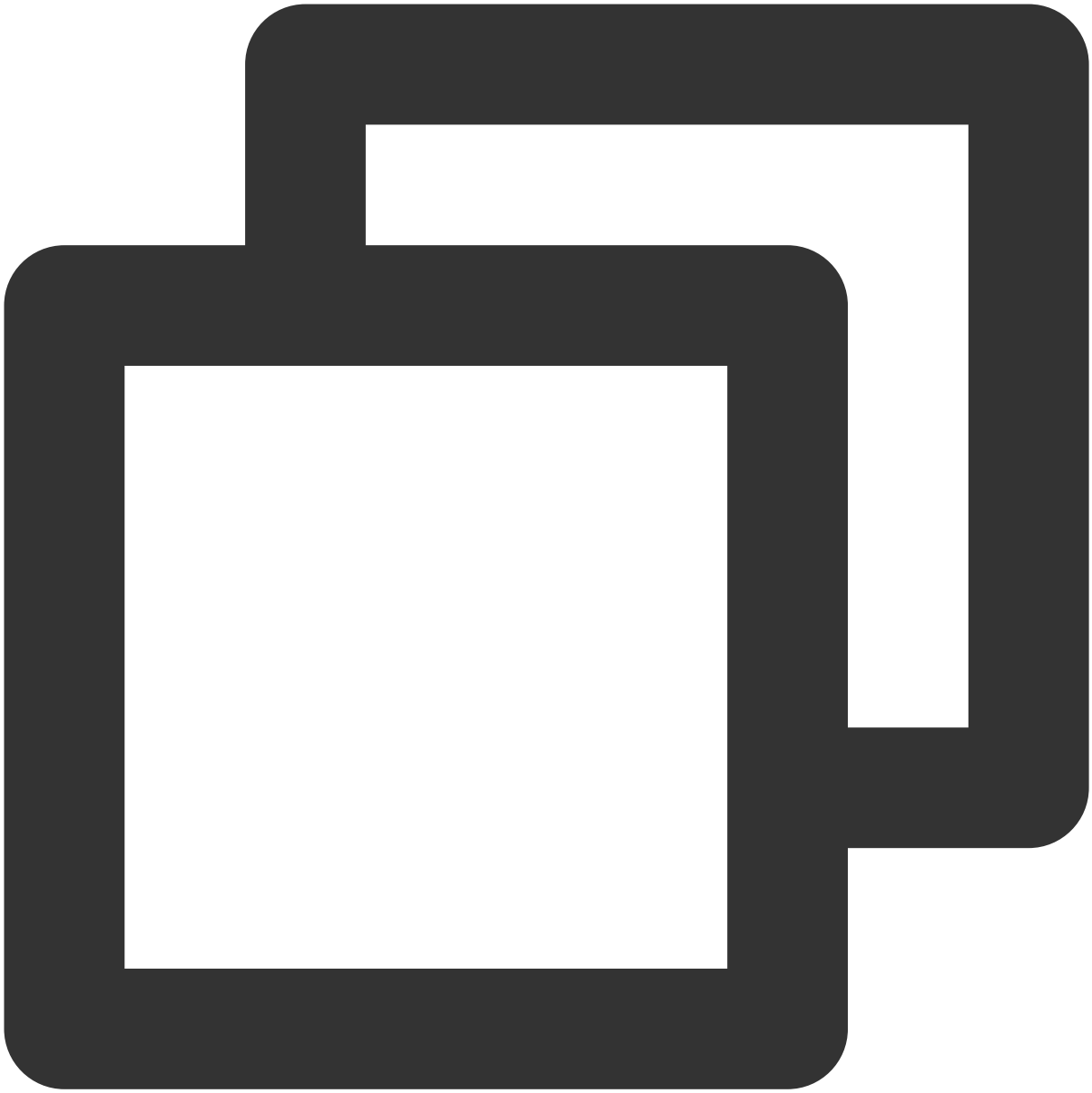
参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

disableDeviceForAllUserByAdmin

控制当前房间内所有用户是否可打开音频流、视频流采集设备的权限状态，例如：全员禁止打开麦克风、全员禁止打开摄像头、全员禁止打开屏幕分享
（目前仅会议场景下可用，并且只有管理员或群主能够调用）。





```
virtual void disableDeviceForAllUserByAdmin(TUIMediaDevice device, bool isDisable,
```

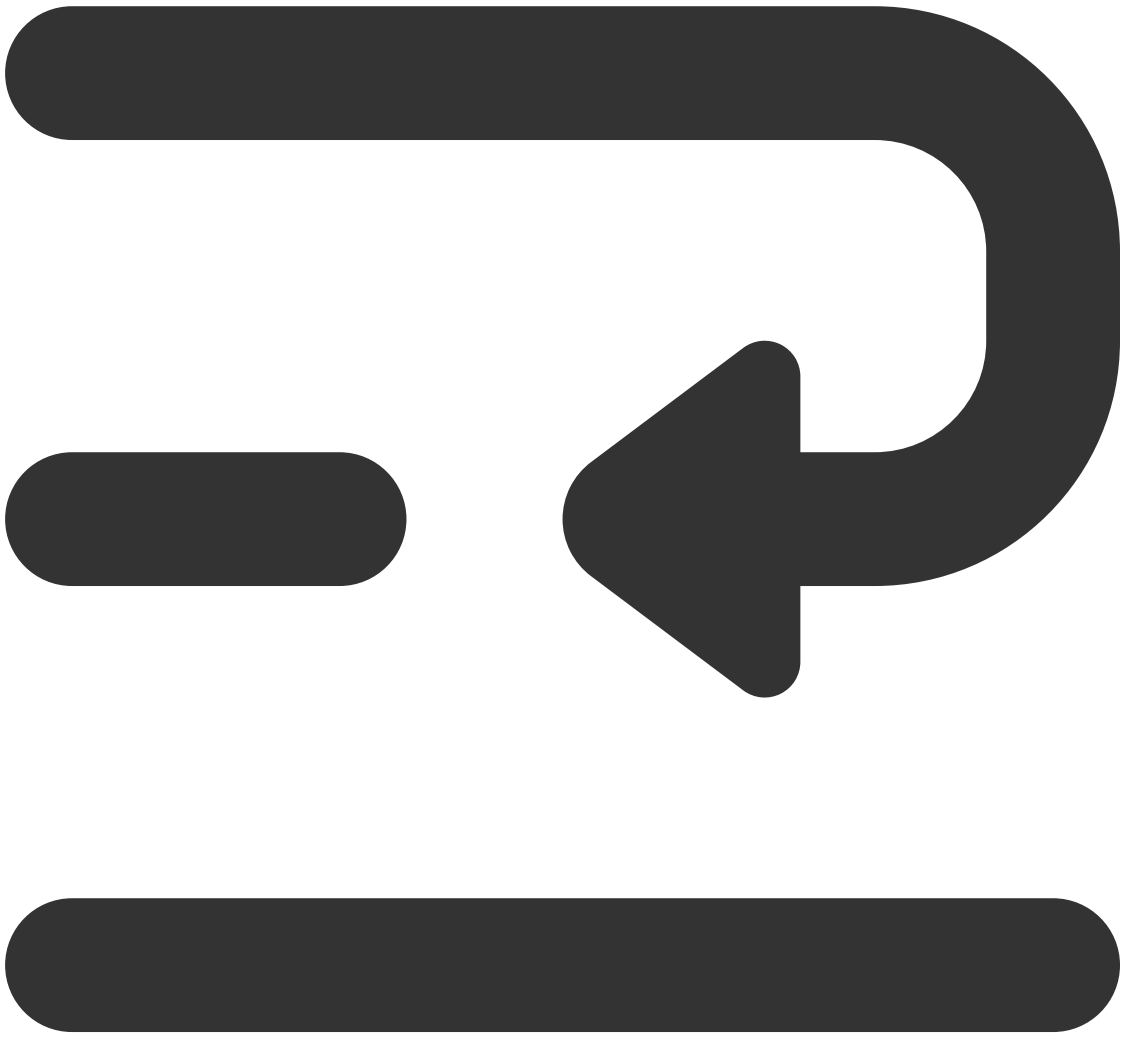
参数如下表所示：

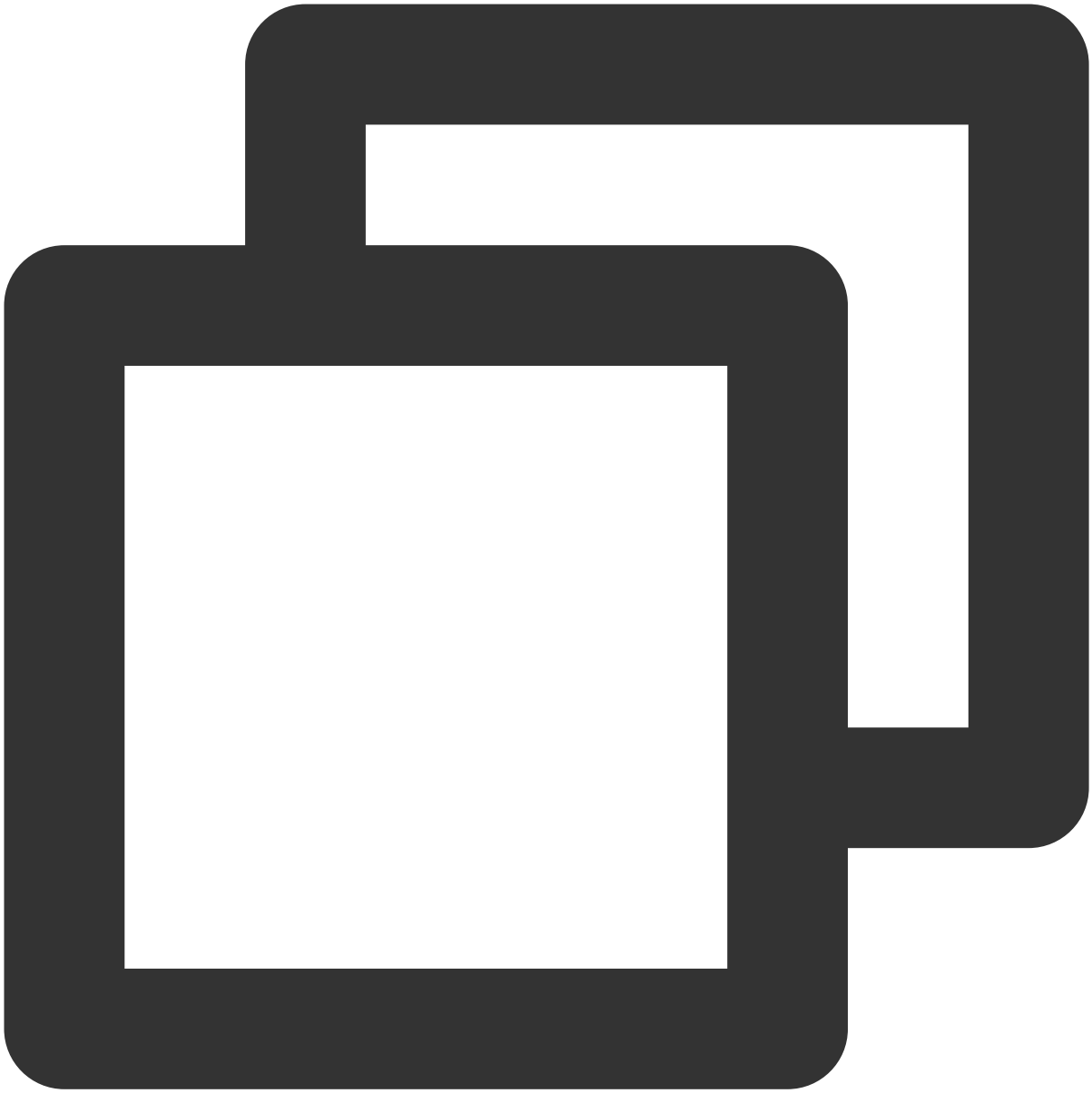
参数	类型	含义
device	TUIMediaDevice	设备类型。详细定义可以参考 TUIRoomDefine.h 中 TUIMediaDevice 的定义
isDisable	bool	是否禁用

callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

openRemoteDeviceByAdmin

请求远端用户打开媒体设备
（目前仅会议场景下可用，并且只有管理员或群主能够调用）。





```
virtual TUIRequest openRemoteDeviceByAdmin(const char* userId, TUIMediaDevice devic
```

参数如下表所示：

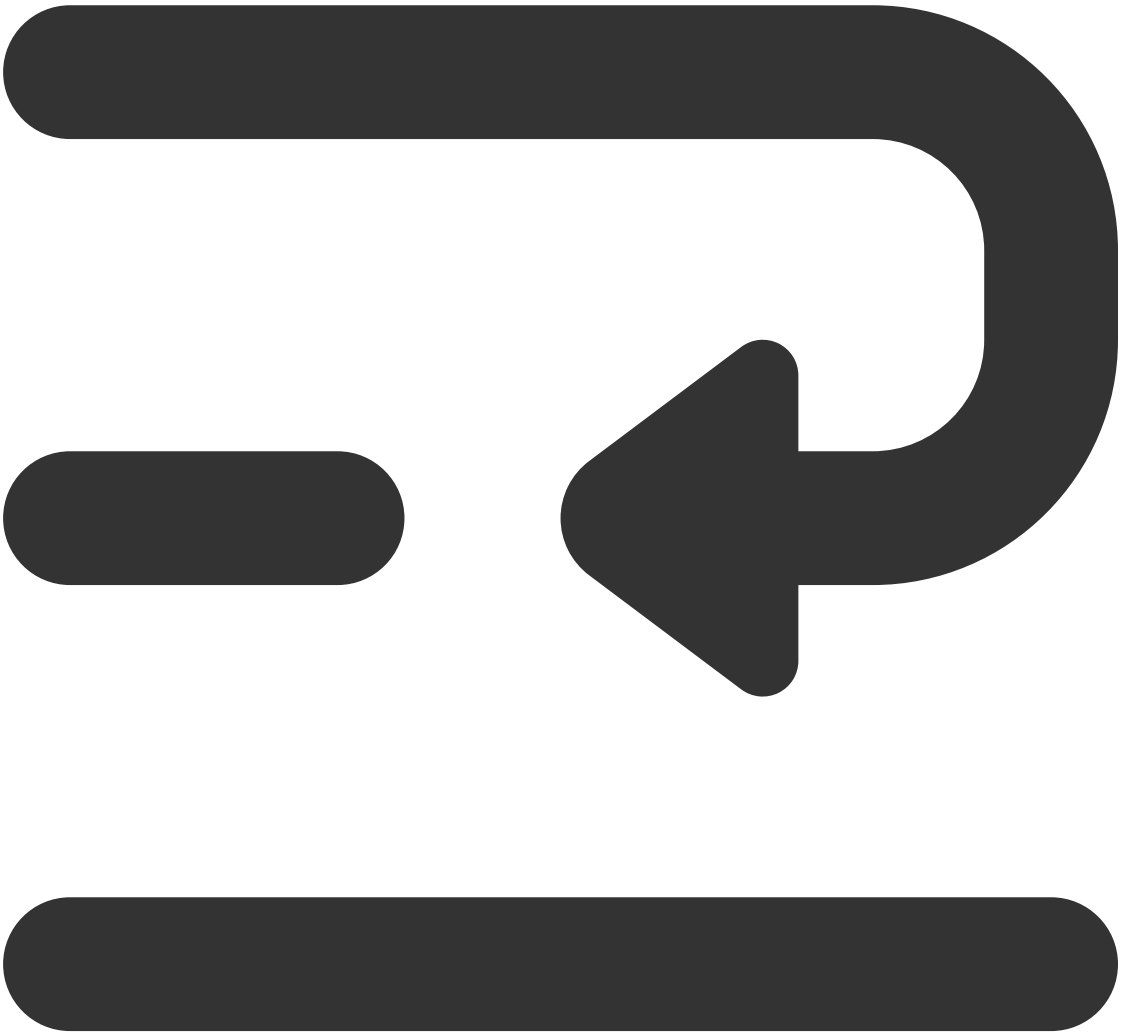
参数	类型	含义
userId	const char*	用户 ID
device	TUIMediaDevice	设备类型。详细定义可以参考 TUIRoomDefine.h 中 TUIMediaDevice 的定义

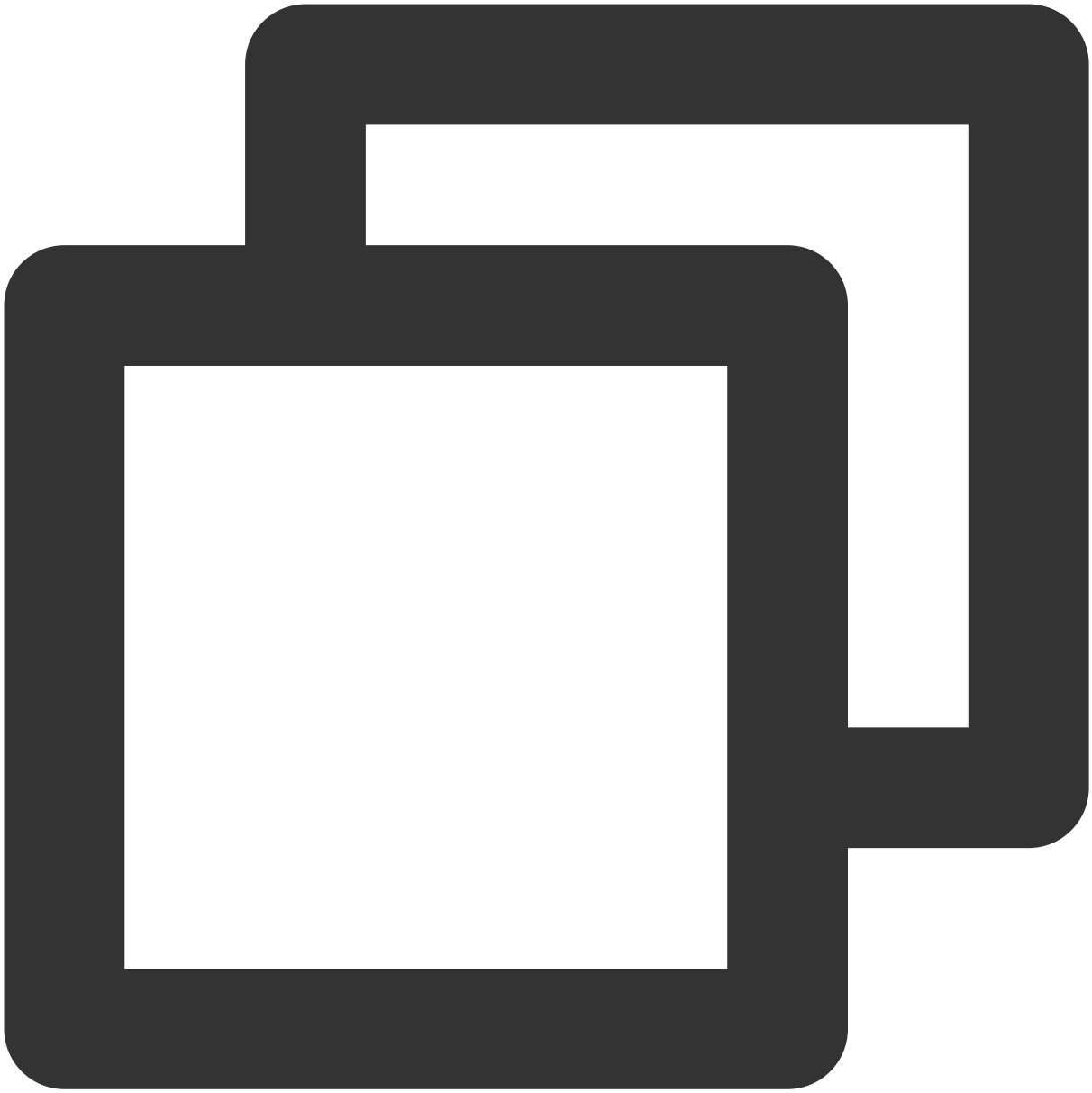
timeout	int	超时时间
callback	TUIRequestCallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIRequestCallback 的定义

closeRemoteDeviceByAdmin

关闭远端用户媒体设备

（目前仅会议场景下可用，并且只有管理员或群主能够调用）。





```
virtual void closeRemoteDeviceByAdmin(const char* userId, TUIMediaDevice device, TU
```

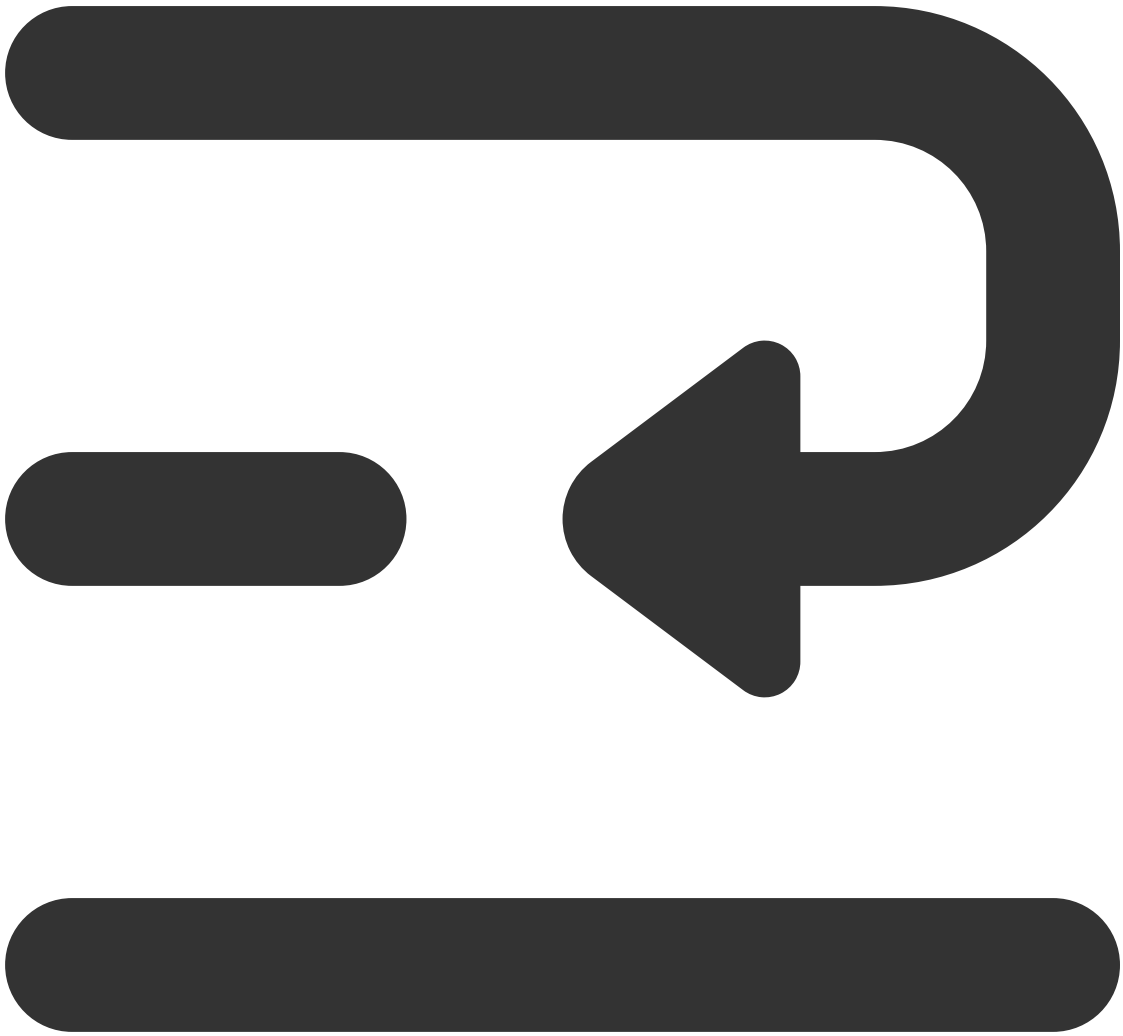
参数如下表所示：

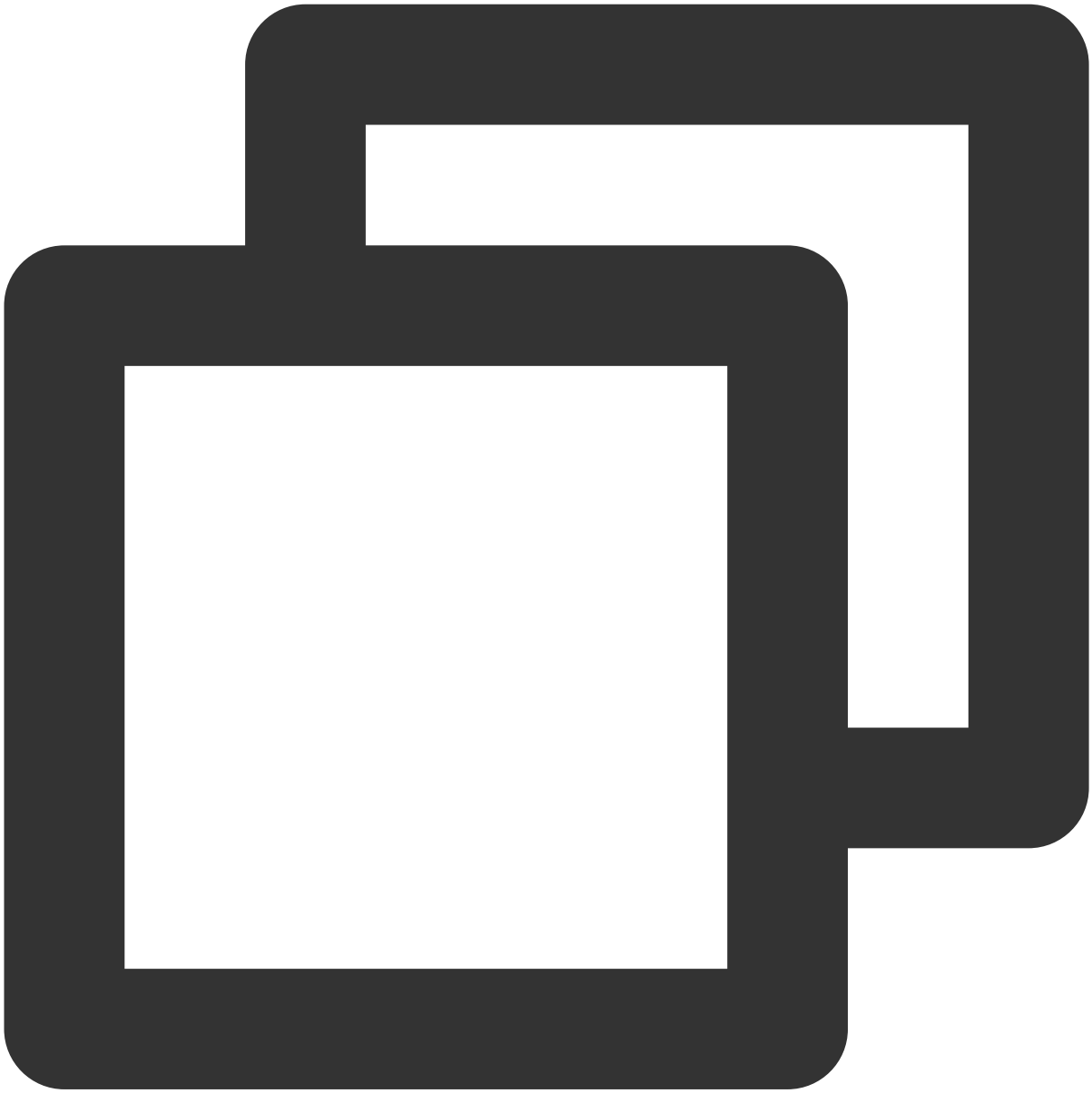
参数	类型	含义
userId	const char*	用户 ID
device	TUIMediaDevice	设备类型。详细定义可以参考 TUIRoomDefine.h 中 TUIMediaDevice 的定义

callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

applyToAdminToOpenLocalDevice

请求打开本地媒体设备
（目前仅会议场景下可用，并且只有普通成员调用）。





```
virtual TUIRequest applyToAdminToOpenLocalDevice(TUIMediaDevice device, int timeout
```

参数如下表所示：

参数	类型	含义
device	TUIMediaDevice	设备类型。详细定义可以参考 TUIRoomDefine.h 中 TUIMediaDevice 的定义
timeout	int	超时时间

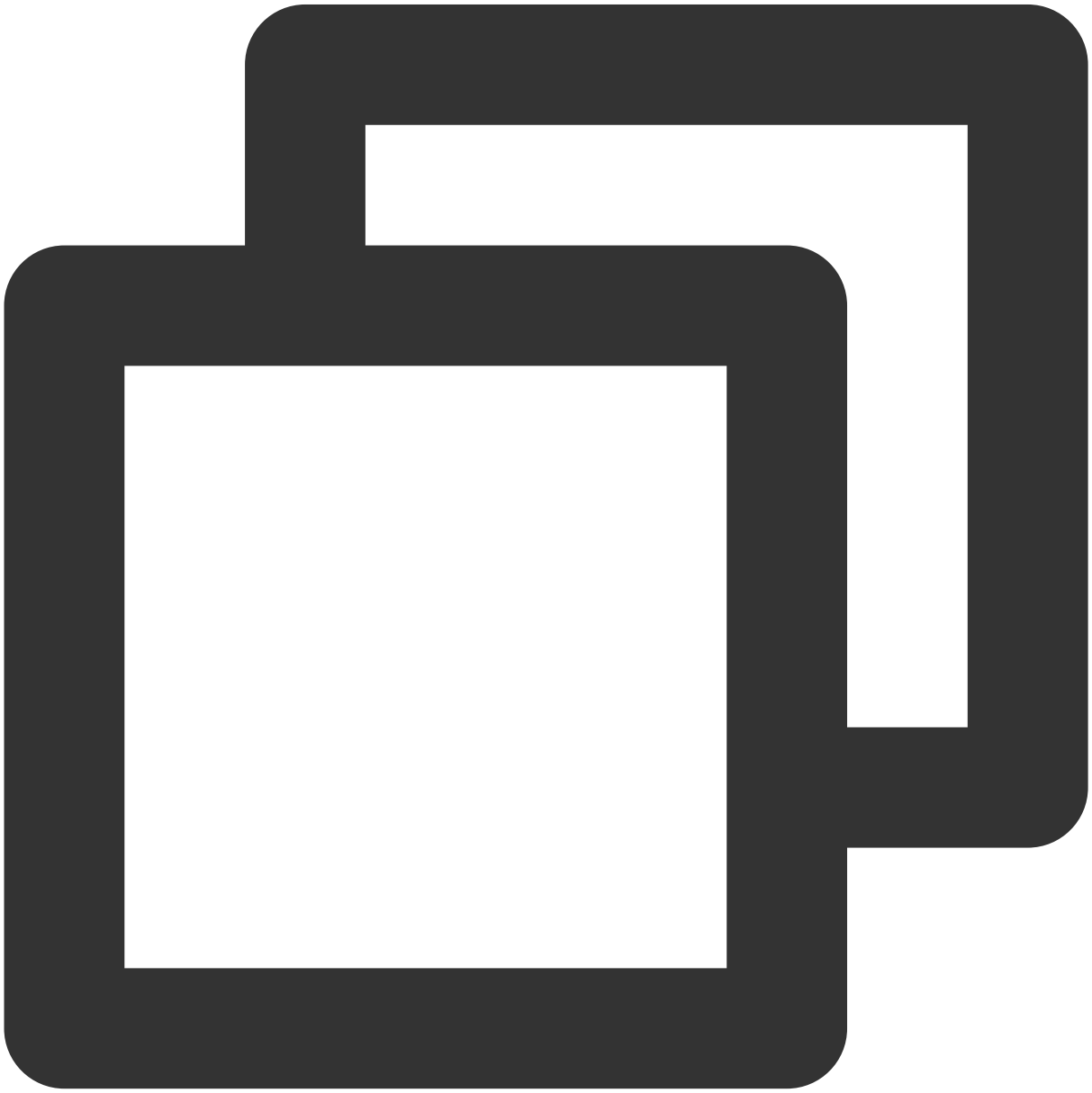
callback	TUIRequestCallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIRequestCallback 的定义

setMaxSeatCount

设置最大麦位数（仅支持进房前和创建房间时设置）。

roomType 为 TUIRoomType::kConference (教育及会议场景) 时，maxSeatCount 值不做限制；

roomType 为 TUIRoomType::kLivingRoom (直播场景) 时，maxSeatCount 最大限制为 16；



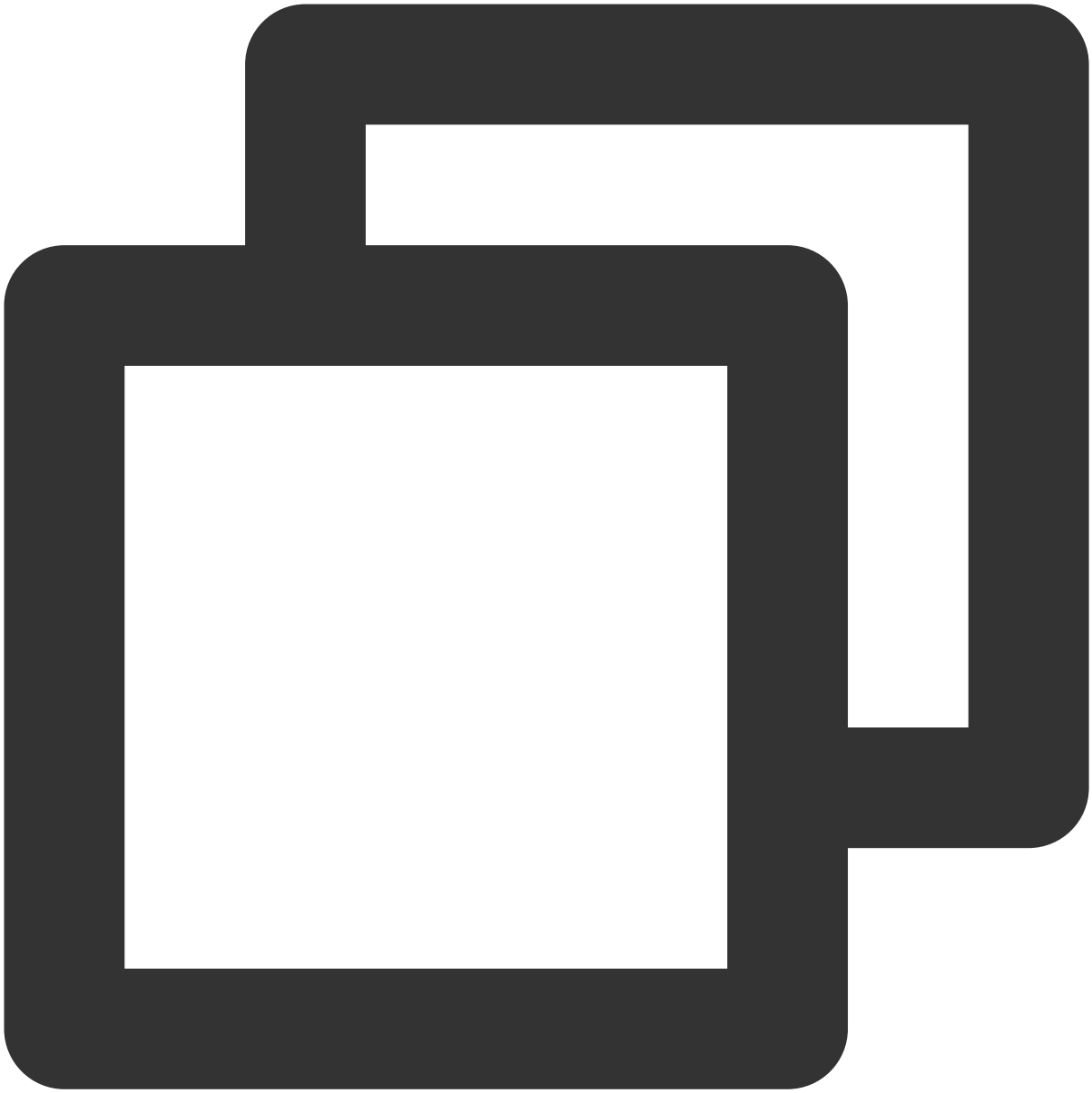
```
virtual void setMaxSeatCount(uint32_t maxSeatCount, TUICallback* callback) = 0;
```

参数如下表所示：

参数	类型	含义
maxSeatCount	uint32_t	最大麦位数量限制
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

getSeatList

获取麦位列表。



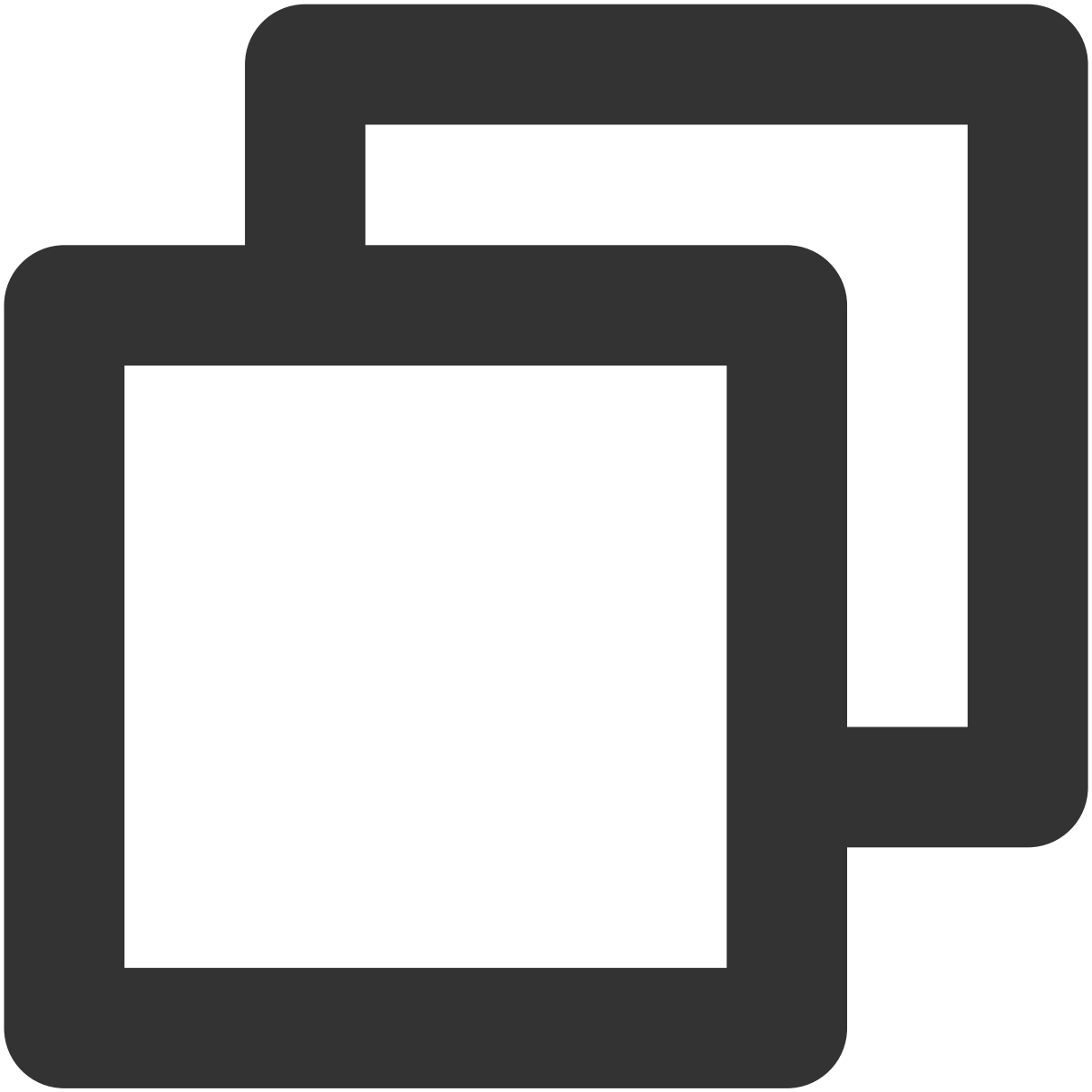
```
virtual void getSeatList(TUIListCallback<TUISeatInfo>* callback) = 0;
```

参数如下表所示：

参数	类型	含义
callback	TUIListCallback<TUISeatInfo>*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIListCallback 和 TUISeatInfo 的定义

lockSeatByAdmin

锁定麦位。



```
virtual void lockSeatByAdmin(int seatIndex, const TUISeatLockParams& lockParams, TU
```

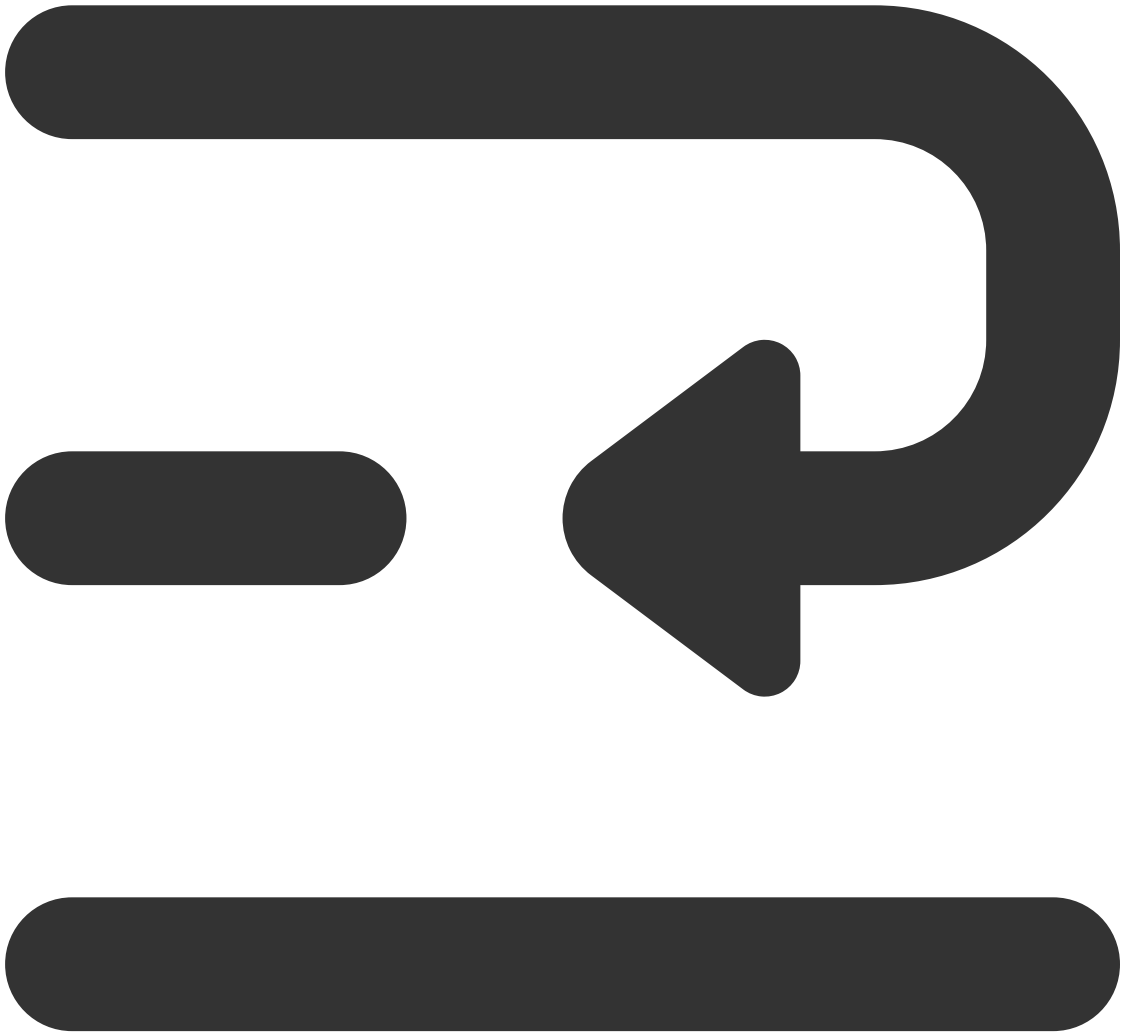
参数如下表所示：

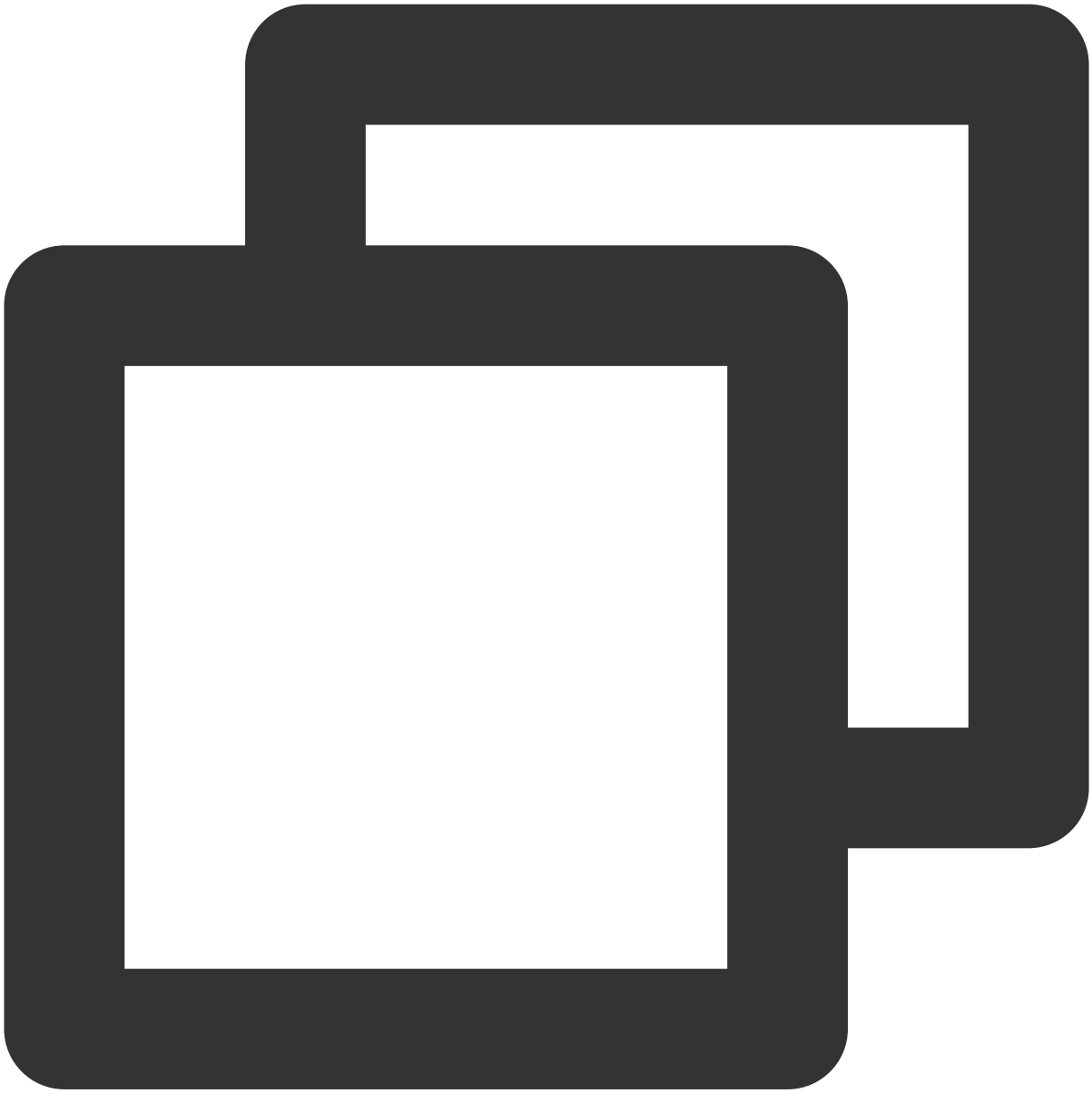
参数	类型	含义
seatIndex	int	麦位编号
lockParams	TUISeatLockParams	锁定麦位相关参数，详细定义可以参考

		TUIRoomDefine.h 中 TUISeatLockParams 的定义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

takeSeat

本地上麦。





```
virtual TUIRequest takeSeat(int seatIndex, int timeout, TUIRequestCallback* callbac
```

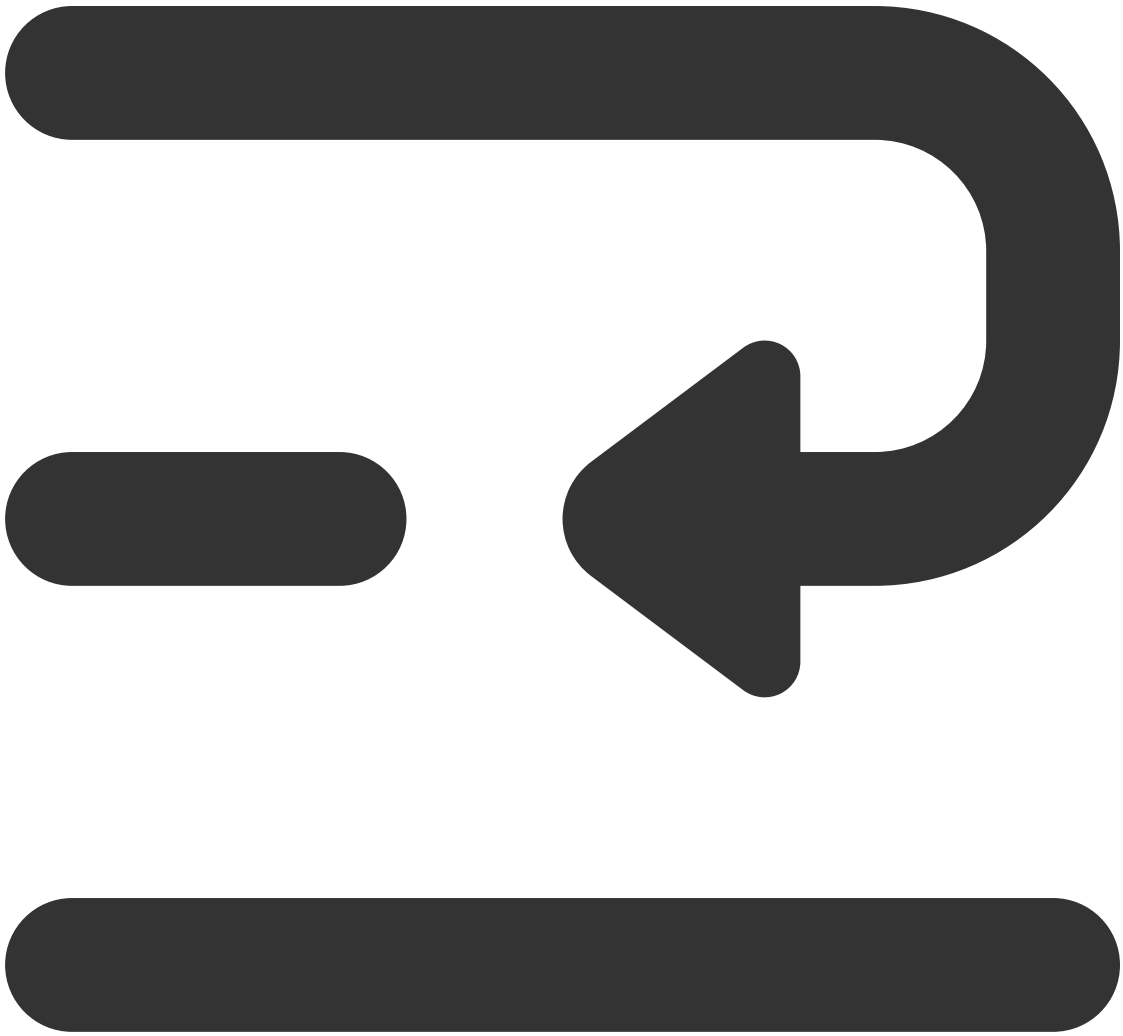
参数如下表所示：

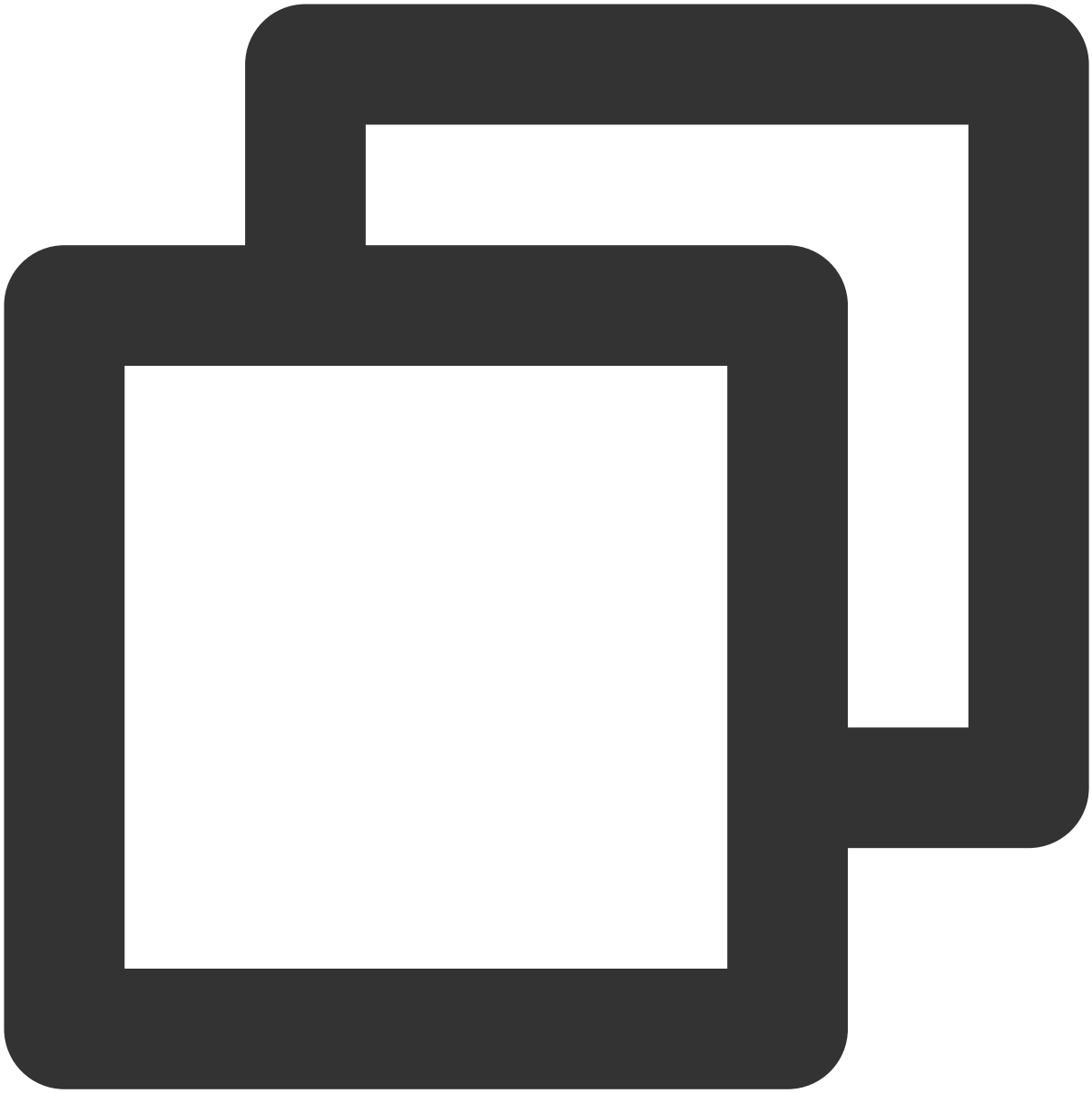
参数	类型	含义
seatIndex	int	麦位编号
timeout	int	超时时间
callback	TUIRequestCallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考TUIRoomDefine.h 中 TUIRequestCallback 的定义

leaveSeat

本地下麦。





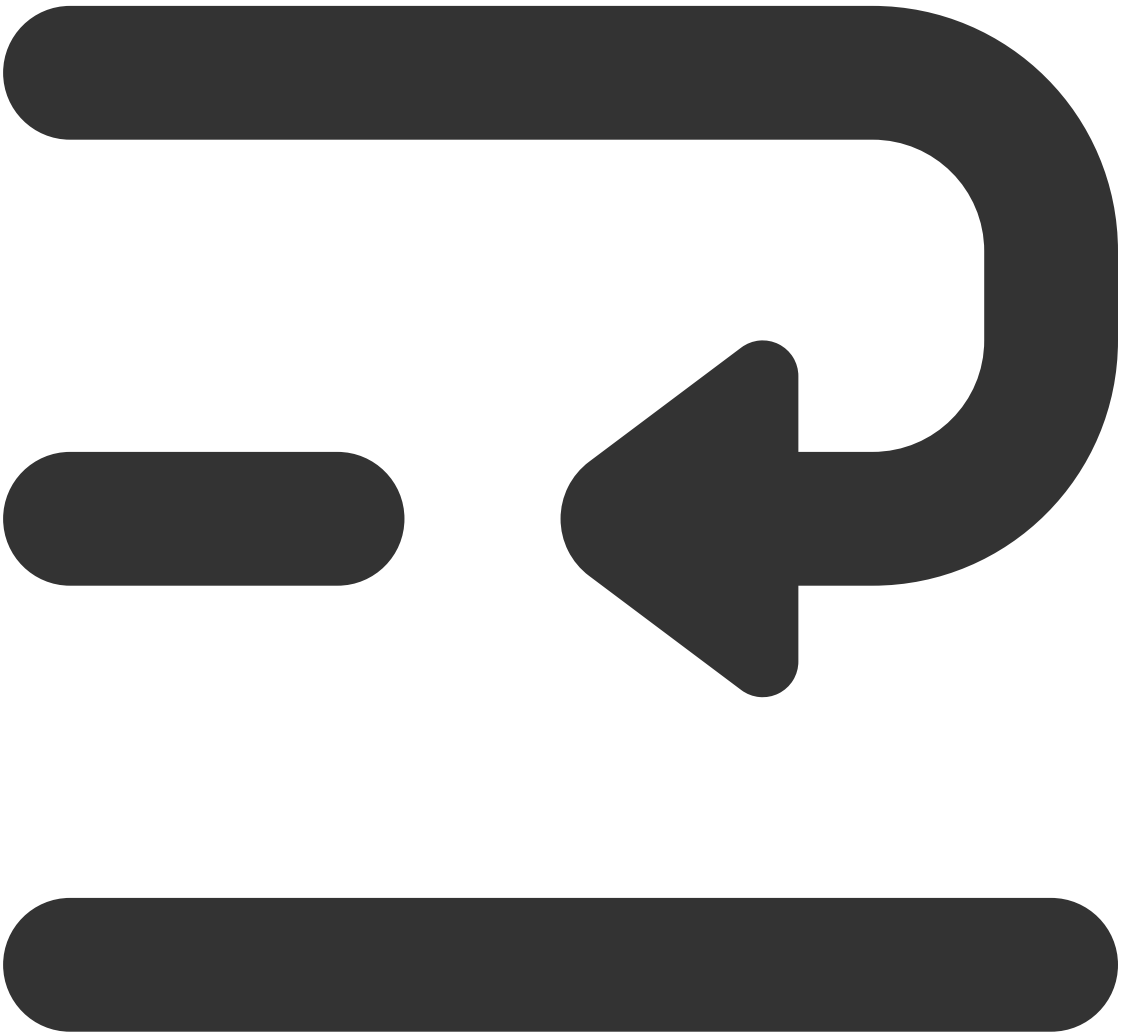
```
virtual void leaveSeat(TUICallback* callback) = 0;
```

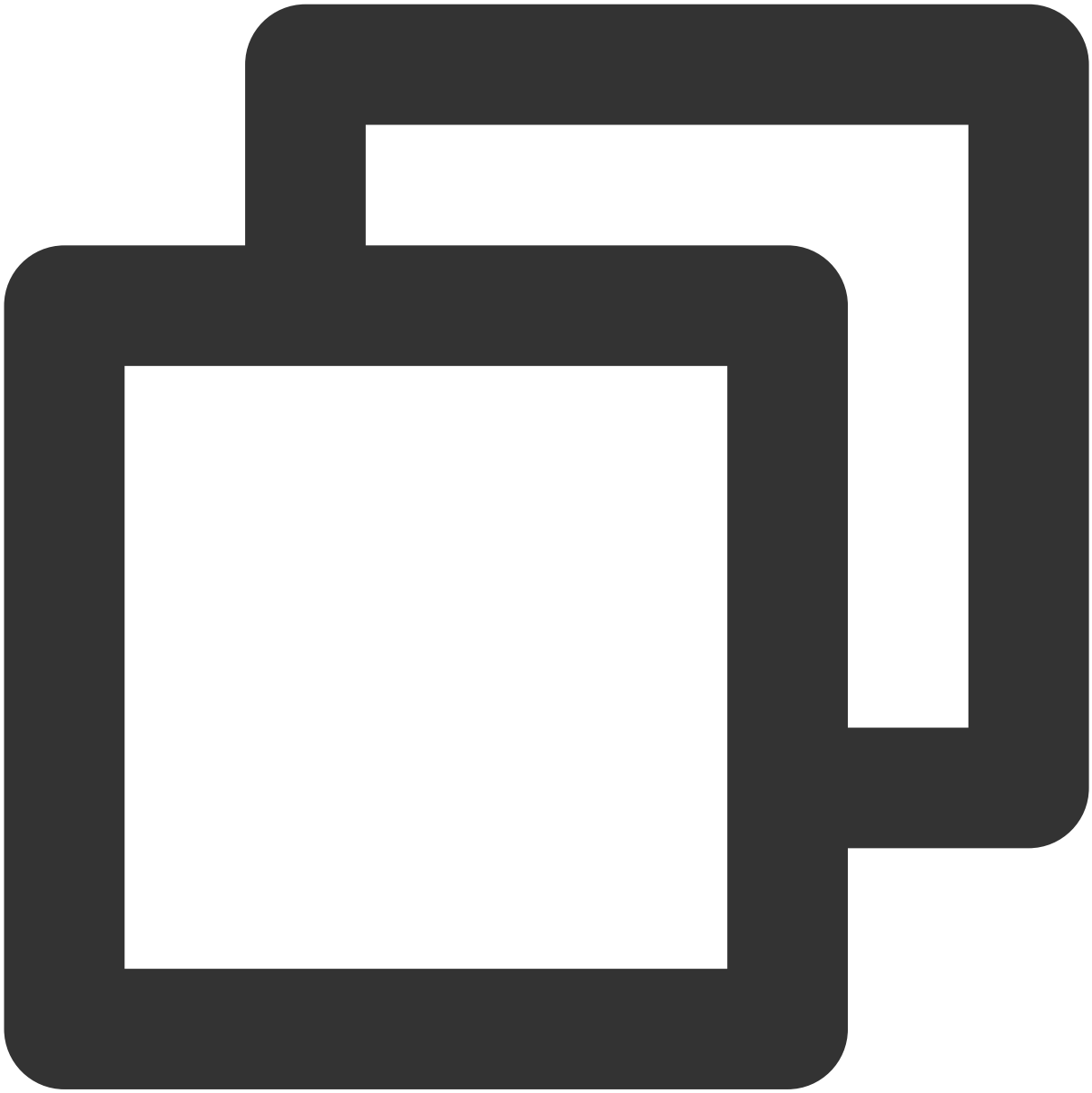
参数如下表所示：

参数	类型	含义
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

takeUserOnSeatByAdmin

主持人邀请用户上麦。





```
virtual TUIRequest takeUserOnSeatByAdmin(int seatIndex, const char* userId, int tim
```

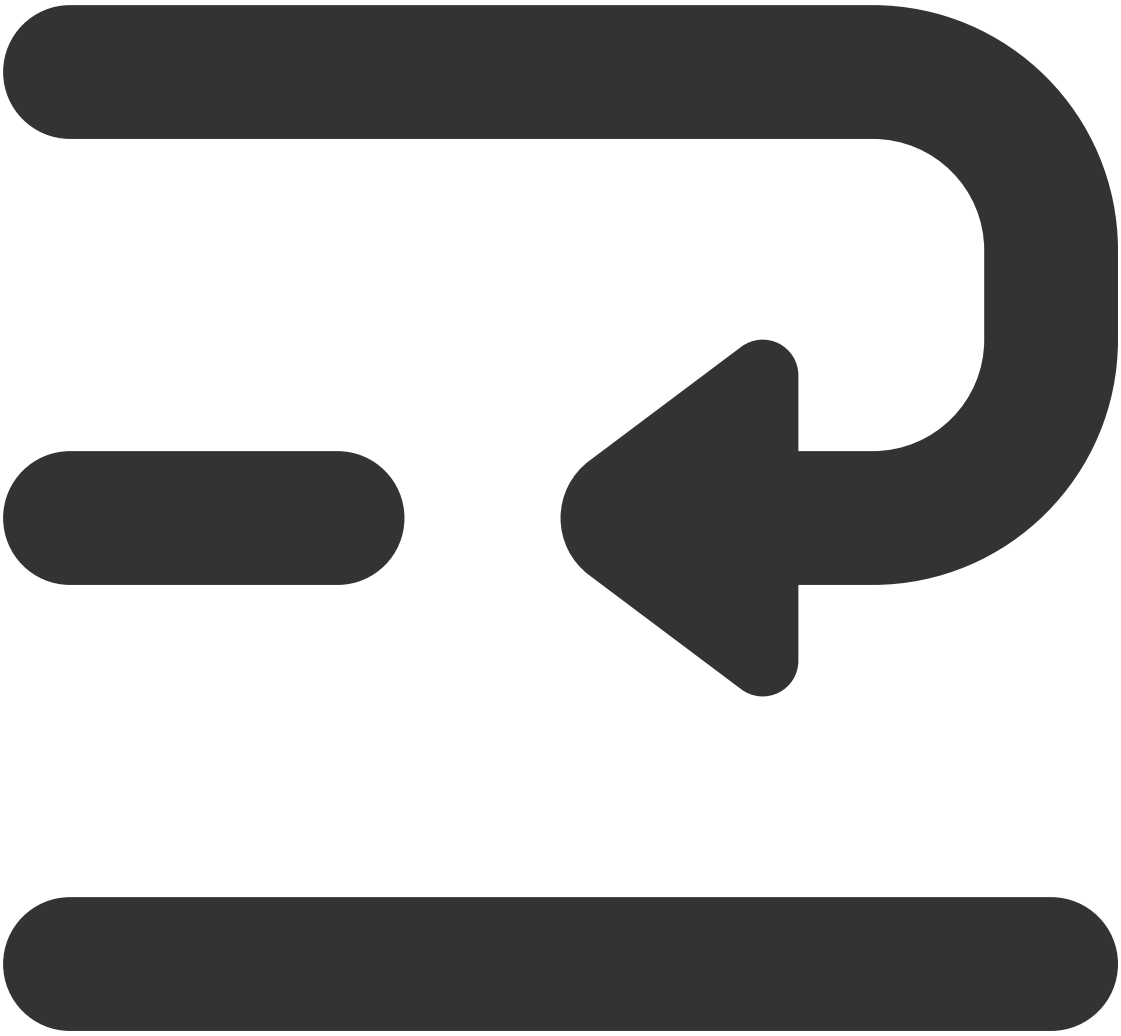
参数如下表所示：

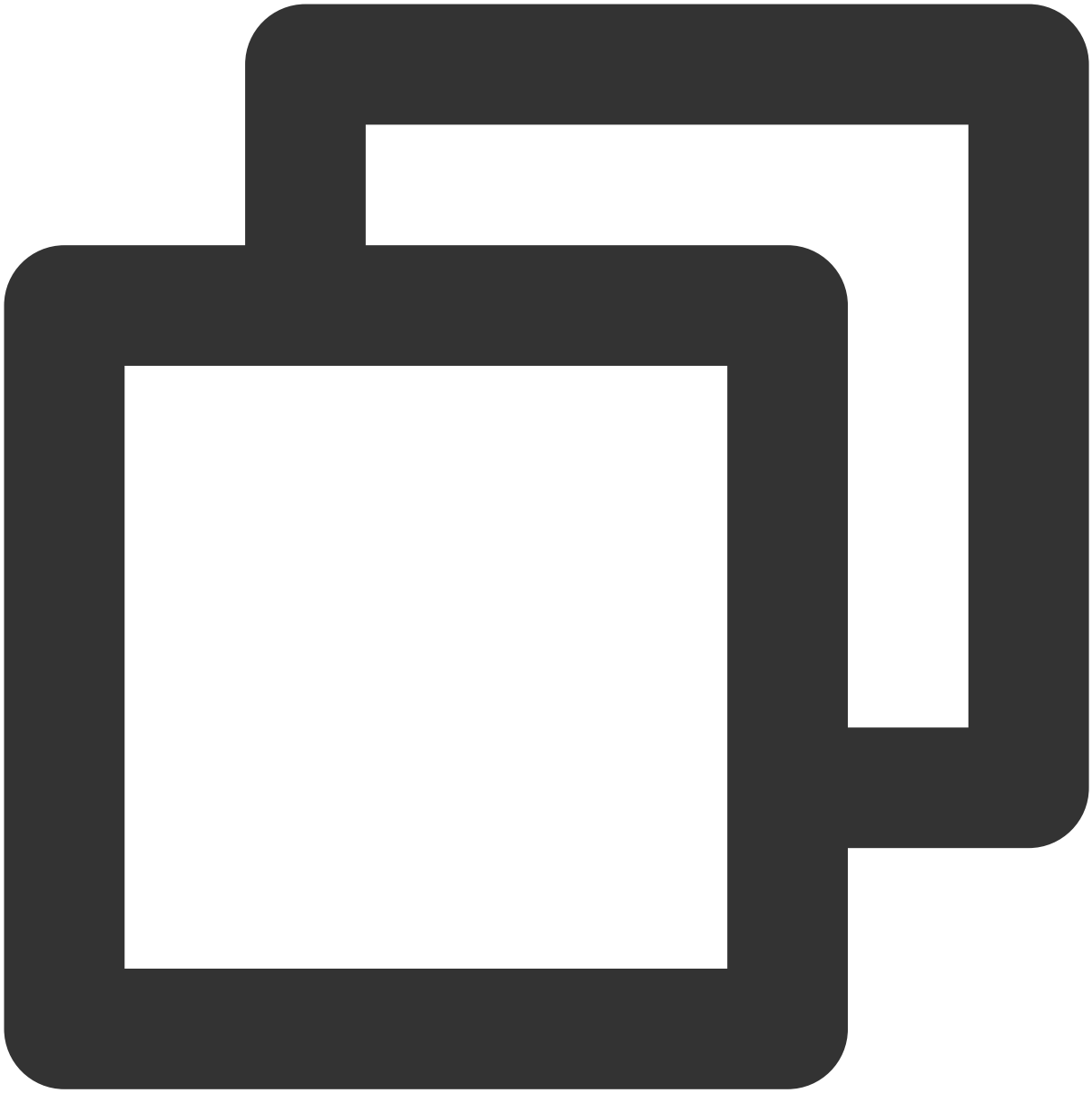
参数	类型	含义
seatIndex	int	麦位编号
userId	const char*	用户 ID
timeout	int	超时时间

callback	TUIRequestCallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUIRequestCallback 的定义

kickUserOffSeatByAdmin

主持人邀请用户下麦。





```
virtual void kickUserOffSeatByAdmin(int seatIndex, const char* userId, TUICallback*
```

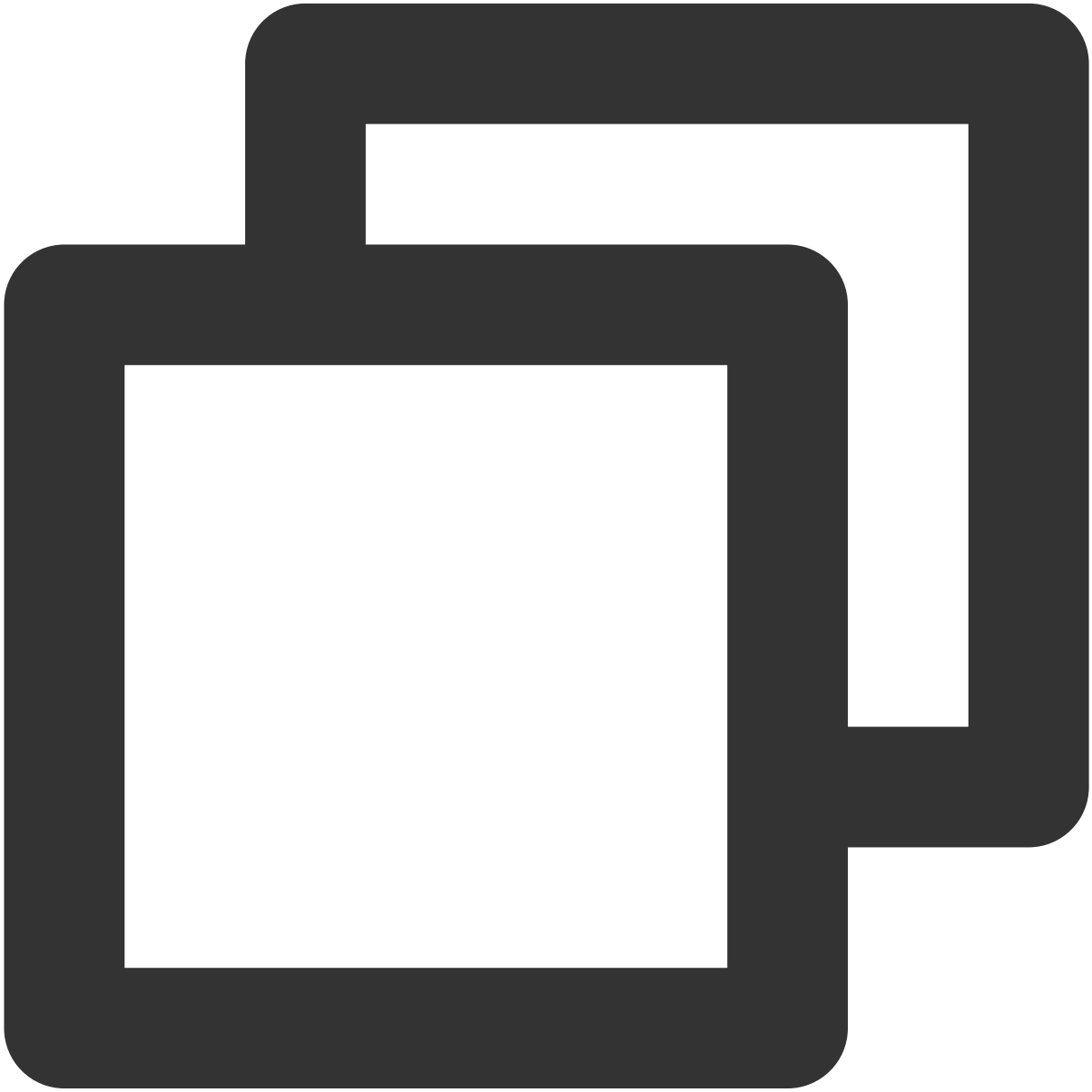
参数如下表所示：

参数	类型	含义
seatIndex	int	麦位编号
userId	const char*	用户 ID
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

sendTextMessage

发送本文消息。



```
virtual void sendTextMessage(const char* message, TUICallback* callback) = 0;
```

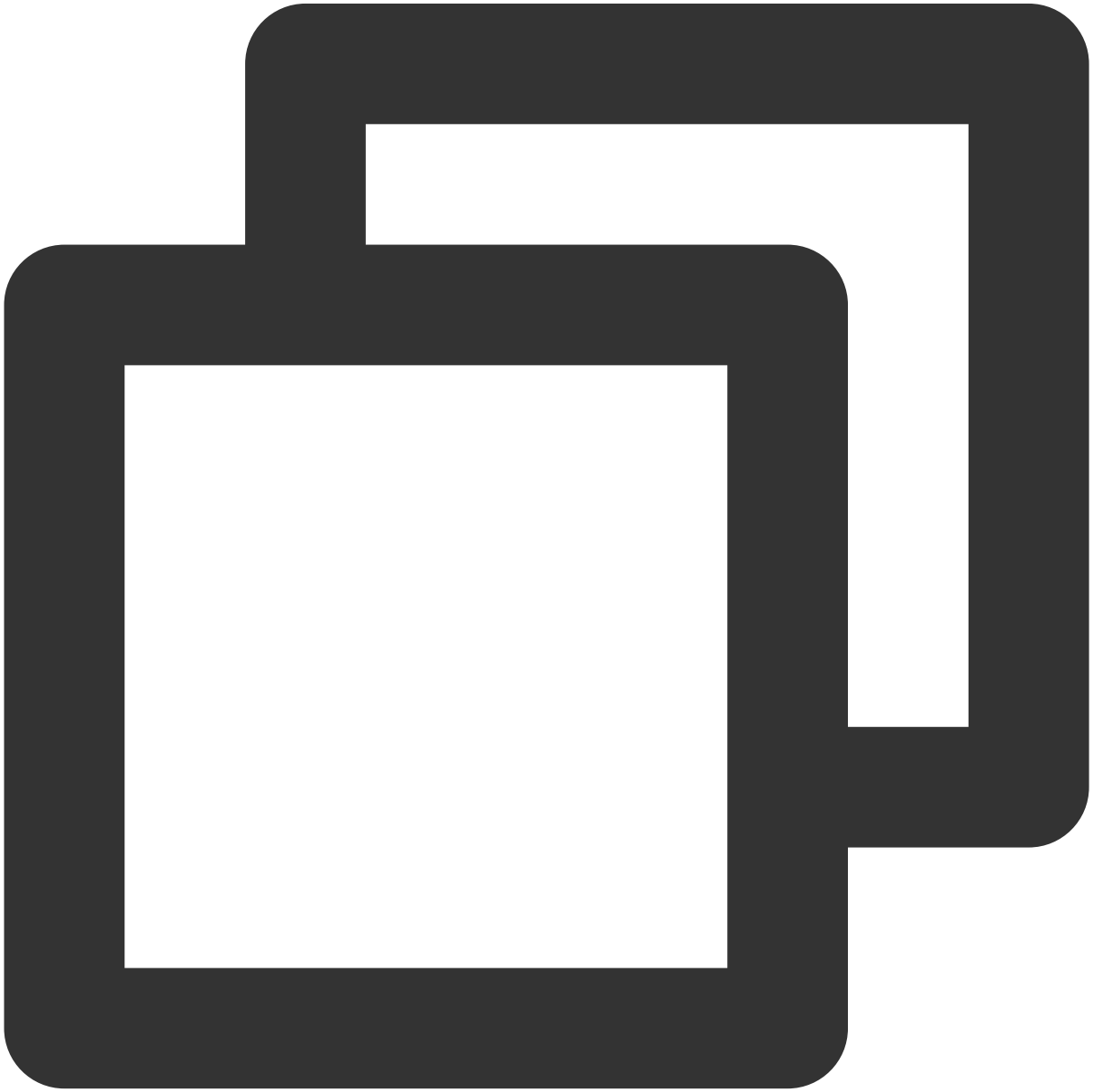
参数如下表所示：

参数	类型	含义

message	const char*	消息内容
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

sendCustomMessage

发送自定义消息。



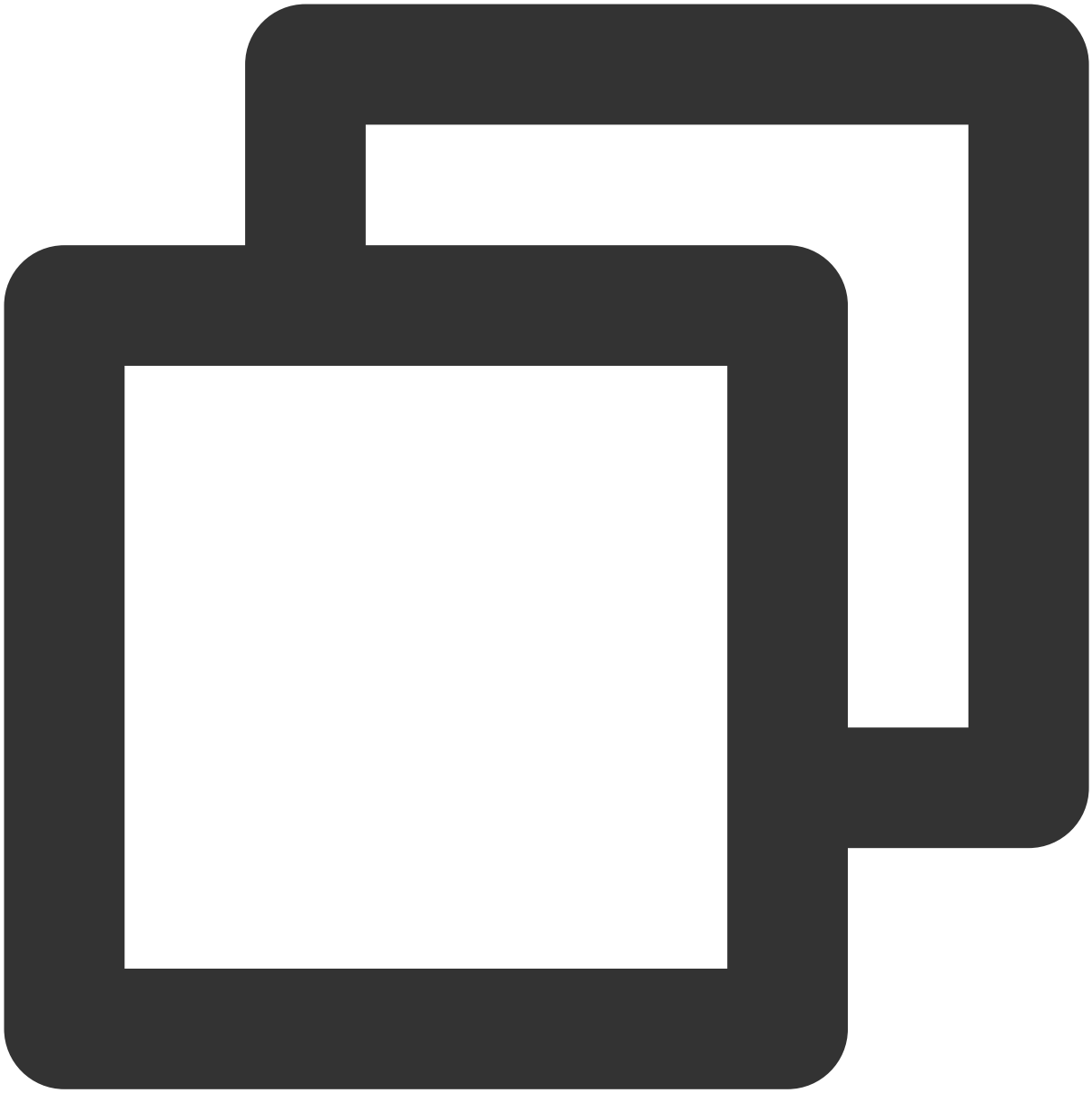
```
virtual void sendCustomMessage(const char* message, TUICallback* callback) = 0;
```

参数如下表所示：

参数	类型	含义
message	const char*	消息内容
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

disableSendingMessageByAdmin

禁用远端用户的发送文本消息能力（只有管理员或房主能够调用）



```
virtual void disableSendingMessageByAdmin(const char* userId, bool isDisable, TUICa
```

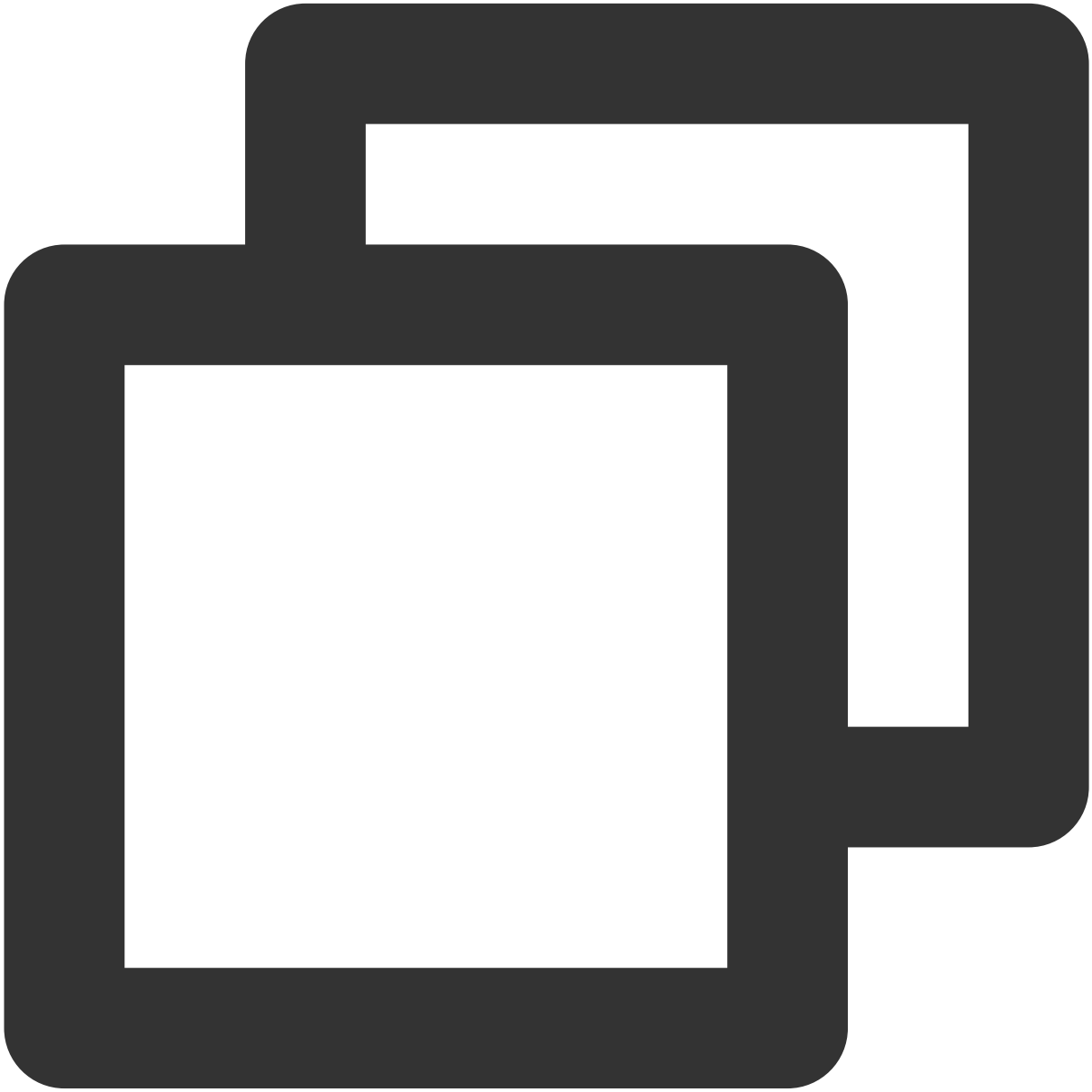
参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
isDisable	bool	是否禁用
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失

		败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

disableSendingMessageForAllUser

禁用所有用户的发送文本消息能力（只有管理员或房主能够调用）



```
virtual void disableSendingMessageForAllUser(bool isDisable, TUICallback* callback)
```

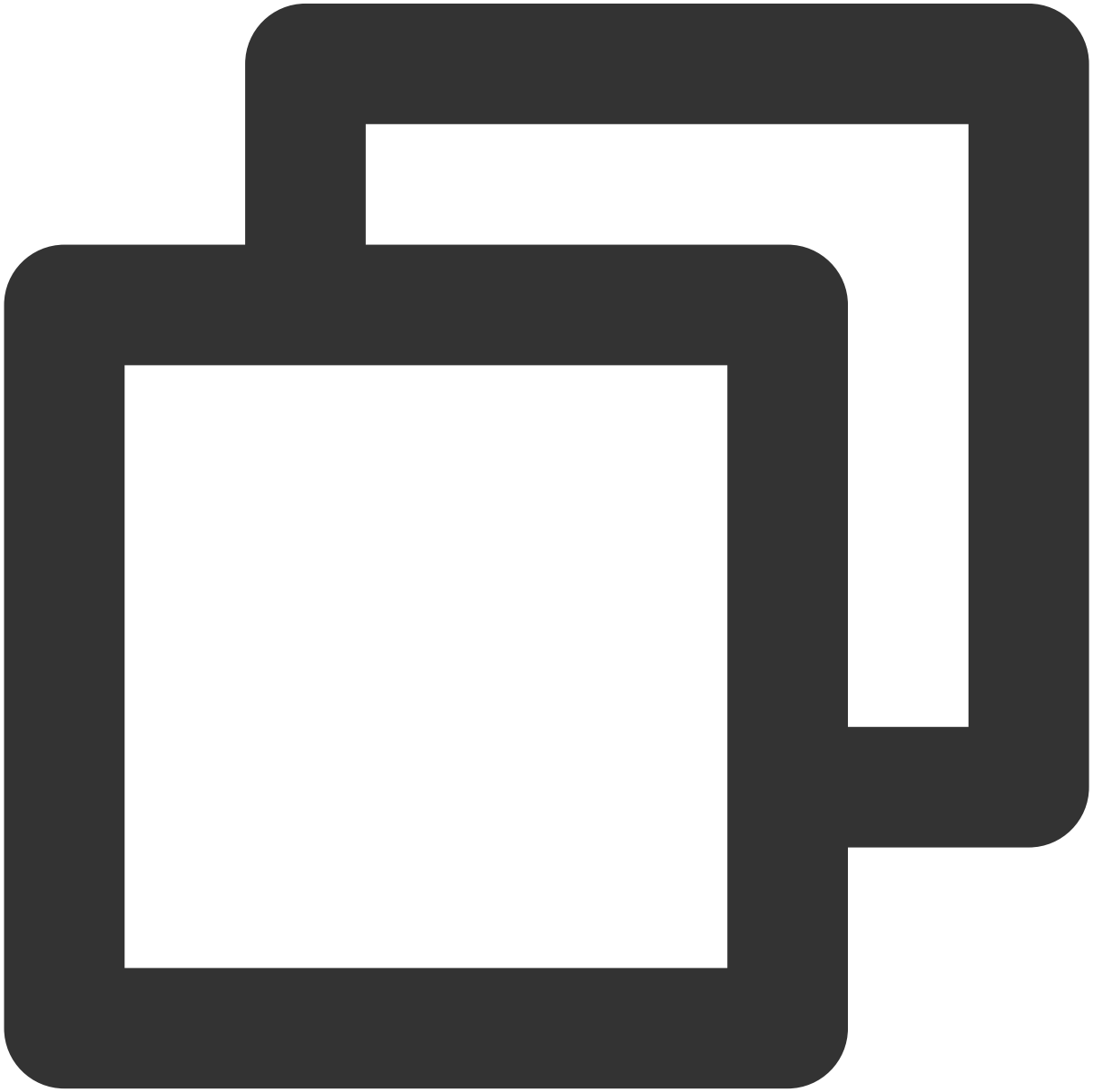
参数如下表所示：

参数	类型	含义

isDisable	bool	是否禁用
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

cancelRequest

取消请求。



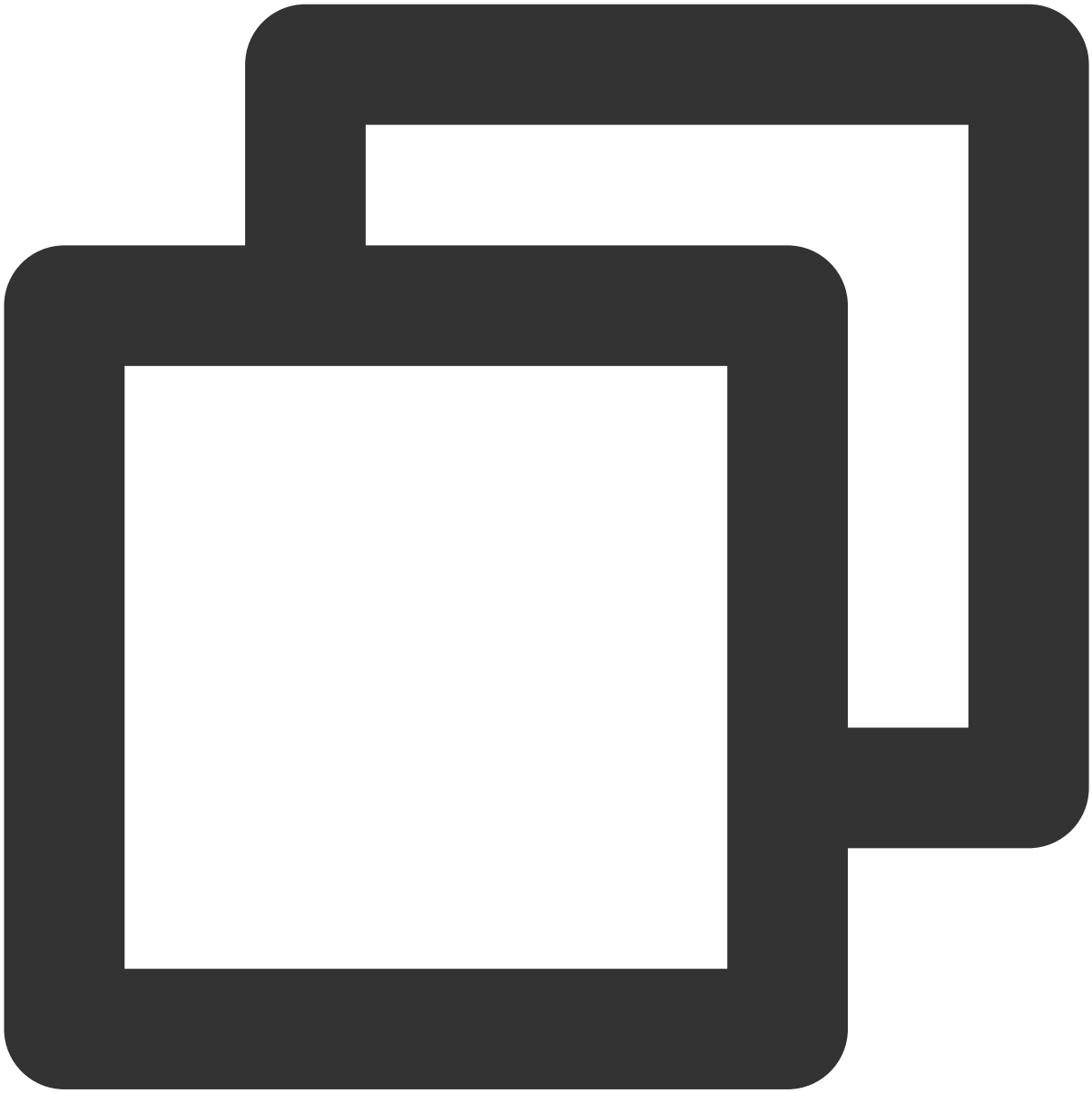
```
virtual void cancelRequest(const char* requestId, TUICallback* callback) = 0;
```

参数如下表所示：

参数	类型	含义
requestId	const char*	请求 ID。发送请求的接口返回。
callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

responseRemoteRequest

回复请求。



```
virtual void responseRemoteRequest(const char* requestId, bool agree, TUICallback*
```

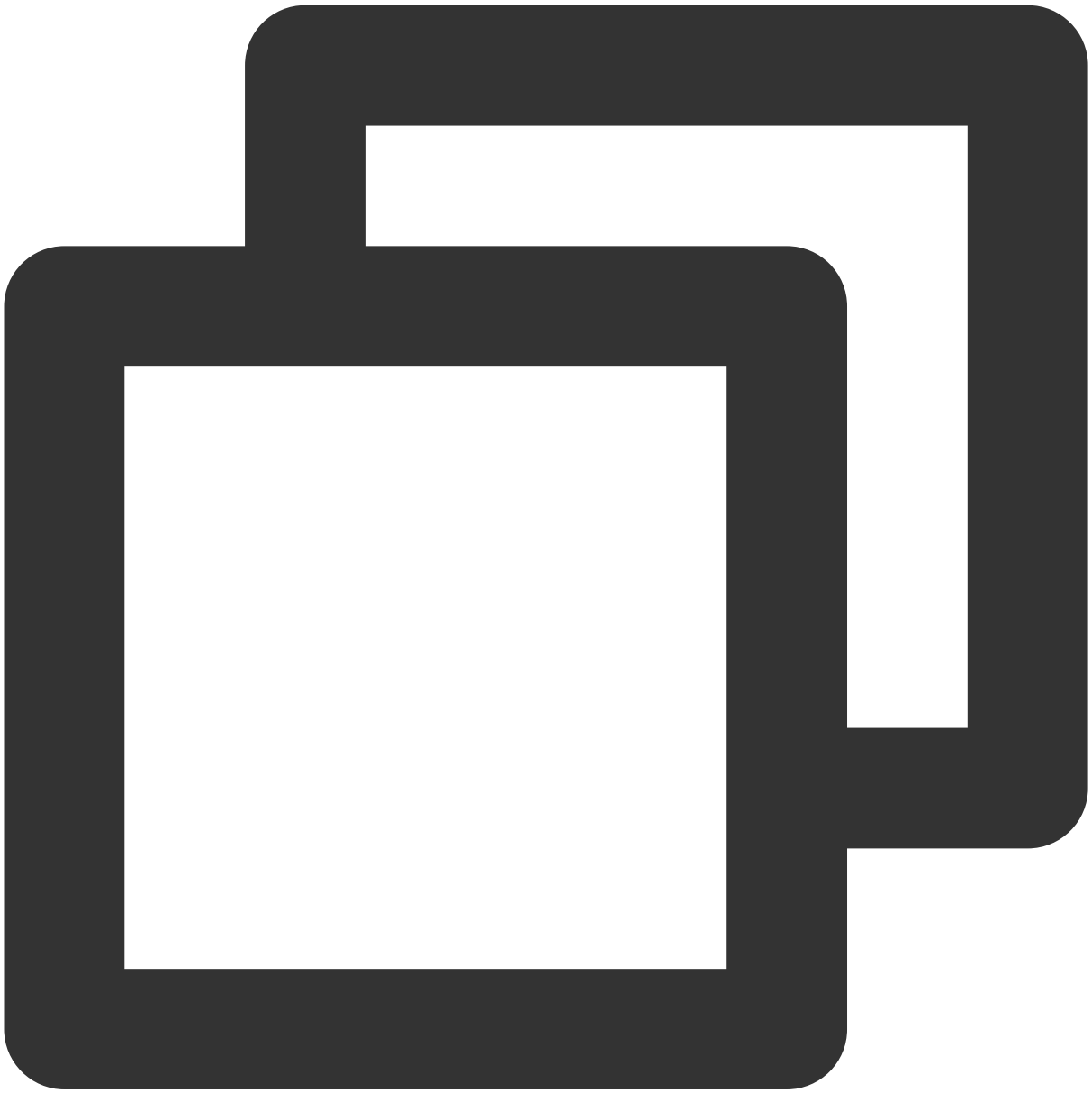
参数如下表所示：

参数	类型	含义
requestId	const char*	请求 ID。发送请求的接口返回或者 OnRequestReceived事件通知。
agree	bool	是否同意 true: 同意请求，false: 拒绝请求

callback	TUICallback*	调用接口的回调，用于通知接口调用的成功或者失败。详细定义可以参考TUIRoomDefine.h 中 TUICallback 的定义

getTRTCCloud

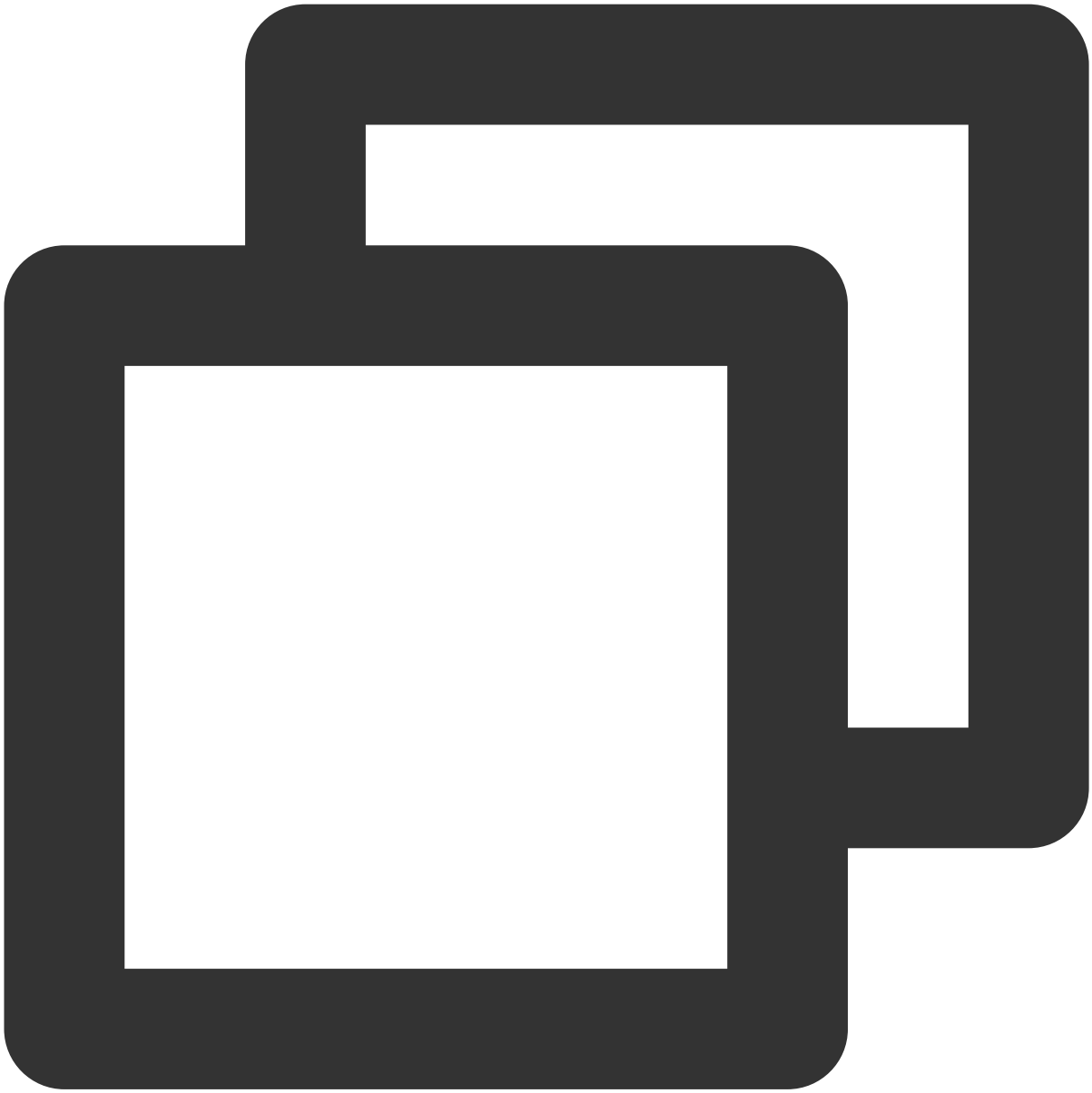
获得TRTC实例对象。



```
virtual void* getTRTCCloud() = 0;
```

getDeviceManager

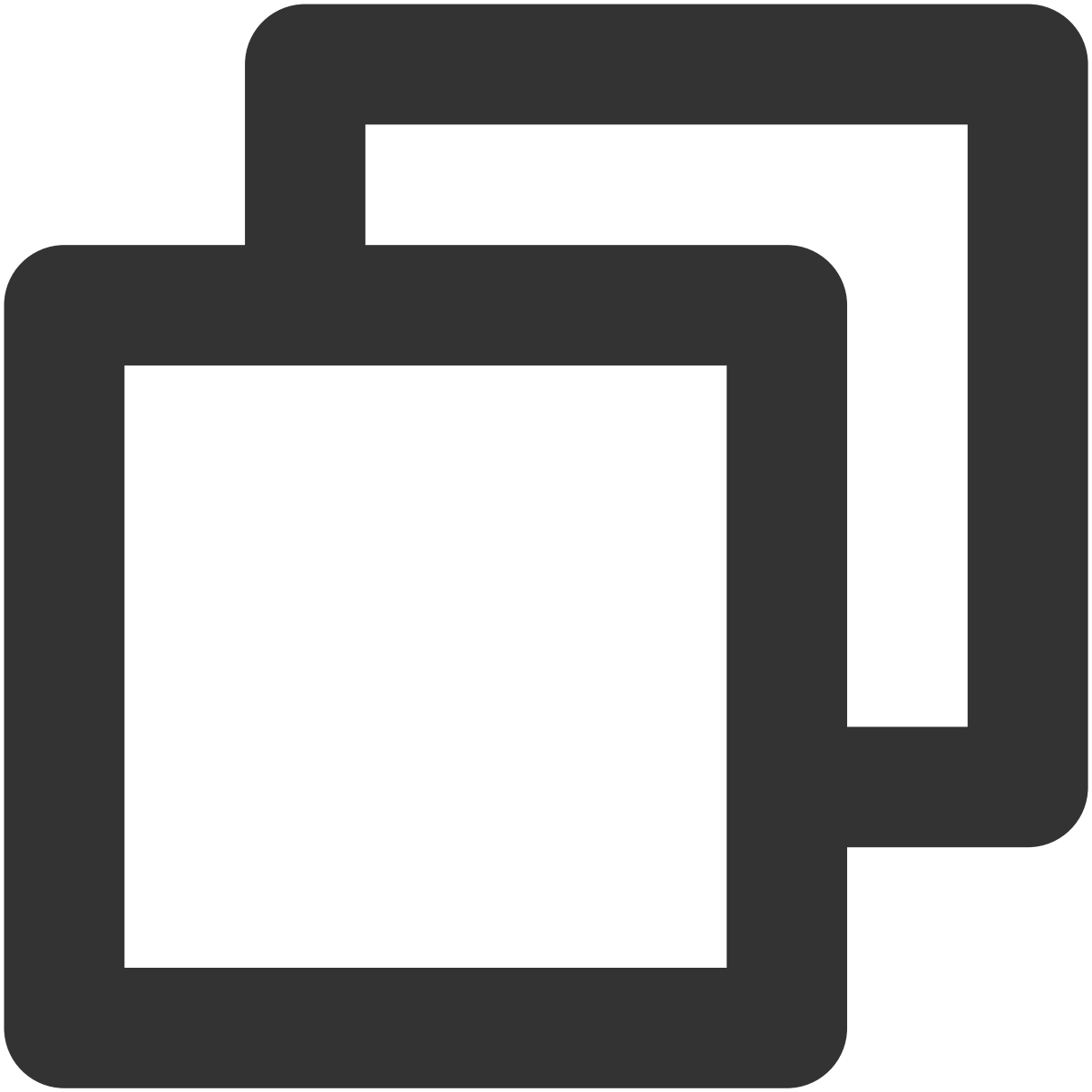
获得设备管理对象。



```
virtual liteav::ITXDeviceManager* getDeviceManager() = 0;
```

getAudioEffectManager

获得音效管理对象。



```
virtual liteav::ITXAudioEffectManager* getAudioEffectManager() = 0;
```

TUIRoomObserver

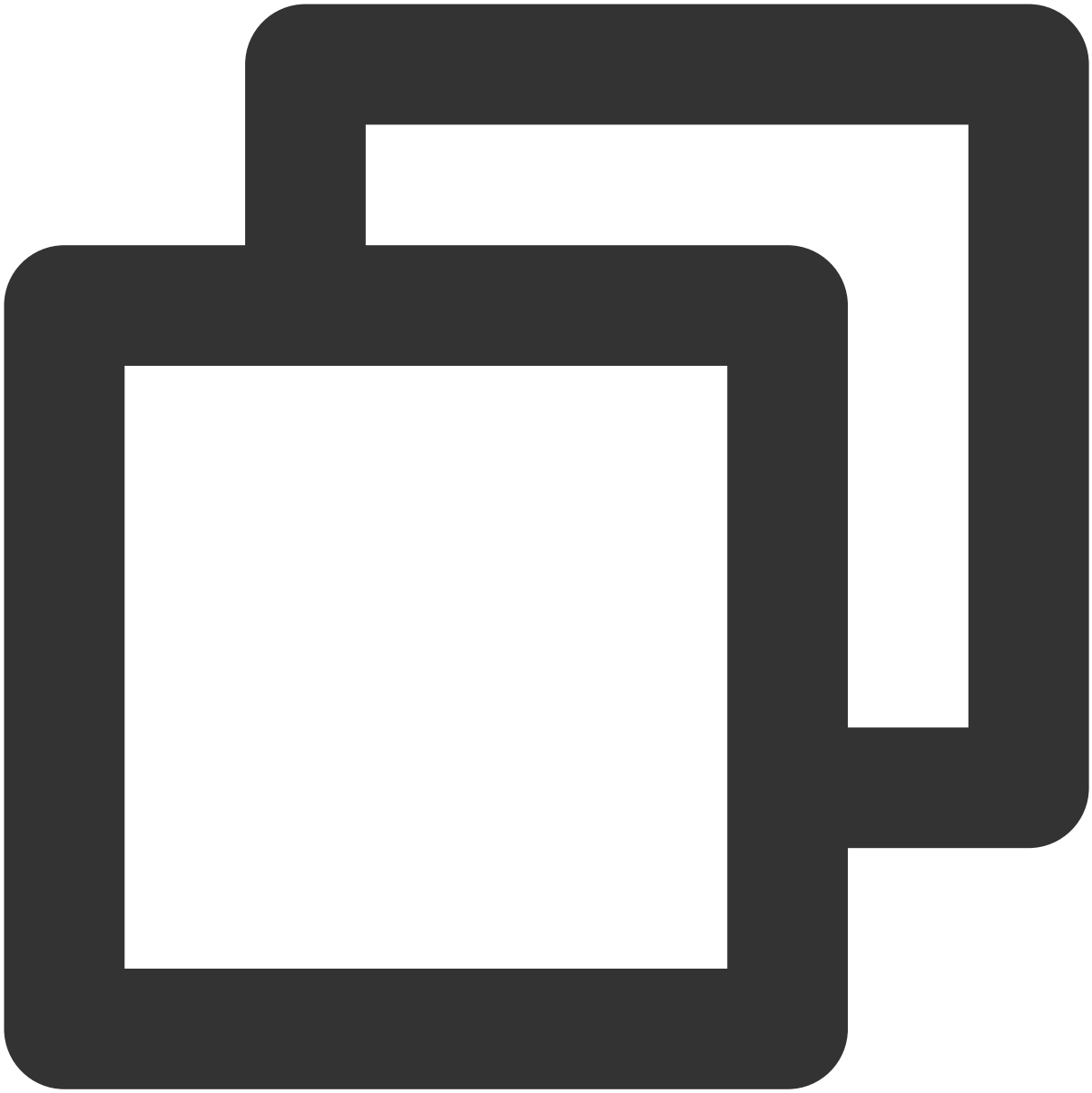
最近更新时间：2023-10-30 10:55:55

TUIRoomObserver 回调事件详情

TUIRoomObserver 类是 TUIRoomEngine 对应的回调事件类，您可以通过此回调接口，监听自己感兴趣的回调事件。

onError

错误信息事件。



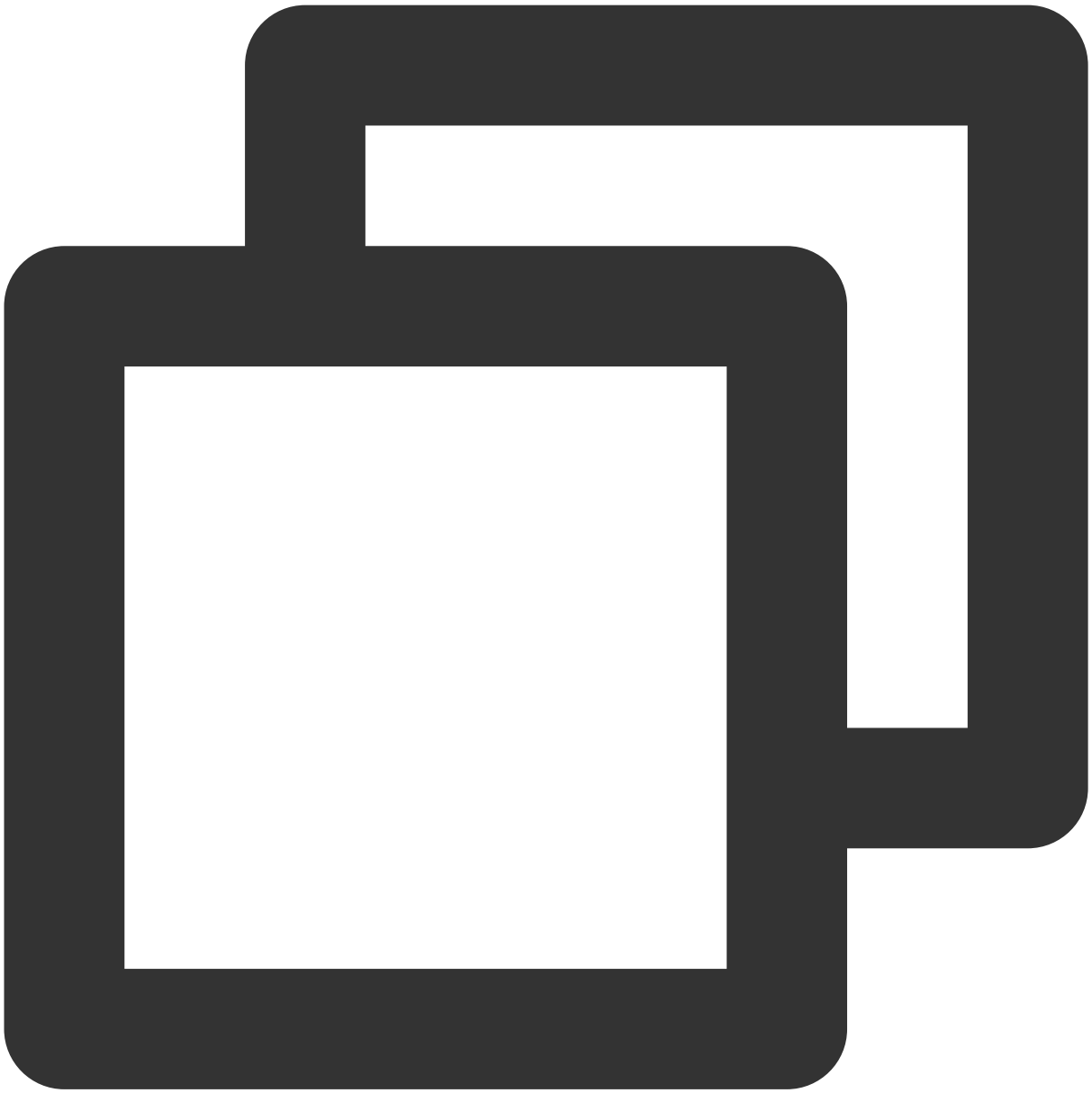
```
virtual void onError(TUIError errorCode, const char* message) = 0;
```

参数如下表所示：

参数	类型	含义
errorCode	TUIError	错误码
message	const char*	错误信息

onKickedOffLine

用户被踢下线事件。



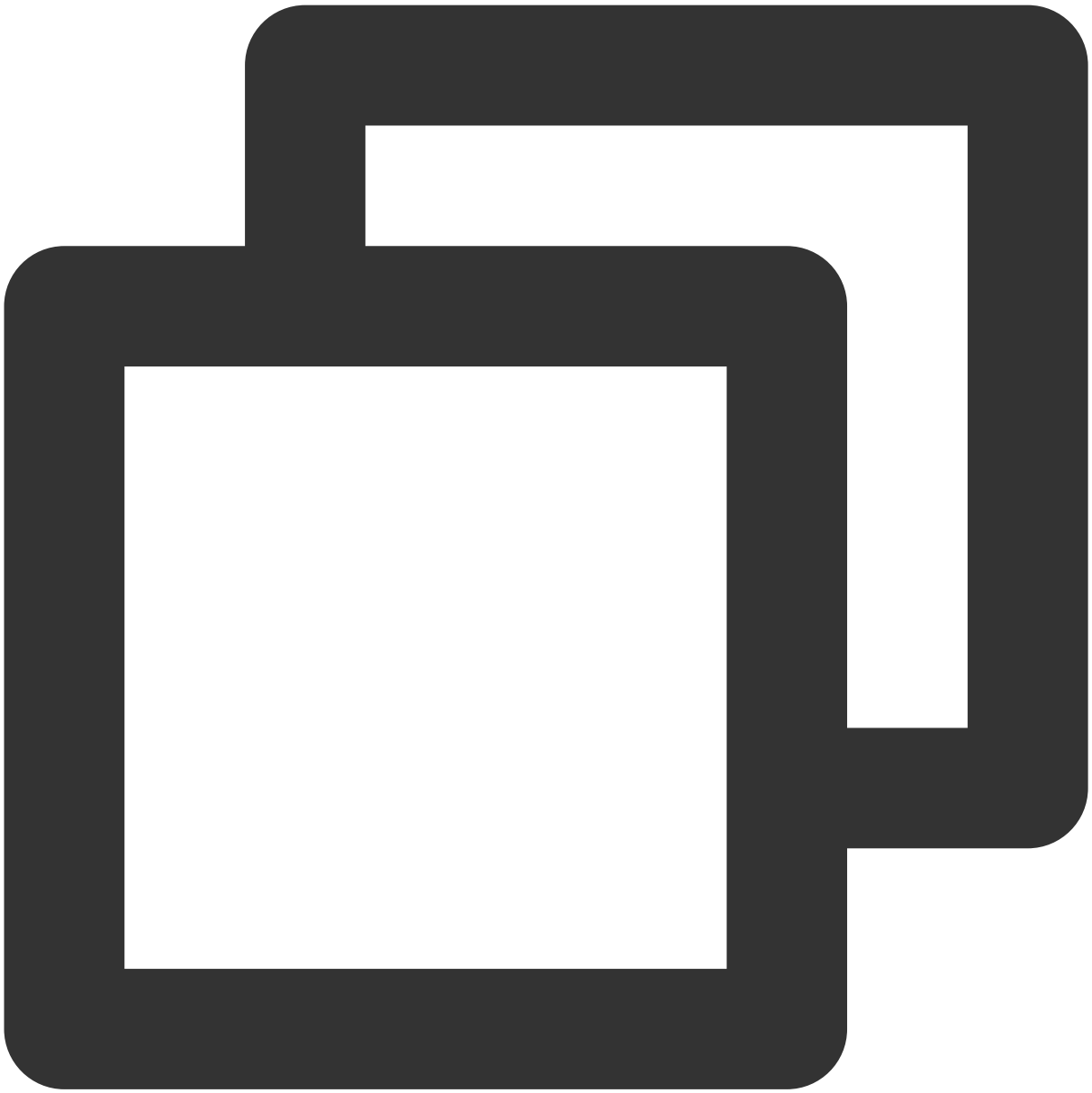
```
virtual void onKickedOffLine(const char* message) = 0;
```

参数如下表所示：

参数	类型	含义
message	const char*	被踢下线的描述

onUserSigExpired

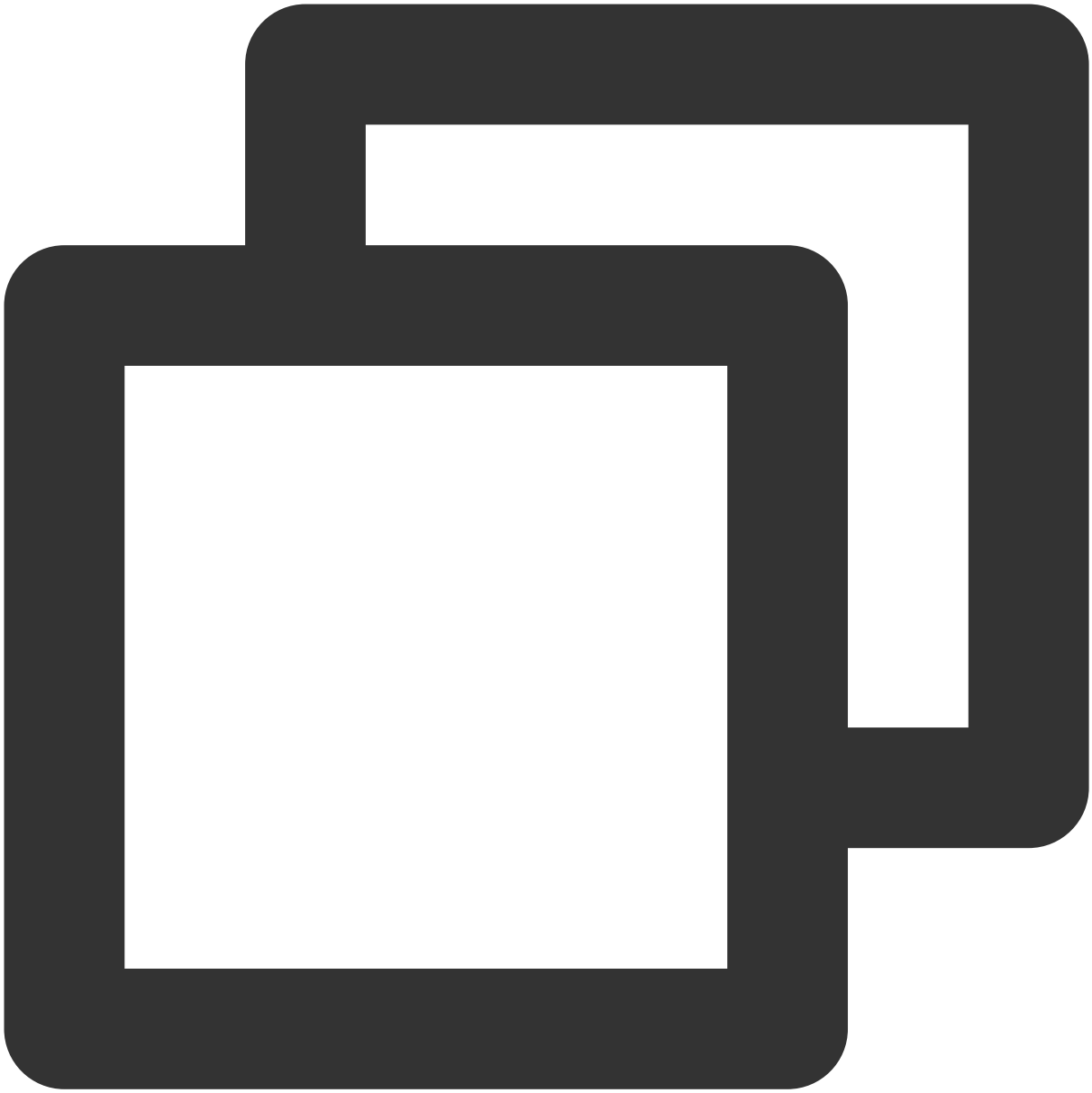
用户凭证超时事件。



```
virtual void onUserSigExpired() = 0;
```

onRoomNameChanged

房间名称更改事件。



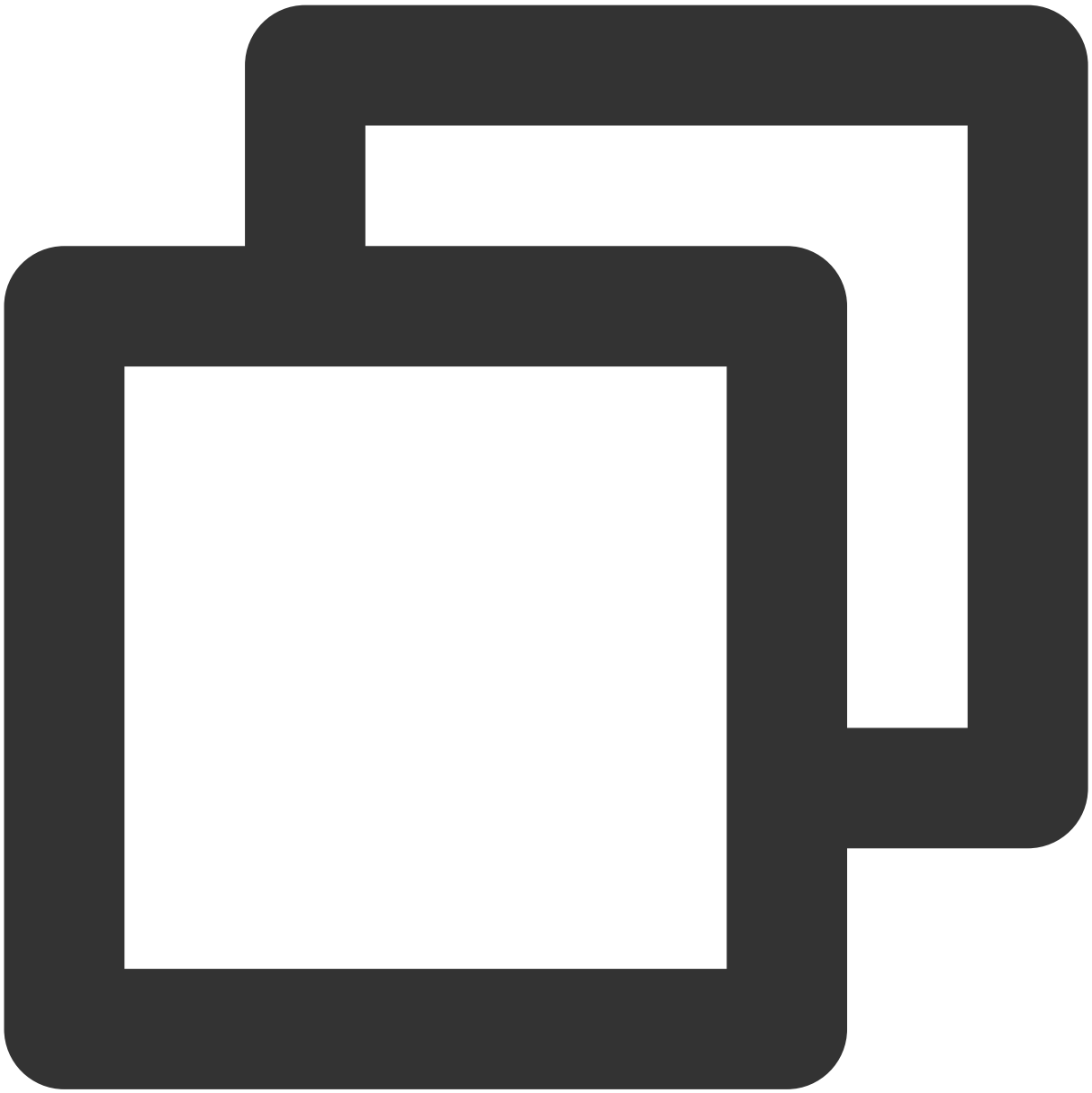
```
virtual void onRoomNameChanged(const char* roomId, const char* roomName) = 0;
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
roomName	const char*	房间名称

onAllUserMicrophoneDisableChanged

房间内所有用户麦克风被禁用事件。



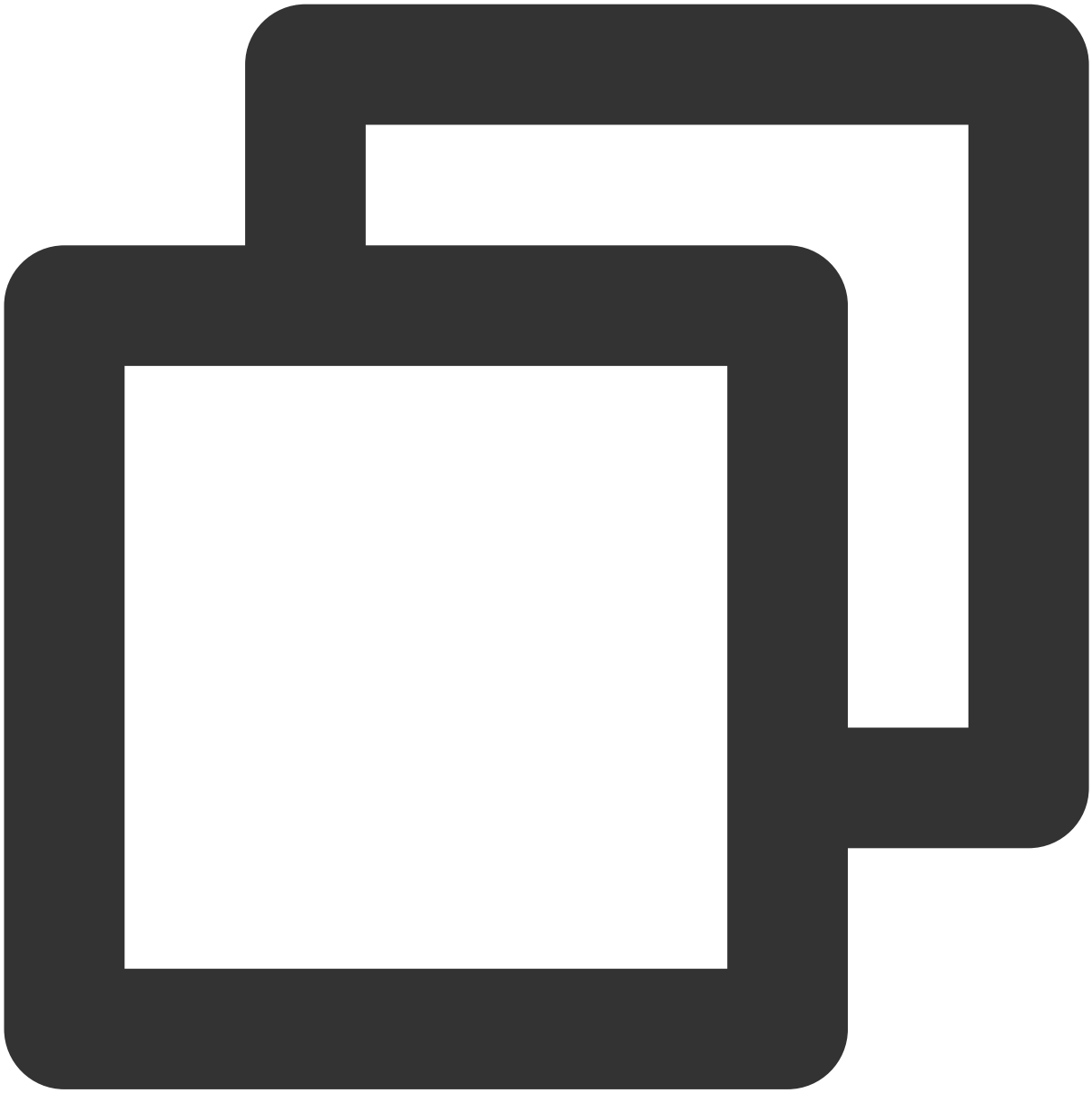
```
virtual void onAllUserMicrophoneDisableChanged(const char* roomId, bool isDisable)
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
isDisable	bool	是否被禁用

onAllUserCameraDisableChanged

房间内所有用户摄像头被禁用事件。



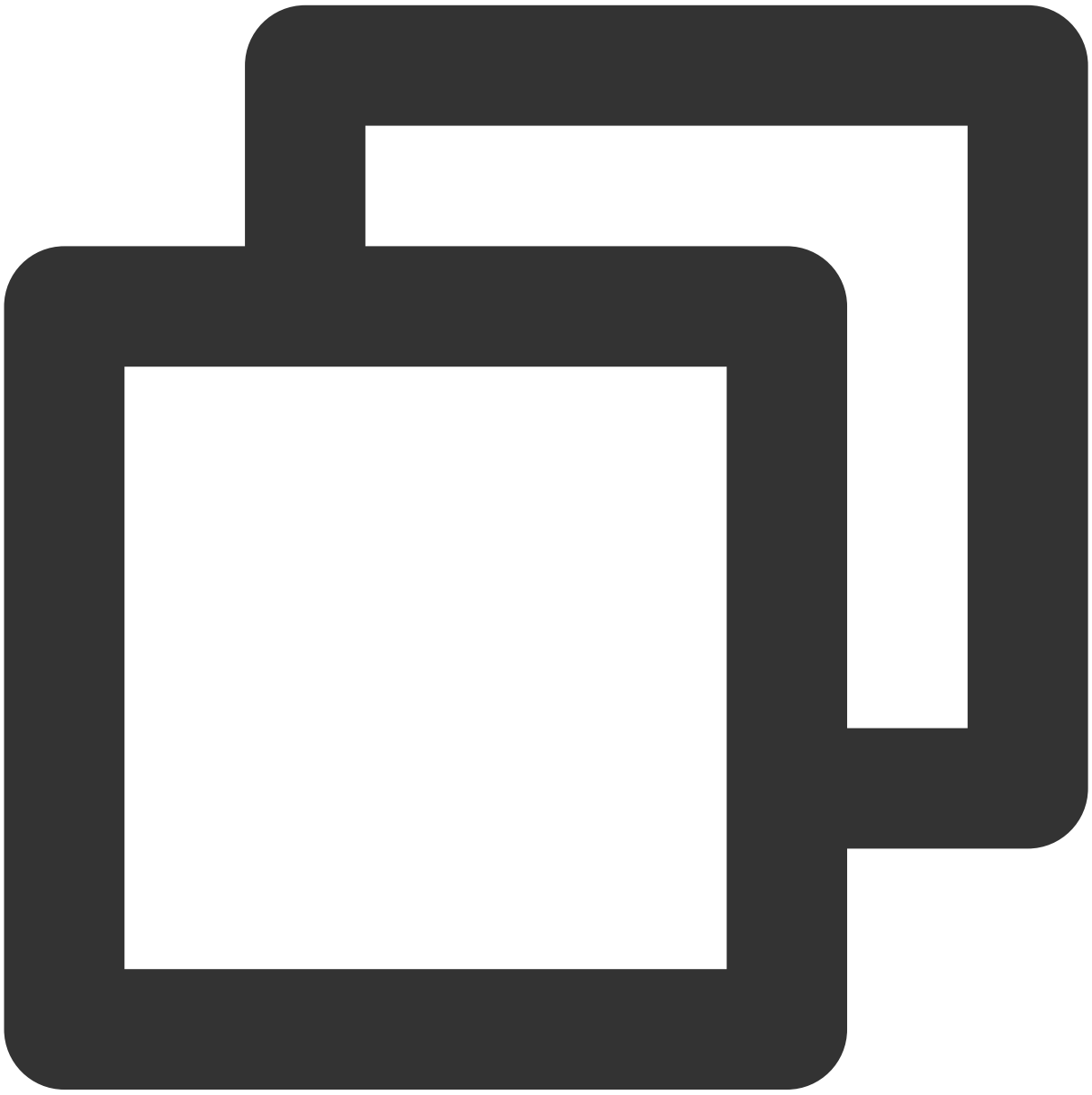
```
virtual void onAllUserCameraDisableChanged(const char* roomId, bool isDisable) = 0;
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
isDisable	bool	是否被禁用

onSendMessageForAllUserDisableChanged

房间内用户发送文本消息被禁用事件。



```
virtual void onSendMessageForAllUserDisableChanged(const char* roomId, bool isDisab
```

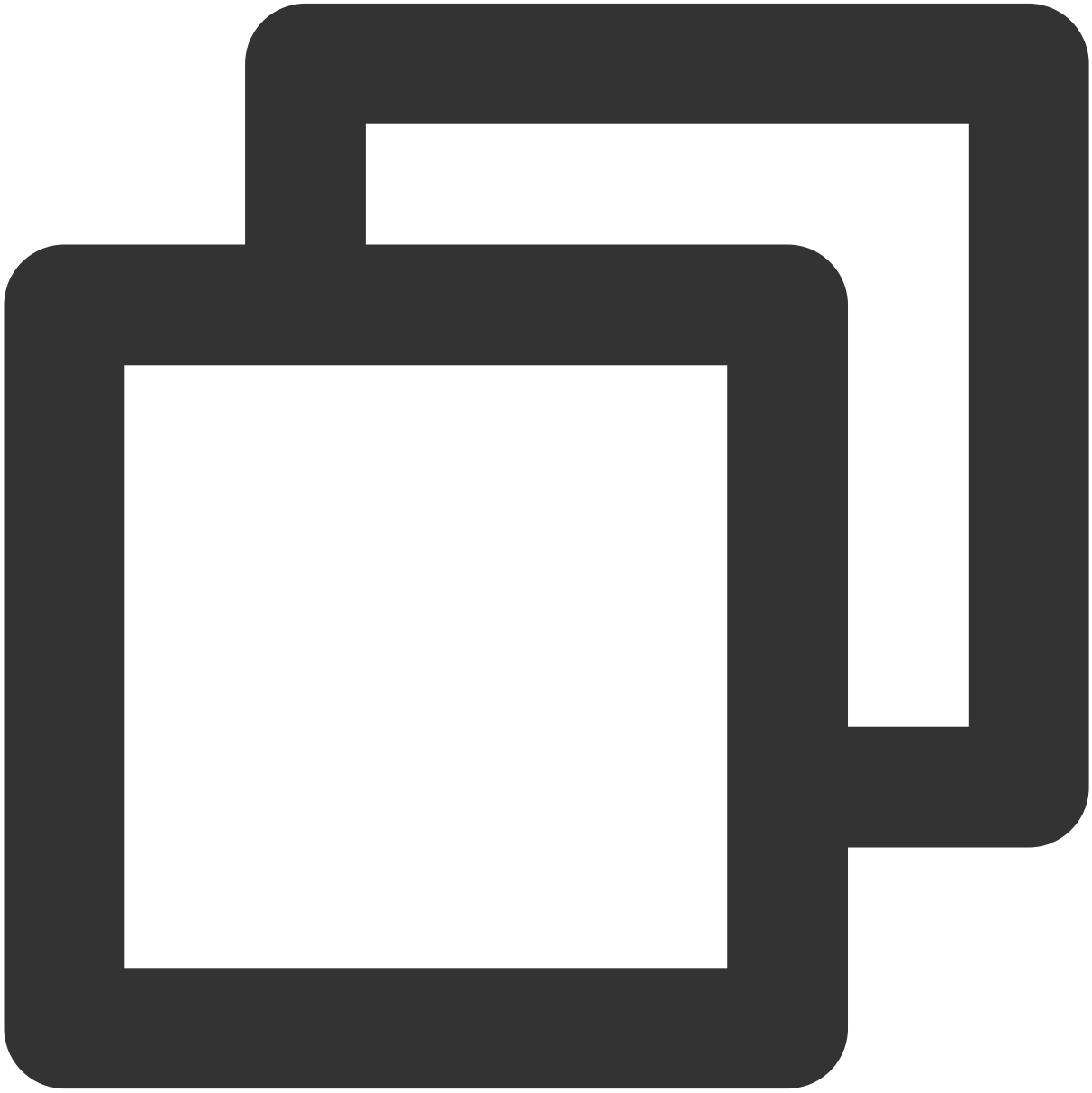
参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID

isDisable	bool	是否被禁用

onRoomDismissed

房间被解散事件。



```
virtual void onRoomDismissed(const char* roomId) = 0;
```

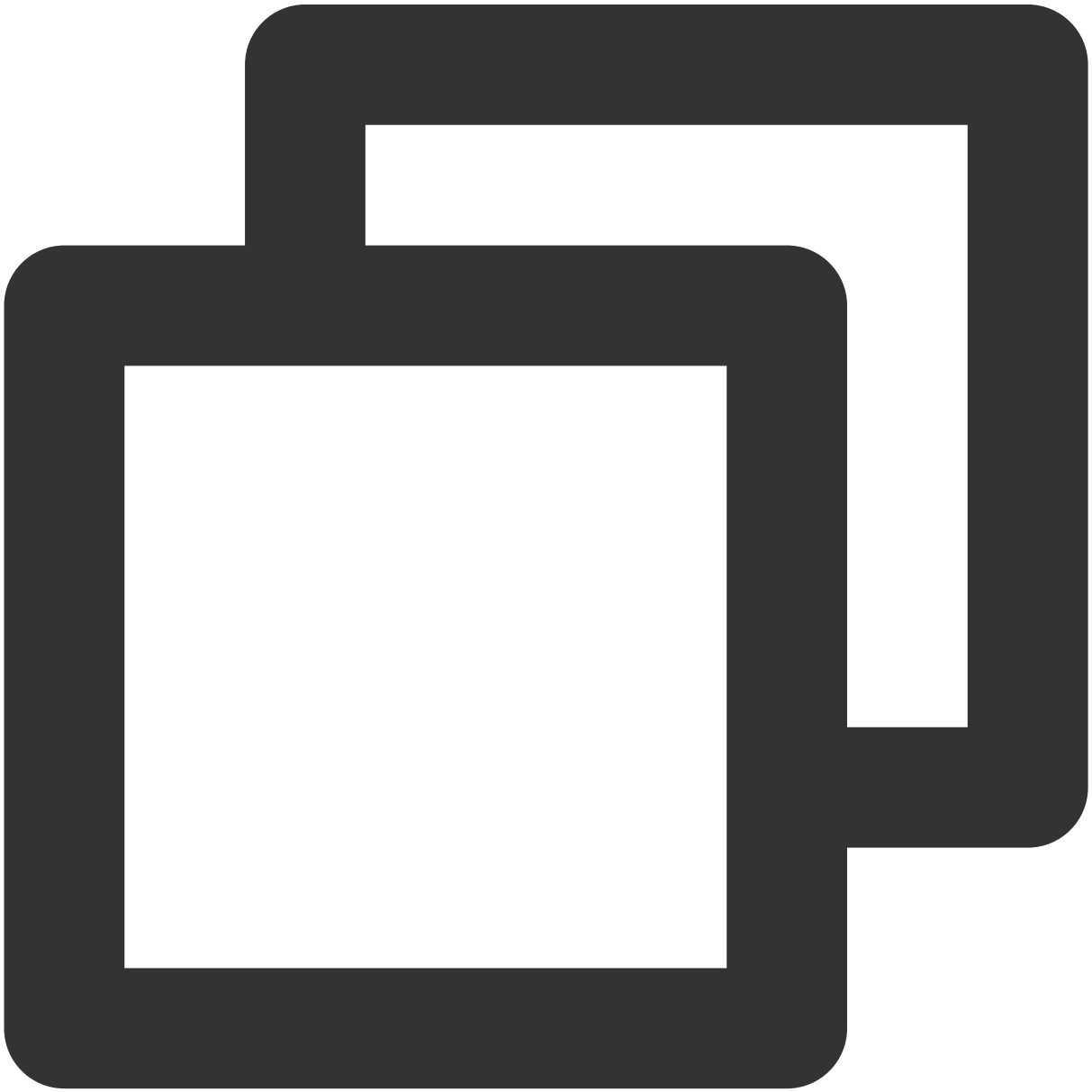
参数如下表所示：

参数	类型	含义

roomId	const char*	房间 ID

onKickedOutOfRoom

被踢出房间事件。



```
virtual void onKickedOutOfRoom(const char* roomId, const char* message) = 0;
```

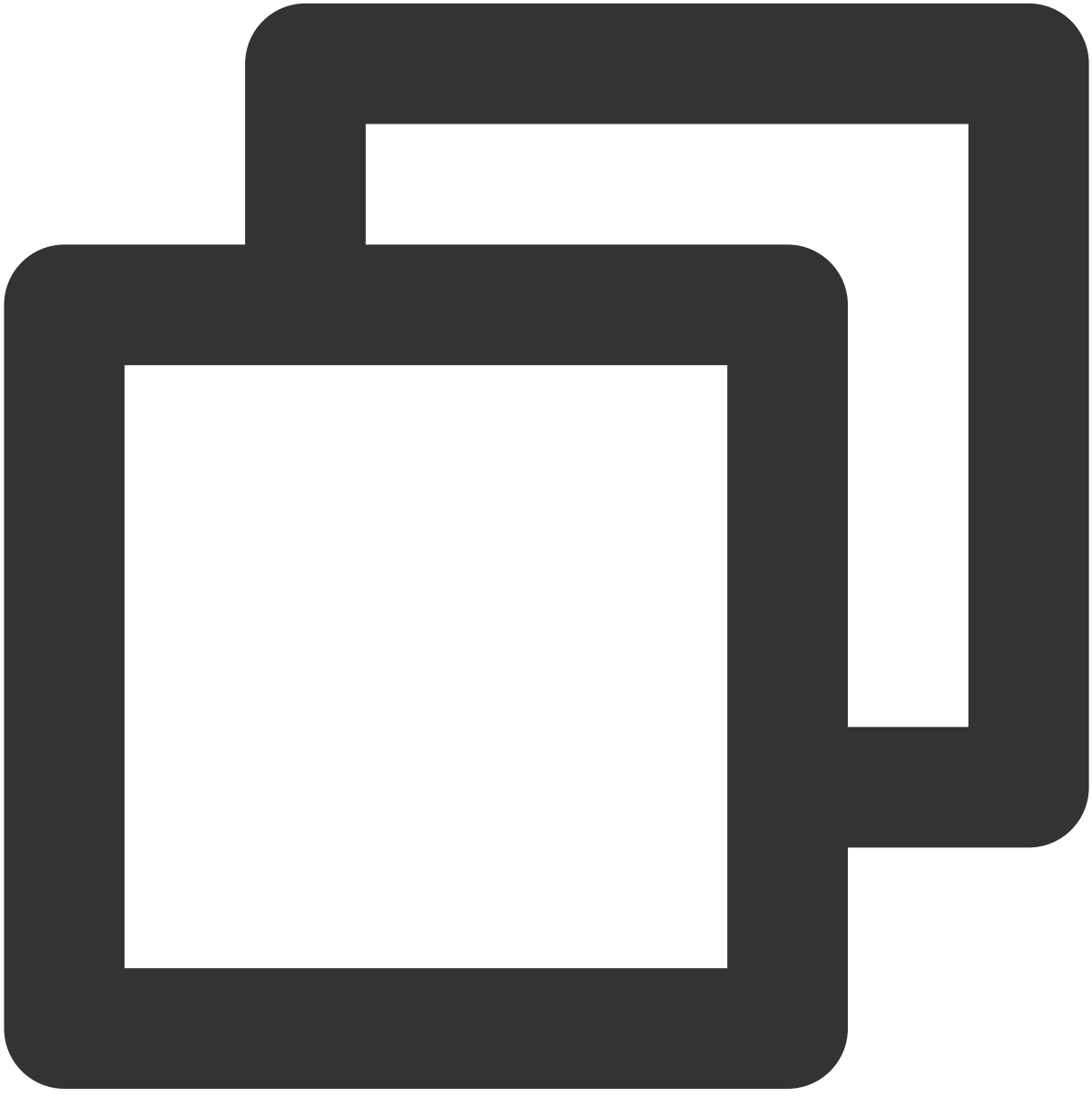
参数如下表所示：

参数	类型	含义

roomId	const char*	房间 ID
message	const char*	被踢出的描述

onRoomSpeechModeChanged

房间发言模式更改事件。



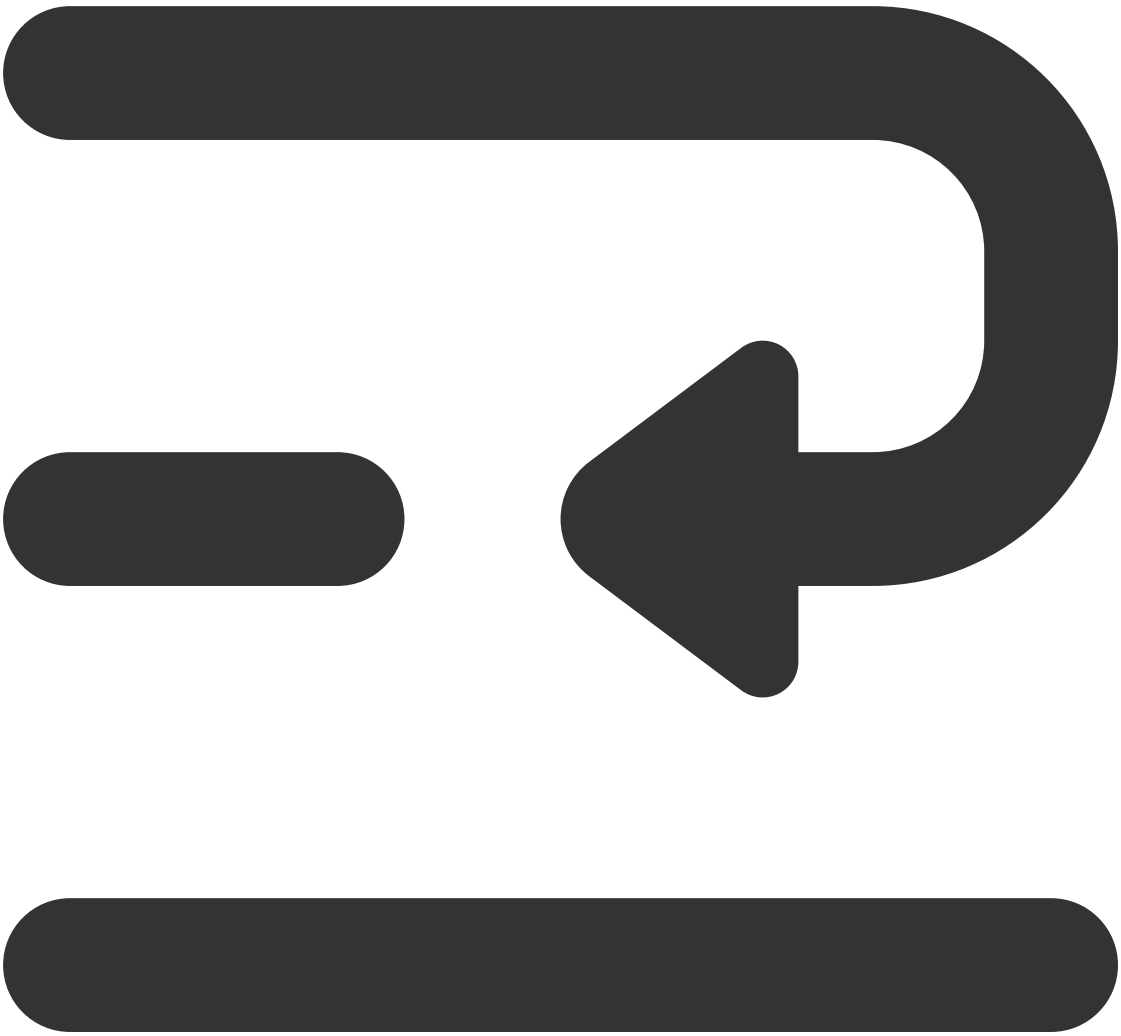
```
virtual void onRoomSpeechModeChanged(const char* roomId, TUISpeechMode speechMode)
```

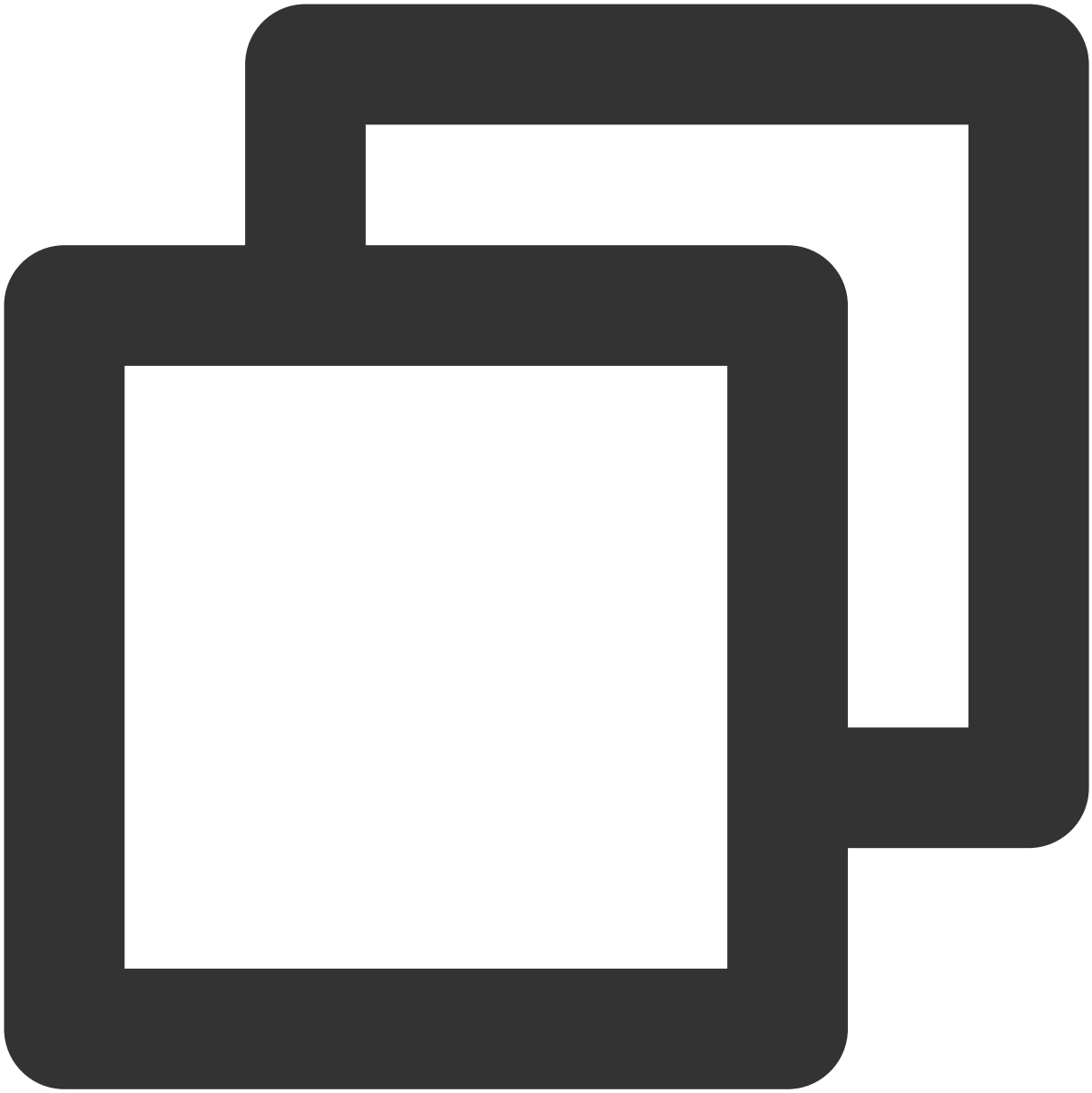
参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
speechMode	TUISpeechMode	发言模式

onRemoteUserEnterRoom

远端用户进房事件。





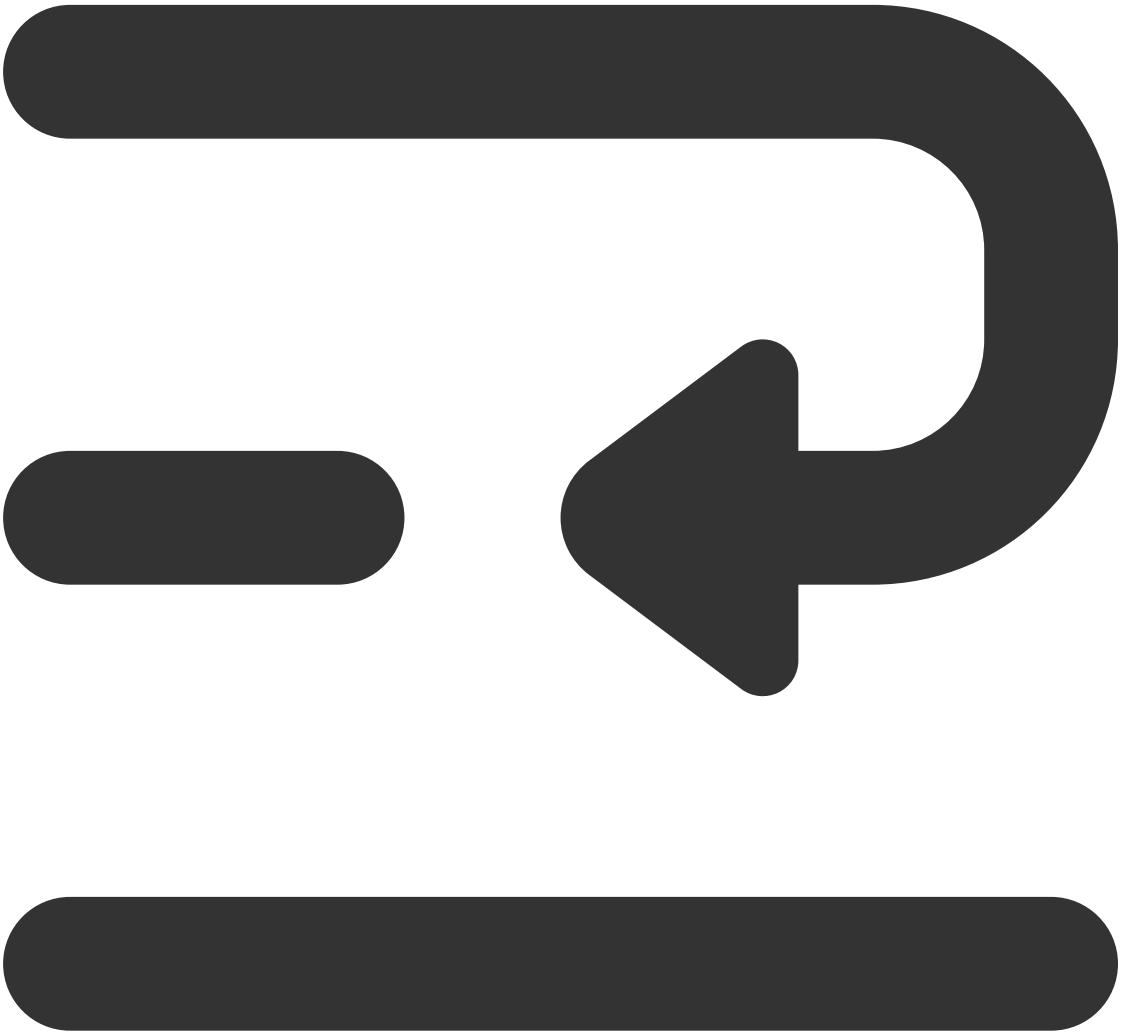
```
virtual void onRemoteUserEnterRoom(const char* roomId, const TUIUserInfo& userInfo)
```

参数如下表所示：

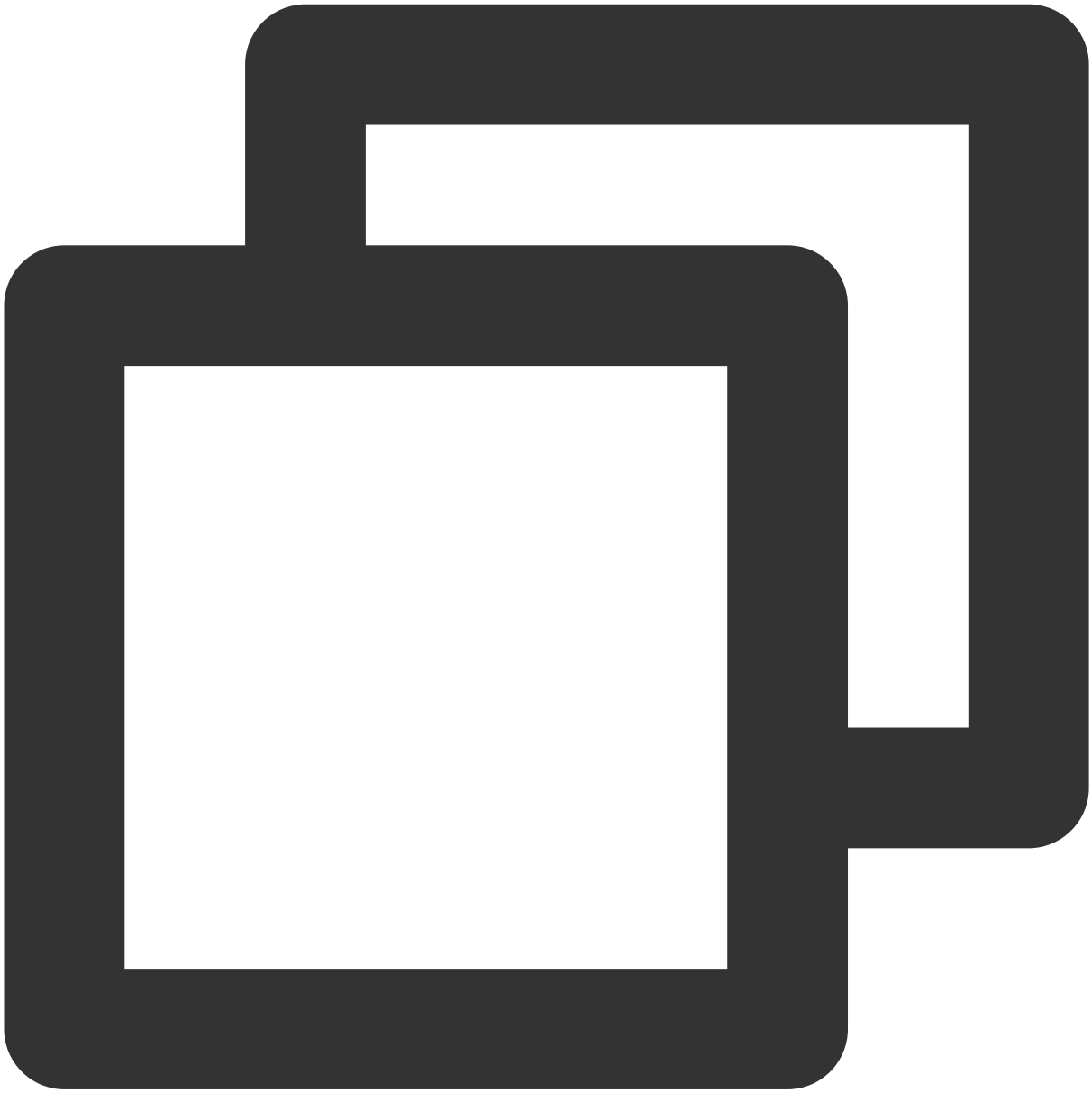
参数	类型	含义
roomId	const char*	房间 ID
userInfo	const TUIUserInfo&	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件。



版权所有：腾讯云计算（北京）有限责任公司



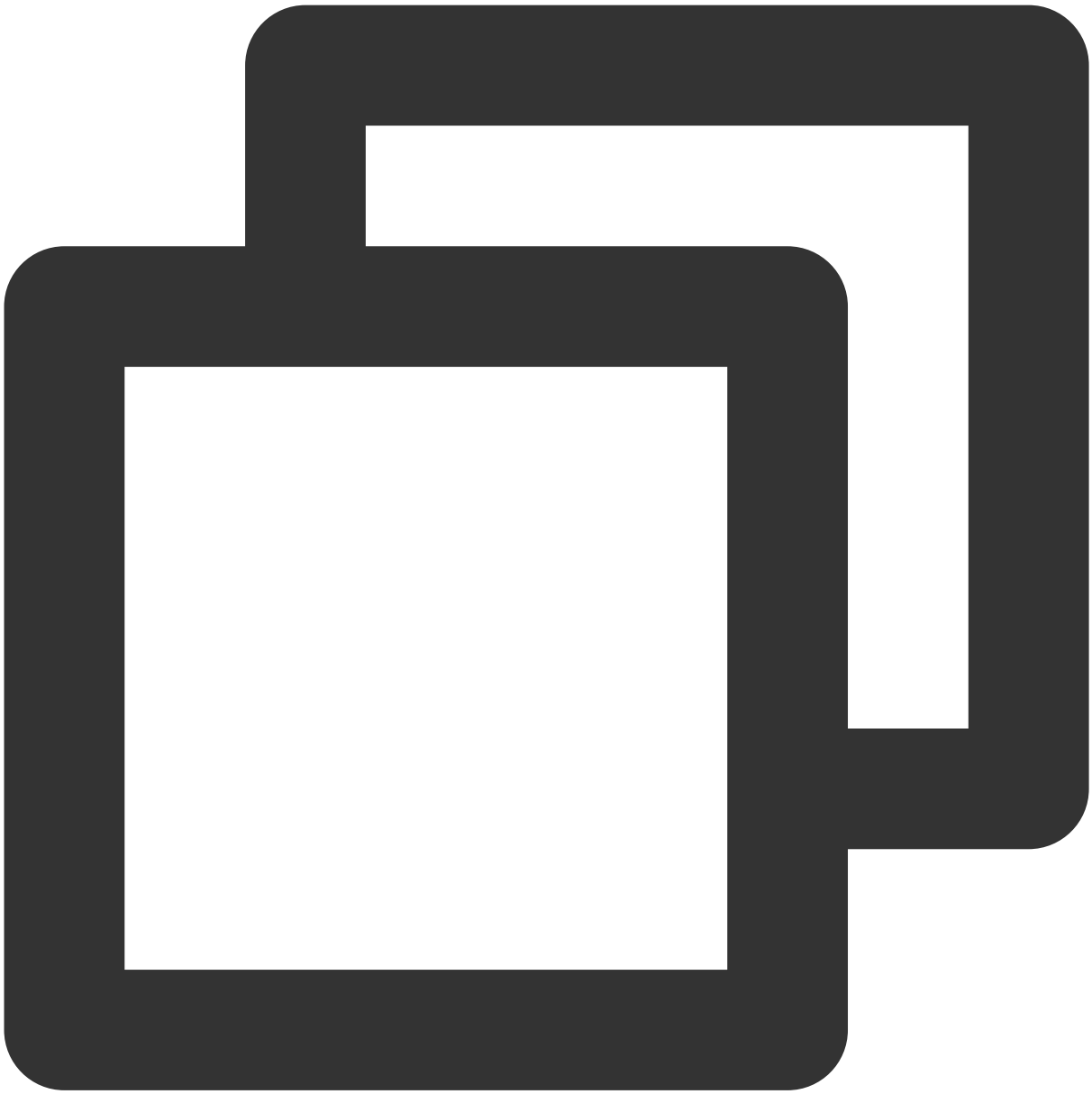
```
virtual void OnRemoteUserLeaveRoom(const char* roomId, const TUIUserInfo& userInfo)
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
userInfo	const TUIUserInfo&	用户信息

onUserRoleChanged

用户角色发生变化事件。



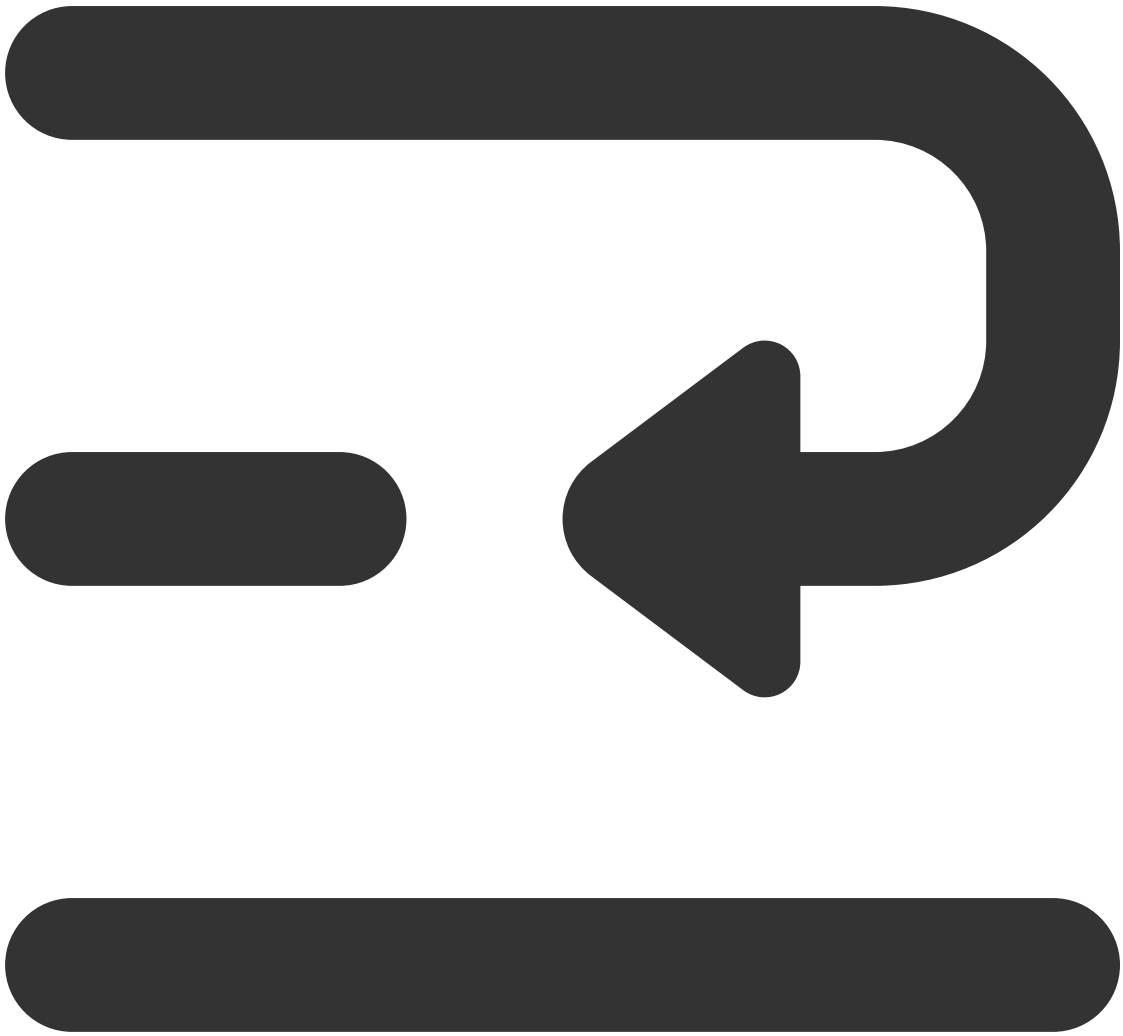
```
virtual void onUserRoleChanged(const char* userId, const TUIRole& userRole) = 0;
```

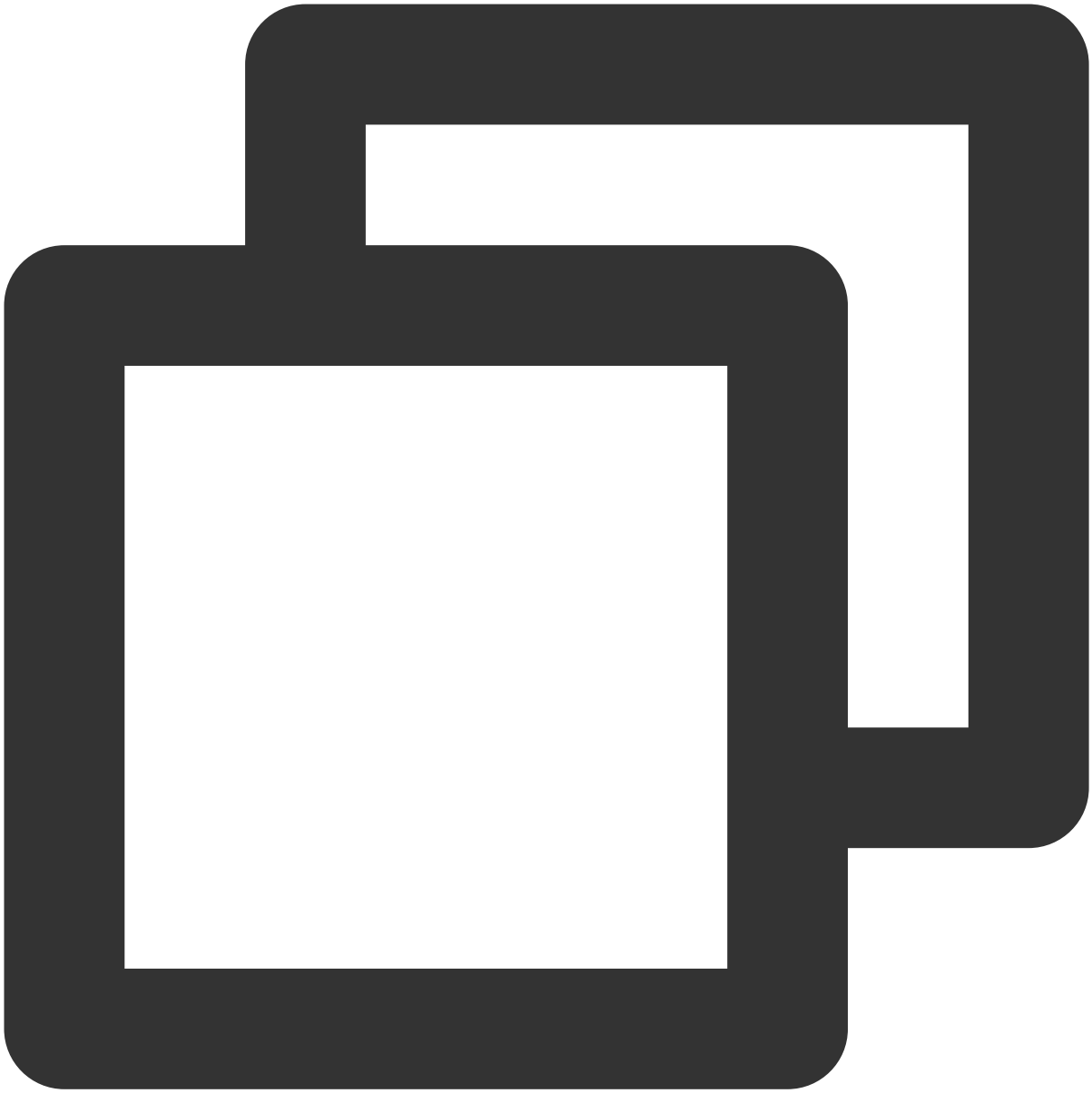
参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
userRole	const TUIRole&	用户角色

onUserVideoStateChanged

用户视频状态发生变化事件。





```
virtual void onUserVideoStateChanged(const char* userId, TUIVideoStreamType streamT
```

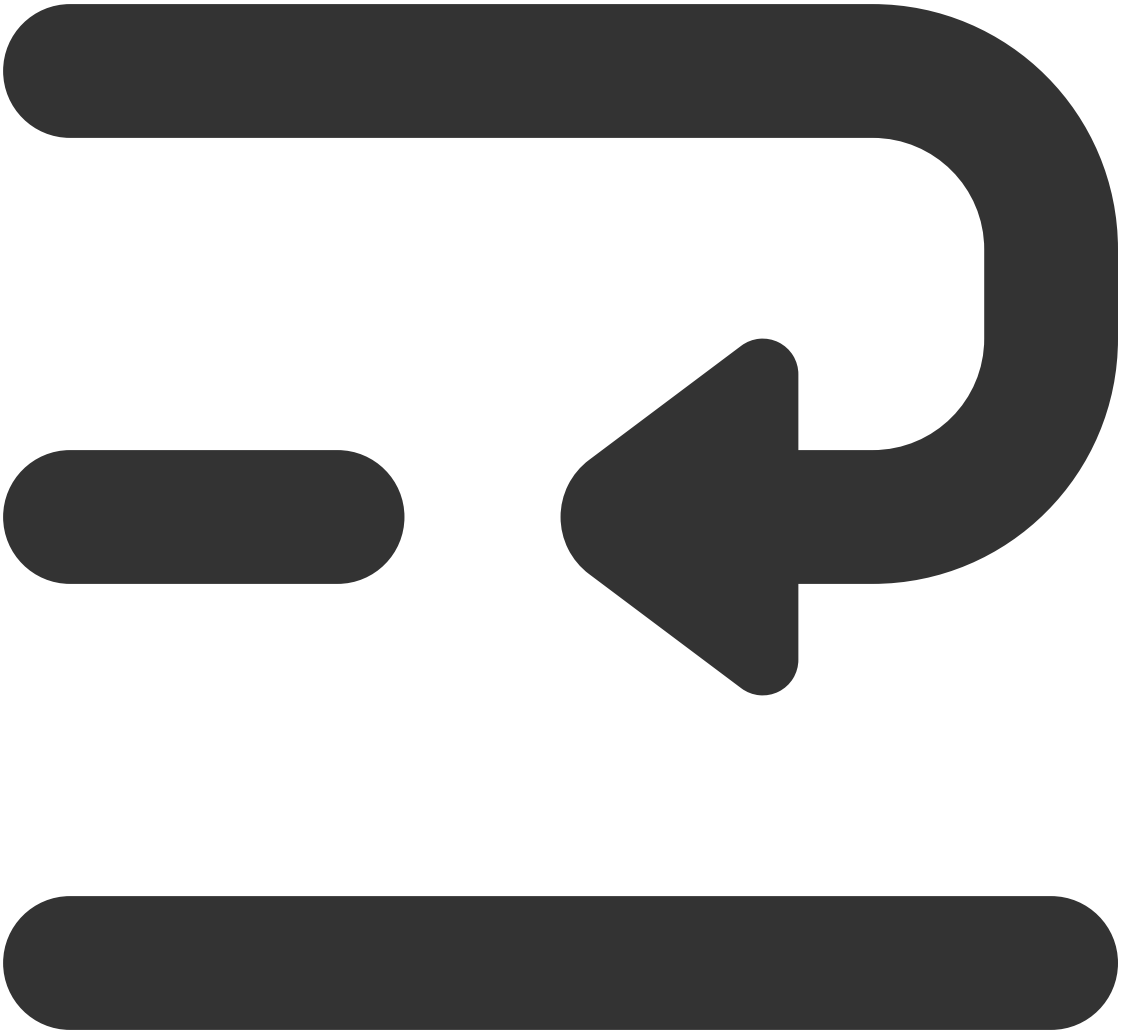
参数如下表所示：

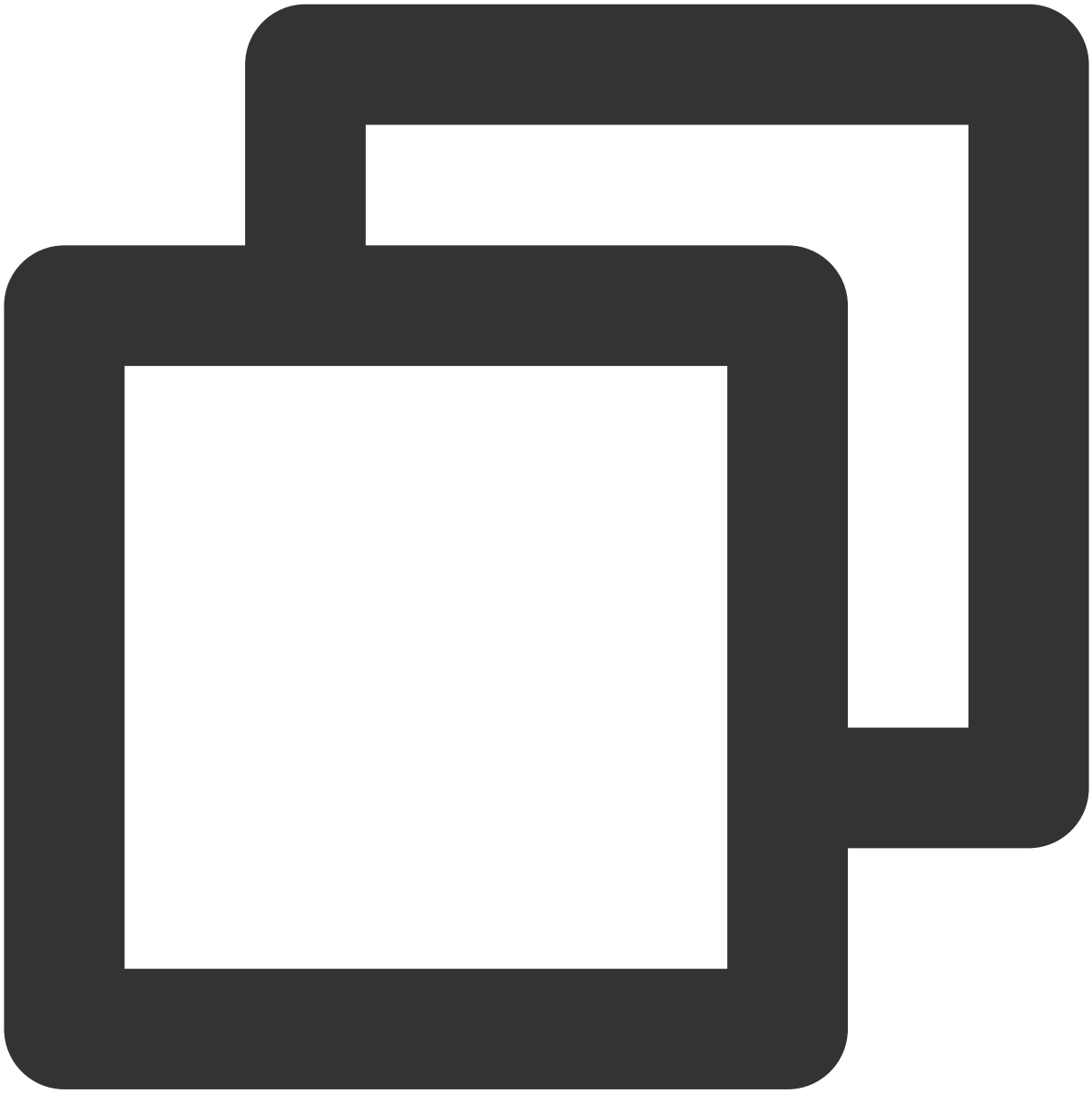
参数	类型	含义
userId	const char*	用户 ID
streamType	TUIVideoStreamType	视频流类型
hasVideo	bool	是否有视频流

reason	TUIChangeReason	视频流发生变化原因

onUserAudioStateChanged

用户音频状态发生变化事件。





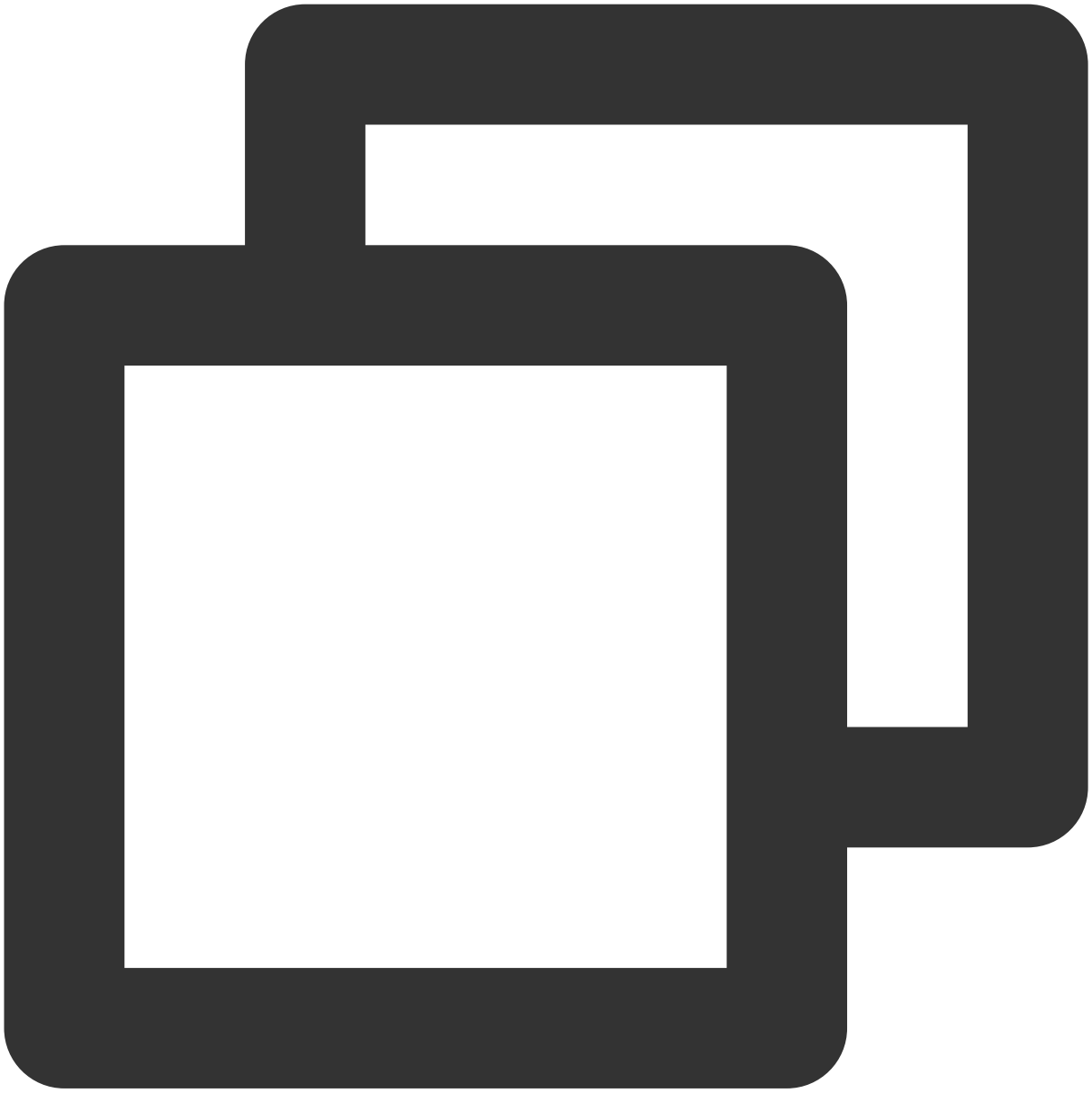
```
virtual void onUserAudioStateChanged(const char* userId, bool hasAudio, TUIChangeRe
```

参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID
hasAudio	bool	是否有音频流
reason	TUIChangeReason	音频流发生变化原因

onUserVoiceVolumeChanged

用户音量变化事件。



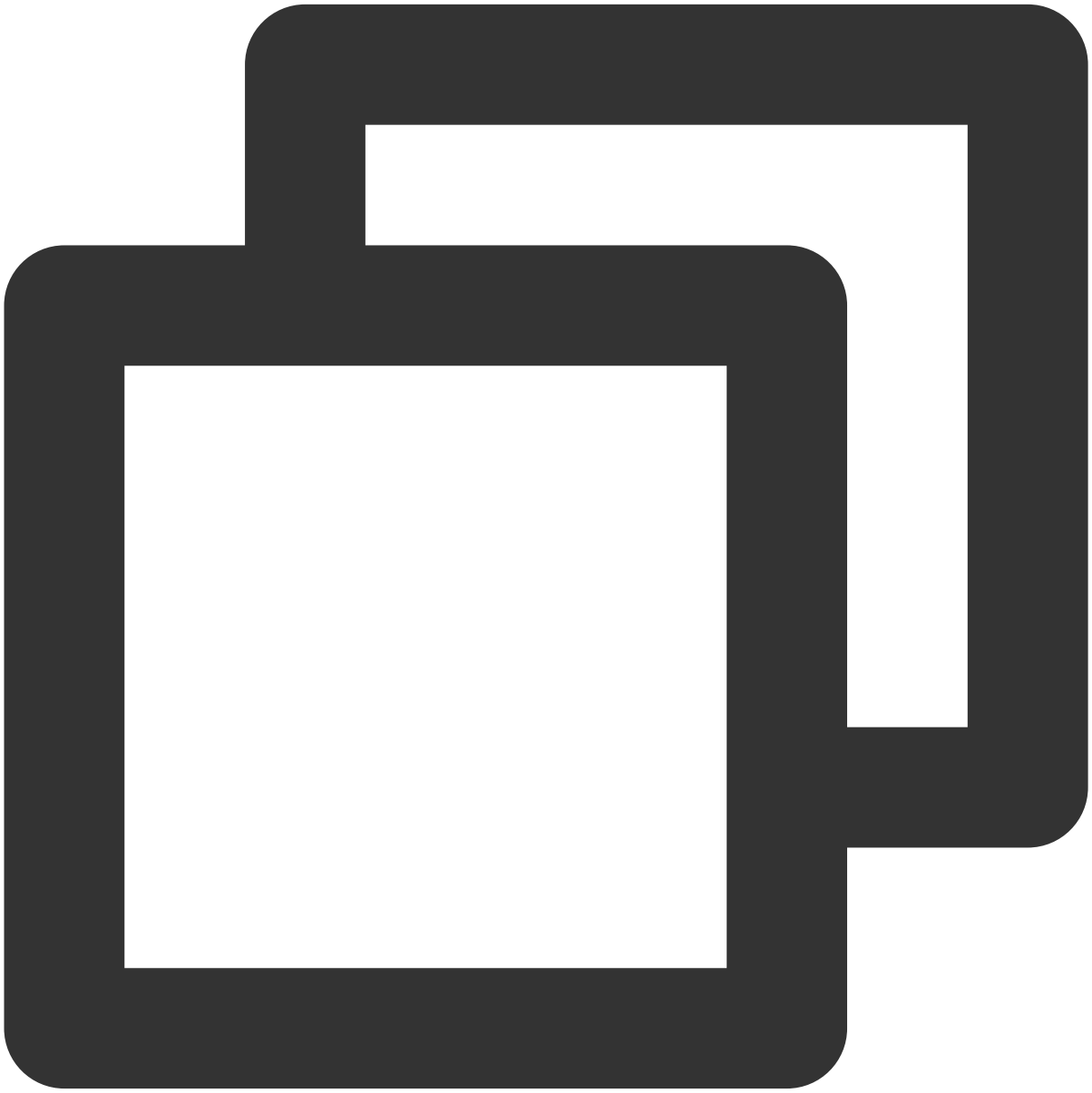
```
virtual void onUserVoiceVolumeChanged(TUIMap<const char*, int>* volumeMap) = 0;
```

参数如下表所示：

参数	类型	含义
volumeMap	TUIMap<const char*, int>*	用户音量 map

onSendMessageForUserDisableChanged

用户文本消息发送能力发生变化事件。



```
virtual void onSendMessageForUserDisableChanged(const char* roomId, const char* use
```

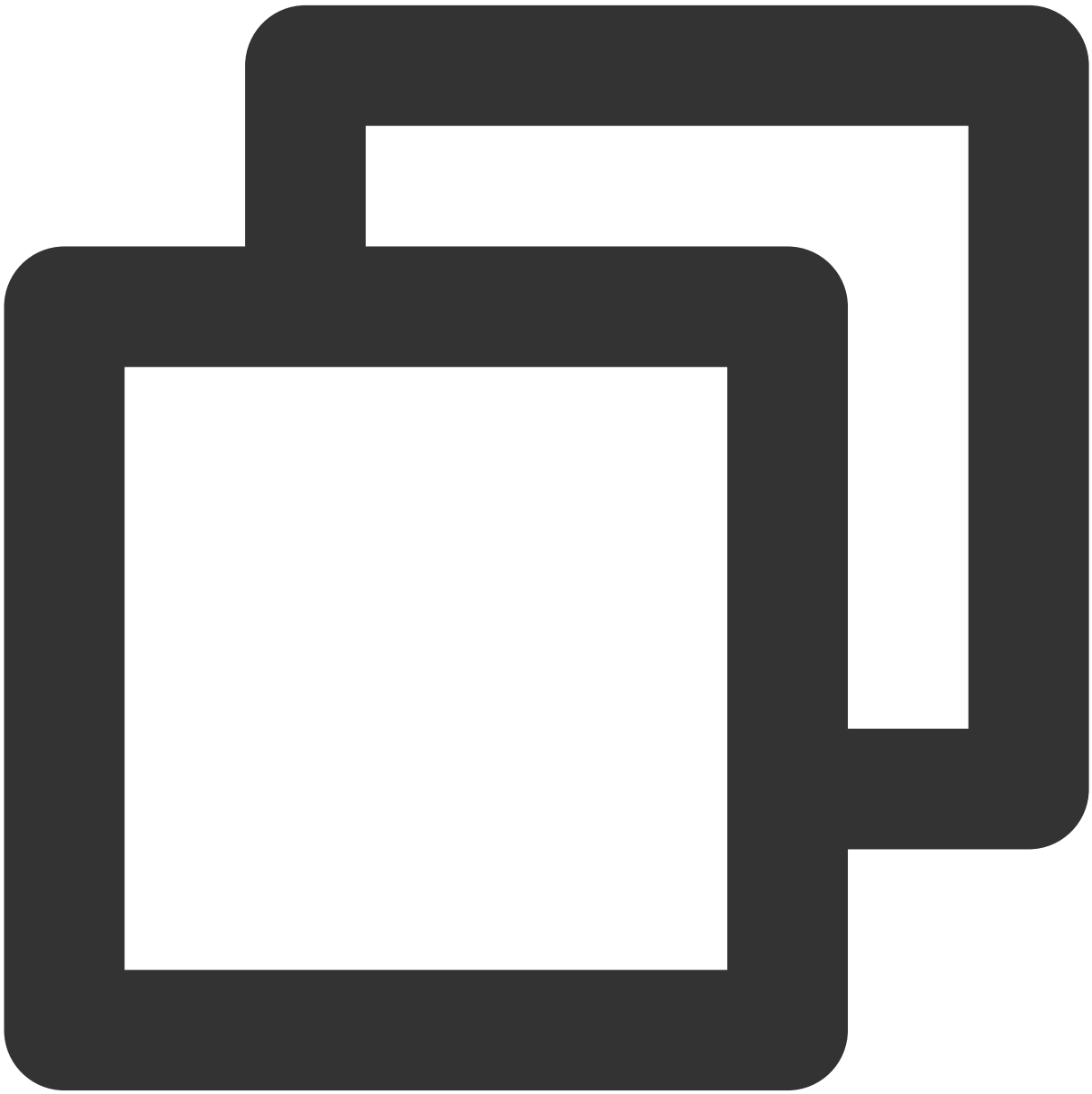
参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
userId	const char*	用户 ID

isDisable	bool	是否被禁止发送消息，true : 用户被禁止发送文本和自定义消息；false : 用户被解除禁止，可以发送文本和自定义消息

onUserNetworkQualityChanged

用户网络状态变化事件。



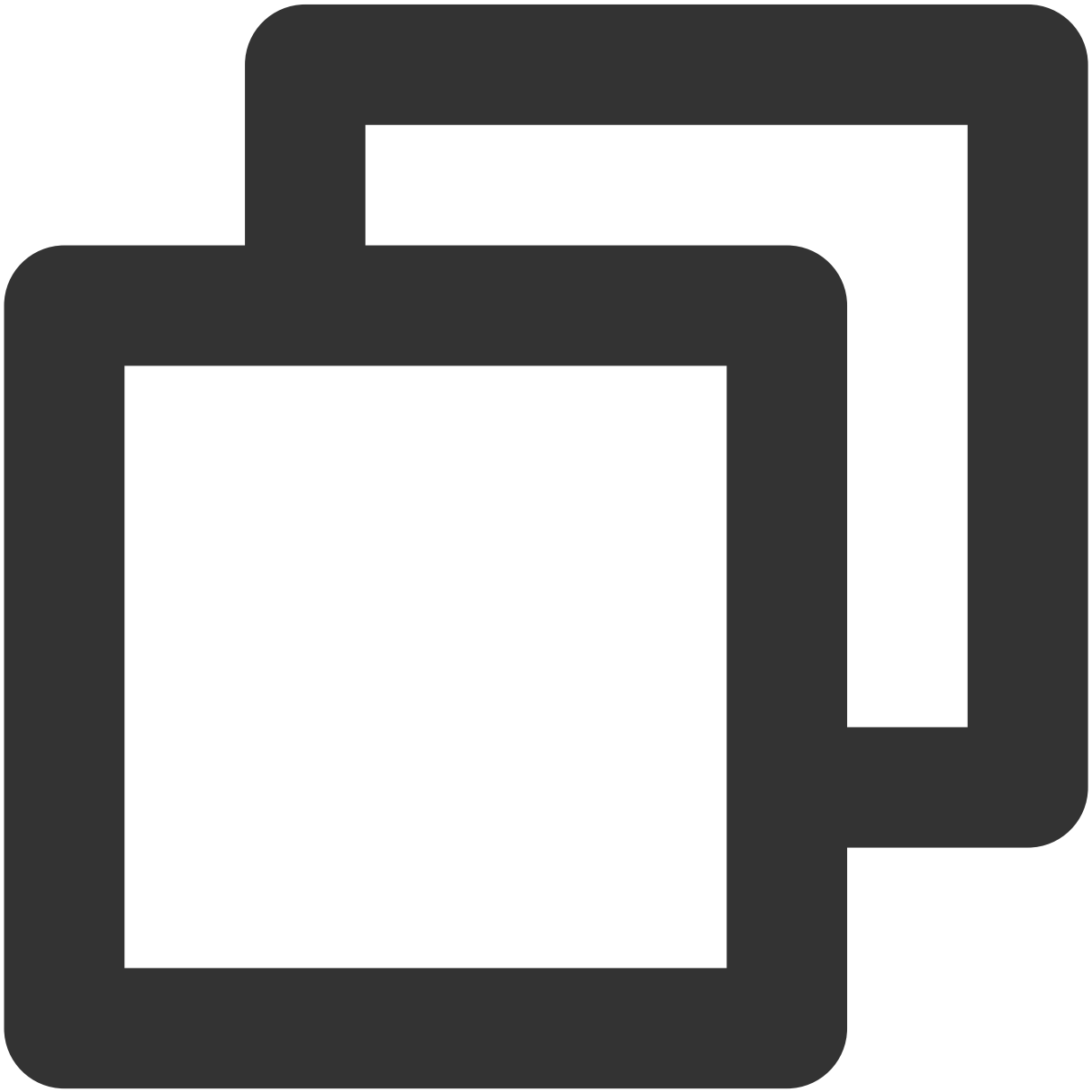
```
virtual void onUserNetworkQualityChanged(TUIList<TUINetwork>* networkList) = 0;
```

参数如下表所示：

参数	类型	含义
networkList	TUIList<TUINetwork>*	用户网络状态 list

onUserScreenCaptureStopped

屏幕分享停止的事件回调。



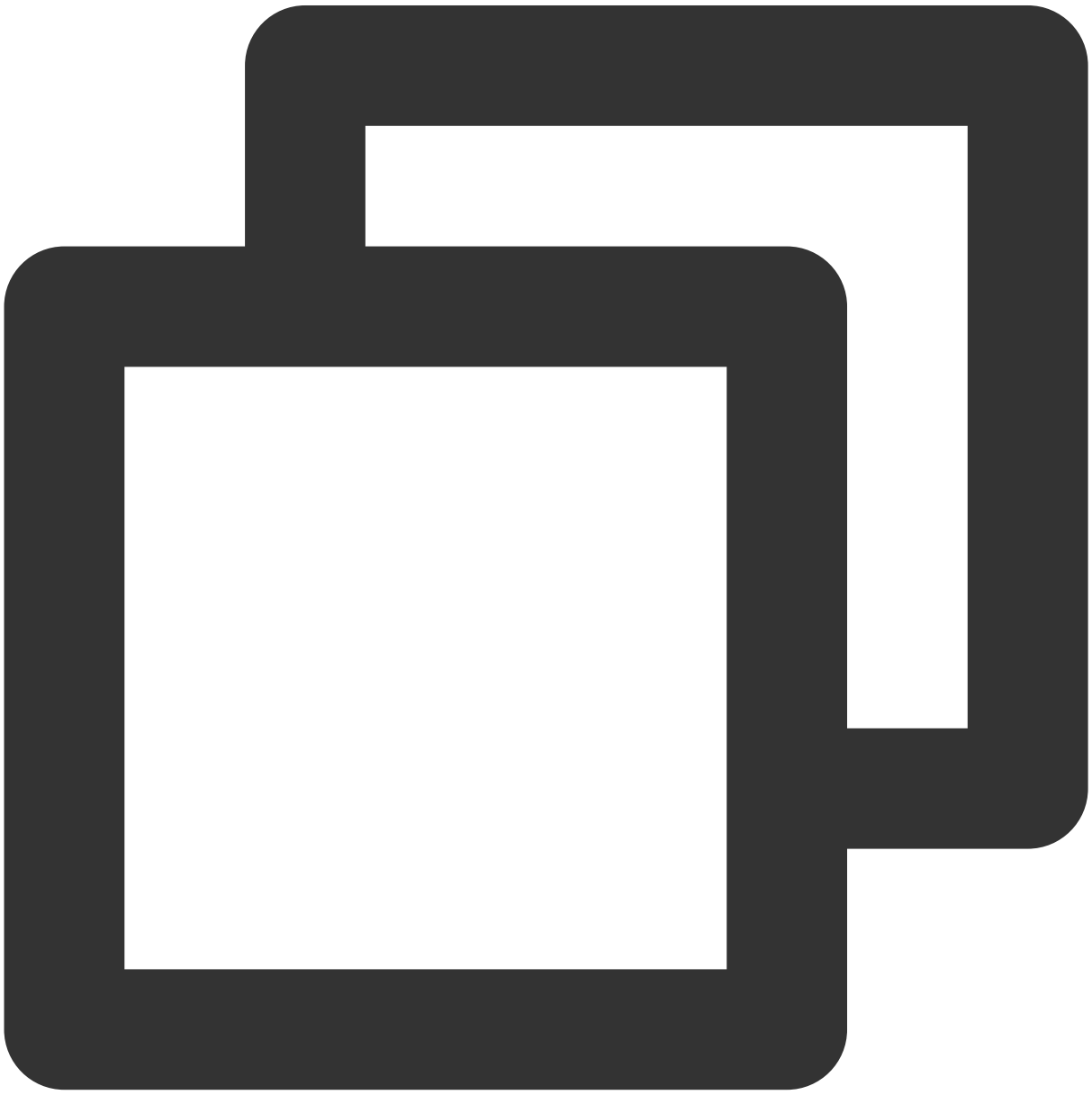
```
virtual void onUserScreenCaptureStopped(int reason) = 0;
```

参数如下表所示：

参数	类型	含义
reason	int	停止原因，0：用户主动停止；1：屏幕窗口关闭导致停止；2：表示屏幕分享的显示屏状态变更（如接口被拔出、投影模式变更等）

onRoomMaxSeatCountChanged

上麦控制发生变化事件。



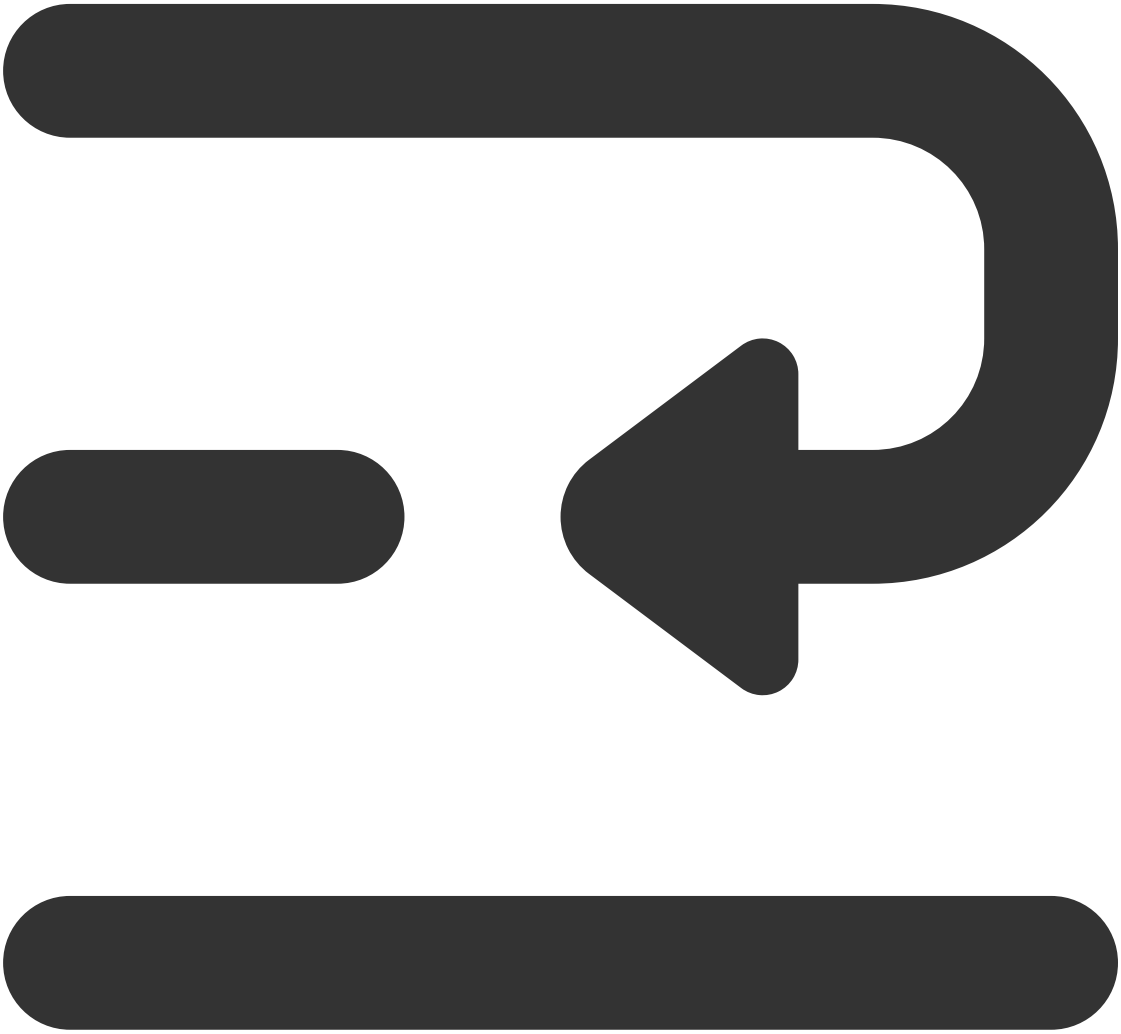
```
virtual void onRoomMaxSeatCountChanged(const char* roomId, int maxSeatCount) = 0;
```

参数如下表所示：

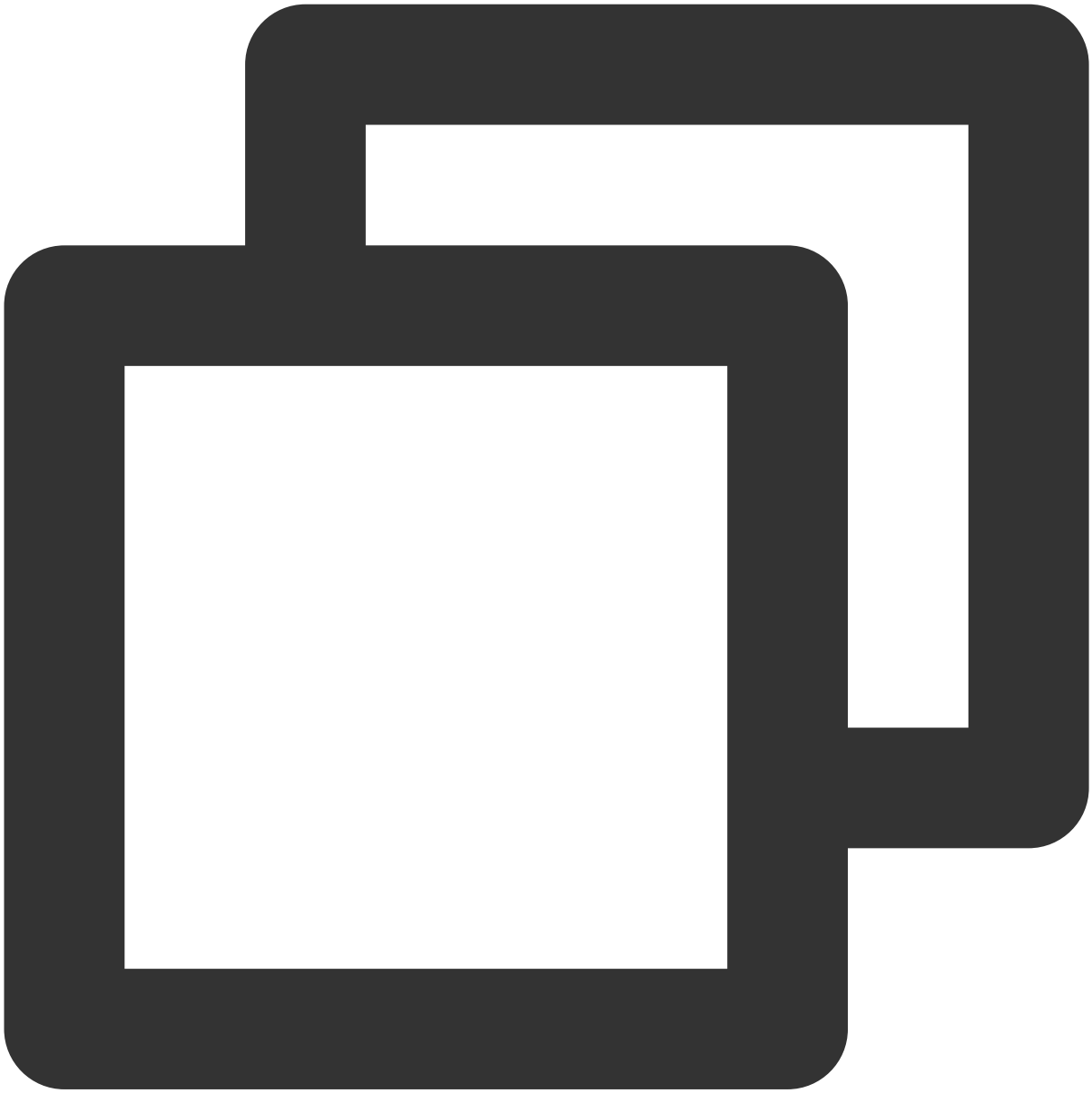
参数	类型	含义
roomId	const char*	房间 ID
maxSeatCount	int	最大麦上用户数量

onSeatListChanged

麦位列表发生变化事件。



版权所有：腾讯云计算（北京）有限责任公司



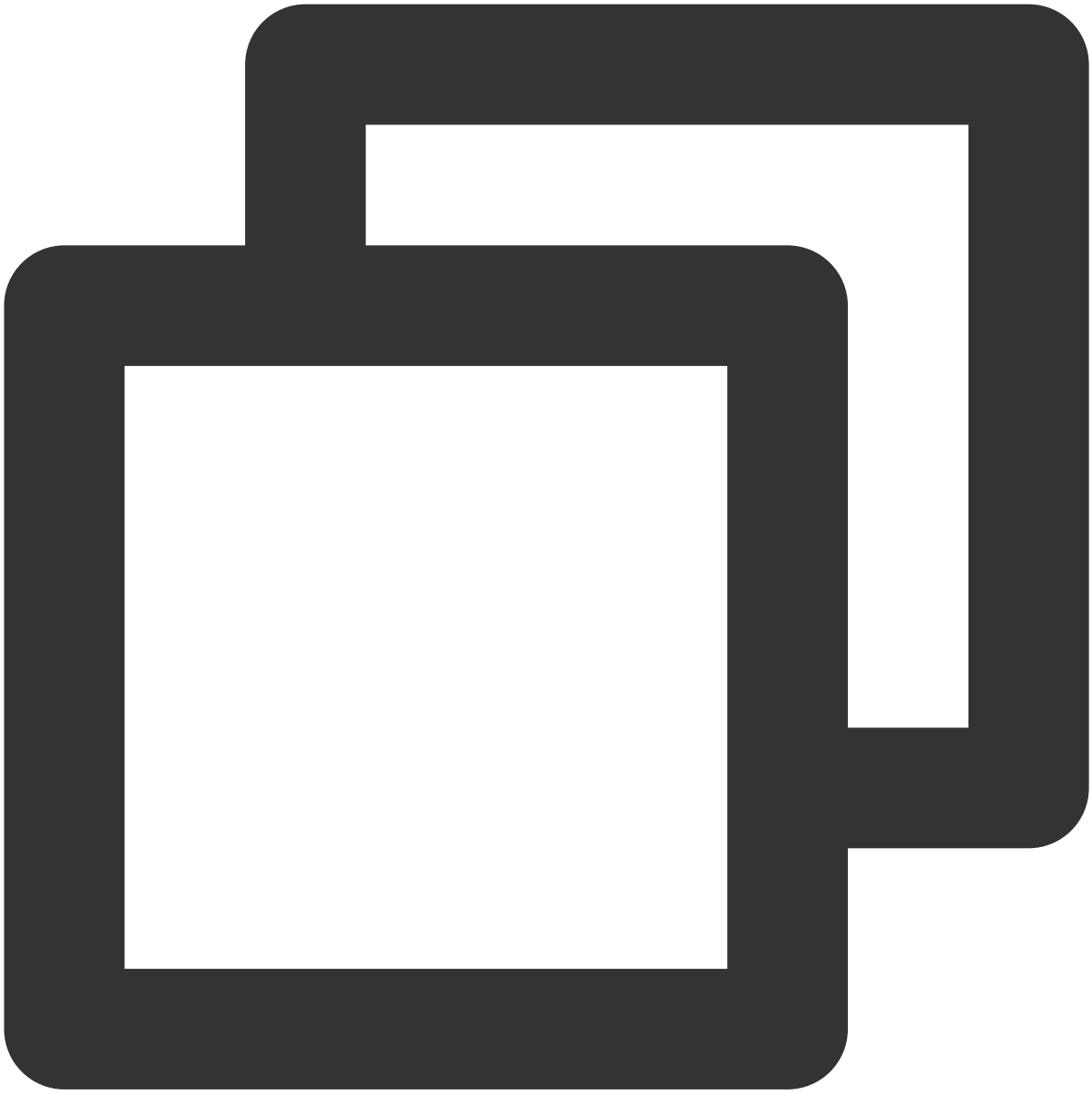
```
virtual void onSeatListChanged(TUIList<TUISeatInfo>* seatList, TUIList<TUISeatInfo>
```

参数如下表所示：

参数	类型	含义
seatList	TUIList<TUISeatInfo>*	目前麦上最新的用户列表，包含新上麦的用户
usersSeated	TUIList<TUISeatInfo>*	新上麦的用户列表
usersLeft	TUIList<TUISeatInfo>*	新下麦的用户列表

onKickedOffSeat

被踢下麦位事件。



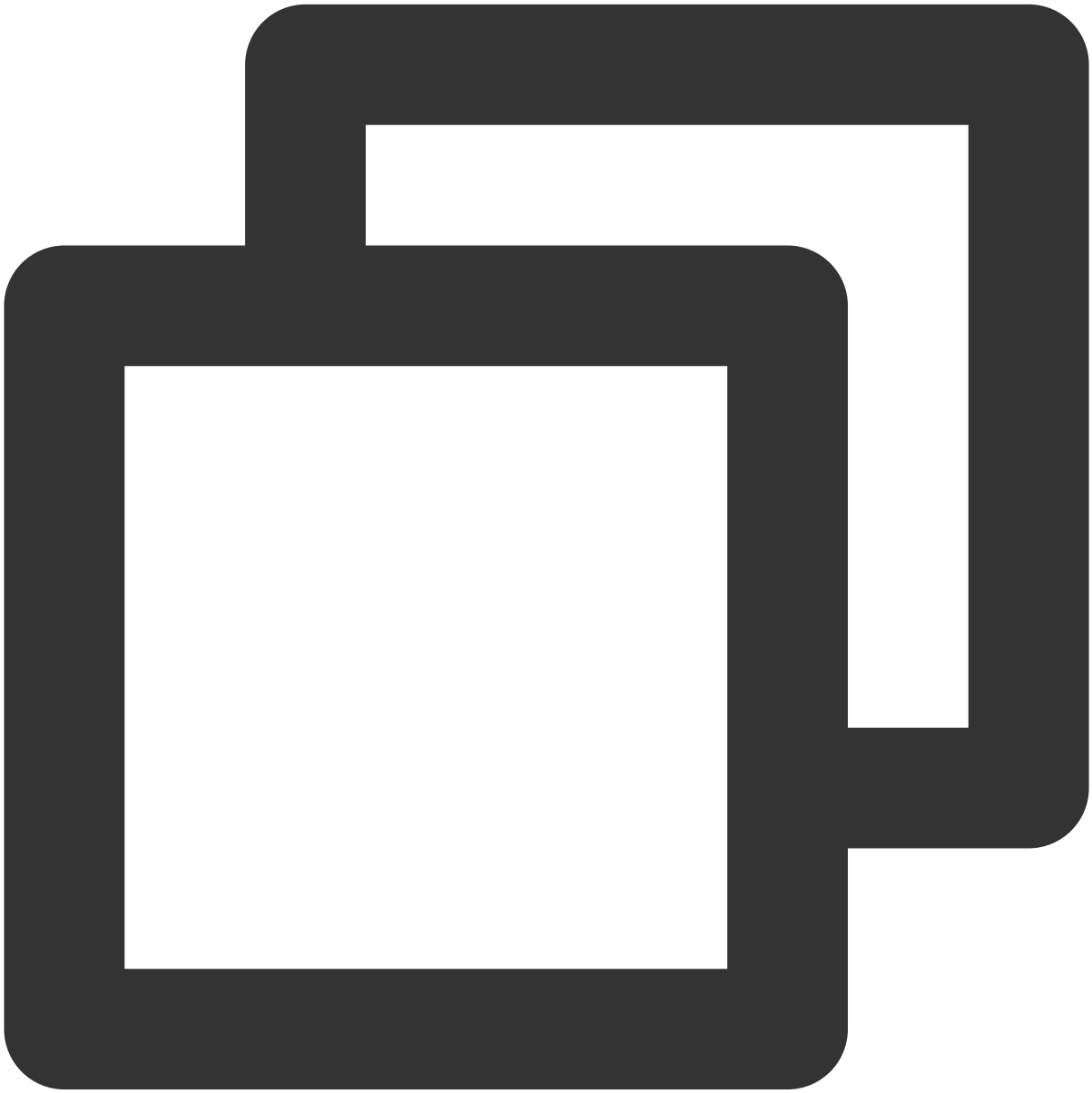
```
virtual void onKickedOffSeat(const char* userId) = 0;
```

参数如下表所示：

参数	类型	含义
userId	const char*	用户 ID

onRequestReceived

收到请求消息事件。



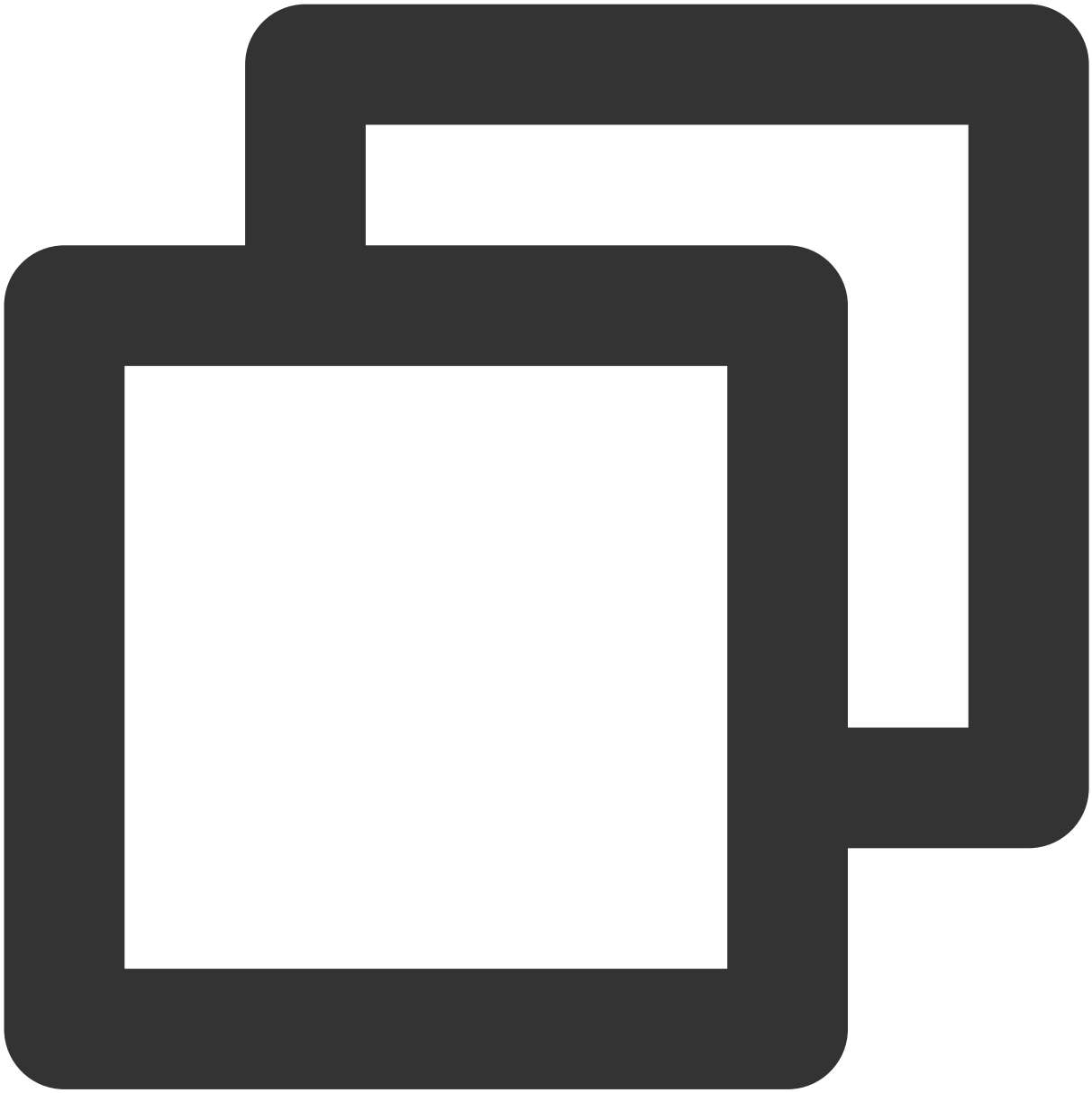
```
virtual void onRequestReceived(const TUIRequest* request) = 0;
```

参数如下表所示：

参数	类型	含义
request	const TUIRequest*	请求内容

onRequestCancelled

收到请求被取消的事件。



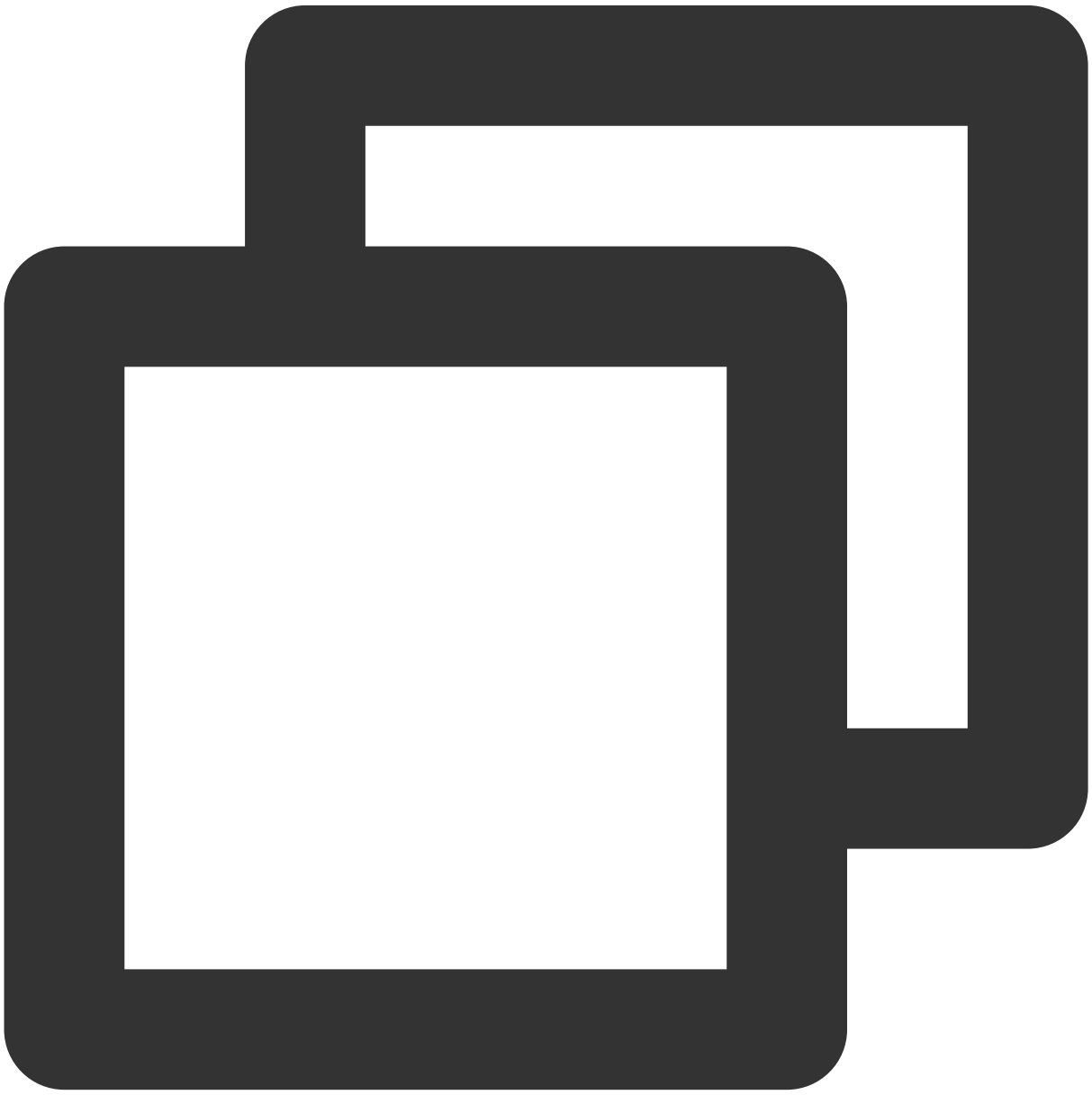
```
virtual void OnRequestCancelled(const char* requestId, const char* userId) = 0;
```

参数如下表所示：

参数	类型	含义
requestId	const char*	请求 ID
userId	const char*	用户 ID

onReceiveTextMessage

收到文本消息事件。



```
virtual void onReceiveTextMessage(const char* roomId, const TUIMessage& message) =
```

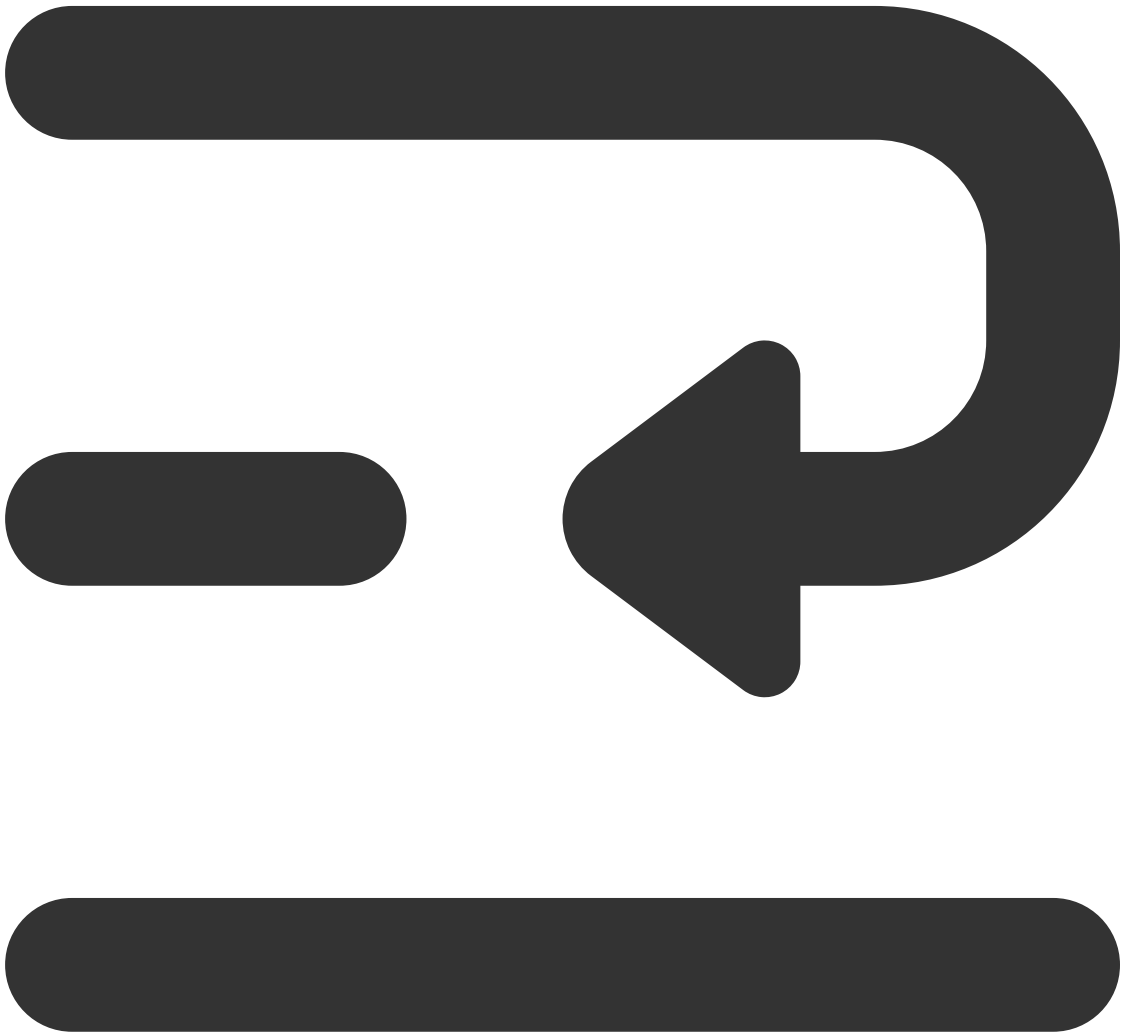
参数如下表所示：

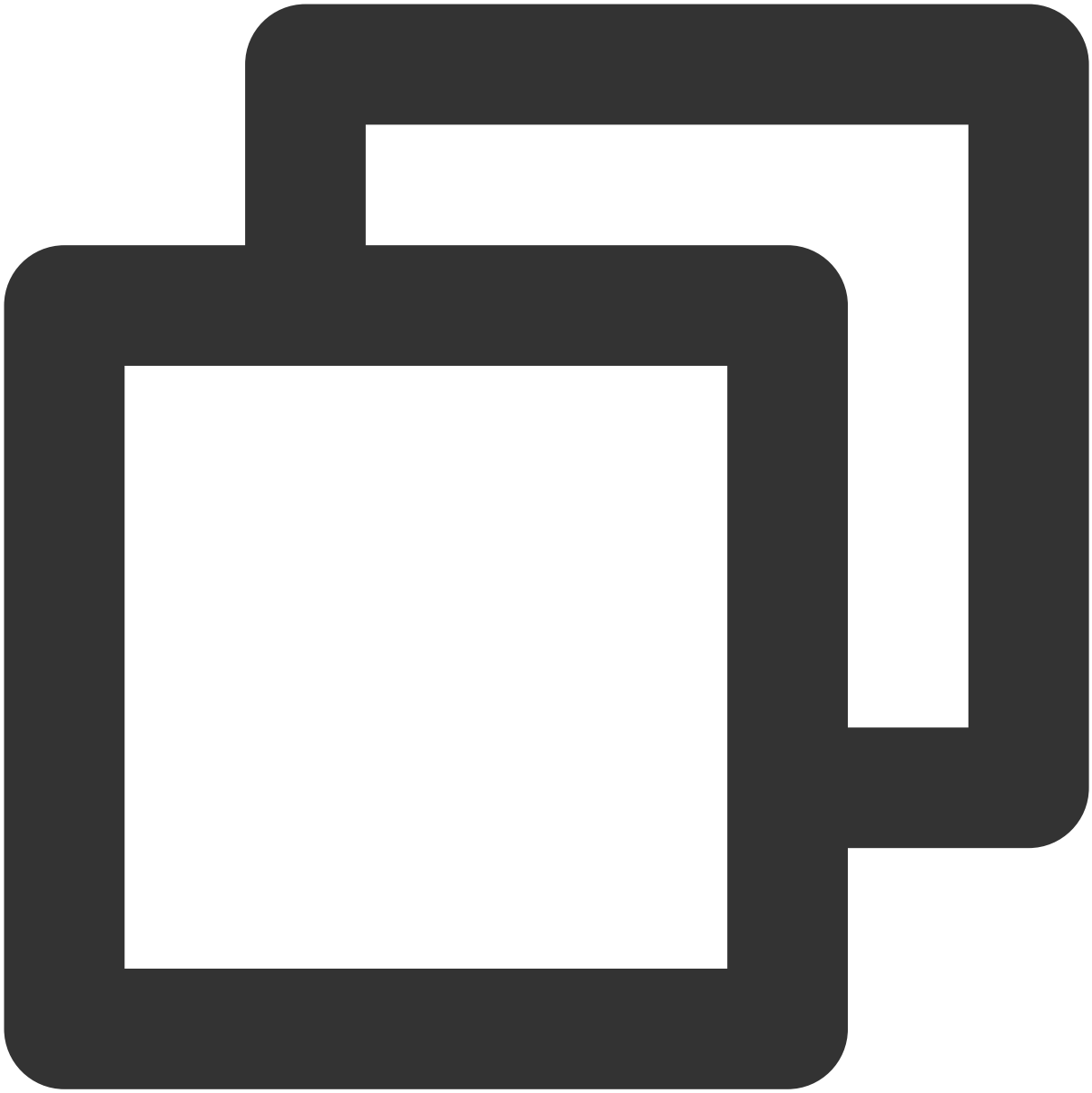
参数	类型	含义
roomId	const char*	房间 ID

message	const TUIMessage&	消息内容

onReceiveCustomMessage

收到自定义消息事件。





```
virtual void onReceiveCustomMessage(const char* roomId, const TUIMessage& message)
```

参数如下表所示：

参数	类型	含义
roomId	const char*	房间 ID
message	const TUIMessage&	消息内容

版权所有：腾讯云计算（北京）有限责任公司

类型定义

最近更新时间：2023-10-30 10:55:42

枚举定义

TUIRoomDefine

类型	说明
TUIRoomType	房间类型
TUISpeechMode	房间麦控模式
TUIMediaDevice	房间内媒体设备类型
TUIRole	房间内角色类型
TUIVideoQuality	视频质量
TUIAudioQuality	音频质量
TUIVideoStreamType	视频流类型
TUIChangeReason	更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）
TUICaptureSourceType	屏幕分享捕获源类型
TUIRequestAction	请求类型

TUICommonDefine

类型	说明
TUINetworkQuality	网络质量

常用结构

TUIRoomDefine

类型	说明
TUIRoomInfo	房间信息
TUILoginUserInfo	用户登录信息

TUIUserInfo	房间内用户信息
TUISeatInfo	房间内座位信息
TUISeatLockParams	锁定麦位操作参数
TUIUserVoiceVolume	房间内用户音量
TUIRequest	信令请求
TUIShareTarget	屏幕分享采集源信息

TUICommonDefine

类型	说明
TUINetworkInfo	网络质量信息
TUIMessage	消息
TUIImageBuffer	图片信息

TUIRoomType

房间类型

枚举	取值	描述
kConference	1	Conference 类型房间，适用于会议，教育场景，该房间中麦位是无序的
kLivingRoom	2	LivingRoom类型房间，适用于直播场景，该房间中麦位是有序的

TUISpeechMode

麦控模式

枚举	取值	描述
kFreeToSpeak	1	自由发言模式。
kApplyToSpeak	2	申请发言模式。（仅在会议类型房间下生效）
kApplySpeakAfterTakingSeat	3	上麦发言模式。

TUIMediaDevice

房间内媒体设备类型

枚举	取值	描述
kMicrophone	1	麦克风。
kCamera	2	摄像头
kApplyScreenSharing	3	屏幕共享

TUIRole

房间内角色类型

枚举	取值	描述
kRoomOwner	0	房主，一般指房间的创建者，房间内最高权限拥有者
kAdministrator	1	房间管理员
kGeneralUser	2	房间内普通成员

TUIVideoQuality

视频质量

枚举	取值	描述
kVideoQuality_360P	1	低清360P
kVideoQuality_540P	2	标清540P
kVideoQuality_720P	3	高清720P
kVideoQuality_1080P	4	超清1080P

TUIAudioQuality

音频质量

枚举	取值	描述
kAudioQualitySpeech	0	人声模式
kAudioQualityDefault	1	默认模式

版权所有：腾讯云计算（北京）有限责任公司

kAudioQualityMusic	2	音乐模式

TUIVideoStreamType

视频流类型

枚举	取值	描述
kCameraStream	0	高清摄像头视频流
kScreenStream	1	屏幕分享视频流
kCameraStreamLow	2	低清摄像头视频流

TUIChangeReason

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

枚举	取值	描述
kChangedBySelf	0	自己操作
kChangedByAdmin	1	房主或管理员操作

TUICaptureSourceType

屏幕分享捕获源类型

枚举	取值	描述
kUnknown	-1	未知类型
kWindow	0	窗口类型
kScreen	1	屏幕

TUIRequestAction

请求类型

枚举	取值	描述
kInvalidAction	0	无效请求
kRequestToOpenRemoteCamera	1	请求远端用户打开摄像头
kRequestToOpenRemoteMicrophone	2	请求远端用户打开麦克风
kRequestToConnectOtherRoom	3	请求连接到其他房间

版权所有：腾讯云计算（北京）有限责任公司

kRequestToTakeSeat	4	请求上麦
kRequestRemoteUserOnSeat	5	请求远端用户上麦
kApplyToAdminToOpenLocalCamera	6	向管理员请求打开本地摄像头
kApplyToAdminToOpenLocalMicrophone	7	向管理员请求打开本地麦克风

TUINetworkQuality

网络质量

枚举	取值	描述
kQualityUnknown	0	未定义
kQualityExcellent	1	当前网络非常好
kQualityGood	2	当前网络比较好
kQualityPoor	3	当前网络一般
kQualityBad	4	当前网络较差
kQualityVeryBad	5	当前网络很差
kQualityDown	6	当前网络不满足 TRTC 的最低要求

TUIRoomInfo

房间信息

字段	类型	描述
roomId	const char*	房间ID(创建房间必填参数)
ownerId	const char*	主持人ID，默认为房间创建者（只读）
roomType	TUIRoomType	房间类型
name	const char*	房间名称
speechMode	TUISpeechMode	房间麦控模式
isCameraDisableForAllUser	bool	是否禁止打开摄像头，默认 false

isMicrophoneDisableForAllUser	bool	是否禁止打开麦克风，默认 false
isMessageDisableForAllUser	bool	是否禁止发送消息，默认 false
maxSeatCount	int	最大麦位数量
createTime	long	房间创建时间（只读）
memberCountt	int	房间内成员数量

TUILoginUserInfo

用户登录信息

字段	类型	描述
userId	const char*	用户ID
userName	const char*	用户名称
avatarUrl	const char*	用户头像URL

TUIUserInfo

房间内用户信息

字段	类型	描述
userId	const char*	用户ID
userName	const char*	用户名称
avatarUrl	const char*	用户头像URL
userRole	TUIRole	用户角色类型
hasAudioStream	bool	是否有音频流
hasVideoStream	bool	是否有视频流
hasScreenStream	bool	是否有屏幕分享流

TUISeatInfo

房间内座位信息

字段	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

index	int	麦位序号
userId	const char*	用户 ID
isLocked	bool	麦位是否被锁定，默认 false
isVideoLocked	bool	麦位是否被禁止打开视频，默认 false
isAudioLocked	bool	麦位是否被禁止打开音频，默认 false

TUISeatLockParams

锁定麦位操作参数

字段	类型	描述
lockSeat	bool	锁定麦位，默认 false
lockVideo	bool	锁定麦位摄像头，默认 false
lockAudio	bool	锁定麦位麦克风，默认 false

TUIUserVoiceVolume

房间内用户音量

字段	类型	描述
userId	const char*	用户 ID
volume	int	音量 用于承载所有正在说话的用户的音量大小，取值范围 0 - 100

TUIRequest

信令请求

字段	类型	描述
requestId	uint32_t	请求 ID
requestAction	TUIRequestAction	请求类型
userId	const char*	用户 ID
content	const char*	信令内容

版权所有：腾讯云计算（北京）有限责任公司

timestamp	uint32_t	时间戳

TUIShareTarget

信令请求

字段	类型	描述
id	TUISourceId	采集源的ID，对于窗口，该字段代表窗口的 ID；对于屏幕，该字段代表显示器的 ID
sourceType	TUICaptureSourceType	采集源类型
sourceName	const char*	采集源名称
thumbnailImage	TUIImageBuffer	缩略图
iconImage	TUIImageBuffer	图标
isMinimized	bool	是否最小化

TUINetworkInfo

网络质量信息

字段	类型	描述
userId	const char*	用户ID
quality	TUINetworkQuality	网络质量
upLoss	int	上行丢包率
downLoss	int	下行丢包率
delay	int	网络延迟

TUIMessage

消息详情

字段	类型	描述
messageId	const char*	消息 ID
message	const char*	消息文本
timestamp	int64_t	消息时间

版权所有：腾讯云计算（北京）有限责任公司

userId	const char*	消息发送者
userName	const char*	消息发送者昵称
avatarUrl	const char*	消息发送者头像

TUIImageBuffer

图片信息

字段	类型	描述
buffer	const char*	图片数据缓存地址
length	uint32_t	长度
width	uint32_t	宽度
height	uint32_t	高度

Web
API 概览

最近更新时间：2023-11-16 14:21:39

TUIRoomKit (含 UI 组件)

TUIRoomKit 接口列表

API	描述
init	初始化 TUIRoomKit 组件
createRoom	主持人创建房间
enterRoom	普通成员加入房间

TUIRoomKit 事件类型

事件	描述
onCreateRoom	创建房间回调
onEnterRoom	进入房间回调
onDestroyRoom	销毁房间回调
onExitRoom	普通成员退出房间回调
onKickedOutOfRoom	普通成员被主持人踢出房间通知
onKickedOffLine	用户账号在其他地方登录，被踢下线
onUserSigExpired	用户 userSig 过期，请重新获取 userSig

TUIRoomEngine (无 UI 接口)

TUIRoomEngine 接口列表

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEngine 静态方法

API	描述
once	监听 TUIRoomEngine ready 事件。 **注意：TUIRoomEngine.init 以外的所有方法必须在监听到 TUIRoomEngine ready 事件且 TUIRoomEngine.init 方法执行成功后执行。**
login	登录 TUIRoomEngine
setSelfInfo	设置当前用户基本信息（用户名、用户头像）
getSelfInfo	获取当前用户基本信息（用户名、用户头像）
logout	登出 TUIRoomEngine

roomEngine 房间管理 API

API	描述
createRoom	创建房间
enterRoom	进入房间
destroyRoom	销毁房间
exitRoom	离开房间
fetchRoomInfo	获取房间信息
updateRoomNameByAdmin	更新房间的名字（仅群主或者管理员可以调用）
updateRoomSpeechModeByAdmin	更新房间的发言模式（仅群主或者管理员可以调用）
getUserList	获取当前房间用户列表
getUserInfo	获取用户的详细信息

roomEngine 音视频 API

API	描述
setLocalVideoView	设置本地流的渲染位置
openLocalCamera	本地摄像头视频流采集
closeLocalCamera	关闭本地摄像头

版权所有：腾讯云计算（北京）有限责任公司

openLocalMicrophone	打开本地麦克风
closeLocalMicrophone	关闭本地麦克风
updateVideoQuality	设置本地视频的参数
updateAudioQuality	设置本地音频的参数
startScreenSharing	开始屏幕共享
stopScreenSharing	停止屏幕共享
startPushLocalVideo	开始向远端推本地视频流
stopPushLocalVideo	停止向远端推本地视频流
startPushLocalAudio	开始向远端推本地音频流
stopPushLocalAudio	停止向远端推本地音频流
setRemoteVideoView	设置远端流渲染的区域
startPlayRemoteVideo	开始播放远端用户视频流
stopPlayRemoteVideo	停止播放远端用户视频流
muteRemoteAudioStream	停止远端用户的音频流

roomEngine 成员管理 API

API	描述
openRemoteDeviceByAdmin	请求远端用户打开媒体设备
applyToAdminToOpenLocalDevice	参会者向主持人申请打开设备
closeRemoteDeviceByAdmin	关闭远端用户媒体设备
cancelRequest	取消已经发出的请求
responseRemoteRequest	回复远端用户的请求
changeUserRole	改变用户的角色
kickRemoteUserOutOfRoom	将用户踢出房间
disableDeviceForAllUserByAdmin	禁用/打开所有用户的媒体设备
disableSendingMessageForAllUser	禁止/允许所有用户发送消息

版权所有：腾讯云计算（北京）有限责任公司

disableSendingMessageByAdmin	禁止/允许某个用户发送消息

roomEngine 麦位管理 API

API	描述
setMaxSeatCount	设置房间麦位最大值
getSeatList	获取麦位信息
takeSeat	获取麦位
leaveSeat	释放麦位
takeUserOnSeatByAdmin	邀请其他人上麦（仅房间主持人和管理员可调用此方法）
kickUserOffSeatByAdmin	将其他人踢下麦（仅房间主持人和管理员可调用此方法）
lockSeatByAdmin	锁定某个麦位状态（仅房间主持人和管理员可调用此方法）

roomEngine 消息发送 API

API	描述
sendTextMessage	发送文本消息
sendCustomMessage	发送自定义消息

roomEngine 设备管理 API

API	描述
getCameraDevicesList	获取摄像头设备列表
getMicDevicesList	获取麦克风设备列表
getSpeakerDevicesList	获取扬声器设备列表
setCurrentCameraDevice	设置要使用的摄像头设备
setCurrentMicDevice	设置要使用的麦克风设备
setCurrentSpeakerDevice	设置要使用的扬声器设备
getCurrentCameraDevice	获取当前正在使用的摄像头设备

版权所有：腾讯云计算（北京）有限责任公司

getCurrentMicDevice	获取当前正在使用的麦克风设备
getCurrentSpeakerDevice	获取当前正在使用的扬声器设备
startCameraDeviceTest	开始进行摄像头测试
stopCameraDeviceTest	停止摄像头测试

roomEngine 事件监听 API

API	描述
on	监听 TUIRoomEvents 事件
off	取消监听 TUIRoomEvents 事件

roomEngine 其他 API

API	描述
getTRTCCloud	获取 trtcCloud 实例
getTIM	获取 tim 实例

TUIRoomEngine 事件类型

TUIRoomEvent 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

EVENT	描述
TUIRoomEvents.onError	错误事件
TUIRoomEvents.onKickedOutOfRoom	踢出房间事件
TUIRoomEvents.onKickedOffLine	当前用户被踢下线事件
TUIRoomEvents.onUserSigExpired	userSig 过期事件
TUIRoomEvents.onRoomDismissed	主持人销毁房间事件
TUIRoomEvents.onRoomNameChanged	房间名称修改事件
TUIRoomEvents.onRoomSpeechModeChanged	房间发言模式修改事件
TUIRoomEvents.onAllUserCameraDisableChanged	所有成员摄像头使用权限改变事件

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEvents.onAllUserMicrophoneDisableChanged	所有成员麦克风使用权限改变事件
TUIRoomEvents.onSendMessageForAllUserDisableChanged	所有成员发送消息状态改变事件
TUIRoomEvents.onRoomMaxSeatCountChanged	房间最大麦位数修改事件
TUIRoomEvents.onRemoteUserEnterRoom	远端用户进入房间事件
TUIRoomEvents.onRemoteUserLeaveRoom	远端用户离开房间事件
TUIRoomEvents.onUserRoleChanged	用户角色改变事件
TUIRoomEvents.onUserVideoStateChanged	用户视频状态改变事件
TUIRoomEvents.onUserAudioStateChanged	用户音频状态改变事件
TUIRoomEvents.onSendMessageForUserDisableChanged	用户发送消息状态事件
TUIRoomEvents.onUserVoiceVolumeChanged	用户音量改变事件
TUIRoomEvents.onUserNetworkQualityChanged	用户网络质量改变事件
TUIRoomEvents.onSeatListChanged	麦位列表改变事件
TUIRoomEvents.onKickedOffSeat	用户被踢下麦事件
TUIRoomEvents.onRequestReceived	请求接收事件
TUIRoomEvents.onRequestCancelled	请求取消事件
TUIRoomEvents.onReceiveTextMessage	接收文本消息事件
TUIRoomEvents.onReceiveCustomMessage	接收自定义消息事件
TUIRoomEvents.onDeviceChange	设备变更事件
TUIRoomEvents.onUserScreenCaptureStopped	屏幕分享停止事件 当用户使用浏览器自带的**停止分享**按钮结束屏幕共享时，该用户会收到 'onUserScreenCaptureStopped' 事件用来修改屏幕共享状态。

TUIRoomKit

最近更新时间：2023-11-16 14:20:45

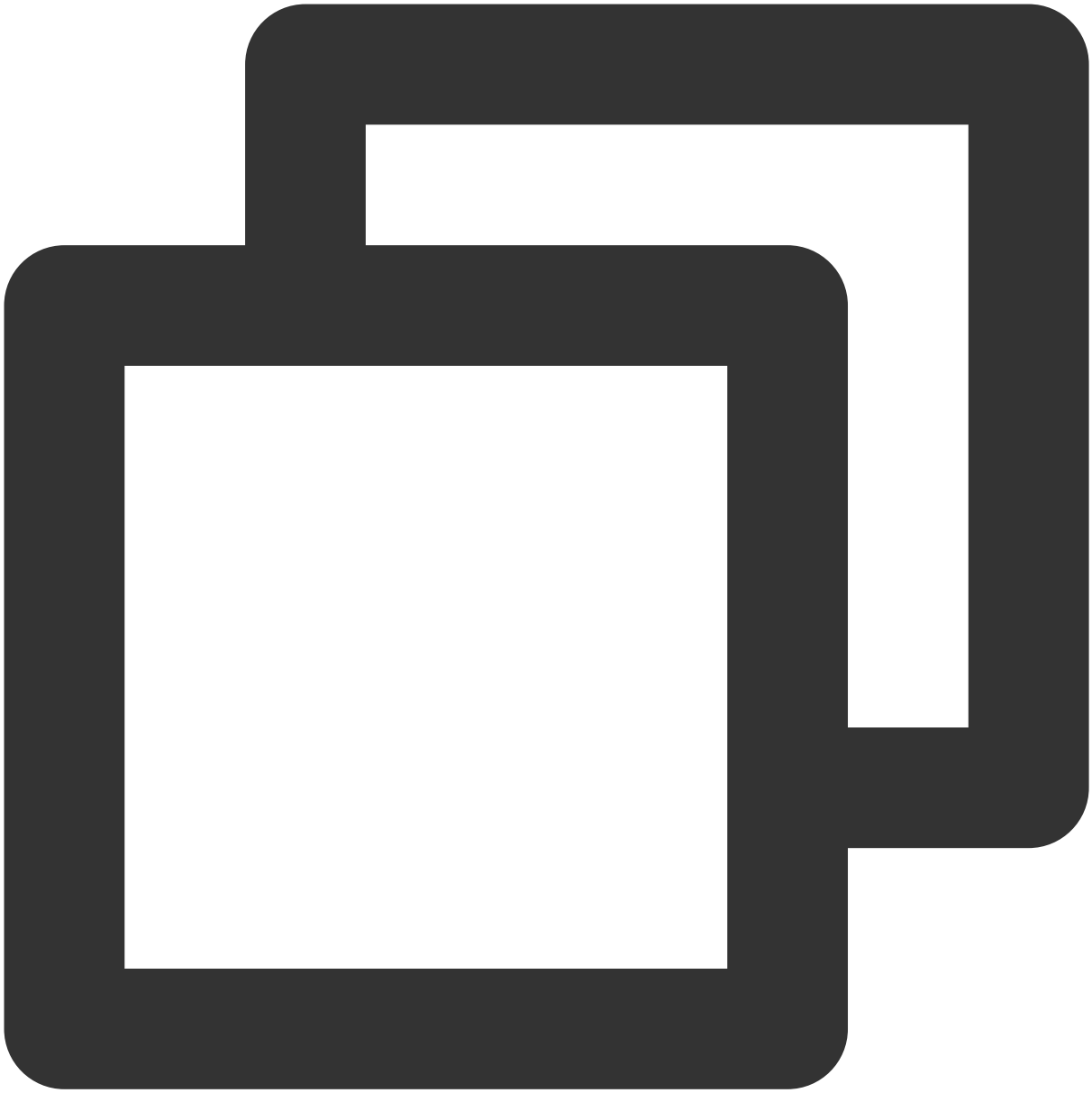
API 简介

TUIRoomKit API 是人视频会议组件的**含 UI 接口**，使用 TUIRoomKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，如果您想要体验并调试视频会议效果，请阅读 Demo 快速跑通，如果您想把我们的功能直接嵌入到您的项目中，请阅读快速接入(TUIRoomKit)。

API 概览

在页面中引用 TUIRoomKit 组件。例如：在 `App.vue` 组件中引入 TUIRoomKit 组件。
TUIRoomKit 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。
主持人及普通成员可通过 init 方法向 TUIRoomKit 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。

版权所有：腾讯云计算（北京）有限责任公司



```
<template>
  <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
// 引入 TUIRoom 组件，注意确认引入路径是否正确
import Room from './TUIRoom/index.vue';
// 获取 TUIRoom 组件元素，用于调用 TUIRoom 组件的方法
const TUIRoomRef = ref();
```

```
onMounted(async () => {
  // 初始化 TUIRoom 组件
  // 主持人在创建房间前需要先初始化 TUIRoom 组件
  // 普通成员在进入房间前需要先初始化 TUIRoom 组件
  await TUIRoomRef.value.init({
    // 获取 sdkAppId 请您参考 步骤一
    sdkAppId: 0,
    // 用户在您业务中的唯一标示 Id
    userId: '',
    // 本地开发调试可在 https://console.tencentcloud.com/trtc/usersigtool 页面快速生成 u
    userSig: '',
    // 用户在您业务中使用的昵称
    userName: '',
    // 用户在您业务中使用的头像链接
    avatarUrl: '',
  })
  // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
  await handleCreateRoom();
})

// 主持人创建房间，该方法只在创建房间时调用
async function handleCreateRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模式) 两种
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
  const roomId = '123456';
  const roomMode = 'FreeToSpeak';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
  try {
    await TUIRoomRef.value.createRoom({ roomId, roomName: roomId, roomMode, roomPar
  } catch (error: any) {
    alert('TUIRoomKit.createRoom error: ' + error.message);
  }
}

// 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
async function handleEnterRoom() {
  // roomId 为用户进入的房间号，要求 roomId 类型为 string
  // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认媒体设备
  const roomId = '123456';
  const roomParam = {
    isOpenCamera: true,
    isOpenMicrophone: true,
  }
```

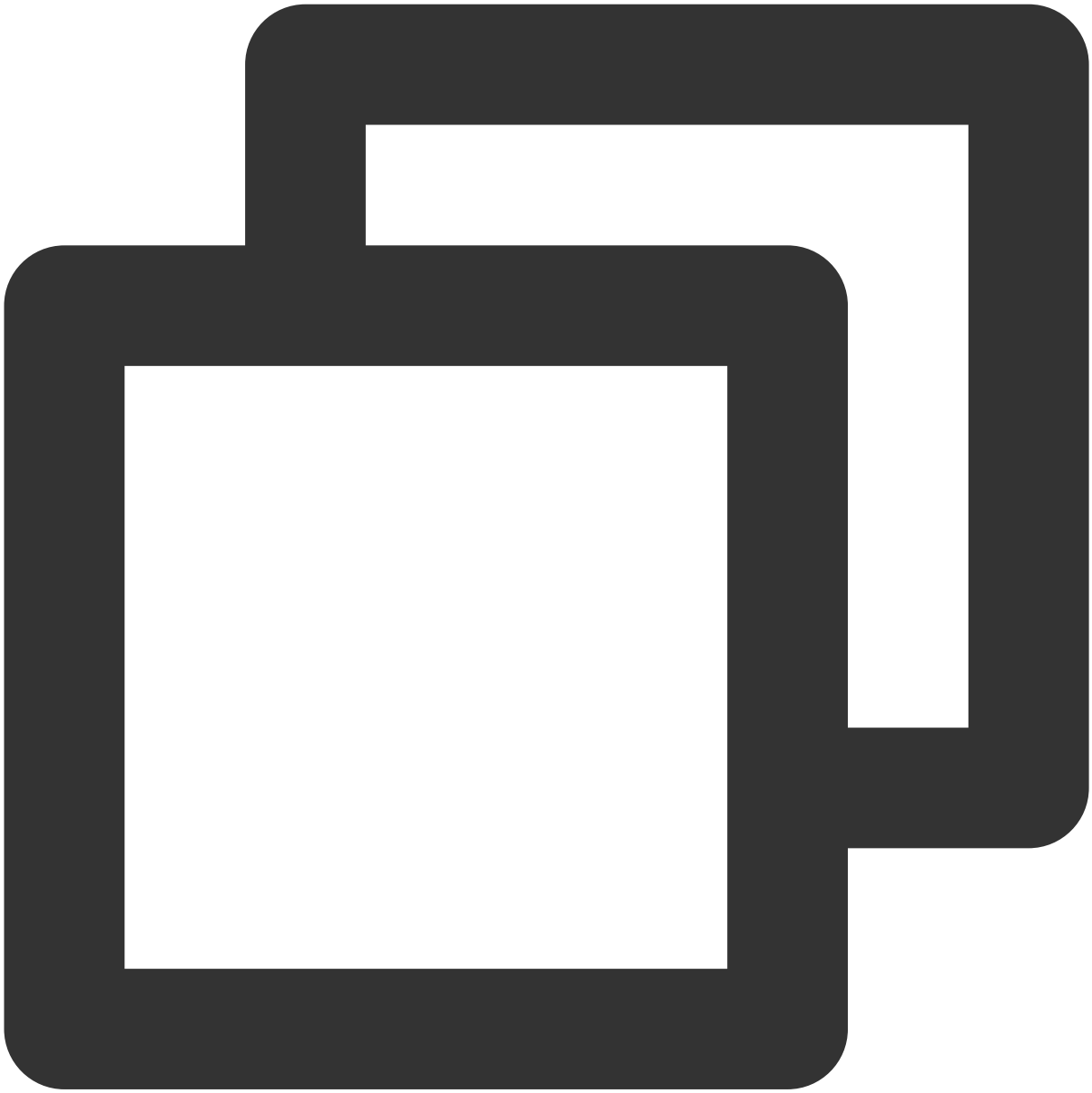
```
  try {
    await TUIRoomRef.value.enterRoom({ roomId, roomParam });
  } catch (error: any) {
    alert('TUIRoomKit.enterRoom error: ' + error.message);
  }
}
</script>

<style lang="scss">
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
  width: 100%;
  height: 100%;
  * {
    box-sizing: border-box;
  }
}
</style>
```

API 详情

TUIRoomkit 接口

init

初始化 TUIRoomKit 数据，任何使用 TUIRoomKIt 的用户都需要调用 init 方法。



```
TUIRoomRef.value.init(roomData);
```

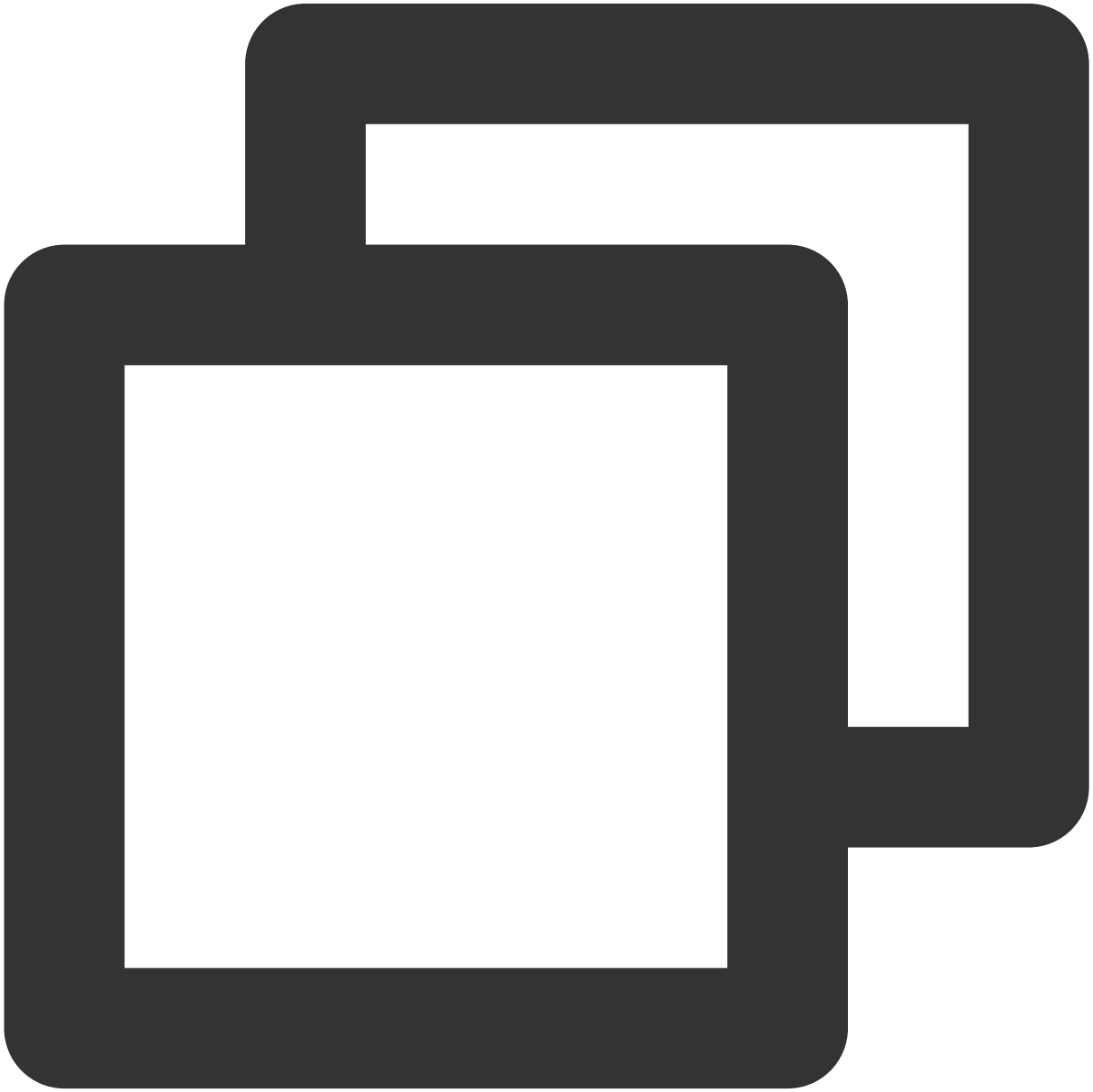
参数如下表所示：

参数	类型	含义
roomData	object	
roomData.sdkAppId	number	客户的 SDKAppId
roomData.userId	string	用户的唯一 ID

roomData.userSig	string	用户的 UserSig
roomData.userName	string	用户的昵称
roomData.avatarUrl	string	用户的头像地址

createRoom

主持人创建房间。



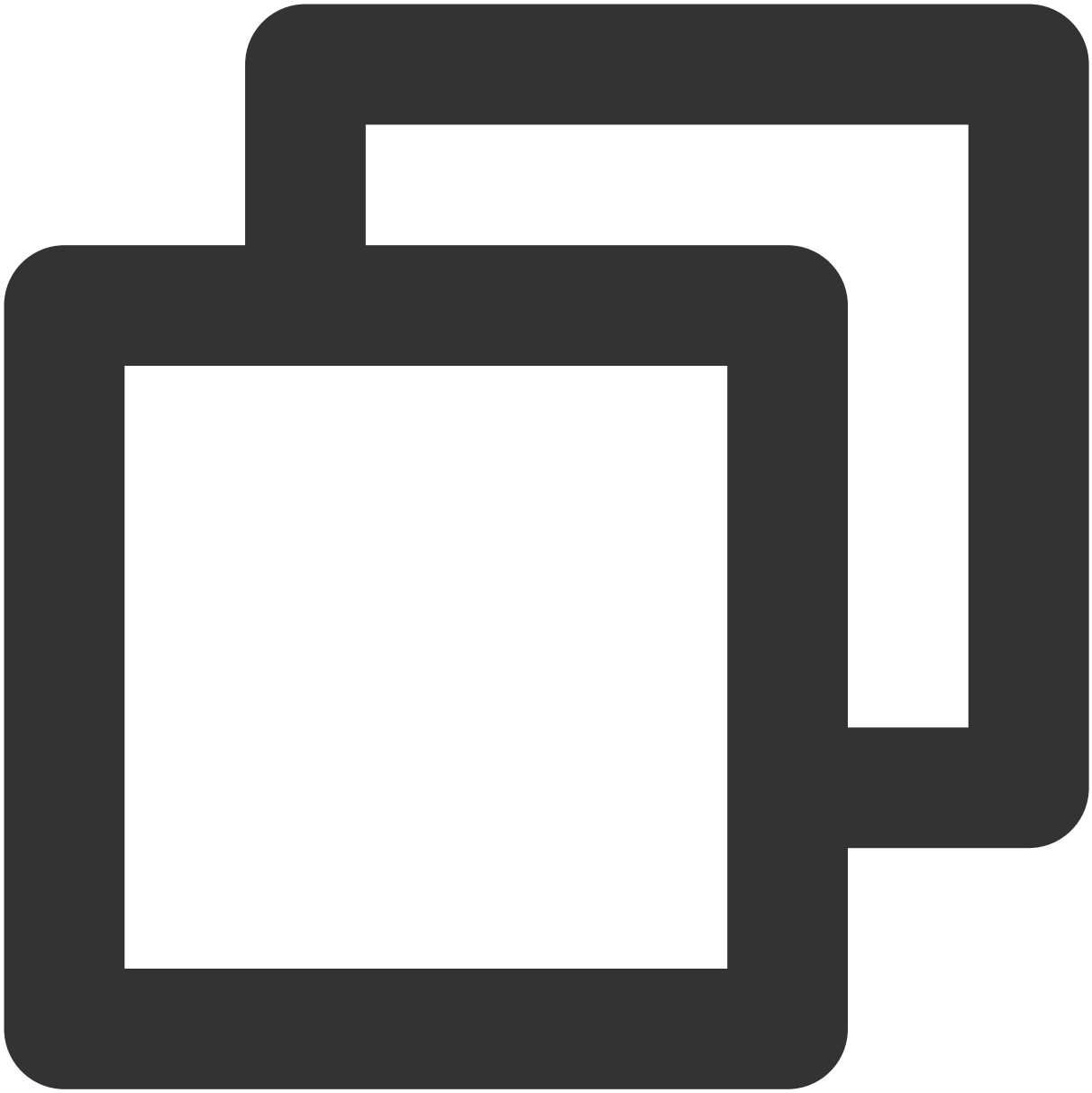
```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

参数如下表所示：

参数	类型	含义
roomId	string	房间 ID
roomMode	string	房间模式，**FreeToSpeak**（自由发言模式）和 **SpeakAfterTakingSeat**（上台发言模式），默认为 **FreeToSpeak**
roomParam	Object	非必填
roomParam.isOpenCamera	string	非必填，进房是否打开摄像头，默认为关闭
roomParam.isOpenMicrophone	string	非必填，进房是否打开麦克风，默认为关闭
roomParam.defaultCameraId	string	非必填，默认摄像头设备 ID
roomParam.defaultMicrophoneId	string	非必填，默认麦克风设备 ID
roomParam.defaultSpeakerId	String	非必填，默认扬声器设备 ID

enterRoom

普通成员加入房间。



```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

参数如下表所示：

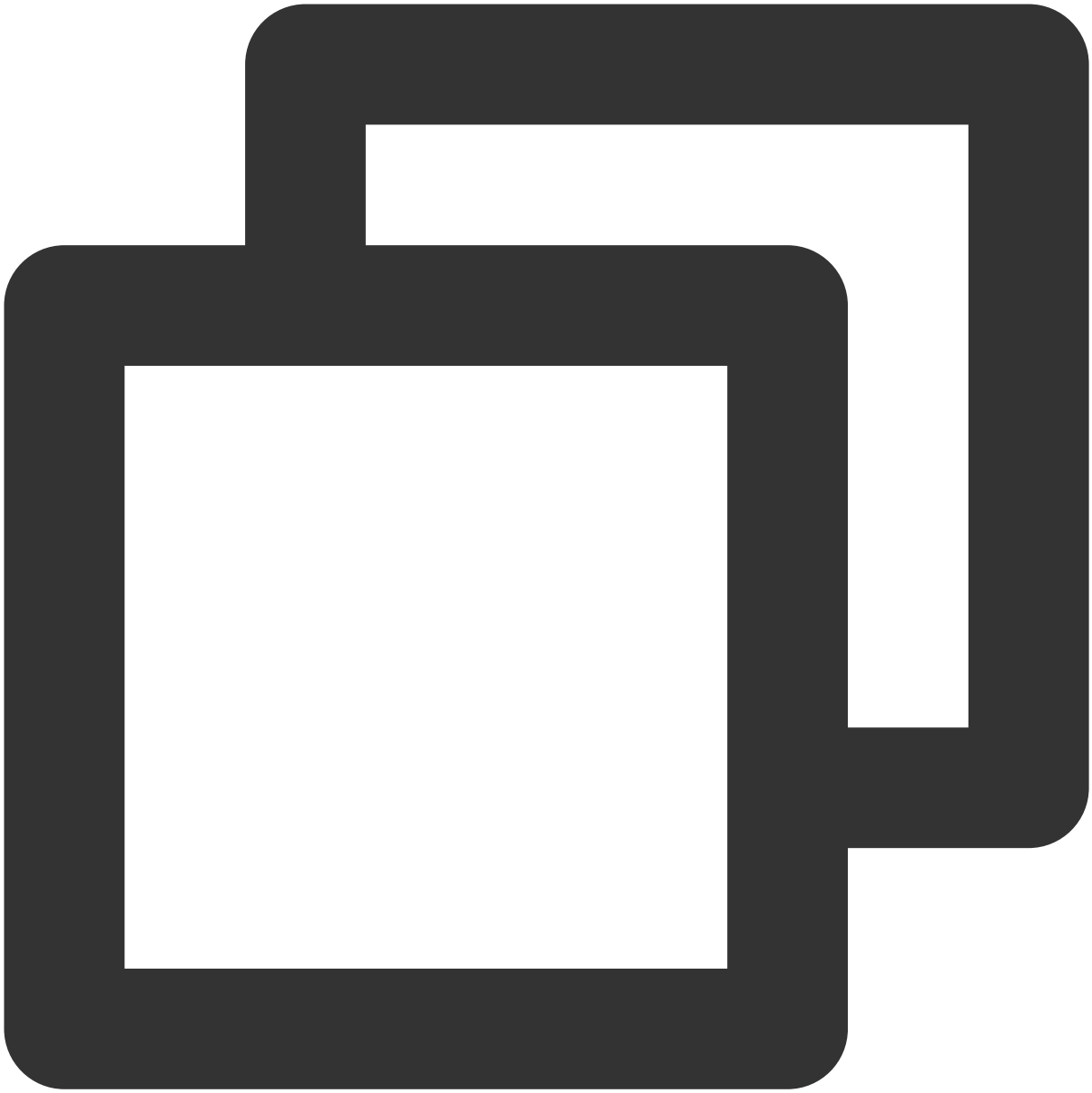
参数	类型	含义
roomId	string	房间 ID
roomParam	Object	非必填
roomParam.isOpenCamera	string	非必填，进房是否打开摄像头，默认为关闭

roomParam.isOpenMicrophone	string	非必填，进房是否打开麦克风，默认为关闭
roomParam.defaultCameraId	string	非必填，默认摄像头设备 ID
roomParam.defaultMicrophoneId	string	非必填，默认麦克风设备 ID
roomParam.defaultSpeakerId	String	非必填，默认扬声器设备 ID

TUIRoomkit 事件

onCreateRoom

创建房间回调。

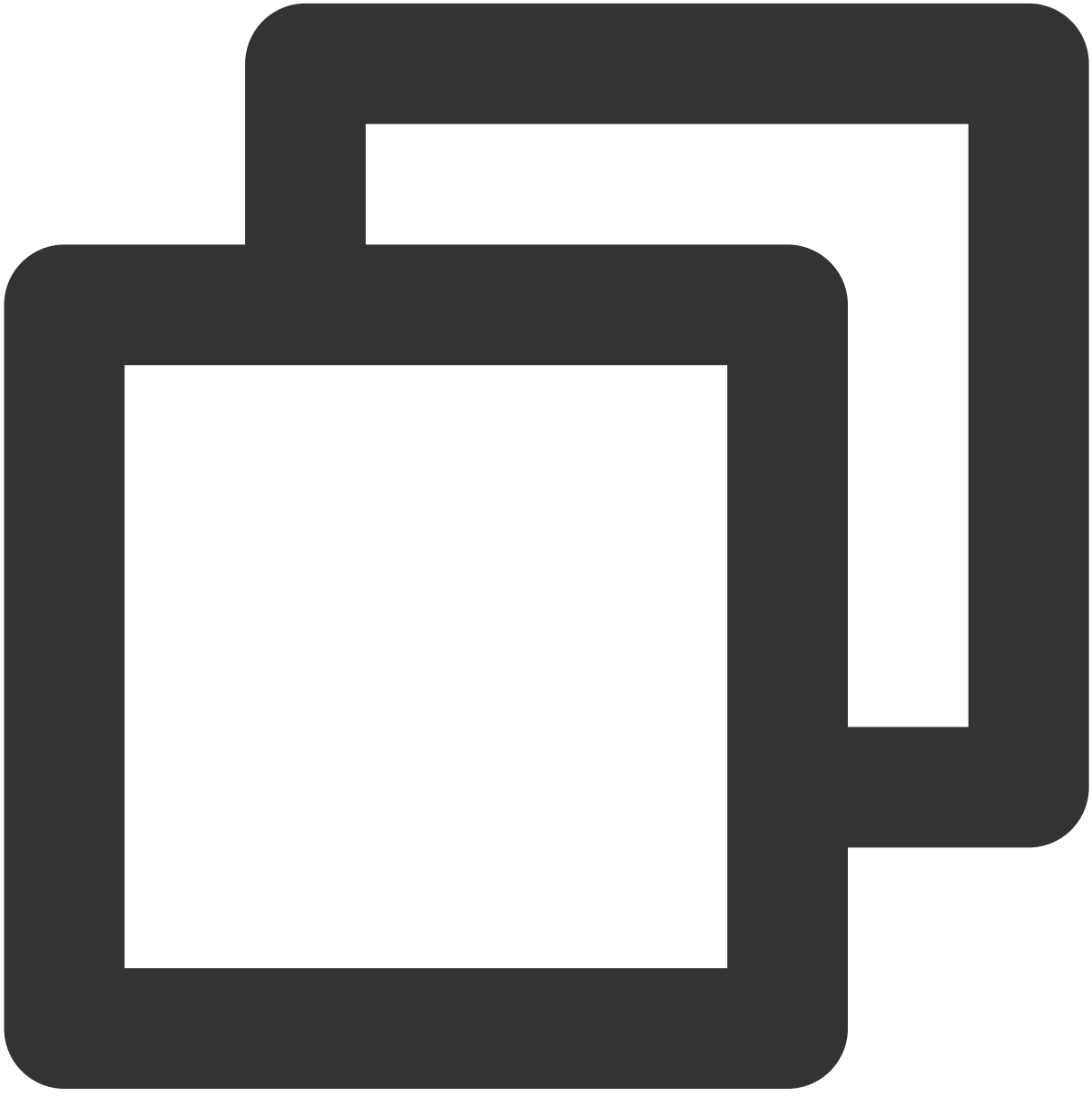


```
<template>
  <room ref="TUIRoomRef" @on-create-room="onCreateRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onCreateRoom(info) {
    if (info.code === 0) {
      console.log('创建房间成功')
```

```
    }
  }
</script>
```

onEnterRoom

进入房间回调。

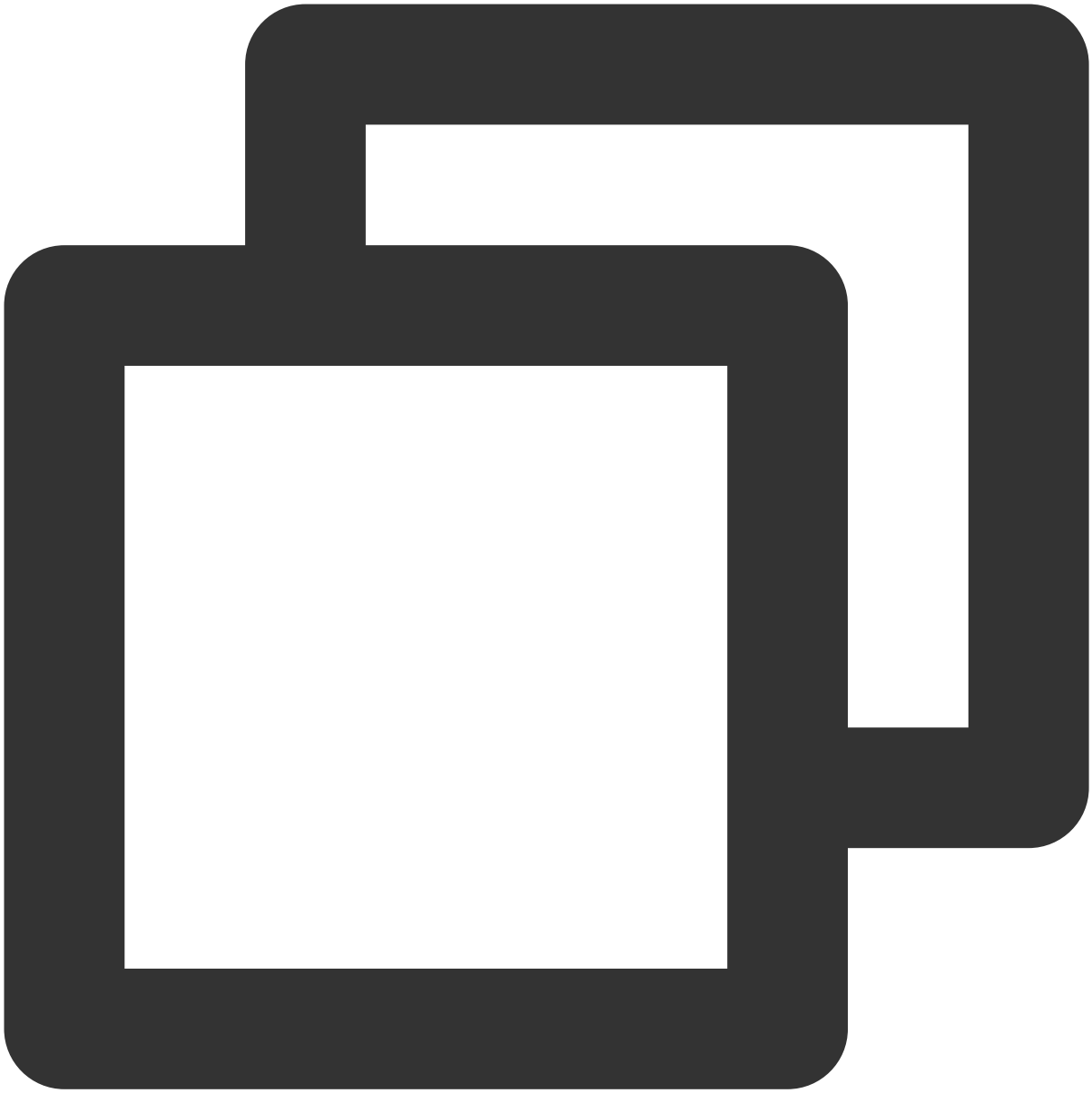


```
<template>
  <room ref="TUIRoomRef" @on-enter-room="onEnterRoom"></room>
</template>
```

```
<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onEnterRoom(info) {
    if (info.code === 0) {
      console.log('进入房间成功')
    }
  }
</script>
```

onDestroyRoom

主持人销毁房间的回调。

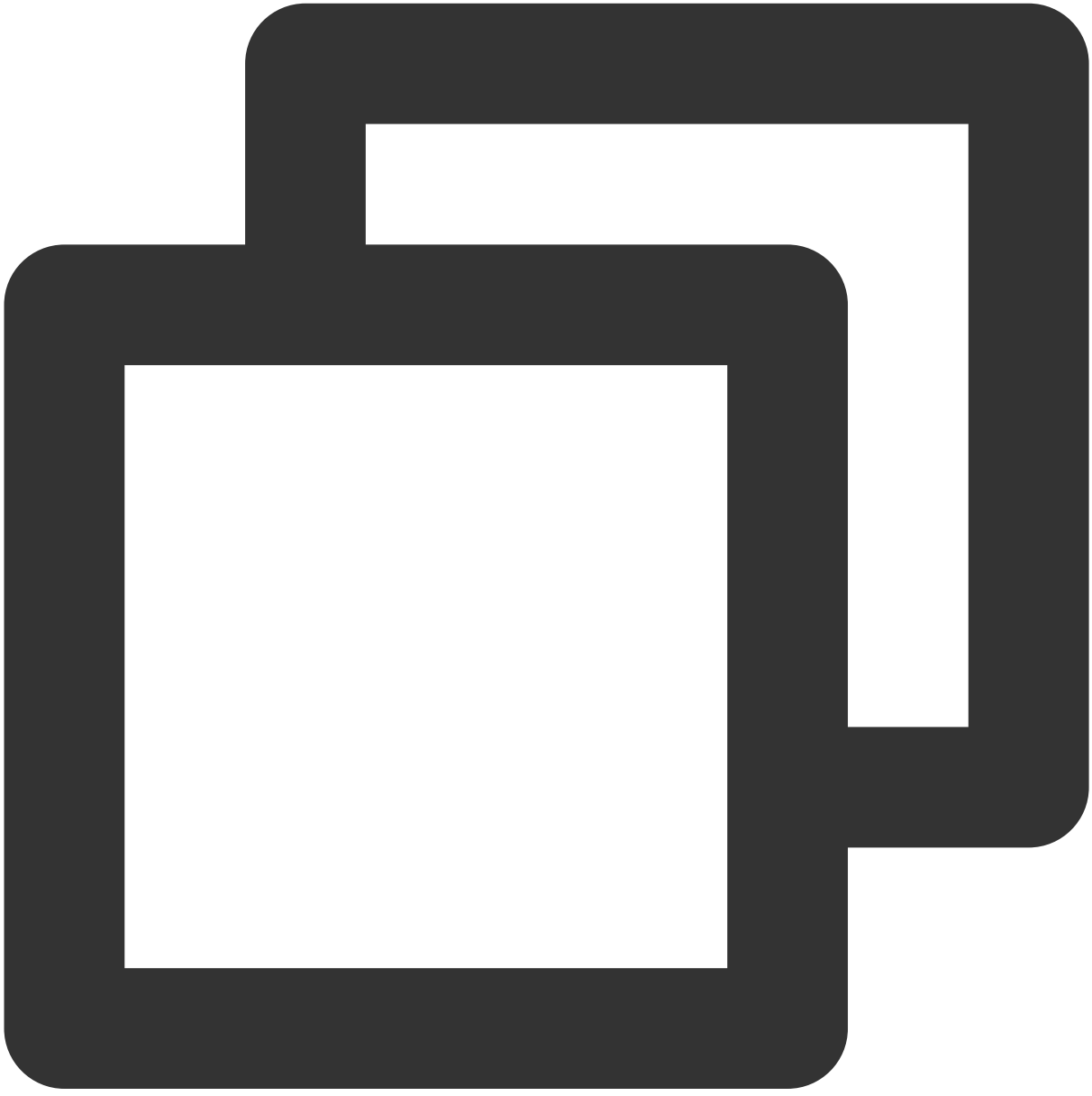


```
<template>
  <room ref="TUIRoomRef" @on-destroy-room="onDestroyRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onDestroyRoom(info) {
    if (info.code === 0) {
      console.log('主持人销毁成功')
```

```
    }
  }
</script>
```

onExitRoom

普通成员退出房间的回调。

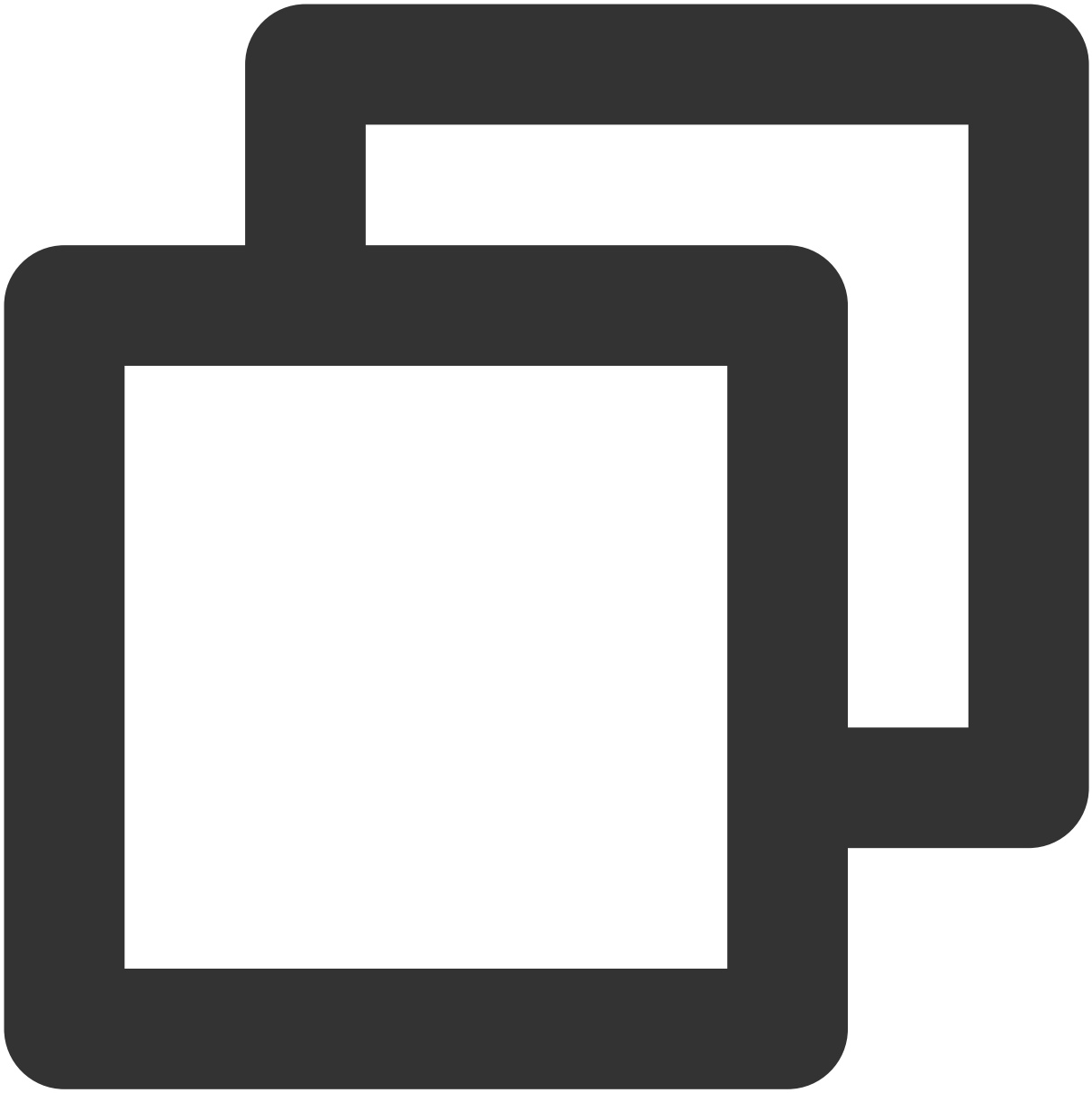


```
<template>
  <room ref="TUIRoomRef" @on-exit-room="onExitRoom"></room>
</template>
```

```
<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onExitRoom(info) {
    if (info.code === 0) {
      console.log('普通成员退出房间成功')
    }
  }
</script>
```

onKickedOutOfRoom

普通成员被主持人踢出房间通知。

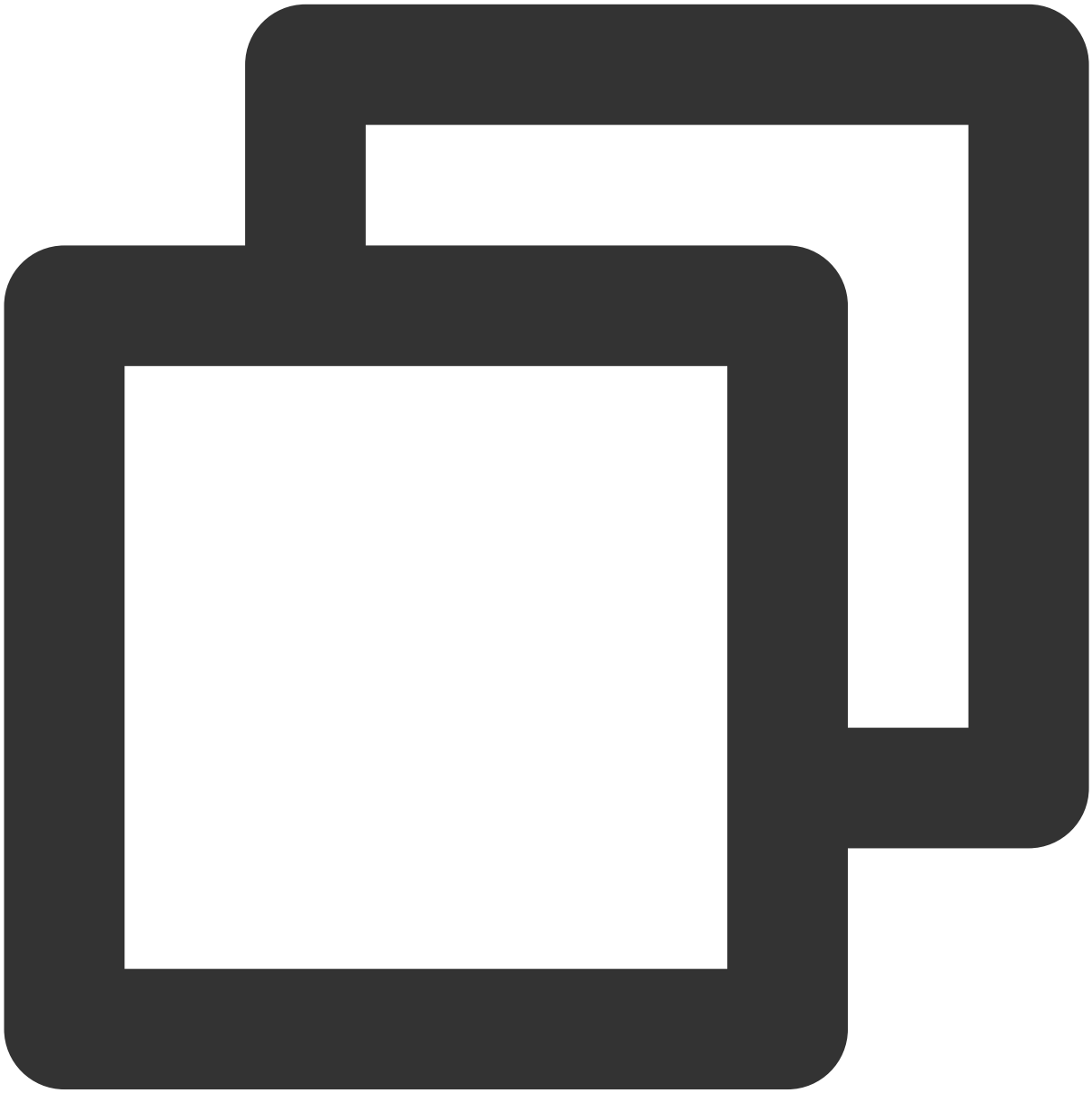


```
<template>
  <room ref="TUIRoomRef" @on-kicked-out-of-room="onKickedOutOfRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onKickedOutOfRoom({ roomId, message }) {
    console.log('普通成员被主持人踢出房间', roomId, message);
  }
```

```
</script>
```

onKickedOffLine

用户账号在其他地方登录，被踢下线。

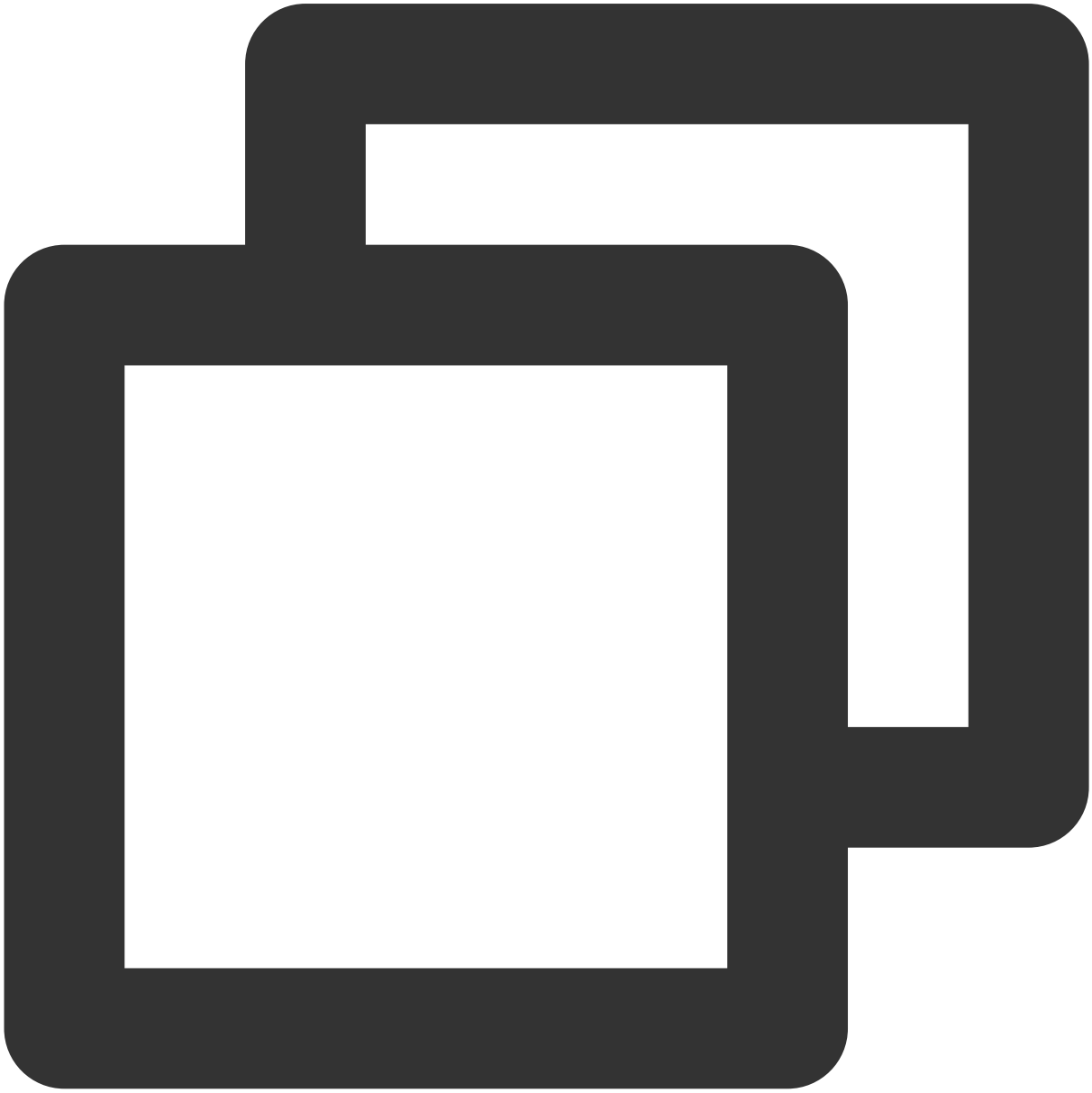


```
<template>
  <room ref="TUIRoomRef" @on-kick-off-line="onKickedOffLine"></room>
</template>

<script setup lang="ts">
```

```
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onKickedOffLine({ message }) {
    console.log('成员被踢下线', message);
  }
</script>
```

onUserSigExpired

用户 userSig 过期，请重新获取 userSig。



```
<template>
  <room ref="TUIRoomRef" @on-user-sig-expired="onUserSigExpired"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onUserSigExpired() {
    console.log('用户 userSig 过期，请重新生成 userSig.');
  }
</script>
```

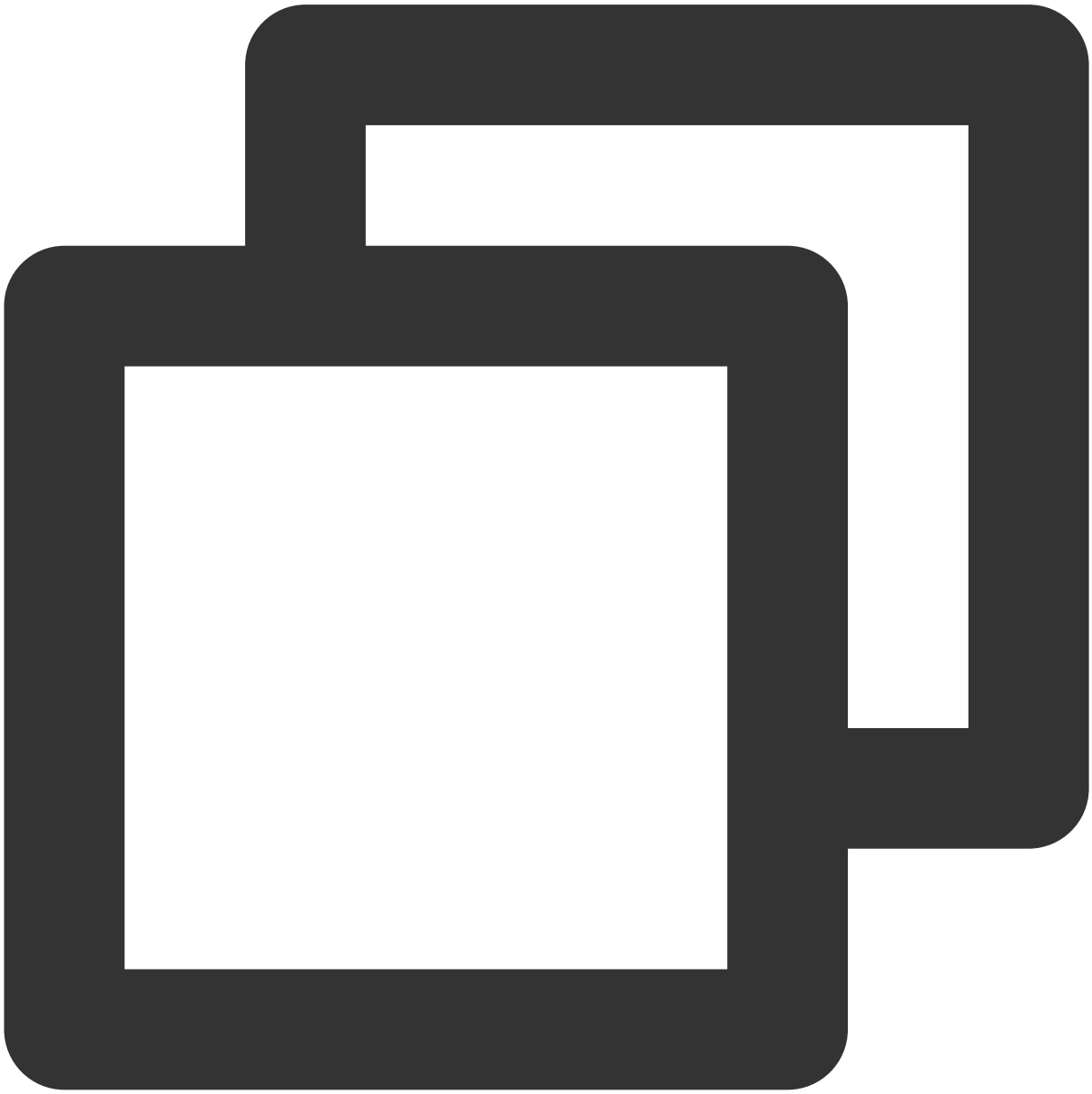
TUIRoomEngine

最近更新时间：2024-01-10 17:27:13

TUIRoomEngine 简介

TUIRoomEngine SDK 提供了房间管理，多人实时音视频互动，屏幕分享，成员管理，即时聊天等功能。

安装方式：



```
// 使用 npm
npm i @tencentcloud/tuiroom-engine-js --save

// 使用 pnpm
pnpm i @tencentcloud/tuiroom-engine-js --save

// 使用 yarn
yarn add @tencentcloud/tuiroom-engine-js
```

TUIRoomEngine API

TUIRoomEngine 静态方法

API	描述
once	监听 TUIRoomEngine ready 事件。 **注意：TUIRoomEngine.login 以外的所有方法必须在监听到 TUIRoomEngine ready 事件且 TUIRoomEngine.login 方法执行成功后执行。**
login	登录 TUIRoomEngine
setSelfInfo	设置当前用户基本信息（用户名、用户头像）
getSelfInfo	获取当前用户基本信息（用户名、用户头像）
logout	登出 TUIRoomEngine

roomEngine 房间管理 API

API	描述
createRoom	创建房间
enterRoom	进入房间
destroyRoom	销毁房间
exitRoom	离开房间
fetchRoomInfo	获取房间信息
updateRoomNameByAdmin	更新房间的名字（仅群主或者管理员可以调用）

updateRoomSpeechModeByAdmin	更新房间的发言模式（仅群主或者管理员可以调用）
getUserList	获取当前房间用户列表
getUserInfo	获取用户的详细信息

roomEngine 音视频 API

API	描述
setLocalVideoView	设置本地流的渲染位置
openLocalCamera	本地摄像头视频流采集
closeLocalCamera	关闭本地摄像头
openLocalMicrophone	打开本地麦克风
closeLocalMicrophone	关闭本地麦克风
updateVideoQuality	设置本地视频的参数
updateAudioQuality	设置本地音频的参数
startScreenSharing	开始屏幕共享
stopScreenSharing	停止屏幕共享
startPushLocalVideo	开始向远端推本地视频流
stopPushLocalVideo	停止向远端推本地视频流
startPushLocalAudio	开始向远端推本地音频流
stopPushLocalAudio	停止向远端推本地音频流
setRemoteVideoView	设置远端流渲染的区域
startPlayRemoteVideo	开始播放远端用户视频流
stopPlayRemoteVideo	停止播放远端用户视频流
muteRemoteAudioStream	停止远端用户的音频流

roomEngine 成员管理 API

API	描述

版权所有：腾讯云计算（北京）有限责任公司

openRemoteDeviceByAdmin	请求远端用户打开媒体设备
applyToAdminToOpenLocalDevice	参会者向主持人申请打开设备
closeRemoteDeviceByAdmin	关闭远端用户媒体设备
cancelRequest	取消已经发出的请求
responseRemoteRequest	回复远端用户的请求
changeUserRole	改变用户的角色
kickRemoteUserOutOfRoom	将用户踢出房间
disableDeviceForAllUserByAdmin	禁用/打开所有用户的媒体设备
disableSendingMessageForAllUser	禁止/允许所有用户发送消息
disableSendingMessageByAdmin	禁止/允许某个用户发送消息

roomEngine 麦位管理 API

API	描述
setMaxSeatCount	设置房间麦位最大值
getSeatList	获取麦位信息
takeSeat	获取麦位
leaveSeat	释放麦位
takeUserOnSeatByAdmin	邀请其他人上麦（仅房间主持人和管理员可调用此方法）
kickUserOffSeatByAdmin	将其他人踢下麦（仅房间主持人和管理员可调用此方法）
lockSeatByAdmin	锁定某个麦位状态（仅房间主持人和管理员可调用此方法）

roomEngine 消息发送 API

API	描述
sendTextMessage	发送文本消息
sendCustomMessage	发送自定义消息

版权所有：腾讯云计算（北京）有限责任公司

roomEngine 设备管理 API

API	描述
getCameraDevicesList	获取摄像头设备列表
getMicDevicesList	获取麦克风设备列表
getSpeakerDevicesList	获取扬声器设备列表
setCurrentCameraDevice	设置要使用的摄像头设备
setCurrentMicDevice	设置要使用的麦克风设备
setCurrentSpeakerDevice	设置要使用的扬声器设备
getCurrentCameraDevice	获取当前正在使用的摄像头设备
getCurrentMicDevice	获取当前正在使用的麦克风设备
getCurrentSpeakerDevice	获取当前正在使用的扬声器设备
startCameraDeviceTest	开始进行摄像头测试
stopCameraDeviceTest	停止摄像头测试

roomEngine 事件监听 API

API	描述
on	监听 TUIRoomEvents 事件
off	取消监听 TUIRoomEvents 事件

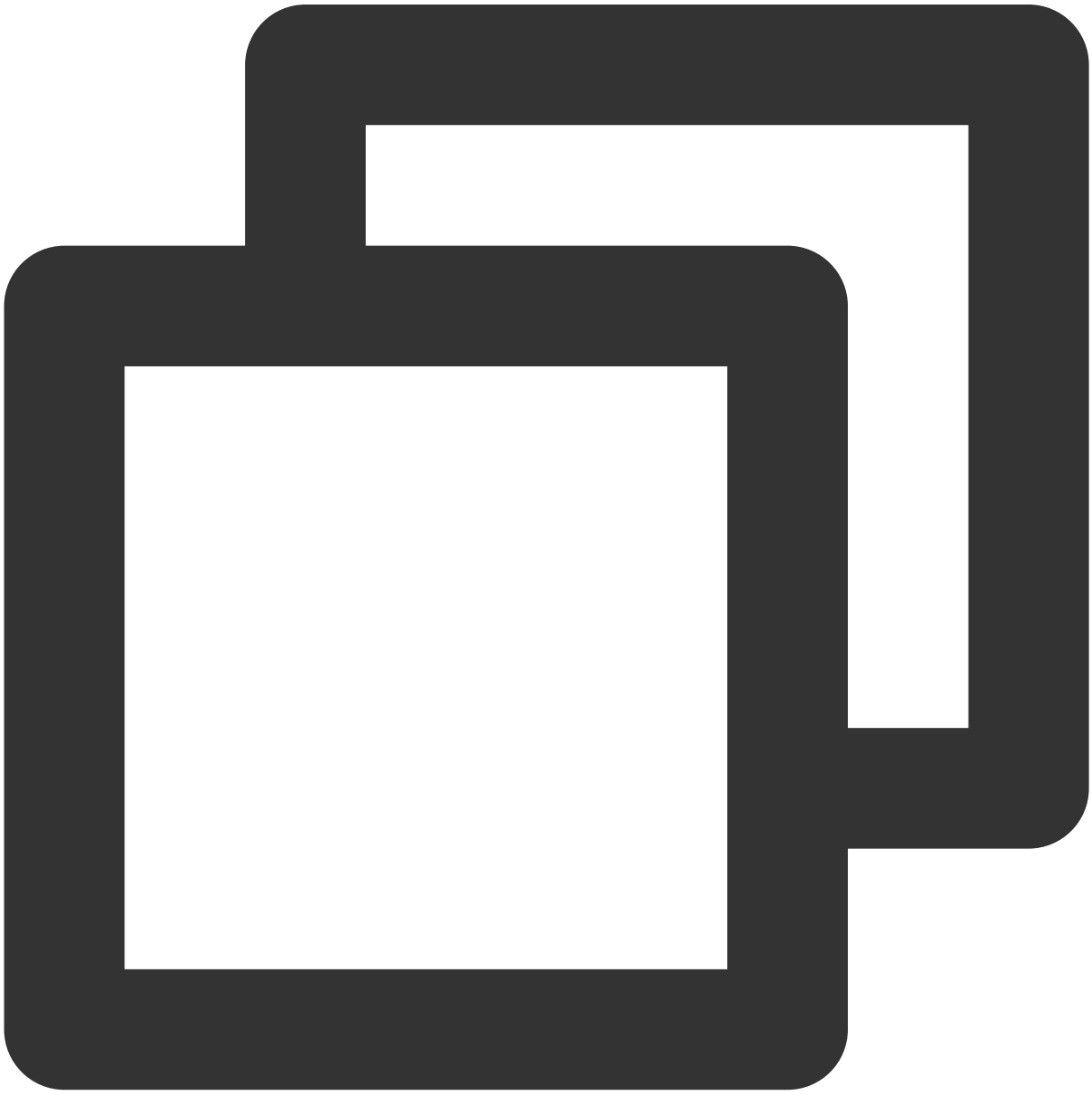
roomEngine 其他 API

API	描述
getTRTCCloud	获取 trtcCloud 实例
getTIM	获取 tim 实例

API 详情

once

监听 TUIRoomEngine 'ready' 事件



```
TUIRoomEngine.once('ready', () => {
  const roomEngine = new TUIRoomEngine();
```

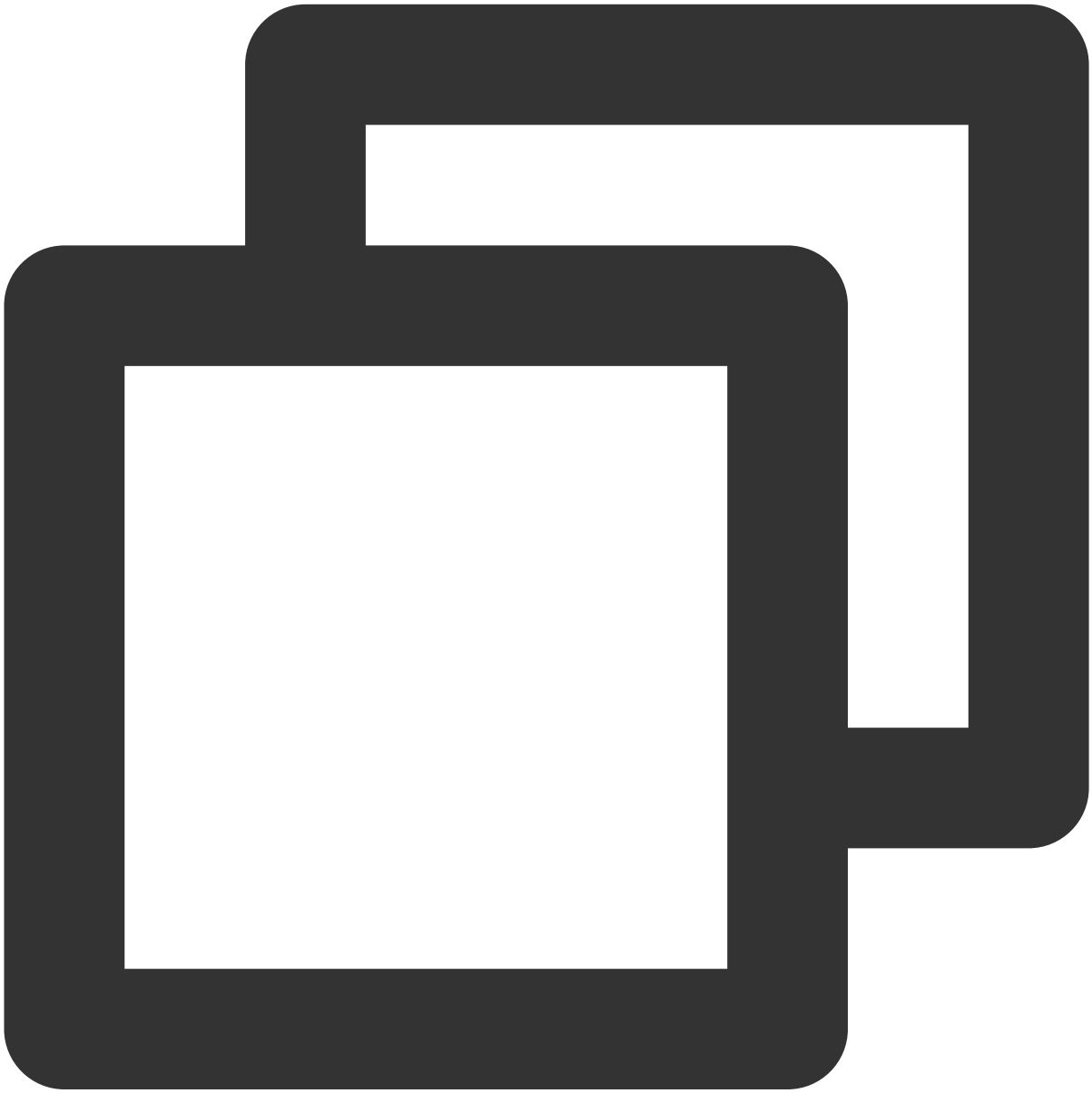
```
  await TUIRoomEngine.login({
    sdkAppId: 0,    // 填写您申请的 sdkAppId
    userId: '',     // 填写您业务对应的 userId
    userSig: '',    // 填写服务器或本地计算的 userSig
  });

  await roomEngine.createRoom({
    roomId: '12345',    // 填入您的房间 Id，注意房间 Id 要求为字符串类型
    name: 'Test Room',      // 填入您的房间名称，房间名称默认为 roomId，最长 30 字节
    roomType: TUIRoomType.kGroup, // 设置房间类型为 TUIRoomType.kGroup 类型
  });
});
```

login

说明：

v1.0.0 版本该接口命名为 TUIRoomEngine.init，v1.0.1 及其以上版本请使用 TUIRoomEngine.login 登录 TUIRoomEngine。

登录 TUIRoomEngine， 您必须在登录 TUIRoomEngine 之后才可以调用 TUIRoomEngine 及其实例的其他方法。



```
// 登录 TUIRoomEngine
await TUIRoomEngine.login({
 sdkAppId: 0,    // 填写您申请的 sdkAppId
 userId: '',     // 填写您业务对应的 userId
 userSig: '',    // 填写服务器或本地计算的 userSig
});
```

参数：

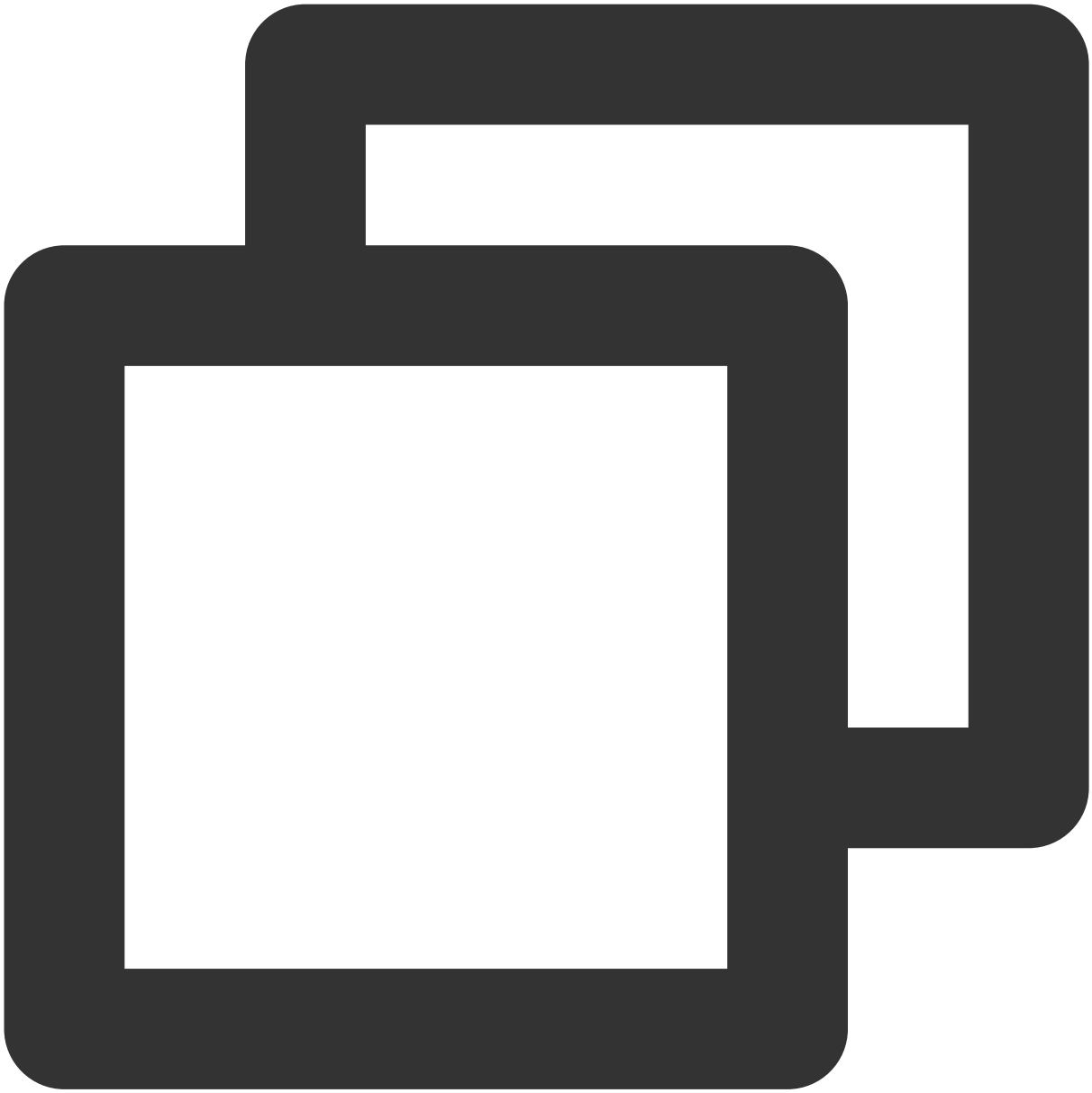
参数	类型	说明	默认值	含义

sdkAppId	number	必填	-	在 实时音视频控制台 单击 **应用管理** > **创建应用** 创建新应用之后，即可在 应用信息 中获取 sdkAppId 信息。
userId	string	必填	-	用户ID建议限制长度为32字节，只允许包含大小写英文字母(a-zA-Z)、数字(0-9)及下划线和连词符。
userSig	string	必填	-	userSig 签名 计算 userSig 的方式请参考 UserSig 相关。
tim	TIM	非必填	-	若您在接入 roomEngine 的同时想使用即时通信 SDK 的更多能力，您可以将创建的 tim 实例传入 TUIRoomEngine。tim 实例的创建方式请参考 TIM.create 。

Returns *Promise<void>*

setSelfInfo

设置当前用户基本信息（用户名、用户头像）



```
// 设置当前用户用户名及用户头像
await TUIRoomEngine.setSelfInfo({
 userName: '',      // 填写您的新用户名
 avatarUrl: '',     // 填写您的新头像地址
});
```

参数：

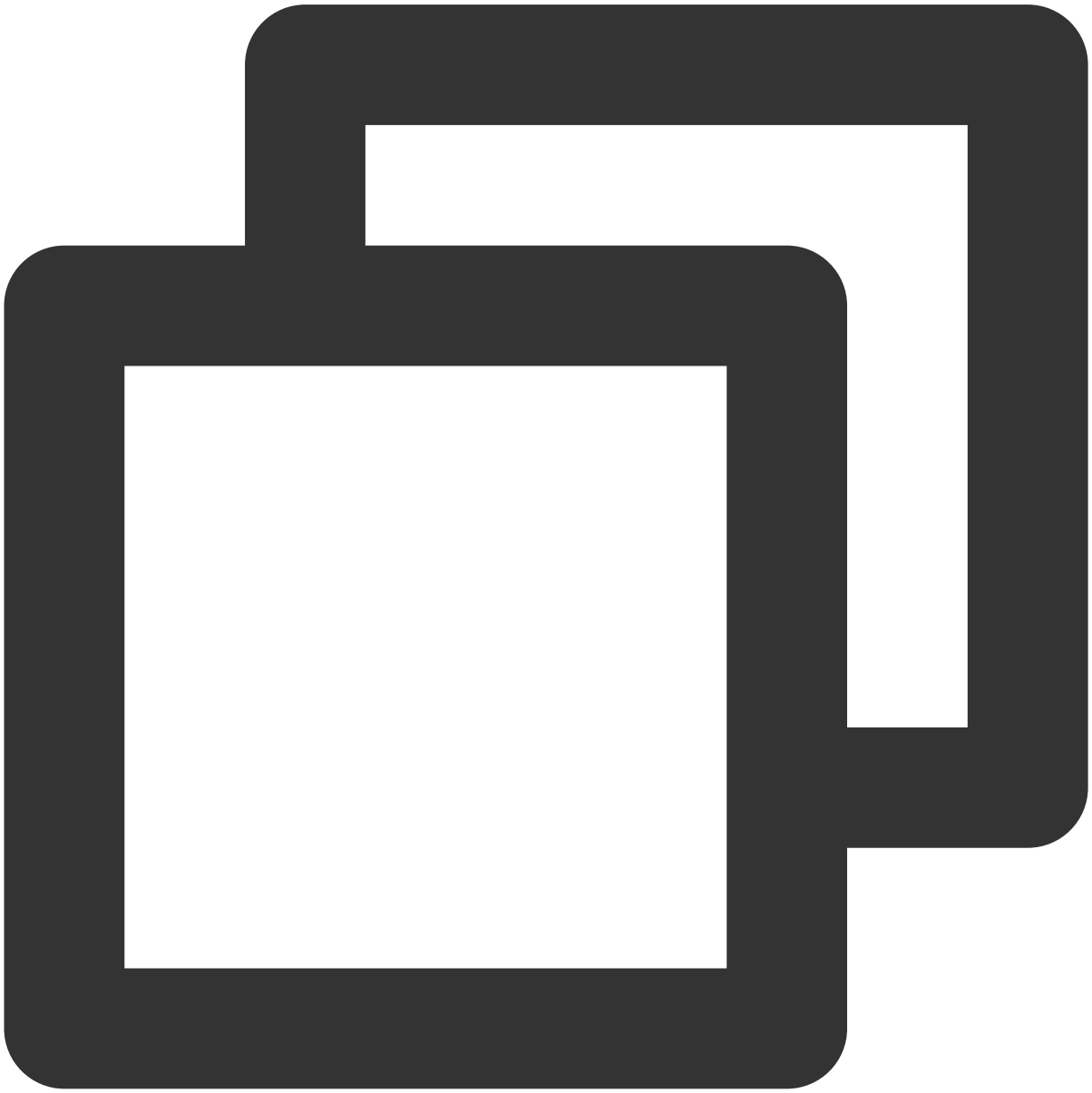
参数	类型	说明	默认值	含义
userName	string	必填	-	用户名

avatarUrl	string	必填	-	用户头像

Returns *Promise<void>*

getSelfInfo

获取当前用户基本信息（用户名、用户头像）



```
// 获取当前用户用户名及用户头像
```

```
const loginUserInfo = await TUIRoomEngine.getSelfInfo();
```

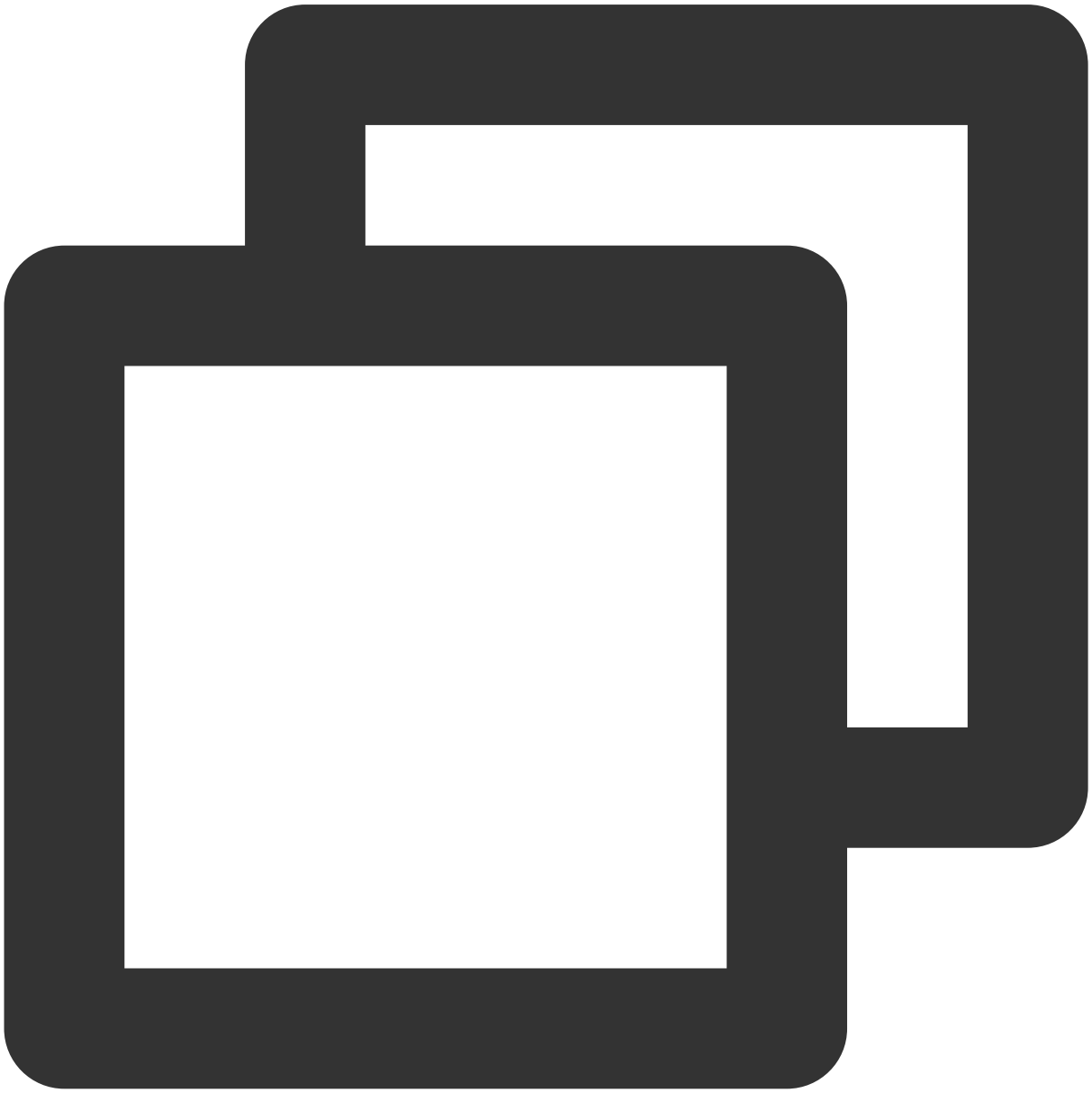
Returns *Promise<*TUILoginUserInfo*> loginUserInfo*

logout

说明：

该接口自 v1.0.1 版本支持。

登出 TUIRoomEngine

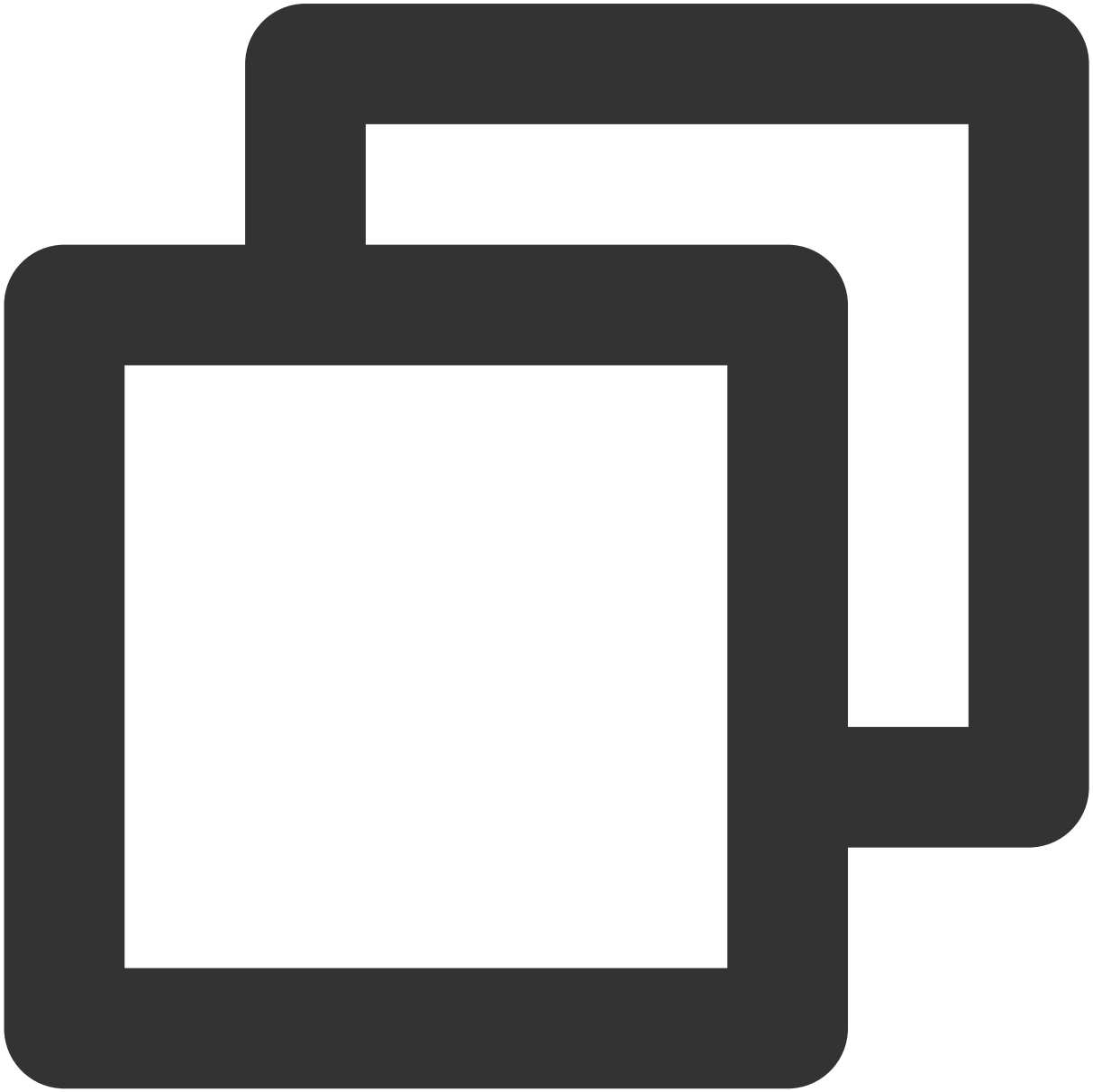


```
// 登出 TUIRoomEngine
await TUIRoomEngine.logout();
```

Returns *Promise<void>*

createRoom

主持人创建房间，调用 createRoom 的用户为房间的所有者。创建房间时可设置房间 Id，房间名称，房间类型，发言模式，是否允许加入用户开启音视频，发送消息等功能。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({
    roomId: '12345',    // 填入您的房间 Id，注意房间 Id 要求为字符串类型
    roomName: 'Test Room',      // 填入您的房间名称，房间名称默认为 roomId，最长 30 字节
    roomType: TUIRoomType.kConference, // 设置房间类型为 TUIRoomType.kConference 类型
    speechMode: TUISpeechMode.kFreeToSpeak, // 设置发言模式为自由发言模式
    isMicrophoneDisableForAllUser: false,  // 设置是否允许加入用户打开麦克风
    isCameraDisableForAllUser: false,  // 设置是否允许加入用户打开摄像头
    isMessageDisableForAllUser: false,  // 设置是否允许加入用户发送消息
});
```

参数：

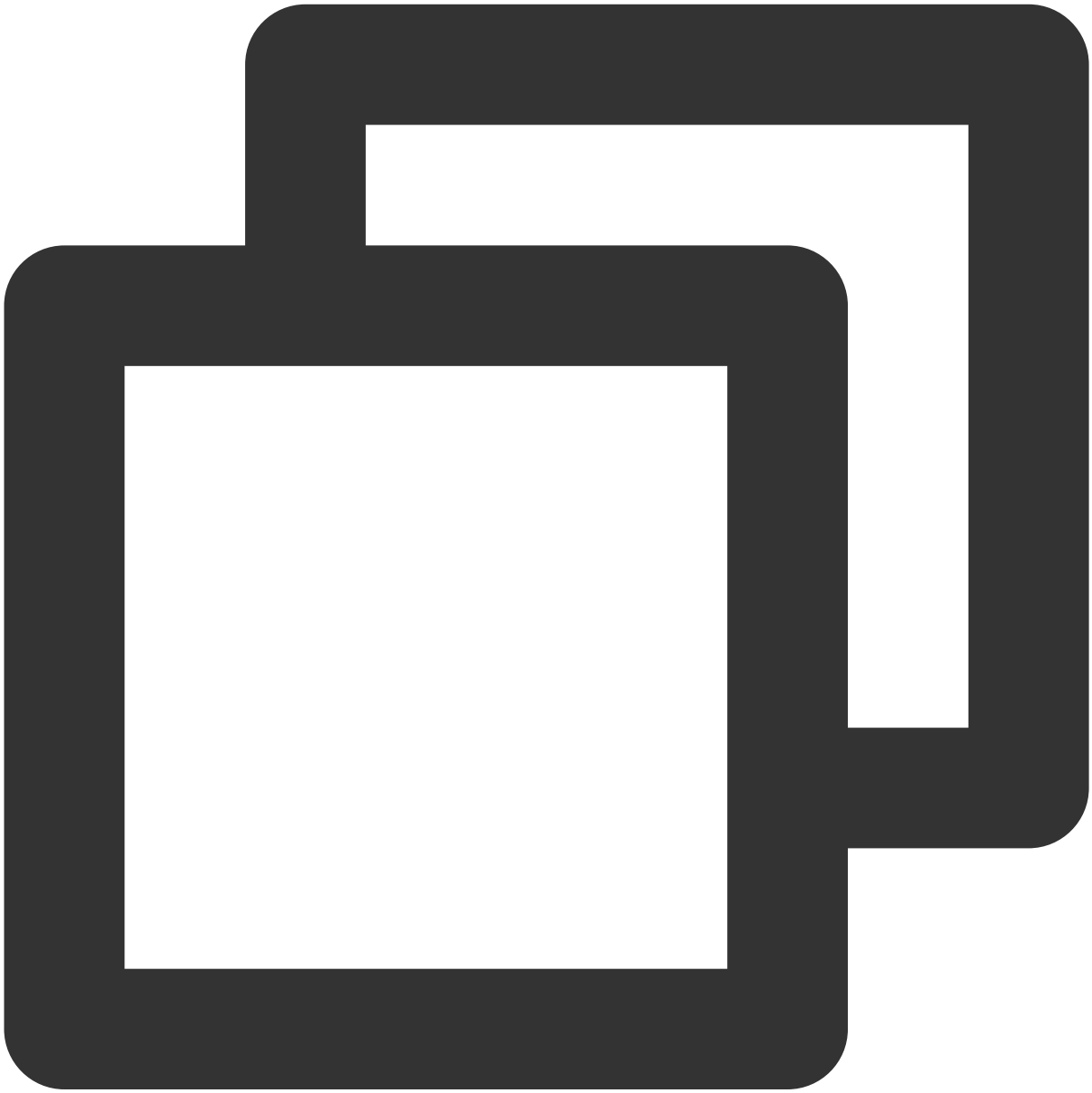
参数	类型	说明	默认值	含义
roomId	string	必填	-	房间 ID，roo 支持以下范围 大小写英文字 数字（0-9） 空格 ! # $ % & | ~ ,
roomName	string	选填	roomId	房间名称，默 符串
roomType	TUIRoomType	选填	TUIRoomType.kConference	房间类型 办公协同、医 景，roomTyp TUIRoomTyp 电商直播、语 TUIRoomTyp
speechMode	TUISpeechMode	选填	TUISpeechMode.kFreeToSpeak	房间内发言模 roomType 为 (教育及会议场 speechMode TUISpeechM 房默认可以打 speechMode TUISpeechM 房默认不打开 人申请打开摄 TUISpeechM TUISpeechM 相切换 speechMode

				TUISpeechM 用户进房后需 取打开摄像头 roomType 为 (直播场景) 时 speechMode TUISpeechM 需要主持人同 speechMode TUISpeechM 上麦需要主持 TUISpeechM TUISpeechM 模式可互相切
isMicrophoneDisableForAllUser	boolean	选填	false	是否开启全员
isCameraDisableForAllUser	boolean	选填	false	是否开启全员
isMessageDisableForAllUser	boolean	选填	false	是否允许成员
maxSeatCount	number	选填	-	最大麦位数量 roomType 为 (教育及会议场 做限制 roomType 为 (直播场景) 时 16
enableCDNStreaming	boolean	选填	false	是否开启 CD
cdnStreamDomain	string	选填	''	直播推流域名

Returns *Promise<void>*

enterRoom

进入房间接口



```
const roomEngine = new TUIRoomEngine();
const roomInfo = await roomEngine.enterRoom({
    roomId: '12345',
});
```

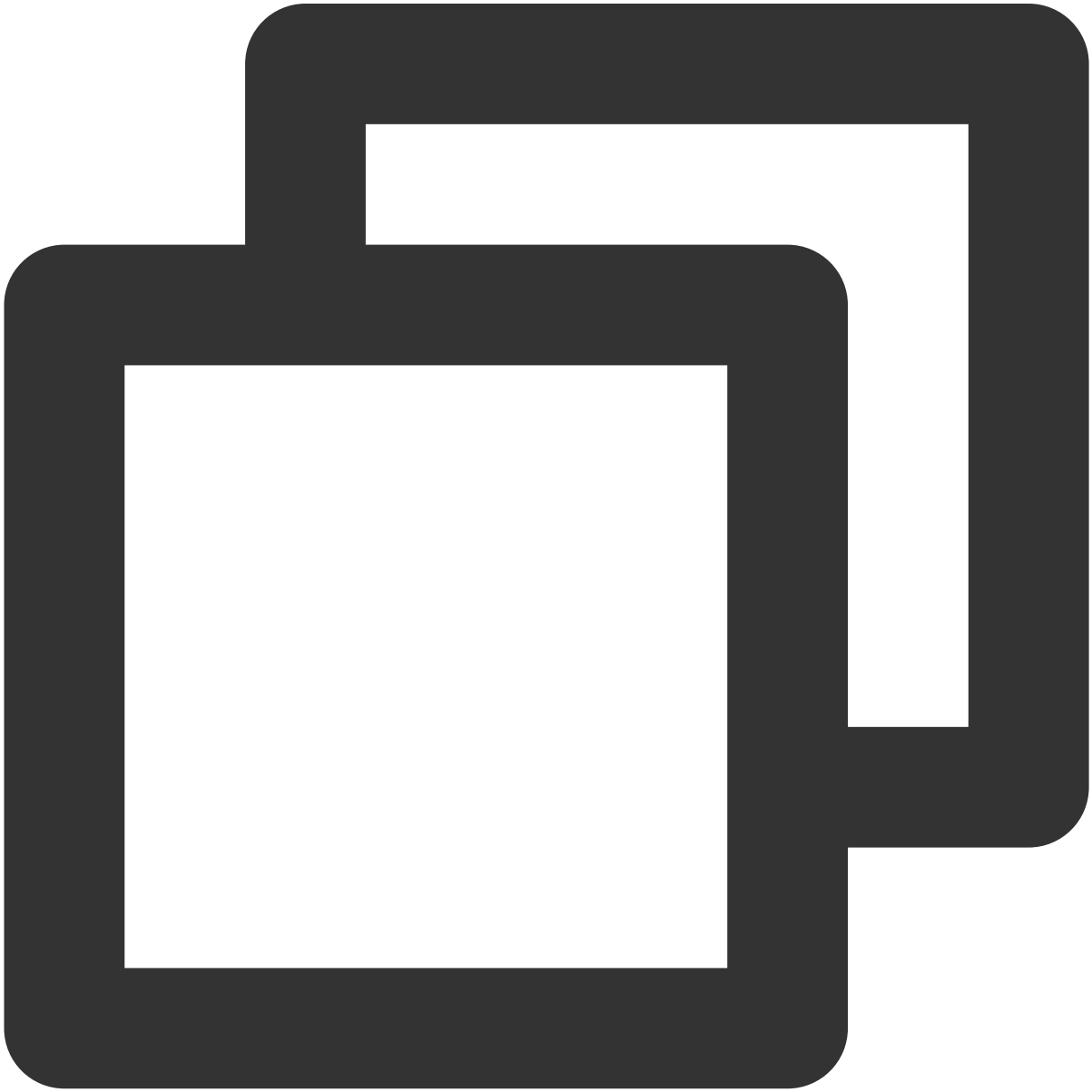
参数：

参数	类型	说明	默认值	含义
roomId	string	必填	-	房间号

Returns Promise<TUIRoomInfo> roomInfo
该接口返回当前房间信息

destroyRoom

解散房间接口，解散房间必须由房间所有者发起，解散房间之后房间不可进入。

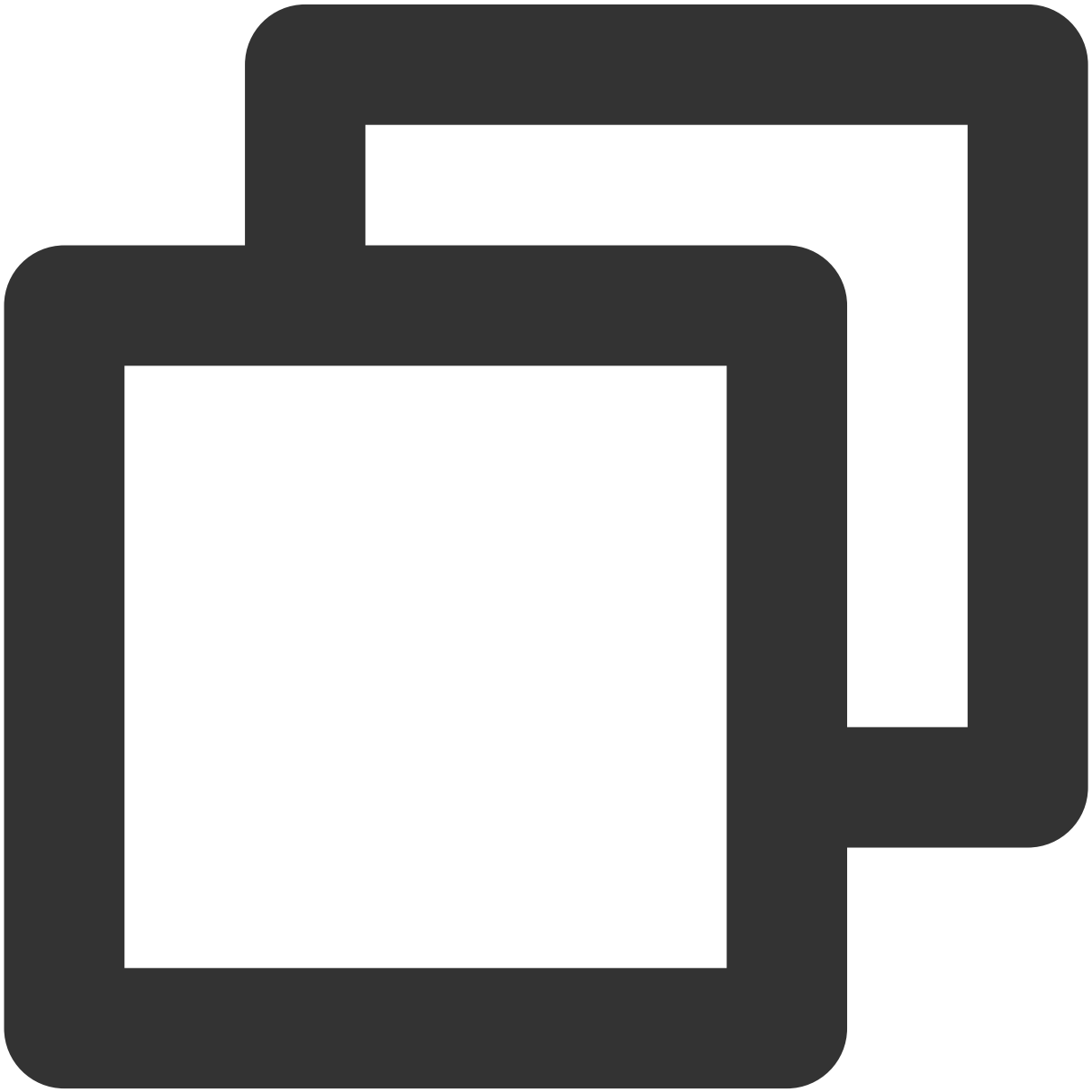


```
const roomEngine = new TUIRoomEngine();
await roomEngine.destroyRoom();
```

Returns *Promise<void>*

exitRoom

离开房间接口，用户在执行 enterRoom 之后可通过 exitRoom 离开房间。

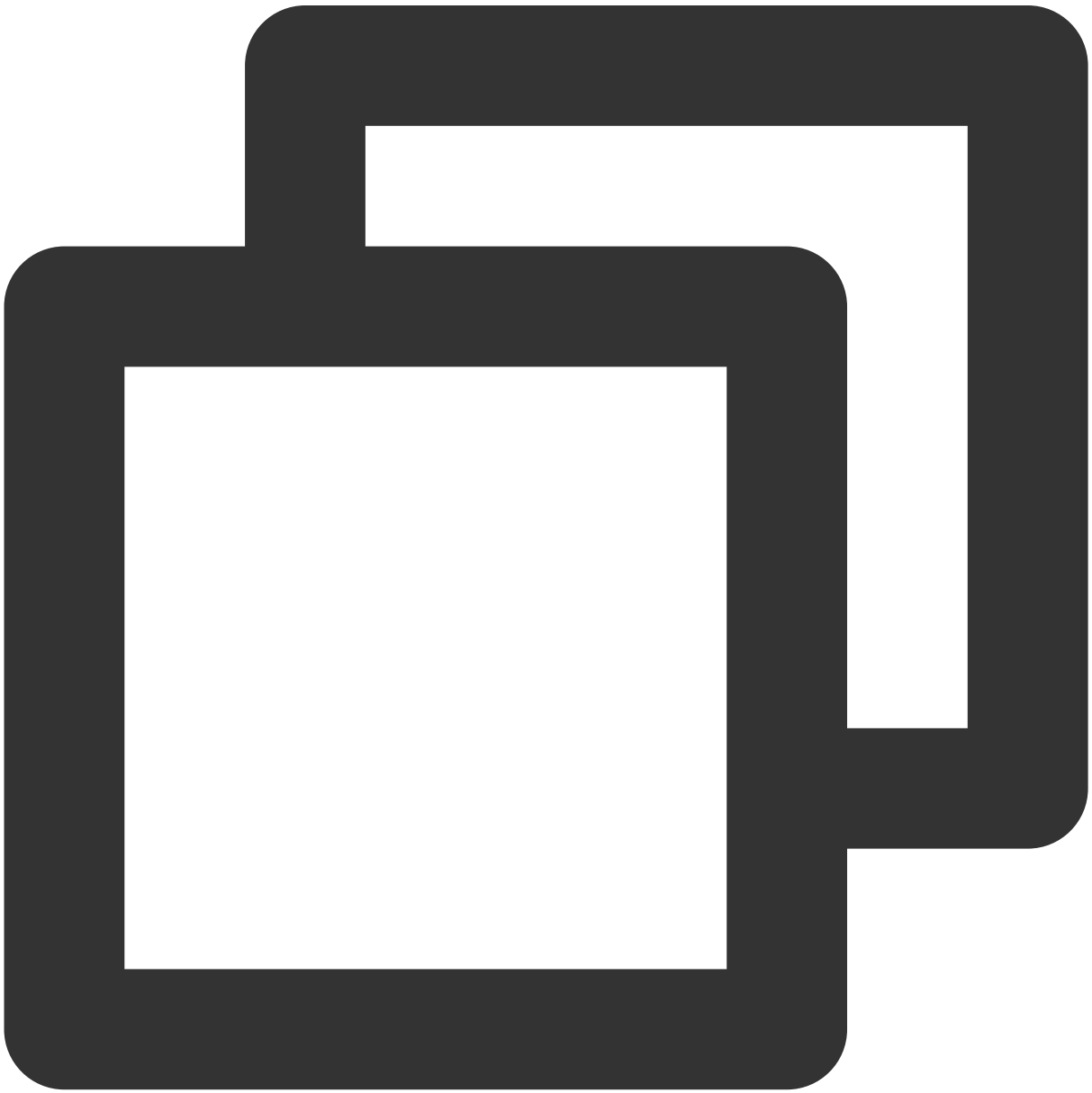


```
const roomEngine = new TUIRoomEngine();
await roomEngine.exitRoom();
```

Returns *Promise<void>*

fetchRoomInfo

获取房间信息

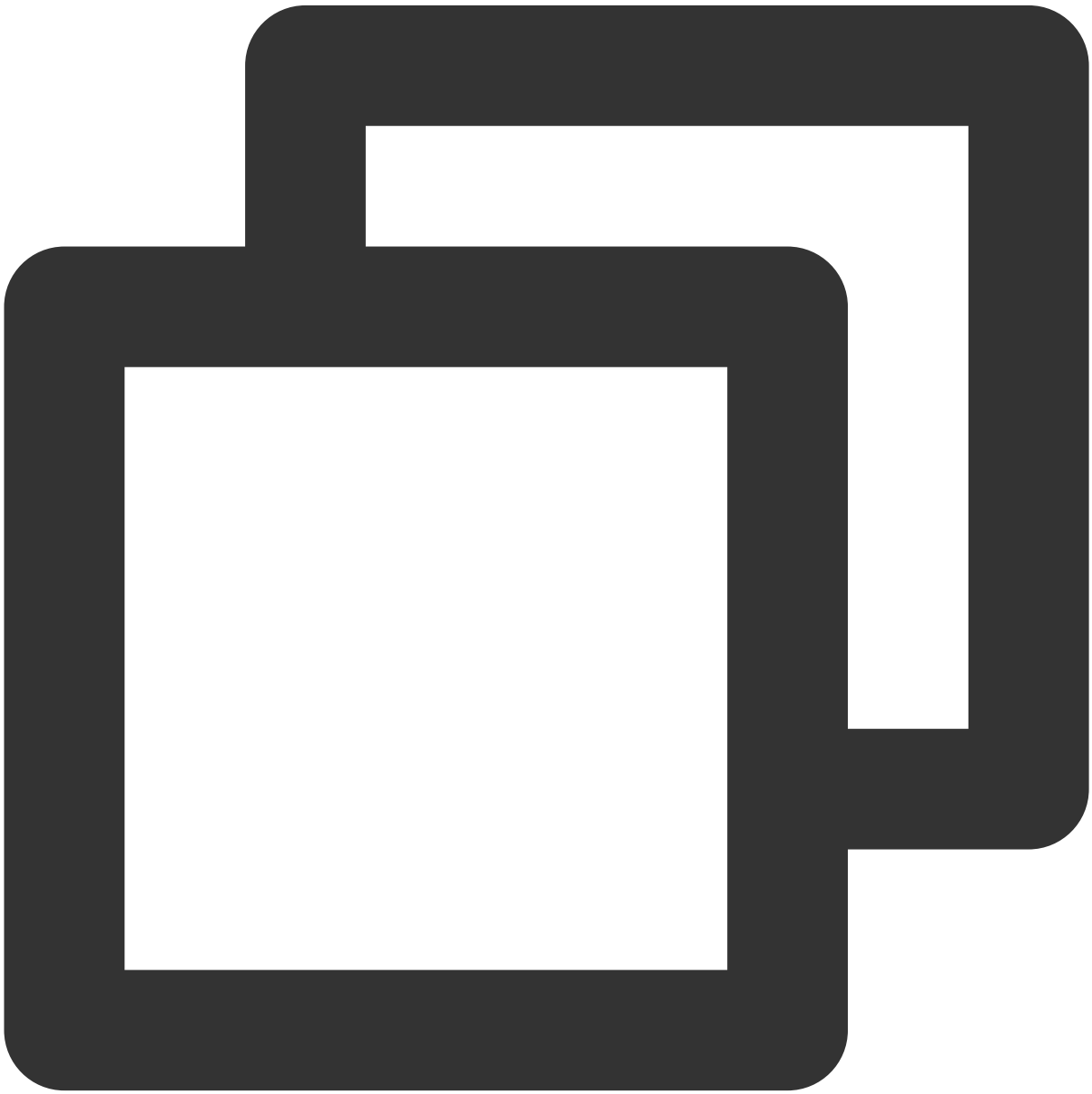


```
const roomEngine = new TUIRoomEngine();
const roomInfo = roomEngine.fetchRoomInfo();
```

Returns：*Promise<TUIRoomInfo> roomInfo*

updateRoomNameByAdmin

更新当前房间的名字（仅群主或者管理员可以调用）



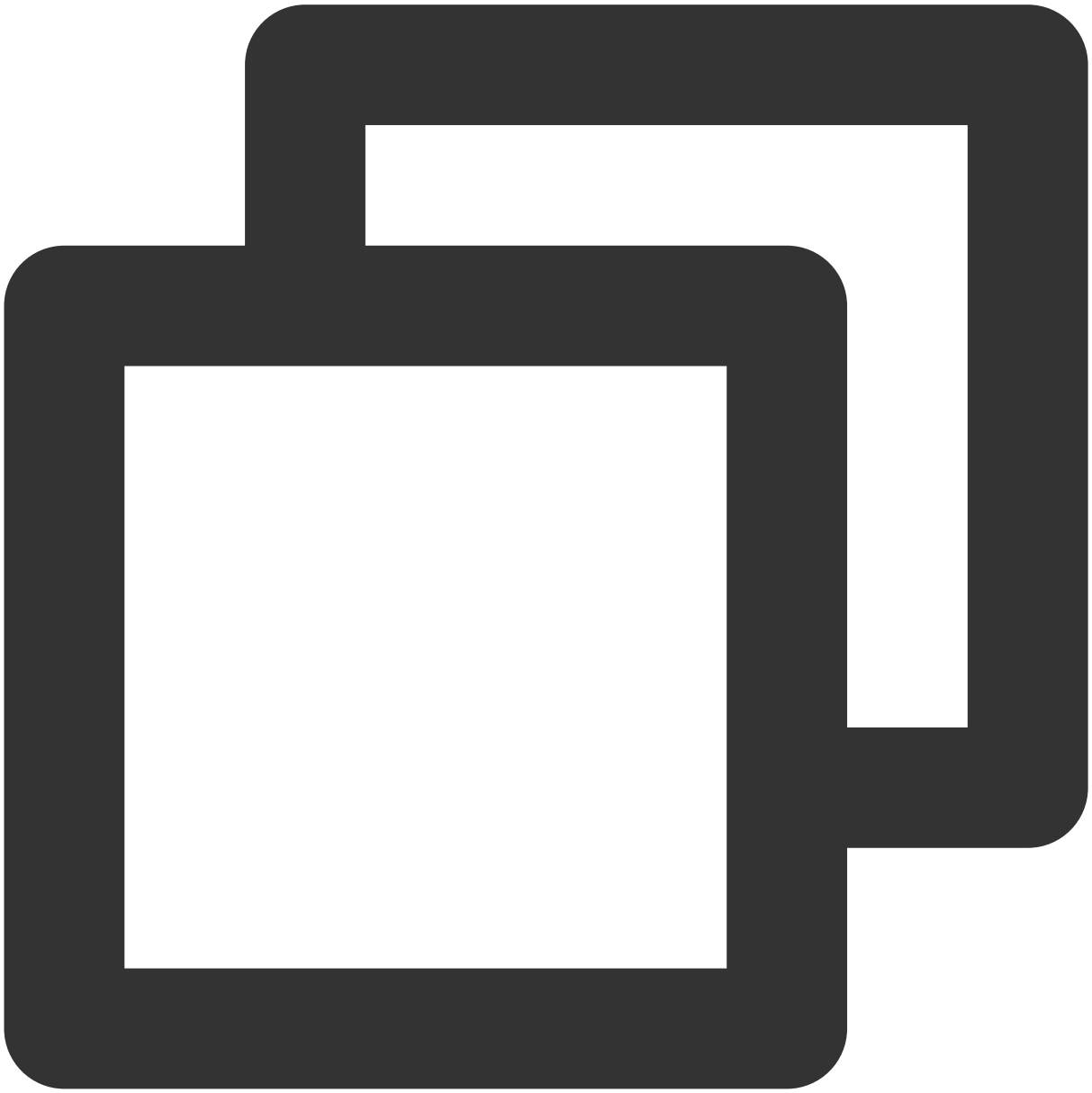
```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomNameByAdmin({ roomName: '新的名字' });
```

参数：

参数	类型	说明	默认值	含义
roomName	string	必填	-	更新房间的名字，要求 roomName 不为空字符串

updateRoomSpeechModeByAdmin

更新房间的发言模式（仅群主或者管理员可以调用）



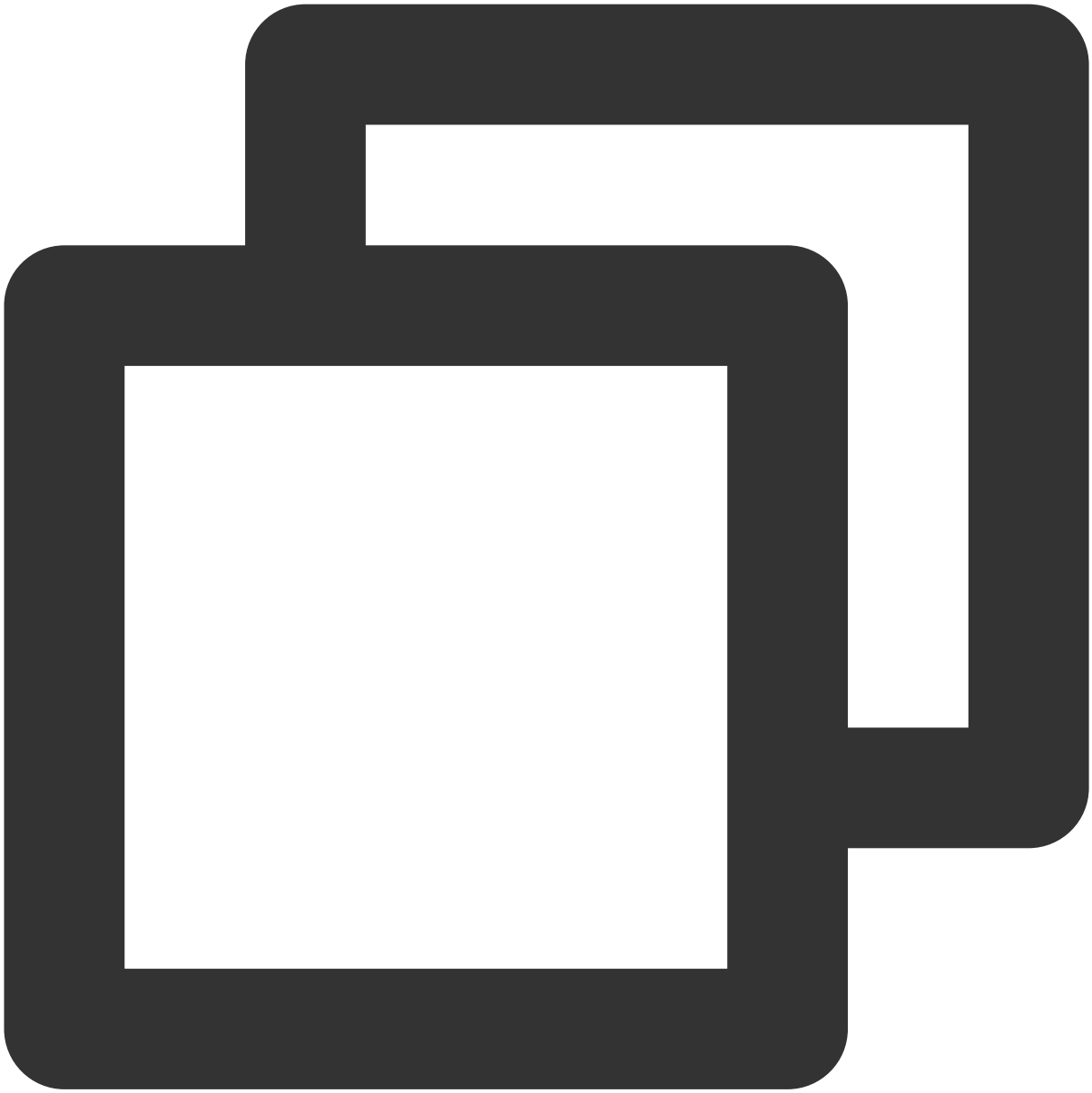
```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomSpeechModeByAdmin({
  speechMode: TUISpeechMode.kSpeakAfterTakingSeat  // 更新为上麦后发言模式
});
```

参数：

参数	类型	说明	默认值	含义
speechMode	TUISpeechMode	必填	-	更新房间的发言模式

getUserList

获取当前房间用户列表，注意该接口一次拉取的用户列表量最大为 100 个



```
const roomEngine = new TUIRoomEngine();
const userList = [];
let result;
do {
  result = await globalProperties.roomEngine.getUserList();
  userList.push(...result.userInfoList);
} while (result.nextSequence !== 0)
```

参数：

参数	类型	说明	默认值	含义

nextSequence	number	选填	0	偏移量，默认从 0 开始拉取用户

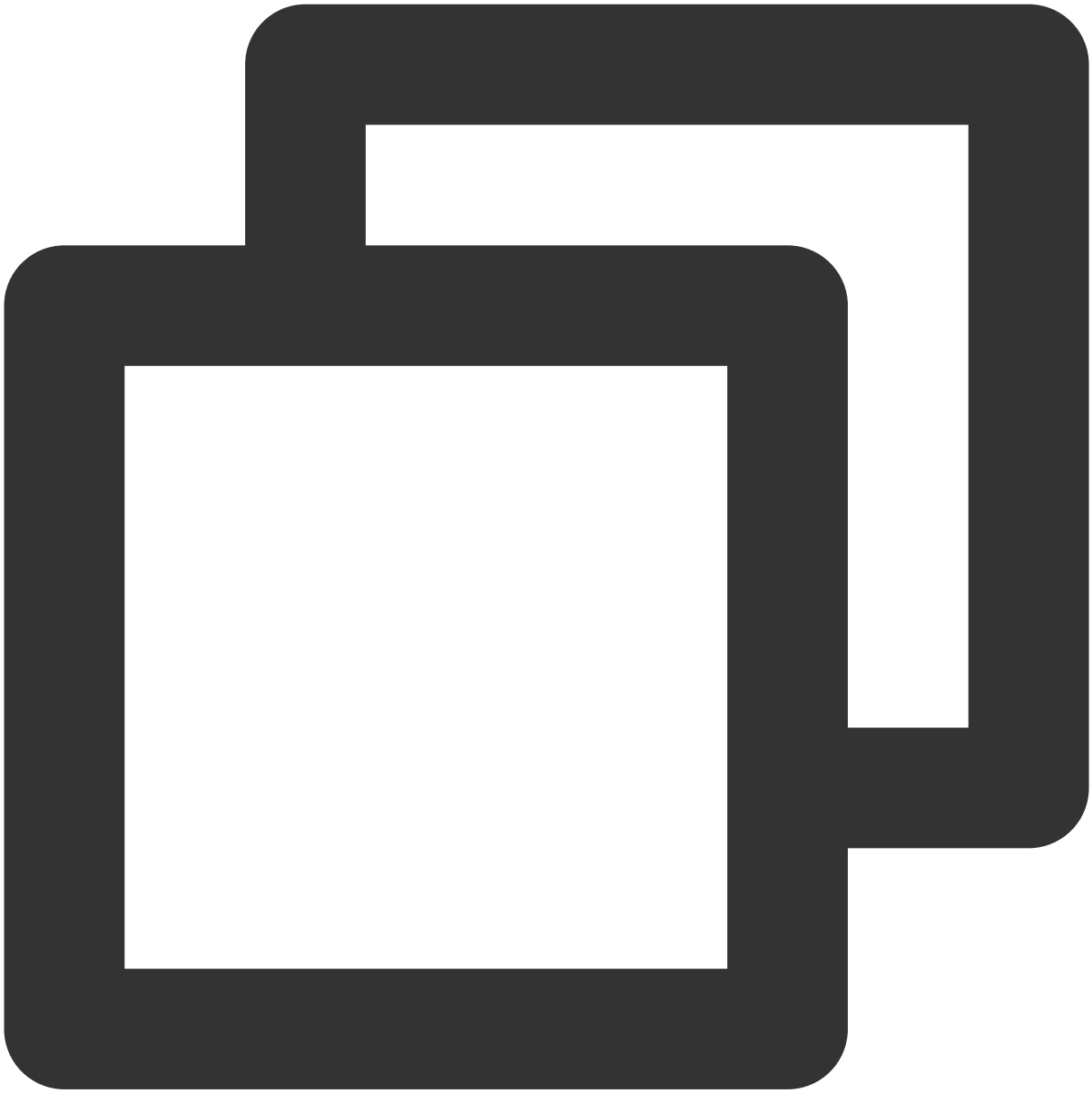
Returns：*Promise<Array> result*

result.nextSequence 下一次拉取群组用户的偏移量，如果 result.nextSequence 为 0 则代表 userList 全部拉取完毕

result.userInfoList 本次拉取的 userList

getUserInfo

获取用户的详细信息



```
const roomEngine = new TUIRoomEngine();
const userList = [];
const userInfo = await roomEngine.getUserInfo({
    userId: 'user_12345',
});
```

参数：

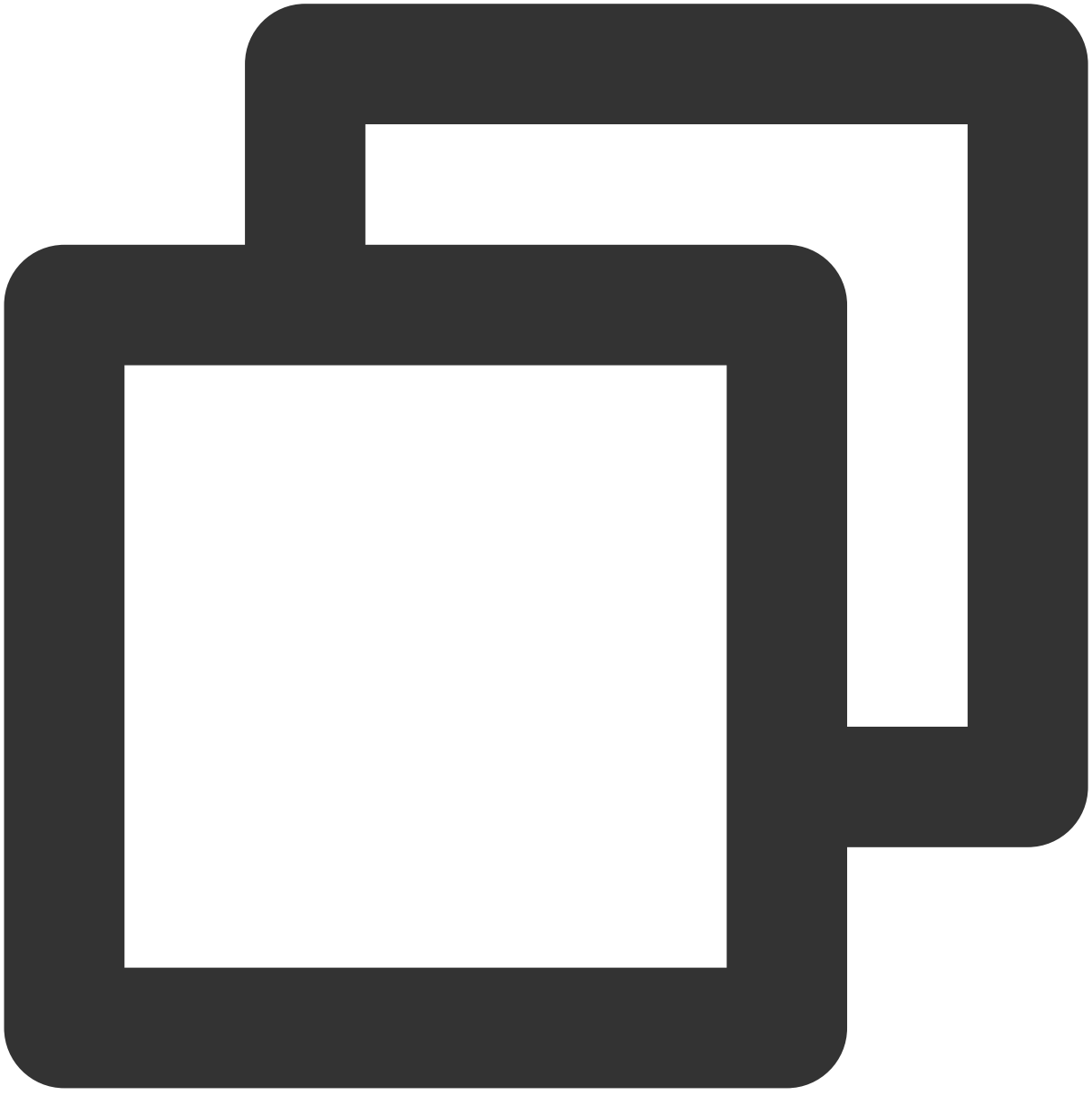
参数	类型	说明	默认值	含义

userId	string	必填	-	根据 userId 获取该用户的详细信息

Returns：*Promise<Array<TUIUserInfo>> userInfoList*
该接口返回指定用户的用户信息

setLocalVideoView

设置本地流的渲染位置



```
const roomEngine = new TUIRoomEngine();

// 设置本地摄像头流的播放区域为 id 是 'preview-camera' 的 div 元素
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});

// 设置本地屏幕分享流的播放区域为 id 是 'preview-screen' 的 div 元素
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kScreenStream,
  view: 'preview-screen',
});
```

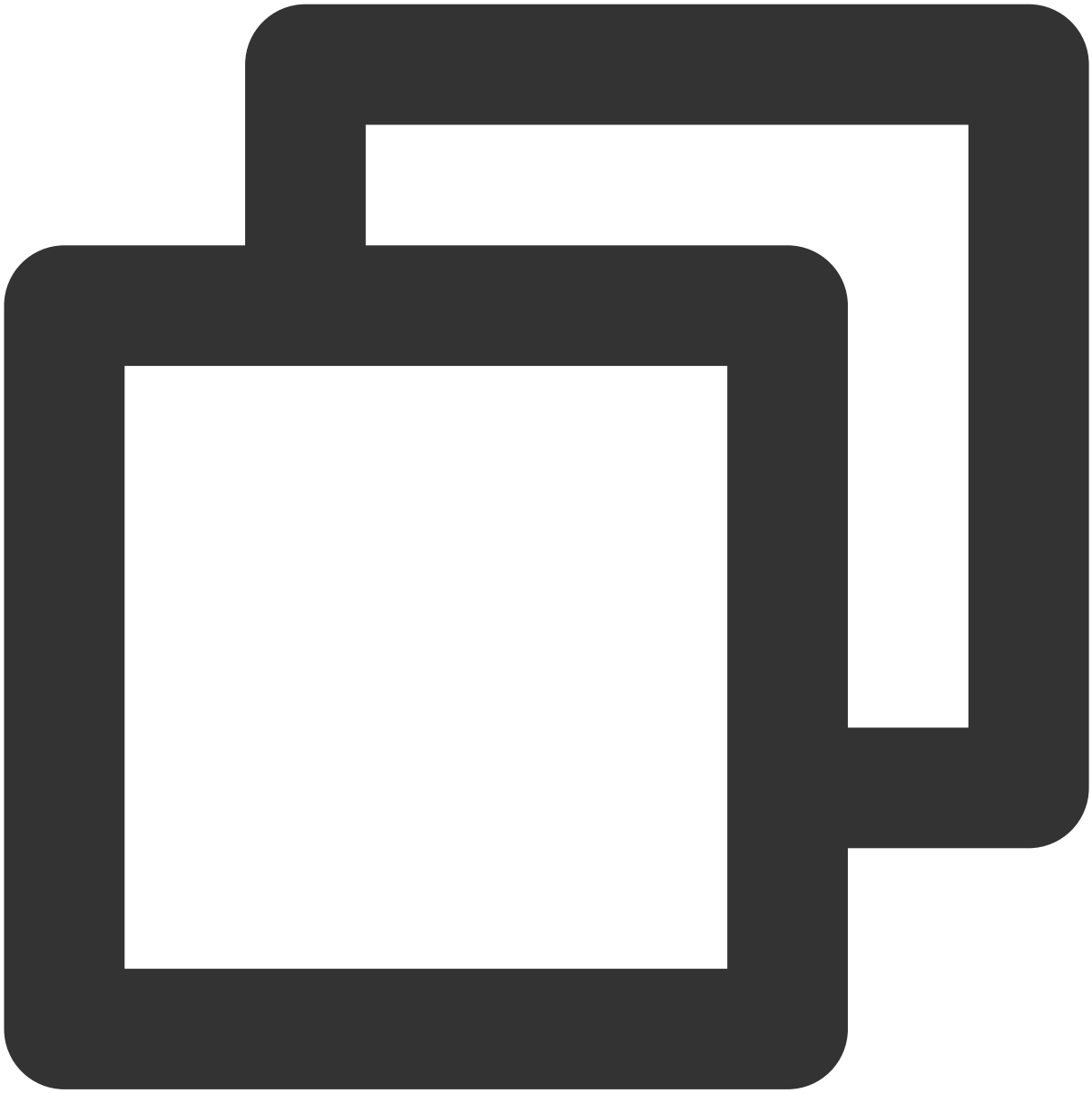
参数：

参数	类型	说明	默认值	含义
streamType	TUIVideoStreamType	必填	-	本地流类型
view	string	必填	-	streamType 对应的流渲染的 div 元素的 id

Returns：*Promise<void>*

openLocalCamera

打开本地摄像头，开始视频流采集

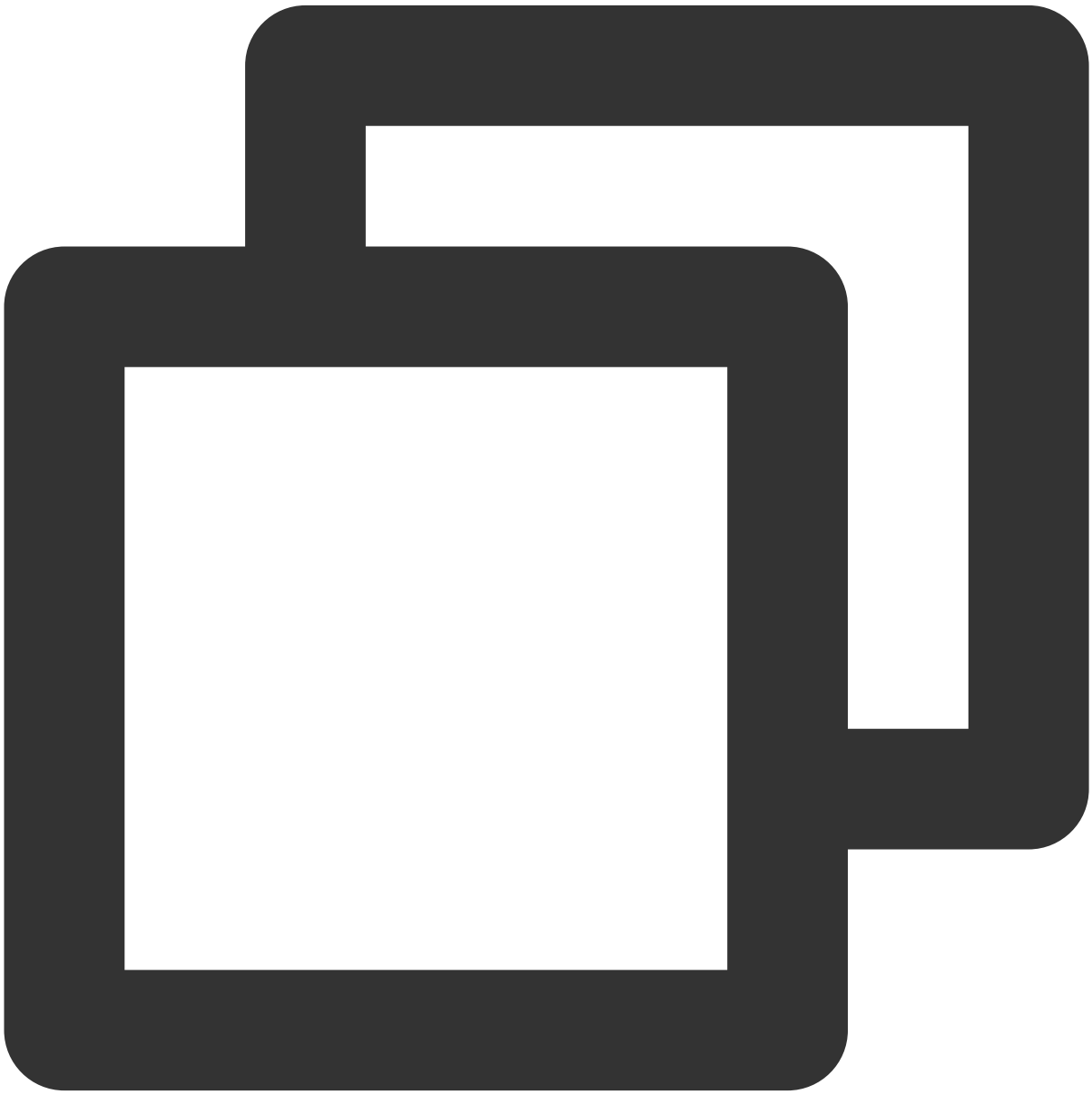


```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});
await roomEngine.openLocalCamera();
```

Returns：*Promise<void>*

closeLocalCamera

关闭本地摄像头

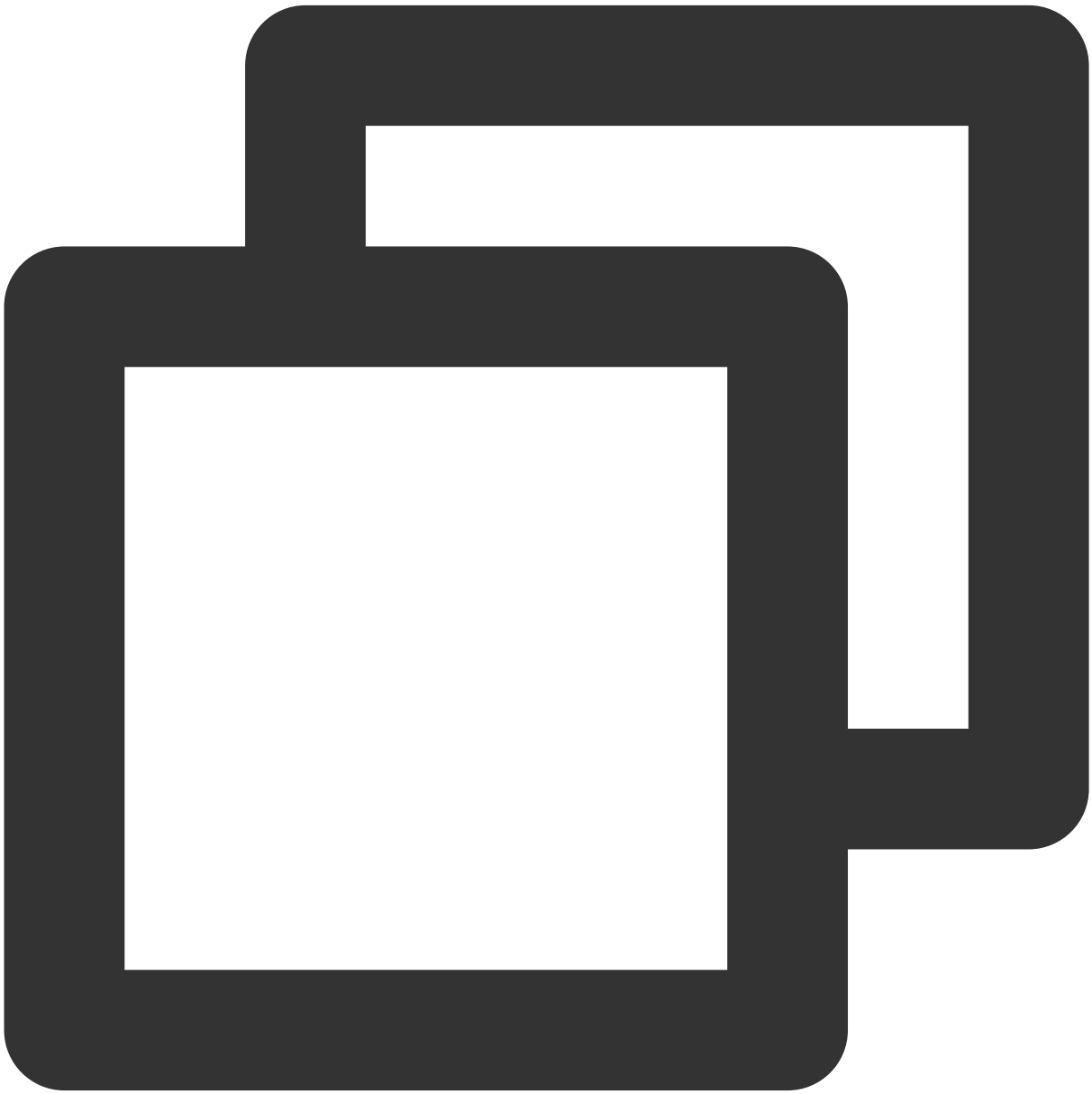


```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeLocalCamera();
```

Returns：*Promise<void>*

openLocalMicrophone

打开本地麦克风, 开始采集音频流

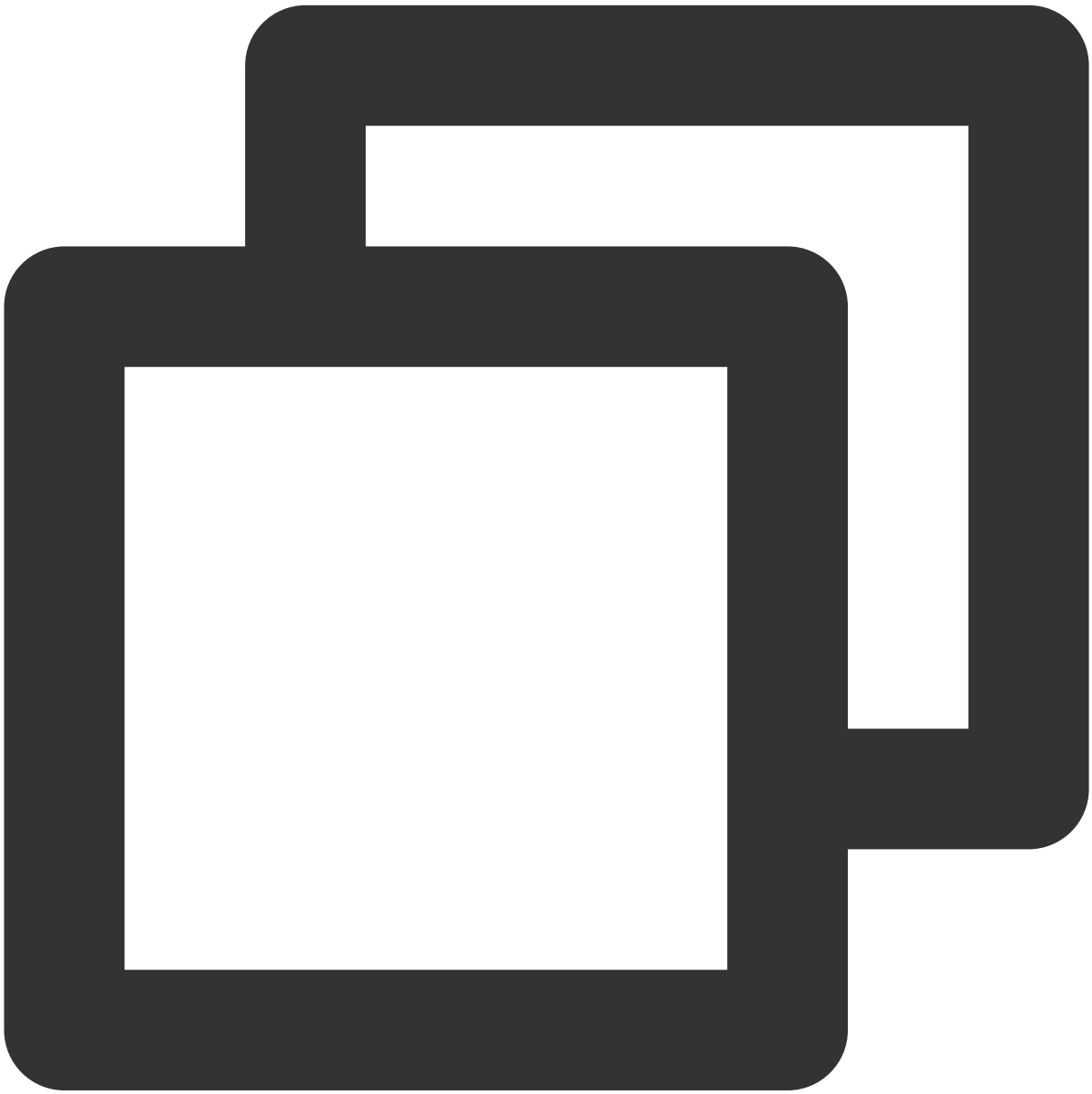


```
const roomEngine = new TUIRoomEngine();
await roomEngine.openLocalMicrophone();
```

Returns：*Promise<void>*

closeLocalMicrophone

关闭本地麦克风

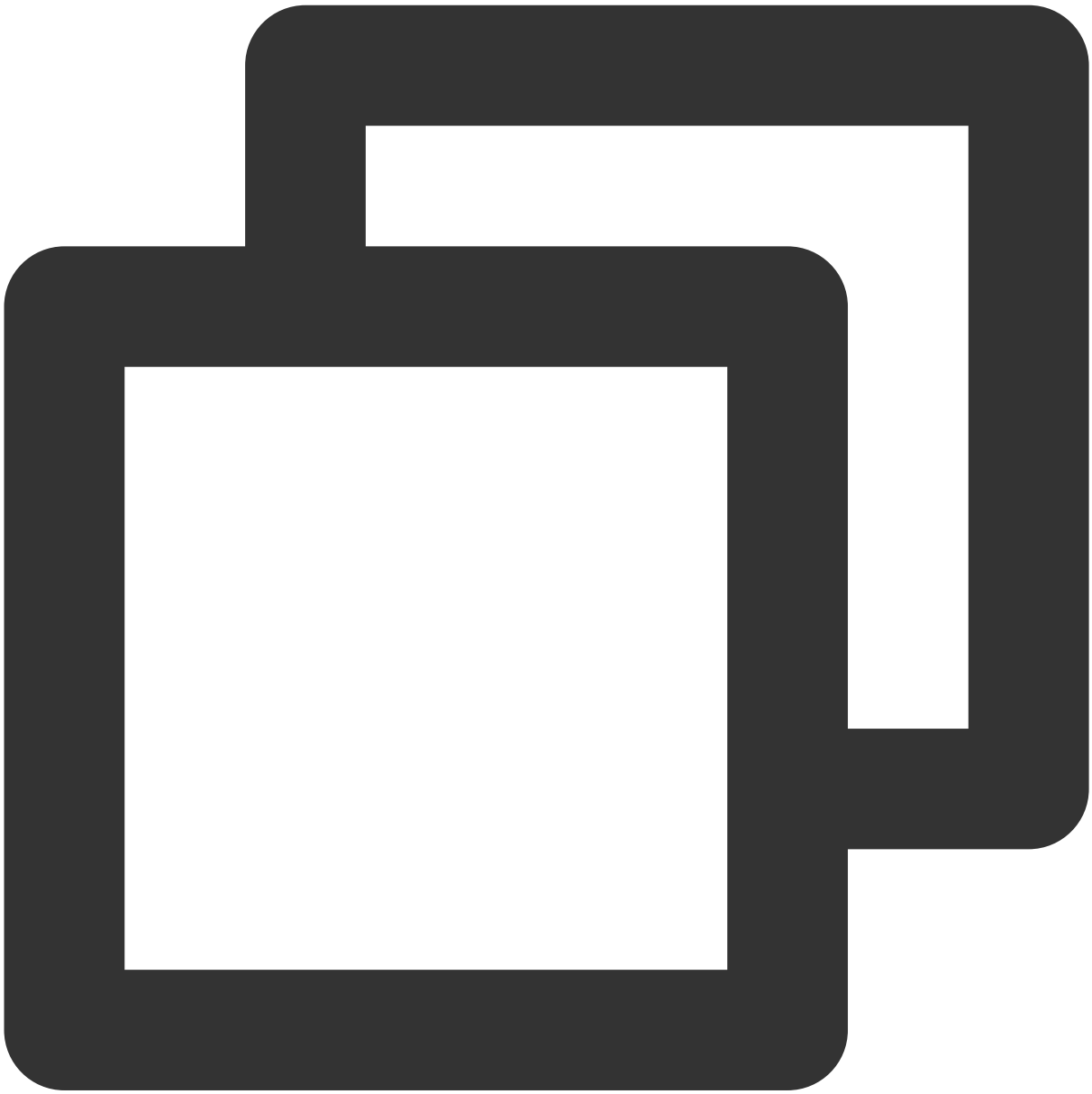


```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeLocalMicrophone();
```

Returns：*Promise<void>*

updateVideoQuality

设置本地视频流的编码参数，默认为 TUIVideoProfile.kVideoQuality_720P



```
const roomEngine = new TUIRoomEngine();
await roomEngine.updateVideoQuality({
    quality: TUIVideoQuality.kVideoQuality_540p,
});
```

参数：

参数	类型	说明	默认值	含义
quality	TUIVideoQuality	必填	-	清晰 TUIVideoProfile.kVideoQuality_360P 标清 TUIVideoProfile.kVideoQuality_540P 高清 TUIVideoProfile.kVideoQuality_720P 超清 TUIVideoProfile.kVideoQuality_1080P

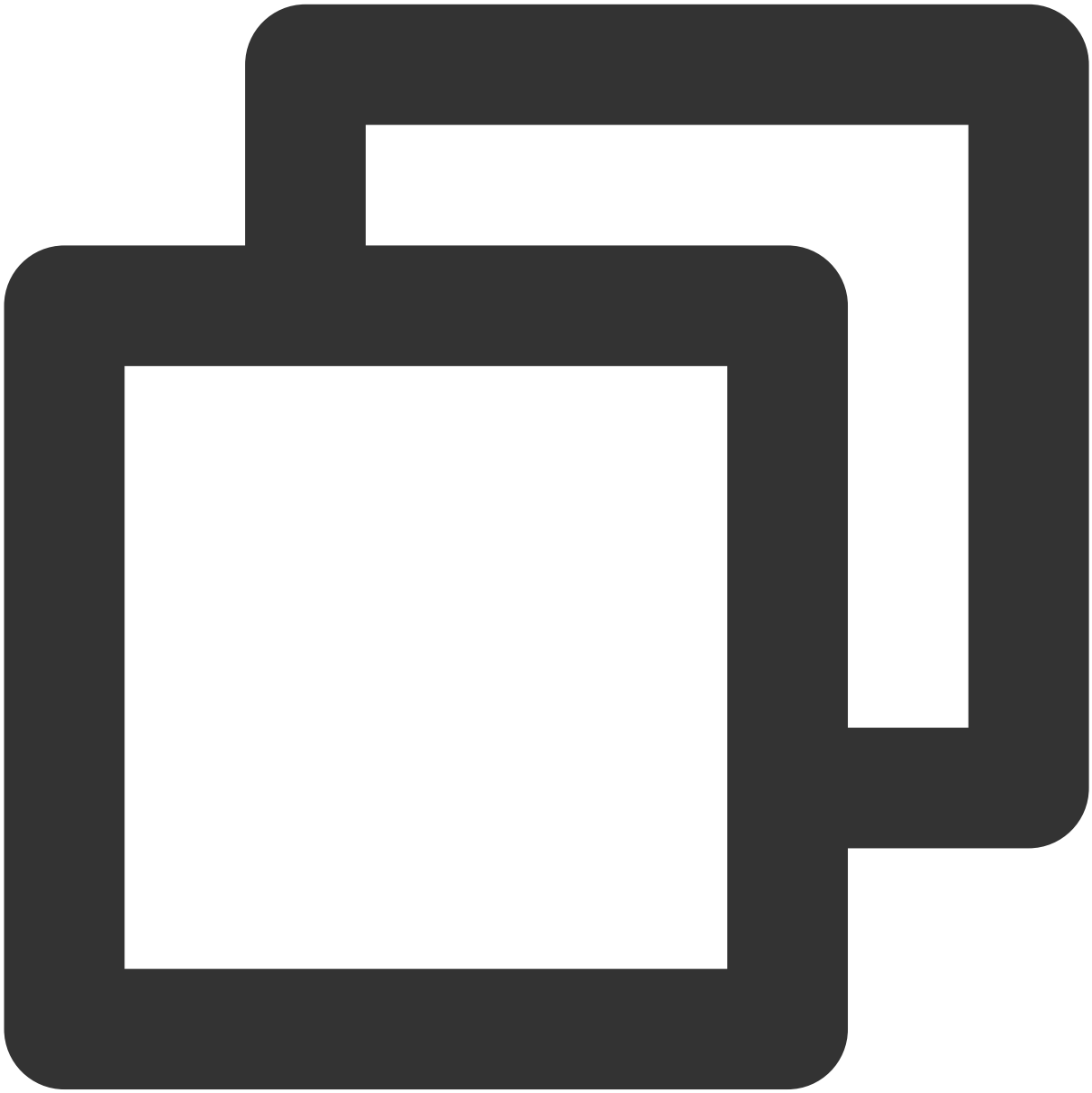
Returns：*Promise<void>*

updateAudioQuality

设置本地音频的参数

注意：

该方法需要在 openLocalMicrophone 之前进行设置，否则不会生效。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalAudioProfile({
    audioProfile: TUIAudioProfile.kAudioProfileSpeech,
});
```

参数：

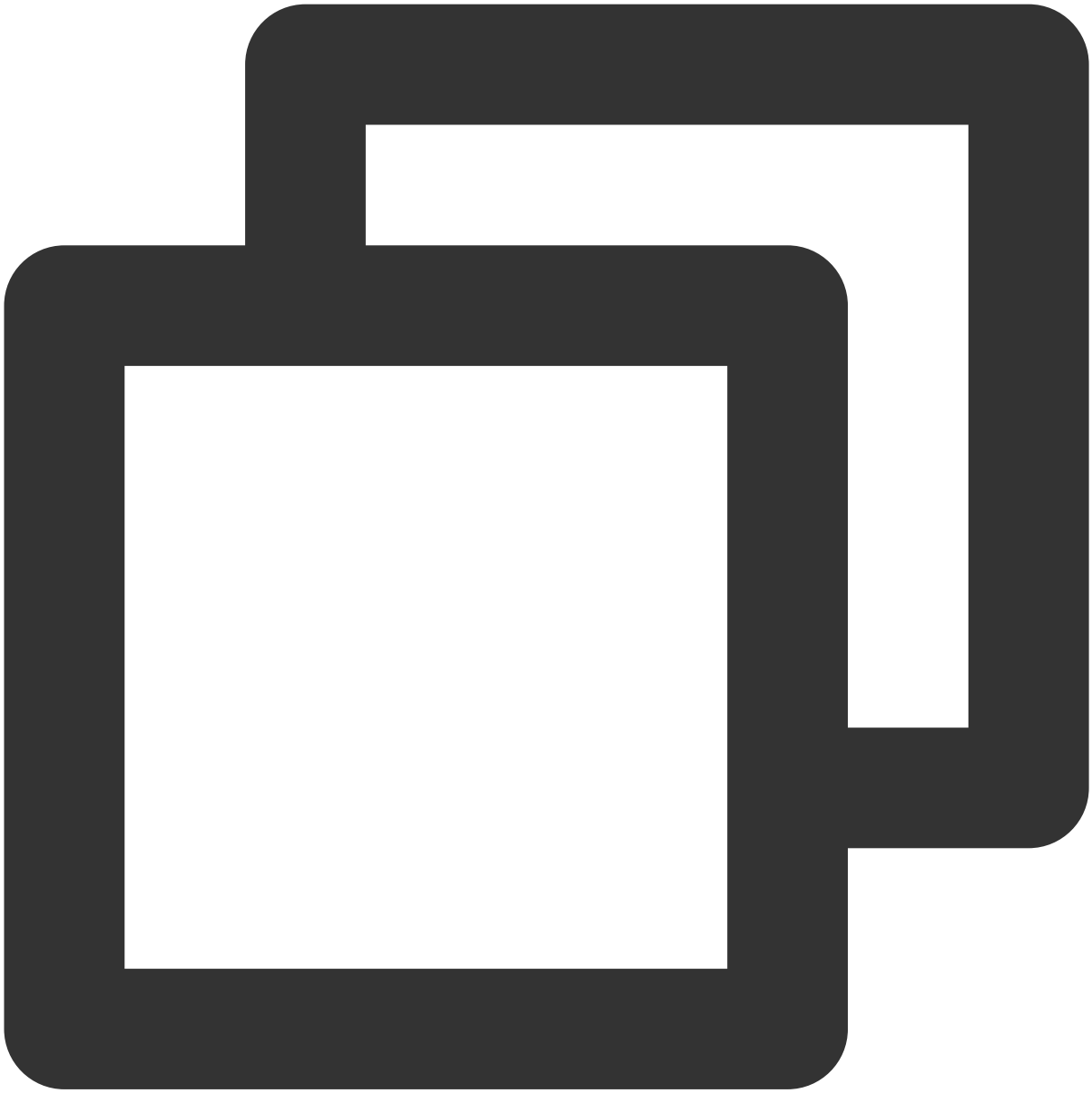
参数	类型	说明	默认值	含义
audioProfile	TUIAudioProfile	必填	-	TUIAudioProfile.kAudioProfileSpeech：语言模式；

				采样率：16k TUIAudioProfile.kAudioProfileDefault：标准模式（或者默认模式）；采样率：48k TUIAudioProfile.kAudioProfileMusic：音乐模式；采样率：48k

Returns：*Promise<void>*

startScreenSharing

开始屏幕共享



```
const roomEngine = new TUIRoomEngine();
// 设置本地屏幕分享流的播放区域为 id 是 'preview-screen' 的 div 元素，可以不设置
await roomEngine.setLocalRenderView({
 streamType: TUIVideoStreamType.kScreenStream,
 view: 'preview-screen',
});
// example 1: 开始屏幕分享
await roomEngine.startScreenSharing();
// example 2: 开始屏幕分享(可采集系统音频)
await roomEngine.startScreenSharing({ screenAudio: true });
```

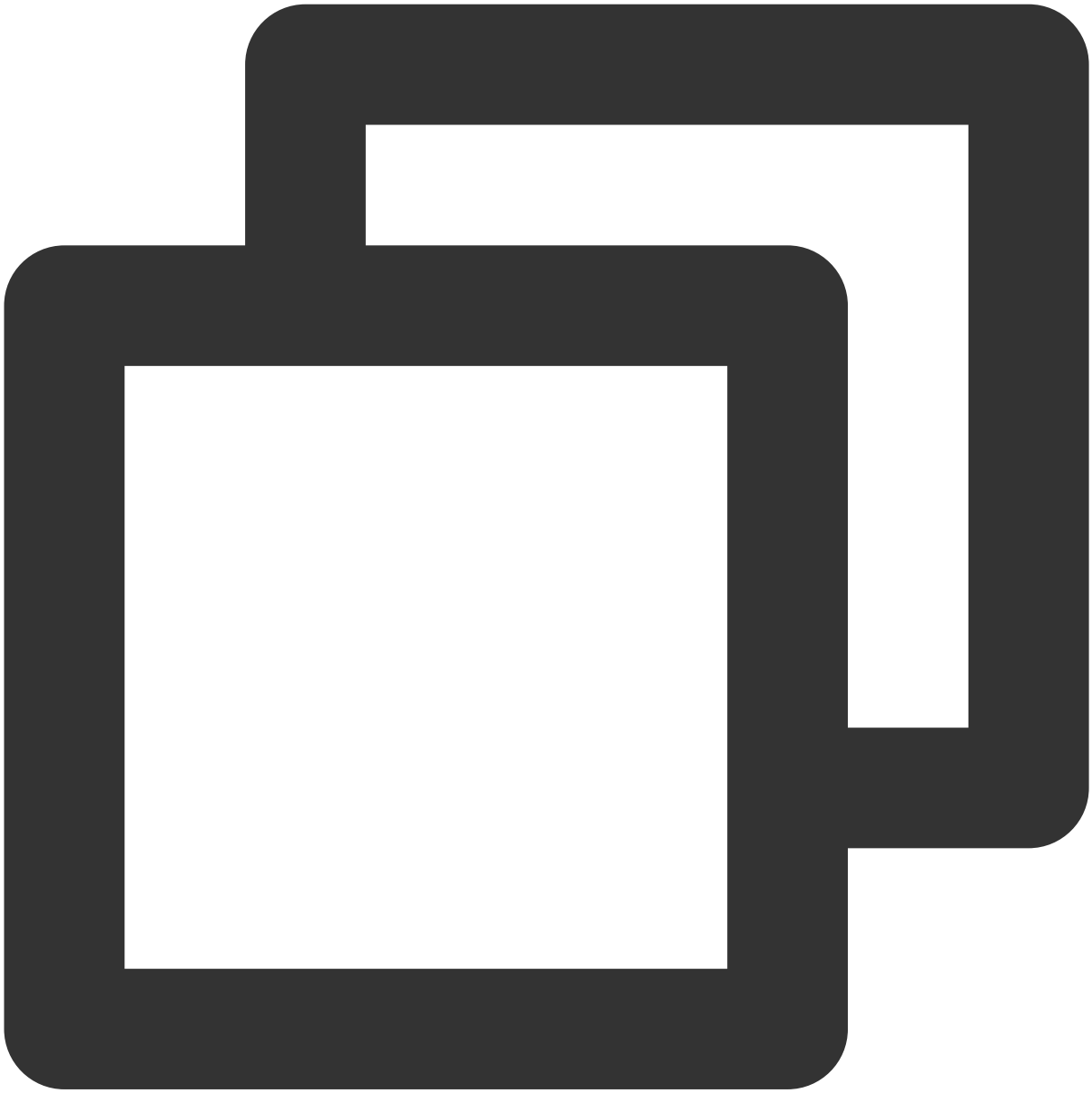
参数：

参数	类型	说明	默认值	含义
screenAudio	boolean	选填	false	web 端屏幕共享是否可选共享系统声音，screenAudio 默认值为 false

Returns：*Promise<void>*

stopScreenSharing

停止屏幕共享

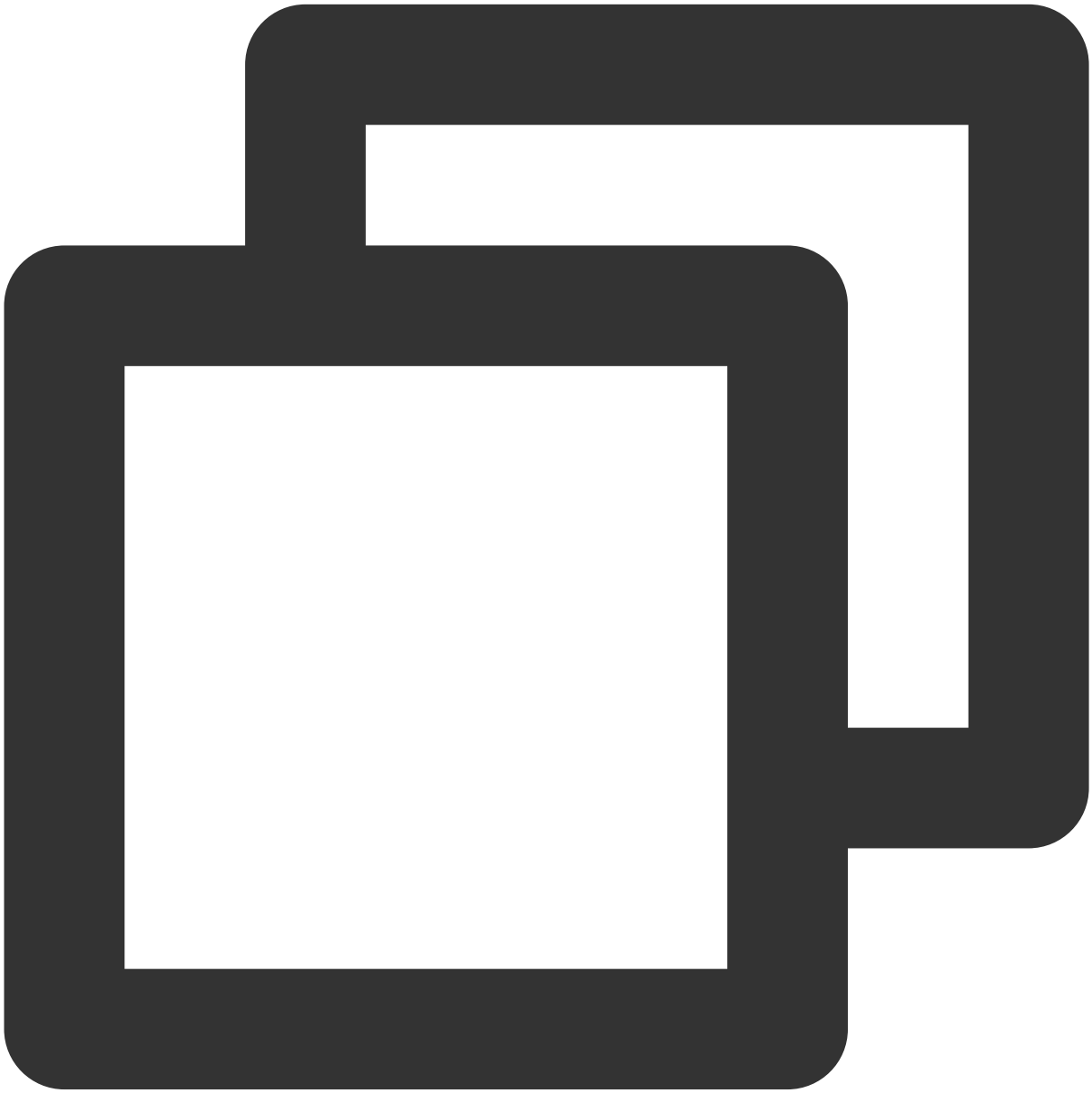


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopScreenSharing();
```

Returns：*Promise<void>*

startPushLocalVideo

进房后会默认向远端推本地视频流，该接口用于停止推流后重新向远端推本地视频流

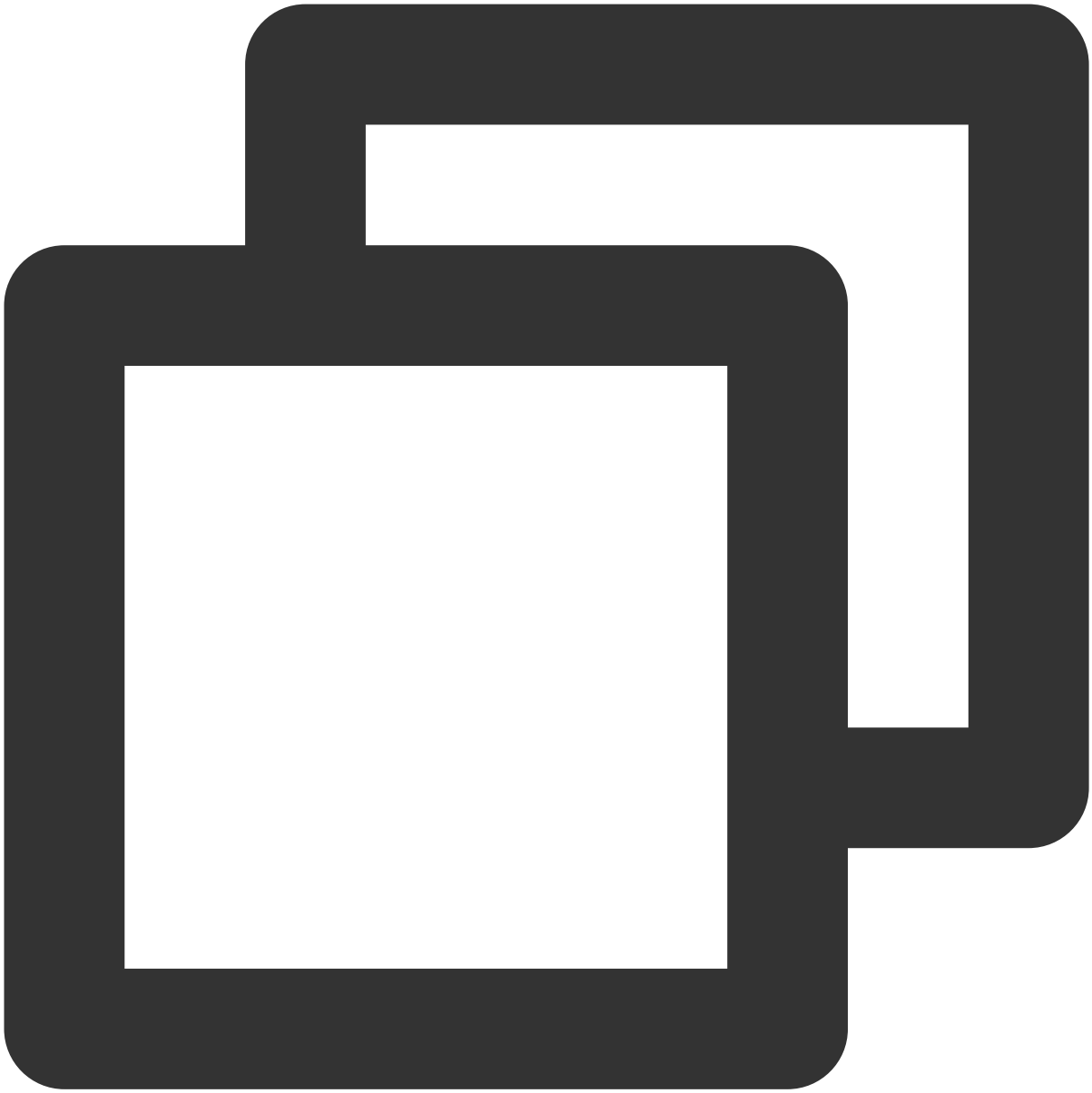


```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPushLocalVideo();
```

Returns：*Promise<void>*

stopPushLocalVideo

停止向远端推本地视频流

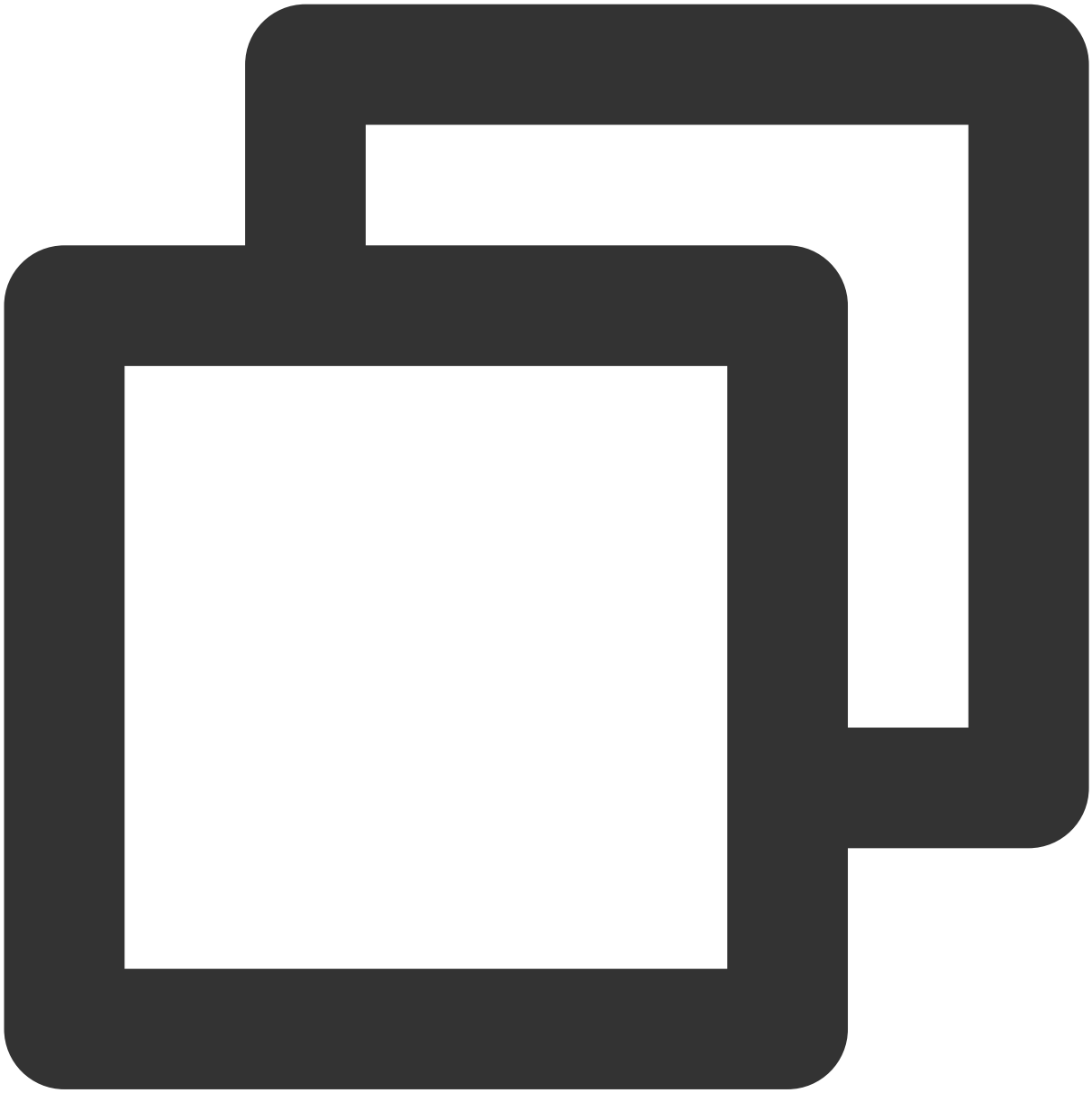


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPushLocalVideo();
```

Returns：*Promise<void>*

startPushLocalAudio

进房后会默认向远端推本地音频流，该接口用于停止推流后重新向远端推本地音频流

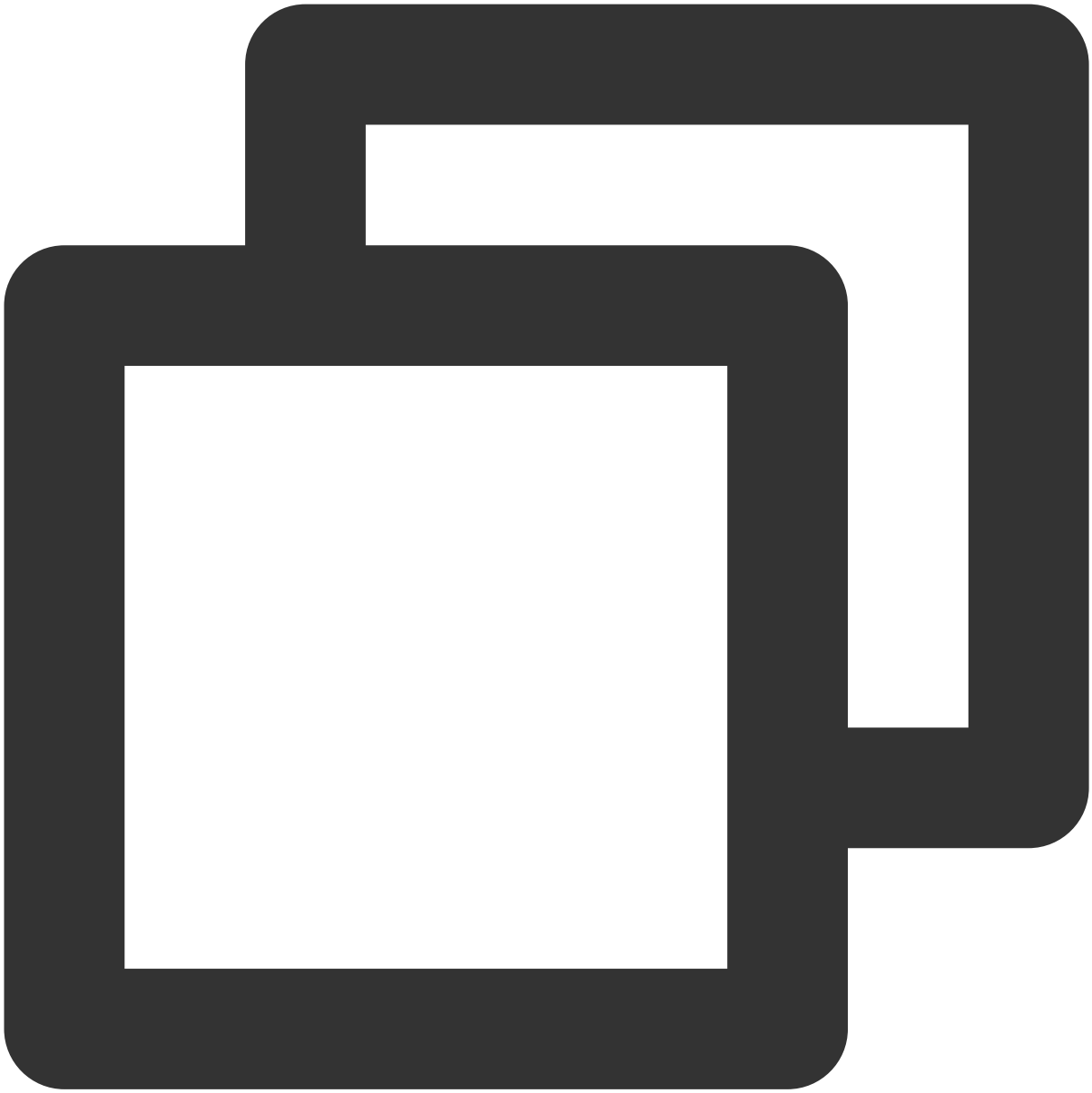


```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPushLocalAudio();
```

Returns：*Promise<void>*

stopPushLocalAudio

停止向远端推本地音频流

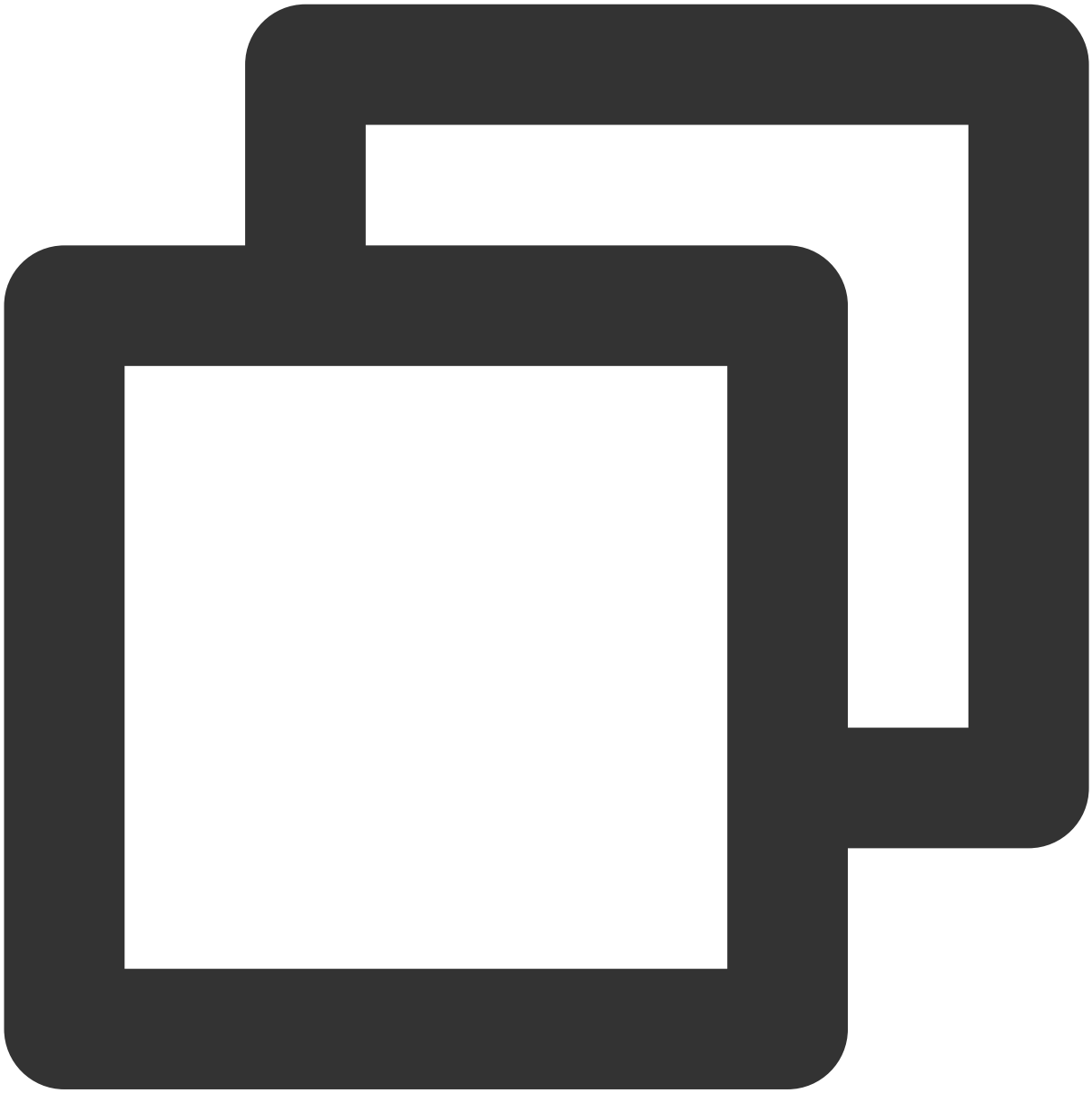


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPushLocalAudio();
```

Returns：*Promise<void>*

setRemoteVideoView

设置远端流渲染的区域



```
const roomEngine = new TUIRoomEngine();

// 设置远端用户视频流在 id 为 'remote_preview_camera' 的区域播放
await roomEngine.setRemoteVideoView({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
   view: 'remote_preview_camera',
});
// 设置远端用户屏幕分享流在 id 为 'remote_preview_screen' 的区域播放
await roomEngine.setRemoteVideoView({
   userId: 'user_1234',
```

```
    streamType: TUIVideoStreamType.kScreenStream,
    view: 'remote_preview_screen',
});
```

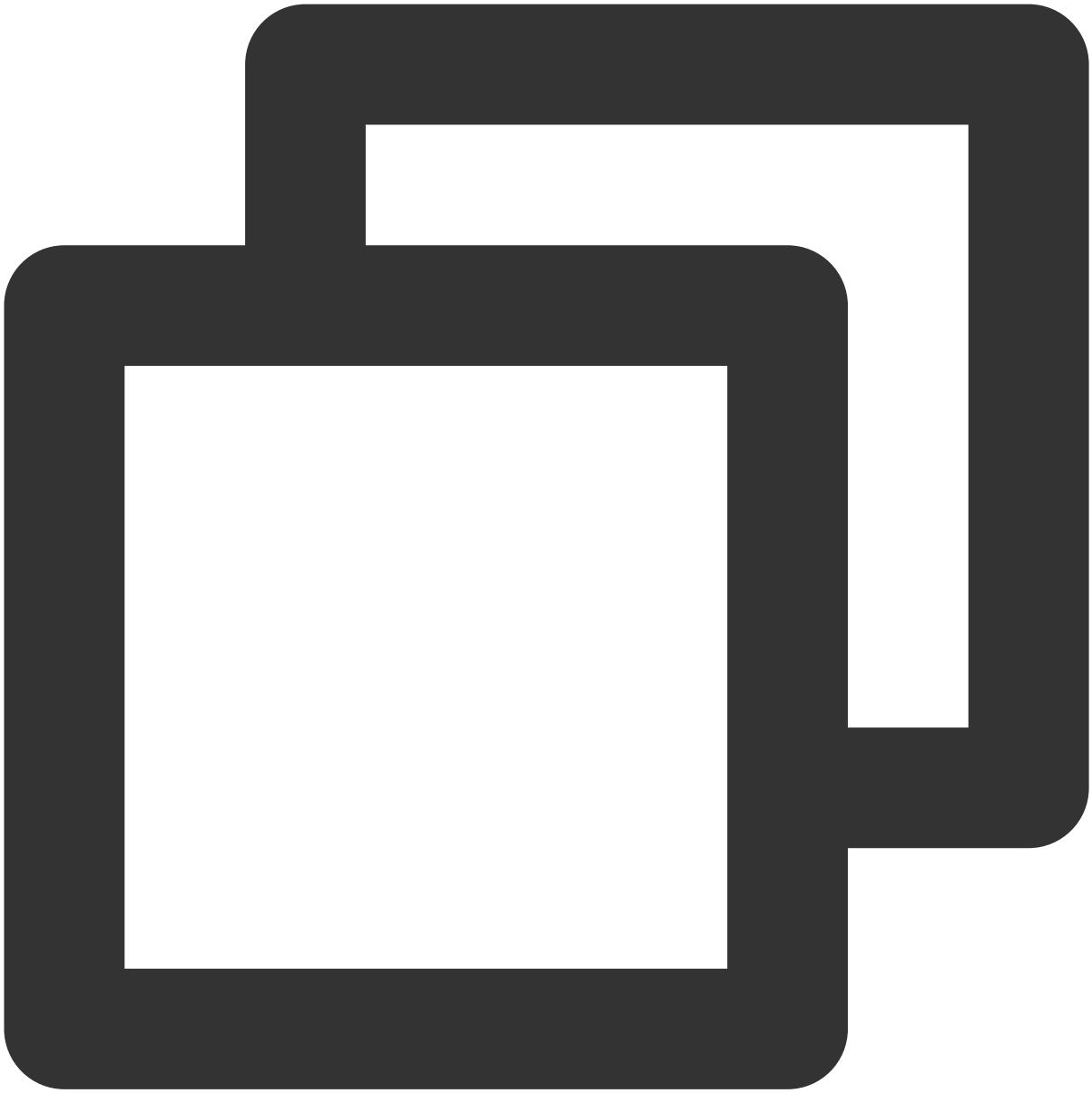
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型
view	string	必填	-	播放远端用户流的 div 元素的 id

Returns：*Promise<void>*

startPlayRemoteVideo

开始播放远端用户视频流



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPlayRemoteVideo({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
});
```

参数：

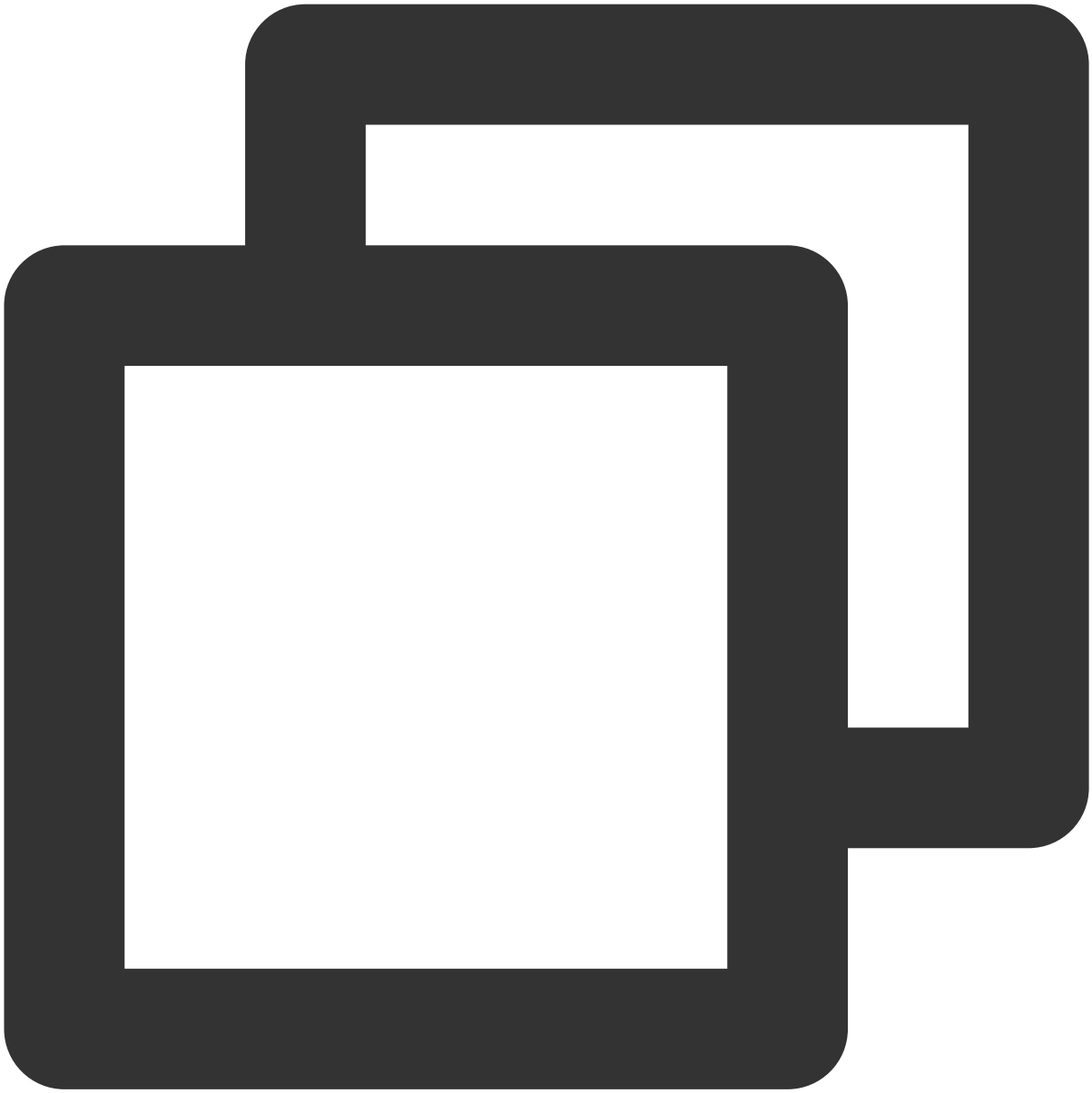
参数	类型	说明	默认值	含义

userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型 TUIVideoStreamType.kCameraStream 视频流 TUIVideoStreamType.kScreenStream 屏幕分享流 TUIVideoStreamType.kCameraStreamLow 低清视频流

Returns：*Promise<void>*

stopPlayRemoteVideo

停止播放远端用户视频流



```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPlayRemoteVideo({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
});
```

参数：

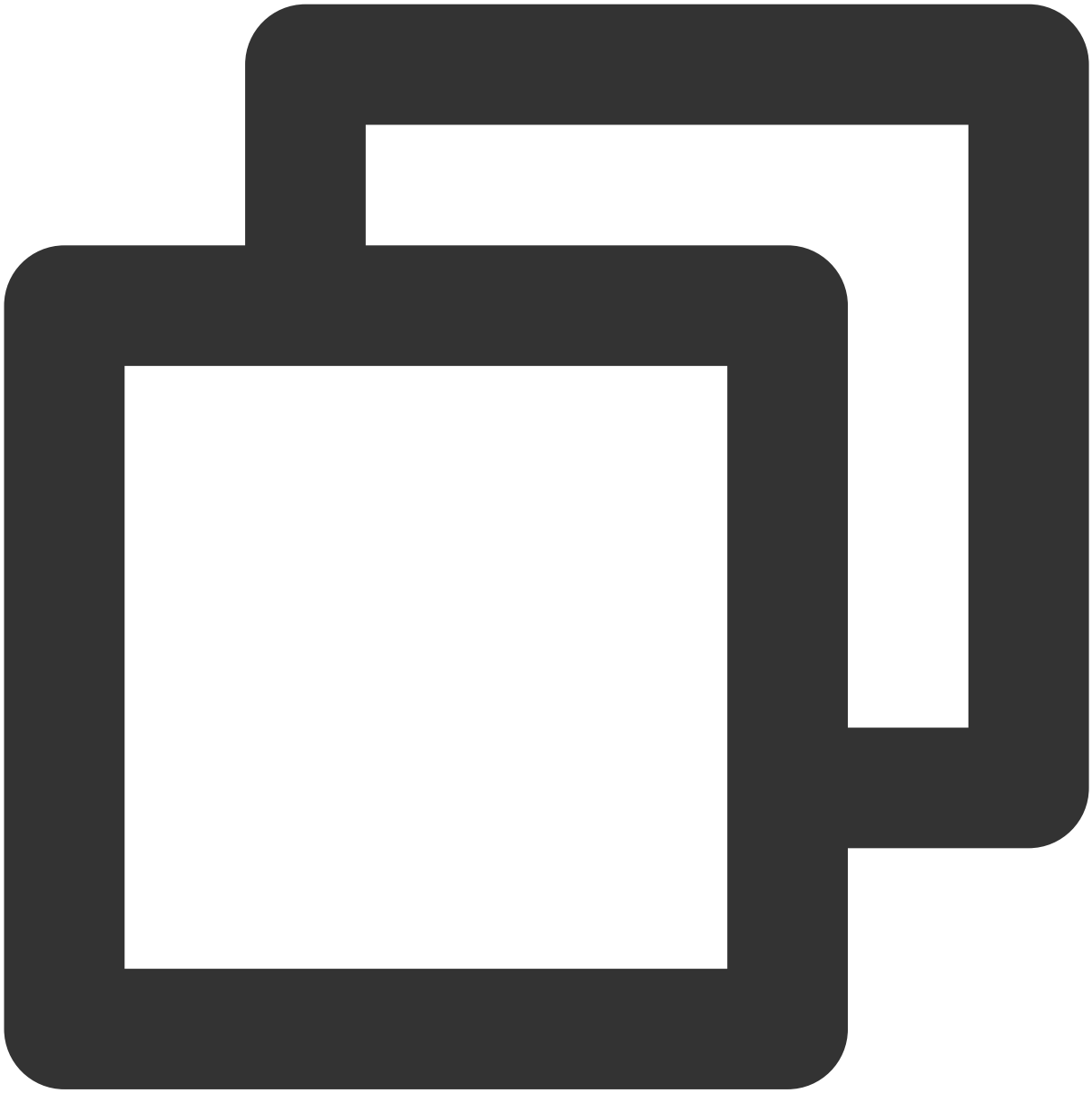
参数	类型	说明	默认值	含义

userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型 TUIVideoStreamType.kCameraStream 视频流 TUIVideoStreamType.kScreenStream 屏幕分享流 TUIVideoStreamType.kCameraStreamLow 低清视频流

Returns：*Promise<void>*

muteRemoteAudioStream

停止远端用户的音频流

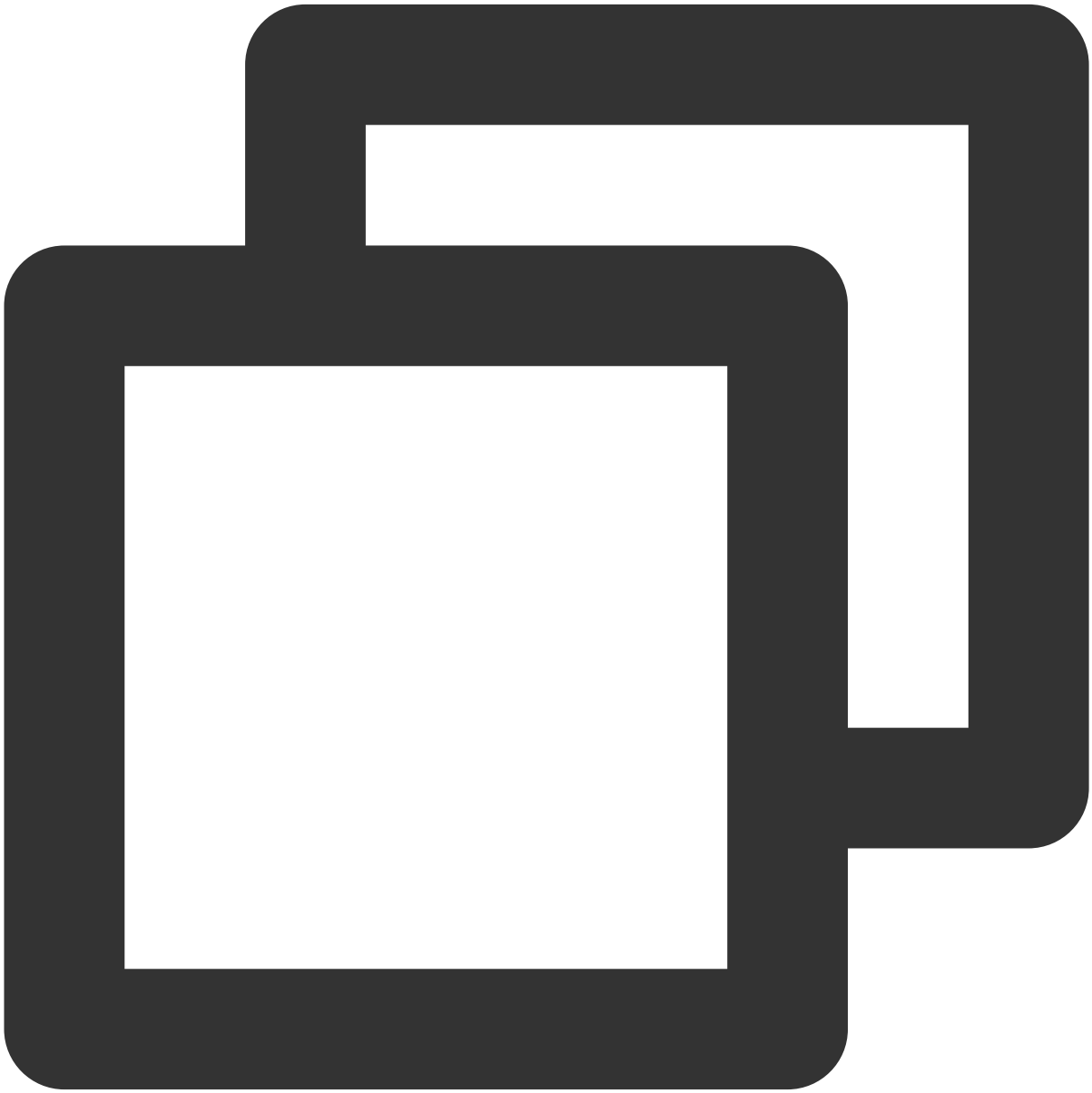


```
const roomEngine = new TUIRoomEngine();
await roomEngine.muteRemoteAudioStream({
  userId: 'user_1234',
  isMute: true,
});
```

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
isMute	boolean	必填	-	是否停止远端用户的音频

openRemoteDeviceByAdmin

请求远端用户打开媒体设备



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.openRemoteDeviceByAdmin({
    userId: 'user_1234',
    device: TUIMediaDevice.kMicrophone  //请求的设备为麦克风
```

```
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
     switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
         // 请求被接受
         break;
       case TUIRequestCallbackType.kRequestRejected:
         // 请求被拒绝
         break;
       case TUIRequestCallbackType.kRequestCancelled:
         // 请求已取消
         break;
       case TUIRequestCallbackType.kRequestTimeout:
         // 请求超时
         break;
       case TUIRequestCallbackType.kRequestError:
         // 请求错误
         break;
       default:
         break;
     }
   },
});
```

参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）
timeout	number	必填	-	超时时间。若 timeout 设置为 0，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string> requestId*

该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

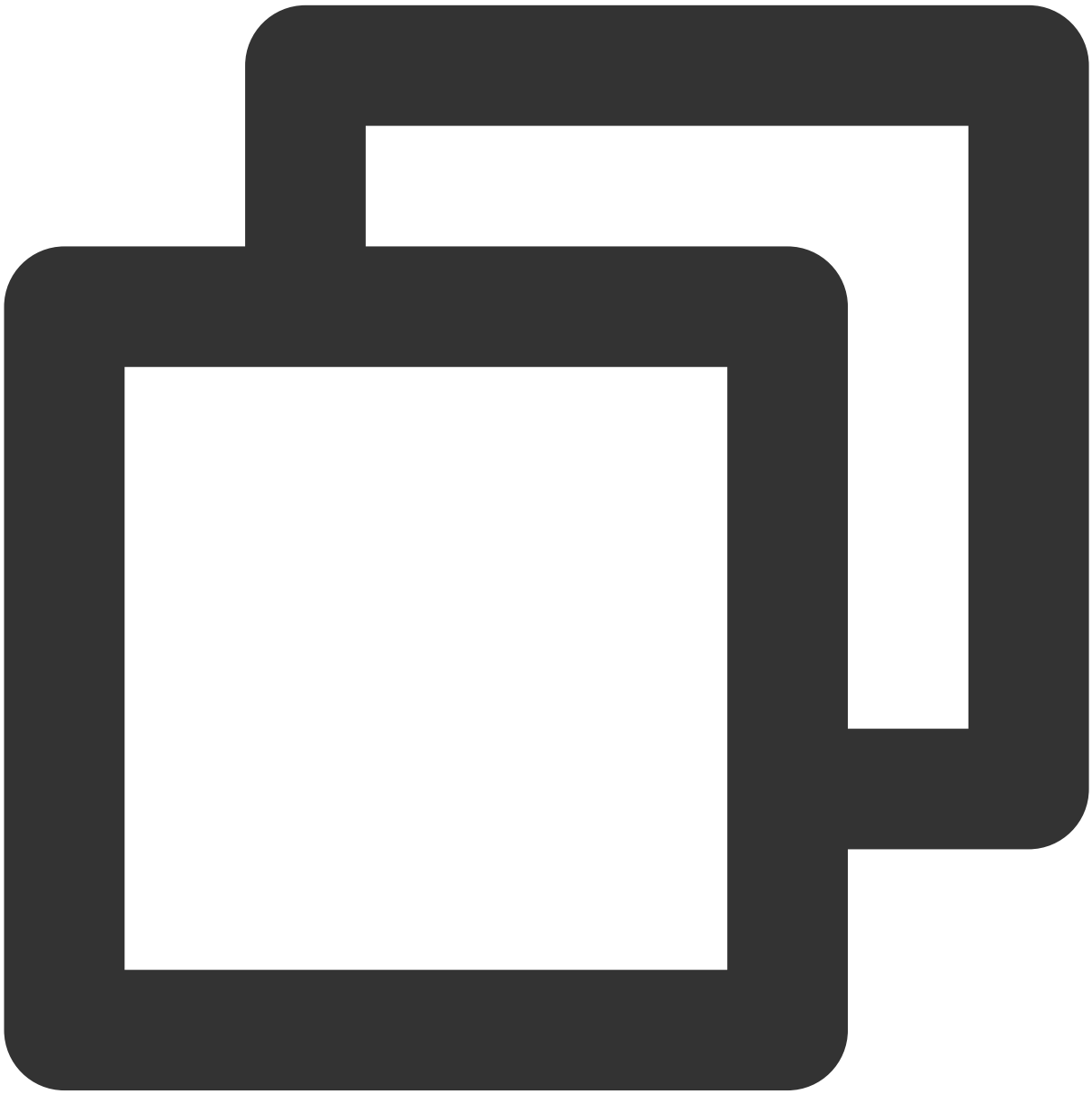
说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

版权所有：腾讯云计算（北京）有限责任公司

applyToAdminToOpenLocalDevice

参会者向主持人申请打开设备



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.applyToAdminToOpenLocalDevice({
    device: TUIMediaDevice.kMicrophone  //请求的设备为麦克风
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
     switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
```

```
          // 请求被接受
          break;
        case TUIRequestCallbackType.kRequestRejected:
          // 请求被拒绝
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
      }
    },
});
```

参数：

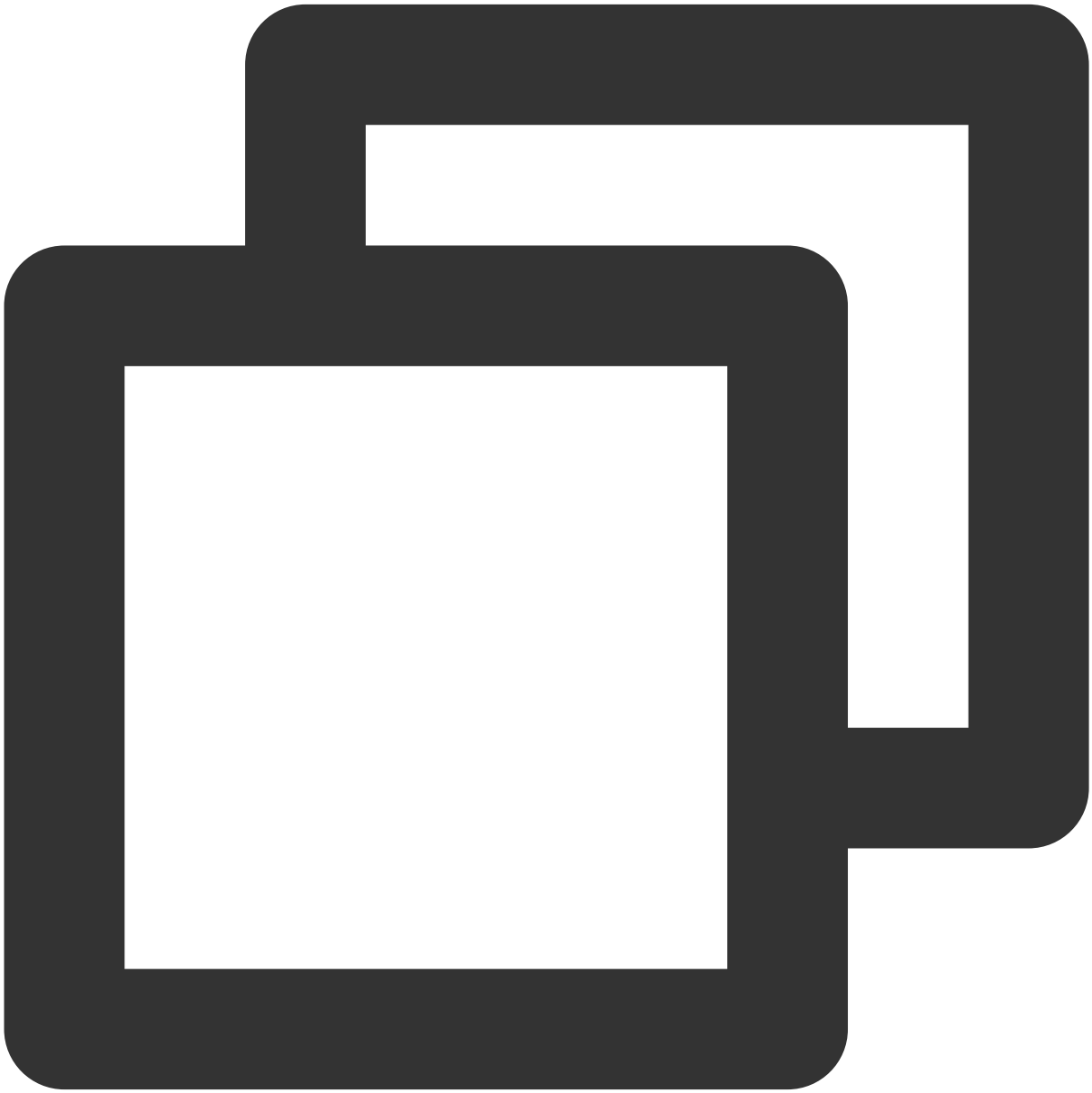
参数	类型	说明	默认值	含义
device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）
timeout	number	必填	-	超时时间。若 timeout 设置为 0，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string> requestId*

该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

closeRemoteDeviceByAdmin

关闭远端用户媒体设备



```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeRemoteDeviceByAdmin({
   userId: 'user_1234',
   device: TUIMediaDevice.kMicrophone, //关闭麦克风
 });
```

参数：

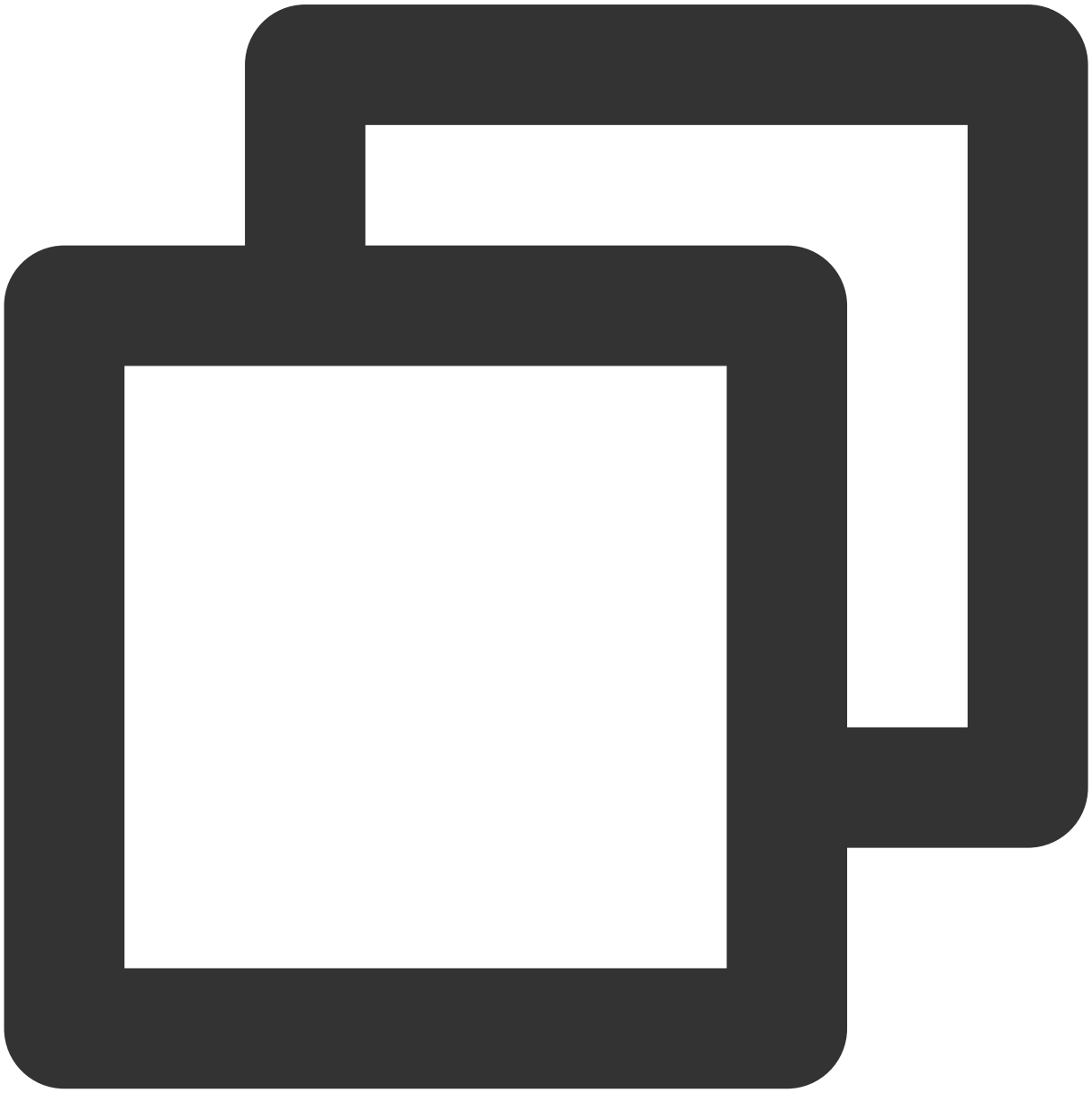
参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID

device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）

Returns：*Promise<void>*

cancelRequest

取消已经发出的请求



```
const roomEngine = new TUIRoomEngine();
await roomEngine.cancelRequest({
   requestId: '',    // 请使用实际 requestId
});
```

参数：

参数	类型	说明	默认值	含义
requestId	string	必填	-	请求 ID

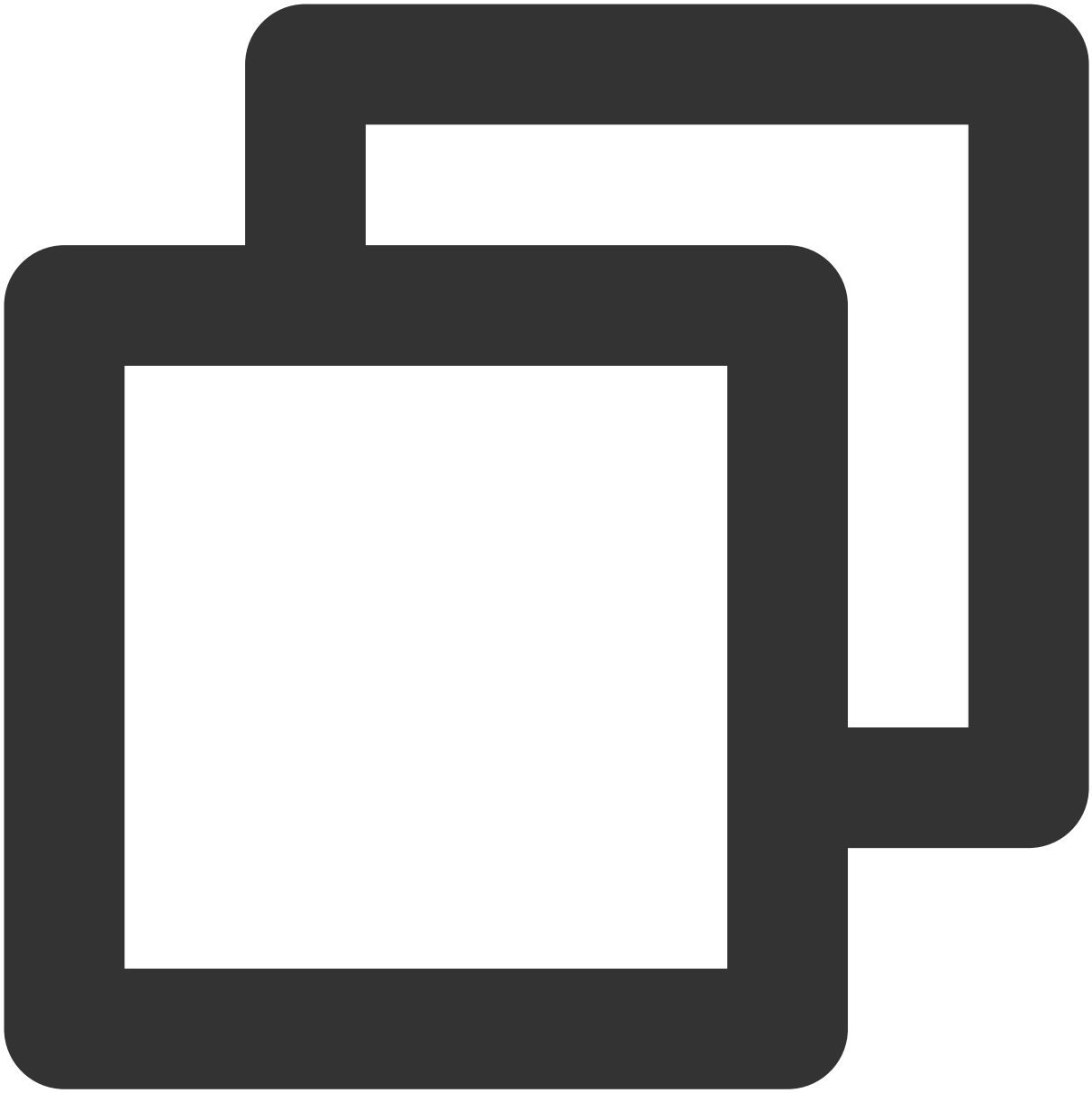
Returns：*Promise<void>*

说明：

v1.0.2 及其以上版本该接口接收的 requestId 参数类型为 string；v1.0.0 及 v1.0.1 版本该接口接收的 requestId 参数类型为 number；

responseRemoteRequest

回复远端用户的请求



```
const roomEngine = new TUIRoomEngine();
// 同意远端的请求
await roomEngine.responseRemoteRequest({
 requestId: '',     // 请使用实际 requestId
 agree: true,
});
// 拒绝远端的请求
await roomEngine.responseRemoteRequest({
 requestId: '',     // 请使用实际 requestId
 agree: false,
});
```

参数：

参数	类型	说明	默认值	含义
requestId	string	必填	-	请求 ID
agree	boolean	必填	-	是否同意

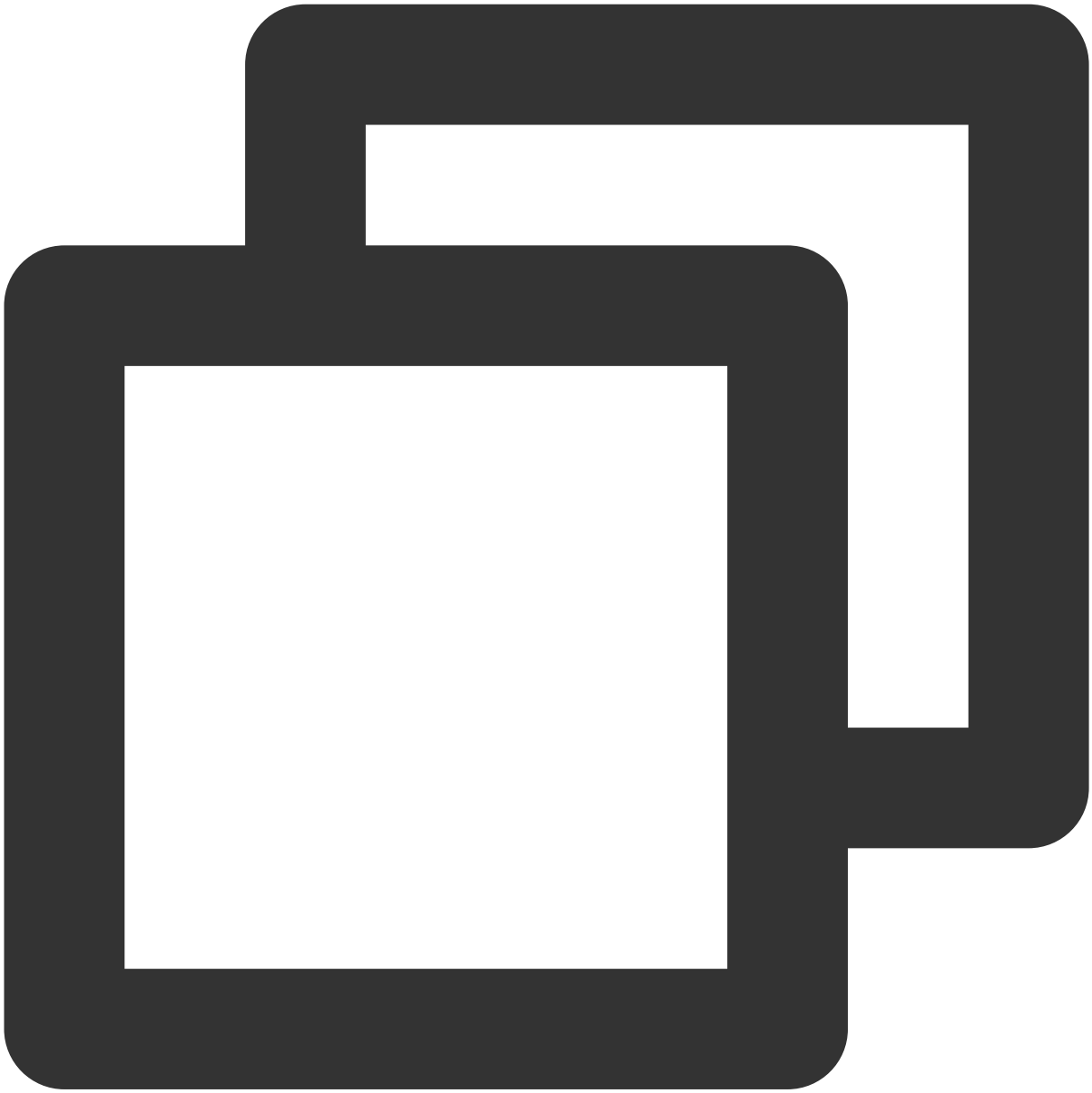
Returns：*Promise<void>*

说明：

v1.0.2 及其以上版本该接口接收的 requestId 参数类型为 string；v1.0.0 及 v1.0.1 版本该接口接收的 requestId 参数类型为 number；

disableDeviceForAllUserByAdmin

禁止/允许所有用户打开媒体设备（此接口对于房主和管理员无效）



```
// Example 1: 禁止所有用户打开麦克风
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: true,
})
// Example 2: 允许所有用户打开麦克风
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: false,
})
```

版权所有：腾讯云计算（北京）有限责任公司

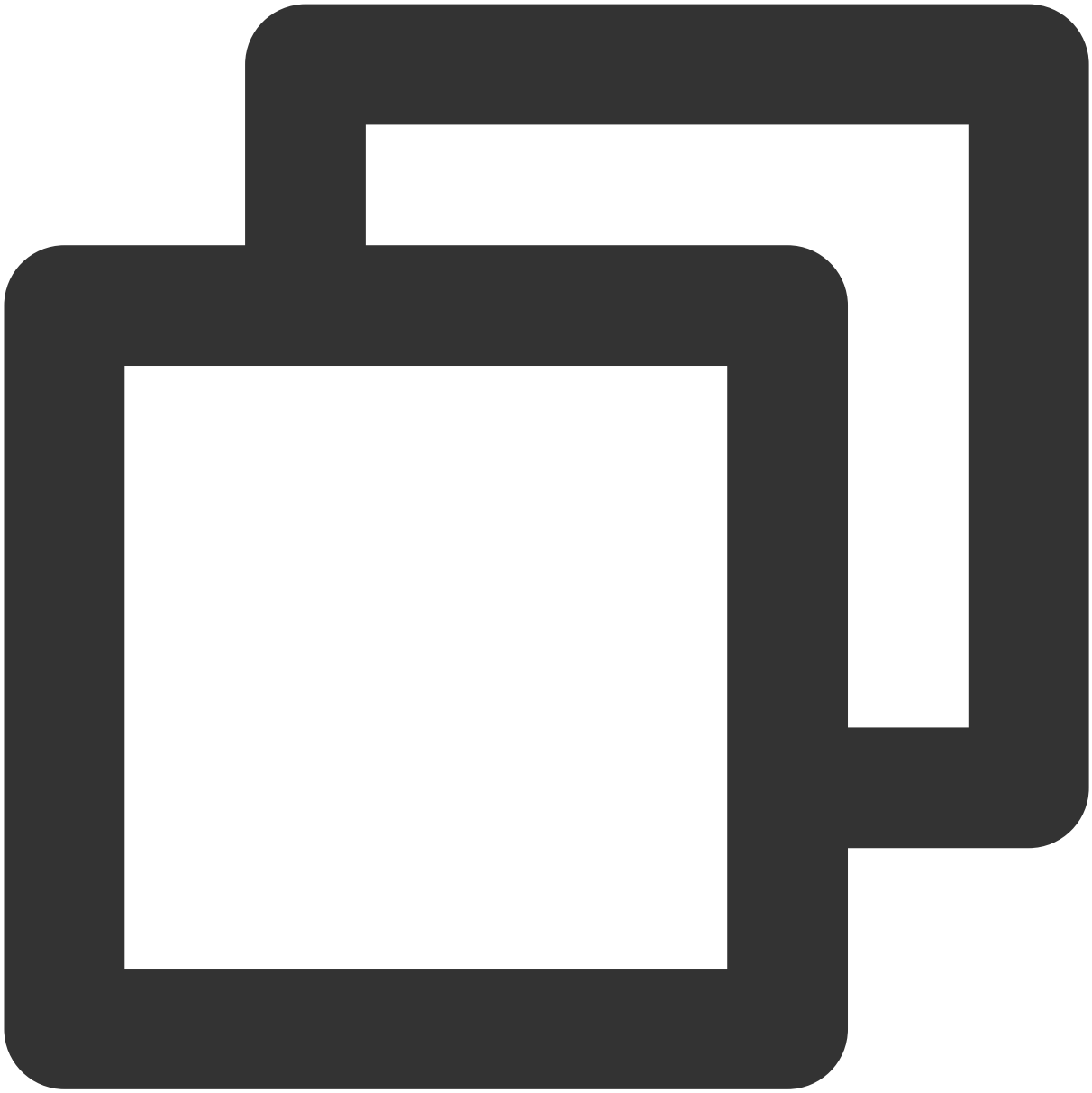
参数：

参数	类型	说明	默认值	含义
device	TUIMediaDevice	必填	-	被禁用的媒体设备类型（摄像头/麦克风/屏幕共享）
isDisable	boolean	必填	-	是否被禁止

Returns：*Promise<void>*

disableSendingMessageForAllUser

所有用户的是否允许发消息（此接口对于房主和管理员无效）



```
await roomEngine.disableSendingMessageForAllUser({
  isDisable: true,
});
```

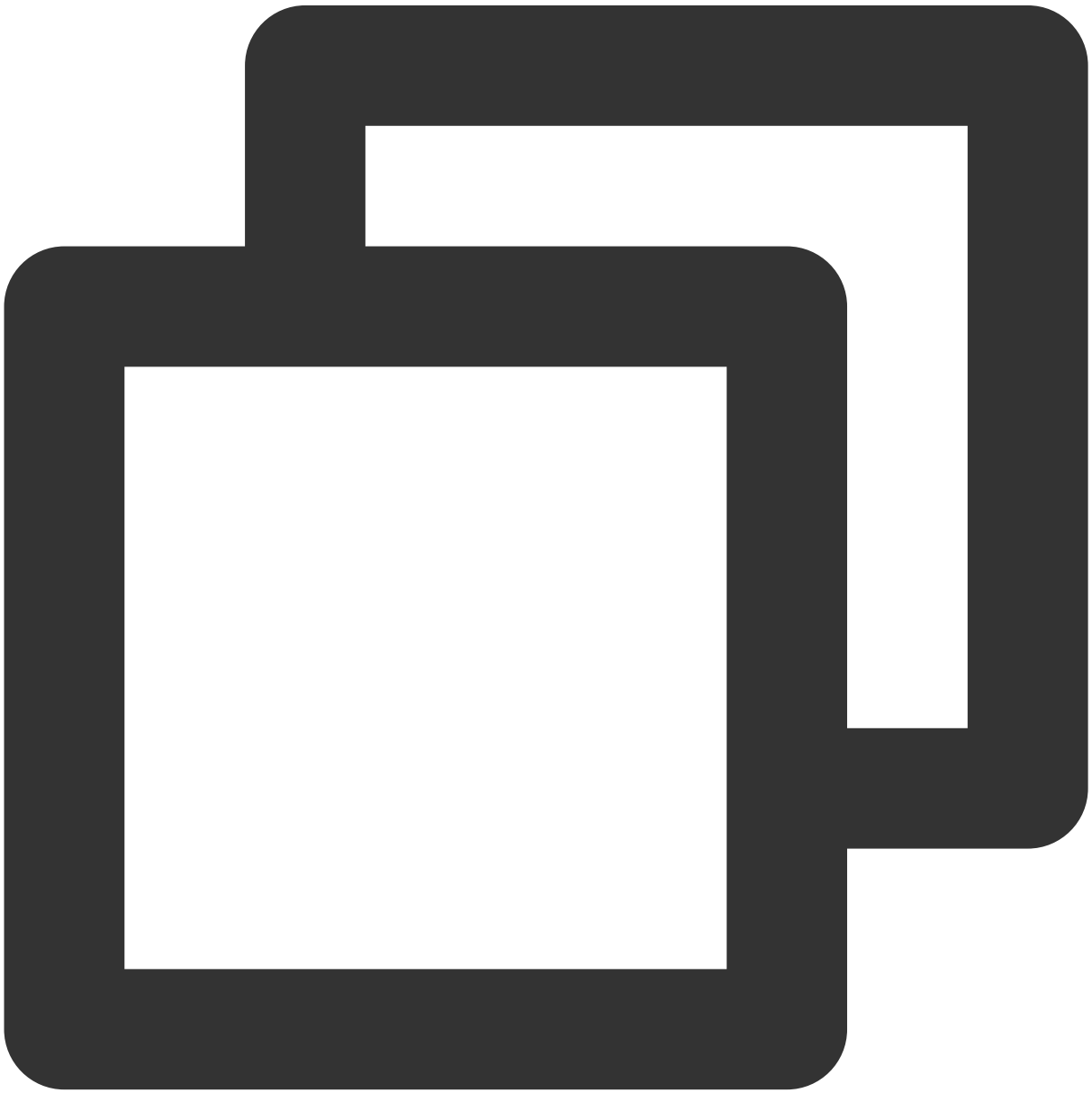
参数：

参数	类型	说明	默认值	含义
isDisable	boolean	必填	-	是否被禁用

Returns：*Promise<void>*

disableSendingMessageByAdmin

特定用户是否被允许发消息



```
await roomEngine.disableSendingMessageByAdmin({
  userId: 'user_1234',
  isDisable: true,
```

```
});
```

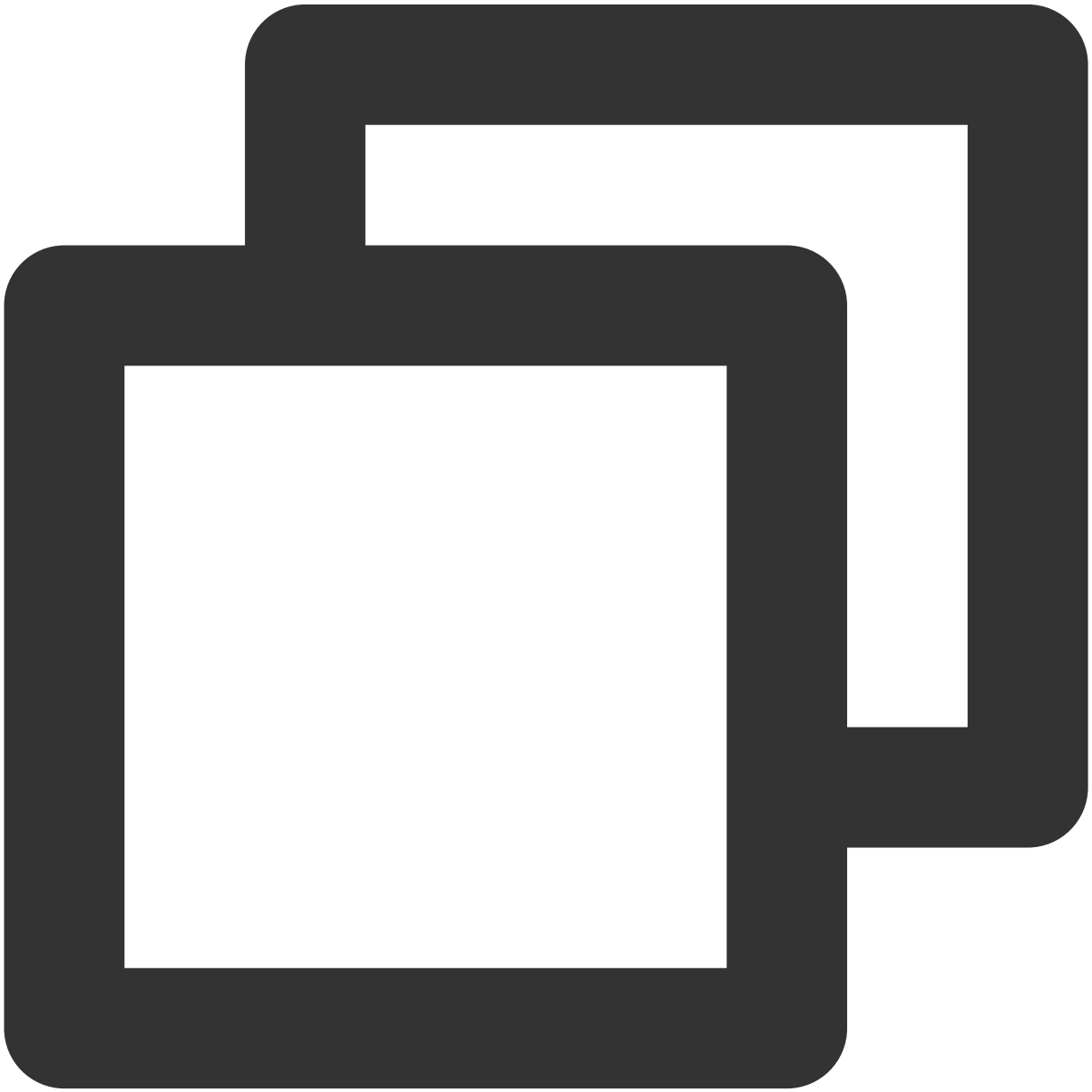
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
isDisable	boolean	必填	-	是否被禁用

Returns：*Promise<void>*

changeUserRole

改变用户的角色（仅主持人可调用该接口）



```
const roomEngine = new TUIRoomEngine();

// 将房间移交给用户 user_1234
await roomEngine.changeUserRole({
   userId: 'user_1234',
   role: TUIRole.kRoomOwner,
});

// 将用户 user_1234 设置为房间管理员
await roomEngine.changeUserRole({
  userId: 'user_1234',
```

```
  userRole: TUIRole.kAdministrator,
});
```

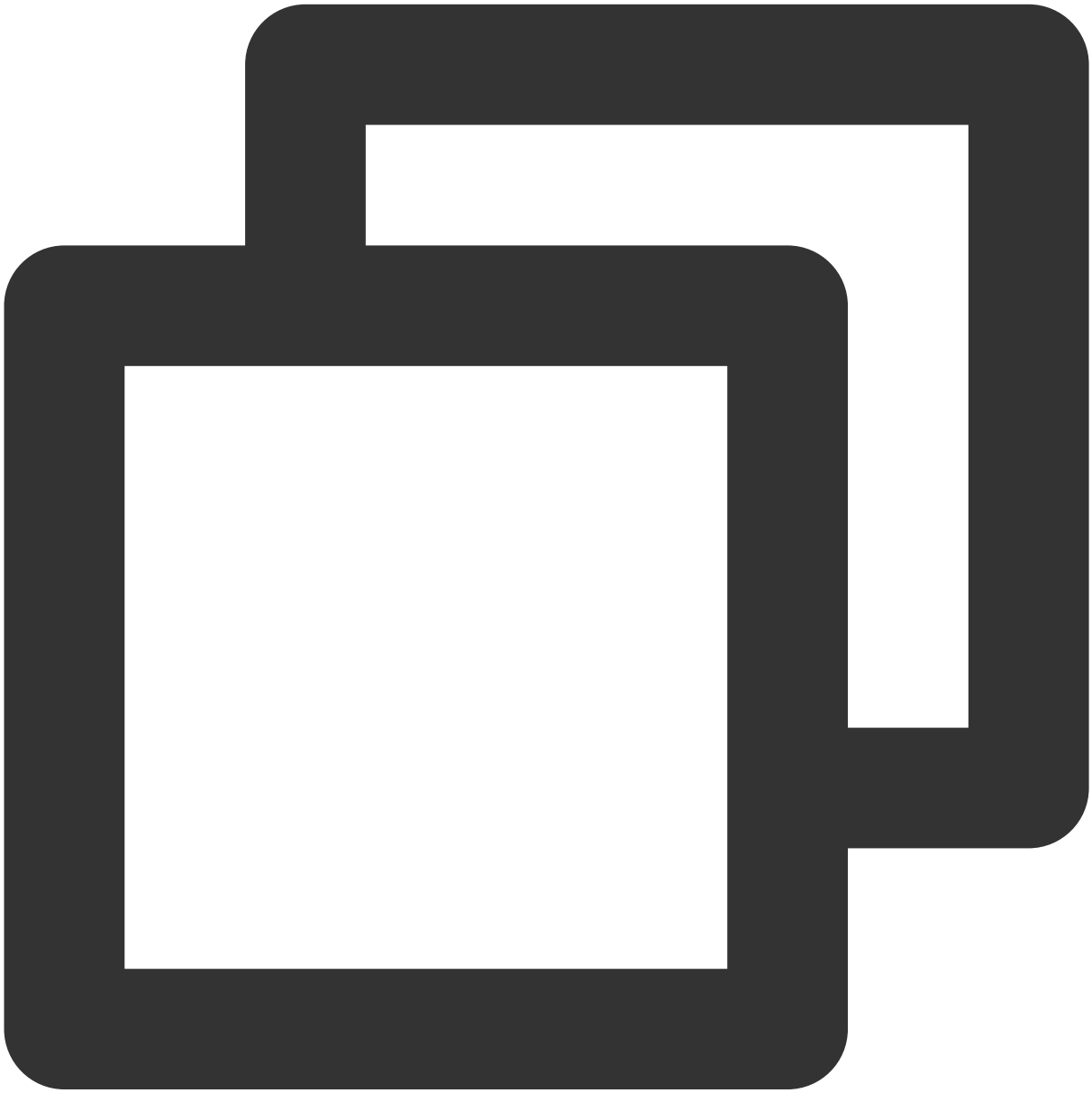
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
userRole	TUIRole	必填	-	用户角色 主持人 TUIRole.kRoomOwner 管理员 TUIRole.kAdministrator 普通成员 TUIRole.kGeneralUser

Returns：*Promise<void>*

kickRemoteUserOutOfRoom

将用户踢出房间（仅主持人和管理员可调用该接口）



```
const roomEngine = new TUIRoomEngine();
await roomEngine.kickRemoteUserOutOfRoom({
   userId: 'user_1234',
});
```

参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID

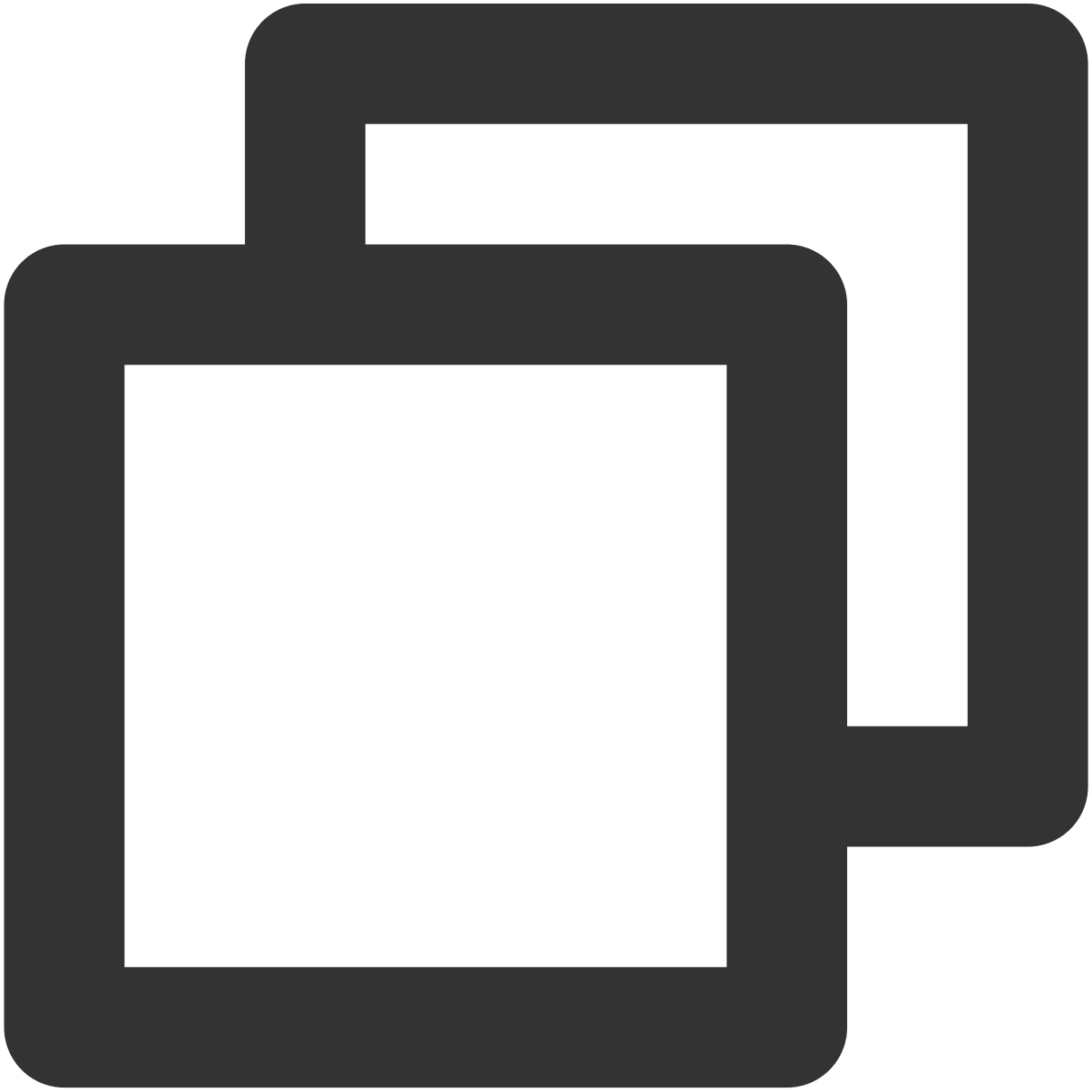
Returns：*Promise<void>*

setMaxSeatCount

设置房间麦位最大值

roomType 为 TUIRoomType.kConference (教育及会议场景) 时，maxSeatCount 值不做限制；

roomType 为 TUIRoomType.kLivingRoom (直播场景) 时，maxSeatCount 最大限制为 16；



```
const roomEngine = new TUIRoomEngine();
```

```
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.setMaxSeatCount({ maxSeatCount: 16 })
```

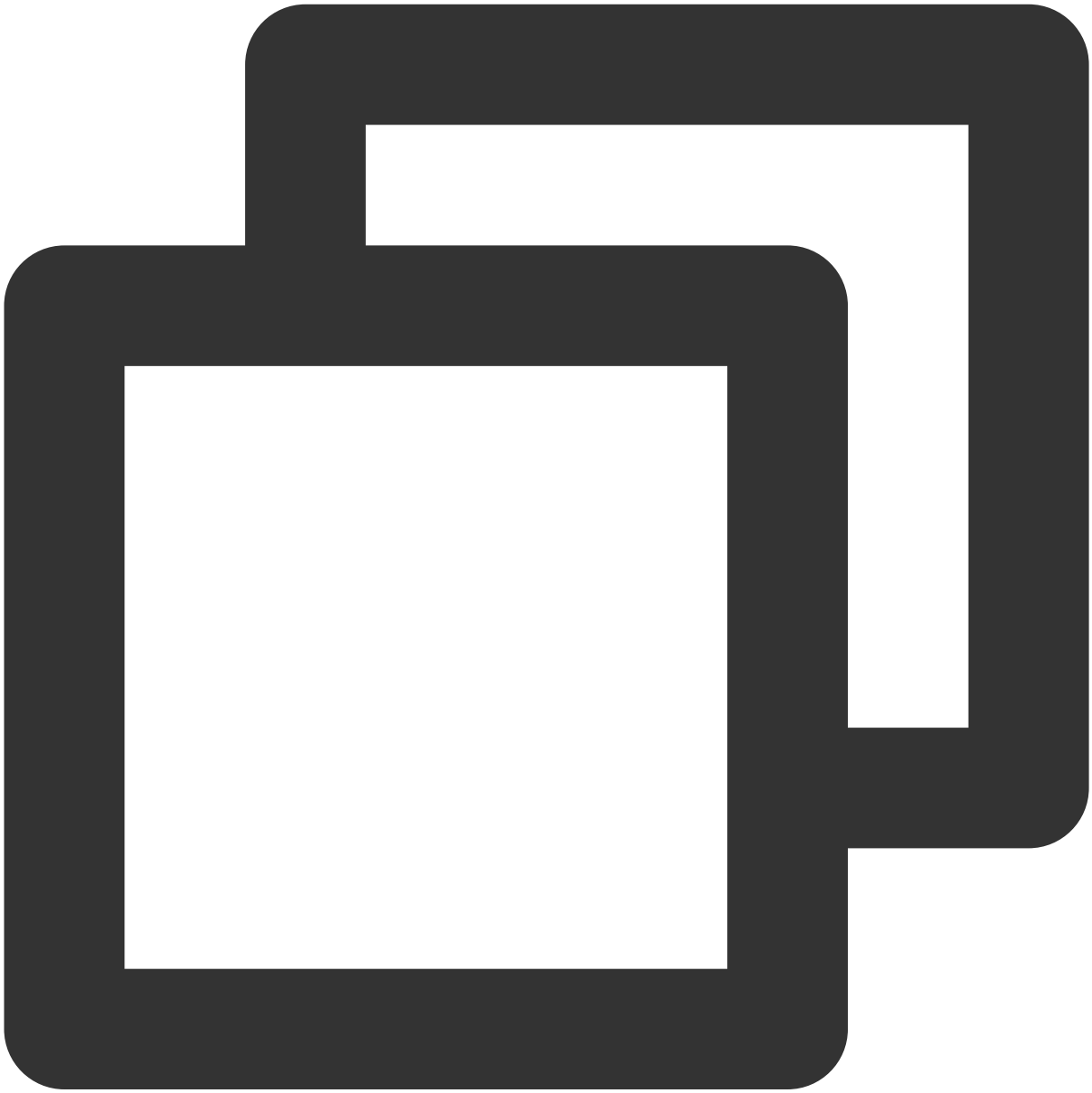
参数：

参数	类型	说明	默认值	含义
maxSeatCount	number	必填	-	设置房间麦位最大值

Returns：Promise<void>

getSeatList

获取麦位列表



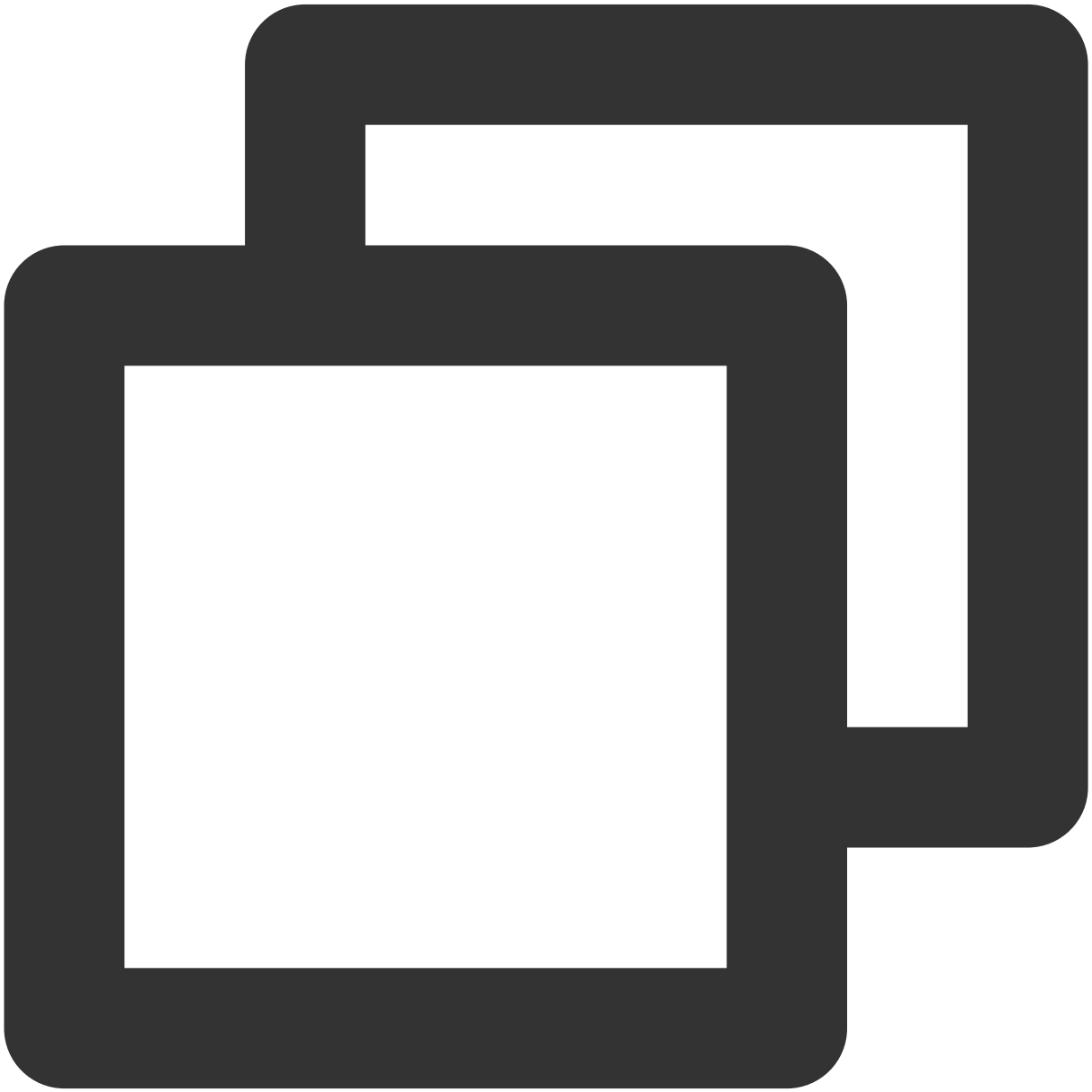
```
const roomEngine = new TUIRoomEngine();
const seatList = await roomEngine.getSeatList();
```

Returns：*Promise<*TUISeatInfo*[]>* seatList

seatList 为当前房间所有麦位列表

takeSeat

麦下用户可调用 takeSeat 成为麦上用户，仅麦上用户可发布本地音视频流。
当 roomInfo.roomType 为 TUIRoomType.kConference 且 roomInfo.speechMode 为 TUISpeechMode.kSpeakAfterTakingSeat 时，普通用户调用 takeSeat 方法需要等待主持人/管理员的同意后成为麦上用户。
当 roomInfo.roomType 为 TUIRoomType.kLivingRoom 且 roomInfo.speechMode 为 TUISpeechMode.kFreeToSpeak 时, 普通用户调用 takeSeat 方法成功后即为麦上用户
主持人&管理员调用 takeSeat 成功后即为麦上用户。
麦上用户的变更通过 TUIRoomEvents.onSeatListChanged 通知所有用户。



```
const roomEngine = new TUIRoomEngine();
```

```
// 情景一：主持人/管理员上麦
// 情景二：当 roomInfo.roomType 为 TUIRoomType.kConference
// 且 roomInfo.speechMode 为 TUISpeechMode.kSpeakAfterTakingSeat 时，普通用户上麦
await roomEngine.takeSeat({
 seatIndex: -1,
 timeout: 0,
});

// 情景三：当 roomInfo.enableSeatControl 为 true 时，普通用户上麦
const requestId = await roomEngine.instance?.takeSeat({
   seatIndex: -1,
   timeout: 0,
   requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
      switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
         // 请求被接受
         break;
       case TUIRequestCallbackType.kRequestRejected:
         // 请求被拒绝
         break;
       case TUIRequestCallbackType.kRequestCancelled:
         // 请求已取消
         break;
       case TUIRequestCallbackType.kRequestTimeout:
         // 请求超时
         break;
       case TUIRequestCallbackType.kRequestError:
         // 请求错误
         break;
       default:
         break;
   }
 },
});
```

参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index，无麦位序号时设置为 -1
timeout	number	必填	-	超时时间。若 timeout 设置为 0，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：Promise<string> requestId

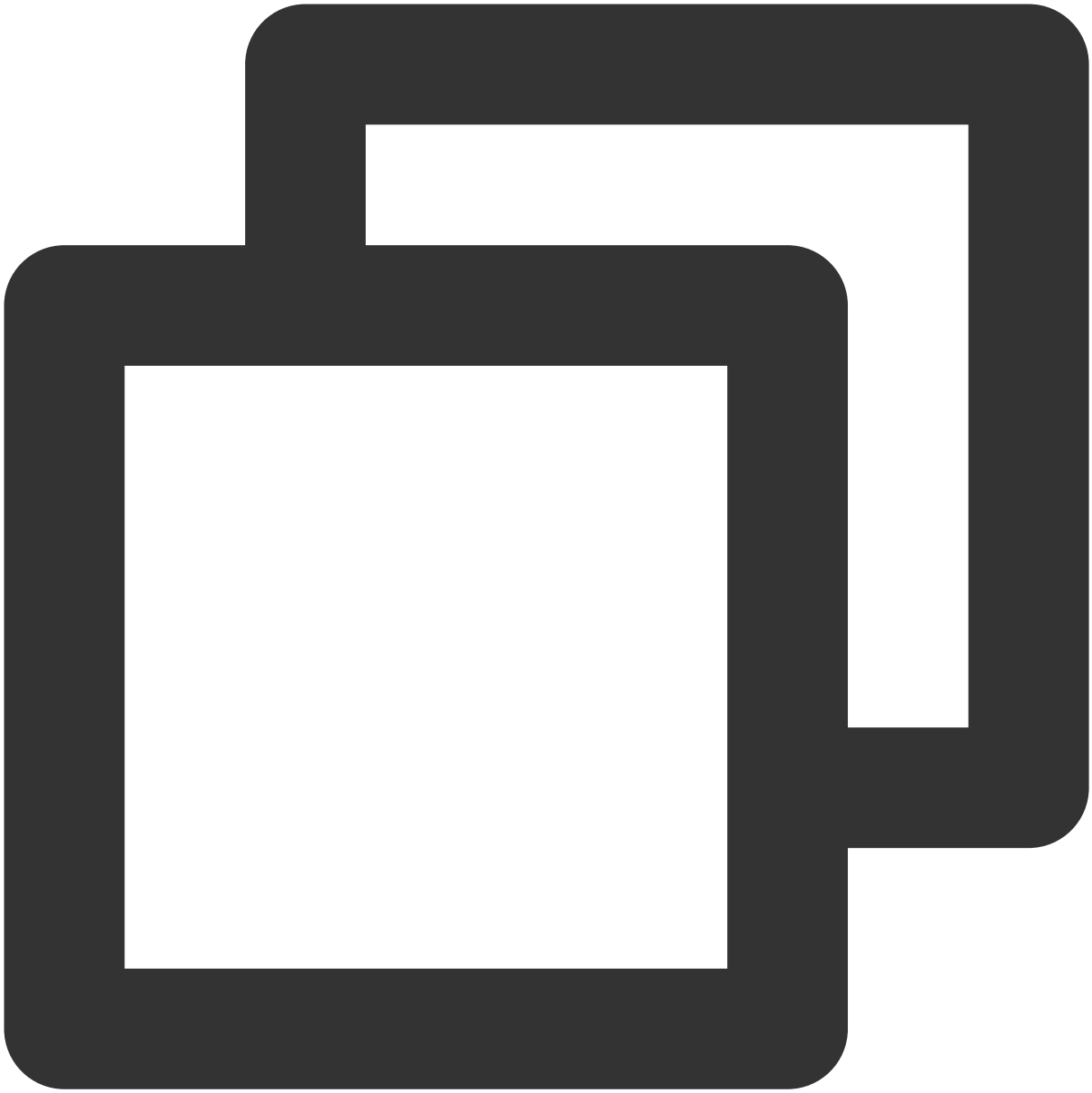
当 roomInfo.enableSeatControl 为 true，普通用户调用该接口时返回 requestId，普通用户可使用该 requestId 调用 cancelRequest 接口取消上麦请求。

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

leaveSeat

释放麦位

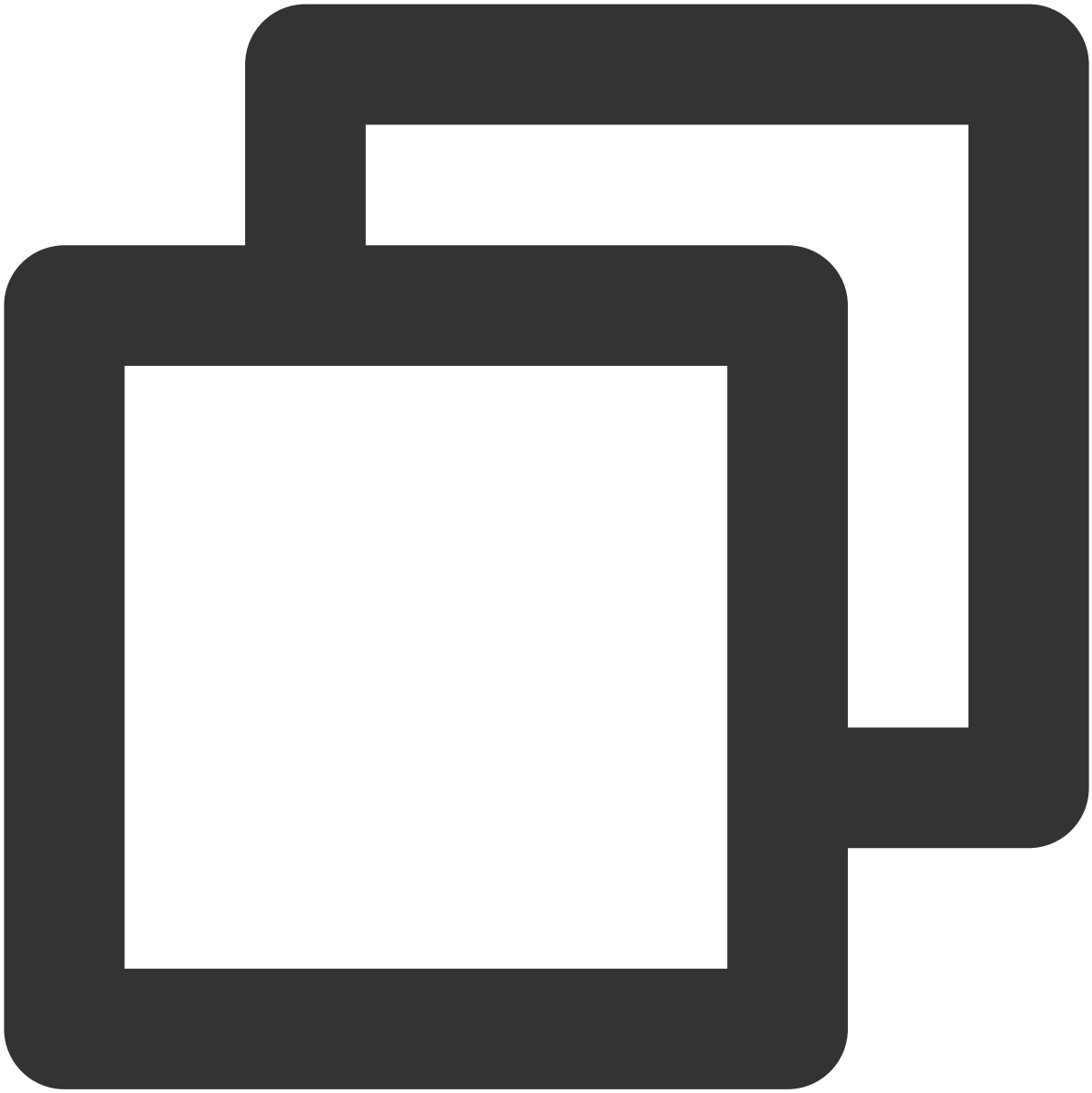


```
const roomEngine = new TUIRoomEngine();
await roomEngine.leaveSeat();
```

Returns：*Promise<void>*

takeUserOnSeatByAdmin

邀请其他人上麦



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.takeUserOnSeatByAdmin({
    seatIndex: 0,
    userId: 'user_1234',
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
     switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
         // 请求被接受
         break;
       case TUIRequestCallbackType.kRequestRejected:
```

```
          // 请求被拒绝
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
      }
    },
});
```

参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index
userId	string	必填	-	用户 ID
timeout	number	必填	-	超时时间，设置为 0 时，无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string>* requestId

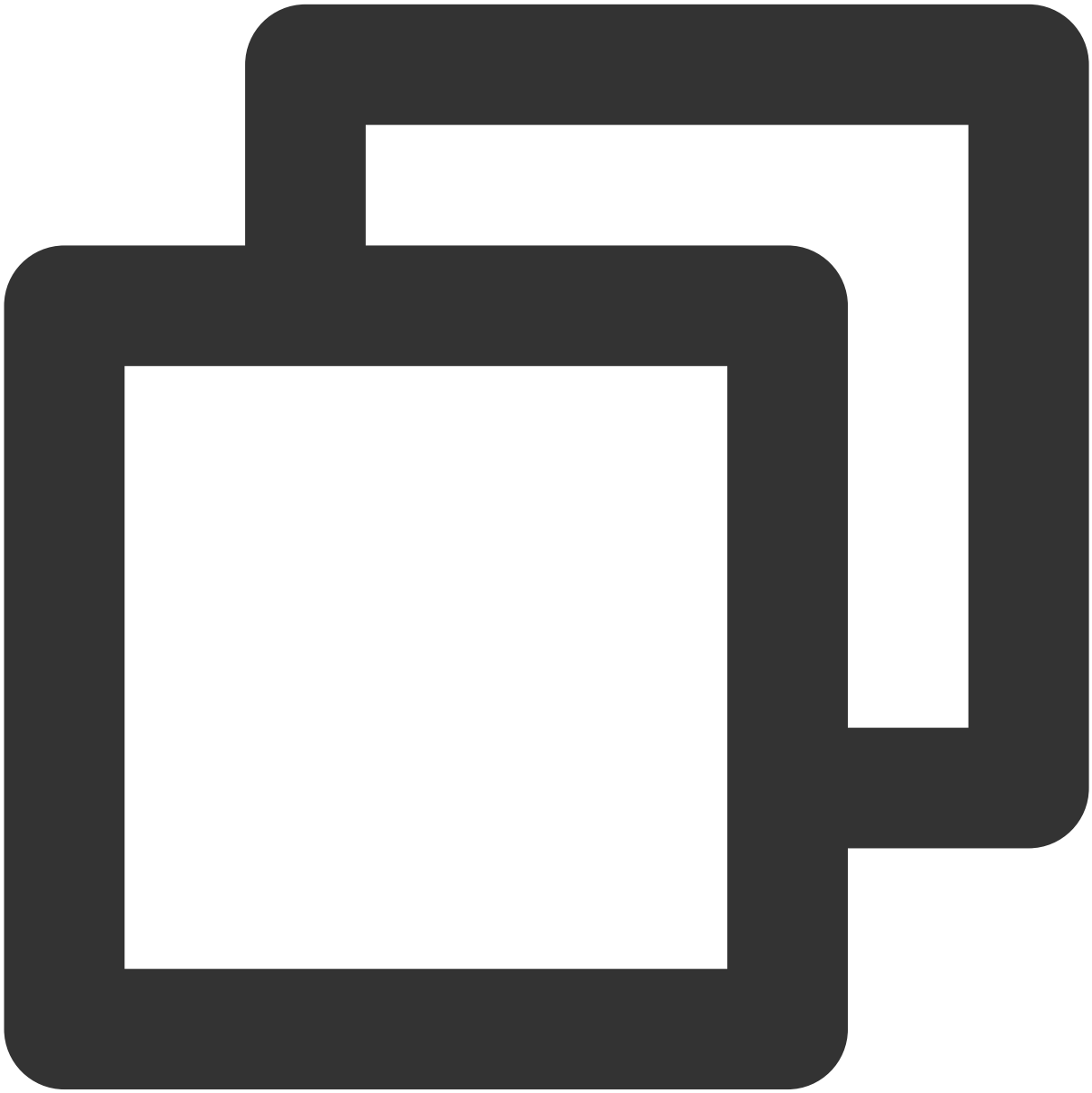
该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

kickUserOffSeatByAdmin

要求其他人下麦



```
const roomEngine = new TUIRoomEngine();
const requestId = await roomEngine.kickUserOffSeatByAdmin({
   seatIndex: 0,
   userId: 'user_1234',
});
```

参数：

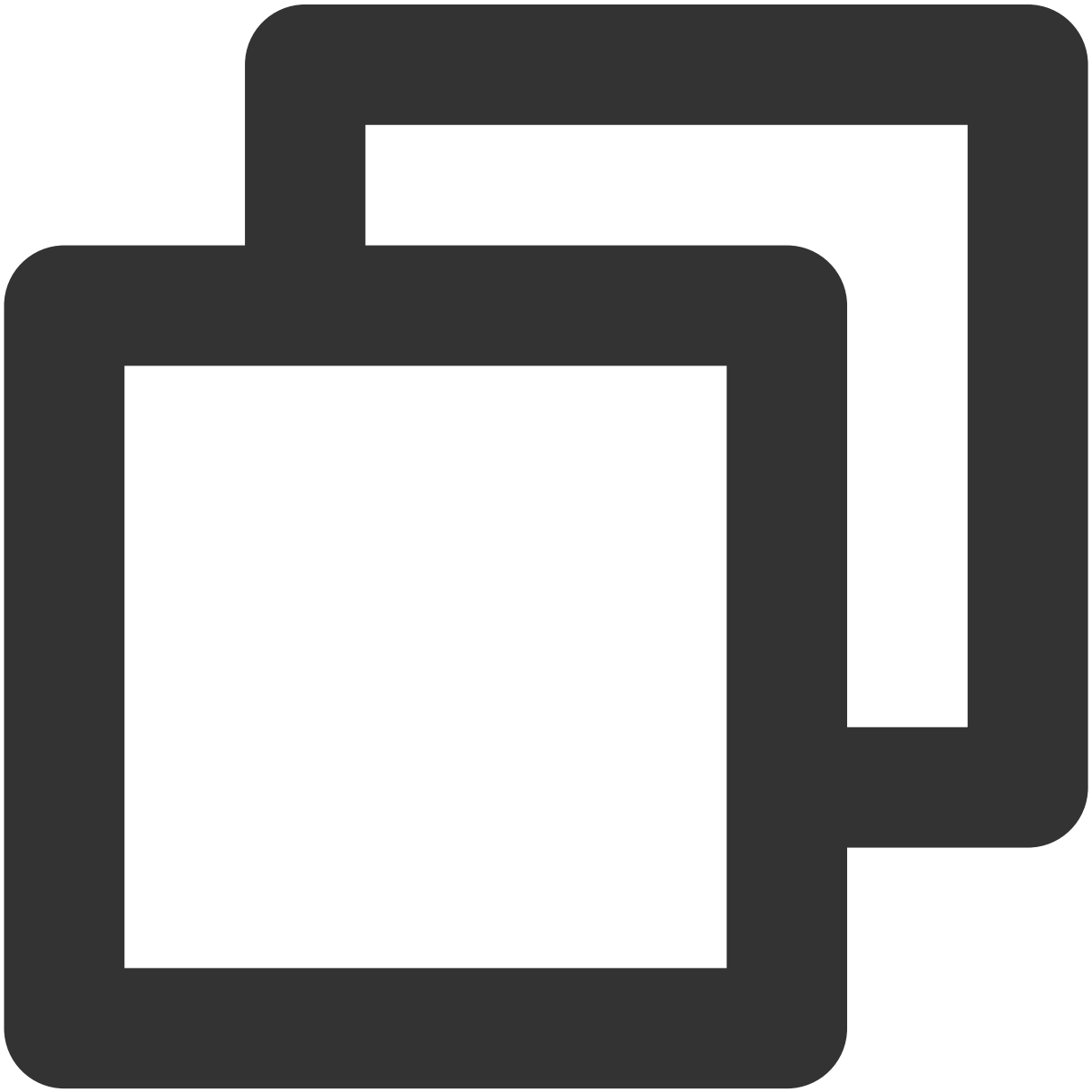
参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index

userId	string	必填	-	用户 ID

Returns：*Promise<void>*

lockSeatByAdmin

锁定某个麦位状态（仅房间主持人和管理员可调用此方法）



```
const roomEngine = new TUIRoomEngine();
```

```
await roomEngine.lockSeatByAdmin({
   seatIndex: 0,
   lockParams: {
      lockSeat: true,
      lockVideo: true,
      lockAudio: true,
   },
});
```

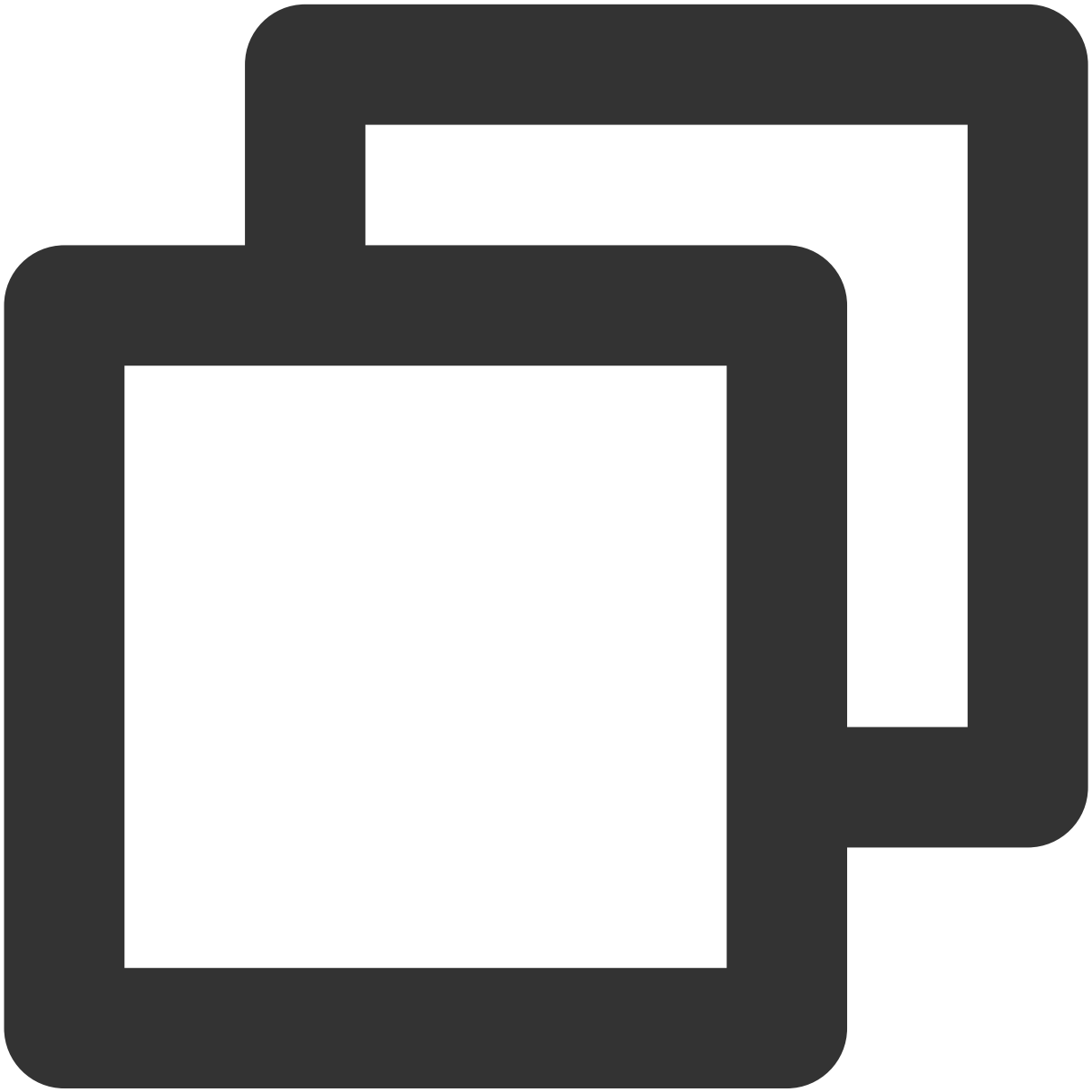
参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位索引
lockParams	TUISeatLockParams	必填	-	锁麦参数

Returns：*Promise<void>*

sendTextMessage

发送文本消息



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendTextMessage({
 messageText: 'hello, everyone',
});
```

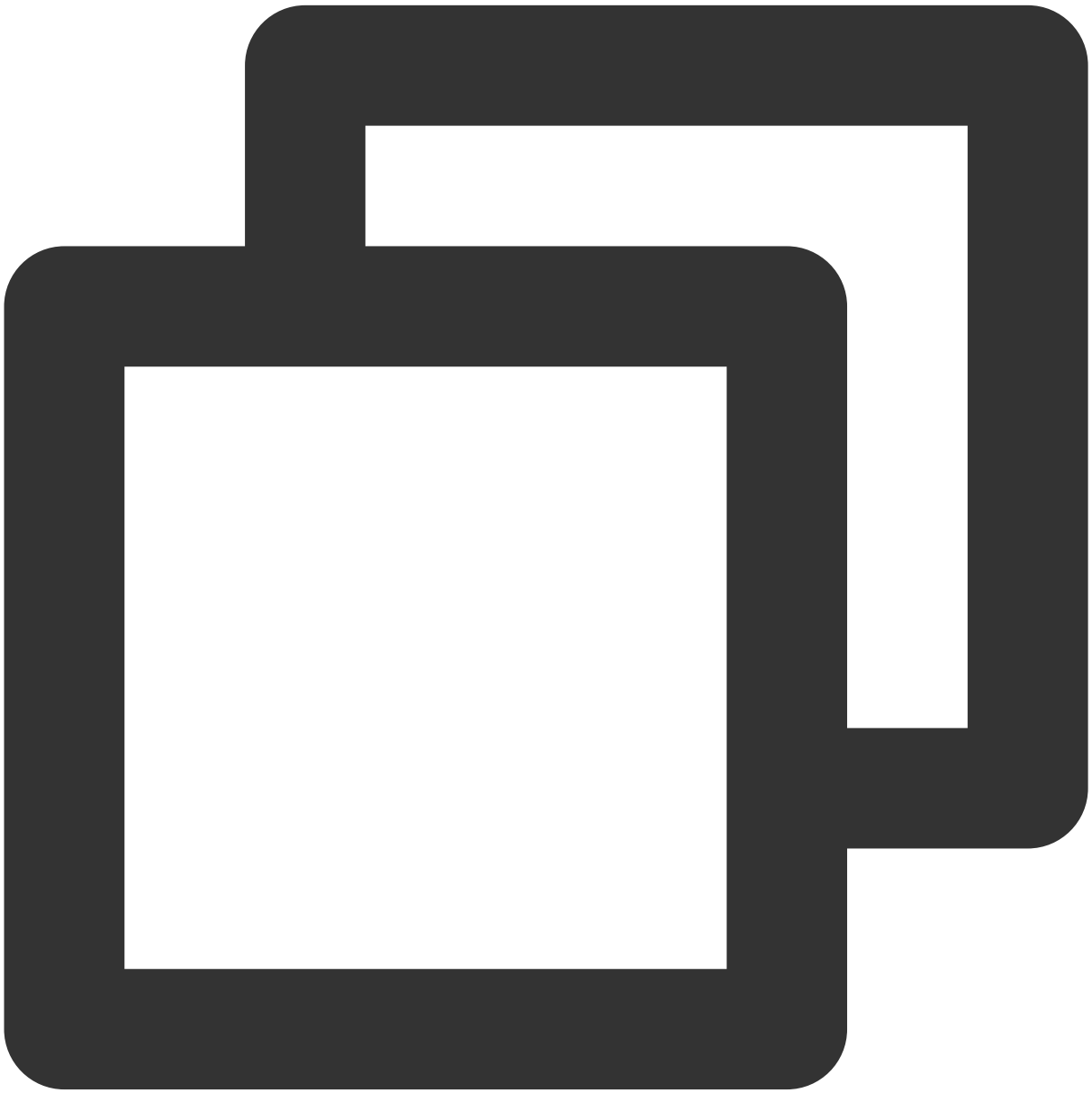
参数：

参数	类型	说明	默认值	含义
messageText	string	必填	-	文本消息内容

Returns：*Promise<void>*

sendCustomMessage

发送自定义消息



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendCustomMessage({
 messageText: '{ data:'', description: ''}',
```

```
});
```

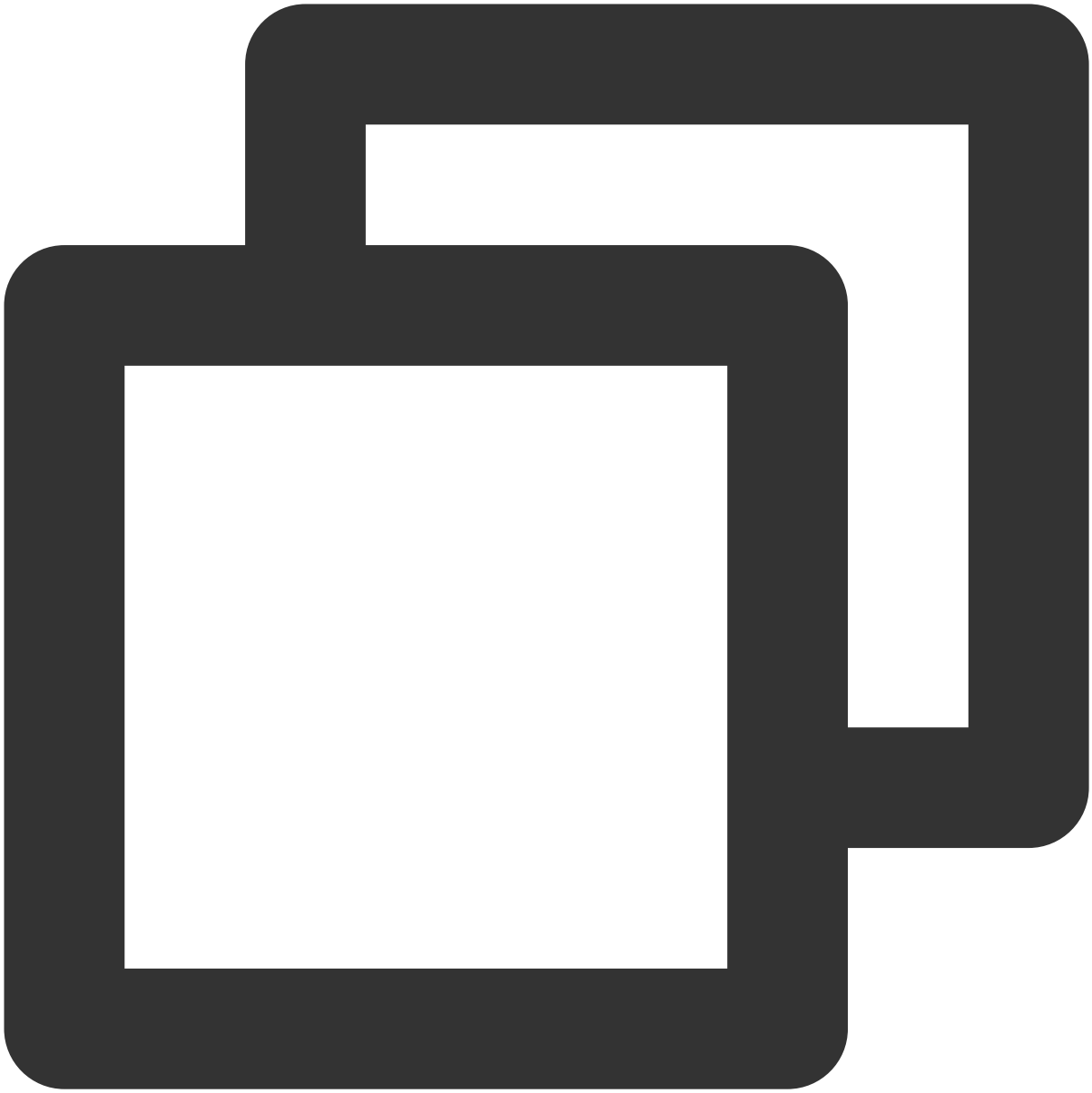
参数：

参数	类型	说明	默认值	含义
messageText	string	必填	-	自定义消息内容

Returns：*Promise<void>*

getCameraDevicesList

获取摄像头设备列表

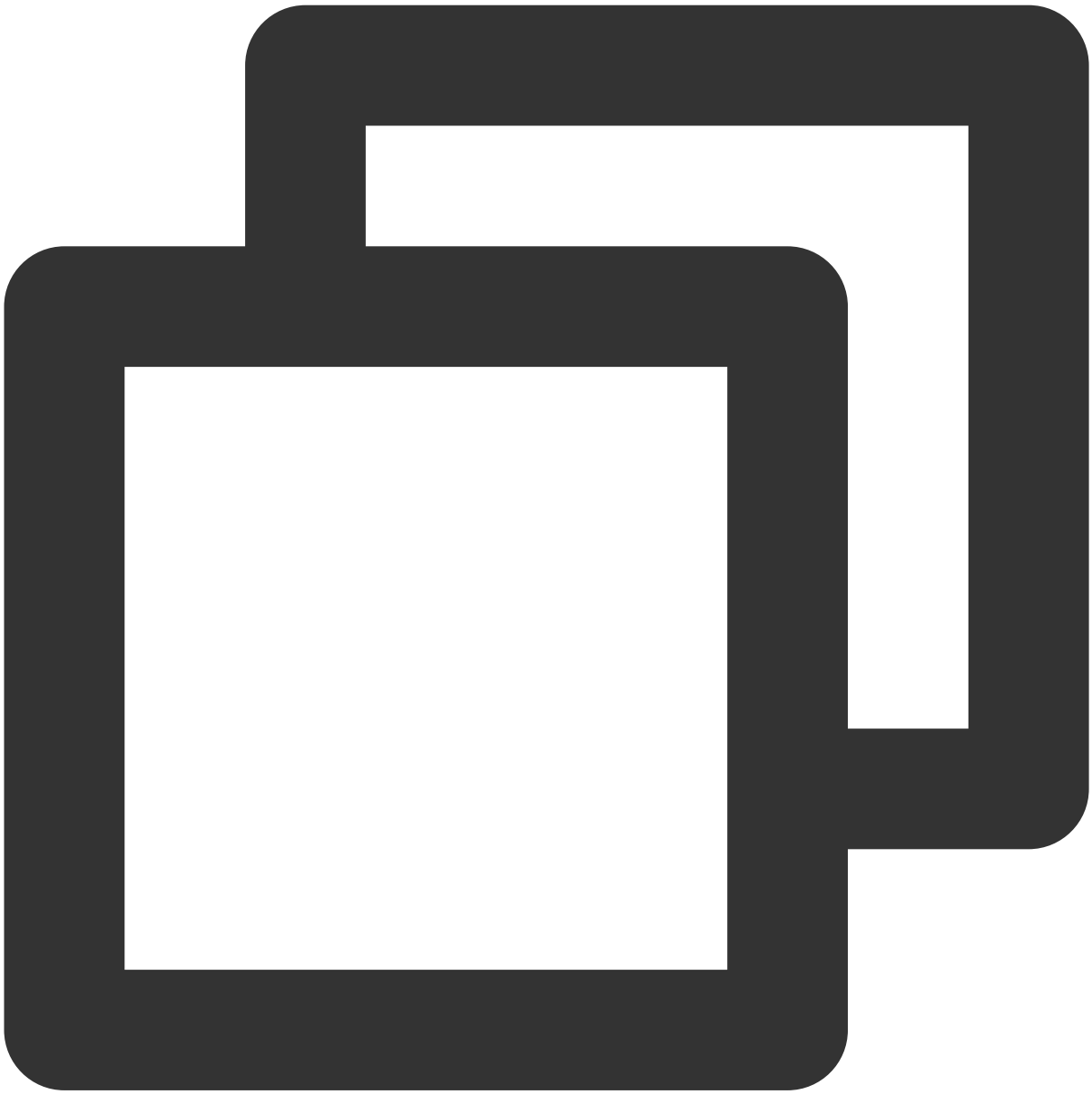


```
const roomEngine = new TUIRoomEngine();
const cameraList = await roomEngine.getCameraDevicesList();
for (i = 0; i < cameraList.length; i++) {
   var camera = cameraList[i];
   console.info("camera deviceName: " + camera.deviceName + " deviceId:" + camera.
}
```

Returns: *Promise*<TRTCDeviceInfo*[]*> cameraList

getMicDevicesList

获取麦克风设备列表



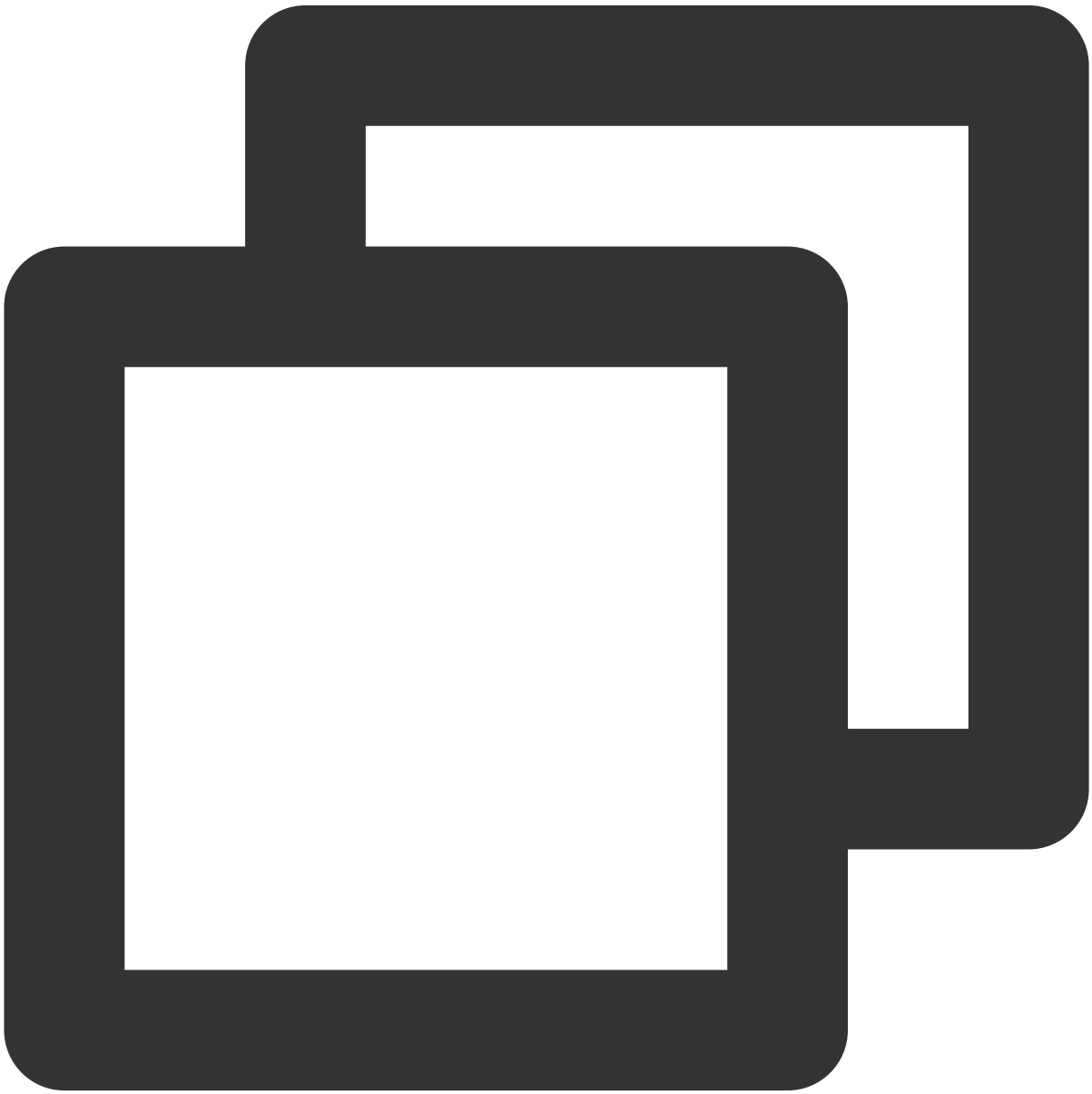
```
const roomEngine = new TUIRoomEngine();
const micList = await roomEngine.getMicDevicesList();
for (i = 0; i < micList.length; i++) {
   var mic = micList[i];
   console.info("mic deviceName: " + mic.deviceName + " deviceId:" + mic.deviceId)
}
```

版权所有：腾讯云计算（北京）有限责任公司

Returns: *Promise*<TRTCDeviceInfo*[]*> micList

getSpeakerDevicesList

获取扬声器设备列表



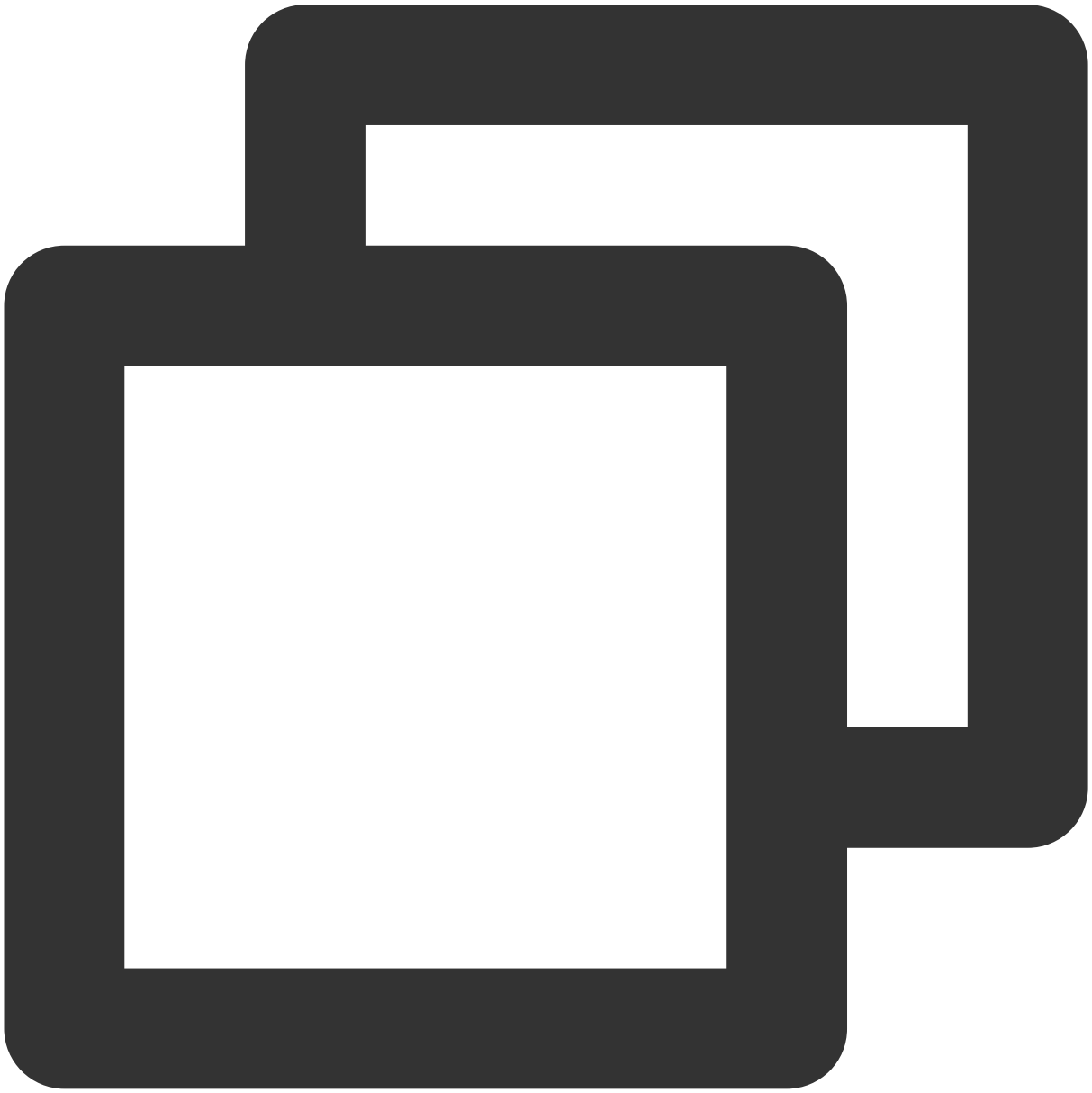
```
const roomEngine = new TUIRoomEngine();
const speakerList = await roomEngine.getSpeakerDevicesList();
for (i = 0; i < speakerList.length; i++) {
```

```
    var speaker = speakerList[i];
    console.info("speaker deviceName: " + speaker.deviceName + " deviceId:" + speak
}
```

Returns: *Promise<*TRTCDeviceInfo*[]>* speakerList

setCurrentCameraDevice

设置要使用的摄像头设备



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentCameraDevice({ deviceId: '' });
```

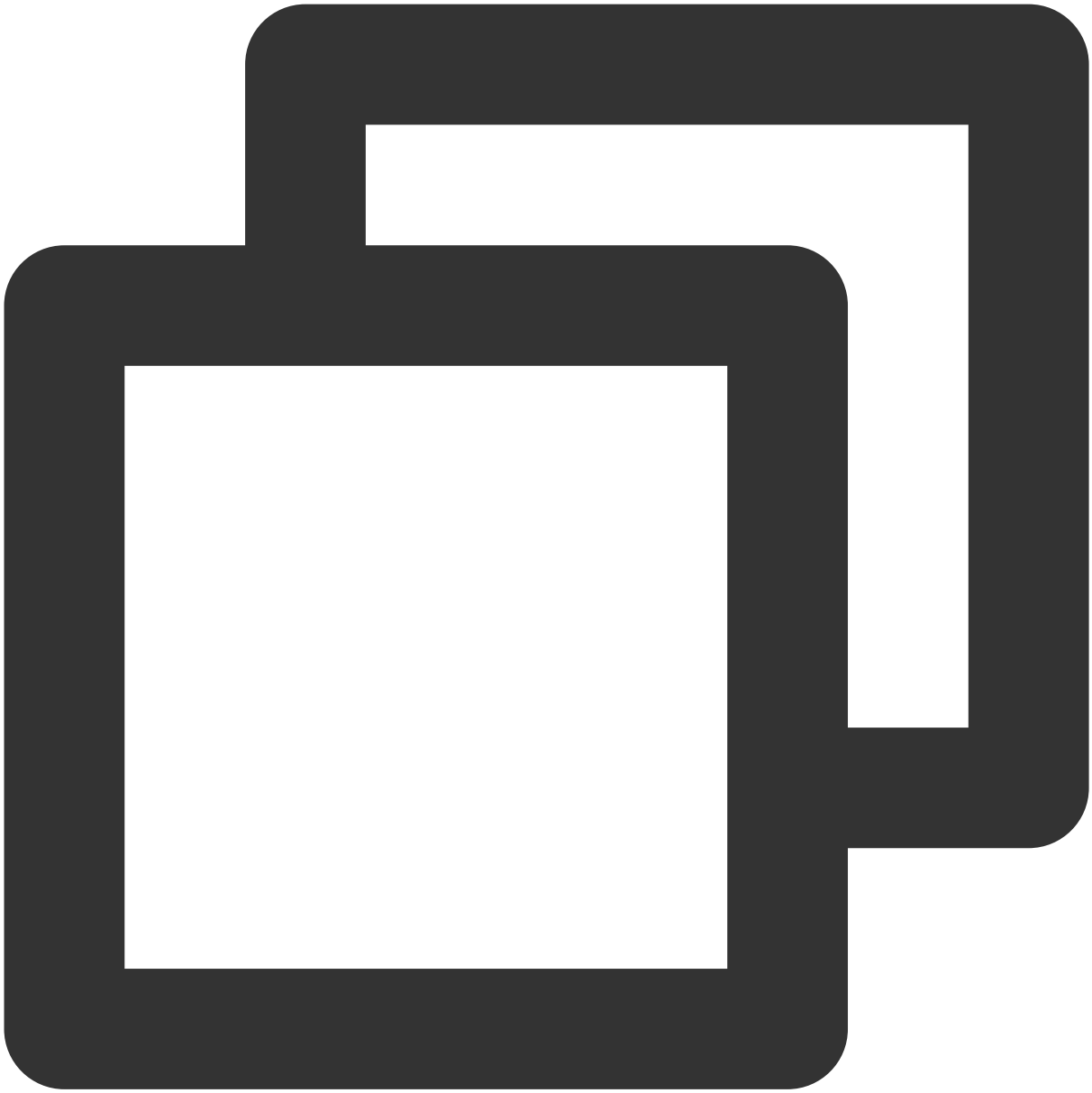
参数：

参数	类型	说明	默认值	含义
deviceId	string	必填	-	从 getCameraDevicesList 中得到的设备 ID

Returns：*void*

setCurrentMicDevice

设置要使用的麦克风设备



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentMicDevice({ deviceId: '' });
```

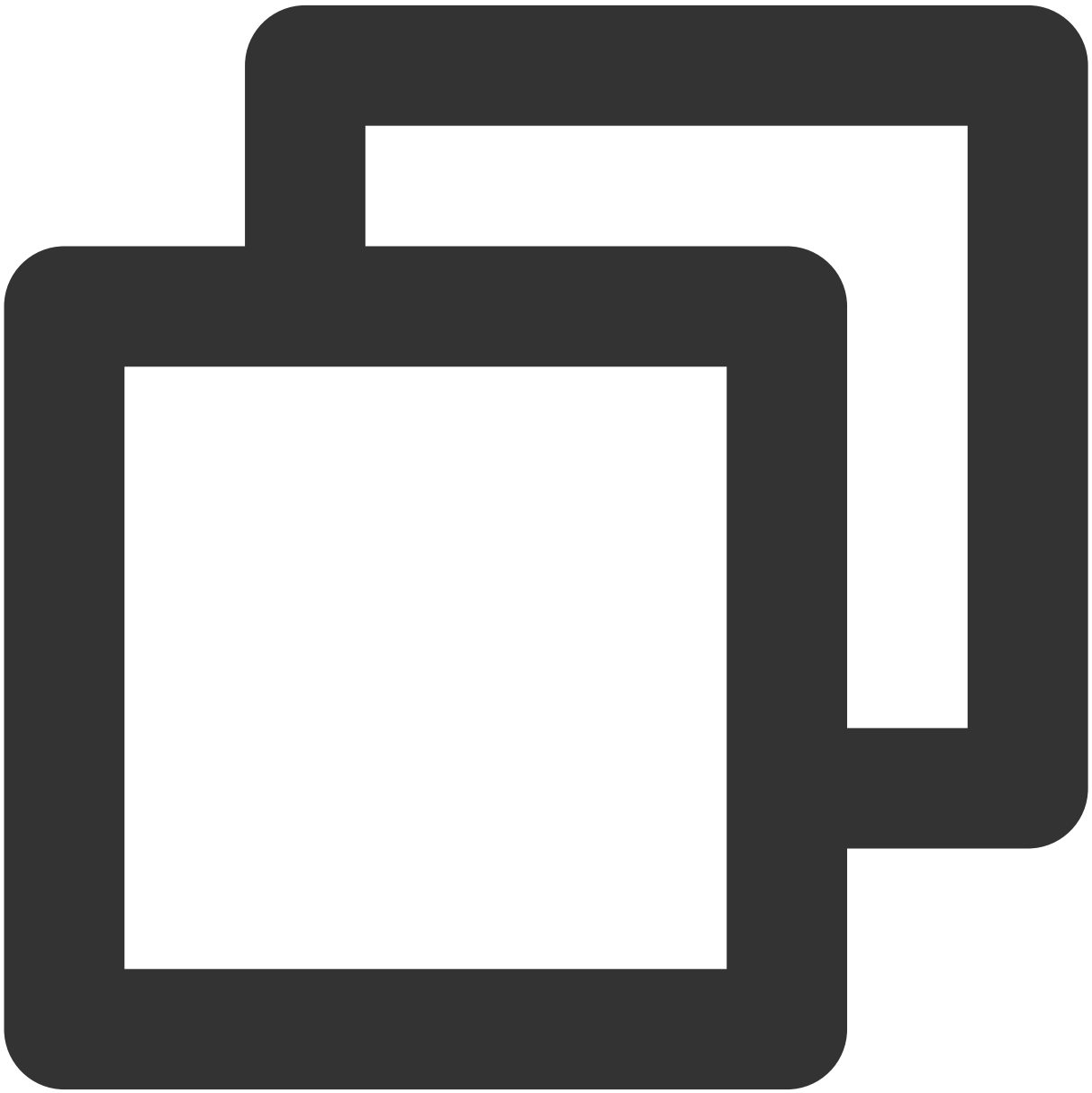
参数：

参数	类型	说明	默认值	含义
deviceId	string	必填	-	从 getMicDevicesList 中得到的设备 ID

Returns：*void*

setCurrentSpeakerDevice

设置要使用的扬声器设备



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentSpeakerDevice({ deviceId: '' });
```

参数：

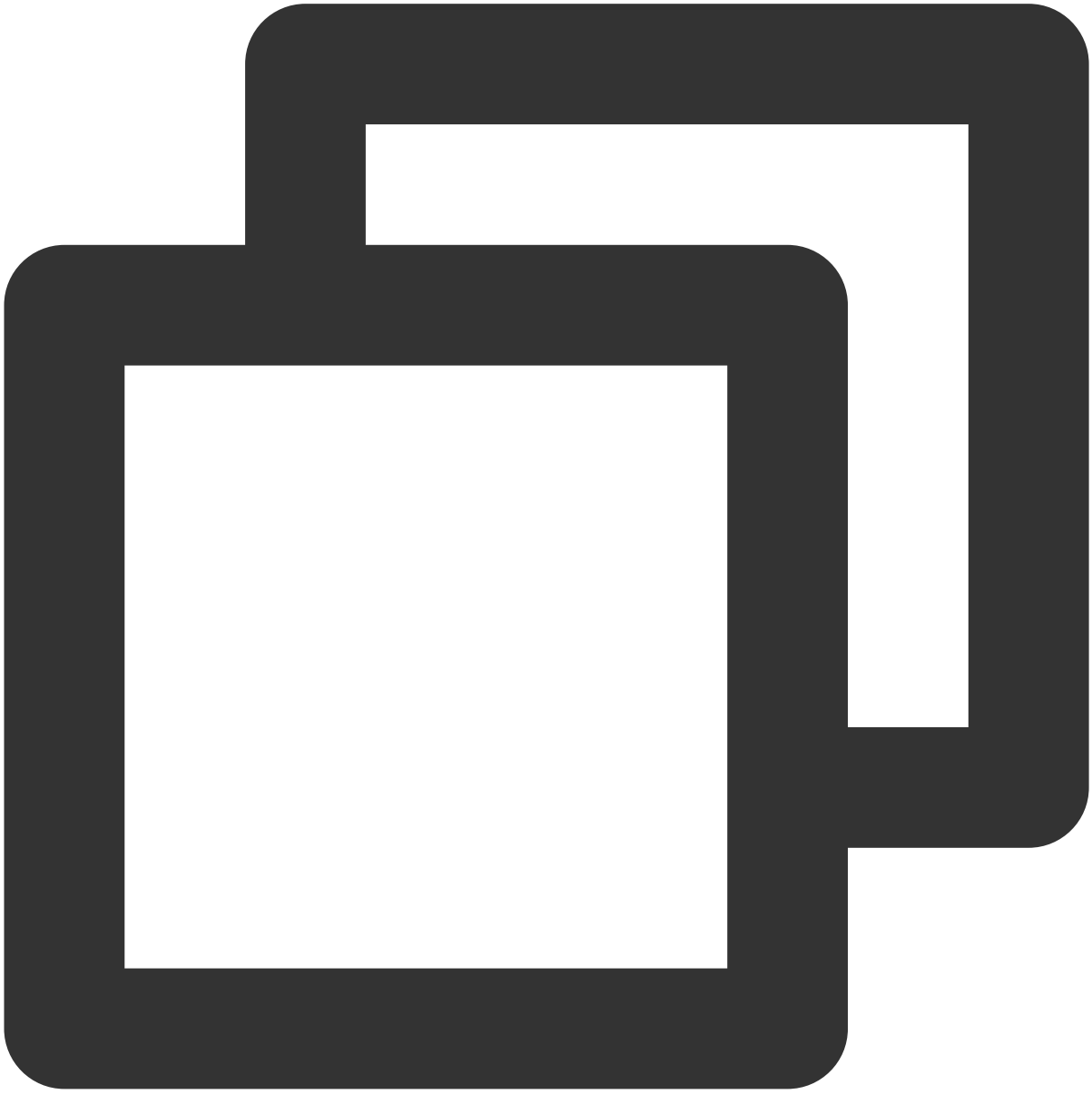
参数	类型	说明	默认值	含义

deviceId	string	必填	-	从 getSpeakerDevicesList 中得到的设备 ID

Returns：*void*

getCurrentCameraDevice

获取当前正在使用的摄像头设备



```
const roomEngine = new TUIRoomEngine();
```

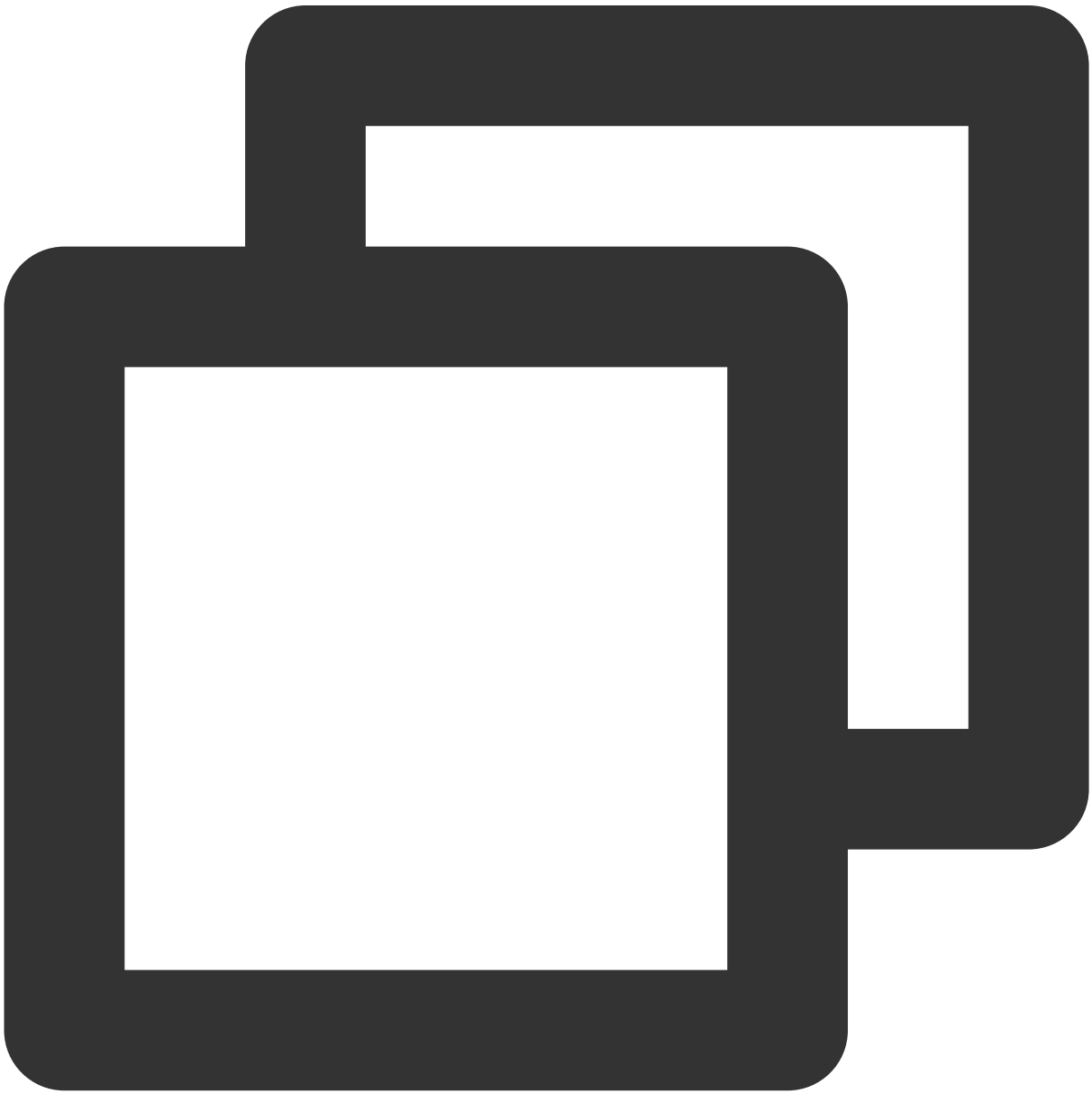
```
const currentCameraDevice = roomEngine.getCurrentCameraDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

getCurrentMicDevice

获取当前正在使用的麦克风设备



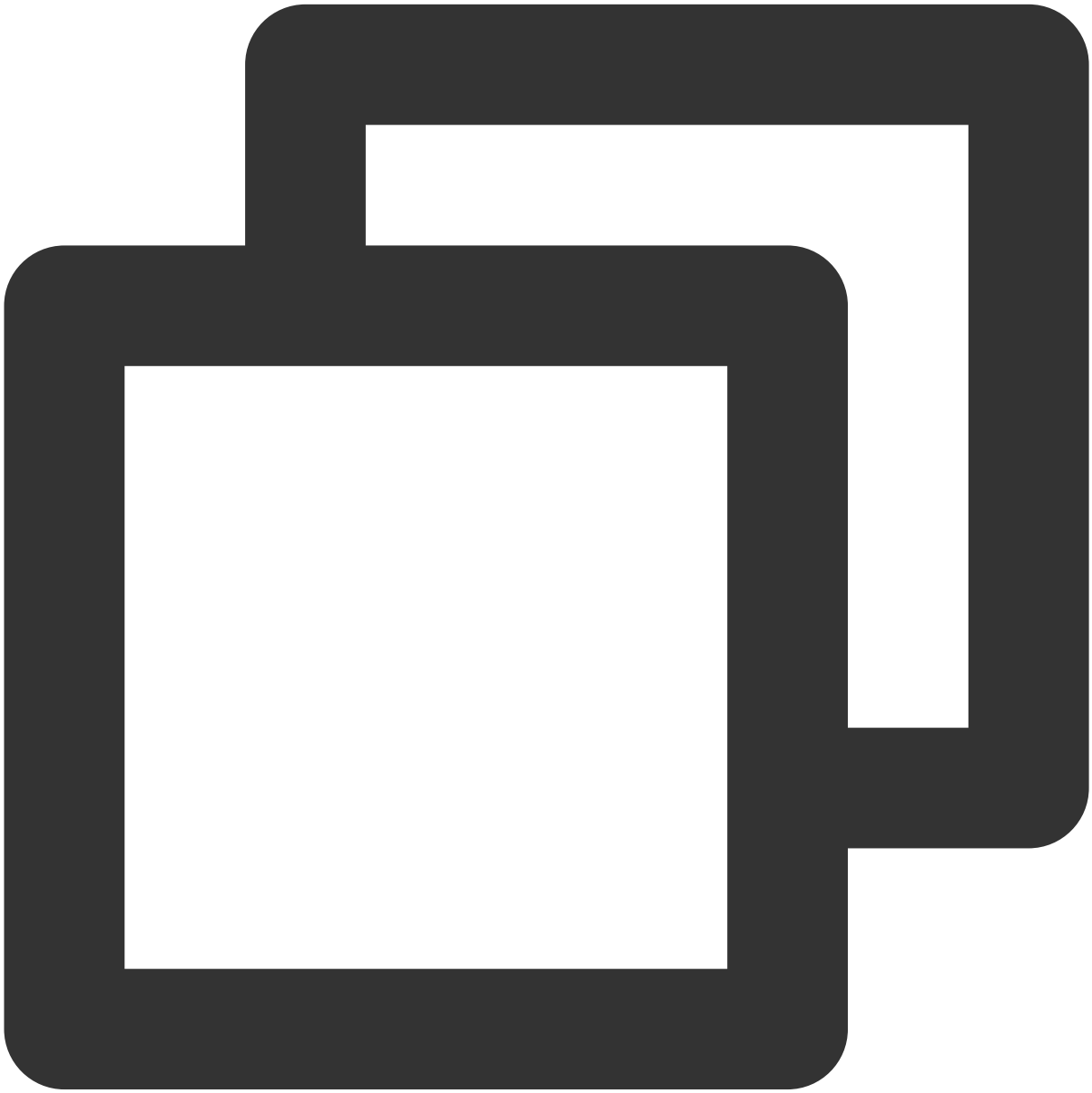
```
const roomEngine = new TUIRoomEngine();
const currentMicDevice = roomEngine.getCurrentMicDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

getCurrentSpeakerDevice

获取当前正在使用的扬声器设备



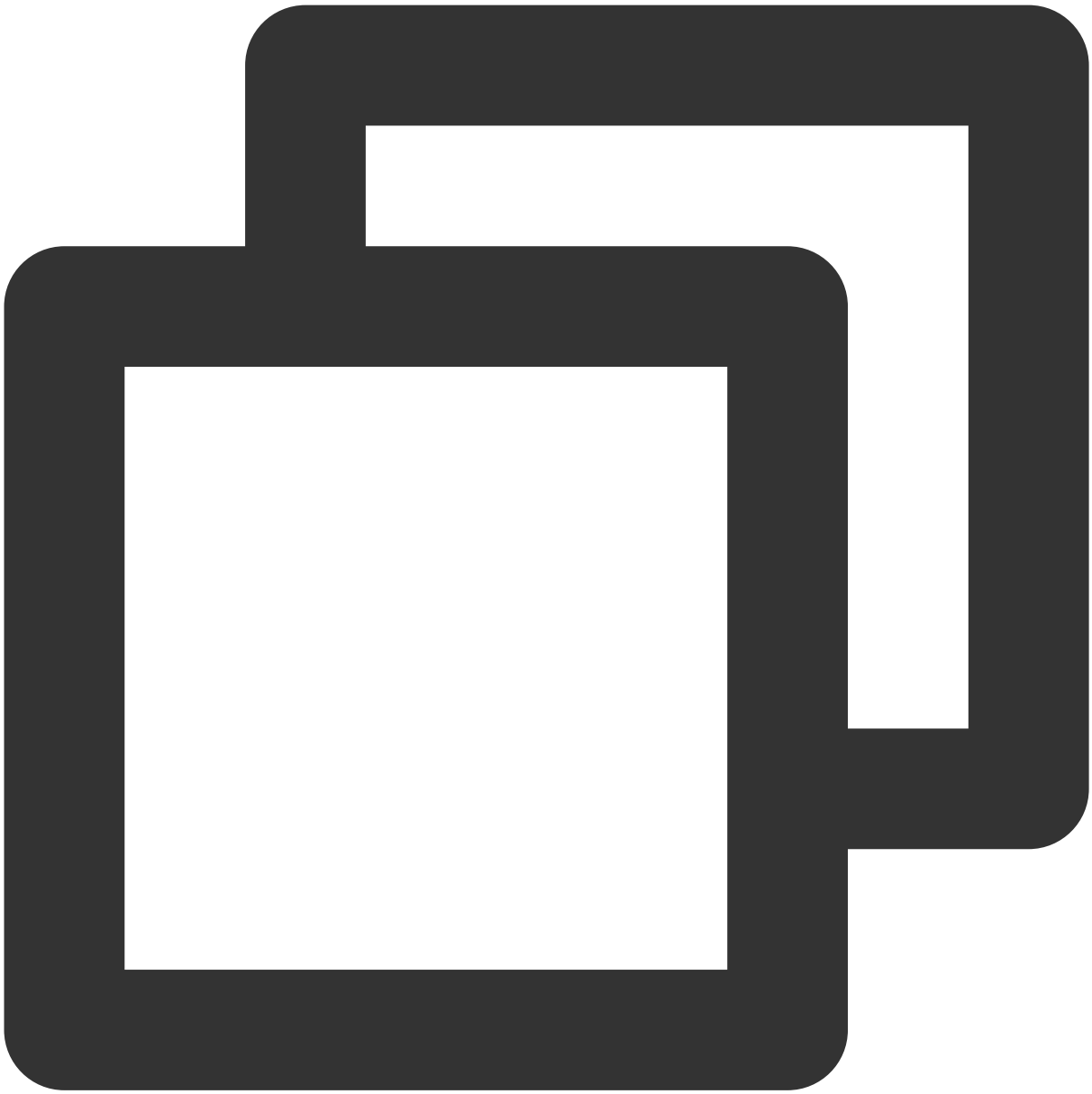
```
const roomEngine = new TUIRoomEngine();
const currentSpeakerDevice = roomEngine.getCurrentSpeakerDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

startCameraDeviceTest

开始进行摄像头测试



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startCameraDeviceTest({ view: 'test-preview' });
```

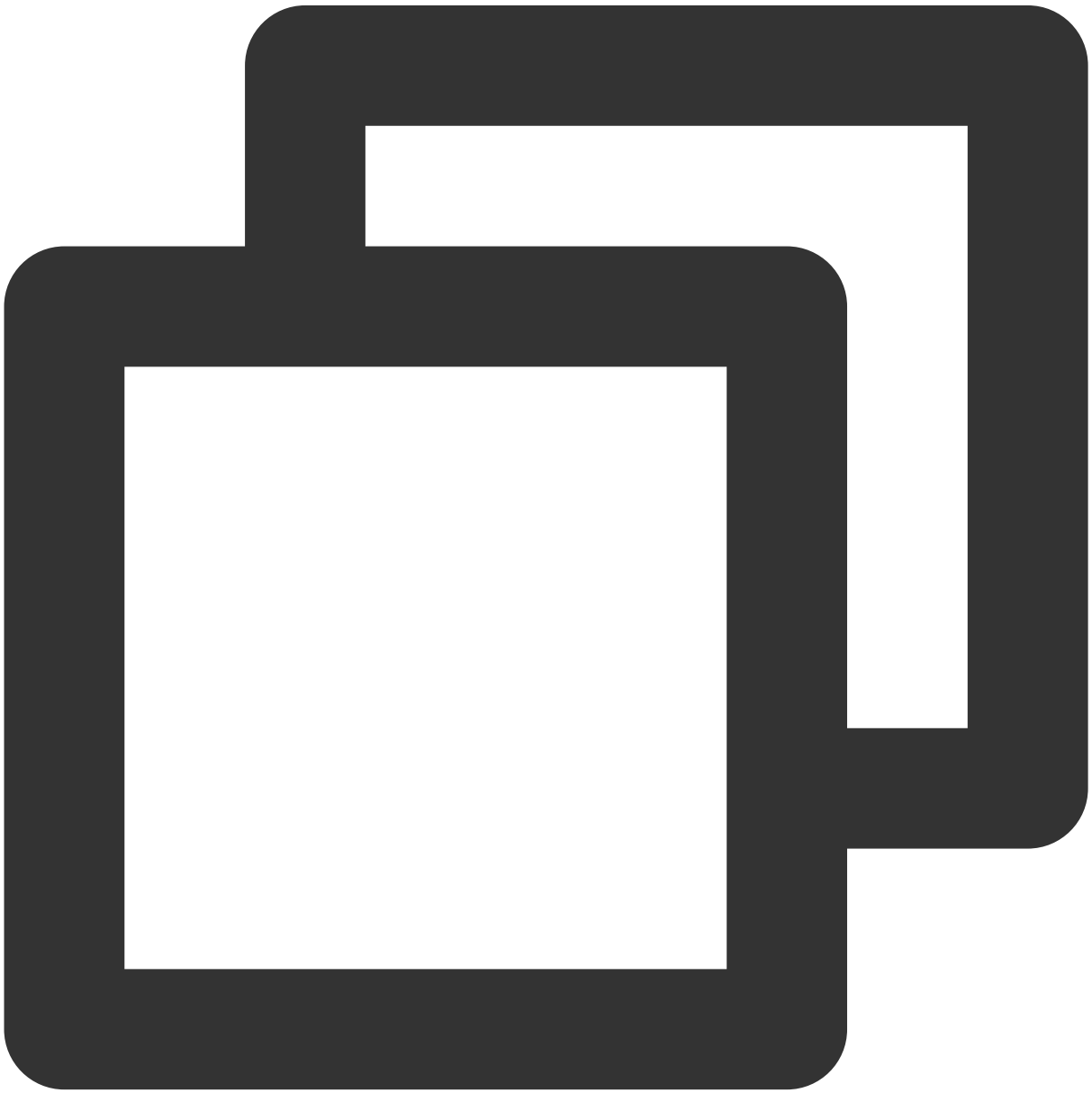
参数：

参数	类型	说明	默认值	含义
view	string	必填	-	显示摄像头测试的视频区域, 传入的 view 为承载预览画面 div 元素的 Id

Returns：*Promise<void>*

stopCameraDeviceTest

停止摄像头测试

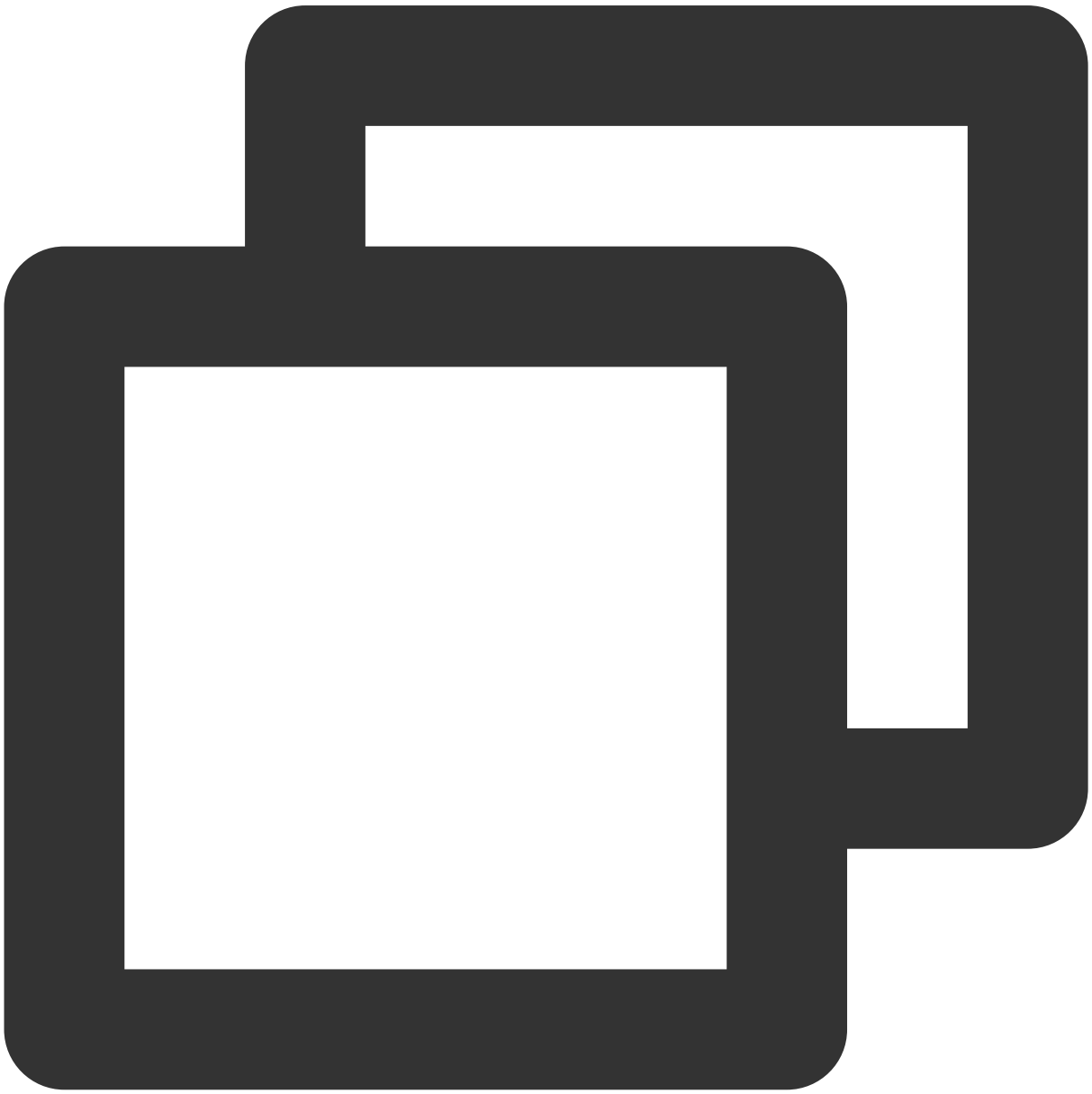


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopCameraDeviceTest();
```

Returns：*Promise<void>*

on

监听 roomEngine 的事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(event, func);
```

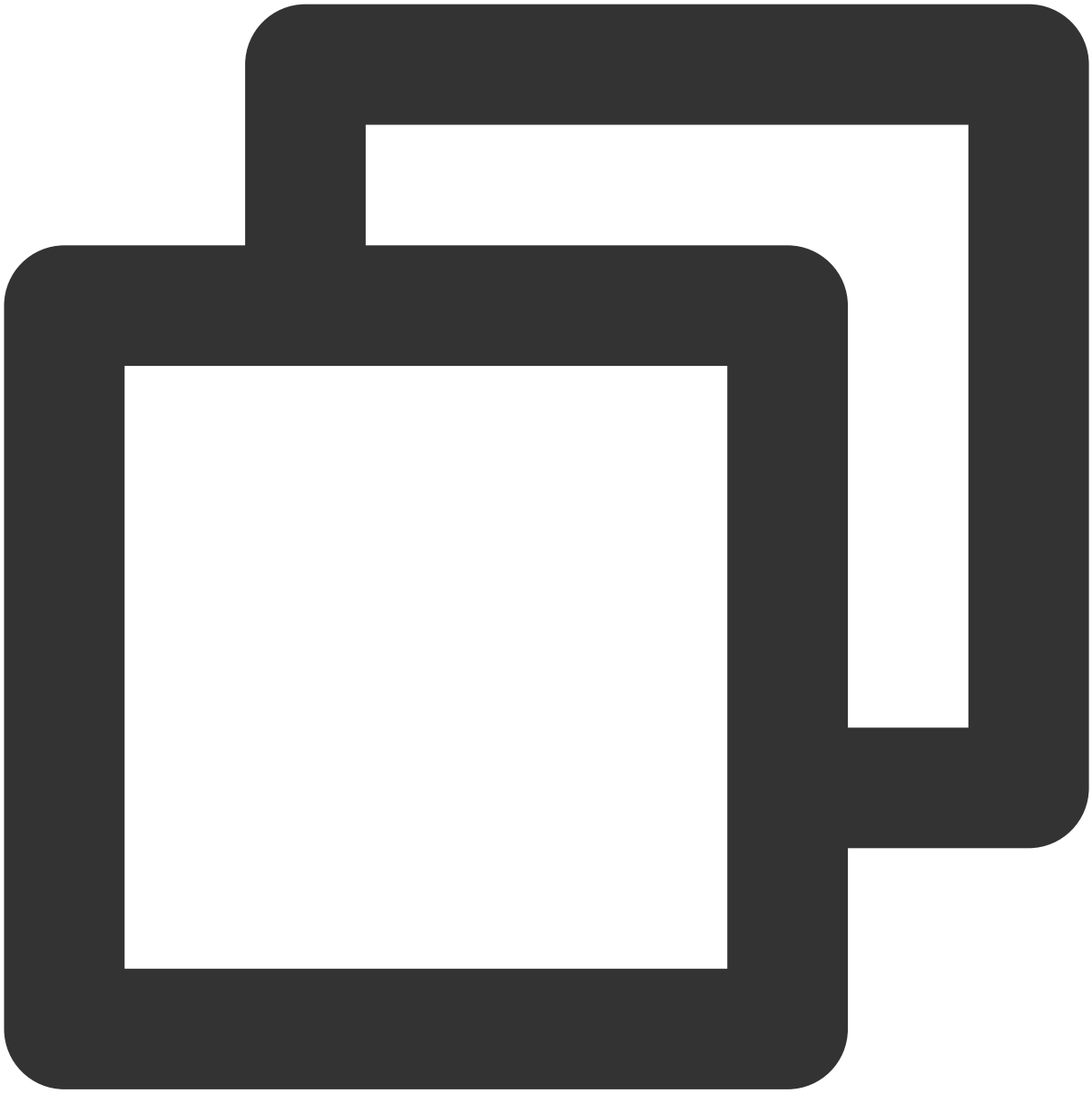
参数：

参数	类型	说明	默认值	含义
event	TUIRoomEvents	必填	-	TUIRoomEngine 事件列表
func	Function	必填	-	事件回调函数

Returns：*void*

off

取消监听 roomEngine 的事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.off(event, func);
```

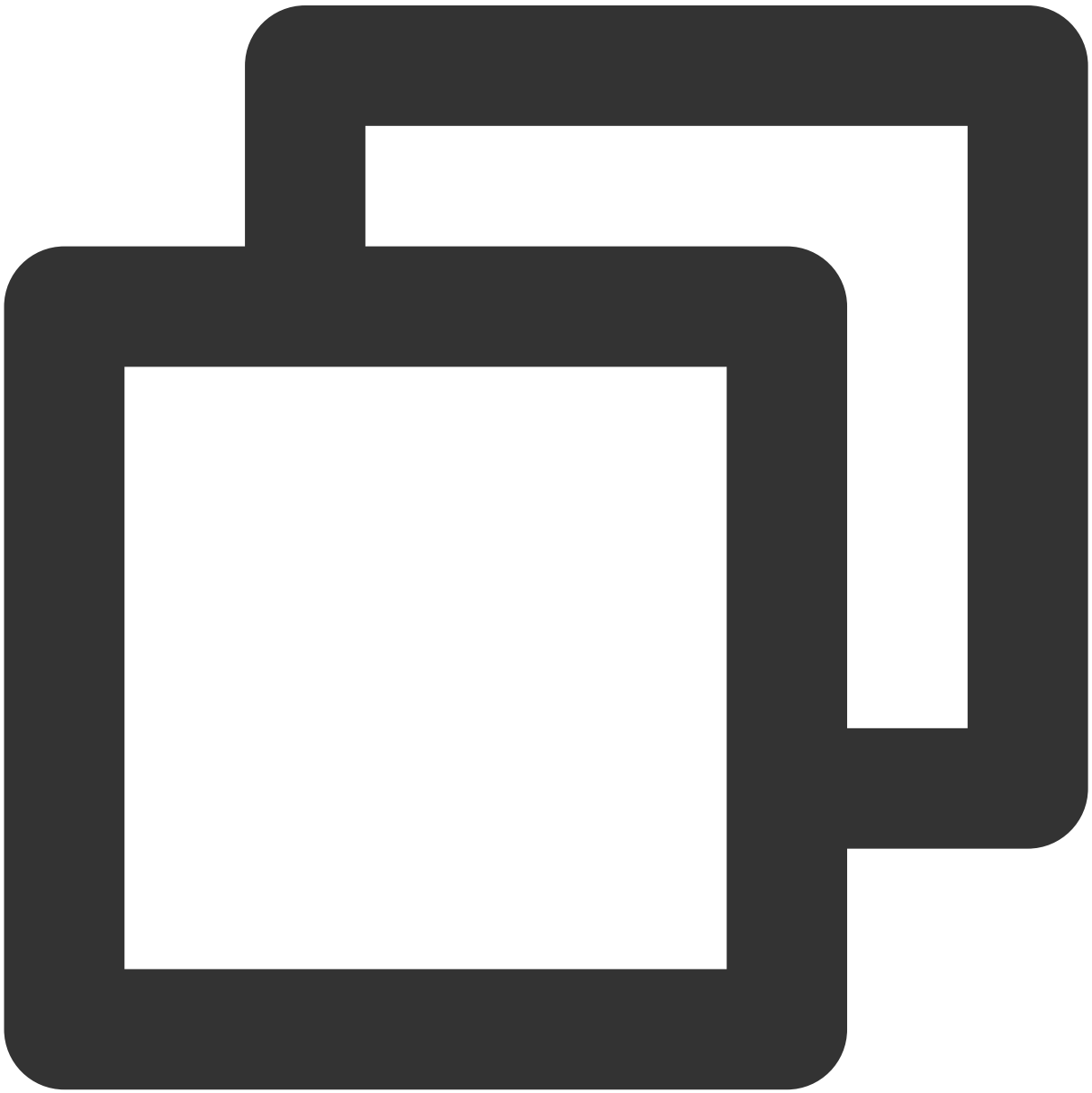
参数：

参数	类型	说明	默认值	含义
event	TUIRoomEvents	必填	-	TUIRoomEngine 事件列表
func	Function	必填	-	事件回调函数

Returns：*void*

getTRTCCloud

获取 trtcCloud 实例, web 端 trtcCloud 能力请查看：TRTCCloud API 文档

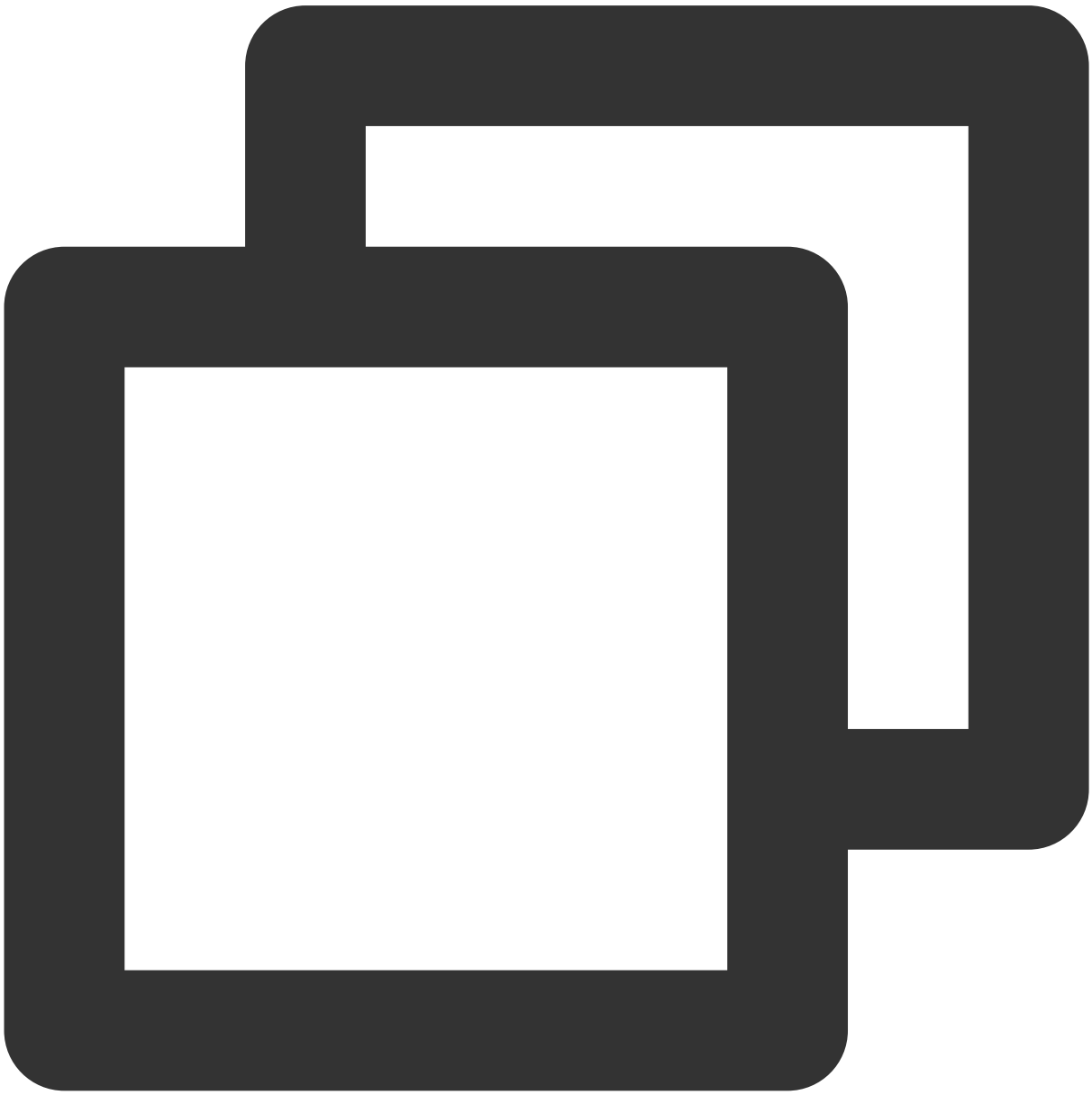


```
const roomEngine = new TUIRoomEngine();
const trtcCloud = roomEngine.getTRTCCloud();
```

Returns：TRTCCloud

getTIM

获取 tim 实例, web 端 tim 能力请查看： IM API 文档



```
const roomEngine = new TUIRoomEngine();
const trtcCloud = roomEngine.getTIM();
```

Returns：*TIM*

TUIRoomEvents

最近更新时间：2023-11-14 17:06:00

TUIRoomEvent API 简介

TUIRoomEvent API 是音视频通话组件的**事件接口**。

事件列表

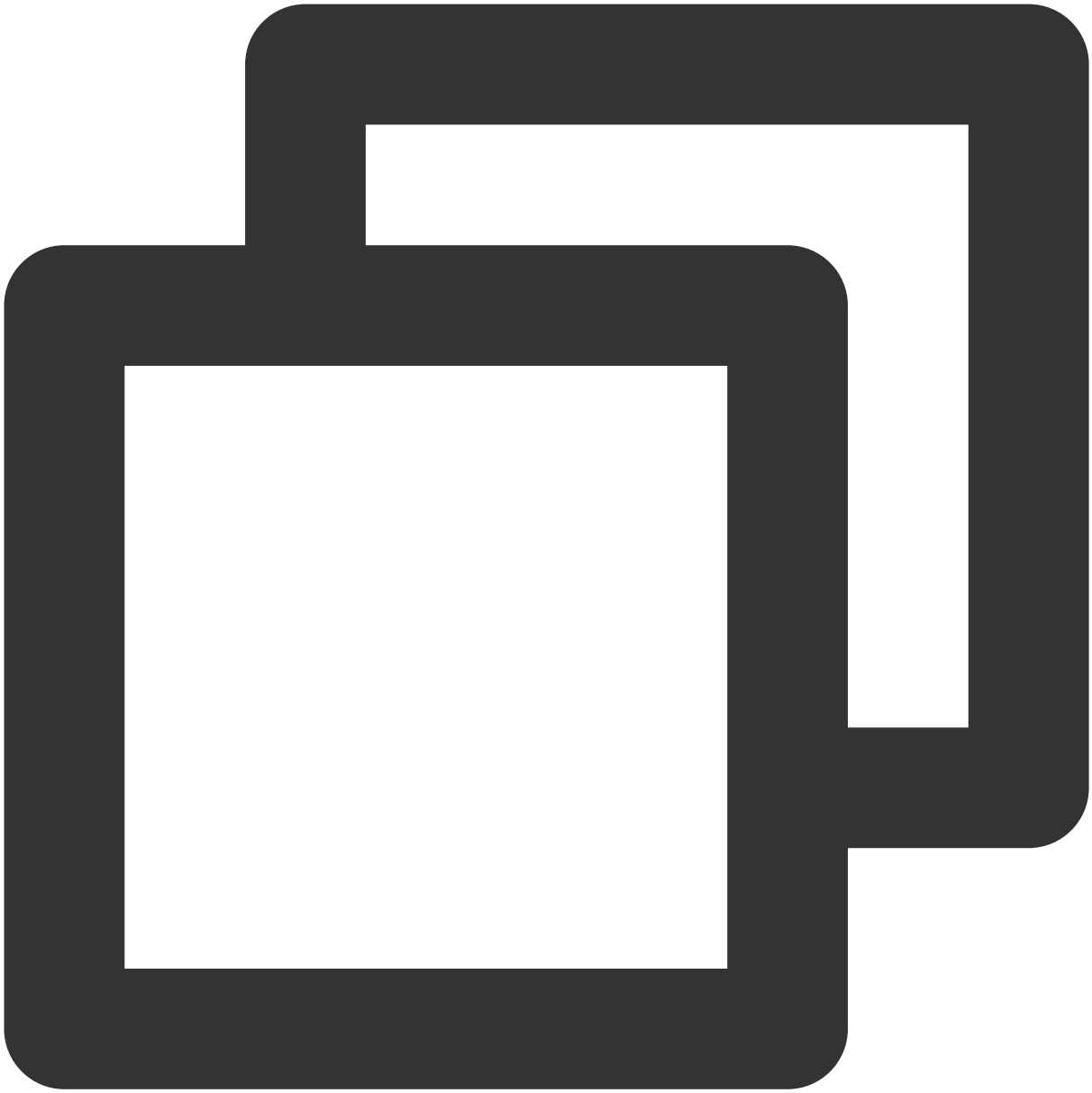
EVENT	描述
TUIRoomEvents.onError	错误事件
TUIRoomEvents.onKickedOutOfRoom	踢出房间事件
TUIRoomEvents.onKickedOffLine	当前用户被踢下线事件
TUIRoomEvents.onUserSigExpired	userSig 过期事件
TUIRoomEvents.onRoomDismissed	主持人销毁房间事件
TUIRoomEvents.onRoomNameChanged	房间名称修改事件
TUIRoomEvents.onRoomSpeechModeChanged	房间发言模式修改事件
TUIRoomEvents.onAllUserCameraDisableChanged	所有成员摄像头使用权限改变事件
TUIRoomEvents.onAllUserMicrophoneDisableChanged	所有成员麦克风使用权限改变事件
TUIRoomEvents.onSendMessageForAllUserDisableChanged	所有成员发送消息状态改变事件
TUIRoomEvents.onRoomMaxSeatCountChanged	房间最大麦位数修改事件
TUIRoomEvents.onRemoteUserEnterRoom	远端用户进入房间事件
TUIRoomEvents.onRemoteUserLeaveRoom	远端用户离开房间事件
TUIRoomEvents.onUserRoleChanged	用户角色改变事件
TUIRoomEvents.onUserVideoStateChanged	用户视频状态改变事件
TUIRoomEvents.onUserAudioStateChanged	用户音频状态改变事件

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEvents.onSendMessageForUserDisableChanged	用户发送消息状态事件
TUIRoomEvents.onUserVoiceVolumeChanged	用户音量改变事件
TUIRoomEvents.onUserNetworkQualityChanged	用户网络质量改变事件
TUIRoomEvents.onSeatListChanged	麦位列表改变事件
TUIRoomEvents.onKickedOffSeat	用户被踢下麦事件
TUIRoomEvents.onRequestReceived	请求接收事件
TUIRoomEvents.onRequestCancelled	请求取消事件
TUIRoomEvents.onReceiveTextMessage	接收文本消息事件
TUIRoomEvents.onReceiveCustomMessage	接收自定义消息事件
TUIRoomEvents.onDeviceChange	设备变更事件
TUIRoomEvents.onUserScreenCaptureStopped	屏幕分享停止事件 当用户使用浏览器自带的**停止分享**按钮结束屏幕共享时，该用户会收到 'onUserScreenCaptureStopped' 事件用来修改屏幕共享状态。

onError

错误事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onError, (error) => {
 console.log('TUIRoomError error', error);
})
```

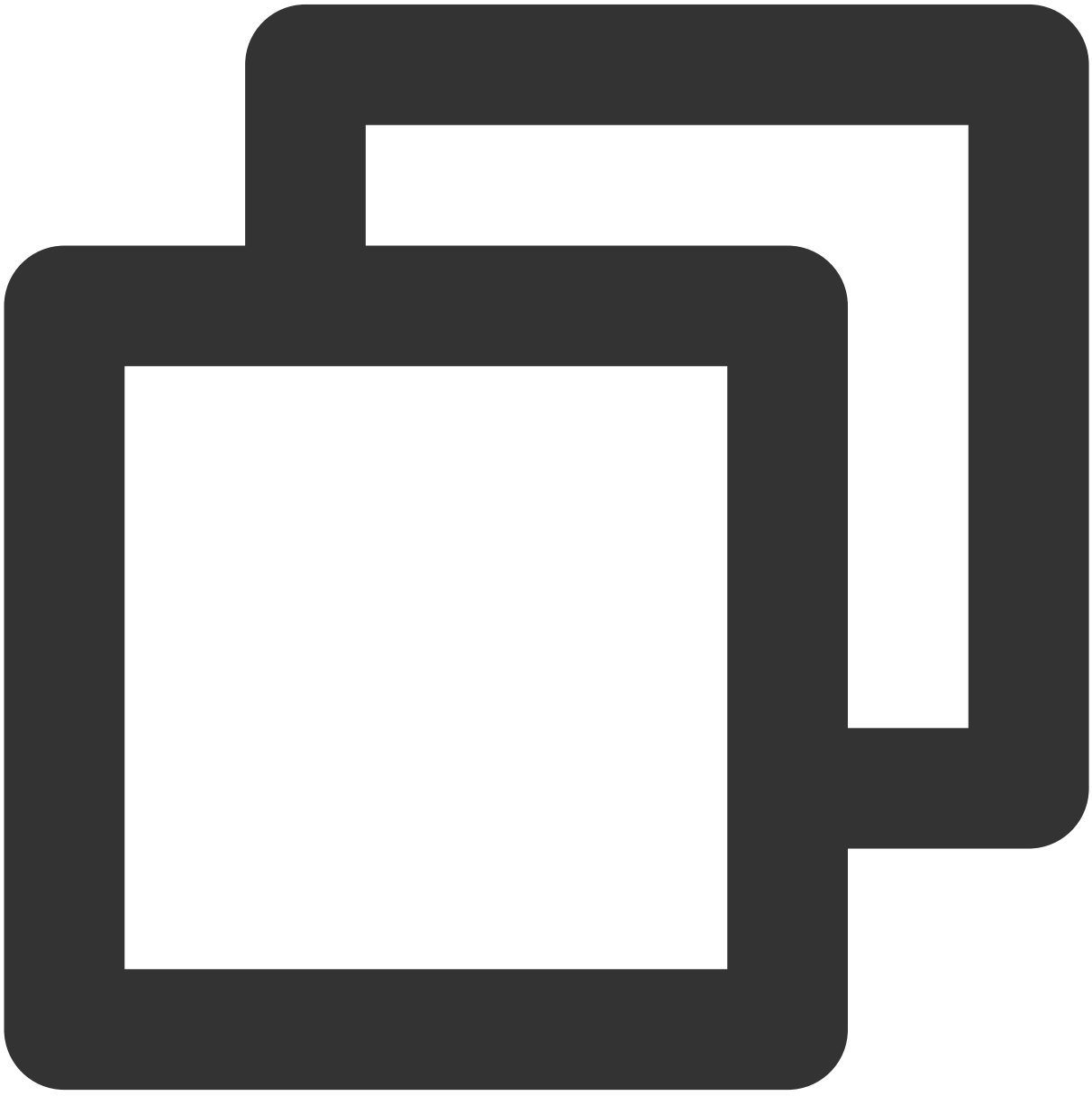
参数如下表所示：

参数	类型	含义
code	number	错误代码

message	string	错误信息

onKickedOutOfRoom

踢出房间事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOutOfRoom, ({ roomId, message }) => {
  console.log('roomEngine.onKickedOutOfRoom', roomId, message);
```

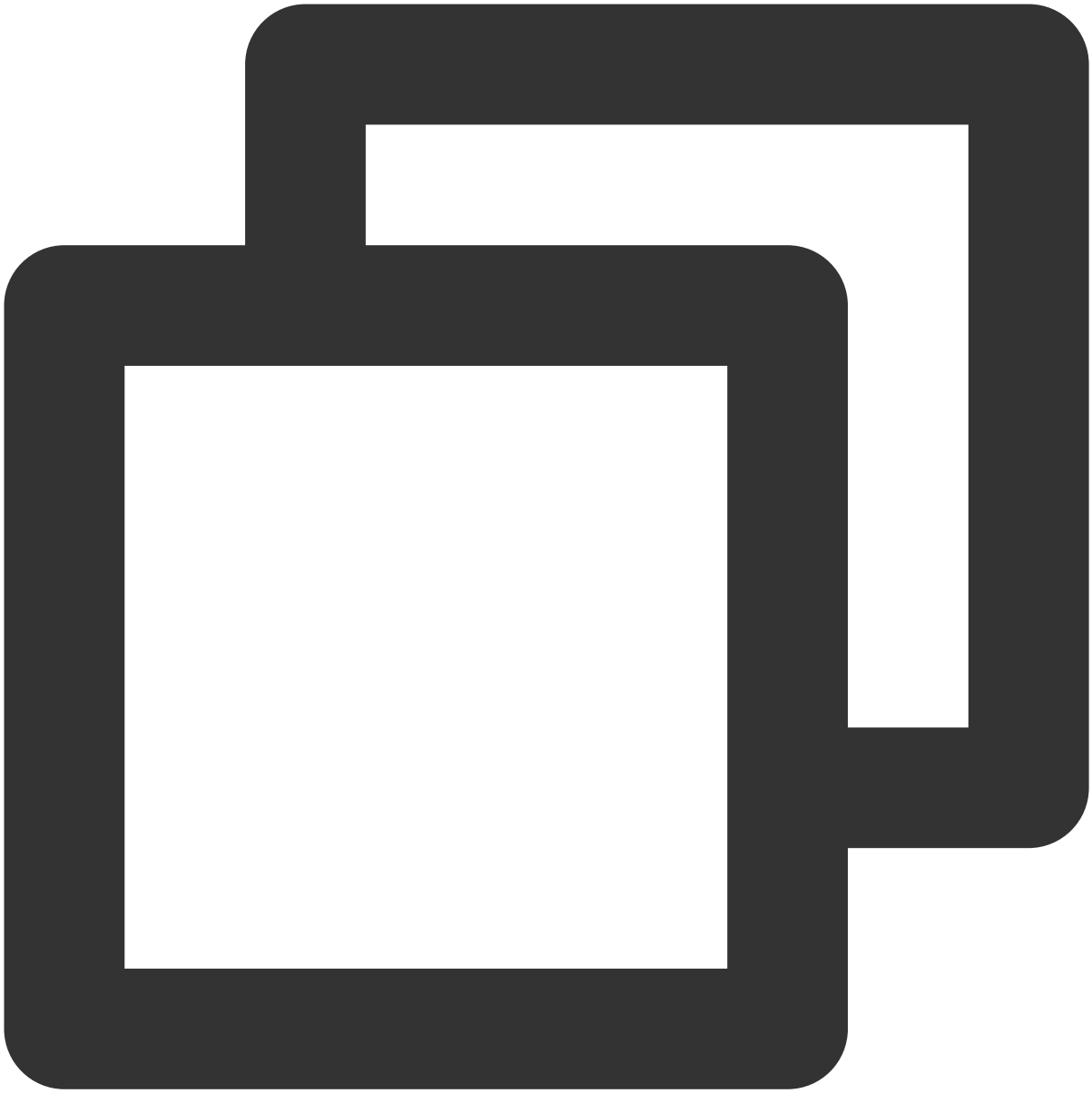
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
message	string	踢出房间信息

onKickedOffLine

当前用户被踢下线事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOffLine, ({ message }) => {
  console.log('roomEngine.onKickedOffLine', message);
});
```

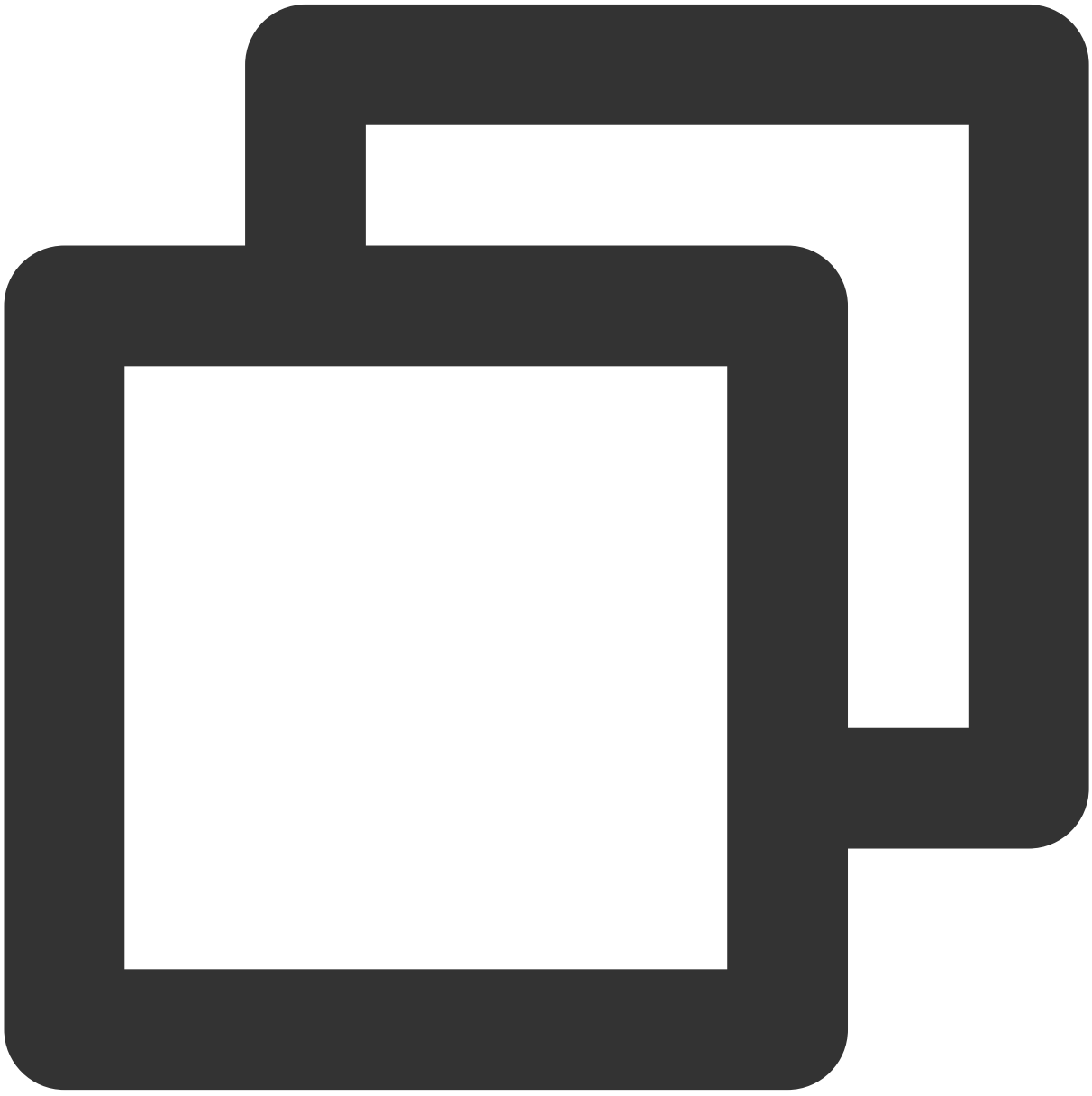
参数如下表所示：

参数	类型	含义
roomId	string	房间号

message	string	用户在其他端登录信息

onUserSigExpired

userSig 过期事件

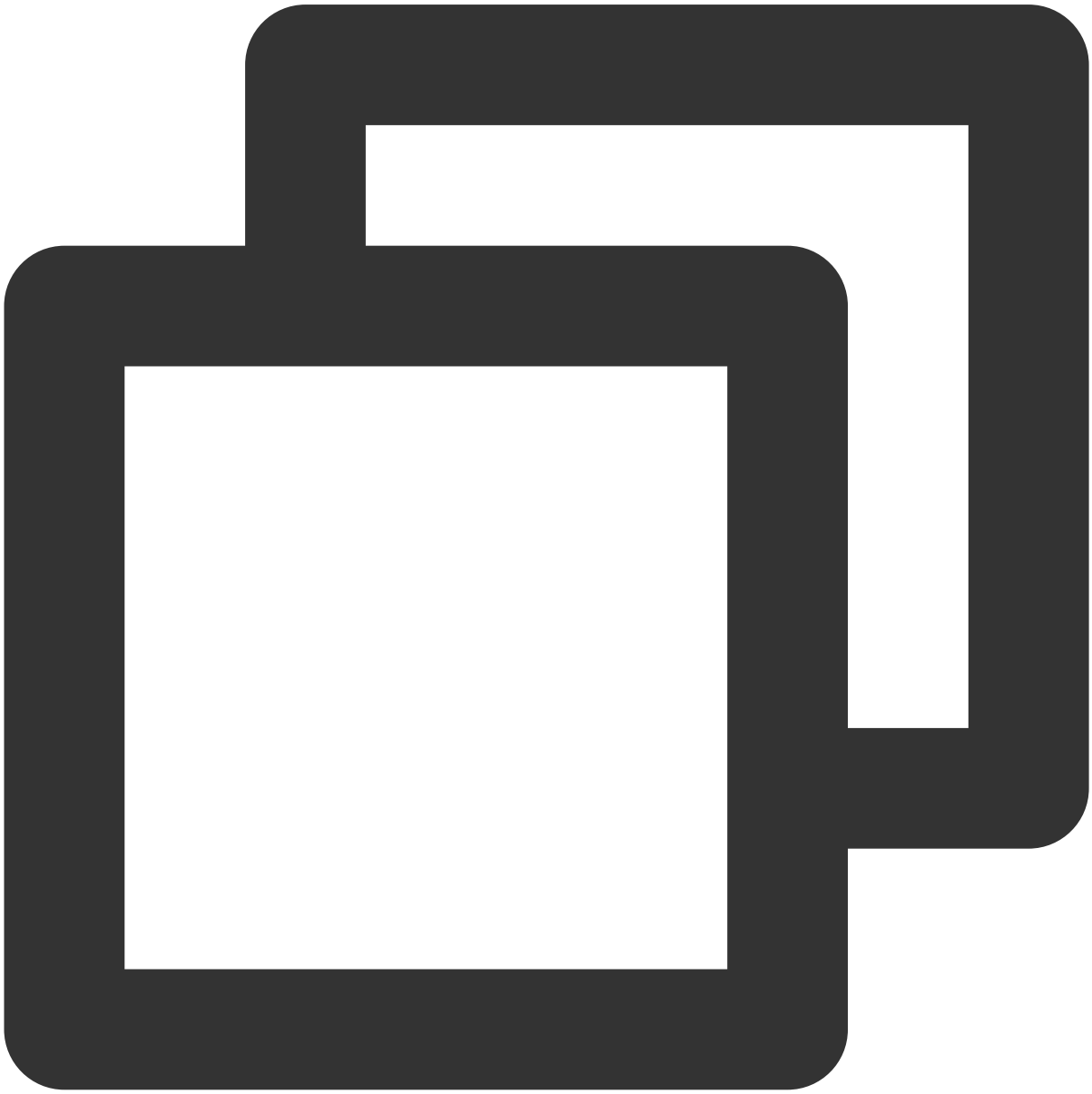


```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserSigExpired, () => {
  console.log('roomEngine.onUserSigExpired');
```

```
});
```

onRoomDismissed

主持人销毁房间事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomDismissed, ({ roomId }) => {
  console.log('roomEngine.onRoomDismissed', roomId);
```

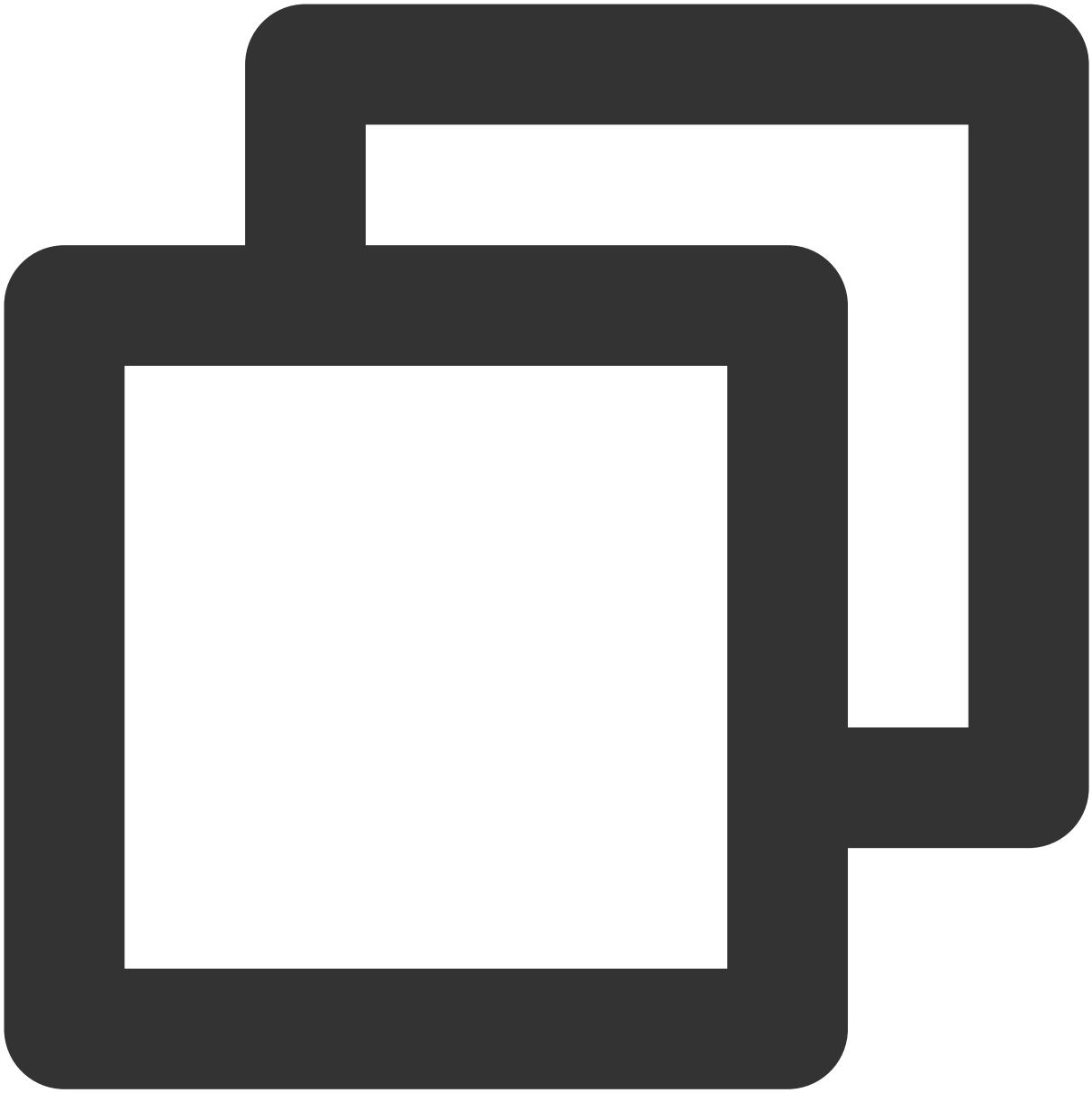
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号

onRoomNameChanged

房间名称修改事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomNameChanged, ({ roomId, roomName }) => {
  console.log('roomEngine.onRoomNameChanged', roomId, roomName);
});
```

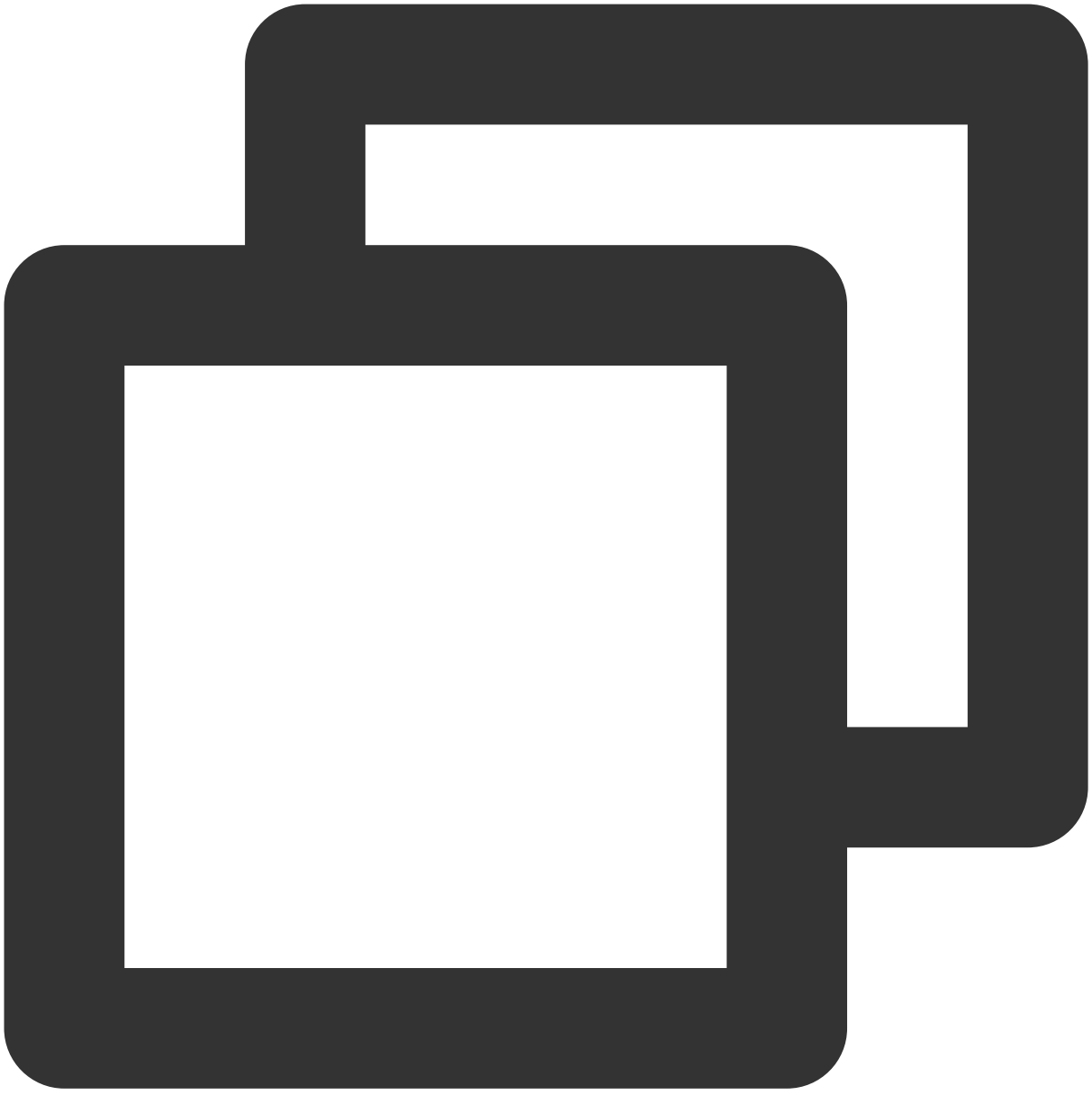
参数如下表所示：

参数	类型	含义
roomId	string	房间号

roomName	string	房间名称

onRoomSpeechModeChanged

房间名称修改事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomSpeechModeChanged, ({ roomId, speechMode }) => {
  console.log('roomEngine.onRoomSpeechModeChanged', roomId, speechMode);
```

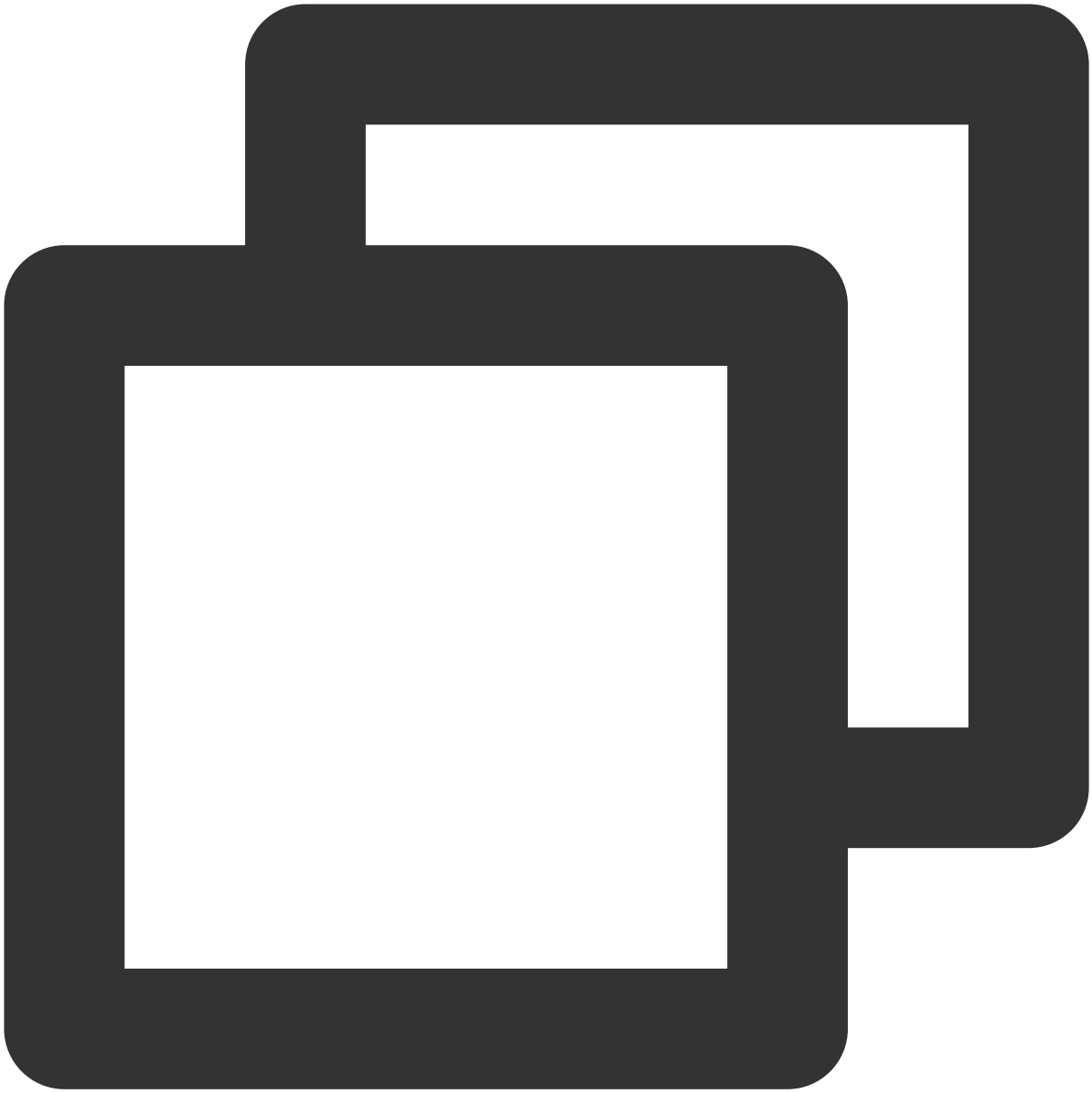
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
speechMode	TUISpeechMode	发言模式

onAllUserCameraDisableChanged

所有成员摄像头使用权限改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserCameraDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserCameraDisableChanged', isDisable);
});
```

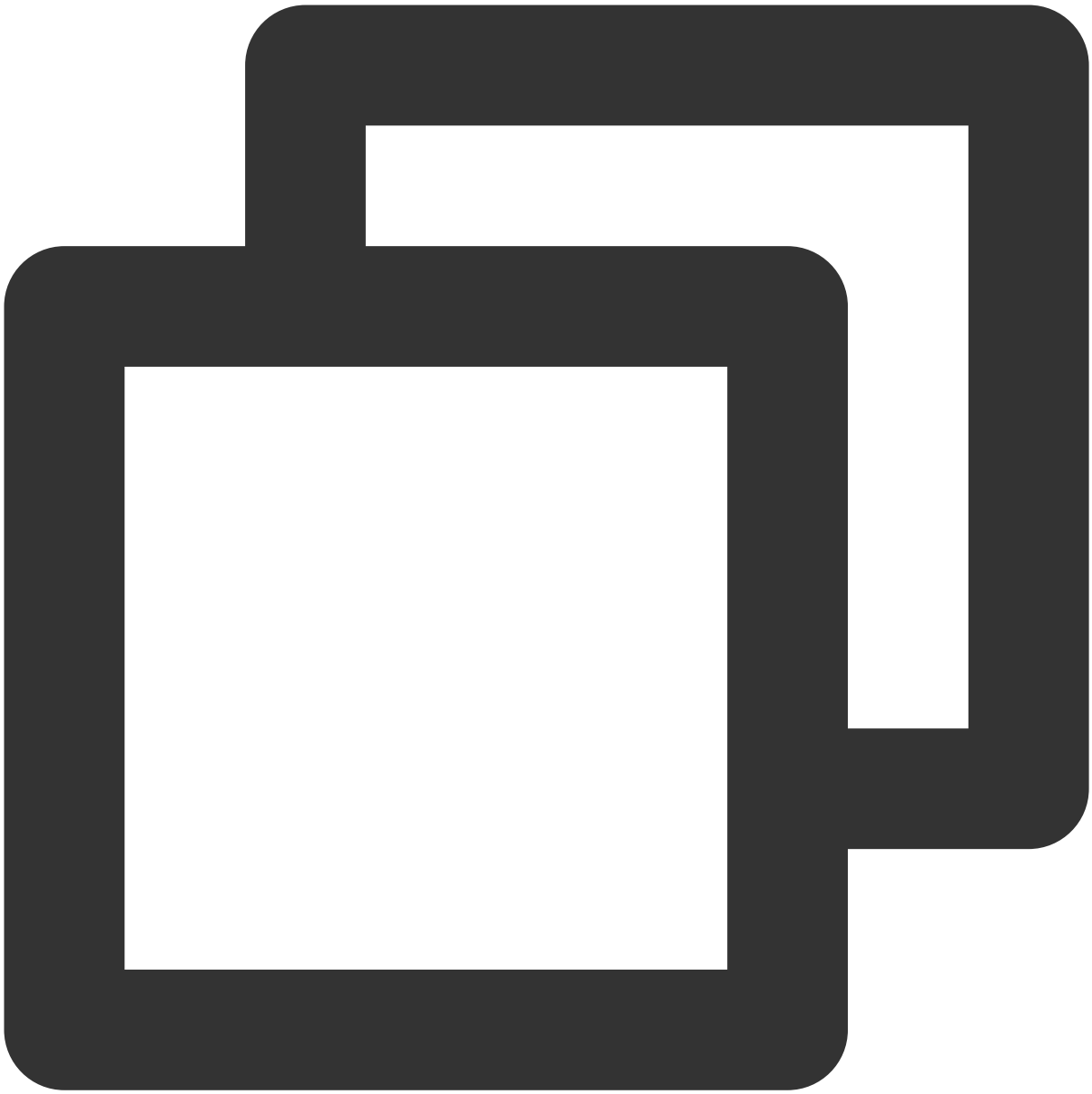
参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许使用摄像头

版权所有：腾讯云计算（北京）有限责任公司

onAllUserMicrophoneDisableChanged

所有成员麦克风使用权限改变事件



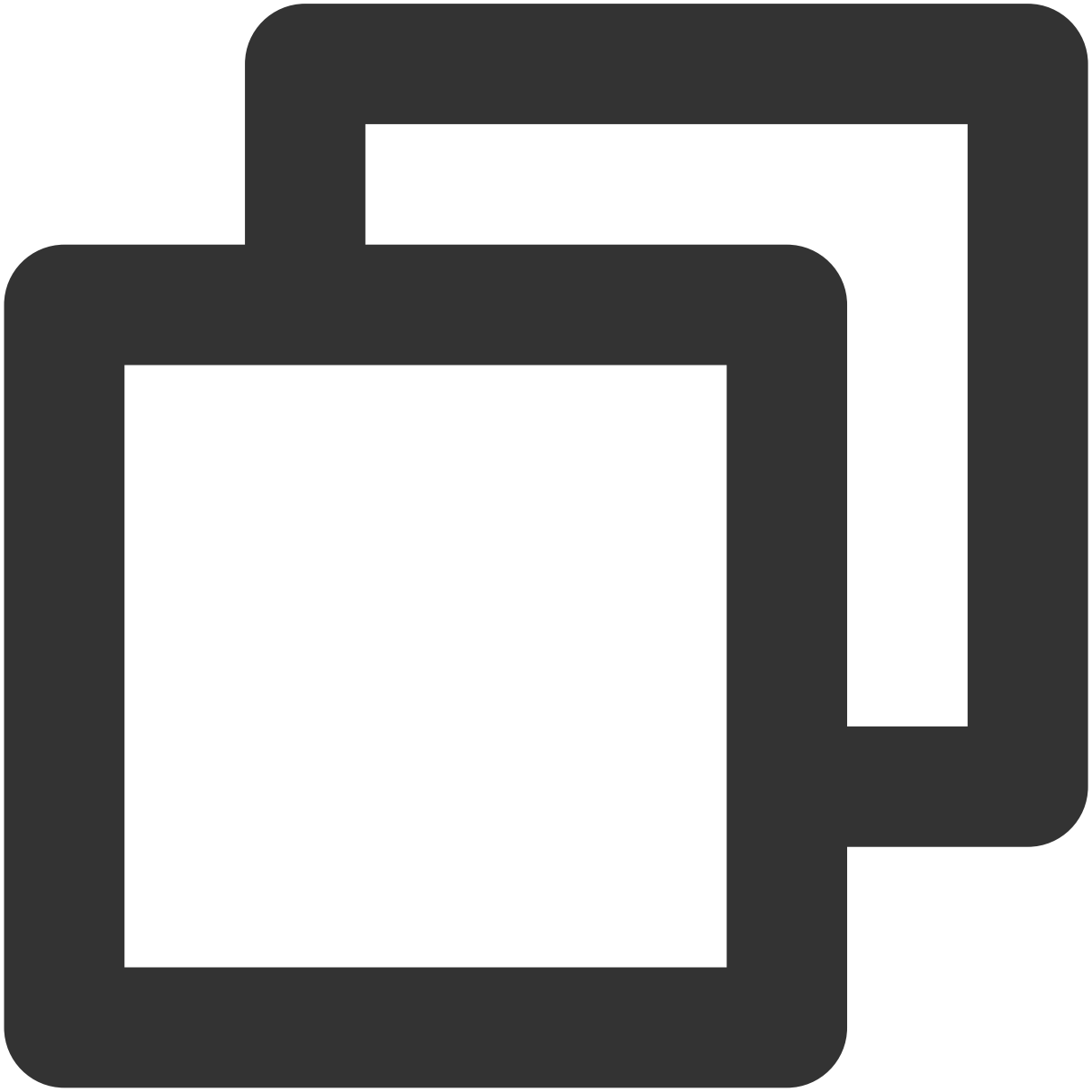
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserMicrophoneDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserMicrophoneDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许使用麦克风

onSendMessageForAllUserDisableChanged

所有成员发送消息权限改变事件



```
const roomEngine = new TUIRoomEngine();
```

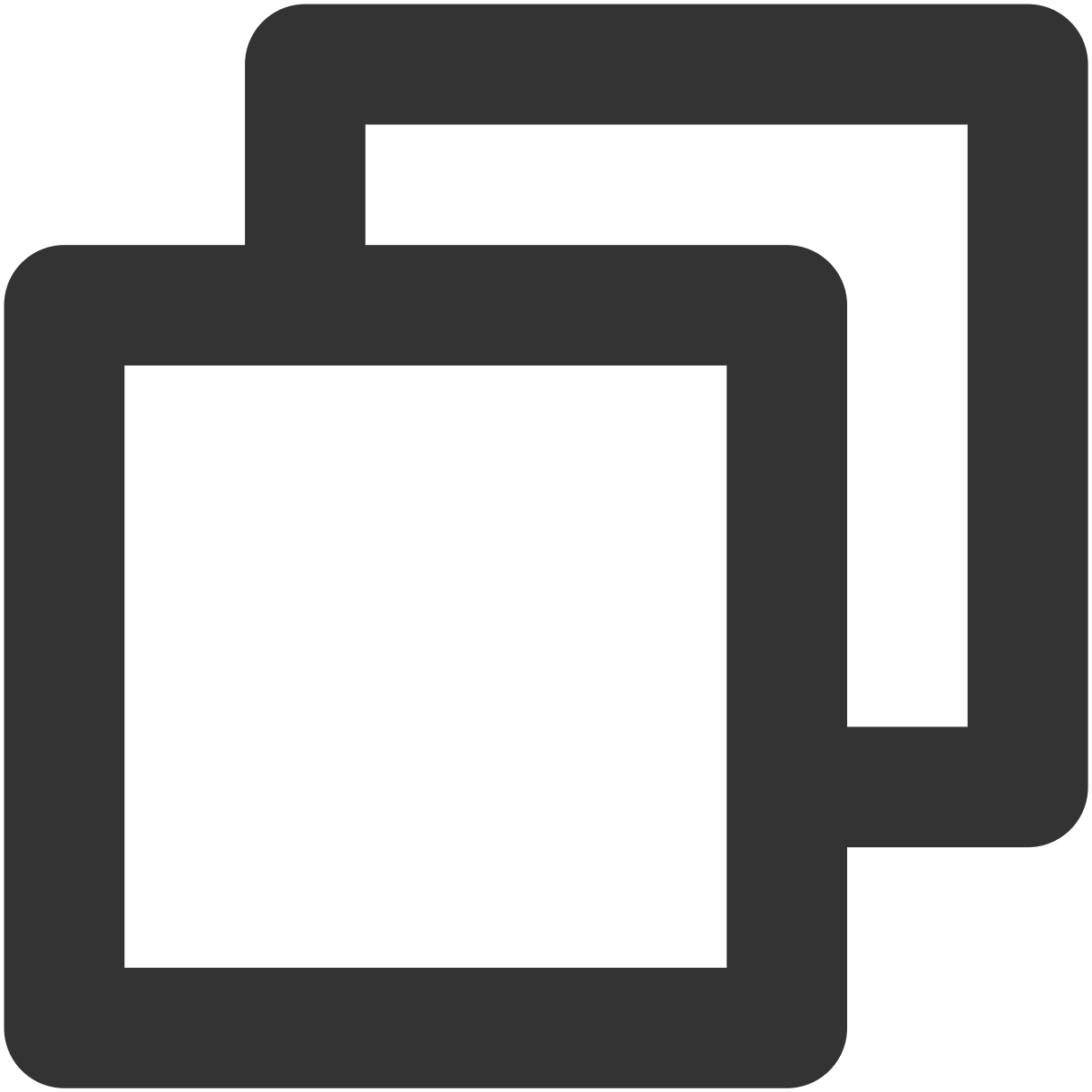
```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable }) =>
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许发送文字消息

onRoomMaxSeatCountChanged

房间最大麦位数修改事件



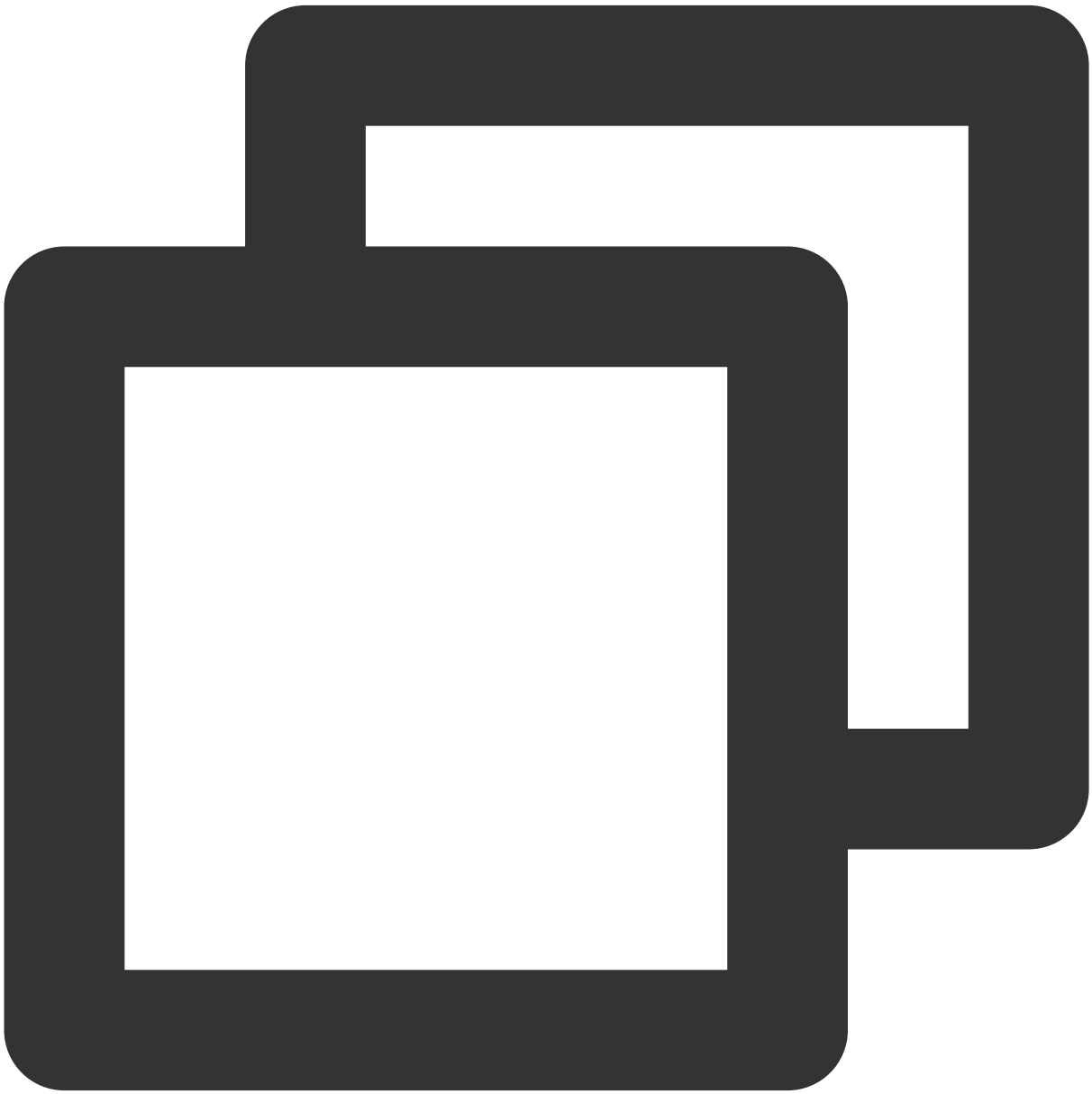
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomMaxSeatCountChanged, ({ maxSeatNumber }) => {
  console.log('roomEngine.onRoomMaxSeatCountChanged', maxSeatNumber);
});
```

参数如下表所示：

参数	类型	含义
maxSeatNumber	number	最大麦位数量

onRemoteUserEnterRoom

远端用户进入房间事件



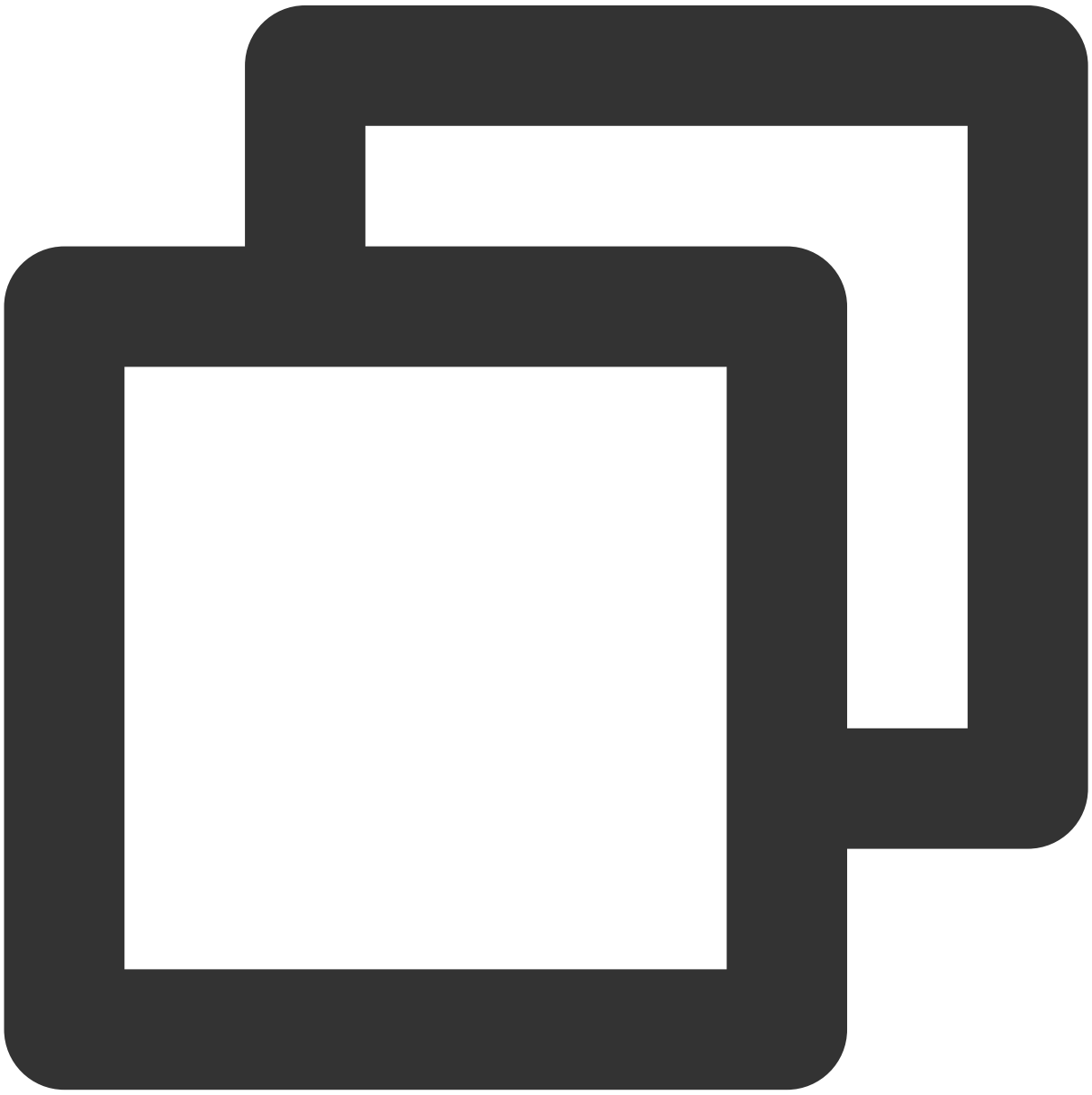
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserEnterRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserEnterRoom', roomId, userInfo);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
userInfo	TUIUserInfo	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件



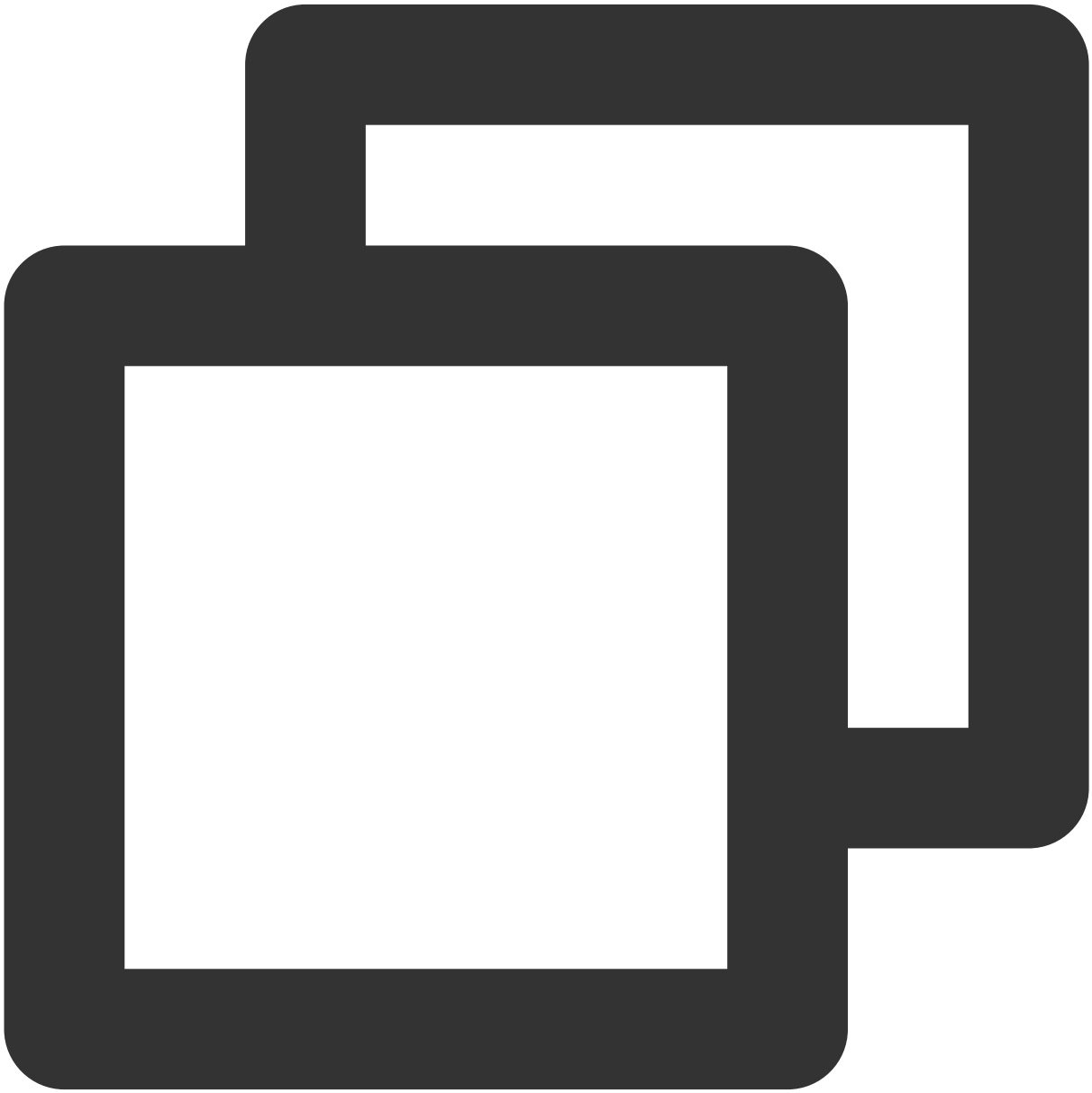
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserLeaveRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserLeaveRoom', roomId, userInfo);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
userInfo	TUIUserInfo	用户信息

onUserRoleChanged

用户角色改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserRoleChanged, ({ userId, userRole }) => {
  console.log('roomEngine.onUserRoleChanged', userId, userRole);
});
```

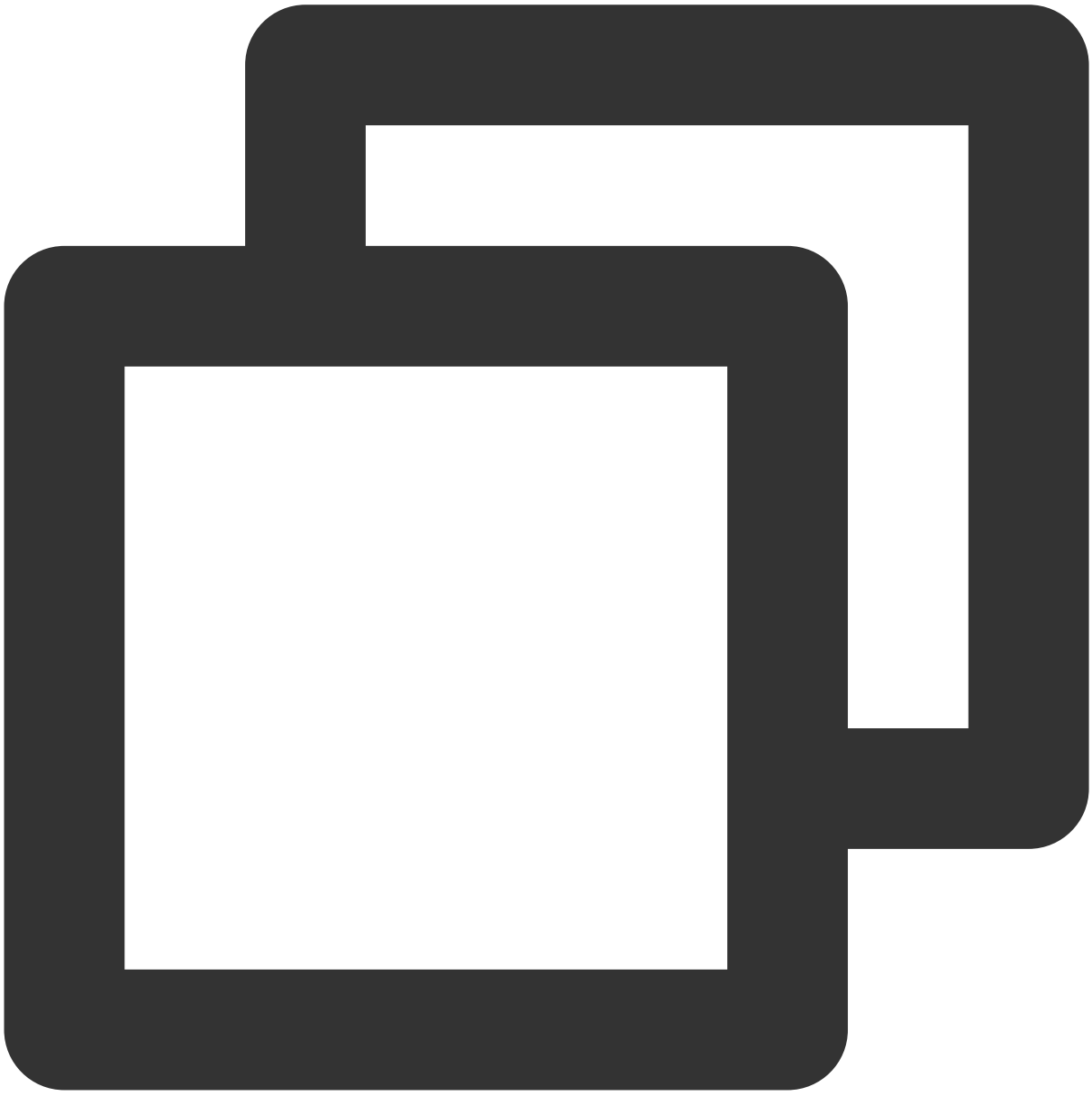
参数如下表所示：

参数	类型	含义
userId	string	用户 Id

userRole	TUIRole	用户变更后的角色

onUserVideoStateChanged

用户视频状态改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVideoStateChanged, ({ userId, streamType, hasVide
  console.log('roomEngine.onUserVideoStateChanged', userId, streamType, hasVideo, r
```

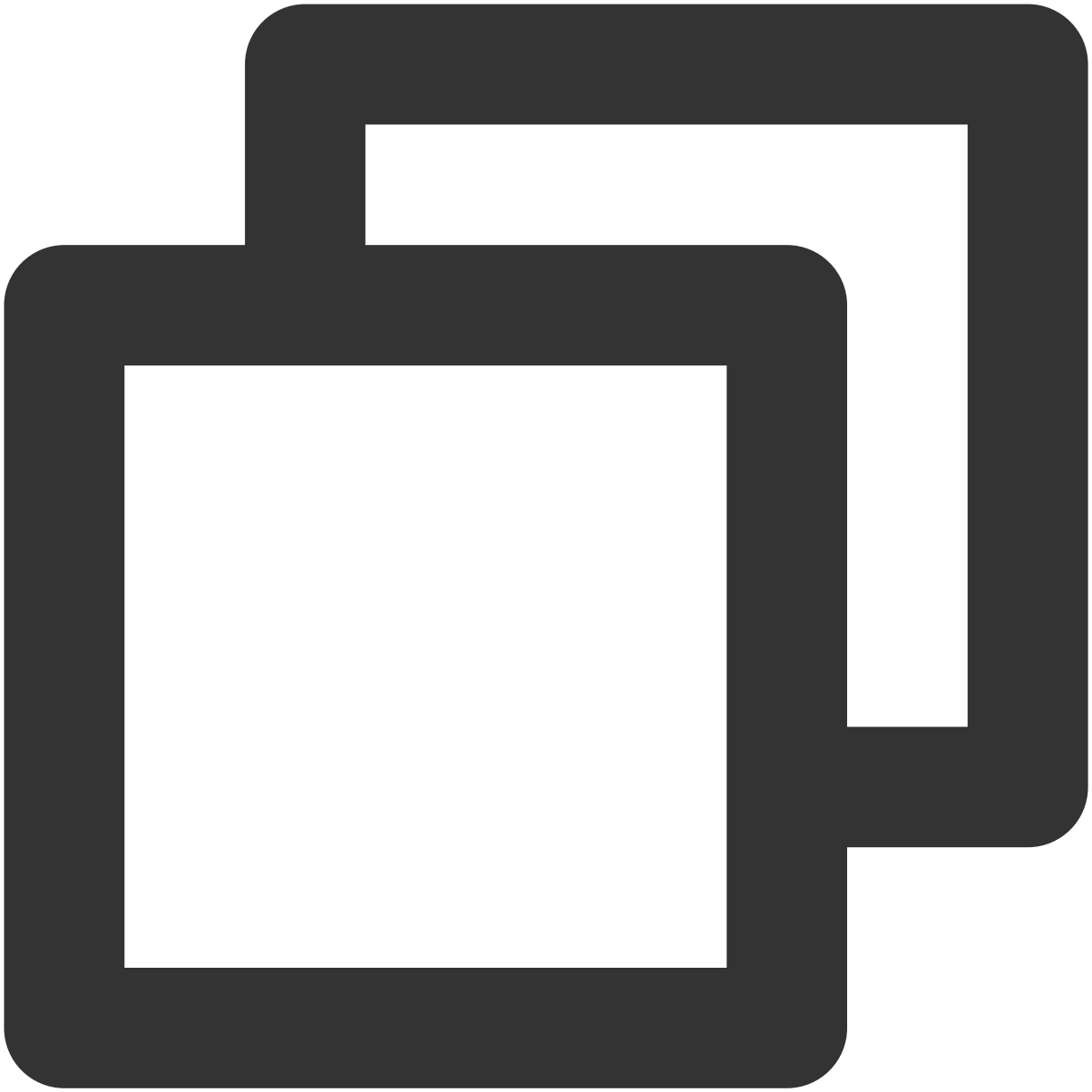
```
});
```

参数如下表所示：

参数	类型	含义
userId	string	用户Id
streamType	TUIVideoStreamType	用户流类型
hasVideo	boolean	是否有视频流
reason	TUIChangeReason	变更原因, 自己操作/主持人操作

onUserAudioStateChanged

用户音频状态改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserAudioStateChanged, ({ userId, hasAudio, reason }) =>
  console.log('roomEngine.onUserAudioStateChanged', userId, hasAudio, reason);
});
```

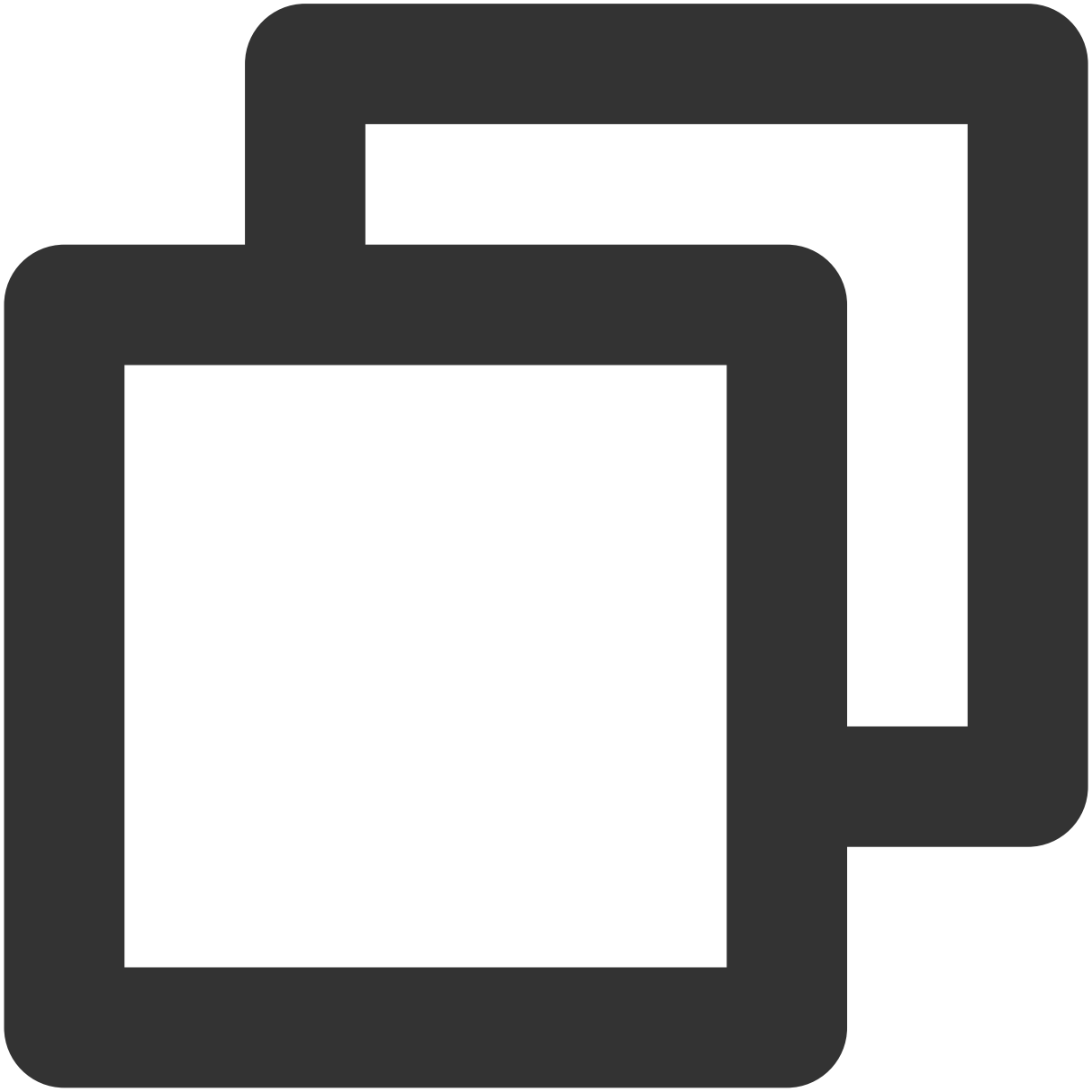
参数如下表所示：

参数	类型	含义
userId	string	用户Id

hasVideo	boolean	是否有音频流
reason	TUIChangeReason	变更原因，自己操作/主持人操作

onSendMessageForUserDisableChanged

有成员摄像头使用权限修改事件



```
const roomEngine = new TUIRoomEngine();
```

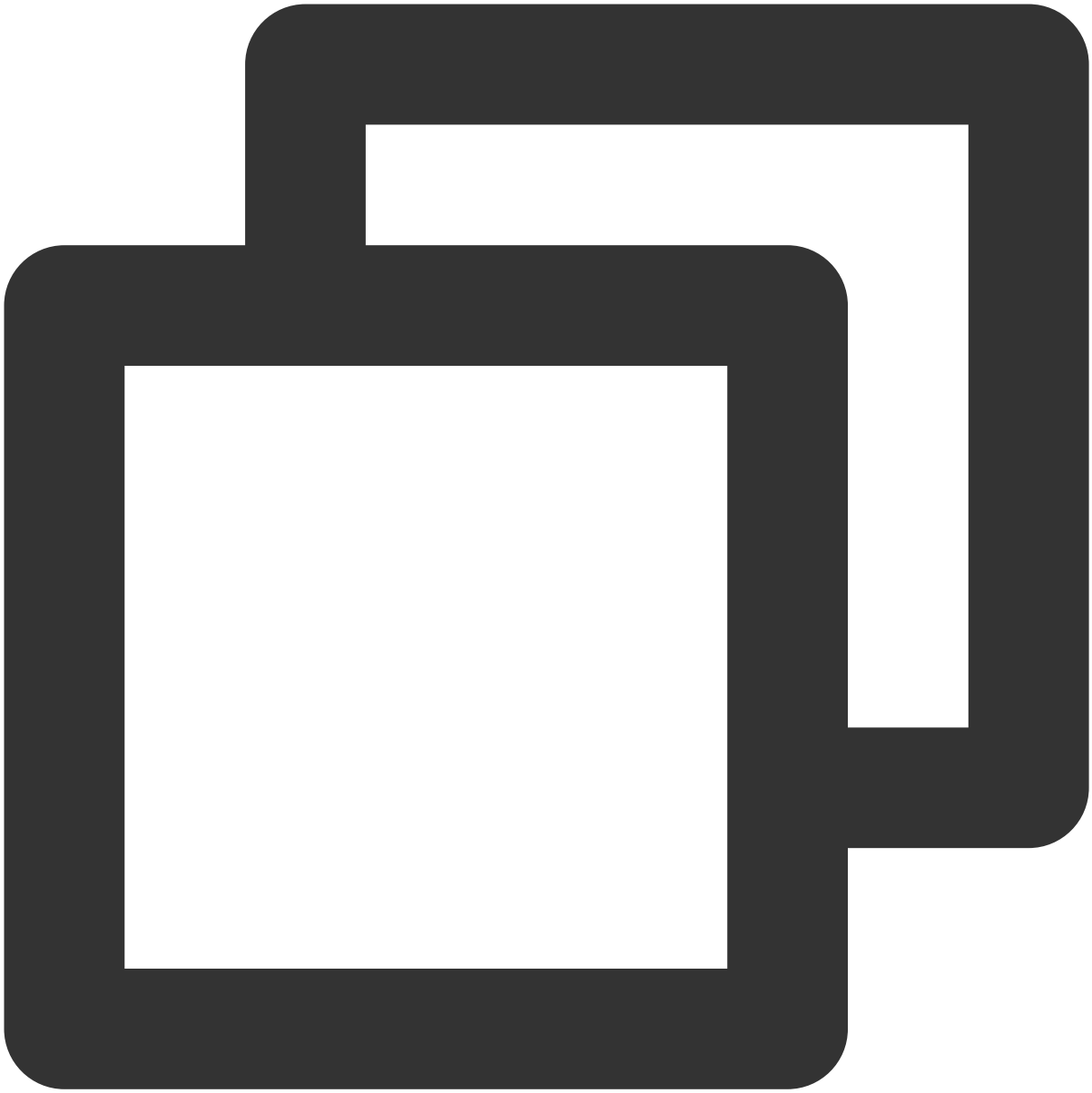
```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable }) =>
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许发送文字消息

onUserVoiceVolumeChanged

用户音量改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVoiceVolumeChanged, ({ userVolumeList }) => {
  userVolumeList.forEach(userVolume => {
    console.log('roomEngine.onUserVoiceVolumeChanged', userVolume.userId, userVolum
  })
});
```

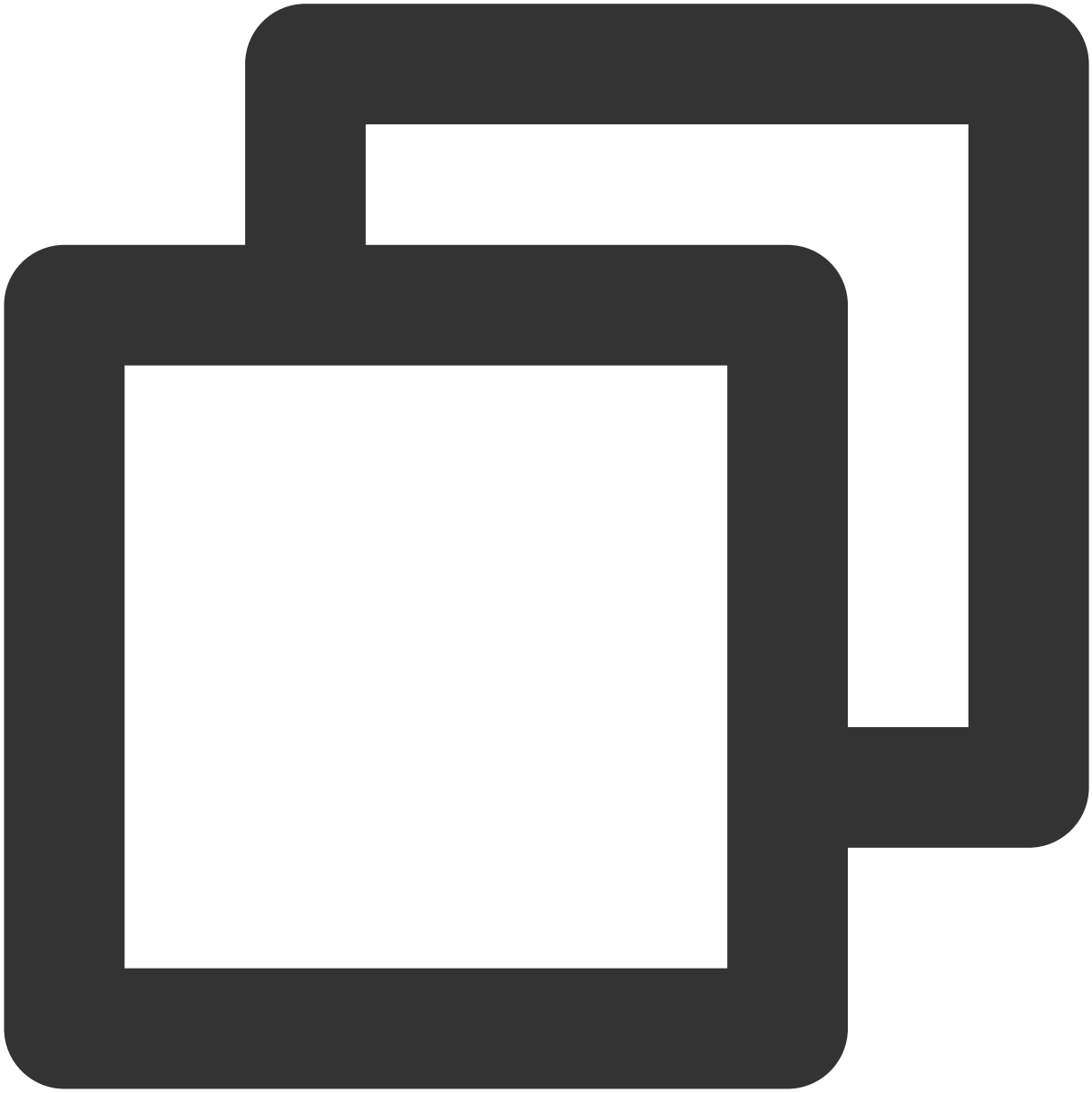
参数如下表所示：

参数	类型	含义

userVolumeList	Array.<object>	房间内所有用户的音量, 包含 userId 及 volume 信息，volume 区间为 1～100

onUserNetworkQualityChanged

用户网络质量改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserNetworkQualityChanged, ({ userNetworkList }) => {
```

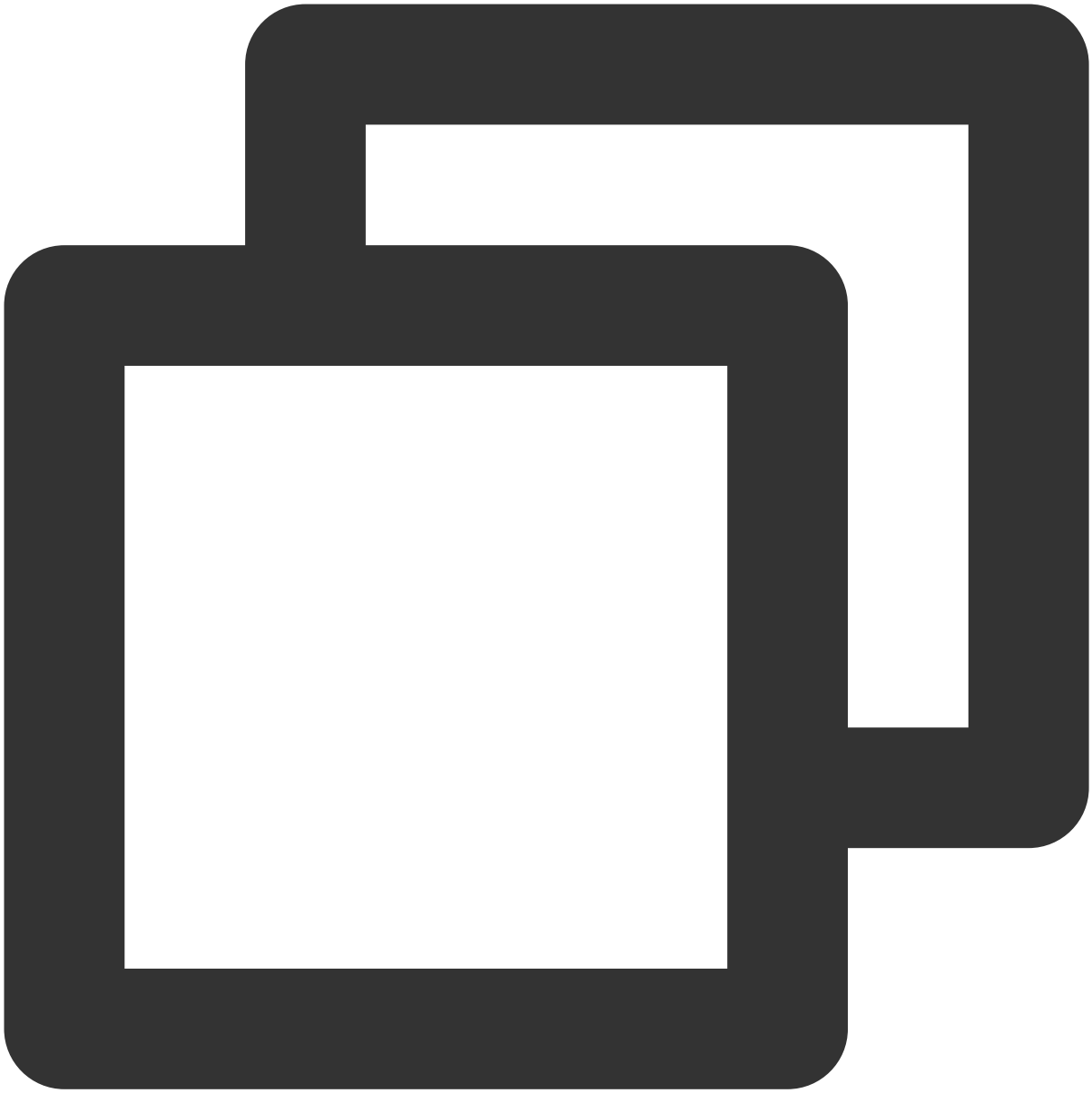
```
    userNetworkList.forEach(userNetwork => {
      console.log('roomEngine.onUserNetworkQualityChanged', userNetwork.userId, userN
    })
  });
```

参数如下表所示：

参数	类型	含义
networkMap	TUINetworkQuality	遍历网络质量等级

onSeatListChanged

麦位列表改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSeatListChanged, ({ seatList, seatedList, leftList }) =>
  console.log('roomEngine.onSeatListChanged',seatList, seatedList, leftList);
});
```

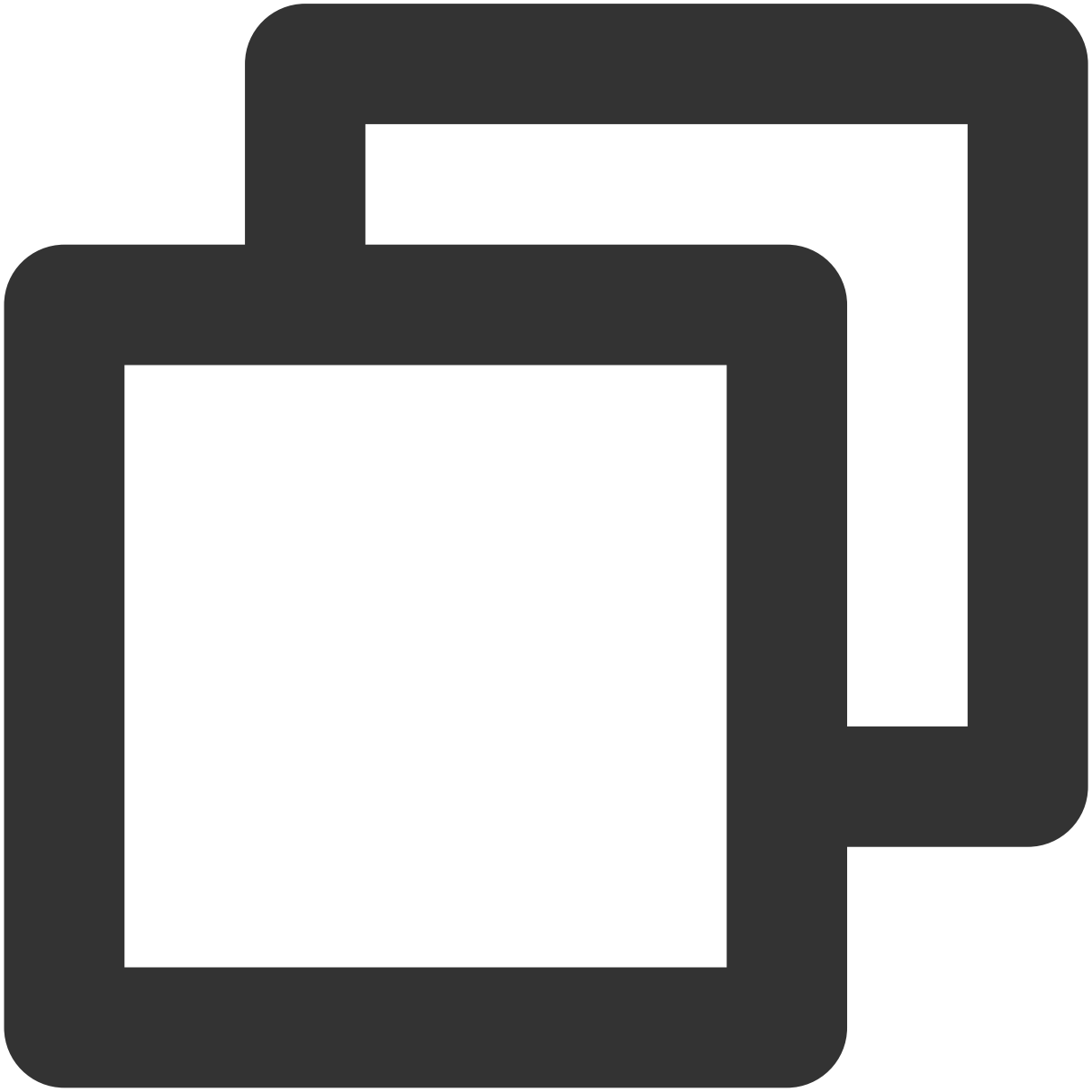
参数如下表所示：

参数	类型	含义
seatList	Array.<TUISeatInfo>	麦位列表

seatedList	Array.<TUISeatInfo>	新增麦位信息
leftList	Array.<TUISeatInfo>	离开的麦位信息

onKickedOffSeat

麦位列表改变事件



```
const roomEngine = new TUIRoomEngine();
```

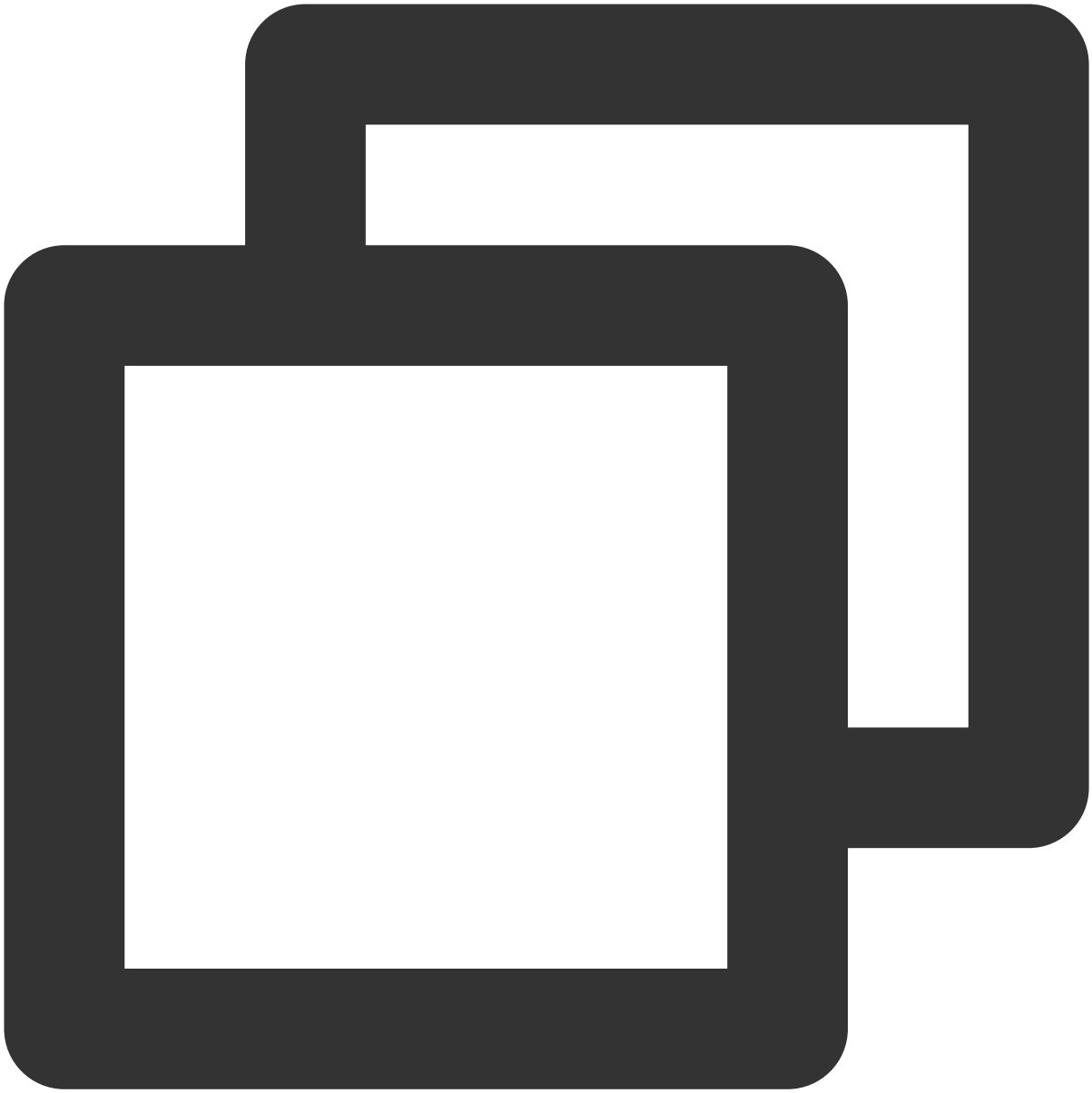
```
roomEngine.on(TUIRoomEvents.onKickedOffSeat, ({ userId }) => {
  console.log('roomEngine.onKickedOffSeat', userId);
});
```

参数如下表所示：

参数	类型	含义
userId	String	被踢下麦的用户 Id

onRequestReceived

错误事件



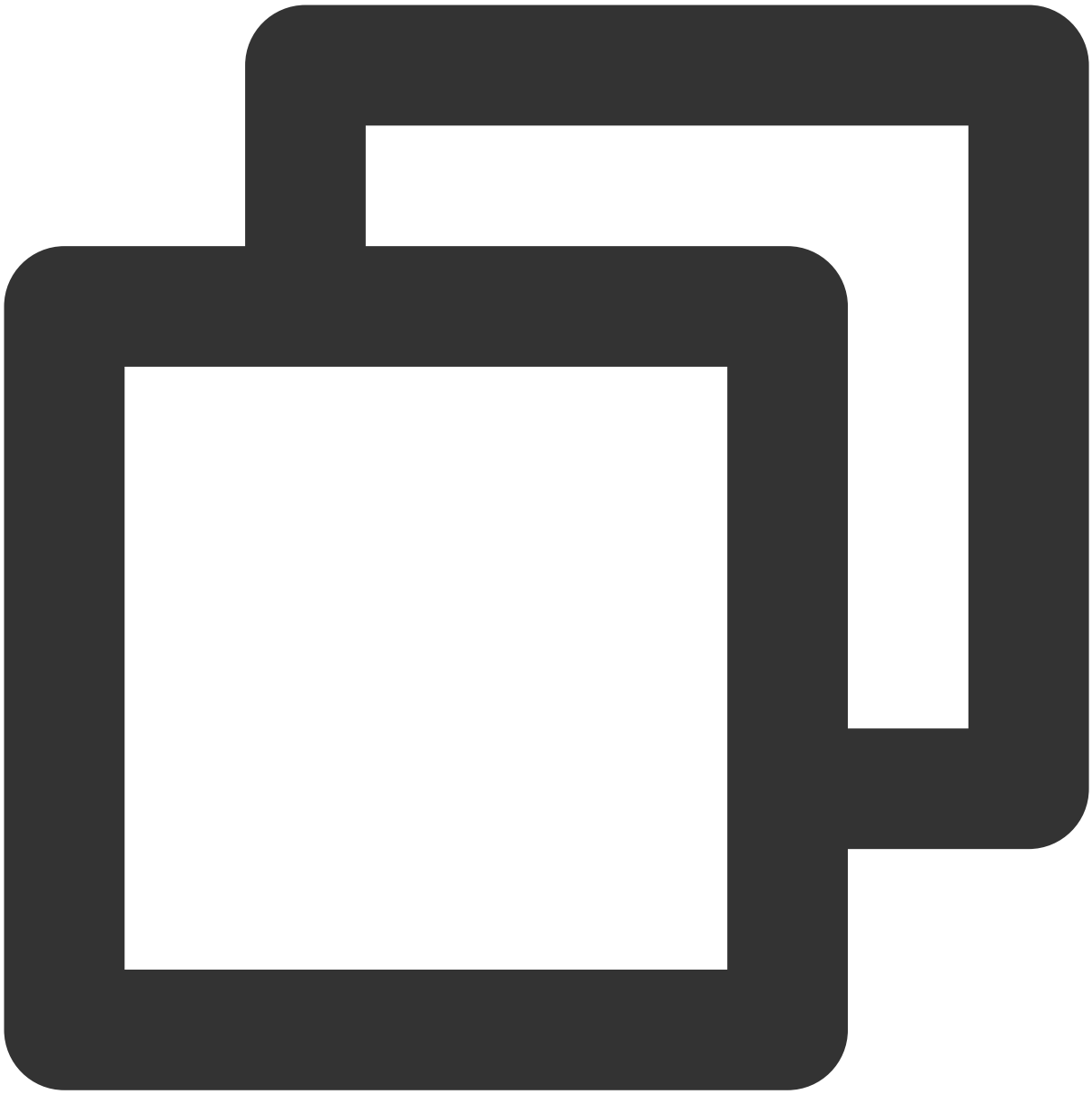
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestReceived, ({ request }) => {
  console.log('roomEngine.onRequestReceived', request);
});
```

参数如下表所示：

参数	类型	含义
request	TUIRequest	请求接收

onRequestCancelled

请求取消事件



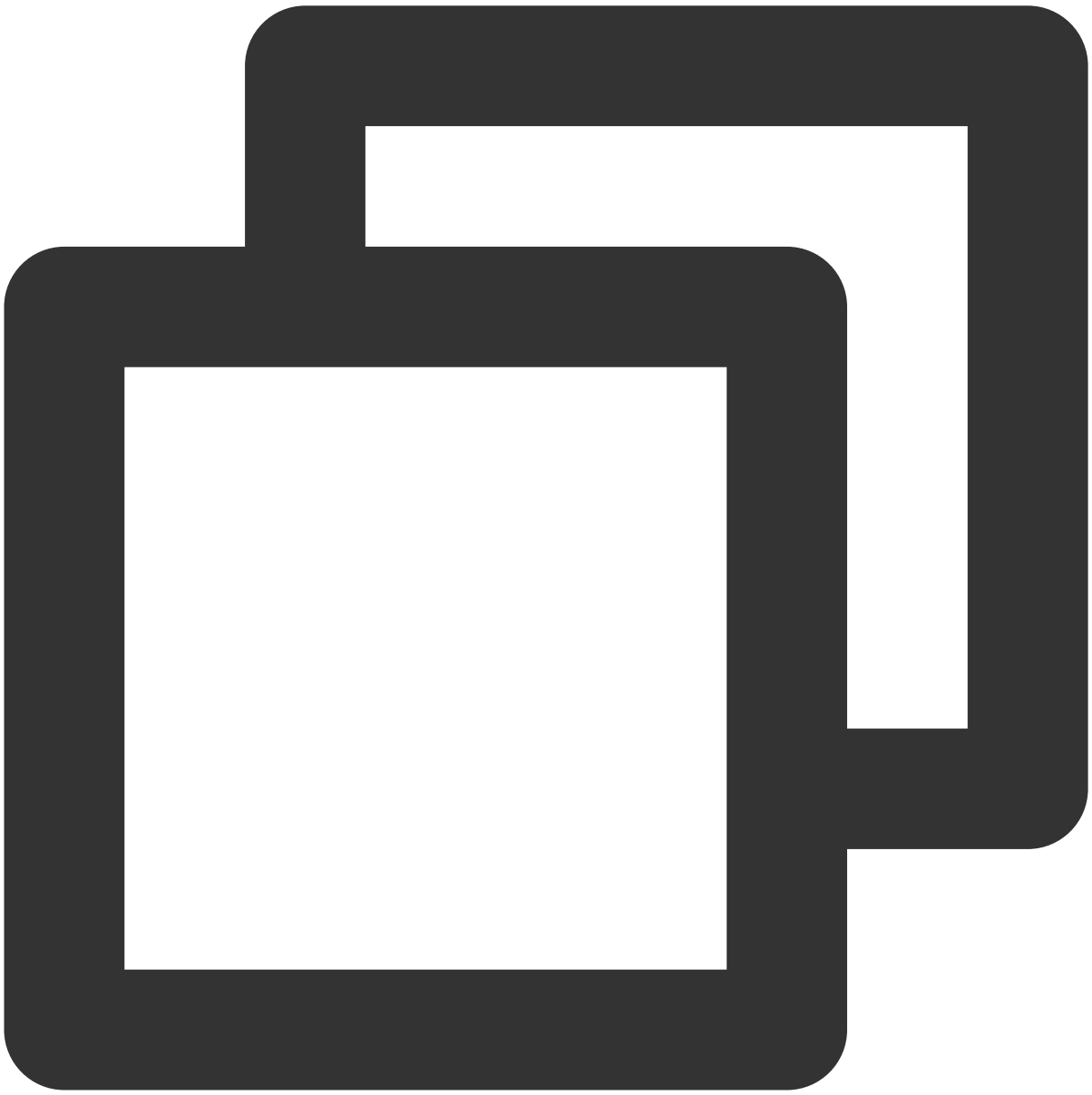
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestCancelled, ({ requestId, userId }) => {
  console.log('roomEngine.onRequestCancelled', requestId, userId);
});
```

参数如下表所示：

参数	类型	含义
requestId	string	请求 Id
userId	string	取消请求的用户 Id

onReceiveTextMessage

接收文本消息事件



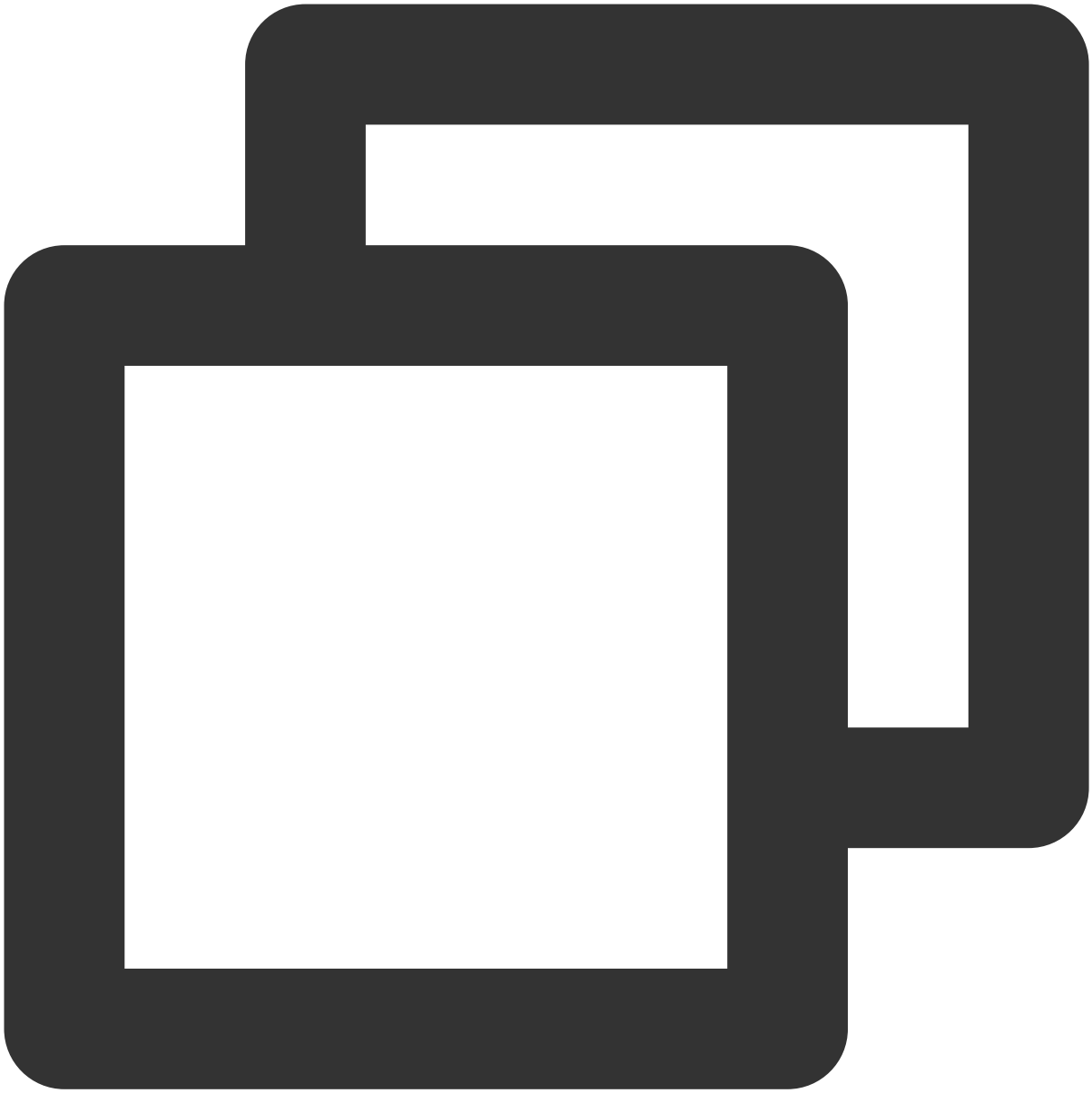
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveTextMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveTextMessage', roomId, message);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间 Id
message	TUIMessage	接收文本消息

onReceiveCustomMessage

接收自定义消息事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveCustomMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveCustomMessage', roomId, message);
});
```

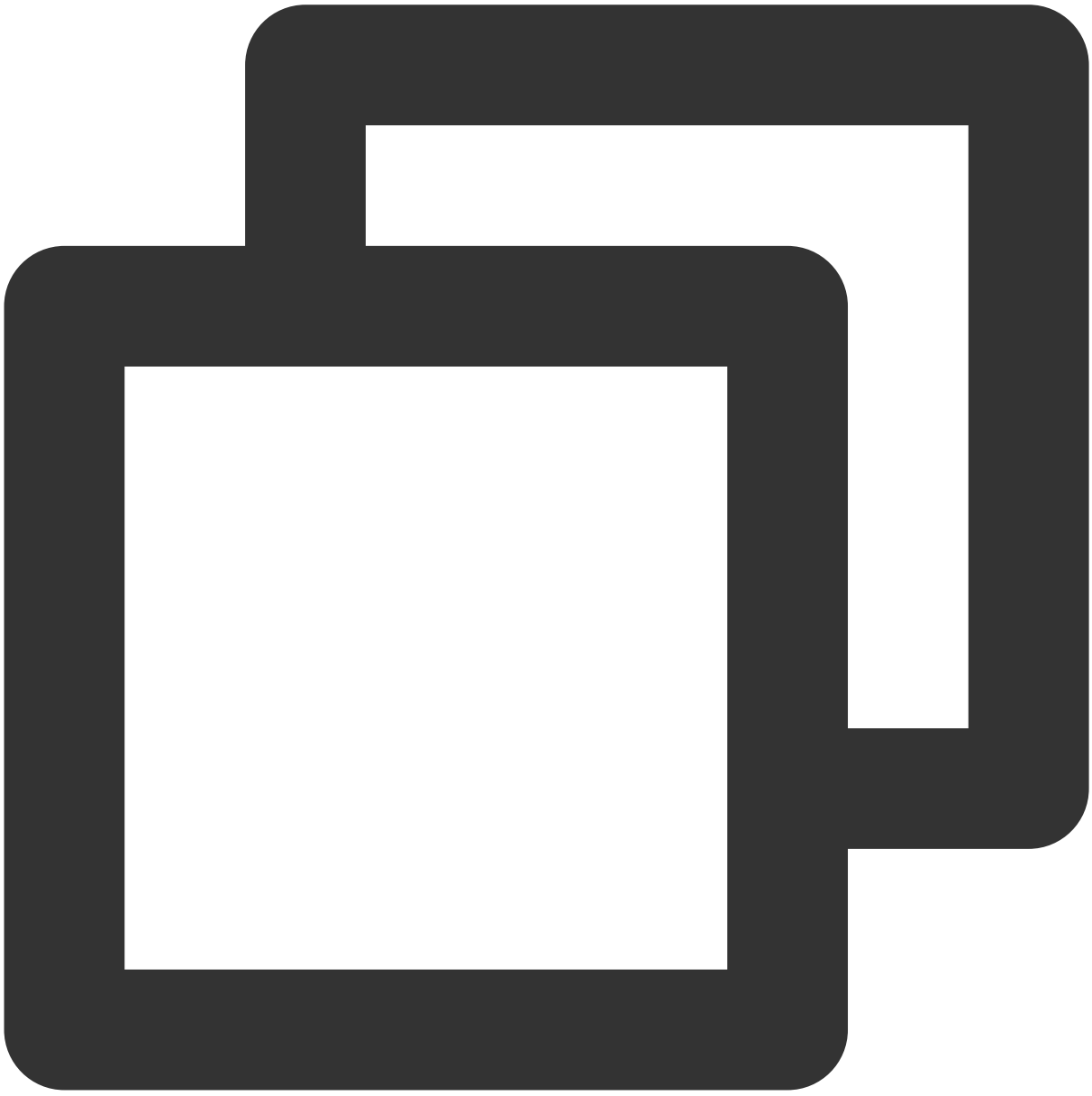
参数如下表所示：

参数	类型	含义
roomId	string	房间 Id

message	TUIMessage	接收文本消息

onDeviceChange

设备变更事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onDeviceChange, ({ deviceId, type, state }) => {
  console.log('roomEngine.onDeviceChange', deviceId, type, state);
```

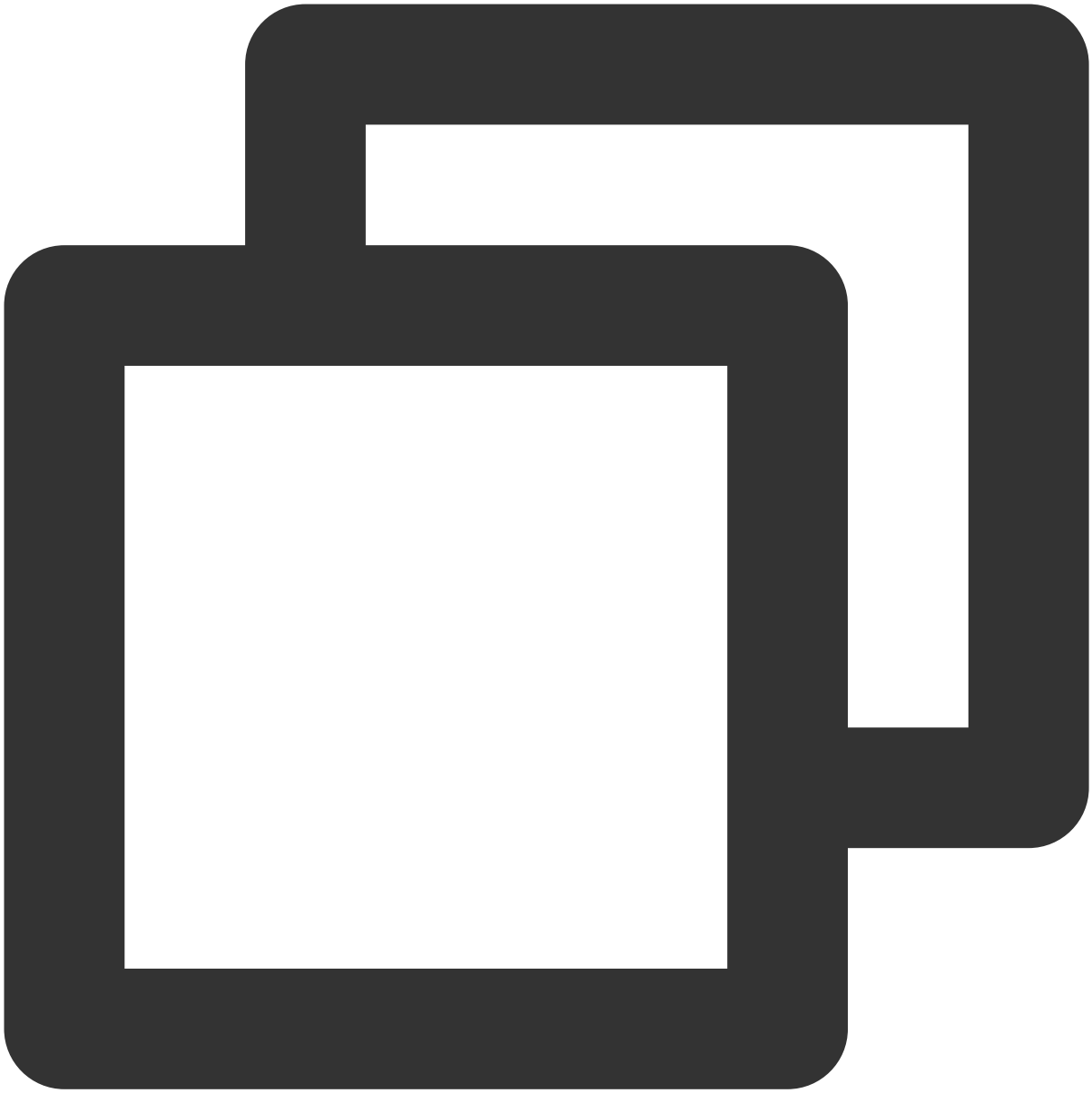
```
});
```

参数如下表所示：

参数	类型	含义
deviceId	string	设备 Id
type	TRTCDeviceType	设备类型
state	TRTCDeviceState	设备变更状态

onUserScreenCaptureStopped

屏幕分享停止事件，当用户使用浏览器自带的**停止分享**按钮结束屏幕共享时，该用户会收到 `'onUserScreenCaptureStopped'` 事件用来修改屏幕共享状态。



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserScreenCaptureStopped, () => {
  console.log('roomEngine.onUserScreenCaptureStopped', deviceId, type, state);
});
```

TUIRoomEngine Defines

最近更新时间：2023-11-14 17:04:52

本文介绍 TUIRoomEnigne web 端关键类型定义。

枚举值

TUIRole

用户角色，TUIRoomEngine 共提供三种用户角色，分别是主持人，管理员，普通用户。

字段	类型	描述
kRoomOwner	number	主持人角色
kAdministrator	number	管理员角色
kGeneralUser	number	普通用户角色

TUIVideoProfile

视频分辨率

字段	类型	描述
kLowDefinition	number	低清
kStandardDefinition	number	标清
kHighDefinition	number	高清
kSuperDefinition	number	超清

TUIAudioProfile

音频分辨率

字段	类型	描述
kAudioProfileSpeech	number	语音模式
kAudioProfileDefault	number	标准模式（默认模式）
kAudioProfileMusic	number	音乐模式

版权所有：腾讯云计算（北京）有限责任公司

TUIVideoStreamType

视频流类型

字段	类型	描述
kCameraStream	number	摄像头视频流
kScreenStream	number	屏幕分享视频流
kCameraStreamLow	number	低清摄像头视频流

TUINetworkQuality

网络状态

字段	类型	描述
kQualityUnknown	number	网络状况未知
kQualityExcellent	number	网络状况极佳
kQualityGood	number	网络状况较好
kQualityPoor	number	网络状况一般
kQualityBad	number	网络状况差
kQualityVeryBad	number	网络状况极差
kQualityDown	number	网络连接已断开

TUIRoomType

房间类型

字段	类型	描述
kGroup	number	Group 类型房间，适用于会议，教育场景，该房间中麦位是无序的，且没有数量限制
kOpen	number	Open 类型房间，适用于直播场景，该房间中麦位是有序的，且有数量限制

TUISpeechMode

发言类型

字段	类型	描述

kFreeToSpeak	number	自由发言模式
kApplyToSpeak	number	举手发言模式
kSpeakAfterTakingSeat	number	就坐后发言（抢麦位）

TUICaptureSourceType

屏幕分享类型

字段	类型	描述
kWindow	number	该分享目标是某一个 Windows 或 Mac 窗口 todo(只有 electron)
kScreen	number	该分享目标是整个 Windows 桌面或 Mac 桌面

TUIChangeReason

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

字段	类型	描述
kChangedBySelf	number	自己操作
kChangedByAdmin	number	房主或管理员操作

TUIRequestAction

房间类型

字段	类型	描述
kInvalidAction	number	无效操作
kRequestToOpenRemoteCamera	number	请求远端打开摄像头
kRequestToOpenRemoteMicrophone	number	请求远端打开麦克风
kRequestToConnectOtherRoom	number	请求远端夸房连麦，web 端暂不支持
kRequestToTakeSeat	number	请求上麦
kRequestRemoteUserOnSeat	number	请求远端上麦

TUIRequestCallbackType

请求类型

字段	类型	描述
kRequestAccepted	number	对端已接受
kRequestRejected	number	对端已拒绝
kRequestCancelled	number	请求已取消
kRequestTimeout	number	请求超时
kRequestError	number	请求错误

类型定义

TUIRoomInfo

房间信息，用户可使用 roomEngine.getRoomInfo 获取房间信息。

名称	类型	描述
roomId	string	房间号，字符串类型房间号
roomType	TUIRoomType	房间类型
owner	string	房间主持人的 userId
name	string	房间名称
createTime	string	房间创建时间
roomMemberCount	number	当前房间总人数
maxSeatCount	number	房间最大麦位数量
enableVideo	boolean	是否允许加入用户打开音频
enableAudio	boolean	是否允许加入用户打开视频
enableMessage	boolean	是否允许加入用户发送消息
enableSeatControl	boolean	是否开启麦位控制

TUIUserInfo

用户信息

名称	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

userId	string	用户 Id
userName	string	用户名称
avatarUrl	string	用户头像地址
userRole	TUIRole	用户角色
hasAudioStream	boolean	是否有音频流
hasVideoStream	boolean	是否有视频流
hasScreenStream	boolean	是否有屏幕分享流

TUIMessage

消息信息

名称	类型	描述
messageId	string	消息 Id
message	string	消息
timestamp	number	时间戳信息, 精确到秒
userId	string	用户 Id
userName	string	用户名称
avatarUrl	string	用户头像地址

TUIRequest

请求信息

名称	类型	描述
requestAction	TUIRequestAction	请求类型
timestamp	number	请求发起时间
requestId	string	请求 Id v1.0.2 及其以上版本 requestId 类型为 string； v1.0.0 及 v1.0.1 版本 requestId 类型为 number；
userId	string	发起请求的用户 Id

content	string	其他内容

TUIRequestCallback

请求回调信息

名称	类型	描述
requestCallbackType	TUIRequestCallbackType	请求回调类型，接受/拒绝/取消/超时/错误
requestId	string	请求 Id v1.0.2 及其以上版本 requestId 类型为 string； v1.0.0 及 v1.0.1 版本 requestId 类型为 number；
userId	string	用户 Id
code	number	请求响应码
message	string	请求状态补充说明

TUISeatInfo

麦位信息

名称	类型	描述
index	number	麦位序号
userId	string	麦位对应的用户 Id
locked	boolean	当前麦位是否被锁
videoMuted	boolean	当前麦位是否禁止视频
audioMuted	boolean	当前麦位是否禁止音频

版权所有：腾讯云计算（北京）有限责任公司

Electron
API 概览

最近更新时间：2024-02-29 10:59:49

TUIRoomEngine (无 UI 接口)

TUIRoomEngine 静态方法

API	描述
once	监听 TUIRoomEngine ready 事件。 **注意：TUIRoomEngine.init 以外的所有方法必须在监听到 TUIRoomEngine ready 事件且 TUIRoomEngine.init 方法执行成功后执行**
login	登录 TUIRoomEngine
setSelfInfo	设置当前用户基本信息（用户名、用户头像）
getSelfInfo	获取当前用户基本信息（用户名、用户头像）
logout	登出 TUIRoomEngine

roomEngine 房间管理 API

API	描述
createRoom	创建房间
enterRoom	进入房间
destroyRoom	销毁房间
exitRoom	离开房间
fetchRoomInfo	获取房间信息
updateRoomNameByAdmin	更新房间的名字（仅群主或者管理员可以调用）
updateRoomSpeechModeByAdmin	更新房间的发言模式（仅群主或者管理员可以调用）
getUserList	获取当前房间用户列表
getUserInfo	获取用户的详细信息

版权所有：腾讯云计算（北京）有限责任公司

roomEngine 音视频 API

API	描述
setLocalVideoView	设置本地流的渲染位置
openLocalCamera	本地摄像头视频流采集
closeLocalCamera	关闭本地摄像头
openLocalMicrophone	打开本地麦克风
closeLocalMicrophone	关闭本地麦克风
updateVideoQuality	设置本地视频的参数
updateAudioQuality	设置本地音频的参数
startPushLocalVideo	开始向远端推本地视频流
stopPushLocalVideo	停止向远端推本地视频流
startPushLocalAudio	开始向远端推本地音频流
stopPushLocalAudio	停止向远端推本地音频流
setRemoteVideoView	设置远端流渲染的区域
startPlayRemoteVideo	开始播放远端用户视频流
stopPlayRemoteVideo	停止播放远端用户视频流
muteRemoteAudioStream	停止远端用户的音频流

roomEngine 成员管理 API

API	描述
openRemoteDeviceByAdmin	请求远端用户打开媒体设备
applyToAdminToOpenLocalDevice	参会者向主持人申请打开设备
closeRemoteDeviceByAdmin	关闭远端用户媒体设备
cancelRequest	取消已经发出的请求
responseRemoteRequest	回复远端用户的请求
changeUserRole	改变用户的角色

版权所有：腾讯云计算（北京）有限责任公司

kickRemoteUserOutOfRoom	将用户踢出房间
disableDeviceForAllUserByAdmin	禁用/打开所有用户的媒体设备
disableSendingMessageForAllUser	禁止/允许所有用户发送消息
disableSendingMessageByAdmin	禁止/允许某个用户发送消息

roomEngine 屏幕分享 API

API	描述
startScreenSharingElectron	开始屏幕共享
stopScreenSharingElectron	停止屏幕共享
getScreenSharingTarget	获取屏幕分享列表
selectScreenSharingTarget	切换屏幕分享窗口

roomEngine 麦位管理API

API	描述
setMaxSeatCount	设置房间麦位最大值
getSeatList	获取麦位信息
takeSeat	获取麦位
leaveSeat	释放麦位
takeUserOnSeatByAdmin	邀请其他人上麦（仅房间主持人和管理员可调用此方法）
kickUserOffSeatByAdmin	将其他人踢下麦（仅房间主持人和管理员可调用此方法）
lockSeatByAdmin	锁定某个麦位状态（仅房间主持人和管理员可调用此方法）

roomEngine 消息发送 API

API	描述
sendTextMessage	发送文本消息
sendCustomMessage	发送自定义消息

版权所有：腾讯云计算（北京）有限责任公司

roomEngine 设备管理 API

API	描述
getCameraDevicesList	获取摄像头设备列表
getMicDevicesList	获取麦克风设备列表
getSpeakerDevicesList	获取扬声器设备列表
setCurrentCameraDevice	设置要使用的摄像头设备
setCurrentMicDevice	设置要使用的麦克风设备
setCurrentSpeakerDevice	设置要使用的扬声器设备
getCurrentCameraDevice	获取当前正在使用的摄像头设备
getCurrentMicDevice	获取当前正在使用的麦克风设备
getCurrentSpeakerDevice	获取当前正在使用的扬声器设备
startCameraDeviceTest	开始进行摄像头测试
stopCameraDeviceTest	停止摄像头测试

roomEngine 事件监听 API

API	描述
on	监听 TUIRoomEvents 事件
off	取消监听 TUIRoomEvents 事件

roomEngine 其他 API

API	描述
getTRTCCloud	获取 trtcCloud 实例
getTIM	获取 tim 实例

事件类型定义

TUIRoomEvent 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

EVENT	描述
TUIRoomEvents.onError	错误事件
TUIRoomEvents.onKickedOutOfRoom	踢出房间事件
TUIRoomEvents. onKickedOffLine	当前用户被踢下线
TUIRoomEvents.onUserSigExpired	userSig 过期事件
TUIRoomEvents.onRoomDismissed	主持人销毁房间事件
TUIRoomEvents.onRoomNameChanged	房间名称修改事件
TUIRoomEvents.onRoomSpeechModeChanged	房间发言模式修改事件
TUIRoomEvents.onAllUserCameraDisableChanged	所有成员摄像头使用权限改变事件
TUIRoomEvents.onAllUserMicrophoneDisableChanged	所有成员麦克风使用权限改变事件
TUIRoomEvents.onSendMessageForAllUserDisableChanged	所有成员发送消息状态改变事件
TUIRoomEvents.onRoomMaxSeatCountChanged	房间最大麦位数修改事件
TUIRoomEvents.onRemoteUserEnterRoom	远端用户进入房间事件
TUIRoomEvents.onRemoteUserLeaveRoom	远端用户离开房间事件
TUIRoomEvents.onUserRoleChanged	用户角色改变事件
TUIRoomEvents.onUserVideoStateChanged	用户视频状态改变事件
TUIRoomEvents.onUserAudioStateChanged	用户音频状态改变事件
TUIRoomEvents.onSendMessageForUserDisableChanged	用户发送消息状态事件
TUIRoomEvents.onUserVoiceVolumeChanged	用户音量改变事件
TUIRoomEvents.onUserNetworkQualityChanged	用户网络质量改变事件
TUIRoomEvents.onSeatListChanged	麦位列表改变事件
TUIRoomEvents.onKickedOffSeat	用户被踢下麦事件
TUIRoomEvents.onRequestReceived	请求接收事件
TUIRoomEvents.onRequestCancelled	请求取消事件

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEvents.onReceiveTextMessage	接收文本消息事件
TUIRoomEvents.onReceiveCustomMessage	接收自定义消息事件
TUIRoomEvents.onDeviceChange	设备变更事件
TUIRoomEvents.onUserScreenCaptureStopped	屏幕分享停止事件，当用户使用浏览器自带的【停止分享】按钮结束屏幕共享时，该用户会收到 'onUserScreenCaptureStopped' 事件用来修改屏幕共享状态。

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomKit

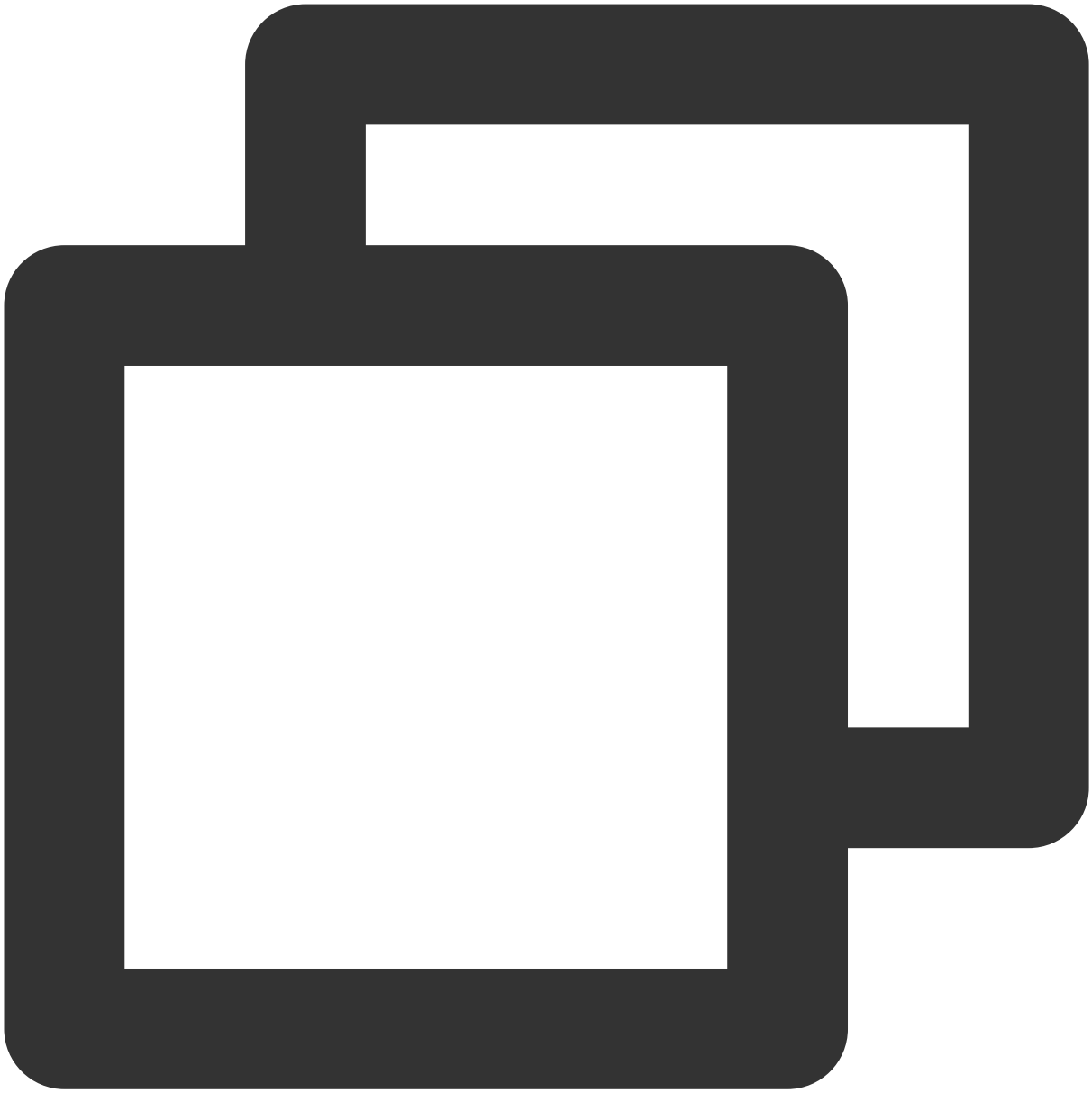
最近更新时间：2023-11-16 14:47:45

API 简介

TUIRoomkit API 是人视频会议组件的**含 UI 接口**，使用 TUIRoomkit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，如果您想要体验并调试视频会议效果，请阅读 Demo 快速跑通，如果您想把我们的功能直接嵌入到您的项目中，请阅读 快速接入(TUIRoomKit)。

API 概览

在页面中引用 TUIRoom 组件。例如：在 `App.vue` 组件中引入 TUIRoom 组件。
TUIRoom 组件将用户分为主持人角色及普通成员角色。组件对外提供了 init、createRoom、enterRoom 方法。
主持人及普通成员可通过 init 方法向 TUIRoom 组件初始化应用及用户数据，主持人可通过 createRoom 方法创建并加入房间，普通成员可通过 enterRoom 方法加入主持人已经创建好的房间。



```
<template>
    <room ref="TUIRoomRef"></room>
</template>

<script setup lang="ts">
    import { ref, onMounted } from 'vue';
    // 引入 TUIRoom 组件，注意确认引入路径是否正确
    import Room from './TUIRoom/index.vue';
    // 获取 TUIRoom 组件元素，用于调用 TUIRoom 组件的方法
    const TUIRoomRef = ref();
```

版权所有：腾讯云计算（北京）有限责任公司

```
onMounted(async () => {
    // 初始化 TUIRoom 组件
    // 主持人在创建房间前需要先初始化 TUIRoom 组件
    // 普通成员在进入房间前需要先初始化 TUIRoom 组件
    await TUIRoomRef.value.init({
        // 获取 sdkAppId 请您参考 步骤一
        sdkAppId: 0,
        // 用户在您业务中的唯一标示 Id
        userId: '',
        // 本地开发调试可在 https://console.tencentcloud.com/trtc/usersigtool 页面
        // 注意 userSig 与 userId 为一一对应关系
        userSig: '',
        // 用户在您业务中使用的昵称
        userName: '',
        // 用户在您业务中使用的头像链接
        avatarUrl: '',
    })
    // 默认执行创建房间，实际接入可按需求择机执行 handleCreateRoom 方法
    await handleCreateRoom();
})

    // 主持人创建房间，该方法只在创建房间时调用
async function handleCreateRoom() {
    // roomId 为用户进入的房间号，要求 roomId 类型为 number
    // roomMode 包含'FreeToSpeak'(自由发言模式) 和'SpeakAfterTakingSeat'(上台发言模
    // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认
    const roomId = '123456';
    const roomMode = 'FreeToSpeak';
    const roomParam = {
        isOpenCamera: true,
        isOpenMicrophone: true,
    };
    const createRoomInfo = {
        roomId,
        roomName: 'User defined room name' || roomId,
        roomMode,
        roomParam
    };
    await TUIRoomRef.value.createRoom(createRoomInfo);
}

    // 普通成员进入房间，该方法在普通成员进入已创建好的房间时调用
async function handleEnterRoom() {
    // roomId 为用户进入的房间号，要求 roomId 类型为 number
    // roomParam 指定了用户进入房间的默认行为（是否默认开启麦克风，是否默认开启摄像头，默认
    const roomId = '123456';
    const roomParam = {
```

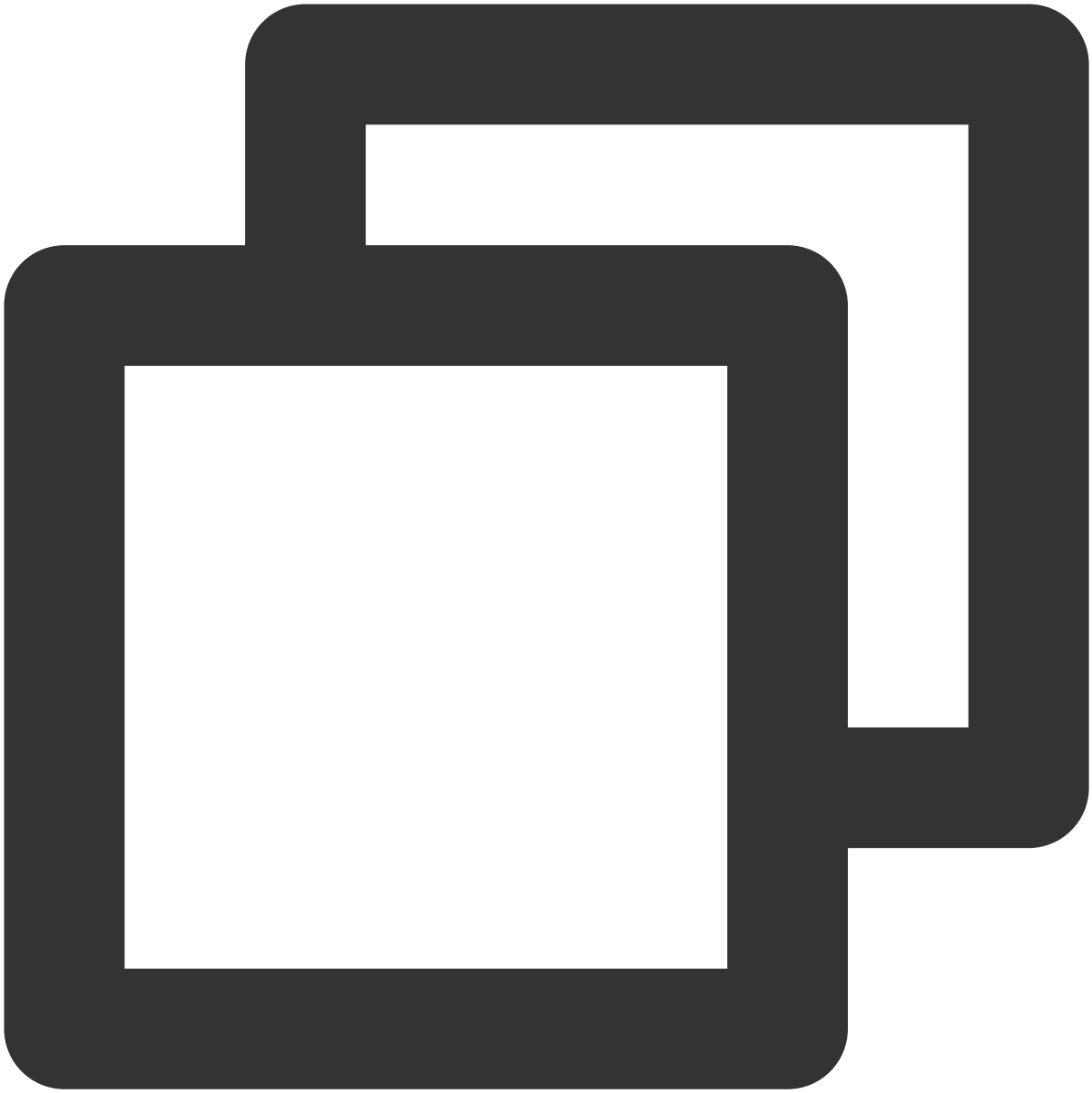
```
            isOpenCamera: true,
            isOpenMicrophone: true,
        };
        const enterRoomInfo = {
            roomId,
            roomParam
        };
        await TUIRoomRef.value.enterRoom(enterRoomInfo);
    }
</script>

<style>
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
}

#app {
    width: 100%;
    height: 100%;
}
</style>
```

API 详情

TUIRoomkit 接口

init

初始化 TUIRoom 数据，任何使用 TUIRoom 的用户都需要调用 init 方法。



```
TUIRoomRef.value.init(roomData);
```

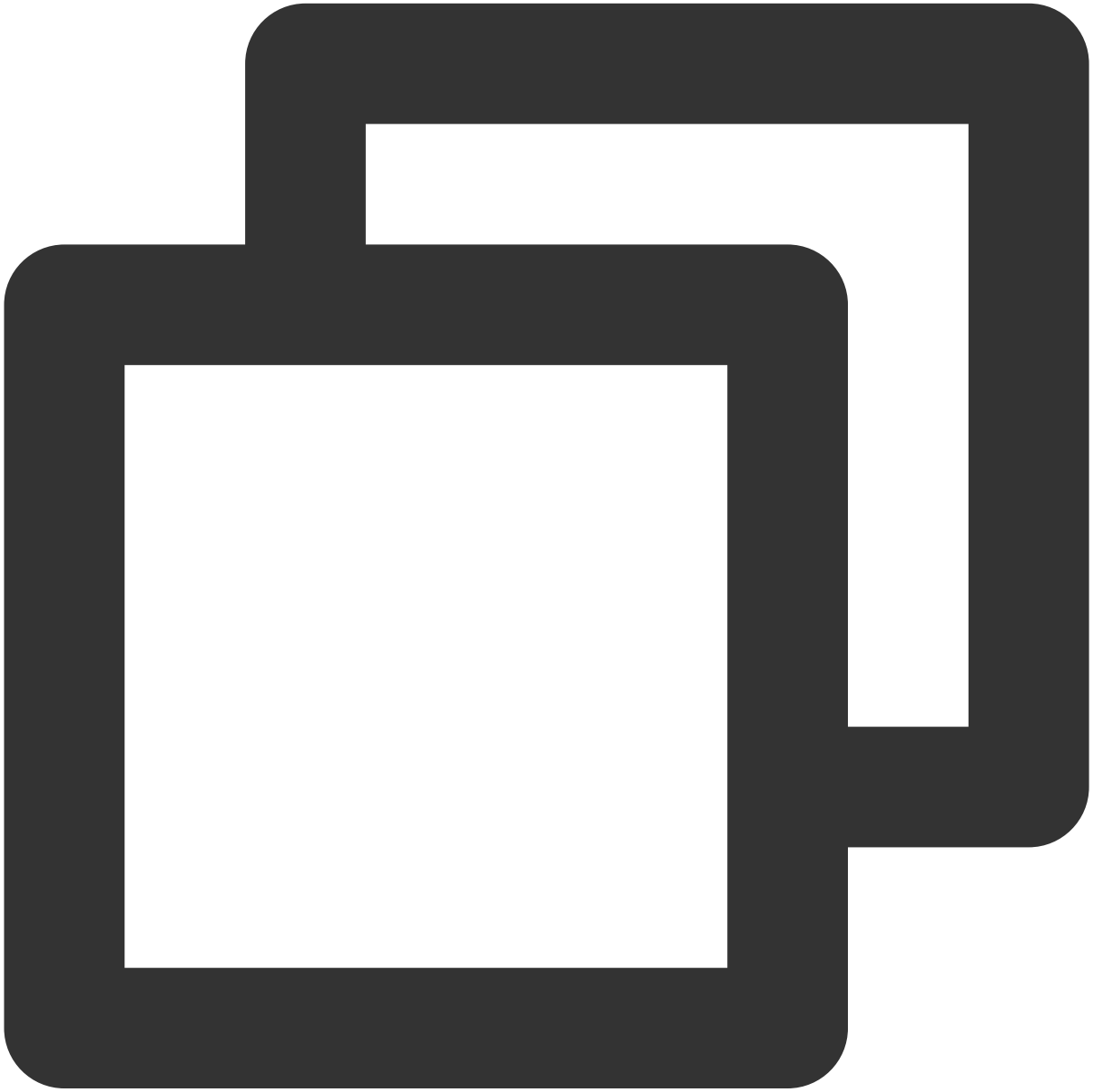
参数如下表所示：

参数	类型	含义
roomData	object	-
roomData.sdkAppId	number	客户的 SDKAppId
roomData.userId	string	用户的唯一 ID

roomData.userSig	string	用户的 UserSig
roomData.userName	string	用户的昵称
roomData.avatarUrl	string	用户的头像地址

createRoom

主持人创建房间。



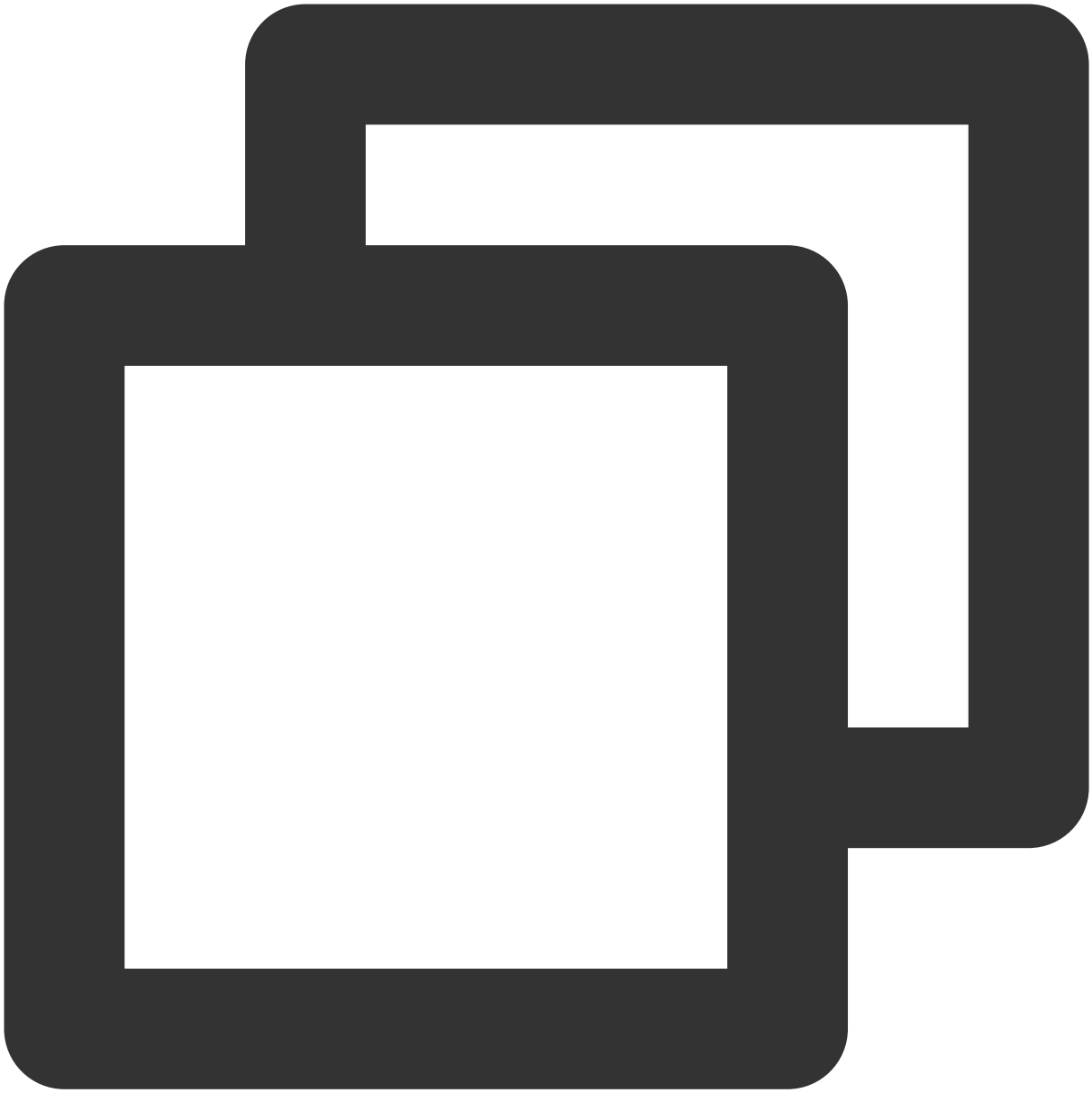
```
TUIRoomRef.value.createRoom(roomId, roomMode, roomParam);
```

参数如下表所示：

参数	类型	含义
roomId	string	房间 ID
roomMode	string	房间模式，'FreeSpeech'（自由发言模式）和 'ApplySpeech'（举手发言模式），默认为 'FreeSpeech'
roomParam	Object	非必填
roomParam.isOpenCamera	string	非必填，进房是否打开摄像头，默认为关闭
roomParam.isOpenMicrophone	string	非必填，进房是否打开麦克风，默认为关闭
roomParam.defaultCameraId	string	非必填，默认摄像头设备 ID
roomParam.defaultMicrophoneId	string	非必填，默认麦克风设备 ID
roomParam.defaultSpeakerId	String	非必填，默认扬声器设备 ID

enterRoom

普通成员加入房间。



```
TUIRoomRef.value.enterRoom(roomId, roomParam);
```

参数如下表所示：

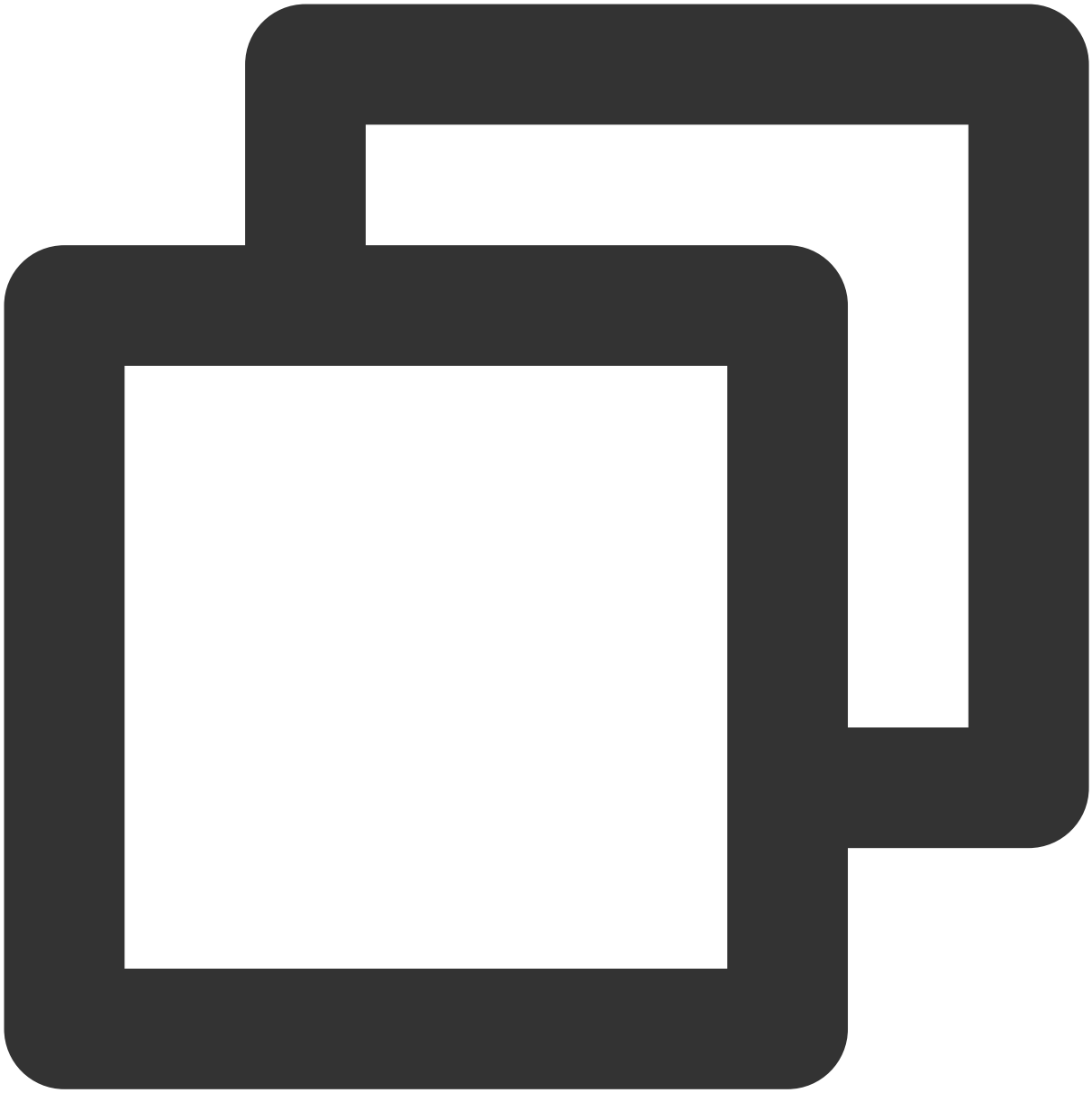
参数	类型	含义
roomId	string	房间 ID
roomParam	Object	非必填
roomParam.isOpenCamera	string	非必填，进房是否打开摄像头，默认为关闭

roomParam.isOpenMicrophone	string	非必填，进房是否打开麦克风，默认为关闭
roomParam.defaultCameraId	string	非必填，默认摄像头设备 ID
roomParam.defaultMicrophoneId	string	非必填，默认麦克风设备 ID
roomParam.defaultSpeakerId	String	非必填，默认扬声器设备 ID

TUIRoomkit 事件

onCreateRoom

创建房间回调。

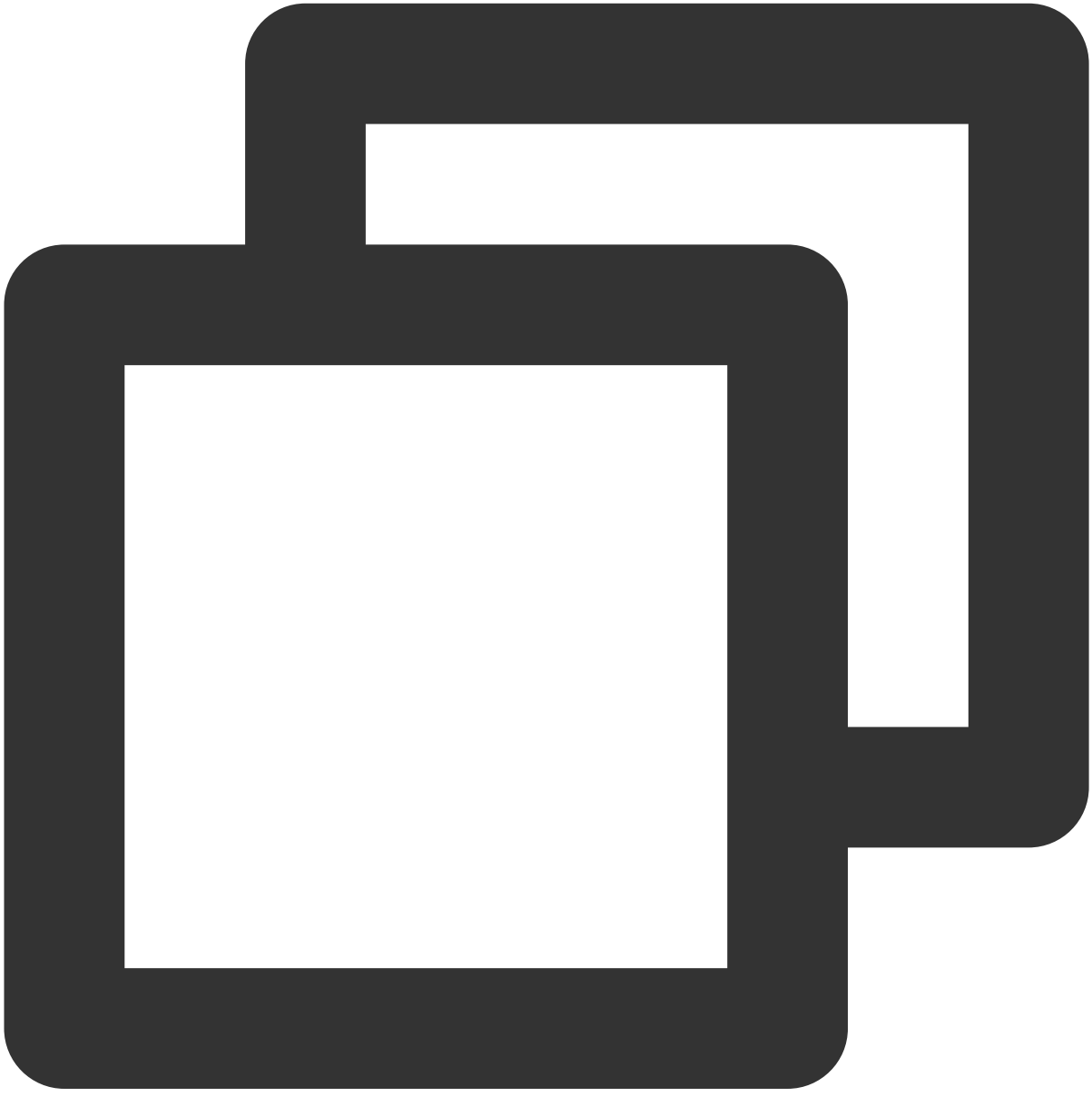


```
<template>
  <room ref="TUIRoomRef" @on-create-room="onCreateRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onCreateRoom(info) {
    if (info.code === 0) {
      console.log('创建房间成功')
```

```
    }
  }
</script>
```

onEnterRoom

进入房间回调。

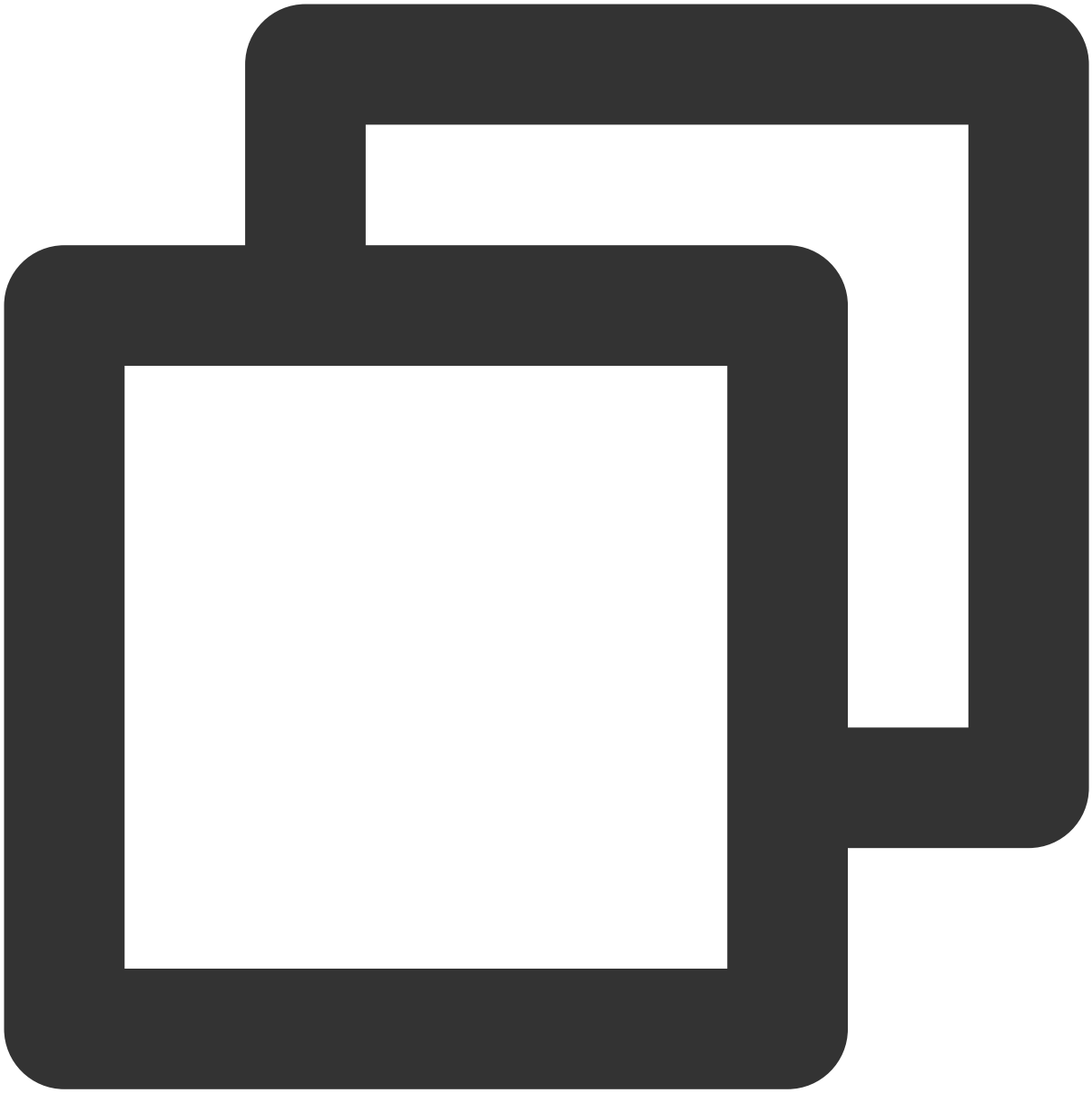


```
<template>
  <room ref="TUIRoomRef" @on-enter-room="onEnterRoom"></room>
</template>
```

```
<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onEnterRoom(info) {
    if (info.code === 0) {
      console.log('进入房间成功')
    }
  }
</script>
```

onDestroyRoom

主持人销毁房间通知。

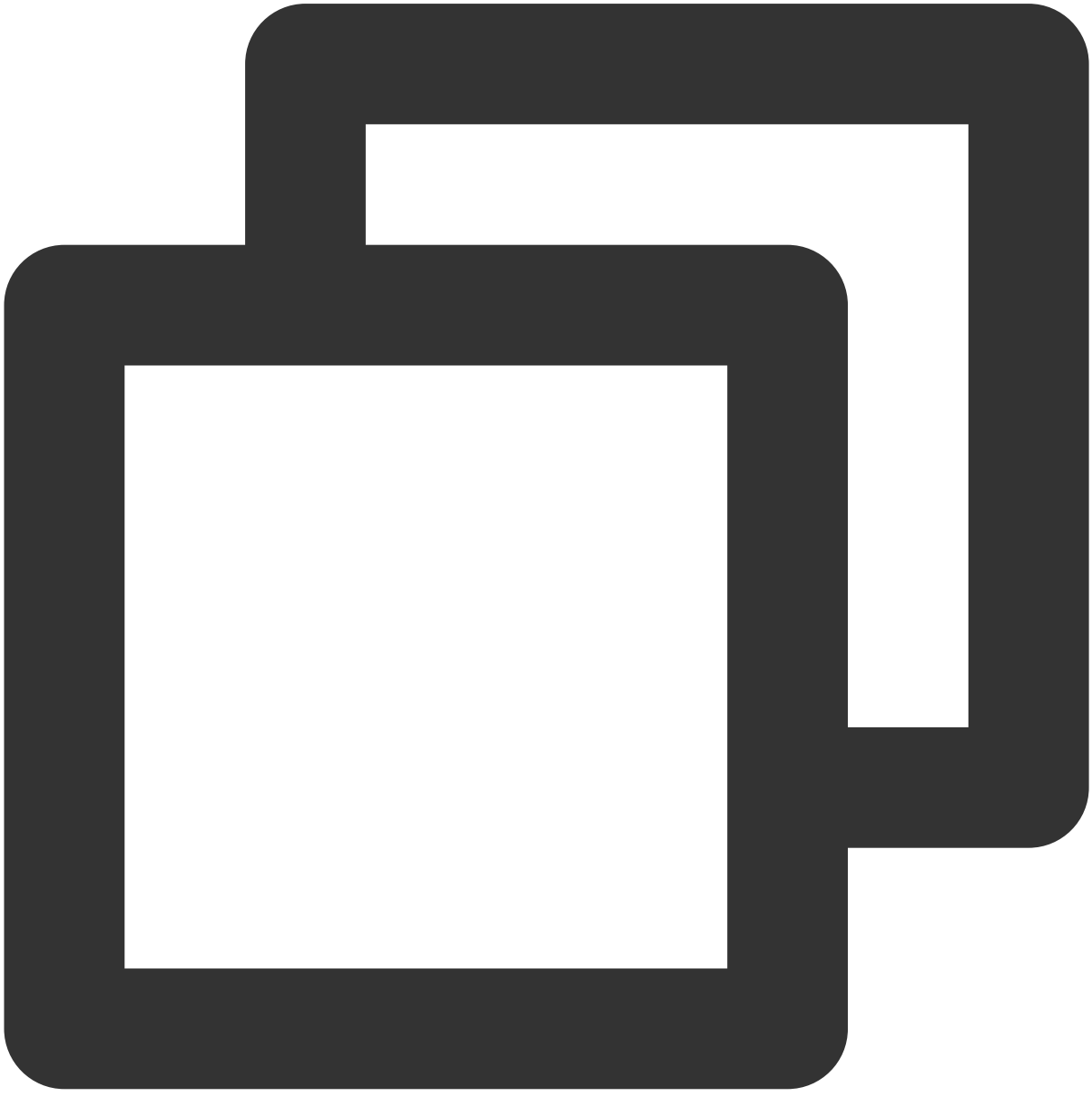


```
<template>
  <room ref="TUIRoomRef" @on-destroy-room="onDestroyRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onDestroyRoom(info) {
    if (info.code === 0) {
      console.log('主持人销毁成功')
```

```
    }
  }
</script>
```

onExitRoom

普通成员退出房间通知。

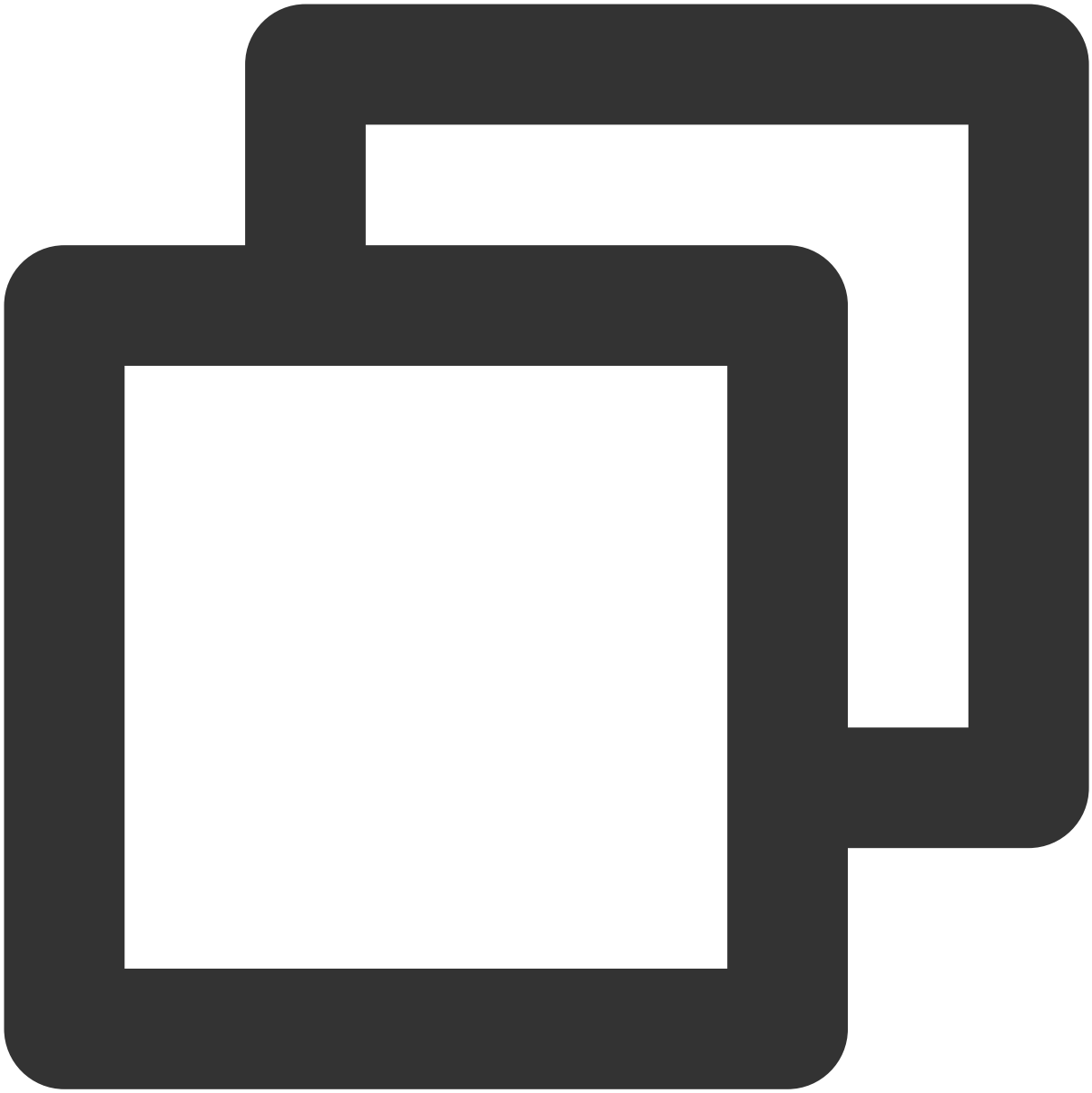


```
<template>
  <room ref="TUIRoomRef" @on-exit-room="onExitRoom"></room>
</template>
```

```
<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onExitRoom(info) {
    if (info.code === 0) {
      console.log('普通成员退出房间成功')
    }
  }
</script>
```

onKickedOutOfRoom

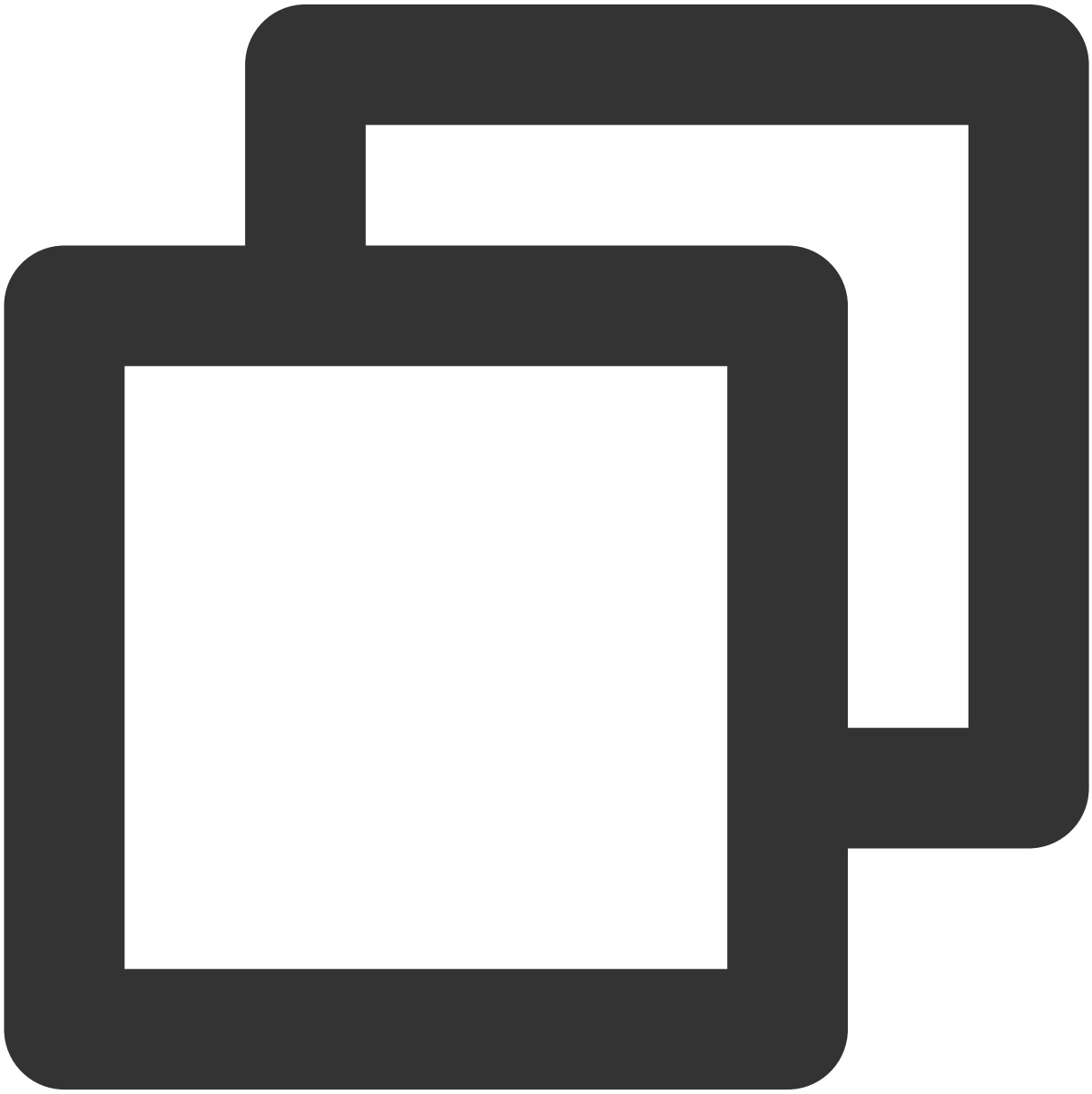
普通成员被主持人踢出房间通知。



```
<template>
  <room ref="TUIRoomRef" @on-kicked-out-of-room="onKickedOutOfRoom"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onKickedOutOfRoom({ roomId, message }) {
    console.log('普通成员被主持人踢出房间', roomId, message);
  }
```

```
</script>
```

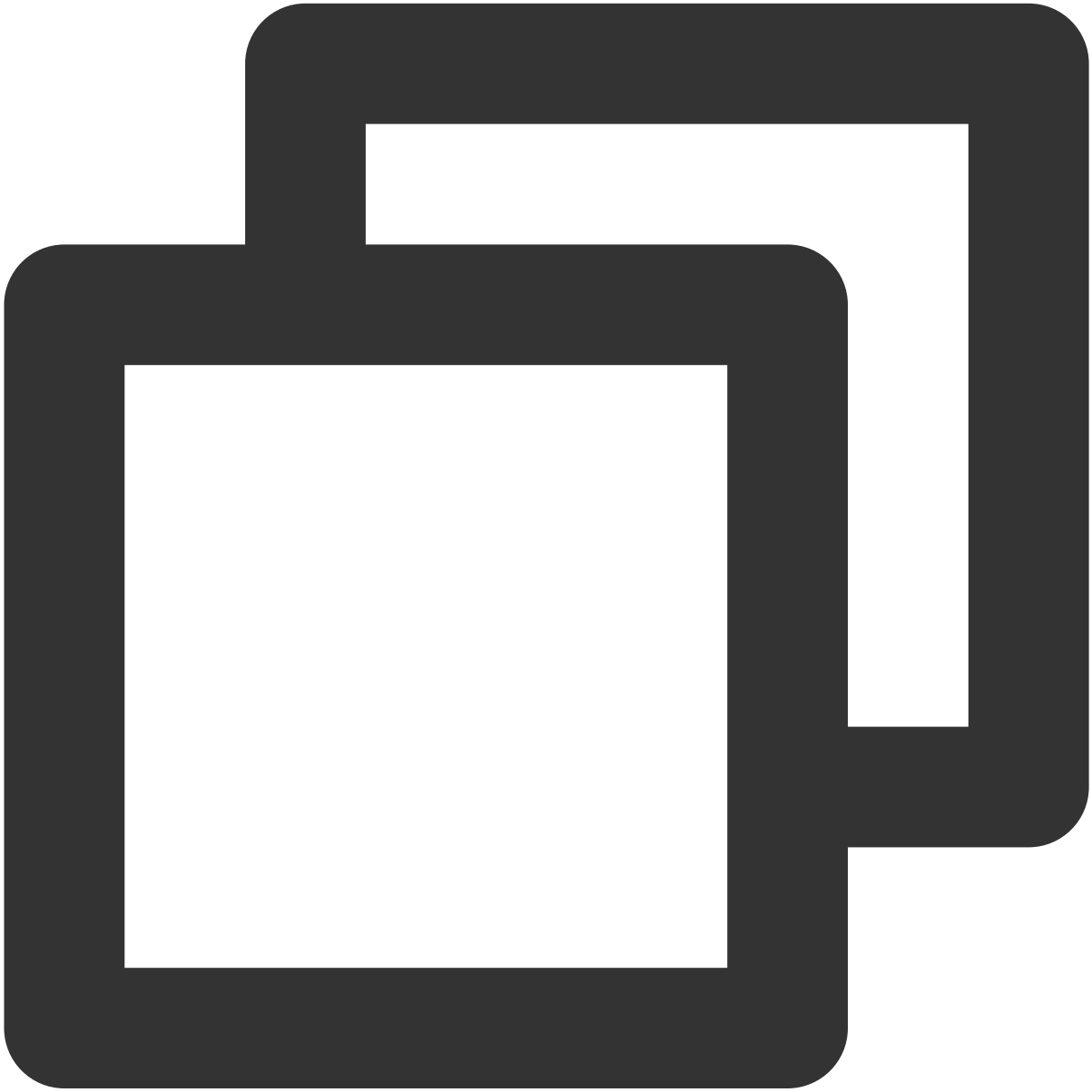
onKickedOffLine



```
<template>
  <room ref="TUIRoomRef" @on-kick-off-line="onKickedOffLine"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
```

```
  import Room from './TUIRoom/index.vue';

  function onKickedOffLine({ message }) {
    console.log('成员被踢下线', message);
  }
</script>
```

onUserSigExpired



```
<template>
```

```
  <room ref="TUIRoomRef" @on-user-sig-expired="onUserSigExpired"></room>
</template>

<script setup lang="ts">
  // 引入 TUIRoom 组件，注意确认引入路径是否正确
  import Room from './TUIRoom/index.vue';

  function onUserSigExpired() {
    console.log('用户 userSig 过期，请重新生成 userSig.');
  }
</script>
```

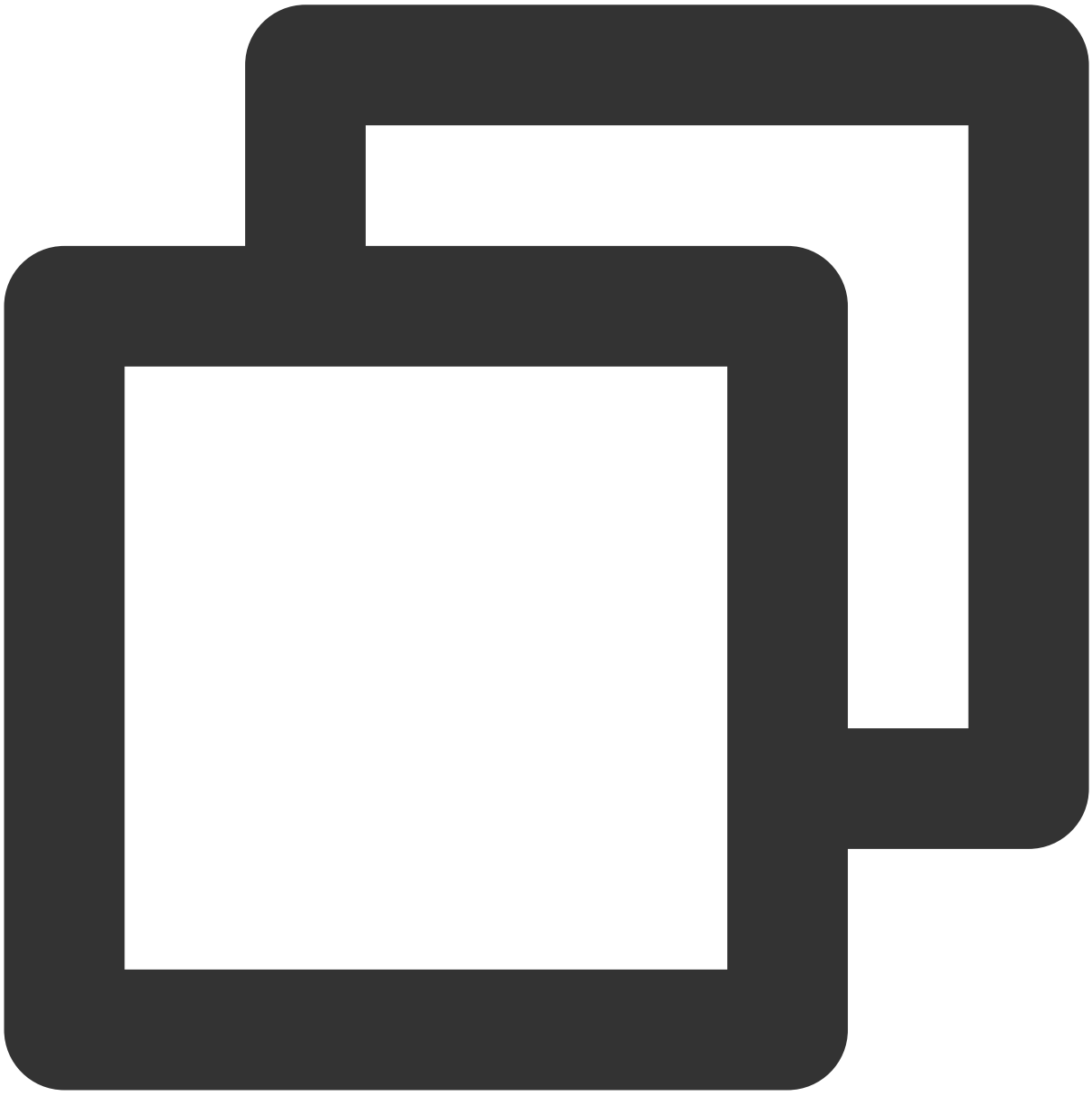
TUIRoomEngine

最近更新时间：2023-11-16 14:36:12

TUIRoomEngine 简介

TUIRoomEngine SDK 提供了房间管理，多人实时音视频互动，屏幕分享，成员管理，即时聊天等功能。

安装方式：



```
// 使用 npm
npm i @tencentcloud/tuiroom-engine-electron --save

// 使用 pnpm
pnpm i @tencentcloud/tuiroom-engine-electron --save

// 使用 yarn
yarn add @tencentcloud/tuiroom-engine-electron
```

TUIRoomEngine API

TUIRoomEngine 静态方法

API	描述
once	监听 TUIRoomEngine ready 事件。 **注意：TUIRoomEngine.login 以外的所有方法必须在监听到 TUIRoomEngine ready 事件且 TUIRoomEngine.login 方法执行成功后执行。**
login	登录 TUIRoomEngine
setSelfInfo	设置当前用户基本信息（用户名、用户头像）
getSelfInfo	获取当前用户基本信息（用户名、用户头像）
logout	登出 TUIRoomEngine

roomEngine 房间管理 API

API	描述
createRoom	创建房间
enterRoom	进入房间
destroyRoom	销毁房间
exitRoom	离开房间
fetchRoomInfo	获取房间信息
updateRoomNameByAdmin	更新房间的名字（仅群主或者管理员可以调用）

updateRoomSpeechModeByAdmin	更新房间的发言模式（仅群主或者管理员可以调用）
getUserList	获取当前房间用户列表
getUserInfo	获取用户的详细信息

roomEngine 音视频 API

API	描述
setLocalVideoView	设置本地流的渲染位置
openLocalCamera	本地摄像头视频流采集
closeLocalCamera	关闭本地摄像头
openLocalMicrophone	打开本地麦克风
closeLocalMicrophone	关闭本地麦克风
updateVideoQuality	设置本地视频的参数
updateAudioQuality	设置本地音频的参数
startPushLocalVideo	开始向远端推本地视频流
stopPushLocalVideo	停止向远端推本地视频流
startPushLocalAudio	开始向远端推本地音频流
stopPushLocalAudio	停止向远端推本地音频流
setRemoteVideoView	设置远端流渲染的区域
startPlayRemoteVideo	开始播放远端用户视频流
stopPlayRemoteVideo	停止播放远端用户视频流
muteRemoteAudioStream	停止远端用户的音频流

roomEngine 成员管理 API

API	描述
openRemoteDeviceByAdmin	请求远端用户打开媒体设备
applyToAdminToOpenLocalDevice	参会者向主持人申请打开设备

版权所有：腾讯云计算（北京）有限责任公司

closeRemoteDeviceByAdmin	关闭远端用户媒体设备
cancelRequest	取消已经发出的请求
responseRemoteRequest	回复远端用户的请求
changeUserRole	改变用户的角色
kickRemoteUserOutOfRoom	将用户踢出房间
disableDeviceForAllUserByAdmin	禁用/打开所有用户的媒体设备
disableSendingMessageForAllUser	禁止/允许所有用户发送消息
disableSendingMessageByAdmin	禁止/允许某个用户发送消息

roomEngine 屏幕分享 API

API	描述
startScreenSharingElectron	开始屏幕共享
stopScreenSharingElectron	停止屏幕共享
getScreenSharingTarget	获取屏幕分享列表
selectScreenSharingTarget	切换屏幕分享窗口

roomEngine 麦位管理API

API	描述
setMaxSeatCount	设置房间麦位最大值
getSeatList	获取麦位信息
takeSeat	获取麦位
leaveSeat	释放麦位
takeUserOnSeatByAdmin	邀请其他人上麦（仅房间主持人和管理员可调用此方法）
kickUserOffSeatByAdmin	将其他人踢下麦（仅房间主持人和管理员可调用此方法）
lockSeatByAdmin	锁定某个麦位状态（仅房间主持人和管理员可调用此方法）

roomEngine 消息发送API

版权所有：腾讯云计算（北京）有限责任公司

API	描述
sendTextMessage	发送文本消息
sendCustomMessage	发送自定义消息

roomEngine 设备管理 API

API	描述
getCameraDevicesList	获取摄像头设备列表
getMicDevicesList	获取麦克风设备列表
getSpeakerDevicesList	获取扬声器设备列表
setCurrentCameraDevice	设置要使用的摄像头设备
setCurrentMicDevice	设置要使用的麦克风设备
setCurrentSpeakerDevice	设置要使用的扬声器设备
getCurrentCameraDevice	获取当前正在使用的摄像头设备
getCurrentMicDevice	获取当前正在使用的麦克风设备
getCurrentSpeakerDevice	获取当前正在使用的扬声器设备
startCameraDeviceTest	开始进行摄像头测试
stopCameraDeviceTest	停止摄像头测试

roomEngine 事件监听 API

API	描述
on	监听 TUIRoomEvents 事件
off	取消监听 TUIRoomEvents 事件

roomEngine 其他 API

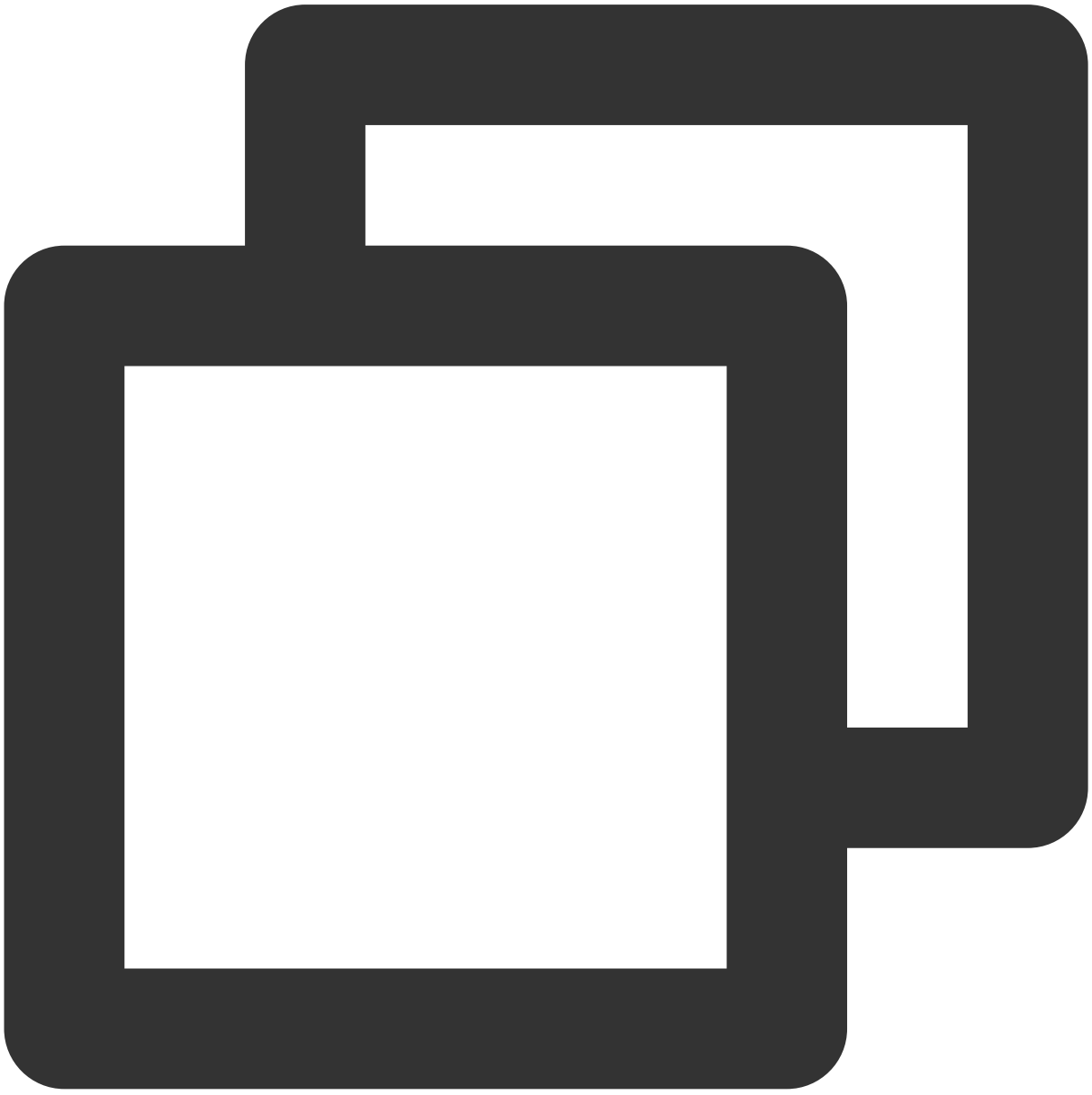
API	描述

getTRTCCloud	获取 trtcCloud 实例
getTIM	获取 tim 实例

API 详情

once

监听 TUIRoomEngine 'ready' 事件



```
TUIRoomEngine.once('ready', () => {
  const roomEngine = new TUIRoomEngine();

  await TUIRoomEngine.init({
    sdkAppId: 0,    // 填写您申请的 sdkAppId
    userId: '',     // 填写您业务对应的 userId
    userSig: '',    // 填写服务器或本地计算的 userSig
  });

  await roomEngine.createRoom({
    roomId: '12345',    // 填入您的房间 Id，注意房间 Id 要求为字符串类型
    name: 'Test Room',      // 填入您的房间名称，房间名称默认为 roomId，最长 30 字节
    roomType: TUIRoomType.kGroup, // 设置房间类型为 TUIRoomType.kGroup 类型
  });
});
```

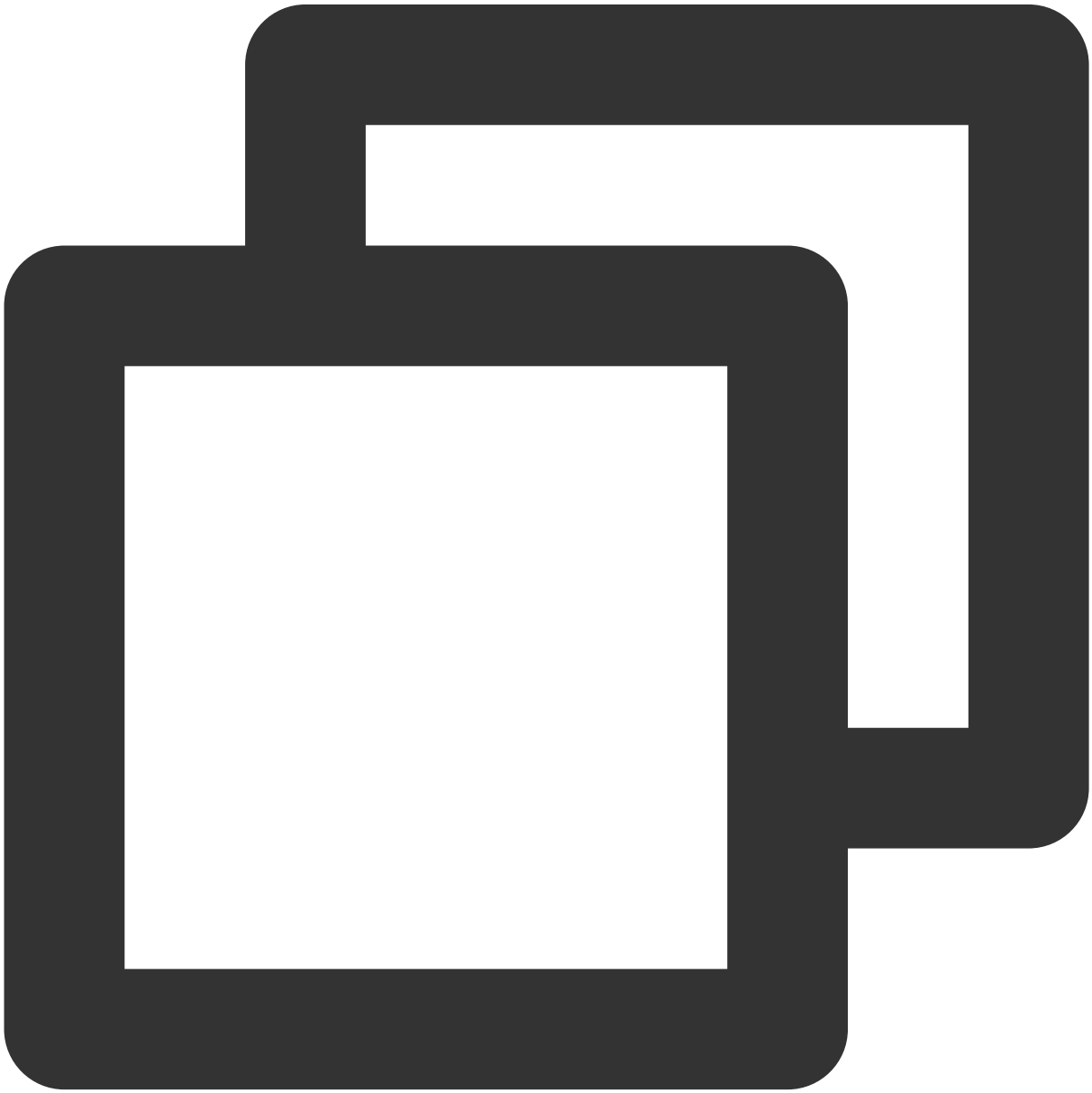
login

说明：

v1.0.0 版本该接口命名为 TUIRoomEngine.init 接口，v1.0.1 及其以上版本请使用 TUIRoomEngine.login 登录 TUIRoomEngine。

登录 TUIRoomEngine， 您必须在登录 TUIRoomEngine 之后才可以调用 TUIRoomEngine 及其实例的其他方法。

版权所有：腾讯云计算（北京）有限责任公司



```
// 登录 TUIRoomEngine
await TUIRoomEngine.login({
 sdkAppId: 0,    // 填写您申请的 sdkAppId
 userId: '',     // 填写您业务对应的 userId
 userSig: '',    // 填写服务器或本地计算的 userSig
});
```

参数：

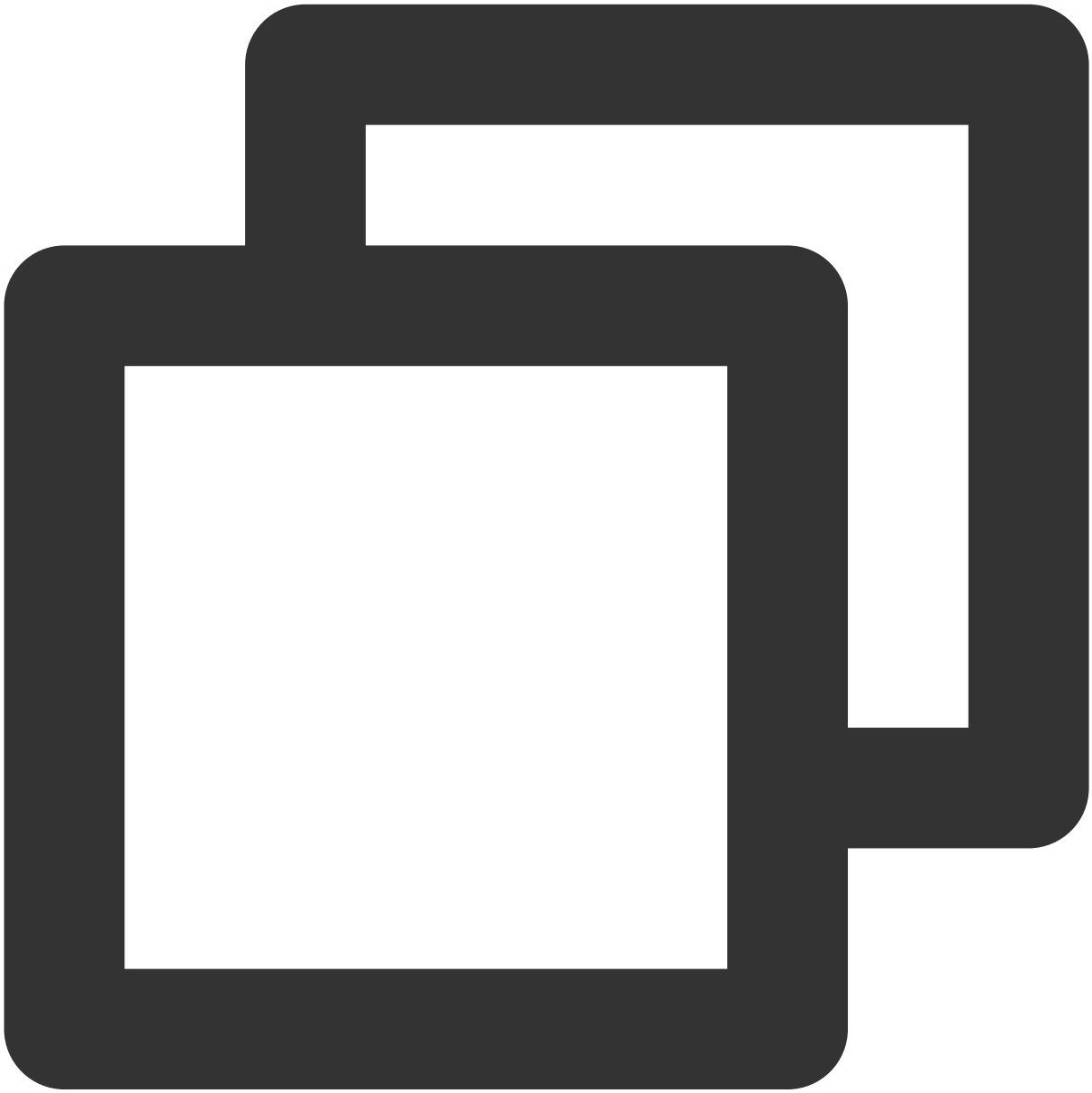
参数	类型	说明	默认值	含义

sdkAppId	number	必填	-	sdkAppId 在 实时音视频控制台 单击 应用管理 > 创建应用 创建新应用之后，即可在 应用信息 中获取 sdkAppId 信息。
userId	string	必填	-	用户ID建议限制长度为32字节，只允许包含大小写英文字母(a-zA-Z)、数字(0-9)及下划线和连词符。
userSig	string	必填	-	userSig 签名 计算 userSig 的方式请参考 UserSig 相关。
tim	TIM	非必填	-	若您在接入 roomEngine 的同时想使用即时通信 SDK 的更多能力，您可以将创建的 tim 实例传入 TUIRoomEngine。tim 实例的创建方式请参考 TIM.create。

Returns *Promise<void>*

setSelfInfo

设置当前用户基本信息（用户名、用户头像）。



```
// 设置当前用户用户名及用户头像
await TUIRoomEngine.setSelfInfo({
 userName: '',      // 填写您的新用户名
 avatarUrl: '',     // 填写您的新头像地址
});
```

参数：

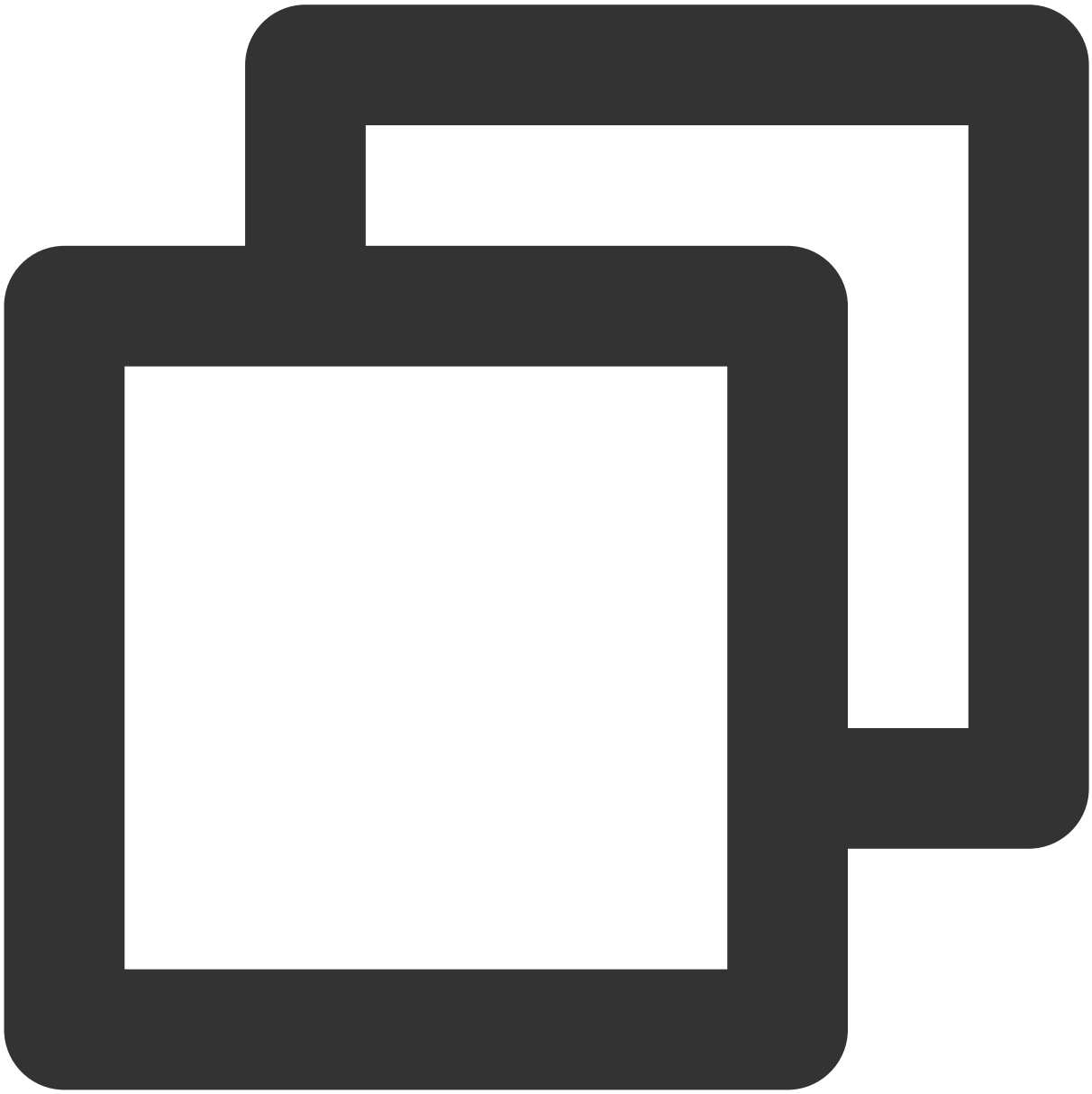
参数	类型	说明	默认值	含义
userName	string	必填	-	用户名

avatarUrl	string	必填	-	用户头像

Returns *Promise<void>*

getSelfInfo

获取当前用户基本信息（用户名、用户头像）。



```
// 获取当前用户用户名及用户头像
const loginUserInfo = await TUIRoomEngine.getSelfInfo();
```

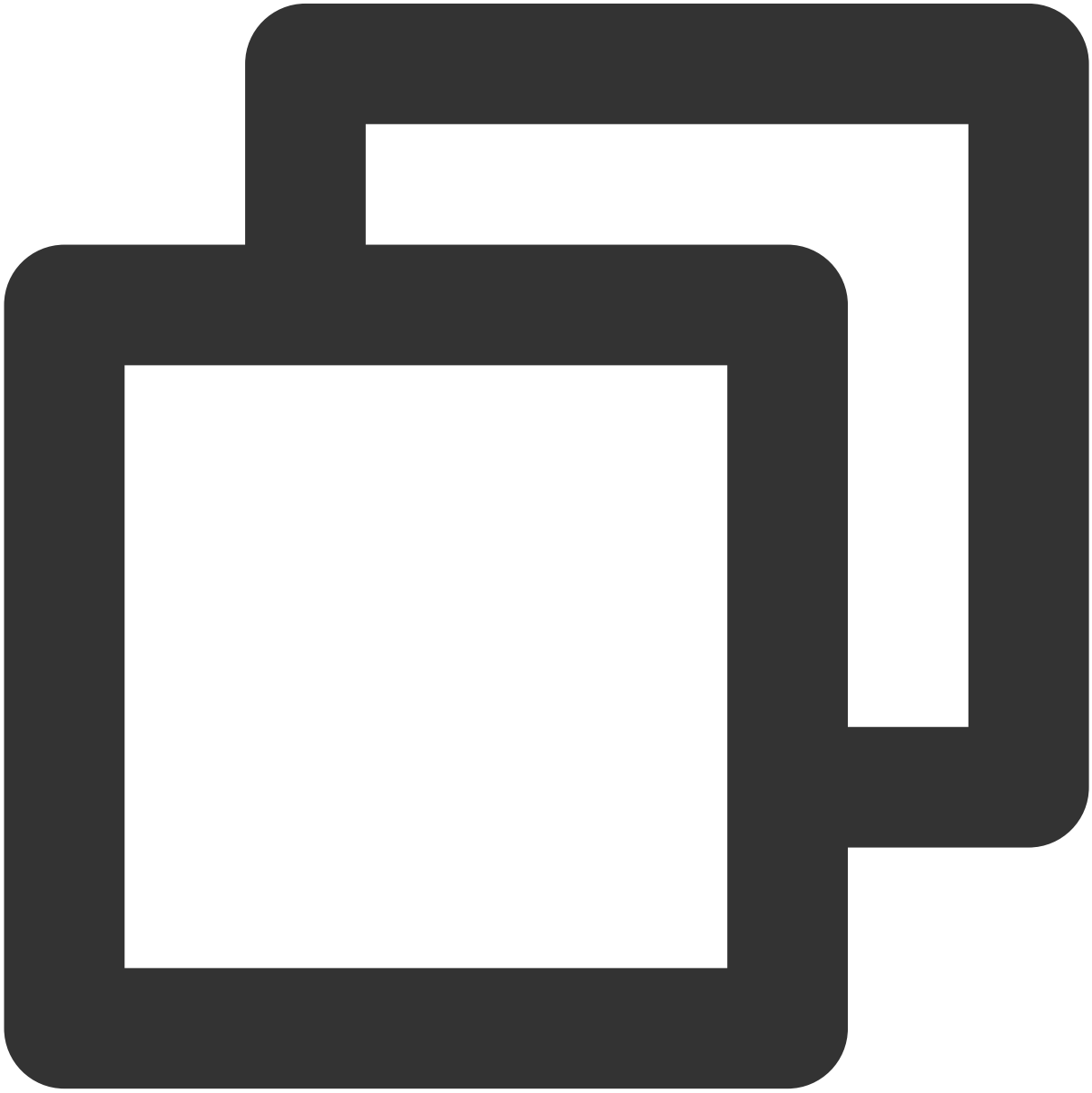
Returns *Promise*<TUILoginUserInfo> *loginUserInfo*

logout

说明：

该接口自 v1.0.1 版本支持。

登出 TUIRoomEngine。

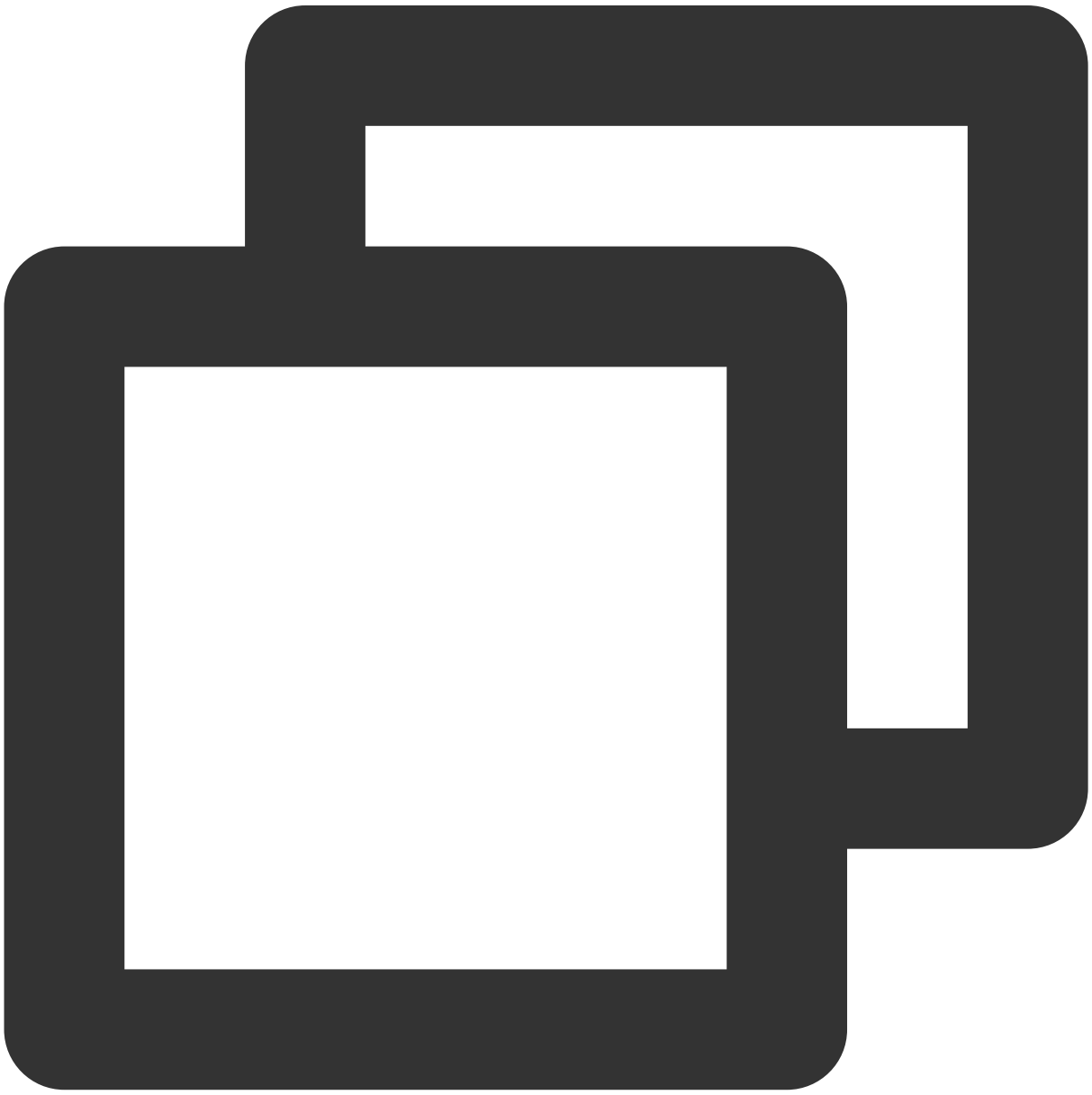


```
// 登出 TUIRoomEngine
await TUIRoomEngine.logout();
```

Returns *Promise<void>*

createRoom

主持人创建房间，调用 createRoom 的用户为房间的所有者。创建房间时可设置房间 ID，房间名称，房间类型，发言模式，是否允许加入用户开启音视频，发送消息等功能。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({
    roomId: '12345',    // 填入您的房间 Id，注意房间 Id 要求为字符串类型
    roomName: 'Test Room',      // 填入您的房间名称，房间名称默认为 roomId，最长 30 字节
```

```
    roomType: TUIRoomType.kConference, // 设置房间类型为 TUIRoomType.kConference 类型
    speechMode: TUISpeechMode.kFreeToSpeak, // 设置发言模式为自由发言模式
    isMicrophoneDisableForAllUser: false,  // 设置是否允许加入用户打开麦克风
    isCameraDisableForAllUser: false,  // 设置是否允许加入用户打开摄像头
    isMessageDisableForAllUser: false,  // 设置是否允许加入用户发送消息
});
```

参数：

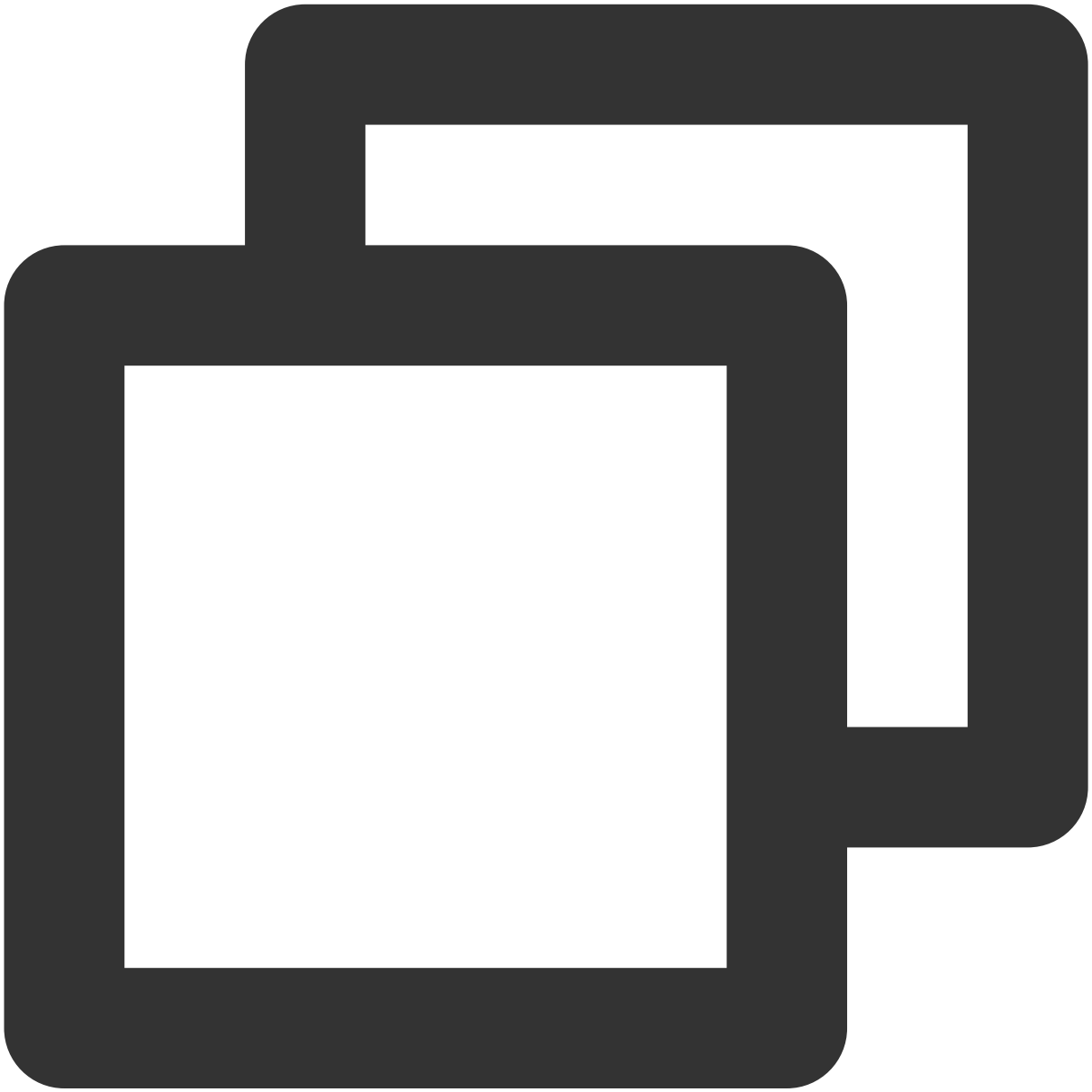
参数	类型	说明	默认值	含义
roomId	string	必填	-	房间 ID，roo 支持以下范围 大小写英文字 数字（0-9） 空格 ! # $ % & | ~ ,
roomName	string	选填	roomId	房间名称，默 符串
roomType	TUIRoomType	选填	TUIRoomType.kConference	房间类型 办公协同、医 景，roomTyp TUIRoomTyp 电商直播、语 TUIRoomTyp
speechMode	TUISpeechMode	选填	TUISpeechMode.kFreeToSpeak	房间内发言模 roomType 为 (教育及会议场 speechMode TUISpeechM 房默认可以打 speechMode TUISpeechM 房默认不打开 人申请打开摄 TUISpeechM TUISpeechM 相切换 speechMode TUISpeechM 用户进房后需 取打开摄像头

				roomType 为(直播场景) 时 speechMode TUISpeechM 需要主持人同 speechMode TUISpeechM 上麦需要主持 TUISpeechM TUISpeechM 模式可互相切
isMicrophoneDisableForAllUser	boolean	选填	false	是否开启全员
isCameraDisableForAllUser	boolean	选填	false	是否开启全员
isMessageDisableForAllUser	boolean	选填	false	是否允许成员
maxSeatCount	number	选填	-	最大麦位数量 roomType 为(教育及会议场 做限制 roomType 为(直播场景) 时 16
enableCDNStreaming	boolean	选填	false	是否开启 CD
cdnStreamDomain	string	选填	''	直播推流域名

Returns *Promise<void>*

enterRoom

进入房间接口。



```
const roomEngine = new TUIRoomEngine();
const roomInfo = await roomEngine.enterRoom({
    roomId: '12345',
});
```

参数：

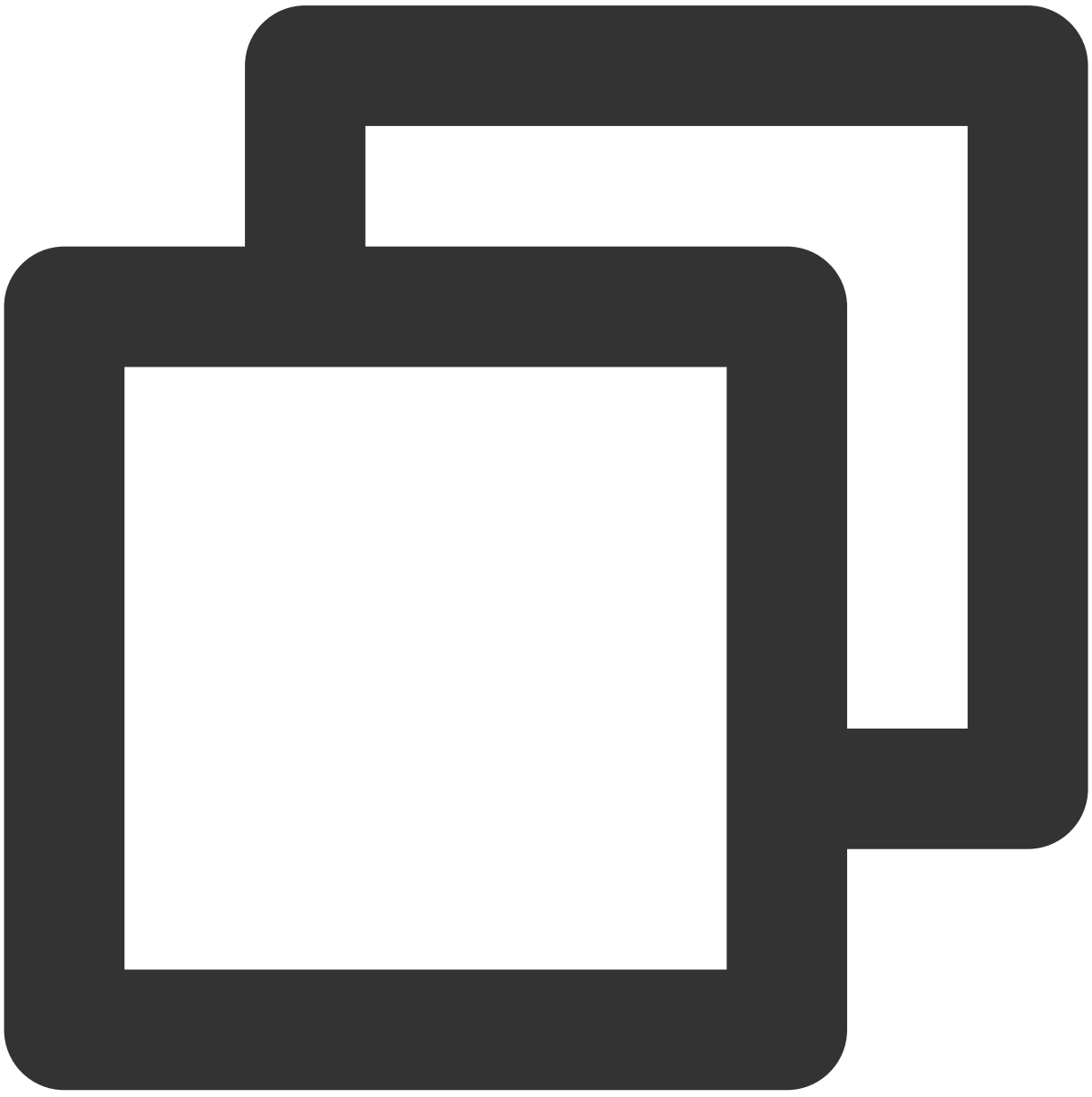
参数	类型	说明	默认值	含义
roomId	string	必填	-	房间号

Returns *Promise<TUIRoomInfo> roomInfo*

该接口返回当前房间信息

destroyRoom

解散房间接口，解散房间必须由房间所有者发起，解散房间之后房间不可进入。

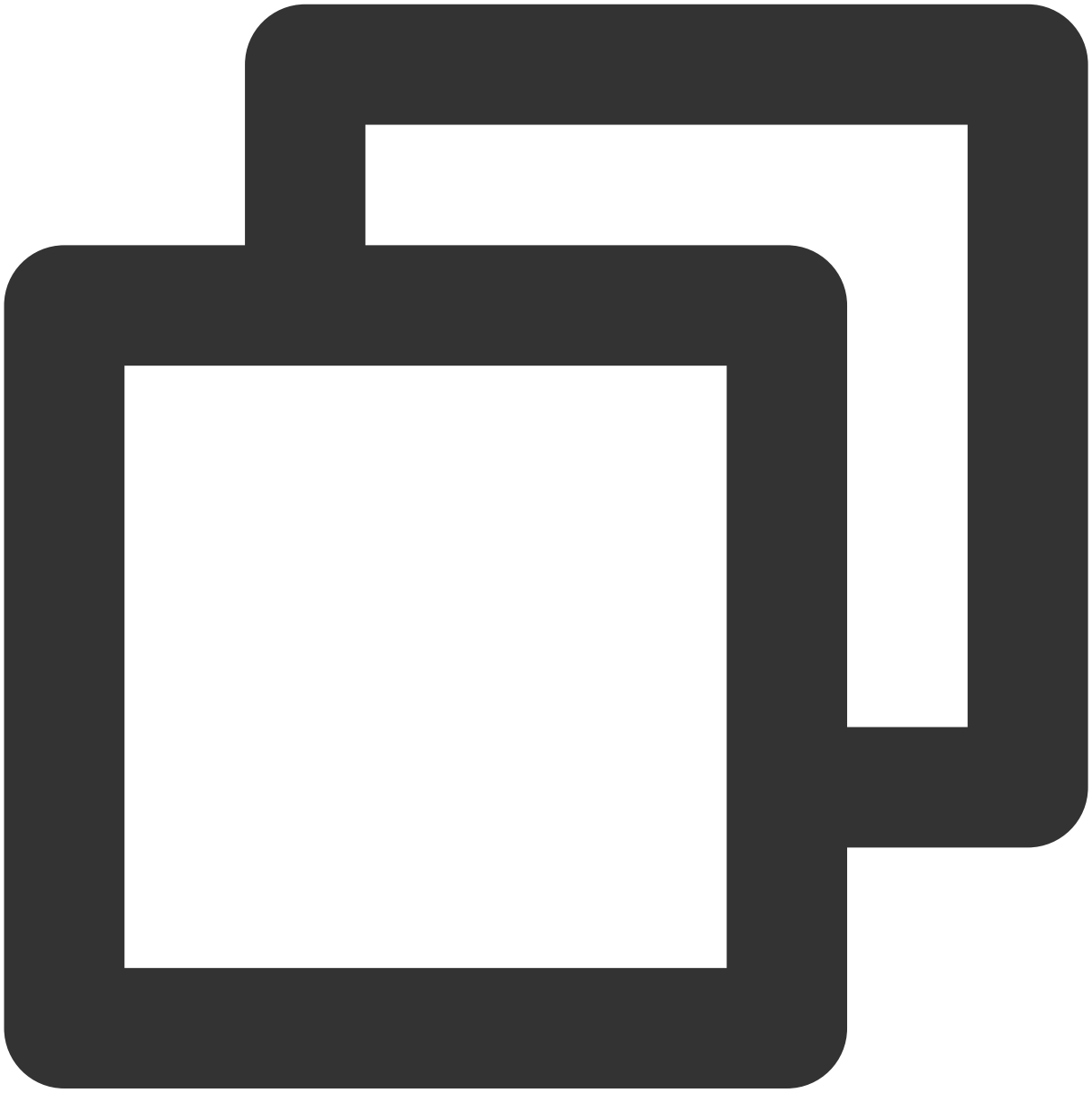


```
const roomEngine = new TUIRoomEngine();
await roomEngine.destroyRoom();
```

Returns *Promise<void>*

exitRoom

离开房间接口，用户在执行 enterRoom 之后可通过 exitRoom 离开房间。

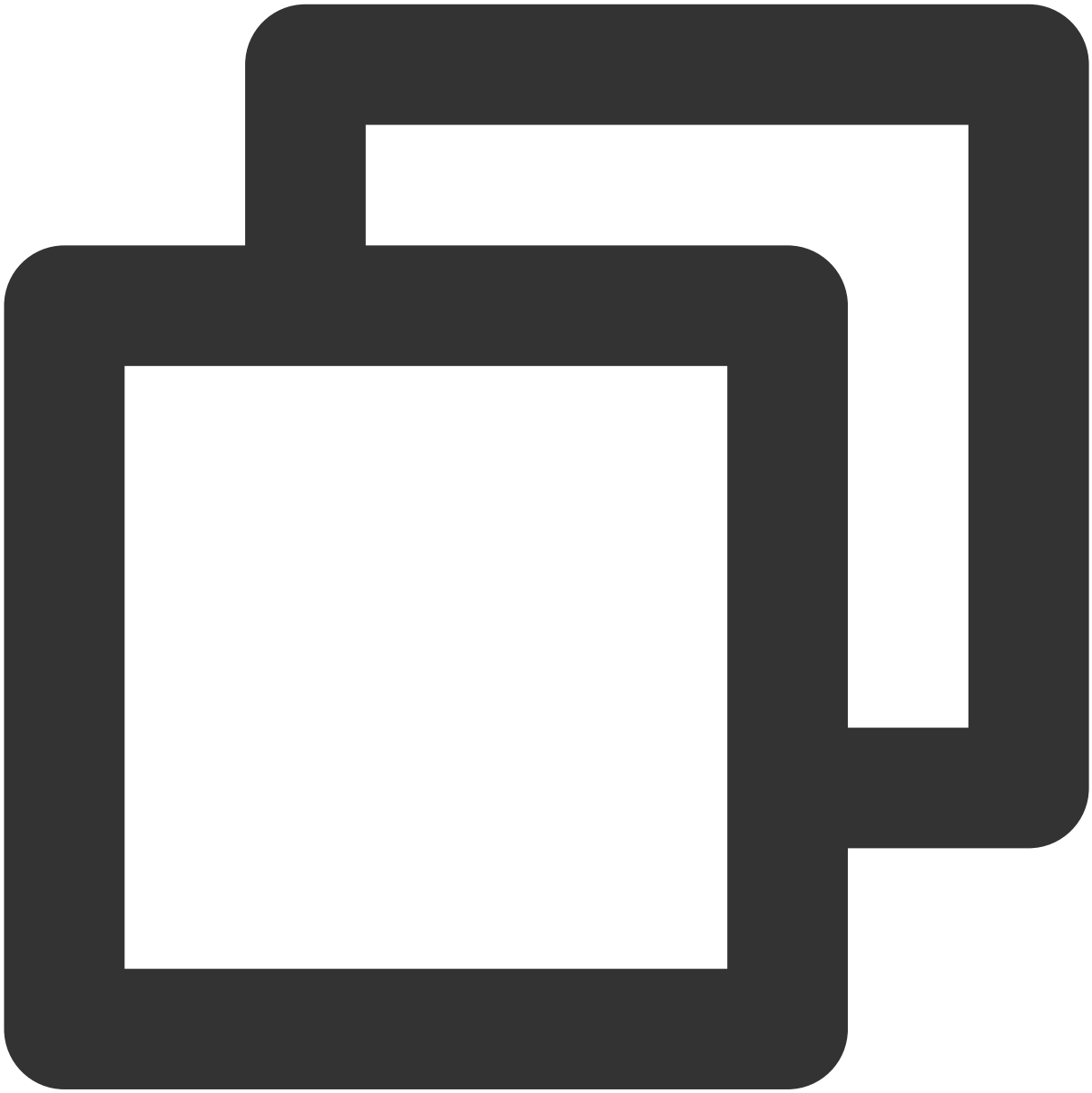


```
const roomEngine = new TUIRoomEngine();
await roomEngine.exitRoom();
```

Returns *Promise<void>*

fetchRoomInfo

获取房间信息。

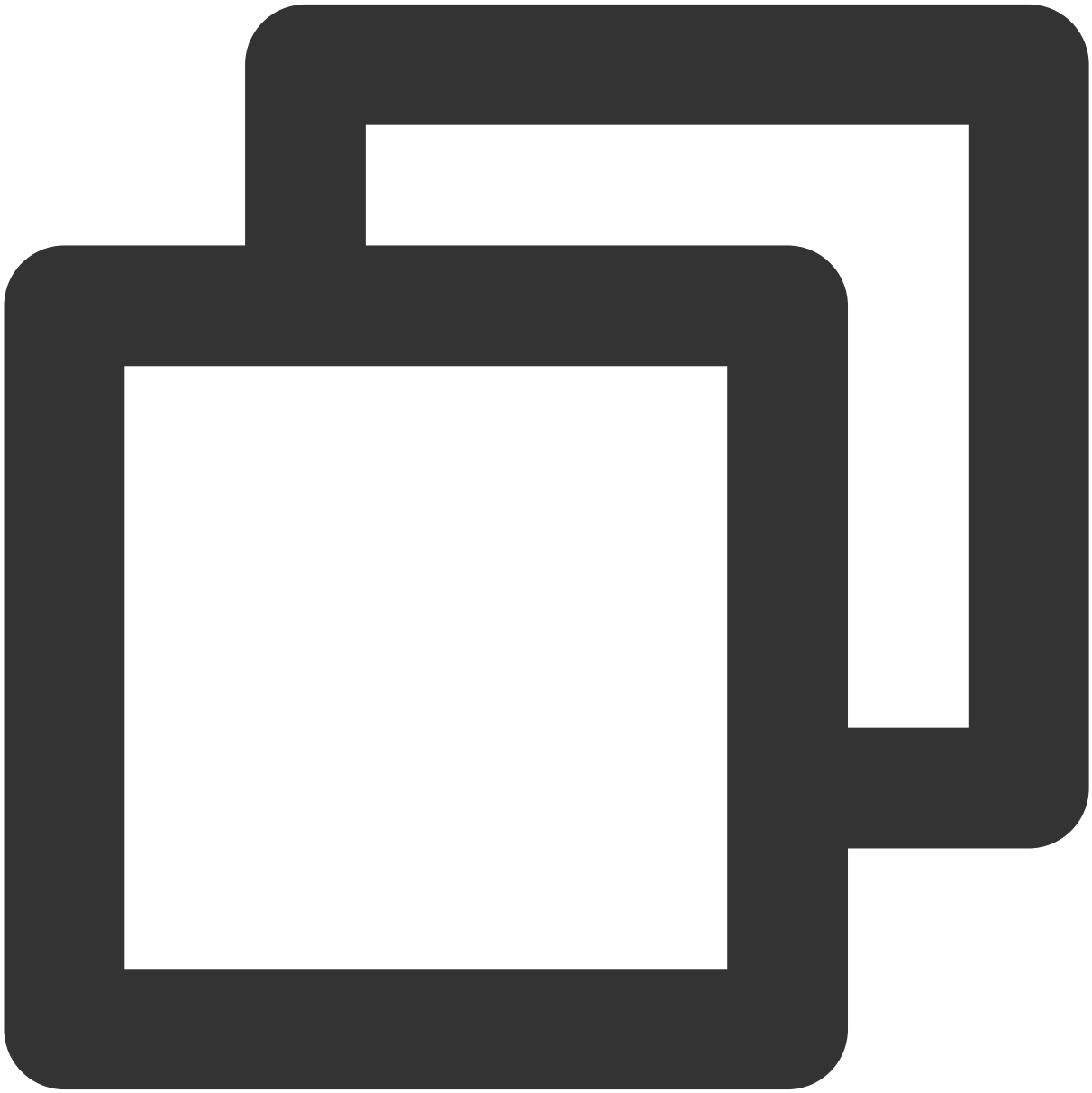


```
const roomEngine = new TUIRoomEngine();
const roomInfo = roomEngine.fetchRoomInfo();
```

Returns：*Promise<*TUIRoomInfo*> roomInfo*

updateRoomNameByAdmin

更新当前房间的名字（仅群主或者管理员可以调用）。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomNameByAdmin({ roomName: '新的名字' });
```

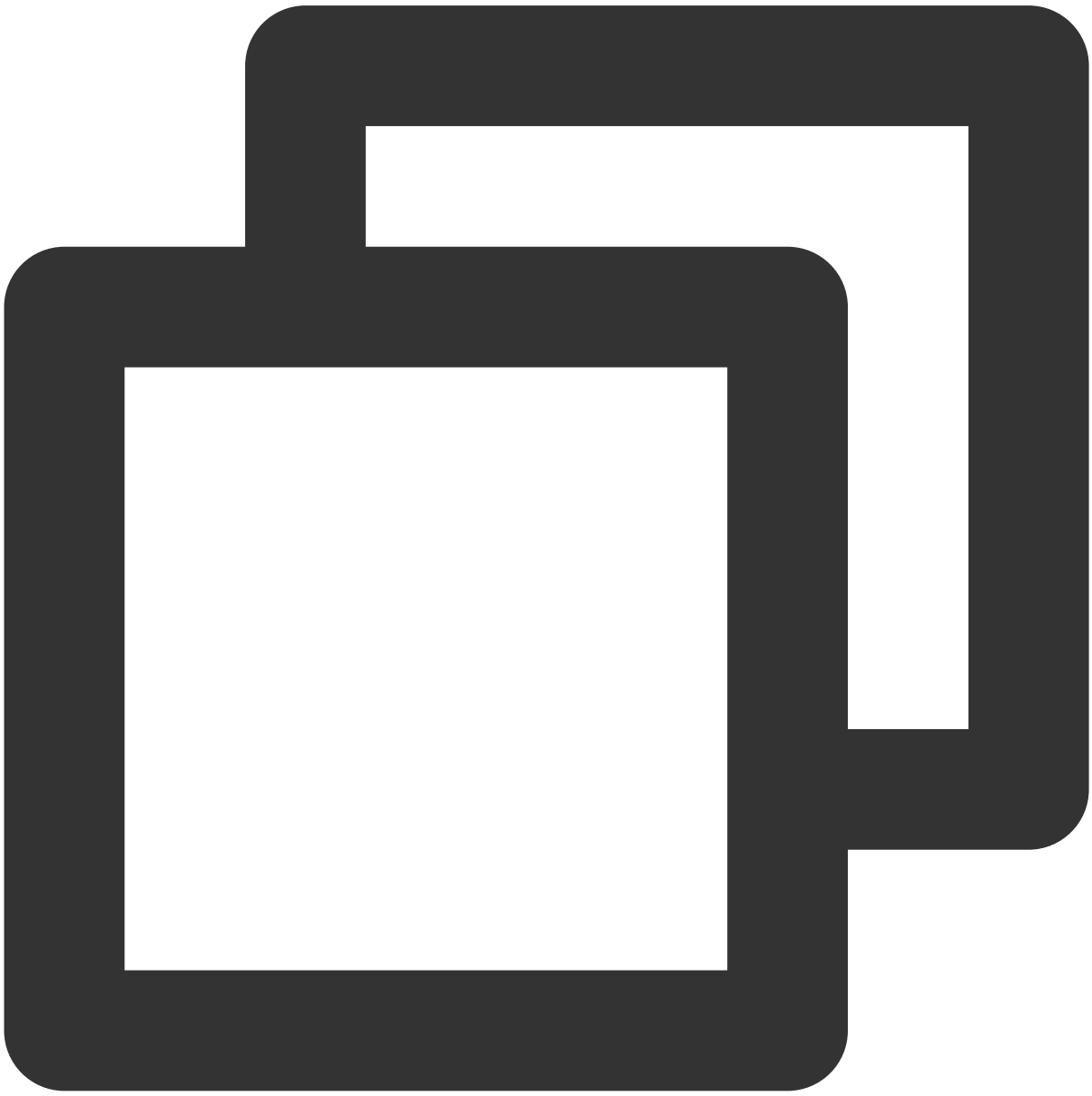
参数：

参数	类型	说明	默认值	含义
roomName	string	必填	-	更新房间的名字，要求 roomName 不为空字符串

Returns *Promise<void>*

updateRoomSpeechModeByAdmin

更新房间的发言模式（仅群主或者管理员可以调用）。



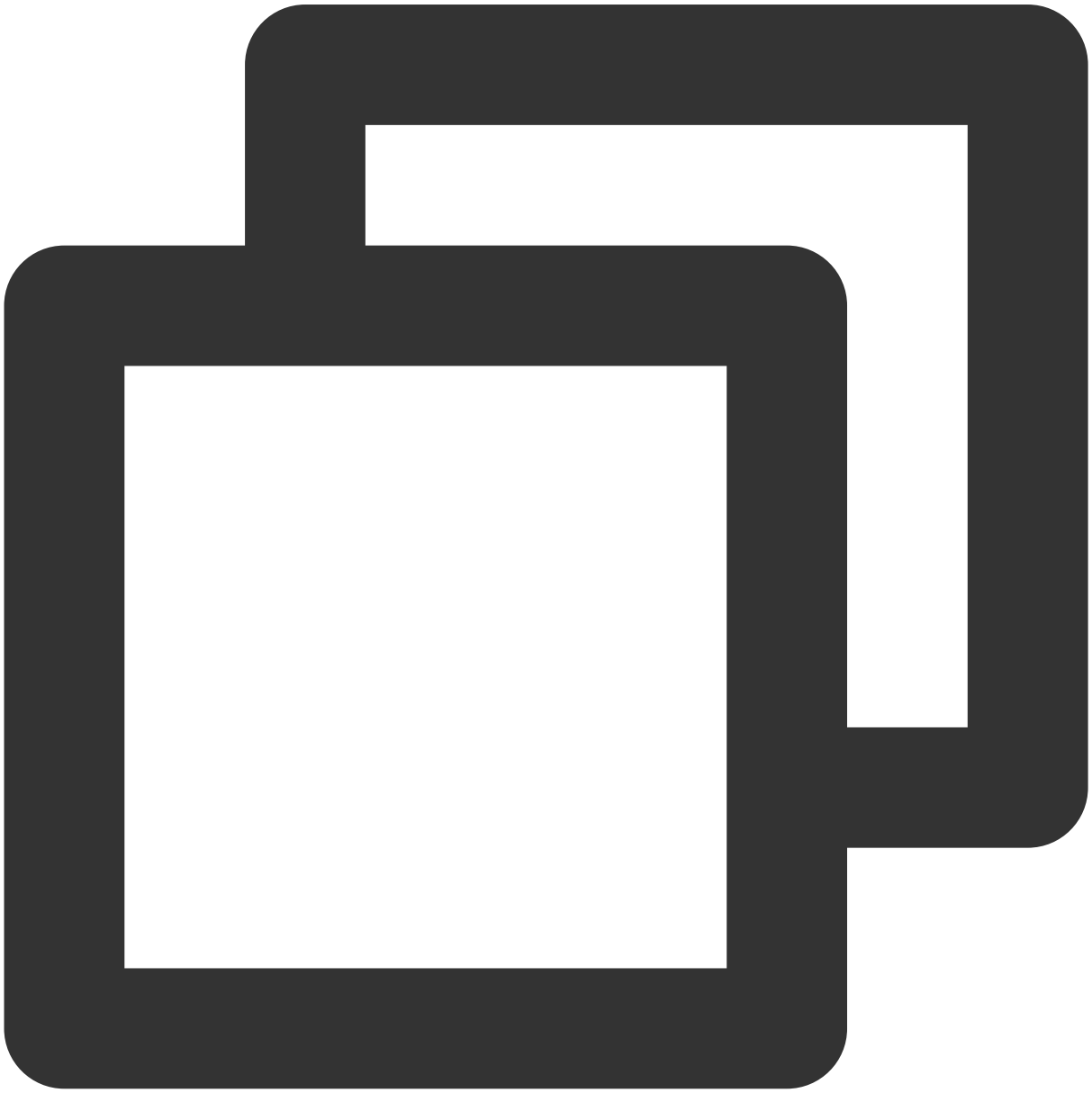
```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.updateRoomSpeechModeByAdmin({
  speechMode: TUISpeechMode.kSpeakAfterTakingSeat  // 更新为上麦后发言模式
});
```

参数：

参数	类型	说明	默认值	含义
speechMode	TUISpeechMode	必填	-	更新房间的发言模式

getUserList

获取当前房间用户列表，注意该接口一次拉取的用户列表量最大为 100 个。



```
const roomEngine = new TUIRoomEngine();
const userList = [];
```

```
let result;
do {
  result = await globalProperties.roomEngine.getUserList();
  userList.push(...result.userInfoList);
} while (result.nextSequence !== 0)
```

参数：

参数	类型	说明	默认值	含义
nextSequence	number	选填	0	偏移量，默认从 0 开始拉取用户

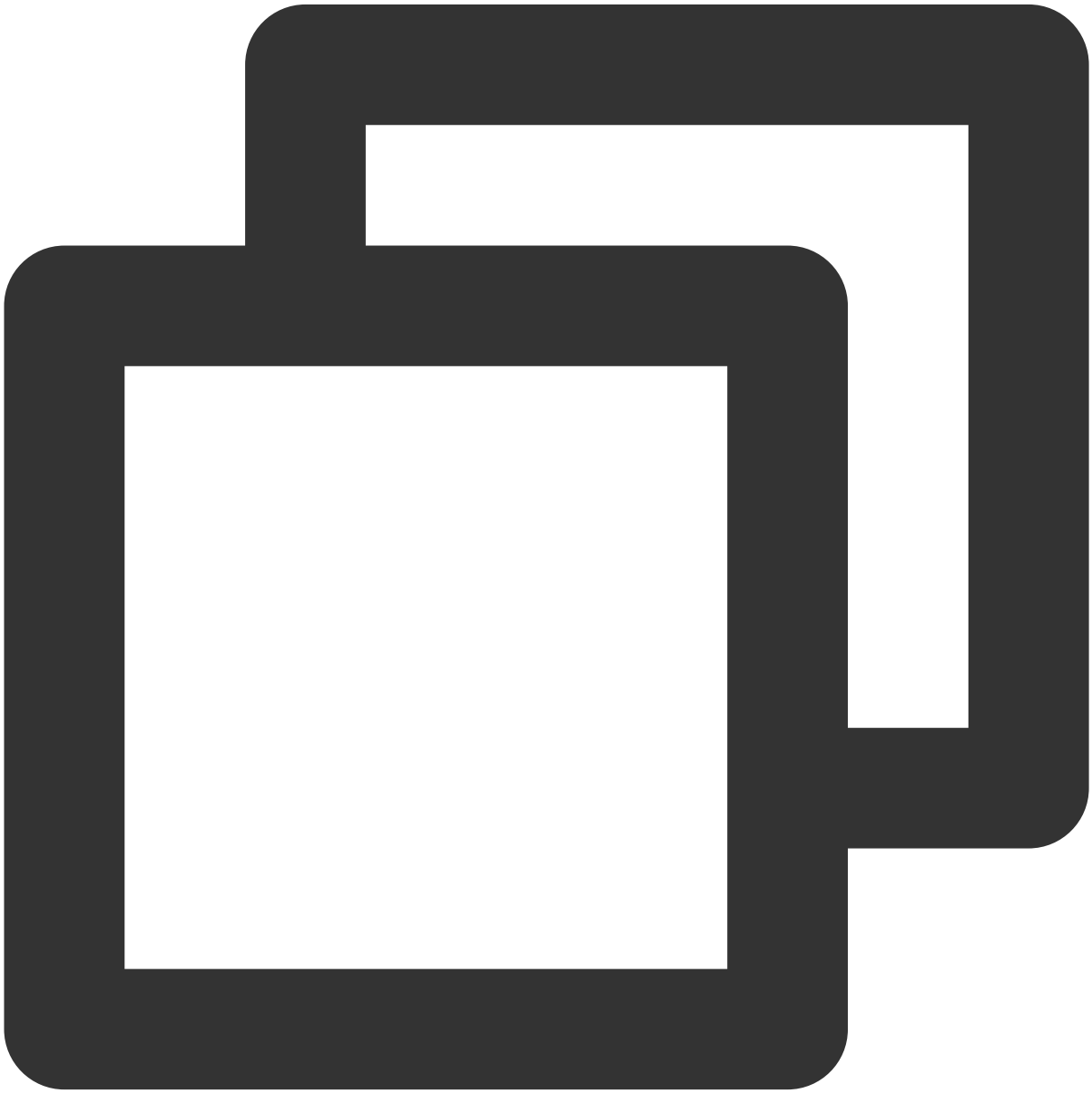
Returns：*Promise<object> result*

result.nextSequence 下一次拉取群组用户的偏移量，如果 result.nextSequence 为 0 则代表 userList 全部拉取完毕

result.userInfoList 本次拉取的 userList

getUserInfo

获取用户的详细信息。



```
const roomEngine = new TUIRoomEngine();
const userList = [];
const userInfo = await roomEngine.getUserInfo({
    userId: 'user_12345',
});
```

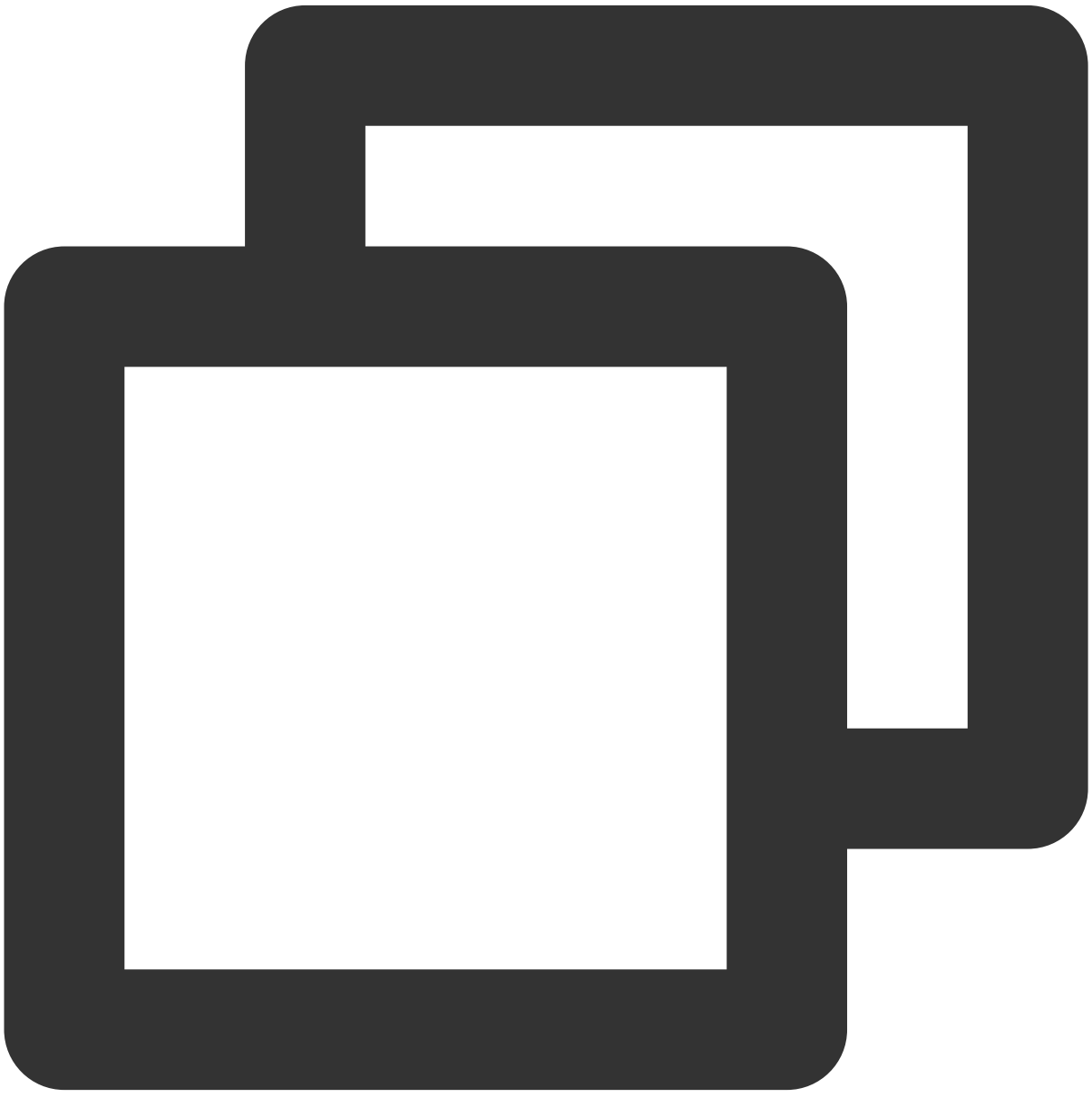
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	根据 userId 获取该用户的详细信息

Returns：*Promise<TUIUserInfo> userInfo*
该接口返回指定用户的用户信息

setLocalVideoView

设置本地流的渲染位置。



```
const roomEngine = new TUIRoomEngine();

// 设置本地摄像头流的播放区域为 id 是 'preview-camera' 的 div 元素
await roomEngine.setLocalVideoView({
```

```
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});

// 设置本地屏幕分享流的播放区域为 id 是 'preview-screen' 的 div 元素
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kScreenStream,
  view: 'preview-screen',
});
```

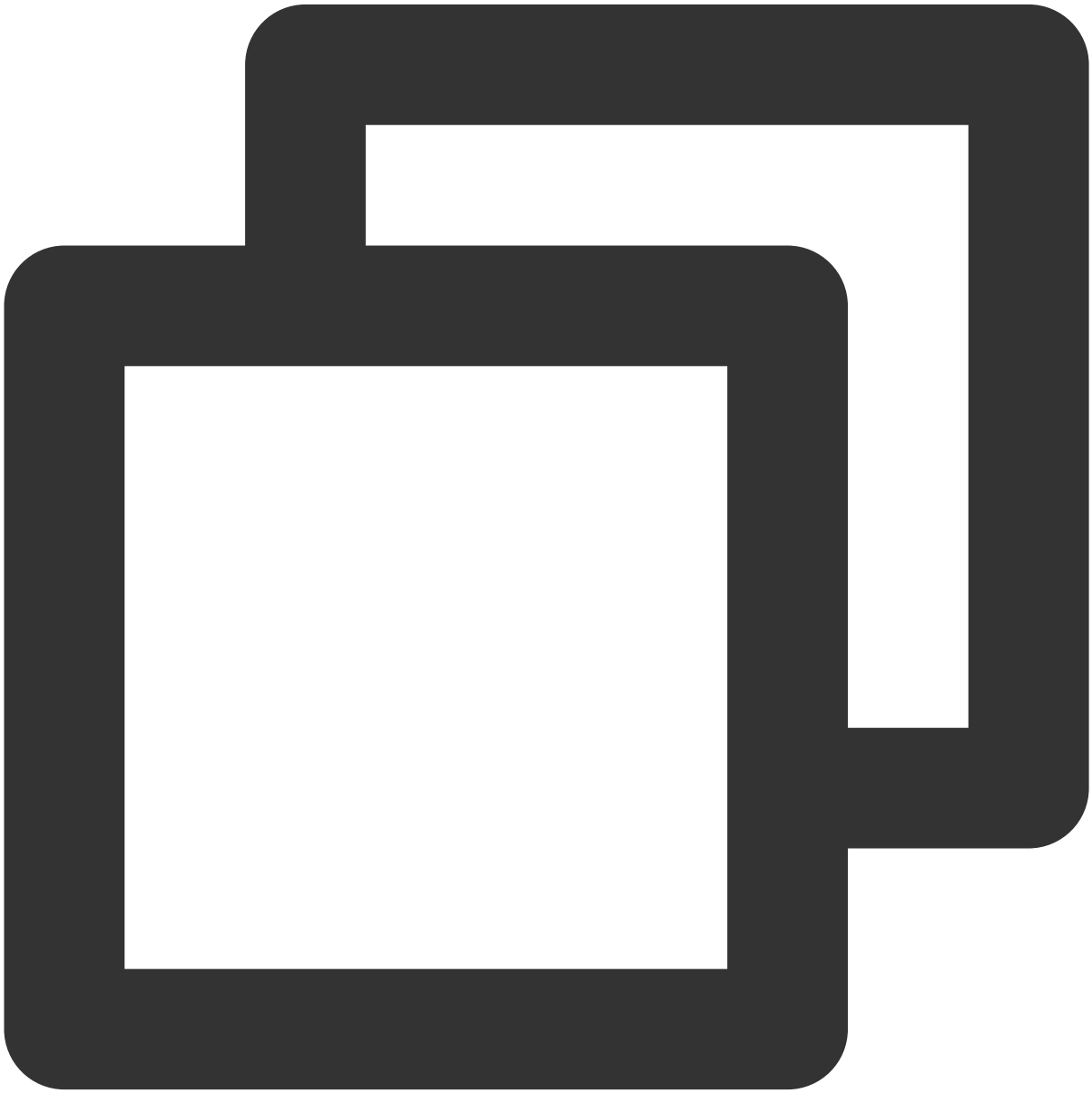
参数：

参数	类型	说明	默认值	含义
streamType	TUIVideoStreamType	必填	-	本地流类型
view	string	必填	-	streamType 对应的流渲染的 div 元素的 id

Returns：*Promise<void>*

openLocalCamera

打开本地摄像头，开始视频流采集。

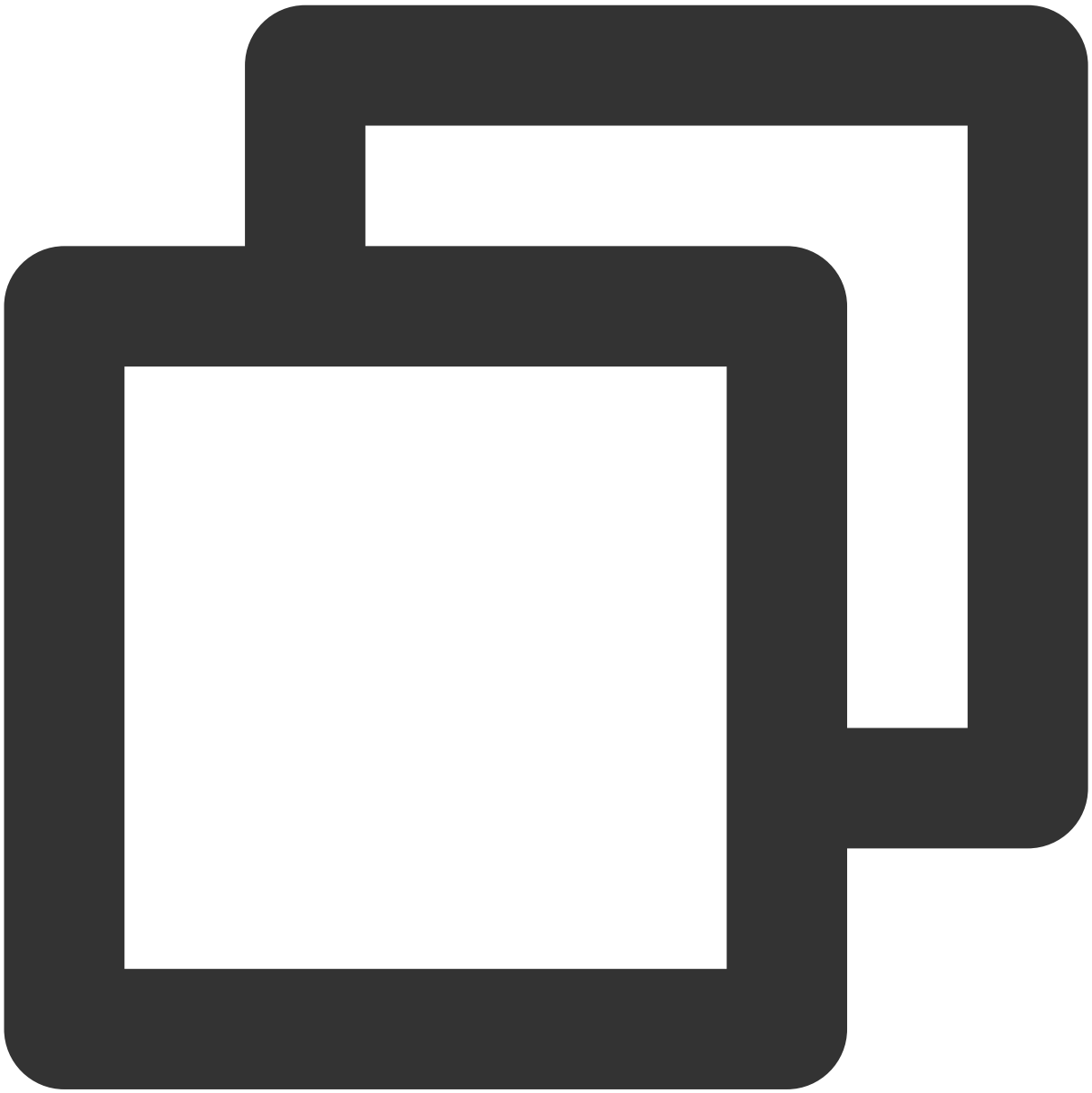


```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalVideoView({
  streamType: TUIVideoStreamType.kCameraStream,
  view: 'preview-camera',
});
await roomEngine.openLocalCamera();
```

Returns：*Promise<void>*

closeLocalCamera

关闭本地摄像头。

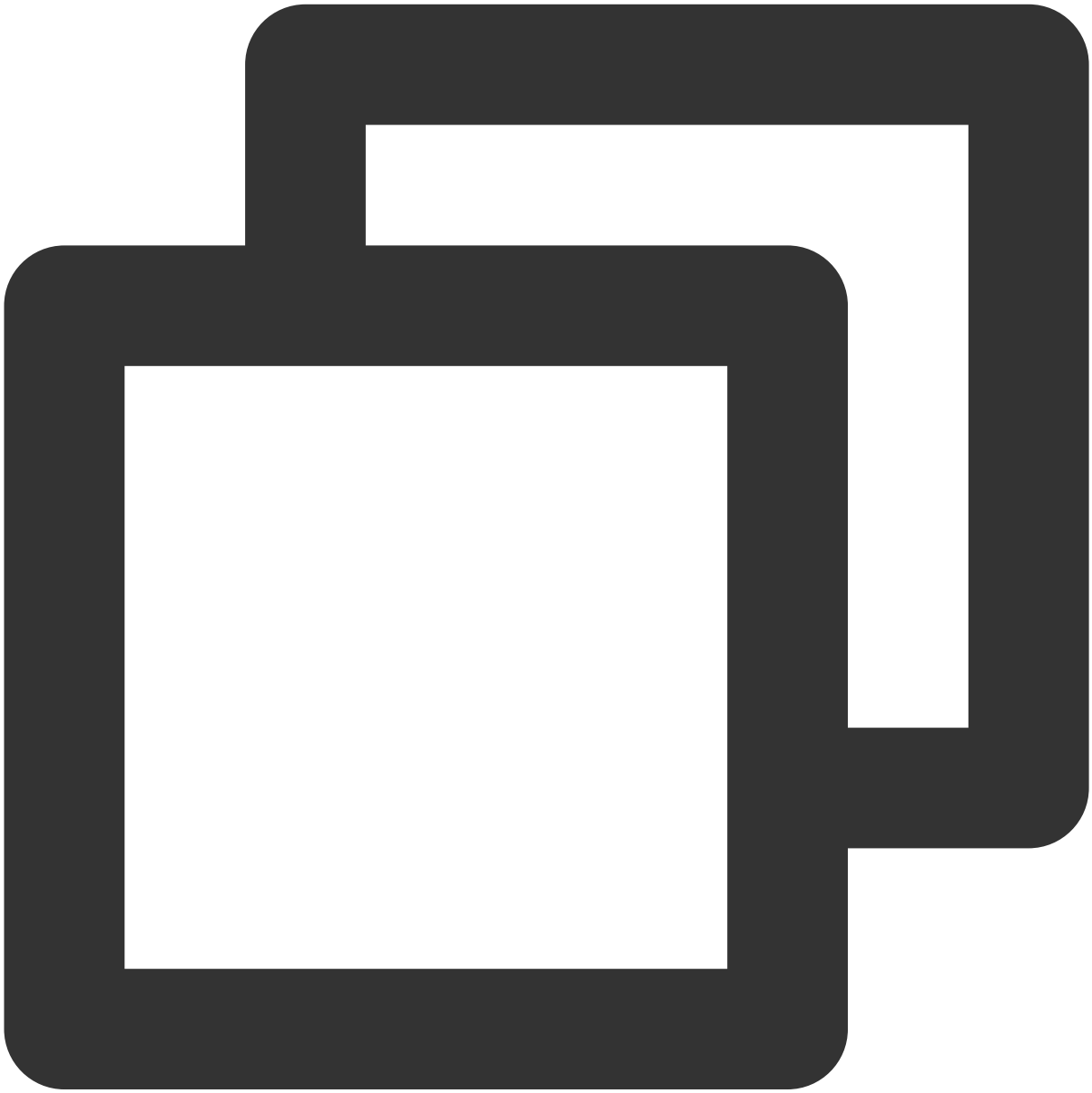


```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeLocalCamera();
```

Returns：*Promise<void>*

openLocalMicrophone

打开本地麦克风, 开始采集音频流。

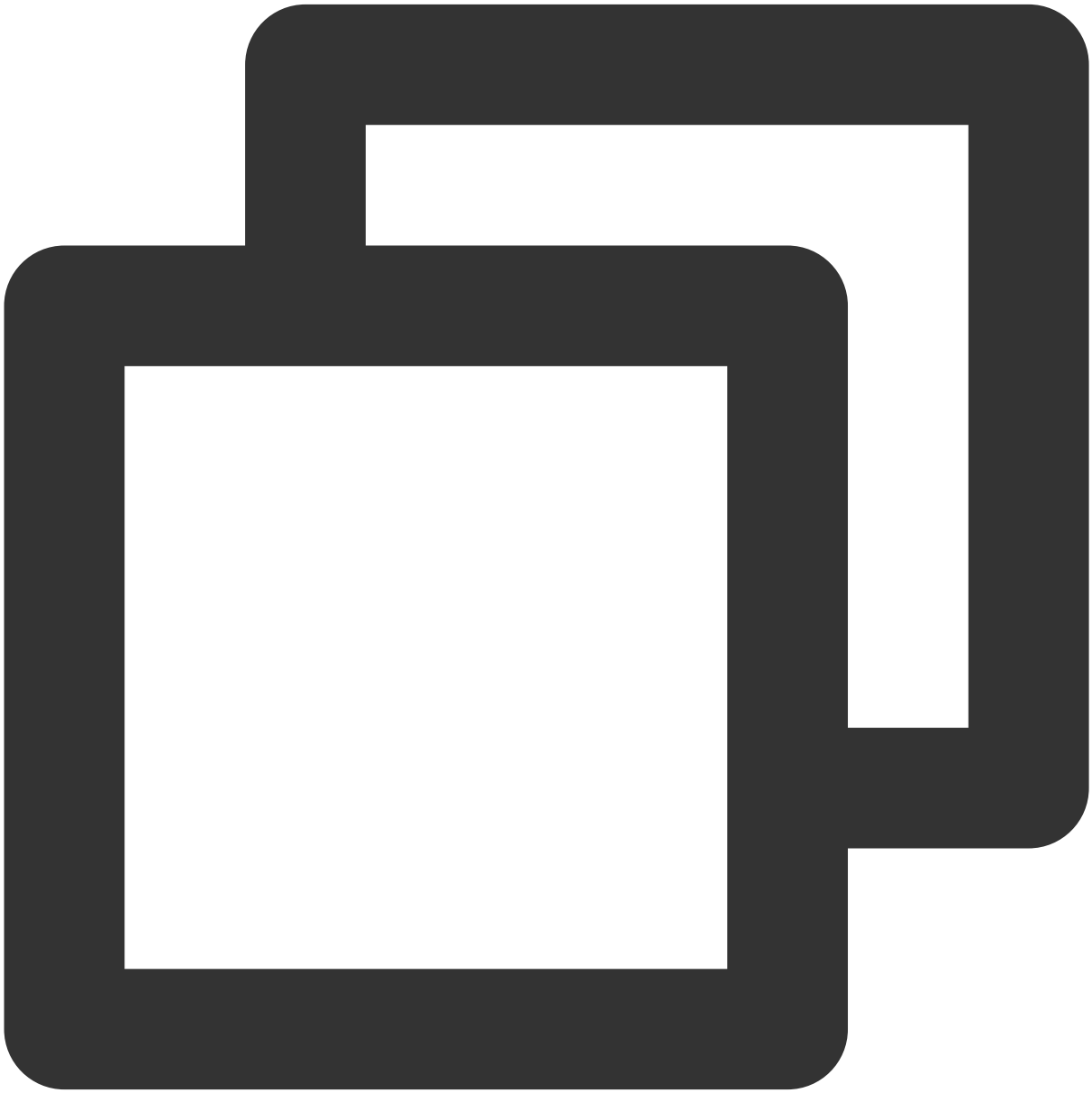


```
const roomEngine = new TUIRoomEngine();
await roomEngine.openLocalMicrophone();
```

Returns：*Promise<void>*

closeLocalMicrophone

关闭本地麦克风。

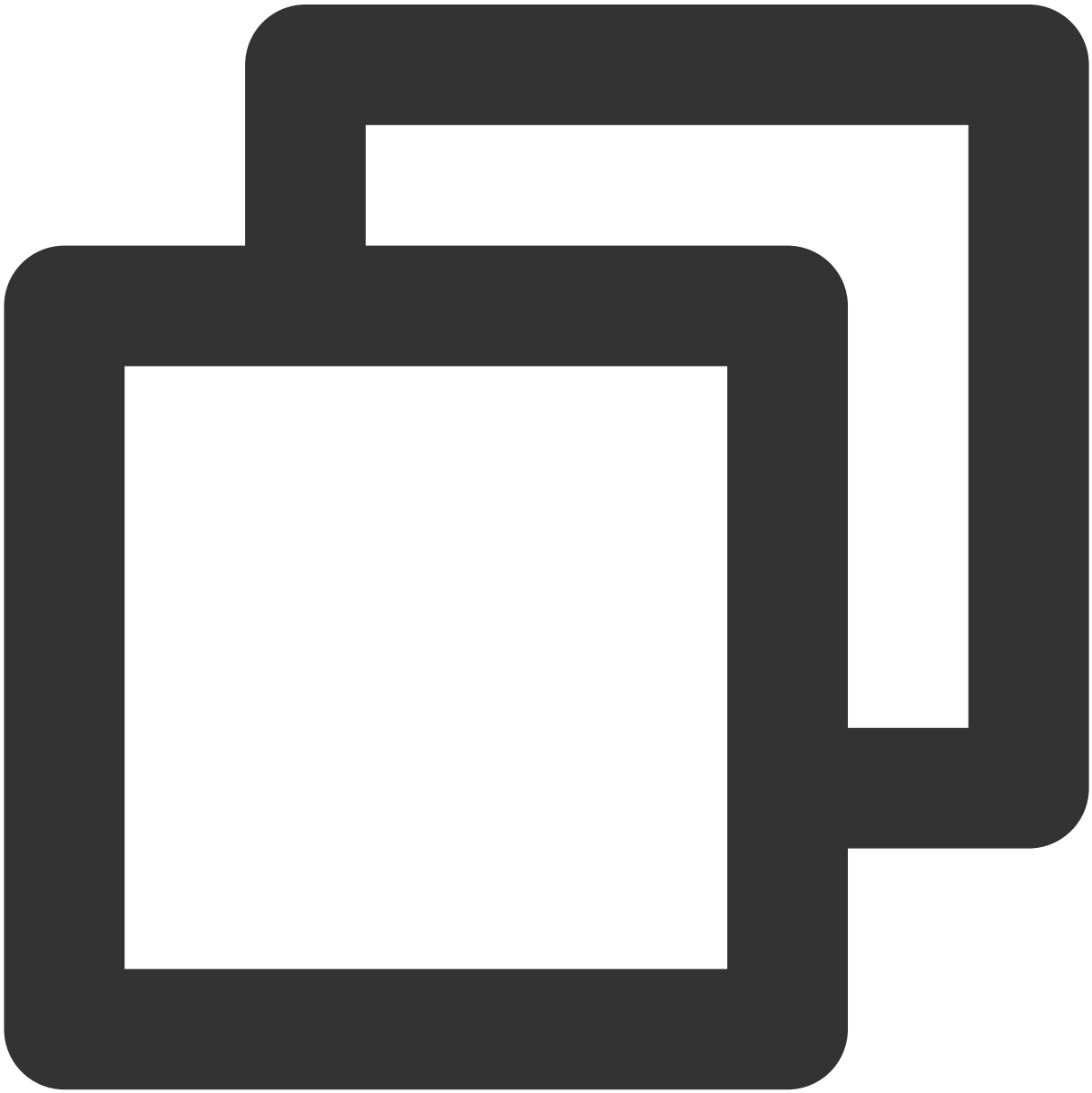


```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeLocalMicrophone();
```

Returns：*Promise<void>*

updateVideoQuality

设置本地视频流的编码参数，默认为 TUIVideoProfile.kVideoQuality_720P。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.updateVideoQuality({
    quality: TUIVideoQuality.kVideoQuality_540p,
});
```

参数：

参数	类型	说明	默认值	含义
quality	TUIVideoQuality	必填	-	清晰 TUIVideoProfile.kVideoQuality_360P

				标清 TUIVideoProfile.kVideoQuality_540P 高清 TUIVideoProfile.kVideoQuality_720P 超清 TUIVideoProfile.kVideoQuality_1080P

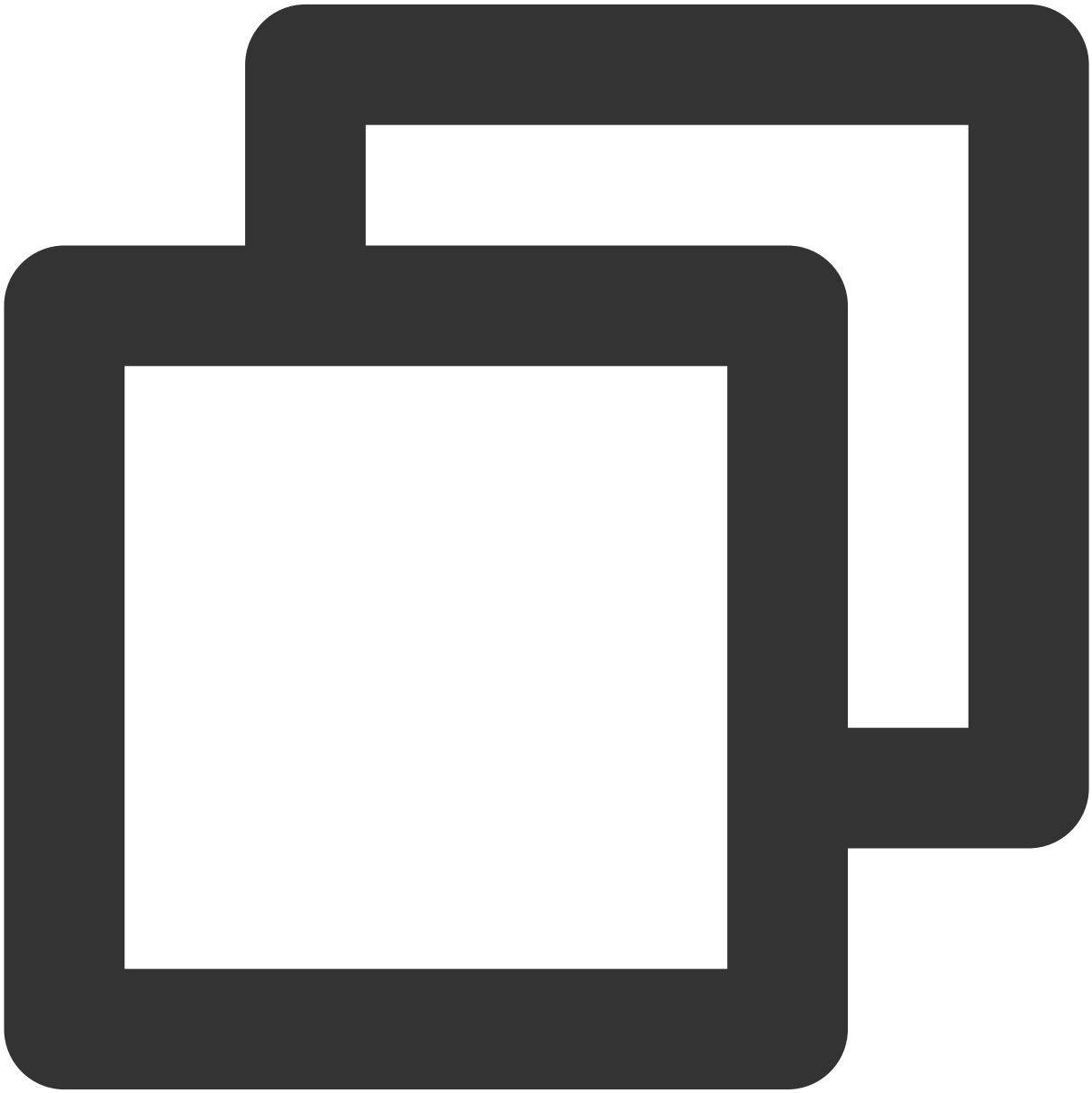
Returns：*Promise<void>*

updateAudioQuality

设置本地音频的参数。

注意：

该方法需要在 openLocalMicrophone 之前进行设置，否则不会生效。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setLocalAudioProfile({
    audioProfile: TUIAudioProfile.kAudioProfileSpeech,
});
```

参数：

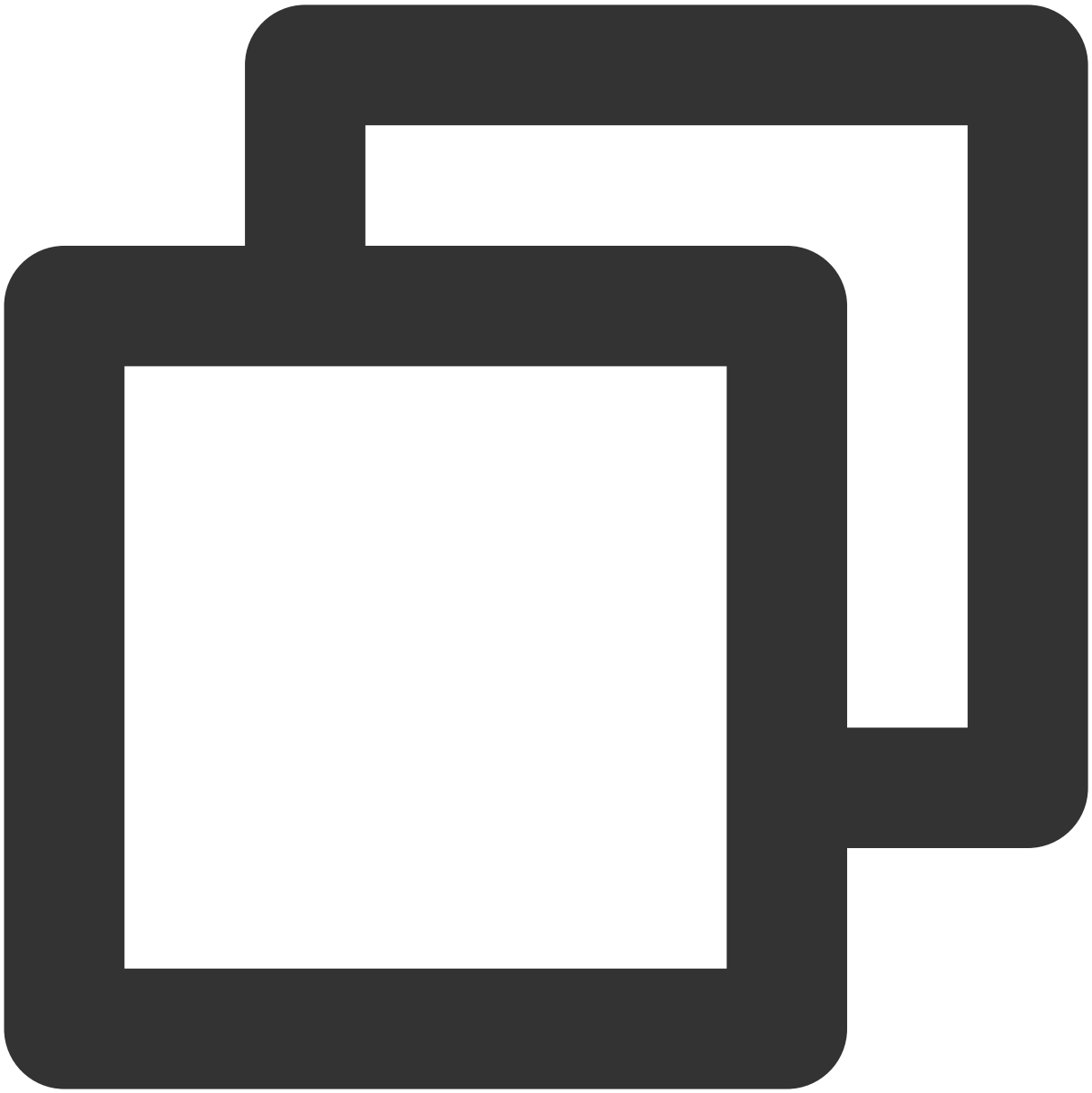
参数	类型	说明	默认值	含义
audioProfile	TUIAudioProfile	必填	-	TUIAudioProfile.kAudioProfileSpeech：语言模式；

				采样率：16k TUIAudioProfile.kAudioProfileDefault：标准模式（或者默认模式）；采样率：48k TUIAudioProfile.kAudioProfileMusic：音乐模式；采样率：48k

Returns：*Promise<void>*

startPushLocalVideo

进房后会默认向远端推本地视频流，该接口用于停止推流后重新向远端推本地视频流。

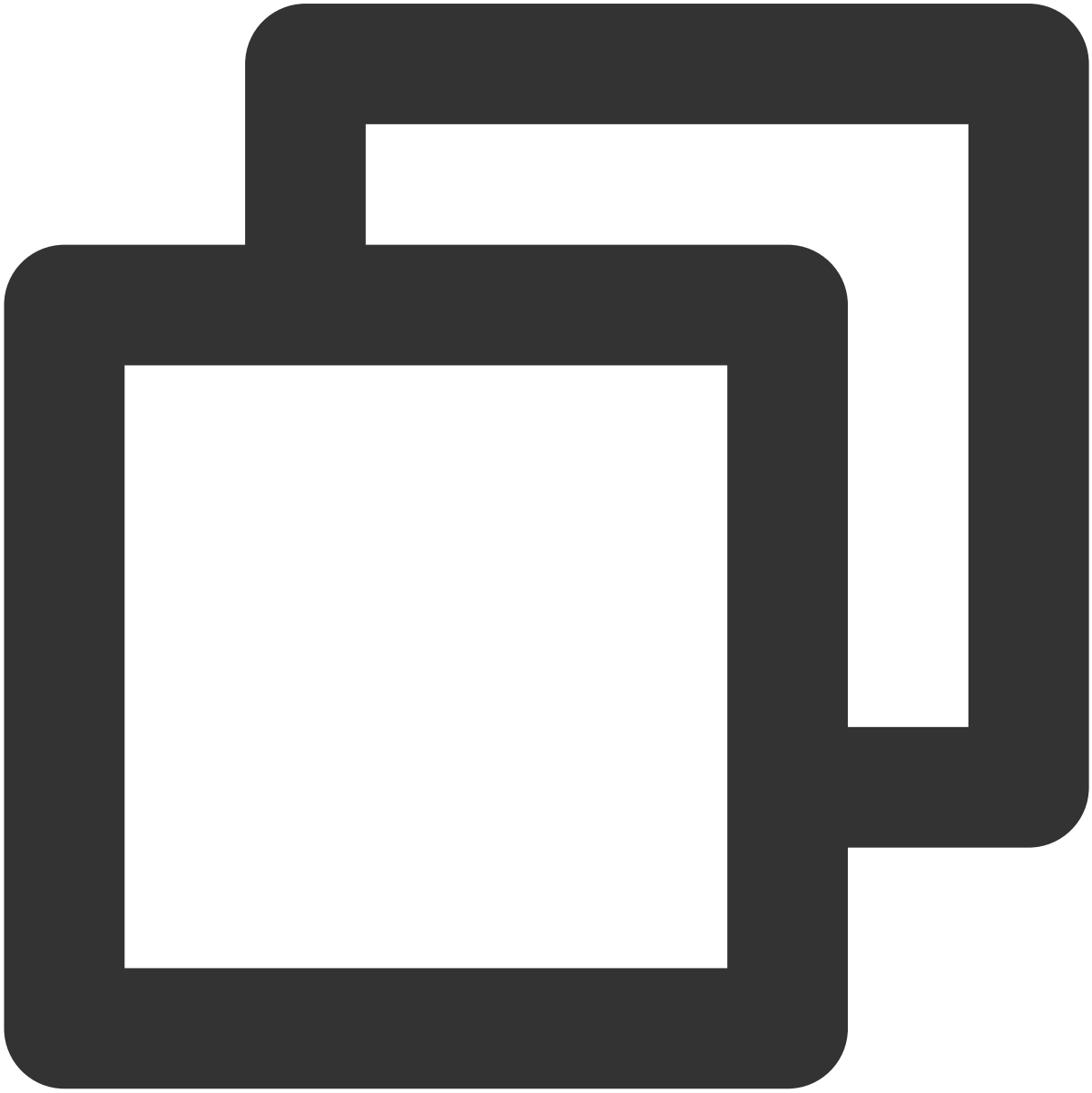


```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPushLocalVideo();
```

Returns：*Promise<void>*

stopPushLocalVideo

停止向远端推本地视频流。

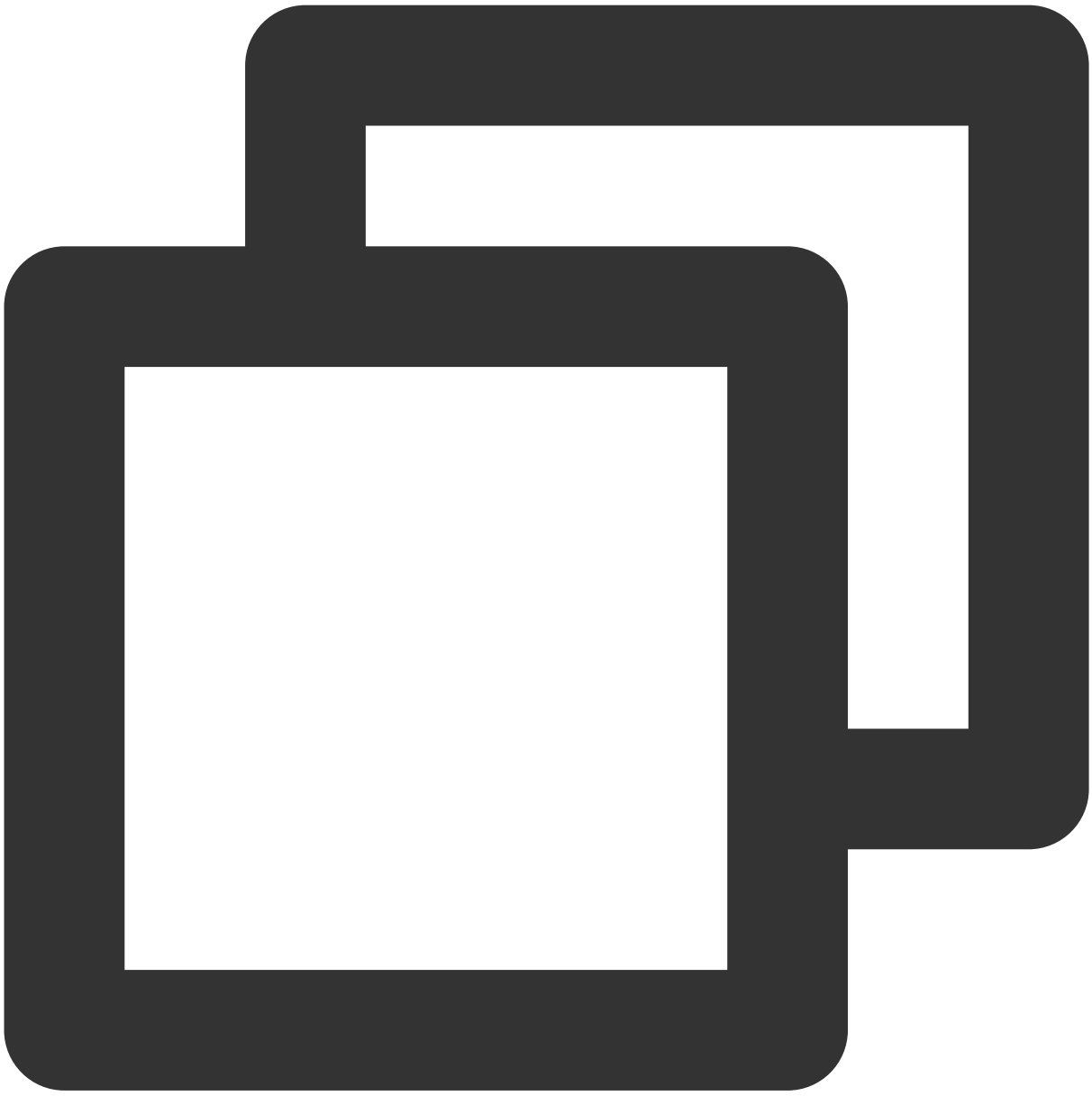


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPushLocalVideo();
```

Returns：*Promise<void>*

startPushLocalAudio

进房后会默认向远端推本地音频流，该接口用于停止推流后重新向远端推本地音频流。

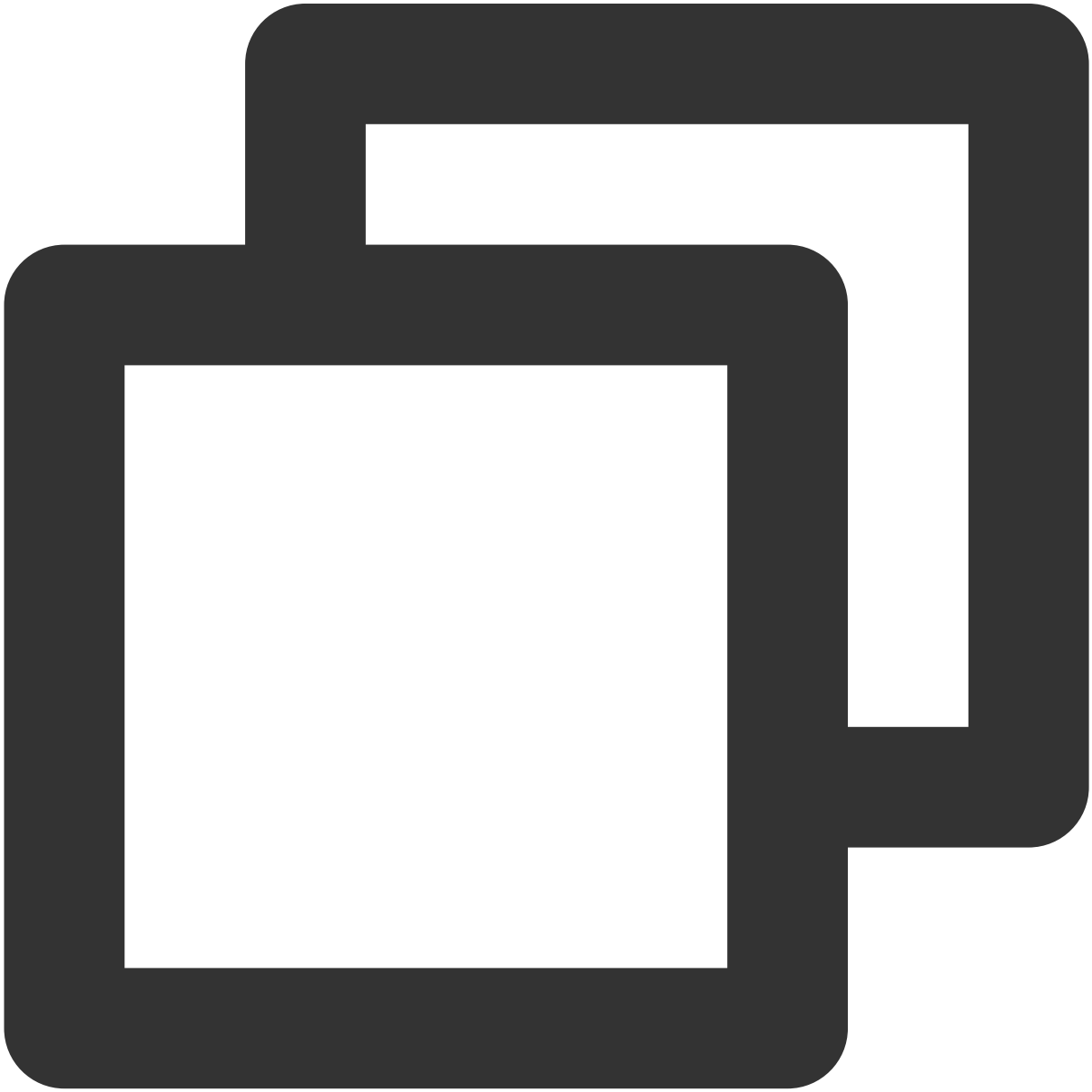


```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPushLocalAudio();
```

Returns：*Promise<void>*

stopPushLocalAudio

停止向远端推本地音频流。

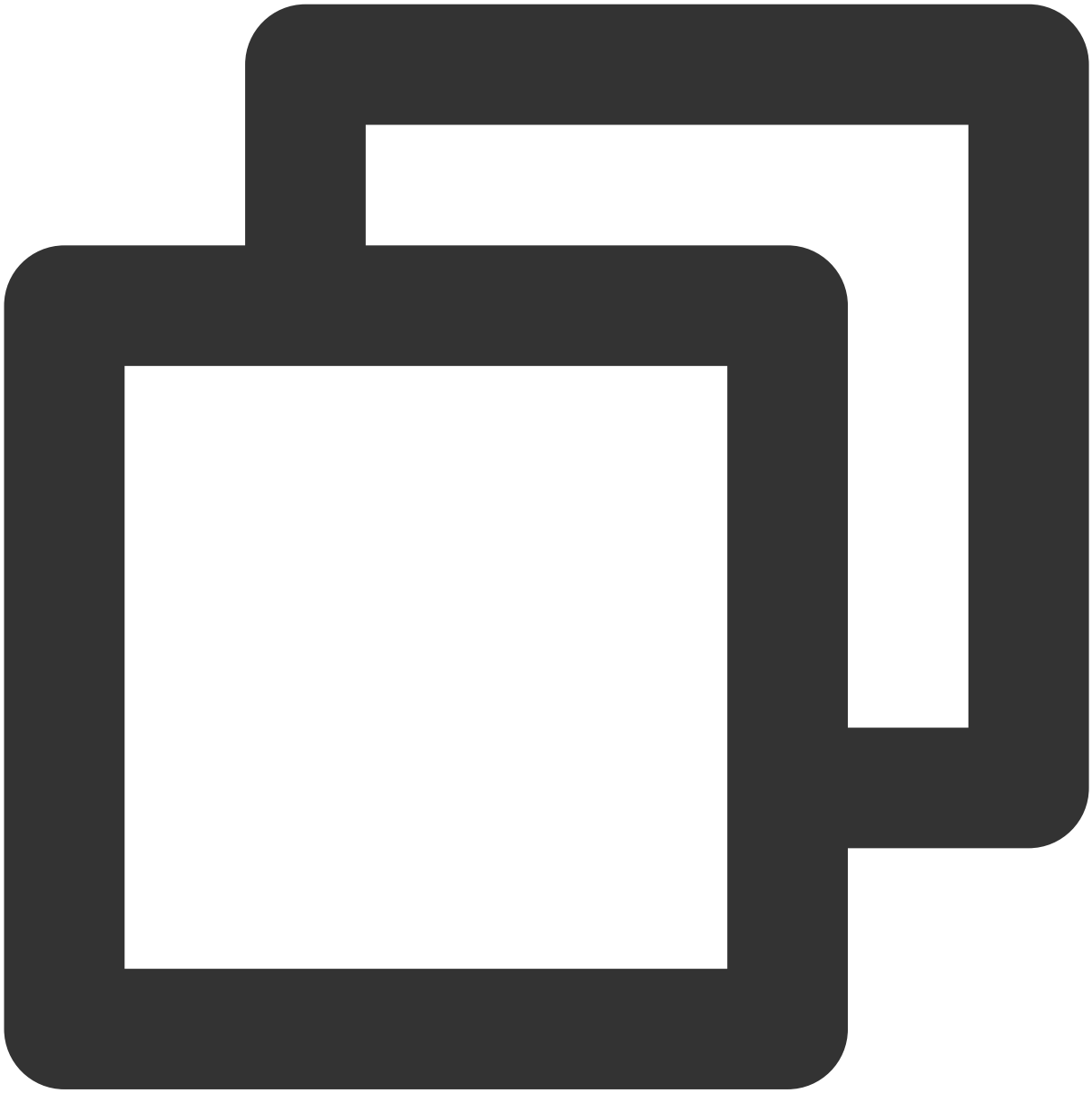


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPushLocalAudio();
```

Returns：*Promise<void>*

setRemoteVideoView

设置远端流渲染的区域。



```
const roomEngine = new TUIRoomEngine();

// 设置远端用户视频流在 id 为 'remote_preview_camera' 的区域播放
await roomEngine.setRemoteVideoView({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
   view: 'remote_preview_camera',
});
// 设置远端用户屏幕分享流在 id 为 'remote_preview_screen' 的区域播放
await roomEngine.setRemoteVideoView({
   userId: 'user_1234',
```

```
    streamType: TUIVideoStreamType.kScreenStream,
    view: 'remote_preview_screen',
});
```

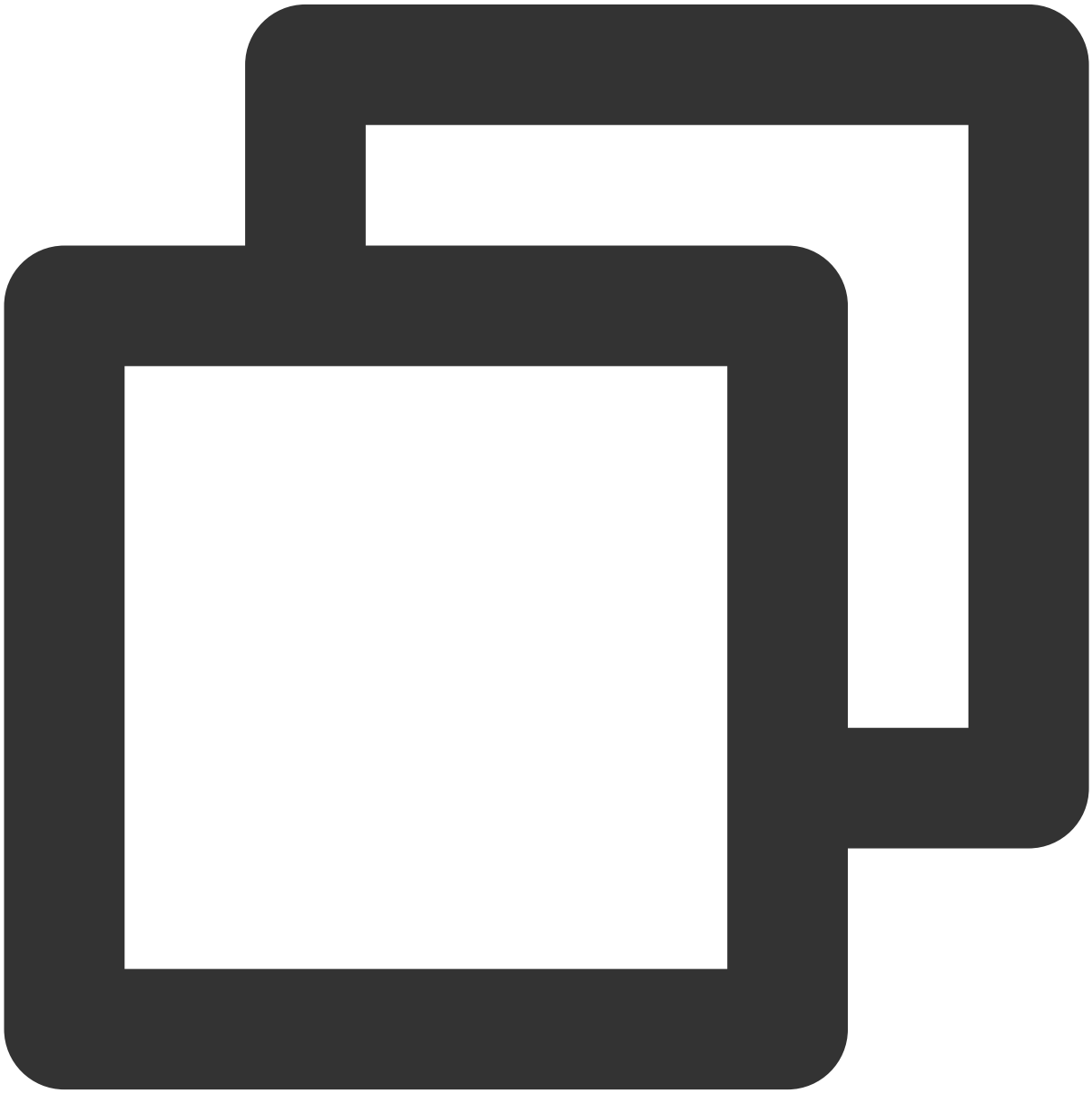
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型
view	string	必填	-	播放远端用户流的 div 元素的 id

Returns：*Promise<void>*

startPlayRemoteVideo

开始播放远端用户视频流。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startPlayRemoteVideo({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
});
```

参数：

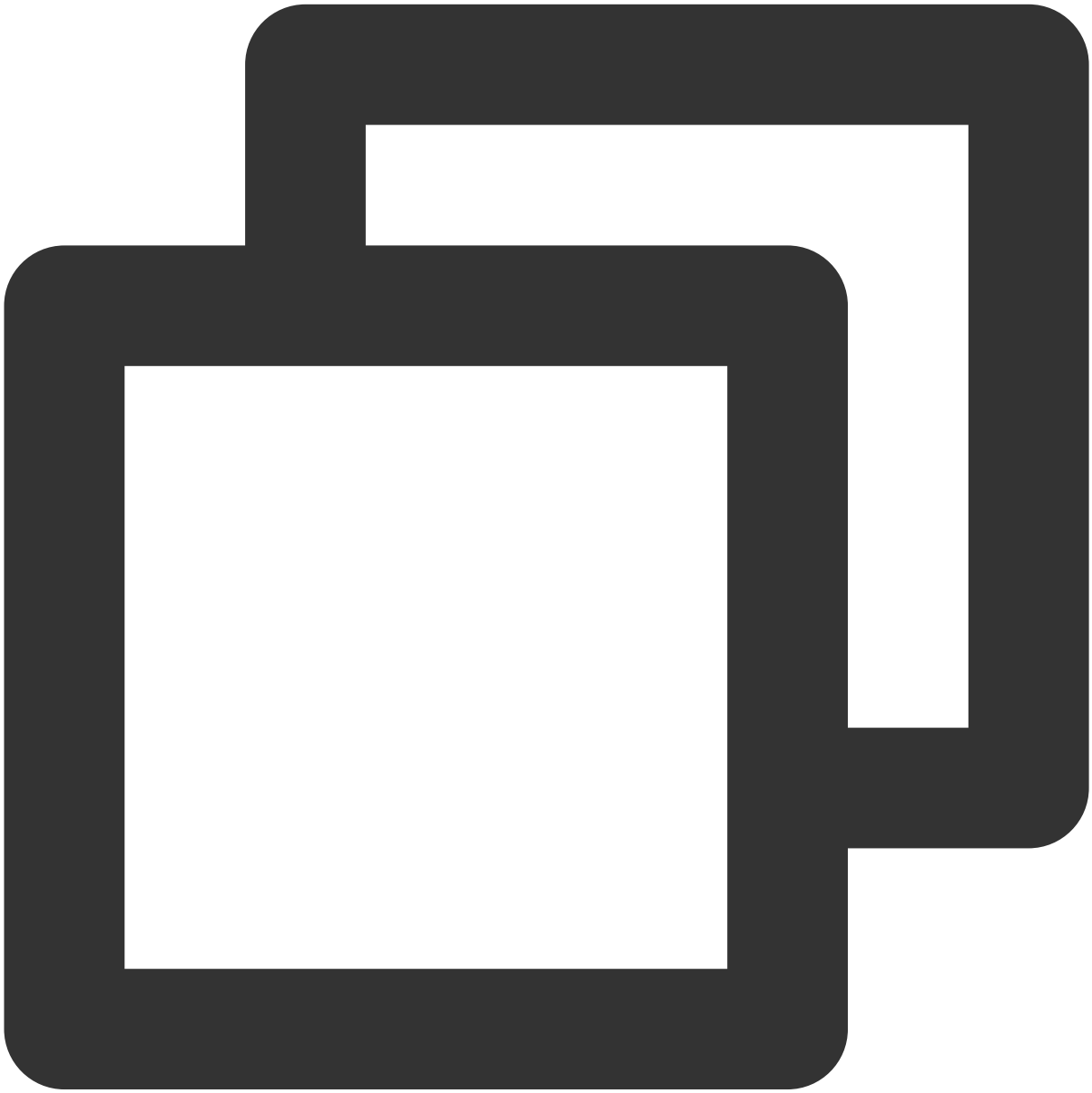
参数	类型	说明	默认值	含义

userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型 TUIVideoStreamType.kCameraStream 视频流 TUIVideoStreamType.kScreenStream 屏幕分享流 TUIVideoStreamType.kCameraStreamLow 低清视频流

Returns：*Promise<void>*

stopPlayRemoteVideo

停止播放远端用户视频流。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopPlayRemoteVideo({
   userId: 'user_1234',
   streamType: TUIVideoStreamType.kCameraStream,
});
```

参数：

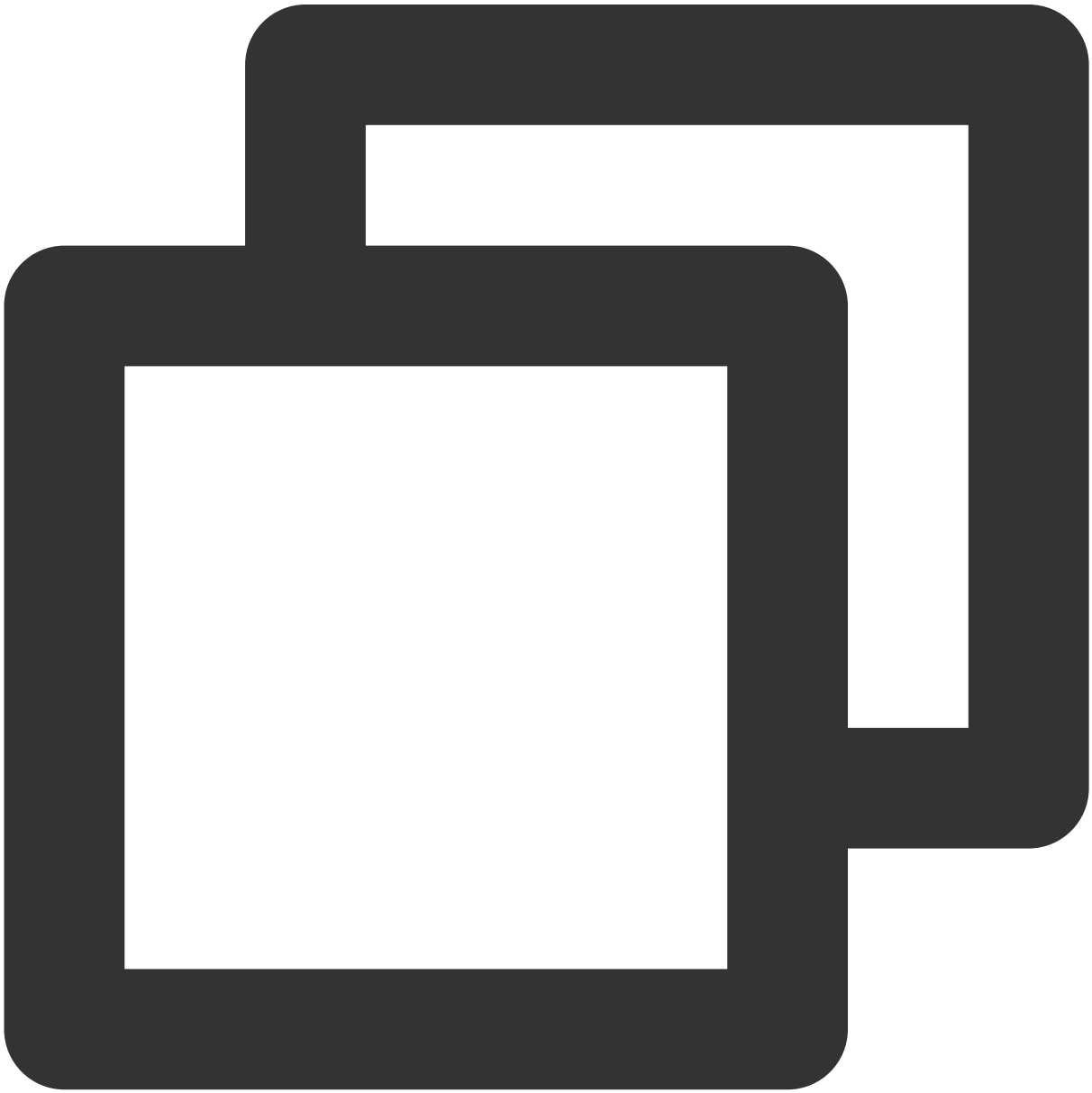
参数	类型	说明	默认值	含义

userId	string	必填	-	用户 ID
streamType	TUIVideoStreamType	必填	-	用户流类型 TUIVideoStreamType.kCameraStream 视频流 TUIVideoStreamType.kScreenStream 屏幕分享流 TUIVideoStreamType.kCameraStreamLow 低清视频流

Returns：*Promise<void>*

muteRemoteAudioStream

停止远端用户的音频流。

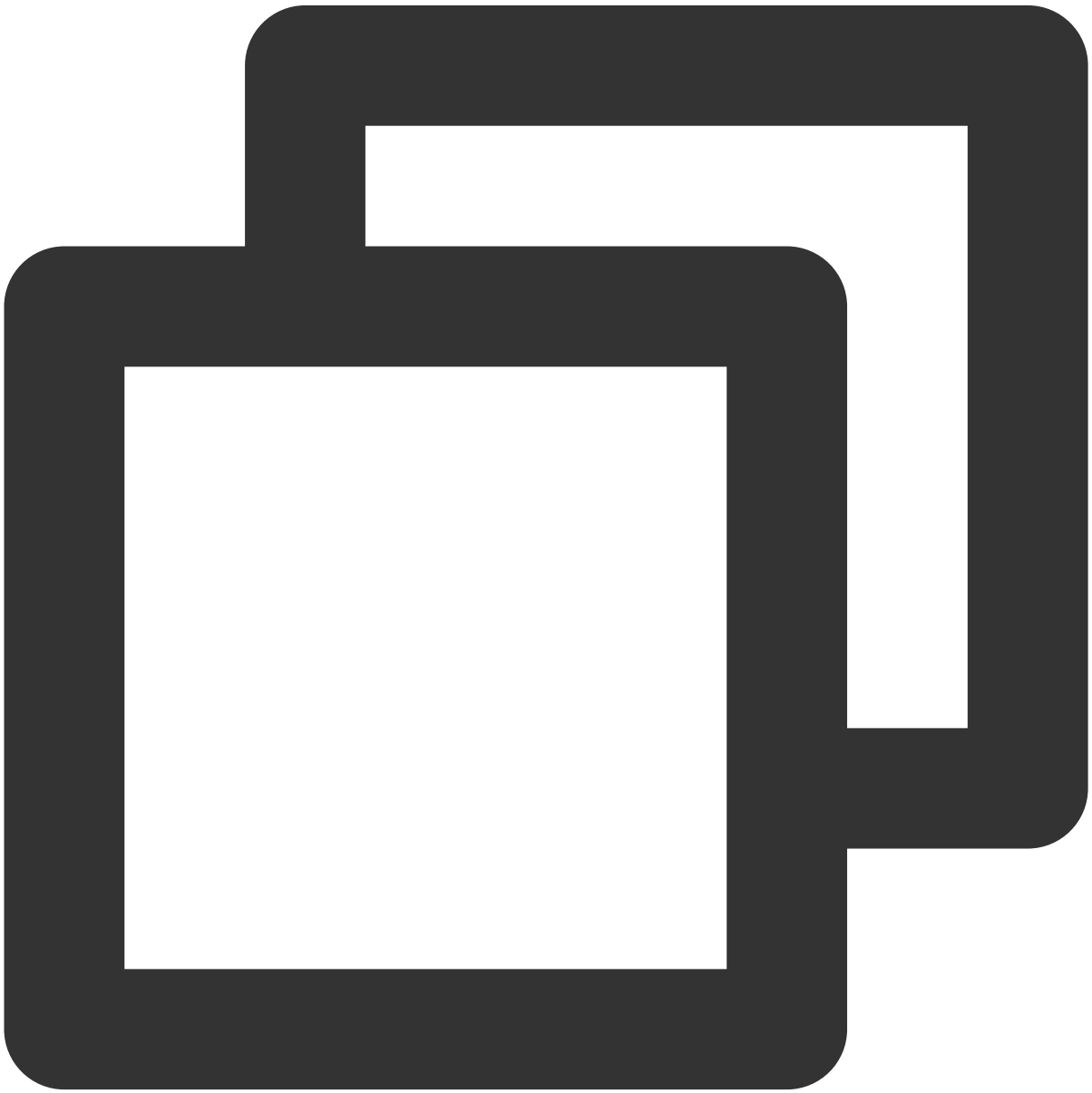


```
const roomEngine = new TUIRoomEngine();
await roomEngine.muteRemoteAudioStream({
  userId: 'user_1234',
  isMute: true,
});
```

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
isMute	boolean	必填	-	是否停止远端用户的音频

openRemoteDeviceByAdmin

请求远端用户打开媒体设备。



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.openRemoteDeviceByAdmin({
    userId: 'user_1234',
    device: TUIMediaDevice.kMicrophone  //请求的设备为麦克风
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
```

```
      switch (requestCallbackType) {
        case TUIRequestCallbackType.kRequestAccepted:
          // 请求被接受
          break;
        case TUIRequestCallbackType.kRequestRejected:
          // 请求被拒绝
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
      }
    },
  });
```

参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）
timeout	number	必填	-	超时时间。若 timeout 设置为 0 ，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string> requestId*

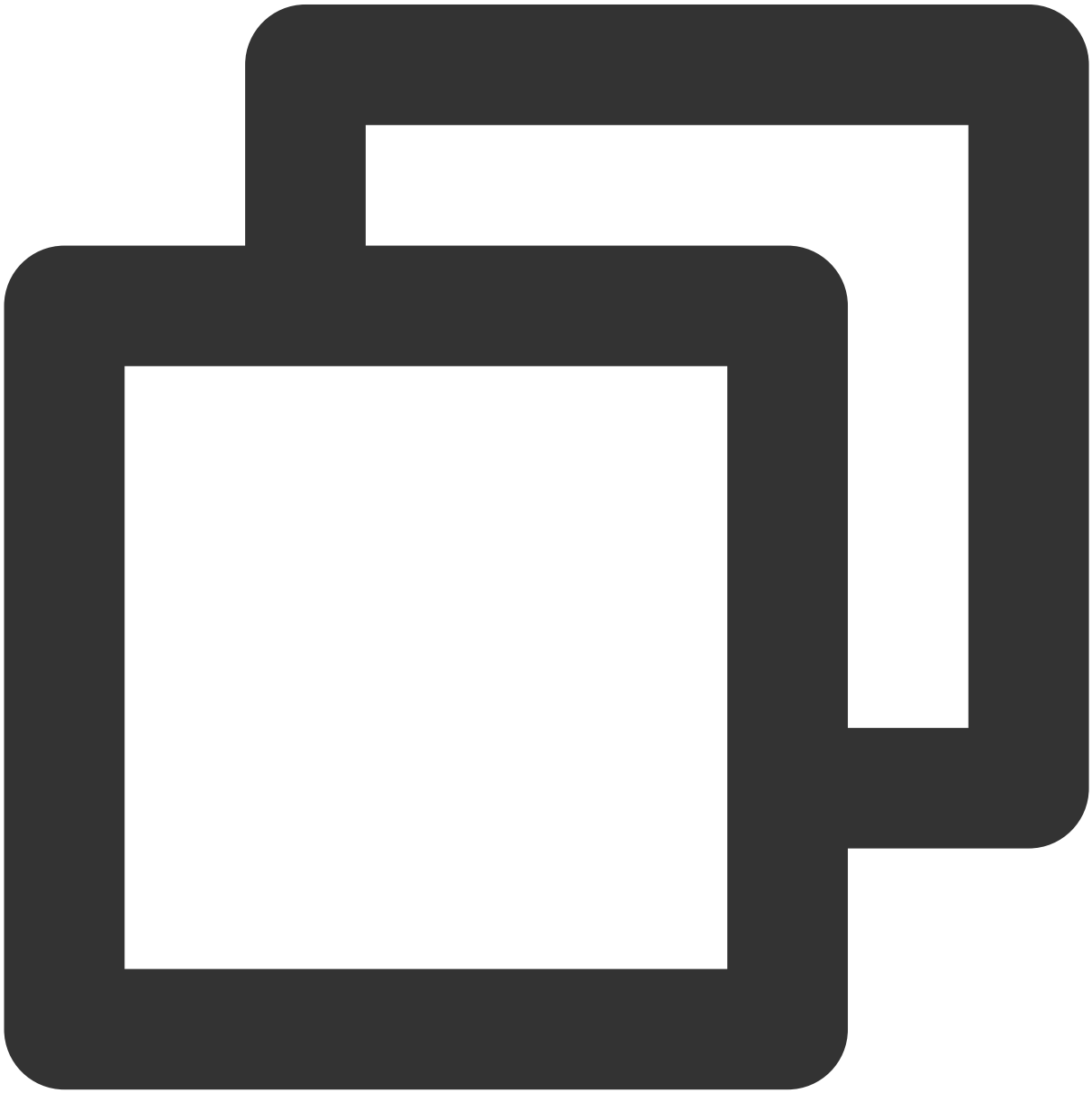
该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

applyToAdminToOpenLocalDevice

参会者向主持人申请打开设备。



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.applyToAdminToOpenLocalDevice({
    device: TUIMediaDevice.kMicrophone  //请求的设备为麦克风
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
     switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
         // 请求被接受
        break;
       case TUIRequestCallbackType.kRequestRejected:
         // 请求被拒绝
```

```
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
      }
    },
});
```

参数：

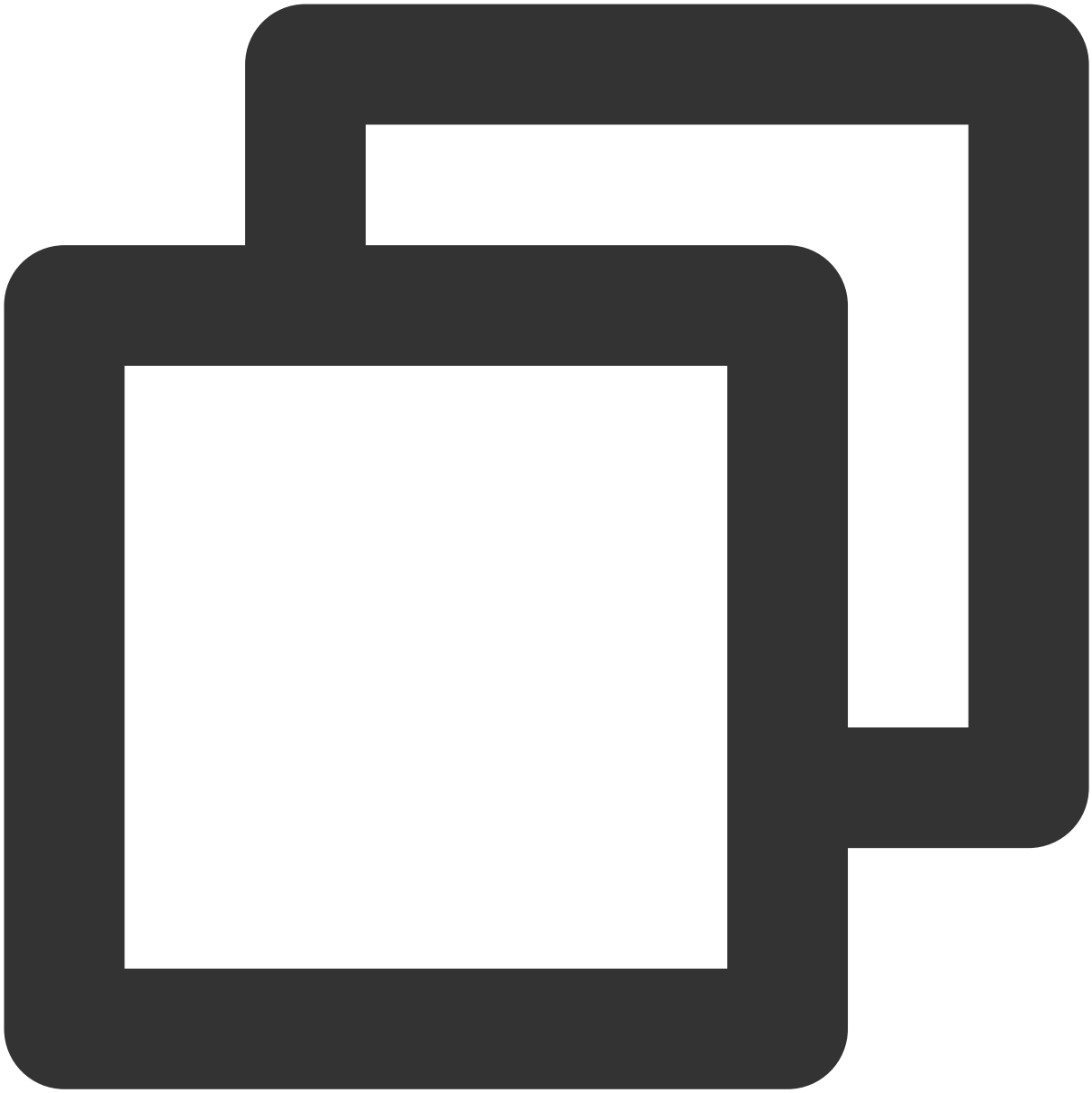
参数	类型	说明	默认值	含义
device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）
timeout	number	必填	-	超时时间。若 timeout 设置为 0，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string> requestId*

该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

closeRemoteDeviceByAdmin

关闭远端用户媒体设备。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.closeRemoteDeviceByAdmin({
   userId: 'user_1234',
   device: TUIMediaDevice.kMicrophone, //关闭麦克风
 });
```

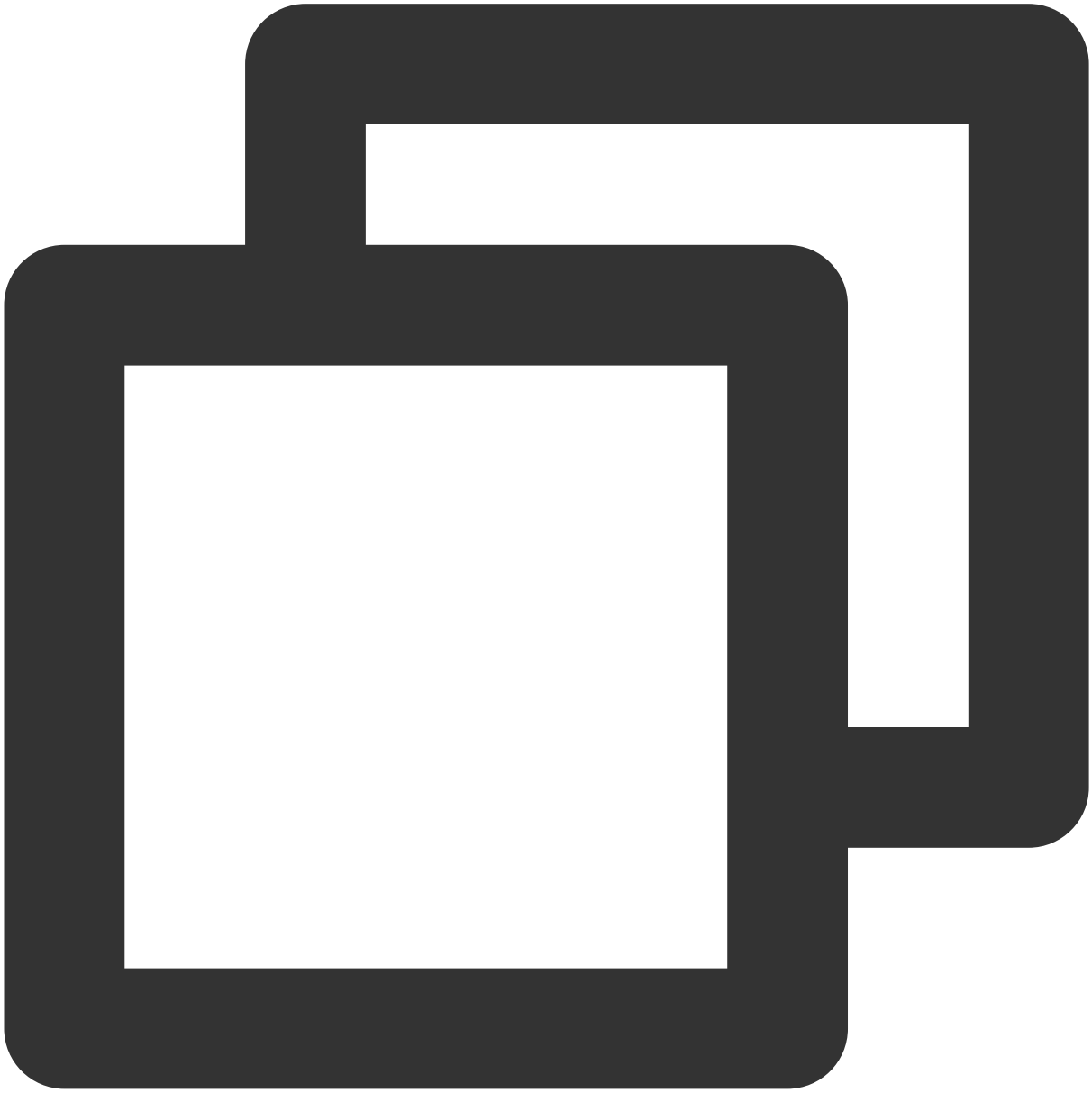
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID

device	TUIMediaDevice	必填	-	媒体设备类型（摄像头/麦克风/屏幕共享）

cancelRequest

取消已经发出的请求。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.cancelRequest({
   requestId: '',    // 请使用实际 requestId
});
```

参数：

参数	类型	说明	默认值	含义
requestId	string	必填	-	请求 ID

Returns：*Promise<string>* requestId

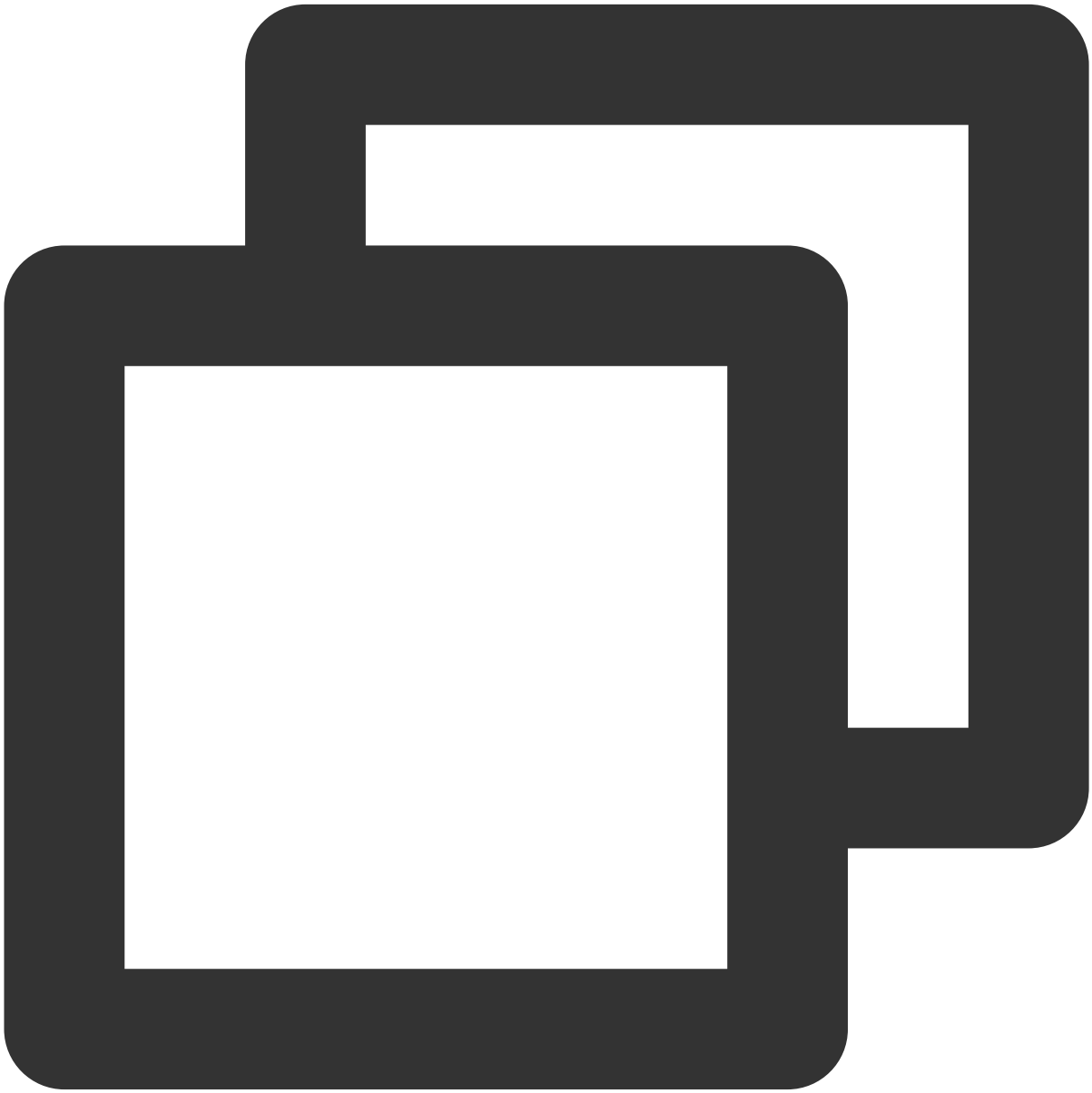
该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

responseRemoteRequest

回复远端用户的请求。



```
const roomEngine = new TUIRoomEngine();
// 同意远端的请求
await roomEngine.responseRemoteRequest({
 requestId: '',     // 请使用实际 requestId
 agree: true,
});
// 拒绝远端的请求
await roomEngine.responseRemoteRequest({
 requestId: '',     // 请使用实际 requestId
 agree: false,
});
```

参数：

参数	类型	说明	默认值	含义
requestId	string	必填	-	请求 ID
agree	boolean	必填	-	是否同意

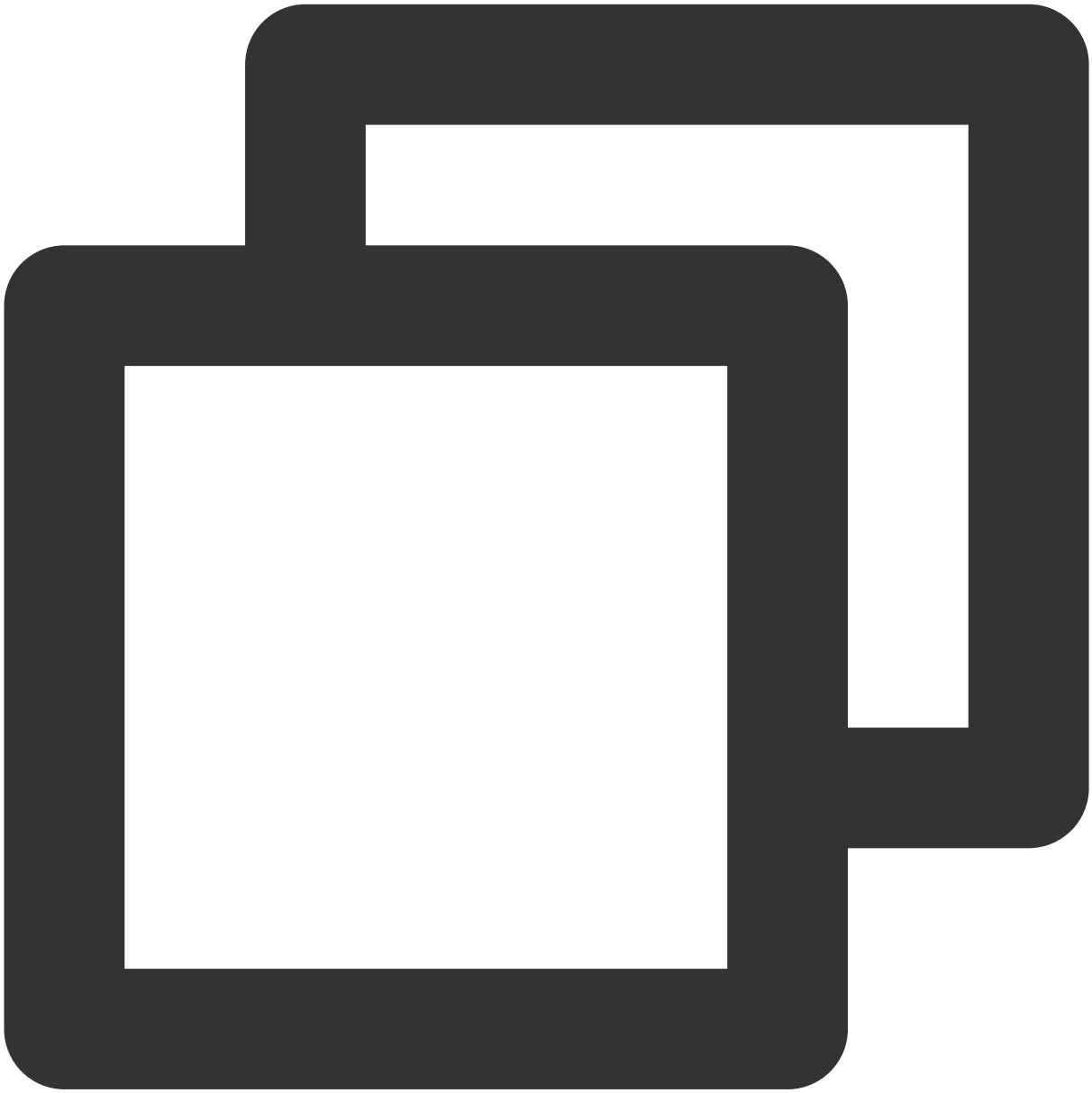
Returns：*Promise<void>*

说明：

v1.0.2 及其以上版本该接口接收的 requestId 参数类型为 string；v1.0.0 及 v1.0.1 版本该接口接收的 requestId 参数类型为 number； disableDeviceForAllUserByAdmin

disableDeviceForAllUserByAdmin

禁止/允许所有用户打开媒体设备（此接口对于房主和管理员无效）。



```
// Example 1: 禁止所有用户打开麦克风
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: true,
})
// Example 2: 允许所有用户打开麦克风
await roomEngine.disableDeviceForAllUserByAdmin({
  device: TUIMediaDevice.kMicrophone,
  isDisable: false,
})
```

版权所有：腾讯云计算（北京）有限责任公司

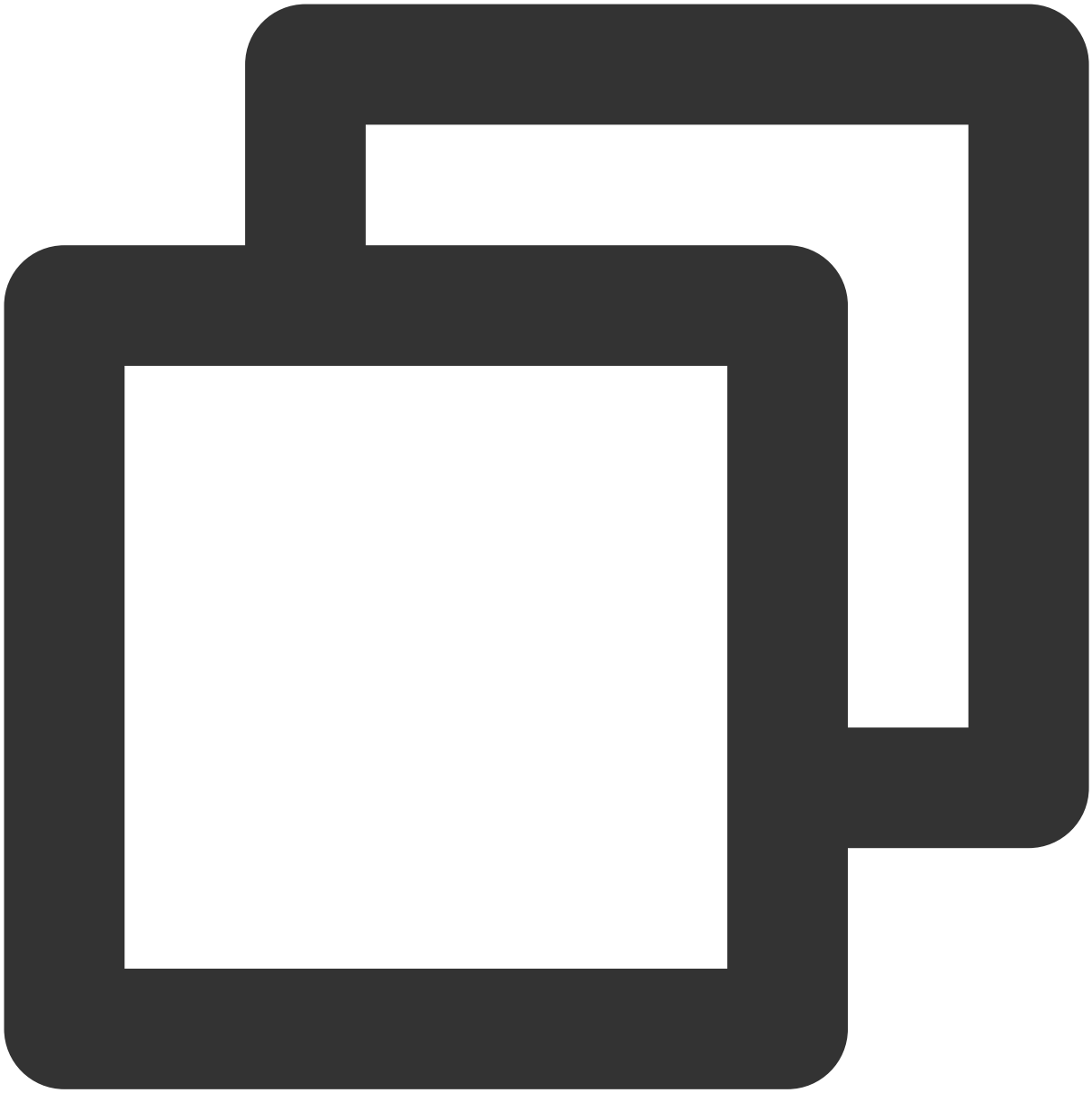
参数：

参数	类型	说明	默认值	含义
device	TUIMediaDevice	必填	-	被禁用的媒体设备类型（摄像头/麦克风/屏幕共享）
isDisable	boolean	必填	-	是否被禁止

Returns：*Promise<void>*

disableSendingMessageForAllUser

所有用户的是否允许发消息（此接口对于房主和管理员无效）。



```
await roomEngine.disableSendingMessageForAllUser({
  isDisable: true,
});
```

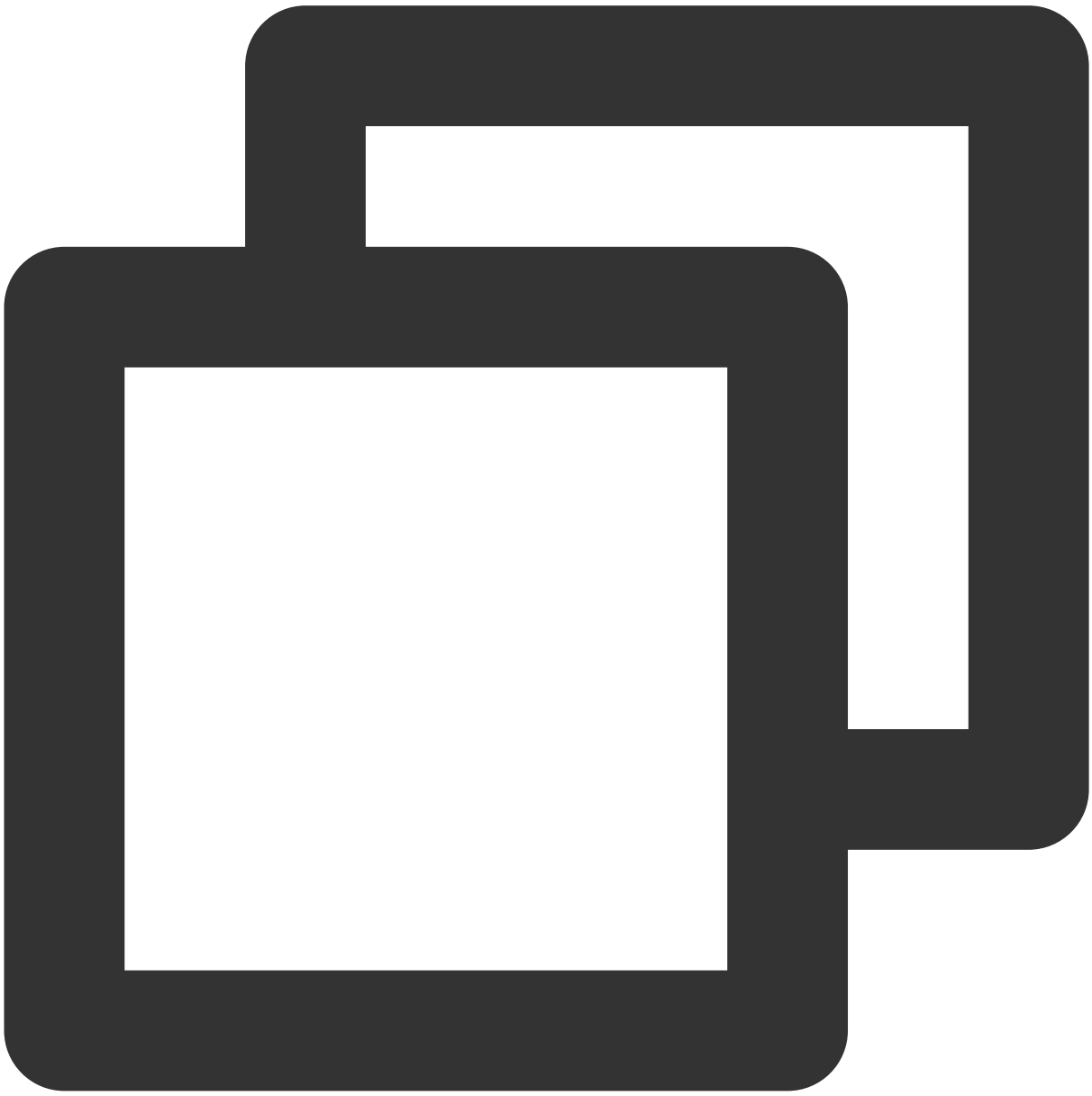
参数：

参数	类型	说明	默认值	含义
isDisable	boolean	必填	-	是否被禁用

Returns：*Promise<void>*

disableSendingMessageByAdmin

特定用户是否被允许发消息。



```
await roomEngine.disableSendingMessageByAdmin({
  userId: 'user_1234',
  isDisable: true,
});
```

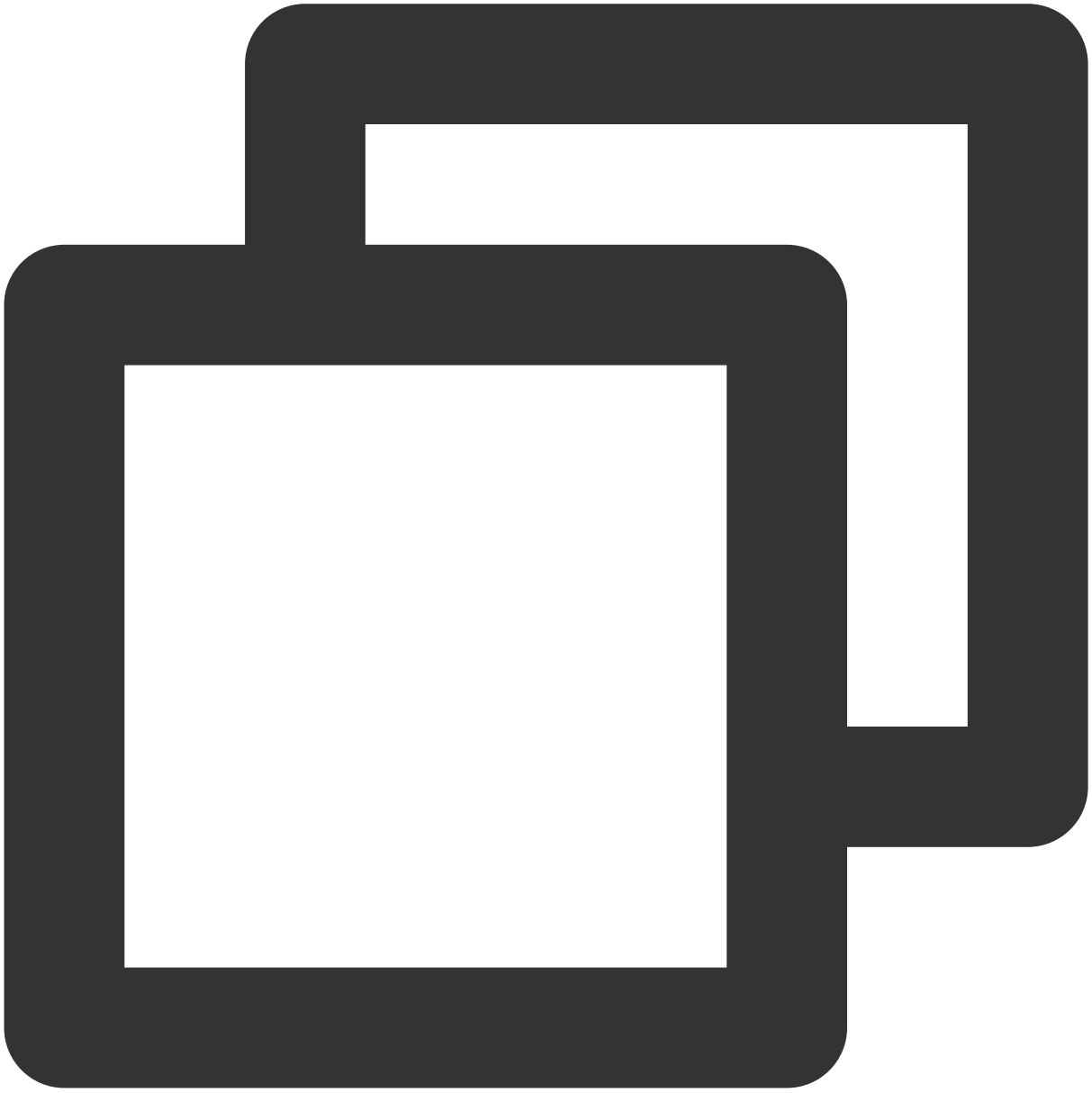
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
isDisable	boolean	必填	-	是否被禁用

Returns：*Promise<void>*

changeUserRole

改变用户的角色（仅主持人可调用该接口）。



```
const roomEngine = new TUIRoomEngine();

// 将房间移交给用户 user_1234
await roomEngine.changeUserRole({
   userId: 'user_1234',
   role: TUIRole.kRoomOwner,
});

// 将用户 user_1234 设置为房间管理员
await roomEngine.changeUserRole({
  userId: 'user_1234',
  userRole: TUIRole.kAdministrator,
});
```

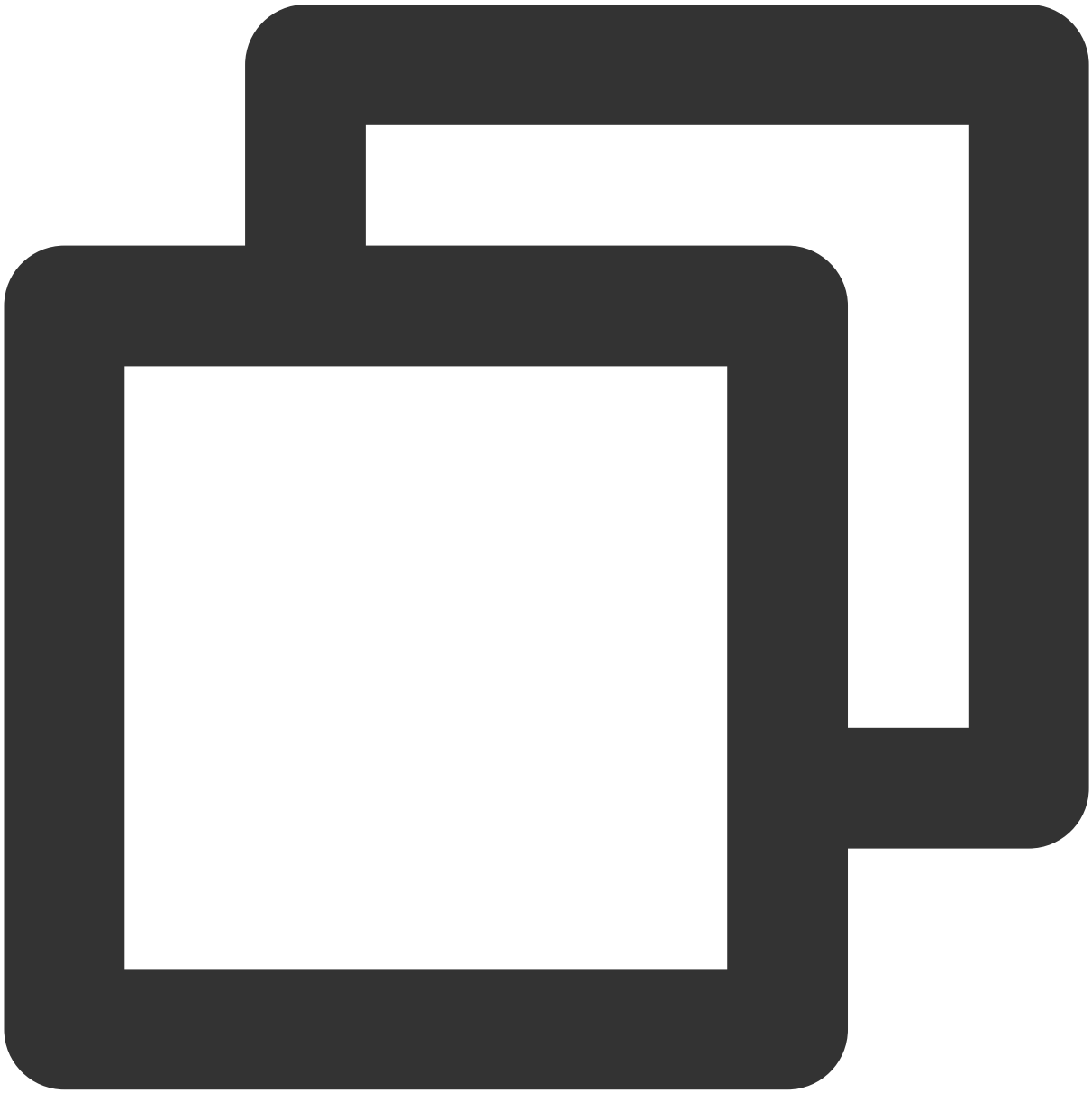
参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID
userRole	TUIRole	必填	-	用户角色 主持人 TUIRole.kRoomOwner 管理员 TUIRole.kAdministrator 普通成员 TUIRole.kGeneralUser

Returns：*Promise<void>*

kickRemoteUserOutOfRoom

将用户踢出房间（仅主持人和管理员可调用该接口）。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.kickRemoteUserOutOfRoom({
   userId: 'user_1234',
});
```

参数：

参数	类型	说明	默认值	含义
userId	string	必填	-	用户 ID

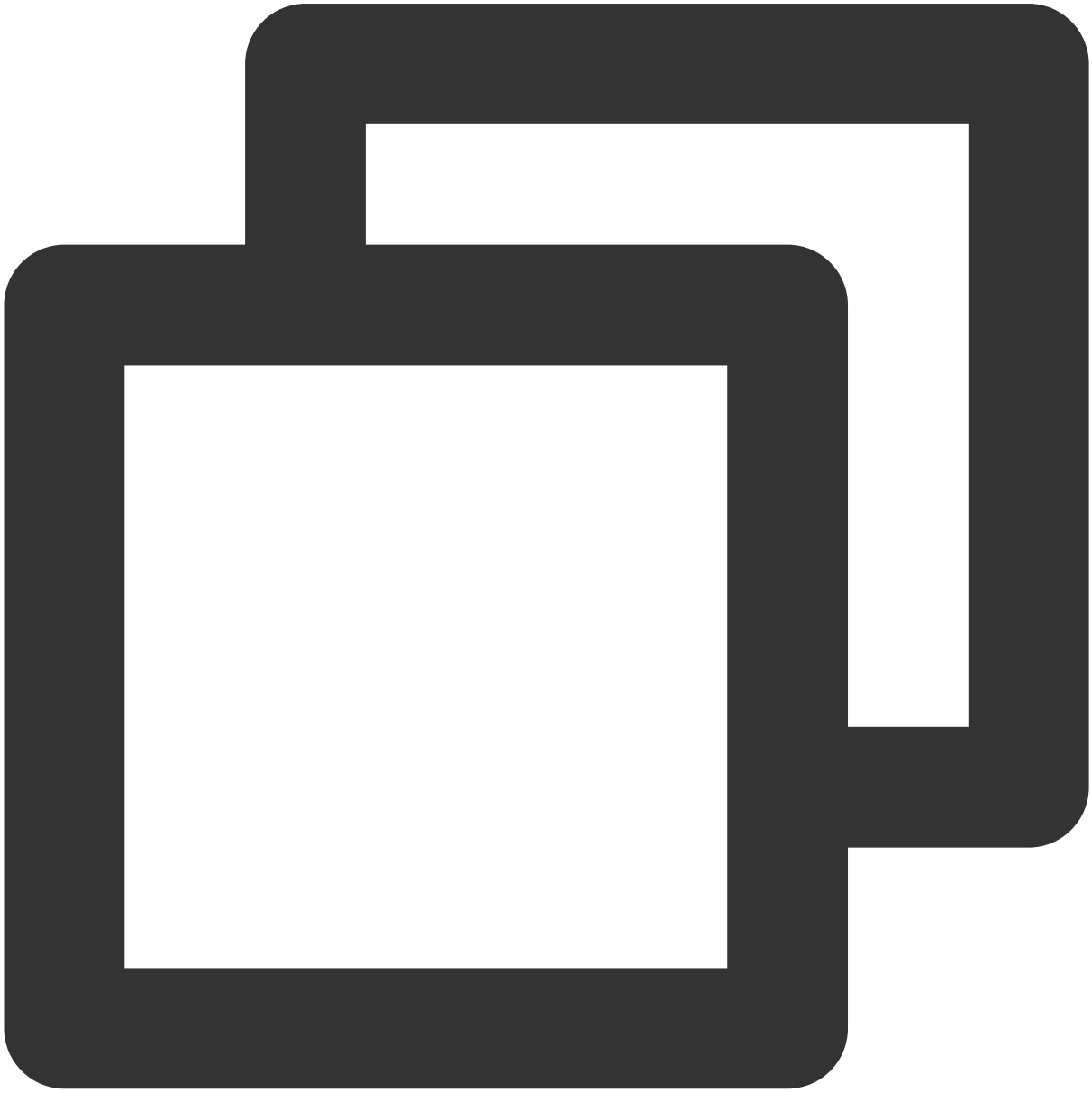
Returns：*Promise<void>*

setMaxSeatCount

设置房间麦位最大值

roomType 为 TUIRoomType.kConference (教育及会议场景) 时，maxSeatCount 值不做限制；

roomType 为 TUIRoomType.kLivingRoom (直播场景) 时，maxSeatCount 最大限制为 16；



```
const roomEngine = new TUIRoomEngine();
await roomEngine.createRoom({ roomId: '12345' });
await roomEngine.setMaxSeatCount({ maxSeatCount: 16 })
```

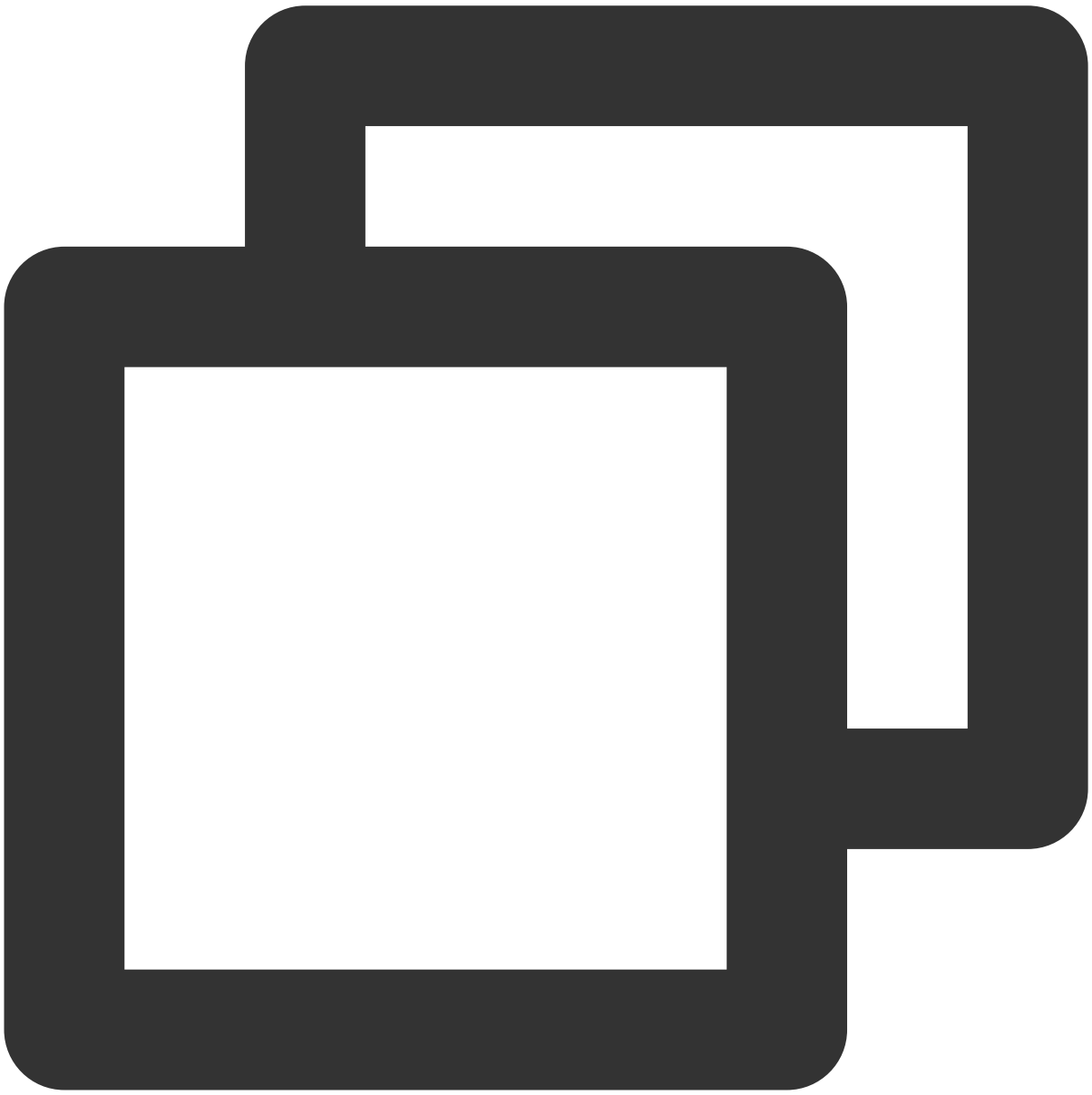
参数：

参数	类型	说明	默认值	含义
maxSeatCount	number	必填	-	设置房间麦位最大值

Returns：Promise<void>

getSeatList

获取麦位列表



```
const roomEngine = new TUIRoomEngine();
const seatList = await roomEngine.getSeatList();
```

Returns：*Promise<TUISeatInfo[]>* seatList

seatList 为当前房间所有麦位列表

takeSeat

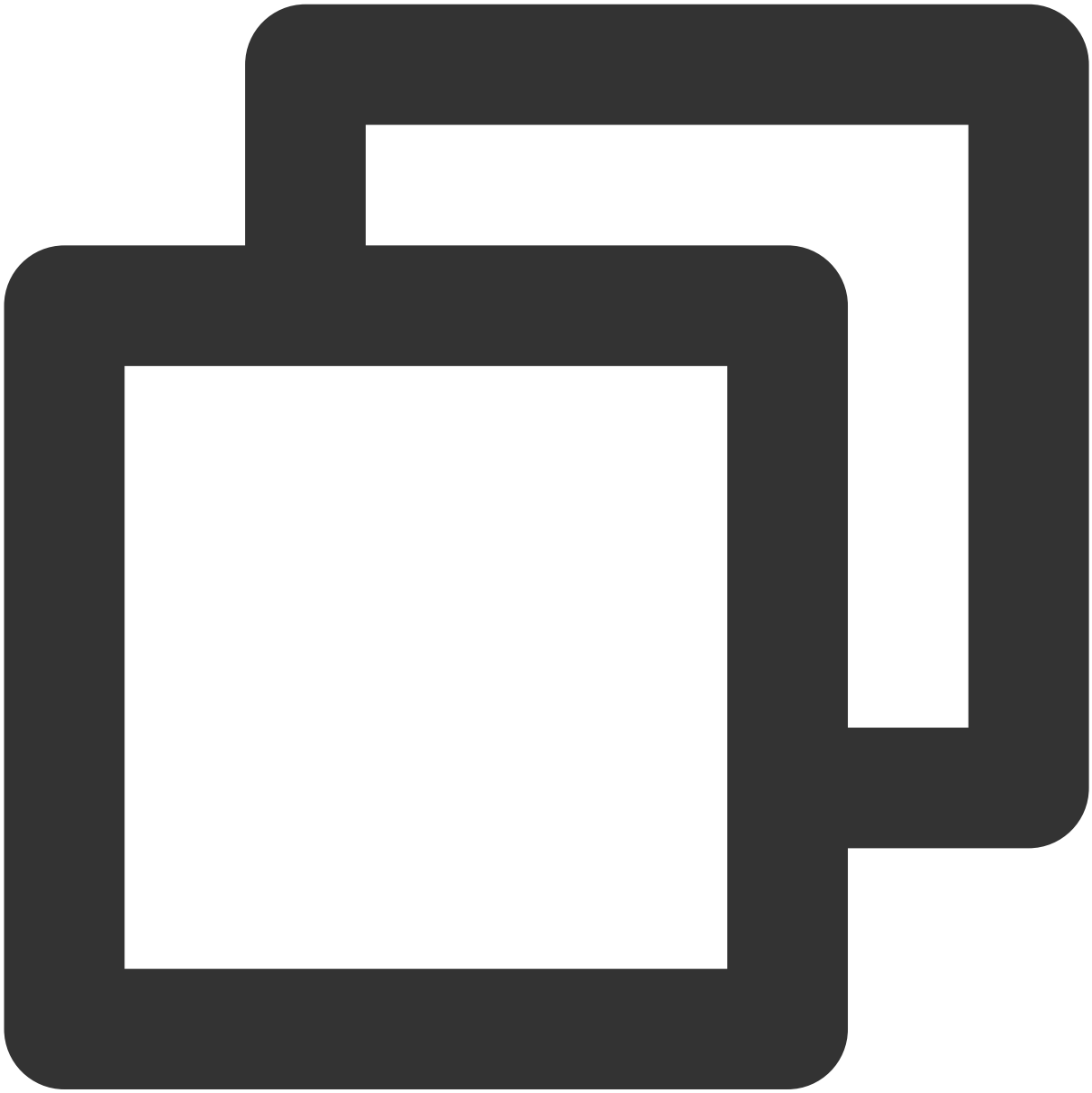
麦下用户可调用 takeSeat 成为麦上用户，仅麦上用户可发布本地音视频流。

当 roomInfo.roomType 为 TUIRoomType.kConference 且 roomInfo.speechMode 为 TUISpeechMode.kSpeakAfterTakingSeat 时，普通用户调用 takeSeat 方法需要等待主持人/管理员的同意后成为麦上用户。

当 roomInfo.roomType 为 TUIRoomType.kLivingRoom 且 roomInfo.speechMode 为 TUISpeechMode.kFreeToSpeak 时, 普通用户调用 takeSeat 方法成功后即为麦上用户

主持人&管理员调用 takeSeat 成功后即为麦上用户。

麦上用户的变更通过 TUIRoomEvents.onSeatListChanged 通知所有用户。



```
const roomEngine = new TUIRoomEngine();
// 情景一：主持人/管理员上麦
// 情景二：当 roomInfo.roomType 为 TUIRoomType.kConference
// 且 roomInfo.speechMode 为 TUISpeechMode.kSpeakAfterTakingSeat 时，普通用户上麦
await roomEngine.takeSeat({
 seatIndex: -1,
 timeout: 0,
});

// 情景三：当 roomInfo.enableSeatControl 为 true 时，普通用户上麦
const requestId = await roomEngine.instance?.takeSeat({
```

```
    seatIndex: -1,
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
       switch (requestCallbackType) {
        case TUIRequestCallbackType.kRequestAccepted:
          // 请求被接受
          break;
        case TUIRequestCallbackType.kRequestRejected:
          // 请求被拒绝
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
    }
  },
});
```

参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index, 无麦位序号时设置为 -1
timeout	number	必填	-	超时时间。若 timeout 设置为 0，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：Promise<string> requestId

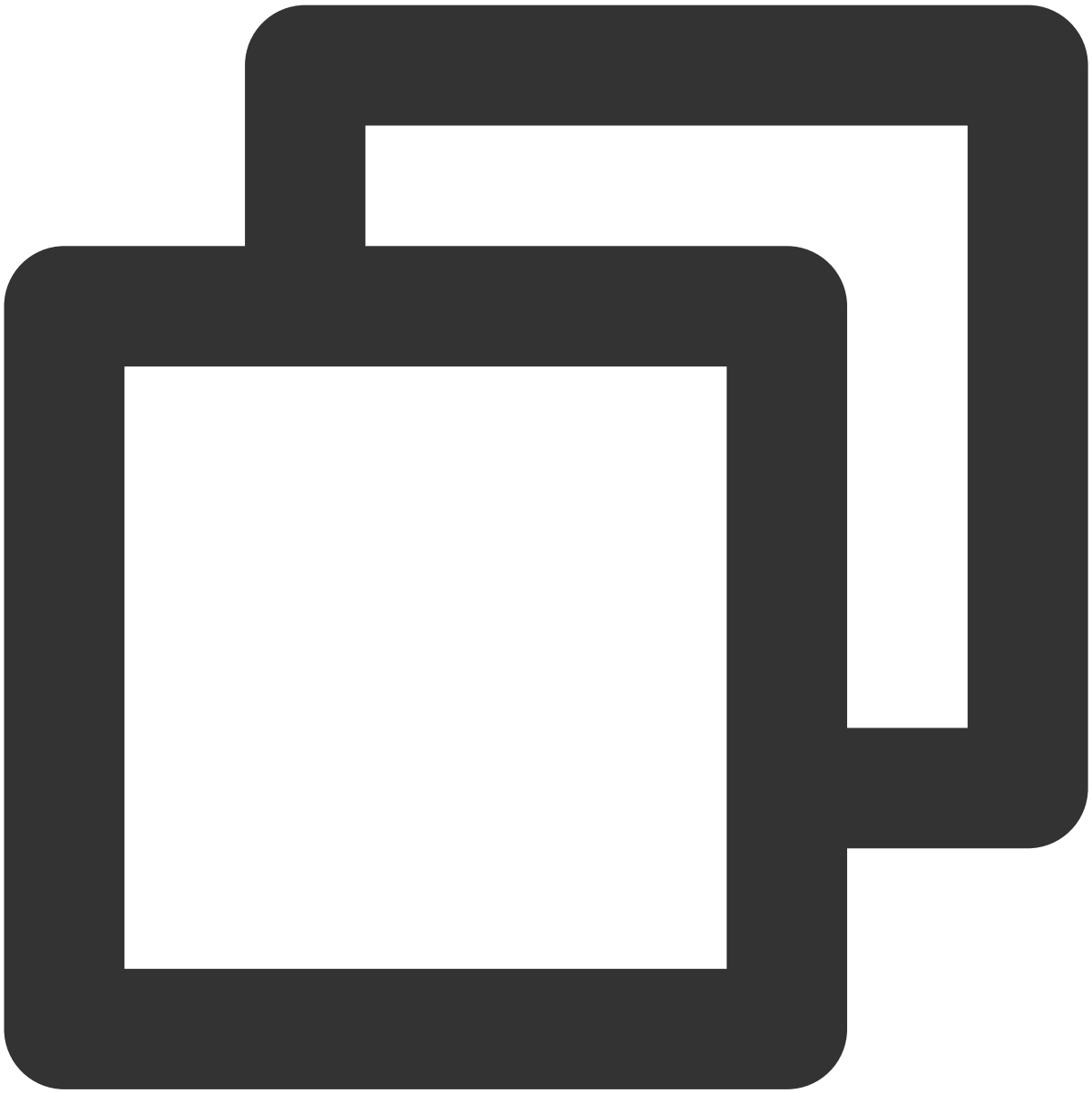
当 roomInfo.enableSeatControl 为 true，普通用户调用该接口时返回 requestId，普通用户可使用该 requestId 调用 cancelRequest 接口取消上麦请求。

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

leaveSeat

释放麦位。

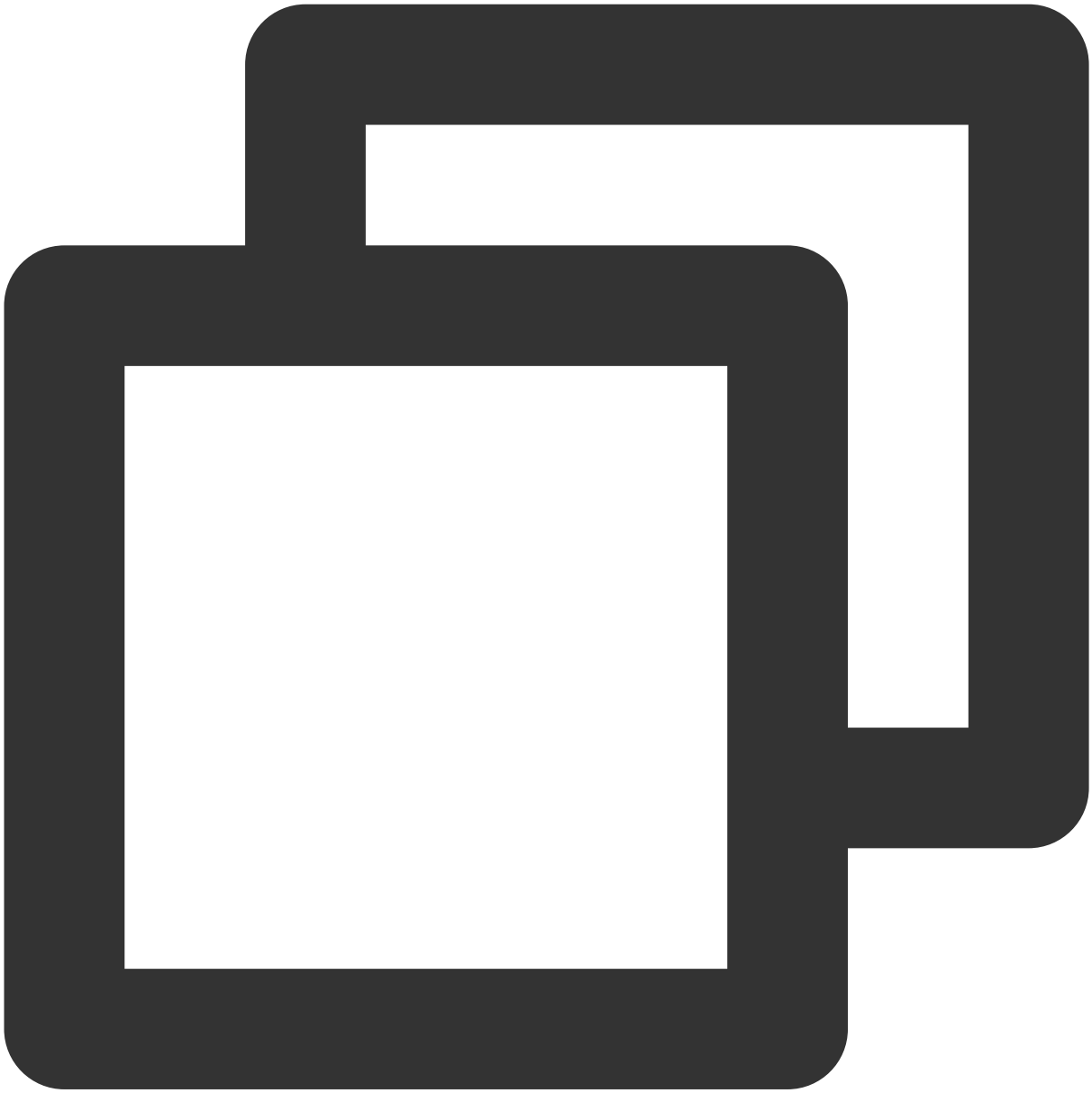


```
const roomEngine = new TUIRoomEngine();
await roomEngine.leaveSeat();
```

Returns：*Promise<void>*

takeUserOnSeatByAdmin

邀请其他人上麦。



```
const roomEngine = new TUIRoomEngine();
const requestId = roomEngine.takeUserOnSeatByAdmin({
    seatIndex: 0,
    userId: 'user_1234',
    timeout: 0,
    requestCallback: ({ requestCallbackType, requestId, userId, code, message }) =>
     switch (requestCallbackType) {
       case TUIRequestCallbackType.kRequestAccepted:
         // 请求被接受
         break;
       case TUIRequestCallbackType.kRequestRejected:
```

```
          // 请求被拒绝
          break;
        case TUIRequestCallbackType.kRequestCancelled:
          // 请求已取消
          break;
        case TUIRequestCallbackType.kRequestTimeout:
          // 请求超时
          break;
        case TUIRequestCallbackType.kRequestError:
          // 请求错误
          break;
        default:
          break;
      }
    },
});
```

参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index
userId	string	必填	-	用户 ID
timeout	number	必填	-	超时时间。若 timeout 设置为 0 ，则无超时时间
requestCallback	Function	选填	空函数	请求回调，用来通知发起方请求被接受/拒绝/取消/超时/错误的回调

Returns：*Promise<string> requestId*

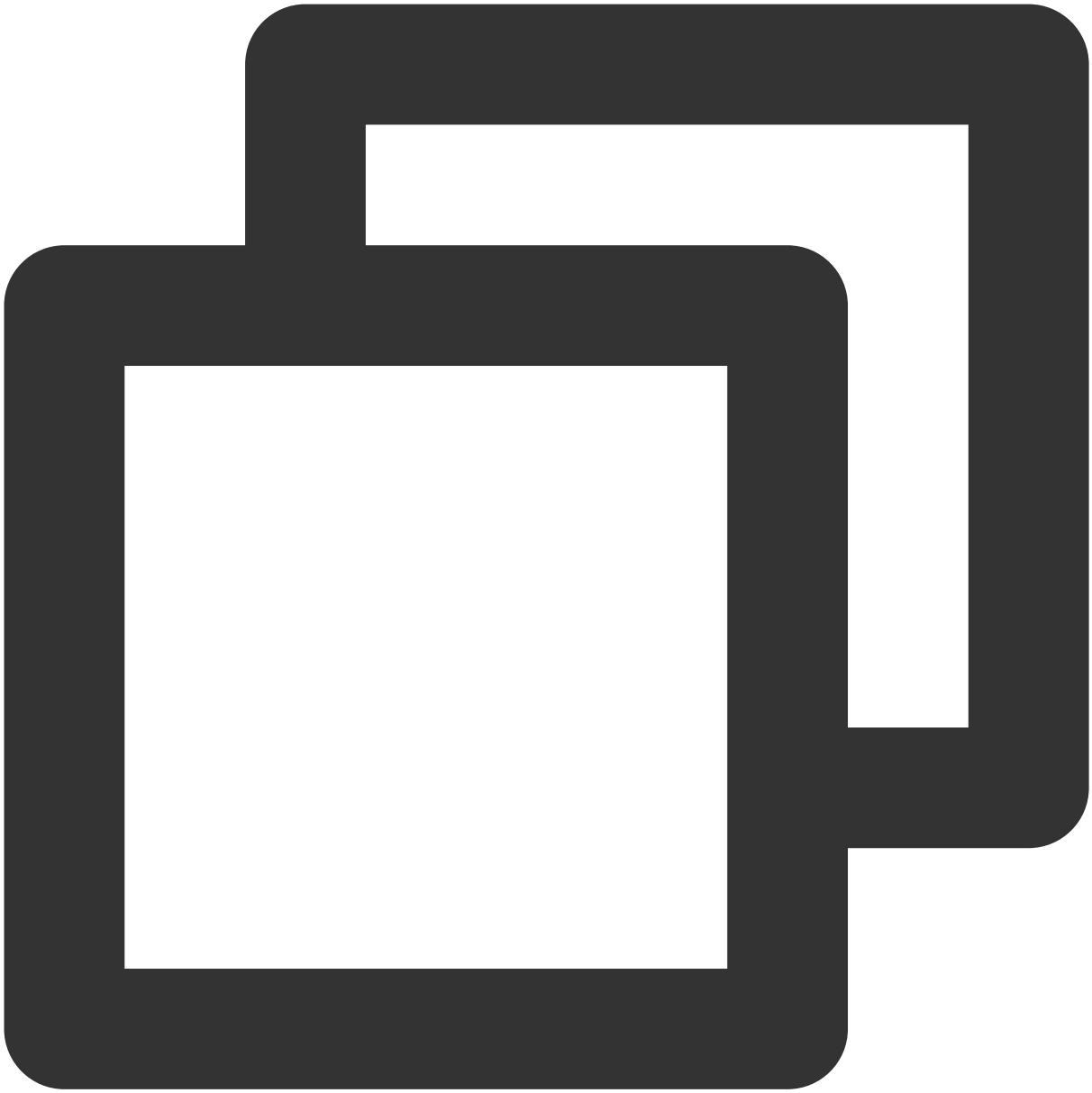
该接口返回 requestId，用户可使用该 requestId 调用 cancelRequest 接口取消请求

说明：

v1.0.2 及其以上版本该接口返回的 requestId 类型为 string；v1.0.0 及 v1.0.1 版本该接口返回的 requestId 类型为 number；

kickUserOffSeatByAdmin

要求其他人下麦。



```
const roomEngine = new TUIRoomEngine();
const requestId = await roomEngine.kickUserOffSeatByAdmin({
   seatIndex: 0,
   userId: 'user_1234',
});
```

参数：

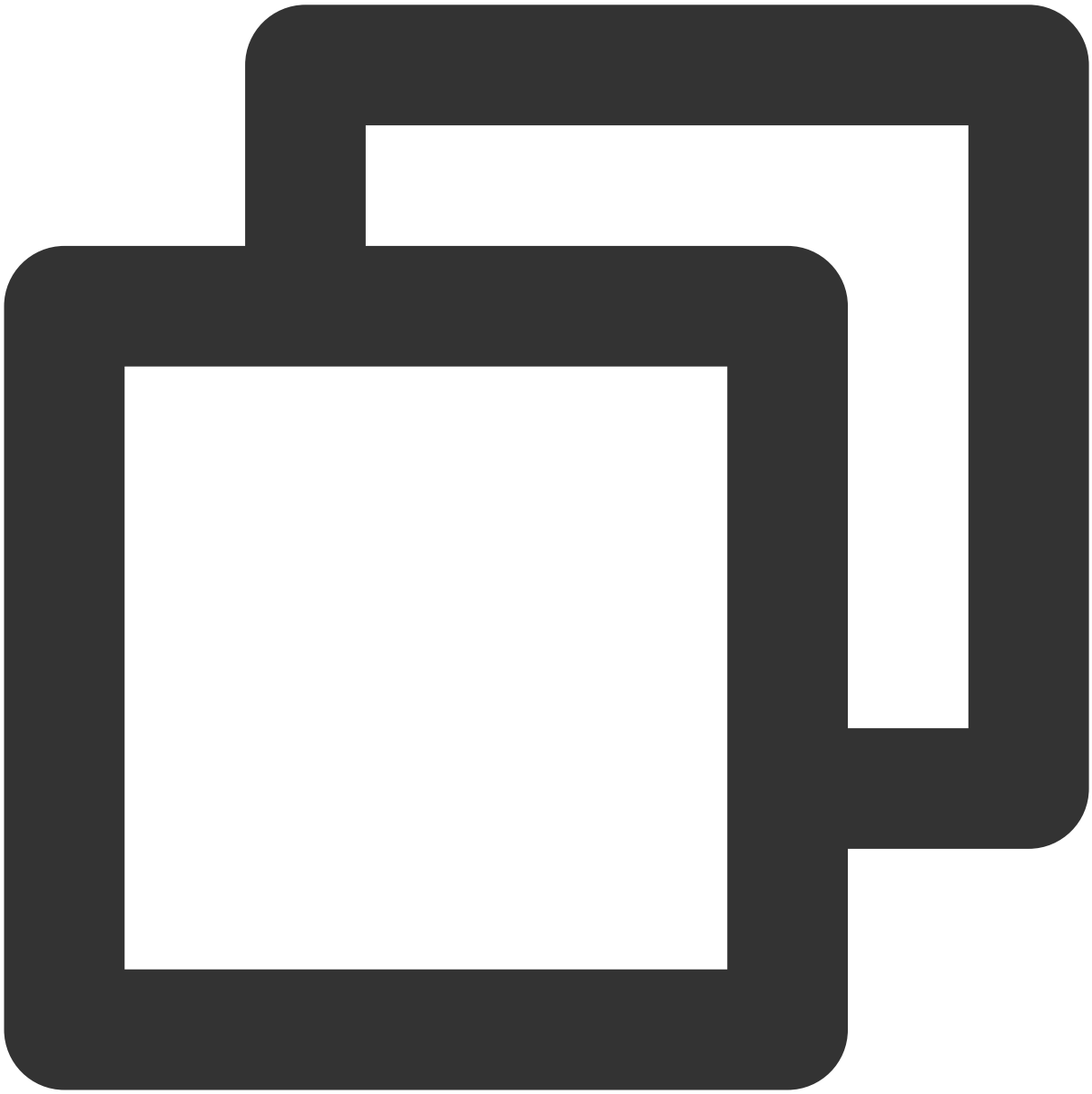
参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位 index

userId	string	必填	-	用户 ID

Returns：*Promise<void>*

lockSeatByAdmin

锁定某个麦位状态（仅房间主持人和管理员可调用此方法）。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.lockSeatByAdmin({
   seatIndex: 0,
```

```
    lockParams: {
        lockSeat: true,
        lockVideo: true,
        lockAudio: true,
    },
});
```

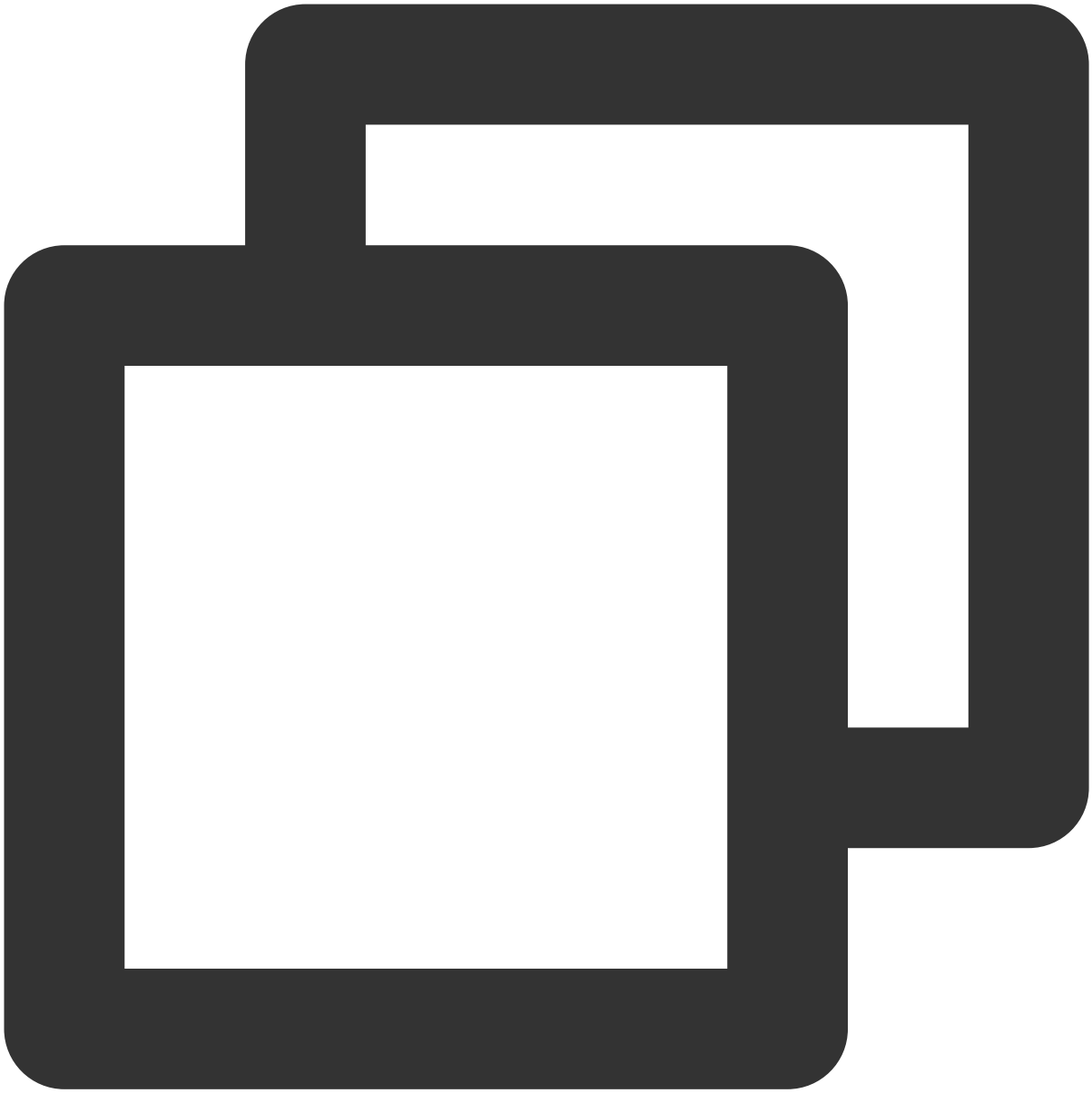
参数：

参数	类型	说明	默认值	含义
seatIndex	number	必填	-	麦位索引
lockParams	TUISeatLockParams	必填	-	锁麦参数

Returns：*Promise<void>*

startScreenSharingElectron

开始屏幕共享 Electron。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startScreenSharingElectron('targetId');
```

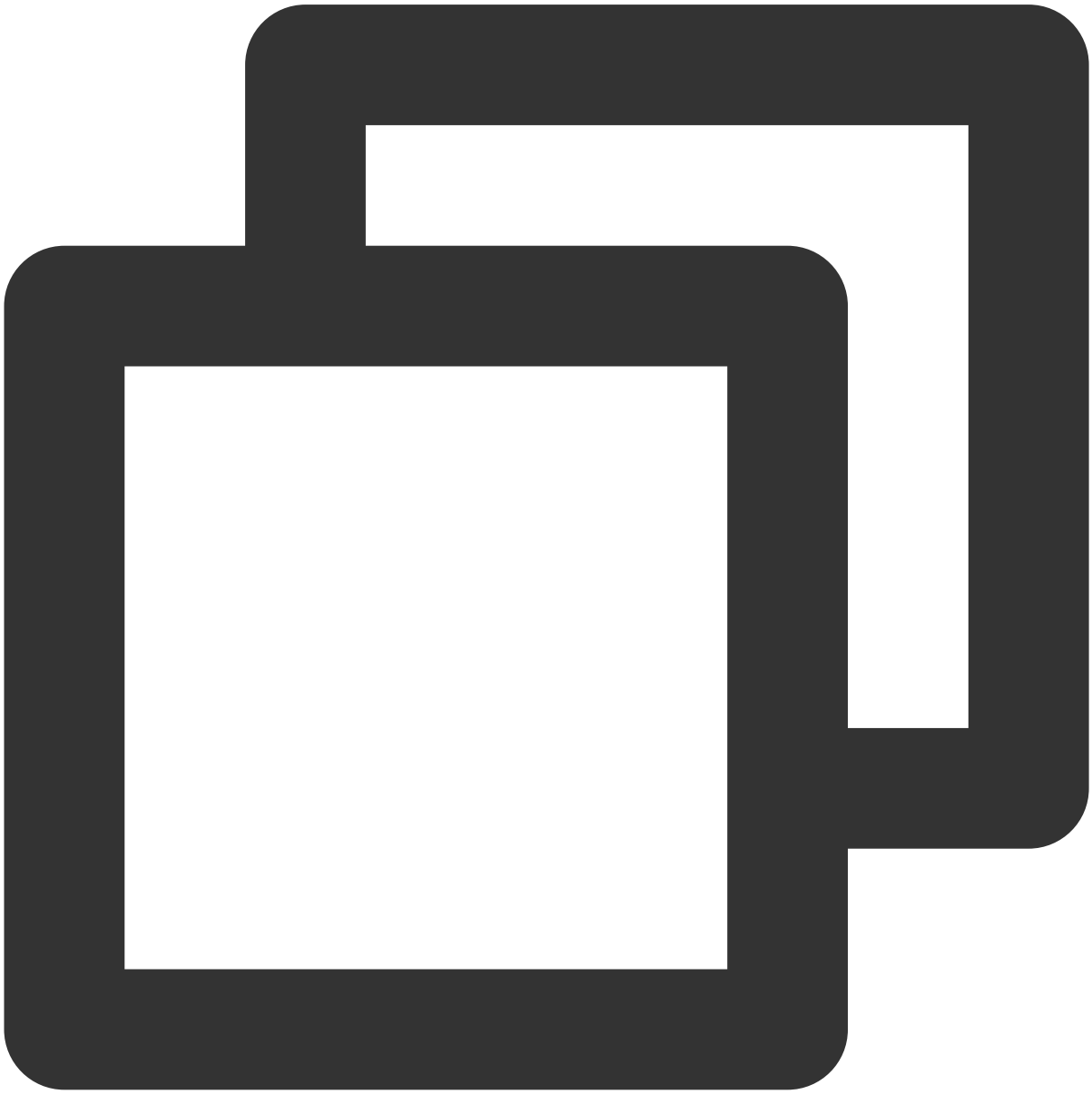
参数：

参数	类型	说明	默认值	含义
targetId	string	选填	-	分享窗口 ID，可从 getScreenSharingTarget 获取

Returns：*Promise<void>*

stopScreenSharingElectron

停止屏幕共享 Electron。

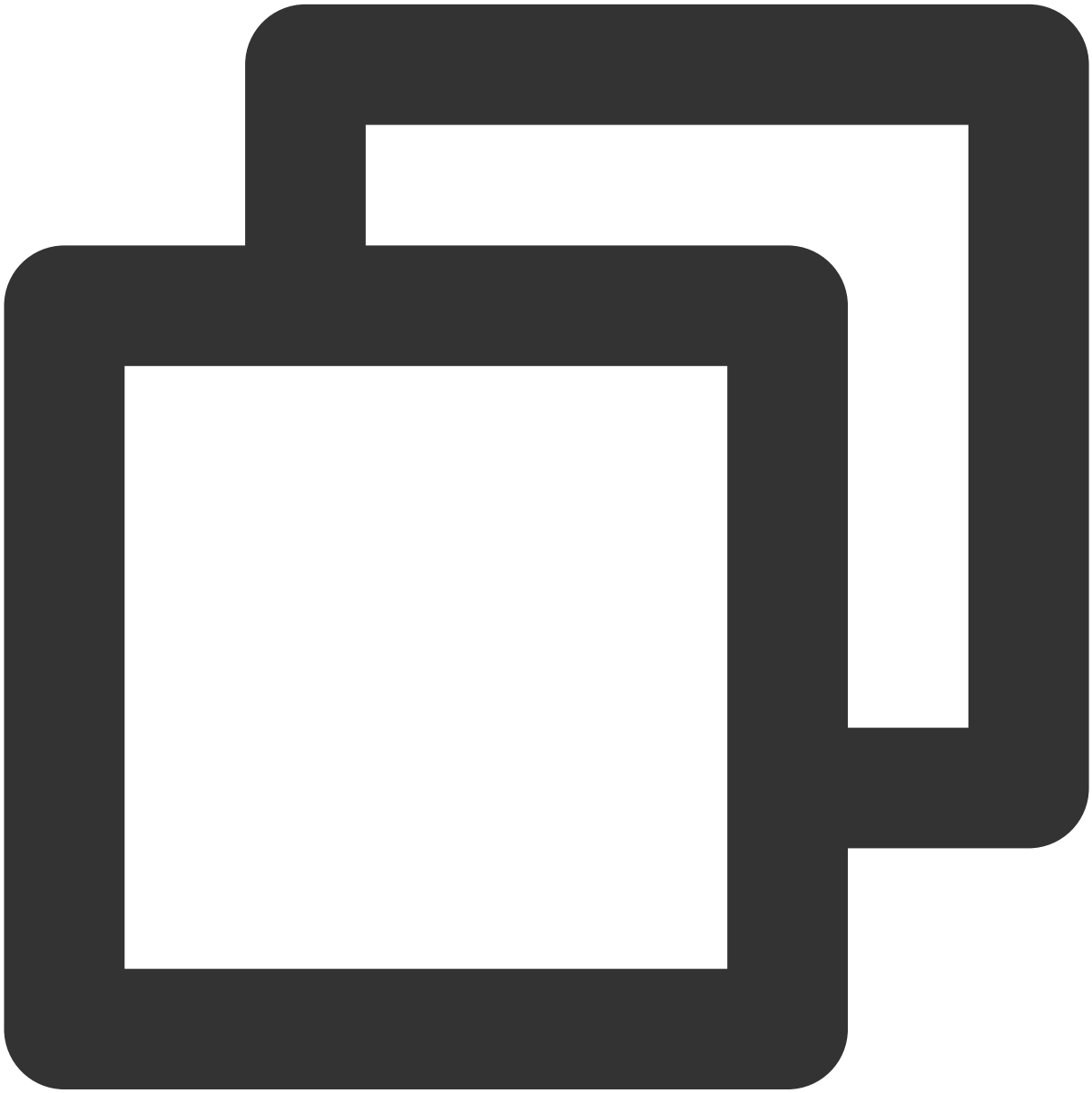


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopScreenSharingElectron();
```

Returns：*Promise<void>*

getScreenSharingTarget

获取屏幕分享列表 Electron。

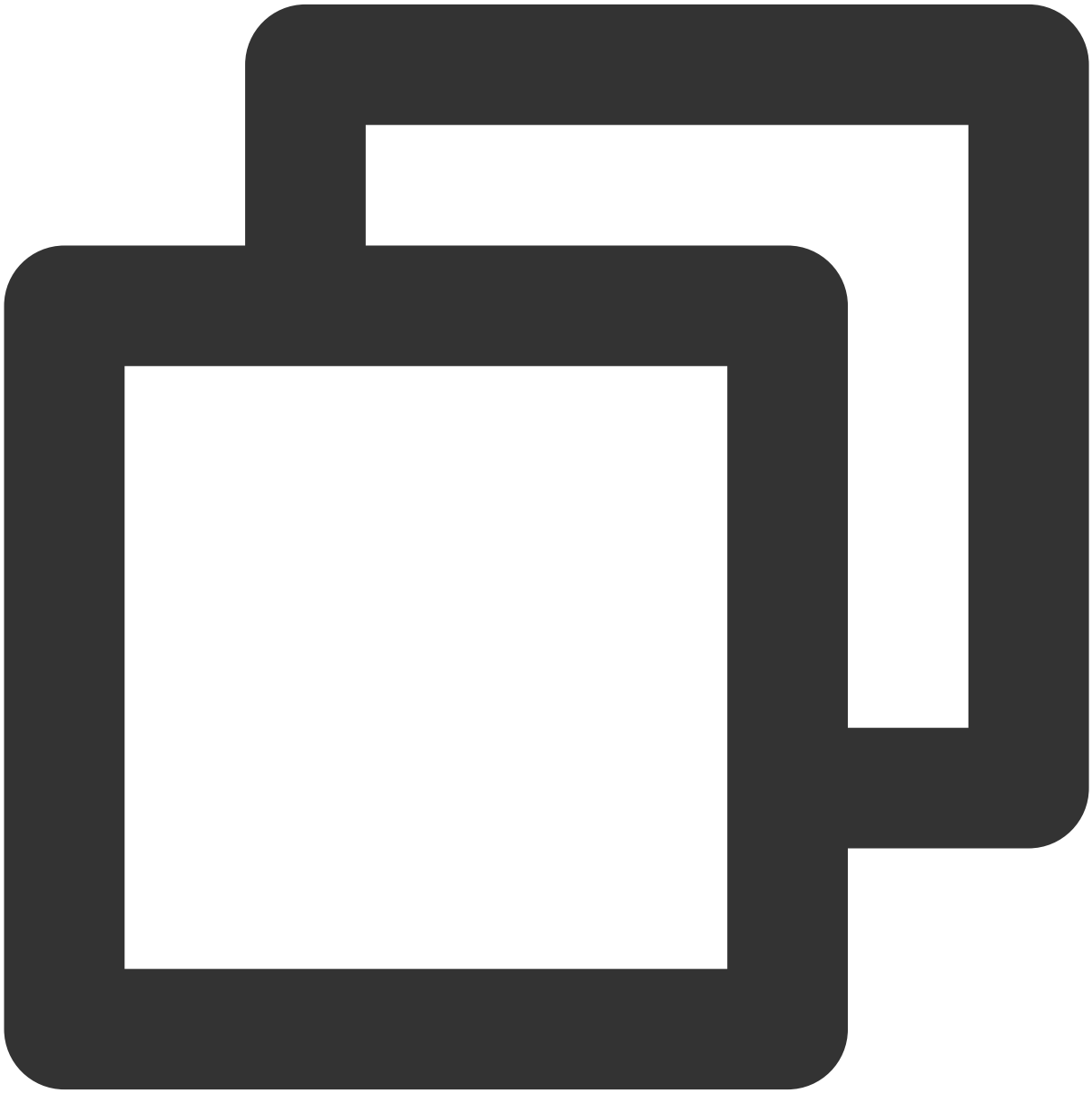


```
const roomEngine = new TUIRoomEngine();
const screenList = await roomEngine.getScreenSharingTarget();
```

Returns：*Promise<void>*

selectScreenSharingTarget

切换屏幕分享窗口 Electron。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.selectScreenSharingTarget('targetId');
```

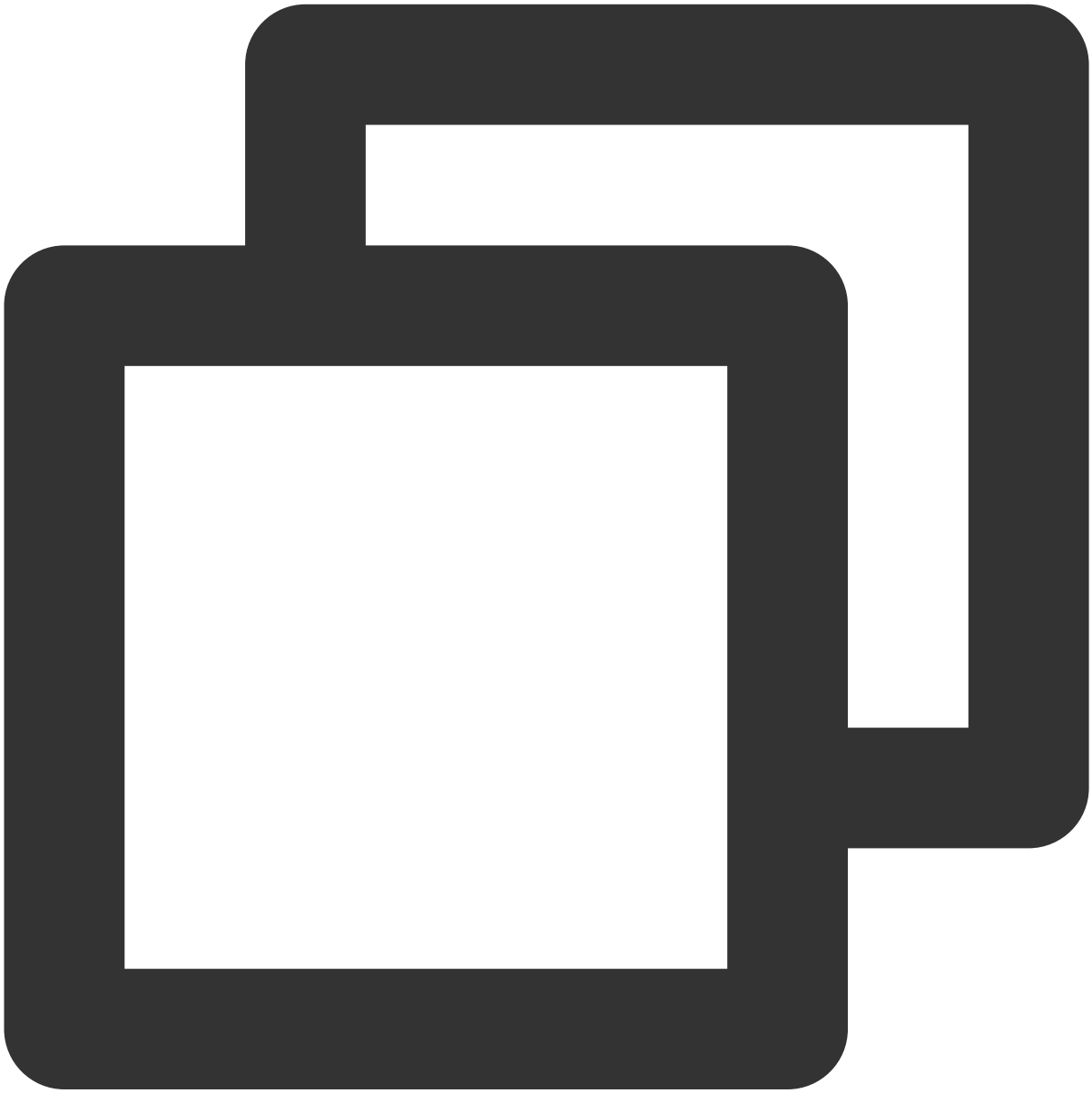
参数：

参数	类型	说明	默认值	含义
targetId	string	必填	-	分享窗口 ID，可从 getScreenSharingTarget 获取

Returns：*Promise<void>*

sendTextMessage

发送文本消息。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendTextMessage({
 messageText: 'hello, everyone',
});
```

参数：

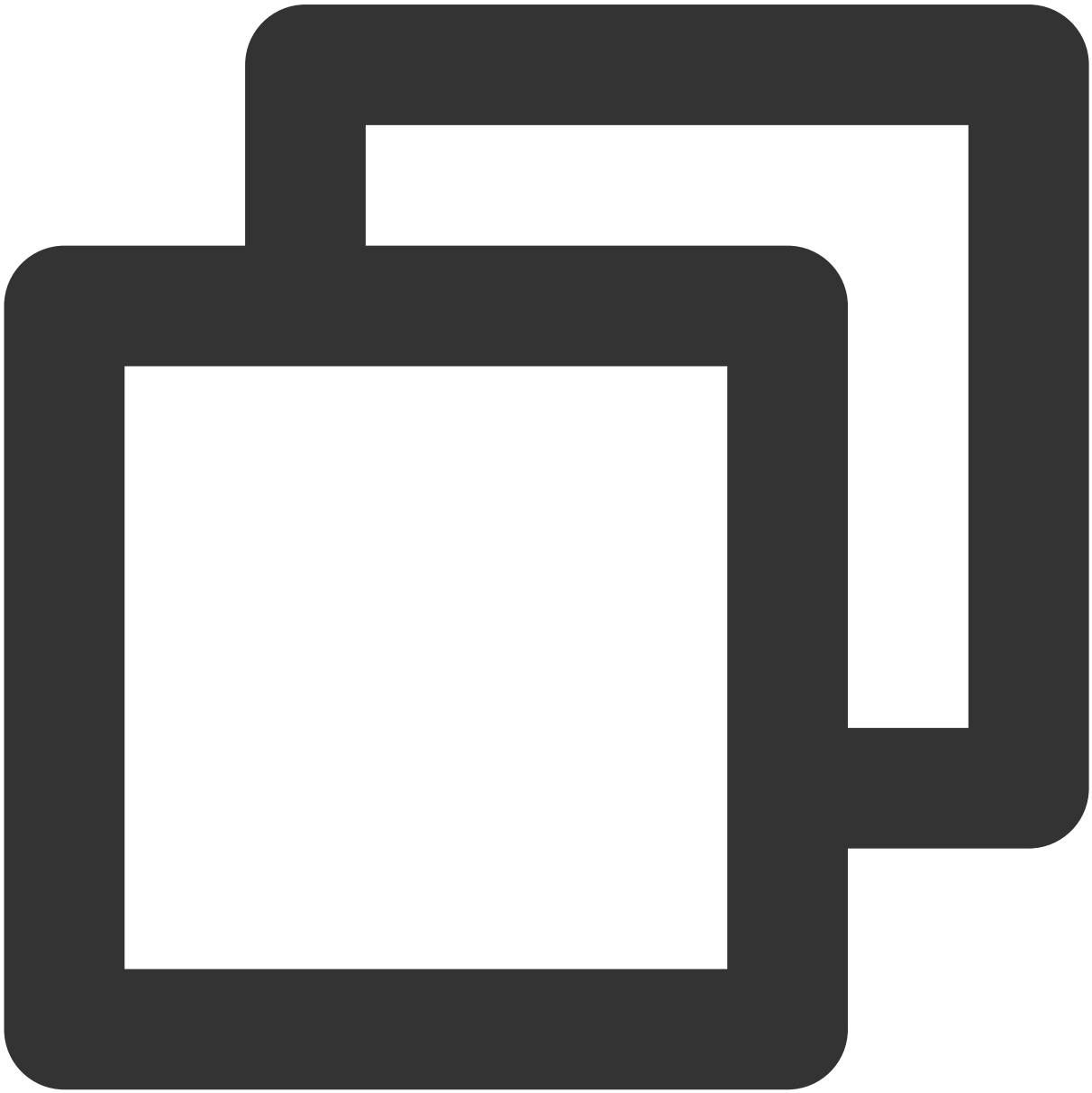
参数	类型	说明	默认值	含义

messageText	string	必填	-	文本消息内容

Returns：*Promise<void>*

sendCustomMessage

发送自定义消息。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.sendCustomMessage({
 messageText: '{ data:'', description: ''}',
```

```
});
```

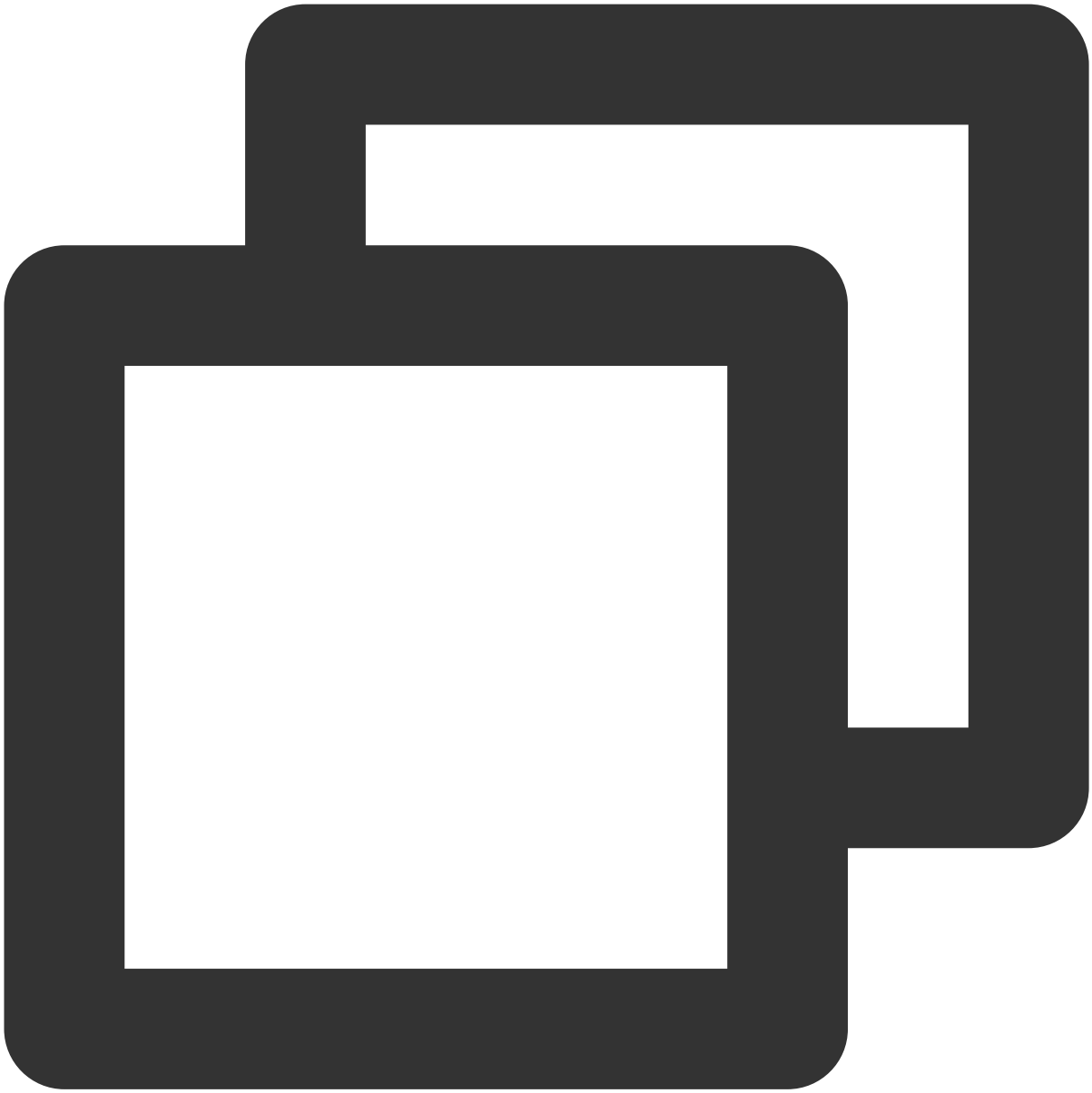
参数：

参数	类型	说明	默认值	含义
messageText	string	必填	-	自定义消息内容

Returns：*Promise<void>*

getCameraDevicesList

获取摄像头设备列表。

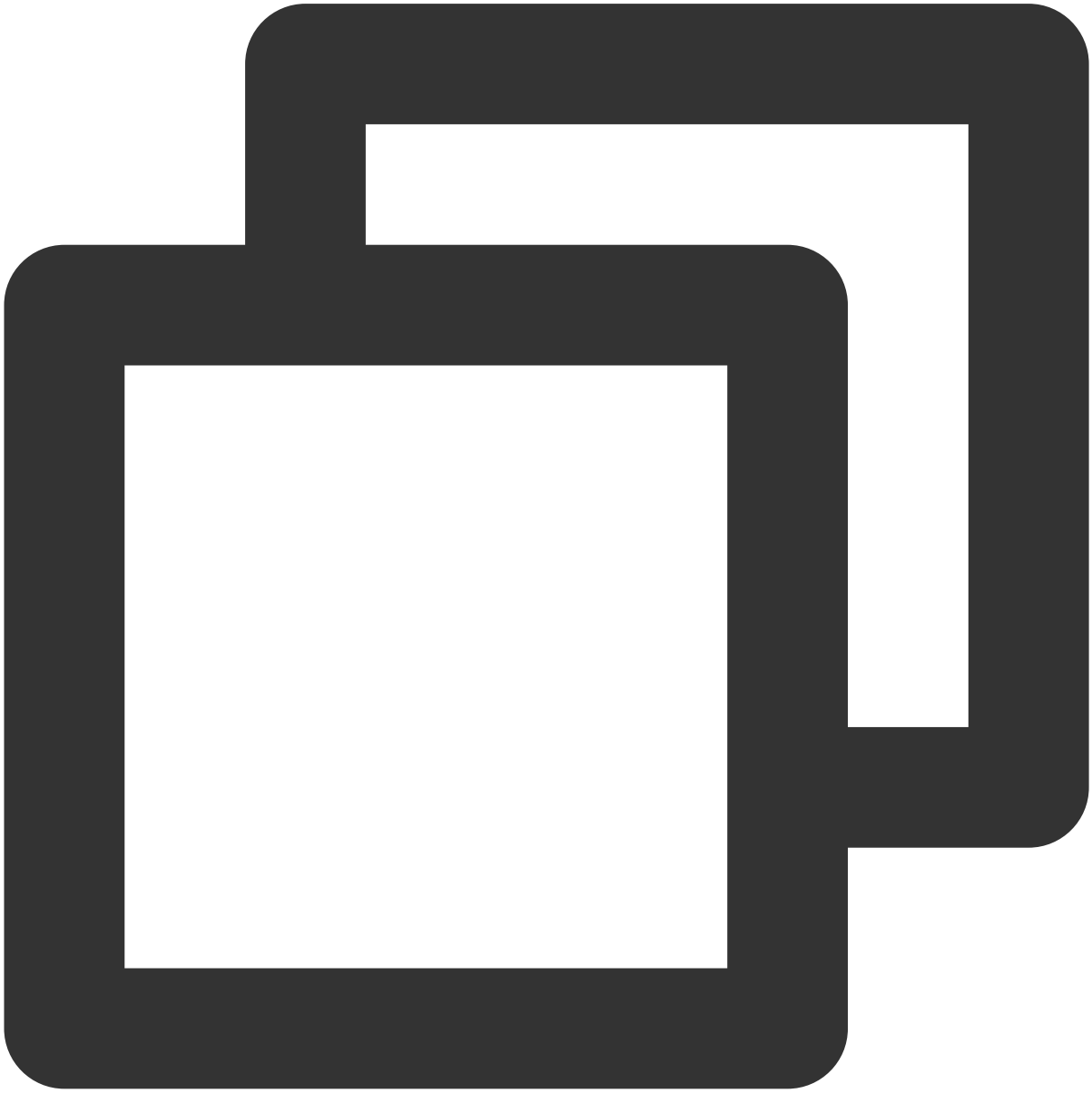


```
const roomEngine = new TUIRoomEngine();
const cameraList = await roomEngine.getCameraDevicesList();
for (i = 0; i < cameraList.length; i++) {
   var camera = cameraList[i];
   console.info("camera deviceName: " + camera.deviceName + " deviceId:" + camera.
}
```

Returns: *Promise<*TRTCDeviceInfo*[]>* cameraList

getMicDevicesList

获取麦克风设备列表。

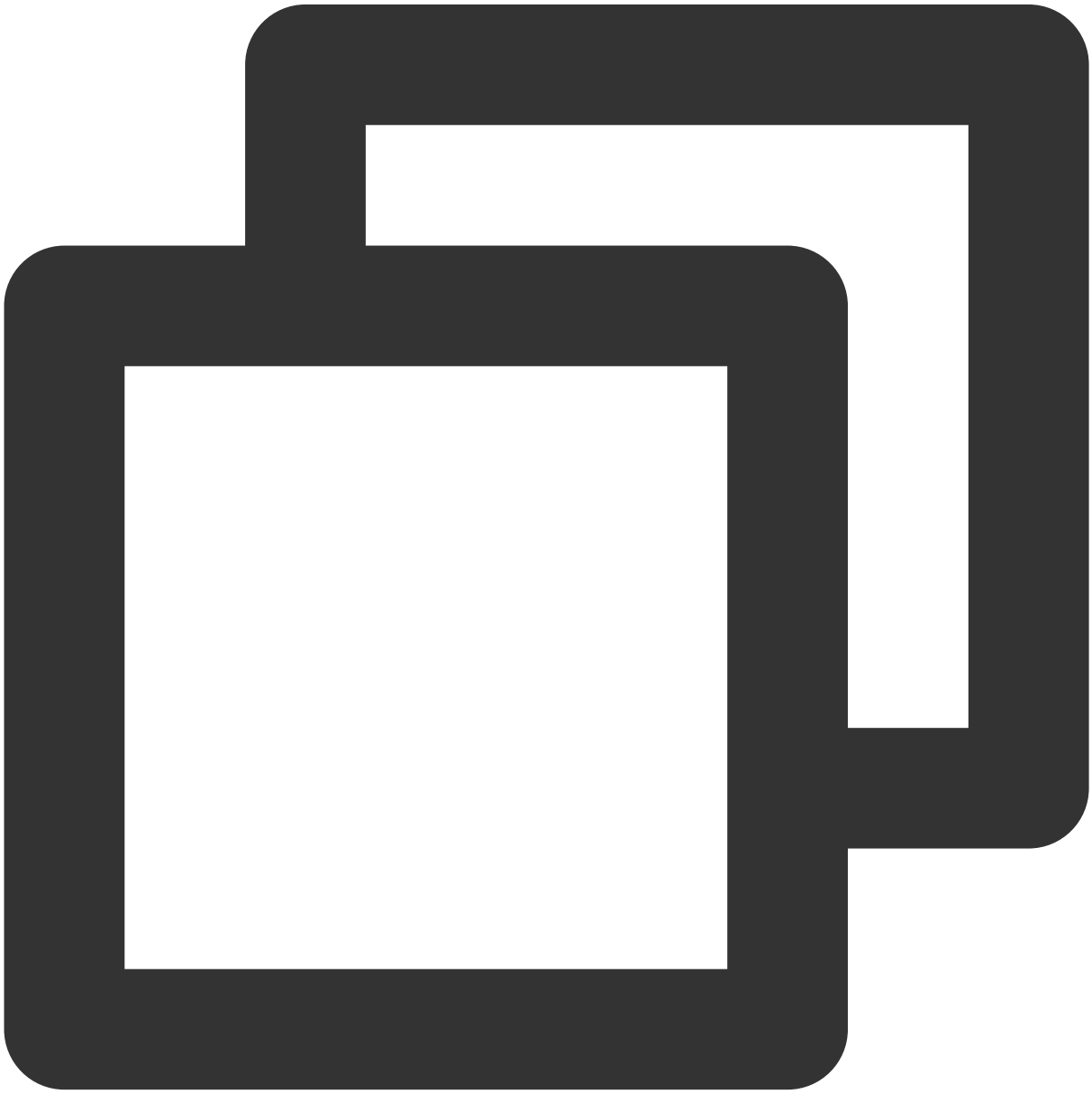


```
const roomEngine = new TUIRoomEngine();
const micList = await roomEngine.getMicDevicesList();
for (i = 0; i < micList.length; i++) {
   var mic = micList[i];
   console.info("mic deviceName: " + mic.deviceName + " deviceId:" + mic.deviceId)
}
```

Returns: *Promise<*TRTCDeviceInfo*[]>* micList

getSpeakerDevicesList

获取扬声器设备列表。

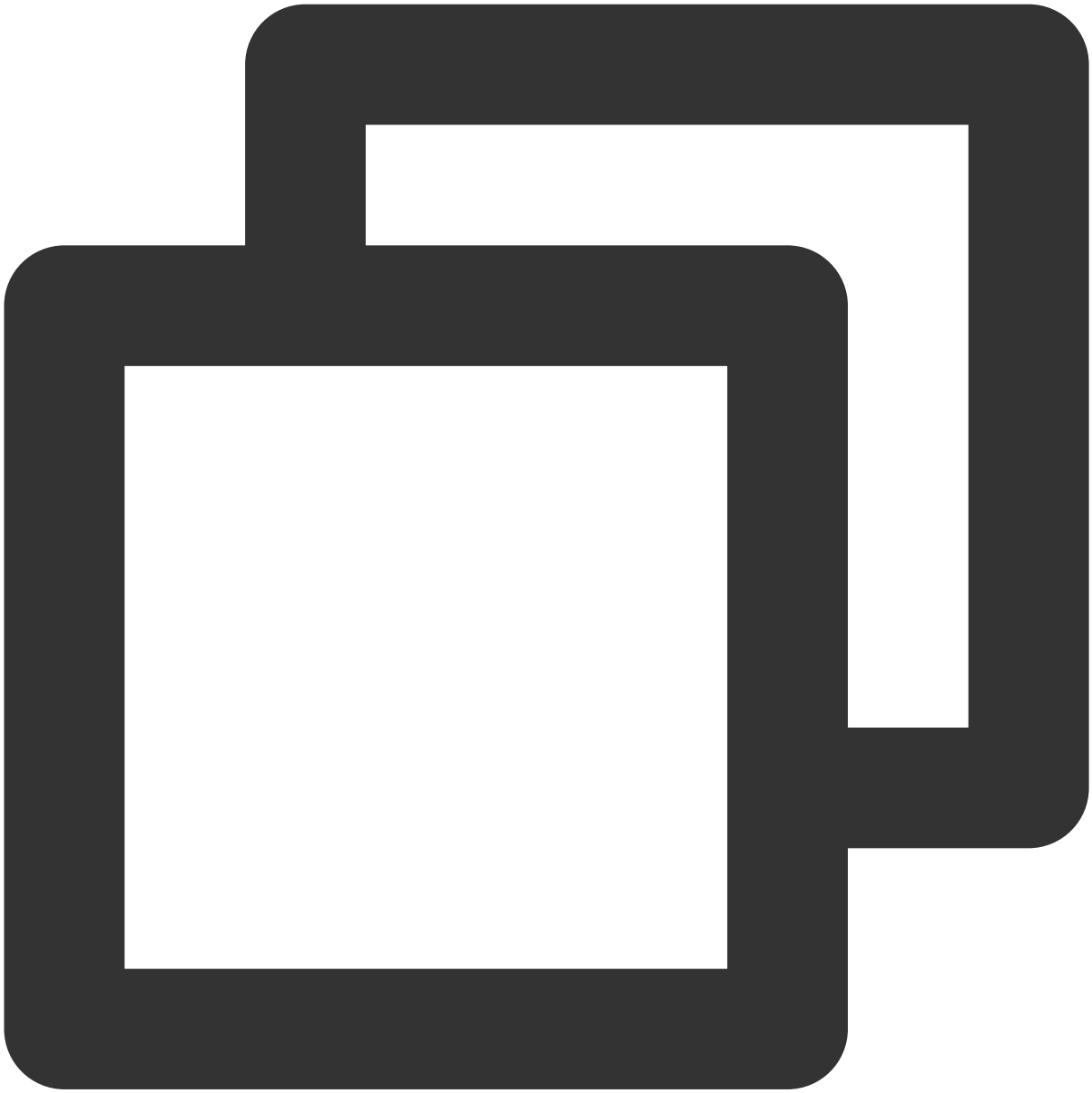


```
const roomEngine = new TUIRoomEngine();
const speakerList = await roomEngine.getSpeakerDevicesList();
for (i = 0; i < speakerList.length; i++) {
   var speaker = speakerList[i];
   console.info("speaker deviceName: " + speaker.deviceName + " deviceId:" + speak
}
```

Returns: *Promise<*TRTCDeviceInfo*[]>* speakerList

setCurrentCameraDevice

设置要使用的摄像头设备。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentCameraDevice({ deviceId: '' });
```

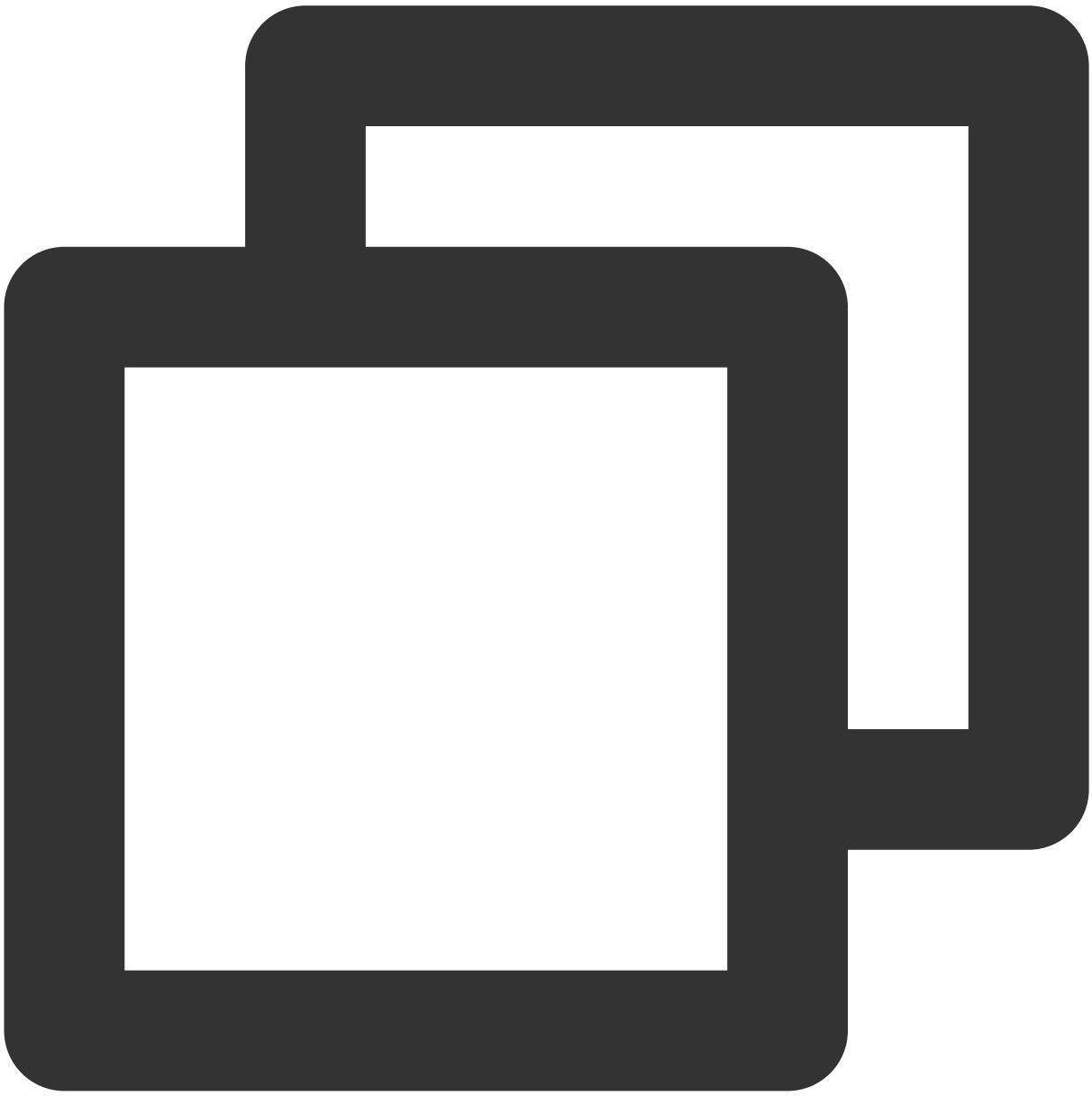
参数：

参数	类型	说明	默认值	含义
deviceId	string	必填	-	从 getCameraDevicesList 中得到的设备 ID

Returns：***void***

setCurrentMicDevice

设置要使用的麦克风设备。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentMicDevice({ deviceId: '' });
```

参数：

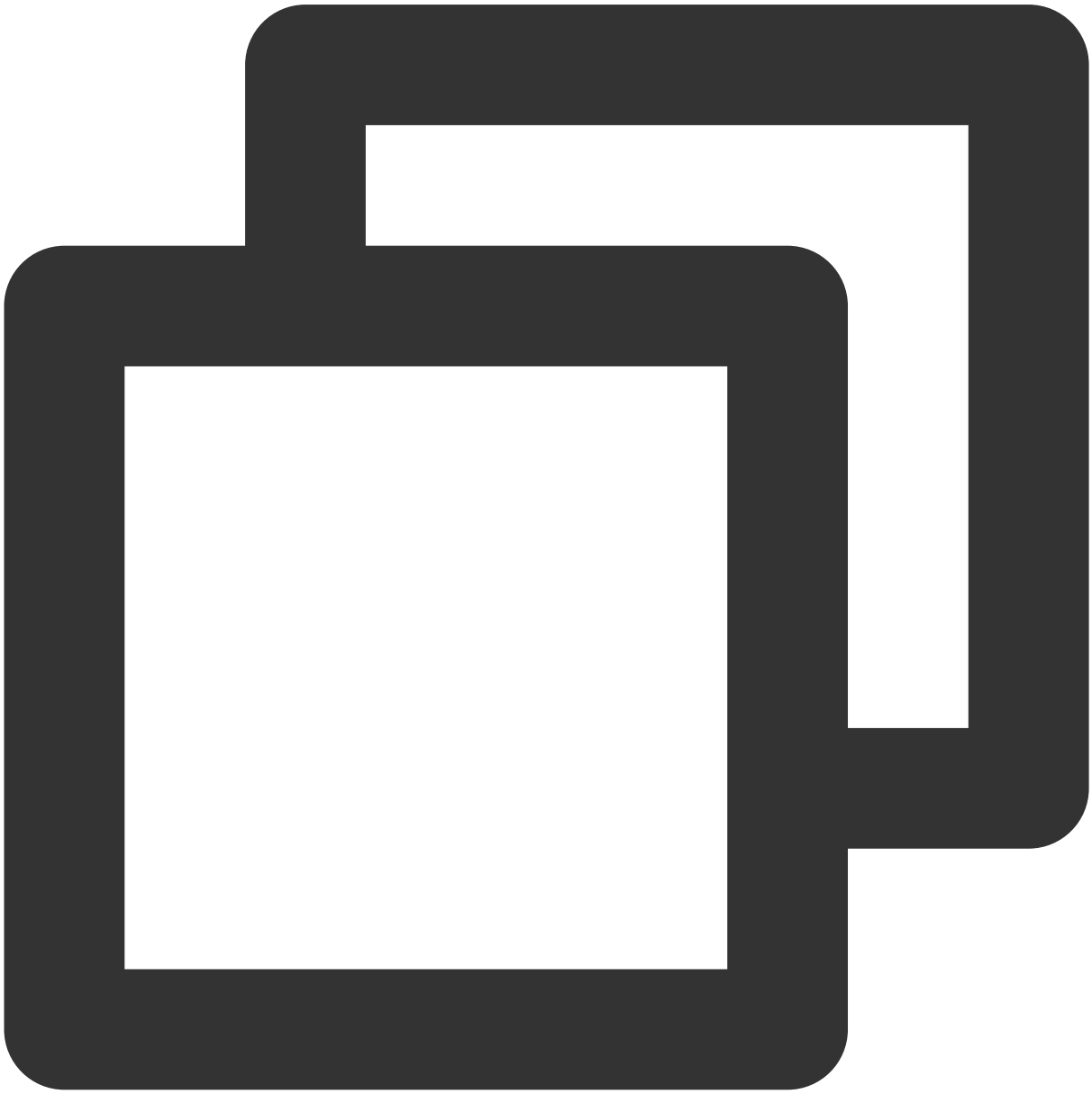
参数	类型	说明	默认值	含义

deviceId	string	必填	-	从 getMicDevicesList 中得到的设备 ID

Returns：***void***

setCurrentSpeakerDevice

设置要使用的扬声器设备。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.setCurrentSpeakerDevice({ deviceId: '' });
```

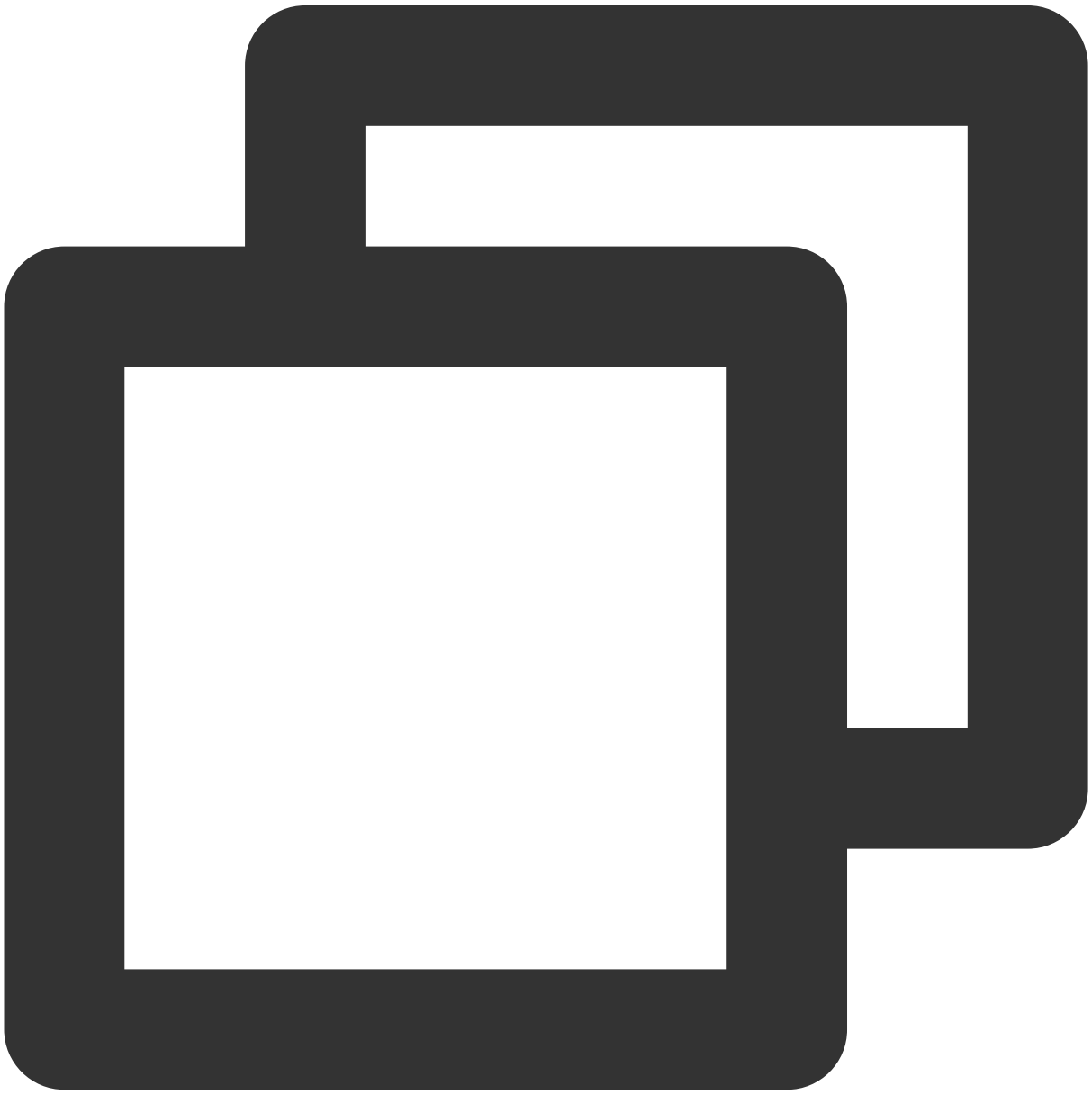
参数：

参数	类型	说明	默认值	含义
deviceId	string	必填	-	从 getSpeakerDevicesList 中得到的设备 ID

Returns：*void*

getCurrentCameraDevice

获取当前正在使用的摄像头设备。



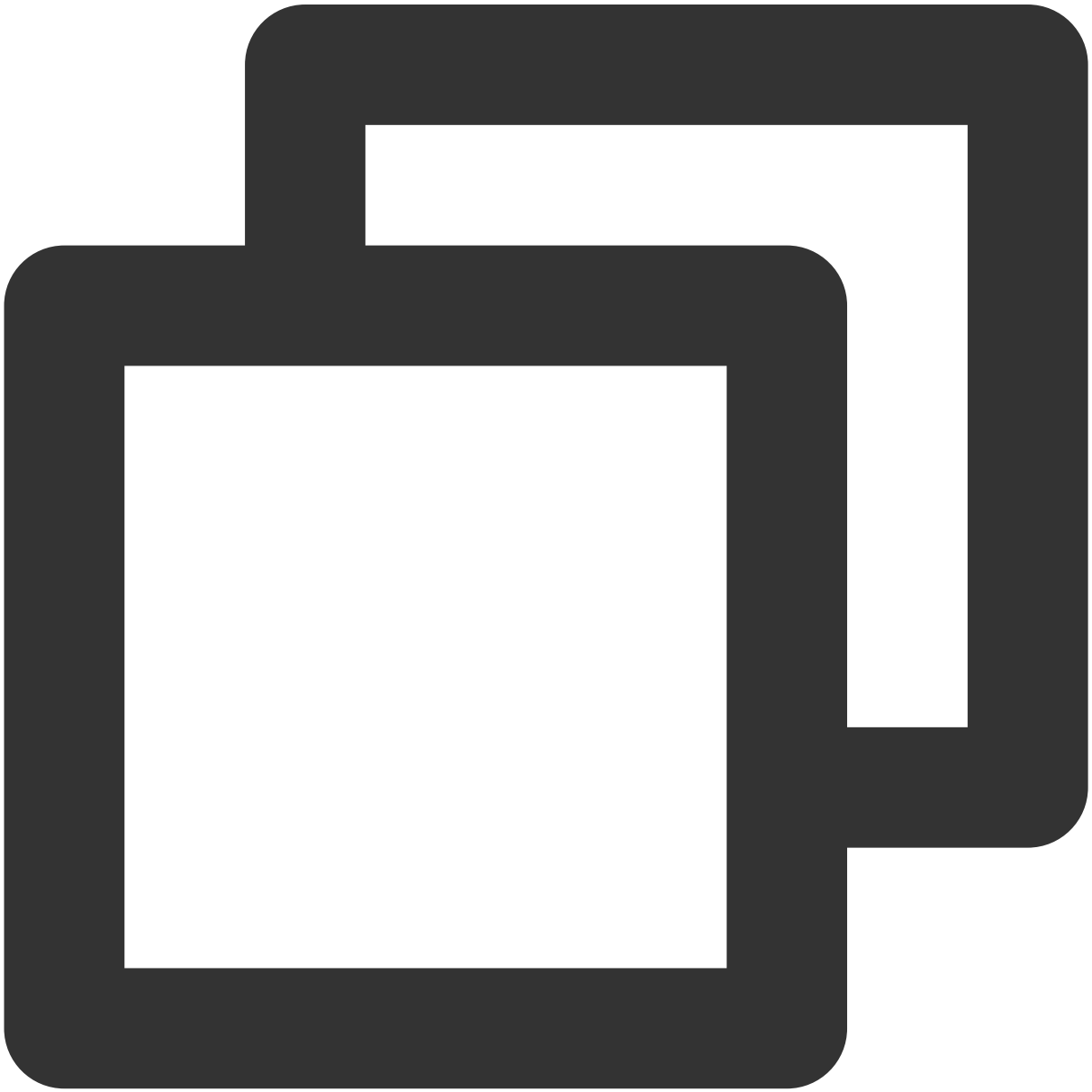
```
const roomEngine = new TUIRoomEngine();
const currentCameraDevice = roomEngine.getCurrentCameraDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

getCurrentMicDevice

获取当前正在使用的麦克风设备。



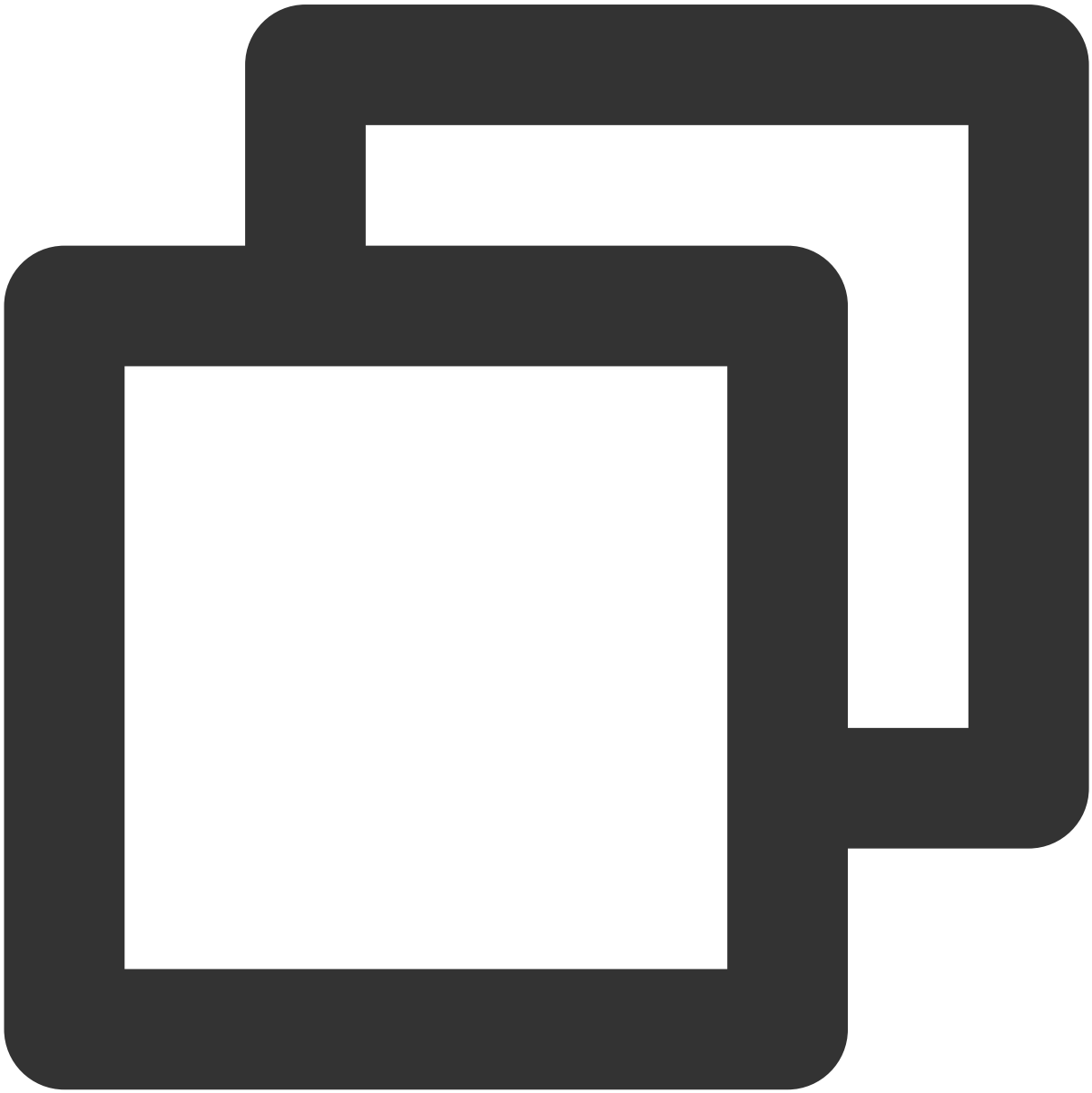
```
const roomEngine = new TUIRoomEngine();
const currentMicDevice = roomEngine.getCurrentMicDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

getCurrentSpeakerDevice

获取当前正在使用的扬声器设备。



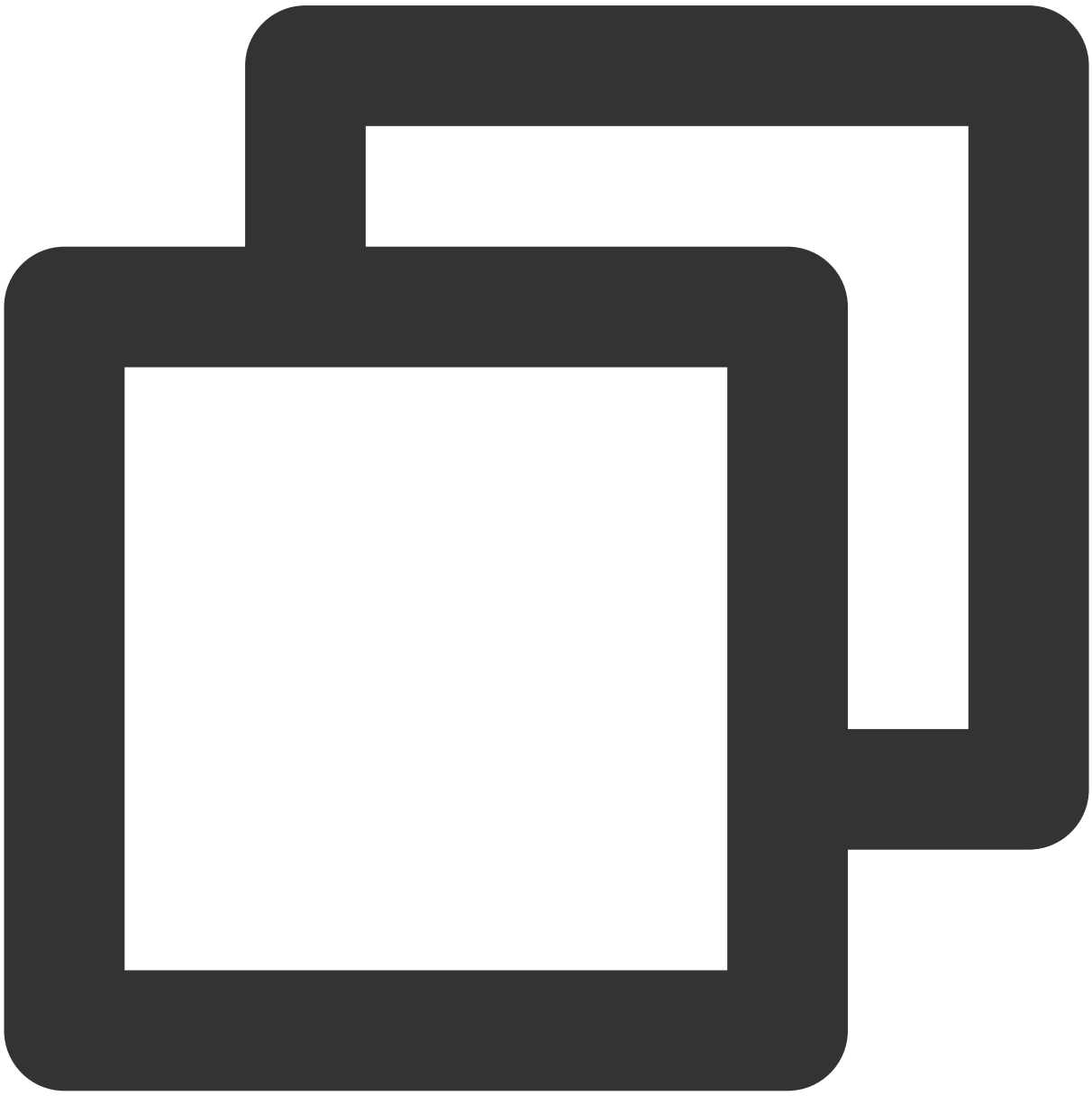
```
const roomEngine = new TUIRoomEngine();
const currentSpeakerDevice = roomEngine.getCurrentSpeakerDevice();
```

Returns：TRTCDeviceInfo

设备信息，能获取设备 ID 和设备名称

startCameraDeviceTest

开始进行摄像头测试。



```
const roomEngine = new TUIRoomEngine();
await roomEngine.startCameraDeviceTest({ view: 'test-preview' });
```

参数：

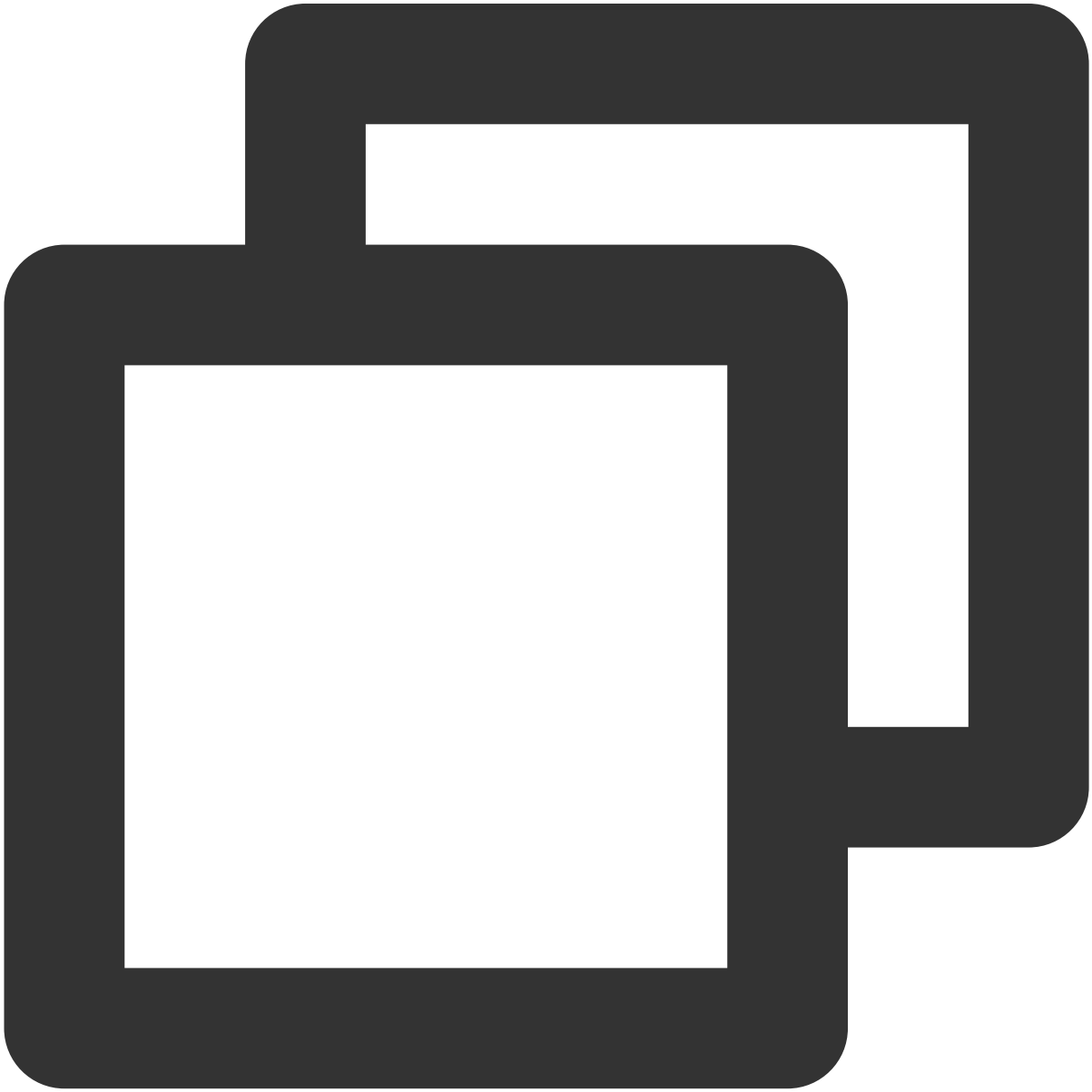
参数	类型	说明	默认值	含义

view	string	必填	-	显示摄像头测试的视频区域, 传入的 view 为承载预览画面 div 元素的 Id

Returns：*Promise<void>*

stopCameraDeviceTest

停止摄像头测试。

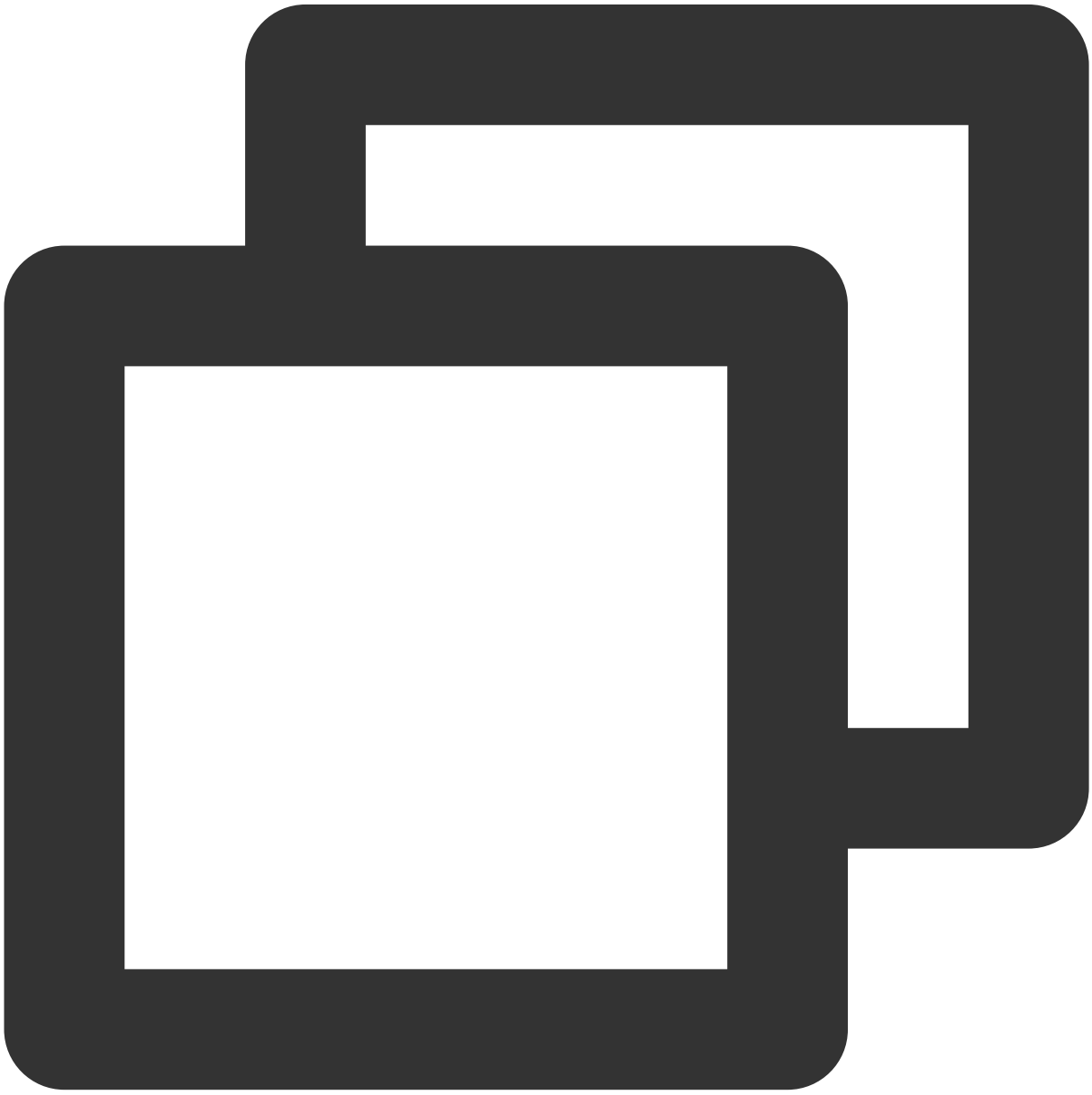


```
const roomEngine = new TUIRoomEngine();
await roomEngine.stopCameraDeviceTest();
```

Returns：*Promise<void>*

on

监听 roomEngine 的事件。



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(event, func);
```

参数：

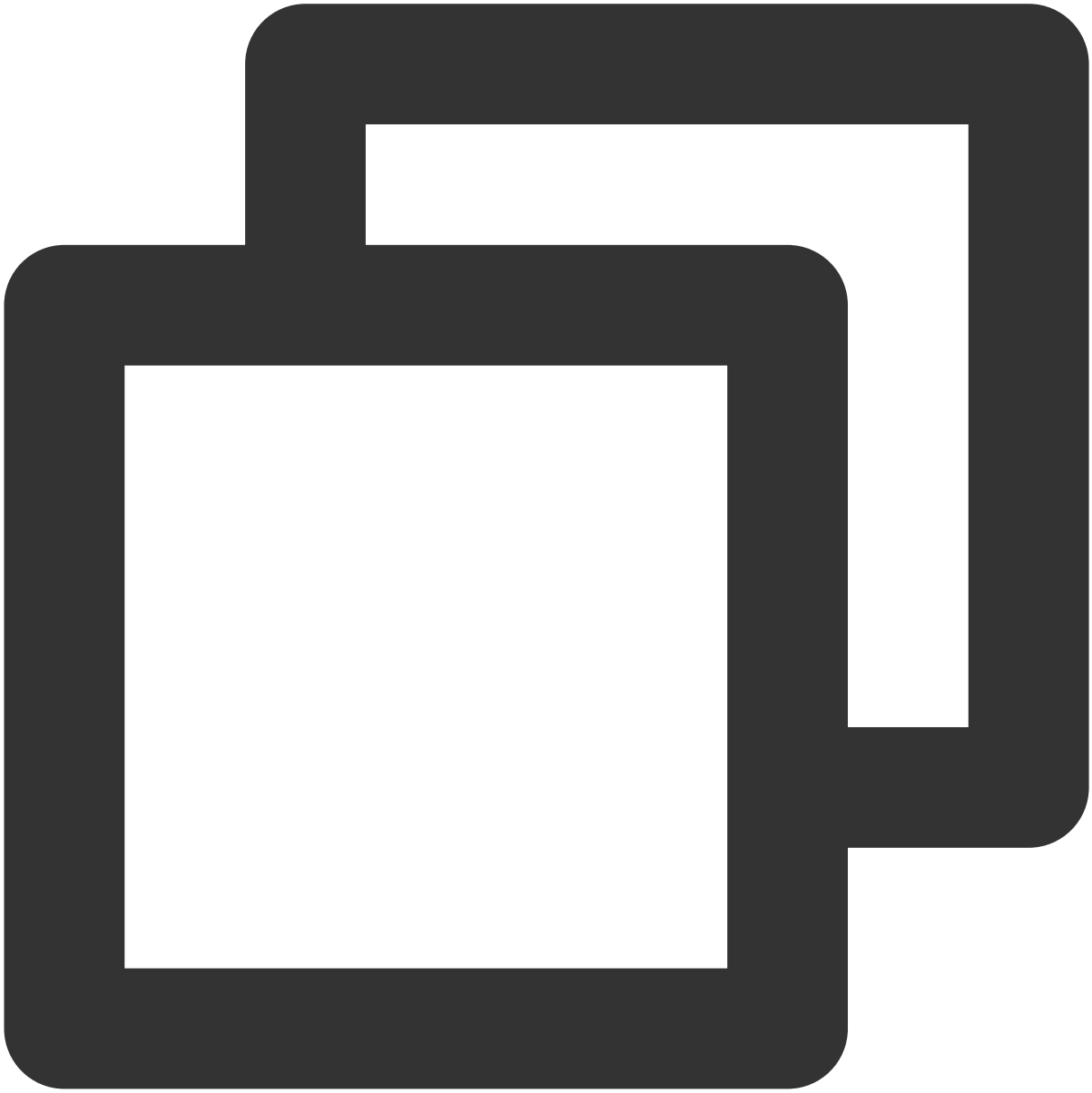
参数	类型	说明	默认值	含义

event	TUIRoomEvents	必填	-	TUIRoomEngine 事件列表
func	Function	必填	-	事件回调函数

Returns：***void***

off

取消监听 roomEngine 的事件。



```
const roomEngine = new TUIRoomEngine();
```

```
roomEngine.off(event, func);
```

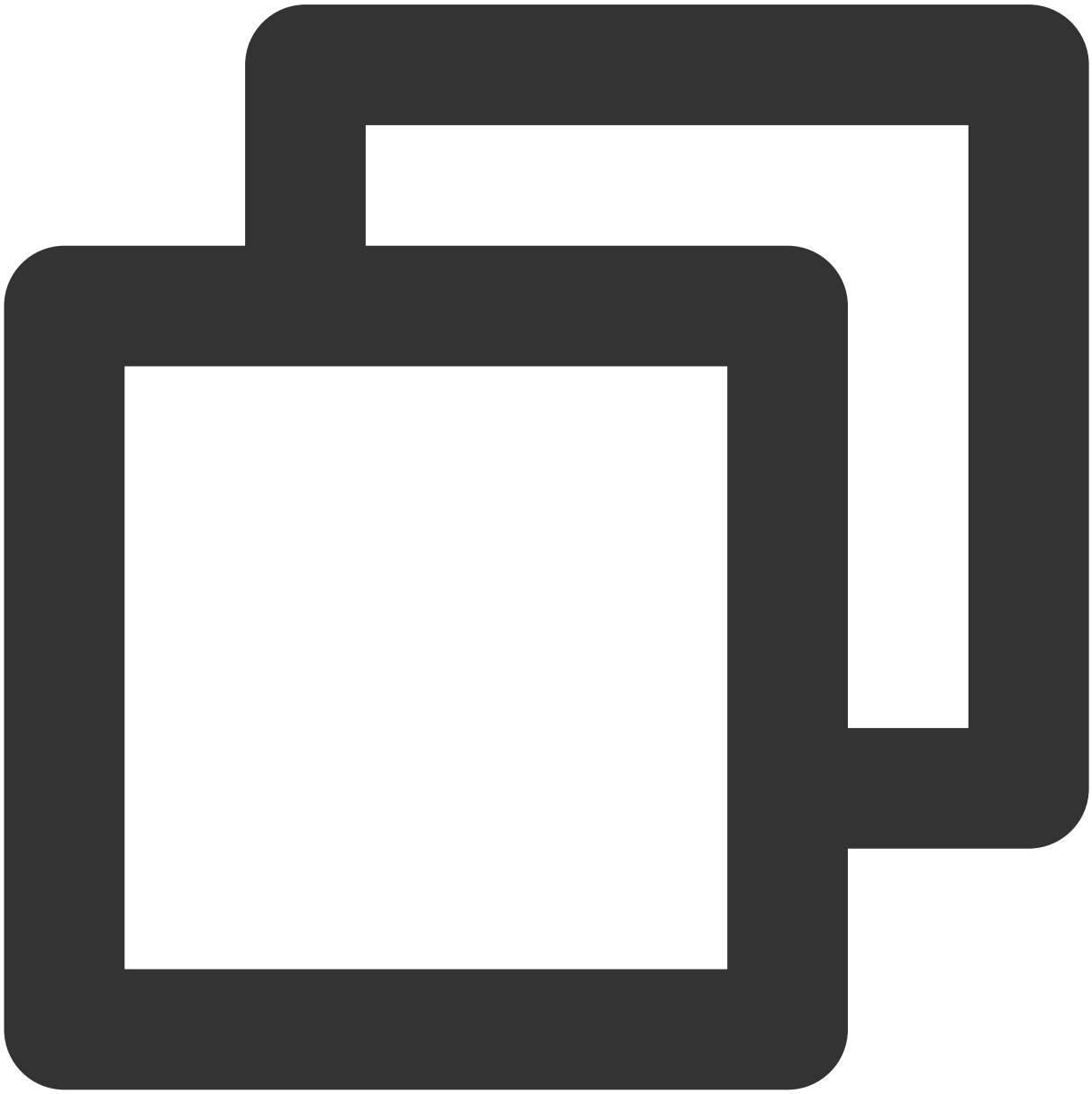
参数：

参数	类型	说明	默认值	含义
event	TUIRoomEvents	必填	-	TUIRoomEngine 事件列表
func	Function	必填	-	事件回调函数

Returns：***void***

getTRTCCloud

获取 trtcCloud 实例，web 端 trtcCloud 能力请查看：TRTCCloud API 文档。

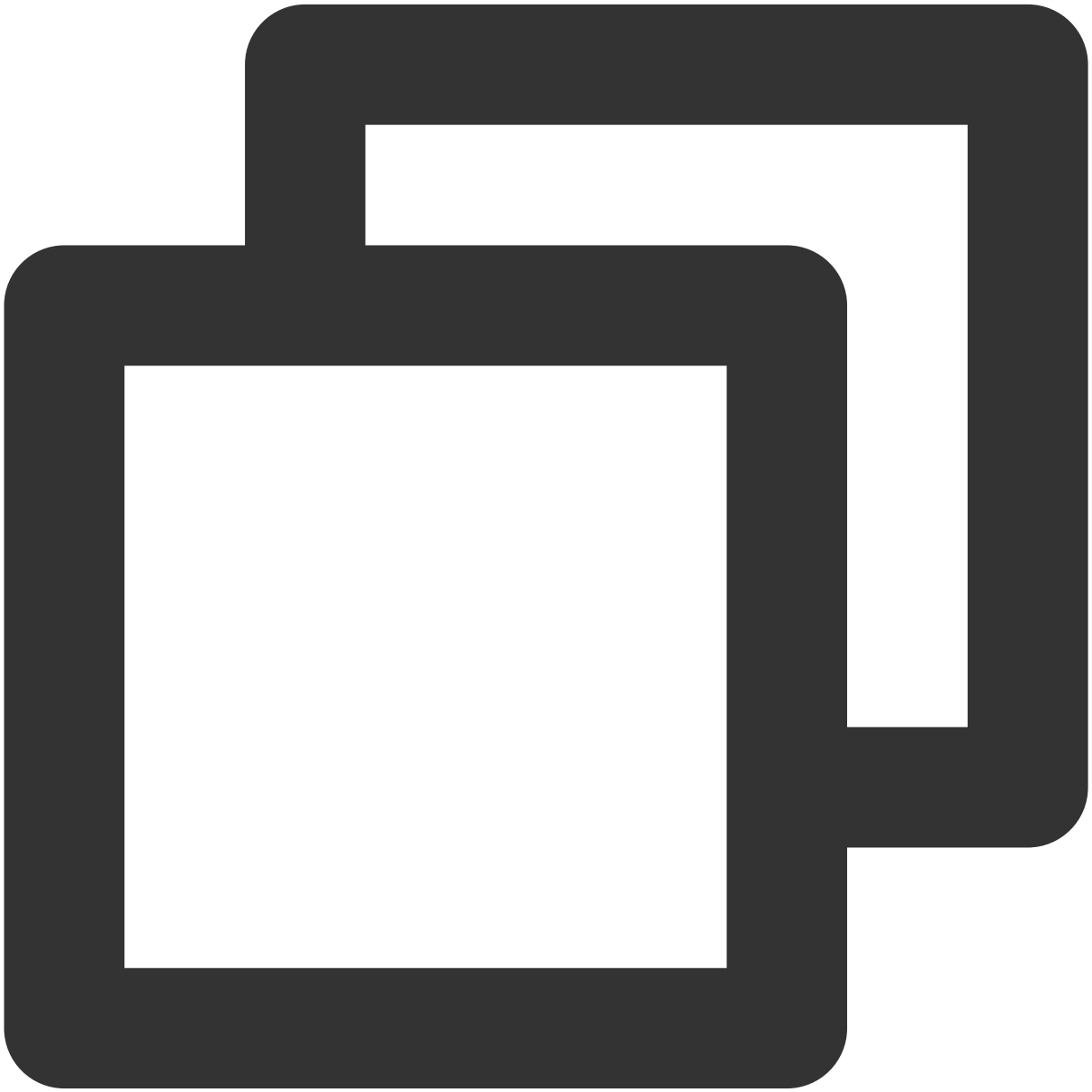


```
const roomEngine = new TUIRoomEngine();
const trtcCloud = roomEngine.getTRTCCloud();
```

Returns：TRTCCloud

getTIM

获取 tim 实例，web 端 tim 能力请查看： IM API 文档。



```
const roomEngine = new TUIRoomEngine();
const trtcCloud = roomEngine.getTIM();
```

Returns：*TIM*

TUIRoomEvent

最近更新时间：2023-11-22 11:26:26

TUIRoomEvent API 简介

TUIRoomEvent API 是多人组件的**事件接口**。

事件列表

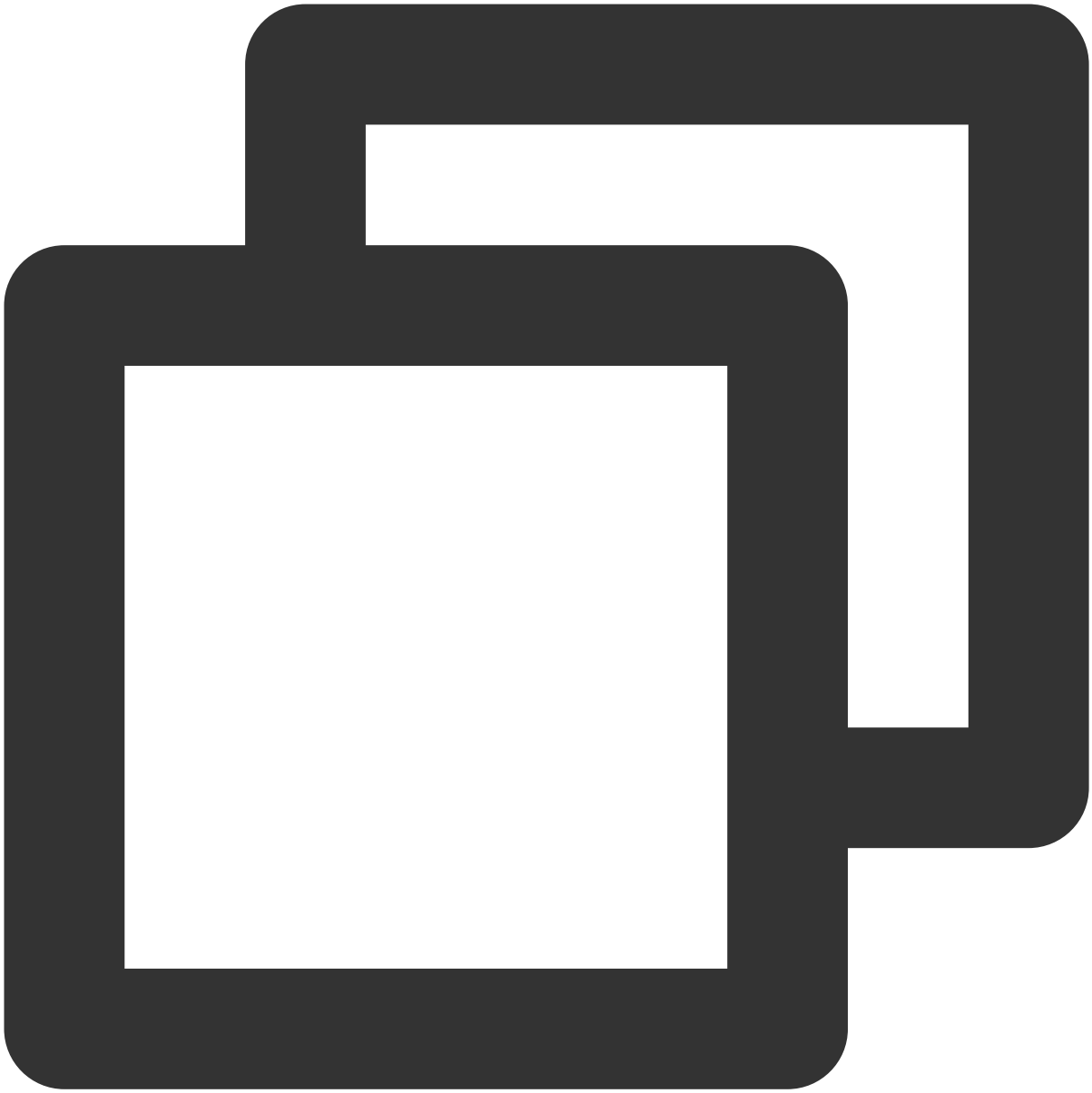
EVENT	描述
TUIRoomEvents.onError	错误事件
TUIRoomEvents.onKickedOutOfRoom	踢出房间事件
TUIRoomEvents. onKickedOffLine	当前用户被踢下线
TUIRoomEvents.onUserSigExpired	userSig 过期事件
TUIRoomEvents.onRoomDismissed	主持人销毁房间事件
TUIRoomEvents.onRoomNameChanged	房间名称修改事件
TUIRoomEvents.onRoomSpeechModeChanged	房间发言模式修改事件
TUIRoomEvents.onAllUserCameraDisableChanged	所有成员摄像头使用权限改变事件
TUIRoomEvents.onAllUserMicrophoneDisableChanged	所有成员麦克风使用权限改变事件
TUIRoomEvents.onSendMessageForAllUserDisableChanged	所有成员发送消息状态改变事件
TUIRoomEvents.onRoomMaxSeatCountChanged	房间最大麦位数修改事件
TUIRoomEvents.onRemoteUserEnterRoom	远端用户进入房间事件
TUIRoomEvents.onRemoteUserLeaveRoom	远端用户离开房间事件
TUIRoomEvents.onUserRoleChanged	用户角色改变事件
TUIRoomEvents.onUserVideoStateChanged	用户视频状态改变事件
TUIRoomEvents.onUserAudioStateChanged	用户音频状态改变事件

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEvents.onSendMessageForUserDisableChanged	用户发送消息状态事件
TUIRoomEvents.onUserVoiceVolumeChanged	用户音量改变事件
TUIRoomEvents.onUserNetworkQualityChanged	用户网络质量改变事件
TUIRoomEvents.onSeatListChanged	麦位列表改变事件
TUIRoomEvents.onKickedOffSeat	用户被踢下麦事件
TUIRoomEvents.onRequestReceived	请求接收事件
TUIRoomEvents.onRequestCancelled	请求取消事件
TUIRoomEvents.onReceiveTextMessage	接收文本消息事件
TUIRoomEvents.onReceiveCustomMessage	接收自定义消息事件
TUIRoomEvents.onDeviceChange	设备变更事件
TUIRoomEvents.onUserScreenCaptureStopped	屏幕分享停止事件，当用户使用浏览器自带的【停止分享】按钮结束屏幕共享时，该用户会收到 'onUserScreenCaptureStopped' 事件用来修改屏幕共享状态。

onError

错误事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onError, (error) => {
 console.log('TUIRoomError error', error);
})
```

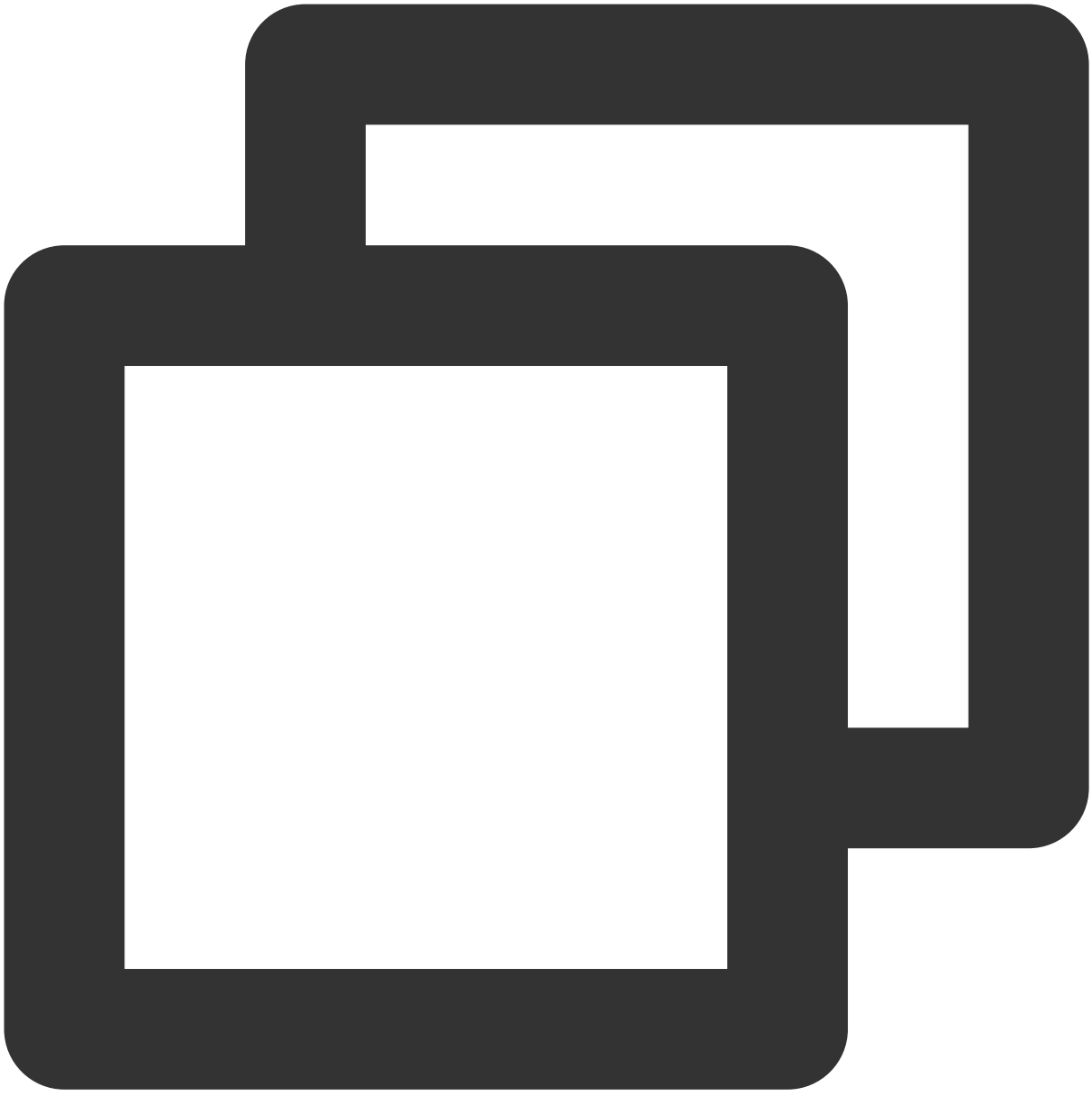
参数如下表所示：

参数	类型	含义
code	number	错误代码

message	string	错误信息

onKickedOutOfRoom

踢出房间事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOutOfRoom, ({ roomId, message }) => {
  console.log('roomEngine.onKickedOutOfRoom', roomId, message);
```

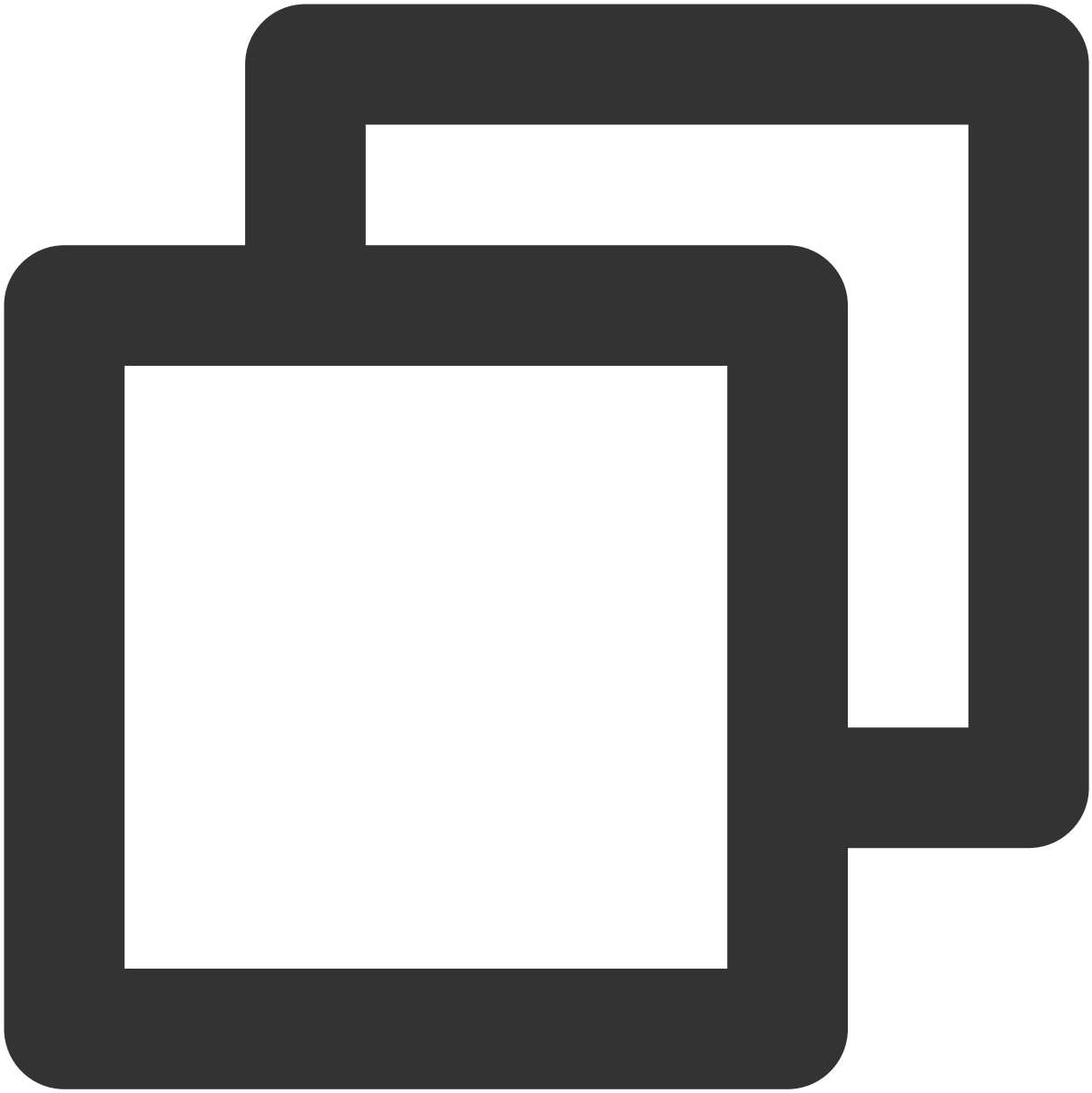
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
message	string	踢出房间信息

onKickedOffLine

当前用户被踢下线



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onKickedOffLine, ({ message }) => {
  console.log('roomEngine.onKickedOffLine', message);
});
```

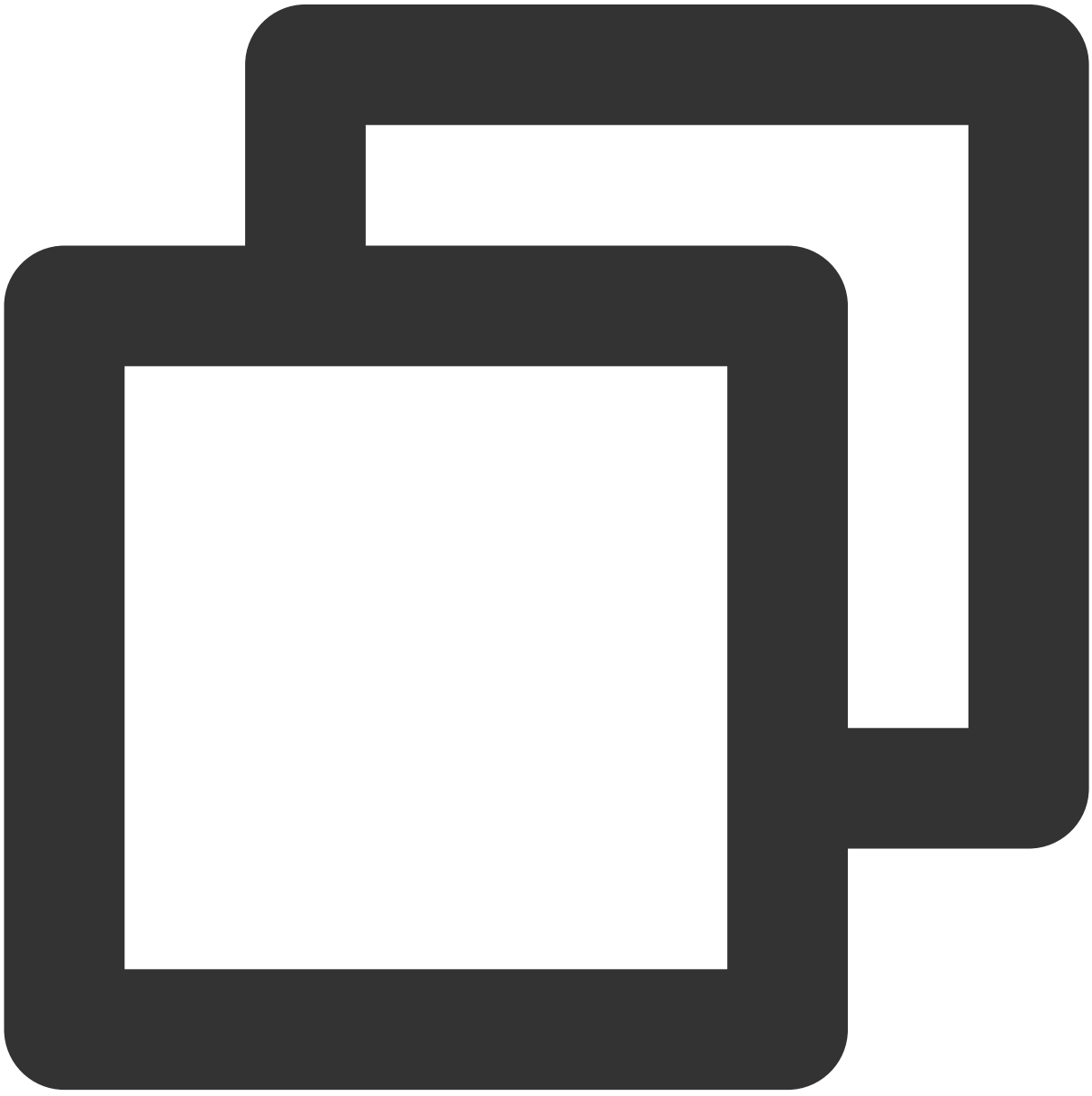
参数如下表所示：

参数	类型	含义
roomId	string	房间号

message	string	用户在其他端登录信息

onUserSigExpired

userSig 过期事件

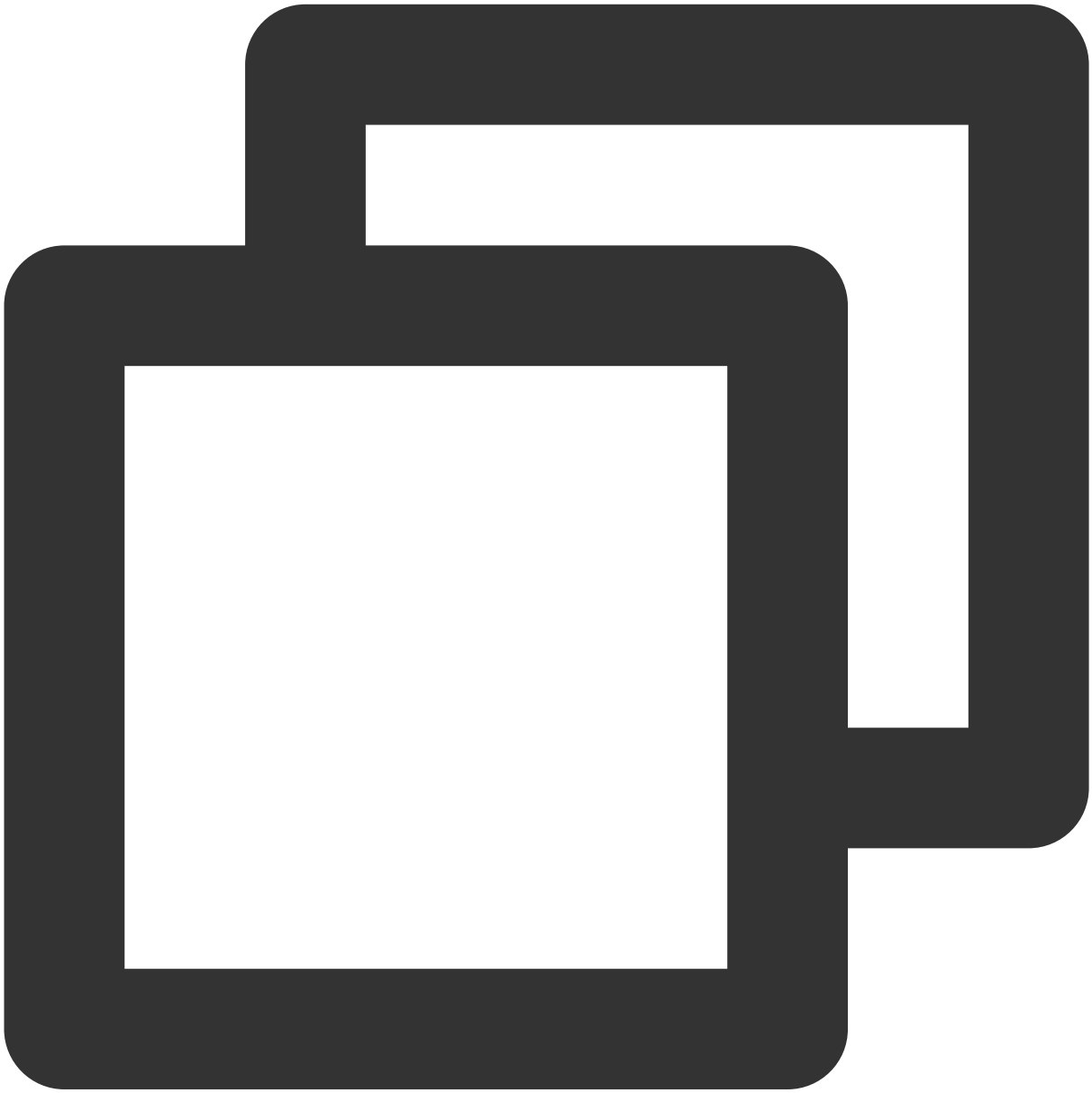


```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserSigExpired, () => {
  console.log('roomEngine.onUserSigExpired');
```

```
});
```

onRoomDismissed

主持人销毁房间事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomDismissed, ({ roomId }) => {
  console.log('roomEngine.onRoomDismissed', roomId);
```

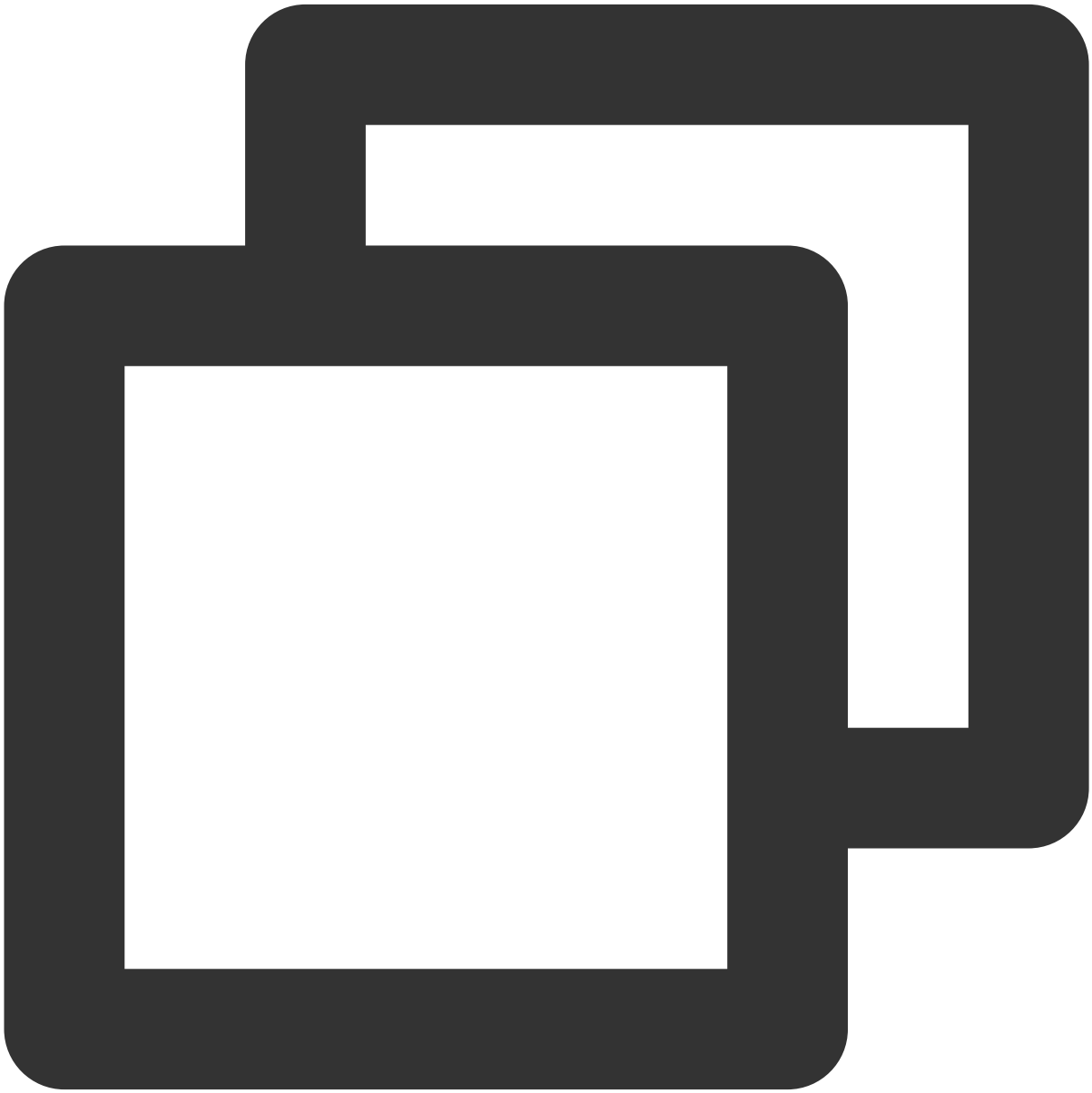
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号

onRoomNameChanged

房间信息修改事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomNameChanged, ({ roomId, roomName }) => {
  console.log('roomEngine.onRoomNameChanged', roomId, roomName);
});
```

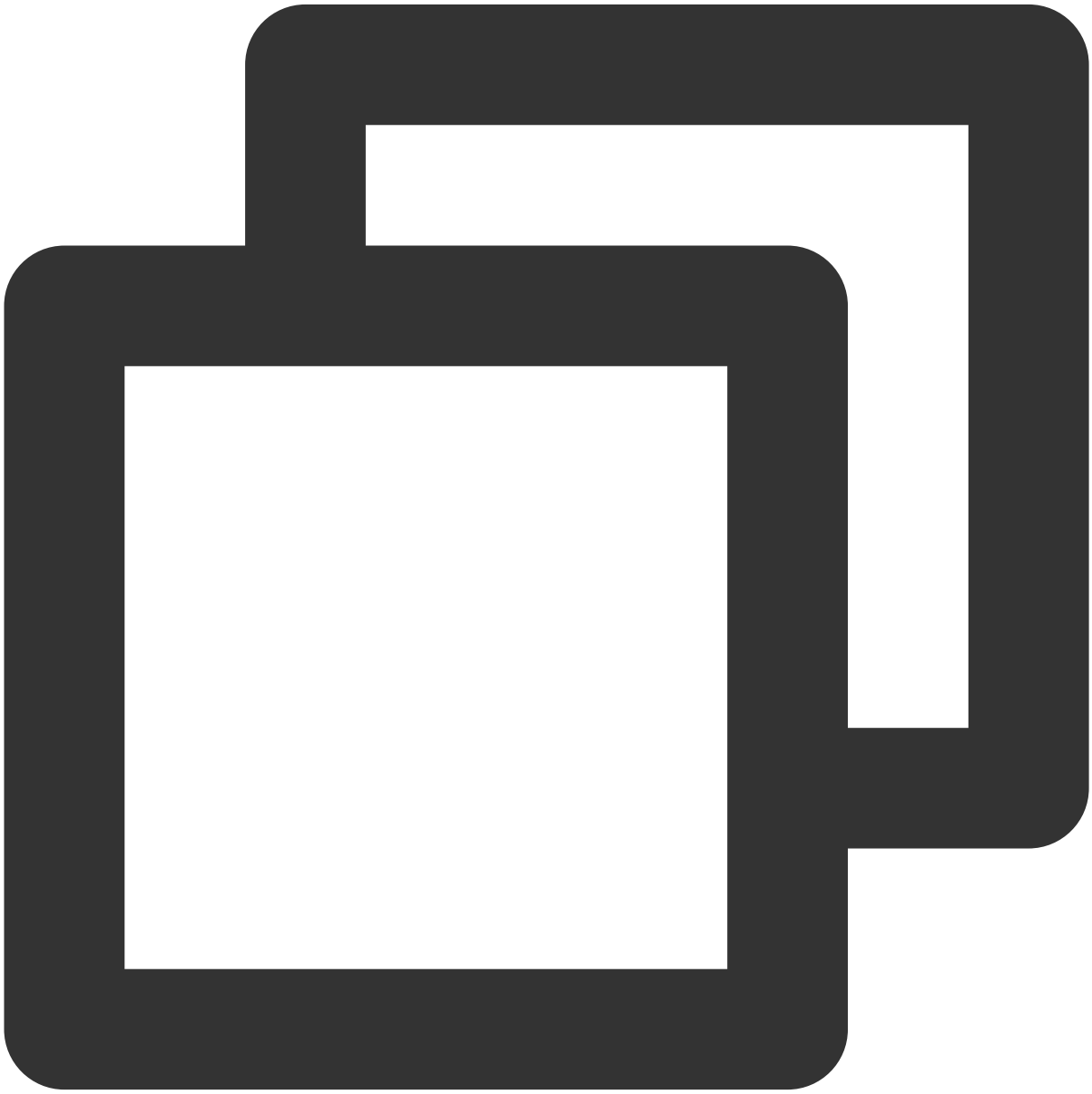
参数如下表所示：

参数	类型	含义
roomId	string	房间号

roomName	string	房间名称

onRoomSpeechModeChanged

房间名称修改事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomSpeechModeChanged, ({ roomId, speechMode }) => {
  console.log('roomEngine.onRoomSpeechModeChanged', roomId, speechMode);
```

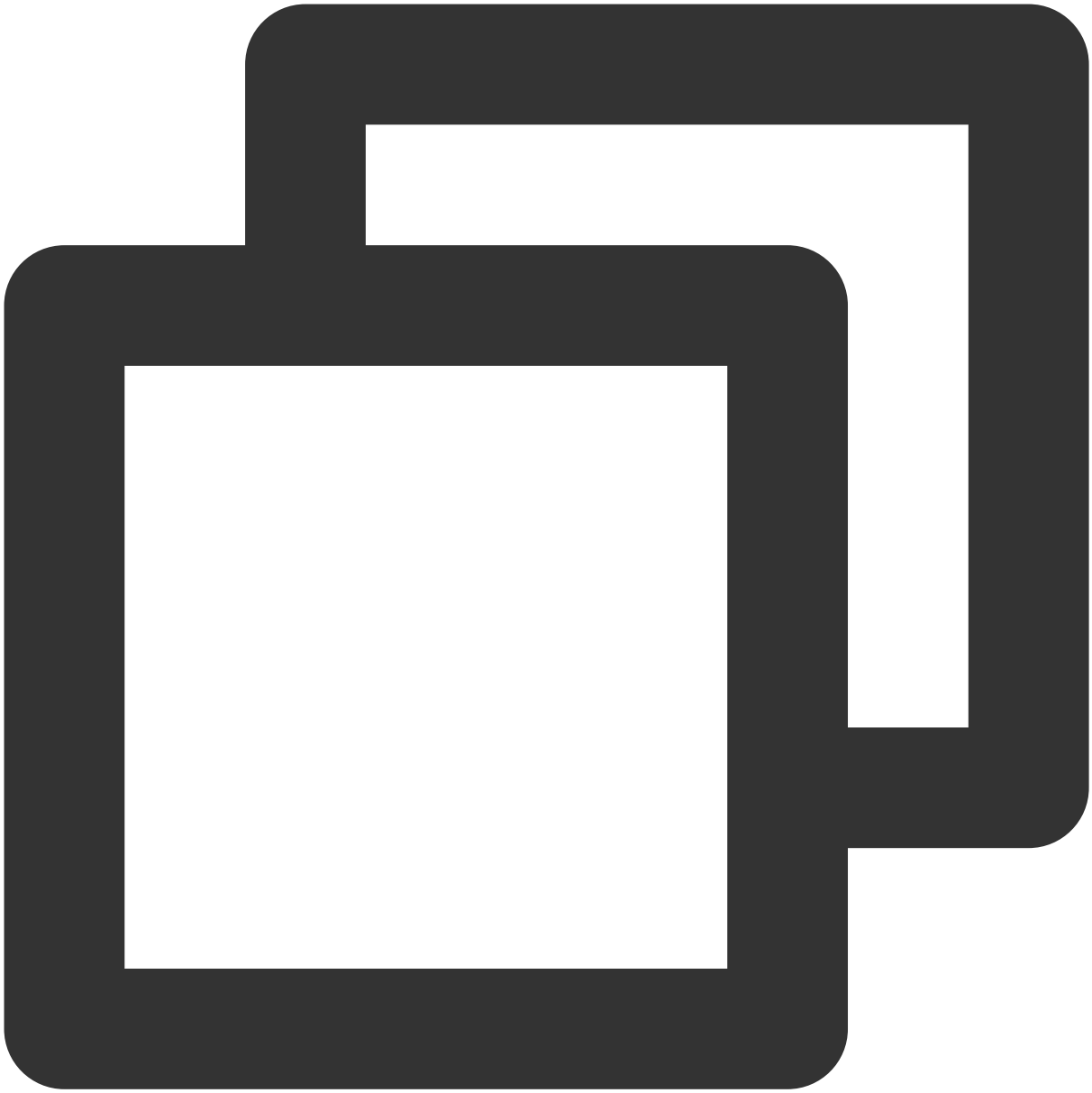
```
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
speechMode	TUISpeechMode	发言模式

onAllUserCameraDisableChanged

所有成员摄像头使用权限改变事件



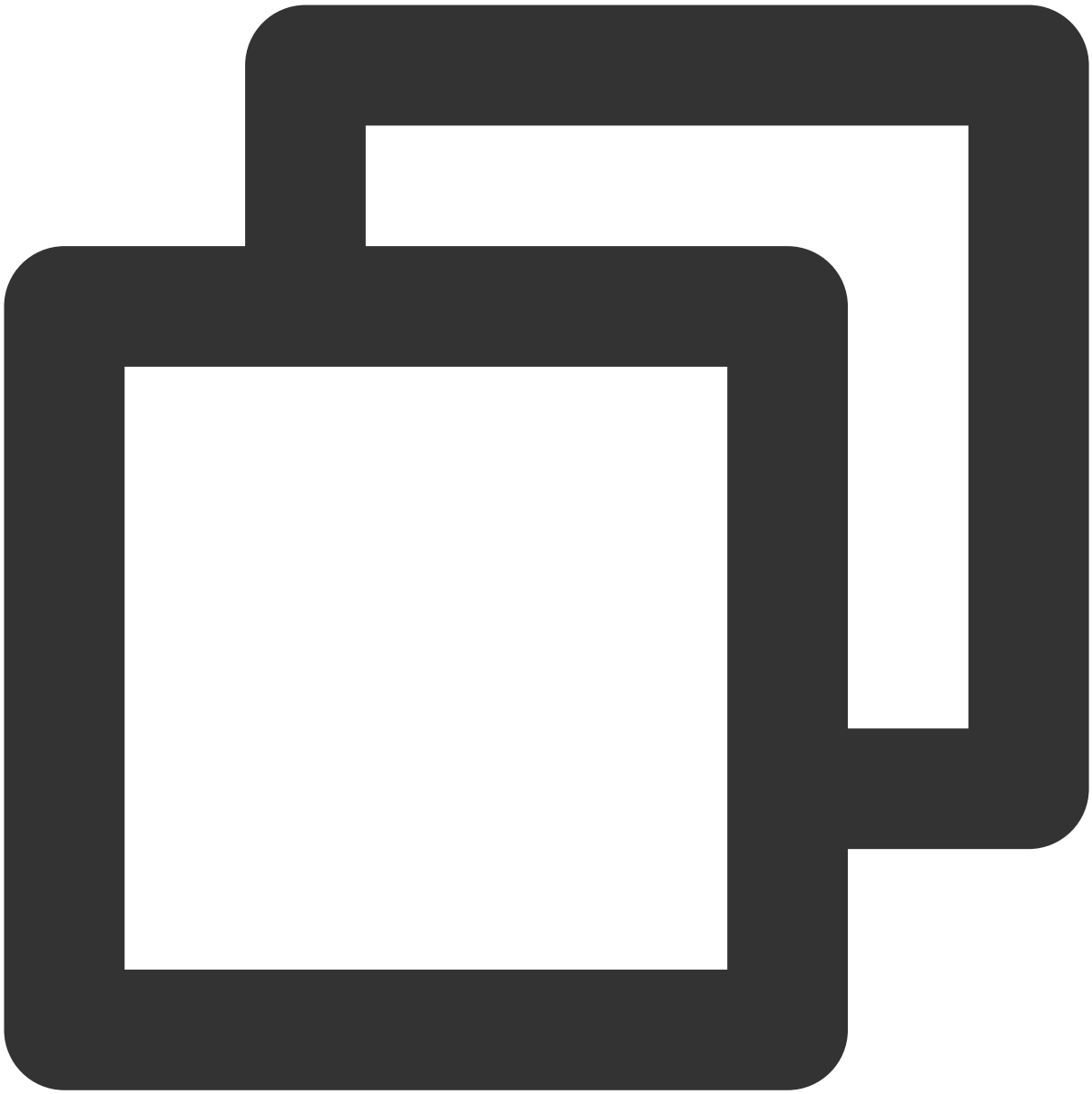
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserCameraDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserCameraDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许使用摄像头

onAllUserMicrophoneDisableChanged

所有成员麦克风使用权限改变事件



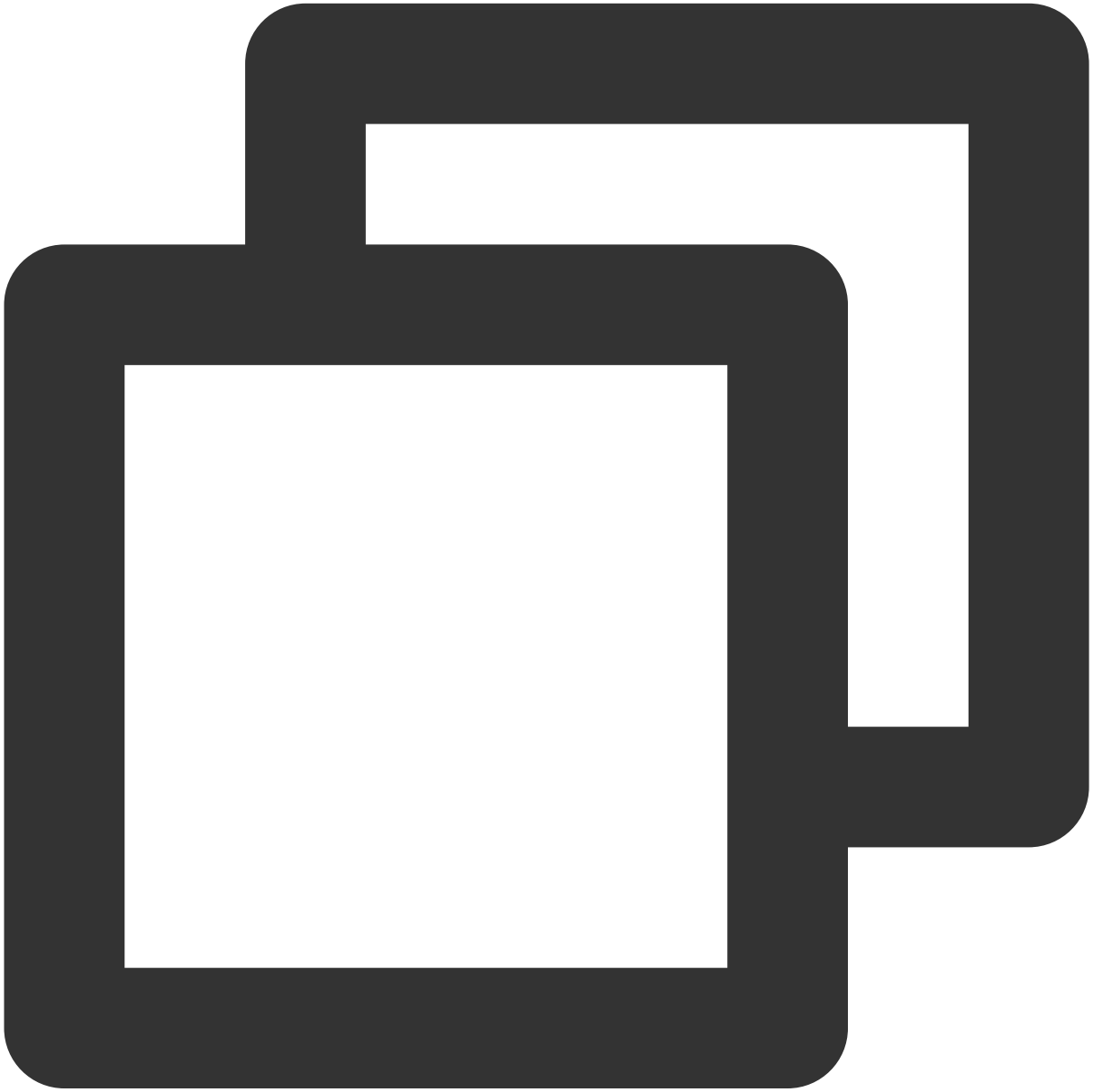
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onAllUserMicrophoneDisableChanged, ({ isDisable }) => {
  console.log('roomEngine.onAllUserMicrophoneDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许使用麦克风

onSendMessageForAllUserDisableChanged

所有成员发送消息权限改变事件



```
const roomEngine = new TUIRoomEngine();
```

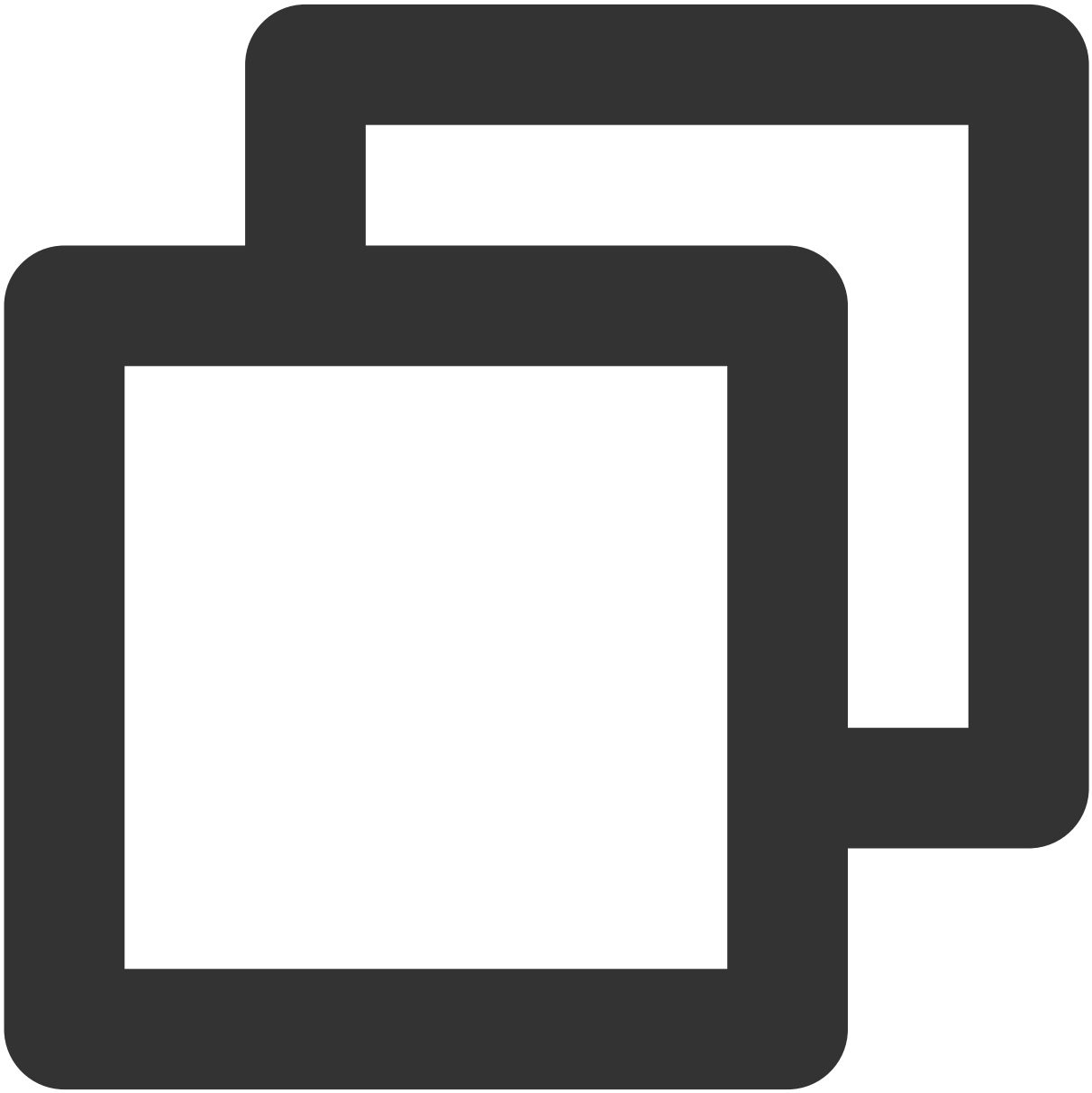
```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable }) =>
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许发送文字消息

onRoomMaxSeatCountChanged

房间最大麦位数修改事件



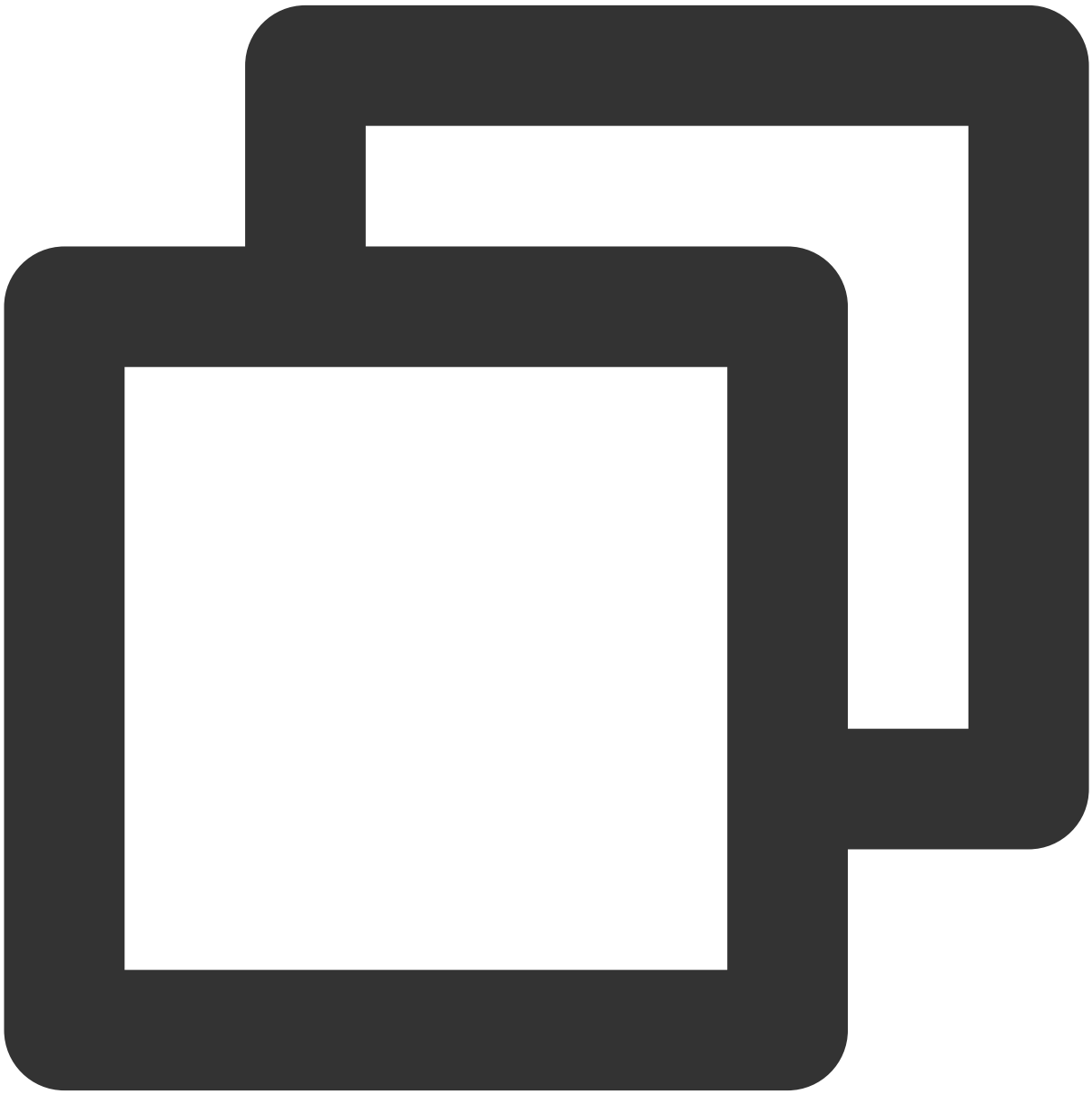
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRoomMaxSeatCountChanged, ({ maxSeatNumber }) => {
  console.log('roomEngine.onRoomMaxSeatCountChanged', maxSeatNumber);
});
```

参数如下表所示：

参数	类型	含义
maxSeatNumber	number	最大麦位数量

onRemoteUserEnterRoom

远端用户进入房间事件



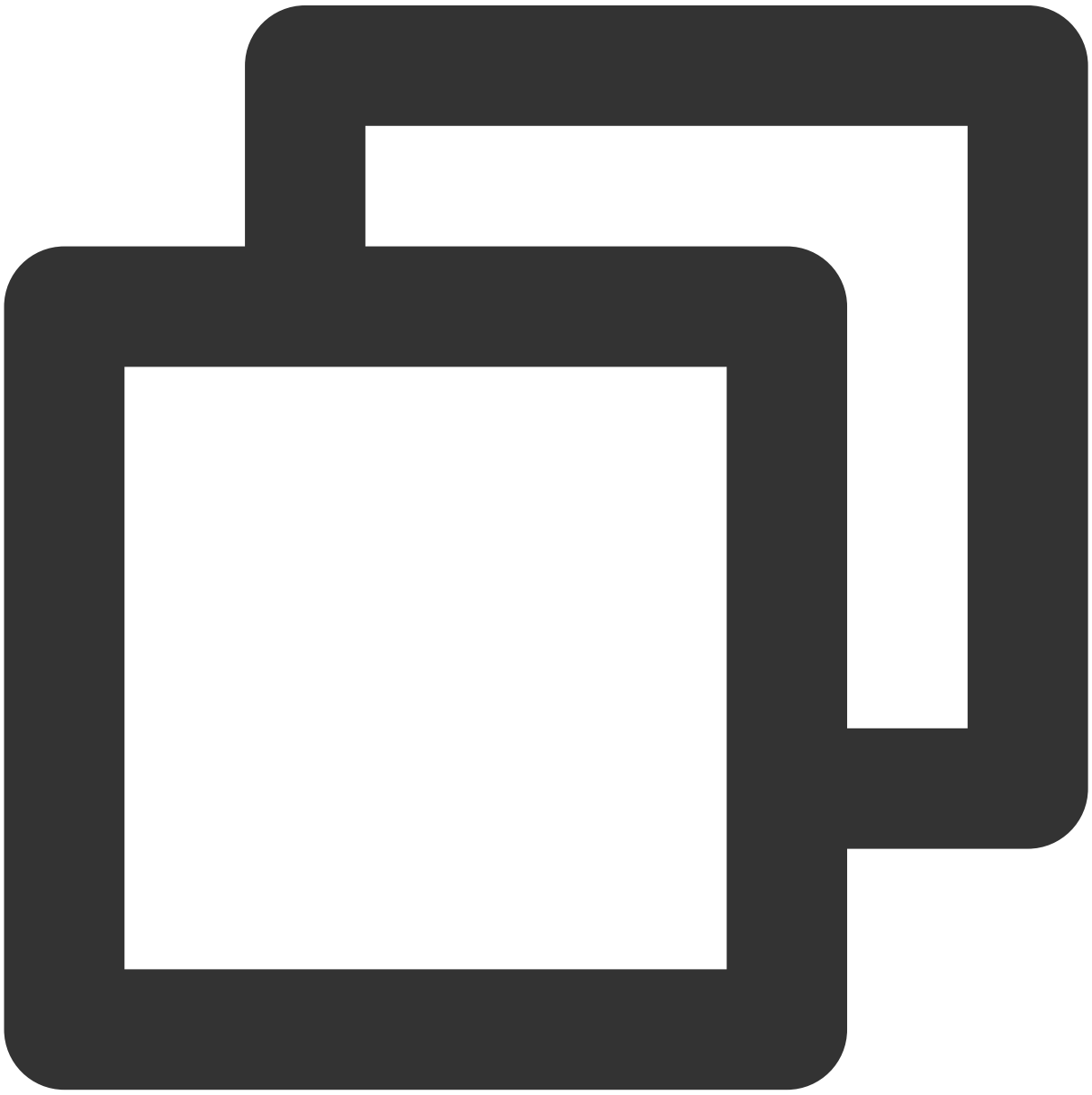
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserEnterRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserEnterRoom', roomId, userInfo);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
userInfo	TUIUserInfo	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件



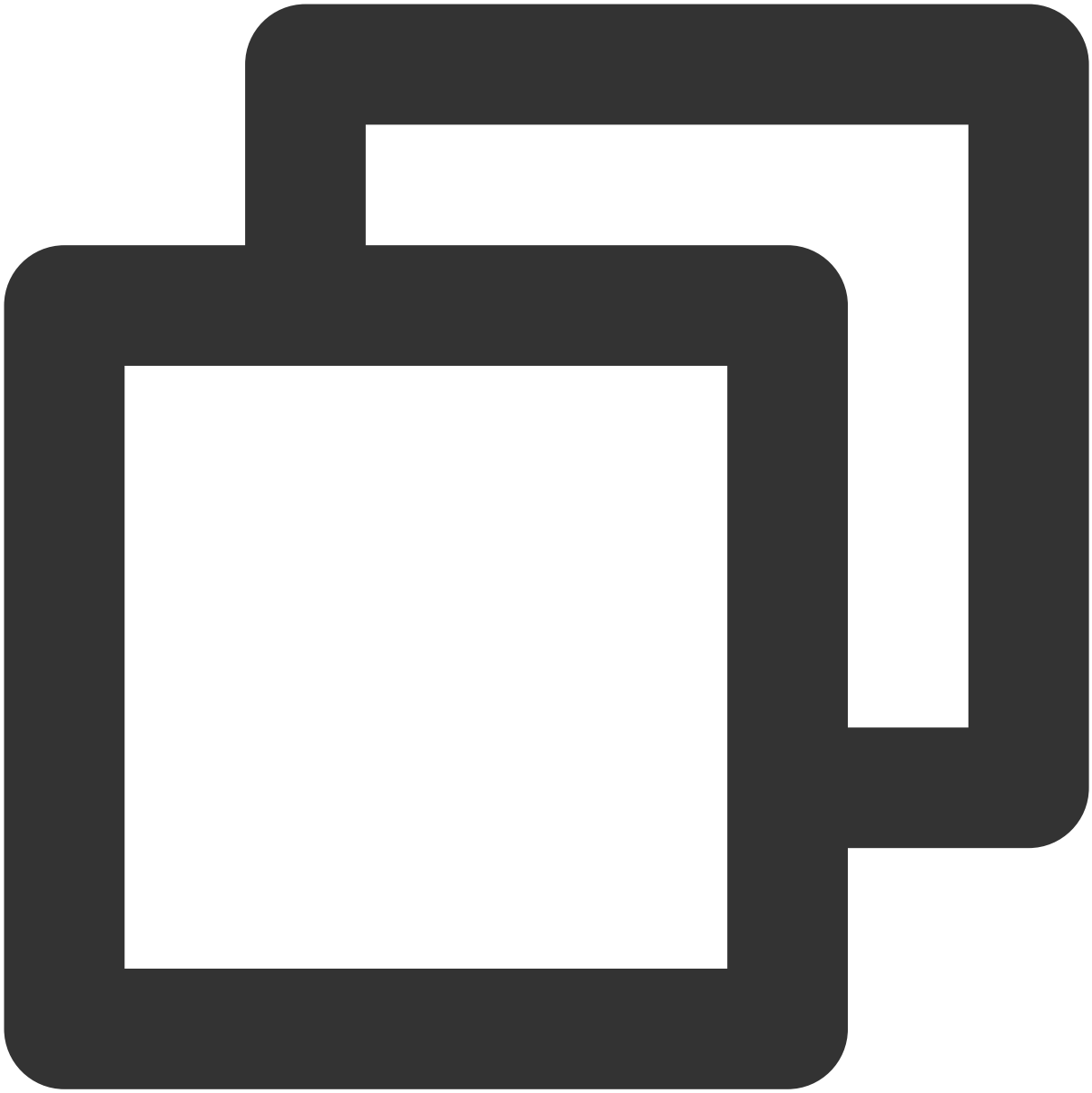
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRemoteUserLeaveRoom, ({ roomId, userInfo }) => {
  console.log('roomEngine.onRemoteUserLeaveRoom', roomId, userInfo);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间号
userInfo	TUIUserInfo	用户信息

onUserRoleChanged

用户角色改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserRoleChanged, ({ userId, userRole }) => {
  console.log('roomEngine.onUserRoleChanged', userId, userRole);
});
```

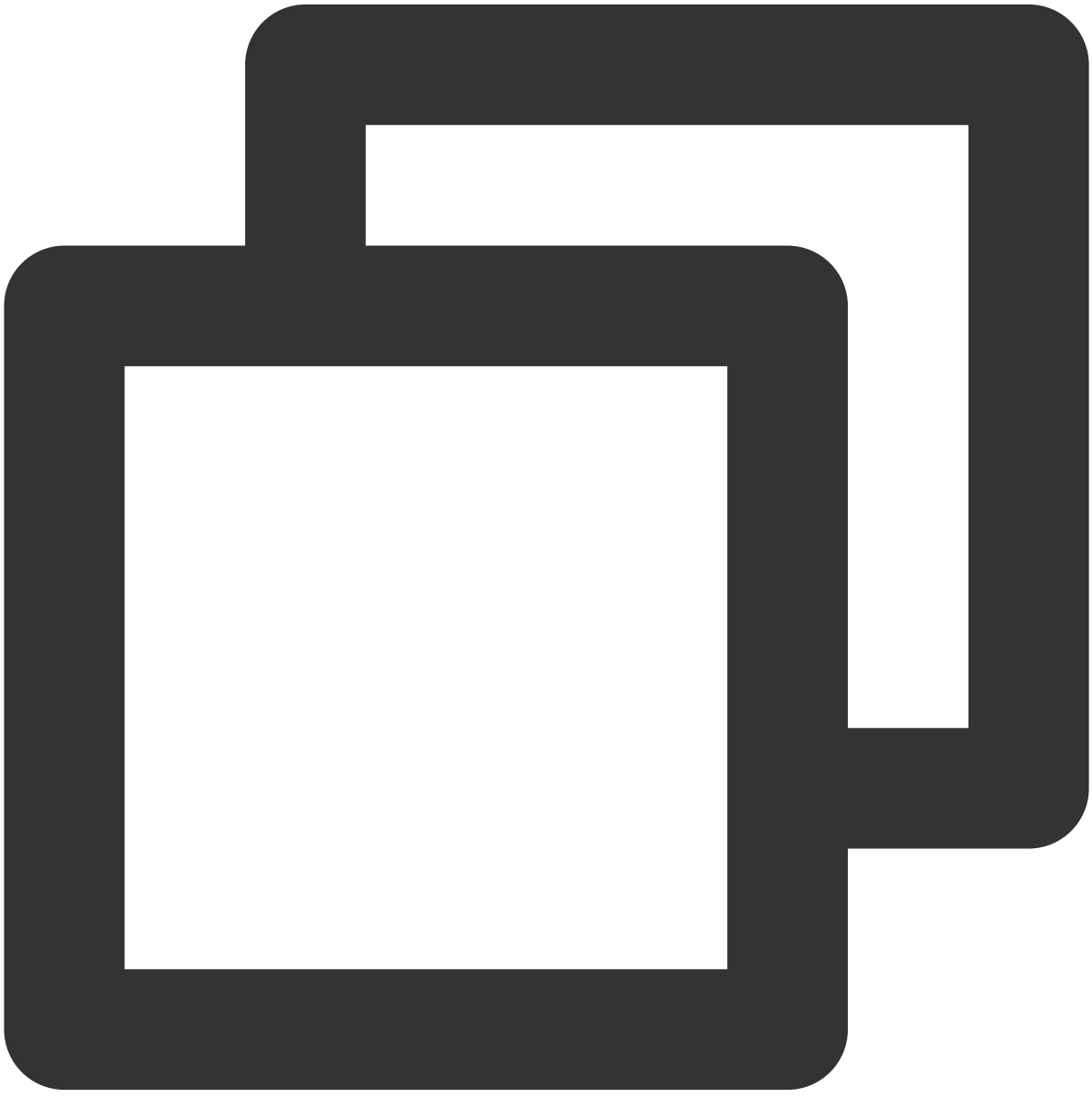
参数如下表所示：

参数	类型	含义
userId	string	用户 ID

userRole	TUIRole	用户变更后的角色

onUserVideoStateChanged

用户视频状态改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVideoStateChanged, ({ userId, streamType, hasVide
  console.log('roomEngine.onUserVideoStateChanged', userId, streamType, hasVideo, r
```

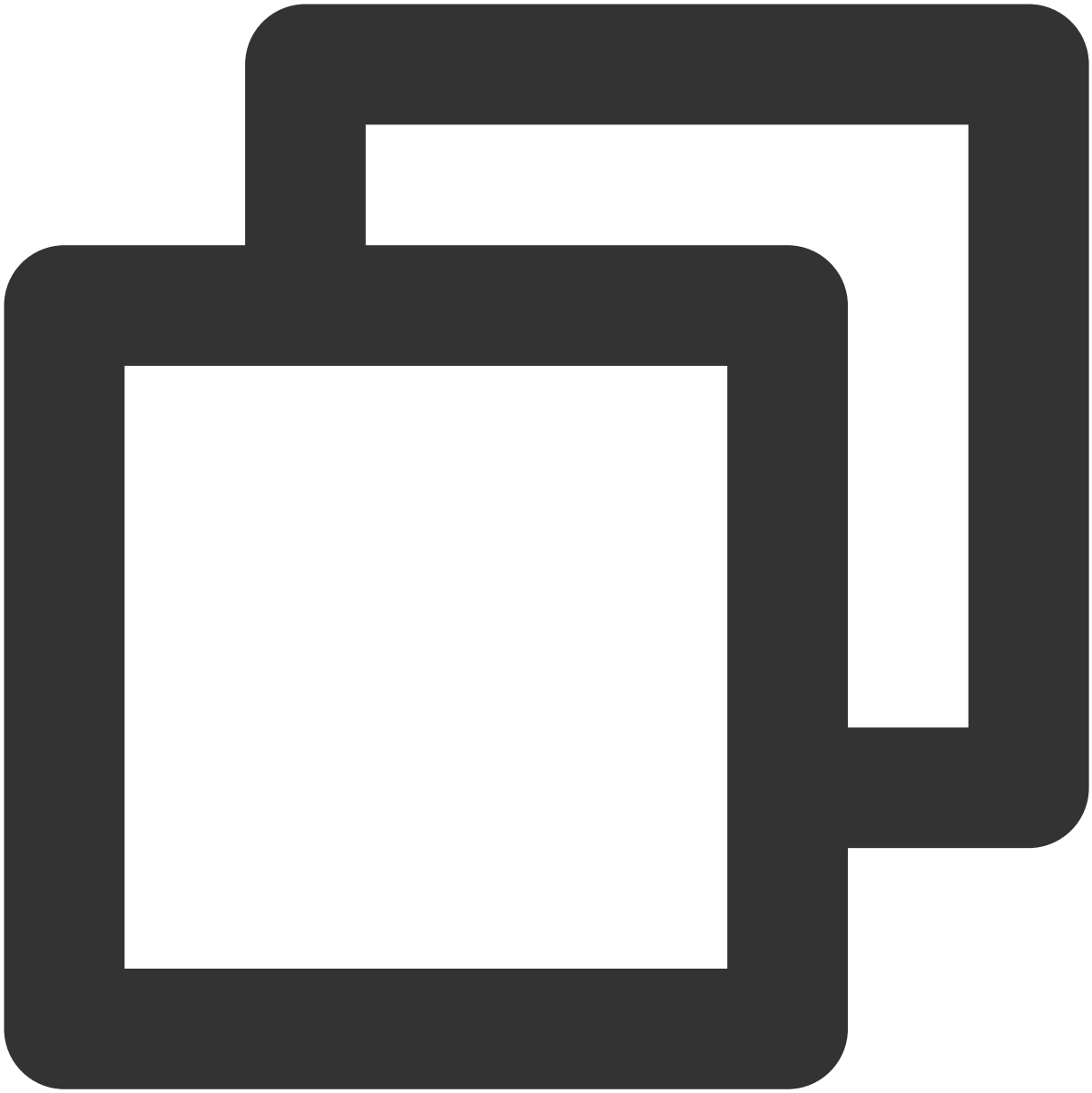
```
});
```

参数如下表所示：

参数	类型	含义
userId	string	用户 ID
streamType	TUIVideoStreamType	用户流类型
hasVideo	boolean	是否有视频流
reason	TUIChangeReason	变更原因, 自己操作/主持人操作

onUserAudioStateChanged

用户音频状态改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserAudioStateChanged, ({ userId, hasAudio, reason }) =>
  console.log('roomEngine.onUserAudioStateChanged', userId, hasAudio, reason);
});
```

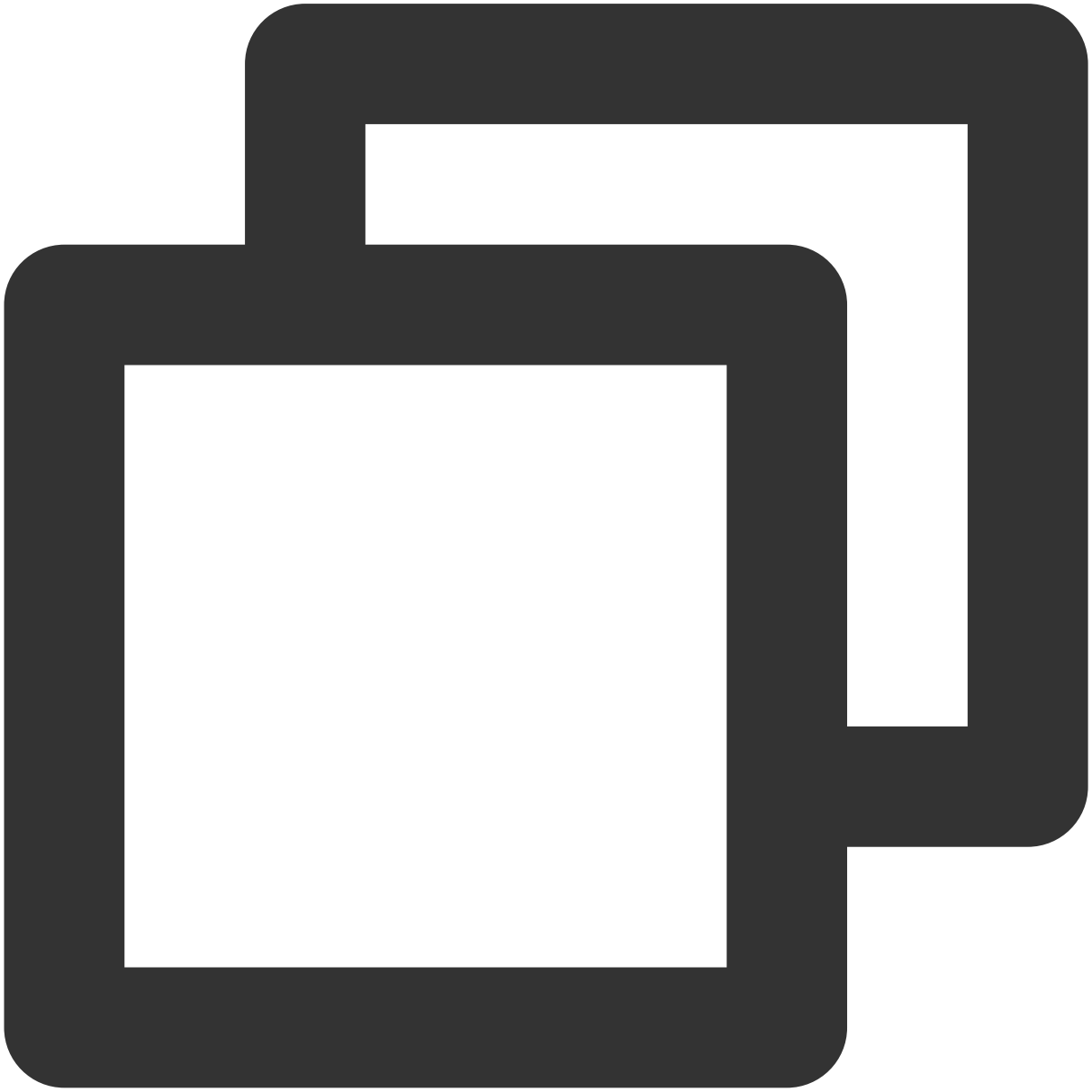
参数如下表所示：

参数	类型	含义
userId	string	用户 ID

hasVideo	boolean	是否有音频流
reason	TUIChangeReason	变更原因，自己操作/主持人操作

onSendMessageForUserDisableChanged

有成员摄像头使用权限修改事件



```
const roomEngine = new TUIRoomEngine();
```

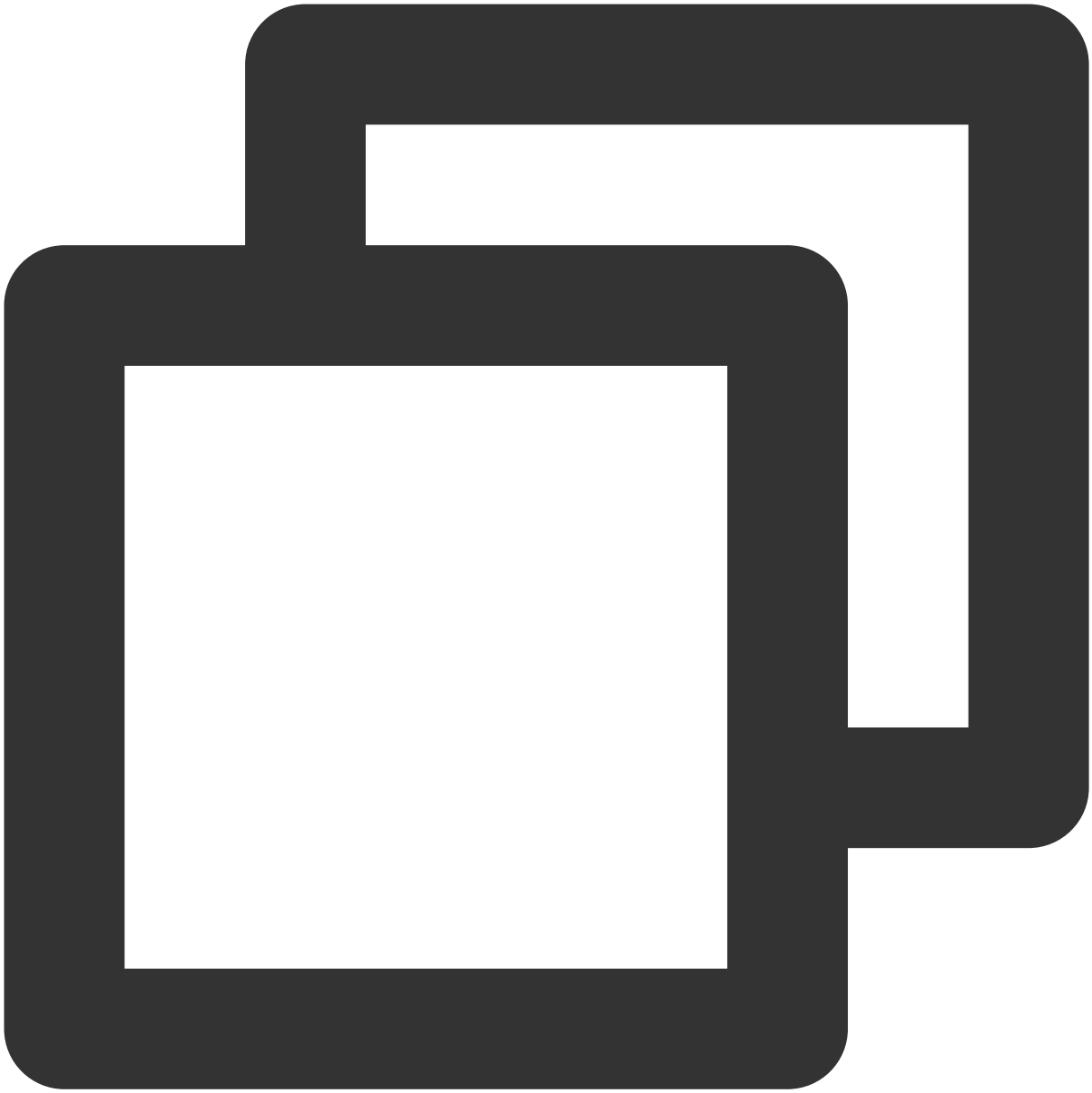
```
roomEngine.on(TUIRoomEvents.onSendMessageForAllUserDisableChanged, ({ isDisable }) =>
  console.log('roomEngine.onSendMessageForAllUserDisableChanged', isDisable);
});
```

参数如下表所示：

参数	类型	含义
isDisable	boolean	是否允许发送文字消息

onUserVoiceVolumeChanged

用户音量改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserVoiceVolumeChanged, ({ userVolumeList }) => {
  userVolumeList.forEach(userVolume => {
    console.log('roomEngine.onUserVoiceVolumeChanged', userVolume.userId, userVolum
  })
});
```

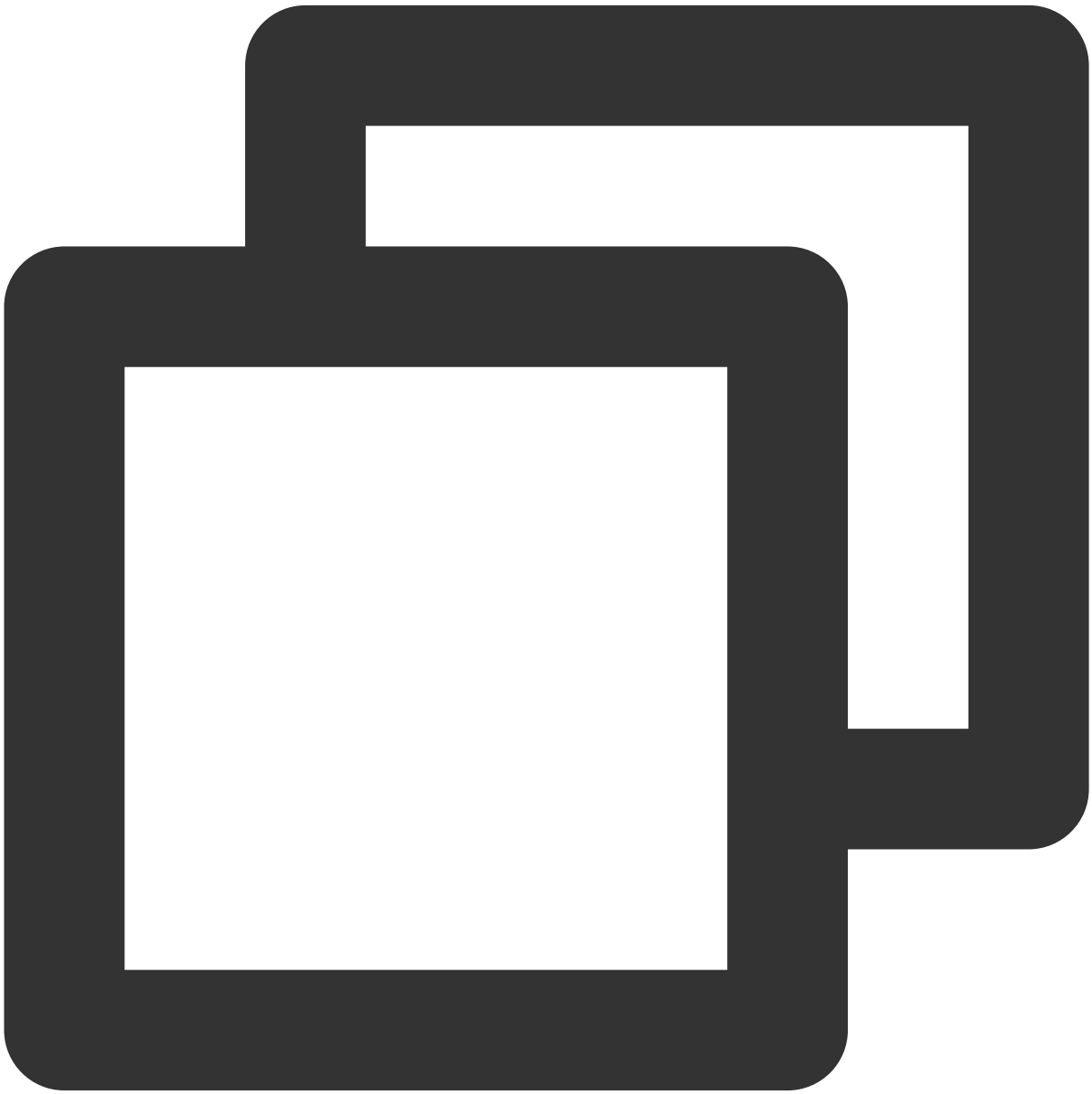
参数如下表所示：

参数	类型	含义

userVolumes	Array.<TRTCVolumeInfo>	房间内所有用户的音量, 包含 userId 及 volume 信息，volume 区间为 1～100;

onUserNetworkQualityChanged

用户网络质量改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserNetworkQualityChanged, ({ userNetworkList }) => {
```

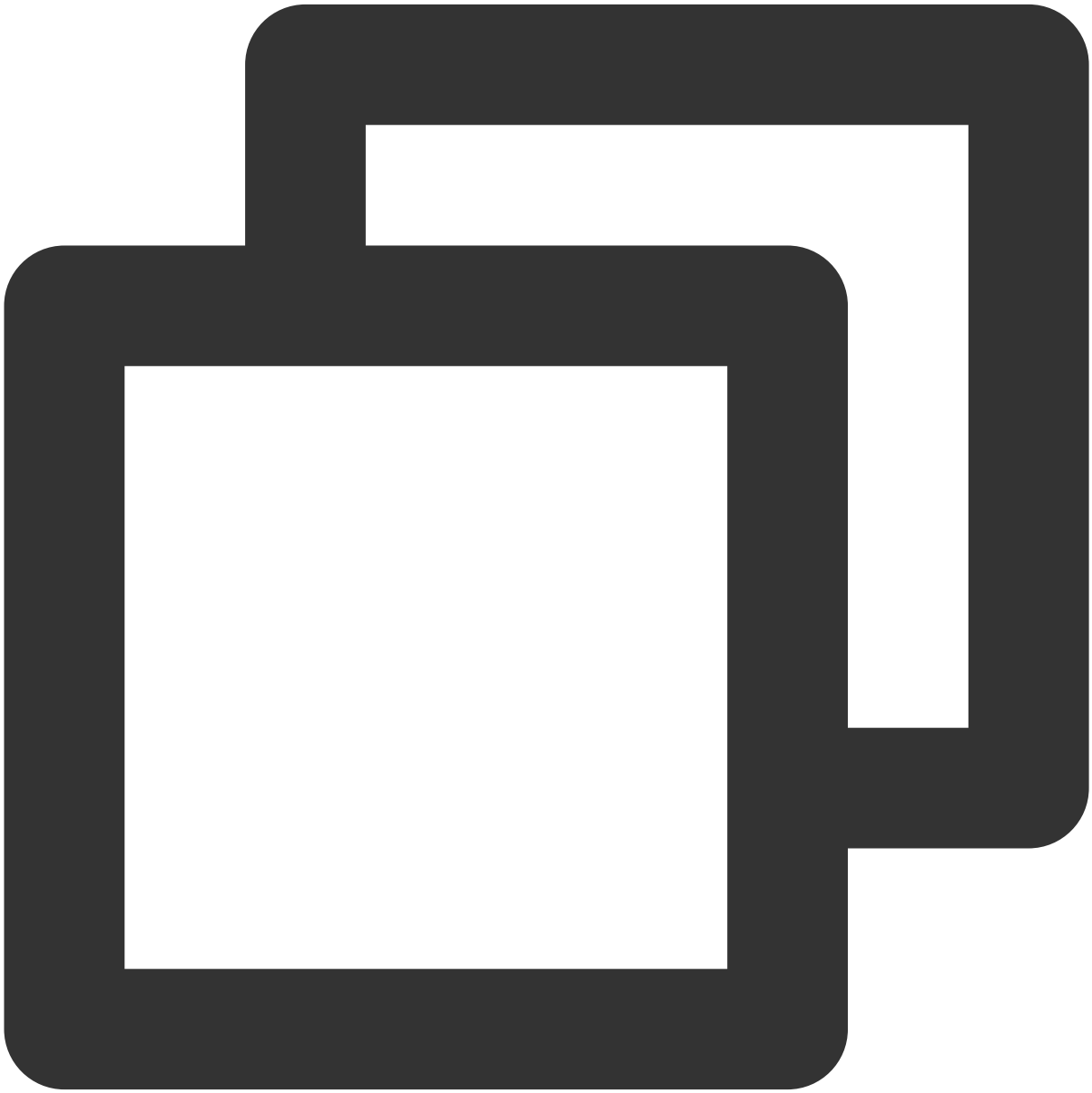
```
    userNetworkList.forEach(userNetwork => {
      console.log('roomEngine.onUserNetworkQualityChanged', userNetwork.userId, userN
    })
  });
```

参数如下表所示：

参数	类型	含义
networkMap	TUINetworkQuality	遍历网络质量等级

onSeatListChanged

麦位列表改变事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onSeatListChanged, ({ seatList, seatedList, leftList }) =>
  console.log('roomEngine.onSeatListChanged',seatList, seatedList, leftList);
});
```

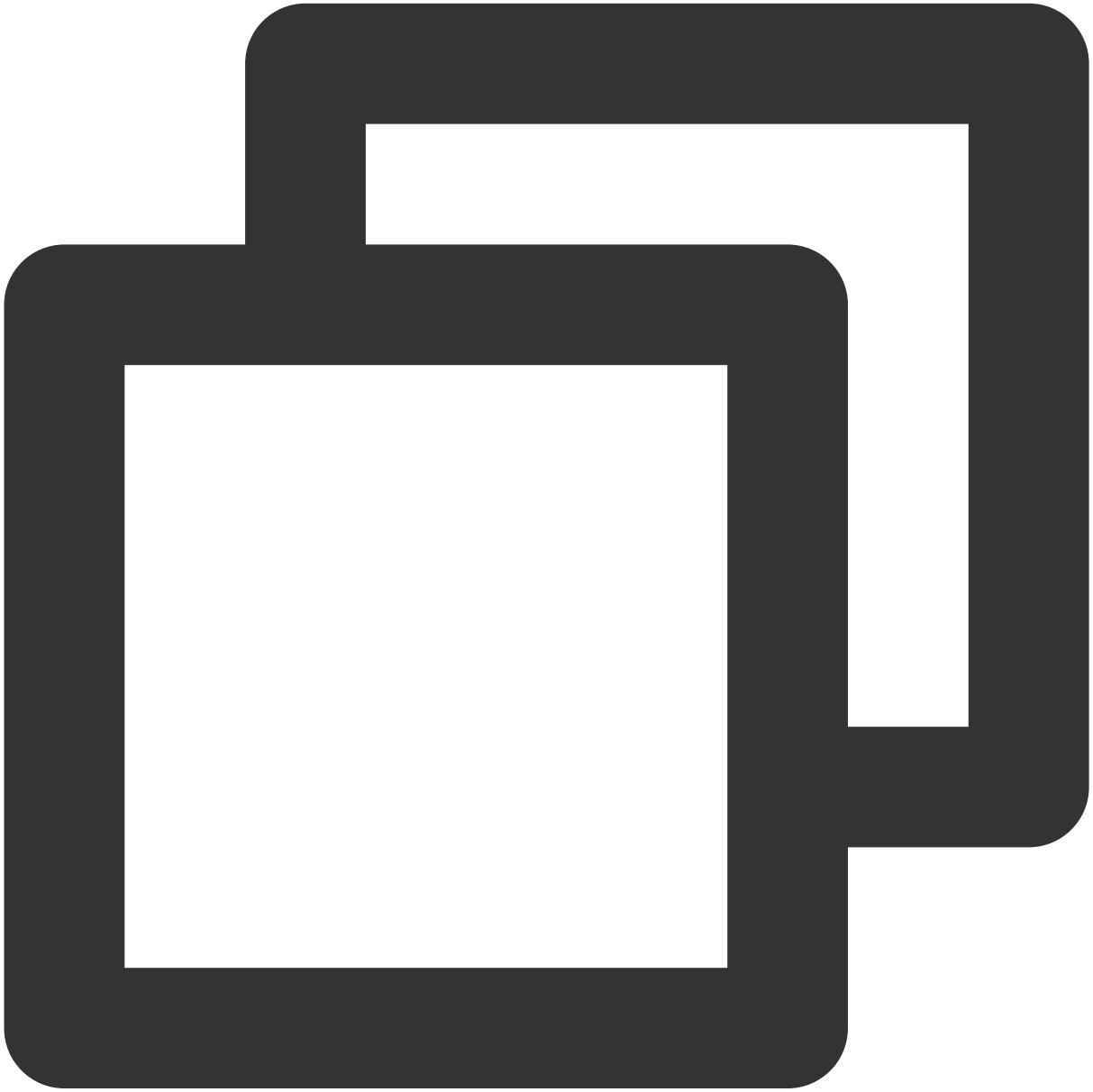
参数如下表所示：

参数	类型	含义
seatList	Array.<TUISeatInfo>	麦位列表

seatedList	Array.<TUISeatInfo>	新增麦位信息
leftList	Array.<TUISeatInfo>	离开的麦位信息

onKickedOffSeat

麦位列表改变事件



```
const roomEngine = new TUIRoomEngine();
```

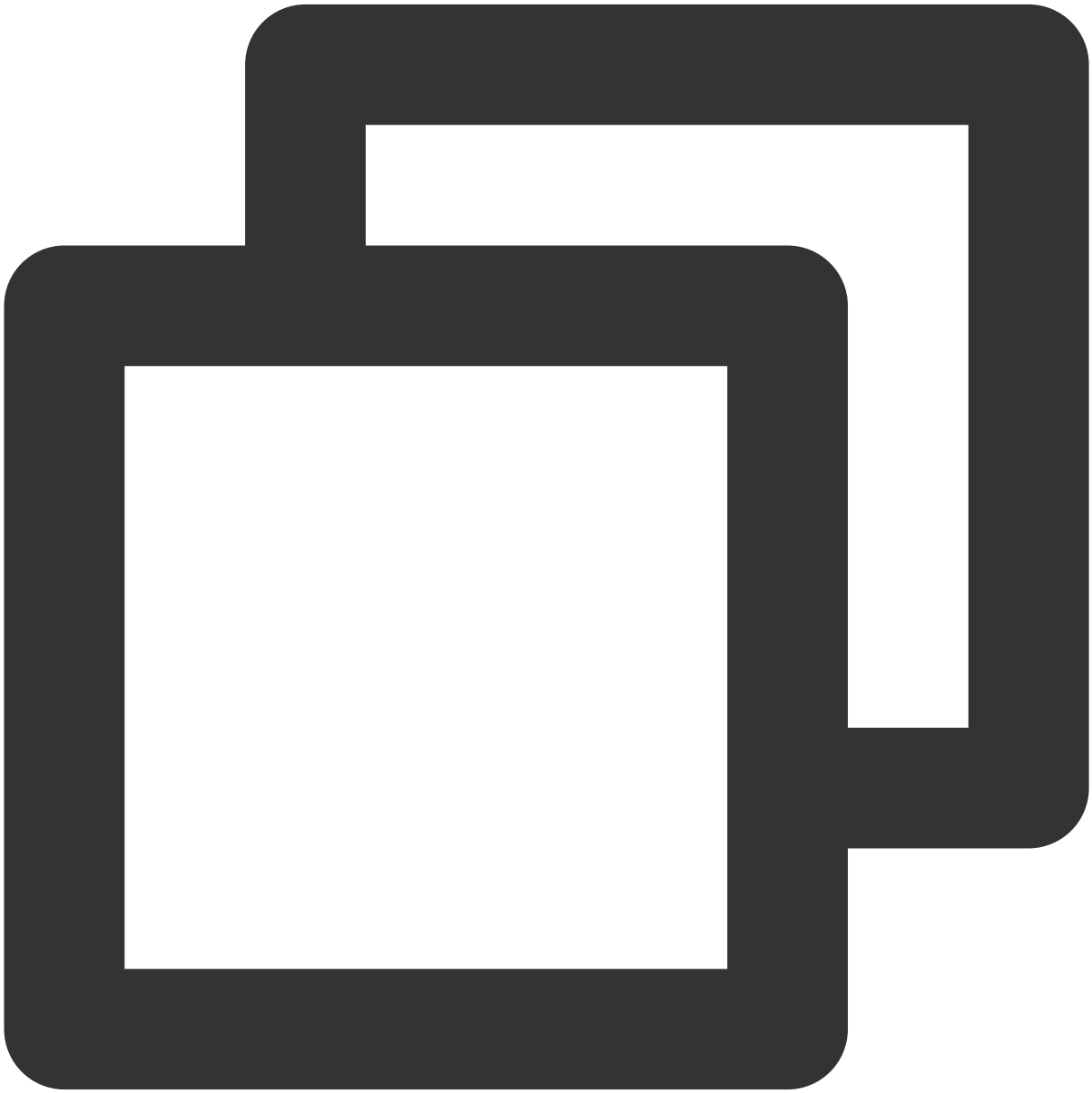
```
roomEngine.on(TUIRoomEvents.onKickedOffSeat, ({ userId }) => {
  console.log('roomEngine.onKickedOffSeat', userId);
});
```

参数如下表所示：

参数	类型	含义
userId	String	被踢下麦的用户 ID

onRequestReceived

错误事件



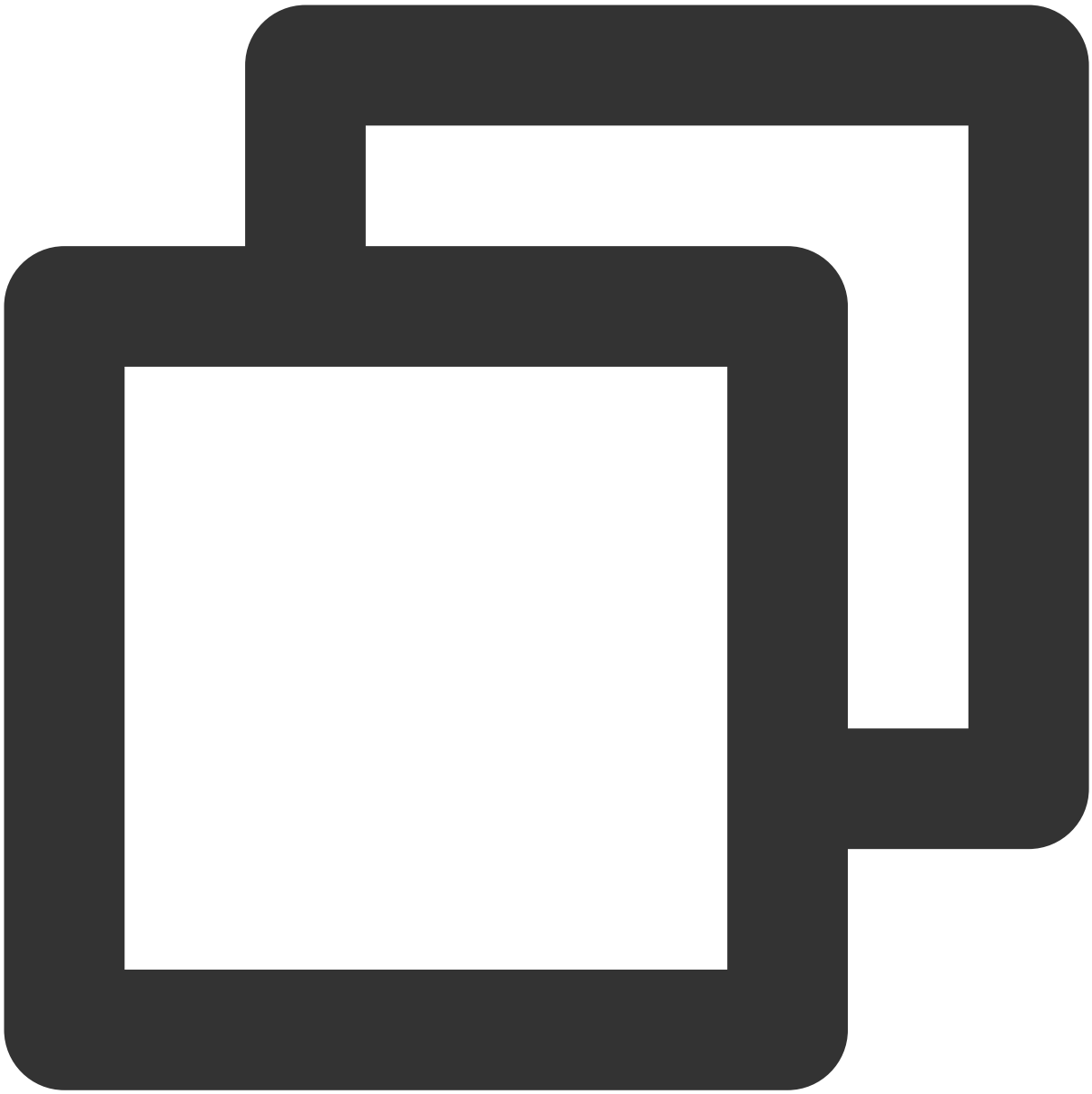
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestReceived, ({ request }) => {
  console.log('roomEngine.onRequestReceived', request);
});
```

参数如下表所示：

参数	类型	含义
request	TUIRequest	请求接收

onRequestCancelled

请求取消事件



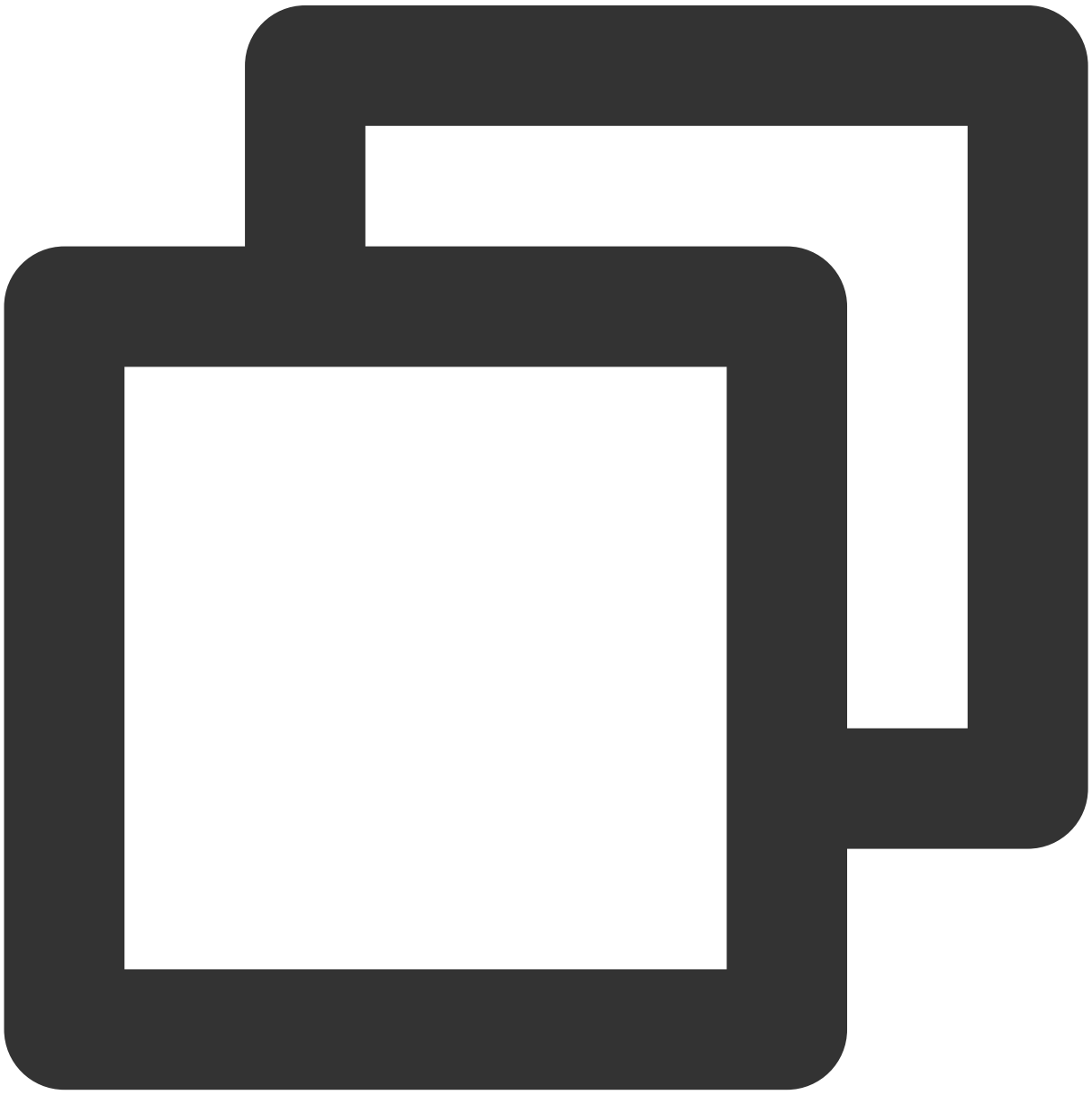
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onRequestCancelled, ({ requestId, userId }) => {
  console.log('roomEngine.onRequestCancelled', requestId, userId);
});
```

参数如下表所示：

参数	类型	含义
requestId	string	请求 ID
userId	string	取消请求的用户 ID

onReceiveTextMessage

接收文本消息事件



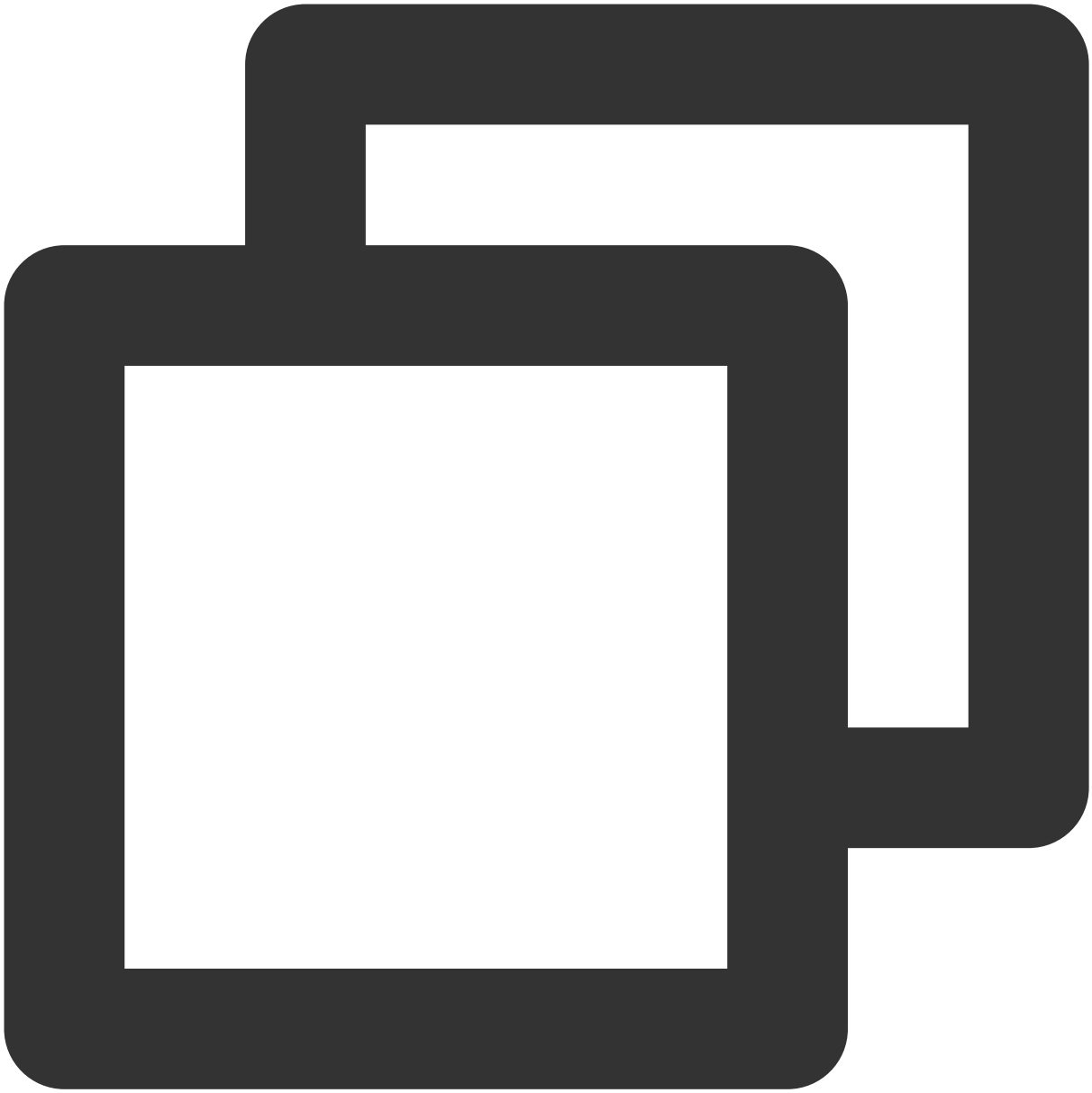
```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveTextMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveTextMessage', roomId, message);
});
```

参数如下表所示：

参数	类型	含义
roomId	string	房间 ID
message	TUIMessage	接收文本消息

onReceiveCustomMessage

接收自定义消息事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onReceiveCustomMessage, ({ roomId, message }) => {
  console.log('roomEngine.onReceiveCustomMessage', roomId, message);
});
```

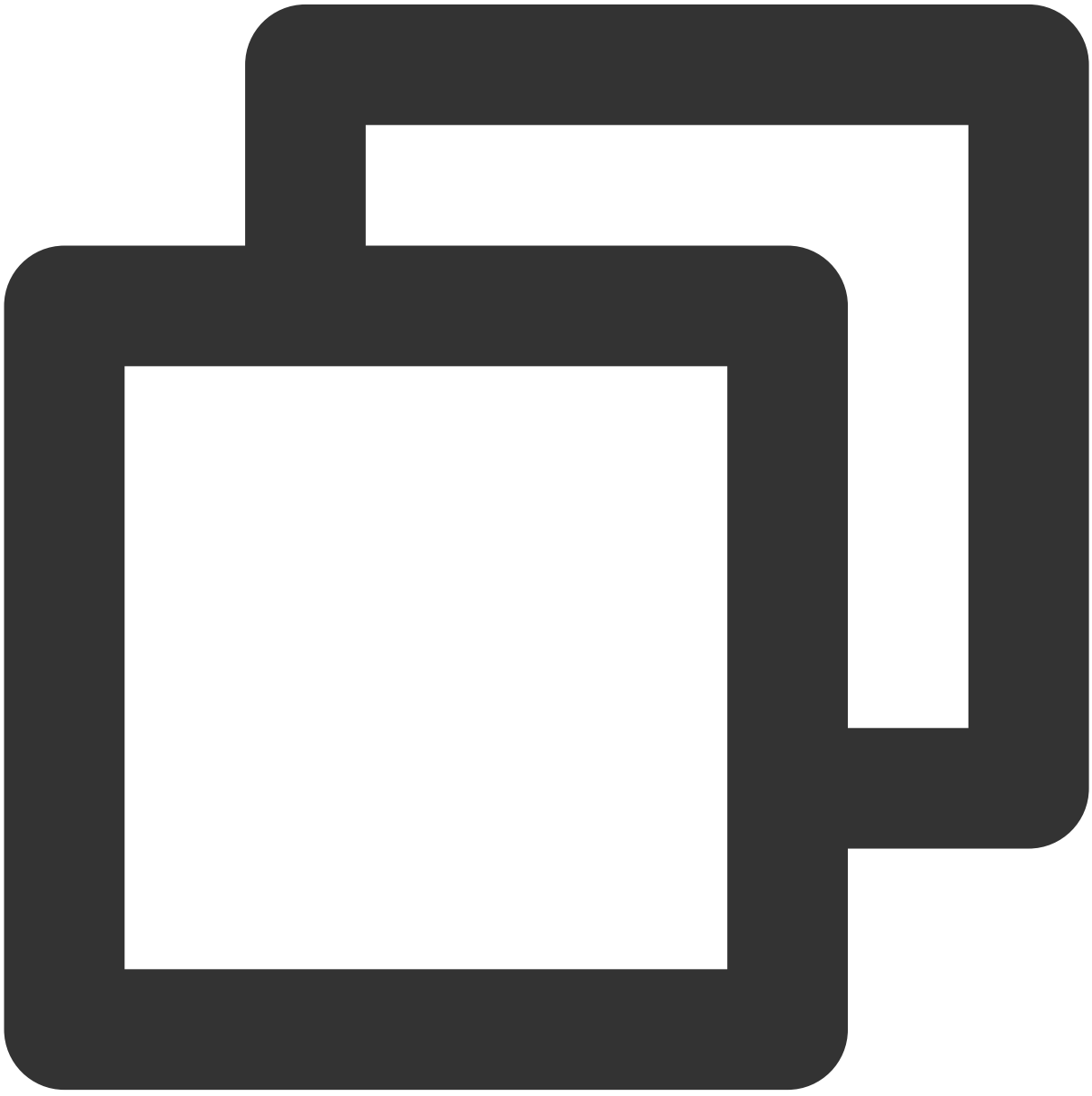
参数如下表所示：

参数	类型	含义
roomId	string	房间 ID

message	TUIMessage	接收文本消息

onDeviceChange

设备变更事件



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onDeviceChange, ({ deviceId, type, state }) => {
  console.log('roomEngine.onReceiveCustomMessage', deviceId, type, state);
```

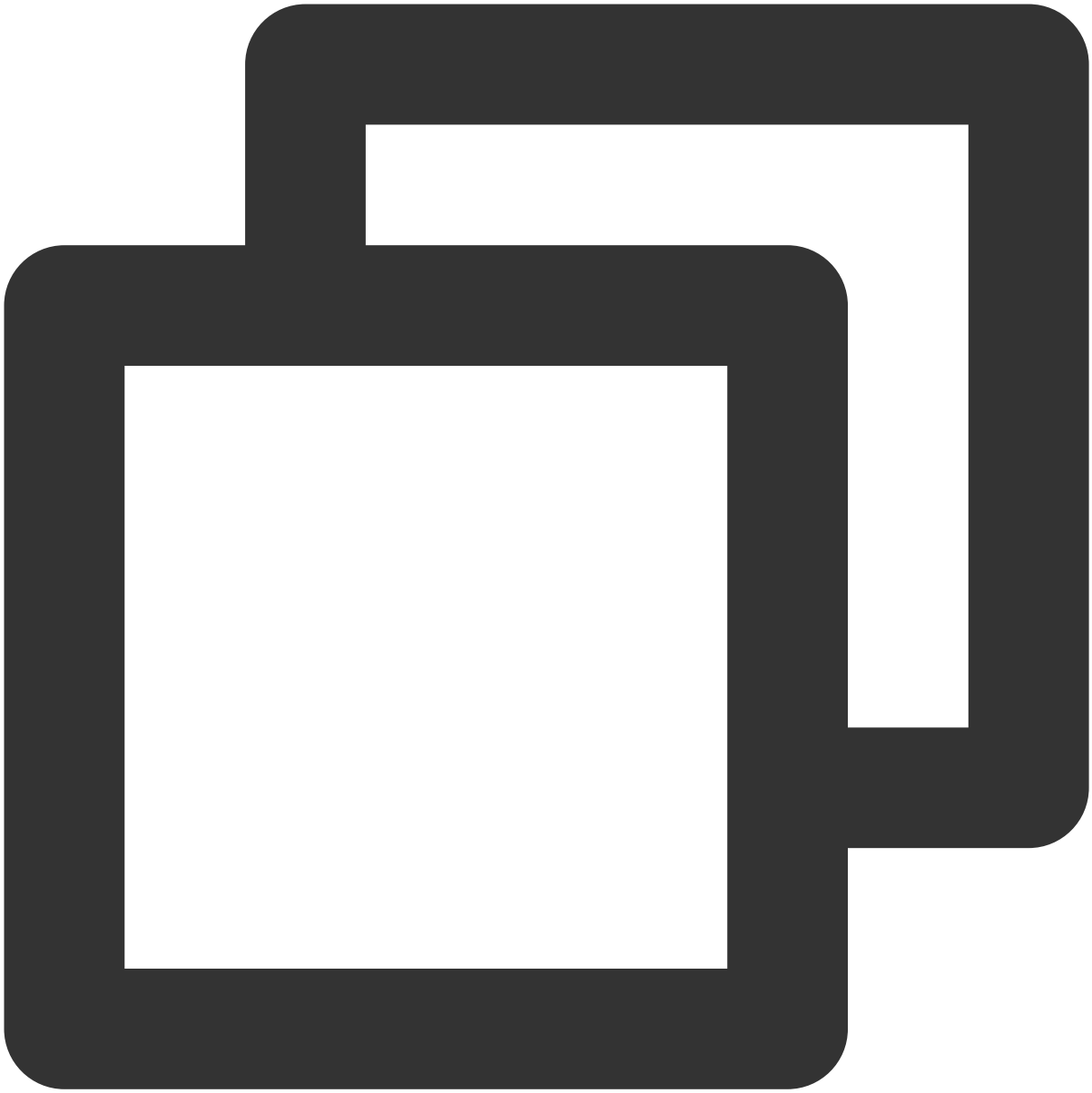
```
});
```

参数如下表所示：

参数	类型	含义
deviceId	string	设备 ID
type	TRTCDeviceType	设备类型
state	TRTCDeviceState	设备变更状态

onUserScreenCaptureStopped

屏幕分享停止事件，当用户使用浏览器自带的 **停止分享** 按钮结束屏幕共享时，该用户会收到 `onUserScreenCaptureStopped` 事件用来修改屏幕共享状态。



```
const roomEngine = new TUIRoomEngine();
roomEngine.on(TUIRoomEvents.onUserScreenCaptureStopped, () => {
  console.log('roomEngine.onReceiveCustomMessage', deviceId, type, state);
});
```

TUIRoomEngine Defines

最近更新时间：2023-11-16 14:31:05

本文介绍 TUIRoomEnigne Electron 端关键类型定义。

枚举值

TUIRole

用户角色，TUIRoomEngine 共提供三种用户角色，分别是主持人，管理员，普通用户。

字段	类型	描述
kRoomOwner	number	主持人角色
kAdministrator	number	管理员角色
kGeneralUser	number	普通用户角色

TUIVideoQuality

视频分辨率

字段	类型	描述
kVideoQuality_360p	number	低清，分辨率为 640 * 360
kVideoQuality_540p	number	标清，分辨率为 960 * 540
kVideoQuality_720p	number	高清，分辨率为 1280 * 720
kVideoQuality_1080p	number	超清，分辨率为 1920 * 1080

TUIAudioProfile

音频分辨率

字段	类型	描述
kAudioProfileSpeech	number	语音模式
kAudioProfileDefault	number	标准模式（默认模式）
kAudioProfileMusic	number	音乐模式

版权所有：腾讯云计算（北京）有限责任公司

TUIVideoStreamType

视频流类型

字段	类型	描述
kCameraStream	number	摄像头视频流
kScreenStream	number	屏幕分享视频流
kCameraStreamLow	number	低清摄像头视频流

TUINetworkQuality

网络状态

字段	类型	描述
kQualityUnknown	number	网络状况未知
kQualityExcellent	number	网络状况极佳
kQualityGood	number	网络状况较好
kQualityPoor	number	网络状况一般
kQualityBad	number	网络状况差
kQualityVeryBad	number	网络状况极差
kQualityDown	number	网络连接已断开

TUIRoomType

房间类型

字段	类型	描述
kGroup	number	Group 类型房间，适用于会议，教育场景，该房间中麦位是无序的，且没有数量限制
kOpen	number	Open类型房间，适用于直播场景，该房间中麦位是有序的，且有数量限制

TUISpeechMode

发言类型

字段	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

kFreeToSpeak	number	自由发言模式
kApplyToSpeak	number	举手发言模式
kSpeakAfterTakingSeat	number	就坐后发言（抢麦位）

TUICaptureSourceType

屏幕分享类型

字段	类型	描述
kWindow	number	该分享目标是某一个 Windows 或 Mac 窗口 todo(只有electron)
kScreen	number	该分享目标是整个 Windows 桌面或 Mac 桌面

TUIChangeReason

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

字段	类型	描述
kChangedBySelf	number	自己操作
kChangedByAdmin	number	房主或管理员操作

TUIMediaDevice

字段	类型	描述
kMicrophone	number	麦克风
kCamera	number	摄像头
kScreen	number	屏幕分享

TUIRequestAction

房间类型

字段	类型	描述
kInvalidAction	number	无效操作
kRequestToOpenRemoteCamera	number	请求远端打开摄像头
kRequestToOpenRemoteMicrophone	number	请求远端打开麦克风

版权所有：腾讯云计算（北京）有限责任公司

kRequestToConnectOtherRoom	number	请求远端夸房连麦，web 端暂不支持
kRequestToTakeSeat	number	请求上麦
kRequestRemoteUserOnSeat	number	请求远端上麦

TUIRequestCallbackType

请求类型

字段	类型	描述
kRequestAccepted	number	对端已接受
kRequestRejected	number	对端已拒绝
kRequestCancelled	number	请求已取消
kRequestTimeout	number	请求超时
kRequestError	number	请求错误

类型定义

TUILoginUserInfo

当前登录用户信息

名称	类型	描述
userId	string	登录用户的 ID
userName	string	登录用户的名称
avatarUrl	string	登录用户的头像

TUIRoomInfo

房间信息，用户可使用 roomEngine.getRoomInfo 获取房间信息。

名称	类型	描述
roomId	string	房间号，字符串类型房间号
roomType	TUIRoomType	房间类型
owner	string	房间主持人的 userId

版权所有：腾讯云计算（北京）有限责任公司

name	string	房间名称
createTime	string	房间创建时间
roomMemberCount	number	当前房间总人数
maxSeatCount	number	房间最大麦位数量
enableVideo	boolean	是否允许加入用户打开音频
enableAudio	boolean	是否允许加入用户打开视频
enableMessage	boolean	是否允许加入用户发送消息
enableSeatControl	boolean	是否开启麦位控制

TUIUserInfo

用户信息

名称	类型	描述
userId	string	用户 ID
userName	string	用户名称
avatarUrl	string	用户头像地址
userRole	TUIRole	用户角色
hasAudioStream	boolean	是否有音频流
hasVideoStream	boolean	是否有视频流
hasScreenStream	boolean	是否有屏幕分享流

TUIMessage

消息信息

名称	类型	描述
messageId	string	消息 ID
message	string	消息
timestamp	number	时间戳信息, 精确到秒
userId	string	用户 ID

userName	string	用户名称
avatarUrl	string	用户头像地址

TUIRequest

请求信息

名称	类型	描述
requestAction	TUIRequestAction	请求类型
timestamp	number	请求发起时间
requestId	string	请求 ID v1.0.2 及其以上版本 requestId 类型为 string； v1.0.0 及 v1.0.1 版本 requestId 类型为 number；
userId	string	发起请求的用户 ID
content	string	其他内容

TUIRequestCallback

请求回调信息

名称	类型	描述
requestCallbackType	TUIRequestCallbackType	请求回调类型，接受/拒绝/取消/超时/错误
requestId	string	请求 ID v1.0.2 及其以上版本 requestId 类型为 string； v1.0.0 及 v1.0.1 版本 requestId 类型为 number；
userId	string	用户 ID
code	number	请求响应码
message	string	请求状态补充说明

TUISeatInfo

麦位信息

名称	类型	描述

版权所有：腾讯云计算（北京）有限责任公司

index	number	麦位序号
userId	string	麦位对应的用户 ID
locked	boolean	当前麦位是否被锁
videoMuted	boolean	当前麦位是否禁止视频
audioMuted	boolean	当前麦位是否禁止音频

TUISeatLockParams

麦位锁定状态

名称	类型	描述
lockSeat	boolean	锁定麦位
lockVideo	boolean	锁定麦位视频
lockAudio	boolean	锁定麦位音频

TUINetwork

网络信息

名称	类型	描述
userId	string	用户 ID
quality	TUINetworkQuality	网络质量
upLoss	number	上行丢包率，单位 (%) 该数值越小越好，目前仅本地用户有该信息
downLoss	number	下行丢包率，单位 (%) 该数值越小越好，目前仅本地用户有该信息
delay	number	网络延迟，单位 ms，目前仅本地用户有该信息

版权所有：腾讯云计算（北京）有限责任公司

Flutter
API概览

最近更新时间：2023-11-21 15:20:30

TUIRoomEngine API 列表

TUIRoomEngine API 是多人音视频房间的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。
TUIRoomEngine

TUIRoomEngine 核心方法

API	描述
createInstance	创建 TUIRoomEngine 实例。
destroyInstance	销毁 TUIRoomEngine 实例。
login	登录接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。
logout	退出登录接口，会有主动离开房间操作、销毁资源。
setSelfInfo	设置本地用户名称和头像。
setLoginUserInfo	设置登录用户信息。
getSelfInfo	获取本地用户登录的基本信息。
addObserver	设置事件回调。
removeObserver	移除事件回调。

房间相关主动接口

API	描述
createRoom	创建房间。
destroyRoom	解散房间。
enterRoom	进入房间。
exitRoom	离开房间。

版权所有：腾讯云计算（北京）有限责任公司

connectOtherRoom	连接其他房间。
disconnectOtherRoom	断开与其他房间的连接。
fetchRoomInfo	获取房间信息。
updateRoomNameByAdmin	更新房间名称（只有管理员或群主能够调用）。
updateRoomSpeechModeByAdmin	设置房间管理模式（只有管理员或群主能够调用）。

本地用户视图渲染、视频管理

API	描述
setLocalVideoView	设置本地用户视频渲染的控件。
openLocalCamera	打开本地摄像头。
closeLocalCamera	关闭本地摄像头。
updateVideoQuality	更新本地视频编码质量设置。
updateVideoQualityEx	设置视频编码器的编码参数。
setVideoResolutionMode	设置视频编码器分辨率模式。
enableGravitySensor	开启重力感应模式。
startPushLocalVideo	开始推送本地视频。
stopPushLocalVideo	停止推送本地视频。
startScreenSharing	开始屏幕分享。
stopScreenSharing	结束屏幕分享。

本地用户音频管理

API	描述
openLocalMicrophone	打开本地麦克风。
closeLocalMicrophone	关闭本地麦克风。
updateAudioQuality	更新本地音频编码质量设置。
muteLocalAudio	停止推送本地音频。

版权所有：腾讯云计算（北京）有限责任公司

unMuteLocalAudio	开始推送本地音频。

远端用户视图渲染、视频管理

API	描述
setRemoteVideoView	设置远端用户视频渲染的控件。
startPlayRemoteVideo	开始播放远端用户视频。
stopPlayRemoteVideo	停止播放远端用户视频。
muteRemoteAudioStream	将远端用户禁音。

房间内用户信息

API	描述
getUserList	获取房间内的成员列表。
getUserInfo	获取成员信息。

房间内用户管理

API	描述
changeUserRole	修改用户角色（只有管理员或群主能够调用）。
kickRemoteUserOutOfRoom	将远端用户移出房间（只有管理员或群主能够调用）。
addCategoryTagForUsers	为用户增加标记（只有房主能够调用）。
removeCategoryTagForUsers	为用户移除标记（只有房主能够调用）。
getUserListByTag	根据标记获取房间内用户信息。

房间内用户发言管理

API	描述
disableDeviceForAllUserByAdmin	控制当前房间内所有用户是否可打开音频流、视频流采集设备的权限状态，例如：全员禁止打开麦克风、全员禁止打开摄像头、全员禁止打开屏幕分享（目前仅会议场景下可用，并且只有管理员或群主能够调用）。
openRemoteDeviceByAdmin	请求远端用户打开媒体设备（只有管理员或群主能够调用）。

版权所有：腾讯云计算（北京）有限责任公司

closeRemoteDeviceByAdmin	关闭远端用户媒体设备（只有管理员或群主能够调用）。
applyToAdminToOpenLocalDevice	请求打开本地媒体设备（普通用户可用）。

房间内麦位管理

API	描述
setMaxSeatCount	设置最大麦位数（仅支持进房前和创建房间时设置）。
getSeatList	获取麦位列表。
lockSeatByAdmin	锁定麦位（只有管理员或群主能够调用，包括位置锁定、音频状态锁定和视频状态锁定）。
takeSeat	申请上麦（自由发言模式下，不需要申请）。
leaveSeat	申请下麦（自由发言模式下，不需要申请）。
takeUserOnSeatByAdmin	主持人/管理员 邀请用户上麦。
kickUserOffSeatByAdmin	主持人/管理员 将用户移下麦。

信令管理

API	描述
cancelRequest	取消请求。
responseRemoteRequest	回复请求。

发送消息

API	描述
sendTextMessage	发送本文消息。
sendCustomMessage	发送自定义消息。
disableSendingMessageByAdmin	禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）。
disableSendingMessageForAllUser	禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）。

版权所有：腾讯云计算（北京）有限责任公司

高级功能

API	描述
switchCamera	切换前置或后置摄像头
setBeautyLevel	设置美颜级别
setWhitenessLevel	设置美白级别

调试相关

API	描述
callExperimentalAPI	调用实验性接口

TUIRoomObserver 回调事件

TUIRoomObserver 是 TUIRoomEngine 对应的回调事件类，您可以通过此回调，来监听自己需要的回调事件。
TUIRoomObserver

TUIRoomObserver

错误回调

API	描述
onError	错误事件回调。

登录状态事件回调

API	描述
onKickedOffLine	其他终端登录被移下线。
onUserSigExpired	用户凭证超时事件。

房间内事件回调

API	描述
onRoomNameChanged	房间名称更改事件。
onAllUserMicrophoneDisableChanged	房间内所有用户麦克风被禁用事件。
onAllUserCameraDisableChanged	房间内所有用户摄像头被禁用事件。
onSendMessageForAllUserDisableChanged	房间内所有用户发送文本消息被禁用事件。
onKickedOutOfRoom	被移出房间事件。
onRoomDismissed	房间被解散事件。
onRoomSpeechModeChanged	房间麦控模式发生变化

房间内用户事件回调

API	描述
onRemoteUserEnterRoom	远端用户进房事件。
onRemoteUserLeaveRoom	远端用户离开房间事件。
onUserRoleChanged	用户角色发生变化事件。
onUserVideoStateChanged	用户视频状态发生变化事件。
onUserAudioStateChanged	用户音频状态发生变化事件。
onUserVoiceVolumeChanged	用户音量变化事件。
onSendMessageForUserDisableChanged	用户文本消息发送能力发生变化事件。
onUserNetworkQualityChanged	用户网络状态变化事件。
onUserScreenCaptureStopped	屏幕分享结束。

房间内麦位事件回调

API	描述
onRoomMaxSeatCountChanged	房间内最大麦位数发生变化事件（仅在会议类型房间生效）。
onSeatListChanged	麦位列表发生变化事件。
onKickedOffSeat	收到用户被移下麦事件。

版权所有：腾讯云计算（北京）有限责任公司

请求信令事件回调

API	描述
onRequestReceived	收到请求消息事件。
onRequestCancelled	收到请求被取消的事件。

房间内消息事件回调

API	描述
onReceiveTextMessage	收到普通文本消息事件。
onReceiveCustomMessage	收到自定义消息事件。

版权所有：腾讯云计算（北京）有限责任公司

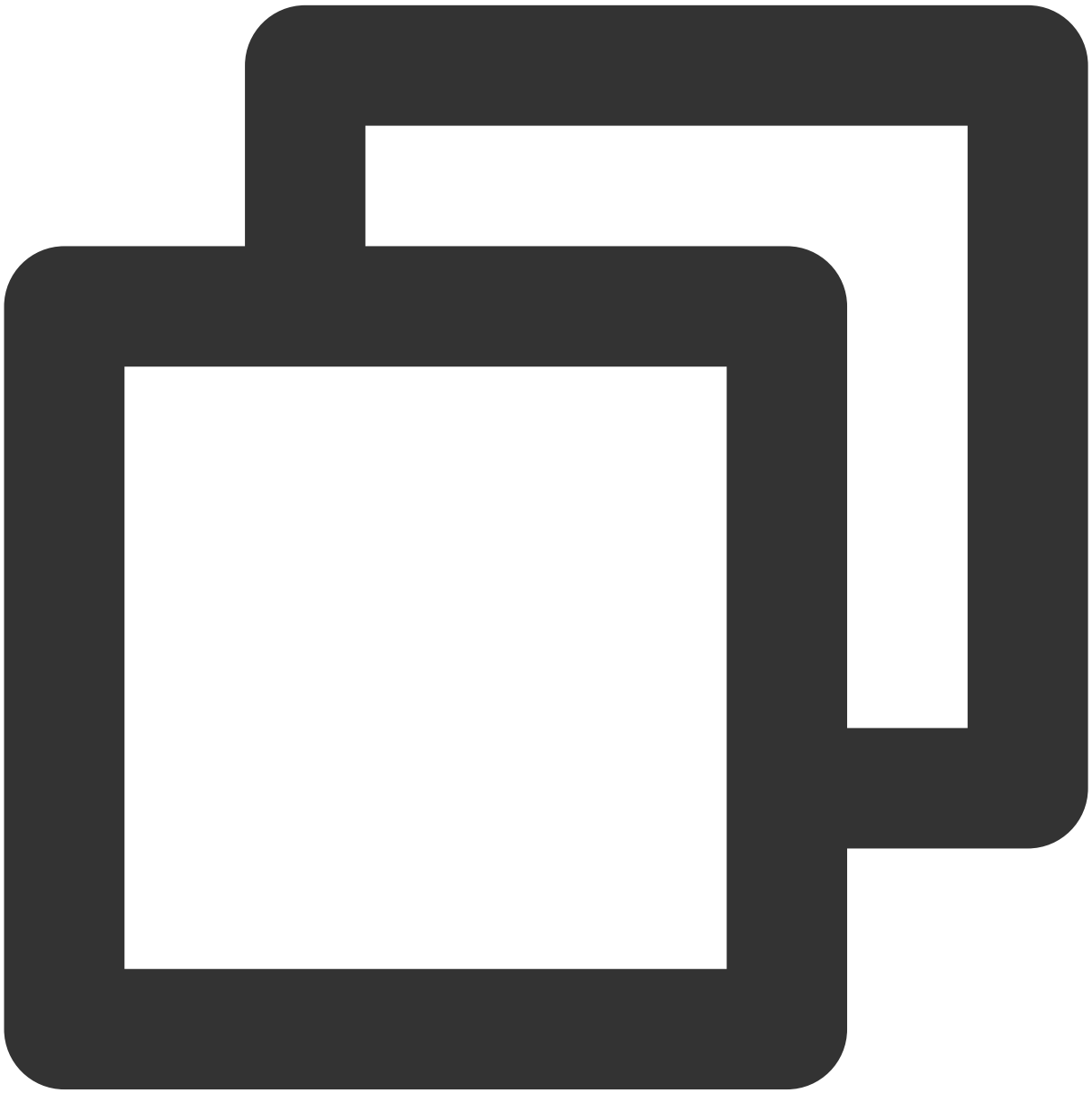
TUIRoomObserver

最近更新时间：2023-11-14 16:34:28

TUIRoomEngine 事件回调

onError

错误事件。

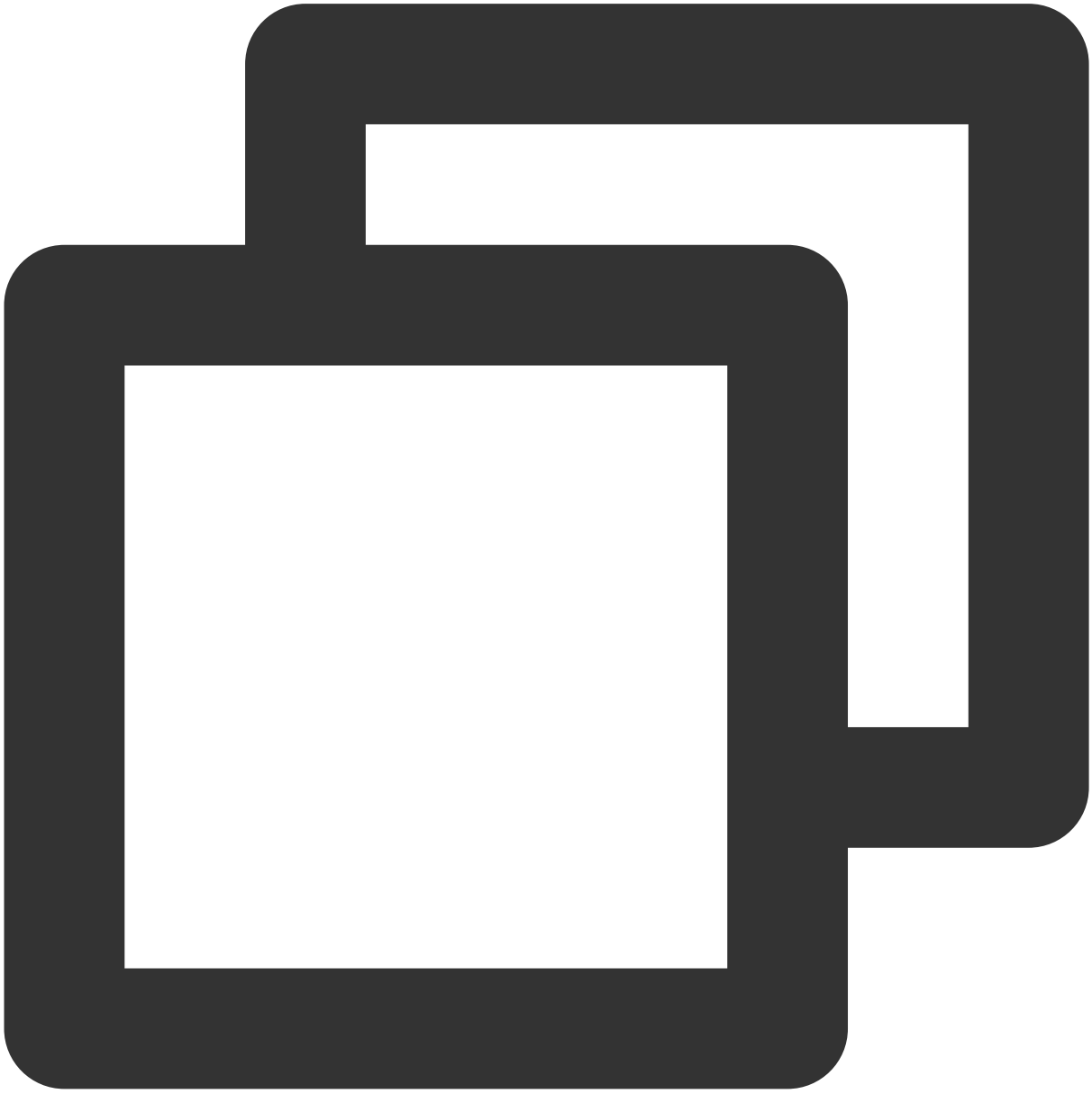


```
OnError onError = (TUIError errorCode, String message) {}
```

参数	类型	描述
errorCode	TUIError	错误码
message	String	错误信息

onKickedOffLine

用户被移下线事件。

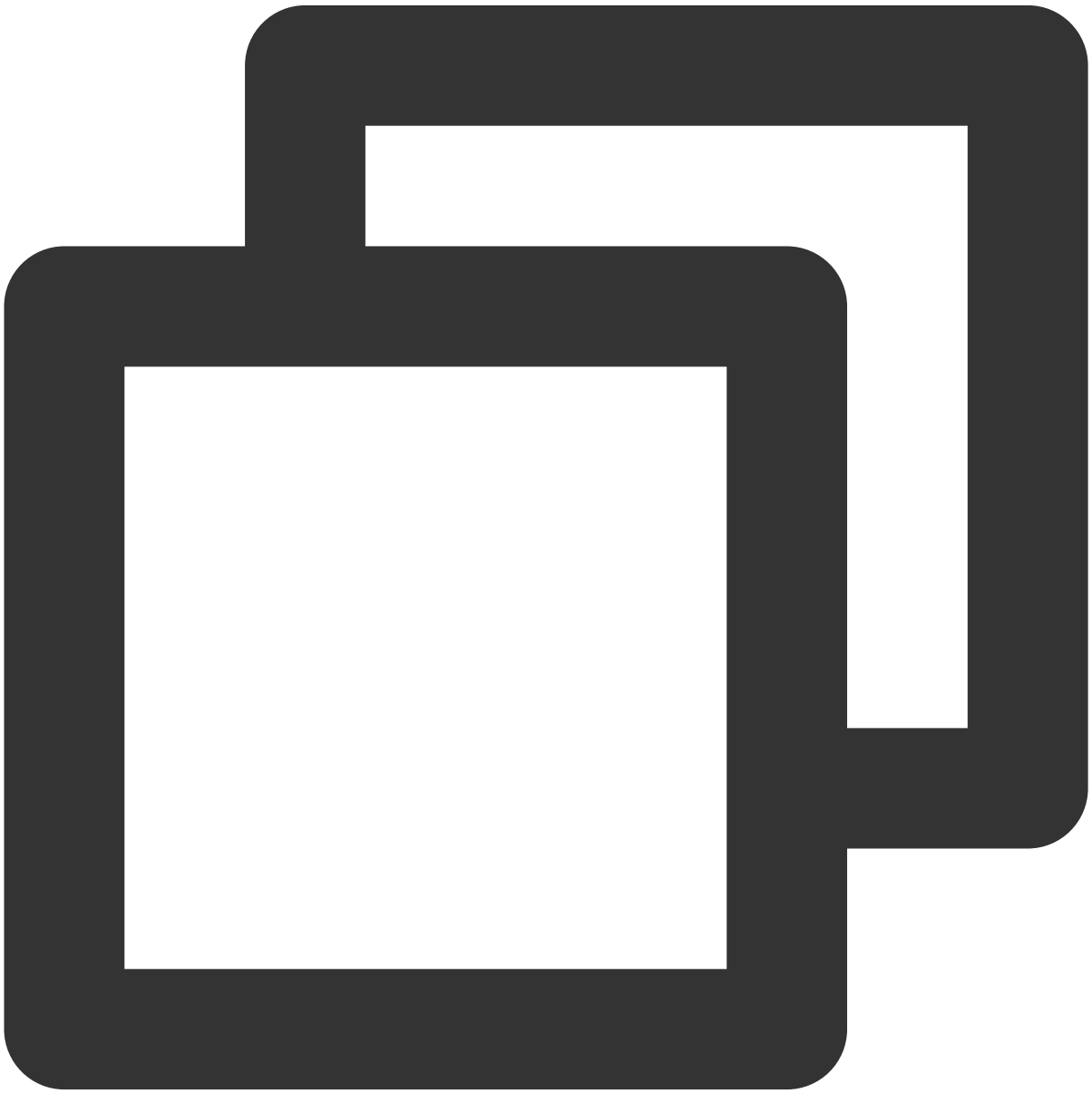


```
OnKickedOffLine onKickedOffLine = (String message) {}
```

参数	类型	描述
message	String	被移下线的描述

onUserSigExpired

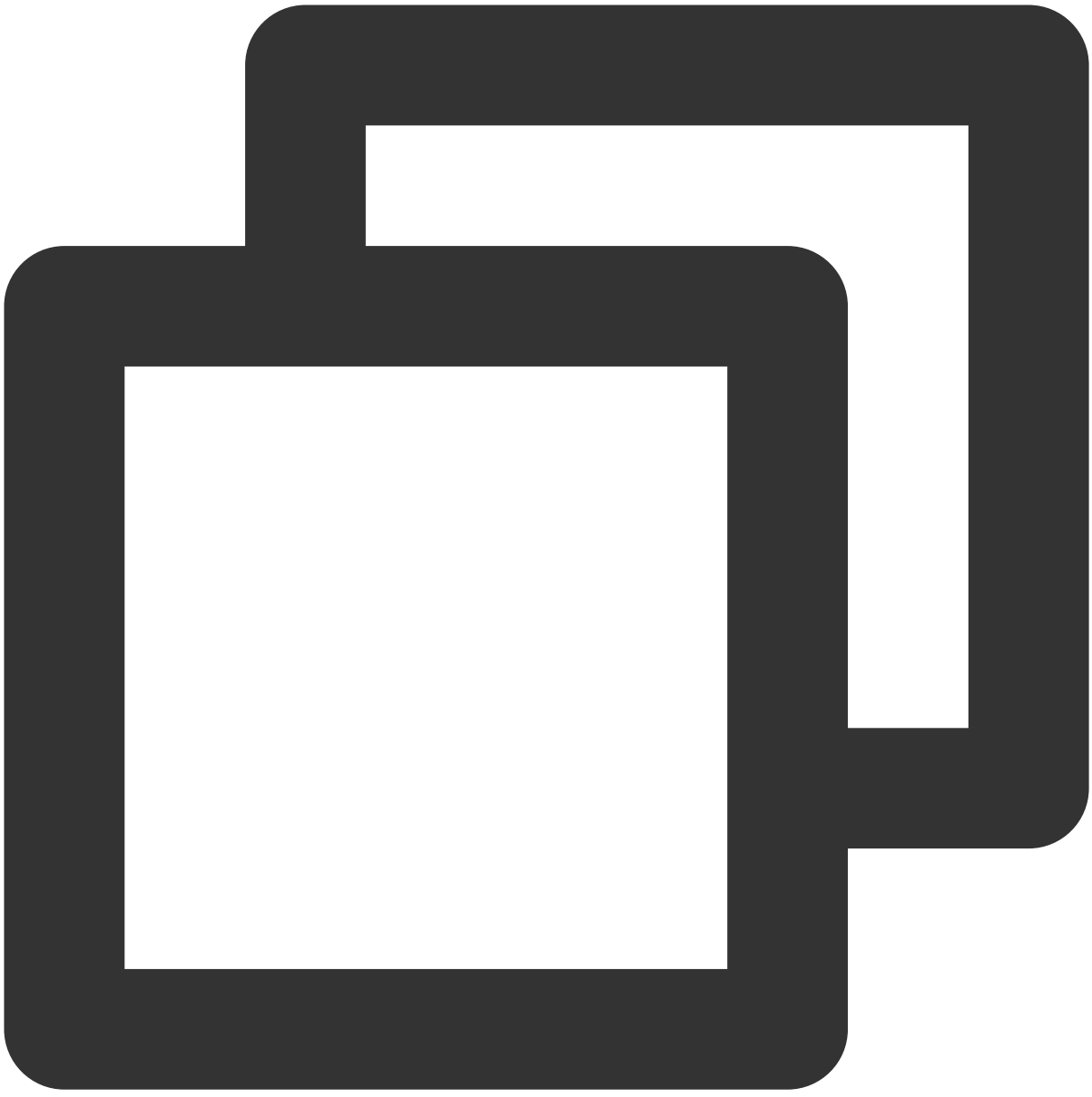
`userSig` 过期事件。



```
OnUserSigExpired onUserSigExpired = () {}
```

onRoomNameChanged

房间名称更改事件。



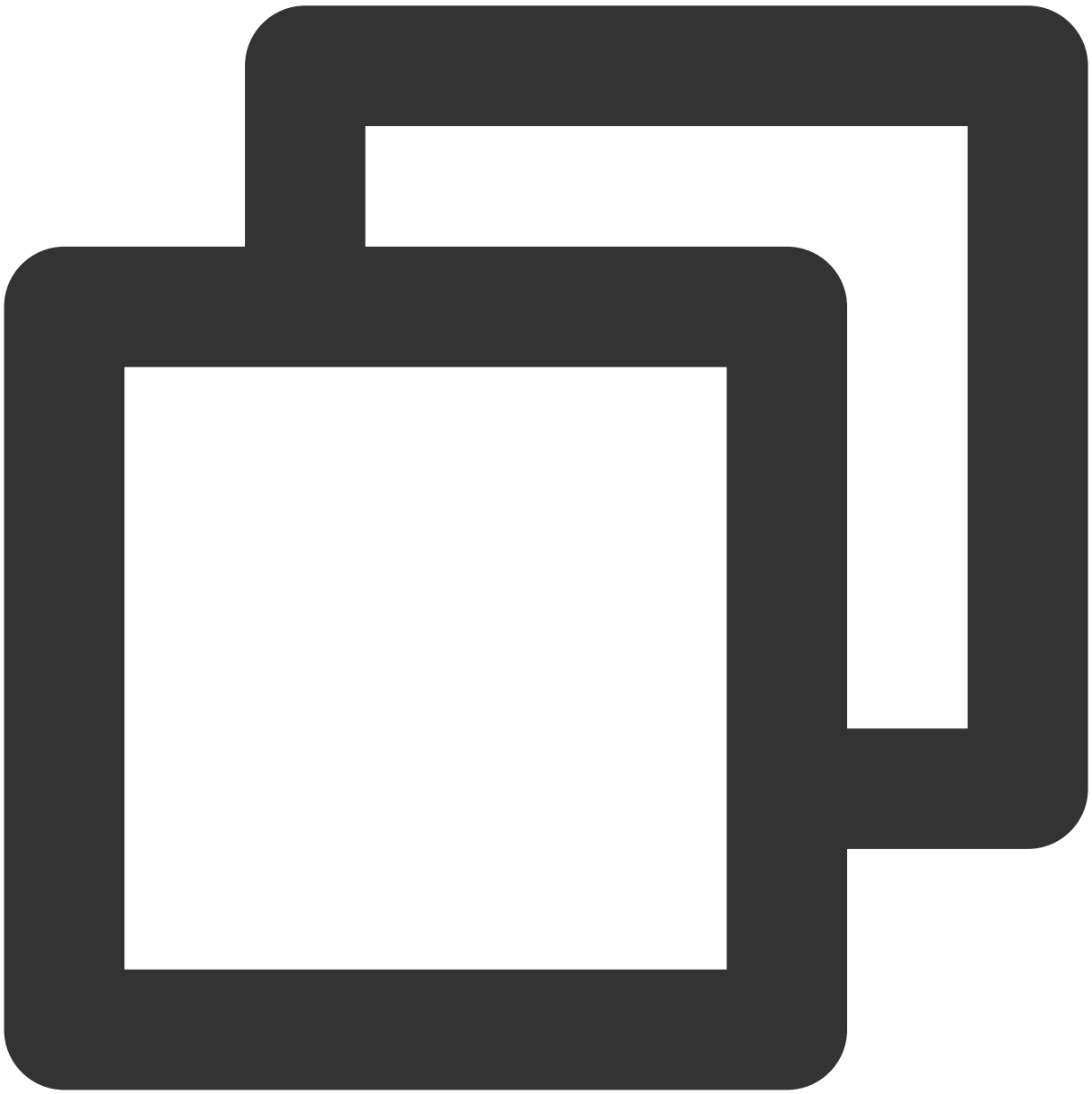
```
OnRoomNameChanged onRoomNameChanged = (String roomId, String roomName) {}
```

参数	类型	描述

roomId	String	房间 ID
roomName	String	房间名称

onAllUserMicrophoneDisableChanged

房间内所有用户麦克风被禁用事件。

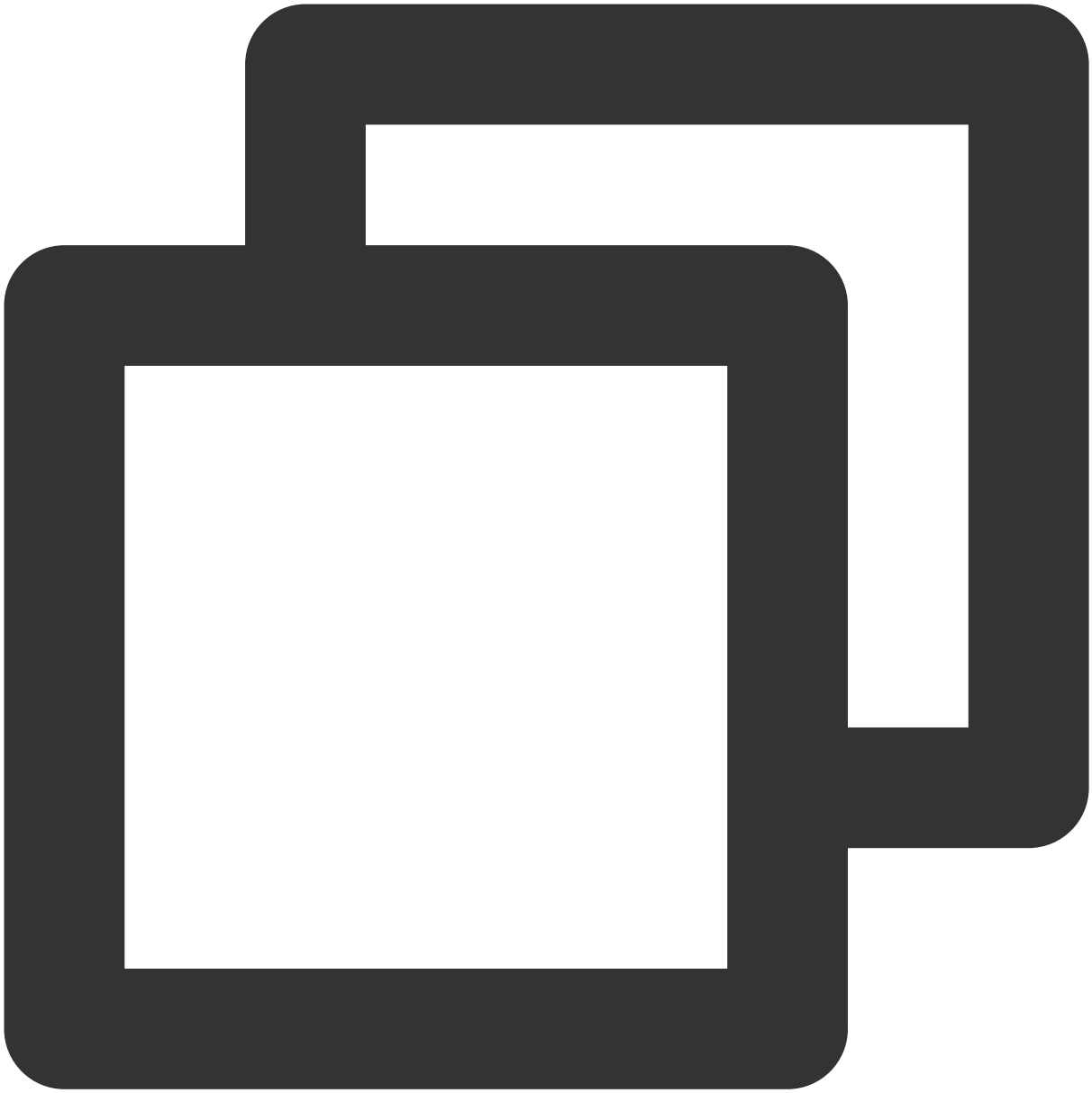


```
OnAllUserMicrophoneDisableChanged onAllUserMicrophoneDisableChanged = (String roomI
```

参数	类型	描述
roomId	String	房间 ID
isDisable	bool	是否被禁用

onAllUserCameraDisableChanged

房间内所有用户摄像头被禁用事件。

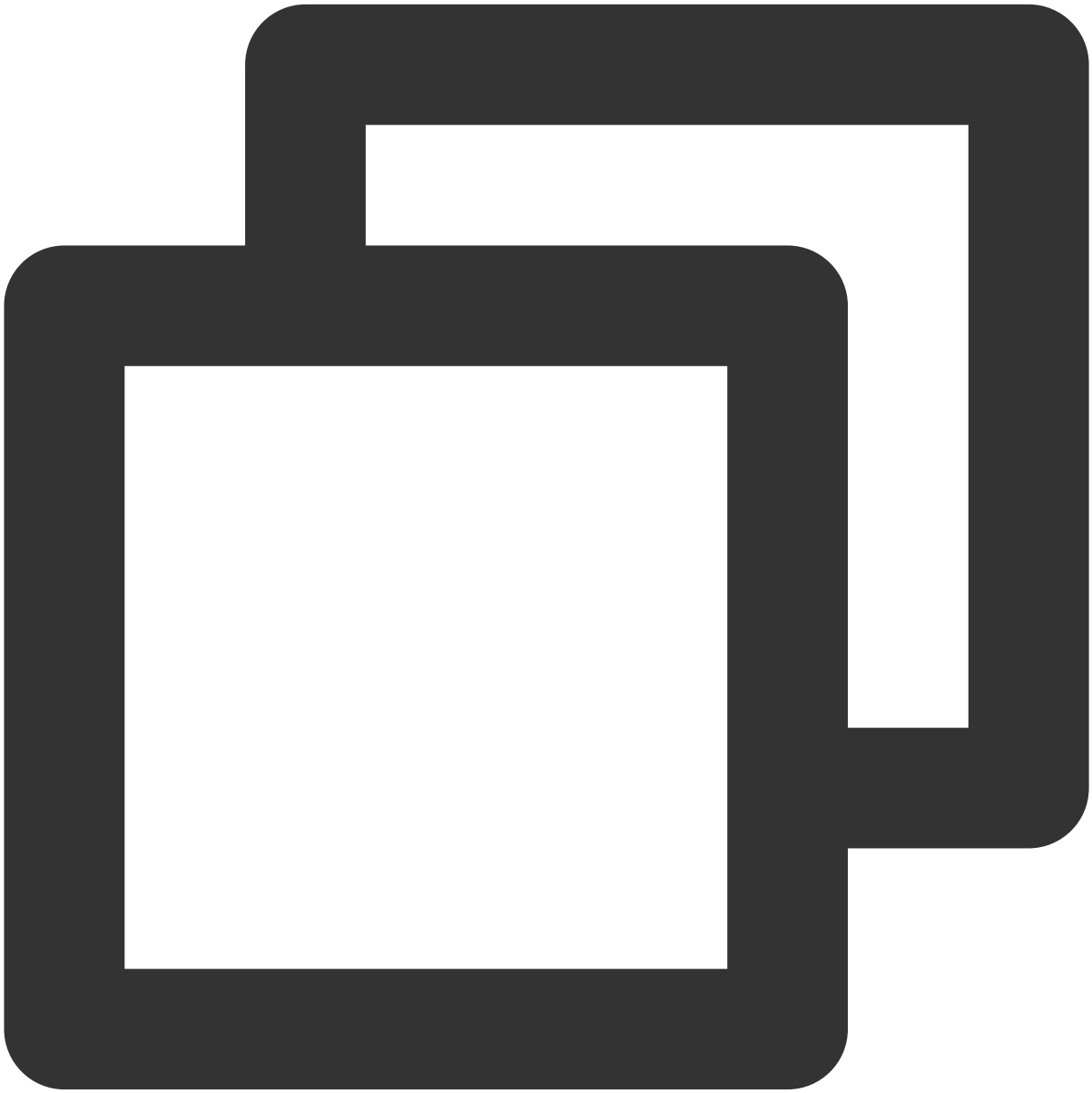


```
OnAllUserCameraDisableChanged onAllUserCameraDisableChanged = (String roomId, bool
```

参数	类型	描述
roomId	String	房间 ID
isDisable	bool	是否被禁用

onSendMessageForAllUserDisableChanged

房间内所有用户发送文本消息被禁用事件。

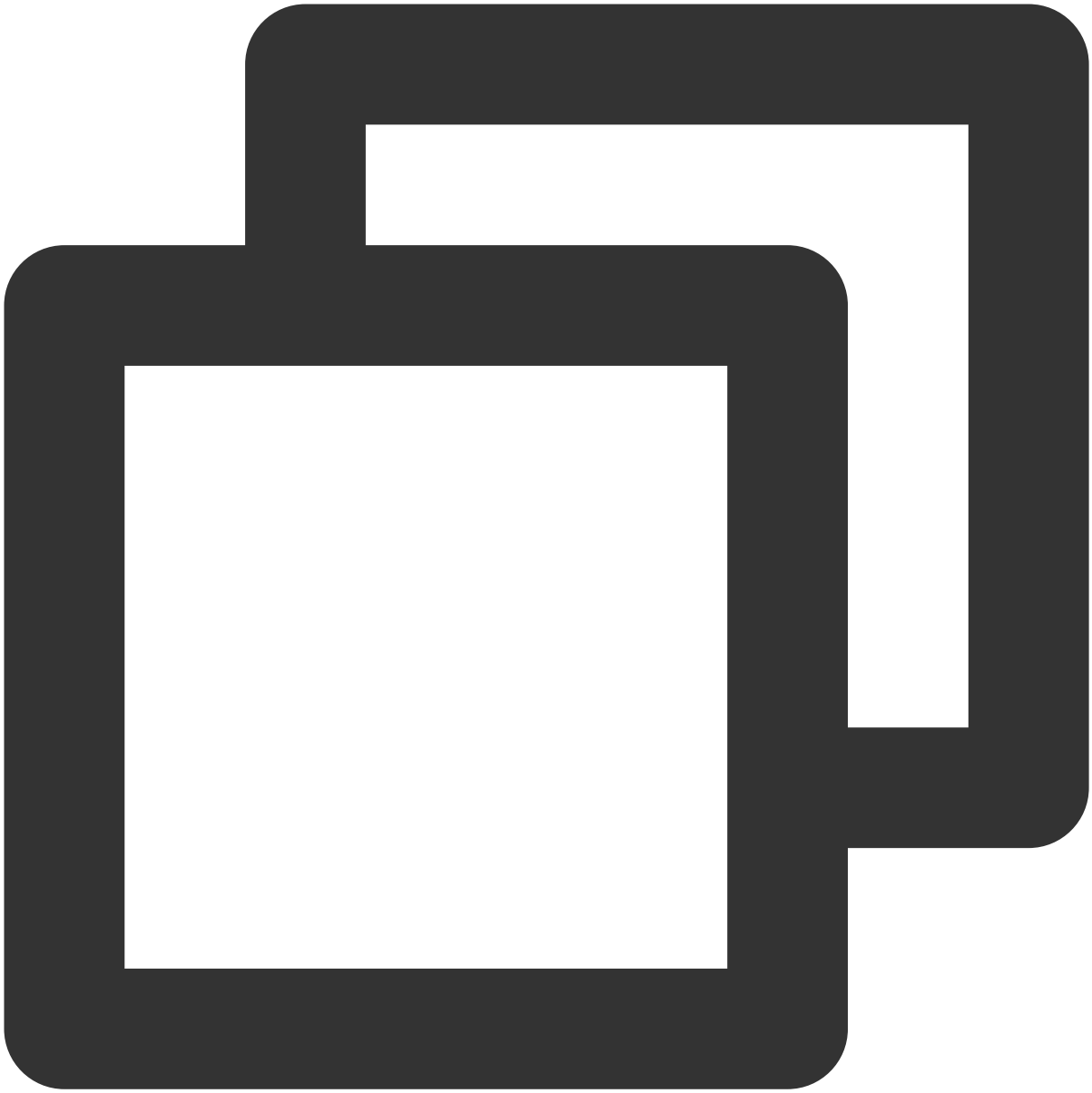


```
OnSendMessageForAllUserDisableChanged onSendMessageForAllUserDisableChanged = (Stri
```

参数	类型	描述
roomId	String	房间 ID
isDisable	bool	是否被禁用

onRoomDismissed

房间被解散事件。

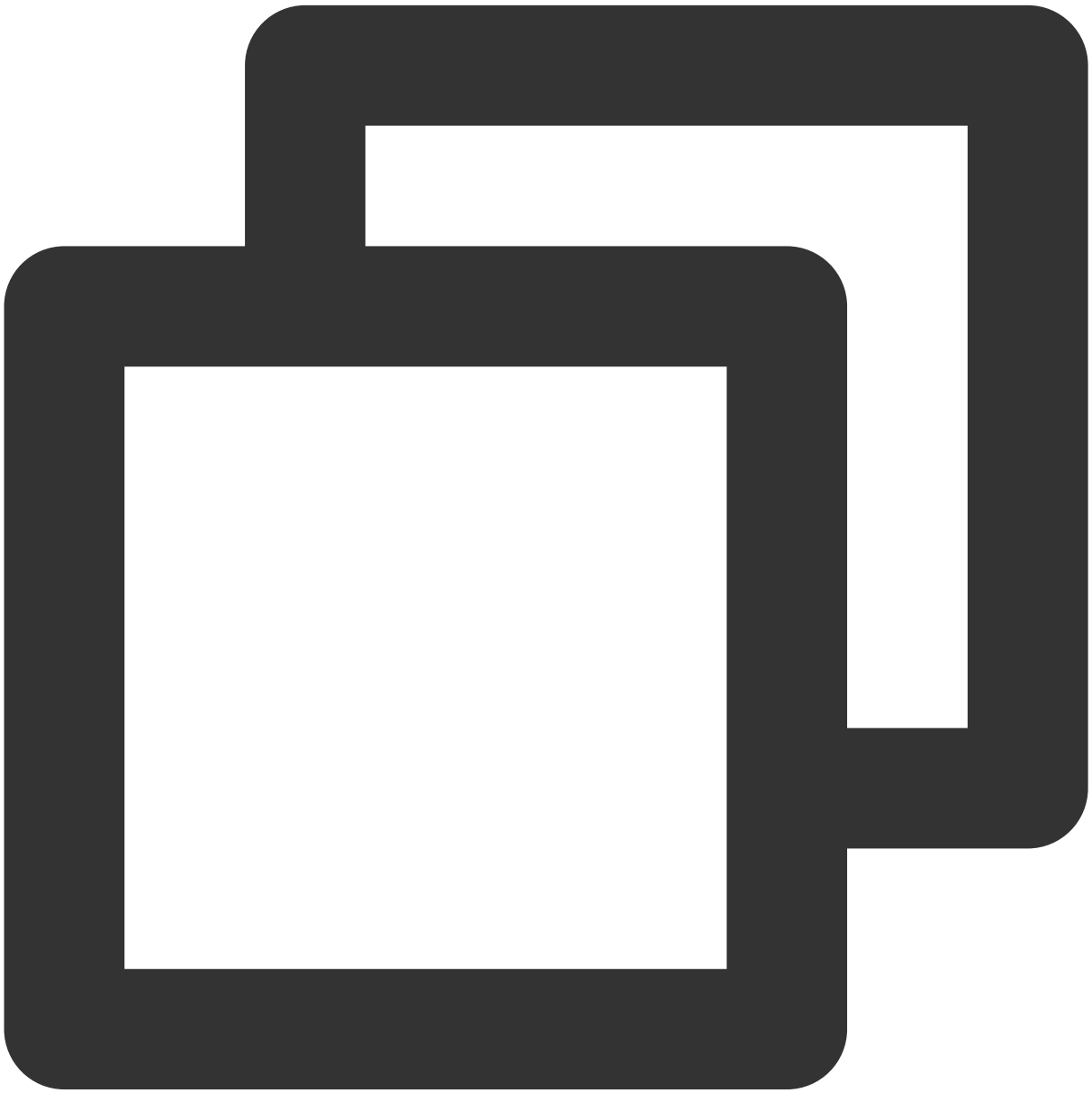


```
OnRoomDismissed onRoomDismissed = (String roomId) {}
```

参数	类型	描述
roomId	String	房间 ID

onKickedOutOfRoom

移出房间事件。

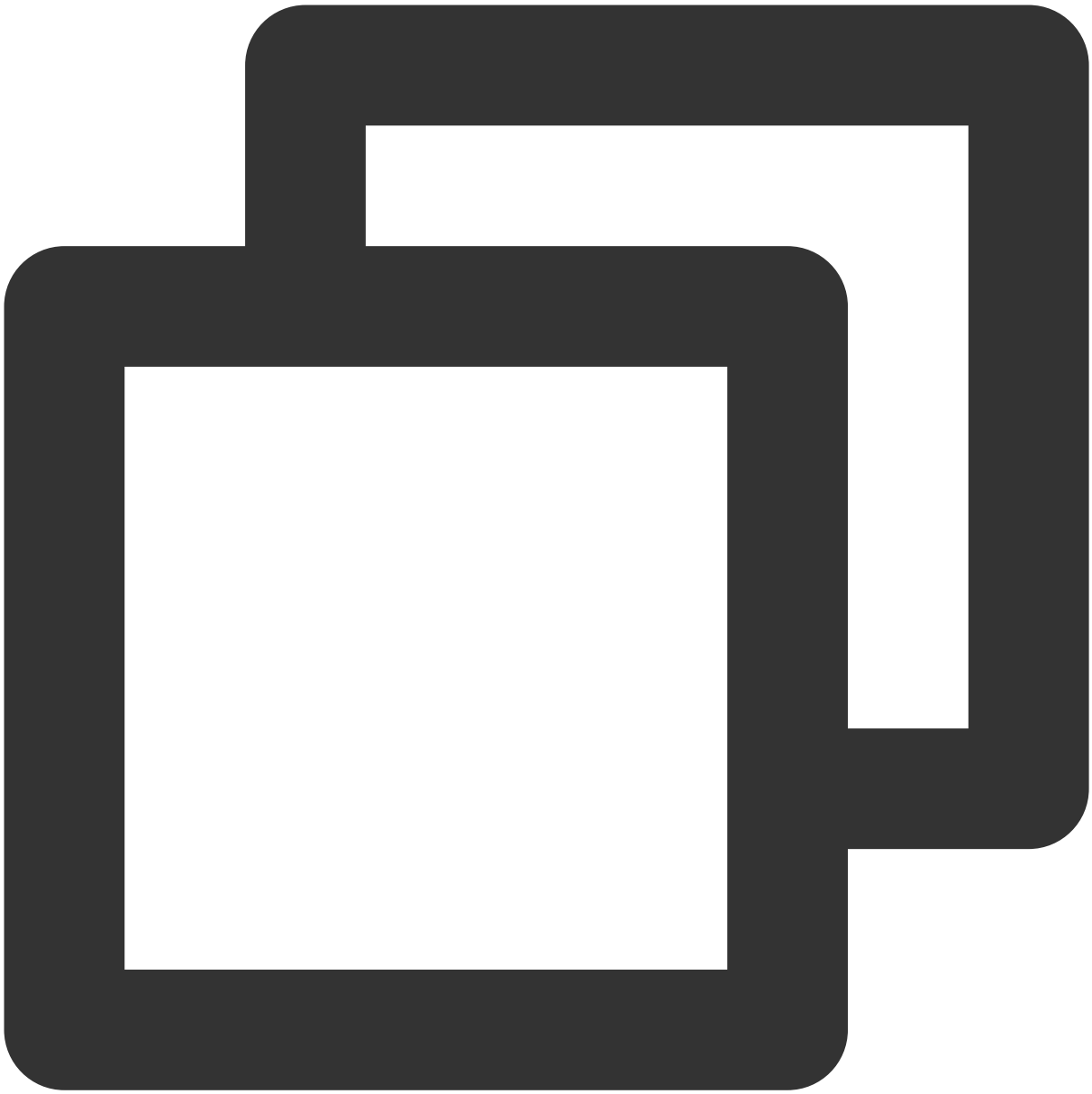


```
OnKickedOutOfRoom onKickedOutOfRoom = (String roomId, String message) {}
```

参数	类型	描述
roomId	String	房间 ID
message	String	被移出的描述

onRoomSpeechModeChanged

房间麦控模式发生变化。

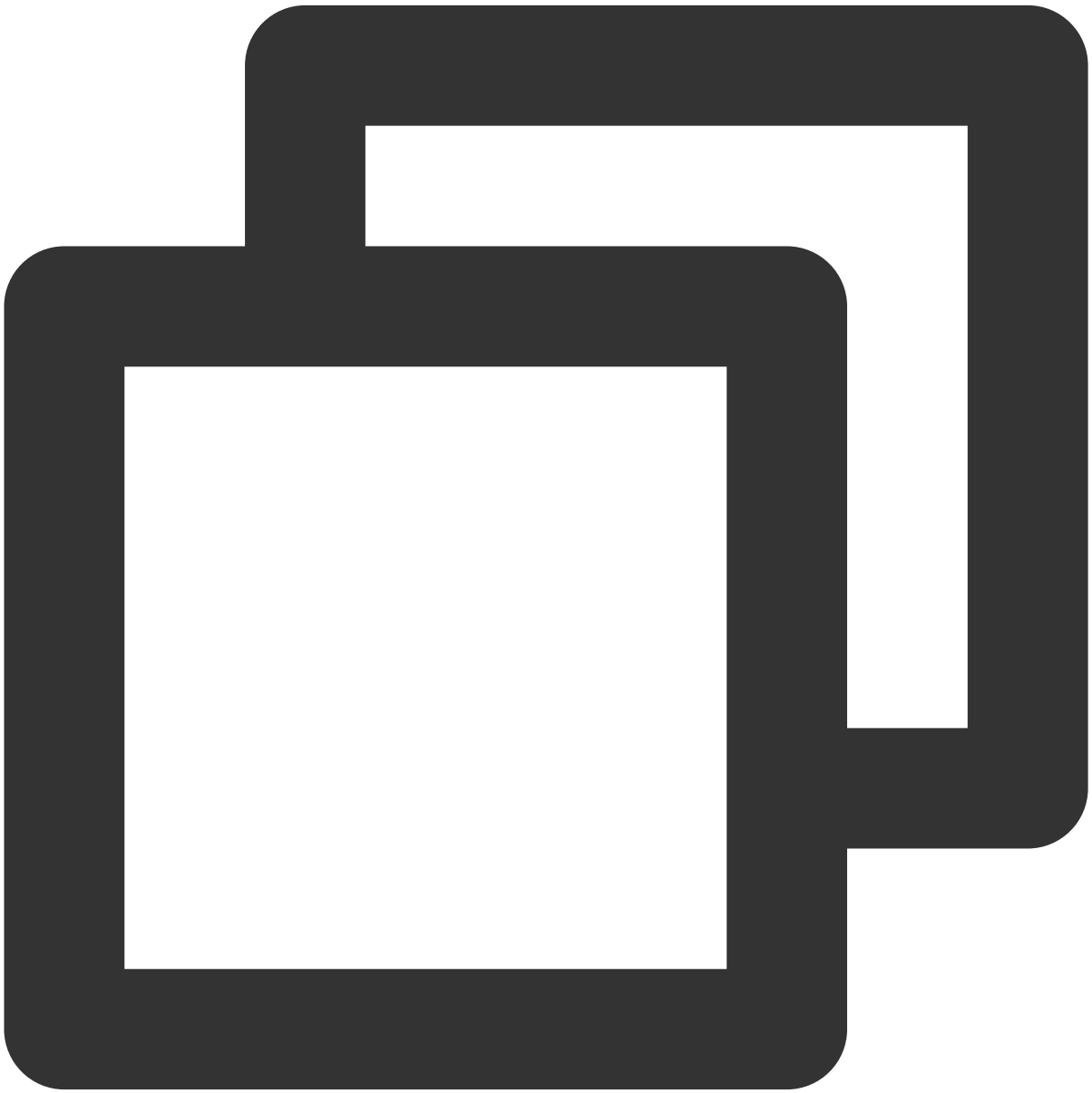


```
OnRoomSpeechModeChanged onRoomSpeechModeChanged = (String roomId, TUISpeechMode spe
```

参数	类型	描述
roomId	String	房间 ID
speechMode	TUISpeechMode	房间麦控模式

onRemoteUserEnterRoom

远端用户进入房间事件。



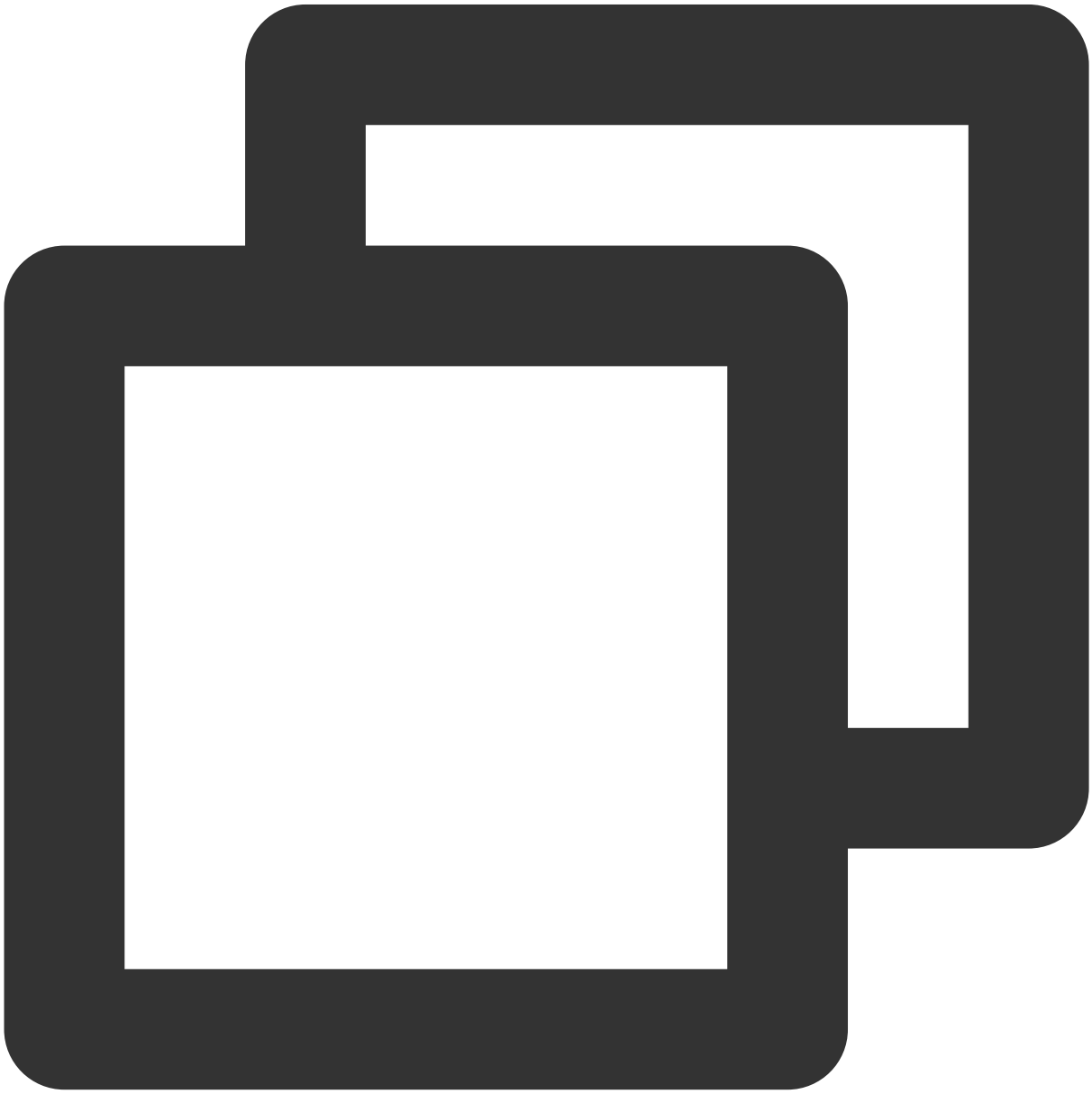
```
OnRemoteUserEnterRoom onRemoteUserEnterRoom = (String roomId, TUIUserInfo userInfo)
```

参数	类型	描述
roomId	String	房间 ID

userInfo	TUIUserInfo	用户信息

onRemoteUserLeaveRoom

远端用户离开房间事件。



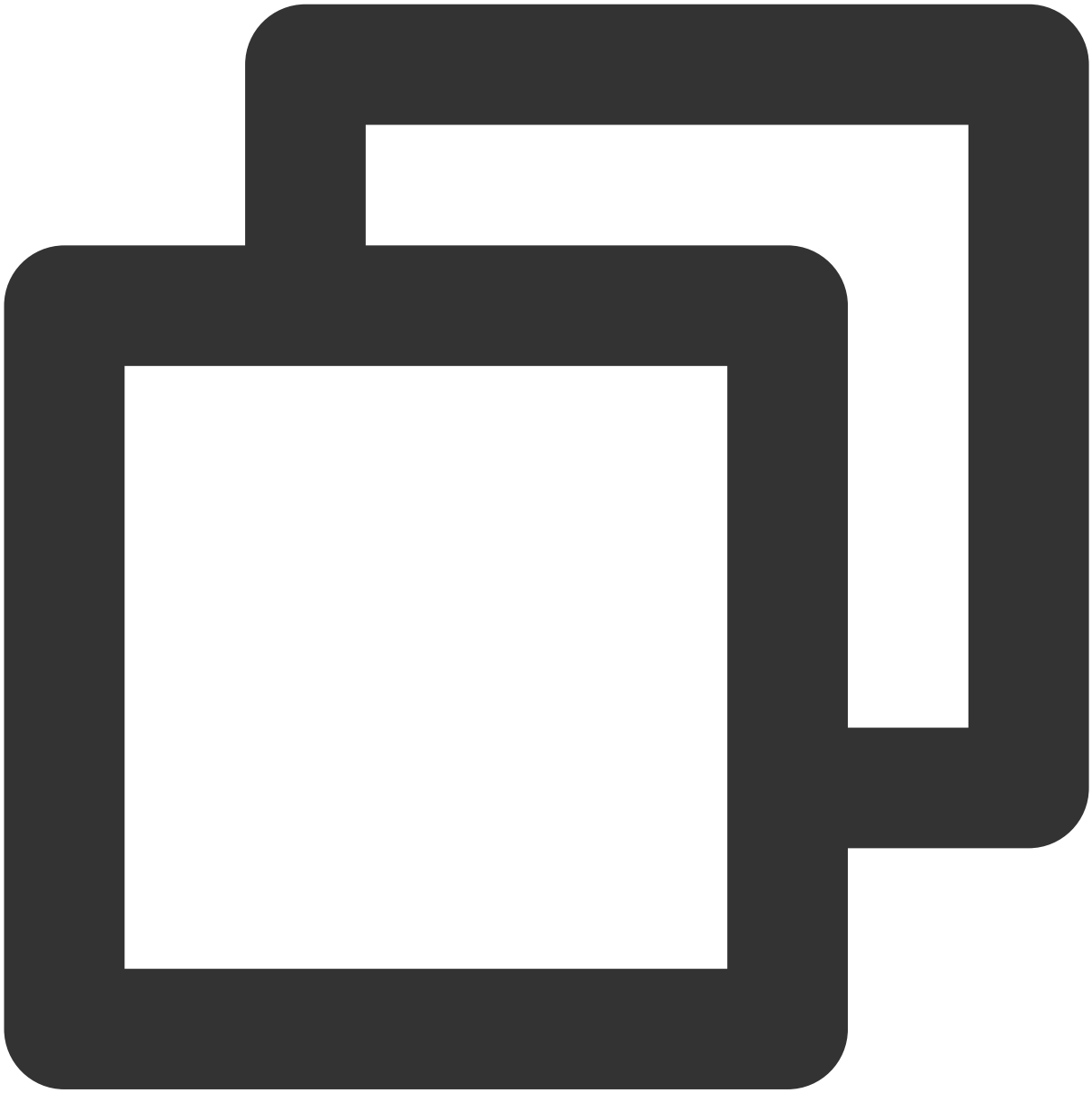
```
OnRemoteUserLeaveRoom onRemoteUserLeaveRoom = (String roomId, TUIUserInfo userInfo)
```

参数	类型	描述
roomId	String	房间 ID

userInfo	TUIUserInfo	用户信息

onUserRoleChanged

用户角色改变事件。



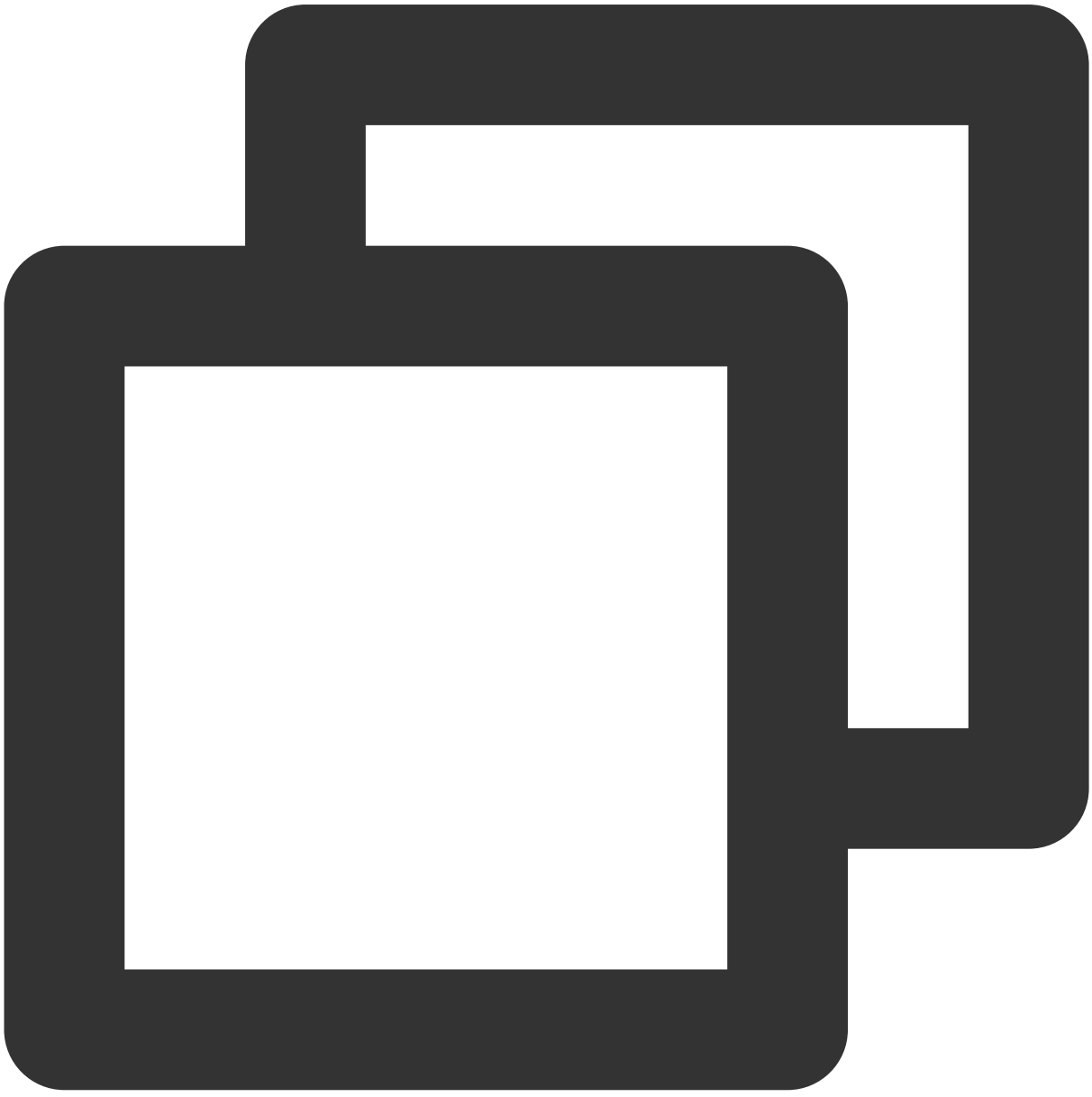
```
OnUserRoleChanged onUserRoleChanged = (String userId, TUIRole role) {}
```

参数	类型	描述
userId	String	用户 ID

role	TUIRole	用户角色

onUserVideoStateChanged

用户视频状态改变事件。



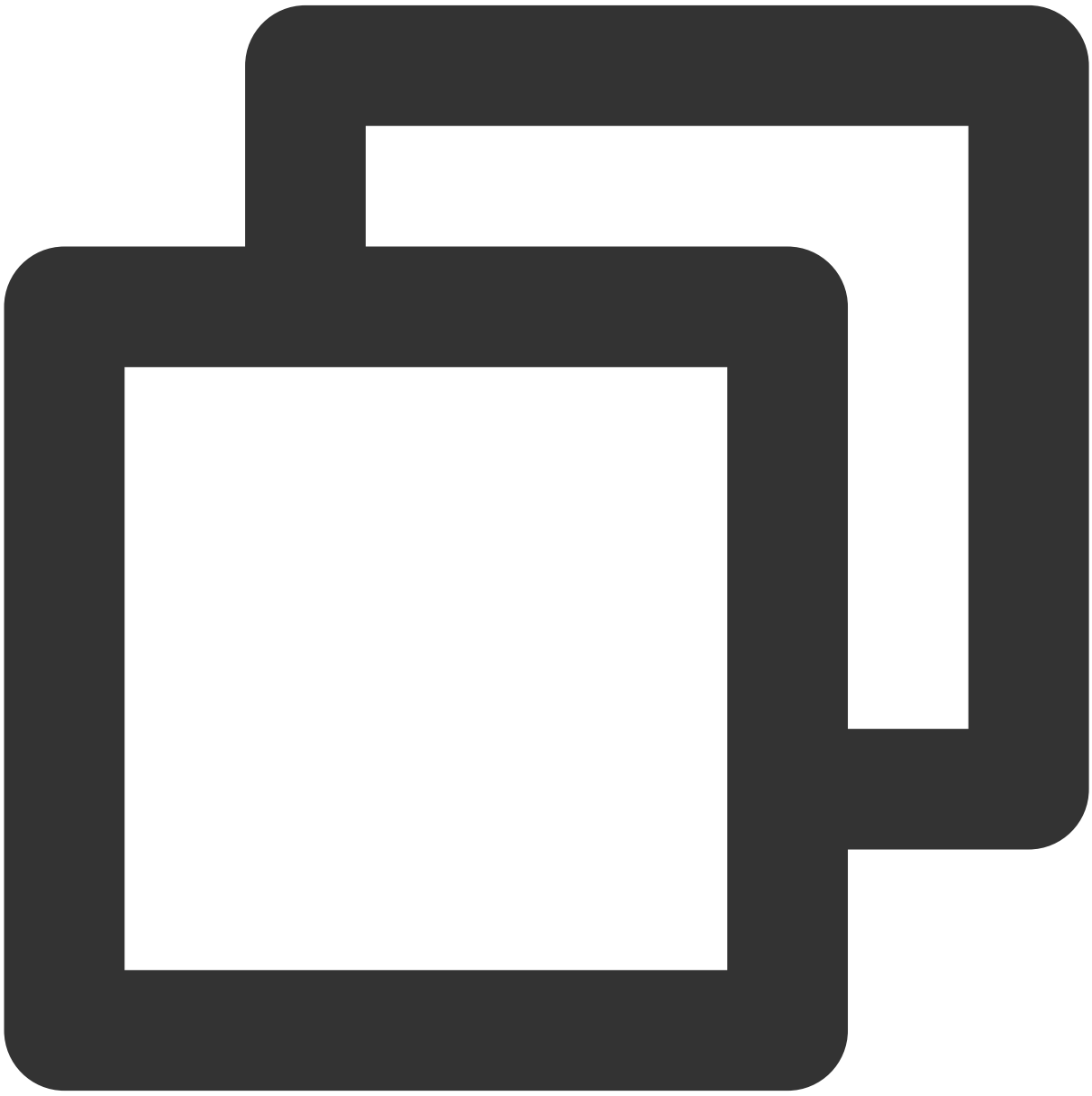
```
OnUserVideoStateChanged onUserVideoStateChanged = (String userId, TUIVideoStreamTyp
```

参数	类型	描述

userId	String	用户 ID
streamType	TUIVideoStreamType	视频流类型
hasVideo	bool	是否有视频流
reason	TUIChangeReason	视频流发生变化原因

onUserAudioStateChanged

用户音频状态改变事件。

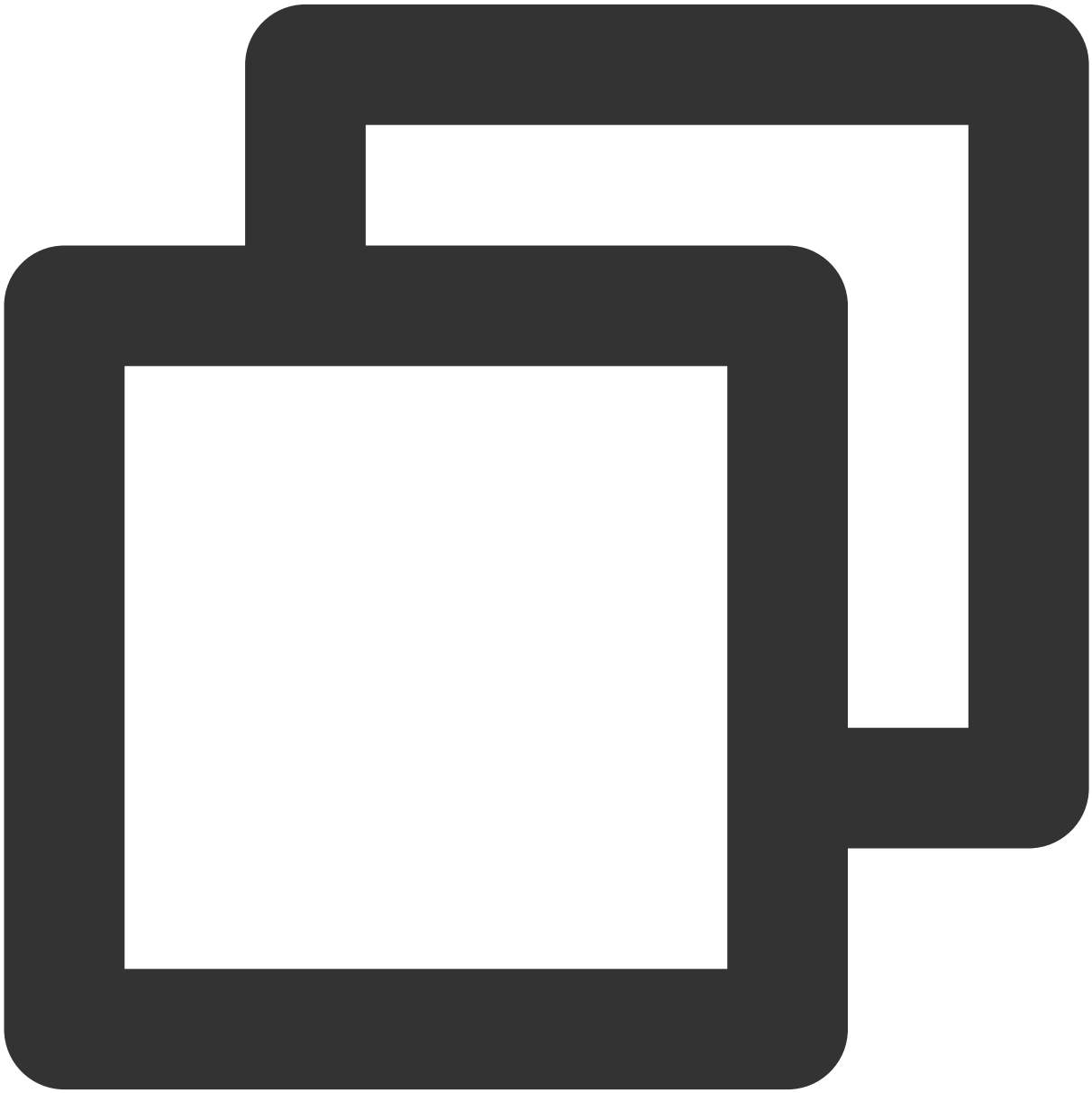


```
OnUserAudioStateChanged onUserAudioStateChanged = (String userId, bool hasAudio, TU
```

参数	类型	描述
userId	String	用户 ID
hasAudio	bool	是否有音频流
reason	TUIChangeReason	视频流发生变化原因

onUserVoiceVolumeChanged

用户音量改变事件。

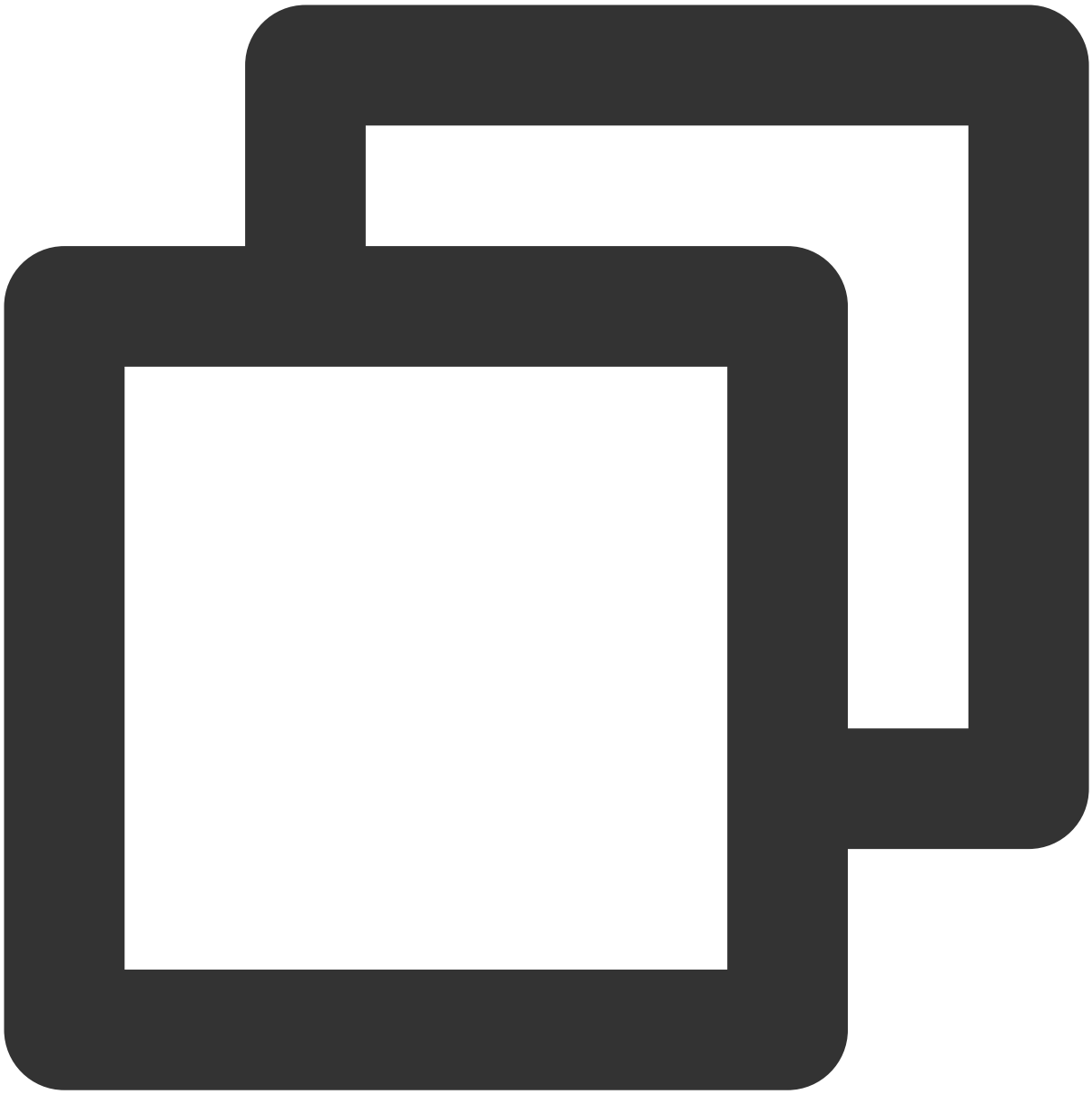


```
OnUserVoiceVolumeChanged onUserVoiceVolumeChanged = (Map<String, int> volumeMap) {}
```

参数	类型	描述
volumeMap	Map	用户音量 Map key：userId value：用于承载所有正在说话的用户的音量大小。 取值范围 0 - 100

onSendMessageForUserDisableChanged

用户文本消息发送能力发生变化事件。

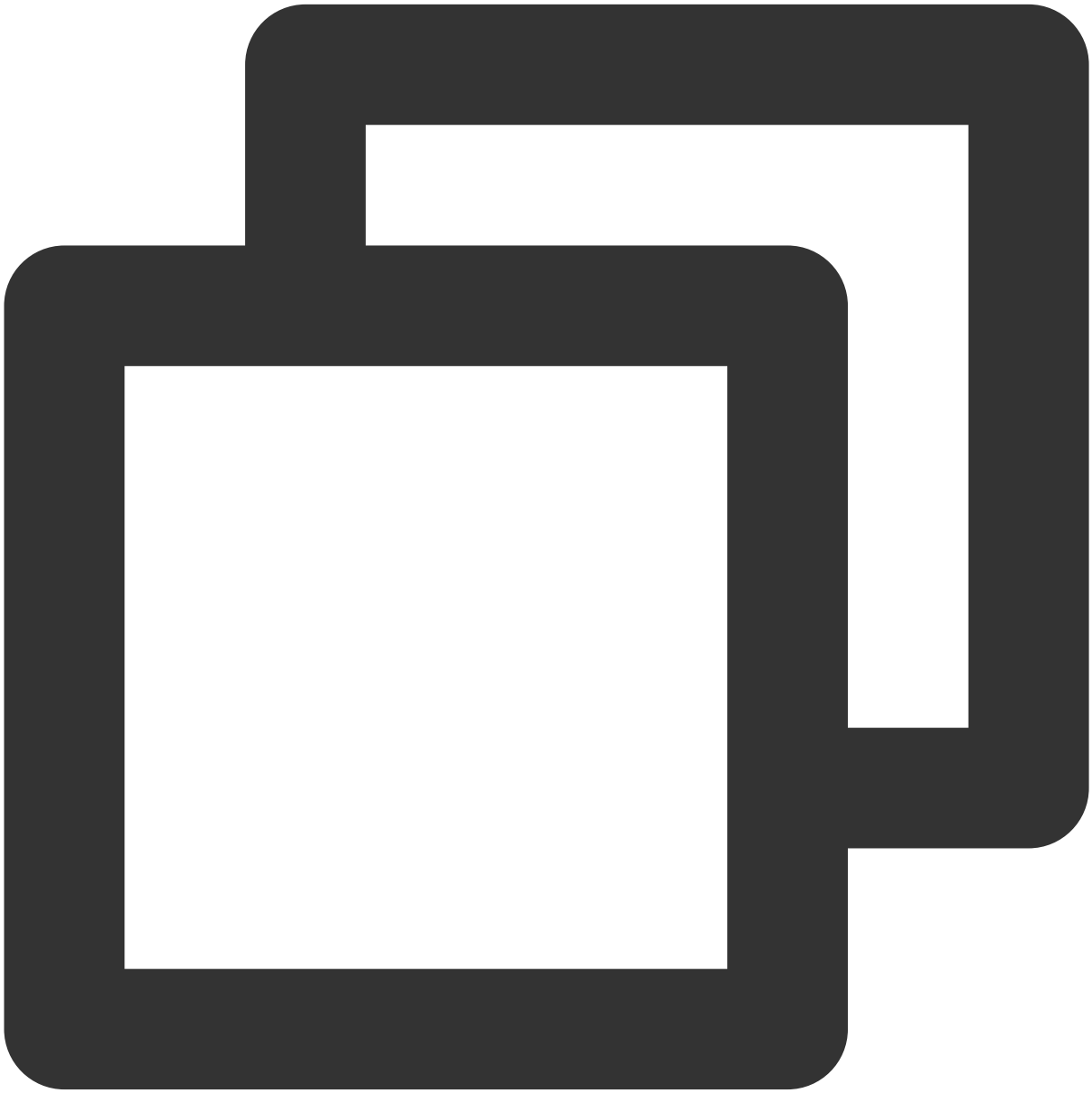


```
OnSendMessageForUserDisableChanged onSendMessageForUserDisableChanged = (String roo
```

参数	类型	描述
roomId	String	房间 ID
userId	String	用户 ID
isDisable	bool	是否被禁止发送文本消息

onUserNetworkQualityChanged

用户网络质量改变事件。

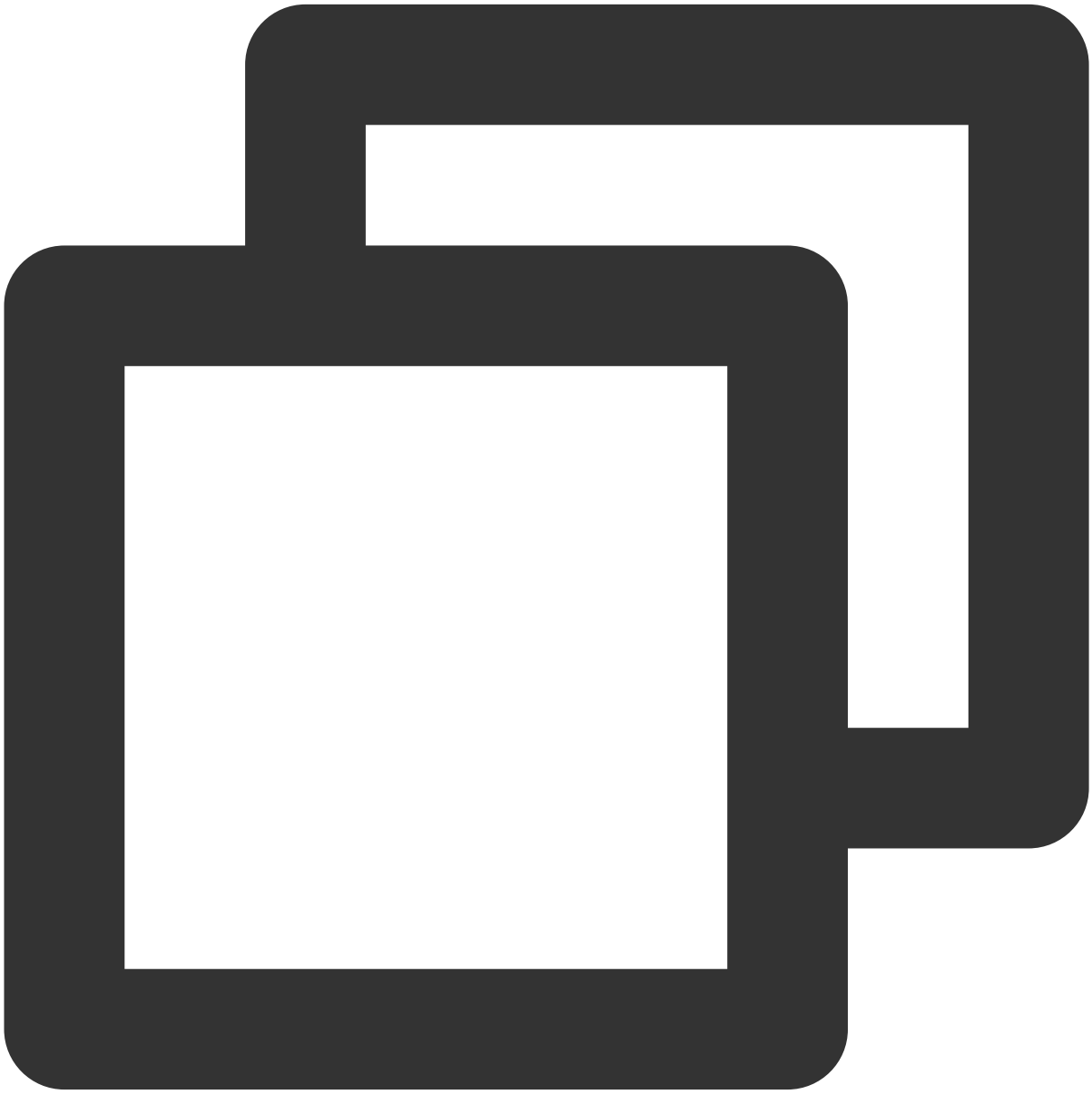


```
OnUserNetworkQualityChanged onUserNetworkQualityChanged = (Map<String, TUINetwork>
```

参数	类型	描述
networkMap	Map	用户网络状态 Map key：userId value：网络状况

onUserScreenCaptureStopped

屏幕分享结束事件。



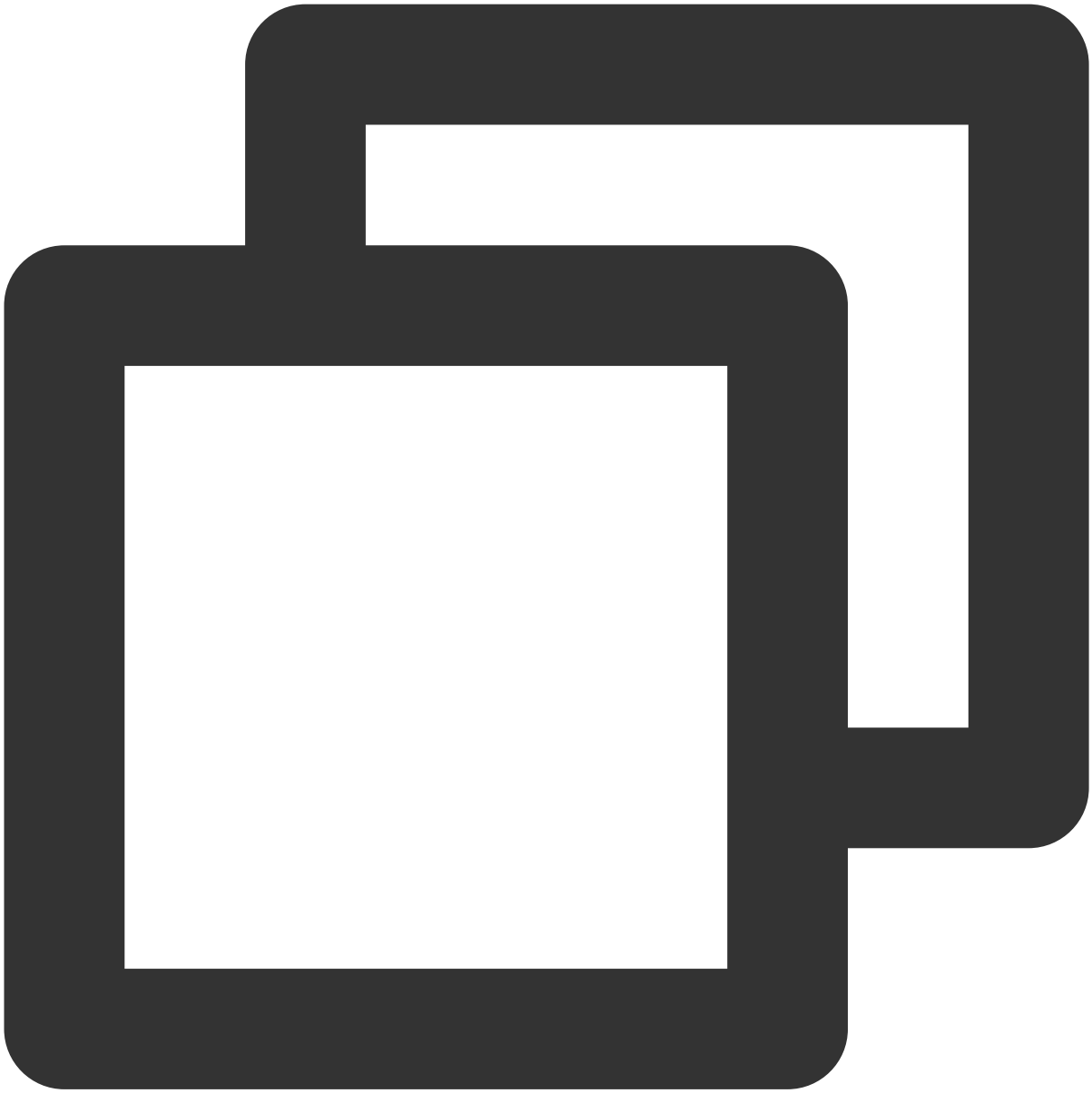
```
OnUserScreenCaptureStopped onUserScreenCaptureStopped = (int reason) {}
```

参数	类型	描述
reason	int	停止原因 0：用户主动停止 1：屏幕窗口关闭导致停止

		2：表示屏幕分享的显示屏状态变更（如接口被拔出、投影模式变更等）

onRoomMaxSeatCountChanged

房间内最大麦位数发生变化事件（仅在会议类型房间生效）。



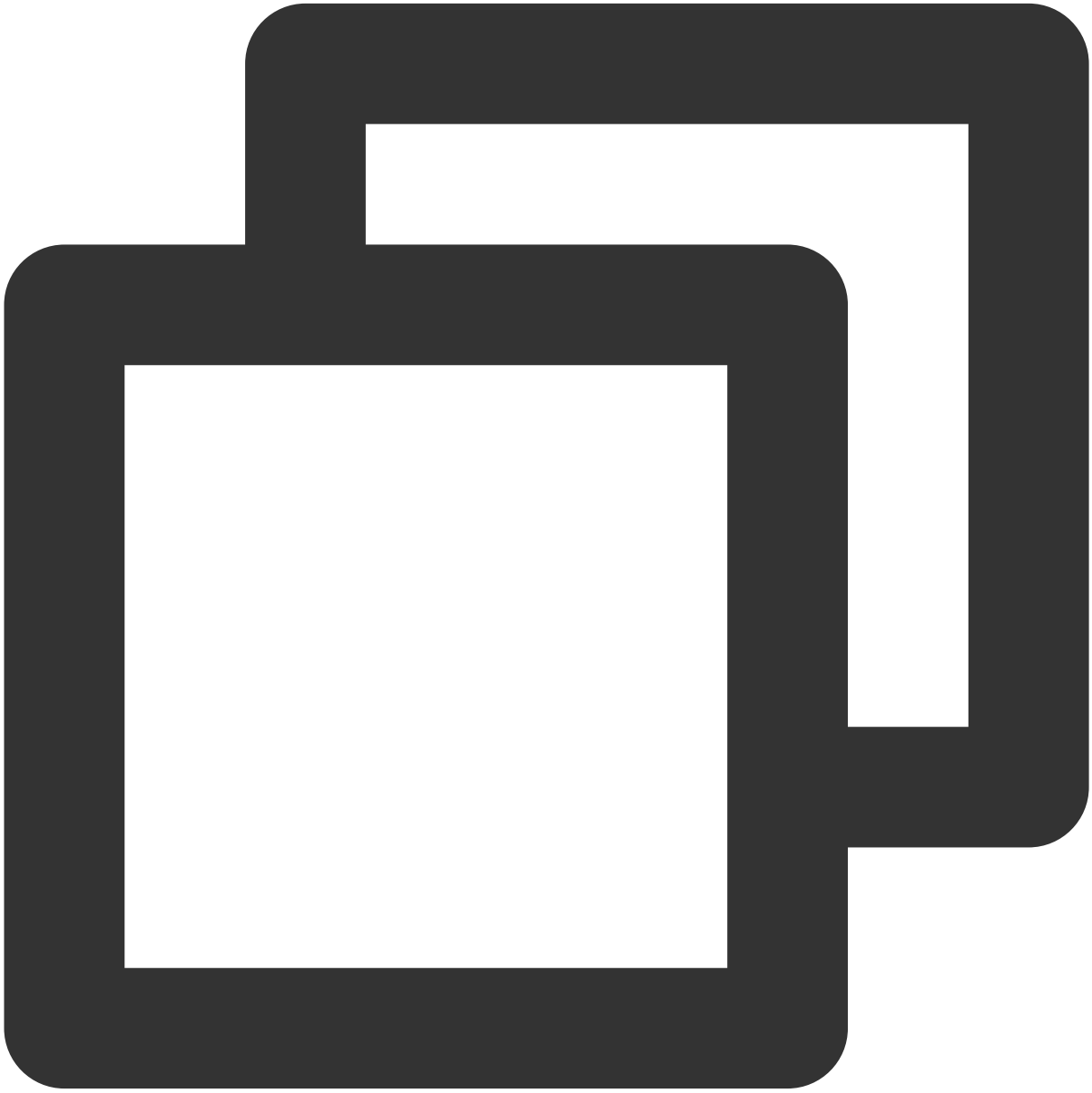
```
OnRoomMaxSeatCountChanged onRoomMaxSeatCountChanged = (String roomId, int maxSeatCo
```

参数	类型	描述

roomId	String	房间 ID
maxSeatCount	int	房间内最大麦位数量

onSeatListChanged

麦位列表改变事件。



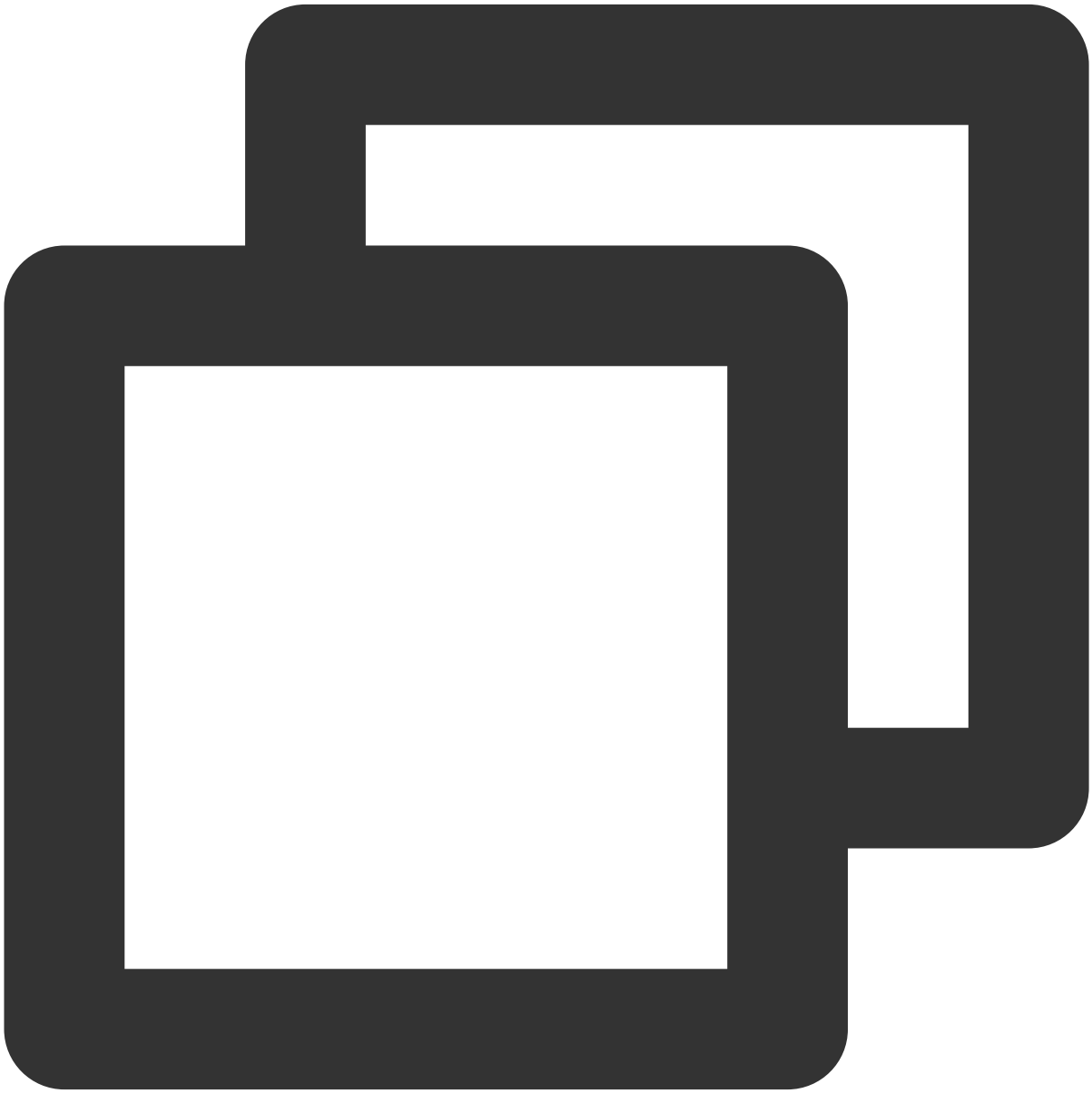
```
OnSeatListChanged onSeatListChanged = (List<TUISeatInfo> seatList, List<TUISeatInfo
```

参数	类型	描述

seatList	List<TUISeatInfo>	目前麦上最新的用户列表，包含新上麦的用户
seatedList	List<TUISeatInfo>	新上麦的用户列表
leftList	List<TUISeatInfo>	新下麦的用户列表

onKickedOffSeat

收到用户被移下麦事件。

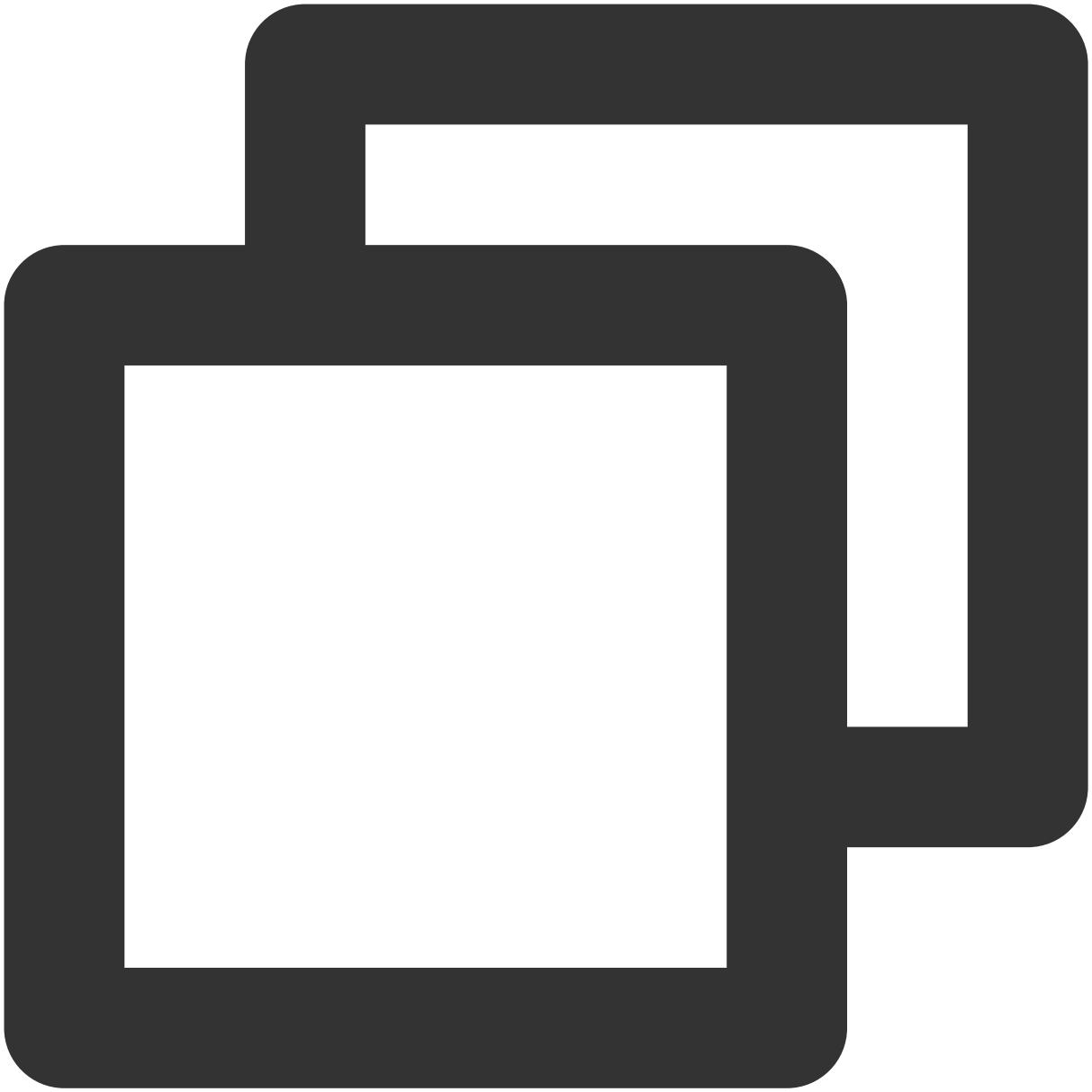


```
OnKickedOffSeat onKickedOffSeat = (String userId) {}
```

参数	类型	描述
userId	String	操作移人的（主持人/管理员）用户 ID

onRequestReceived

请求接收事件。

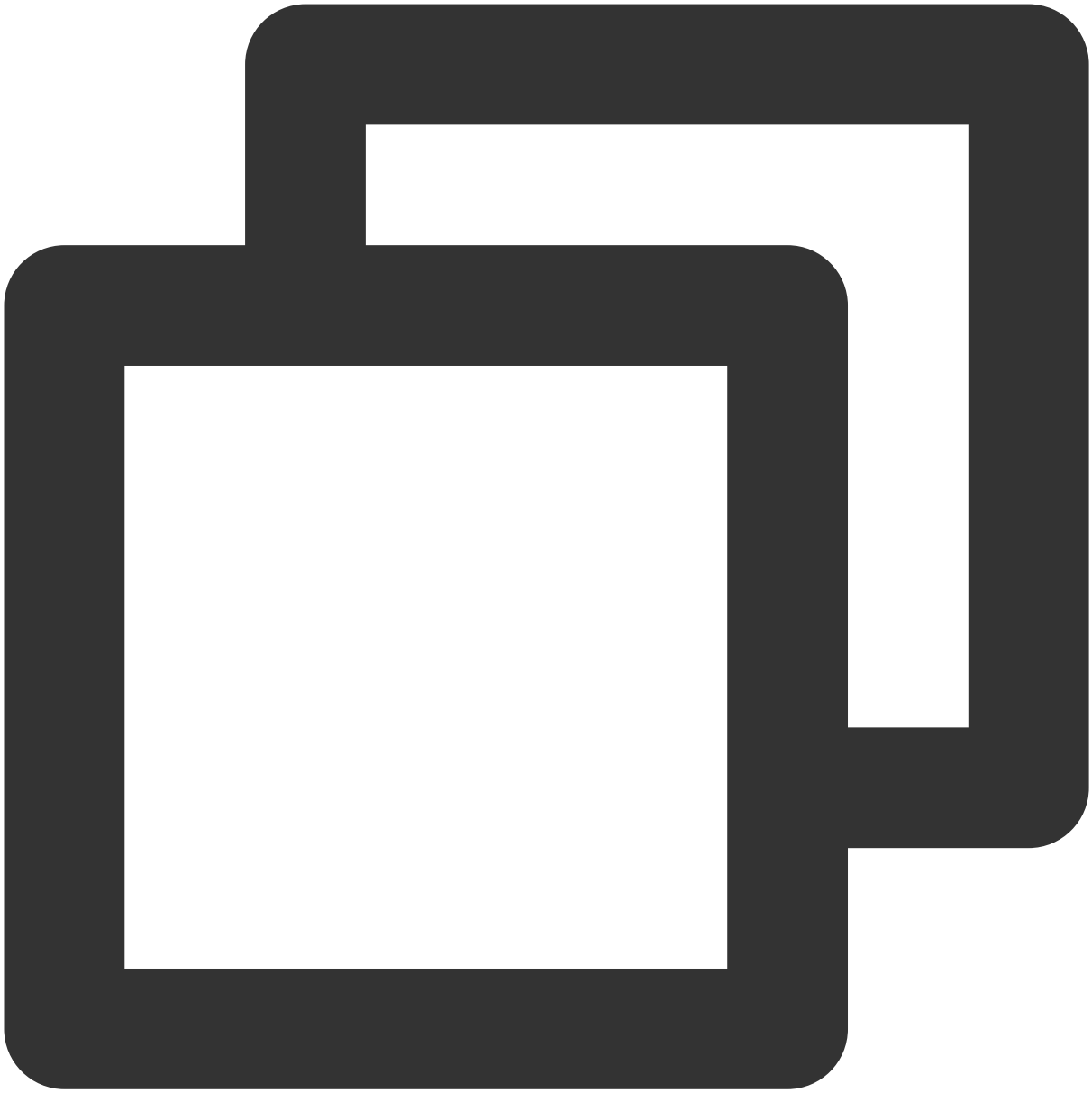


```
OnRequestReceived onRequestReceived = (TUIRequest request) {}
```

参数	类型	描述
request	TUIRequest	请求内容

onRequestCancelled

请求取消事件。

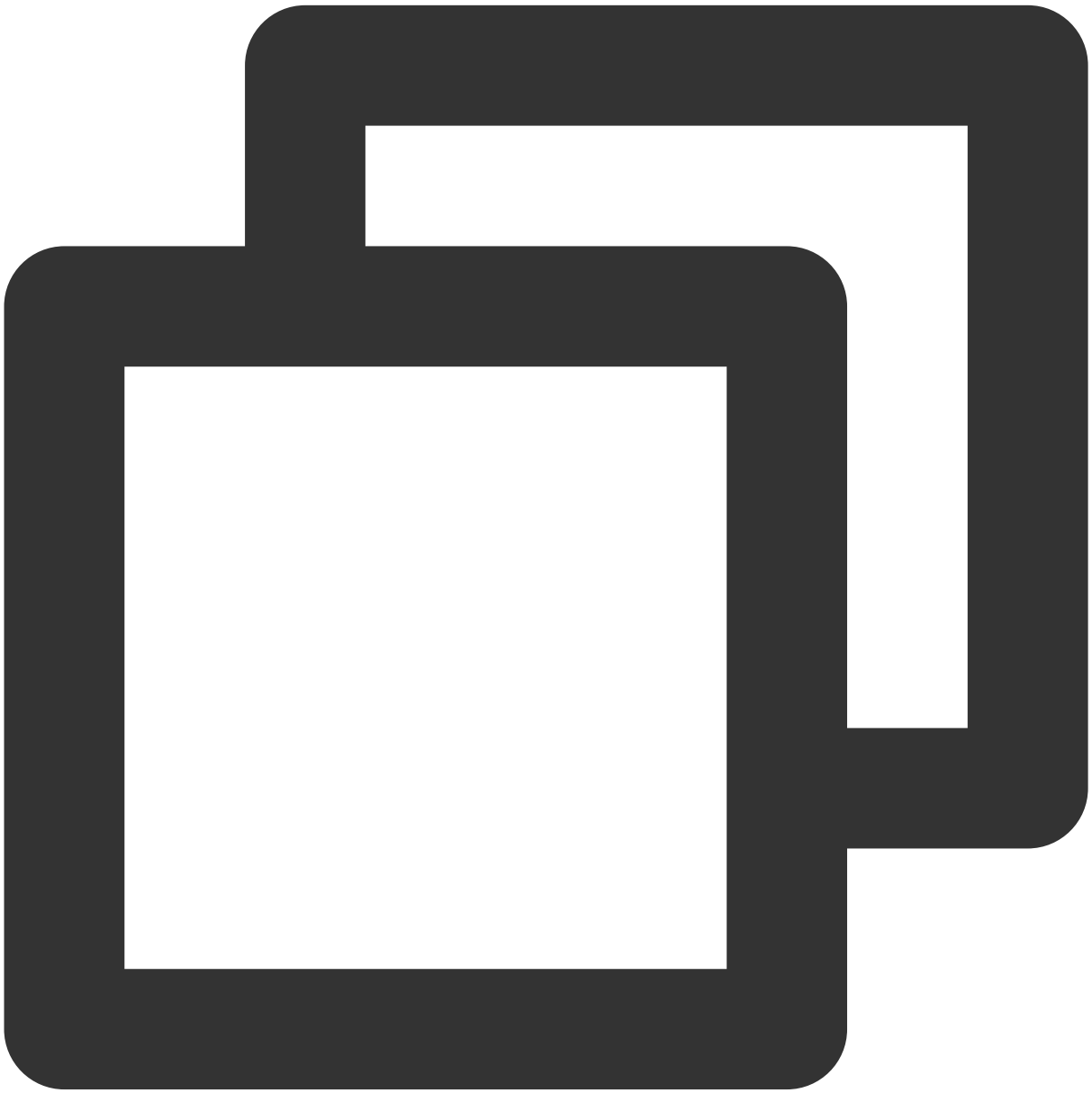


```
OnRequestCancelled onRequestCancelled = (String requestId, String userId) {}
```

参数	类型	描述
requestId	String	请求 ID
userId	String	取消信令的用户 ID

onReceiveTextMessage

接收文本消息事件。

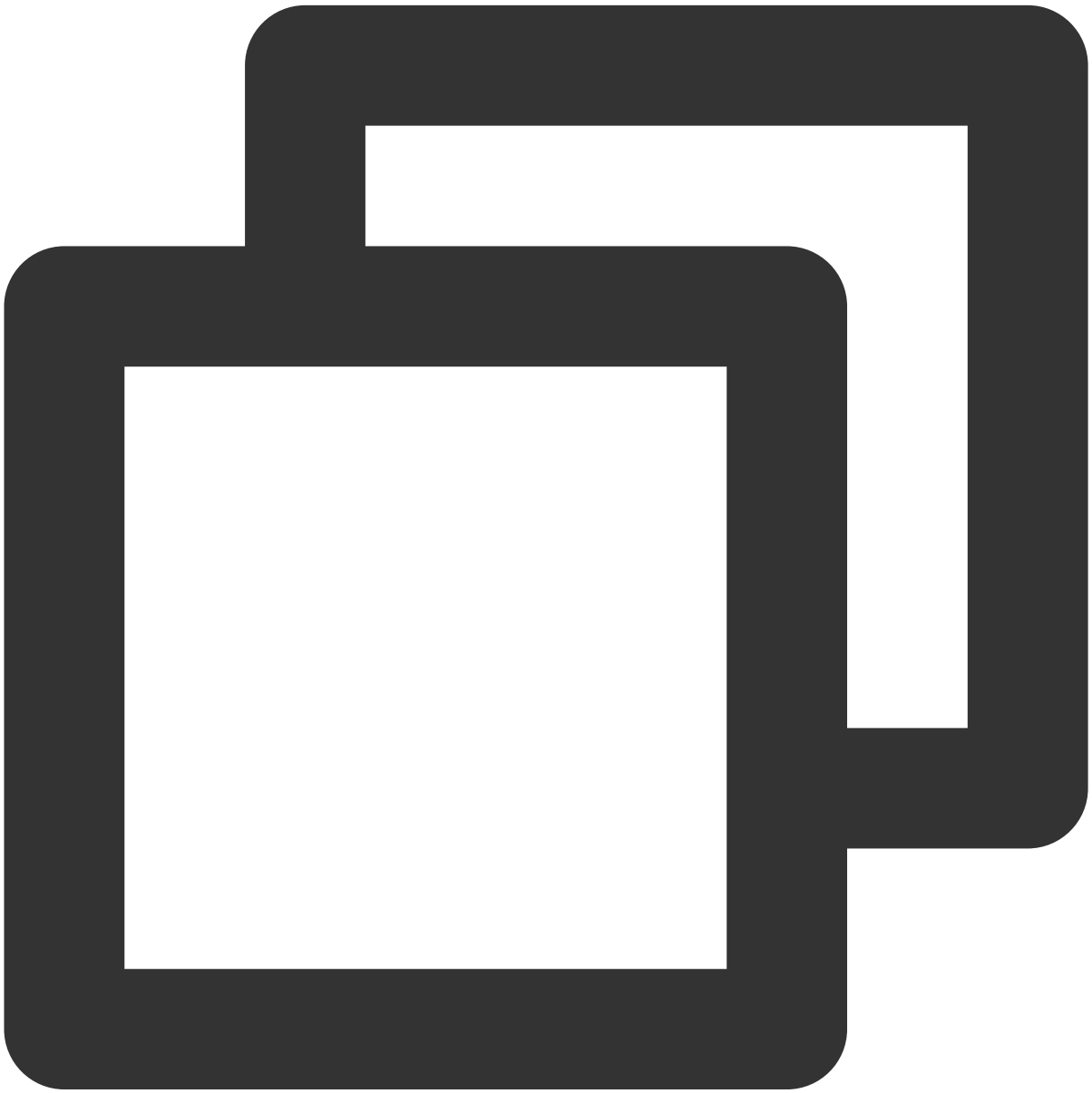


```
OnReceiveTextMessage onReceiveTextMessage = (String roomId, TUIMessage message) {}
```

参数	类型	描述
roomId	String	房间 ID
message	TUIMessage	消息内容

onReceiveCustomMessage

接收自定义消息事件。



```
OnReceiveCustomMessage onReceiveCustomMessage = (String roomId, TUIMessage message)
```

参数	类型	描述
roomId	String	房间 ID
message	TUIMessage	消息内容

版权所有：腾讯云计算（北京）有限责任公司

TUIRoomEngine

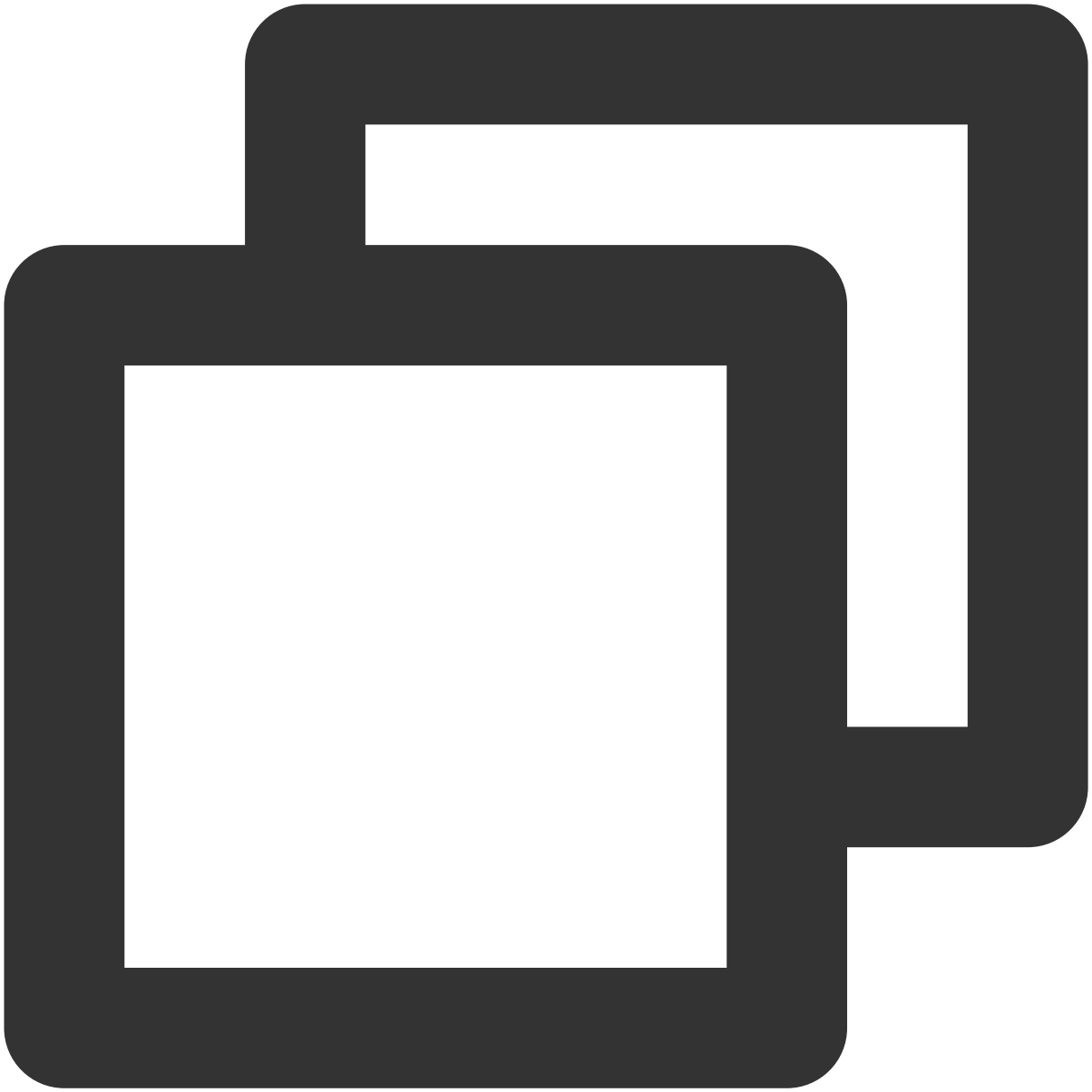
最近更新时间：2023-11-21 15:01:34

TUIRoomEngine API 简介

TUIRoomEngine API 是多人音视频房间的无 UI 接口，您可以使用这套 API 根据您的业务需求自定义封装。

createInstance

创建 TUIRoomEngine 实例。

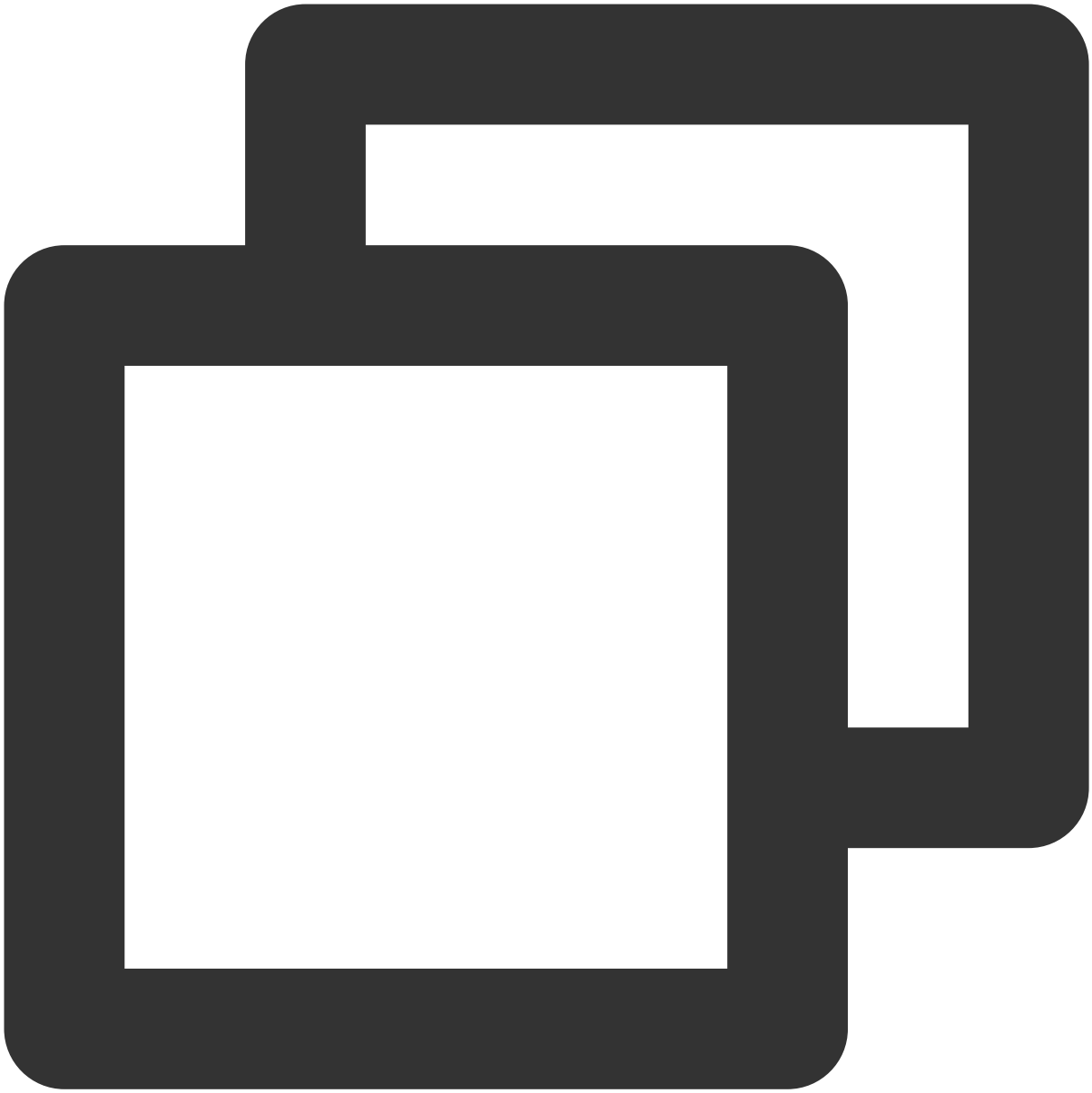


```
static TUIRoomEngine createInstance()
```

return:TUIRoomEngine 实例。

destroyInstance

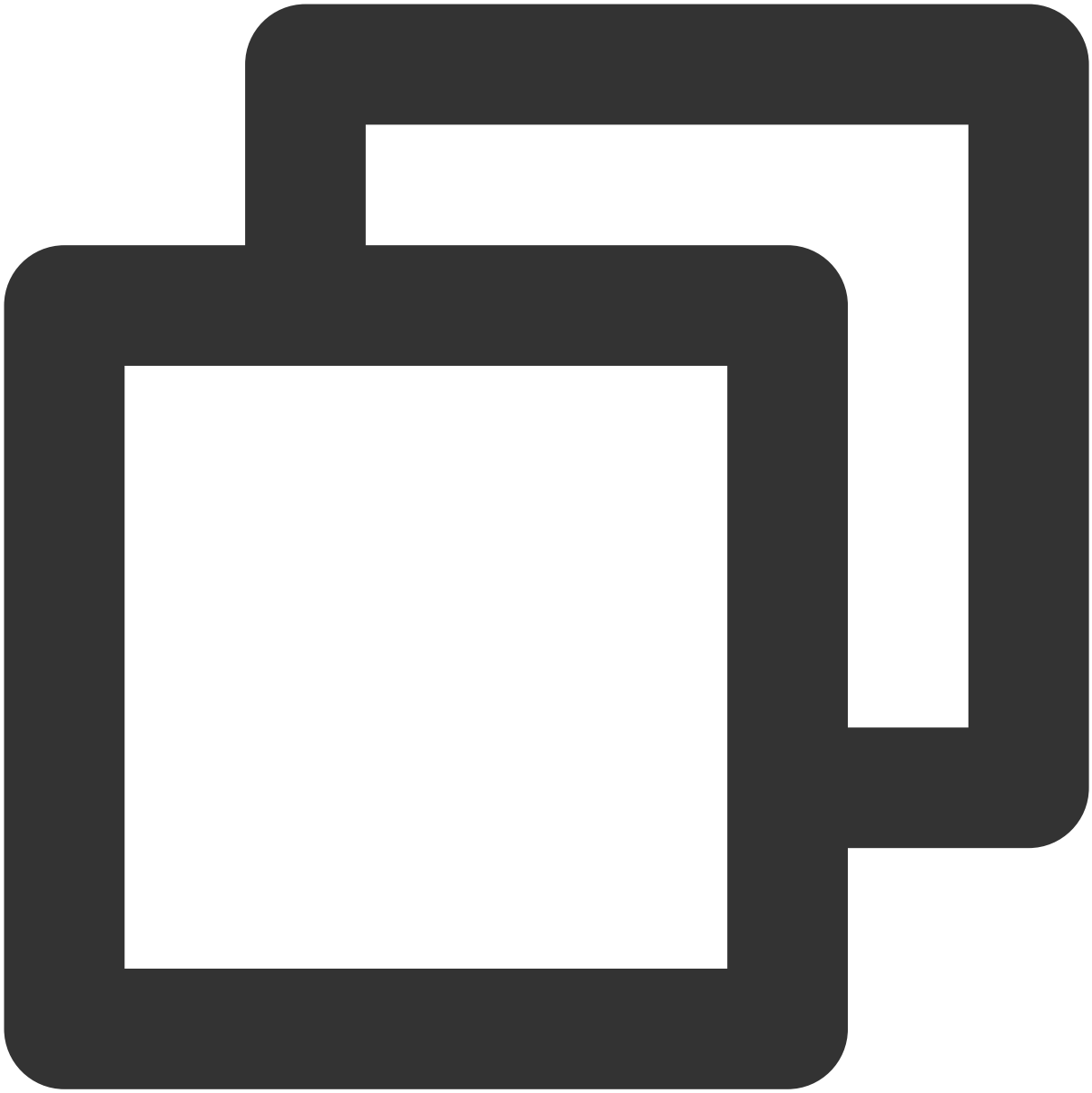
销毁TUIRoomEngine实例。



```
void destroyInstance()
```

login

登录 roomEngine 接口，您需要先初始化用户信息后才能进入房间，并进行一系列的操作。



```
static Future<TUIActionCallback> login(int sdkAppId,
                                       String userId,
                                       String userSig)
```

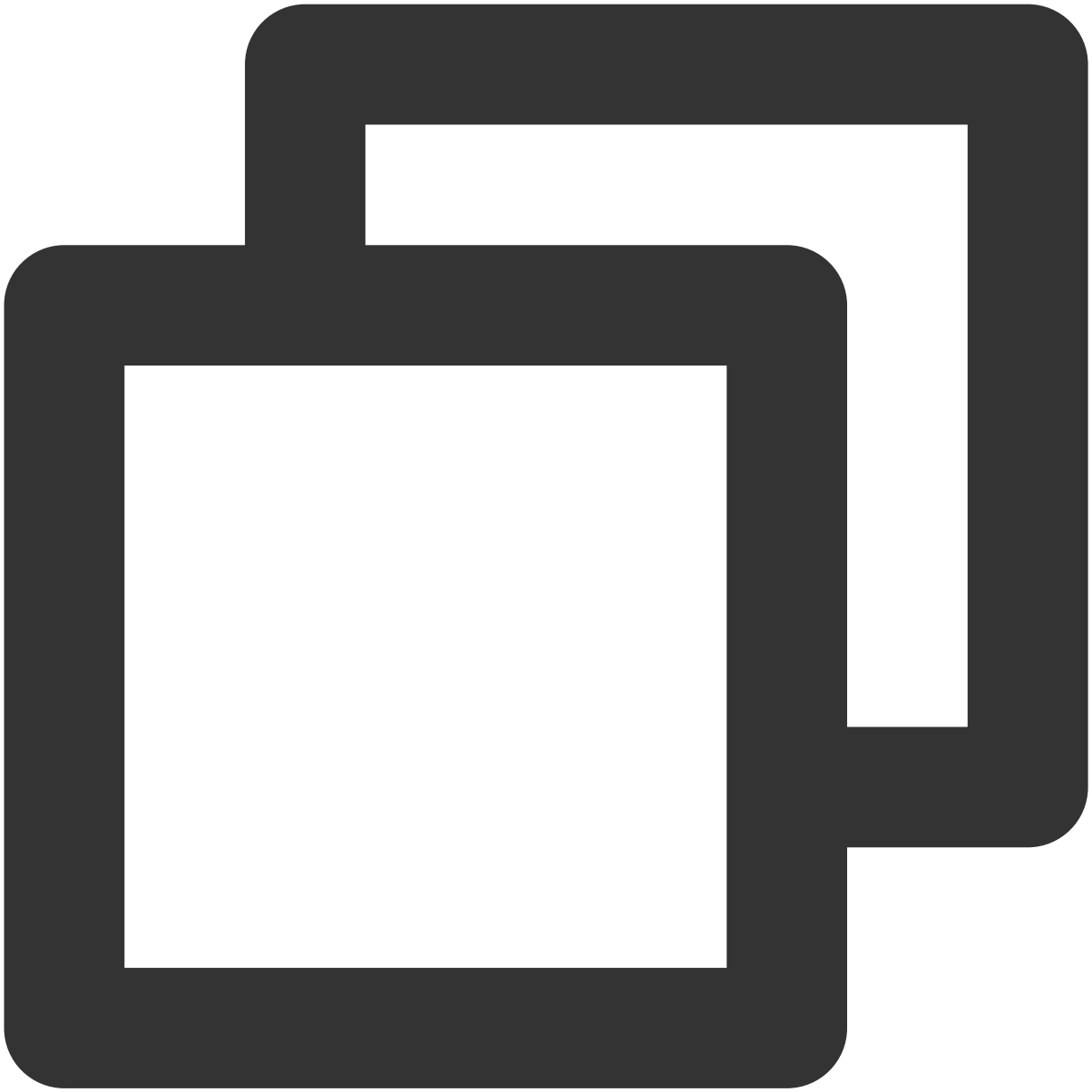
参数：

参数	类型	含义
sdkAppId	int	应用信息中获取 sdkAppId 信息
userId	String	用户 ID

userSig	String	userSig 签名。计算 userSig 的方式请参考UserSig 相关

logout

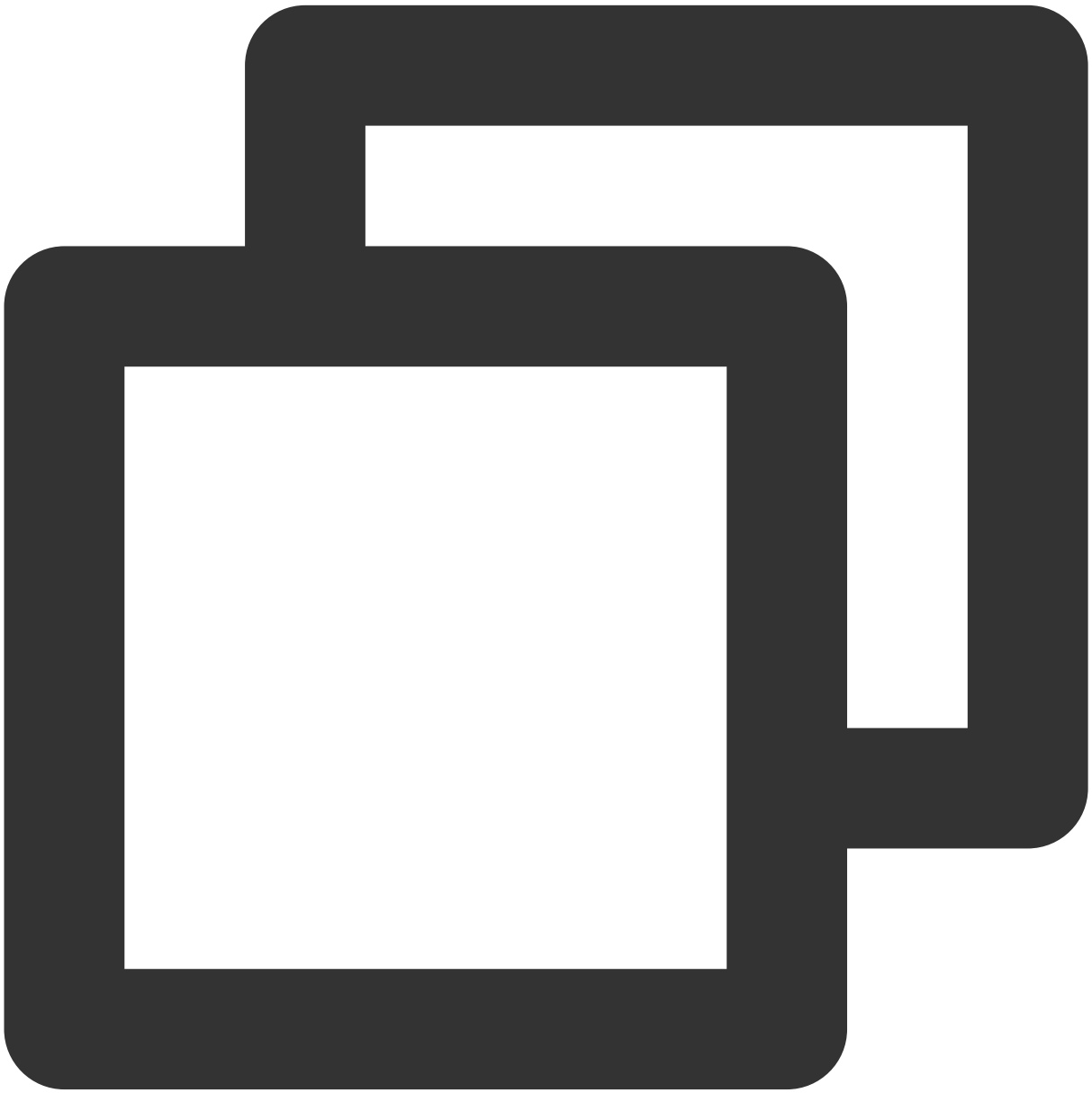
退出登录接口，会有主动离开房间操作、销毁资源。



```
static Future<TUIActionCallback> logout()
```

setSelfInfo

设置本地用户名称和头像。



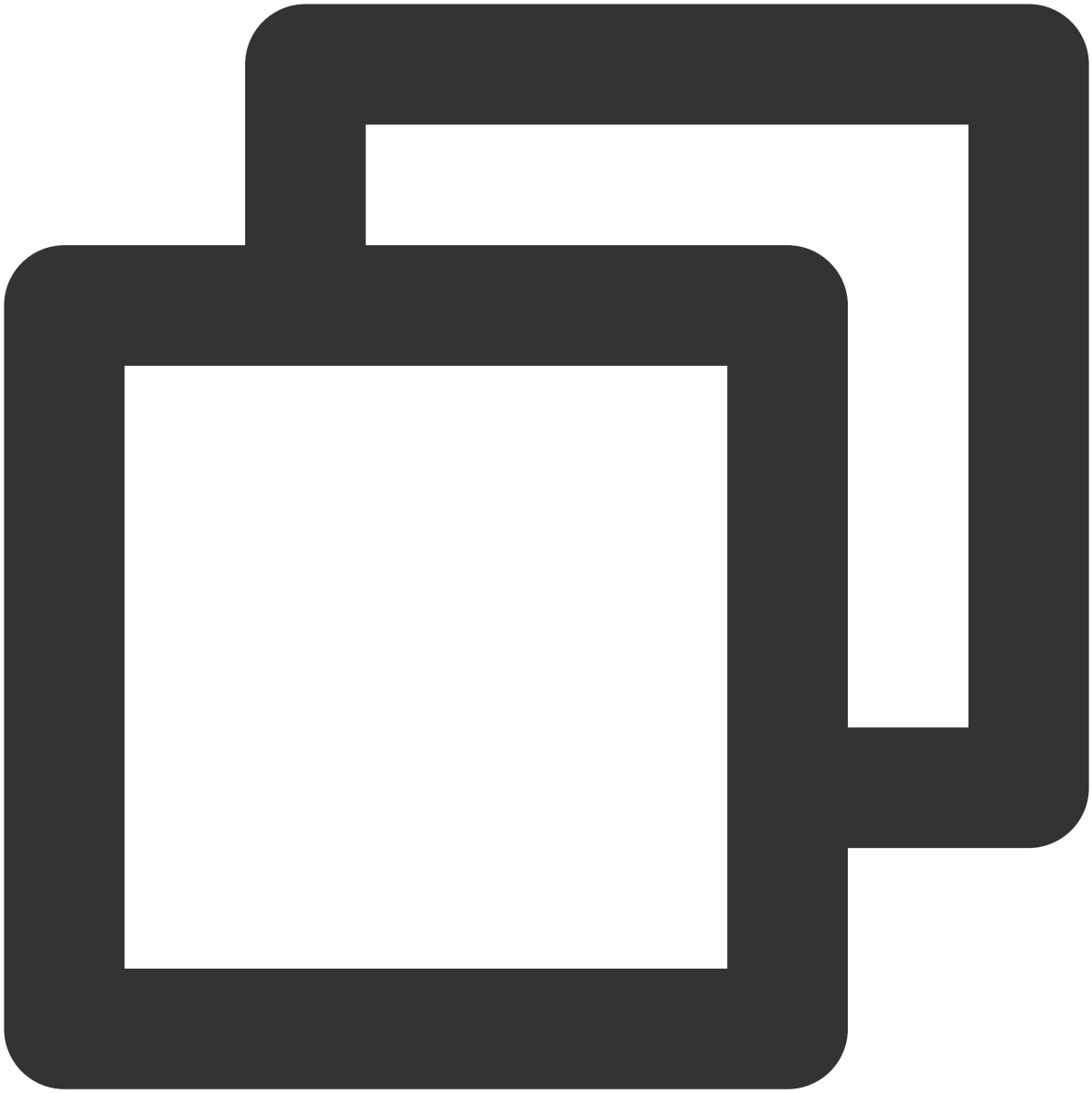
```
static Future<TUIActionCallback> setSelfInfo(String userName, String avatarURL)
```

参数：

参数	类型	含义
userName	String	用户名
avatarUrl	String	用户头像

setLoginUserInfo

设置登录用户信息。



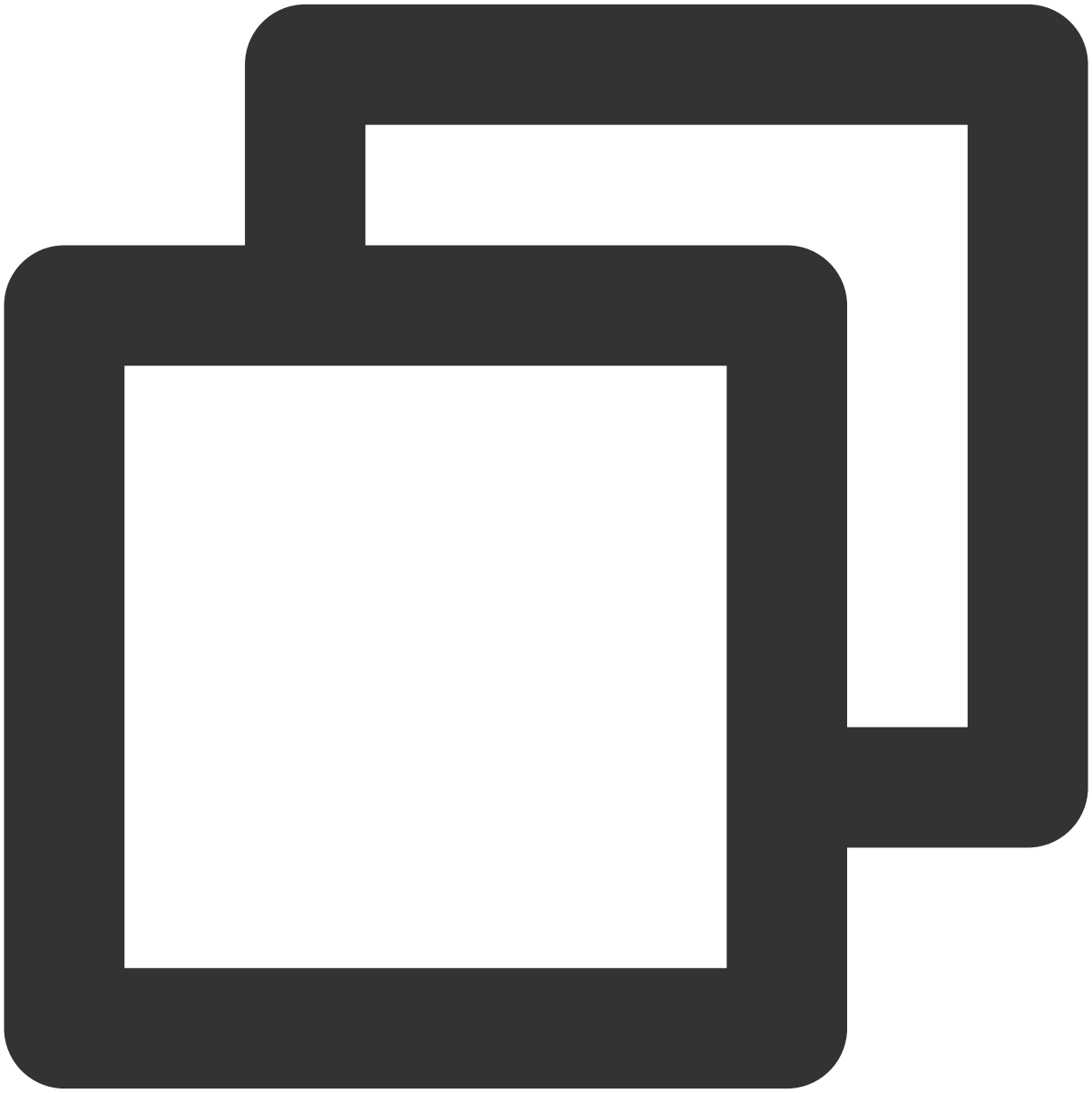
```
static Future<TUIActionCallback> setLoginUserInfo(TUILoginUserInfo userInfo)
```

参数：

参数	类型	含义
userInfo	TUILoginUserInfo	用户信息

getSelfInfo

获取本地用户登录的基本信息。

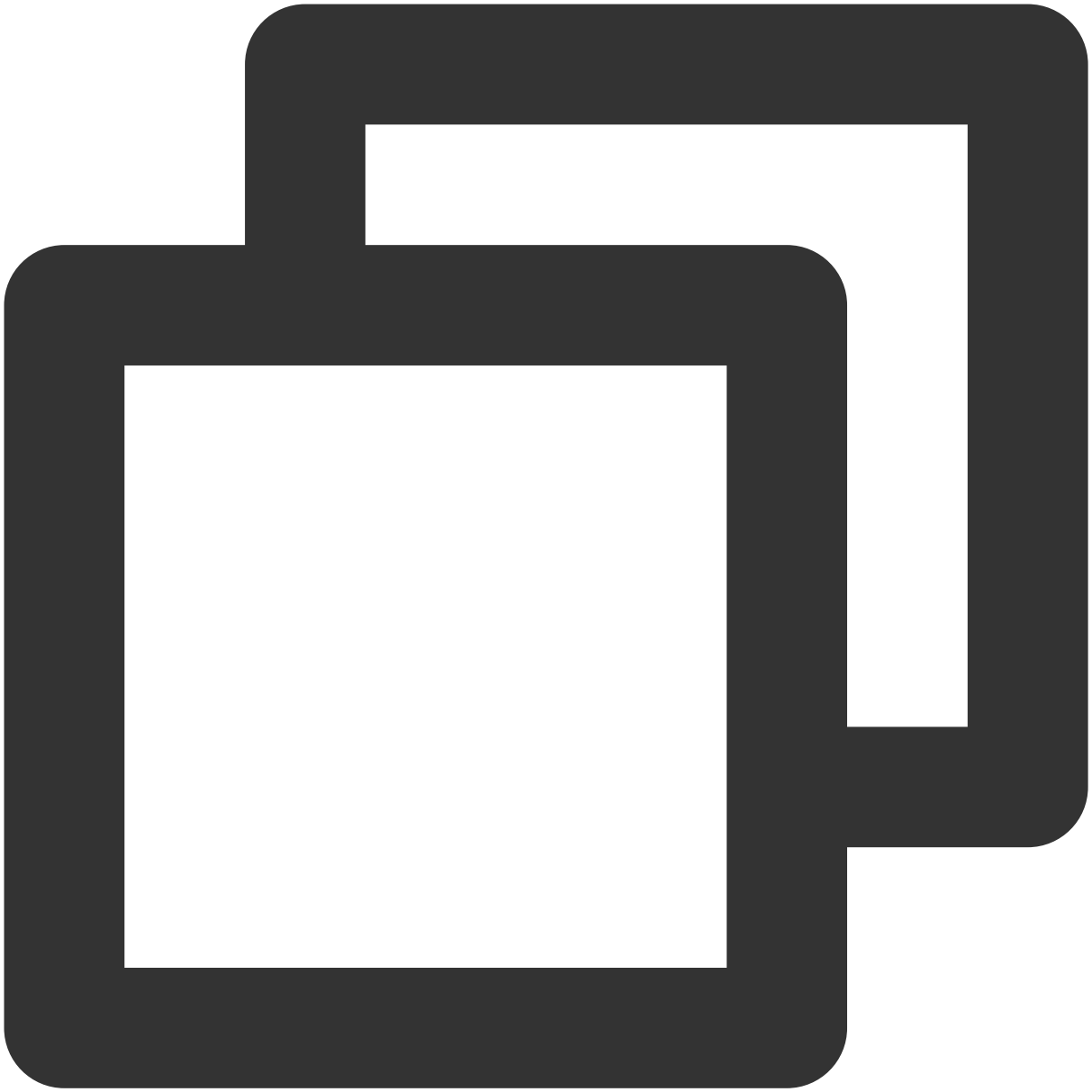


```
static TUILoginUserInfo getSelfInfo()
```

return:用户登录信息。

addObserver

添加 TUIRoomEngine 事件回调。



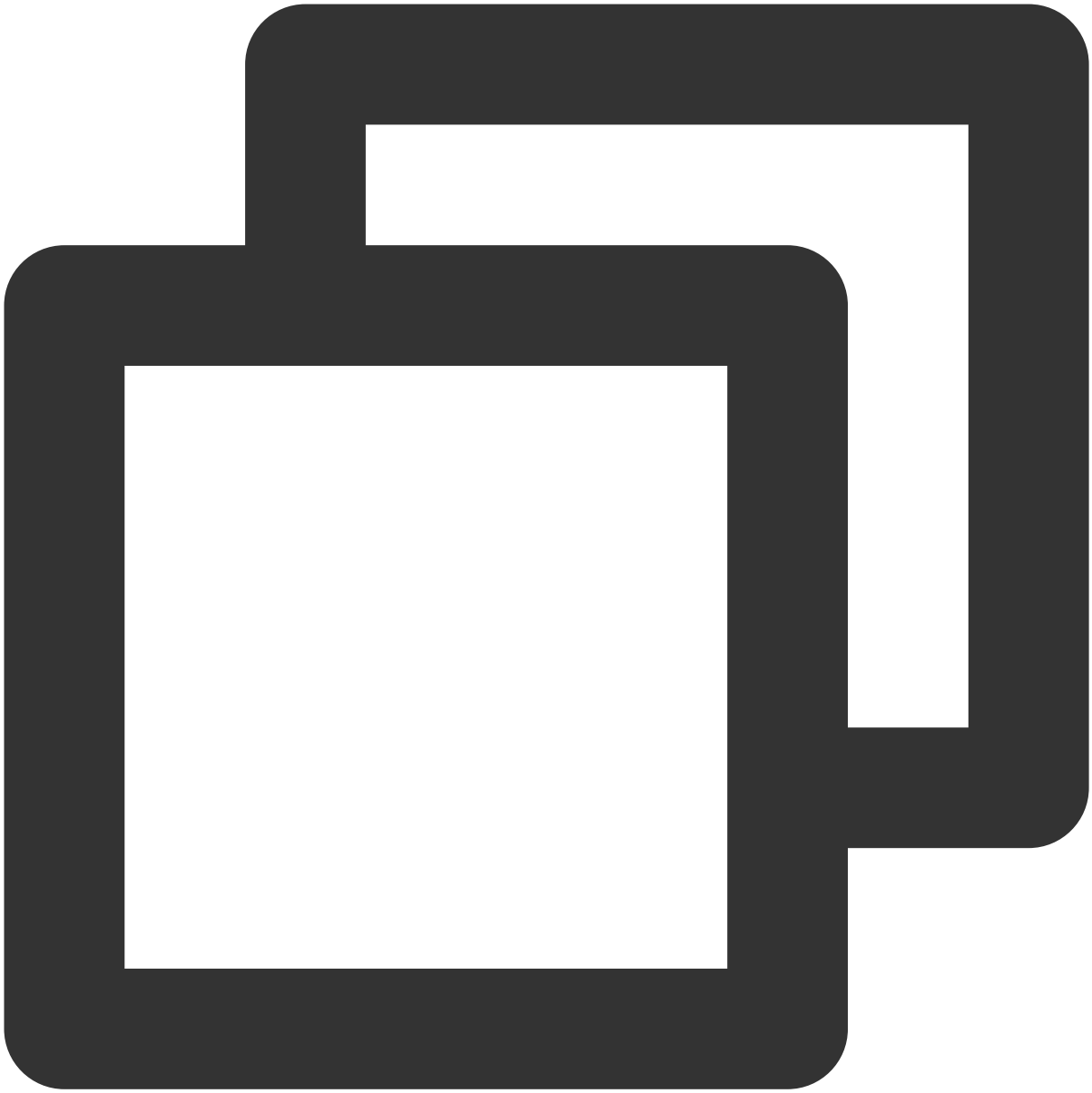
```
void addObserver(TUIRoomObserver observer)
```

参数：

参数	类型	含义
observer	TUIRoomObserver	TUIRoomEngine 事件回调

removeObserver

移除 TUIRoomEngine 事件回调。

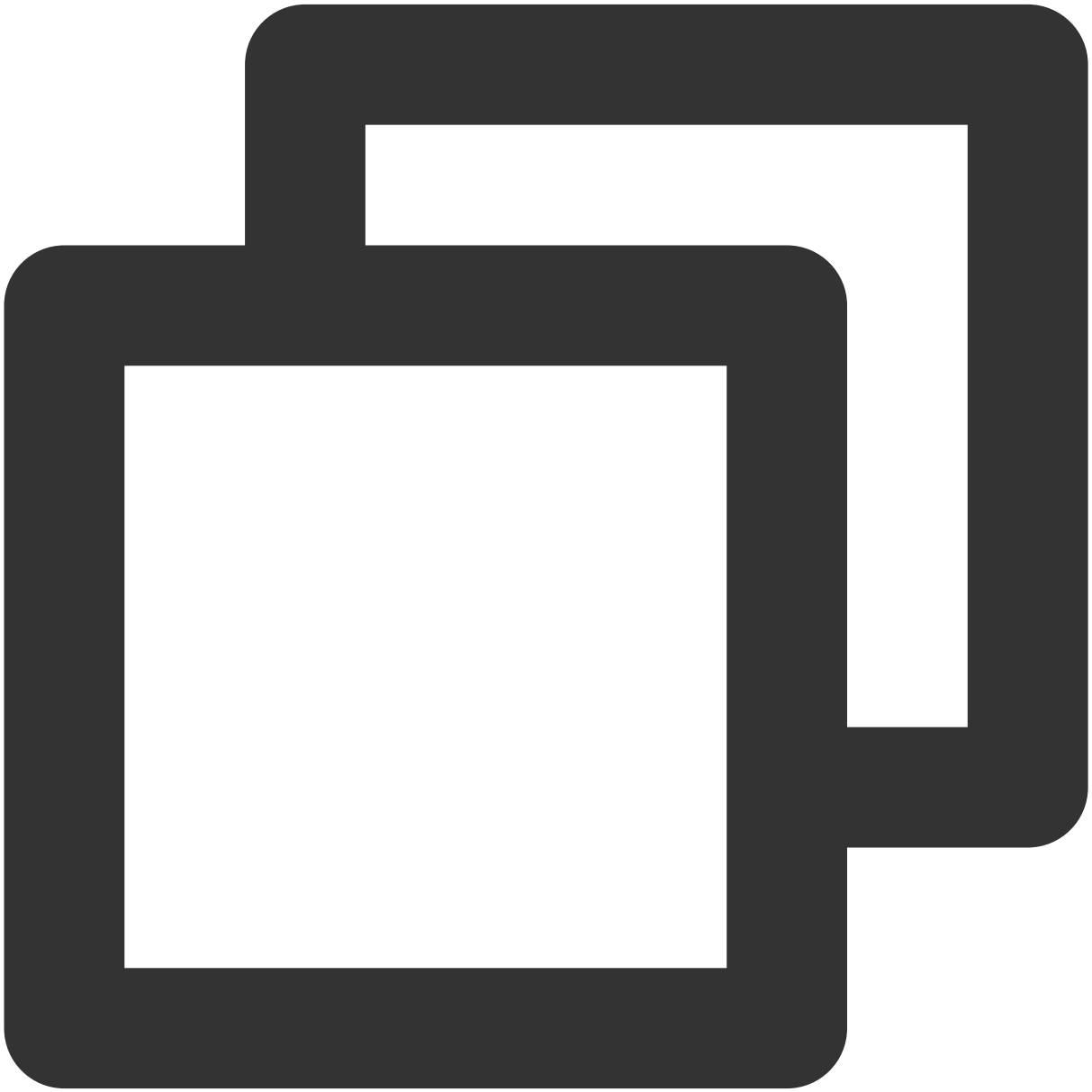


```
void removeObserver(TUIRoomObserver observer)
```

参数	类型	含义
observer	TUIRoomObserver	TUIRoomEngine 事件回调

createRoom

主持人创建房间，调用 createRoom 的用户为房间的所有者。创建房间时可设置房间 ID、房间名称、以及房间是否允许加入用户开启音视频、发送消息等功能。



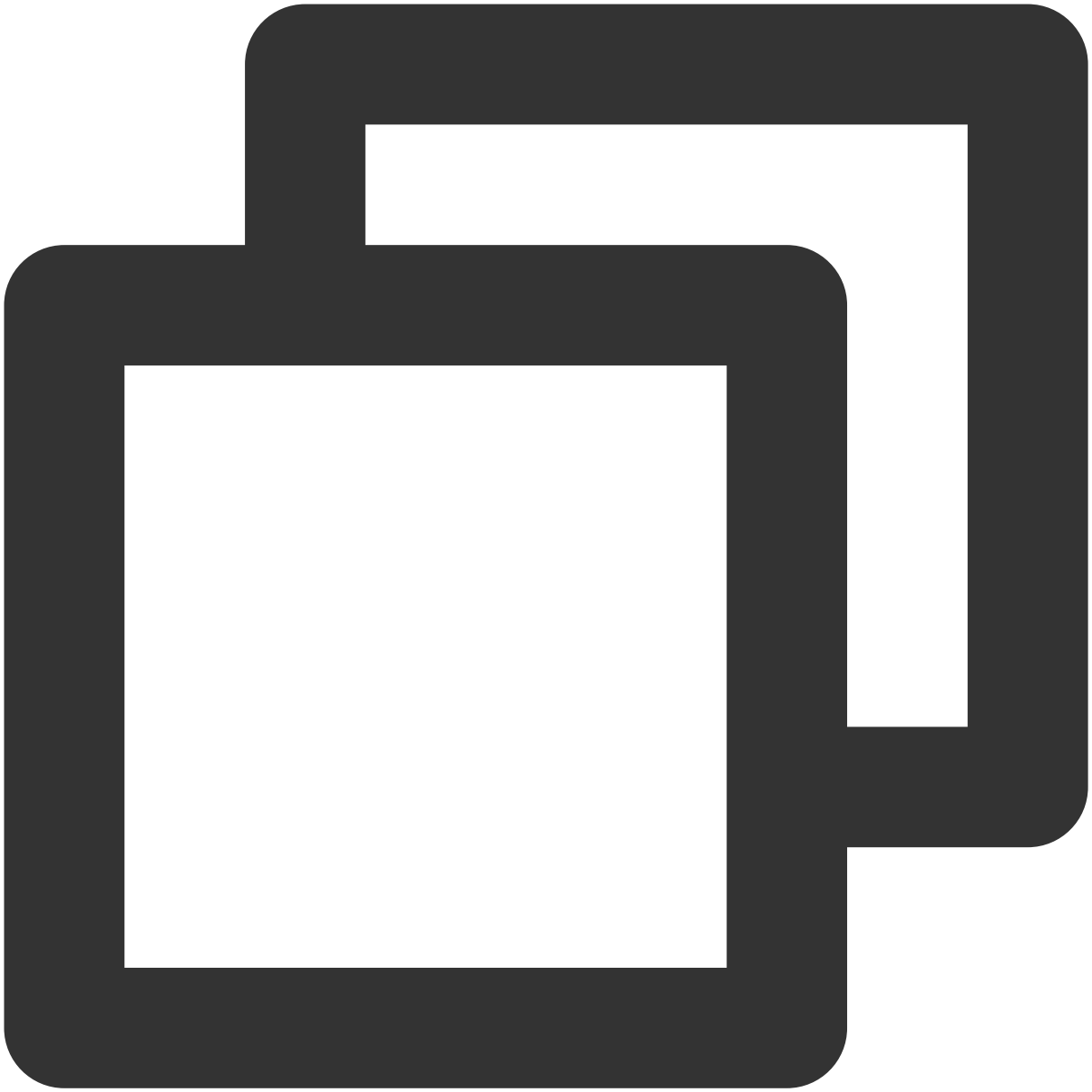
```
Future<TUIActionCallback> createRoom(TUIRoomInfo roomInfo)
```

参数：

参数	类型	含义
roomInfo	TUIRoomInfo	房间基本信息

destroyRoom

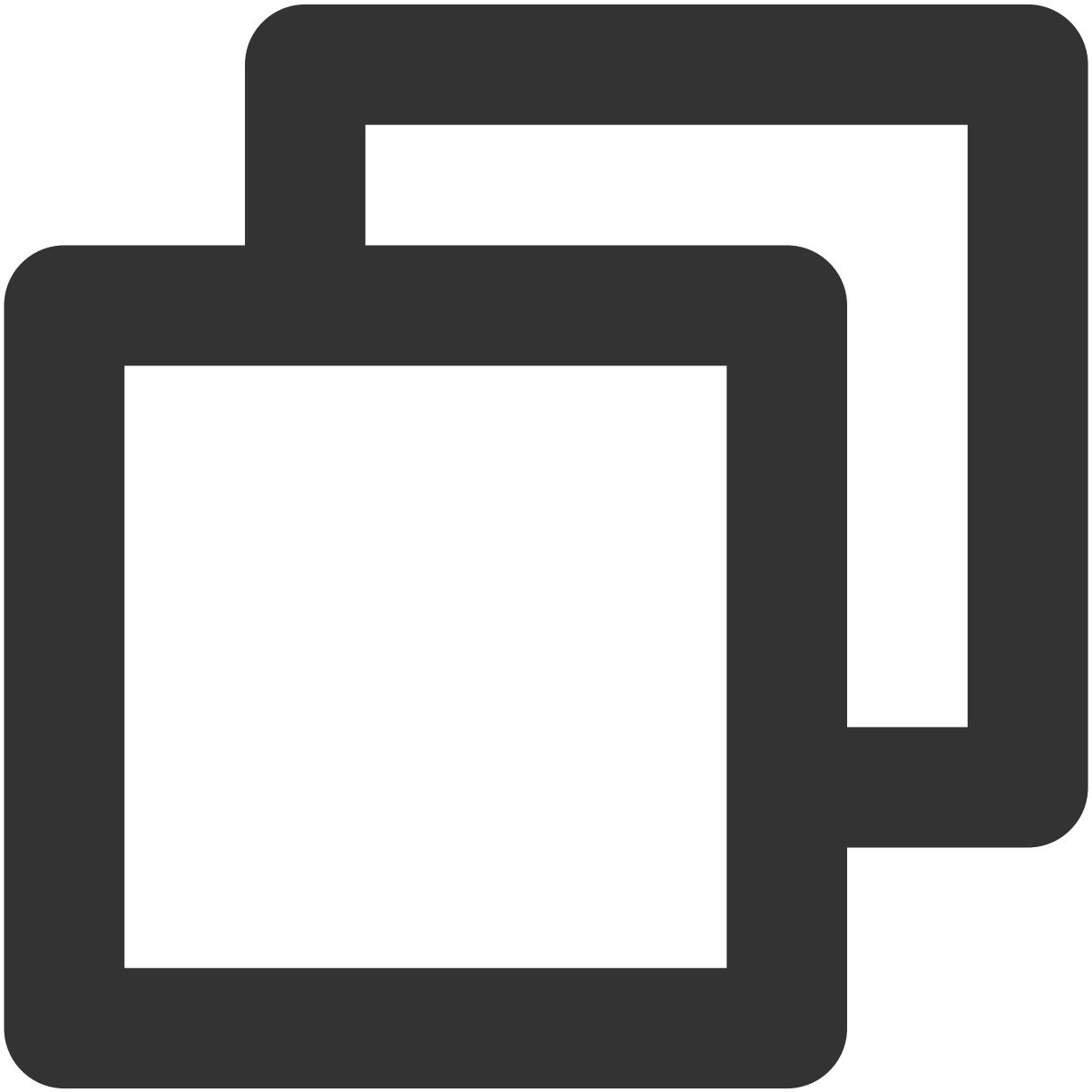
销毁房间接口，销毁房间必须由房间所有者发起，销毁房间之后房间不可进入。



```
Future<TUIActionCallback> destroyRoom()
```

enterRoom

进入房间接口。



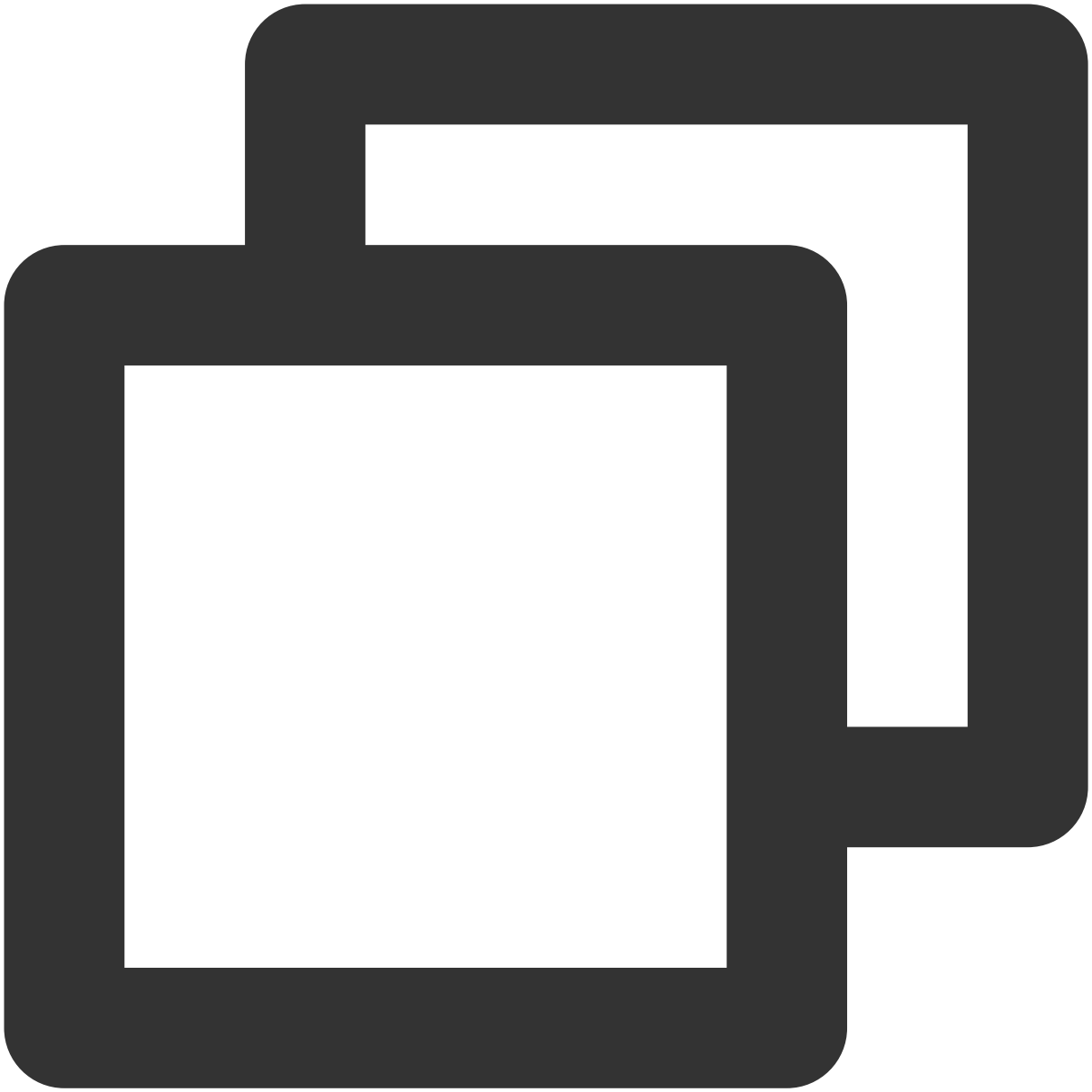
```
Future<TUIValueCallBack<TUIRoomInfo>> enterRoom(String roomId)
```

参数：

参数	类型	含义
roomId	String	房间号，字符串类型

exitRoom

离开房间接口，用户在执行 enterRoom 之后可通过 exitRoom 离开房间。



```
Future<TUIActionCallback> exitRoom(bool syncWaiting)
```

参数：

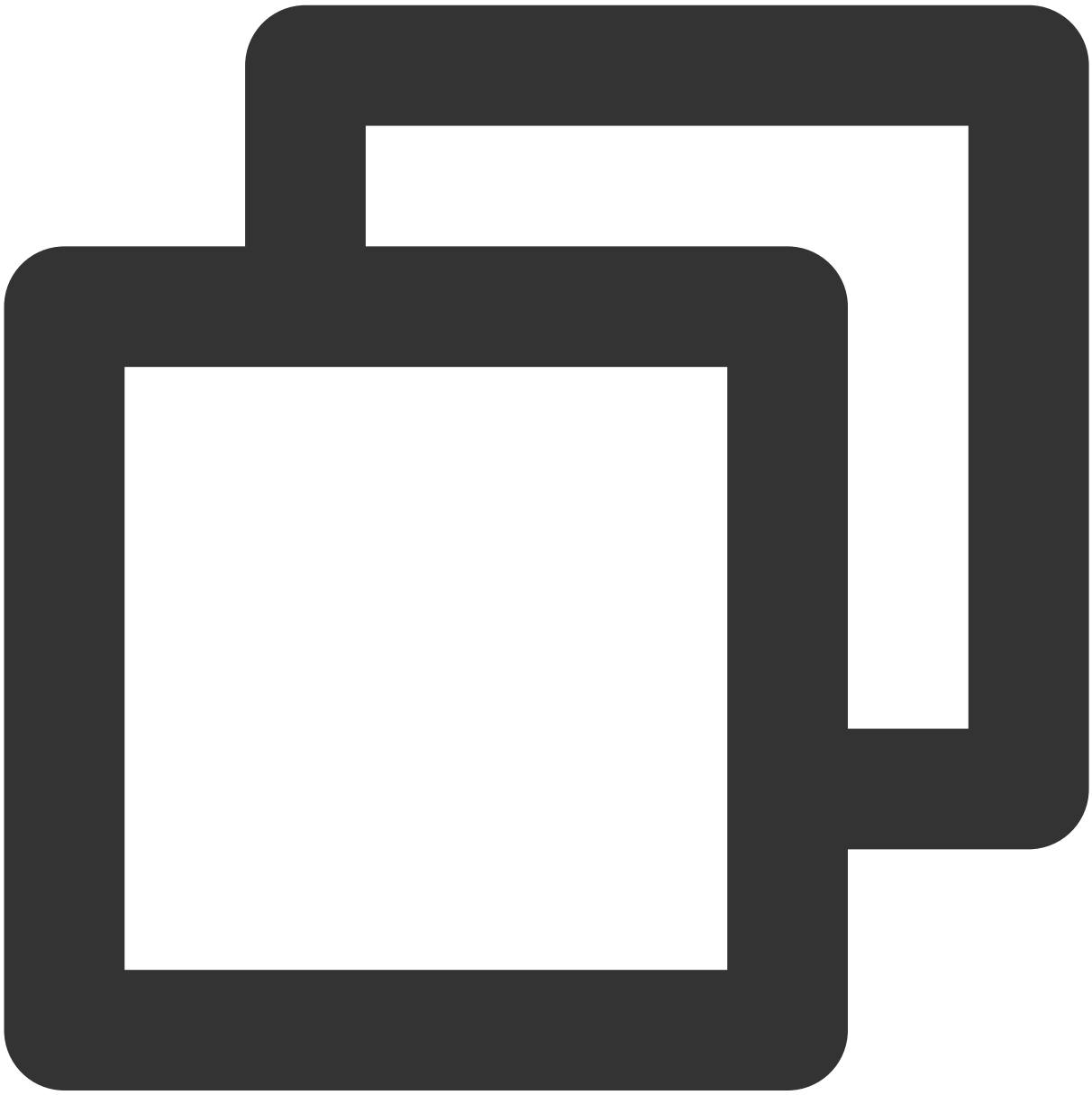
参数	类型	含义
syncWaiting	bool	是否同步退出房间

connectOtherRoom

连接其他房间。

说明：

用于直播场景下的申请跨房连麦。



```
TUIRequest connectOtherRoom(String roomId,
                            String userId,
                            int timeout,
                            TUIRequestCallback? requestCallback)
```

参数：

参数	类型	含义

roomId	String	房间 ID
userId	String	用户 ID
timeout	int	时间
callback	TUIRequestCallback	连接其他房间请求回调

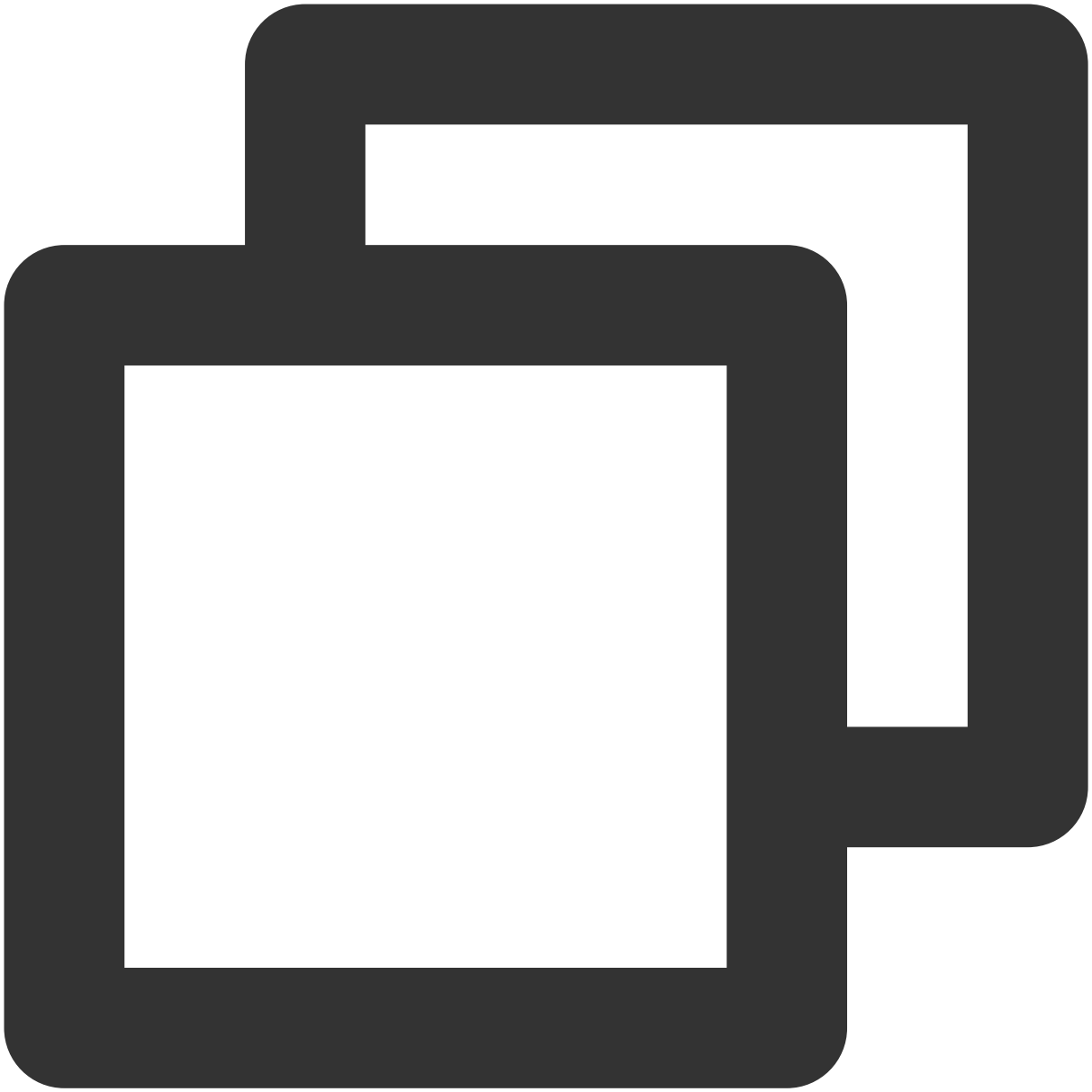
Return：请求体

disconnectOtherRoom

断开与其他房间的连接。

说明：

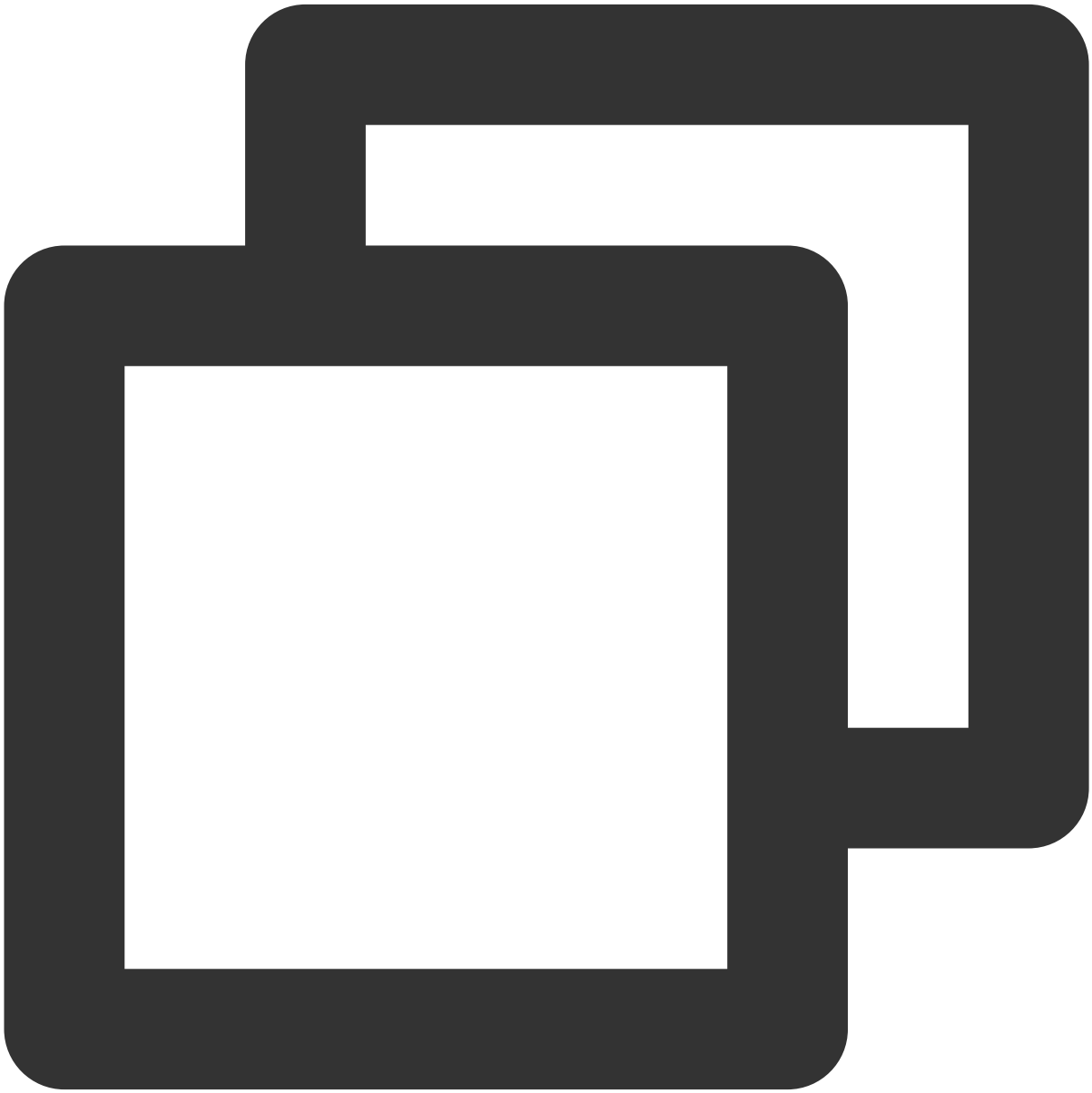
用于直播场景下的断开跨房连麦。



```
Future<TUIActionCallback> disconnectOtherRoom()
```

fetchRoomInfo

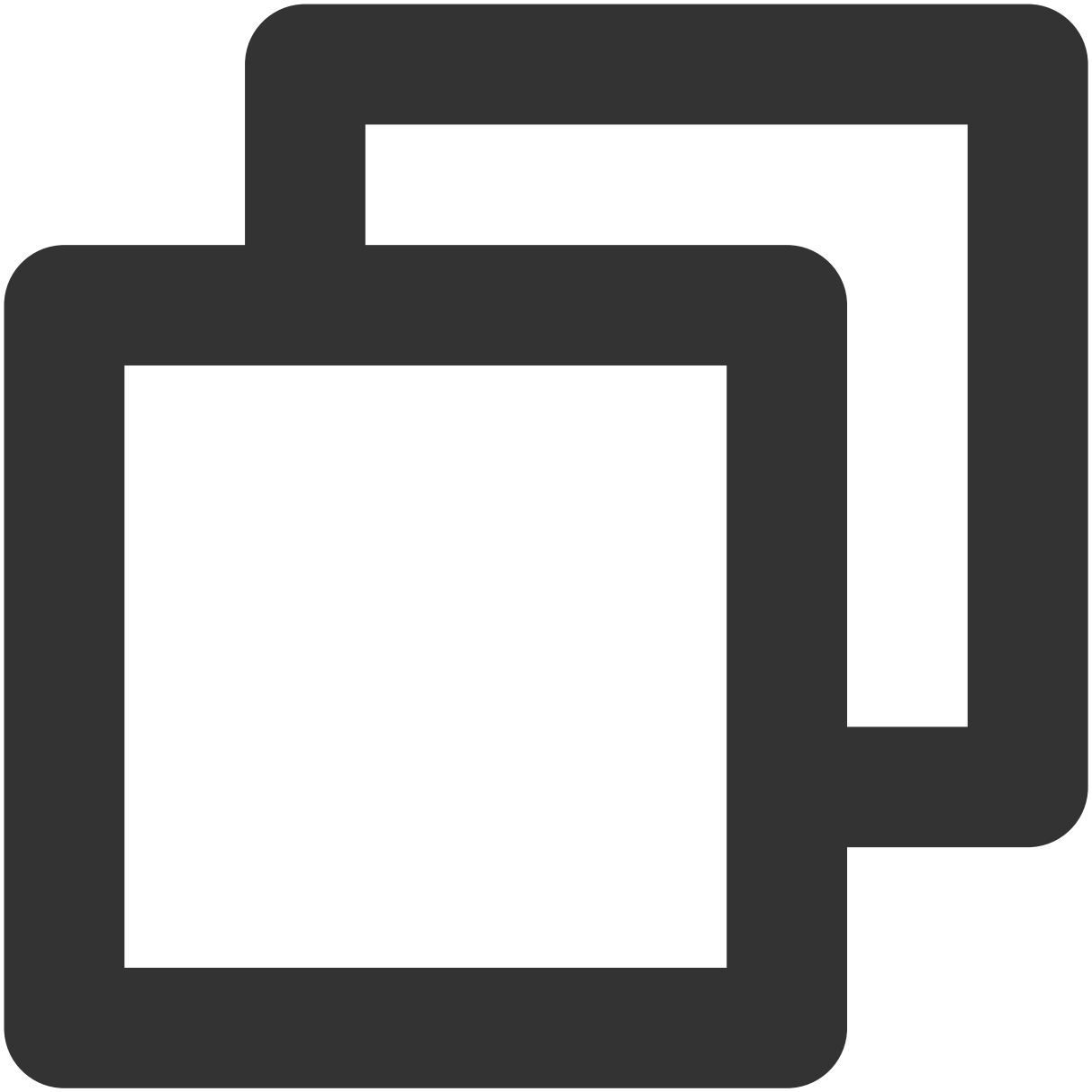
获取房间信息。



```
Future<TUIValueCallBack<TUIRoomInfo>> fetchRoomInfo()
```

updateRoomNameByAdmin

更新房间名称。



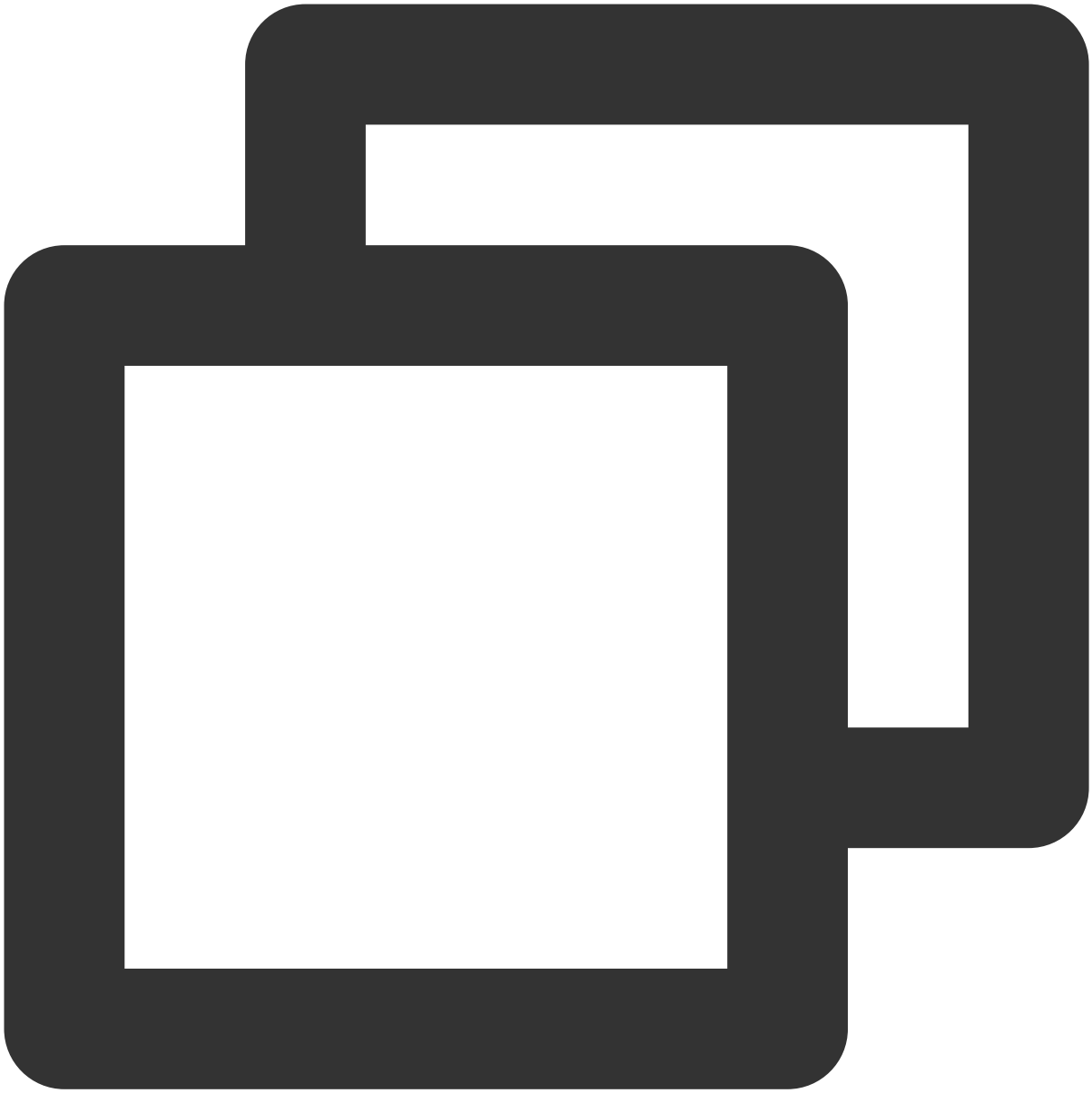
```
Future<TUIActionCallback> updateRoomNameByAdmin(String roomName)
```

参数：

参数	类型	含义
roomName	String	房间名称

updateRoomSpeechModeByAdmin

设置房间管理模式（只有管理员或群主能够调用）。

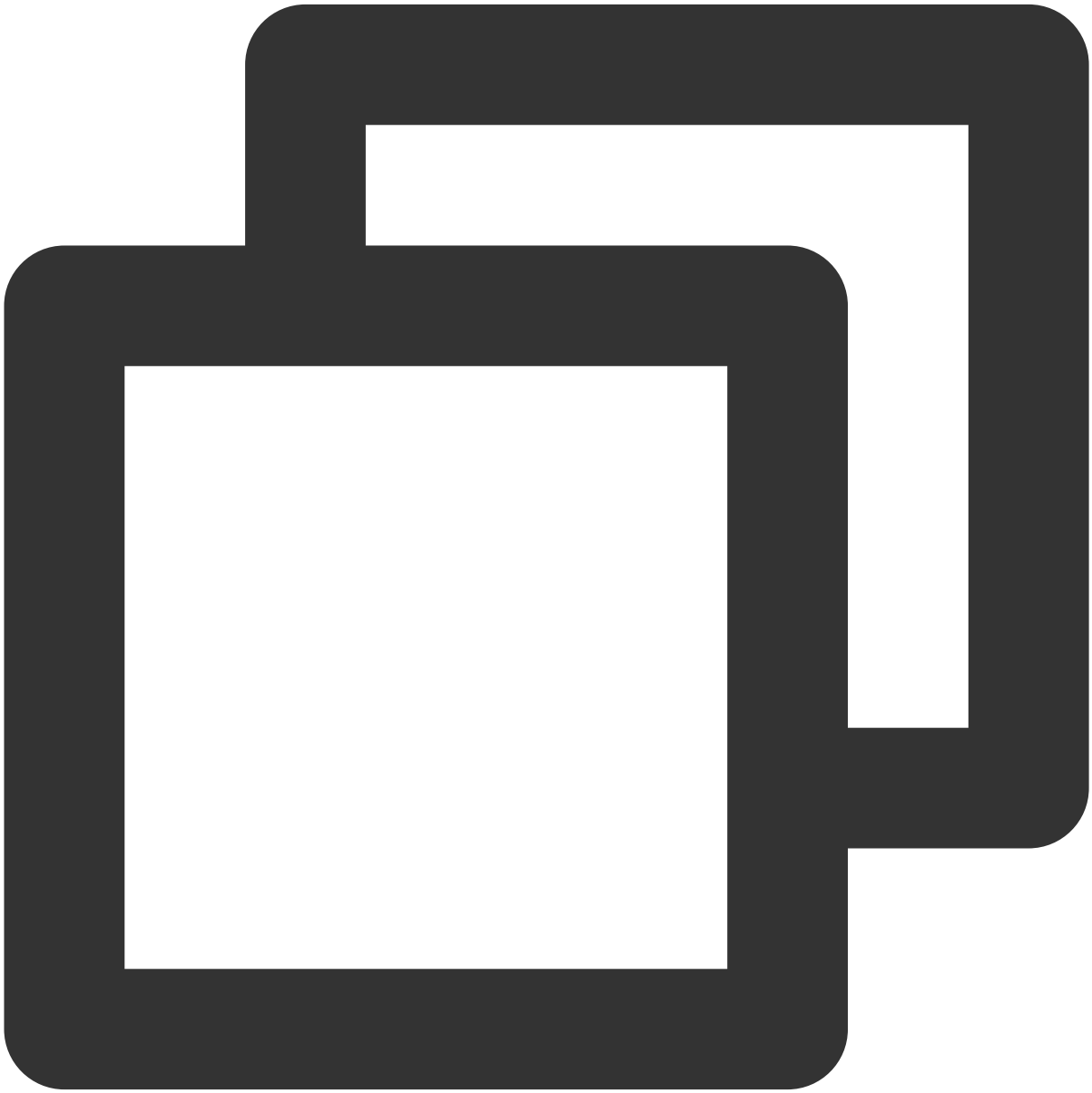


```
Future<TUIActionCallback> updateRoomSpeechModeByAdmin(TUISpeechMode mode)
```

参数	类型	含义
mode	TUISpeechMode	房间模式

setLocalVideoView

设置本地用户视频渲染的视图控件。



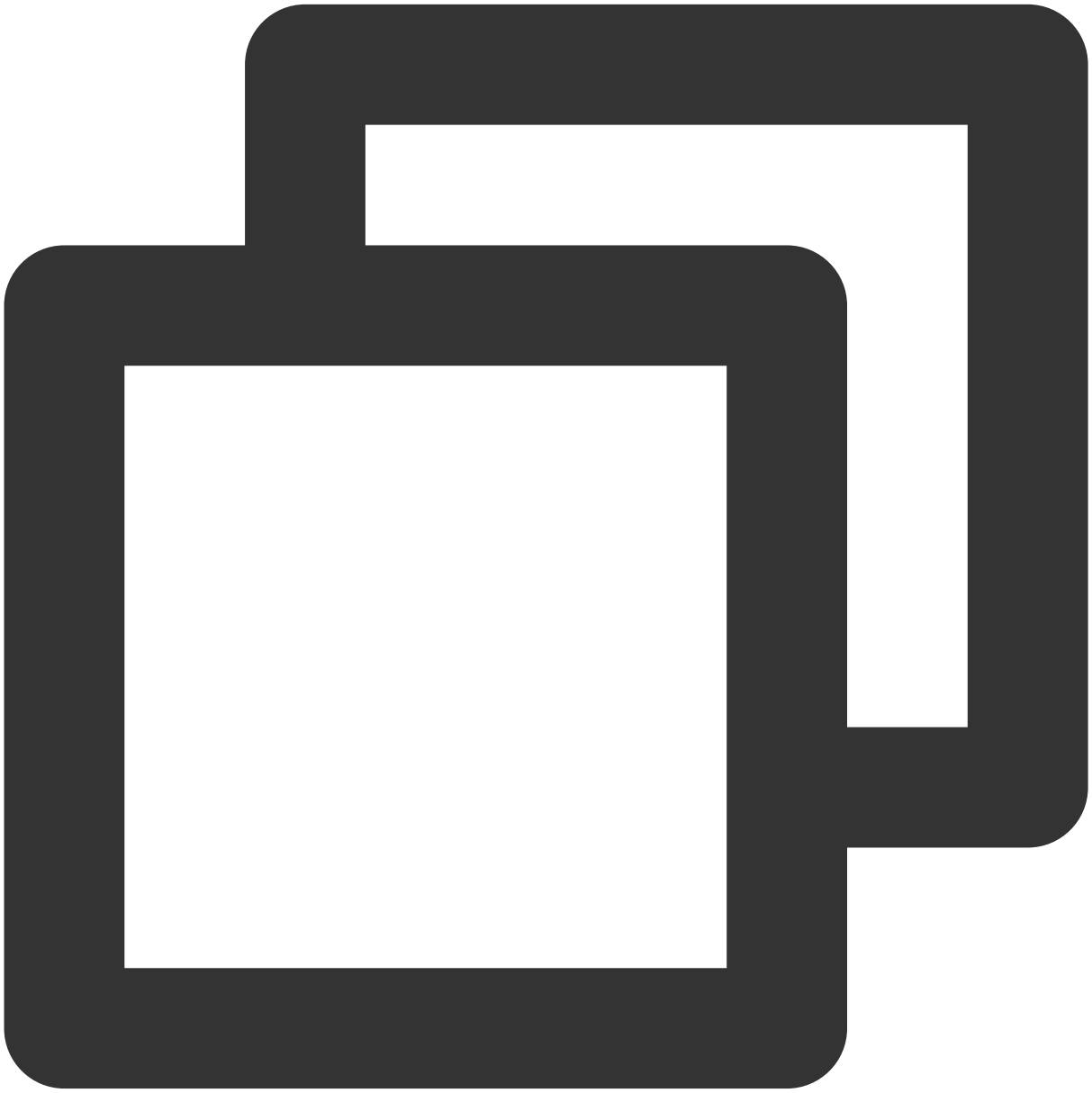
```
void setLocalVideoView(int viewId)
```

参数：

参数	类型	含义
viewId	int	待渲染 view 指针的 int64 类型值，通过此 viewId 可转换为对应原生平台的视图，视频画面将渲染于此视图上

openLocalCamera

打开本地摄像头，开始视频流采集。



```
Future<TUIActionCallback> openLocalCamera(bool isFront,
                                          TUIVideoQuality quality)
```

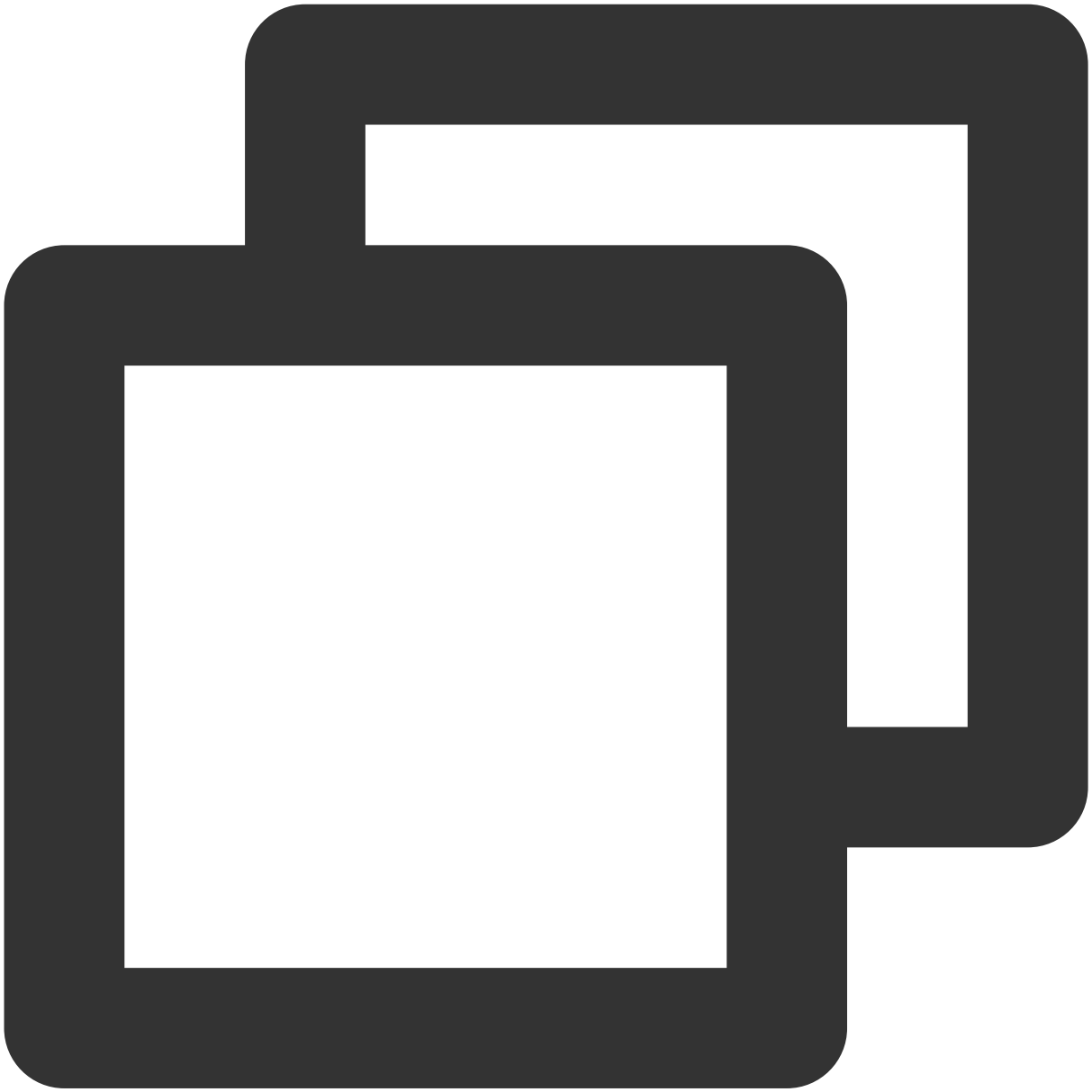
参数：

参数	类型	含义
isFront	bool	是否使用前置摄像头

quality	TUIVideoQuality	视频质量

closeLocalCamera

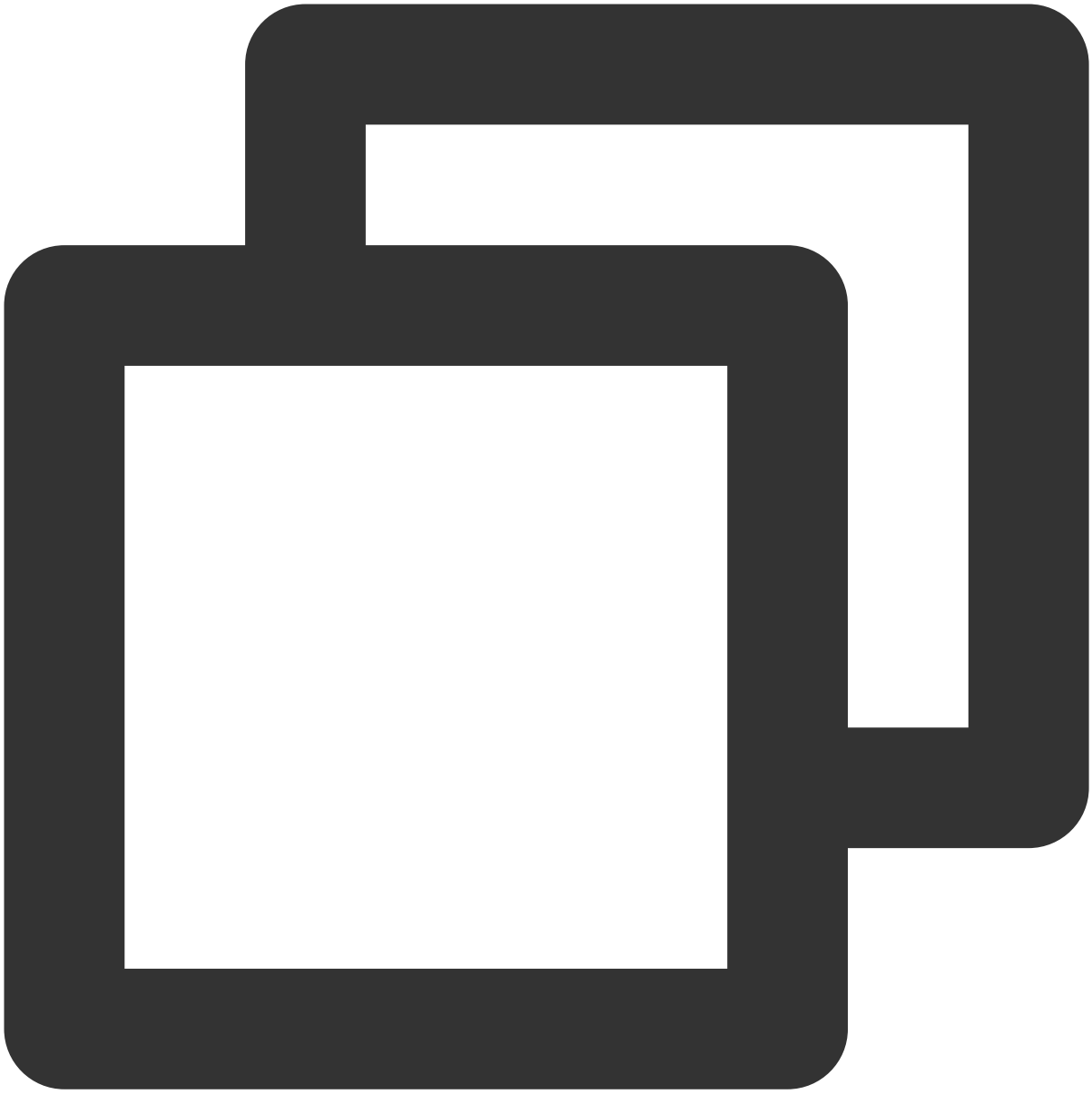
关闭本地摄像头。



```
void closeLocalCamera()
```

updateVideoQuality

设置本地视频的参数。



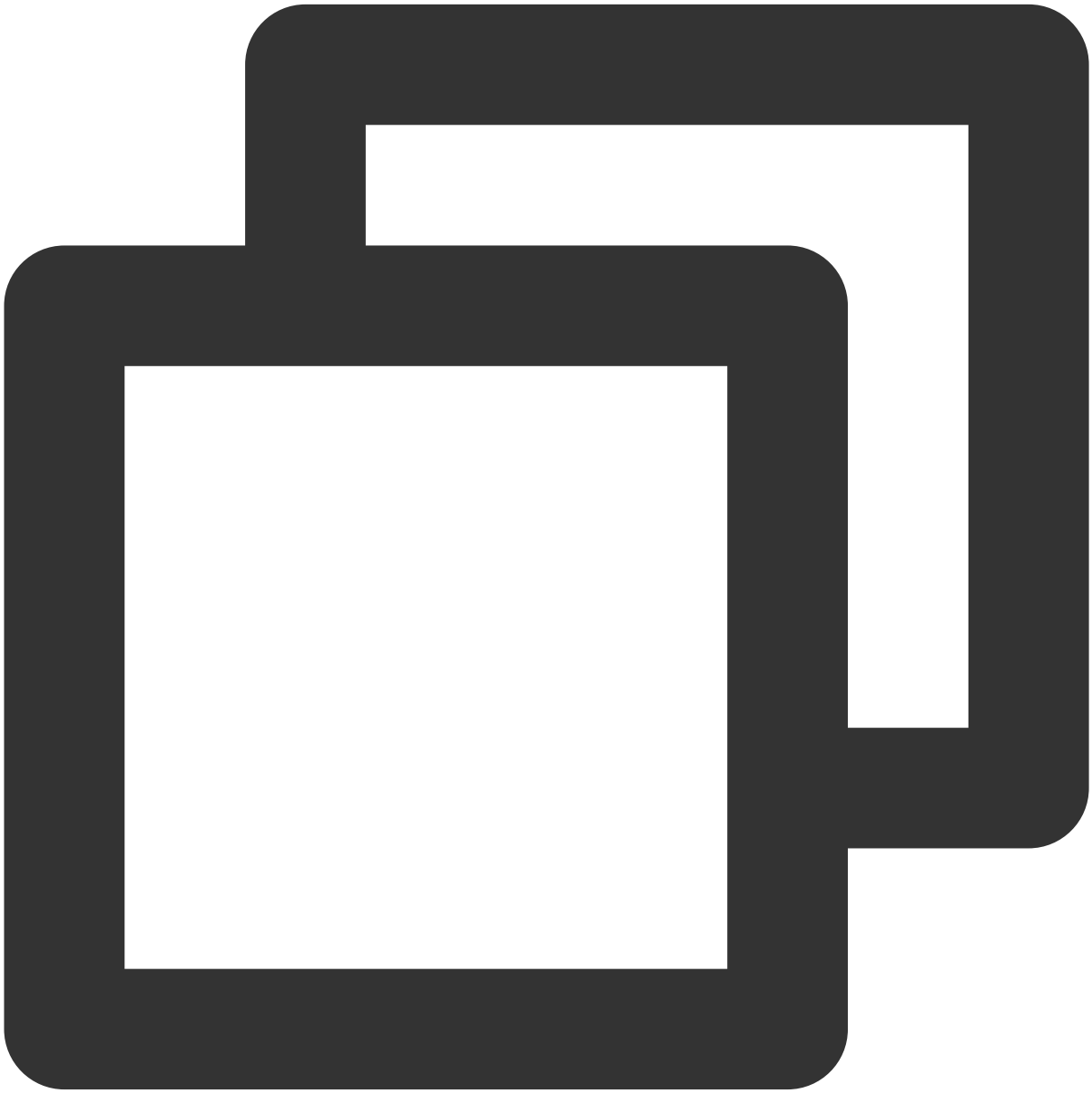
```
void updateVideoQuality(TUIVideoQuality quality)
```

参数：

参数	类型	含义
quality	TUIVideoQuality	视频质量

updateVideoQualityEx

设置本地视频编码器参数。

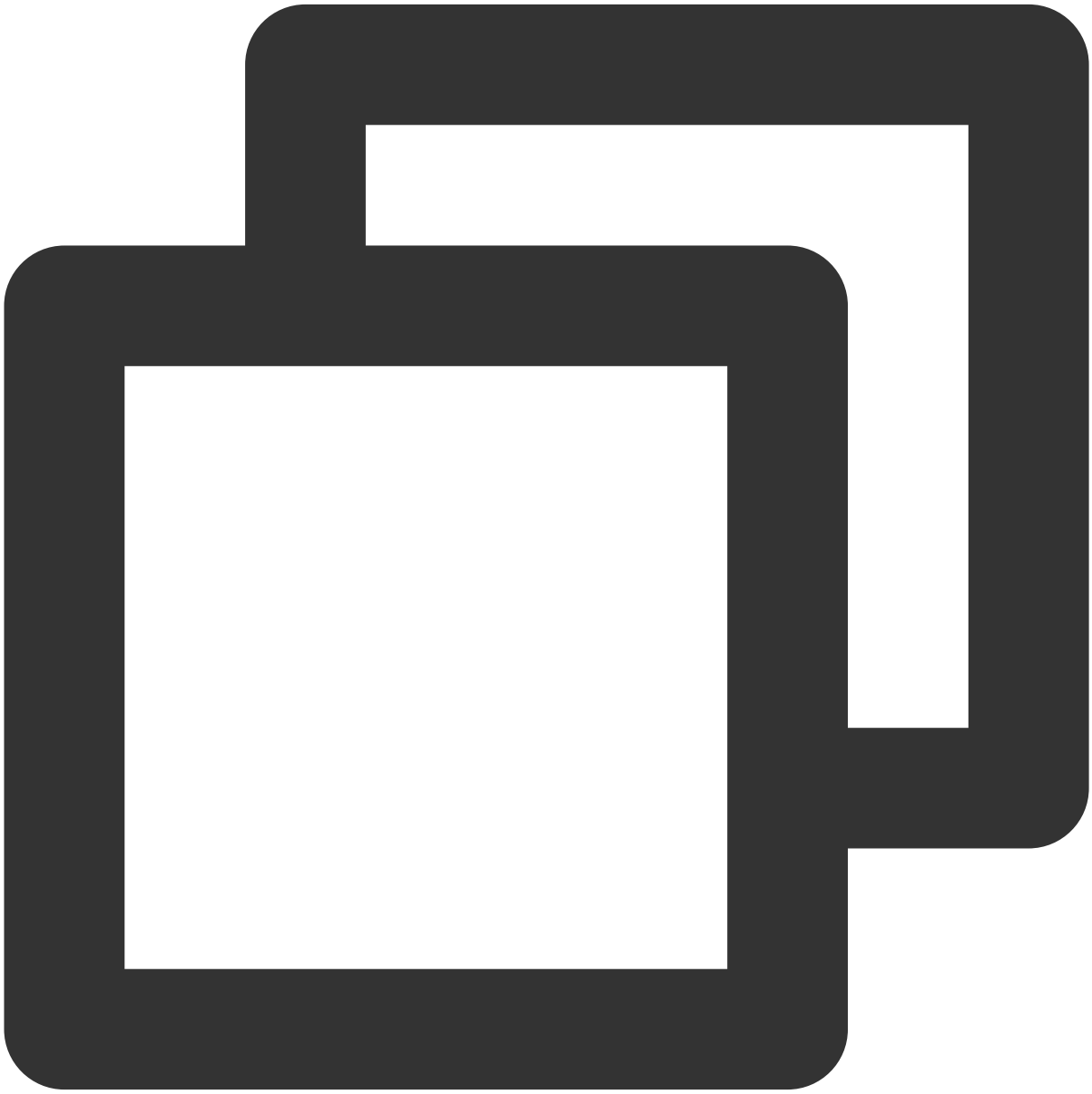


```
void updateVideoQualityEx(
      TUIVideoStreamType streamType, TUIRoomVideoEncoderParams params);
```

参数	类型	含义
streamType	TUIVideoStreamType	视频流类型
params	TUIRoomVideoEncoderParams	视频编码器参数

setVideoResolutionMode

设置视频编码器分辨率模式（横屏分辨率 or 竖屏分辨率）。



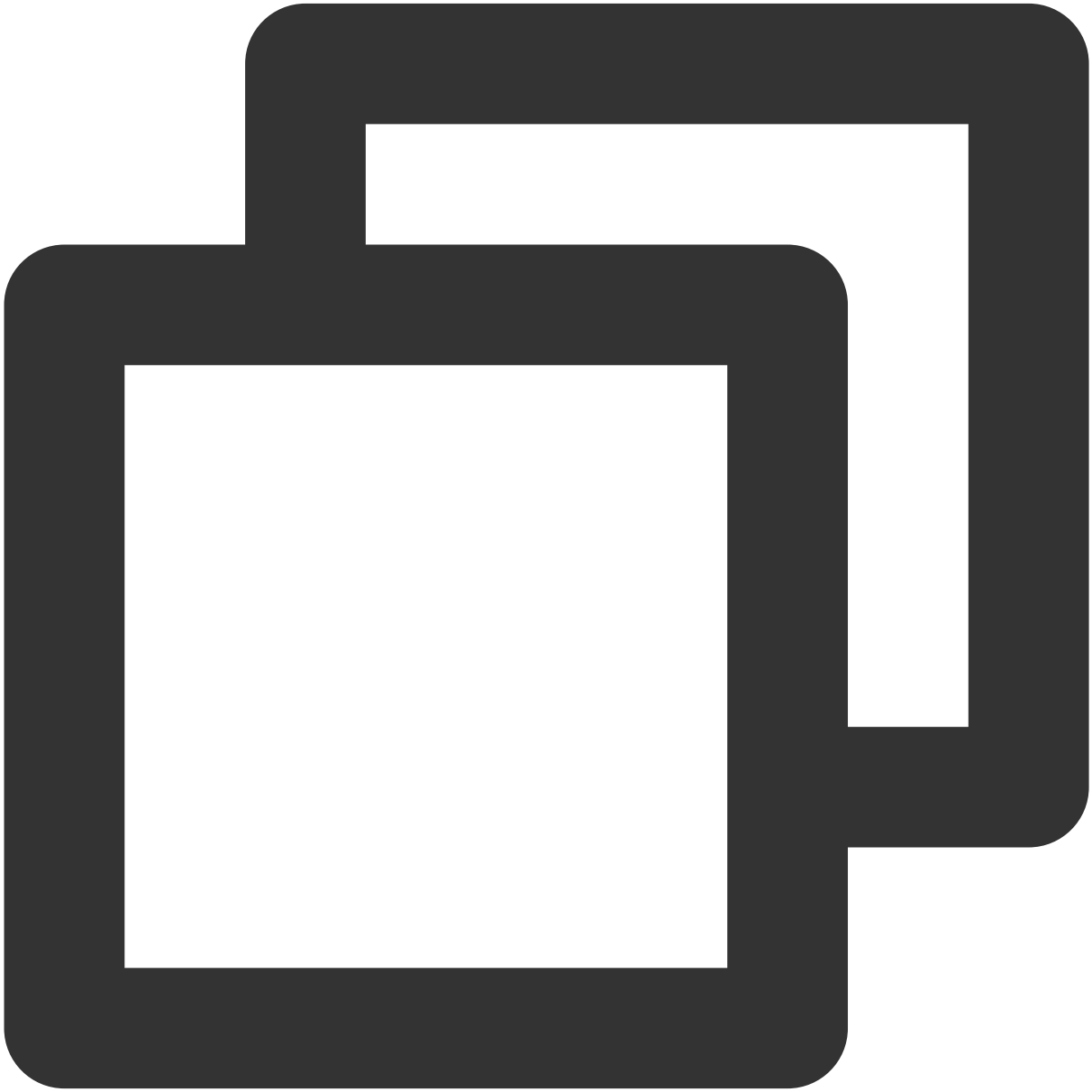
```
void setVideoResolutionMode(
      TUIVideoStreamType streamType, TUIResolutionMode resolutionMode);
```

参数	类型	含义
streamType	TUIVideoStreamType	视频流类型

resolutionMode	TUIResolutionMode	分辨率模式

enableGravitySensor

开启重力感应模式。

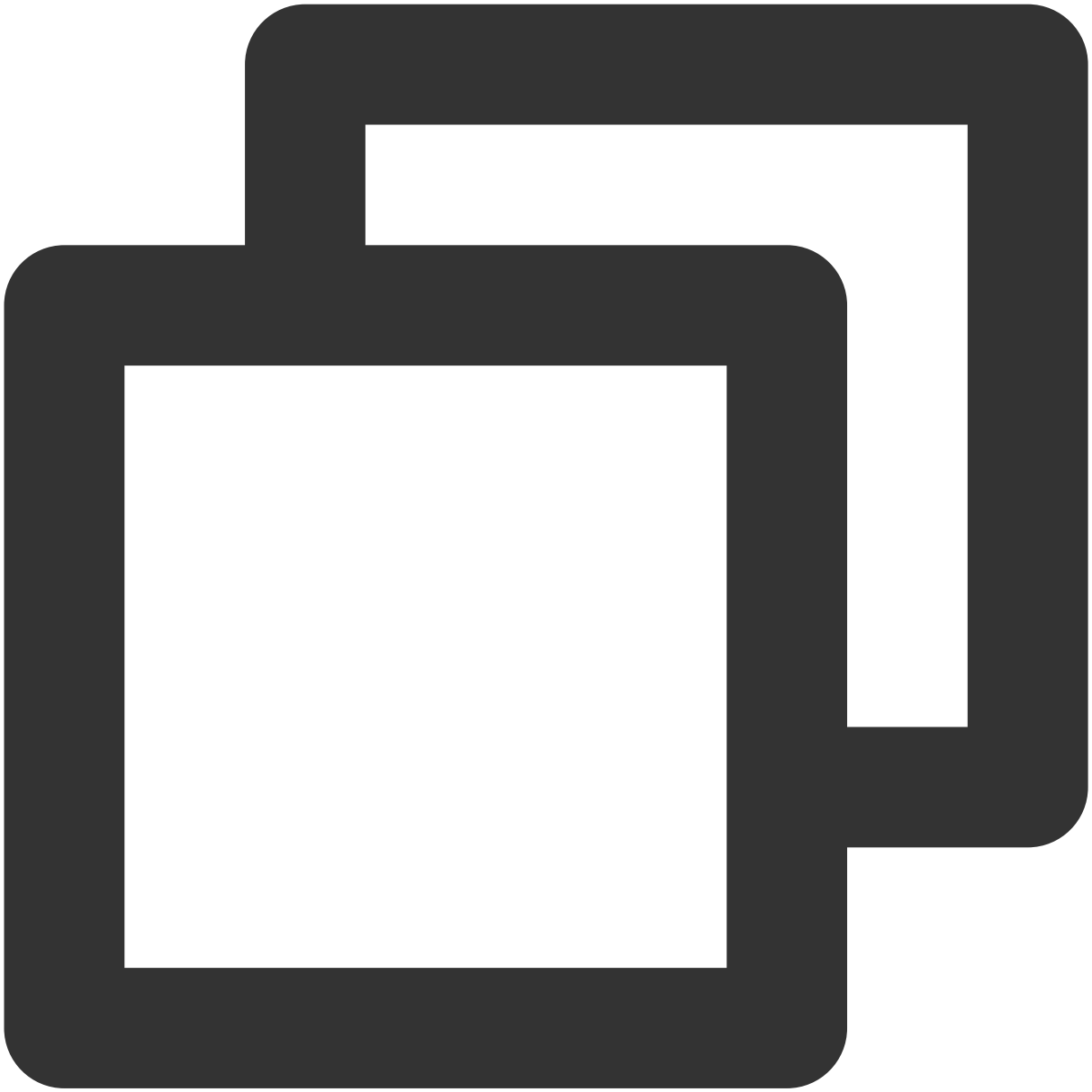


```
void enableGravitySensor(bool enable);
```

参数	类型	含义
enable	bool	是否开启

startPushLocalVideo

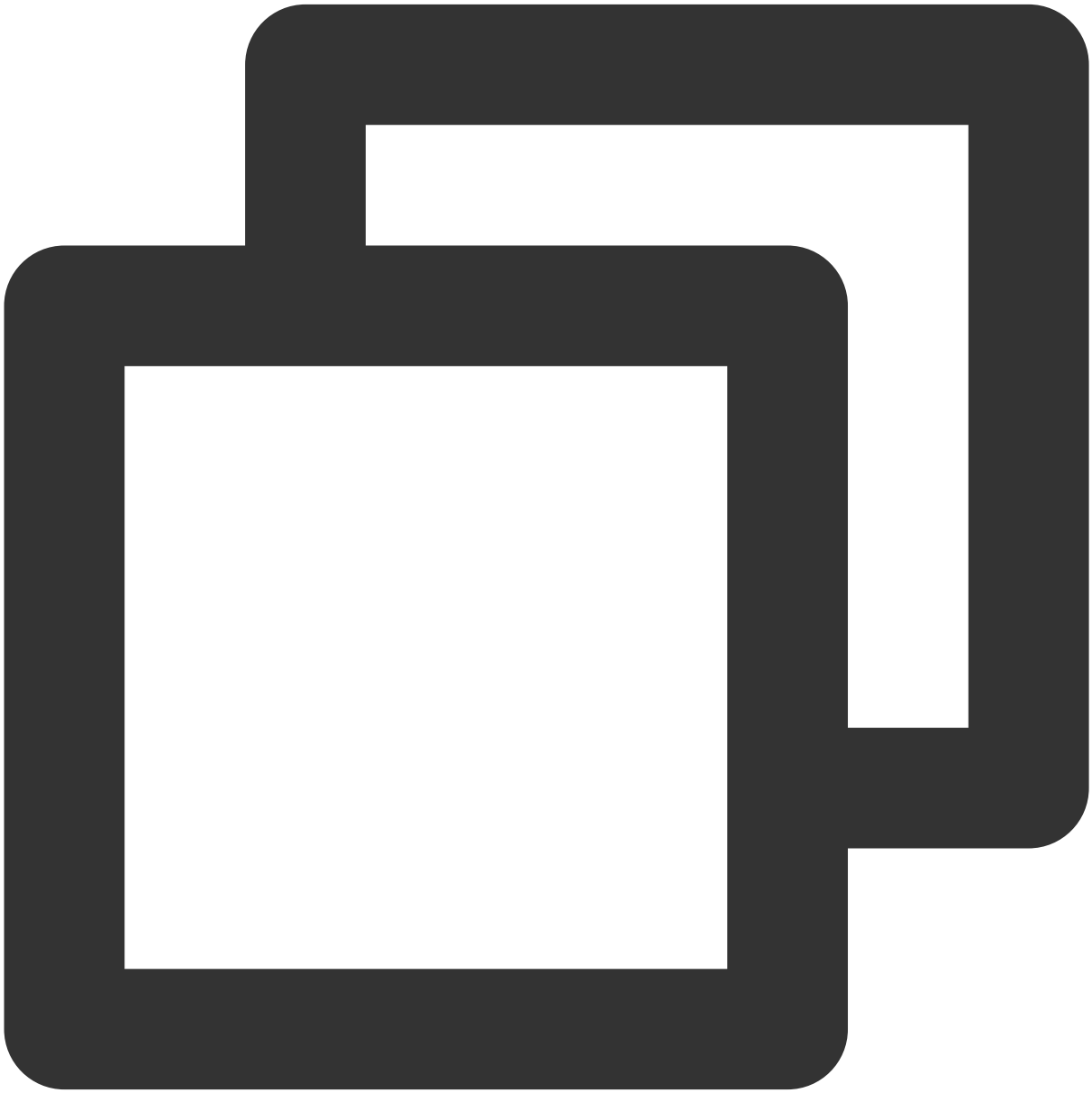
开始向远端推本地视频流。



```
void startPushLocalVideo()
```

stopPushLocalVideo

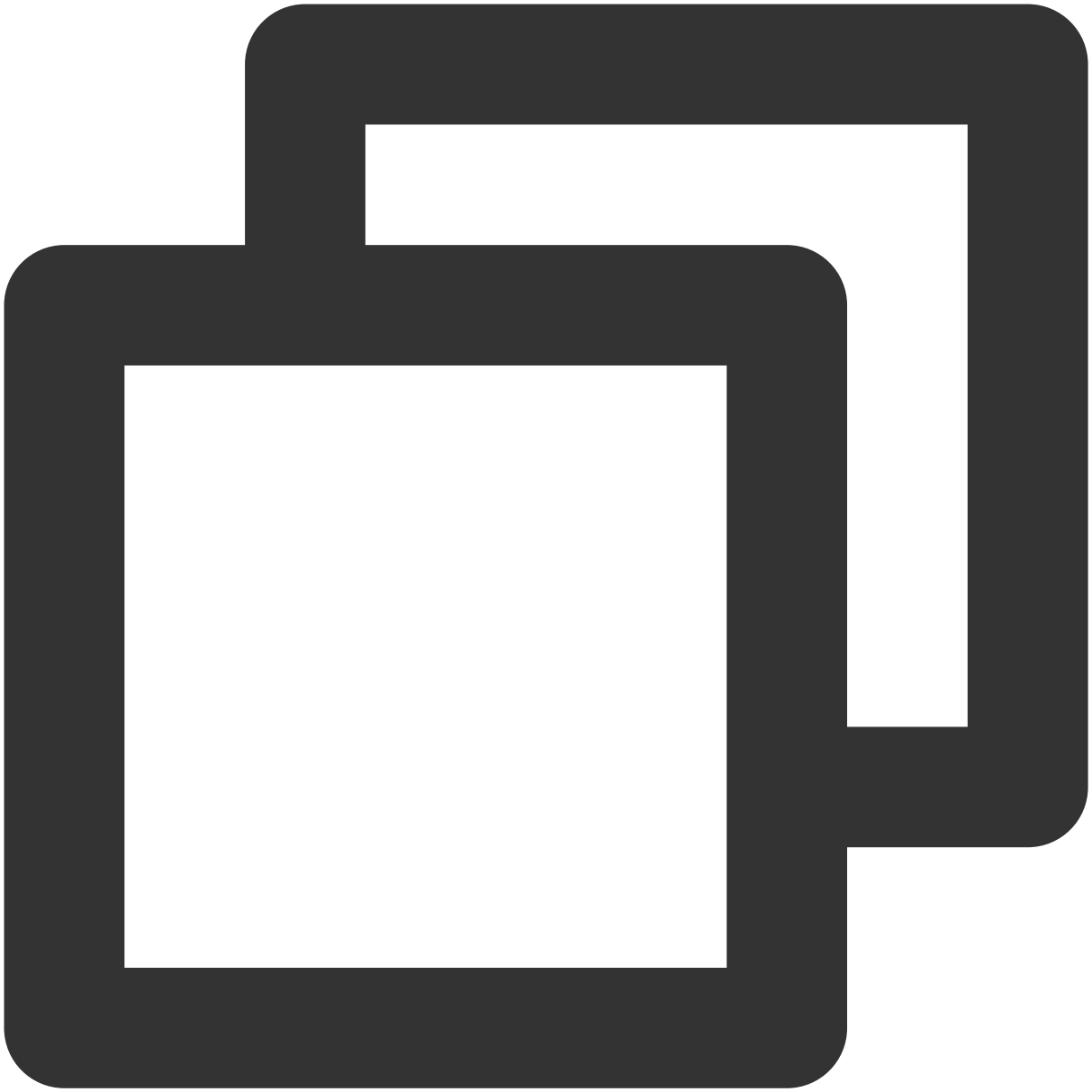
停止向远端推本地视频流。



```
void stopPushLocalVideo()
```

startScreenSharing

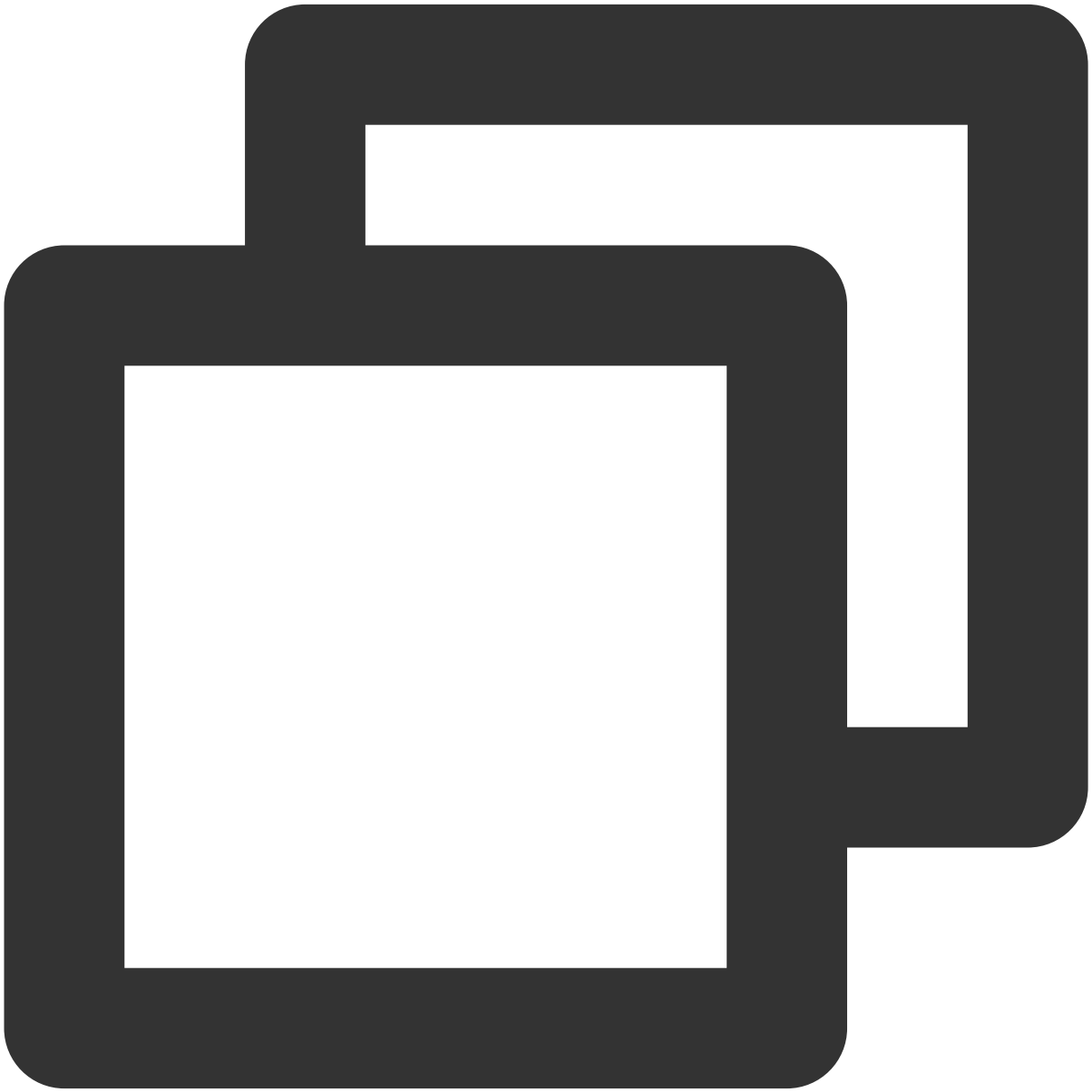
开始屏幕分享



```
Future<void> startScreenSharing({String appGroup = ''})
```

stopScreenSharing

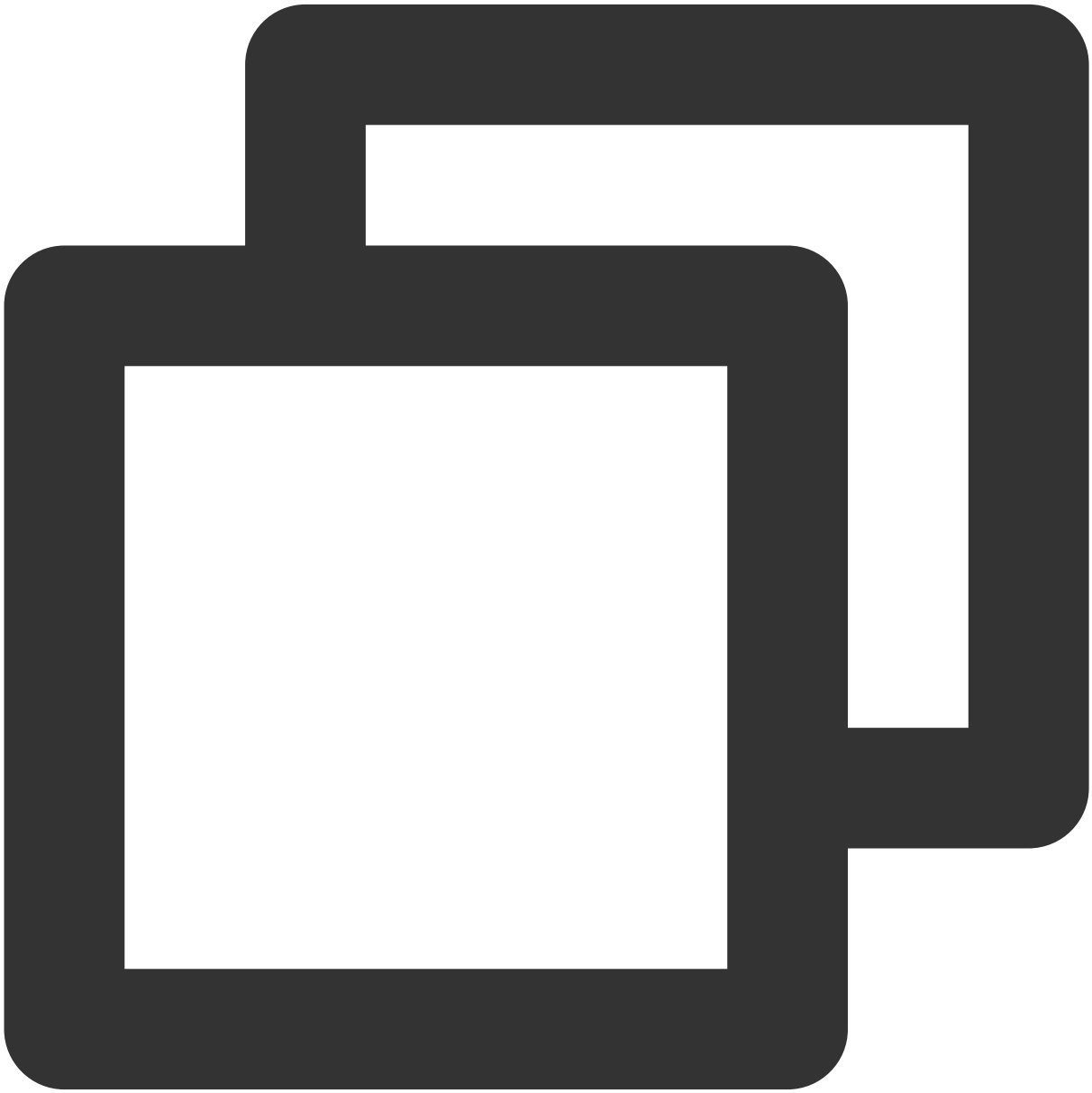
结束屏幕分享



```
Future<void> stopScreenSharing()
```

openLocalMicrophone

打开本地麦克风。

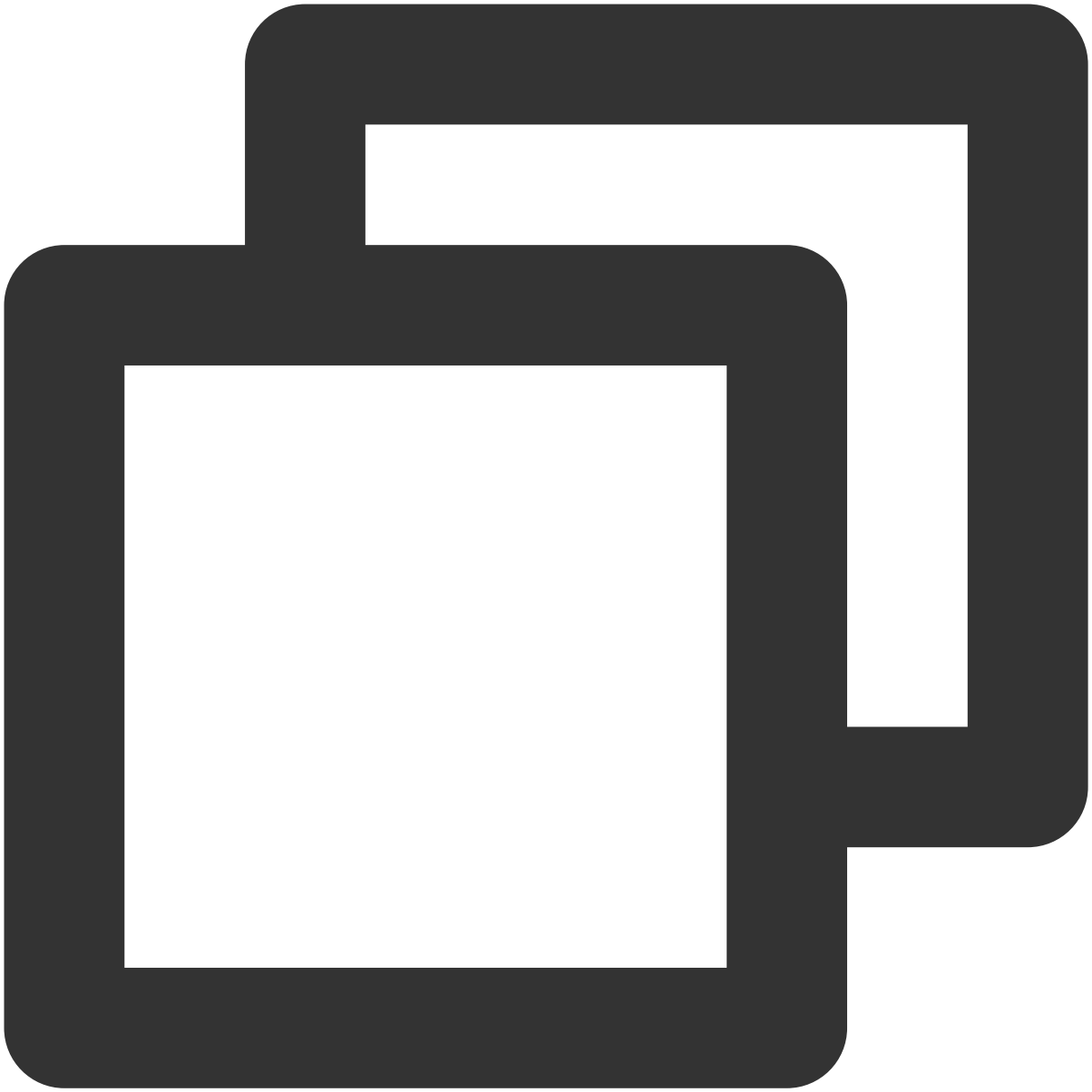


```
Future<TUIActionCallback> openLocalMicrophone(TUIAudioQuality quality)
```

参数	类型	含义
quality	TUIAudioQuality	音频质量

closeLocalMicrophone

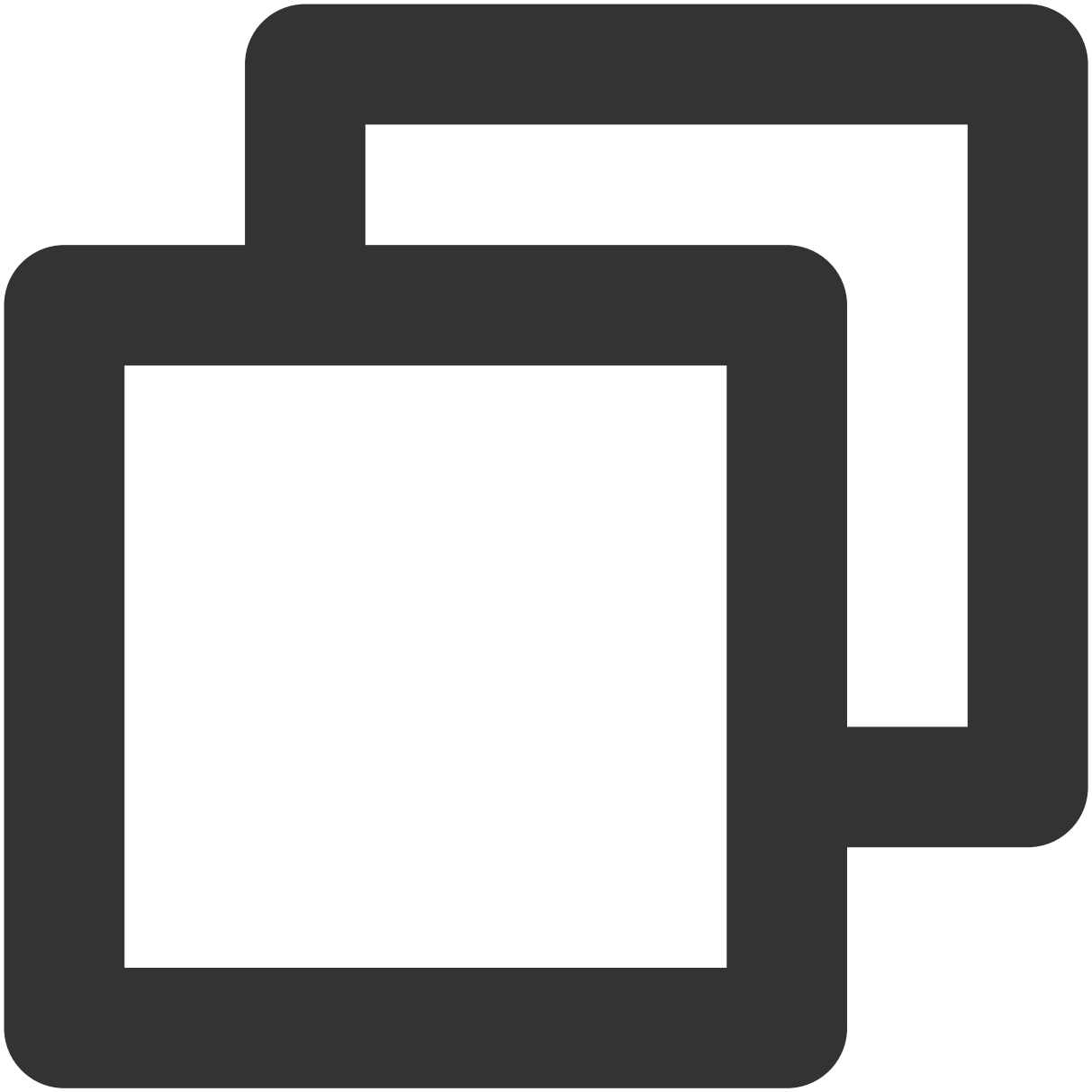
关闭本地麦克风。



```
void closeLocalMicrophone()
```

updateAudioQuality

更新本地音频编码质量设置。



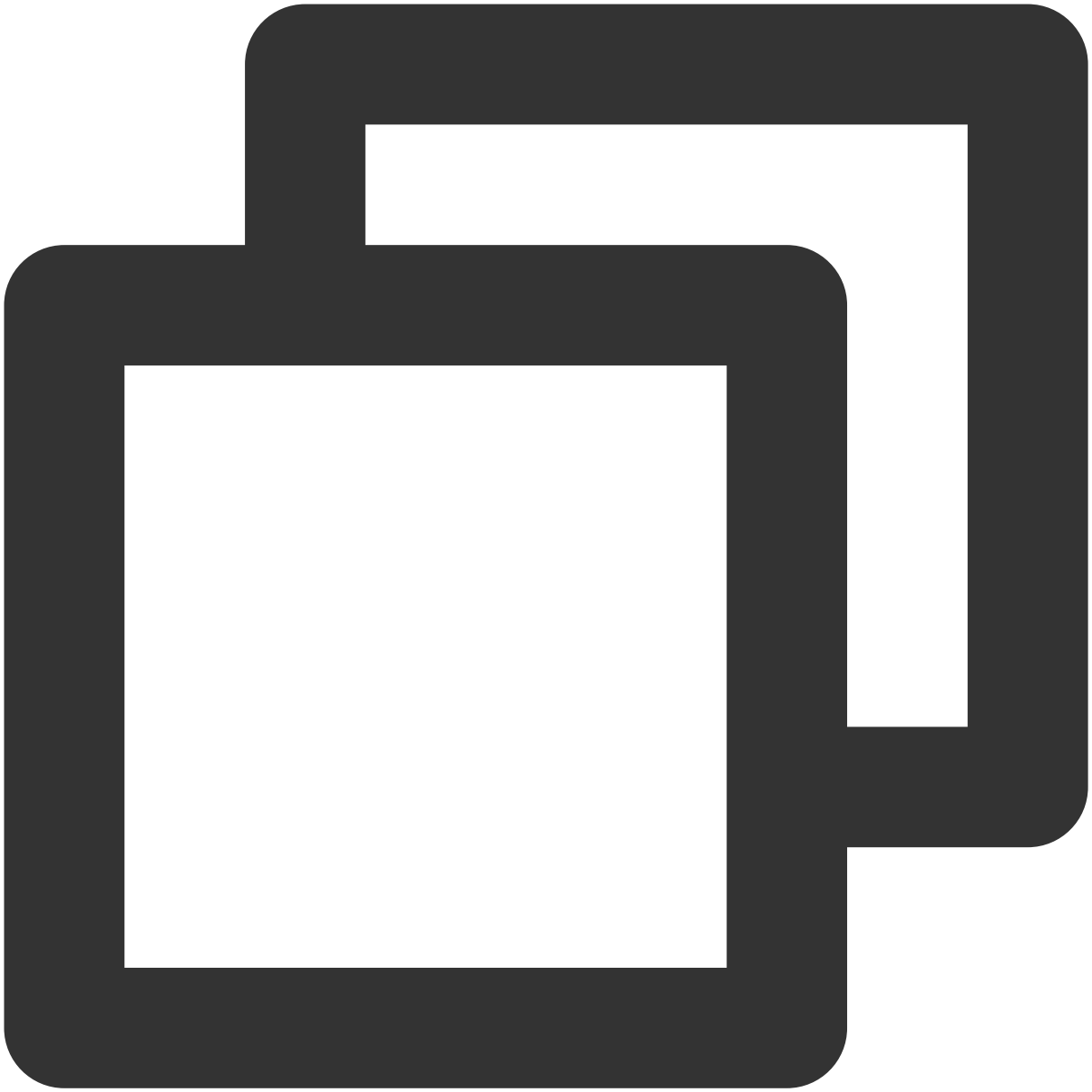
```
void updateAudioQuality(TUIAudioQuality quality)
```

参数：

参数	类型	含义
quality	TUIAudioQuality	音频质量

muteLocalAudio

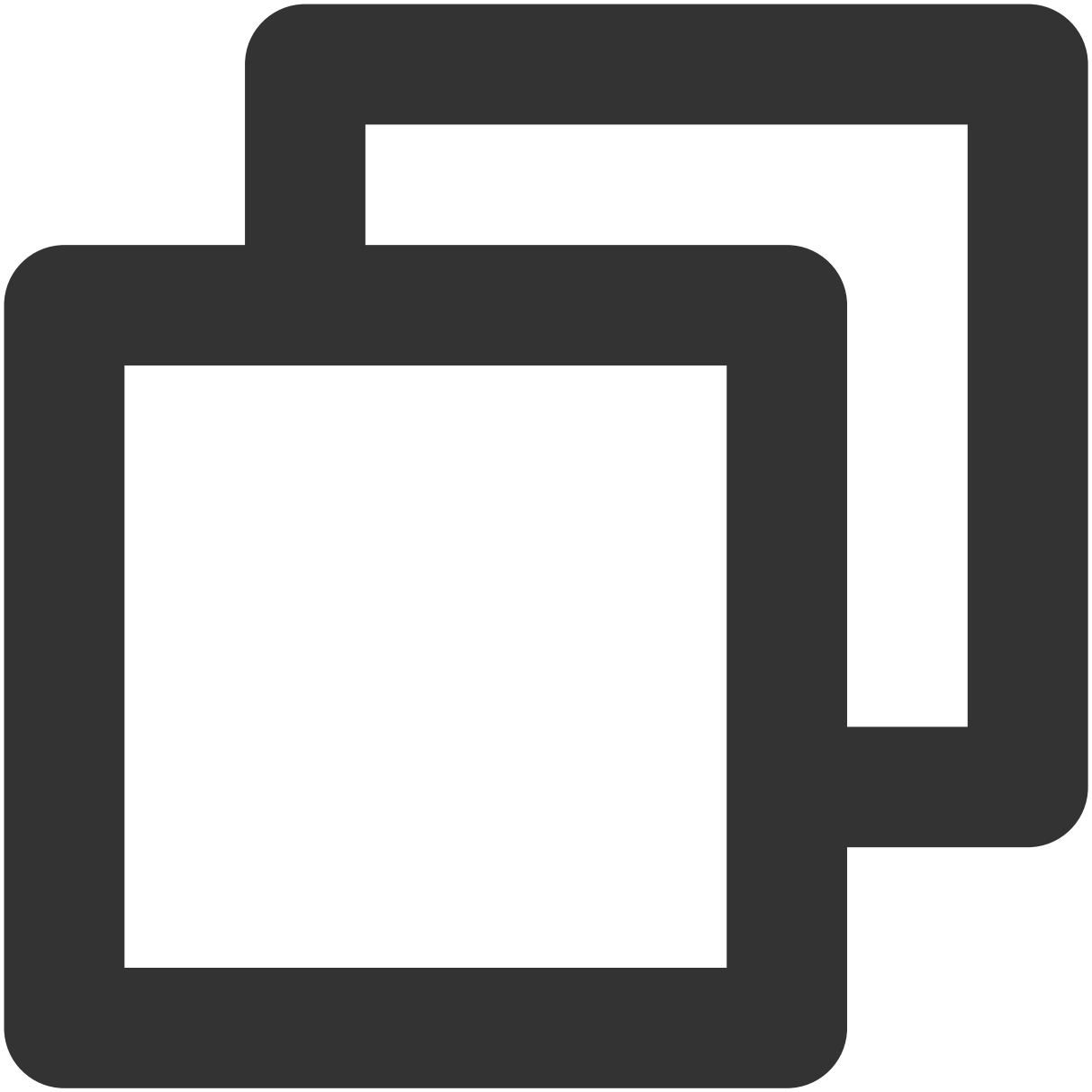
停止向远端推本地音频流。



```
Future<TUIActionCallback> muteLocalAudio()
```

unMuteLocalAudio

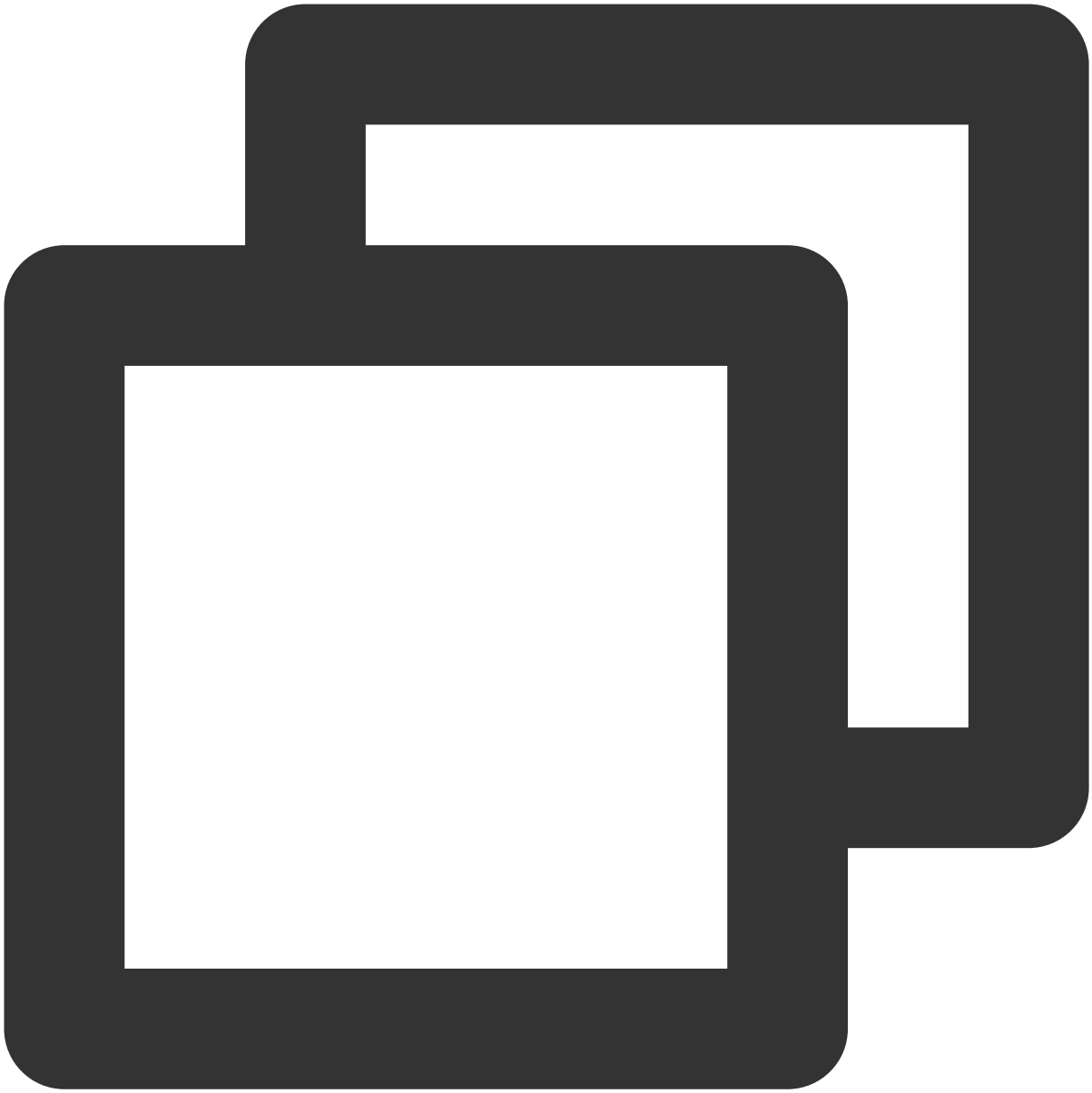
开始向远端推本地音频流。



```
Future<TUIActionCallback> unMuteLocalAudio()
```

setRemoteVideoView

设置远端用户视频渲染的视图控件。



```
void setRemoteVideoView(String userId,
                        TUIVideoStreamType streamType,
                        int viewId)
```

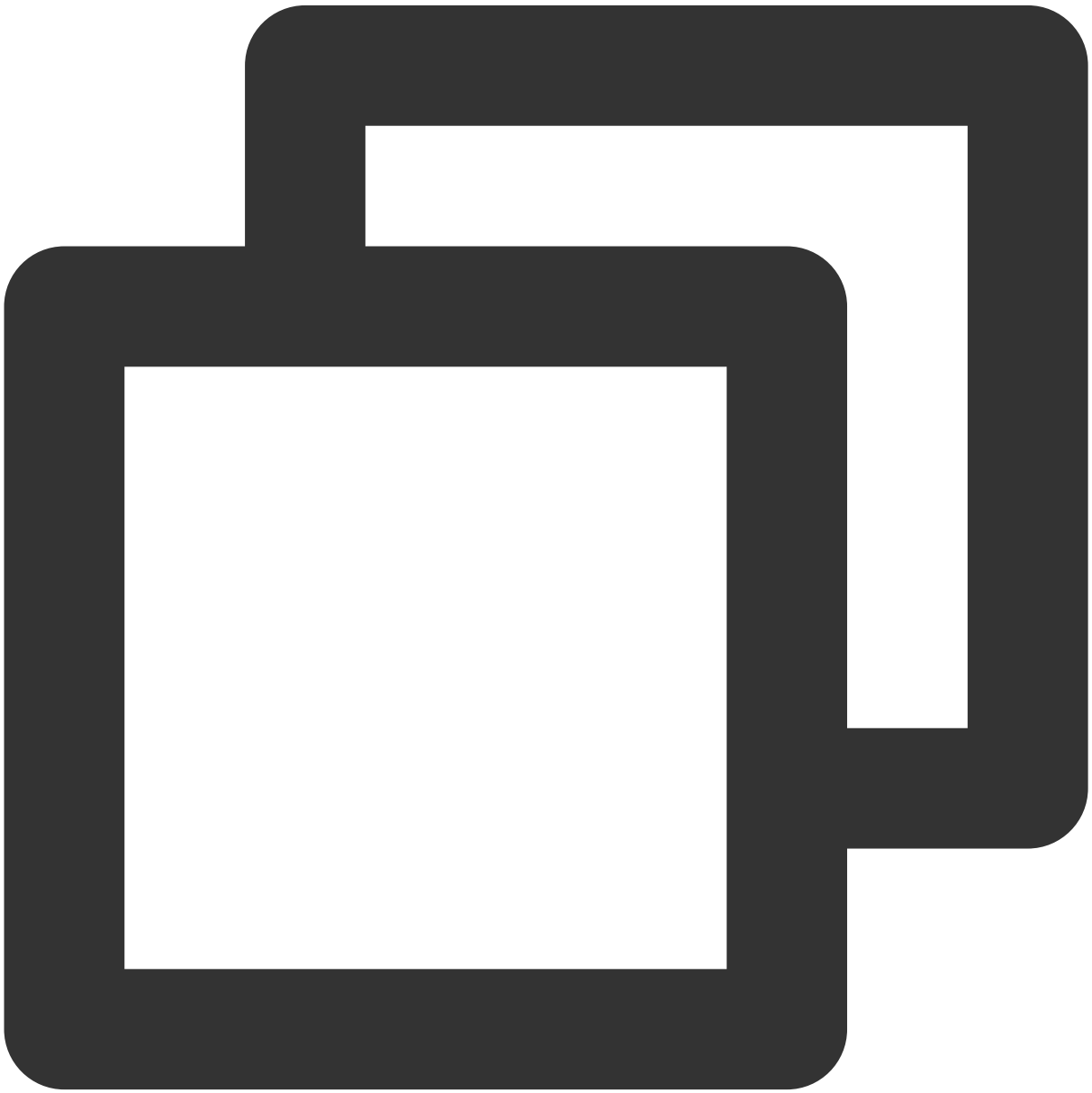
参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIVideoStreamType	用户流类型

viewId	int	待渲染 view 指针的 int64 类型值，通过此 viewId 可转换为对应原生平台的视图，视频画面将渲染于此视图上

startPlayRemoteVideo

开始播放远端用户视频流。



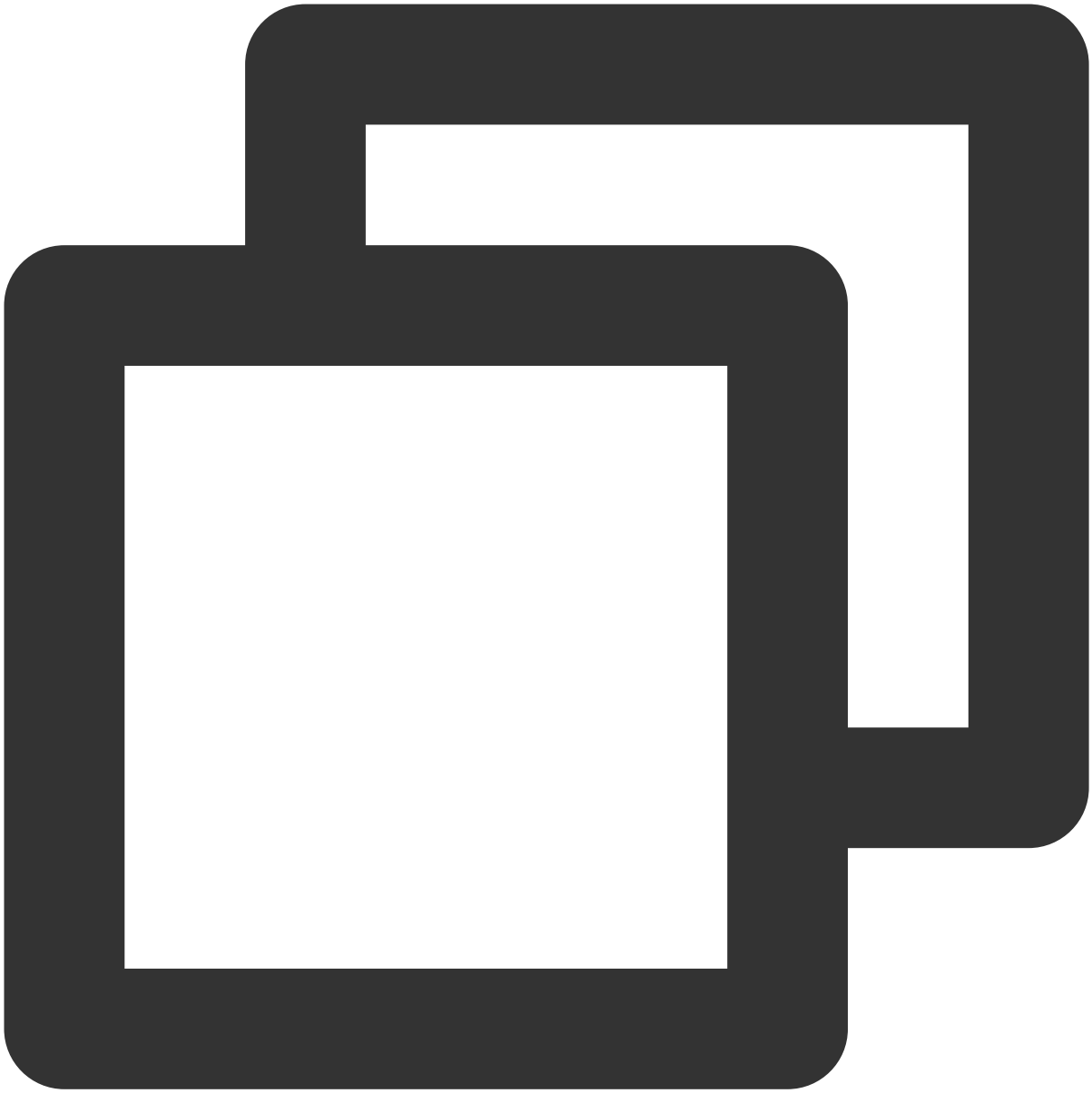
```
void startPlayRemoteVideo(String userId,
                          TUIVideoStreamType streamType,
                          TUIPlayCallback? callback)
```

参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIVideoStreamType	用户流类型
callback	TUIPlayCallback?	播放结果回调

stopPlayRemoteVideo

停止播放远端用户视频流。



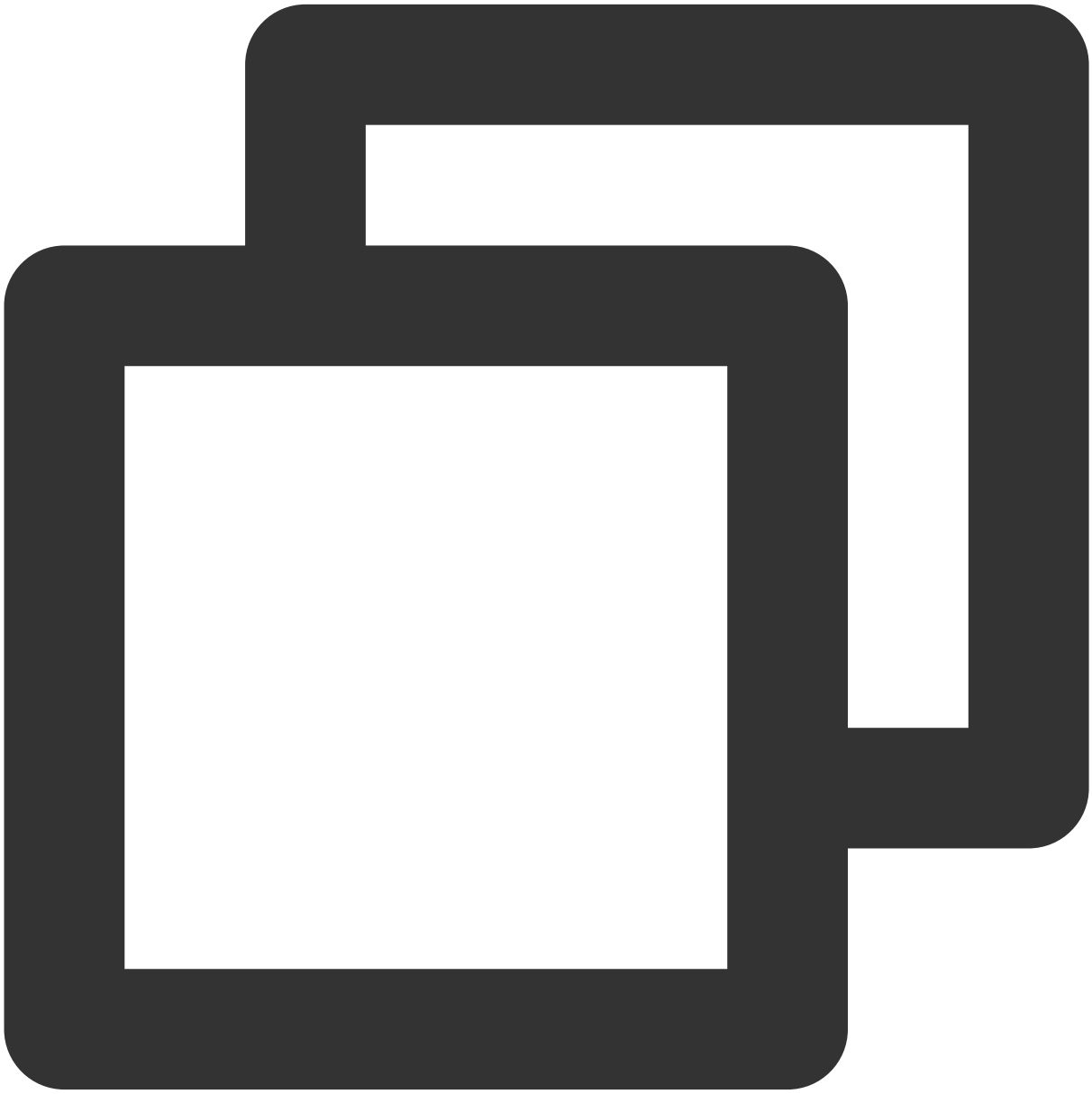
```
void stopPlayRemoteVideo(String userId,
                         TUIVideoStreamType streamType)
```

参数：

参数	类型	含义
userId	String	用户 ID
streamType	TUIVideoStreamType	用户流类型

muteRemoteAudioStream

将远端用户禁音。



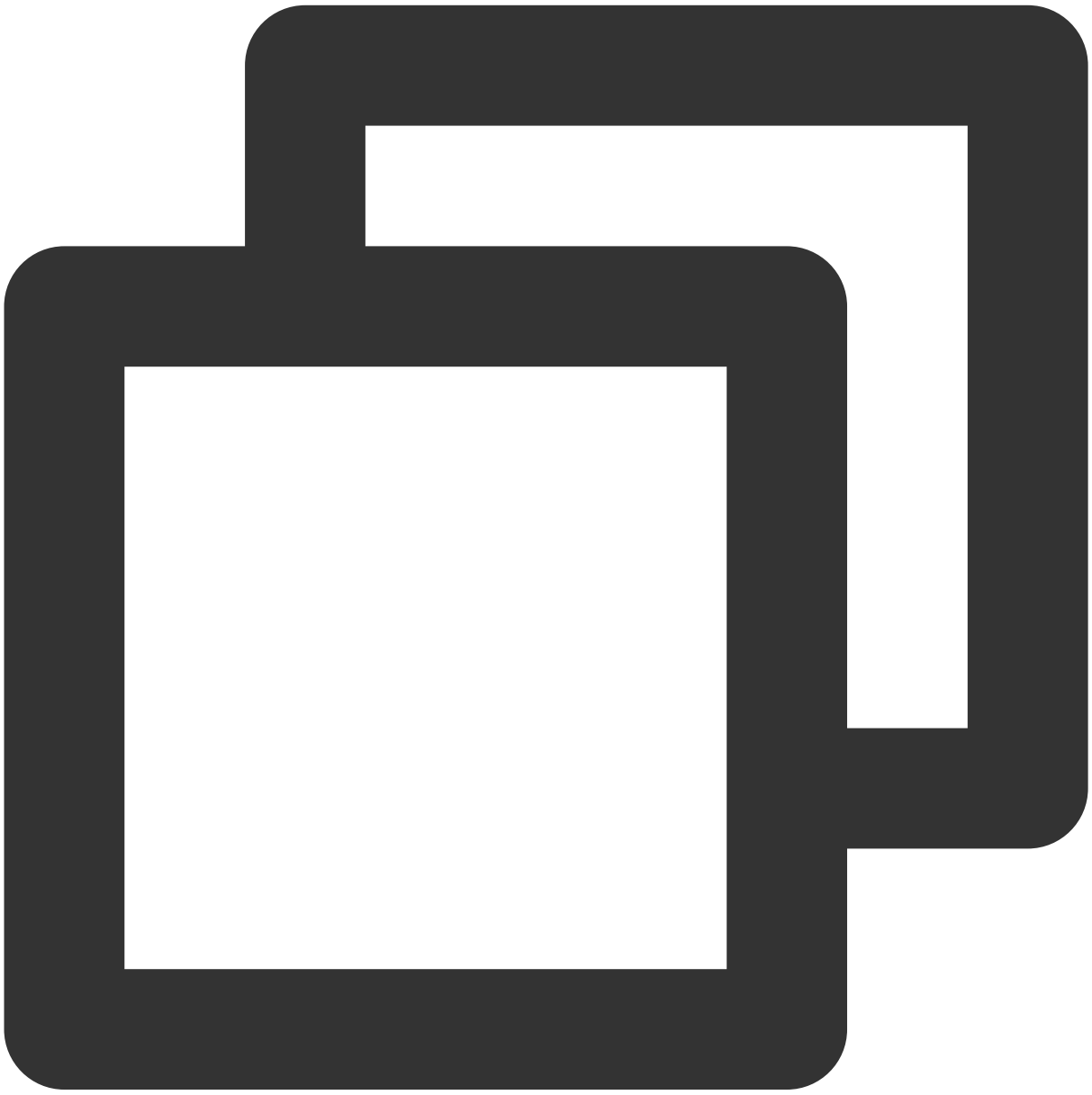
```
void muteRemoteAudioStream(String userId, boolean isMute);
```

参数：

参数	类型	含义
userId	String	用户 ID
isMute	bool	是否禁音

getUserList

获取当前房间用户列表，注意该接口一次拉取的用户列表量最大为 100 个。



```
Future<TUIValueCallBack<TUIUserListResult>> getUserList(int nextSequence)
```

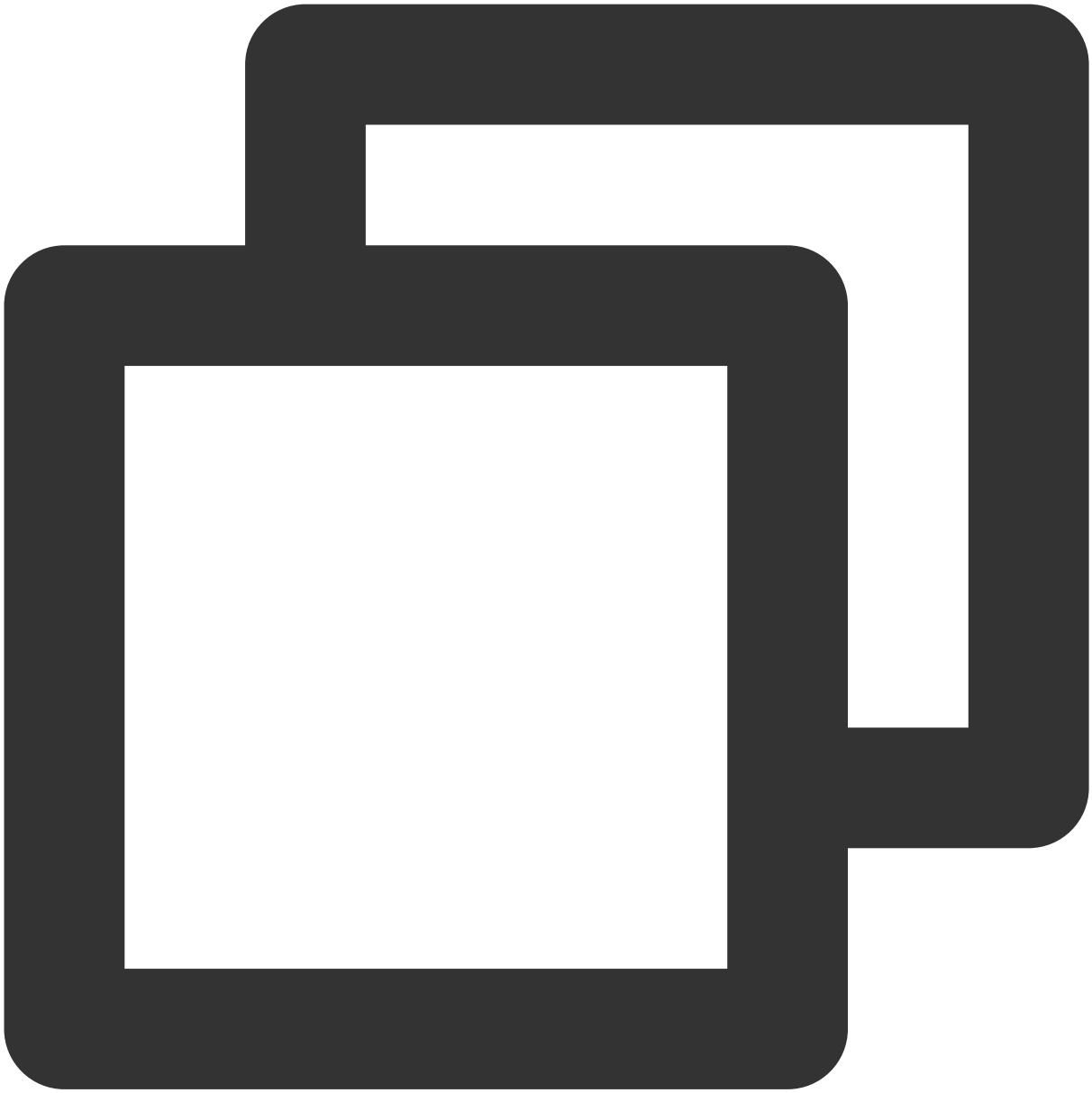
参数：

参数	类型	含义
nextSequence	int	分页拉取标志，第一次拉取填0。回调成功如果

		nextSeq 不为零，需要分页，传入 nextSeq 再次拉取，直至 callback 中 nextSeq 为0

getUserInfo

获取用户的详细信息。



```
Future<TUIValueCallBack<TUIUserInfo>> getUserInfo(String userId)
```

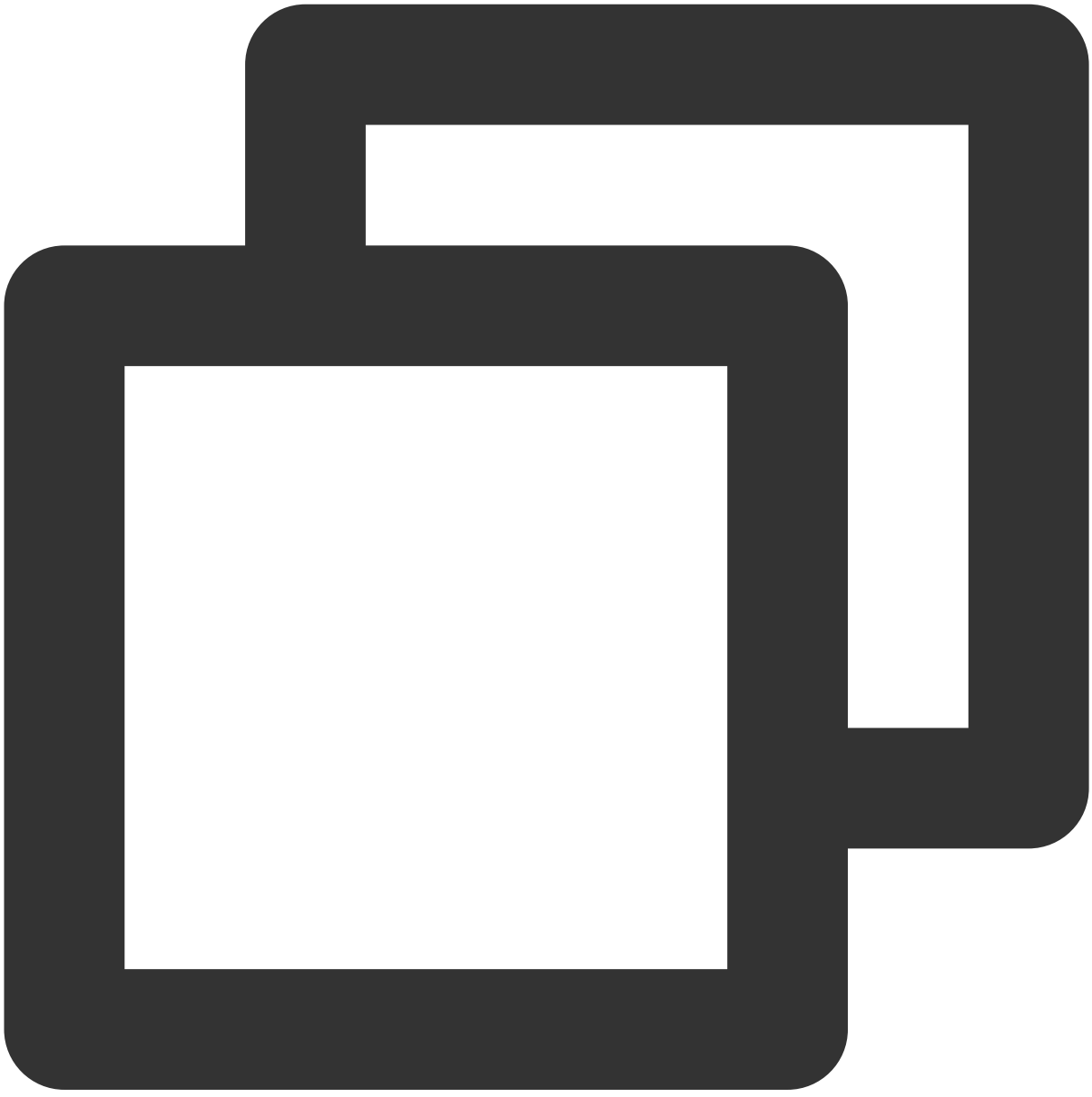
参数：

参数	类型	含义

userId	String	根据 userId 获取该用户的详细信息

changeUserRole

改变用户的角色，只有管理员或群主能够调用。



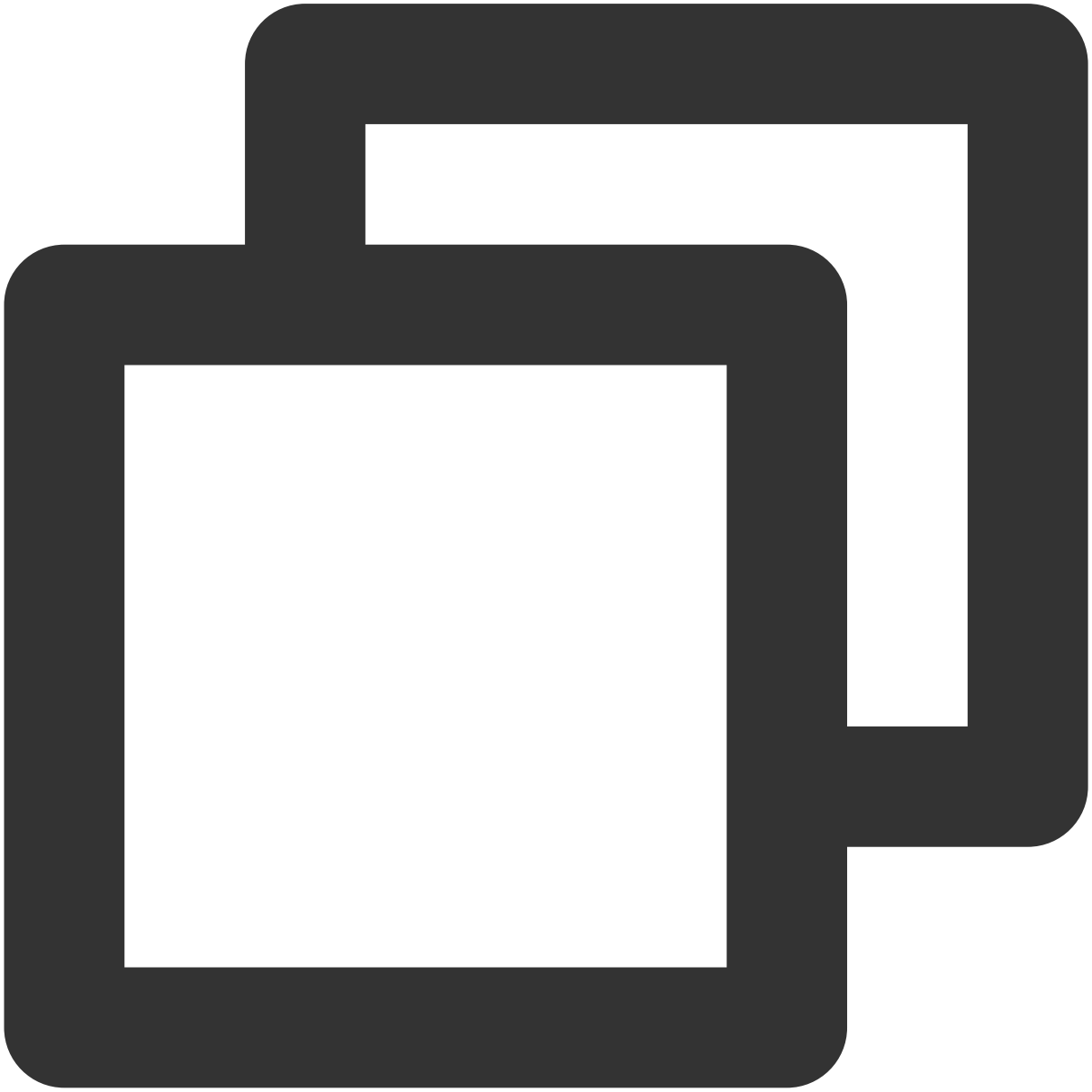
```
Future<TUIActionCallback> changeUserRole(String userId,
                                         TUIRole role)
```

参数：

参数	类型	含义
userId	String	用户 ID
role	TUIRole	用户角色

kickRemoteUserOutOfRoom

将用户移出房间，只有管理员或群主能够调用。



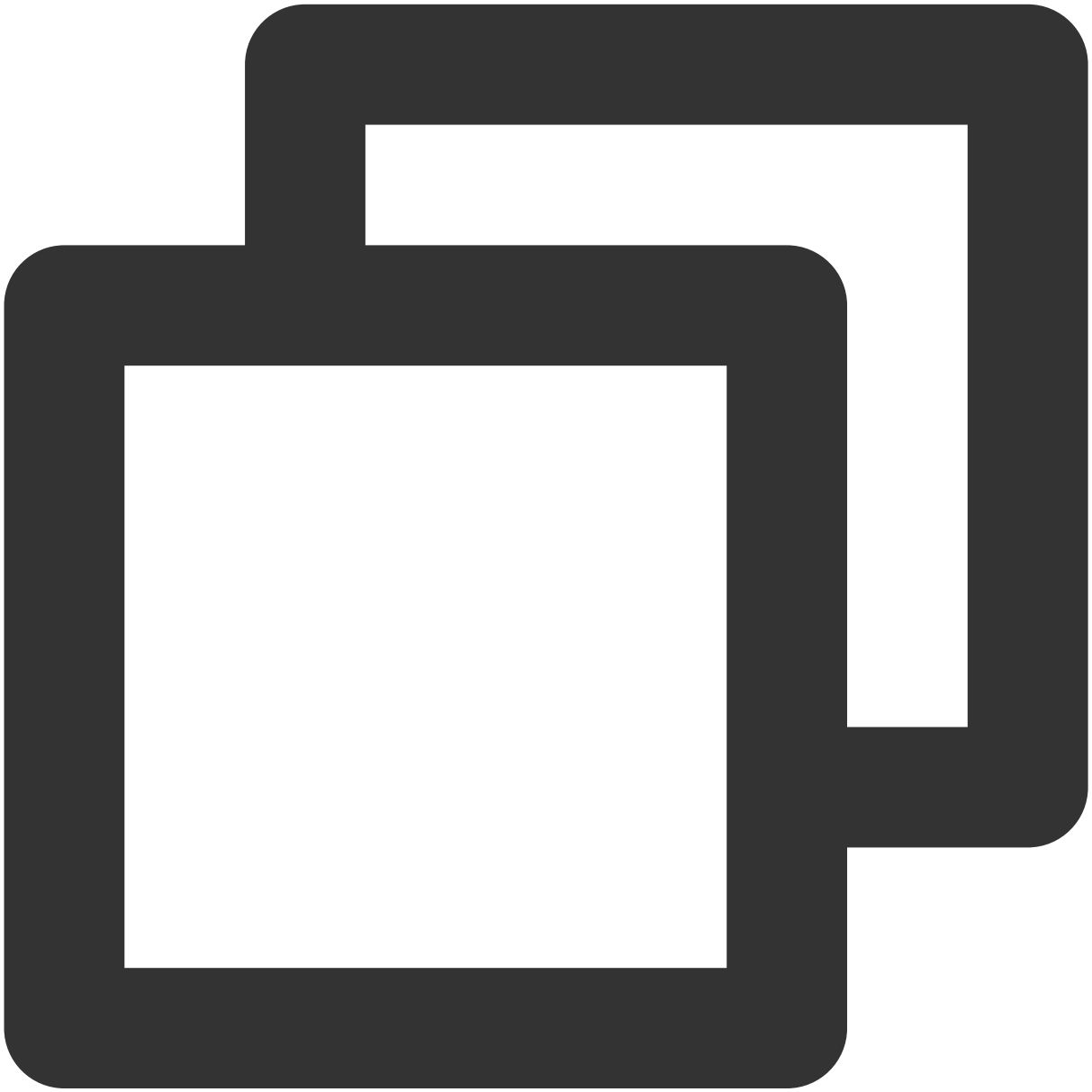
```
Future<TUIActionCallback> kickRemoteUserOutOfRoom(String userId)
```

参数：

参数	类型	含义
userId	String	用户 ID

addCategoryTagForUsers

为用户增加标记，只有房主能够调用



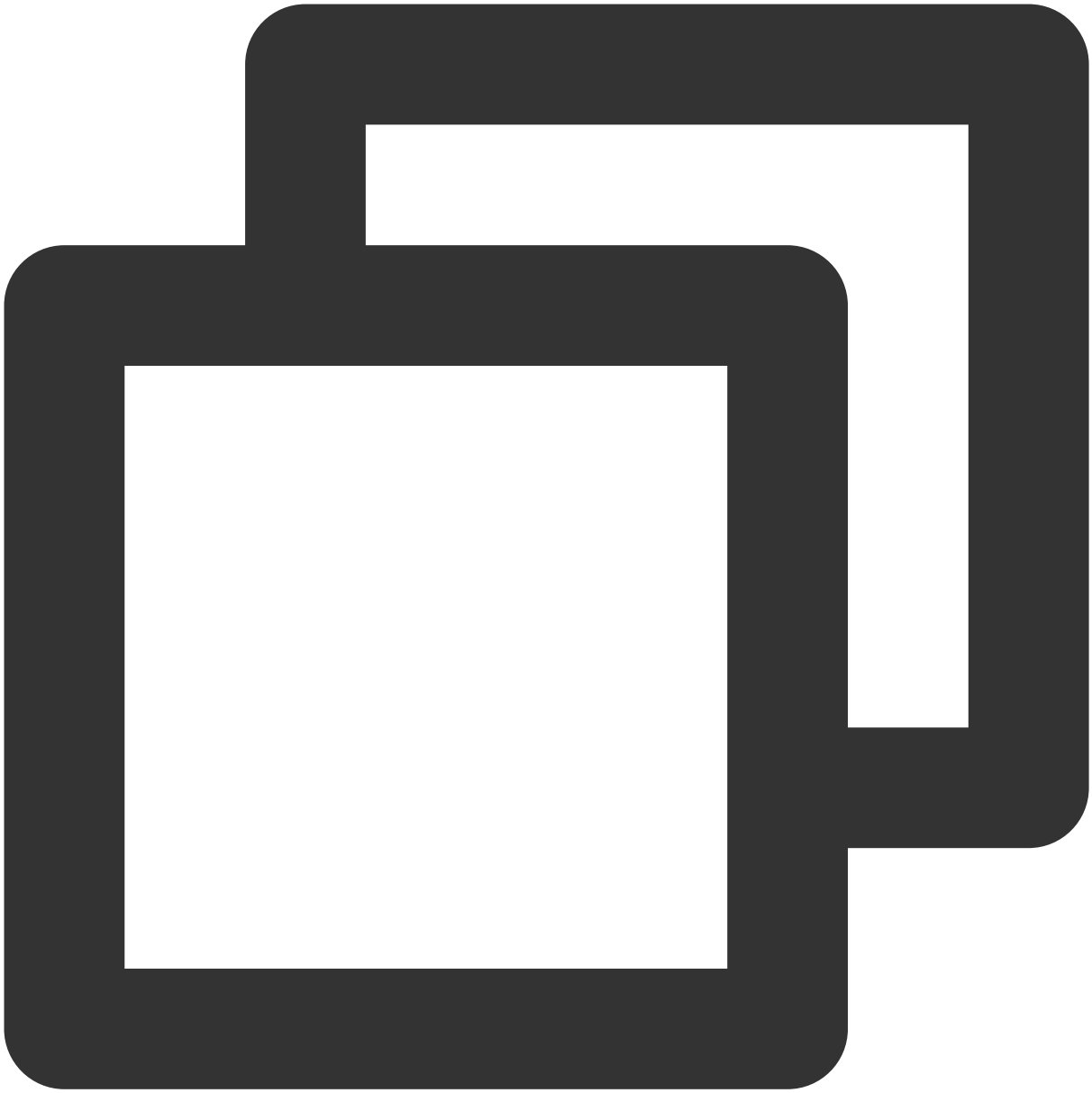
```
Future<TUIActionCallback> addCategoryTagForUsers(int tag, List<String> userList);
```

参数：

参数	类型	含义
tag	int	标记类型。数字类型，大于等于 1000，您可以自定义。
userList	List<String>	用户列表

removeCategoryTagForUsers

为用户移除标记，只有房主能够调用



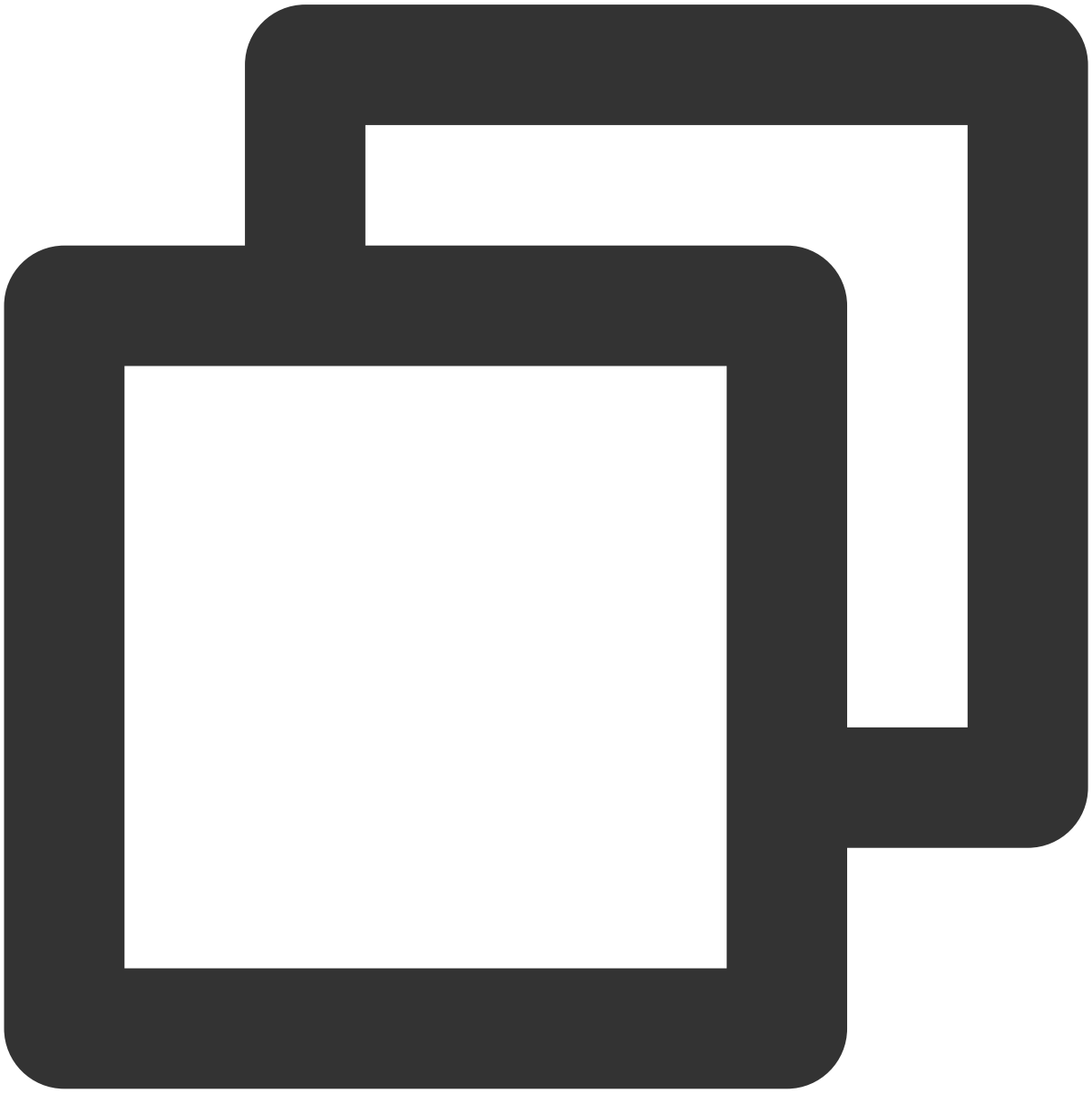
```
Future<TUIActionCallback> removeCategoryTagForUsers(int tag, List<String> userList)
```

参数：

参数	类型	含义
tag	int	类型。数字类型，大于等于 1000，您可以自定义。
userList	List<String>	用户列表

getUserListByTag

根据标记获取房间内用户信息
为用户移除标记，只有房主能够调用



```
Future<TUIValueCallBack<TUIUserListResult>> getUserListByTag(int tag, int nextSeque
```

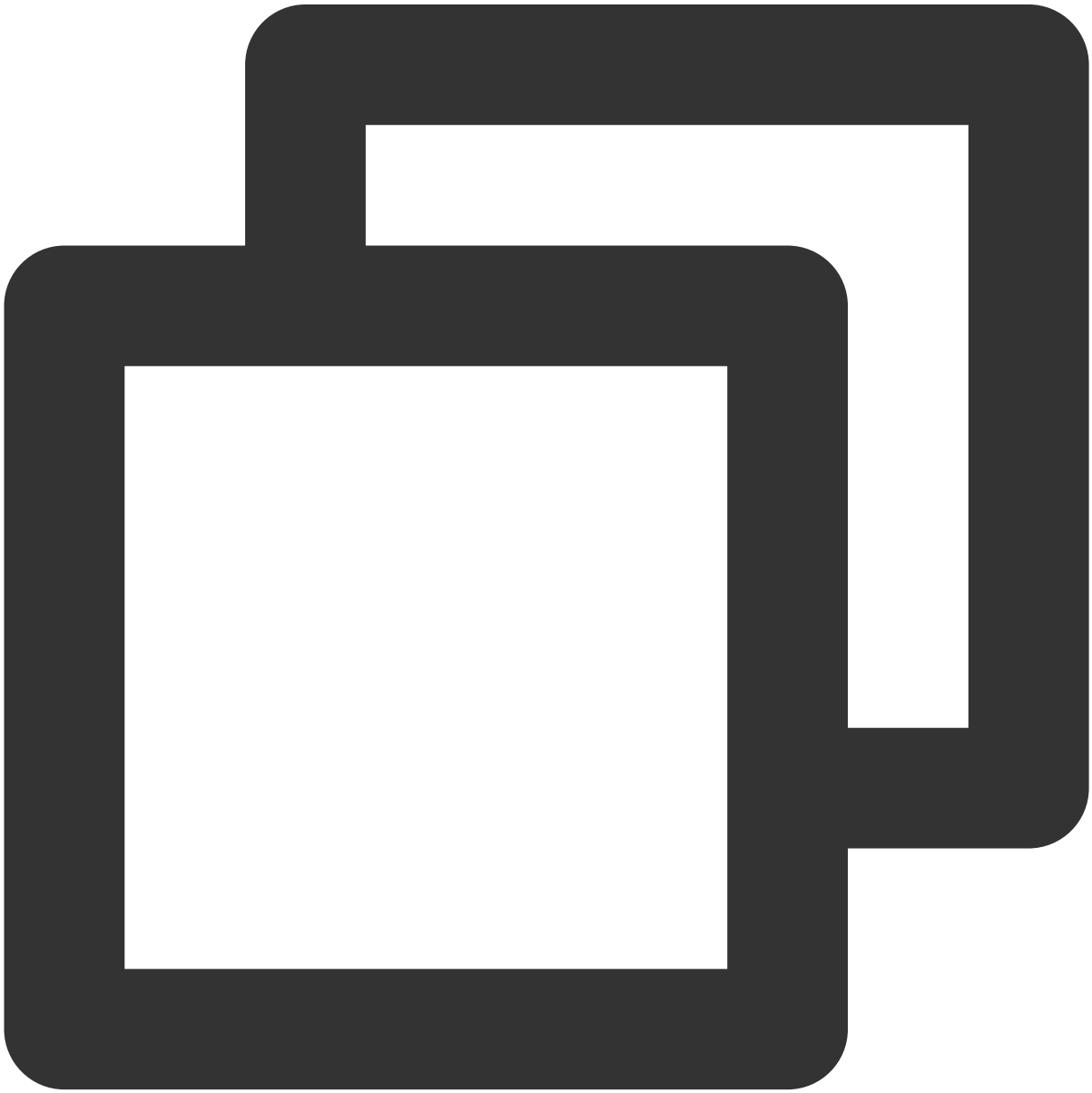
参数：

参数	类型	含义
tag	int	类型。数字类型，大于等于 1000，您可以自定义。

nextSequence	int	分页拉取标志，第一次拉取填0，回调成功 如果 callback 返回的数据中 nextSequence 不为零，需要分页，传入再次拉取，直至为0

disableDeviceForAllUserByAdmin

全体用户媒体设备管理，只有管理员或群主能够调用。

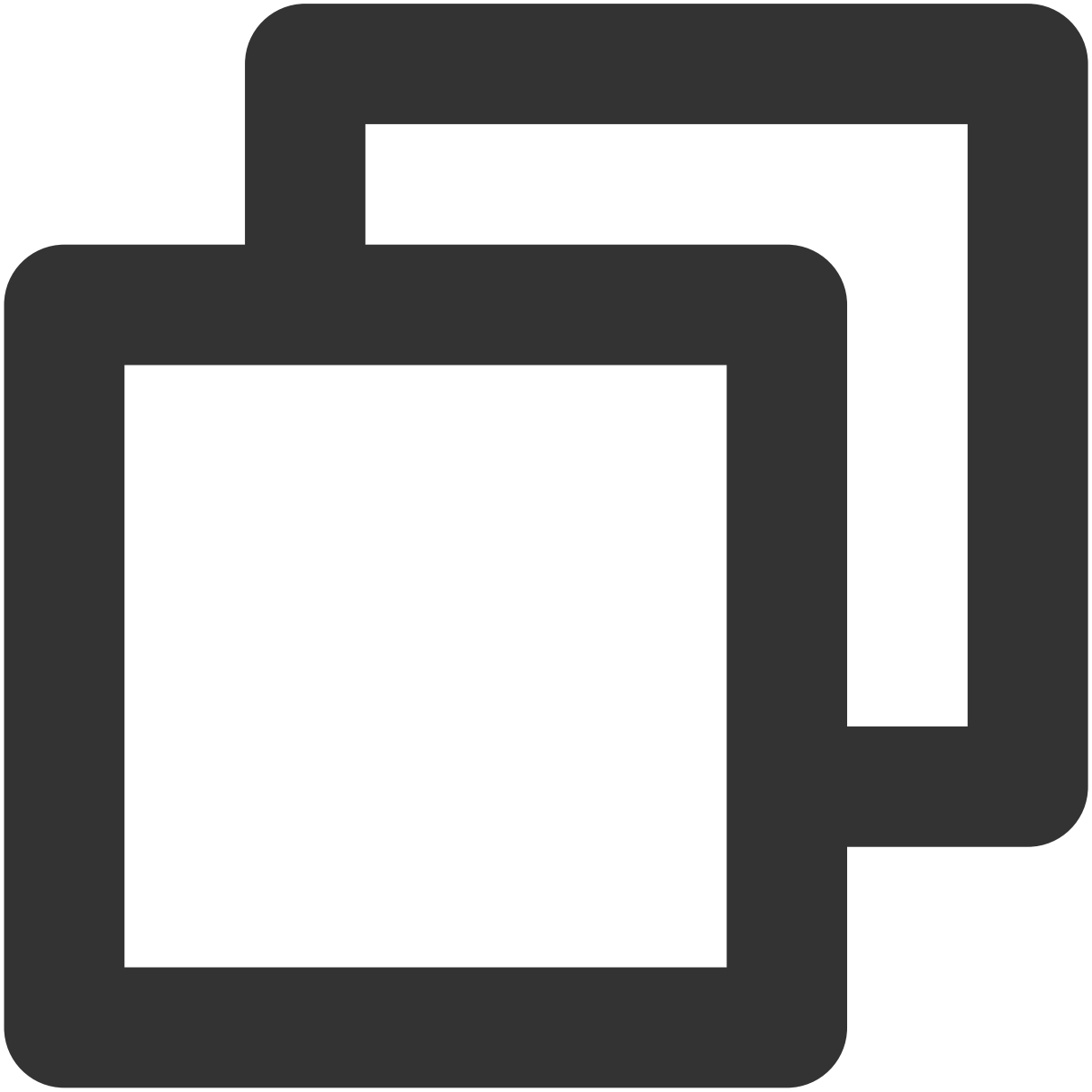


```
Future<TUIActionCallback> disableDeviceForAllUserByAdmin(TUIMediaDevice device,
                                                         bool isDisable)
```

参数	类型	含义
device	TUIMediaDevice	设备
isDisable	bool	是否禁用

openRemoteDeviceByAdmin

请求远端用户打开媒体设备，只有管理员或群主能够调用。



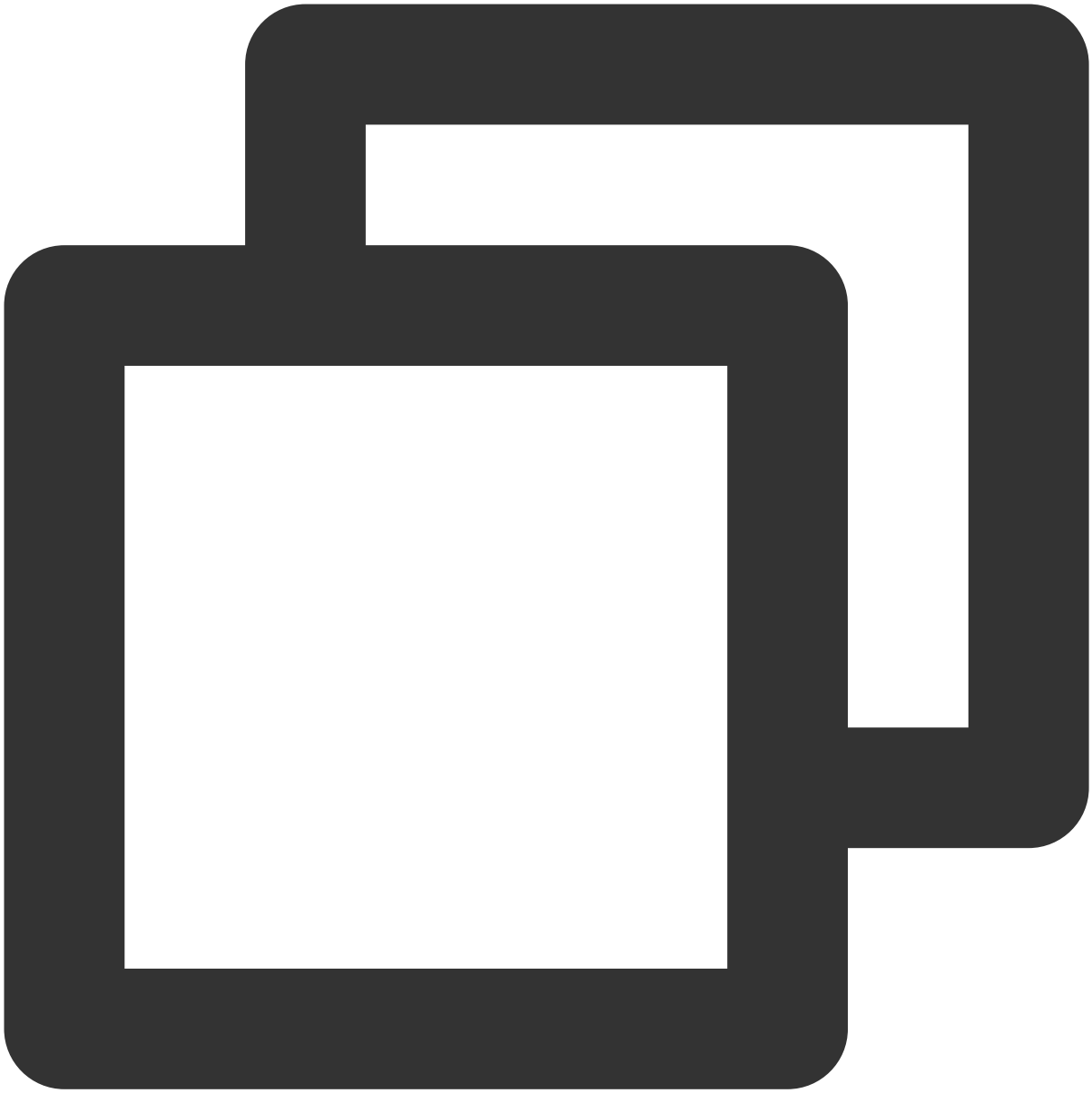
```
TUIRequest openRemoteDeviceByAdmin(String userId,
```

```
                                TUIMediaDevice device,
                                int timeout,
                                TUIRequestCallback? requestCallback)
```

参数	类型	含义
userId	String	用户 ID
device	TUIMediaDevice	设备
timeout	int	超时时间，单位秒。如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
requestCallback	TUIRequestCallback?	操作结果回调

closeRemoteDeviceByAdmin

关闭远端用户媒体设备，只有管理员或群主能够调用。

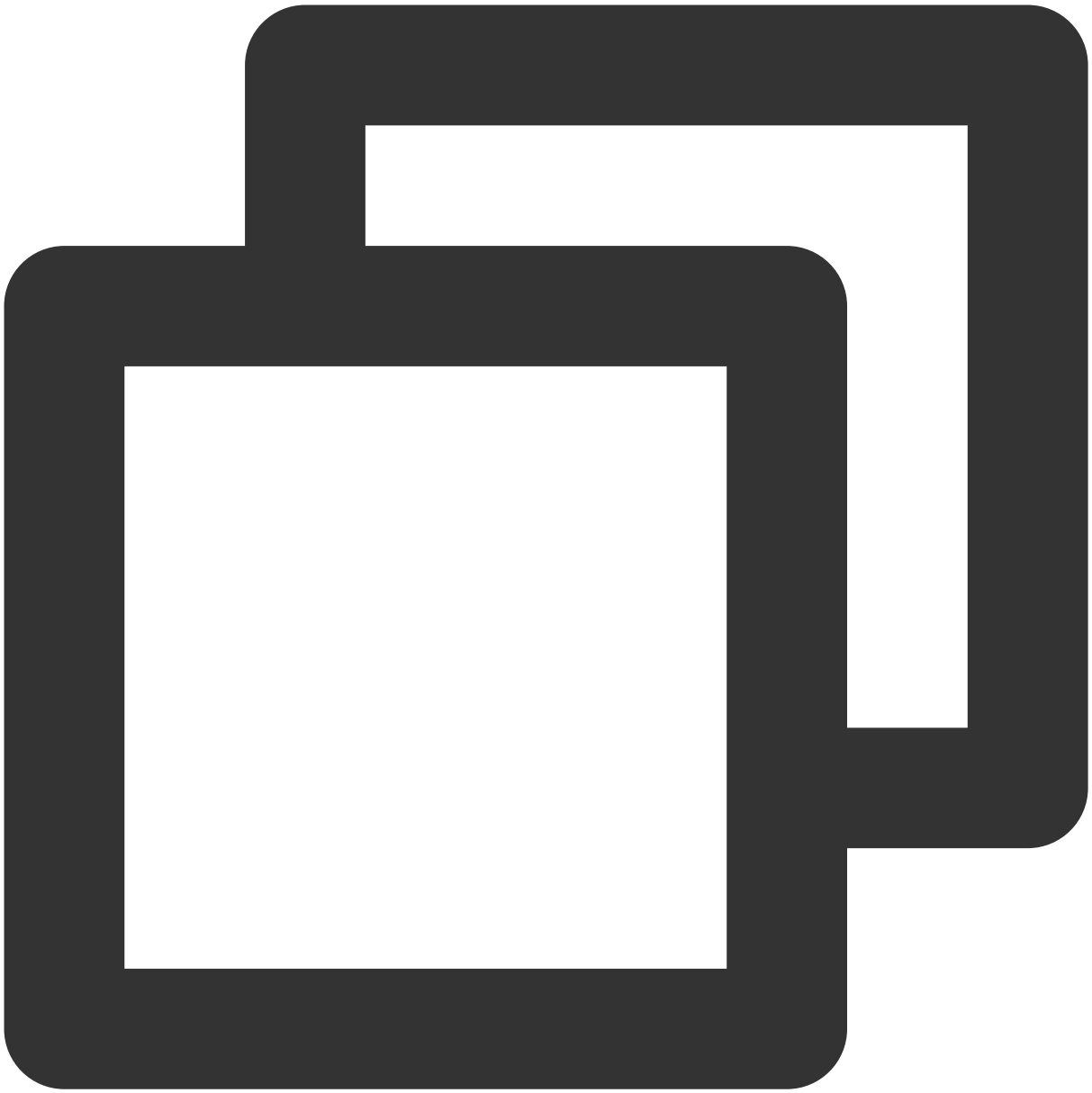


```
Future<TUIActionCallback> closeRemoteDeviceByAdmin(String userId,
                                                   TUIMediaDevice device)
```

参数	类型	含义
userId	String	用户ID
device	TUIMediaDevice	设备

applyToAdminToOpenLocalDevice

全体用户媒体设备管理 lock。



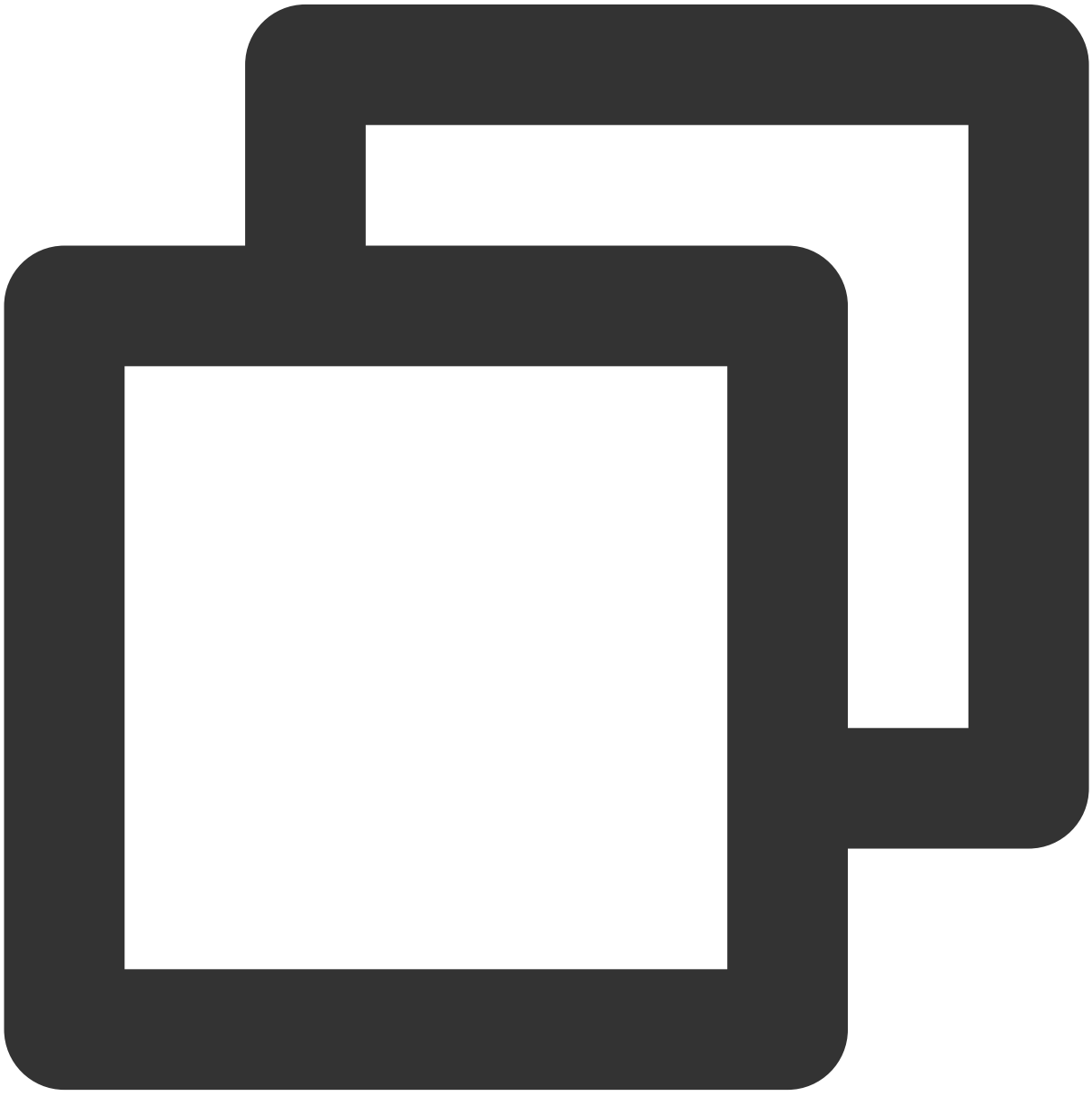
```
TUIRequest applyToAdminToOpenLocalDevice(TUIMediaDevice device,
                                         int timeout,
                                         TUIRequestCallback? requestCallback)
```

参数	类型	含义
device	TUIMediaDevice	设备
timeout	int	超时时间，单位秒。如果设置为 0，SDK 不会做超时

		检测，也不会触发超时回调
callback	TUIRequestCallback?	操作结果回调

setMaxSeatCount

设置最大麦位数，仅支持进房前和创建房间时设置。



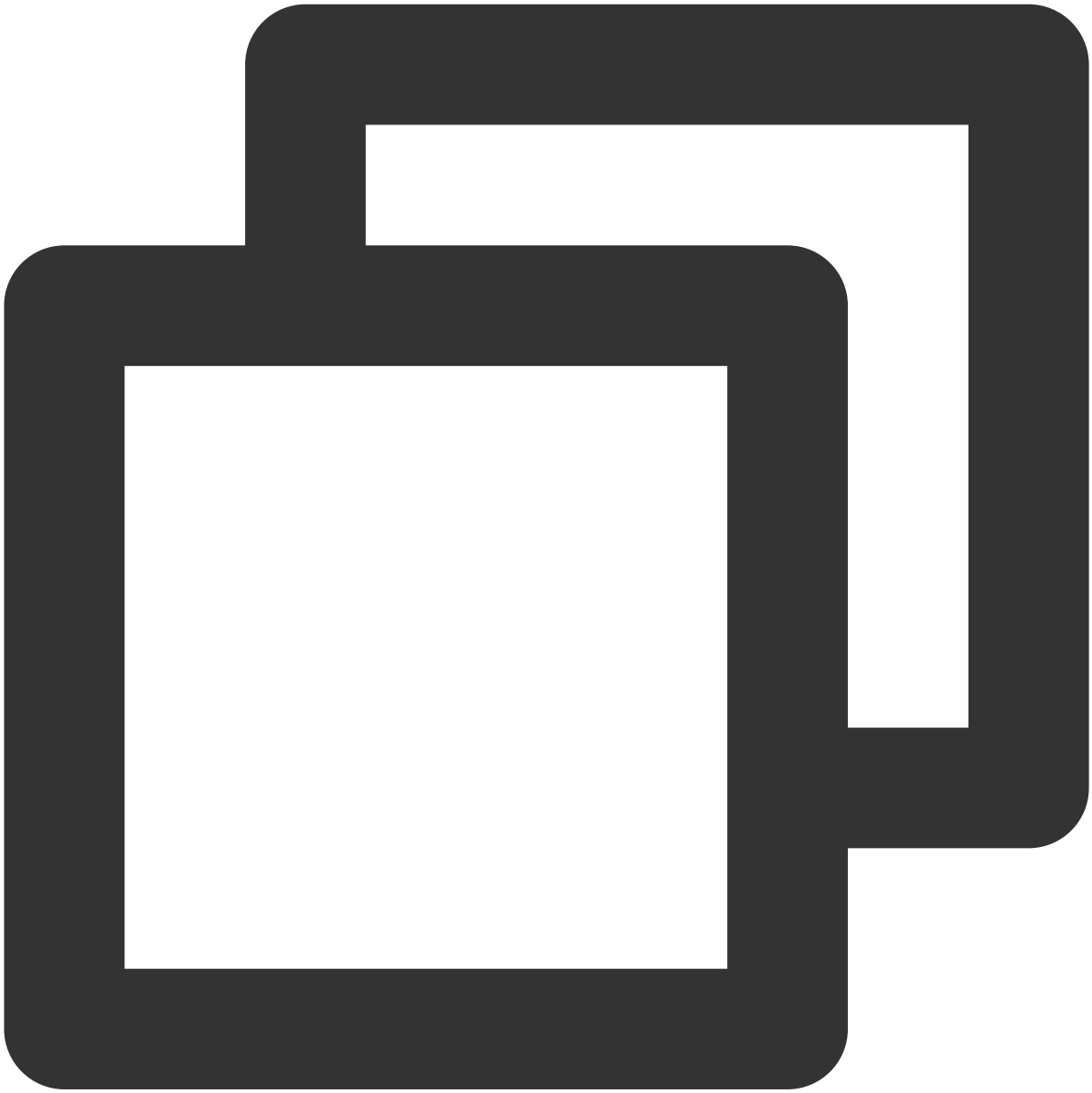
```
Future<TUIActionCallback> setMaxSeatCount(int maxSeatCount)
```

参数	类型	含义

maxSeatCount	int	最大麦位数

lockSeatByAdmin

锁定麦位（包括位置锁定，音频状态锁定，视频状态锁定）。



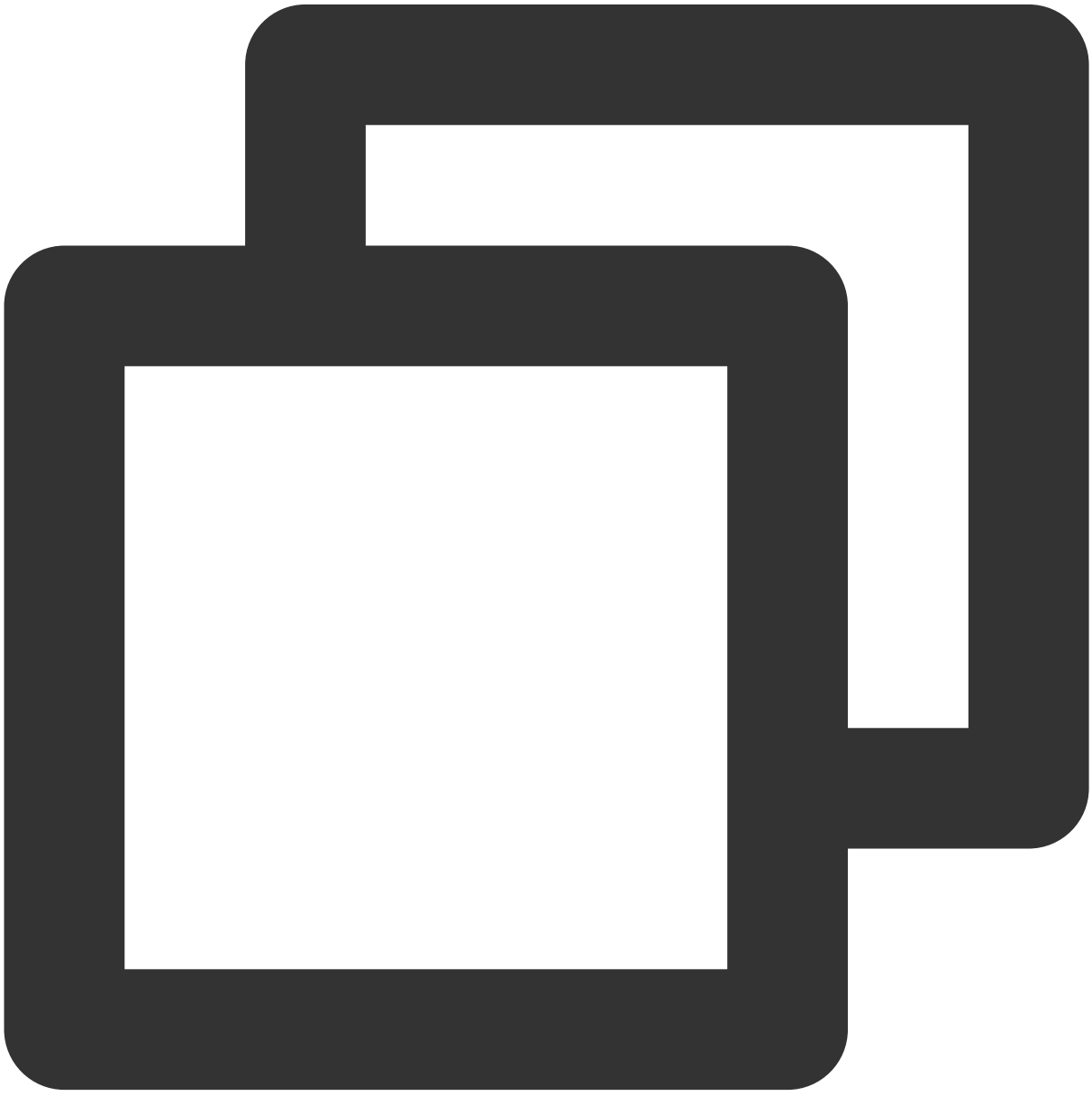
```
Future<TUIActionCallback> lockSeatByAdmin(int seatIndex,
                                  TUISeatLockParams lockParams)
```

参数	类型	含义

seatIndex	int	麦位编号
lockParams	TUISeatLockParams	锁麦参数

getSeatList

获取麦位列表。



```
Future<TUIValueCallBack<List<TUISeatInfo>>> getSeatList()
```

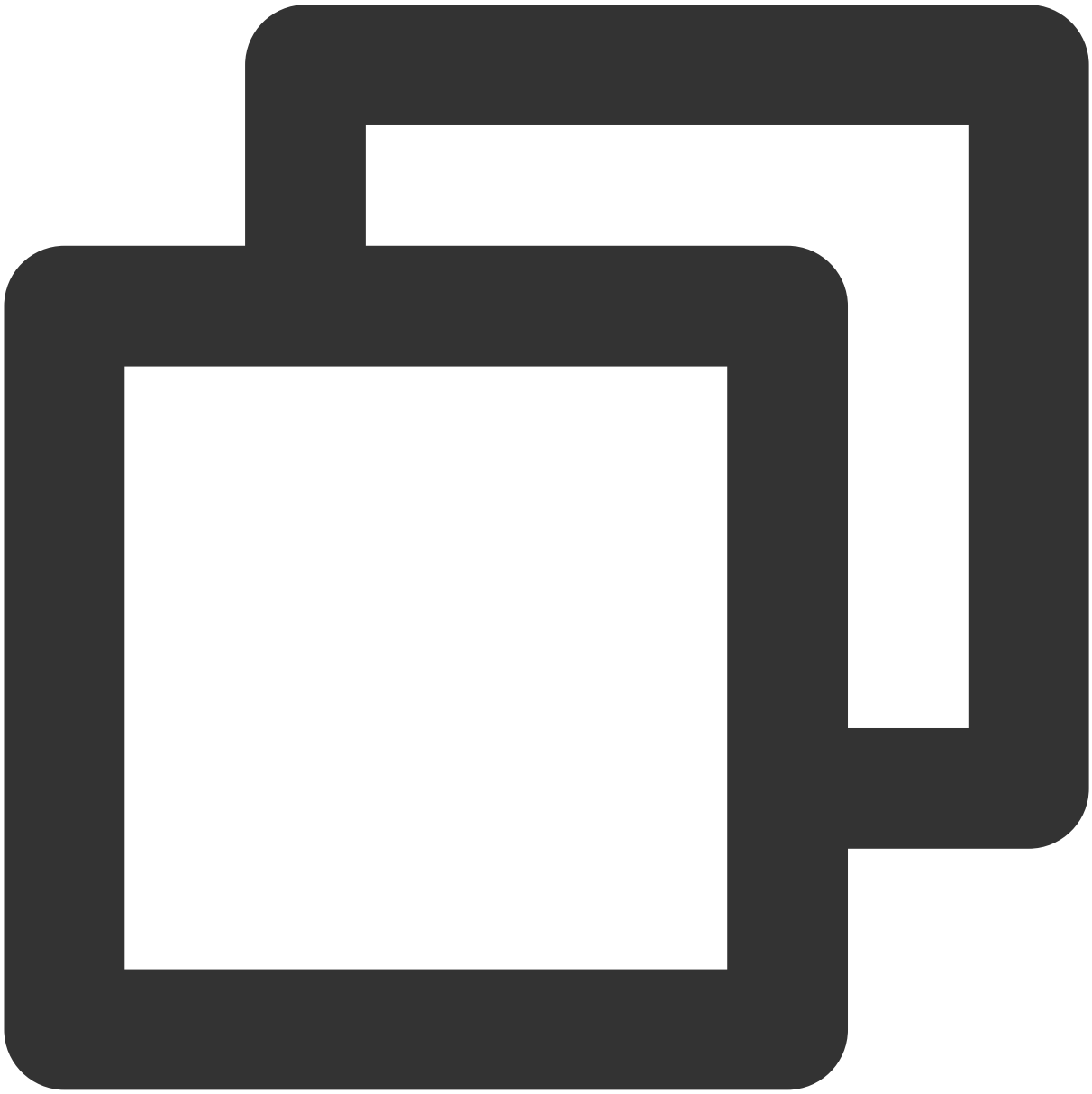
takeSeat

本地上麦。

说明：

会议场景：applyToSpeak 模式需要向主持人或管理员发起申请才允许上麦，其他模式不支持上麦。

直播场景：freeToSpeak 模式可以自由上麦，上麦后开麦发言；applySpeakAfterTakingSeat 模式需要向主持人或管理员发起申请才允许上麦；其他模式不支持上麦。



```
TUIRequest takeSeat(int seatIndex,
                    int timeout,
                    TUIRequestCallback? requestCallback)
```

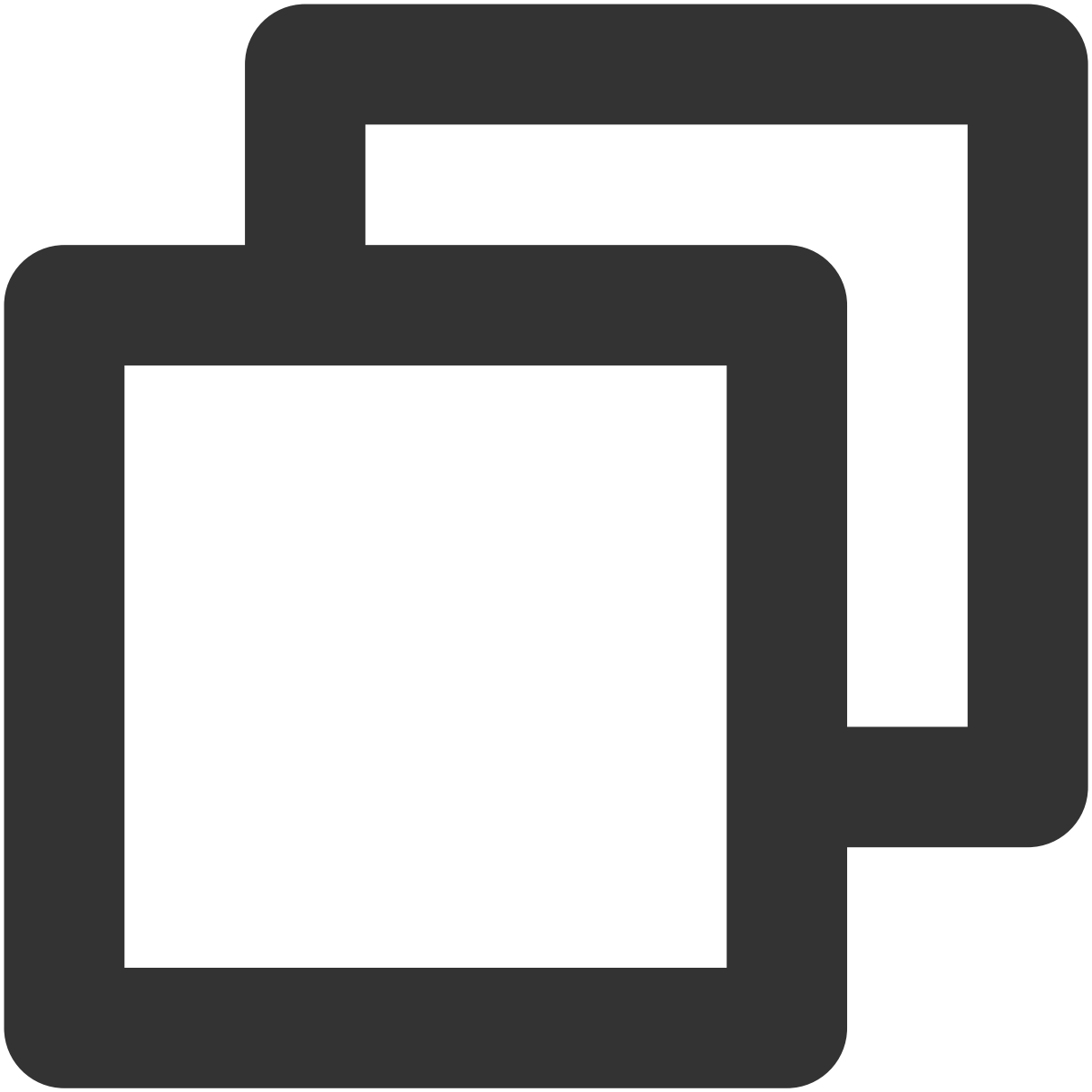
参数：

参数	类型	含义
seatIndex	int	麦位编号
timeout	int	超时时间，单位秒。如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
requestCallback	TUIRequestCallback?	调用接口的回调，用于通知请求的回调状态

Return: 请求体

leaveSeat

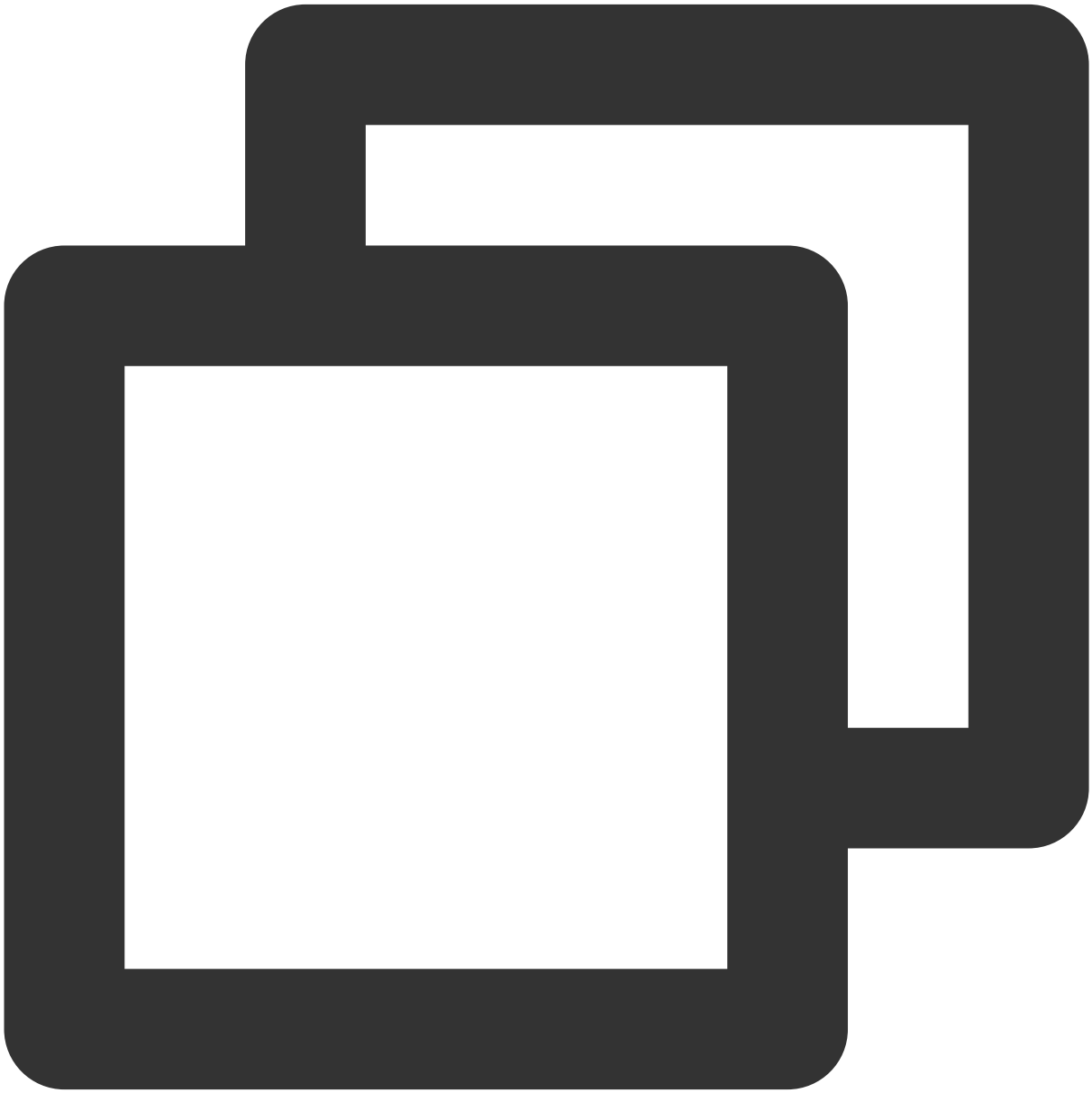
本地下麦。



```
Future<TUIActionCallback> leaveSeat()
```

takeUserOnSeatByAdmin

主持人/管理员 邀请用户上麦。



```
TUIRequest takeUserOnSeatByAdmin(int seatIndex,
                                 String userId,
                                 int timeout,
                                 TUIRequestCallback? requestCallback)
```

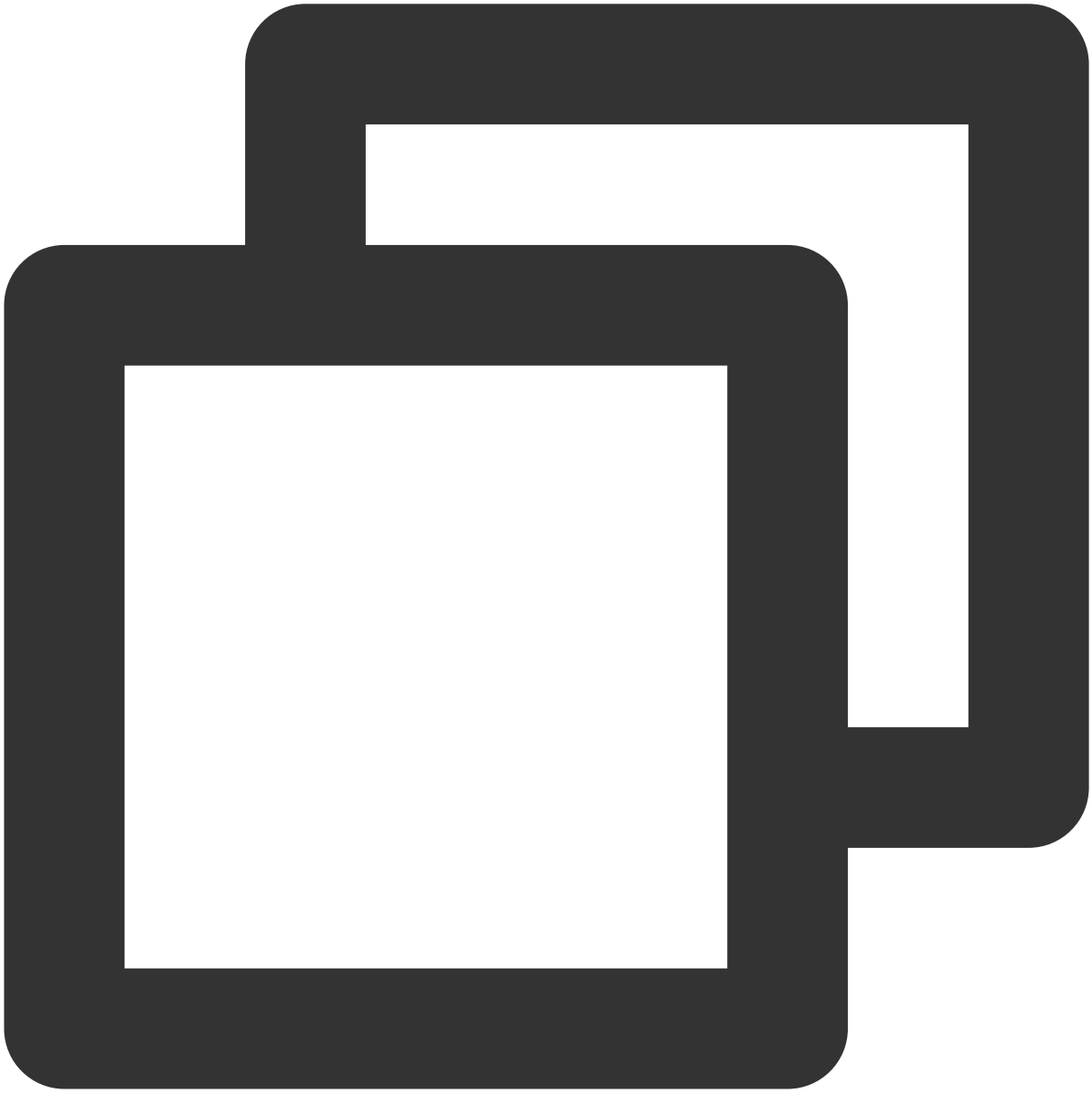
参数	类型	含义
seatIndex	int	麦位编号
userId	String	用户 ID

timeout	int	超时时间，单位秒。如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
requestCallback	TUIRequestCallback?	调用接口的回调，用于通知请求的回调状态

Return: 请求体

kickUserOffSeatByAdmin

主持人/管理员 将用户下麦。

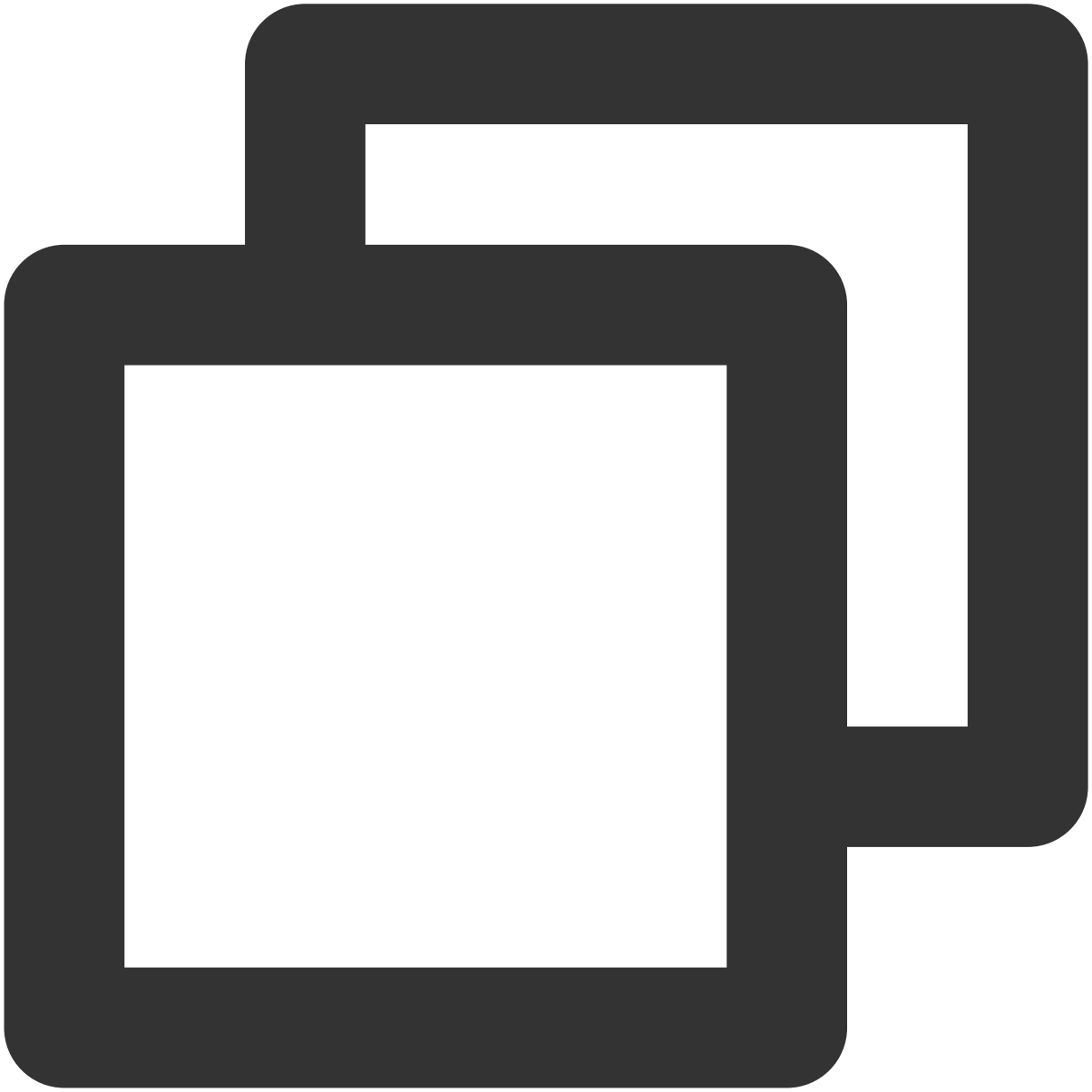


```
Future<TUIActionCallback> kickUserOffSeatByAdmin(int seatIndex,
                                                 String userId)
```

参数	类型	含义
seatIndex	int	麦位编号
userId	String	用户 ID

sendTextMessage

发送文本消息。



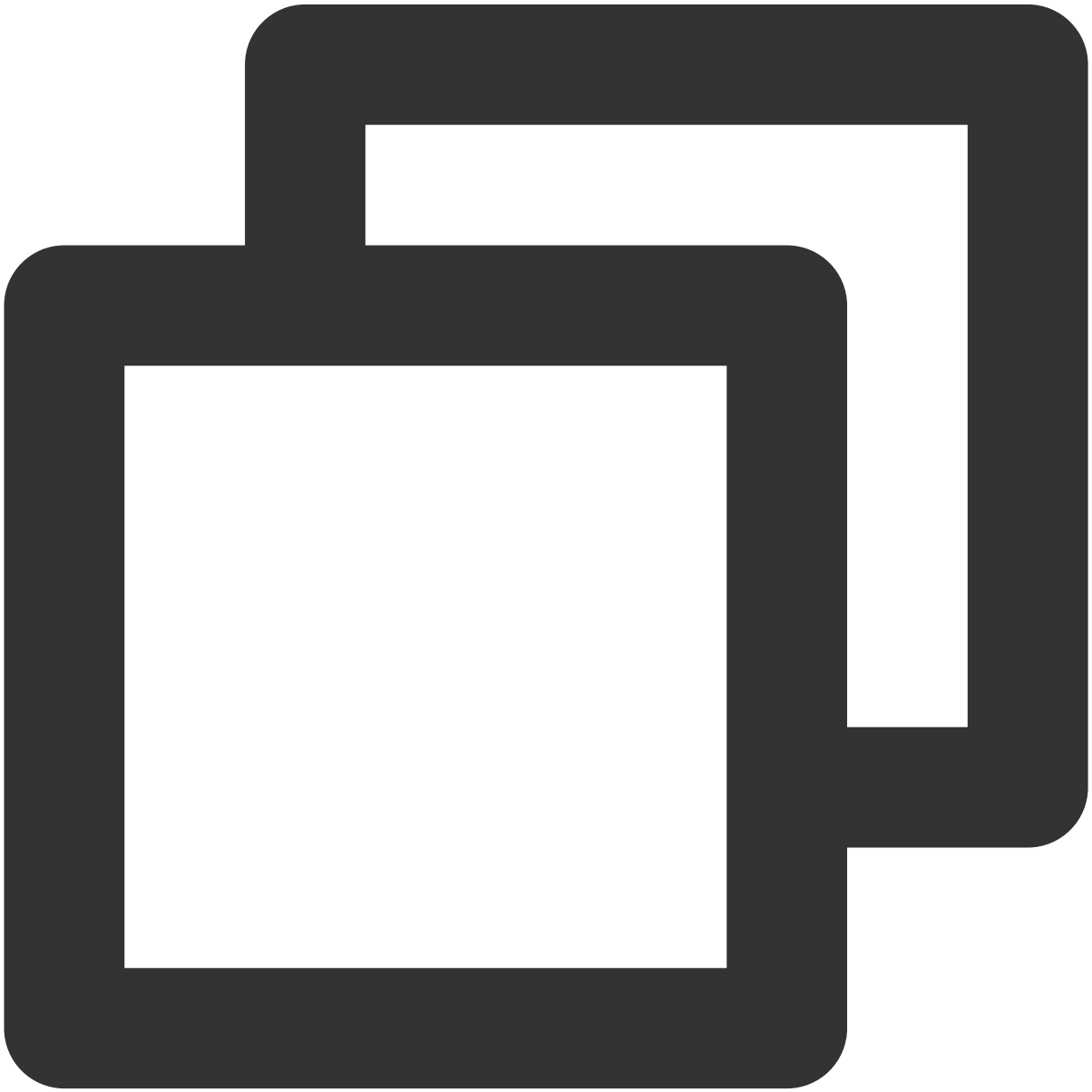
```
Future<TUIActionCallback> sendTextMessage(String message)
```

参数：

参数	类型	含义
message	String	文本消息内容

sendCustomMessage

发送自定义消息。



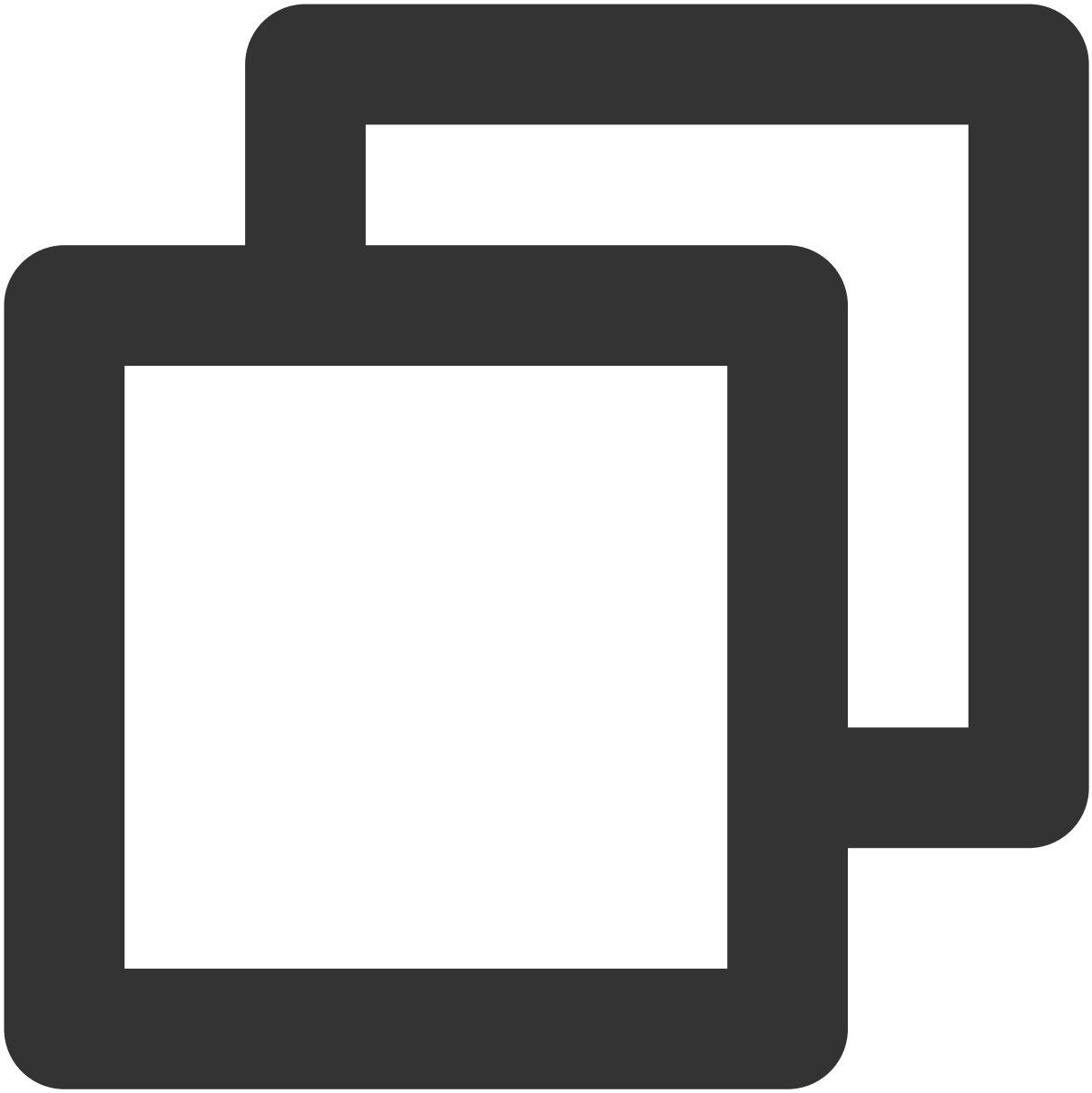
```
Future<TUIActionCallback> sendCustomMessage(String message)
```

参数：

参数	类型	含义
message	String	自定义消息内容

disableSendingMessageByAdmin

禁用远端用户的发送文本消息能力（只有管理员或群主能够调用）。

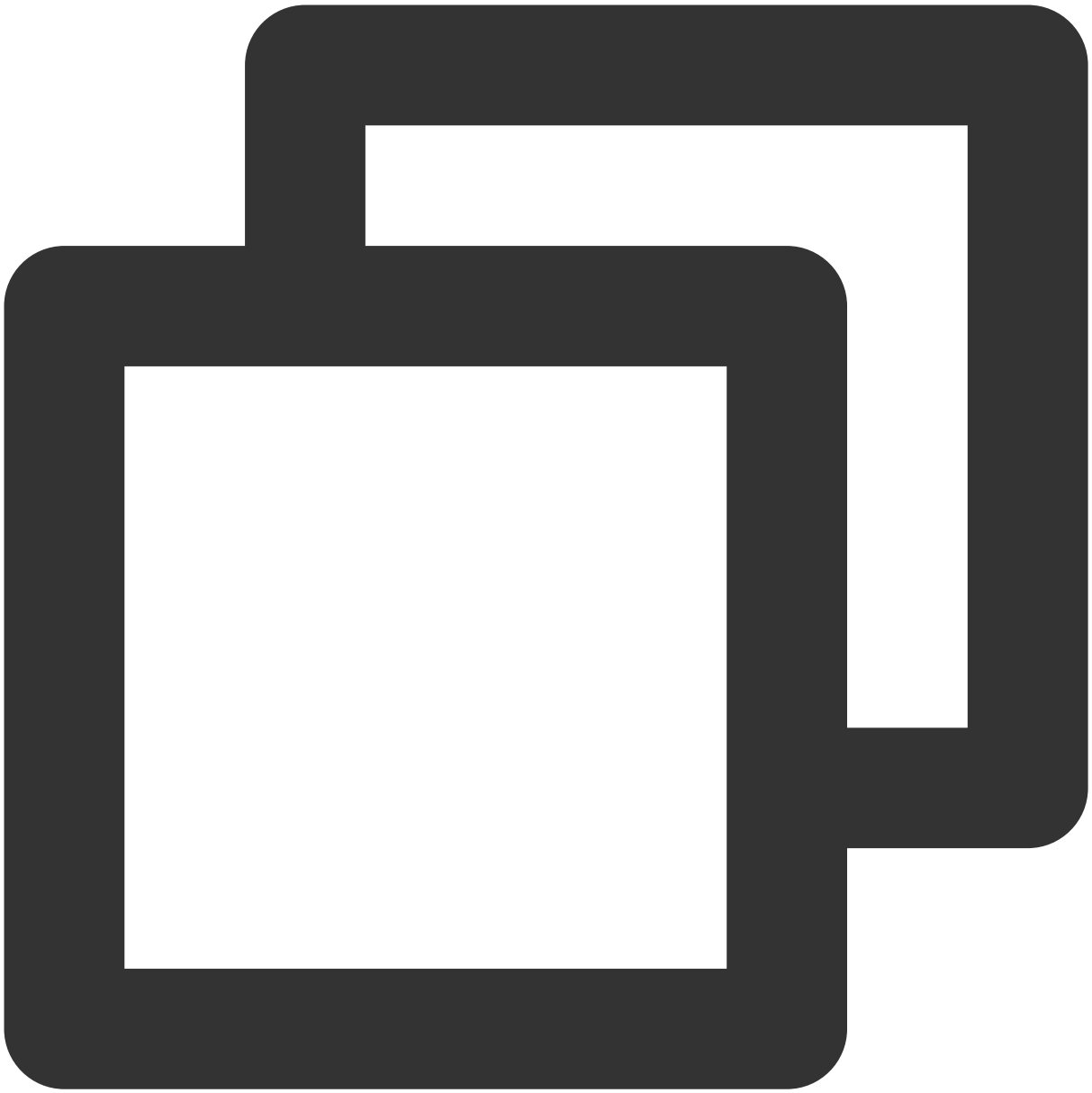


```
Future<TUIActionCallback> disableSendingMessageByAdmin(String userId,
                                                       bool isDisable)
```

参数	类型	含义
userId	String	用户 ID
isDisable	bool	是否禁用

disableSendingMessageForAllUser

禁用所有用户的发送文本消息能力（只有管理员或群主能够调用）。

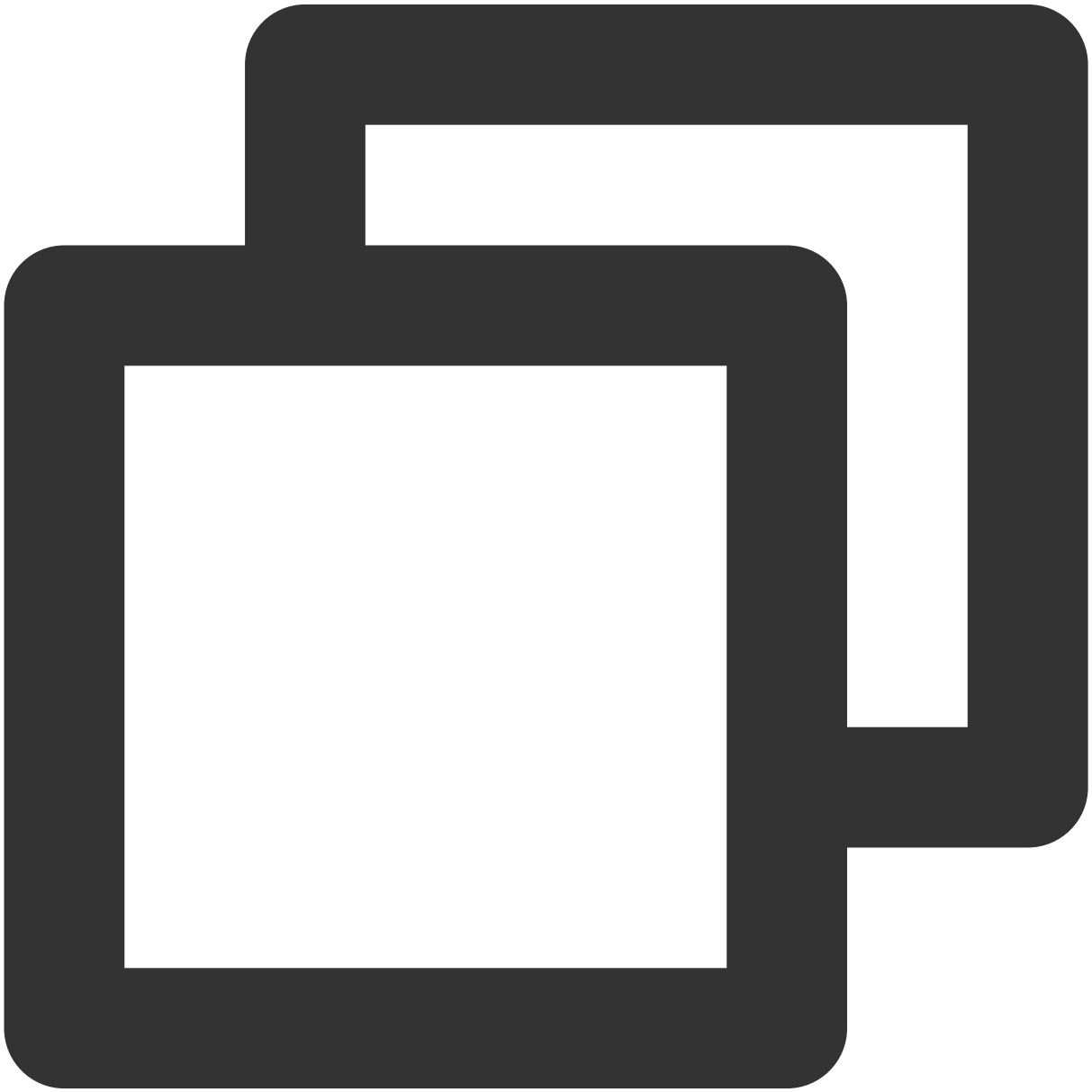


```
Future<TUIActionCallback> disableSendingMessageForAllUser(bool isDisable)
```

参数	类型	含义
isDisable	bool	是否禁用

cancelRequest

取消已经发出的请求。



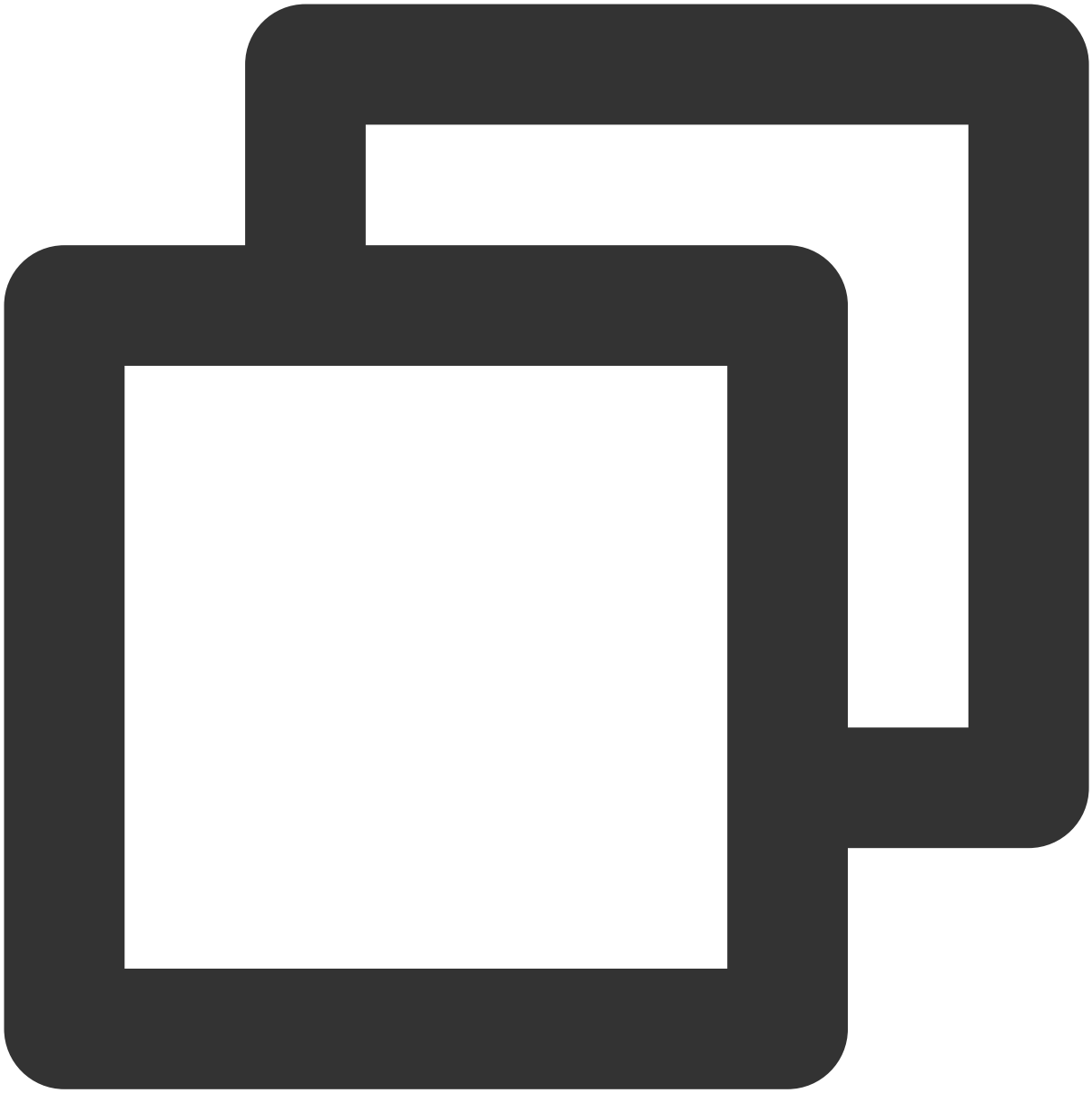
```
Future<TUIActionCallback> cancelRequest(String requestId)
```

参数：

参数	类型	含义
requestId	String	请求 ID

responseRemoteRequest

回复远端用户的请求。



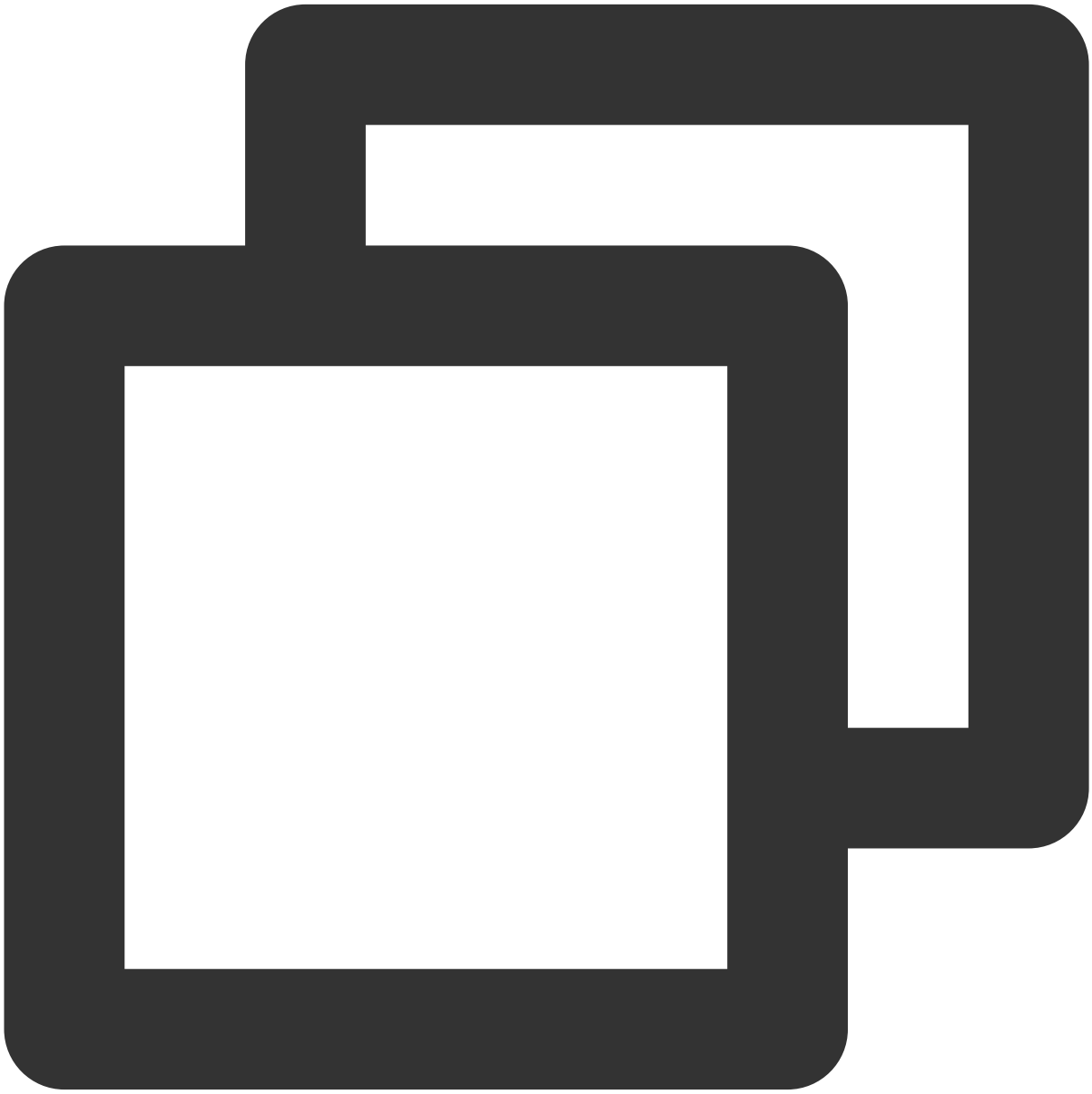
```
Future<TUIActionCallback> responseRemoteRequest(String requestId,
                                                bool agree)
```

参数：

参数	类型	含义
requestId	String	请求 ID
agree	bool	是否同意

callExperimentalAPI

调用实验性接口。

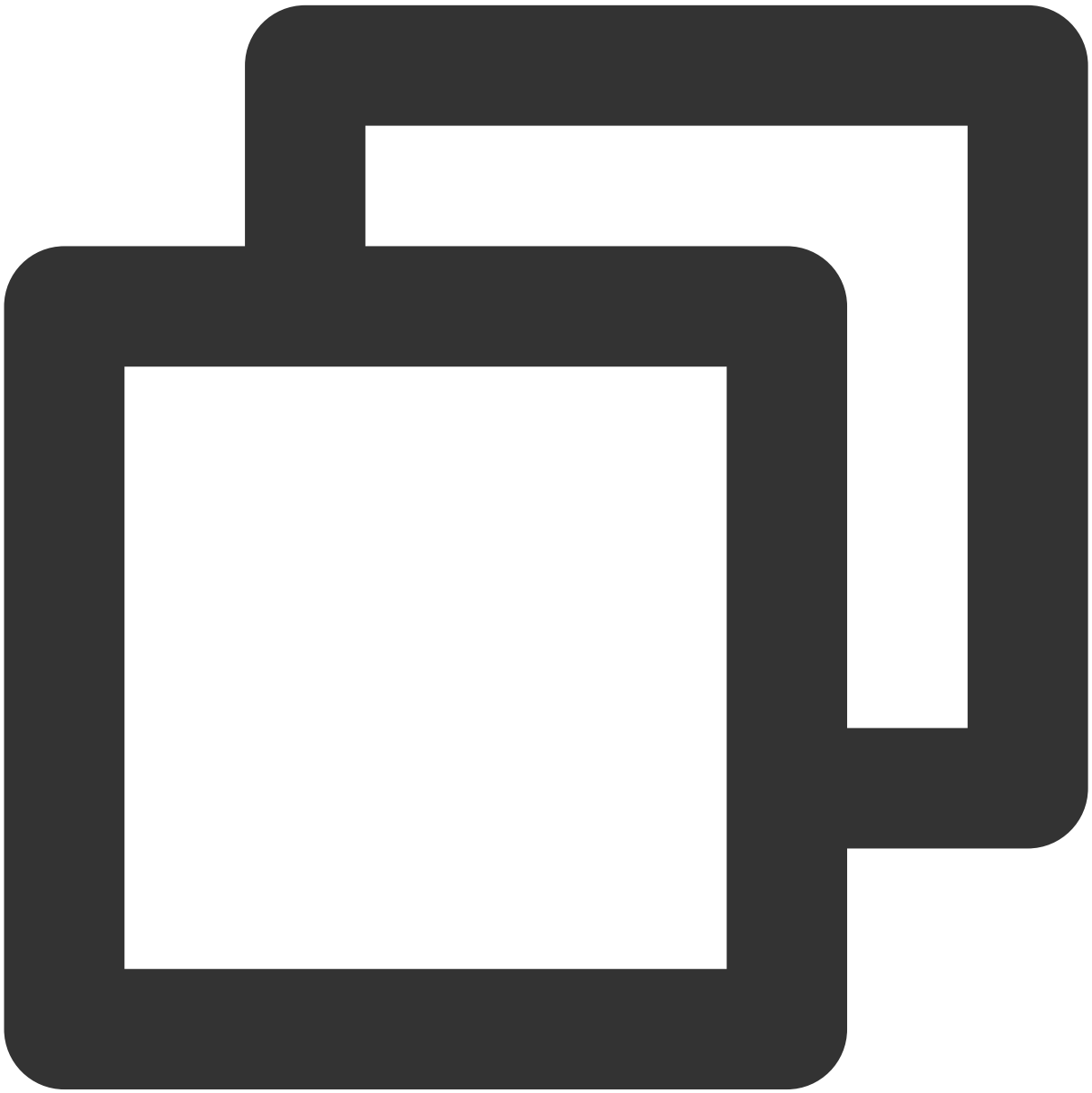


```
void callExperimentalAPI(String jsonStr);
```

参数	类型	含义
jsonStr	String	接口信息

switchCamera

切换前置或后置摄像头（仅适用于移动端）。

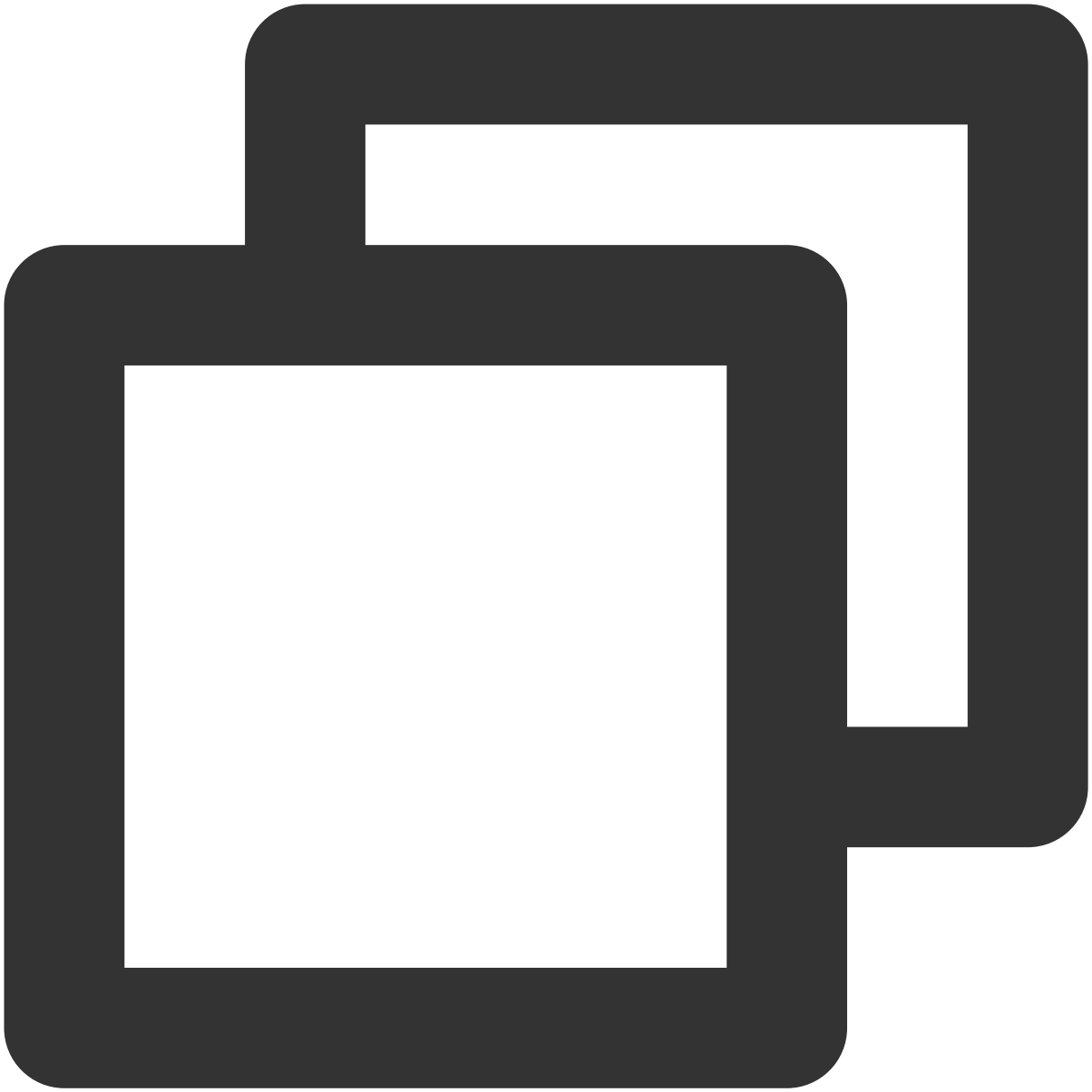


```
Future<int?> switchCamera(bool isFrontCamera);
```

参数	类型	含义
isFrontCamera	bool	是否前置

setBeautyLevel

设置美颜级别。

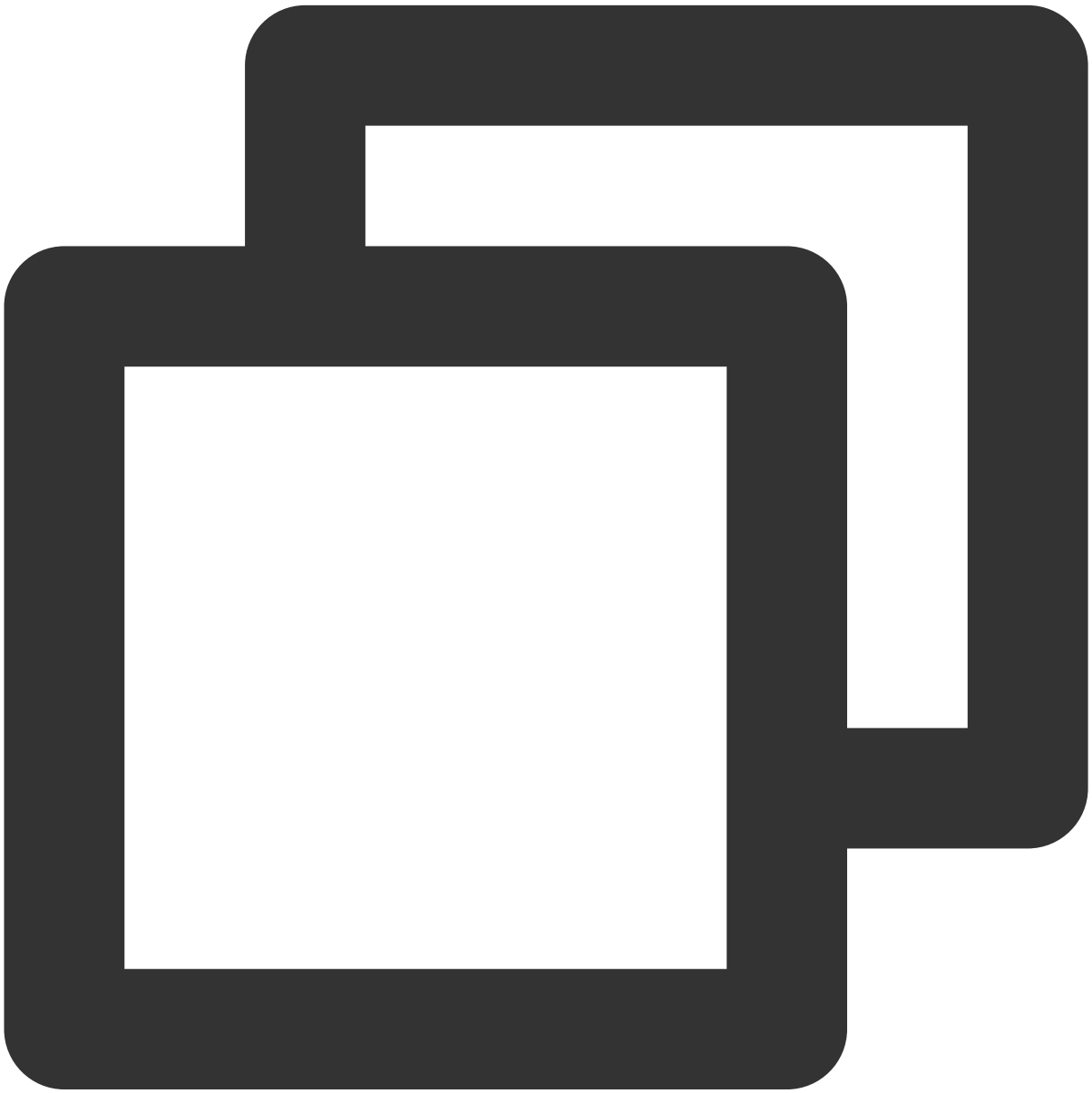


```
void setBeautyLevel(int beautyStyle, int beautyLevel);
```

参数	类型	含义
beautyStyle	int	美颜风格
beautyLevel	int	美颜级别

setWhitenessLevel

设置美白级别。



```
void setWhitenessLevel(int whitenessLevel);
```

参数	类型	含义
whitenessLevel	int	美白级别

类型定义

最近更新时间：2023-11-21 15:18:54

枚举定义

TUIRoomDefine

类型	说明
TUIRoomType	房间类型
TUISpeechMode	麦控模式
TUIMediaDevice	房间内媒体设备类型
TUIRole	房间内角色类型
TUIVideoQuality	视频质量
TUIAudioQuality	音频质量
TUIVideoStreamType	视频流类型
TUIChangeReason	更改原因（用户音视频状态变更操作原因：自己主动修改或者被房主、管理员修改）
TUICaptureSourceType	屏幕分享捕获源类型
TUIRequestAction	请求类型
TUIResolutionMode	分辨率模式（横屏 or 竖屏）

TUICommonDefine

类型	说明
TUIError	错误码
TUINetworkQuality	网络质量

常用结构

TUIRoomDefine

类型	说明

TUIRoomInfo	房间信息
TUILoginUserInfo	用户登录信息
TUIUserInfo	房间内用户信息
TUISeatInfo	房间内座位信息
TUISeatLockParams	锁定麦位操作参数
TUIUserVoiceVolume	房间内用户音量
TUIRequest	信令请求
TUIActionCallback	用户操作回调
TUIPlayCallback	视频播放回调
TUIRequestCallback	用户请求回调
TUIUserListResult	用户列表信息
TUIUserVoiceVolume	用户音量信息
TUIValueCallBack<T>	含返回值（T）回调
TUIRoomVideoEncoderParams	视频编码器参数

TUICommonDefine

类型	说明
TUINetwork	网络质量信息
TUIMessage	消息
TUIImageBuffer	图片信息

TUIRoomType

房间类型

枚举	取值	描述
conference	1	会议类型房间，适用于会议和教育场景。在该房间中，用户可以选择自由发言模式或者申请发言、上麦发言等不同模式
livingRoom	2	直播类型房间，适用于直播场景。在该房间中，用户可以自由发言，并且房间采用麦位控制模式，每个麦位都有编号

版权所有：腾讯云计算（北京）有限责任公司

TUISpeechMode

麦控模式

枚举	取值	描述
freeToSpeak	1	自由发言模式
applyToSpeak	2	申请发言模式（仅在会议类型房间下生效）
applySpeakAfterTakingSeat	3	上麦发言模式

说明：

房间类型、麦控模式与上麦（takeSeat）的关系

房间类型	麦控模式		
	freeToSpeak	applyToSpeak	applySpeakAfterTakingSeat
conference	不支持上麦	不支持上麦	需要向房主/管理员申请（takeSeat），待同意后方可开麦克风/摄像头发言
livingRoom	可以自由上麦	不支持上麦	需要向房主/管理员申请（takeSeat），待同意后方可开麦克风/摄像头发言

TUIMediaDevice

房间内媒体设备类型

枚举	取值	描述
microphone	1	麦克风
camera	2	摄像头
screen	3	屏幕共享

TUIRole

房间内角色类型

枚举	取值	描述
roomOwner	0	房主。一般指房间的创建者，房间内最高权限拥有者
administrator	1	房间管理员
generalUser	2	房间内普通成员

TUIVideoQuality

视频质量

枚举	取值	描述
videoQuality_360P	1	低清 360P
videoQuality_540P	2	标清 540P
videoQuality_720P	3	高清 720P
videoQuality_1080P	4	超清 1080P

TUIAudioQuality

音频质量

枚举	取值	描述
audioProfileSpeech	0	人声模式
audioProfileDefault	1	默认模式
audioProfileMusic	2	音乐模式

TUIVideoStreamType

视频流类型

枚举	取值	描述
cameraStream	0	高清摄像头视频流
screenStream	1	屏幕分享视频流
cameraStreamLow	2	低清摄像头视频流

TUIChangeReason

版权所有：腾讯云计算（北京）有限责任公司

更改原因（用户音视频状态变更操作原因: 自己主动修改 或者 被房主、管理员修改）

枚举	取值	描述
changedBySelf	0	自己操作
changedByAdmin	1	房主或管理员操作

TUICaptureSourceType

屏幕分享捕获源类型

枚举	取值	描述
unknown	-1	未定义
window	0	窗口
screen	1	屏幕

TUIRequestAction

请求类型

枚举	取值	描述
invalidAction	0	无效请求
requestToOpenRemoteCamera	1	请求远端用户打开摄像头
requestToOpenRemoteMicrophone	2	请求远端用户打开麦克风
requestToConnectOtherRoom	3	请求连接到其他房间
requestToTakeSeat	4	请求上麦
requestRemoteUserOnSeat	5	请求远端用户上麦
applyToAdminToOpenLocalCamera	6	向管理员请求打开本地摄像头
applyToAdminToOpenLocalMicrophone	7	向管理员请求打开本地麦克风

TUIResolutionMode

枚举	取值	描述
landscape	0	横屏
portrait	1	竖屏

版权所有：腾讯云计算（北京）有限责任公司

TUIError

错误码

枚举	取值	描述
success	0	操作成功
errFailed	-1	暂未归类的通用错误
errFreqLimit	-2	请求被限频，请稍后重试
errRepeatOperation	-3	重复操作，请检查您的接口调用是否重复
errSDKAppIDNotFound	-1000	未找到 SDKAppID，请在 腾讯云视立方 SDK 控制台 确认应用信息
errInvalidParameter	-1001	调用 API 时，传入的参数不合法
errSdkNotInitialized	-1002	SDK 未初始化
errPermissionDenied	-1003	无操作权限
errRequirePayment	-1004	该功能需要开通额外的套餐
errCameraStartFailed	-1100	打开摄像头失败
errCameraNotAuthorized	-1101	摄像头未授权
errCameraOccupy	-1102	摄像头被占用
errCameraDeviceEmpty	-1103	当前无摄像头设备
errMicrophoneStartFailed	-1104	麦克风打开失败
errMicrophoneNotAuthorized	-1105	麦克风未授权
errMicrophoneOccupy	-1106	麦克风被占用
errMicrophoneDeviceEmpty	-1107	当前无麦克风设备
errGetScreenSharingTargetFailed	-1108	获取屏幕分享对象失败
errStartScreenSharingFailed	-1109	开启屏幕分享失败
errRoomIdNotExist	-2100	进房时房间不存在，或许已被解散
errOperationInvalidBeforeEnterRoom	-2101	需要进房后才可使用此功能

errExitNotSupportedForRoomOwner	-2102	房主不支持退房操作，Conference（会议）房间类型：可以先转让房主，再退房。LivingRoom（直播）房间类型：房主只能解散房间
errOperationNotSupportedInCurrentRoomType	-2103	当前房间类型下不支持该操作
errOperationNotSupportedInCurrentSpeechMode	-2104	当前发言模式下不支持该操作
errRoomIdInvalid	-2105	创建房间 ID 非法，自定义 ID 必须为可打印 ASCII 字符（0x20-0x7e），最长48个字节
errRoomIdOccupied	-2106	房间 ID 已被使用，请选择别的房间 ID
errRoomNameInvalid	-2107	房间名称非法，名称最长30字节，字符编码必须是 UTF-8，如果包含中文
errAlreadyInOtherRoom	-2108	当前用户已在别的房间内，需要先退房才能加入新的房间： 单个 roomEngine 实例只支持用户进入一个房间，如果要进入不同的房间请先退房或者使用新的 roomEngine 实例
errUserNotExist	-2200	用户不存在
errUserNotEntered	-2201	用户不在当前房间内
errUserNeedOwnerPermission	-2300	需要房主权限才能操作
errUserNeedAdminPermission	-2301	需要房主或者管理员权限才能操作
errRequestNoPermission	-2310	信令请求无权限，例如取消非自己发起的邀请
errRequestIdInvalid	-2311	信令请求 ID 无效或已经被处理过
errMaxSeatCountLimit	-2340	最大麦位超出套餐包数量限制
errAlreadyInSeat	-2341	当前用户已经在麦位上
errSeatOccupied	-2342	当前麦位已经有人了
errSeatLocked	-2343	当前麦位被锁
errSeatIndexNotExist	-2344	麦位编号不存在
errUserNotInSeat	-2345	当前用户没有在麦上
errAllSeatOccupied	-2346	上麦人数已满

版权所有：腾讯云计算（北京）有限责任公司

errOpenMicrophoneNeedSeatUnlock	-2360	当前麦位音频被锁
errOpenMicrophoneNeedPermissionFromAdmin	-2361	需要向房主或管理员申请后打开麦克风
errOpenCameraNeedSeatUnlock	-2370	当前麦位视频被锁, 需要由房主解锁麦位后，才能打开摄像头
errOpenCameraNeedPermissionFromAdmin	-2371	需要向房主或管理员申请后打开摄像头
errSendMessageDisabledForAll	-2380	当前房间已开启全员禁言
errSendMessageDisabledForCurrent	-2381	当前房间内，您已被禁言

TUINetworkQuality

网络质量

枚举	取值	描述
qualityUnknown	0	未定义
qualityExcellent	1	当前网络非常好
qualityGood	2	当前网络比较好
qualityPoor	3	当前网络一般
qualityBad	4	当前网络较差
qualityVeryBad	5	当前网络很差
qualityDown	6	当前网络不满足 TRTC 的最低要求

TUIRoomInfo

房间信息

字段	类型	描述
roomId	String	房间 ID
roomType	TUIRoomType	房间类型
ownerId	String	主持人 ID。默认为房间创建者（只读）
name	String	房间名称。默认为房间 ID
speechMode	TUISpeechMode	房间发言模式

createTime	int	房间创建时间（只读）
memberCount	int	房间内成员数量（只读）
maxSeatCount	int	最大麦位数量（仅支持进房前和创建房间时设置）
isCameraDisableForAllUser	bool	是否禁止打开摄像头（创建房间可选参数）。默认值：false
isMicrophoneDisableForAllUser	bool	是否禁止打开麦克风（创建房间可选参数）。默认值：false
isMessageDisableForAllUser	bool	是否禁止发送消息（创建房间可选参数）。默认值：false
enableCDNStreaming	bool	是否开启 CDN 直播（创建房间可选参数，直播房间使用）。默认值：false
cdnStreamDomain	String	直播推流域名（创建房间可选参数，直播房间使用）。默认值：空

TUILoginUserInfo

用户登录信息

字段	类型	含义
userId	String	用户 ID
userName	String	用户名称
avatarUrl	String	用户头像 URL
customInfo	Map<String, String>	自定义信息

TUIUserInfo

房间内用户信息

字段	类型	描述
userId	String	用户 ID
userName	String	用户名称
avatarUrl	String	用户头像 URL
userRole	TUIRole	用户角色类型

hasAudioStream	bool	是否有音频流。默认值：false
hasVideoStream	bool	是否有视频流。默认值：false
hasScreenStream	bool	是否有屏幕分享流。默认值：false

TUISeatInfo

房间内座位信息

字段	类型	描述
index	int	麦位序号
userId	String	用户 ID
isLocked	bool	麦位是否被锁定。默认 false
isVideoLocked	bool	麦位是否被禁止打开摄像头。默认 false
isAudioLocked	bool	麦位是否被禁止打开麦克风。默认 false

TUISeatLockParams

锁定麦位操作参数

字段	类型	含义
lockSeat	bool	锁定麦位。默认 false
lockVideo	bool	锁定麦位摄像头。默认 false
lockAudio	bool	锁定麦位麦克风。默认 false

TUIUserVoiceVolume

房间内用户音量

字段	类型	描述
userId	String	用户 ID
volume	int	音量。用于承载所有正在说话的用户的音量大小，取值范围 0 - 100

TUIRequest

信令请求

字段	类型	描述
requestId	String	请求 ID
requestAction	TUIRequestAction	请求类型
userId	String	用户 ID
content	String	信令内容
timestamp	int	时间戳

TUIActionCallback

字段	类型	描述
code	TUIError	错误码
message	String?	错误信息

TUIPlayCallback

字段	类型	描述
onPlaying	(String userId) {}	正在播放回调
onLoading	(String userId) {}	正在加载回调
onPlayError	(String userId, TUIError code, String message) {}	播放错误回调

TUIRequestCallback

字段	类型	描述
onAccepted	(String requestId, String userId) {}	请求被接受回调
onRejected	(String requestId, String userId, String message) {}	请求被拒绝回调
onCancelled	(String requestId, String userId) {}	请求取消回调
onTimeout	(String requestId, String userId) {}	请求超时回调
onError	(String requestId, String userId, TUIError error, String message) {}	请求错误回调

版权所有：腾讯云计算（北京）有限责任公司

TUIUserListResult

字段	类型	描述
nextSequence	int	分页拉取标志，若返回的 nextSequence 不为零，需要使用返回的 nextSequence 再次拉取，直至返回为0
userInfoList	List<TUIUserInfo>	本次调用返回的用户列表

TUIUserVoiceVolume

字段	类型	描述
userId	String	用户 ID
volume	int	用户音量大小

TUIValueCallBack<T>

字段	类型	描述
code	TUIError	错误码
message	String?	错误信息
data	T?	返回数据，示例：若 T 为 TUIUserInfo，则 TUIValueCallBack<TUIUserInfo> 的 data 字段类型为 TUIUserInfo

TUIRoomVideoEncoderParams

字段	类型	描述
videoResolution	TUIVideoQuality	视频质量
resolutionMode	TUIResolutionMode	分辨率模式
fps	int	视频采集帧率
bitrate	int	目标视频码率

TUINetwork

网络质量信息

字段	类型	描述

userId	String	用户 ID
quality	TUINetworkQuality	网络质量
upLoss	int	上行丢包率
downLoss	int	下行丢包率
delay	int	网络延迟

TUIMessage

消息

字段	类型	描述
messageId	String	消息 ID
message	String	消息文本
timestamp	int	消息时间
userId	String	消息发送者
userName	String	消息发送者昵称
avatarUrl	String	消息发送者头像

TUIImageBuffer

图片信息

字段	类型	描述
buffer	String	图片数据缓存地址
length	int	长度
width	int	宽度
height	int	高度

版权所有：腾讯云计算（北京）有限责任公司

常见问题

Web

最近更新时间：2023-10-30 11:03:10

环境相关问题

TUIRoomKit Web 支持的平台有哪些？

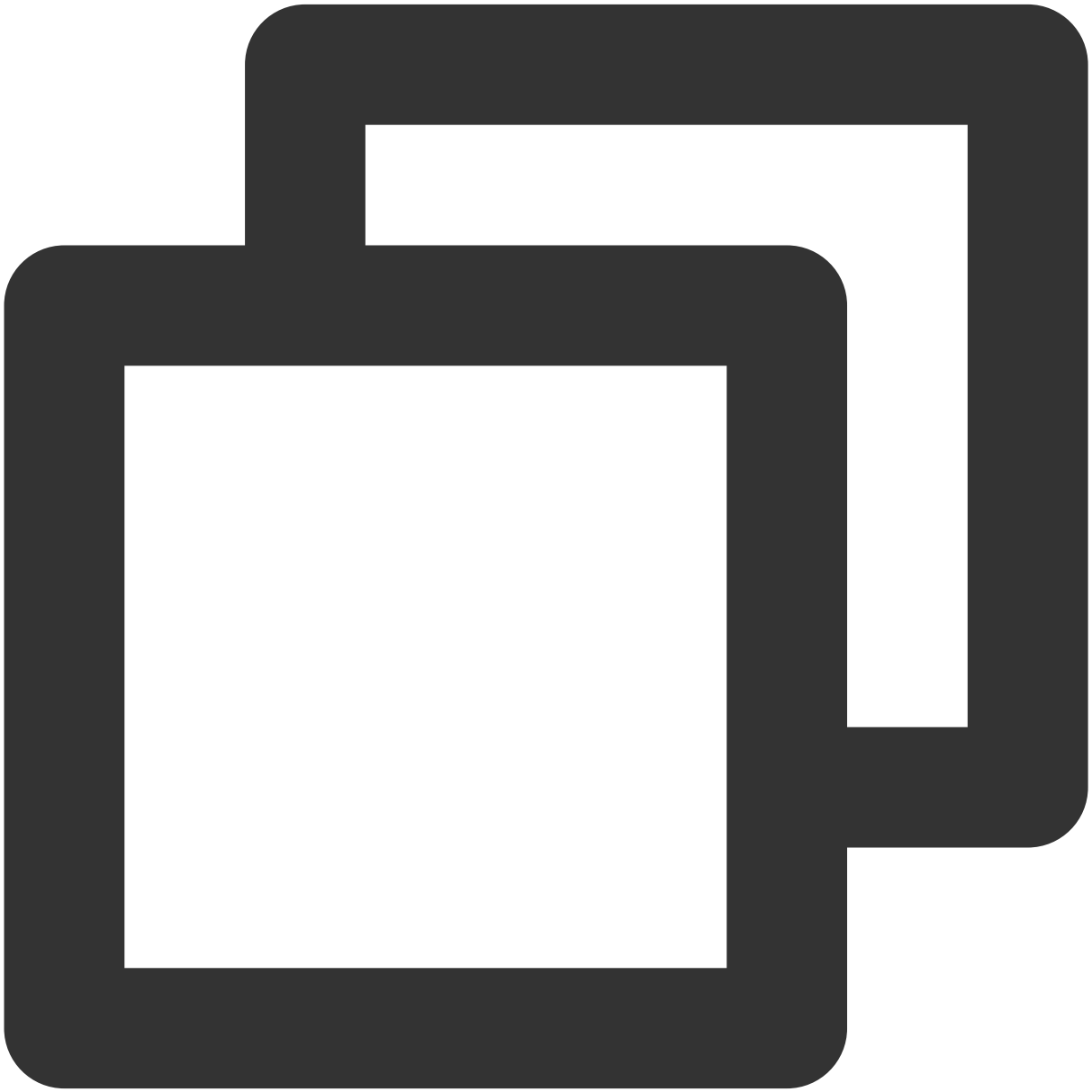
TUIRoomEngine Web 支持的平台请参考：TRTC Web SDK 对浏览器的支持情况。
对于上述没有列出的环境，您可以在当前浏览器打开 TRTC 能力测试 测试是否完整的支持 WebRTC 的功能。

为什么本地开发测试能正常使用 TUIRoomKit, 但是部署到线上无法使用？

出于对用户安全、隐私等问题的考虑，浏览器限制网页只有在安全的环境下（例如 `https` 、`localhost` 、 `file://` 等协议），才能采集麦克风、摄像头。HTTP 协议是不安全的，浏览器会禁止在 HTTP 协议下采集媒体设备。
若您在本地开发测试一切正常，但是页面部署后，却无法正常采集摄像头、麦克风。则请检查您的网页是否部署到了 HTTP 协议上，若是，请使用 HTTPS 部署您的网页，并确保具备合格的 HTTPS 安全证书。
更多详情请参见 URL域名及协议限制说明。

TUIRoomKit Web 端是否支持使用 iframe 集成？

支持的。在 iframe 中集成 TUIRoom Web 需要给 iframe 标签增加属性来开启相关权限，具体参考如下。

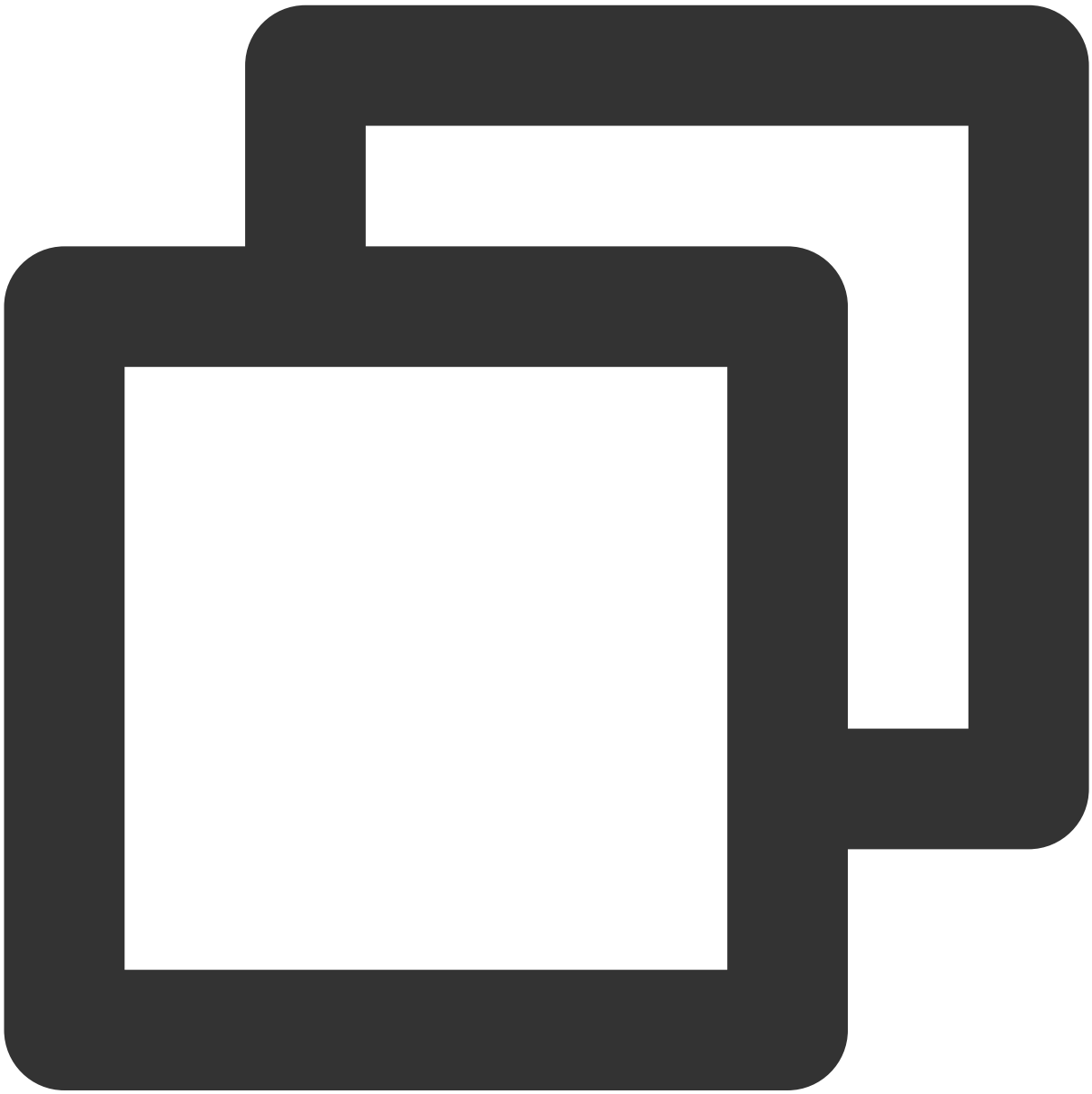


```
// 开启麦克风、摄像头、屏幕分享、全屏权限
<iframe allow="microphone; camera; display-capture; display; fullscreen;">
```

编译相关问题

Webpack 5 引入 TUIRoomEngine sdk 报错：webpack < 5 used to include polyfills for node.js core modules by default. This is no longer the case. Verify if you need this module and configure a polyfill for it.

报错原因是由于在 webpack5 中移除了 nodejs 核心模块的 polyfill 自动引入，你可在 vue.config.js 中添加 configureWebpack 配置解决该问题。



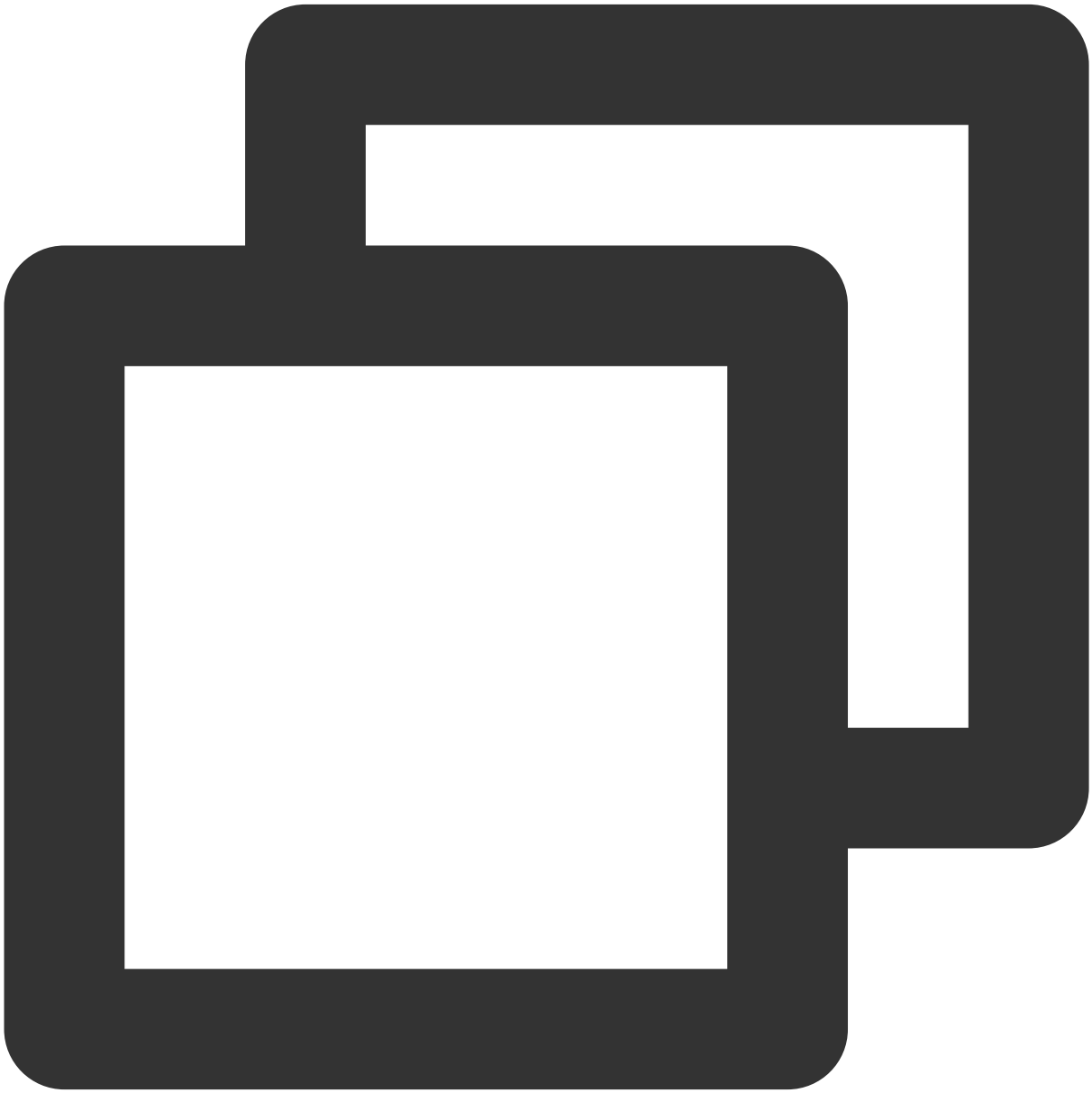
```
module.exports = defineConfig({
  // ...
  configureWebpack: (config) => {
    config.resolve.fallback = {
      ...config.resolve.fallback,
      url: false,
      path: false,
      fs: false,
```

```
        crypto: false,
      };
    }
  });
```

功能相关问题

项目中可以同时引入 tim-js-sdk 和 @tencentcloud/tuiroom-engine-js 吗？

可以的。你可以通过 tim-js-sdk 中的 TIM 创建 tim 实例对象，在 TUIRoomEngine 初始化时传入 init 接口。



```
await TUIRoomEngine.init({
 sdkAppId: 0,    // 填写您申请的 sdkAppId
 userId: '',     // 填写您业务对应的 userId
 userSig: '',    // 填写服务器或本地计算的 userSig
 tim,     // 传入 tim 实例
});
```

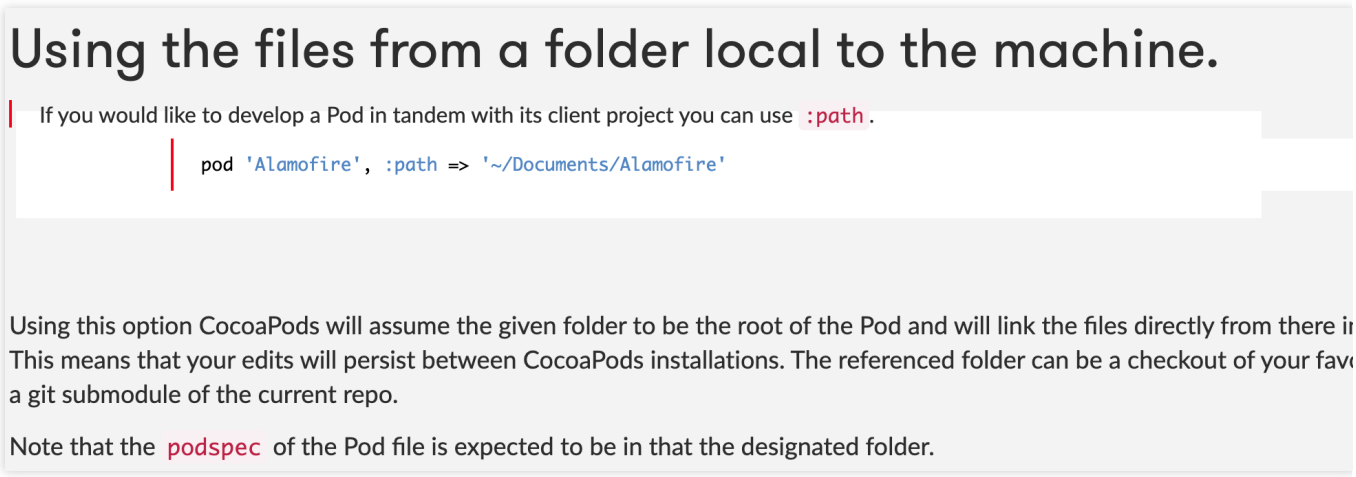
同时，你也可以通过 roomEngine.getTIM 方法获取 TUIRoomEngine 内部使用的 tim 实例。

iOS

最近更新时间：2023-10-30 14:10:49

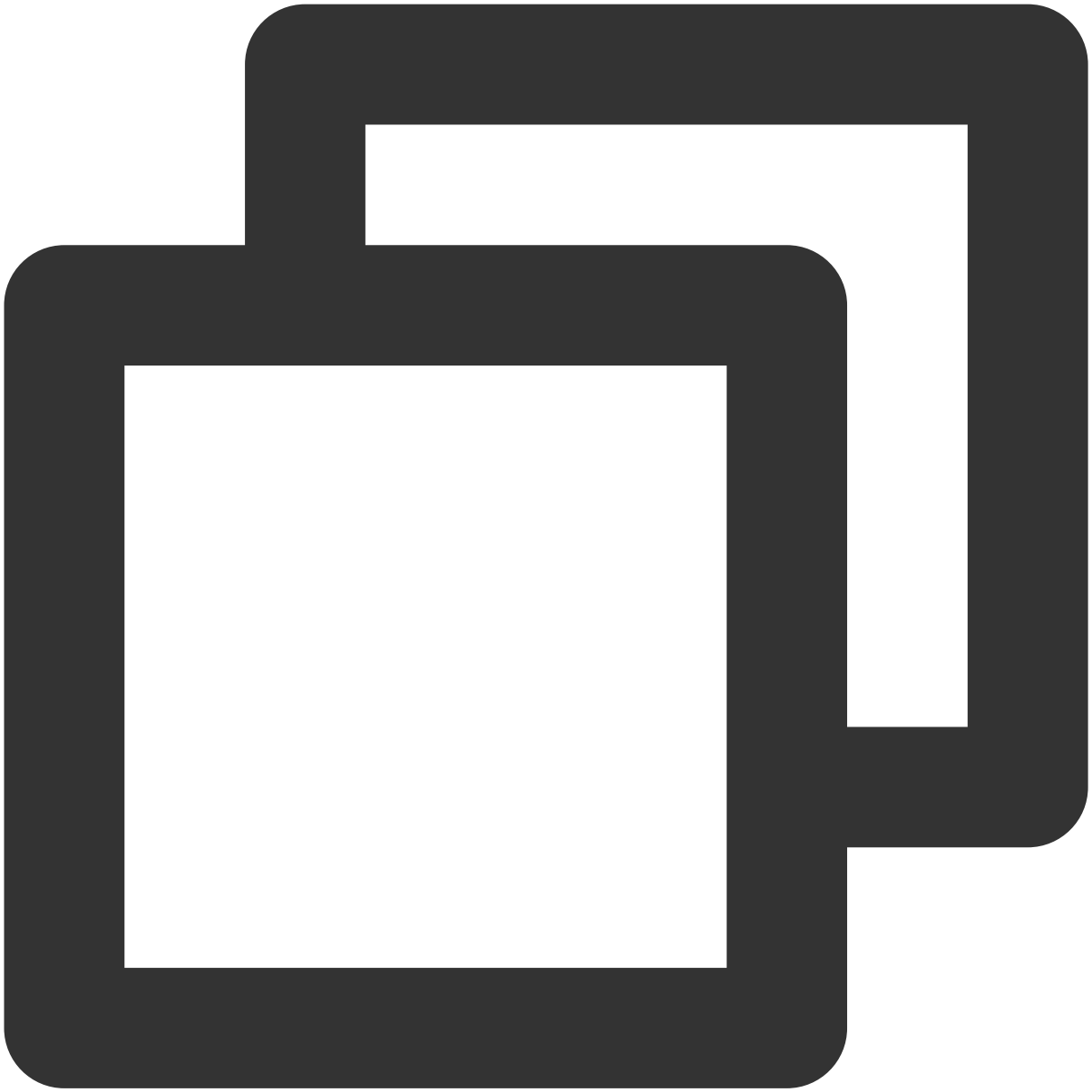
我需要自己修改 UI，每次更新 pod 后就会刷新源码，导致修改丢失，该如何处理？

建议Fork TUIRoomKit 仓库 到您个人github账号下，然后采用 本地 pod 或者 git pod 路径 的方式在您的工程引用对应代码。
详情参考Pod 官网文档：



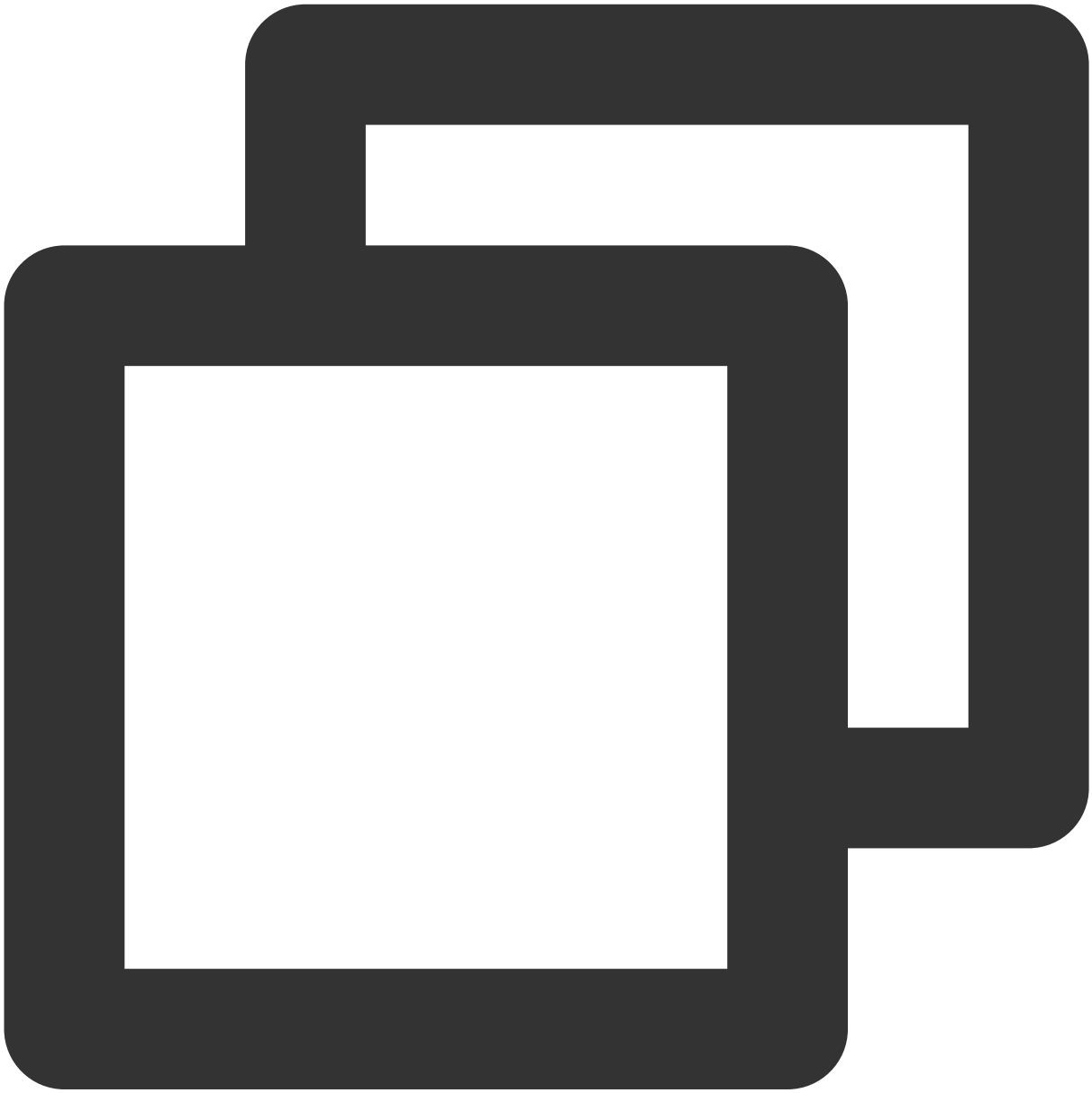
TUIRoomKit 和自己集成的音视频库冲突了？

腾讯云的 音视频库 不能同时集成，可能存在符号冲突，可以按照下面的场景处理。
1. 如果您使用了 TXLiteAVSDK_TRTC 库，不会发生符号冲突。可直接在 Podfile 文件中添加依赖，



```
pod 'TUIRoomKit'
```

2. 如果您使用了 `TXLiteAVSDK_Professional` 库，会产生符号冲突。您可在 `Podfile` 文件中添加依赖，



```
pod 'TUIRoomKit/Professional'
```

3. 如果您使用了 `TXLiteAVSDK_Enterprise` 库，会产生符号冲突。建议升级到 `TXLiteAVSDK_Professional` 后使用 `TUIRoomKit/Professional` 。

如何查看 TRTC 日志？

TRTC 的日志默认压缩加密，后缀为 .xlog。日志是否加密是可以通过 setLogCompressEnabled 来控制，生成的文件名里面含 C(compressed) 的就是加密压缩的，含 R(raw) 的就是明文的。

iOS&Mac： `sandbox的Documents/log`

Android：

6.7及之前的版本：`/sdcard/log/tencent/liteav`

6.8之后的版本：`/sdcard/Android/data/包名/files/log/tencent/liteav/`

8.5之后的版本：`/sdcard/Android/data/包名/files/log/liteav/`

Windows：

8.8之前的版本：`%appdata%/tencent/liteav/log`

8.8以及8.8之后的版本：`%appdata%/liteav/log`

Web：打开浏览器控制台，或使用 vConsole 记录 SDK 打印信息

说明：

查看 .xlog 文件需要下载 解密工具，在 Python 2.7环境中放到 xlog 文件同目录下直接使用 `python decode_mars_log_file.py` 运行即可。

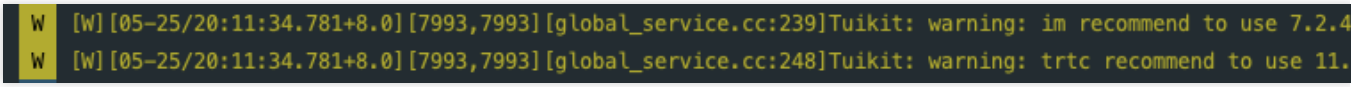
查看 .clog 文件（9.6 版本以后新的日志格式）需要下载 解密工具，在 Python 2.7 环境中放到 clog 文件同目录下直接使用 `python decompress_clog.py` 运行即可。

Android

最近更新时间：2023-10-30 11:06:54

TUIRoomKit 对依赖的 TRTC 和 IM SDK 有什么要求吗？

TUIRoomEngine在登录时会进行版本校验，提示您使用正确的 `TRTC SDK` 与 `IM SDK` 版本



TUIRoomKit 可以去掉 IM SDK 依赖吗？

不可以，多人音视频房间需要 IM 去做信令相关的互通，包括麦控，弹幕等功能，需要有 IM SDK 的依赖。

TUIRoomKit 是否支持会议预定等功能？

目前尚不支持，该功能在规划中，您可以关注我们的更新日志来第一时间获取该功能的上线情况。

集成了 TUIRoomKit 后，如何快速修改用户名和用户头像？

通过调用 `TUIRoomKit` 的 `setSelfInfo` 接口设置。

Electron

最近更新时间：2023-10-30 11:07:40

环境相关问题

trtc-electron-sdk 是否兼容官方 Electron v12.0.1 版本?

兼容的，trtc-electron-sdk 没有特别依赖 electron 自身的 SDK，所以没有相关的版本依赖。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2023-11-21 14:54:58

iOS上运行时报错：Error (Xcode): Library not found for -ld64？

`Xcode` 15 换了编译的 `linker`， `RoomEngine Flutter` 在1.6.0做了适配，老版本的xcode无法编译，使用最新的 `Xcode` 即可。

工程同时集成了tencent_calls_uikit 和 rtc_room_engine ，会出现加入room的房间会打断call的通话等两个组件互相干扰的问题，如何解决？

这是由于 `RTC` 业务的独占性导致的，我们推荐您使用两个标志位 `callState` 、 `roomState` 分别用来记录当前是否在通话中，是否在会议中，通过这两个标志位来实现二者业务的隔离，来避免此问题的发生。例如，如果当前正在通话中，则禁止进入房间内。

错误码

最近更新时间：2023-10-30 11:08:32

通用错误码

错误码	描述
0	操作成功
-1	暂未归类的通用错误
-2	请求被限频，请稍后重试
-1000	未找到 SDKAppID，请在 腾讯云视立方 SDK 控制台 确认应用信息
-1001	调用 API 时，传入的参数不合法，检查入参是否合法
-1002	未登录,请调用 Login 接口
-1003	获取权限失败，当前未授权音/视频权限，请查看是否开启设备权限
-1004	该功能需要开通额外的套餐，请在 腾讯云视立方 SDK 控制台 按需开通对应套餐

本地用户渲染，视频管理，音频管理 API 回调错误定义

错误码	描述
-1100	系统问题，打开摄像头失败。检查摄像头设备是否正常
-1101	摄像头没有系统授权, 检查系统授权
-1102	摄像头被占用，检查是否有其他进程使用摄像头
-1103	当前无摄像头设备，请插入摄像头设备解决该问题
-1104	系统问题，打开麦克风失败。检查麦克风设备是否正常
-1105	麦克风没有系统授权，检查系统授权
-1106	麦克风被占用
-1107	当前无麦克风设备
-1108	获取屏幕分享对象失败，检查屏幕录制权限

版权所有：腾讯云计算（北京）有限责任公司

-1109	开启屏幕分享失败，检查房间内是否有人正在屏幕分享

房间管理相关 API 回调错误定义

错误码	描述
-2100	进房时房间不存在，或许已被解散
-2101	需要进房后才可使用此功能
-2102	房主不支持退房操作，Conference（会议）房间类型: 可以先转让房主，再退房。LivingRoom（直播）房间类型: 房主只能解散房间
-2103	当前房间类型下不支持该操作
-2104	当前发言模式下不支持该操作
-2105	创建房间 ID 非法，自定义 ID 必须为可打印 ASCII 字符（0x20-0x7e），最长48个字节
-2106	房间 ID 已被使用，请选择别的房间 ID
-2107	房间名称非法，名称最长30字节，字符编码必须是 UTF-8 ，如果包含中文
-2108	当前用户已在别的房间内，单个 roomEngine 实例只支持用户进入一个房间，如果要进入不同的房间请先退房或者使用新的 roomEngine 实例

房间内用户信息 API 回调错误定义

错误码	描述
-2200	未找到该用户
-2201	房间内未找到该用户

房间内用户发言管理 API 回调错误定义&房间内麦位管理 API 回调错误定义

错误码	描述
-2300	需要房主权限才能操作
-2301	需要房主或者管理员权限才能操作
-2310	信令请求无权限，比如取消非自己发起的邀请。
-2311	信令请求 ID 无效或已经被处理过。
-2340	最大麦位超出套餐包数量限制

版权所有：腾讯云计算（北京）有限责任公司

-2341	当前用户已经在麦位上
-2342	当前麦位已经有人了
-2343	当前麦位被锁
-2344	麦位编号不存在
-2345	当前用户没有在麦上
-2346	上麦人数已满
-2360	当前麦位音频被锁
-2361	需要向房主或管理员申请后打开麦克风
-2370	当前麦位视频被锁, 需要由房主解锁麦位后，才能打开摄像头
-2371	需要向房主或管理员申请后打开摄像头
-2380	当前房间已开启全员禁言
-2381	当前房间内，您已被已禁言

版权所有：腾讯云计算（北京）有限责任公司

语音互动直播
集成 TUIVoiceRoom (Android)

最近更新时间：2022-09-06 14:19:19

组件介绍

TUIVoiceRoom 是一个开源的音视频 UI 组件，通过在项目中集成 TUIVoiceRoom 组件，您只需要编写几行代码就可以为您的 App 添加“多人语音聊天”等场景，TUIVoiceRoom 支持 iOS等平台，基本功能如下图所示：

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。



组件集成

步骤一：下载并导入 TUIVoiceRoom 组件

单击进入 Github，选择克隆/下载代码，然后拷贝 Android/Source 目录到您的工程中，并完成如下导入动作：

- 在 `setting.gradle` 中完成导入，参考如下：

```
include ':Source'
```

- 在 app 的 build.gradle 文件中添加对 Source 的依赖：

```
api project(':Source')
```

- 在根目录的 `build.gradle` 文件中添加 `TRTC SDK` 和 `IM SDK` 的依赖：

```
ext {
liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

步骤二：配置权限及混淆规则

在 AndroidManifest.xml 中配置 App 的权限，SDK 需要以下权限（6.0以上的 Android 系统需要动态申请麦克风权限等）：

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

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

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

步骤三：初始化并登录

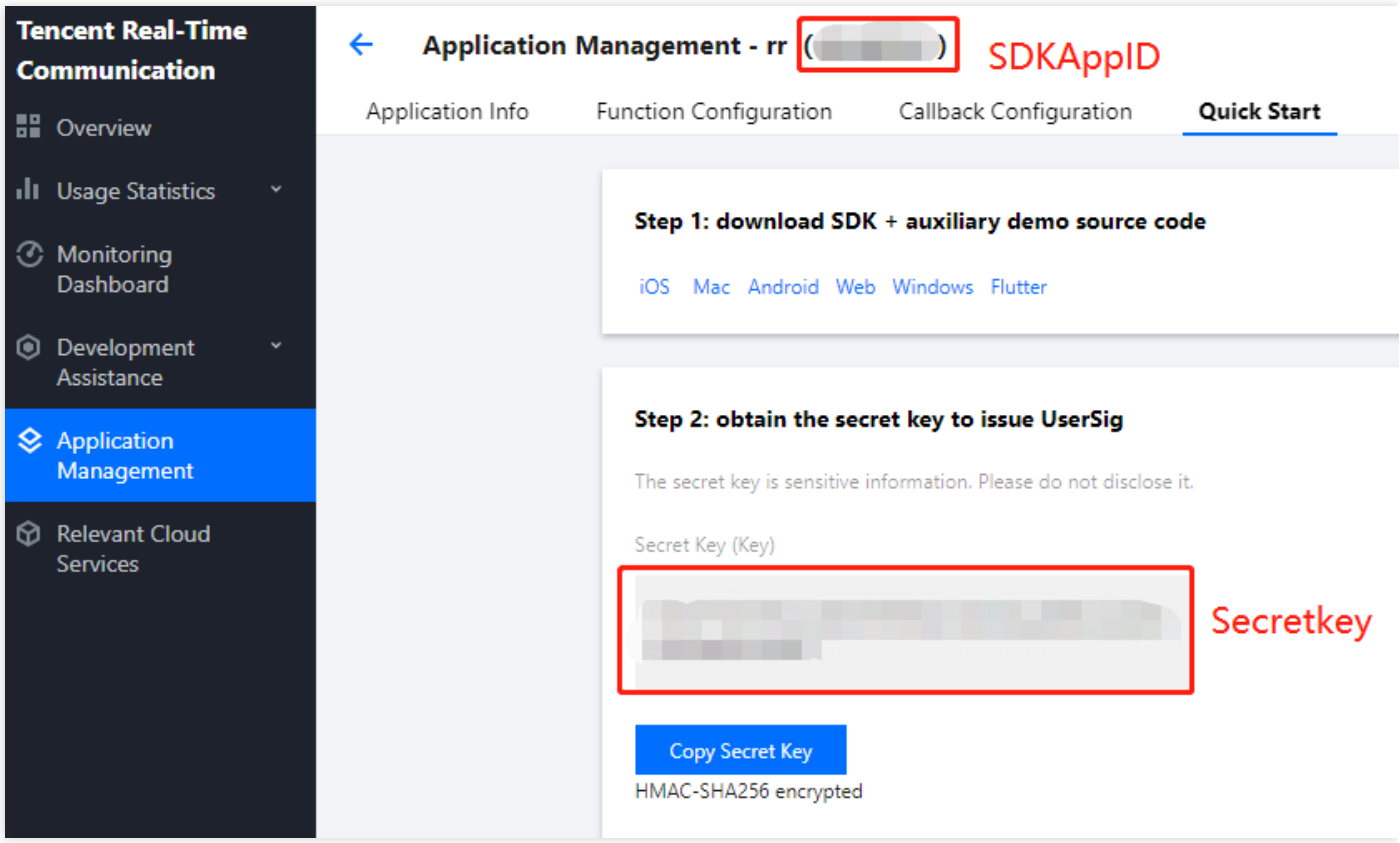
```
// 1.初始化,
TRTCVoiceRoom mTRTCVoiceRoom = TRTCVoiceRoom.sharedInstance(this);
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
```

版权所有：腾讯云计算（北京）有限责任公司

```
);
// 2.登录,
mTRTCVoiceRoom.login(SDKAppID, userId, userSig, new TRTCVoiceRoomCallback.ActionC
allback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//登录成功
}
}
});
```

参数说明：

- **SDKAppID**：**TRTC 应用 ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



- **Secretkey**：**TRTC 应用密钥**和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。建议结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的userSig，更多信息见 如何计算及使用 UserSig。

步骤四：实现语音聊天房间

1. 实现房主创建语音聊天房间 TRTCVoiceRoom#createRoom

```
// 1.房主调用创建房间
int roomId = 12345; //房间id
final TRTCVoiceRoomDef.RoomParam roomParam = new TRTCVoiceRoomDef.RoomParam();
roomParam.roomName = "房间名称";
roomParam.needRequest = false; // 上麦是否需要房主确认
roomParam.seatCount = 7; // 房间座位数，这里一共7个座位，房主占了一个后听众剩下6个座位
roomParam.coverUrl = "房间封面图的 URL 地址";
mTRTCVoiceRoom.createRoom(roomId, roomParam, new TRTCVoiceRoomCallback.ActionCa
llback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//创建成功
}
}
});
```

2. 实现听众加入语音聊天房间 TRTCVoiceRoom#enterRoom

```
// 1.听众调用加入房间
mTRTCVoiceRoom.enterRoom(roomId, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//进房成功
}
}
});
```

3. 实现听众主动上麦 TRTCVoiceRoom#enterSeat

```
// 1: 听众调用上麦
int seatIndex = 2; //麦位的index
mTRTCVoiceRoom.enterSeat(seatIndex, new TRTCVoiceRoomCallback.ActionCallback()
{
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//操作成功
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
}
});
// 2.收到 onSeatListChange 回调，刷新您的麦位列表
@Override
public void onSeatListChange(final List<TRTCVoiceRoomDef.SeatInfo> seatInfoLis
t) {
}
```

4. 实现房主抱人上麦 TRTCVoiceRoom#pickSeat

```
// 1: 房主调用抱人麦位
int seatIndex = 2; //麦位的index
String userId = "123"; //需要上麦的用户id
mTRTCVoiceRoom.pickSeat(1, userId, new TRTCVoiceRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//操作成功
}
}
});
// 2.收到 onSeatListChange 回调，刷新您的麦位列表
@Override
public void onSeatListChange(final List<TRTCVoiceRoomDef.SeatInfo> seatInfoLis
t) {
}
```

5. 实现听众申请上麦 TRTCVoiceRoom#sendInvitation

```
// 听众端视角
// 1.听众调用申请上麦
String seatIndex = "1"; //麦位的index
String userId = "123"; //用户id
String inviteId = mTRTCVoiceRoom.sendInvitation("takeSeat", userId, seatIndex,
null);
// 2.收到邀请的同意请求，正式上麦
@Override
public void onInviteeAccepted(String id, String invitee) {
if(id.equals(inviteId)) {
mTRTCVoiceRoom.enterSeat(index, null);
}
}
// 房主端视角
```

```
// 1.房主收到请求
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd,
final String content) {
if (cmd.equals("takeSeat")) {
// 2.房主同意听众请求
mTRTCVoiceRoom.acceptInvitation(id, null);
}
}
```

6. 实现房主邀请上麦 TRTCVoiceRoom#sendInvitation

```
// 房主端视角
// 1.房主调用请求抱人上麦
String seatIndex = "1"; //麦位的index
String userId = "123"; //用户id
String inviteId = mTRTCVoiceRoom.sendInvitation("pickSeat", userId, seatIndex,
null);
// 2.收到邀请的同意请求，正式上麦
@Override
public void onInviteeAccepted(String id, String invitee) {
if(id.equals(inviteId)) {
mTRTCVoiceRoom.pickSeat(index, null);
}
}
// 听众端视角
// 1.听众收到请求
@Override
public void onReceiveNewInvitation(final String id, String inviter, String cmd,
final String content) {
if (cmd.equals("pickSeat")) {
// 2.听众同意房主请求
mTRTCVoiceRoom.acceptInvitation(id, null);
}
}
```

7. 实现文字聊天 TRTCVoiceRoom#sendRoomTextMsg

```
// 发送端：发送文本消息
mTRTCVoiceRoom.sendRoomTextMsg("Hello Word!", null);
// 接收端：监听文本消息
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
@Override
public void onRecvRoomTextMsg(String message, TRTCVoiceRoomDef.UserInfo userInf
```

版权所有：腾讯云计算（北京）有限责任公司

```
o) {
Log.d(TAG, "收到来自" + userInfo.userName + "的消息:" + message);
}
});
```

8. 实现弹幕消息 TRTCVoiceRoom#sendRoomCustomMsg

```
// 发送端：您可以通过自定义 Cmd 来区分弹幕和点赞消息
// eg:"CMD_DANMU"表示弹幕消息, "CMD_LIKE"表示点赞消息
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_DANMU", "Hello world", null);
mTRTCVoiceRoom.sendRoomCustomMsg("CMD_LIKE", "", null);
// 接收端：监听自定义消息
mTRTCVoiceRoom.setDelegate(new TRTCVoiceRoomDelegate() {
@Override
public void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.Us
erInfo userInfo) {
if ("CMD_DANMU".equals(cmd)) {
// 收到弹幕消息
Log.d(TAG, "收到来自" + userInfo.userName + "的弹幕消息:" + message);
} else if ("CMD_LIKE".equals(cmd)) {
// 收到点赞消息
Log.d(TAG, userInfo.userName + "给您点了个赞！");
}
}
});
```

常见问题

如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

集成 TUIVoiceRoom (iOS)

最近更新时间：2022-09-06 14:15:51

组件介绍

TUIVoiceRoom 是一个开源的音视频 UI 组件，通过在项目中集成 TUIVoiceRoom 组件，您只需要编写几行代码就可以为您的 App 添加“多人语音聊天”等场景。TUIVoiceRoom 同时支持Android等平台，基本功能如下图所示：

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。



组件集成

步骤一：下载并导入 TUIVoiceRoom 组件

在您的 xcode 工程 `Podfile` 文件同一级目录下创建 `TUIVoiceRoom` 文件夹，将 Github仓库 iOS 目录 下的 TXAppBasic、Resources、Source、TUIVoiceRoom.podspec 等文件拷贝至您在自己工程创建的 `TUIVoiceRoom` 目录下。并完成如下导入动作：

- 打开工程的 Podfile 文件，引入 TUIVocieRoom.podspec，参考如下：

```
# path 为TXAppBasic.podspec相对于Podfile文件的相对路径
pod 'TXAppBasic', :path => "TUIVoiceRoom/TXAppBasic/"
# path 为TUIVoiceRoom.podspec相对于Podfile文件的相对路径
pod 'TUIVoiceRoom', :path => "TUIVoiceRoom/", :subspecs => ["TRTC"]
```

- 终端进入 Podfile 所在的目录下，执行 `pod install` ，参考如下：

```
pod install
```

步骤二：配置权限及混淆规则

在 info.plist 文件中需要添加 `Privacy > Microphone Usage Description` 申请麦克风权限。

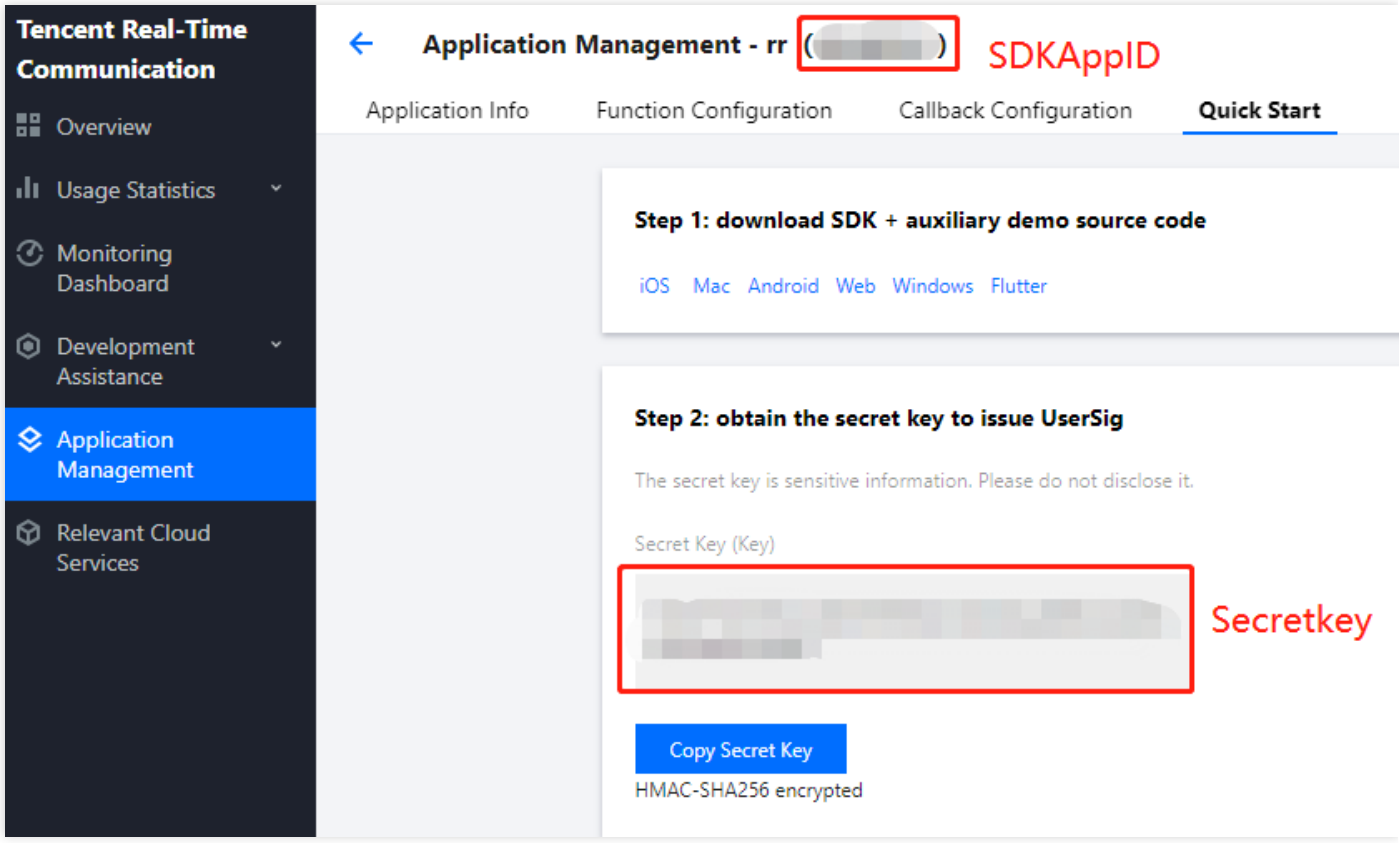
```
<key>NSMicrophoneUsageDescription</key>
<string>VoiceRoomApp需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
```

步骤三：初始化并登录

```
// 初始化
let mTRTCVoiceRoom = TRTCVoiceRoom.shared()
// 登录
mTRTCVoiceRoom.login(sdkAppID: SDKAppID, userId: userId, userSig: userSig) { cod
e, message in
if code == 0 {
//登录成功
}
}
```

参数说明：

- **SDKAppID**：**TRTC 应用 ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



- **Secretkey**：**TRTC 应用密钥**和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。建议结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的userSig，更多信息见 如何计算及使用 UserSig。

步骤四：实现语音聊天房间

1. **实现房主创建语音聊天房间 TRTCVoiceRoom#createRoom**

```
// 初始化语聊房参数
let roomParam = VoiceRoomParam()
roomParam.roomName = "房间名称"
roomParam.needRequest = false // 听众上麦是否需要房主同意
roomParam.coverUrl = "房间封面图的 URL 地址"
roomParam.seatCount = 7 // 房间座位数，这里一共7个座位，房主占了一个后听众剩下6个座位
roomParam.seatInfoList = []
// 初始化麦位信息
for _ in 0..< param.seatCount {
let seatInfo = VoiceRoomSeatInfo()
param.seatInfoList.append(seatInfo)
}
```

```
// 房主端创建房间
mTRTCVoiceRoom.createRoom(roomID: yourRoomID, roomParam: roomParam) { (code, me
ssage) in
if code == 0 {
// 创建成功
}
}
```

2. 实现听众加入语音聊天房间 TRTCVoiceRoom#enterRoom

```
// 1.听众调用加入房间
mTRTCVoiceRoom.enterRoom(roomID: roomID) { (code, message) in
// 进入房间结果回调
if code == 0 {
// 进房成功
}
}
```

3. 实现听众主动上麦 TRTCVoiceRoom#enterSeat

```
// 1: 听众调用上麦
let seatIndex = 2; //麦位的index
mTRTCVoiceRoom.enterSeat(seatIndex: 2) { (code, message) in
if code == 0 {
// 上麦成功
}
}
// 2.收到 onSeatListChange 回调，刷新您的麦位列表
@Override
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
// 刷新的麦位列表
}
```

4. 实现房主抱人上麦 TRTCVoiceRoom#pickSeat

```
// 1: 房主调用抱人麦位
let seatIndex = 2; //麦位的index
let userId = "123"; //需要上麦的用户id
mTRTCVoiceRoom.pickSeat(seatIndex: 1, userId: "123") { (code, message) in
if code == 0 {
}
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 2.收到 onSeatListChange 回调，刷新您的麦位列表
func onSeatListChange(seatInfoList: [VoiceRoomSeatInfo]) {
// 刷新的麦位列表
}
```

5. 实现听众申请上麦 TRTCVoiceRoom#sendInvitation

```
// 听众端视角
// 1.听众调用申请上麦
let seatIndex = "1"; //麦位的index
let userId = "123"; //用户id
let inviteId = mTRTCVoiceRoom.sendInvitation(cmd: "takeSeat", userId: ownerUser
Id, content: "1") { (code, message) in
// 发送结果回调
}
// 2.收到邀请的同意请求，正式上麦
func onInviteeAccepted(identifier: String, invitee: String) {
if identifier == selfID {
self.mTRTCVoiceRoom.enterSeat(seatIndex: ) { (code, message) in
// 上麦结果回调
}
}
}
// 房主端视角
// 1.房主收到请求
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, c
ontent: String) {
if cmd == "takeSeat" {
// 2.房主同意听众请求
self.mTRTCVoiceRoom.acceptInvitation(identifier: identifier, callback: nil)
}
}
```

6. 实现房主邀请上麦 TRTCVoiceRoom#sendInvitation

```
// 房主端视角
// 1.房主调用 sendInvitation，请求抱听众“123”上2号麦
let inviteId = self.mTRTCVoiceRoom.sendInvitation(cmd: "pickSeat", userId: owne
rUserId, content: "2") { (code, message) in
// 发送结果回调
}
// 2.收到邀请的同意请求，正式上麦
func onInviteeAccepted(identifier: String, invitee: String) {
if identifier == selfID {
```

```
self.mTRTCVoiceRoom.pickSeat(seatIndex: ) { (code, message) in
// 上麦结果回调
}
}
}
// 听众端视角
// 1.听众收到请求
func onReceiveNewInvitation(identifier: String, inviter: String, cmd: String, c
ontent: String) {
if cmd == "pickSeat" {
// 2.听众同意房主请求
self.mTRTCVoiceRoom.acceptInvitation(identifier: identifier, callback: nil)
}
}
```

7. 实现文字聊天 TRTCVoiceRoom#sendRoomTextMsg

```
// 发送端：发送文本消息
self.mTRTCVoiceRoom.sendRoomTextMsg(message: message) { (code, message) in

}
// 接收端：监听文本消息
func onRecvRoomTextMsg(message: String, userInfo: VoiceRoomUserInfo) {
//收到的message信息处理方法
}
```

8. 实现弹幕消息 TRTCVoiceRoom#sendRoomCustomMsg

```
// 例如：发送端：您可以通过自定义Cmd来区分弹幕和点赞消息
// eg:"CMD_DANMU"表示弹幕消息, "CMD_LIKE"表示点赞消息
self.mTRTCVoiceRoom.sendRoomCustomMsg(cmd: “CMD_DANMU”, message: "hello world",
callback: nil)
self.mTRTCVoiceRoom.sendRoomCustomMsg(cmd: "CMD_LIKE", message: "", callback: n
il)
// 接收端：监听自定义消息
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: VoiceRoomUserI
nfo) {
if cmd == "CMD_DANMU" {
// 收到弹幕消息
}
if cmd == "CMD_LIKE" {
// 收到点赞消息
}
}
```

版权所有：腾讯云计算（北京）有限责任公司

常见问题

如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

TUIVoiceRoom API 查询

TRTCVoiceRoom (iOS)

最近更新时间：2022-07-22 15:13:44

TRTCVoiceRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的语音聊天室开播，听众进入语聊房间收听/互动。
- 房主可以邀请听众上麦、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，可以和其他人语音互动，也可以随时下麦成为普通的听众。
- 支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物等。

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCVoiceRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 语音聊天室（iOS）。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCVoiceRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateHandler	设置事件回调所在的线程。
login	登录。

API	描述
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 nil，则获取房间内所有人的信息。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
moveSeat	移动麦位 (麦上主播端可调用) 。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。
sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述
sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCVoiceRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	语聊房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。
onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

版权所有：腾讯云计算（北京）有限责任公司

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

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

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

SDK 基础函数

sharedInstance

获取 TRTCVoiceRoom 单例对象。

```
/**
* 获取 TRTCVoiceRoom 单例对象
*
* - returns: TRTCVoiceRoom 实例
* - note: 可以调用 {@link TRTCVoiceRoom#destroySharedInstance()} 销毁单例对象
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

销毁 TRTCVoiceRoom 单例对象。

> 说明：
> 销毁实例后，外部缓存的 TRTCVoiceRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
/**
* 销毁 TRTCVoiceRoom 单例对象
*
* - note: 销毁实例后，外部缓存的 TRTCVoiceRoom 实例不能再使用，需要重新调用 {@link TRTCVoiceRoom#sharedInstance()} 获取新实例
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

TRTCVoiceRoom 事件回调，您可以通过 TRTCVoiceRoomDelegate 获得 TRTCVoiceRoom 的各种状态通知。

```
/**
* 设置组件回调接口
*
* 您可以通过 TRTCVoiceRoomDelegate 获得 TRTCVoiceRoom 的各种状态通知
*
* - parameter delegate 回调接口
* - note: TRTCVoiceRoom 中的回调事件，默认是在 Main Queue 中回调给您；如果您需要指定事件回调所在的队列，可使用 {@link TRTCVoiceRoom#setDelegateQueue(queue)}
*/
- (void)setDelegate:(id<TRTCVoiceRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate(delegate:));
```

> 说明：
> setDelegate 是 TRTCVoiceRoom 的代理回调。

setDelegateQueue

设置事件回调所在的线程队列，默认发送动主线程 MainQueue 中。

```
/**
* 设置事件回调所在的队列
*
```

```
* - parameter queue 队列，TRTCVoiceRoom 中的各种状态通知回调，会派发到您指定的queue。
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(q
ueue:));
```

参数如下表所示：

参数	类型	含义
queue	dispatch_queue_t	TRTCVoiceRoom 中的各种状态通知，会派发到您指定的线程队列里去。

login

登录。

```
- (void)login:(int)sdkAppID
userId:(NSString *)userId
userSig:(NSString *)userSig
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:u
serSig:callback:));
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在**实时音视频控制台 >**应用管理> 应用信息中查看 SDKAppID。
userId	NSString	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	NSString	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
- (void)logout:(ActionCallback _Nullable)callback NS_SWIFT_NAME(logout(callback
:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

版权所有：腾讯云计算（北京）有限责任公司

setSelfProfile

修改个人信息。

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

参数如下表所示：

参数	类型	含义
userName	NSString	昵称。
avatarURL	NSString	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
- (void)createRoom:(int)roomID roomParam:(VoiceRoomParam *)roomParam callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个语聊房间列表，腾讯云暂不提供语聊房间列表的管理服务，请自行管理您的语聊房间列表。
roomParam	VoiceRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

房主开播的正常调用流程如下：

版权所有：腾讯云计算（北京）有限责任公司

1. 房主调用 `createRoom` 创建新的语音聊天室，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
- (void)destroyRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(destroyRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(enterRoom(roomID:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	NSInteger	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的语音聊天室列表，可能包含多个语聊房间的 roomId 和房间信息。
2. 听众选择一个语音聊天室，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

版权所有：腾讯云计算（北京）有限责任公司

exitRoom

退出房间。

```
- (void)exitRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(exitRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> 说明：
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(VoiceRoomInfoCallback _Nullable)callback NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

参数如下表所示：

参数	类型	含义
roomIdList	NSArray<NSNumber>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定 userId 的用户信息。

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(VoiceRoomUserListCallback _Nullable)callback NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

参数如下表所示：

参数	类型	含义

参数	类型	含义
userIdList	NSArray<NSString>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> 说明：
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(enterSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

moveSeat

移动麦位 (麦上主播端可调用)。

> 说明：
> 移动麦位成功后，房间内所有成员会收到 `onSeatListChange` 、 `onAnchorLeaveSeat` 和 `onAnchorEnterSeat` 的事件通知。(主播调用后，只是修改麦位座位号信息，并不会切换该用户的主播身份。)

```
- (NSInteger)moveSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(moveSeat(seatIndex:callback:))
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要移动到的麦位序号。
callback	ActionCallback	操作回调。

返回值：

返回值	类型	含义
code	NSInteger	移动麦位操作结果（0为成功，其它为失败，10001为接口调用限频）。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> 说明：
> 下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)leaveSeat:(ActionCallback _Nullable)callback NS_SWIFT_NAME(leaveSeat(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> 说明：
> 房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(pickSeat(seatIndex:userId:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要抱上麦的麦位序号。
userId	NSString	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> 说明：
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(kickSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> 说明：
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(muteSeat(seatIndex:isMute:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要操作的麦位序号。
isMute	BOOL	YES：静音对应麦位；NO：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> 说明：
> 房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(closeSeat(seatIndex:isClose:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	NSInteger	需要操作的麦位序号。
isClose	BOOL	YES：封禁对应麦位； NO：解封对应麦位。
callback	ActionCallback	操作回调。

版权所有：腾讯云计算（北京）有限责任公司

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
- (void)startMicrophone;
```

stopMicrophone

停止麦克风采集。

```
- (void)stopMicrophone;
```

setAudioQuality

设置音质。

```
- (void)setAuidoQuality:(NSInteger)quality NS_SWIFT_NAME(setAuidoQuality(quality
:));
```

参数如下表所示：

参数	类型	含义
quality	NSInteger	音频质量，详情请参见 TRTC SDK。

muteLocalAudio

静音/取消静音本地的音频。

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

参数如下表所示：

参数	类型	含义
mute	BOOL	静音/取消静音，详情请参见 TRTC SDK。

setSpeaker

设置开启扬声器。

```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

参数如下表所示：

参数	类型	含义
useSpeaker	BOOL	YES：扬声器；NO：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
- (void)setAudioCaptureVolume:(NSInteger)voluem NS_SWIFT_NAME(setAudioCaptureVolume(volume:));
```

参数如下表所示：

参数	类型	含义
volume	NSInteger	采集音量，0 - 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

参数如下表所示：

参数	类型	含义
volume	NSInteger	播放音量，0 - 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute NS_SWIFT_NAME(muteRemoteAudio(userId:mute:));
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
userId	NSString	指定的用户 ID。
mute	BOOL	YES：开启静音；NO：关闭静音。

muteAllRemoteAudio

静音/解除静音所有成员。

```
- (void)muteAllRemoteAudio:(BOOL)isMute NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

参数如下表所示：

参数	类型	含义
mute	BOOL	YES：开启静音；NO：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable:));
```

参数如下表所示：

参数	类型	含义
enable	BOOL	YES：开启耳返；NO：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

参数如下表所示：

参数	类型	含义
message	NSString	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	NSString	命令字，由开发者自定义，主要用于区分不同消息类型。
message	NSString	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
- (NSString *)sendInvitation:(NSString *)cmd
userId:(NSString *)userId
content:(NSString *)content
```

```
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendInvitation(cmd:user
Id:content:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	NSString	业务自定义指令。
userId	NSString	邀请的用户 ID。
content	NSString	邀请的内容。
callback	ActionCallback	发送结果回调。

返回值：

返回值	类型	含义
inviteId	NSString	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(acceptInvitation(id:callback:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(rejectInvitation(id:callback:));
```

参数如下表所示：

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
id	NSString	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(cancelInvitation(id:callback:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCVoiceRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> 说明：
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
- (void)onError:(int)code
message:(NSString*)message
NS_SWIFT_NAME(onError(code:message:));
```

参数如下表所示：

参数	类型	含义

参数	类型	含义
code	int	错误码。
message	NSString	错误信息。

onWarning

警告回调。

```
- (void)onWarning:(int)code
message:(NSString *)message
NS_SWIFT_NAME(onWarning(code:message:));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	NSString	警告信息。

onDebugLog

Log 回调。

```
- (void)onDebugLog:(NSString *)message
NS_SWIFT_NAME(onDebugLog(message:));
```

参数如下表所示：

参数	类型	含义
message	NSString	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
- (void)onRoomDestroy:(NSString *)roomId
NS_SWIFT_NAME(onRoomDestroy(roomId:));
```

参数如下表所示：

参数	类型	含义
roomId	NSString	房间 ID。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
- (void)onRoomInfoChange:(VoiceRoomInfo *)roomInfo
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

参数如下表所示：

参数	类型	含义
roomInfo	VoiceRoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用muteLocalAudio，房间内的其他用户都会收到此通知。

```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

参数如下表所示：

参数	类型	含义
userId	NSString	用户 ID。
mute	BOOL	YES：静音麦位； NO：解除静音。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:
(NSInteger)totalVolume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

参数如下表所示：

参数	类型	含义

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
userVolumes	NSArray	用户列表。
totalVolume	NSInteger	音量大小，取值：0 - 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
- (void)onSeatInfoChange:(NSArray<VoiceRoomSeatInfo *> *)seatInfolist
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

参数如下表所示：

参数	类型	含义
seatInfoList	NSArray<VoiceRoomSeatInfo>	全量的麦位列表。

onAnchorEnterSeat

有成员上麦(主动上麦/房主抱人上麦)。

```
- (void)onAnchorEnterSeat:(NSInteger)index
user:(VoiceRoomUserInfo *)user
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	NSInteger	成员上麦的麦位。
user	VoiceRoomUserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
- (void)onAnchorLeaveSeat:(NSInteger)index
user:(VoiceRoomUserInfo *)user
```

版权所有：腾讯云计算（北京）有限责任公司

```
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	NSInteger	下麦的麦位。
user	VoiceRoomUserInfo	上麦用户的详细信息。

onSeatMute

房主禁麦。

```
- (void)onSeatMute:(NSInteger)index
isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(index:isMute:));
```

参数如下表所示：

参数	类型	含义
index	NSInteger	操作的麦位。
isMute	BOOL	YES：静音麦位； NO：解除静音。

onSeatClose

房主封麦。

```
- (void)onSeatClose:(NSInteger)index
isClose:(BOOL)isClose
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

参数如下表所示：

参数	类型	含义
index	NSInteger	操作的麦位。
isClose	BOOL	YES：封禁麦位； NO： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
- (void)onAudienceEnter:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	VoiceRoomUserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
- (void)onAudienceExit:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	VoiceRoomUserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
- (void)onRecvRoomTextMsg:(NSString *)message
userInfo:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

参数如下表所示：

参数	类型	含义
message	NSString	文本消息。
userInfo	VoiceRoomUserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
- (void)onRecvRoomCustomMsg:(NSString *)command
message:(NSString *)message
userInfo:(VoiceRoomUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(command:message:userInfo:));
```

参数如下表所示：

参数	类型	含义
command	NSString	命令字，由开发者自定义，主要用于区分不同消息类型。
message	NSString	文本消息。
userInfo	VoiceRoomUserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
- (void)onReceiveNewInvitation:(NSString *)identifier
inviter:(NSString *)inviter
cmd:(NSString *)cmd
content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(id:inviter:cmd:content:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
inviter	NSString	邀请人的用户 ID。
cmd	NSString	业务指定的命令字，由开发者自定义。
content	NSString	业务指定的内容。

onInviteeAccepted

被邀请者接受邀请。

```
- (void)onInviteeAccepted:(NSString *)identifier
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(id:invitee:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
invitee	NSString	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
- (void)onInviteeRejected:(NSString *)identifier
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(id:invitee:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
invitee	NSString	被邀请人的用户 ID。

onInvitationCancelled

邀请人取消邀请。

```
- (void)onInvitationCancelled:(NSString *)identifier
invitee:(NSString *)invitee NS_SWIFT_NAME(onInvitationCancelled(id:invitee:));
```

参数如下表所示：

参数	类型	含义
id	NSString	邀请 ID。
inviter	NSString	邀请人的用户 ID。

版权所有：腾讯云计算（北京）有限责任公司

TRTCVoiceRoom (Android)

最近更新时间：2022-07-22 15:13:44

TRTCVoiceRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的语音聊天室开播，听众进入语聊房间收听/互动。
- 房主可以邀请听众上麦、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，可以和其他人语音互动，也可以随时下麦成为普通的听众。
- 支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物等。

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCVoiceRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 语音聊天室（Android）。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCVoiceRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateHandler	设置事件回调所在的线程。
login	登录。

API	描述
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 null，则获取房间内所有人的信息。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
moveSeat	移动麦位（麦上主播端可调用）。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。
sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述
sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCVoiceRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	语聊房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。
onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

版权所有：腾讯云计算（北京）有限责任公司

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

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

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

SDK 基础函数

sharedInstance

获取 TRTCVoiceRoom 单例对象。

```
public static synchronized TRTCVoiceRoom sharedInstance(Context context);
```

参数如下表所示：

参数	类型	含义
context	Context	Android 上下文，内部会转为 ApplicationContext 用于系统 API 调用

destroySharedInstance

销毁 TRTCVoiceRoom 单例对象。

> 说明：
> 销毁实例后，外部缓存的 TRTCVoiceRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
public static void destroySharedInstance();
```

setDelegate

TRTCVoiceRoom 事件回调，您可以通过 TRTCVoiceRoomDelegate 获得 TRTCVoiceRoom 的各种状态通知。

```
public abstract void setDelegate(TRTCVoiceRoomDelegate delegate);
```

> 说明：
> setDelegate 是 TRTCVoiceRoom 的代理回调。

setDelegateHandler

设置事件回调所在的线程。

```
public abstract void setDelegateHandler(Handler handler);
```

参数如下表所示：

参数	类型	含义
handler	Handler	TRTCVoiceRoom 中的各种状态通知，会派发到您指定的 handler 线程。

login

登录。

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在**实时音视频控制台 >**应用管理> 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
public abstract void logout(TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
public abstract void createRoom(int roomId, TRTCVoiceRoomDef.RoomParam roomParam,
TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个语聊房间列表，腾讯云暂不提供语聊房间列表的管理服务，请自行管理您的语聊房间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

房主开播的正常调用流程如下：

1. 房主调用 `createRoom` 创建新的语音聊天室，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
public abstract void destroyRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
public abstract void enterRoom(int roomId, TRTCVoiceRoomCallback.ActionCallback c
allback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的语音聊天室列表，可能包含多个语聊房间的 roomId 和房间信息。
2. 听众选择一个语音聊天室，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

exitRoom

退出房间。

```
public abstract void exitRoom(TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> 说明：
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCVoiceRoomCallback.RoomInfoCallback callback);
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定userId的用户信息。

```
public abstract void getUserInfoList(List<String> userIdList, TRTCVoiceRoomCallback.UserListCallback userlistcallback);
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> 说明：
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
public abstract void enterSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

moveSeat

移动麦位 (麦上主播端可调用)。

> 说明：
> 移动麦位成功后，房间内所有成员会收到 `onSeatListChange` 、 `onAnchorLeaveSeat` 和 `onAnchorEnterSeat` 的事件通知。(主播调用后，只是修改麦位座位号信息，并不会切换该用户的主播身份。)

```
public abstract int moveSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要移动到的麦位序号。
callback	ActionCallback	操作回调。

返回值：

返回值	类型	含义
code	int	移动麦位操作结果（0为成功，其它为失败，10001 为接口调用限频）。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> 说明：
> 下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
public abstract void leaveSeat(TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> 说明：
> 房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
public abstract void pickSeat(int seatIndex, String userId, TRTCVoiceRoomCallbac
k.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要抱上麦的麦位序号。
userId	String	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> 说明：
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
public abstract void kickSeat(int seatIndex, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> 说明：
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isMute	boolean	true：静音对应麦位；false：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> 说明：
> 房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

版权所有：腾讯云计算（北京）有限责任公司

```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isClose	boolean	true：封禁对应麦位； false：解封对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
public abstract void startMicrophone();
```

stopMicrophone

停止麦克风采集。

```
public abstract void stopMicrophone();
```

setAudioQuality

设置音质。

```
public abstract void setAudioQuality(int quality);
```

参数如下表所示：

参数	类型	含义
quality	int	音频质量，详情请参见 TRTC SDK。

muteLocalAudio

静音/取消静音本地的音频。

```
public abstract void muteLocalAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	静音/取消静音，详情请参见 TRTC SDK。

setSpeaker

设置开启扬声器。

```
public abstract void setSpeaker(boolean useSpeaker);
```

参数如下表所示：

参数	类型	含义
useSpeaker	boolean	true：扬声器；false：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
public abstract void setAudioCaptureVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	采集音量，0 - 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

```
public abstract void setAudioPlayoutVolume(int volume);
```

参数如下表所示：

参数	类型	含义

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
volume	int	播放音量，0 - 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	指定的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音/解除静音所有成员。

```
public abstract void muteAllRemoteAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	true：开启静音；false：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

参数如下表所示：

参数	类型	含义
enable	boolean	true：开启耳返；false：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

```
public abstract TXAudioEffectManager getAudioEffectManager();
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
public abstract void sendRoomTextMsg(String message, TRTCVoiceRoomCallback.Action
Callback callback);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCVoiceRoomC
allback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
public abstract String sendInvitation(String cmd, String userId, String content,
TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	业务自定义指令。
userId	String	邀请的用户 ID。
content	String	邀请的内容。
callback	ActionCallback	发送结果回调。

返回值：

返回值	类型	含义
inviteId	String	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
public abstract void acceptInvitation(String id, TRTCVoiceRoomCallback.ActionCall
back callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

版权所有：腾讯云计算（北京）有限责任公司

```
public abstract void rejectInvitation(String id, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
public abstract void cancelInvitation(String id, TRTCVoiceRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCVoiceRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> 说明：
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
void onError(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。

```
void onWarning(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	警告信息。

onDebugLog

Log 回调。

```
void onDebugLog(String message);
```

参数如下表所示：

参数	类型	含义
message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
void onRoomDestroy(String roomId);
```

参数如下表所示：

参数	类型	含义
roomId	String	房间 ID。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
void onRoomInfoChange(TRTCVoiceRoomDef.RoomInfo roomInfo);
```

参数如下表所示：

参数	类型	含义
roomInfo	RoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用 muteLocalAudio，房间内的其他用户都会收到此通知。

```
void onUserMicrophoneMute(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID。
mute	boolean	true：静音麦位； false：解除静音。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalV
olume);
```

参数如下表所示：

参数	类型	含义
userVolumes	ListList<trtcclouddef.trtcvolumeinfo>	用户列表。
totalVolume	int	音量大小，取值：0 - 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

参数如下表所示：

参数	类型	含义
seatInfoList	List<SeatInfo>	全量的麦位列表。

onAnchorEnterSeat

有成员上麦(主动上麦/房主抱人上麦)。

```
void onAnchorEnterSeat(int index, TRTCVoiceRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	成员上麦的麦位。
user	UserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
void onAnchorLeaveSeat(int index, TRTCVoiceRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	下麦的麦位。
user	UserInfo	上麦用户的详细信息。

onSeatMute

房主禁麦。

版权所有：腾讯云计算（北京）有限责任公司

```
void onSeatMute(int index, boolean isMute);
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isMute	boolean	true：静音麦位； false：解除静音。

onSeatClose

房主封麦。

```
void onSeatClose(int index, boolean isClose);
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isClose	boolean	true：封禁麦位； false： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
void onAudienceEnter(TRTCVoiceRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
void onAudienceExit(TRTCVoiceRoomDef.UserInfo userInfo);
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
void onRecvRoomTextMsg(String message, TRTCVoiceRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCVoiceRoomDef.UserInfo us
erInfo);
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
void onReceiveNewInvitation(String id, String inviter, String cmd, String content);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
cmd	String	业务指定的命令字，由开发者自定义。
content	String	业务指定的内容。

onInviteeAccepted

被邀请者接受邀请。

```
void onInviteeAccepted(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
void onInviteeRejected(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

版权所有：腾讯云计算（北京）有限责任公司

onInvitationCancelled

邀请人取消邀请。

```
void onInvitationCancelled(String id, String inviter);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
</trtcclouddef.trtcvolumeinfo>		

视频互动直播
集成 TUILiveRoom (Android)

最近更新时间：2022-09-06 14:27:26

组件介绍

TUILiveRoom 是一个开源的视频直播 UI 组件，通过在项目中集成 TUILiveRoom 组件，您只需要编写几行代码就可以为您的 App 添加“视频互动直播”场景。TUILiveRoom 包含 Android、iOS、小程序等平台的源代码，基本功能如下图所示：

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。



组件集成

步骤一：下载并导入 TUILiveRoom 组件

单击进入 Github，选择克隆/下载代码，然后拷贝 `Android/debug` ， `Android/tuiaudioeffect` ， `Android/tuibarrage` ， `Android/tuibeauty` ， `Android/tuigift` 和 `Android/tuiliveroom` 目录到您的工程中，并完成如下导入动作：

- 在 `setting.gradle` 中完成导入，参考如下：

```
include ':debug'
include ':tuibeauty'
include ':tuibarrage'
include ':tuiaudioeffect'
include ':tuigift'
include ':tuiliveroom'
```

- 在 `app` 的 `build.gradle` 文件中添加对 `tuiliveroom` 的依赖：

```
api project(":tuiliveroom")
```

- 在根目录的 `build.gradle` 文件中添加 `TRTC SDK` 和 `IM SDK` 的依赖：

```
ext {
liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

步骤二：配置权限及混淆规则

1. 在 AndroidManifest.xml 中配置 App 的权限，SDK 需要以下权限（6.0以上的 Android 系统需要动态申请相机、读取存储权限）：

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

版权所有：腾讯云计算（北京）有限责任公司

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

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

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

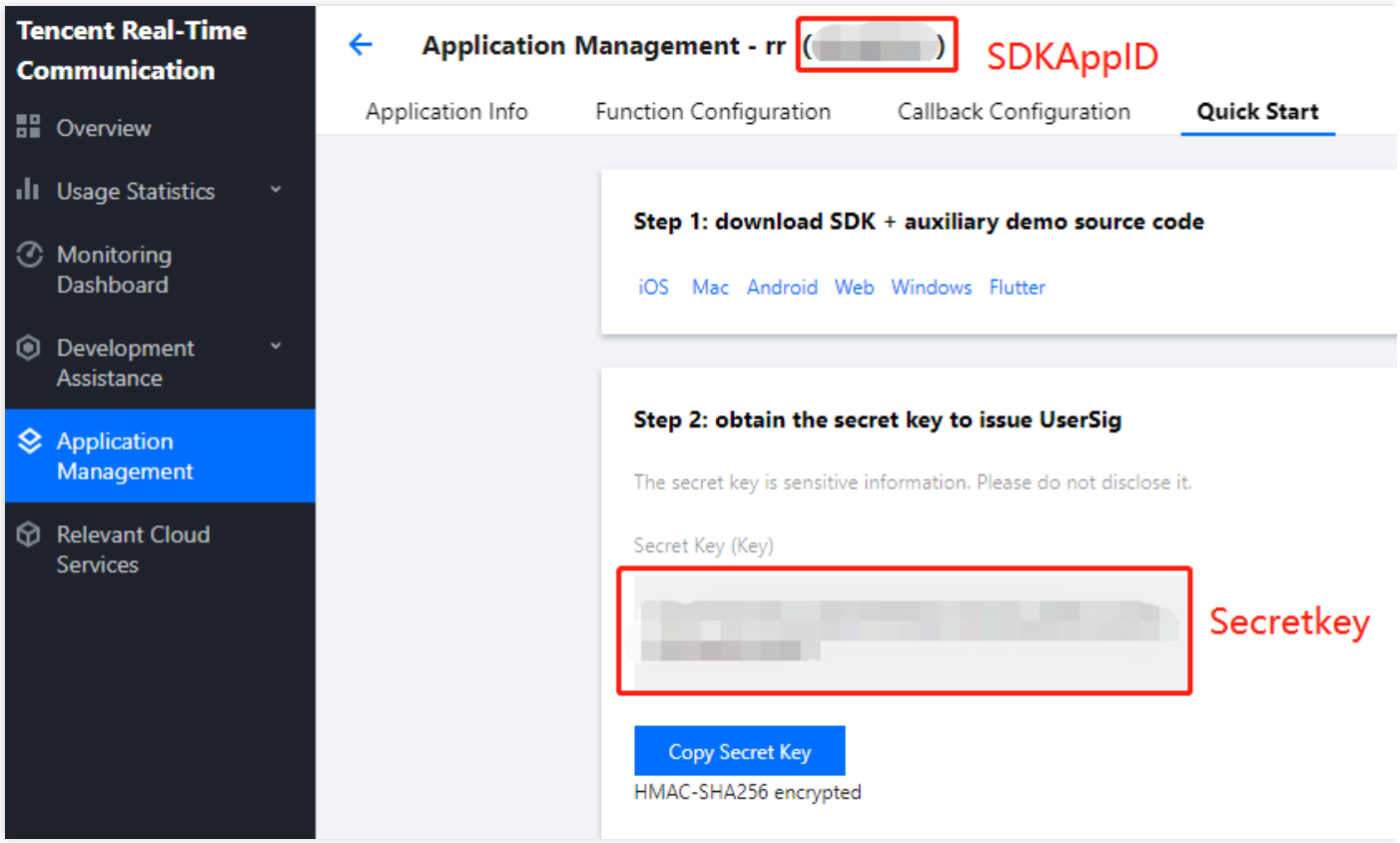
步骤三：初始化并登录组件

```
// 1.添加事件监听及登录
TUILogin.addLoginListener(new TUILoginListener() {
@Override
public void onConnecting() { // 正在连接中
super.onConnecting();
}
@Override
public void onConnectSuccess() { // 连接成功通知
super.onConnectSuccess();
}
@Override
public void onConnectFailed(int errorCode, String errorMsg) { // 连接失败通知
super.onConnectFailed(errorCode, errorMsg);
}
@Override
public void onKickedOffline() { // 登录被踢下线通知（示例：账号在其他设备登录）
super.onKickedOffline();
}
@Override
public void onUserSigExpired() { // userSig过期通知
super.onUserSigExpired();
}
});
TUILogin.login(mContext, "Your SDKAppID", "Your userId", "Your userSig", new TUIC
allback() {
@Override
public void onSuccess() {
}
@Override
public void onError(int errorCode, String errorMsg) {
Log.d(TAG, "errorCode: " + errorCode + " errorMsg:" + errorMsg);
}
```

```
});
// 2.初始化TUILiveRoom组件
TUILiveRoom mLiveRoom = TUILiveRoom.sharedInstance(mContext);
```

参数说明：

- **SDKAppID**：**TRTC 应用 ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



- **Secretkey**：**TRTC 应用密钥**和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。建议结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的userSig，更多信息请参见 如何计算及使用 UserSig。

步骤四：实现视频互动直播间

1. **主播端开播**

```
mLiveRoom.createRoom(int roomId, String roomName, String coverUrl);
```

2. **观众端观看**

```
mLiveRoom.enterRoom(roomId);
```

3. 观众与主播连麦 TRTCLiveRoom#requestJoinAnchor

```
// 1.观众端发起连麦请求
// LINK_MIC_TIMEOUT为超时时间
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);
mTRTCLiveRoom.requestJoinAnchor(mSelfUserId + "请求和您连麦", LINK_MIC_TIMEOUT
new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
// 主播接受了观众的请求
TXCloudVideoView view = new TXCloudVideoView(context);
parentView.add(view);
// 观众启动预览，开启推流
mTRTCLiveRoom.startCameraPreview(true, view, null);
mTRTCLiveRoom.startPublish(mSelfUserId + "_stream", null);
}
}
});
// 2.主播端收到连麦请求
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onRequestJoinAnchor(final TRTCLiveRoomDef.TRTCLiveUserInfo userInf
o,
String reason, final int timeout) {
// 同意对方的连麦请求
mTRTCLiveRoom.responseJoinAnchor(userInfo.userId, true, "同意连麦");
}
@Override
public void onAnchorEnter(final String userId) {
// 主播收到连麦观众的上麦通知
TXCloudVideoView view = new TXCloudVideoView(context);
parentView.add(view);
// 主播播放观众画面
mTRTCLiveRoom.startPlay(userId, view, null);
}
});
```

4. 主播与主播 PK TRTCLiveRoom#requestRoomPK

```
// 主播 A 创建12345的房间
mLiveRoom.createRoom(12345, "roomA", "Your coverUrl");
// 主播 B 创建54321的房间
mLiveRoom.createRoom(54321, "roomB", "Your coverUrl");
// 主播 A:
TRTCLiveRoom mTRTCLiveRoom=TRTCLiveRoom.sharedInstance(mContext);
// 1.主播 A 向主播 B 发起 PK 请求
mTRTCLiveRoom.requestRoomPK(54321, "B",
new TRTCLiveRoomCallback.ActionCallback() {
@Override
public void onCallback(int code, String msg) {
// 5.收到是否同意的回调
if (code == 0) {
// 用户接受
} else {
// 用户拒绝
}
}
});
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onAnchorEnter(final String userId) {
// 6.收到 B 进房的通知
mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
}
});
// 主播 B：
// 2.主播 B 收到主播 A 的消息
mTRTCLiveRoom.setDelegate(new TRTCLiveRoomDelegate() {
@Override
public void onRequestRoomPK(
final TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, final int timeout) {
// 3.主播 B 回复主播 A 接受请求
mTRTCLiveRoom.responseRoomPK(userInfo.userId, true, "");
}
@Override
public void onAnchorEnter(final String userId) {
// 4.主播 B 收到主播 A 进房的通知，播放主播 A 的画面
mTRTCLiveRoom.startPlay(userId, mTXCloudVideoView, null);
}
});
```

常见问题

如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

集成 TUILiveRoom (iOS)

最近更新时间：2023-11-20 17:51:47

组件介绍

TUILiveRoom 是一个开源的视频直播 `UI` 组件，通过在项目中集成 TUILiveRoom 组件，您只需要编写几行代码就可以为您的 App 添加“视频互动直播”场景。TUILiveRoom包含 Android、iOS、小程序等平台的源代码，基本功能如下图所示：

说明

TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

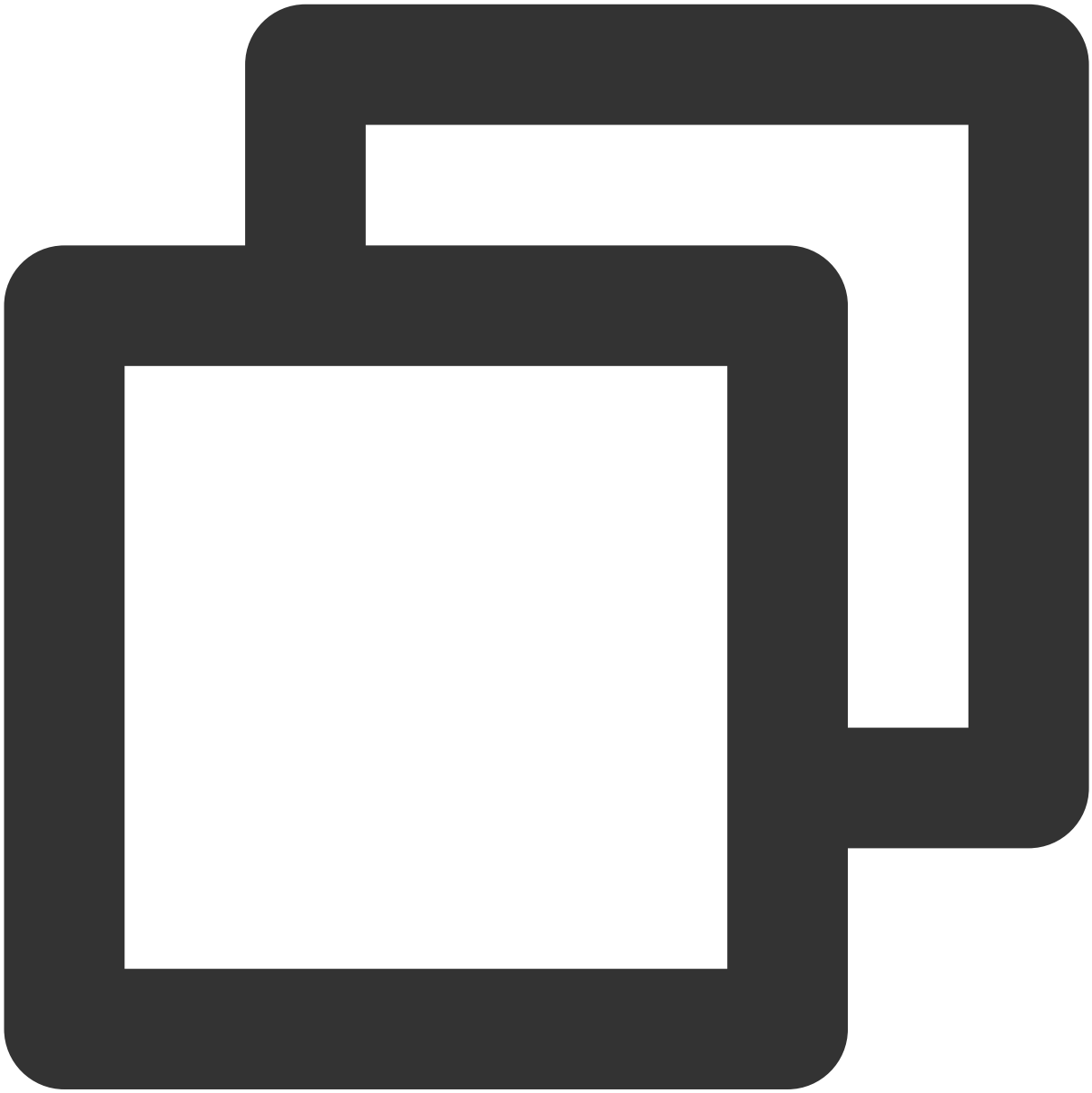


组件集成

步骤一：导入 TUILiveRoom 组件

通过 cocoapods 导入组件，具体步骤如下：

1. 在您的工程 `Podfile` 文件同一级目录下创建 `TUILiveRoom` 文件夹。
2. 单击进入 **Github/TUILiveRoom** ，选择克隆/下载代码，然后将 **TUILiveRoom/iOS/** 目录下的 `Source` 、 `Resources` 、 `TUIBeauty` 、 `TUIAudioEffect` 、 `TUIBarrage` 、 `TUIGift` 、 `TUIKitCommon` 文件夹 和 `TUILiveRoom.podspec` 文件拷贝到您在 `步骤1` 创建的 TUILiveRoom 文件夹下。
3. 在您的 Podfile 文件中添加以下依赖，之后执行 `pod install` 命令，完成导入。



```
# :path => "指向TUILiveRoom.podspec的相对路径"
pod 'TUILiveRoom', :path => "./TUILiveRoom/TUILiveRoom.podspec", :subspecs => ["TRT
# :path => "指向TUIKitCommon.podspec的相对路径"
pod 'TUIKitCommon', :path => "./TUILiveRoom/TUIKitCommon/"
# :path => "指向TUIBeauty.podspec的相对路径"
pod 'TUIBeauty', :path => "./TUILiveRoom/TUIBeauty/"
# :path => "指向TUIAudioEffect.podspec的相对路径"
pod 'TUIAudioEffect', :path => "./TUILiveRoom/TUIAudioEffect/", :subspecs => ["TRTC
# :path => "指向TUIBarrage.podspec的相对路径"
pod 'TUIBarrage', :path => "./TUILiveRoom/TUIBarrage/"
# :path => "指向TUIGift.podspec的相对路径"
```

```
pod 'TUIGift', :path => "./TUILiveRoom/TUIGift/"
```

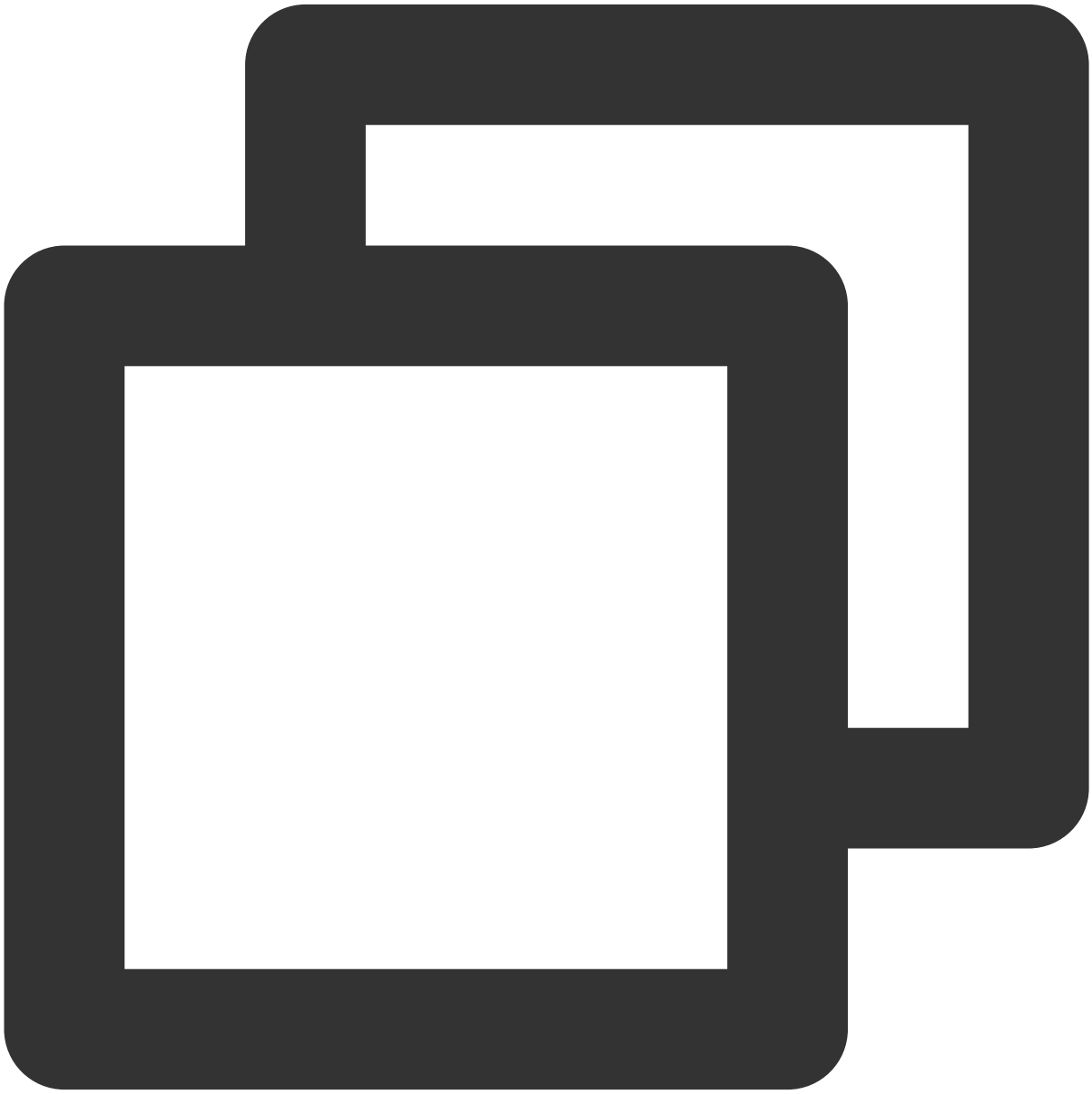
注意

`Source` 、 `Resources` 文件夹 和 `TUILiveRoom.podspec` 文件必需在同一目录下。

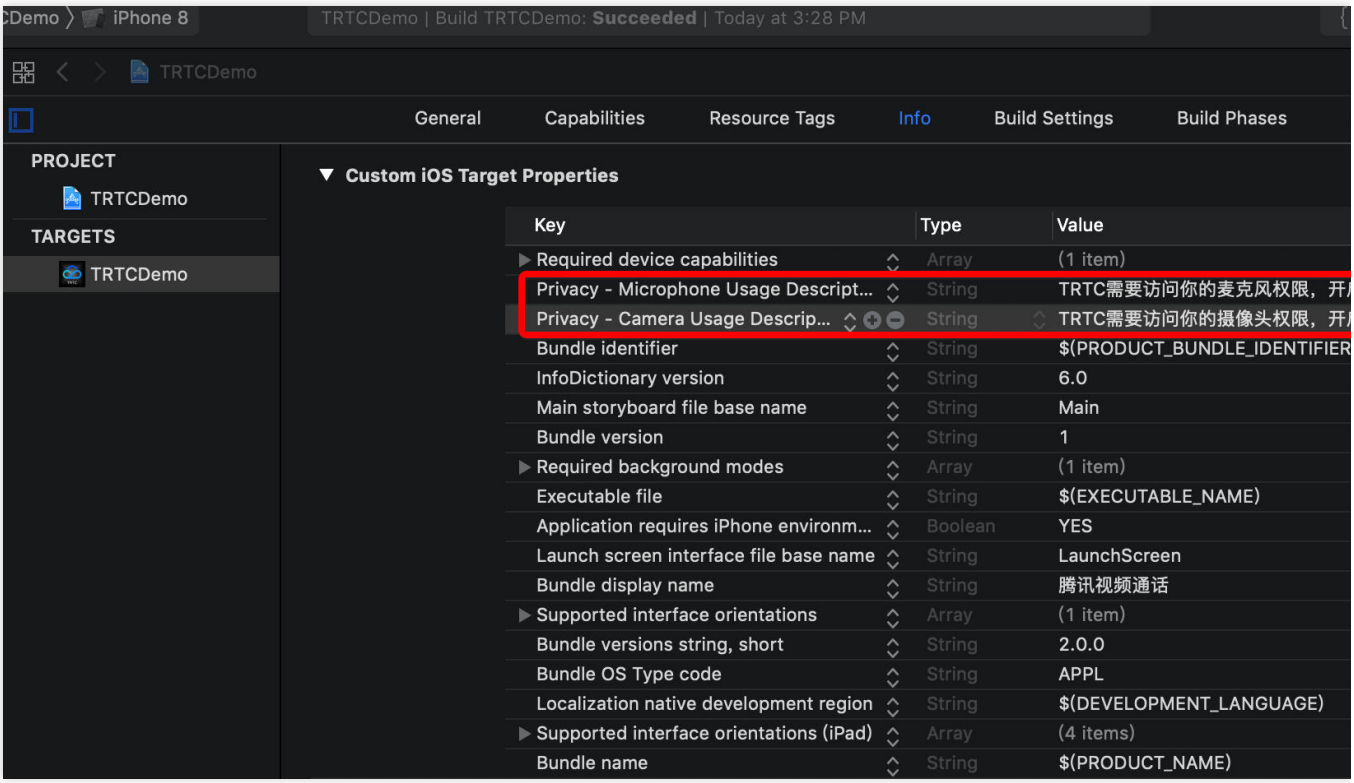
TUIKitCommon.podspec 在 TUIKitCommon文件夹下。

步骤二：配置权限

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



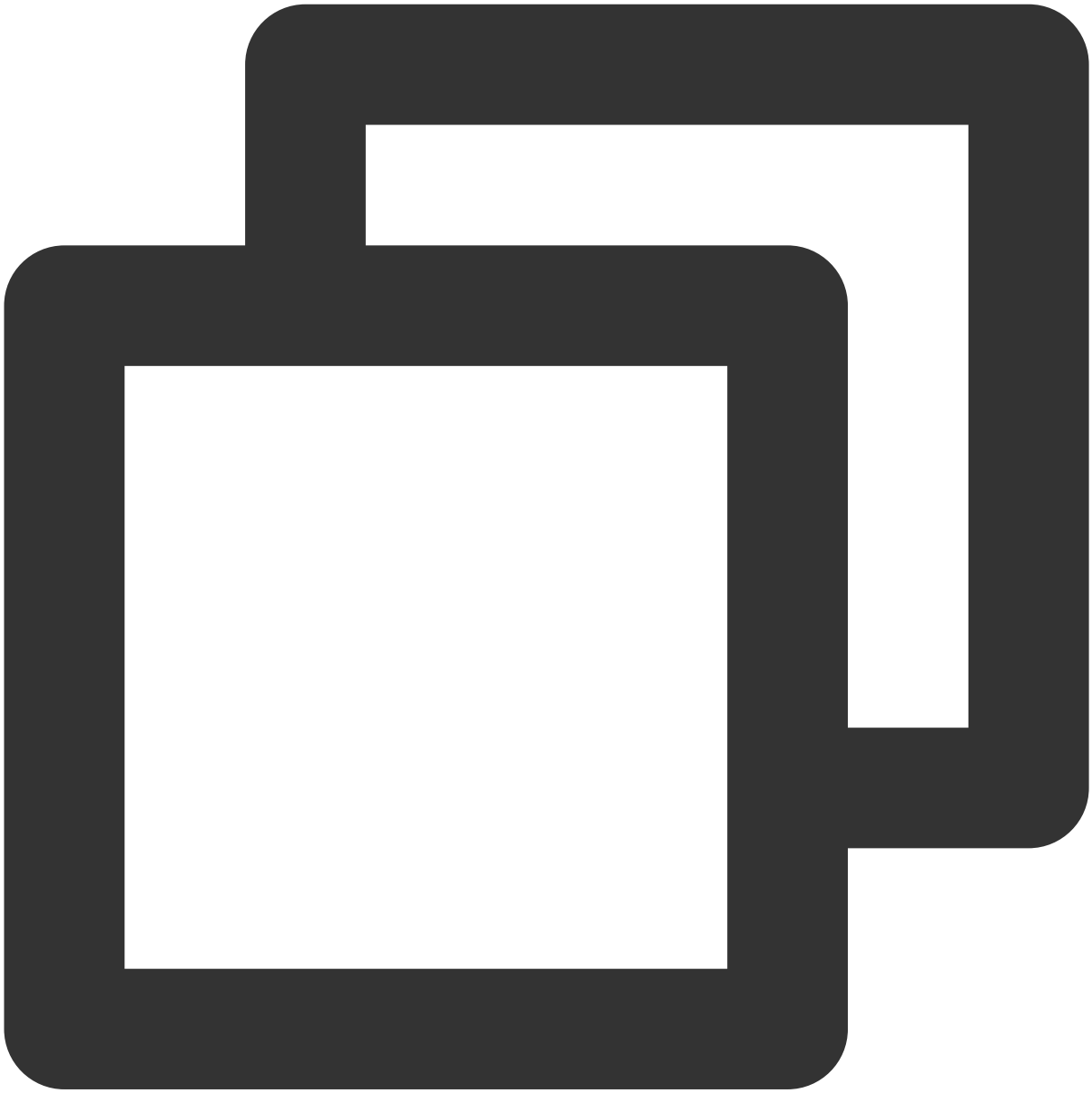
```
<key>NSCameraUsageDescription</key>
<string>RoomApp需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>RoomApp需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
```



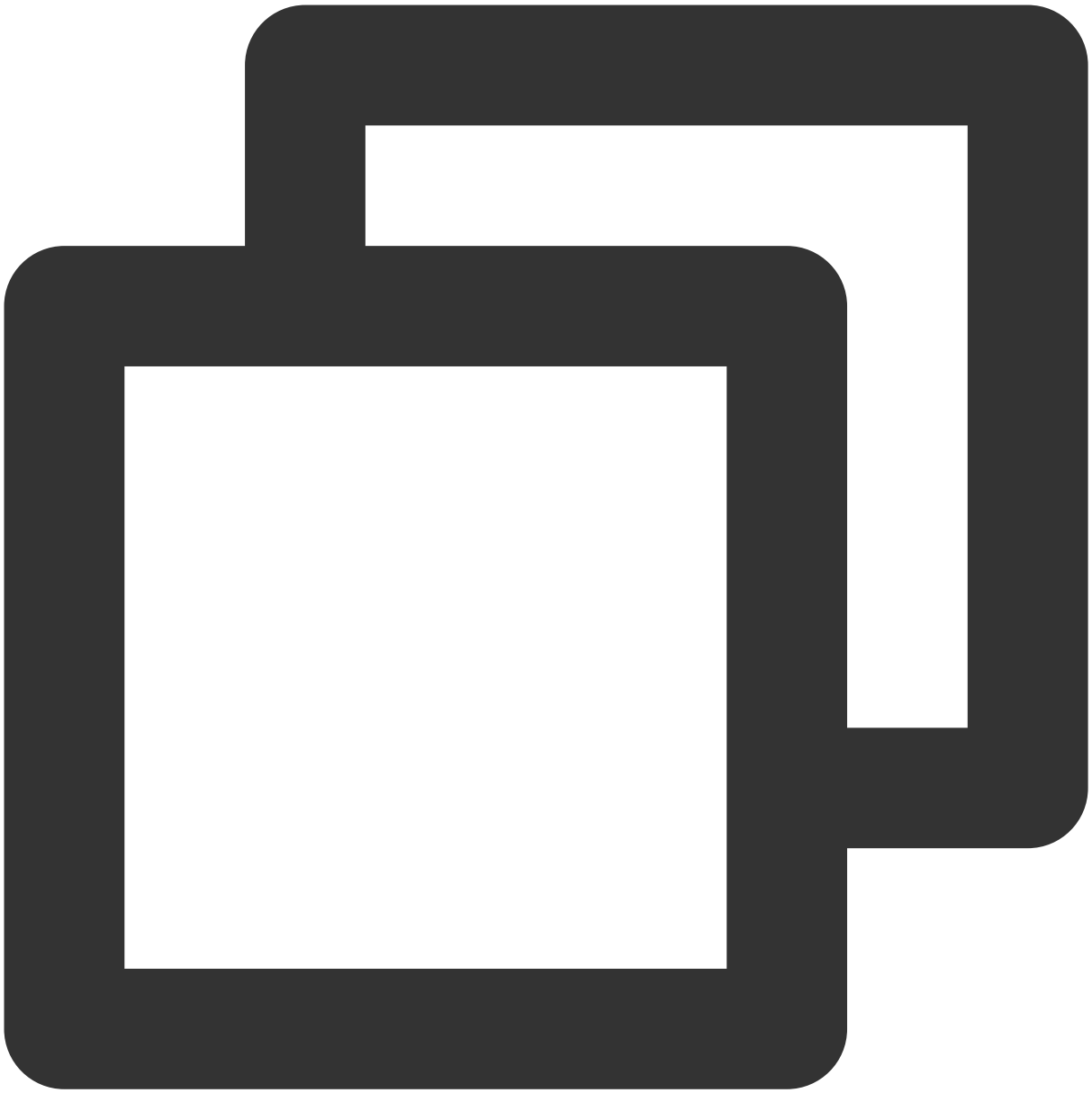
步骤三：初始化并登录组件

Objective-C

Swift



```
@import TUILiveRoom;
@import TUICore;

// 1.组件登录
[TUILogin login:@"您的SDKAppID" userID:@"您的UserID" userSig:@"您的UserSig" succ:^{

} fail:^(int code, NSString *msg) {

}];
// 2.初始化TUILiveRoom实例
TUILiveRoom *mLiveRoom = [TUILiveRoom sharedInstance];
```



```
import TUILiveRoom
import TUICore

// 1.组件登录
TUILogin.login("您的SDKAppID", userID: "您的UserID", userSig: "您的UserSig") {

} fail: { code, msg in

}
// 2.初始化TUILiveRoom实例
```

```
let mLiveRoom = TUILiveRoom.sharedInstance
```

参数说明：

SDKAppID：**TRTC 应用 ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



SecretKey：**TRTC 应用密钥**，和 SDKAppID 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。

UserID：当前用户的 ID，字符串类型，长度不超过32字节，不支持使用特殊字符，建议使用英文或数字，可结合业务实际账号体系自行设置。

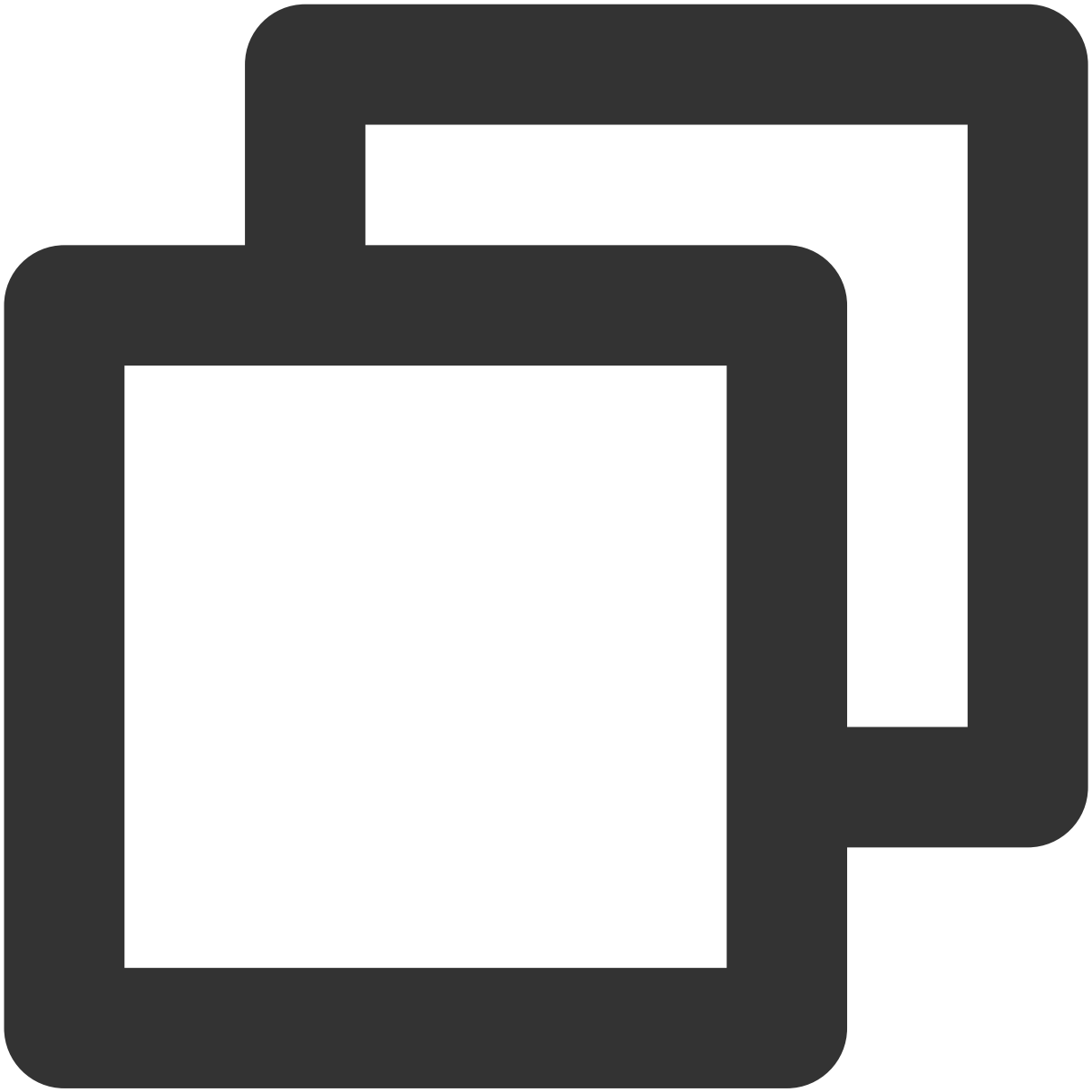
UserSig：根据 SDKAppId、UserID，SecretKey 等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的 UserSig，也可以参照我们的 TUILiveRoom 示例工程 自行计算，更多信息见 如何计算及使用 UserSig。

步骤四：实现视频互动直播间

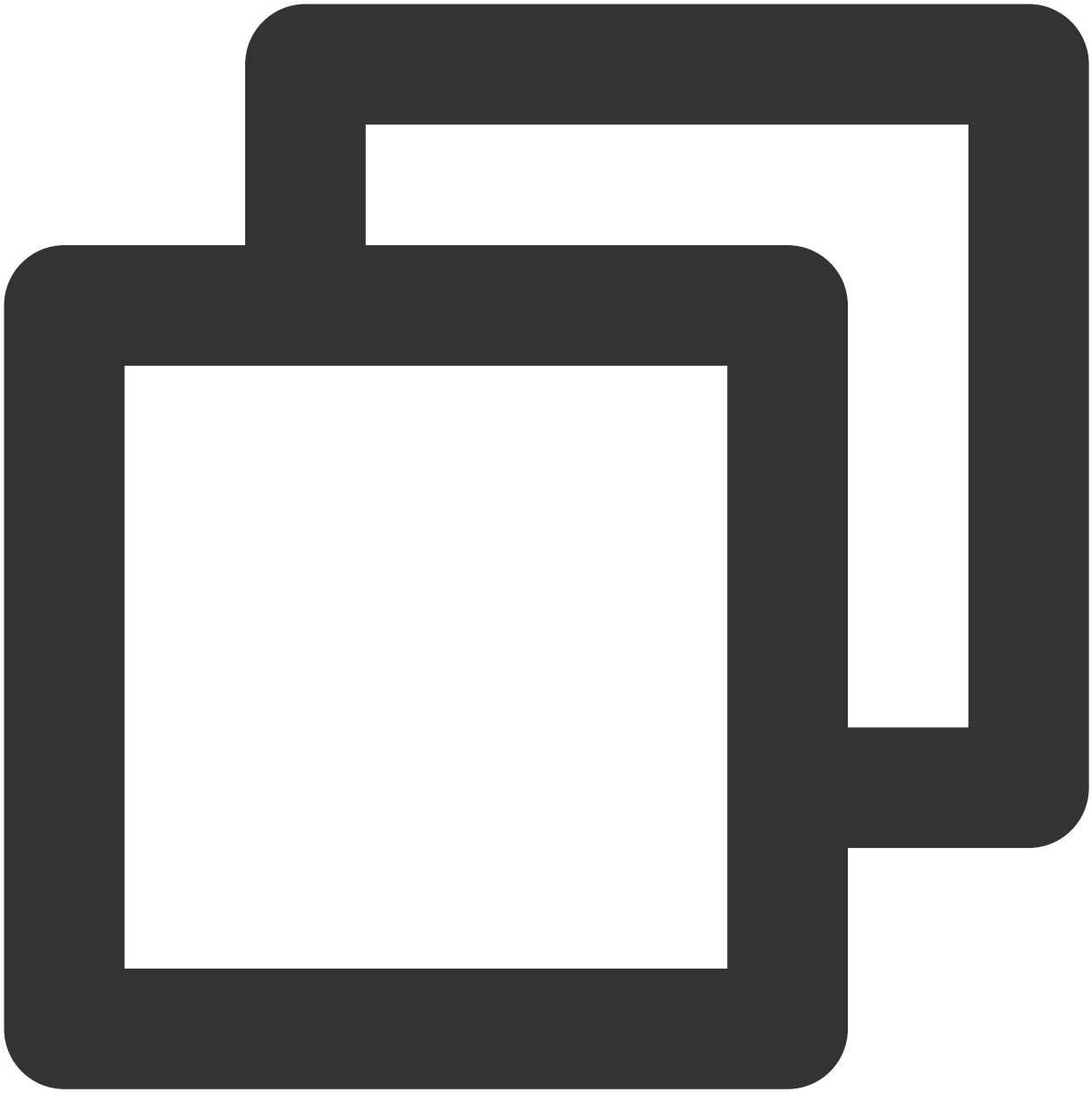
1. **主播端开播**

Objective-C

Swift



```
[mLiveRoom createRoomWithRoomId:123 roomName:@"test room" coverUrl:@""];
```

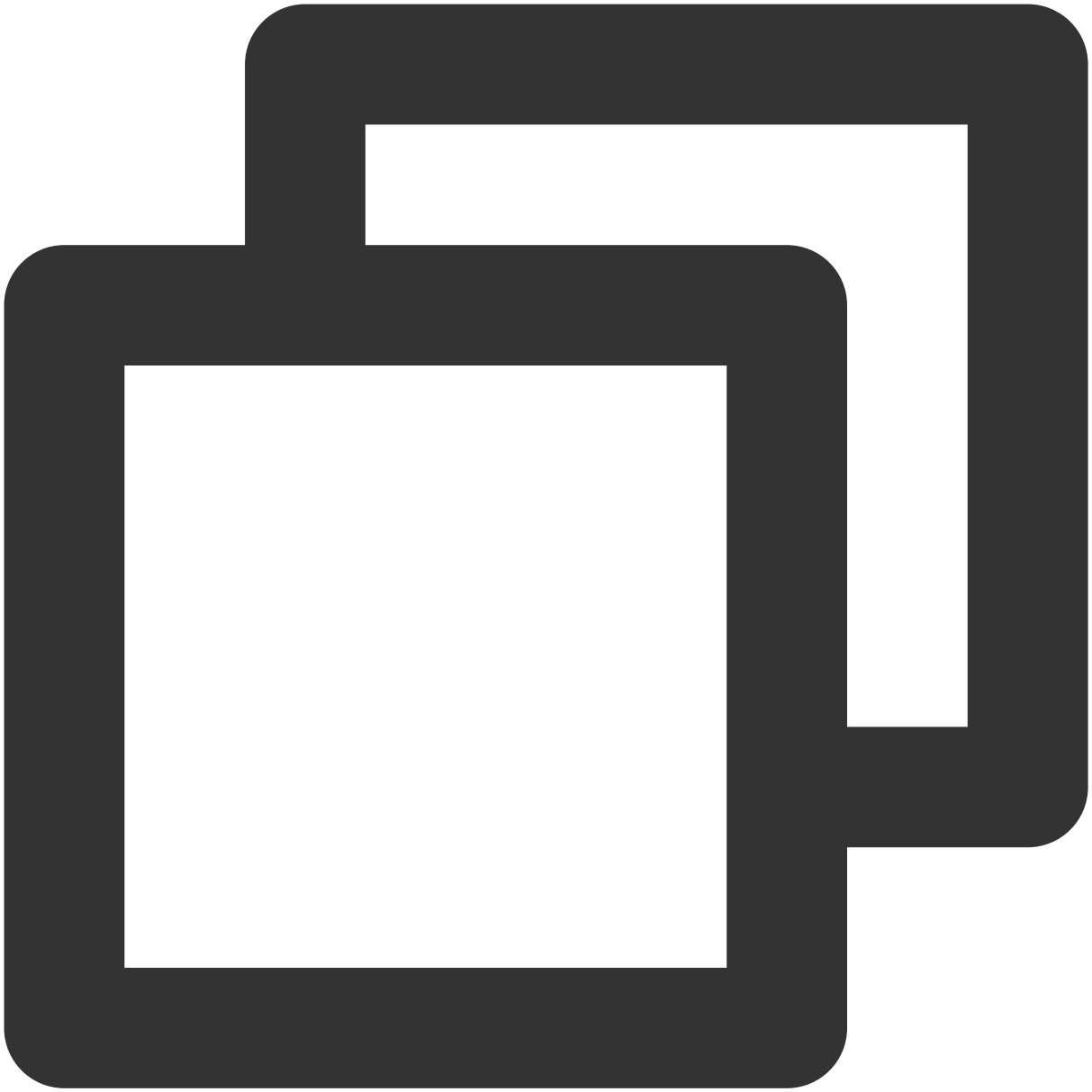


```
mLiveRoom.createRoom(roomId: 123, roomName: "test room", coverUrl:"")
```

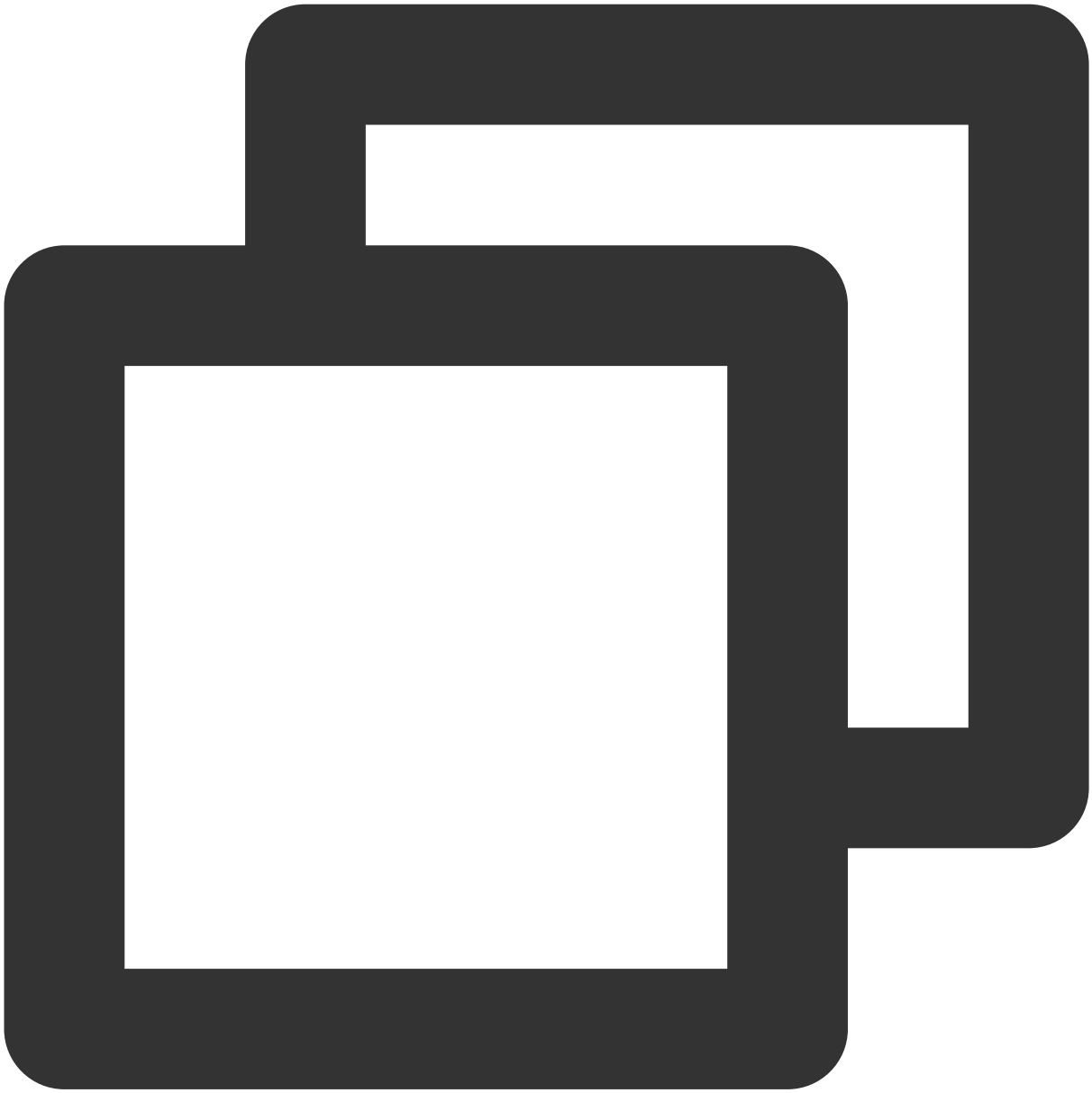
2. 观众端观看

Objective-C

Swift



```
[mLiveRoom enterRoomWithRoomId:123];
```

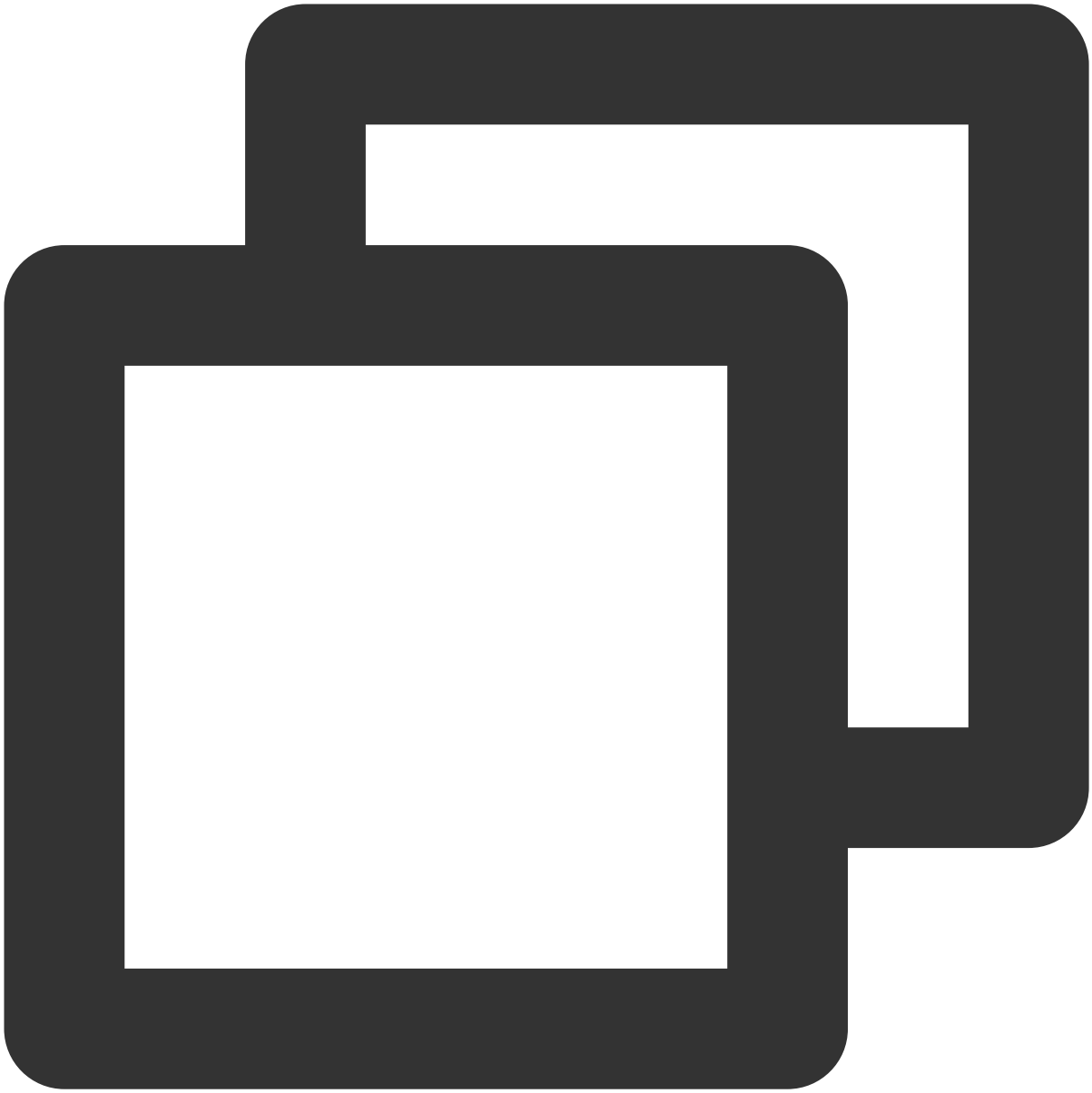


```
mLiveRoom.createRoom(roomId: 123)
```

3. **观众与主播连麦** TRTCLiveRoom#requestJoinAnchor

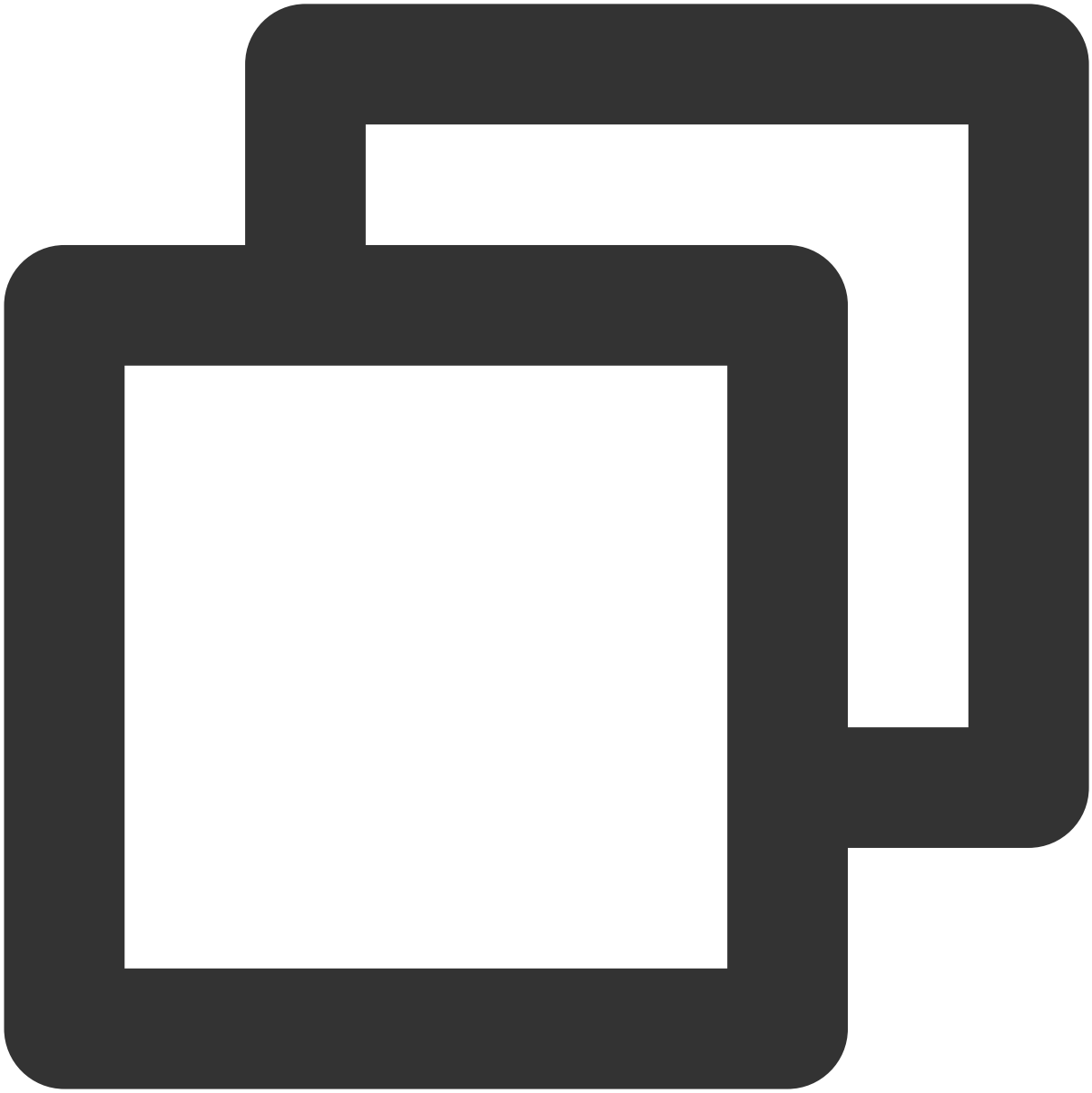
Objective-C

Swift



```
// 1.观众端发起连麦请求
[TRTCLiveRoom shareInstance].delegate = self;
// @param mSelfUserId String 当前用户id
NSString *mSelfUserId = @"1314";
[[TRTCLiveRoom shareInstance] requestJoinAnchor:[NSString stringWithFormat:@"%@ 请求
    if (agreed) {
        // 主播接受了观众的请求
      UIView *playView = [UIView new];
            [self.view addSubView:playView];
        // 观众启动预览，开启推流
      [[TRTCLiveRoom shareInstance] startCameraPreviewWithFrontCamera:YES view:play
```

```
        [[TRTCLiveRoom shareInstance] startPublishWithStreamID:[NSString stringWithFo
    }
}];

// 2.主播端收到连麦请求
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestJoinAnchor:(TRTCLiveUser
    // 同意对方的连麦请求
    [[TRTCLiveRoom shareInstance] responseJoinAnchor:user.userId agree:YES reason:@
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID
    // 主播收到连麦观众的上麦通知
    UIView *playView = [UIView new];
    [self.view addSubview:playView];
    // 主播播放观众画面
    [[TRTCLiveRoom shareInstance] startPlayWithUserID:userID view:playView callback
}
```



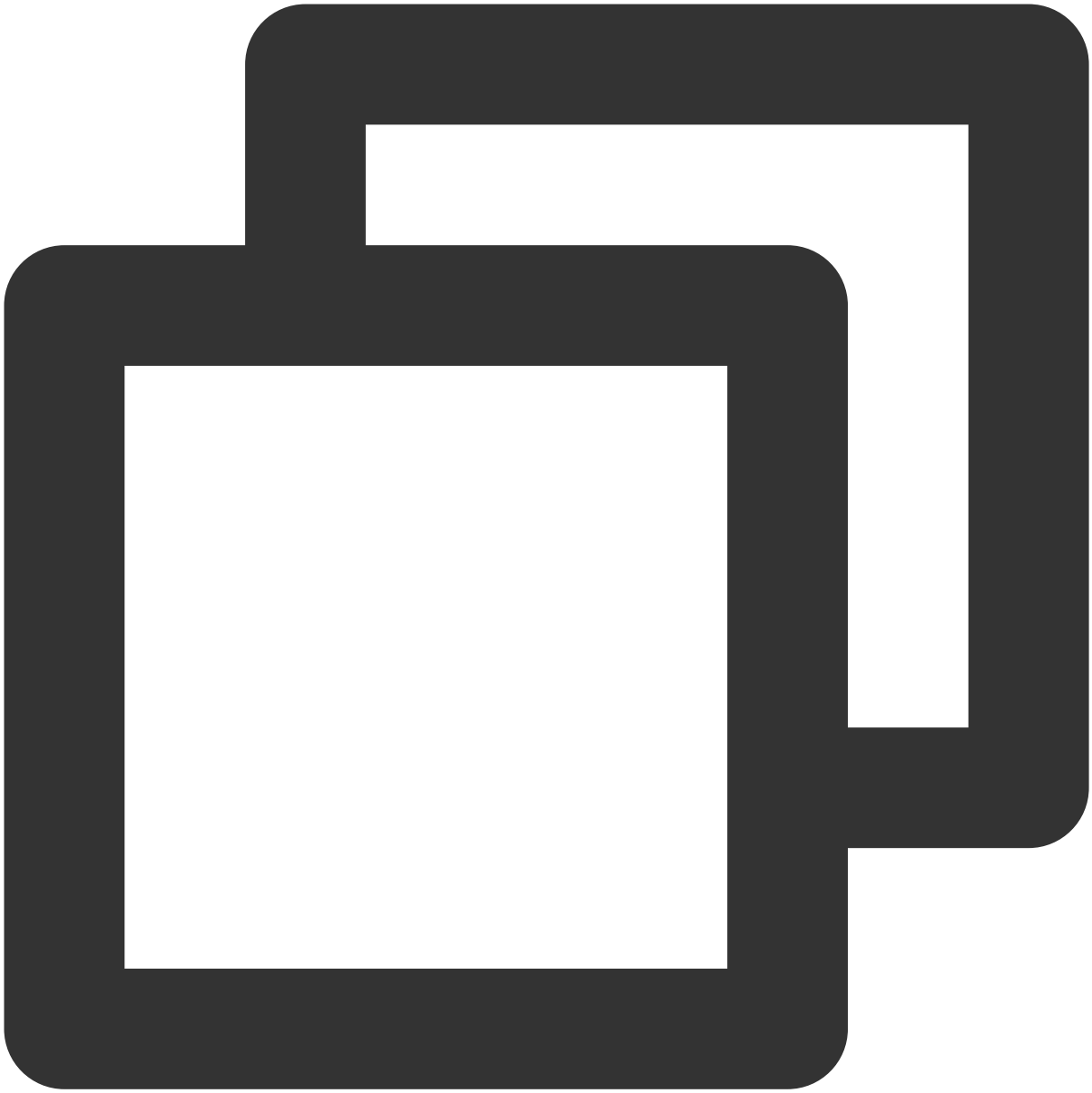
```
// 1.观众端发起连麦请求
TRTCLiveRoom.shareInstance().delegate = self
let mSelfUserId = "1314"
TRTCLiveRoom.shareInstance().requestJoinAnchor(reason: mSelfUserId + "请求和您连麦",
    guard let self = self else { return }
  if agree {
        // 主播接受了观众的请求
        let playView = UIView()
        self.view.addSubView(playView)
        // 观众启动预览，开启推流
        TRTCLiveRoom.shareInstance().startCameraPreview(frontCamera: true, view: pl
```

```
            TRTCLiveRoom.shareInstance().startPublish(streamID: mSelfUserId + "_stream"
    }
}

// 2.主播端收到连麦请求
extension ViewController: TRTCLiveRoomDelegate {

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestJoinAnchor user: TRTCL
        // 同意对方的连麦请求
        TRTCLiveRoom.shareInstance().responseRoomPK(userID: user.userId, agree: tru
    }

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
        // 主播收到连麦观众的上麦通知
        let playView = UIView()
        view.addSubview(playView)
        // 主播播放观众画面
        TRTCLiveRoom.shareInstance().startPlay(userID: userID, view: playView);
    }
}
```

4. **主播与主播 PK** TRTCLiveRoom#requestRoomPK

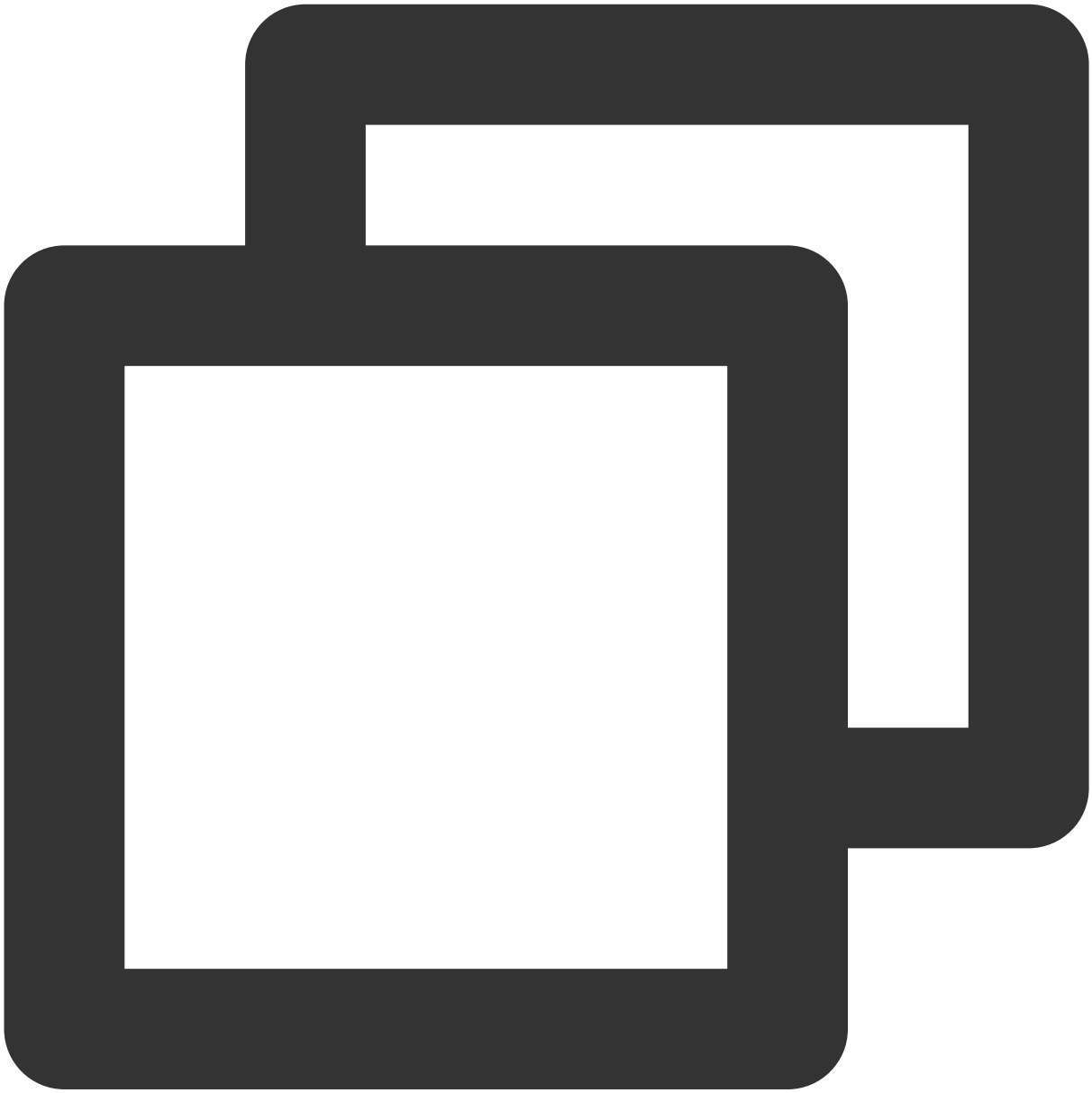
Objective-C

Swift



```
// 主播 A 创建12345的房间
[[TUILiveRoom sharedInstance] createRoomWithRoomId:12345 roomName:@"roomA" coverUrl
// 主播 B 创建54321的房间
[[TUILiveRoom sharedInstance] createRoomWithRoomId:54321 roomName:@"roomB" coverUrl

// 主播 A
// 主播 A 向 主播 B 发起PK请求
[[TRTCLiveRoom shareInstance] requestRoomPKWithRoomID:543321 userID:@"roomB userId"
    if (agreed) {
        // 用户B接受
    } else {
```

```
        // 用户B拒绝
    }
}];

// 主播 B：
// 2.主播 B 收到主播 A 的消息
#pragma mark - TRTCLiveRoomDelegate
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onRequestRoomPK:(TRTCLiveUserInfo
    // 3.主播 B 回复主播 A 接受请求
    [[TRTCLiveRoom shareInstance] responseRoomPKWithUserID:user.userId agree:YES re
}

- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom onAnchorEnter:(NSString *)userID
    // 4.主播 B 收到主播 A 进房的通知，播放主播 A 的画面
    [[TRTCLiveRoom shareInstance] startPlayWithUserID:userID view:playAView callbac
}
```



```
// 主播 A 创建12345的房间
TUILiveRoom.sharedInstance.createRoom(roomId: 12345, roomName: "roomA")
// 主播 B 创建54321的房间
TUILiveRoom.sharedInstance.createRoom(roomId: 54321, roomName: "roomB")

// 主播 A
// 主播 A 向 主播 B 发起PK请求
TRTCLiveRoom.shareInstance().requestRoomPK(roomID: 543321, userID: "roomB userId",
     guard let self = self else { return }
    if agreed {
        // 用户B接受
```

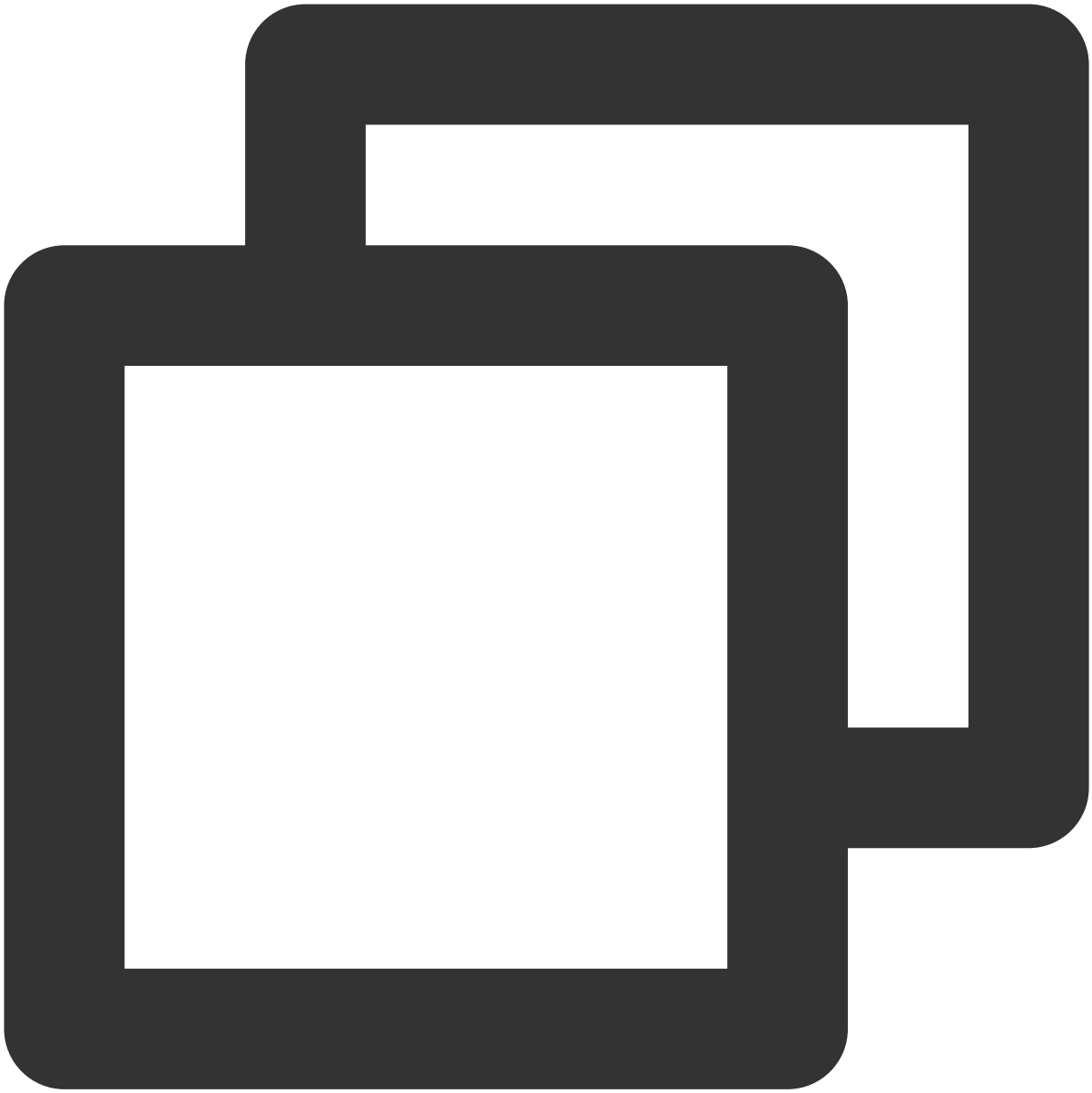
```
    } else {
        // 用户B拒绝
    }
}

// 主播 B：
// 2.主播 B 收到主播 A 的消息
extension ViewController: TRTCLiveRoomDelegate {
    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onRequestRoomPK user: TRTCLiveU
        // 3.主播 B 回复主播 A 接受请求
        TRTCLiveRoom.shareInstance().responseRoomPK(userID: user.userId, agree: tru
    }

    func trtcLiveRoom(_ trtcLiveRoom: TRTCLiveRoom, onAnchorEnter userID: String) {
        // 4.主播 B 收到主播 A 进房的通知，播放主播 A 的画面
        TRTCLiveRoom.shareInstance().startPlay(userID: userID, view: playAView);
    }
}
```

步骤五：美颜特效（可选）

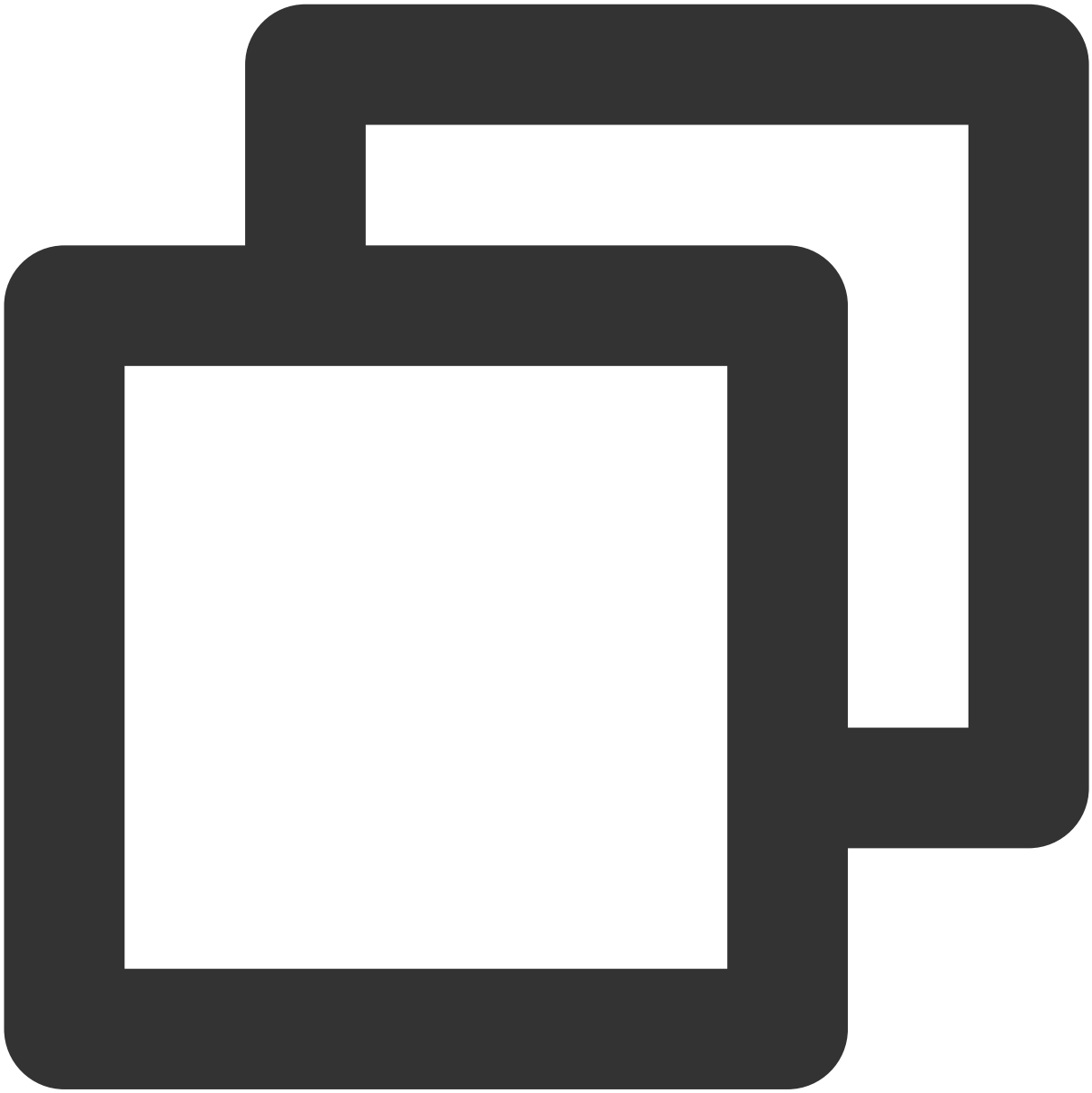
TUILiveRoom 美颜使用了 腾讯特效 SDK，在使用美颜功能时，需要先设置 XMagic License，XMagic License 申请请参见 XMagic License 申请指引。

Objective-C

Swift



```
@import TUIBeauty;

- (void)setXMagicLicense {
    [[TUIBeautyView getBeautyService] setLicenseUrl:@"XMagicLicenseURL" key:@"XMagi

    }];
}
```



```
import TUIBeauty

func setXMagicLicence() {
      // [Option] Tencent Effect: XMagic Beauty License
      TUIBeautyView.getBeautyService().setLicenseUrl(XMagicLicenseURL, key: XMagicL
          debugPrint("auth result code:\\(code) msg:\\(msg)")
      }
}
```

交流与反馈

如果您是开发者，欢迎您加入我们的技术交流 QQ 群：**770645461**，进行技术交流和产品沟通。

版权所有：腾讯云计算（北京）有限责任公司

集成 TUIPusher&TUIPlayer (Web)

最近更新时间：2022-09-06 14:22:23

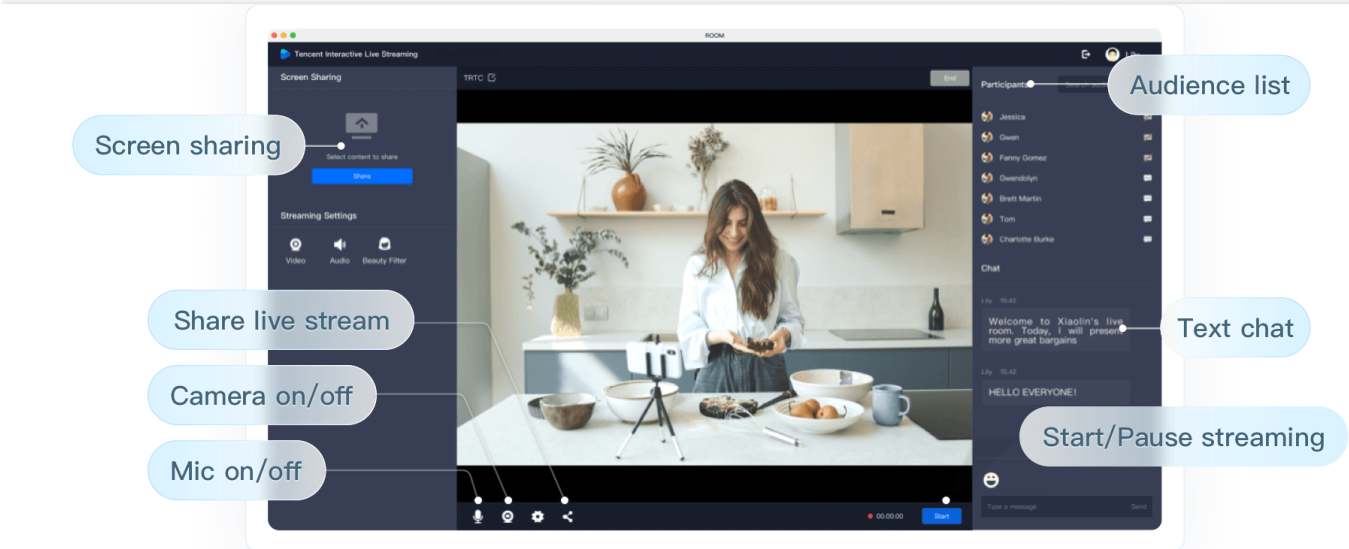
组件介绍

TUIPusher & TUIPlayer 是 Web 端开源的含 UI 直播互动组件。TUIPusher & TUIPlayer 集成 实时音视频 TRTC 、 即时通信 IM 等基础 SDK，为企业直播、电商带货、行业培训、远程教学等多种直播场景提供快速上线 Web 端直播推拉流工具的解决方案。

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TUIPusher & TUIPlayer 的优势：

- 贴合直播场景需求，提供了含 UI 的直播场景通用解决方案，覆盖了直播场景常见功能（如设备选择、美颜、直播推流、观众拉流、聊天等），助力业务快速上线。
- 直接接入腾讯云实时音视频 TRTC、腾讯云即时通信 IM 以及腾讯云超级播放器 TCPlayer 等基础 SDK，方便客户灵活扩展业务功能。
- Web 端易于用户使用，易于功能迭代的天然优势。



快速体验

为了方便您快速体验 TUIPusher & TUIPlayer 的功能，我们结合用户管理系统和房间管理系统提供了 TUIPusher 体验链接 及 TUIPlayer 体验链接。

> 注意：
> 同时体验 TUIPusher 和 TUIPlayer 需要使用两个不同的账号登录。

TUIPusher 推流组件功能介绍

- 支持采集摄像头和麦克风的流并推流
  - 可根据需求设置视频参数（帧率，分辨率，码率）
  - 支持开启美颜并设置视频美颜参数
- 支持采集屏幕分享流并推流
- 支持推流到腾讯云实时音视频后台，推流到腾讯云 CDN
- 支持在线聊天室，和在线观众进行聊天互动
- 支持获取观众列表，对在线观众进行禁言操作

TUIPlayer 拉流组件功能介绍

- 支持同时播放音视频流和屏幕分享流
- 支持在线聊天室，和主播及其他观众进行聊天互动
- 支持超低延时直播（300ms 延时），快直播（1000ms 以内延时）以及标准直播（支持超高并发观看）三种拉流线路
- 兼容桌面浏览器及移动端浏览器，支持移动端浏览器横屏观看

> 注意：
> 部分浏览器不支持 WebRTC，只能使用标准直播线路观看，如需体验其他线路，请尝试更换浏览器。

组件集成

步骤一：开通腾讯云服务

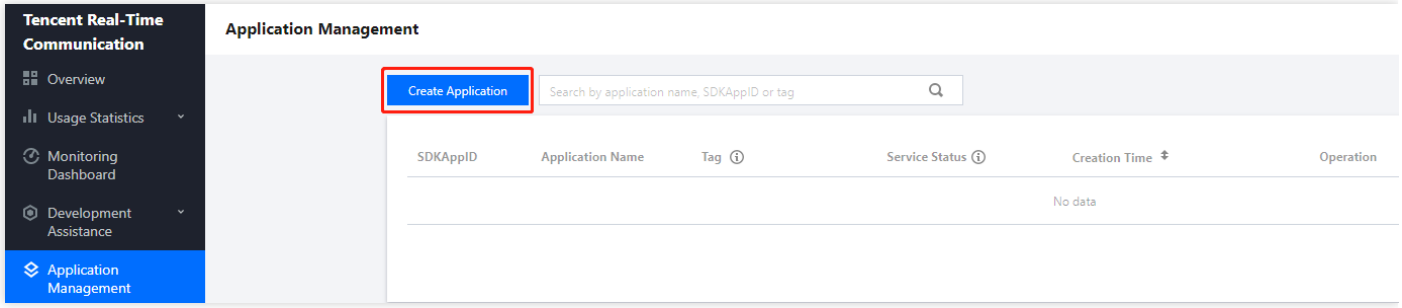
> 注意：

- TUIPusher & TUIPlayer 基于腾讯云实时音视频和即时通信服务进行开发。实时音视频 TRTC 应用与 即时通信 IM 应用的 SDKAppID 一致，才能复用账号与鉴权。
- 即时通信 IM 应用针对文本消息，提供基础版本的 安全打击 能力，如果希望使用自定义不雅词功能，可以单击 **升级** 。
- 本地计算 UserSig 的方式仅用于本地开发调试，请勿直接发布到线上，一旦 SECRETKEY 泄露，攻击者就可以盗用您的腾讯云流量。正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

- 方式1：基于实时音视频
- 方式2：基于即时通信 IM

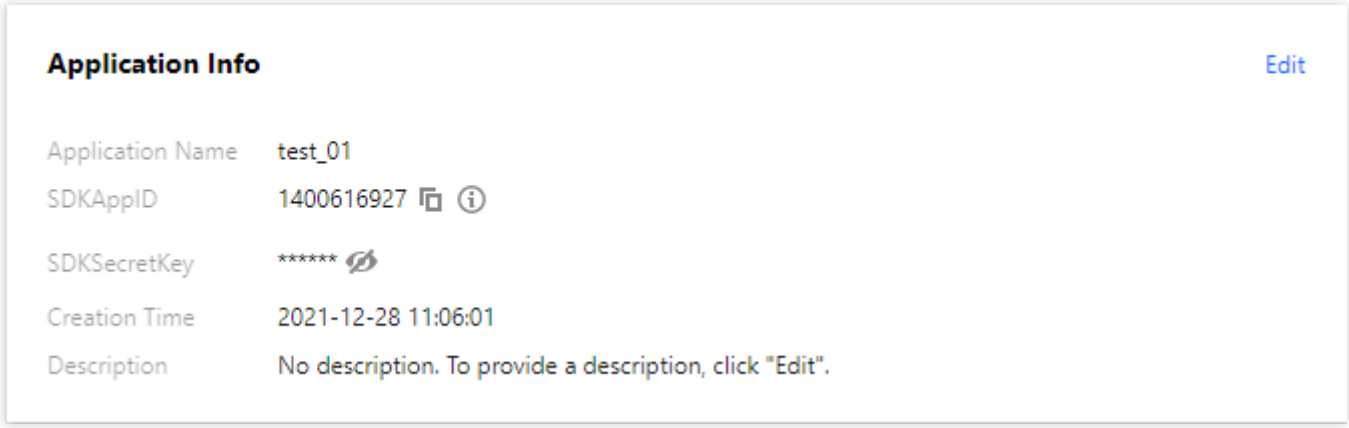
步骤1：创建实时音视频 TRTC 应用

1. 注册腾讯云账号 并开通 实时音视频 和 即时通信 服务。
2. 在 实时音视频控制台 单击 **应用管理 > 创建应用** 创建新应用。

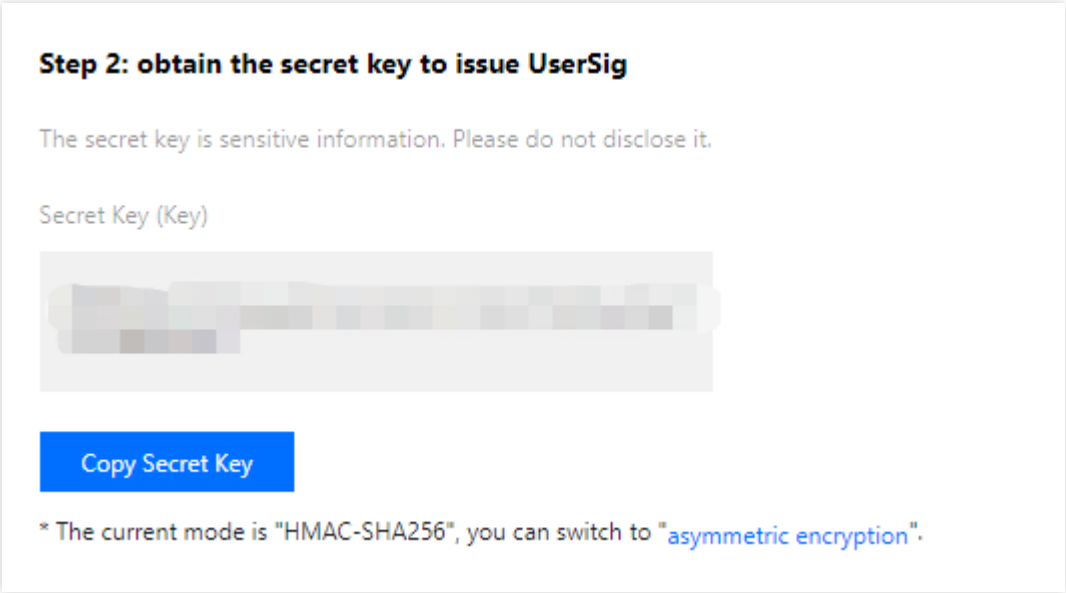


步骤2: 获取 TRTC 密钥信息

1. 在 **应用管理 > 应用信息** 中获取 SDKAppID 信息。



2. 在 **应用管理 > 快速上手** 中获取应用的 secretKey 信息。



说明

- 首次创建实时音视频应用的腾讯云账号，可获赠一个10000分钟的音视频资源免费试用包。
- 创建实时音视频应用之后会自动创建一个 SDKAppID 相同的即时通信 IM 应用，可在 即时通信控制台 配置该应用的套餐信息。

步骤二：项目准备

1. 在 GitHub 下载 TUIPusher & TUIPlayer 代码。

2. 为 TUIPusher & TUIPlayer 安装依赖。

```
cd Web/TUIPusher
npm install
cd Web/TUIPlayer
npm install
```

3. 将 sdkAppId 和 secretKey 填入 `TUIPusher/src/config/basic-info-config.js` 及 `TUIPlayer/src/config/basic-info-config.js` 配置文件中。

4. 本地开发环境运行 TUIPusher & TUIPlayer。

```
cd Web/TUIPusher
npm run serve
cd Web/TUIPlayer
npm run serve
```

5. 可打开 `http://localhost:8080` 和 `http://localhost:8081` 体验 TUIPusher 和 TUIPlayer 功能。

6. 可更改 `TUIPusher/src/config/basic-info-config.js` 及 `TUIPlayer/src/config/basic-info-config.js` 配置文件中的房间，主播及观众等信息，**注意保持 TUIPusher 和 TUIPlayer 的房间信息，主播信息一致**。

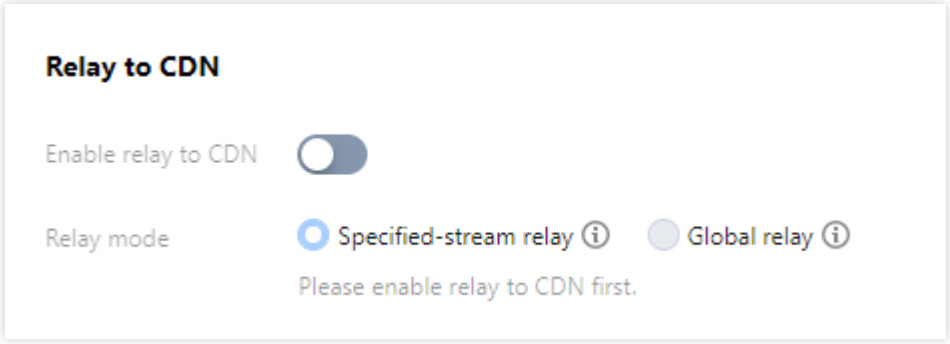
注意：

- 完成以上配置，您可以使用 TUIPusher & TUIPlayer 进行超低延时直播，如您需要支持快直播和标准直播，请继续阅读 步骤三：旁路直播。
- 本地计算 UserSig 的方式仅用于本地开发调试，请勿直接发布到线上，一旦您的 `SECRETKEY` 泄露，攻击者就可以盗用您的腾讯云流量。
- 正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

步骤三：旁路直播

TUIPusher & TUIPlayer 实现的快直播和标准直播依托于腾讯云 云直播服务，因此支持快直播和标准直播线路需要您开启旁路推流功能。

1. 在 **实时音视频控制台** 中为您正在使用的应用开启旁路推流配置，可按需开启指定流旁路或全局自动旁路。



2. 请在 **域名管理** 页面添加自有播放域名，具体请参见 添加自有域名。
3. 在 `TUIPlayer/src/config/basic-info-config.js` 配置文件中配置播放域名。

完成以上配置，您可以体验 TUIPusher & TUIPlayer 支持超低延时直播，快直播以及标准直播的所有功能。

步骤四：生产环境应用

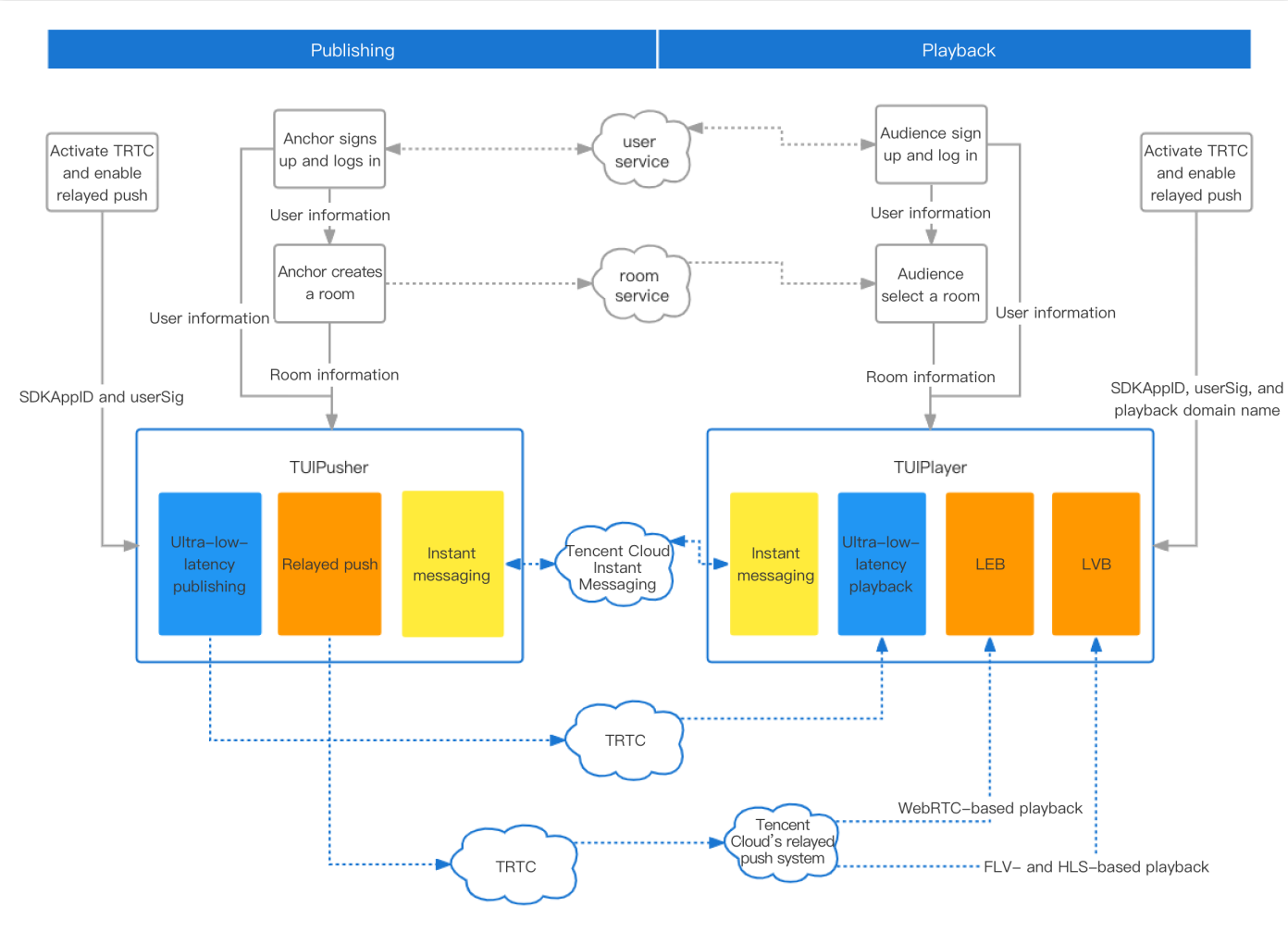
当您将 TUIPusher & TUIPlayer 用于生产应用时，在接入 TUIPusher & TUIPlayer 之外，您需要：

- 创建用户管理系统，用于管理产品用户信息，包括但不限于用户 ID，用户名，用户头像等。
- 创建房间管理系统，用于管理产品直播间信息，包括但不限于直播间 ID、直播间名称，直播间主播信息等。
- 服务端生成 UserSig。

> 注意：
>
> - 本文生成 UserSig 的方式，是在客户端根据您填入的 sdkAppId 及 secretKey 生成 userSig，该方式的 secretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通 TUIPusher & TUIPlayer 进行功能调试**。
> - 正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的应用向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

- 参考 `TUIPusher/src/pusher.vue` 及 `TUIPlayer/src/player.vue` 文件，将用户信息、直播间信息、SDKAppId 及 UserSig 等账号信息提交到 vuex 的 store 进行全局存储，您就可以跑通推拉流两个客户端的所

有功能。详细业务流程参见下图：



相关问题

Web 端如何实现美颜功能？

请参见 开启美颜。

Web 端如何实现屏幕共享？

请参见 屏幕分享。

Web 端如何实现云端录制?

1. 开启**云端录制**功能，具体操作请参见 实现云端录制与回放。
2. 开启**云端录制**> **指定用户录制**之后，Web 端可通过在调用 TRTC.createClient 接口时传入 userDefineRecordId 参数开启录制。

Web 端如何实现推流到 CDN ？

Web 端推流到 CDN 请参见 实现推流到 CDN。

Web 端如何实现快直播拉流？

实现快直播拉流的方式是通过 Web SDK 推流到 CDN 之后使用 WebRTC 协议拉流。

注意事项

平台支持

操作系统	浏览器类型	浏览器最低版本要求	TUIPlayer	TUIPusher	TUIPusher 屏幕分享
Mac OS	桌面版 Safari 浏览器	11+	支持	支持	支持（需要 Safari13+ 版本）
Mac OS	桌面版 Chrome 浏览器	56+	支持	支持	支持（需要 Chrome72+ 版本）
Mac OS	桌面版 Firefox 浏览器	56+	支持	支持	支持（需要 Firefox66+ 版本）
Mac OS	桌面版 Edge 浏览器	80+	支持	支持	支持
Mac OS	桌面版微信内嵌网页	-	支持	不支持	不支持
Mac OS	桌面版企业微信内嵌网页	-	支持	不支持	不支持
Windows	桌面版 Chrome 浏览器	56+	支持	支持	支持（需要 Chrome72+ 版本）
Windows	桌面版 QQ 浏览器（极速内核）	10.4+	支持	支持	不支持
Windows	桌面版 Firefox 浏览器	56+	支持	支持	支持（需要 Firefox66+ 版本）
Windows	桌面版 Edge 浏览器	80+	支持	支持	支持

版权所有：腾讯云计算（北京）有限责任公司

操作系统	浏览器类型	浏览器最低版本要求	TUIPlayer	TUIPusher	TUIPusher 屏幕分享
Windows	桌面版微信内嵌网页	-	支持	不支持	不支持
Windows	桌面版企业微信内嵌网页	-	支持	不支持	不支持
iOS	微信内嵌浏览器	-	支持	不支持	不支持
iOS	企业微信内嵌浏览器	-	支持	不支持	不支持
iOS	移动版 Safari 浏览器	-	支持	不支持	不支持
iOS	移动版 Chrome 浏览器	-	支持	不支持	不支持
Android	微信内嵌浏览器	-	支持	不支持	不支持
Android	企业微信浏览器	-	支持	不支持	不支持
Android	移动版 Chrome 浏览器	-	支持	不支持	不支持
Android	移动版 QQ 浏览器	-	支持	不支持	不支持
Android	移动版 Firefox 浏览器	-	支持	不支持	不支持
Android	移动端 UC 浏览器	-	支持（仅支持标准直播观看）	不支持	不支持

域名要求

出于对用户安全、隐私等问题的考虑，浏览器限制网页在 HTTPS 协议下才能正常使用 TUIPusher & TUIPlayer 的全部功能。为确保生产环境用户顺畅接入和体验 TUIPusher & TUIPlayer 的全部功能，请使用 HTTPS 协议访问音视频应用页面。

> 注意：
> 本地开发可以通过 `http://localhost` 协议进行访问。

URL 域名及协议支持情况请参考如下表格：

应用场景	协议	TUIPlayer	TUIPusher	TUIPusher 屏幕分享	备注
生产环境	HTTPS 协议	支持	支持	支持	推荐
生产环境	HTTP 协议	支持	不支持	不支持	-
本地开发环境	`http://localhost`	支持	支持	支持	推荐
本地开发环境	`http://127.0.0.1`	支持	支持	支持	-
本地开发环境	`http://[本机IP]`	支持	不支持	不支持	-

防火墙限制

在使用 TUIPusher & TUIPlayer 时，用户可能因防火墙限制导致无法正常进行音视频通话，请参考 应对防火墙限制相关 将相应端口及域名添加至防火墙白名单中。

结语

在后续的迭代中, TRTC Web 端推拉流组件会逐渐与 iOS、Andriod 等各端连通，并在 Web 端实现观众连麦、高级美颜、自定义布局、转推多平台、上传图片文字音乐等能力，欢迎大家多多使用、提出您的宝贵意见。

如果有任何需要或者反馈，可以联系：colleenyu@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

TUILiveRoom API 查询
TRTCLiveRoom API (iOS)

最近更新时间：2022-07-22 15:13:44

TRTCLiveRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的，支持以下功能：

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

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCLiveRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 视频连麦直播（iOS）。

- TRTC SDK：使用 TRTC SDK 作为低延时直播组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现直播聊天室的功能，同时，通过 IM 消息串联主播间的连麦流程。

TRTCLiveRoom API 概览

SDK 基础函数

API	描述
delegate	设置事件回调。
login	登录。
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（主播调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（主播调用）。
enterRoom	进入房间（观众调用）。
exitRoom	离开房间（观众调用）。
getRoomInfos	获取房间列表的详细信息。
getAnchorList	获取房间内所有的主播列表，enterRoom() 成功后调用才有效。
getAudienceList	获取房间内所有的观众信息，enterRoom() 成功后调用才有效。

推拉流相关接口函数

API	描述
startCameraPreview	开启本地视频的预览画面。
stopCameraPreview	停止本地视频采集及预览。
startPublish	开始直播（推流）。
stopPublish	停止直播（推流）。
startPlay	播放远端视频画面，可以在普通观看和连麦场景中调用。
stopPlay	停止渲染远端视频画面。

主播和观众连麦

API	描述
requestJoinAnchor	观众请求连麦。
responseJoinAnchor	主播处理连麦请求。
kickoutJoinAnchor	主播踢除连麦观众。

主播跨房间 PK

API	描述

API	描述
requestRoomPK	主播请求跨房 PK。
responseRoomPK	主播响应跨房 PK 请求。
quitRoomPK	退出跨房 PK。

音视频控制相关接口函数

API	描述
switchCamera	切换前后摄像头。
setMirror	设置是否镜像展示。
muteLocalAudio	静音本地音频。
muteRemoteAudio	静音远端音频。
muteAllRemoteAudio	静音所有远端音频。

背景音乐音效相关接口函数

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

美颜滤镜相关接口函数

API	描述
getBeautyManager	获取美颜管理对象 TXBeautyManager。

消息发送相关接口函数

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。
sendRoomCustomMsg	发送自定义文本消息。

调试相关接口函数

版权所有：腾讯云计算（北京）有限责任公司

API	描述
showVideoDebugLog	是否在界面中展示debug信息。

TRTCLiveRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	直播房间信息变更回调。

主播和观众进出事件回调

API	描述
onAnchorEnter	收到新主播进房通知。
onAnchorExit	收到主播退房通知。
onAudienceEnter	收到观众进房通知。
onAudienceExit	收到观众退房通知。

主播和观众连麦事件回调

API	描述
onRequestJoinAnchor	主播收到观众连麦请求时的回调。
onKickoutJoinAnchor	连麦观众收到被踢出连麦的通知。

版权所有：腾讯云计算（北京）有限责任公司

主播 PK 事件回调

API	描述
onRequestRoomPK	收到请求跨房 PK 通知。
onQuitRoomPK	收到断开跨房 PK 通知。

消息事件回调

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

SDK 基础函数

delegate

TRTCLiveRoom 事件回调，您可以通过 TRTCLiveRoomDelegate 获得 TRTCLiveRoom 的各种状态通知。

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

> 说明：
> delegate 是 TRTCLiveRoom 的代理回调。

login

登录。

```
/// 登录到组件系统
/// - Parameters:
/// - sdkAppID: 您可以在实时音视频控制台 > 【[应用管理](https://console.tencentcloud.com/trtc/app)】> 应用信息中查看 SDKAppID。
/// - userID: 当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（\_）。
/// - userSig: 腾讯云设计的一种安全保护签名，获取方式请参见 [如何计算及使用 UserSig](https://www.tencentcloud.com/document/product/647/35166)。
/// - config: 全局配置信息，请在登录时初始化，登录之后不可变更。 isAttachedTUIKit 项目中是否引入并使用TUIKit
```

```
/// - callback: 登录回调，成功时 code 为0。
/// - Note:
/// - userSig 建议设定 7 天，能够有效规避 usersign 过期导致的 IM 收发消息失败、TRTC 连麦失败等情况
- (void)loginWithSdkAppID:(int)sdkAppID
userID:(NSString *)userID
userSig:(NSString *)userSig
config:(TRTCLiveRoomConfig *)config
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(login(sdkAppID:userID:userSig:config:callback:));
```

参数如下表所示：

参数	类型	含义
sdkAppID	Int	您可以在实时音视频控制台 >【应用管理】> 应用信息中查看 SDKAppID。
userID	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
config	TRTCLiveRoomConfig	全局配置信息，请在登录时初始化，登录之后不可变更。 • useCDNFirst 属性：用于设置观众观看方式。true 表示普通观众通过 CDN 观看，计费便宜但延时较高。false 表示普通观众通过低延时观看，计费价格介于 CDN 和连麦之间，但延迟可控制在1s以内。 • CDNPlayDomain 属性：在 useCDNFirst 设置为 true 时才会生效，用于指定 CDN 观看的播放域名，您可以登录直播控制台 >【域名管理】页面中进行设置。
callback	(_ code: Int, _ message: String?) -> Void	登录回调，成功时 code 为0。

logout

登出。

```
// 退出登录
/// - Parameter callback: 登出回调，成功时 code 为0
- (void)logout:(Callback _Nullable)callback
NS_SWIFT_NAME(logout(_:));
```

参数如下表所示：

参数	类型	含义
callback	(_ code: Int, _ message: String?) -> Void	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
/// 设置用户信息，您设置的用户信息会被存储于腾讯云 IM 云服务中。
/// - Parameters:
/// - name: 用户昵称
/// - avatarURL: 用户头像地址
/// - callback: 个人信息设置回调，成功时 code 为0
- (void)setSelfProfileWithName:(NSString *)name
avatarURL:(NSString * _Nullable)avatarURL
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(setSelfProfile(name:avatarURL:callback:));
```

参数如下表所示：

参数	类型	含义
name	String	昵称。
avatarURL	String	头像地址。
callback	(_ code: Int, _ message: String?) -> Void	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（主播调用）。

```
/// 创建房间（主播调用），若房间不存在，系统将自动创建一个新房间。
/// 主播开播的正常调用流程是：
/// 1.【主播】调用 startCameraPreview() 打开摄像头预览，此时可以调整美颜参数。
/// 2.【主播】调用 createRoom() 创建直播间，房间创建成功与否会通过 callback 通知给主播。
/// 3.【主播】调用 startPublish() 开始推流。
/// - Parameters:
/// - roomID: 房间标识，需要由您分配并进行统一管理。多个 roomid 可以汇总成一个直播间列表，腾讯云暂不提供直播间列表的管理服务，请自行管理您的直播间列表。
```

```
/// - roomParam: TRTCCreateRoomParam | 房间信息，用于房间描述的信息，例如房间名称，封面信息等。如果房间列表和房间信息都由您自行管理，可忽略该参数。
/// - callback: 进入房间的结果回调，成功时 code 为0。
/// - Note:
/// - 主播开始直播的时候调用，可重复创建自己已创建过的房间。
- (void)createRoomWithRoomID:(UInt32)roomID
roomParam:(TRTCCreateRoomParam *)roomParam
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

参数如下表所示：

参数	类型	含义
roomID	UInt32	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个直播间列表，腾讯云暂不提供直播间列表的管理服务，请自行管理您的直播间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息，例如房间名称，封面信息等。如果房间列表和房间信息都由您自行管理，可忽略该参数。
callback	(_ code: Int, _ message: String?) -> Void	创建房间的结果回调，成功时 code 为0。

主播开播的正常调用流程如下：

1. 【主播】调用 `startCameraPreview()` 打开摄像头预览，此时可以调整美颜参数。
2. 【主播】调用 `createRoom()` 创建直播间，房间创建成功与否会通过 callback 通知给主播。
3. 【主播】调用 `starPublish()` 开始推流。

destroyRoom

销毁房间（主播调用）。主播在创建房间后，可以调用该函数来销毁房间。

```
/// 销毁房间（主播调用）
/// 主播在创建房间后，可以调用这个函数来销毁房间。
/// - Parameter callback: 销毁房间的结果回调，成功时 code 为0。
/// - Note:
/// - 主播在创建房间后，可以调用该函数来销毁房间。
- (void)destroyRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(destroyRoom(callback:));
```

参数如下表所示：

参数	类型	含义

参数	类型	含义
callback	(_ code: Int, _ message: String?) -> Void	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（观众调用）。

```
/// 进入房间（观众调用）
/// 观众观看直播的正常调用流程是：
/// 1.【观众】向您的服务端获取最新的直播间列表，其中有多个直播间的 roomid 和房间信息。
/// 2.【观众】观众选择一个直播间以后，调用 enterRoom() 进入该房间。
/// 3.【观众】如果您的服务器所管理的房间列表中包含每一个房间的主播 userID，则可以直接在 enterRoom() 成功后调用 startPlay(userID) 即可播放主播的画面。
/// 如果您管理的房间列表只有 roomid 也没有关系，观众在 enterRoom() 成功后很快会收到来自 TRTCLiveRoomDelegate 中的 onAnchorEnter(userID) 回调。
/// 此时使用回调中的 userID 调用 startPlay(userID) 即可播放主播的画面。
/// - Parameters:
/// - roomID: 房间标识。
/// - useCDNFirst: 是否优先使用CDN播放
/// - cdnDomain: CDN域名
/// - callback: 进入房间的结果回调，成功时 code 为0。
/// - Note:
/// - 观众进入直播房间的时候调用
/// - 主播不可调用这个接口进入自己已创建的房间，而要用createRoom
- (void)enterRoomWithRoomID:(UInt32)roomID
useCDNFirst:(BOOL)useCDNFirst
cdnDomain:(NSString * _Nullable)cdnDomain
callback:(Callback)callback
NS_SWIFT_NAME(enterRoom(roomID:useCDNFirst:cdnDomain:callback:));
```

参数如下表所示：

参数	类型	含义
roomID	UInt32	房间标识。
callback	(_ code: Int, _ message: String?) -> Void	进入房间的结果回调，成功时 code 为0。

观众观看直播的正常调用流程如下：

1. 【观众】向您的服务端获取最新的直播间列表，可能包含多个直播间的 roomID 和房间信息。
2. 【观众】观众选择一个直播间，并调用 `enterRoom()` 进入该房间。
3. 【观众】调用 `startPlay(userID)` 并传入主播的 userID 开始播放。

- 若直播间列表已包含主播端的 userID 信息，观众端可直接调用 `startPlay(userID)` 即可开始播放。
- 若在进房前暂未获取主播的 userID，观众端在进房后会收到 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userID)` 的事件回调，该回调中携带主播的 userID 信息，再调用 `startPlay(userID)` 即可播放。

exitRoom

离开房间。

```
/// 退出房间（观众调用）
/// - Parameter callback: 退出房间的结果回调，成功时 code 为0。
/// - Note:
/// - 观众离开直播房间的时候调用
/// - 主播不可调用这个接口离开房间
- (void)exitRoom:(Callback _Nullable)callback
NS_SWIFT_NAME(exitRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	(_ code: Int, _ message: String?) -> Void	退出房间的结果回调，成功时 code 为0。

getRoomInfos

获取房间列表的详细信息，房间信息是主播在创建 `createRoom()` 时通过 roomInfo 设置的。

> 说明：
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
/// 获取房间列表的详细信息
/// 其中的信息是主播在创建 createRoom() 时通过 roomInfo 设置进来的，如果房间列表和房间信息都由您自行管理，可忽略该函数。
/// - Parameter roomIDs: 房间号列表
/// - Parameter callback: 房间详细信息回调
- (void)getRoomInfosWithRoomIDs:(NSArray<NSNumber *> *)roomIDs
callback:(RoomInfoCallback _Nullable)callback
NS_SWIFT_NAME(getRoomInfos(roomIDs:callback:));
```

参数如下表所示：

参数	类型	含义
roomIDs	[UInt32]	房间号列表。
callback	(_ code: Int, _ message: String?, _ roomList: [TRTCLiveRoomInfo]) -> Void	房间详细信息回调。

getAnchorList

获取房间内所有的主播列表，`enterRoom()` 成功后调用才有效。

```
/// 获取房间内所有的主播列表，enterRoom() 成功后调用才有效。
/// - Parameter callback: 用户详细信息回调
- (void)getAnchorList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAnchorList(callback:));
```

参数如下表所示：

参数	类型	含义
callback	(_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> Void	用户详细信息回调。

getAudienceList

获取房间内所有的观众信息，`enterRoom()` 成功后调用才有效。

```
/// 获取房间内所有的观众信息，enterRoom() 成功后调用才有效。
/// - Parameter callback: 用户详细信息回调
- (void)getAudienceList:(UserListCallback _Nullable)callback
NS_SWIFT_NAME(getAudienceList(callback:));
```

参数如下表所示：

参数	类型	含义
callback	(_ code: Int, _ message: String, _ userList: [TRTCLiveUserInfo]) -> Void	用户详细信息回调。

推拉流相关接口函数

startCameraPreview

开启本地视频的预览画面。

```
/// 开启本地视频的预览画面
/// - Parameters:
/// - frontCamera: true：前置摄像头；false：后置摄像头。
/// - view: 承载视频画面的控件。
/// - callback: 操作回调。
- (void)startCameraPreviewWithFrontCamera:(BOOL)frontCamera
view:(UIView *)view
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startCameraPreview(frontCamera:view:callback:));
```

参数如下表所示：

参数	类型	含义
frontCamera	Bool	true：前置摄像头；false：后置摄像头。
view	UIView	承载视频画面的控件。
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

stopCameraPreview

停止本地视频采集及预览。

```
/// 停止本地视频采集及预览
- (void)stopCameraPreview;
```

startPublish

开始直播（推流），适用于以下场景：

- 主播开播的时候调用
- 观众开始连麦时调用

```
/// 开始直播（推流），适用于如下两种场景：
/// 1. 主播开播的时候调用
/// 2. 观众开始连麦时调用
/// - Parameters:
/// - streamID: 用于绑定直播 CDN 的 streamId，如果您希望您的观众通过直播 CDN 进行观看，需要指定当前主播的直播 streamId。
/// - callback: 操作回调
- (void)startPublishWithStreamID:(NSString *)streamID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPublish(streamID:callback:));
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
streamID	String	用于绑定直播 CDN 的 streamID，如果您希望观众通过直播 CDN 进行观看，需要指定当前主播的直播 streamID。
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

stopPublish

停止直播（推流），适用于以下场景：

- 主播结束直播时调用。
- 观众结束连麦时调用。

```
/// 停止直播（推流），适用于如下两种场景：
/// 1. 主播结束直播时调用
/// 2. 观众结束连麦时调用
/// - Parameter callback: 操作回调。
- (void)stopPublish:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPublish(callback:));
```

参数如下表所示：

参数	类型	含义
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

startPlay

播放远端视频画面，可以在普通观看和连麦场景中调用。

```
/// 播放远端视频画面，可以在普通观看和连麦场景中调用
/// 【普通观看场景】
/// 1. 如果您的服务器所管理的房间列表中包含每一个房间的主播 userID，则可以直接在 enterRoom() 成功后调用 startPlay(userID) 即可播放主播的画面。
/// 2. 如果您管理的房间列表只有 roomid 也没有关系，观众在 enterRoom() 成功后很快会收到来自 TRTCLiveRoomDelegate 中的 onAnchorEnter(userID) 回调。
/// 此时使用回调中的 userID 调用 startPlay(userID) 即可播放主播的画面。
/// 【直播连麦场景】
/// 发起连麦后，主播会收到来自 TRTCLiveRoomDelegate 中的 onAnchorEnter(userID) 回调，此时使用回调中的 userID 调用 startPlay(userID) 即可播放连麦画面。
```

```
/// - Parameters:
/// - userID: 需要观看的用户 ID。
/// - view: 承载视频画面的 view 控件。
/// - callback: 操作回调。
- (void)startPlayWithUserID:(NSString *)userID
view:(UIView *)view
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(startPlay(userID:view:callback:));
```

参数如下表所示：

参数	类型	含义
userID	String	需要观看的用户 ID。
view	UIView	承载视频画面的 view 控件。
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

普通观看场景

- 若直播间列表已包含主播端的 userID 信息，观众端可以直接在 `enterRoom()` 成功后调用 `startPlay(userID)` 播放主播的画面。
- 若在进房前暂未获取主播的 userID，观众端在进房后会收到 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userID)` 的事件回调，该回调中携带主播的 userID 信息，再调用 `startPlay(userID)` 即可播放主播的画面。

直播连麦场景

发起连麦后，主播会收到来自 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userID)` 回调，此时使用回调中的 userID 调用 `startPlay(userID)` 即可播放连麦画面。

stopPlay

停止渲染远端视频画面。需在 `onAnchorExit()` 回调时，调用该接口。

```
/// 停止渲染远端视频画面
/// - Parameters:
/// - userID: 对方的用户信息。
/// - callback: 操作回调。
/// - Note:
/// - 在 onAnchorExit 回调时，调用这个接口
- (void)stopPlayWithUserID:(NSString *)userID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(stopPlay(userID:callback:));
```

参数如下表所示：

参数	类型	含义
userID	String	对方的用户信息。
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

主播和观众连麦

requestJoinAnchor

观众请求连麦。

```
/// 观众端请求连麦
/// - Parameters:
/// - reason: 连麦请求原因。
/// - responseCallback: 请求连麦的回调。
/// - Note: 观众发起请求后，主播端会收到`onRequestJoinAnchor`回调
- (void)requestJoinAnchor:(NSString *)reason
timeout:(double)timeout
responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestJoinAnchor(reason:timeout:responseCallback:));
```

参数如下表所示：

参数	类型	含义
reason	String	连麦原因。
timeout	long	主播响应回调。
responseCallback	(_ agreed: Bool, _ reason: String?) -> Void	主播响应回调。

主播和观众的连麦流程如下：

1. 【观众】调用 `requestJoinAnchor()` 向主播发起连麦请求。
2. 【主播】会收到 `TRTCLiveRoomDelegate` 的 `onRequestJoinAnchor()` 回调通知。
3. 【主播】调用 `responseJoinAnchor()` 决定是否接受来自观众的连麦请求。
4. 【观众】会收到 responseCallback 回调通知，该通知会携带主播的处理结果。
5. 【观众】如果请求被同意，则调用 `startCameraPreview()` 开启本地摄像头。
6. 【观众】调用 `startPublish()` 正式进入推流状态。
7. 【主播】一旦观众进入连麦状态，主播会收到 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知。

版权所有：腾讯云计算（北京）有限责任公司

8. 【主播】主播调用 `startPlay()` 即可看到连麦观众的视频画面。
9. 【观众】如果直播间里已有其他观众正在跟主播连麦，新加入的连麦观众会收到 `onAnchorEnter()` 通知，调用 `startPlay()` 播放其他连麦者的视频画面。

responseJoinAnchor

主播处理连麦请求。主播在收到 `TRTCLiveRoomDelegate` 的 `onRequestJoinAnchor()` 回调后，需要调用该接口来处理观众的连麦请求。

```
/// 主播回复观众连麦请求
/// - Parameters:
/// - user: 观众 ID。
/// - agree: true：同意；false：拒绝。
/// - reason: 同意/拒绝连麦的原因描述。
/// - Note: 主播回复后，观众端会收到`requestJoinAnchor`传入的`responseCallback`回调
- (void)responseJoinAnchor:(NSString *)userID
agree:(BOOL)agree
reason:(NSString *)reason
NS_SWIFT_NAME(responseJoinAnchor(userID:agree:reason:));
```

参数如下表所示：

参数	类型	含义
userID	String	观众 ID。
agree	Bool	true：同意；false：拒绝。
reason	String?	同意/拒绝连麦的原因描述。

kickoutJoinAnchor

主播踢除连麦观众。主播调用此接口踢除连麦观众后，被踢连麦观众会收到 `TRTCLiveRoomDelegate` 的 `onKickoutJoinAnchor()` 回调通知。

```
/// 主播踢除连麦观众
/// - Parameters:
/// - userID: 连麦观众 ID。
/// - callback: 操作回调。
/// - Note: 主播调用此接口踢除连麦观众后，被踢连麦观众会收到 trtcLiveRoomOnKickoutJoinAnchor() 回调通知
- (void)kickoutJoinAnchor:(NSString *)userID
callback:(Callback _Nullable)callback
NS_SWIFT_NAME(kickoutJoinAnchor(userID:callback:));
```

参数如下表所示：

参数	类型	含义
userID	String	连麦观众 ID。
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

主播跨房间 PK

requestRoomPK

主播请求跨房 PK。

```
/// 主播请求跨房 PK
/// - Parameters:
/// - roomID: 被邀约房间 ID。
/// - userID: 被邀约主播 ID。
/// - responseCallback: 请求跨房 PK 的结果回调。
/// - Note: 发起请求后，对方主播会收到 `onRequestRoomPK` 回调
- (void)requestRoomPKWithRoomID:(UInt32)roomID
userID:(NSString *)userID
responseCallback:(ResponseCallback _Nullable)responseCallback
NS_SWIFT_NAME(requestRoomPK(roomID:userID:responseCallback:));
```

参数如下表所示：

参数	类型	含义
roomID	UInt32	被邀约房间 ID。
userID	String	被邀约主播 ID。
responseCallback	(_ agreed: Bool, _ reason: String?) -> Void	请求跨房 PK 的结果回调。

主播和主播之间可以跨房间 PK，两个正在直播中的主播 A 和 B 之间的跨房 PK 流程如下：

1. 【主播 A】调用 `requestRoomPK()` 向主播 B 发起连麦请求。
2. 【主播 B】会收到 `TRTCLiveRoomDelegate` 的 `onRequestRoomPK()` 回调通知。
3. 【主播 B】调用 `responseRoomPK()` 决定是否接受主播 A 的 PK 请求。
4. 【主播 B】如果接受主播 A 的请求，等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 来显示主播 A 的视频画面。
5. 【主播 A】会收到 `responseCallback` 回调通知，该通知会携带来自主播 B 的处理结果。

版权所有：腾讯云计算（北京）有限责任公司

6. 【主播 A】如果请求被同意，等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 显示主播 B 的视频画面。

responseRoomPK

主播响应跨房 PK 请求。主播响应后，对方主播会收到 `requestRoomPK` 传入的 `responseCallback` 回调。

```
/// 响应跨房 PK 请求
/// 主播响应其他房间主播的 PK 请求。
/// - Parameters:
/// - user: 发起 PK 请求的主播 ID
/// - agree: true：同意；false：拒绝
/// - reason: 同意/拒绝 PK 的原因描述
/// - Note: 主播回复后，对方主播会收到 `requestRoomPK` 传入的 `responseCallback` 回调
- (void)responseRoomPKWithUserID:(NSString *)userID
agree:(BOOL)agree
reason:(NSString *)reason
NS_SWIFT_NAME(responseRoomPK(userID:agree:reason:));
```

参数如下表所示：

参数	类型	含义
userID	String	发起 PK 请求的主播 ID。
agree	Bool	true：同意；false：拒绝。
reason	String?	同意/拒绝 PK 的原因描述。

quitRoomPK

退出跨房 PK。PK 中的任何一个主播退出跨房 PK 状态后，另一个主播会收到 `TRTCLiveRoomDelegate` 的 `trtcLiveRoomOnQuitRoomPK()` 回调通知。

```
/// 主播退出跨房 PK
/// - Parameter callback: 退出跨房 PK 的结果回调
/// - Note: 当两个主播中的任何一个退出跨房 PK 状态后，另一个主播会收到 `trtcLiveRoomOnQuitRoomPK` 回调通知。
- (void)quitRoomPK:(Callback _Nullable)callback
NS_SWIFT_NAME(quitRoomPK(callback:));
```

参数如下表所示：

参数	类型	含义

参数	类型	含义
callback	(_ code: Int, _ message: String?) -> Void	操作回调。

音视频控制相关接口函数

switchCamera

切换前后摄像头。

```
/// 切换前后摄像头
- (void)switchCamera;
```

setMirror

设置是否镜像展示。

```
/// 设置是否镜像展示
/// - Parameter isMirror: 开启/关闭镜像。
- (void)setMirror:(BOOL)isMirror
NS_SWIFT_NAME(setMirror(isMirror:));
```

参数如下表所示：

参数	类型	含义
isMirror	Bool	开启/关闭镜像。

muteLocalAudio

静音本地音频。

```
/// 静音本地音频。
/// - Parameter isMuted: true：开启静音；false：关闭静音。
- (void)muteLocalAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteLocalAudio(isMuted:));
```

参数如下表所示：

参数	类型	含义
isMuted	Bool	true：开启静音；false：关闭静音。

版权所有：腾讯云计算（北京）有限责任公司

muteRemoteAudio

静音远端音频。

```
/// 静音远端音频
/// - Parameters:
/// - userID: 远端的用户ID。
/// - isMuted: true：开启静音；false：关闭静音。
- (void)muteRemoteAudioWithUserID:(NSString *)userID isMuted:(BOOL)isMuted
NS_SWIFT_NAME(muteRemoteAudio(userID:isMuted:));
```

参数如下表所示：

参数	类型	含义
userID	String	远端的用户 ID。
isMuted	Bool	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音所有远端音频。

```
/// 静音所有远端音频
/// - Parameter isMuted: true：开启静音；false：关闭静音。
- (void)muteAllRemoteAudio:(BOOL)isMuted
NS_SWIFT_NAME(muteAllRemoteAudio(_:));
```

参数如下表所示：

参数	类型	含义
isMuted	Bool	true：开启静音；false：关闭静音。

setAudioQuality

设置音频质量

```
/// 设置音频质量，支持的值为1 2 3，代表低中高
/// - Parameter quality 音频质量
- (void)setAudioQuality:(NSInteger)quality
NS_SWIFT_NAME(setAudioiQuality(quality:));
```

参数如下表所示：

参数	类型	含义
quality	NSInteger	1：语音；2：标准； 3：音乐。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

```
/// 获取音效管理对象
- (TXAudioEffectManager *)getAudioEffectManager;
```

美颜滤镜相关接口函数

getBeautyManager

获取美颜管理对象 TXBeautyManager。

```
/* 获取美颜管理对象 TXBeautyManager
*
* 通过美颜管理，您可以使用以下功能：
* - 设置"美颜风格"、“美白”、“红润”、“大眼”、“瘦脸”、“v脸”、“下巴”、“短脸”、“小鼻”、“亮眼”、“白牙”、“祛眼袋”、“祛皱纹”、“祛法令纹”等美容效果。
* - 调整“发际线”、“眼间距”、“眼角”、“嘴形”、“鼻翼”、“鼻子位置”、“嘴唇厚度”、“脸型”
* - 设置人脸挂件（素材）等动态效果
* - 添加美妆
* - 进行手势识别
*/
- (TXBeautyManager *)getBeautyManager;
```

通过美颜管理，您可以使用以下功能：

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

版权所有：腾讯云计算（北京）有限责任公司

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
/// 发送文本消息，房间内所有成员都可见
/// - Parameters:
/// - message: 文本消息。
/// - callback: 发送回调。
- (void)sendRoomTextMsg:(NSString *)message callback:(Callback _Nullable)callback
NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	(_ code: Int, _ message: String?) -> Void	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
/// 发送自定义消息
/// - Parameters:
/// - command: 命令字，由开发者自定义，主要用于区分不同消息类型
/// - message: 本文消息。
/// - callback: 发送回调。
- (void)sendRoomCustomMsgWithCommand:(NSString *)command message:(NSString *)mess
age callback:(Callback _Nullable)callback
NS_SWIFT_NAME(sendRoomCustomMsg(command:message:callback:));
```

参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	(_ code: Int, _ message: String?) -> Void	发送结果回调。

调试相关接口函数

showVideoDebugLog

是否在界面中展示 debug 信息。

```
/// 是否在界面中展示debug信息
/// - Parameter isShow: 开启/关闭 Debug 信息显示。
- (void)showVideoDebugLog:(BOOL)isShow
NS_SWIFT_NAME(showVideoDebugLog(_:));
```

参数如下表所示：

参数	类型	含义
isShow	Bool	开启/关闭 Debug 信息显示。

TRTCLiveRoomDelegate事件回调

通用事件回调

onError

错误回调。

> 说明：
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onError:(NSInteger)code
message:(NSString *)message
NS_SWIFT_NAME(trtcLiveRoom(_:onError:message:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
code	Int	错误码。
message	String?	错误信息。

onWarning

警告回调。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onWarning:(NSInteger)code
message:(NSString *)message
NS_SWIFT_NAME(trtcLiveRoom(_:onWarning:message:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
code	Int	错误码 TRTCWarningCode。
message	String?	警告信息。

onDebugLog

Log 回调。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onDebugLog:(NSString *)log
NS_SWIFT_NAME(trtcLiveRoom(_:onDebugLog:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
log	String	日志信息。

房间事件回调

onRoomDestroy

版权所有：腾讯云计算（北京）有限责任公司

房间被销毁的回调。主播退房时，房间内的所有用户都会收到此通知。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRoomDestroy:(NSString *)roomID
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomDestroy:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
roomID	String	房间 ID。

onRoomInfoChange

直播房间信息变更回调。多用于直播连麦、PK下房间状态变化通知场景。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRoomInfoChange:(TRTCLiveRoomInfo *)info
NS_SWIFT_NAME(trtcLiveRoom(_:onRoomInfoChange:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
info	TRTCLiveRoomInfo	房间信息。

主播和观众进出事件回调

onAnchorEnter

收到新主播进房通知。连麦观众和跨房 PK 主播进房后观众会收到新主播的进房事件，您可以调用 `TRTCLiveRoom` 的 `startPlay()` 显示该主播的视频画面。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAnchorEnter:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorEnter:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
userID	String	新进房用户 ID。

onAnchorExit

收到主播退房通知。房间内的主播（和连麦中的观众）会收到新主播的退房事件，您可以调用 `TRTCLiveRoom` 的 `stopPlay()` 关闭该主播的视频画面。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAnchorExit:(NSString *)userID
NS_SWIFT_NAME(trtcLiveRoom(_:onAnchorExit:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
userID	String	退房用户 ID。

onAudienceEnter

收到观众进房通知。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAudienceEnter:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceEnter:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
user	TRTCLiveUserInfo	进房观众信息。

onAudienceExit

收到观众退房通知。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onAudienceExit:(TRTCLiveUserInfo *)user
```

```
NS_SWIFT_NAME(trtcLiveRoom(_:onAudienceExit:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
user	TRTCLiveUserInfo	退房观众信息。

主播和观众连麦事件回调

onRequestJoinAnchor

主播收到观众连麦请求时的回调。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRequestJoinAnchor:(TRTCLiveUserInfo *)user
reason:(NSString * _Nullable)reason
timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestJoinAnchor:reason:timeout:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
user	TRTCLiveUserInfo	请求连麦观众信息。
reason	String?	连麦原因描述。
timeout	Double	处理请求的超时时间。

onKickoutJoinAnchor

连麦观众收到被踢出连麦的通知。连麦观众收到被主播踢除连麦的消息，您需要调用 `TRTCLiveRoom` 的 `stopPublish()` 退出连麦。

```
- (void)trtcLiveRoomOnKickoutJoinAnchor:(TRTCLiveRoom *)liveRoom
NS_SWIFT_NAME(trtcLiveRoomOnKickoutJoinAnchor(_:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。

主播 PK 事件回调

onRequestRoomPK

收到请求跨房 PK 通知。主播收到其他房间主播的 PK 请求，如果同意 PK ，您需要等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 来播放邀约主播的流。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRequestRoomPK:(TRTCLiveUserInfo *)user
timeout:(double)timeout
NS_SWIFT_NAME(trtcLiveRoom(_:onRequestRoomPK:timeout:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
user	TRTCLiveUserInfo	发起跨房连麦的主播信息。
timeout	Double	处理请求的超时时间。

onQuitRoomPK

收到断开跨房 PK 通知。

```
- (void)trtcLiveRoomOnQuitRoomPK:(TRTCLiveRoom *)liveRoom
NS_SWIFT_NAME(trtcLiveRoomOnQuitRoomPK(_:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomTextMsg:(NSString *)message
fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomTextMsg:fromUser:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
message	String	文本消息。
user	TRTCLiveUserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
- (void)trtcLiveRoom:(TRTCLiveRoom *)trtcLiveRoom
onRecvRoomCustomMsgWithCommand:(NSString *)command
message:(NSString *)message
fromUser:(TRTCLiveUserInfo *)user
NS_SWIFT_NAME(trtcLiveRoom(_:onRecvRoomCustomMsg:message:fromUser:));
```

参数如下表所示：

参数	类型	含义
trtcLiveRoom	TRTCLiveRoomImpl	当前 TRTCLiveRoom 组件实例。
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
user	TRTCLiveUserInfo	发送者用户信息。

版权所有：腾讯云计算（北京）有限责任公司

TRTCLiveRoom API (Android)

最近更新时间：2024-03-11 10:27:03

TRTCLiveRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的，支持以下功能：
主播创建新的直播间开播，观众进入直播间观看。
主播和观众进行视频连麦互动。
两个不同房间的主播 PK 互动。
支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。
说明：
TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。
TRTCLiveRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 视频连麦直播（Android）。
TRTC SDK：使用 TRTC SDK 作为低延时直播组件。
IM SDK：使用 IM SDK 的 AVChatroom 实现直播聊天室的功能，同时，通过 IM 消息串联主播间的连麦流程。

TRTCLiveRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateHandler	设置事件回调所在的线程。
login	登录。
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（主播调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（主播调用）。
enterRoom	进入房间（观众调用）。
exitRoom	离开房间（观众调用）。
getRoomInfos	获取房间列表的详细信息。
getAnchorList	获取房间内所有的主播列表，enterRoom() 成功后调用才有效。
getAudienceList	获取房间内所有的观众信息，enterRoom() 成功后调用才有效。

推拉流相关接口函数

API	描述
startCameraPreview	开启本地视频的预览画面。
stopCameraPreview	停止本地视频采集及预览。
startPublish	开始直播（推流）。
stopPublish	停止直播（推流）。
startPlay	播放远端视频画面，可以在普通观看和连麦场景中调用。
stopPlay	停止渲染远端视频画面。

主播和观众连麦

API	描述
requestJoinAnchor	观众请求连麦。
responseJoinAnchor	主播处理连麦请求。
kickoutJoinAnchor	主播踢除连麦观众。

主播跨房间 PK

API	描述
requestRoomPK	主播请求跨房 PK。

版权所有：腾讯云计算（北京）有限责任公司

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

音视频控制相关接口函数

API	描述
switchCamera	切换前后摄像头。
setMirror	设置是否镜像展示。
muteLocalAudio	静音本地音频。
muteRemoteAudio	静音远端音频。
muteAllRemoteAudio	静音所有远端音频。

背景音乐音效相关接口函数

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

美颜滤镜相关接口函数

API	描述
getBeautyManager	获取美颜管理对象 TXBeautyManager。

消息发送相关接口函数

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。
sendRoomCustomMsg	发送自定义文本消息。

调试相关接口函数

API	描述
showVideoDebugLog	是否在界面中展示 debug 信息。

版权所有：腾讯云计算（北京）有限责任公司

TRTCLiveRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	直播房间信息变更回调。

主播和观众进出事件回调

API	描述
onAnchorEnter	收到新主播进房通知。
onAnchorExit	收到主播退房通知。
onAudienceEnter	收到观众进房通知。
onAudienceExit	收到观众退房通知。

主播和观众连麦事件回调

API	描述
onRequestJoinAnchor	主播收到观众连麦请求时的回调。
onKickoutJoinAnchor	连麦观众收到被踢出连麦的通知。

主播 PK 事件回调

API	描述

版权所有：腾讯云计算（北京）有限责任公司

onRequestRoomPK	收到请求跨房 PK 通知。
onQuitRoomPK	收到断开跨房 PK 通知。

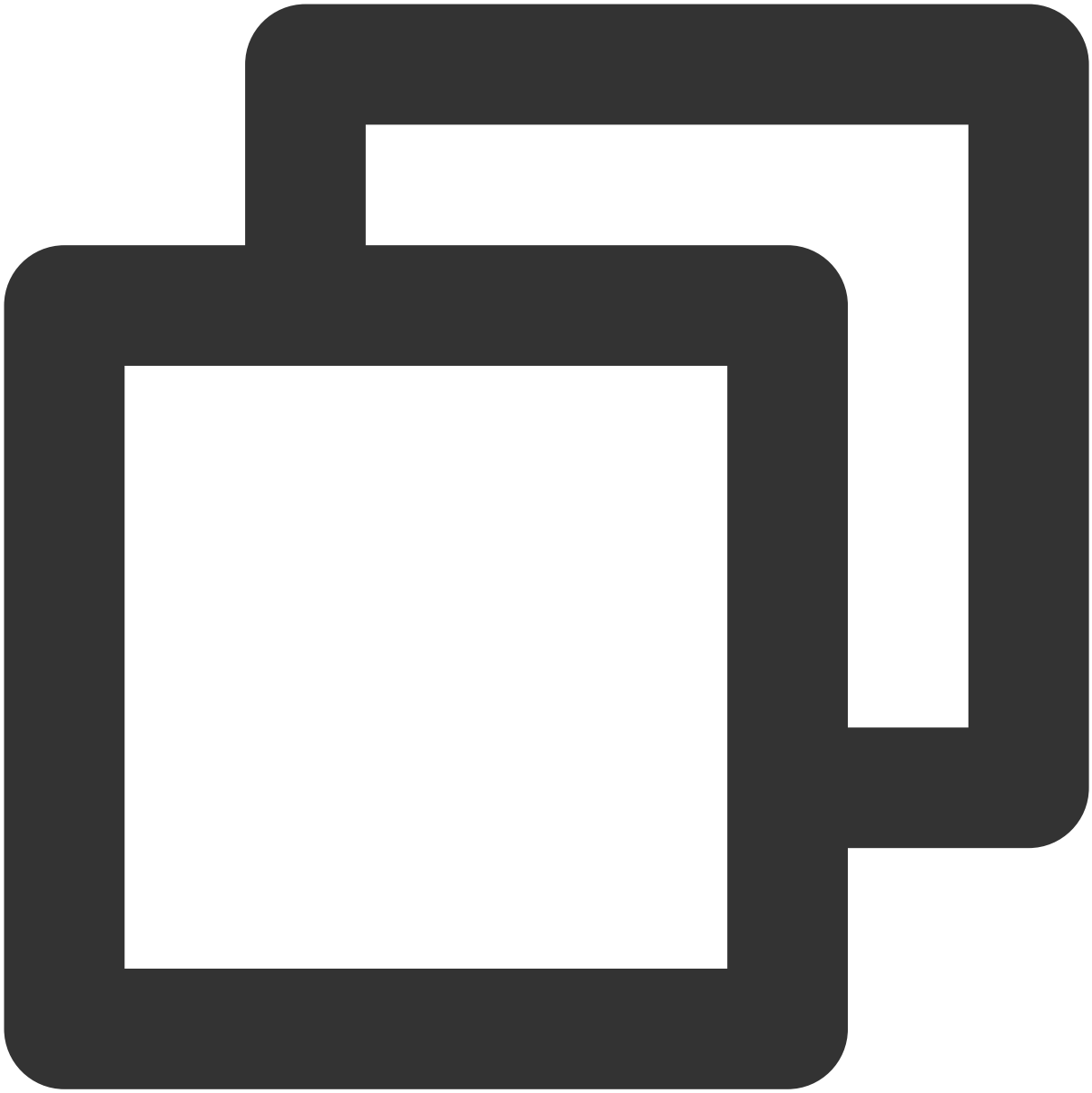
消息事件回调

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

SDK 基础函数

sharedInstance

获取 TRTCLiveRoom 单例对象。



```
public static synchronized TRTCLiveRoom sharedInstance(Context context);
```

参数如下表所示：

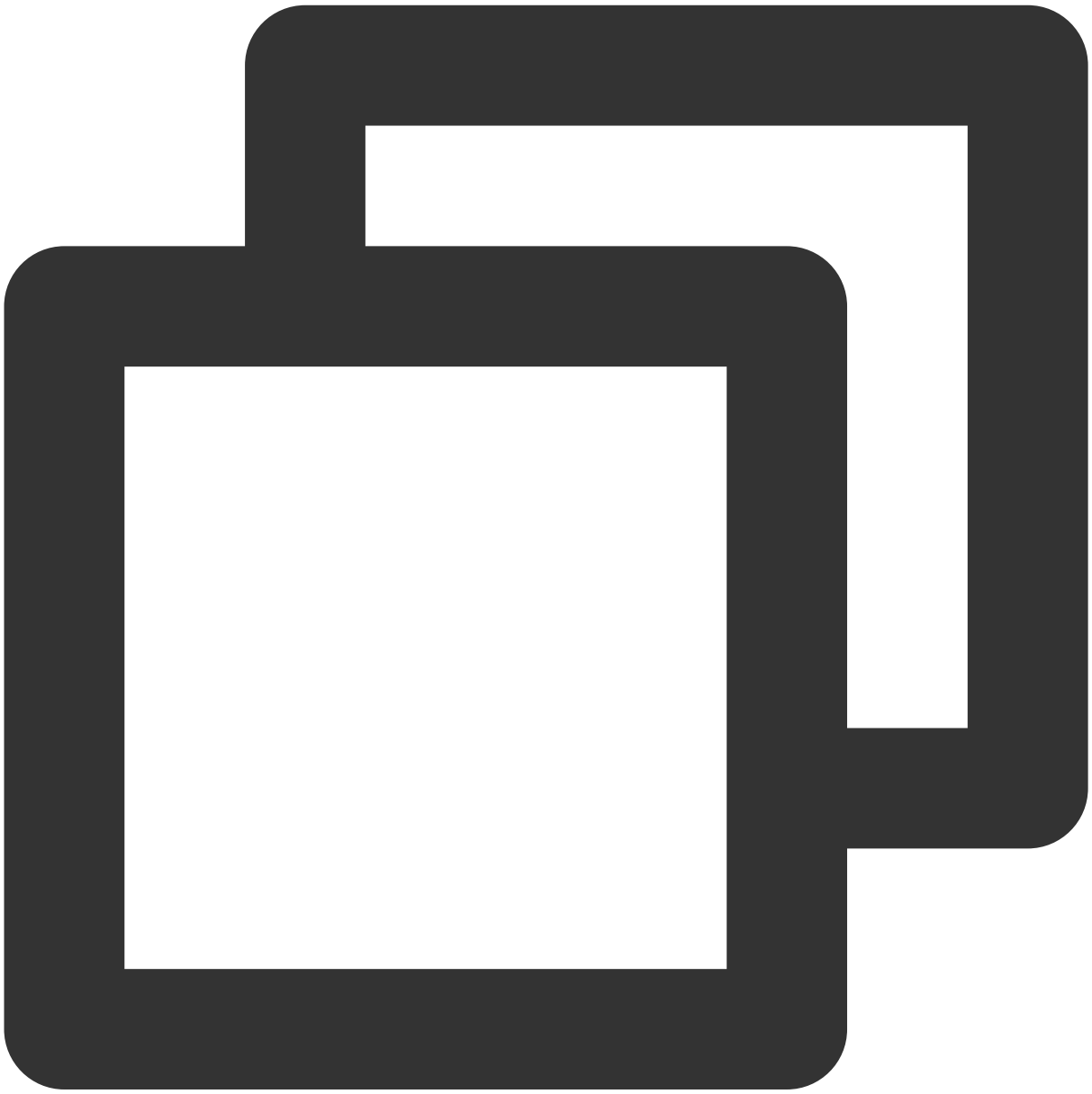
参数	类型	含义
context	Context	Android 上下文，内部会转为 ApplicationContext 用于系统 API 调用

destroySharedInstance

销毁 TRTCLiveRoom 单例对象。

说明：

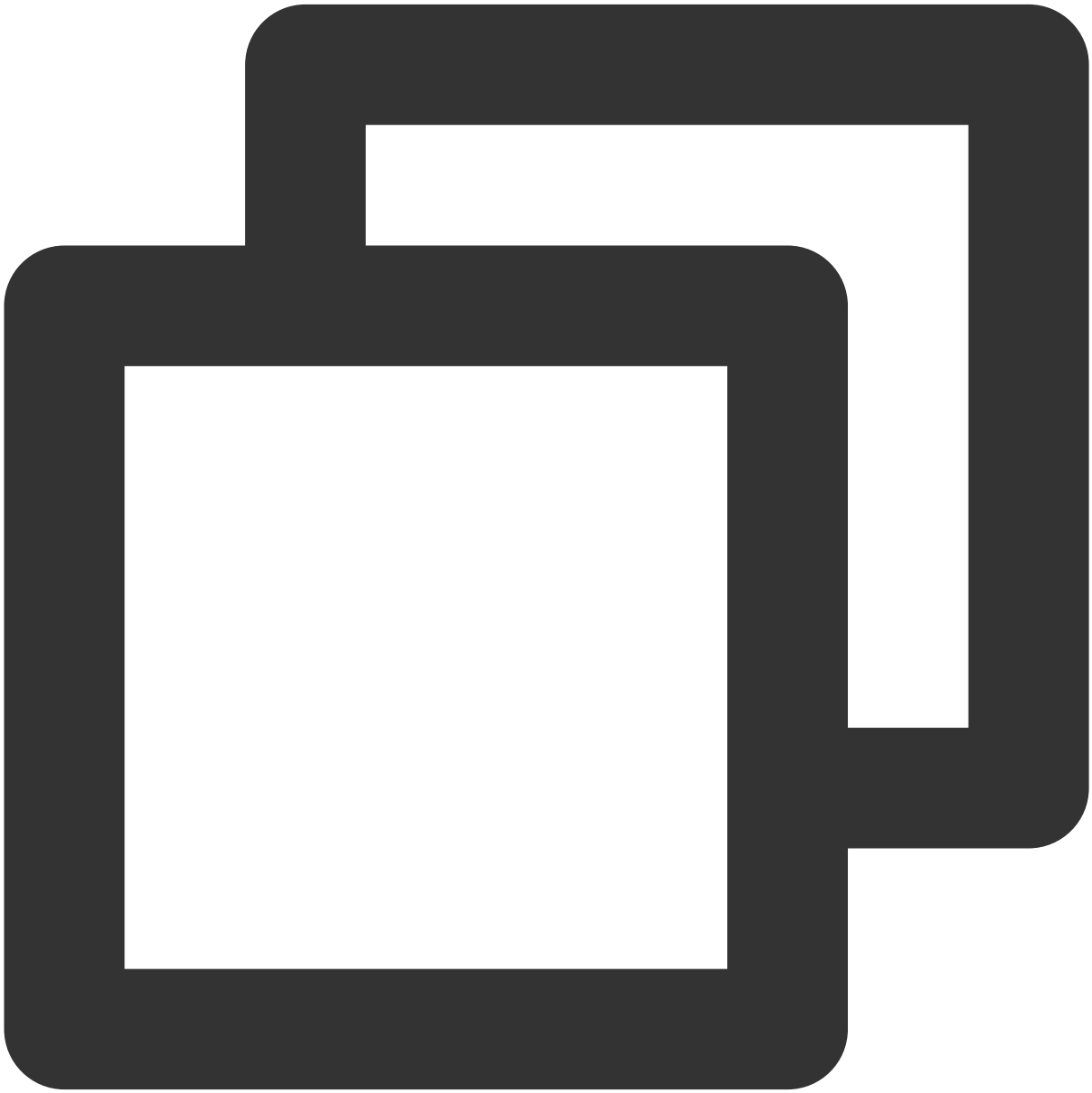
销毁实例后，外部缓存的 TRTCLiveRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。



```
public static void destroySharedInstance();
```

setDelegate

TRTCLiveRoom 事件回调，您可以通过 TRTCLiveRoomDelegate 获得 TRTCLiveRoom 的各种状态通知。



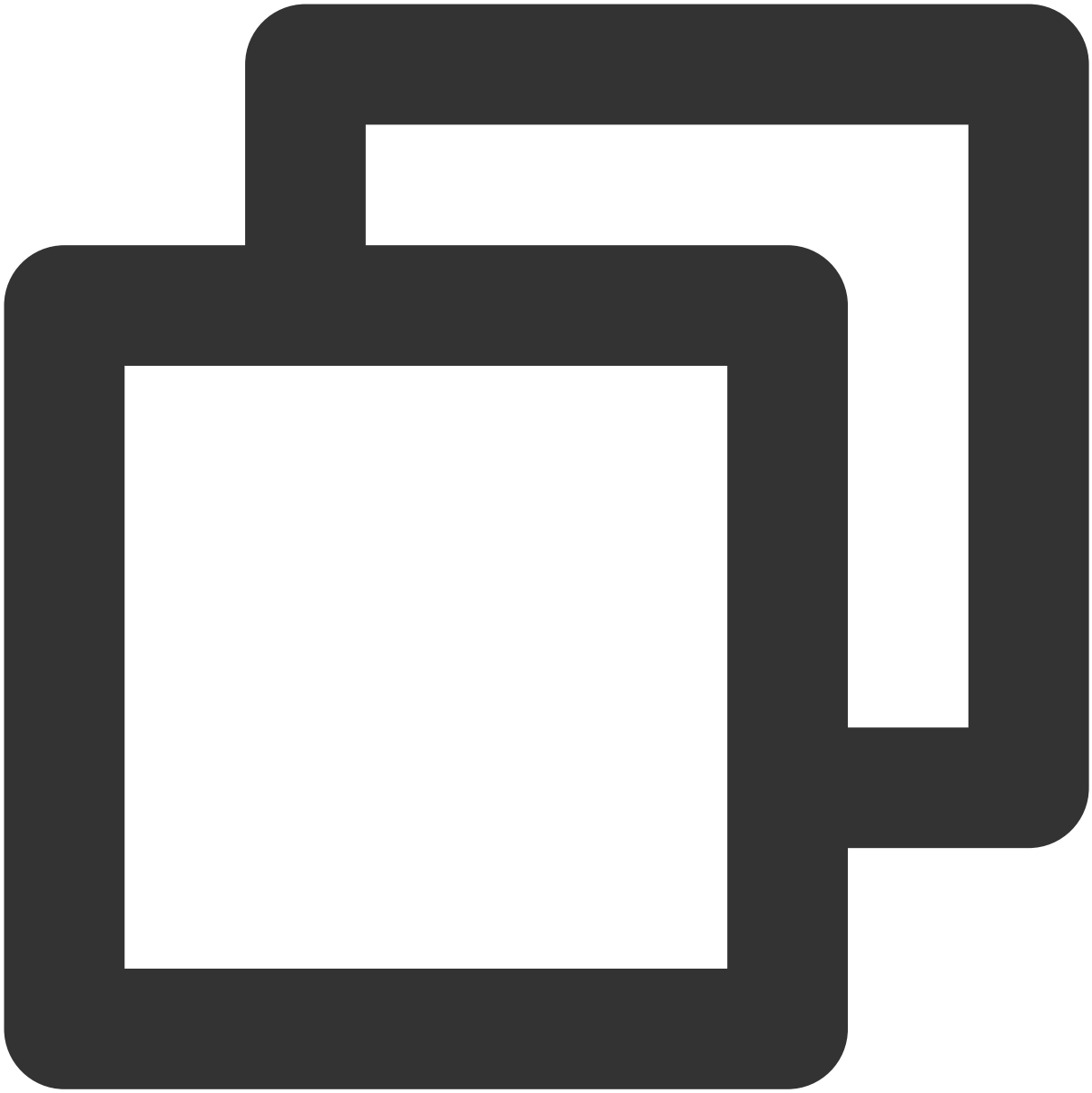
```
public abstract void setDelegate(TRTCLiveRoomDelegate delegate);
```

说明：

setDelegate 是 TRTCLiveRoom 的代理回调。

setDelegateHandler

设置事件回调所在的线程。



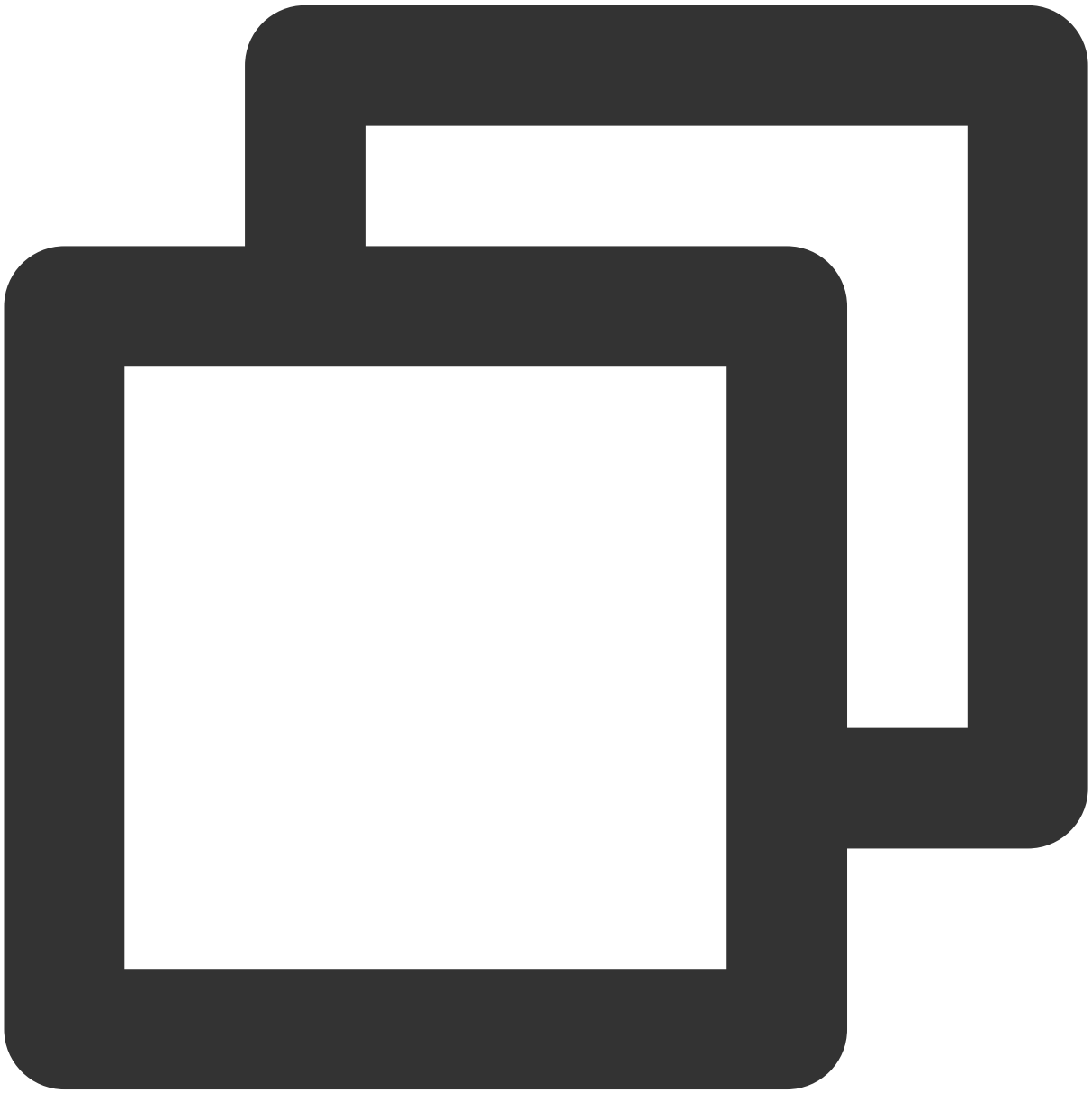
```
public abstract void setDelegateHandler(Handler handler);
```

参数如下表所示：

参数	类型	含义
handler	Handler	TRTCLiveRoom 中的各种状态通知回调会通过该 handler 通知给您，请勿与 setDelegate 混用。

login

登录。



```
public abstract void login(int sdkAppId,
 String userId, String userSig,
 TRTCLiveRoomDef.TRTCLiveRoomConfig config,
 TRTCLiveRoomCallback.ActionCallback callback);
```

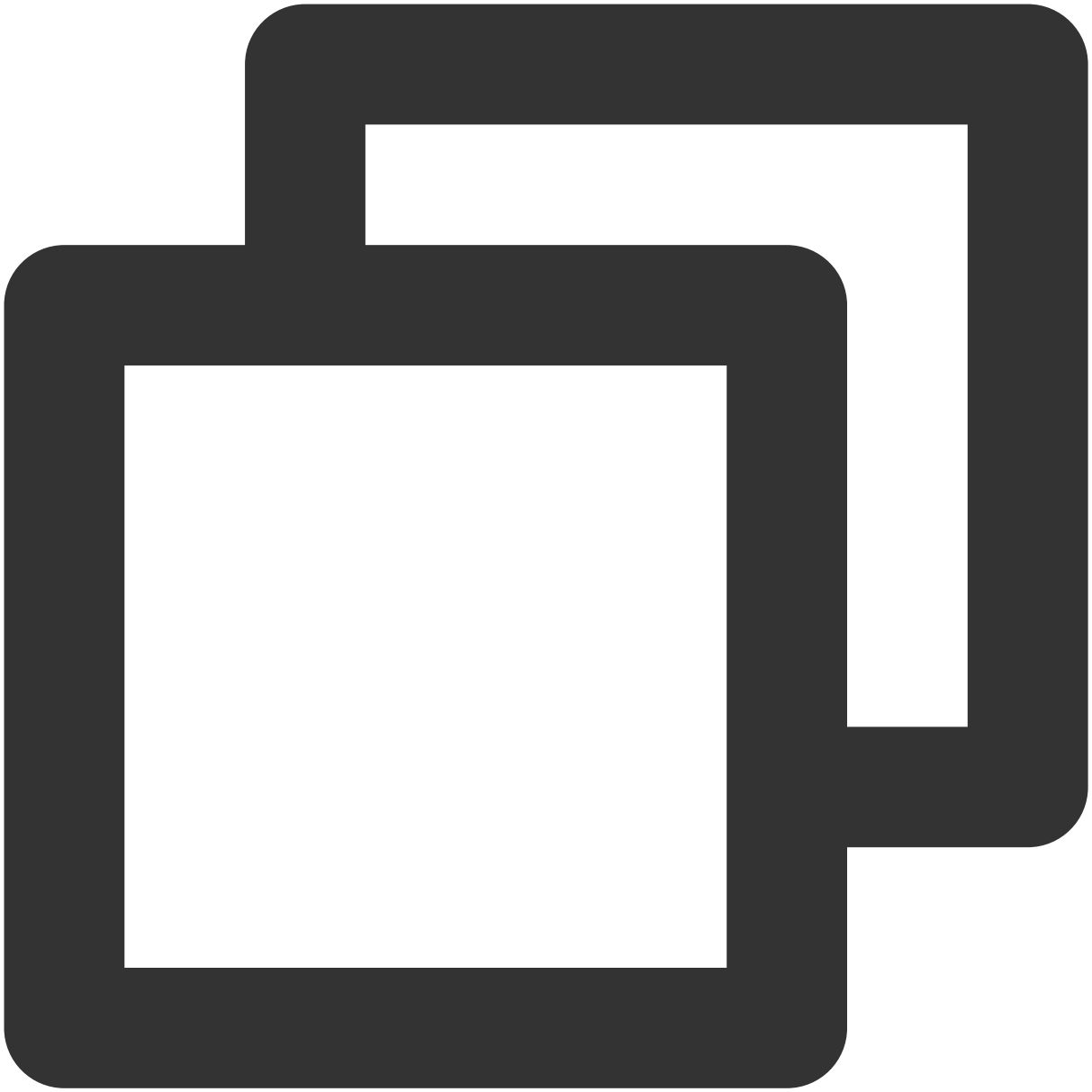
参数如下表所示：

参数	类型	含义

sdkAppId	int	您可以在实时音视频控制台 > 应用管理 > 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
config	TRTCLiveRoomConfig	全局配置信息，请在登录时初始化，登录之后不可变更。 useCDNFirst 属性：用于设置观众观看方式。true 表示普通观众通过 CDN 观看，计费便宜但延时较高。false 表示普通观众通过低延时观看，计费价格介于 CDN 和连麦之间，但延迟可控制在1s以内。 CDNPlayDomain 属性：在 useCDNFirst 设置为 true 时才会生效，用于指定 CDN 观看的播放域名，您可以登录直播控制台 >【域名管理】页面中进行设置。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。



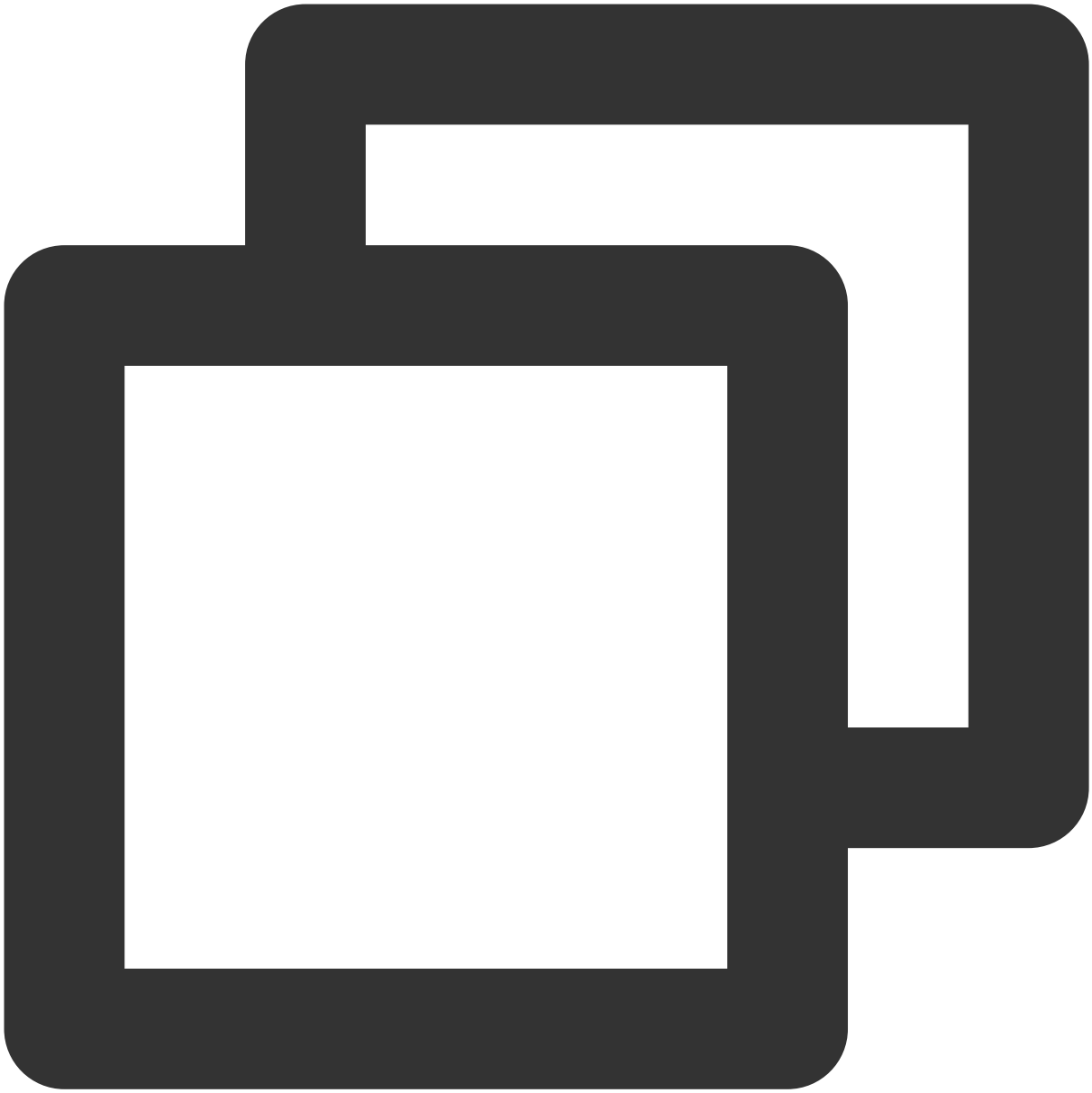
```
public abstract void logout(TRTCLiveRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。



```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCLiveRoom
```

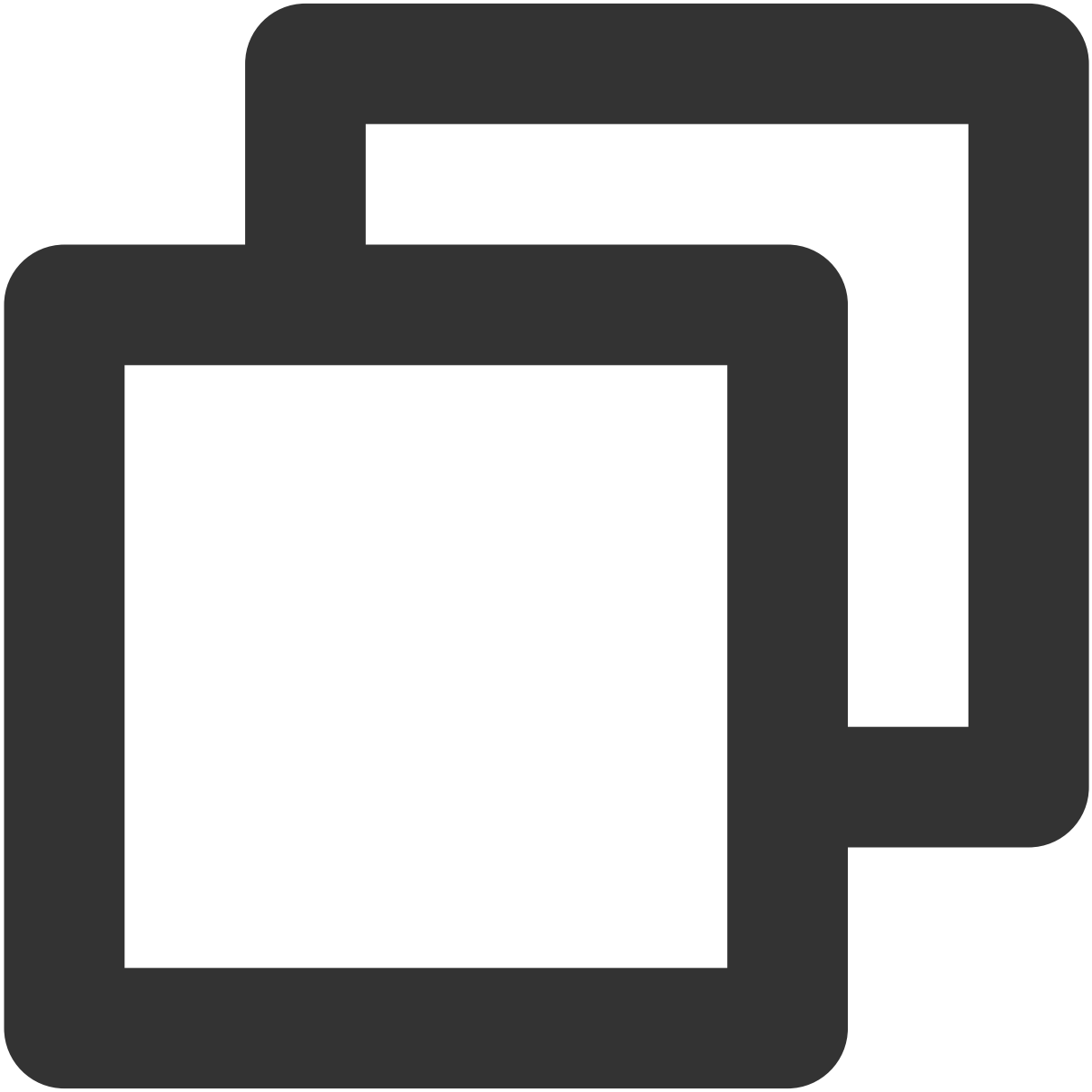
参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（主播调用）。



```
public abstract void createRoom(int roomId, TRTCLiveRoomDef.TRTCCreateRoomParam roo
```

参数如下表所示：

参数	类型	含义

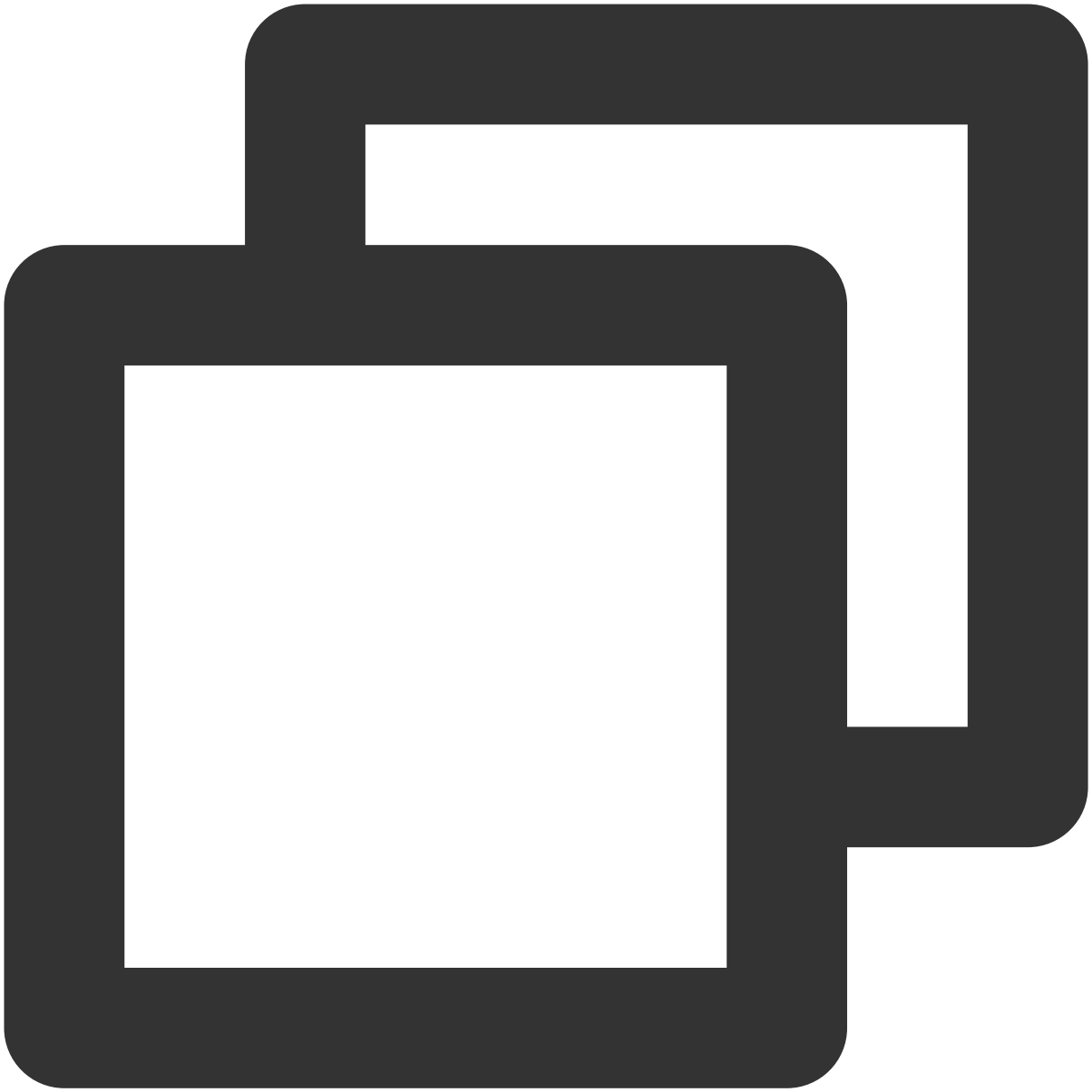
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个直播间列表，腾讯云暂不提供直播间列表的管理服务，请自行管理您的直播间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息，例如房间名称，封面信息等。如果房间列表和房间信息都由您的服务器自行管理，可忽略该参数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

主播开播的正常调用流程如下：

1. 【主播】调用 `startCameraPreview()` 打开摄像头预览，此时可以调整美颜参数。
2. 【主播】调用 `createRoom()` 创建直播间，房间创建成功与否会通过 ActionCallback 通知给主播。
3. 【主播】调用 `starPublish()` 开始推流。

destroyRoom

销毁房间（主播调用）。主播在创建房间后，可以调用该函数来销毁房间。



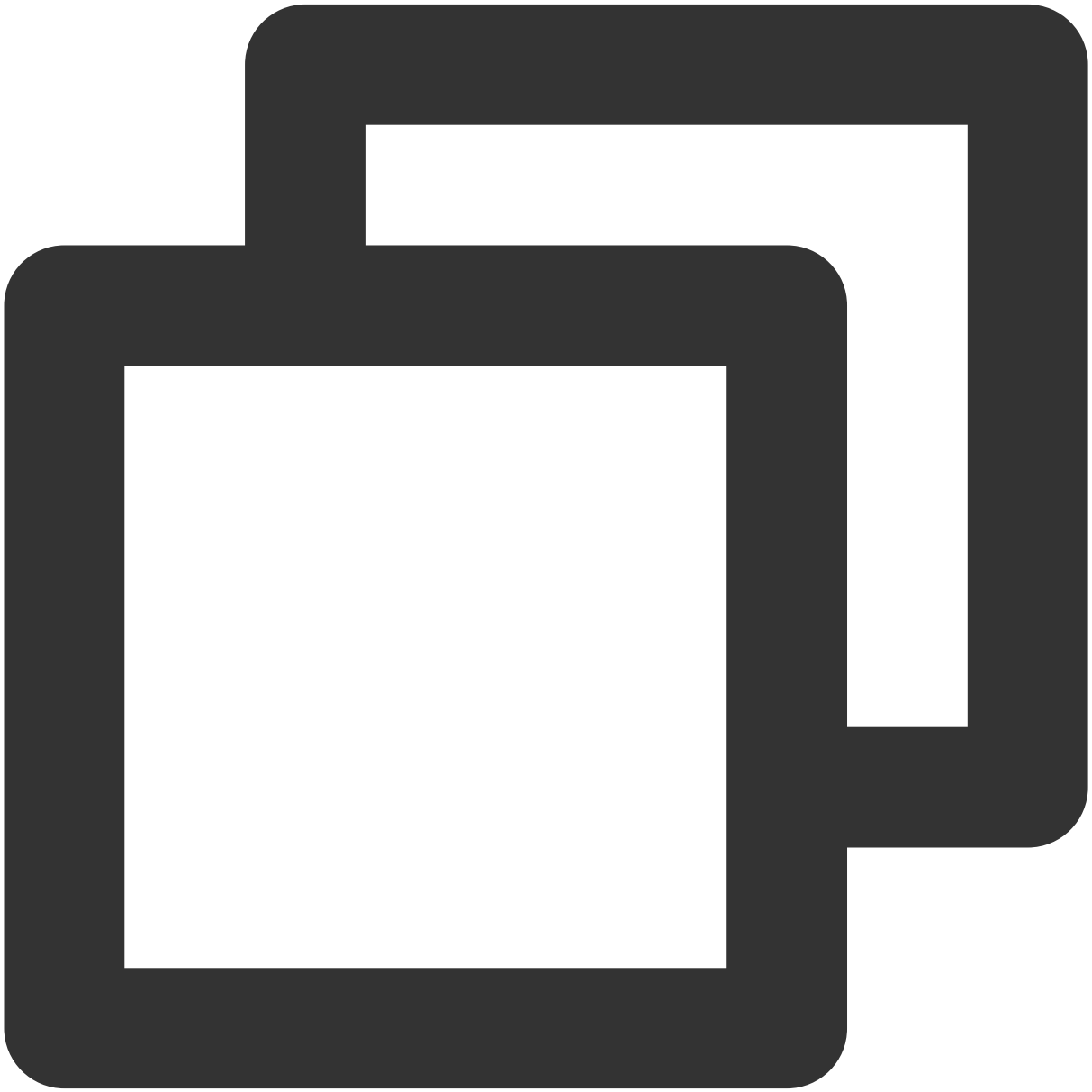
```
public abstract void destroyRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（观众调用）。



```
public abstract void enterRoom(int roomId, TRTCLiveRoomCallback.ActionCallback call
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

观众观看直播的正常调用流程如下：

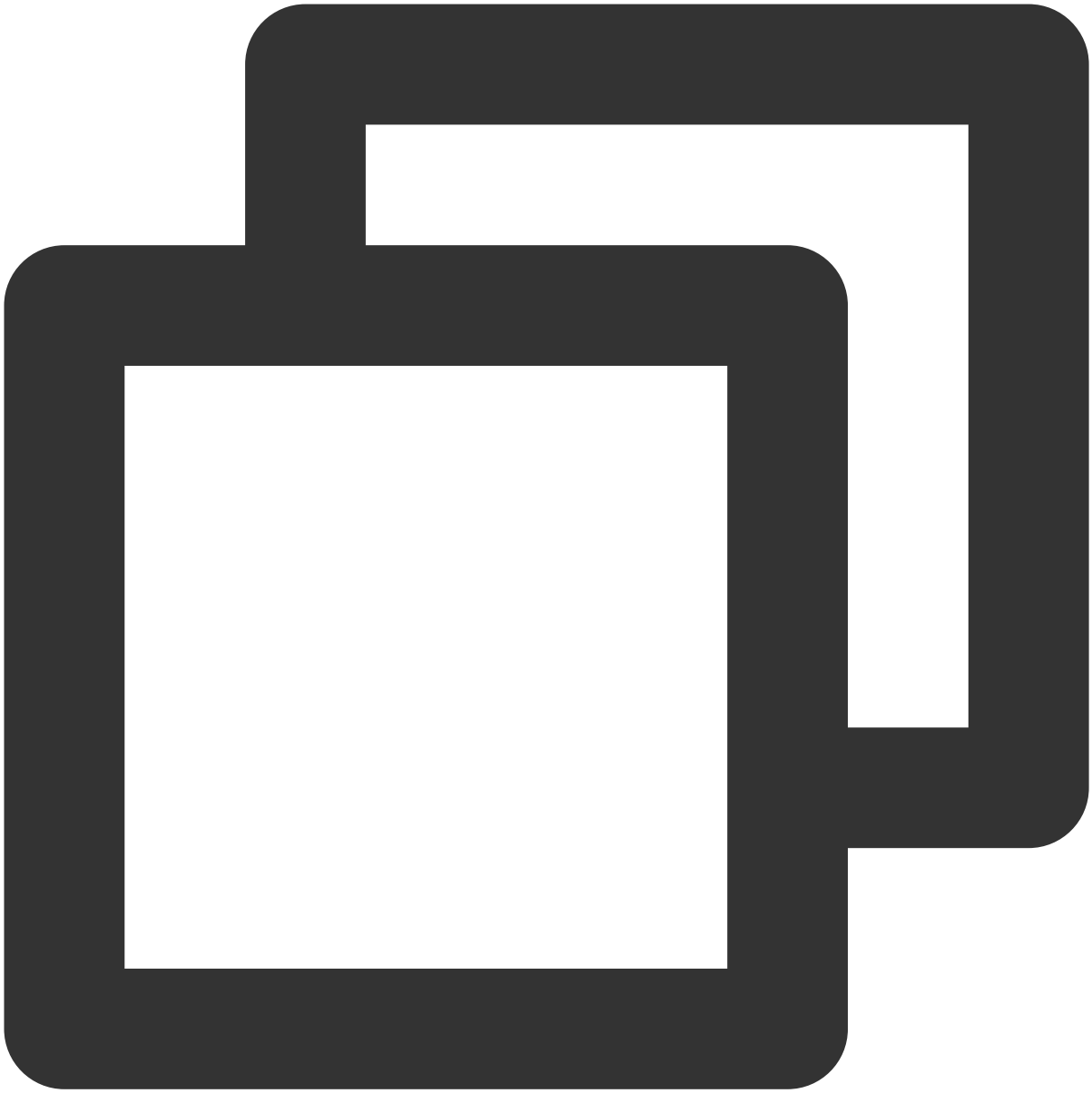
1. 【观众】向您的服务端获取最新的直播间列表，可能包含多个直播间的 roomID 和房间信息。
2. 【观众】观众选择一个直播间，并调用 `enterRoom()` 进入该房间。
3. 【观众】调用 `startPlay(userId)` 并传入主播的 userId 开始播放。

若直播间列表已包含主播端的 userId 信息，观众端可直接调用 `startPlay(userId)` 即可开始播放。

若在进房前暂未获取主播的 userId，观众端在进房后会收到 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userId)` 的事件回调，该回调中携带主播的 userId 信息，再调用 `startPlay(userId)` 即可播放。

exitRoom

离开房间。



```
public abstract void exitRoom(TRTCLiveRoomCallback.ActionCallback callback);
```

参数如下表所示：

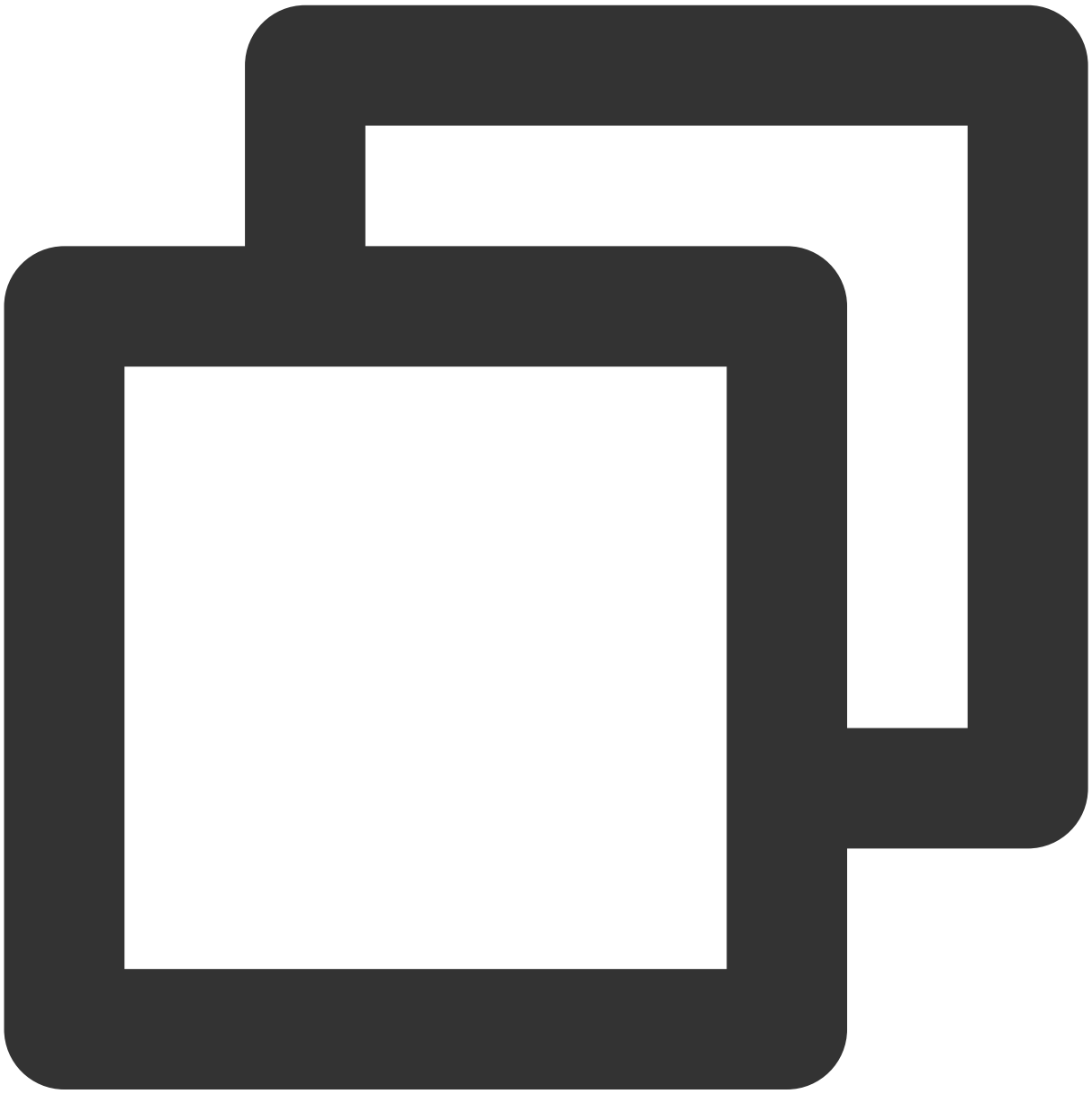
参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfos

获取房间列表的详细信息，房间信息是主播在创建 `createRoom()` 时通过 roomInfo 设置的。

说明：

如果房间列表和房间信息都由您自行管理，可忽略该函数。



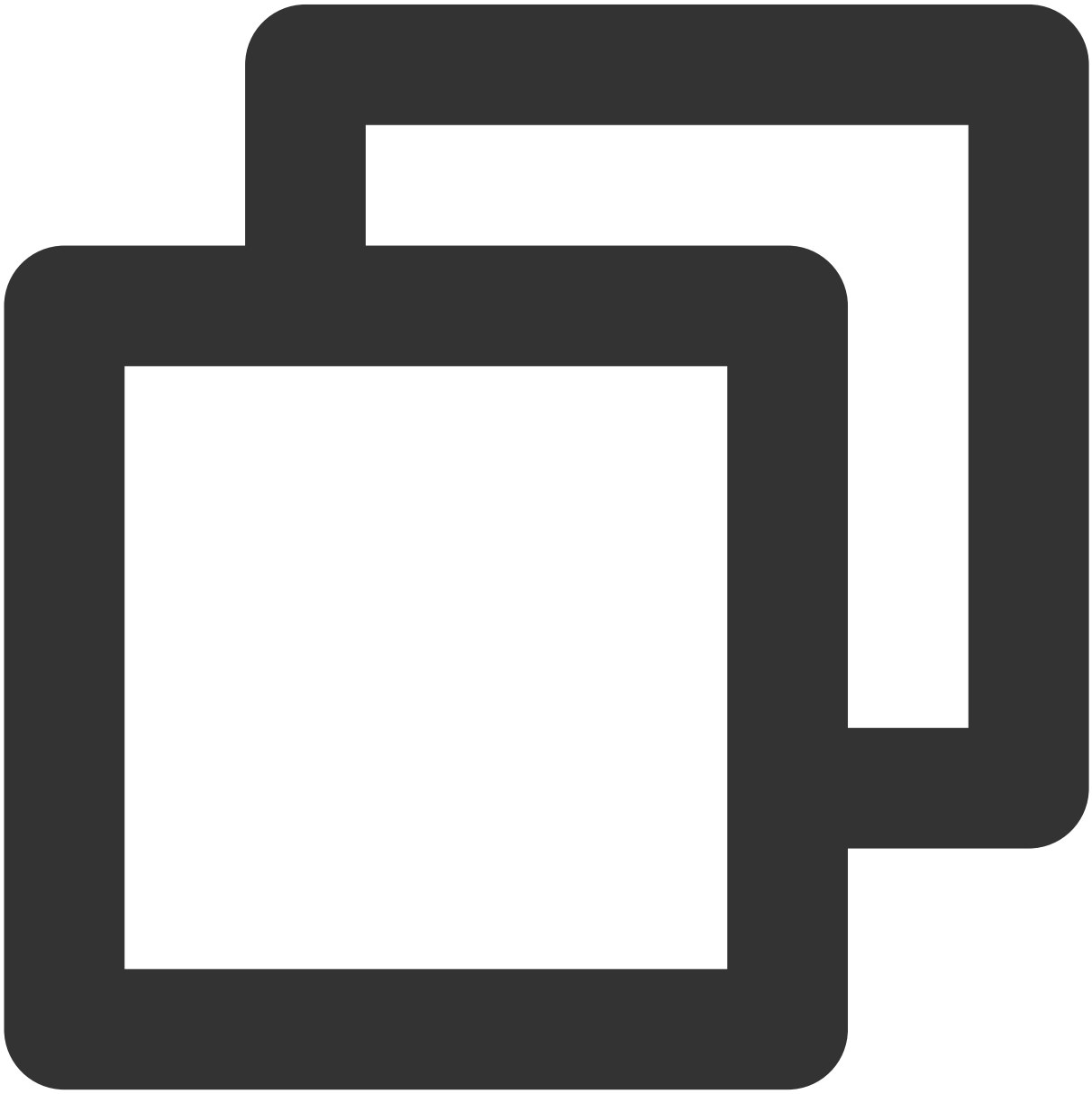
```
public abstract void getRoomInfos(List<Integer> roomIdList, TRTCLiveRoomCallback.Ro
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getAnchorList

获取房间内所有的主播列表，`enterRoom()` 成功后调用才有效。



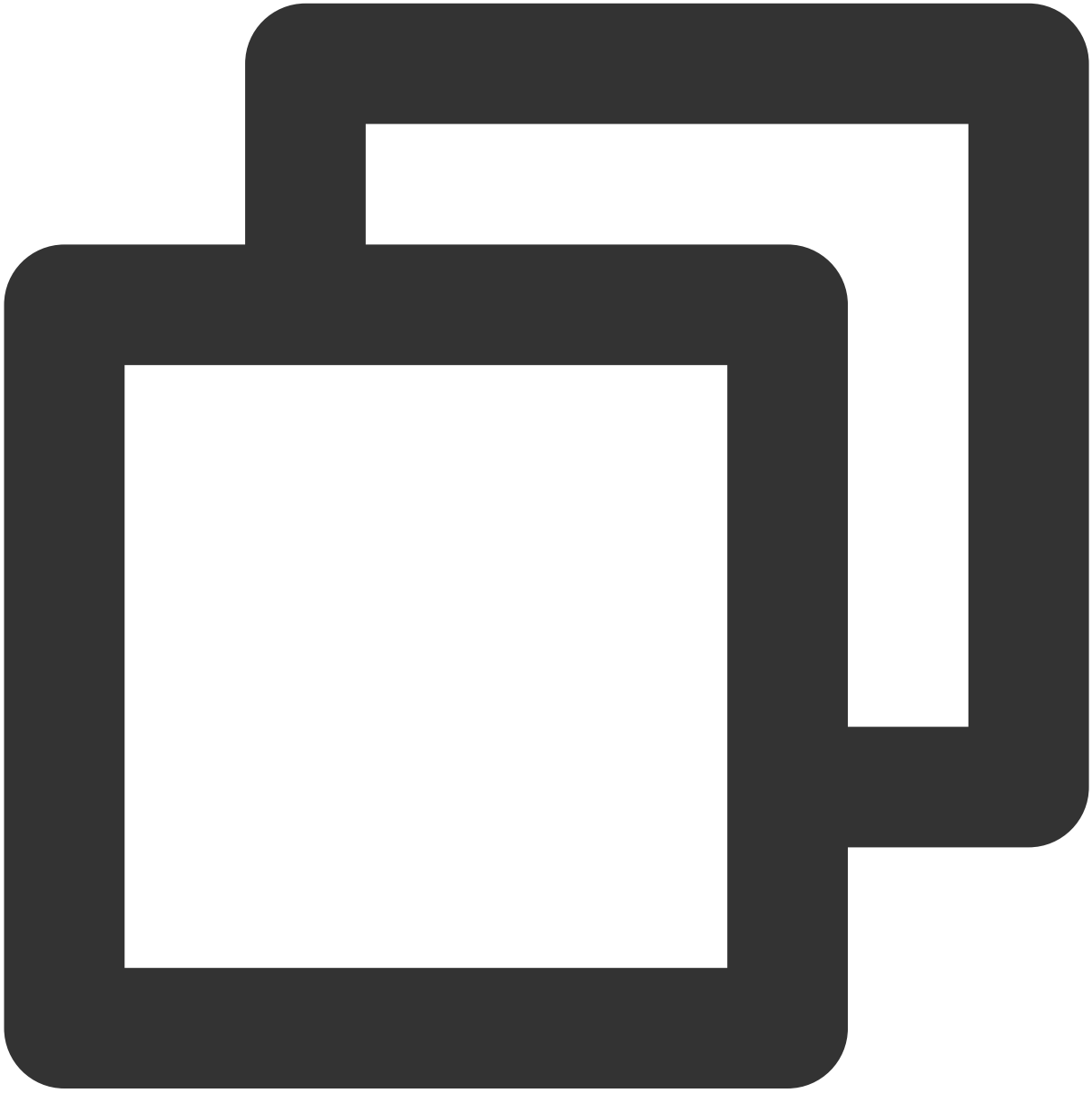
```
public abstract void getAnchorList(TRTCLiveRoomCallback.UserListCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	UserListCallback	用户详细信息回调。

getAudienceList

获取房间内所有的观众信息， `enterRoom()` 成功后调用才有效。



```
public abstract void getAudienceList(TRTCLiveRoomCallback.UserListCallback callback
```

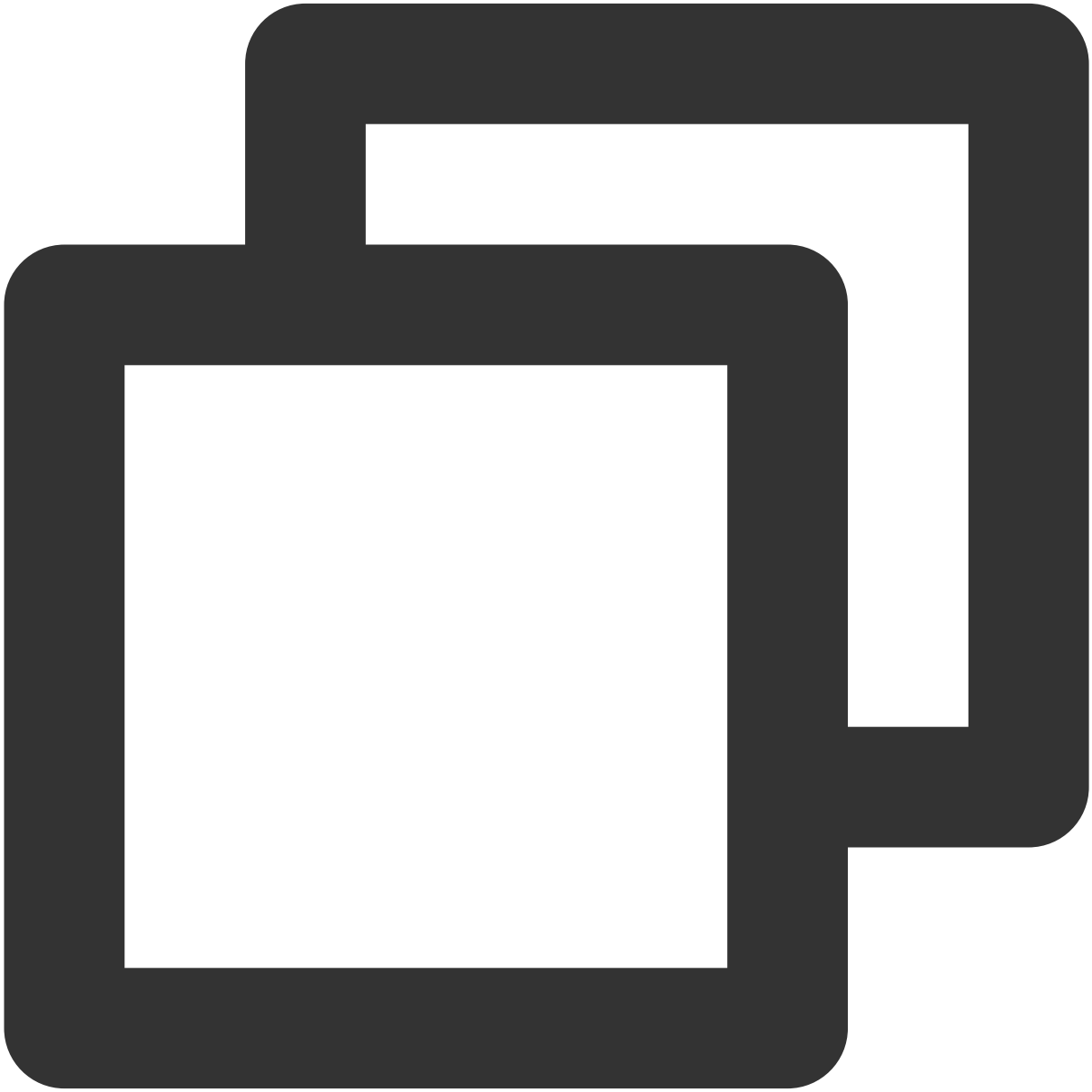
参数如下表所示：

参数	类型	含义
callback	UserListCallback	用户详细信息回调。

推拉流相关接口函数

startCameraPreview

开启本地视频的预览画面。



```
public abstract void startCameraPreview(boolean isFront, TXCloudVideoView view, TRT
```

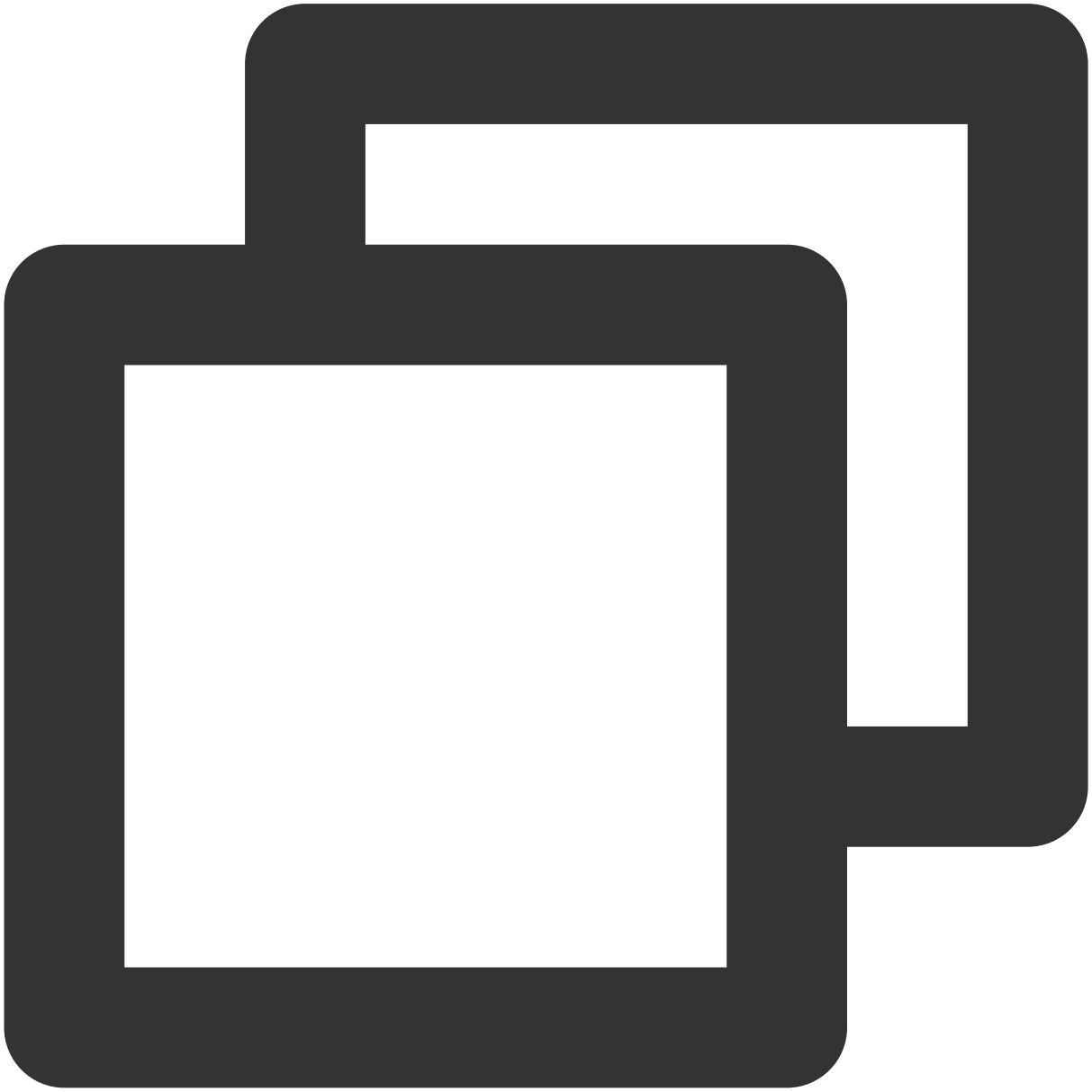
参数如下表所示：

参数	类型	含义

isFront	boolean	true：前置摄像头；false：后置摄像头。
view	TXCloudVideoView	承载视频画面的控件。
callback	ActionCallback	操作回调。

stopCameraPreview

停止本地视频采集及预览。



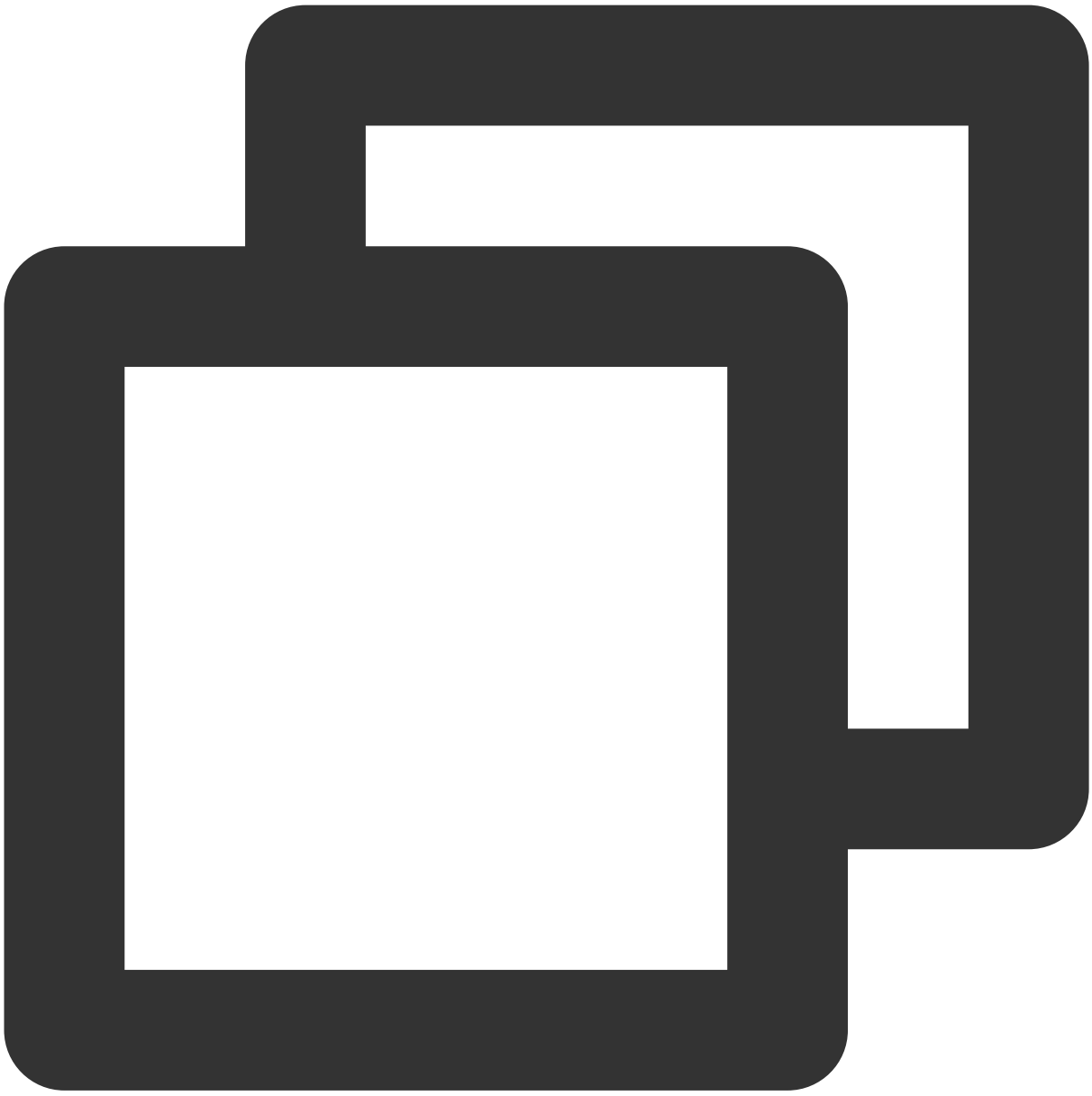
```
public abstract void stopCameraPreview();
```

startPublish

开始直播（推流），适用于以下场景：

主播开播的时候调用

观众开始连麦时调用



```
public abstract void startPublish(String streamId, TRTCLiveRoomCallback.ActionCallb
```

参数如下表所示：

参数	类型	含义

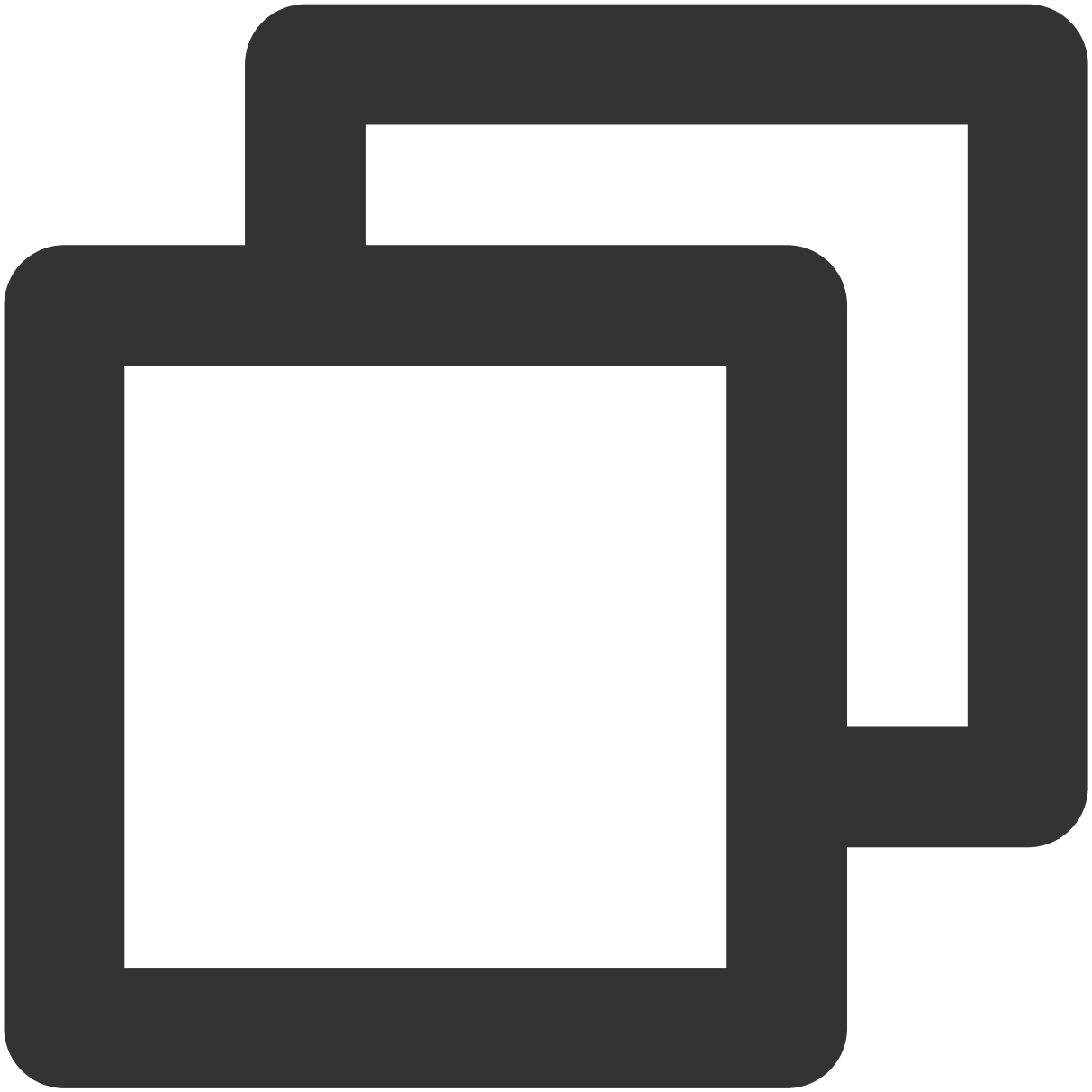
streamId	String	用于绑定直播 CDN 的 streamId，如果您希望观众通过直播 CDN 进行观看，需要指定当前主播的直播 streamId。
callback	ActionCallback	操作回调。

stopPublish

停止直播（推流），适用于以下场景：

主播结束直播时调用

观众结束连麦时调用



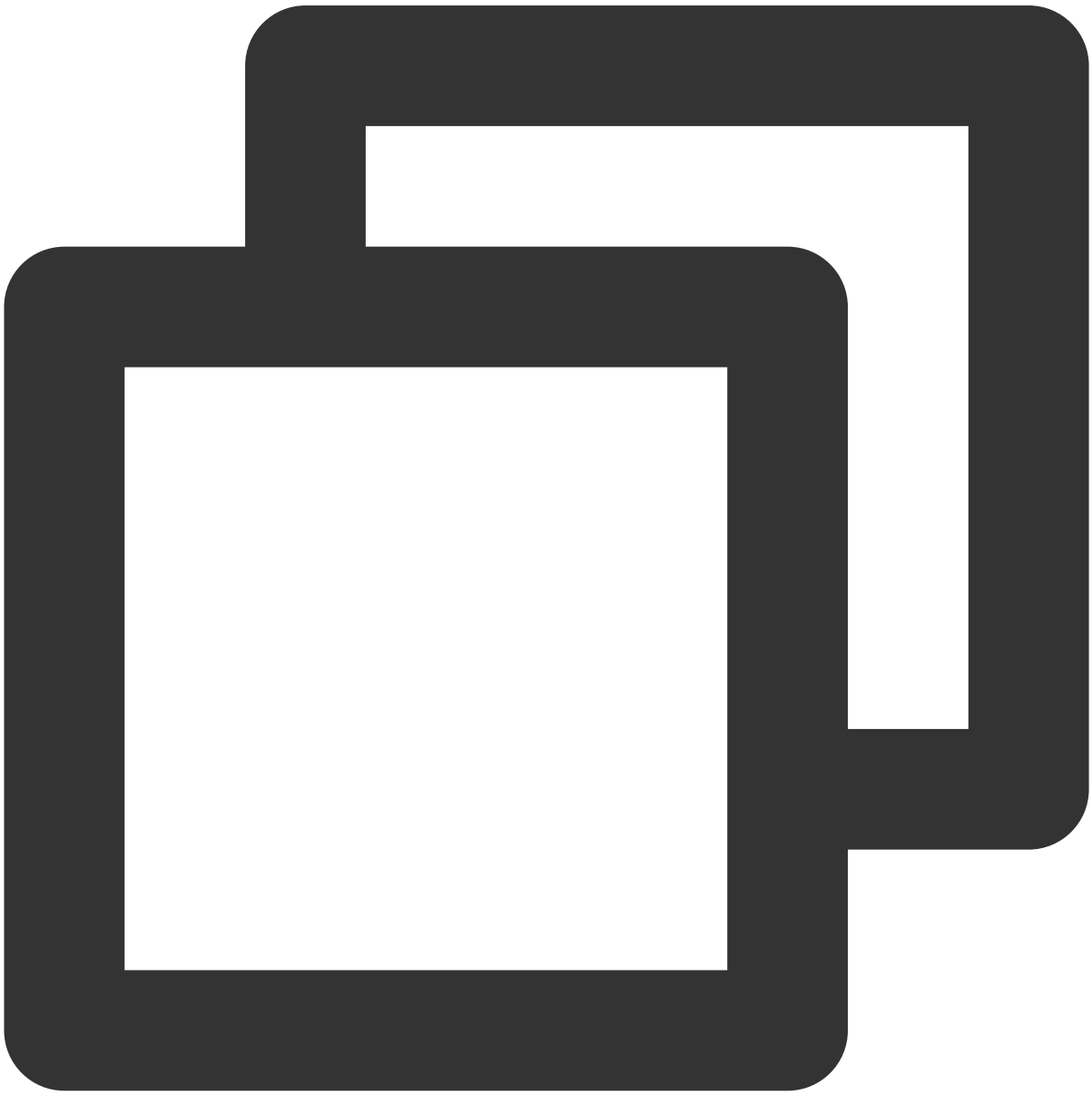
```
public abstract void stopPublish(TRTCLiveRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

startPlay

播放远端视频画面，可以在普通观看和连麦场景中调用。



```
public abstract void startPlay(String userId, TXCloudVideoView view, TRTCLiveRoomCa
```

参数如下表所示：

参数	类型	含义
userId	String	需要观看的用户id。
view	TXCloudVideoView	承载视频画面的 view 控件。
callback	ActionCallback	操作回调。

普通观看场景

若直播间列表已包含主播端的 userId 信息，观众端可以直接在 `enterRoom()` 成功后调用 `startPlay(userId)` 播放主播的画面。

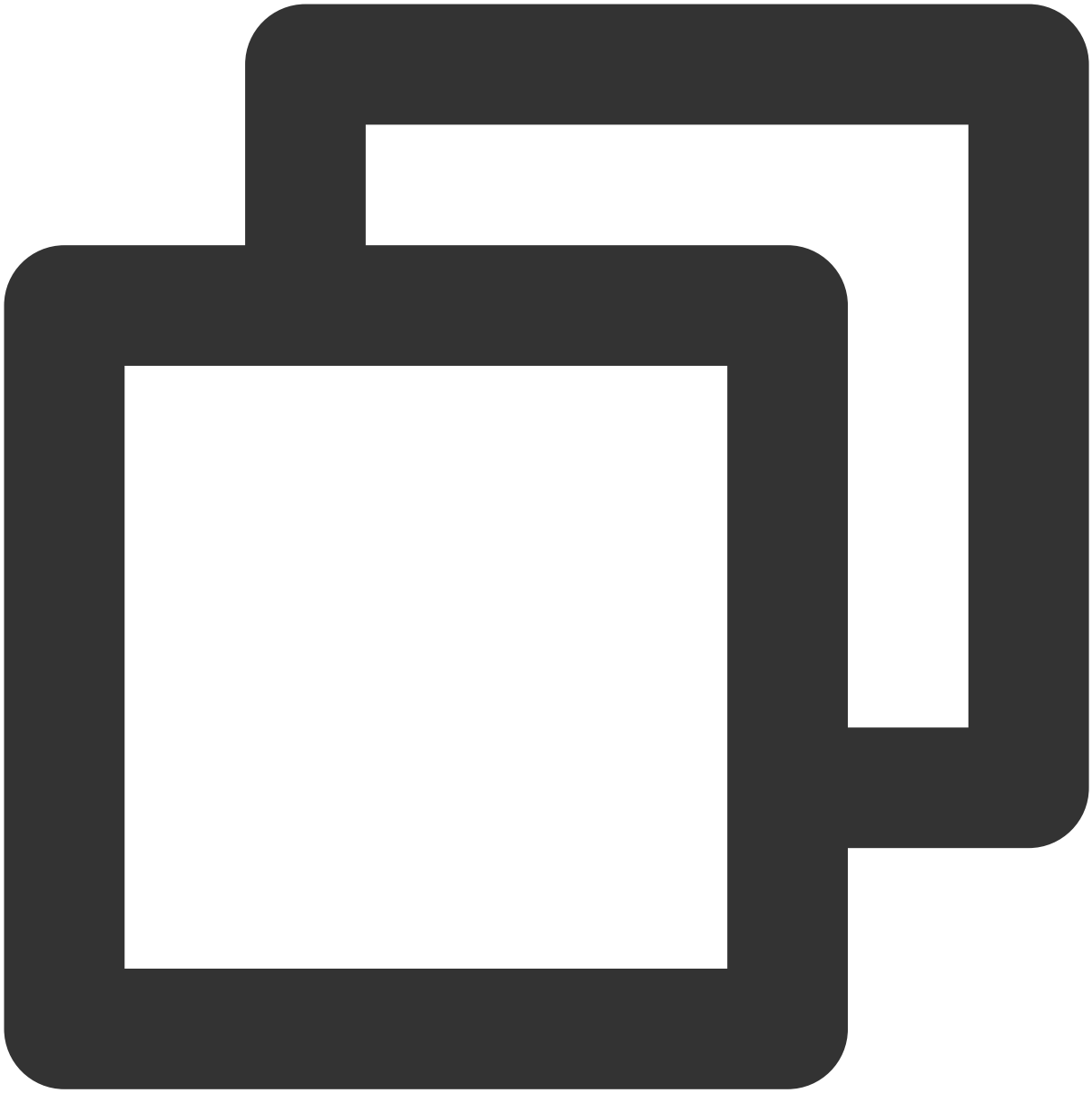
若在进房前暂未获取主播的 userId，观众端在进房后会收到 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userId)` 的事件回调，该回调中携带主播的 userId 信息，再调用 `startPlay(userId)` 即可播放主播的画面。

直播连麦场景

发起连麦后，主播会收到来自 `TRTCLiveRoomDelegate` 中的 `onAnchorEnter(userId)` 回调，此时使用回调中的 userId 调用 startPlay(userId) 即可播放连麦画面。

stopPlay

停止渲染远端视频画面。需在 `onAnchorExit()` 回调时，调用该接口。



```
public abstract void stopPlay(String userId, TRTCLiveRoomCallback.ActionCallback ca
```

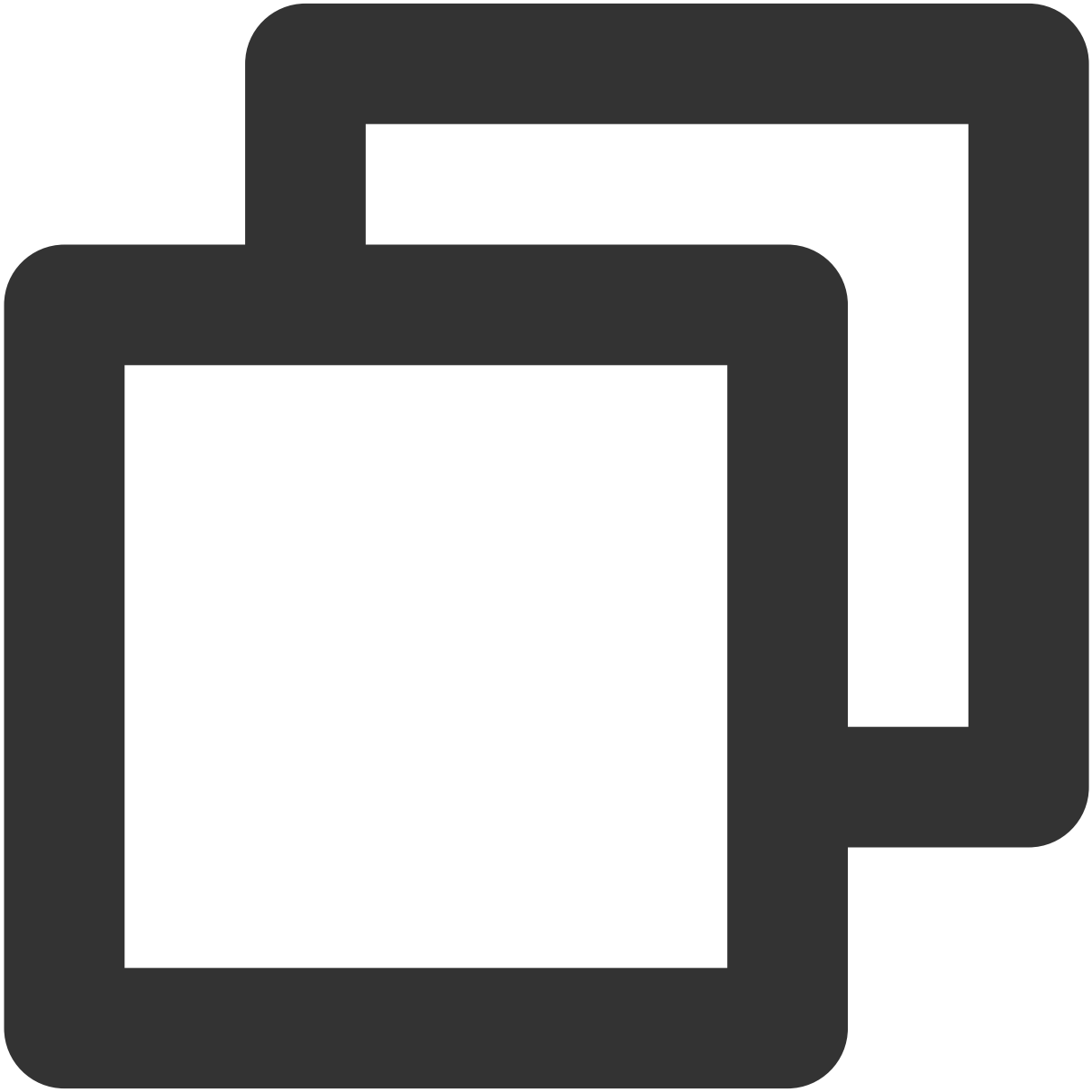
参数如下表所示：

参数	类型	含义
userId	String	对方的用户信息。
callback	ActionCallback	操作回调。

主播和观众连麦

requestJoinAnchor

观众请求连麦。



```
public abstract void requestJoinAnchor(String reason, int timeout, TRTCLiveRoomCall
```

参数如下表所示：

参数	类型	含义

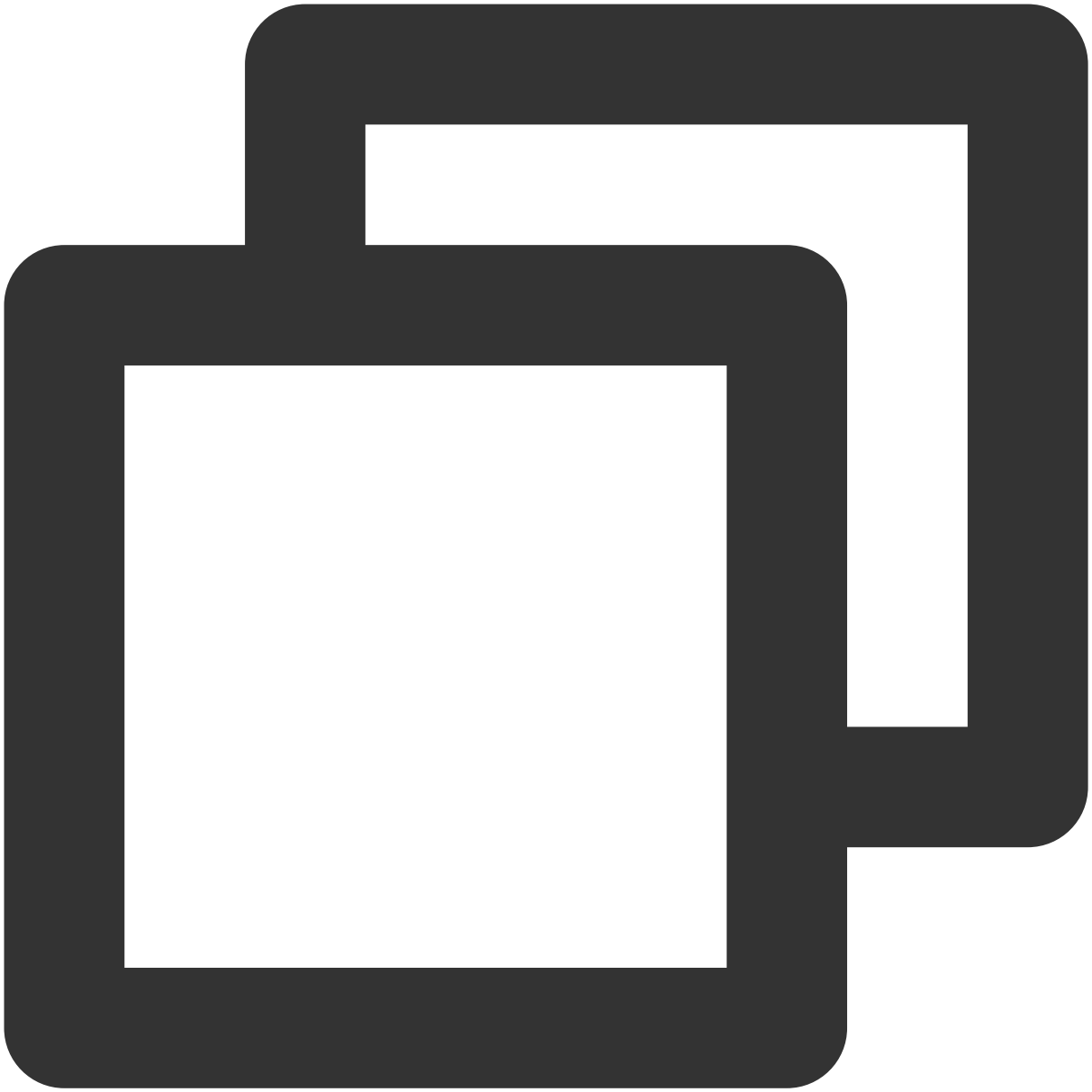
reason	String	连麦原因。
timeout	int	超时时间。
callback	ActionCallback	主播响应回调。

主播和观众的连麦流程如下：

1. 【观众】调用 `requestJoinAnchor()` 向主播发起连麦请求。
2. 【主播】会收到 `TRTCLiveRoomDelegate` 的 `onRequestJoinAnchor()` 回调通知。
3. 【主播】调用 `responseJoinAnchor()` 决定是否接受来自观众的连麦请求。
4. 【观众】会收到 responseCallback 回调通知，该通知会携带主播的处理结果。
5. 【观众】如果请求被同意，则调用 `startCameraPreview()` 开启本地摄像头。
6. 【观众】然后调用 `startPublish()` 正式进入推流状态。
7. 【主播】一旦观众进入连麦状态，主播会收到 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知。
8. 【主播】主播调用 `startPlay()` 即可看到连麦观众的视频画面。
9. 【观众】如果直播间里已有其他观众正在跟主播连麦，新加入的连麦观众会收到 `onAnchorEnter()` 通知，调用 `startPlay()` 播放其他连麦者的视频画面。

responseJoinAnchor

主播处理连麦请求。主播在收到 `TRTCLiveRoomDelegate` 的 `onRequestJoinAnchor()` 回调后需要调用此接口来处理观众的连麦请求。



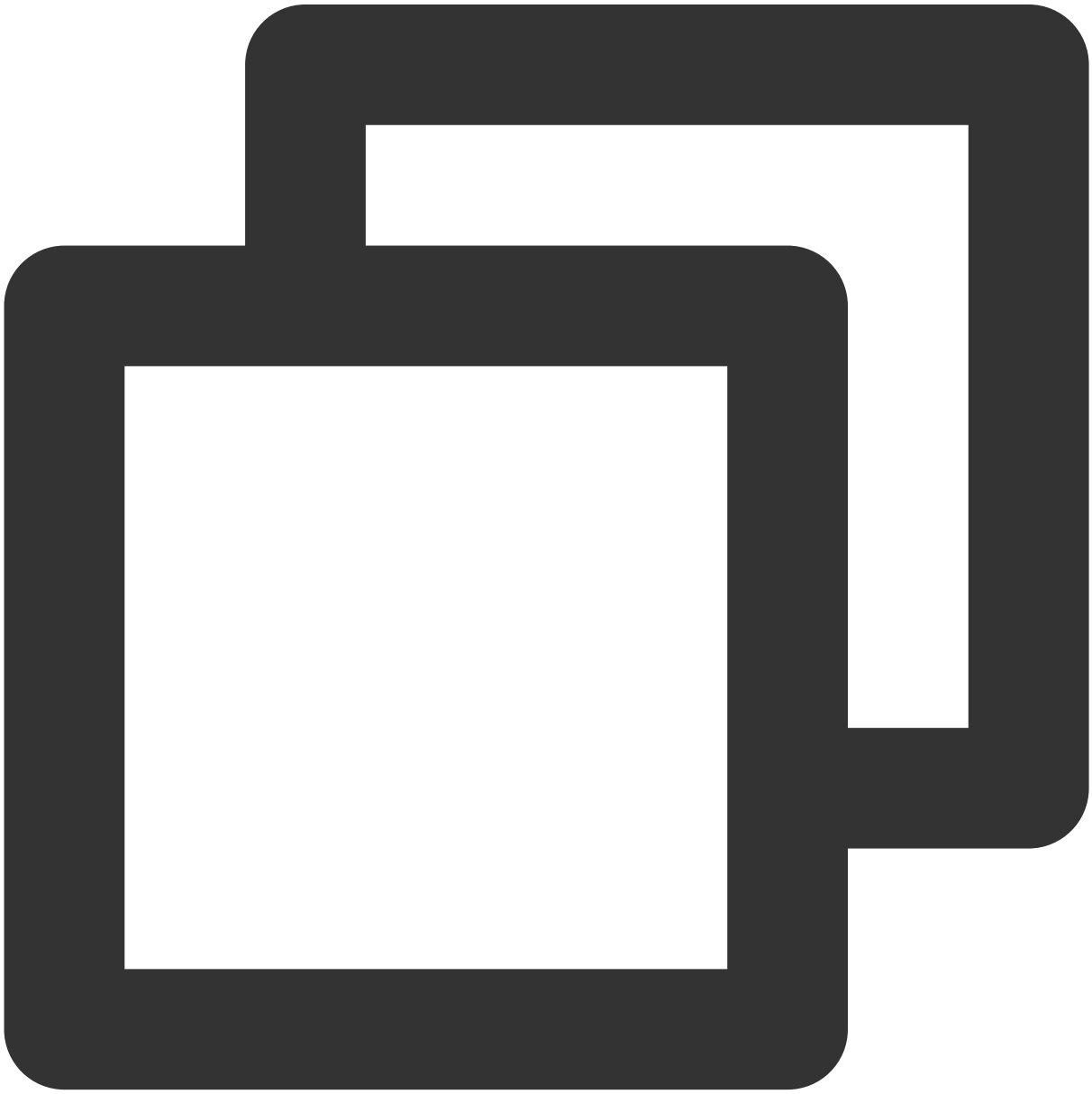
```
public abstract void responseJoinAnchor(String userId, boolean agree, String reason
```

参数如下表所示：

参数	类型	含义
userId	String	观众 ID。
agree	boolean	true：同意；false：拒绝。
reason	String	同意/拒绝连麦的原因描述。

kickoutJoinAnchor

主播踢除连麦观众。主播调用此接口踢除连麦观众后，被踢连麦观众会收到 `TRTCLiveRoomDelegate` 的 `onKickoutJoinAnchor()` 回调通知。



```
public abstract void kickoutJoinAnchor(String userId, TRTCLiveRoomCallback.ActionCa
```

参数如下表所示：

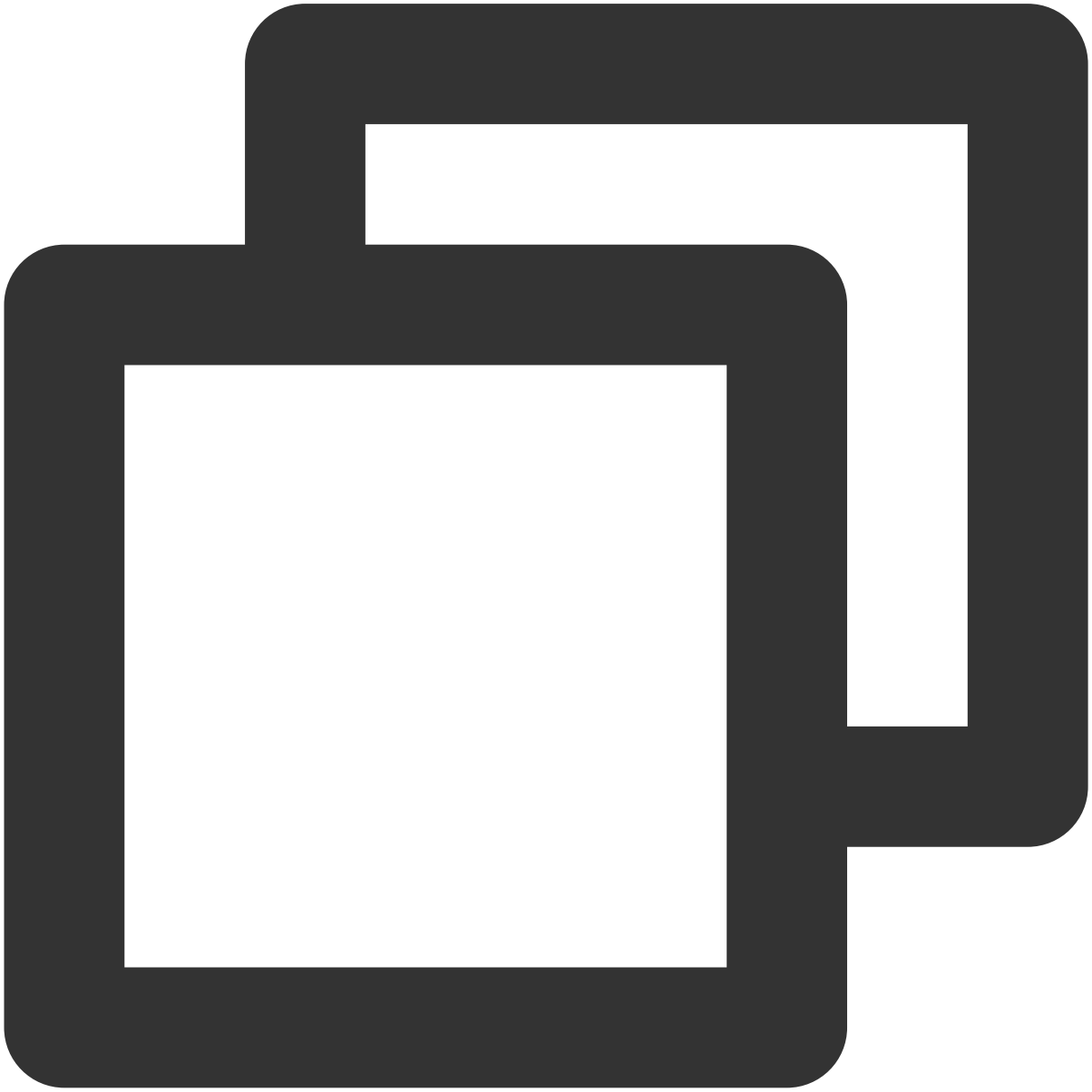
参数	类型	含义
userId	String	连麦观众 ID。

callback	ActionCallback	操作回调。

主播跨房间 PK

requestRoomPK

主播请求跨房 PK。



```
public abstract void requestRoomPK(int roomId, String userId, TRTCLiveRoomCallback.
```

参数如下表所示：

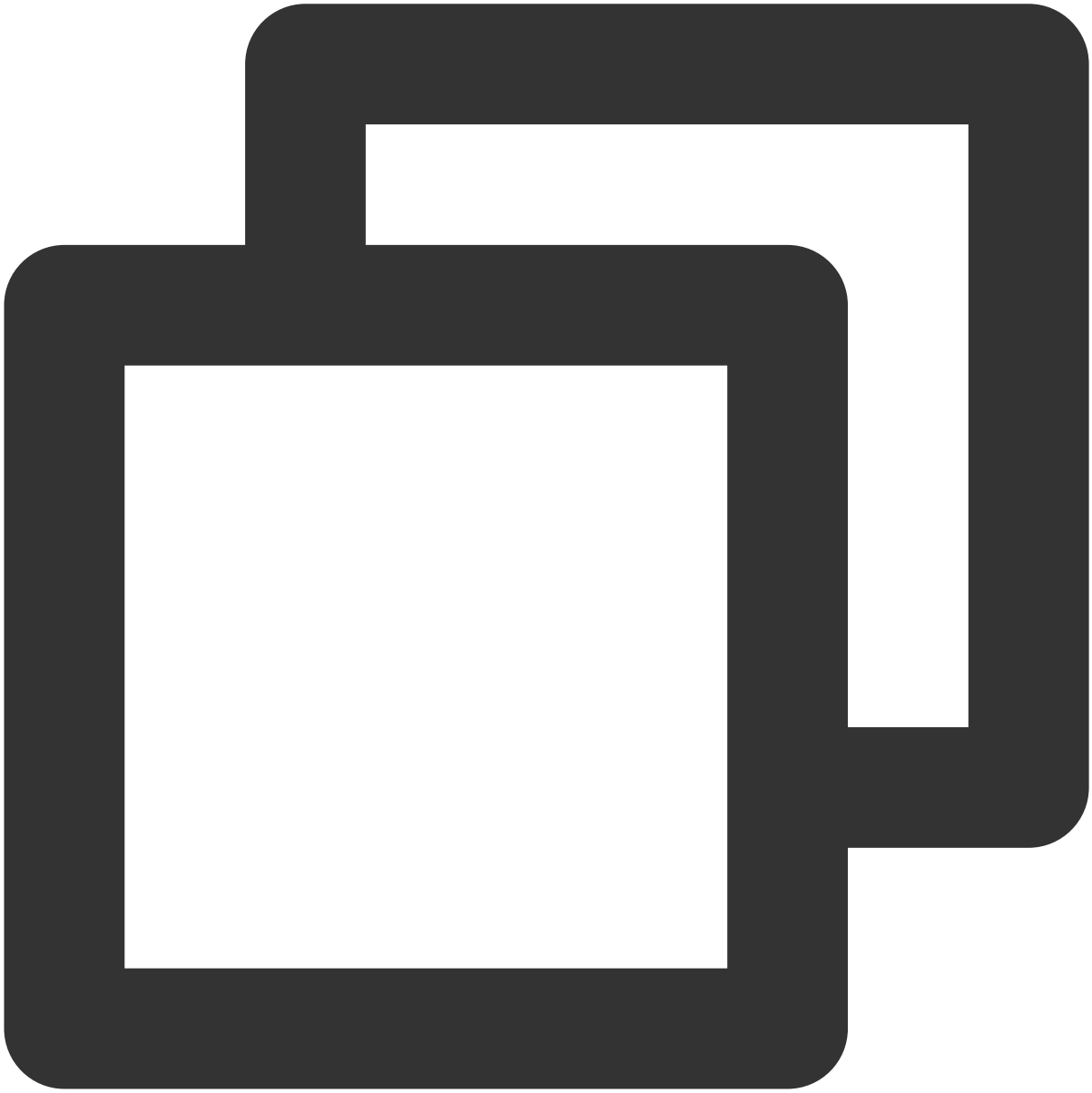
参数	类型	含义
roomId	int	被邀约房间 ID。
userId	String	被邀约主播 ID。
callback	ActionCallback	请求跨房 PK 的结果回调。

主播和主播之间可以跨房间 PK，两个正在直播中的主播 A 和 B 之间的跨房 PK 流程如下：

1. 【主播 A】调用 `requestRoomPK()` 向主播 B 发起连麦请求。
2. 【主播 B】会收到 `TRTCLiveRoomDelegate` 的 `onRequestRoomPK()` 回调通知。
3. 【主播 B】调用 `responseRoomPK()` 决定是否接受主播 A 的 PK 请求。
4. 【主播 B】如果接受主播 A 的要求，等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 来显示主播 A 的视频画面。
5. 【主播 A】会收到 `responseCallback` 回调通知，该通知会携带来自主播 B 的处理结果。
6. 【主播 A】如果请求被同意，等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 显示主播 B 的视频画面。

responseRoomPK

主播响应跨房 PK 请求。主播响应后，对方主播会收到 `requestRoomPK` 传入的 `responseCallback` 回调。



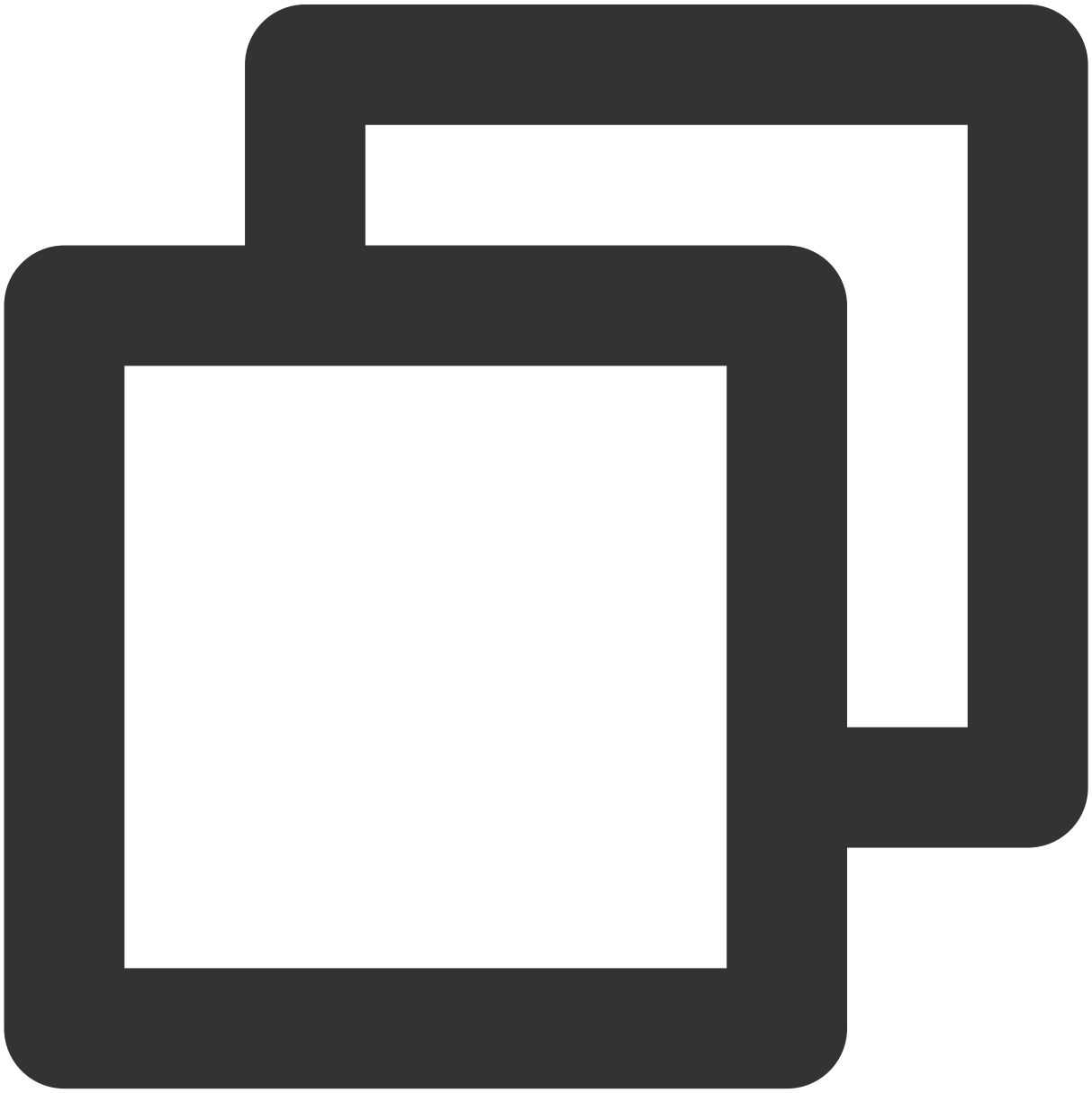
```
public abstract void responseRoomPK(String userId, boolean agree, String reason);
```

参数如下表所示：

参数	类型	含义
userId	String	发起 PK 请求的主播 ID。
agree	boolean	true：同意；false：拒绝。
reason	String	同意/拒绝 PK 的原因描述。

quitRoomPK

退出跨房 PK。PK 中的任何一个主播退出跨房 PK 状态后，另一个主播会收到 `TRTCLiveRoomDelegate` 的 `onQuitRoomPk()` 回调通知。



```
public abstract void quitRoomPK(TRTCLiveRoomCallback.ActionCallback callback);
```

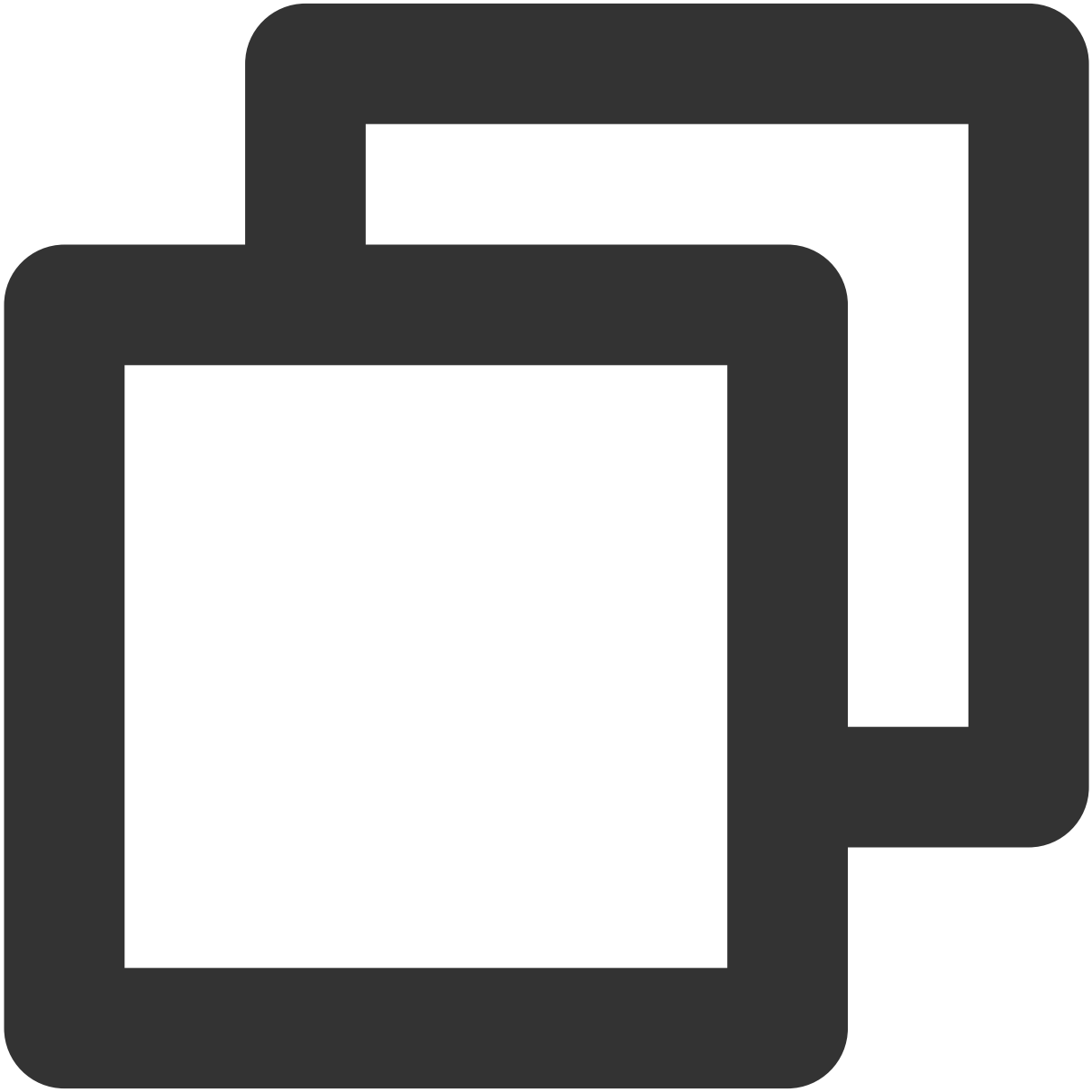
参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

音视频控制相关接口函数

switchCamera

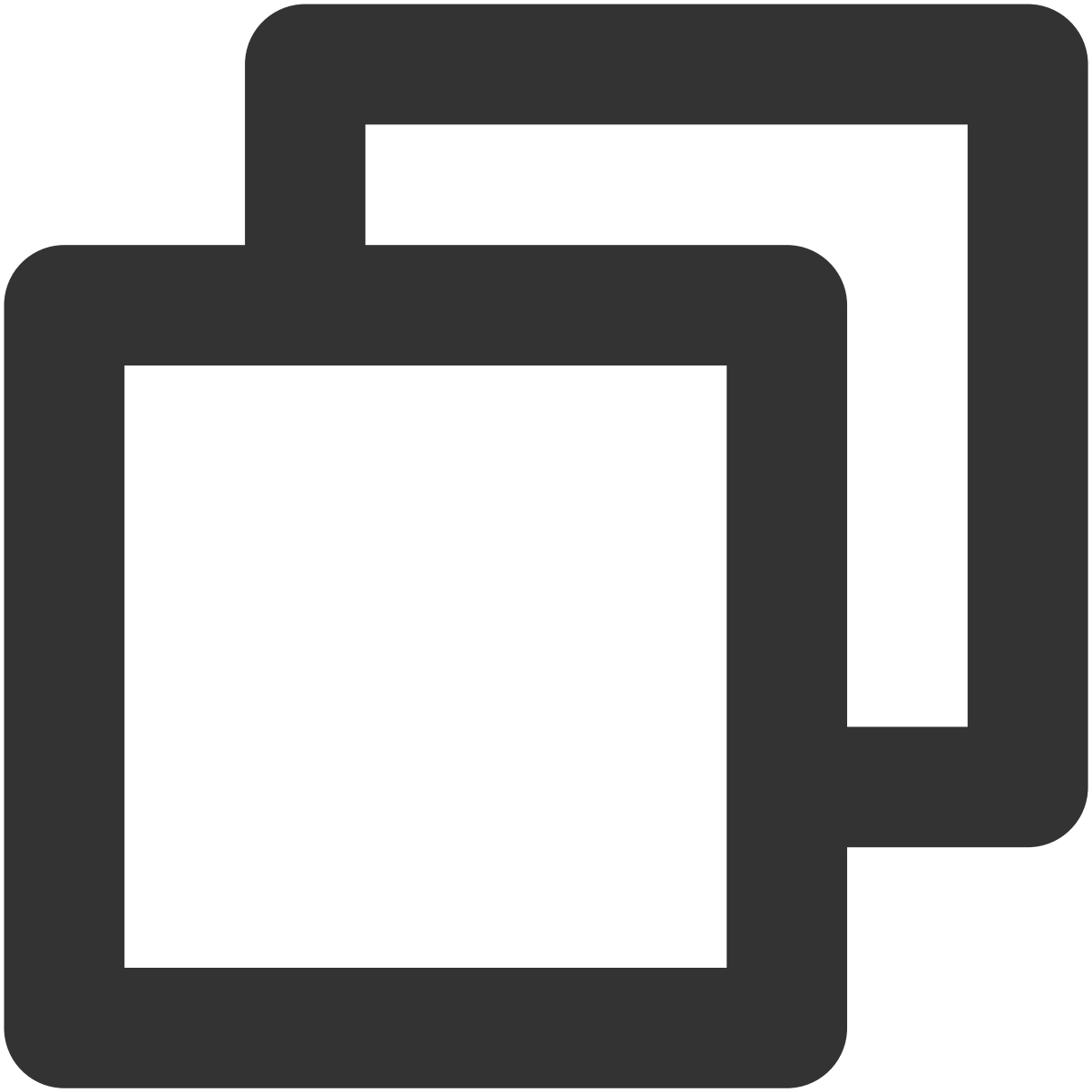
切换前后摄像头。



```
public abstract void switchCamera();
```

setMirror

设置是否镜像展示。



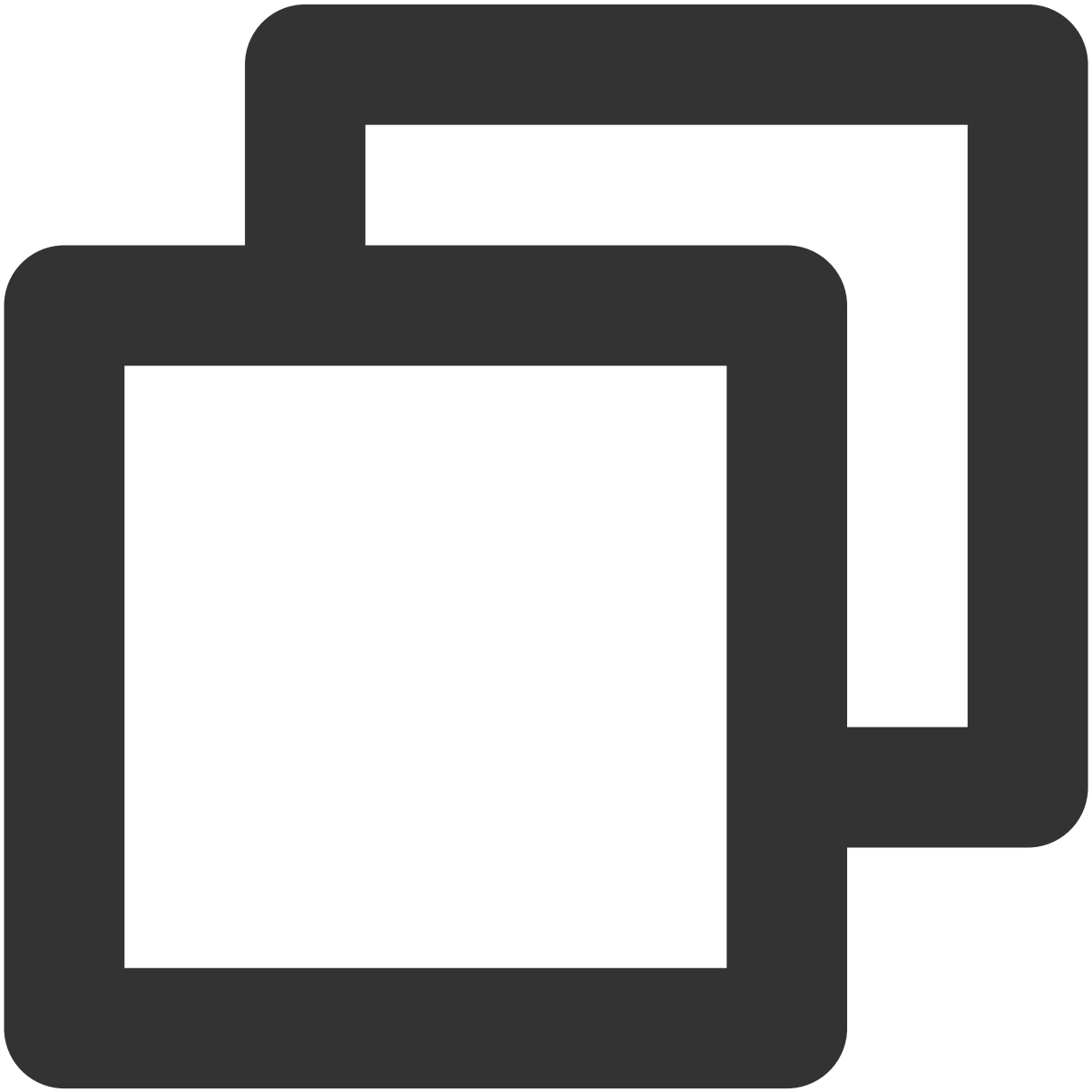
```
public abstract void setMirror(boolean isMirror);
```

参数如下表所示：

参数	类型	含义
isMirror	boolean	开启/关闭镜像。

muteLocalAudio

静音本地音频。



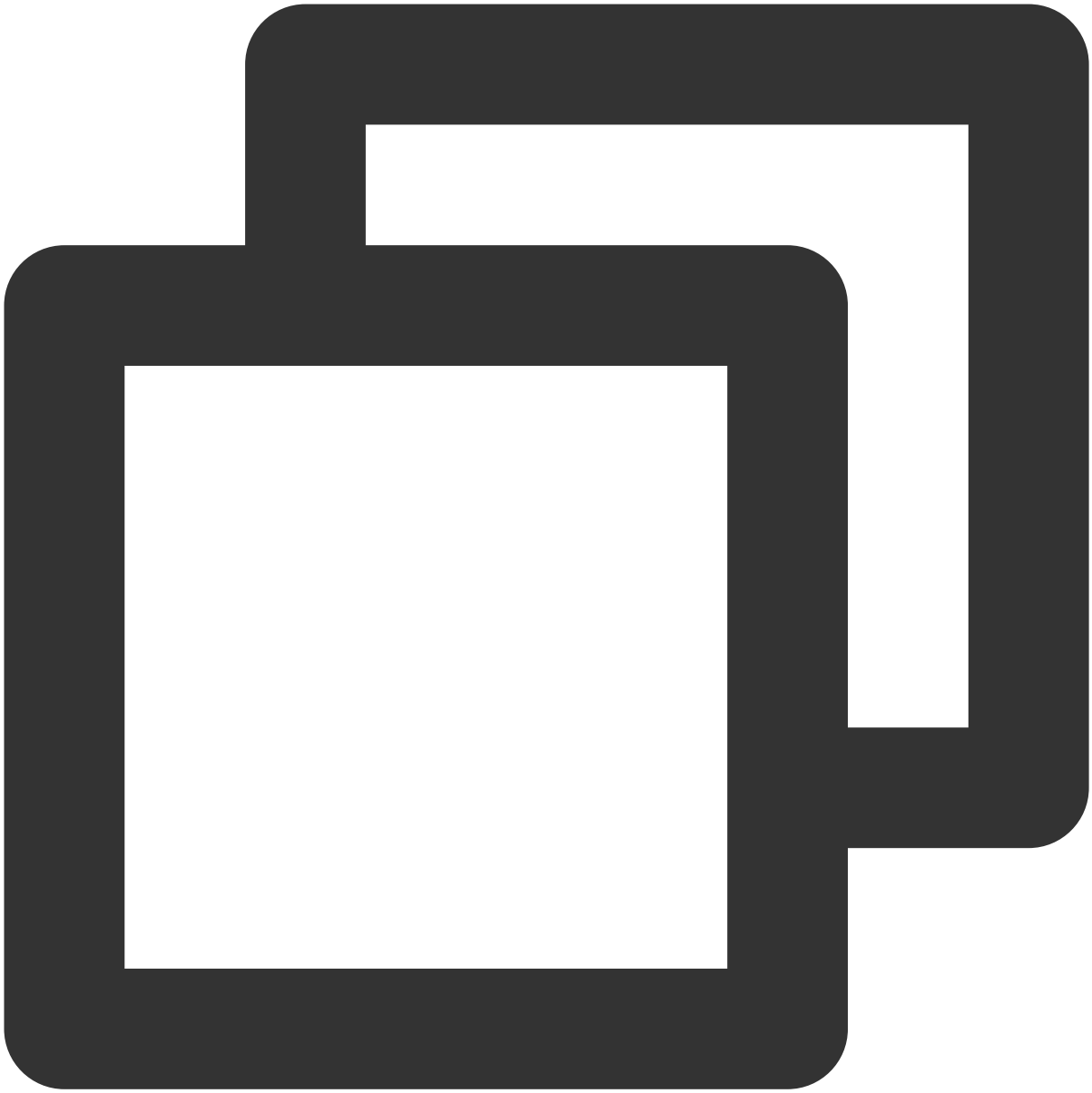
```
public abstract void muteLocalAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	true：开启静音；false：关闭静音。

muteRemoteAudio

静音远端音频。



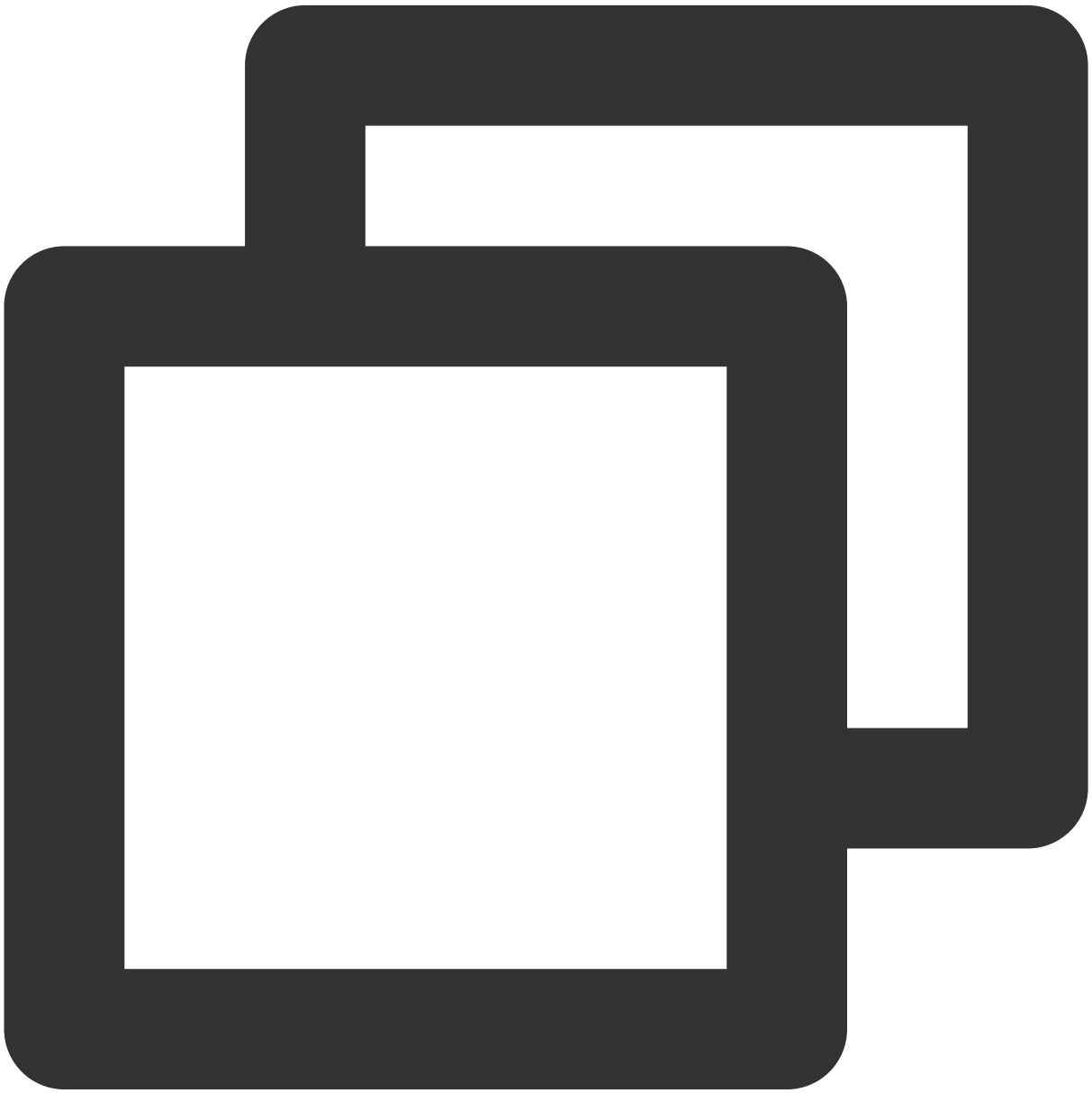
```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	远端的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音所有远端音频。



```
public abstract void muteAllRemoteAudio(boolean mute);
```

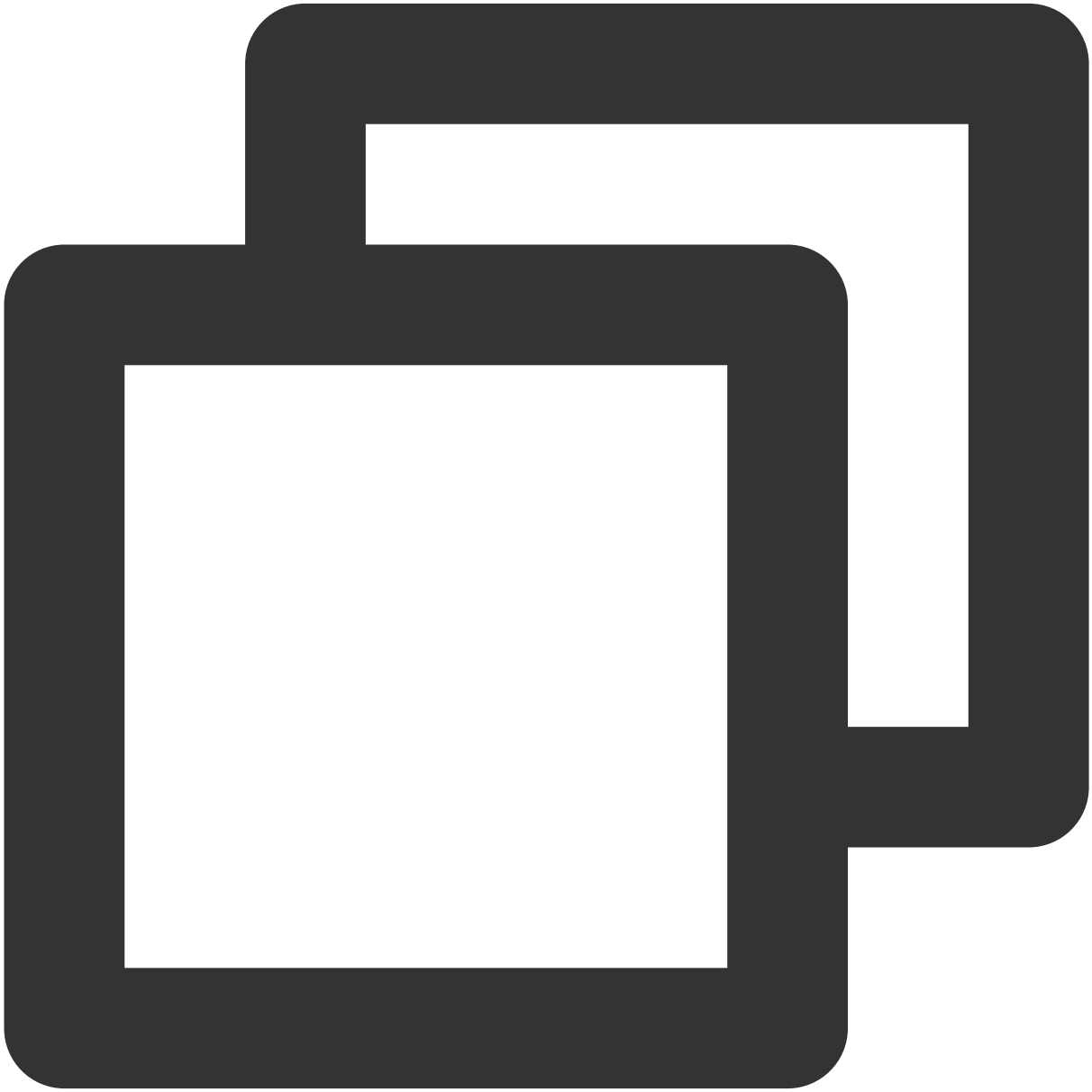
参数如下表所示：

参数	类型	含义
mute	boolean	true：开启静音；false：关闭静音。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

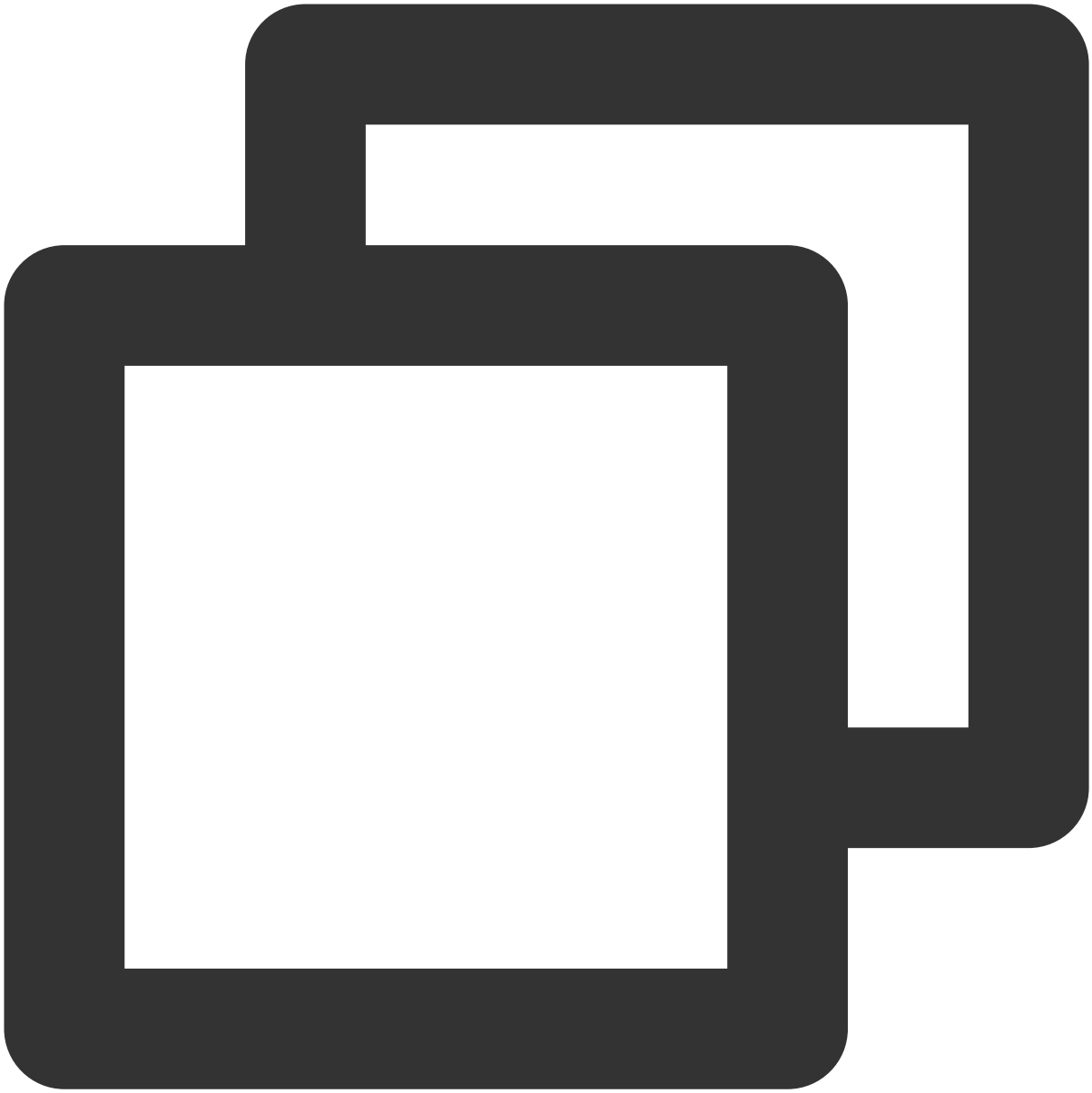


```
public abstract TXAudioEffectManager getAudioEffectManager();
```

美颜滤镜相关接口函数

getBeautyManager

获取美颜管理对象 TXBeautyManager。



```
public abstract TXBeautyManager getBeautyManager();
```

通过美颜管理，您可以使用以下功能：
设置“美颜风格”、“美白”、“红润”、“大眼”、“瘦脸”、“V脸”、“下巴”、“短脸”、“小鼻”、“亮眼”、“白牙”、“祛眼袋”、“祛皱纹”、“祛法令纹”等美容效果。
调整“发际线”、“眼间距”、“眼角”、“嘴形”、“鼻翼”、“鼻子位置”、“嘴唇厚度”、“脸型”。
设置人脸挂件（素材）等动态效果。

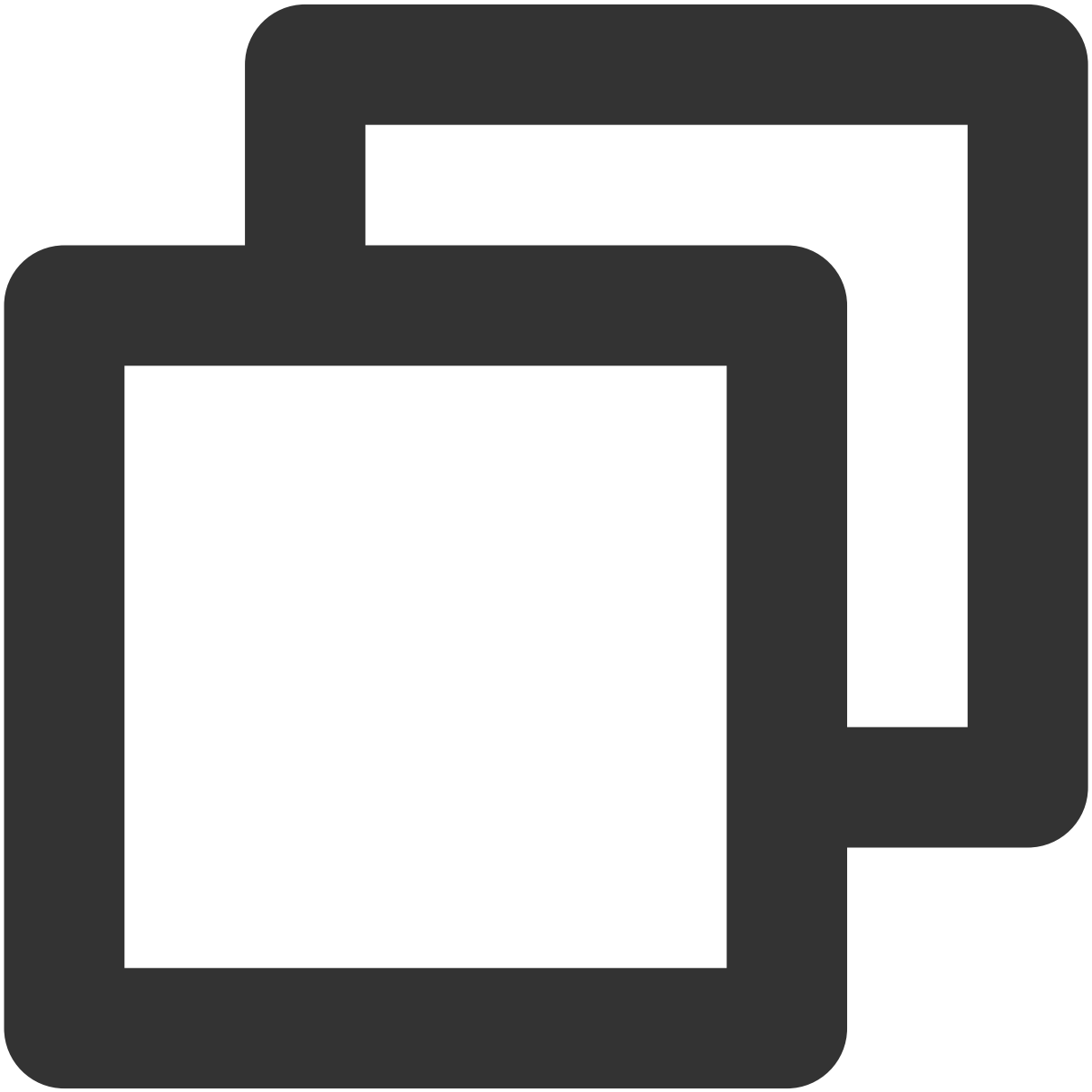
版权所有：腾讯云计算（北京）有限责任公司

添加美妆。
进行手势识别。

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。



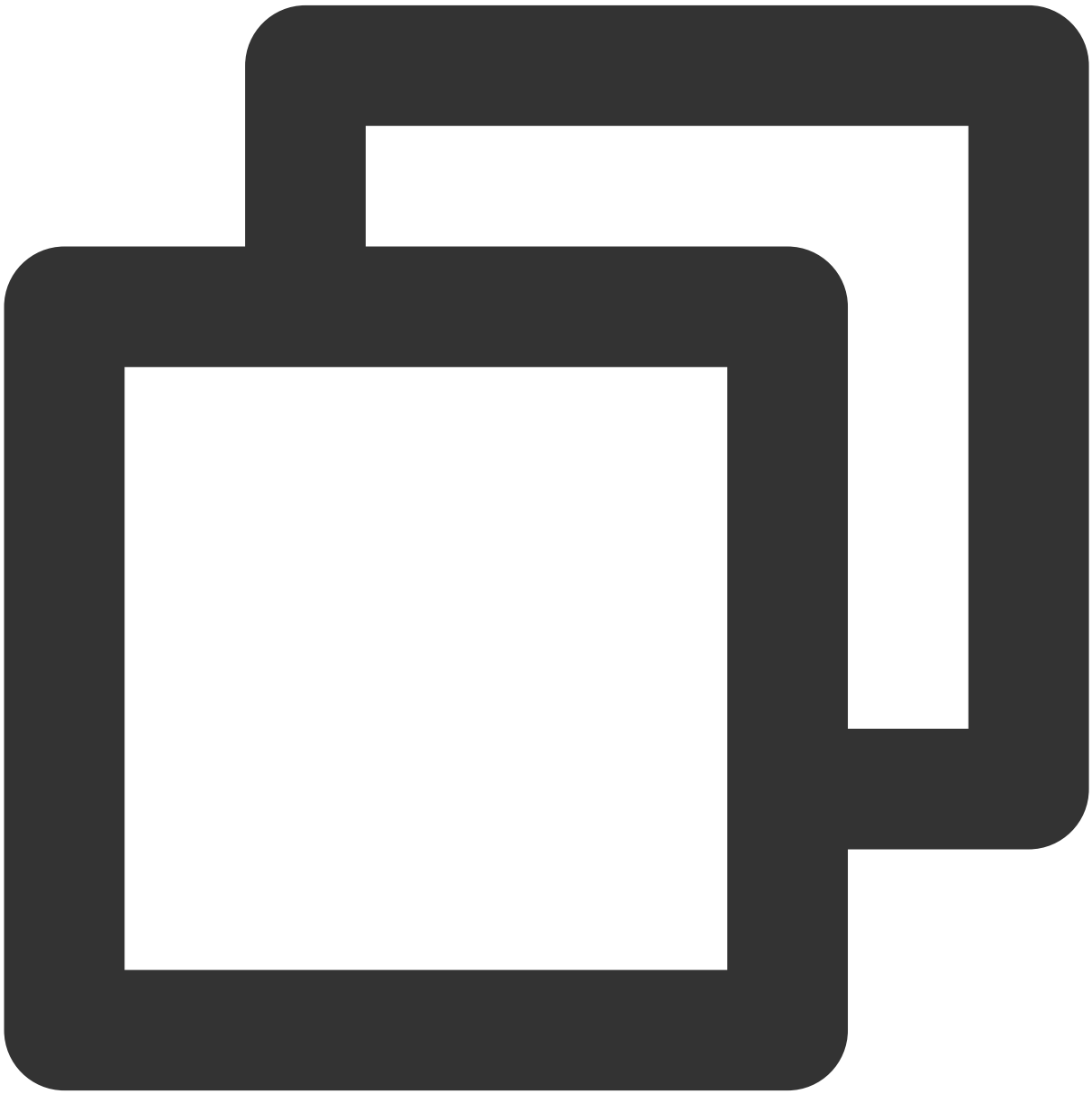
```
public abstract void sendRoomTextMsg(String message, TRTCLiveRoomCallback.ActionCal
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。



```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCLiveRoomCall
```

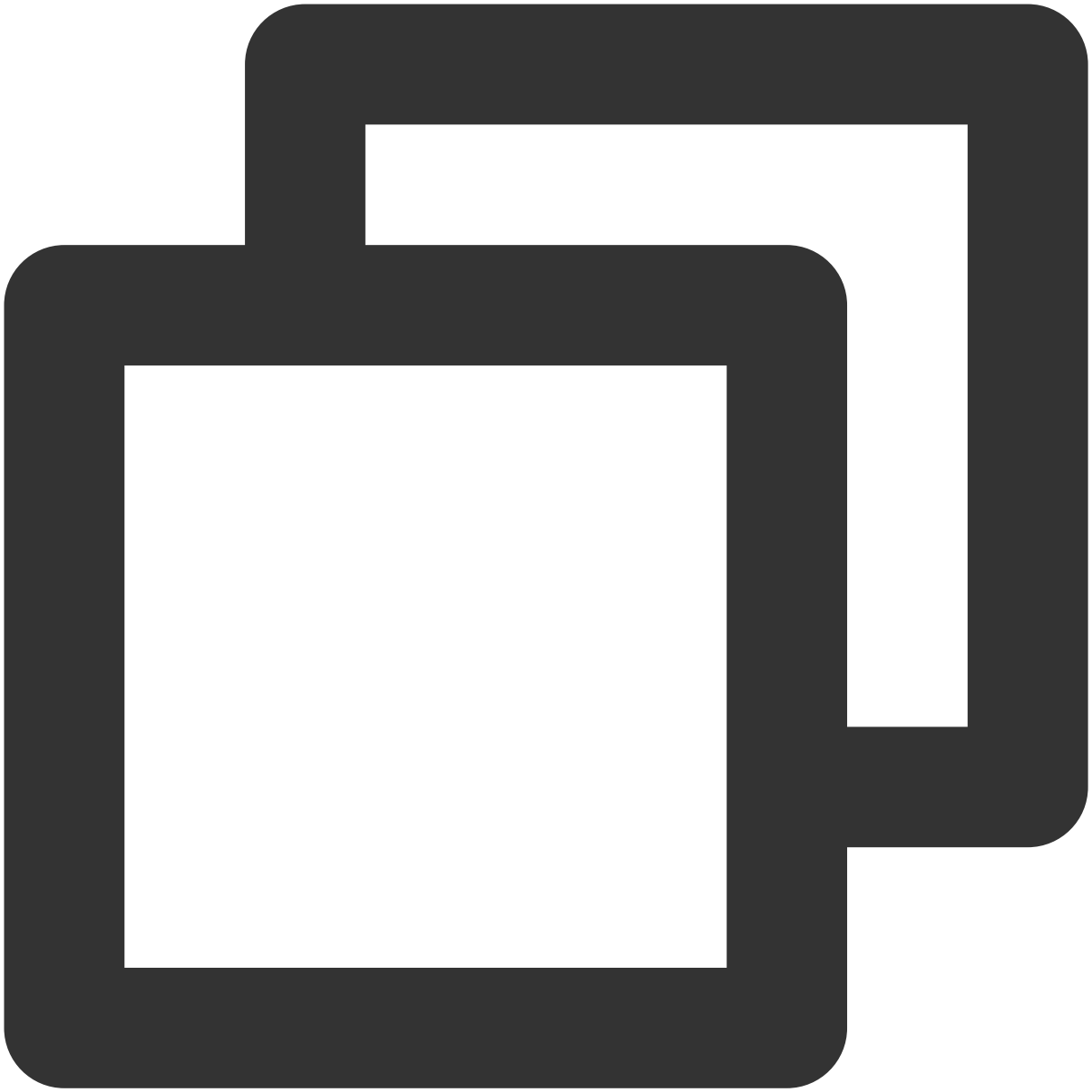
参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

调试相关接口函数

showVideoDebugLog

是否在界面中展示debug信息。



```
public abstract void showVideoDebugLog(boolean isShow);
```

参数如下表所示：

参数	类型	含义
isShow	boolean	开启/关闭 Debug 信息显示。

TRTCLiveRoomDelegate事件回调

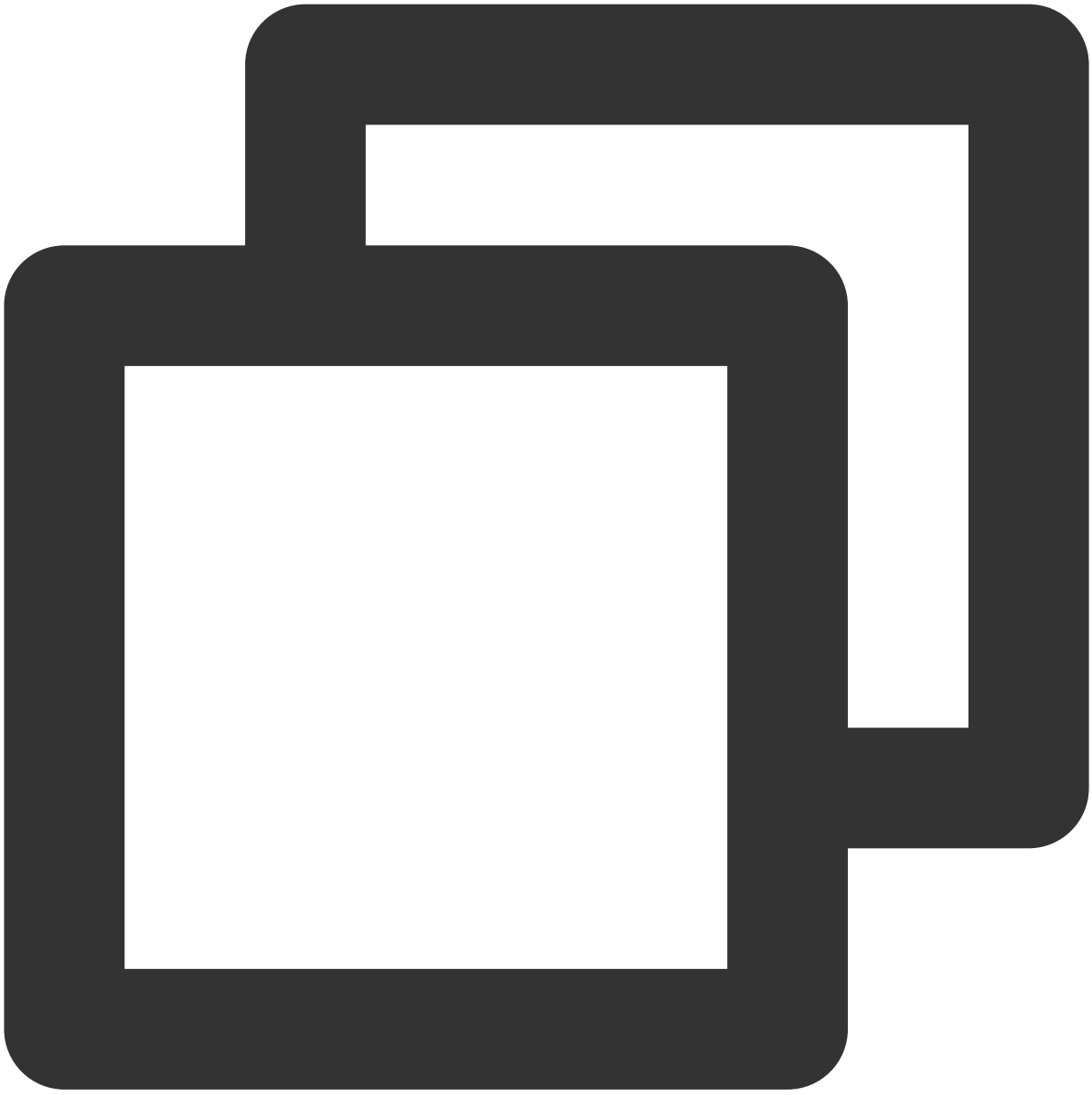
通用事件回调

onError

错误回调。

说明：

SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。



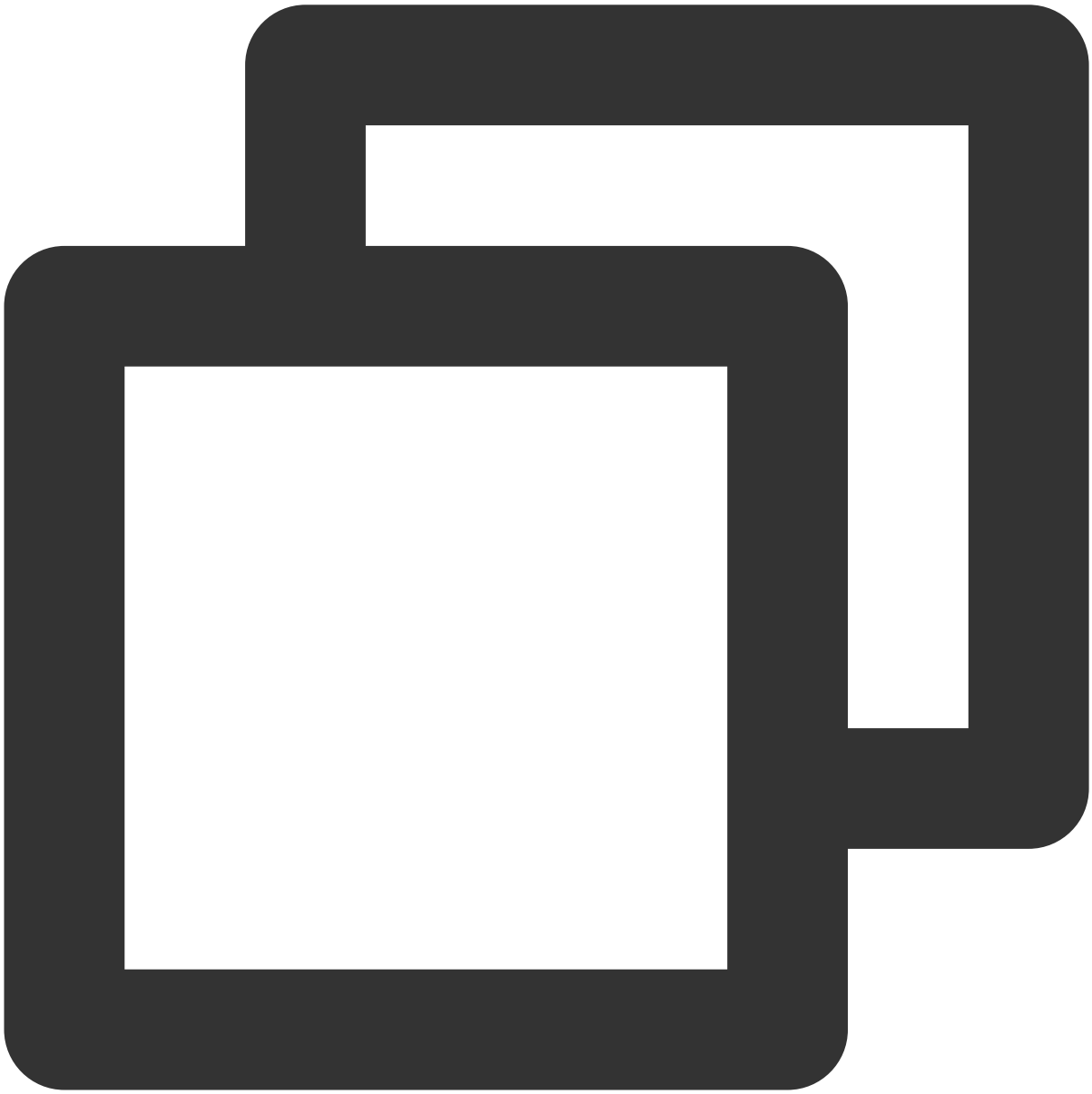
```
void onError(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。



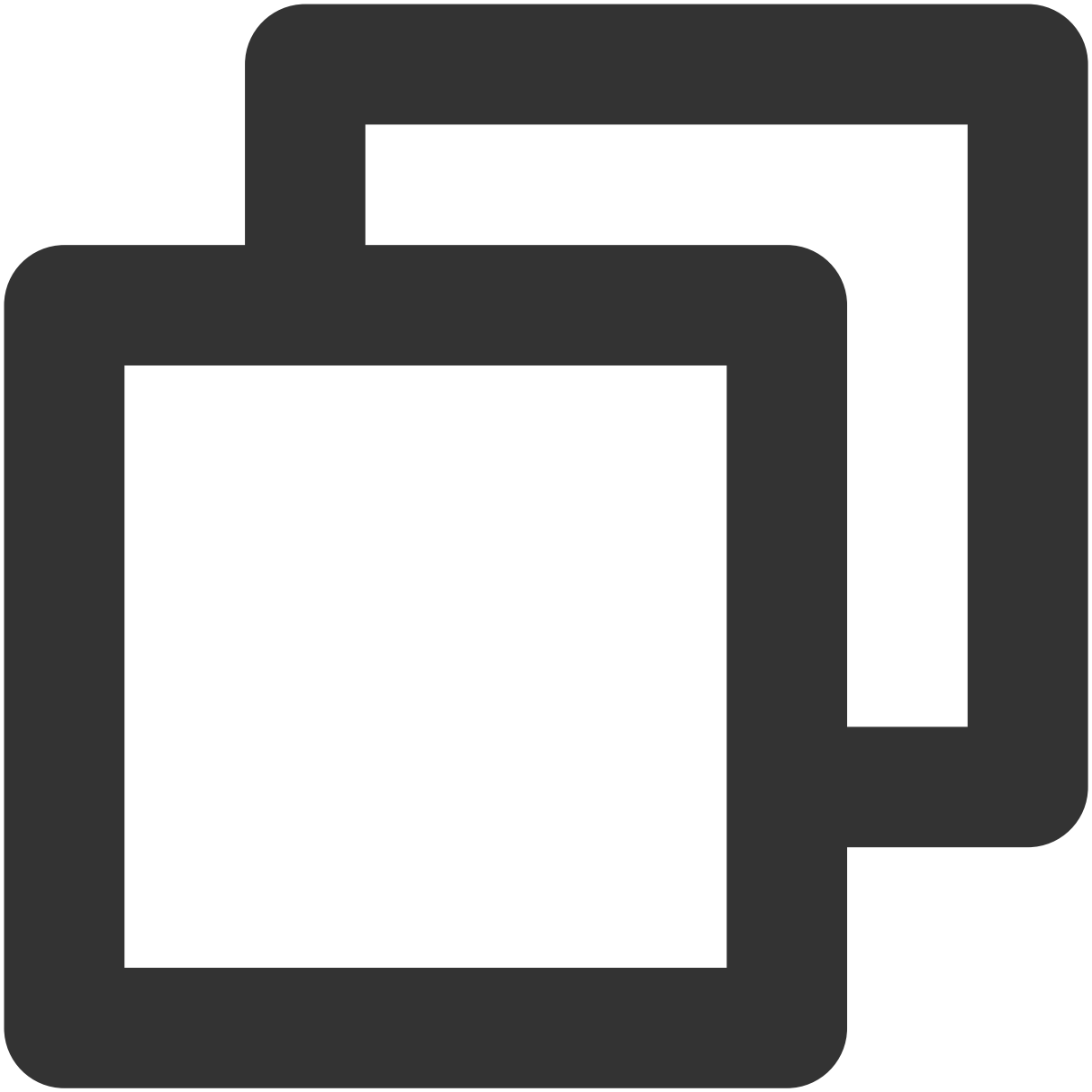
```
void onWarning(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	警告信息。

onDebugLog

Log 回调。



```
void onDebugLog(String message);
```

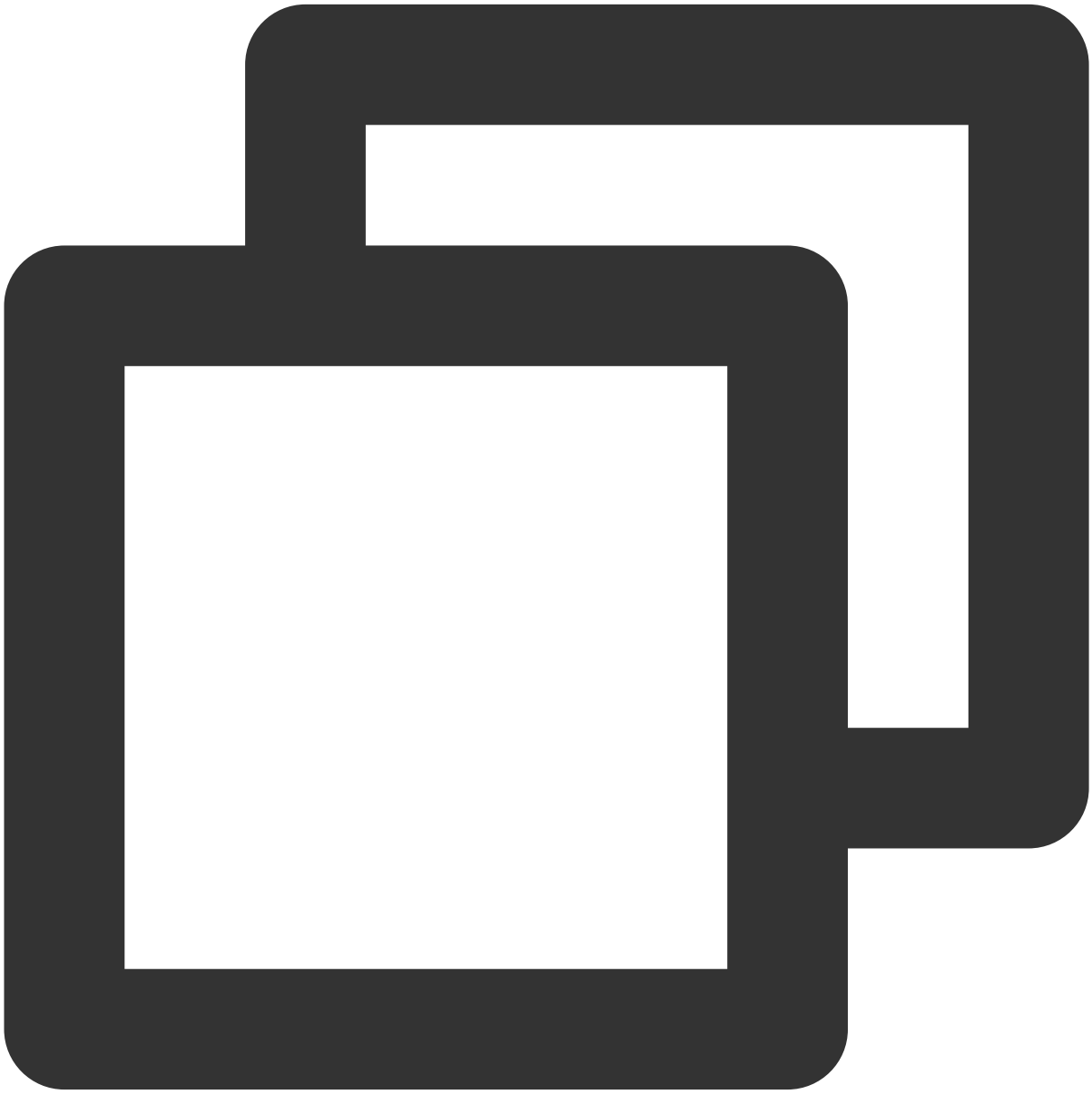
参数如下表所示：

参数	类型	含义
message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。主播退房时，房间内的所有用户都会收到此通知。



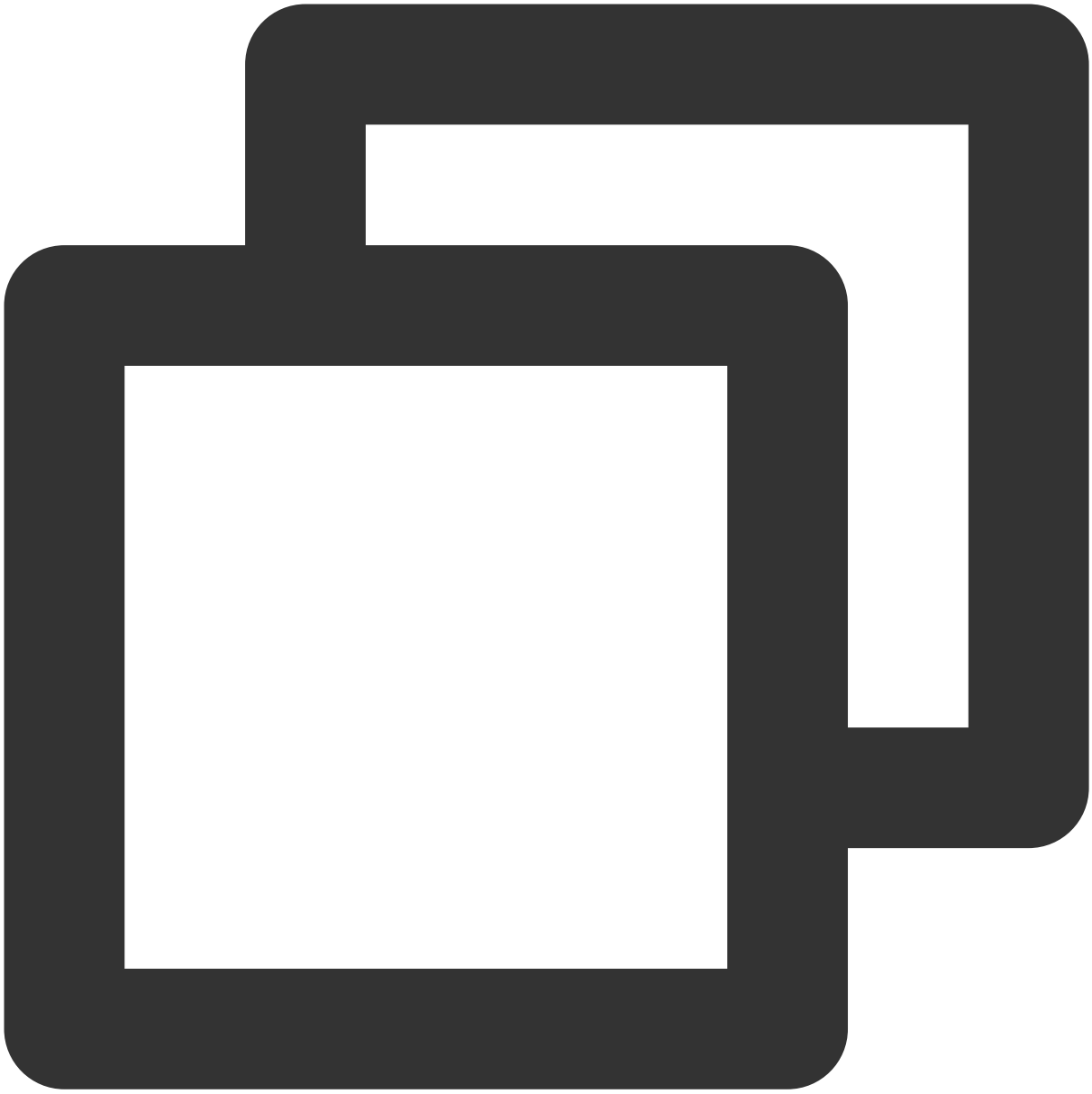
```
void onRoomDestroy(String roomId);
```

参数如下表所示：

参数	类型	含义
roomId	String	房间 ID。

onRoomInfoChange

直播房间信息变更回调。多用于直播连麦、PK下房间状态变化通知场景。



```
void onRoomInfoChange(TRTCLiveRoomDef.TRTCLiveRoomInfo roomInfo);
```

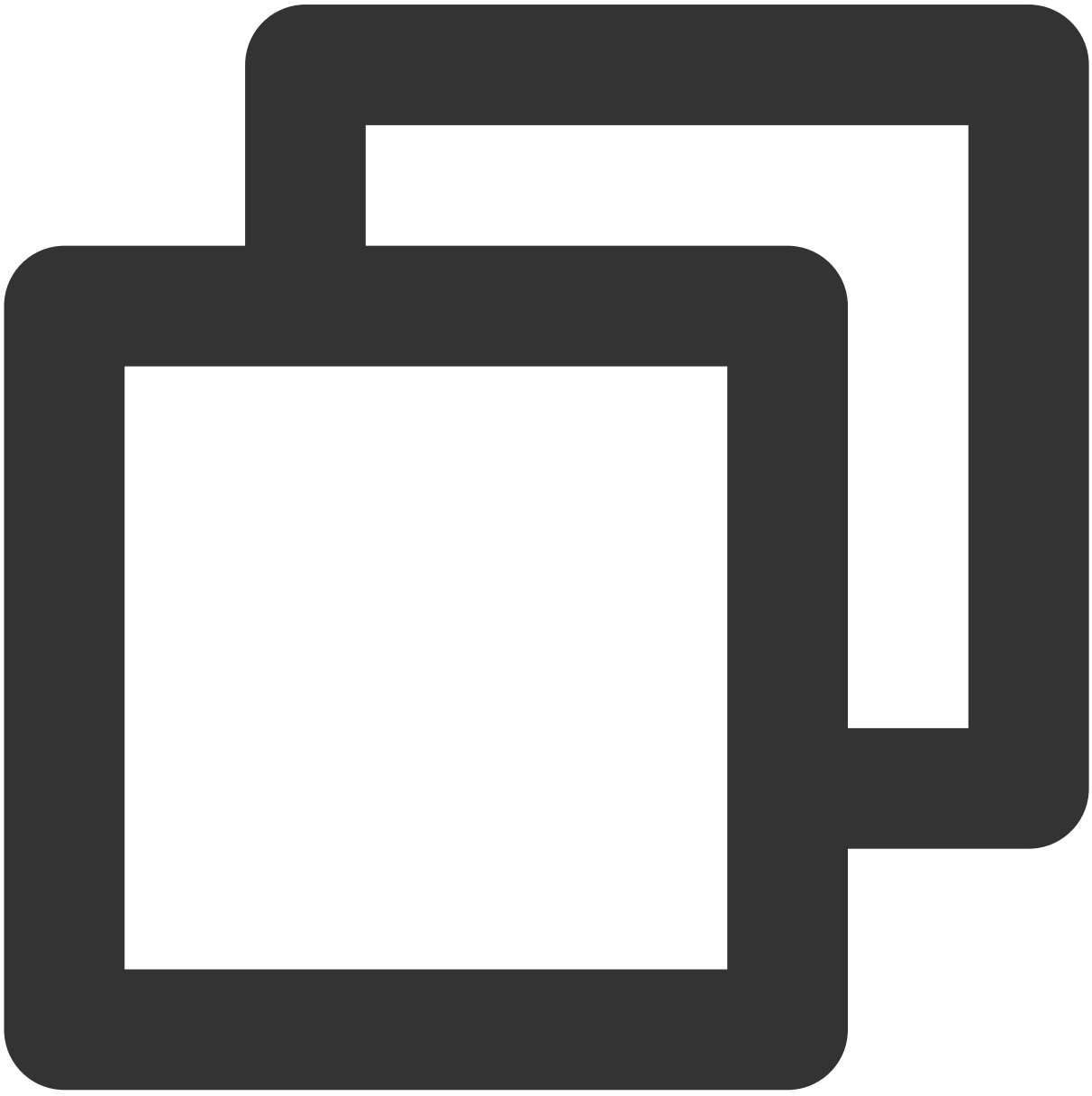
参数如下表所示：

参数	类型	含义
roomInfo	TRTCLiveRoomInfo	房间信息。

主播和观众进出事件回调

onAnchorEnter

收到新主播进房通知。连麦观众和跨房 PK 主播进房后观众会收到新主播的进房事件，您可以调用 `TRTCLiveRoom` 的 `startPlay()` 显示该主播的视频画面。



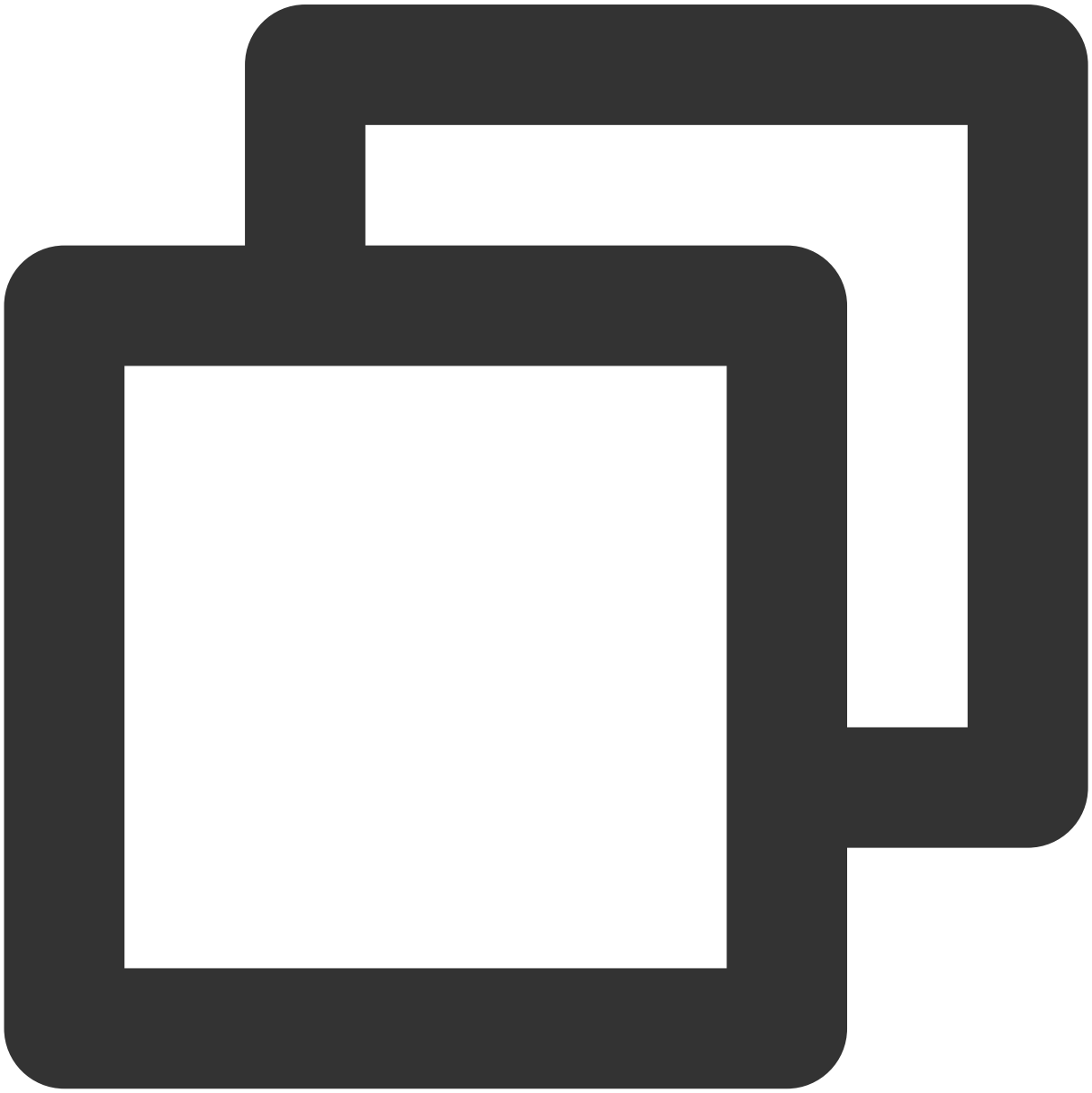
```
void onAnchorEnter(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	新进房主播 ID。

onAnchorExit

收到主播退房通知。房间内的主播（和连麦中的观众）会收到新主播的退房事件，您可以调用 `TRTCLiveRoom` 的 `stopPlay()` 关闭该主播的视频画面。



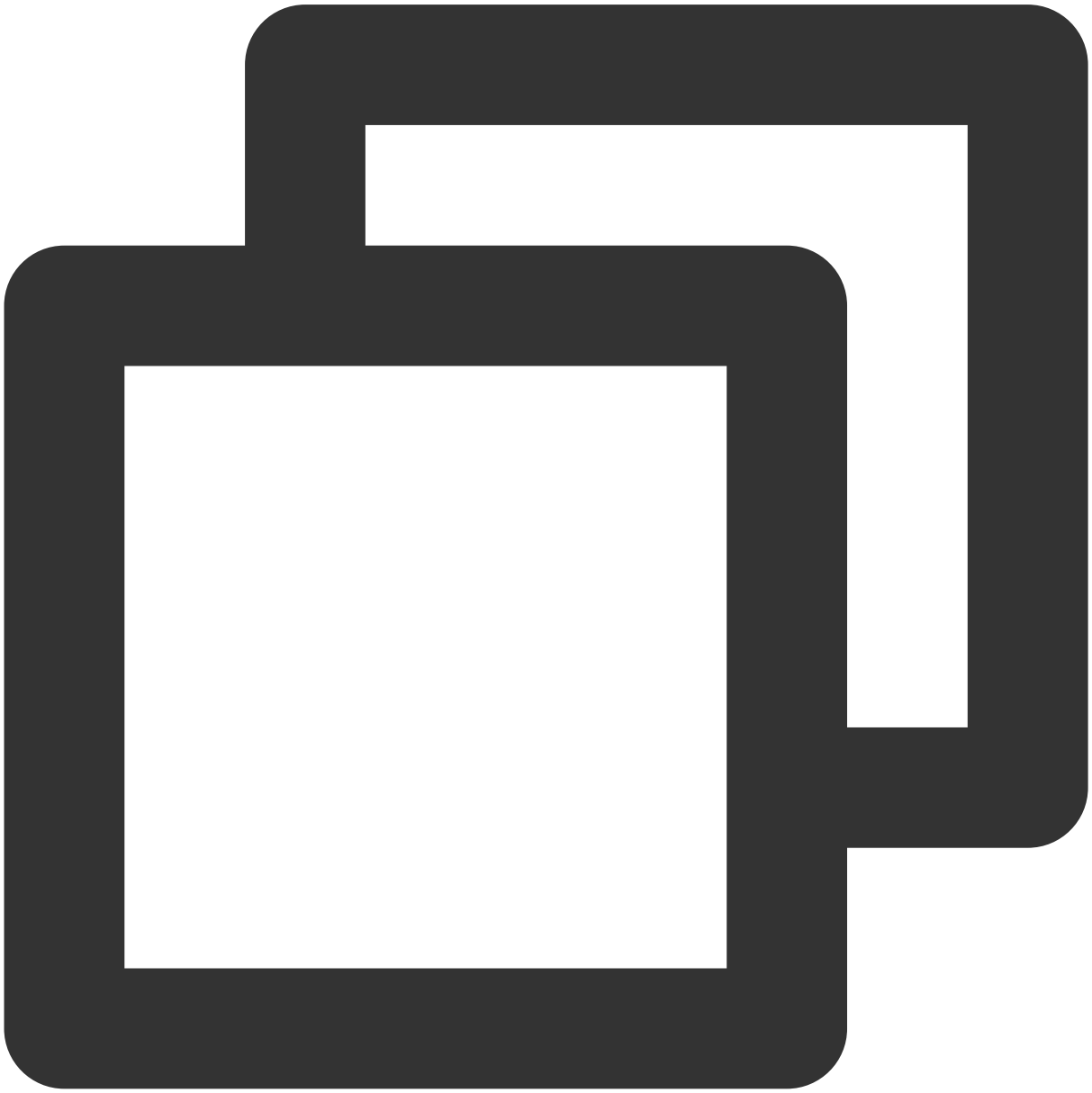
```
void onAnchorExit(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	退房用户 ID。

onAudienceEnter

收到观众进房通知。



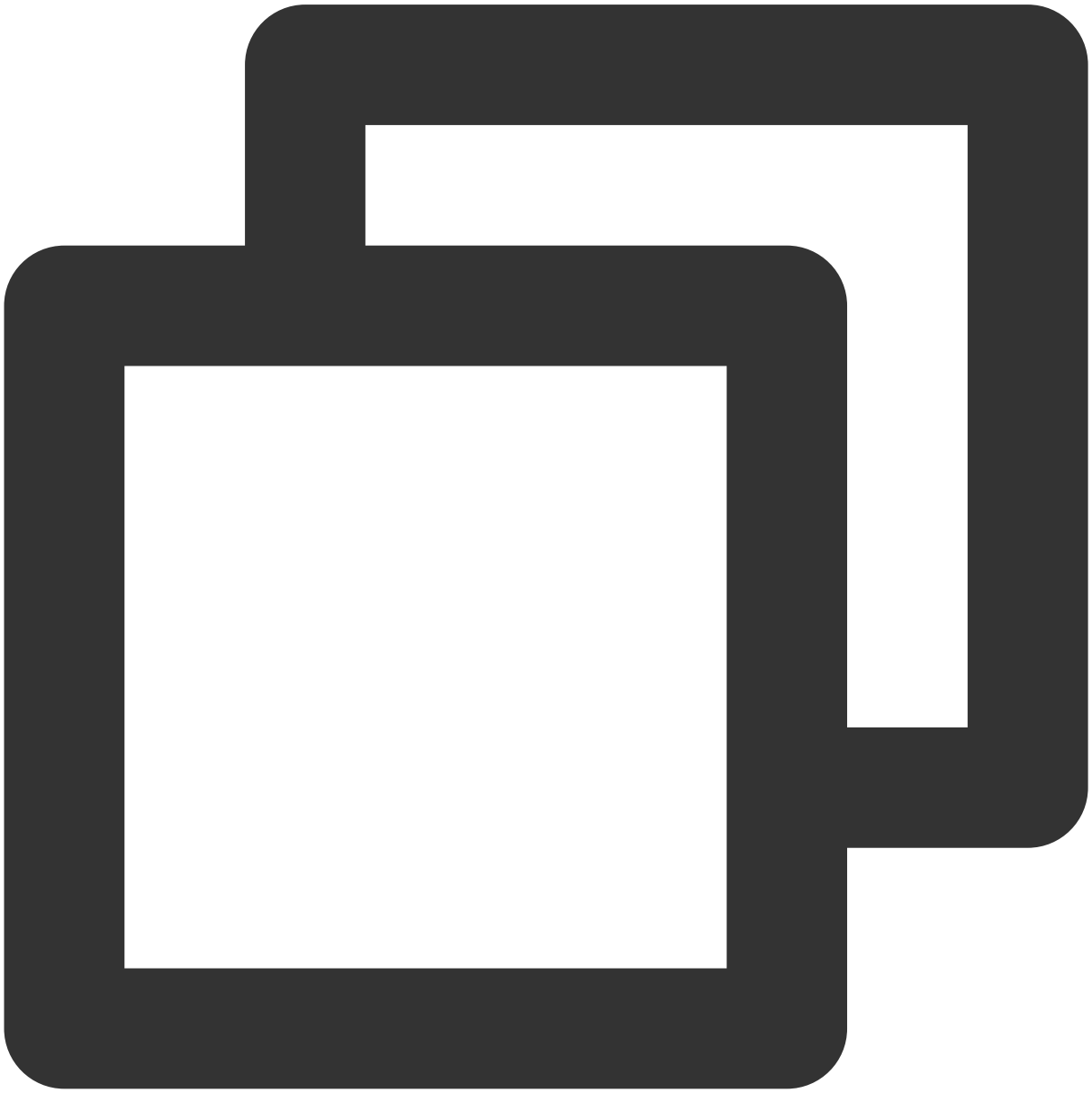
```
void onAudienceEnter(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	TRTCLiveUserInfo	进房观众信息。

onAudienceExit

收到观众退房通知。



```
void onAudienceExit(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

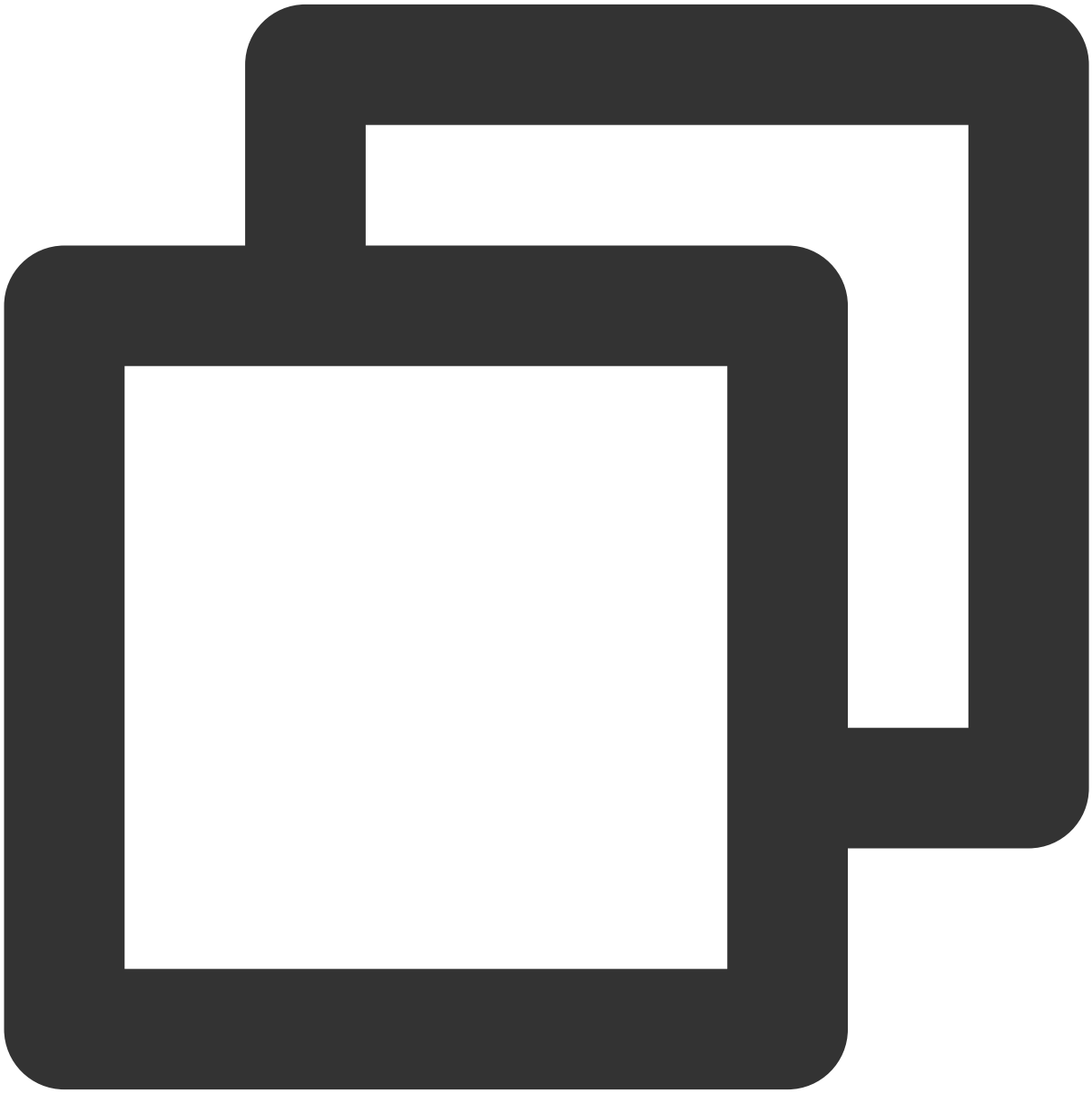
参数如下表所示：

参数	类型	含义
userInfo	TRTCLiveUserInfo	退房观众信息。

主播和观众连麦事件回调

onRequestJoinAnchor

主播收到观众连麦请求时的回调。



```
void onRequestJoinAnchor(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, String reason,
```

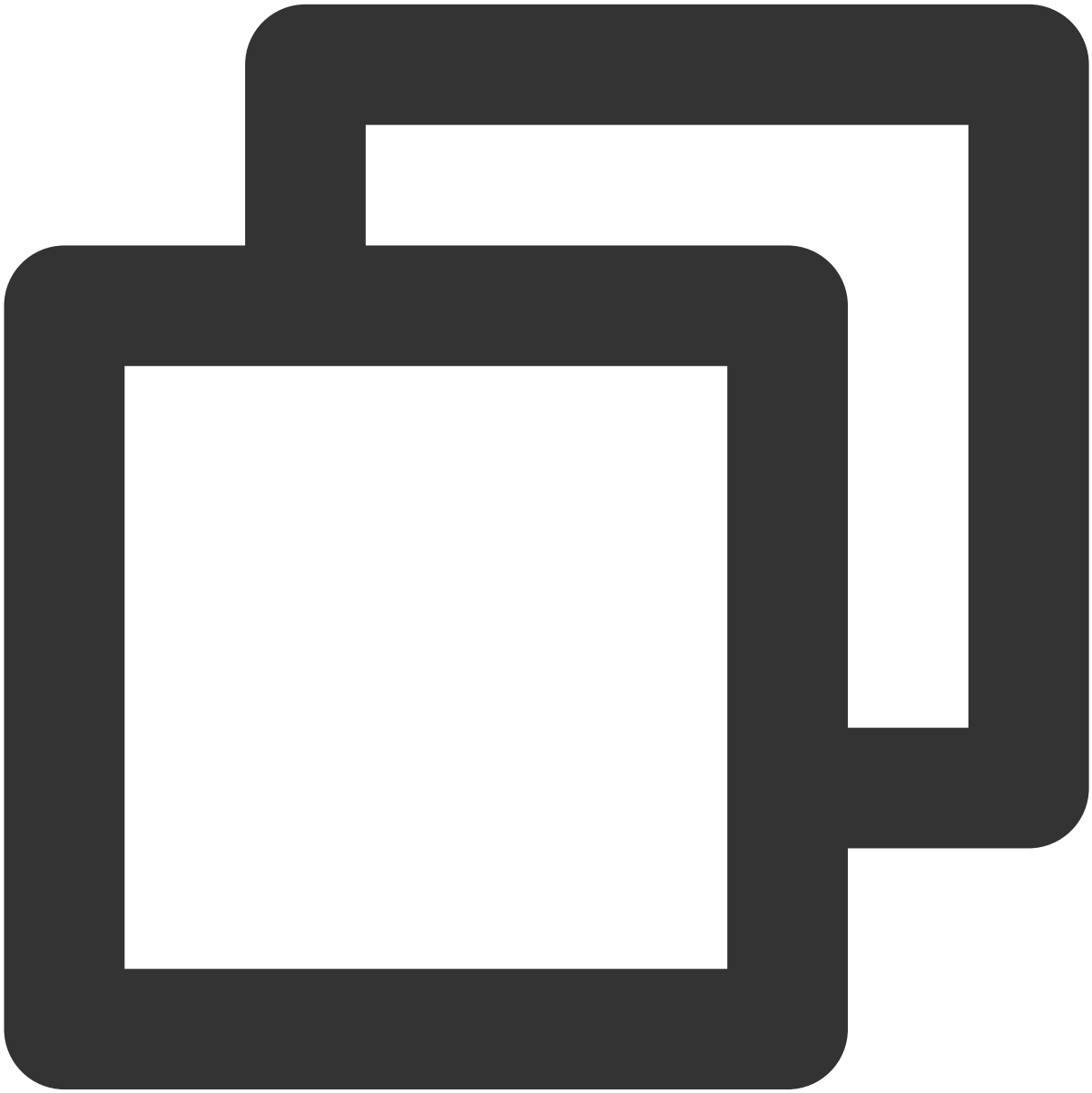
参数如下表所示：

参数	类型	含义
userInfo	TRTCLiveUserInfo	请求连麦观众信息。
reason	String	连麦原因描述。
timeout	int	处理请求的超时时间，如果上层超过该时间没有处理，则会自动将该次请求

		废弃。

onKickoutJoinAnchor

连麦观众收到被踢出连麦的通知。连麦观众收到被主播踢除连麦的消息，您需要调用 `TRTCLiveRoom` 的 `stopPublish()` 退出连麦。

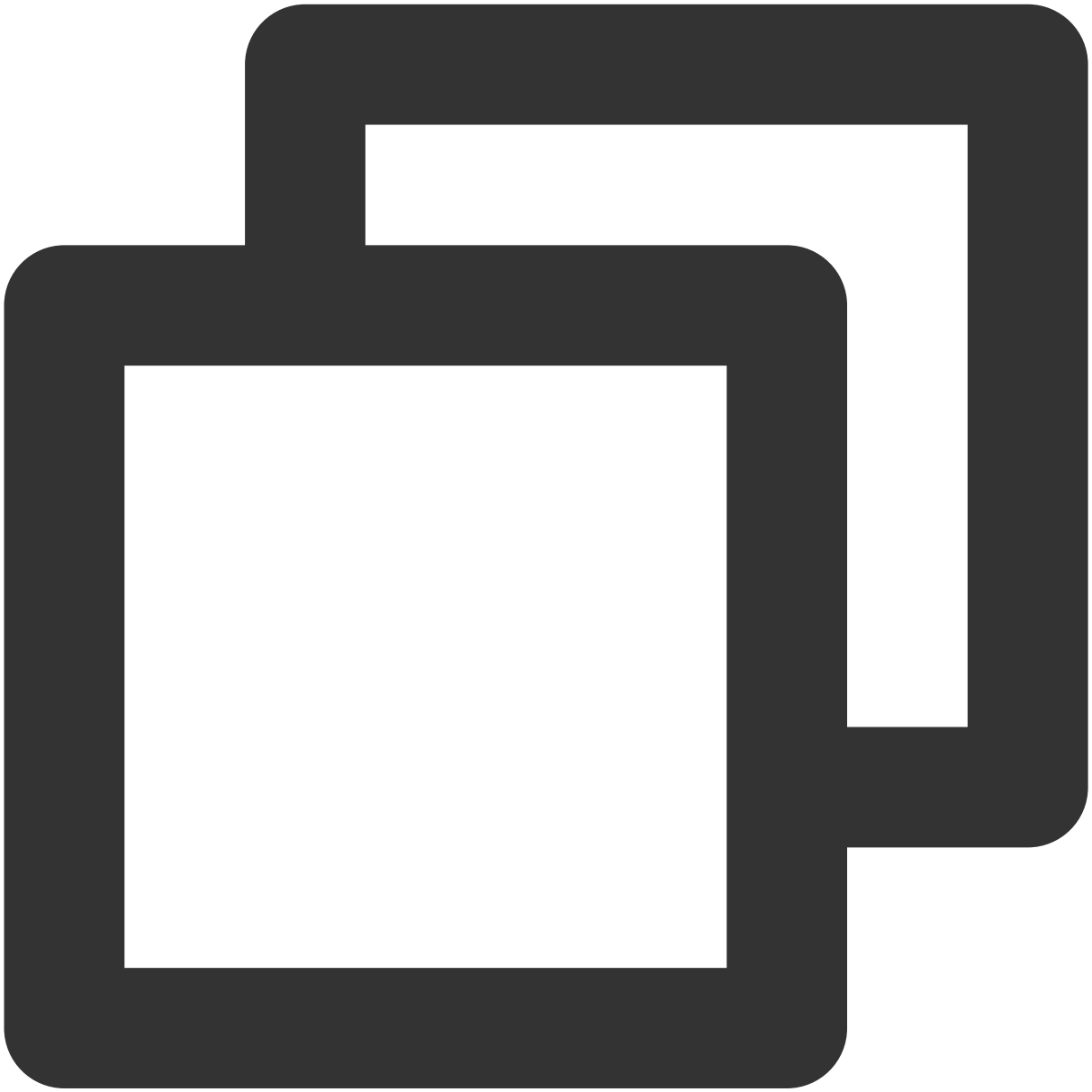


```
void onKickoutJoinAnchor();
```

主播 PK 事件回调

onRequestRoomPK

收到请求跨房 PK 通知。主播收到其他房间主播的 PK 请求，如果同意 PK ，您需要等待 `TRTCLiveRoomDelegate` 的 `onAnchorEnter()` 通知，然后调用 `startPlay()` 来播放邀约主播的流。



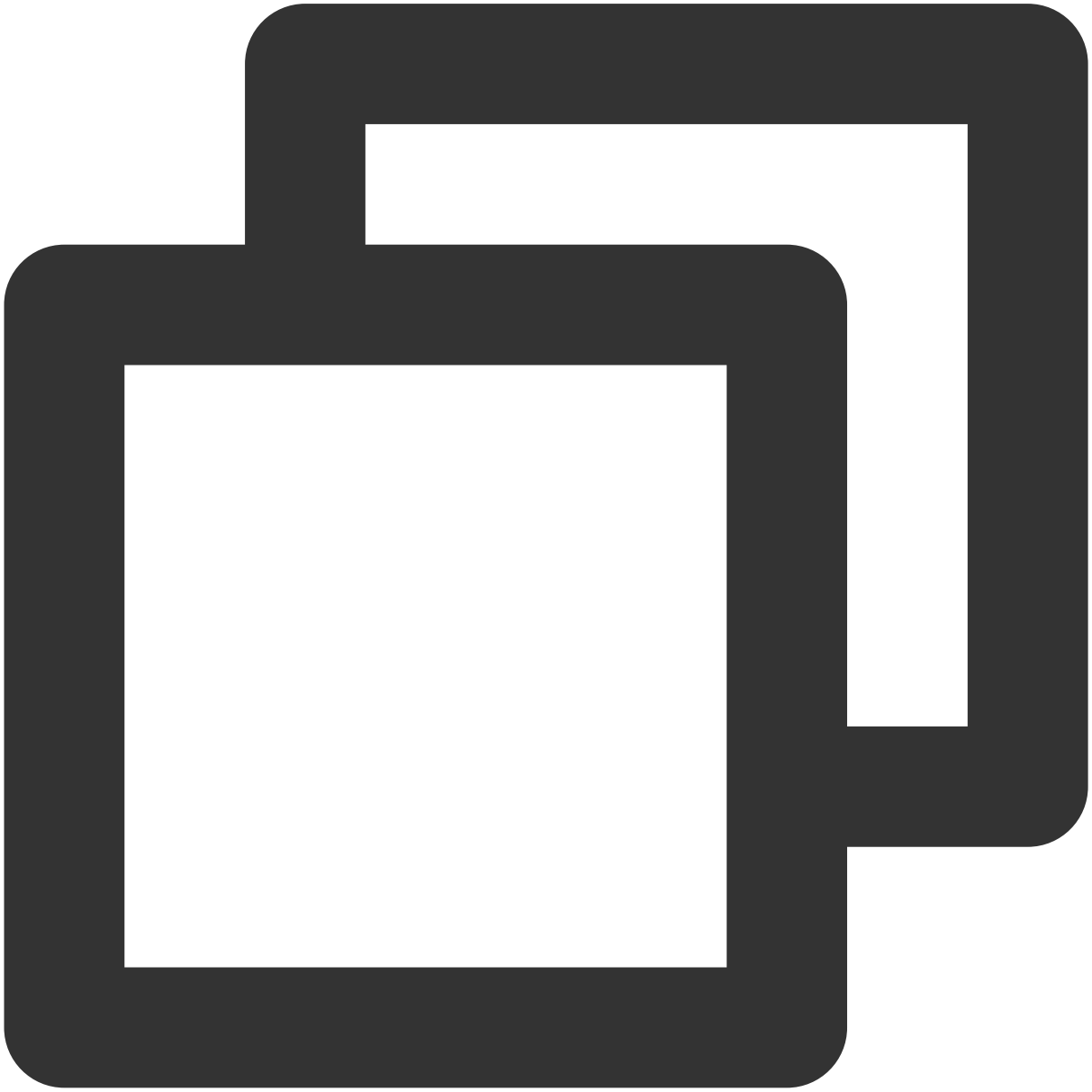
```
void onRequestRoomPK(TRTCLiveRoomDef.TRTCLiveUserInfo userInfo, int timeout);
```

参数如下表所示：

参数	类型	含义
userInfo	TRTCLiveUserInfo	发起跨房连麦的主播信息。
timeout	int	处理请求的超时时间。

onQuitRoomPK

收到断开跨房 PK 通知。

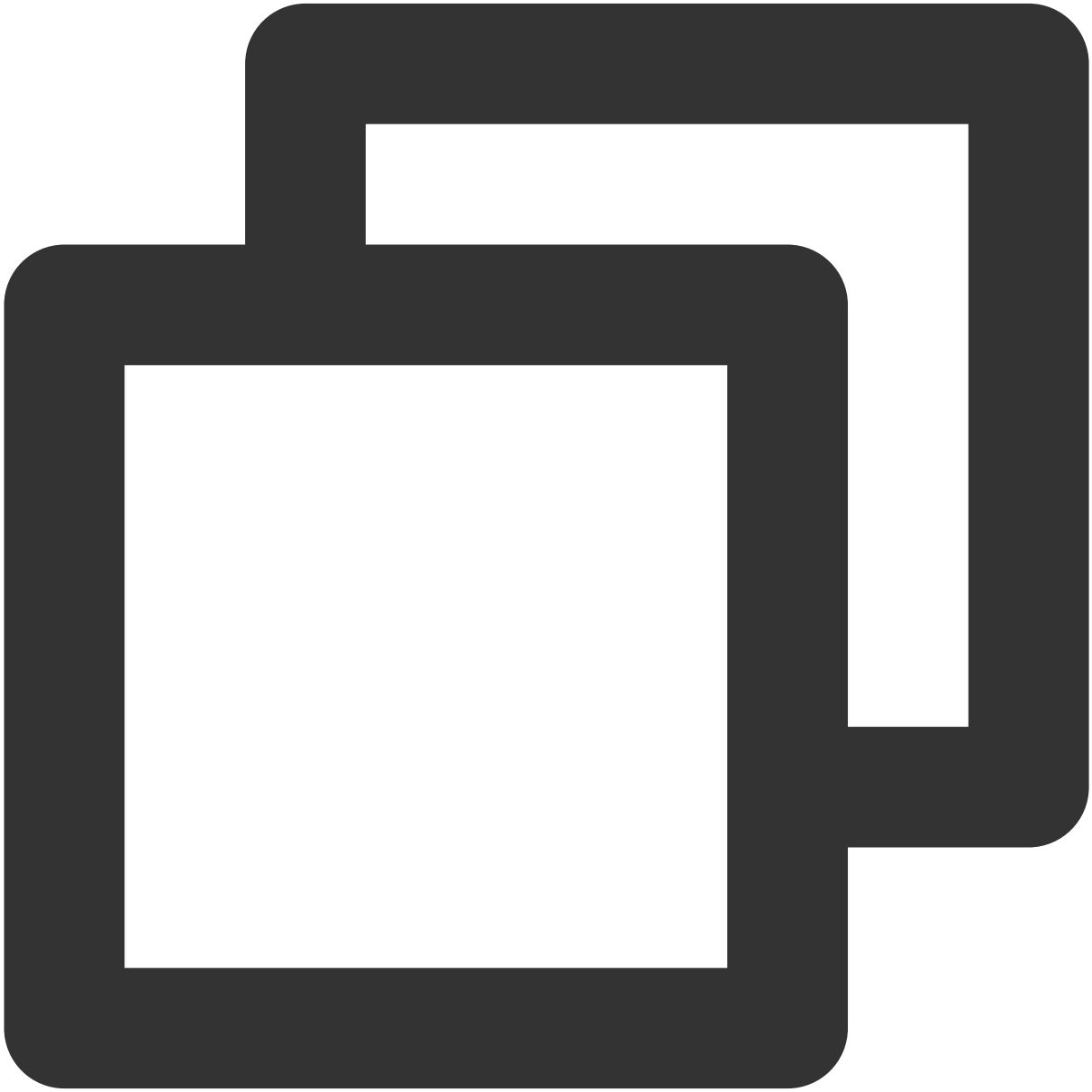


```
void onQuitRoomPK();
```

消息事件回调

onRecvRoomTextMsg

收到文本消息。



```
void onRecvRoomTextMsg(String message, TRTCLiveRoomDef.TRTCLiveUserInfo userInfo);
```

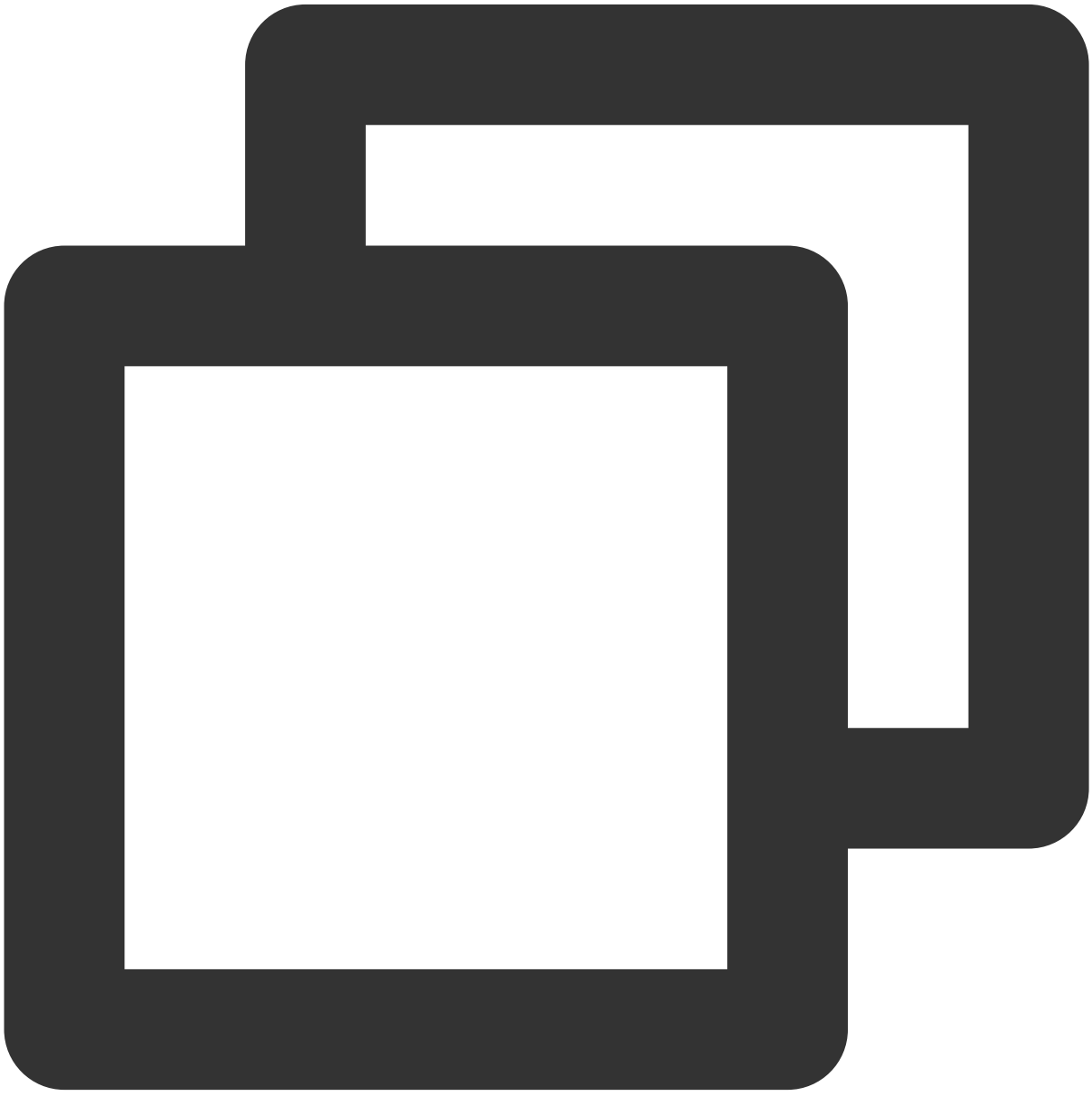
参数如下表所示：

参数	类型	含义

message	String	文本消息。
userInfo	TRTCLiveUserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。



```
void onRecvRoomCustomMsg(String cmd, String message, TRTCLiveRoomDef.TRTCLiveUserIn
```

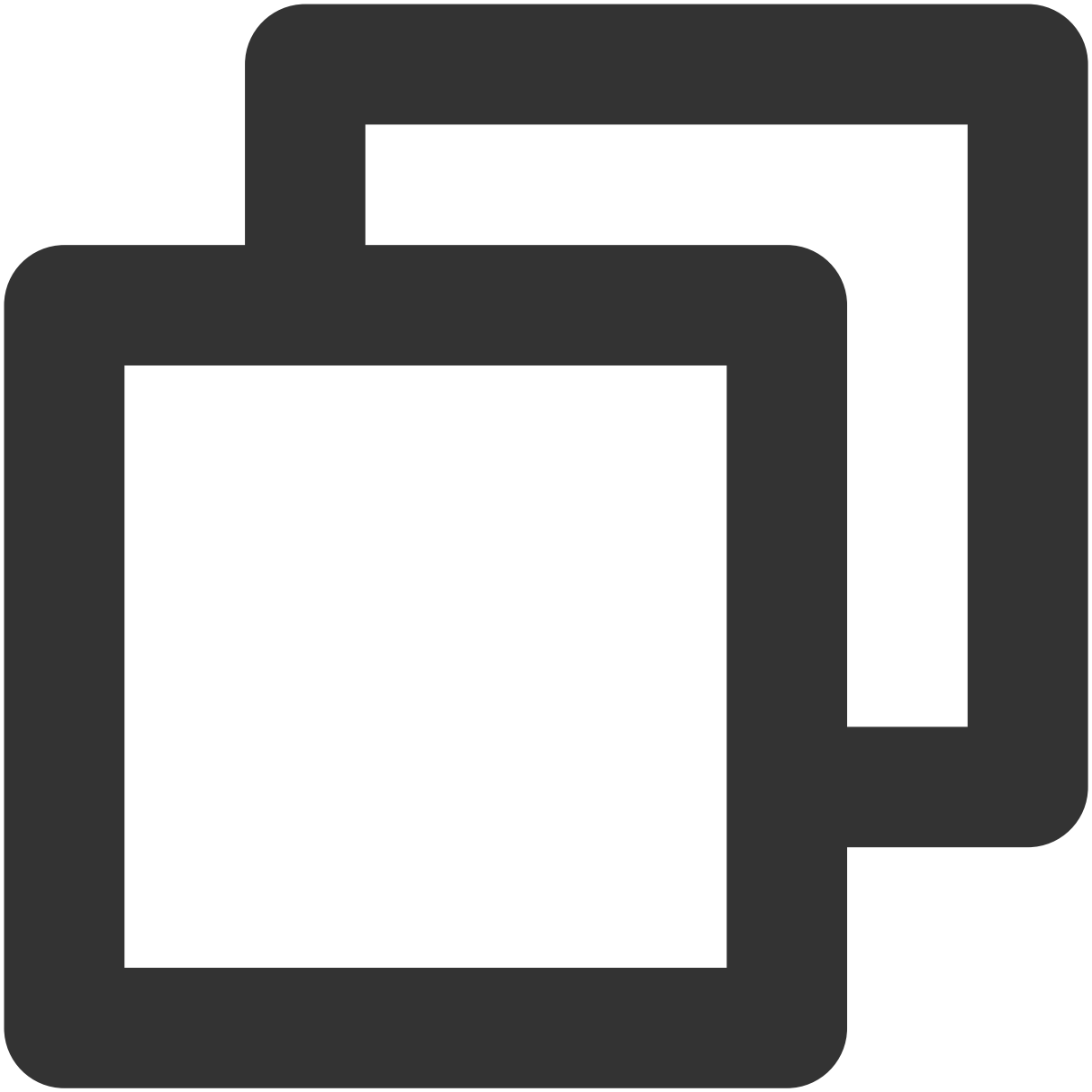
参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	TRTCLiveUserInfo	发送者用户信息。

TRTCAudioEffectManager

playBGM

播放背景音乐。



```
void playBGM(String url, int loopTimes, int bgmVol, int micVol, TRTCCloud.BGMNotify
```

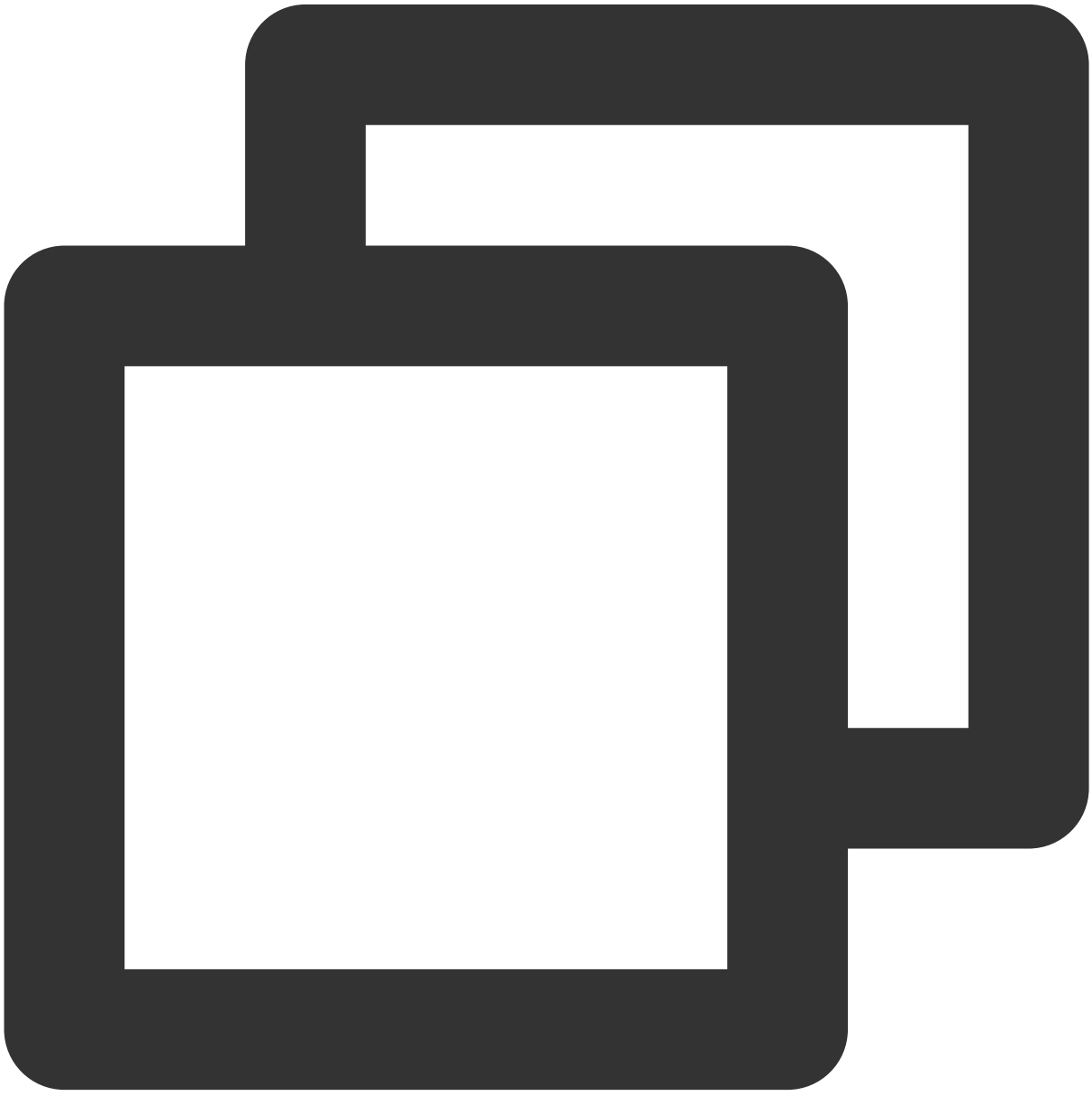
参数如下表所示：

参数	类型	含义
url	String	背景音乐文件路径。
loopTimes	int	循环次数
bgmVol	int	BGM 音量

micVol	int	采集音量
notify	TRTCCloud.BGMNotify	播放通知

stopBGM

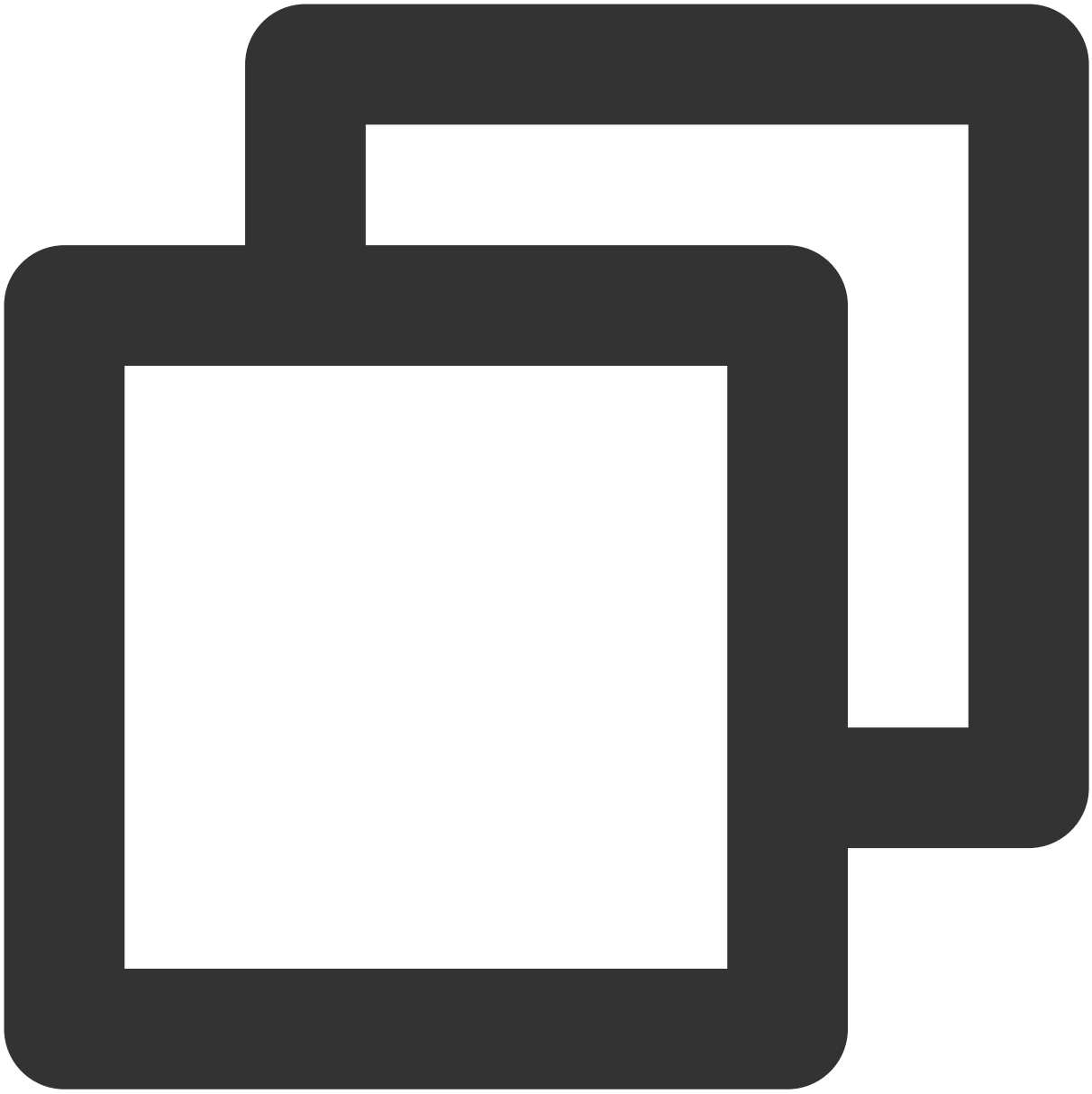
停止播放背景音乐。



```
void stopBGM();
```

pauseBGM

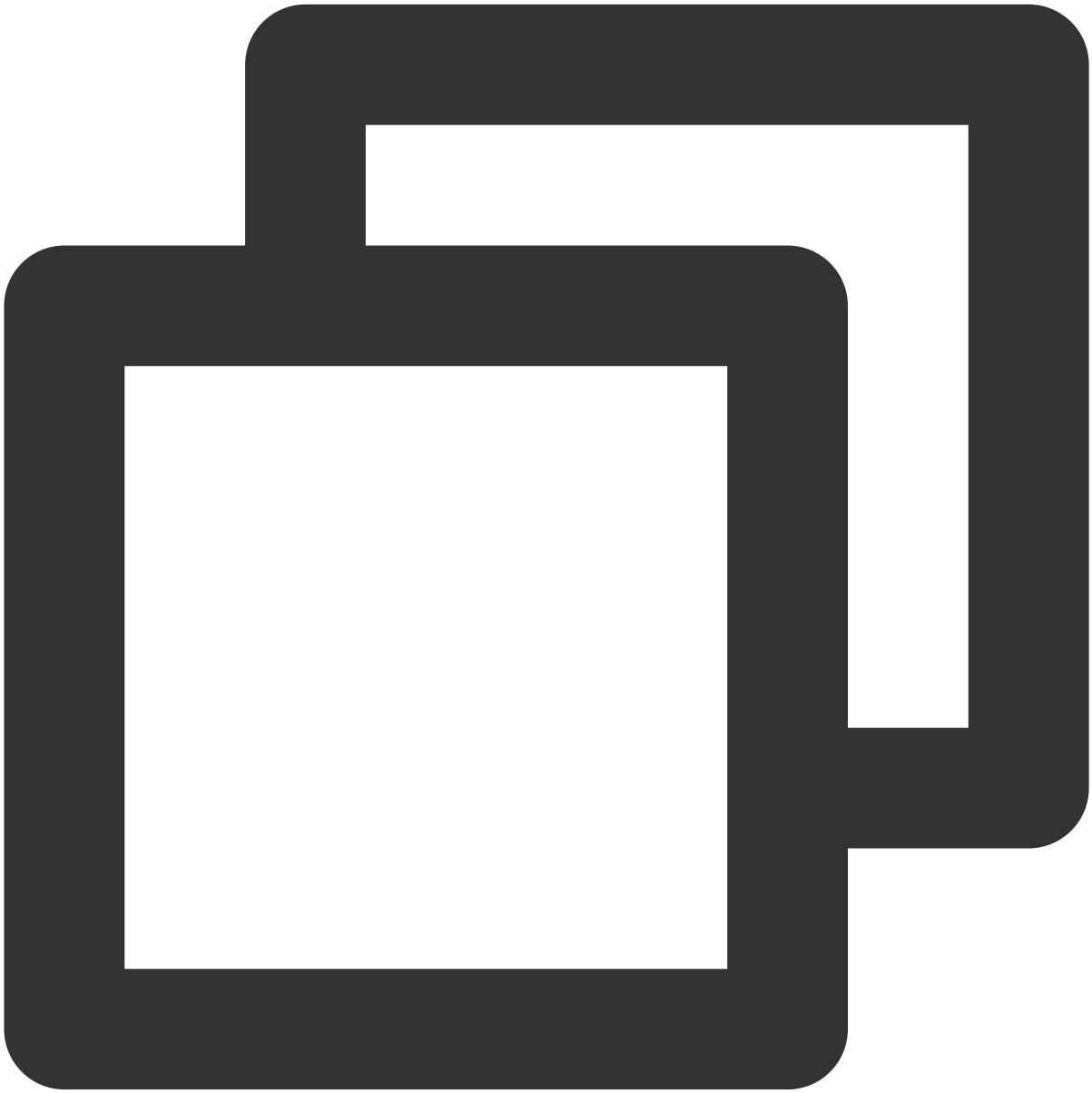
暂停播放背景音乐。



```
void pauseBGM();
```

resumeBGM

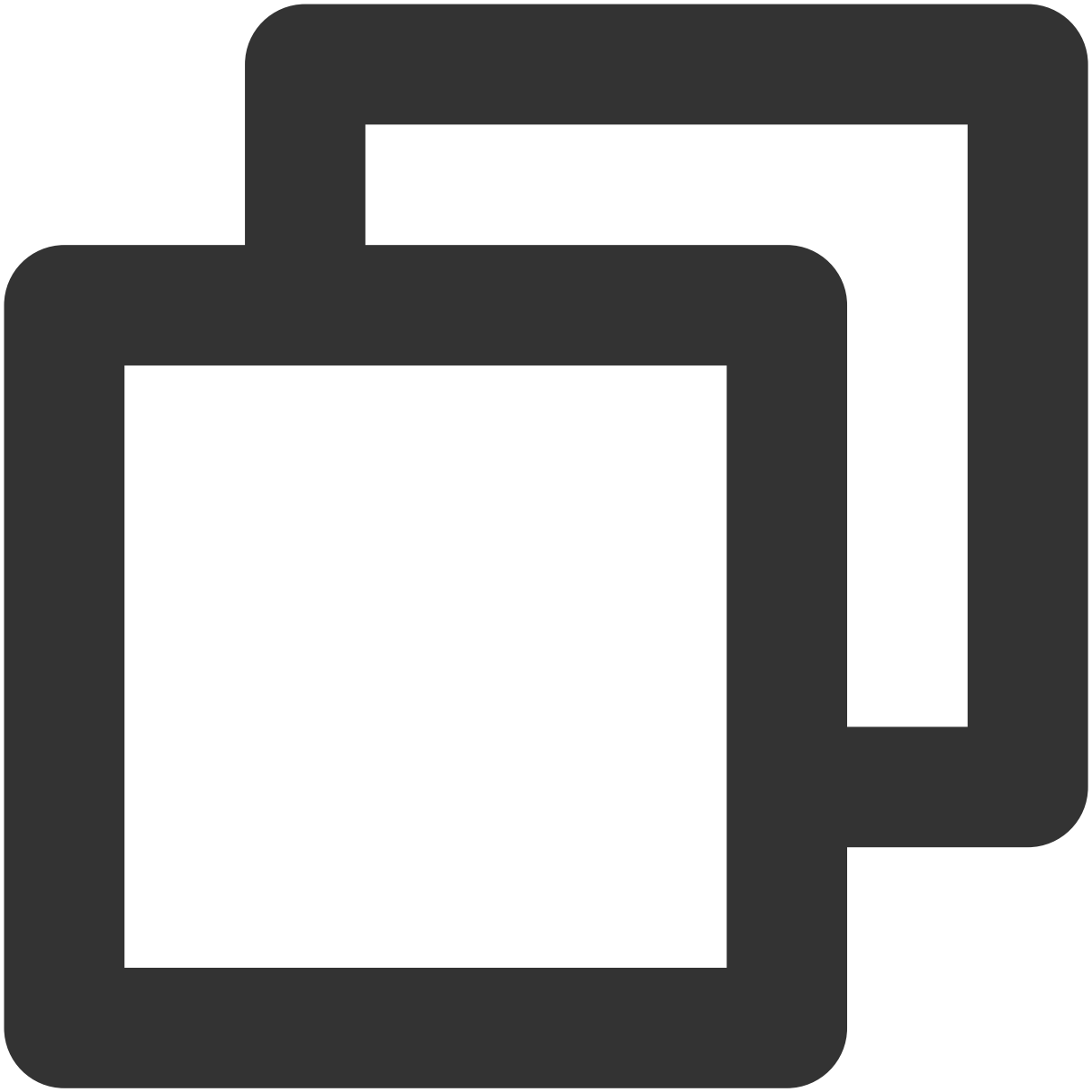
继续播放背景音乐。



```
void resumeBGM();
```

setBGMVolume

设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景音的音量大小。



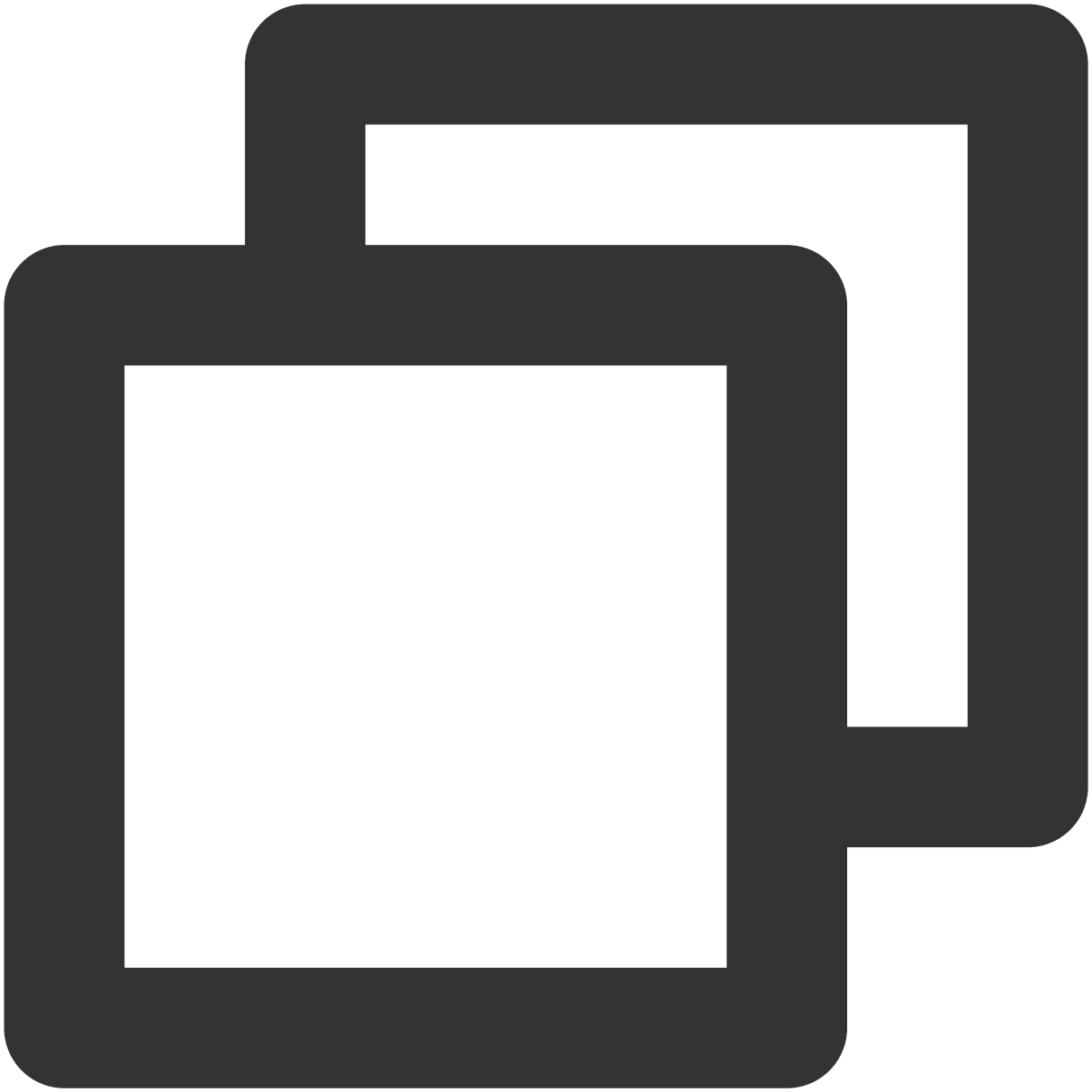
```
void setBGMVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	音量大小，100表示正常音量，取值范围为0 - 100，默认值为100。

setBGMPosition

设置背景音乐播放进度。



```
int setBGMPosition(int position);
```

参数如下表所示：

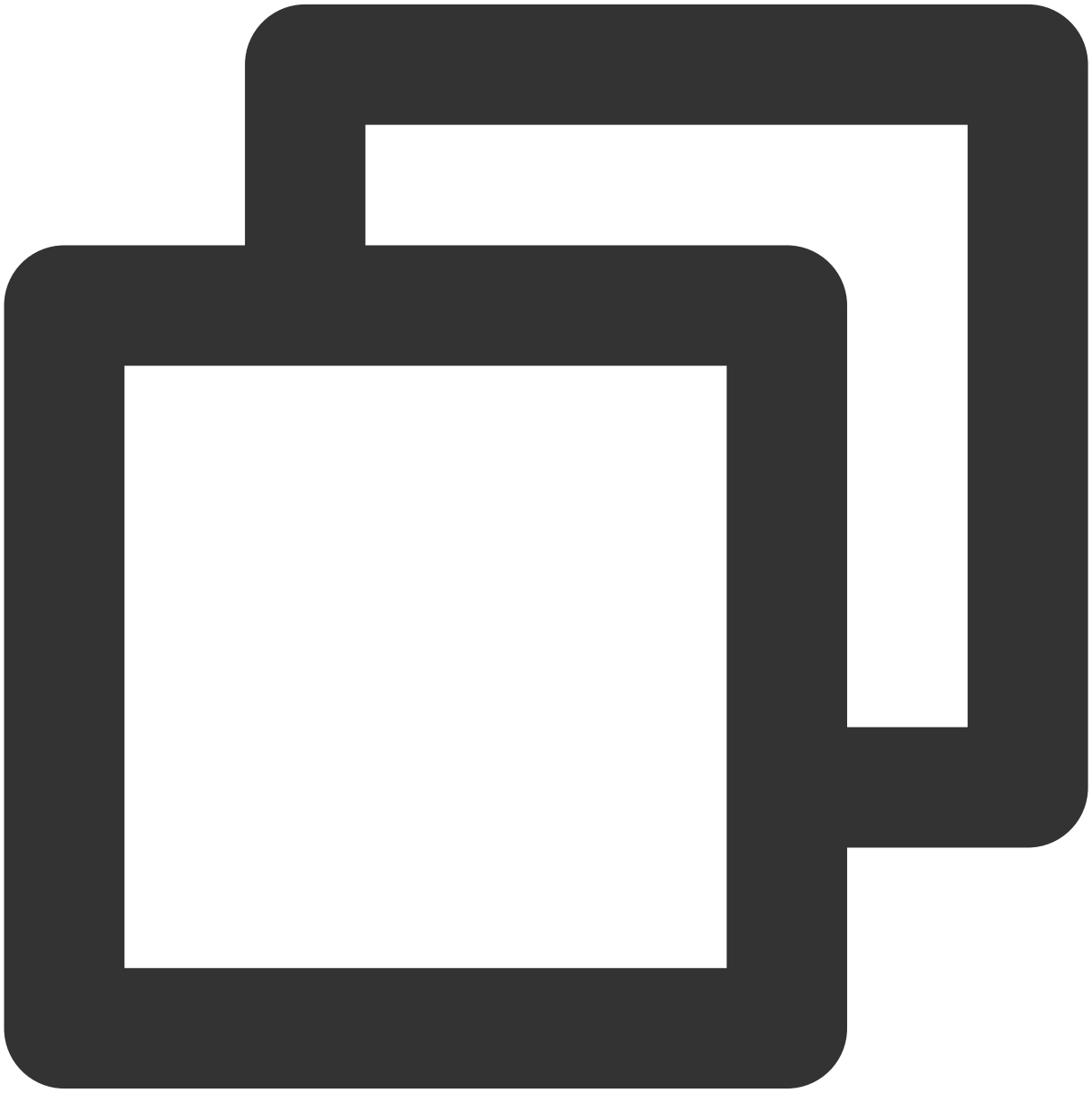
参数	类型	含义
position	int	背景音乐播放进度，单位为毫秒（ms）。

返回

0：成功。

setMicVolume

设置麦克风的音量大小，播放背景音乐混音时使用，用来控制麦克风音量大小。



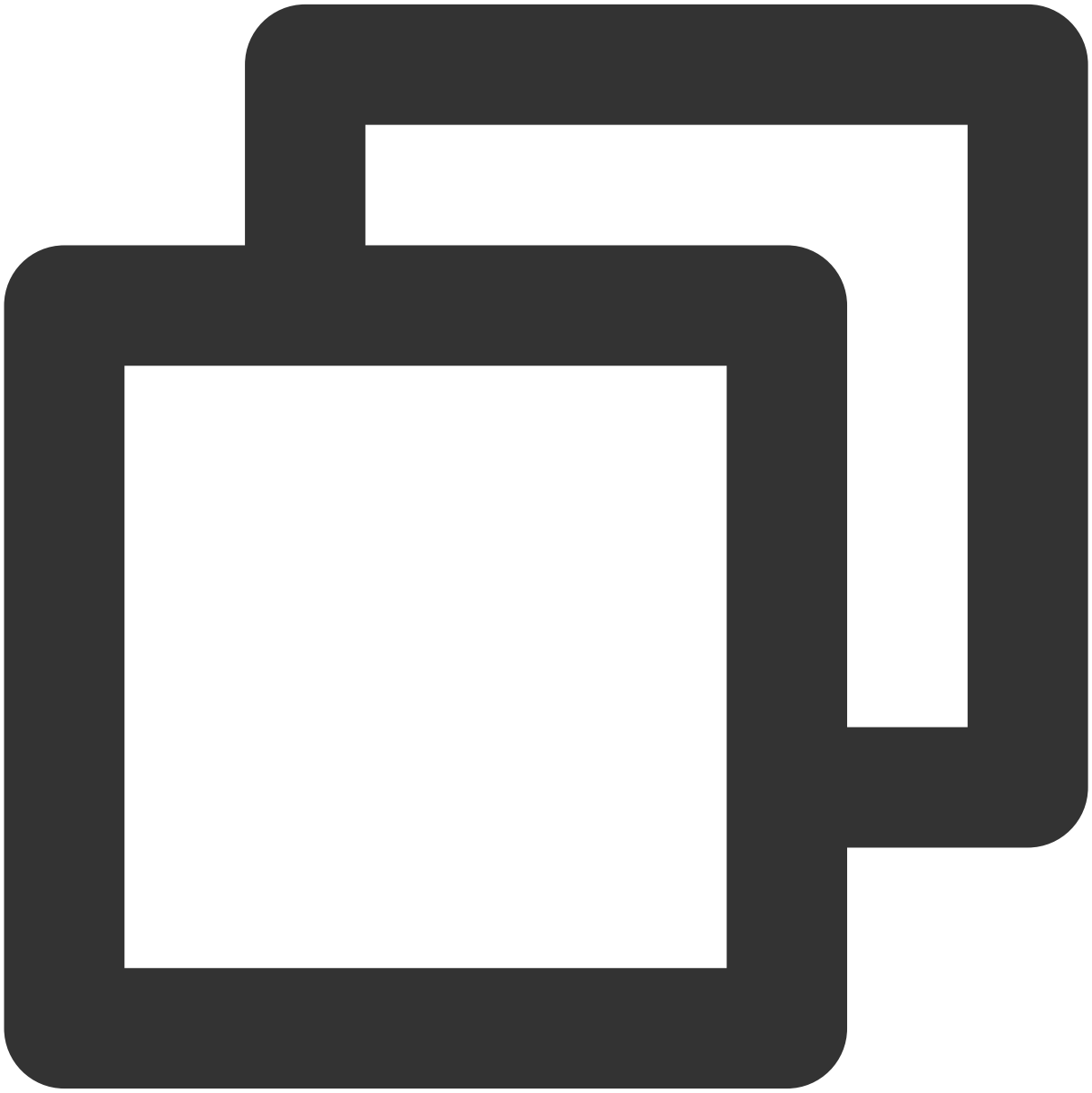
```
void setMicVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	Int	音量大小，取值0 - 100，默认值为100。

setReverbType

设置混响效果。



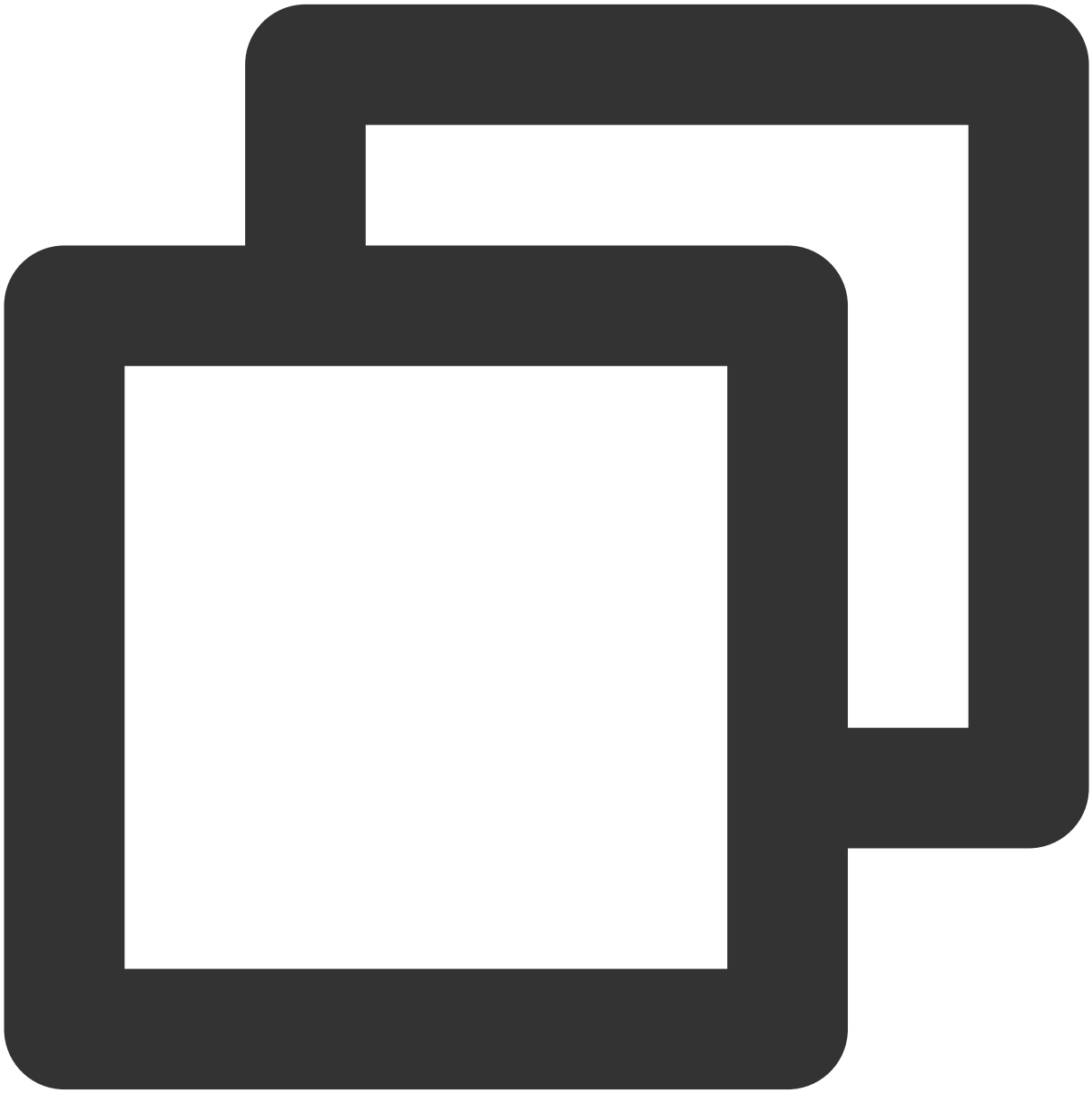
```
void setReverbType(int reverbType);
```

参数如下表所示：

参数	类型	含义
reverbType	int	混响类型，详情请参见 `TRTCCloudDef` 中的 TRTC_REVERB_TYPE 定义。

setVoiceChangerType

设置变声类型。



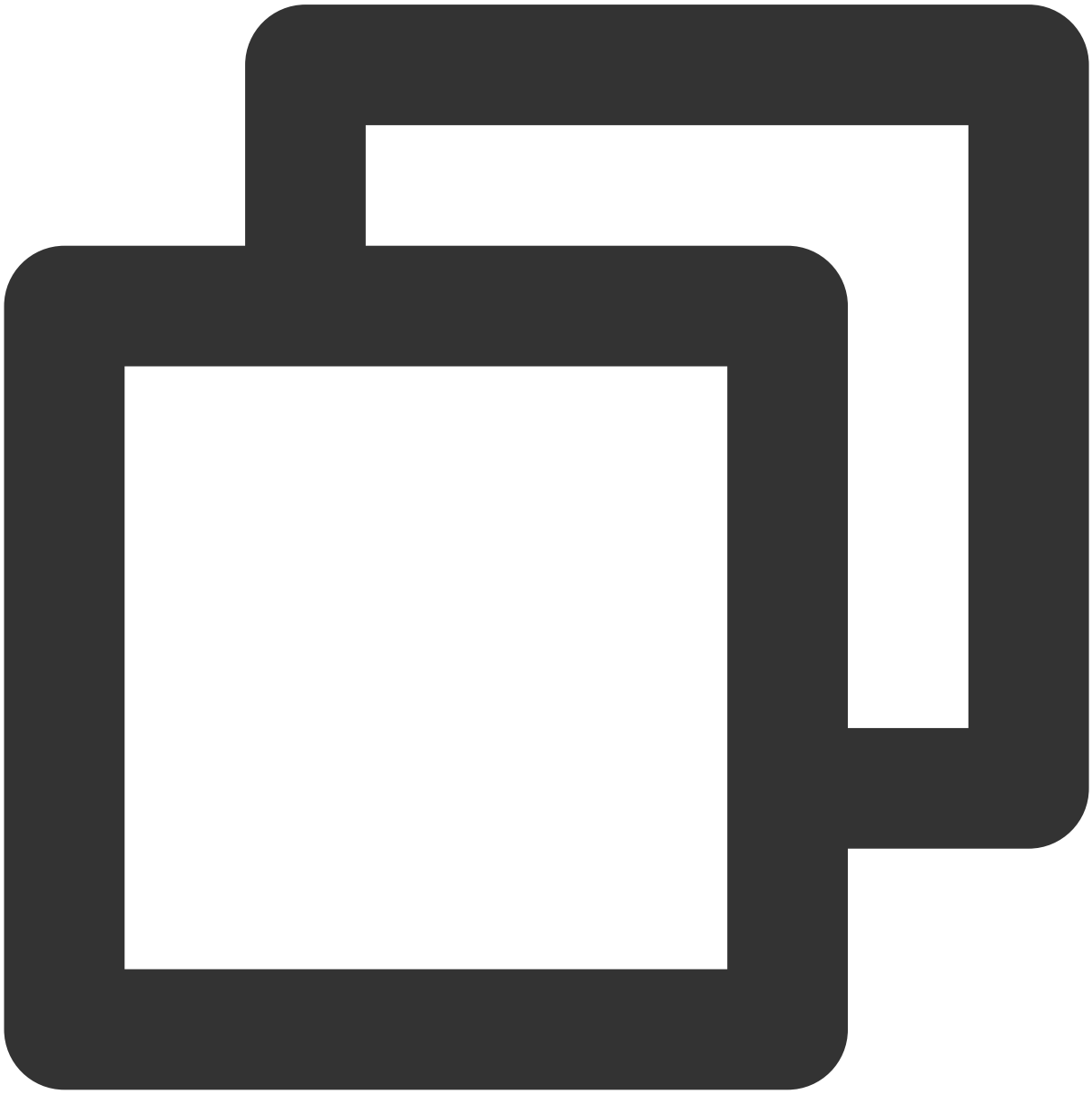
```
void setVoiceChangerType(int type);
```

参数如下表所示：

参数	类型	含义
type	int	混响类型，详情请参见 `TRTCCloudDef` 中的 TRTC_VOICE_CHANGER_TYPE 定义。

playAudioEffect

播放音效，每个音效都需要您指定具体的 ID，您可以通过该 ID 对音效的开始、停止、音量等进行设置。支持 aac、mp3 以及 m4a 格式。



```
void playAudioEffect(int effectId, String path, int count, boolean publish, int vol
```

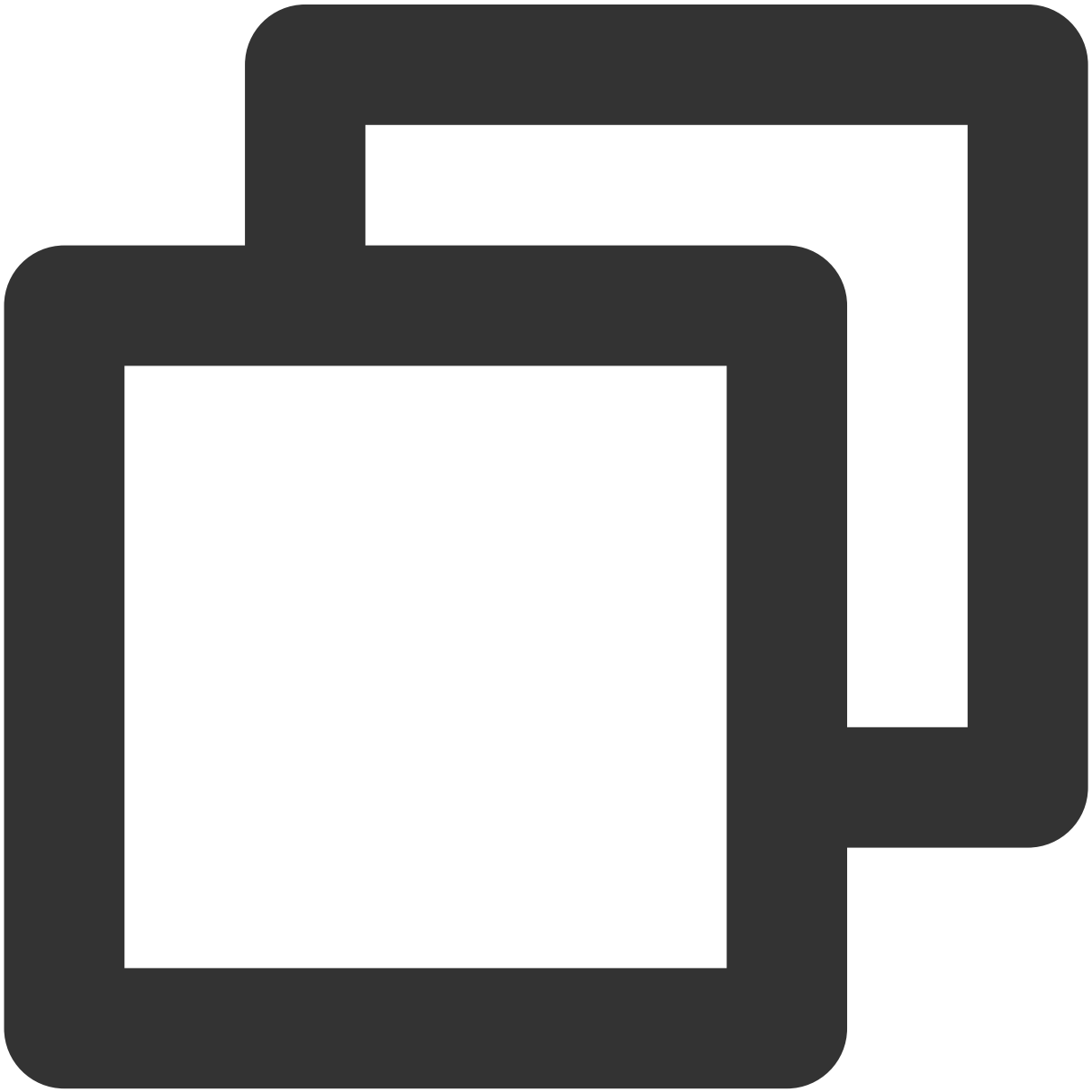
参数如下表所示：

参数	类型	含义
effectId	int	音效 ID。

path	String	音效路径。
count	int	循环次数。
publish	boolean	是否推送 / true 推送给观众, false 本地预览。
volume	int	音量大小，取值范围为0 - 100，默认值为100。

pauseAudioEffect

暂停音效播放。



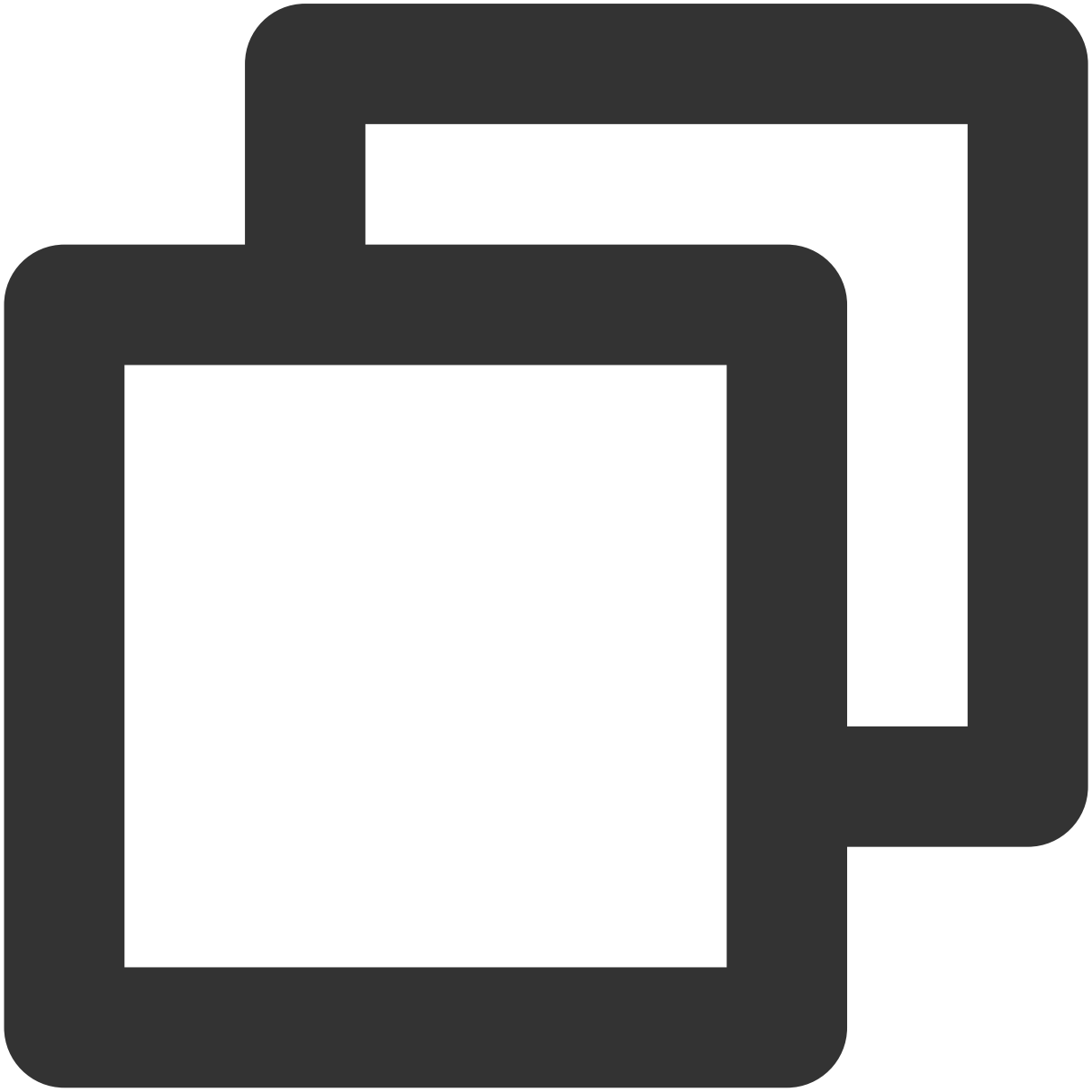
```
void pauseAudioEffect(int effectId);
```

参数如下表所示：

参数	类型	含义
effectId	int	音效 ID。

resumeAudioEffect

恢复音效播放。



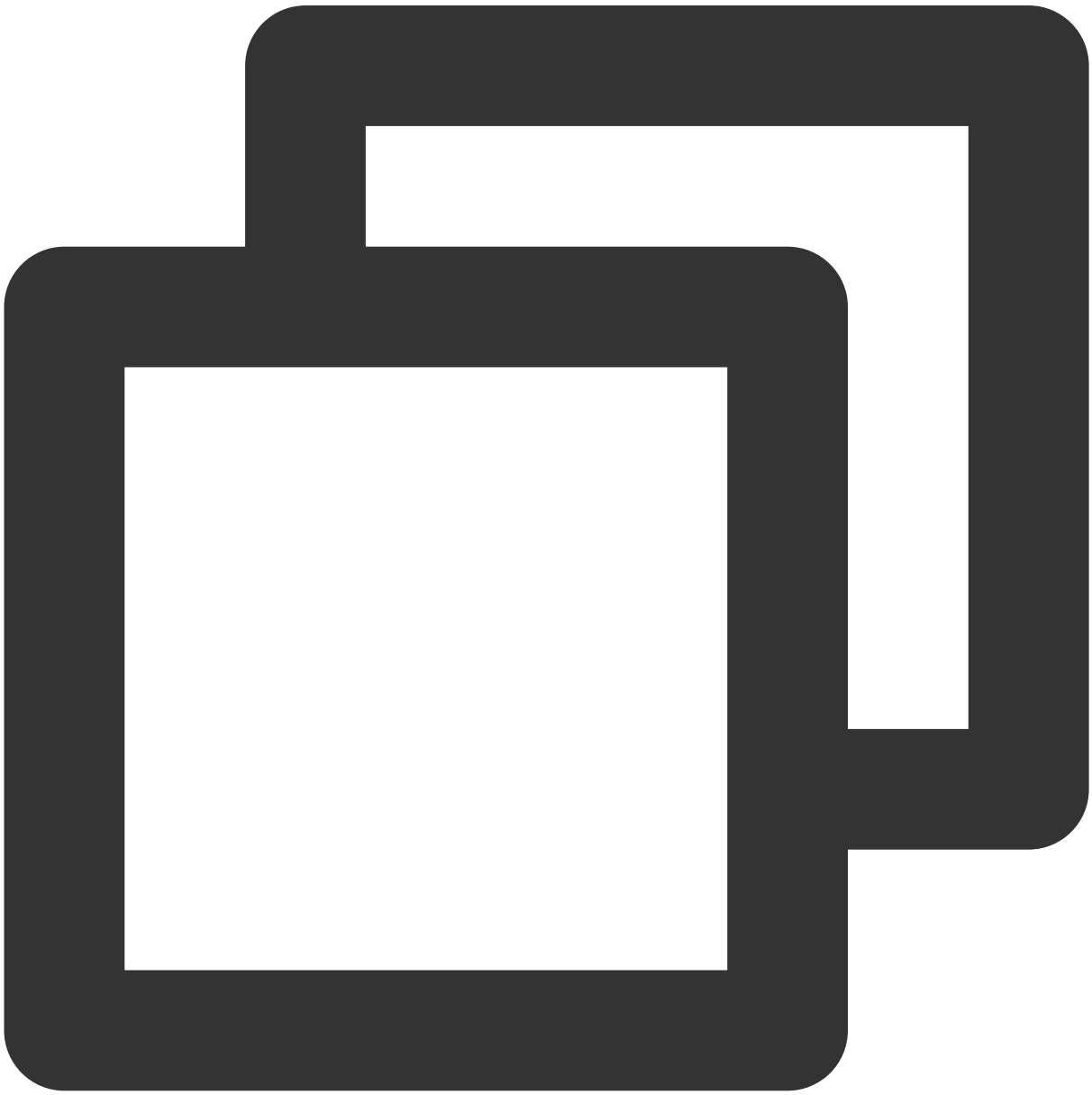
```
void resumeAudioEffect(int effectId);
```

参数如下表所示：

参数	类型	含义
effectId	int	音效 ID。

stopAudioEffect

停止音效播放。



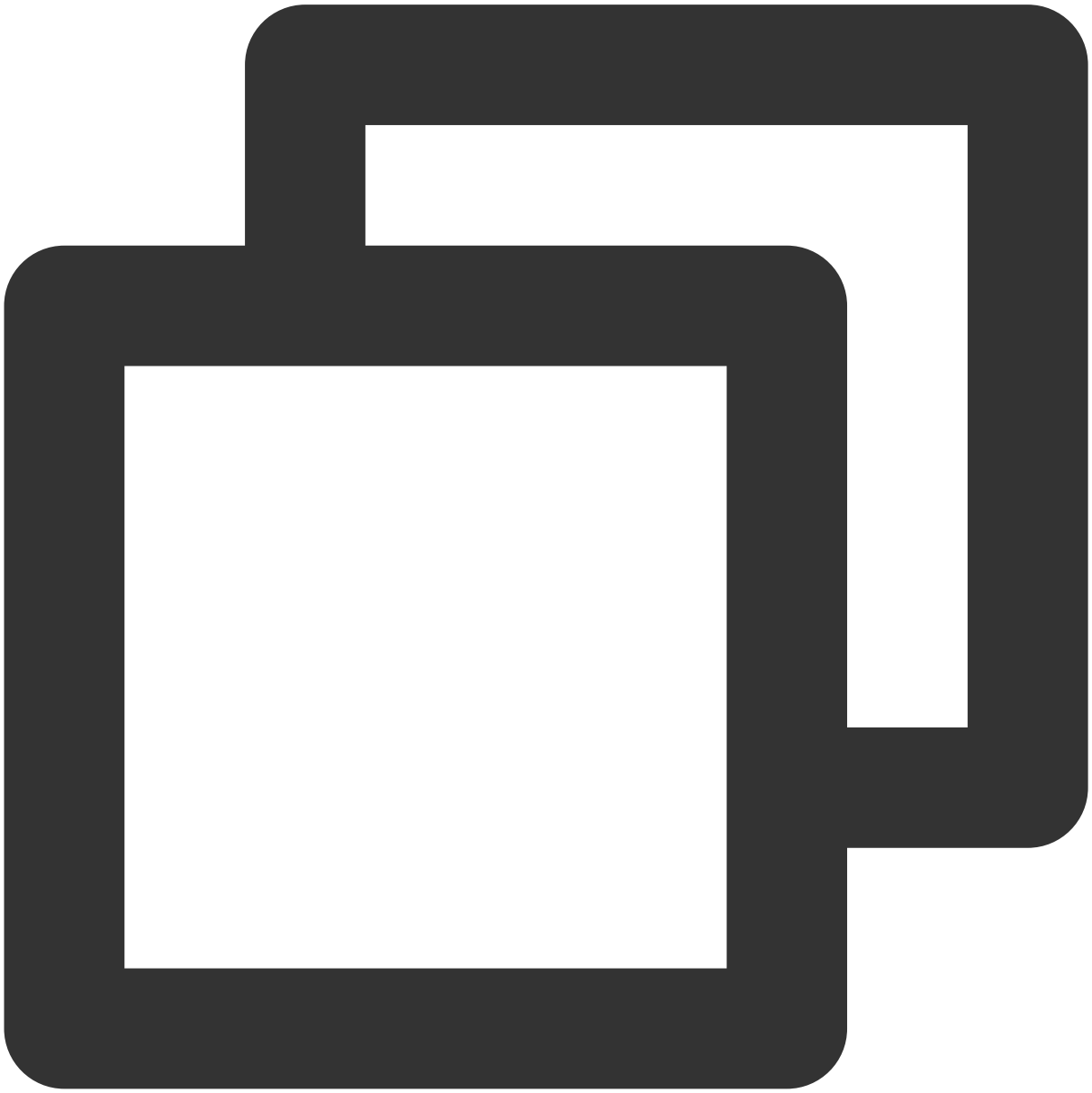
```
void stopAudioEffect(int effectId);
```

参数如下表所示：

参数	类型	含义
effectId	int	音效 ID。

stopAllAudioEffects

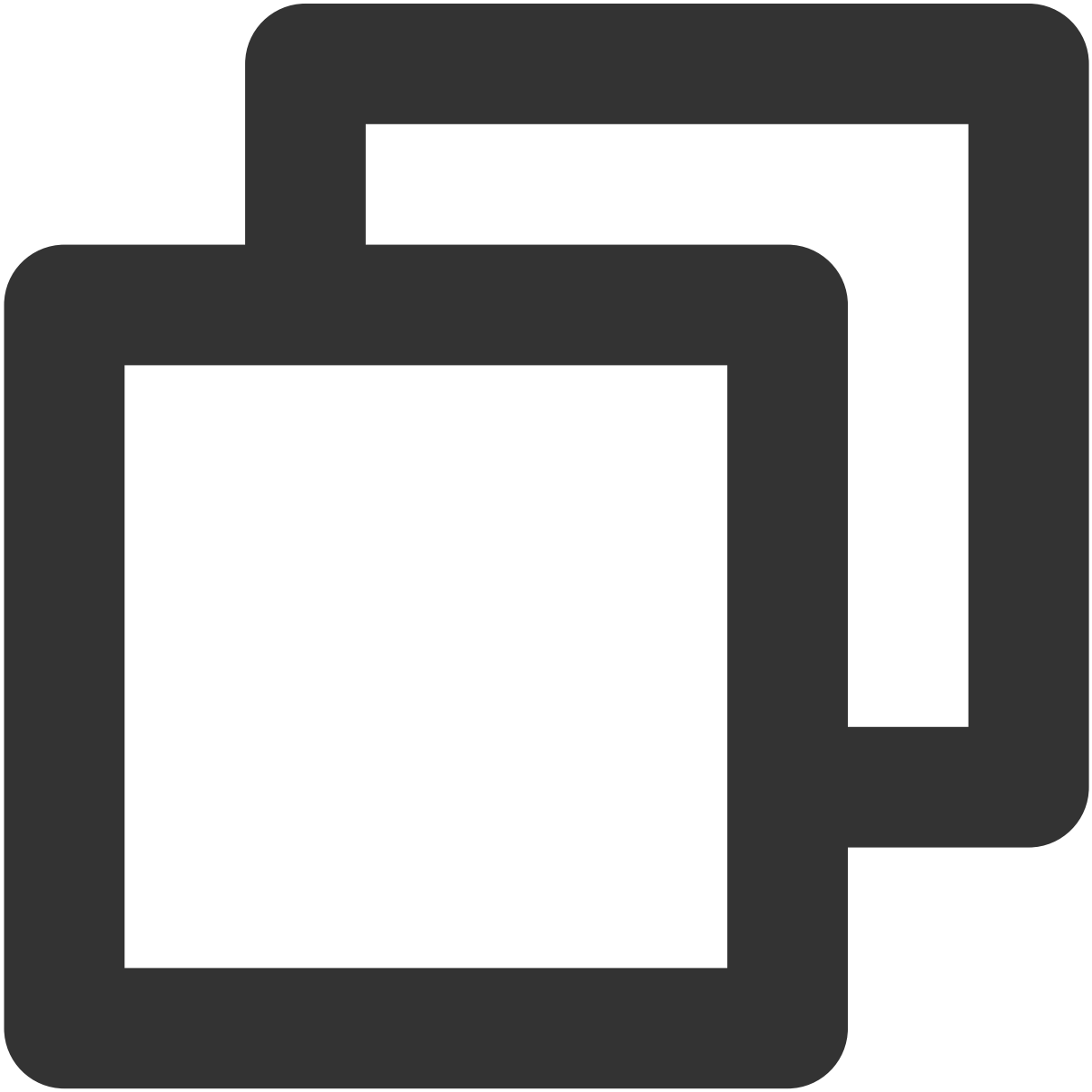
停止全部音效播放。



```
void stopAllAudioEffects();
```

setAudioEffectVolume

设置音效音量。



```
void setAudioEffectVolume(int effectId, int volume);
```

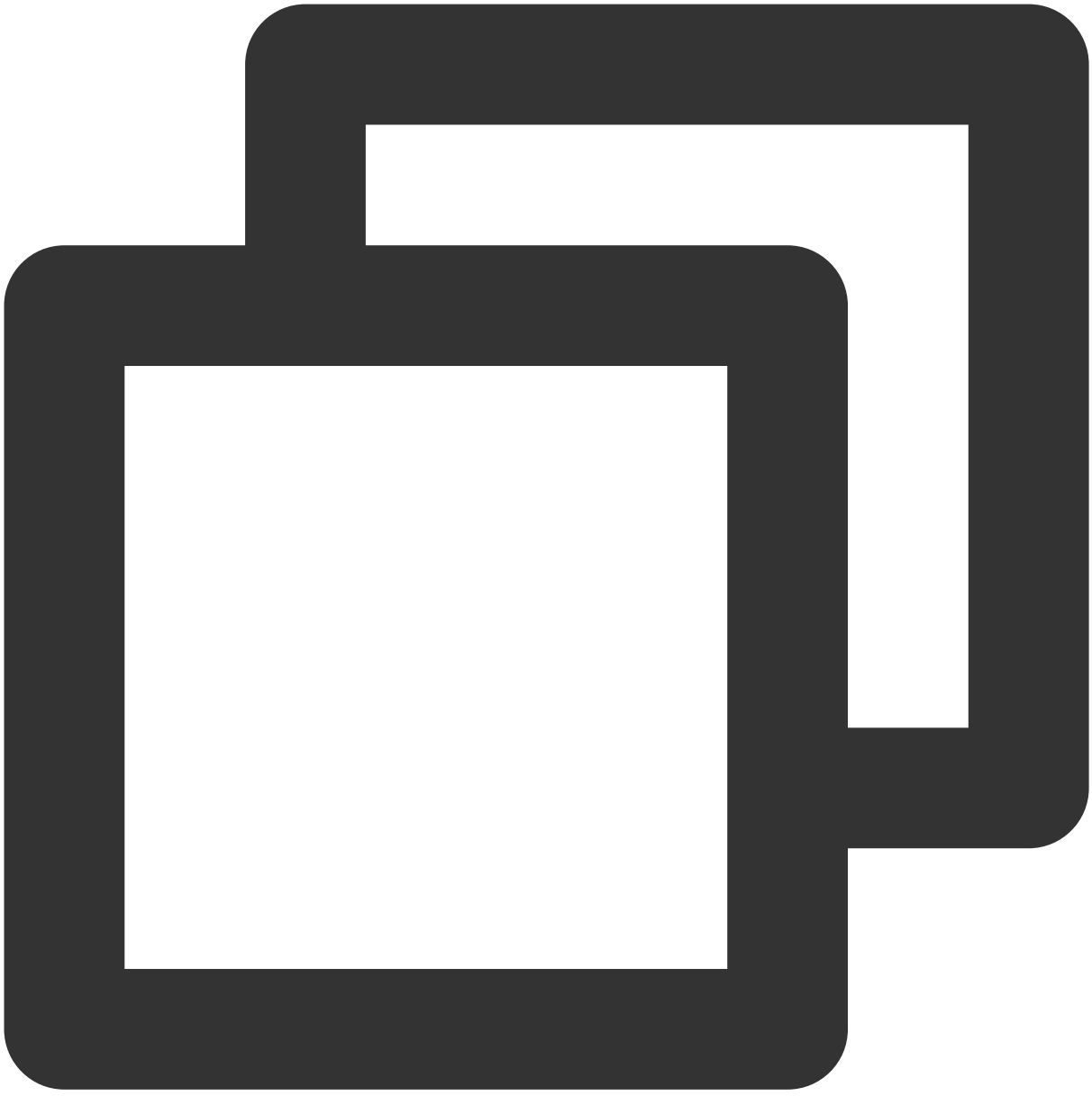
参数如下表所示：

参数	类型	含义

effectId	int	音效 ID。
volume	int	音量大小，取值范围为0 - 100，默认值为100。

setAllAudioEffectsVolume

设置所有音效的音量。



```
void setAllAudioEffectsVolume(int volume);
```

参数如下表所示：

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

在线 K 歌
快速集成（TUIKaraoke）
iOS

最近更新时间：2022-11-10 11:57:41

组件介绍

TUIKaraoke 是一个开源的音视频 UI 组件，通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。
TUIKaraoke 同时支持 Android 平台的源代码，基本功能如下图所示：

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

版权所有：腾讯云计算（北京）有限责任公司



组件集成

步骤一：下载并导入 TUIKaraoke 组件

单击进入 Github，选择克隆/下载代码，然后拷贝 iOS目录下的 `Source`、`Resources`、`TXAppBasic` 文件夹和 `TUIKaraoke.podspec` 文件到您的工程中，并完成如下导入动作：

- 在您的 `Podfile` 文件内添加导入命令，参考如下：

```
pod 'TUIKaraoke', :path => "./", :subspecs => ["TRTC"]
pod 'TXLiteAVSDK_TRTC'
pod 'TXAppBasic', :path => "TXAppBasic/"
```

- 打开终端，进入 `Podfile` 文件所在目录下执行安装命令，参考如下：

```
pod install
```

步骤二：配置权限

在您的工程的 info.plist文件中配置 App 的权限，SDK 需要以下权限（iOS 系统需要动态申请麦克风）：

```
<key>NSMicrophoneUsageDescription</key>
<string>Karaoke 需要访问您的麦克风权限</string>
```

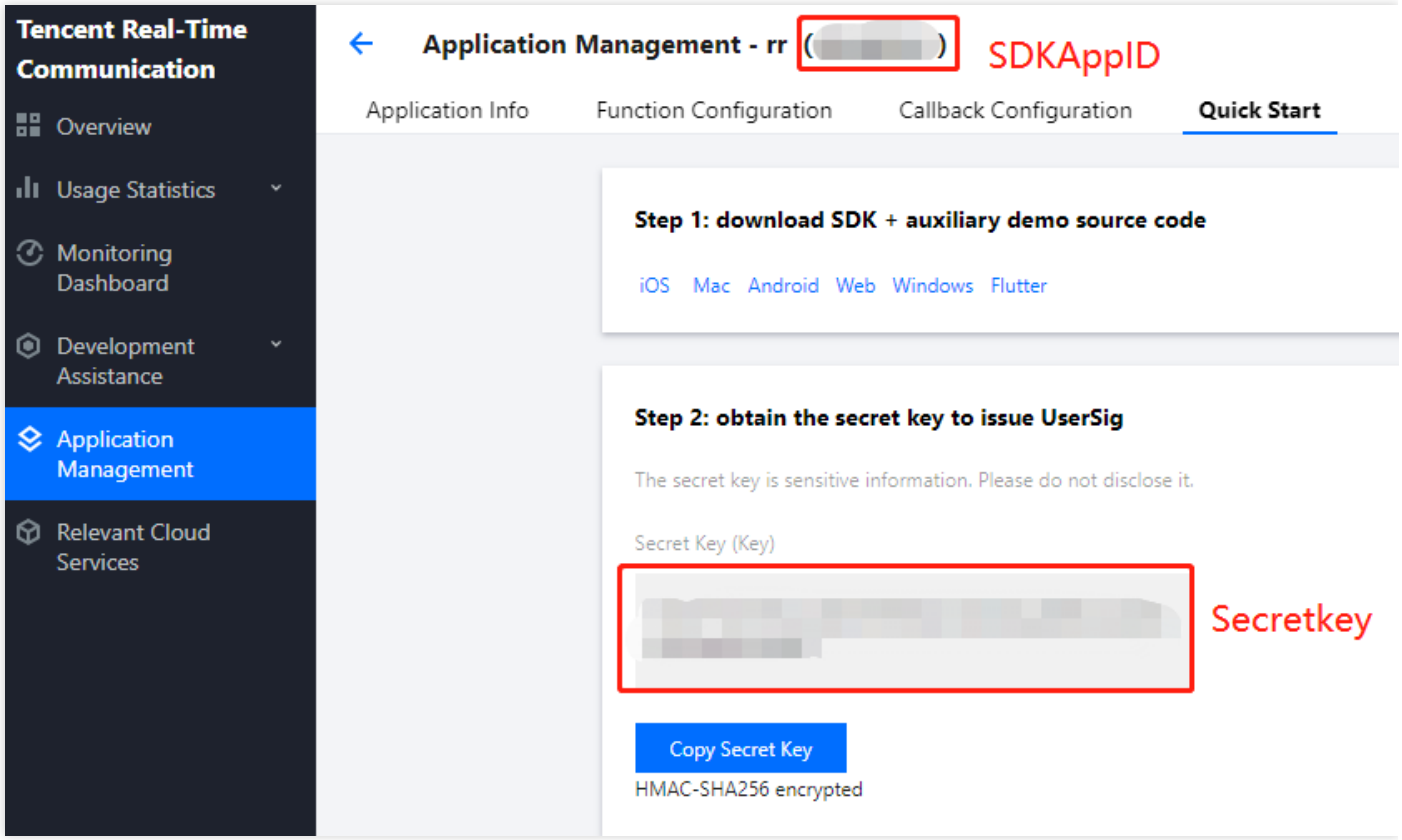
步骤三：初始化并登录

相关接口说明请参见 TUIKaraoke。

```
// 1.初始化
let karaokeRoom = TRTCKaraokeRoom.shared()
karaokeRoom.setDelegate(delegate: self)
// 2.登录
karaokeRoom.login(SDKAppID: Int32(SDKAppID), UserId: UserId, UserSig: ProfileMana
ger.shared.curUserSig()) { code, message in
if code == 0 {
//登录成功
}
}
```

参数说明：

- **SDKAppID**：**TRTC 应用ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



- **Secretkey**：**TRTC 应用密钥**，和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，长度不超过32字节，不支持使用特殊字符，建议使用英文或数字，可结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey 等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的 UserSig，更多信息见 如何计算及使用 UserSig。

步骤四：实现在线KTV场景

1. **主播创建房间 TUIKaraoke.createRoom**

```
int roomId = "房间ID";
let param = RoomParam.init()
param.roomName = "房间名称";
param.needRequest = false; // 上麦是否需要房主确认
param.seatCount = 8; // 房间座位数，一共8个座位
param.coverUrl = "房间封面图URL";
karaokeRoom.createRoom(roomID: Int32(roomInfo.roomID), roomParam: param) { [weak self] (code, message) in
guard let `self` = self else { return }
if code == 0 {
//创建成功
```

```
}
}
```

2. 听众进入房间 TUIKaraoke.enterRoom

```
karaokeRoom.enterRoom(roomID: roomInfo.roomID) { [weak self] (code, message) in
guard let `self` = self else { return }
if code == 0 {
//进房成功
}
}
```

3. 听众主动上麦 TUIKaraoke.enterSeat

```
// 1.听众调用上麦
int seatIndex = 1;
karaokeRoom.enterSeat(seatIndex: seatIndex) { [weak self] (code, message) in
guard let `self` = self else { return }
if code == 0 {
//上麦成功
}
}
// 2.收到 onSeatListChange 回调，刷新麦位列表
func onSeatListChange(seatInfoList: [SeatInfo]) {
}
```

说明：
其他关于麦位管理的相关操作，您可参考 TUIKaraoke接口文档 按需实现，或者可以参考 TUIKaraoke 示例工程。

4. 实现音乐播放并体验 KTV 场景

您可以根据自己的业务获取音乐 ID 和 URL 链接，播放歌曲。详情请参见 TUIKaraoke 音乐播放接口。

```
//播放音乐
karaokeRoom.startPlayMusic(musicID: musicID, originalUrl: muscicLocalPath, acco
mpanyUrl: accompanyLocalPath);
//停止音乐
karaokeRoom.stopPlayMusic();
```

版权所有：腾讯云计算（北京）有限责任公司

完成以上步骤，就可以实现 KTV 基本功能。如果您的业务还需要聊天、发送礼物等功能，可以接入以下能力。

步骤五：文字聊天功能（可选）

如果您需要实现各主播或听众之间文字聊天的功能，可以通过以下方法发送或接收聊天信息。
相关接口说明请参见 TRTCKaraokeRoom.sendRoomTextMsg。

```
// 发送端：发送文本消息
karaokeRoom.sendRoomTextMsg(message: message) { [weak self] (code, message) in
if code == 0 {
//发送成功
}
}
// 接收端：监听文本消息
karaokeRoom.setDelegate(delegate: self)
func onRecvRoomTextMsg(message: String, userInfo: UserInfo) {
debugPrint("收到来自：" + userInfo.userName + "的消息：" + message)
}
```

步骤六：礼物发送功能（可选）

如果您需要实现礼物发送和接收功能，可以通过以下方法发送礼物或接收礼物并展示。

```
// 发送端：通过自定义 "IMCMD_GIFT" 来区分礼物消息
karaokeRoom.sendRoomCustomMsg(cmd: kSendGiftCmd, message: message) { code, msg in
if (code == 0) {
//发送成功
}
}
// 接收端：监听礼物消息
karaokeRoom.setDelegate(delegate: self)
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo: UserInfo) {
if cmd == kSendGiftCmd {
debugPrint("收到来自：" + userInfo.userName + "的礼物：" + message)
}
}
```

常见问题

TUIKaraoke 组件支持变声、变调、混响等音效功能吗？

支持，具体请参见 TUIKaraoke 示例工程。

？如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2022-11-10 11:55:52

组件介绍

TUIKaraoke 是一个开源的音视频 UI 组件，通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。TUIKaraoke 同时支持 iOS 平台的源代码，基本功能如下图所示：



组件集成

步骤一：下载并导入 TUIKaraoke 组件

单击进入 Github ，选择克隆/下载代码，然后拷贝 Android目录下的 Source 和 Debug 目录到您的工程中，并完成如下导入动作：

- 在 `setting.gradle` 中完成导入，参考如下：

```
include ':Source'
include ':Debug'
```

- 在 app 的 build.gradle 文件中添加对 TUIKaraoke 的依赖：

```
api project(':Source')
```

- 在根目录的 `build.gradle` 文件中添加 `TRTC SDK` 和 `IM SDK` 的依赖：

```
ext {
liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTCl:latest.release"
imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

步骤二：配置权限及混淆规则

在 AndroidManifest.xml 中配置 App 的权限，SDK 需要以下权限（6.0以上的 Android 系统需要动态申请麦克风、读取存储权限等）：

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" /> // 使用场景：悬浮窗功能需要此权限；
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" /> // 使用场景：使用蓝牙耳机时需要此权限；
```

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

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

步骤三：初始化并登录

相关接口详情请参见 TUIKaraoke。

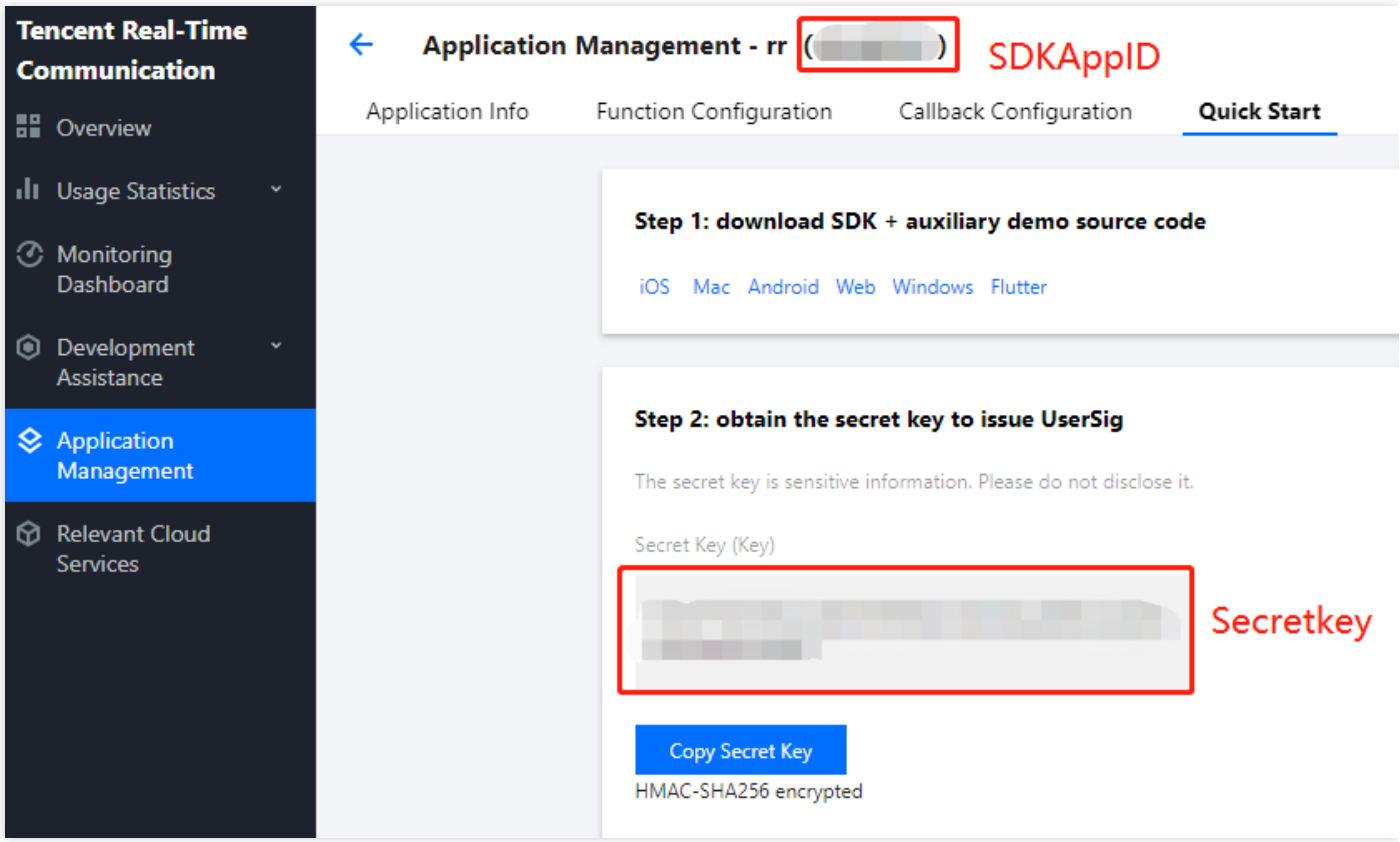
```
// 1.初始化
TRTCKaraokeRoom mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 2.登录
mTRTCKaraokeRoom.login(SDKAppID, UserID, UserSig, new TRTCKaraokeRoomCallback.Act
ionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//登录成功
}
}
});
```

参数说明：

- **SDKAppID**：**TRTC 应用ID**，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**，SDKAppID 信息如下图所示：



- **Secretkey**：**TRTC 应用密钥**，和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，长度不超过32字节，不支持使用特殊字符，建议使用英文或数字，可结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey 等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的 UserSig，更多信息见 如何计算及使用 UserSig。

步骤四：实现在线KTV场景

1. 主播创建房间 TUIKaraoke.createRoom

```
int roomId = "房间ID";
TRTCKaraokeRoomDef.RoomParam roomParam = new TRTCKaraokeRoomDef.RoomParam();
roomParam.roomName = "房间名称";
roomParam.needRequest = false; // 上麦是否需要房主确认
roomParam.seatCount = 8; // 房间座位数，一共8个座位
roomParam.coverUrl = "房间封面图URL";
mTRTCKaraokeRoom.createRoom(roomId, roomParam, new TRTCKaraokeRoomCallback.Acti
onCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//创建成功
}
}
});
```

2. 听众进入房间 TUIKaraoke.enterRoom

```
mTRTCKaraokeRoom.enterRoom(roomId, new TRTCKaraokeRoomCallback.ActionCallback()
{
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//进房成功
}
}
});
```

3. 听众主动上麦 TUIKaraoke.enterSeat

```
// 1.听众调用上麦
int seatIndex = 1;
mTRTCKaraokeRoom.enterSeat(seatIndex, new TRTCKaraokeRoomCallback.ActionCallbac
k() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//上麦成功
}
}
});
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 2.收到 onSeatListChange 回调，刷新麦位列表
@Override
public void onSeatListChange(final List<TRTCKaraokeRoomDef.SeatInfo> seatInfoLi
st) {
}
```

说明：
其他关于麦位管理的相关操作，您可参考 TUIKaraoke接口文档 按需实现，或者可以参考我们的 TUIKaraoke 示例工程。

4. **实现音乐播放并体验 KTV 场景**

您可以根据自己的业务获取音乐 ID 和 URL链接播放歌曲，接口详情请参见 TUIKaraoke 音乐播放接口。

```
//播放音乐
mTRTCKaraokeRoom.startPlayMusic(musicID,url);
//停止音乐
mTRTCKaraokeRoom.stopPlayMusic();
```

完成以上步骤，您可以实现KTV基本功能。如果您的业务还需要文字聊天、发送礼物等功能，可以接入以下能力。

步骤五：文字聊天功能（可选）

如果您需要实现各主播或听众之间文字聊天的功能，可以通过以下方法发送或接收聊天信息。
相关接口详情请参见 TRTCKaraokeRoom.sendRoomTextMsg。

```
// 发送端：发送文本消息
mTRTCKaraokeRoom.sendRoomTextMsg("Hello Word!", new TRTCKaraokeRoomCallback.Actio
nCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//发送成功
}
}
});
// 接收端：监听文本消息
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
@Override
public void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInf
o) {
Log.d(TAG, "收到来自" + userInfo.userName + "的消息:" + message);
```

```
}
});
```

步骤六：礼物发送功能（可选）

如果您需要实现礼物发送和接收功能，可以通过以下方法发送或接收礼物并展示。

```
// 发送端：通过自定义 "CMD_GIFT" 来区分礼物消息
mTRTCKaraokeRoom.sendRoomCustomMsg("CMD_GIFT",date, new TRTCKaraokeRoomCallback.A
ctionCallback() {
@Override
public void onCallback(int code, String msg) {
if (code == 0) {
//发送成功
}
}
});
// 接收端：监听礼物消息
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
@Override
public void onRecvRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomDef.Us
erInfo userInfo) {
if ("CMD_GIFT".equals(cmd)) {
// 收到礼物消息
Log.d(TAG, "收到来自" + userInfo.userName + "的礼物:" + message);
}
}
});
```

常见问题

TUIKaraoke 组件支持变声、变调、混响等音效功能吗？

支持。

> 说明：
> 如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

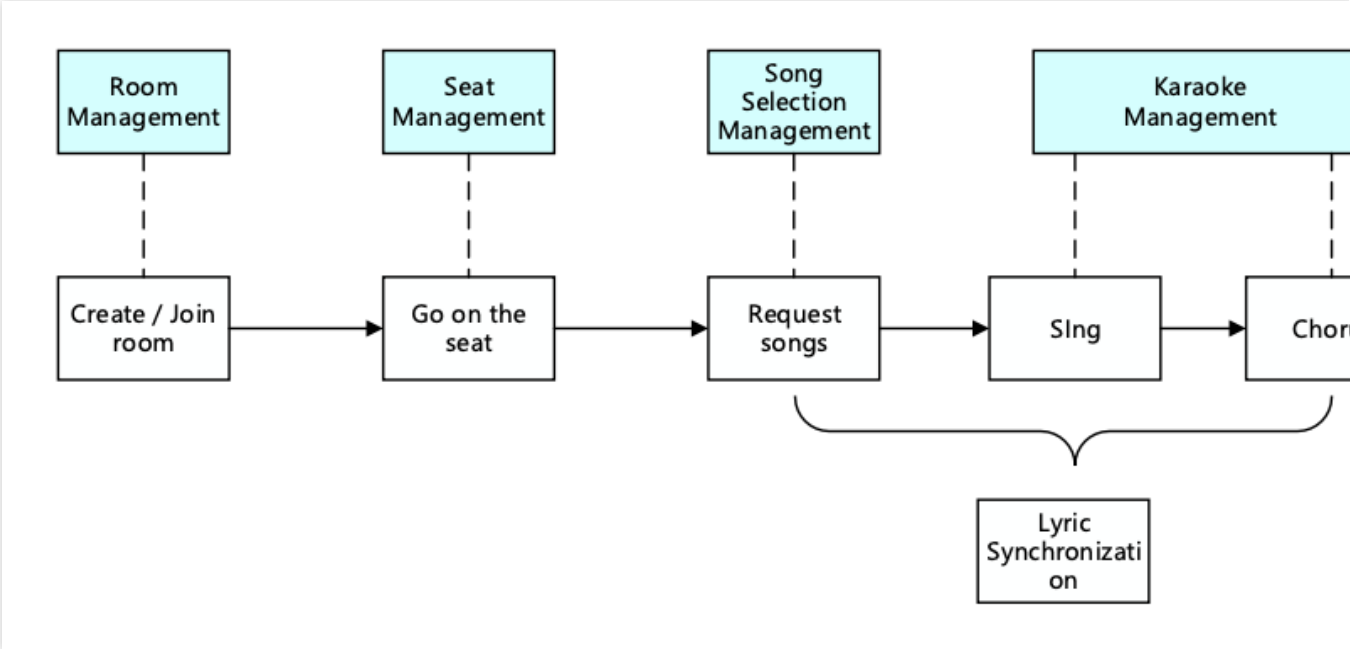
方案介绍（TUIKaraoke）
实现步骤

最近更新时间：2023-09-26 16:42:08

通常实现一个完整的在线 K 歌场景，需要涉及多个功能模块：房间管理、麦位管理、点歌管理、K 歌管理等。每个功能模块下的关键动作及功能点如下表所示。接下来会逐个介绍各个功能模块，通过介绍可以对搭建K歌房所需的功能有个完整的认知。

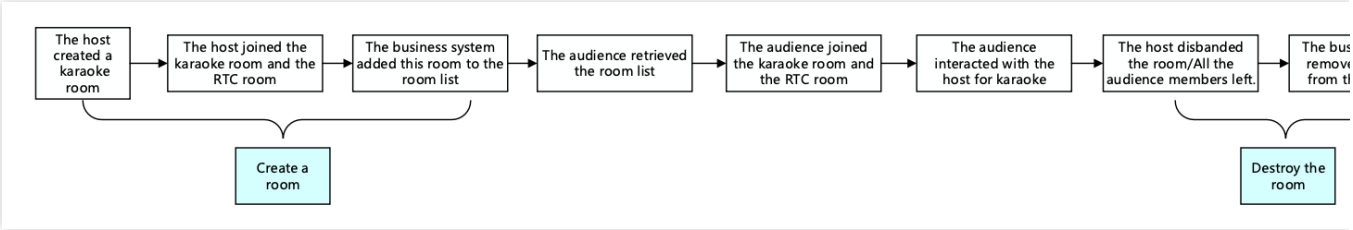
房间管理	麦位管理	点歌管理	K歌管理
房间列表	上/下麦	歌单展示	唱歌玩法
创建房间	麦位控制	搜索歌曲	切歌
加入房间	锁麦位	点歌	人声音量调节
退出房间	邀请上麦	歌曲置顶	混响/声效
销毁房间	麦位禁言	已点列表	歌词同步

房主创建 K 歌房，用户可以选择感兴趣的歌房加入。进入房间后用户可以上麦参与互动，上麦后可以和房主语音互动。当然，用户也可以选择直接上麦参与合唱，这是两种不同的 K 歌玩法。在线 K 歌场景整体的业务流程如下图所示。



房间管理

房间管理是主要负责对房间列表的维护。主要包含的功能：创建房间、加入房间、销毁房间、退出房间。而且 K 歌房间有区别于普通房间，需要有单独的 K 歌房间标识符，以启动相关的组件管理：点歌管理、K 歌管理等功能。
创建房间：用户登录业务系统后，可以创建房间，创建房间后房间列表要做新增操作。
销毁房间：所有用户退出房间后，需要销毁房间，销毁房间后房间列表要做删除操作。



说明：
房间管理是实现在线K歌的必要模块，但并非主要功能模块，具体实现可以结合业务系统和 TRTC SDK，详见语聊房场景接入方案。

麦位管理

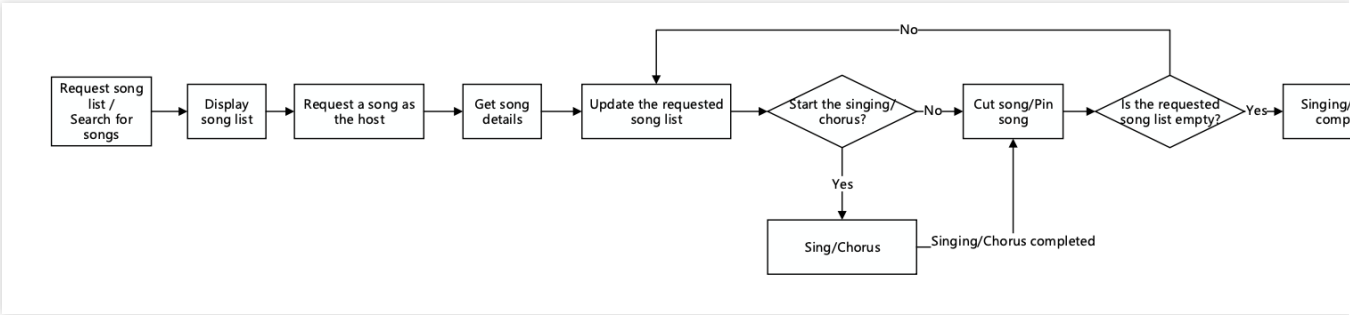
歌房内的麦位一般都是有序且有限的。麦位管理主要负责根据业务场景定义房间内的麦位数量，以及当前房间所有麦位的状态管理。麦位管理主要包含的功能：上/下麦、锁麦位、邀请上麦、麦位禁言等。
用户进入房间后，只有空闲状态的麦位才可以申请上麦。
房主同意用户上麦后，需要修改麦位状态为非空闲状态。
用户停止推流下麦后，需要重置麦位状态。
房主有权锁定麦位、邀请上麦、强制下麦、麦上禁言等。
说明：
麦位管理是实现在线K歌的必要模块，但并非主要功能模块，具体实现可以结合业务系统和 IM SDK，详见语聊房场景接入方案。

点歌管理

基本介绍

点歌管理属于在线 K 歌场景比较重要的一环，主要包含的功能：歌单展示、搜索歌曲、点歌和排麦、已点列表。而且每个 K 歌房间要有一个维护已点歌曲列表和自动排麦功能，都是需要业务后台来实现，而歌单展示、搜索歌曲需要结合音速达直播音乐版权引擎（Yinsuda Authorized Music for Live Streaming）实现。

实现流程



整个点歌管理中，主要涉及业务端 App、业务后台、音速达后台，其中各自的职能分别为：

业务端 App：

调用点歌接口上报歌曲信息.

调用切歌接口通知业务后台更新已点歌单列表.

调用演唱确认接口通知业务后台。

业务后台：

维护已点歌单列表。

下发通知告诉业务端 App 更新当前已点歌曲列表。

音速达后台：

提供获取直播互动歌曲推荐 歌单列表/歌单详情 的接口。

提供 获取直播互动曲目详情 的接口（播放playToken、歌词下载URL）。

提供 搜索直播互动歌曲 的接口。

K 歌管理

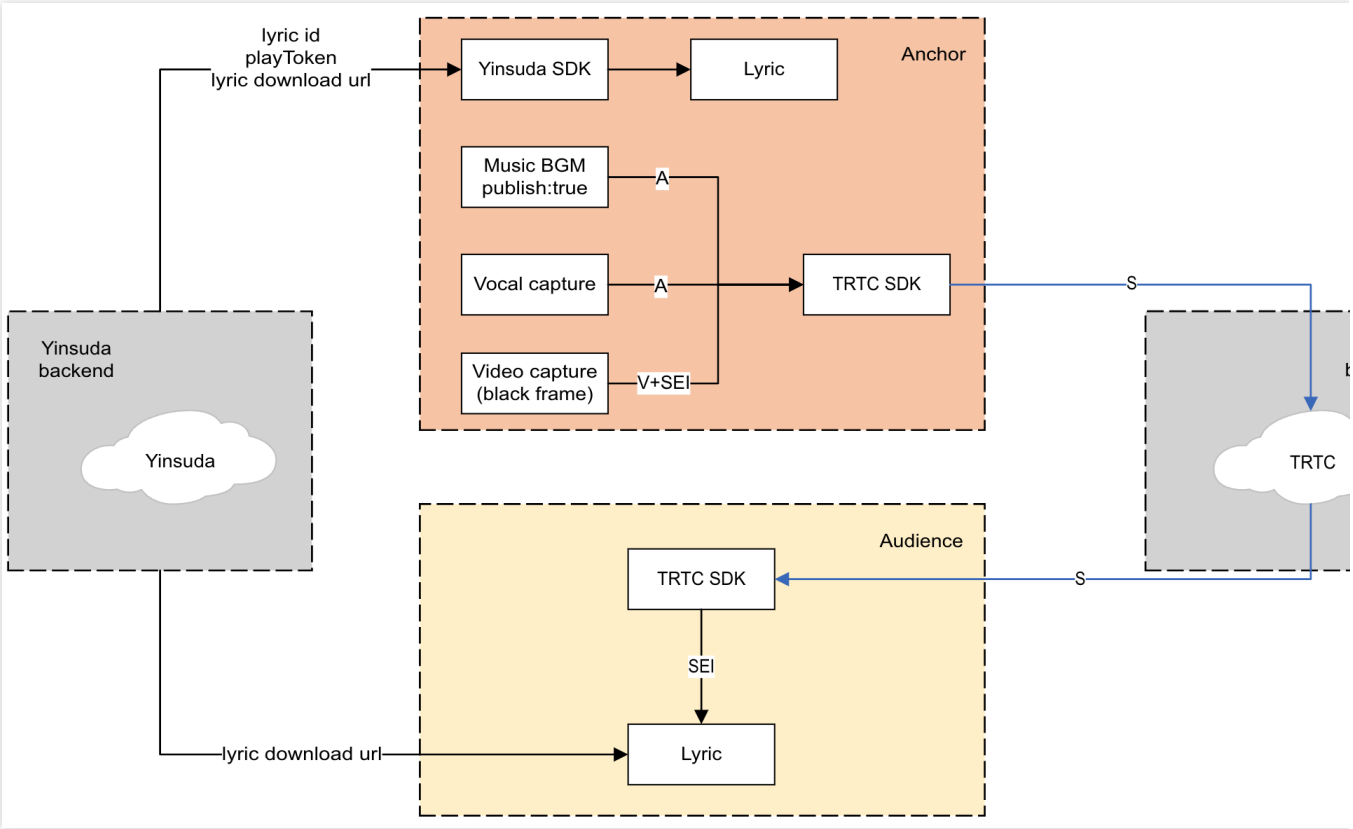
K 歌系统主要包含功能：唱歌玩法、开始/停止/切歌、人声音量调节、混响/声效、歌词同步。下面我们将通过演唱和实时合唱两种典型的 K 歌玩法来详细介绍 K 歌管理模块的实现流程。

演唱

主要是多人互动的KTV场景下，主播上麦后，才可以进行点歌，主播点歌成功后，会统一在点歌台展示，主播选择开始演唱。

（1）方案架构

在整体方案上，主要是用到的音速达SDK，实现歌曲下载，音速达后台，获取歌曲 playToken 和歌词下载地址；用到的 TRTC SDK，来实现演唱者人声的推流、歌曲的播放以及推流。整体的方案架构如下：



（2）具体实现

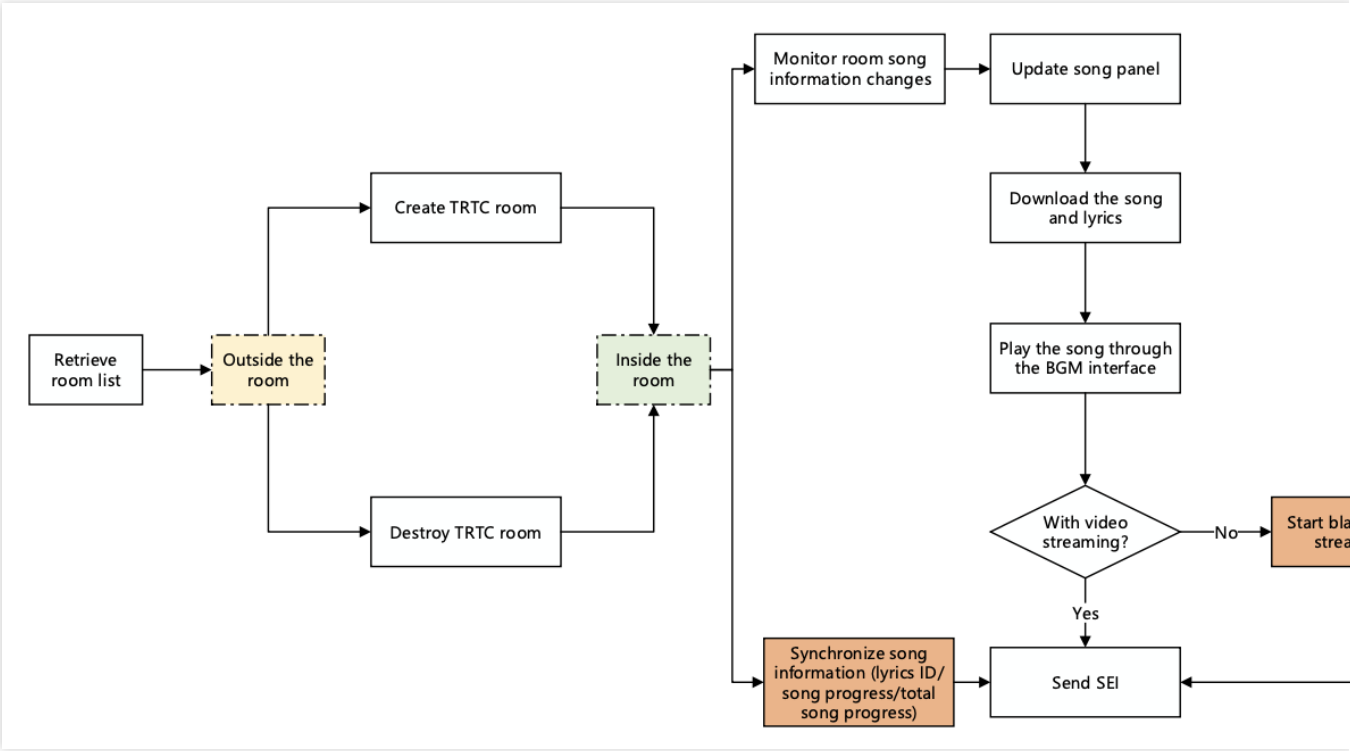
在演唱的场景，会区分不同角色有不同的实现流程，分为演唱者、观众 2种角色。

角色	描述	区别
演唱者	KTV 房间的演唱者，是房间的房主点歌以后演变而来的，房主创建房间后自动上麦点歌并演唱，退房后房间自动解散，已点歌曲自动清空	角色必须为主播 上行音视频（无视频上行黑帧） 播放 BGM 发送 SEI 信息（发送歌词信息） 点歌
观众	KTV 房间的观众，播放演唱者的流	角色为观众，亦可上麦成为主播 下行音视频流 接收 SEI 信息（接收歌词信息）

不同角色的基本实现流程如下：

房主

观众



房主创建并加入 TRTC 的房间，自动上麦后点歌成为演唱者。

点歌后进行歌曲/歌词的下载，然后通过播放 BGM 接口播放歌曲。

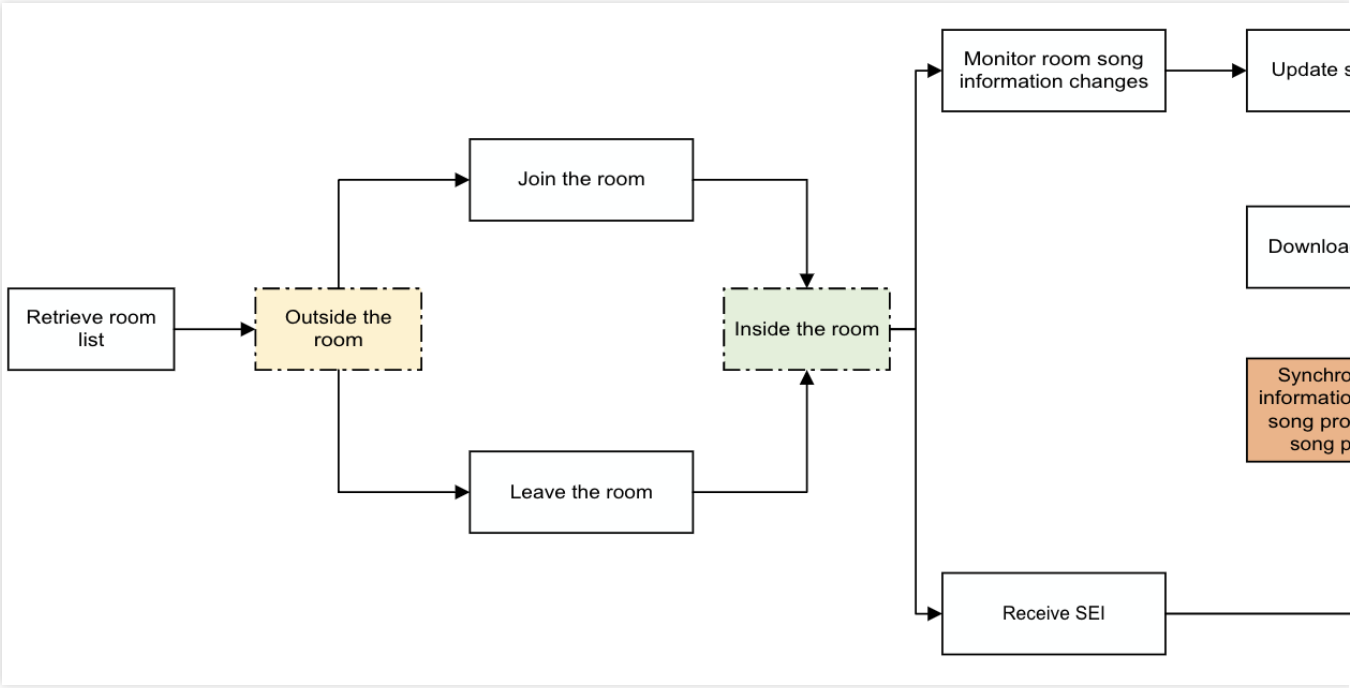
演唱者没有带视频上行的话，需要开启视频上行。

通过 SEI 信息同步所有人的歌词进度。

演唱者在唱歌过程中可以随时切歌，并重新开始对歌曲/歌词进行下载，下载完成后进行演唱。

房主退房后将解散 TRTC 房间。

版权所有：腾讯云计算（北京）有限责任公司



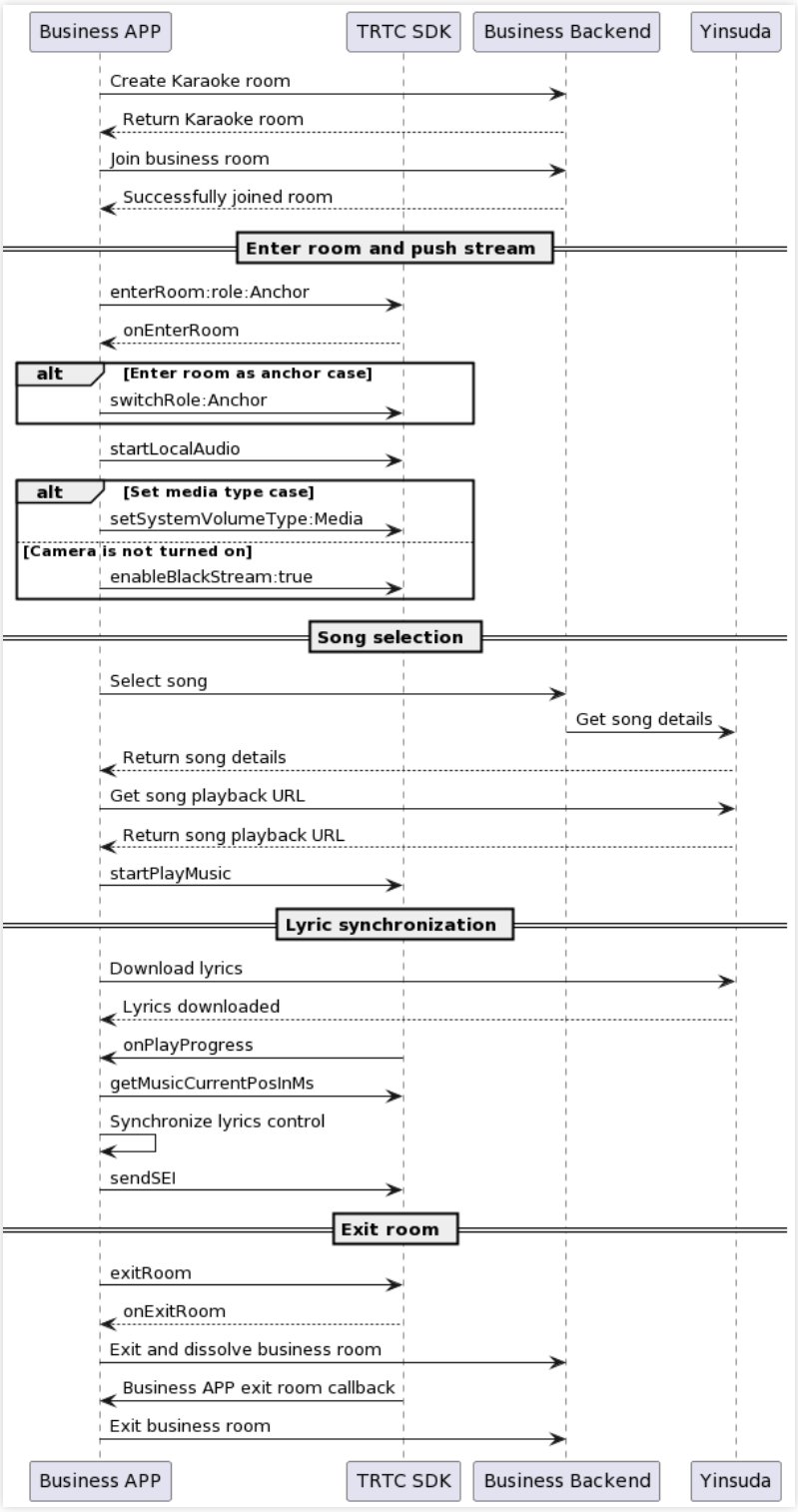
观众加入 TRTC 的房间。
监听房间歌曲变化，并加载歌词。
拉取演唱者的流。
解析演唱者发送的 SEI 信息，并同步歌词。

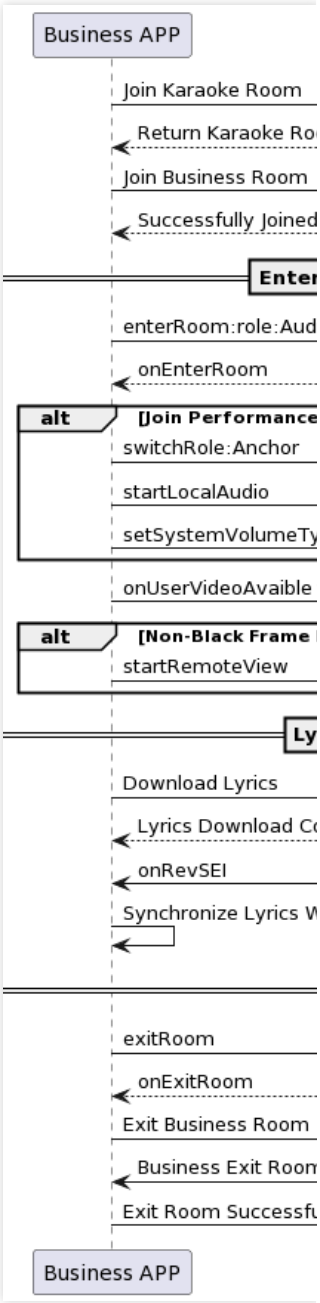
主要是要监听歌曲的 SEI 信息，更新对应的歌曲控件。

（3）API 调用时序

不同角色的 API 时序调用如下：

房主	观众





说明：

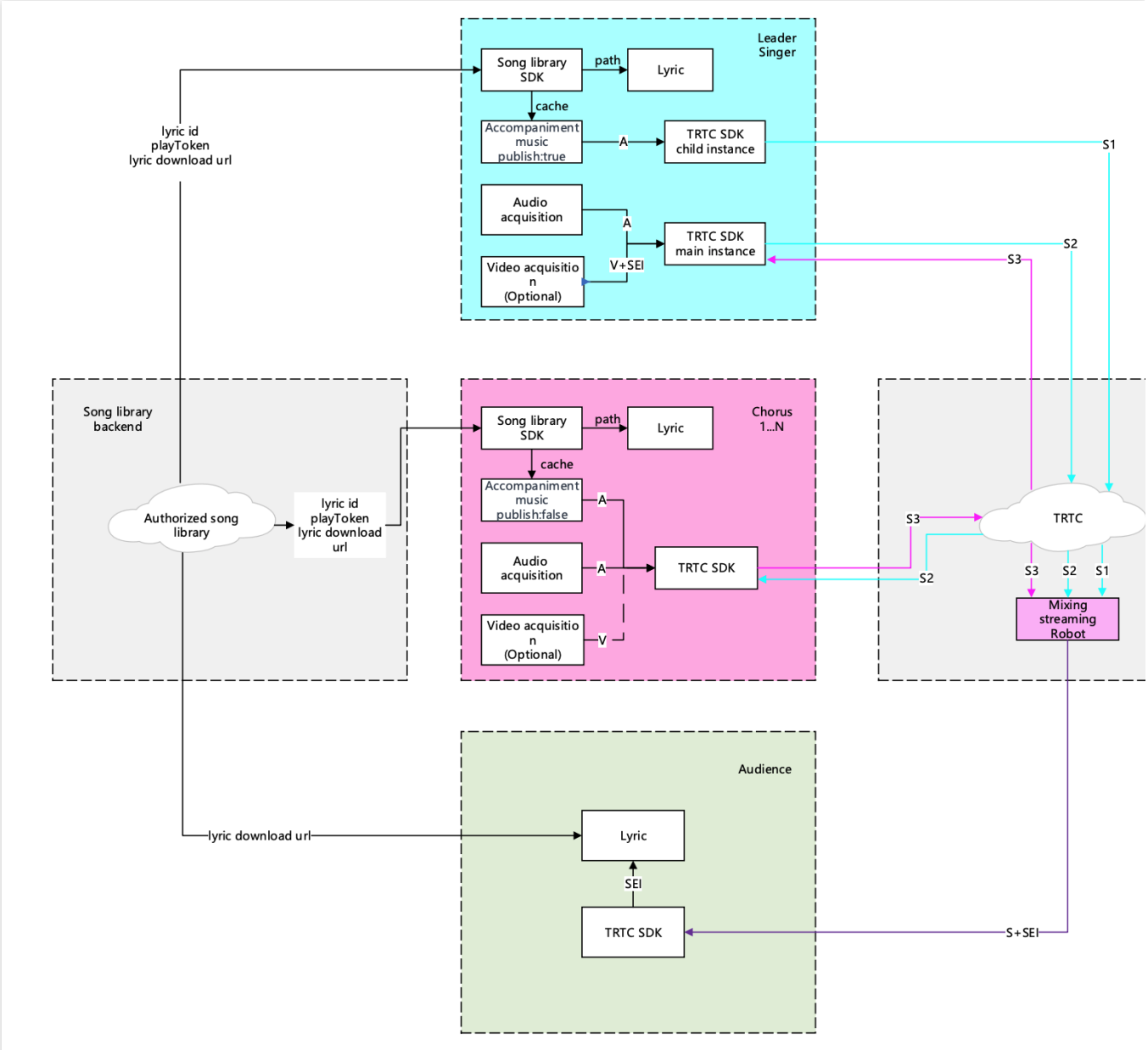
鉴于以上实现方案需要一定的技术门槛，而 TRTC 官网提供的TUIKaraoke 开源的音视频 UI 组件，可以通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。

实时合唱

实时合唱是指各端在连麦的基础上，同时播放歌曲，然后上麦进行合唱。多人模式下合唱者之间都能听到彼此声音，几乎感受不到延迟，达到了真正意义上的实时合唱。

（1）方案架构

在媒体流方面，合唱者互相进行推拉流，同时会由一名***主唱者推出音乐***，其他***合唱者在本地播放音乐***，经过 NTP 进行时间同步。另外，歌曲和所有合唱者的声音都通过混流机器人进行混流处理形成一条流，并回推到 TRTC 房间，观众只需拉一条流即可听到各端同步的声音，完美实现多人合唱的效果。实时合唱方案架构如下图所示。



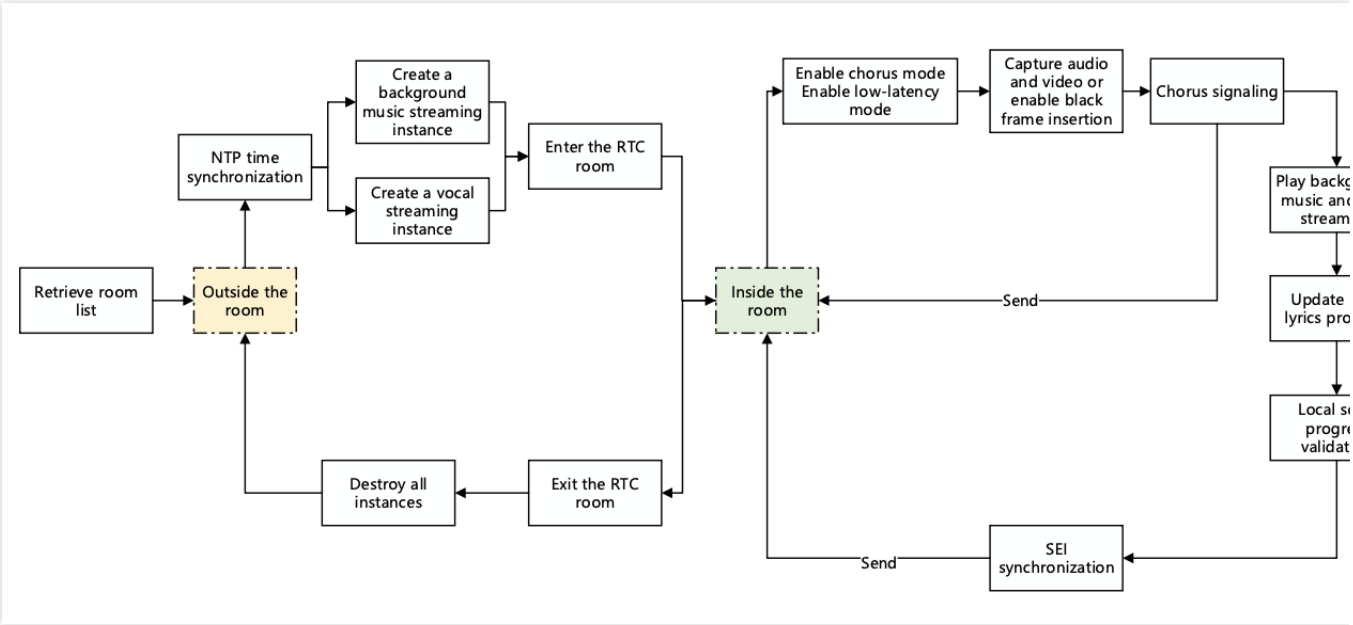
该方案的优点在于：
降低了端到端的时延。
提供了用户中途加入合唱的解决方案。
精准同步不同端之间的音乐、歌词、人声。
改善各端设备性能和本地时间不精准的情况，降低网络环境造成的时延影响。
说明：
根据业务需要，可以选择纯音频场景或音视频场景的实时合唱方案；若为纯音频场景则需要补黑帧以发送 SEI 消息用于歌词同步；
主唱需要使用子实例同时上行音乐及人声；其他合唱者仅互相拉取人声流，同时本地播放音乐；观众只需拉取一路混流；
图中展示的是 RTC 观看方案，混流机器人将合流回推至 RTC 房间；CDN 观看方案下，混流机器人将合流转推至直播 CDN，观众拉取 CDN 流观看。

（2）具体实现

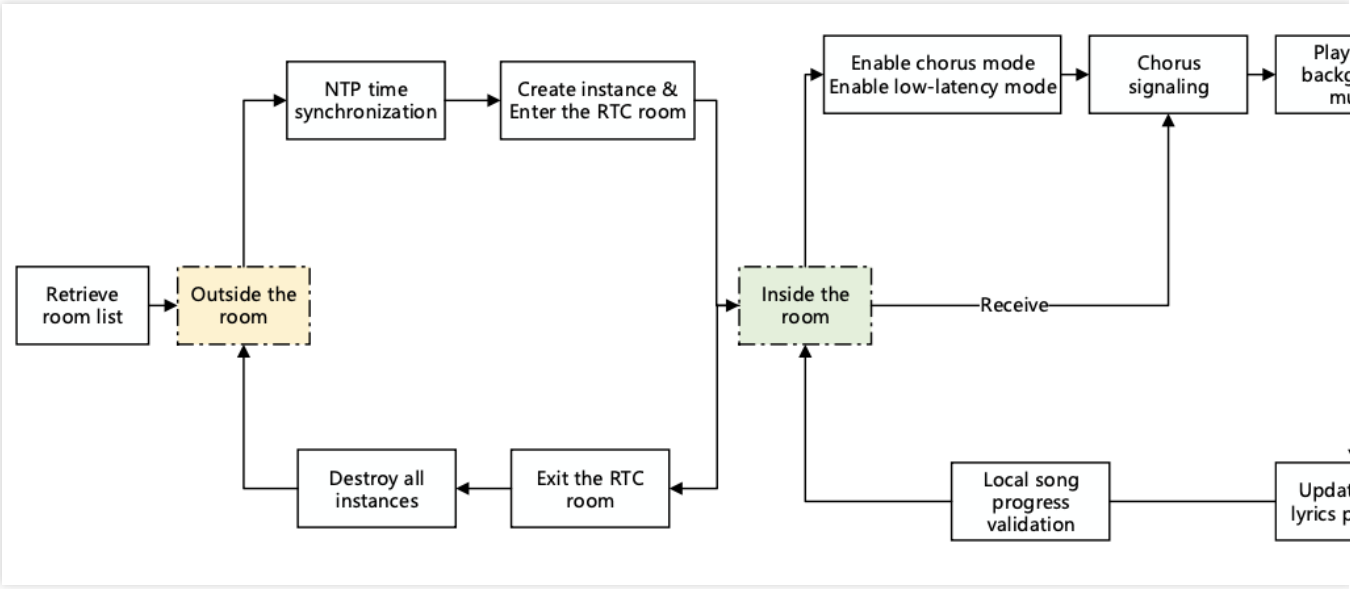
我们可以将在线 K 歌房里的用户划分为三种角色：主唱、合唱和观众，如下表所示。

角色	描述	区别
主唱	主唱负责点歌和发送合唱信令，以及 SEI 的发送	角色为 Anchor 上行音乐和人声 点歌及发起合唱 混流回推 发送 SEI 消息
合唱	合唱者可以接收并处理合唱信令，麦上参与合唱	角色为 Anchor 上行人声 本地播放音乐 接收合唱
观众	进入歌房后，在麦下拉流的观众，也可上麦合唱	角色为 Audience 下行混流 接收 SEI 消息 申请上麦成为 Anchor

不同角色的基本实现流程如下图所示：
主唱
合唱
观众

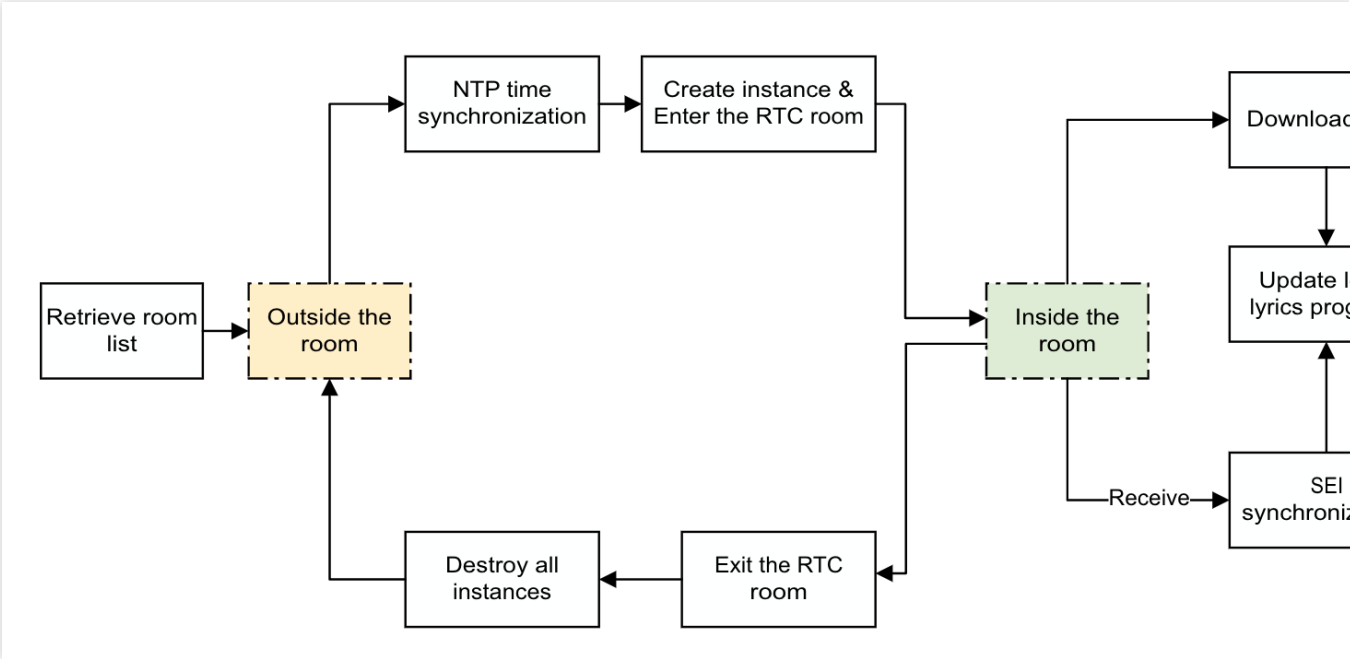


主唱需要点播歌曲，发送合唱信令。
主唱创建子实例推送人声和音乐，并拉取其他合唱者的人声。
在推流后，主唱负责发起混流转推任务。
开唱后，播放音乐，通过播放进度回调同步歌词。
需要通过 SEI 发送歌曲进度，以便观众端同步歌词。
所有演唱者需要根据 NTP 校准本地歌曲播放进度。



合唱者推送一路人声流，拉麦上合唱用户的人声流。

合唱者需要监听并接收合唱信令，预加载音乐资源。
开唱后，本地播放音乐，合唱者通过播放进度回调同步歌词。
所有演唱者需要根据 NTP 校准本地歌曲播放进度。

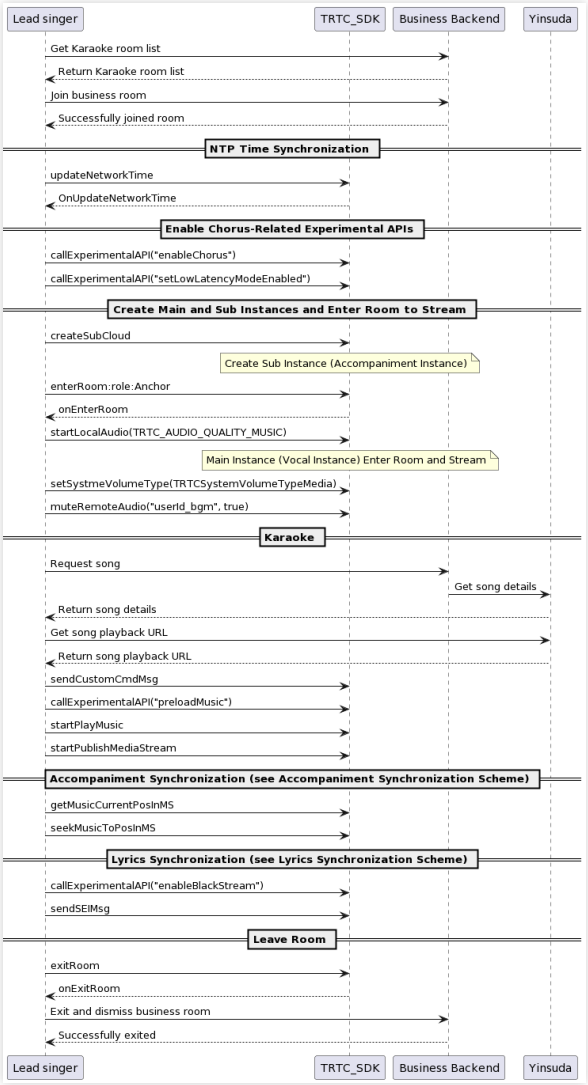


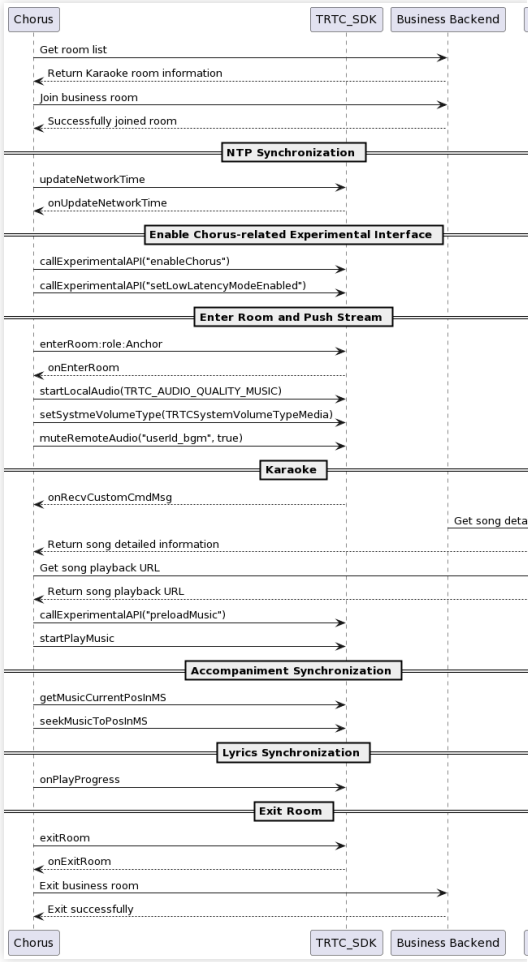
拉混流收听合唱。
解析混流中 SEI 的歌曲进度信息，用于同步歌词；
上麦后停止拉混流，转为拉麦上人声流，同时开启合唱模式。

（3）API 调用时序

不同角色的 API 调用时序如下：

主唱 API 时序	合唱 API 时序





说明：

鉴于以上实现方案需要一定的技术门槛，而 TRTC 官网提供的TUIKaraoke 开源的音视频 UI 组件，可以通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加实时合唱场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。

歌曲同步

iOS

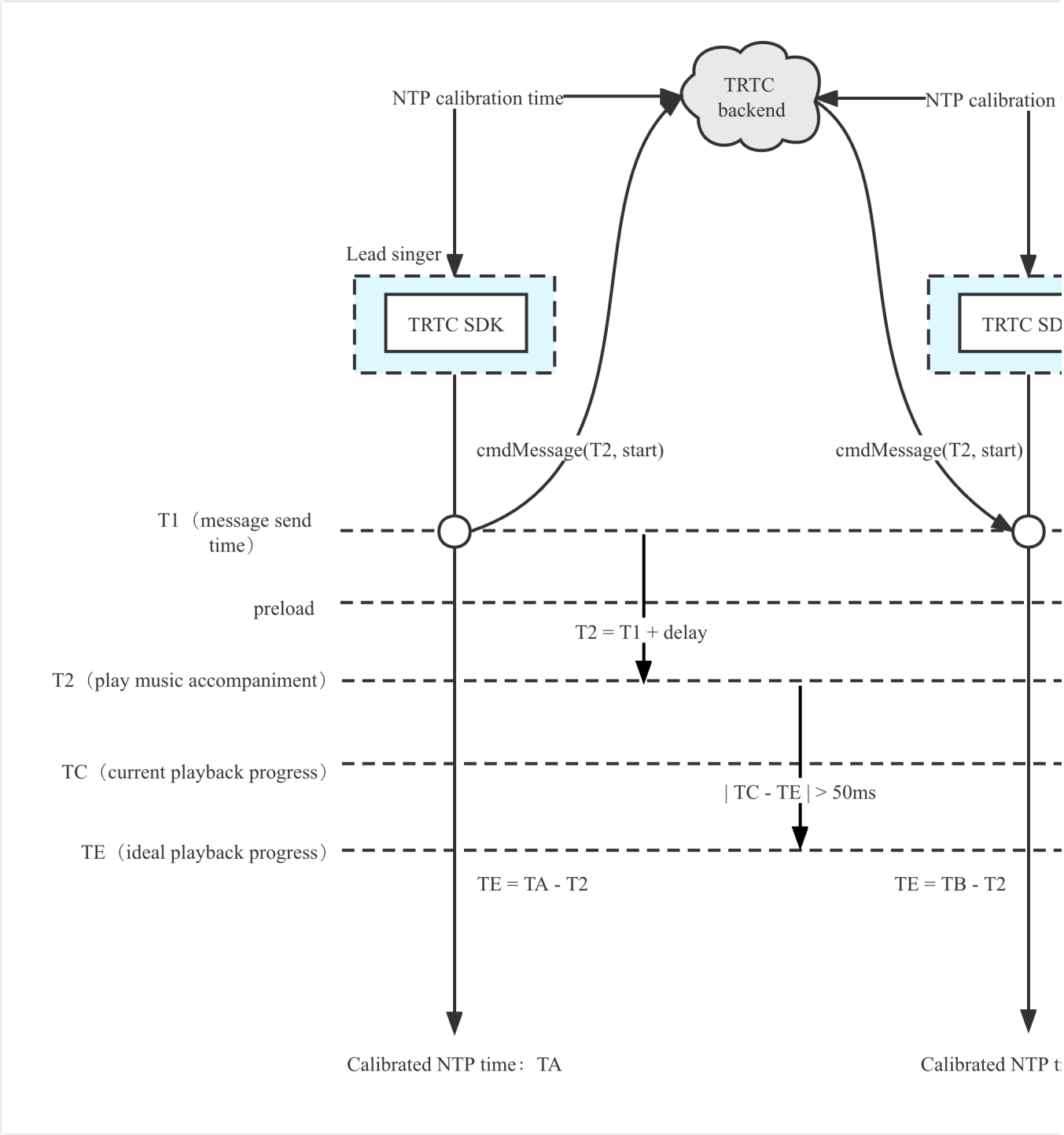
最近更新时间：2023-09-27 14:44:45

实时方案中需要在开唱后实时同步歌曲进度，避免因歌曲误差而增加端到端延迟。同步歌曲需要基于 NTP 时间，不同设备的本地时钟并不一致，存在一定误差，因此需要引入腾讯云自研 NTP 服务。同时，中途加入合唱的用户也需要同步歌曲进度，只有同步进度后，才能参与合唱。

实现流程

歌曲同步的做法为：主唱用户约定在未来某个时间点（如延迟 N 秒后）开始播放歌曲，其他用户参与合唱。各端的时间都以 NTP 时间为准，NTP 时间会在 TRTC SDK 初始化后开始同步。

版权所有：腾讯云计算（北京）有限责任公司



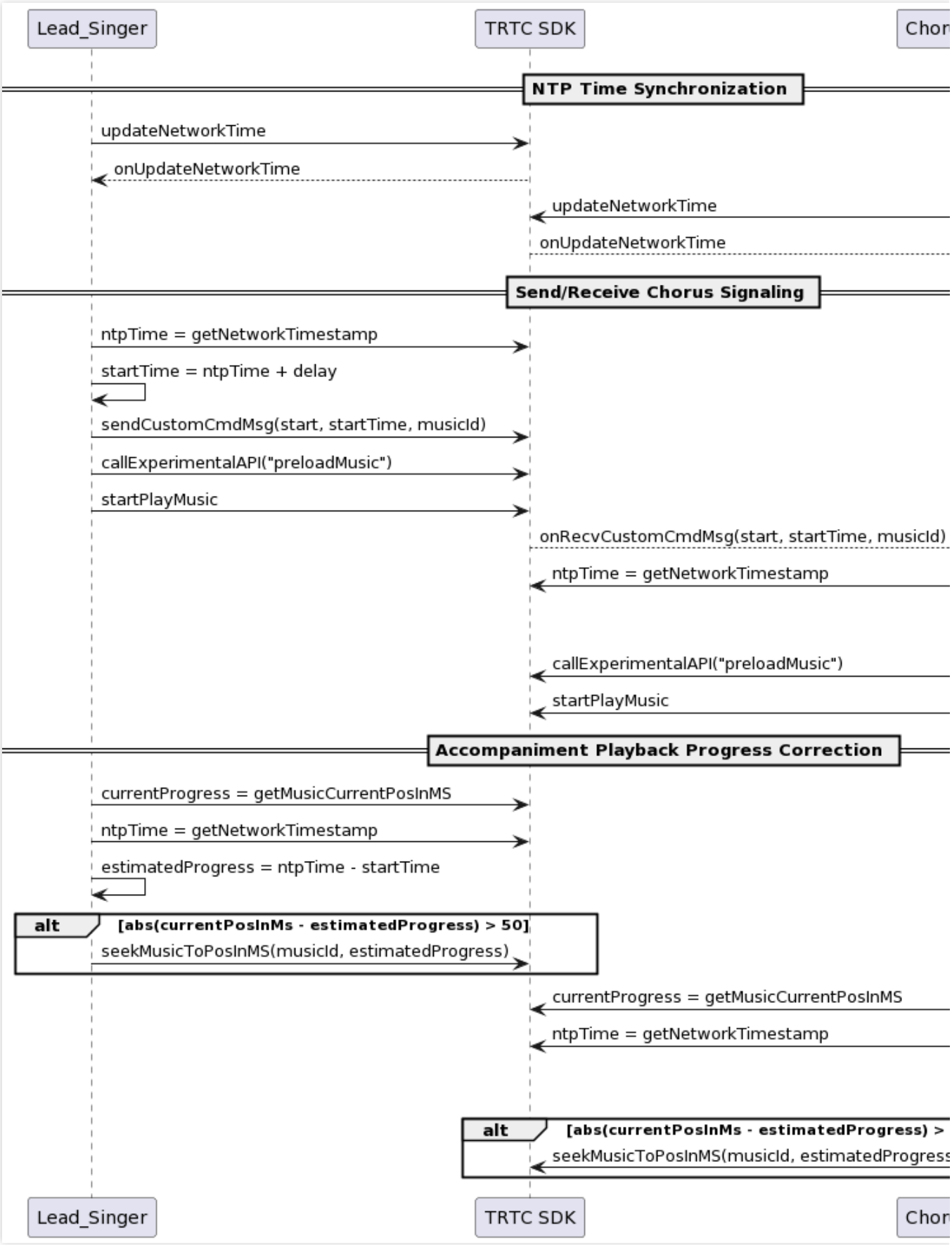
具体流程如下：

1. 各端进行 NTP 校时，向 TRTC 云端更新并获取最新的 NTP 时间 T。
2. 主唱端发送合唱信令（自定义消息），约定合唱开始时间 T2。
3. 本地根据 T2 预加载歌曲，定时播放。
4. 其他合唱用户接收到合唱信令后执行步骤 3。
5. 过程中对本地歌曲播放进度进行校验，当 TE 与 TC 差值超过 50ms 即 seek 校准。

说明：

这里的 50ms 误差是个典型值，可以根据业务容忍度适当调整，建议在 50ms 上下浮动。

时序图

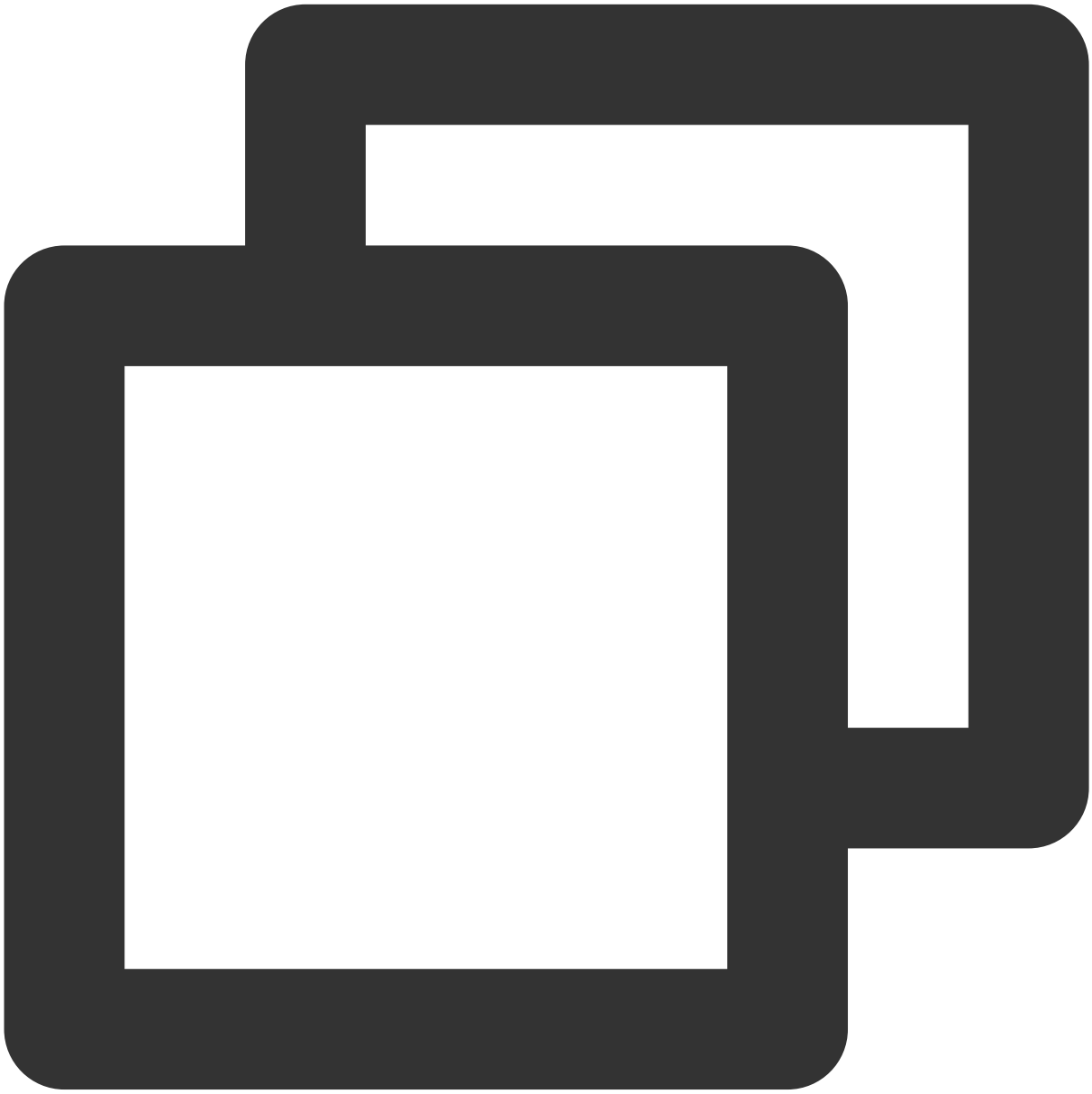


版权所有：腾讯云计算（北京）有限责任公司

歌曲同步时序主要可以分为三个部分：NTP 校时、发送及接收合唱信令、歌曲播放进度修正。下面将针对这四部分给出具体的代码实现。

关键代码实现

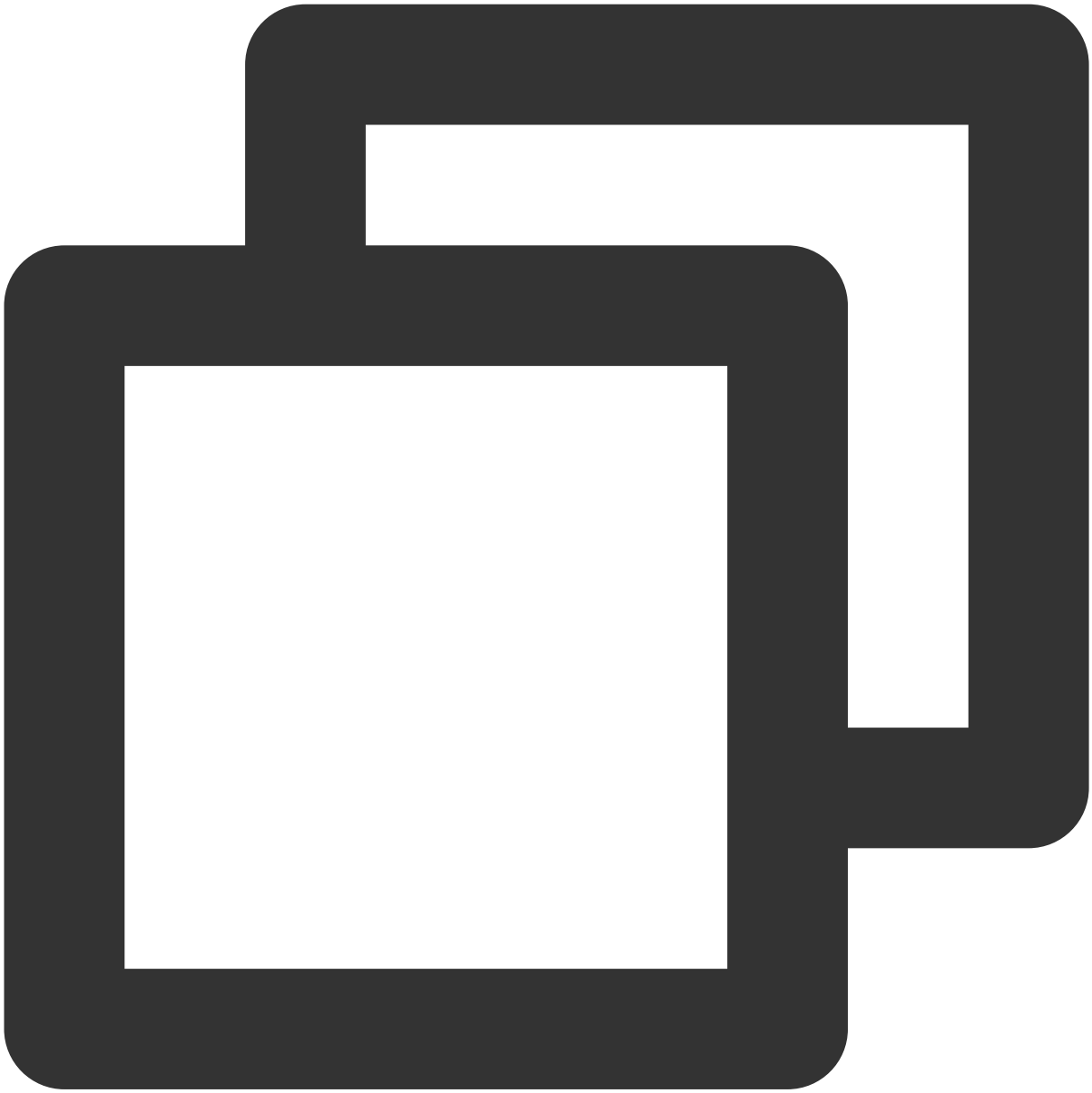
1. NTP 校时服务



```
//进房时调用NTP校时接口
[TXLiveBase updateNetworkTime];
```

```
// TXLiveBaseDelegate 回调内判断是否校时成功
- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: 校时成功且偏差在30ms以内；1: 校时成功但偏差可能在30ms以上；-1: 校时失败
    if (errCode == 0) {
        // 调用TXLivebase的getNetworkTimestamp即可获取NTP时间戳
        NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    } else {
        // 重新调用updateNetworkTime启动一次校时
        [TXLiveBase updateNetworkTime];
    }
}
```

2. 主唱端发送合唱信令

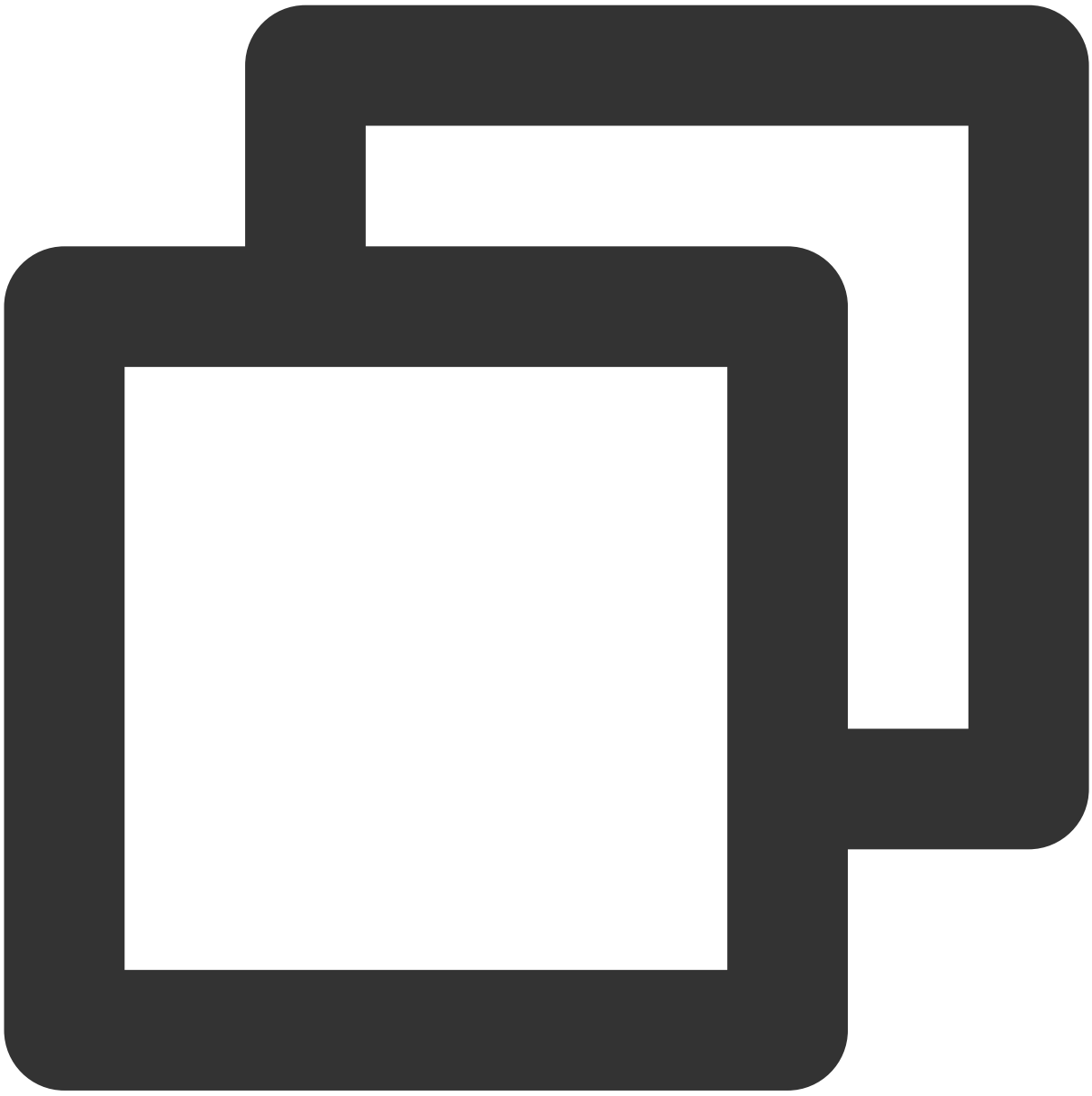


```
NSDictionary *json = @{
                        @"cmd": @"startChorus",
                        // 约定时间合唱
                        @"startPlayMusicTS": @(startTs),
                        @"musicId": @"musicId",
                        @"musicDuration": @(musicDuration),
                      };
NSString *jsonString = [self jsonStringFrom:json];
[trtcCloud sendCustomMessage:jsonString reliable:NO];
```

说明：

建议主唱以固定时间频率循环向房间内发送合唱信令消息，以便合唱者可以中途加入合唱；
不使用 SEI 消息发送合唱信令的原因： SEI 信息会插入到视频帧中，导致观众侧拉取的视频流中携带很多无效信息。

3. 合唱端接收合唱信令



```
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UIn

    NSString *msg = [[NSString alloc] initWithData:message encoding:NSUTF8StringEnc
    NSError *error;
```

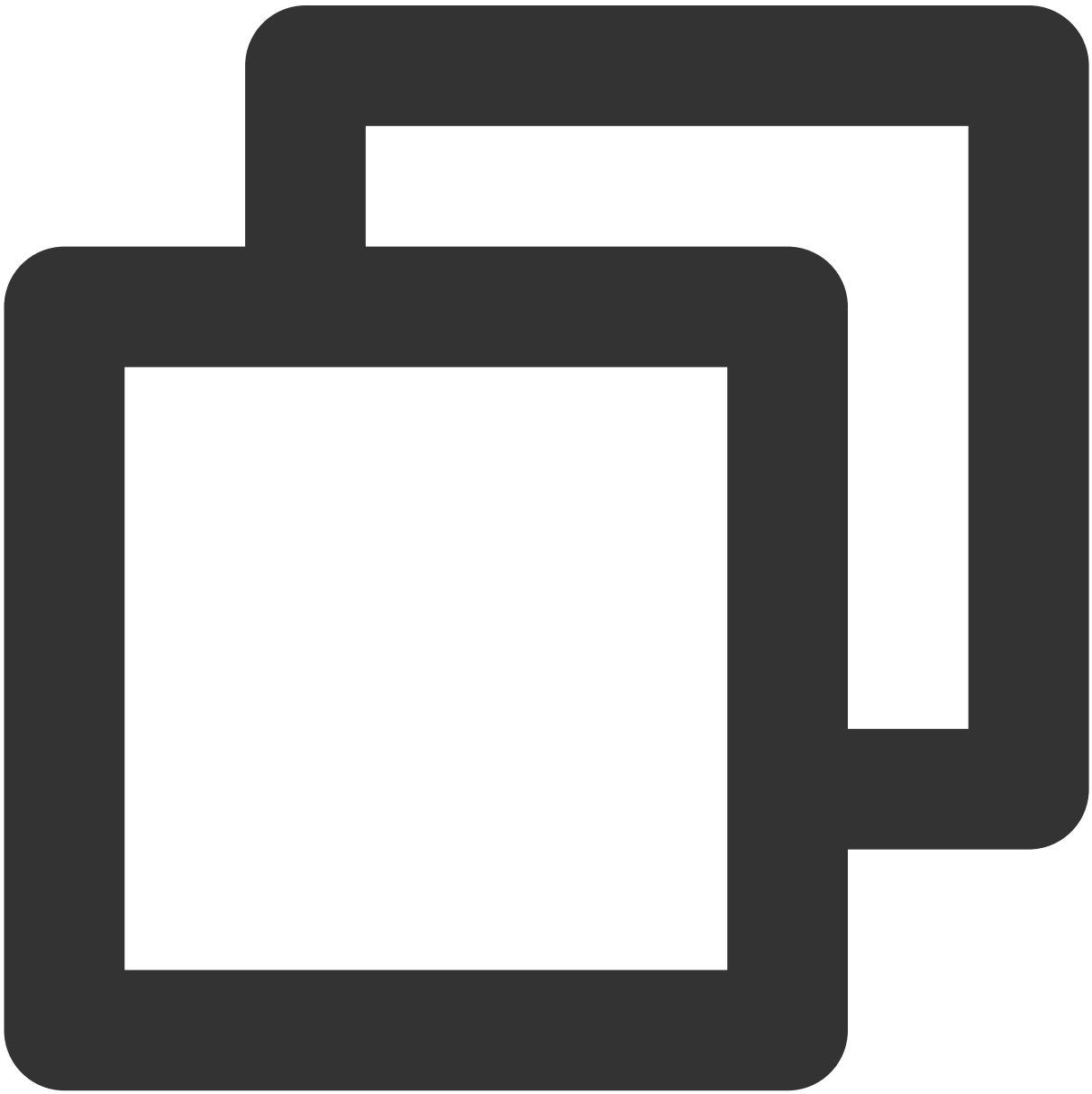
```
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[msg dataUsingEnco
                                                      options:NSJSONReadingMutab
                                                        error:&error];

NSObject *cmdObj = [json objectForKey:@"cmd"];

NSInteger musicDuration = [[json objectForKey:@"musicDuration"] integerValue];

NSString *cmd = (NSString *)cmdObj;

// 合唱命令
if ([cmd isEqualToString:@"startChorus"]) {
    // 合唱开始时间
    NSObject *startPlayMusicTsObj = [json objectForKey:@"startPlayMusicTS"];
    NSString *musicId = [json objectForKey:@"musicId"];

    NSInteger startPlayMusicTs = ((NSNumber *)startPlayMusicTsObj).longLongValu
    // 合唱约定时间和当前时间差值
    NSInteger startDelayMS = labs(startPlayMusicTs - [TXLiveBase getNetworkTime
    // 开启预加载，根据合唱约定时间和当前ntp差值，跳跃歌曲进度
    NSDictionary *jsonDict = @{
        @"api": @"preloadMusic",
        @"params": @{
                @"musicId": @(musicId),
                @"path": path,
                @"startTimeMS": @(startDelayMS),
        }
    };
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSU
    [subCloud callExperimentalAPI:jsonString];

    // 播放歌曲
    TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
    param.ID = musicId;
    param.path = url;
    param.loopCount = 0;
    param.publish = NO;

    [[subCloud getAudioEffectManager] startPlayMusic:param onStart:^(NSInteger
        // 开始播放回调
    } onProgress:^(NSInteger progressMs, NSInteger durationMs) {
        // 歌曲进度回调
    } onComplete:^(NSInteger errCode) {
        // 播放完成回调
    }];
```

```
    }
}
```

说明：

合唱端接收到第一个 startChorus 信令后，状态应从“未合唱”转为“合唱中”，此后不再响应 startChorus 信令，避免重复启动 BGM 播放。

4. 歌曲播放进度修正



```
self.startPlayChorusMusicTs; //最初约定的合唱时间
// 当前播放进度
```

```
NSInteger currentProgress = [[self audioEffecManager] getMusicCurrentPosInMS:self.c
// 当前歌曲的理想播放时间进度
NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] - self.startPlayChor
if (estimatedProgress >= 0 && labs(currentProgress - estimatedProgress) > 50) {
    // 当播放进度超过50ms，进行修正
    [[subCloud getAudioEffectManager] seekMusicToPosInMS:self.currentPlayMusicID pt
}
```

Android

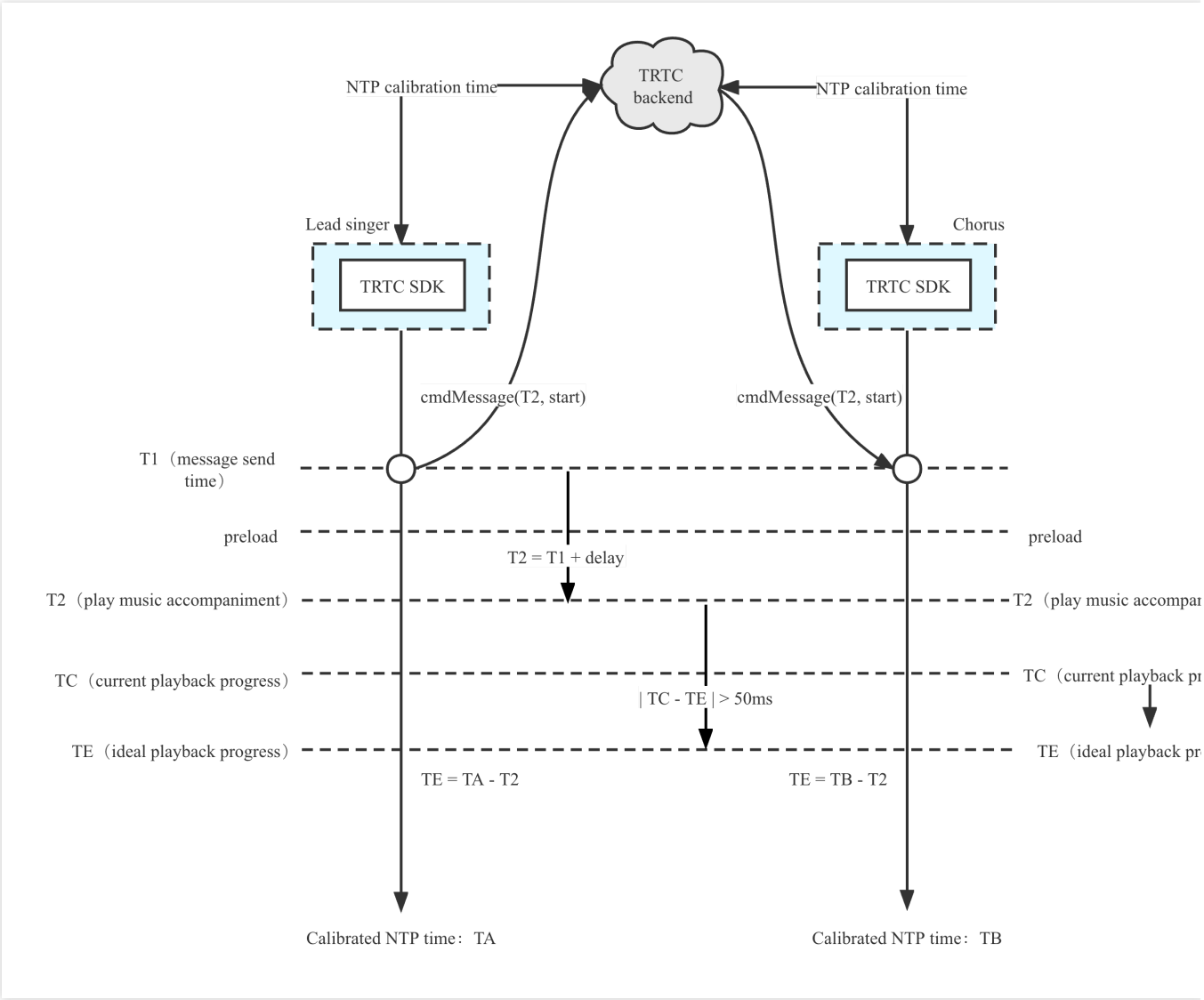
最近更新时间：2023-09-26 16:45:26

实时方案中需要在开唱后实时同步歌曲进度，避免因歌曲误差而增加端到端延迟。同步歌曲需要基于 NTP 时间，不同设备的本地时钟并不一致，存在一定误差，因此需要引入腾讯云自研 NTP 服务。同时，中途加入合唱的用户也需要同步歌曲进度，只有同步进度后，才能参与合唱。

实现流程

歌曲同步的做法为：主唱用户约定在未来某个时间点（如延迟 N 秒后）开始播放歌曲，其他用户参与合唱。各端的时间都以 NTP 时间为准，NTP 时间会在 TRTC SDK 初始化后开始同步。

版权所有：腾讯云计算（北京）有限责任公司



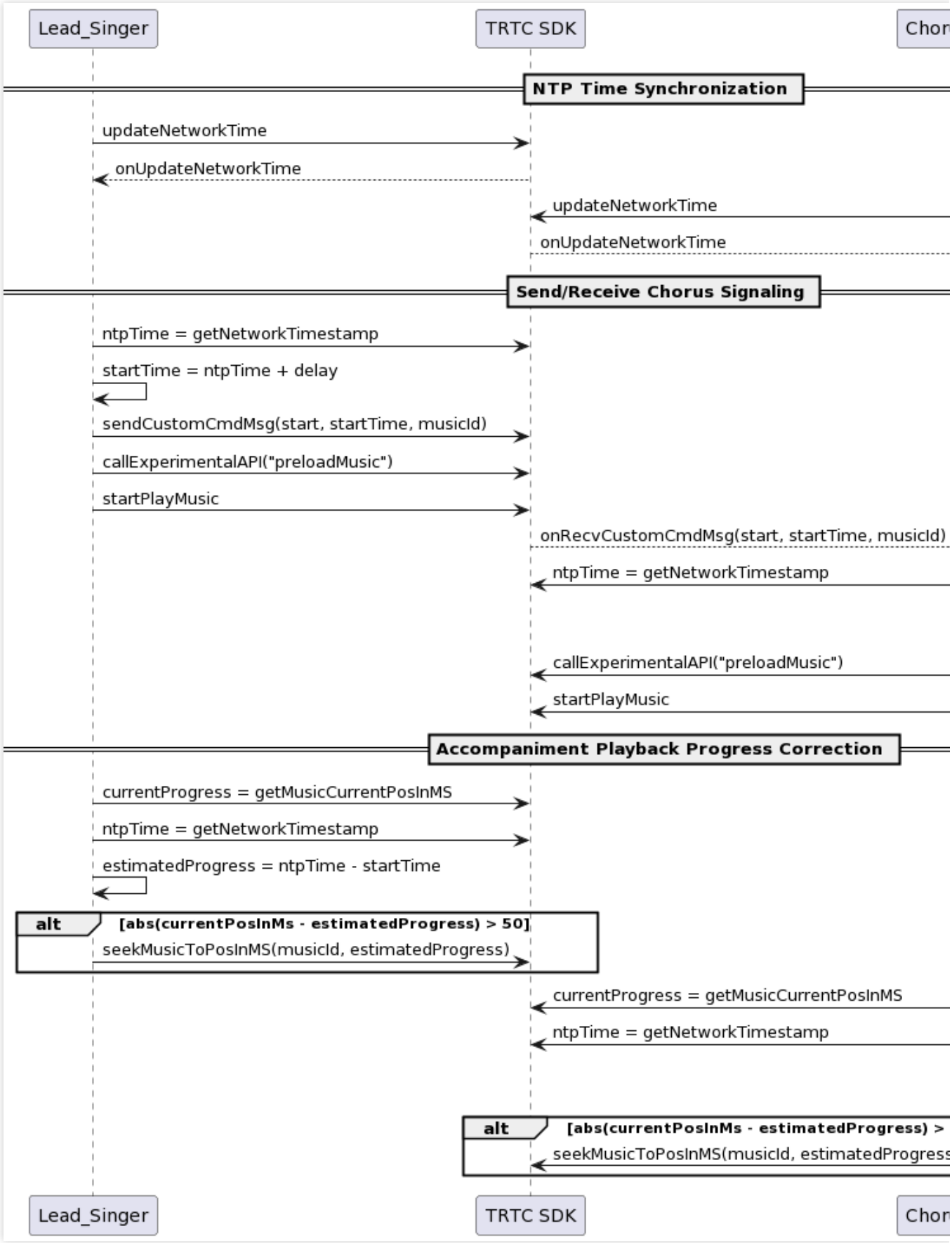
具体流程如下：

1. 各端进行 NTP 校时，向 TRTC 云端更新并获取最新的 NTP 时间 T。
2. 主唱端发送合唱信令（自定义消息），约定合唱开始时间 T2。
3. 本地根据 T2 预加载歌曲，定时播放。
4. 其他合唱用户接收到合唱信令后执行步骤 3。
5. 过程中对本地歌曲播放进度进行校验，当 TE 与 TC 差值超过 50ms 即 seek 校准。

说明：

这里的 50ms 误差是个典型值，可以根据业务容忍度适当调整，建议在 50ms 上下浮动。

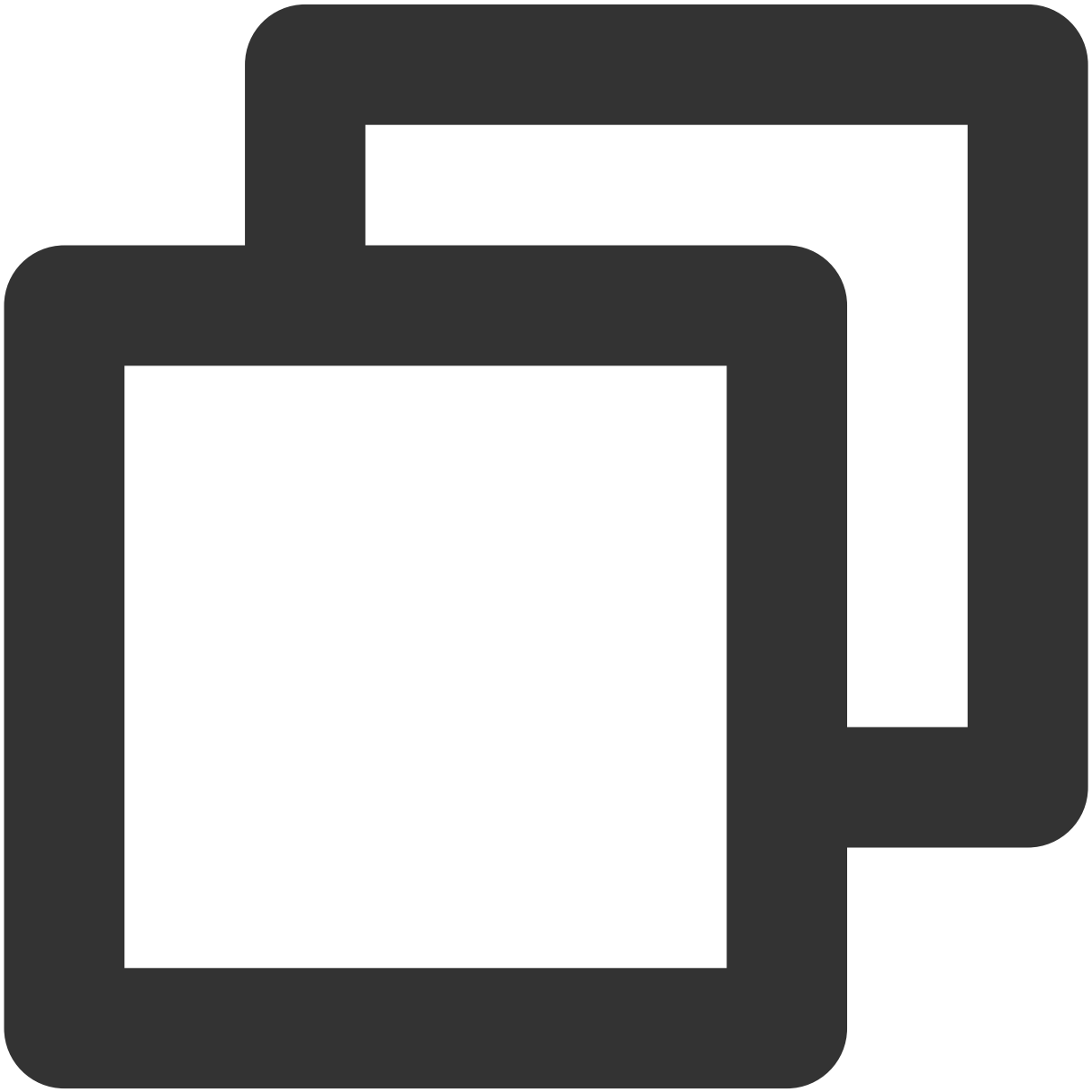
时序图


Lead_Singer
TRTC SDK
Chor
NTP Time Synchronization
updateNetworkTime
onUpdateNetworkTime
updateNetworkTime
onUpdateNetworkTime
Send/Receive Chorus Signaling
ntpTime = getNetworkTimestamp
startTime = ntpTime + delay
sendCustomCmdMsg(start, startTime, musicId)
callExperimentalAPI("preloadMusic")
startPlayMusic
onRecvCustomCmdMsg(start, startTime, musicId)
ntpTime = getNetworkTimestamp
callExperimentalAPI("preloadMusic")
startPlayMusic
Accompaniment Playback Progress Correction
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
alt
[abs(currentPosInMs - estimatedProgress) >
seekMusicToPosInMS(musicId, estimatedProgress
Lead_Singer
TRTC SDK
Chor

歌曲同步时序主要可以分为三个部分：NTP 校时、发送及接收合唱信令、歌曲播放进度修正。下面将针对这四部分给出具体的代码实现。

关键代码实现

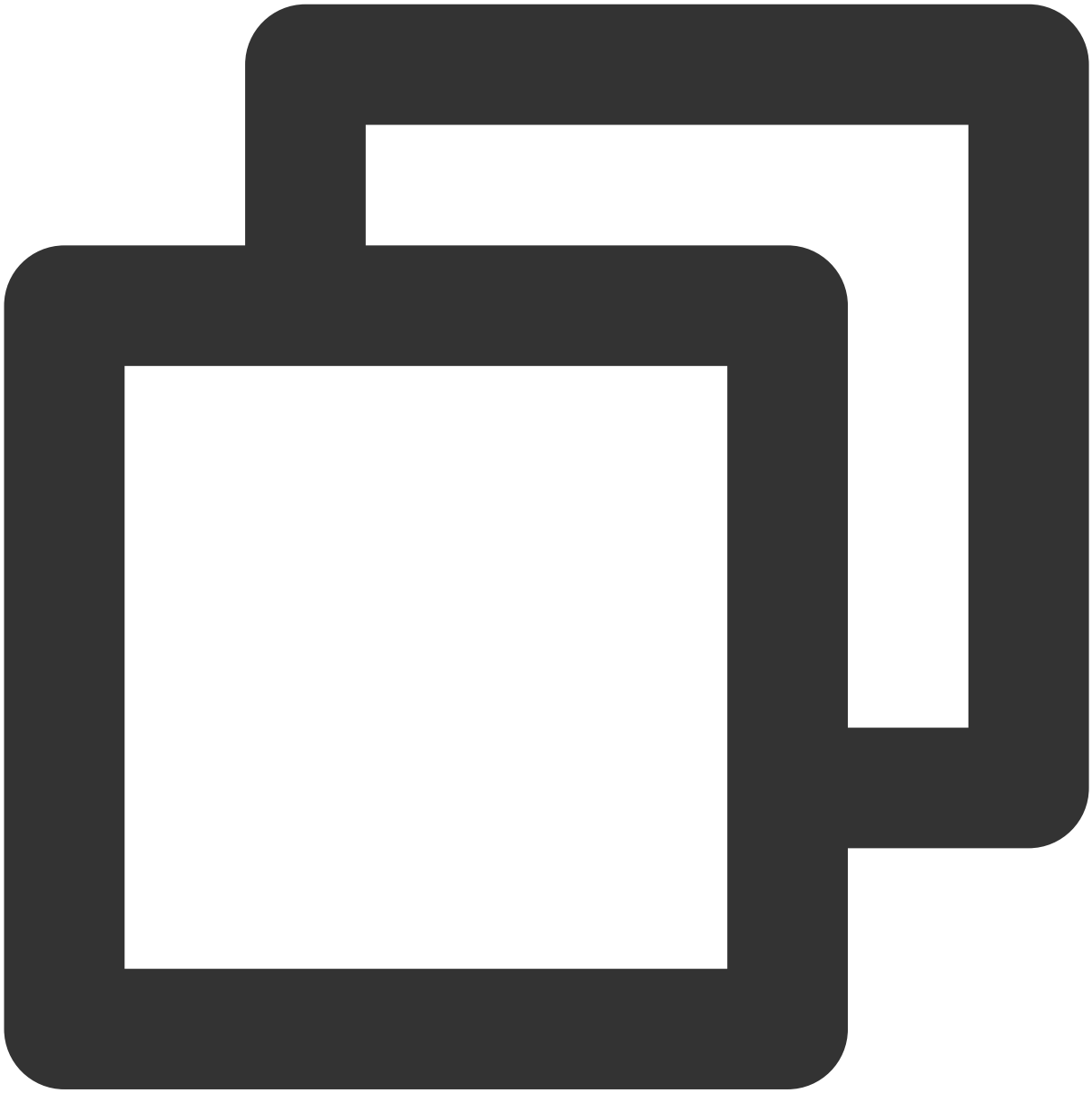
1. NTP 校时服务



```
TXLiveBase.setListener(new TXLiveBaseListener() {
```

```
    @Override
    public void onUpdateNetworkTime(int errCode, String errMsg) {
        super.onUpdateNetworkTime(errCode, errMsg);
        // errCode 0: 校时成功且偏差在30ms以内；1: 校时成功但偏差可能在30ms以上；-1: 校时失败
        if (errCode == 0) {
            // 调用TXLivebase的getNetworkTimestamp即可获取NTP时间戳
            long ntpTime = TXLiveBase.getNetworkTimestamp();
        } else {
            // 重新调用updateNetworkTime启动一次校时
            TXLiveBase.updateNetworkTime();
        }
    }
});
TXLiveBase.updateNetworkTime();
```

2. 主唱端发送合唱信令



```
JSONObject jsonObject = new JSONObject();
jsonObject.put("cmd", "startChorus");
// 约定时间合唱
jsonObject.put("startPlayMusicTS", startTs);
jsonObject.put("musicId", "musicId");
String body = jsonObject.toString();
mTRTCCloud.sendCustomCmdMsg(0, body.getBytes(), false, false);
```

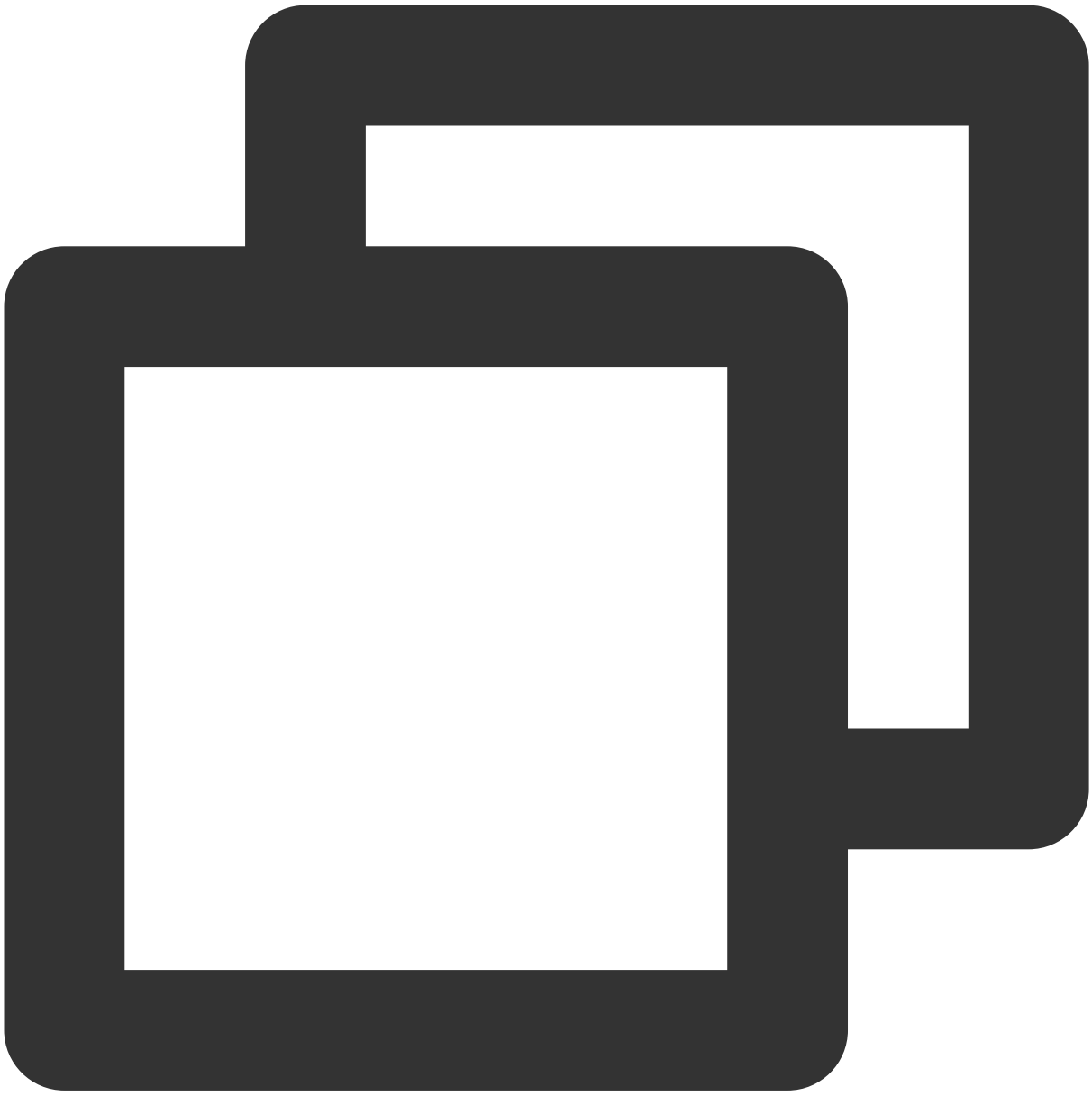
说明：

建议主唱以固定时间频率循环向房间内发送合唱信令消息，以便合唱者可以中途加入合唱。

版权所有：腾讯云计算（北京）有限责任公司

不使用 SEI 消息发送合唱信令的原因：SEI 信息会插入到视频帧中，导致观众侧拉取的视频流中携带很多无效信息。

3. 合唱端接收合唱信令



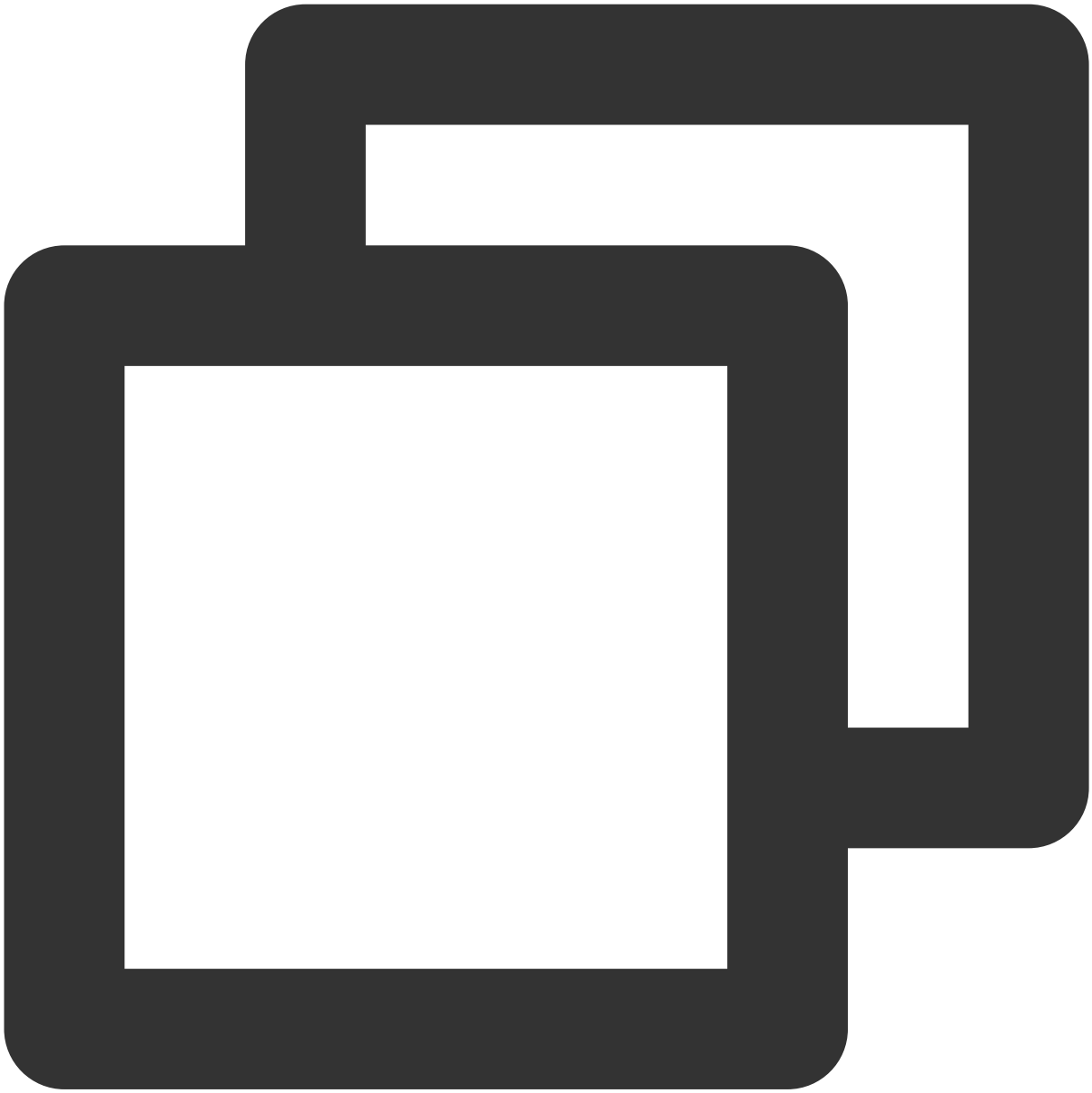
```
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[] message) {
    JSONObject json = new JSONObject(new String(message, "UTF-8"));
    String cmd = json.getString("cmd");
    // 合唱命令
    if (cmd.equals("startChorus")) {
    // 合唱开始时间
```

```
    long startPlayMusicTs = json.getLong("startPlayMusicTS");
    int musicId = json.getInt("musicId");
    // 合唱约定时间和当前时间差值
    long delayMs = Math.abs(startPlayMusicTs - getNtpTime());
    // 开启预加载，根据合唱约定时间和当前ntp差值，跳跃歌曲进度
    mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"preloadMusic\\",\\"params\\": {\\
    // 播放歌曲
    TXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMusi
    param.publish = false;
    mTRTCCloud.getAudioEffectManager().startPlayMusic(param);
}
```

说明：

合唱端接收到第一个 startChorus 信令后，状态应从“未合唱”转为“合唱中”，此后不再响应 startChorus 信令，避免重复启动 BGM 播放。

4. 歌曲播放进度修正



```
long mStartPlayMusicTs = "最初约定的合唱时间";
long currentProgress = subCloud.getAudioEffectManager().getMusicCurrentPosInMS(musi
// 当前歌曲的理想播放时间进度
long estimatedProgress = getNtpTime() - mStartPlayMusicTs;
// 当播放进度超过50ms，进行修正
if (estimatedProgress >= 0 &&; Math.abs(currentProgress - estimatedProgress) > 50)
    subCloud.getAudioEffectManager().seekMusicToPosInMS(mMusicID, (int) estimatedPr
}
```

歌词同步
iOS

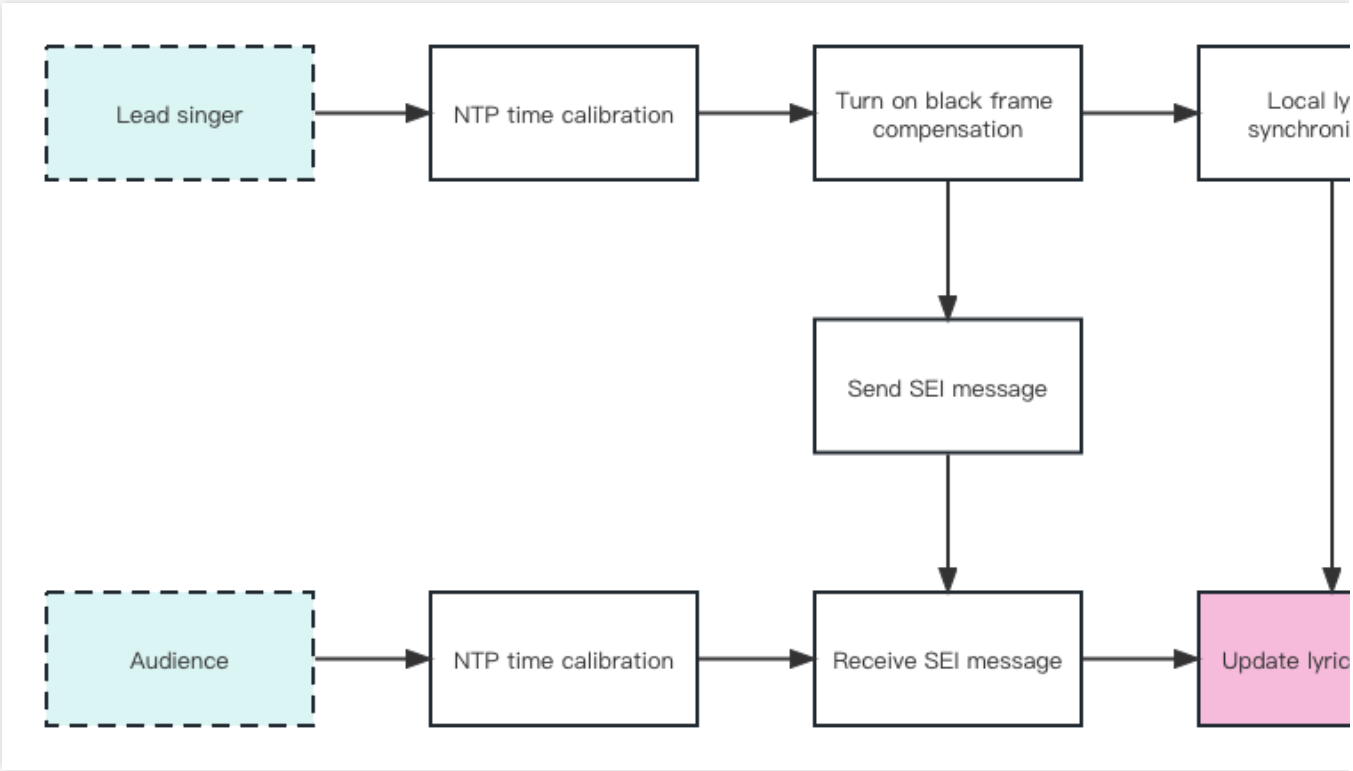
最近更新时间：2023-09-27 14:36:32

1.1 实现流程

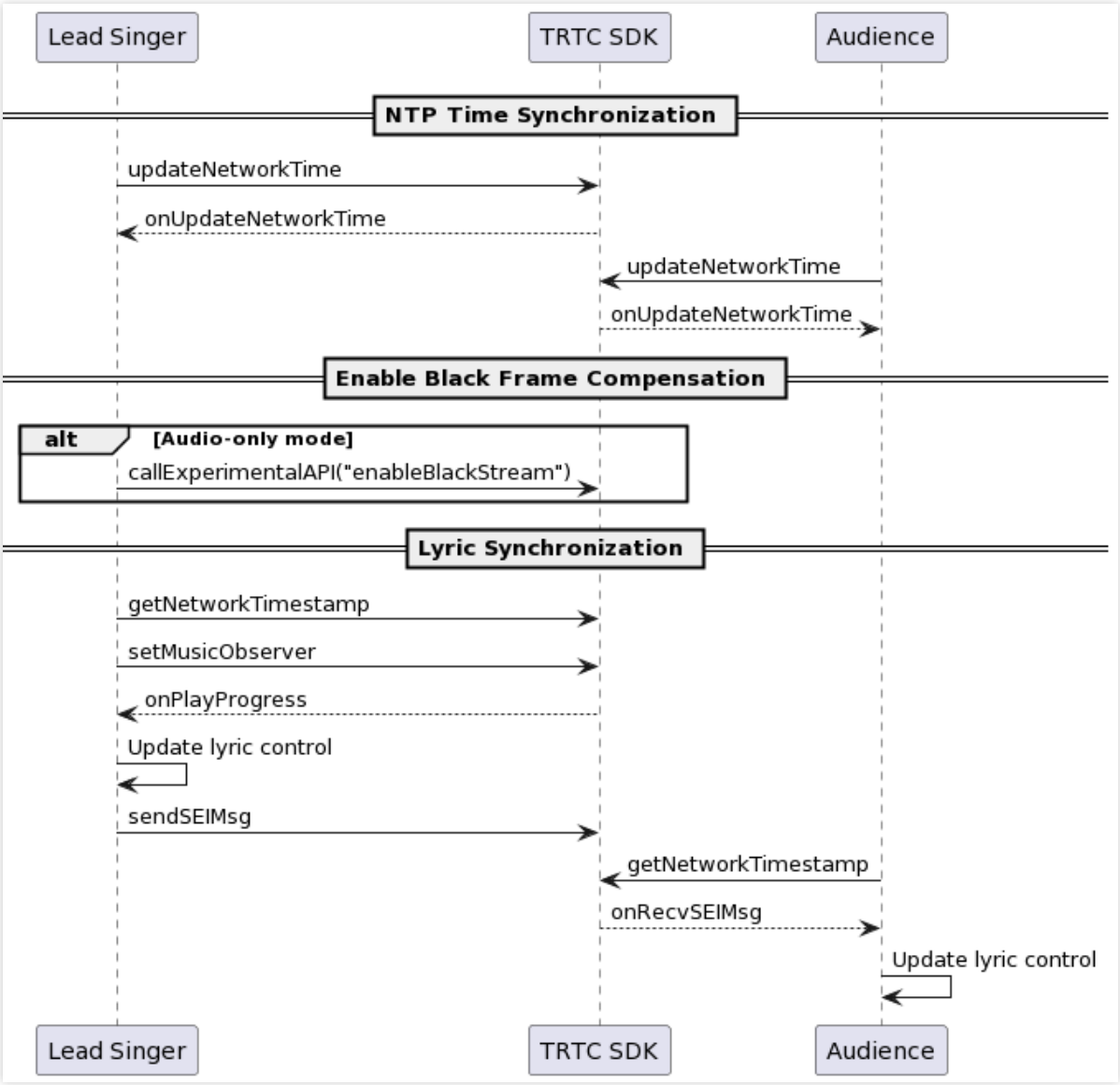
歌词同步方案中，三种不同角色的动作如下：

主唱	合唱	观众
NTP 校时 开启补黑帧 发送 SEI 消息 本地歌词同步 更新歌词控件	NTP 校时 本地歌词同步 更新歌词控件	NTP 校时 接收 SEI 消息 更新歌词控件

其中，主唱及合唱根据同步后的歌曲播放进度，在本地更新歌词进度；观众端则需要接收由主唱端发送的，包含最新歌词进度的 SEI 消息来更新本地的歌词进度。



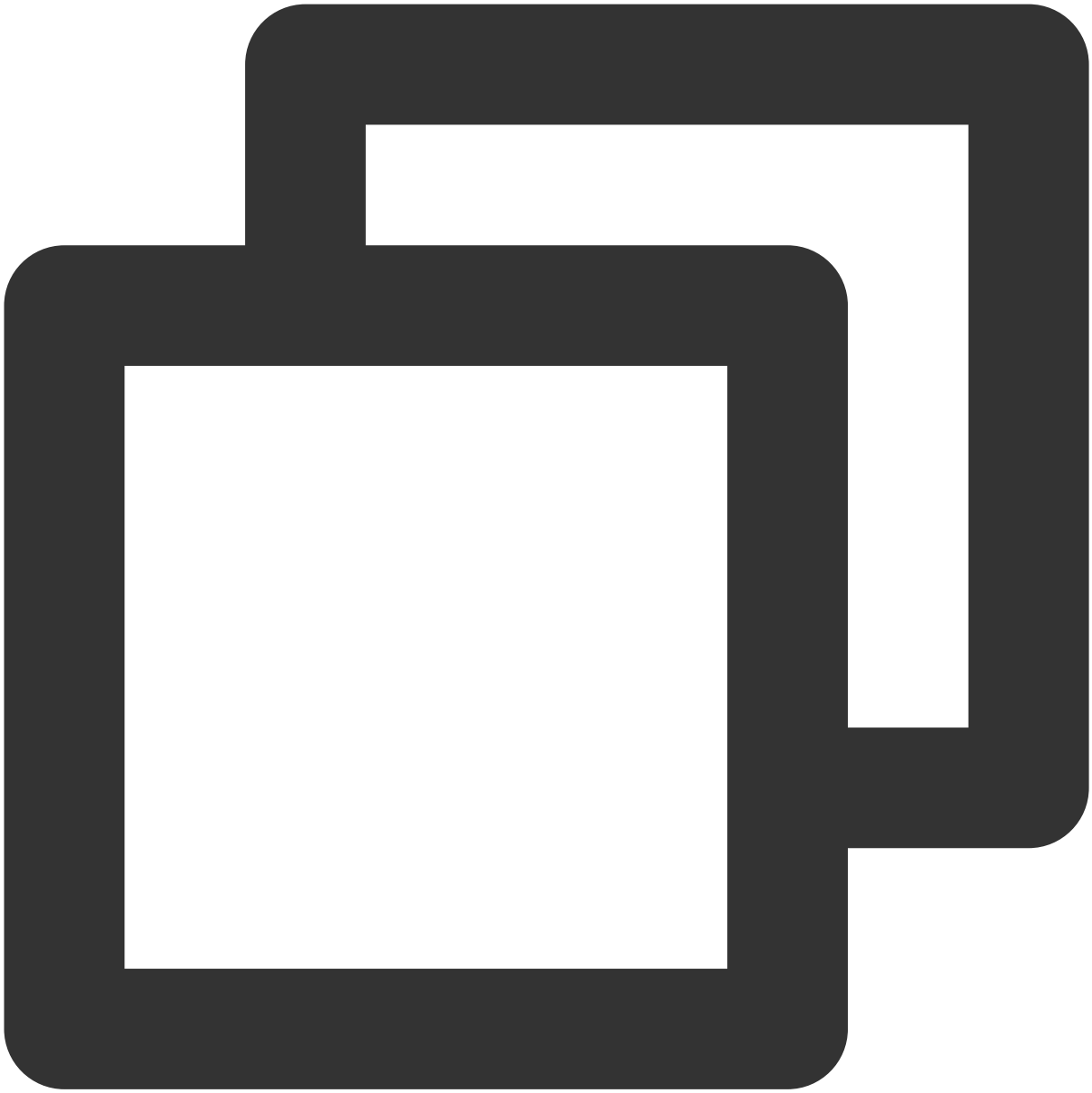
时序图



歌词同步时序主要可以分为三个部分：NTP 校时、开启补黑帧、本地及远端歌词同步。NTP 校时的代码实现在歌曲同步中已经给出，下面将针对后两个部分给出具体的代码实现。

关键代码实现

1. 开启补黑帧



```
// 纯音频模式下，主实例（人声实例）需要开启补黑帧以携带 SEI 消息
NSDictionary *jsonDic = @{
                            @"api": @"enableBlackStream",
                            @"params":
                                 @{
                                    @"enable": @(1)
                                  }
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[trtcCloud callExperimentalAPI:jsonString];
```

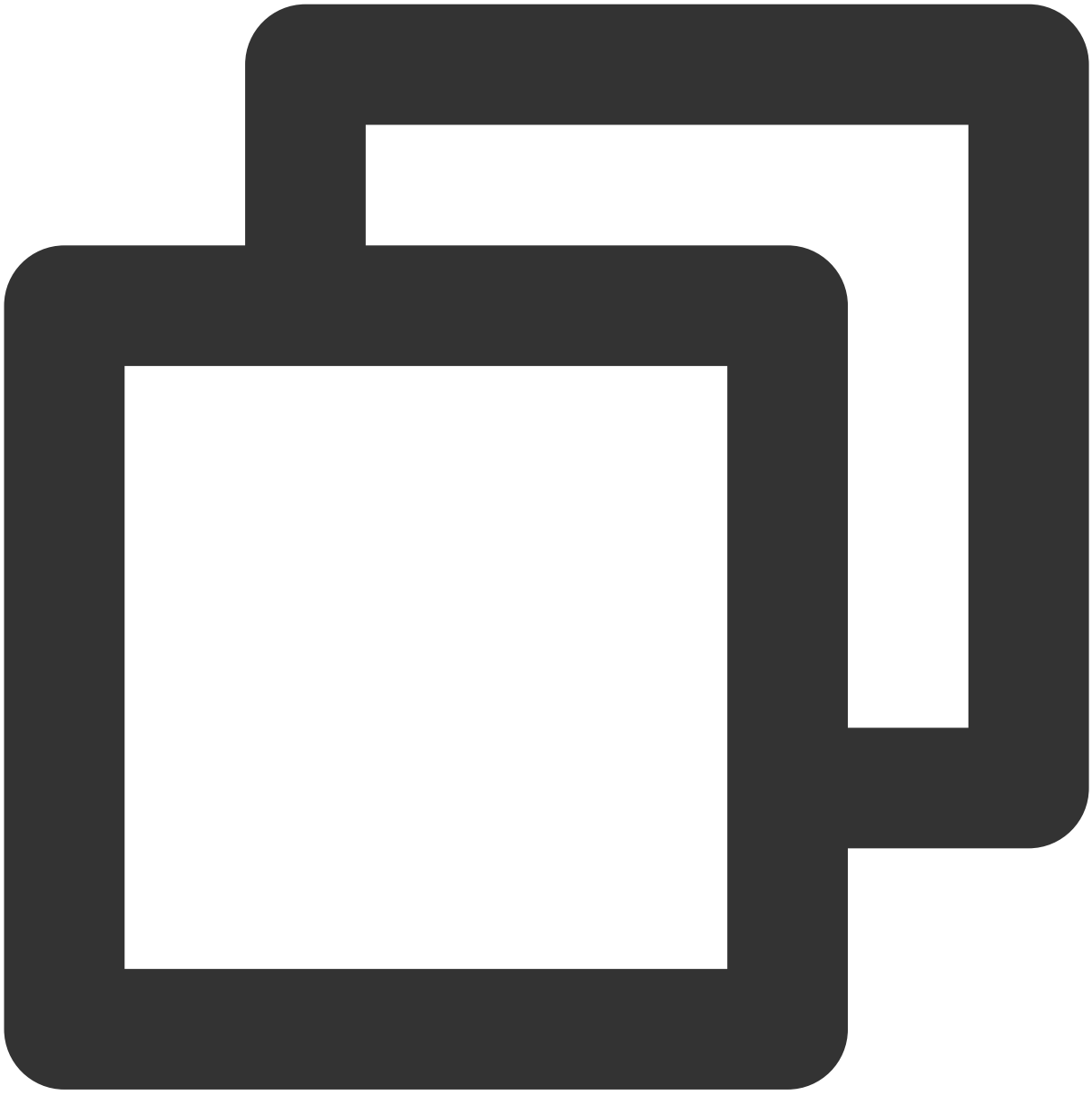
说明：

该实验性接口 enableBlackStream 需要在进房之后调用；

在 Android 端，enable 参数的值类型为布尔型，在 iOS 端为整型；

接收端需要在收到 onUserVideoAvailable(userId, true) 时调用 startRemoteView(userId, null)。

2. 通过 SEI 消息发送歌曲进度



```
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs, NSInteger duratio
    //当前 ntp 时间
```

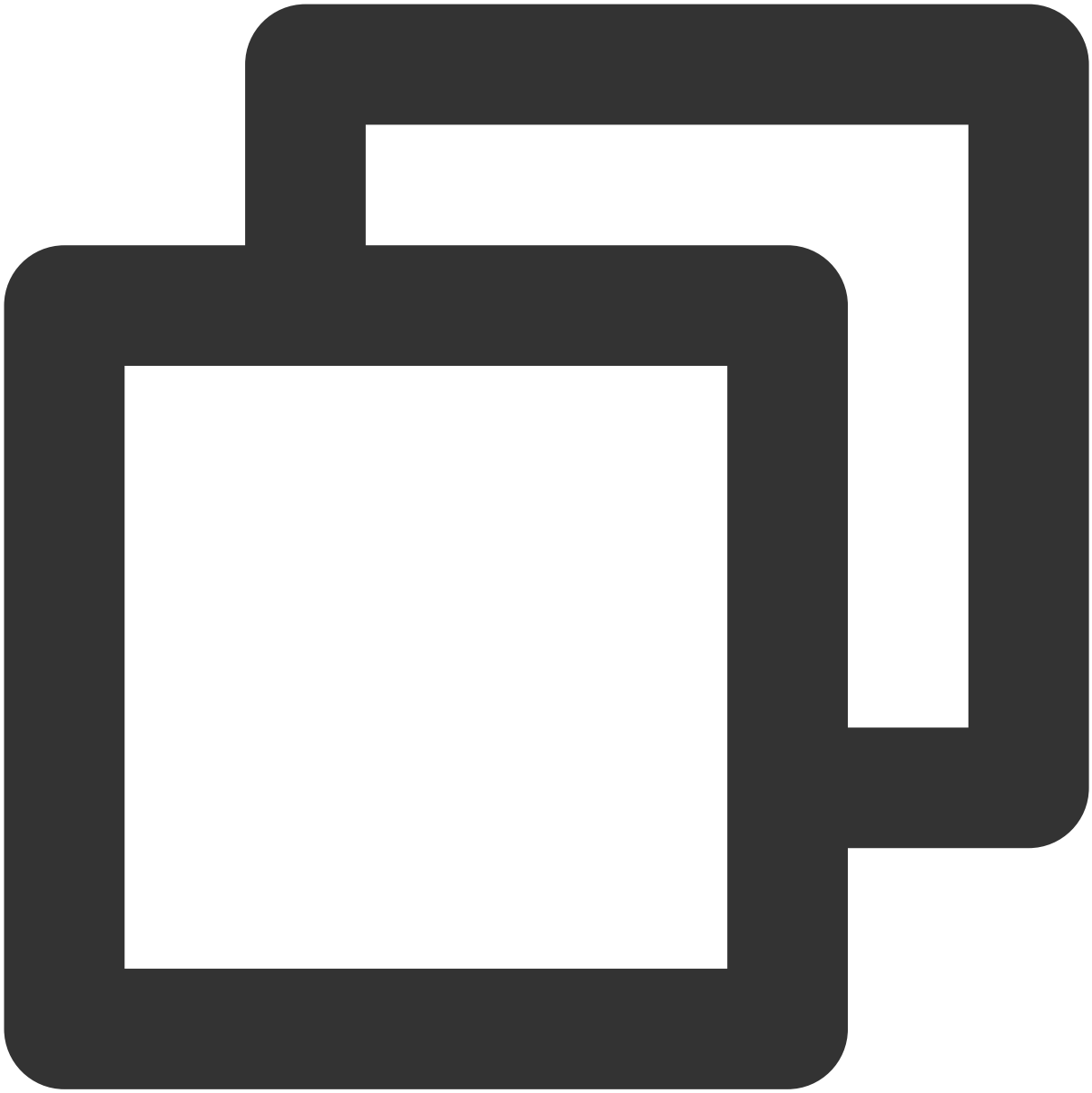
```
    NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    //通知歌曲进度，用户会在这里进行歌词的滚动
    NSDictionary *progressMsg = @{
            @"bgmProgressTime":@(progressMs),
            @"ntpTime":@(ntpTime),
            @"musicId": @(musicId),
            @"duration": @(durationMs),
    };
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:progressMsg options:
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8S
    [trtcCloud sendSEIMsg:[jsonString dataUsingEncoding:NSUTF8StringEncoding] repea
};
```

说明：

主唱发送 SEI 消息的频率由背景音乐播放事件回调的频率决定，一般为 200ms；

不直接使用 CMD 消息发送歌曲进度的原因：SEI 通道传输的信令可以伴随视频帧一直传输到直播 CDN 上，对于拉取 CDN 流的观众具有更好的兼容性。

3. 本地及远端歌词同步



```
// 本地歌词同步
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs, NSInteger duratio
    ...
    // TODO 更新歌词控件逻辑：
    // 根据最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
    ...
};

// 远端歌词同步
- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
    NSError *err = nil;
```

```
    NSDictionary *dic = [NSJSONSerialization JSONObjectWithData:message options:NSJ
    if (err || ![dic isKindOfClass:[NSDictionary class]]) {
        // 解析出错
        return;
    }
    NSInteger bgmProgressTime = [[dic objectForKey:@"bgmProgressTime"] integerValue
    NSInteger ntpTime = [[dic objectForKey:@"ntpTime"] integerValue];
    int32_t musicId = [[dic objectForKey:@"musicId"] intValue];
    NSInteger duration = [[dic objectForKey:@"duration"] integerValue];
    ...
    // TODO 更新歌词控件逻辑：
    // 根据接收到的最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
    ...
}
```

说明：

如果复用 TUIKaraoke 组件的歌词控件，请参照TUIKaraoke TRTCLyricView部分的代码逻辑同步歌词控件进度。

Android

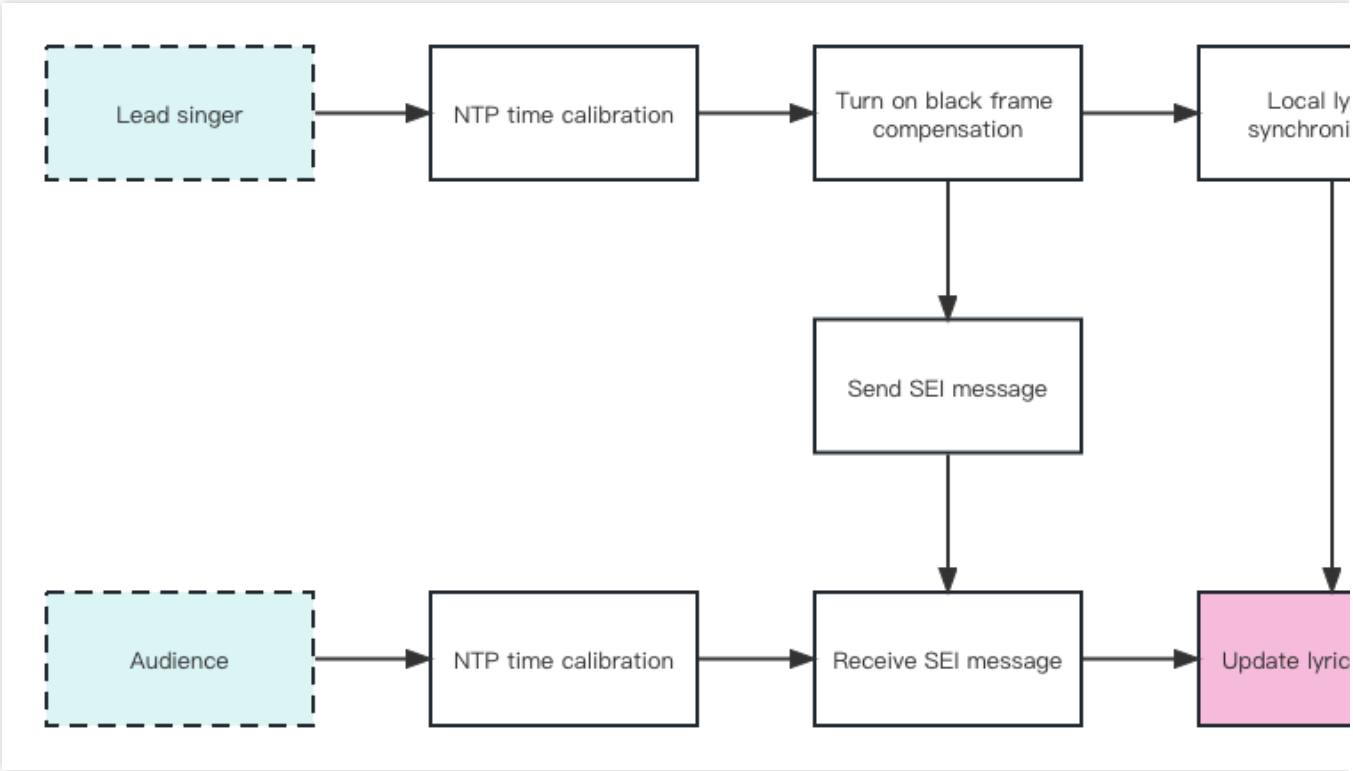
最近更新时间：2023-09-27 11:22:08

实现流程

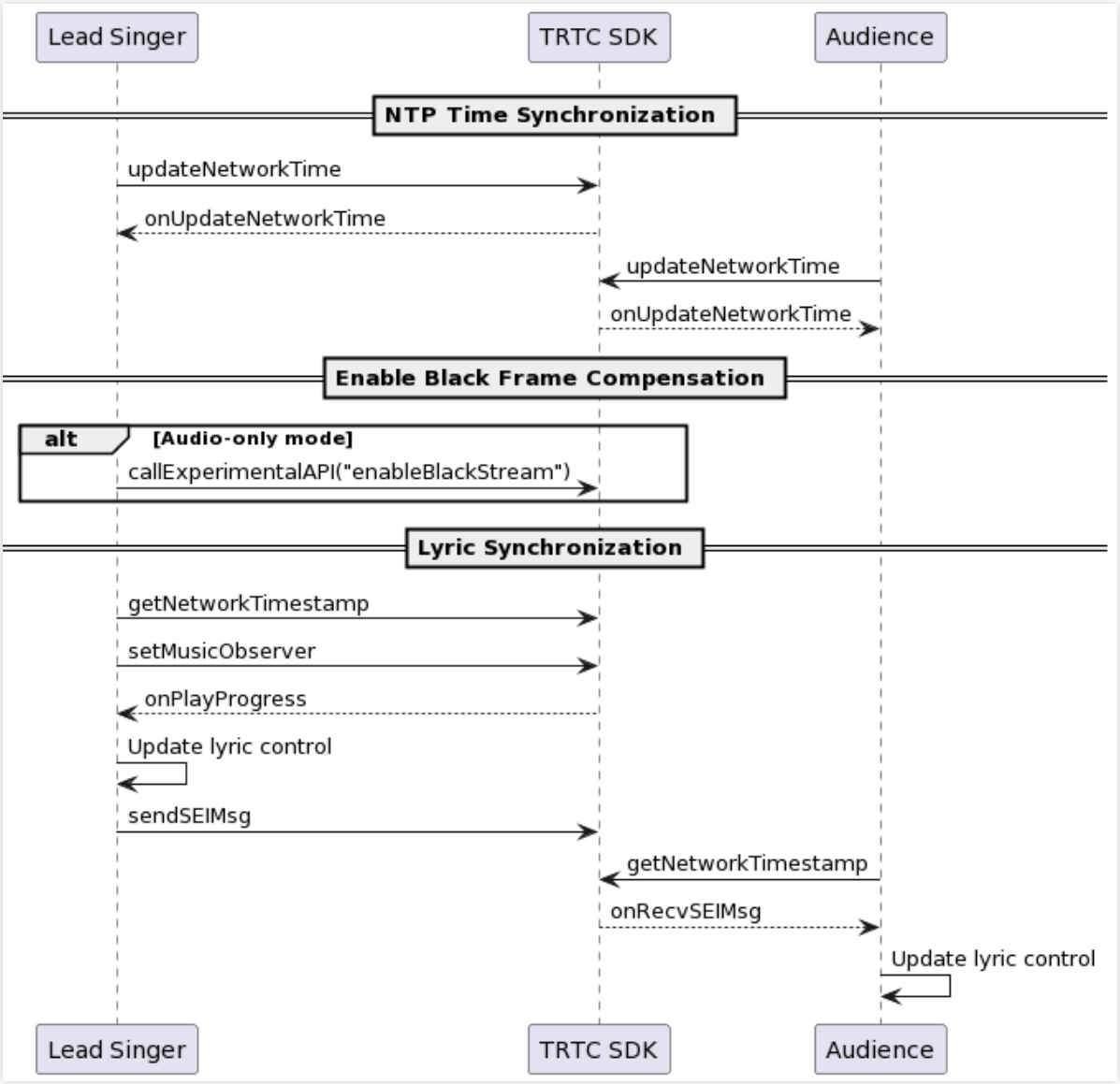
歌词同步方案中，三种不同角色的动作如下：

主唱	合唱	观众
NTP 校时 开启补黑帧 发送 SEI 消息 本地歌词同步 更新歌词控件	NTP 校时 本地歌词同步 更新歌词控件	NTP 校时 接收 SEI 消息 更新歌词控件

其中，主唱及合唱根据同步后的歌曲播放进度，在本地更新歌词进度；观众端则需要接收由主唱端发送的，包含最新歌词进度的 SEI 消息来更新本地的歌词进度。



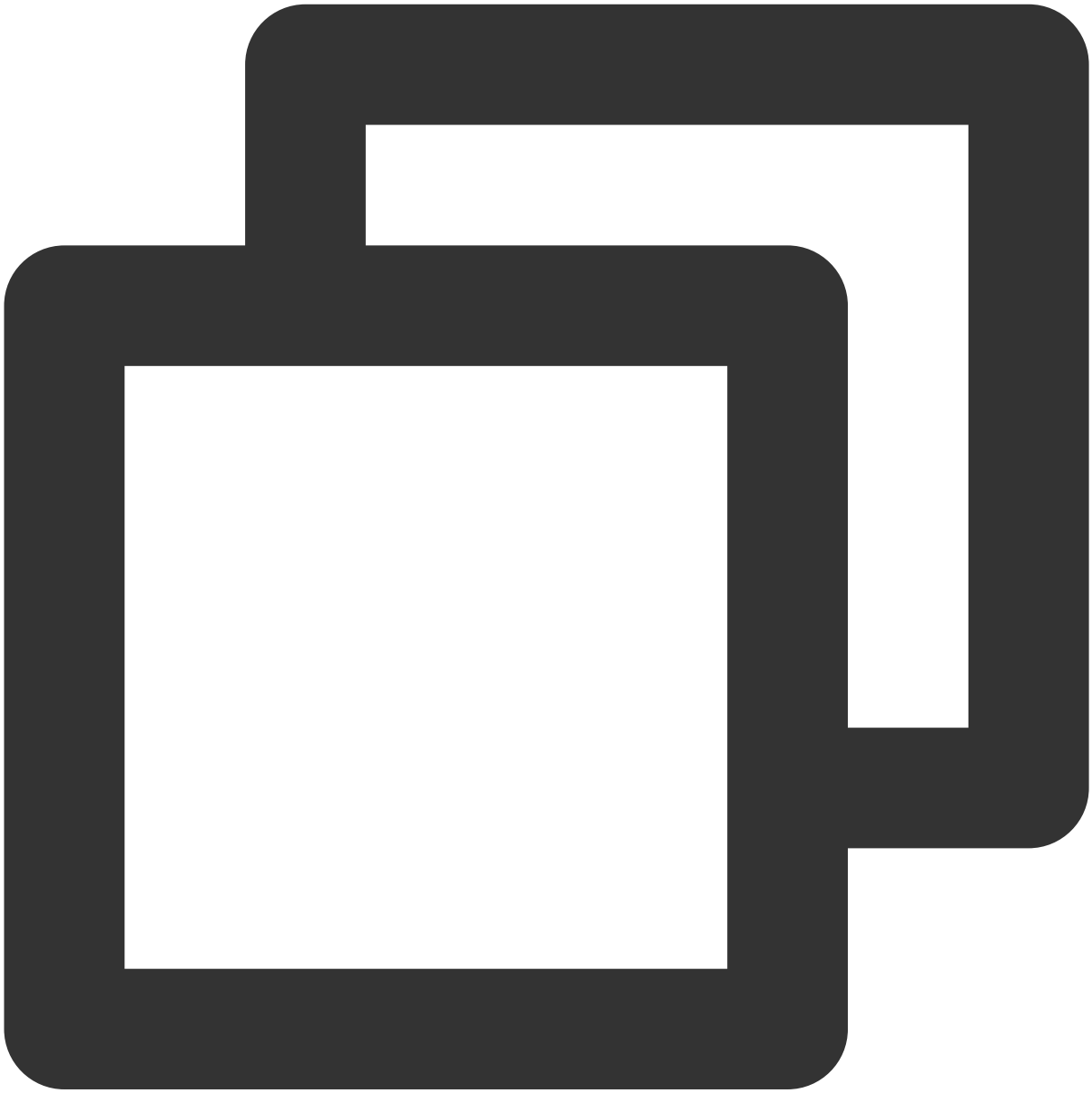
时序图



歌词同步时序主要可以分为三个部分：NTP 校时、开启补黑帧、本地及远端歌词同步。NTP 校时的代码实现在歌曲同步中已经给出，下面将针对后两个部分给出具体的代码实现。

关键代码实现

1. 开启补黑帧



```
// 纯音频模式下，主实例（人声实例）需要开启补黑帧以携带 SEI 消息
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"enableBlackStream\\",\\"params\\": {\
```

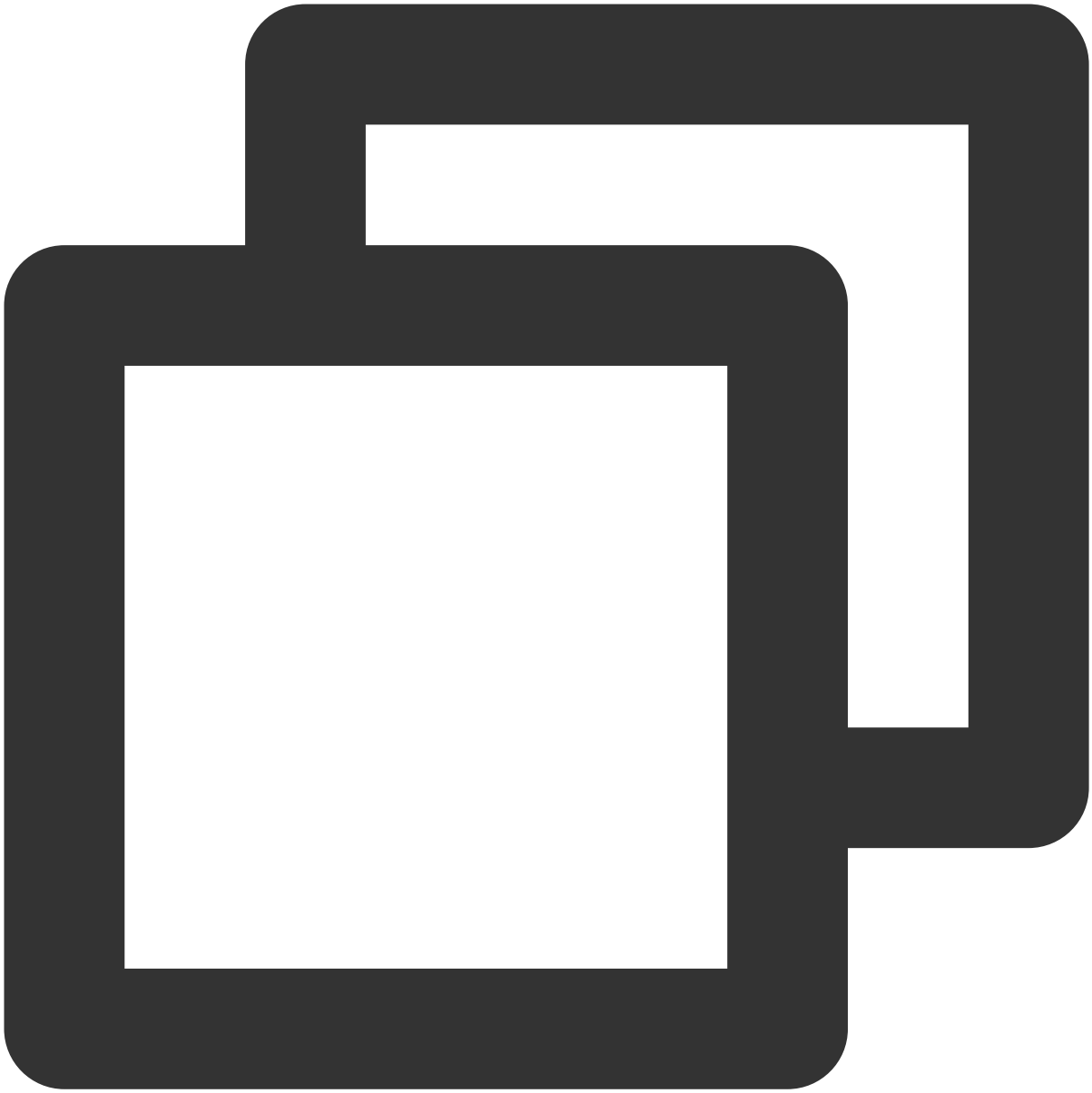
说明：

该实验性接口 enableBlackStream 需要在进房之后调用。

在 Android 端，enable 参数的值类型为布尔型，在 iOS 端为整型。

接收端需要在收到 onUserVideoAvailable(userId, true) 时调用 startRemoteView(userId, null)。

2. 通过 SEI 消息发送歌曲进度



```
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new TXAudioEffectManager.TXMu
    @Override
    public void onPlayProgress(int id, long curPtsMS, long durationMS) {
        JSONObject jsonObject = new JSONObject();
        // 当前 ntp 时间
        long ntpTime = TXLiveBase.getNetworkTimestamp();
        jsonObject.put("bgmProgressTime", curTime);
        jsonObject.put("ntpTime", ntpTime);
        jsonObject.put("musicId", musicId);
        jsonObject.put("duration", duration);
        jsonObject.toString().getBytes();
```

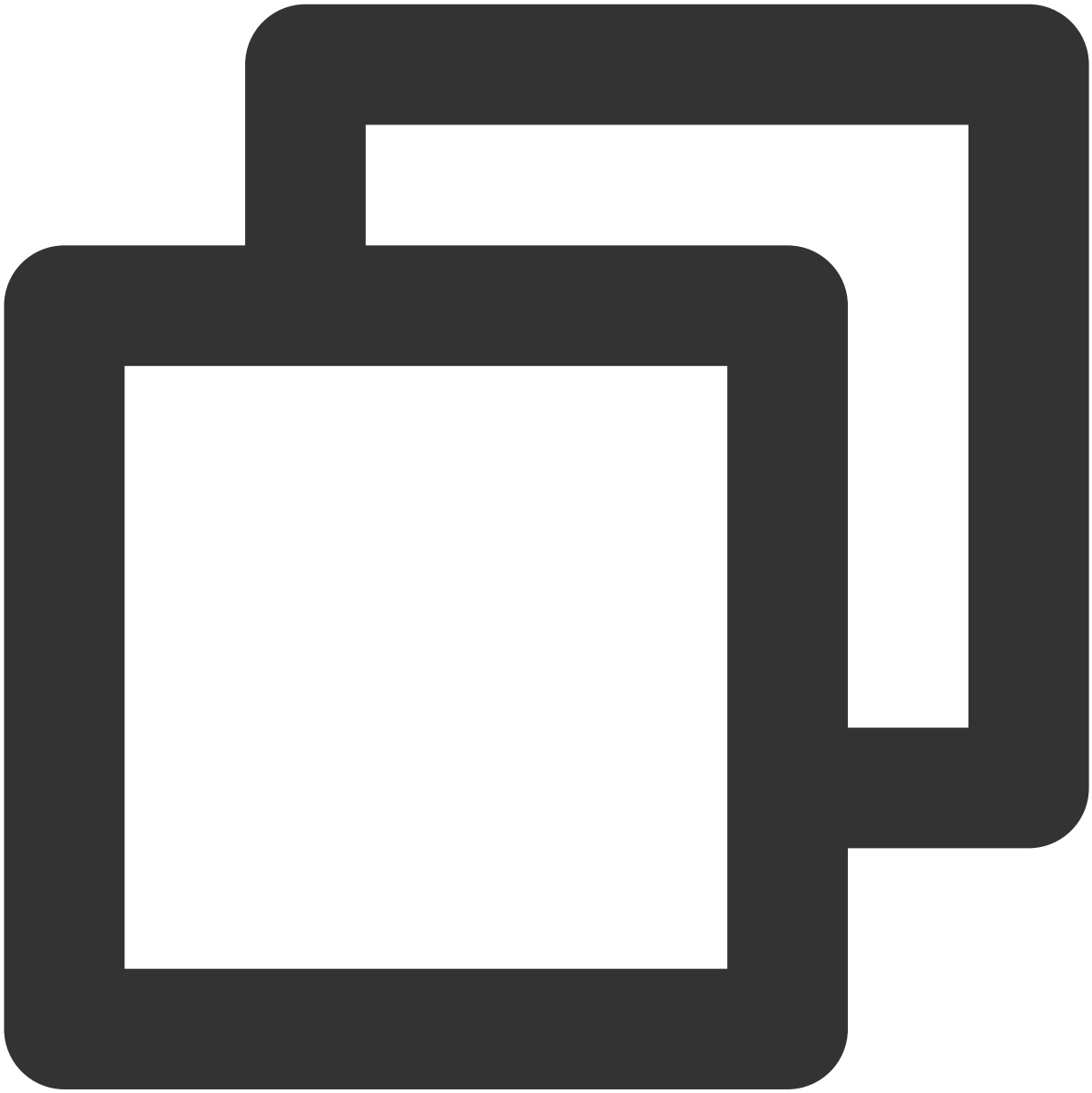
```
            mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
        }
    }
```

说明：

主唱发送 SEI 消息的频率由背景音乐播放事件回调的频率决定，一般为 200ms；

不直接使用 CMD 消息发送歌曲进度的原因：SEI 通道传输的信令可以伴随视频帧一直传输到直播 CDN 上，对于拉取 CDN 流的观众具有更好的兼容性。

3. 本地及远端歌词同步



```
// 本地歌词同步
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new TXAudioEffectManager.TXMu
    @Override
    public void onPlayProgress(int id, long curPtsMS, long durationMS) {
        ...
        // TODO 更新歌词控件逻辑：
        // 根据最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
        ...
    }
}
```

```
// 远端歌词同步
@Override
public void onRecvSEIMsg(String userId, byte[] data) {
    String result = new String(data);
    JSONObject jsonObject = new JSONObject(result);
    long bgmProgressTime = jsonObject.getLong("bgmProgressTime");
    long ntpTime = jsonObject.getLong("ntpTime");
    String musicId = jsonObject.getString("musicId");
    long duration = jsonObject.getLong("duration");
    ...
    // TODO 更新歌词控件逻辑：
    // 根据接收到的最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
    ...
}
```

说明：

如果复用 TUIKaraoke 组件的歌词控件，请参照TUIKaraoke LyricsView部分的代码逻辑同步歌词控件进度。

人声同步
iOS

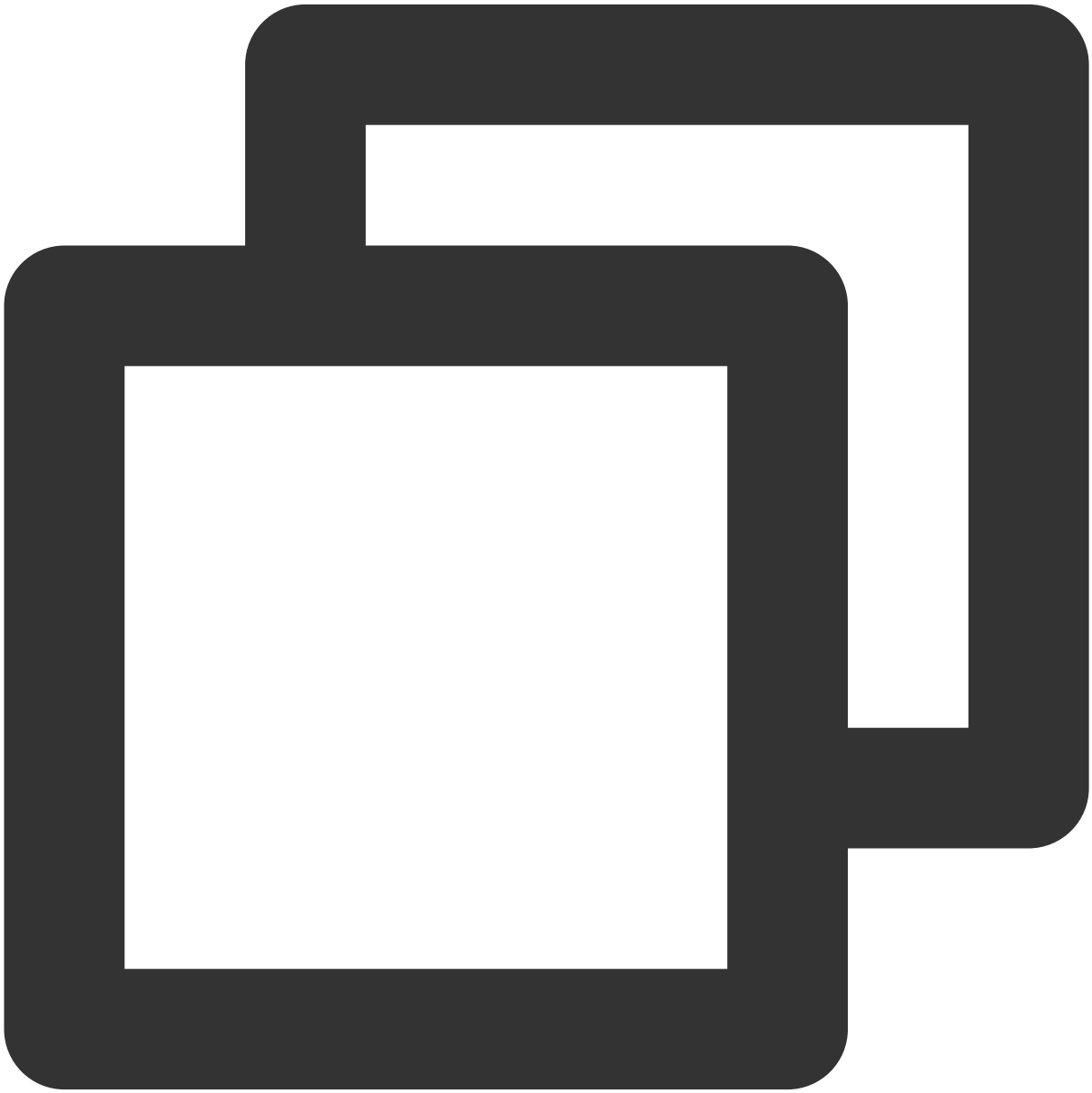
最近更新时间：2023-09-26 16:53:03

人声与歌曲同步介绍

因为本地人声采集的 jitter buffer、歌曲播放混音的 jitter buffer 以及声音播放到人耳到歌唱存在有一定的 GAP 的，所以演唱者完全对着歌词和 BGM 播放时候，在远端观众感觉 BGM 播放、人声、歌词是有一定的延迟的，合唱方案在 TRTC SDK 内部使用了使用低延迟的 AAudio 采集，具体只需要在进房后开启合唱模式与低延时模式。

具体代码实现

开启合唱模式



```
// 主实例（人声实例）开启合唱模式（调低缓冲区间、音频冗余保护）
NSDictionary *jsonDic = @{
                            @"api": @"enableChorus",
                            @"params": @{
                                        @"enable": @(YES),
                                        @"audioSource": @(0)
                                       }
                         };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[trtcCloud callExperimentalAPI:jsonString];
```

```
// 子实例（伴奏实例）开启合唱模式（调低缓冲区间、音频冗余保护）
NSDictionary *jsonDic = @{
                                @"api": @"enableChorus",
                                @"params": @{
                                            @"enable": @(YES),
                                            @"audioSource": @(1)
                                          }
                            };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

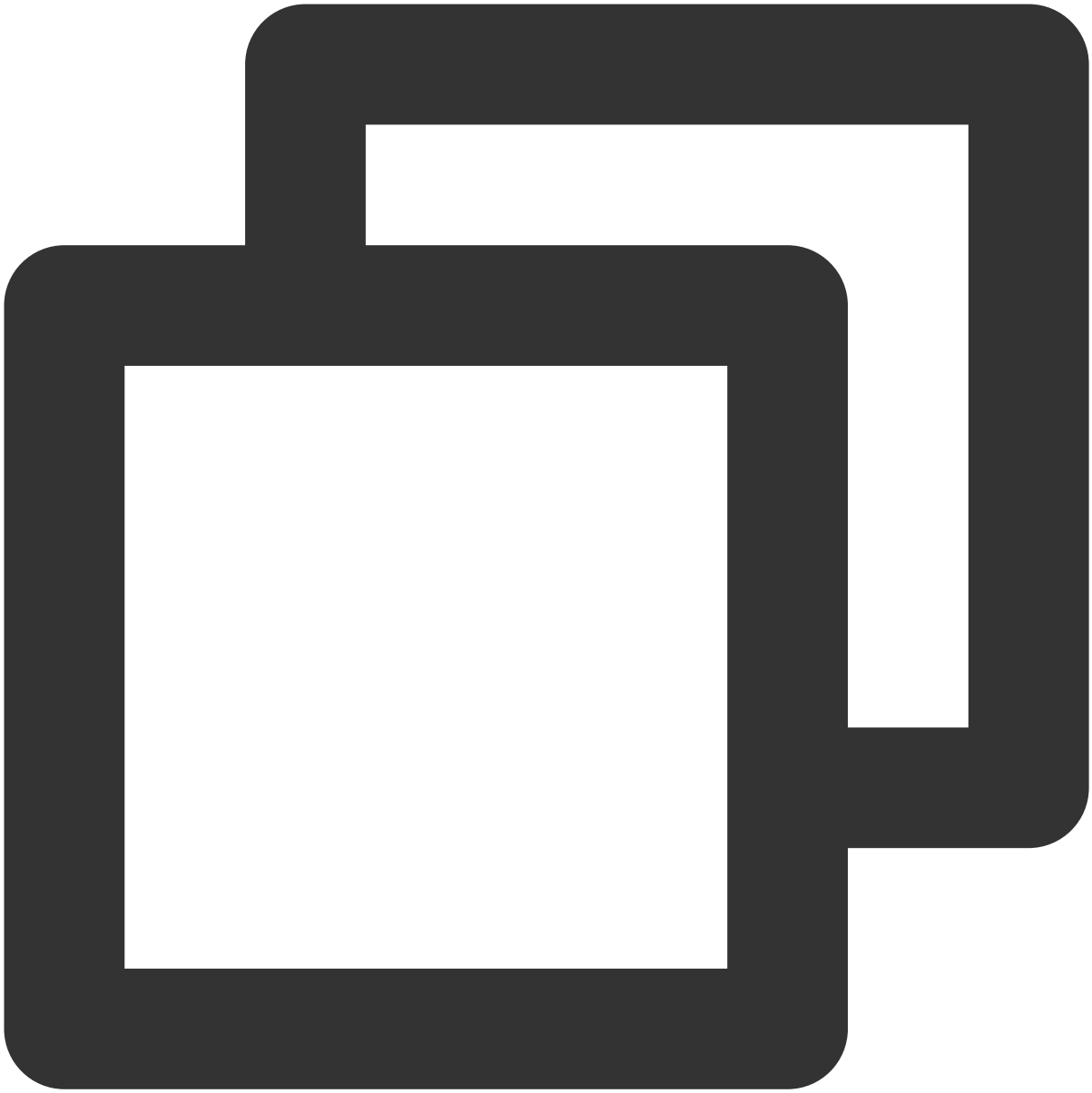
说明：

开启合唱模式的实验性接口 enableChorus 的参数设置：

audioSource：0（人声）

audioSource：1（伴奏）

开启低延时模式（高性能音频 AAudio）



```
// 主实例（人声实例）开启高性能音频 AAudio
NSDictionary *jsonDic = @{
                            @"api": @"setLowLatencyModeEnabled",
                            @"params": @{
                                        @"enable": @(1)
                                      }
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[trtcCloud callExperimentalAPI:jsonString];
// 子实例（伴奏实例）开启高性能音频 AAudio
```

```
NSDictionary *jsonDic = @{
                            @"api": @"setLowLatencyModeEnabled",
                            @"params": @{
                                        @"enable": @(1)
                                      }
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

Android

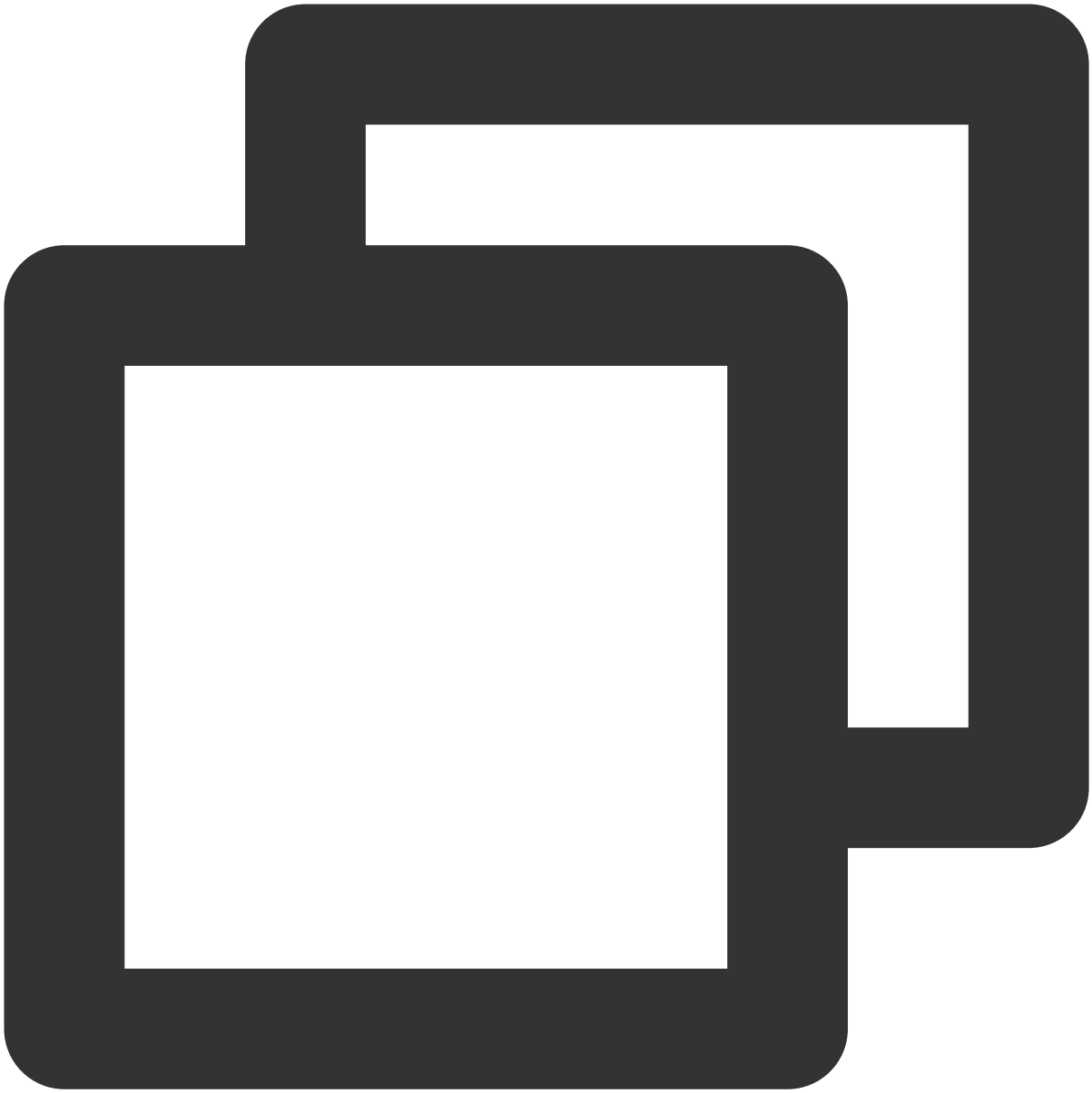
最近更新时间：2023-09-26 16:53:25

人声与歌曲同步介绍

因为本地人声采集的 jitter buffer、歌曲播放混音的 jitter buffer 以及声音播放到人耳到歌唱存在有一定的 GAP 的，所以演唱者完全对着歌词和 BGM 播放时候，在远端观众感觉 BGM 播放、人声、歌词是有一定的延迟的。合唱方案在 TRTC SDK 内部使用了使用低延迟的 AAudio 采集，具体只需要在进房后开启合唱模式与低延时模式。

具体代码实现

开启合唱模式



```
// 主实例（人声实例）开启合唱模式（调低缓冲区间、音频冗余保护）
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"enableChorus\\",\\"params\\":{\\"enab
// 子实例（伴奏实例）开启合唱模式（调低缓冲区间、音频冗余保护）
subCloud.callExperimentalAPI("{\\"api\\":\\"enableChorus\\",\\"params\\":{\\"enable
```

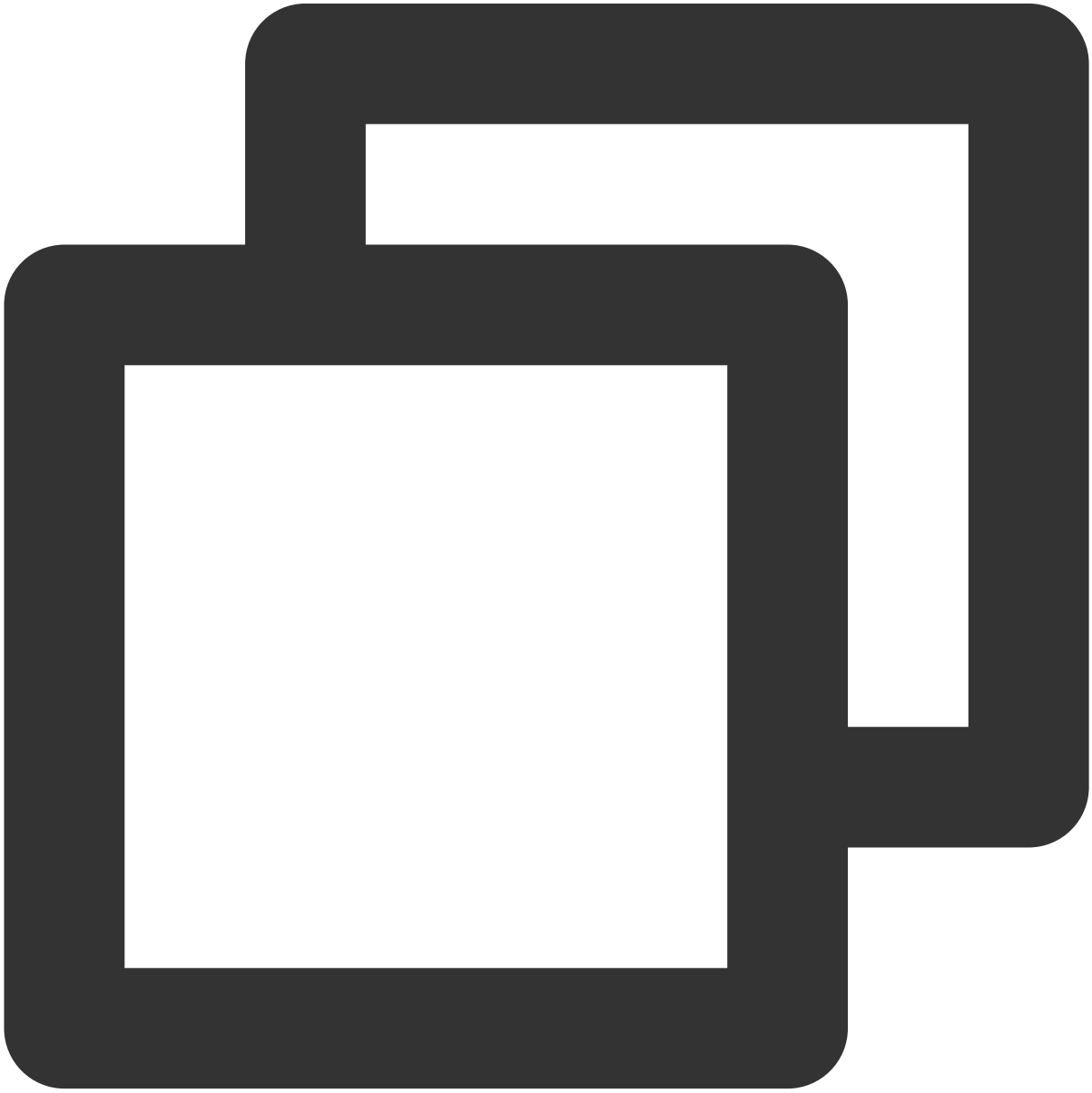
说明：

开启合唱模式的实验性接口 enableChorus 的参数设置：

audioSource：0（人声）。

audioSource：1（伴奏）。

开启低延时模式（高性能音频 AAudio）



```
// 主实例（人声实例）开启高性能音频 AAudio
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"setLowLatencyModeEnabled\\",\\"params
// 子实例（伴奏实例）开启高性能音频 AAudio
subCloud.callExperimentalAPI("{\\"api\\":\\"setLowLatencyModeEnabled\\",\\"params\\
```

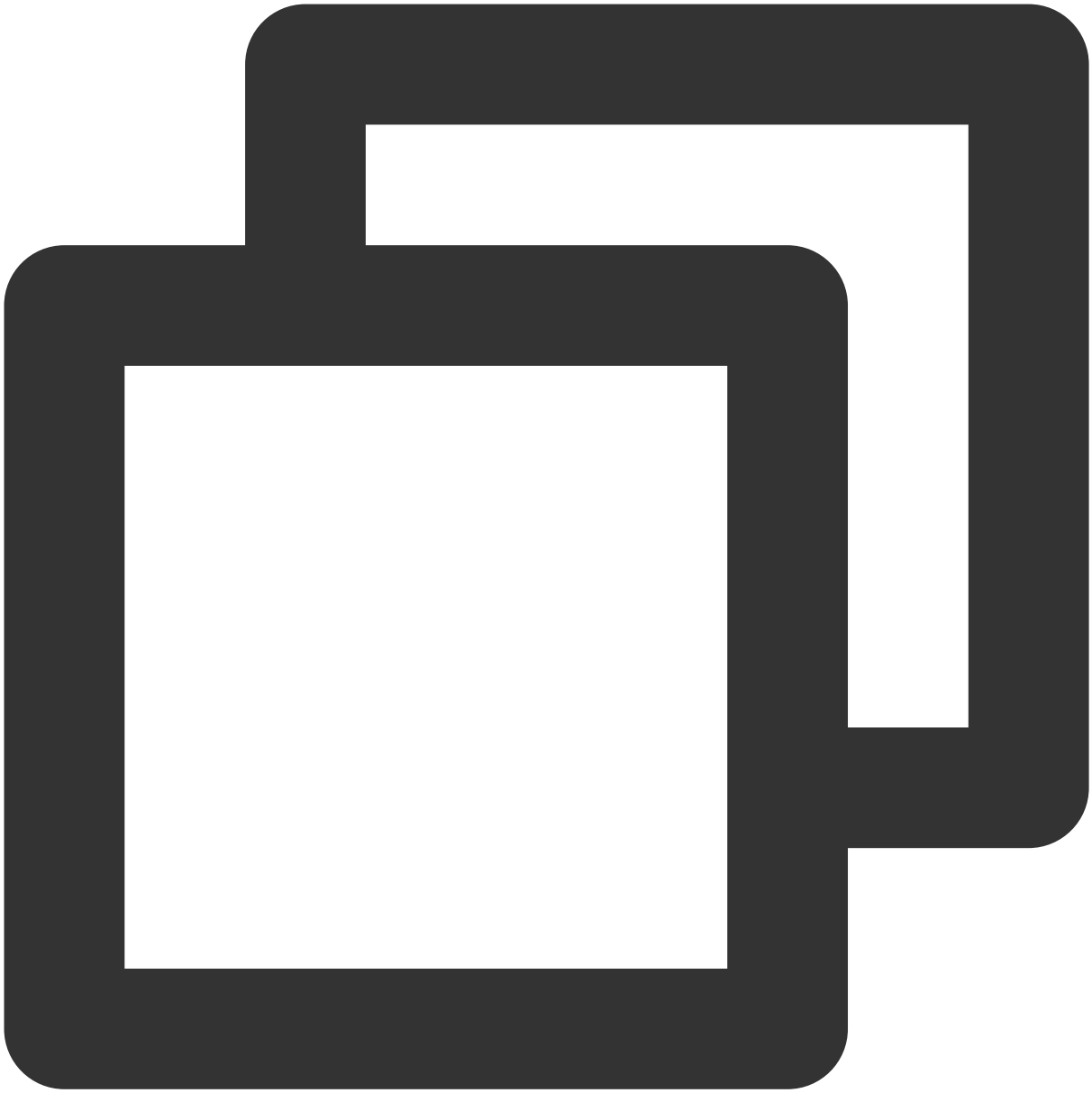
混流方案

iOS

最近更新时间：2023-09-26 16:53:49

具体代码实现

1. 创建主子实例

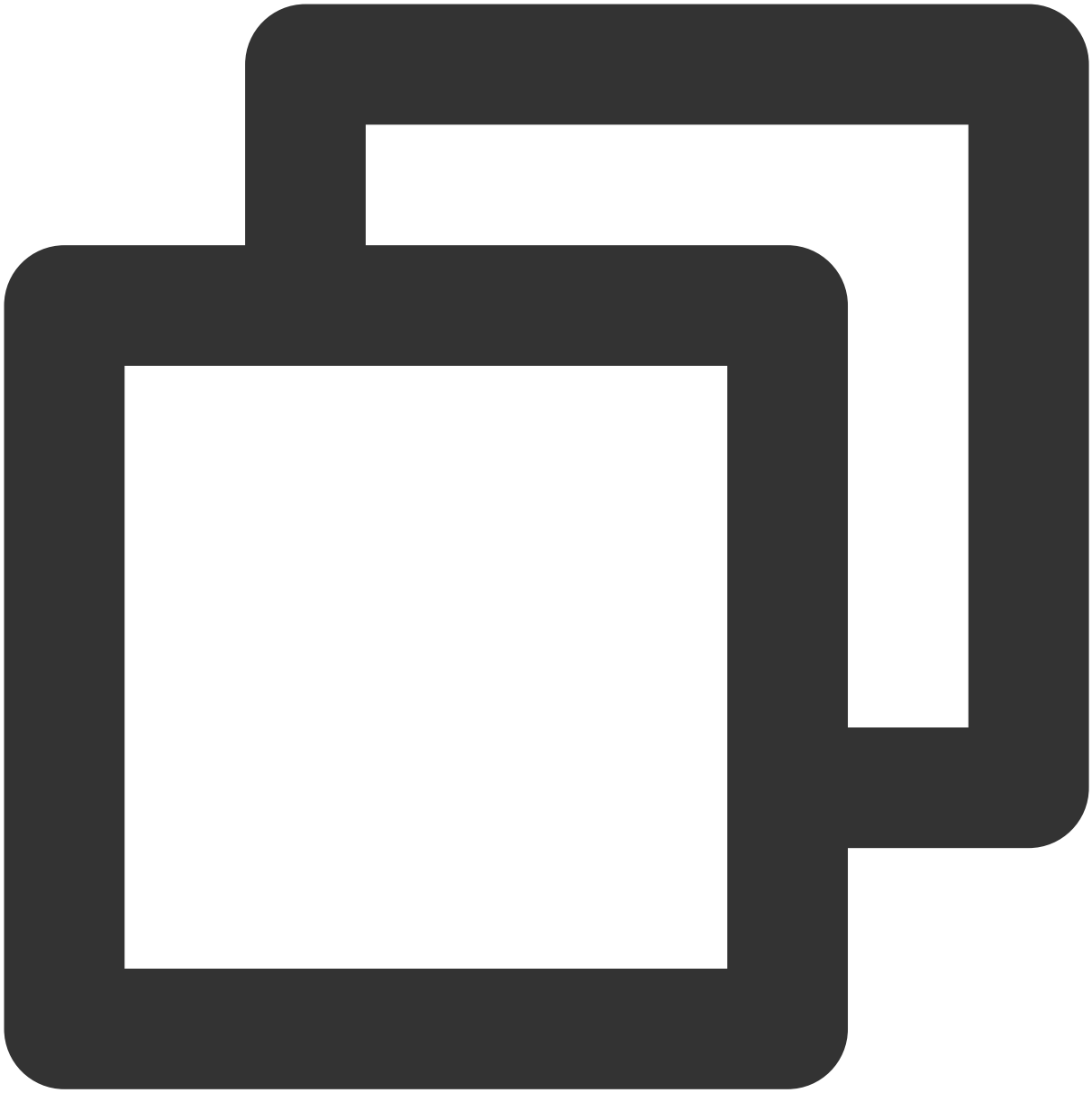


```
// 创建 TRTCCloud 主实例（人声实例）
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
// 创建 TRTCCloud 子实例（伴奏实例）
TRTCCloud *subCloud = [trtcCloud createSubCloud];
```

注意：

实时合唱方案中，主唱端需要分别创建主实例-人声实例和子实例-伴奏实例，分别用于上行人声及伴奏音乐。

2. 人声实例进房推流



```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = sdkAppId;
params.userId = userId;
params.userSig = userSign;
params.role = TRTCRoleAnchor;
params.roomId = roomIdIntValue;
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
// 打开音频上行，设置音质
[trtcCloud startLocalAudio:TRTCAudioQualityMusic];
// 设置媒体类型
[trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
```

```
// 静音远端伴奏音乐
[trtcCloud muteRemoteAudio:remoteAudioId mute:YES];
```

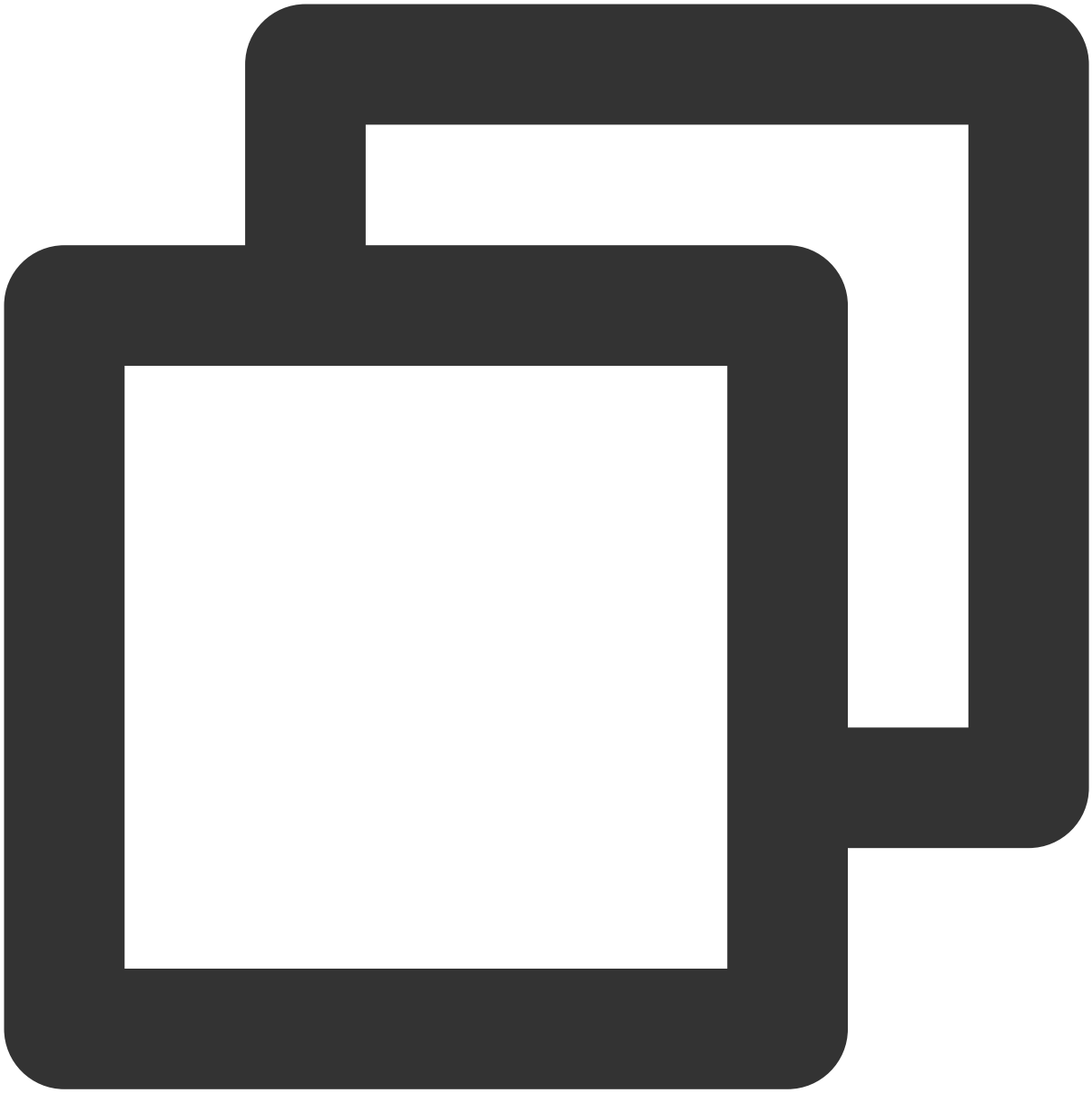
注意：

纯 RTC 音频场景下，进房场景推荐选用 VOICE_CHATROOM。

若有视频或转推 CDN 需求，进房场景则须选用 LIVE，VOICE_CHATROOM 会在转推时添加纯音频参数，从而导致 SEI 消息无法透传。

主唱/合唱端需要 muteRemoteAudio(true) 取消订阅伴奏实例上行的音频流，否则会重复播放本地及远端的伴奏音乐。

3. 伴奏实例进房推流



```
TRTCParams *bgmParams = [[TRTCParams alloc] init];
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = [NSString stringWithFormat:@"%@%@",userId,@"_bgm"];
bgmParams.userSig = bgmUserSign;
bgmParams.role = TRTCRoleAnchor;
bgmParams.roomId = roomIdIntValue;
[subCloud enterRoom:bgmParams appScene:TRTCAppSceneLIVE];
//设置媒体类型
[subCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
// 开启预加载
NSDictionary *jsonDict = @{
```

```
                                @"api": @"preloadMusic",
                                @"params": @{
                                            @"musicId": @(self.currentPlayMusicID),
                                            @"path": path,
                                            @"startTimeMS": @(startMs),
                                           }
                               };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:0 error
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
// 播放伴奏音乐并推流（在约定时间播放）
TXAudioMusicParam *musicParam = [[TXAudioMusicParam alloc] init];
musicParam.ID = musicID;
musicParam.path = url;
musicParam.loopCount = 0;
musicParam.publish = YES;
// 将伴奏音乐传到远端
param.publish = YES;
[[subCloud getAudioEffectManager] startPlayMusic:musicParam onStart:startBlock onPr
```

注意：

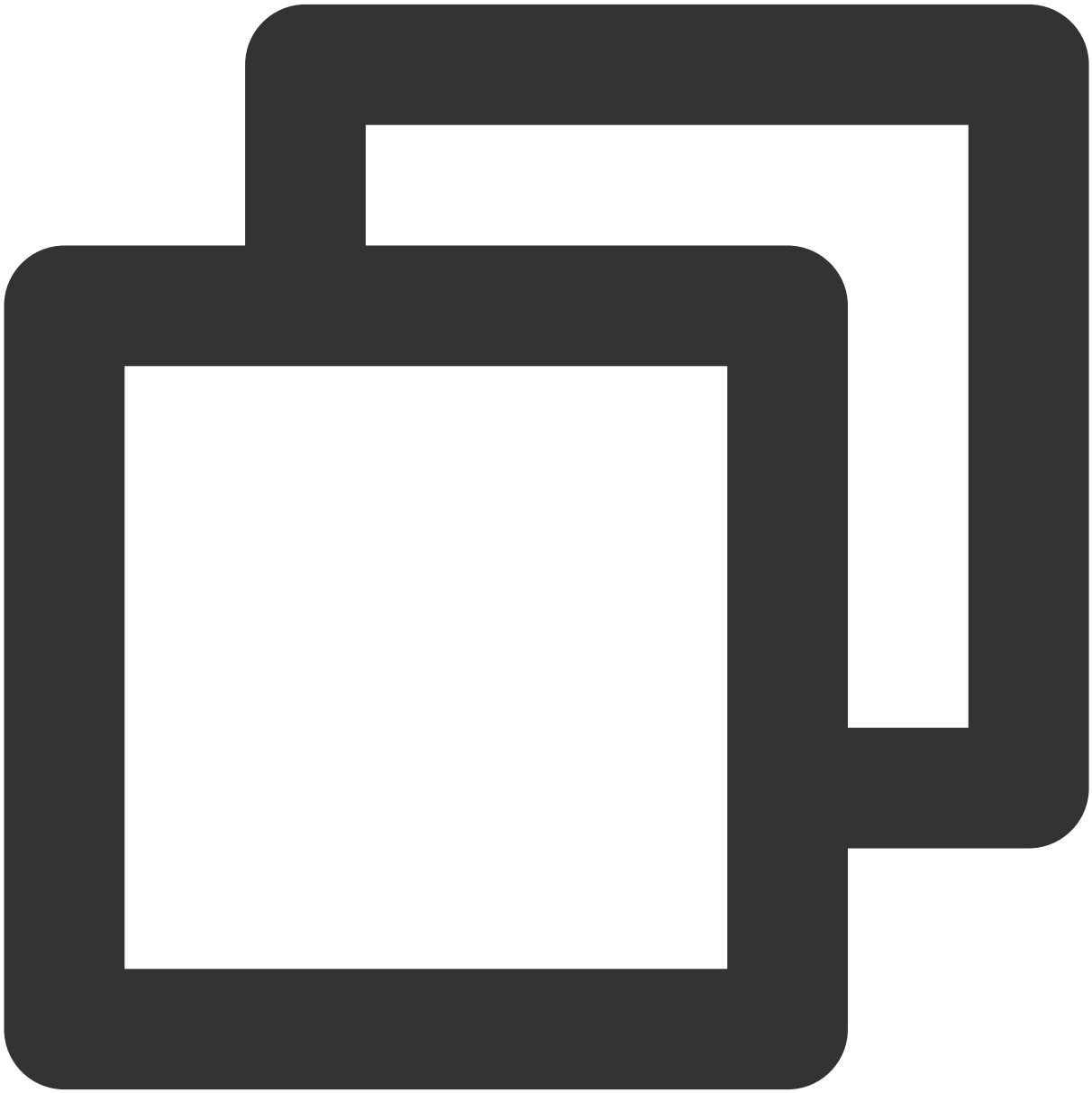
注意区分主实例和子实例的 userId，确保不重复且容易辨别。

伴奏实例播放背景音乐参数 musicParam 设置：

publish：YES（音乐在本地播放的同时，远端用户也能听到该音乐）

publish：NO（默认值，只能在本地听到该音乐，远端用户听不到）

4. 发起混流转码回推



```
// 创建 TRTCPublishTarget 对象
TRTCPublishTarget *publishTarget = [[TRTCPublishTarget alloc] init];
// 混流后回推到房间，若发布到 CDN 应填 TRTCPublishMixStreamToCdn
publishTarget.mode = TRTCPublishMixStreamToRoom;
// 混流机器人的 userid，不能和房间内其他用户的 userid 重复
publishTarget.mixStreamIdentity = [NSString stringWithFormat:@"%@%@",userId,@"_mix"

// 设置转码后的音频流的编码参数
TRTCStreamEncoderParam *streamEncoderParam = [[TRTCStreamEncoderParam alloc] init];
streamEncoderParam.videoEncodedFPS = 15;
streamEncoderParam.videoEncodedGOP = 3;
```

```
streamEncoderParam.videoEncodedKbps = 30;
streamEncoderParam.audioEncodedSampleRate = 48000;
streamEncoderParam.audioEncodedChannelNum = 2;
streamEncoderParam.audioEncodedKbps = 64;
streamEncoderParam.audioEncodedCodecType = 2;

// 设置音频混流参数
TRTCStreamMixingConfig *streamMixingConfig = [[TRTCStreamMixingConfig alloc] init];
// 支持填写空值，会自动将所有主播的音频混合输出
streamMixingConfig.audioMixUserList = @[];

// 发起混流转推请求
[trtcCloud startPublishMediaStream:publishTarget encoderParam:streamEncoderParam mi
```

注意：

优先选择主唱通过混流机器人向后台发起混流转推，将伴奏音乐和各路人声混合后回推至 TRTC 房间，或转推至直播 CDN。

自动订阅模式下，参与混流转码的主播默认互相拉取单流，不接收回推房间的混流；观众自动拉取回推房间的混流，不再接收单流。

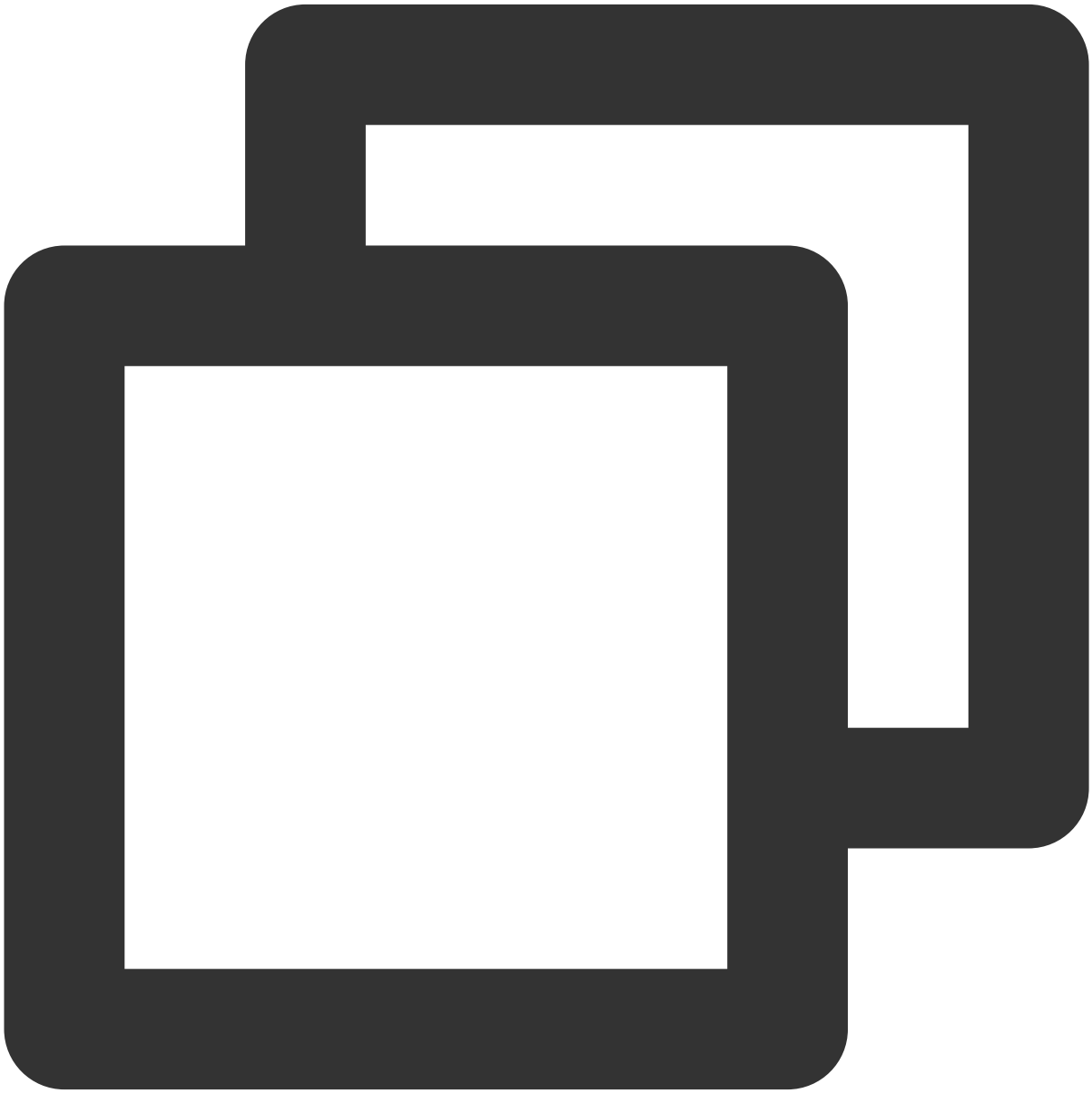
这里的混流转推方法 startPublishMediaStream 采用全新的后台架构，旧版应用需提供 SdkAppId 申请升级后方可使用。

Android

最近更新时间：2023-09-26 16:54:20

具体代码实现

1. 创建主子实例



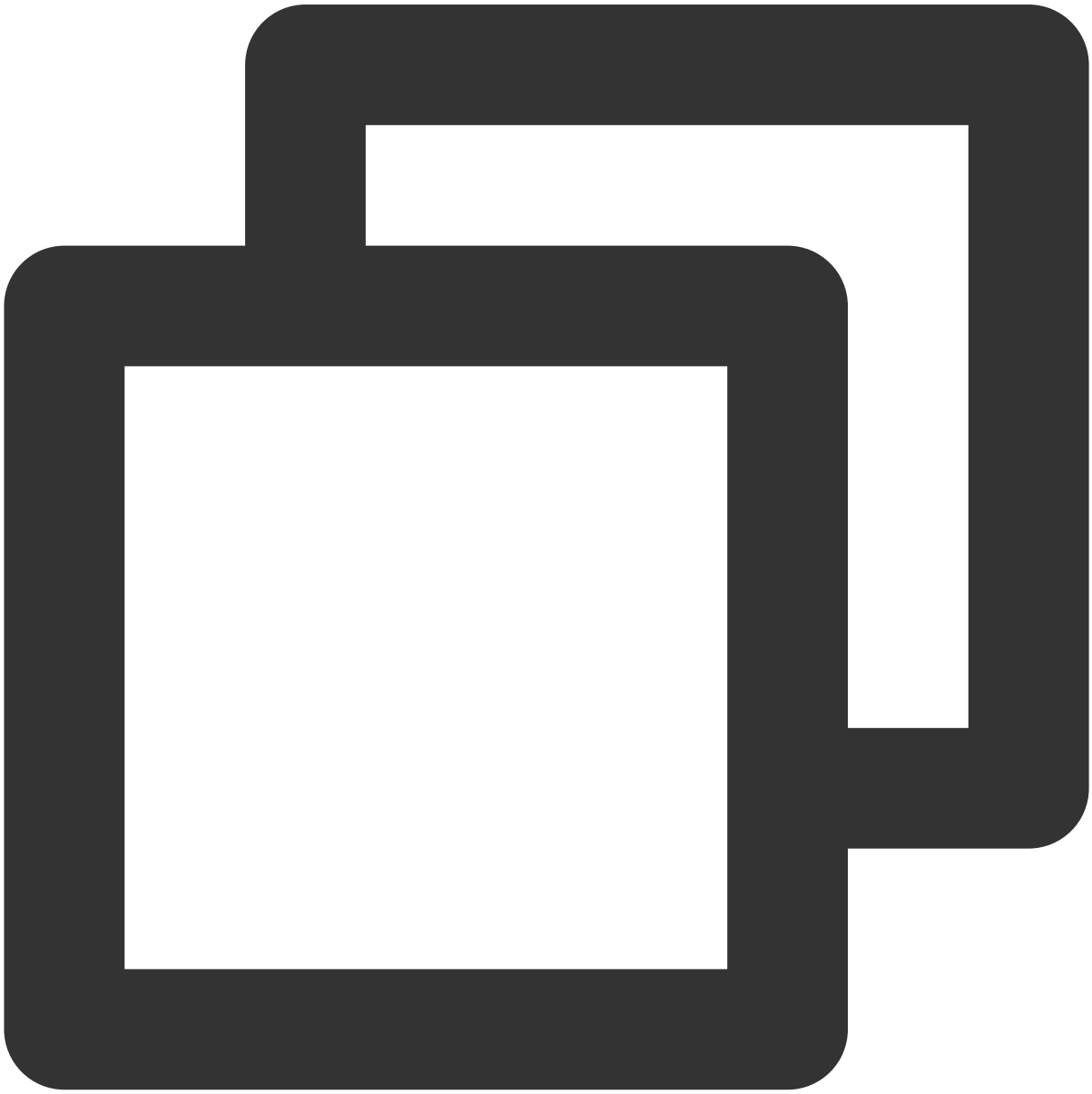
```
// 创建 TRTCCloud 主实例（人声实例）
```

```
TRTCCloud mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
// 创建 TRTCCloud 子实例（伴奏实例）
TRTCCloud subCloud = mTRTCCloud.createSubCloud();
```

说明：

实时合唱方案中，主唱端需要分别创建主实例-人声实例和子实例-伴奏实例，分别用于上行人声及伴奏音乐。

2. 人声实例进房推流



```
TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
params.sdkAppId = sdkAppId;
```

```
params.userId = mUserId;
params.userSig = userSig;
params.role = TRTCCloudDef.TRTCRoleAnchor;
params.roomId = mRoomId;
mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
// 打开音频上行，设置音质
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
// 设置媒体类型
mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
// 静音远端伴奏音乐
mTRTCCloud.muteRemoteAudio(mUserId + "_bgm", true);
```

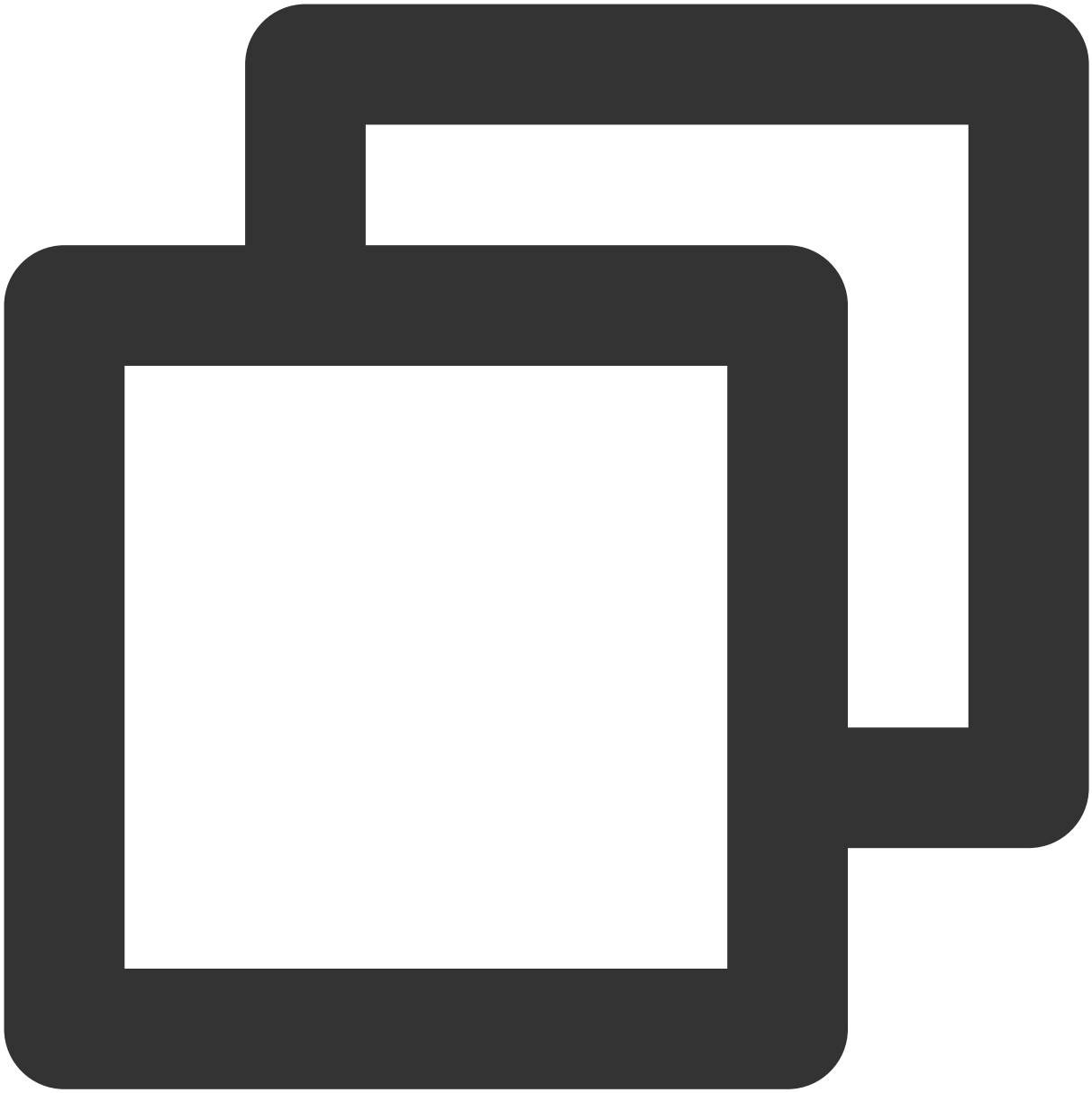
注意：

纯 RTC 音频场景下，进房场景推荐选用 VOICE_CHATROOM。

若有视频或转推 CDN 需求，进房场景则须选用 LIVE，VOICE_CHATROOM 会在转推时添加纯音频参数，从而导致 SEI 消息无法透传。

主唱/合唱端需要 muteRemoteAudio(true) 取消订阅伴奏实例上行的音频流，否则会重复播放本地及远端的伴奏音乐。

3. 伴奏实例进房推流



```
TRTCCloudDef.TRTCParams bgmParams = new TRTCCloudDef.TRTCParams();
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = mUserId + "_bgm";
bgmParams.userSig = userSig;
bgmParams.role = TRTCCloudDef.TRTCRoleAnchor;
bgmParams.roomId = mRoomId;
subCloud.enterRoom(bgmParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
//设置媒体类型
subCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);

// 开启预加载
```

```
subCloud.callExperimentalAPI("{\\"api\\":\\"preloadMusic\\",\\"params\\": {\\"music
// 播放伴奏音乐并推流（在约定时间播放）
TXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMusicPar
// 将伴奏音乐传到远端
param.publish = true;
subCloud.getAudioEffectManager().startPlayMusic(param);
```

注意：

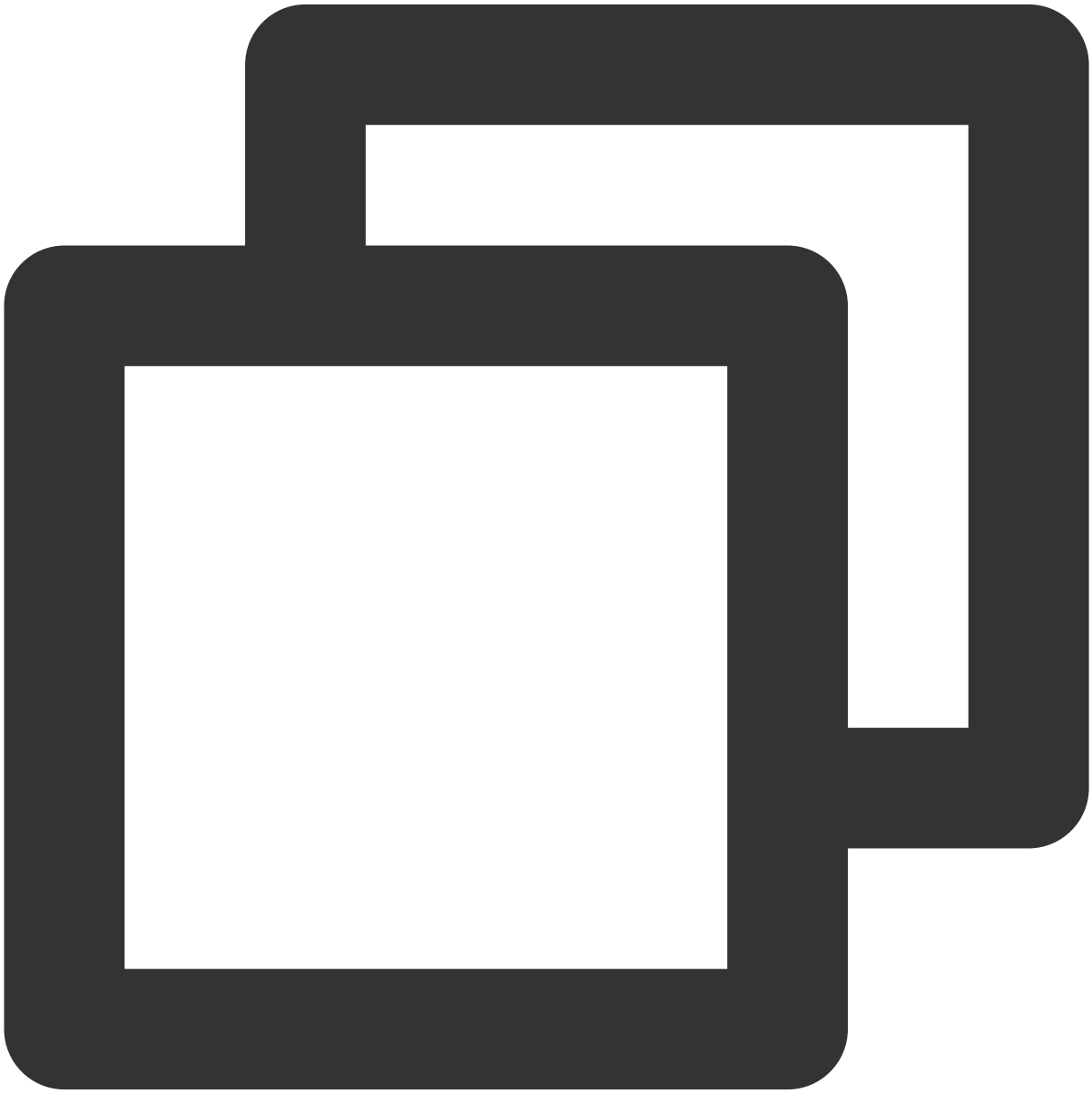
注意区分主实例和子实例的 userId，确保不重复且容易辨别；

伴奏实例播放背景音乐参数 AudioMusicParam 设置：

publish：true（音乐在本地播放的同时，远端用户也能听到该音乐）

publish：false（默认值，只能在本地听到该音乐，远端用户听不到）

4. 发起混流转码回推



```
// 创建 TRTCPublishTarget 对象
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
// 混流后回推到房间，若发布到 CDN 应填 TRTC_PublishMixStream_ToCdn
target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
target.mixStreamIdentity.intRoomId = Integer.parseInt(mRoomId);
// 混流机器人的 userid，不能和房间内其他用户的 userid 重复
target.mixStreamIdentity.userId = mUserId + "_mix";

// 设置转码后的音频流的编码参数
TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new TRTCCloudDef.TRTCS
trtcStreamEncoderParam.audioEncodedChannelNum = 2;
```

```
trtcStreamEncoderParam.audioEncodedKbps = 64;
trtcStreamEncoderParam.audioEncodedCodecType = 2;
trtcStreamEncoderParam.audioEncodedSampleRate = 48000;

// 设置音频混流参数
TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new TRTCCloudDef.TRTCS
// 支持填写空值，会自动将所有主播的音频混合输出
trtcStreamMixingConfig.audioMixUserList = null;

// 发起混流转推请求
mTRTCCloud.startPublishMediaStream(target, trtcStreamEncoderParam, trtcStreamMixing
```

注意：

优先选择主唱通过混流机器人向后台发起混流转推，将伴奏音乐和各路人声混合后回推至 TRTC 房间，或转推至直播 CDN。

自动订阅模式下，参与混流转码的主播默认互相拉取单流，不接收回推房间的混流；观众自动拉取回推房间的混流，不再接收单流。

这里的混流转推方法 startPublishMediaStream 采用全新的后台架构，旧版应用需提供 SdkAppId 申请升级后方可使用。

API 参考 （TUIKaraoke）
TRTCKaraoke(iOS)

最近更新时间：2022-07-22 15:13:44

TRTCKaraokeRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的 Karaoke 房间开播，听众进入 Karaoke 房间收听/互动。
- 房主可以管理点歌、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，上麦后可以点歌和唱歌，也可以随时下麦成为普通的听众。
- 支持发送礼物和各种文本、自定义消息，自定义消息可用于实现弹幕、点赞等。

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCKaraokeRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 Karaoke（iOS）。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCKaraokeRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateQueue	设置事件回调所在的线程。
login	登录。

API	描述
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 nil，则获取房间内所有人的信息。

音乐播放接口

API	描述
startPlayMusic	开始播放音乐。
stopPlayMusic	停止播放音乐。
pausePlayMusic	暂停播放音乐。
resumePlayMusic	恢复播放音乐。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述
sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCKaraokeRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	语聊房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

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

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

歌曲事件回调

API	描述
onMusicProgressUpdate	歌曲播放进度的回调。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
onMusicPrepareToPlay	准备播放音乐的回调。
onMusicCompletePlaying	播放完成音乐的回调。

SDK 基础函数

sharedInstance

获取 TRTCKaraokeRoom 单例对象。

```
/**
* 获取 TRTCKaraokeRoom 单例对象
*
* - returns: TRTCKaraokeRoom 实例
* - note: 可以调用 {@link TRTCKaraokeRoom#destroySharedInstance()} 销毁单例对象
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

销毁 TRTCKaraokeRoom 单例对象。

> 说明：
> 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
/**
* 销毁 TRTCKaraokeRoom 单例对象
*
* - note: 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例不能再使用，需要重新调用 {@link TRTC
KaraokeRoom#sharedInstance()} 获取新实例
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

TRTCKaraokeRoom 事件回调，您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知。

版权所有：腾讯云计算（北京）有限责任公司

```
/**
* 设置组件回调接口
*
* 您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知
*
* - parameter delegate 回调接口
* - note: TRTCKaraokeRoom 中的回调事件，默认是在 Main Queue 中回调给您；如果您需要指定事件回调所在的队列，可使用 {@link TRTCKaraokeRoom#setDelegateQueue(queue)}
*/
- (void)setDelegate:(id<TRTCKaraokeRoomDelegate>)delegate NS_SWIFT_NAME(setDelegate(delegate:));
```

说明：

setDelegate 是 TRTCKaraokeRoom 的代理回调。

setDelegateQueue

设置事件回调所在的线程队列，默认发送动主线程 MainQueue 中。

```
/**
* 设置事件回调所在的队列
*
* - parameter queue 队列，TRTCKaraokeRoom 中的各种状态通知回调，会派发到您指定的queue。
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue NS_SWIFT_NAME(setDelegateQueue(queue:));
```

参数如下表所示：

参数	类型	含义
queue	dispatch_queue_t	TRTCKaraokeRoom 中的各种状态通知，会派发到您指定的线程队列里去。

login

登录。

```
- (void)login:(int)sdkAppID
userId:(NSString *)userId
userSig:(NSString *)userSig
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(login(sdkAppID:userId:userSig:callback:));
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在实时音视频控制台 >【应用管理】> 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
- (void)logout:(ActionCallback _Nullable)callback NS_SWIFT_NAME(logout(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString *)avatarURL callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
- (void)createRoom:(int)roomID roomParam:(RoomParam *)roomParam callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个语聊房间列表，腾讯云暂不提供语聊房间列表的管理服务，请自行管理您的语聊房间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

房主开播的正常调用流程如下：

1. 房主调用 `createRoom` 创建新的 Karaoke 房间，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
- (void)destroyRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(destroyRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(enterRoom(roomID:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的 Karaoke 房间列表，可能包含多个语聊房间的 roomId 和房间信息。
2. 听众选择一个 Karaoke 房间，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

exitRoom

退出房间。

```
- (void)exitRoom:(ActionCallback _Nullable)callback NS_SWIFT_NAME(exitRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> 说明：
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:(KaraokeInfoCallback _Nullable)callback NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定 userId 的用户信息。

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList callback:(KaraokeUserListCallback _Nullable)callback NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

音乐播放接口

startPlayMusic

播放音乐（上麦后调用）。

说明：

- 播放音乐后，自身会收到 `onMusicPrepareToPlay` 的事件通知。
- 音乐播放中，房间内所有成员会不断收到 `onMusicProgressUpdate` 的事件通知。
- 音乐播放完成，自身会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)startPlayMusic:(int32_t)musicID originalUrl:(NSString *)originalUrl accompanyUrl:(NSString *)backingUrl NS_SWIFT_NAME(startPlayMusic(musicID:originalUrl:a
```

```
ccompanyUrl:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	音乐的 ID。
originalUrl	String	原唱音乐的绝对路径。
accompanyUrl	String	伴奏音乐的绝对路径。

调用该接口后会停止上一个正在播放的歌曲。

stopPlayMusic

停止播放音乐（播放音乐时调用）。

> 说明：
> 停止播放后，会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)stopPlayMusic NS_SWIFT_NAME(stopPlayMusic());
```

pausePlayMusic

暂停正在播放的音乐（播放音乐时调用）。

> 说明：
>
> - `onMusicProgressUpdate` 的事件通知会暂停
> - 不会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)pausePlayMusic NS_SWIFT_NAME(pausePlayMusic());
```

resumePlayMusic

恢复暂停过的音乐（暂停后调用）。

> 说明：
> 不会收到 `onMusicPrepareToPlay` 的事件通知。

```
- (void)resumePlayMusic NS_SWIFT_NAME(resumePlayMusic());
```

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> 说明：
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callbac
k NS_SWIFT_NAME(enterSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> 说明：
> 下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)leaveSeat:(ActionCallback _Nullable)callback NS_SWIFT_NAME(leaveSeat(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> 说明：
> 房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(pickSeat(seatIndex:userId:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要抱上麦的麦位序号。
userId	String	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> 说明：
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(kickSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> 说明：
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:(ActionCallbac
k _Nullable)callback NS_SWIFT_NAME(muteSeat(seatIndex:isMute:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isMute	boolean	true：静音对应麦位；false：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> 说明：

房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(closeSeat(seatIndex:isClose:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isClose	boolean	true：封禁对应麦位； false：解封对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
- (void)startMicrophone;
```

stopMicrophone

停止麦克风采集。

```
- (void)stopMicrophone;
```

setAudioQuality

设置音质。

```
- (void)setAuidoQuality:(NSInteger)quality NS_SWIFT_NAME(setAuidoQuality(quality:));
```

参数如下表所示：

参数	类型	含义
quality	int	音频质量，详情请参见 TRTC SDK。

muteLocalAudio

静音/取消静音本地的音频。

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

参数如下表所示：

参数	类型	含义
mute	boolean	静音/取消静音，详情请参见 TRTC SDK。

setSpeaker

设置开启扬声器。

```
- (void)setSpeaker:(BOOL)userSpeaker NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

参数如下表所示：

参数	类型	含义
useSpeaker	boolean	true：扬声器；false：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
- (void)setAudioCaptureVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioCaptureVolume(volume:));
```

参数如下表所示：

参数	类型	含义
volume	int	采集音量，0 - 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

版权所有：腾讯云计算（北京）有限责任公司

```
- (void)setAudioPlayoutVolume:(NSInteger)volume NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

参数如下表所示：

参数	类型	含义
volume	int	播放音量，0 - 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute NS_SWIFT_NAME(muteRemoteAudio(userId:mute:));
```

参数如下表所示：

参数	类型	含义
userId	String	指定的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音/解除静音所有成员。

```
- (void)muteAllRemoteAudio:(BOOL)isMute NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

参数如下表所示：

参数	类型	含义
isMute	boolean	true：开启静音；false：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
- (void)setVoiceEarMonitorEnable:(BOOL)enable NS_SWIFT_NAME(setVoiceEarMonitor(enable:));
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
enable	boolean	true：开启耳返；false：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
- (NSString *)sendInvitation:(NSString *)cmd
userId:(NSString *)userId
content:(NSString *)content
callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(sendInvitation(cmd:user
Id:content:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	String	业务自定义指令。
userId	String	邀请的用户 ID。
content	String	邀请的内容。
callback	ActionCallback	发送结果回调。

返回值：

返回值	类型	含义
inviteId	String	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback _Nullabl
e)callback NS_SWIFT_NAME(acceptInvitation(identifier:callback:));
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(rejectInvitation(identifier:callback:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback _Nullable)callback NS_SWIFT_NAME(cancelInvitation(identifier:callback:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCKaraokeRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> 说明：
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
- (void)onError:(int)code
message:(NSString*)message
NS_SWIFT_NAME(onError(code:message:));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。

```
- (void)onWarning:(int)code
message:(NSString *)message
NS_SWIFT_NAME(onWarning(code:message:));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	警告信息。

onDebugLog

Log 回调。

```
- (void)onDebugLog:(NSString *)message
NS_SWIFT_NAME(onDebugLog(message:));
```

参数如下表所示：

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
- (void)onRoomDestroy:(NSString *)message
NS_SWIFT_NAME(onRoomDestroy(message:));
```

参数如下表所示：

参数	类型	含义
message	String	回调信息。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
- (void)onRoomInfoChange:(KaraokeInfo *)roomInfo
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

参数如下表所示：

参数	类型	含义
roomInfo	RoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用muteLocalAudio，房间内的其他用户都会收到此通知。

```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

参数如下表所示：

参数	类型	含义

参数	类型	含义
userId	String	用户 ID。
mute	boolean	音量大小，取值：0 - 100。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes totalVolume:(NSInteger)totalVolume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

参数如下表所示：

参数	类型	含义
userVolumes	List	用户列表。
totalVolume	int	音量大小，取值：0 - 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
- (void)onSeatInfoChange:(NSArray<KaraokeSeatInfo *> *)seatInfolist
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

参数如下表所示：

参数	类型	含义
seatInfoList	List<SeatInfo>	全量的麦位列表。

onAnchorEnterSeat

有成员上麦(主动上麦/房主抱人上麦)。

```
- (void)onAnchorEnterSeat:(NSInteger)index
user:(KaraokeUserInfo *)user
```

```
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	int	成员上麦的麦位。
user	UserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
- (void)onAnchorLeaveSeat:(NSInteger)index
user:(KaraokeUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	int	下麦的麦位。
user	UserInfo	上麦用户的详细信息。

onSeatMute

房主禁麦。

```
- (void)onSeatMute:(NSInteger)index
isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(index:isMute:));
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isMute	boolean	true：静音麦位； false：解除静音。

onSeatClose

房主封麦。

```
- (void)onSeatClose:(NSInteger)index
isClose:(BOOL)isClose
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isClose	boolean	true：封禁麦位； false： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
- (void)onAudienceEnter:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
- (void)onAudienceExit:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
- (void)onRecvRoomTextMsg:(NSString *)message
userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
- (void)onRecvRoomCustomMsg:(NSString *)cmd
message:(NSString *)message
userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(cmd:message:userInfo:));
```

参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
- (void)onReceiveNewInvitation:(NSString *)identifier
inviter:(NSString *)inviter
cmd:(NSString *)cmd
```

```
content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(identifier:inviter:cmd:content:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
cmd	String	业务指定的命令字，由开发者自定义。
content	UserInfo	业务指定的内容。

onInviteeAccepted

被邀请者接受邀请。

```
- (void)onInviteeAccepted:(NSString *)identifier
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(identifier:invitee:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
- (void)onInviteeRejected:(NSString *)identifier
invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(identifier:invitee:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

版权所有：腾讯云计算（北京）有限责任公司

onInvitationCancelled

邀请人取消邀请。

```
- (void)onInvitationCancelled:(NSString *)identifier
invitee:(NSString *)invitee NS_SWIFT_NAME(onInvitationCancelled(identifier:invite
e:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	邀请人的用户 ID。

音乐播放状态回调

onMusicPrepareToPlay

准备播放音乐的回调

```
- (void)onMusicPrepareToPlay:(int32_t)musicID
NS_SWIFT_NAME(onMusicPrepareToPlay(musicID:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

onMusicProgressUpdate

歌曲播放进度的回调

```
- (void)onMusicProgressUpdate:(int32_t)musicID
progress:(NSInteger)progress total:(NSInteger)total
NS_SWIFT_NAME(onMusicProgressUpdate(musicID:progress:total:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
progress	NSInteger	当前播放时间，单位：ms。
total	NSInteger	总时间，单位：ms。

onMusicCompletePlaying

播放完成音乐的回调

```
- (void)onMusicCompletePlaying:(int32_t)musicID
NS_SWIFT_NAME(onMusicCompletePlaying(musicID:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

TRTCKaraoke(Android)

最近更新时间：2022-07-22 15:13:44

TRTCKaraokeRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的 Karaoke 房间开播，听众进入 Karaoke 房间收听/互动。
- 房主可以管理点歌、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，上麦后可以点歌和唱歌，也可以随时下麦成为普通的听众。
- 支持发送礼物和各种文本、自定义消息，自定义消息可用于实现弹幕、点赞等。

> 说明：
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 - 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCKaraokeRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 Karaoke（Android）。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCKaraokeRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateHandler	设置事件回调所在的线程。
login	登录。

API	描述
logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 null，则获取房间内所有人的信息。

音乐播放接口

API	描述
startPlayMusic	开始播放音乐。
stopPlayMusic	停止播放音乐。
pausePlayMusic	暂停播放音乐。
resumePlayMusic	恢复播放音乐。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述
sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCKaraokeRoomDelegate API 概览

通用事件回调

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

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	Karaoke 房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。

版权所有：腾讯云计算（北京）有限责任公司

API	描述
onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

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

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

歌曲事件回调

API	描述

版权所有：腾讯云计算（北京）有限责任公司

API	描述
onMusicProgressUpdate	歌曲播放进度的回调。
onMusicPrepareToPlay	准备播放音乐的回调。
onMusicCompletePlaying	播放完成音乐的回调。

SDK 基础函数

sharedInstance

获取 TRTCKaraokeRoom 单例对象。

```
public static synchronized TRTCKaraokeRoom sharedInstance(Context context);
```

参数如下表所示：

参数	类型	含义
context	Context	Android 上下文，内部会转为 ApplicationContext 用于系统 API 调用

destroySharedInstance

销毁 TRTCKaraokeRoom 单例对象。

> 说明：
> 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
public static void destroySharedInstance();
```

setDelegate

TRTCKaraokeRoom 事件回调，您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知。

```
public abstract void setDelegate(TRTCKaraokeRoomDelegate delegate);
```

版权所有：腾讯云计算（北京）有限责任公司

说明：
setDelegate 是 TRTCKaraokeRoom 的代理回调。

setDelegateHandler

设置事件回调所在的线程。

```
public abstract void setDelegateHandler(Handler handler);
```

参数如下表所示：

参数	类型	含义
handler	Handler	TRTCKaraokeRoom 中的各种状态通知，会派发到您指定的 handler 线程。

login

登录。

```
public abstract void login(int sdkAppId,
String userId, String userSig,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在实时音视频控制台 >【应用管理】> 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
public abstract void logout(TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
public abstract void setSelfProfile(String userName, String avatarURL, TRTCKaraok
eRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
public abstract void createRoom(int roomId, TRTCKaraokeRoomDef.RoomParam roomPara
m, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个 Karaoke 房间列表，腾讯云暂不提供 Karaoke 房间列表的管理服务，请自行管理您的 Karaoke 房间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

版权所有：腾讯云计算（北京）有限责任公司

房主开播的正常调用流程如下：

1. 房主调用 `createRoom` 创建新的 Karaoke 房间，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
public abstract void destroyRoom(TRTCKaraokeRoomCallback.ActionCallback callback)
;
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
public abstract void enterRoom(int roomId, TRTCKaraokeRoomCallback.ActionCallback
callback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的 Karaoke 房间列表，可能包含多个 Karaoke 房间的 roomId 和房间信息。
2. 听众选择一个 Karaoke 房间，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。

版权所有：腾讯云计算（北京）有限责任公司

5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

exitRoom

退出房间。

```
public abstract void exitRoom(TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> 说明：
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
public abstract void getRoomInfoList(List<Integer> roomIdList, TRTCKaraokeRoomCallback.RoomInfoCallback callback);
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定userId的用户信息。

```
public abstract void getUserInfoList(List<String> userIdList, TRTCKaraokeRoomCallback.UserListCallback userlistcallback);
```

参数如下表所示：

参数	类型	含义

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
userIdList	List<String>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

音乐播放接口

startPlayMusic

播放音乐（上麦后调用）。

> 说明：
>
> - 播放音乐后，自身会收到 `onMusicPrepareToPlay` 的事件通知。
> - 音乐播放中，房间内所有成员会不断收到 `onMusicProgressUpdate` 的事件通知。
> - 音乐播放完成，自身会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void startPlayMusic(int musicID, String originalUrl, String accompanyUrl);
```

参数如下表所示：

参数	类型	含义
musicID	int	音乐的 ID。
originalUrl	String	原唱音乐的绝对路径。
accompanyUrl	String	伴奏音乐的绝对路径。

调用该接口后会停止上一个正在播放的歌曲。

stopPlayMusic

停止播放音乐（播放音乐时调用）。

> 说明：
> 停止播放后，会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void stopPlayMusic();
```

pausePlayMusic

暂停正在播放的音乐（播放音乐时调用）。

> 说明：
>
> - `onMusicProgressUpdate` 的事件通知会暂停
> - 不会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void pausePlayMusic();
```

resumePlayMusic

恢复暂停过的音乐（暂停后调用）。

> 说明：
> 不会收到 `onMusicPrepareToPlay` 的事件通知。

```
public abstract void resumePlayMusic();
```

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> 说明：
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
public abstract void enterSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> 说明：
> 下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
public abstract void leaveSeat(TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> 说明：
> 房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
public abstract void pickSeat(int seatIndex, String userId, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要抱上麦的麦位序号。
userId	String	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> 说明：
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
public abstract void kickSeat(int seatIndex, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> 说明：
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
public abstract void muteSeat(int seatIndex, boolean isMute, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isMute	boolean	true：静音对应麦位；false：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> 说明：
> 房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

```
public abstract void closeSeat(int seatIndex, boolean isClose, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isClose	boolean	true：封禁对应麦位； false：解封对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
public abstract void startMicrophone();
```

stopMicrophone

停止麦克风采集。

```
public abstract void stopMicrophone();
```

setAudioQuality

设置音质。

```
public abstract void setAudioQuality(int quality);
```

参数如下表所示：

参数	类型	含义
quality	int	音频质量，详情请参见 TRTC SDK。

muteLocalAudio

静音/取消静音本地的音频。

```
public abstract void muteLocalAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	静音/取消静音，详情请参见 TRTC SDK。

setSpeaker

设置开启扬声器。

```
public abstract void setSpeaker(boolean useSpeaker);
```

参数如下表所示：

参数	类型	含义
useSpeaker	boolean	true：扬声器；false：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
public abstract void setAudioCaptureVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	采集音量，0 - 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

```
public abstract void setAudioPlayoutVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	播放音量，0 - 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	指定的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

版权所有：腾讯云计算（北京）有限责任公司

静音/解除静音所有成员。

```
public abstract void muteAllRemoteAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	true：开启静音；false：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

参数如下表所示：

参数	类型	含义
enable	boolean	true：开启耳返；false：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager。

```
public abstract TXAudioEffectManager getAudioEffectManager();
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
public abstract void sendRoomTextMsg(String message, TRTCKaraokeRoomCallback.Acti
onCallback callback);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
public abstract void sendRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
public abstract String sendInvitation(String cmd, String userId, String content, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	业务自定义指令。
userId	String	邀请的用户 ID。
content	String	邀请的内容。
callback	ActionCallback	发送结果回调。

版权所有：腾讯云计算（北京）有限责任公司

返回值：

返回值	类型	含义
inviteId	String	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
public abstract void acceptInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

```
public abstract void rejectInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
public abstract void cancelInvitation(String id, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCKaraokeRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> 说明：
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
void onError(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。

```
void onWarning(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。

参数	类型	含义
message	String	警告信息。

onDebugLog

Log 回调。

```
void onDebugLog(String message);
```

参数如下表所示：

参数	类型	含义
message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
void onRoomDestroy(String roomId);
```

参数如下表所示：

参数	类型	含义
roomId	String	房间 ID。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
void onRoomInfoChange(TRTCKaraokeRoomDef.RoomInfo roomInfo);
```

参数如下表所示：

参数	类型	含义
roomInfo	RoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用 muteLocalAudio，房间内的其他用户都会收到此通知。

```
void onUserMicrophoneMute(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID。
mute	boolean	音量大小，取值：0 - 100。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes, int totalVolume);
```

参数如下表所示：

参数	类型	含义
userVolumes	List	用户列表。
totalVolume	int	音量大小，取值：0 - 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

参数如下表所示：

参数	类型	含义
seatInfoList	List<SeatInfo>	全量的麦位列表。

onAnchorEnterSeat

版权所有：腾讯云计算（北京）有限责任公司

有成员上麦(主动上麦/房主抱人上麦)。

```
void onAnchorEnterSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	成员上麦的麦位。
user	UserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
void onAnchorLeaveSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	下麦的麦位。
user	UserInfo	下麦用户的详细信息。

onSeatMute

房主禁麦。

```
void onSeatMute(int index, boolean isMute);
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isMute	boolean	true：静音麦位； false：解除静音。

onSeatClose

房主封麦。

```
void onSeatClose(int index, boolean isClose);
```

版权所有：腾讯云计算（北京）有限责任公司

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isClose	boolean	true：封禁麦位； false： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
void onAudienceEnter(TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
void onAudienceExit(TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
void onRecvRoomCustomMsg(String cmd, String message, TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
void onReceiveNewInvitation(String id, String inviter, String cmd, String content);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
cmd	String	业务指定的命令字，由开发者自定义。
content	String	业务指定的内容。

版权所有：腾讯云计算（北京）有限责任公司

onInviteeAccepted

被邀请者接受邀请。

```
void onInviteeAccepted(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
void onInviteeRejected(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInvitationCancelled

邀请人取消邀请。

```
void onInvitationCancelled(String id, String inviter);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。

音乐播放状态回调

onMusicPrepareToPlay

准备播放音乐的回调

```
void onMusicPrepareToPlay(int musicID);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。

onMusicProgressUpdate

歌曲播放进度的回调

```
void onMusicProgressUpdate(int musicID, long progress, long total);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。
progress	long	当前播放时间，单位：ms。
total	long	总时间，单位：ms。

onMusicCompletePlaying

播放完成音乐的回调

```
void onMusicCompletePlaying(int musicID);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。

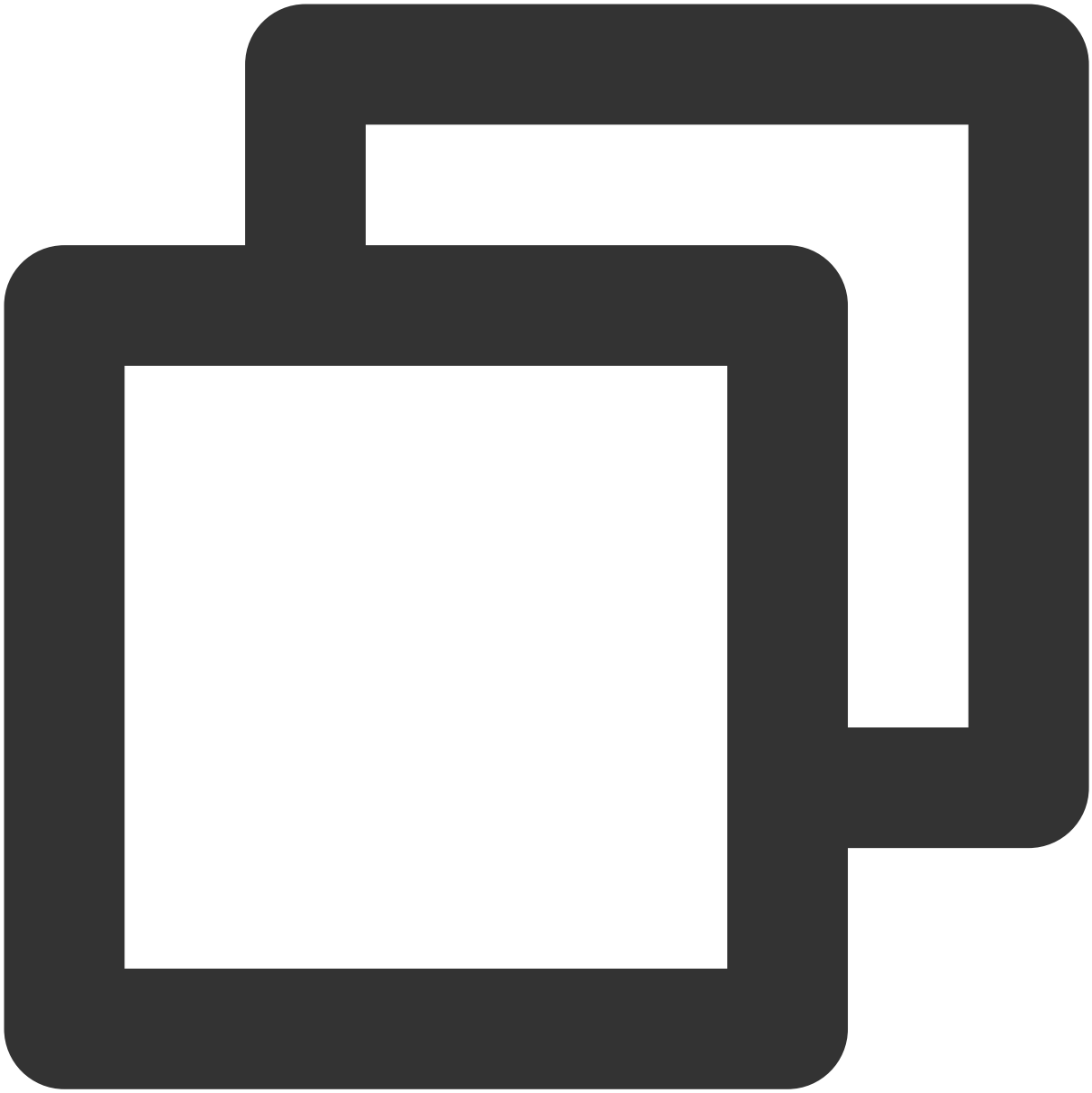
版权所有：腾讯云计算（北京）有限责任公司

常见问题（TUIKaraoke）
iOS

最近更新时间：2023-09-26 17:01:20

耳返相关问题

K 歌场景大概率会用到耳返，如何开启耳返功能？



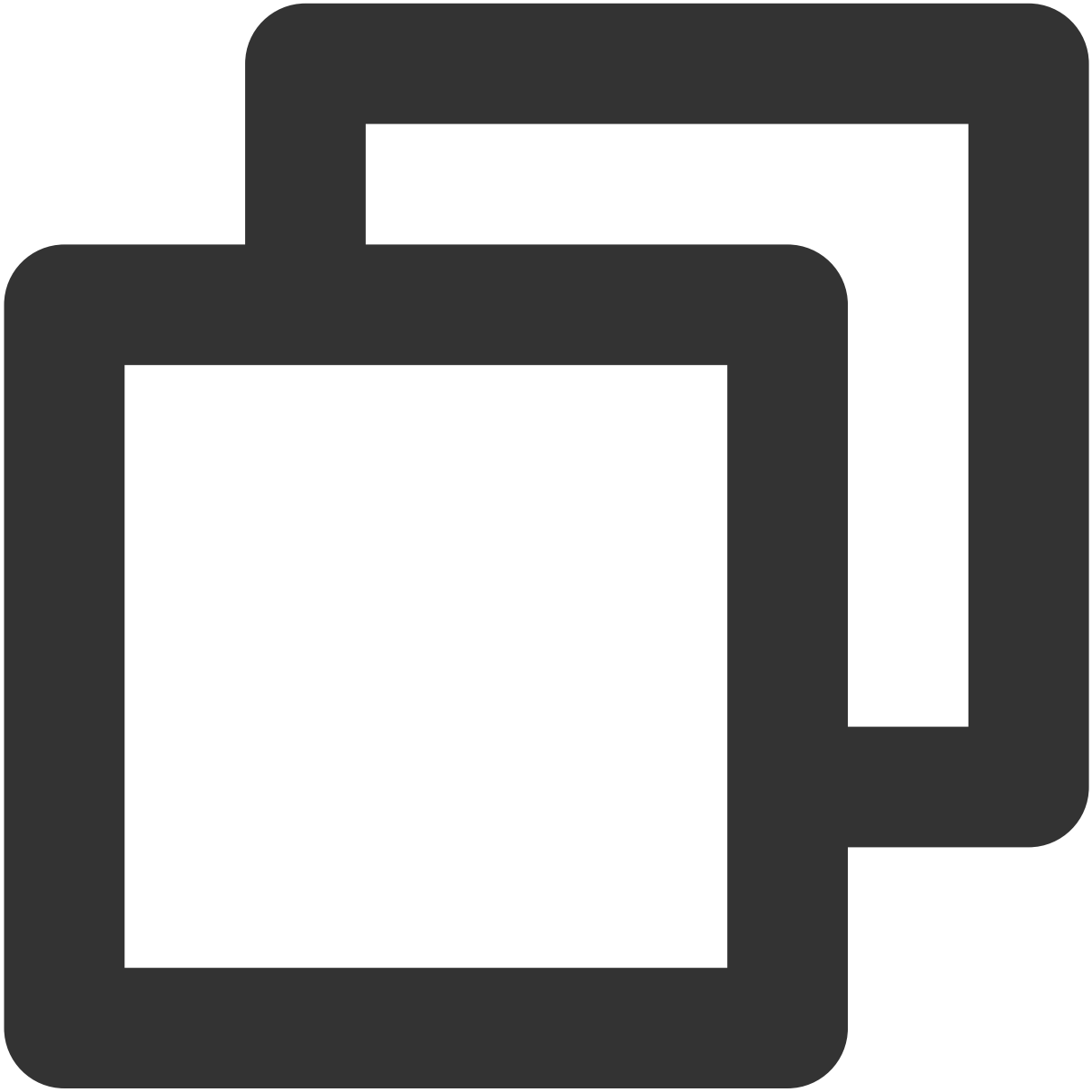
```
[[trtcCloud getAudioEffectManager] enableVoiceEarMonitor:YES];
```

开启耳返功能后没有效果？

由于蓝牙耳机的硬件延迟非常高，请尽量在用户界面上提示主播佩戴有线耳机。 同时也需要注意，并非所有的手机开启此特效后都能达到优秀的耳返效果，TRTC SDK 已经对部分耳返效果不佳的手机屏蔽了该特效。

耳返延迟过高？

请检查是否使用的是蓝牙耳机，由于蓝牙耳机的硬件延迟非常高，请尽量使用有线耳机。
另外，可以尝试通过实验性接口 setSystemAudioKitEnabled 开启硬件耳返来改善耳返延迟过高的问题。目前，对于华为和 Vivo设备，SDK 默认使用硬件耳返，其他设备默认使用软件耳返。



```
// 开启硬件耳返
NSDictionary *jsonDic = @{
                            @"api": @"setSystemAudioKitEnabled",
                            @"params": @{@"enable": @(1)}
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic options:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
```

```
[trtcCloud callExperimentalAPI:jsonString];
```

NTP 校时问题

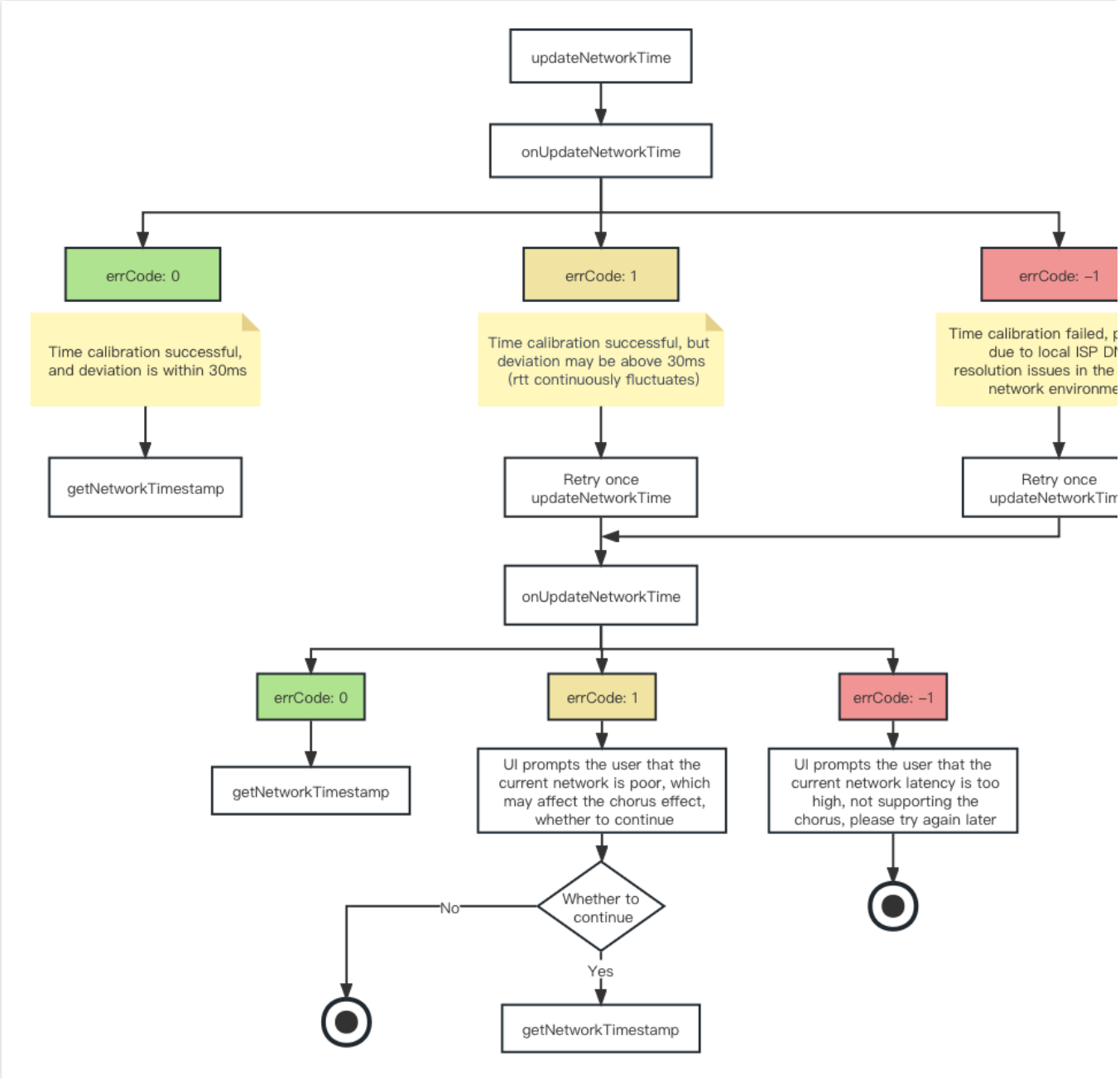
提醒：“NTP time sync finished, but result maybe inaccurate”？

NTP 校时成功，但偏差可能在30ms以上，反映客户端网络环境差，rtt 持续抖动。

提醒：“Error in AddressResolver: No address associated with hostname”？

NTP 校时失败，可能是当前网络环境下本地运营商 DNS 解析暂时异常，请稍后再试。

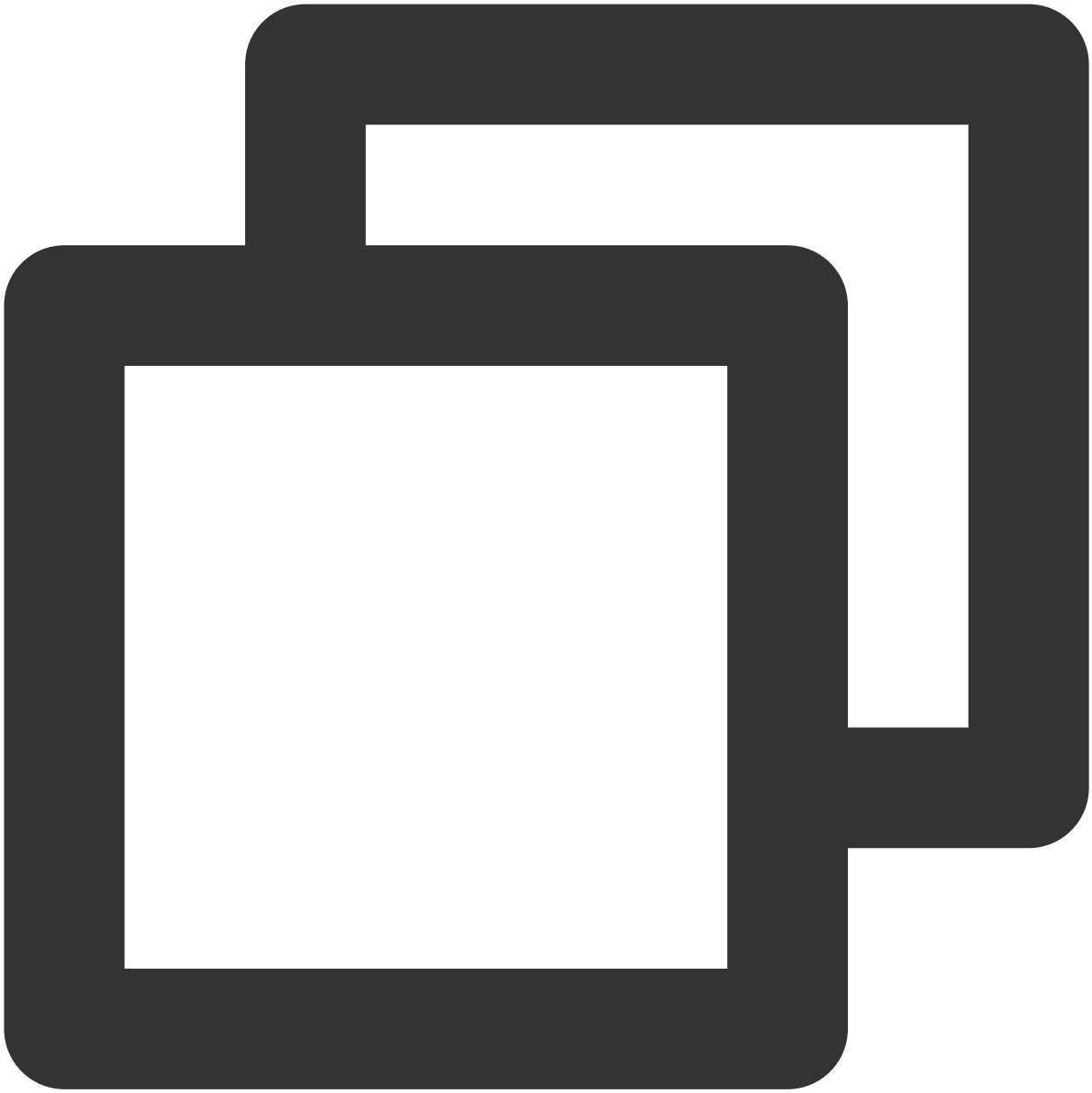
NTP 服务重试处理逻辑？



网络测速建议

在线 K 歌场景对用户的网络条件要求较高，特别是实时合唱，优质稳定的网络环境才能保障良好的 K 歌体验。因此，建议在用户进入房间前对该用户进行一次网络测速，对网络条件不符合要求的用户给予 UI 层的提醒，禁止其加入 K 歌房或参与合唱。

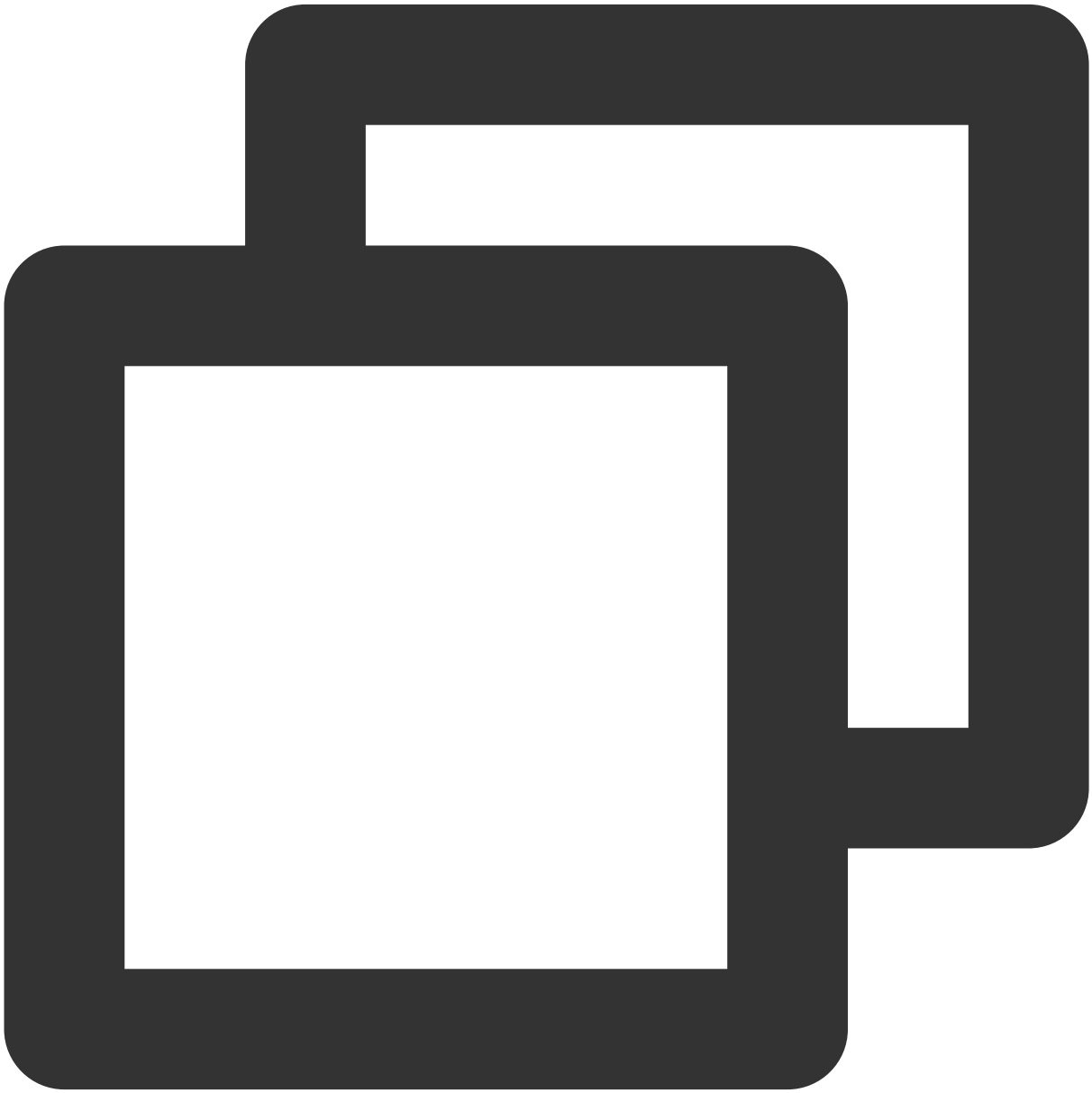
TRTC SDK 网络测速的发起：



```
TRTCSpeedTestParams *speedTestParams = [[TRTCSpeedTestParams alloc] init];
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// 若实际带宽高于预期值，则测试结果即为预期值；若实际带宽低于预期值，则测试结果为实际带宽值
speedTestParams.expectedDownBandwidth = 3000; // 预期的下行带宽，取值范围 10~5000 kbps
speedTestParams.expectedUpBandwidth = 3000; // 预期的上行带宽，取值范围 10~5000 kbps
[trtcCloud startSpeedTest:speedTestParams];
```

注意：

同一时间只允许一项网速测试任务运行；
请在进入房间前进行网速测试，在房间中网速测试会影响正常的音视频传输效果，而且由于干扰过多，网速测试结果也不准确。
TRTC SDK 网络测速结果的回调：



```
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    NSString *tquality = @"未定义";
    switch (result.quality) {
        case TRTCQuality_Unknown:
            tquality = @"未定义";
            break;
```

```
        case TRTCQuality_Excellent:
            tquality = @"当前网络非常好";
            break;
        case TRTCQuality_Good:
            tquality = @"当前网络比较好";
            break;
        case TRTCQuality_Poor:
            tquality = @"当前网络一般";
            break;
        case TRTCQuality_Bad:
            tquality = @"当前网络较差";
            break;
        case TRTCQuality_Vbad:
            tquality = @"当前网络很差";
            break;
        case TRTCQuality_Down:
            tquality = @"当前网络不满足 TRTC 的最低要求";
            break;
        default:
            break;
    }

    if (result.success) {
        [mTextTestResult addObject:@"测速成功！\\n"];
        [mTextTestResult addObject:[NSString stringWithFormat:@"IP 地址：%@ \\n", re
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行丢包率：%f \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行丢包率：%f \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"网络延迟：%u ms \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽：%ld kbps \\
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行带宽：%ld kbps \\
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽：%@ \\n", tq
    } else {
        [mTextTestResult addObject:@"测速成功！\\n"];
        [mTextTestResult addObject:[NSString stringWithFormat:@"errMsg：%@ \\n", res
    }
}
```

中途加入合唱

实时合唱方案理论上对合唱者数量没有限制，支持多人同时参与合唱，也支持中途加入合唱。

下面列出中途加入合唱的关键动作：

NTP 校时

开启合唱实验性接口

进入房间并上麦推流
接收合唱信令，获取伴奏资源及合唱约定时间
计算约定时间与当前时间差，预加载 seek 伴奏
开始参与合唱，并实时同步伴奏进度及歌词进度

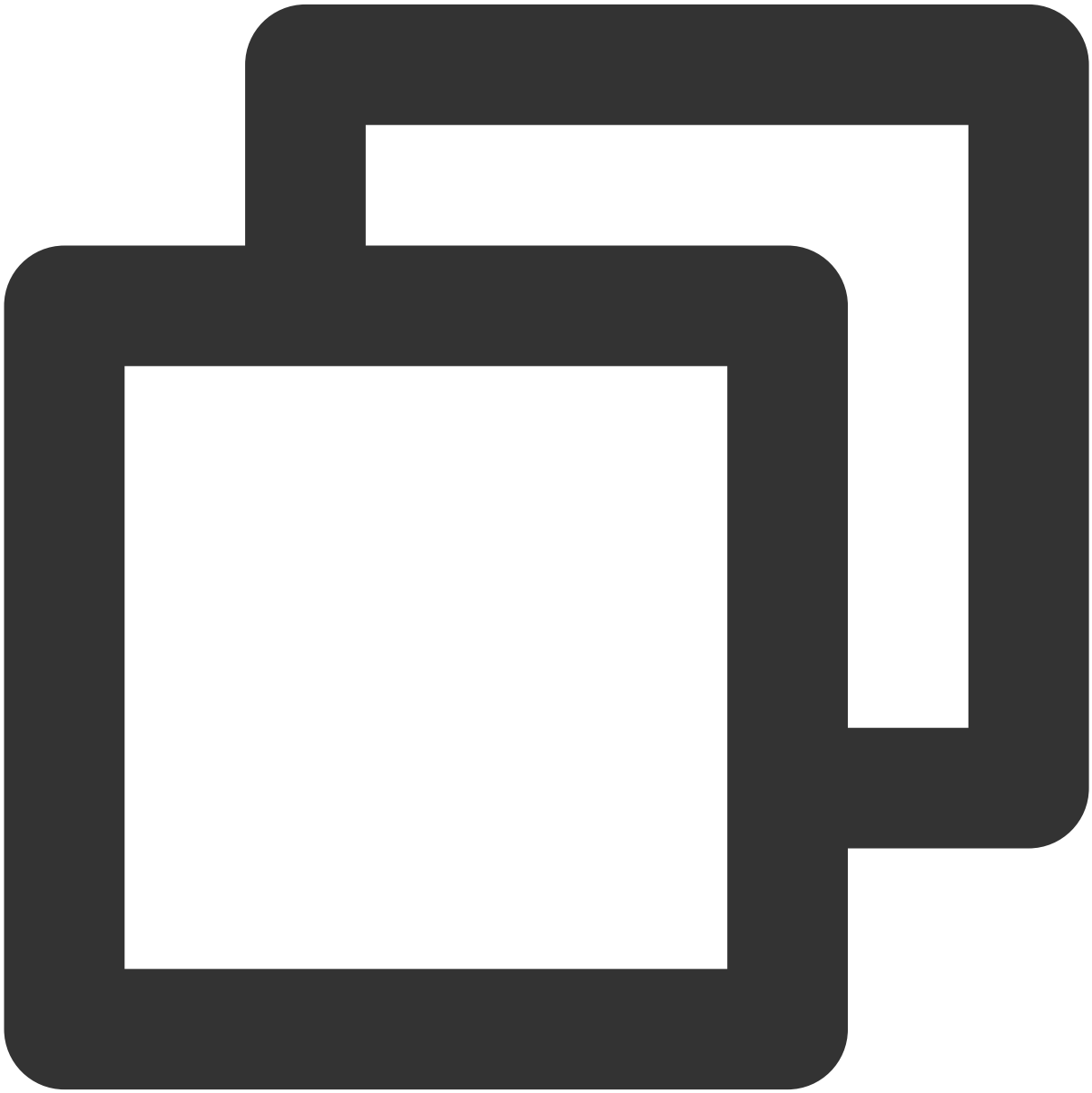
以上关键动作涉及的具体实现流程及代码实现详见歌曲同步、歌词同步、人声同步文档。

Android

最近更新时间：2023-09-27 11:28:24

耳返相关问题

K 歌场景大概率会用到耳返，如何开启耳返功能？



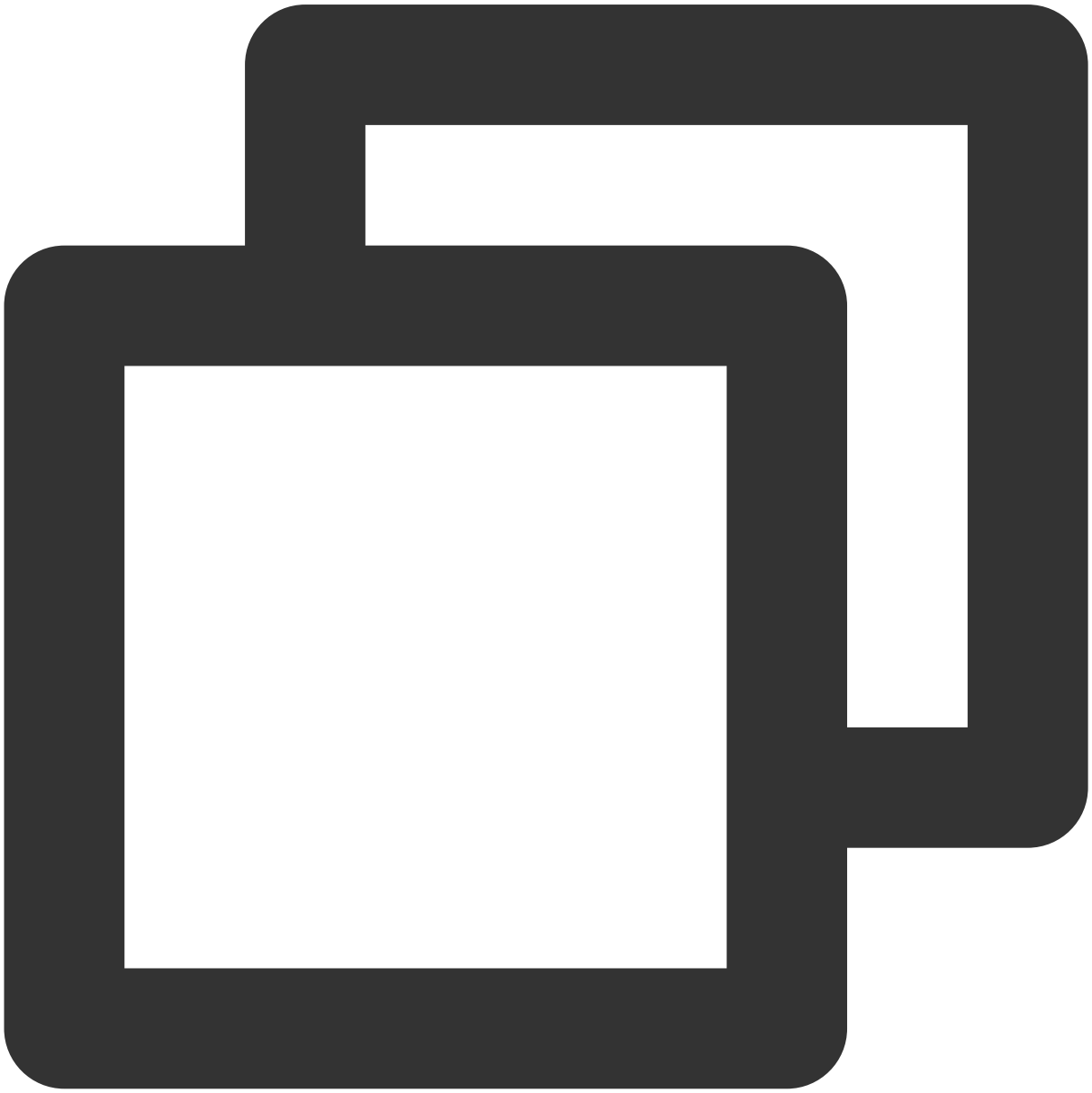
```
mTRTCCloud.getAudioEffectManager().enableVoiceEarMonitor(true)
```

开启耳返功能后没有效果？

由于蓝牙耳机的硬件延迟非常高，请尽量在用户界面上提示主播佩戴有线耳机。 同时也需要注意，并非所有的手机开启此特效后都能达到优秀的耳返效果，TRTC SDK 已经对部分耳返效果不佳的手机屏蔽了该特效。

耳返延迟过高？

请检查是否使用的是蓝牙耳机，由于蓝牙耳机的硬件延迟非常高，请尽量使用有线耳机。
另外，可以尝试通过实验性接口 setSystemAudioKitEnabled 开启硬件耳返来改善耳返延迟过高的问题。目前，对于华为和 Vivo设备，SDK 默认使用硬件耳返，其他设备默认使用软件耳返。



版权所有：腾讯云计算（北京）有限责任公司

```
// 开启硬件耳返
mTRTCCloud.callExperimentalAPI("{\\"api\\":\\"setSystemAudioKitEnabled\\", \\"param
```

NTP 校时问题

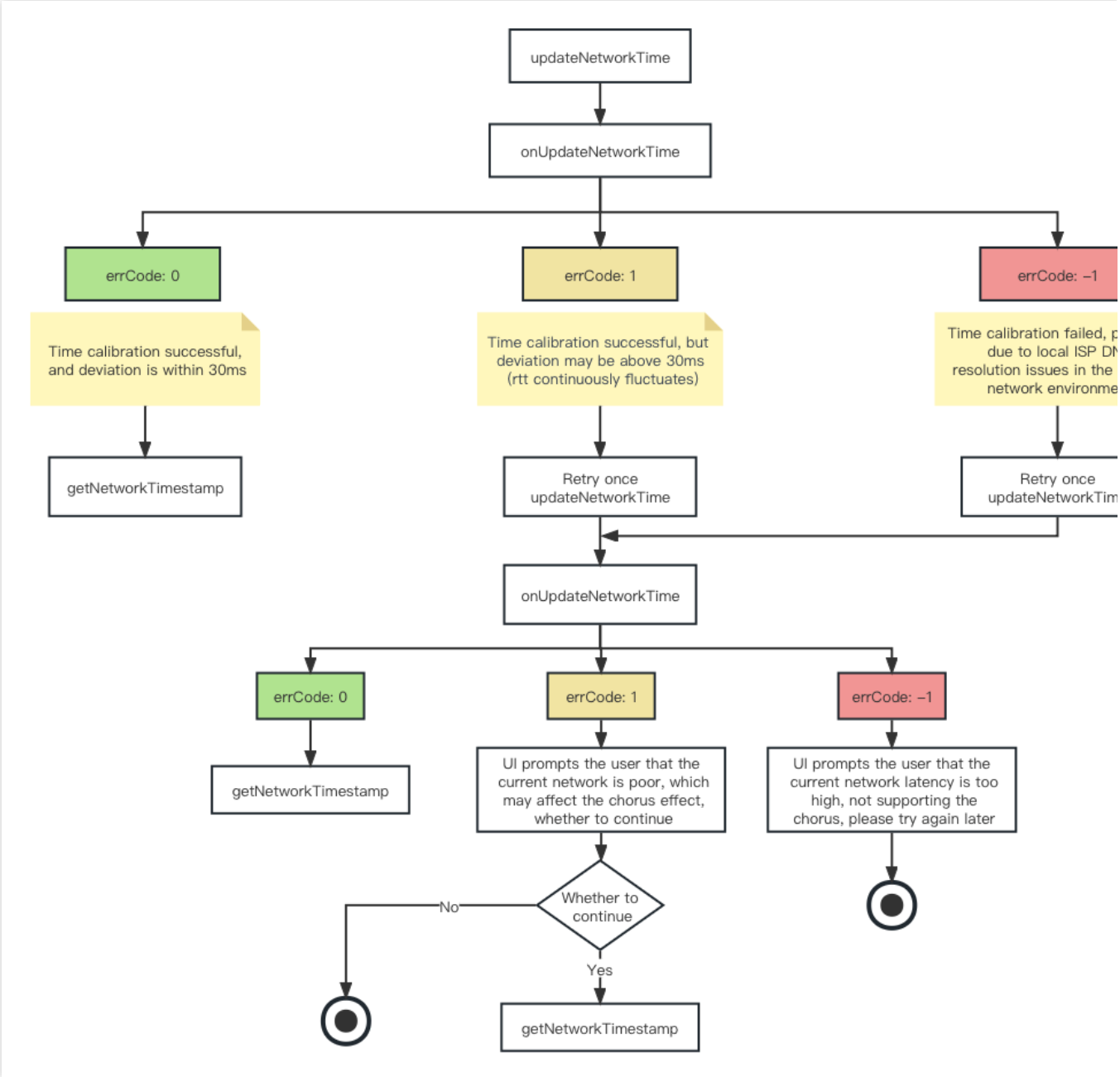
提醒：“NTP time sync finished, but result maybe inaccurate”？

NTP 校时成功，但偏差可能在30ms以上，反映客户端网络环境差，rtt 持续抖动。

提醒：“Error in AddressResolver: No address associated with hostname”？

NTP 校时失败，可能是当前网络环境下本地运营商 DNS 解析暂时异常，请稍后再试。

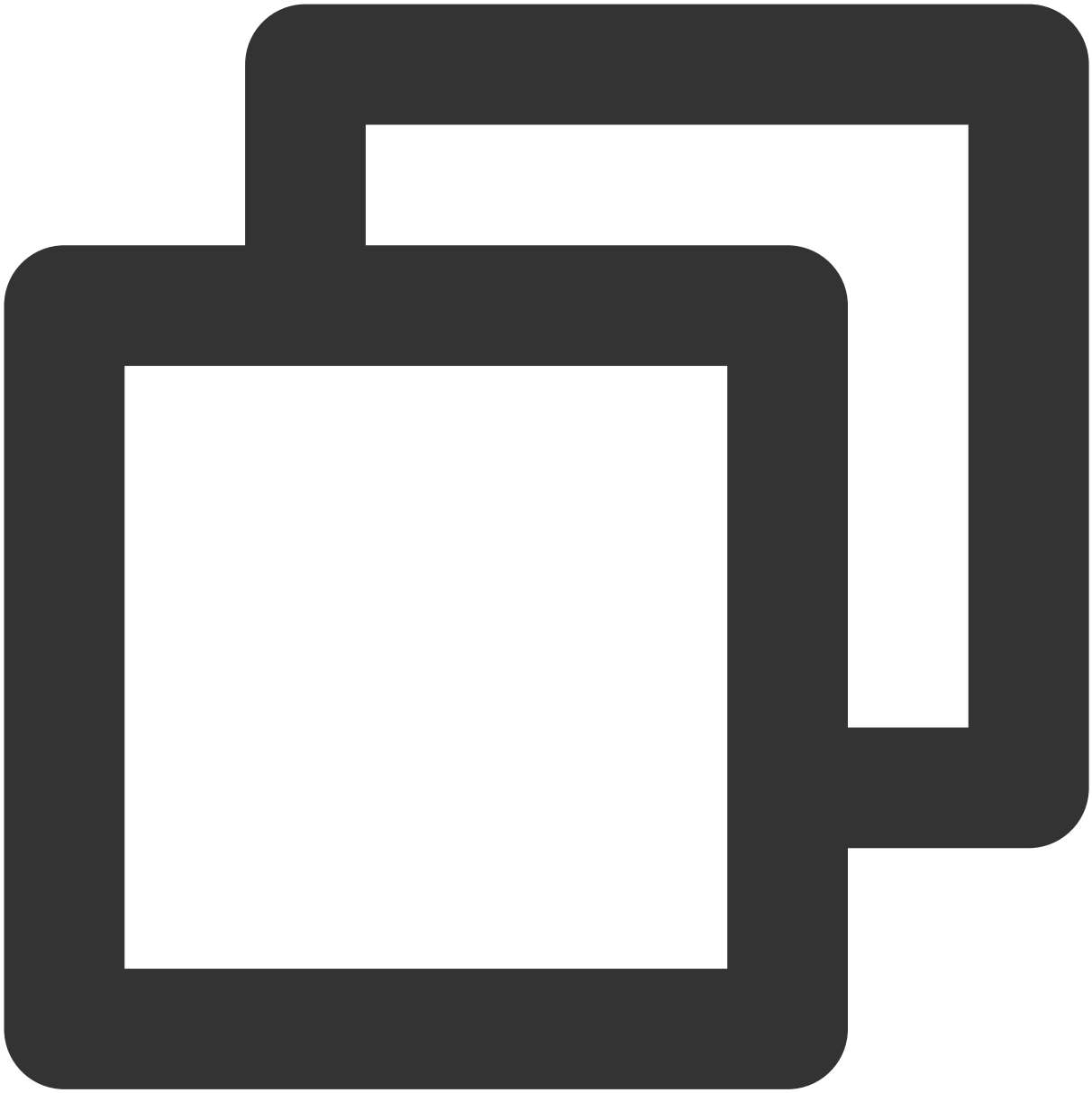
NTP 服务重试处理逻辑？



网络测速建议

在线 K 歌场景对用户的网络条件要求较高，特别是实时合唱，优质稳定的网络环境才能保障良好的 K 歌体验。因此，建议在用户进入房间前对该用户进行一次网络测速，对网络条件不符合要求的用户给予 UI 层的提醒，禁止其加入 K 歌房或参与合唱。

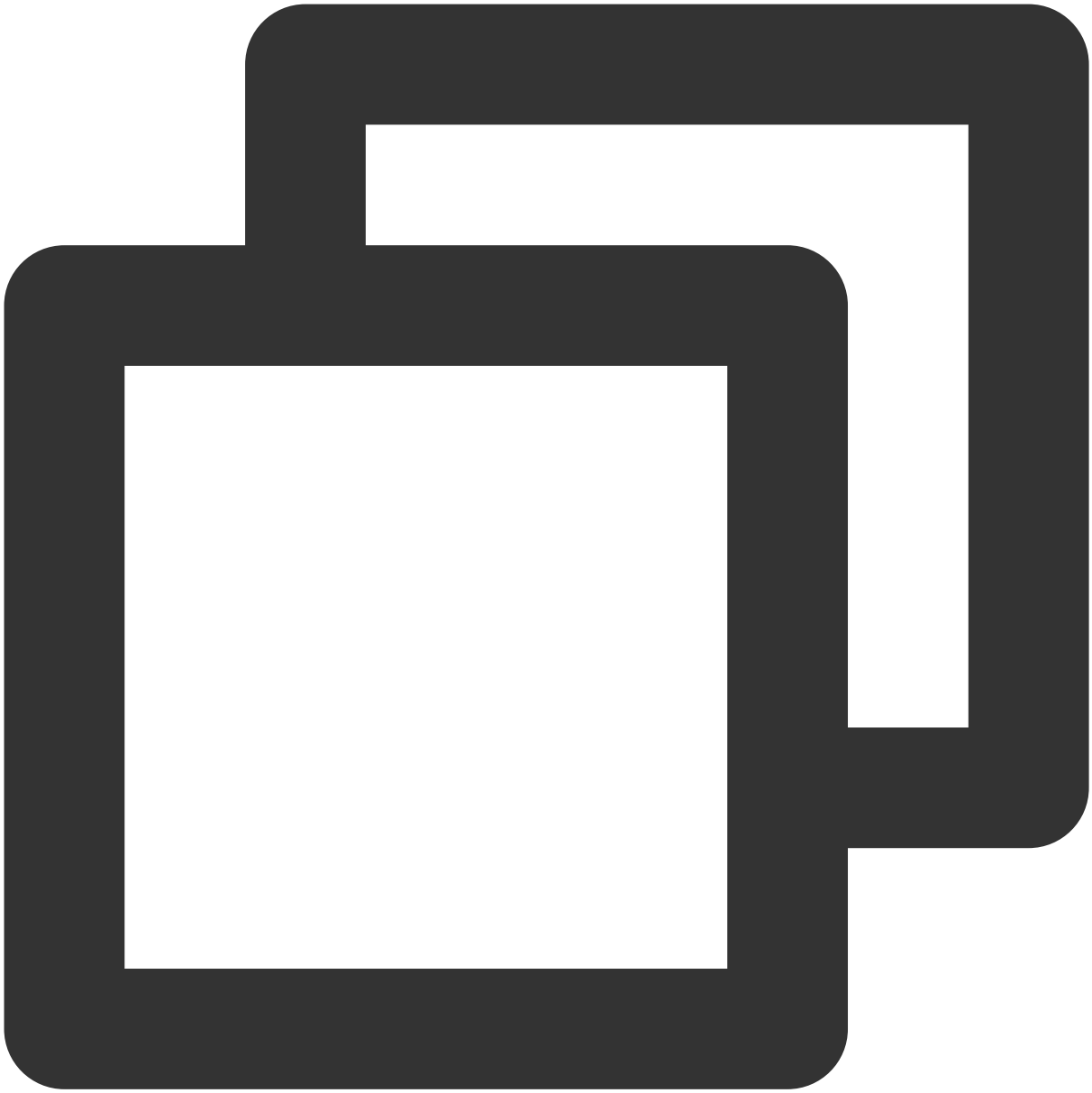
TRTC SDK 网络测速的发起：



```
TRTCCloudDef.TRTCSpeedTestParams speedTestParams = new TRTCCloudDef.TRTCSpeedTestPa
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// 若实际带宽高于预期值，则测试结果即为预期值；若实际带宽低于预期值，则测试结果为实际带宽值
speedTestParams.expectedDownBandwidth = 3000; // 预期的下行带宽，取值范围 10~5000 kbps
speedTestParams.expectedUpBandwidth = 3000; // 预期的上行带宽，取值范围 10~5000 kbps
mTRTCCloud.startSpeedTest(speedTestParams);
```

注意：

同一时间只允许一项网速测试任务运行；
请在进入房间前进行网速测试，在房间中网速测试会影响正常的音视频传输效果，而且由于干扰过多，网速测试结果也不准确。
TRTC SDK 网络测速结果的回调：



```
@Override
public void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result) {
    String tquality = "未定义";
    switch (result.quality) {
        case 0: tquality = "未定义";
            break;
```

```
            case 1: tquality = "当前网络非常好";
                break;
            case 2: tquality = "当前网络比较好";
                break;
            case 3: tquality = "当前网络一般";
                break;
            case 4: tquality = "当前网络较差";
                break;
            case 5: tquality = "当前网络很差";
                break;
            case 6: tquality = "当前网络不满足 TRTC 的最低要求";
                break;
        }
        if (result.success) {
            mTextTestResult.append("测速成功！" + "\\n");
            mTextTestResult.append("IP 地址：" + result.ip + "\\n");
            mTextTestResult.append("上行丢包率：" + result.upLostRate + "\\n");
            mTextTestResult.append("下行丢包率：" + result.downLostRate + "\\n");
            mTextTestResult.append("网络延迟：" + result.rtt + "ms\\n");
            mTextTestResult.append("下行带宽：" + result.availableDownBandwidth + "kbps\\
            mTextTestResult.append("上行带宽：" + result.availableUpBandwidth + "kbps\\n"
            mTextTestResult.append("网络质量：" + tquality + "\\n");
        } else {
            mTextTestResult.append("测速失败！" + "\\n");
            mTextTestResult.append("errMsg：" + result.errMsg + "\\n");
        }
    }
```

中途加入合唱

实时合唱方案理论上对合唱者数量没有限制，支持多人同时参与合唱，也支持中途加入合唱。

下面列出中途加入合唱的关键动作：

NTP 校时

开启合唱实验性接口

进入房间并上麦推流

接收合唱信令，获取伴奏资源及合唱约定时间

计算约定时间与当前时间差，预加载 seek 伴奏

开始参与合唱，并实时同步伴奏进度及歌词进度

以上关键动作涉及的具体实现流程及代码实现详见歌曲同步、歌词同步、人声同步文档。