实时音视频

视频通话（含 UI）

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默示的承诺或保证。

文档目录

- 视频通话（含 UI）
  - 组件介绍（TUICallKit）
  - 开通服务（TUICallKit）
  - 跑通 Demo（TUICallKit）
    - Android
    - iOS
    - Web
    - Flutter
  - 快速接入（TUICallKit）
    - Android
    - iOS
    - Web&H5 (React)
    - Web&H5 (Vue3)
    - Flutter
    - uniapp（Android&iOS）
  - 界面定制（TUICallKit）
    - Android
    - iOS
    - Web
    - Flutter
  - 离线唤醒（TUICallKit）
    - iOS
      - VoIP
      - Notification
    - Android
    - Flutter
      - Notification
      - VoIP(可选)
  - AI 降噪（TUICallKit）
  - 虚拟背景（TUICallKit）
    - iOS
    - Android
    - Web
    - Flutter
  - 云端录制（TUICallKit）

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）
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)
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）
语言设置
Web&H5
iOS
Android
Flutter
uni-app（Android&iOS）
服务端 API（TUICallKit）
通话状态回调
通话状态回调
通话事件回调
回调配置
回调配置 API 列表
创建回调配置
查询回调配置
更新回调配置

删除回调配置
REST API
REST API 简介
通过 callId 获取记录
通过条件获取记录
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
TUICallKit
TUICallEngine
TUICallEvent
错误码表（TUICallkit）
发布日志（TUICallKit）
Web
Android&iOS

Flutter
常见问题（TUICallKit）
iOS
Android
Web
全平台
Flutter

版权所有：腾讯云计算（北京）有限责任公司

视频通话（含 UI）组件介绍（TUICallKit）

最近更新时间：2024-06-17 17:41:52

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	Flutter

是否支持				
语言/框架	Kotlin Java	Swift Objective-C	Vue3 React	Dart

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	Flutter
在线体验				/
快速跑通	Web Demo	Andorid Demo	iOS Demo	Flutter 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）

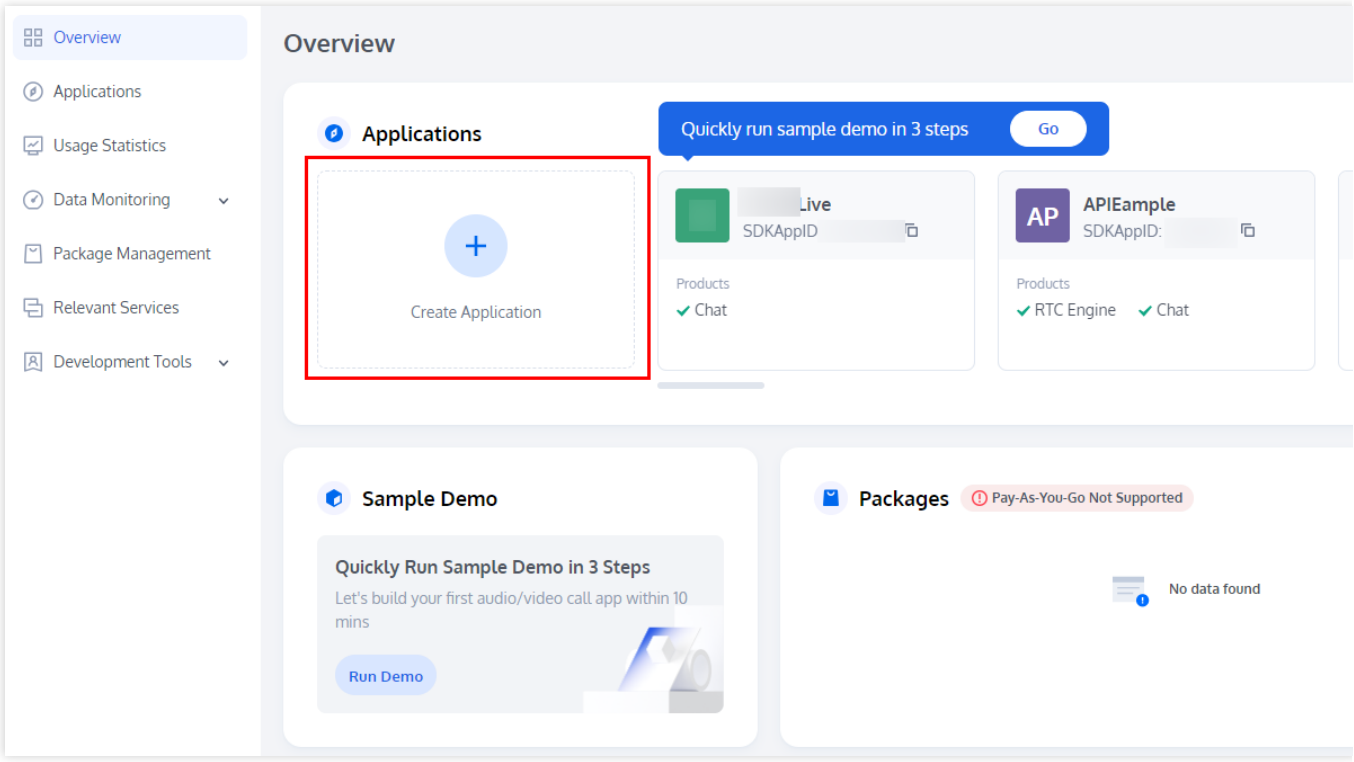
最近更新时间：2024-07-19 11:13:48

开通体验服务

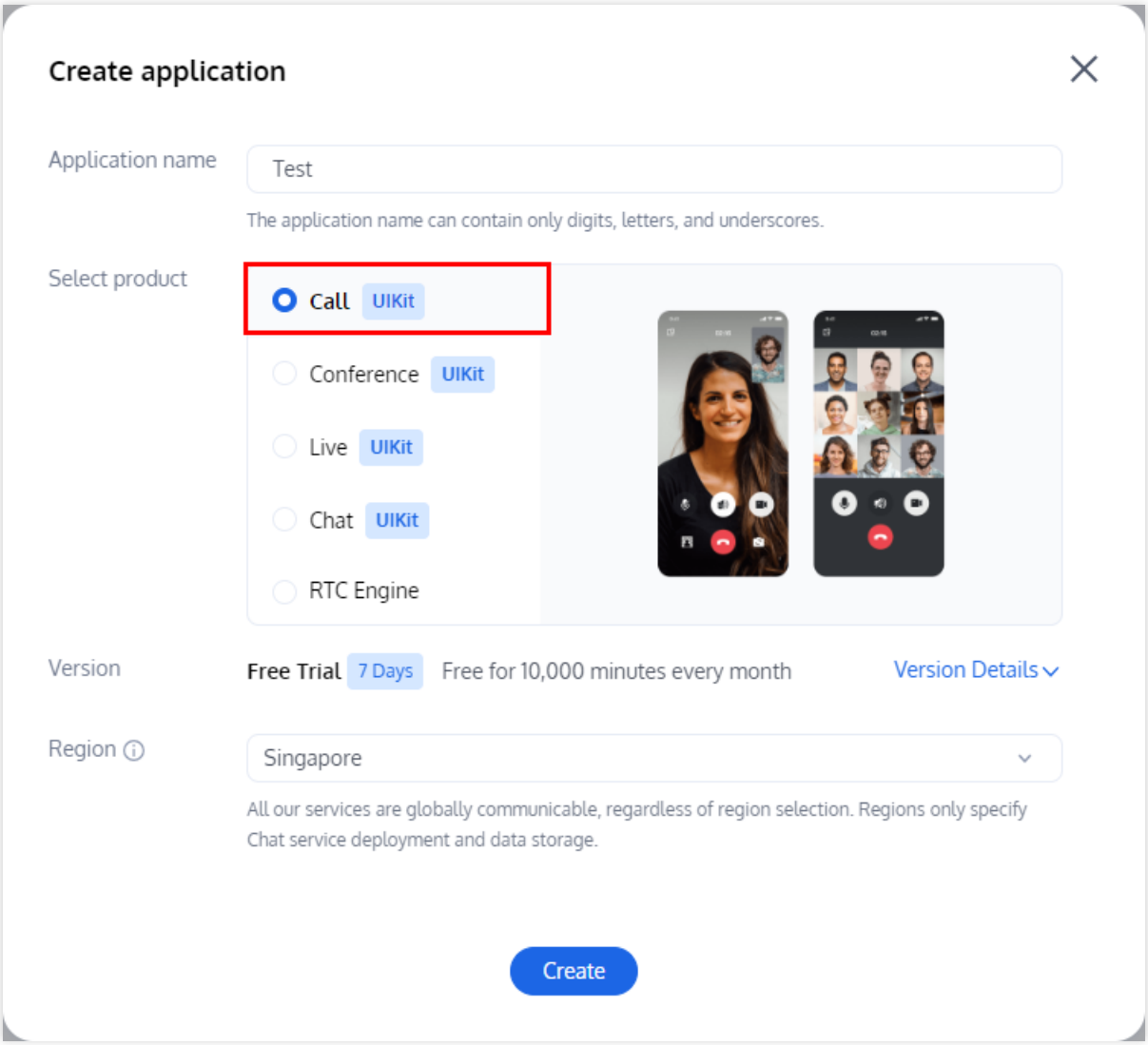
为了您能更好地体验音视频通话 Call 的功能，我们免费为每个 SDKAppID 提供了 7 天体验版（体验版不额外赠送通话时长），每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。

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：

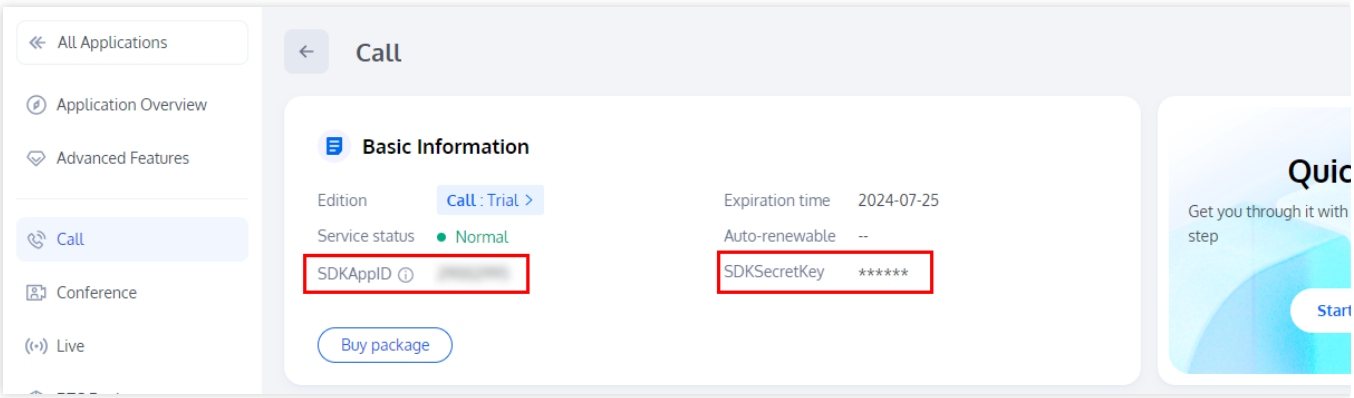
1. 登录到 Tencent RTC 控制台，单击 **Create Application**。



2. 在创建弹窗中选择 **Call** 并输入应用名称，选择数据存储**地区**，单击 **Create**。



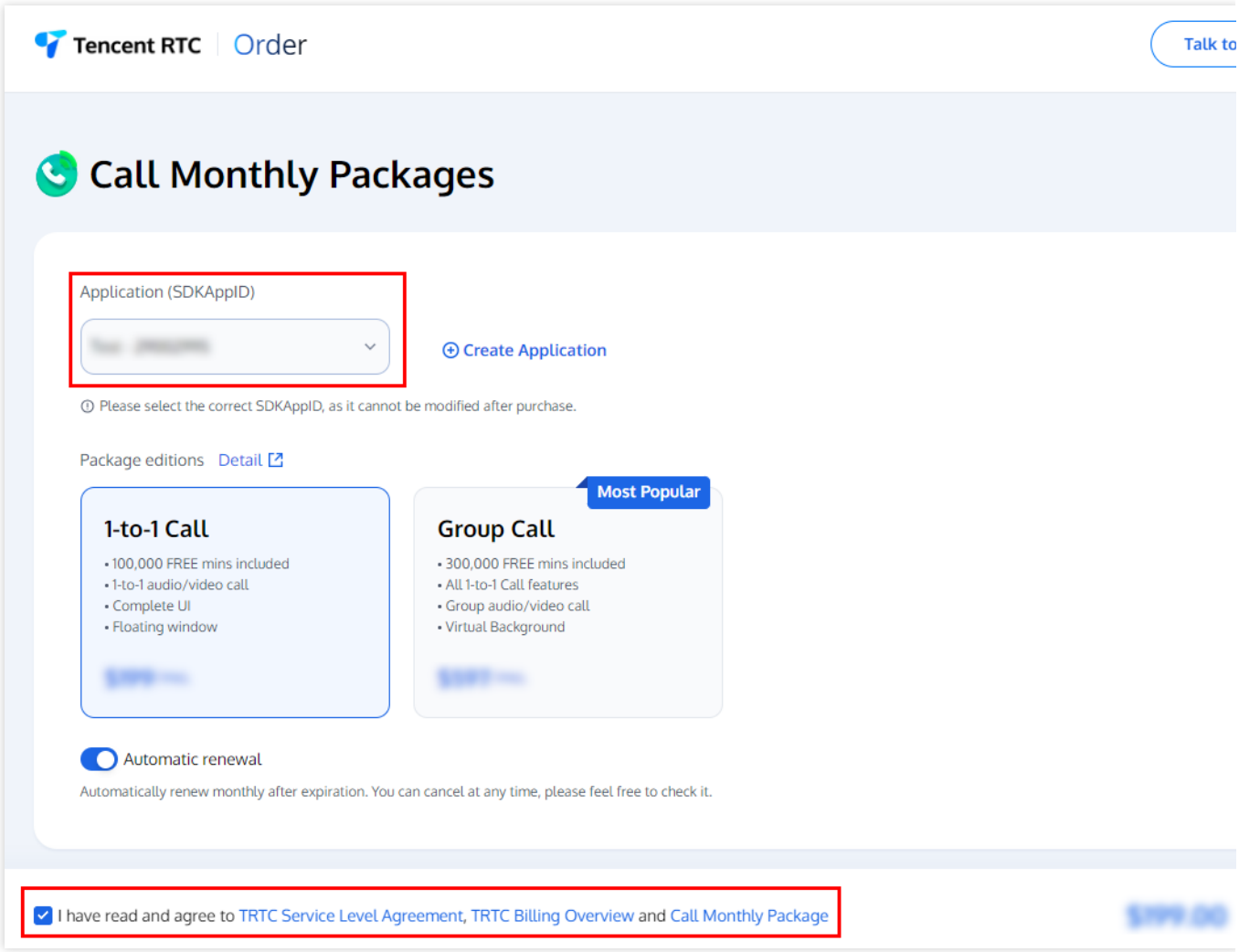
3. 完成应用创建后将默认进入 Call 应用详情页，此时您已快速创建应用并已成功领取音视频通话（TUICallKit）体验版。可在当前 **Call 应用详情页面**或 **Application Overview** 查看信息，并可参见 集成指引 进行集成，这里的 `SDKAppID` 、 `SDKSecretKey` 会在集成指引中使用到。



购买正式版

Call 包月套餐的价格和功能对比详情可参考 Call 包月套餐计费说明，**如果您需要购买 Call 包月套餐，请跟随下面的步骤。**

1. 访问 Tencent RTC 购买页面，选择需要购买的应用（SDKAppID）及套餐类型，确认购买信息并同意相关协议后，单击 **Subscribe now**。



2. 购买完成后，您可前往 Tencent RTC 控制台 查看应用信息。

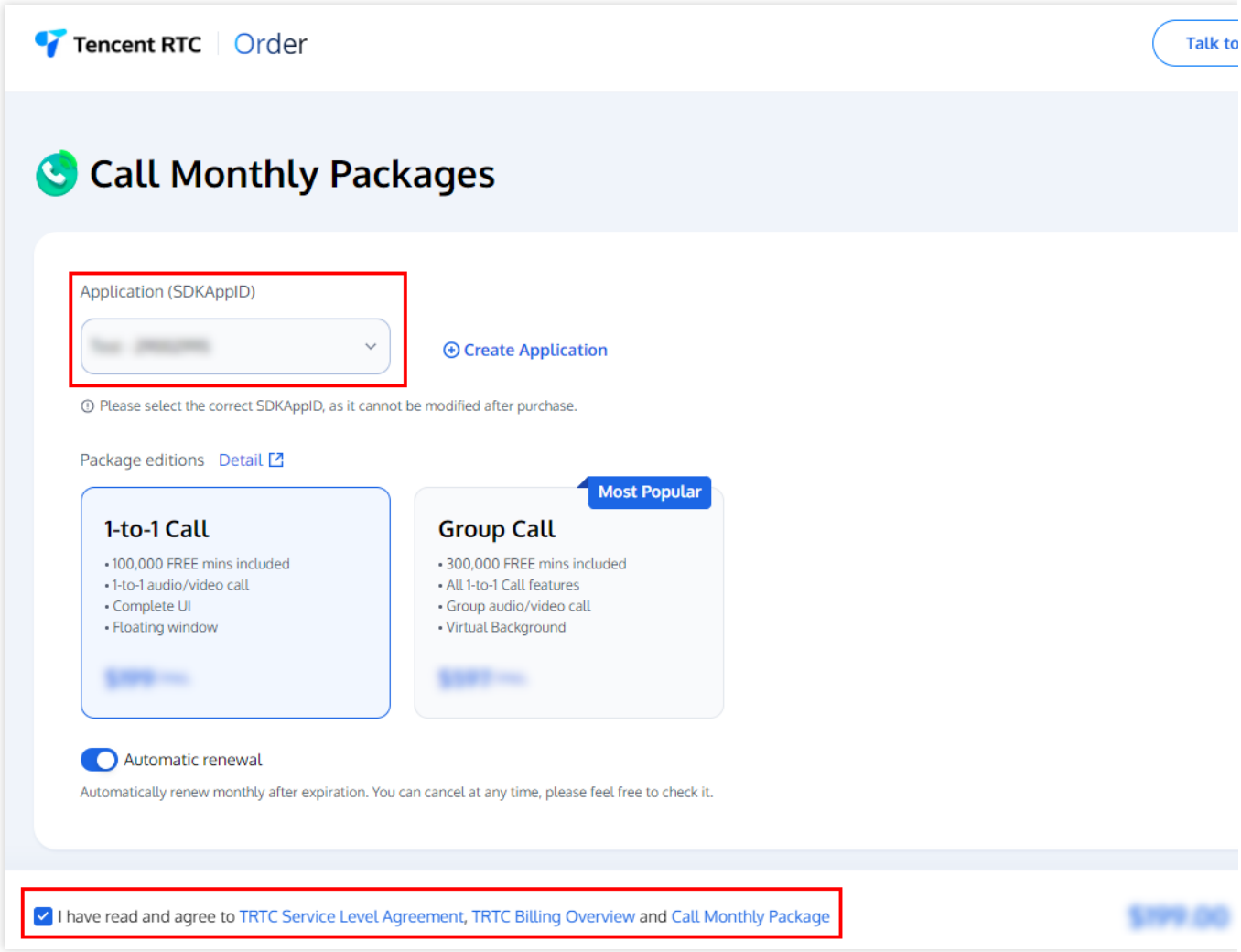
续费正式版

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**。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。

注意：

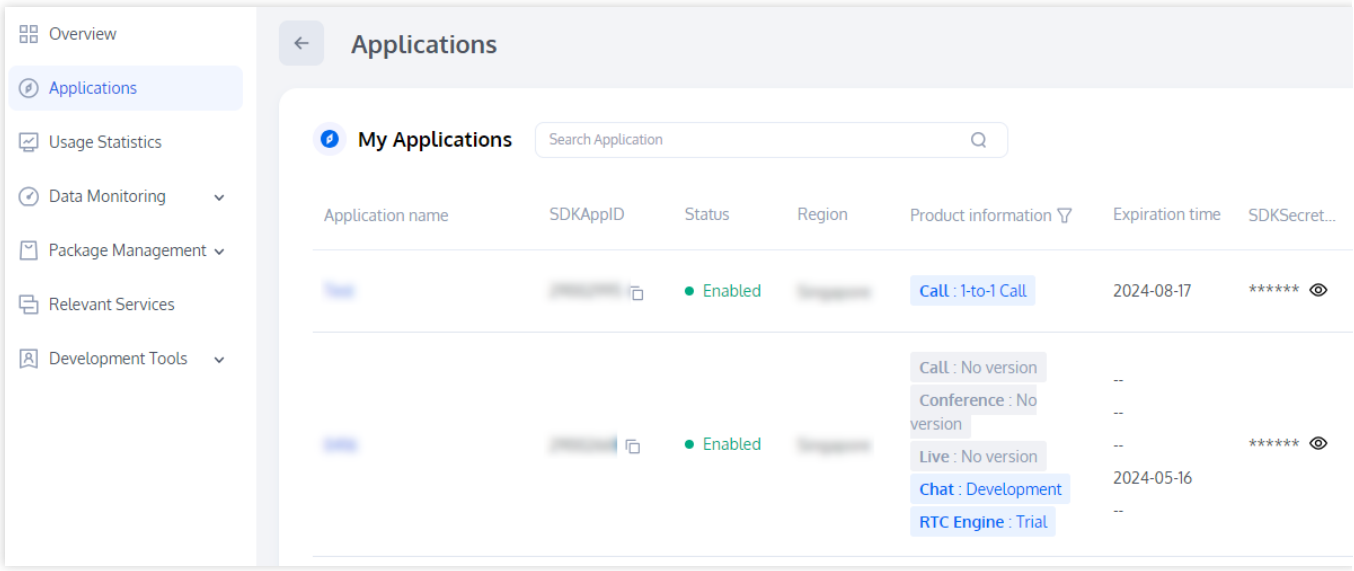
请选择需要续期的应用 SDKAppID。



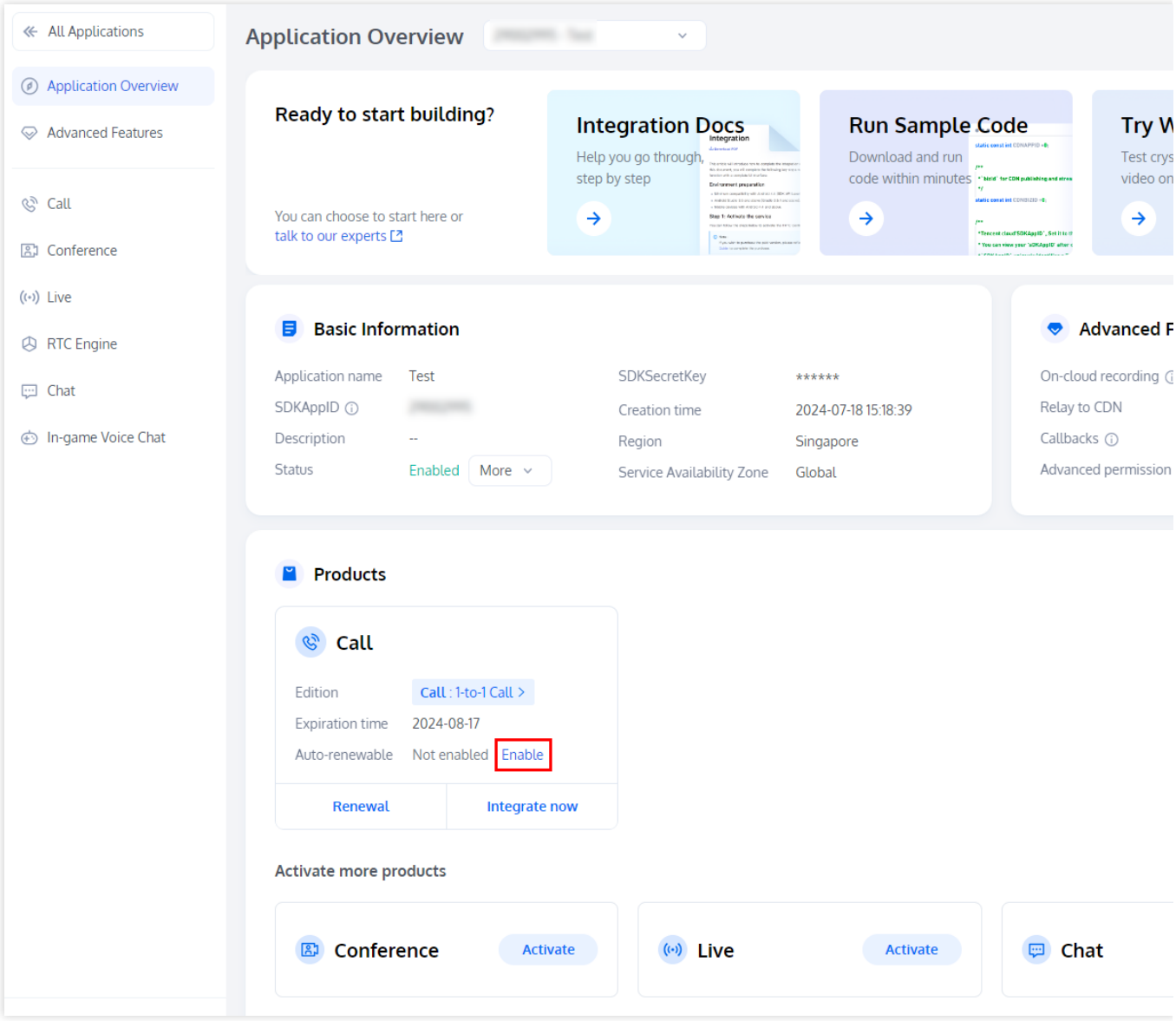
自动续费

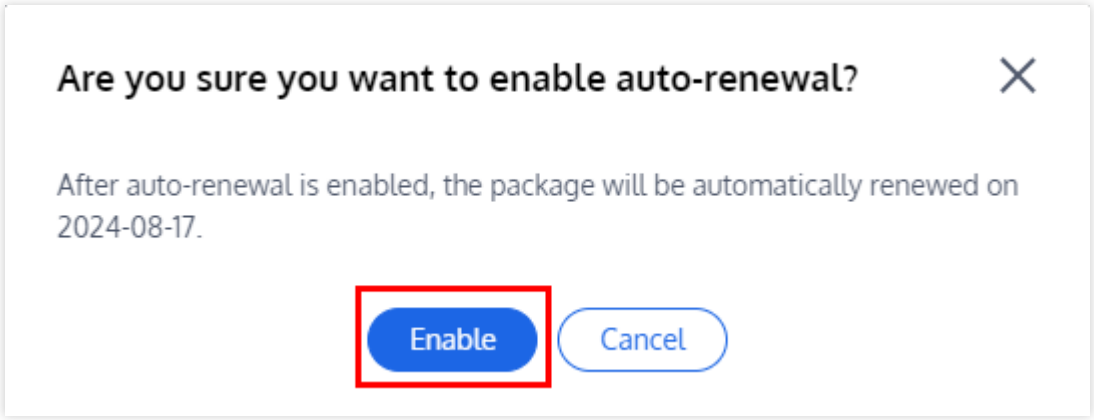
控制台开启自动续期的具体步骤如下：

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



2. 在 Call 产品信息中，单击 **Enable**，将出现确认弹窗，单击 **Enable**。

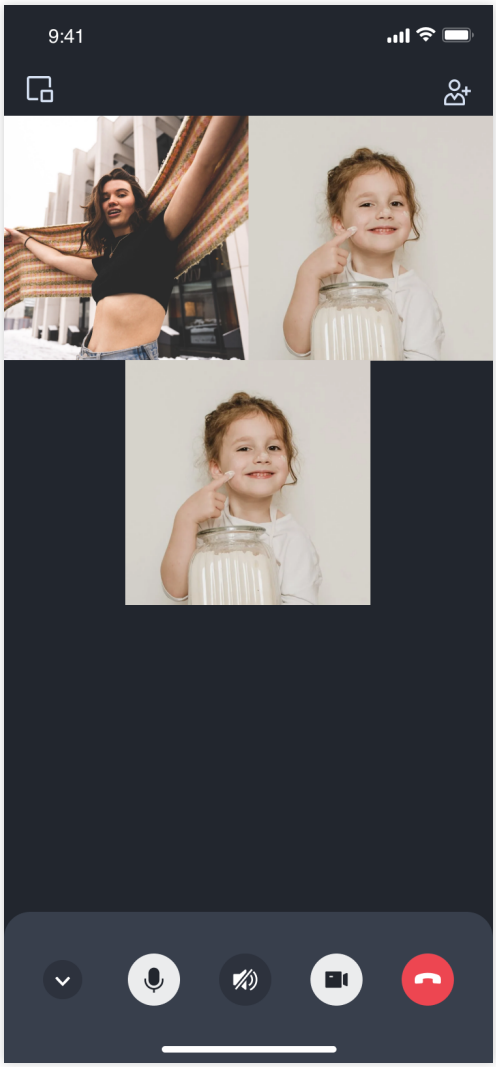

All Applications
Application Overview
Advanced Features
Call
Conference
Live
RTC Engine
Chat
In-game Voice Chat
Application Overview
Ready to start building?
You can choose to start here or talk to our experts
Integration Docs
Help you go through, step by step
Run Sample Code
Download and run code within minutes
Basic Information
Application name
Test
SDKAppID
Description
--
Status
Enabled
More
SDKSecretKey
******
Creation time
2024-07-18 15:18:39
Region
Singapore
Service Availability Zone
Global
Advanced F
On-cloud recording
Relay to CDN
Callbacks
Advanced permission
Products
Call
Edition
Call : 1-to-1 Call
Expiration time
2024-08-17
Auto-renewable
Not enabled
Enable
Renewal
Integrate now
Activate more products
Conference
Activate
Live
Activate
Chat


Are you sure you want to enable auto-renewal?
After auto-renewal is enabled, the package will be automatically renewed on 2024-08-17.
Enable
Cancel

跑通 Demo（TUICallKit）
Android

最近更新时间：2024-06-18 17:27:24

本文将为您介绍如何快速跑通音视频通话 Demo。跟随本文档，您可以在 10 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的音视频通话功能。

视频通话	群组通话
	

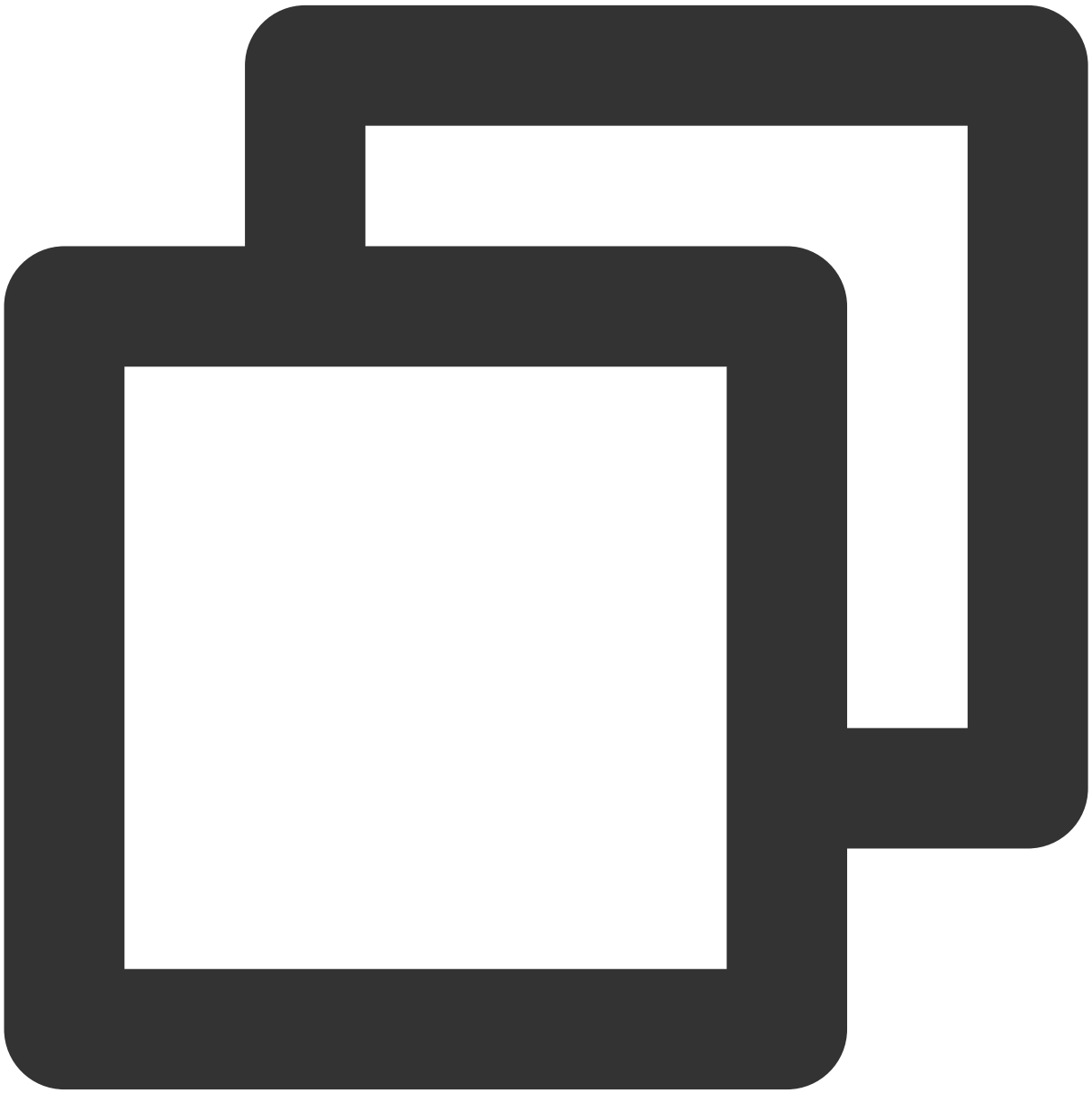
环境准备

Android Studio

两台 Android 5.0 以上的设备。

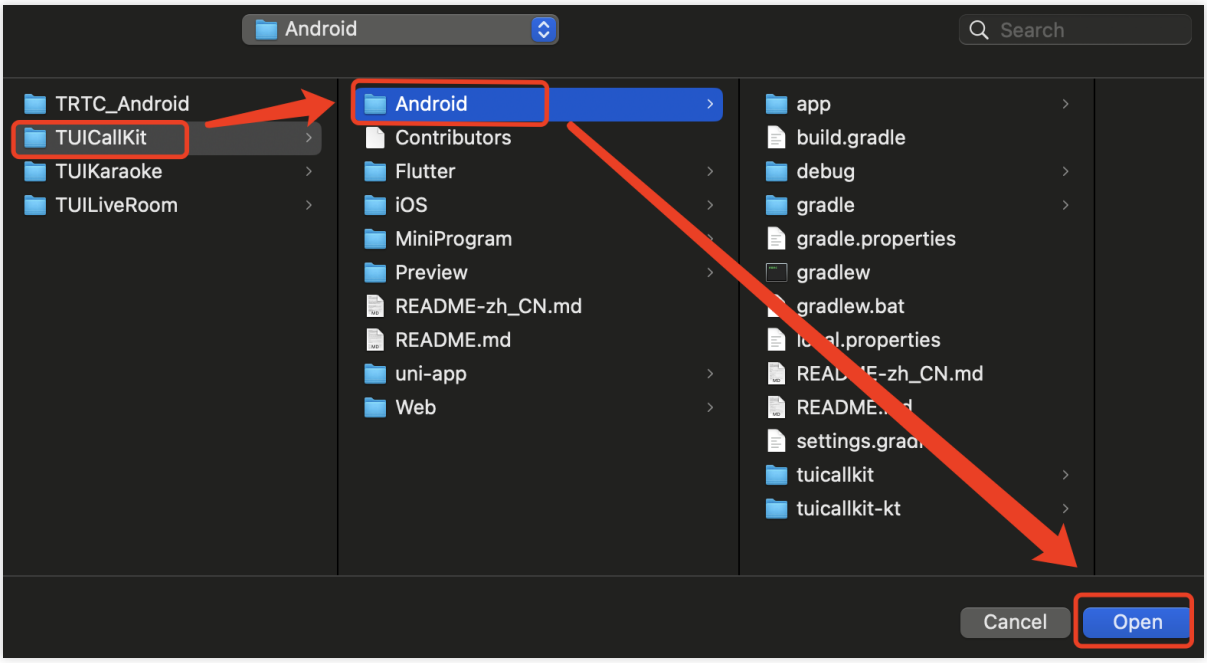
步骤一：下载 Demo

1. 从 github 下载 TUICallKit Demo 源码，或者直接在命令行运行以下命令：



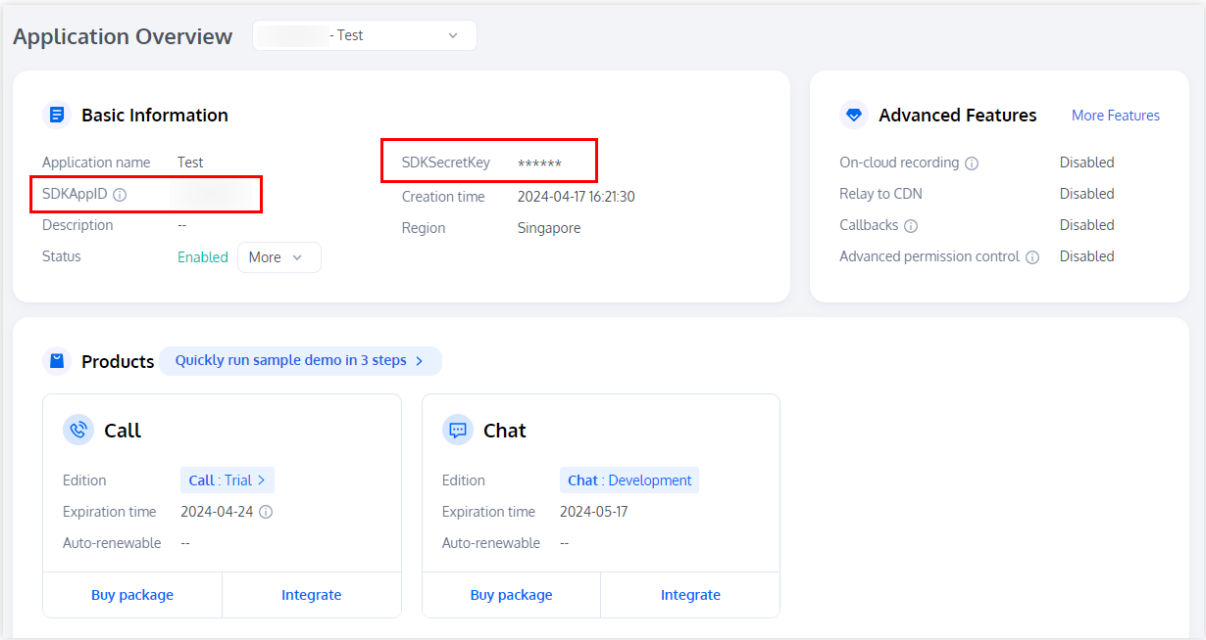
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. 通过 Android Studio 打开 TUICallKit **Android** 项目：

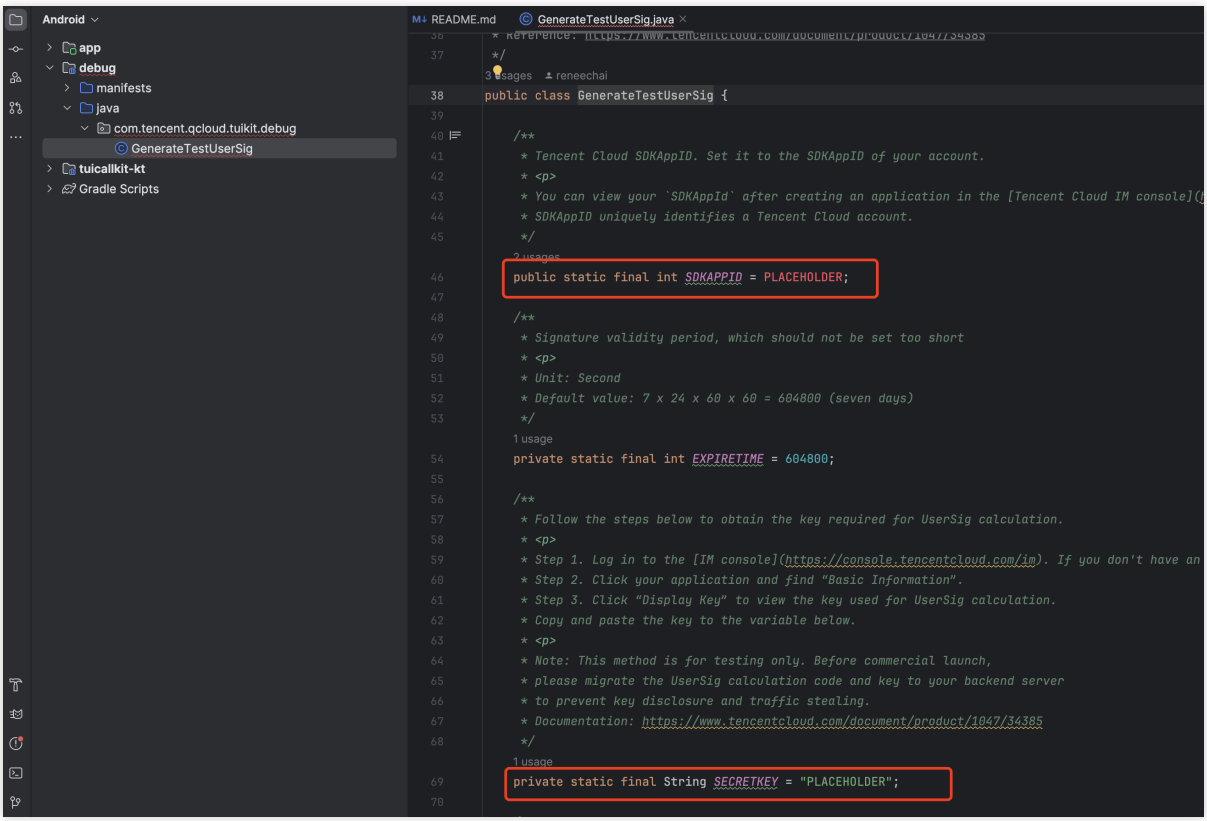


步骤二：配置 Demo

1. 开通音视频服务，获取 SDKAppID 和 SDKSecretKey 。

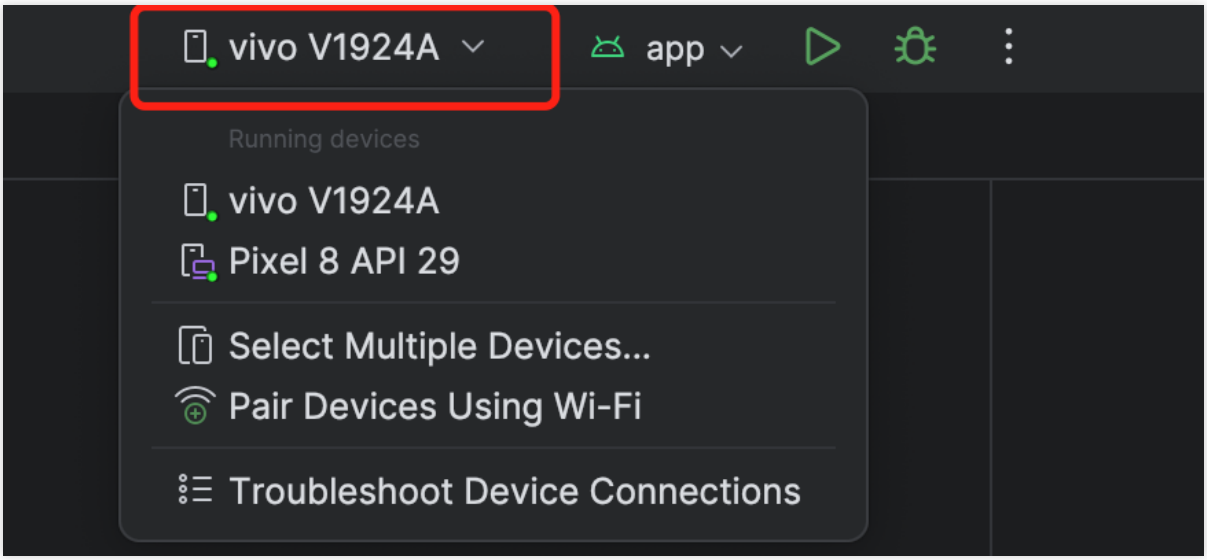


2. 打开 `Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java` 文件，将开通服务时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

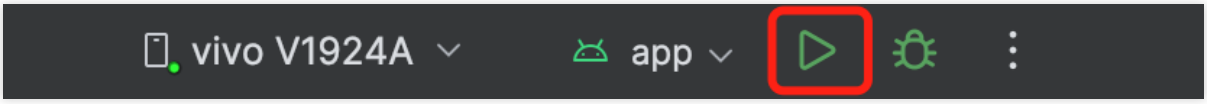


步骤三：跑通 Demo

1. 在 Android Studio 右上角如下图所示处选择您要将 Demo 运行的设备：



2. 选择完成后点击运行，将 TUICallKit Android Demo 运行到目标设备上。



3. 当 Demo 成功运行到设备上后，您可以通过如下步骤来发起一次通话：

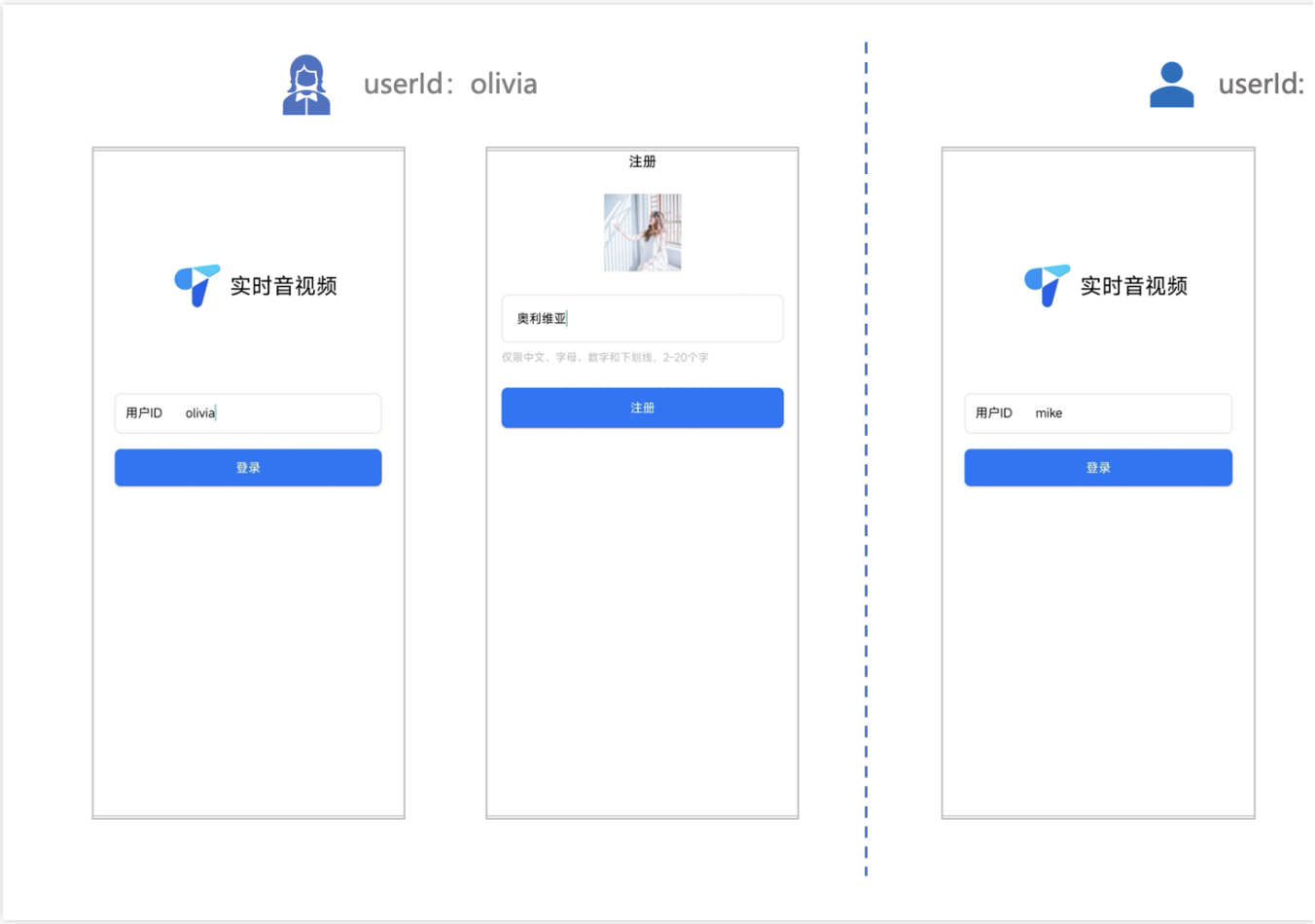
拨打第一个电话

注意：

为了使您可以体验完整的音视频通话流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主叫，一方作为被叫。

1. 登录 & 注册

请您在 `用户ID` 处输入的 ID 。如果你当前的 UserID 未曾被使用过，此时会进入到**注册**界面，您可以在该界面为自己设置头像及昵称。



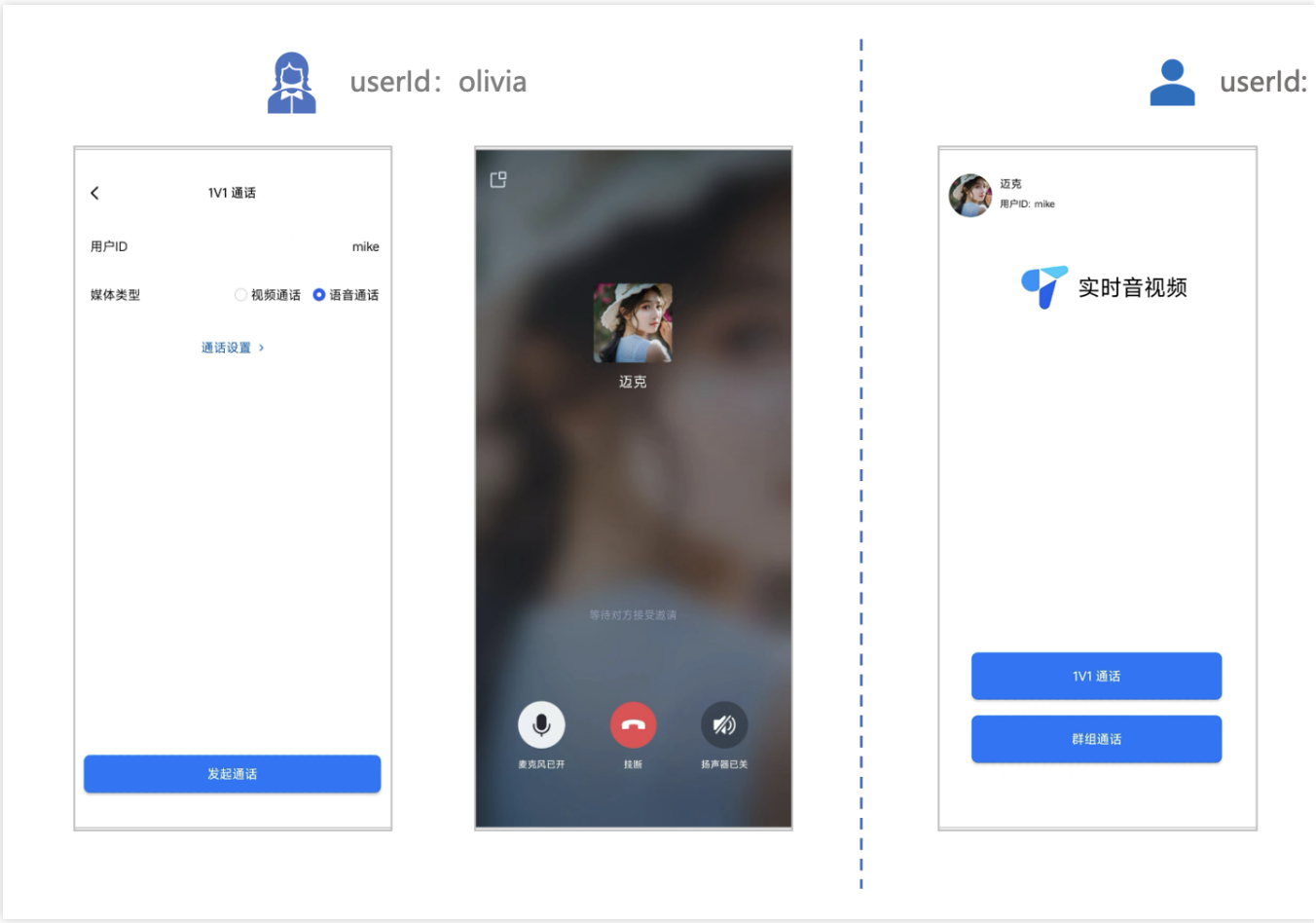
注意：

尽量避免使您的 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。

2. 拨打电话

2.1 主叫方请点击界面上的 **1V1 通话**，在弹出的界面中输入被叫方的 UserID 并选择您想要的通话类型。

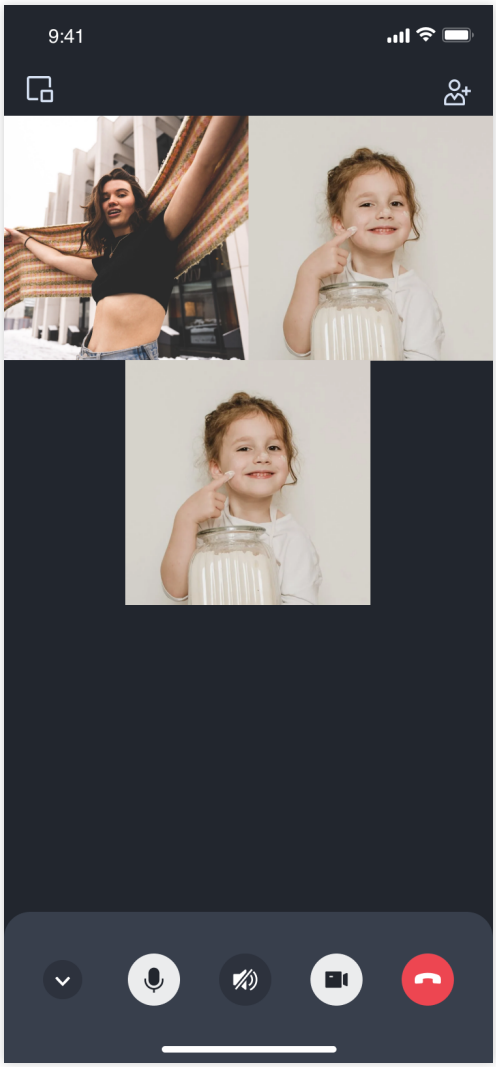
2.2 单击**发起通话**。


userId：olivia
1V1 通话
用户ID
mike
媒体类型
视频通话
语音通话
通话设置 ›
发起通话
迈克
等待对方接受邀请
麦克风已开
挂断
扬声器已关
userId:
迈克
用户ID: mike
实时音视频
1V1 通话
群组通话

iOS

最近更新时间：2024-07-19 18:10:53

本文将为您介绍如何快速跑通音视频通话 Demo。跟随本文档，您可以在 10 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的音视频通话功能。

视频通话	群组通话
	

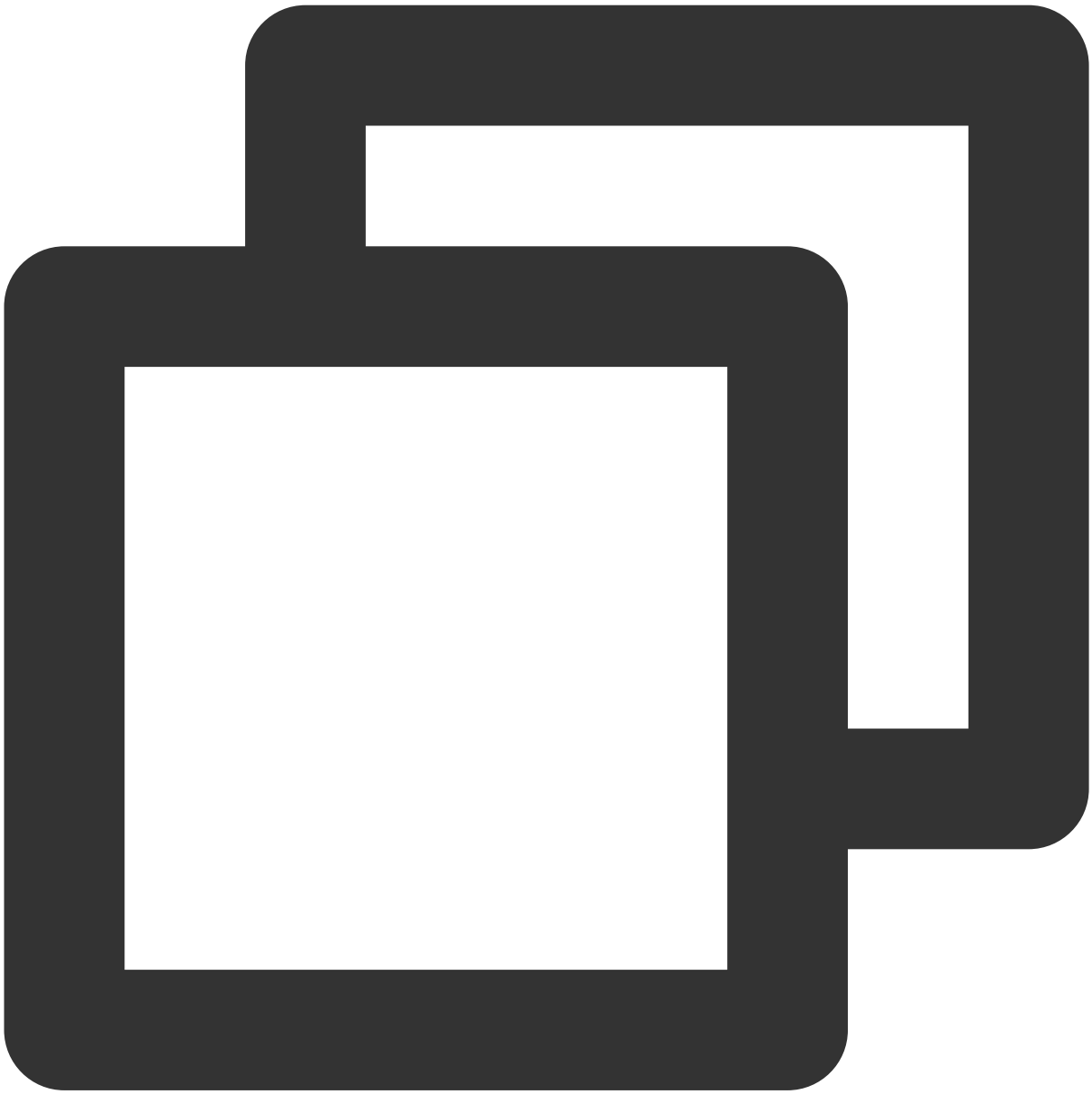
环境准备

Xcode 13 及以上。

两台 iOS 13.0 及以上设备。

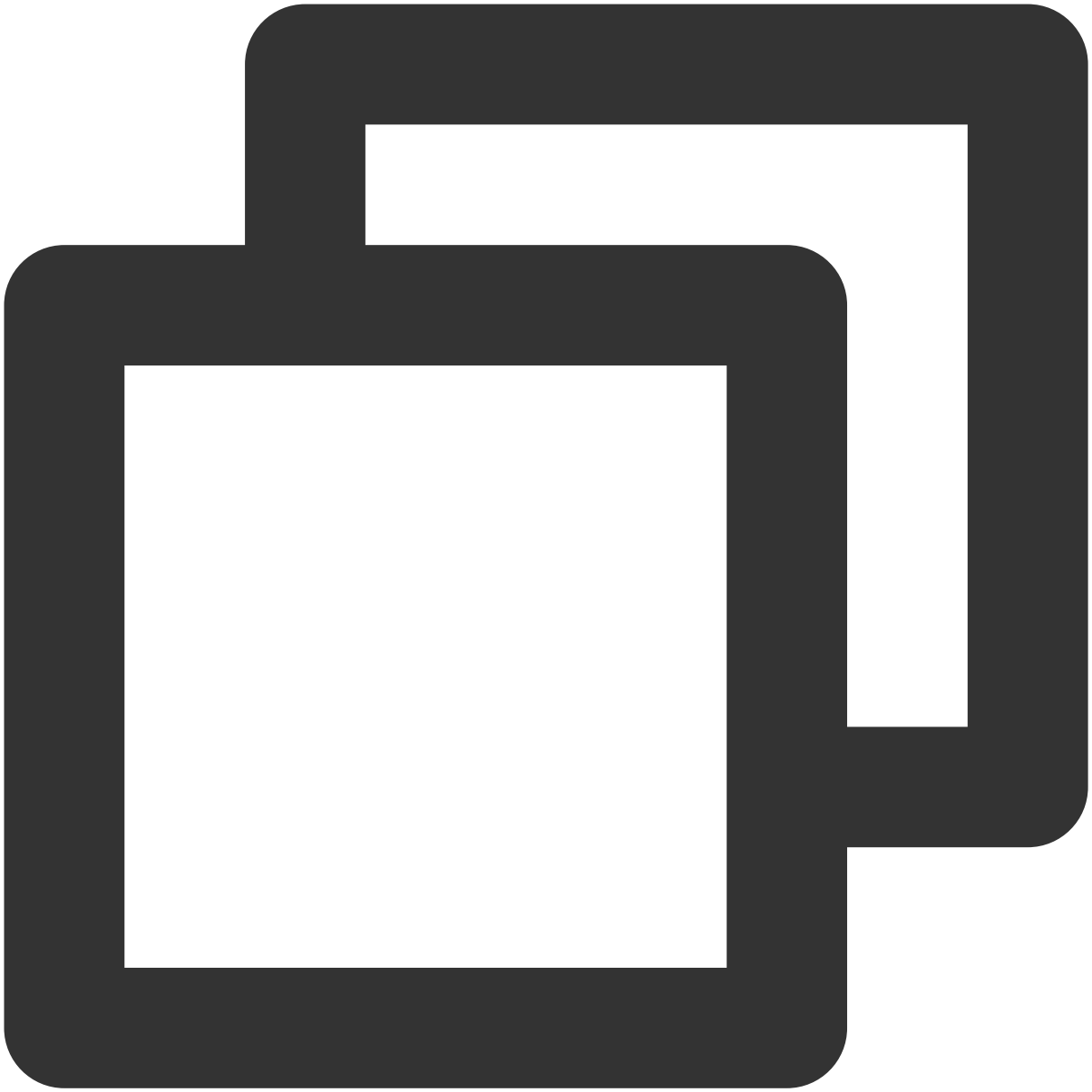
步骤一：下载 Demo

1. 从 github 下载 TUICallKit Demo 源码，或者直接在命令行运行以下命令：



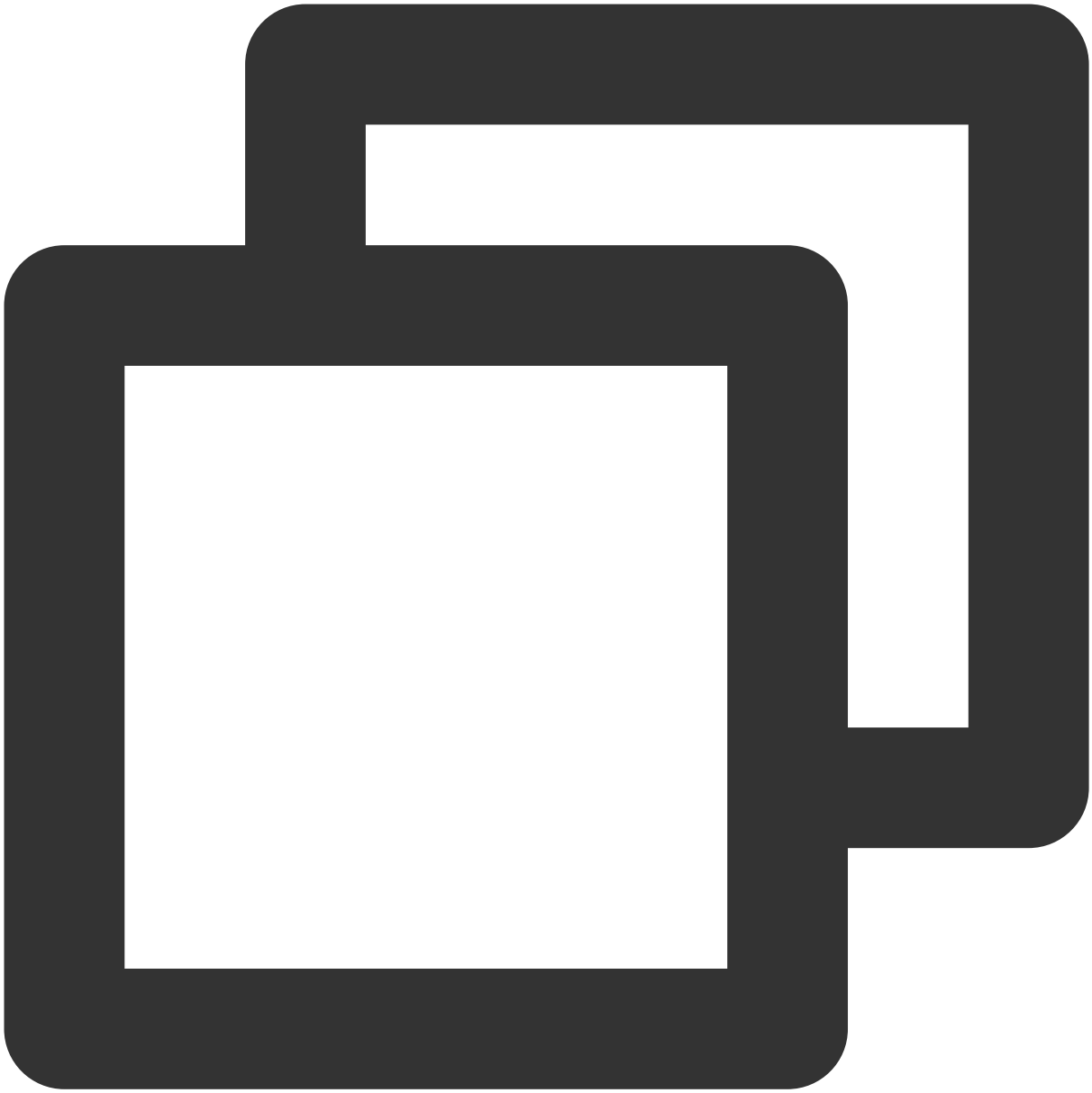
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. 在命令行中进入 iOS 项目目录：



```
cd TUICallKit/iOS/Example
```

3. 加载依赖库：



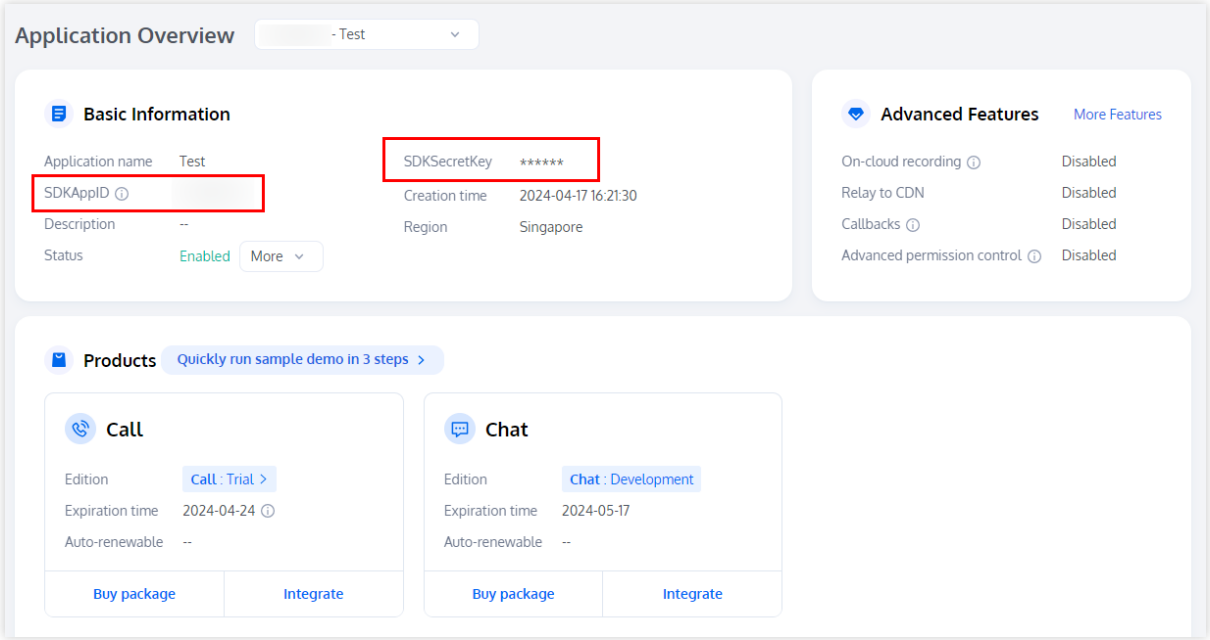
```
pod install
```

注意：

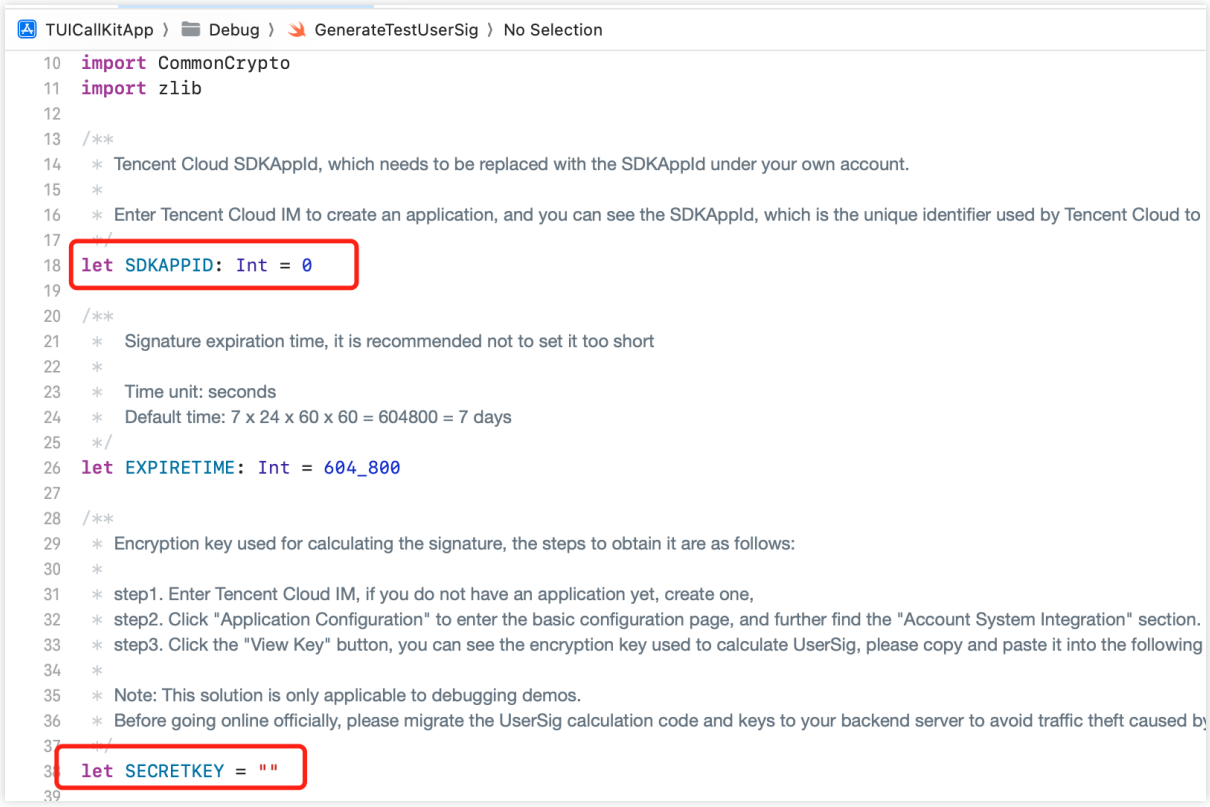
如果您尚未安装 CocoaPods，可以参考 此处 了解如何安装。

步骤二：配置 Demo

1. 开通音视频服务，获取 **SDKAppID** 和 **SDKSecretKey** 。

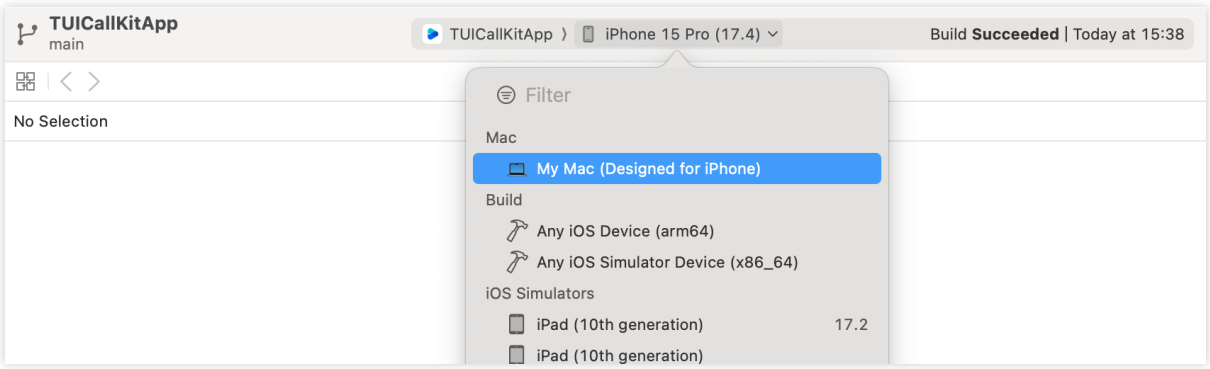


2. 打开 `/iOS/Example/Debug/GenerateTestUserSig.swift` 文件，将开通服务时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：



步骤三：跑通 Demo

1. 在 XCode 如下图所示处选择您要将 Demo 运行的设备：



2. 选择完成后点击运行将我们的 TUICallKit iOS Demo 运行到目标设备上。

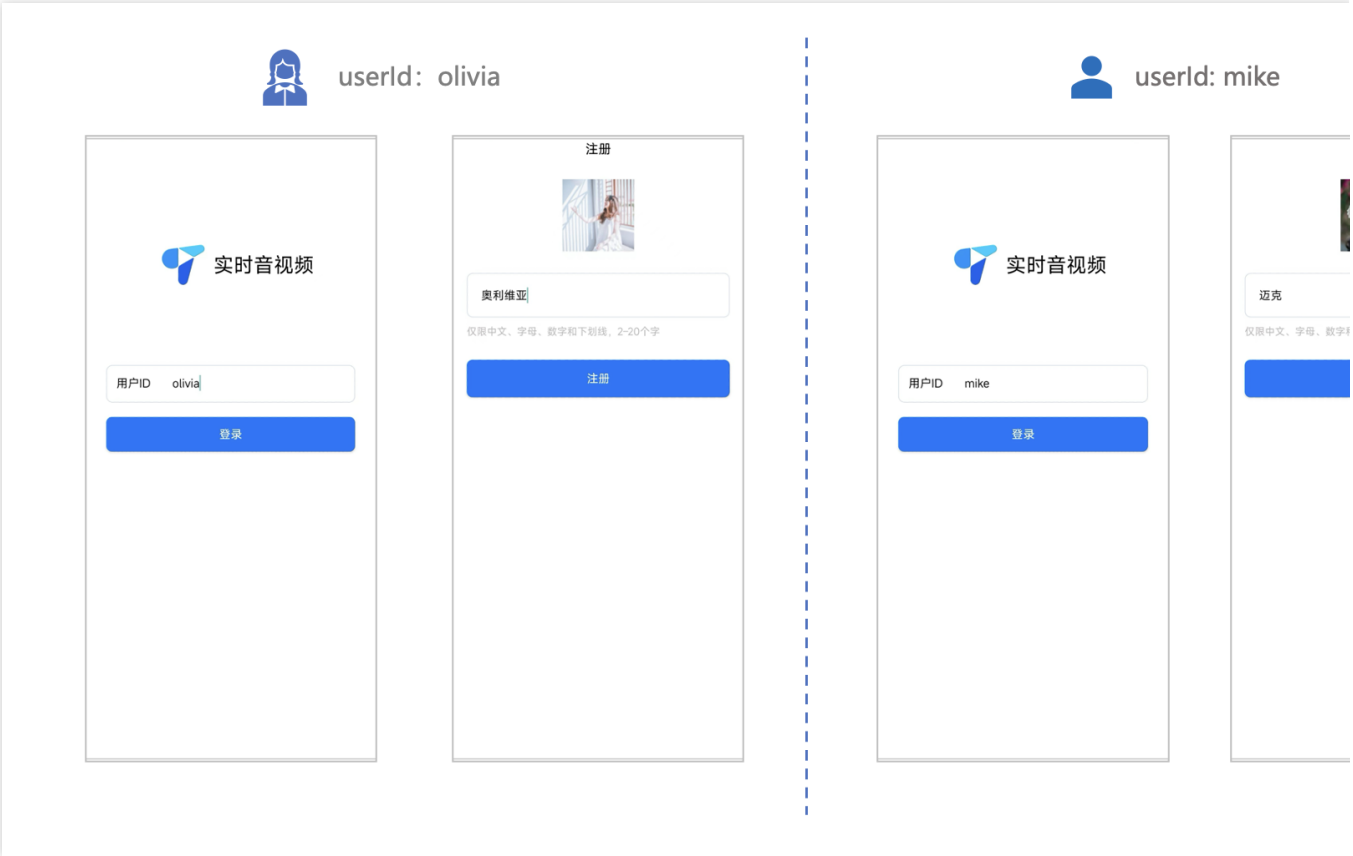
拨打第一个电话

注意：

为了使您可以体验完整的音视频通话流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主叫，一方作为被叫。

1. 登录 & 注册

请您在 `用户ID` 处输入的 ID 。如果你当前的 UserID 未曾被使用过，此时会进入到 **注册** 界面，您可以在该界面为自己设置头像及昵称。



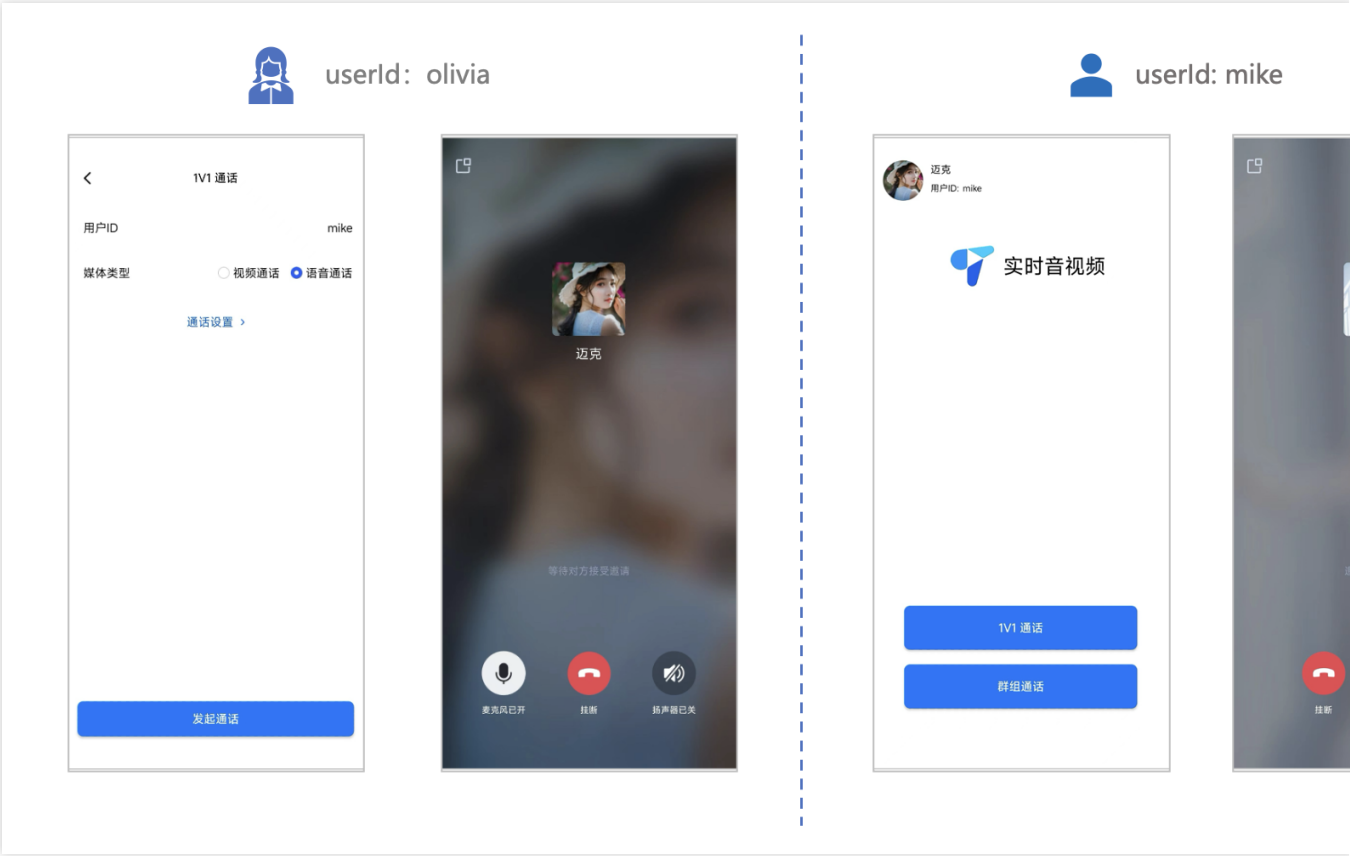
注意：

尽量避免使您的 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。

2. 拨打电话

2.1 主叫方请点击界面上的 **1V1 通话**，在弹出的界面中输入被叫方的 UserID 并选择您想要的通话类型。

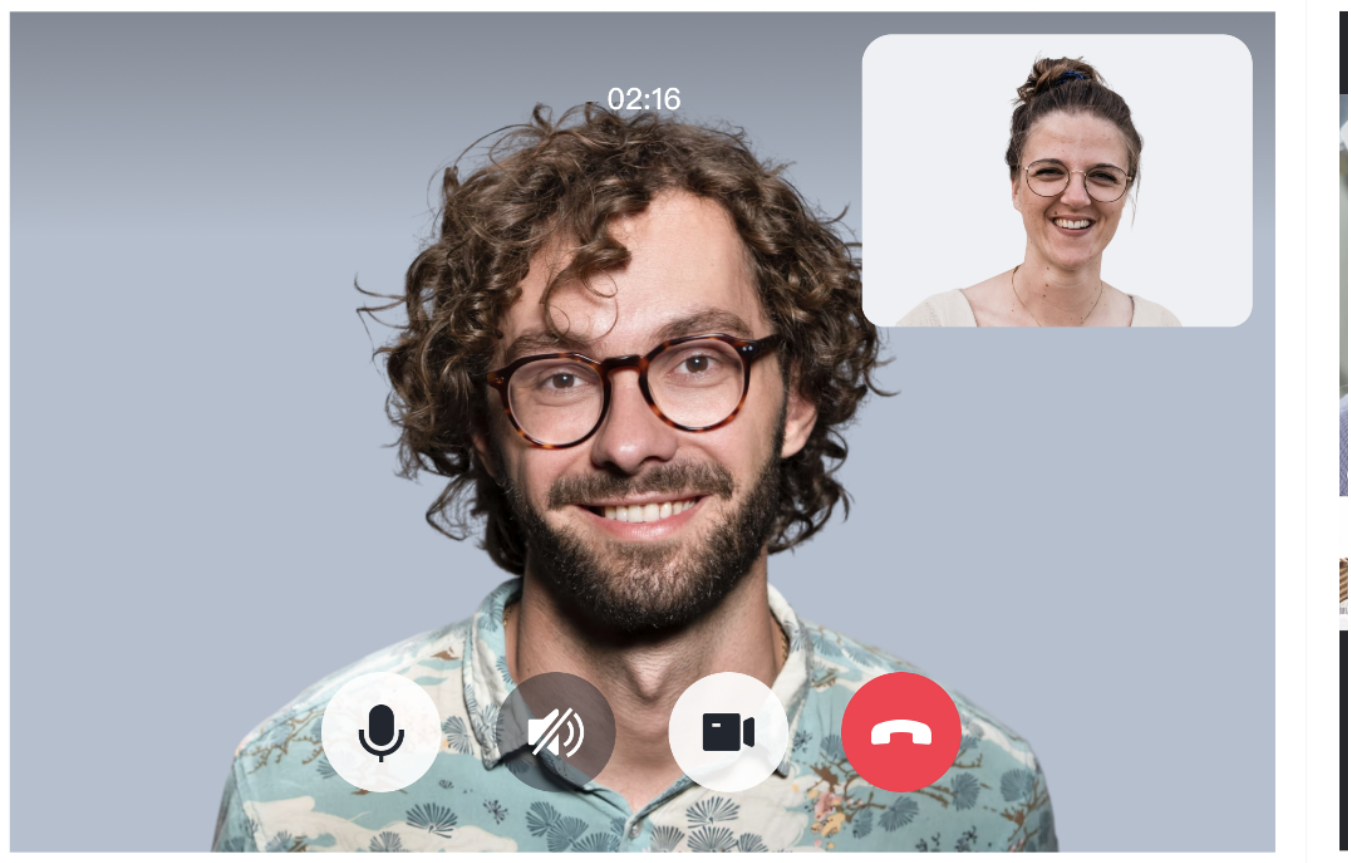
2.2 单击**发起通话**。


userId：olivia
1V1 通话
用户ID
mike
媒体类型
视频通话
语音通话
通话设置 ›
发起通话
迈克
等待对方接受邀请
麦克风已开
挂断
扬声器已关
userId: mike
迈克
用户ID: mike
实时音视频
1V1 通话
群组通话
挂断

Web

最近更新时间：2024-08-09 22:25:01

本文将介绍如何快速实现音视频通话演示。您将在10分钟内完成以下关键步骤，并最终获得一个具有全面用户界面的视频通话功能。

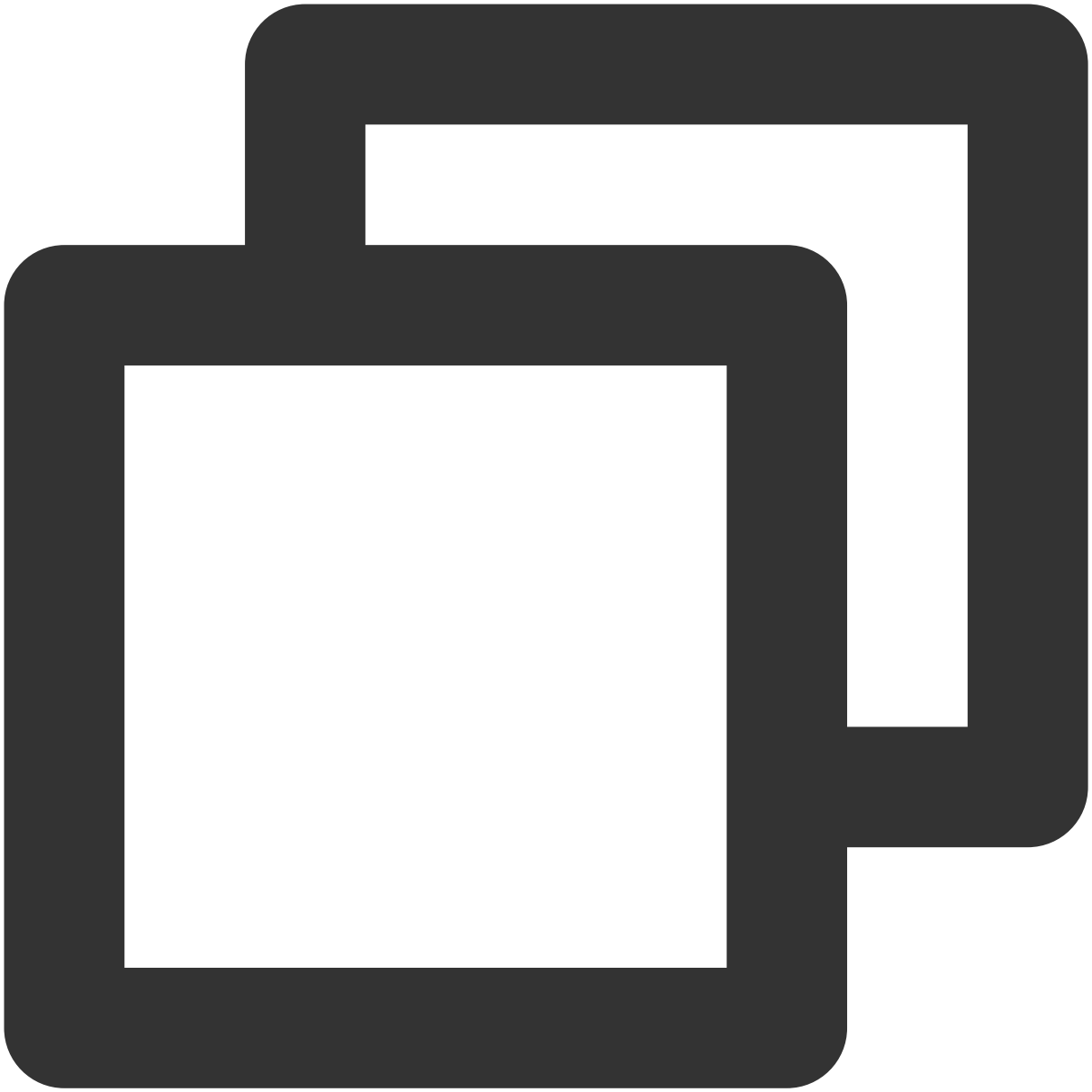


环境准备

Node.js version 16+.
Modern browser, supporting WebRTC API.

步骤一：下载 Demo

1. 打开终端，克隆仓库。

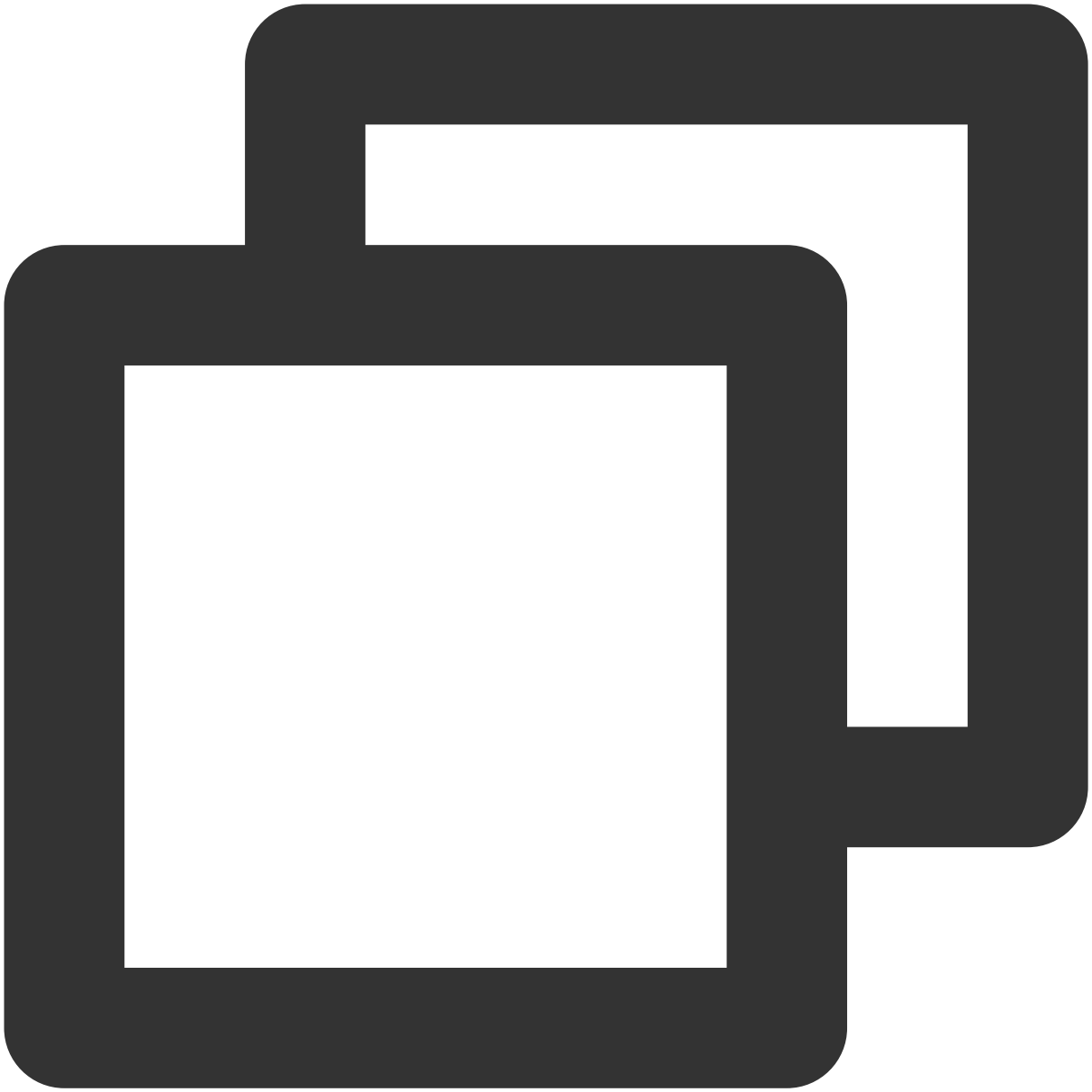


```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

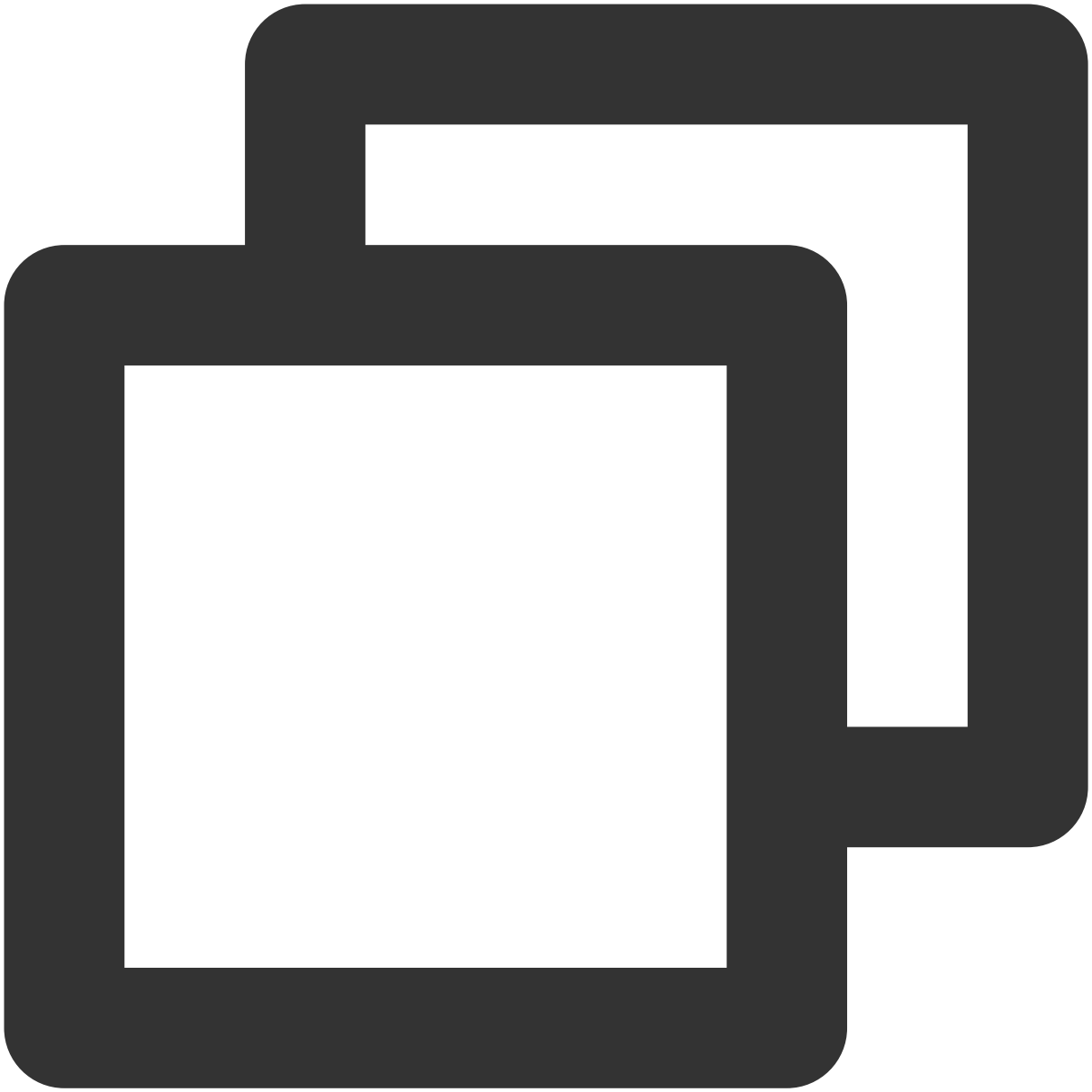
2. 安装依赖。

React

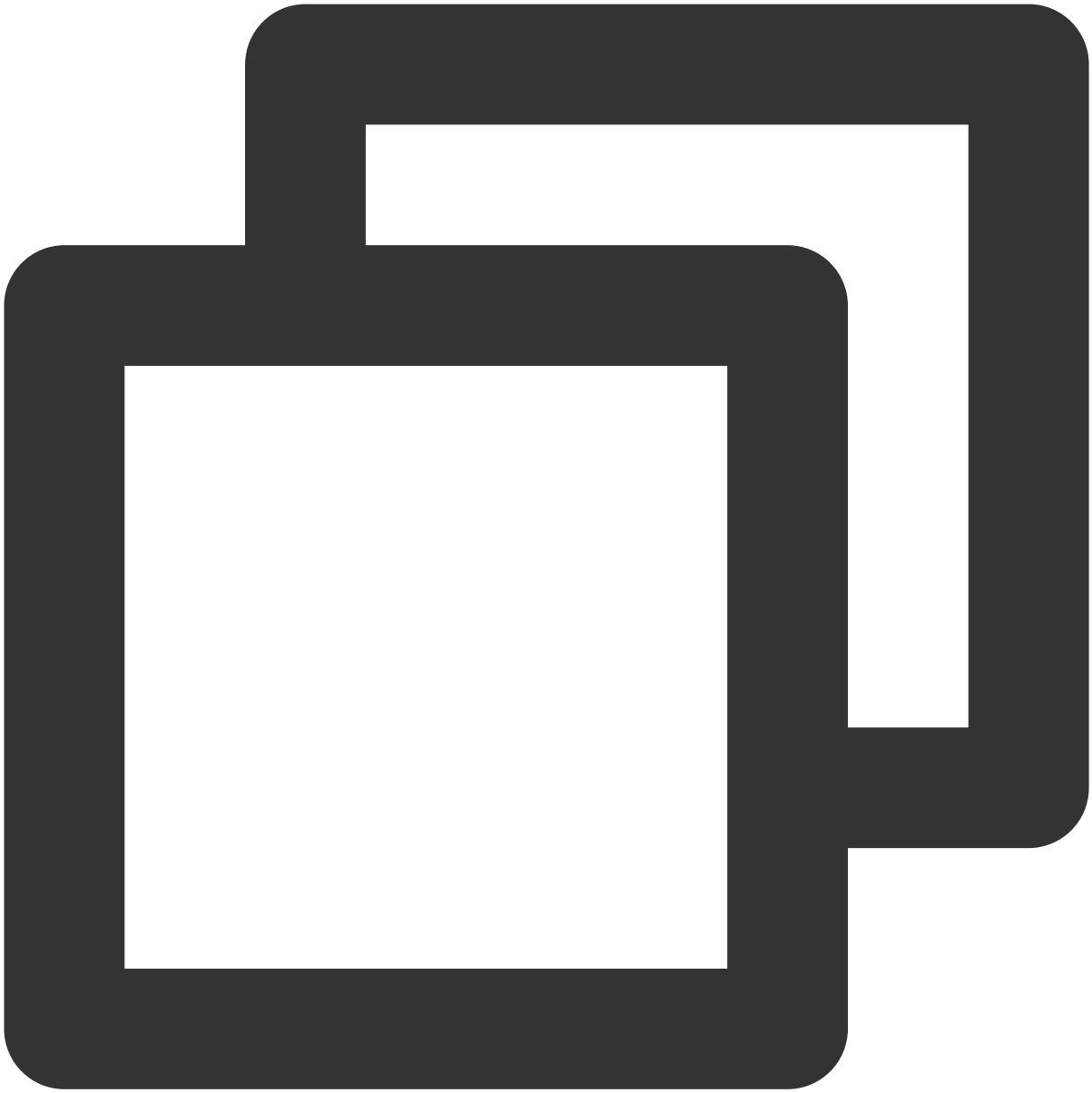
Vue3



```
cd ./TUICallKit/Web/basic-react
```



```
cd ./TUICallKit/Web/basic-vue3
```



```
npm install
```

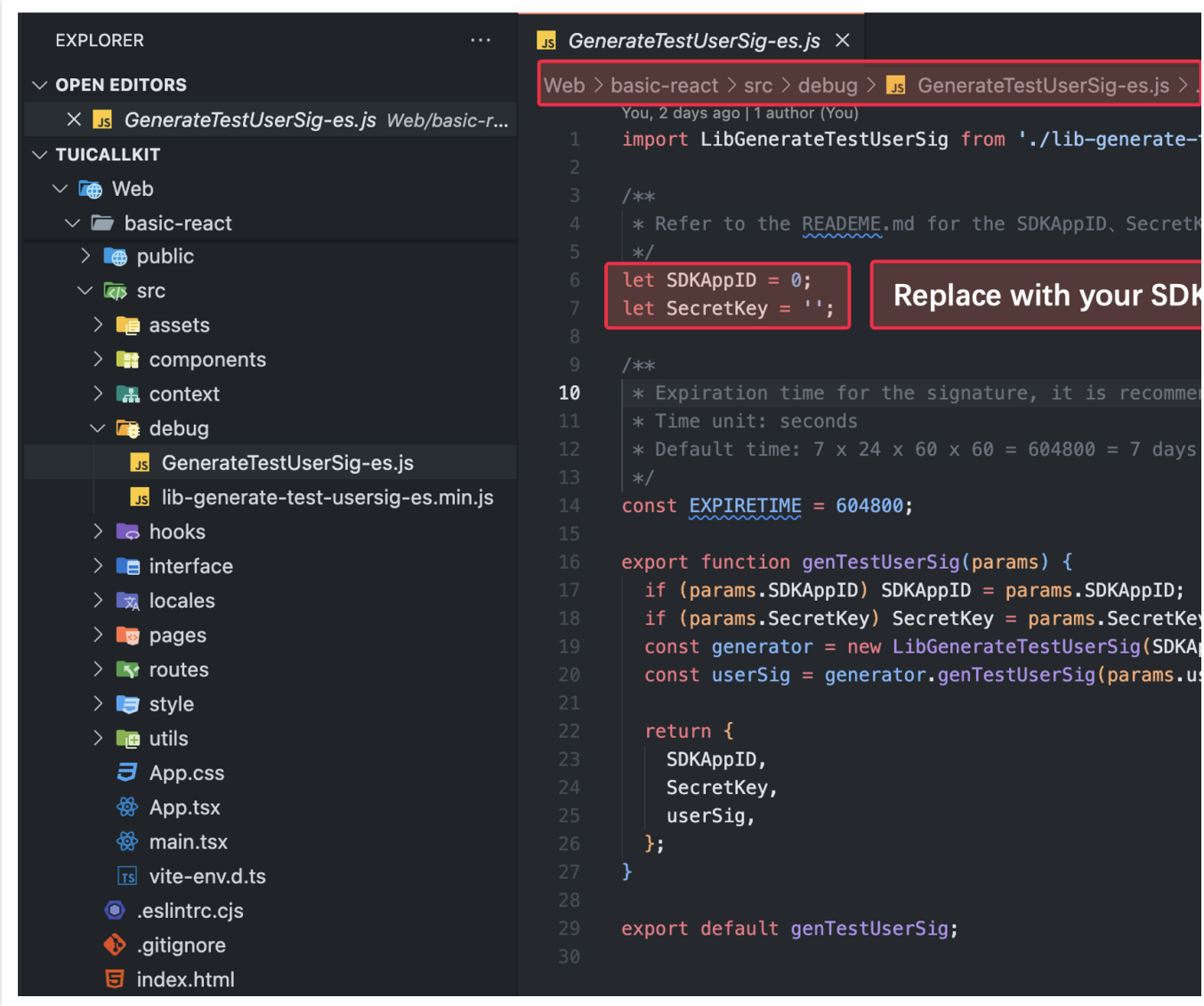
步骤二：配置 Demo

进入开通服务页面 获取 **`SDKAppID、SDKSecretKey`** ，在 `GenerateTestUserSig-es.js` 文件填写。

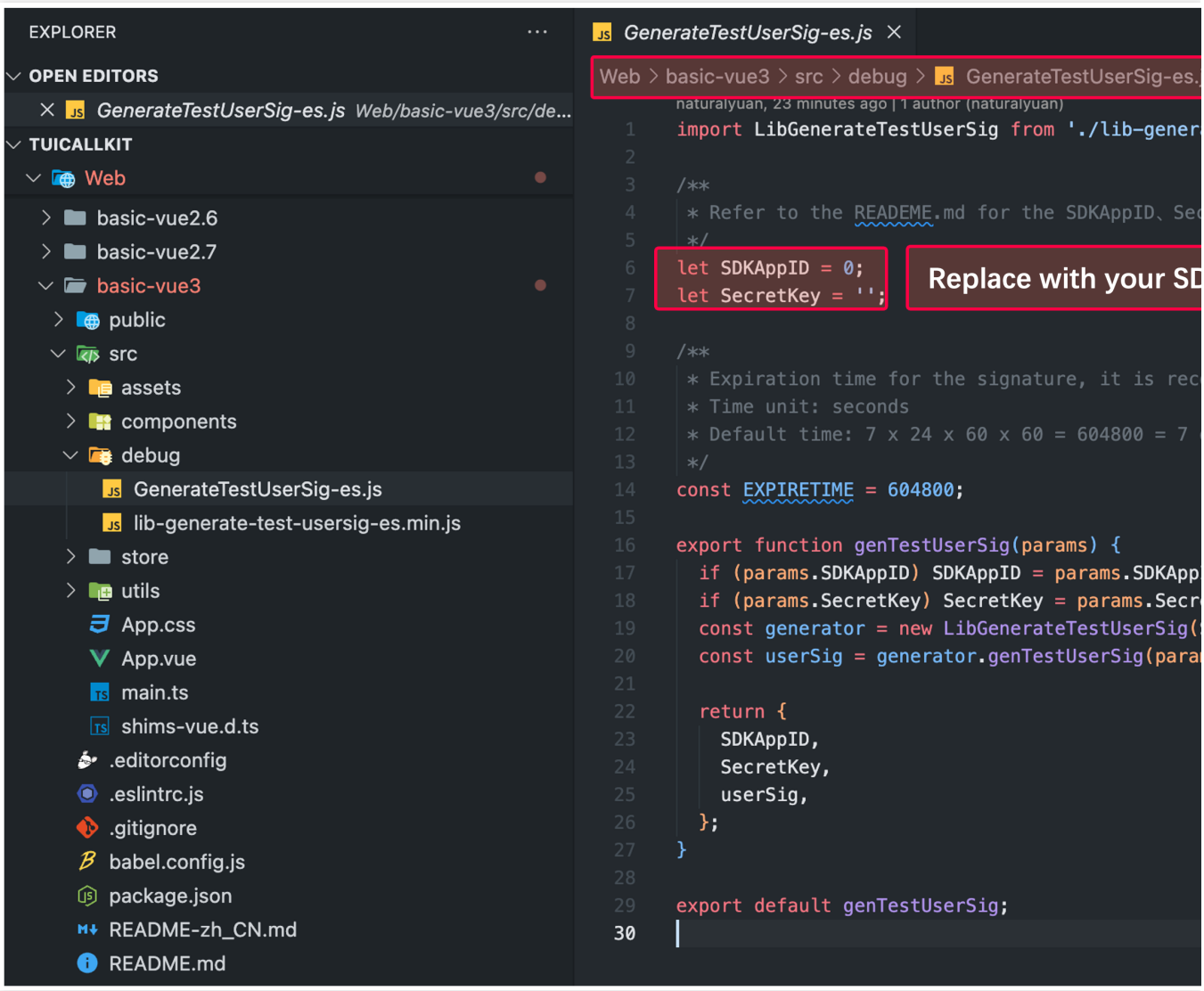
React

Vue3

文件路径： `TUICallKit/Web/basic-react/src/debug/GenerateTestUserSig-es.js`



文件路径： `TUICallKit/Web/basic-vue3/src/debug/GenerateTestUserSig-es.js`

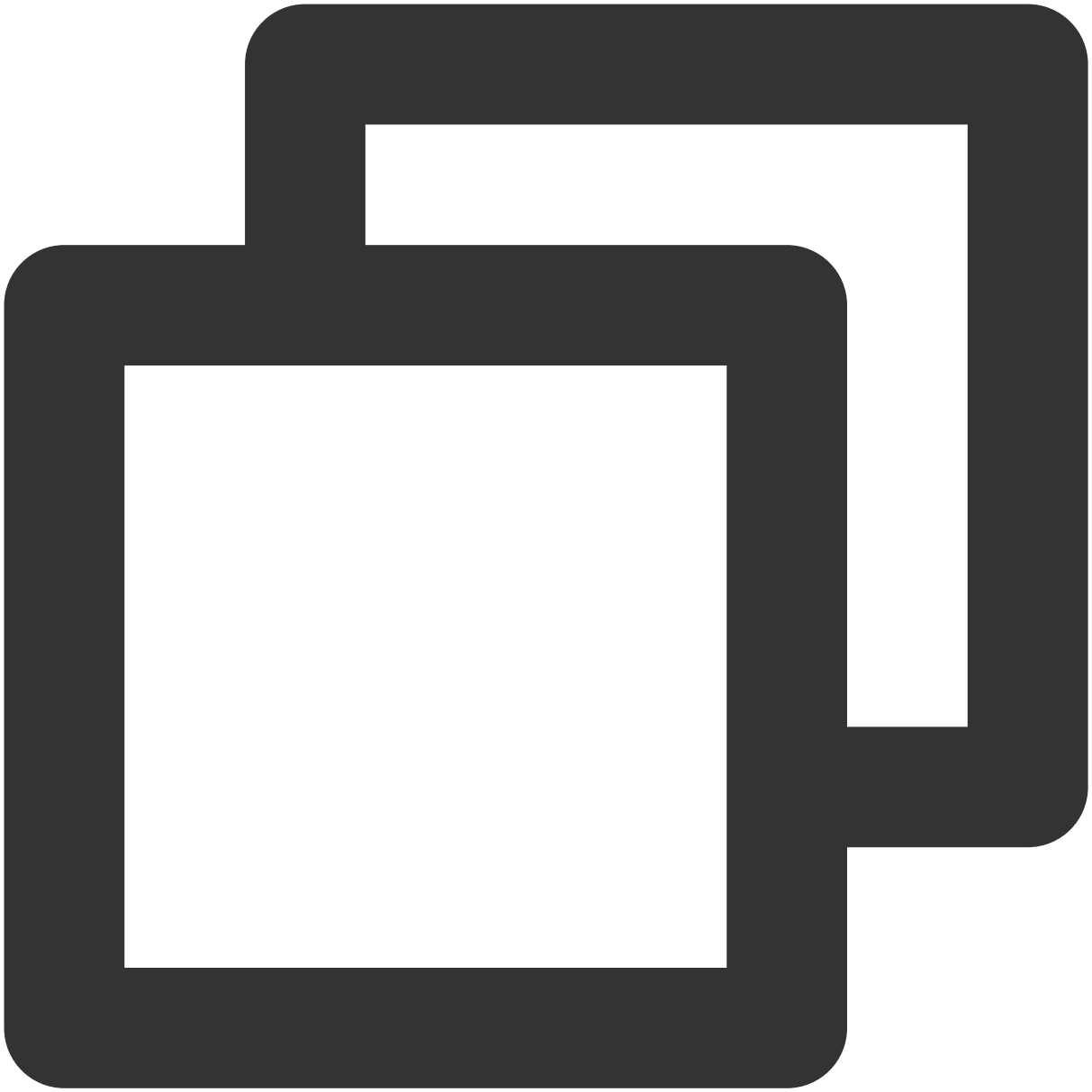


步骤三：运行 Demo

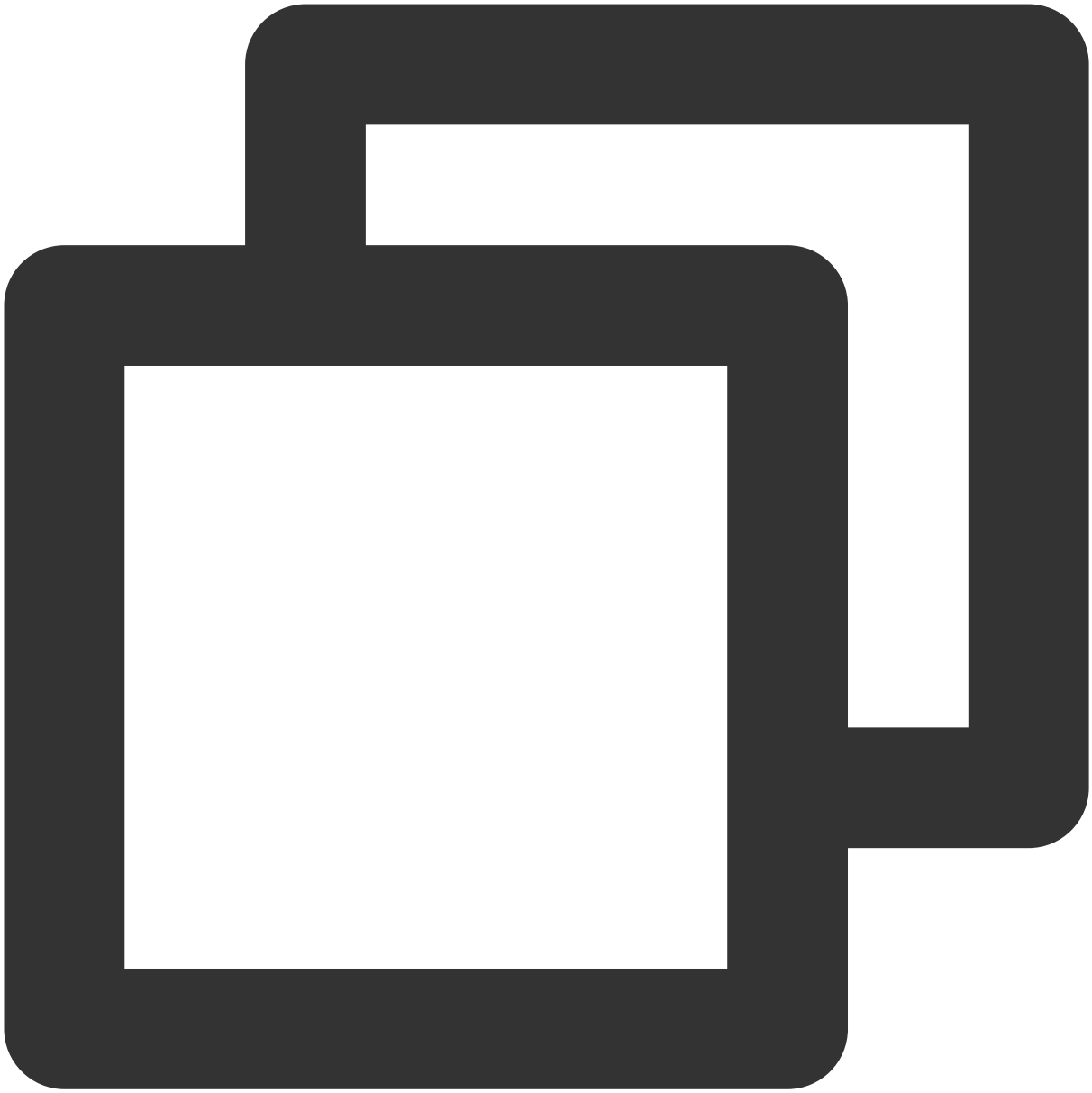
打开终端，复制示例命令运行 Demo。

TUICallKit/Web/basic-react

TUICallKit/Web/basic-vue3



```
npm run dev
```



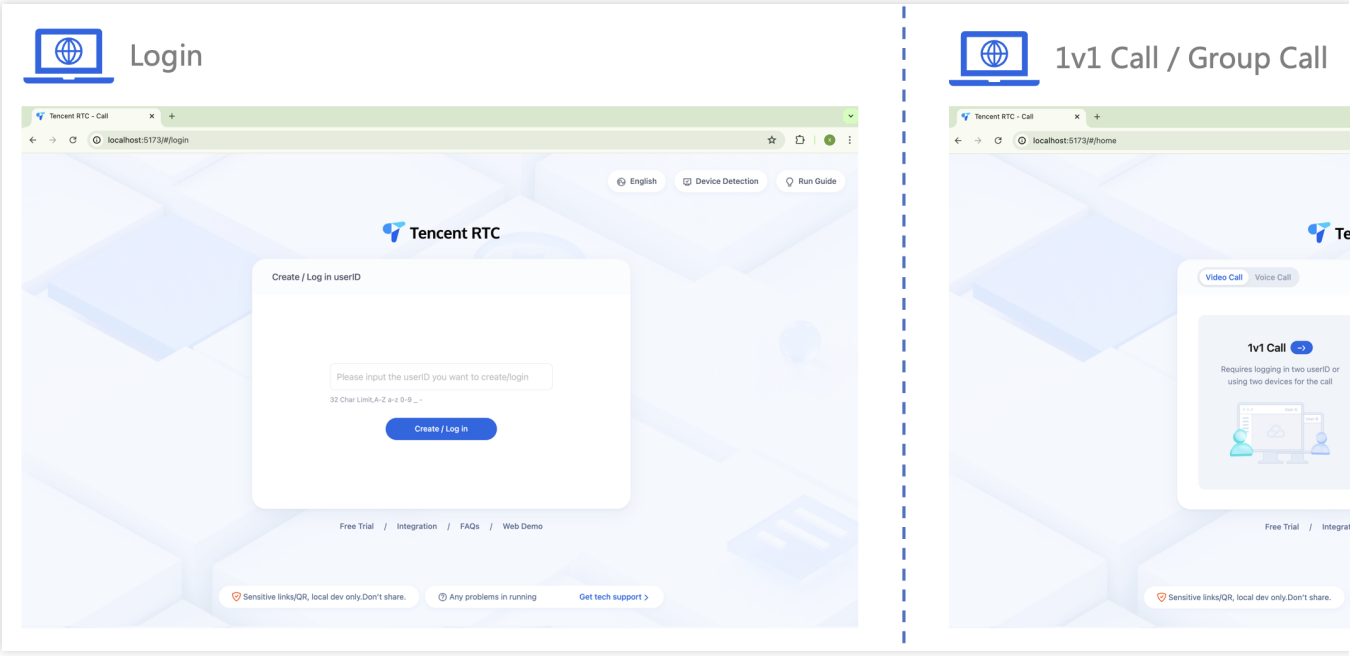
```
npm run serve
```

警告：

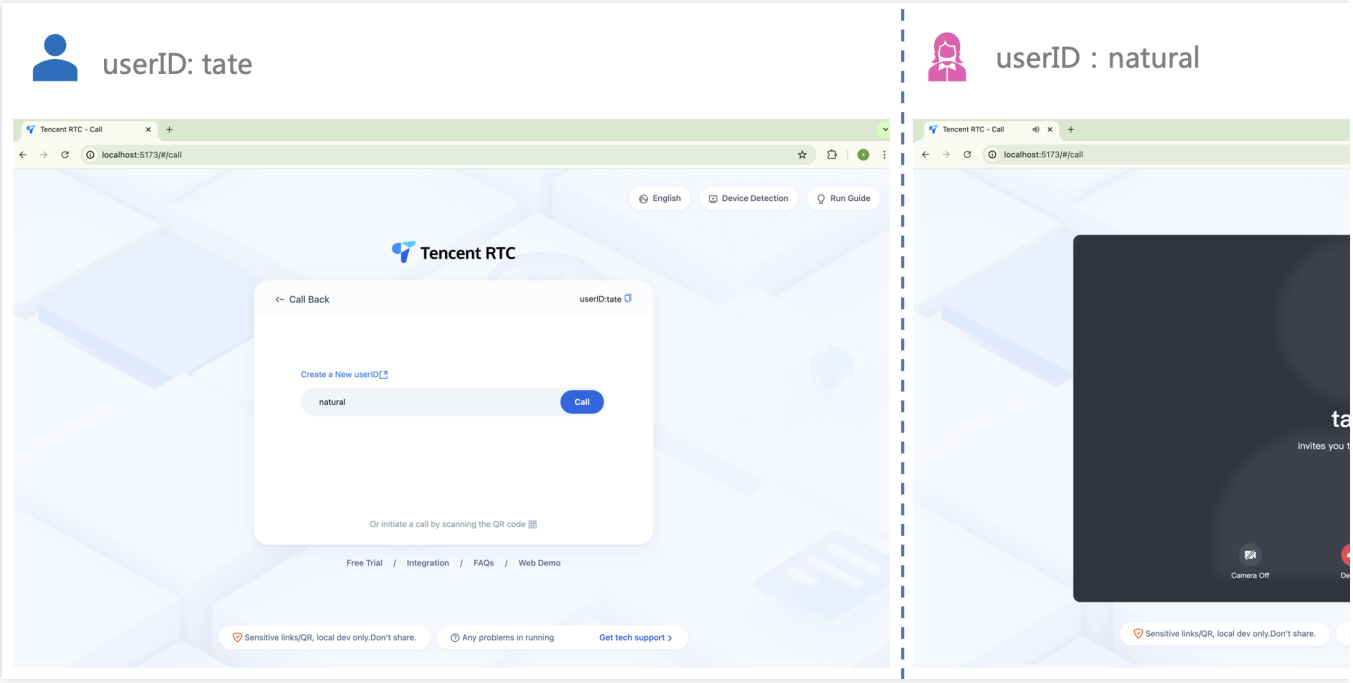
本地环境请在 localhost 协议下访问，公网体验请在 HTTPS 协议下访问，具体参见 网络访问协议说明。

步骤四：拨打您的第一通电话

1. 打开浏览器页面，输入项目运行地址，登录 userID(由您定义) 。



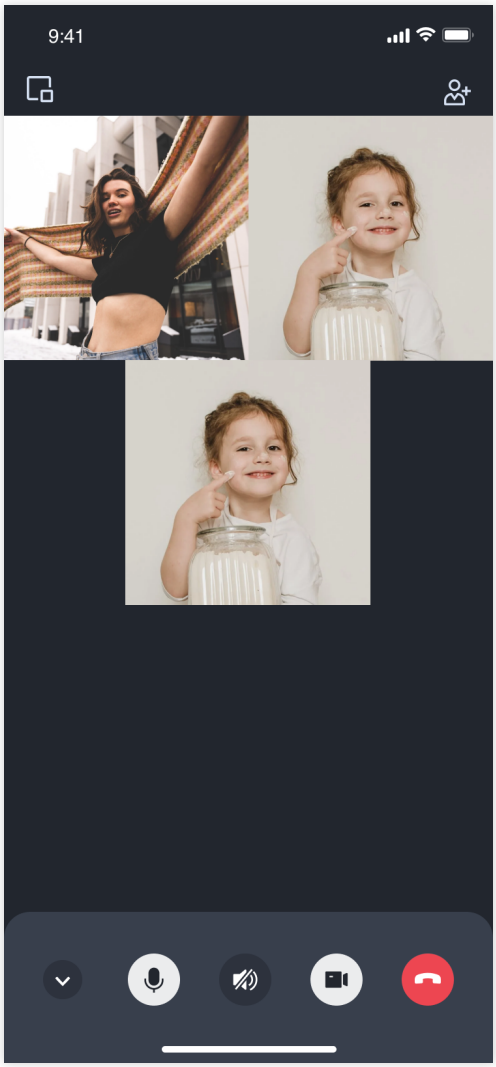
2. 输入被叫方的 userID, 单击 Call 体验您的第一通话。



Flutter

最近更新时间：2024-06-18 17:27:24

本文将为您介绍如何快速跑通音视频通话 Demo。跟随本文档，您可以在 10 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的音视频通话功能。

视频通话	群组通话
	

环境准备

Flutter

Flutter 3.0 及以上版本。

Android

Android Studio 3.5及以上版本。

Android 4.1 及以上版本的 Android 设备。

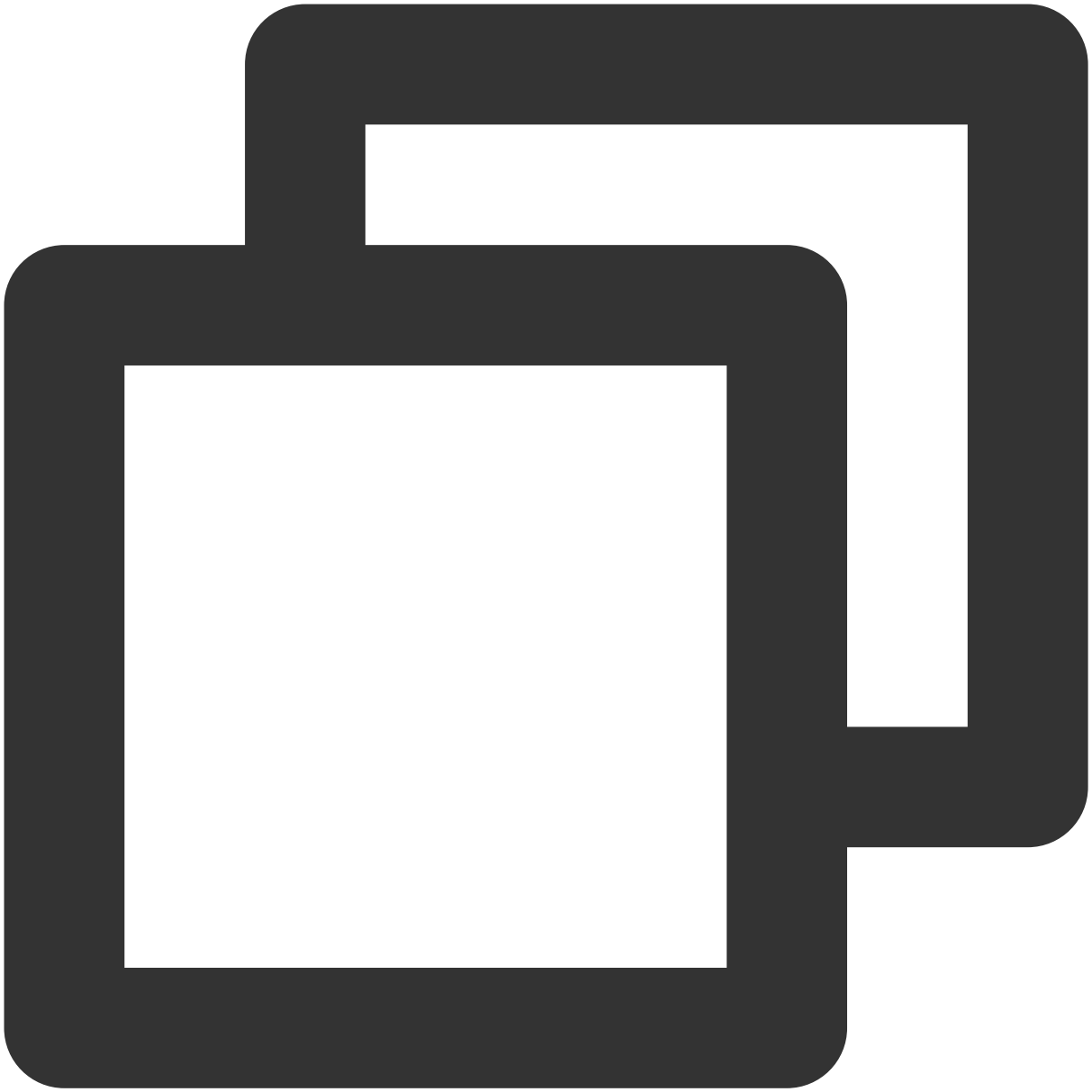
iOS

Xcode 13.0及以上版本。

请确保您的项目已设置有效的开发者签名。

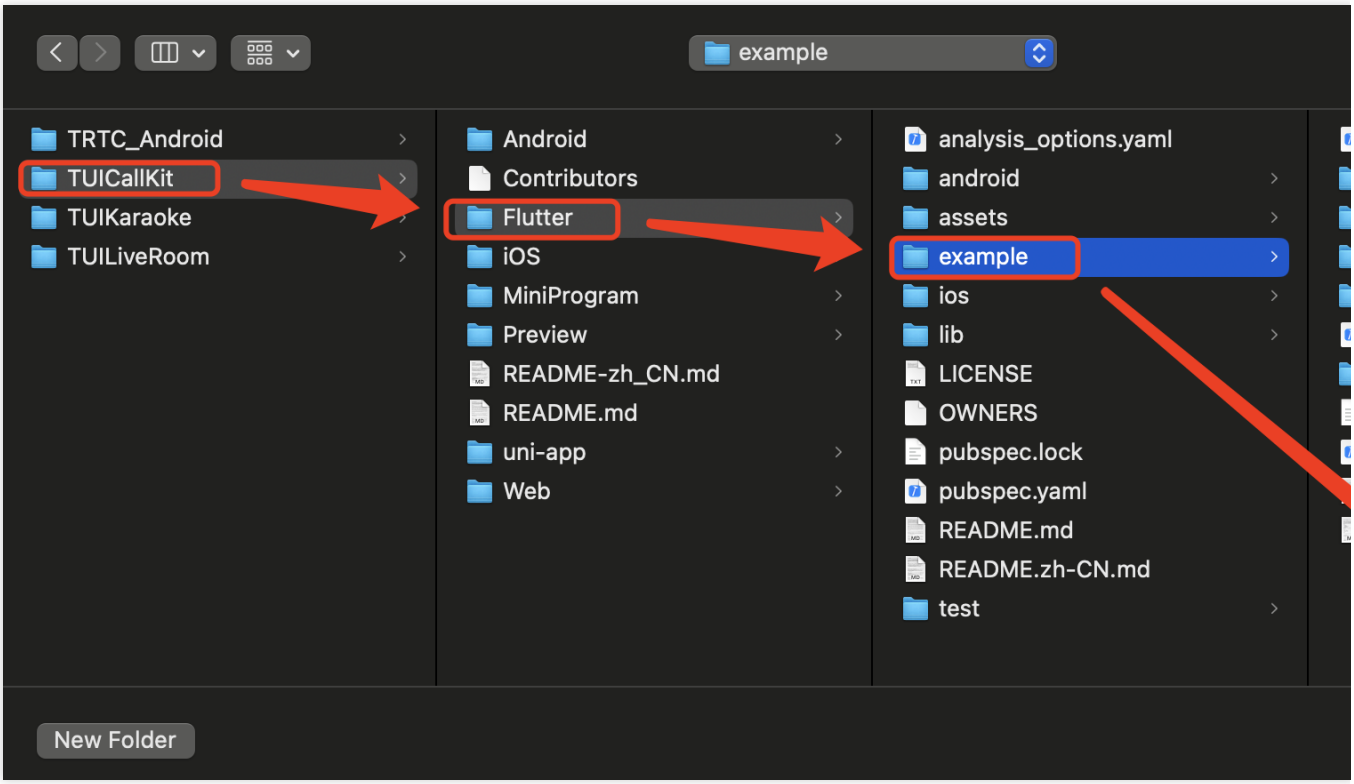
步骤一：下载 Demo

1. 从 github 下载 TUICallKit Demo 源码，或者直接在命令行运行以下命令：



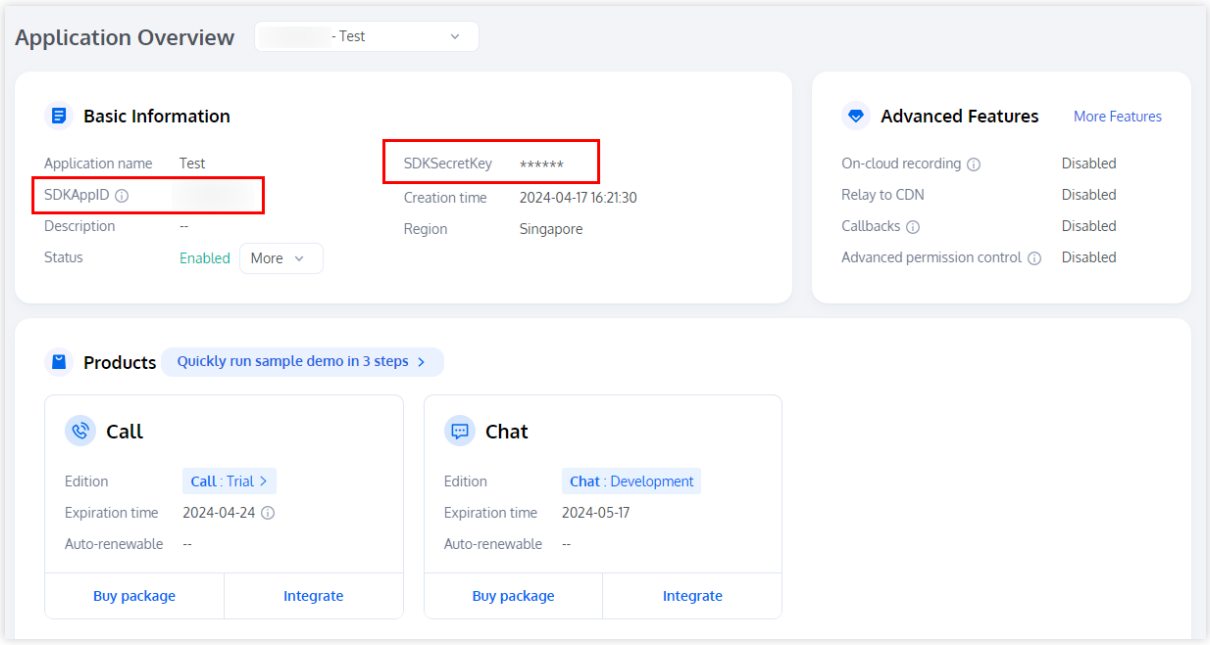
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. 通过 Android Studio 或 VSCode 打开 TUICallKit Flutter 的 example 。以下流程将以 Android Studio 为例：

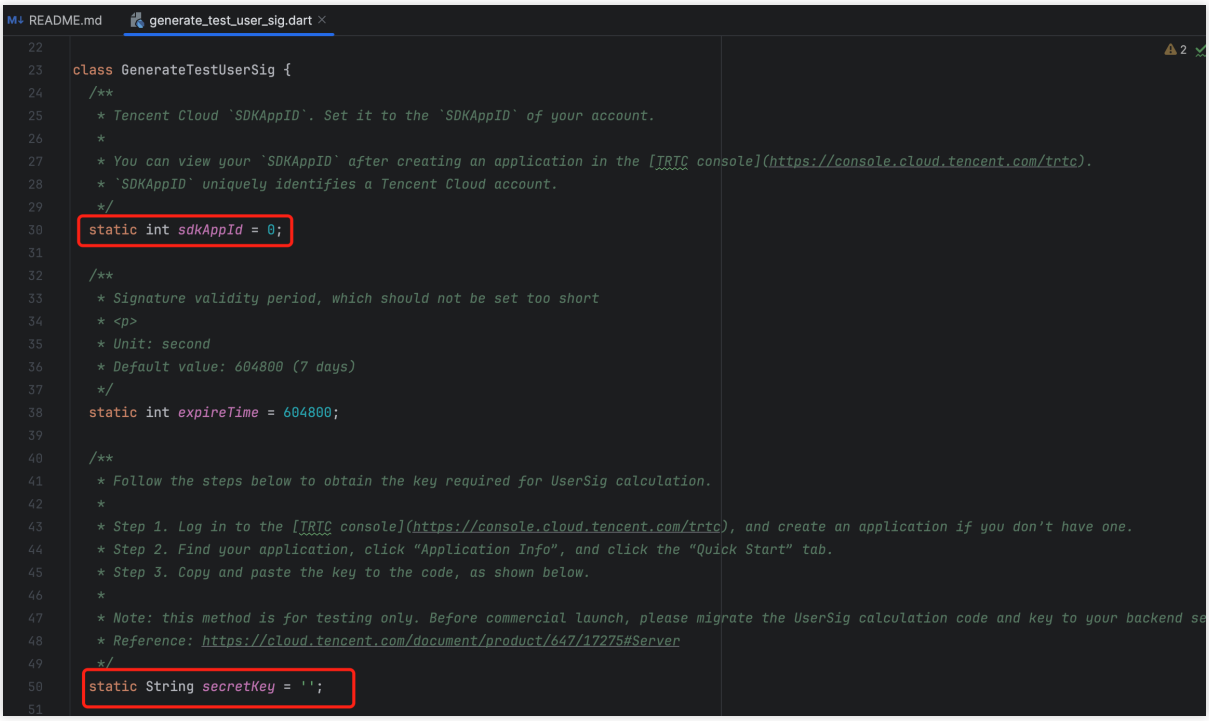


步骤二：配置 Demo

1. 开通音视频服务，获取 **SDKAppID** 和 **SDKSecretKey** 。

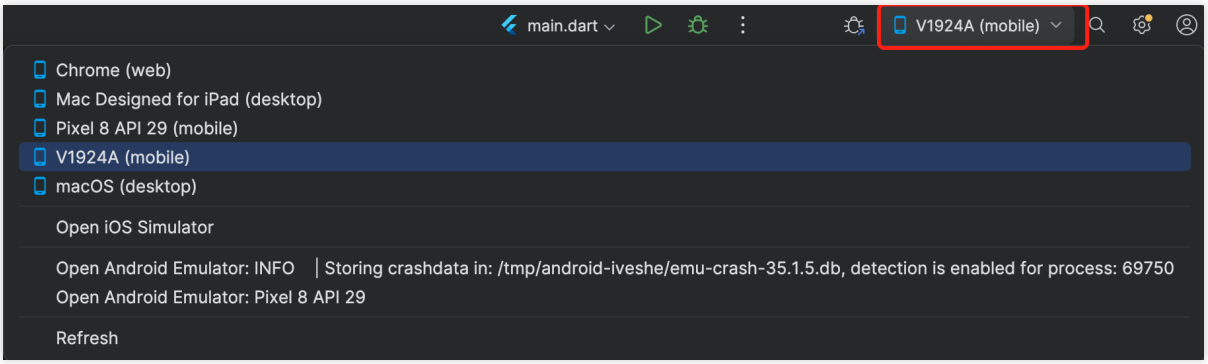


2. 打开 `Flutter/example/lib/debug/generate_test_user_sig.dart` 文件，将开通服务时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

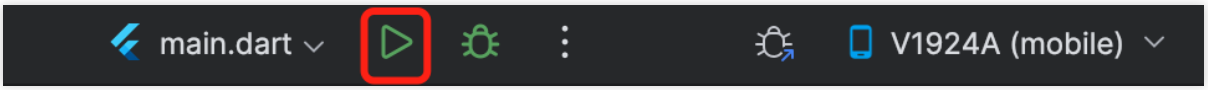


步骤三：跑通 Demo

1. 在 Android Studio 右上角如下图所示处选择您要将 Demo 运行的设备：



2. 选择完成后点击运行，将 TUICallKit Android Demo 运行到目标设备上。



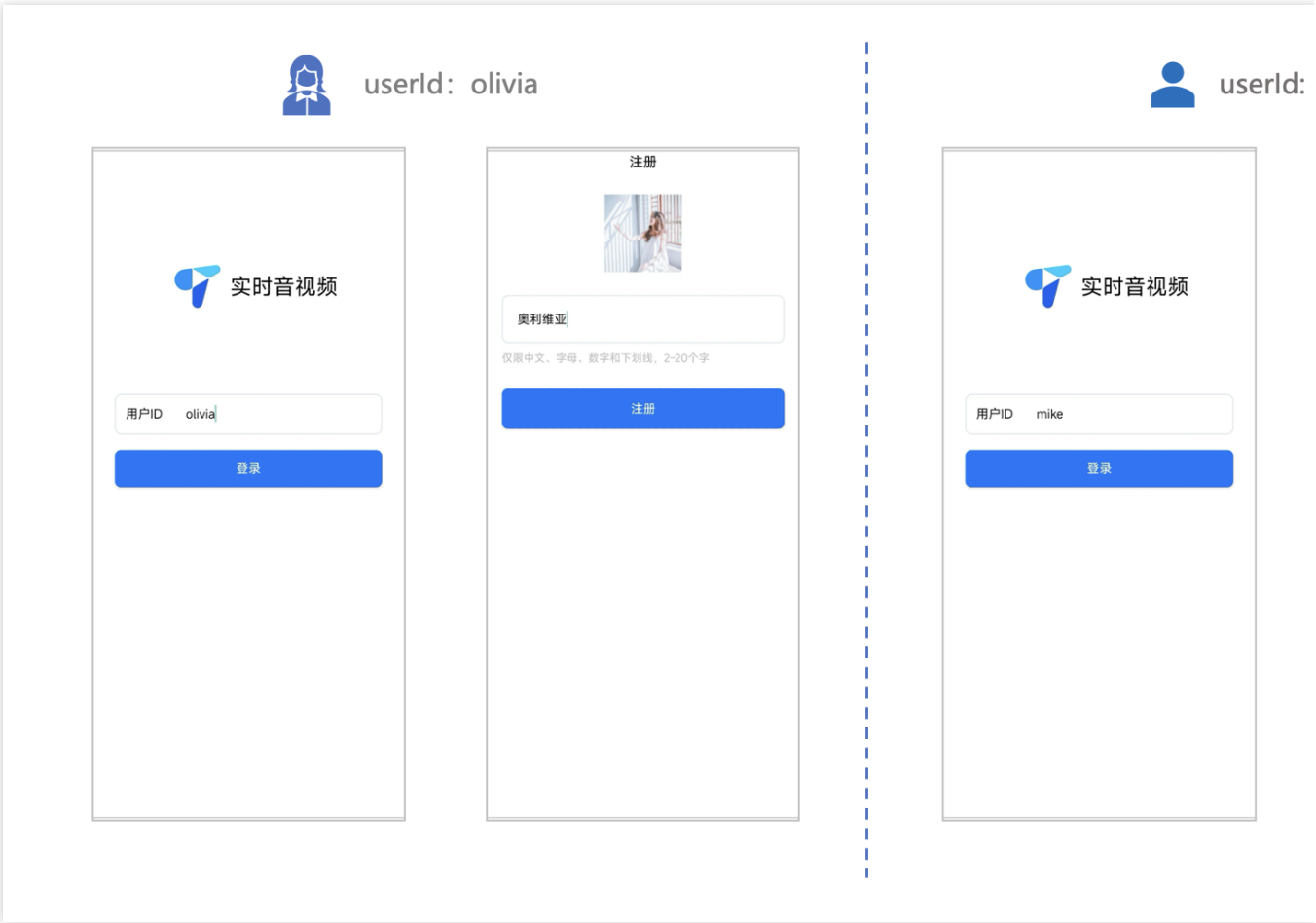
拨打第一个电话

注意：

为了使您可以体验完整的音视频通话流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主叫，一方作为被叫。

1. 登录 & 注册

请您在 `用户ID` 处输入的 ID 。如果你当前的 UserID 未曾被使用过，此时会进入到**注册**界面，您可以在该界面为自己设置头像及昵称。



注意：

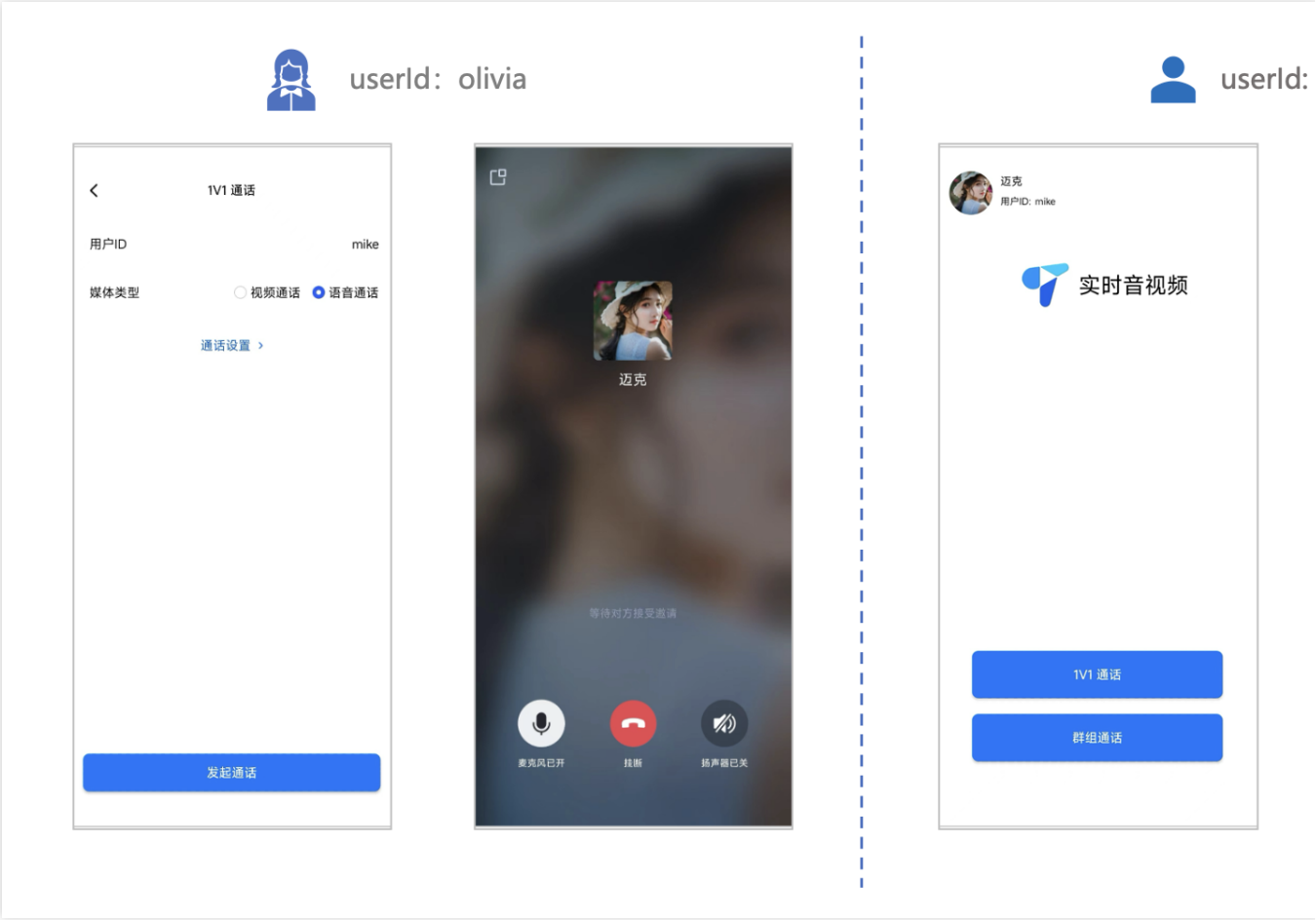
尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建

议您在调试的时候设置一些辨识度高的 UserID。

2. 拨打电话

2.1 主叫方请点击界面上的 **1V1 通话**，在弹出的界面中输入被叫方的 UserID 并选择您想要的通话类型。

2.2 单击**发起通话**。

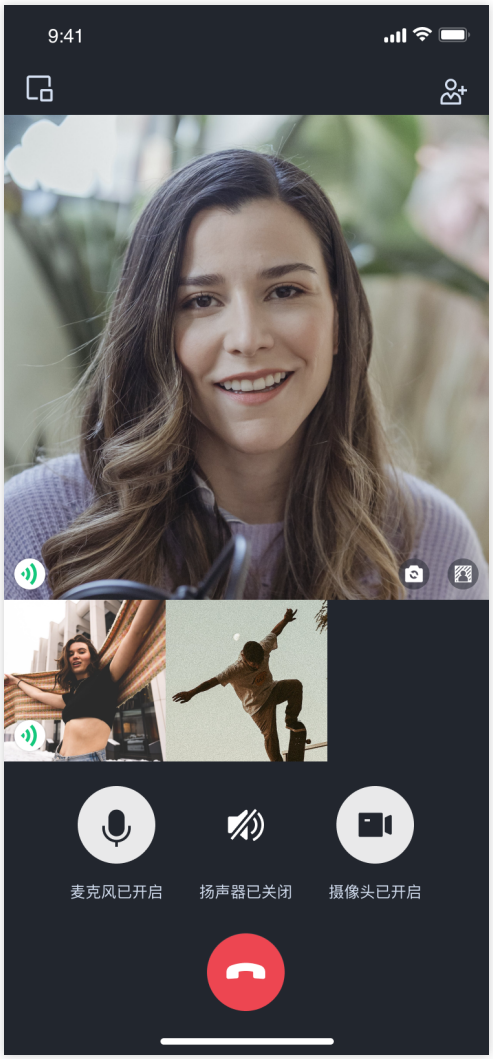


快速接入（TUICallKit）

Android

最近更新时间：2024-05-31 16:42:21

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。

1v1 视频通话	群组通话
	

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。具体步骤详见 开通服务（TUICallKit）。开通服务后，请记录 `SDKAppID` 和 `SDKSecretKey`，在后续的步骤中会用到。

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，如下图所示。

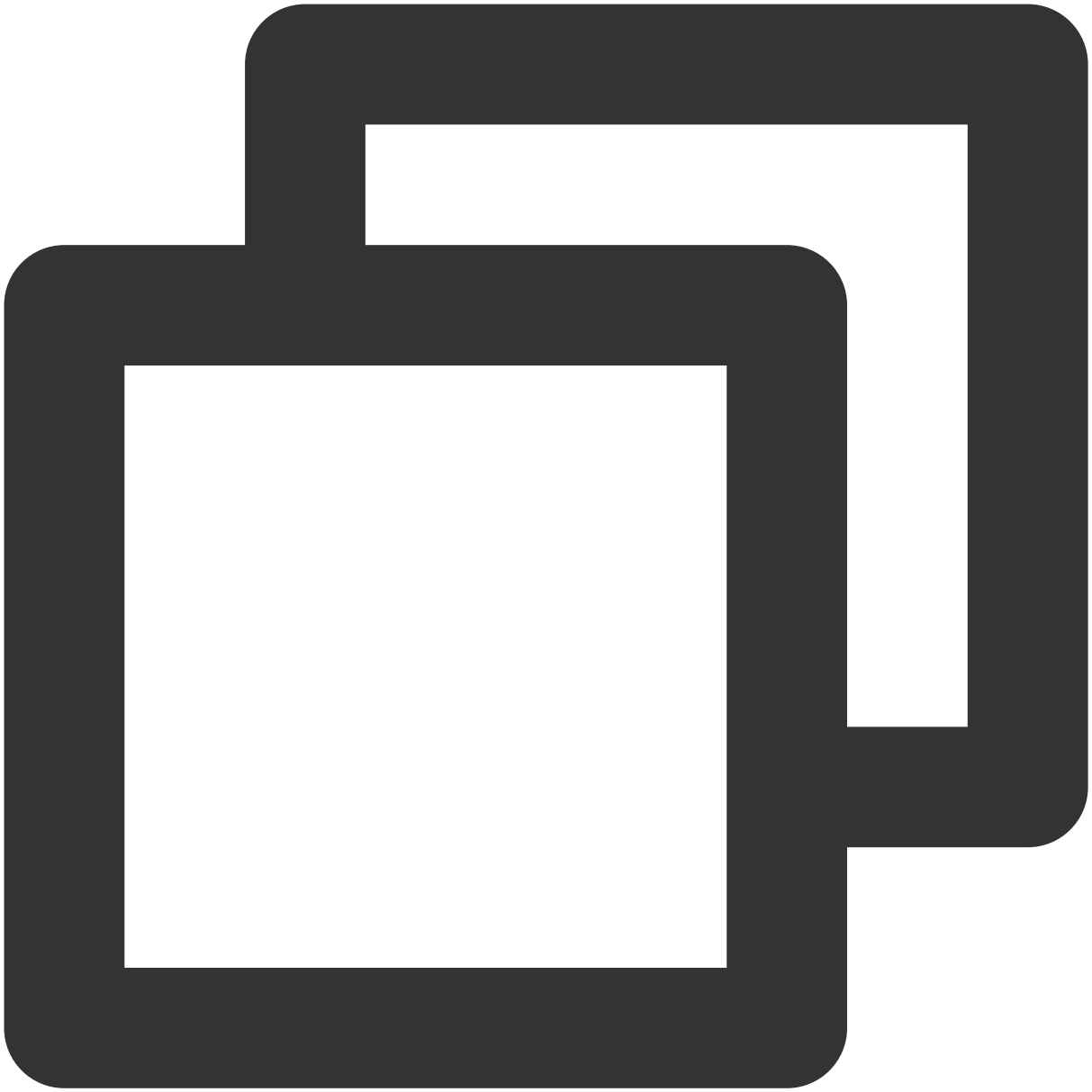


步骤三：工程配置

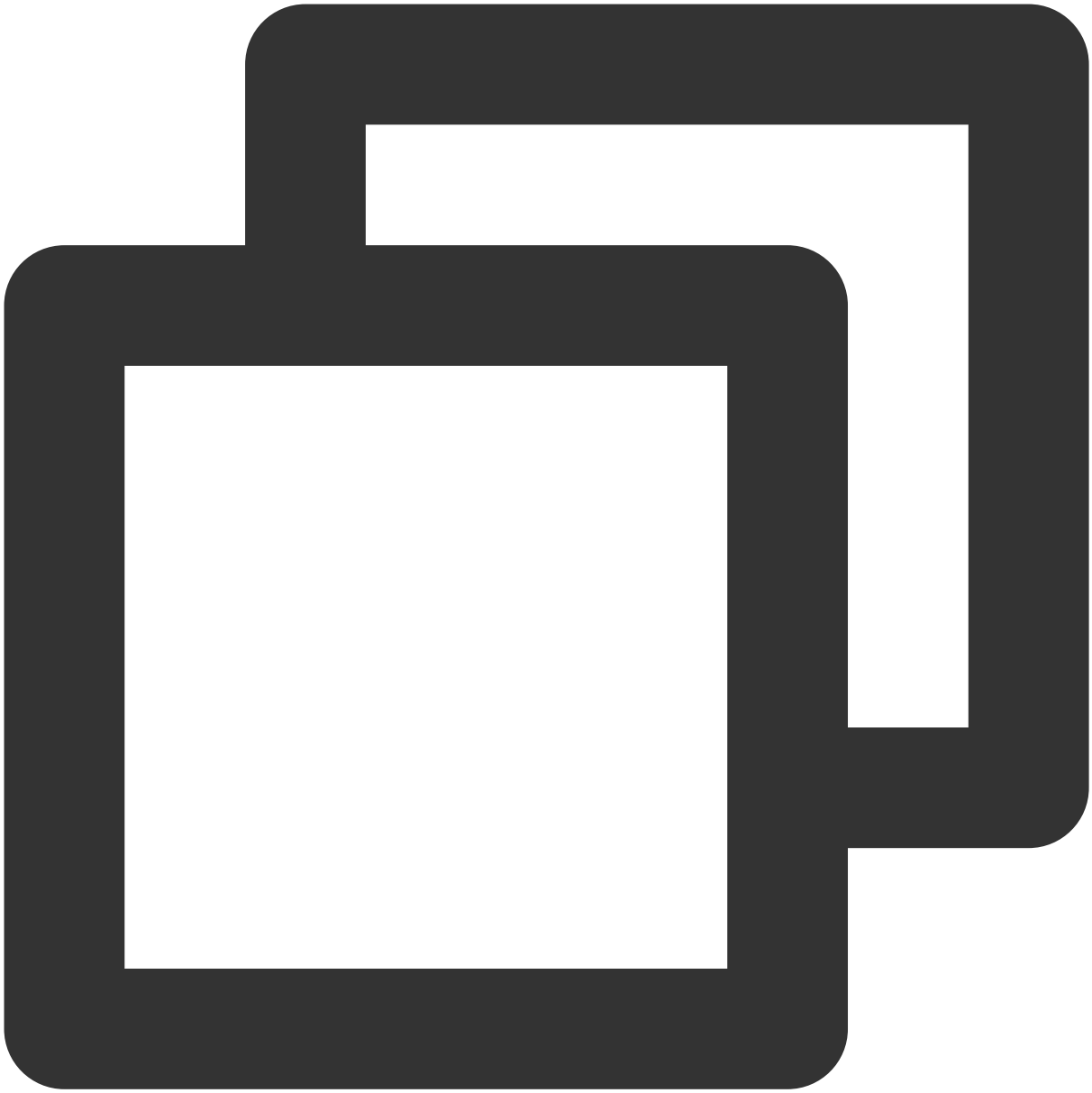
1. 在工程根目录下找到 `settings.gradle.kts`（或`settings.gradle`）文件，在其中增加如下代码，导入 `tuicallkit-kt` 组件到项目中。

setting.gradle.kts

settings.gradle



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

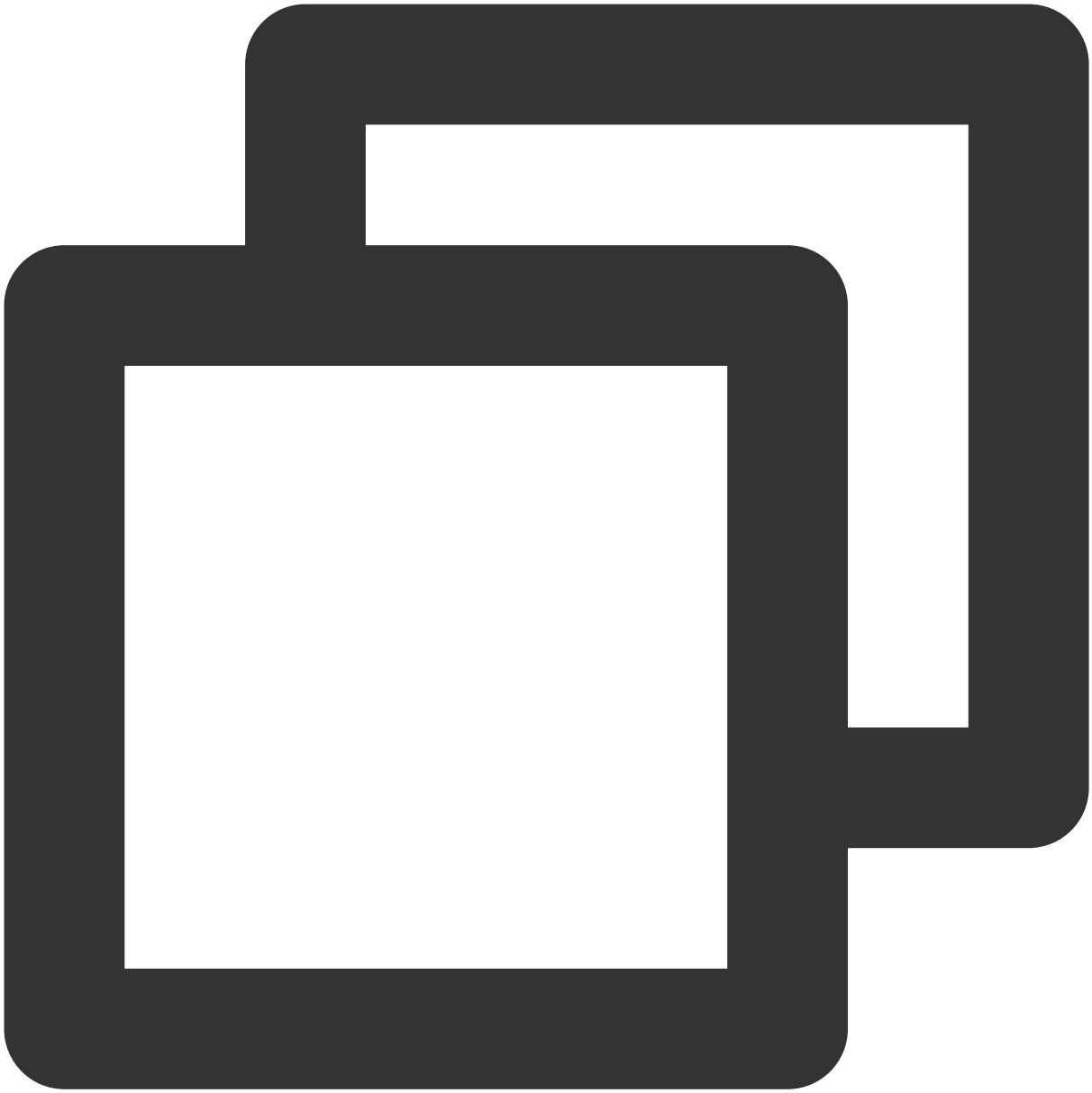


```
include ':tuicallkit-kt'
```

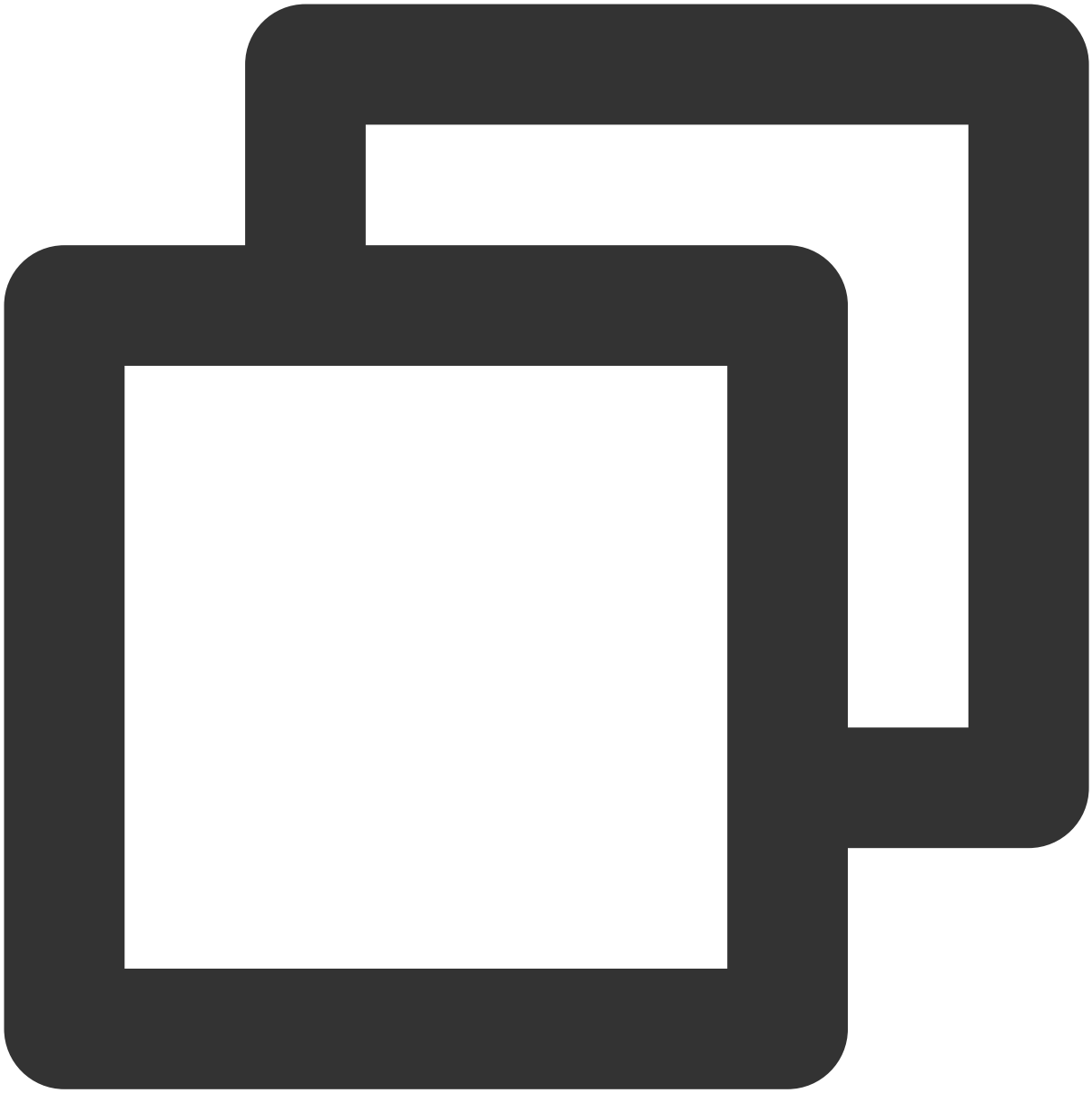
2. 在 app 目录下找到 `build.gradle.kts(或build.gradle)` 文件，在 `dependencies` 中增加如下代码，声明当前 app 对新加入的组件的依赖。

build.gradle.kts

build.gradle



```
dependencies {
    api(project(":tuicallkit-kt"))
}
```

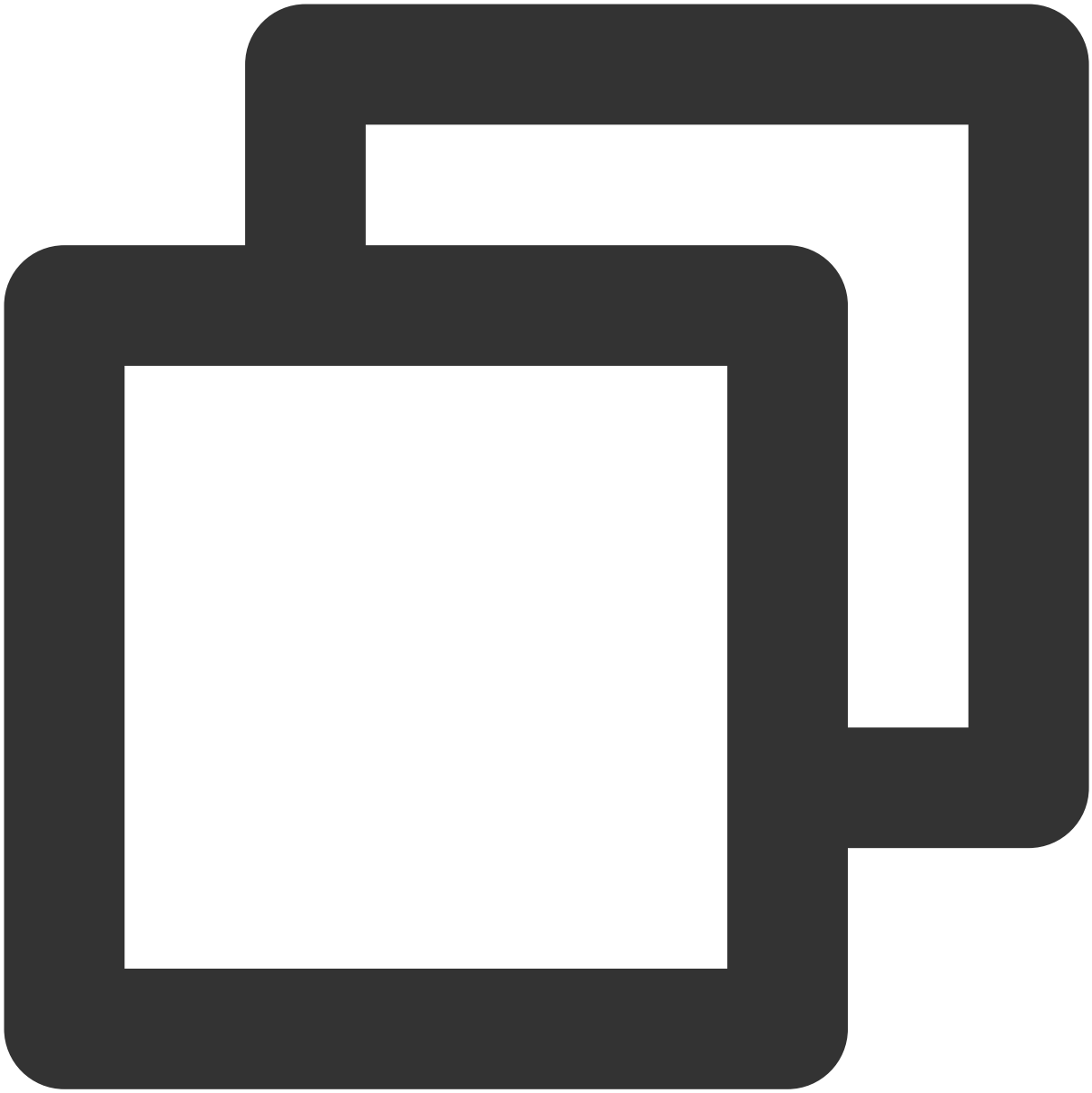


```
dependencies {
    api project(':tuicallkit-kt')
}
```

说明

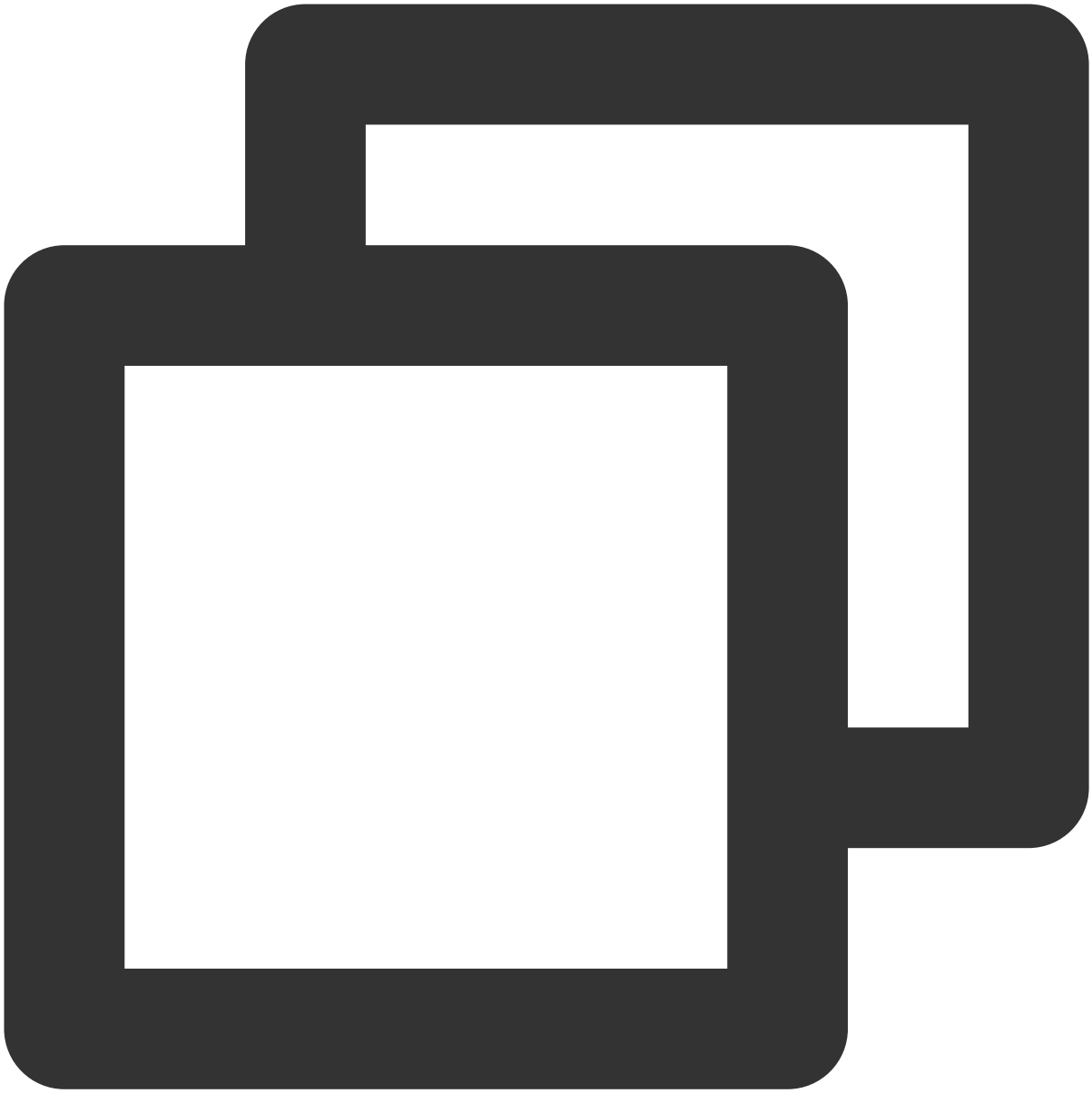
TUICallKit 工程内部已经默认依赖：`TRTC SDK`、`IM SDK`、`tuicallengine` 以及公共库 `tuicore`，不需要开发者单独配置。如需进行版本升级，则修改 `tuicallkit-kt/build.gradle` 文件中的版本号即可。

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 app 目录下的 `proguard-rules.pro` 文件末尾添加如下代码。添加完后，点击右上角的“**Sync Now**”，同步代码。



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

4. 在 app目录下找到 `AndroidManifest.xml` 文件，在 application 节点中添加 `tools:replace="android:allowBackup"` ，覆盖组件内的设置，使用自己的设置。



```
// app/src/main/AndroidManifest.xml

<application
  android:name=".DemoApplication"
  android:allowBackup="false"
  android:icon="@drawable/app_ic_launcher"
  android:label="@string/app_name"
  android:largeHeap="true"
  android:theme="@style/AppTheme"
  tools:replace="android:allowBackup">
```

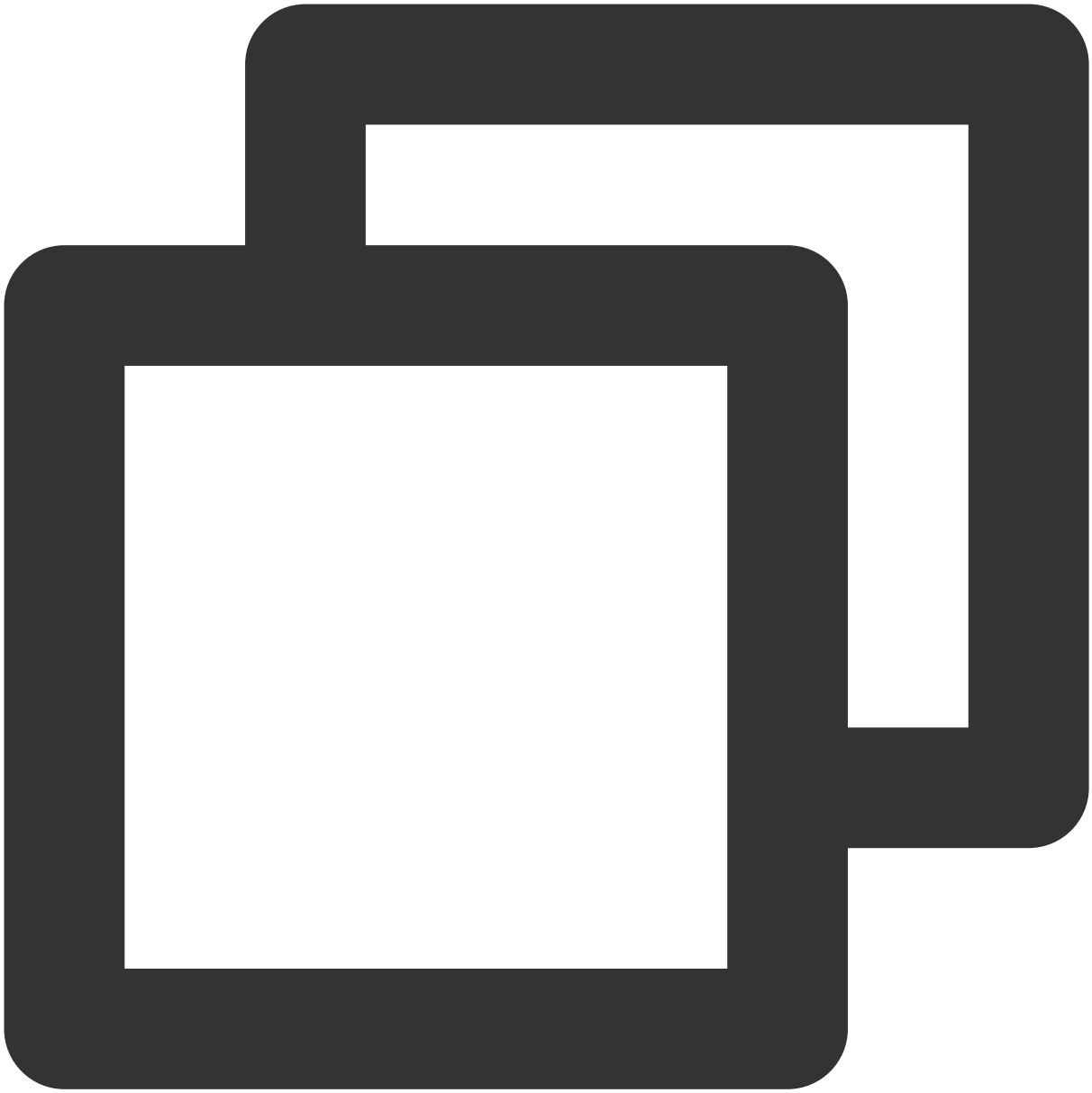
5. 建议您编译并运行一次。如果遇到问题，建议您尝试运行我们的 Github demo 项目。通过比对，您可以找出潜在的区别并解决遇到的问题。在接入和使用过程中，如果遇到问题，欢迎向我们反馈。

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。这一步骤至关重要，只有在成功登录之后，您才能正常使用 TUICallKit 提供的各项功能。

Kotlin

Java



```
import com.tencent.qcloud.tuicore.TUILogin
import com.tencent.qcloud.tuicore.interfaces.TUICallback
import com.tencent.qcloud.tuikit.tuicallkit.debug.GenerateTestUserSig

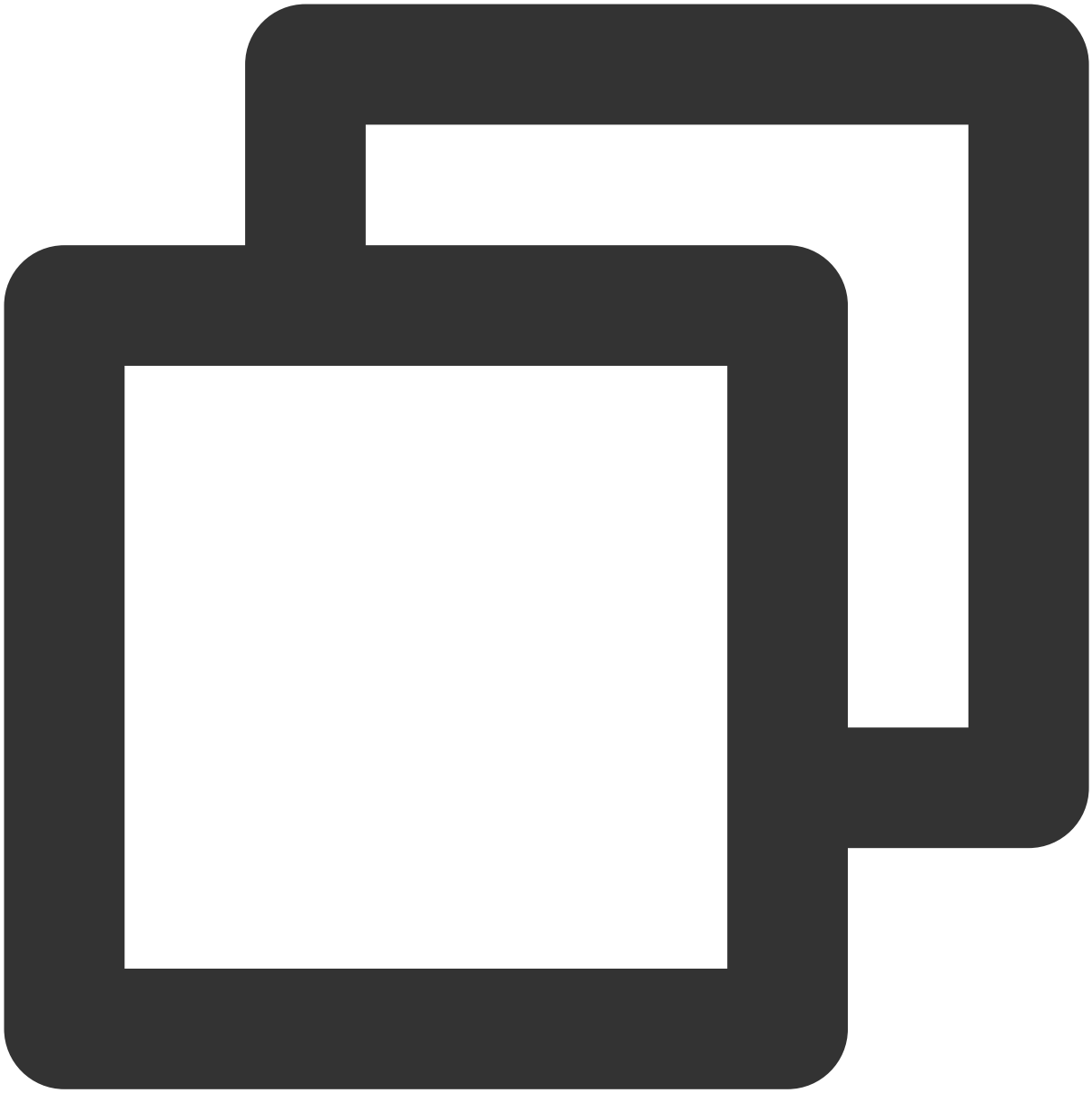
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // begin
        val userID = "denny"      // 请替换为您的 UserId
        val sdkAppID = 0          // 请替换为第一步在控制台得到的SDKAppID
```

```
        val secretKey = "****"    // 请替换为第一步在控制台得到的SecretKey

        val userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secretKe

        TUILogin.login(this, sdkAppId, userId, userSig, object : TUICallback() {
            override fun onSuccess() {
            }
            override fun onError(errorCode: Int, errorMessage: String) {
            }
         })
        // end
    }
}
```



```
import com.tencent.qcloud.tuicore.TUILogin;
import com.tencent.qcloud.tuicore.interfaces.TUICallback;
import com.tencent.qcloud.tuikit.tuicallkit.debug.GenerateTestUserSig;

public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //begin
        String userID = "denny";      // 请替换为您的UserId
```

```
        int sdkAppID = 0;              // 请替换为第一步在控制台得到的SDKAppID
        String secretKey = "****";     // 请替换为第一步在控制台得到的SecretKey

        String userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secre

        TUILogin.login(this, sdkAppId, userId, userSig, new TUICallback() {
            @Override
            public void onSuccess() {
            }

            @Override
            public void onError(int errorCode, String errorMessage) {
            }
        });
        //end
    }
}
```

参数	类型	说明
userID	String	客户根据自己的业务自定义用户 ID，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
sdkAppID	int	在 Tencent RTC 控制台 创建的音视频应用的唯一标识 SDKAppID。
secretKey	String	在 Tencent RTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

说明：

开发环境：如果您正在本地开发调试阶段，可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 userSig。该方法中 SDKSecretKey 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。

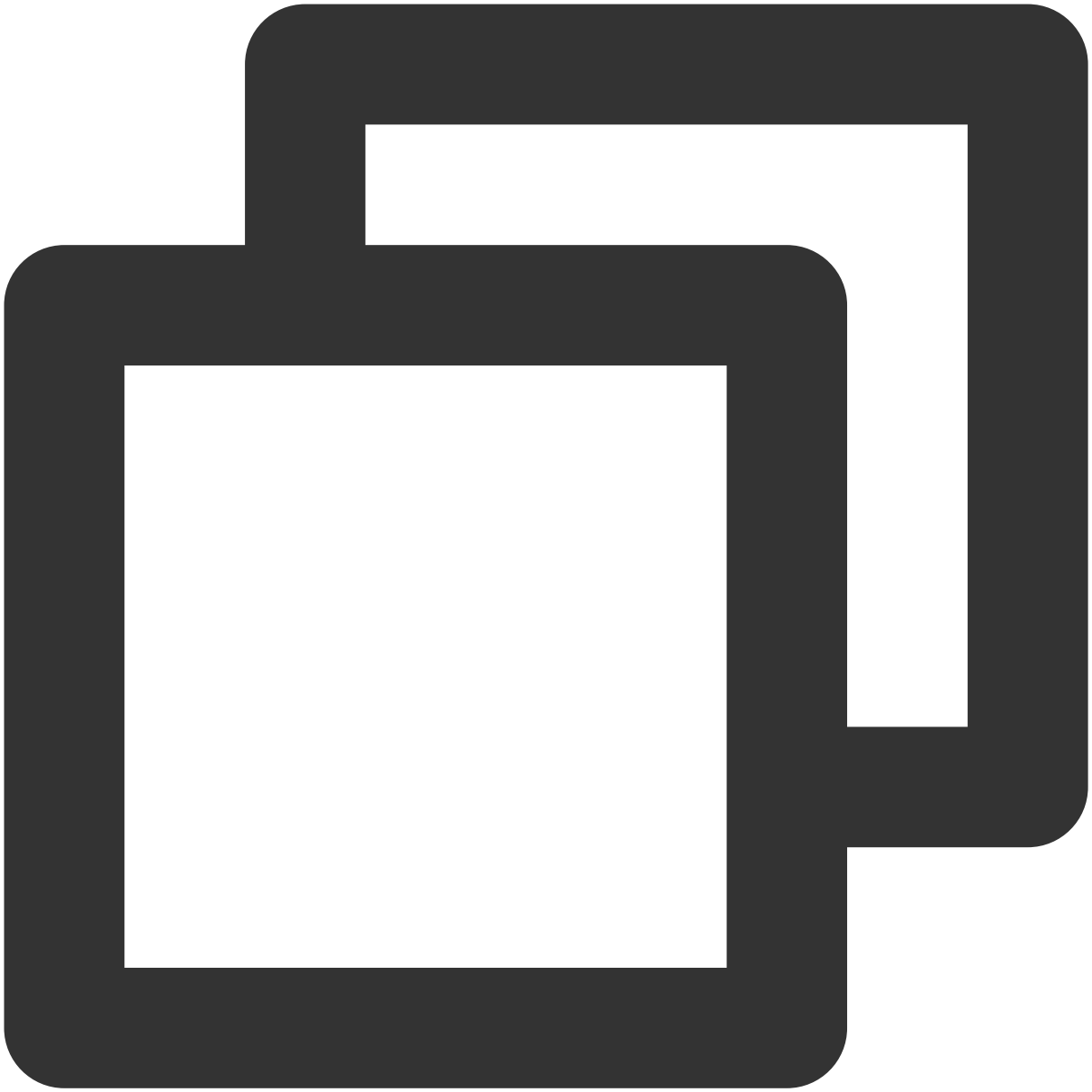
生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤五：拨打您的第一通电话

在上述登录方法调用返回成功后，调用TUICallKit 的 call 方法，指定被叫方的 userID 和通话类型，发起音视频通话，被叫方可接收到来电邀请。

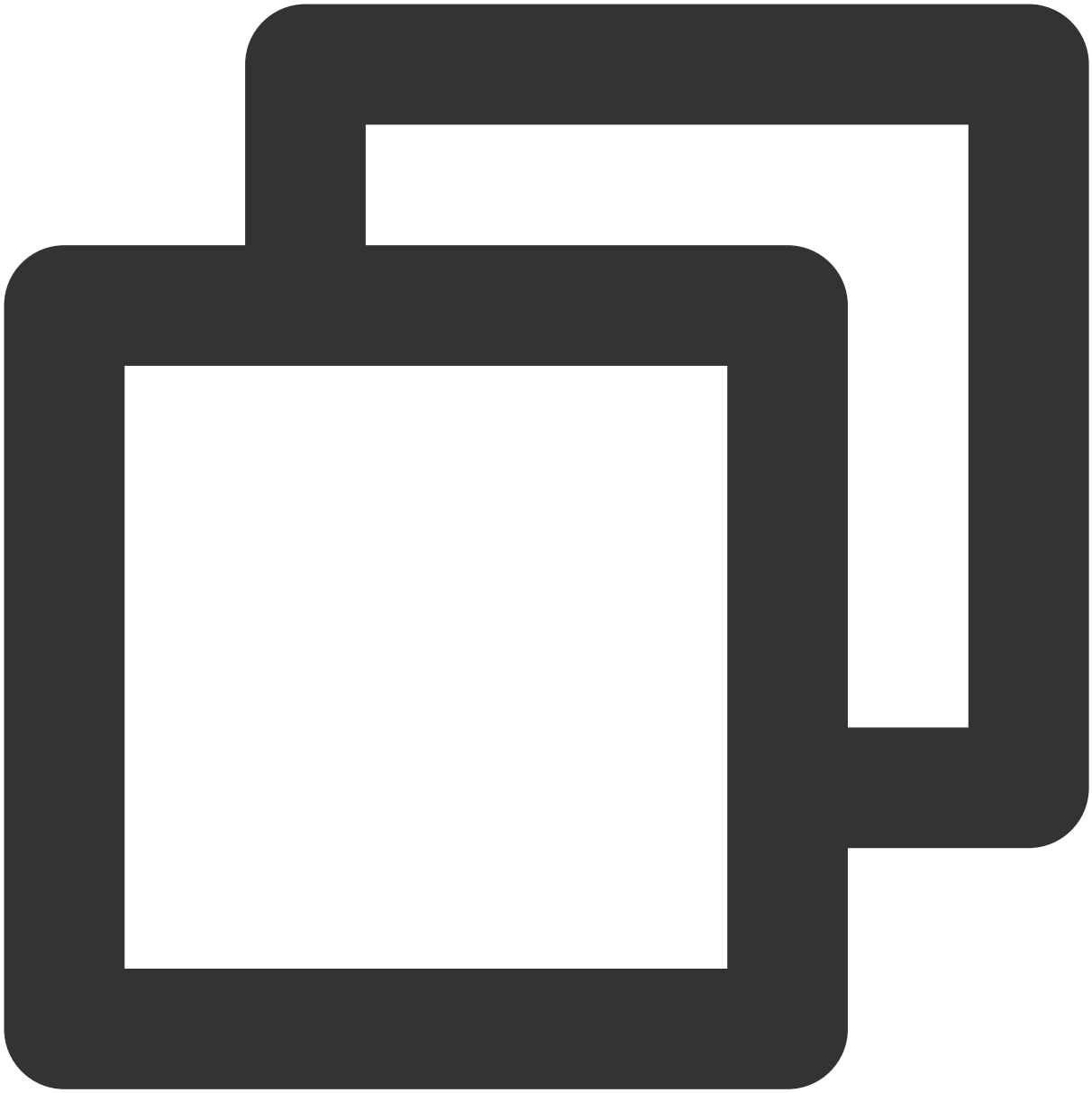
Kotlin

Java



```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

// 发起1对1语音通话(假设被叫方的 userID 为 mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Audio)
```

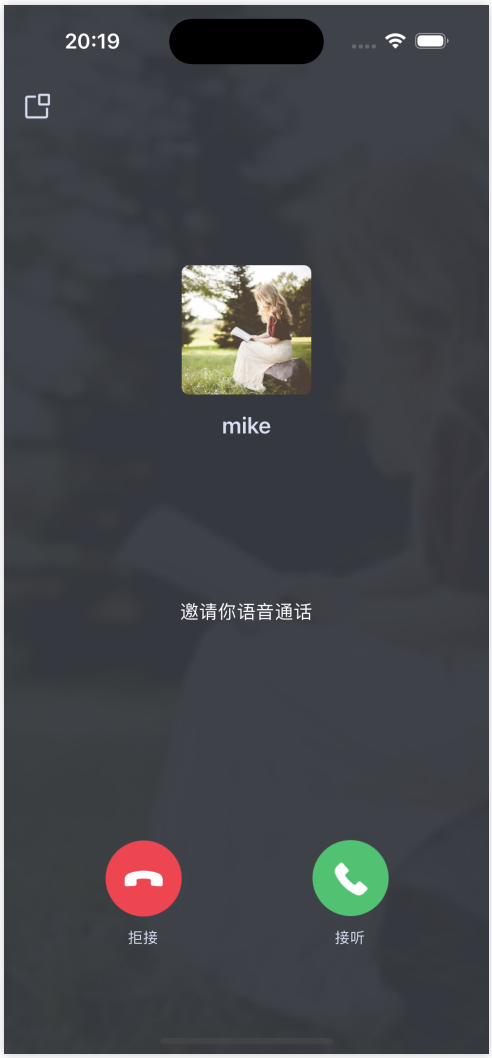


```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

// 发起1对1语音通话(假设被叫方的 userID 为 mike)
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Audio);
```

主叫方发起音频通话	被叫方收到音频通话请求





更多特性

设置昵称、头像
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。

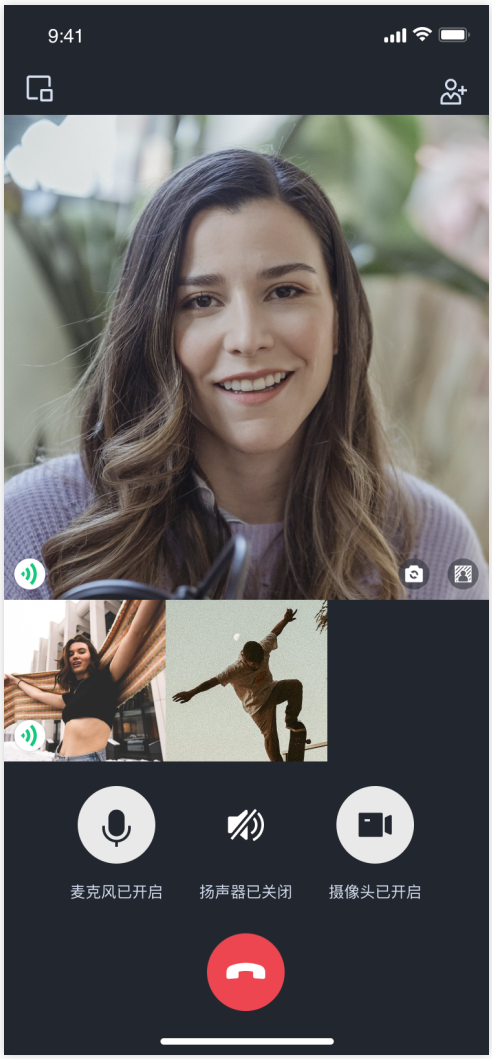
交流与反馈

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：2024-08-15 17:02:17

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。

1v1 视频通话	群组通话
	

环境准备

Xcode 13 及以上。

iOS 13.0 及以上。

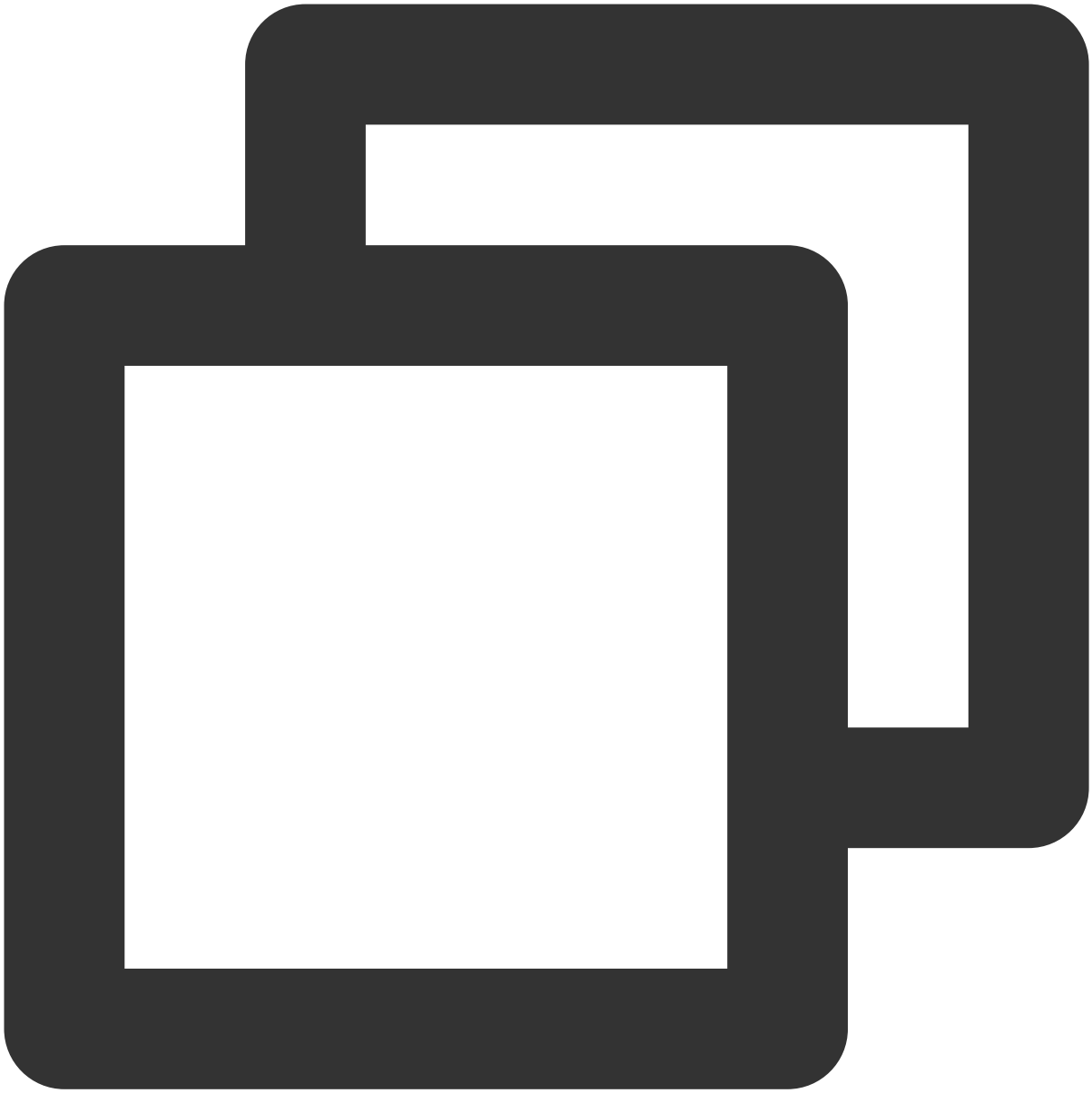
CocoaPods环境安装，点击查看。
如果您的接入和使用中遇到问题，请参见 常见问题。

步骤一：开通服务

请参见 开通服务，获取 `SDKAppID、SDKSecretKey` ，他们将在 步骤四：初始化 TUICallKit 组件 作为**必填参数**使用。

步骤二：导入组件

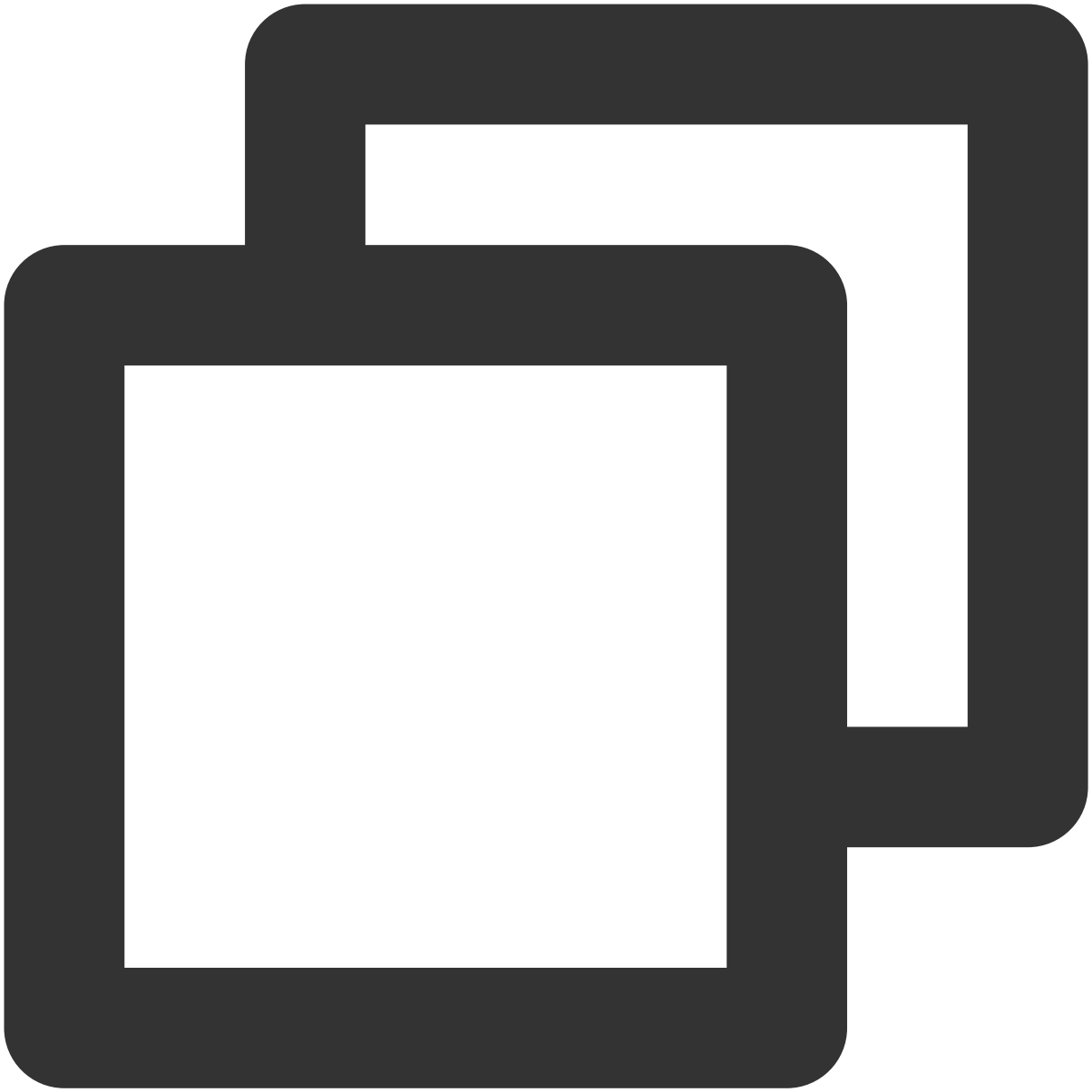
使用 CocoaPods 导入组件，如果您遇到问题，请先参见 环境准备。导入组件具体骤如下：
1. 请在您的 `Podfile` 文件中添加 `pod 'TUICallKit_Swift'` 依赖，如果您遇到任何问题，请参见 Example 工程。



```
target 'xxxx' do
  ...
  pod 'TUICallKit_Swift'
end
```

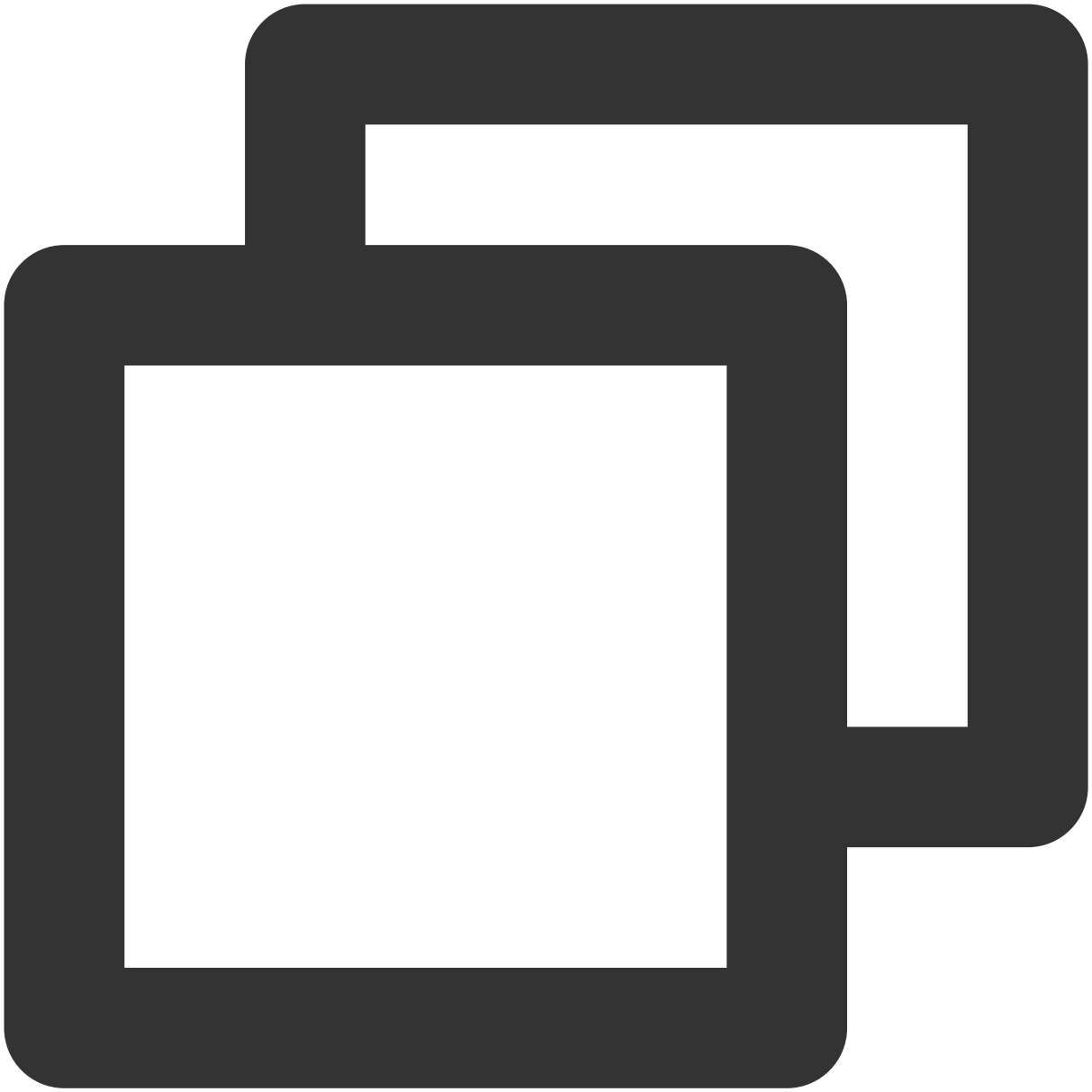
说明：

如果您的项目中缺少 `Podfile` 文件，您需要在终端中 `cd` 到 `xxxx.xcodeproj` 目录，然后，通过执行以下命令来创建 `Podfile` 文件：



```
pod init
```

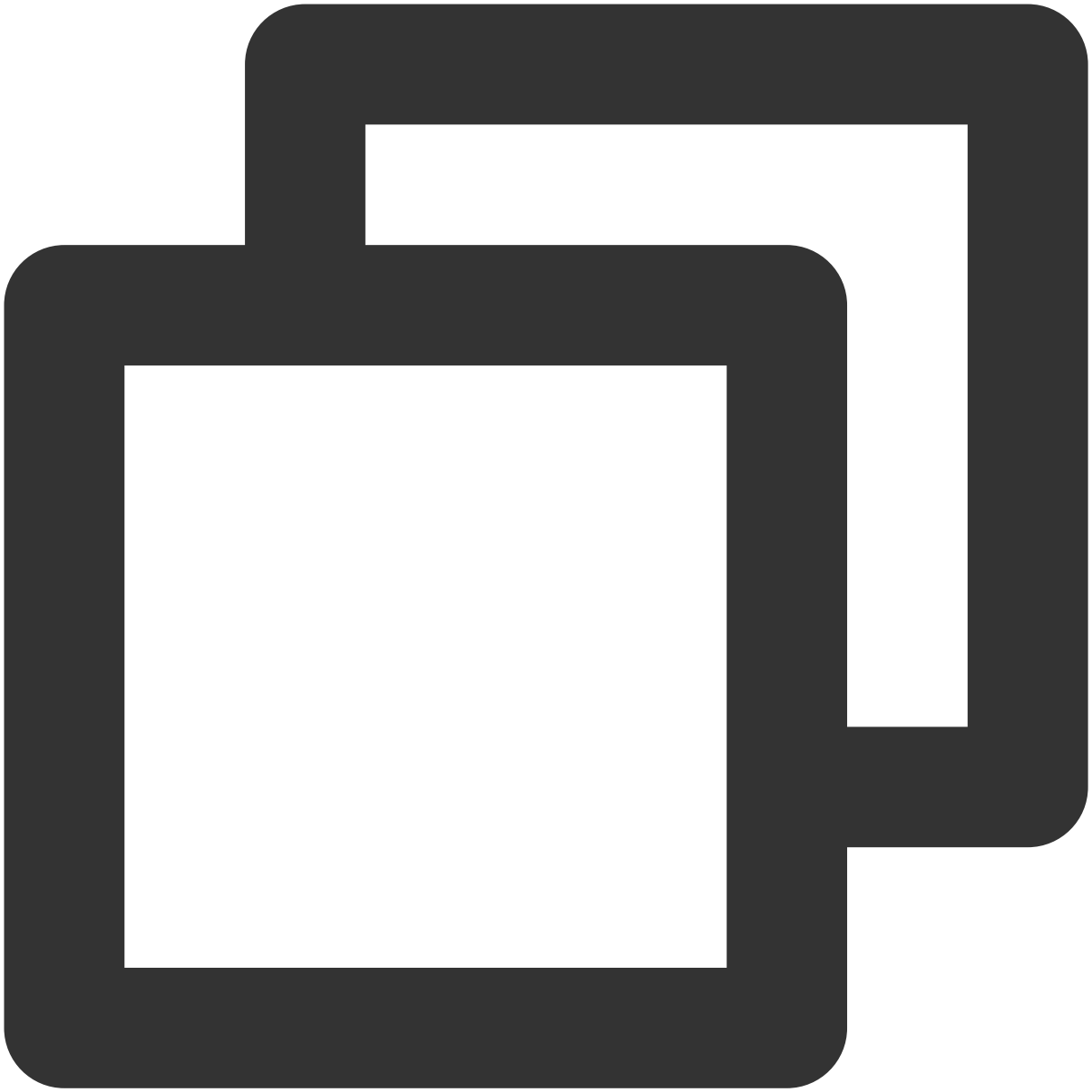
2. 在终端中，首先 `cd` 到 `Podfile` 目录下，然后执行以下命令，安装组件。



```
pod install
```

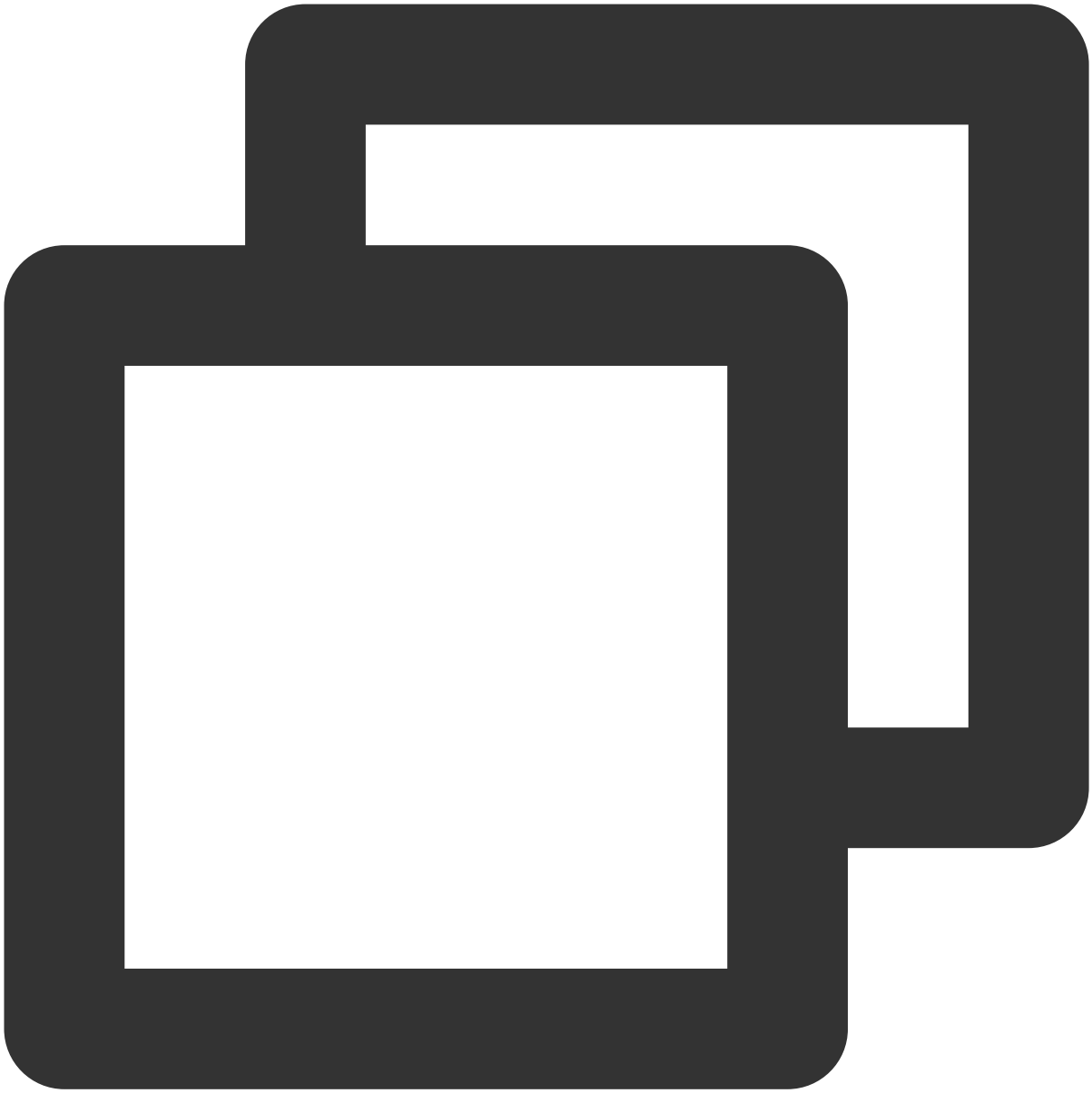
说明：

如果无法安装 TUICallKit 最新版本，可以先删除 **Podfile.lock** 和 **Pods**。然后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。



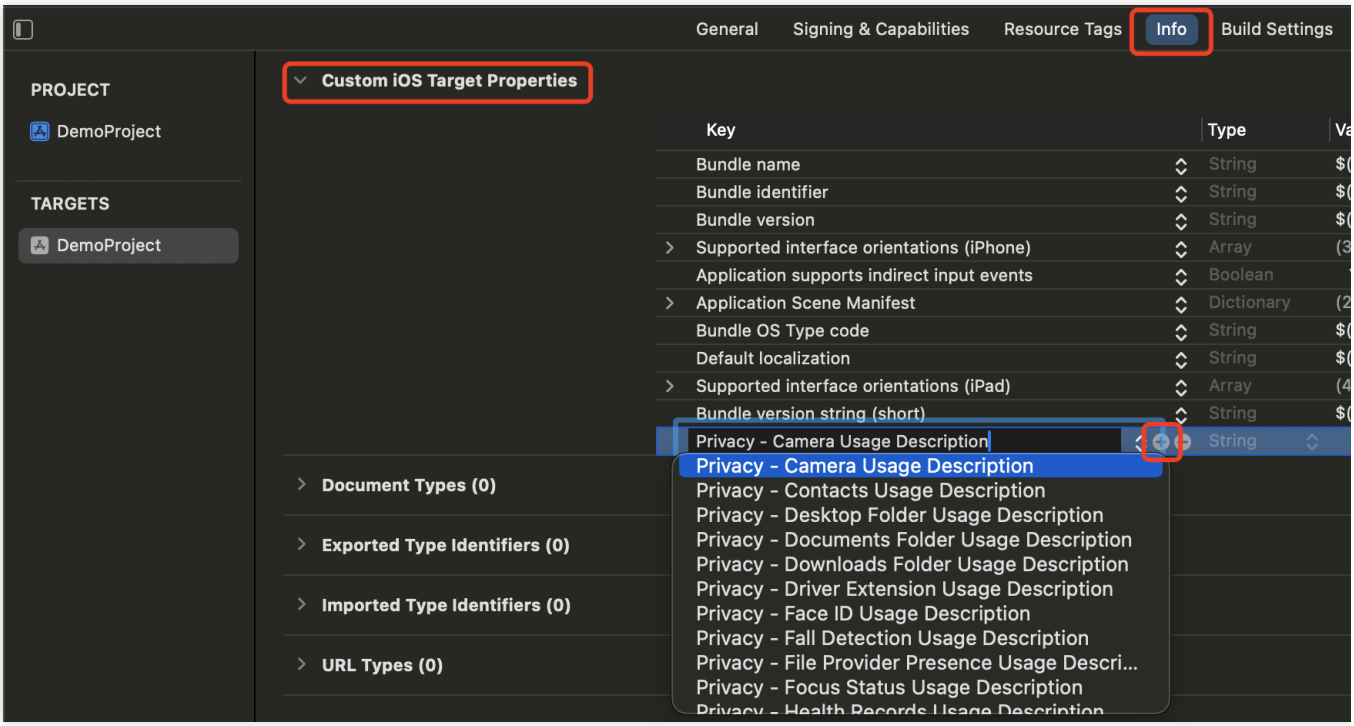
```
pod update
```

3. 建议您编译并运行一次。如果遇到问题，可以参见我们的 常见问题。如果问题仍未解决，您可以尝试运行我们的 Example 工程。在接入和使用过程中，如果遇到问题，欢迎向我们反馈。

步骤三：工程配置

使用音视频功能，需要授权摄像头和麦克风的使用权限，请根据实际项目需要，设置项目所需权限。

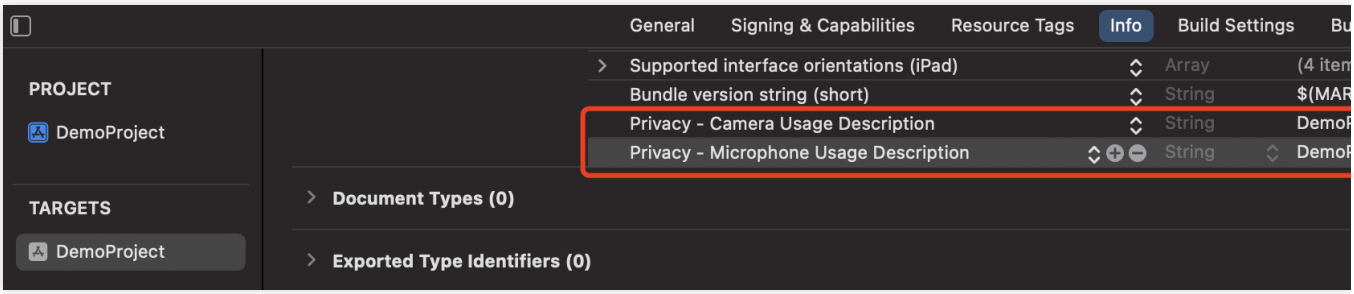
1. 在 **Xcode** 中，选择 **TARGETS** > **Info** > **Custom iOS Target Properties** 菜单。



2. 单击**+**，添加摄像头和麦克风权限。

```
Privacy - Camera Usage Description
```

```
Privacy - Microphone Usage Description
```

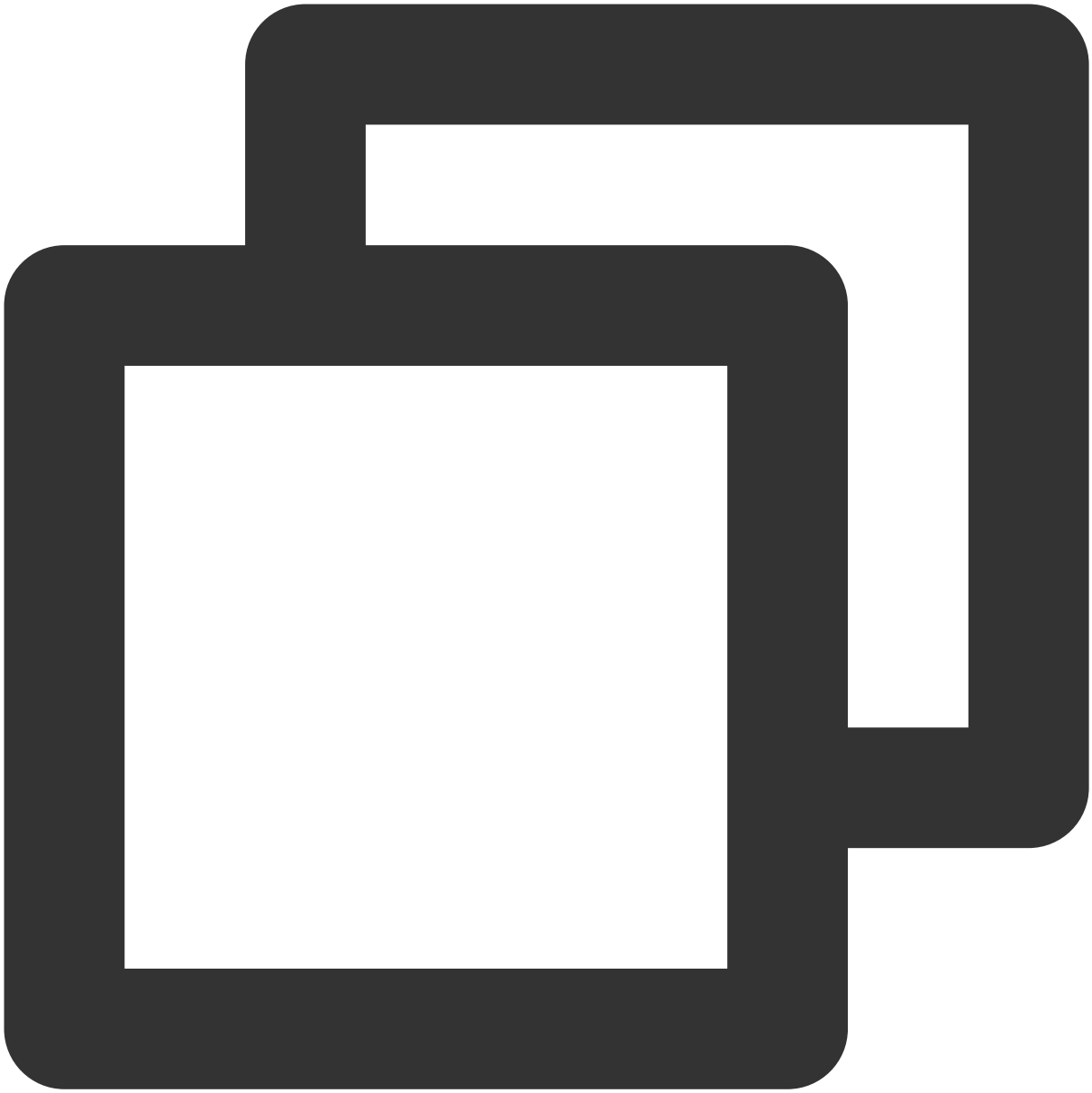


步骤四：登录 TUI 组件

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。这一步骤至关重要，只有在成功登录之后，您才能正常使用 TUICallKit 提供的各项功能。

Swift

Objective-C



```
import TUICore
import TUICallKit_Swift

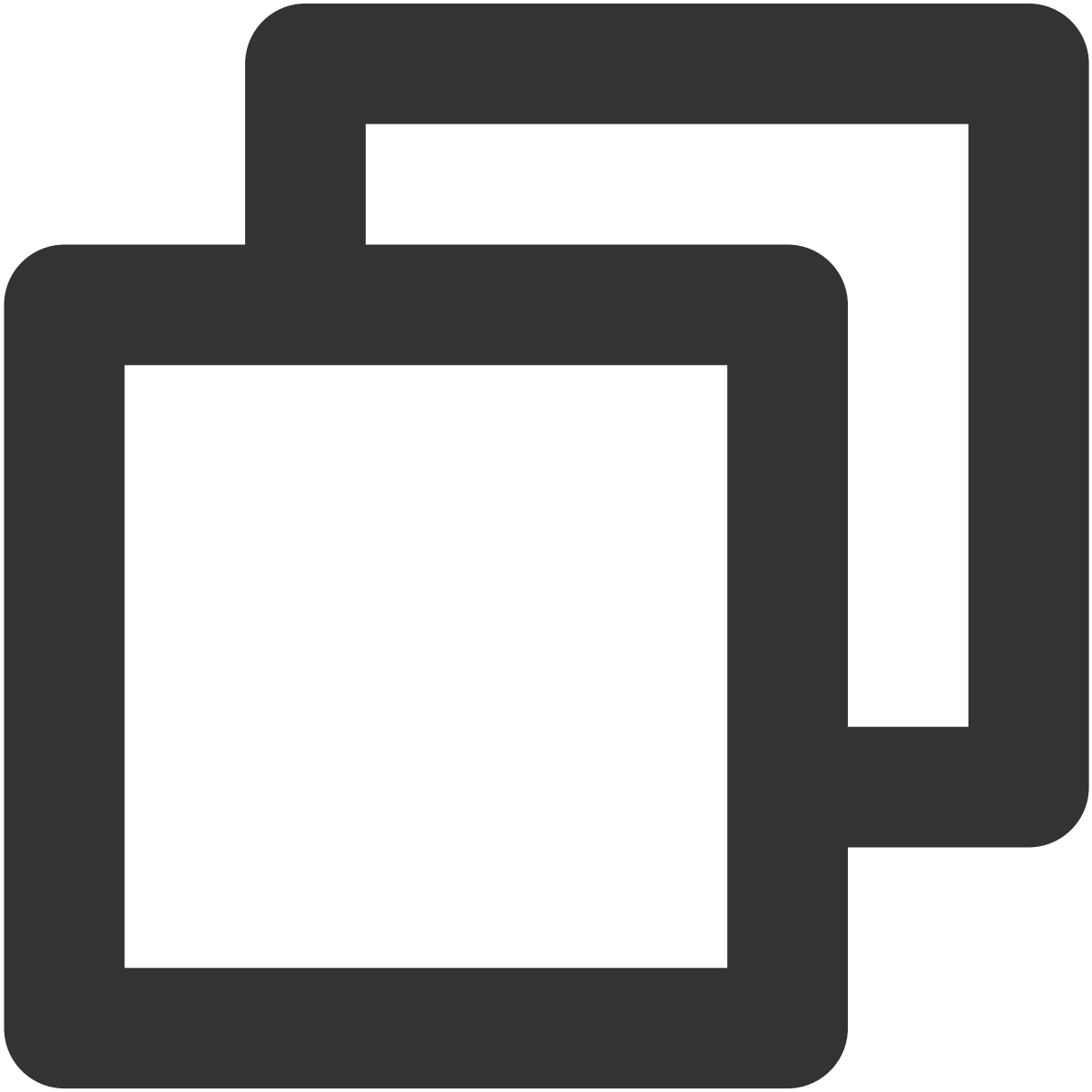
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch

    let userID = "denny"        // 请替换为您的 UserId
    let sdkAppID: Int32 = 0     // 请替换为第一步在控制台得到的 SDKAppID
    let secretKey = "****"      // 请替换为第一步在控制台得到的 SecretKey

    let userSig = GenerateTestUserSig.genTestUserSig(userID: userID, sdkAppID: sdkA
```

```
    TUILogin.login(sdkAppID, userID: userID, userSig: userSig) {
      print("login success")
    } fail: { code, message in
      print("login failed, code: \\(code), error: \\(message ?? "nil")")
    }

    return true
}
```



```
#import <TUICore/TUILogin.h>
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
```

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD

    NSString *userID = @"denny";      // 请替换为您的 UserId
    int sdkAppID = 0;                 // 请替换为第一步在控制台得到的 SDKAppID
    NSString *secretKey = @"****";    // 请替换为第一步在控制台得到的 SecretKey

    NSString *userSig = [GenerateTestUserSig genTestUserSigWithUserID:userID sdkApp

    [TUILogin login:sdkAppID
             userID:userID
            userSig:userSig
               succ:^{
        NSLog(@"login success");
    } fail:^(int code, NSString * _Nullable msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }];

    return YES;
}
```

参数	类型	说明
userID	String	客户根据自己的业务自定义用户 ID，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
sdkAppID	Int32	在 Tencent RTC 控制台 创建的音视频应用的唯一标识 SDKAppID。
secretKey	String	在 Tencent RTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

注意：

开发环境：如果您处于本地开发调试阶段，可以使用本地 GenerateTestUserSig.genTestSig 函数生成 userSig。但请注意，该方法中的 SDKSecretKey 容易被反编译逆向破解。一旦密钥泄露，攻击者可能盗用您的腾讯云流量。

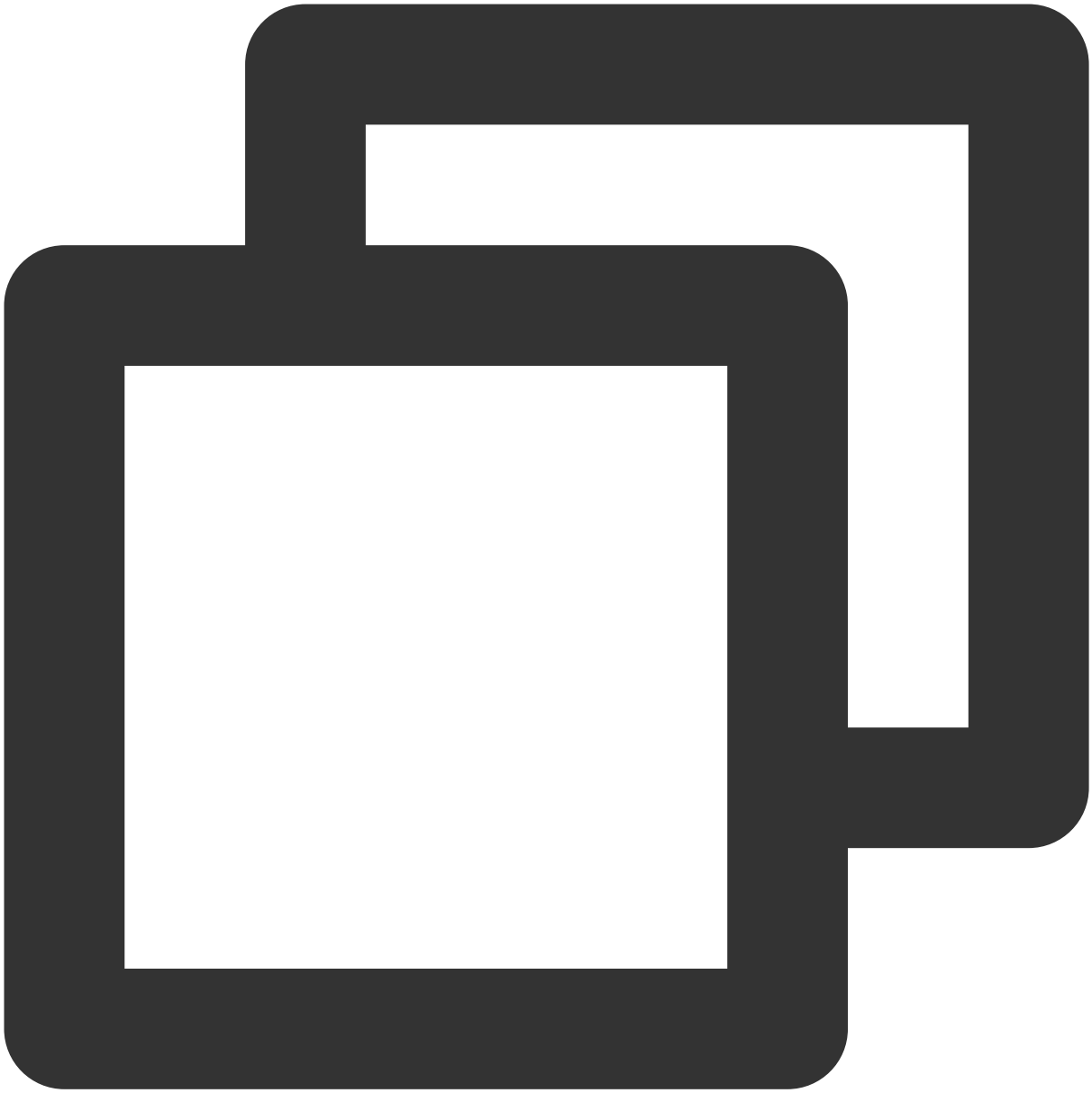
生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

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。

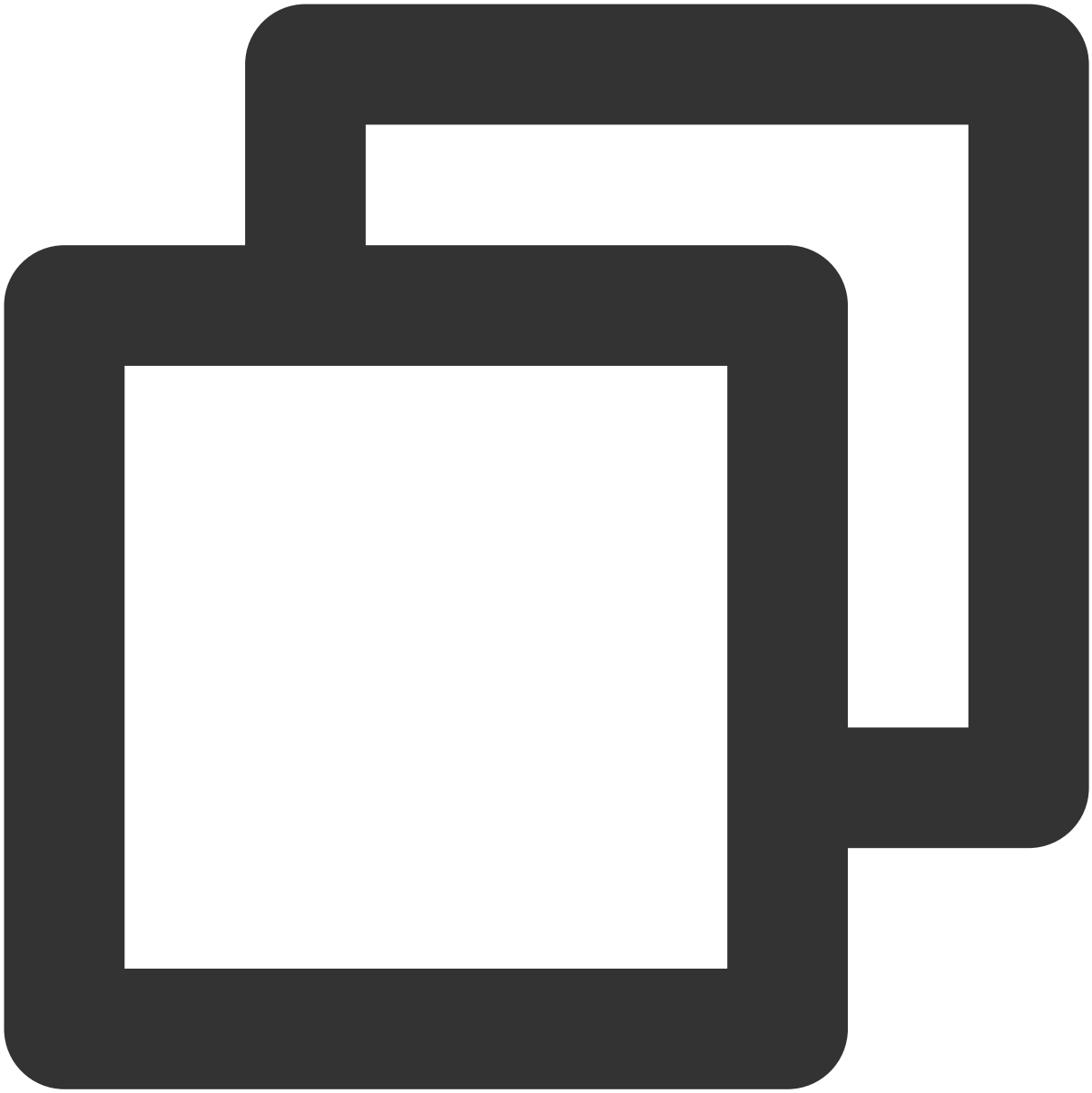
Swift

Objective-C



```
import TUICallKit_Swift
import TUICallEngine

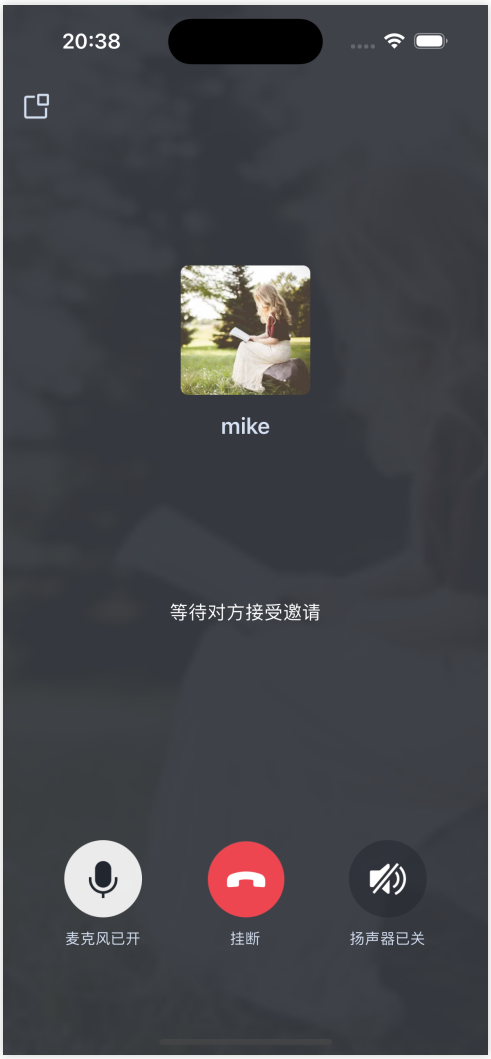
// 发起1对1语音通话(假设 userId 为 mike)
TUICallKit.createInstance().call(userId: "mike", callMediaType: .audio)
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

// 发起1对1语音通话(假设 userId 为 mike)
[[TUICallKit createInstance] callWithUserId:@"mike" callMediaType:TUICallMediaTypeA
```

主叫方	被叫方





更多特性

设置昵称、头像
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

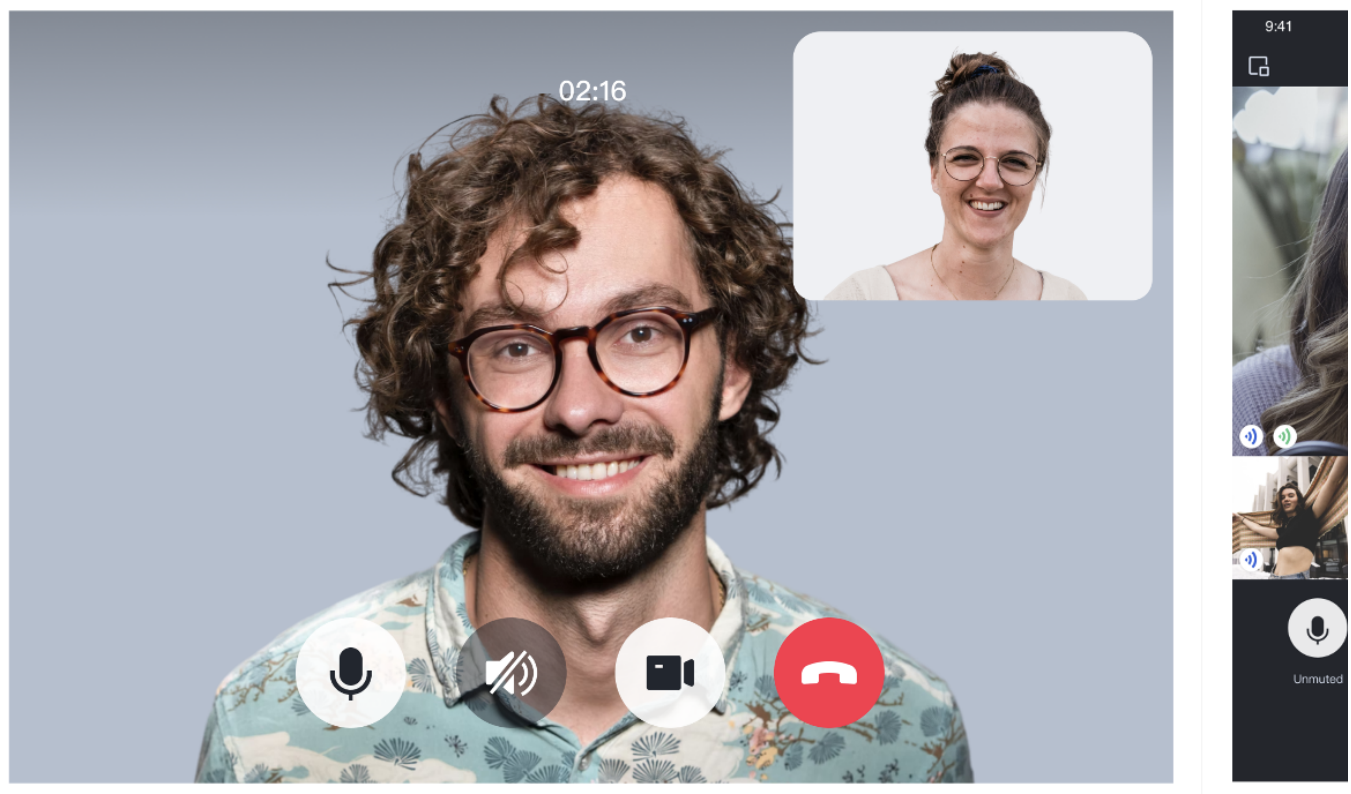
如果有任何需要或者反馈，您可以联系：info_rtc@tencent.com。

版权所有：腾讯云计算（北京）有限责任公司

Web&H5 (React)

最近更新时间：2024-07-22 17:21:41

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。



环境准备

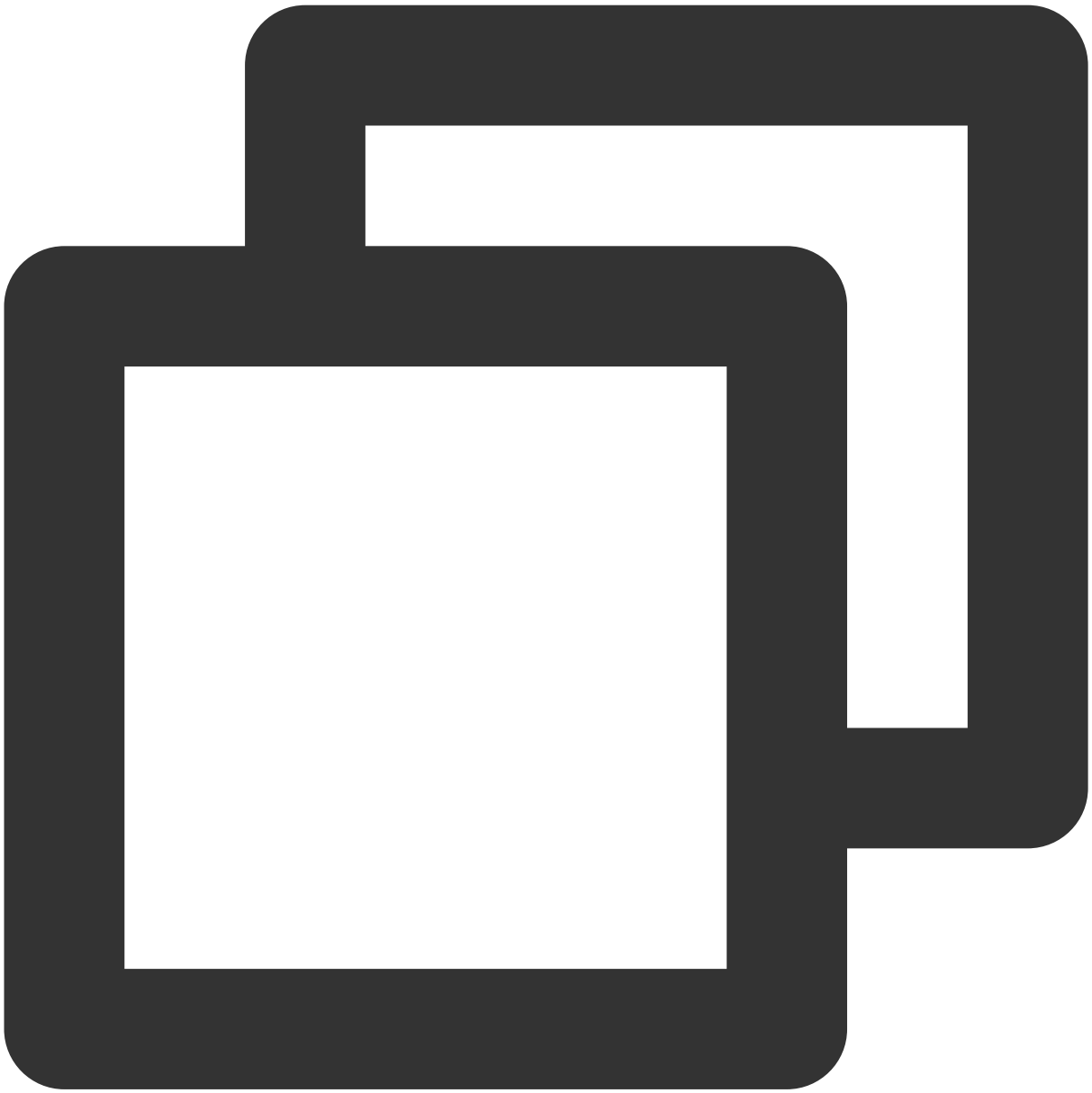
React version 18+.
Node.js version 16+.
Modern browser, supporting WebRTC APIs.

步骤一：开通服务

请参见 开通服务，获取 `SDKAppID、SDKSecretKey` ，他们将在 初始化 TUICallKit 组件 作为**必填参数**使用。

步骤二：下载 TUICallKit 组件

1. 下载 @tencentcloud/call-uikit-react 组件。

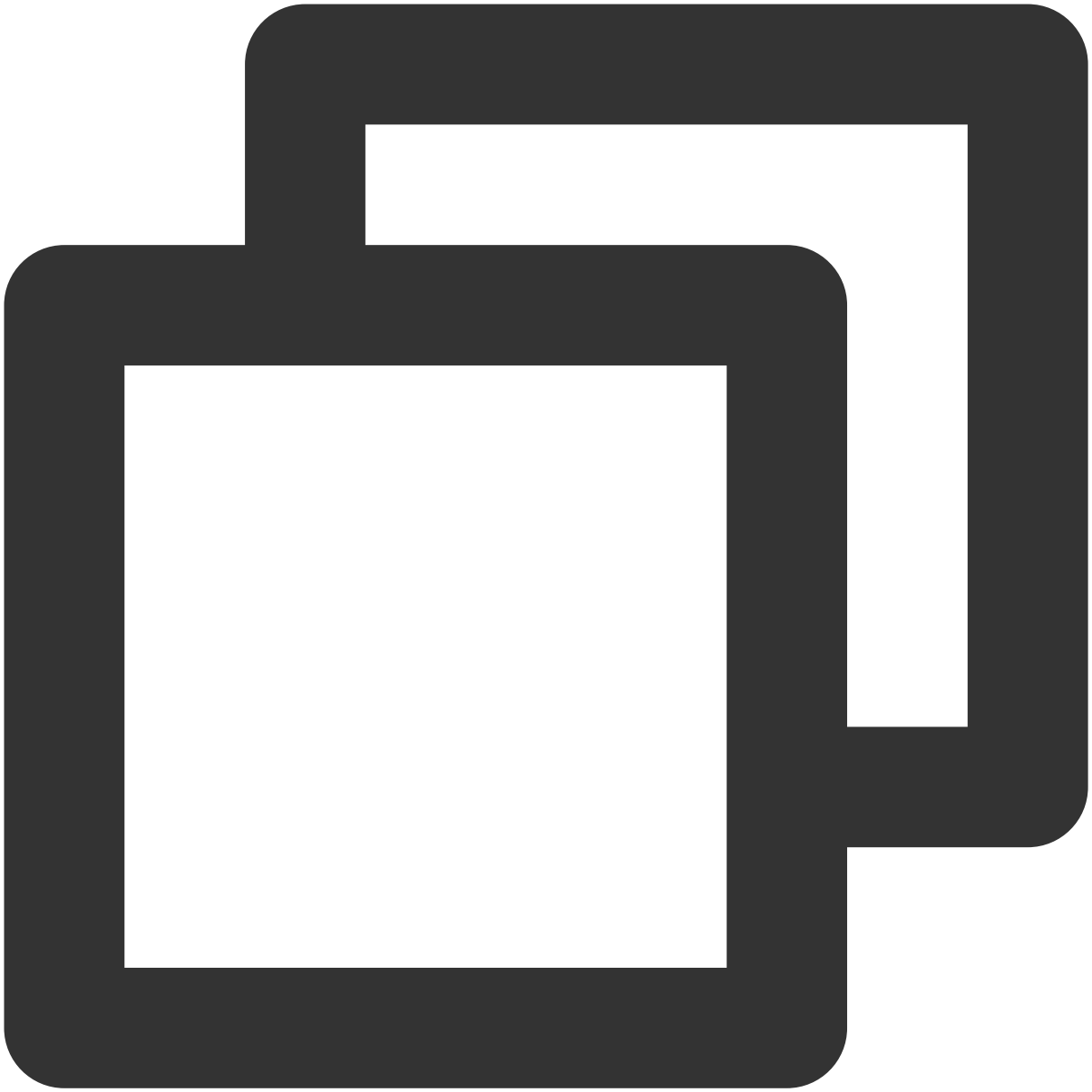


```
npm install @tencentcloud/call-uikit-react
```

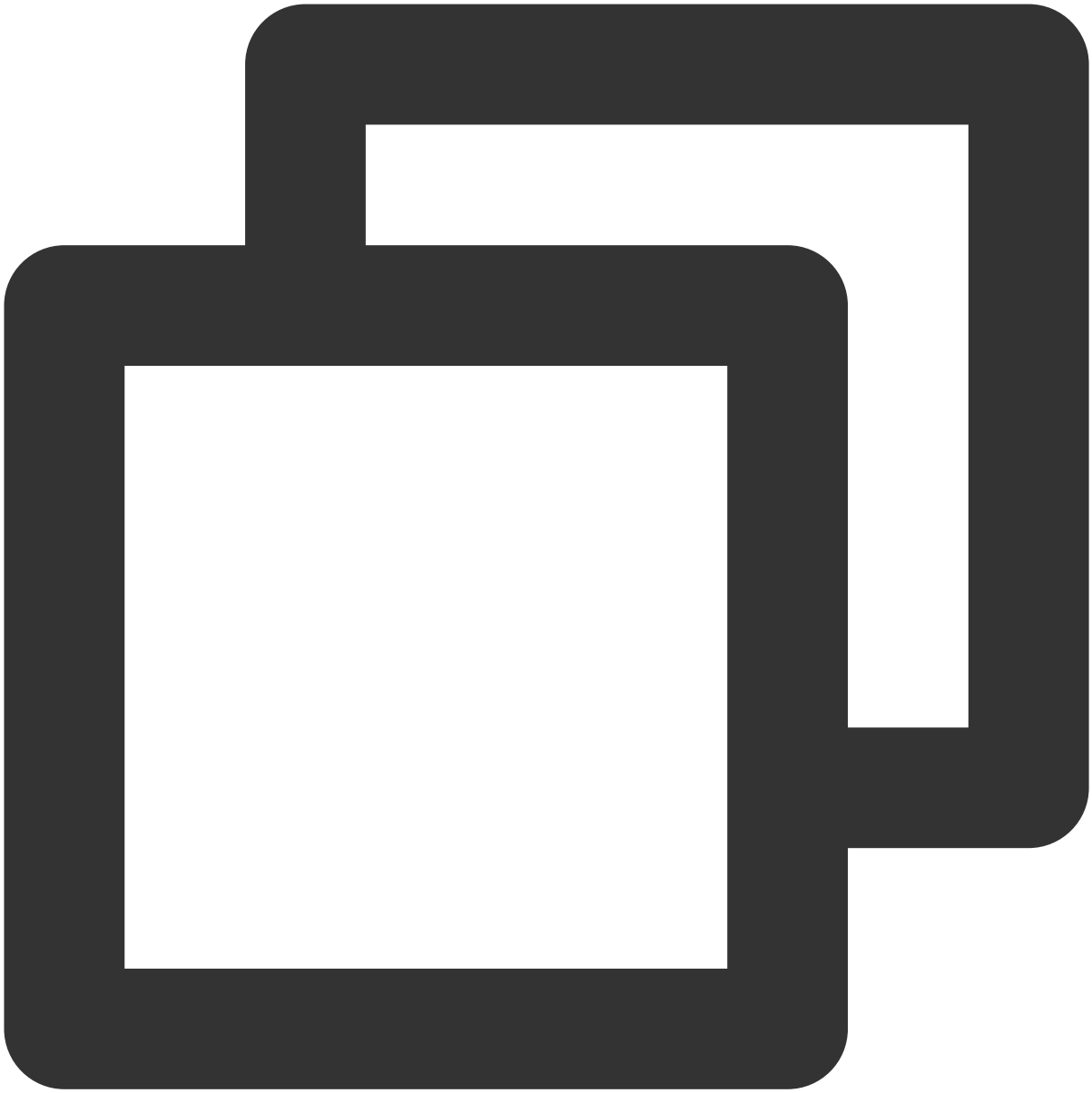
2. 将 **debug** 目录复制到您的项目目录 `src/debug` ，本地生成 userSig 时需要使用。

MacOS

Windows



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

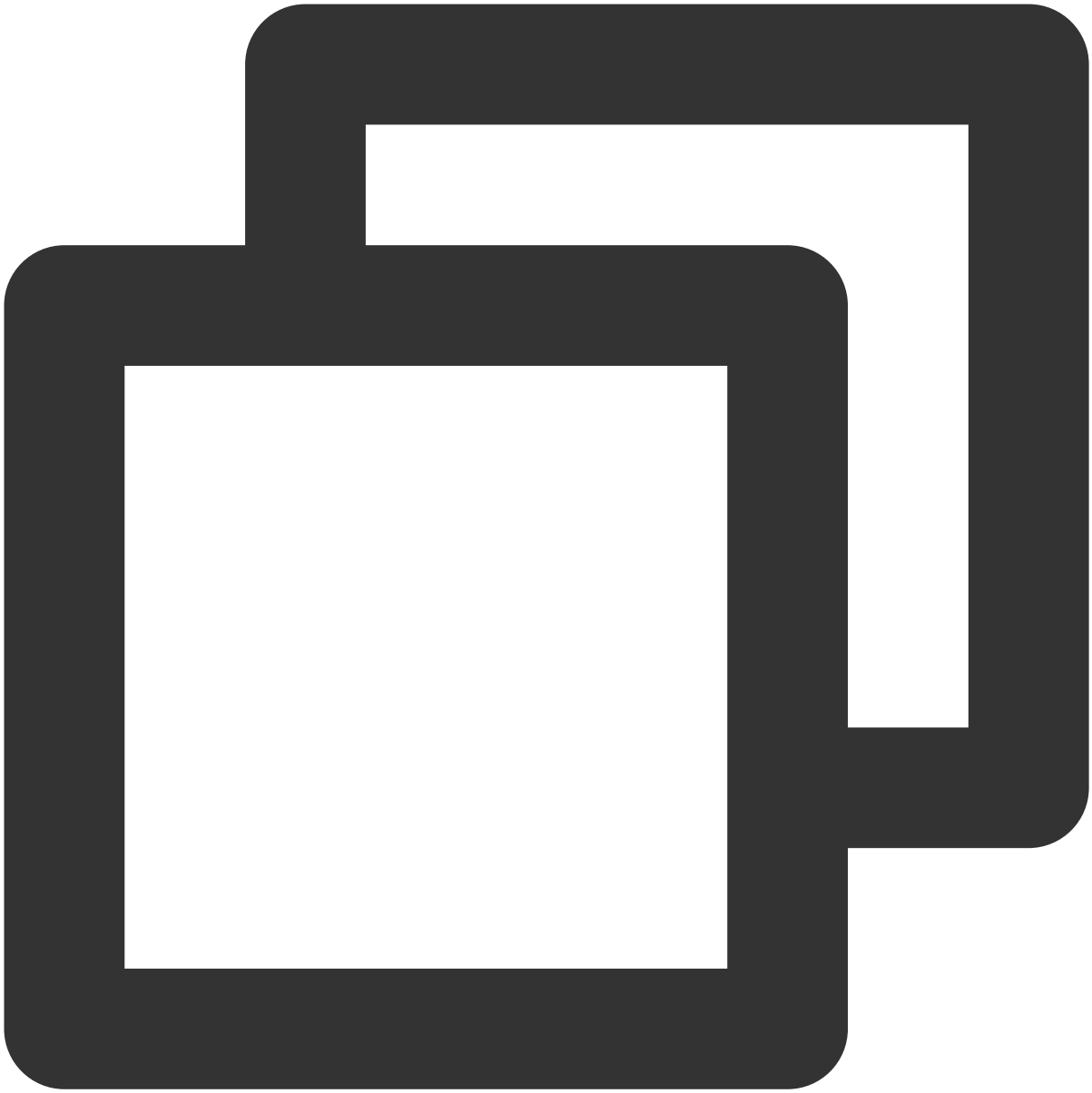


```
xcopy node_modules\\@tencentcloud\\call-uikit-react\\debug .\\src\\debug /i /e
```

步骤三：初始化 TUICallKit 组件

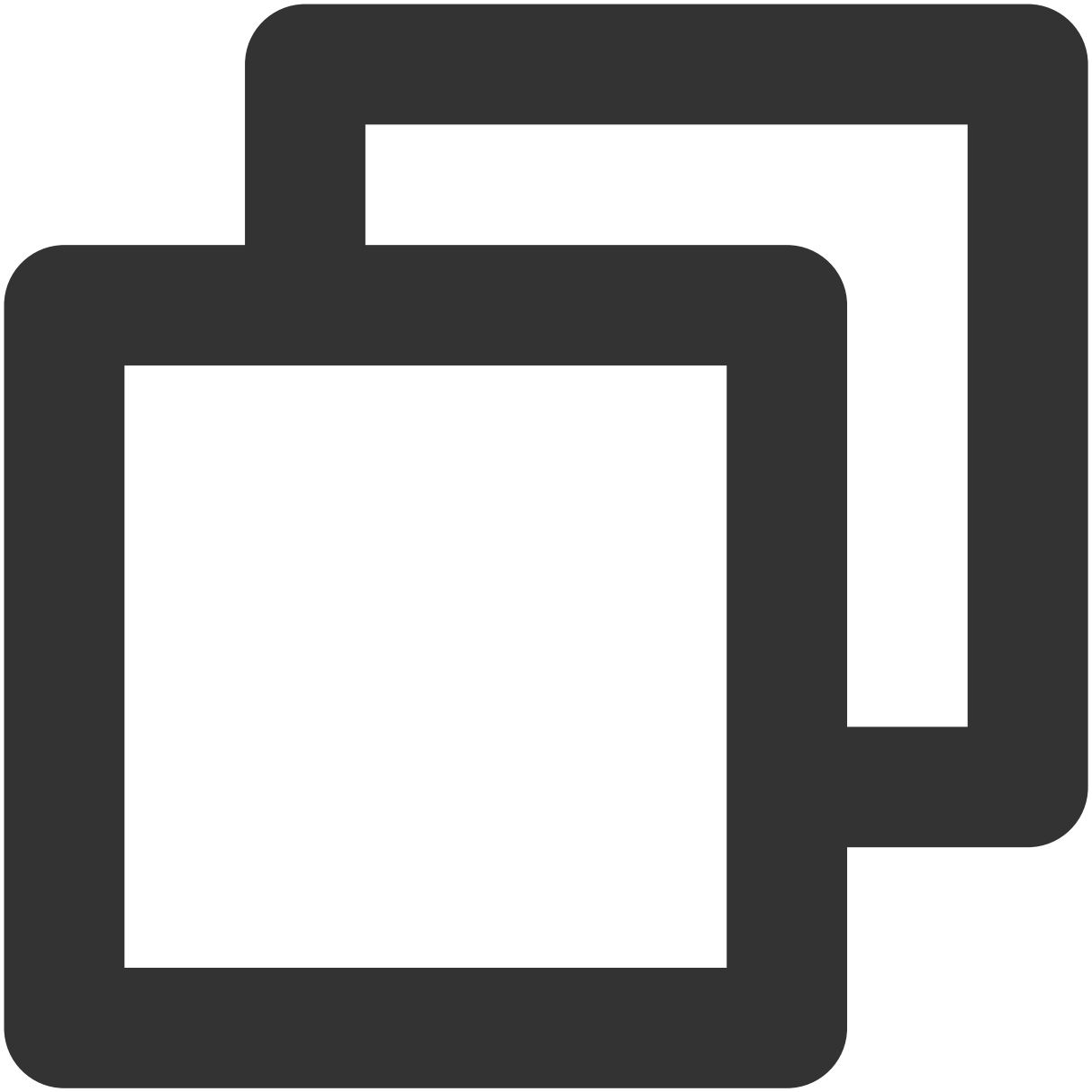
您可以选择在 `/src/App.tsx` 文件引入示例代码。

1. 引入 call-uikit 相关 API 对象。



```
import { useState } from 'react';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to
```

2. 引入<TUICallKit />，该组件包含通话时的完整 UI 交互。

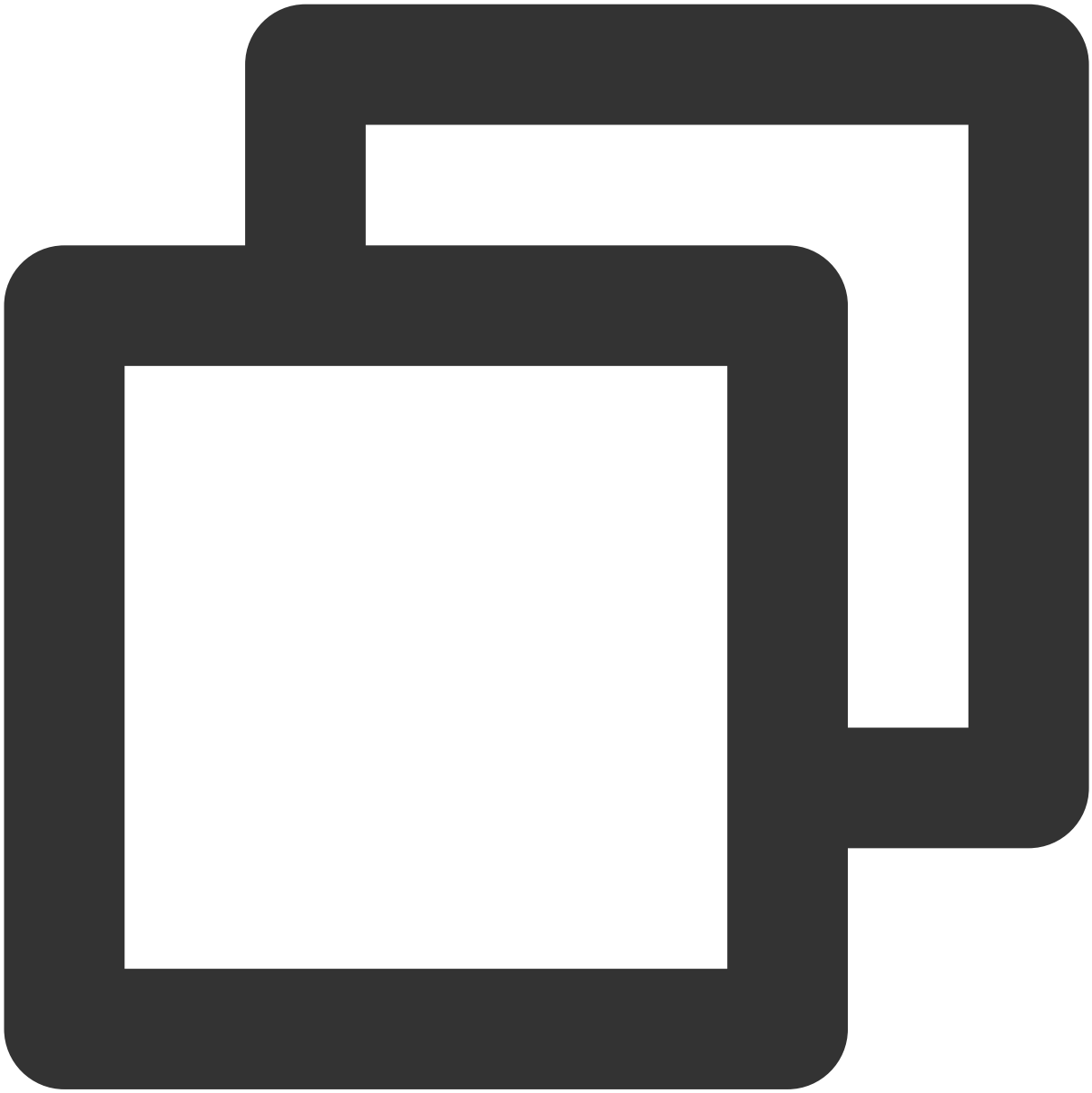


```
return (
  <>
    <span> caller's ID: </span>
    <input type="text" placeholder='input caller userID' value={callerUserID} onCha
    <button onClick={init}> step1. init </button> <br />
    <span> callee's ID: </span>
    <input type="text" placeholder='input callee userID' value={calleeUserID} onCha
    <button onClick={call}> step2. call </button>

    {/* 【1】Import the TUICallKit component: Call interface UI */}
    <TUICallKit />
```

```
    </>
);
```

3. 调用 TUICallKitServer.init API 登录组件，需要在代码中**填写** `SDKAppID、SDKSecretKey` **两个参数。**



```
const SDKAppID = 0;         // TODO: Replace with your SDKAppID (Notice: SDKAppID is
const SDKSecretKey = '';    // TODO: Replace with your SDKSecretKey

const [callerUserID, setCallerUserID] = useState('');
const [calleeUserID, setCalleeUserID] = useState('');
// 【2】Initialize the TUICallKit component
```

```
const init = async () => {
  const { userSig } = GenerateTestUserSig.genTestUserSig({
    userID: callerUserID,
    SDKAppID,
    SecretKey: SDKSecretKey,
  });
  await TUICallKitServer.init({
    userID: callerUserID,
    userSig,
    SDKAppID,
  });
  alert('TUICallKit init succeed');
}
//【3】Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.call({
    userID: calleeUserID.value,
    type: TUICallType.VIDEO_CALL,
  });
};
```

参数	类型	说明
userID	String	**用户的唯一标识符**，由您定义，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
SDKAppID	Number	在 Tencent RTC 控制台 创建的音视频应用的唯一标识。
SDKSecretKey	String	在 Tencent RTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

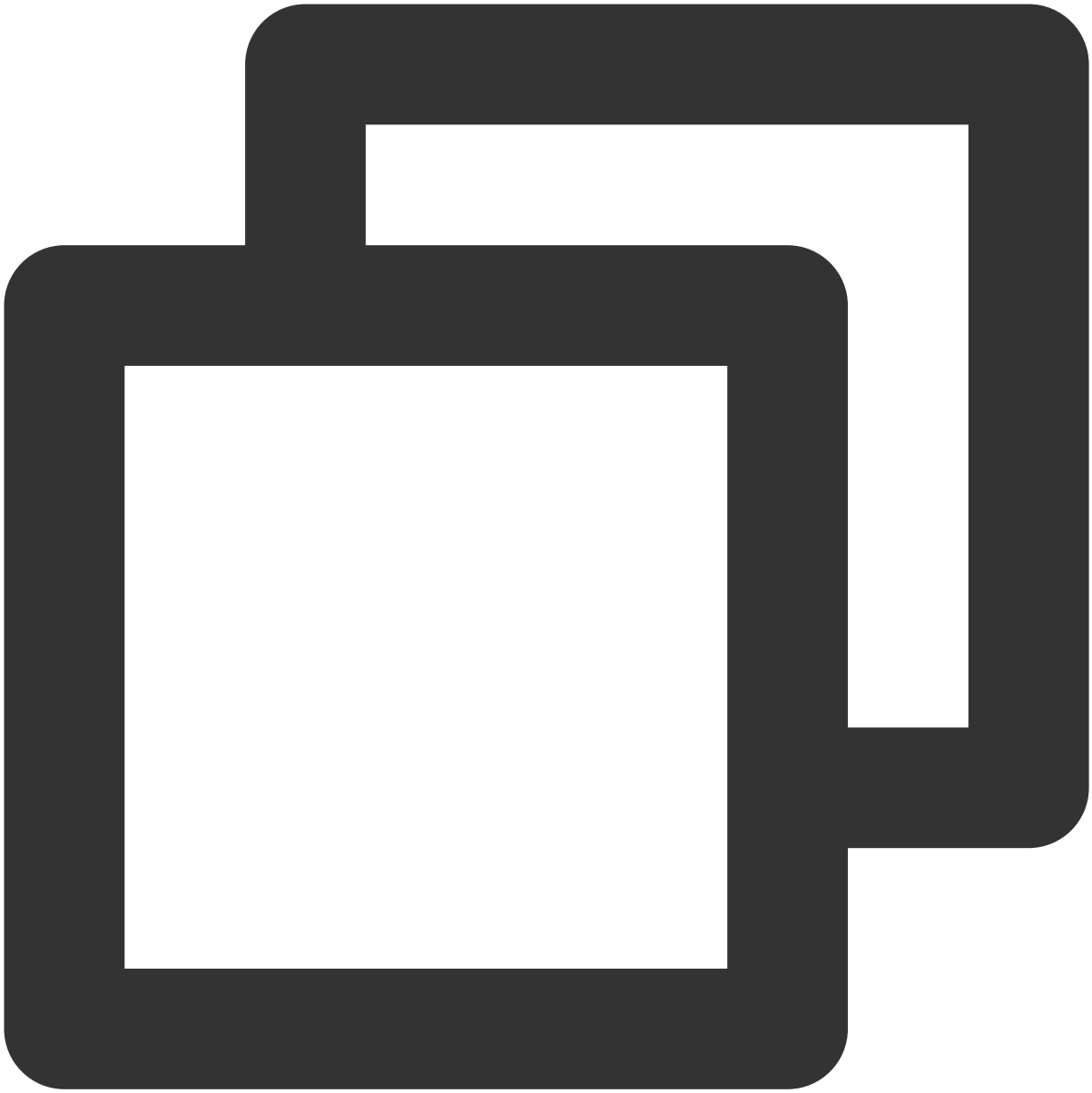
userSig 说明：

开发环境：如果您正在本地跑通 Demo、开发调试，可以采用 `debug` 文件中的 `genTestUserSig`（参考步骤 3.2）函数生成 userSig。该方法中 SDKSecretKey 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。

生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤四：拨打您的第一通电话

1. 调用 TUICallKitServer.call API 拨打通话。



```
//【3】Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.call({
    userID: calleeUserID,
    type: TUICallType.VIDEO_CALL,
  });
};
```

2. 运行项目。

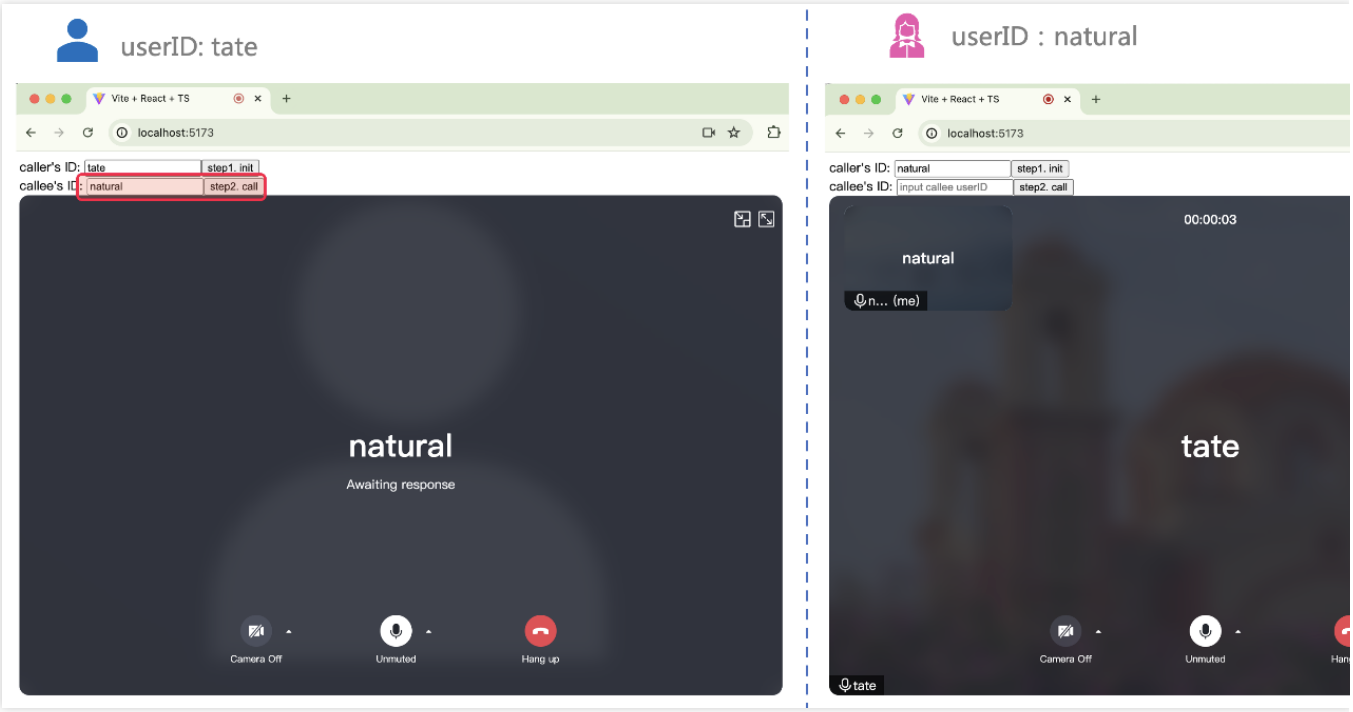
警告：

本地环境请在 localhost 协议下访问，公网体验请在 HTTPS 协议下访问，具体参见 网络访问协议说明。

3. 打开两个浏览器页面，输入不同的 userID(由您定义) 单击 `step1. init` 登录（主叫方和被叫方）。



4. 两个 userID 都登录成功后，单击 `step2. call` 拨打通话，如果您有通话问题，参见 常见问题。



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

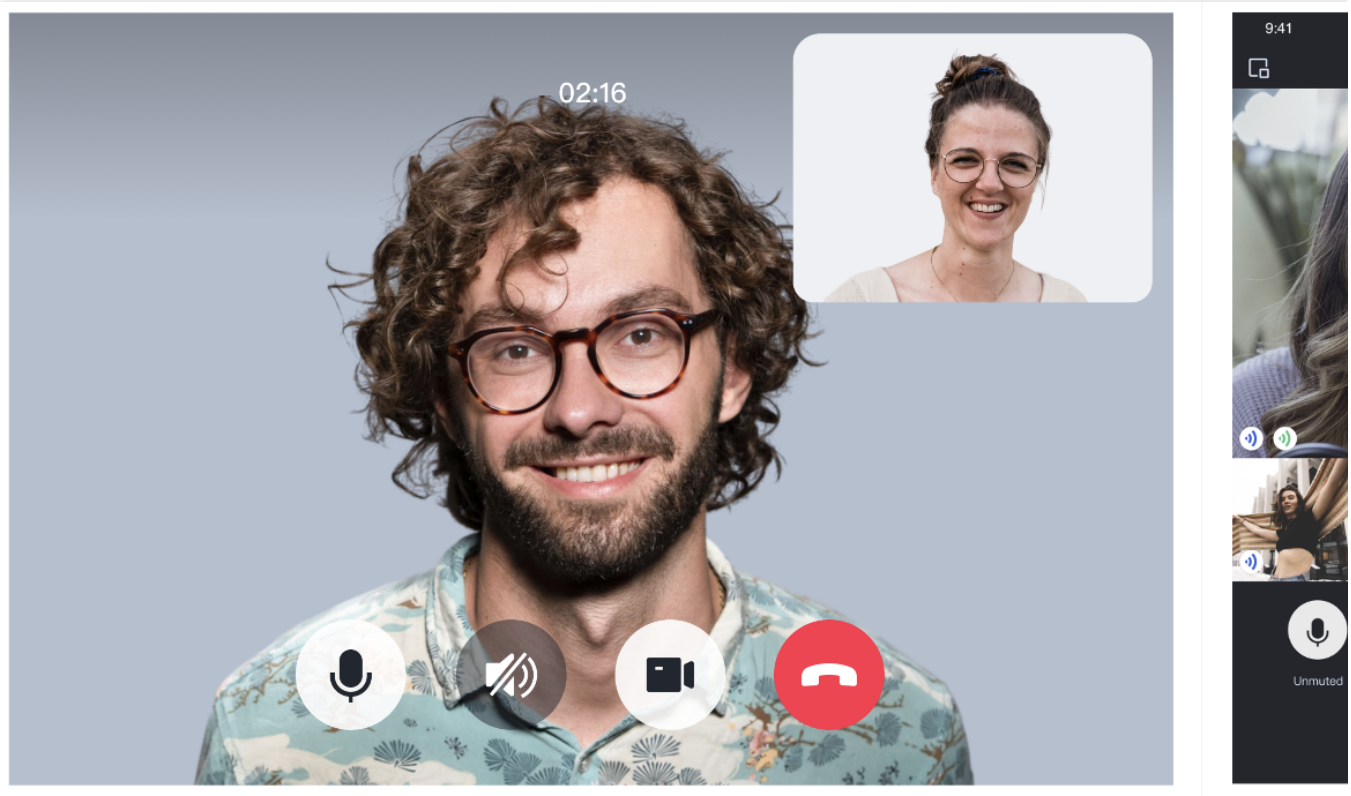
常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。
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：2024-07-22 17:21:41

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。



环境准备

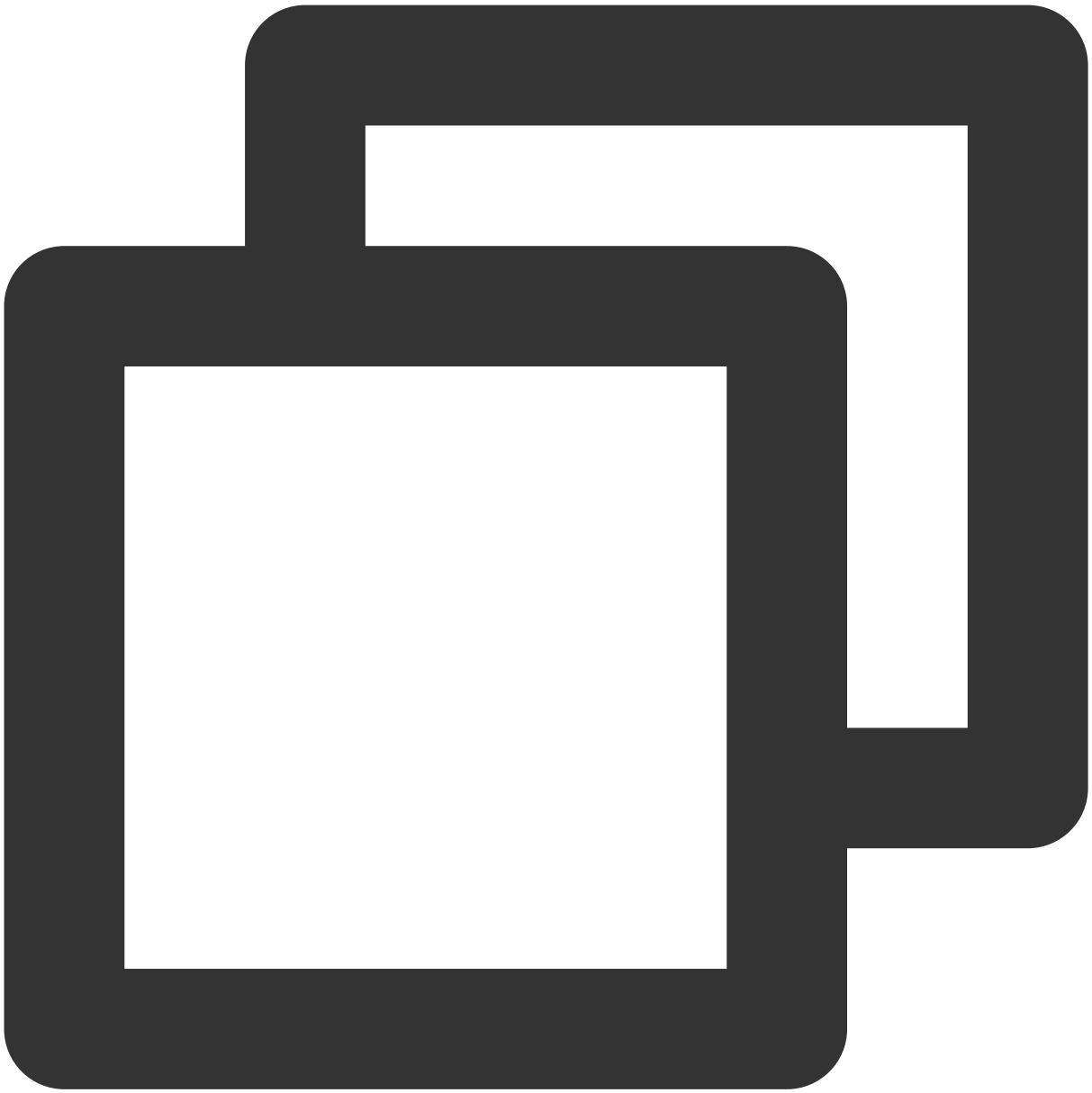
Node.js version 16+.
Modern browser, supporting WebRTC APIs.

步骤一：开通服务

请参见 开通服务，获取 `SDKAppID、SDKSecretKey` ，他们将在 初始化 TUICallKit 组件 作为**必填参数**使用。

步骤二：下载 TUICallKit 组件

1. 下载 @tencentcloud/call-uikit-vue 组件。

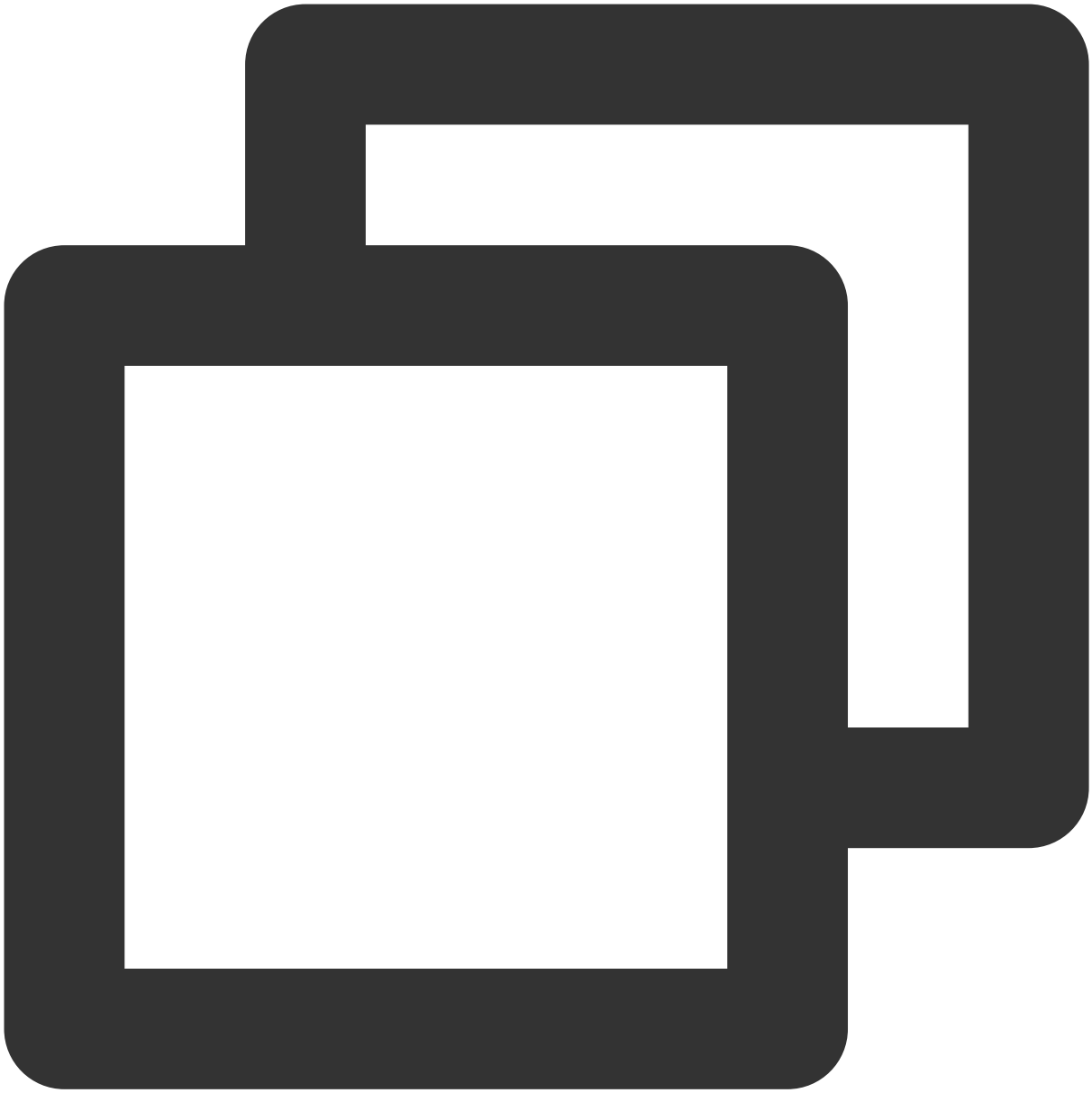


```
npm install @tencentcloud/call-uikit-vue
```

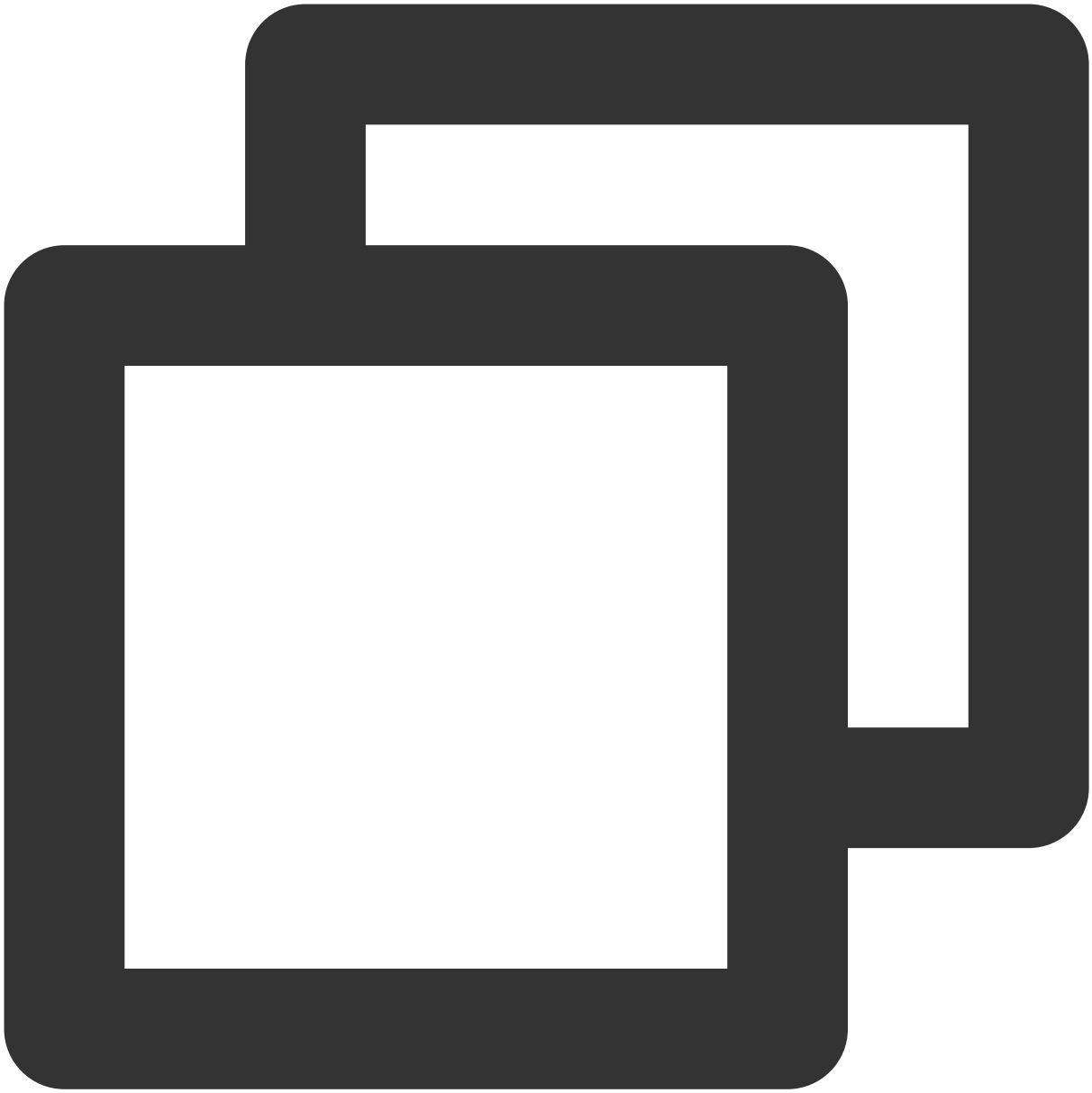
2. 将 **debug** 目录复制到您的项目目录 `src/debug`，本地生成 userSig 时需要使用。

MacOS

Windows



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

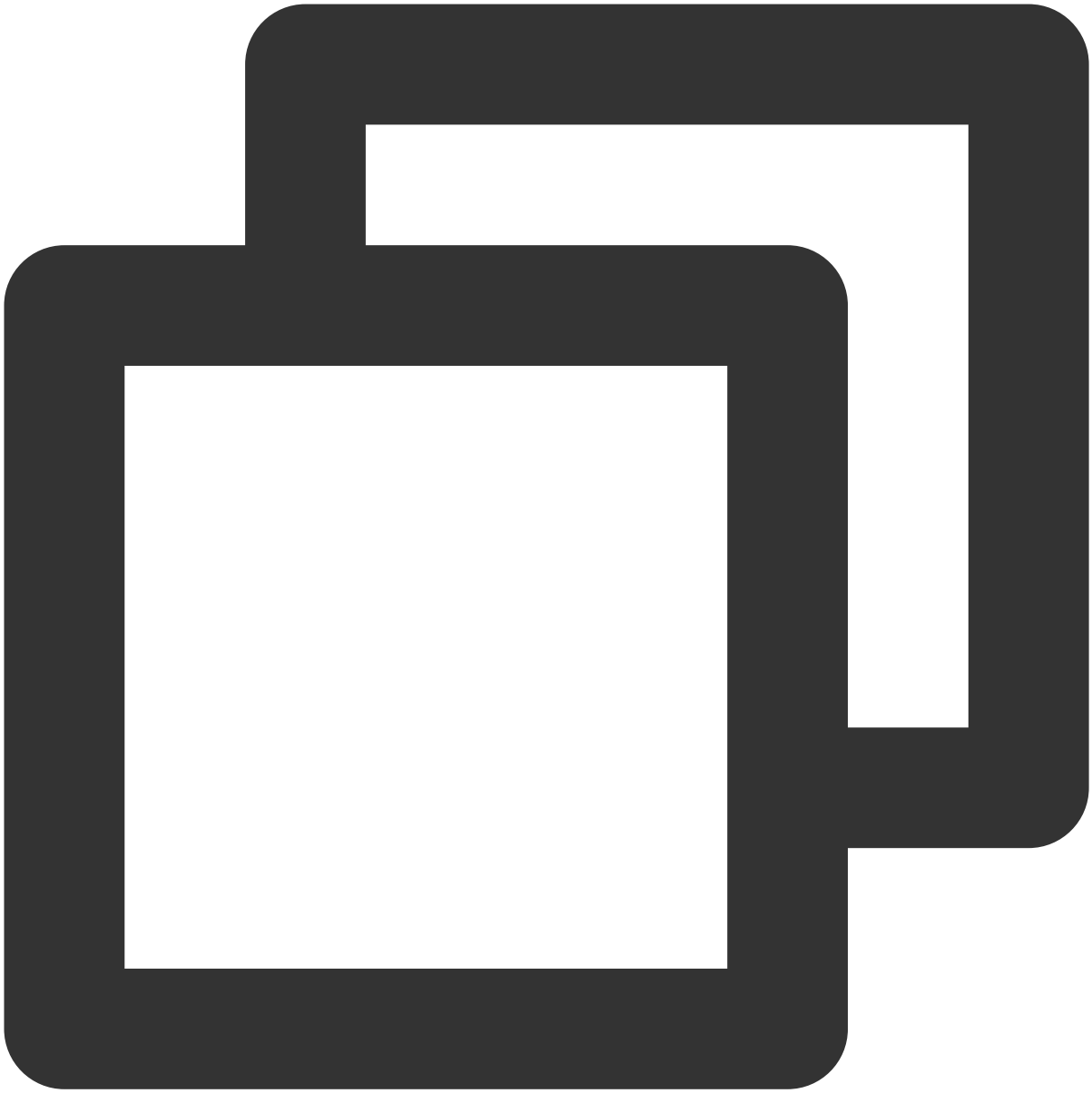


```
xcopy node_modules\\@tencentcloud\\call-uikit-vue\\debug .\\src\\debug /i /e
```

步骤三：初始化 TUICallKit 组件

您可以选择在 `callkit-demo/src/App.vue` 文件引入示例代码，示例代码采用的是 **`Composition API`** 写法。

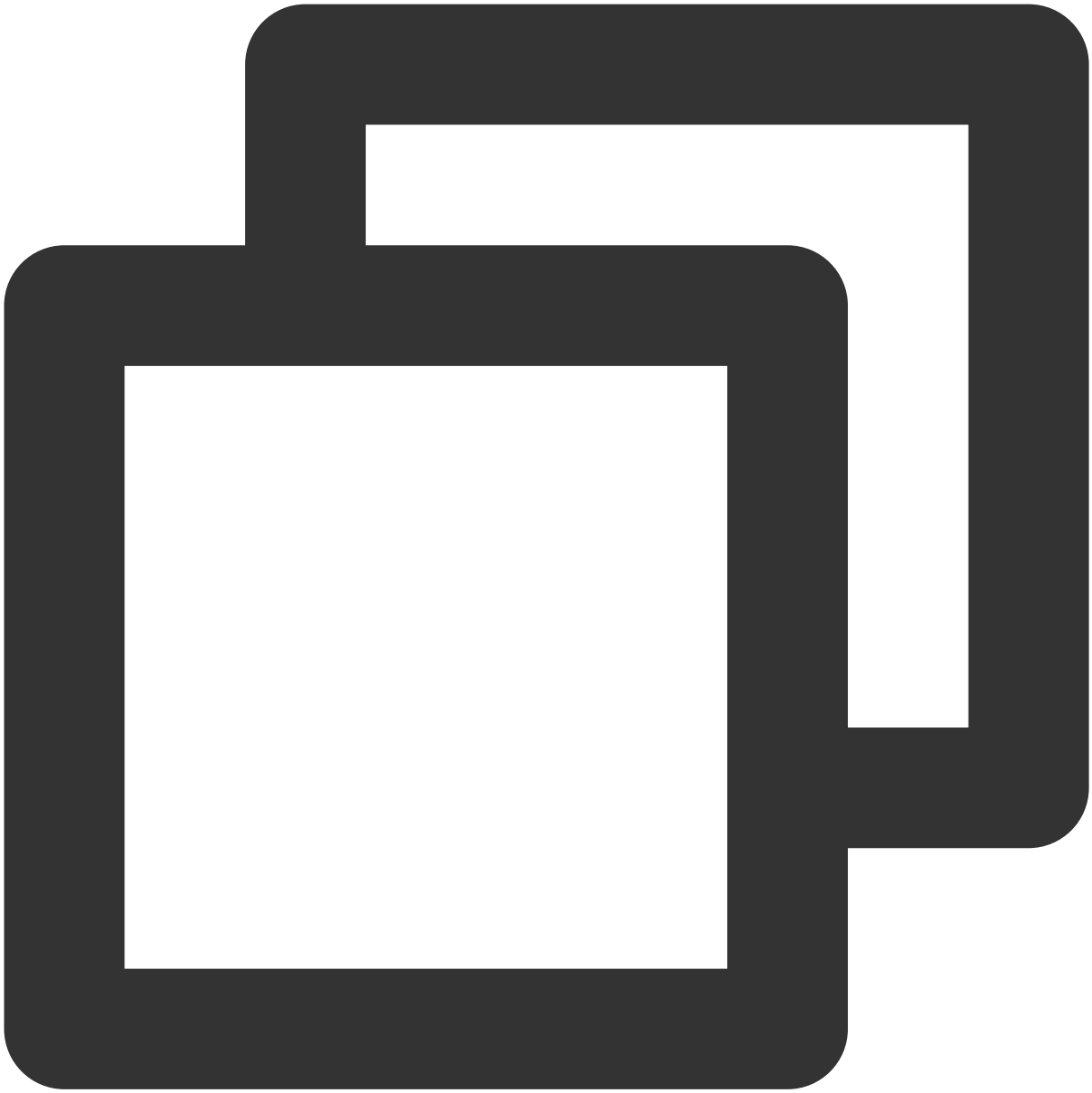
1. 引入 <TUICallKit />，该组件包含通话时的完整 UI 交互。



```
<template>
  <span> caller's ID: </span>
  <input type="text" v-model="callerUserID">
  <button @click="init"> step1. init </button> <br>
  <span> callee's ID: </span>
  <input type="text" v-model="calleeUserID">
  <button @click="call"> step2. call </button>

  <!--【1】Import the TUICallKit component: Call interface UI -->
  <TUICallKit style="width: 650px; height: 500px " />
</template>
```

2. 调用 TUICallKitServer.init API 登录组件，需要在代码中 **填写** **SDKAppID、SDKSecretKey** **两个参数。**



```
<script setup>
import { ref } from 'vue';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to

const SDKAppID = 0;       // TODO: Replace with your SDKAppID (Notice: SDKAppID is
const SDKSecretKey = '';  // TODO: Replace with your SDKSecretKey

const callerUserID = ref('');
```

```
const calleeUserID = ref('');

//【2】Initialize the TUICallKit component
const init = async () => {
  const { userSig } = GenerateTestUserSig.genTestUserSig({
    userID: callerUserID.value,
    SDKAppID,
    SecretKey: SDKSecretKey
  });
  await TUICallKitServer.init({
    userID: callerUserID.value,
    userSig,
    SDKAppID,
  });
  alert('TUICallKit init succeed');
}
</script>
```

参数	类型	说明
userID	String	**用户的唯一标识符，由您定义**，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
SDKAppID	Number	在 Tencent RTC 控制台 创建的音视频应用的唯一标识。
SDKSecretKey	String	在 Tencent RTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

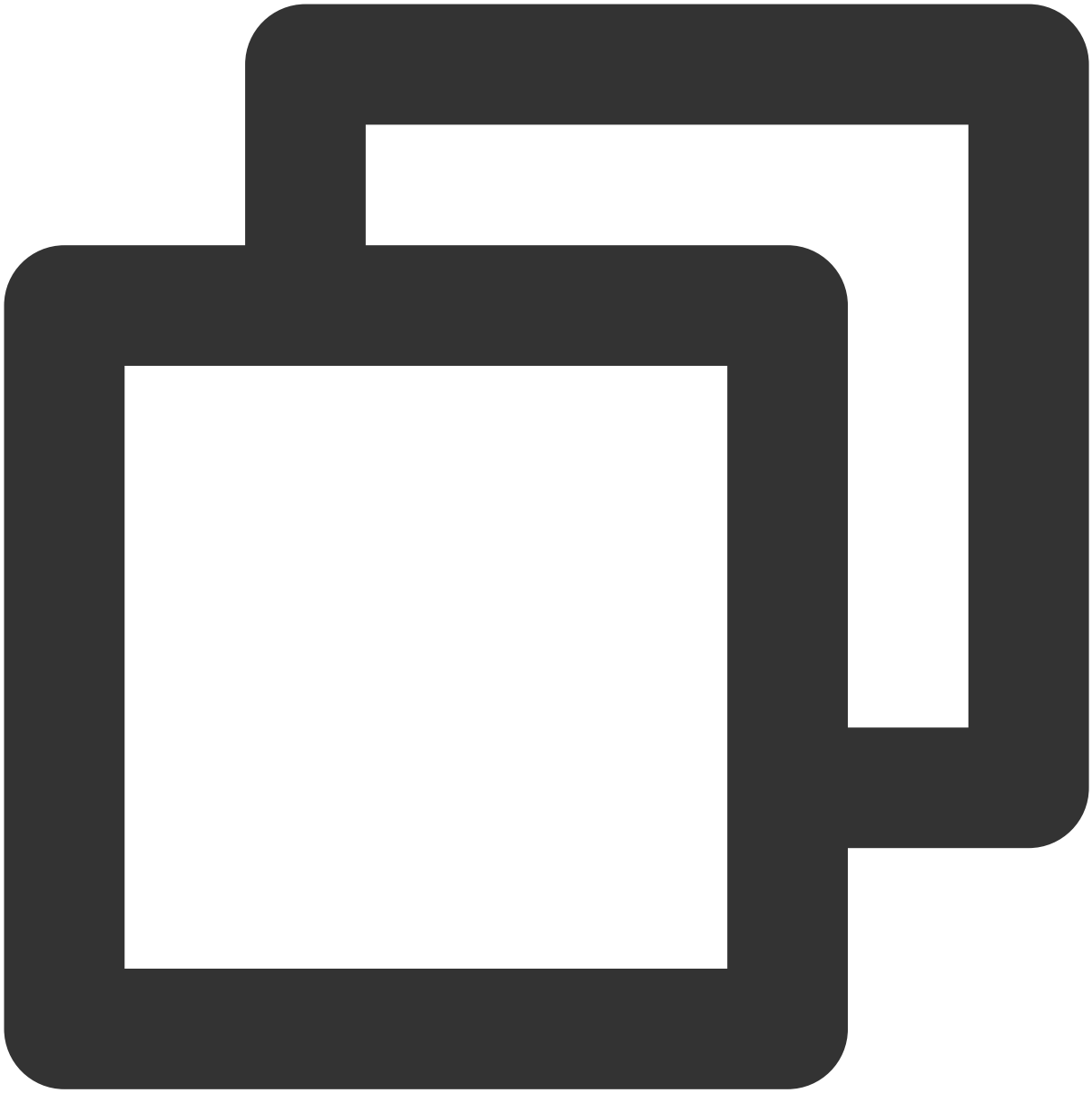
userSig 说明：

开发环境：如果您正在本地跑通 Demo、开发调试，可以采用 debug 文件中的 `genTestUserSig`（参考步骤 3.2）函数生成 userSig。该方法中 SDKSecretKey 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。

生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤四：拨打您的第一通电话

1. 调用 TUICallKitServer.call API 拨打通话。



```
//【3】Make a 1v1 video call
const call = async () => {
  await TUICallKitServer.call({
    userID: calleeUserID.value,
    type: TUICallType.VIDEO_CALL,
  });
};
```

2. 运行项目。

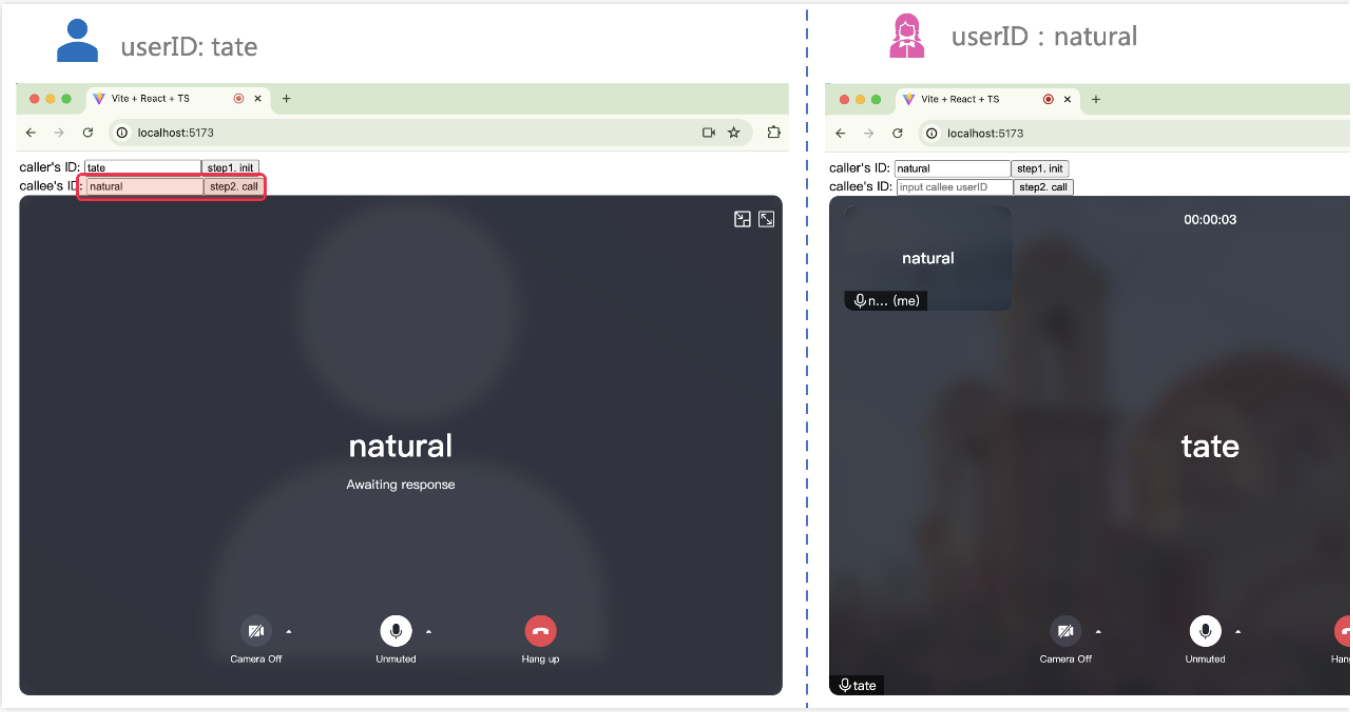
警告：

本地环境请在 localhost 协议下访问，公网体验请在 HTTPS 协议下访问，具体参见 网络访问协议说明。

3. 打开两个浏览器页面，输入不同的 userID(由您定义) 单击 `step1. init` 登录（主叫方和被叫方）。



4. 两个 userID 都登录成功后，单击 `step2. call` 拨打通话，如果您有通话问题，参见 常见问题。



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

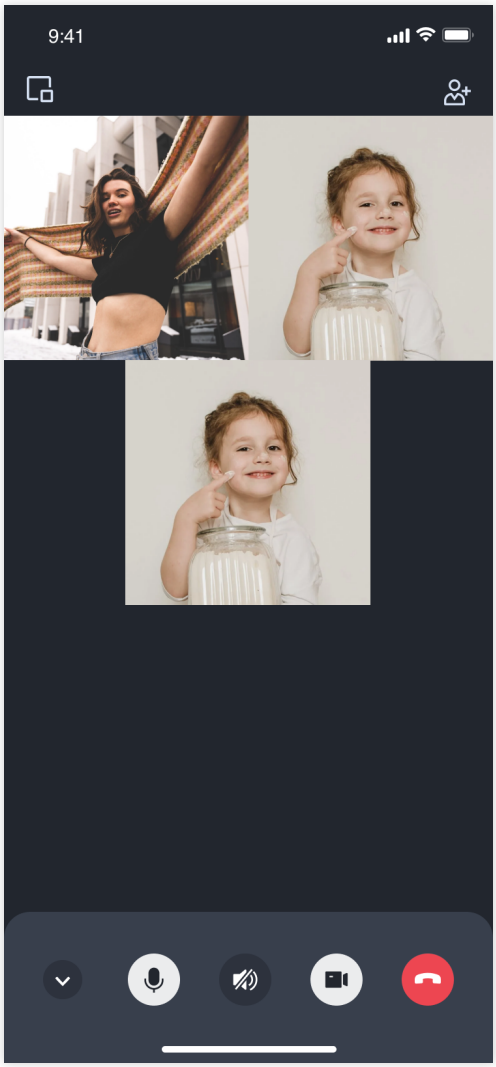
常见问题

如果您的接入和使用中遇到问题，请参见 常见问题。
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：2024-06-20 16:55:01

本文将引导您快速地完成TUICallKit组件的接入工作。跟随本文档，您可以在短短10分钟内完成接入，并最终获得一个具备完整用户界面以及音视频通话功能的应用程序。

视频通话	群组通话
	

环境准备

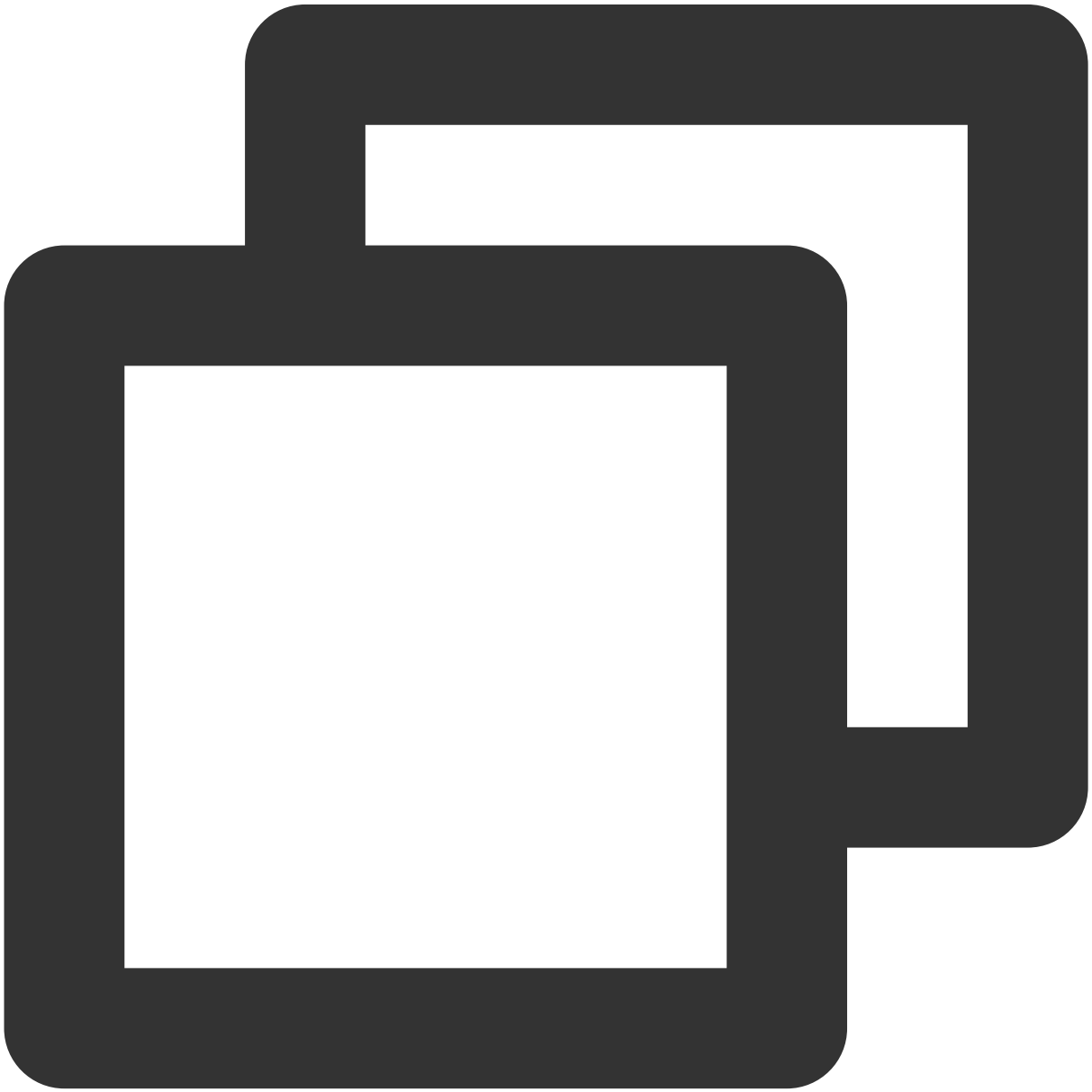
Flutter 3.0 及更高版本。

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，获取 `SDKAppID`、`SDKSecretKey`，它们将在 步骤五 中使用，具体步骤请参见 **开通服务。**

步骤二：导入 TUICallKit 组件

在工程的根目录下，通过命令行执行以下命令安装组件 tencent_calls_uikit 插件。



```
flutter pub add tencent_calls_uikit
```

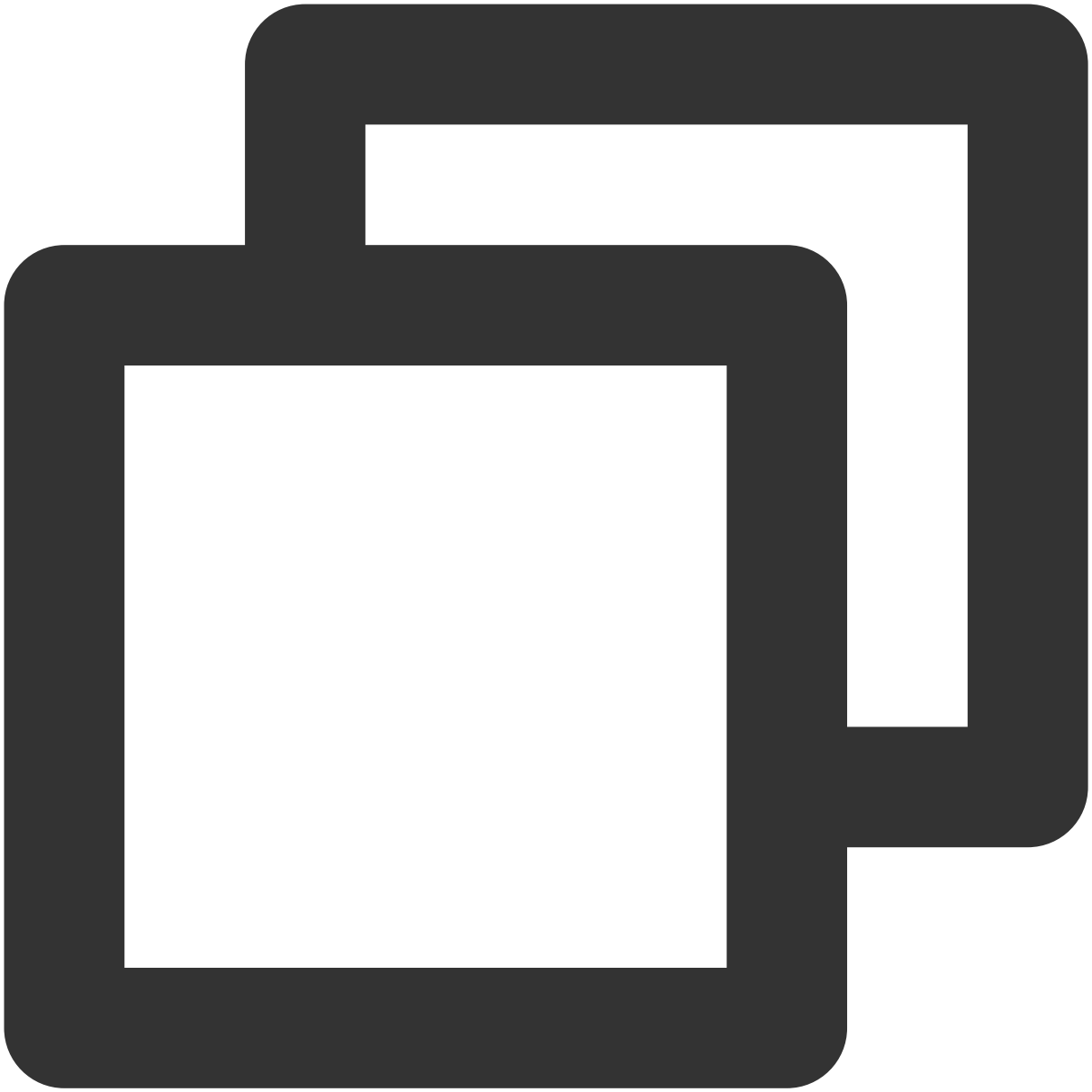
步骤三：完成工程配置

Android

iOS

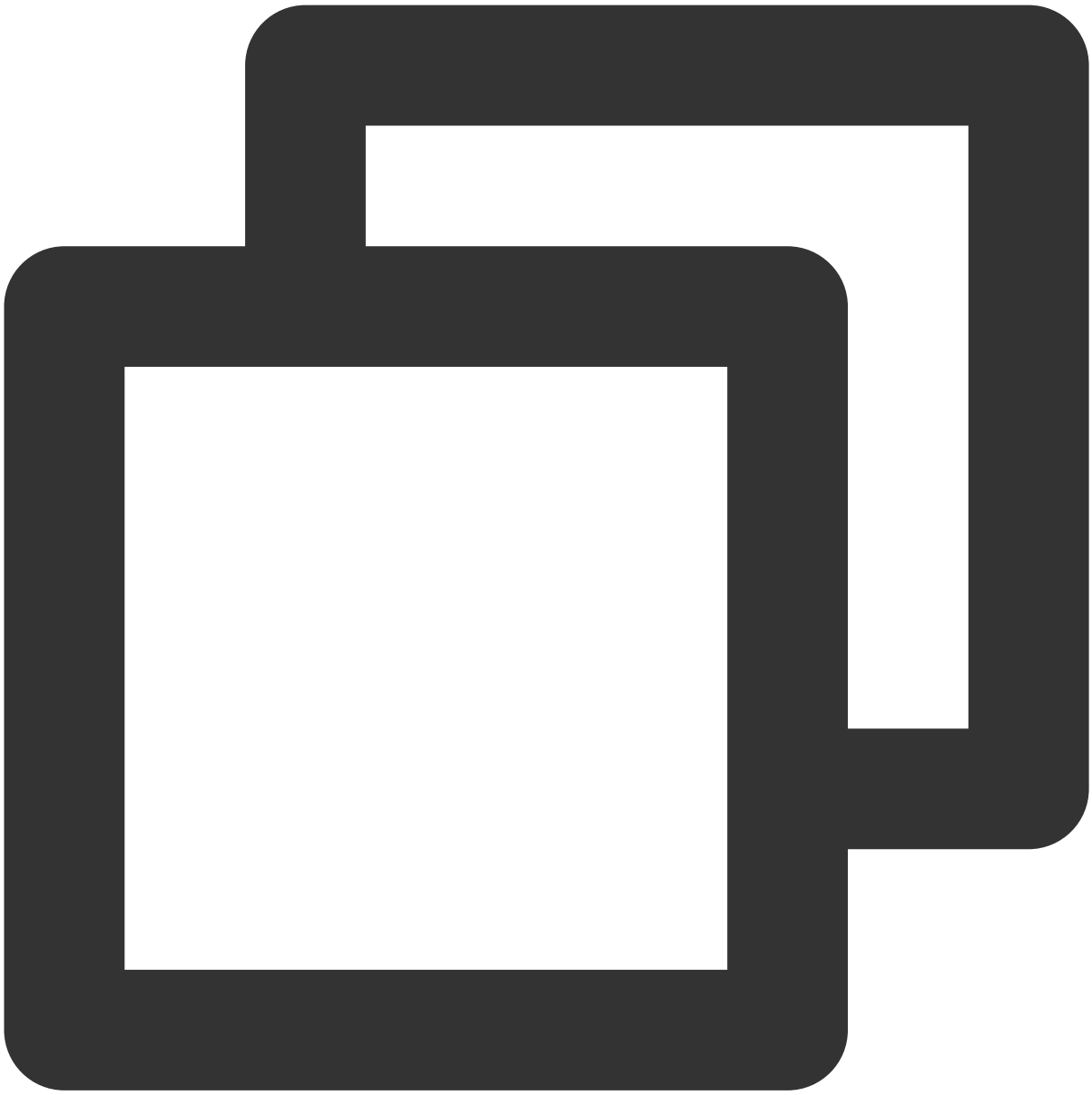
1. 如果您需要编译运行在 Android 平台，由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单。

首先，需要在工程的 `android/app/build.gradle` 文件中配置并开启混淆规则：



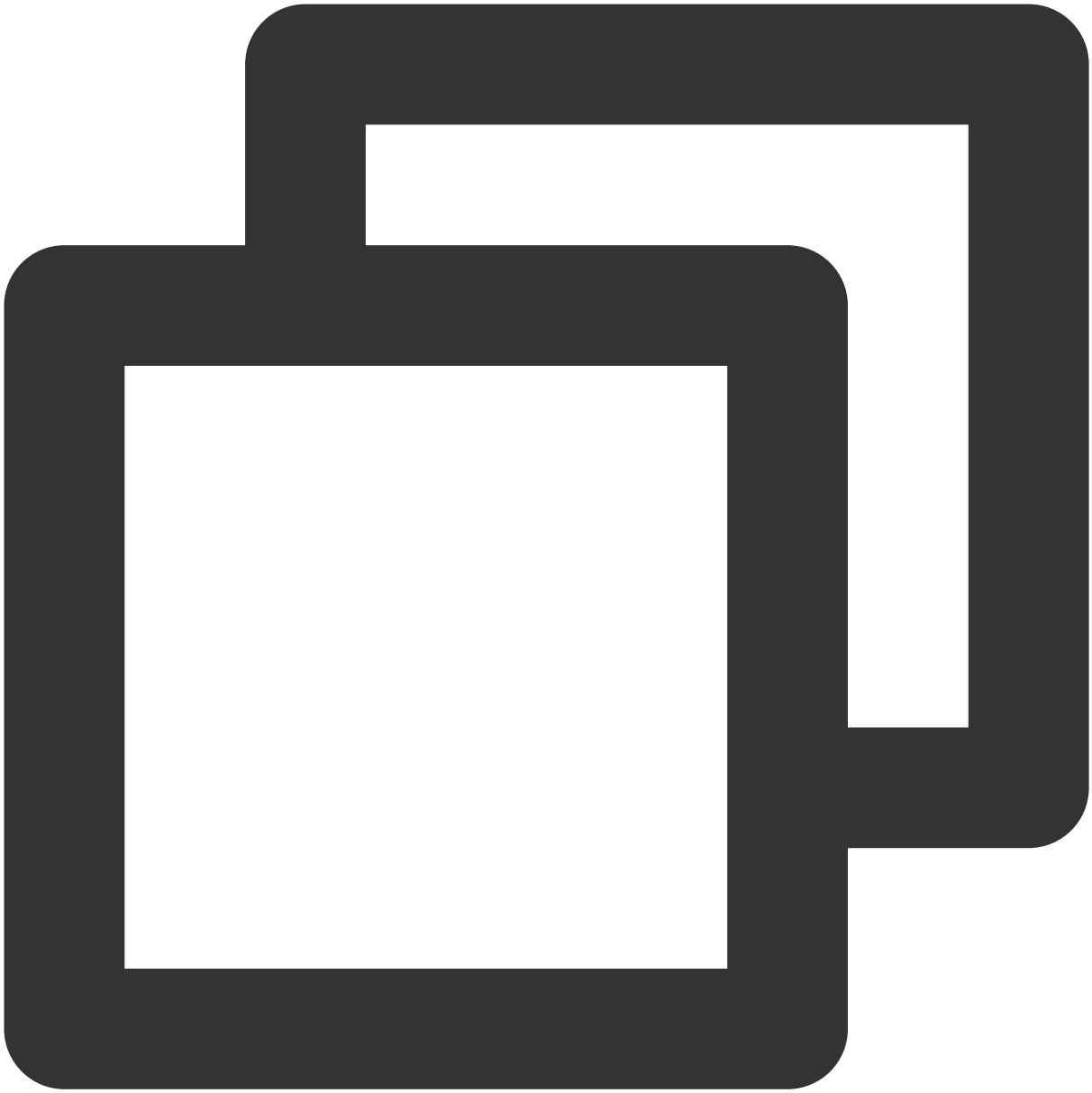
```
  android {
    ......
    buildTypes {
        release {
            ......
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard
        }
    }
}
```

在工程的 `android/app` 目录下创建 `proguard-rules.pro` 文件，并 `proguard-rules.pro` 文件中添加如下代码：



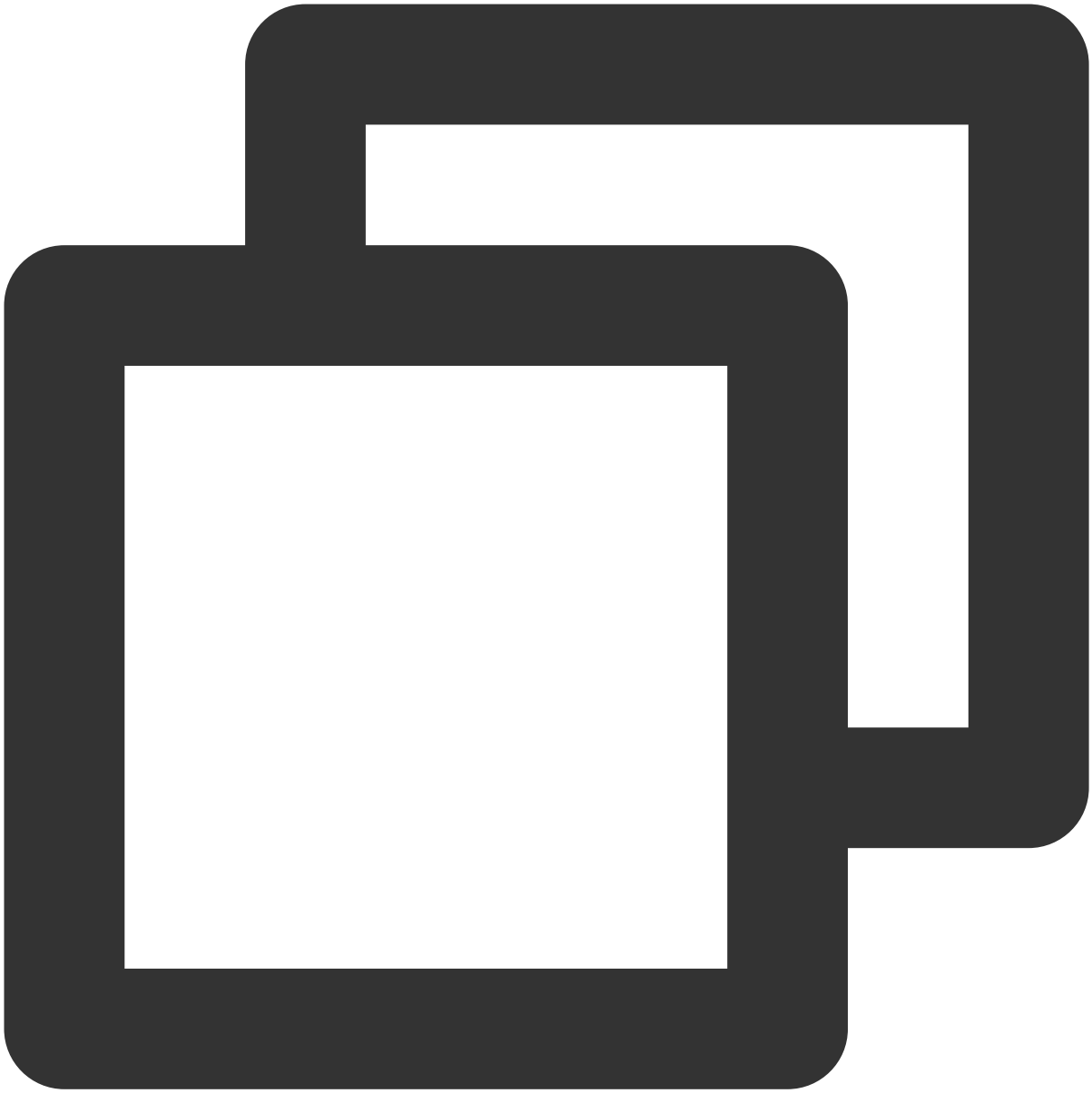
```
-keep class com.tencent.** { *; }
```

2. 在工程的 `android/app/build.gradle` 文件中配置开启 Multidex 支持。



```
  android {
    ......
    defaultConfig {
      ......
      multiDexEnabled true
    }
}
```

1. **可选**如果您需要在 iOS 模拟器上调试，并且您使用的 Mac 电脑使用的是 Intel 芯片，您需要在工程的 `ios/Podfile` 文件中添加以下代码：

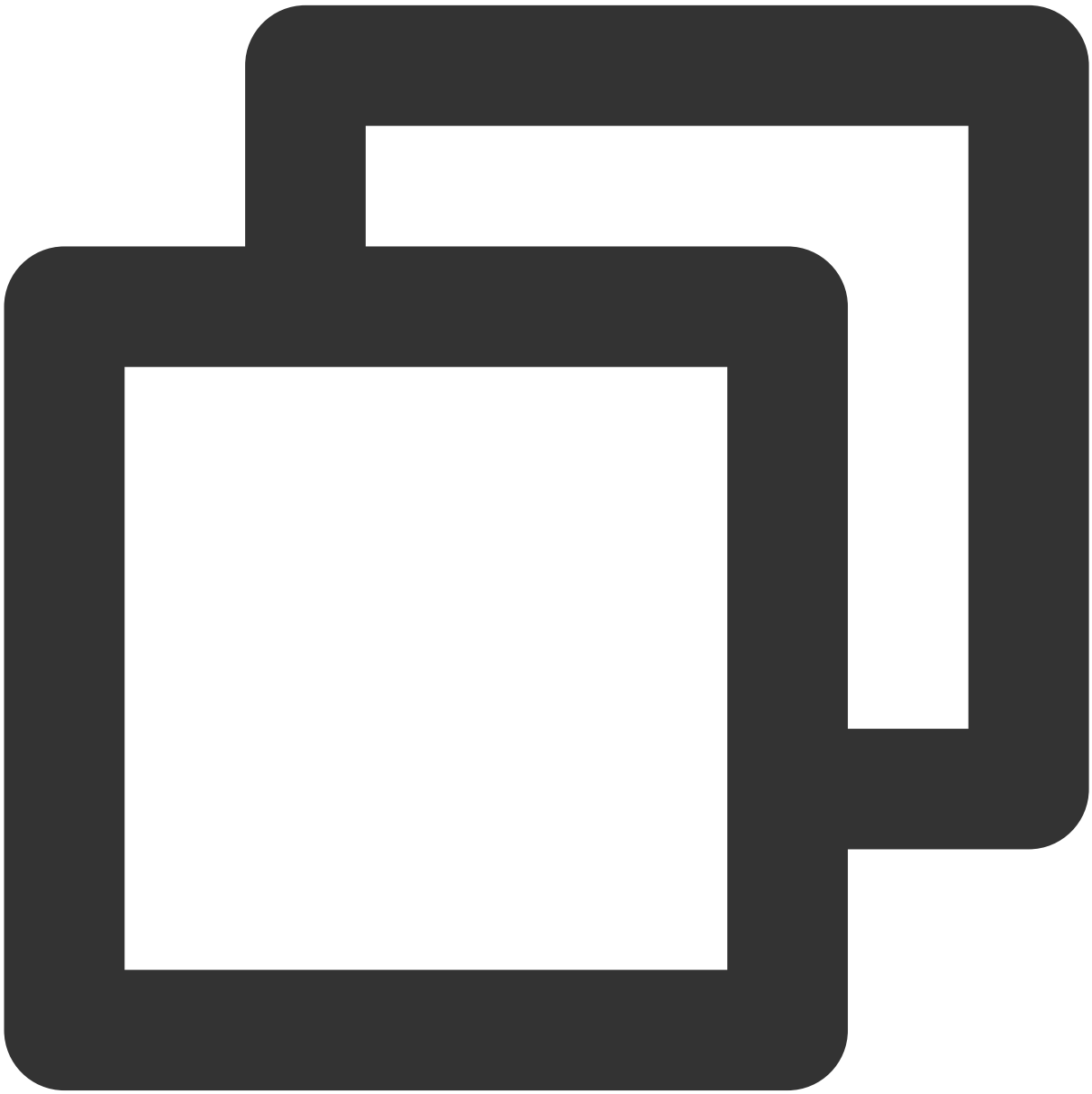


```
target 'xxxx' do
  ......
end
......

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
```

2. 由于 TUICallKit 会使用 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` 的第一级 `<dict>` 目录下添加以下两项，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。

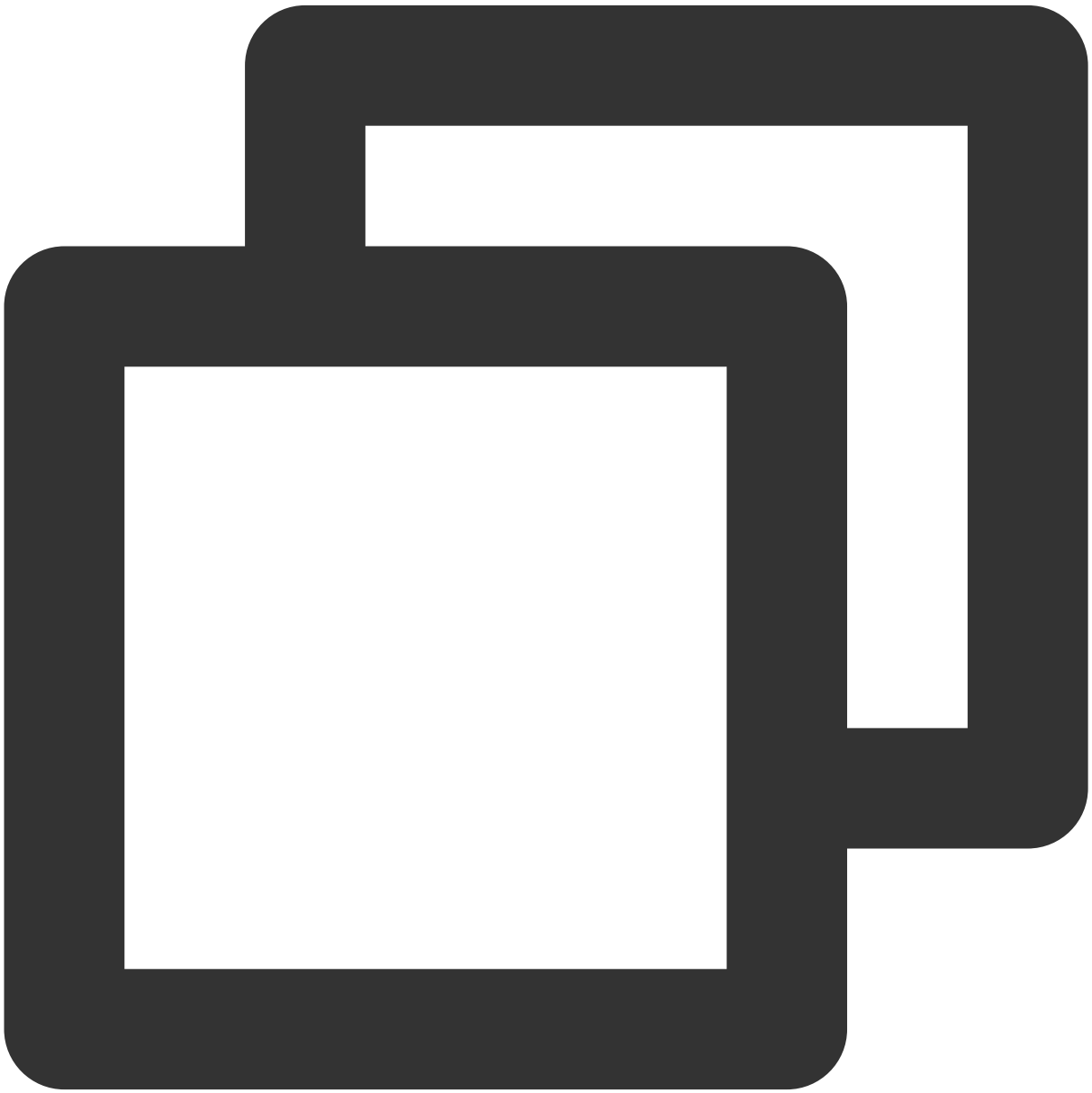


```
<key>NSCameraUsageDescription</key>
<string>CallingApp需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
```

```
<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：设置 navigatorObservers

1. 在 Flutter 应用框架的 navigatorObservers 中添加 TUICallKit.navigatorObserver，以 MateriaApp 框架为例，代码如下：



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

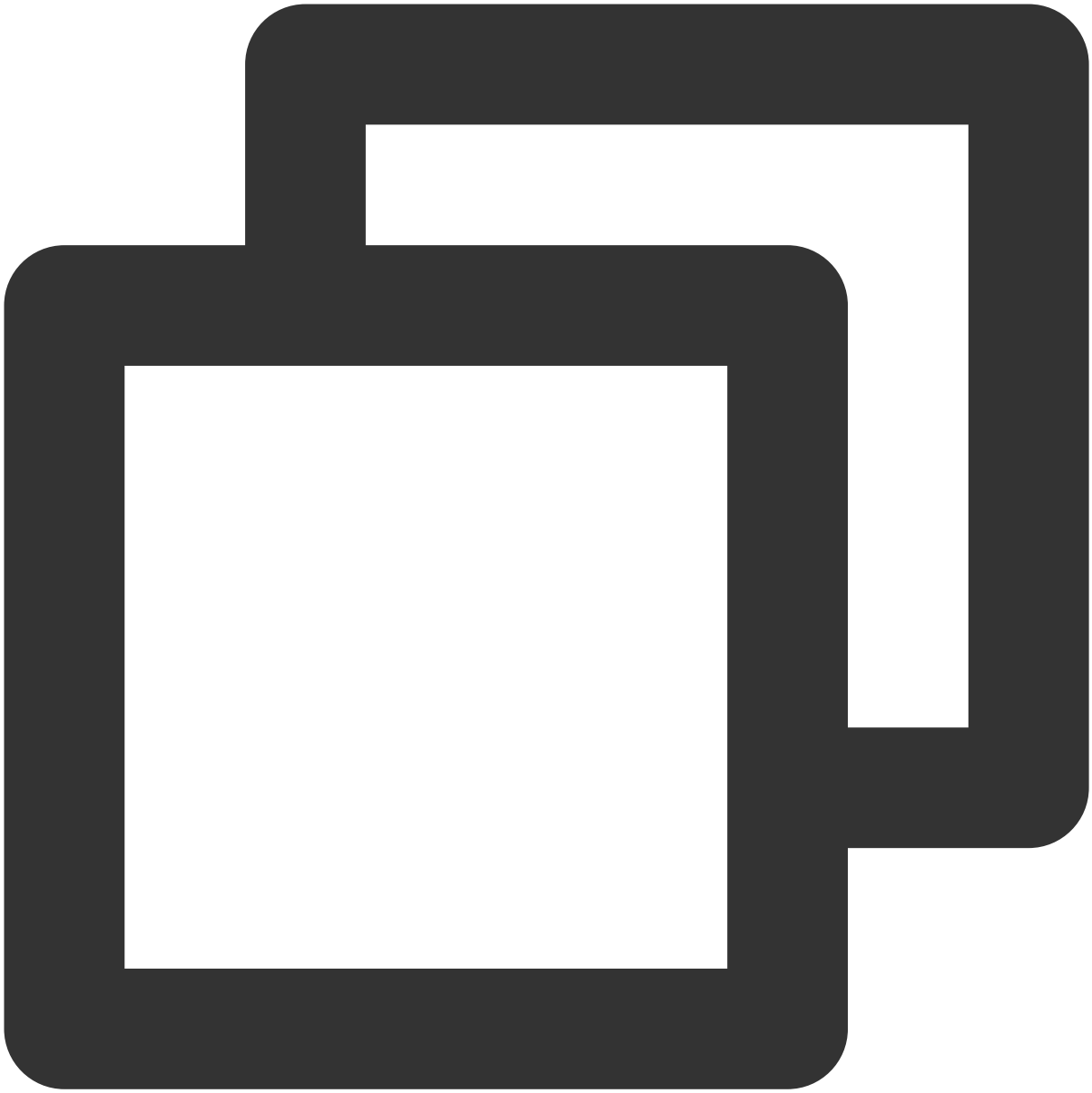
```
  ......

class XXX extends StatelessWidget {
  const XXX({super.key});

 @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUICallKit.navigatorObserver],
      ......
    );
  }
}
```

步骤五：登录 TUICallKit 组件

使用 login 接口完成登录，具体使用可参考如下代码：



```
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';
import 'package:tencent_calls_uikit/debug/generate_test_user_sig.dart';
......

final String userID    = 'xxxxx';  // 请替换为您的UserId
final int    sdkAppID  = 0;        // 请替换为第一步在控制台得到的SDKAppID
final String secretKey = 'xxxx';   // 请替换为第一步在控制台得到的SecretKey

void login() async {
    String userSig  = GenerateTestUserSig.genTestSig(userID, sdkAppID, secretKey);
    TUIResult result = await TUICallKit.instance.login(sdkAppID, userID, userSig);
```

```
    if (result.code.isEmpty) {
      print('Login success');
    } else {
      print('Login failed: ${result.code} ${result.message}');
    }
}
```

参数	类型	说明
userID	String	客户根据自己的业务自定义用户 ID，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
sdkAppID	int	在 Tencent RTC 控制台 创建的音视频应用的唯一标识。
secretKey	String	在 Tencent RTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

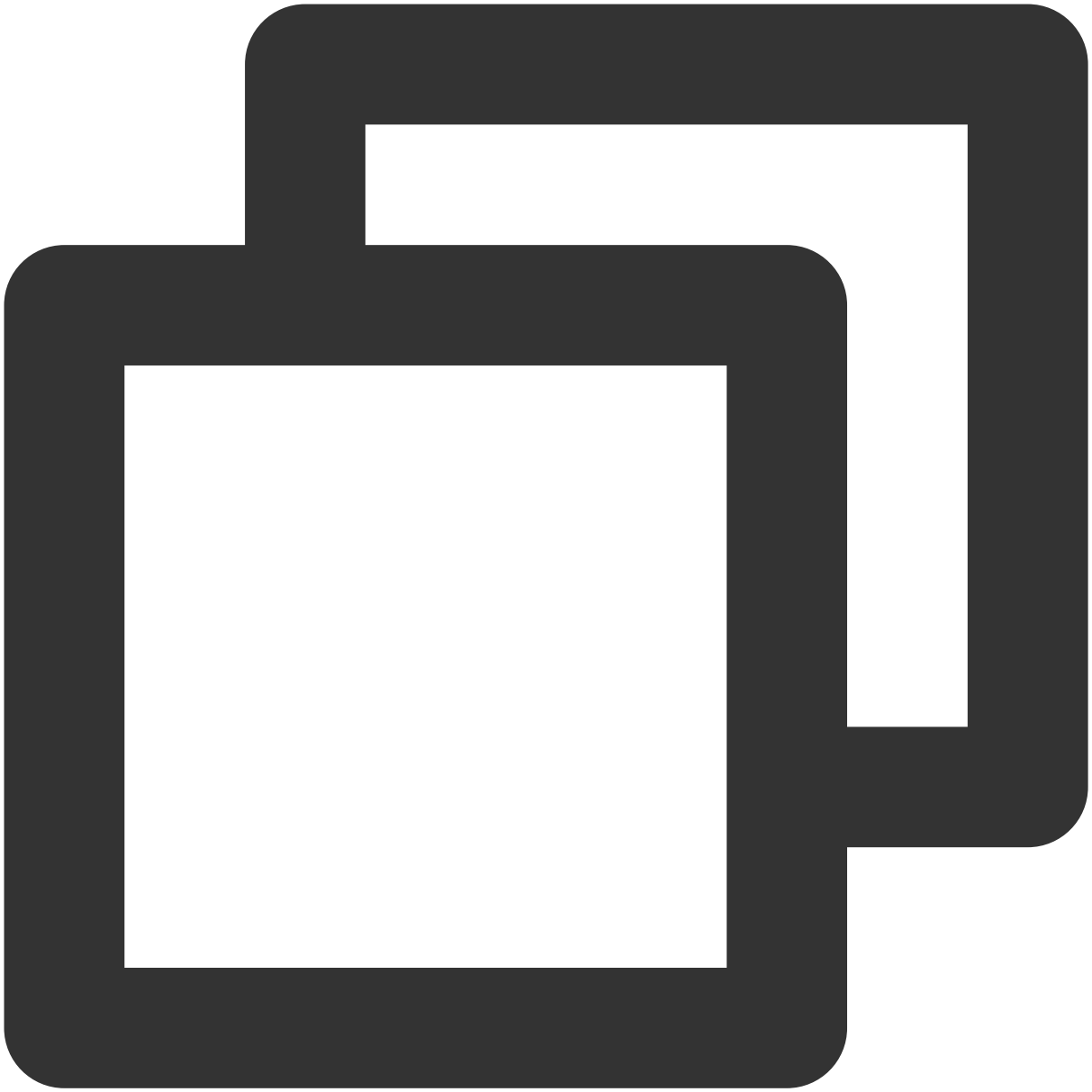
注意：

开发环境：如果您正在本地开发调试阶段，可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 userSig。该方法中 SDKSecretKey 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。

生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

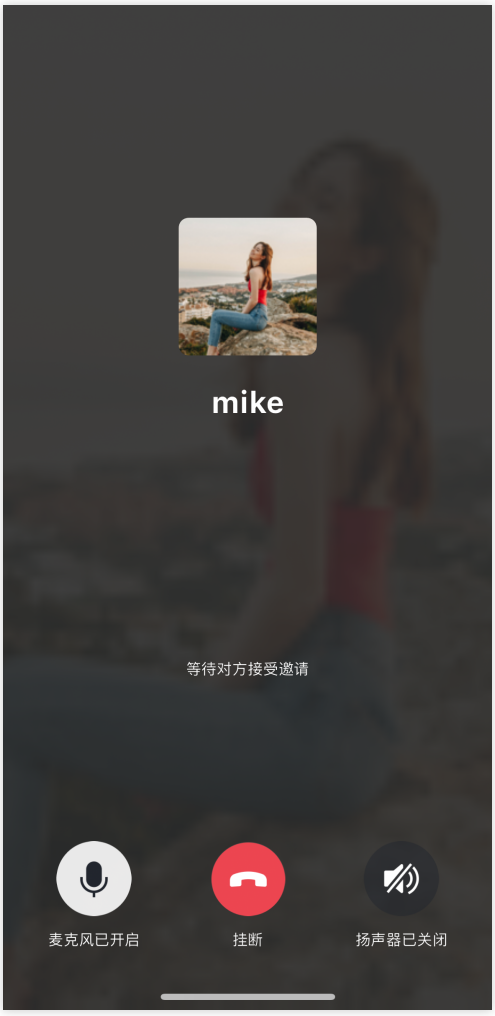
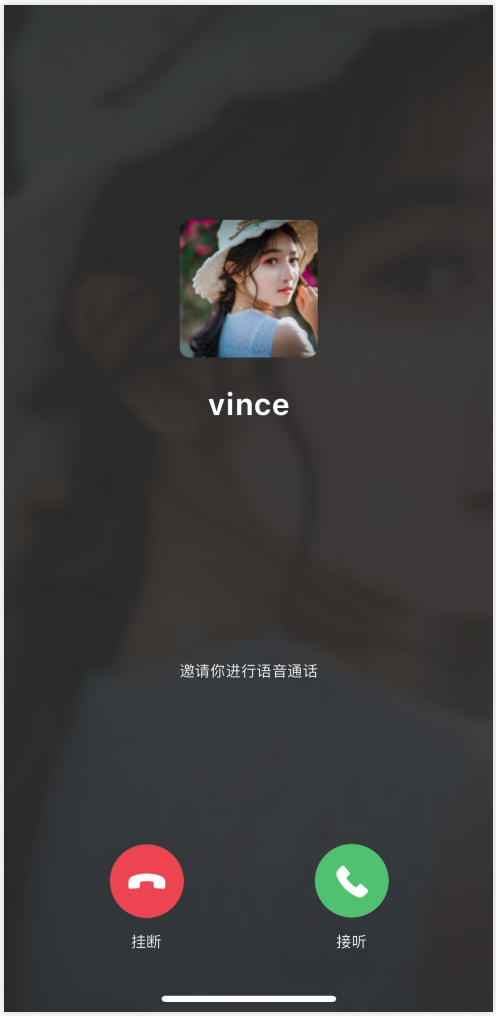
步骤六：拨打您的第一通电话

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。



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

void call() {
    TUICallKit.instance.call('vince', TUICallMediaType.audio);
}
```

	
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（Android&iOS）

最近更新时间：2024-04-28 14:56:43

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

HbuilderX 版本要求：HbuilderX 版本 ≥ 3.94。
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。

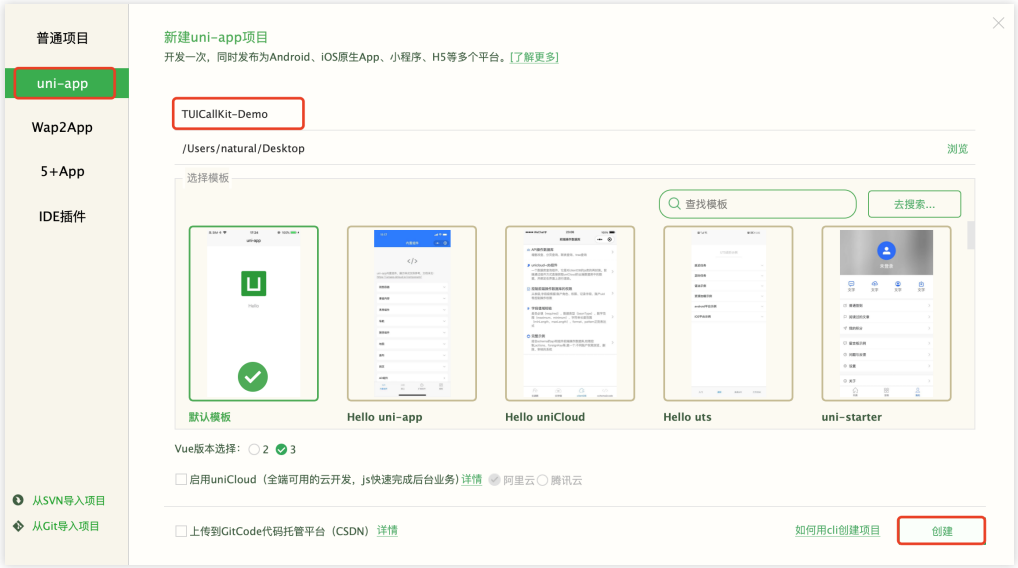
版权所有：腾讯云计算（北京）有限责任公司

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。

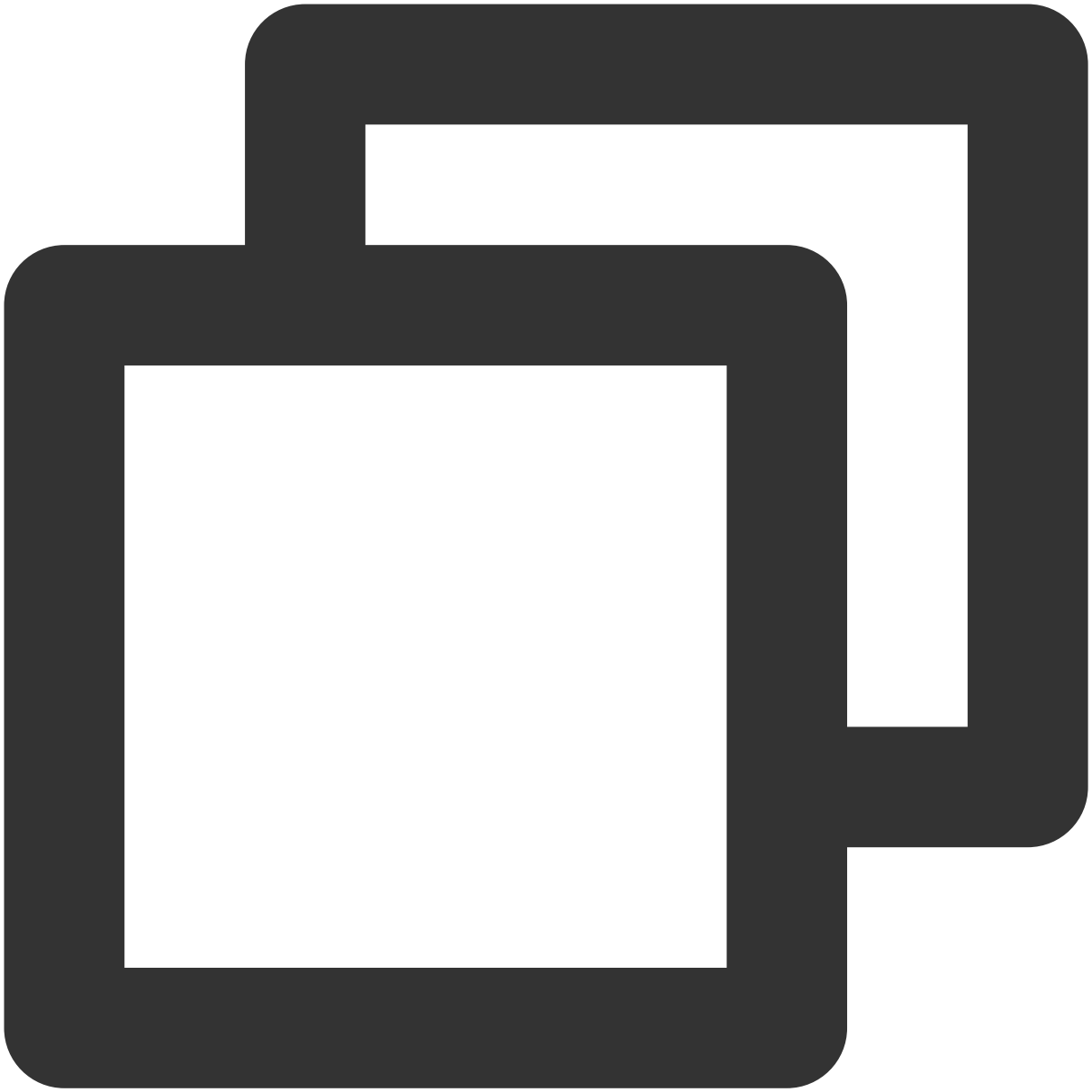


2. 在 `TUICallKit-Demo 项目` 中导入插件。



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>
```

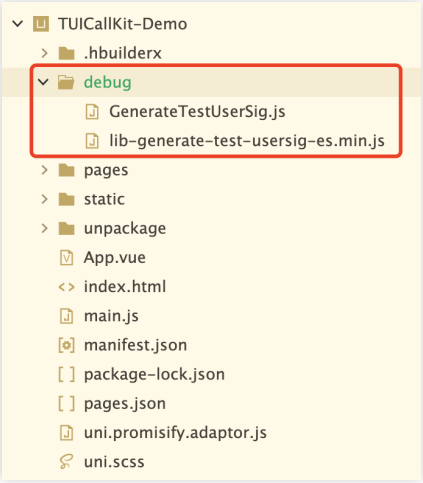
2. 填写 SDKAppID、SDKSecretKey、userSig 参数。

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;
/** ... */

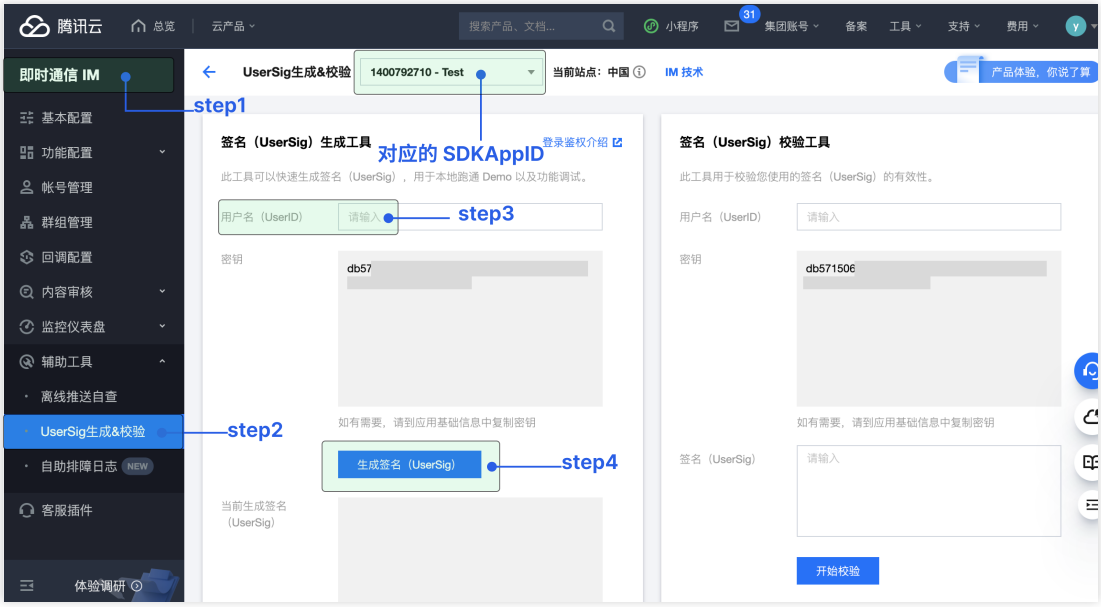
const EXPIRETIME = 604800;
/** ... */

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

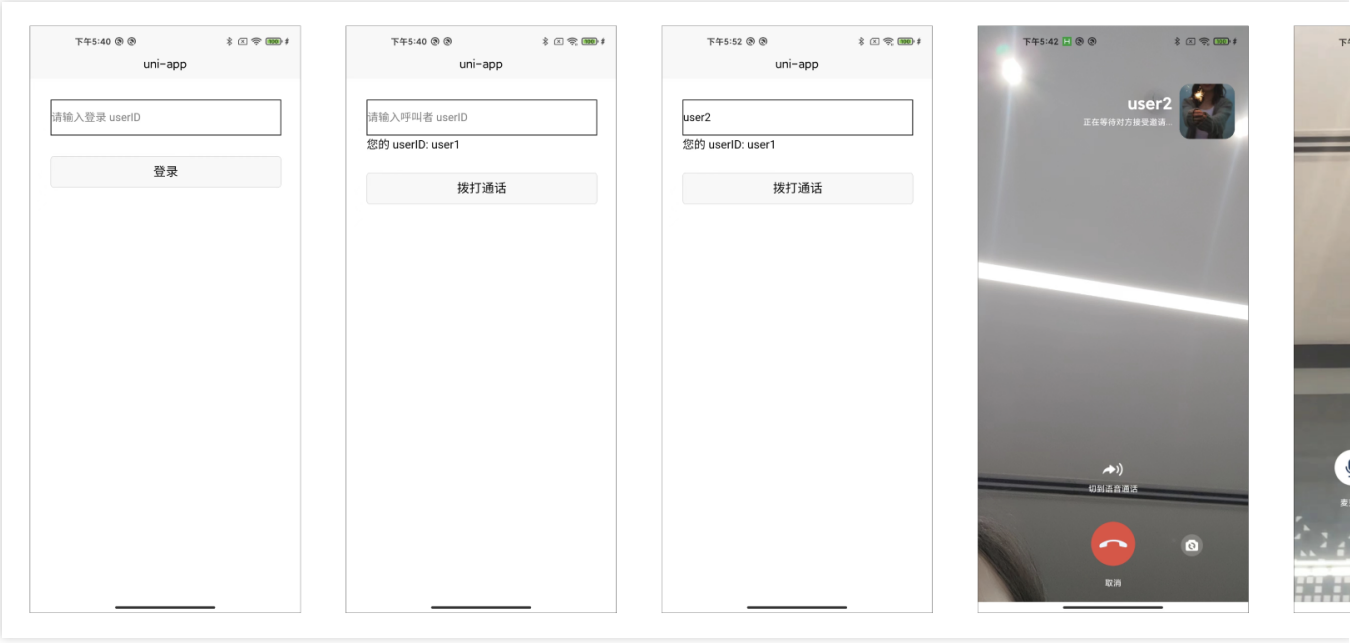
1. 制作自定义调试基座，请选择**传统打包**方式进行打包。



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：2024-05-31 16:42:21

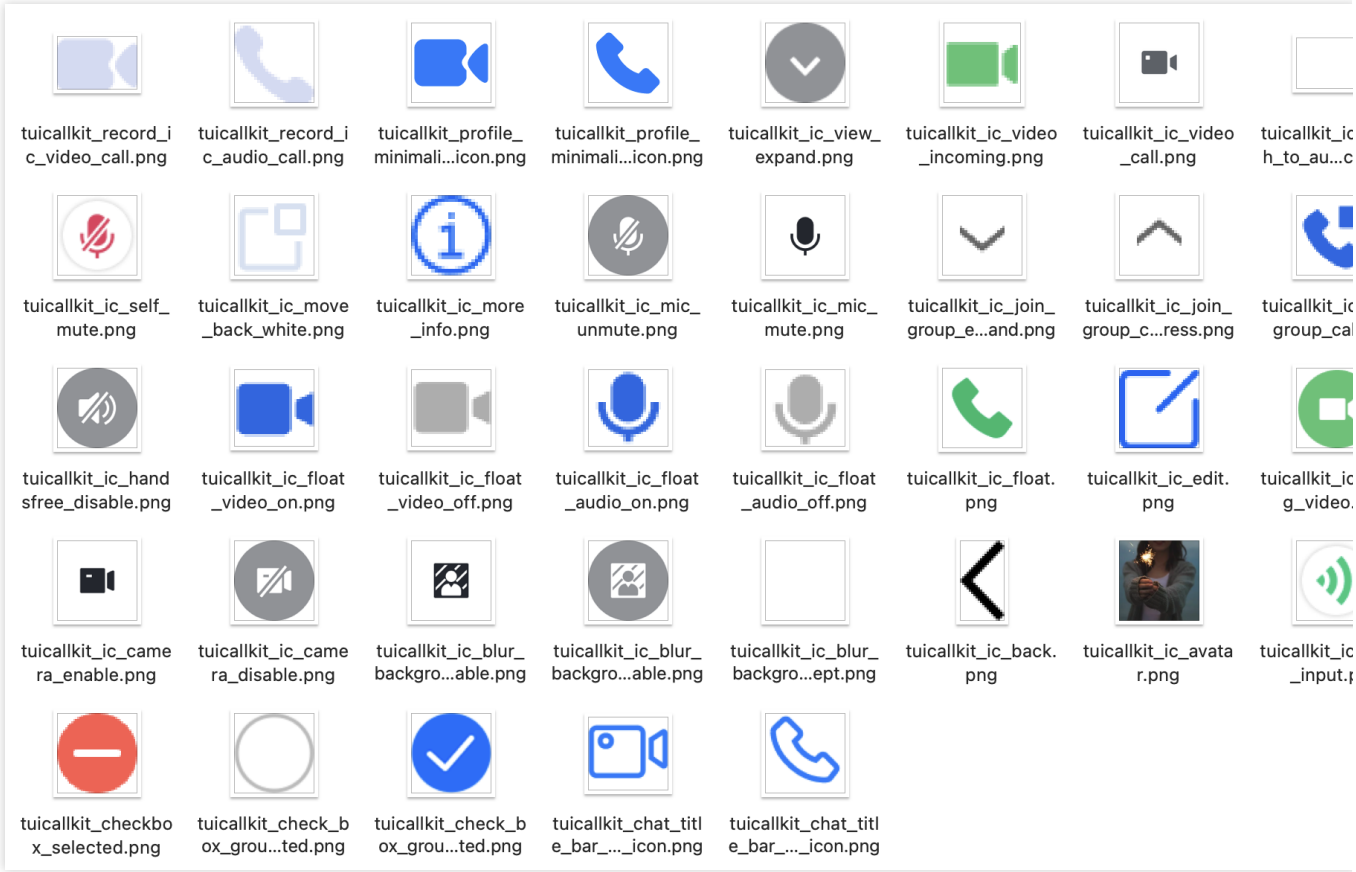
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

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-kt` 文件夹下面。

替换图标

您可以直接替换 `tuicallkit-kt/src/main/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 `tuicallkit-kt/src/main/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 `tuicallkit-kt/src/main/res/values-**/` 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 的通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以删掉 tuicallkit-kt 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1V1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
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

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
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

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2024-05-24 18:35:12

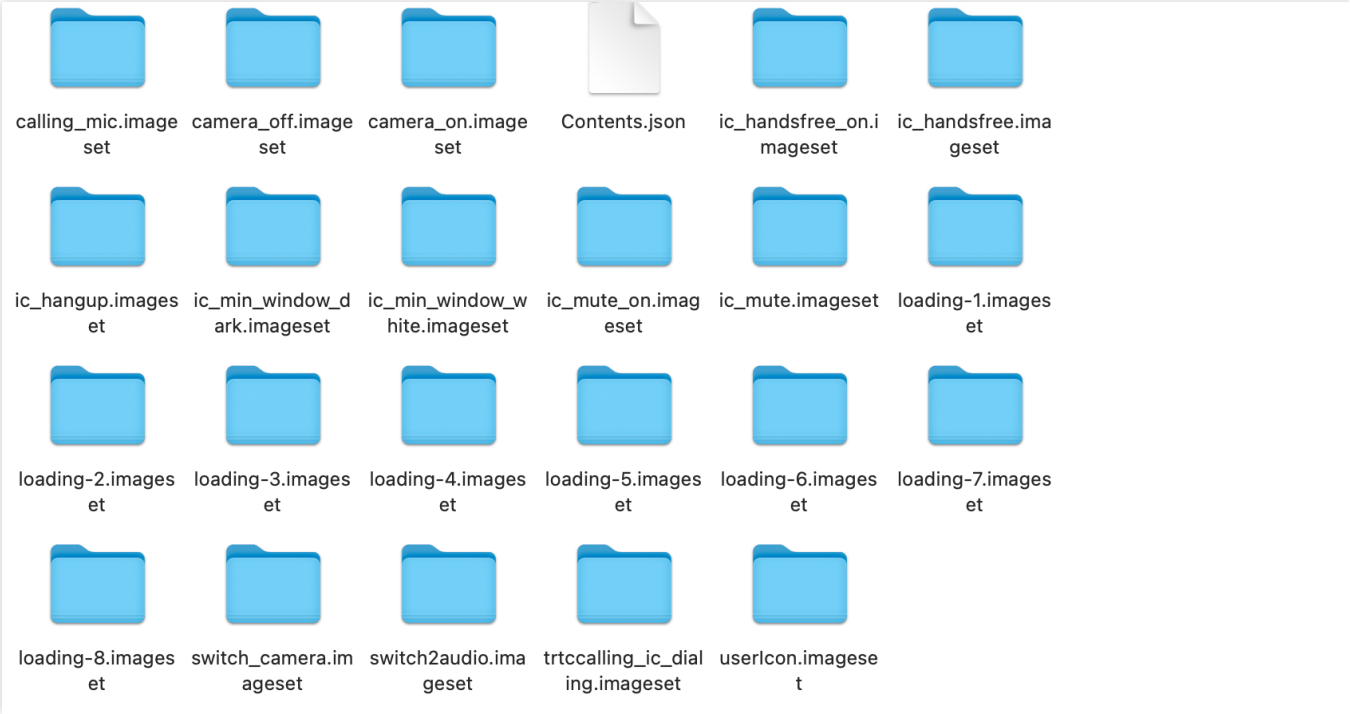
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

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-Swift` 文件夹下面：

替换图标

您可以直接替换 `Resources\\Assets.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

替换文案

您可以通过修改 zh-Hans.lproj 和 en.lproj 中的 `Localized.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

版权所有：腾讯云计算（北京）有限责任公司

Web

最近更新时间：2024-07-29 15:40:46

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

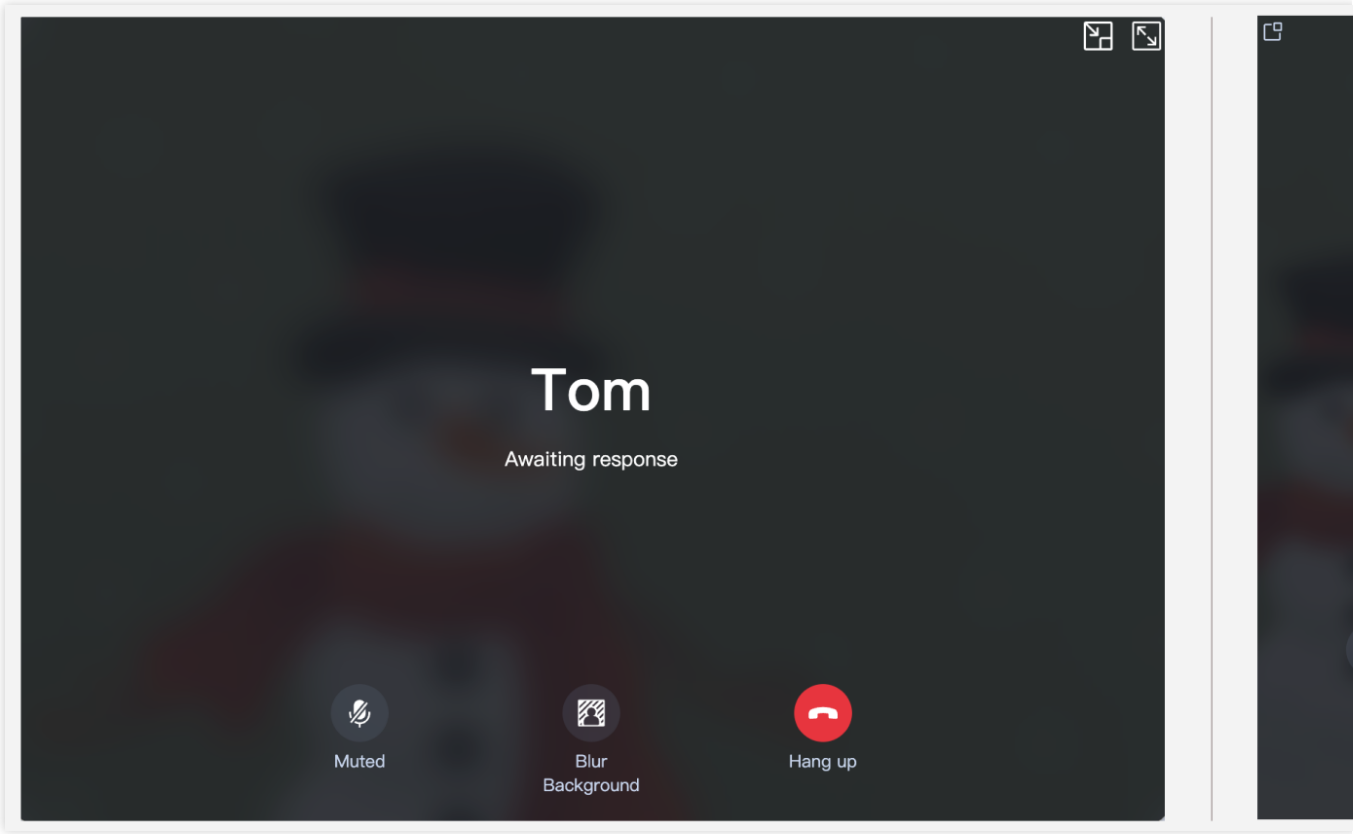
按钮隐藏

调用 hideFeatureButton 接口隐藏按钮，目前支持 Camera，Microphone，SwitchCamera，InviteUser，具体看枚举类型 FeatureButton。

注意：

v3.2.9+ 支持。

以隐藏**摄像头按钮**为例。

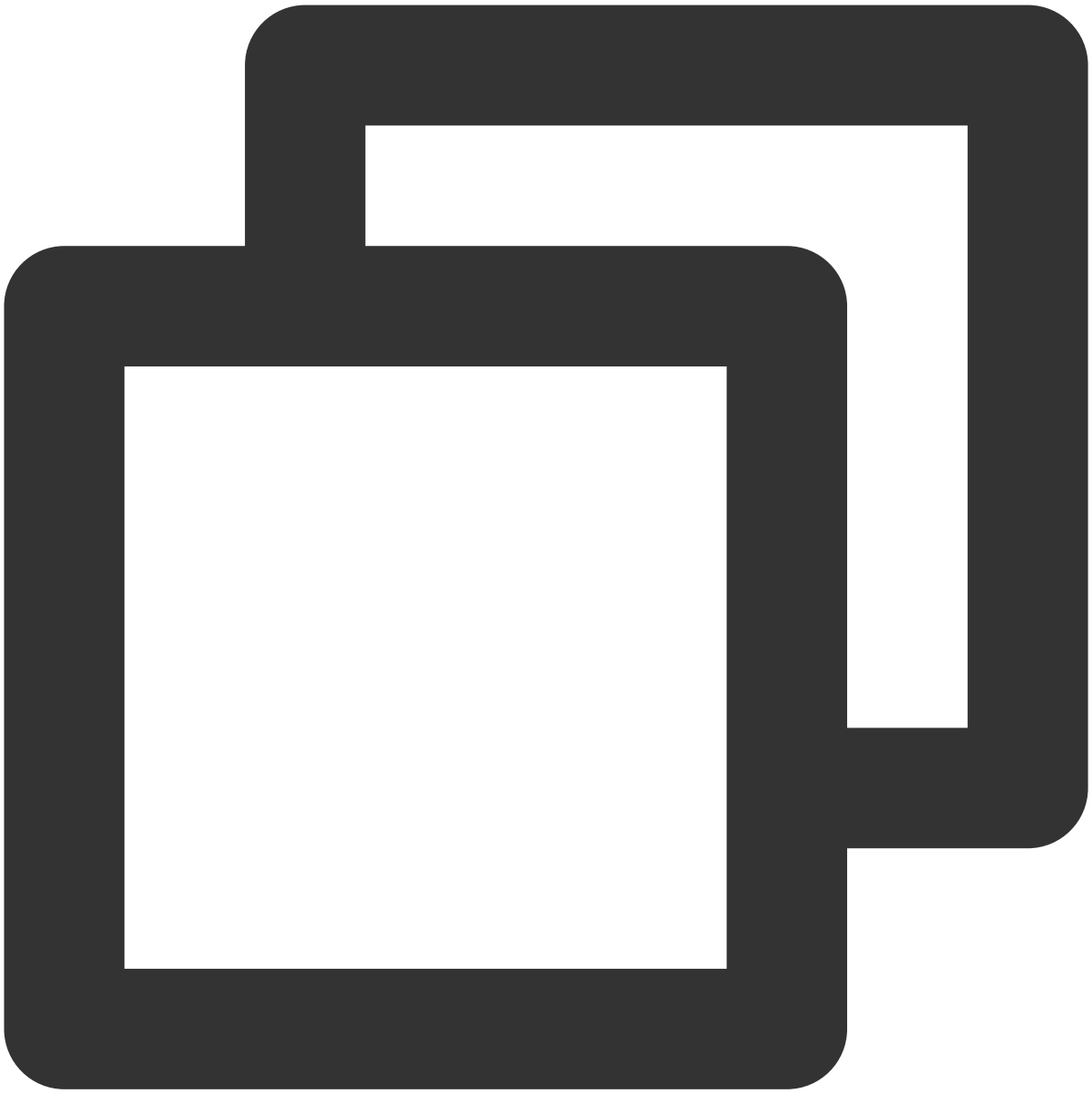


Vue3

Vue2.7

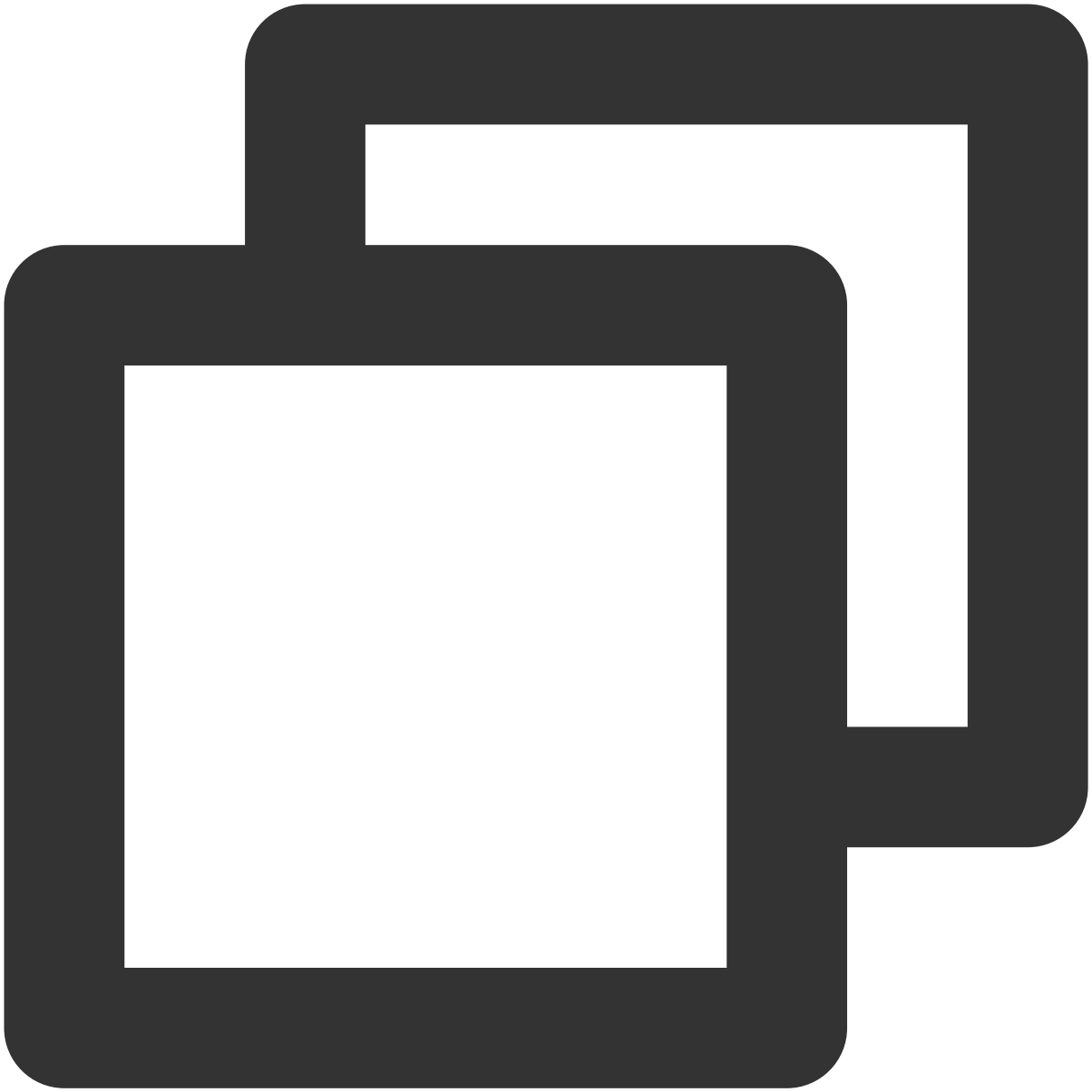
Vue2.6

React



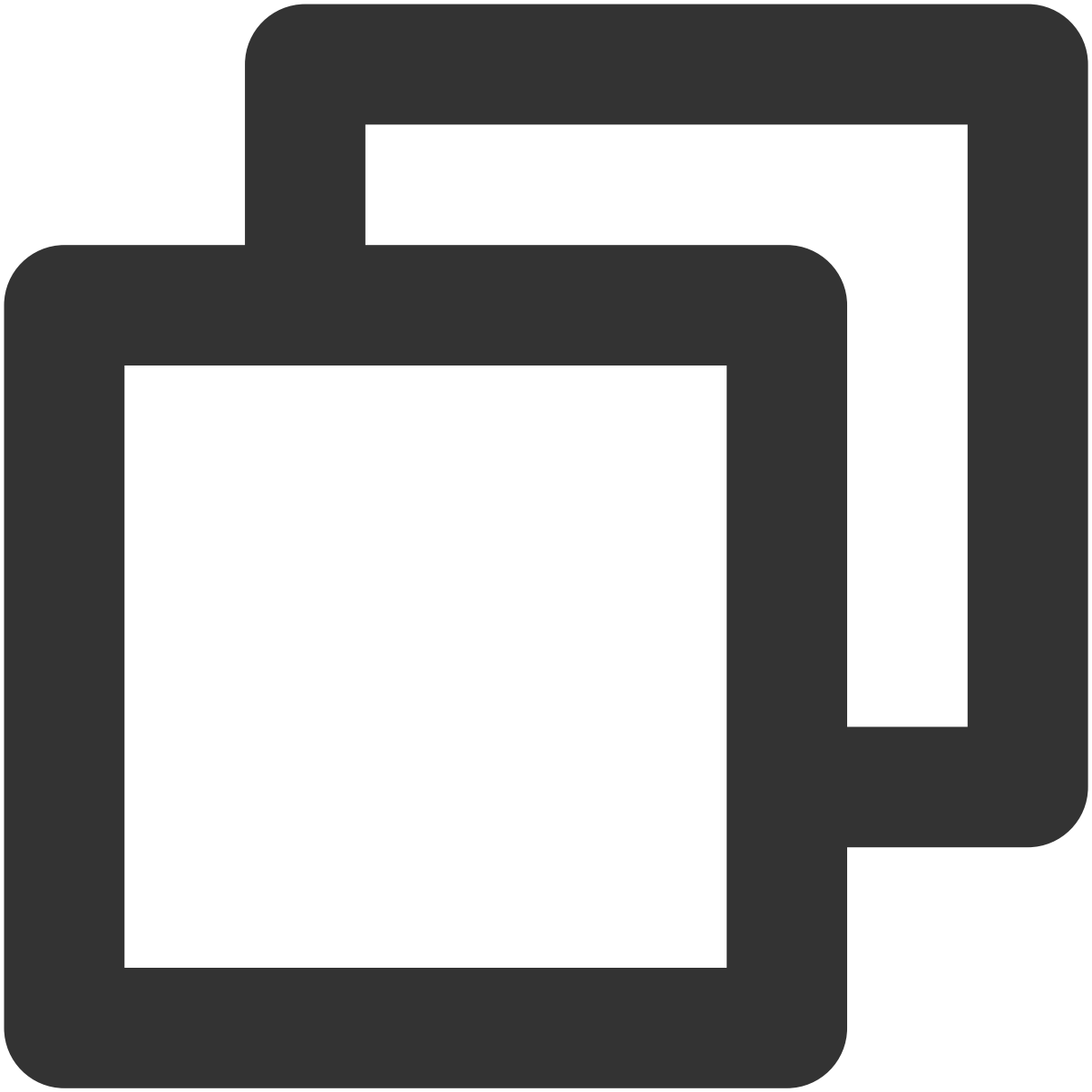
```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```



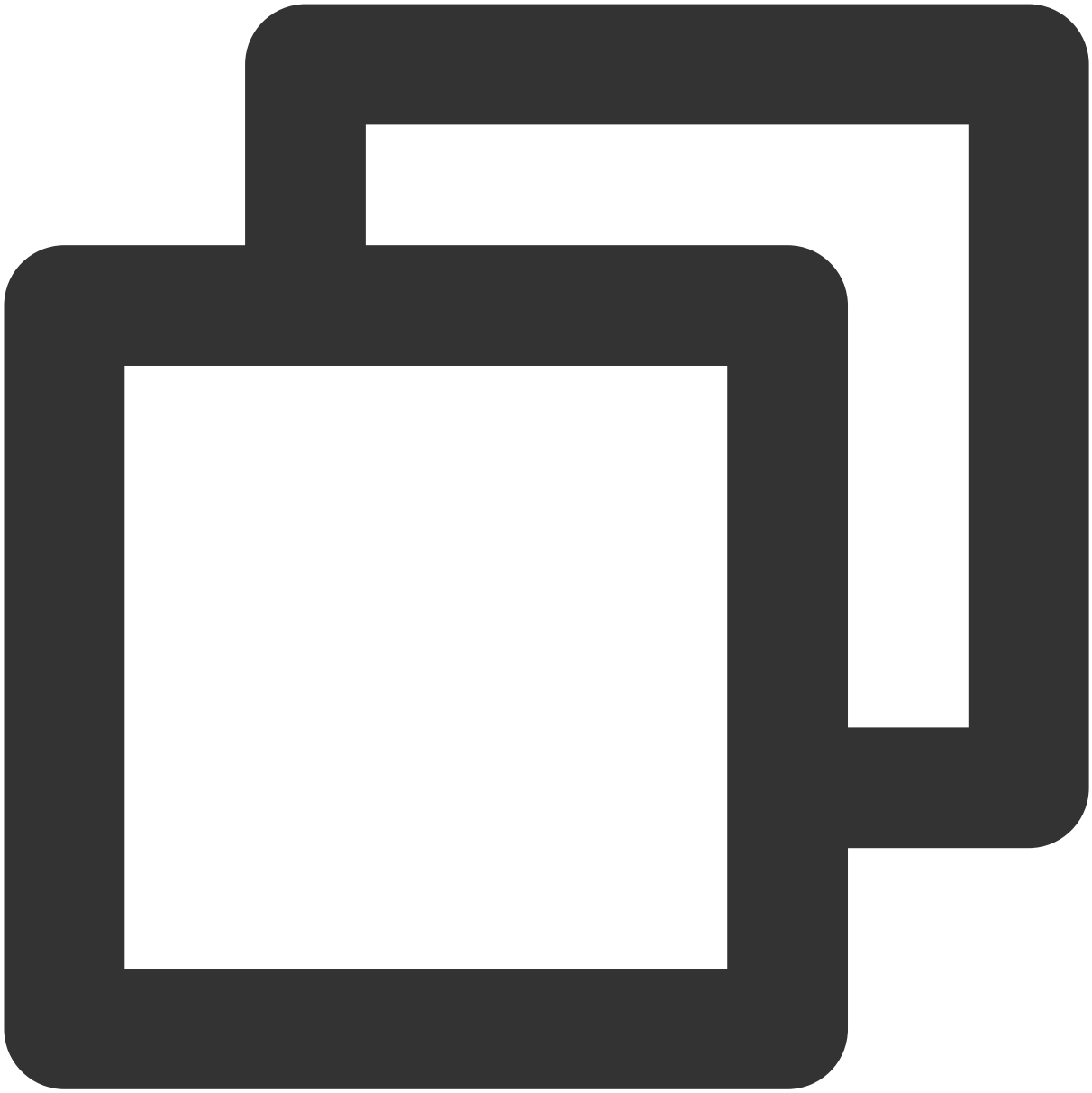
```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-vue2";

TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```



```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-vue2.6";

TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```



```
import { TUICallKitServer, FeatureButton } from "@tencentcloud/call-uikit-react";

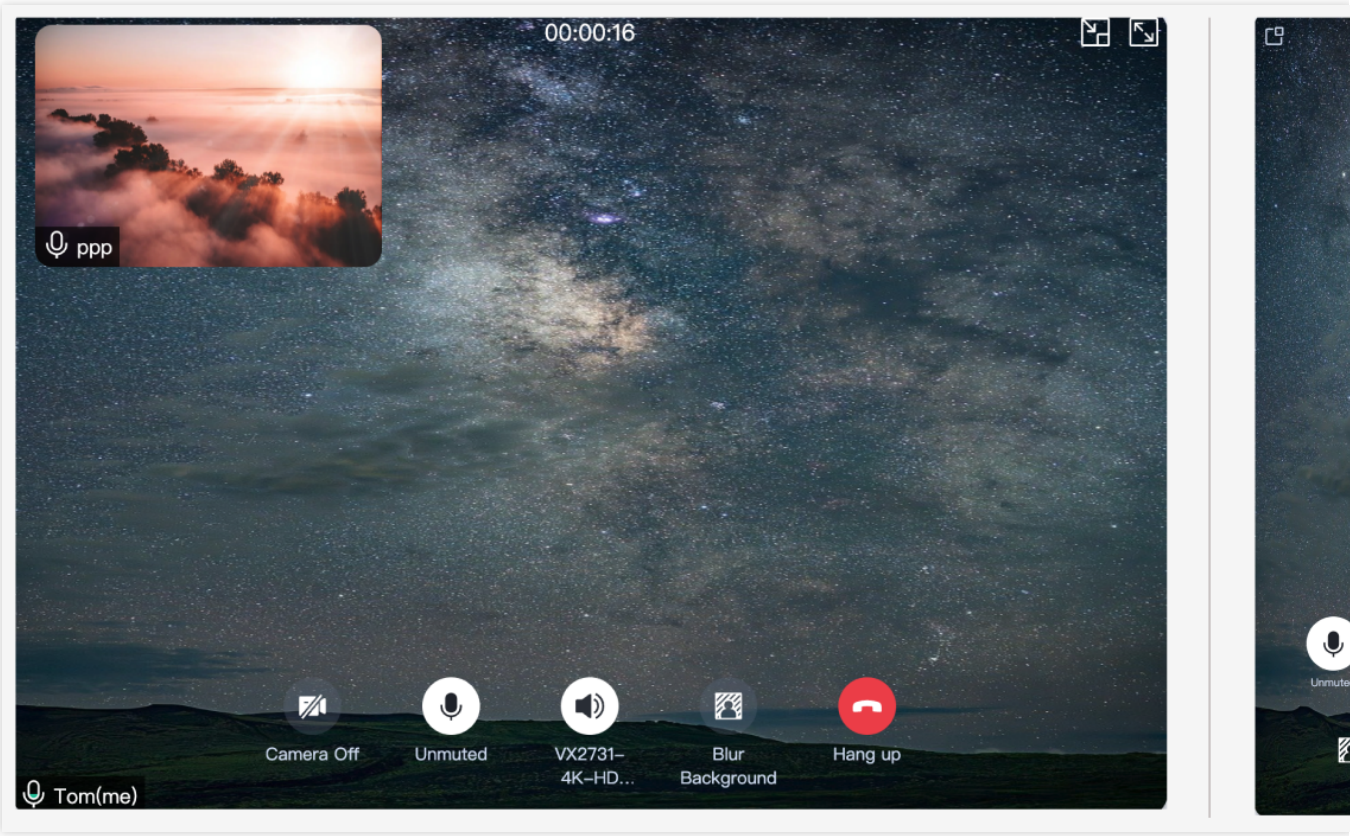
TUICallKitServer.hideFeatureButton(FeatureButton.Camera);
```

自定义通话背景图

通话背景图会在语音通话或者视频通话关闭摄像头后出现，通过调用 setLocalViewBackgroundImage 修改本地用户通话界面背景图，setRemoteViewBackgroundImage 修改远端用户通话界面背景图。

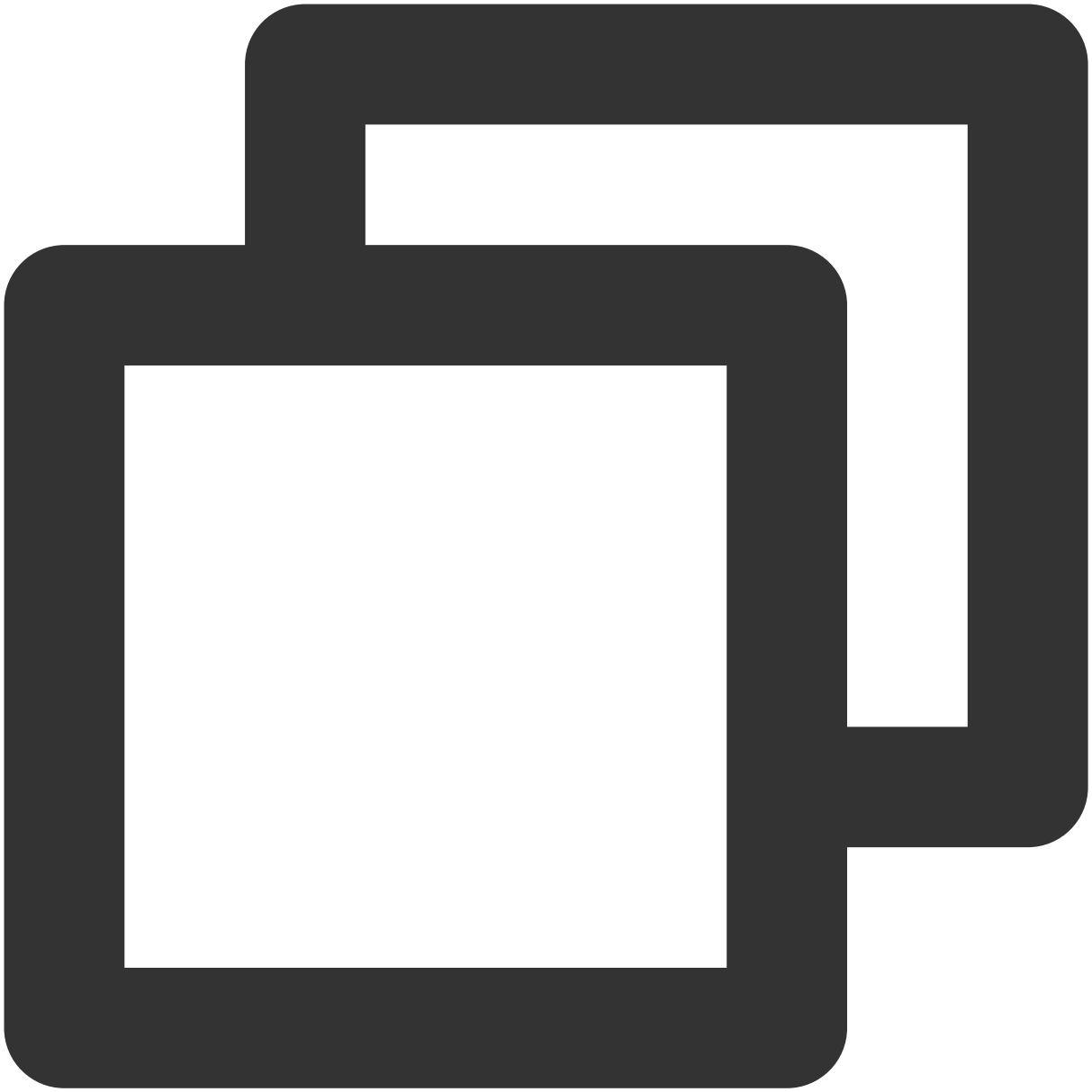
注意：

v3.2.9+ 支持。



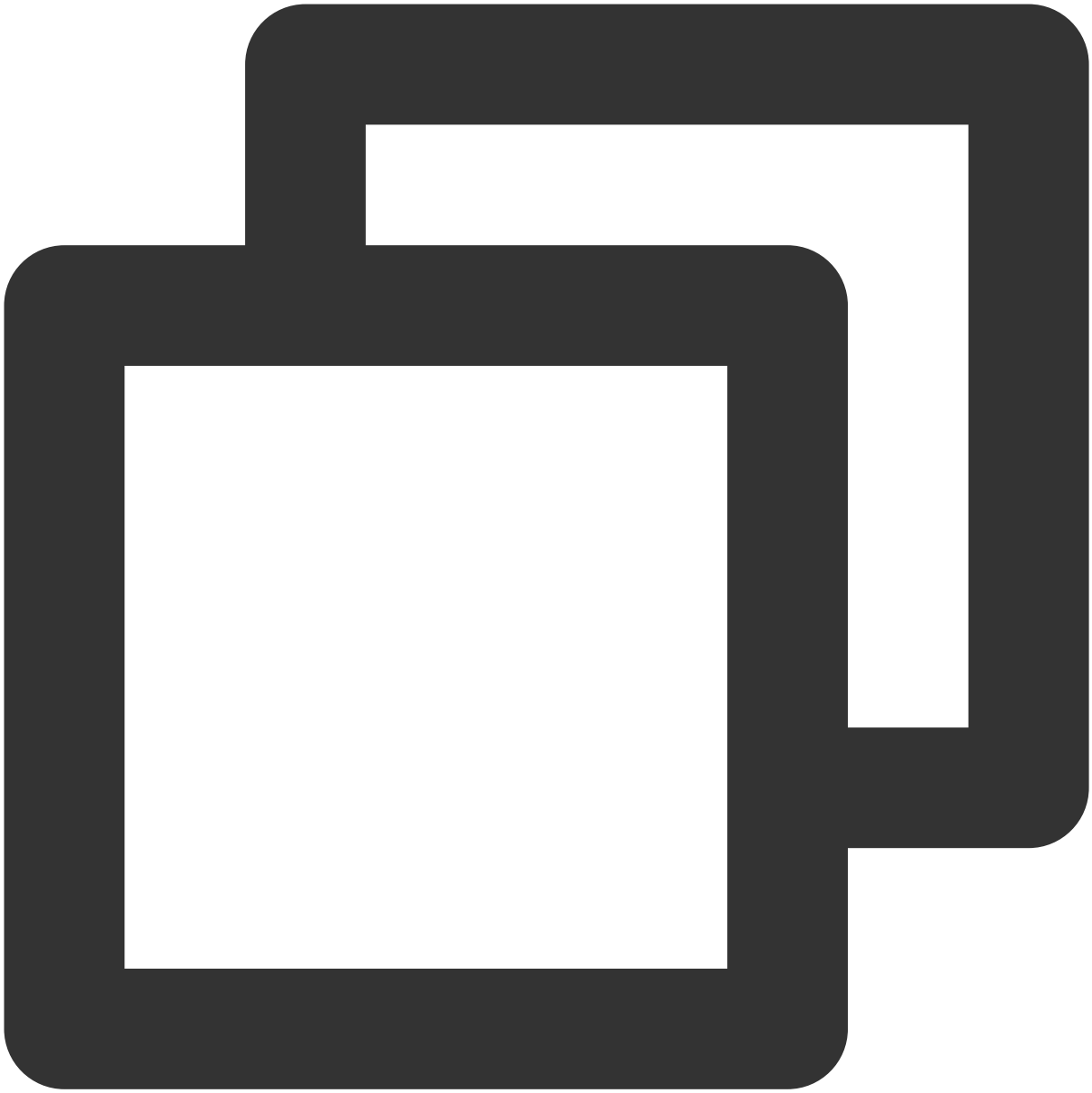
Vue3

React



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setLocalViewBackgroundImage('http://xxx.png');
TUICallKitServer.setRemoteViewBackgroundImage('remoteUserId', 'http://xxx.png');
```



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setLocalViewBackgroundImage('http://xxx.png');
TUICallKitServer.setRemoteViewBackgroundImage('remoteUserId', 'http://xxx.png');
```

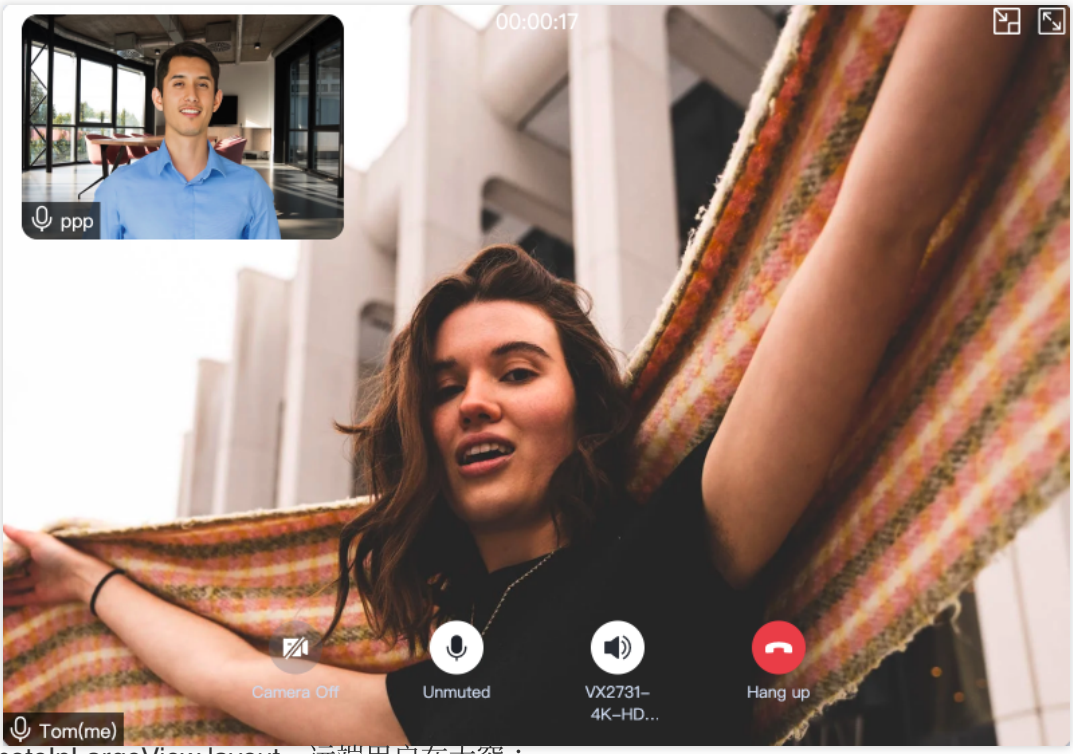
设置布局

注意：

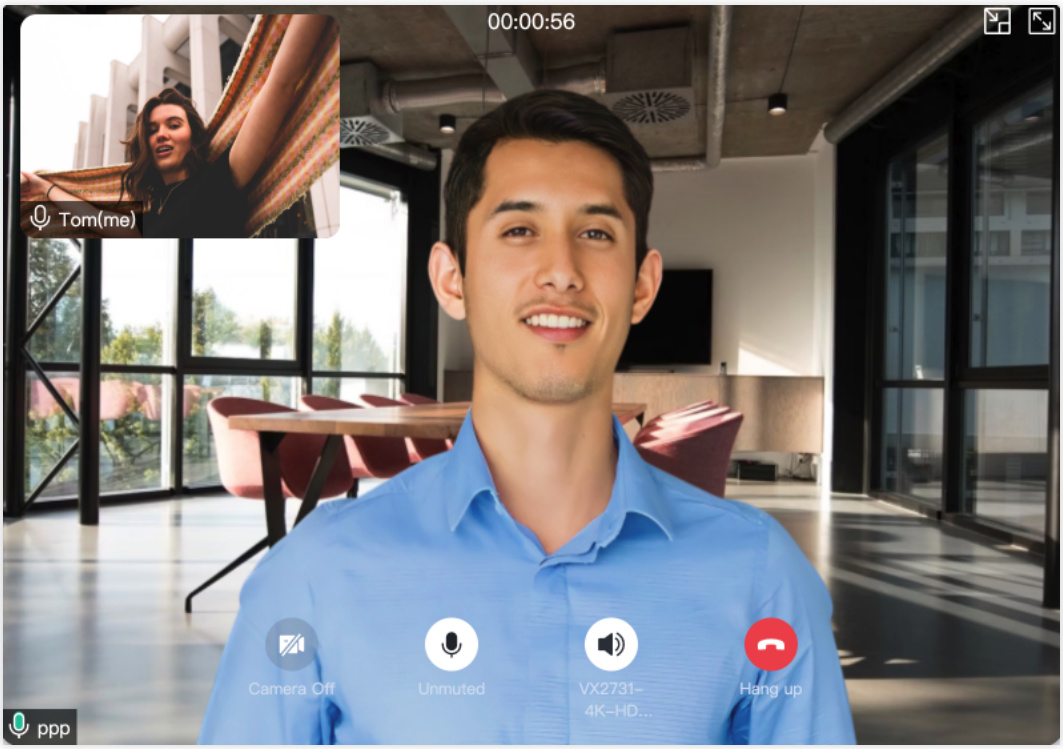
仅 1V1 视频通话可用，v3.3.0+ 支持。

调用 setLayoutMode 设置通话界面布局，目前仅支持 LocalInLargeView，RemoteInLargeView，具体看枚举类型 LayoutMode。

1. LocalInLargeView layout，本地用户在大窗：

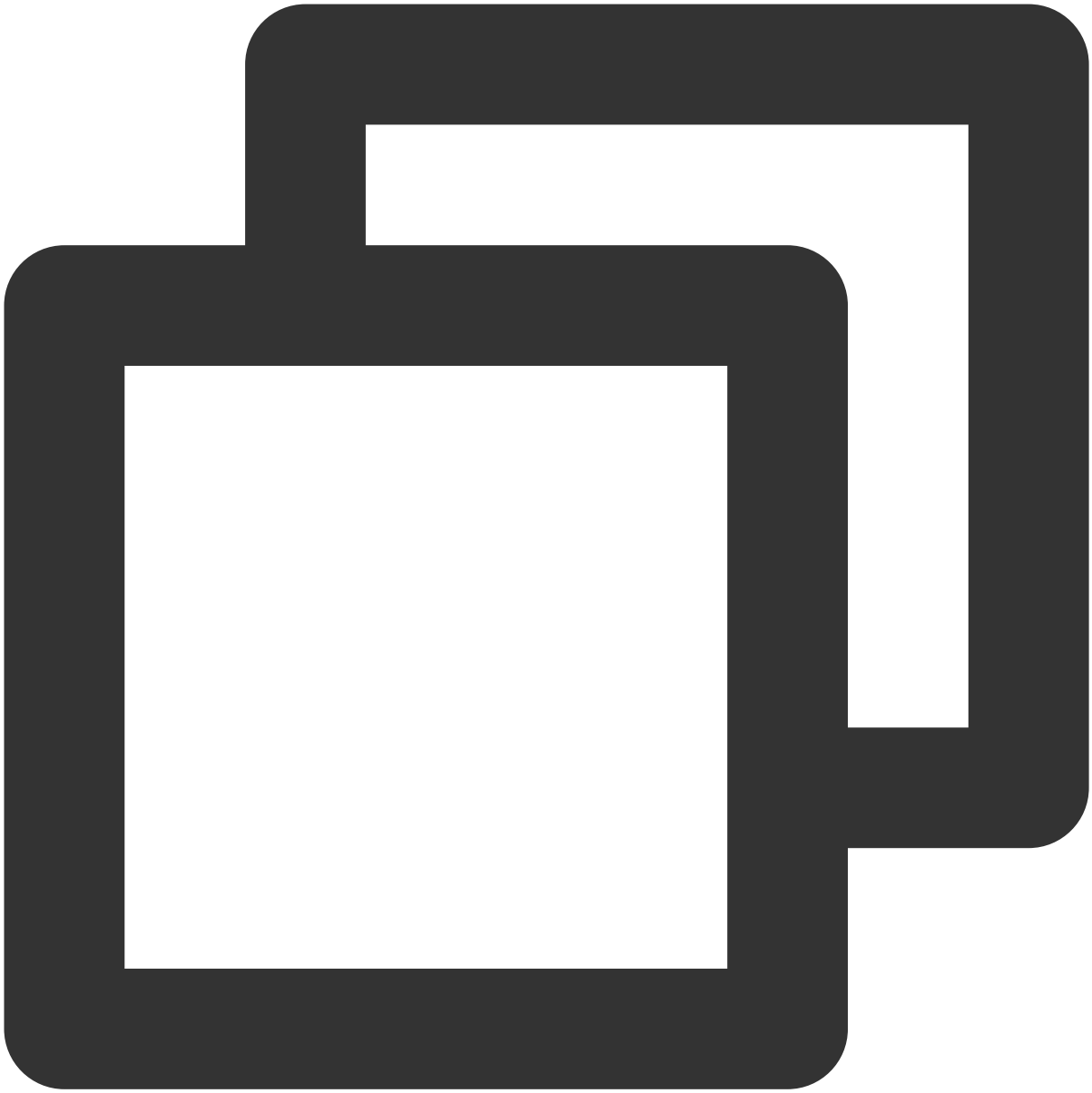


2. RemoteInLargeView layout，远端用户在大窗：



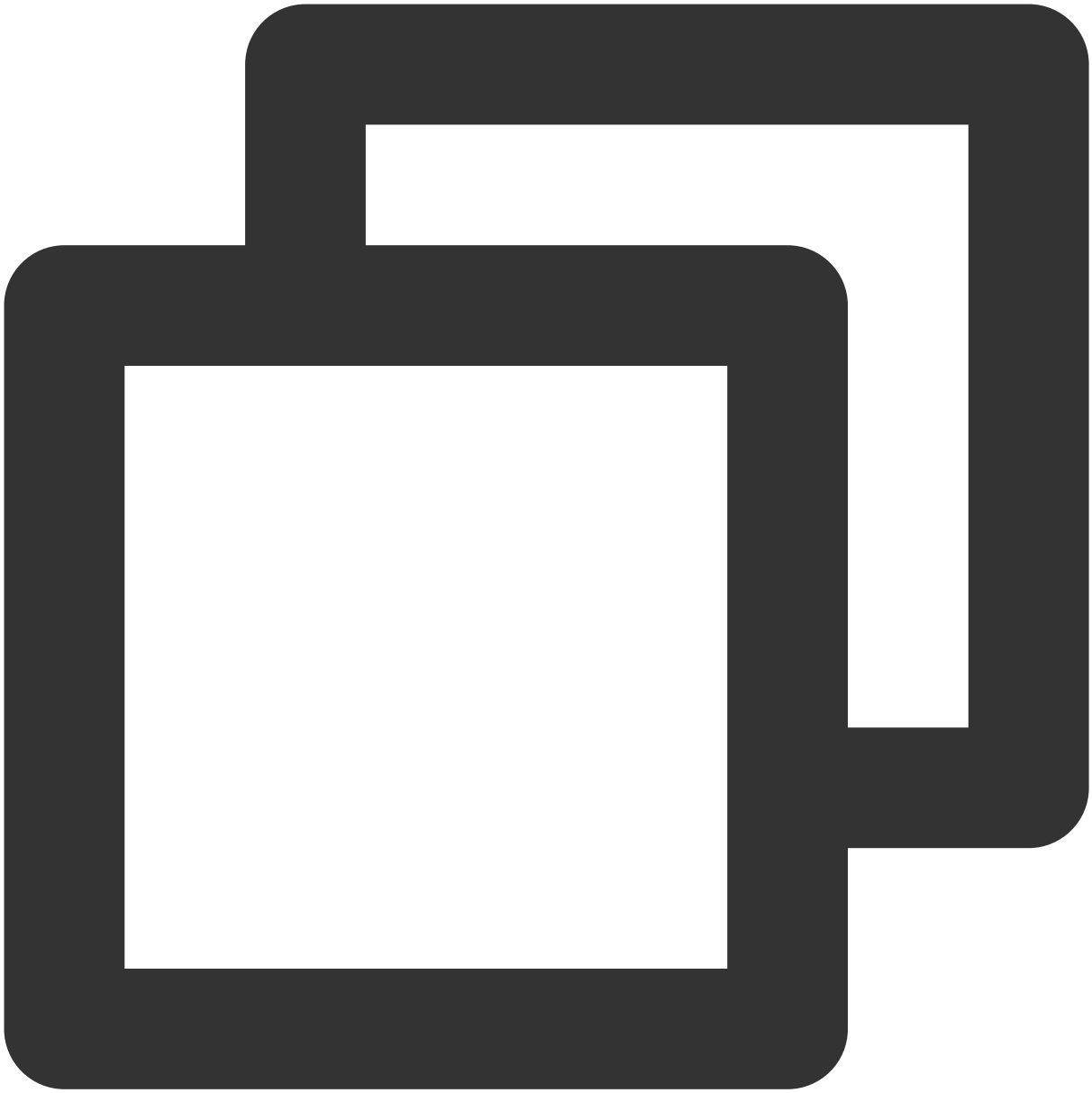
Vue3

React



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

设置摄像头初始状态

注意：

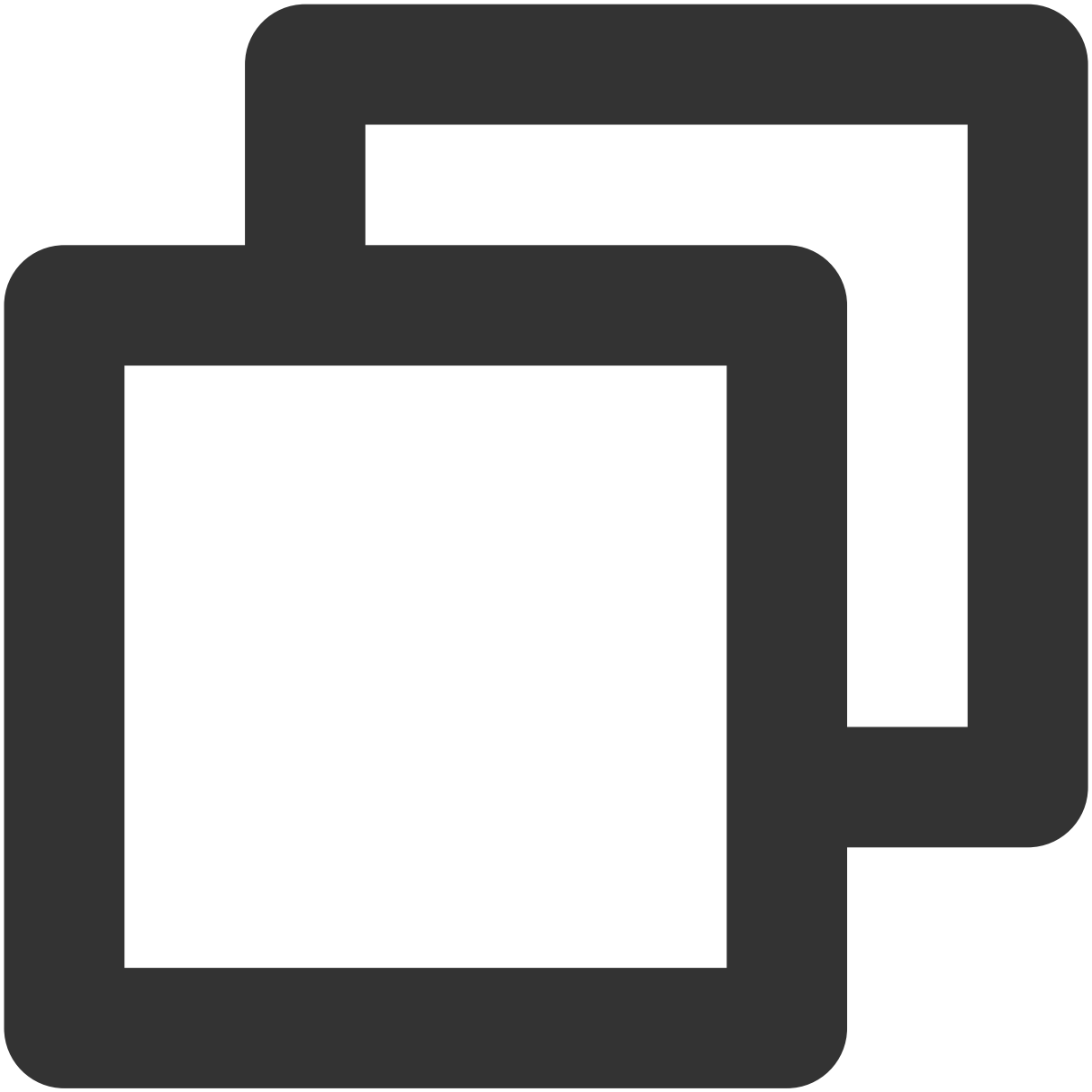
v3.3.0+ 支持。

调用 setCameraDefaultState 设置摄像头按钮初始状态，目前支持开启和关闭。

以默认关闭摄像头为例：

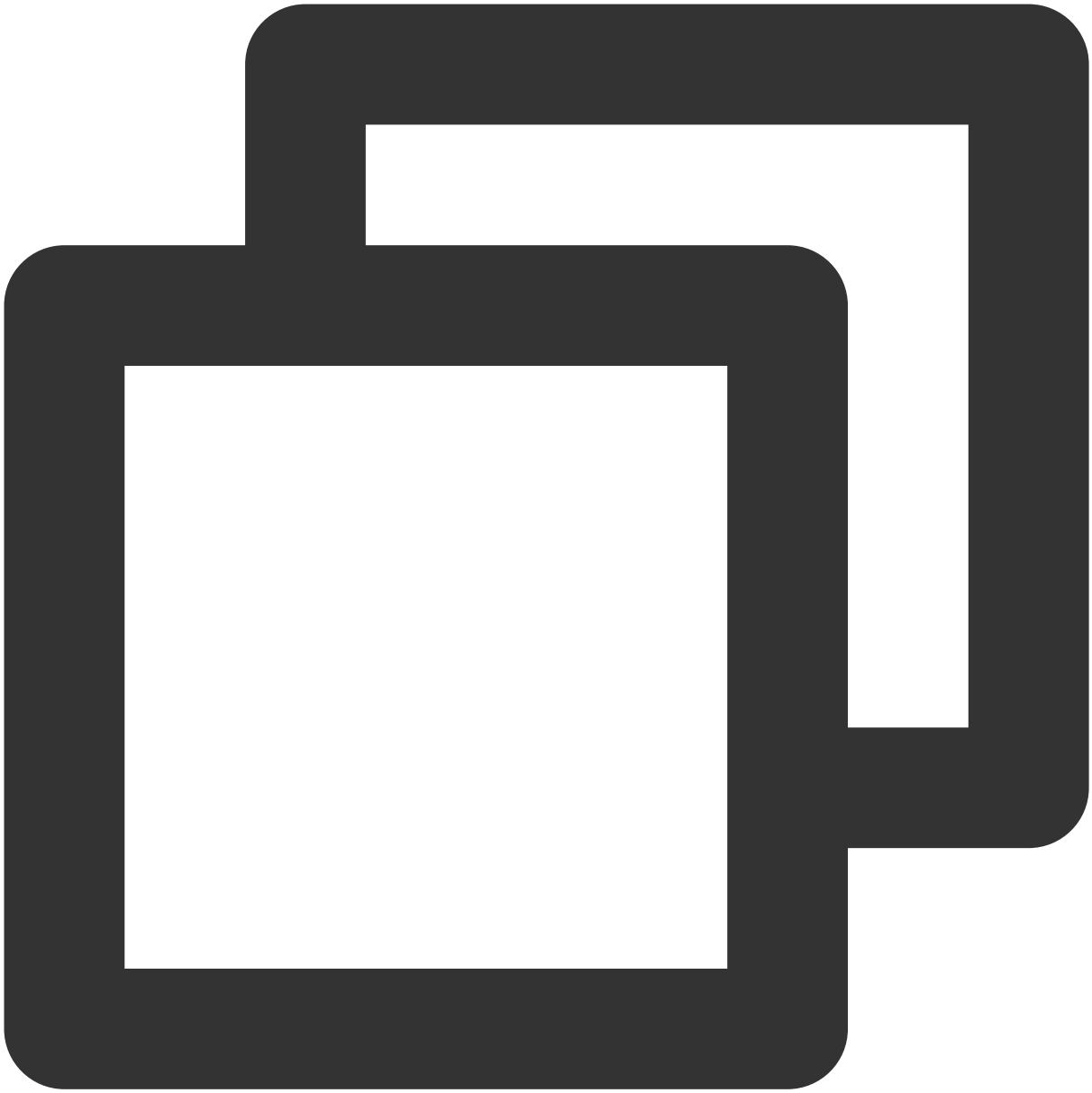
Vue3

React



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue";

TUICallKitServer.setCameraDefaultState(false);
```



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";

TUICallKitServer.setCameraDefaultState(false);
```

替换图标

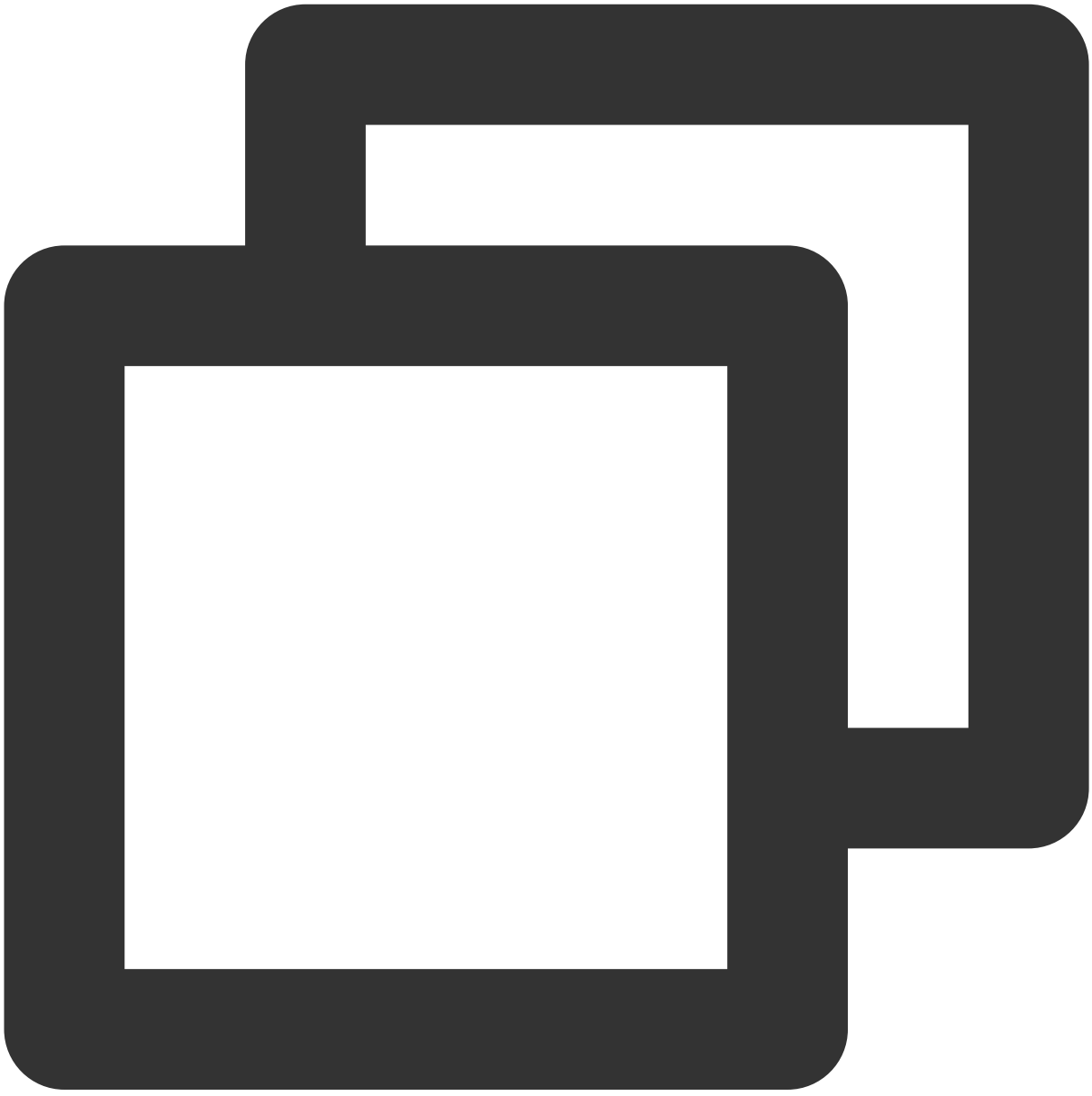
替换图标需要先源码引入，将组件拷贝到您的项目中（源码为 TypeScript 版本）。

注意：

替换图标适用于 `Vue + Typescript` 项目且 `Callkit` 版本号 ≥ 3.2.2，若您采用其他语言或者技术栈，请使用自实现 UI 方案。

1. **下载源码**

Vue3

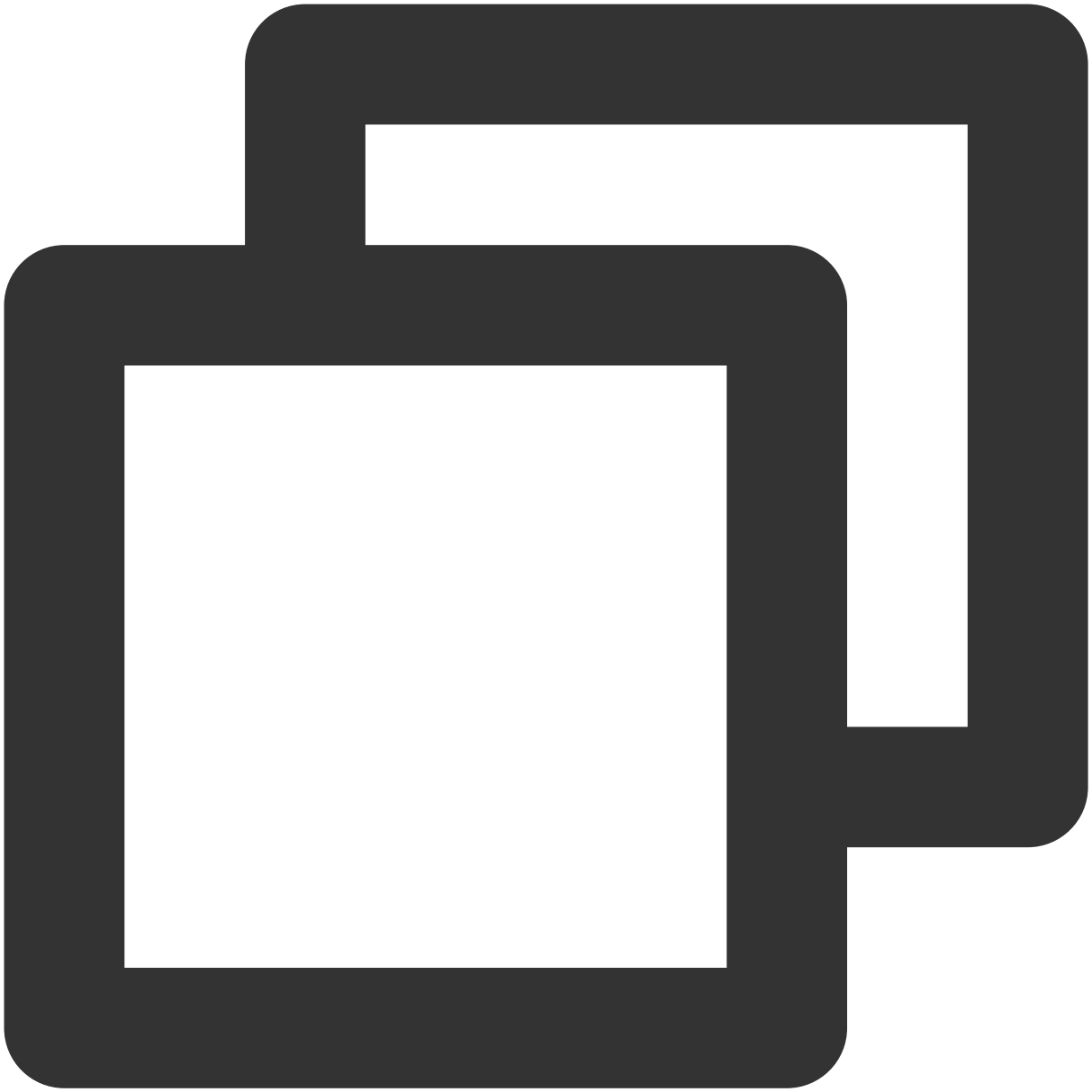


```
npm install @tencentcloud/call-uikit-vue
```

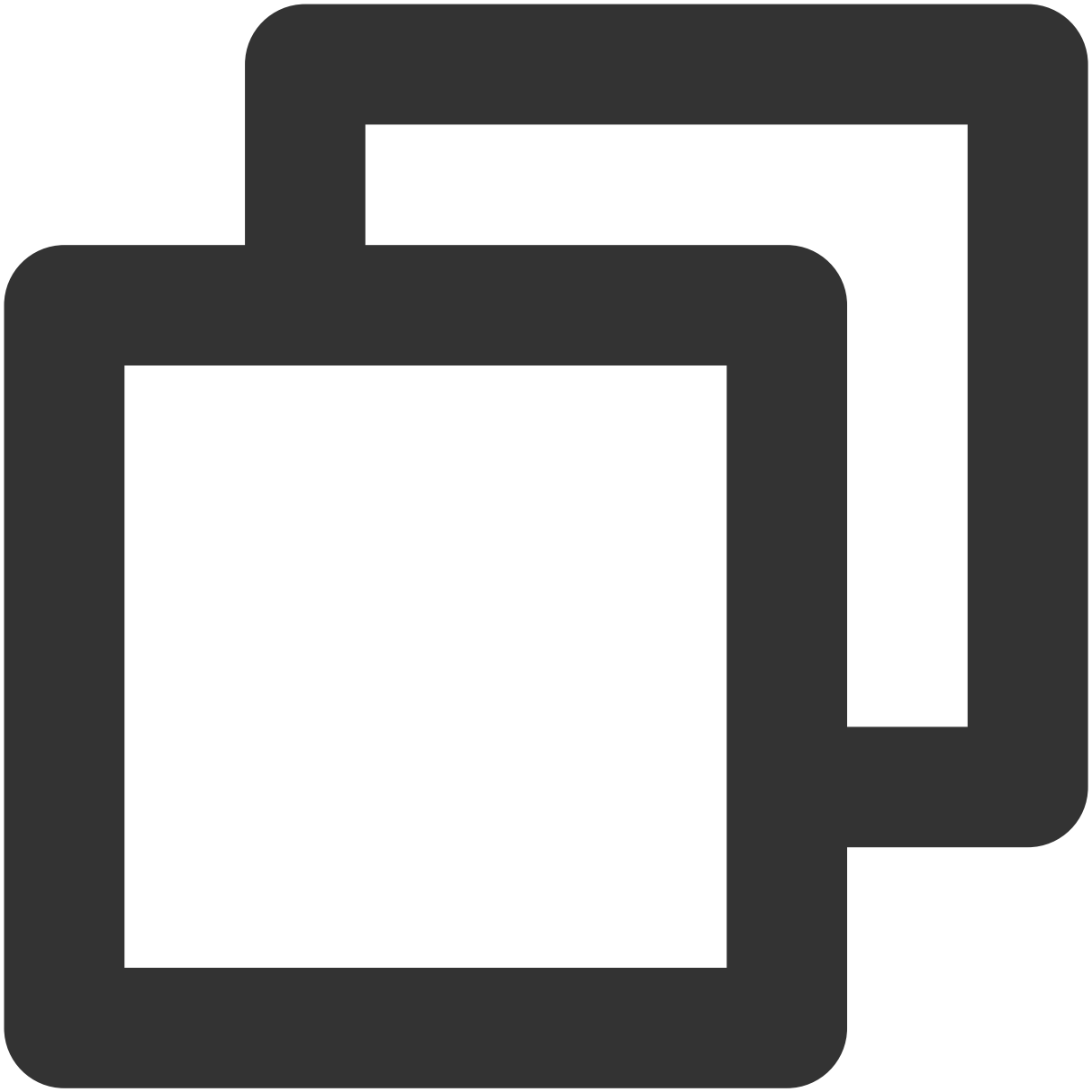
2. **将源码拷贝到自己的项目中，以拷贝到 `src/components/` 目录为例：**

macOS + Vue3

Windows + Vue3



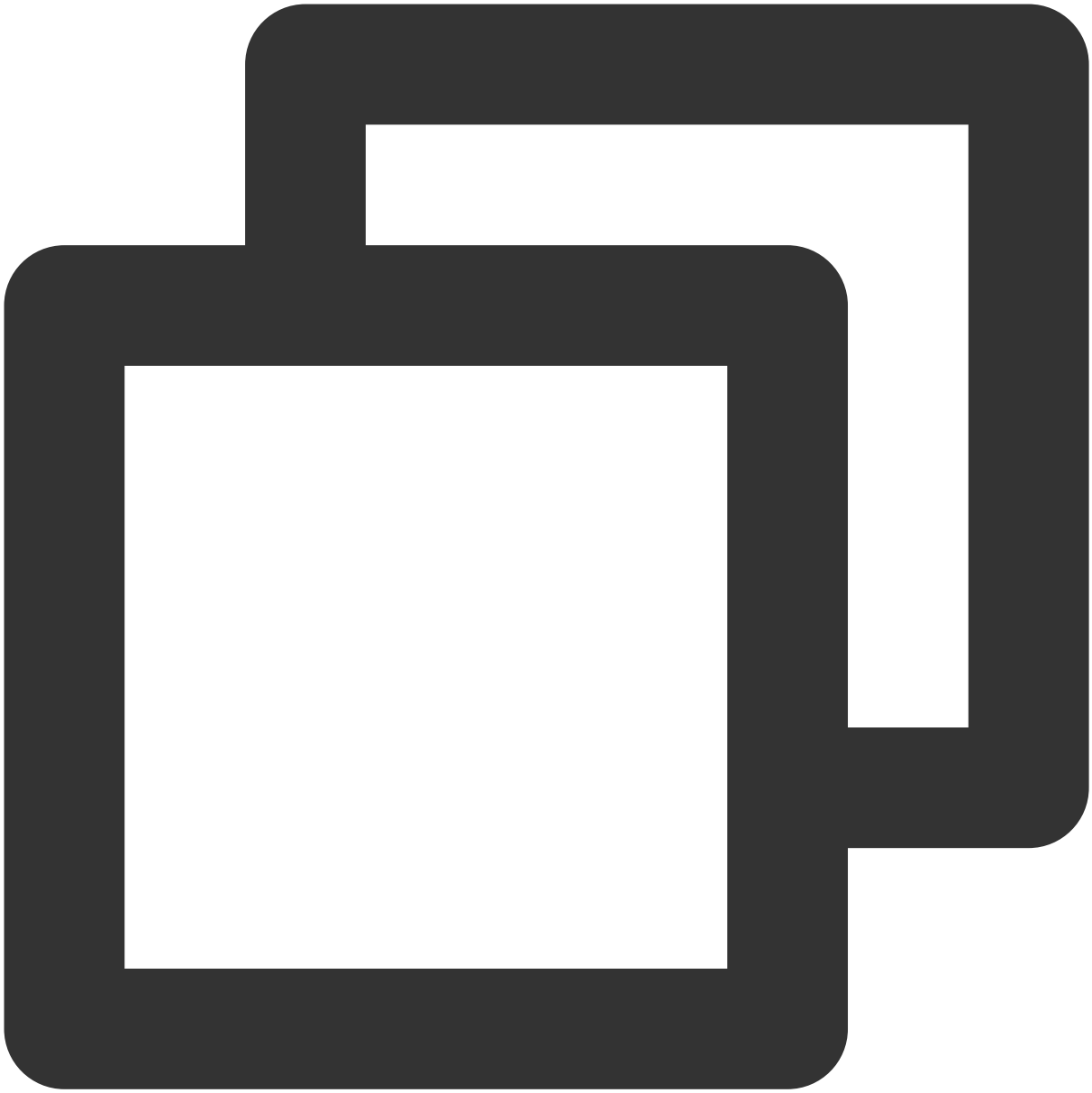
```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uik
```



```
xcopy .\\node_modules\\@tencentcloud\\call-uikit-vue  .\\src\\components\\TUICallKi
```

3. 修改引入路径

需要将 CallKit 改为从本地文件中引入，如下方代码。其他用法细节可参考见 TUICallKit 快速接入。



```
import { TUICallKit, TUICallKitServer, TUICallType } from "./components/TUICallKit/
```

4.

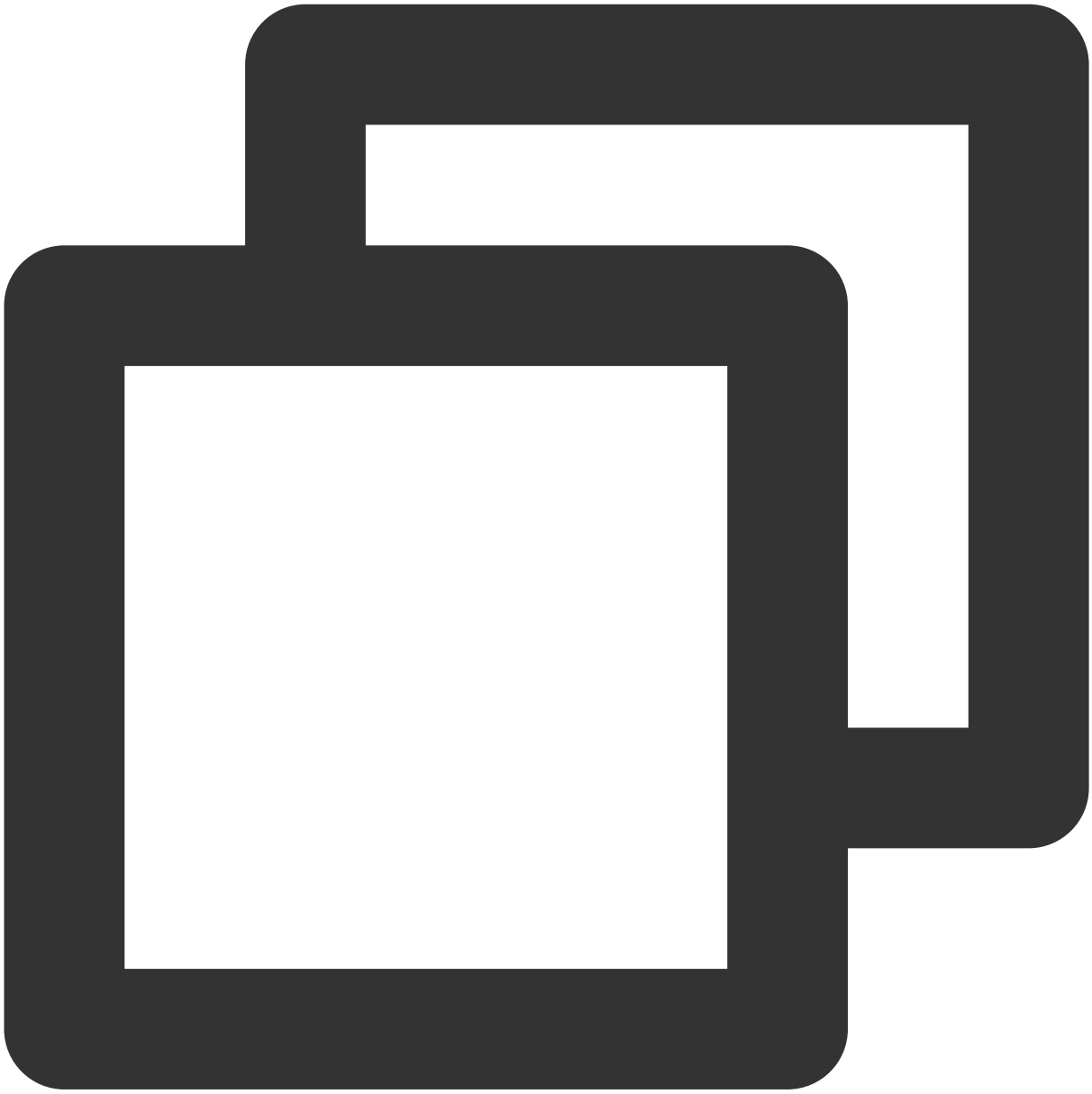
解决源码拷贝可能导致的报错

如果您在使 TUICallKit 组件时遇到了报错，请不要担心，大多数情况下这是由于 ESLint 和 TSConfig 配置不一致造成的。您可以查阅文档，按照要求正确配置即可。如果您需要帮助，请随时联系我们，我们将确保您能够成功地使用此组件。以下是几个常见的问题：

ESLint 报错

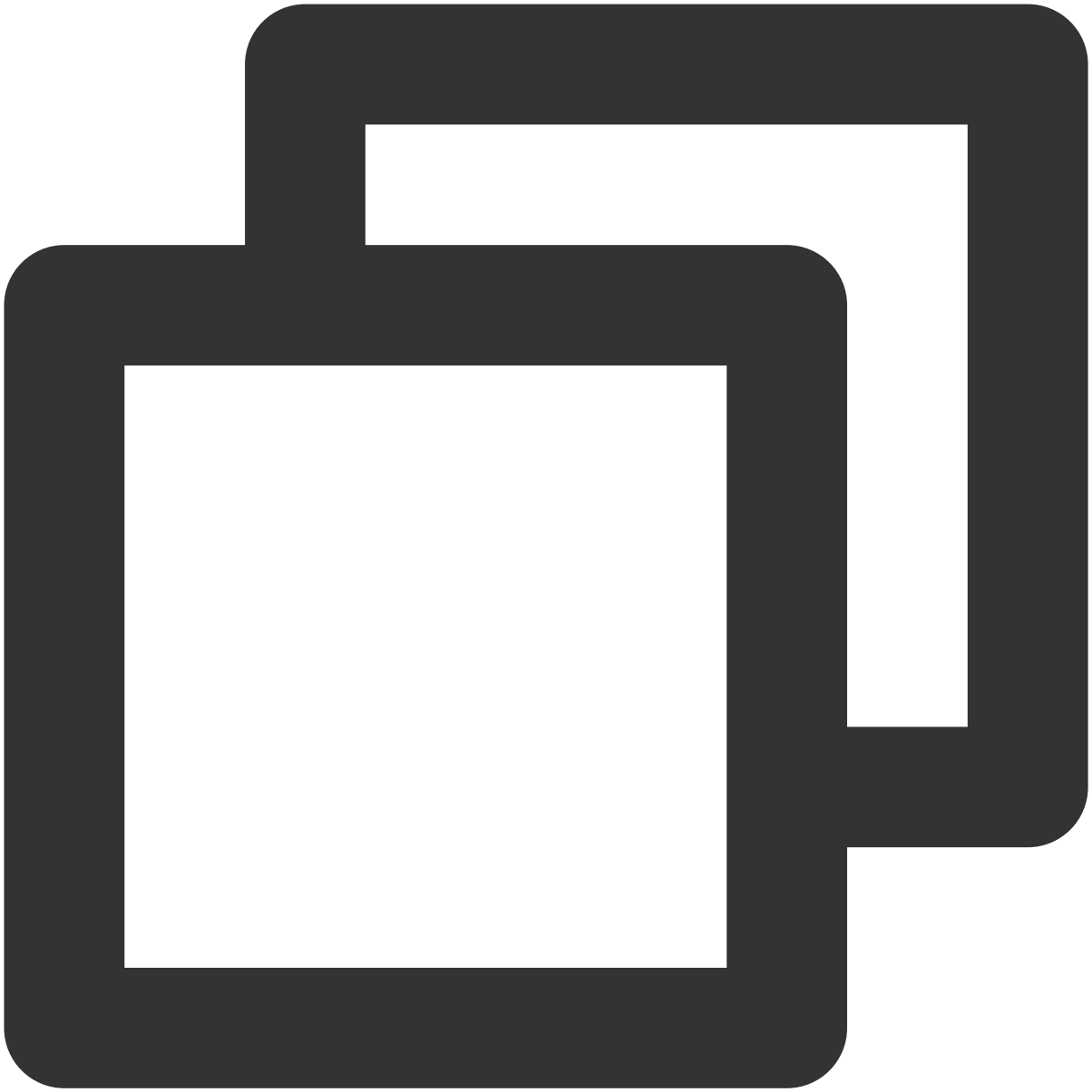
TypeScript 报错

若 TUICallKit 与您项目的代码风格不一致导致报错，可将本组件目录屏蔽，如在项目根目录增加 `.eslintignore` 文件，如：



```
# .eslintignore
src/components/TUICallKit
```

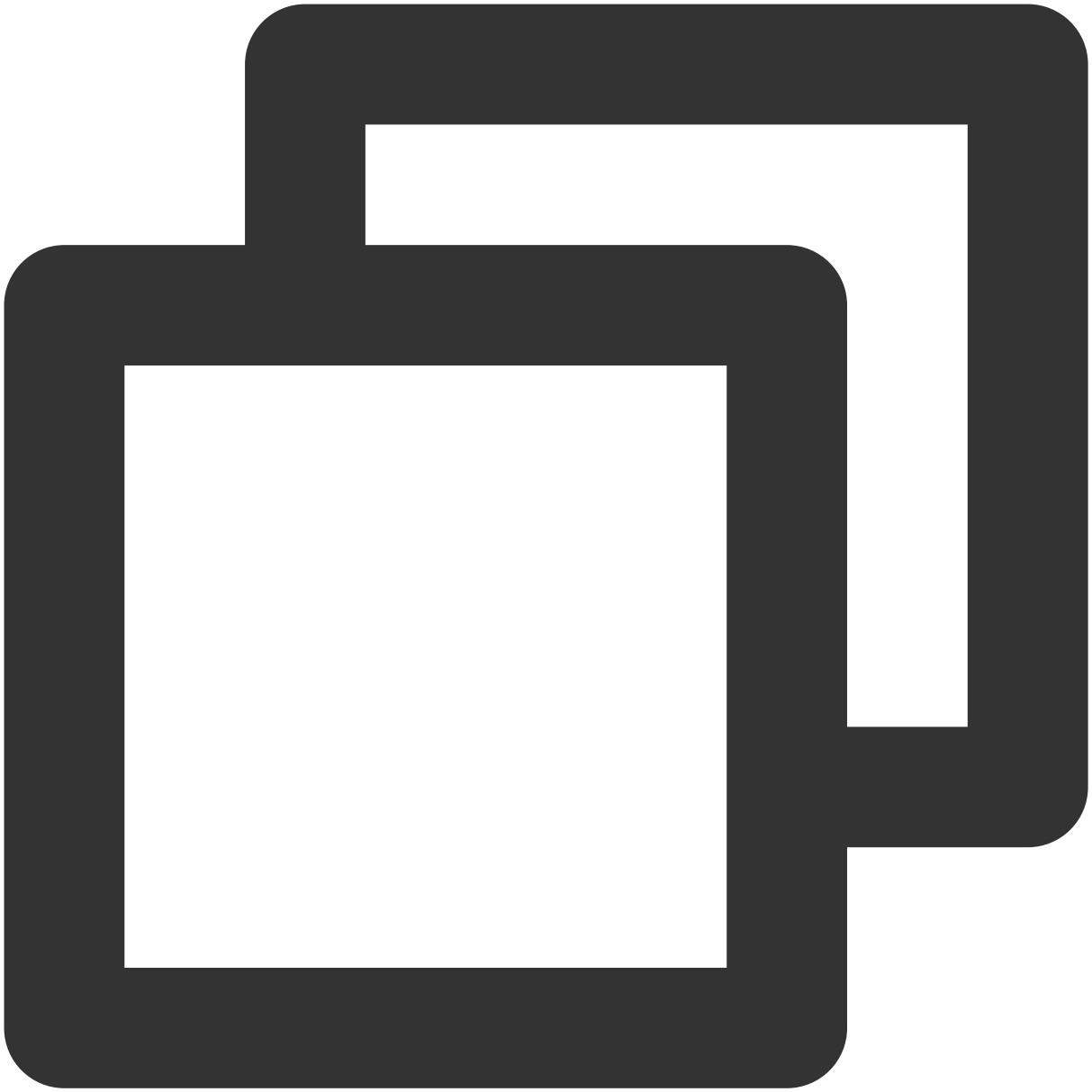
1. 如遇 Cannot find module '../package.json' 报错，是因为 TUICallKit 内引用了 JSON 文件，可在 tsconfig.json 中添加相关配置，示例：



```
{
  "compilerOptions": {
    "resolveJsonModule": true
  }
}
```

其他 TSConfig 问题请参见 TSConfig Reference。

2. 如遇 Uncaught SyntaxError: Invalid or unexpected token 报错，是因为 TUICallKit 使用了装饰器，可在 tsconfig.json 中添加相关配置，示例：



```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

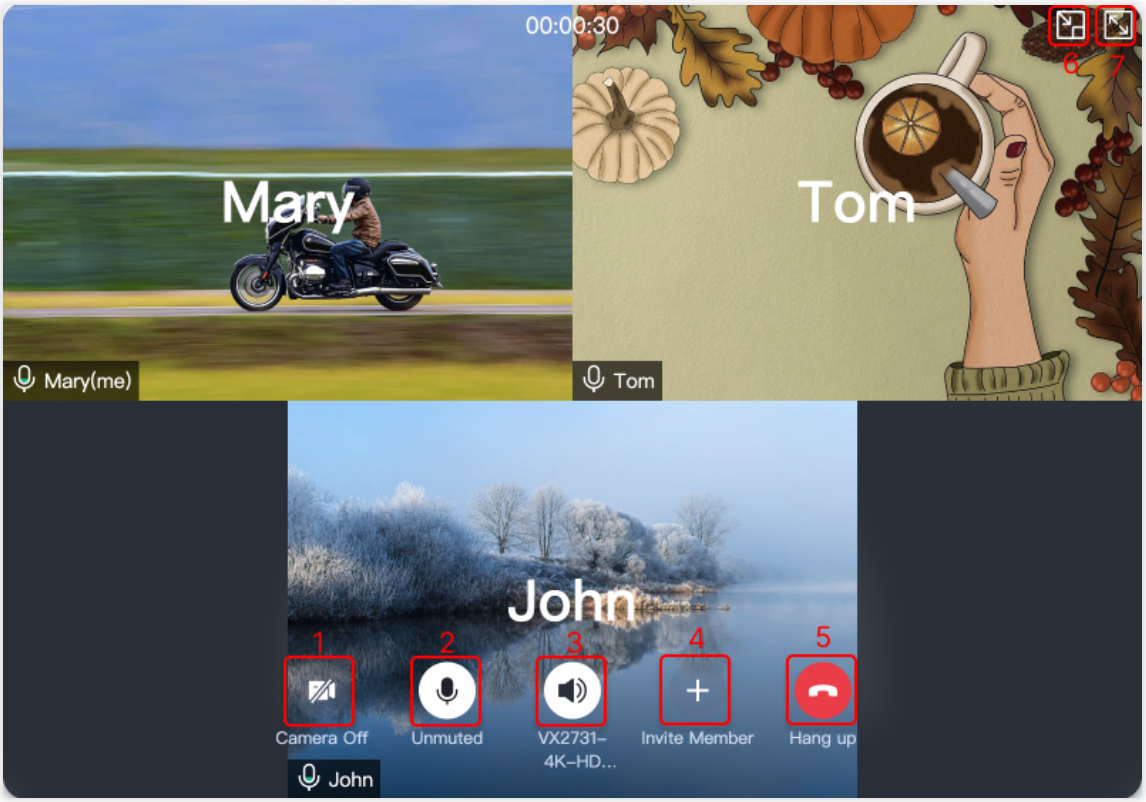
5. 修改 TUICallKit/Components/assets 文件夹下的图标组件

注意：

为了确保整个应用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。

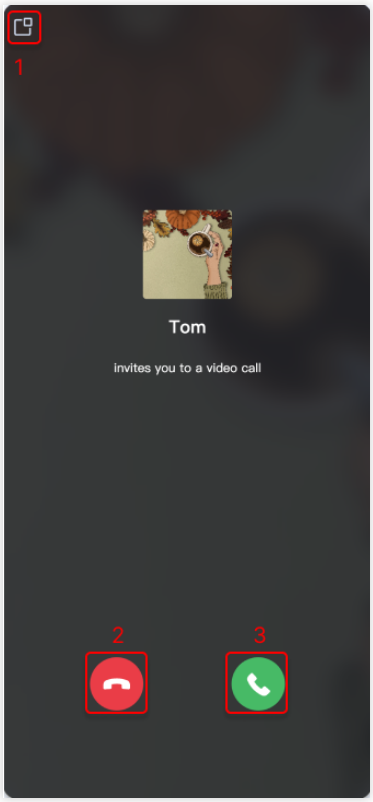
桌面端

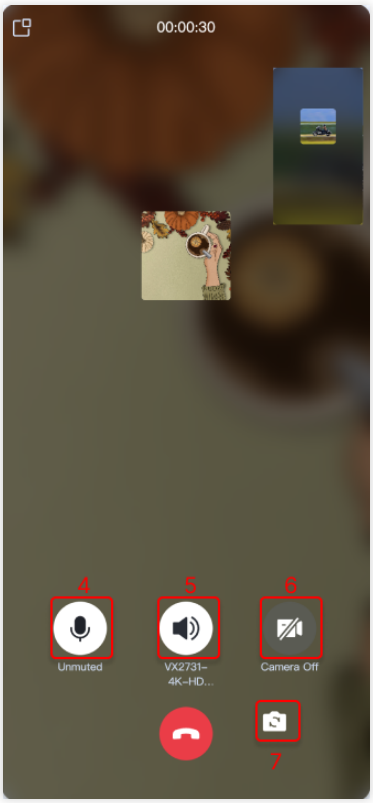
移动端



序号	资源路径
1	/TUICallKit/Components/assets/button/camera-close.svg
2	/TUICallKit/Components/assets/button/microphone-open.svg
3	/TUICallKit/Components/assets/button/speaker-open.svg
4	/TUICallKit/Components/assets/button/desktop/inviteUser.svg
5	/TUICallKit/Components/assets/button/hangup.svg
6	/TUICallKit/Components/assets/button/desktop/minimize.svg
7	/TUICallKit/Components/assets/button/desktop/fullScreen.svg

版权所有：腾讯云计算（北京）有限责任公司


1
Tom
invites you to a video call
2
3


00:00:30
4
Unmuted
5
VX2731-
4K-HD...
6
Camera Off
7

序号	资源路径
1	/TUICallKit/Components/assets/button/mobile/minimize.svg
2	/TUICallKit/Components/assets/button/hangup.svg
3	/TUICallKit/Components/assets/button/accept.svg
4	/TUICallKit/Components/assets/button/microphone-open.svg
5	/TUICallKit/Components/assets/button/speaker-open.svg
6	/TUICallKit/Components/assets/button/camera-close.svg
7	/TUICallKit/Components/assets/button/switchCamera.svg

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

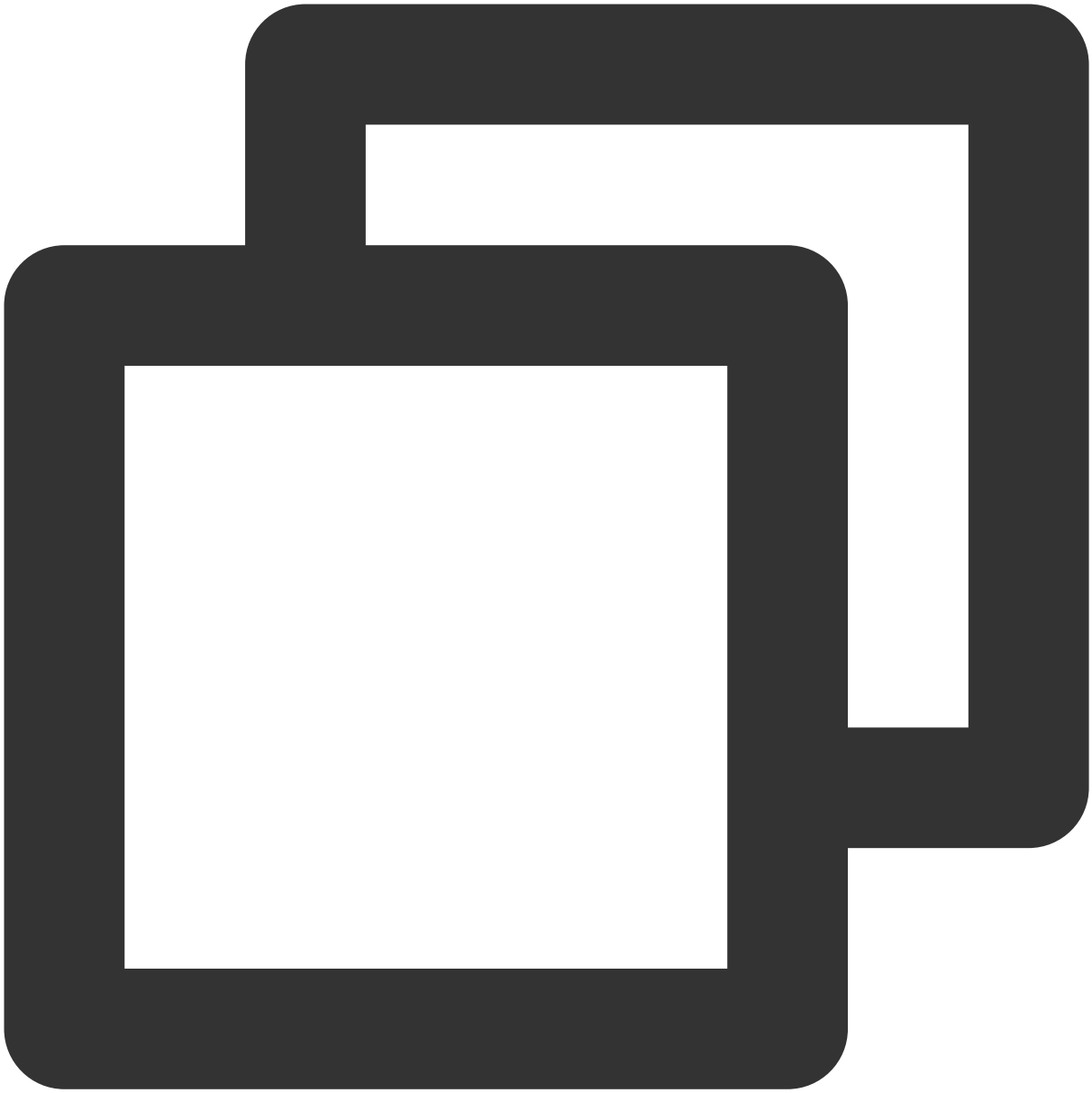
最近更新时间：2024-03-12 14:51:07

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：

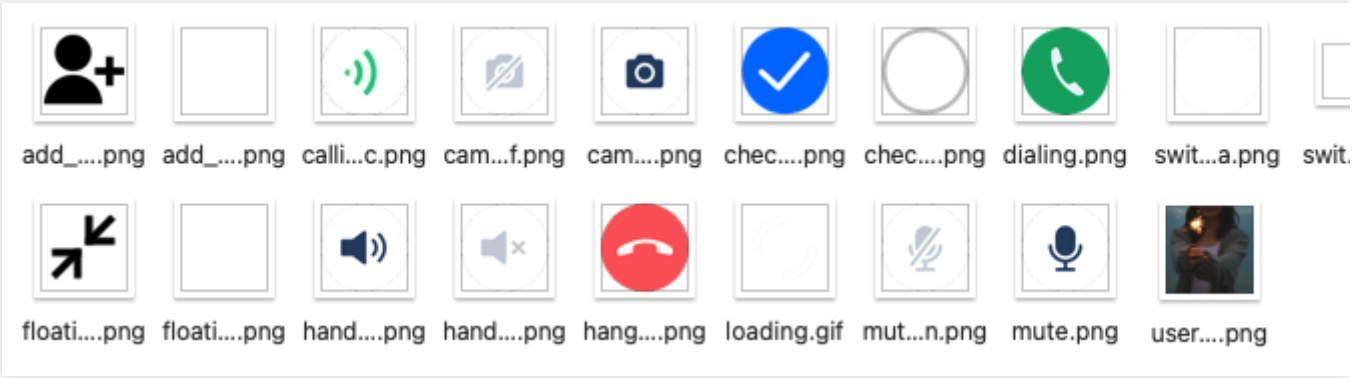
版权所有：腾讯云计算（北京）有限责任公司



```
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：2024-07-19 18:07:32

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。

由于 TIMPush 插件目前尚不支持 iOS 的 VoIP 通知，我们为您提供了一个免费的替代方案以帮助您在 iOS 设备上实现 VoIP 推送。

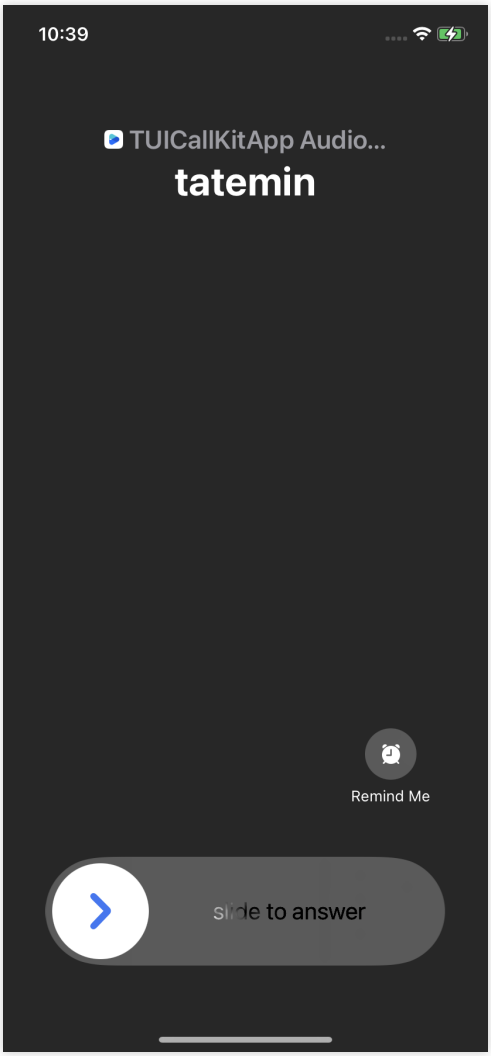
说明：

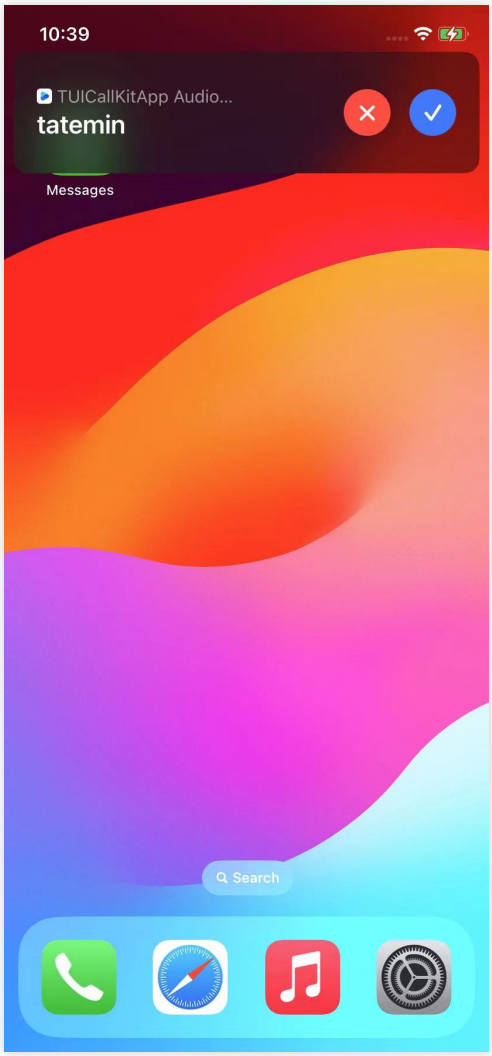
Apple 要求，VoIP Push 在 iOS 13.0 以后需要搭配 CallKit.framework 使用，否则 App 运行后会 crash。

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。

集成效果

锁屏时的效果	应用在后台的效果	





配置 VoIP Push

如果想要接收 VoIP Push，需要遵循如下几个步骤：

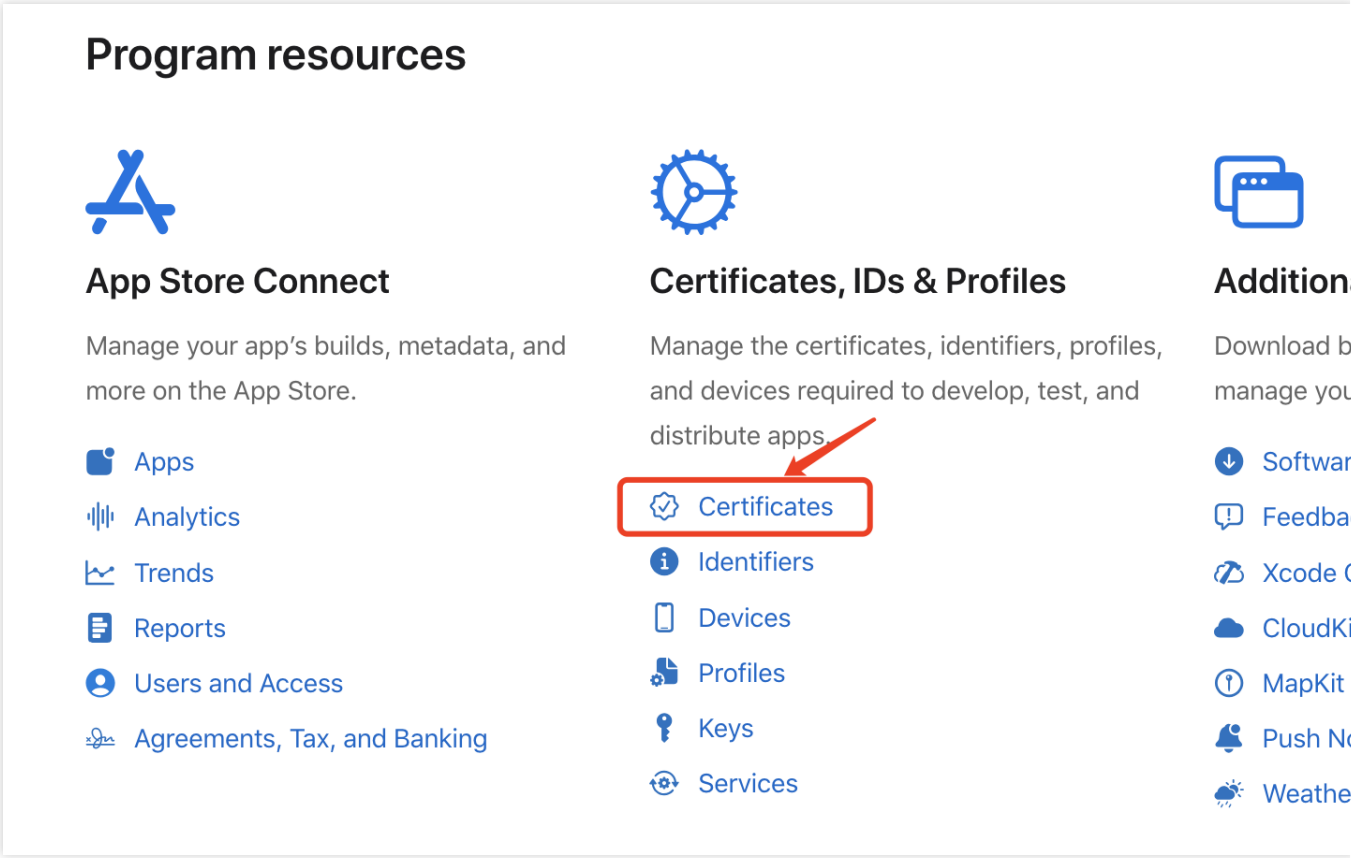
1. 申请 VoIP Push 证书。
2. 上传证书到 Chat 控制台。
3. 完成工程配置。
4. 接入 TUICallKitVoIPExtension 组件。

步骤1：申请 VoIP Push 证书

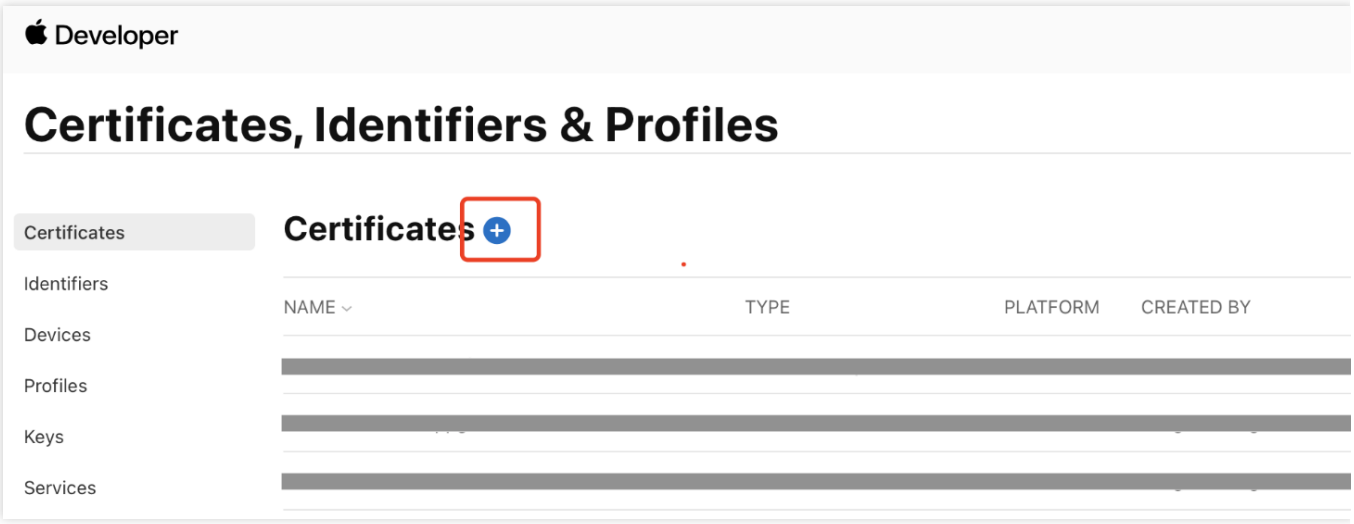
在申请 VoIP Push 证书之前，请先登录 苹果开发者中心 网站，开启 App 的远程推送功能。当您的 AppID 具备了 Push Notification 能力后，按照如下步骤申请并配置 VoIP Push 证书：

1. 登录 苹果开发者中心 网站，单击 Certificates, IDs & Profiles 选项卡中的 Certificates，进入 Certificates, Identifiers & Profiles 页面。

版权所有：腾讯云计算（北京）有限责任公司



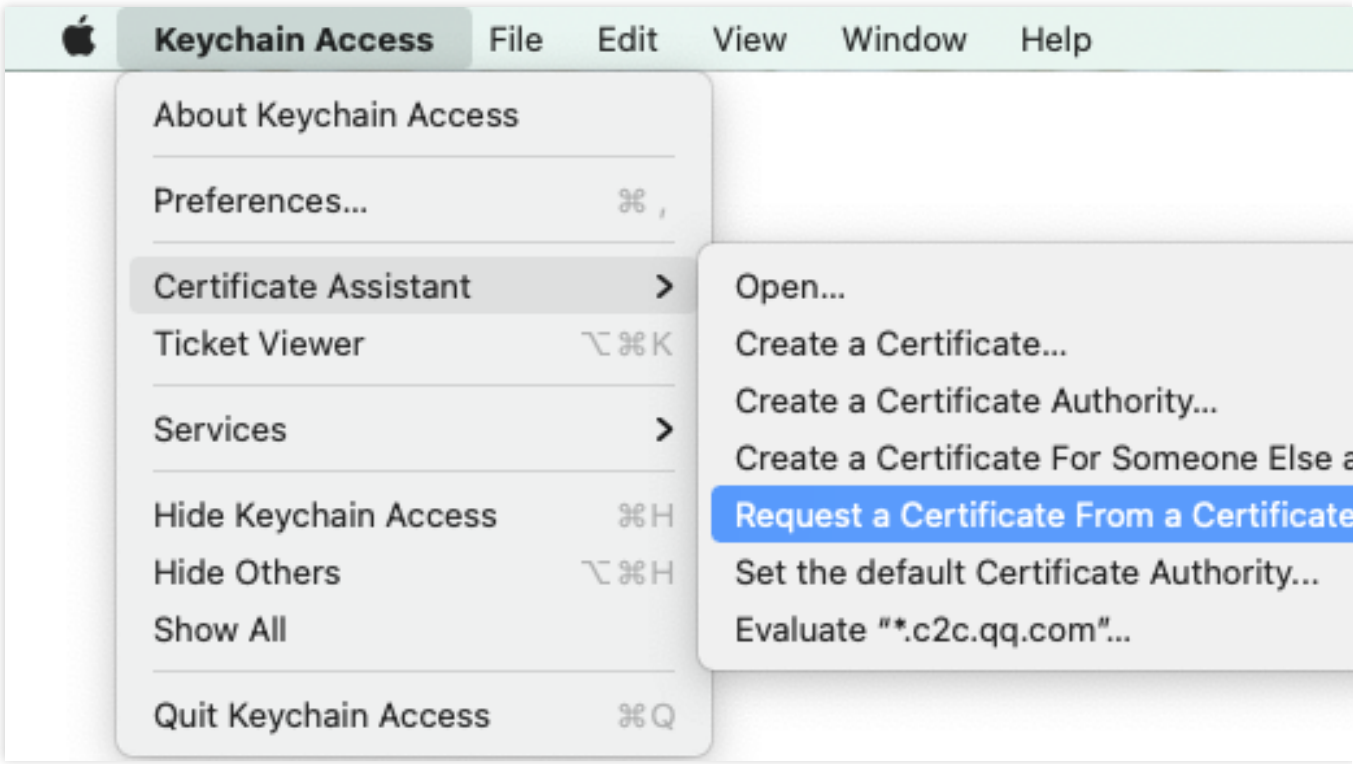
2. 单击 **Certificates** 右侧的 **+**。



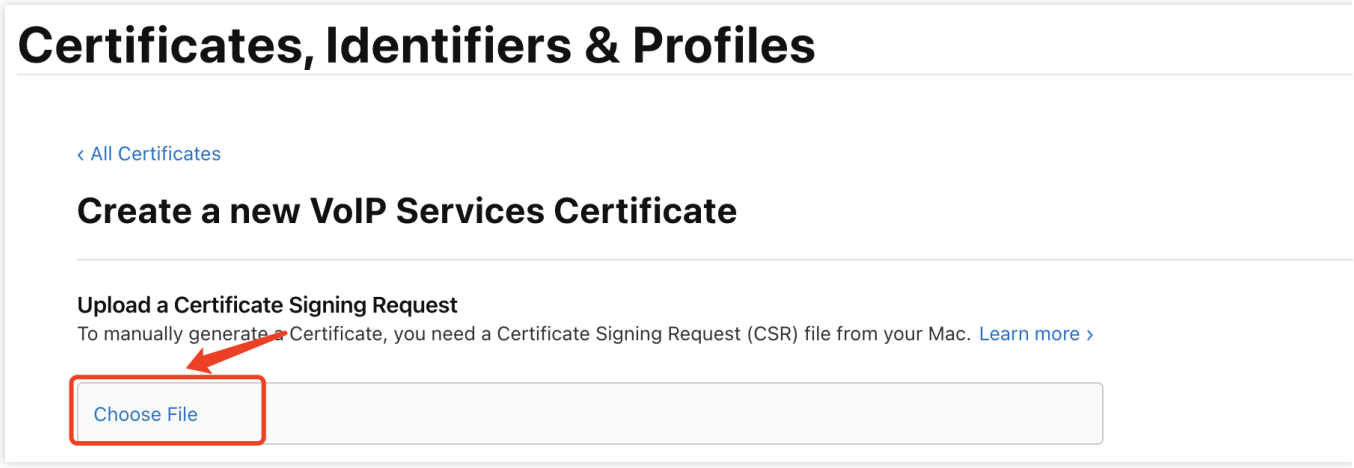
3. 在 **Create a New Certificate** 选项卡中，选择 **VoIP Services Certificate**，并单击 **Continue**。

4. 在 **Select an App ID for your VoIP Service Certificate** 选项卡中，选择您当前的 App 的 BundleID，并单击 **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`）。



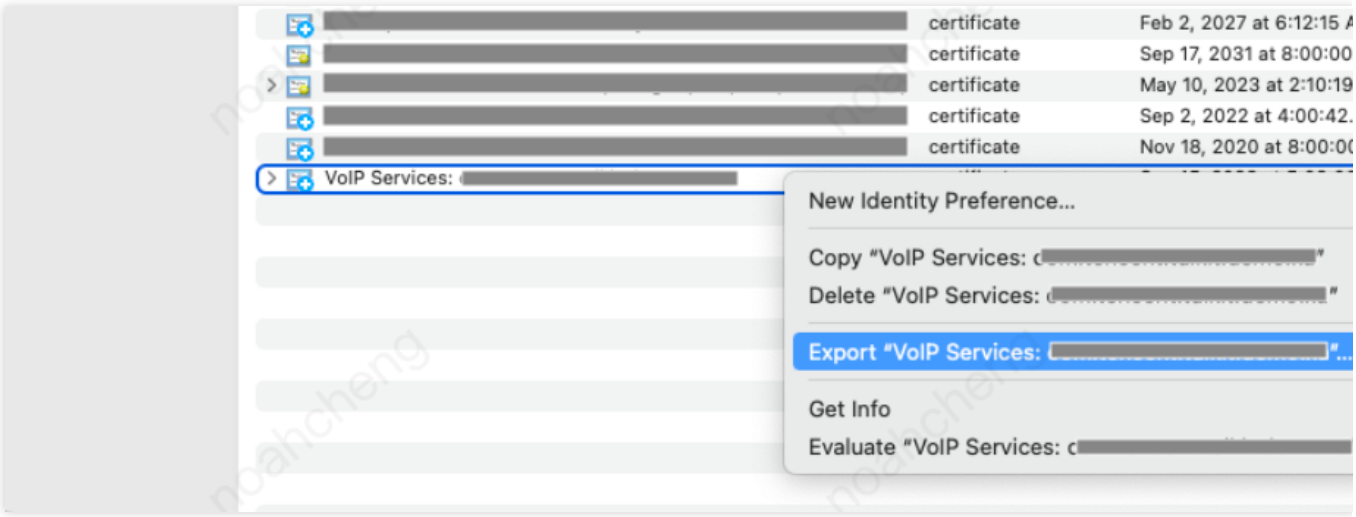
7. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择 **存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。
返回上述 步骤5 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。



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：

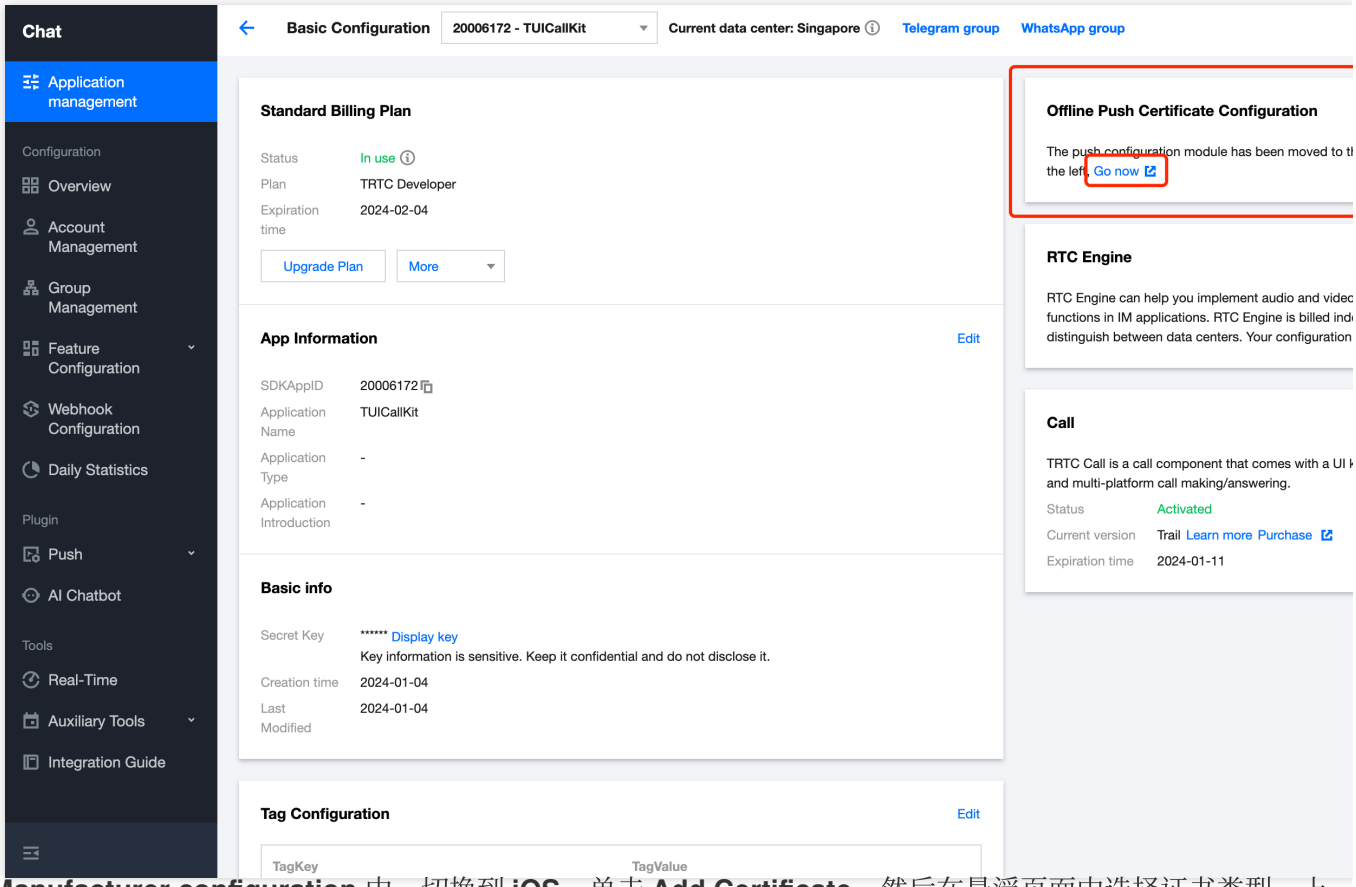
保存 `P12` 文件时，请务必要为其设置密码。



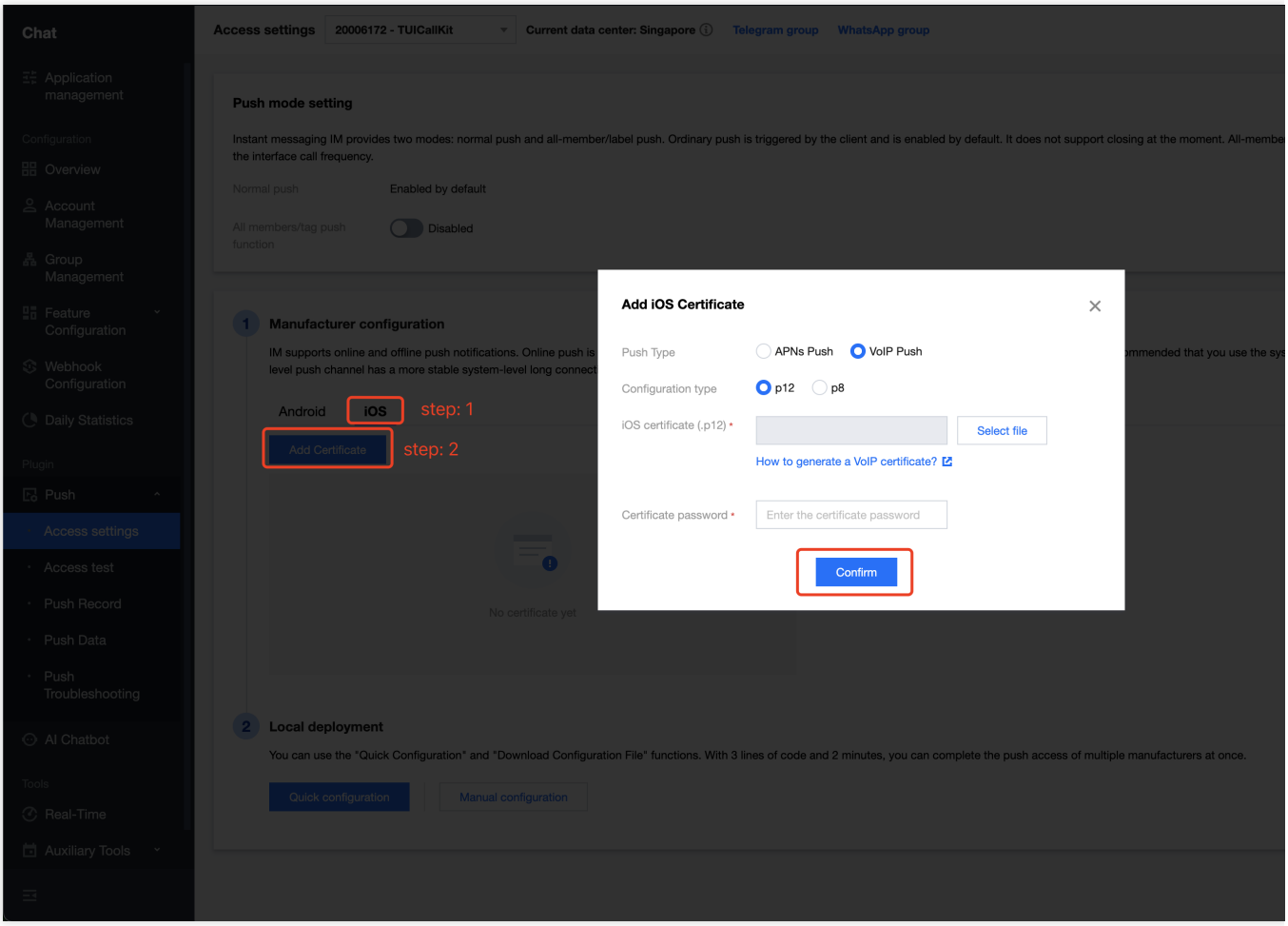
步骤2：上传证书到 Chat 控制台

打开 Chat 控制台，选择您创建的 Chat 应用，并按照如下步骤上传证书：

1. 选择您的 Chat 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**，单击 **Add Certificate**，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。

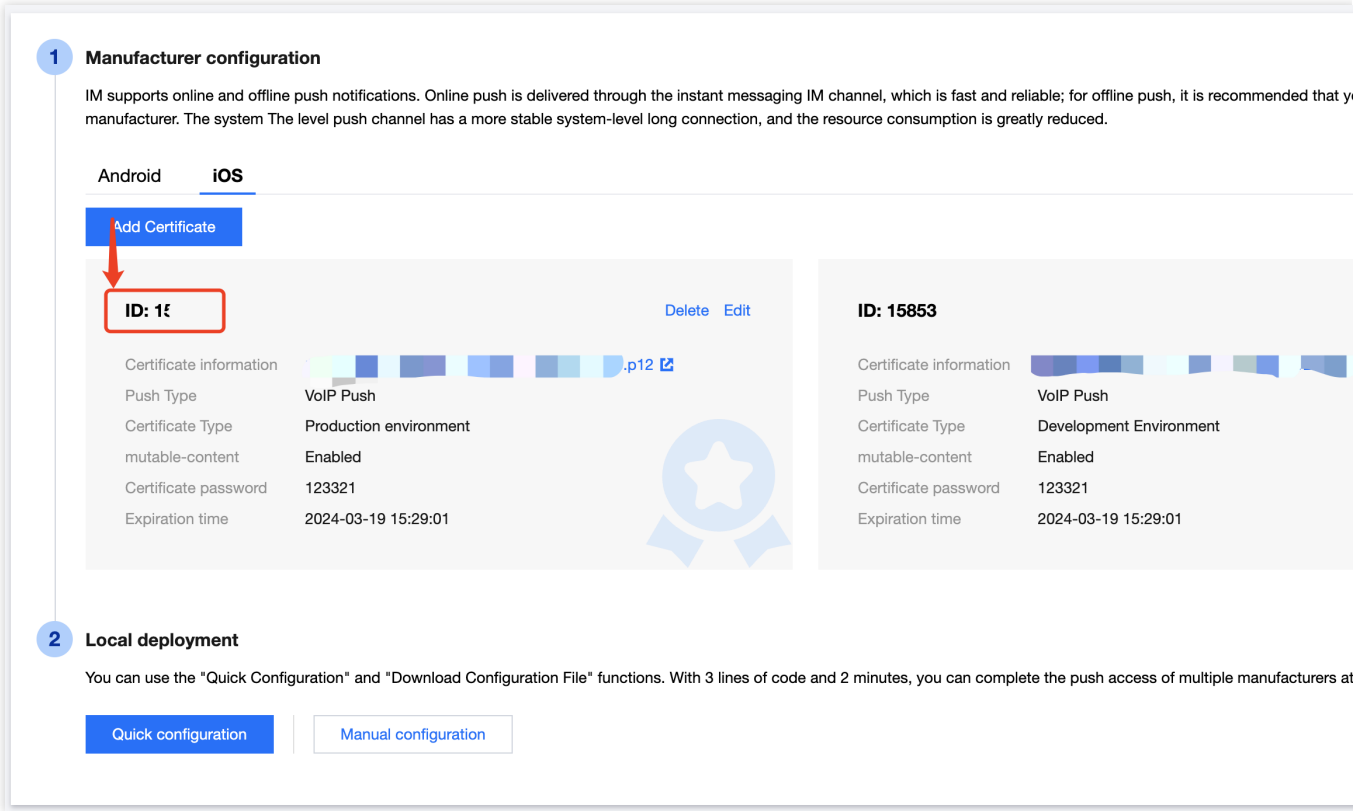
上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

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。

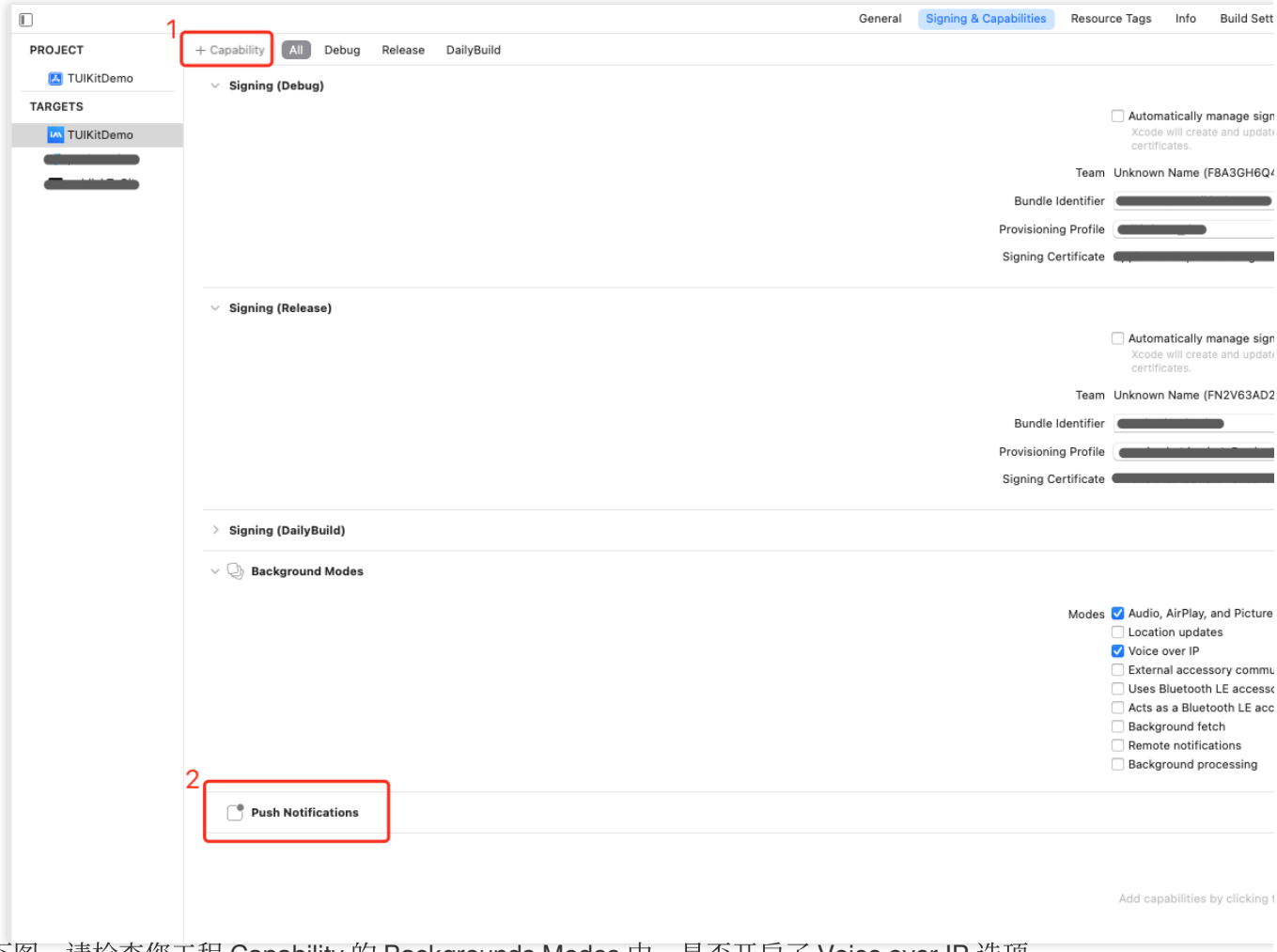


说明：

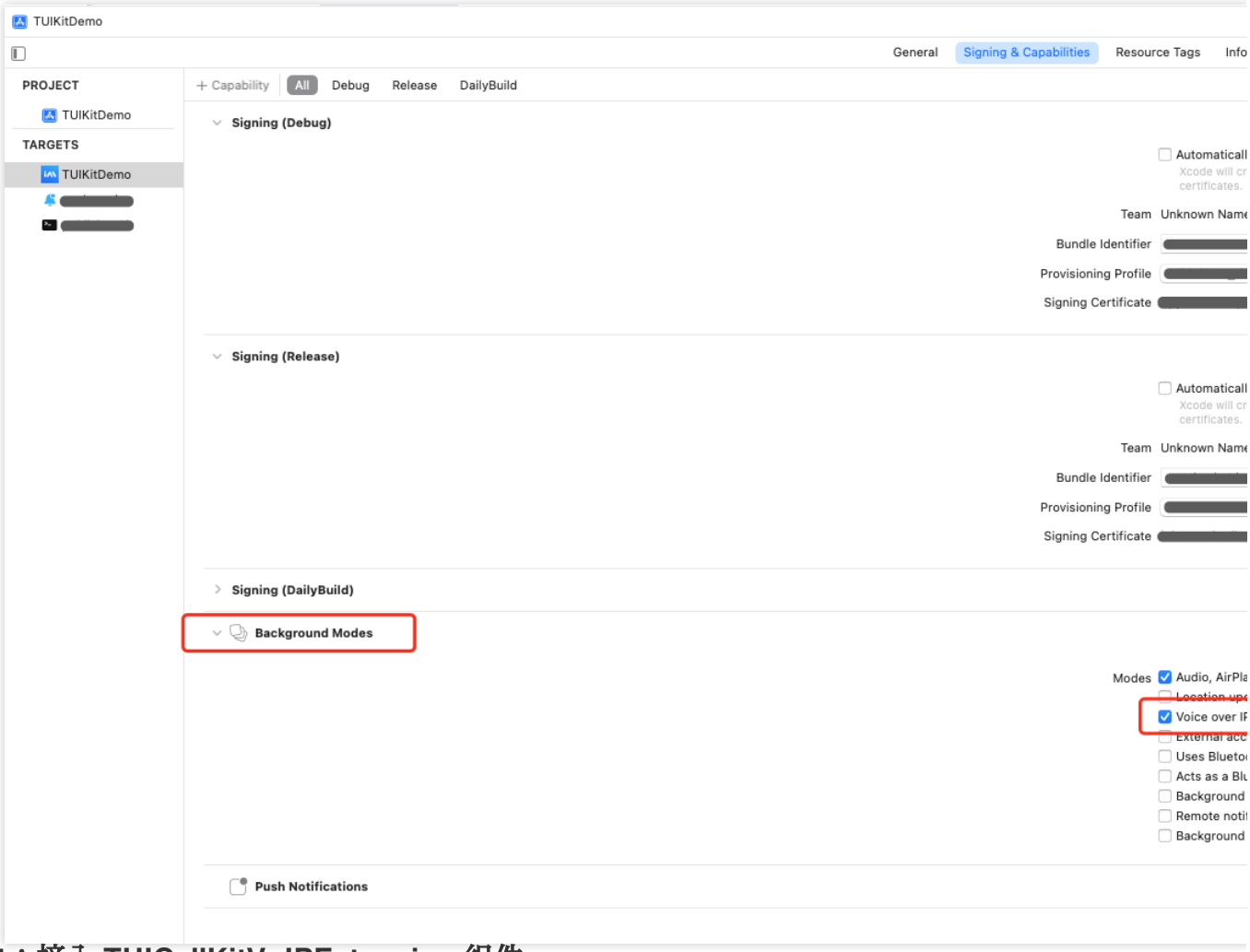
开发环境和生产环境下的证书 ID 要严格区分，并在 步骤4：接入TUICallKitVoIPExtension 组件 时根据实际环境填写。

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。



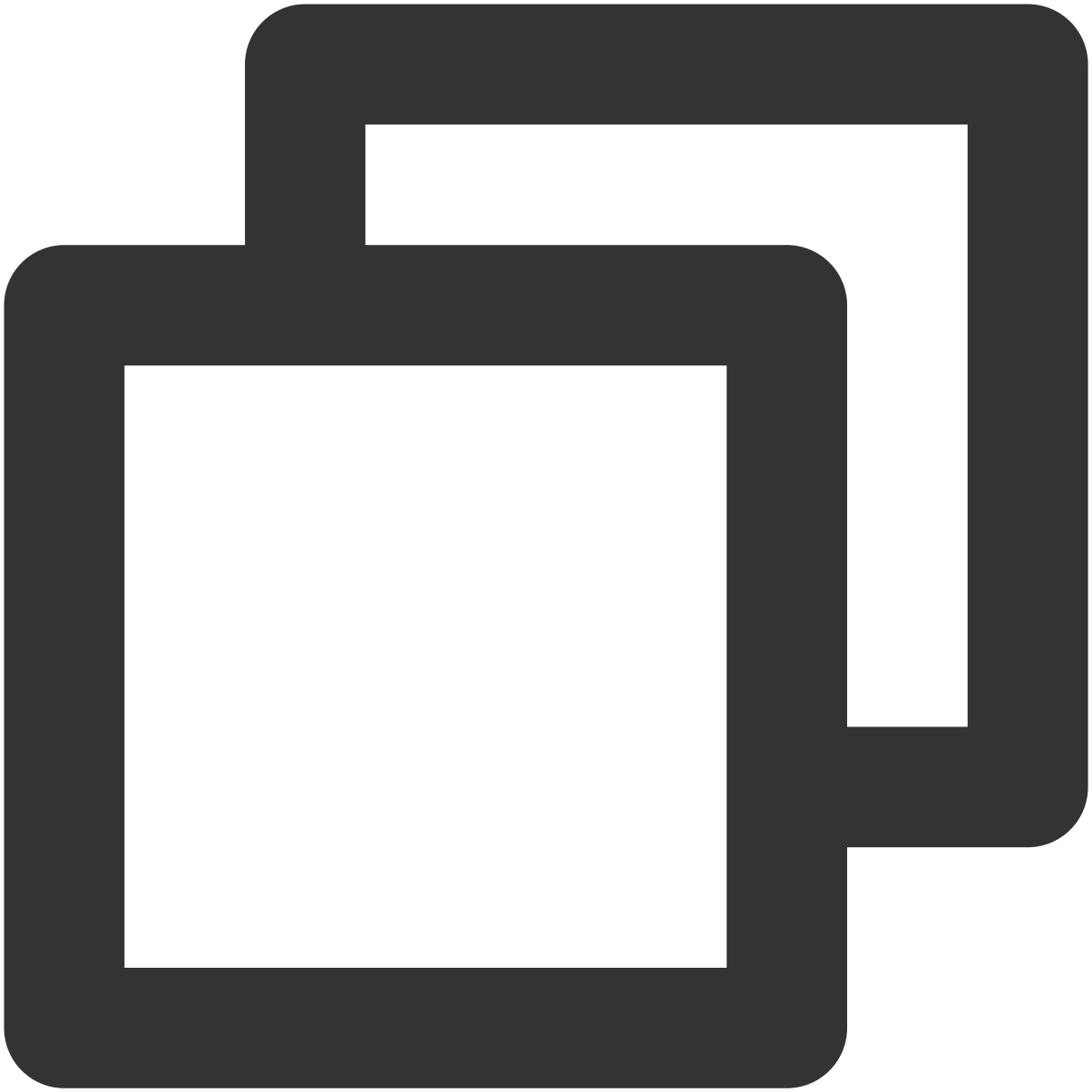
2. 如下图，请检查您工程 Capability 的 Backgrounds Modes 中，是否开启了 Voice over IP 选项。



步骤4：接入 TUICallKitVoIPExtension 组件

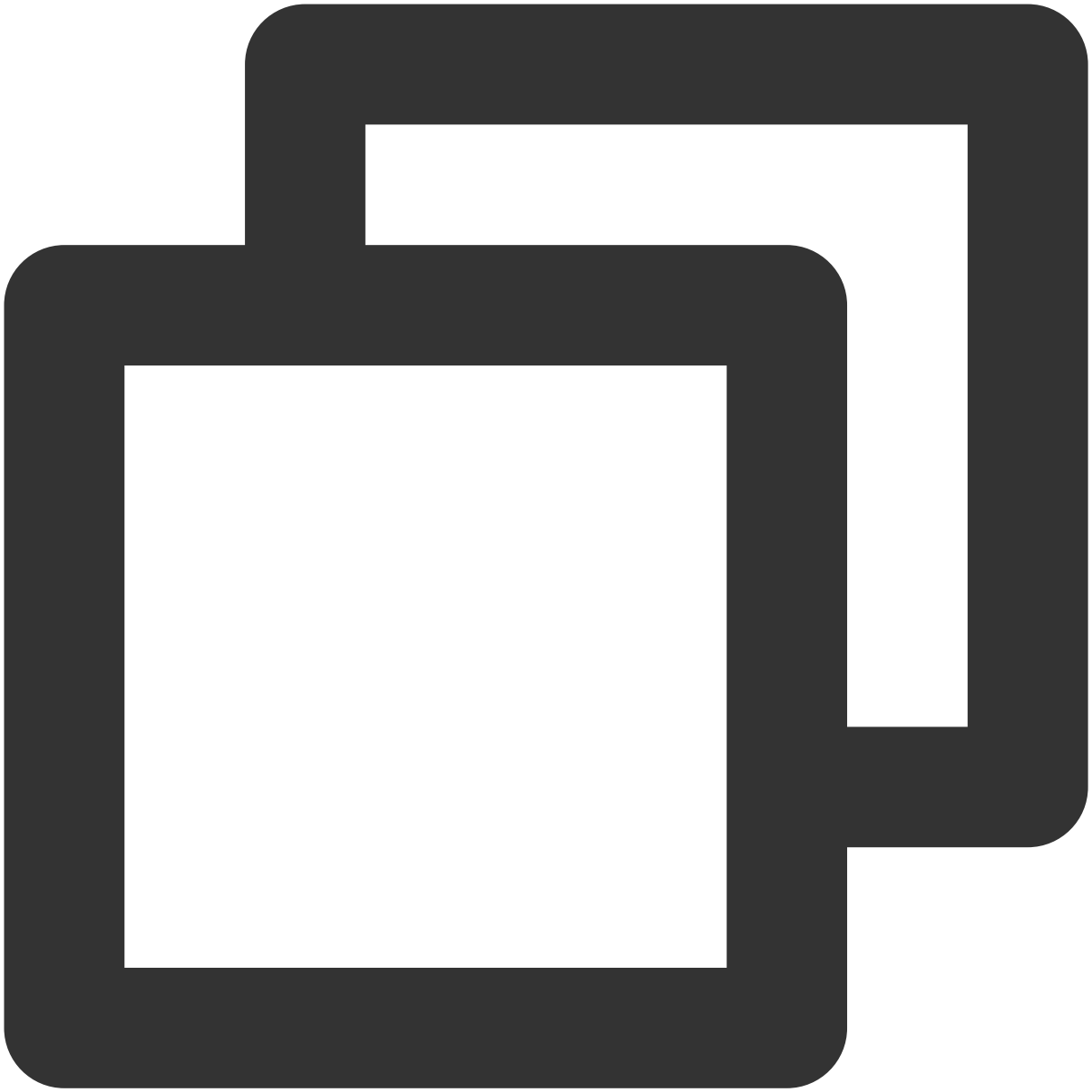
使用 CocoaPods 导入组件，具体步骤如下：

1. 在您的 `Podfile` 文件中添加以下依赖。



```
pod 'TUICallKitVoIPExtension/Professional'
```

2. 执行以下命令，安装组件。

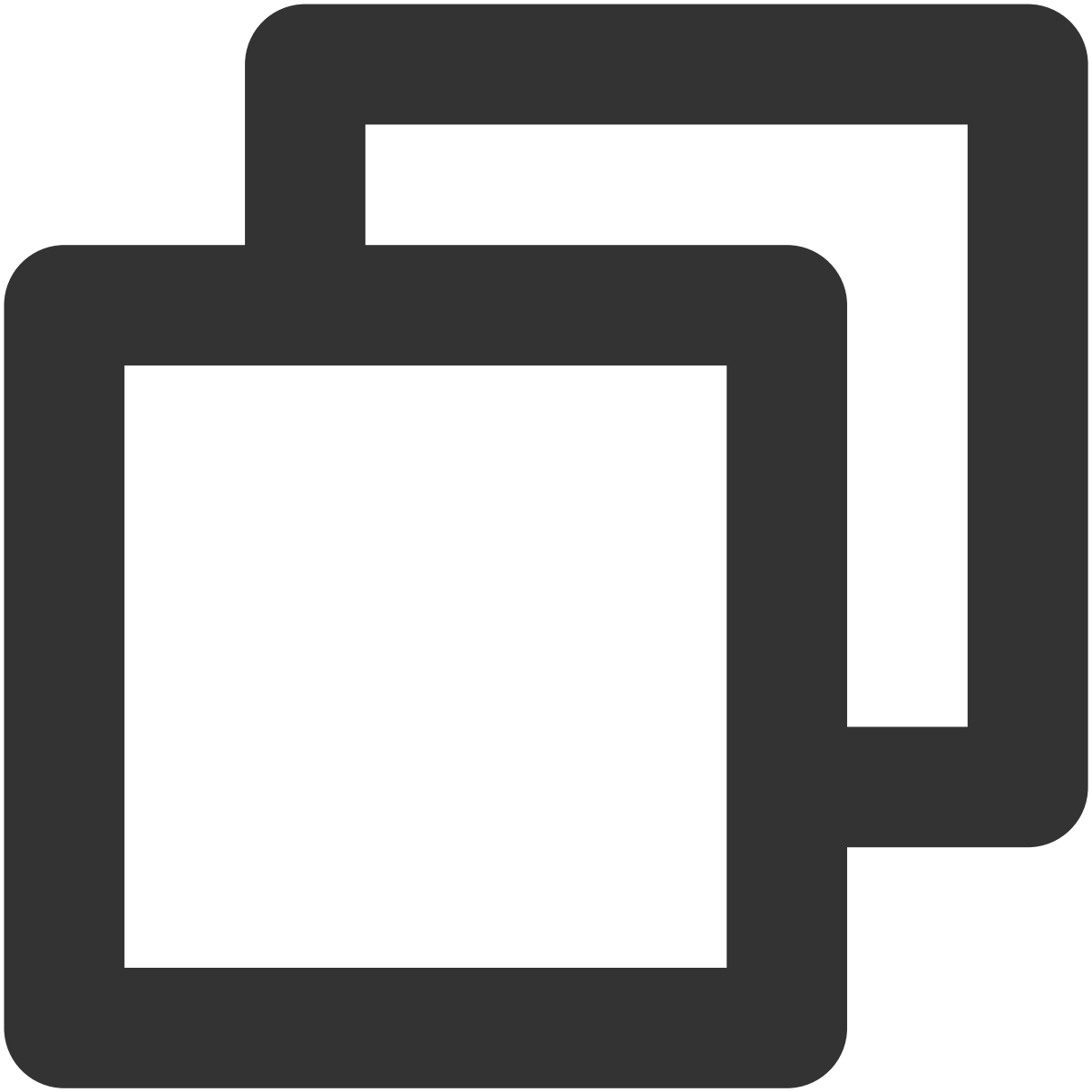


```
pod install
```

3. 上报 步骤 2：记录的证书 ID 。

Swift

Objective-C

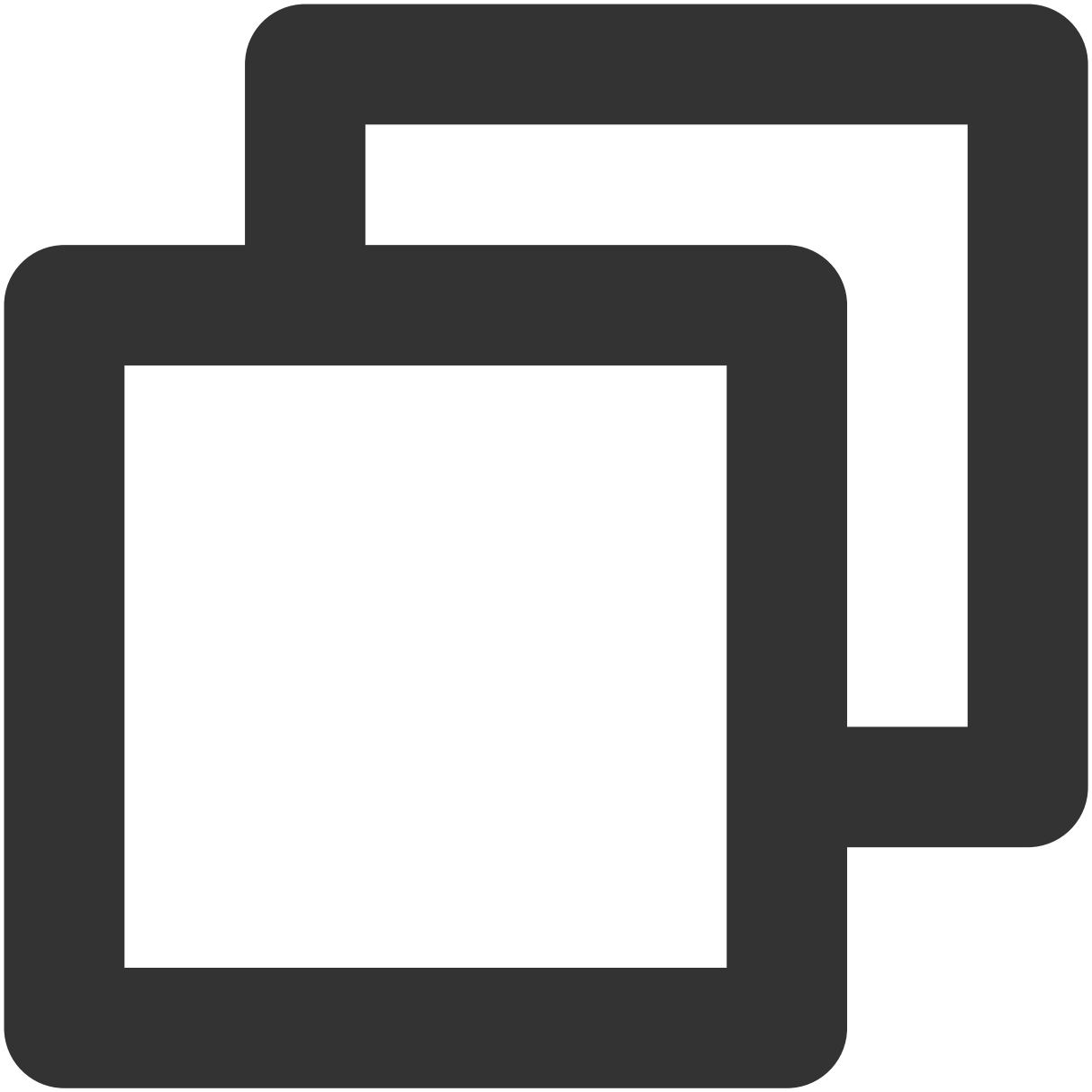


```
import TUICallKitVoIPExtension

func application(_ application: UIApplication, didFinishLaunchingWithOptions launch

    // 上报证书 ID
    TUICallKitVoIPExtension.setCertificateID(1234)

    return true
}
```



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

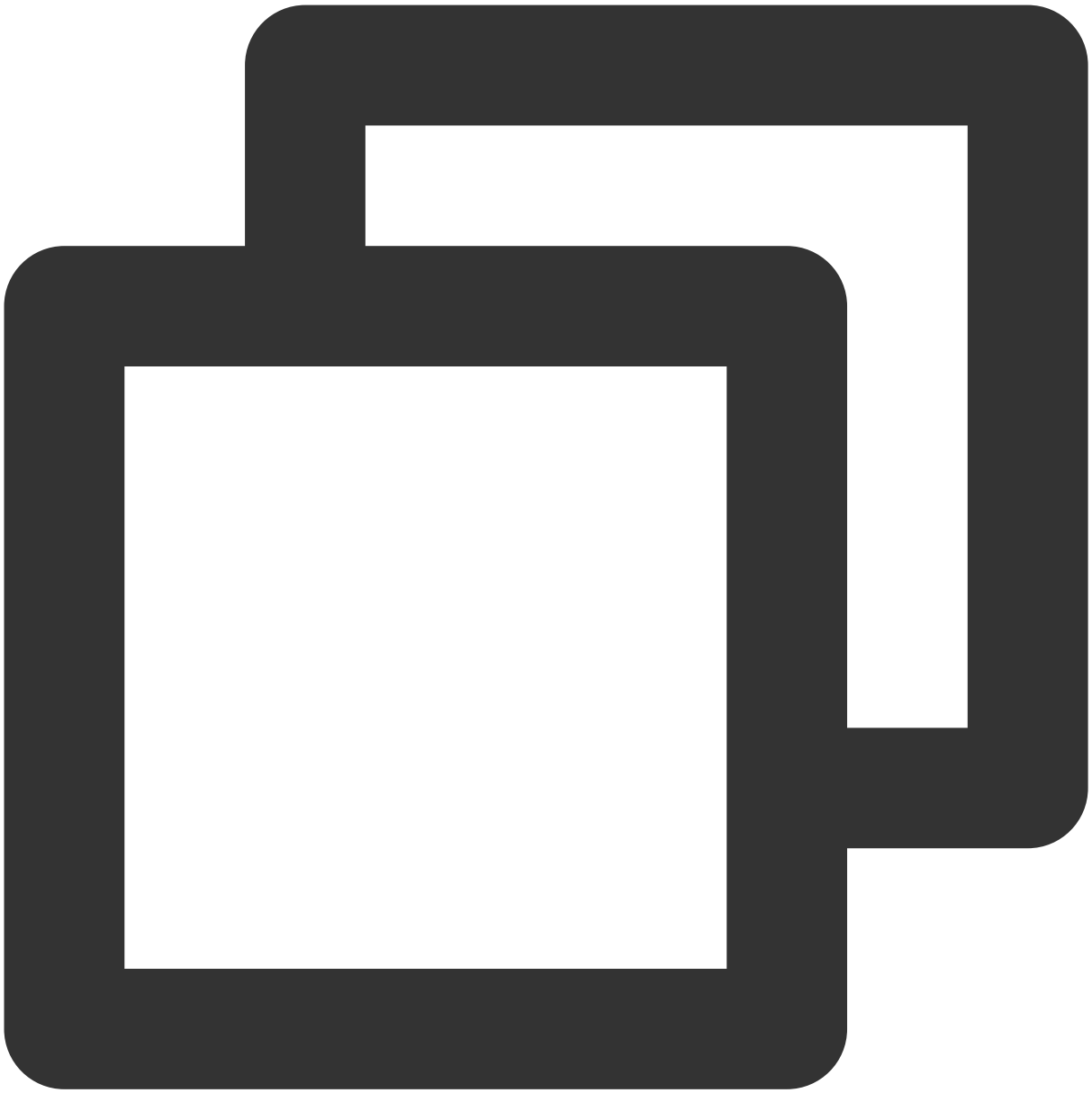
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD

    // 上报证书 ID
    [TUICallKitVoIPExtension setCertificateID:1234];

    return YES;
}
```

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

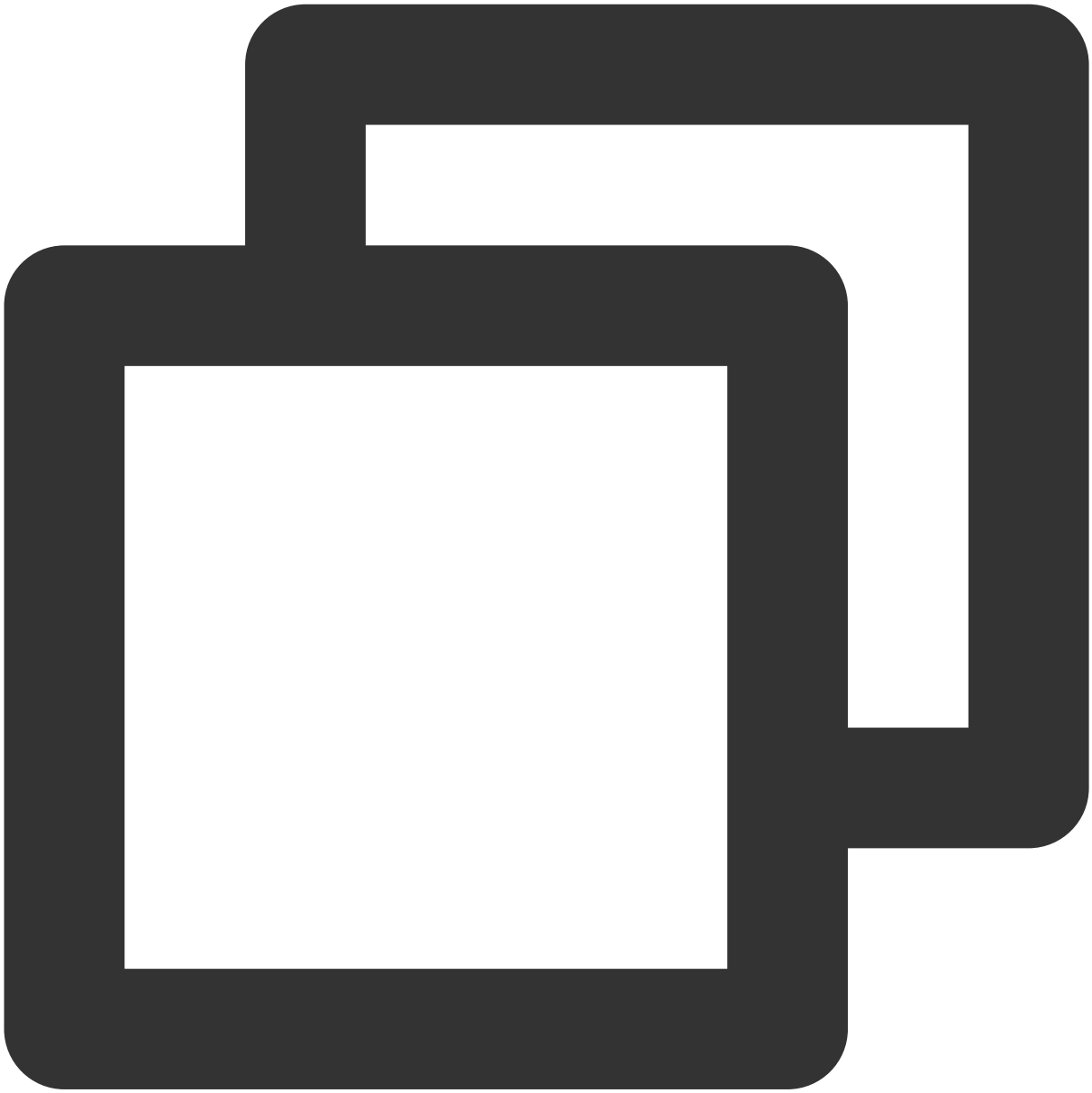
如果您需要拨打 VoIP 通话，需要在调用 call 的时候设置 OfflinePushInfo 中的 iOSPushType 字段为 TUICallIOSOfflinePushTypeVoIP，默认为 `TUICallIOSOfflinePushTypeAPNs` 。

Swift

Java

Dart

Objective-C



```
import TUICallKit_Swift
import TUICallEngine

let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "您有一个新的通话"
```

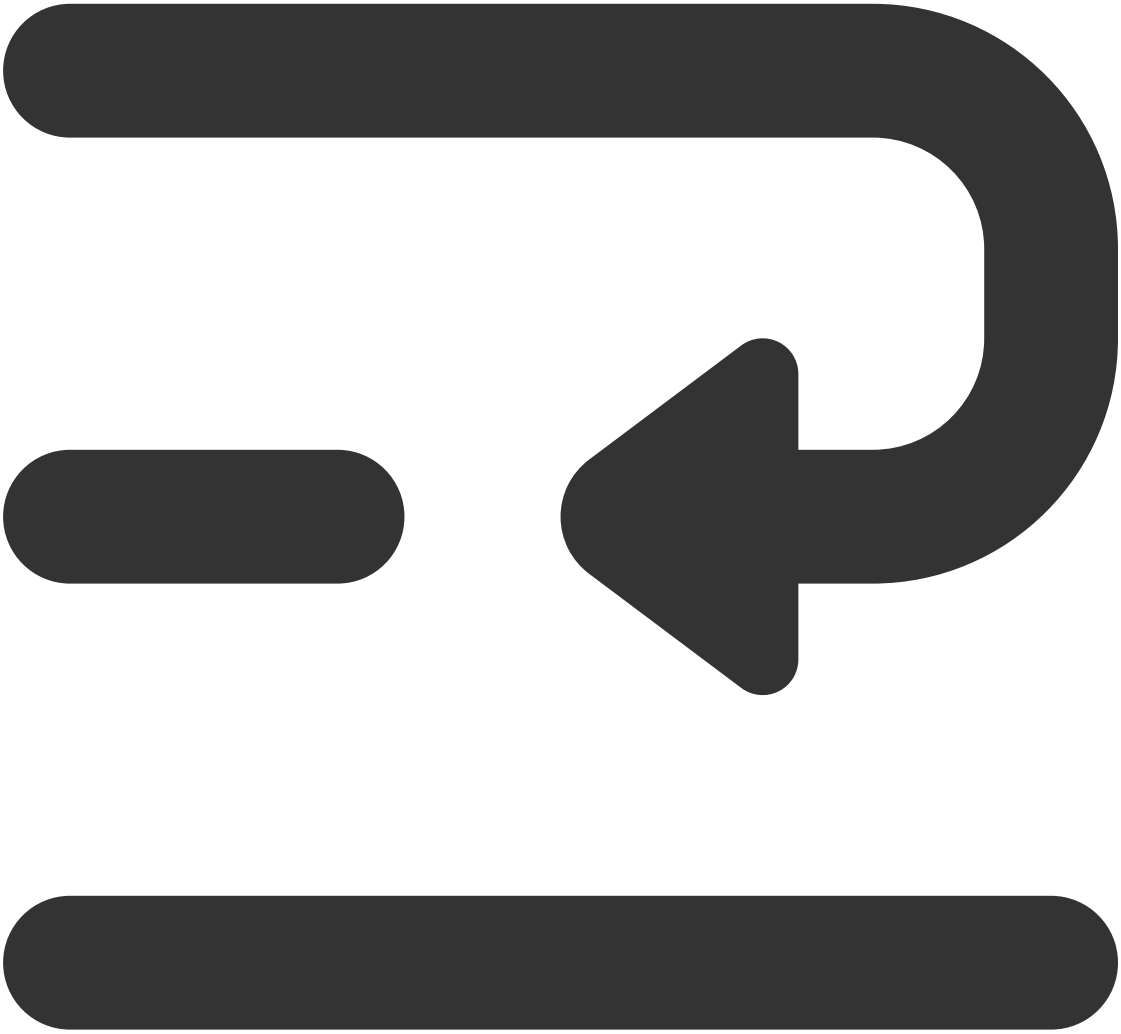
```
pushInfo.iOSPushType = .voip
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "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"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

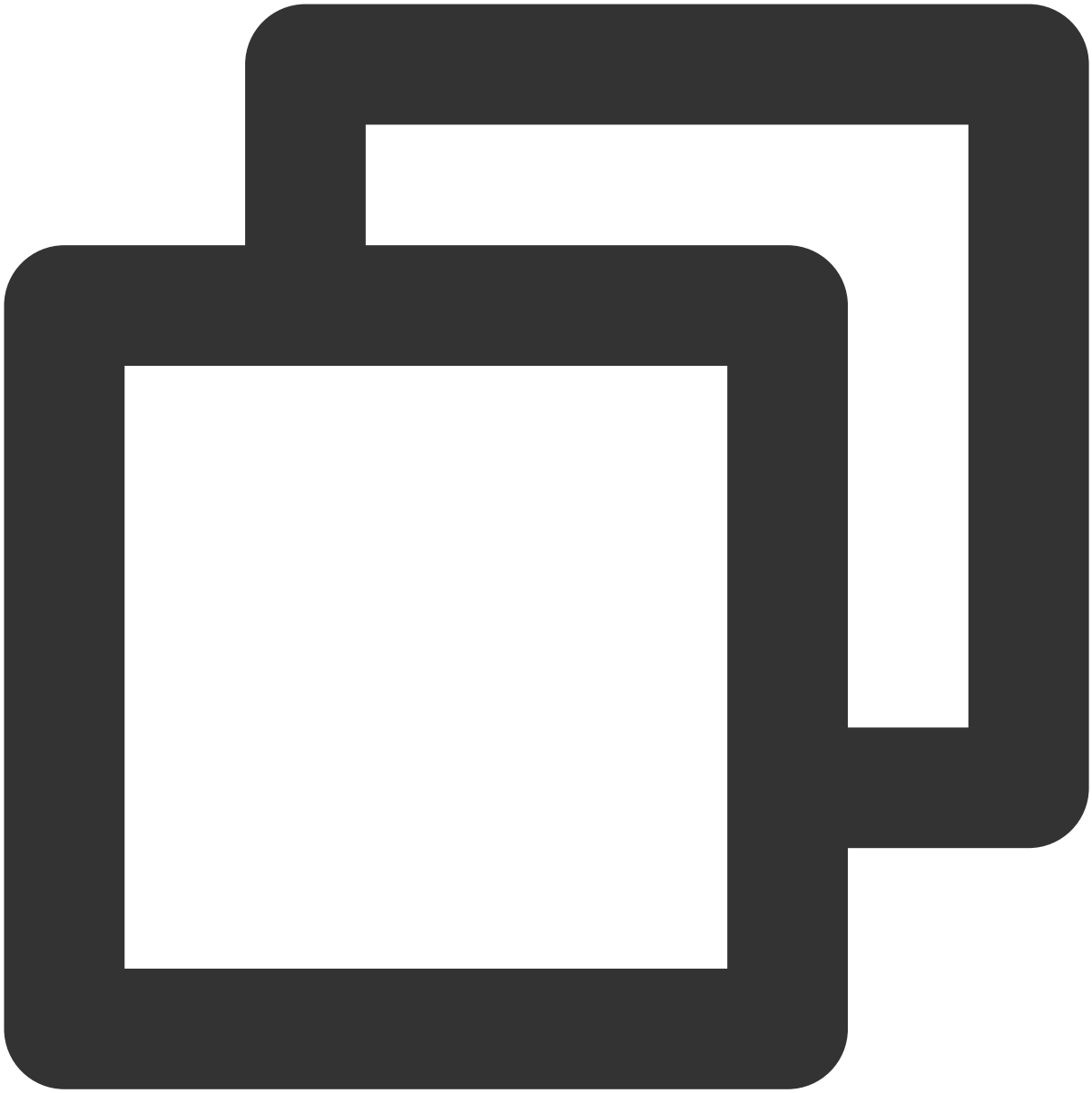
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



版权所有：腾讯云计算（北京）有限责任公司

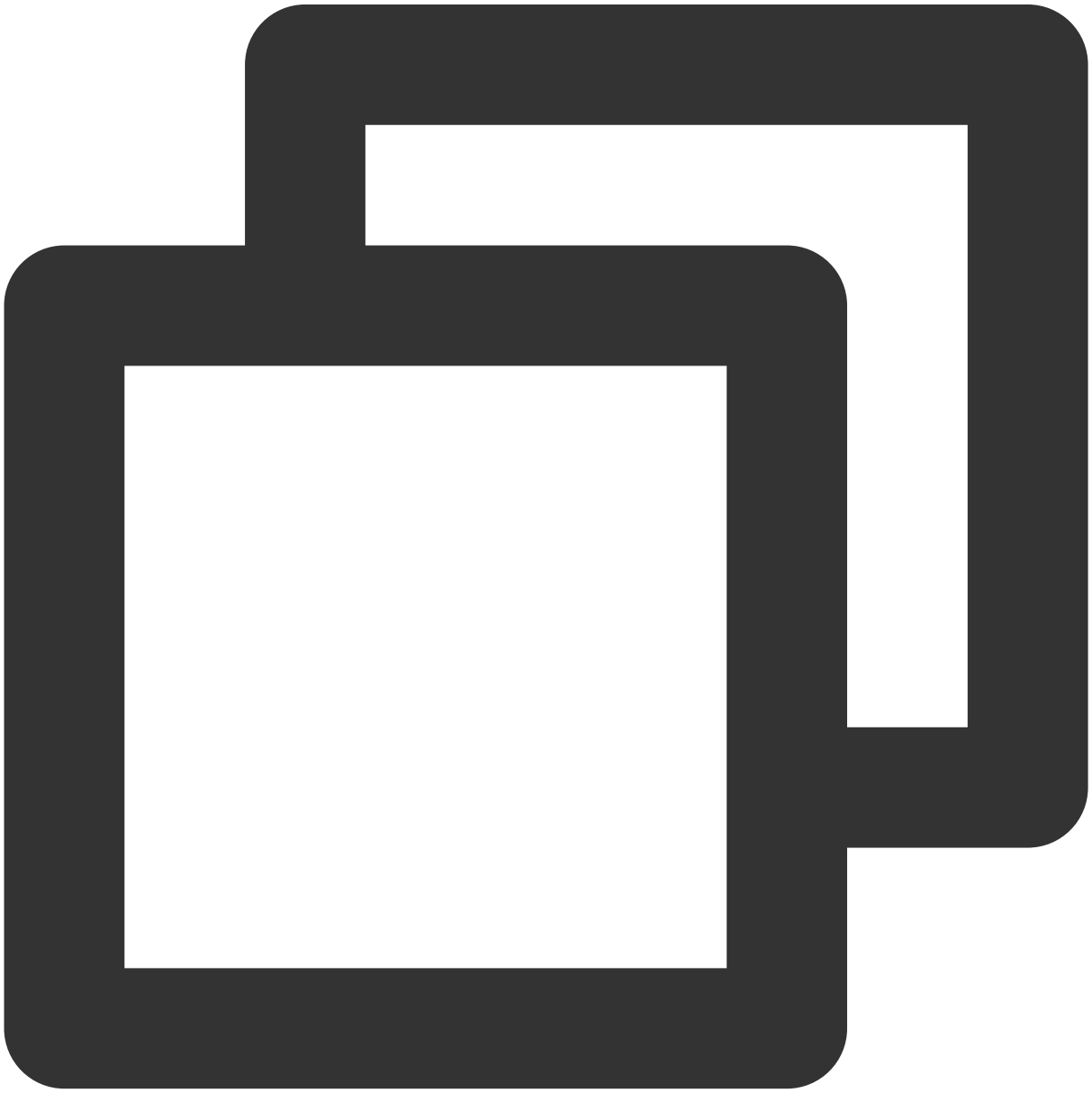


```
TUICallDefine.OfflinePushInfo offlinePushInfo = new TUICallDefine.OfflinePushInfo()
offlinePushInfo.setTitle("");
offlinePushInfo.setDesc("You have receive a new call");
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to 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;

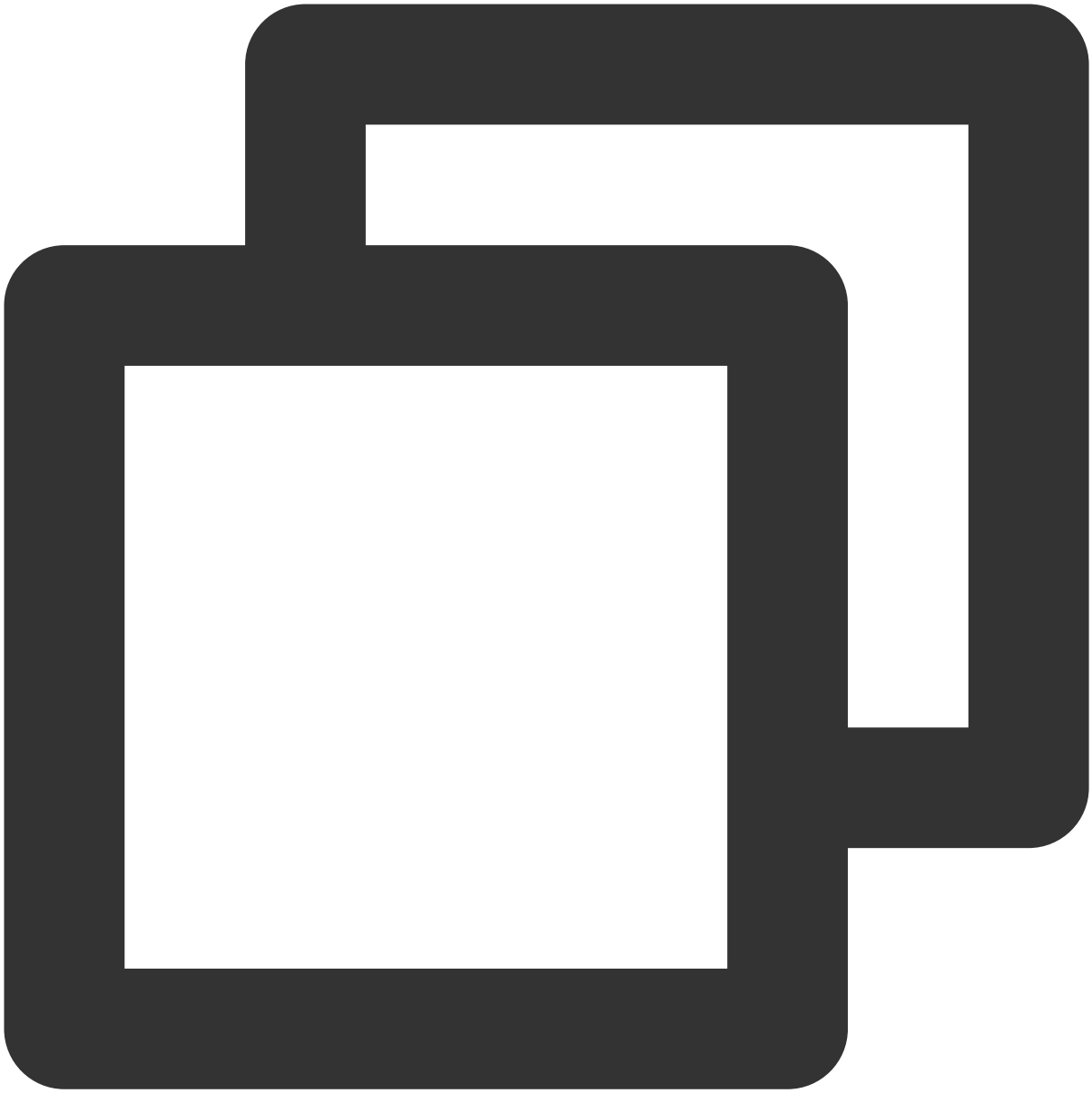
TUICallKit.createInstance(context).call("mike", TUICallDefine.MediaType.Video, para
```



```
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);
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (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";
    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] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

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

}];
```

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：

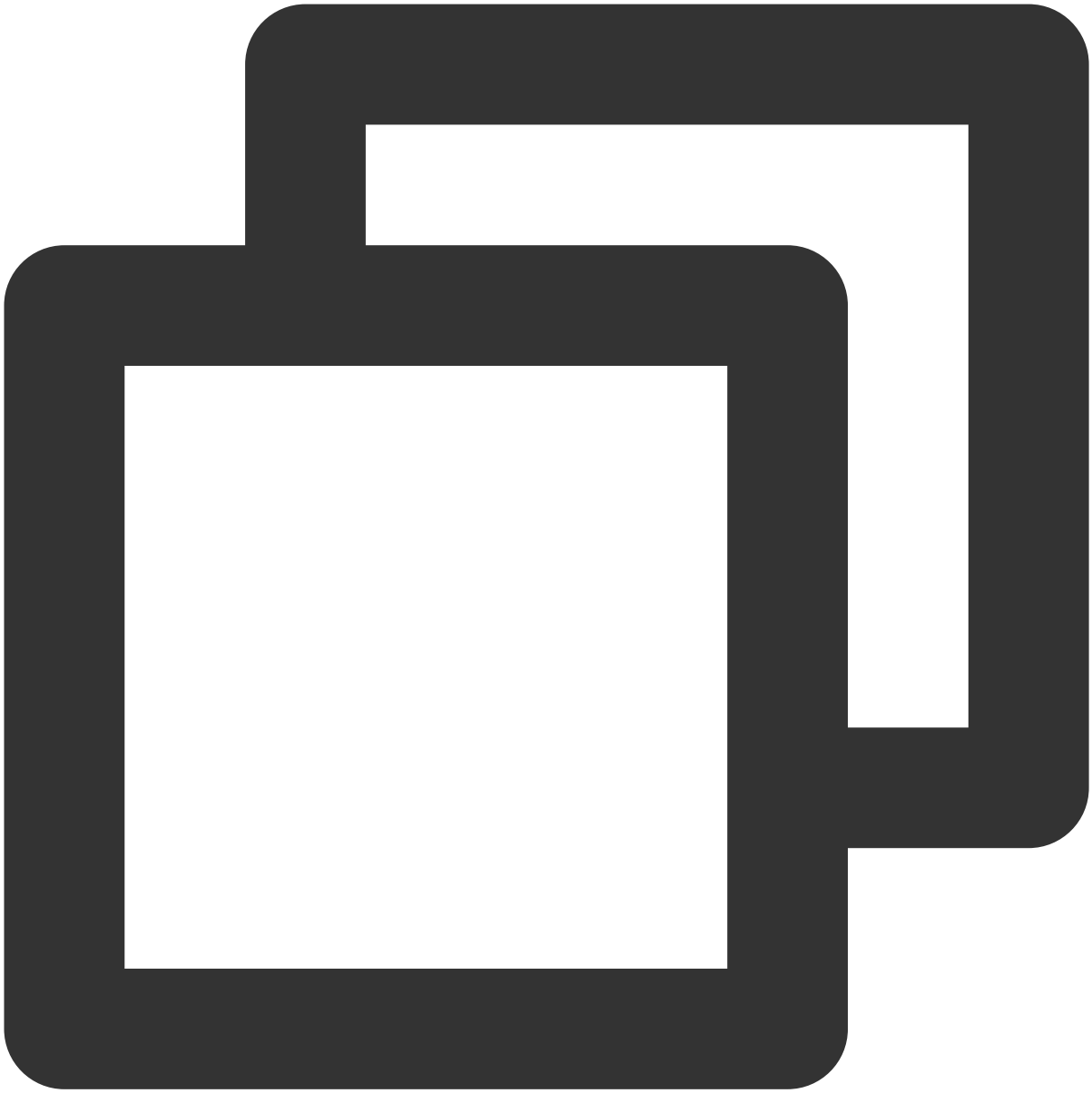
仅支持单人音视频通话的直接拨打。

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。

Swift

Objective-C



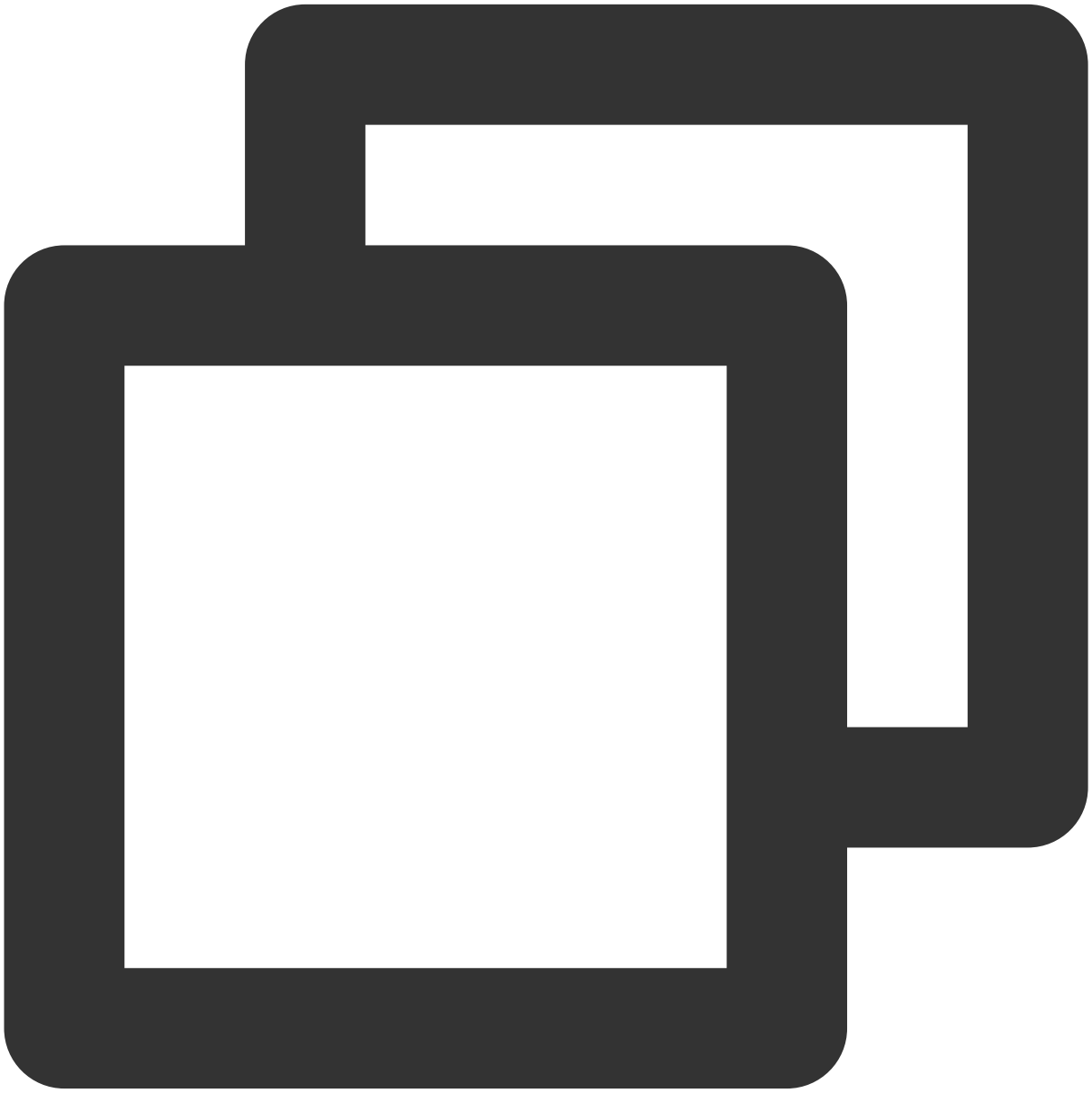
```
import TUICallKitVoIPExtension

/// AppDelegate 中的实现，iOS 13 以前版本的处理
func application(_ application: UIApplication, continue userActivity: NSUserActivit
    TUICallKitVoIPExtension.call(with: userActivity)
    return true
}

/// SceneDelegate 中的实现
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connect
    for userActivity in connectionOptions.userActivities {
```

```
        TUICallKitVoIPExtension.call(with: userActivity);
    }
}

func scene(_ scene: UIScene, continue userActivity: NSUserActivity) {
    TUICallKitVoIPExtension.call(with: userActivity);
}
```



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

/// 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;
}

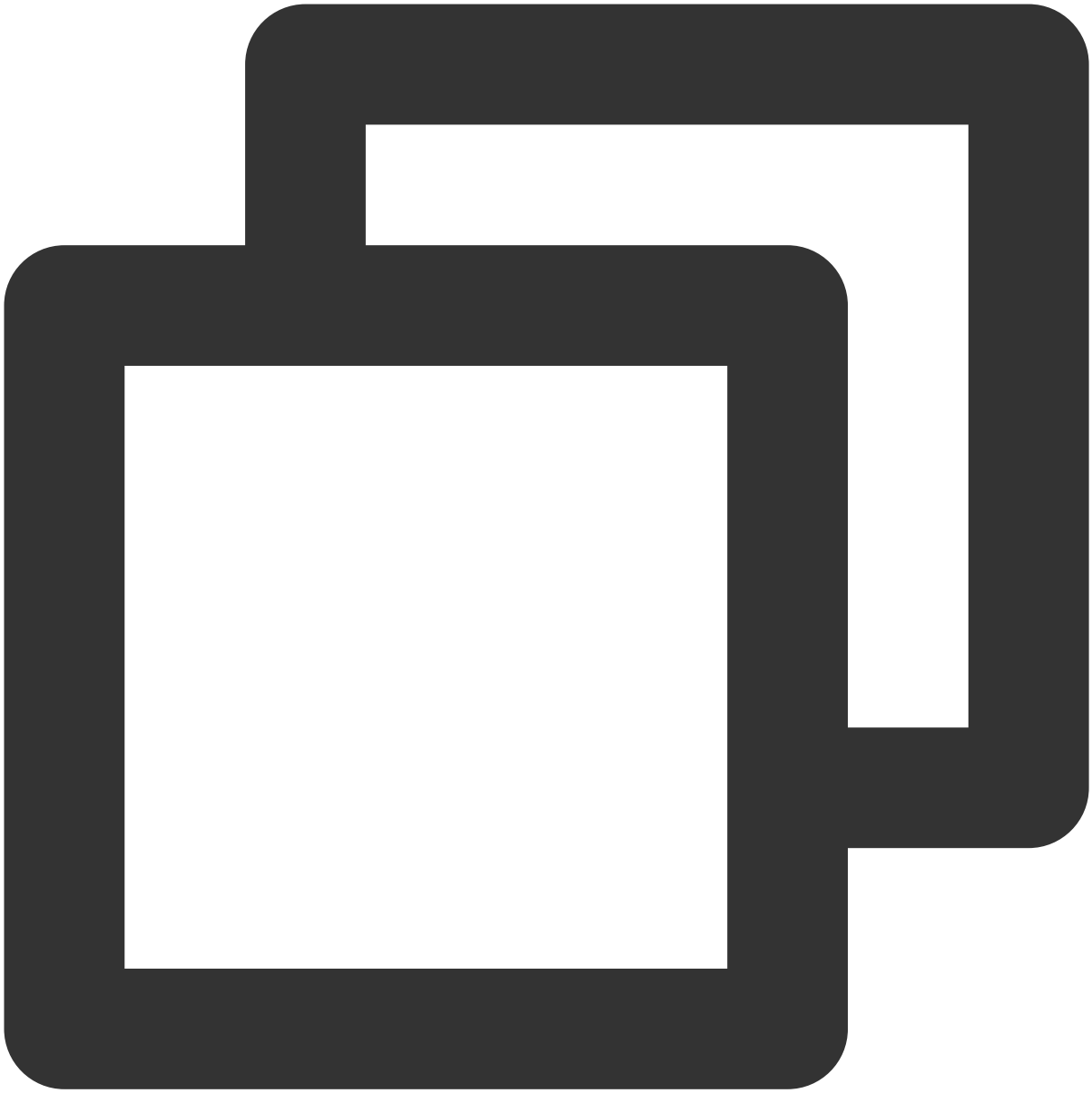
/// SceneDelegate 中的实现
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)sessionptions
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity *
        [TUICallKitVoIPExtension callWith:userActivity];
    }];
}

- (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。

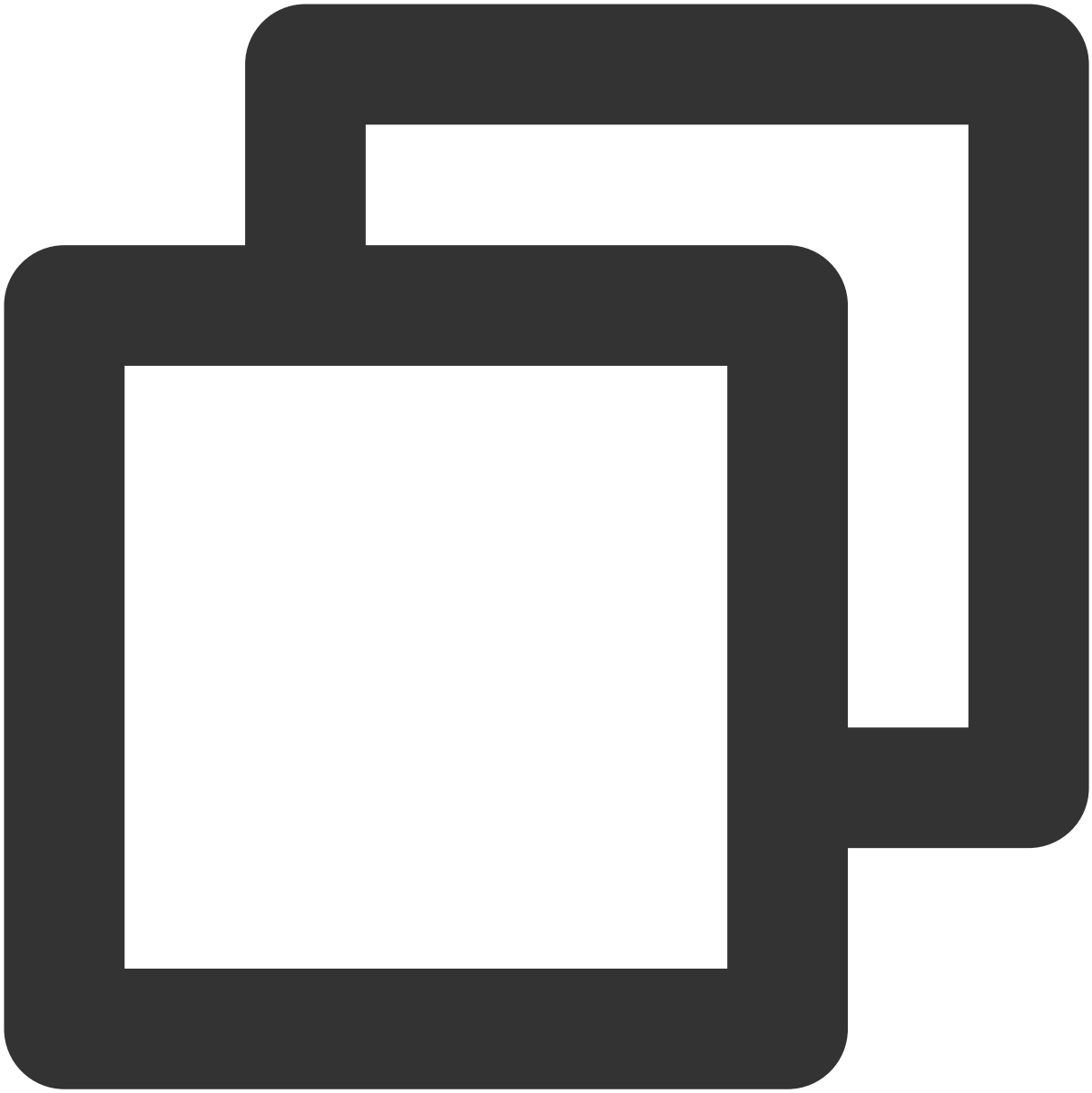
Swift

Objective-C



```
import TUICallKitVoIPExtension

func application(_ application: UIApplication, continue userActivity: NSUserActivit
    TUICallKitVoIPExtension.call(with: userActivity)
    return true
}
```



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

- (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（非常重要）。**

Notification

最近更新时间：2024-07-29 15:23:09

TUICallKit 组件支持通过接入推送插件，实现离线推送的功能。 为了帮助开发者在项目中轻松实现离线推送功能，我们推荐使用 TIMPush 插件（付费），TIMPush 推送插件有以下优势：

接入周期短，预计全厂商接入仅需要 30 分钟。

支持数据统计和链路追踪，方便您查看推送触达率、点击率和转化率等各类指标。

支持全员/标签推送，方便您将营销广告、通知、新闻资讯等内容推送给所有用户或指定群体。

支持 uni-app 和 Flutter 等跨平台框架。

集成效果

锁屏时的效果	在后台时的效果



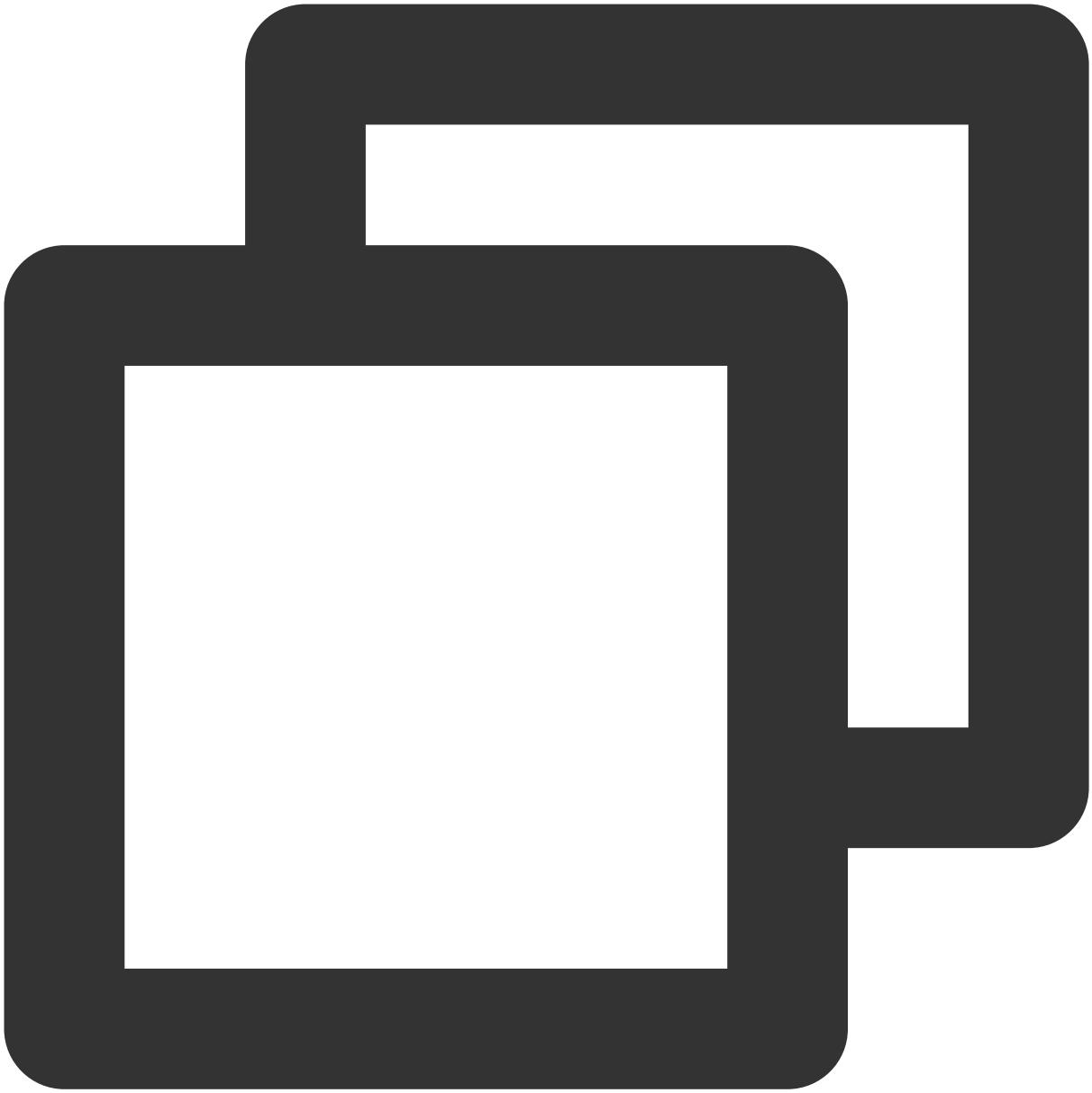


配置离线推送

1. 集成 TIMPush 组件。
2. 开启 App 远程推送。
3. 生成证书。
4. 上传证书到 Tencent RTC 控制台。
5. 完成工程配置
6. 配置推送参数。
7. 拨打离线推送电话。

步骤1：集成 TIMPush 组件

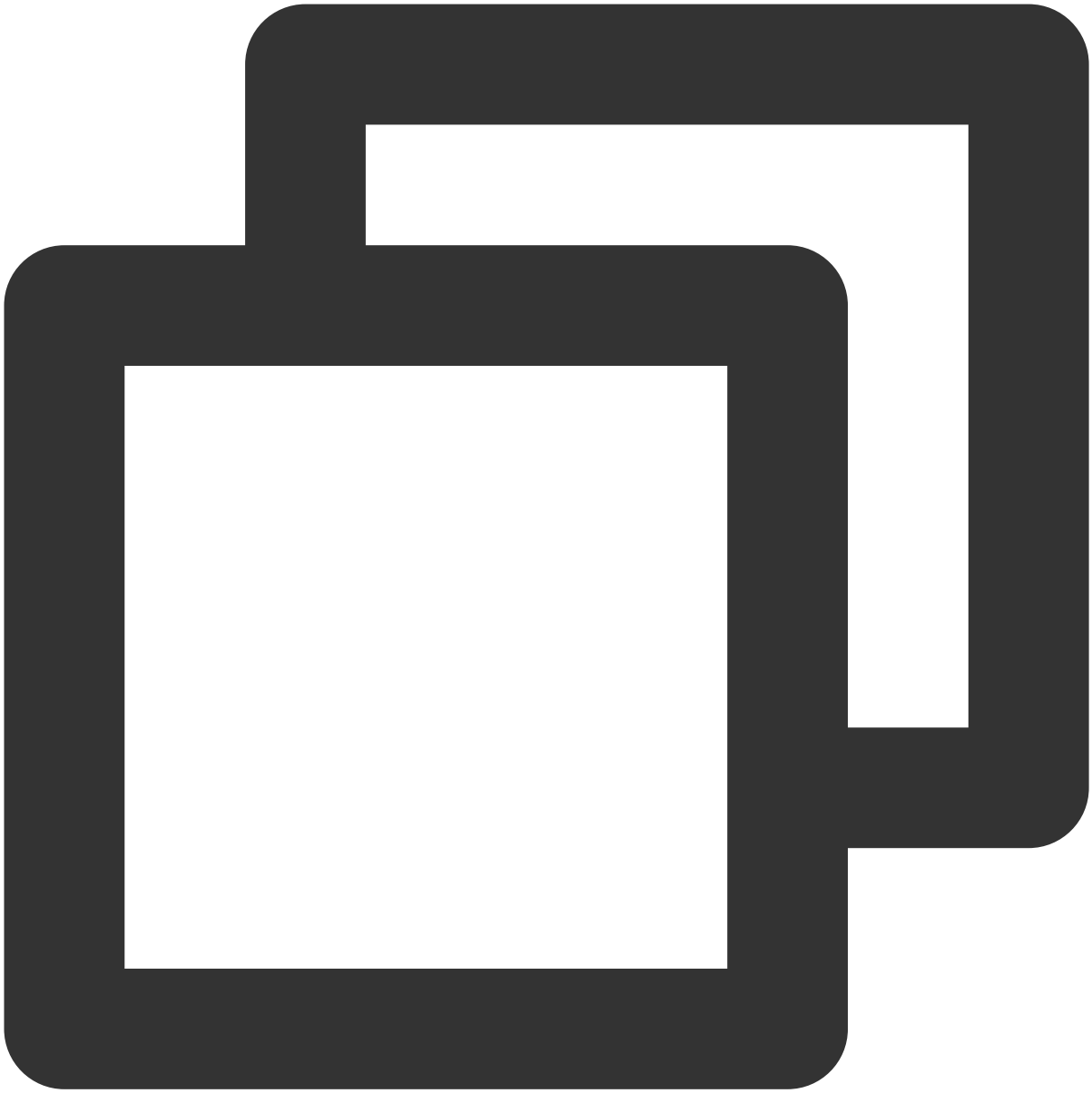
1. TIMPush 组件支持 cocoapods 集成，您需要在 Podfile 中添加组件依赖。



```
target 'YourAppName' do
  # Uncommment the next line if you're using Swift or would like to use dynamic fra
  use_frameworks!
  use_modular_headers!

  # Pods for Example
  pod 'TIMPush', '7.9.5668'
end
```

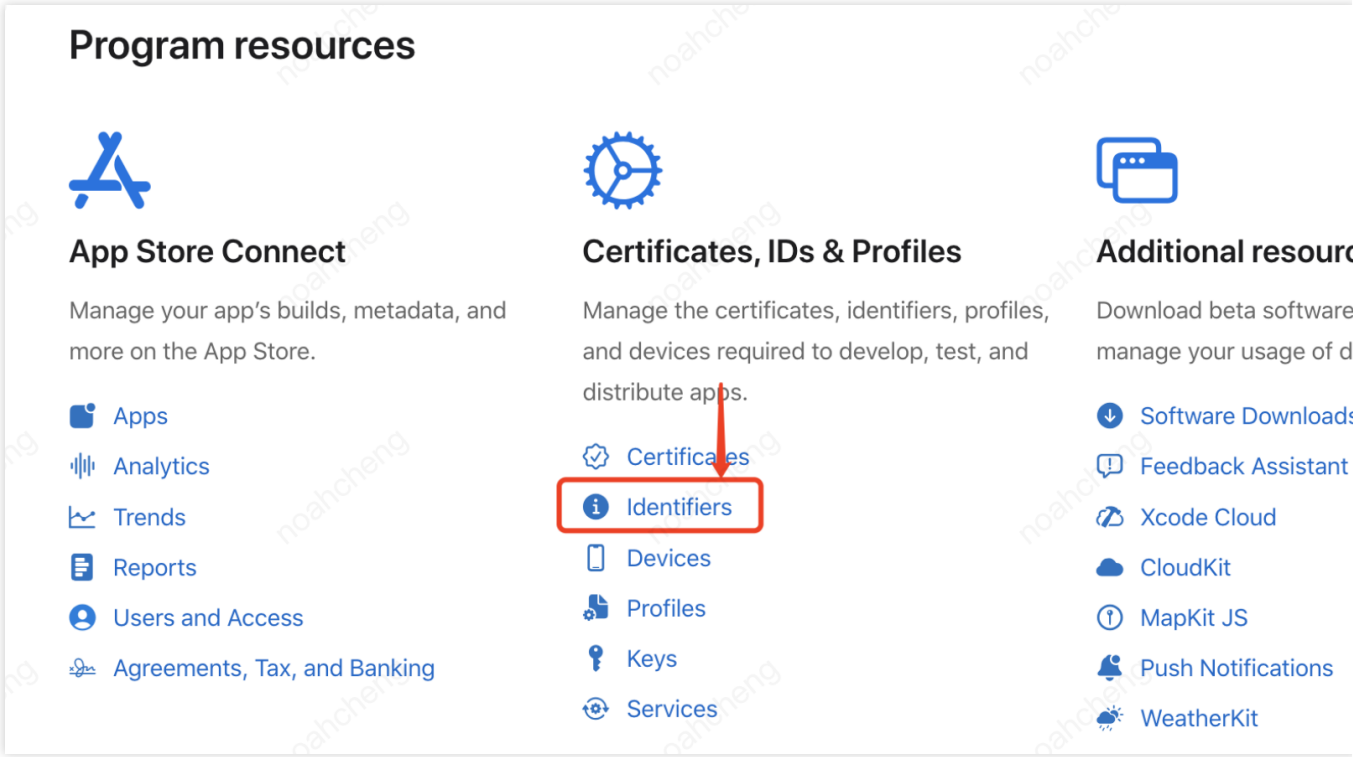
2. 执行以下命令，安装 TIMPush 组件。



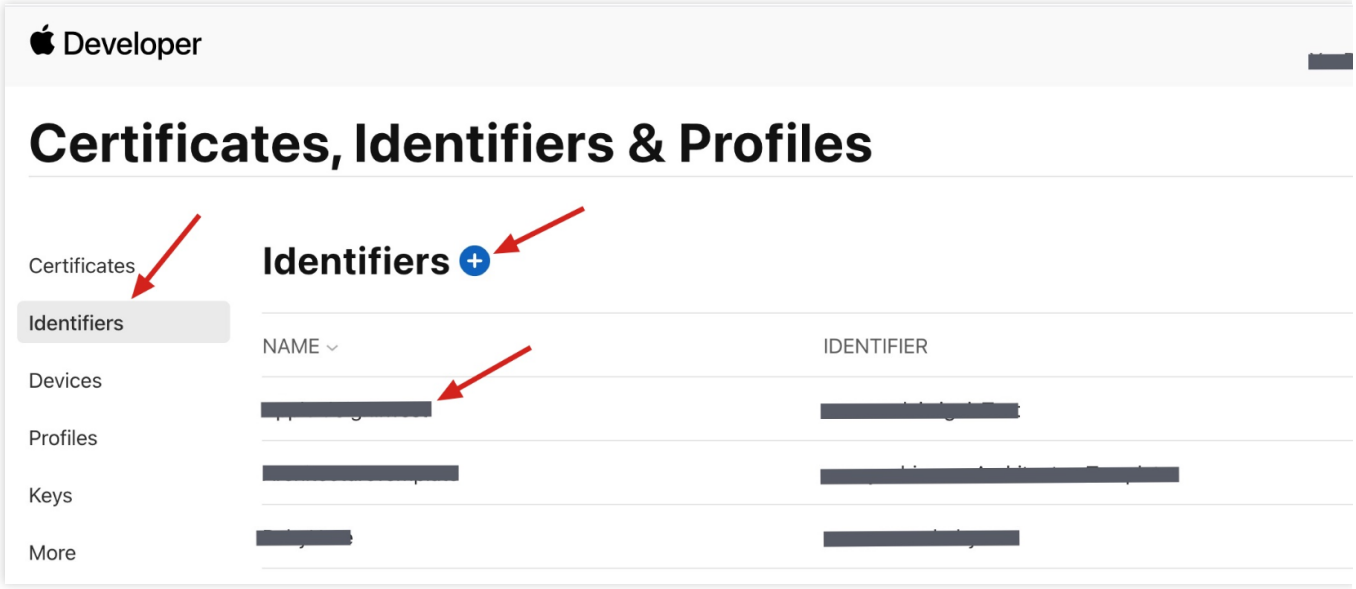
```
pod install
# 如果无法安装 TUIKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。
pod repo update
```

步骤2：开启 APP 远程推送

1. 登录 苹果开发者中心 网站，单击 **Certificates, IDs & Profiles** 选项卡中的 **Identifiers**，进入 **Certificates, Identifiers & Profiles** 页面。



2. 单击 **Identifiers** 右侧的 **+**。



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

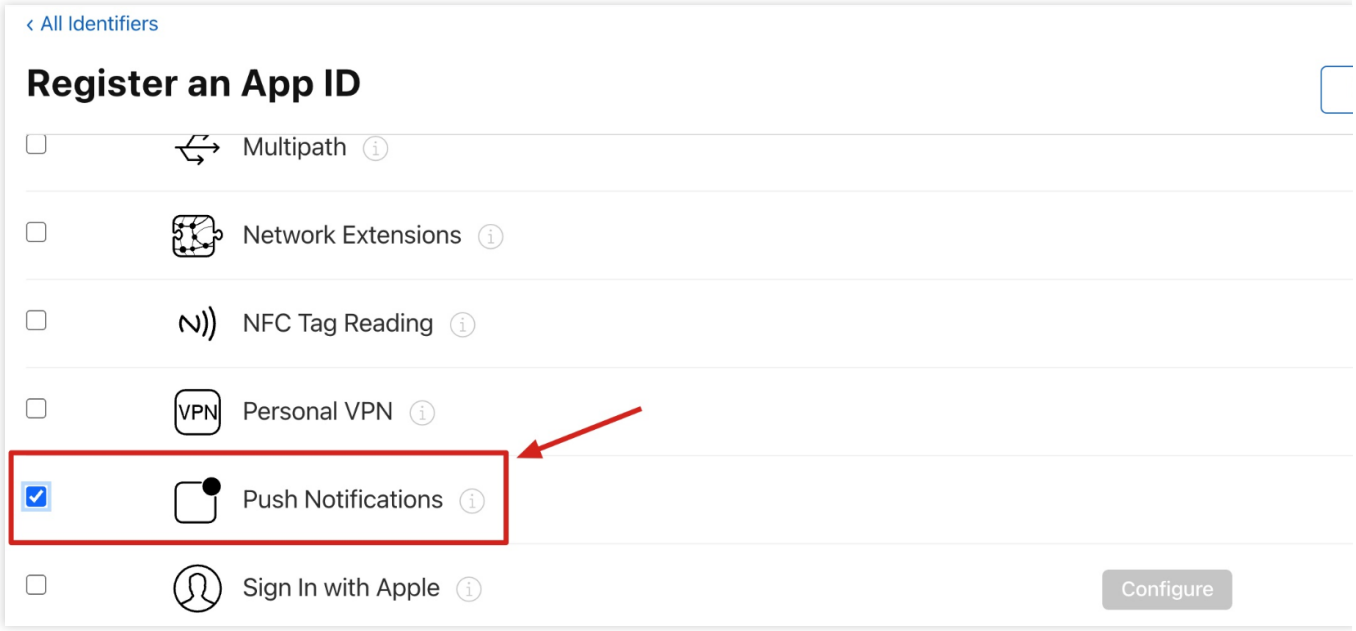
说明：

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。

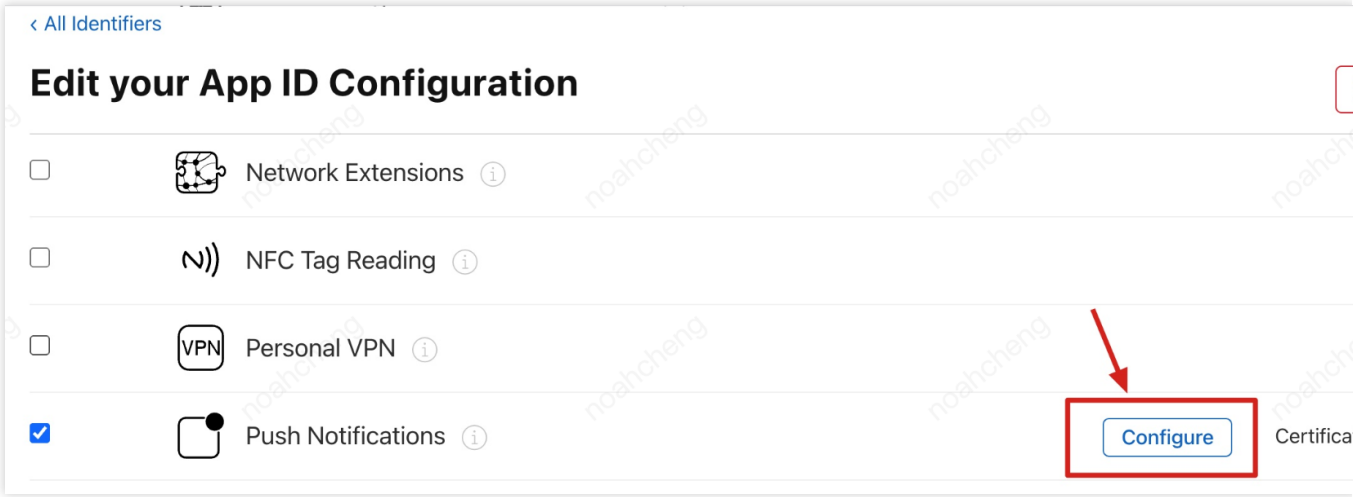
5. 选择 **App**，单击 **Continue** 进行下一步。

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3：生成推送证书

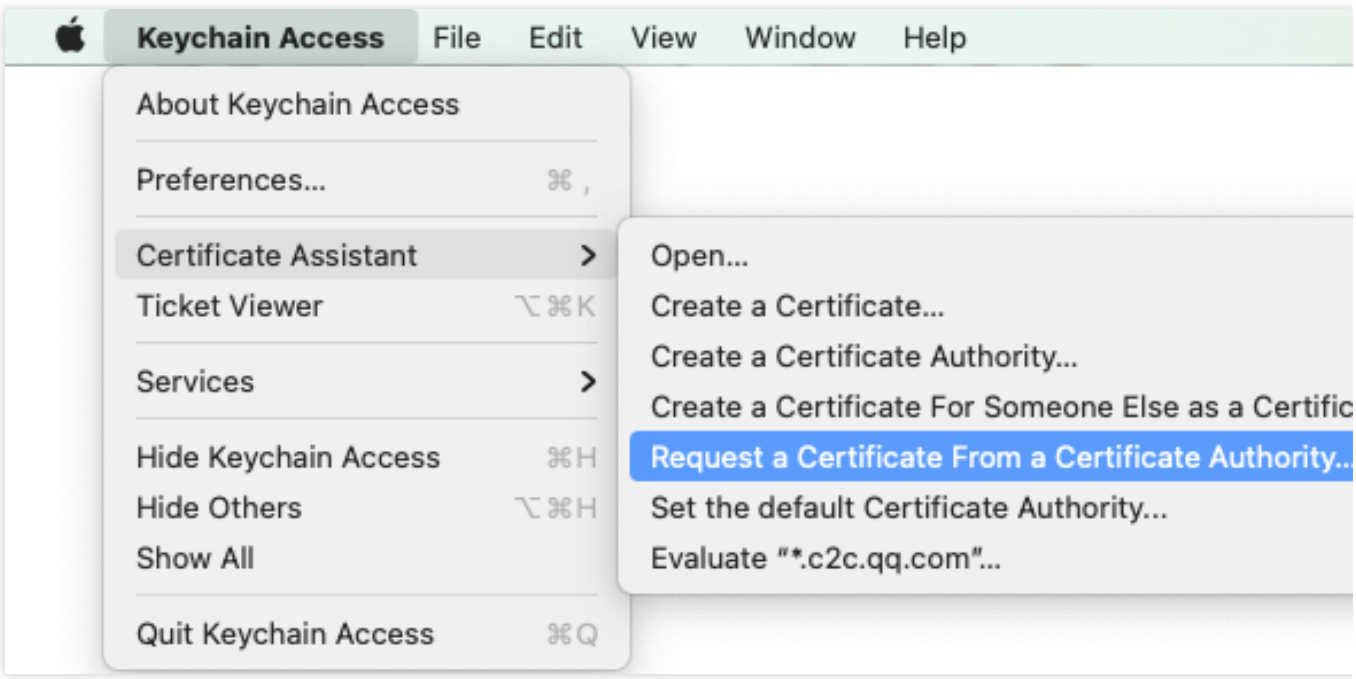
1. 选中您的 **AppID**，找到 **Push Notifications** 配置项，选择 **Configure**。



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。
3.
我

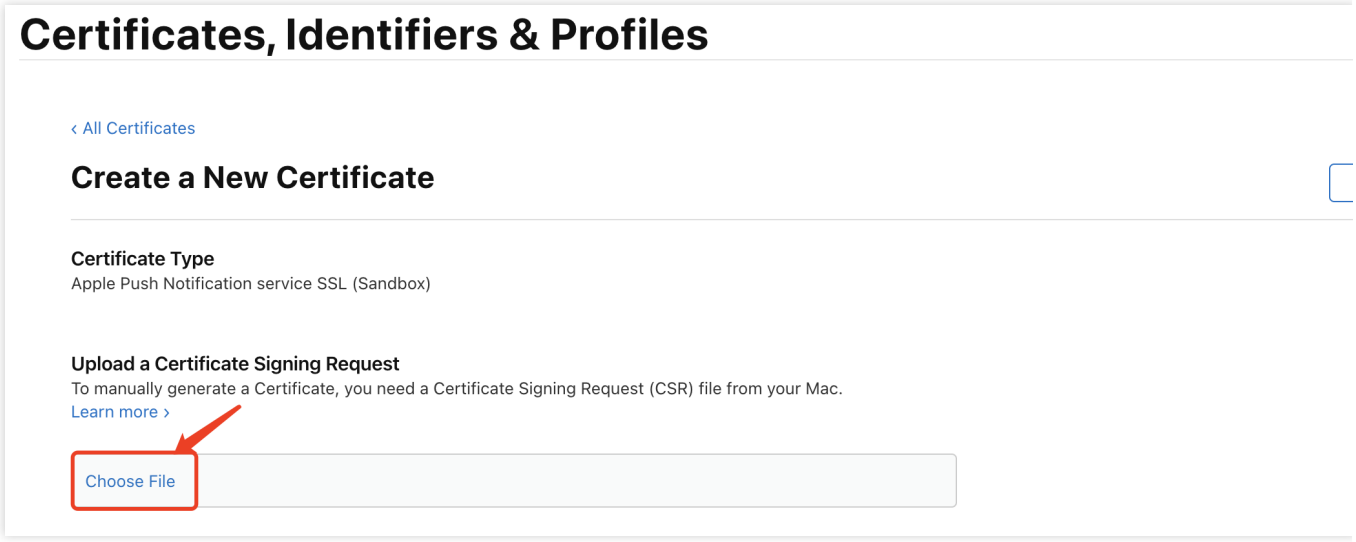
们先选择开发环境（Development）的 **Create Certificate**，系统将提示我们需要一个 **Certificate Signing Request（CSR）**。

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（Saved to disk）**，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。

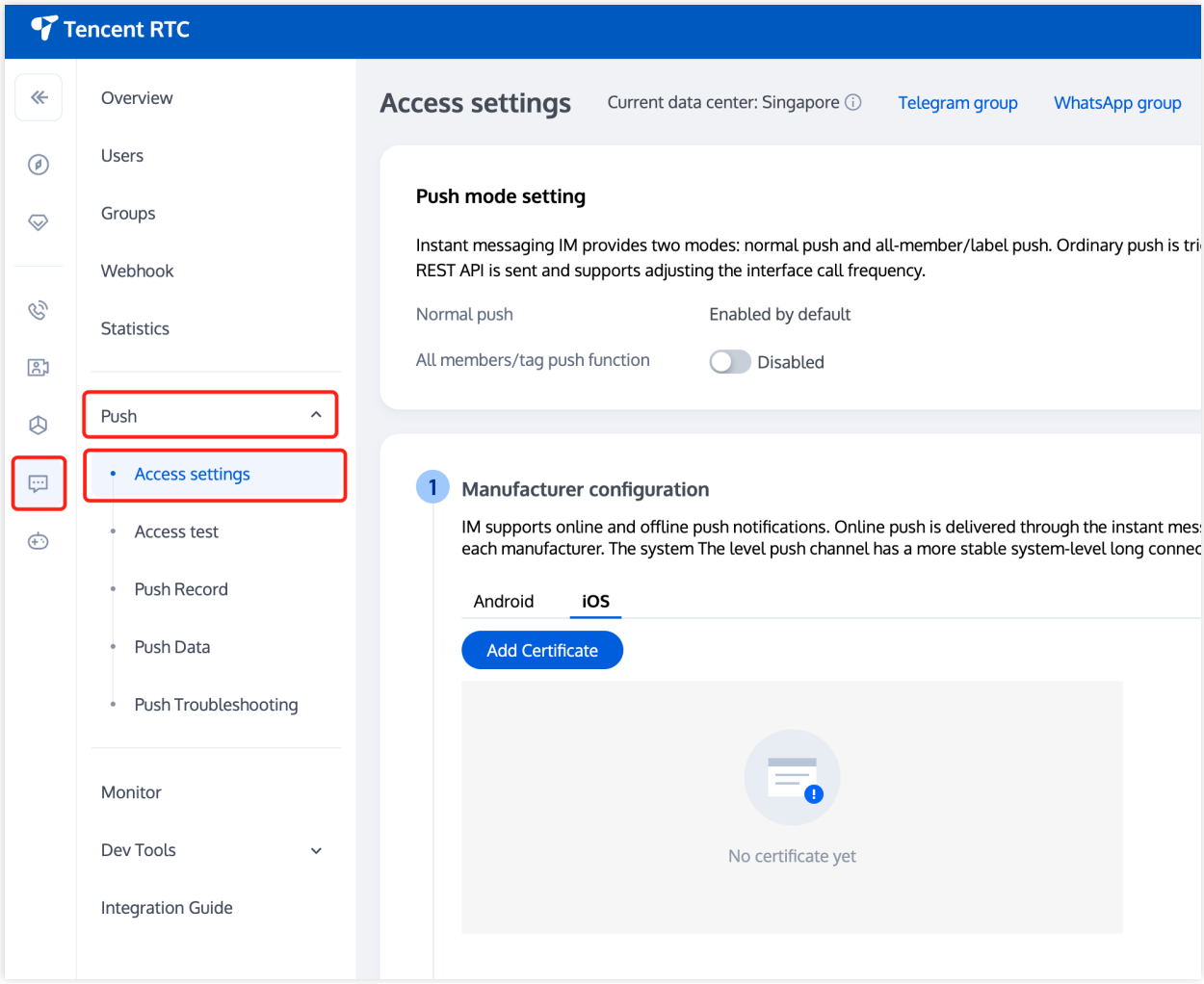
10. 双击打开下载的开发环境和生产环境的 `SSL Certificate` ，系统会将其导入钥匙串中。
11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（ `Apple Development IOS Push Service` ）和生产环境（ `Apple Push Services` ）的 P12 文件。

说明：

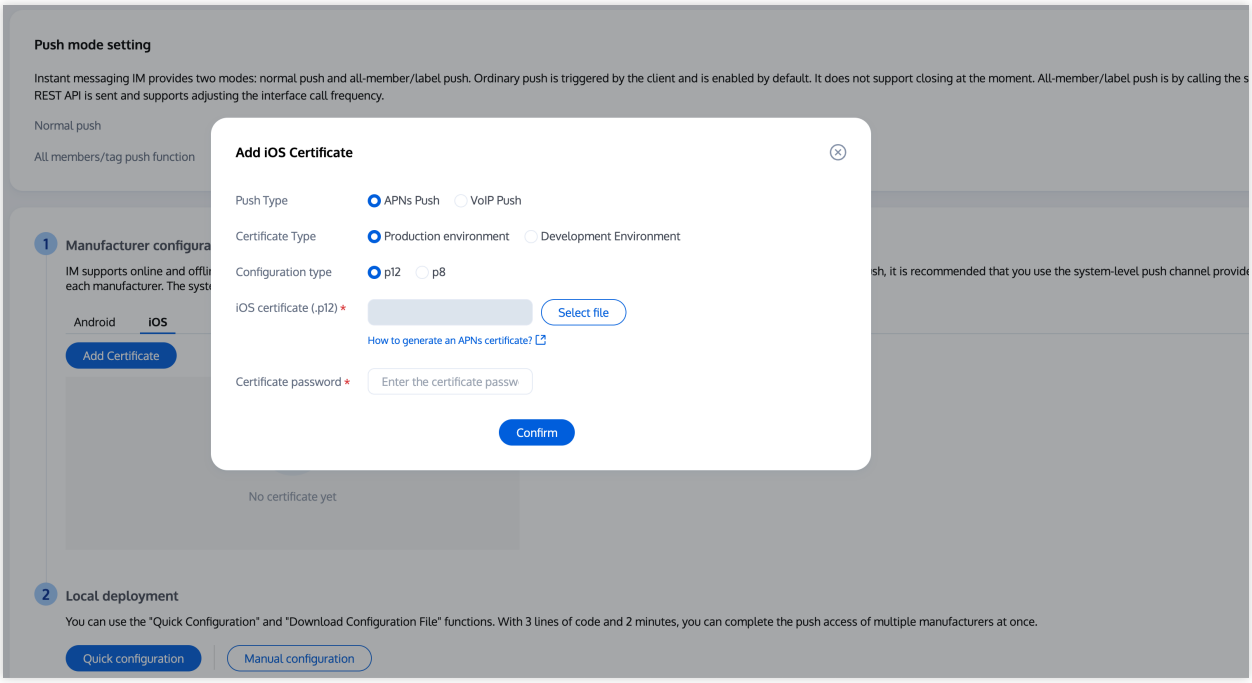
保存 `P12` 文件时，请务必要为其设置密码。

步骤5：上传证书到 Tencent RTC 控制台

1. 登录 Tencent RTC 控制台。
2. 单击目标应用卡片，选择左侧的 **Chat** 标签，点击 **Push** ，点击 **Access settings**。



3. 单击 **iOS 原生离线推送设置**右侧的**添加证书**。
4. 选择证书类型，上传 iOS 证书（p12），设置证书密码，单击**确认**。



注意：

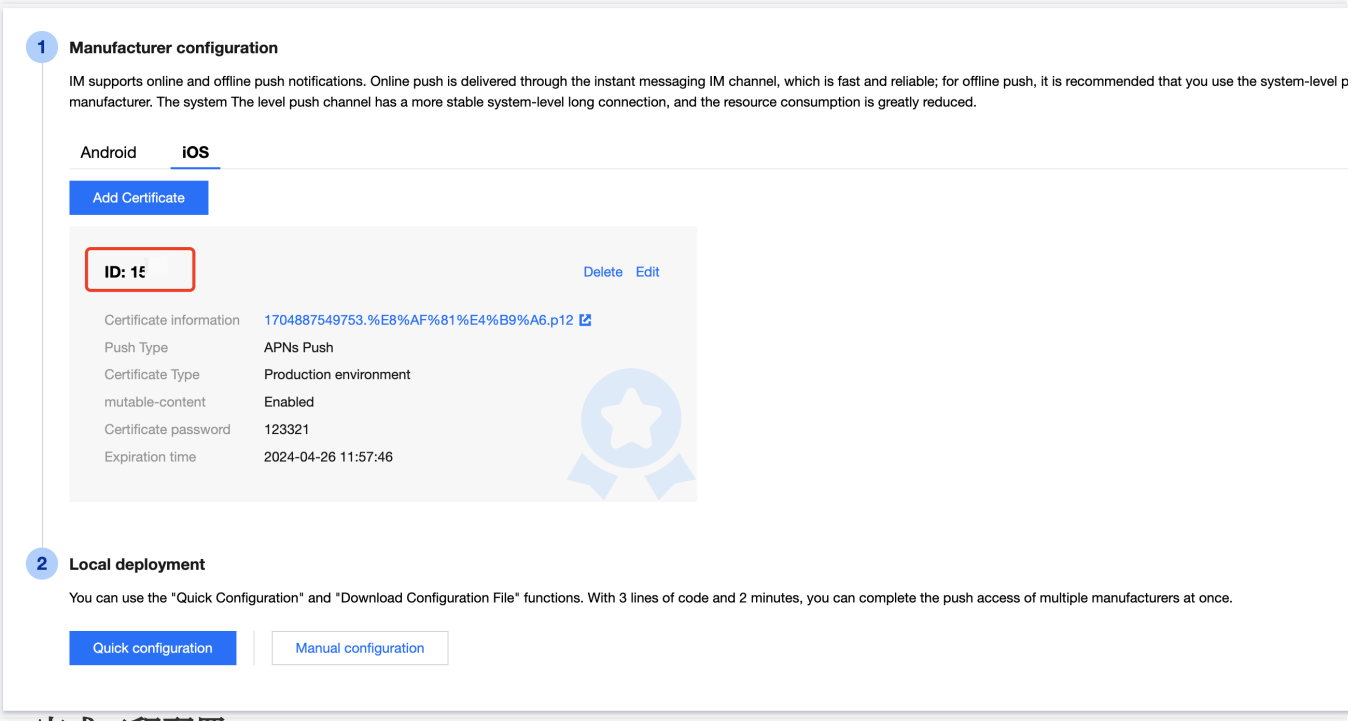
上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

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。

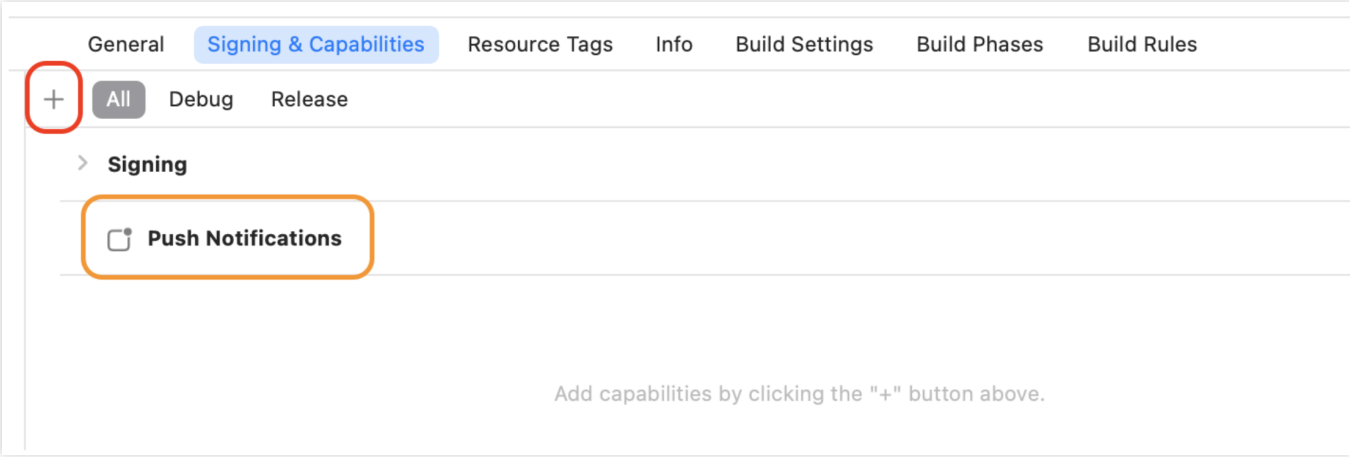
5. 待推送证书信息生成后，记录证书的 ID，将在 步骤6：配置推送参数 作为**必填参数**使用。



步骤6：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** 页面中单击红框中的 **+**，然后选择并添加 **Push Notifications**，添加后的结果如图中黄框所示：

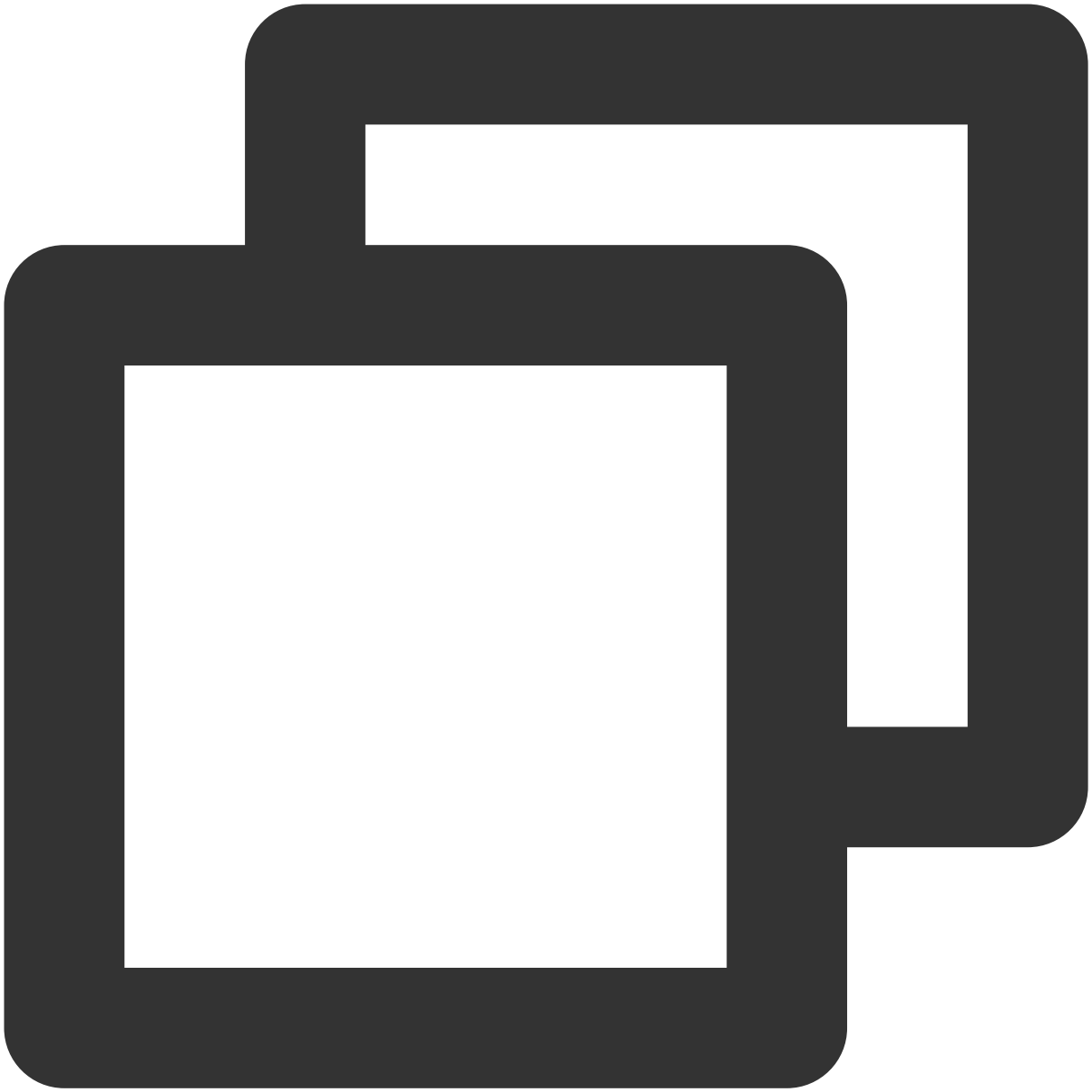


步骤7：配置推送参数

您需要在 AppDelegate 中，实现 `offlinePushCertificateID` 协议方法返回证书 ID 即可。

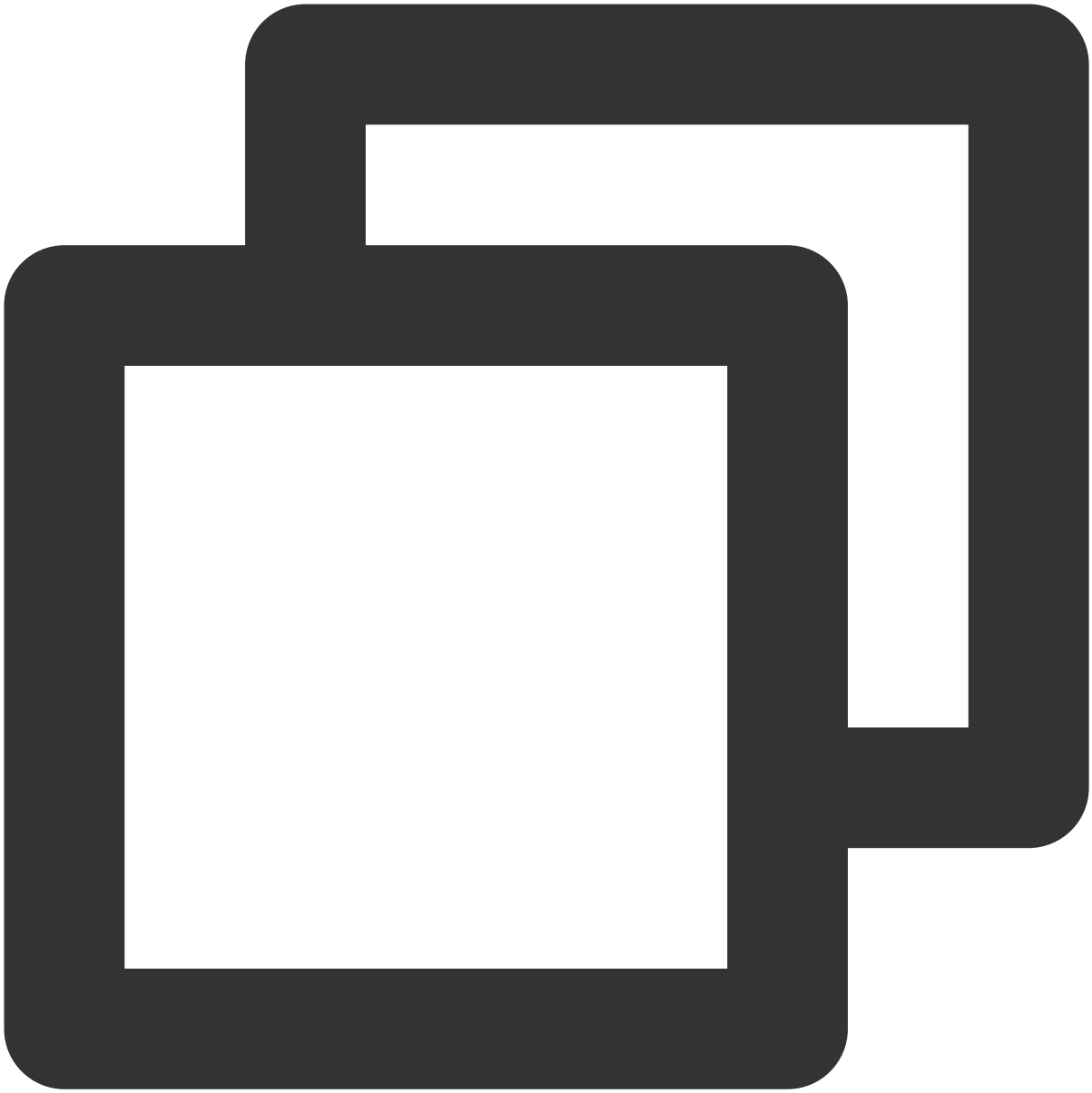
Swift

Objective-C



```
import TIMPush

func offlinePushCertificateID() -> Int32 {
    return kAPNSBusiId
}
```



```
#pragma mark - TIMPush

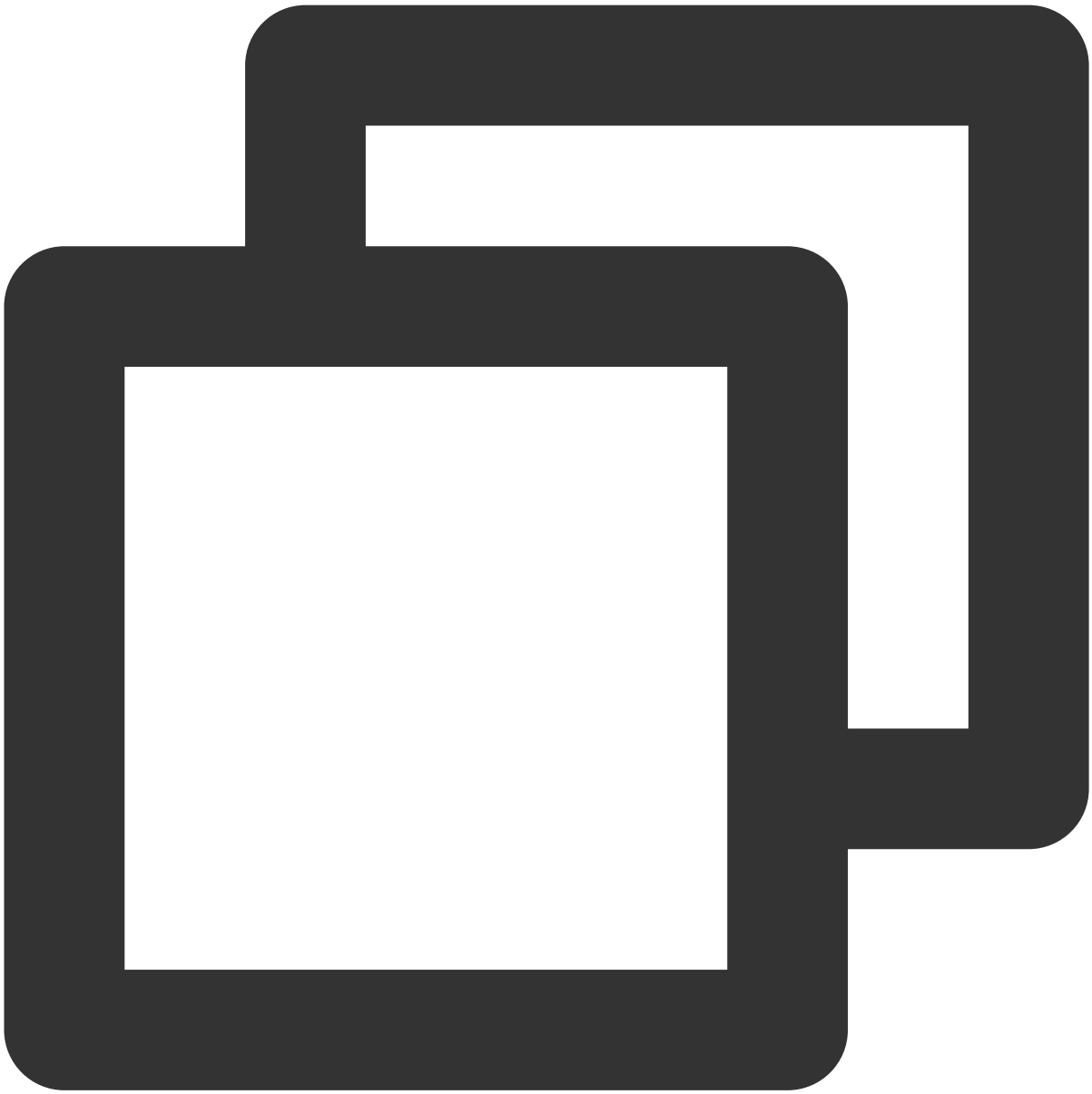
- (int)offlinePushCertificateID {
    return kAPNSBusiId;
}
```

步骤8：拨打离线推送电话

请在调用 call 或者 groupCall 拨打电话的时候设置 params 的 offlinePushInfo 字段。

Swift

Objective-C



```
import TUICallKit_Swift
import TUICallEngine

let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "您有一个新的通话"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
```

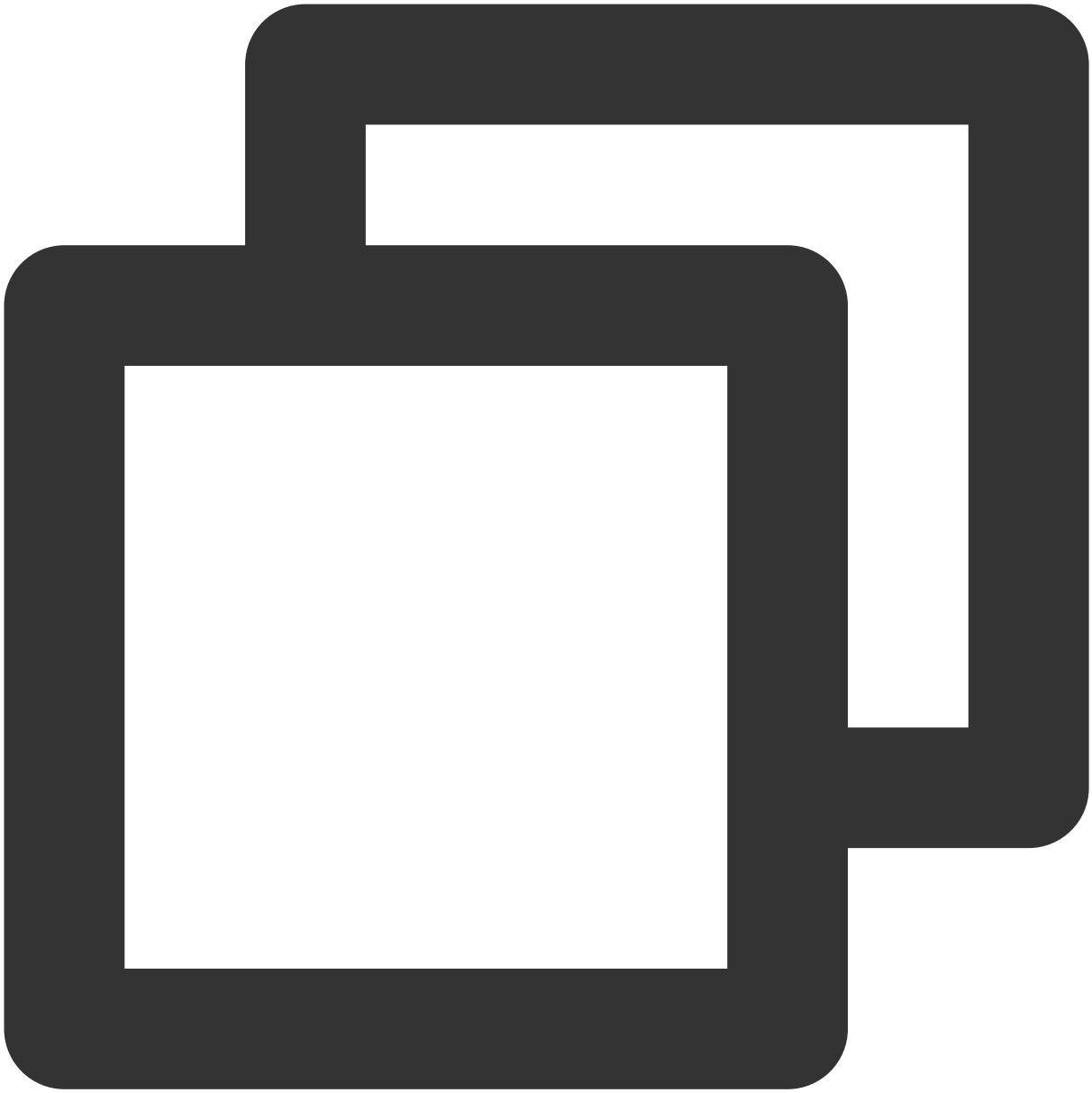
```
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be t
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
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"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

// 仅以单人通话举例、群通话类似
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (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";
    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;
}

// 仅以单人通话举例、群通话类试
[[TUICallKit createInstance] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

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

}];
```

常见问题

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。

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。

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。

Android

最近更新时间：2024-07-24 10:20:03

TIMPush 插件是一款功能强大的付费插件，它为 Google 平台带来了 VoIP calls 的通知机制，结合 `Firebase Cloud Messaging (FCM)` 提供的数据消息能力以及 `TUICallKit` 组件，能够实现具有自定义布局的来电显示界面。

说明：

拥有 Google Mobile Services 的手机。

FCM 数据消息功能仅在TIMPush **7.9.5668**及更高版本中提供。

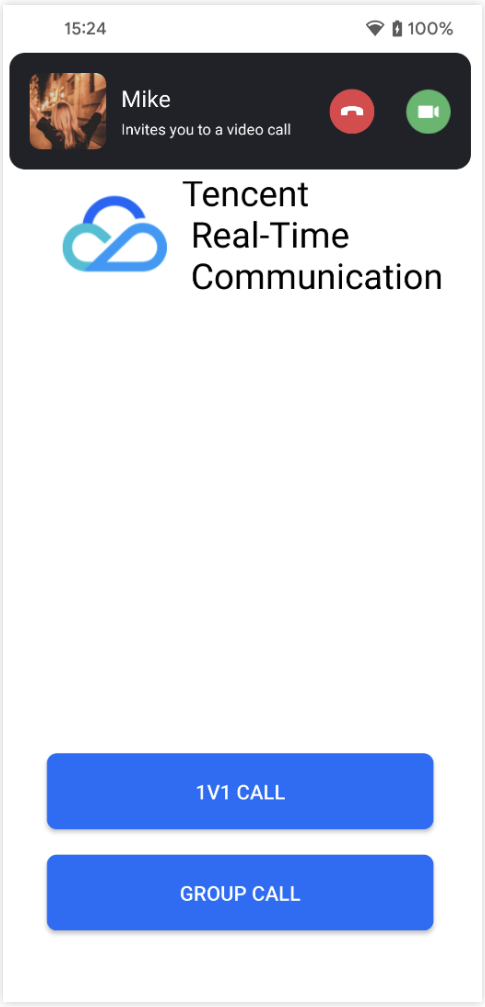
TUICallKit 的支持仅限于 **2.3** 版本及以上。

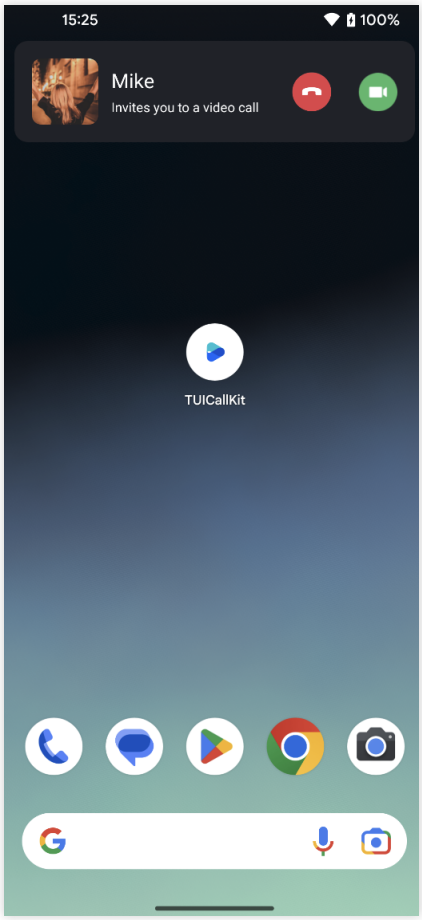
本文详细介绍如何在 TUICallKit 组件中接入 `TIMPush` 插件，使用 FCM 的数据消息能力，实现音视频通话的横幅来电显示效果。

集成效果

TUICallKit 在 GitHub 示例工程上已成功集成了 FCM 数据消息推送，您可以通过参考 Call UIKit 示例工程 来快速实现功能的正常运行。以下示意图展示了 TUICallKit 含 UI 方案集成 FCM 数据消息后，当被叫端的**应用在前台、后台**或者**应用进程不存在**时，被叫端收到来电邀请后的效果。

应用在**前台时**的显示效果	应用在**后台时**或者**离线时**的显示效果





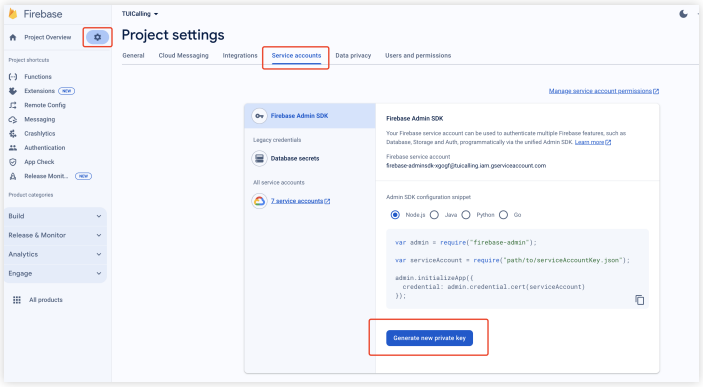
说明：

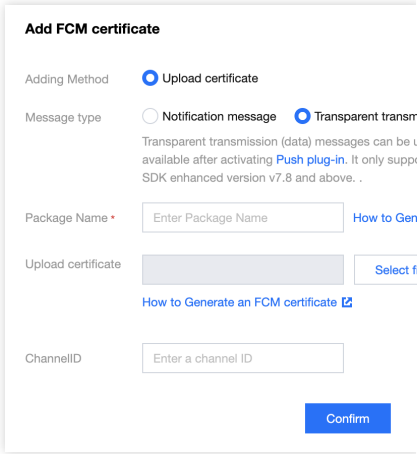
为实现上图良好的通话体验，建议您在应用中开启“通知”权限、“显示在其他应用上层”以及“后台拉起界面”权限，具体方法见：相关权限开启。

准备条件

1. 将您自己的应用注册到 FCM推送平台，得到 **AppID** 和 **AppKey** 等参数以及 `google-services.json` 文件，来实现离线推送功能。
2. 登录Chat 控制台，在**推送管理 > 接入设置**功能栏，选择 FCM，添加 FCM 的证书，其中，消息类型选择**透传（数据）消息**。

厂商推送平台	Chat 控制台配置

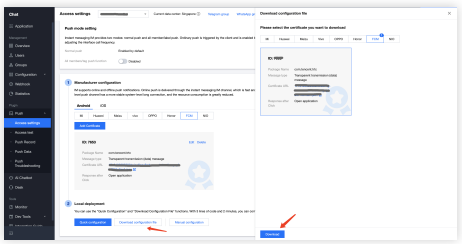
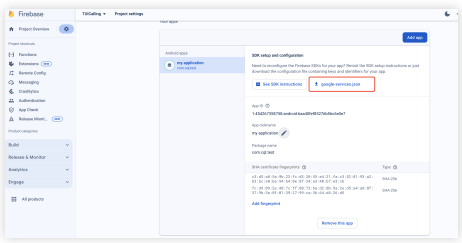




快速接入

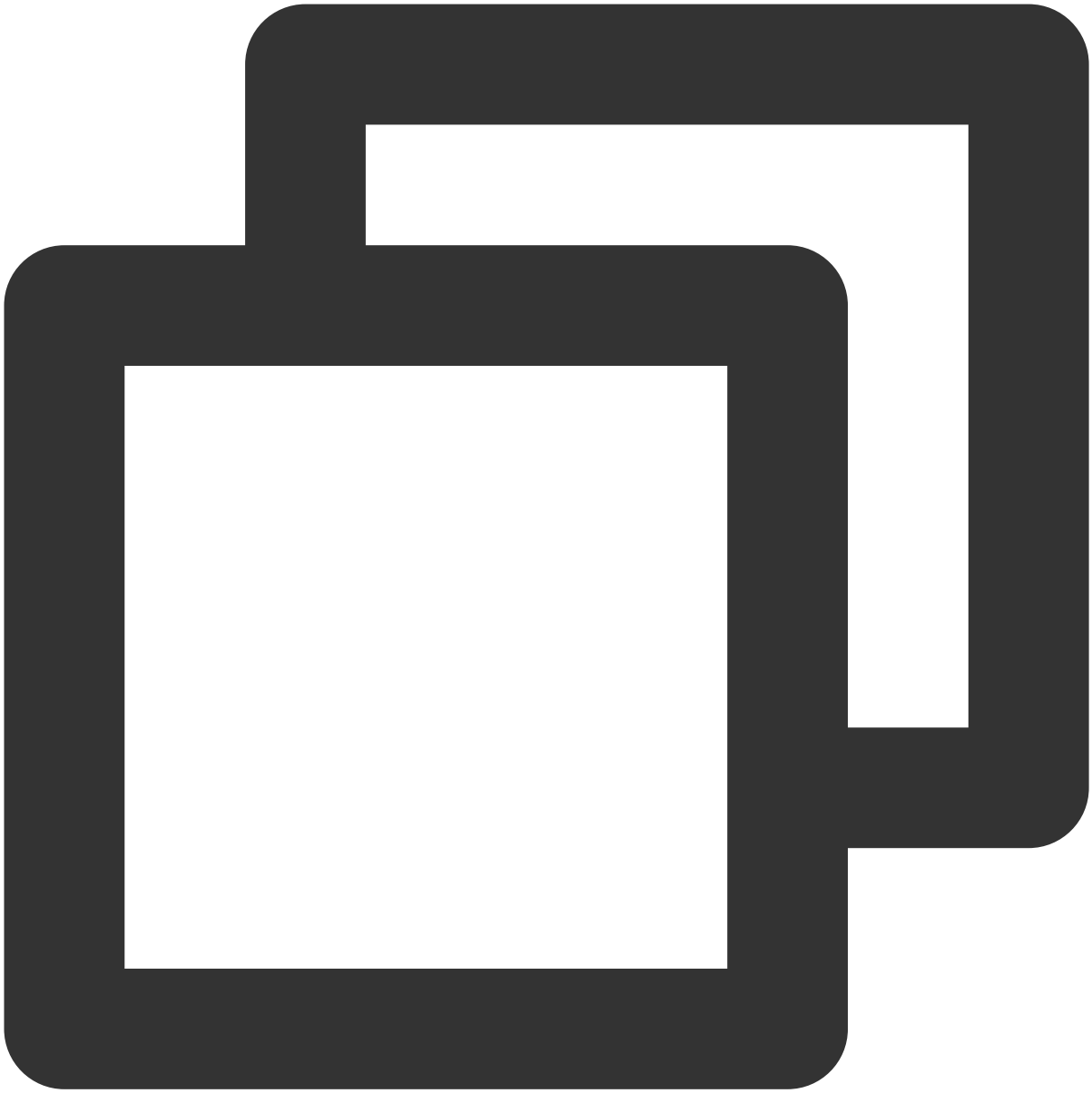
1. 下载并添加配置文件

完成控制台厂商推送信息填写后，下载并添加配置文件到工程。将下载的 `timpush-configs.json` 文件添加到应用模块的 `assets` 目录下，将 `google-services.json` 添加到工程app目录下。

下载文件 timpush-configs.json	下载文件 google-services.json	添加到
		

2. 集成 TIMPush 插件

在项目的 `app/build.gradle` 文件中添加如下依赖：



```
implementation "com.tencent.timpush:timpush:latest.release"
implementation "com.tencent.timpush:fcm:latest.release"
```

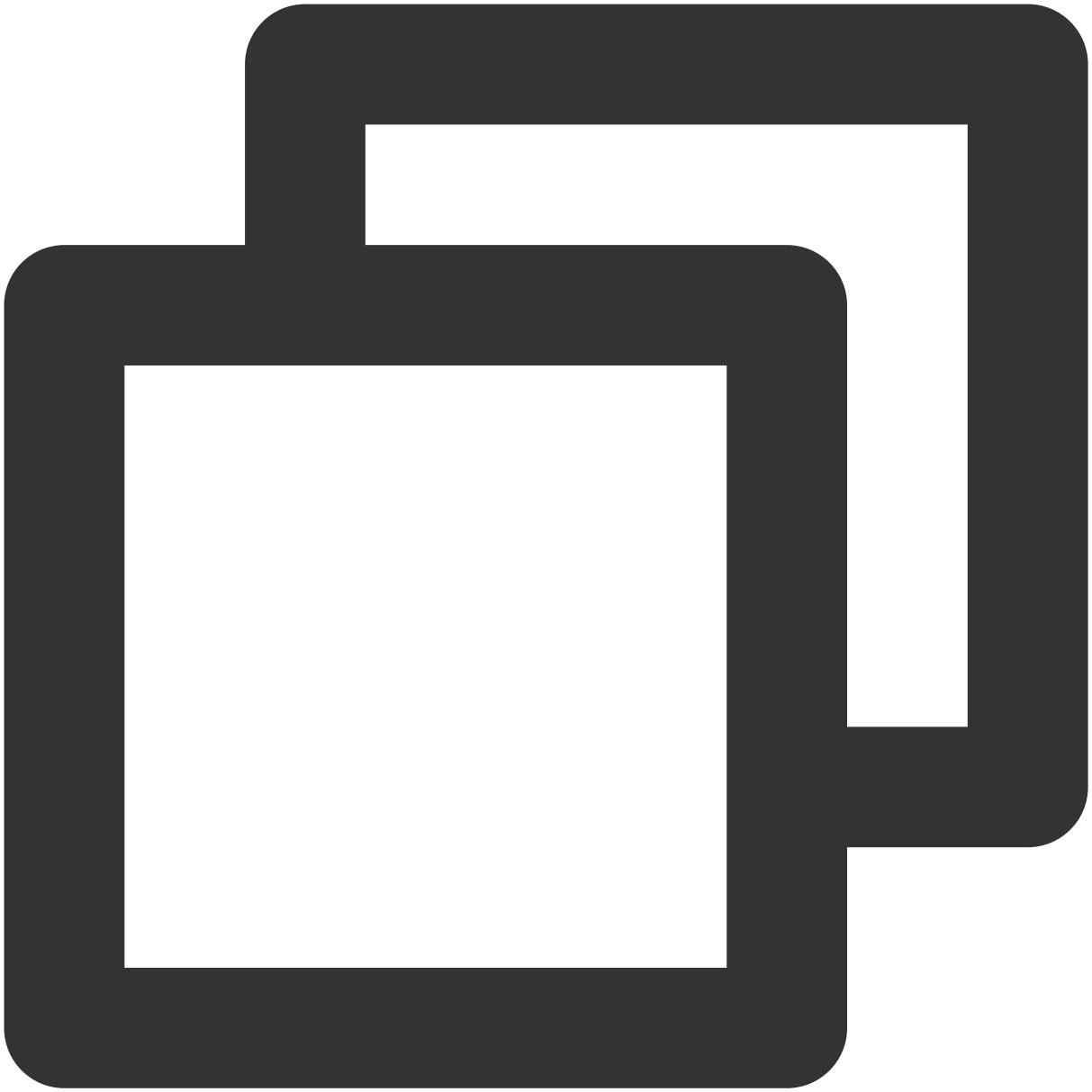
说明：

TIMPush 需要集成 IMSDK 在 **7.9.5668** 版本及以上。

您可以在 `tuicallkit-kt/build.gradle` 文件中修改 Chat SDK 的版本号。

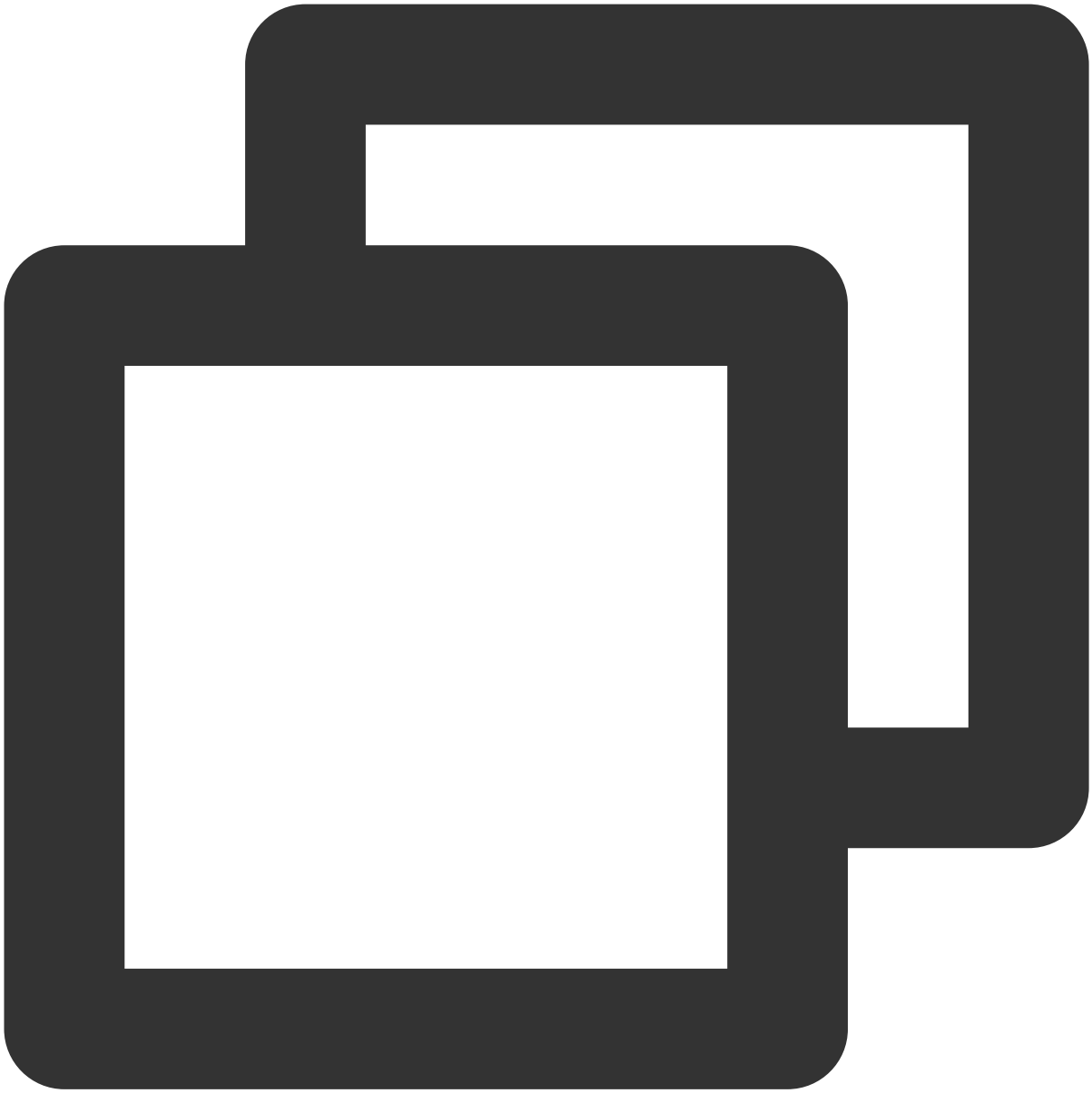
3. 完成工程配置

在项目级 `build.gradle` 文件的 `buildscript -> dependencies` 下添加以下配置。



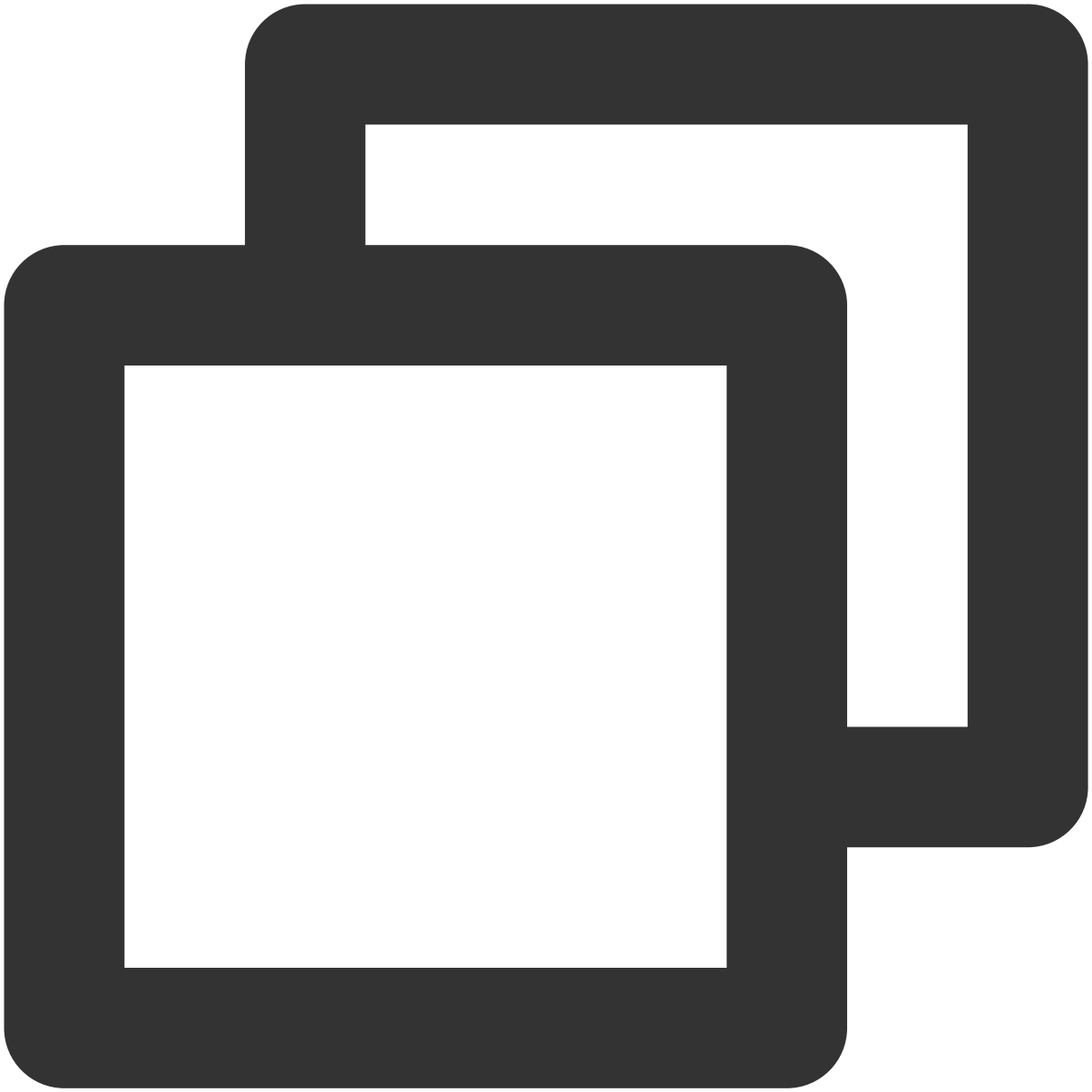
```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.3.15'
    }
}
```

在 `app/build.gradle` 文件中添加下方配置。



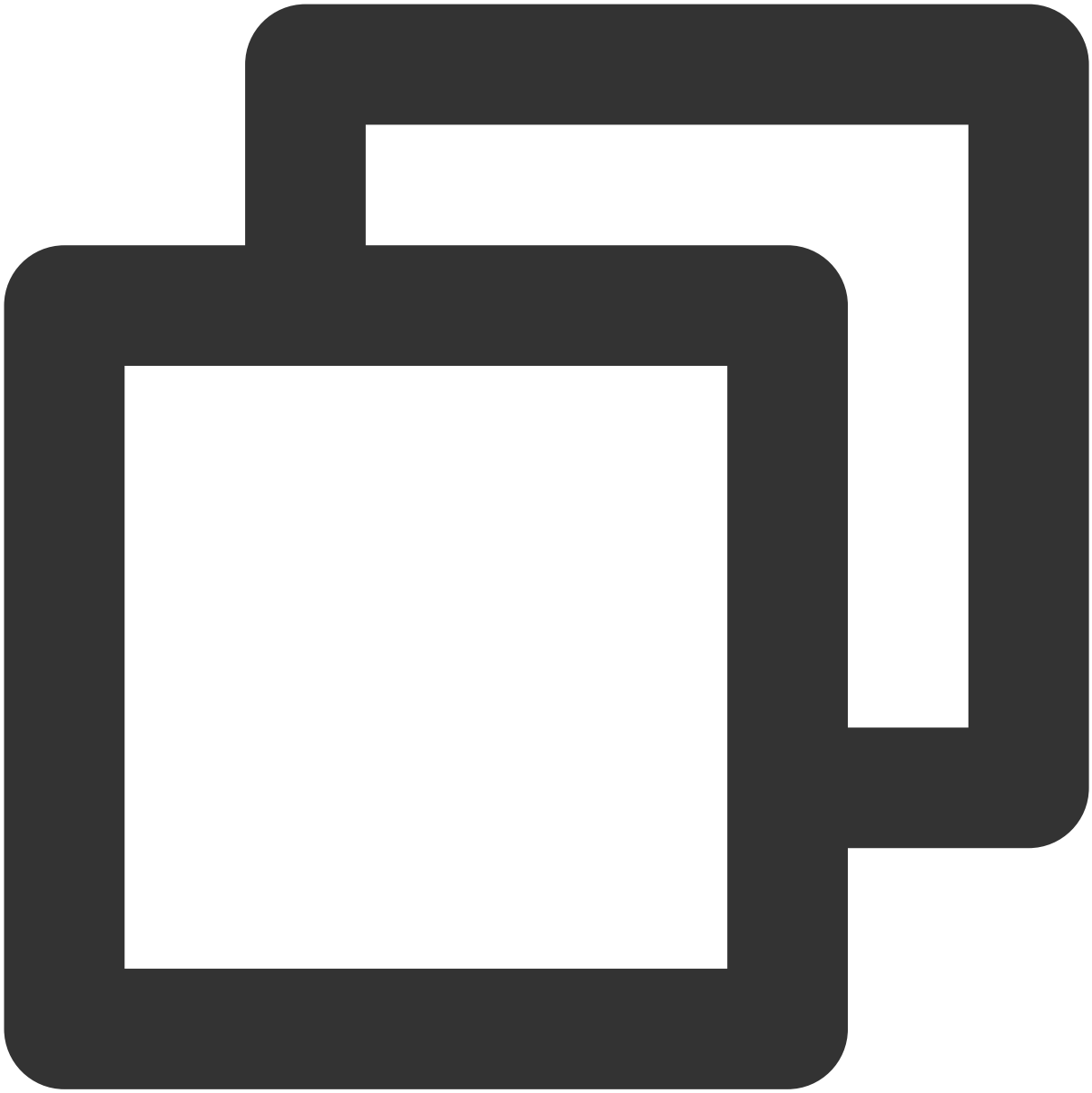
```
apply plugin: 'com.google.gms.google-services'
```

在 `app/proguard-rules.pro` 文件中，将 TIMPush 相关类加入不混淆名单。



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

在 `app/build.gradle` 文件中，将应用包名修改为您的实际应用包名。



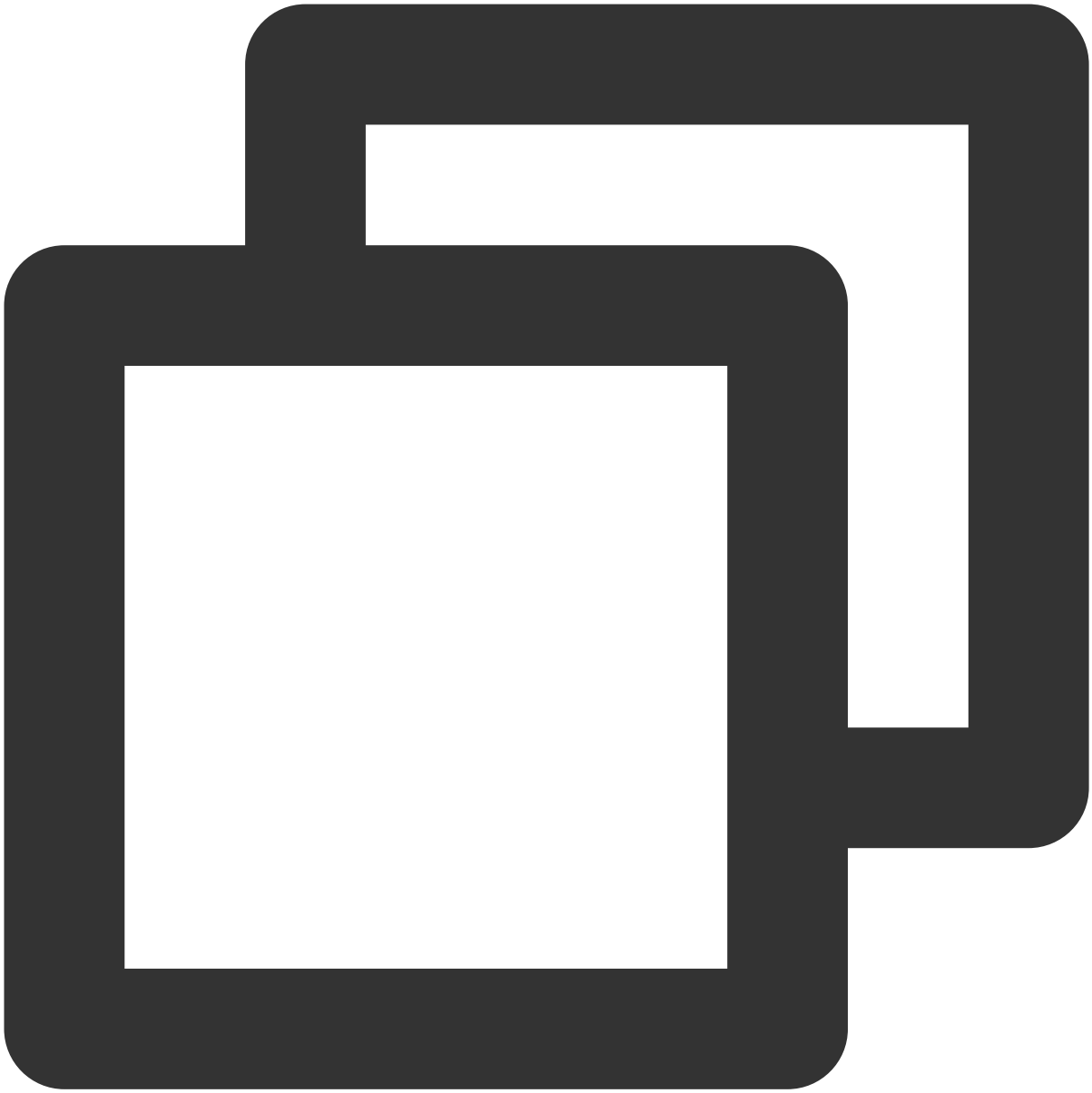
```
applicationId 'com.****.callkit'
```

4. 实现自动登录

在应用的 `application` 类中，监听 TIMPush 发出的事件通知，实现自动登录。

Kotlin

Java

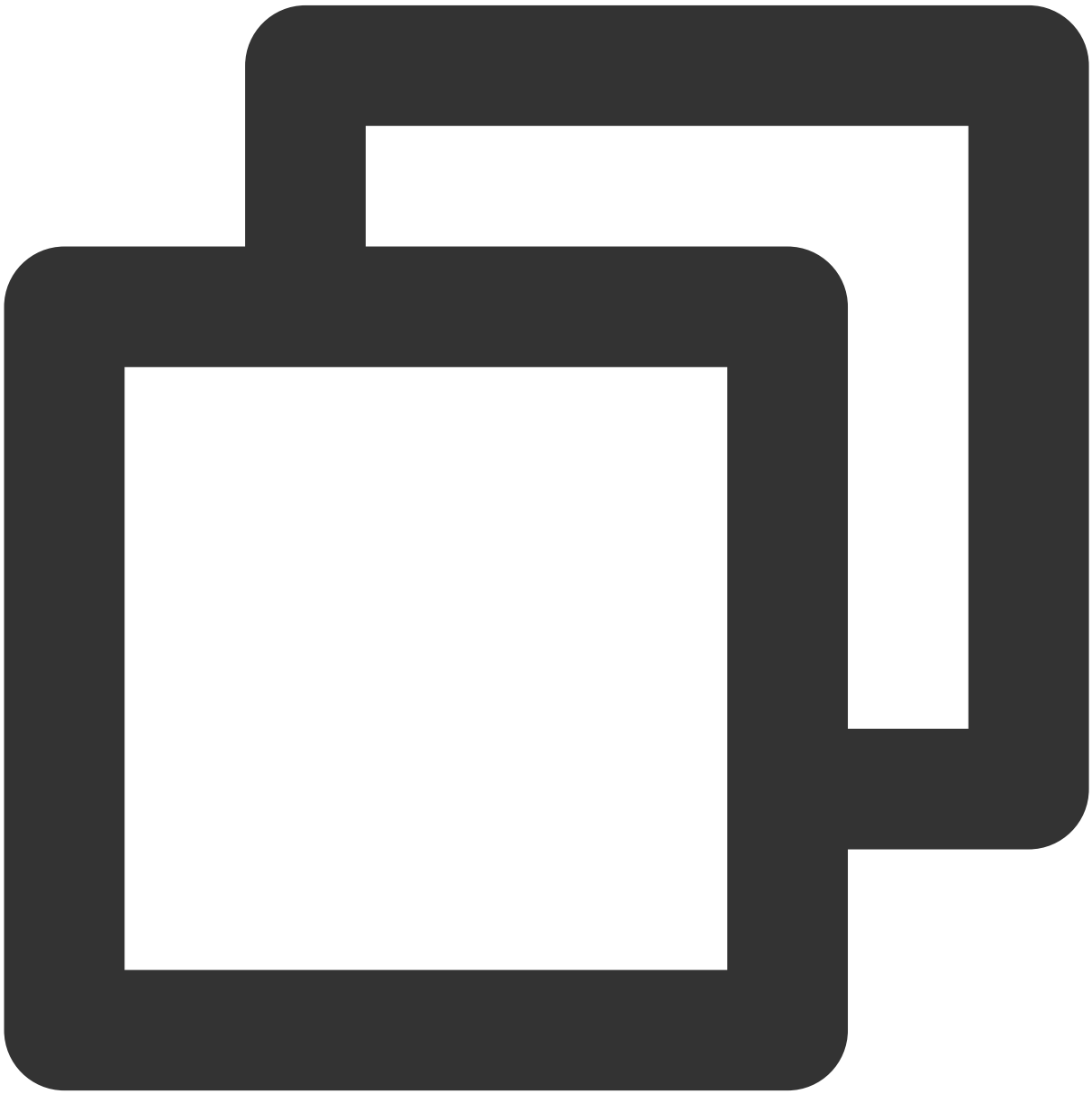


```
import com.tencent.qcloud.tuicore.TUIConstants
import com.tencent.qcloud.tuicore.TUICore
import com.tencent.qcloud.tuicore.interfaces.ITUINotification

class BaseApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        TUICore.registerEvent(TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_
            TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY) { key, su
```

```
            if (TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KEY == key
                && TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY ==
                //you need to login again to launch call activity, please implement
                autoLogin()
            }
        }
    }
}
```



```
import com.tencent.qcloud.tuicore.TUIConstants;
import com.tencent.qcloud.tuicore.TUICore;
import com.tencent.qcloud.tuicore.interfaces.ITUINotification;

public class BaseApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        TUICore.registerEvent(TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_
                TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY, new I
                    @Override
                    public void onNotifyEvent(String key, String subKey, Map<String
                        if (TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KE
                                && TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WA
                            //you need to login again to launch call activity, plea
                            autoLogin();
                        }
                    }
                });
    }
}
```

完成上述步骤后，您可以结合 TUICallKit 使用 TIMPush 的离线推送能力。

说明：

如果您的 Android 程序在接收推送或拉起页面时遇到问题，您可以参见 被叫端来电显示策略 对问题进行排查。

高级功能

自定义铃声

如果您想自定义该铃声，可以替换 `tuicallkit-kt/src/main/res/raw/phone_ringing.mp3` 文件。

注意：

替换掉该铃声后，不论应用在前台、后台或者离线，铃声都会是替换后的铃声。

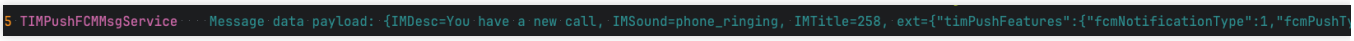
替换后其他厂商手机收到邀请后的铃声也会是替换后的铃声。

常见问题

1. 应用杀死后无法弹出来电UI

确认收到了推送，收不到推送需要确认下 Chat 控制台是否正确上传证书。参见文档上述**快速接入**的第一步，看下是否添加正确。

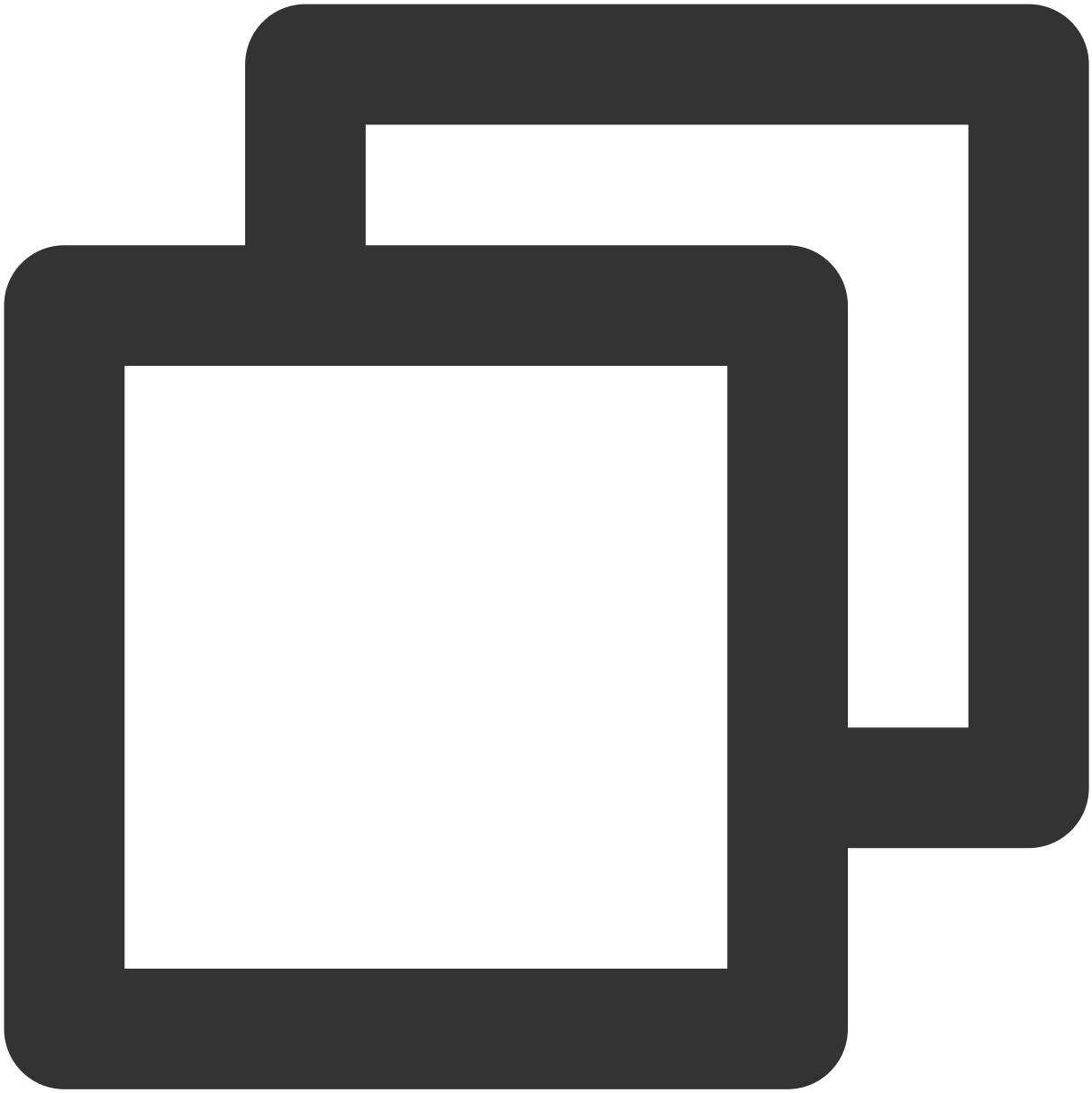
确认控制台选择了FCM 数据消息，对照上述**准备条件**的第二步；
确认收到数据消息，过滤日志（关键字：TIMPush），检查下述日志有打印；



确认实现了**自动登录**。自动登录后才会去拉通话请求，才能显示来电UI。

2. 相关权限开启

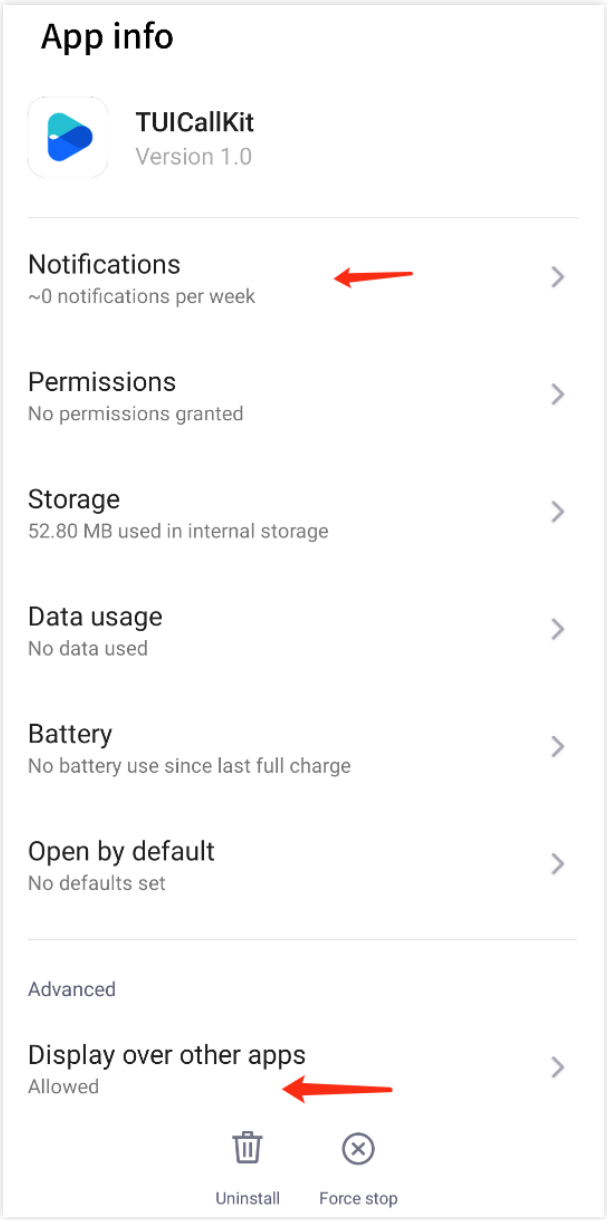
为实现良好的通话体验，建议您在应用中开启“通知”权限、“显示在其他应用上层（悬浮窗）”以及“后台拉起界面”权限，具体方法如下：
代码指引
手动开启
通知权限：便于展示推送通知：请参考通知运行时权限 和 请求运行时权限 根据业务需求自行实现。
悬浮窗权限：用于展示自定义的来电通知，以及通话悬浮窗。
后台拉起界面权限：用于当应用在后台时拉起界面（例如：VIVO手机）。

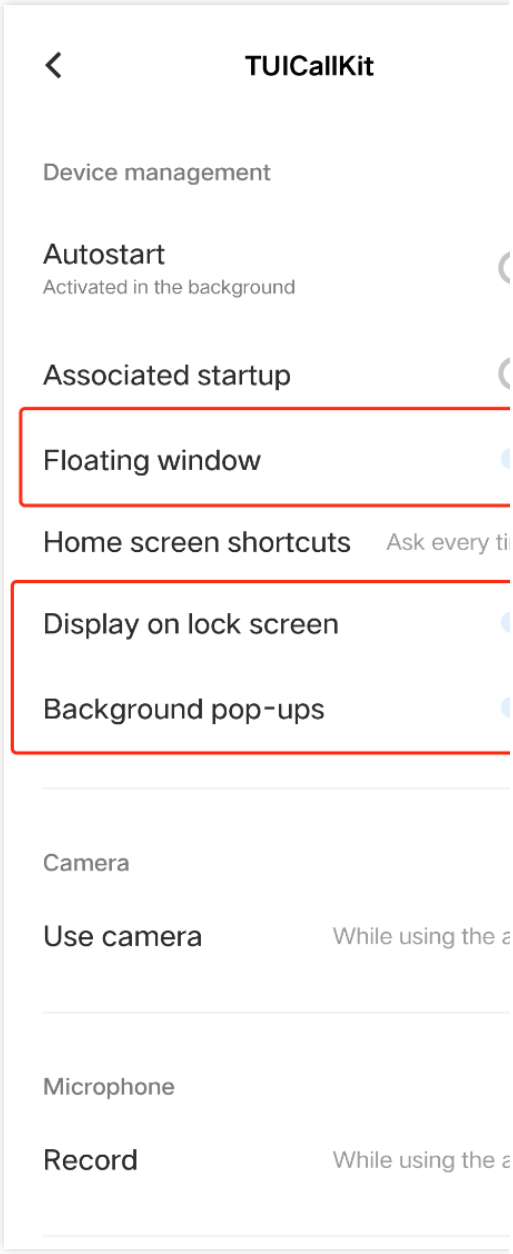


```
fun requestPermission(context: Context?) {
    //In TUICallKit,Please open both OverlayWindows and Background pop-ups permissi
    PermissionRequester.newInstance(
         PermissionRequester.FLOAT_PERMISSION, PermissionRequester.BG_START_PERMISS
        .request()
}
```

安装应用后，您可以长按应用图标，选择“应用信息”，然后开启“通知”权限、“显示在其他应用上层”以及“后台拉起界面”权限。或者您可以到 `手机 > 系统设置 > 应用管理 > 应用` 中手动开启上述权限。

Pixel 4a	VIVO





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 Notification

最近更新时间：2024-07-22 17:00:45

为了帮助开发者在 Flutter 项目中轻松实现离线推送功能，我们推荐使用 TIMPush 插件（付费）。与在 Android 和 iOS 端单独进行集成相比，采用 TIMPush 插件具有以下优势：

接入周期短，预计全厂商接入仅需要 30 分钟。

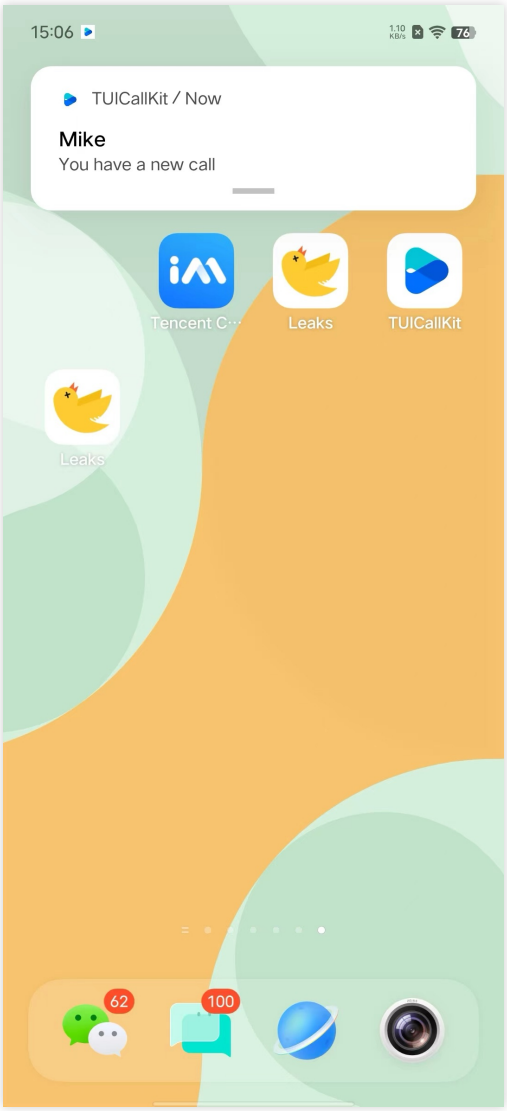
支持数据统计和链路追踪，方便您查看推送触达率、点击率和转化率等各类指标。

支持全员/标签推送，方便您将营销广告、通知、新闻资讯等内容推送给所有用户或指定群体。

支持 uni-app 和 Flutter 等跨平台框架。

本文档将详细介绍如何在 TUICallKit 组件中集成 TIMPush 插件，以实现音视频通话的离线推送能力。

Android	iOS

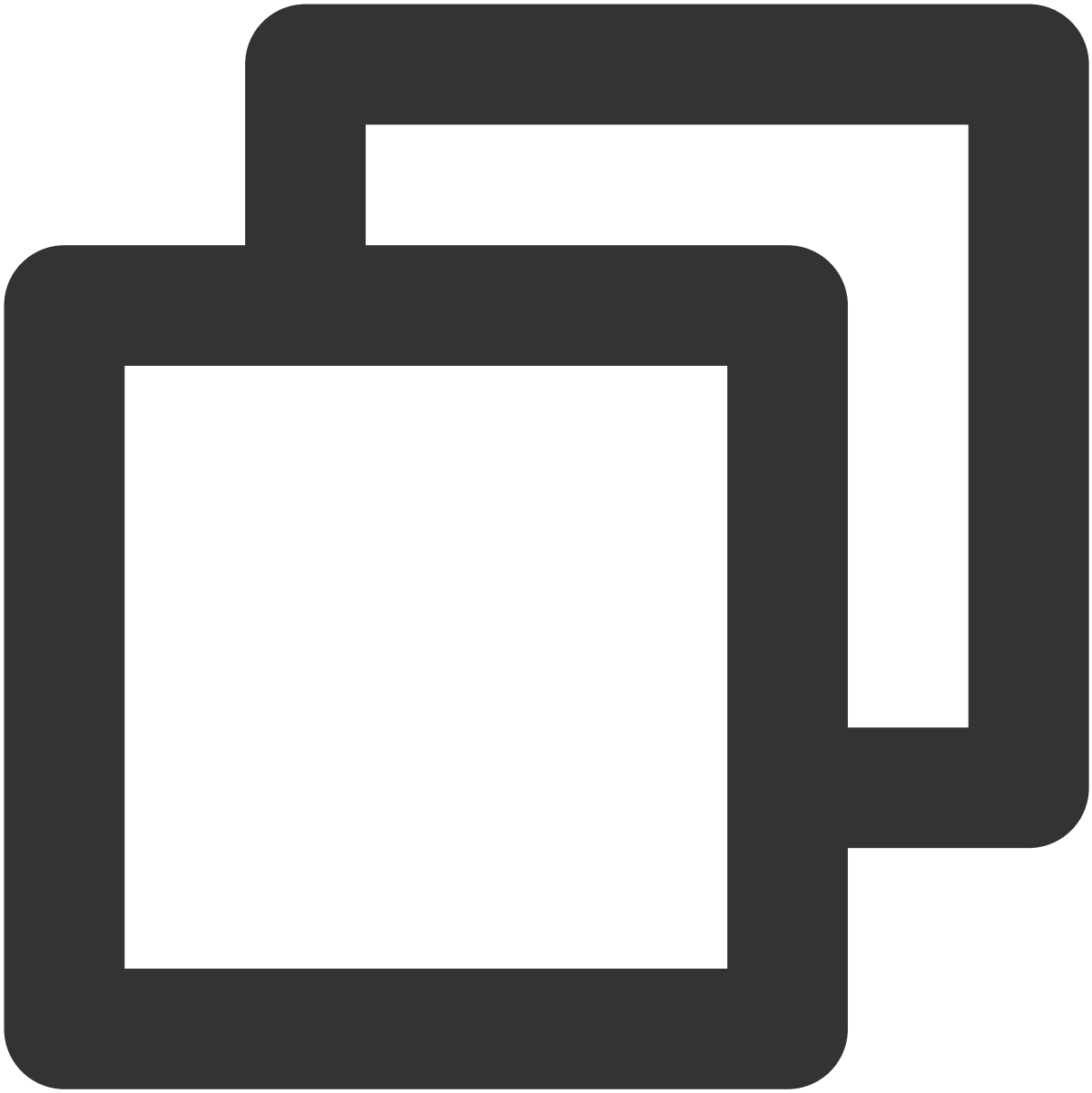




操作步骤

步骤1: 集成消息推送插件

本插件在 pub.dev 的包名为: tencent_cloud_chat_push , 您可以收到将其引入 pubspec.yaml 依赖目录中, 也可以执行下列命令, 自动安装。



```
flutter pub add tencent_cloud_chat_push
```

步骤2: 厂商配置

iOS

Android

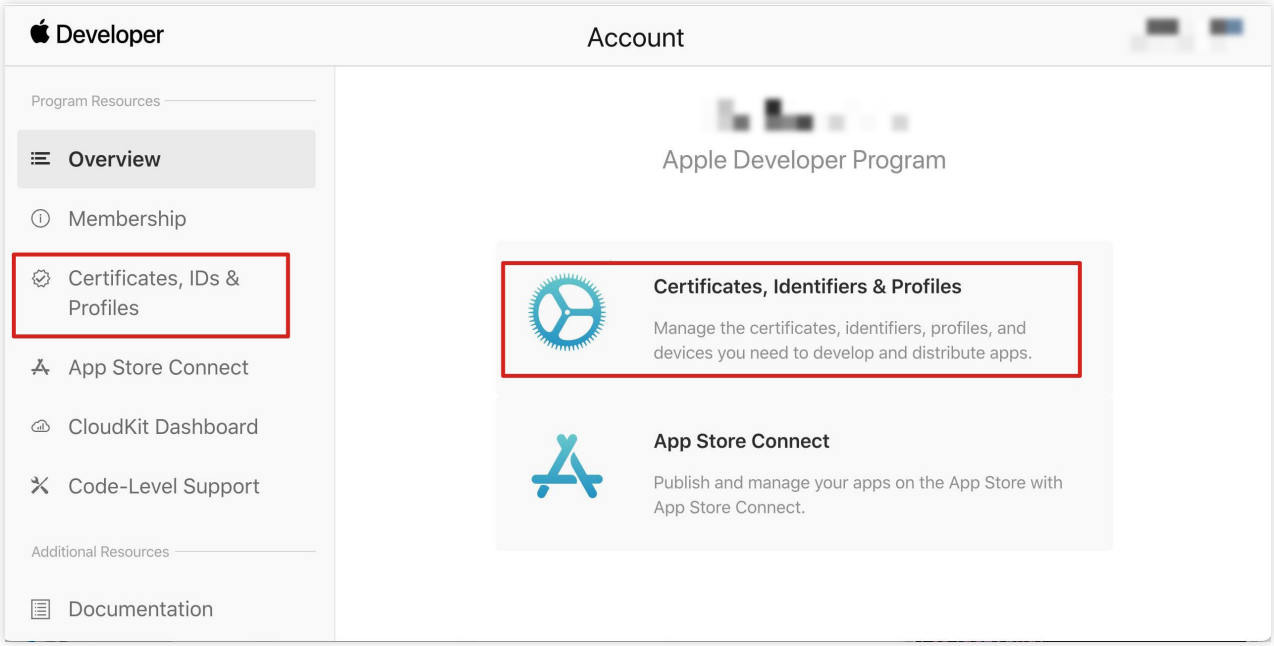
集成消息推送插件之前，需要先向 Apple 申请 APNs 推送证书，然后上传推送证书到 Tencent RTC 控制台 。

操作步骤

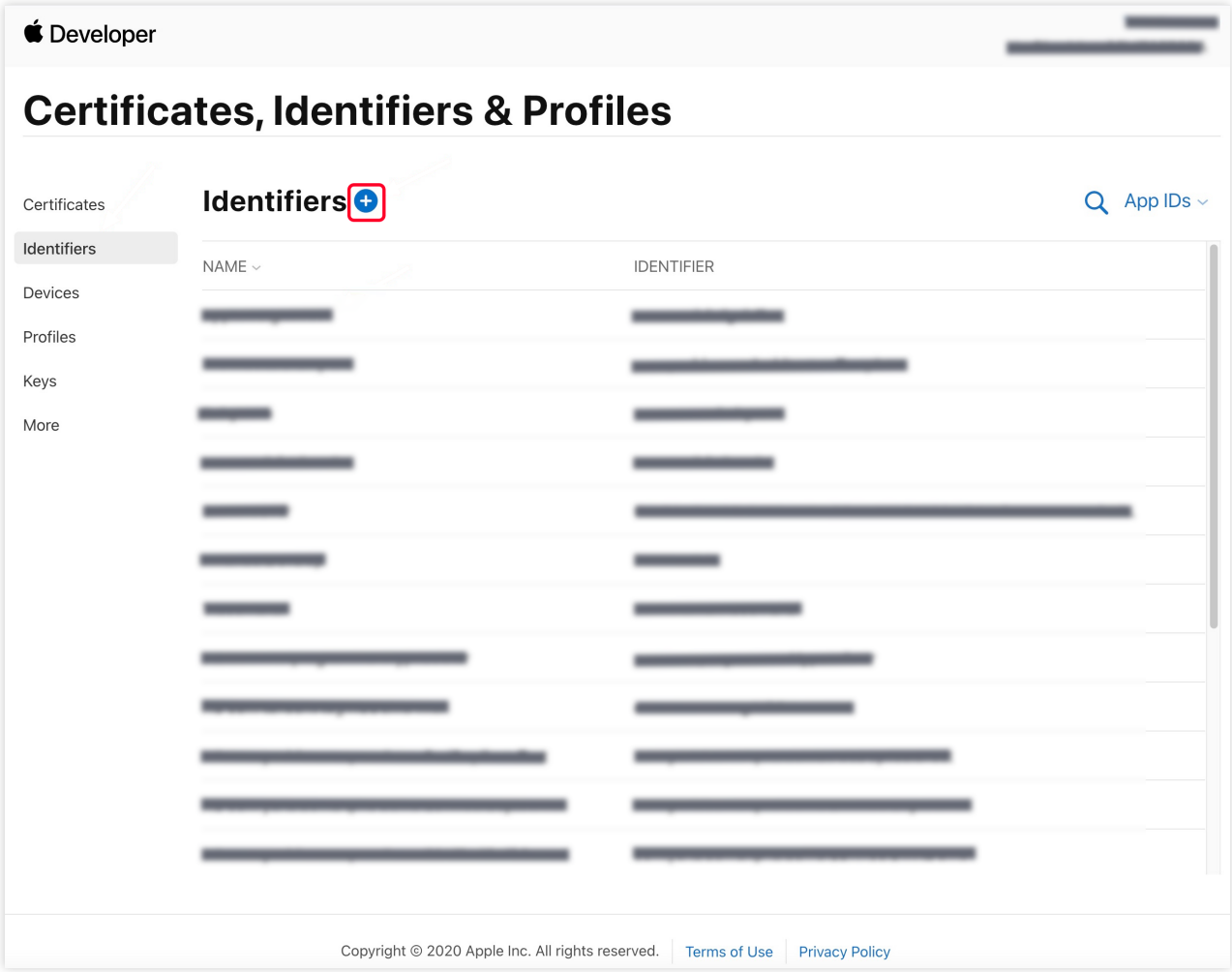
步骤1：申请 APNs 证书

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 **+**。

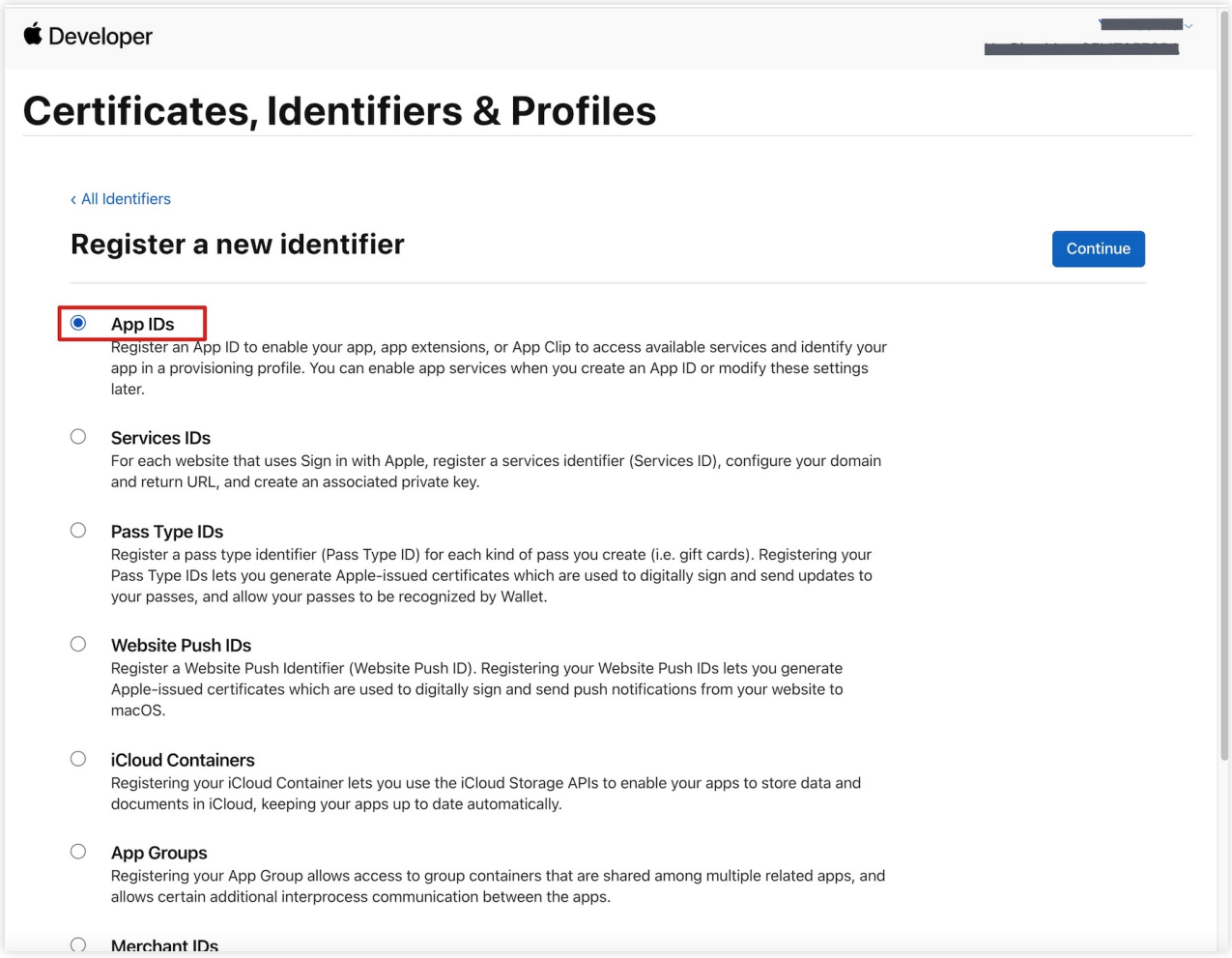


3. 您可以参见如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service`。

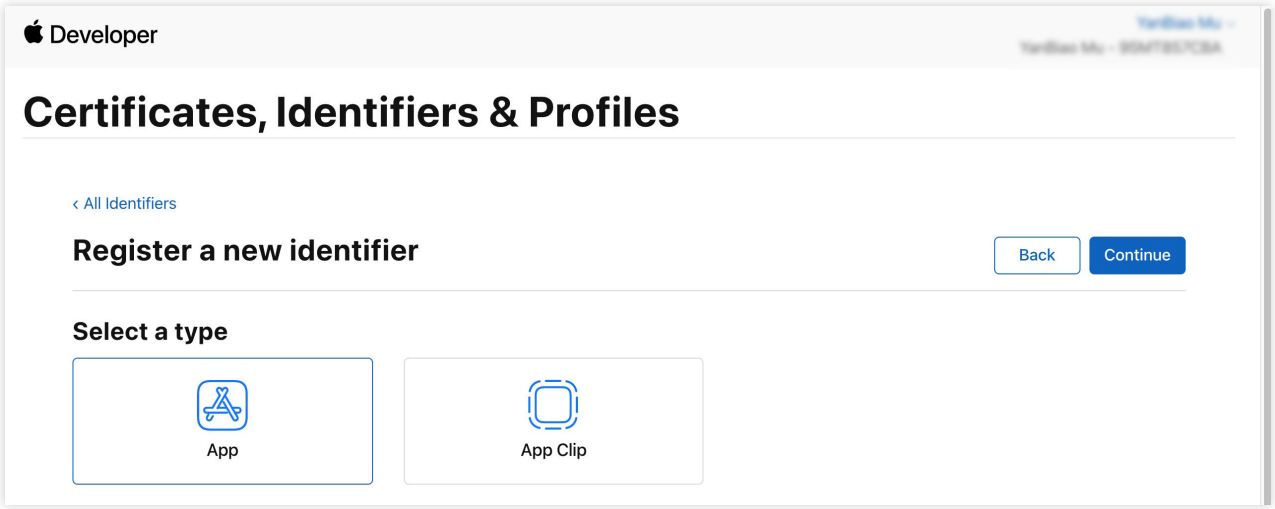
说明

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。

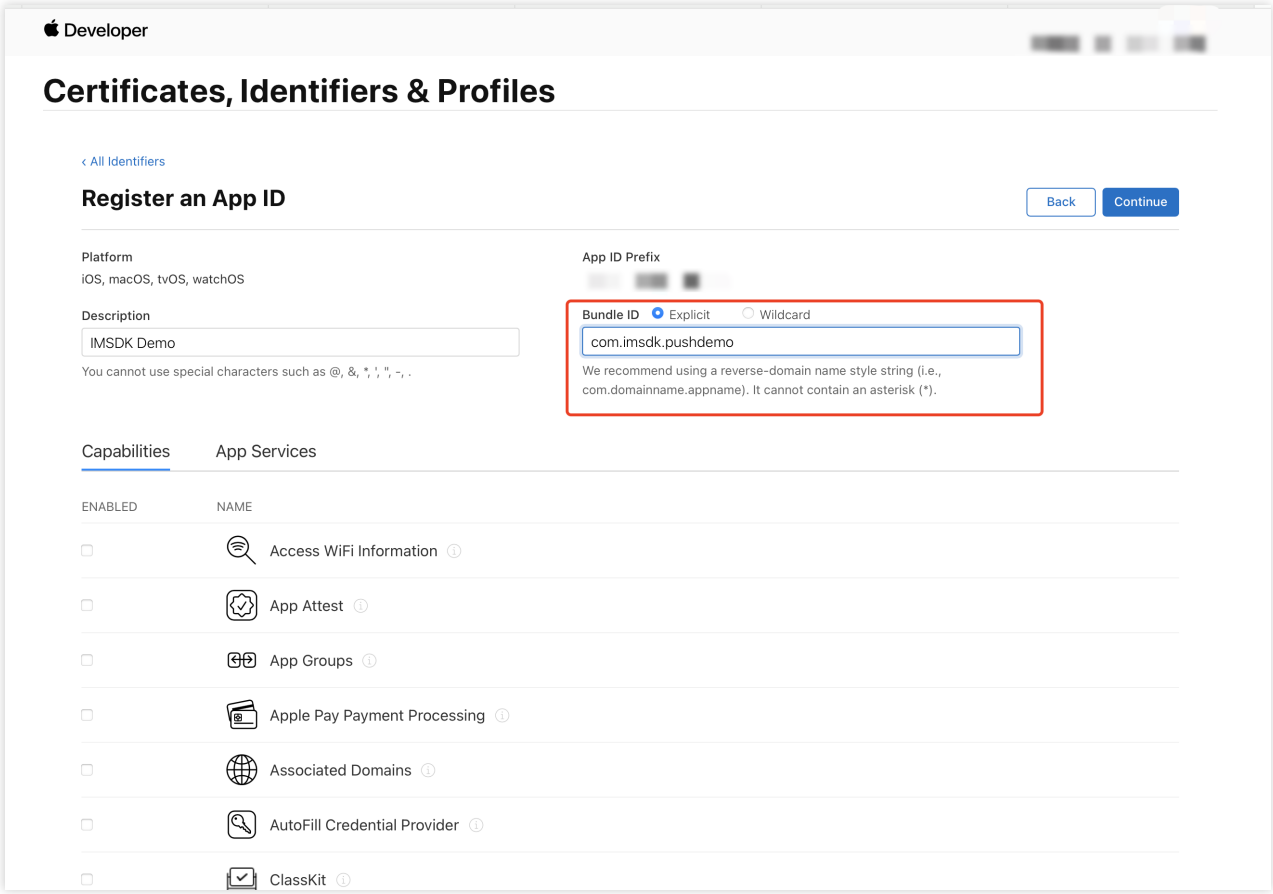
4. 勾选 **App IDs**，单击 **Continue** 进行下一步。



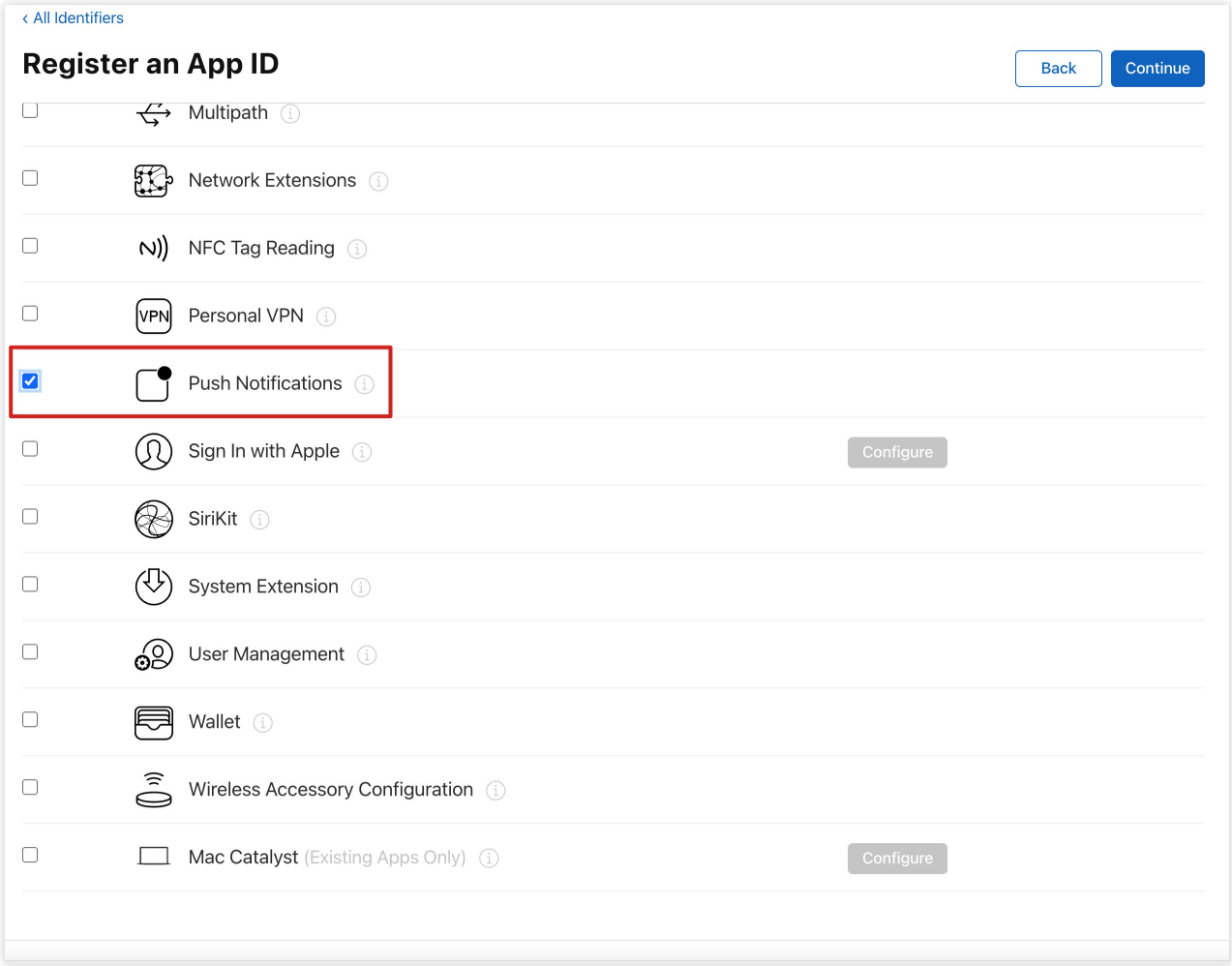
5. 选择 **App**，单击 **Continue** 进行下一步。



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

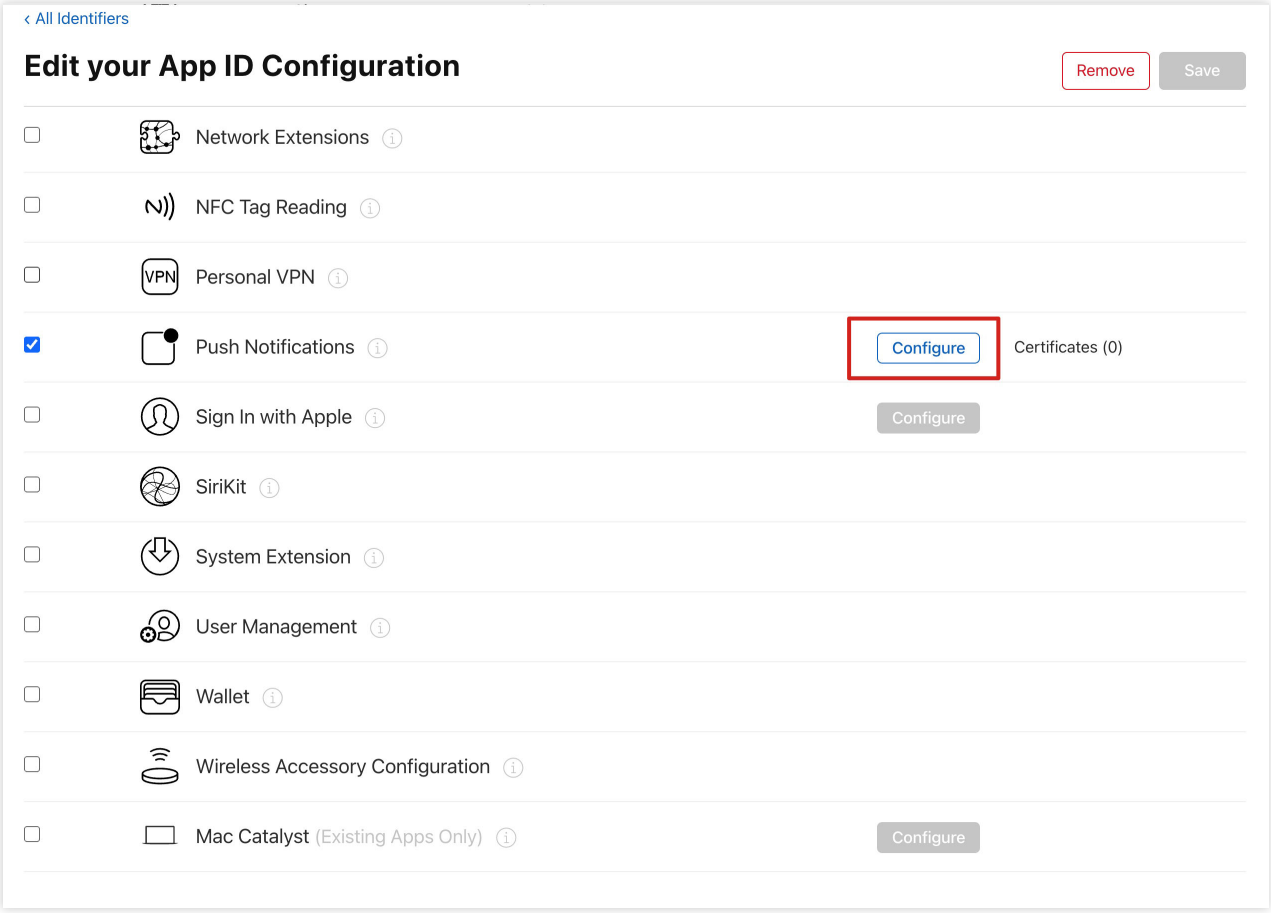


7. 勾选 **Push Notifications**，开启远程推送服务。

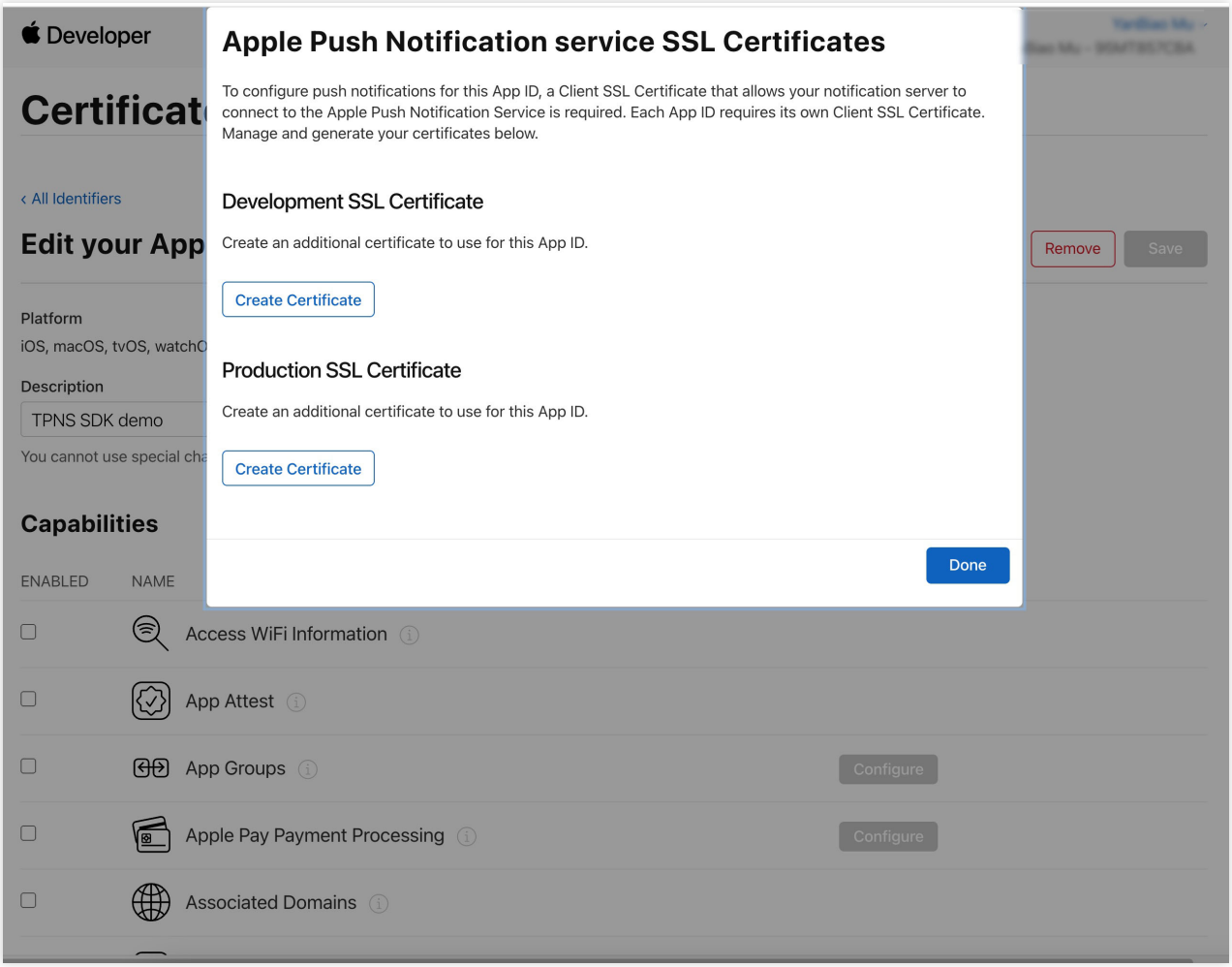


生成证书

1. 选中您的 AppID，选择 **Configure**。



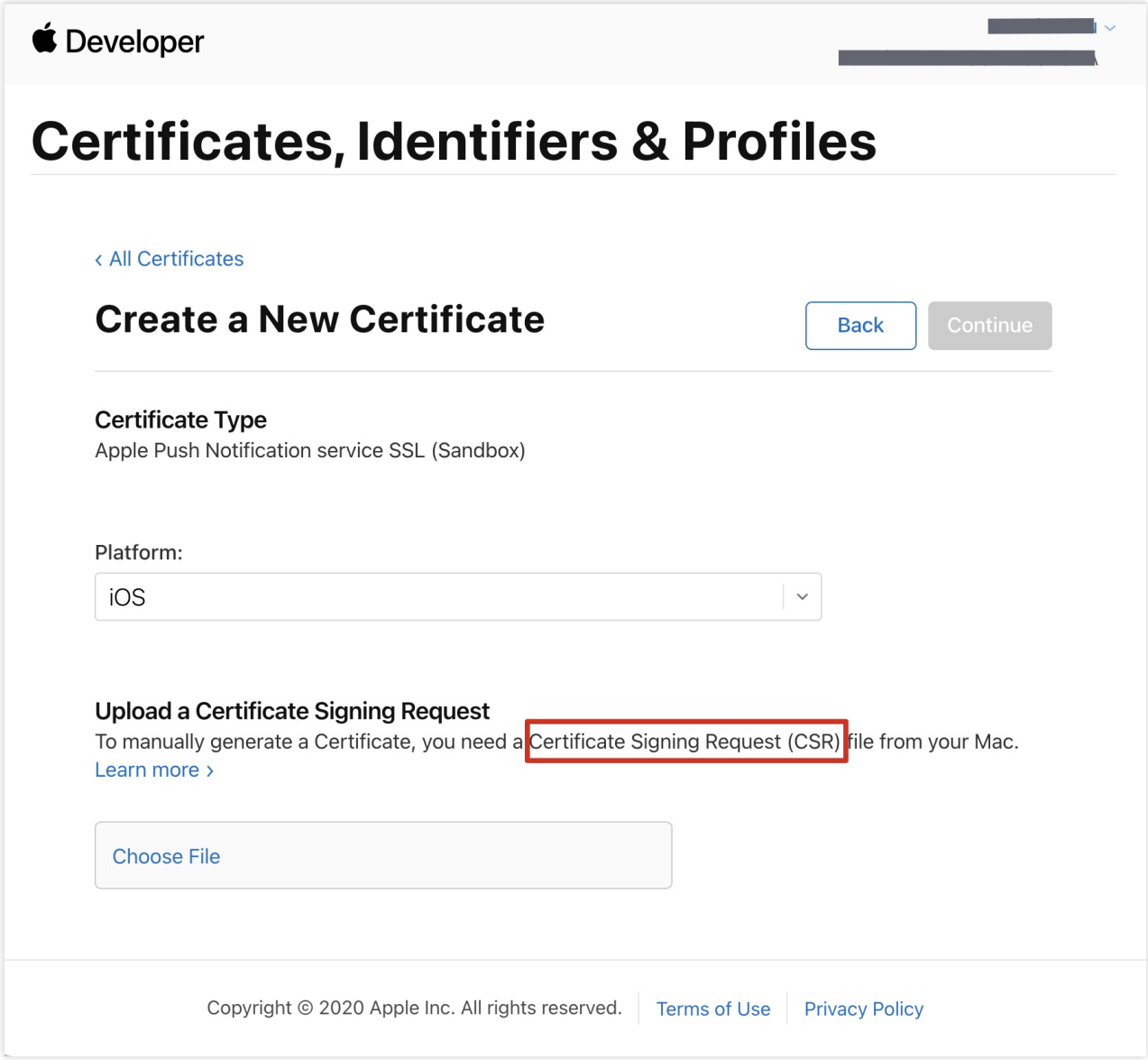
2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate`，分别用于开发环境（Development）和生产环境（Production）的远程推送证书，如下图所示：



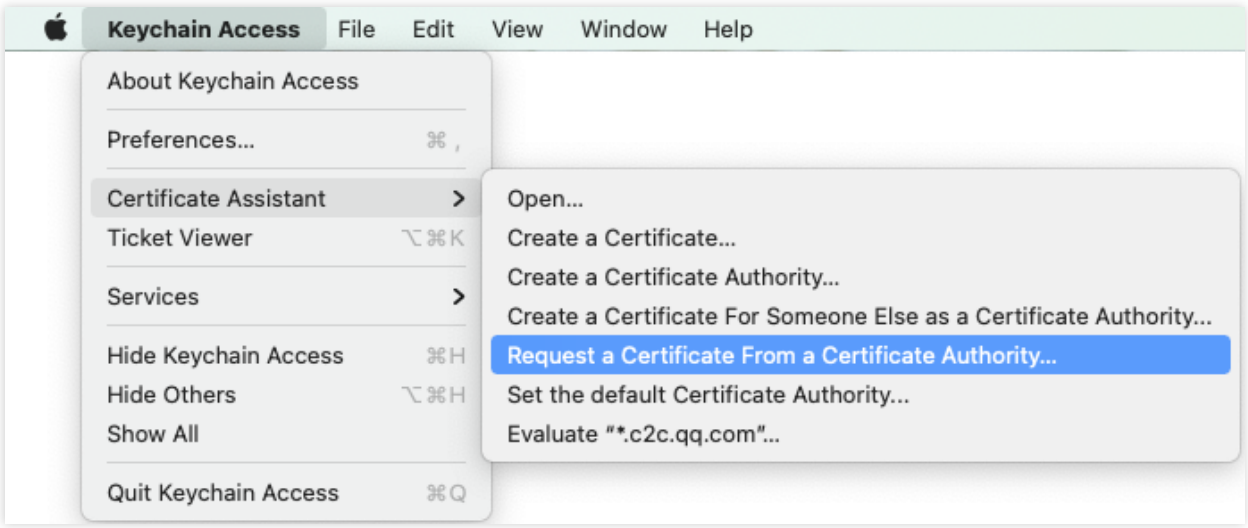
3.

我

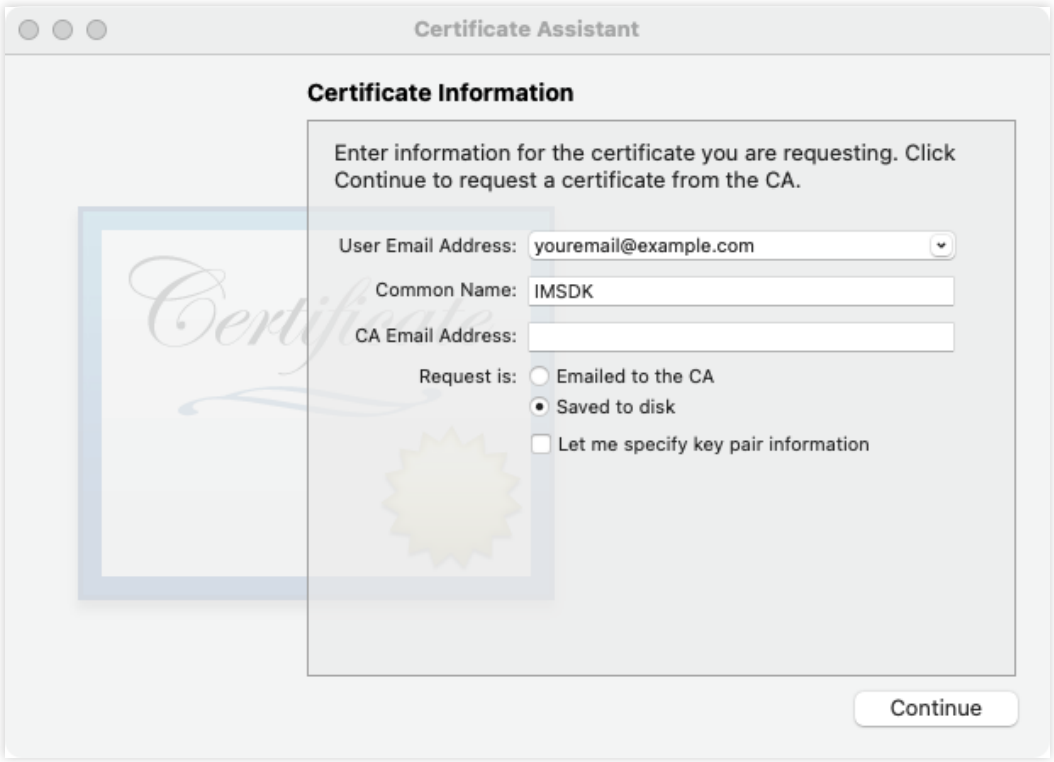
们先选择开发环境（Development）的 **Create Certificate**，系统将提示我们需要一个 Certificate Signing Request（CSR）。



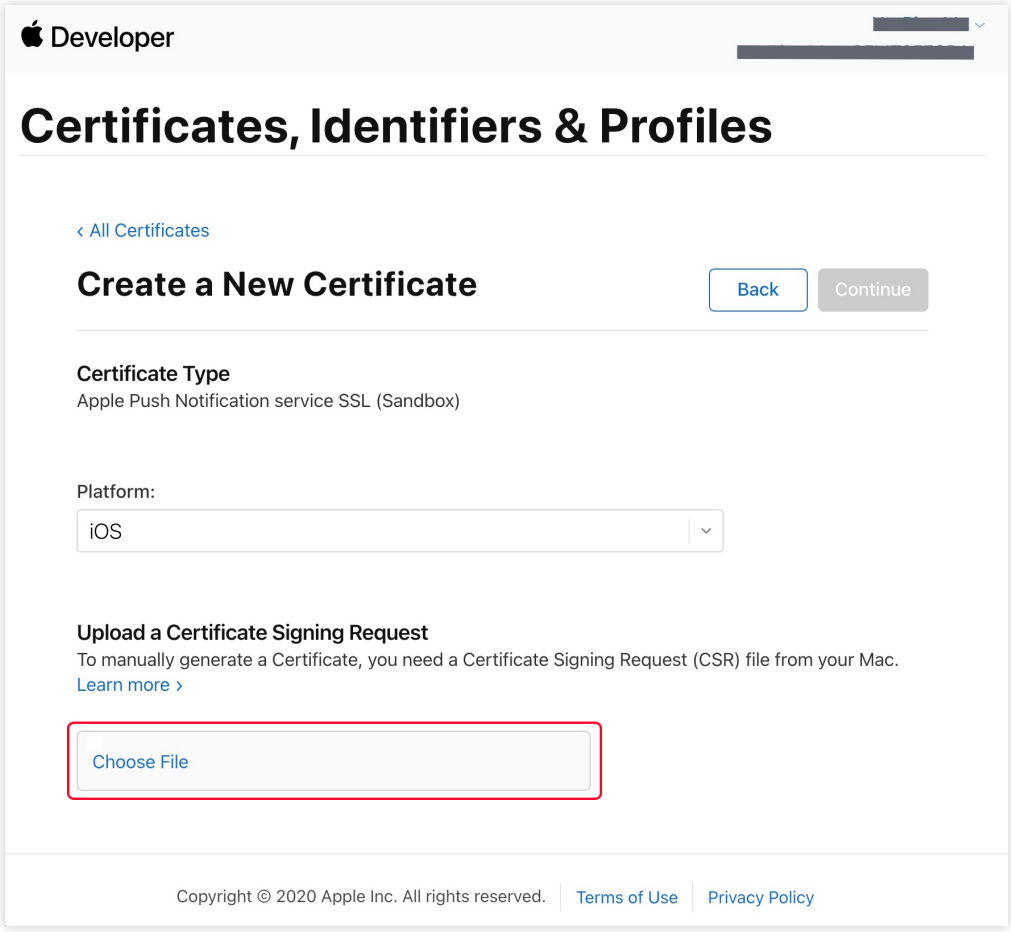
4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



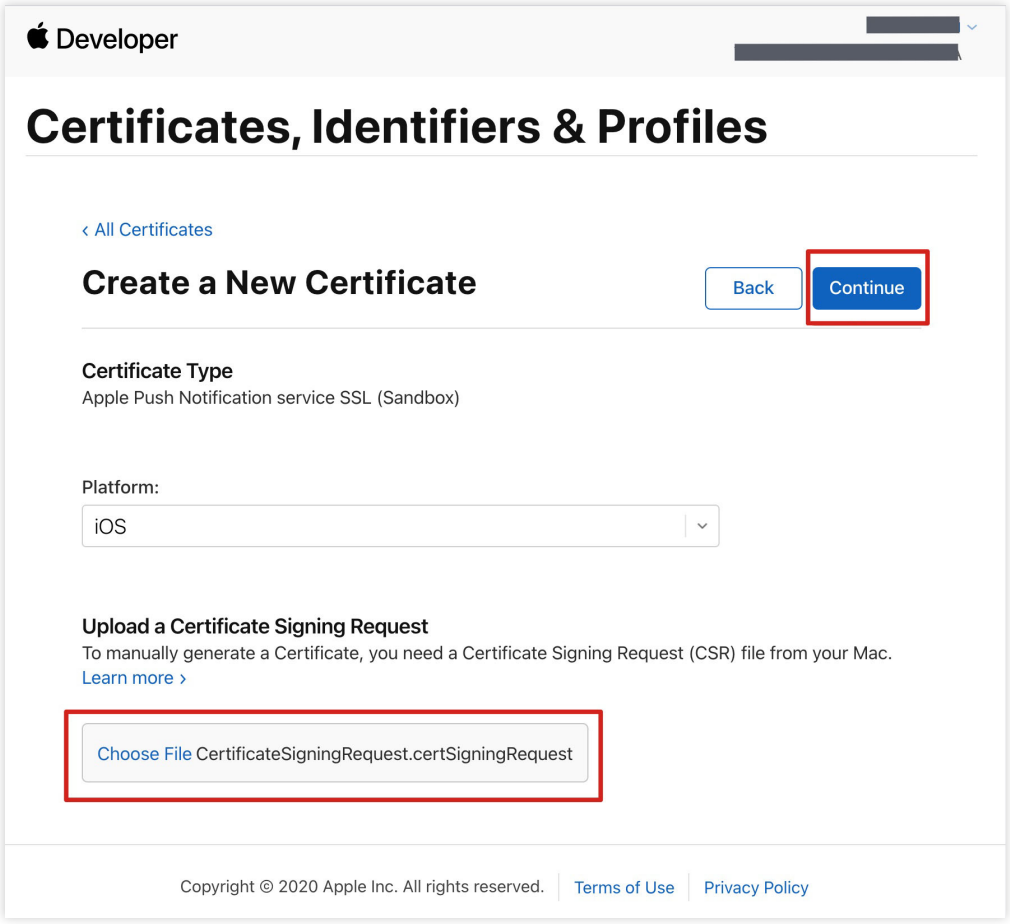
5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。



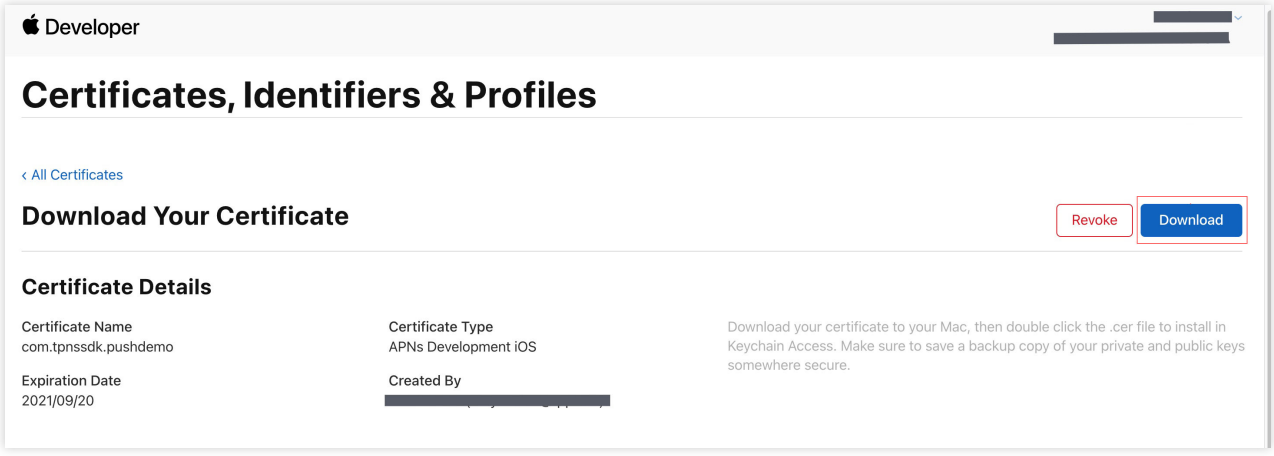
6. 返回上述 步骤3 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。



7. 单击 **Continue**，即可生成推送证书。



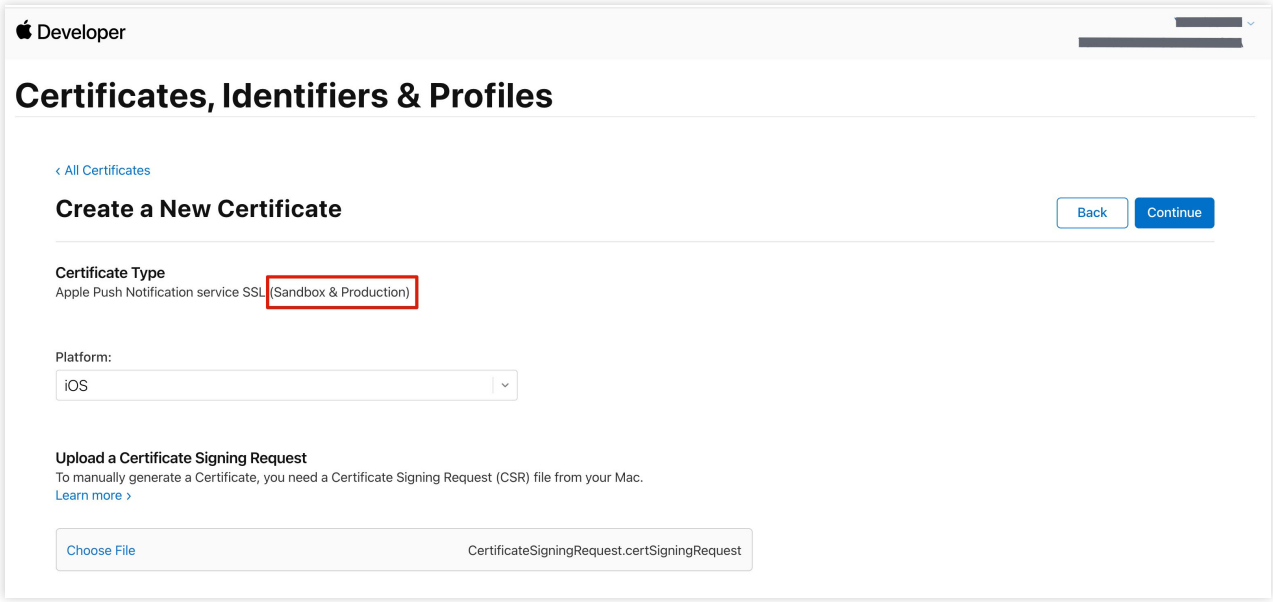
8. 单击 **Download** 下载开发环境的 `Development SSL Certificate` 到本地。

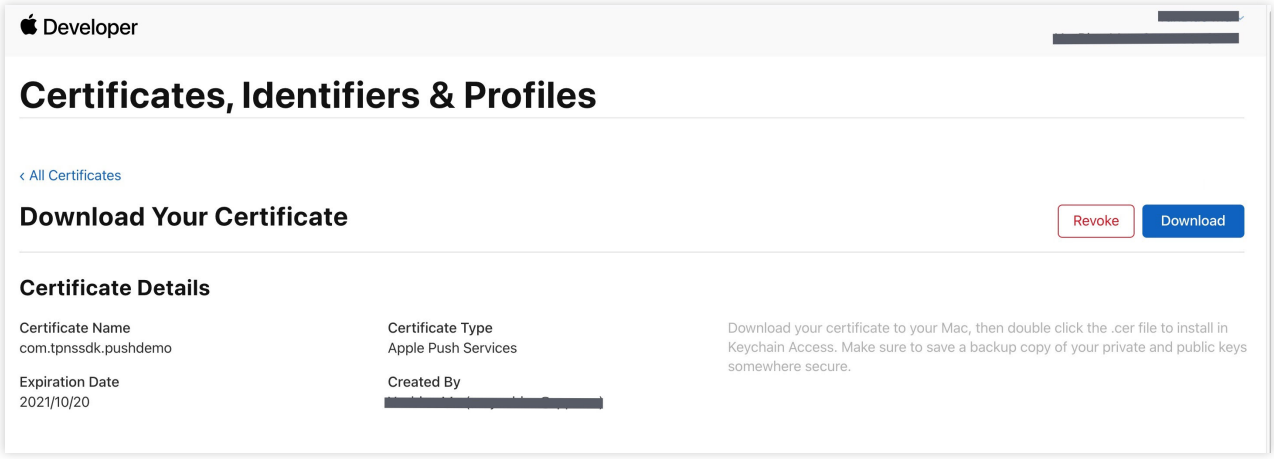


9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

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。

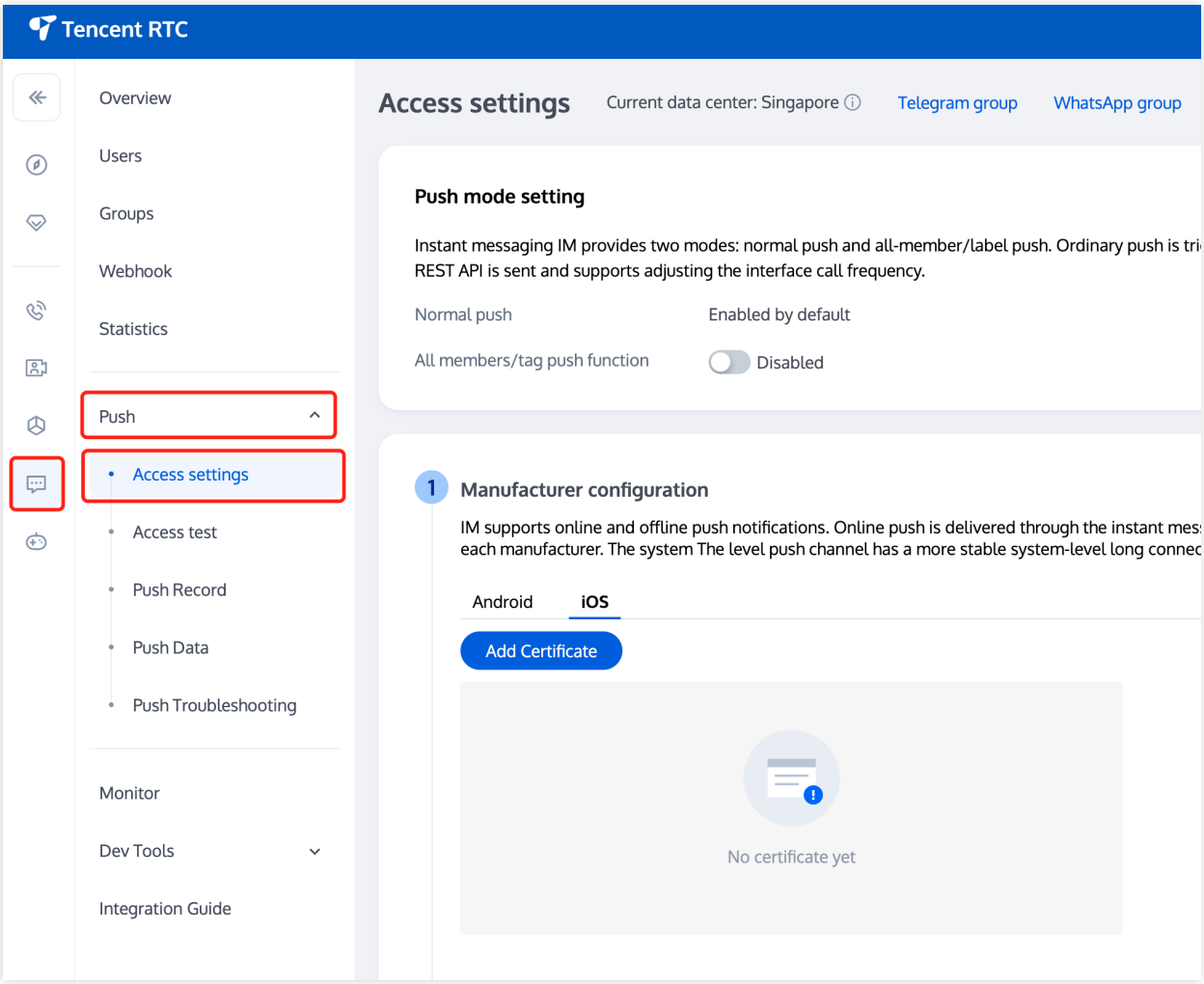
11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（`Apple Development IOS Push Service`）和生产环境（`Apple Push Services`）的 `P12` 文件。

注意：

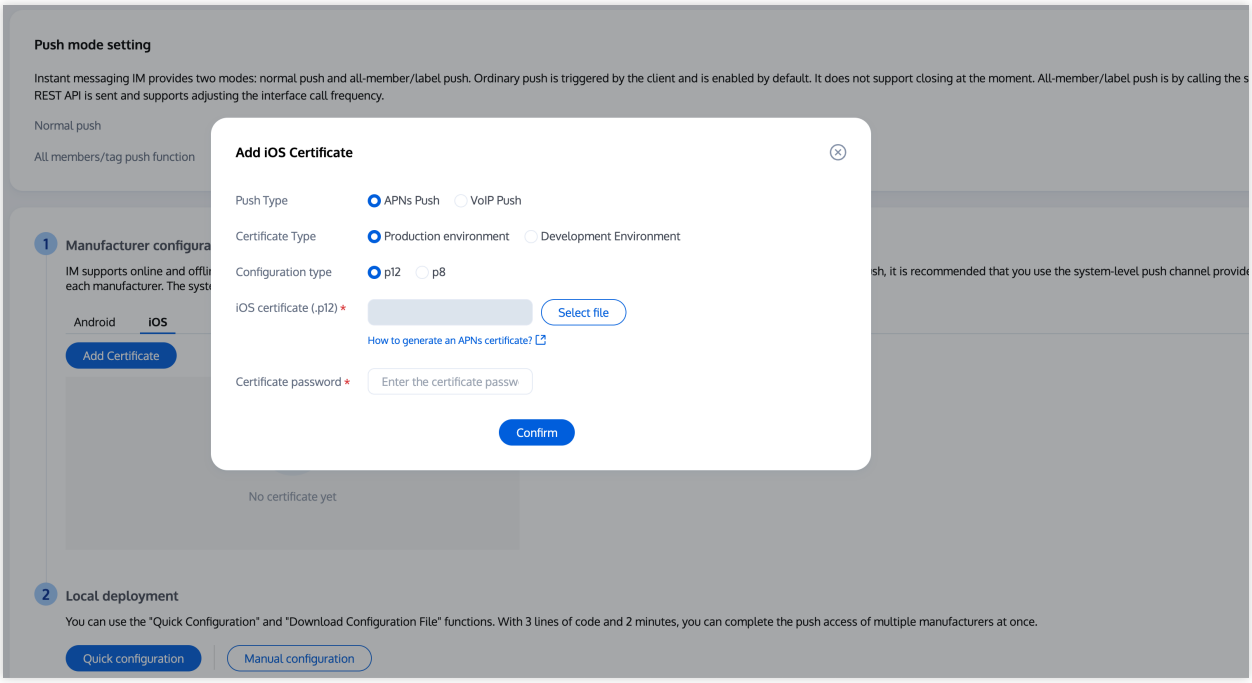
保存 `P12` 文件时，请务必要为其设置密码。

步骤2：上传证书到控制台

1. 登录 Tencent RTC 控制台。
2. 单击目标应用卡片，选择左侧的 **Chat** 标签，点击 **Push**，点击 **Access settings**。



3. 单击 **iOS 原生离线推送设置**右侧的**添加证书**。
4. 选择证书类型，上传 iOS 证书（p12），设置证书密码，单击**确认**。



注意：

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。

操作步骤

步骤1：注册应用到厂商推送平台

离线推送需要将您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，来实现离线推送功能。

步骤2：TRTC 控制台配置

登录腾讯云 Tencent RTC 控制台 ，在**Chat > Push > Access settings**功能栏添加各个厂商推送证书，并将您在步骤一中获取的各厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

注意：

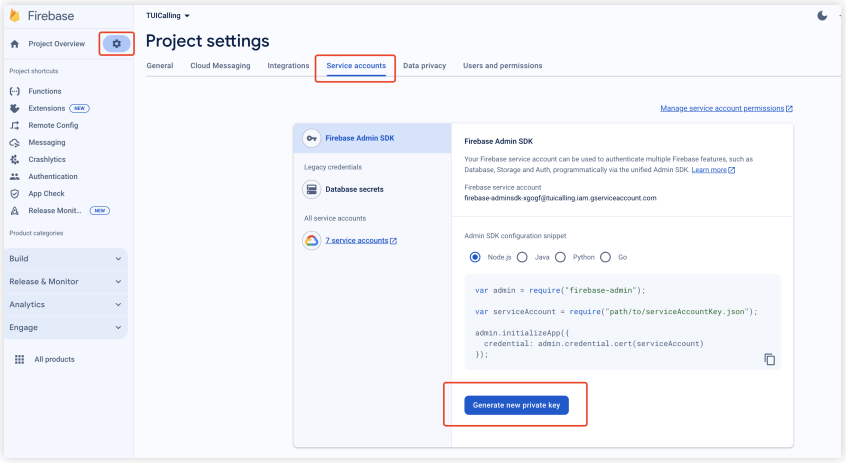
关于**点击后续动作**选项：

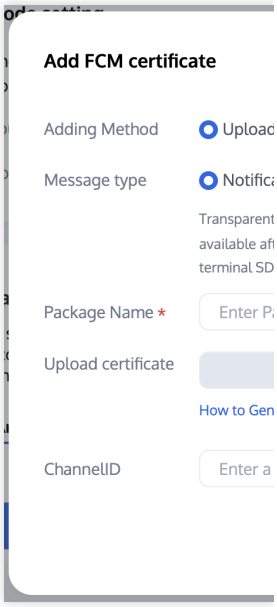
如需使用本插件提供的点击跳转能力, 请保持默认值不变, 即通常是 `打开应用内指定页面` 并带有默认配置.

如需使用上报统计功能, 也请保持此项默认值不变,

Google FCM

厂商推送平台	IM 控制台配置





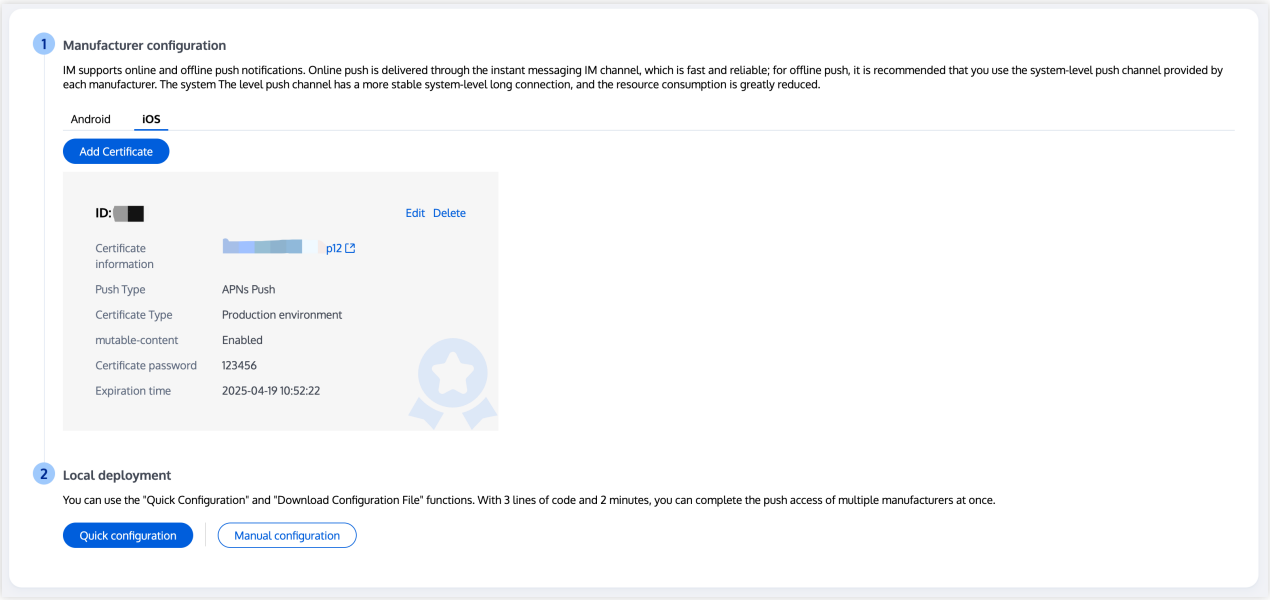
步骤3: 推送参数配置

iOS

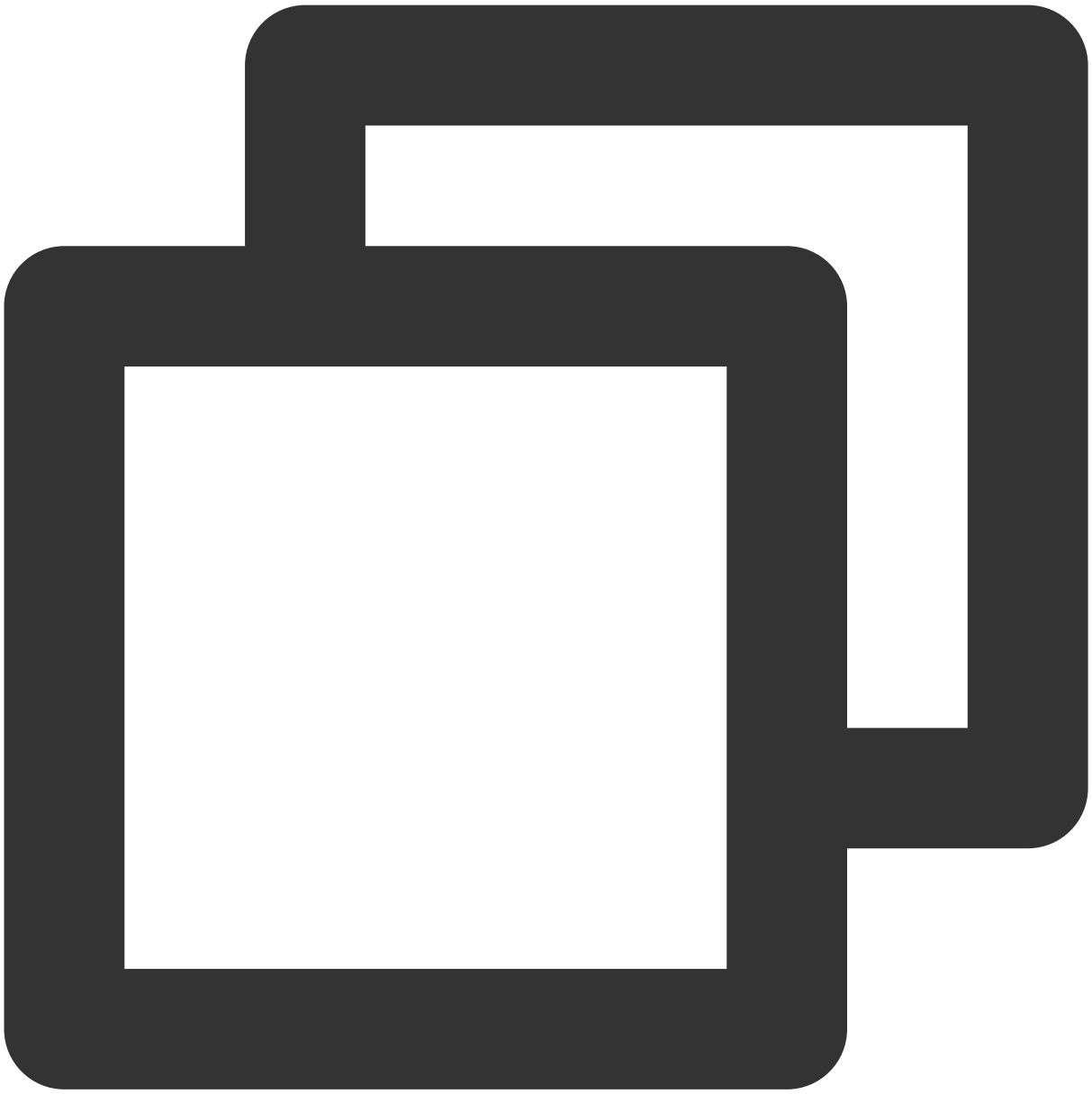
Android

请将您在厂商配置步骤中, 获取到的 iOS APNs 推送证书, 上传至 Chat 控制台。

Chat 控制台会为您分配一个证书 ID，见下图：



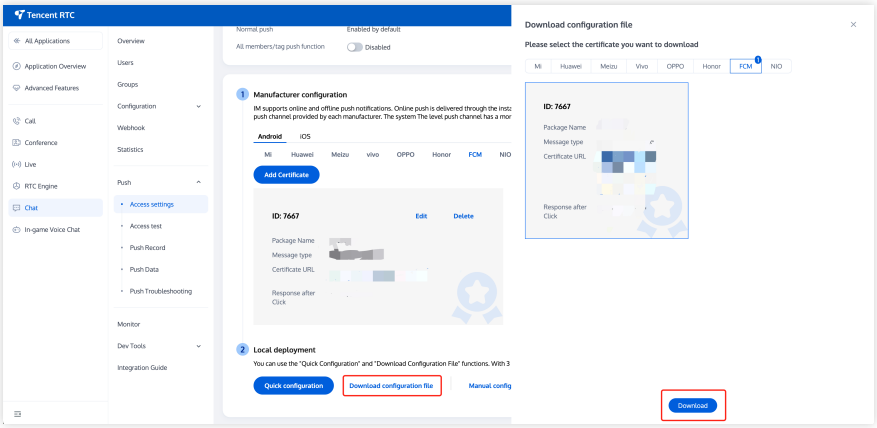
在您应用的启动后尽可能早的位置, 调用 `TencentCloudChatPush().setApnsCertificateID` 方法, 将此证书 ID 传入。

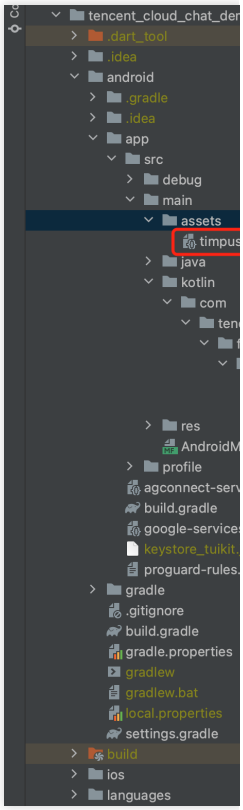


```
TencentCloudChatPush().setApnsCertificateID(apnsCertificateID: Certificate ID);
```

完成控制台厂商推送信息填写后，下载并添加配置文件到工程。将下载的 timpush-configs.json 文件添加到项目的 `android/app/src/main/assets` 目录下, 如果该目录不存在, 请手动创建.

1.选择下载配置文件 timpush-configs.json	2.添加到工程





步骤4: 客户端代码配置

本步骤，需编写若干原生代码，例如：Swift, Java, XML 等。

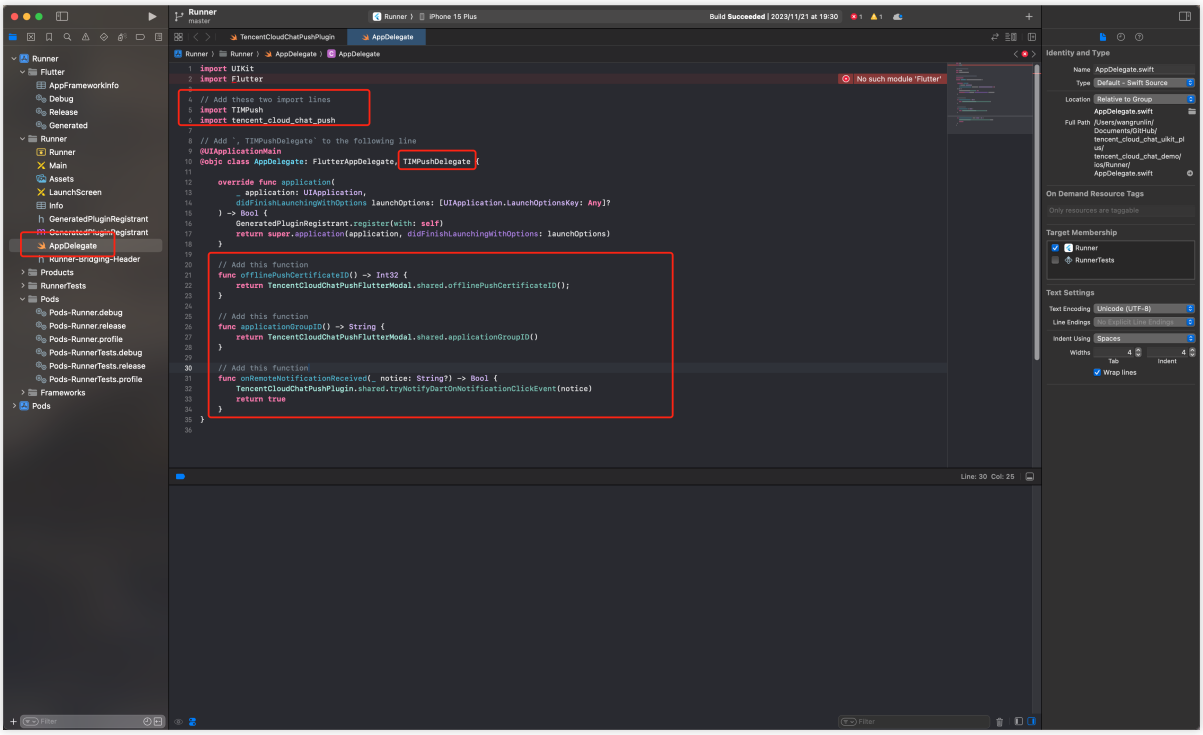
请不要担心, 直接根据说明, 复制我们提供的代码到指定文件即可。

iOS

Android

您可使用 Xcode 编辑，也可直接在 Visual Studio Code 或 Android Studio 中编辑。

打开 `ios/Runner/AppDelegate.swift` 文件，将下列圈出的代码粘贴进入，效果如图所示. 代码附在图片后。




```
import UIKit
import Flutter

// Add these two import lines
import TIMPush
import tencent_cloud_chat_push

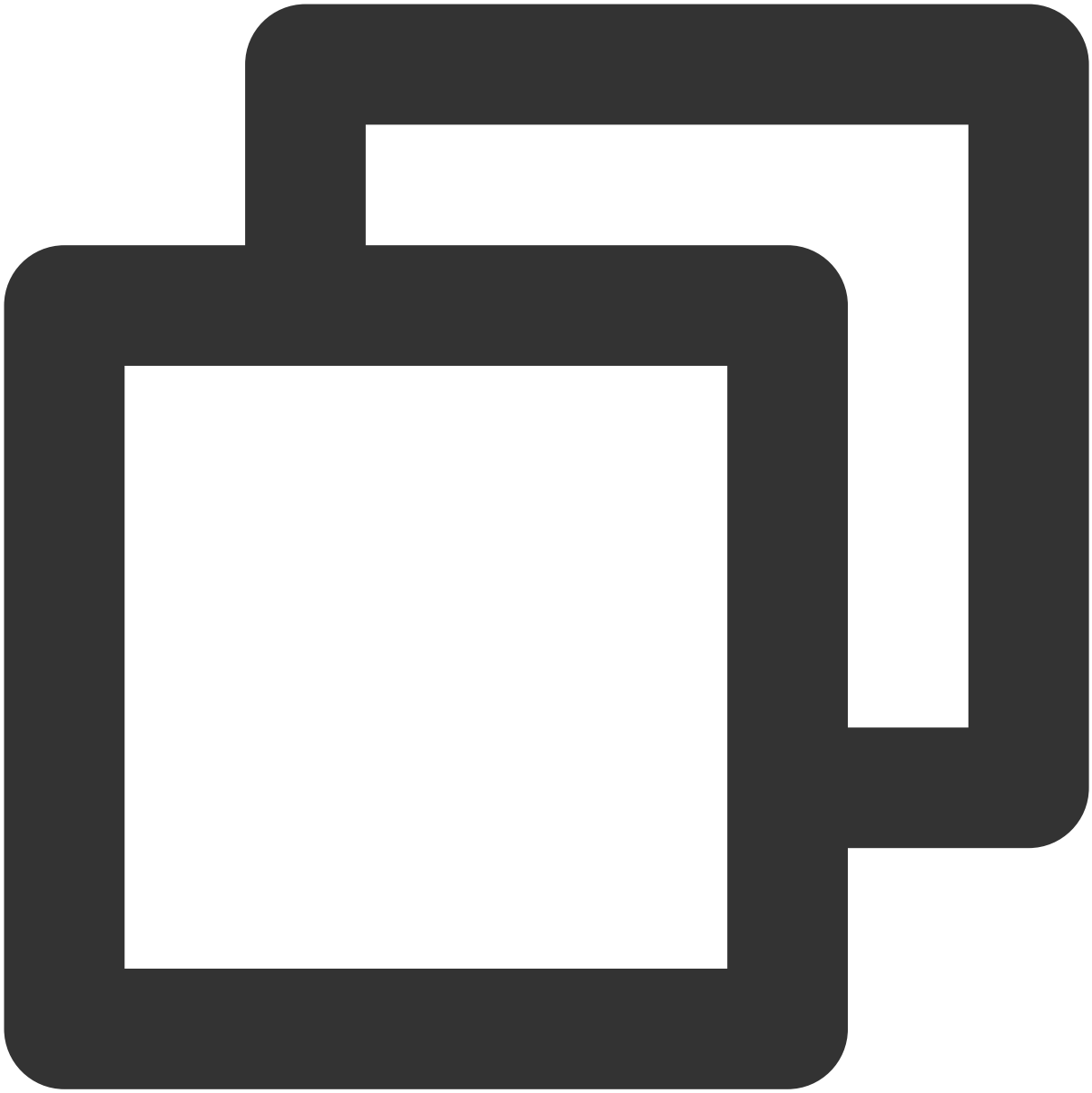
// Add `, TIMPushDelegate` to the following line
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate, TIMPushDelegate {

    override func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
    ) -> Bool {
        GeneratedPluginRegistrant.register(with: self)
        return super.application(application, didFinishLaunchingWithOptions: launchOptions)
    }

    // Add this function
    func offlinePushCertificateID() -> Int32 {
        return TencentCloudChatPushFlutterModal.shared.offlinePushCertificateID();
    }

    // Add this function
    func applicationGroupID() -> String {
        return TencentCloudChatPushFlutterModal.shared.applicationGroupID()
    }

    // Add this function
    func onRemoteNotificationReceived(_ notice: String?) -> Bool {
        TencentCloudChatPushPlugin.shared.tryNotifyDartOnNotificationClickEvent(notice)
        return true
    }
}
```



```
import UIKit
import Flutter

// Add these two import lines
import TIMPush
import tencent_cloud_chat_push

// Add `, TIMPushDelegate` to the following line
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate, TIMPushDelegate {
    override func application(
```

```
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKe
    ) -> Bool {
        GeneratedPluginRegistrant.register(with: self)
        return super.application(application, didFinishLaunchingWithOptions: launch
    }

    // Add this function
    func offlinePushCertificateID() -> Int32 {
        return TencentCloudChatPushFlutterModal.shared.offlinePushCertificateID();
    }

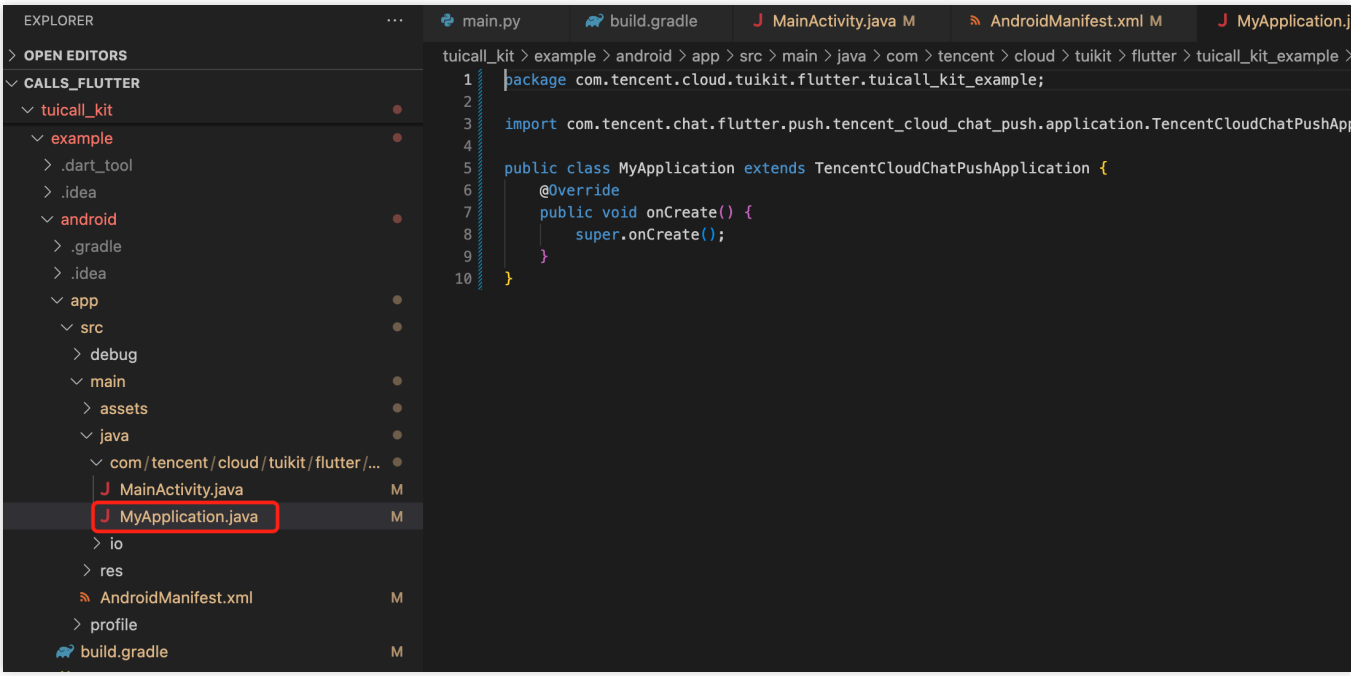
    // Add this function
    func applicationGroupID() -> String {
        return TencentCloudChatPushFlutterModal.shared.applicationGroupID()
    }

    // Add this function
    func onRemoteNotificationReceived(_ notice: String?) -> Bool {
        TencentCloudChatPushPlugin.shared.tryNotifyDartOnNotificationClickEvent(not
        return true
    }
}
```

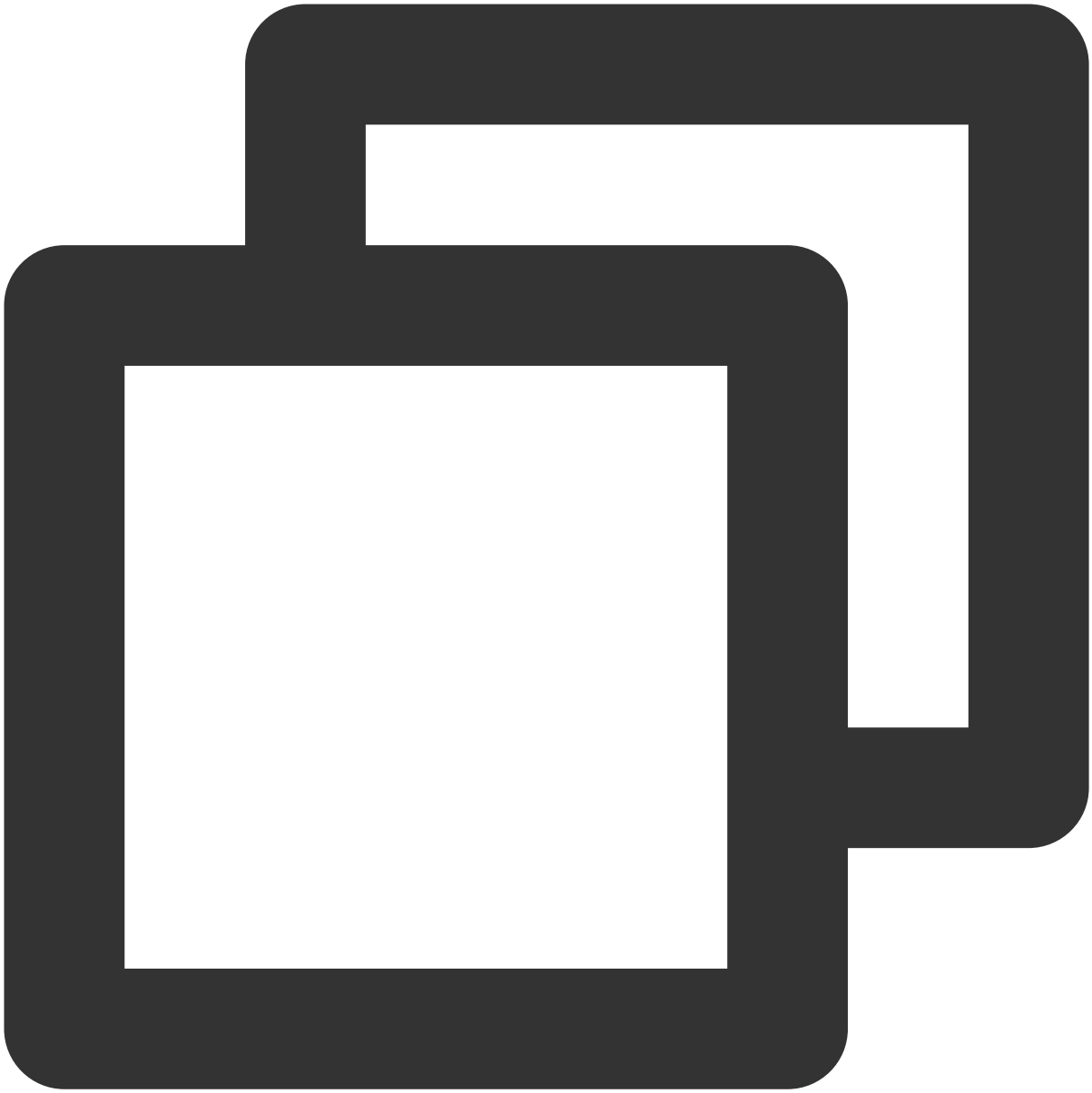
建议使用 Android Studio 完成本部分编辑。

在您项目 android 路径下 `MainActivity` 同级目录中，新建一个新的 Application 文件类, 例如可命名为 `MyApplication` 。

如果您已经自定义了一个 Application 类，则可直接复用，不需要再次创建。



将下列代码粘贴到该文件中, 如上图所示：



```
package xxxx.xxxx.xx

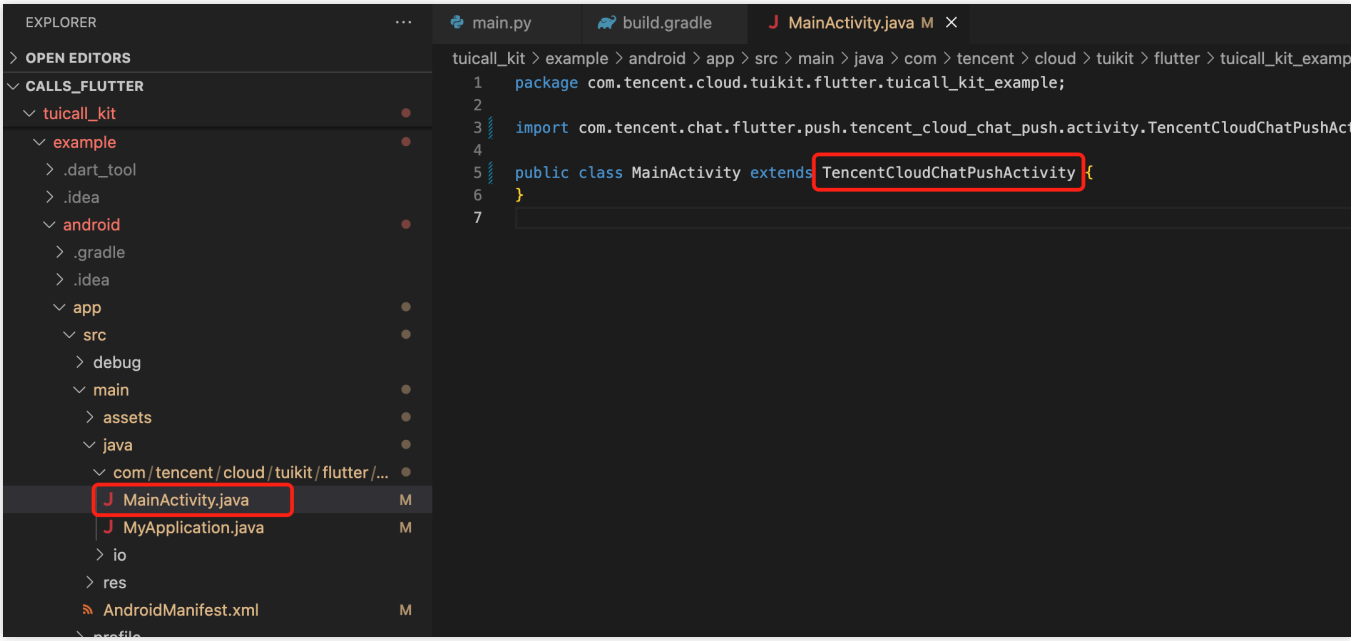
import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

public class MyApplication extends TencentCloudChatPushApplication {
    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

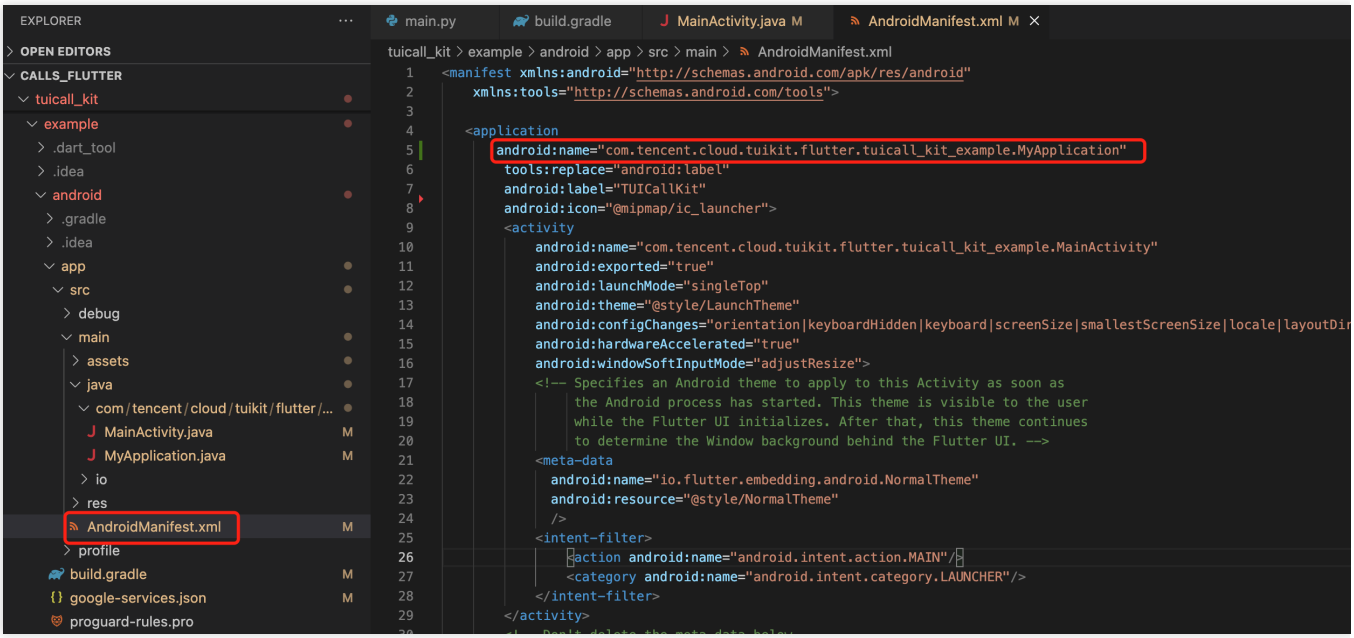
说明：

如果您已经创建了自己的 Application 为了其他用途，请直接 `extends TencentCloudChatPushApplication` 并保证 `onCreate()` 函数中，调用了 `super.onCreate();` 即可。

同时，您还需要修改您的 MainActivity 文件：



打开 `android/app/src/main/AndroidManifest.xml` 文件，为 `<application>` 标签，新增指定一个 `android:name` 参数即可，指向刚制作的自定义 `Application` 类。如图所示：



步骤5: 客户端厂商配置

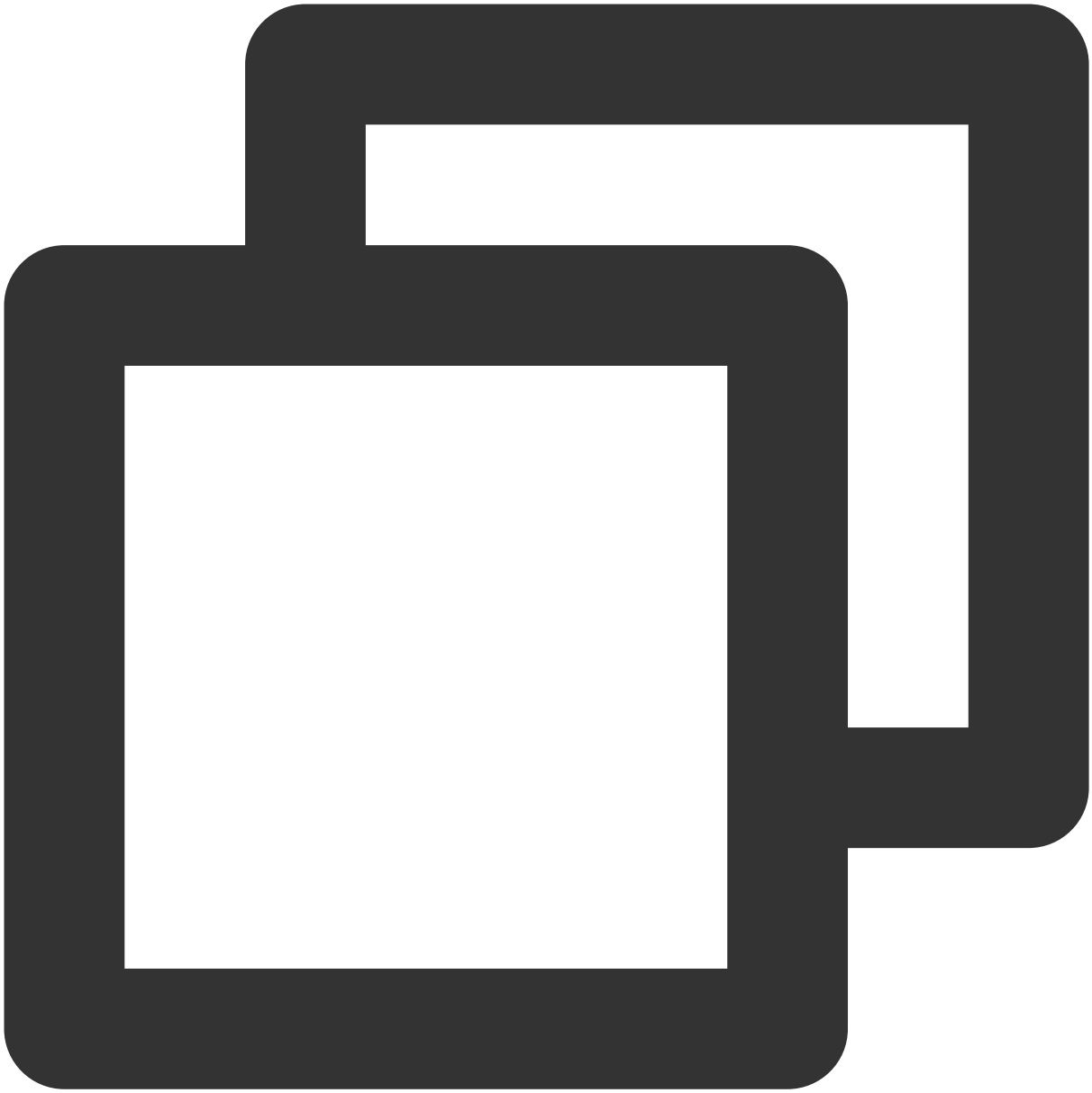
iOS

Android

iOS 端无需进行此步骤。

打开 android/app/build.gradle 文件，在最后，新增 dependencies 配置，并根据需要，引入下列全部或部分厂商的推送包。只有引入对应厂商的推送包，才能启用该厂商的原生推送能力。

以下所述的版本号，和本 Flutter 推送插件（tencent_cloud_chat_push）的版本号，保持一致。



```
dependencies {
     // Google Firebase Cloud Messaging (Google FCM)
     implementation 'com.tencent.timpush:fcm:${推送插件的版本号}'
}
```

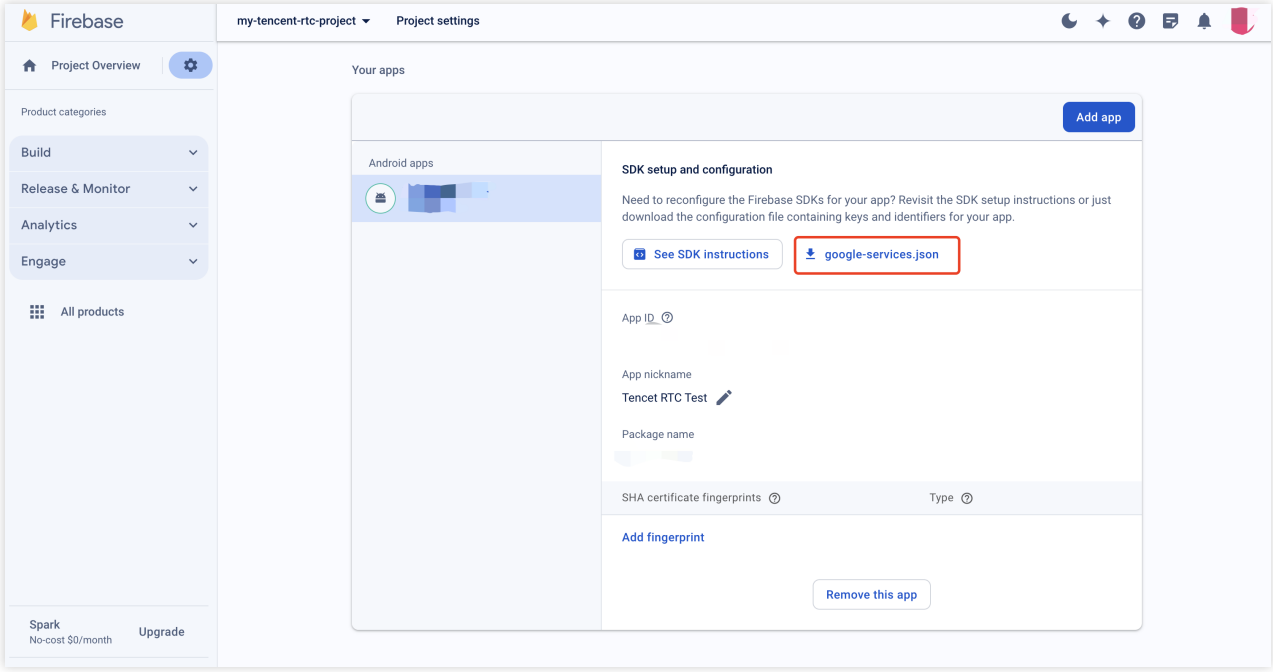
Google FCM 适配

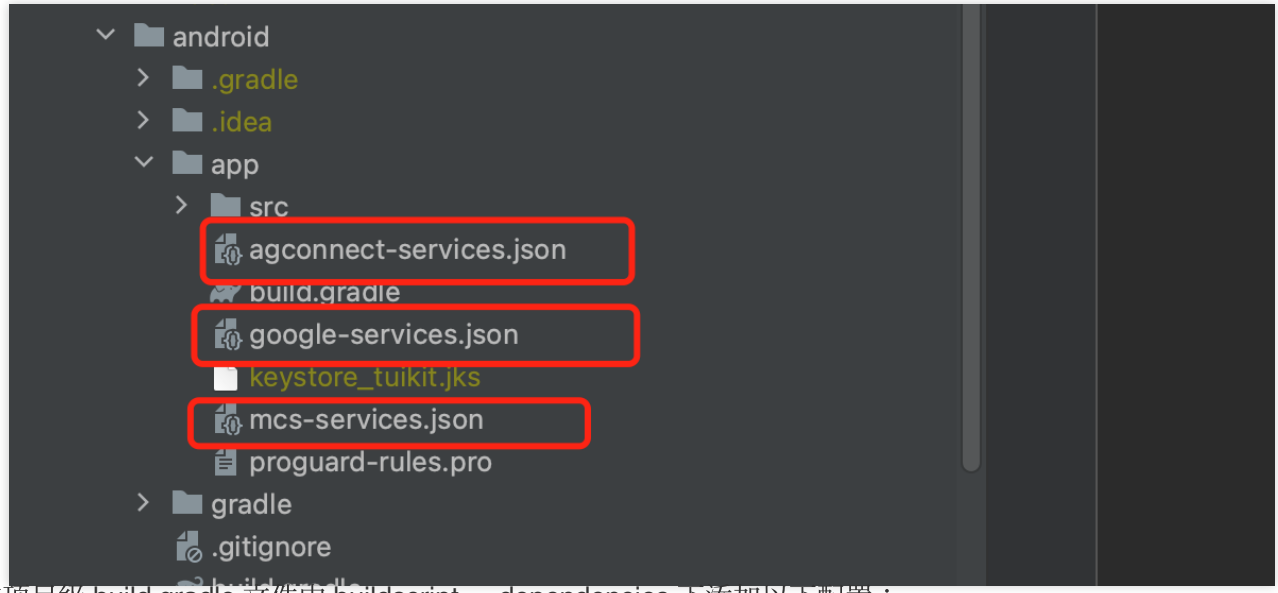
按照厂商方法，集成对应的 plugin 和 json 配置文件。

1.1 下载配置文件添加到工程根目录/Android/app。

Google FCM

操作路径





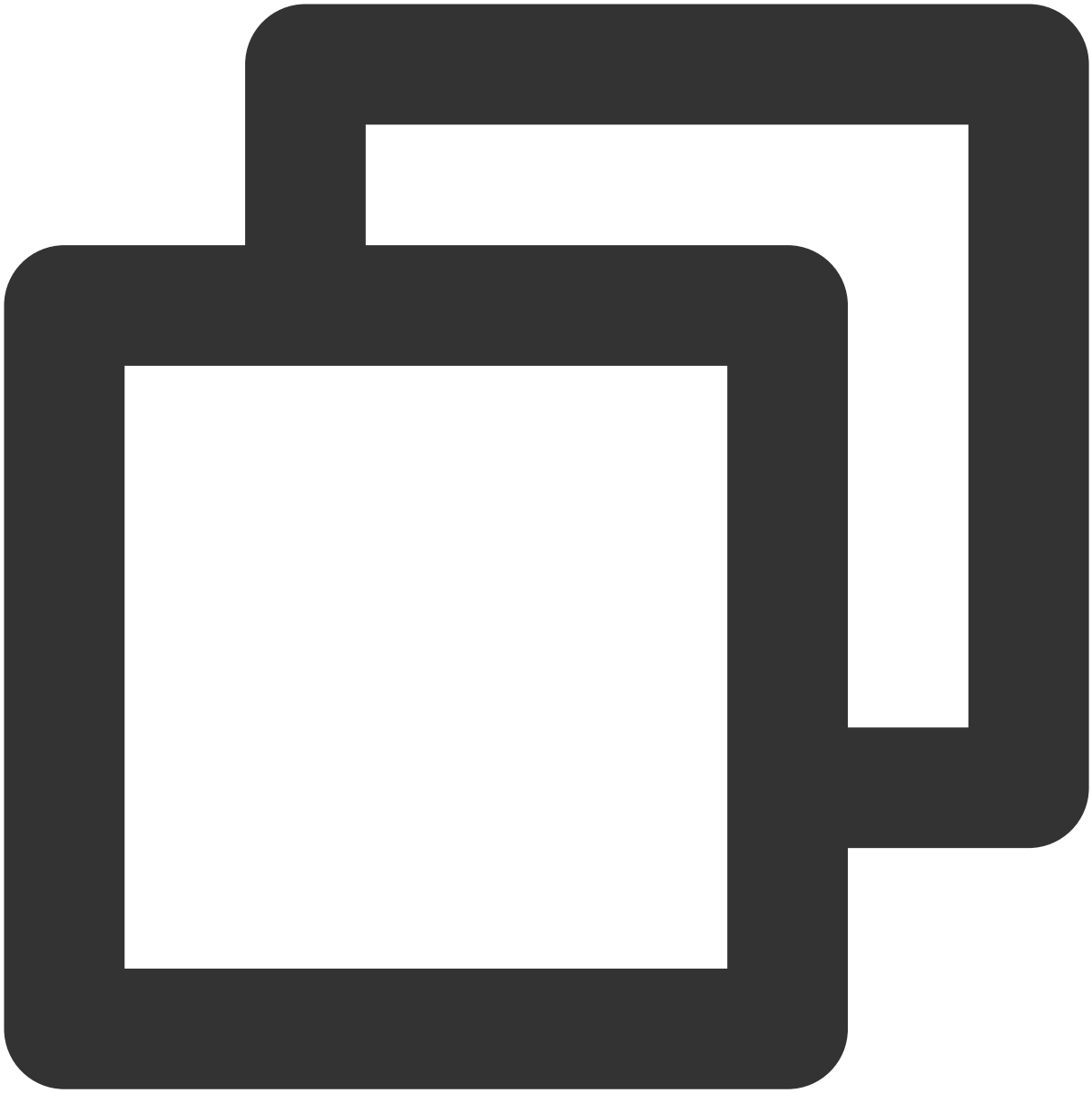
1.2 在项目级 build.gradle 文件中 buildscript -> dependencies 下添加以下配置：

Gradle 7.1 及以上版本

Gradle 7.0 版本

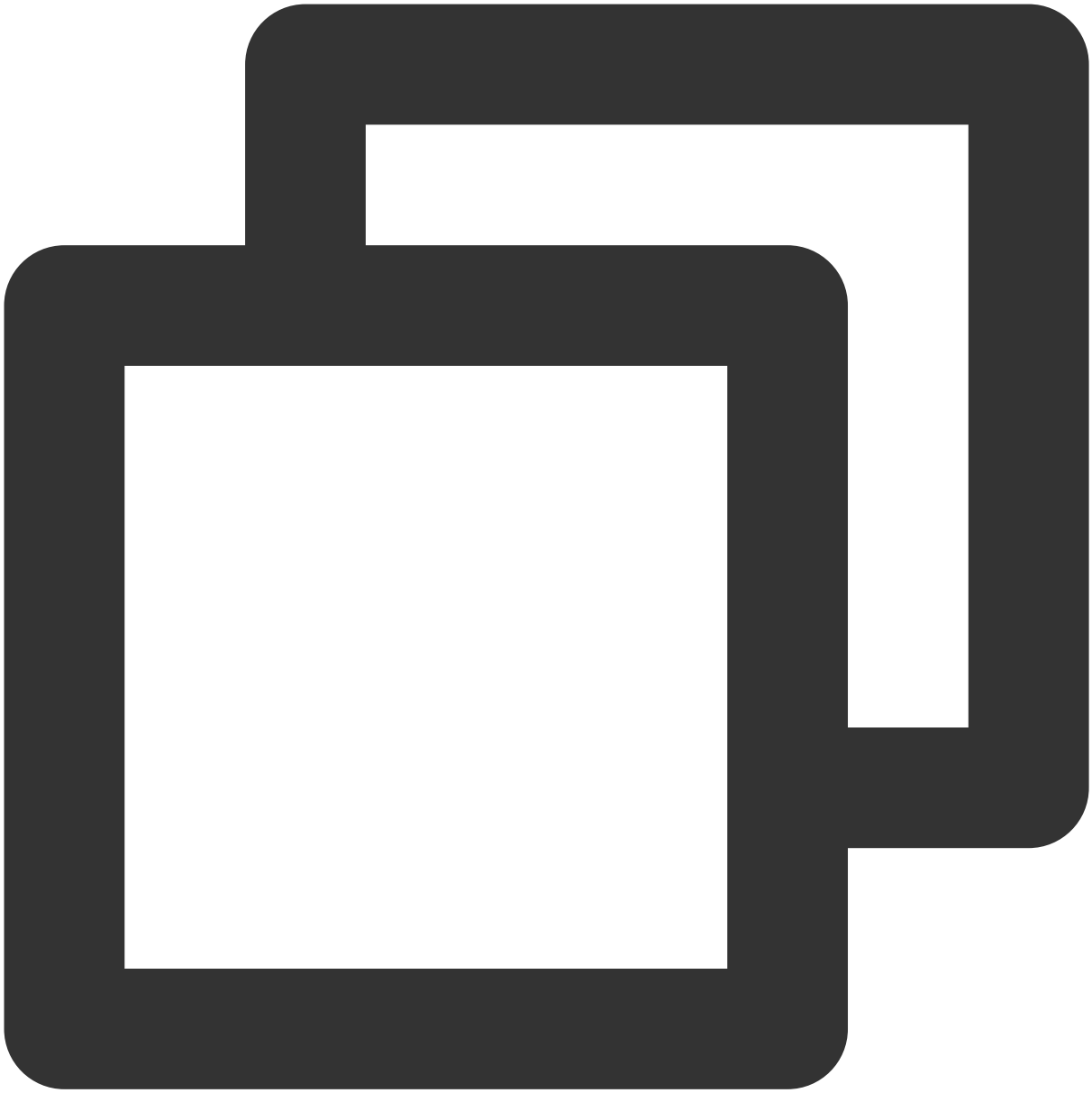
Gradle 7.0 以下版本

在项目级 build.gradle 文件中 buildscript -> dependencies 下添加以下配置：



```
buildscript {
    dependencies {
        ...
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
        classpath 'com.google.gms:google-services:4.4.0'
    }
}
```

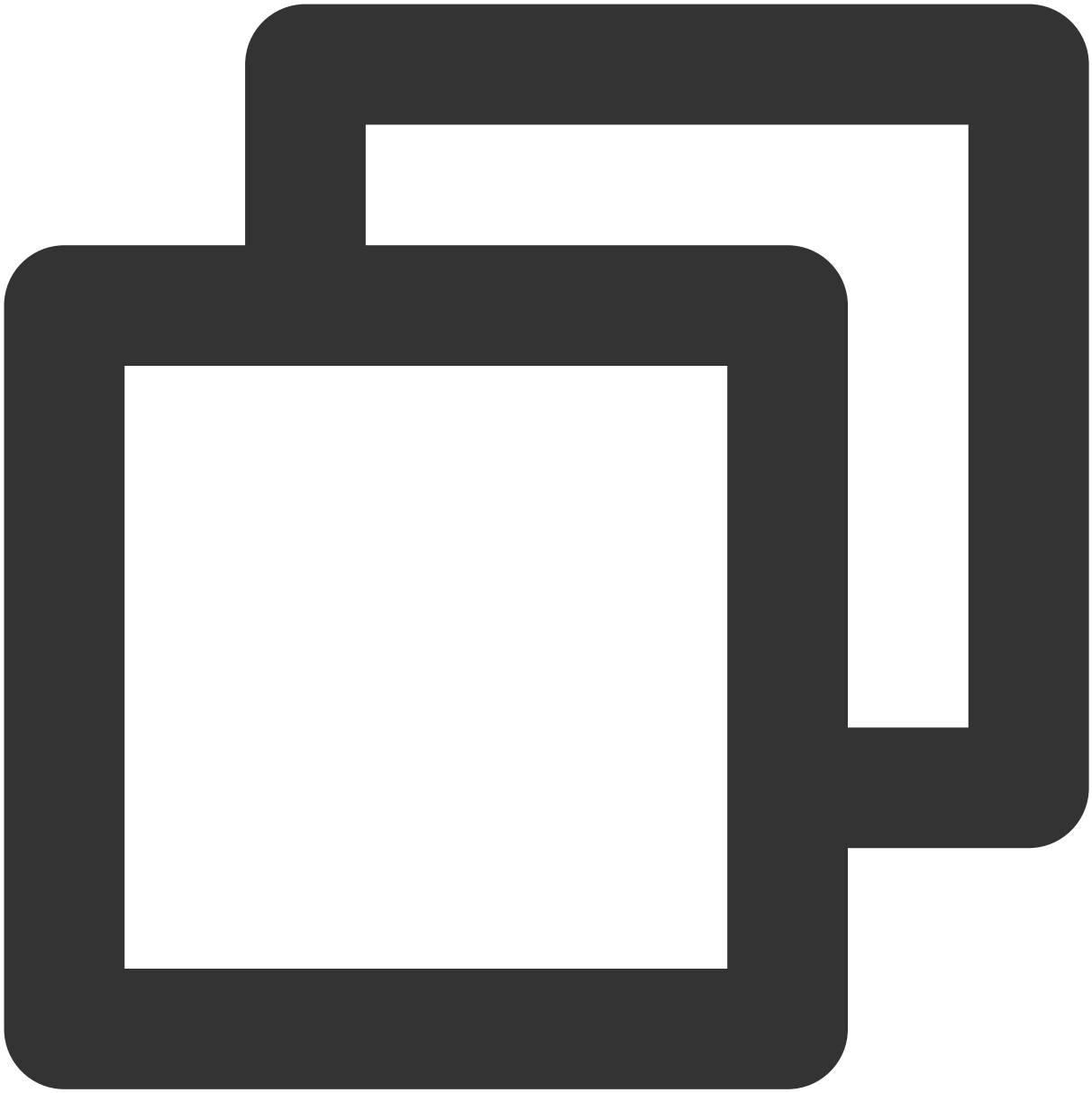
在项目级 settings.gradle 文件中 buildscript -> repositories 和 allprojects -> repositories 下添加以下仓库配置：



```
pluginManagementbuildscript {
    repositories {
        gradlePluginPortal()
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
allprojects {
```

```
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    }
}
```

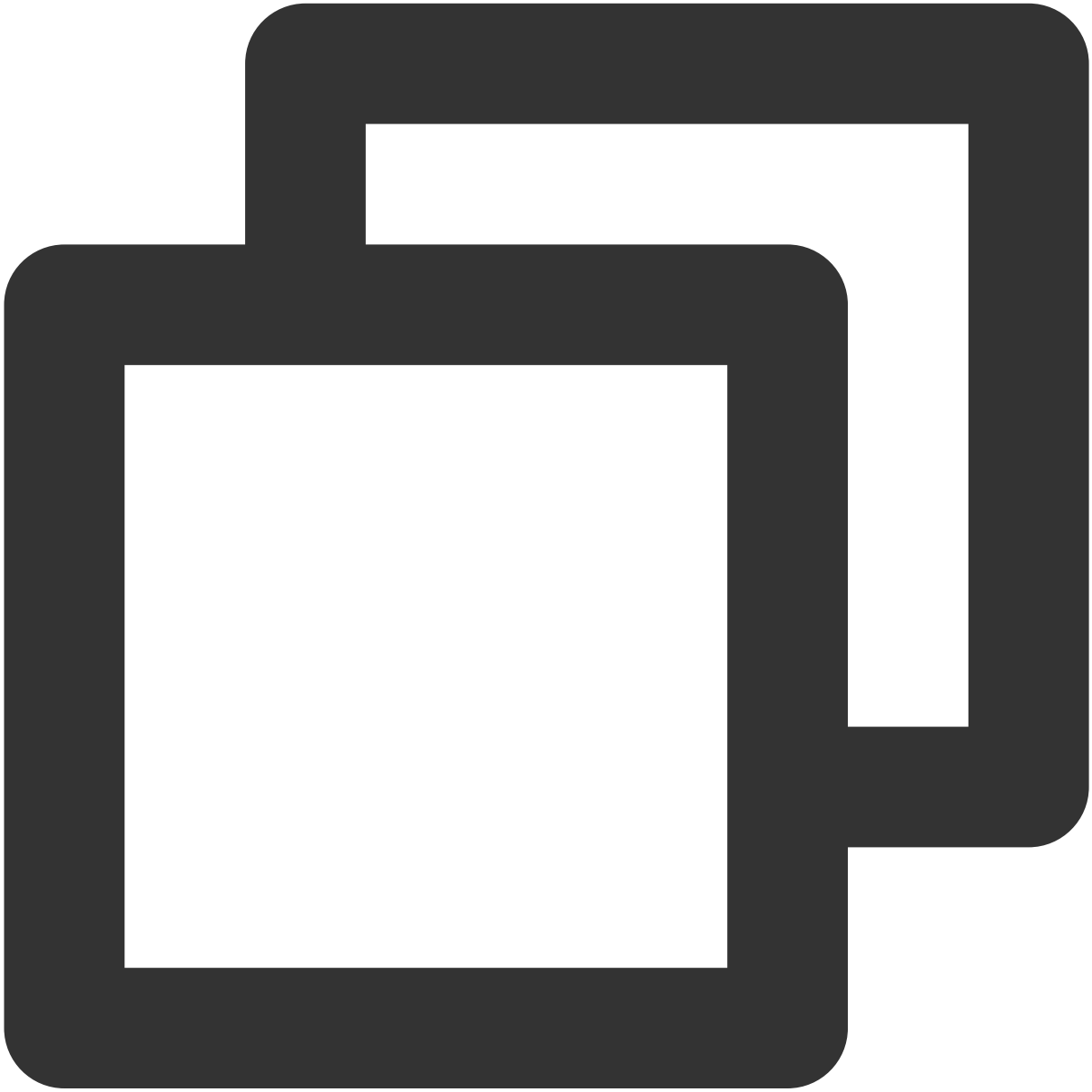
在项目级 build.gradle 文件中 buildscript 下添加以下配置：



```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
```

```
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

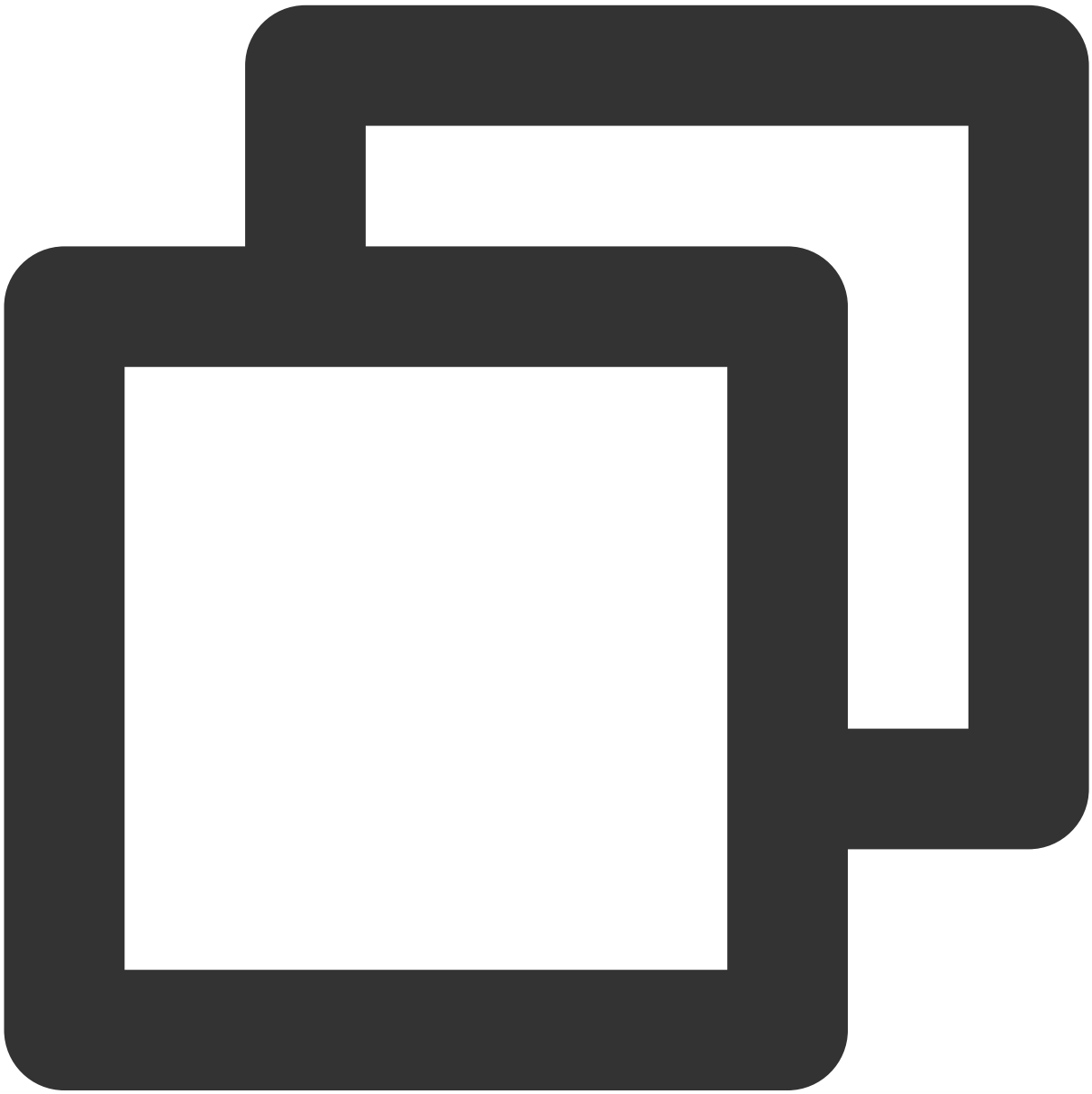
在项目级 settings.gradle 文件中 allprojects -> repositories 下添加以下仓库配置：



```
allprojects {
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
```

```
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

在项目级 build.gradle 文件中 buildscript 和 allprojects 下添加以下配置：

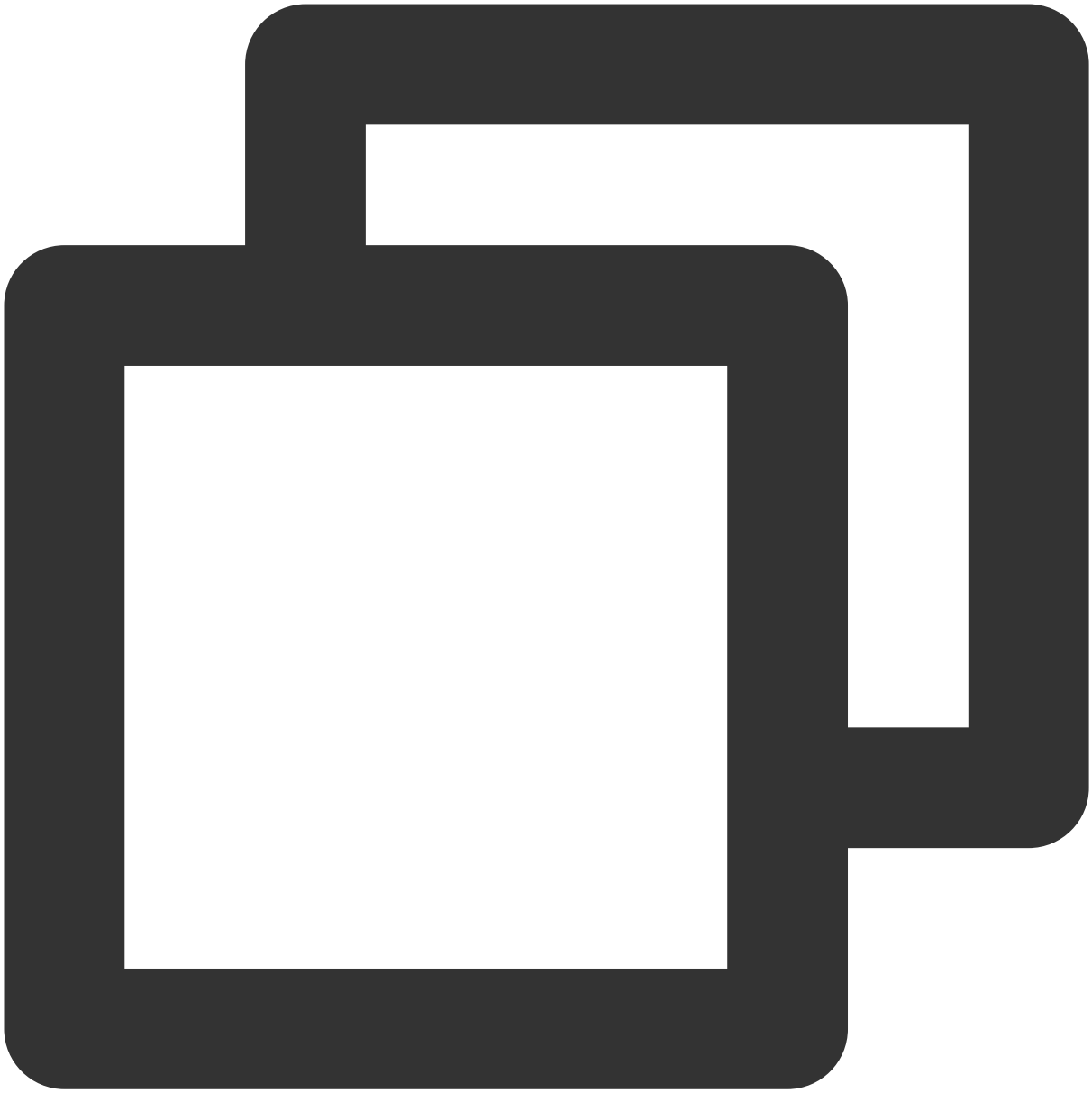


```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
```

```
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}

allprojects {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

1.3 在应用级 build.gradle 文件中添加下方配置：



```
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
apply plugin: 'com.hihonor.mcs.asplugin'
```

步骤6: 处理消息点击回调, 并解析参数

请定义一个函数, 用于接受推送消息点击回调事件.

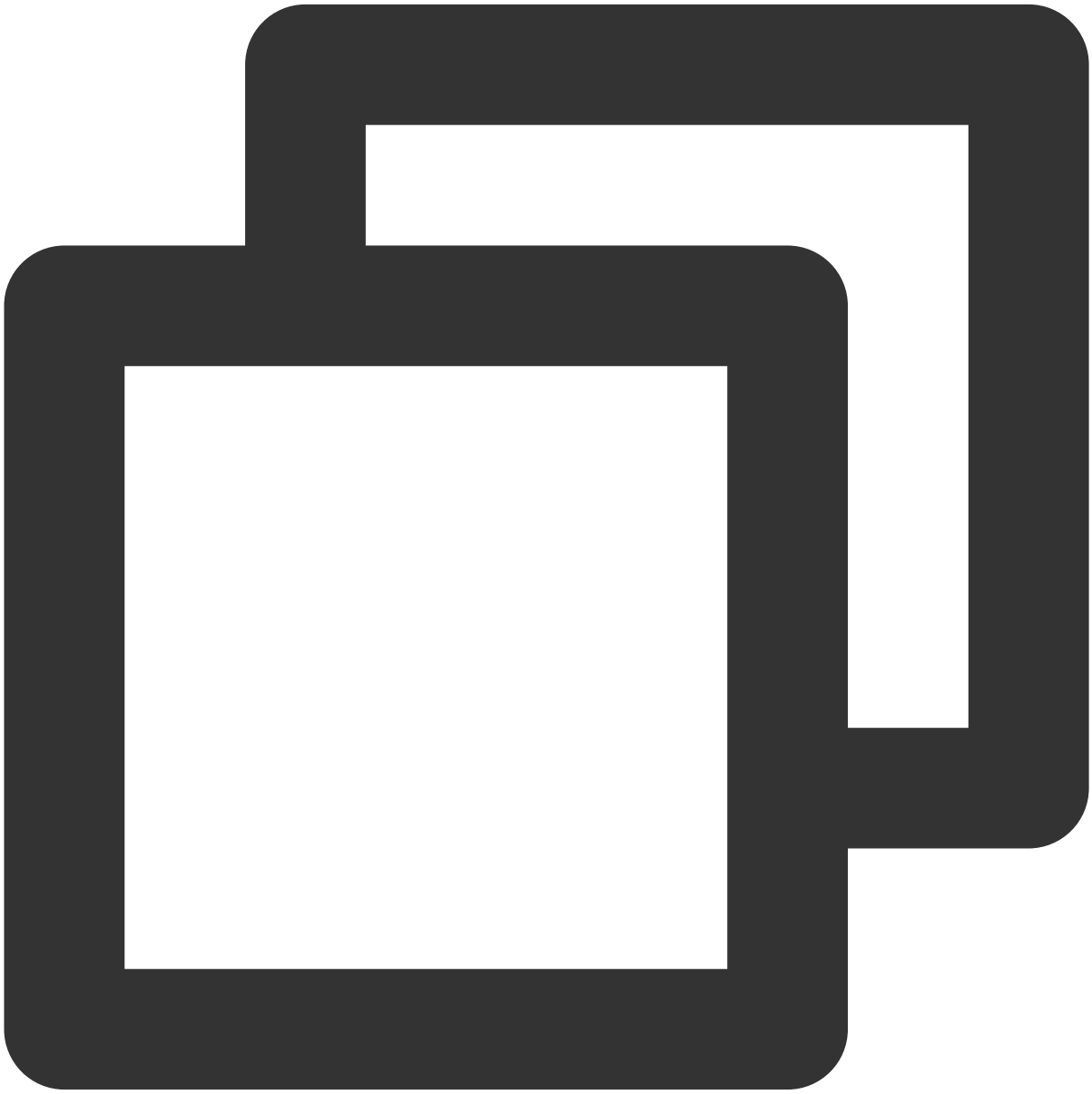
该函数请定义成 `{required String ext, String? userID, String? groupID}` 的入参形式。

其中, ext字段, 为该消息所携带的完整 ext 信息, 由发送方指定, 如果未指定, 则有默认值. 您可根据解析该字段, 跳转至对应页面。

userID 和 groupID 字段，为本插件，自动尝试解析 ext Json String， 获取里面携带的单聊对方 userID 和 群聊 groupID 信息。如果您未自定义 ext 字段，ext 字段由 SDK 或 UIKit 默认指定，则可使用此处的默认解析。如果尝试解析失败, 则为 null 空。

您可定义一个函数来接收该回调，并据此跳转至对应会话页面或您的业务页面。

示例如下:



```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
```

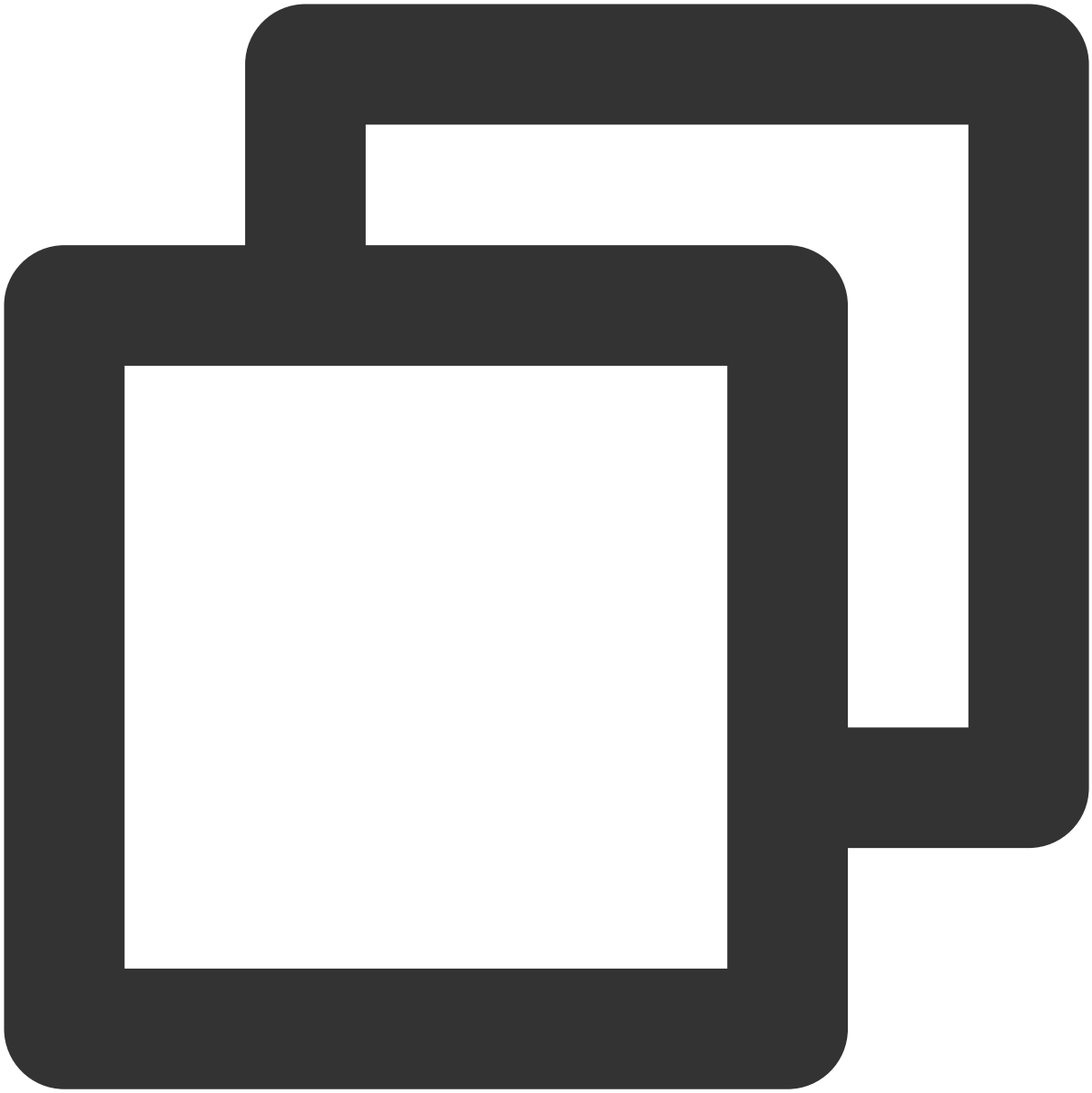
```
    if (userID != null || groupID != null) {
      // 根据 userID 或 groupID 跳转至对应 Message 页面.
    } else {
      // 根据 ext 字段，自己写解析方式，跳转至对应页面.
    }
  }
```

步骤7: 注册推送插件

请在登录完成后，立即注册推送插件。

调用 `TencentCloudChatPush().registerPush` 方法，需传入上一步定义的点击回调函数。

此外，您还可选传入 `apnsCertificateID` iOS 推送证书 ID 及 `androidPushOEMConfig` Android 推送厂商配置。此二项配置已在前序步骤指定，若无修改必要，可不再传入。

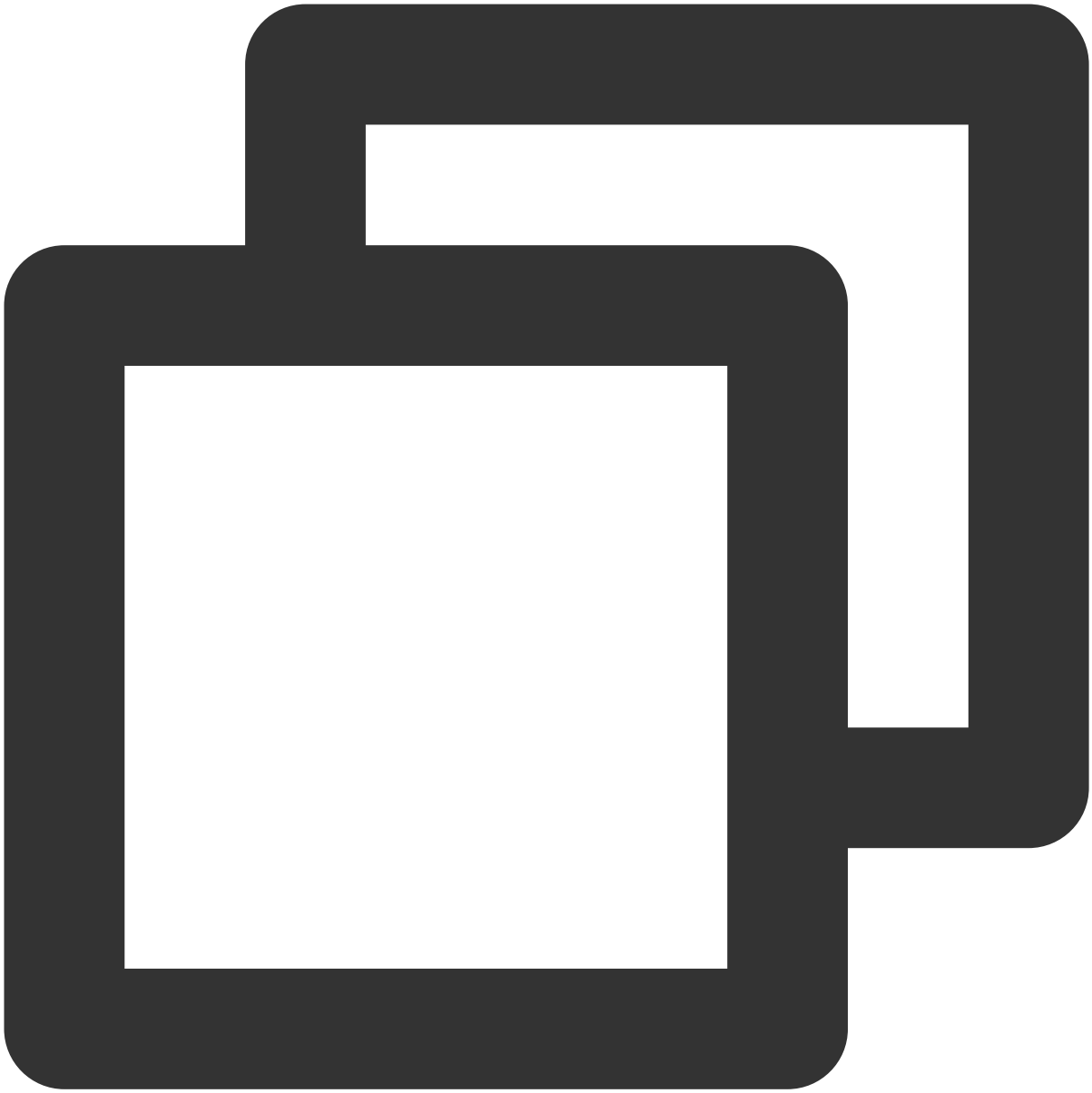


```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

说明：

如果您的应用需要使用**推送插件进行业务消息通知**，并且在**启动后不会立即启动并登录 Chat 模块**，或者在**登录 Chat 模块之前需要通过获取消息点击回调来处理业务导航**，建议您尽早调用 `TencentCloudChatPush().registerOnNotificationClickedEvent` 方法，手动挂载消息单击回调，以便及时获取消息参数。

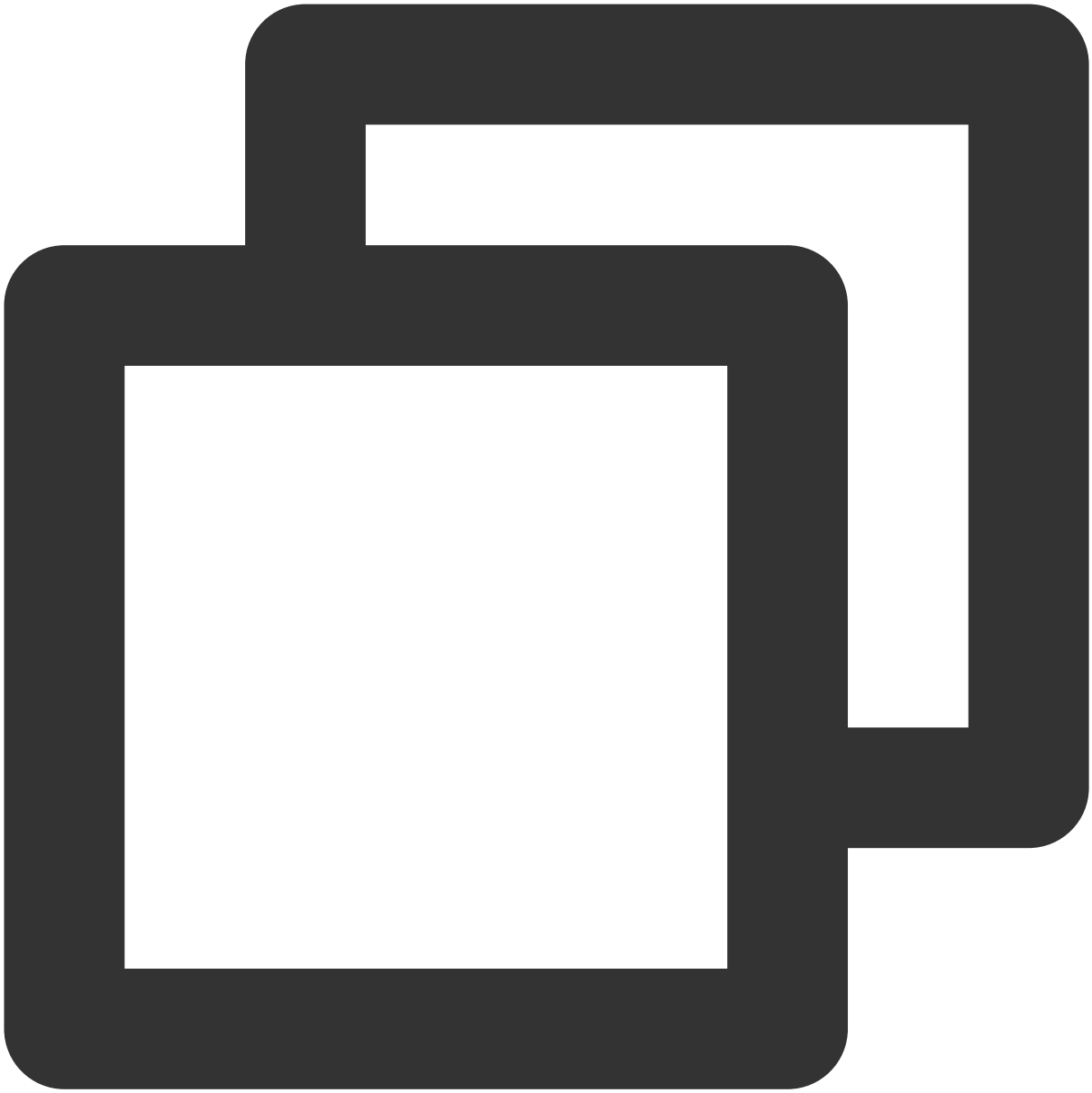
在这种场景下，您可以在调用 `TencentCloudChatPush().registerPush` 之前执行此函数，并尽可能提前将其放置在代码中。



```
TencentCloudChatPush().registerOnNotificationClickedEvent(onNotificationClicked: _o
```

步骤8: 拨打离线推送通话

如果您要拨打离线推送通话，需要在调用 call 或者 groupCall 的时候设置 OfflinePushInfo 。



```
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;
```

版权所有：腾讯云计算（北京）有限责任公司

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

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

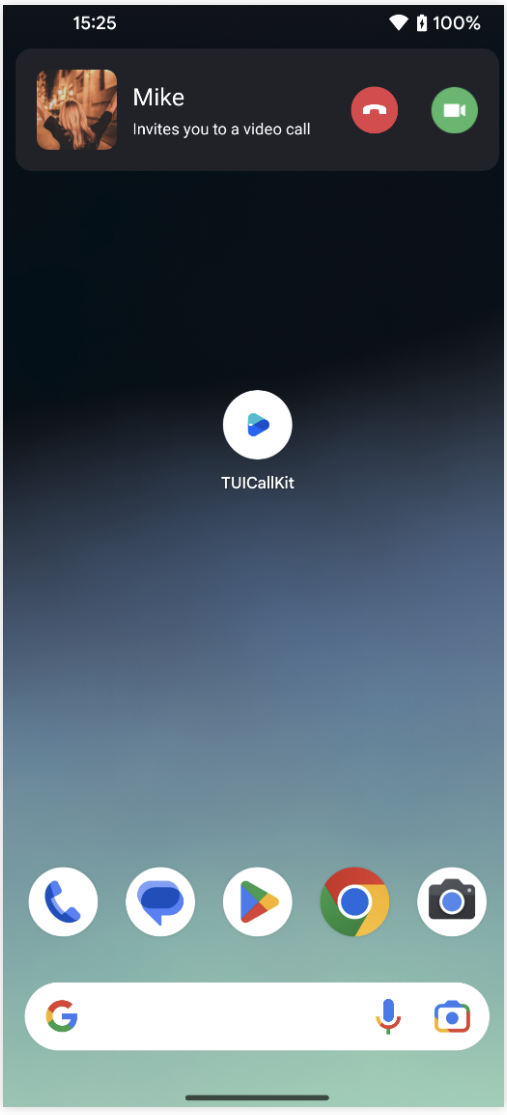
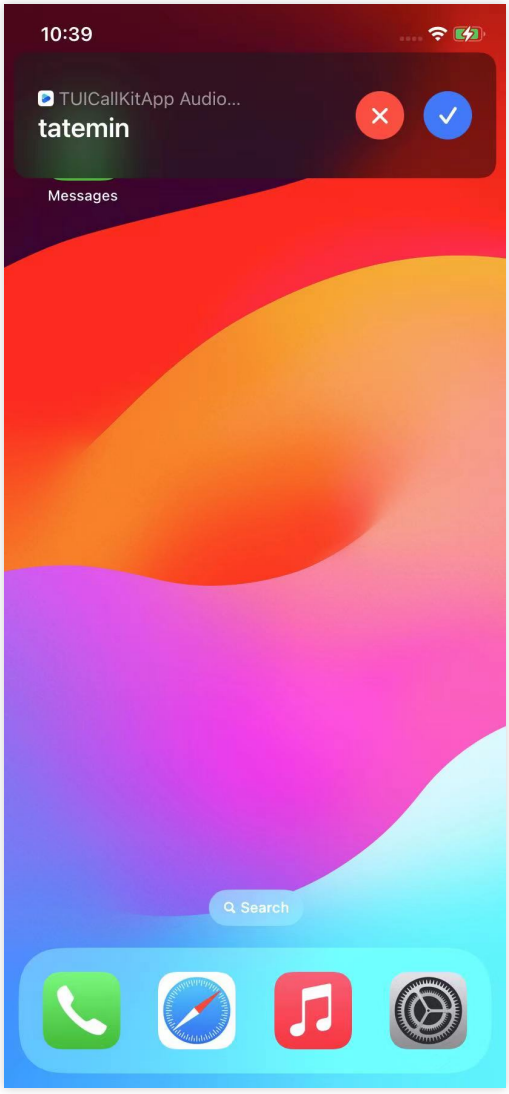
说明：

如果您的 Android 程序在接收推送或拉起页面时遇到问题，您可以参见 被叫端来电显示策略 对问题进行排查。

VoIP(可选)

最近更新时间：2024-07-22 17:00:45

VoIP 通话是一种通过互联网或 IP 网络传输数字语音数据的电话通话技术，具有成本低、音质高、灵活性强和集成其他通信服务的优势。

Android	iOS
	

配置VoIP

iOS

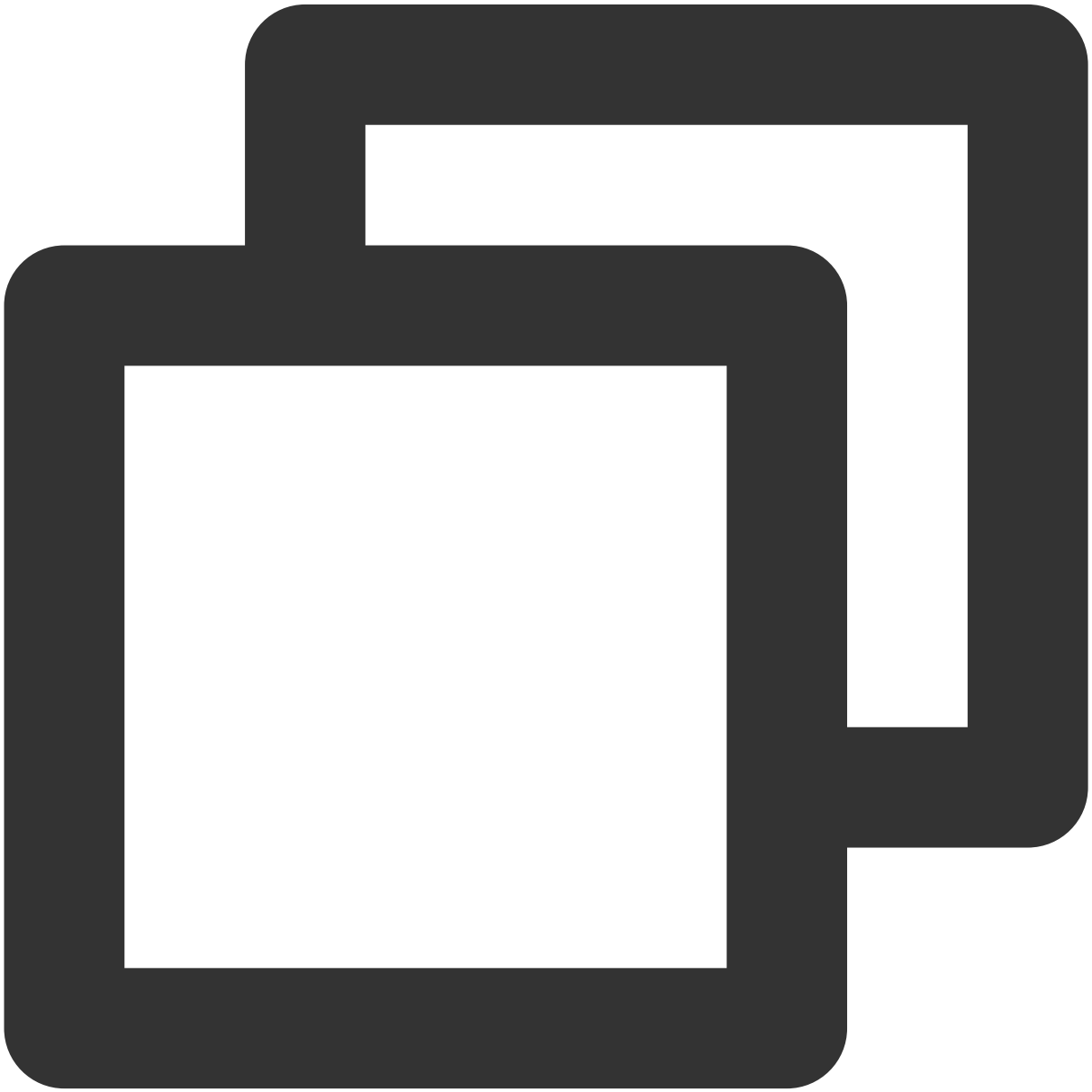
由于 TIMPush 当前尚不支持 iOS 的 VoIP 推送功能，我们为您提供了一个免费的替代方案，以帮助您在 iOS 设备上实现 VoIP 推送：VoIP 离线唤醒配置流程。

Android

您可以使用我们的 TIMPush 插件（付费）来实现 Android 端的 VoIP 通知。

集成消息推送插件

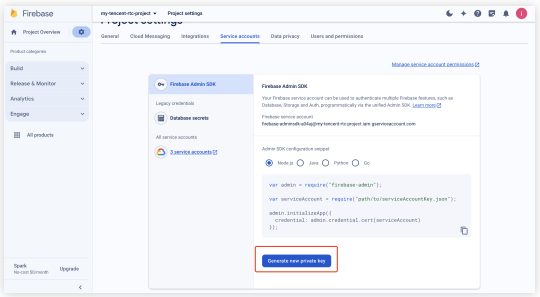
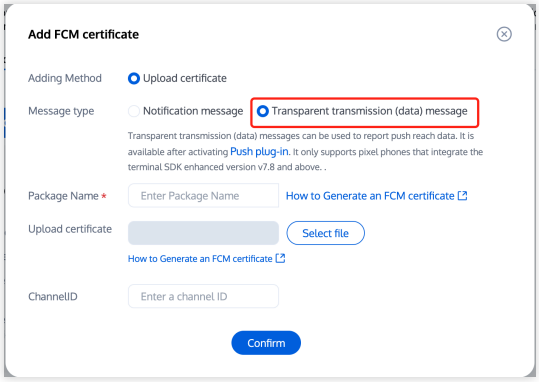
本插件在 pub.dev 的包名为: tencent_cloud_chat_push , 您可以收到将其引入 pubspec.yaml 依赖目录中, 也可以执行下列命令, 自动安装。



```
flutter pub add tencent_cloud_chat_push
```

准备条件

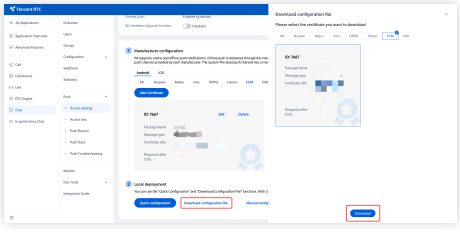
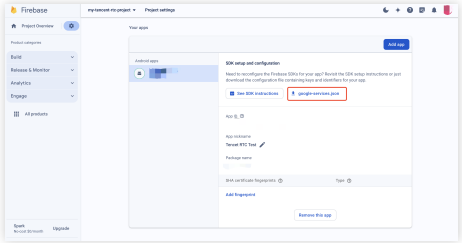
1. 将您自己的应用注册到 FCM推送平台，得到 **AppID** 和 **AppKey** 等参数以及 `google-services.json` 文件，来实现离线推送功能。
2. 登录 Tencent RTC 控制台，选择您的应用， 在 **Chat > Push > Access settings > Android** 功能栏，选择 FCM，添加 FCM 的证书，其中，消息类型选择**Transparent transmission (data) message**。

厂商推送平台	Chat 控制台配置
	

快速接入

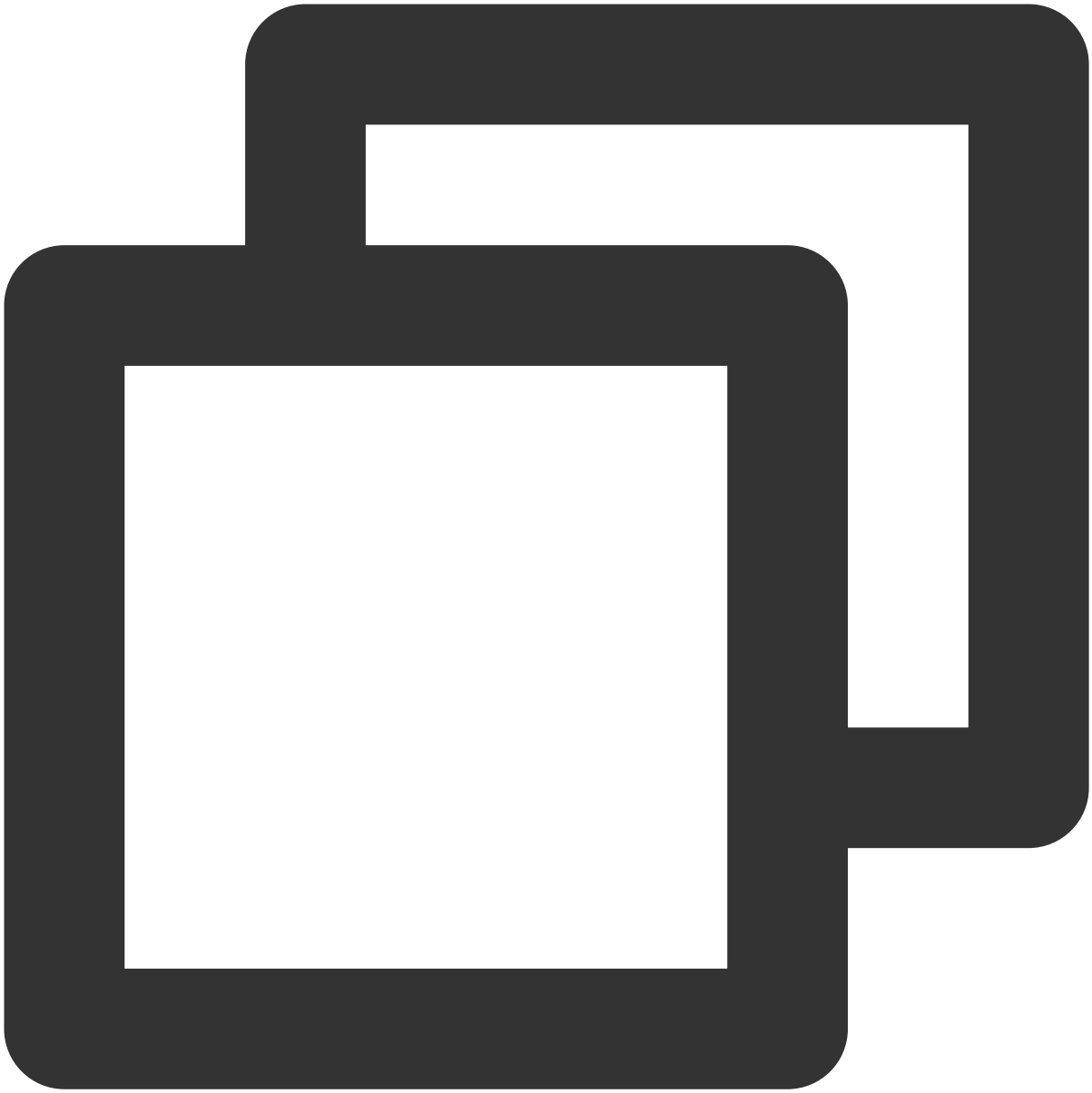
1. 下载并添加配置文件

完成控制台厂商推送信息填写后，下载并添加配置文件到工程。将下载的 `timpush-configs.json` 文件添加到应用模块的 `assets` 目录下，将 `google-services.json` 添加到工程app目录下。

选择下载配置文件 timpush-configs.json	下载文件 google-services.json	添加到
		

2. 集成 TIMPush 插件

在项目的 app 目录下的 `build.gradle` 文件中添加如下依赖：



```
implementation "com.tencent.timpush:fcm: xxxxxx"
```

说明：

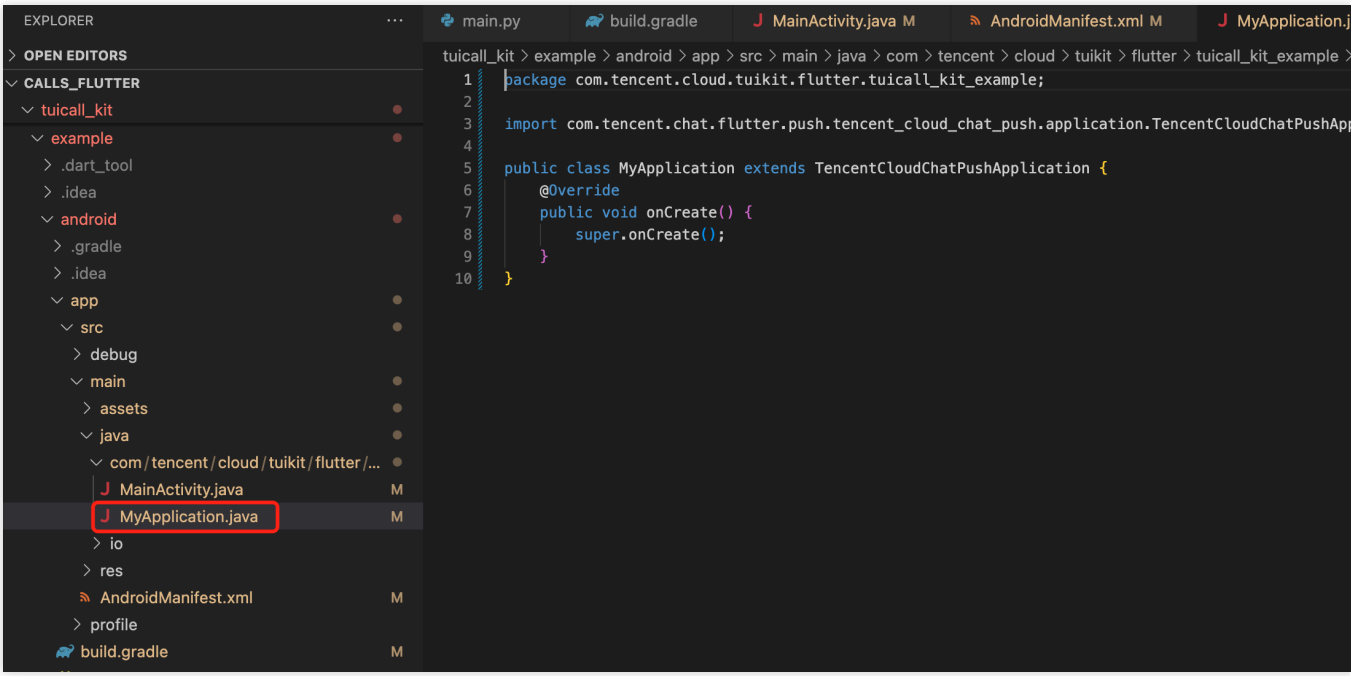
TIMPush 需要集成 Chat SDK 在**7.9.5666**版本及以上。

在 `build.gradle` 中添加的依赖的版本需要与 `tencent_cloud_chat_push` 版本对应。

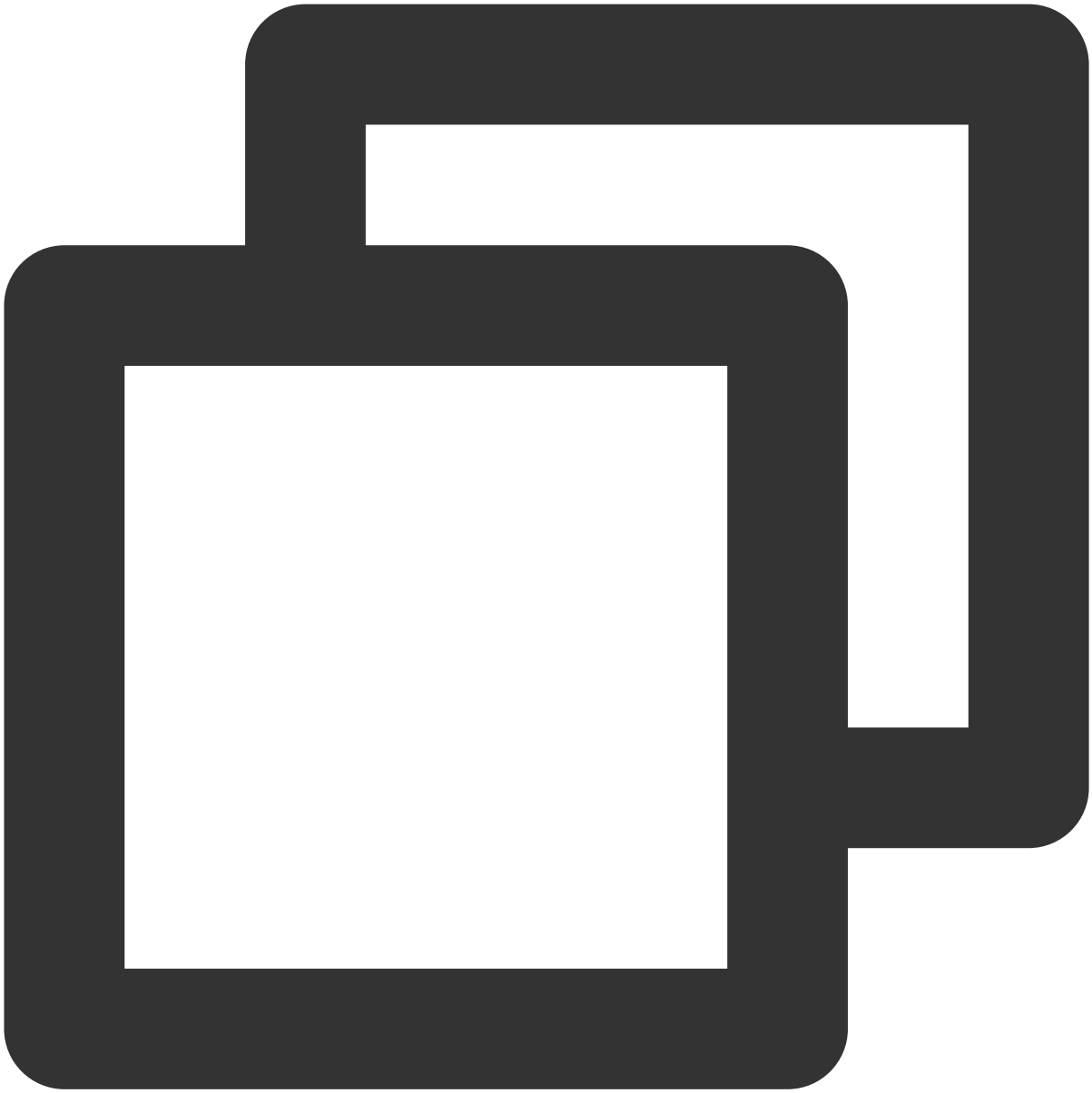
3. 客户端代码配置

在您项目 android 路径下 `MainActivity` 同级目录中，新建一个新的 Application 文件类, 例如可命名为 `MyApplication` 。

如果您已经自定义了一个 Application 类，则可直接复用，不需要再次创建。



将下列代码粘贴到该文件中，如上图所示：



```
package xxxx.xxxx.xx

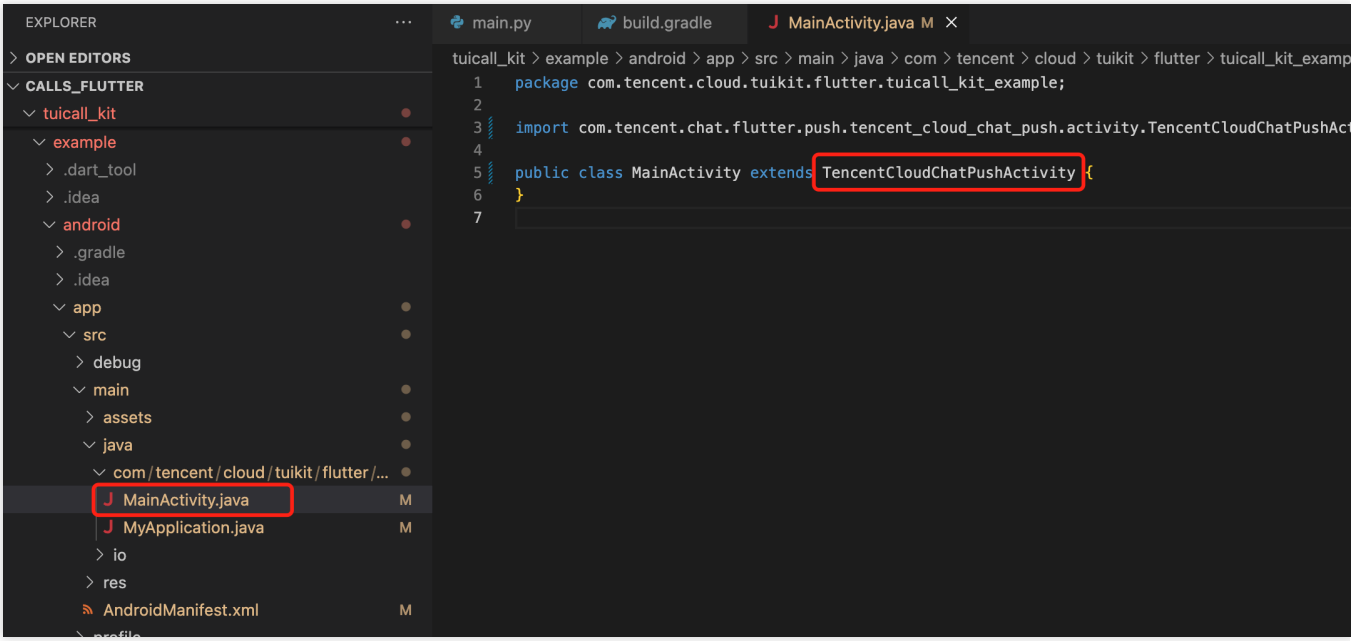
import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

public class MyApplication extends TencentCloudChatPushApplication {
    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

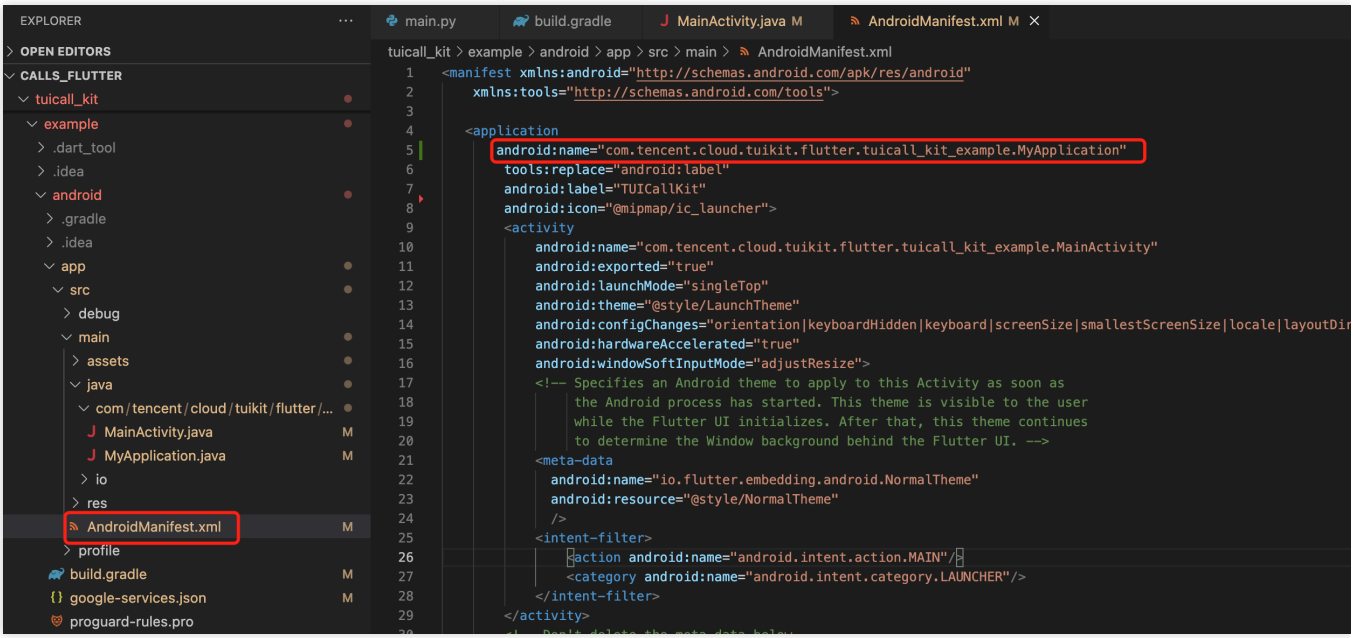
说明：

如果您已经创建了自己的 Application 为了其他用途，请直接 `extends TencentCloudChatPushApplication` 并保证 `onCreate()` 函数中，调用了 `super.onCreate();` 即可。

同时，您还需要修改您的 MainActivity 文件：

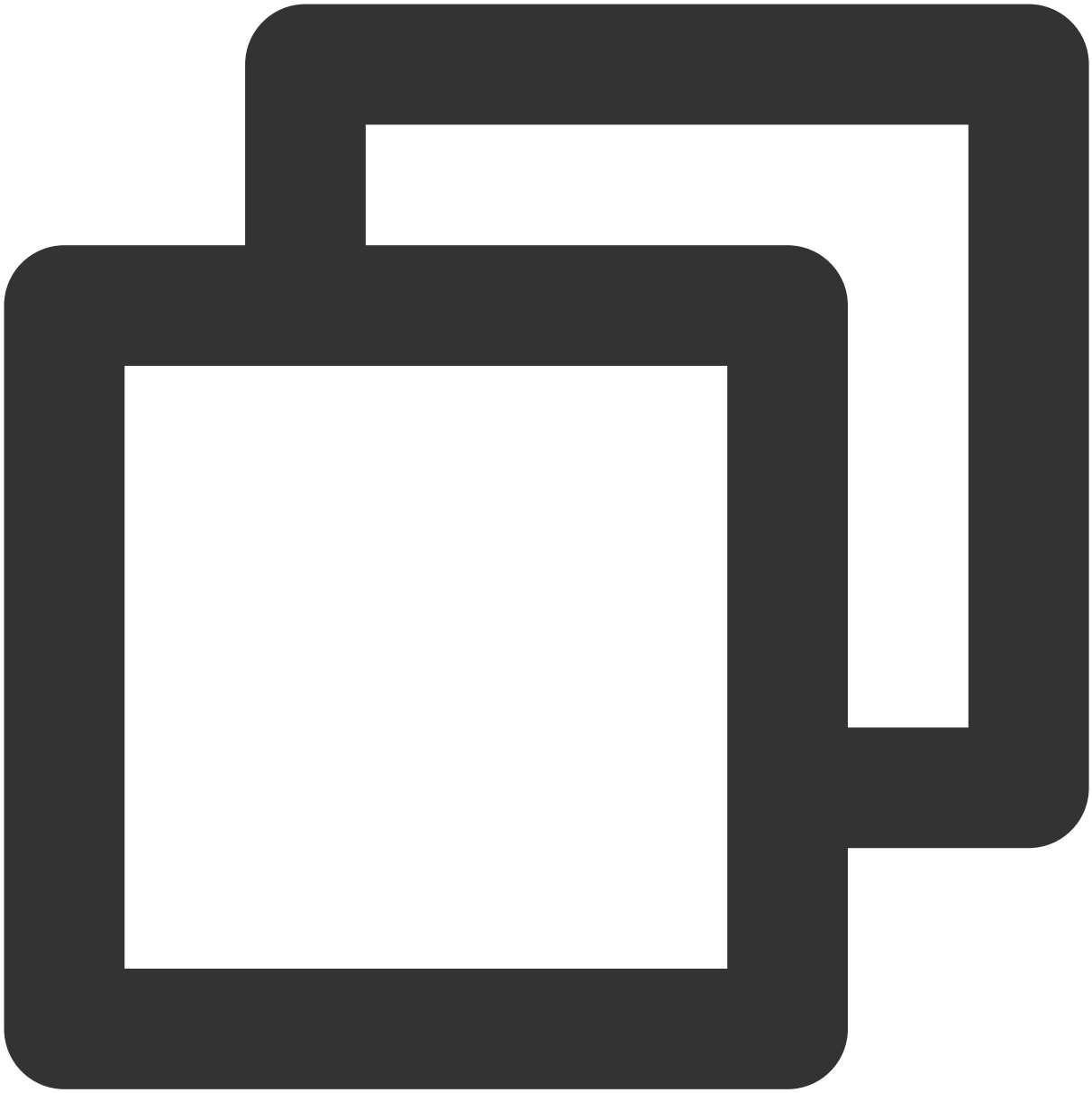


打开 `android/app/src/main/AndroidManifest.xml` 文件，为 `<application>` 标签，新增指定一个 `android:name` 参数即可，指向刚制作的自定义 `Application` 类。如图所示：



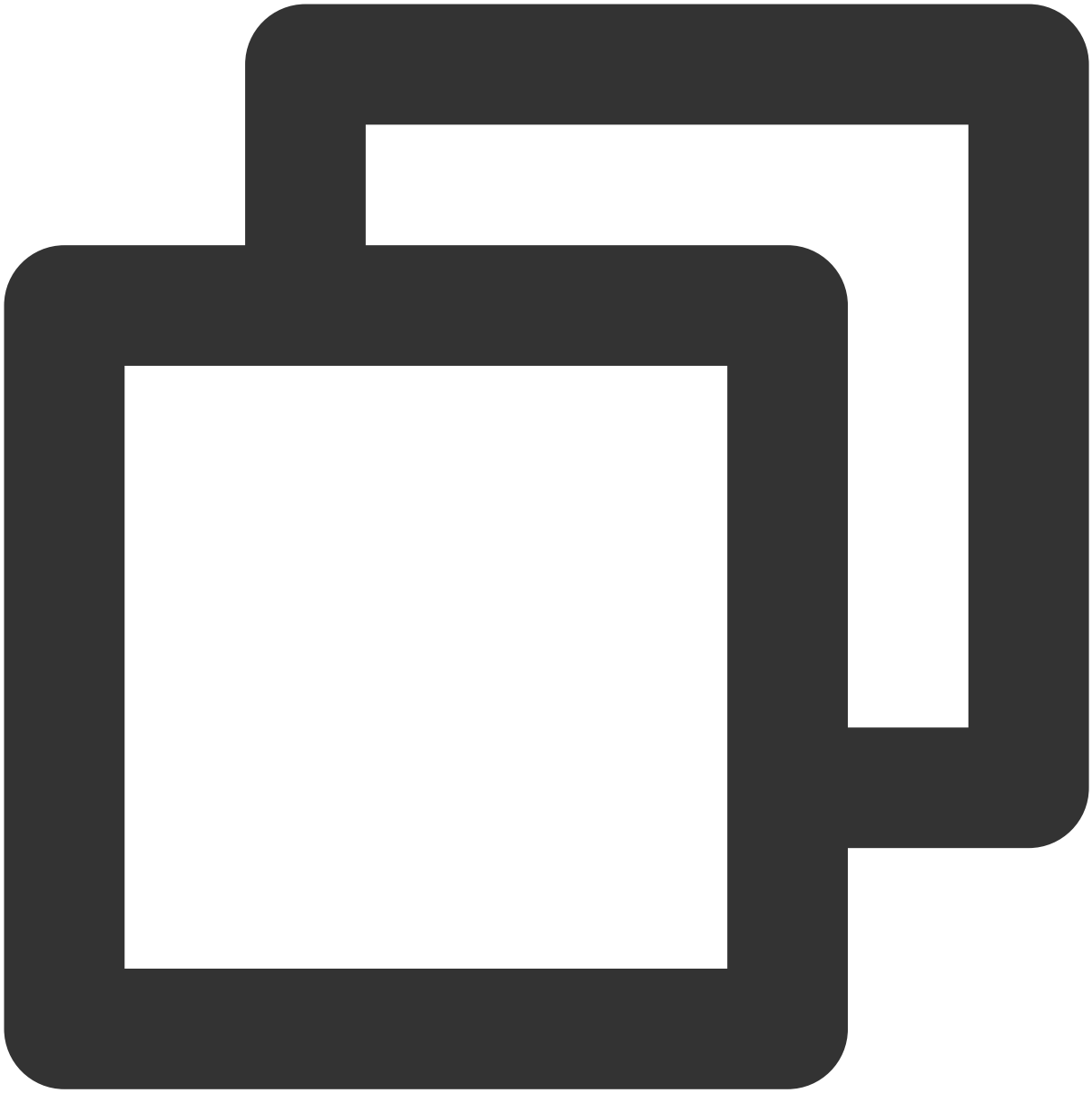
4. 完成工程配置

在项目级 `build.gradle` 文件的 buildscript -> dependencies 下添加以下配置：



```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.3.15'
    }
}
```

在项目的 app 目录下的 `build.gradle` 文件中添加下方配置：



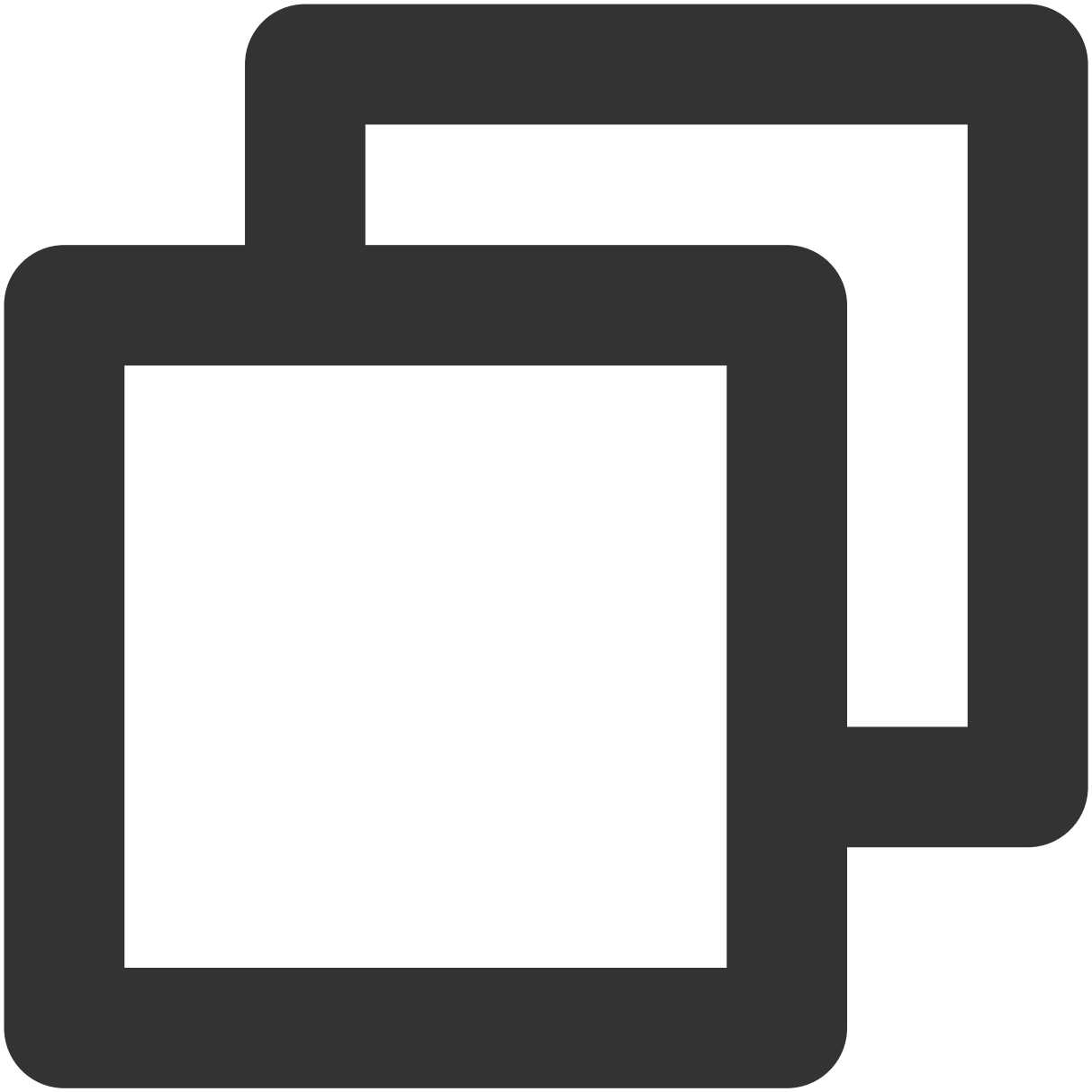
```
apply plugin: 'com.google.gms.google-services'
```

5. 注册推送插件

请在登录完成后，立即注册推送插件。

调用 `TencentCloudChatPush().registerPush` 方法，需传入上一步定义的点击回调函数。

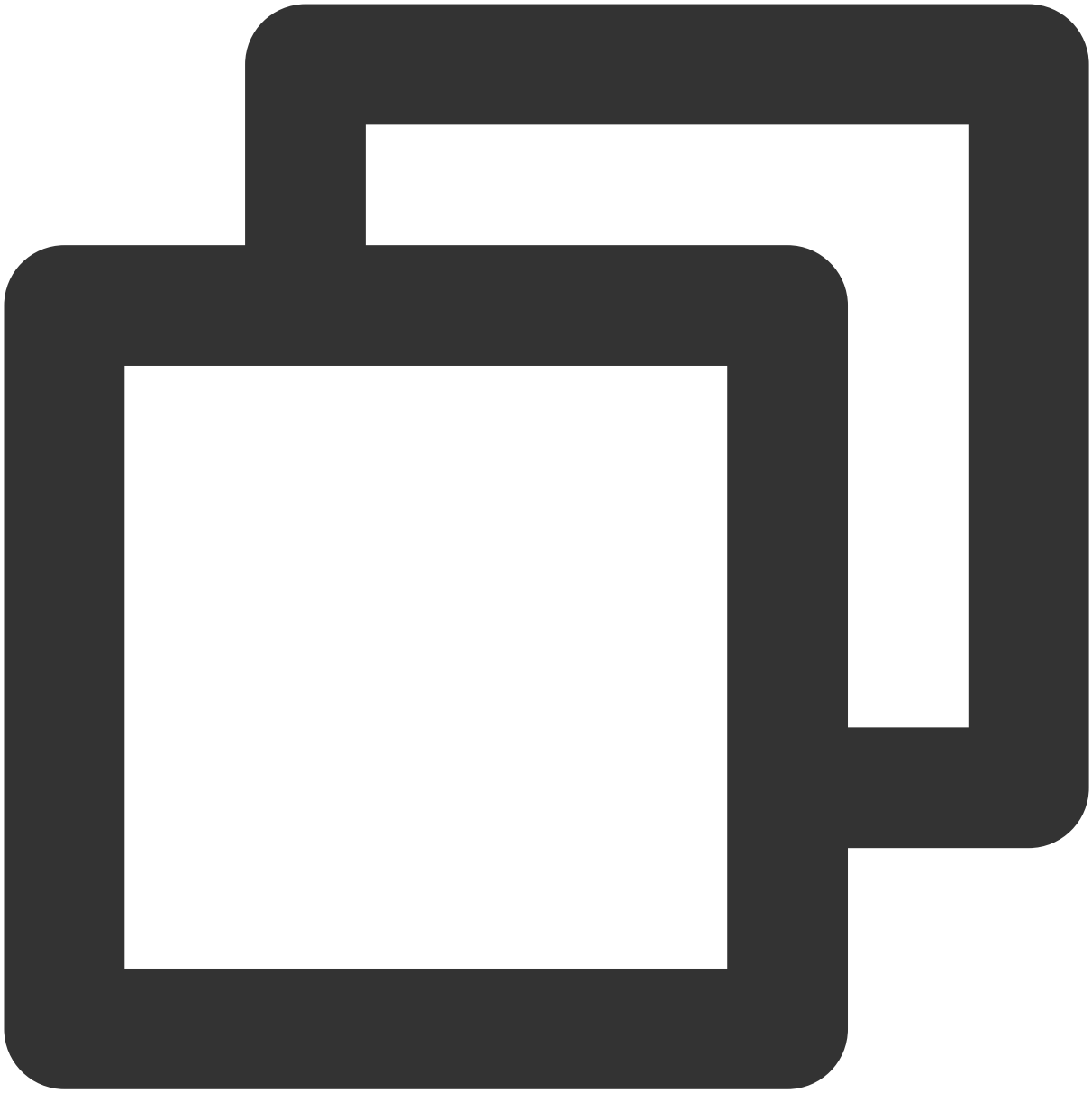
此外，您还可选传入 `apnsCertificateID` iOS 推送证书 ID 及 `androidPushOEMConfig` Android 推送厂商配置。此二项配置已在前序步骤指定，若无修改必要，可不再传入。



```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

6. 实现自动登录

FCM的data模式只能将离线app唤醒，所以还需要您在 `onAppWakeUpEvent` 中实现登录以及拉起APP等操作。

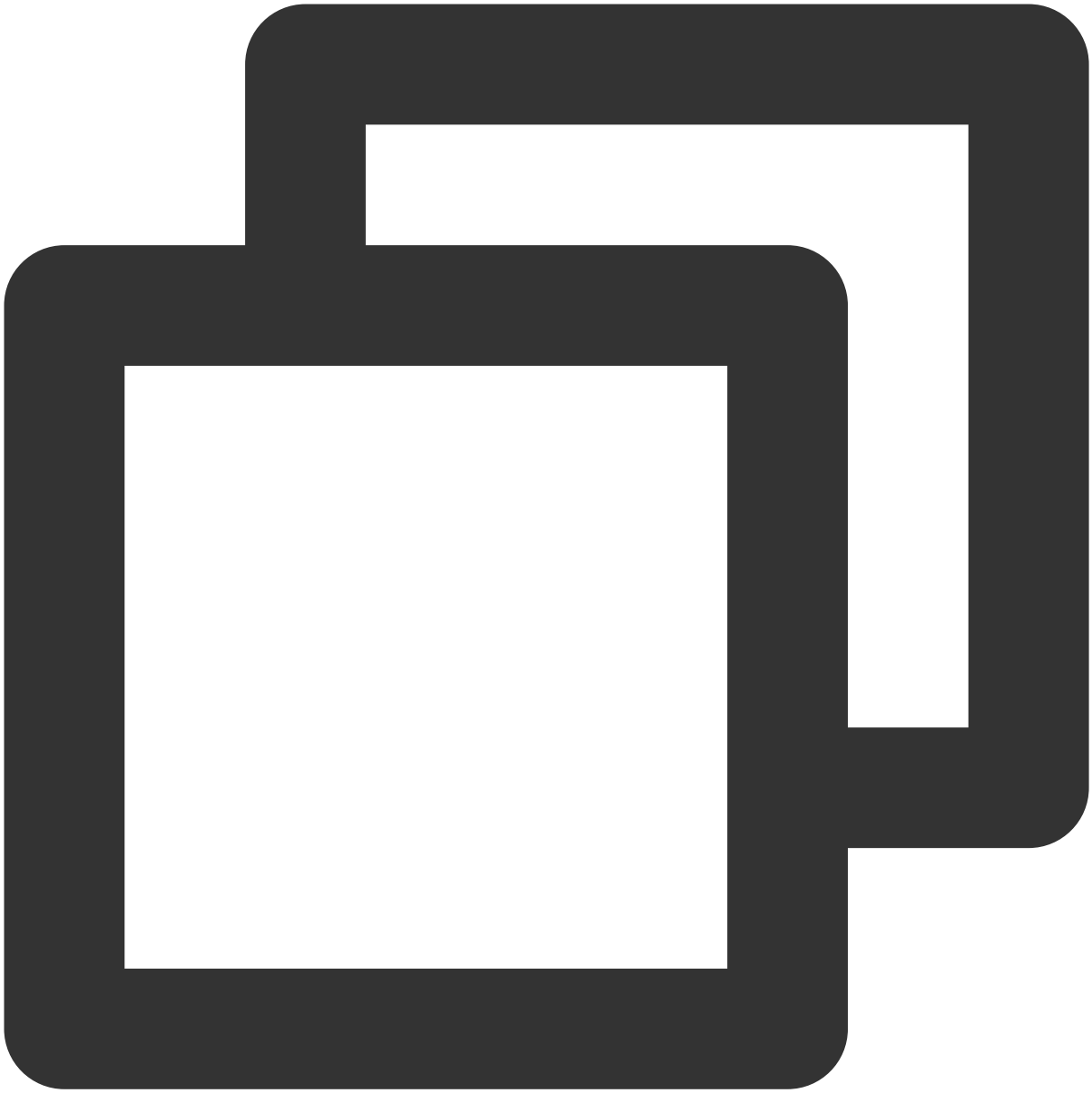


```
TencentCloudChatPush().registerOnAppWakeUpEvent(onAppWakeUpEvent: () {
      // TODO: 登录操作
});
```

完成上述步骤后，您可以结合 TUICallKit 使用 TIMPush 的离线推送能力。

7. 拨打离线推送通话

如果您要拨打离线推送通话，需要在调用 call 时候设置 OfflinePushInfo 。



```
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.androidFCMChannelID = "fcm_push_channel";
 offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.VoIP;

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

版权所有：腾讯云计算（北京）有限责任公司

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

说明：

如果您的 Android 程序在接收推送或拉起页面时遇到问题，您可以参见 被叫端来电显示策略 对问题进行排查。

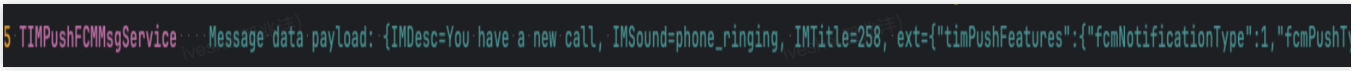
常见问题

应用被终止后，无法显示来电 UI?

确认收到了推送，收不到推送需要确认下 IM 控制台是否正确上传证书。参见文档上述快速接入的 第一步，看下是否添加正确。

确认控制台选择了 FCM 数据消息，对照上述准备条件的 第二步.

确认收到数据消息，过滤日志（关键字：TIMPush），检查下述日志有打印（如果收不到消息，可以使用 排查工具 查看原因）：



确认实现了**自动登录**。自动登录后才会去拉通话请求，才能显示来电 UI。

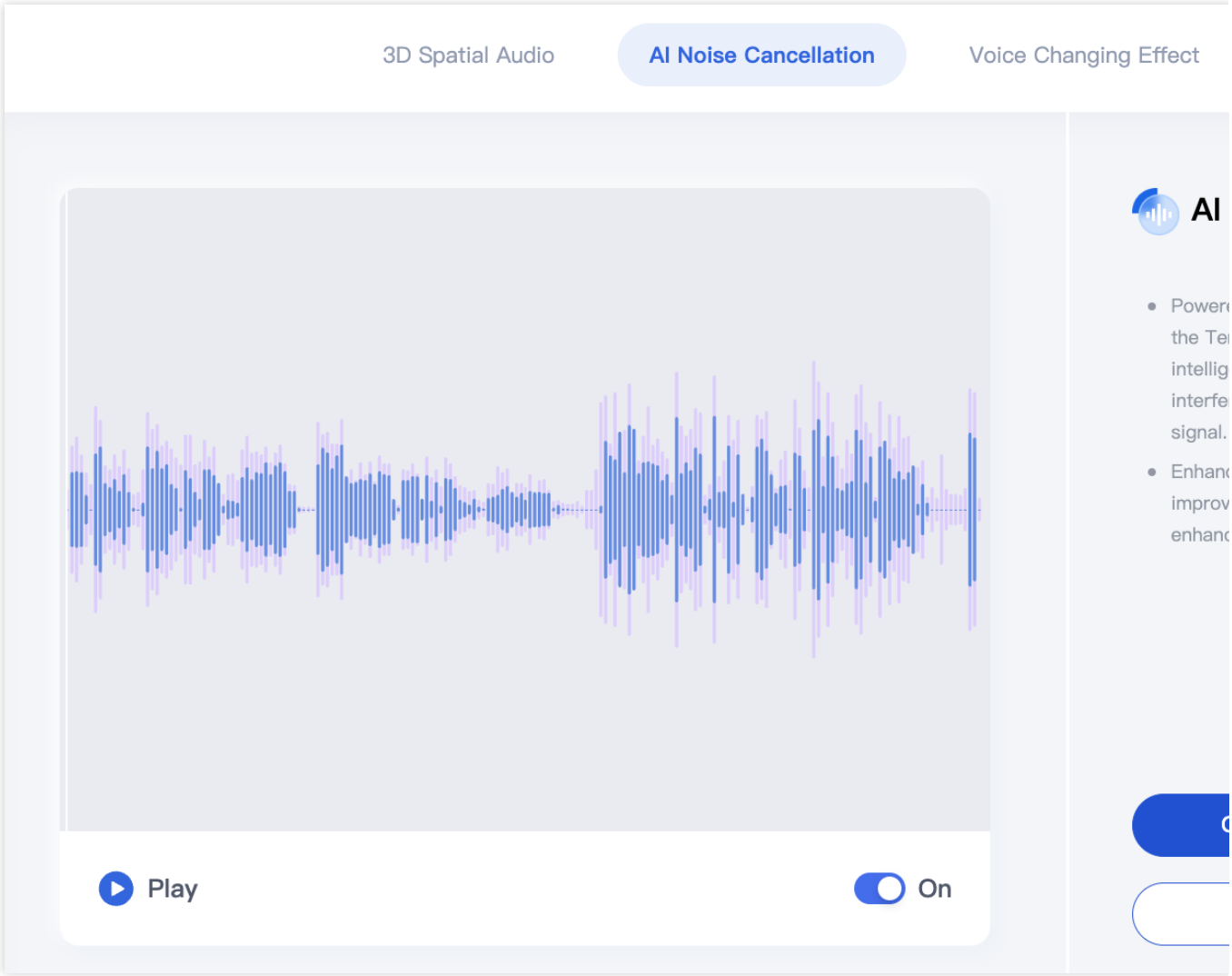
AI 降噪（TUICallKit）

最近更新时间：2024-06-17 17:41:52

AI 降噪源自腾讯天籁实验室的 AI 算法，能够智能检测和去除混合在传播信号中的噪声干扰，从而显著提高语音的质量，提升声音的清晰度，改善用户听感。
AI 降噪特性在 TUICallKit 中的应用，可以让用户在办公室、网吧、商场、户外等各种环境下都能够获得清晰、稳定的声音体验。

在线体验

您也可以进入我们的 Tencent RTC Demo，在线体验 AI 降噪能力带来的优秀声音效果。



开通 AI 降噪

TUICallKit 目前已默认开通 AI 降噪功能，用户无需进行额外的设置或操作，即可在应用中享受高质量的降噪效果。

版权所有：腾讯云计算（北京）有限责任公司

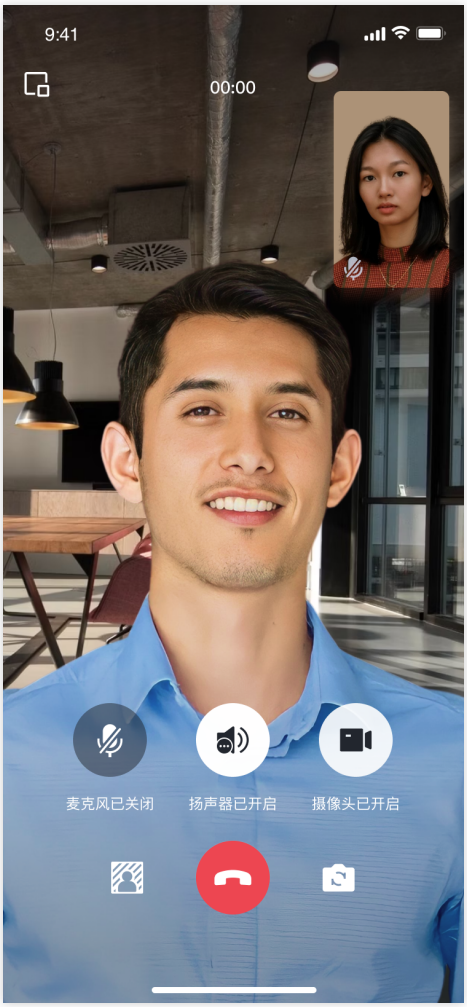
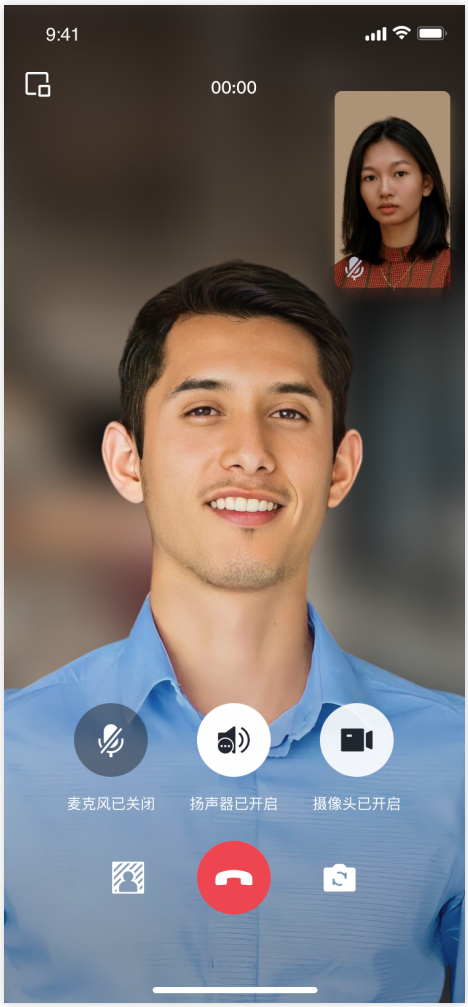
虚拟背景（TUICallKit）
iOS

最近更新时间：2024-07-22 17:00:45

TUICallKit 推出了虚拟背景新特性，让用户在视频通话时，可设置模糊背景或图片背景，隐藏真实通话环境，保护隐私，让通话更有趣。 接下来，本文将详细介绍在 TUICallKit 组件中如何使用这一特性。

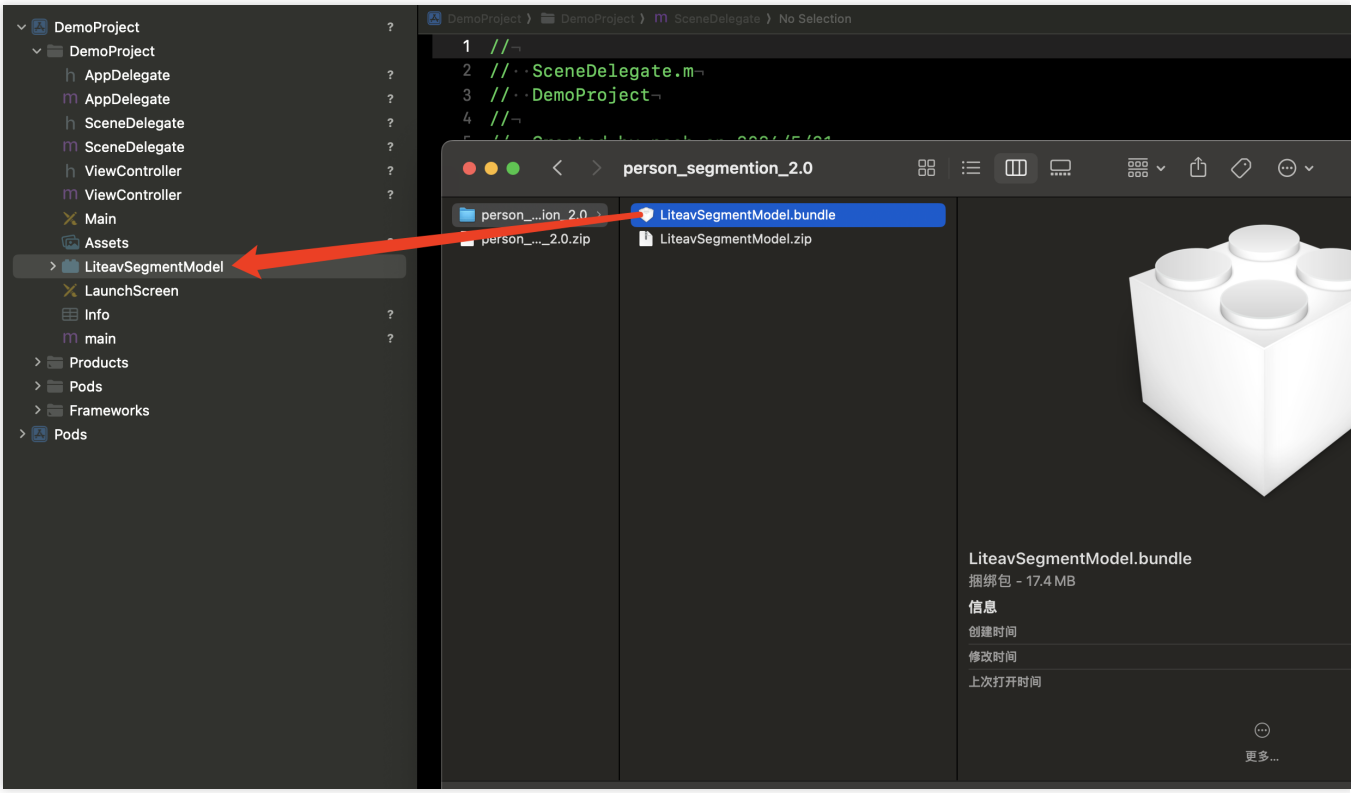
集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

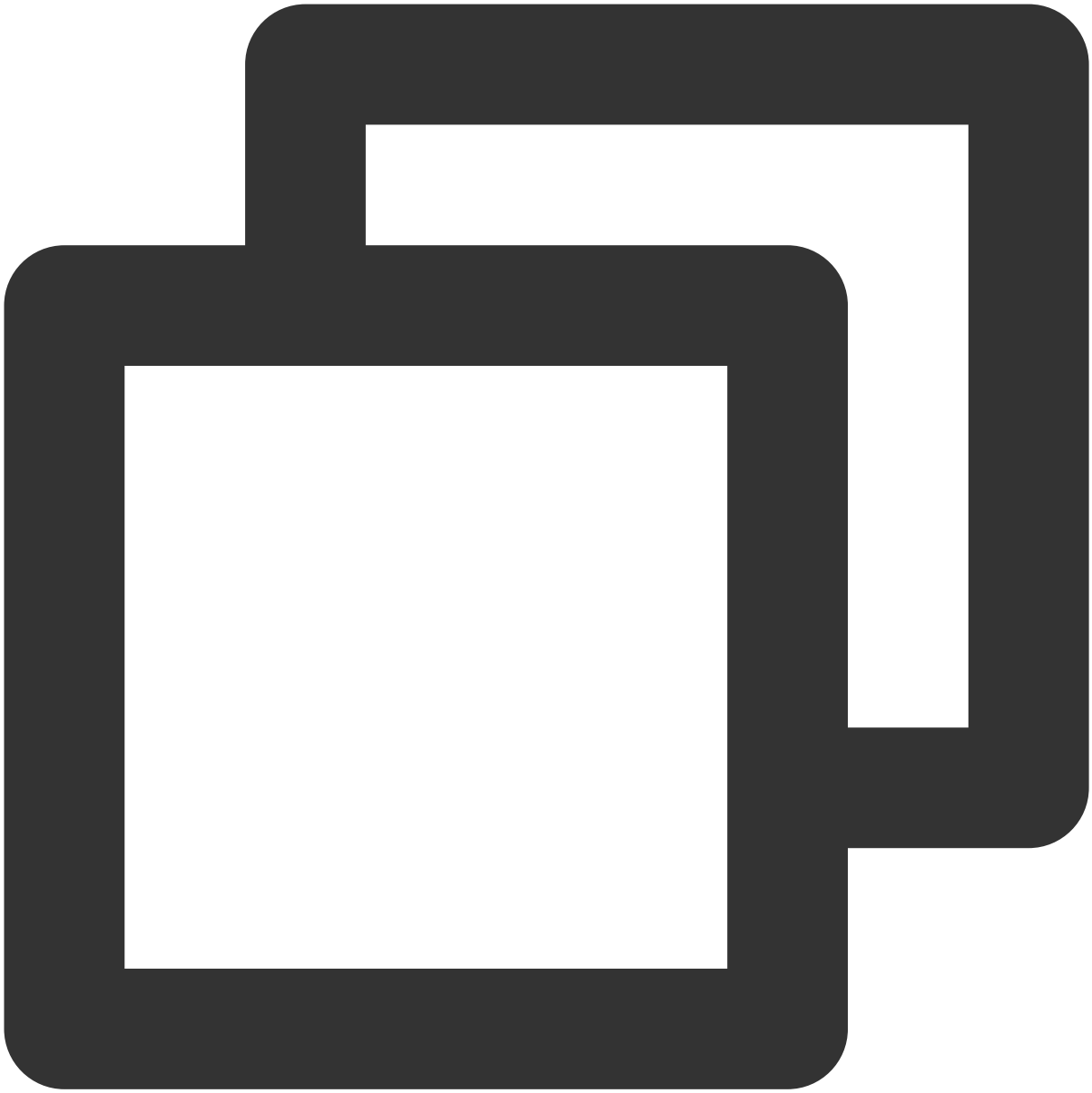
原始摄像头	模糊背景效果	图片
		

准备条件

1. 在使用腾讯云提供的虚拟背景功能前，您需要前往控制台，为应用开通音视频服务，并购买 **群组通话版（140万分钟）** 套餐，具体步骤请参见 开通服务。
2. 下载 虚拟背景模型 文件，解压后将 `LiteavSegmentModel.bundle` 文件拖拽到您的工程中，并确保它在 `MainBundle` 中。



3. 修改您的 Podfile 文件中的依赖，替换为 `Professional` 版本，并指定 TXLiteAVSDK_Professional 版本为 `11.8.15669` 。



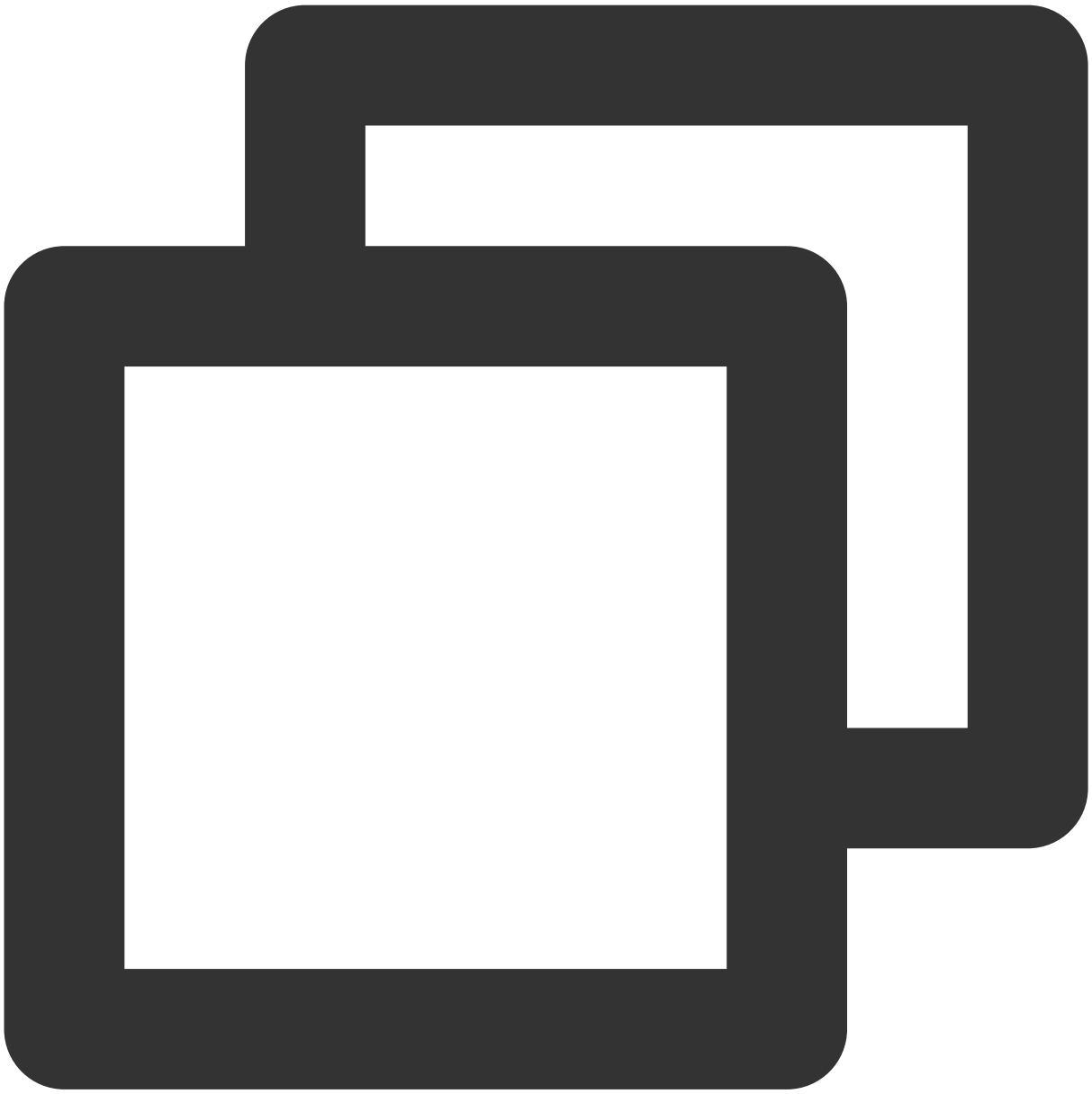
```
pod 'TUICallKit_Swift/Professional'
pod 'TXLiteAVSDK_Professional', '11.8.15669'
```

注意：

TRTC SDK 的版本与模型文件间有对应关系，请确保版本号与模型相匹配，详见下方：模型文件与SDK的匹配。

开启模糊背景

TUICallKit 的 UI 方案支持设置模糊背景。通过调用以下接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

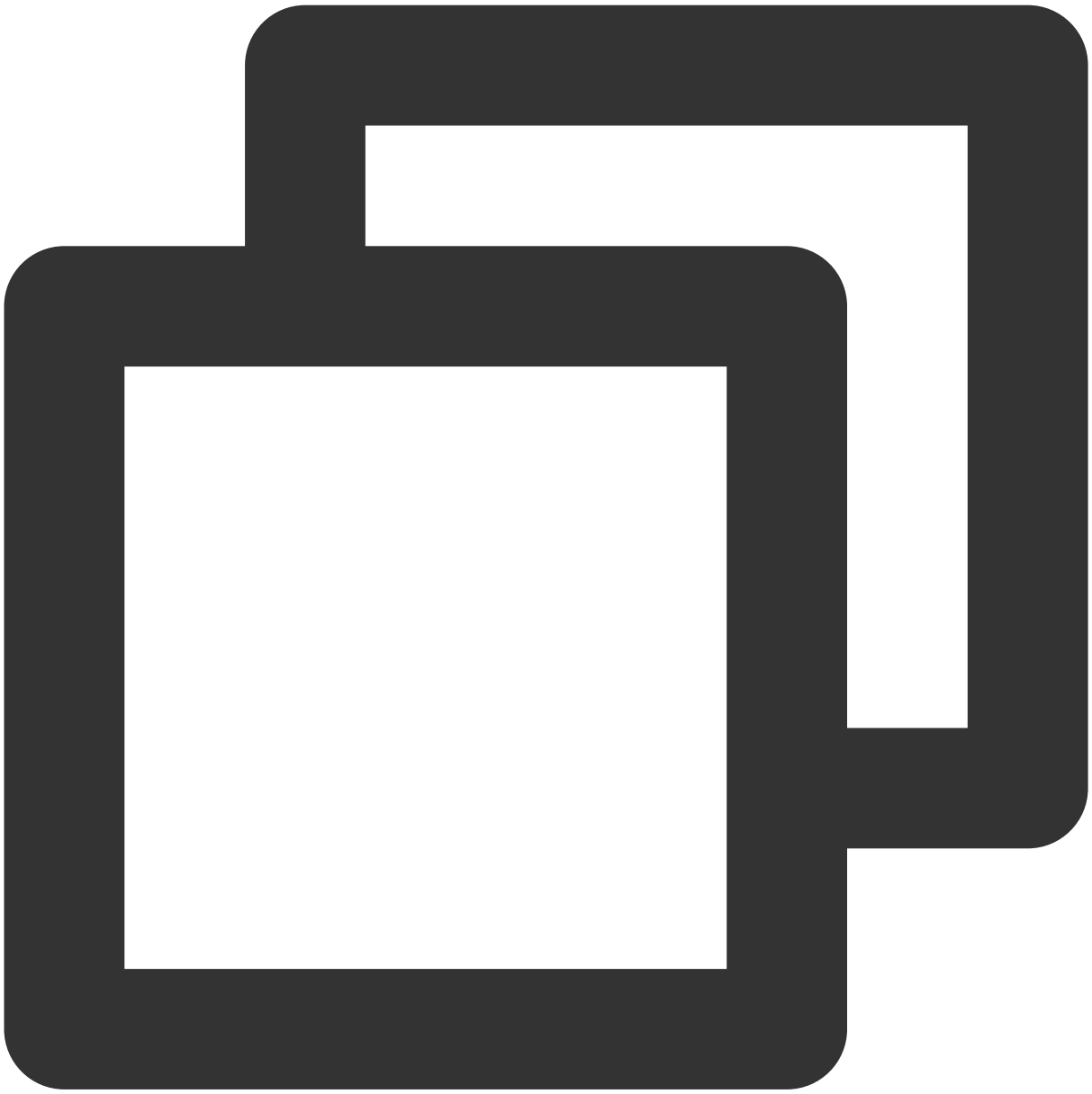


```
import TUICallKit

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

设置图片背景（可选）

图片背景需要用户自行实现，将图片保存在本地后，调用以下接口进行设置（目前只支持本地路径的图片，网络图片暂时不支持）。



```
import TUICallEngine

TUICallEngine.createInstance().setVirtualBackground("imagePath") { code, message in
}
```

常见问题

开启模糊背景无反应或有延迟？

确保已购买音视频通话 `Group Call` 套餐，详见 开通服务。
确保已下载模型文件至本地。
如果没有添加模型文件到本地路径中，在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。
检查模型文件与 SDK 是否匹配。

模型文件与 SDK 的匹配？

TUICallKit 是基于 Chat SDK 和 TRTC SDK 实现的音视频通话场景，虚拟背景是 TRTC SDK 提供的特色功能。要注意的是，虚拟背景模型文件与 TRTC SDK 的版本需要匹配，否则可能会导致模糊背景功能无法正常使用。下表列出了模型文件与 TRTC SDK 的对应关系：

SDK 版本	虚拟背景模型文件下载地址·
pod 'TXLiteAVSDK_Professional', '11.8.15669'	version_2.0 下载
pod 'TXLiteAVSDK_Professional', '11.7.12001'	version_1.0 下载

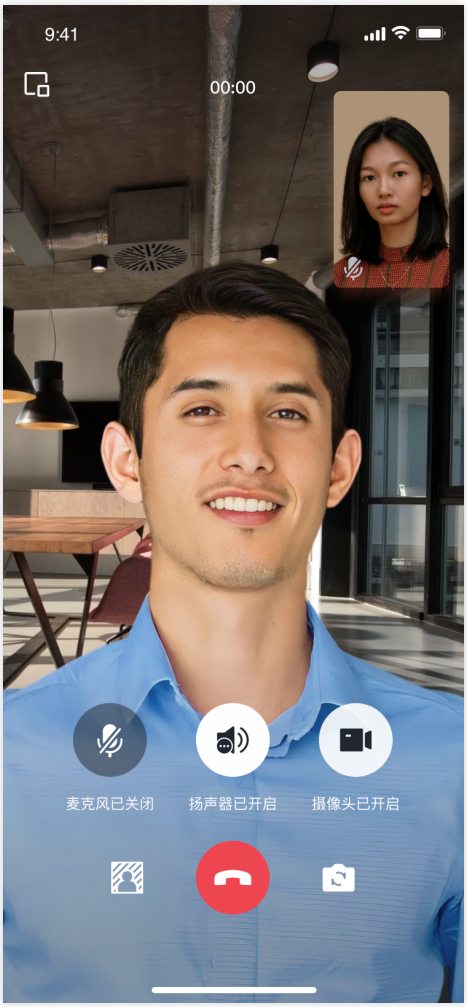
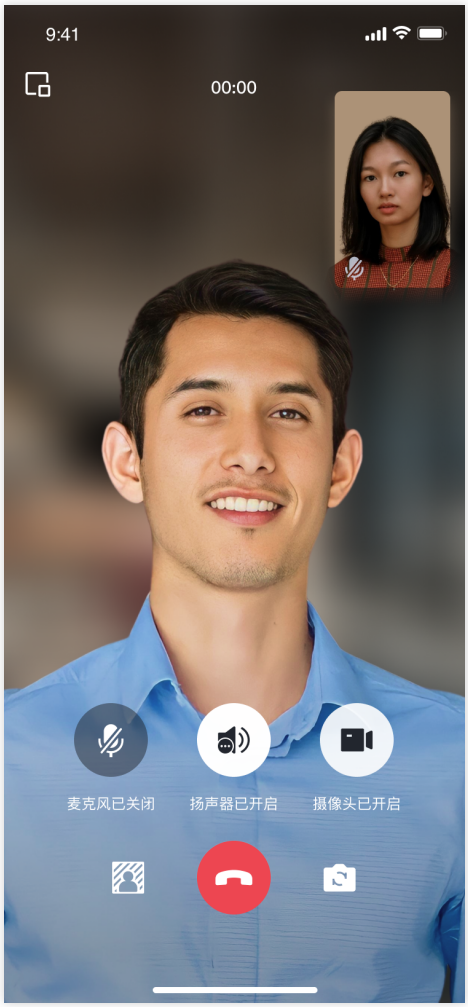
Android

最近更新时间：2024-07-22 17:00:45

TUICallKit 推出了虚拟背景新特性，让用户在视频通话时，可设置模糊背景或图片背景，隐藏真实通话环境，保护隐私，让通话更有趣。 接下来，本文将详细介绍在 TUICallKit 组件中如何使用这一特性。

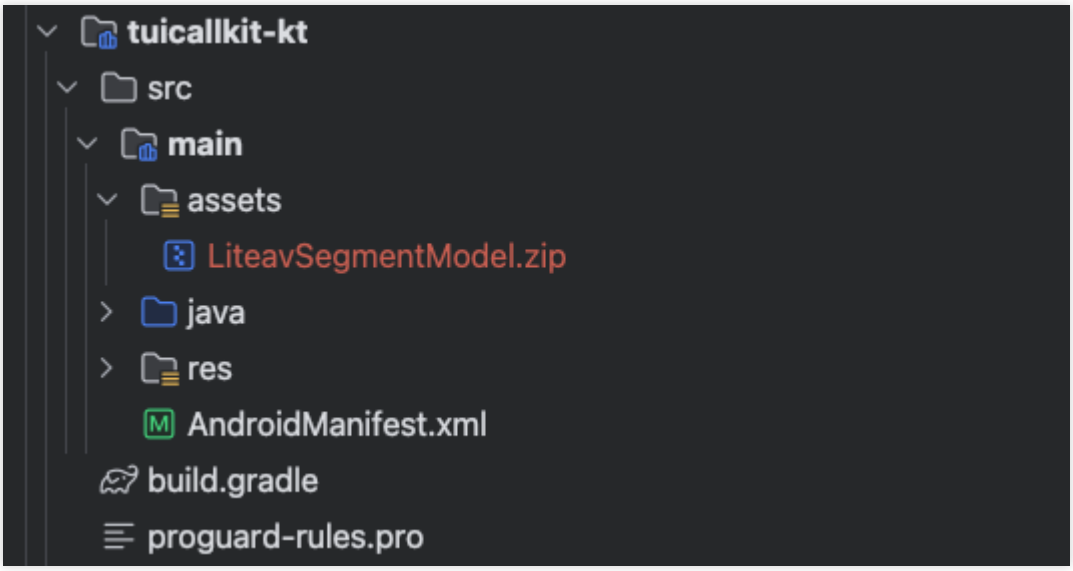
集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

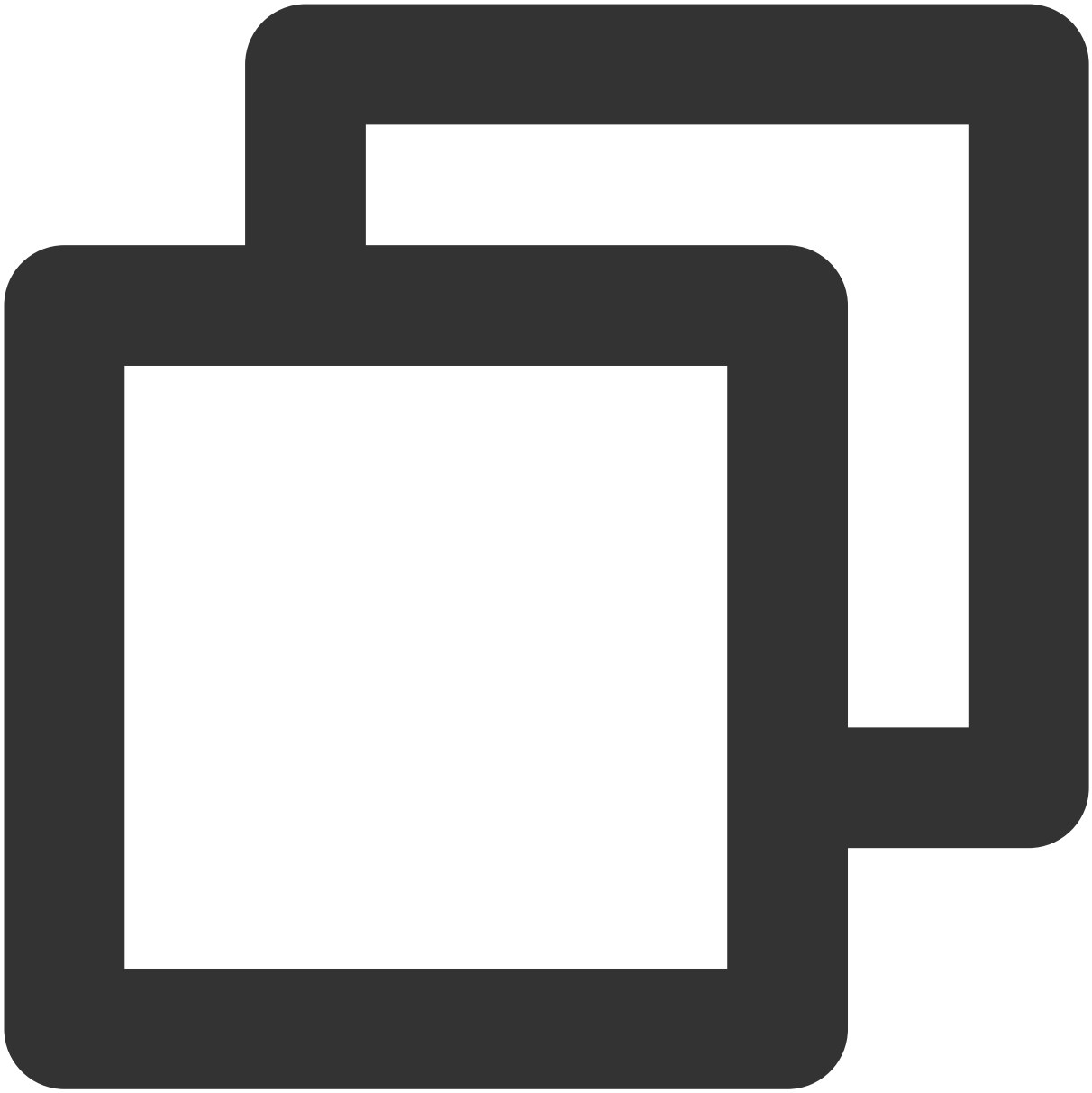
原始摄像头	模糊背景效果	图片
		

准备条件

1. 在使用腾讯云提供的虚拟背景功能前，您需要前往控制台，为应用开通音视频服务，购买 **群组通话版（140万分钟）** 套餐。具体步骤请参见 开通服务。
2. 下载 虚拟背景模型 文件，解压后，拷贝 `LiteavSegmentModel.zip` 文件到项目的 `assets` 目录下。



3. 在工程的 `tuicallkit-kt` 目录下，找到 `build.gradle` 文件，替换文件中的 `TRTC SDK` 版本为 Professional 版本。



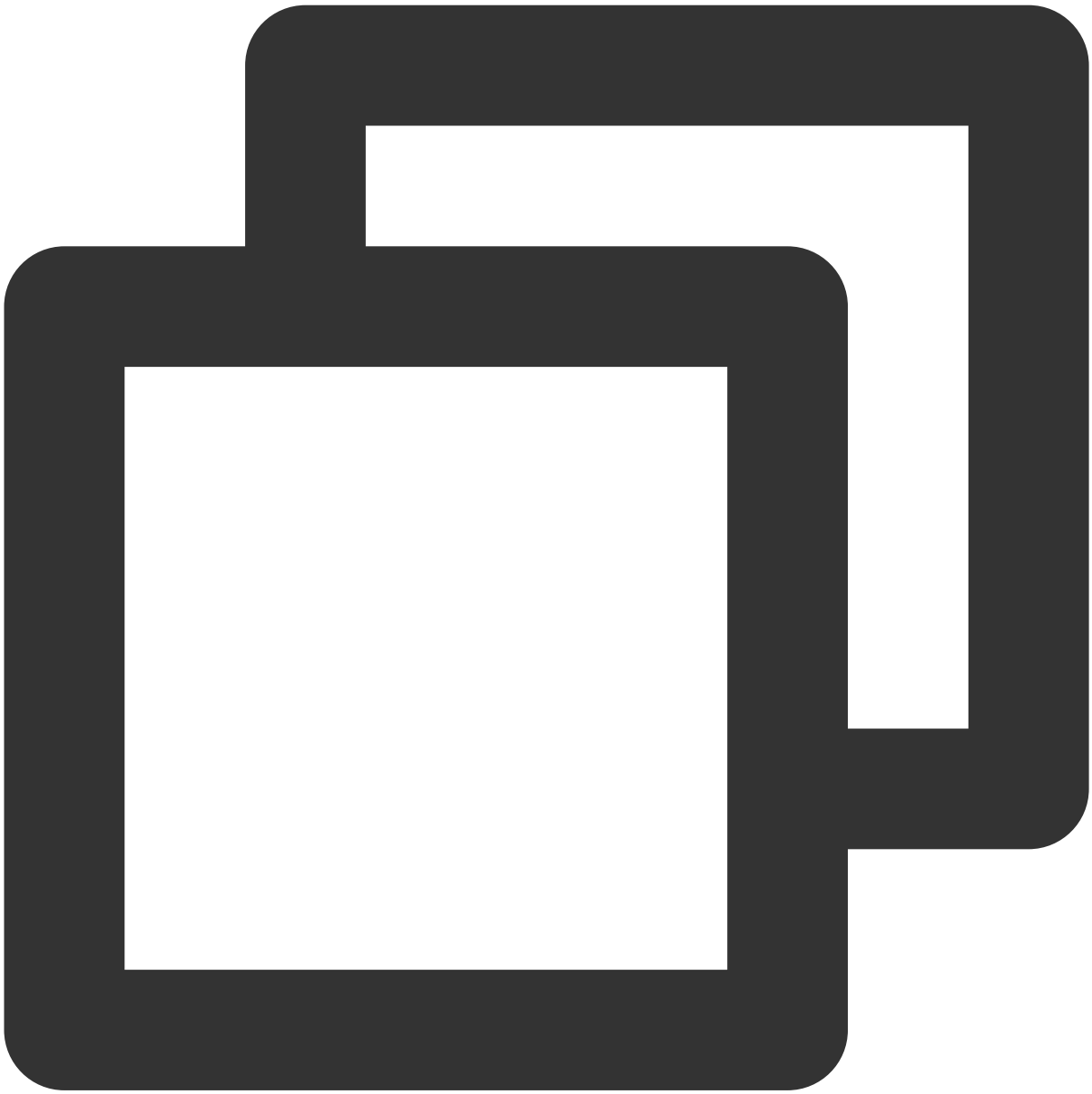
```
api "com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176"
```

注意：

TRTC SDK 的版本与模型文件间有对应关系，请确保版本号与模型相匹配，详见下方：模型文件与 SDK 的匹配。

开启模糊背景

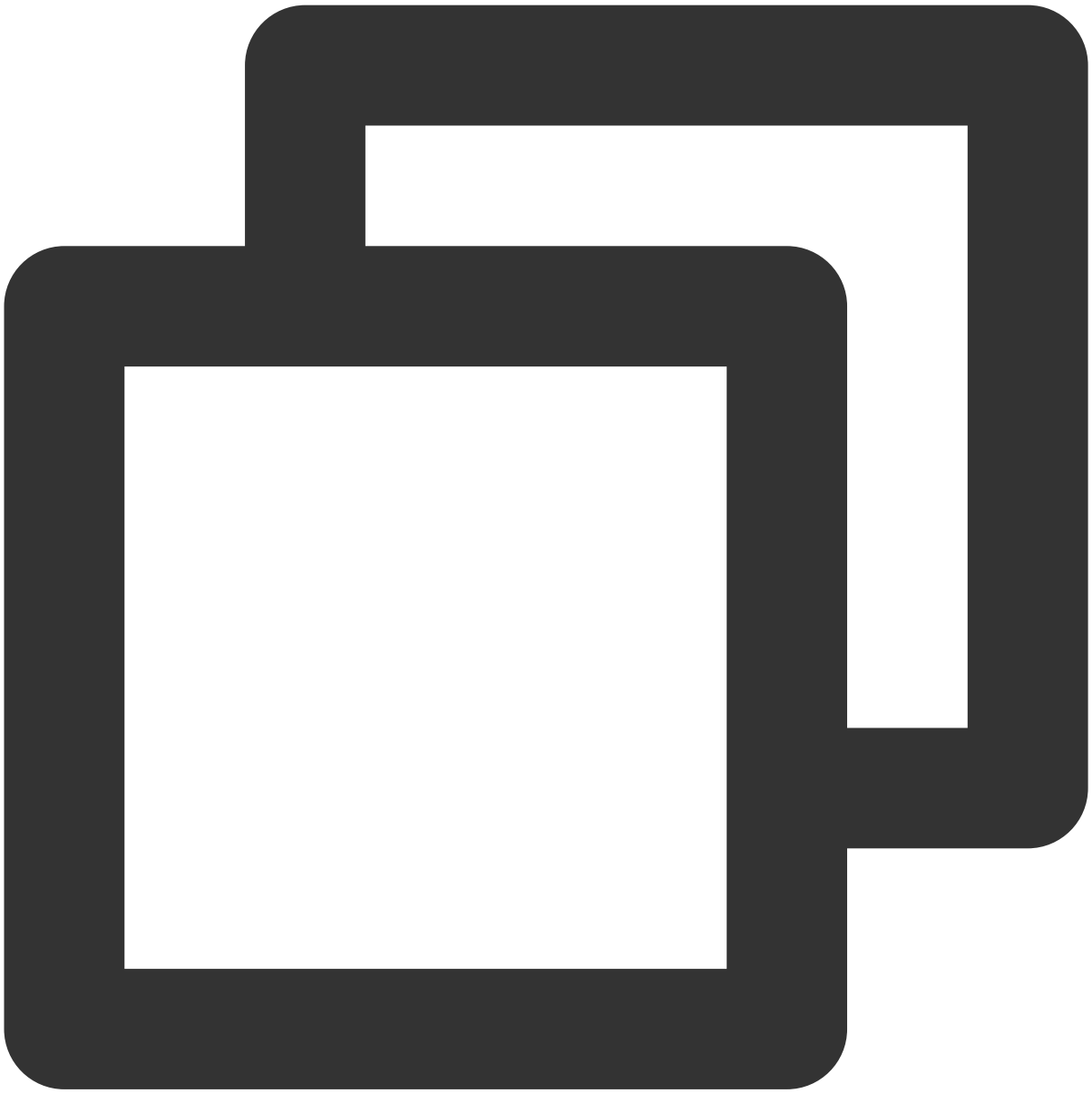
TUICallKit 的 UI 方案支持设置模糊背景。通过调用以下接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。



```
TUICallKit.createInstance(getApplicationContext()).enableVirtualBackground(true);
```

设置图片背景（可选）

图片背景需要用户自行实现，将图片保存在本地后，调用以下接口进行设置（目前只支持本地路径的图片，网络图片暂时不支持）。



```
TUICallEngine.createInstance(context).setVirtualBackground("imagePath", null)
```

常见问题

开启模糊背景无反应或有延迟？

确保已购买音视频通话 `Group Call` 套餐，详见 开通服务。
确保已下载模型文件至本地。
如果没有添加模型文件到本地路径中，在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。
检查模型文件与 SDK 是否匹配。

模型文件与 SDK 的匹配？

TUICallKit 是基于 Chat SDK 和 TRTC SDK 实现的音视频通话场景，虚拟背景是 TRTC SDK 提供的特色功能。虚拟背景模型文件与 TRTC SDK 的版本有匹配关系。若不匹配，可能导致开启模糊背景无效果。请查看下表了解模型文件与 TRTC SDK 的对应关系：

SDK 版本	虚拟背景模型文件下载地址
com.tencent.liteav:LiteAVSDK_Professional:11.7.0.12001	segmention_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmention_2.0

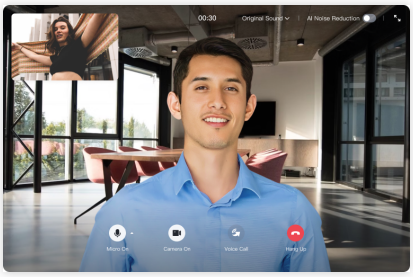
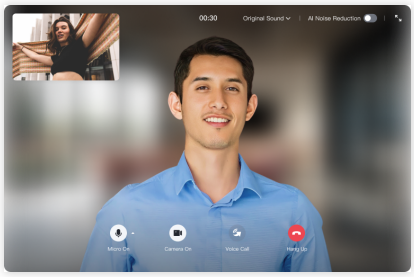
Web

最近更新时间：2024-07-09 16:08:59

TUICallKit 推出了虚拟背景新特性，让用户在视频通话时，可设置模糊背景或图片背景，隐藏真实通话环境，保护隐私，让通话更有趣。 接下来，本文将详细介绍在 TUICallKit 组件中如何使用这一特性。

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

在使用腾讯云提供的虚拟背景功能前，您需要前往控制台，为应用开通音视频服务，购买 **群组通话版（140万分钟）** 套餐。具体步骤请参见 开通服务。

开启模糊背景

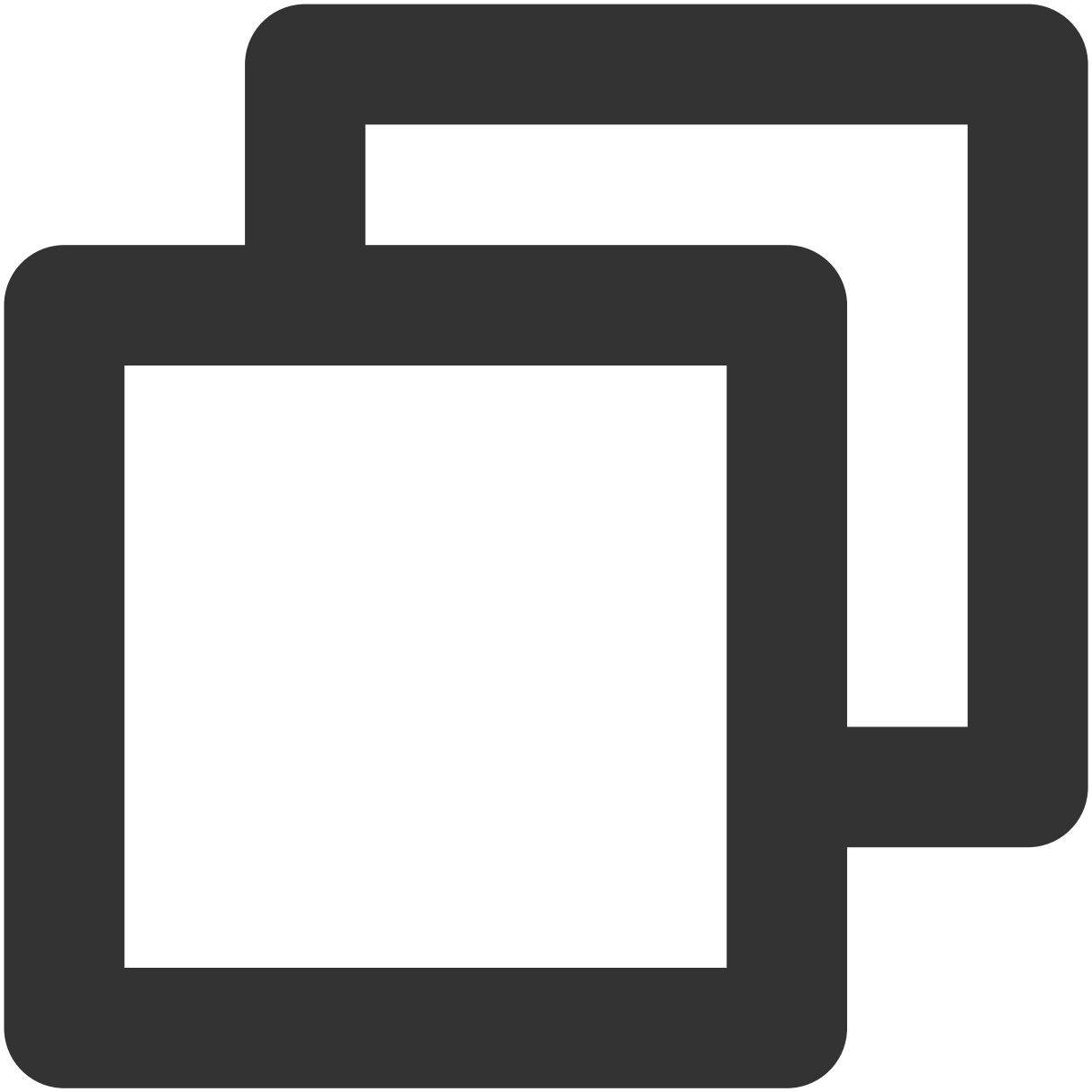
TUICallKit 的 UI 方案支持设置模糊背景。通过调用以下接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

注意：

H5 暂不支持，仅 PC 支持。

TUICallKit v3.2.4+ 支持。

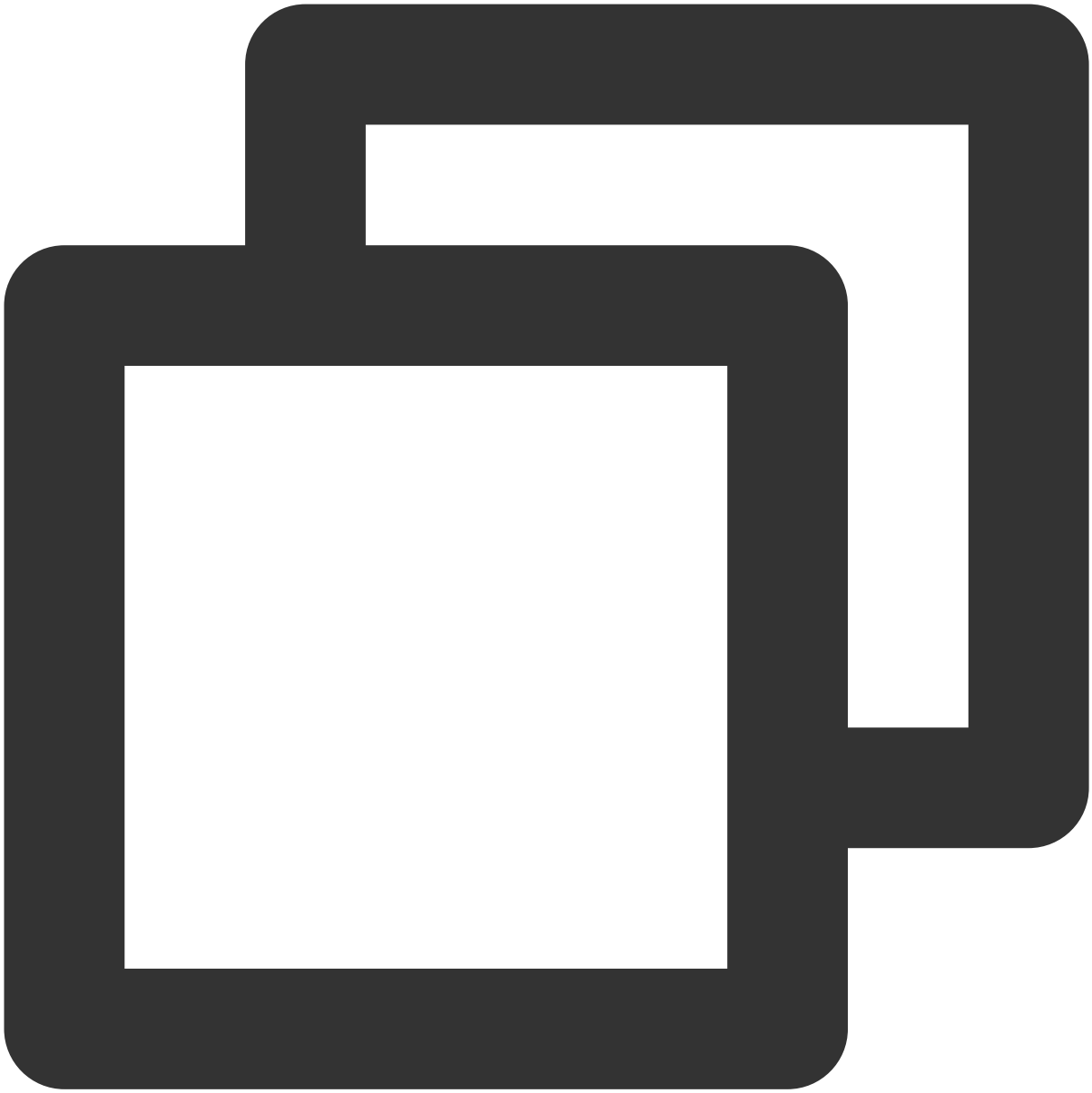
版权所有：腾讯云计算（北京）有限责任公司



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue"; // take @tencentcl
TUICallKitServer.enableVirtualBackground(true);
```

设置图片背景（可选）

图片背景需要用户自行实现，将图片保存在本地后，调用以下接口进行设置。网络图片也支持。



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-vue"; // take @tencentcl
TUICallKitServer.getTUICallEngineInstance().setVirtualBackground(imagePath: string)
```

常见问题

开启模糊背景无反应或有延迟？

确保已购买音视频通话 `Group Call` 套餐，详见 开通服务。

网络较差时，虚拟背景模型文件可能未下载完，从而导致开启虚拟背景失败。

关闭摄像头，是否能开启虚拟背景？

不能。

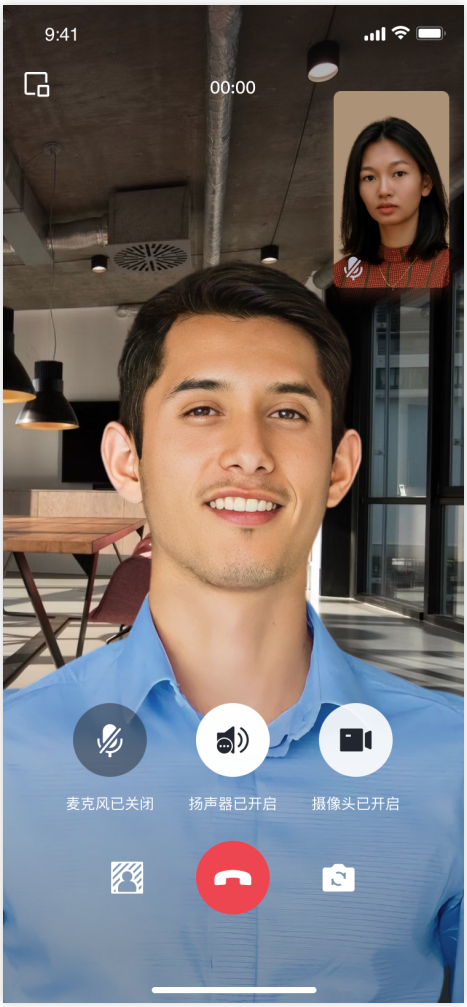
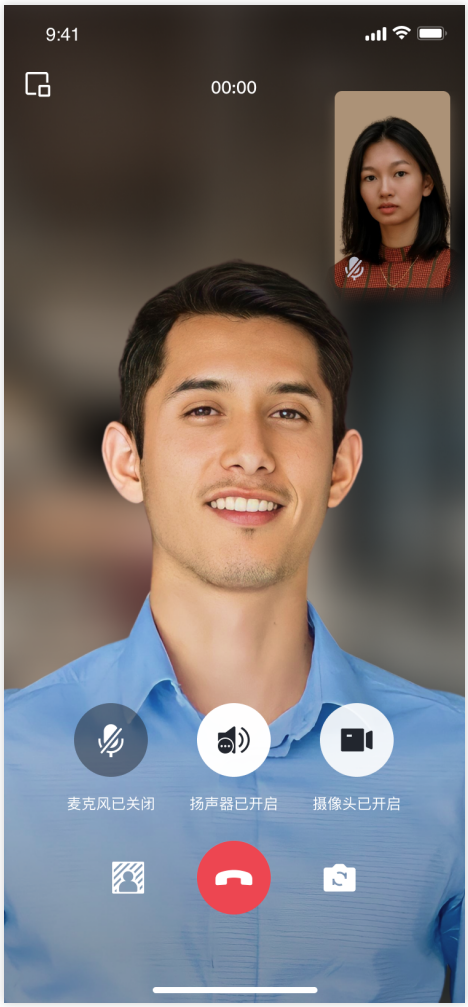
Flutter

最近更新时间：2024-07-22 17:00:45

TUICallKit 推出了虚拟背景新特性，让用户在视频通话时，可设置模糊背景或图片背景，隐藏真实通话环境，保护隐私，让通话更有趣。 接下来，本文将详细介绍在 TUICallKit 组件中如何使用这一特性。

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片
		

准备条件

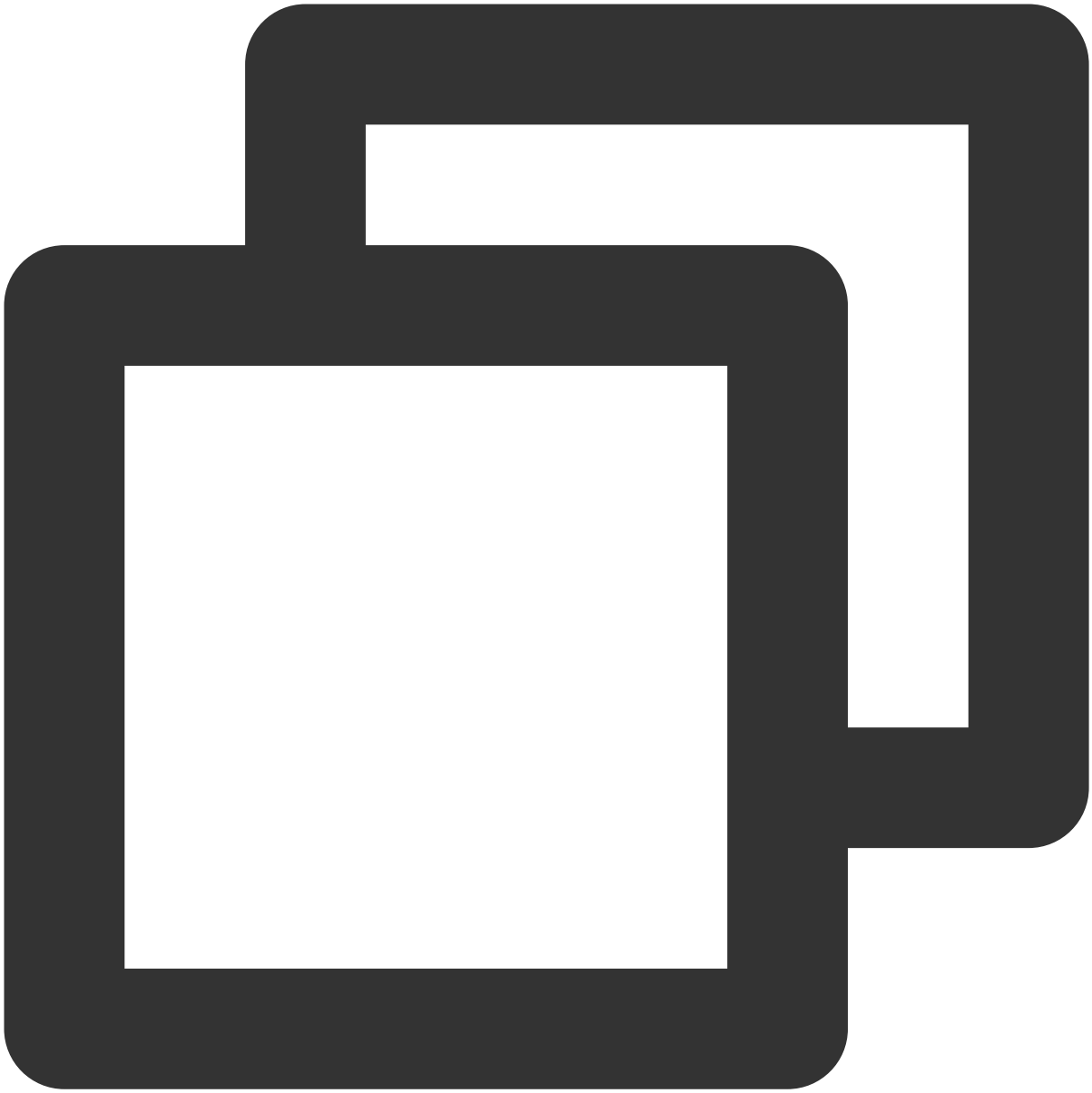
1. 在使用腾讯云提供的虚拟背景功能前，您需要前往控制台，为应用开通音视频服务，购买 **群组通话版（140万分钟）** 套餐。具体步骤请参见 开通服务。
2. 指定 LiteAVSDK_Professional SDK 版本。

虚拟背景在 tencent_calls_uikit: 2.3.2 (LiteAVSDK_Professional 11.7.0.12001) 开始支持，不同的 LiteAVSDK_Professional SDK 使用不同的模型文件。

Android

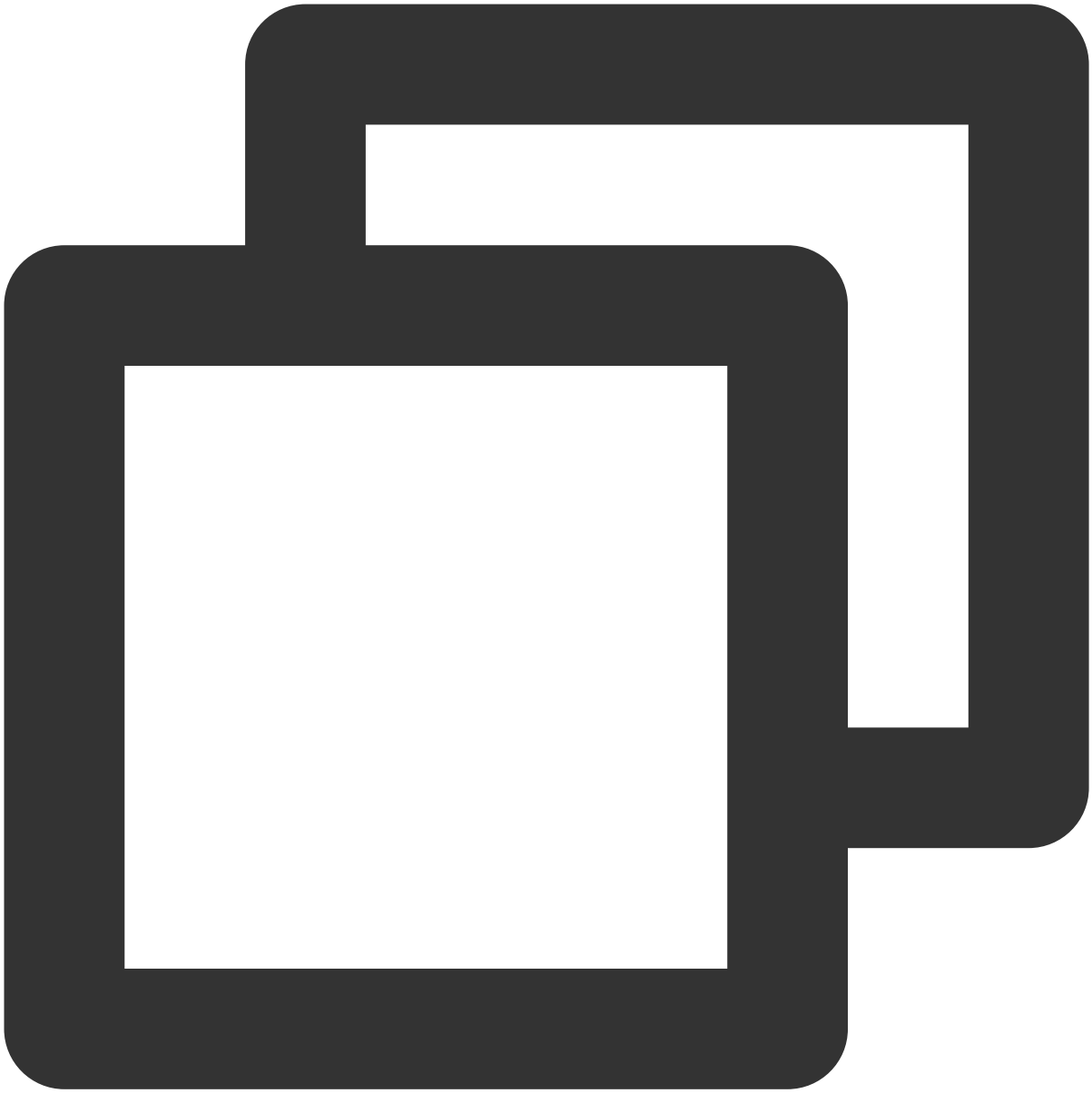
iOS

在 `build.gradle` 文件，指定 TXLiteAVSDK_Professional 版本， 例如指定为 `11.8.0.14176` ，可根据需求和版本迭代修改。



```
api "com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176"
```

修改您的 Podfile 文件中的依赖，指定 TXLiteAVSDK_Professional 版本， 例如指定为 `11.8.15669` ，可根据需求和版本迭代修改。



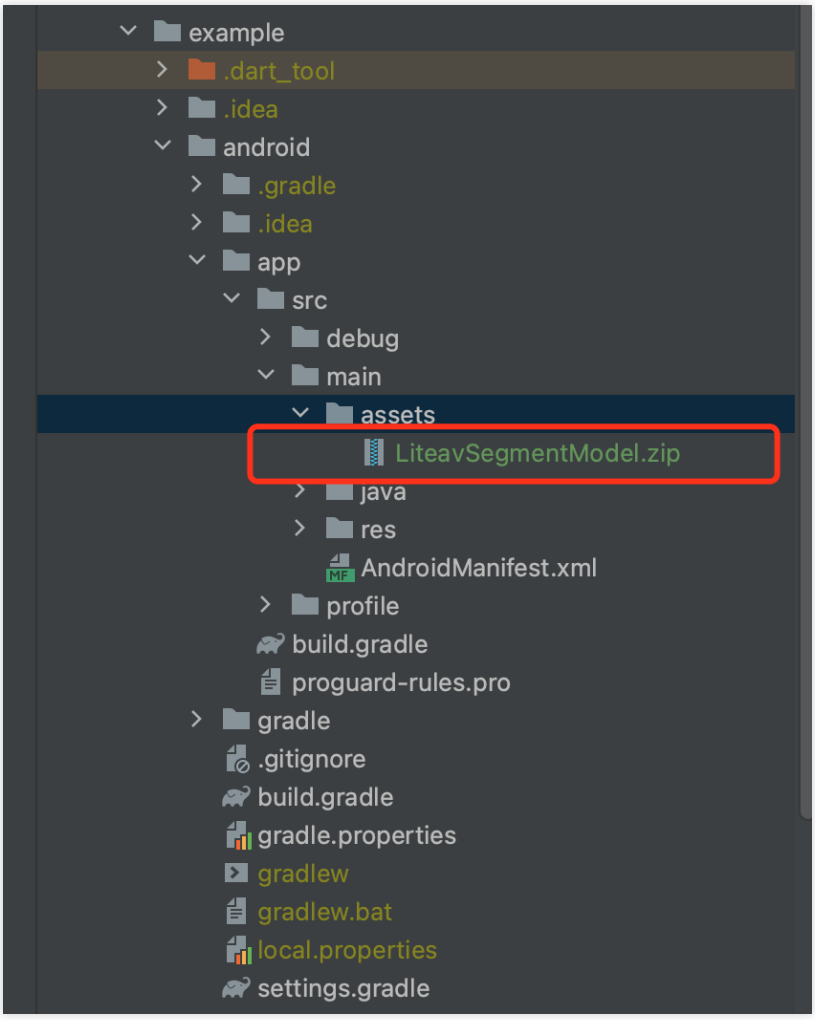
```
pod 'TXLiteAVSDK_Professional', '11.8.15669'
```

3. 根据 模型文件与 SDK 的匹配 情况下载模型文件，并添加到 Android Studio 和 Xcode 工程中。

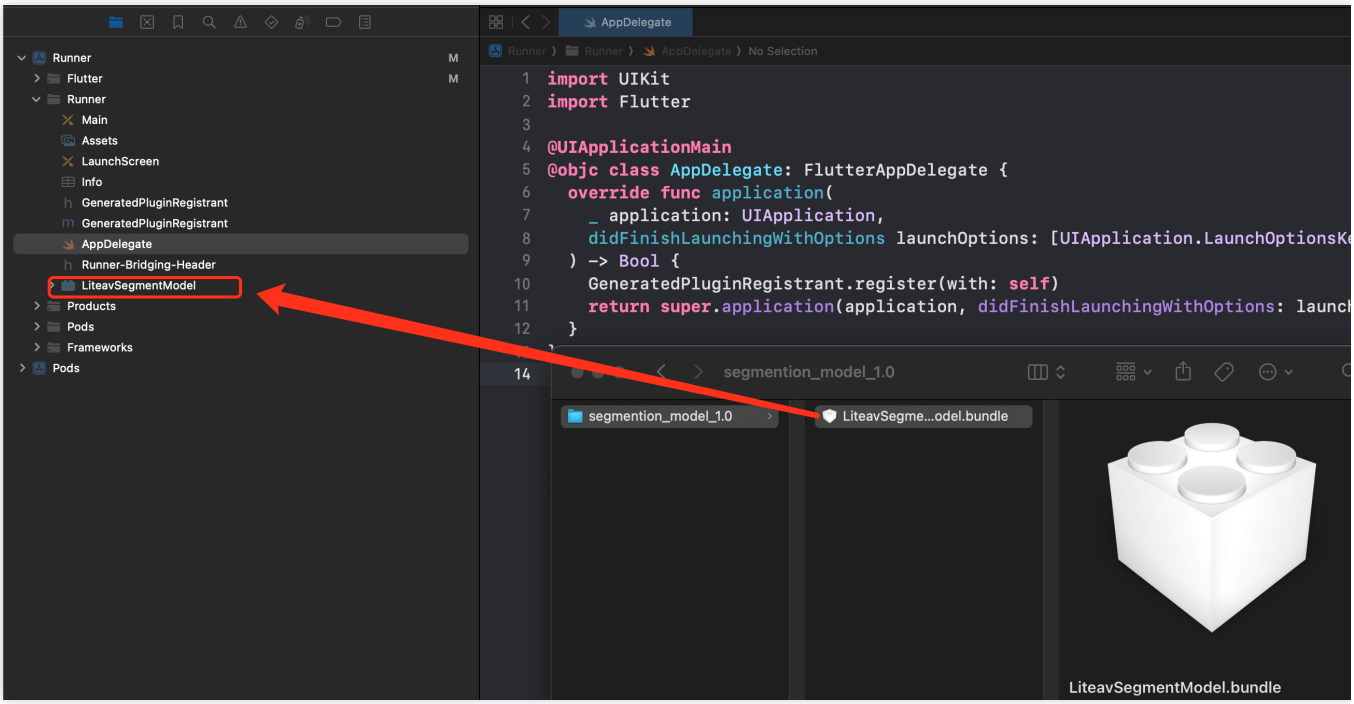
Android

iOS

解压后将 `LiteavSegmentModel.zip` 文件拷贝到 Android 工程中的 `assets` 目录下。

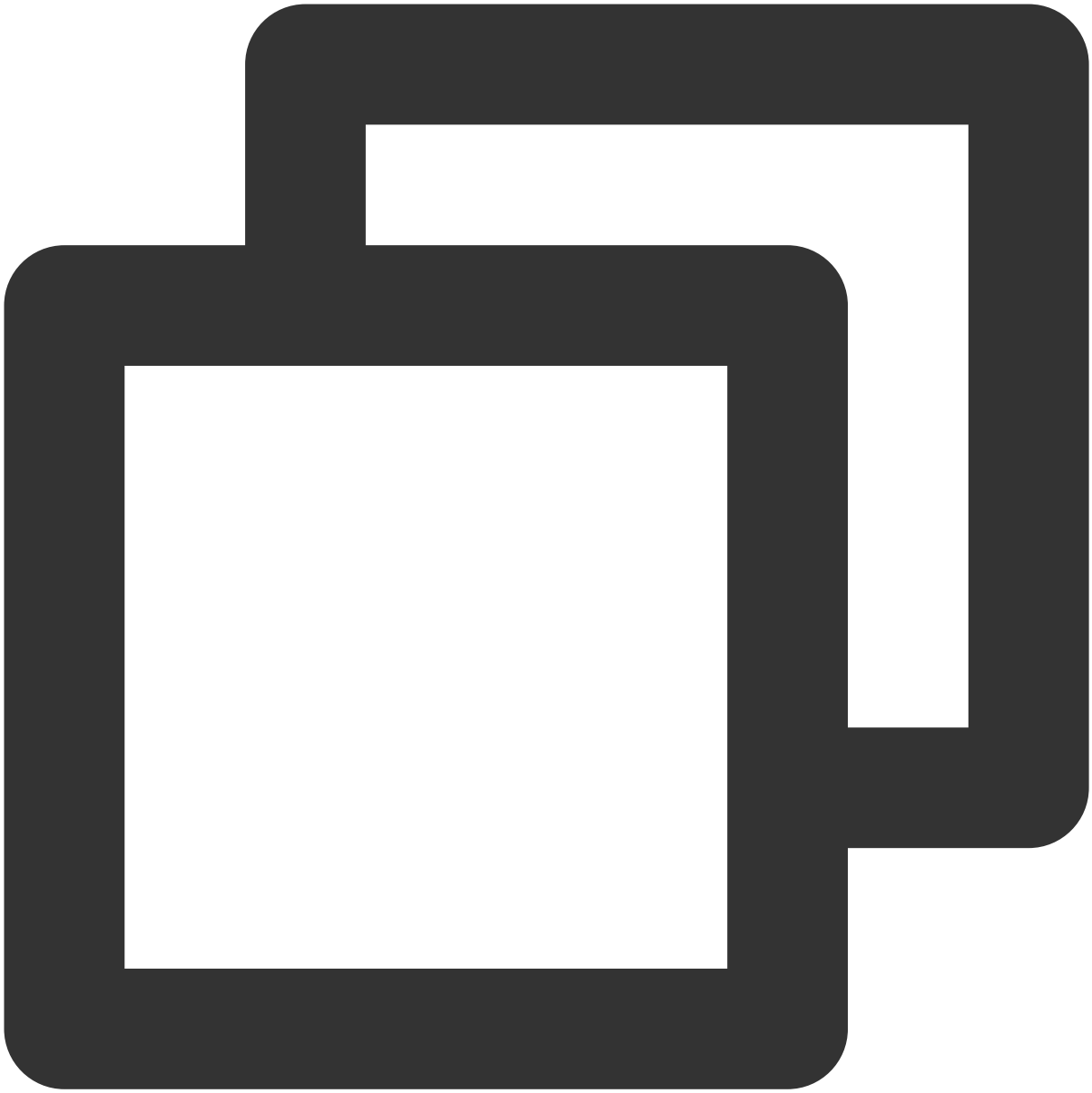


解压后将 `LiteavSegmentModel.bundle` 文件拖拽到您的 Xcode工程中。



开启模糊背景

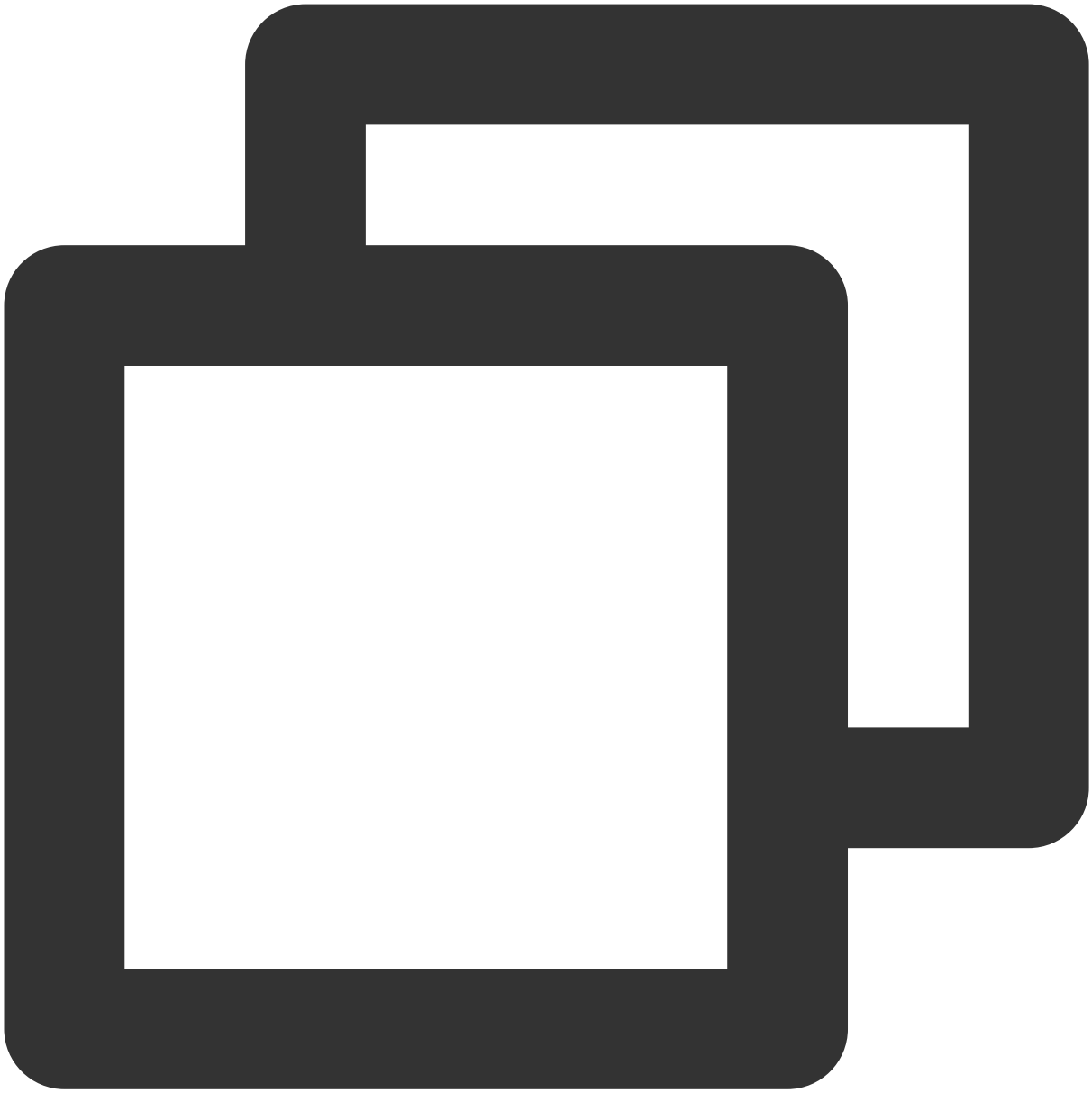
TUICallKit 的 UI 方案支持设置模糊背景。通过调用以下接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。



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

设置图片背景（可选）

图片背景需要用户自行实现，将图片文件添加到 Flutter 工程中（需要在 pubspec.yaml 文件中添加资源）， 并调用接口设置背景图片（目前只支持本地路径的图片，网络图片暂时不支持）。



```
TUICallEngine.instance.setVirtualBackground("***.png", (code, message) { });
```

常见问题

开启模糊背景无反应或有延迟？

确保已购买音视频通话 `Group Call` 套餐，详见 开通服务。

确保已下载模型文件至本地。

如果没有添加模型文件到本地路径中，在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。
检查模型文件与 SDK 是否匹配。

如何将模型文件与 SDK 进行匹配？

TUICallKit 是基于 Chat SDK 和 TRTC SDK 实现的音视频通话场景，虚拟背景是 TRTC SDK 提供的特色功能。要注意的是，虚拟背景模型文件与 TRTC SDK 的版本需要匹配，否则可能会导致模糊背景功能无法正常使用。下表列出了模型文件与 TRTC SDK 的对应关系：

SDK 版本	虚拟背景模型文件下载地址·
com.tencent.liteav:LiteAVSDK_Professional:11.7.0.12001	segmention_1.0
com.tencent.liteav:LiteAVSDK_Professional:11.8.0.14176	segmention_2.0

版权所有：腾讯云计算（北京）有限责任公司

云端录制（TUICallKit）

最近更新时间：2024-05-23 18:05:28

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。

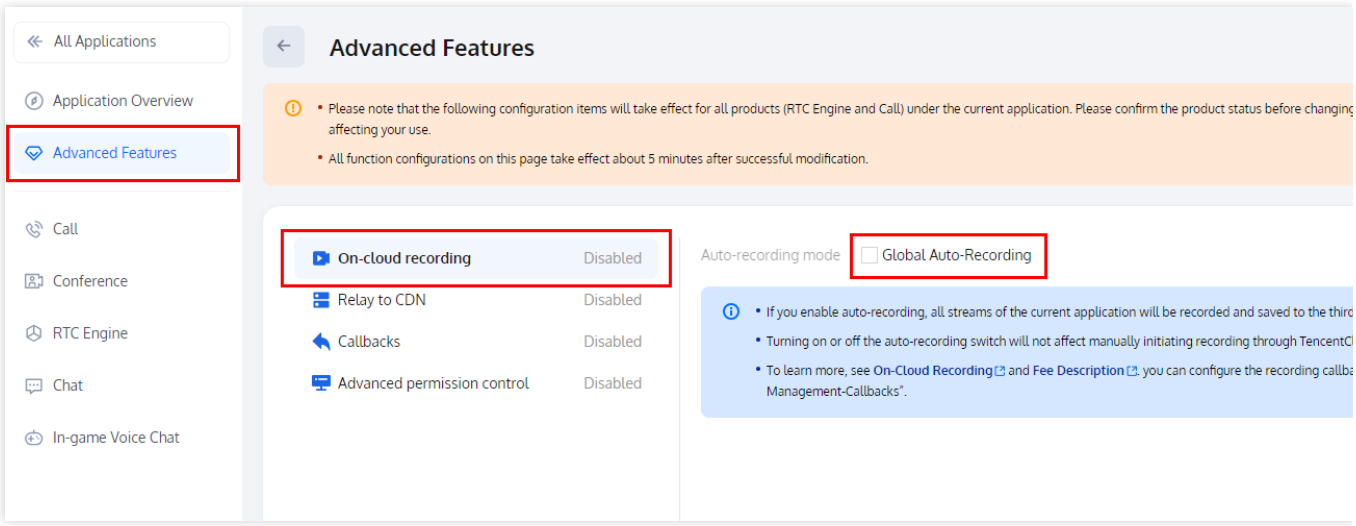
说明：

`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。

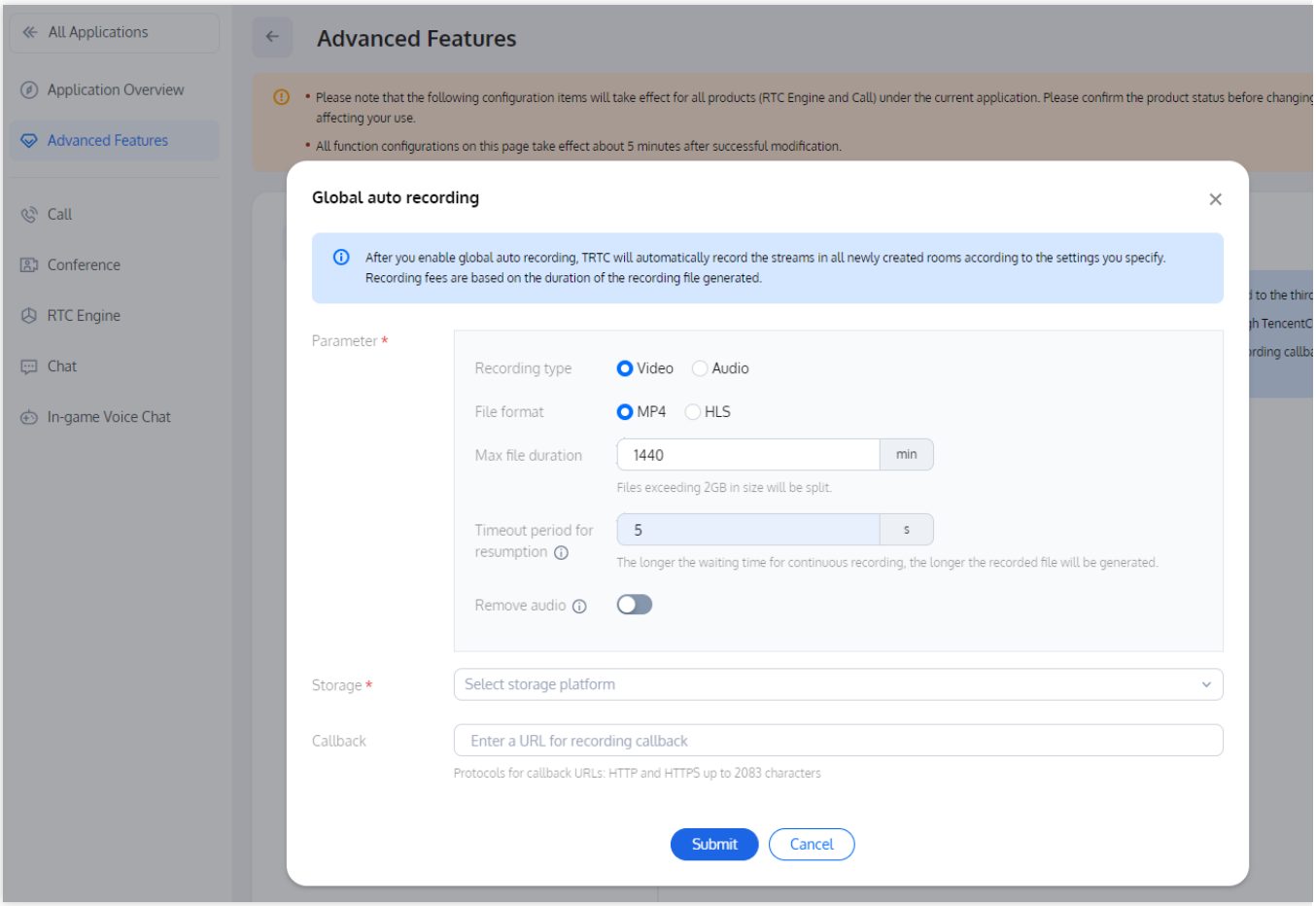
方案一：自动录制方案（推荐）

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。

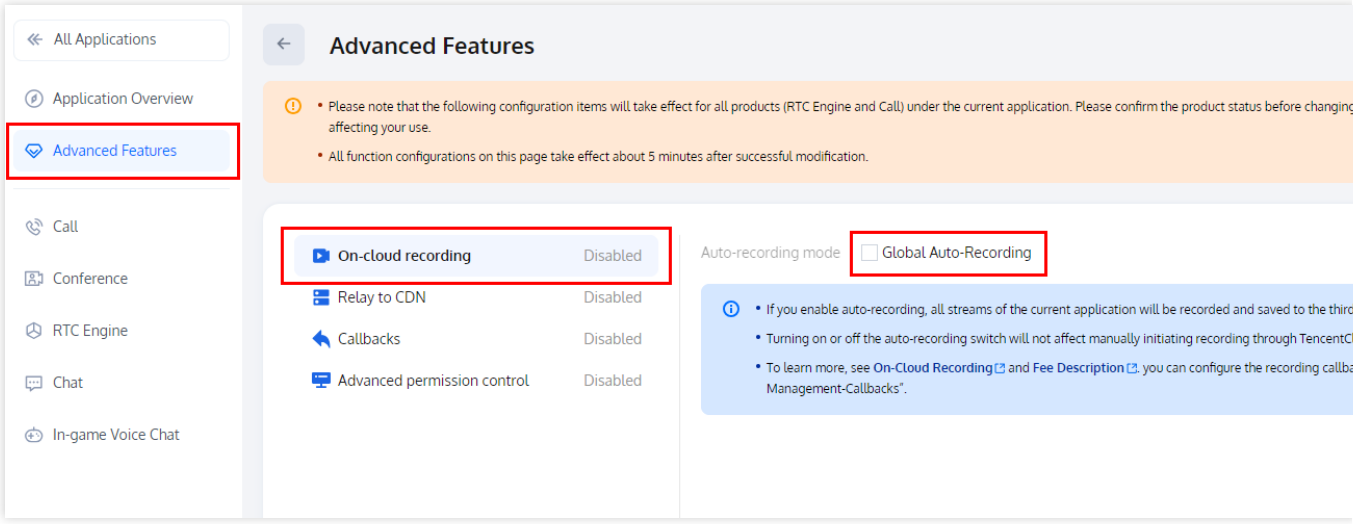
4. **模版创建完成后，勾选**全局自动录制即可完成。

方案二：REST API 录制方案

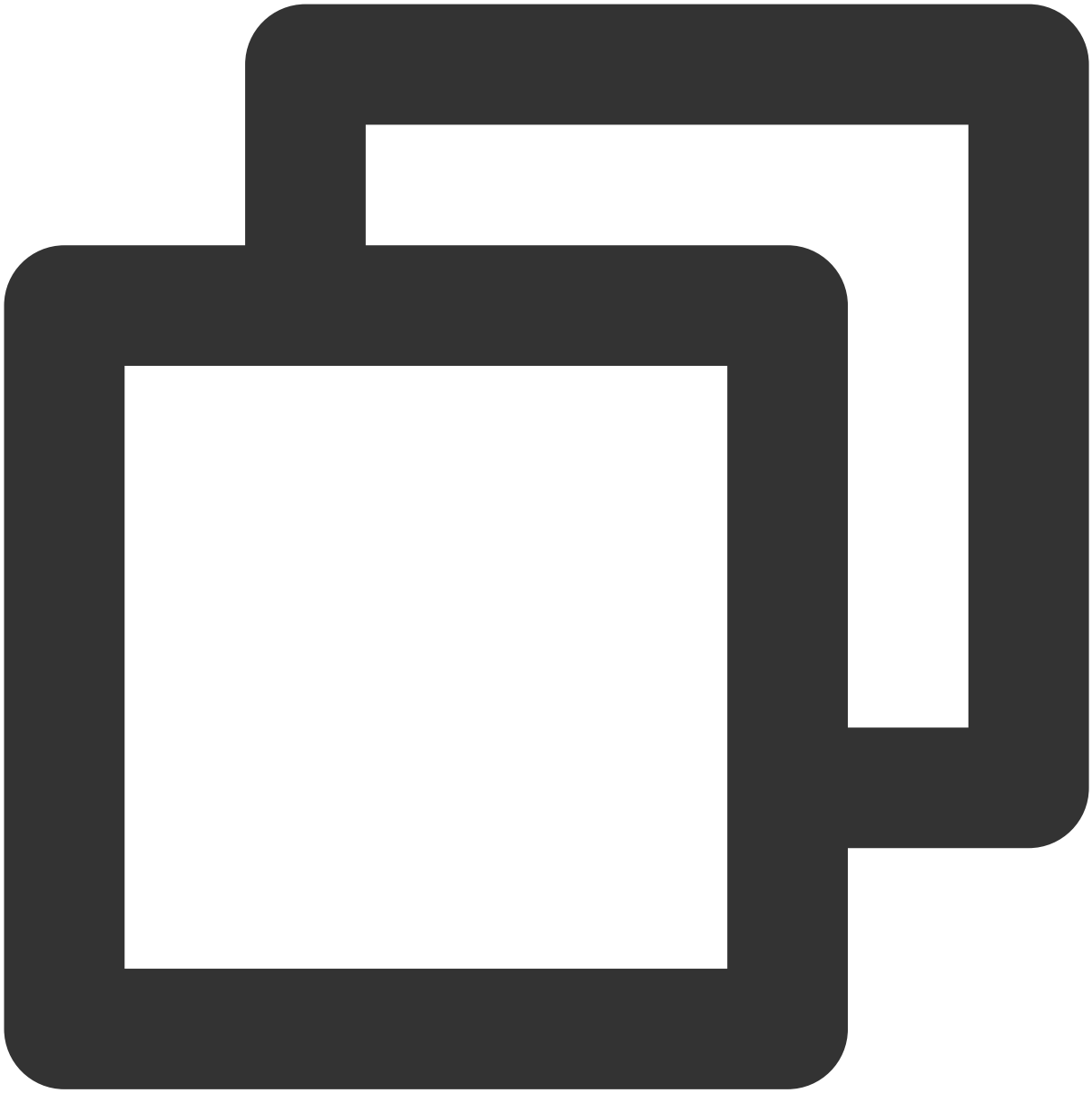
如果自动录制方案不满足您的业务需求，您也可以采用更加灵活的 REST API 录制方案，可以指定录制订阅房间内的主播，自定义合流布局，录制中途更新布局和订阅等，但是**需要搭配业务后台服务，接入复杂，功能强大，关键步骤如下**：

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：2024-05-24 18:35:12

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（Swift）

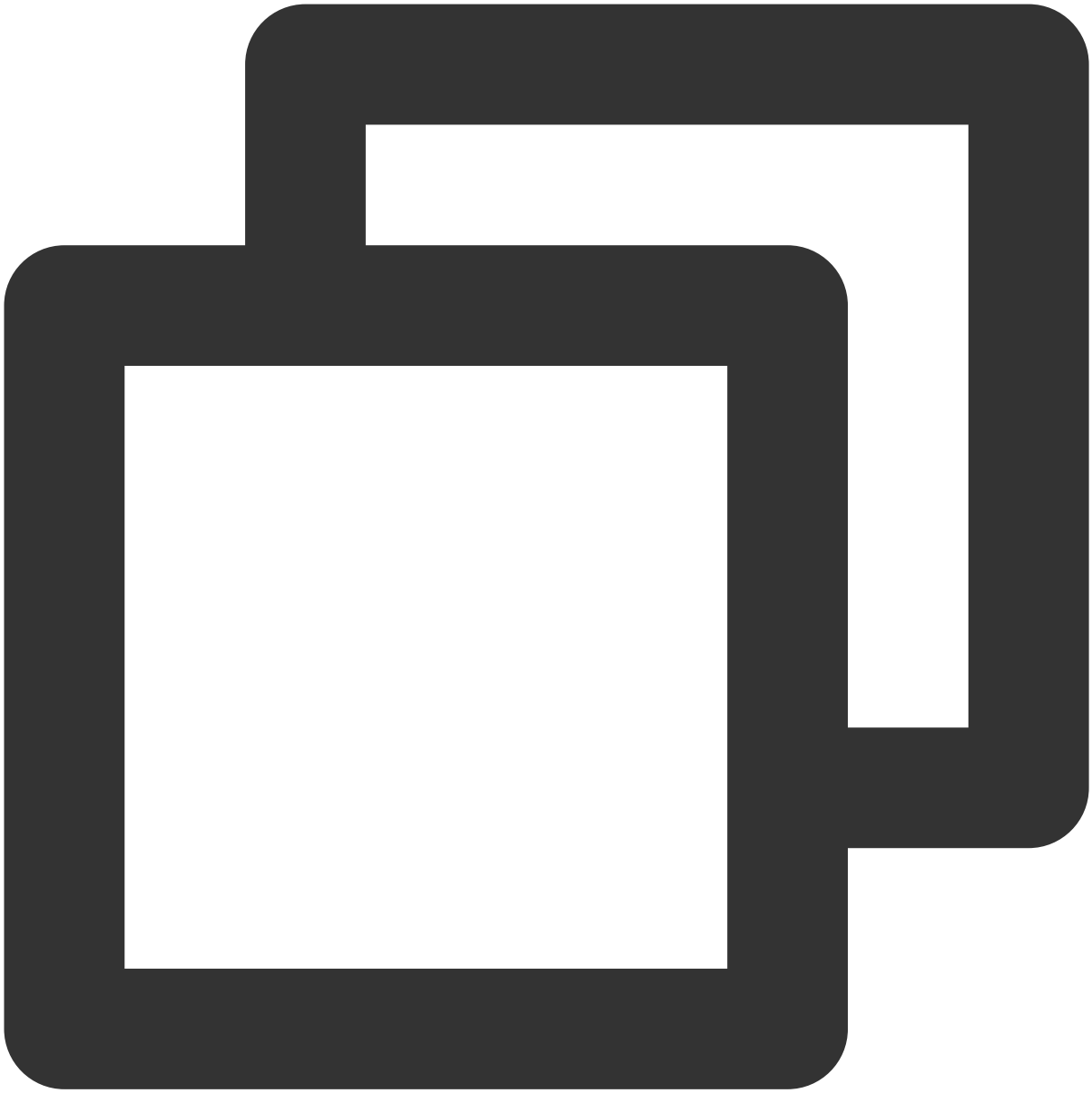
iOS（Objective-C）

Flutter（Dart）

Web&H5

uni-app(Andorid&iOS)

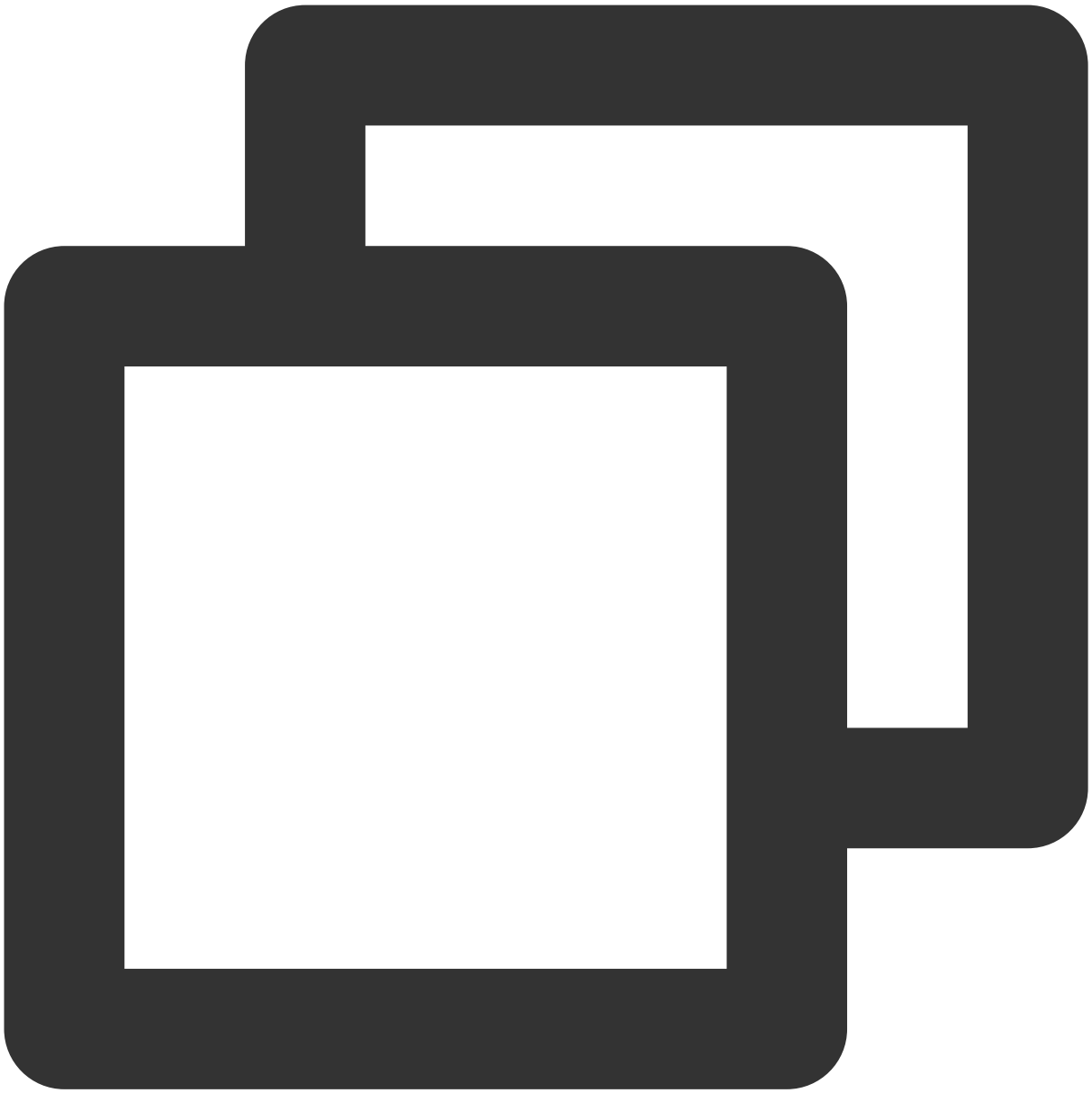
版权所有：腾讯云计算（北京）有限责任公司



```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png
    object : TUICommonDefine.Callback {
        override fun onSuccess() {
        }

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

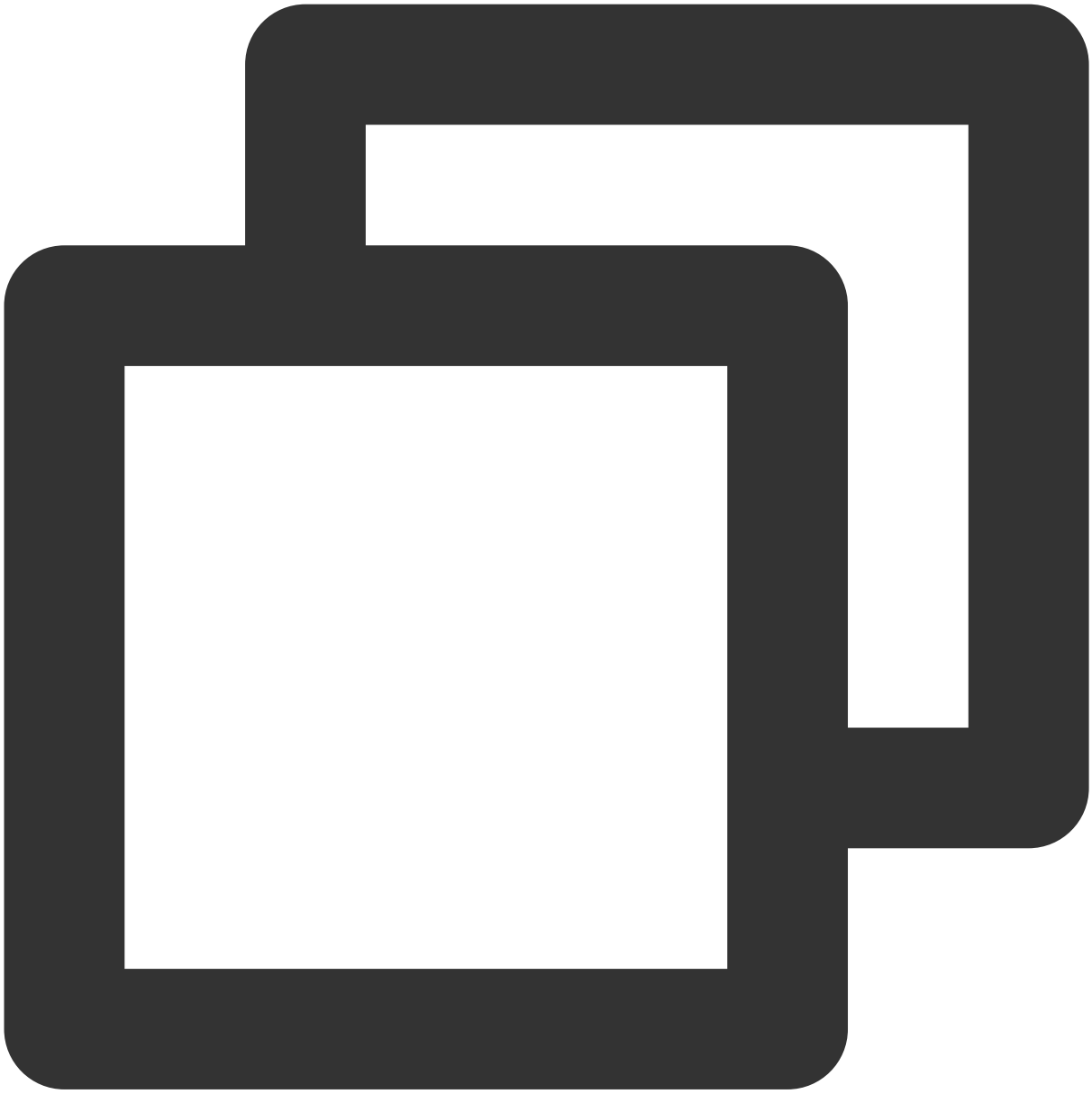


```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png
    @Override
    public void onSuccess() {
    }

    @Override
    public void onError(int errorCode, String errorMessage) {
```

```
    }
});
```

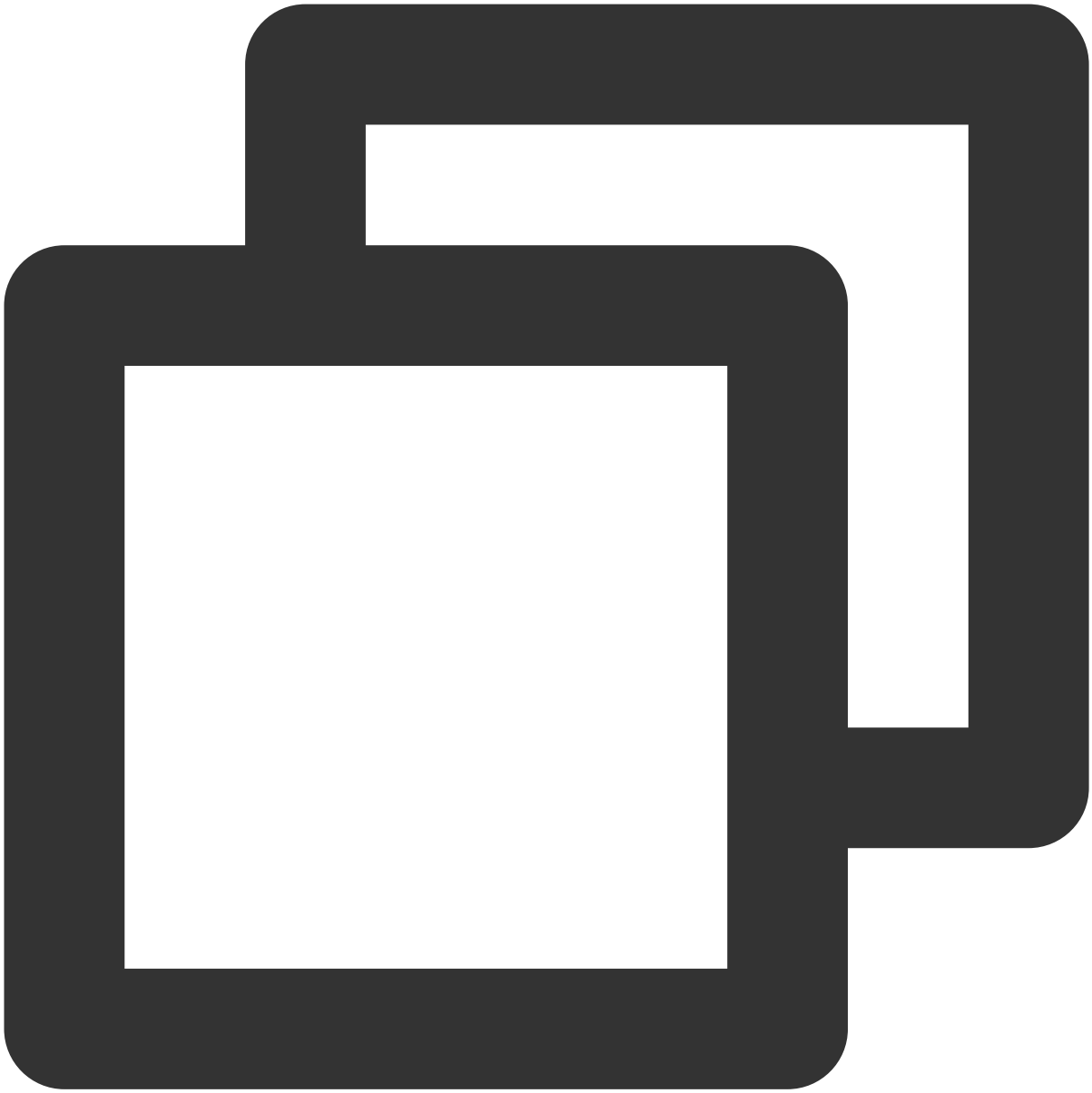


```
import TUICallKit_Swift
import TUICallEngine

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

} fail: { code, message in

}
```

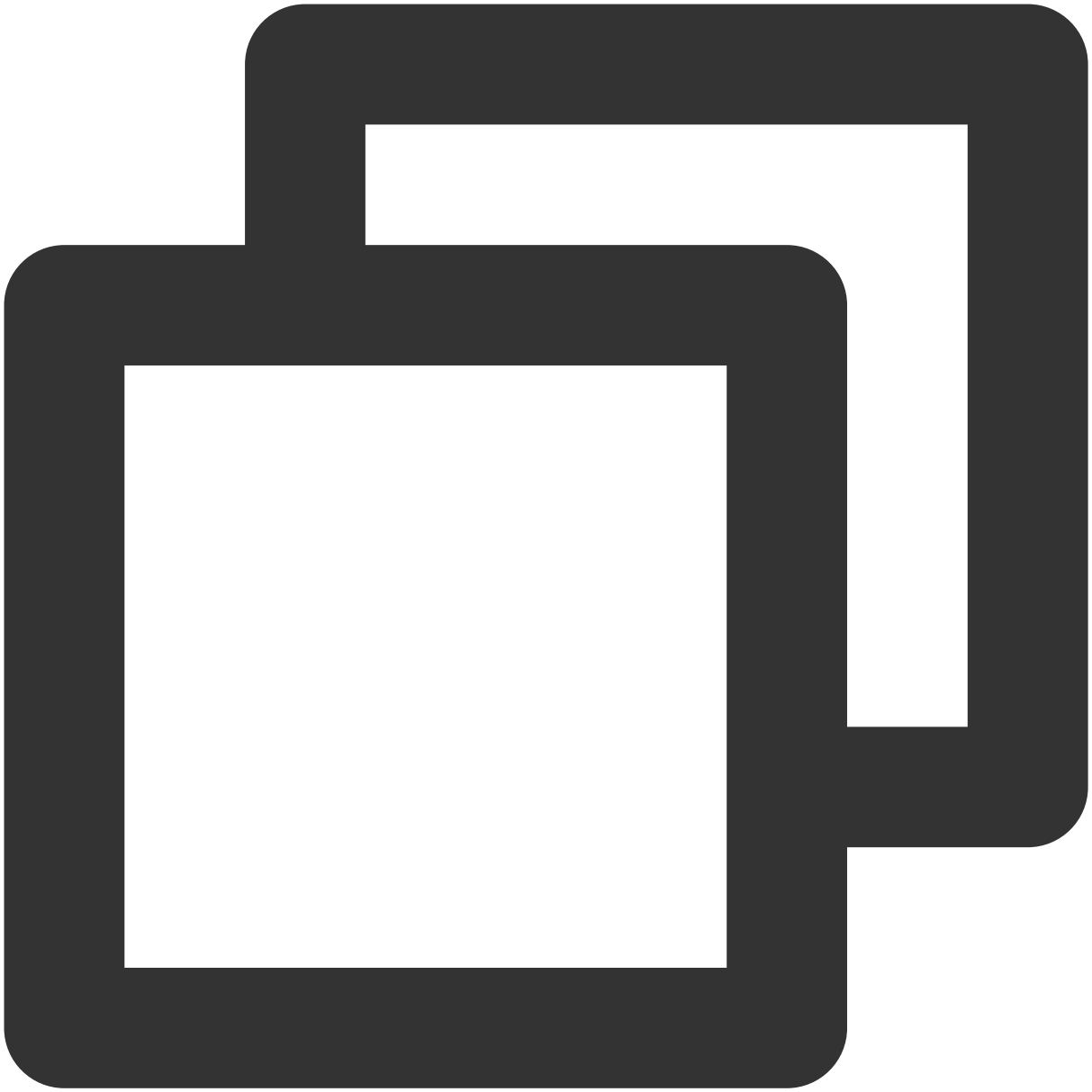


```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] setSelfInfoWithNickname:@"" avatar:@"" succ:^{

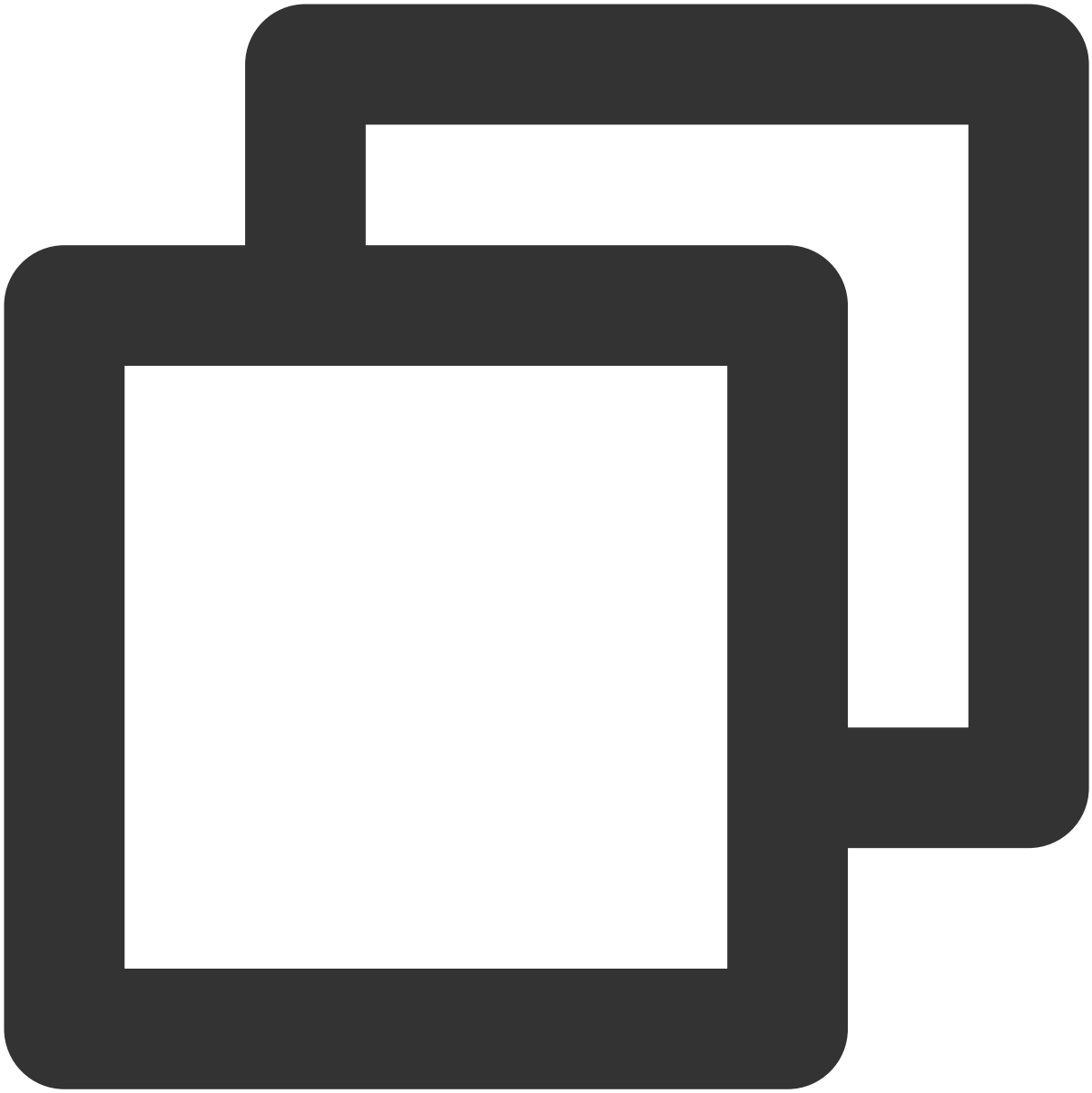
} fail:^(int code, NSString * _Nullable errMsg) {

}];
```

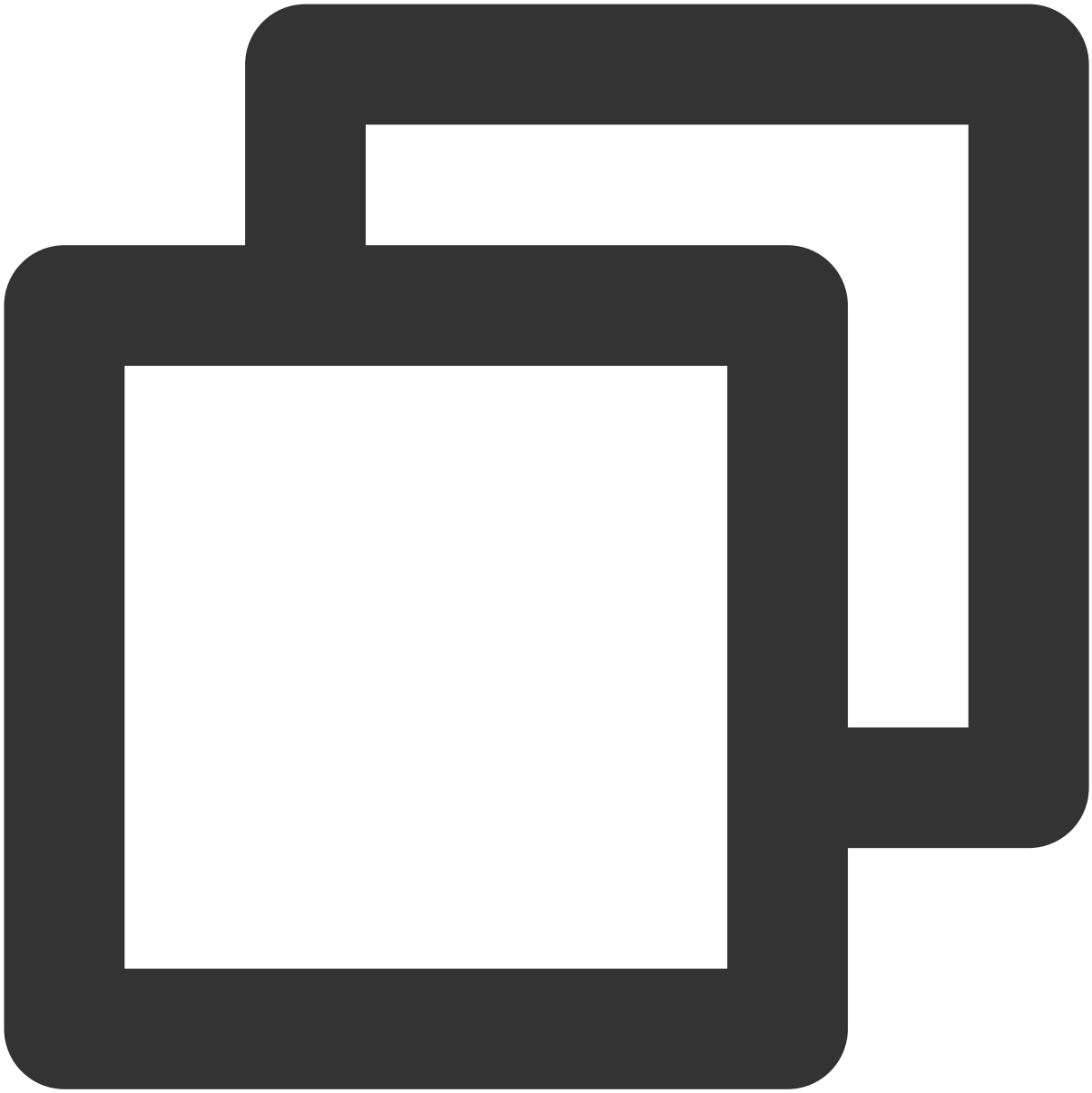


```
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';

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



```
import { TUICallKitServer } from '@tencentcloud/call-uikit-vue';
// import { TUICallKitServer } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.setSelfInfo({ nickName: "jack", avatar: "http://xxx" });
} catch (error) {
  console.error(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`
}
```



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

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。

版权所有：腾讯云计算（北京）有限责任公司

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

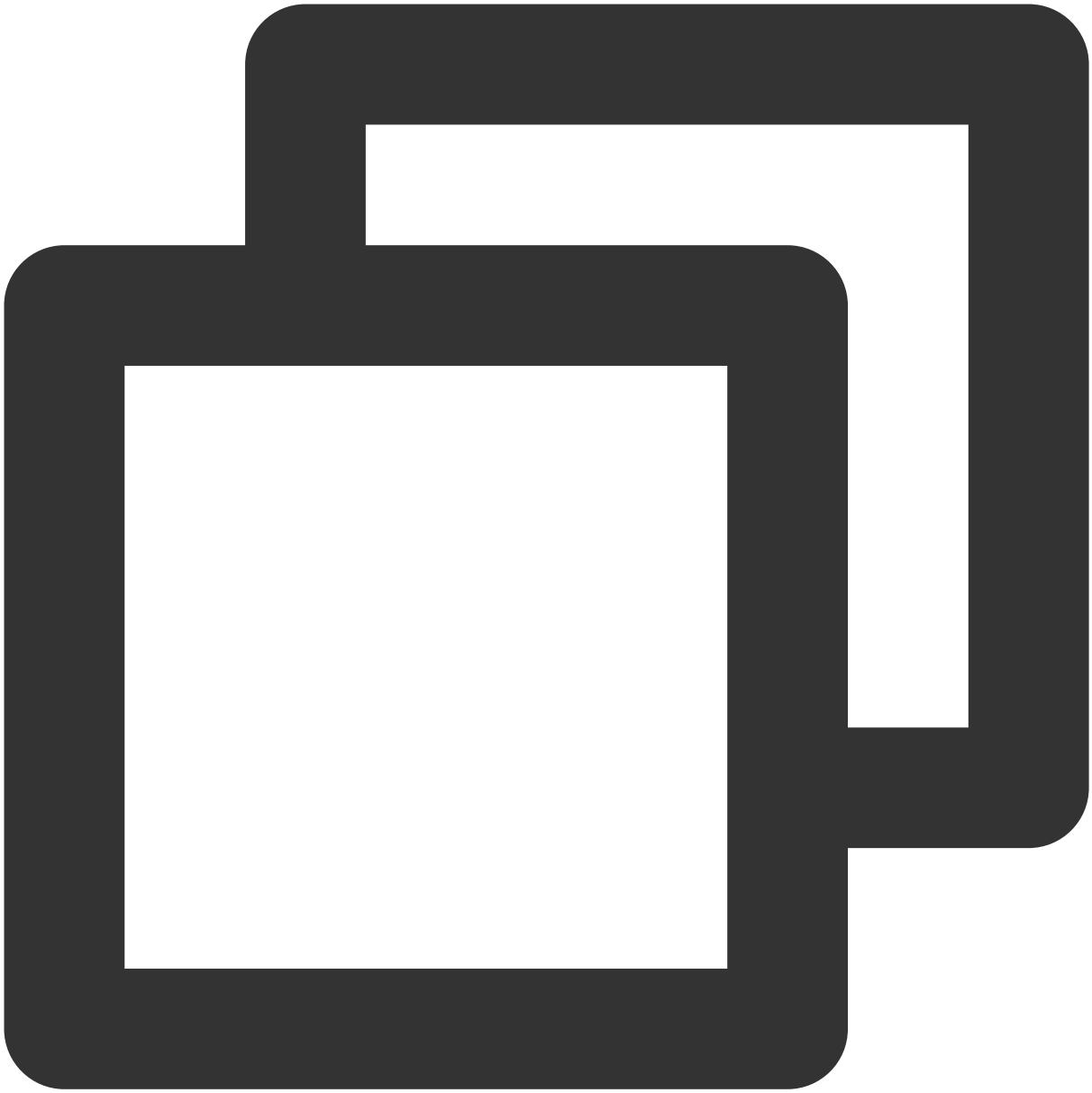
TUICallKit 组件提供了若干个功能开关，可以根据需要选择开启或关闭，具体参见 TUICallKit 属性介绍。

`videoDisplayMode` ：画面显示模式。

`videoResolution` ：通话分辨率。

React

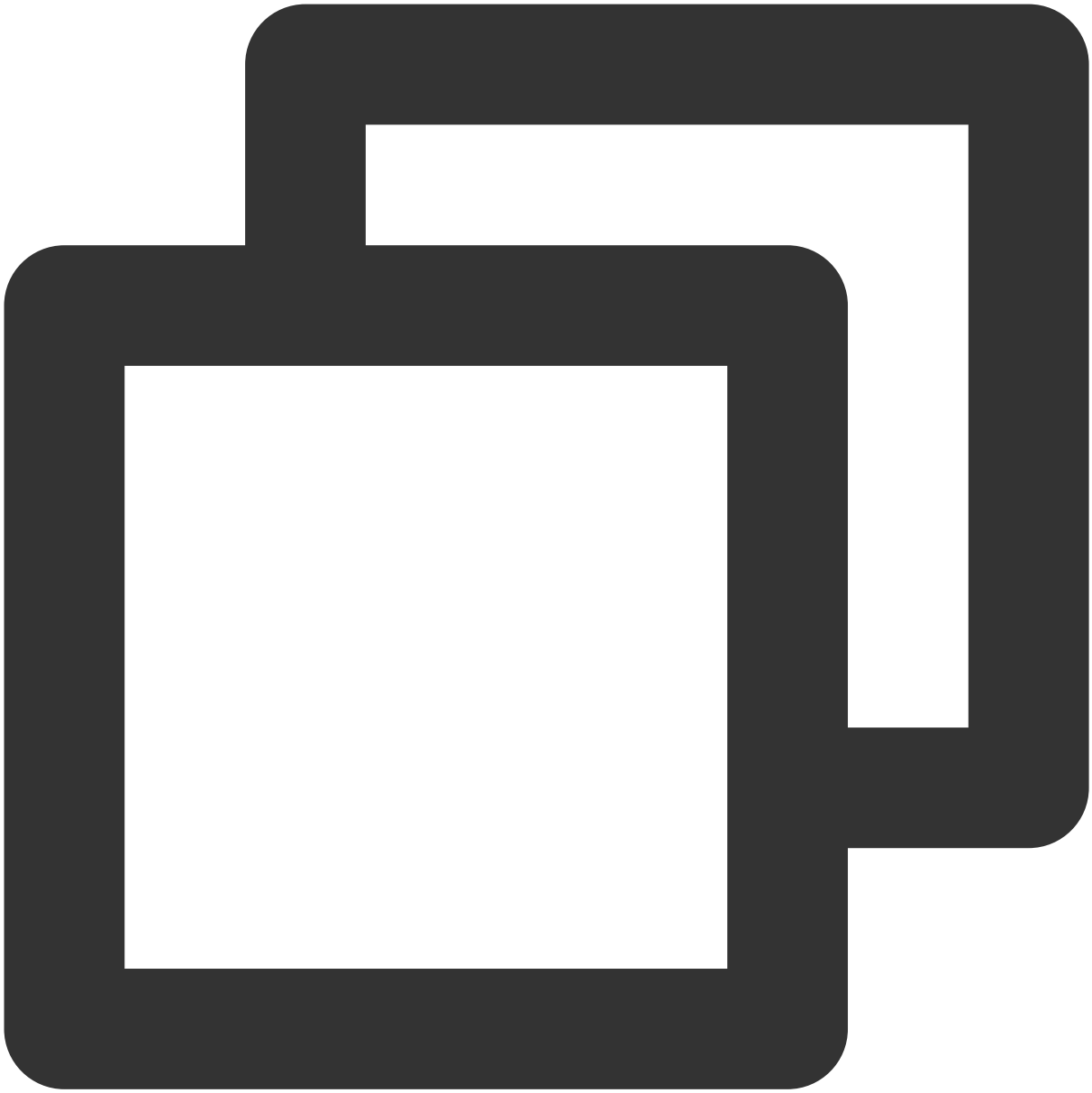
Vue



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

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

版权所有：腾讯云计算（北京）有限责任公司



```
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

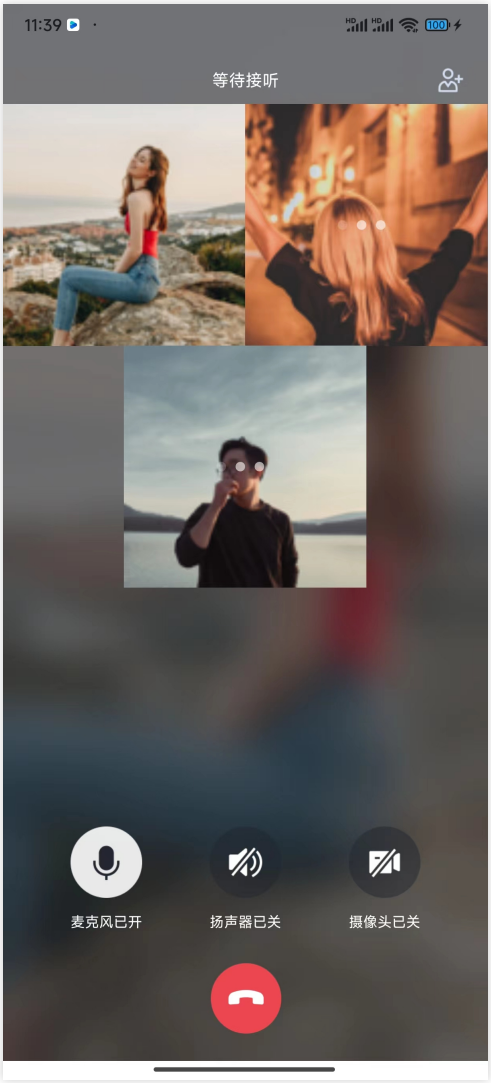
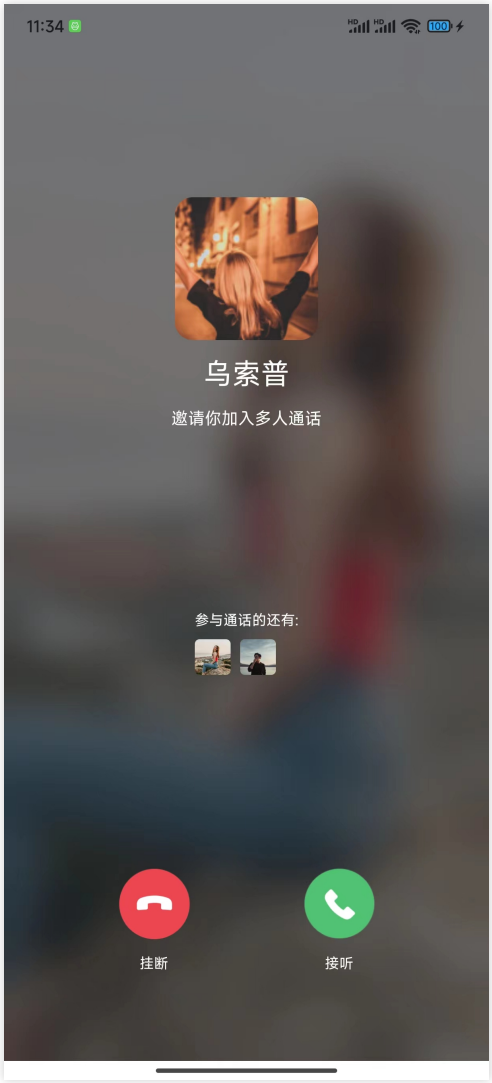
Android&iOS&Flutter

最近更新时间：2024-06-17 17:41:52

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	
		

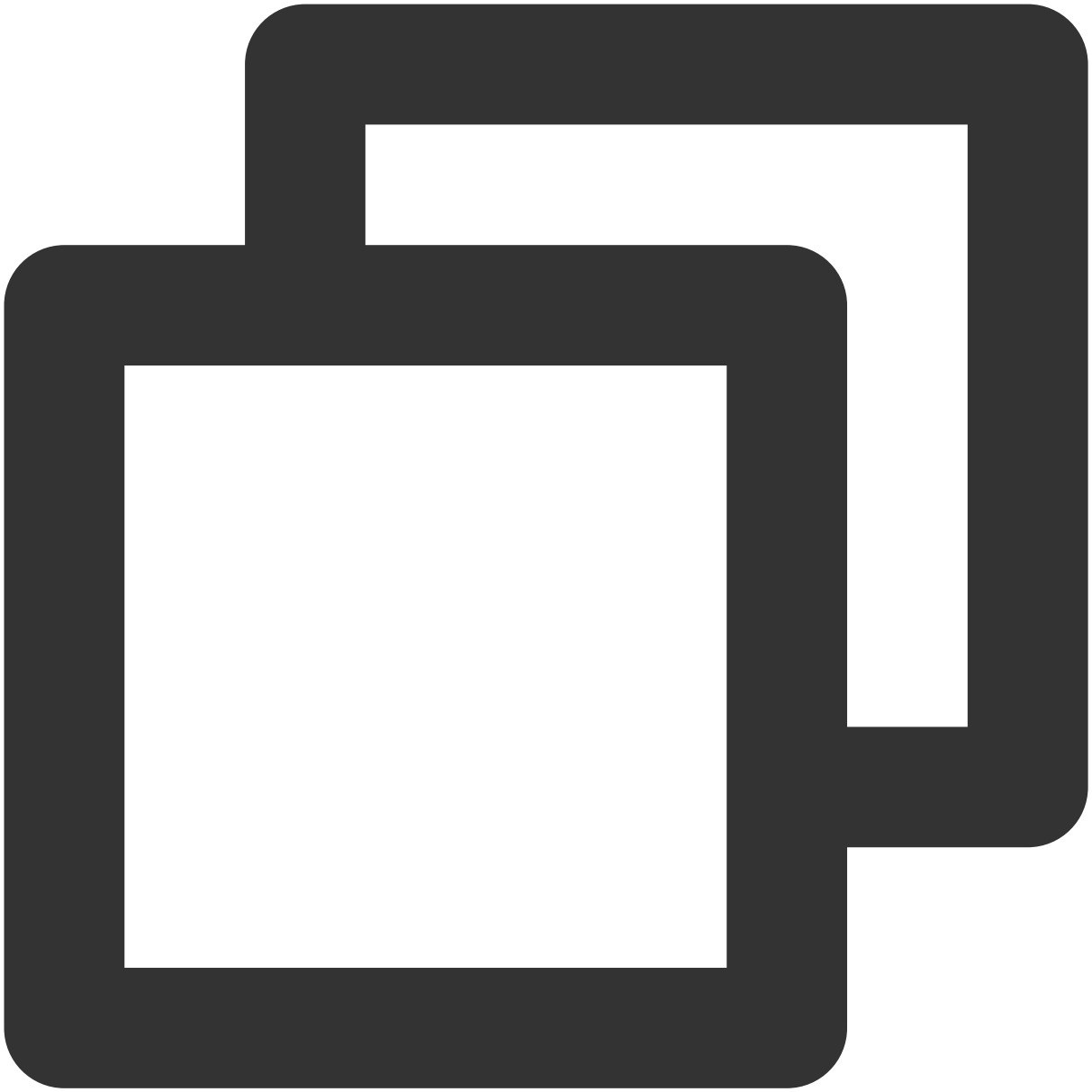
创建 groupID

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 Chat API 创建群，详见 Chat 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

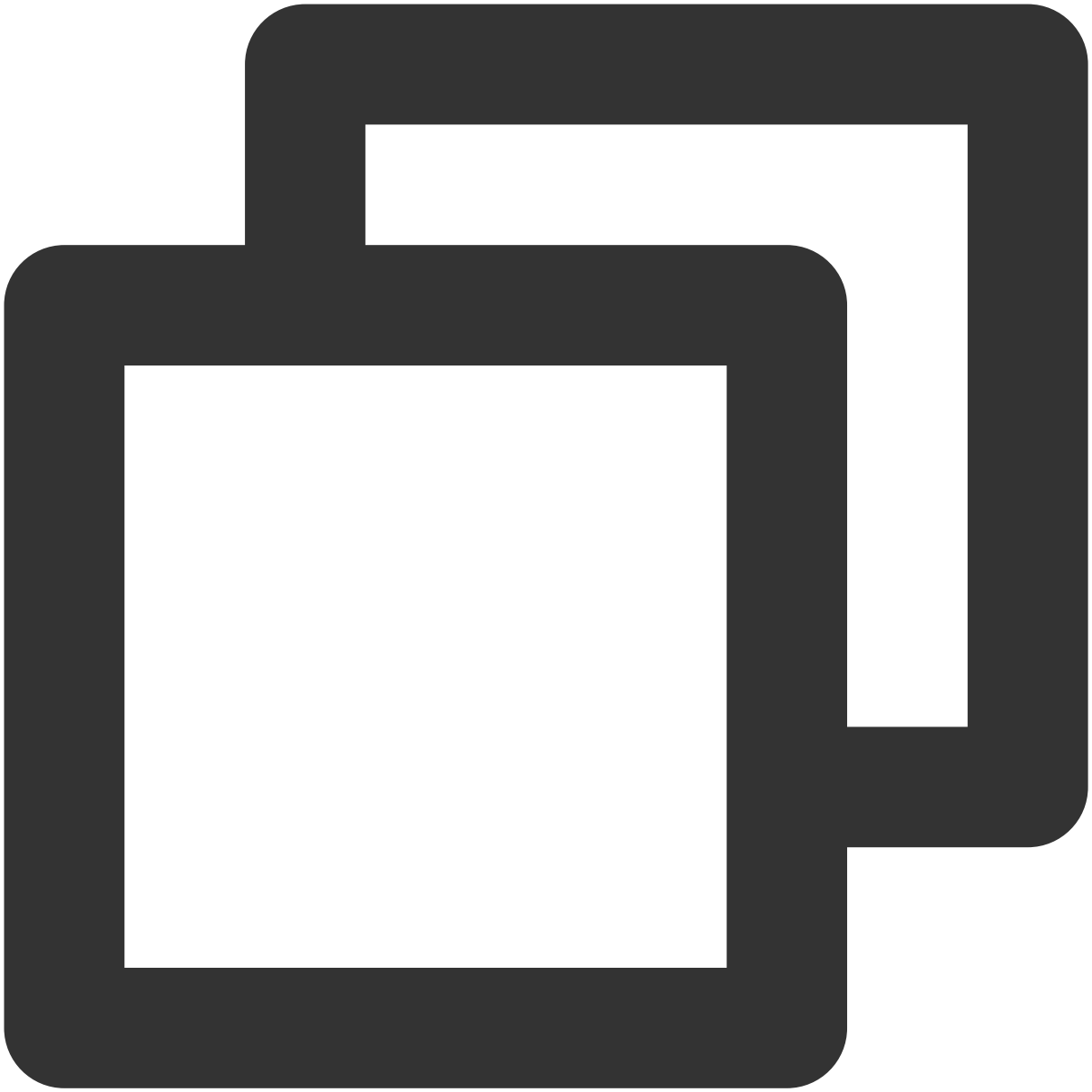
发起群组通话

调用 groupCall API 发起群通话。
Android（Kotlin）
Android（Java）
iOS（Swift）
iOS（Objective-C）
Flutter（Dart）



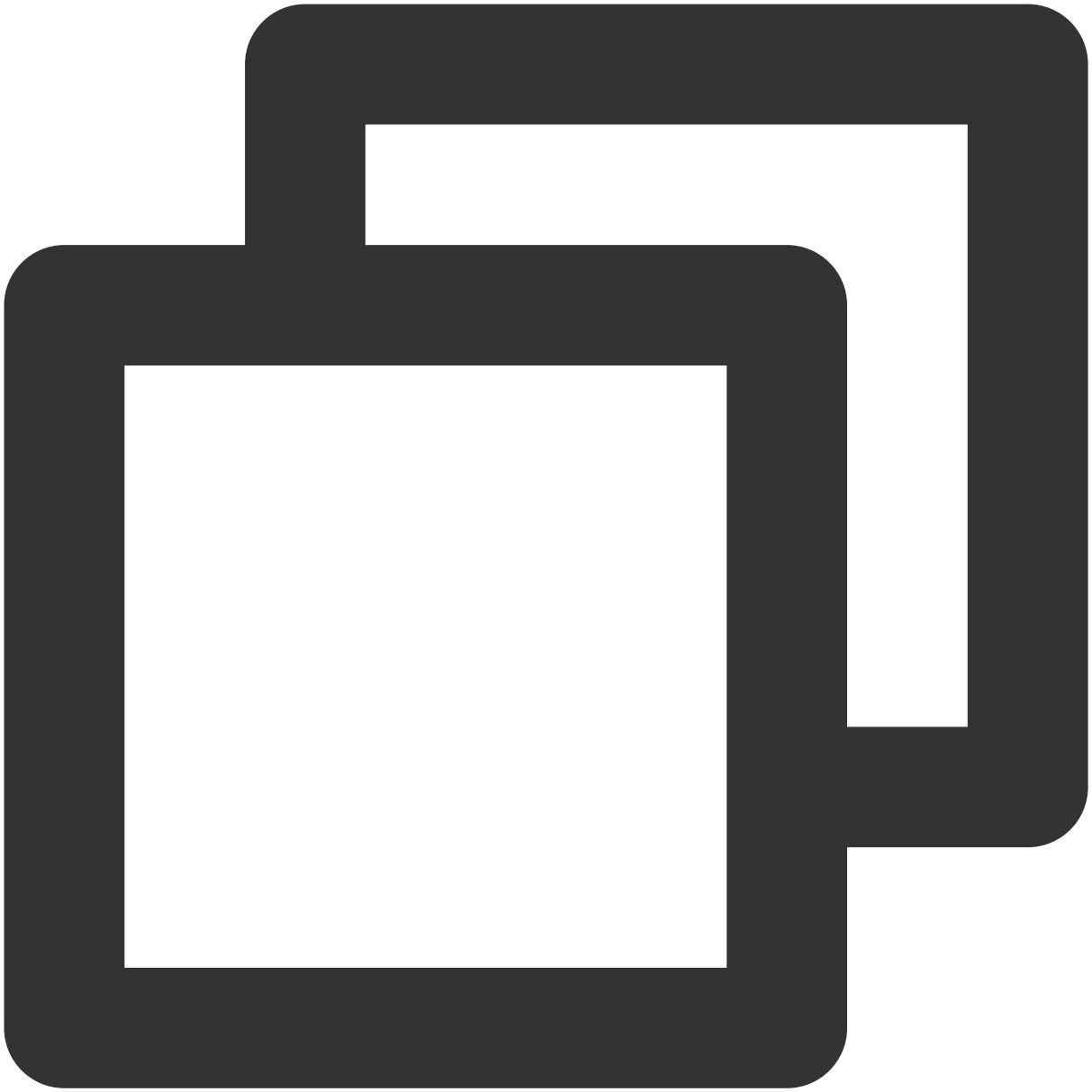
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

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



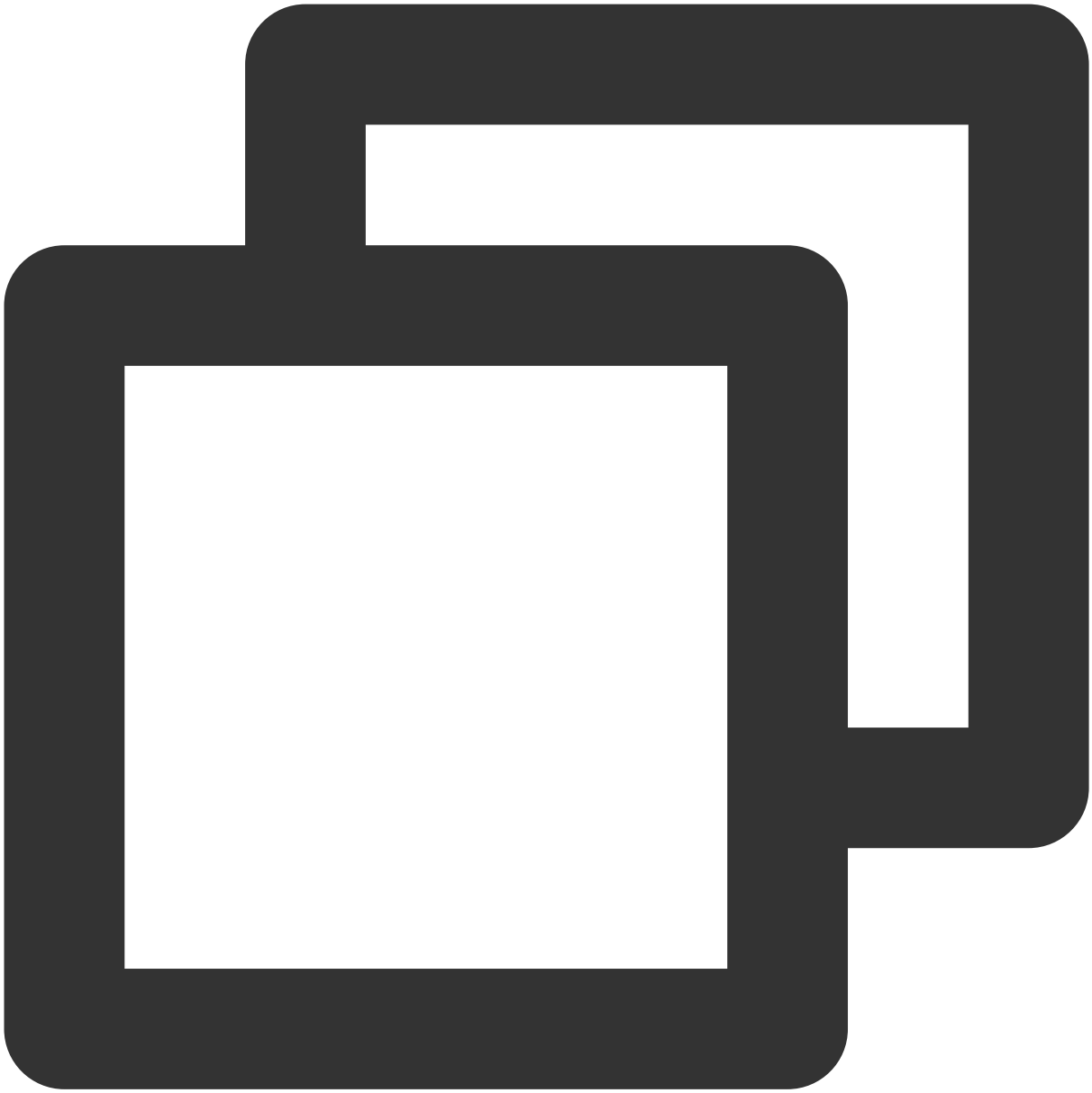
```
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

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



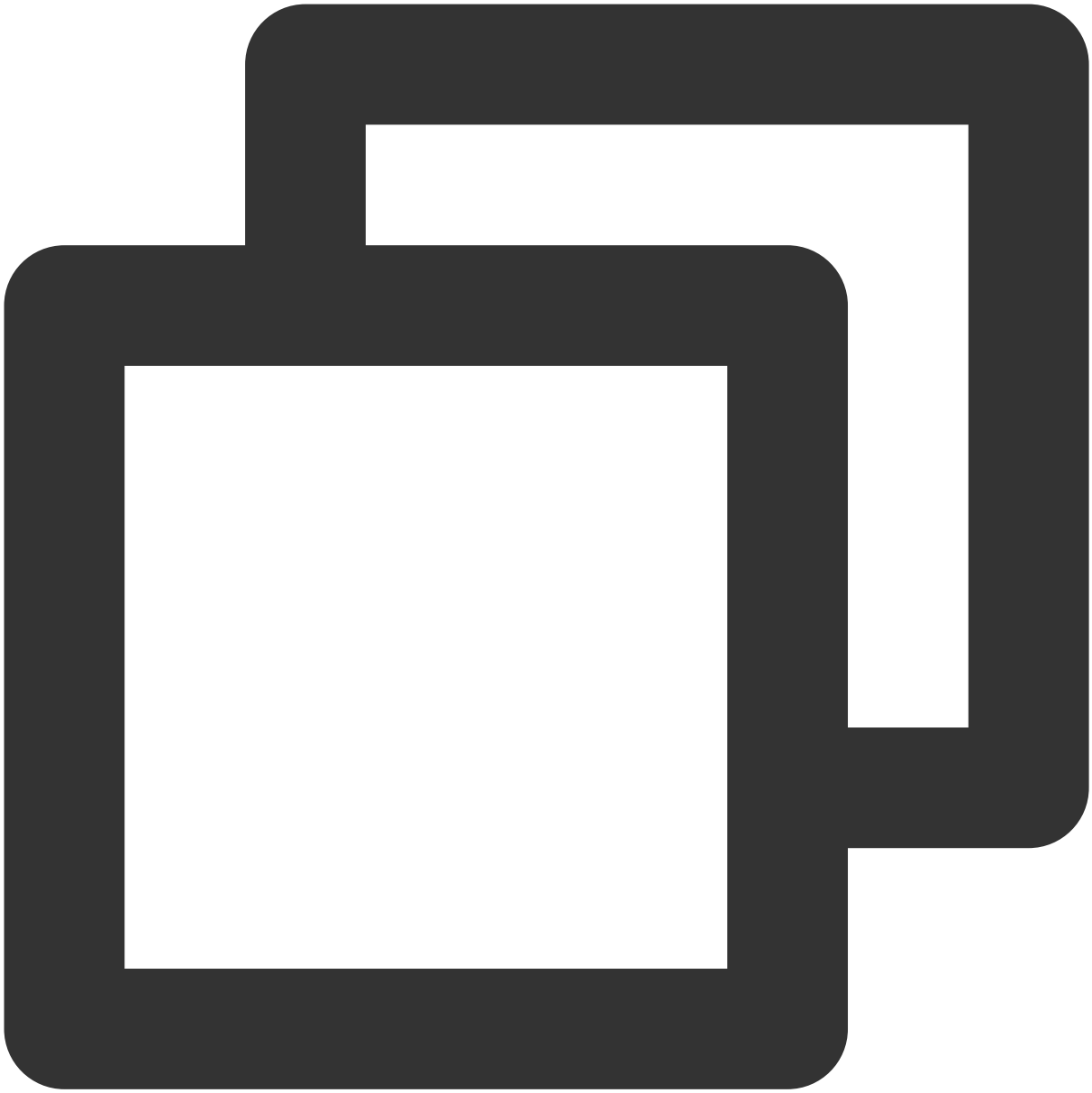
```
import TUICallKit_Swift
import TUICallEngine

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

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



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

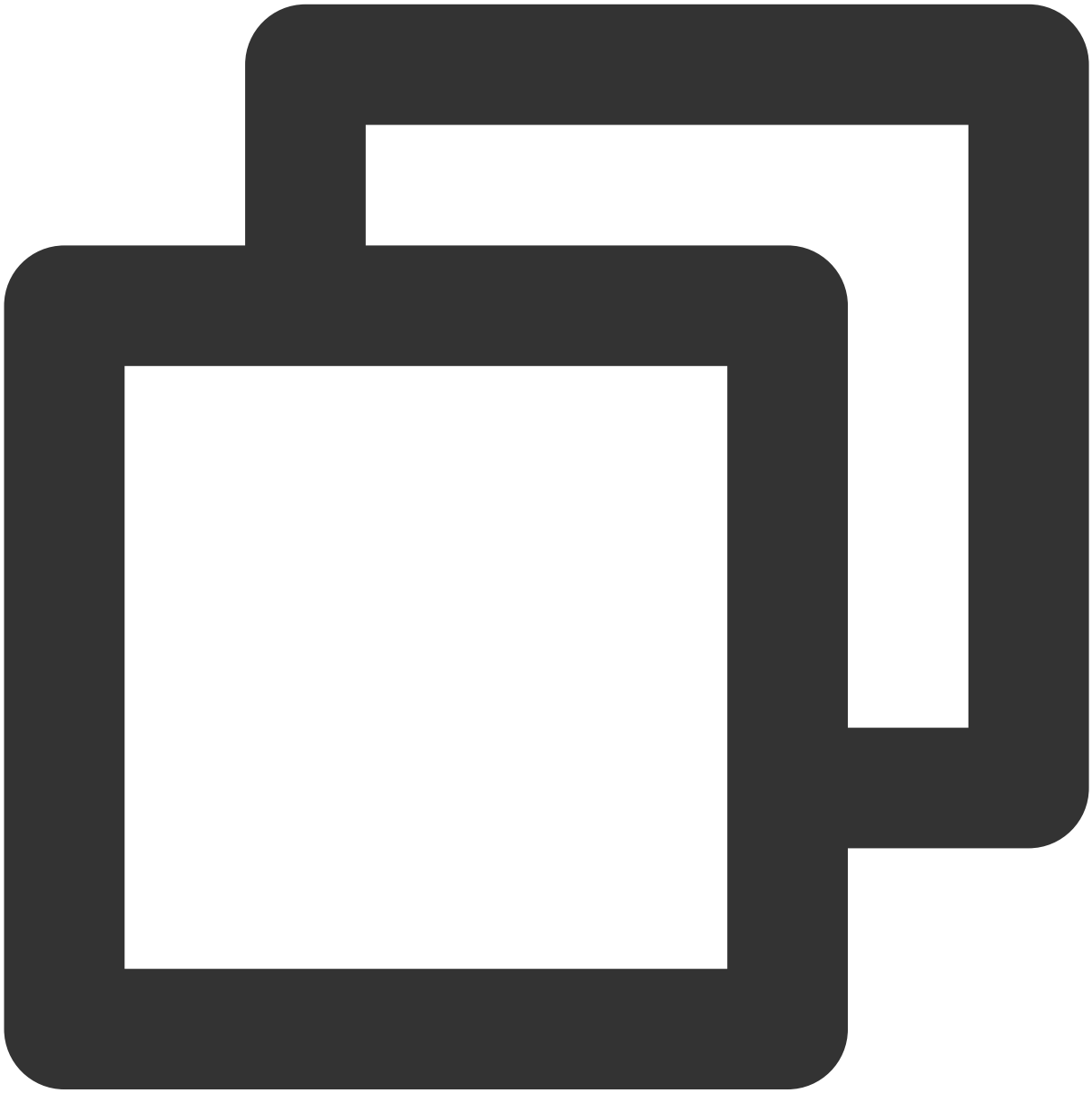
void groupCall() {
    TUICallKit.instance.groupCall('0001', ['denny', 'mike', 'tommy'], TUICallMediaT
}
```

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）

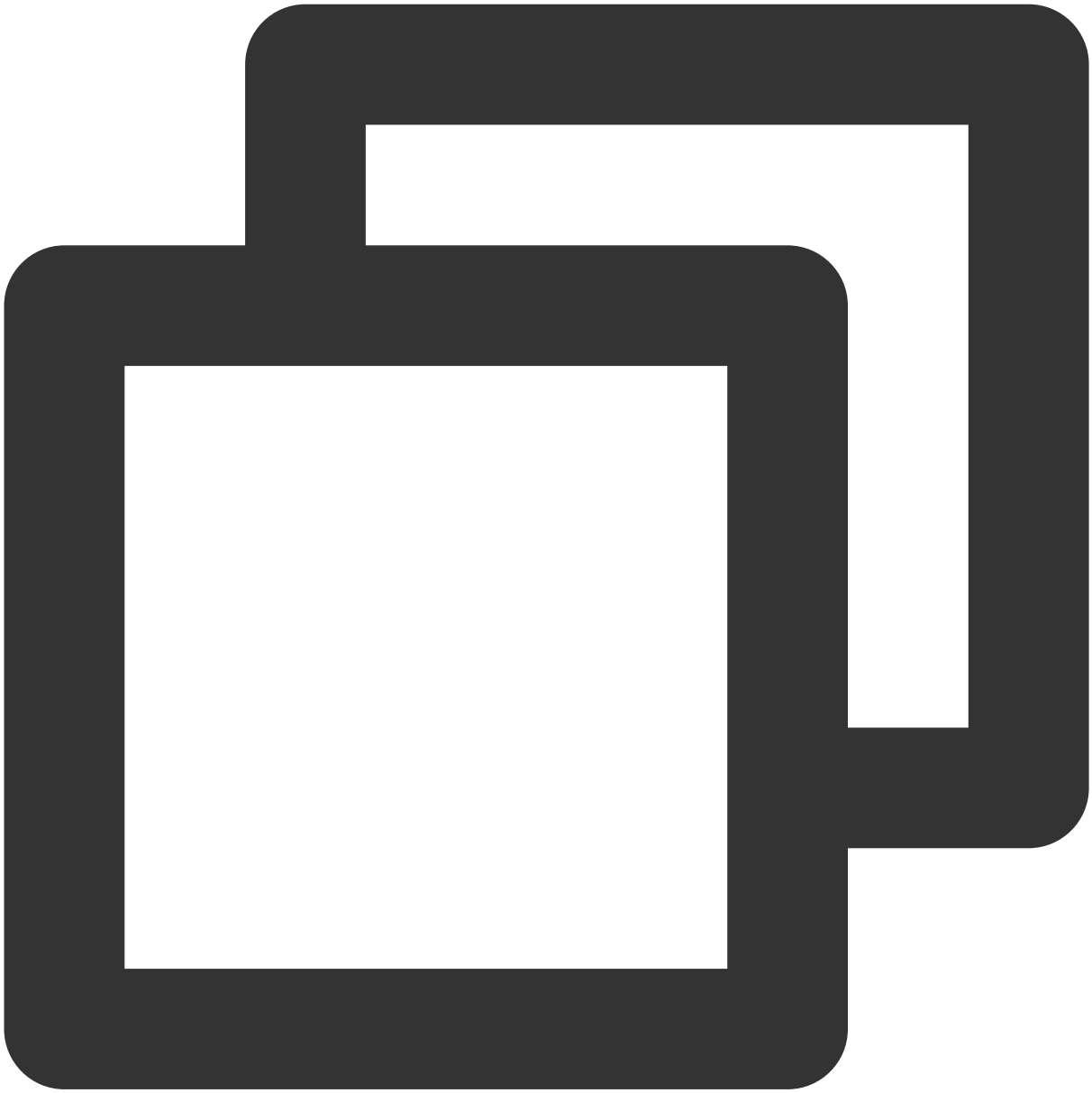
Android（Java）
iOS（Swift）
iOS（Objective-C）
Flutter（Dart）



```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

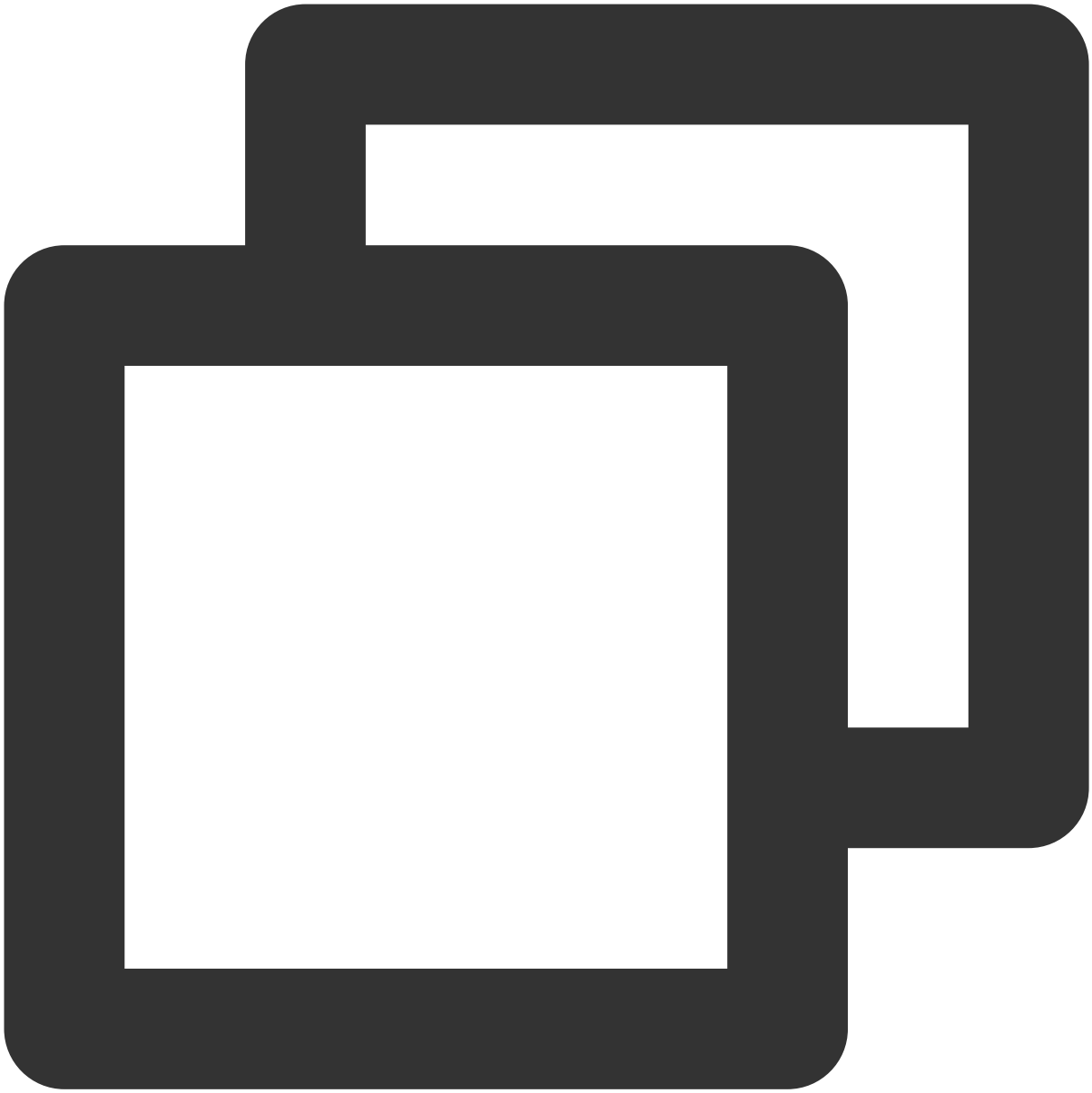
var roomId = TUICommonDefine.RoomId()
roomId.intRoomId = 123321
```

```
var groupId = "12345678"
TUICallKit.createInstance(context).joinInGroupCall(roomId, groupId, TUICallDefine.M
```



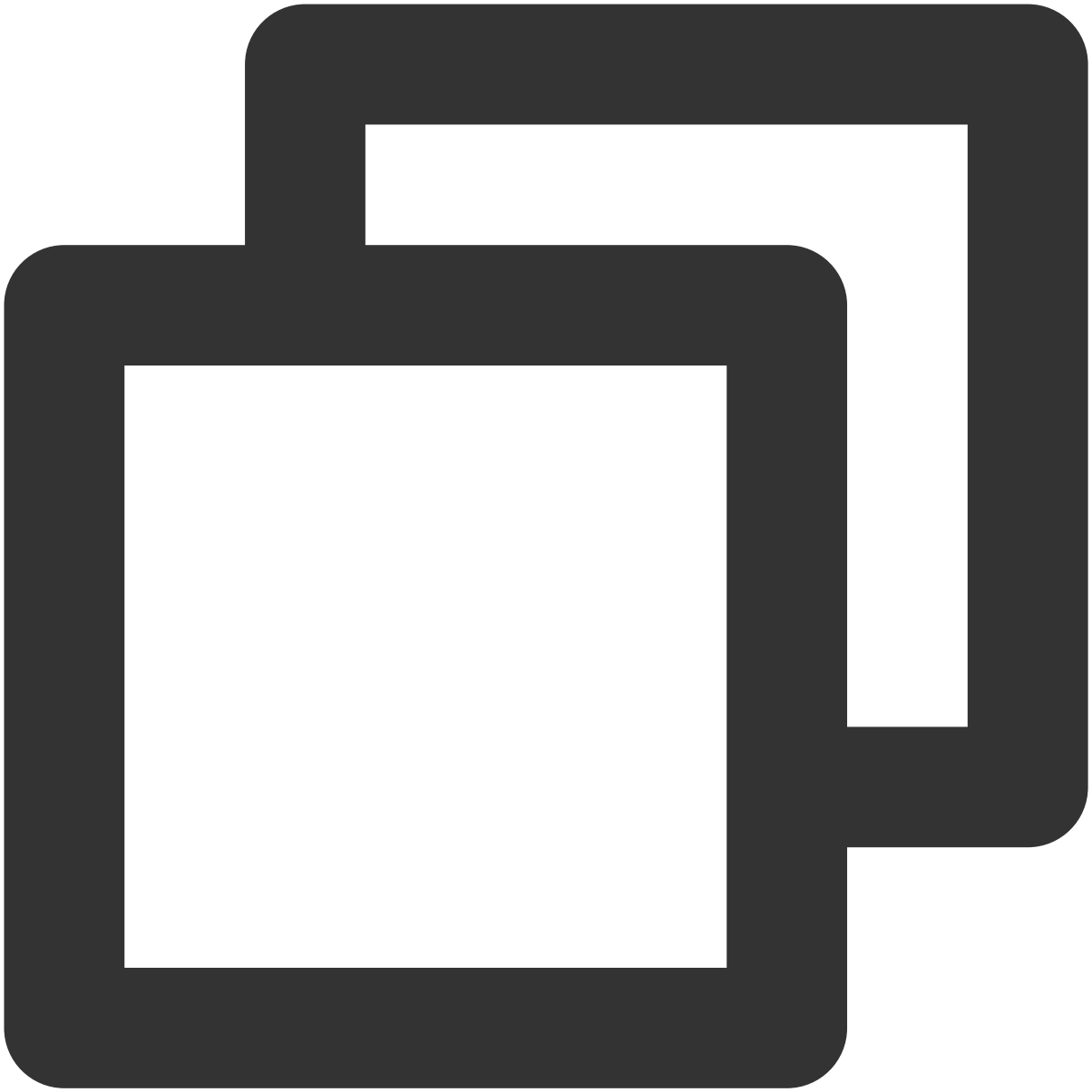
```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit;

TUICommonDefine.RoomId roomId = new TUICommonDefine.RoomId();
roomId.intRoomId = 123321;
String groupId = "12345678";
TUICallKit.createInstance(context).joinInGroupCall(roomId, groupId, TUICallDefine.M
```



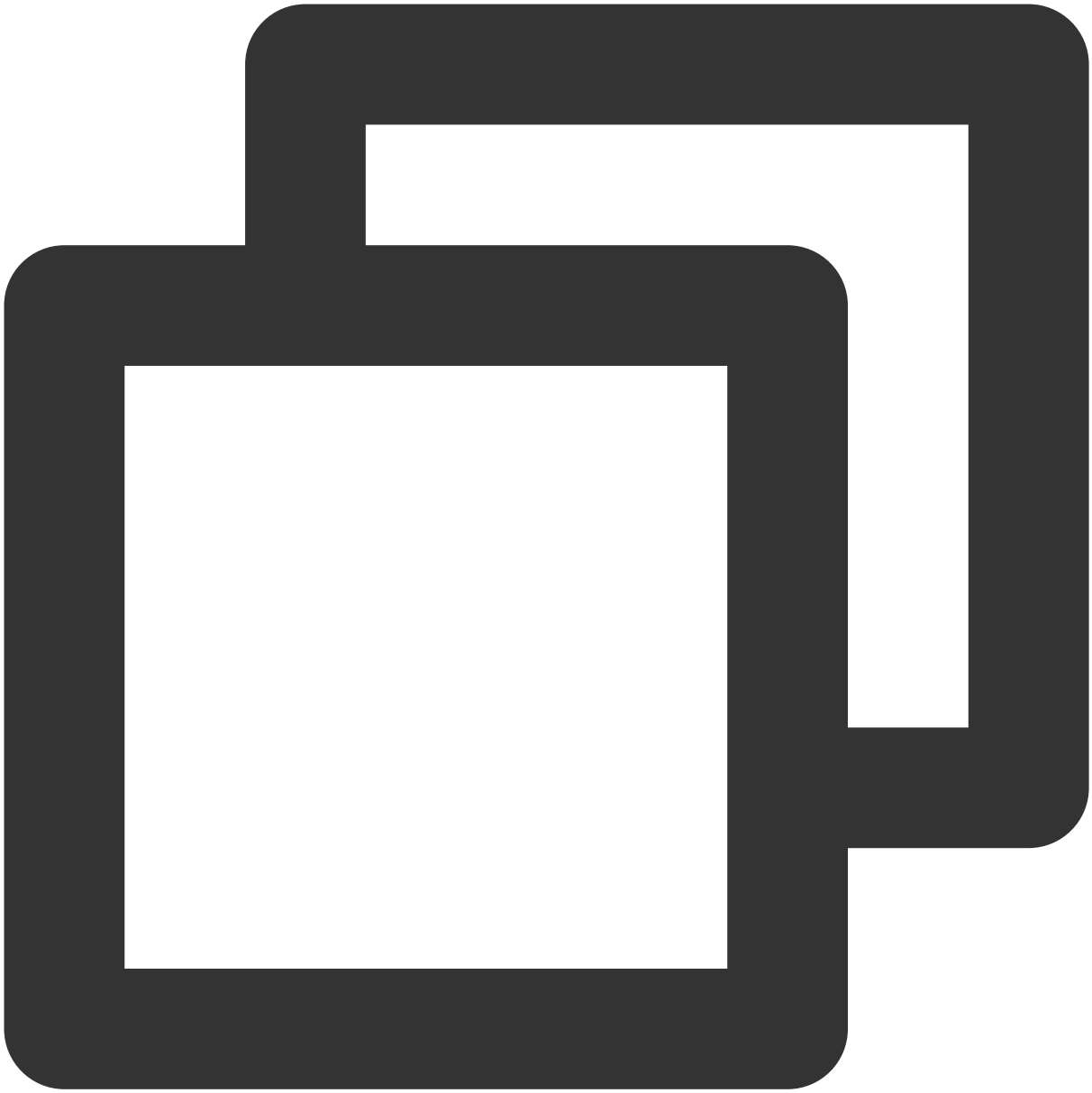
```
import TUICallKit_Swift
import TUICallEngine

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



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

TUIRoomId *roomId = [[TUIRoomId alloc] init];
roomId.intRoomId = 123321;
[[TUICallKit createInstance] joinInGroupCallWithRoomId:roomId
                                               groupId:@"223344"
                                         callMediaType:TUICallMediaTypeVideo];
```



```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

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

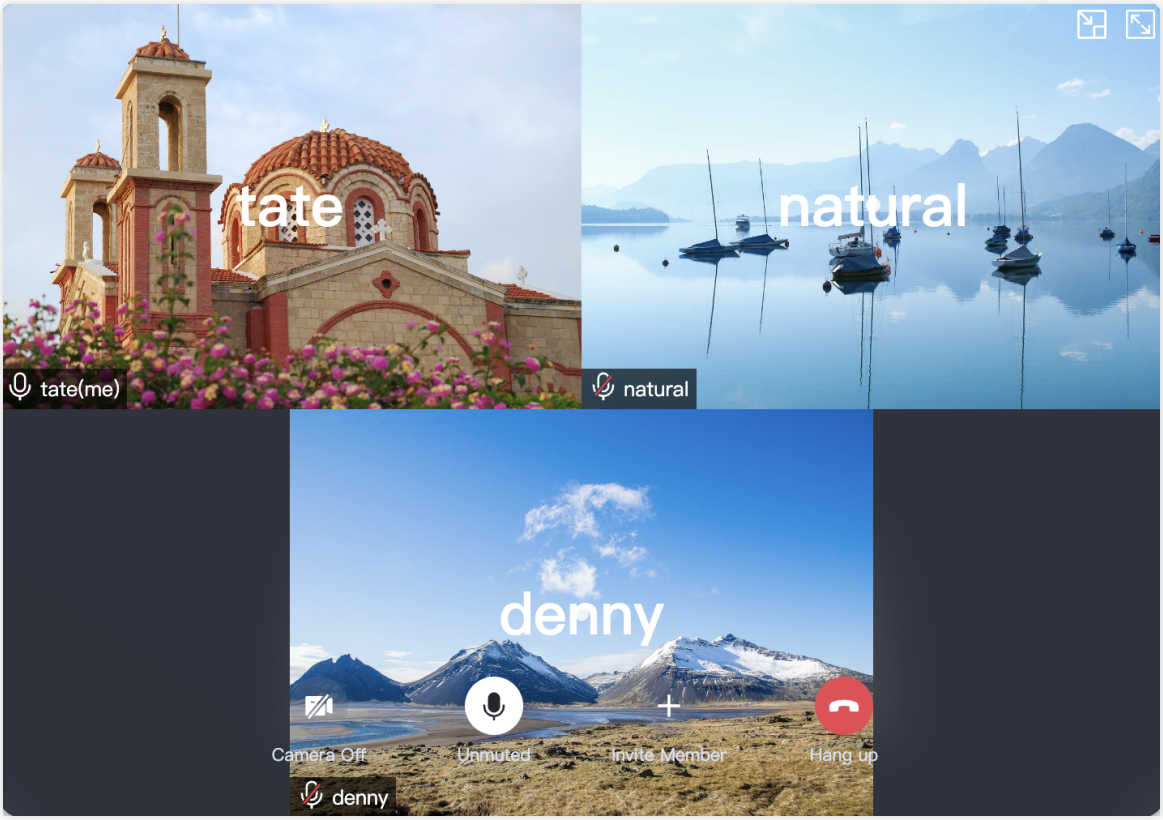
Web&H5

最近更新时间：2024-06-17 17:41:52

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。

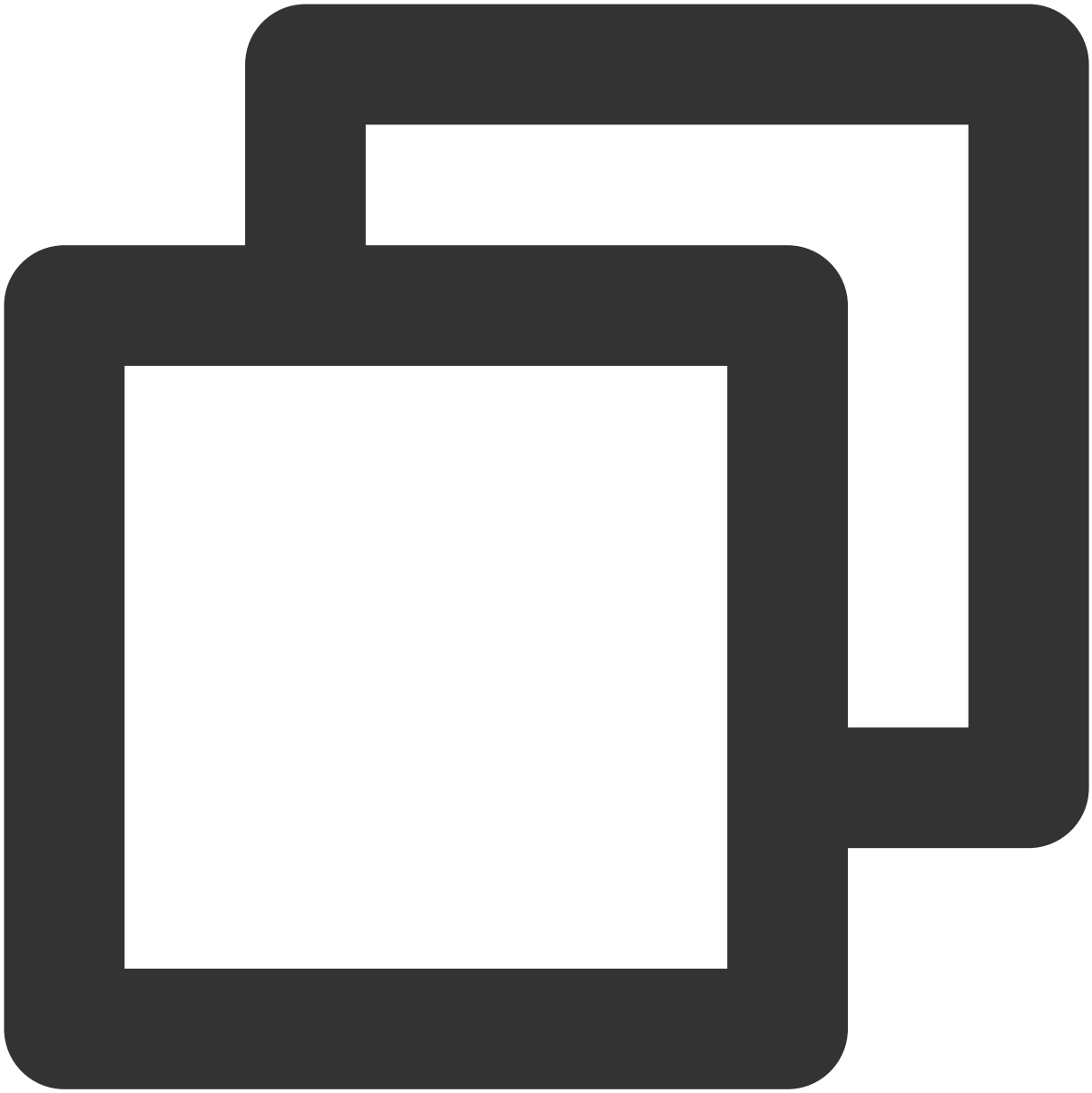
Web 端	移
	

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 Chat API 创建群，详见 Chat 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。



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

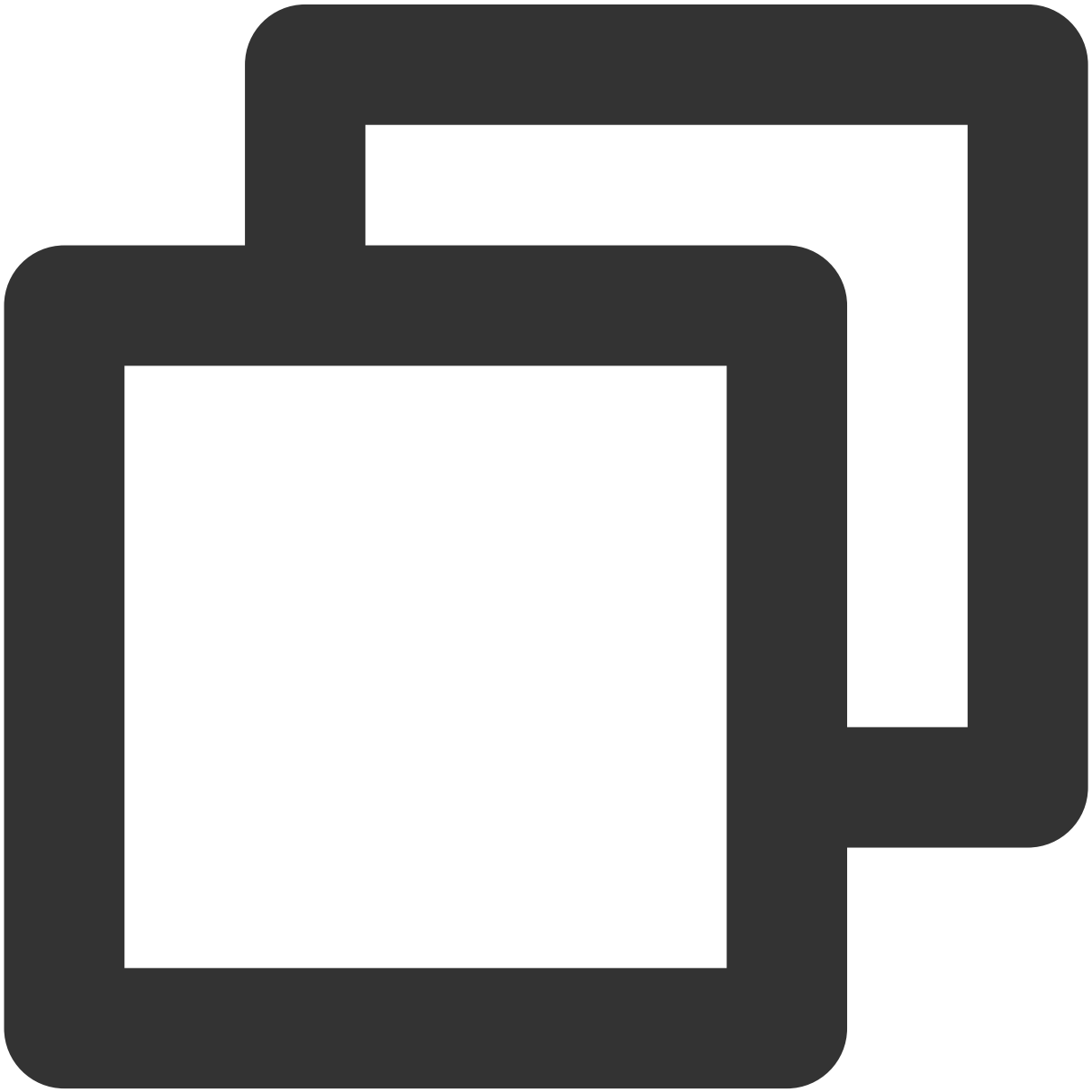
const userIDList: string[] = ['user1', 'user2'];
async function createGroupID() {
  const chat = Chat.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userId => ({ userID: userId }));
  const res = await chat.createGroup({
    type: Chat.TYPES.GRP_PUBLIC, // Must be a public group
  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

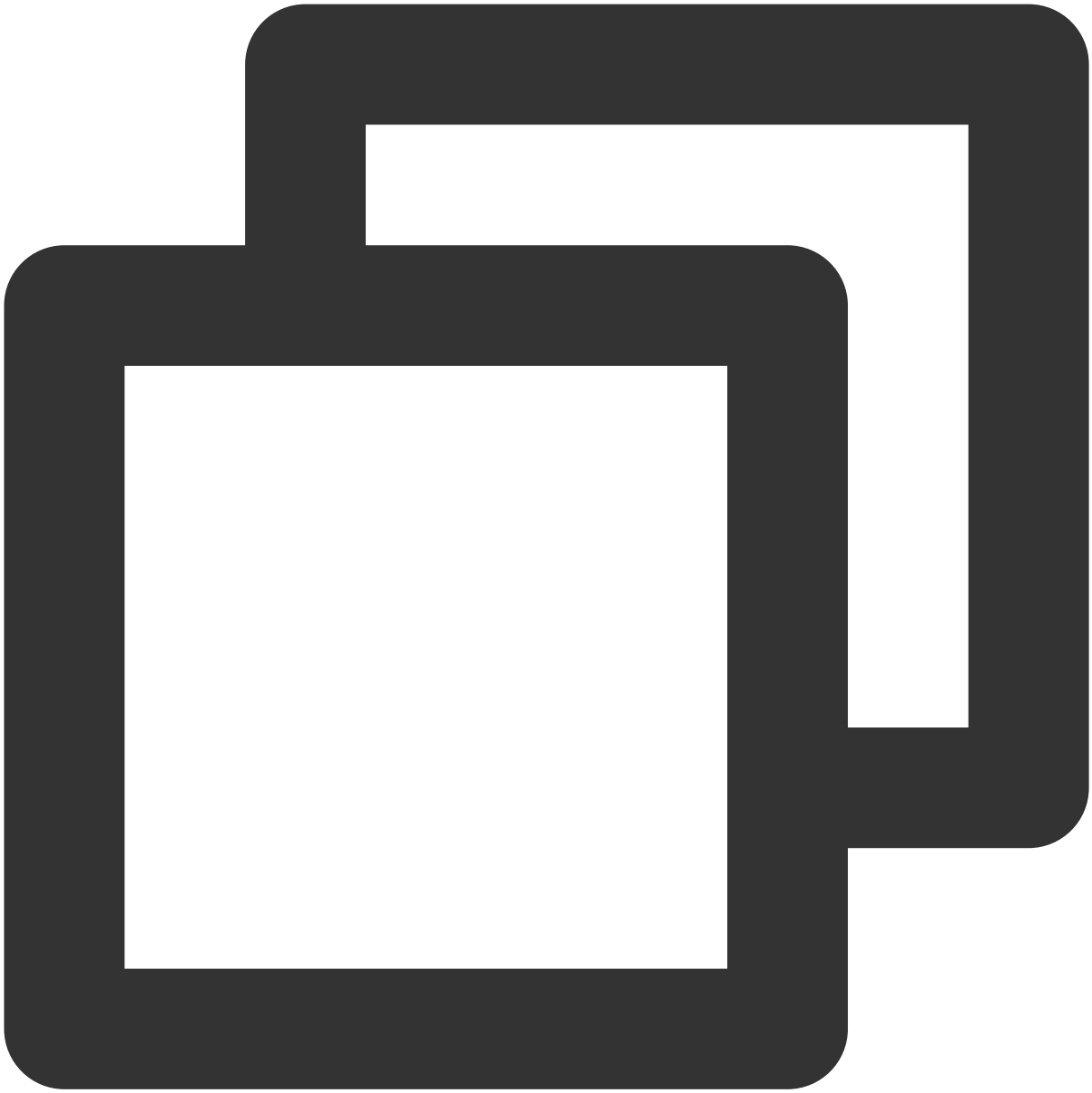
try {
   const params = {
     userIDList: ['user1', 'user2'],
     groupID: 'xxx',
     type: TUICallType.VIDEO_CALL,
   }
  await TUICallKitServer.groupCall(params);
} catch (error: any) {
  console.error(`[TUICallKit] groupCall failed. Reason:${error}`);
}
```

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：

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) {
```

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

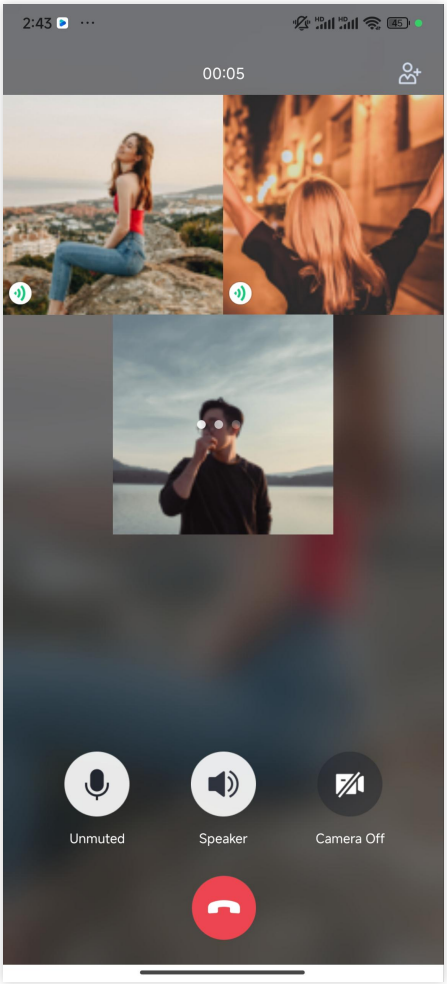
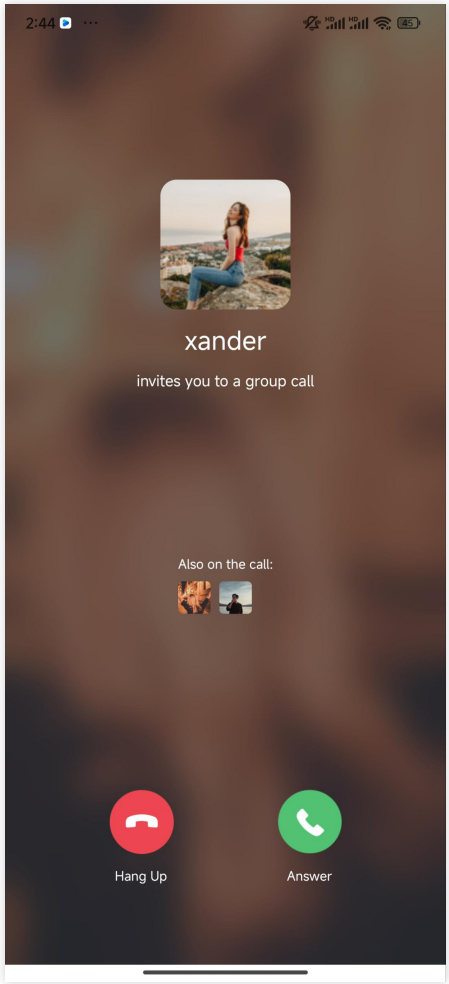
uni-app (Anroid&iOS)

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
		

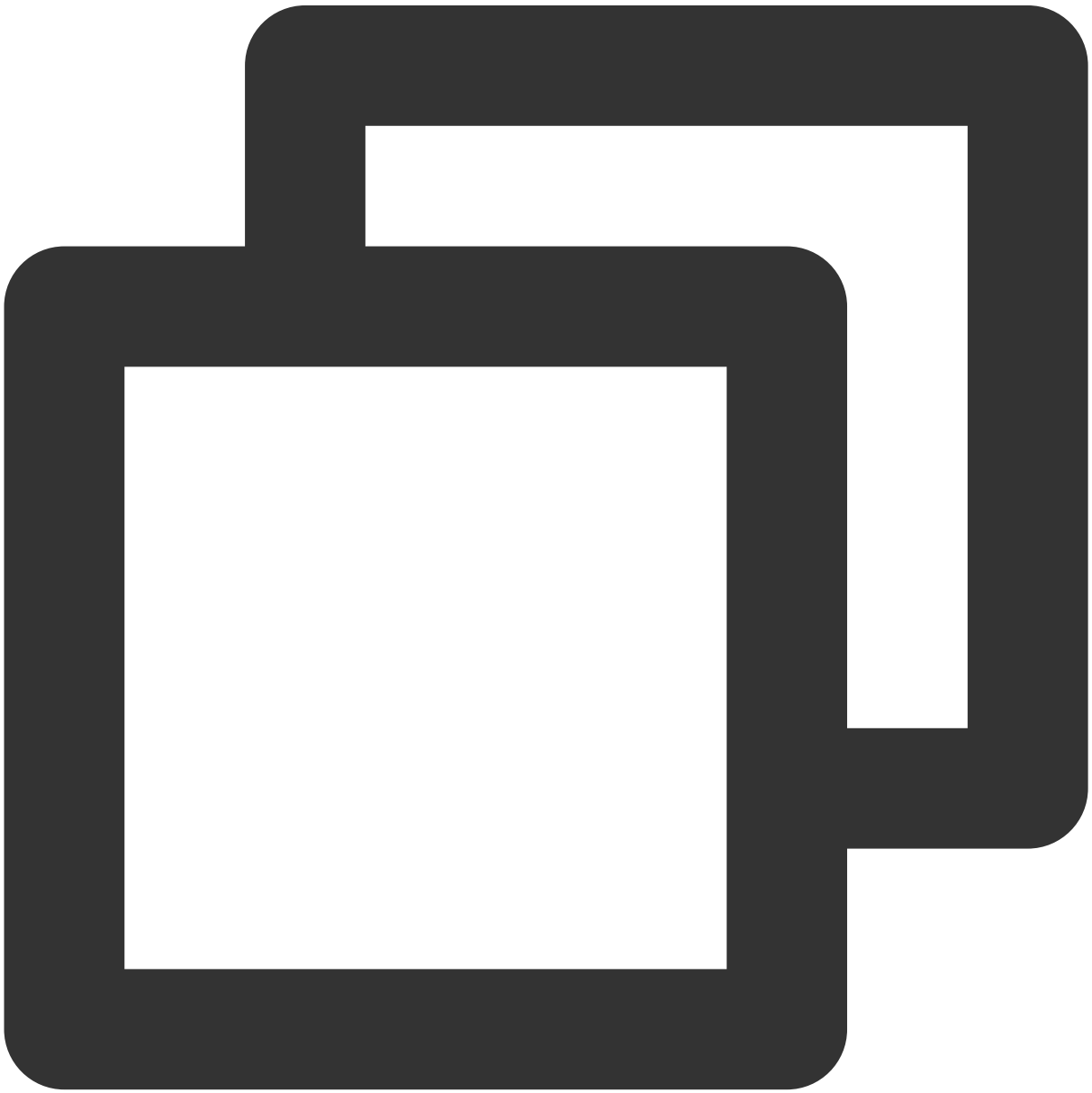
创建 groupID

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

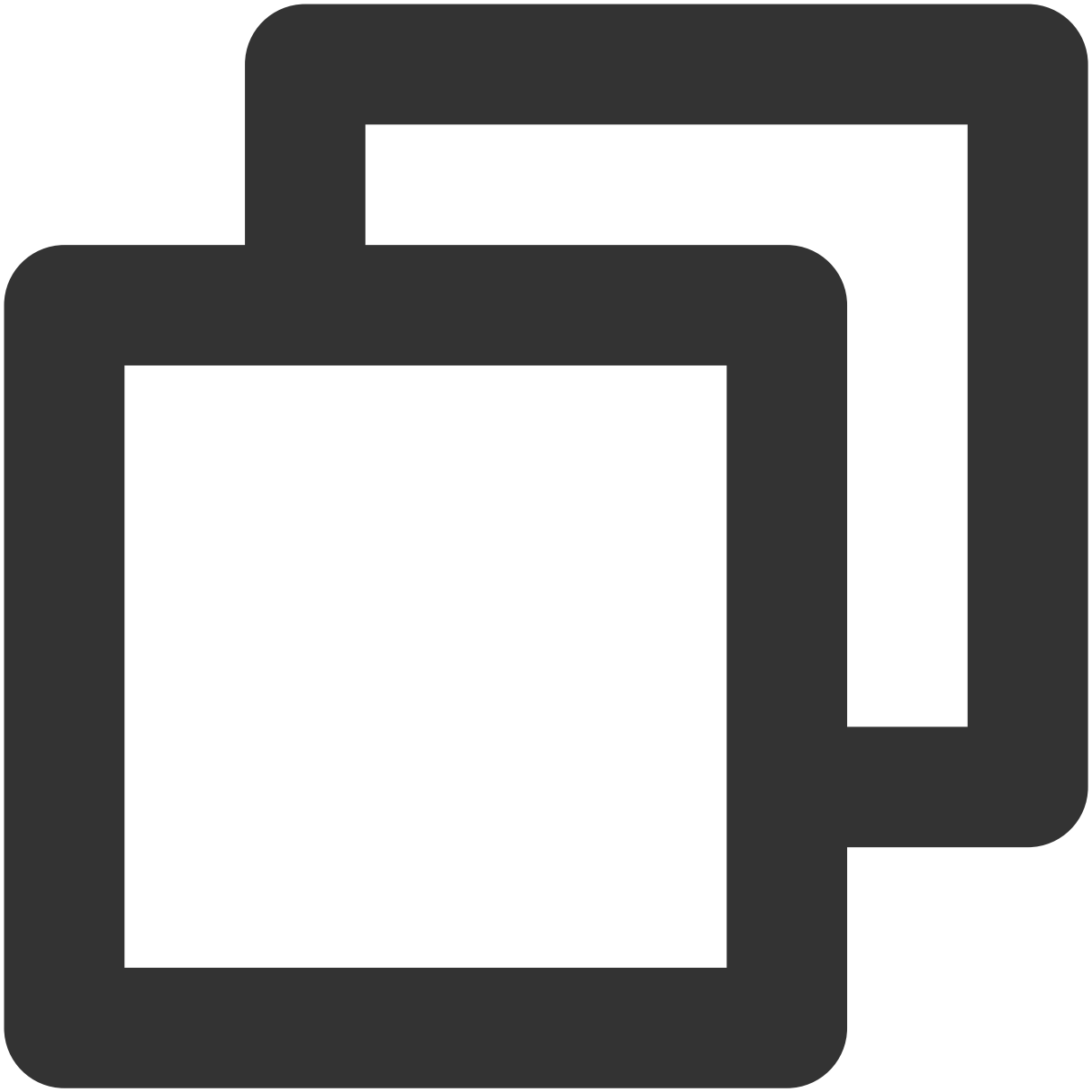
调用 groupCall API 发起群通话。



```
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}`);
```

```
    }
});
```

版权所有：腾讯云计算（北京）有限责任公司

悬浮窗

Android&iOS&Flutter

最近更新时间：2024-05-24 18:35:12

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
	

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：

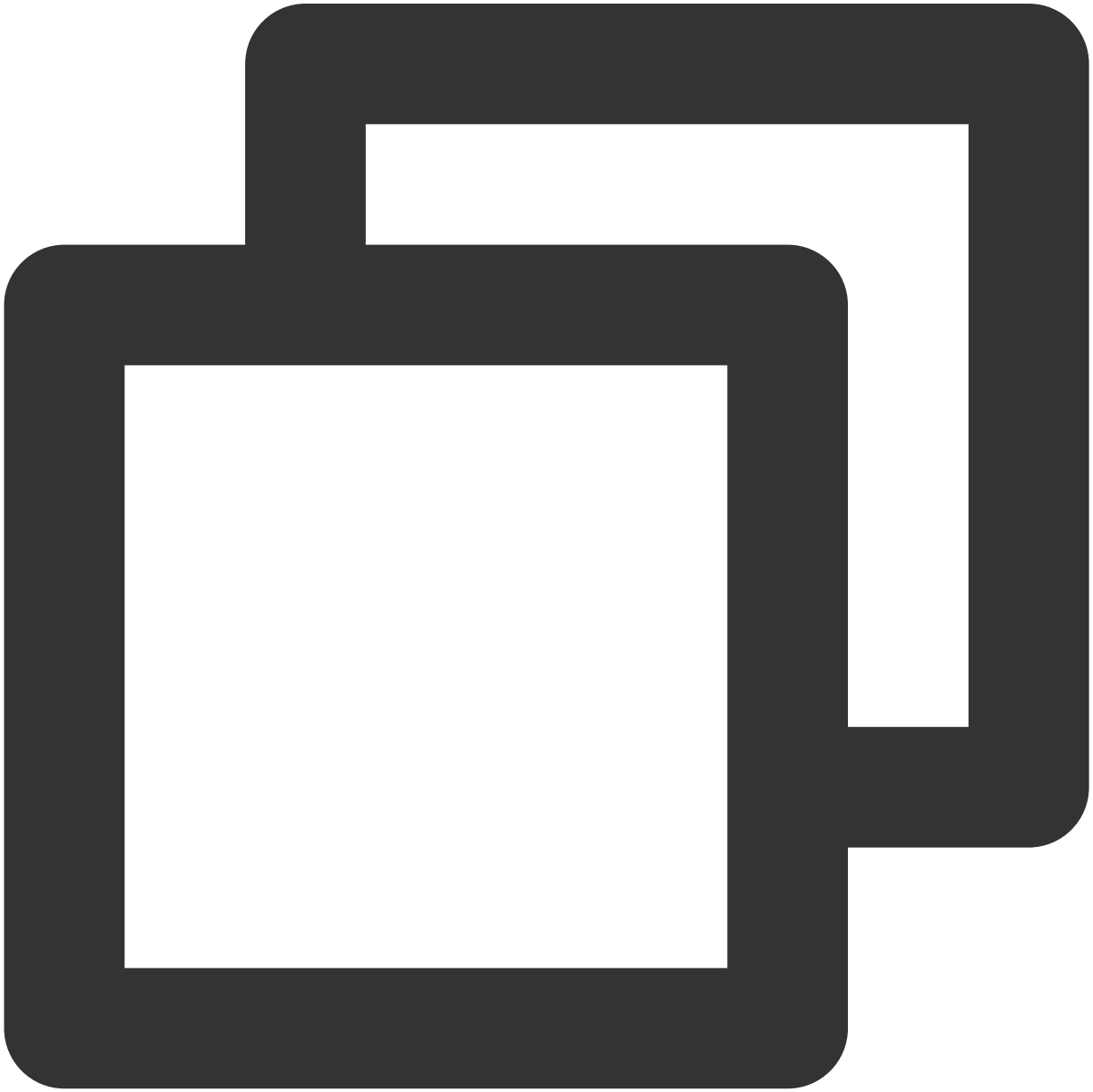
Android（Kotlin）

Android（Java）

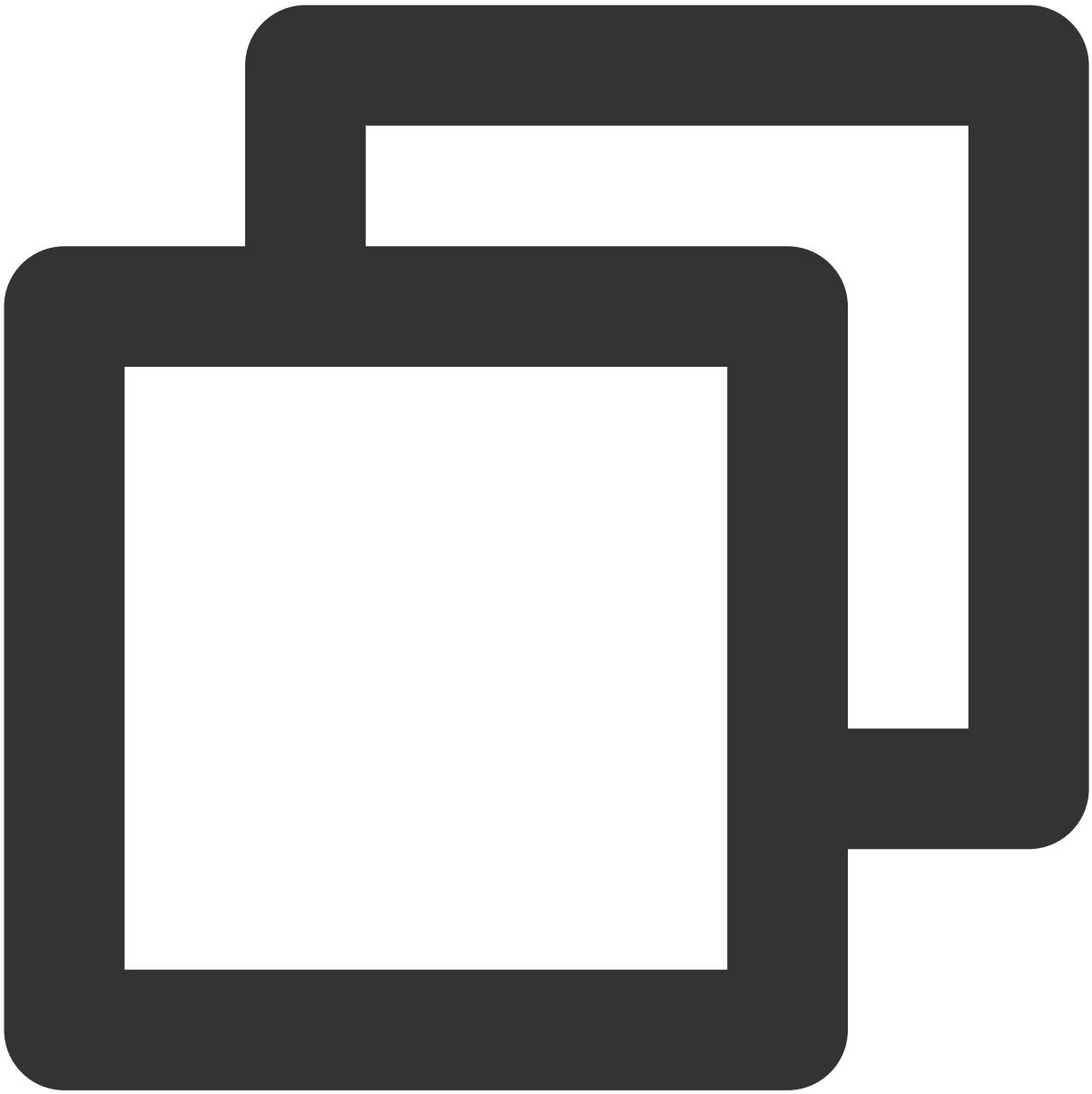
iOS（Swift）

iOS（Objective-C）

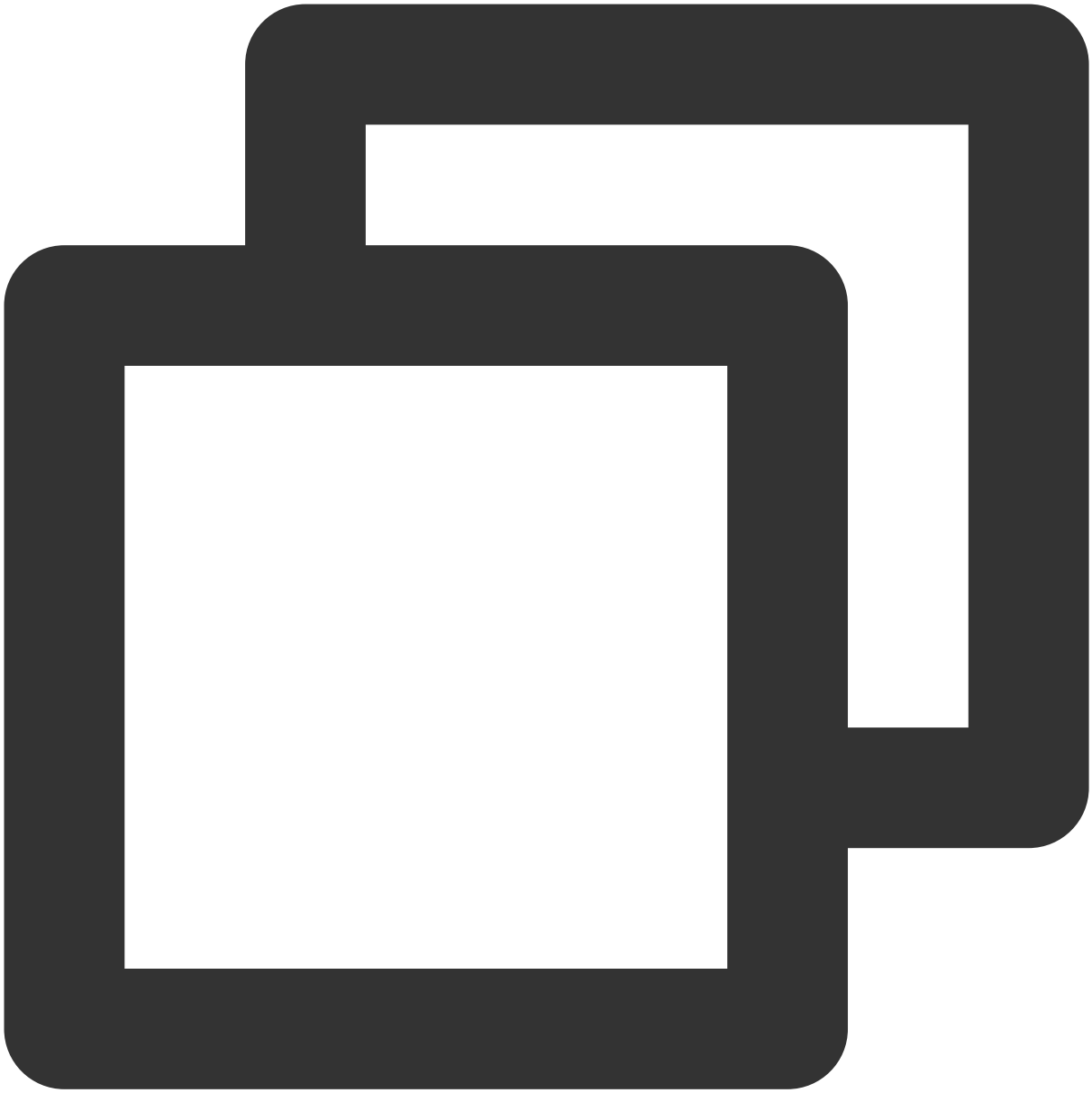
Flutter（Dart）



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

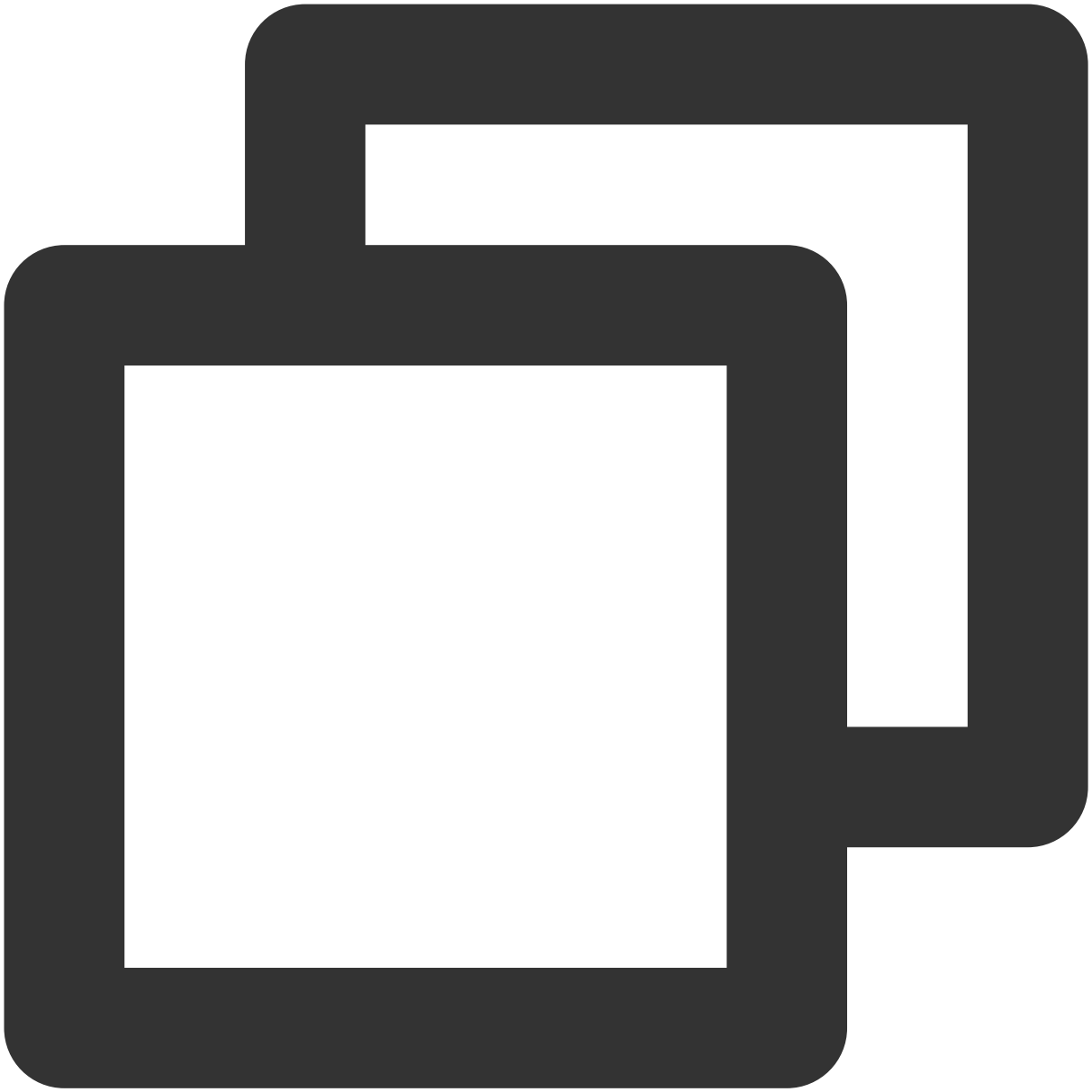


```
TUICallKit.createInstance(context).enableFloatWindow(true);
```



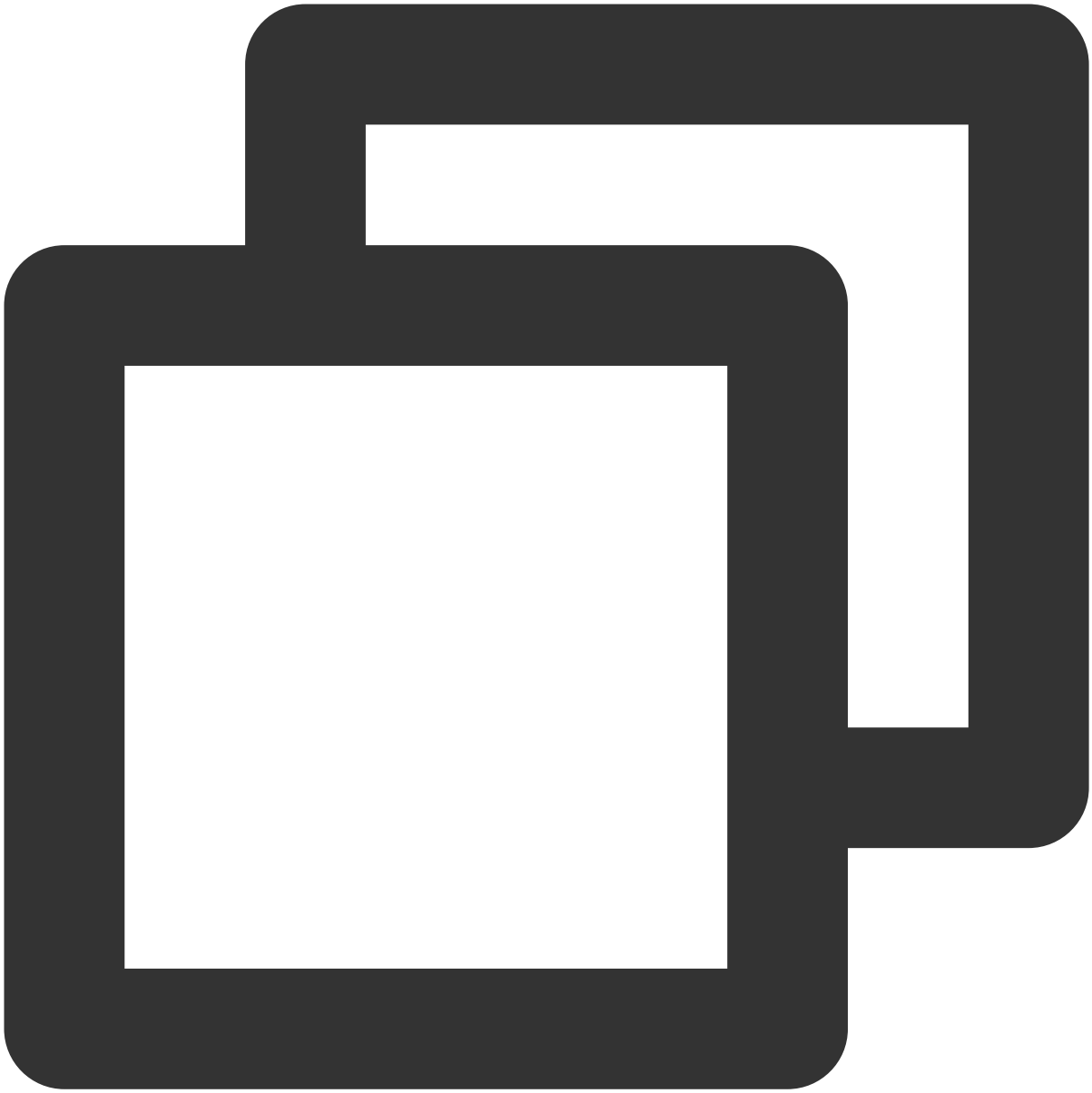
```
import TUICallKit_Swift

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

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



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

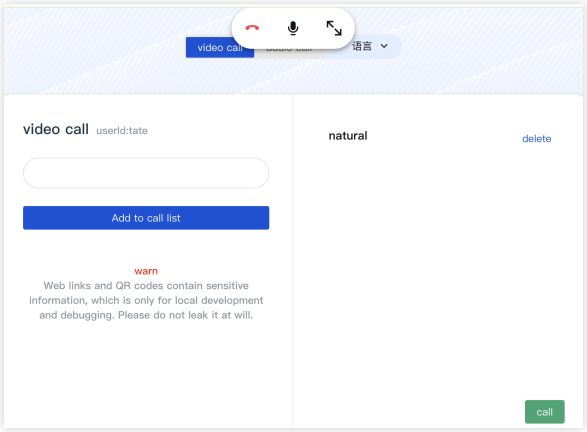
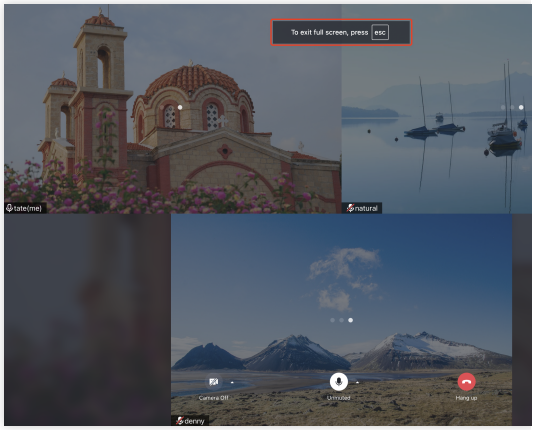
void enableFloatWindow() {
    TUICallKit.instance.enableFloatWindow(true);
}
```

Web&H5

最近更新时间：2024-05-24 18:35:12

本文将介绍悬浮窗功能的使用。

预期效果

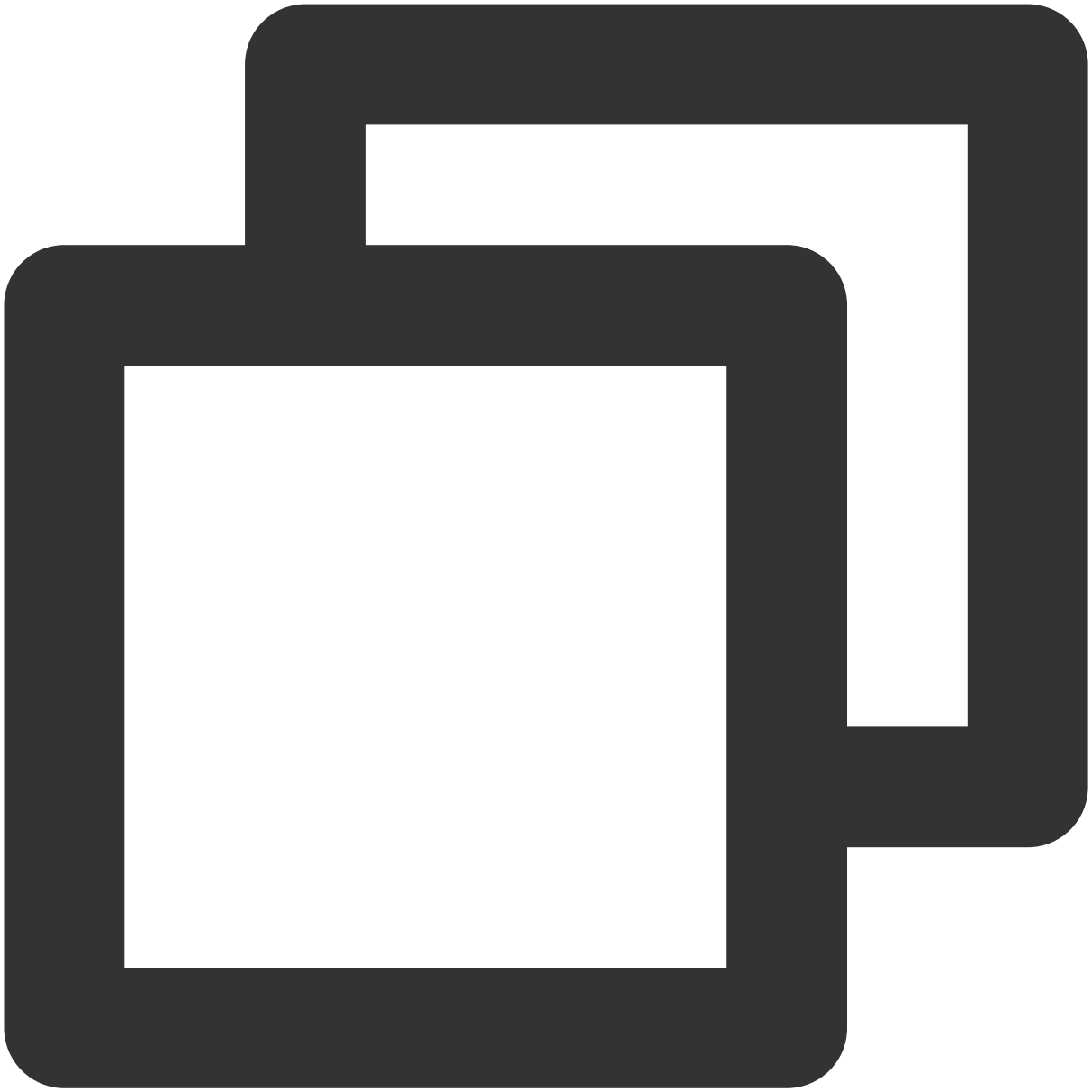
Web 端悬浮窗	web 端全屏
	

悬浮窗功能

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

说明：

v3.1.0+ 支持。



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

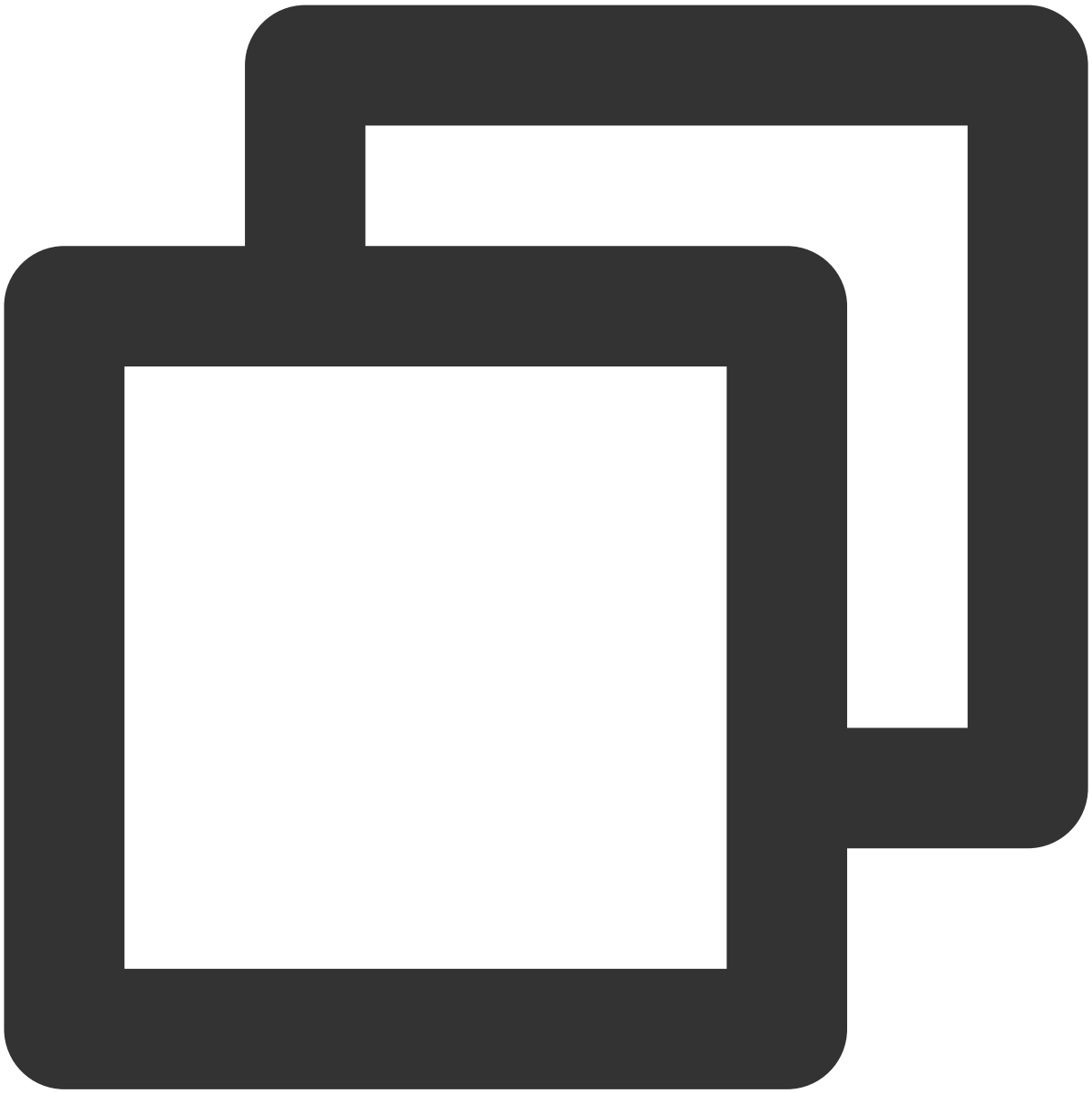
方式二：通过属性控制悬浮窗、全屏的开启/关闭。

`allowedMinimized` 属性控制控制悬浮窗开启/关闭。

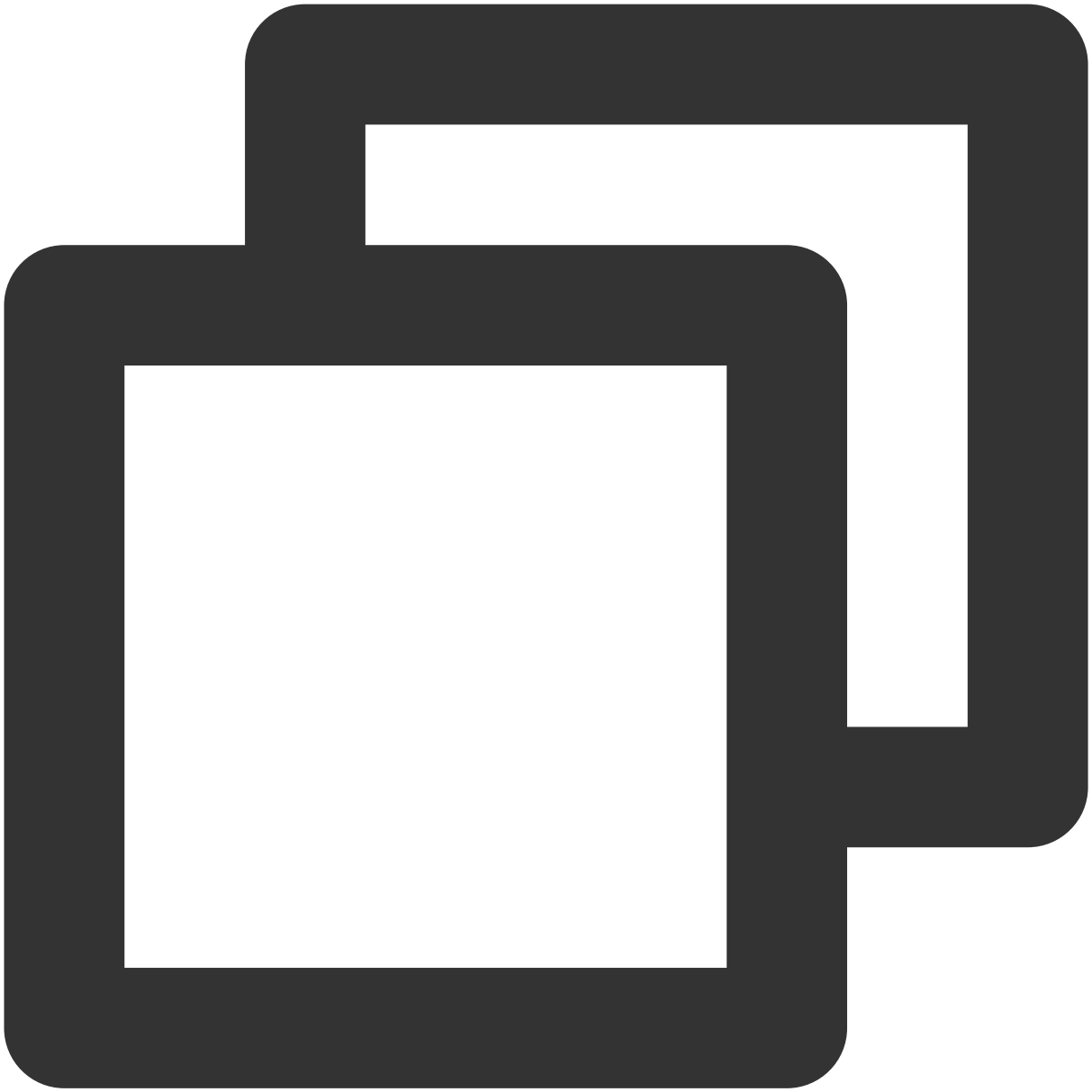
`allowedFullScreen` 属性控制全屏的开启/关闭。

React

Vue



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



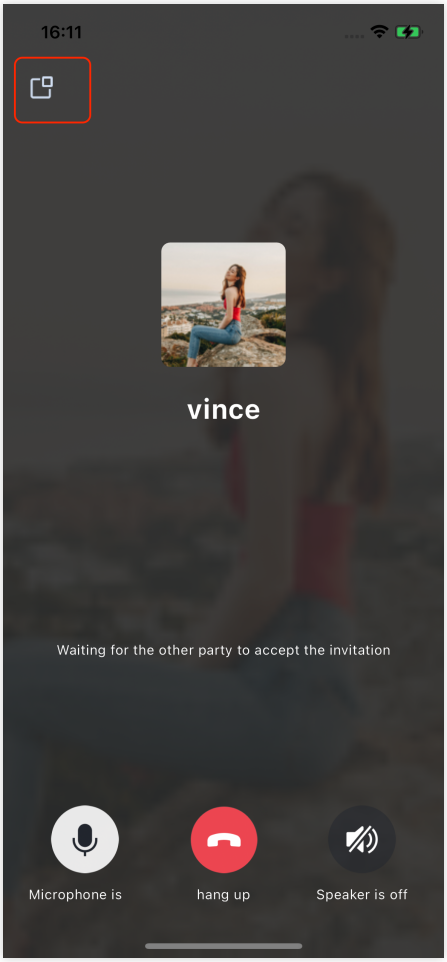
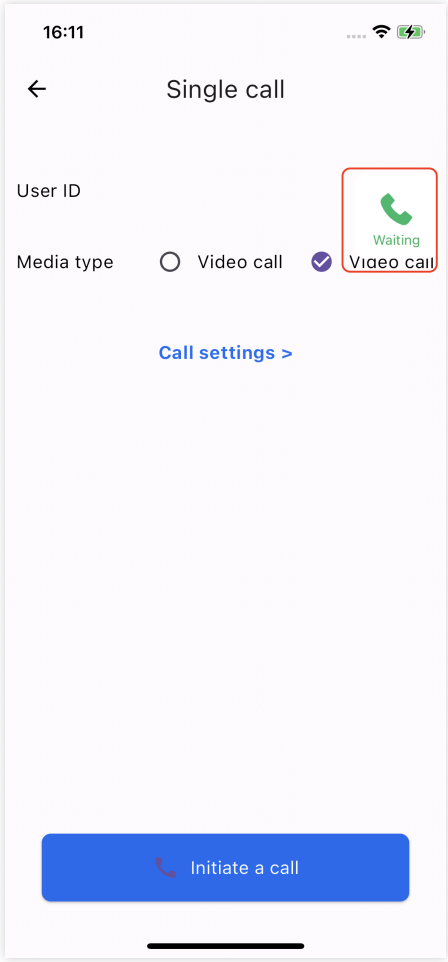
```
<TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
   />
```

uni-app (Anroid&iOS)

最近更新时间：2024-04-03 17:23:11

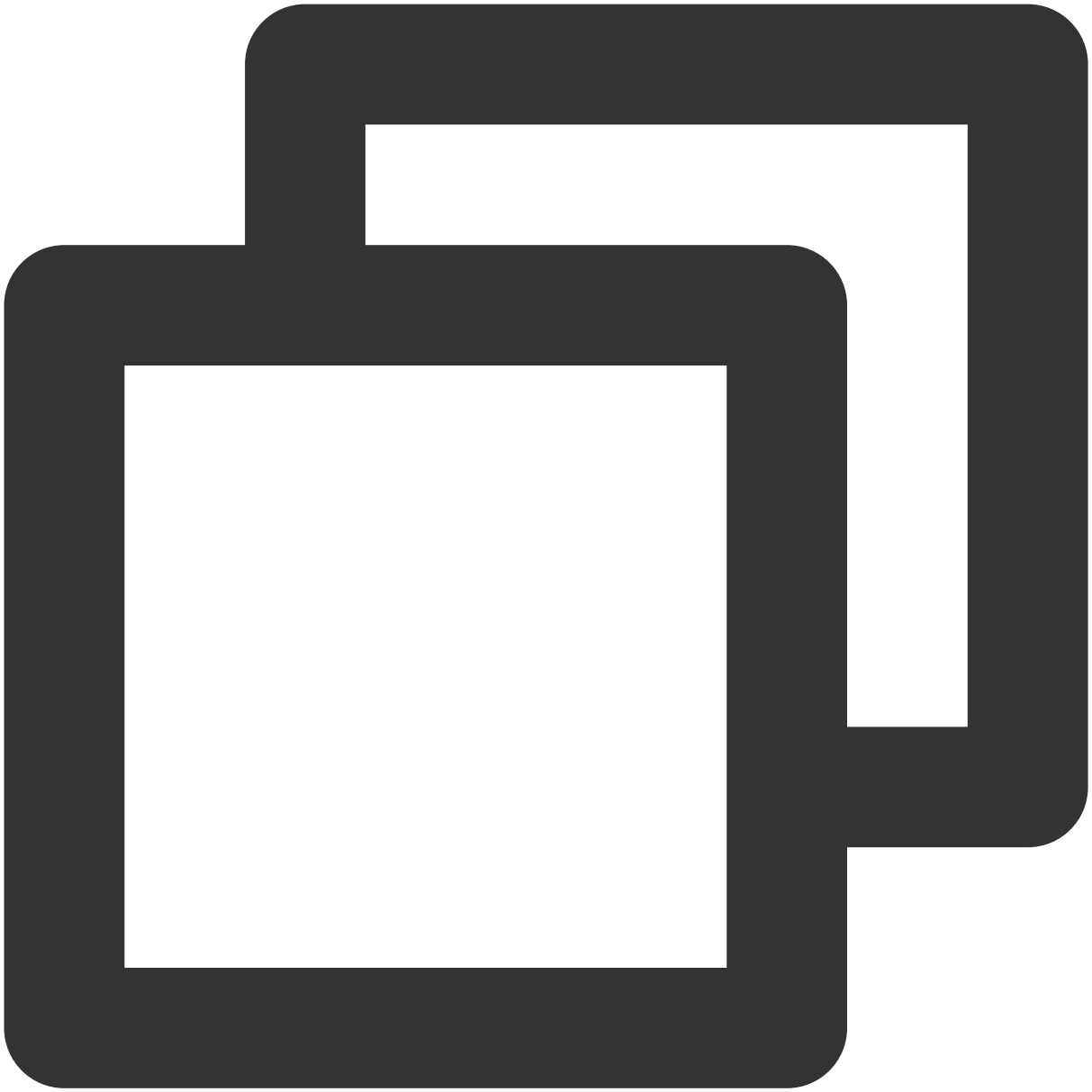
本文介绍如何使用悬浮窗功能。

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。



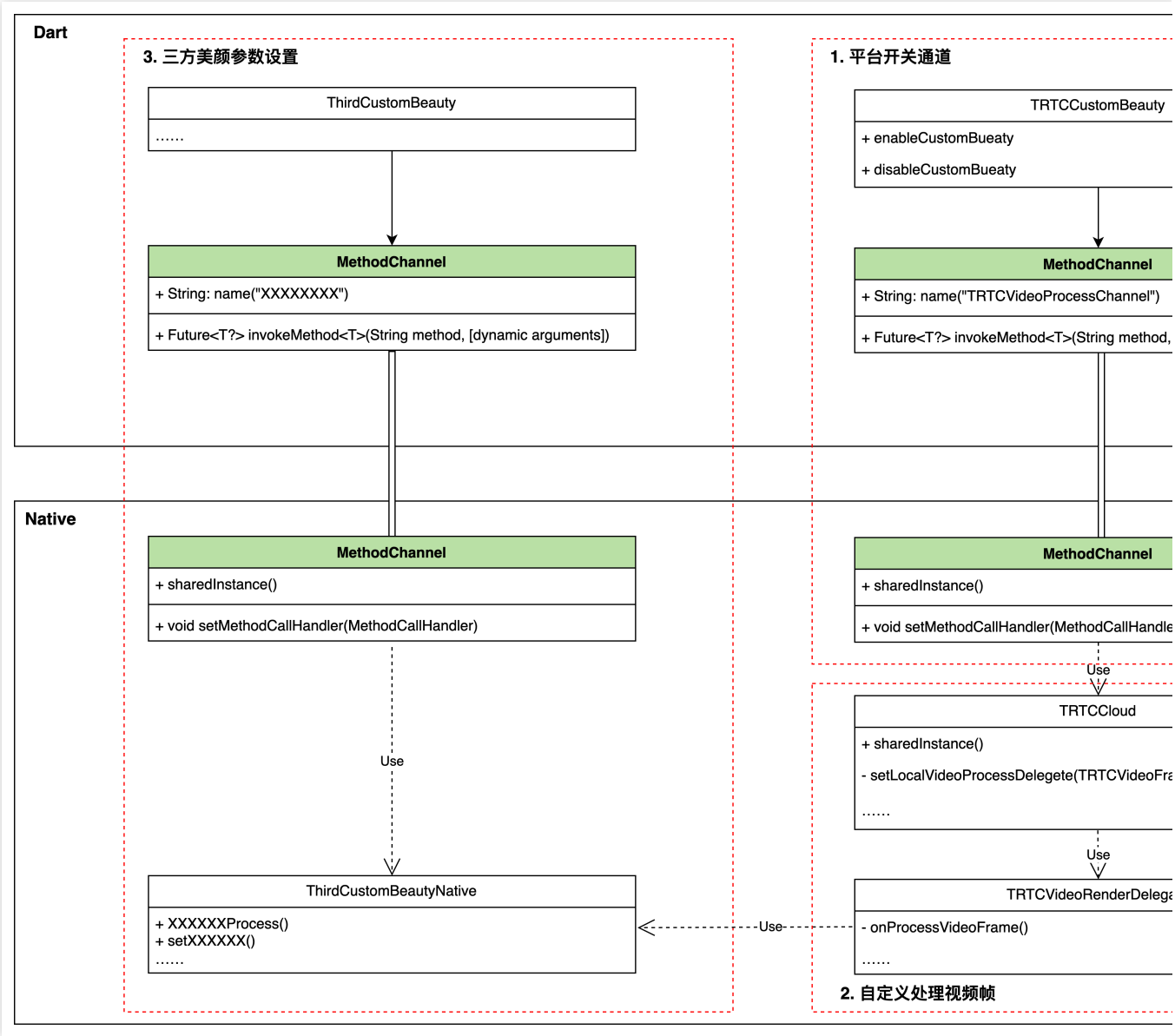
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

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：



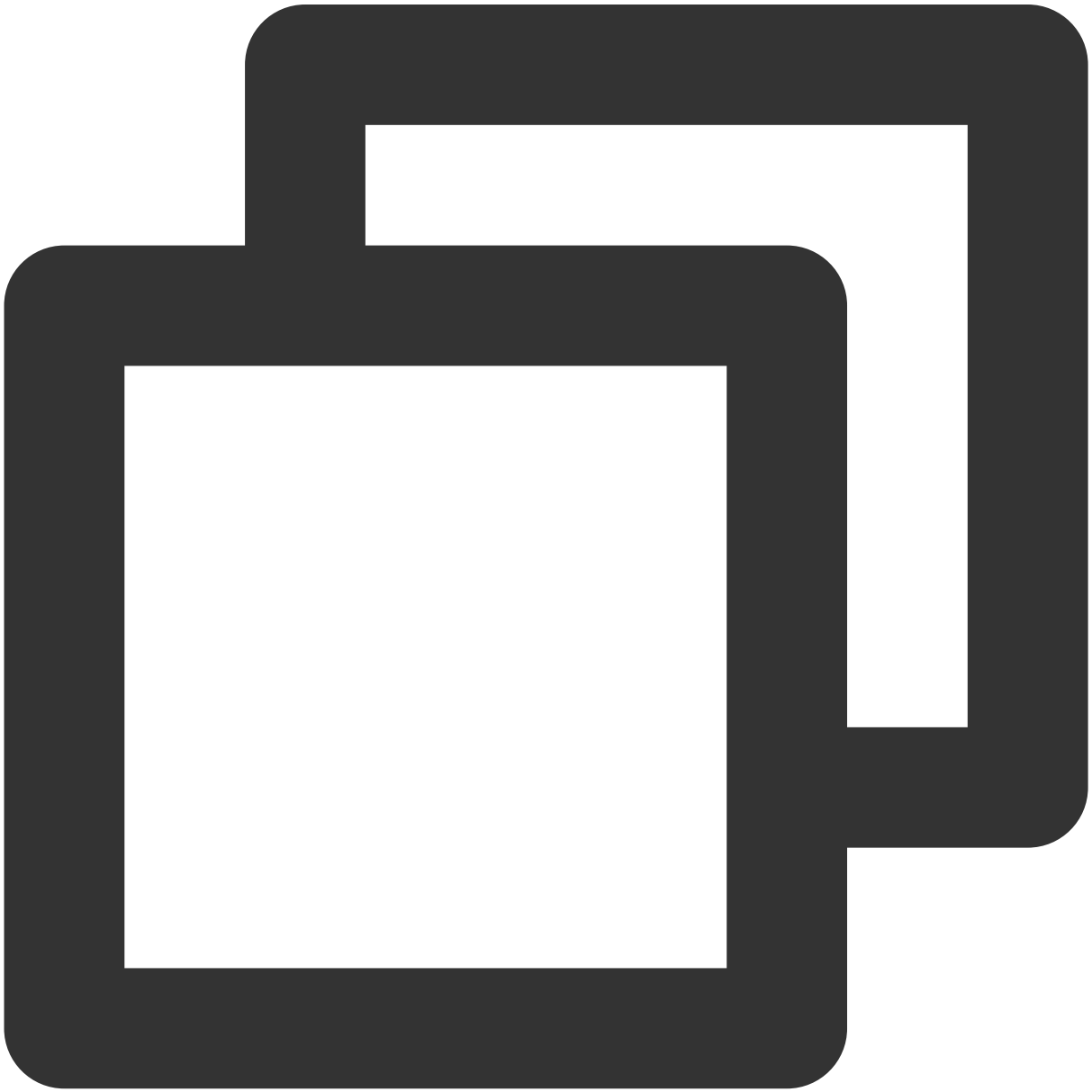
接入方案分为 3 步：
第一步：通过 MethodChannel 通道 开启/关闭 TRTC 自定义渲染逻辑。
第二步：在 TRTC 自定义渲染处理逻辑 onProcessVideoFrame() 中使用 `美颜处理模块` 对原始视频帧进行处理。

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');

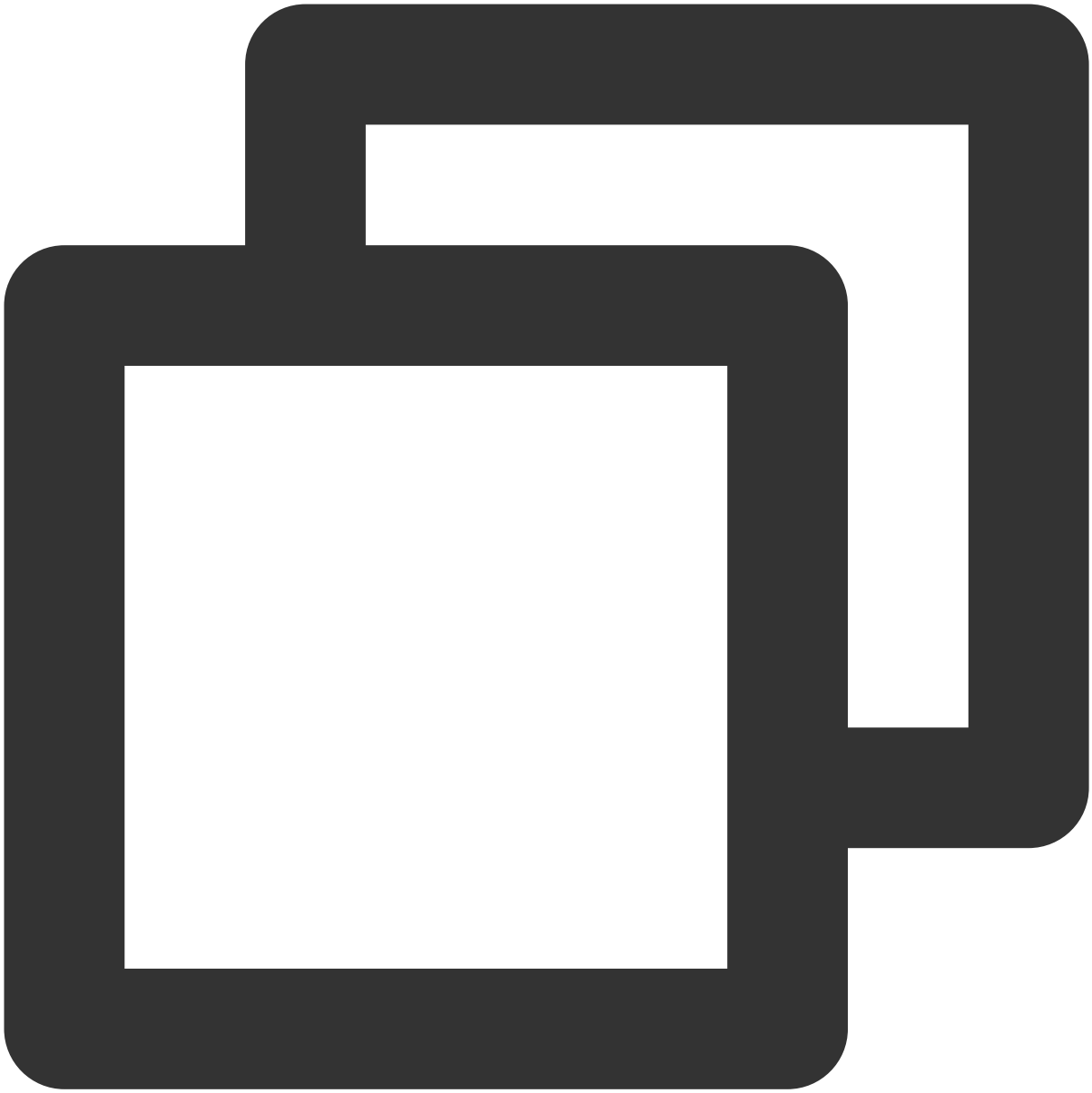
void enableTUICallKitCustomBeauty() async {
   await channel.invokeMethod('enableTUICallKitCustomBeauty');
}

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) -> {
```

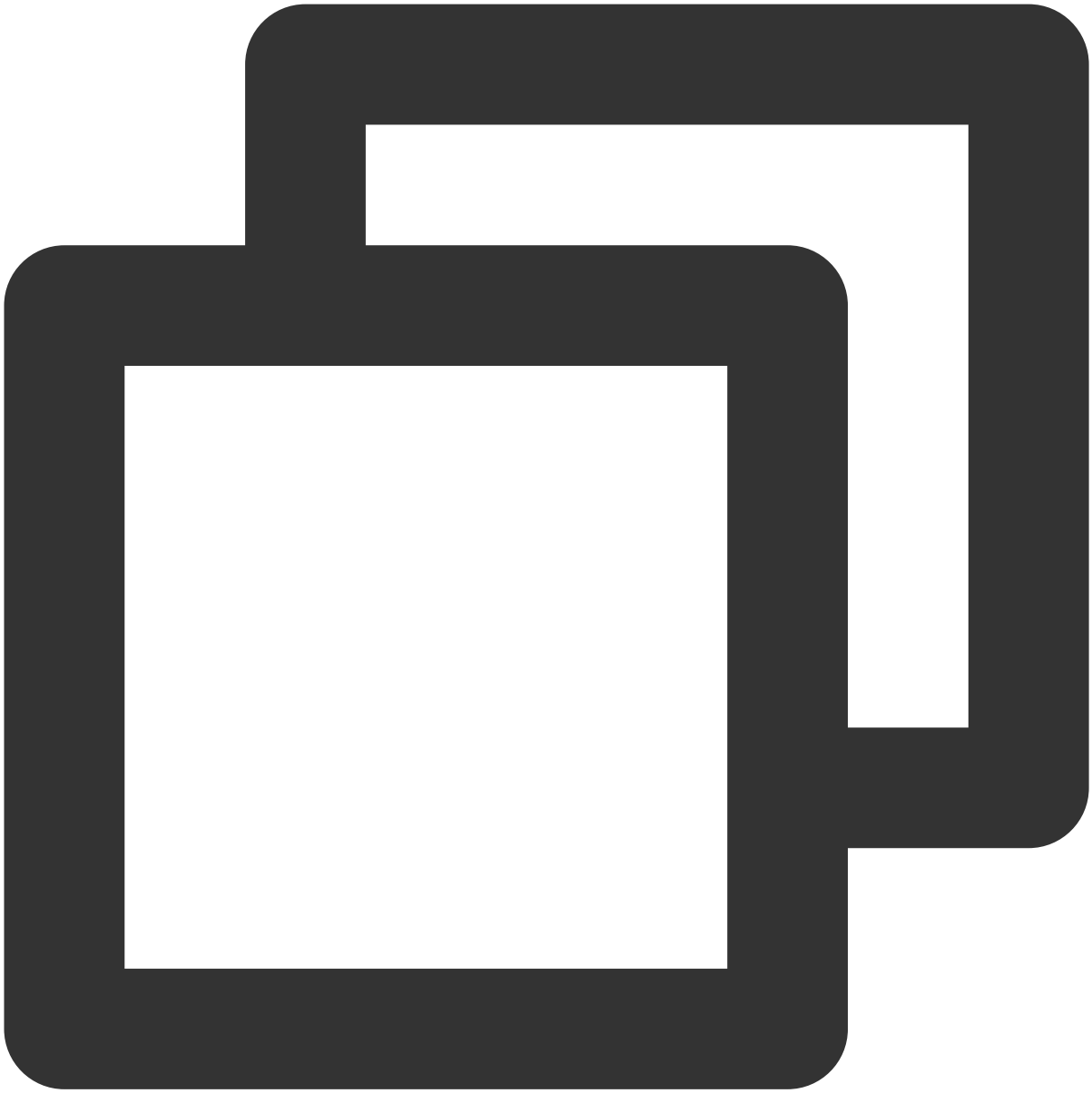
```
            switch (call.method) {
                case "enableTUICallKitCustomBeauty":
                    enableTUICallKitCustomBeauty();
                    break;
                case "disableTUICallKitCustomBeauty":
                    disableTUICallKitCustomBeauty();
                    break;
                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

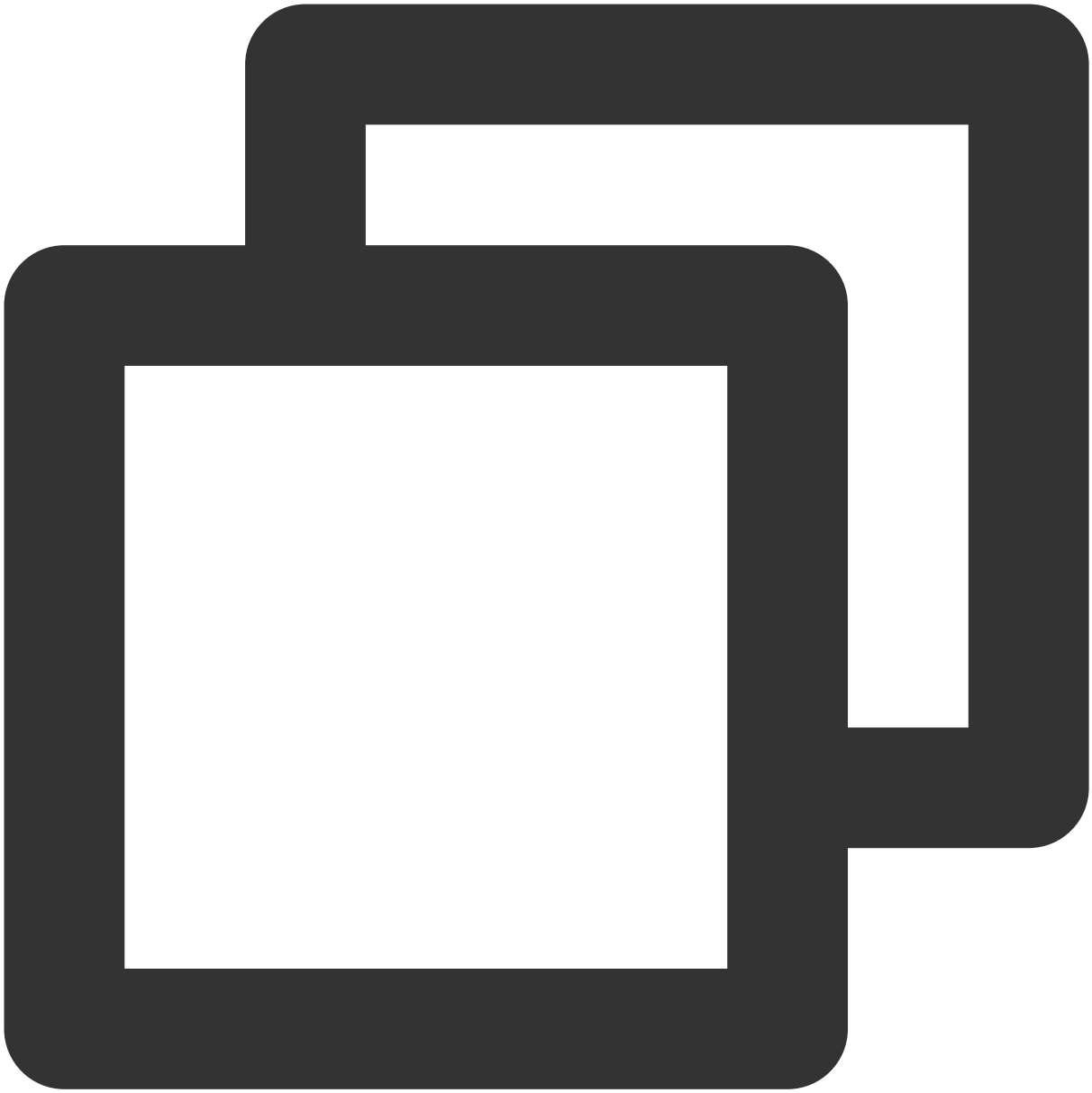
注意：

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);
```

```
}

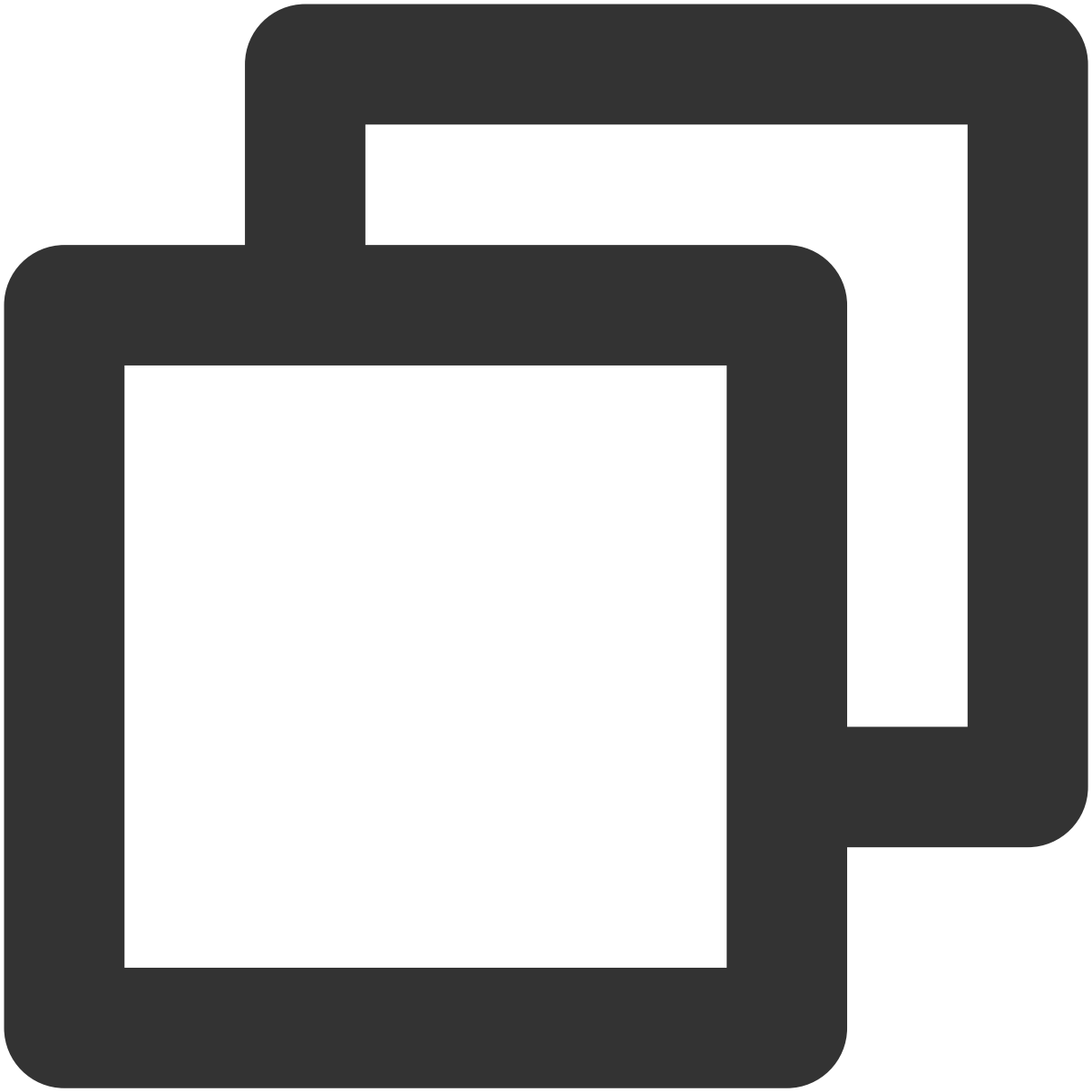
class VideoFrameListerer implements TRTCCloudListener.TRTCVideoFrameListener {
    private XXXBeautyModel  mBeautyModel = XXXBeautyModel.sharedInstance();

    @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：

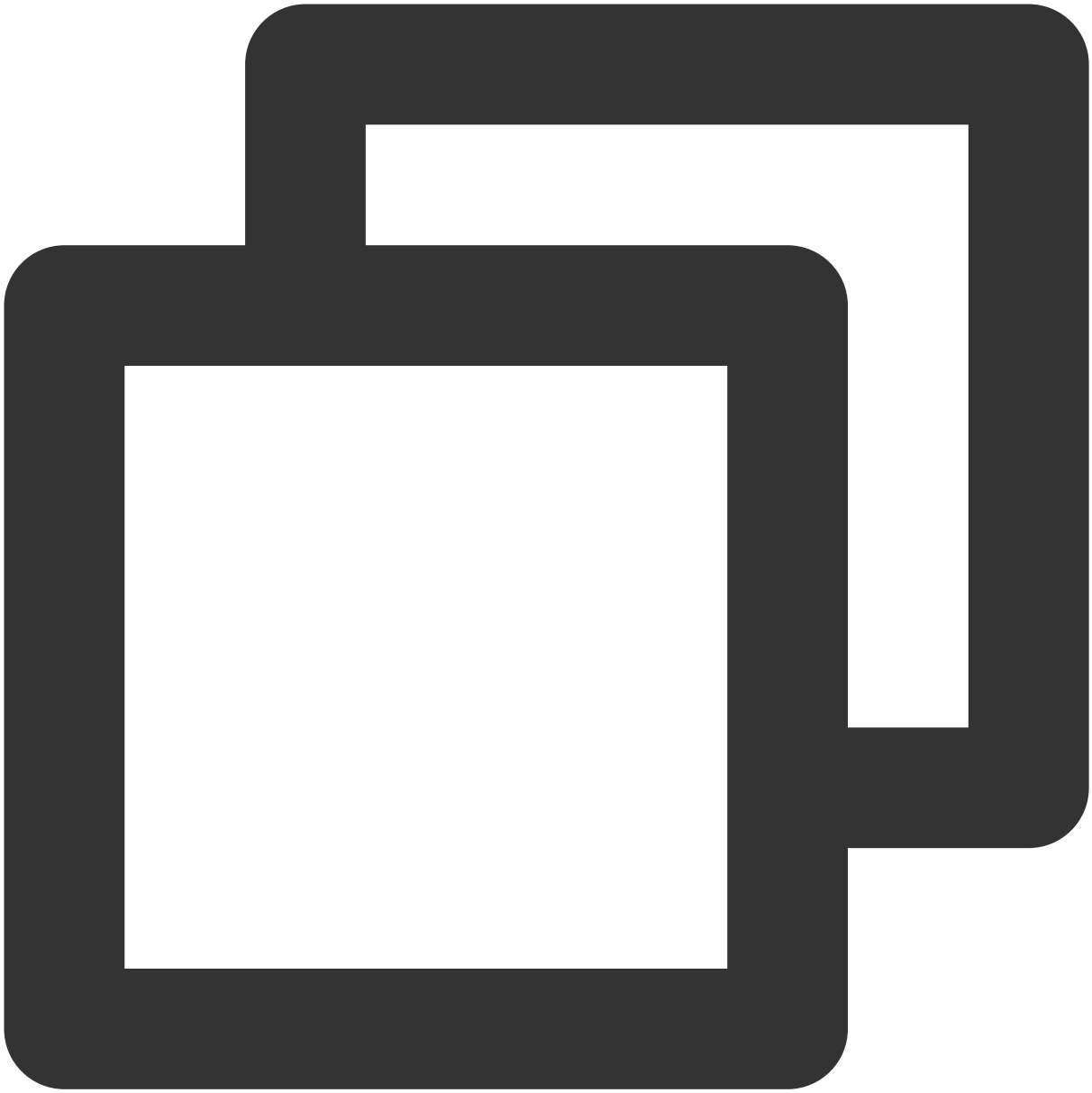
步骤1：美颜资源下载与集成

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：

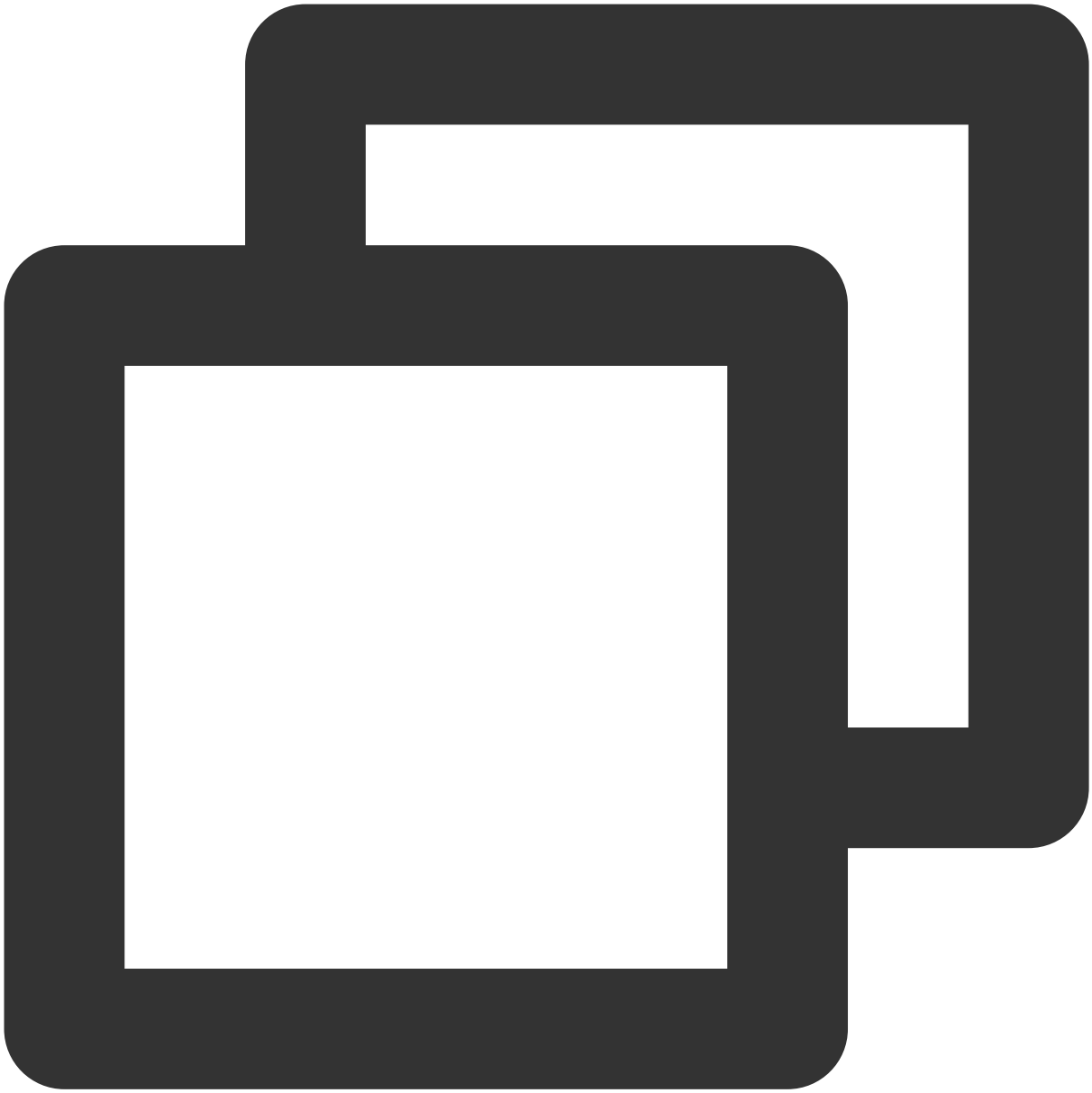


```
dependencies {
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

各套餐对应的 maven 地址，请参见 文档。

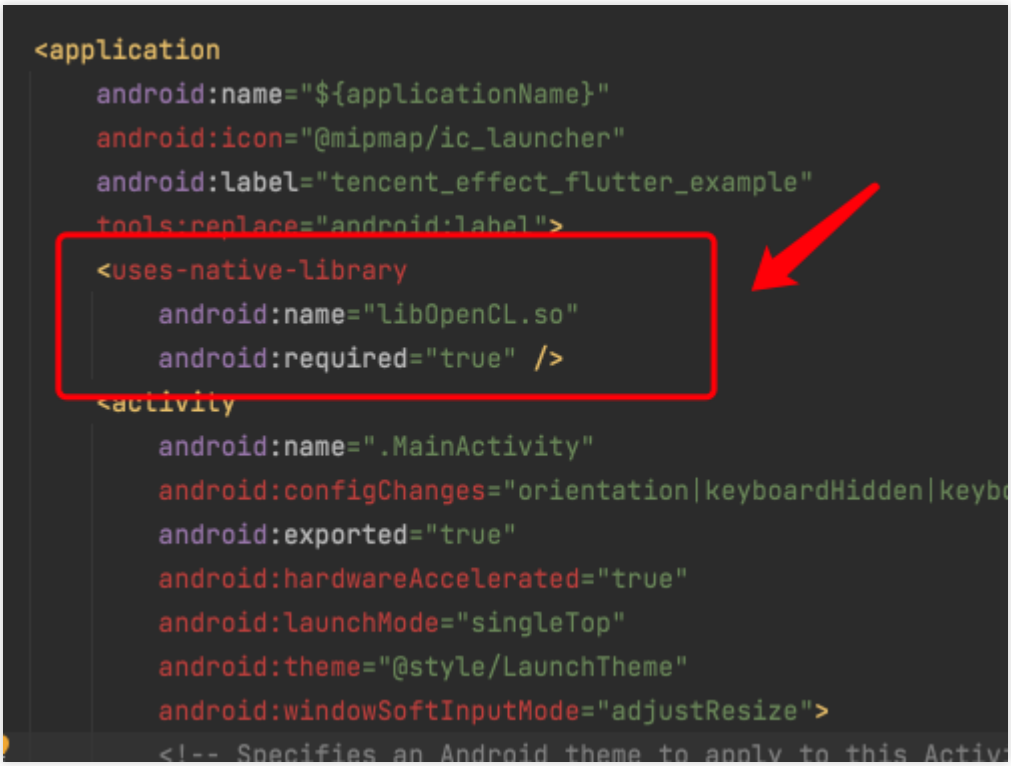
2. 在 app 模块下找到 src/main/assets 文件夹，如果没有则创建，检查下载的 SDK 包中是否有 MotionRes 文件夹，如果有则将此文件夹拷贝到 `../src/main/assets` 目录下。

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-
```

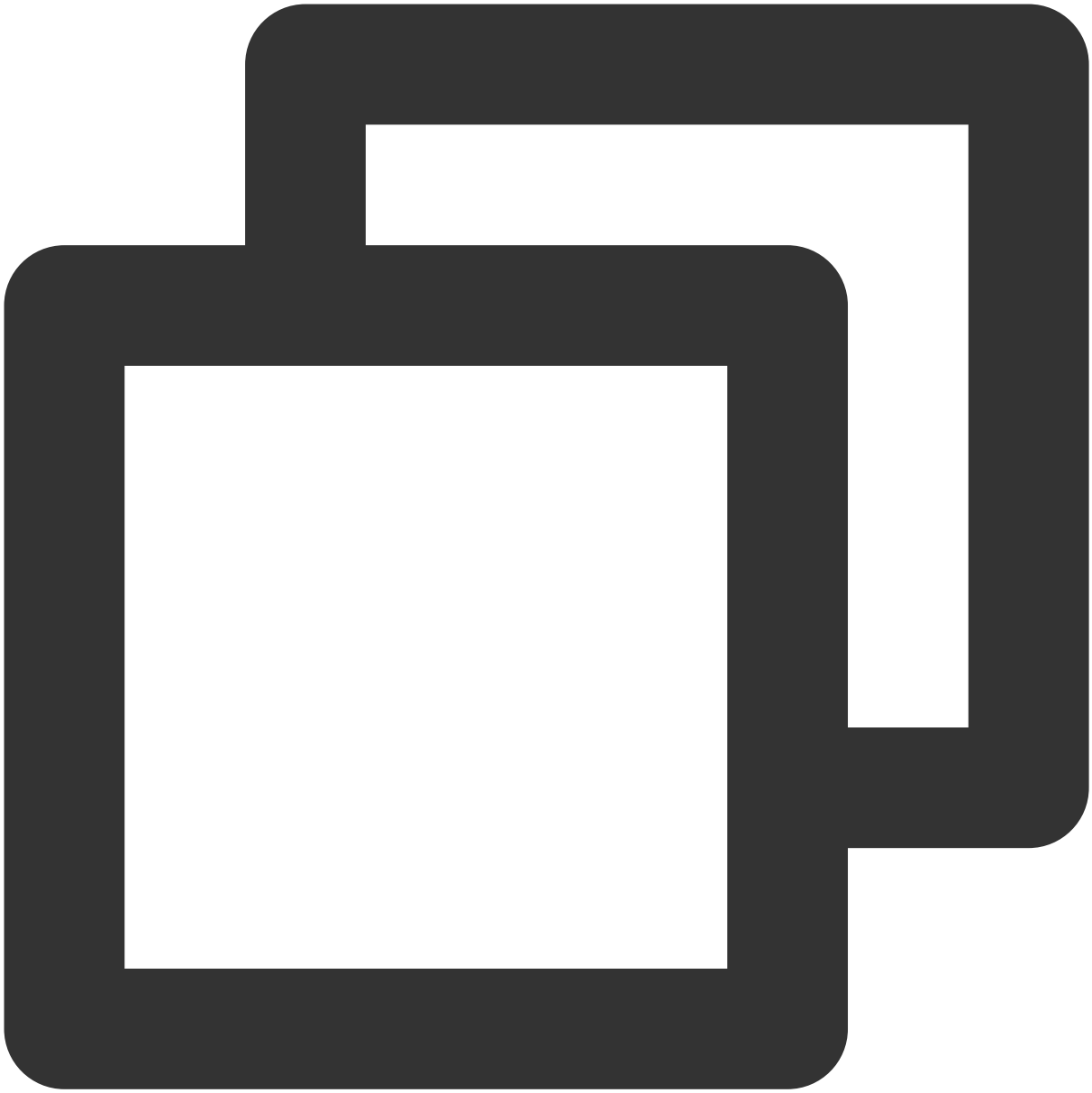
添加后如下图：



4. 混淆配置

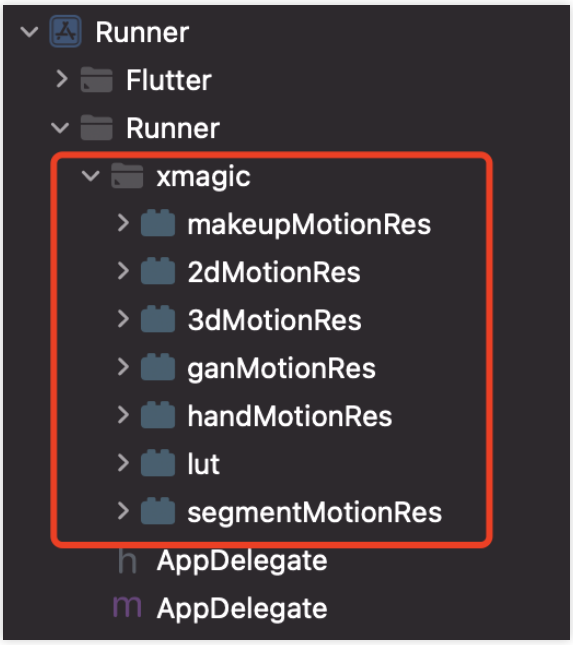
如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。

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.** { *;}
```

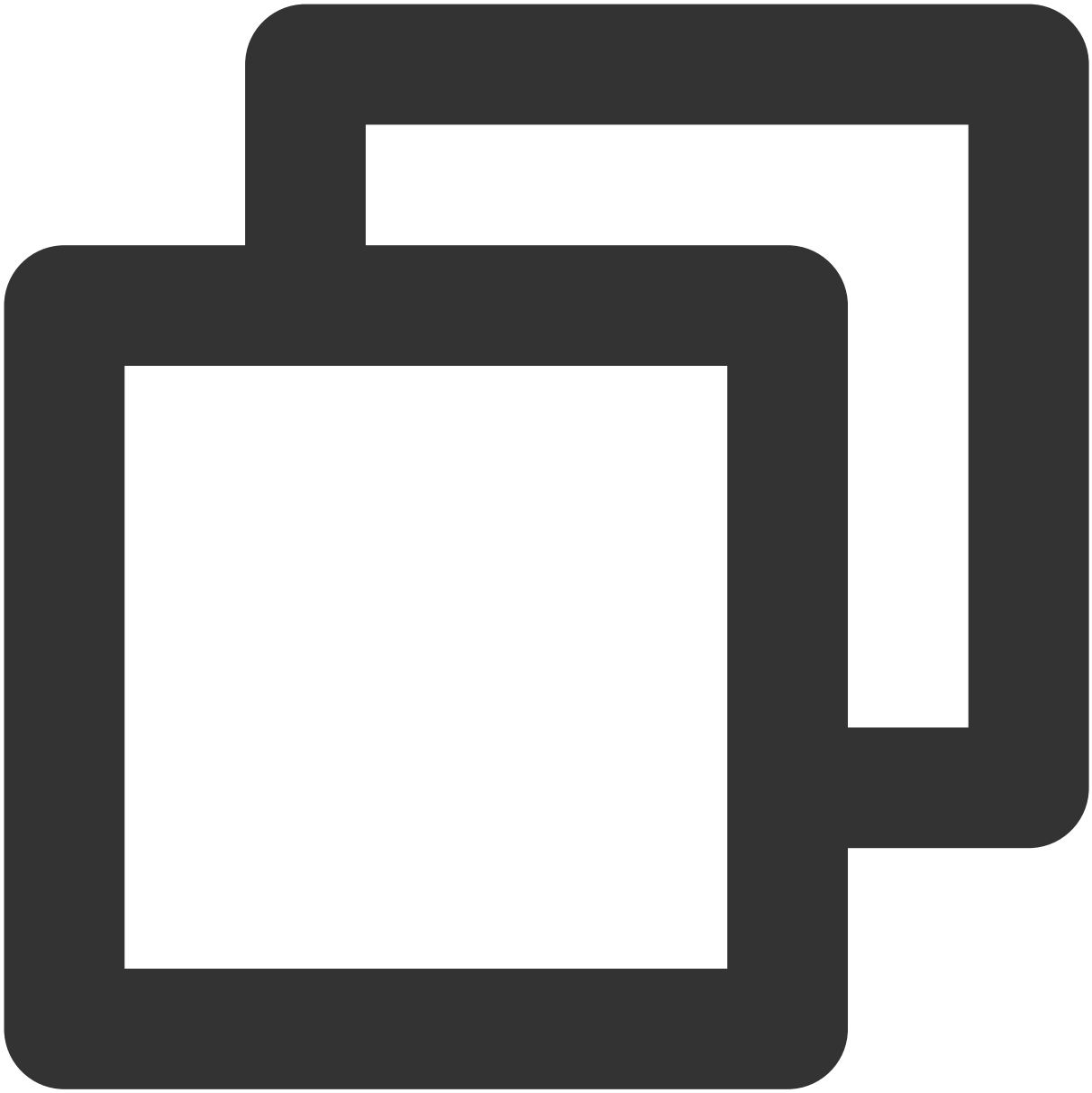
1. 添加美颜资源到您的工程。添加后如下图（您的资源种类跟下图不完全一致）：



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。

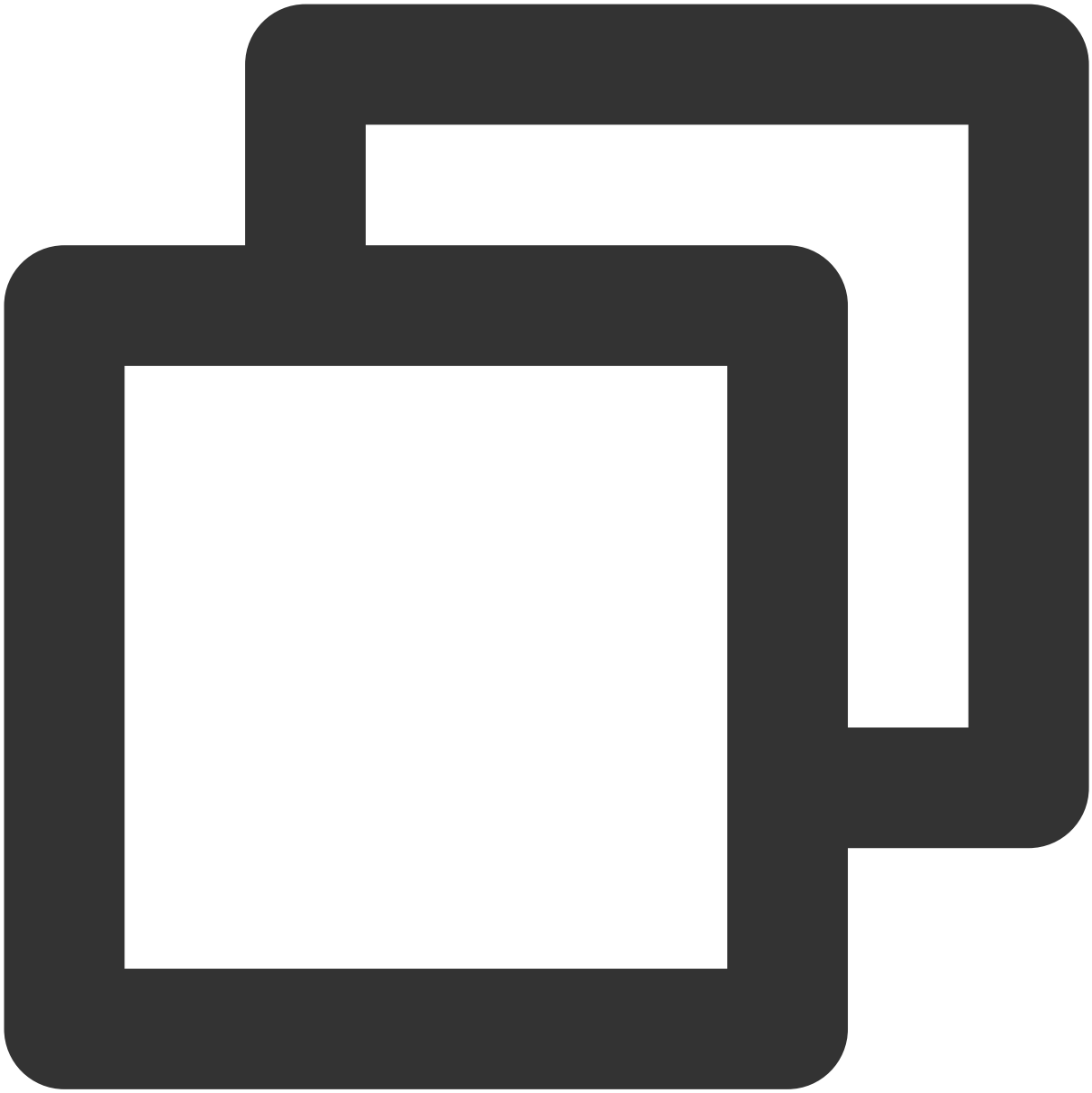
步骤2：引用 Flutter 版本 SDK

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。

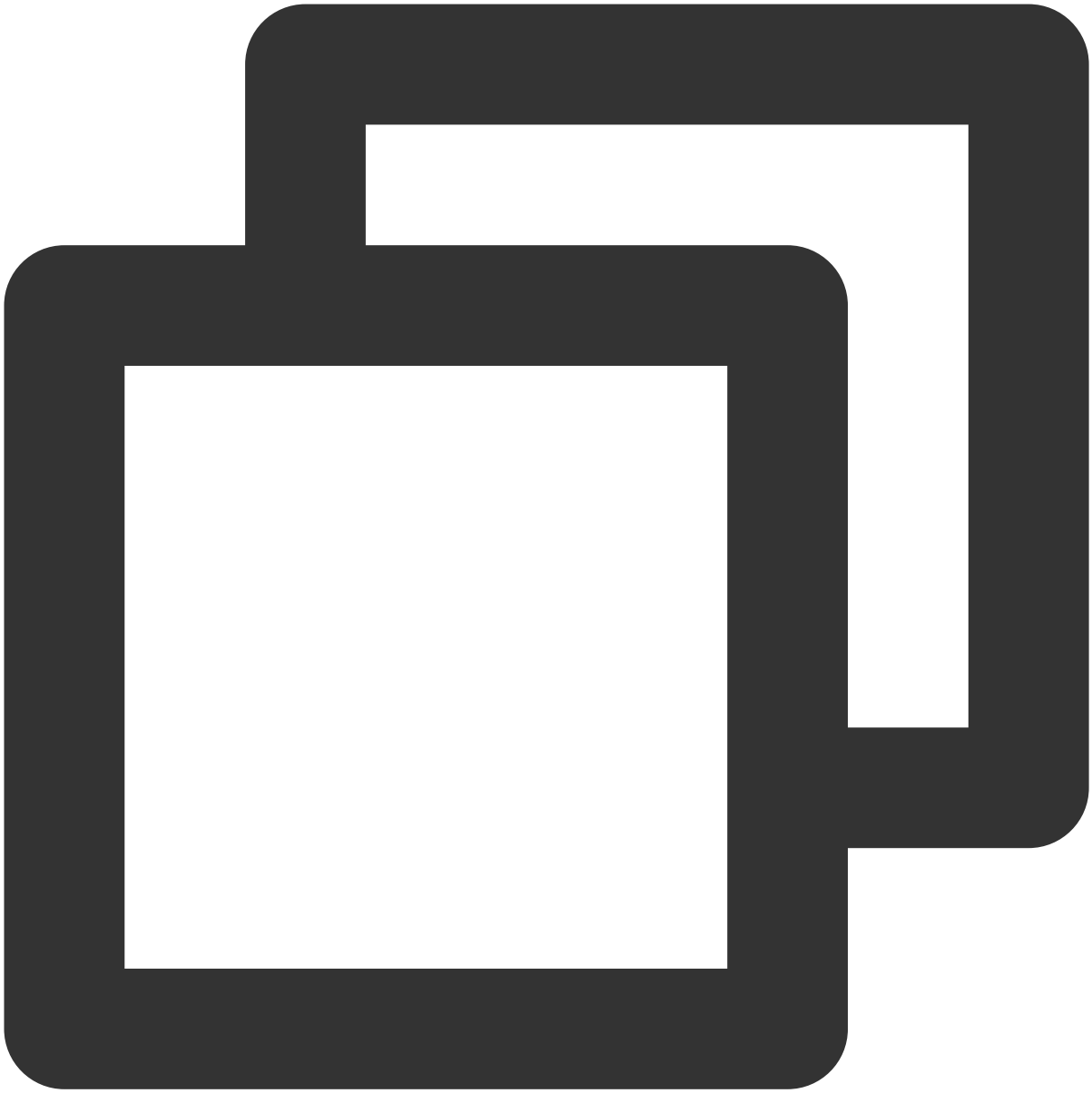
步骤4：在 Native 的 TRTC 自定义渲染逻辑中完成美颜处理

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);
```

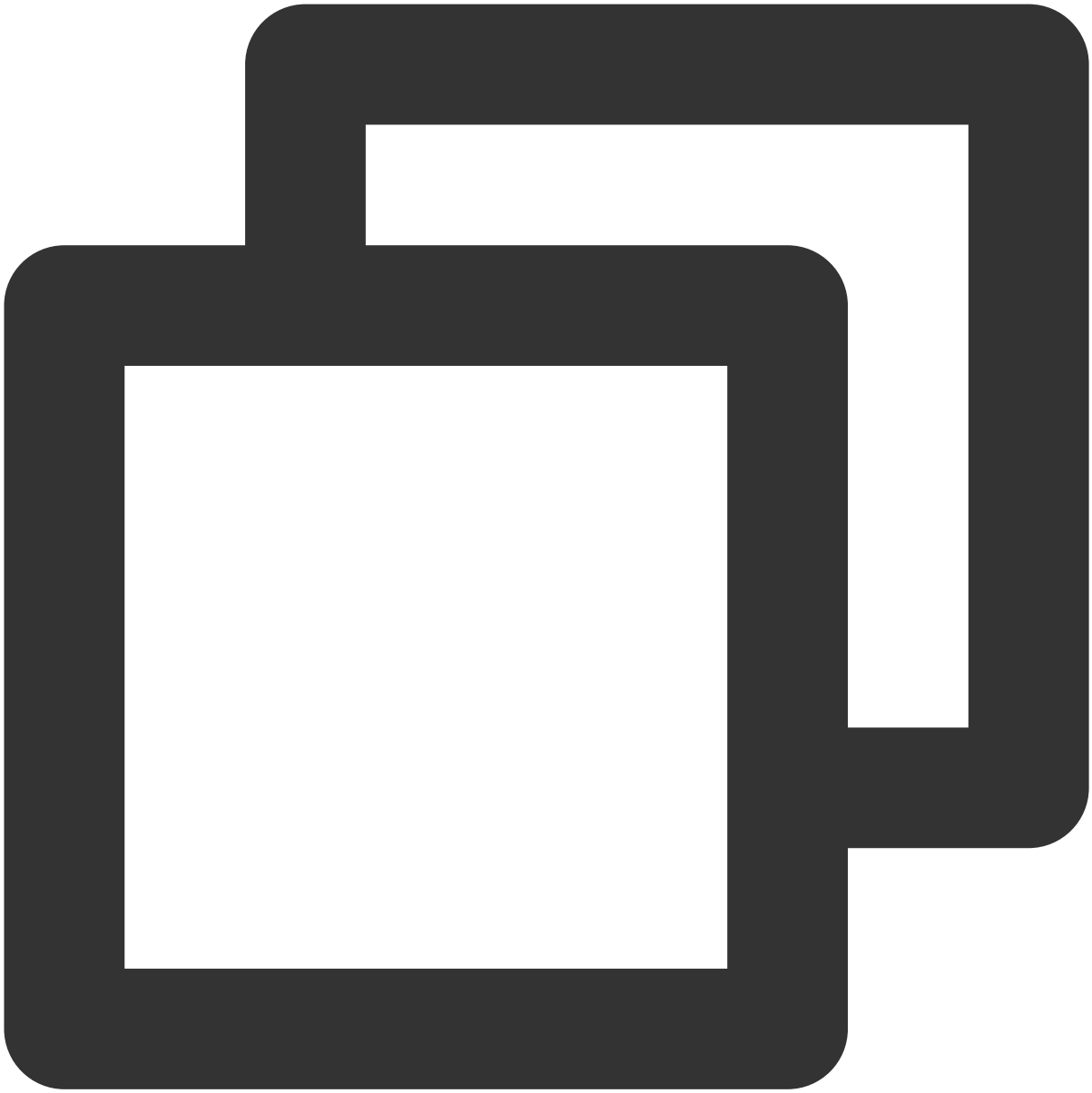
```
}

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：2024-05-24 18:35:12

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 和 调用设置铃声接口，您也可以关闭应用铃声。

1、替换音频文件

如果您通过源码依赖 TUICallKit 组件，您可以替换 tuicallkit-kt/src/main/res/raw 文件夹下的三个音频文件来替换铃声。

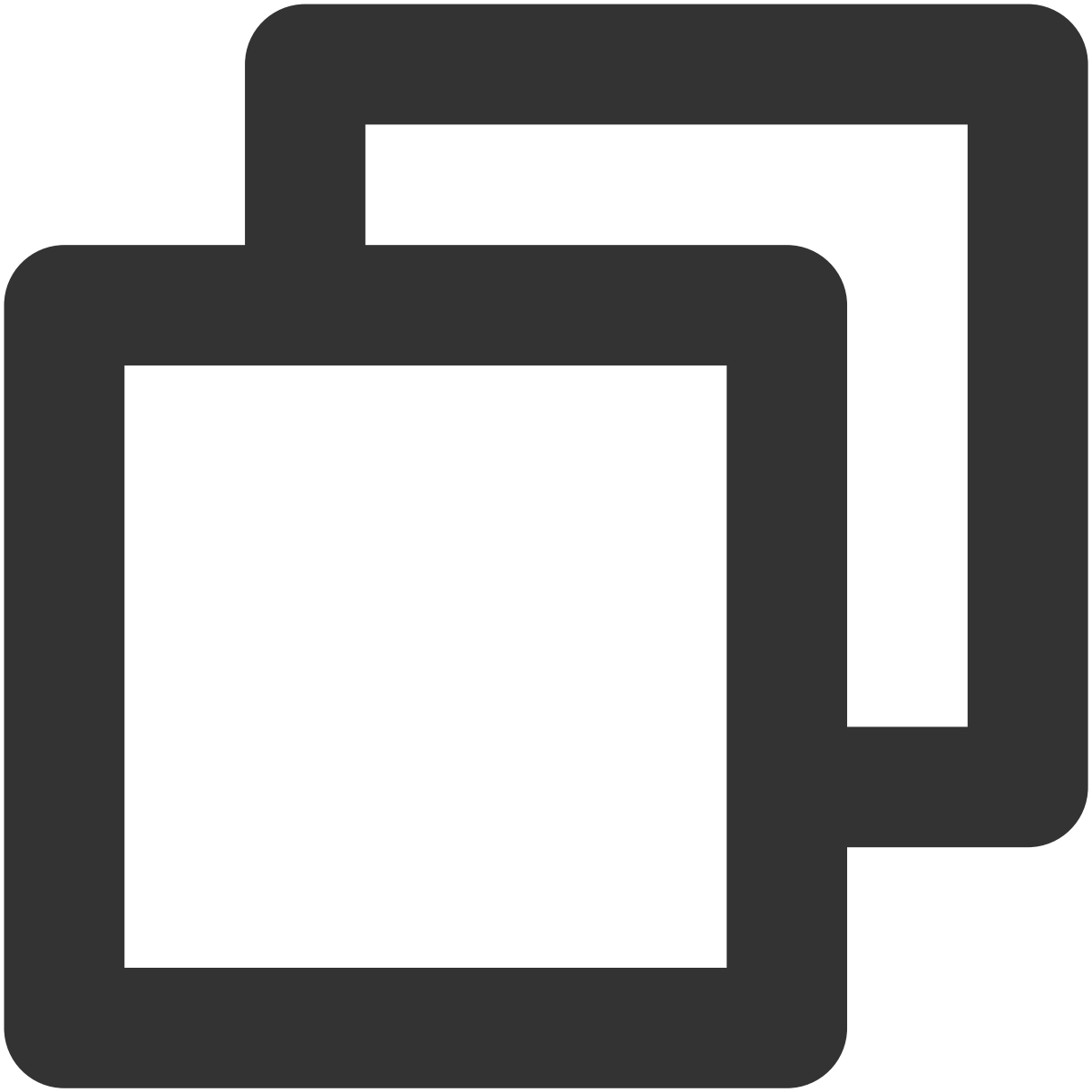
文件名	用途
phone_dialing.mp3	发起呼叫时的铃音
phone_hangup.mp3	被挂断的铃音
phone_ringing.mp3	接到呼叫时的铃音

2、设置铃声接口

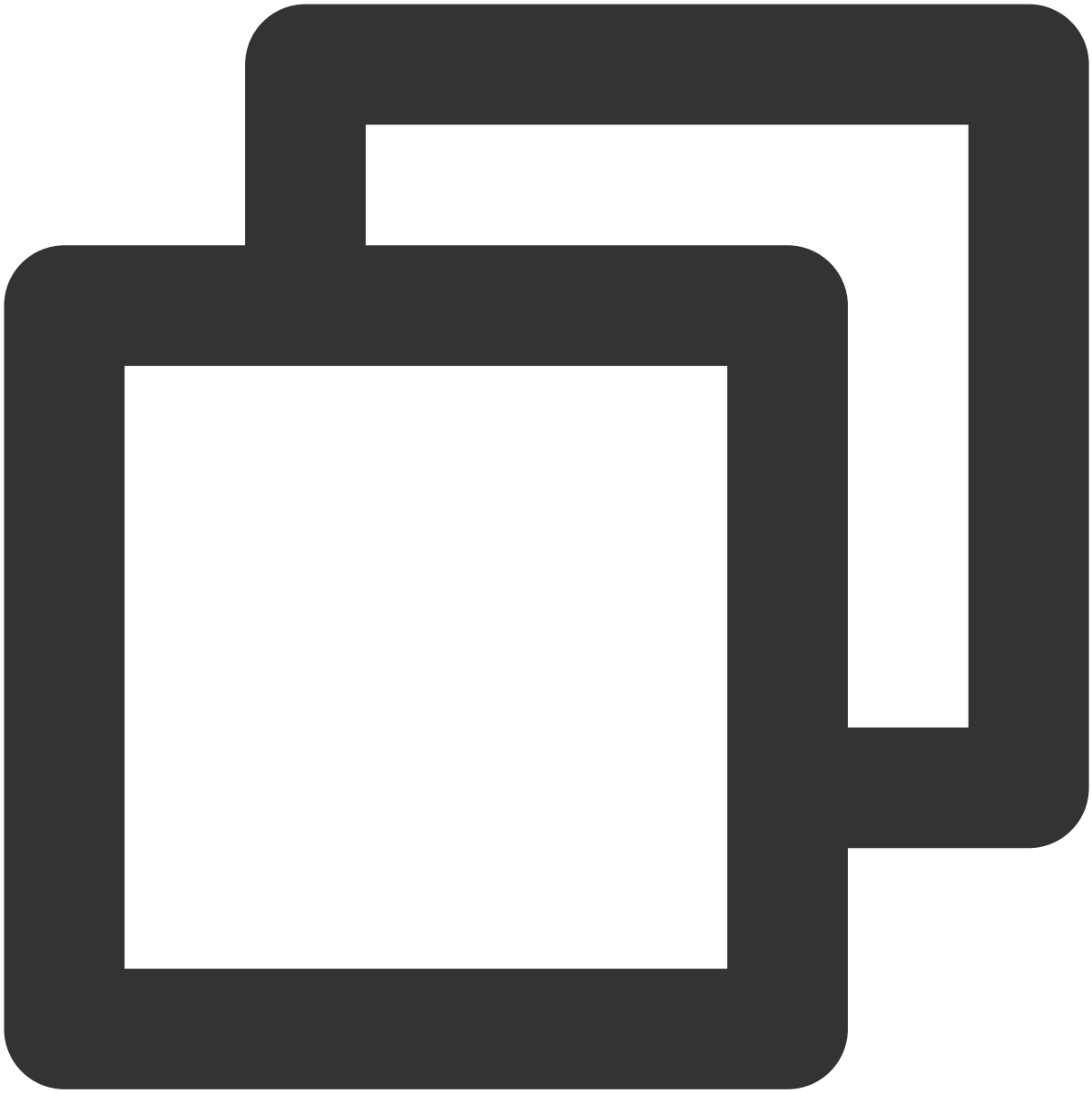
您也可以通过 setCallingBell 接口设置来电铃声。

Kotlin

Java



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



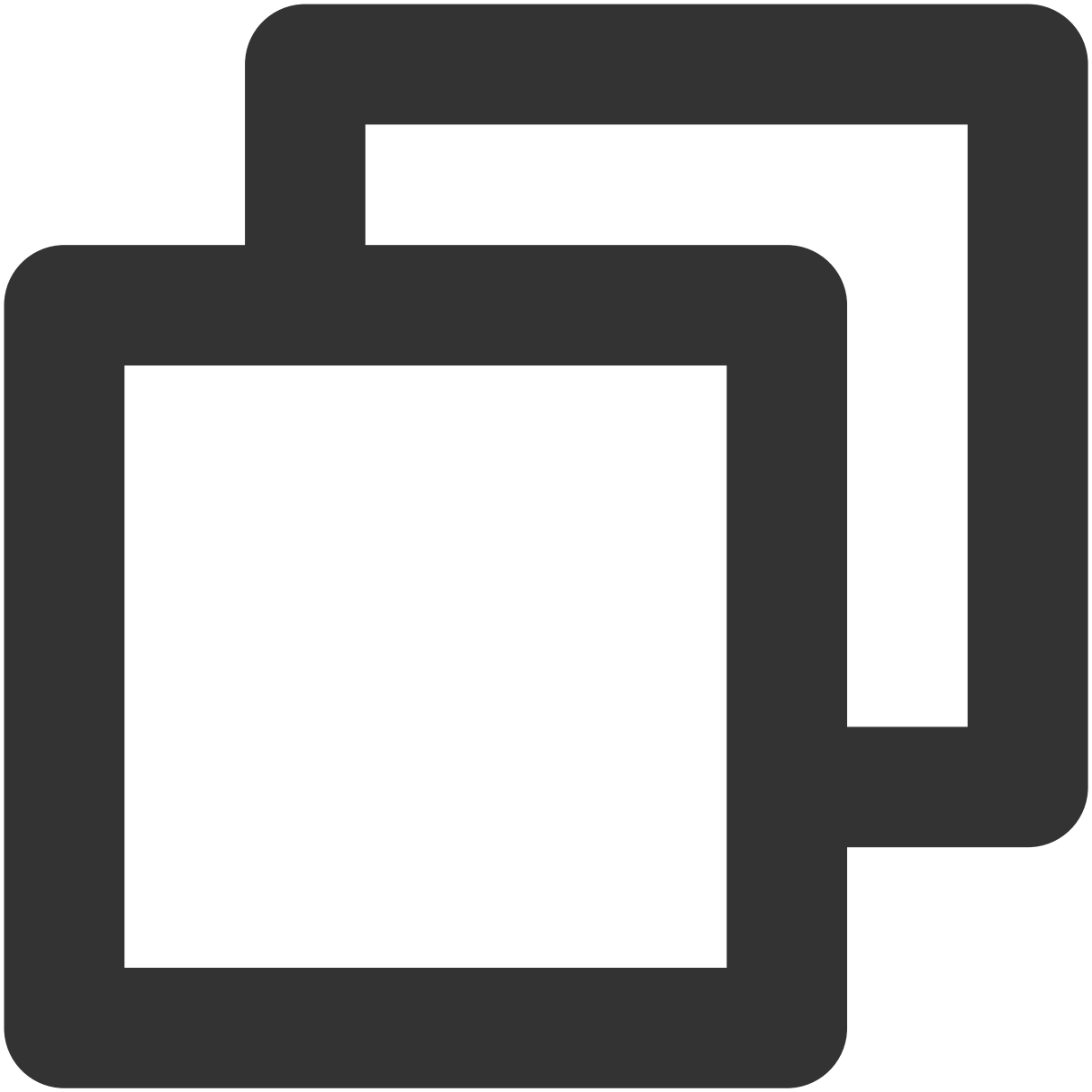
```
TUICallKit.createInstance(context).setCallingBell(filePath);
```

3. 设置静音模式

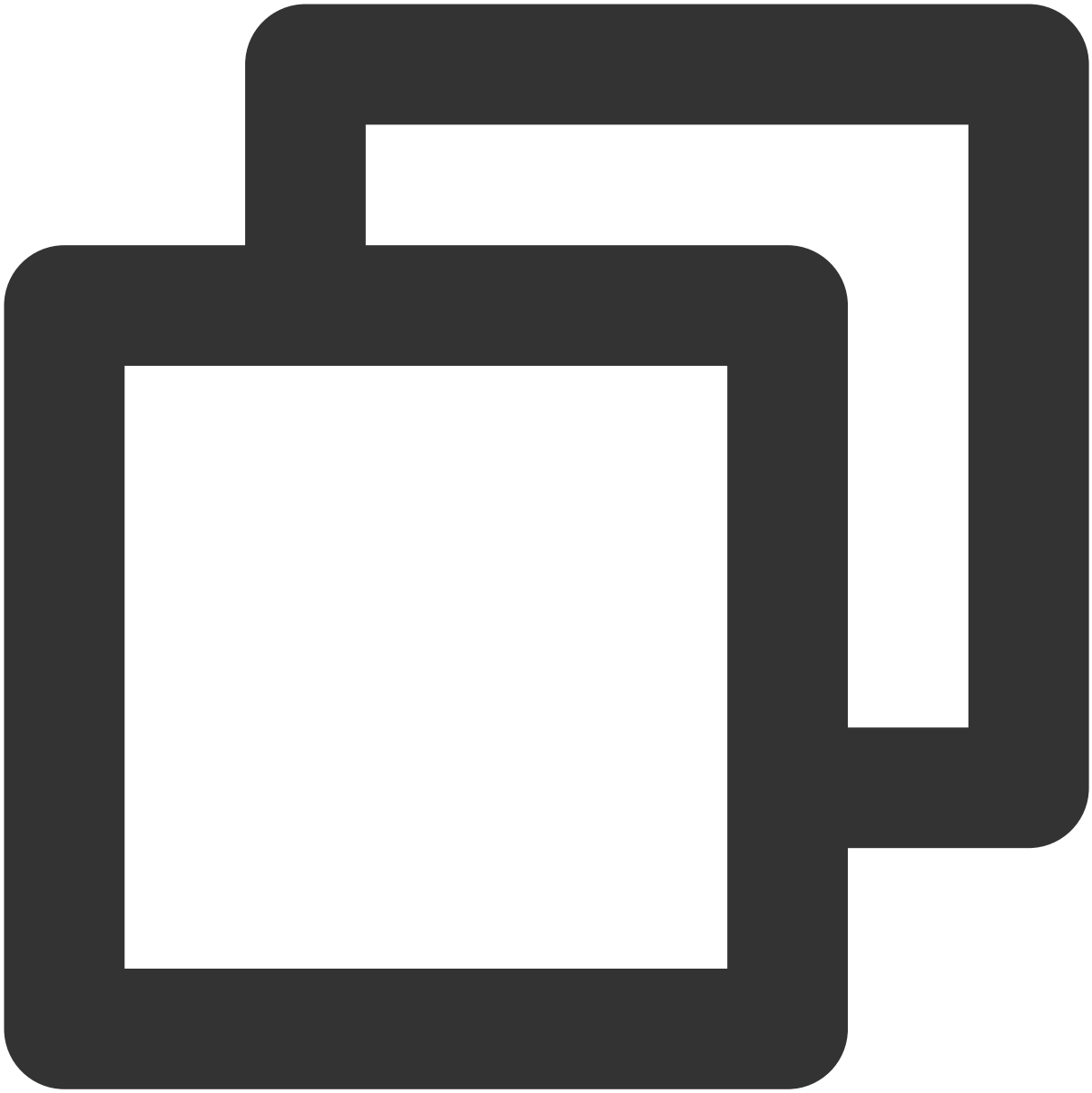
如果您不需要响铃，您可以通过 enableMuteMode 设置静音模式。

Kotlin

Java



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



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

设置离线推送铃音

离线推送铃声设置请参考见：FCM 离线推送自定义来电铃声。

iOS

最近更新时间：2024-05-24 18:35:12

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：

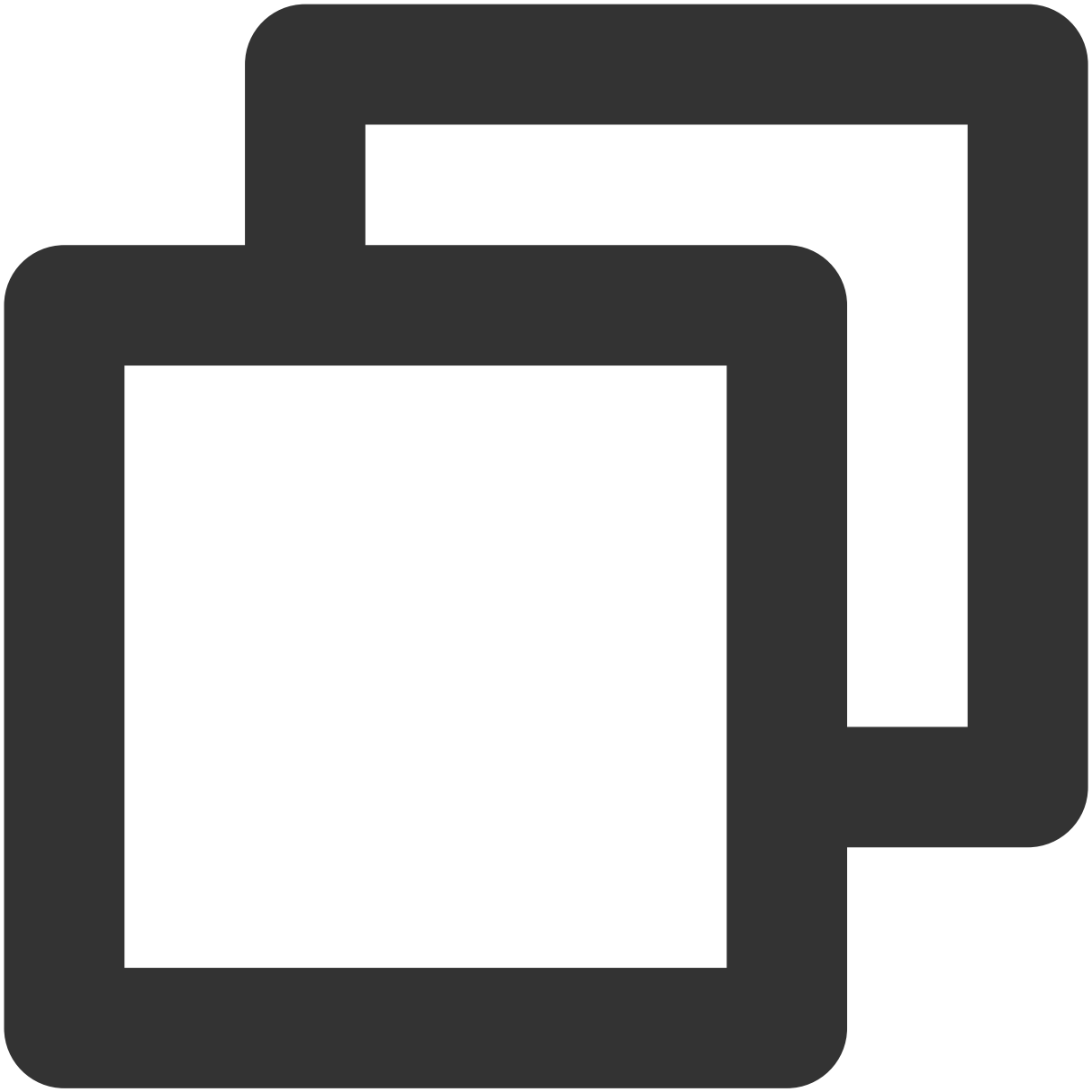
文件名	用途
phone_dialing.mp3	发起呼叫时的铃音
phone_hangup.mp3	被挂断的铃音
phone_ringing.mp3	接到呼叫时的铃音

2、设置铃声接口

您也可以通过 setCallingBell 接口设置来电铃声。

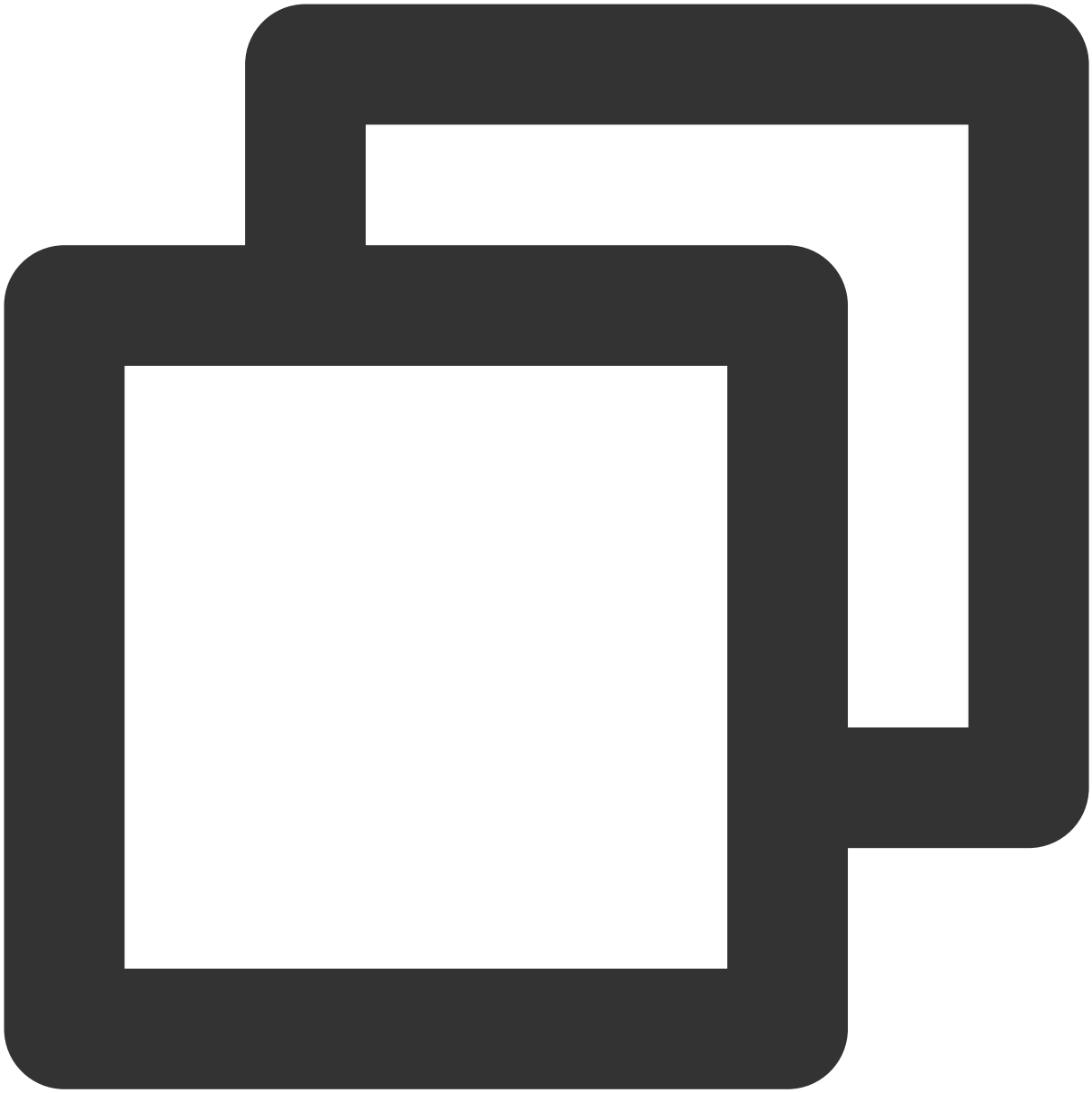
Swift

Objective-C



```
import TUICallKit_Swift

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

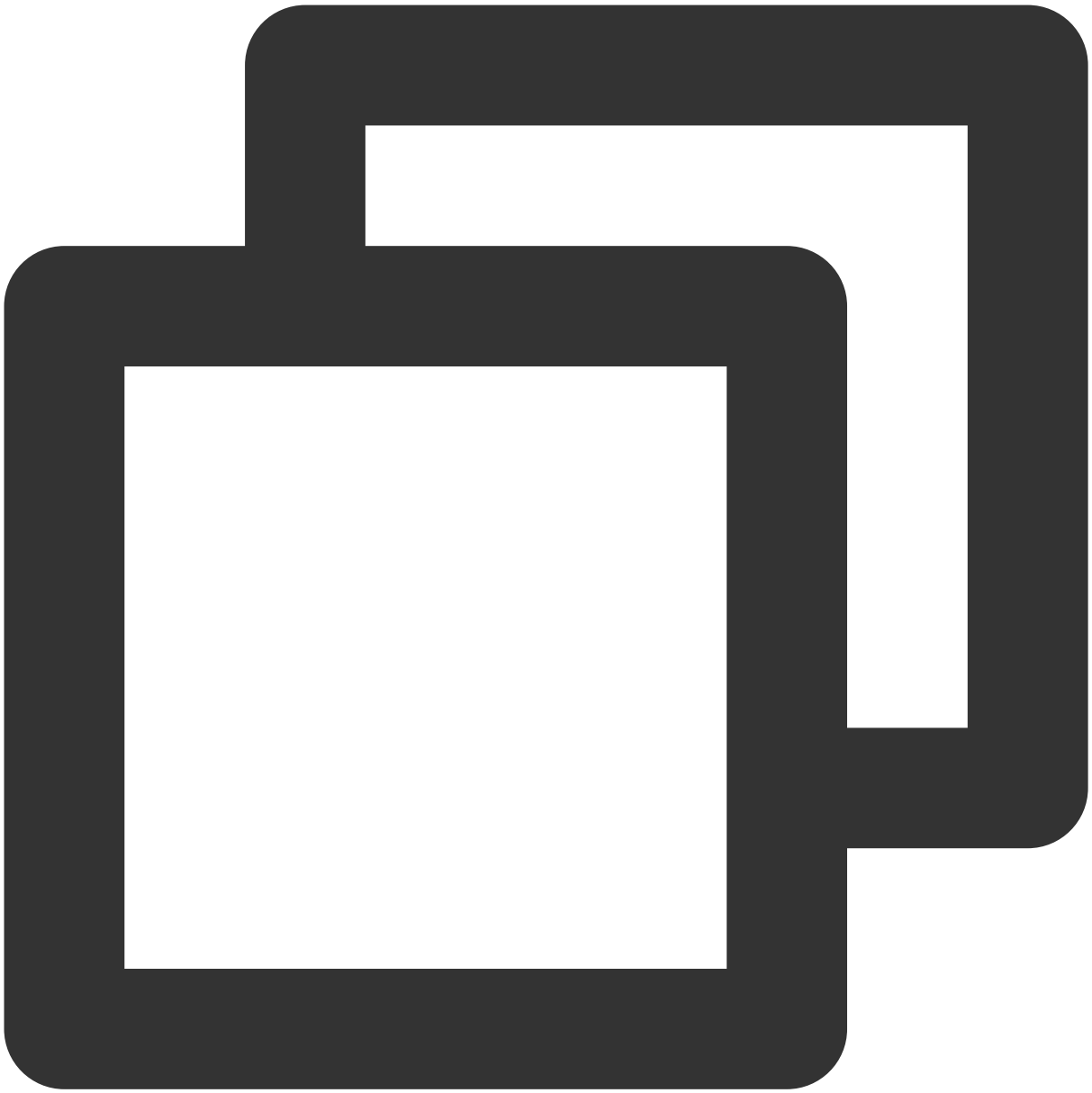
[[TUICallKit createInstance] setCallingBellWithFilePath:@""];
```

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。

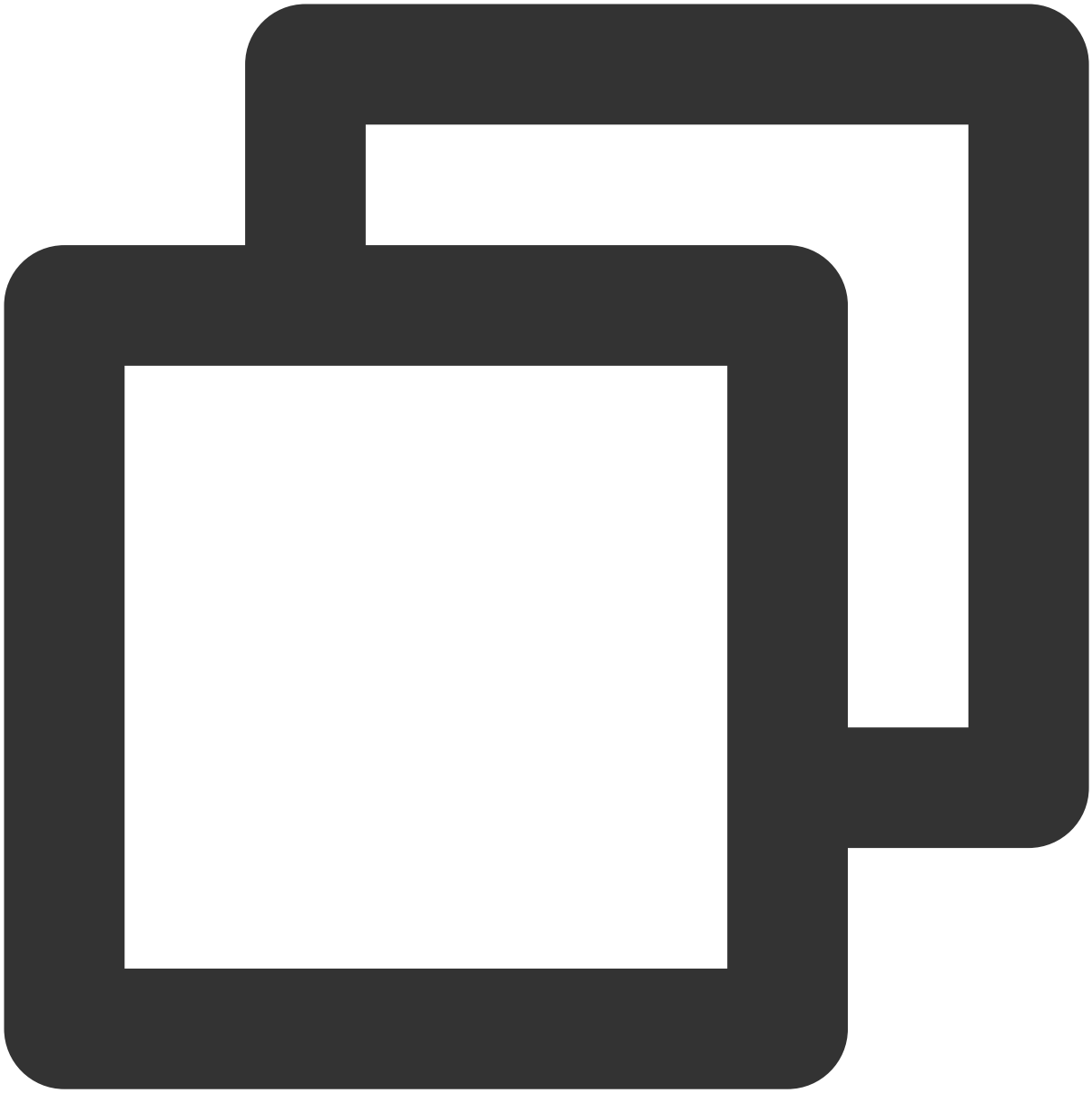
Swift

Objective-C



```
import TUICallKit_Swift

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



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>

[[TUICallKit createInstance] enableMuteModeWithEnable: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。

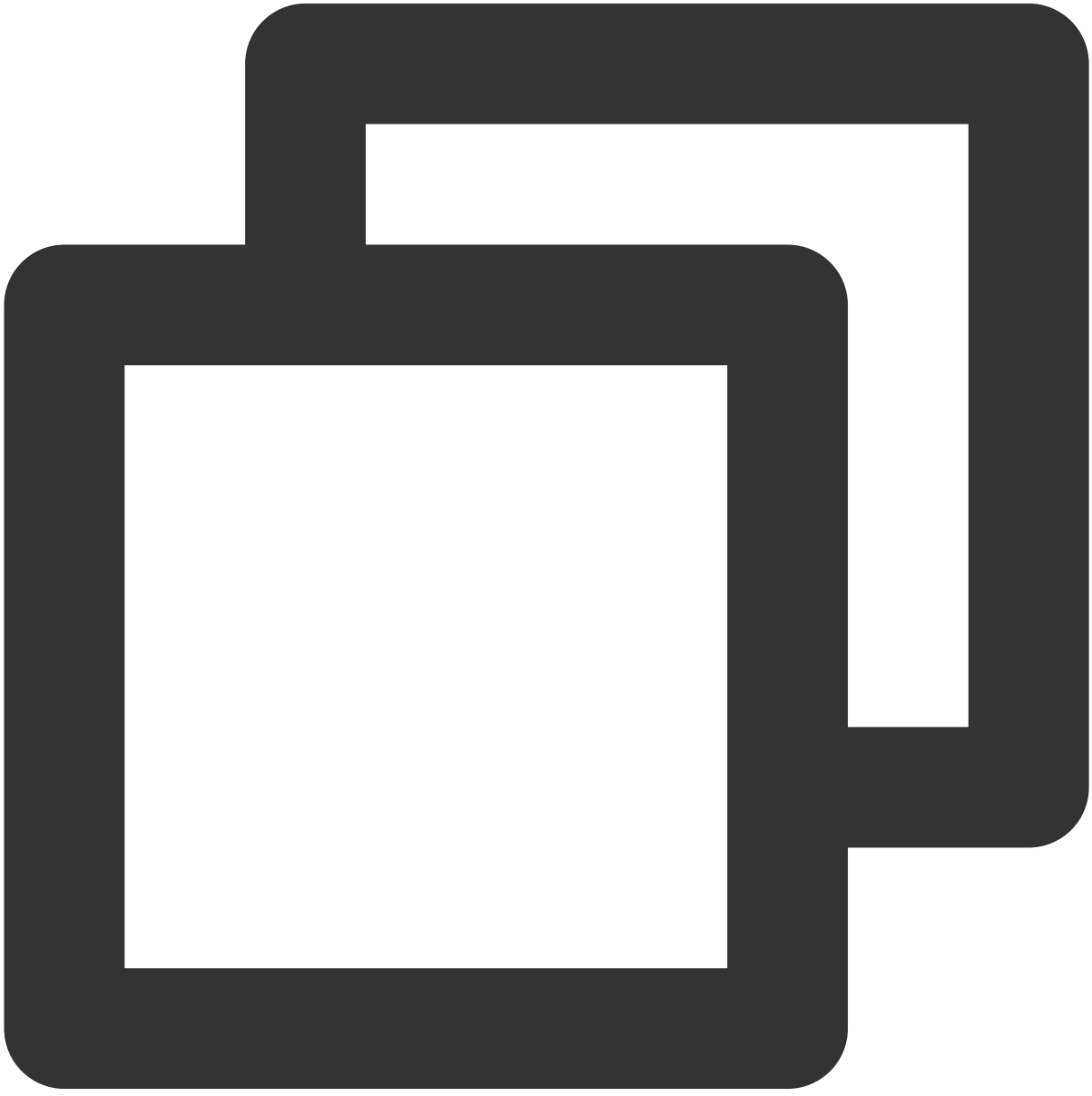
注意：

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。

Swift

Objective-C



```
import TUICallKit_Swift
import TUICallEngine
```

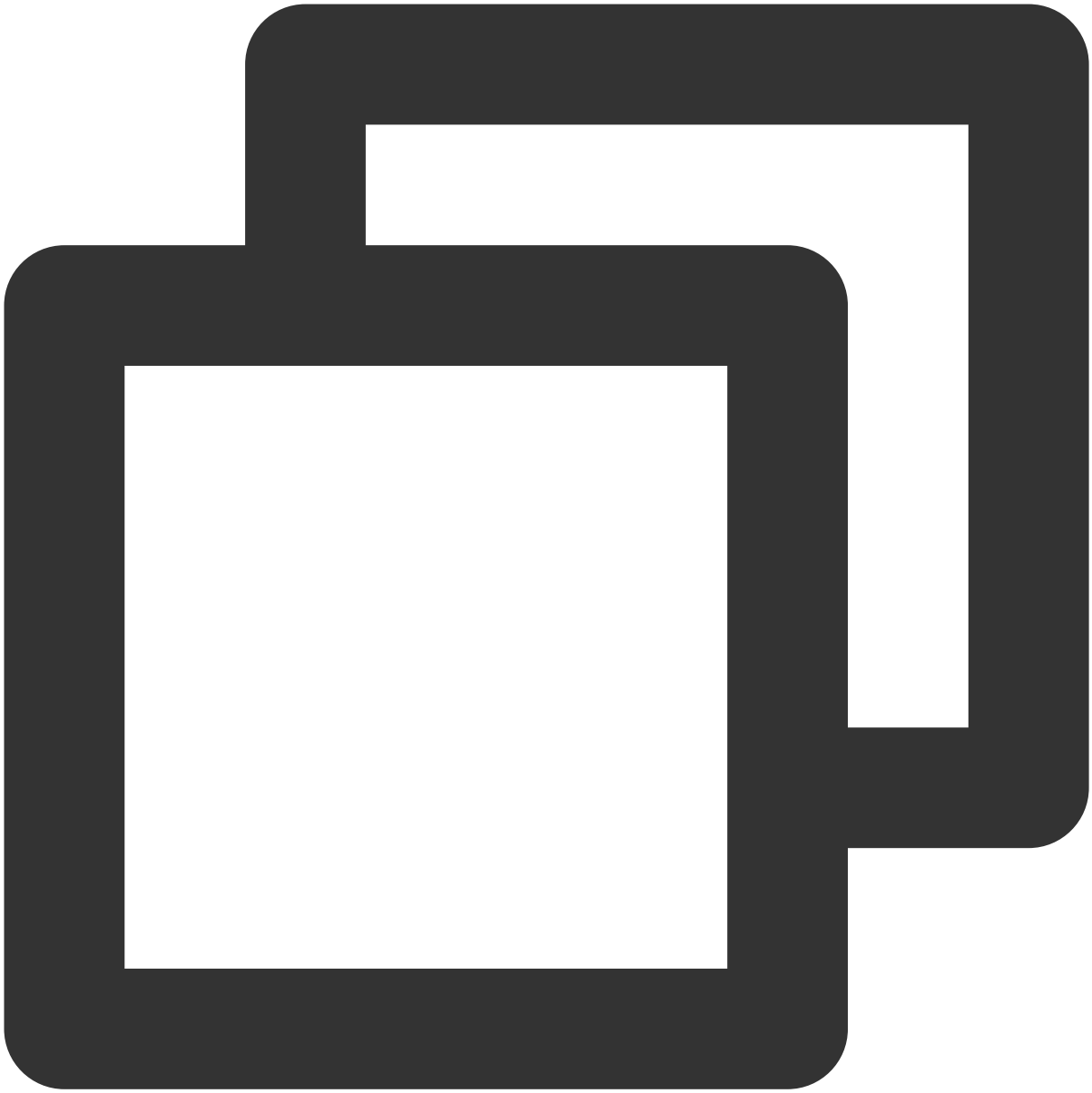
```
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "您有一个新的通话"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// 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"

let params = TUICallParams()
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo

TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: p

} fail: { code, message in

}
```



```
#import <TUICallKit_Swift/TUICallKit_Swift-Swift.h>
#import <TUICallEngine/TUICallEngine.h>

- (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";
    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;
}

[[TUICallKit createInstance] callWithUserId:@"123456"
                                callMediaType:TUICallMediaTypeAudio
                                       params:[self getCallParams] succ:^{

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

}];
```

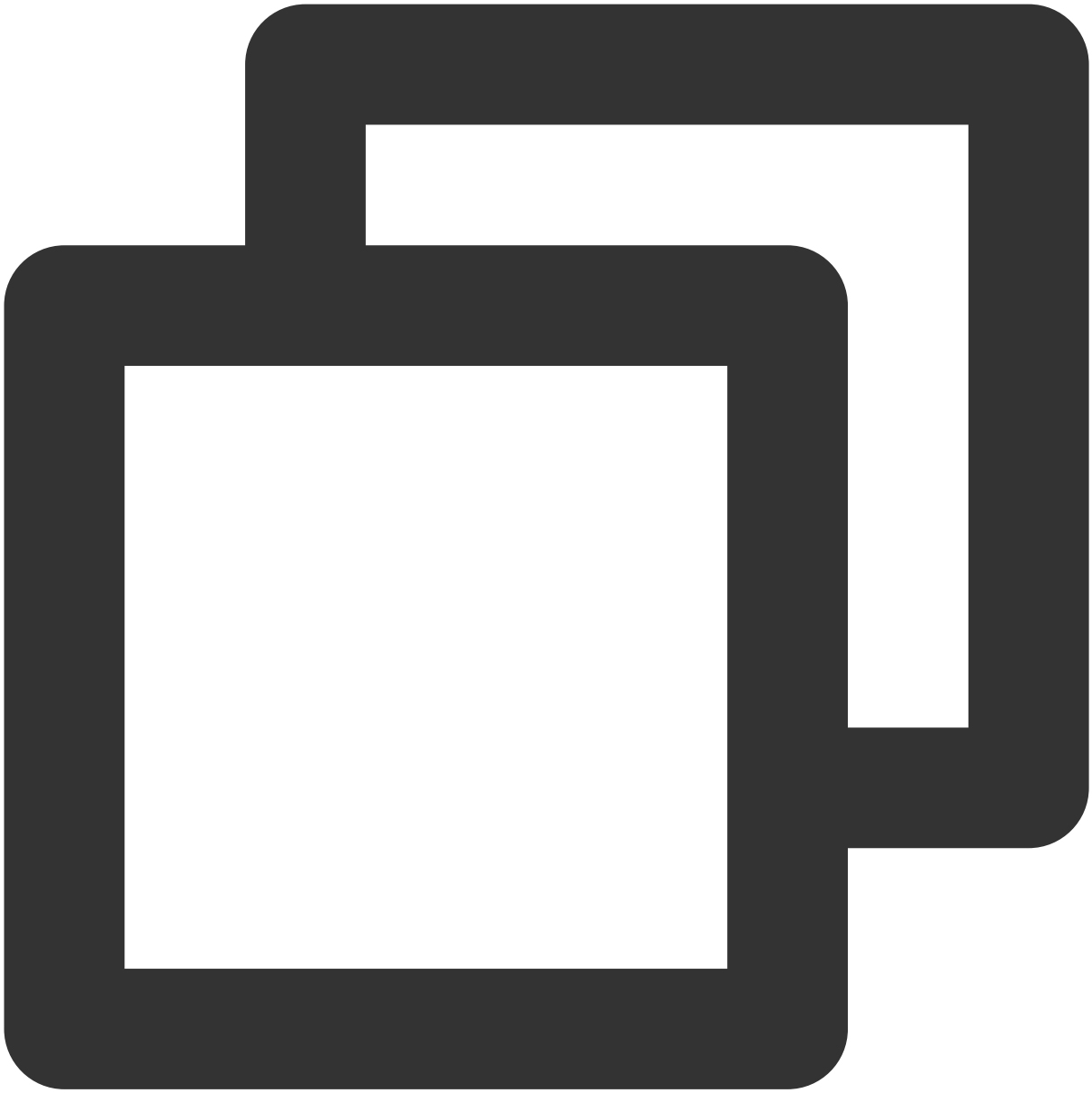
Web&H5

最近更新时间：2024-05-24 18:20:06

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：
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}`);
}
```

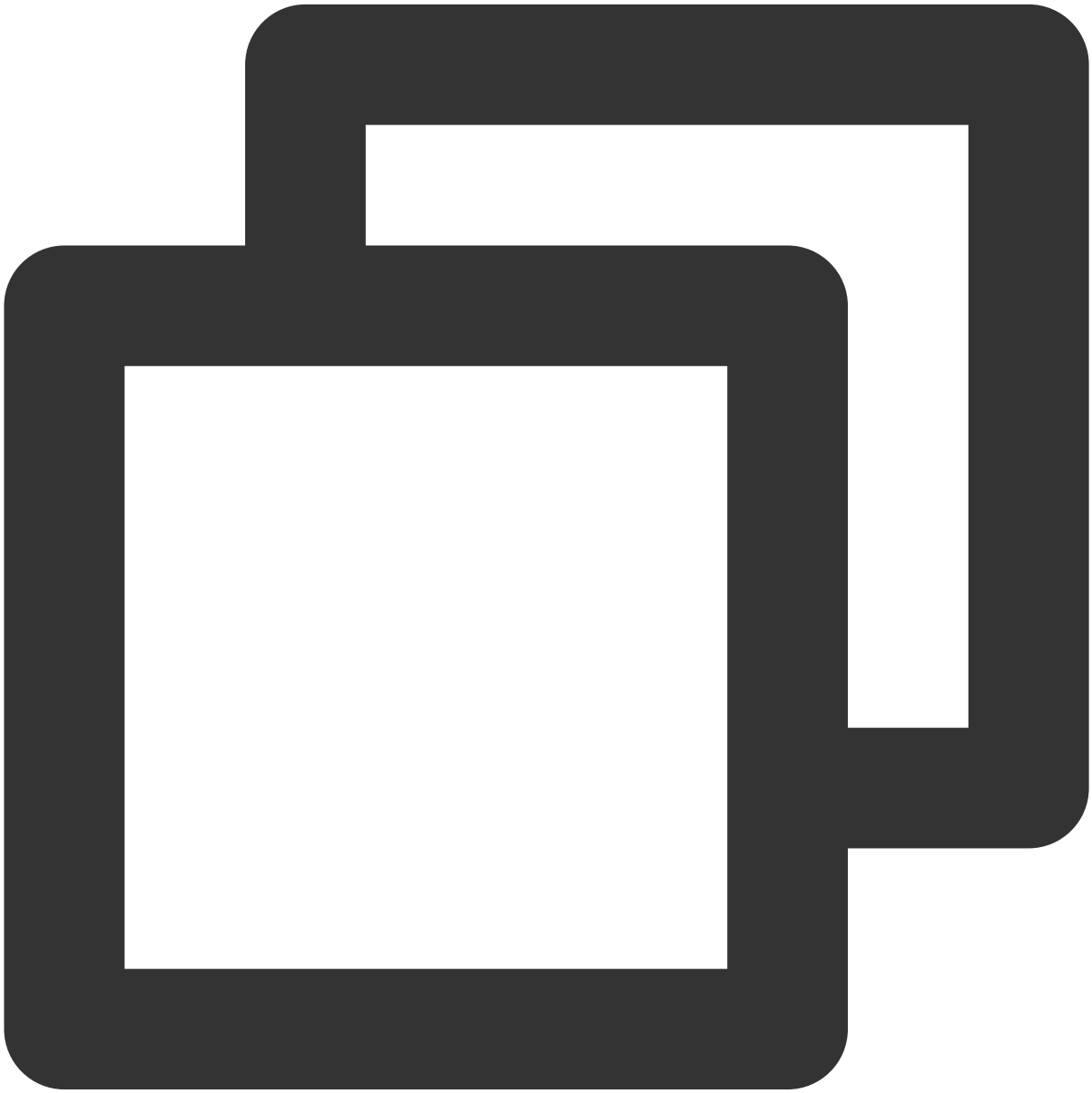
静音来电铃声

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：

v3.1.2+ 支持。



```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean);
} catch (error: any) {
```

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

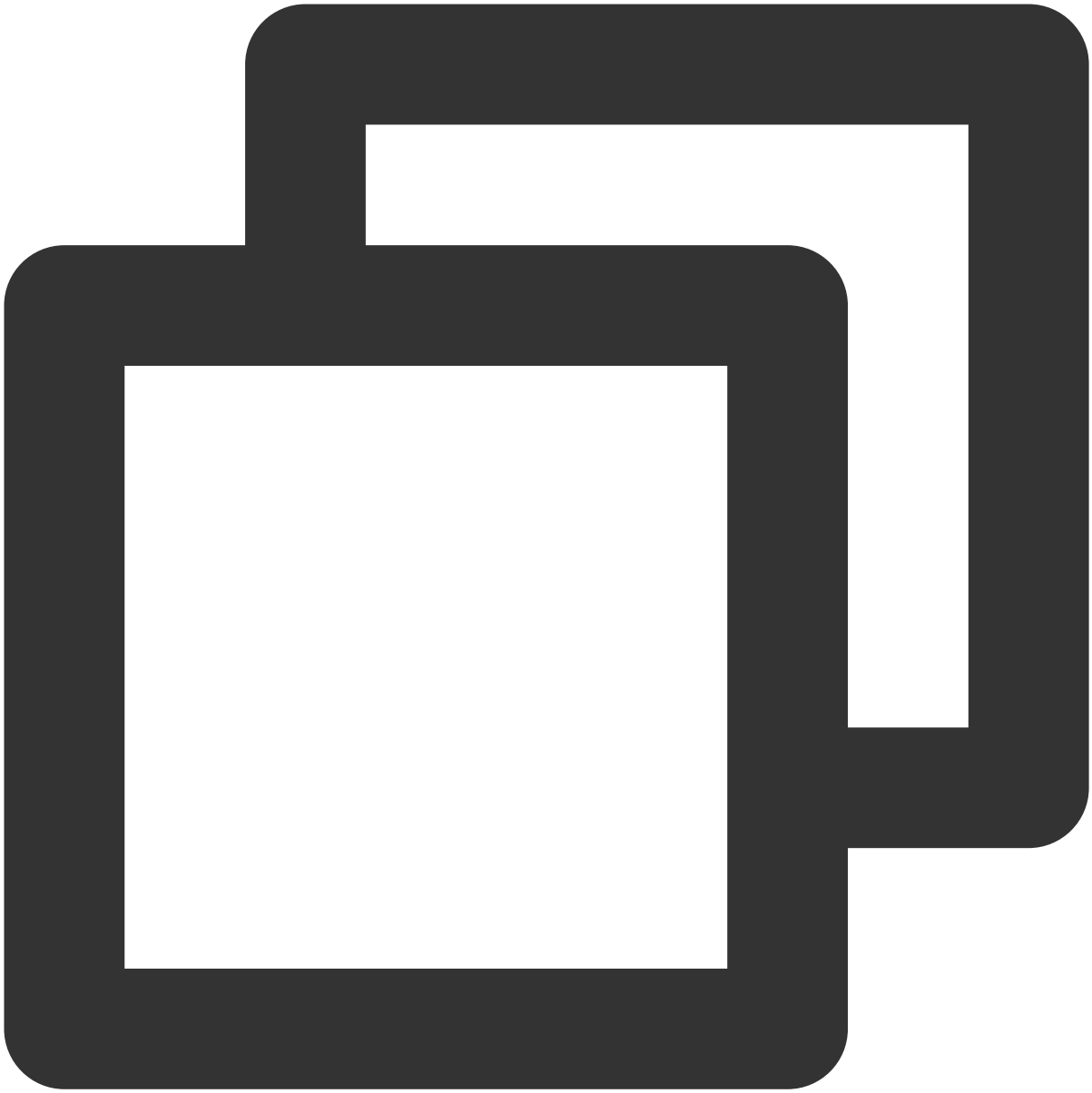
uni-app (Anroid&iOS)

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**。



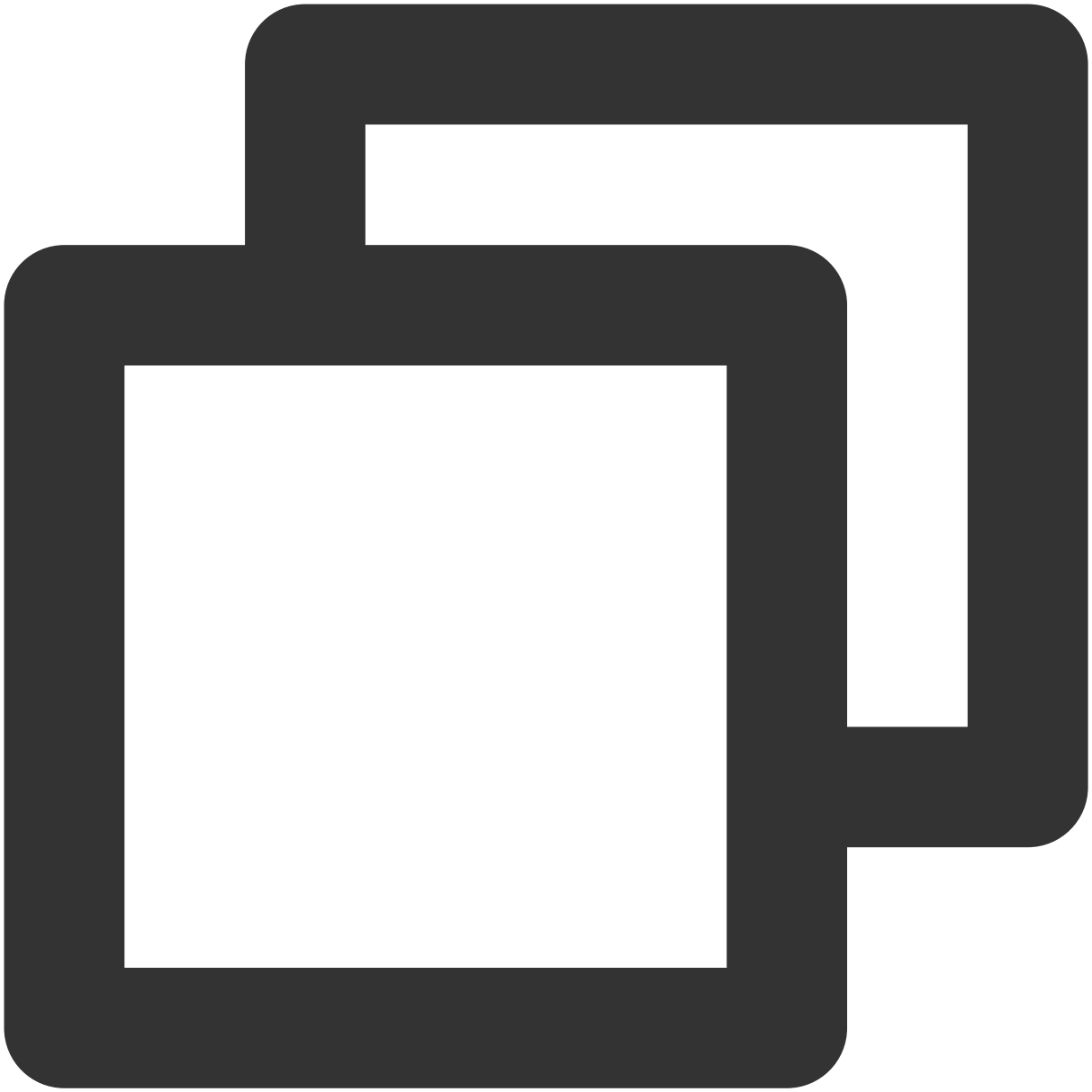
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const tempFilePath = './static/rain.mp3'; // Locally stored audio files
let musicFilePath = '';

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：2024-05-24 18:46:03

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。

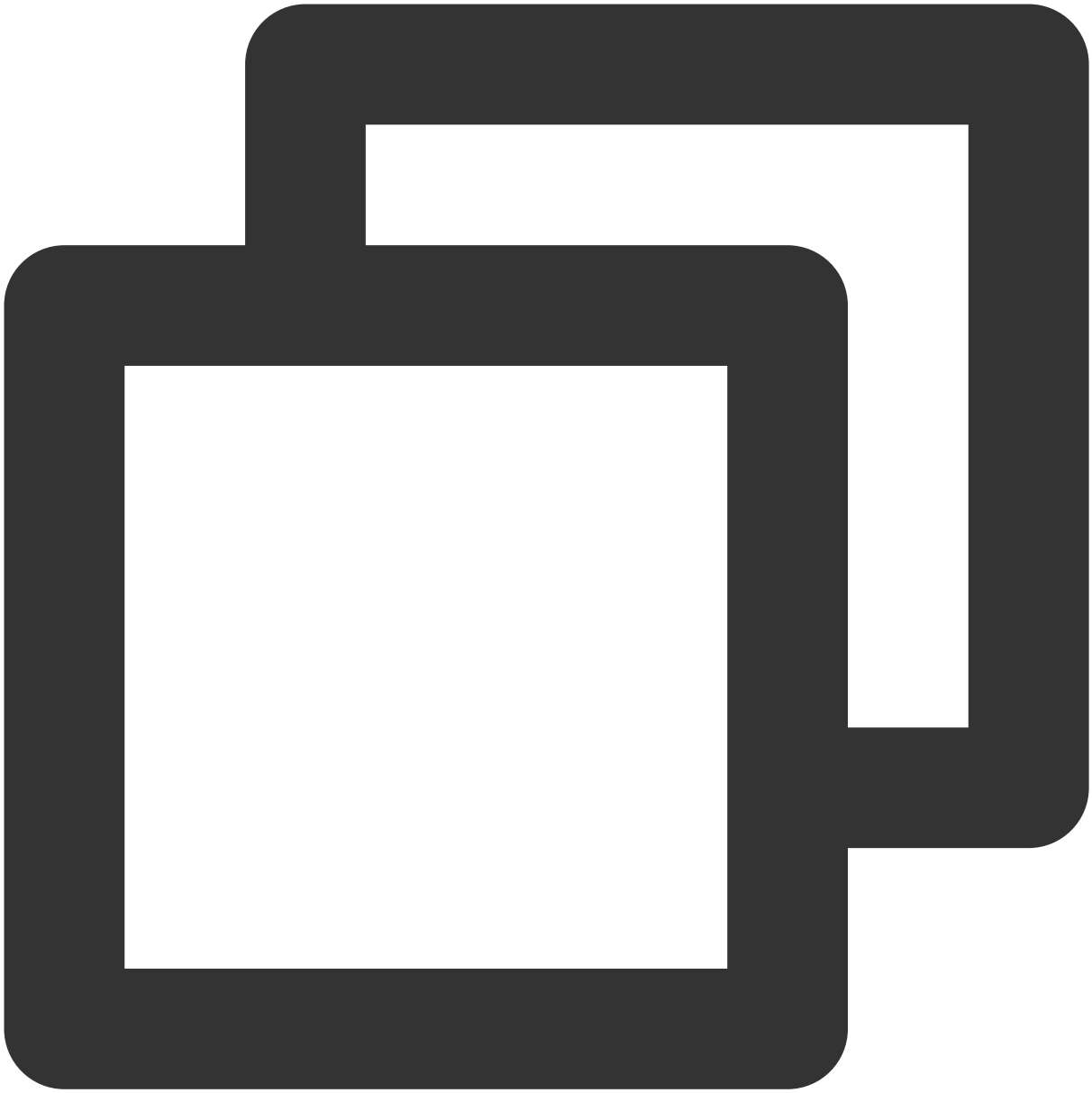
1、替换音频文件

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。

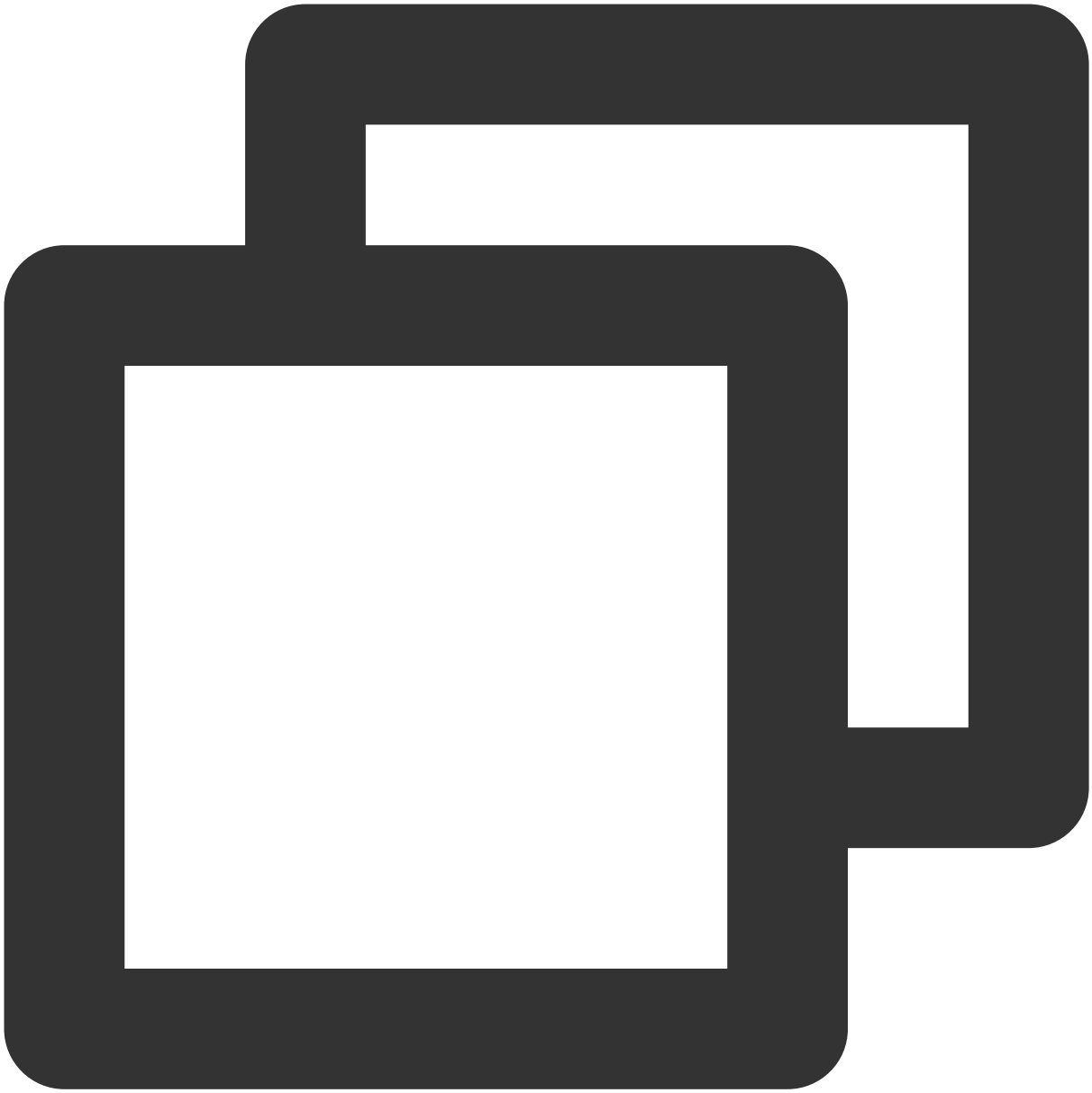


```
import 'package:tencent_calls_uikit/tencent_calls_uikit.dart';

void setCallingBell() {
    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。



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

void enableMuteMode() {
    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：2024-05-24 18:35:12

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）

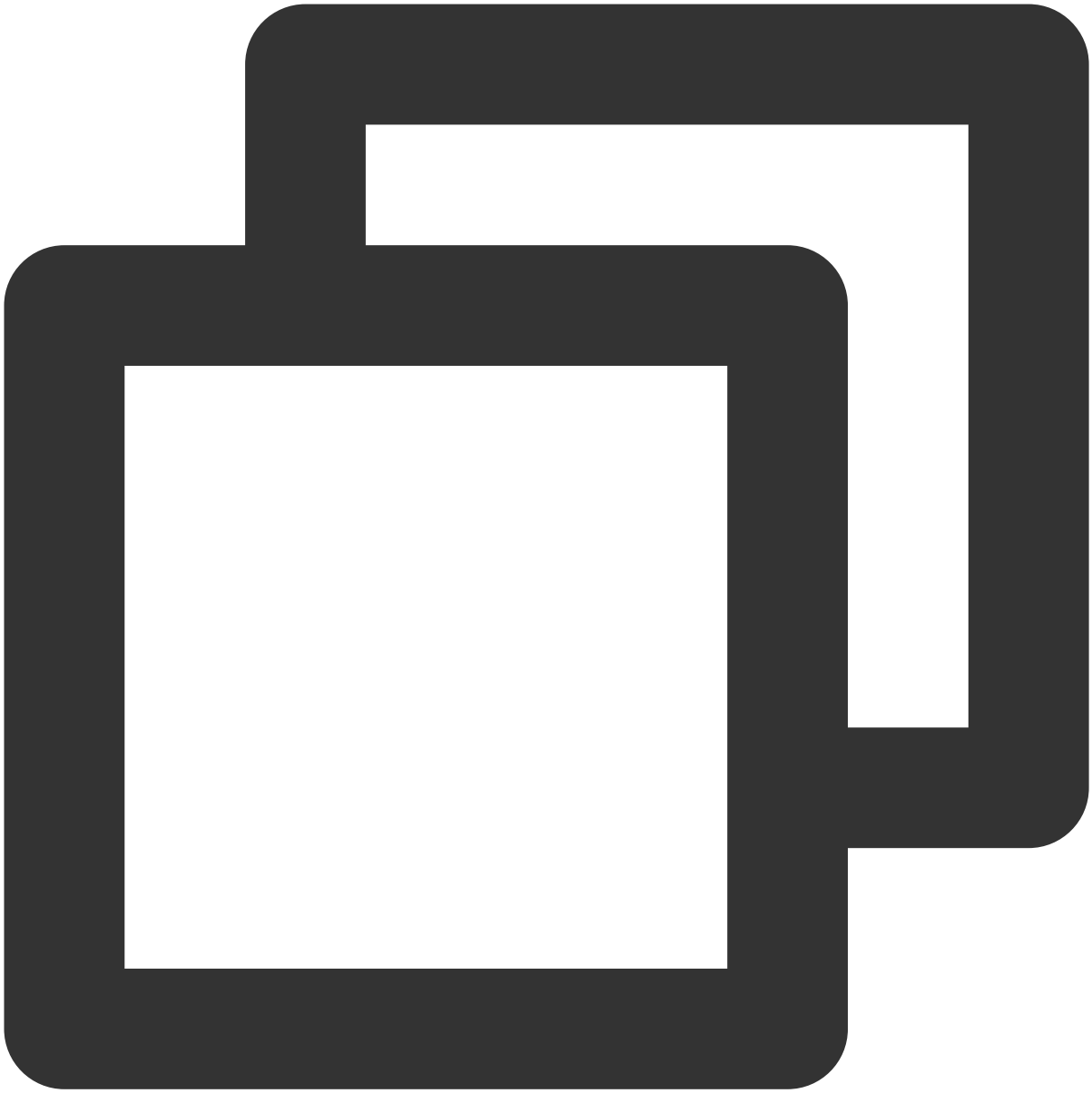
Android（Java）

iOS（Swift）

iOS（Objective-C）

Flutter（Dart）

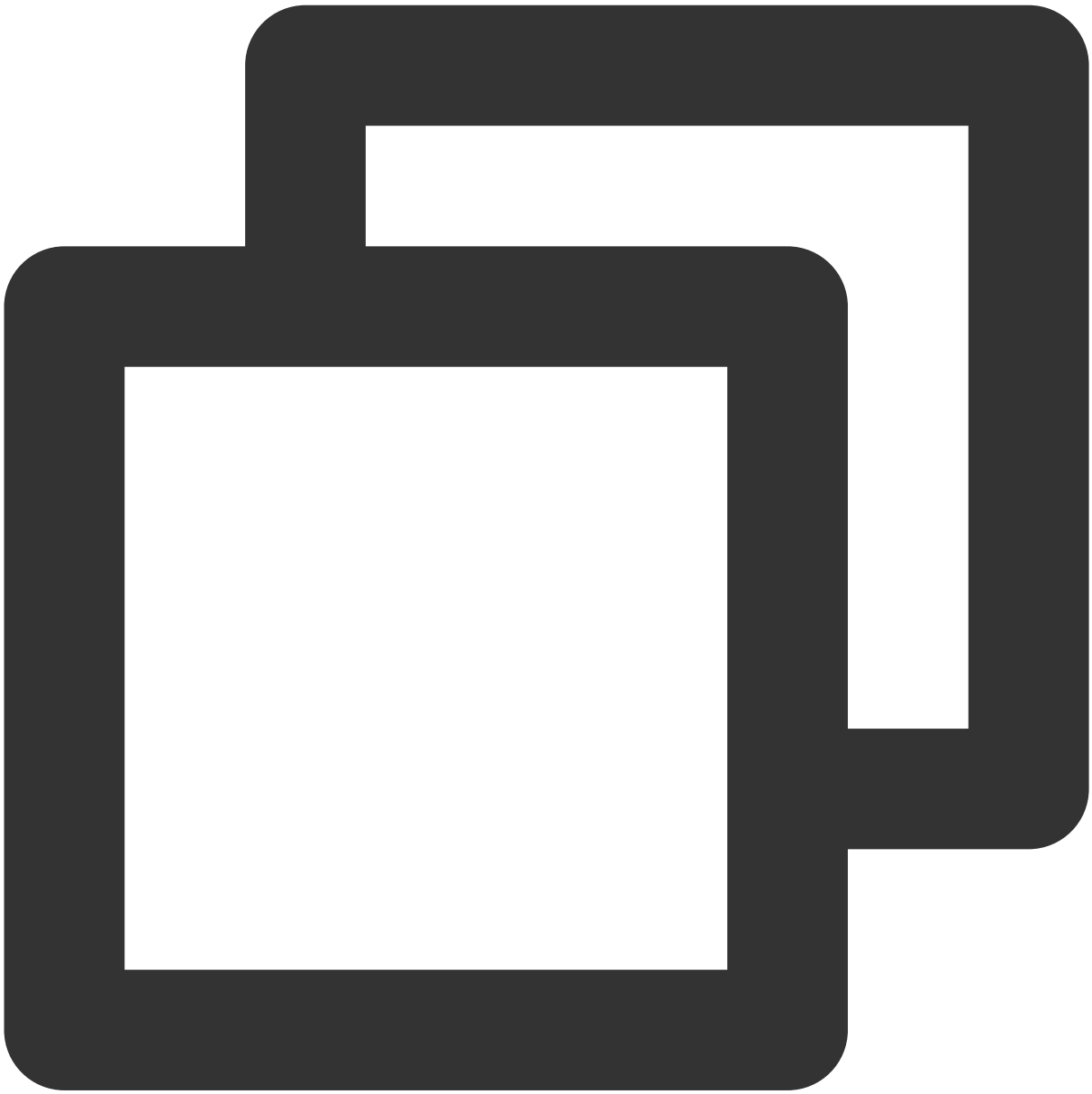
版权所有：腾讯云计算（北京）有限责任公司



```
import com.tencent.qcloud.tuikit.TUICommonDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallEngine
import com.tencent.qcloud.tuikit.tuicallengine.TUICallObserver

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
```

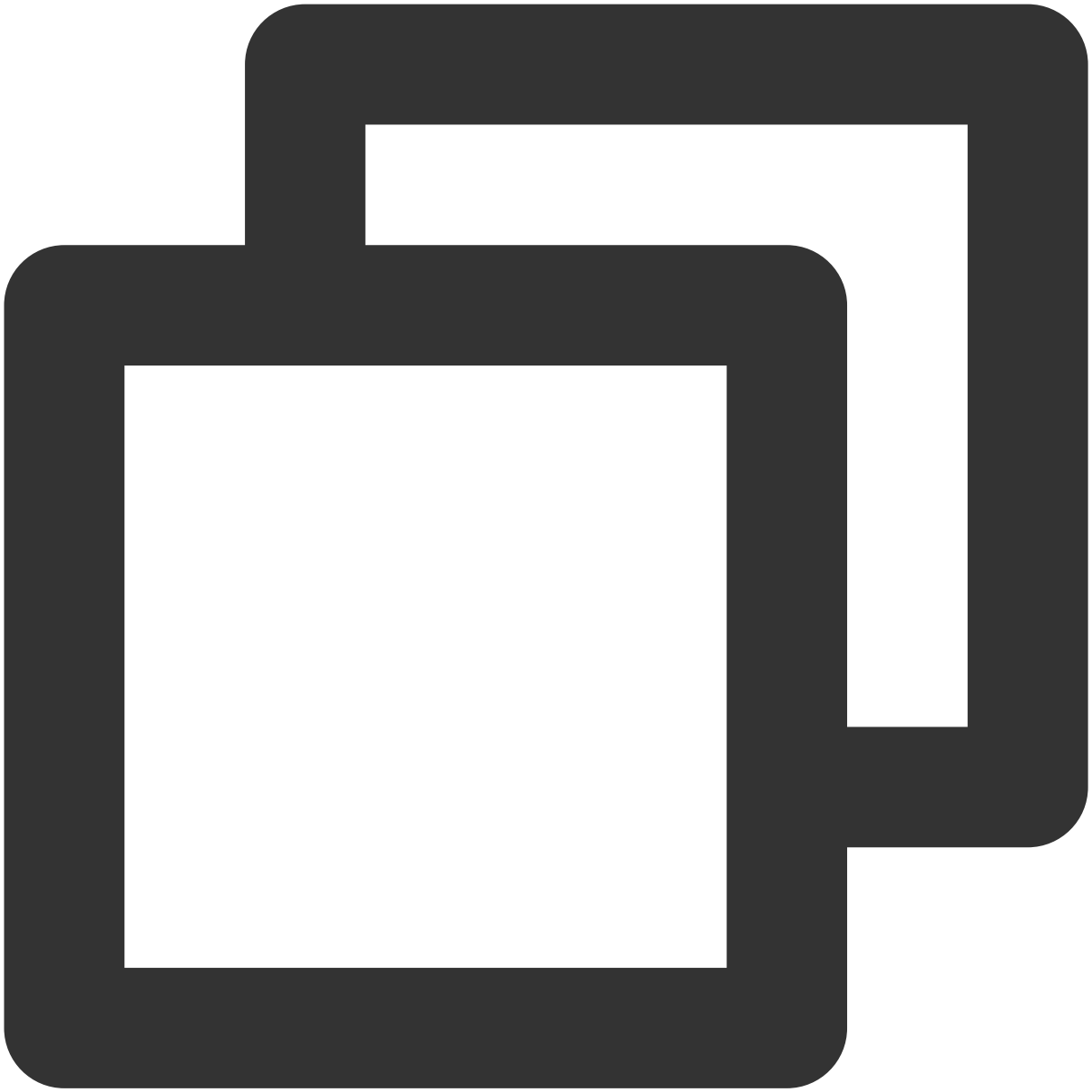
```
    }
}
private fun initData() {
    TUICallEngine.createInstance(context).addObserver(observer)
}
```



```
import com.tencent.qcloud.tuikit.TUICommonDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallDefine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallEngine;
import com.tencent.qcloud.tuikit.tuicallengine.TUICallObserver;
```

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

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



```
import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

func onCallBegin(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole: TUIC

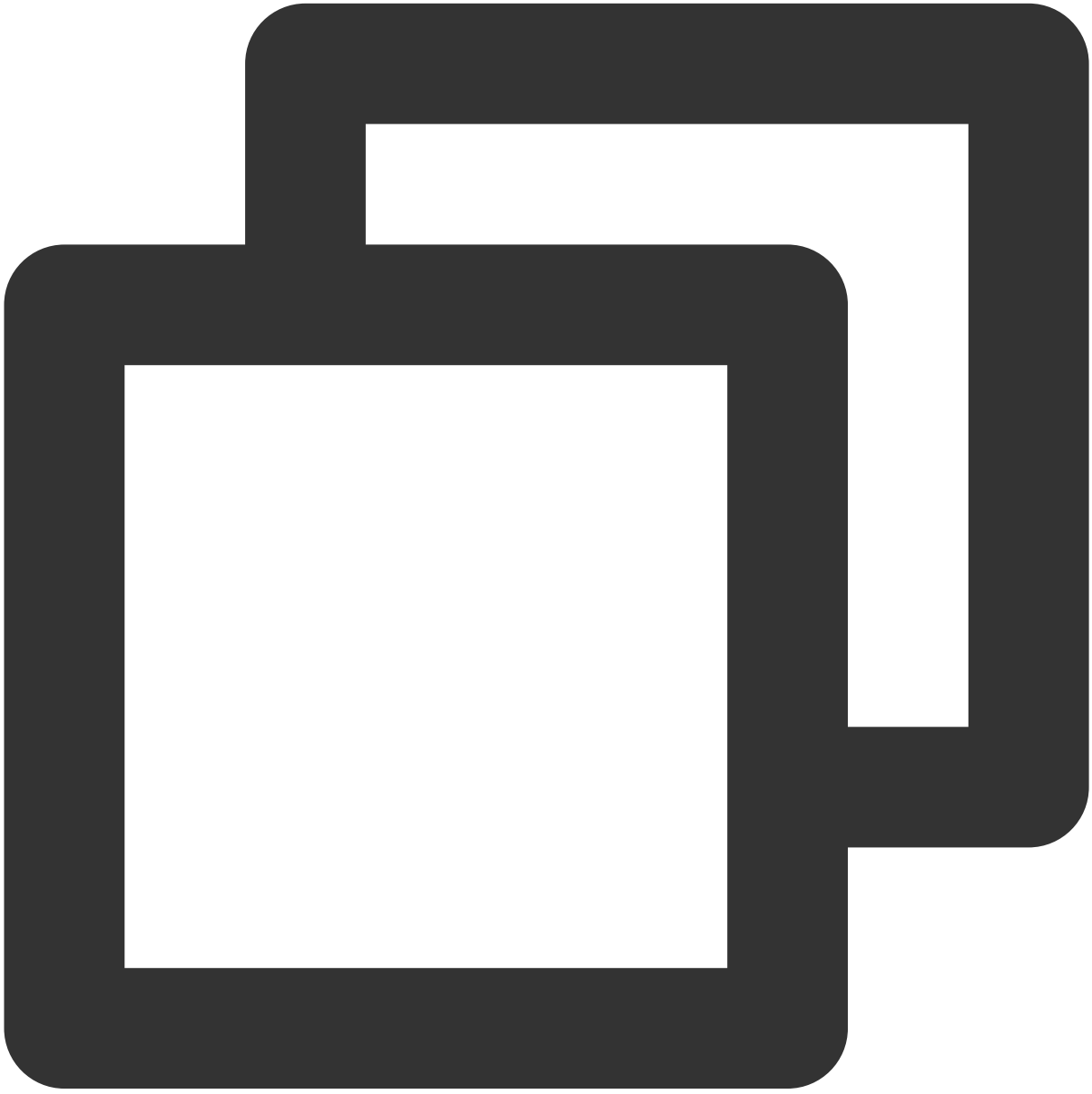
}

func onCallEnd(roomId: TUIRoomId, callMediaType: TUICallMediaType, callRole: TUICal

}
```

```
func onUserNetworkQualityChanged(networkQualityList: [TUINetworkQualityInfo]) {

}
```



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

[[TUICallEngine createInstance] addObserver:self];

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

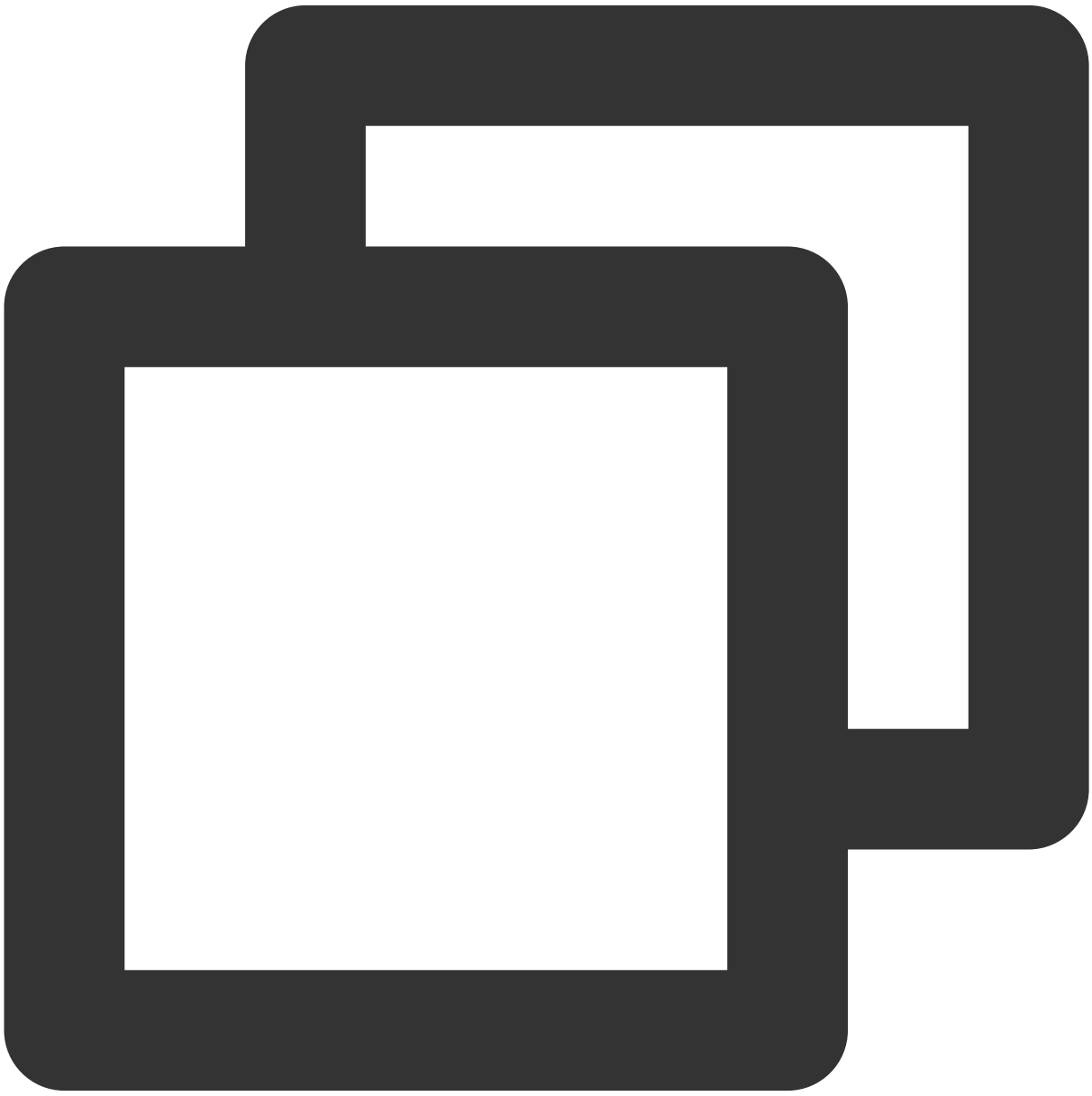
```
}

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

}

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

}
```



```
import 'package:tencent_calls_engine/tencent_calls_engine.dart';
```

```
TUICallObserver observer = TUICallObserver(
    onError: (int code, String message) {

    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole

    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca

    }, onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList

    }, onCallReceived: (String callerId, List<String> calleeIdList, String groupId,

    }
))

void addObserver() {
    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。

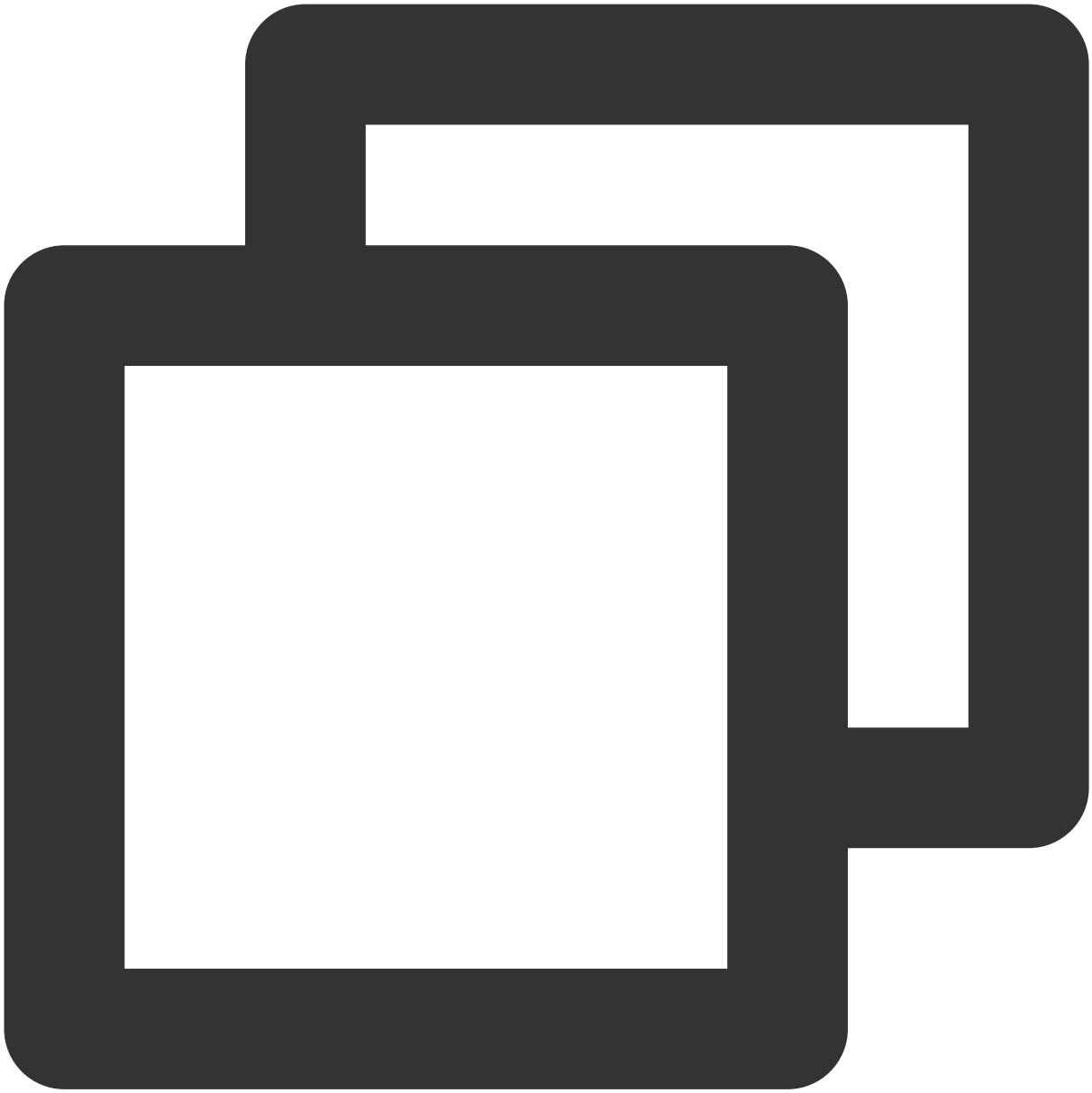
Web&H5

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

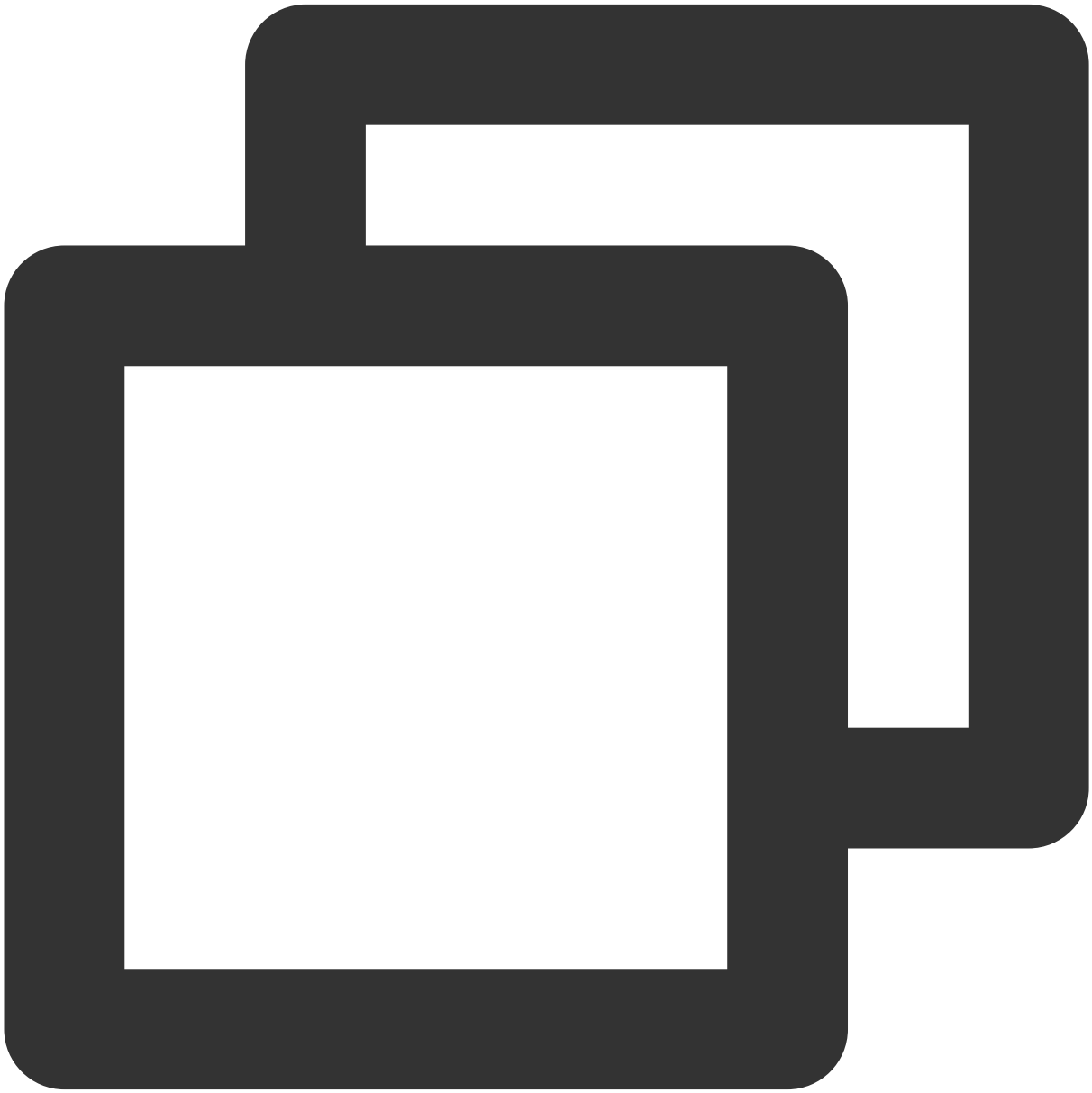
TUICallKit 组件提供了通话状态回调，可以用于业务侧实现更多交互逻辑，具体参考 TUICallKit 属性介绍。

`beforeCalling` ：通话开始前会执行。

`afterCalling` ：通话完成后执行。

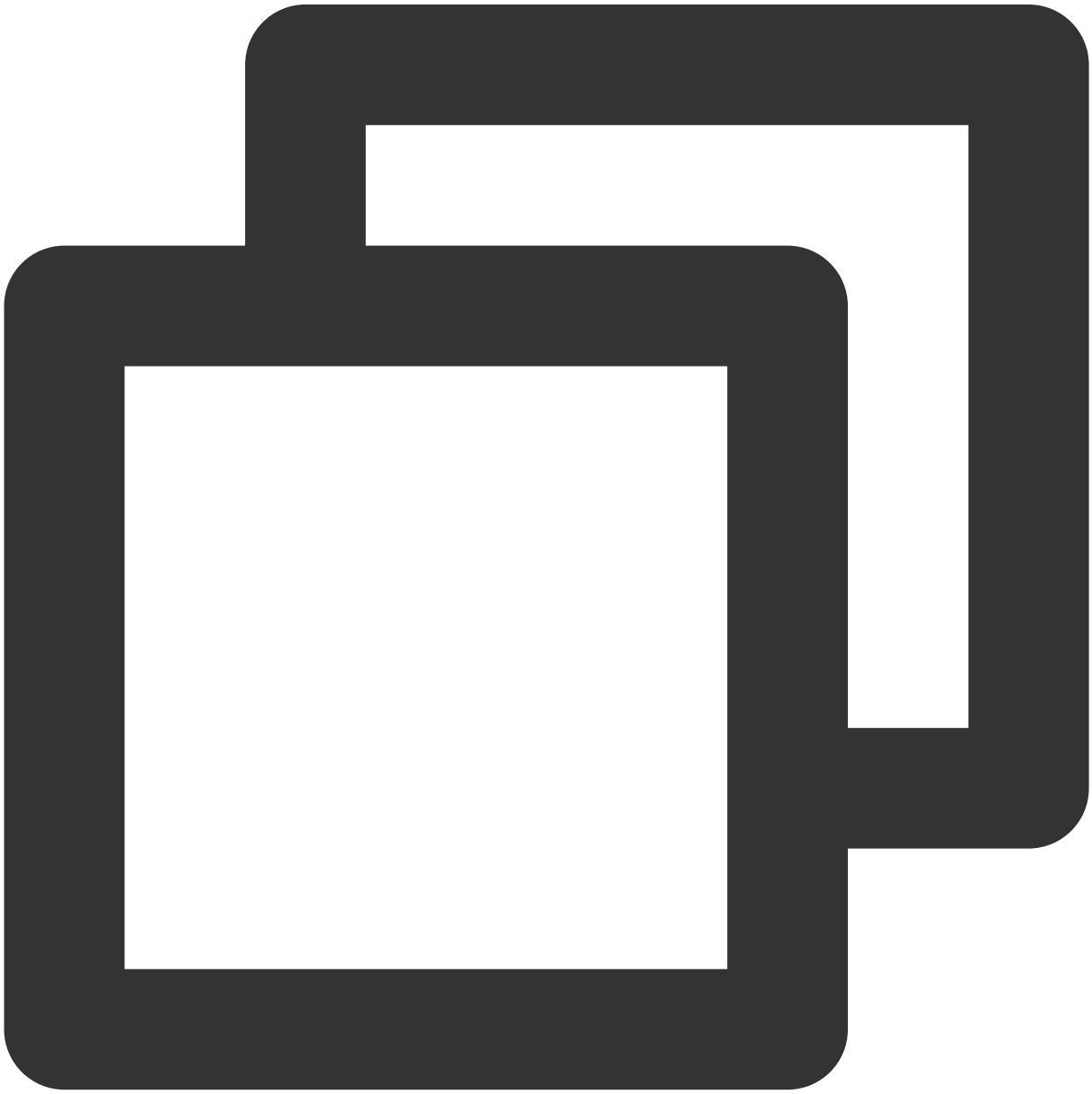
React

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

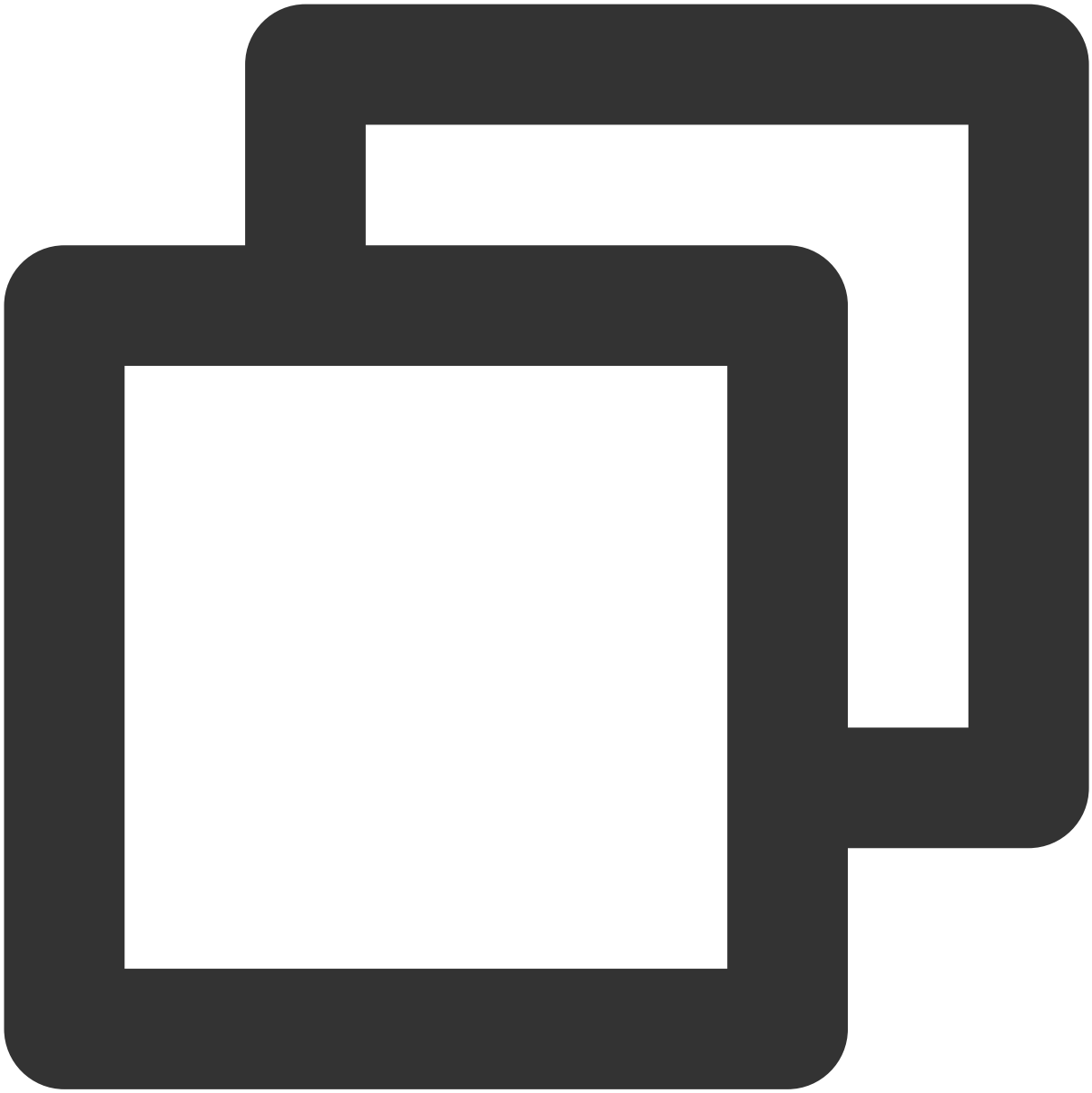
uni-app（Android & iOS）

最近更新时间：2024-04-30 10:43:45

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），请参考如下代码。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin

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

版权所有：腾讯云计算（北京）有限责任公司

语言设置

Web&H5

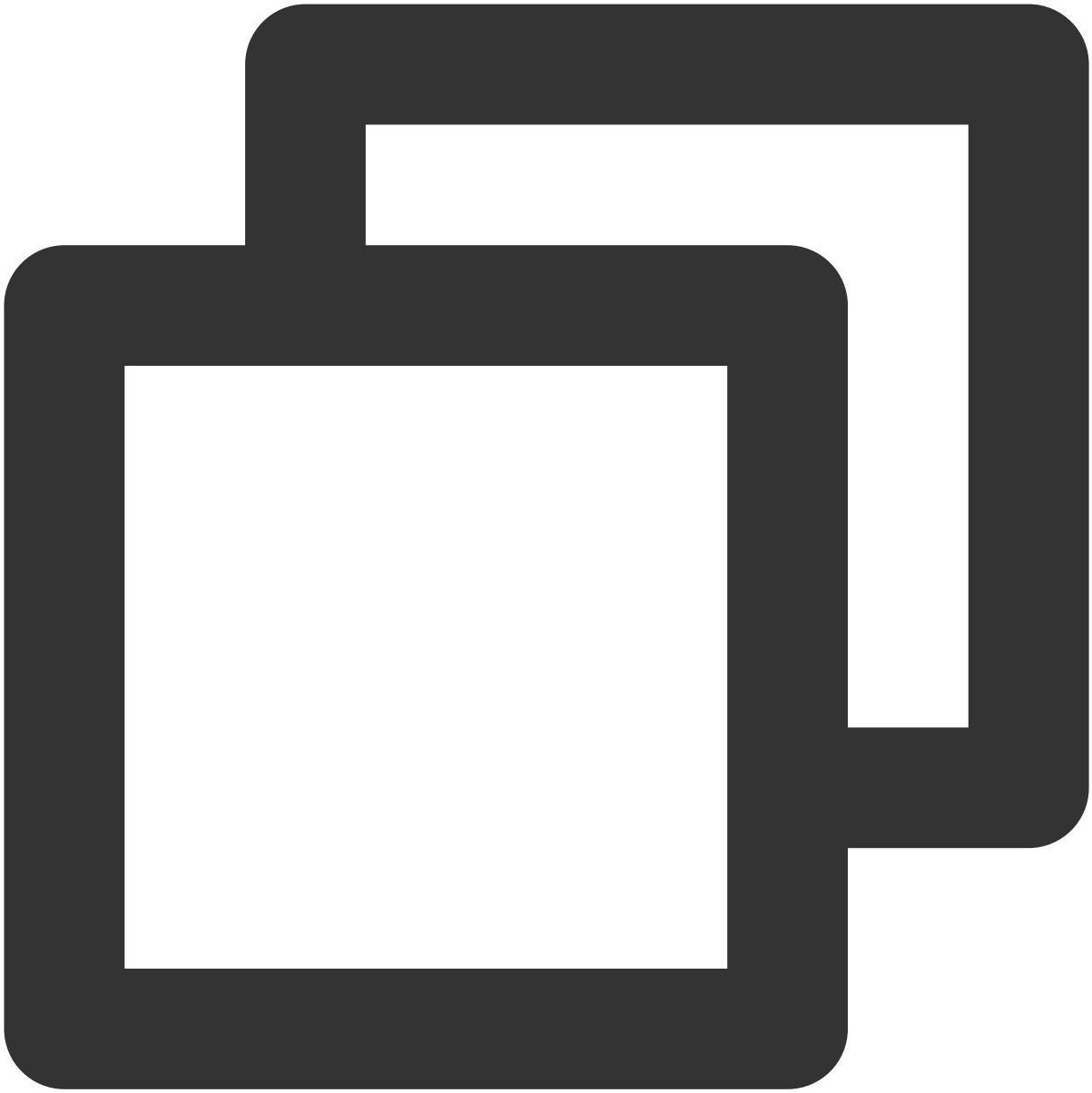
最近更新时间：2024-07-26 10:11:25

支持的语言

目前支持**简体中文、英文和日文**。

切换语言

TUICallKit 会优先获取浏览器语言，如果是中文，英文或者日文中的一种，则以浏览器语言为主，否则使用英文。如果想切换语言，可以使用 setLanguage 接口。



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-vue';

TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn" | "ja_JP"
```

添加新的语言

如果您有其他语言需要支持，可以通过源码集成方式修改语言源文件实现。

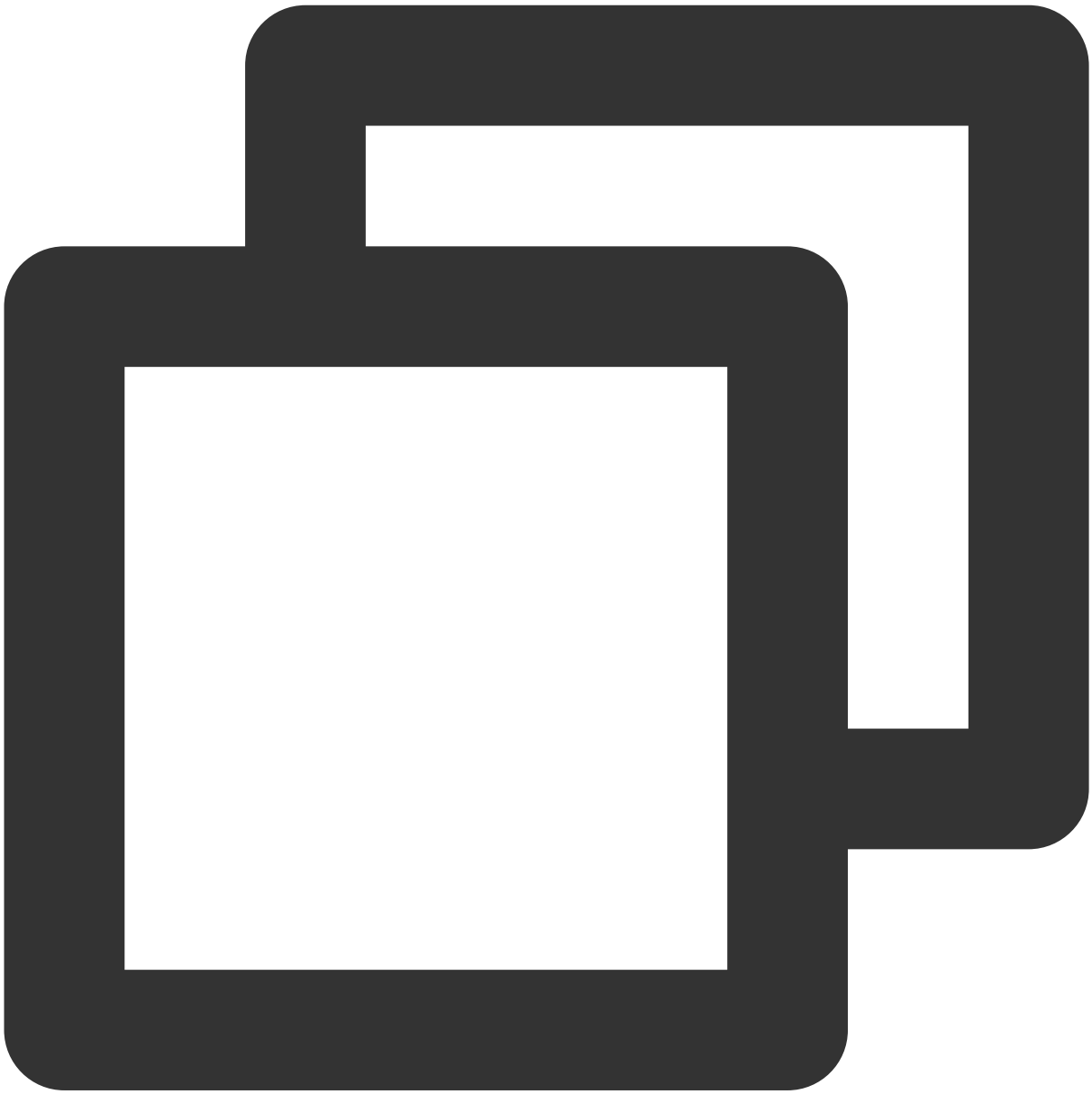
第一步：源码集成

注意：

源码集成适用于 `Vue + Typescript` 项目且 `TUICallKit` 版本号 ≥ 3.2.2。

1. **下载源码**

Vue3

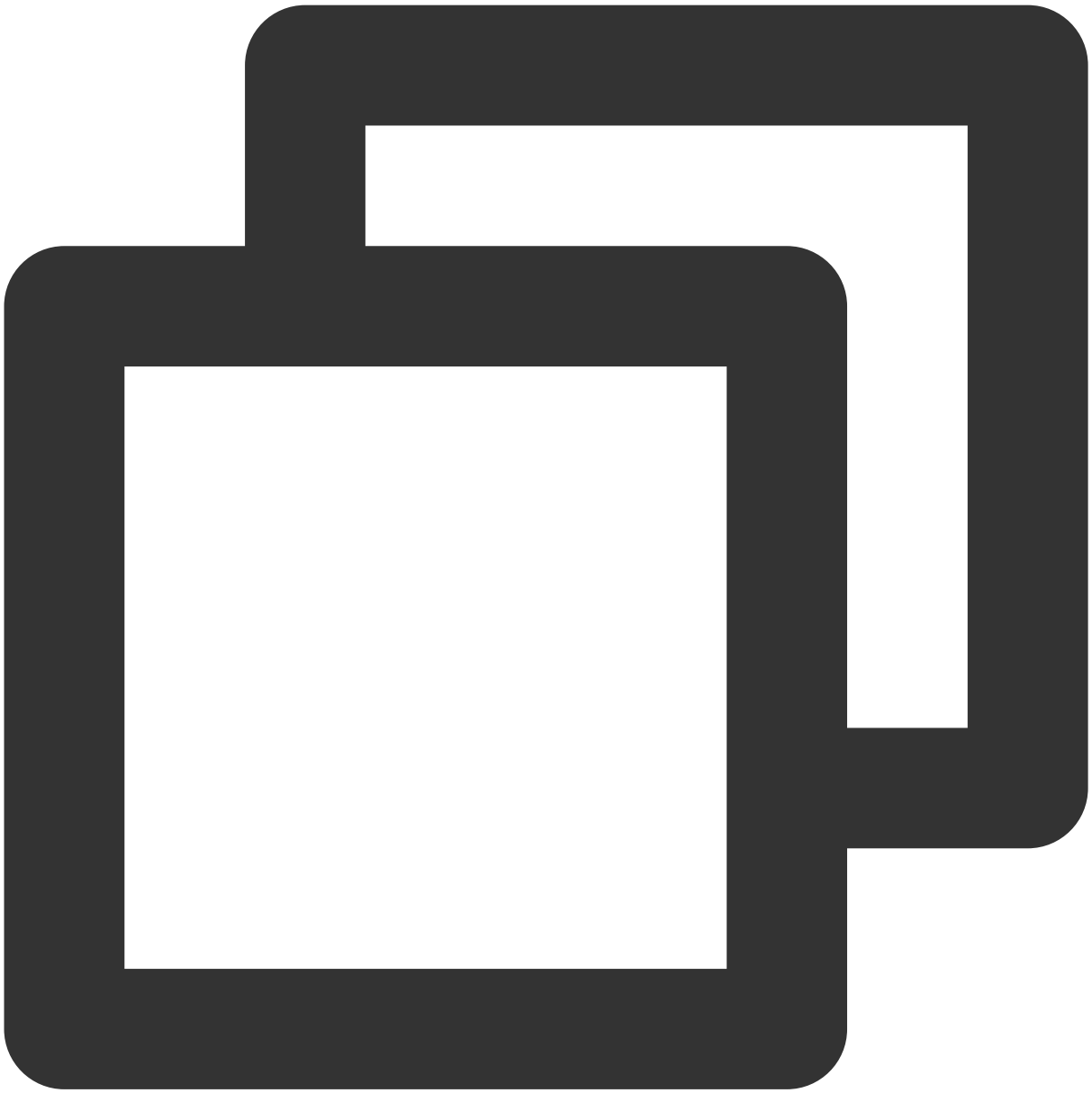


```
npm install @tencentcloud/call-uikit-vue
```

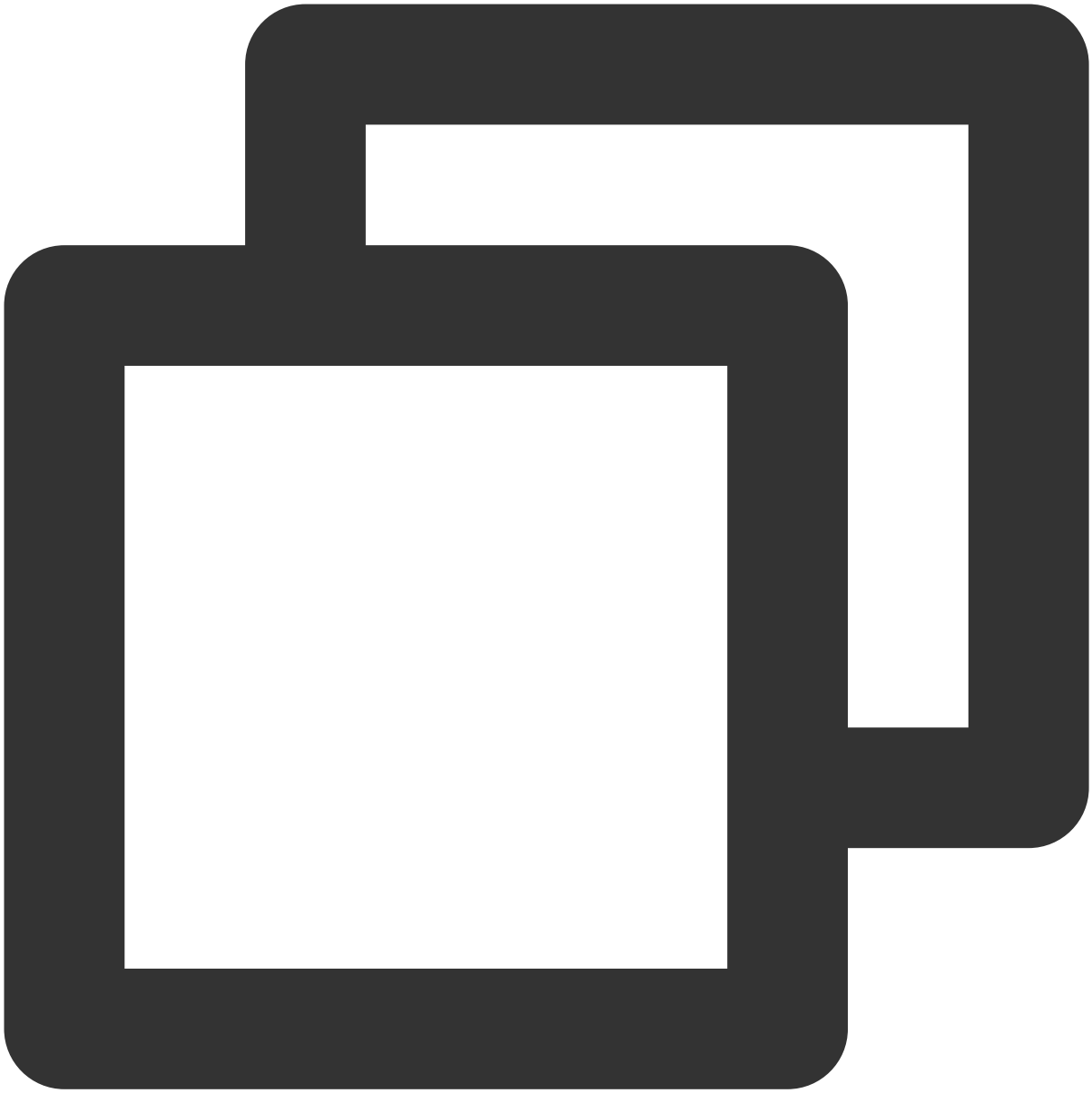
2. **将源码拷贝到自己的项目中，以拷贝到 `src/components/` 目录为例：**

macOS + Vue3

Windows + Vue3



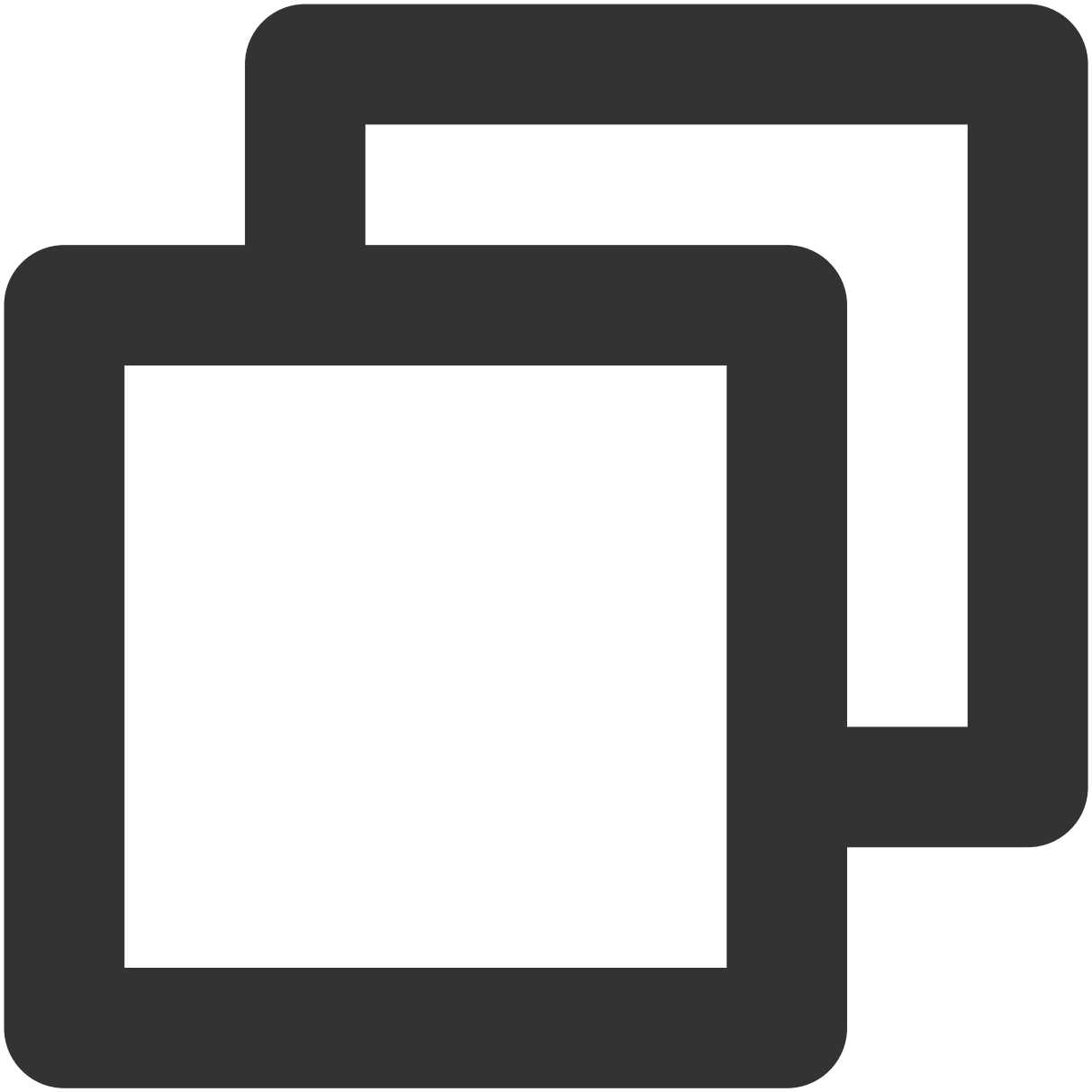
```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uik
```



```
xcopy .\\node_modules\\@tencentcloud\\call-uikit-vue  .\\src\\components\\TUICallKi
```

3. 修改引入路径

需要将 TUICallKit 改为从本地文件中引入，如下方代码。其他用法细节可参考见 TUICallKit 快速接入。



```
import { TUICallKit, TUICallKitServer, TUICallType } from "./components/TUICallKit/
```

4.

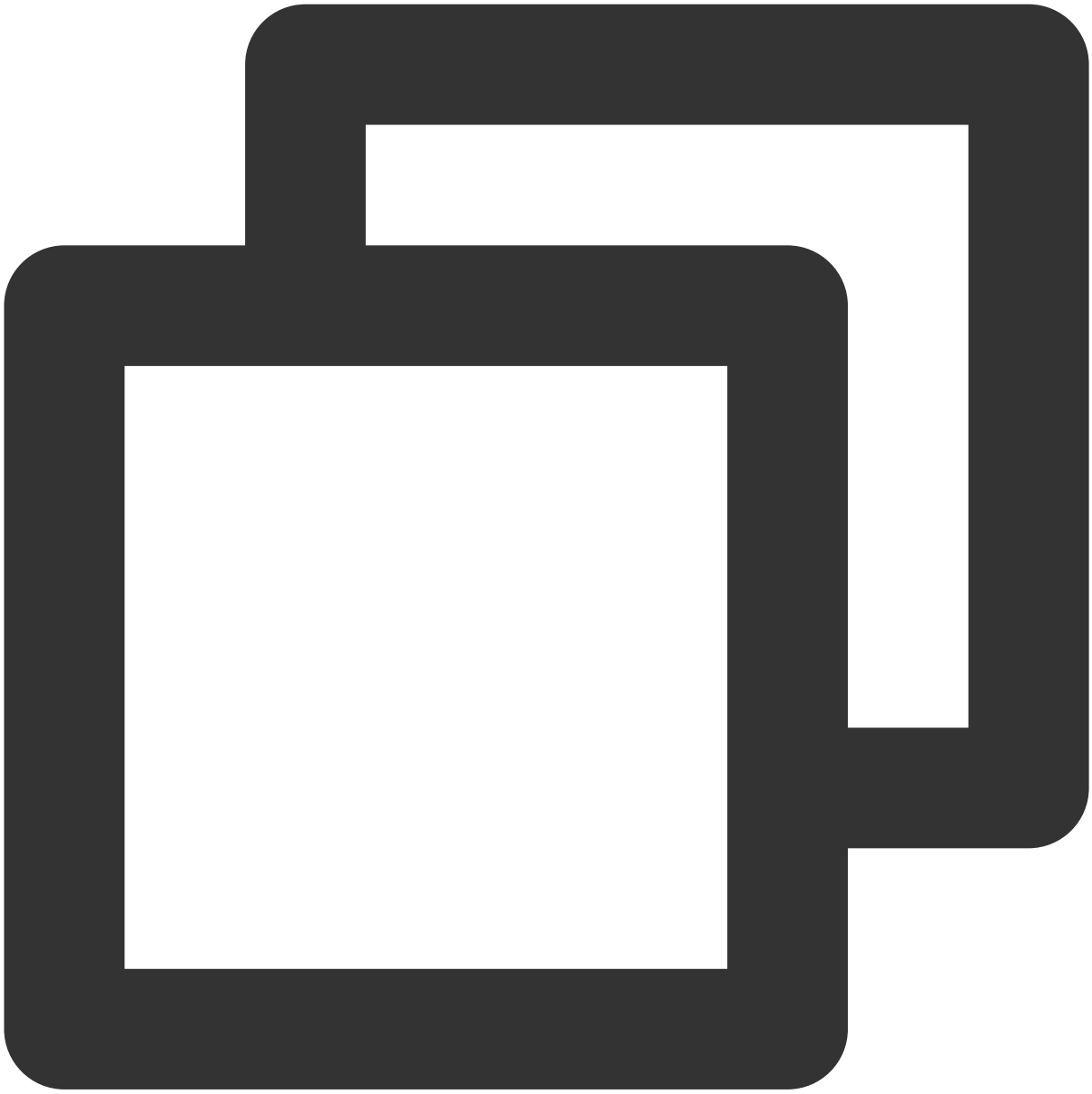
解决源码拷贝可能导致的报错

如果您在使 TUICallKit 组件时遇到了报错，请不要担心，大多数情况下这是由于 ESLint 和 TSConfig 配置不一致造成的。您可以查阅文档，按照要求正确配置即可。如果您需要帮助，请随时联系我们，我们将确保您能够成功地使用此组件。以下是几个常见的问题：

ESLint 报错

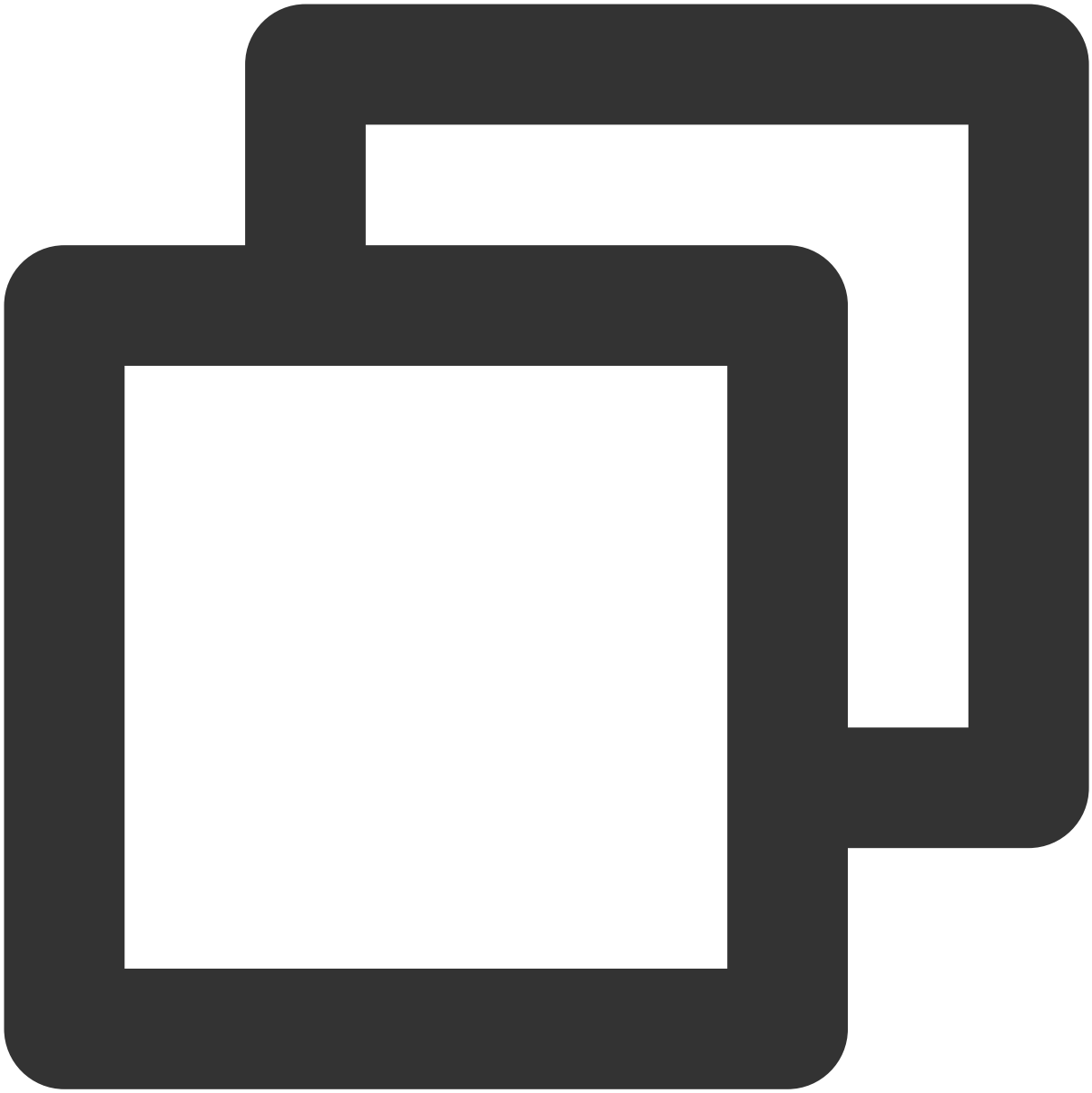
TypeScript 报错

若 TUICallKit 与您项目的代码风格不一致导致报错，可将本组件目录屏蔽，如在项目根目录增加 `.eslintignore` 文件，如：



```
# .eslintignore
src/components/TUICallKit
```

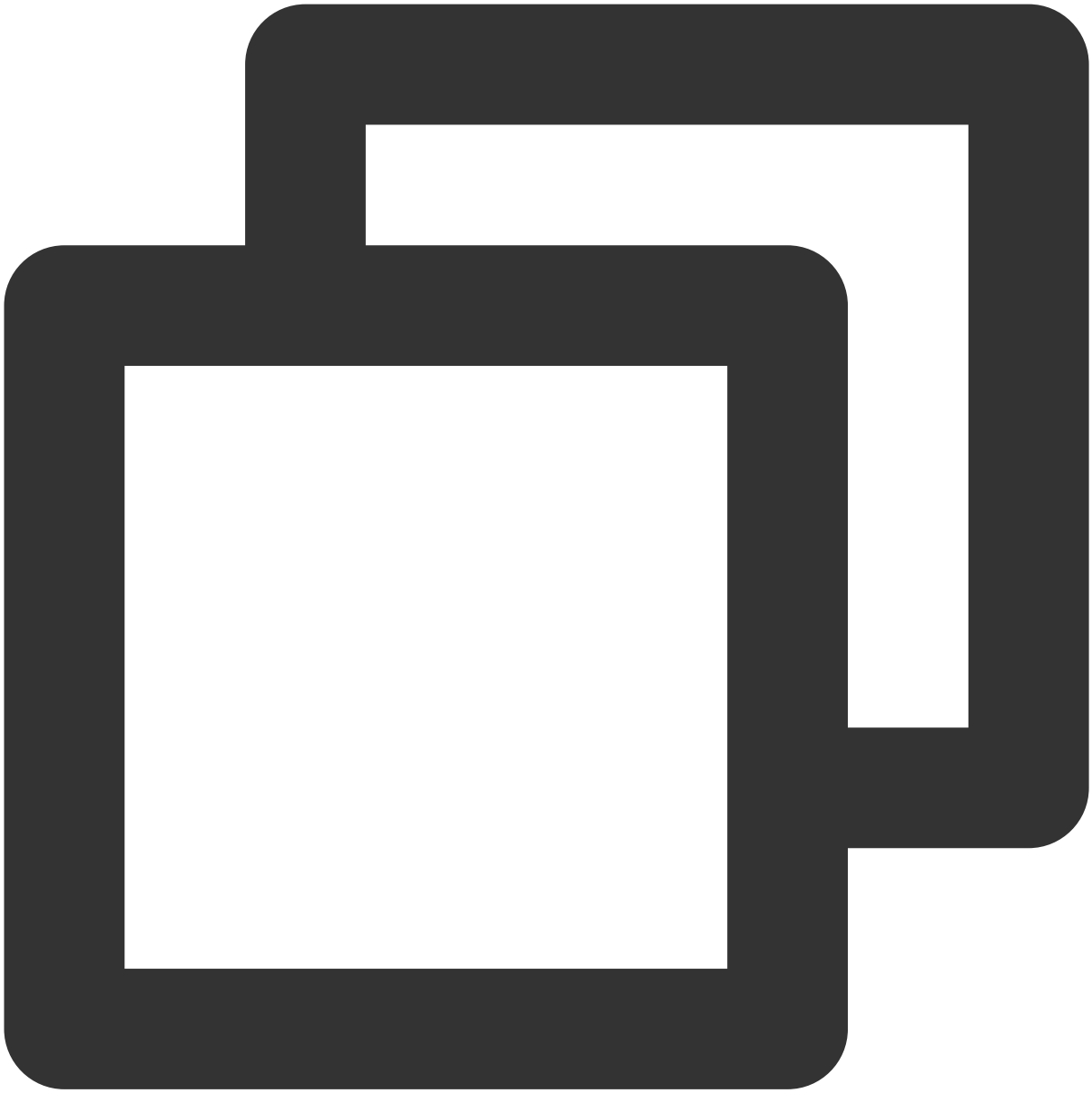
1. 如遇 Cannot find module '../package.json' 报错，是因为 TUICallKit 内引用了 JSON 文件，可在 tsconfig.json 中添加相关配置，示例：



```
{
  "compilerOptions": {
    "resolveJsonModule": true
  }
}
```

其他 TSConfig 问题请参见 TSConfig Reference。

2. 如遇 Uncaught SyntaxError: Invalid or unexpected token 报错，是因为 TUICallKit 使用了装饰器，可在 tsconfig.json 中添加相关配置，示例：



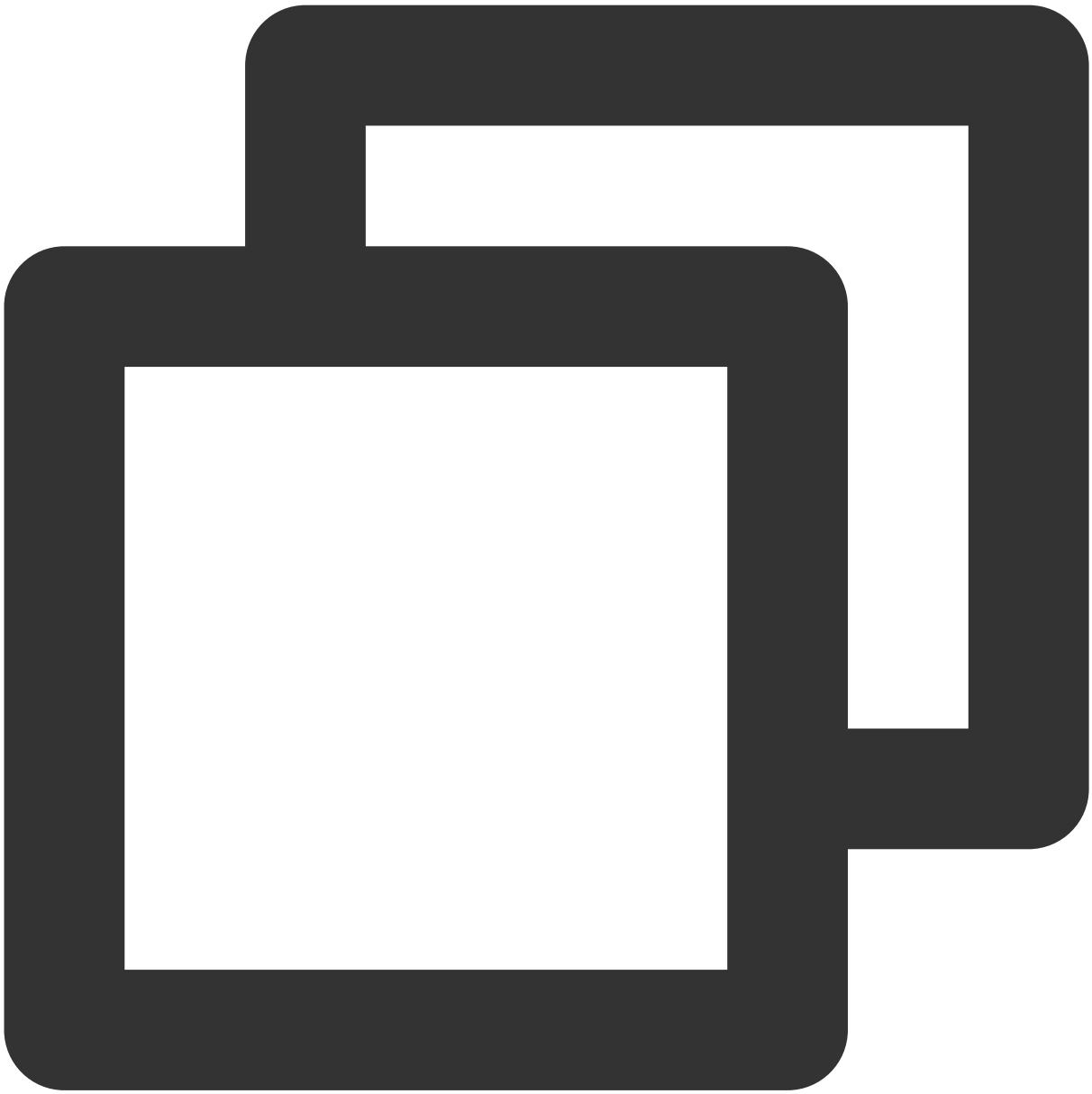
```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

第二步：新增语言包

以新增越南语为例：

1. 创建目标语言源文件。

在 src/components/TUICallKit/src/TUICallService/locales 目录下新增 vi.ts 文件，复制 src/components/TUICallKit/src/TUICallService/locales/zh-cn.ts 内容到 vi.ts，然后将 json 的 value 翻译成越南语。

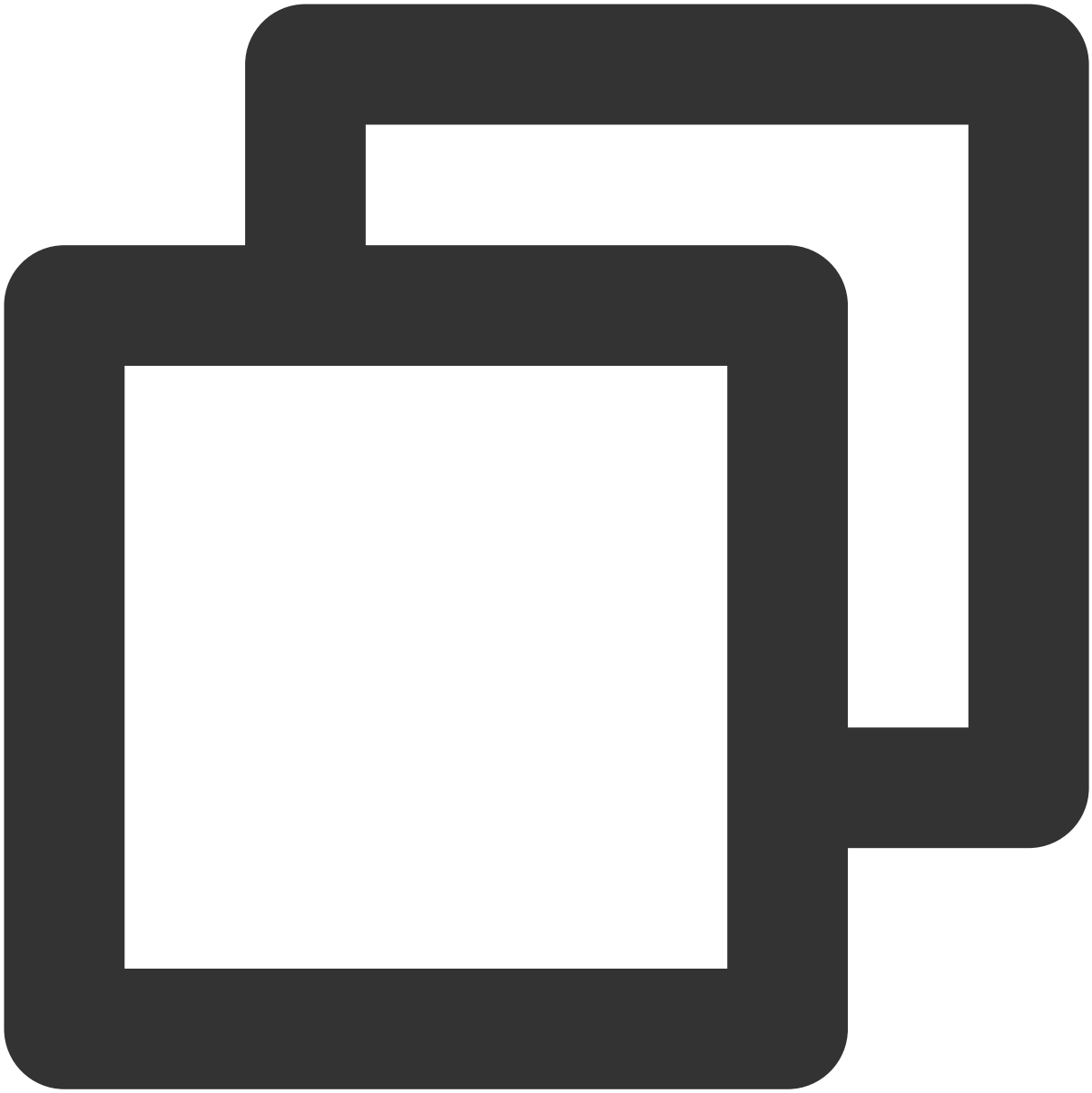


```
export const vi = {     // 注意这里的导出变量
  'hangup': '挂断',
  'reject': '拒绝',
  'accept': '接受',
  'camera': '摄像头',
  'microphone': '麦克风',
  'speaker': '扬声器',
  'open camera': '打开摄像头',
```

```
    'close camera': '关闭摄像头',
    'open microphone': '打开麦克风',
    'close microphone': '关闭麦克风',
    'video-to-audio': '转语音通话',
    'virtual-background': '模糊背景',
    'other side reject call': '对方已拒绝',
    'reject call': '拒绝通话',
    'cancel': '取消通话',
    ...
};
```

2. 从 index.ts 导出

修改 src/components/TUICallKit/src/TUICallService/locales/index.ts 文件。



```
import { TUIStore } from '../CallService/index';
import { NAME, StoreName } from '../const/index';
import { en } from './en';
import { zh } from './zh-cn';
import { ja_JP } from './ja_JP';
import { vi } from './vi';  // 新增语言文件导入

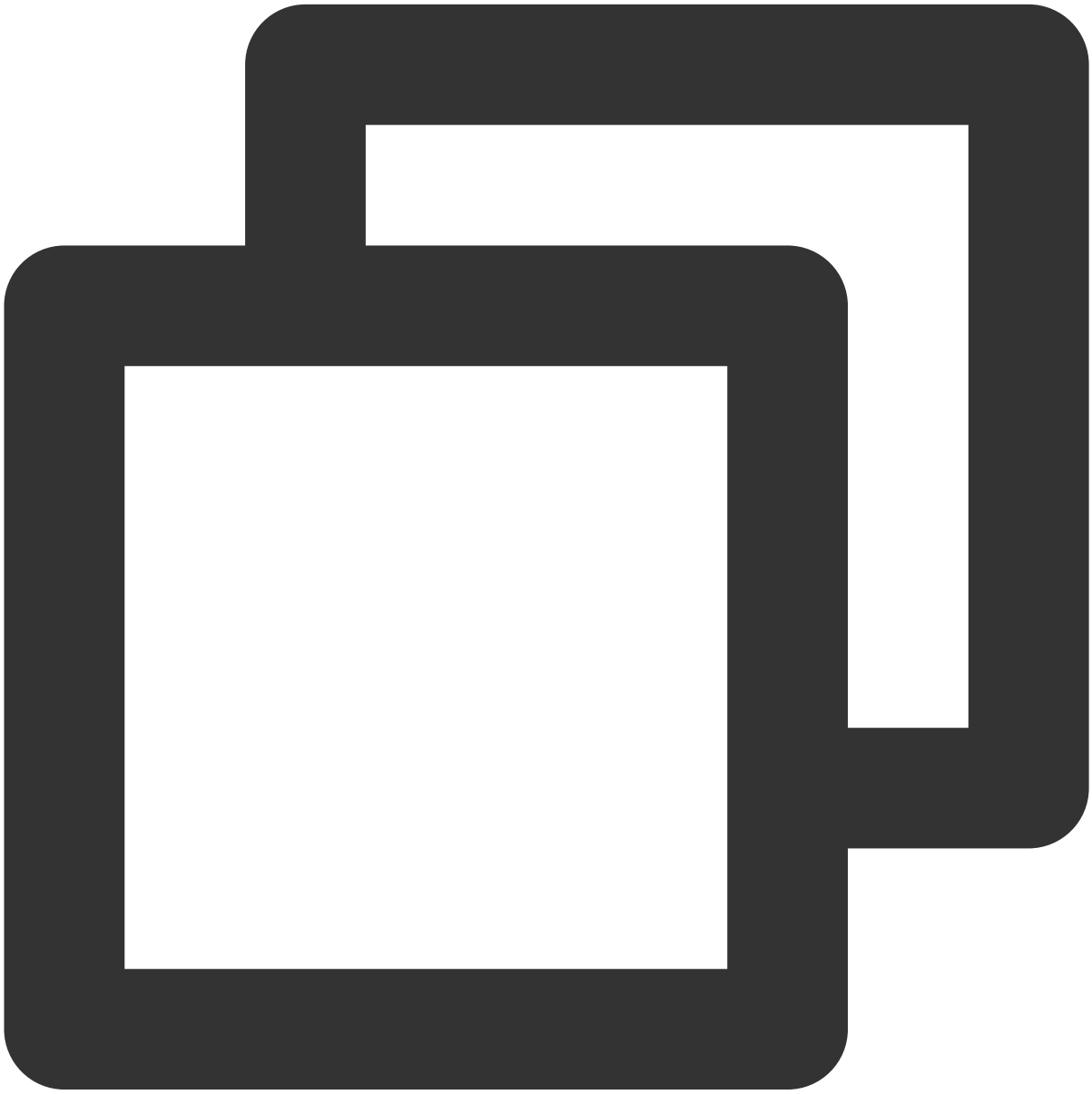
.....

export const languageData: languageDataType = {
  en,
```

```
  'zh-cn': zh,
  ja_JP,
  vi,        // 新增语言文件导出
};
```

3. 新增 LanguageType 枚举。

修改 src/components/TUICallKit/src/TUICallService/const/call.ts

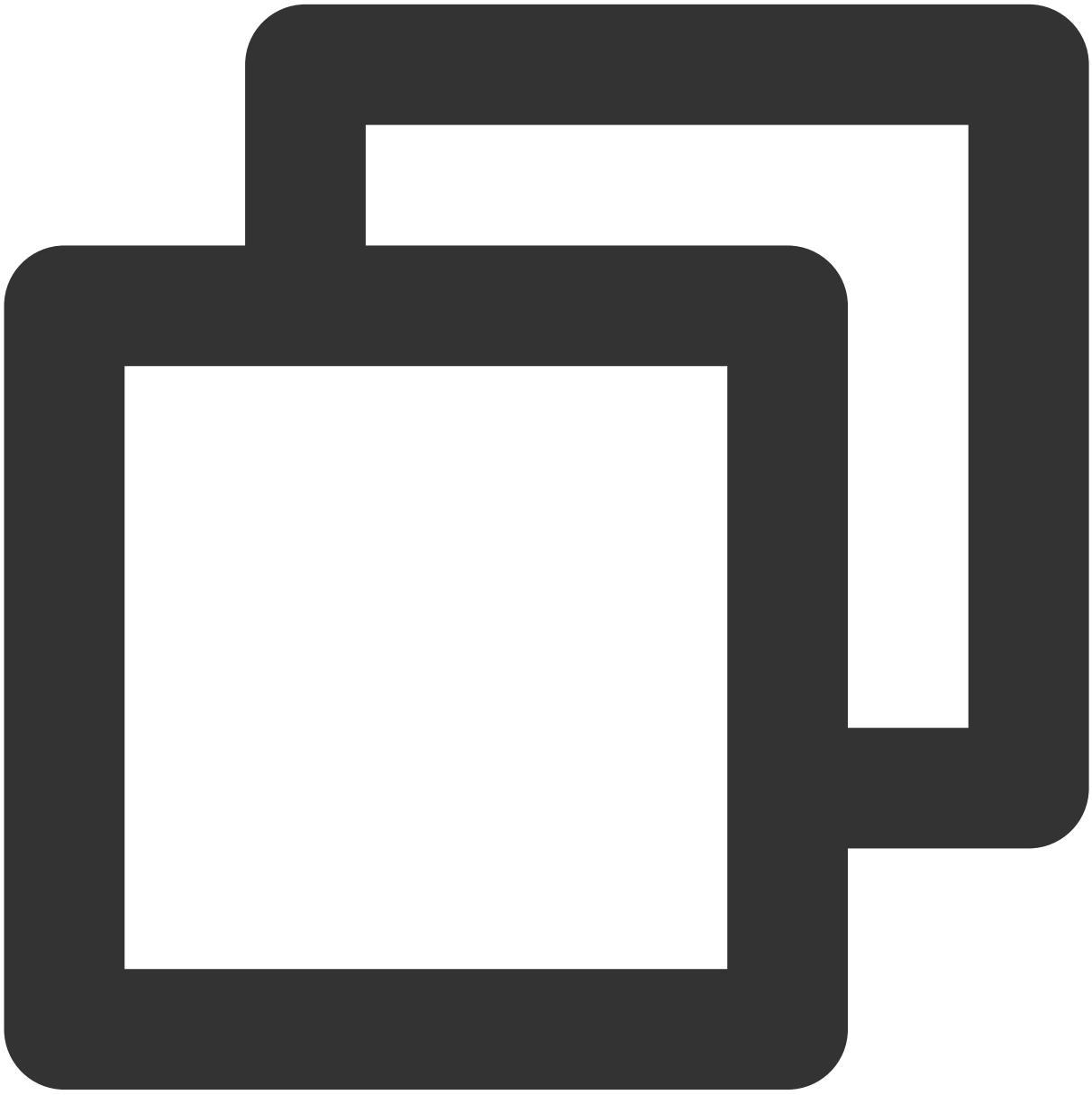


```
export enum LanguageType {
  EN = 'en',
  'ZH-CN' = 'zh-cn',
```

```
  JA_JP = 'ja_JP',
  VI = 'vi',    // 新增枚举类型
}
```

4. 切换语言

在项目中通过调用 setLanguage 接口切换语言。



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-vue';

TUICallKitServer.setLanguage("vi");
```

iOS

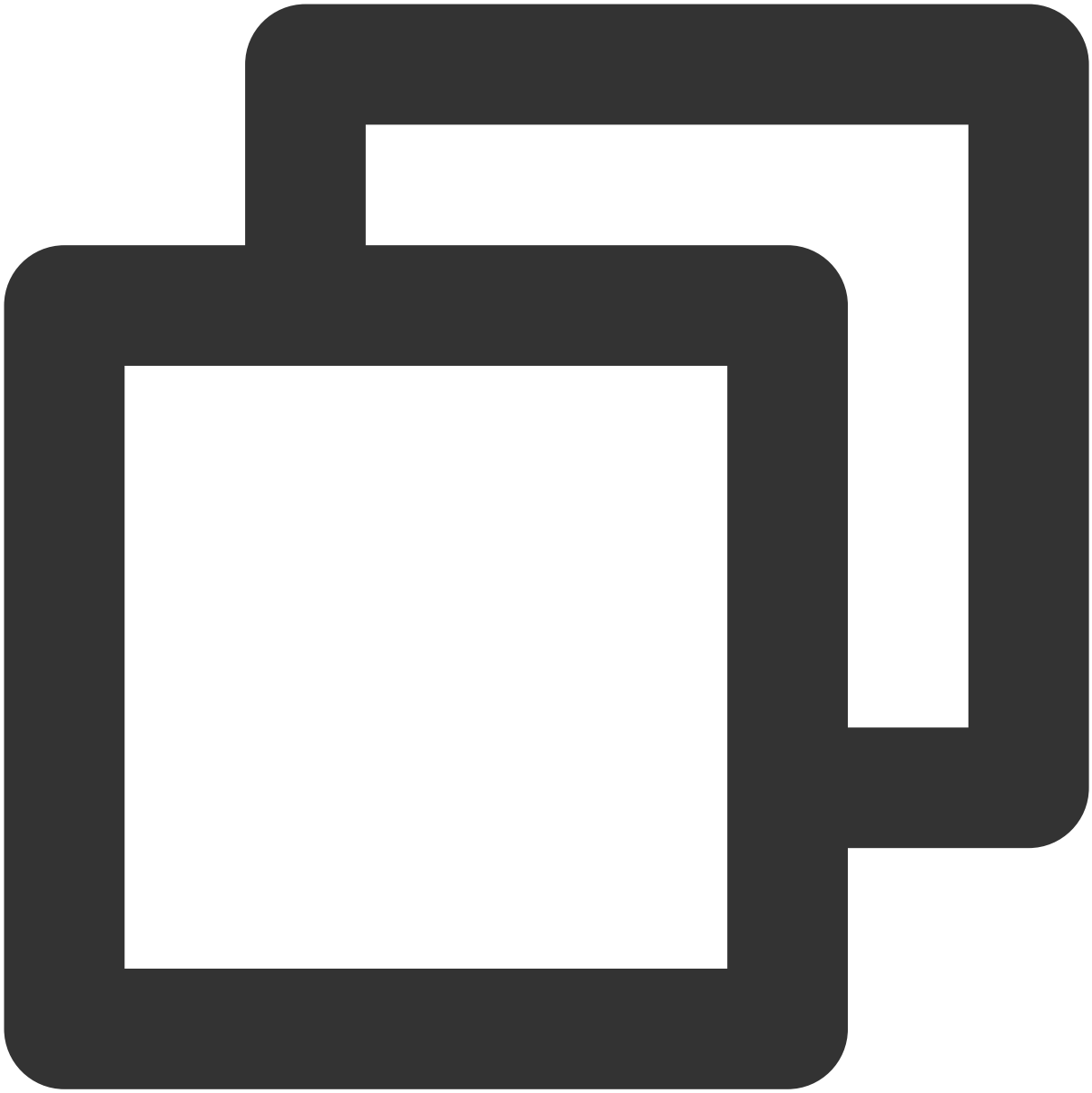
最近更新时间：2024-08-15 11:22:55

支持的语言

目前支持**简体中文、英文、日文和阿拉伯语。**

切换语言

TUICallKit 默认语言与手机系统保持一致 。如果需要切换语言，可以使用 `TUIGlobalization.setPreferredLanguage` 切换语言，以切换英文为例：



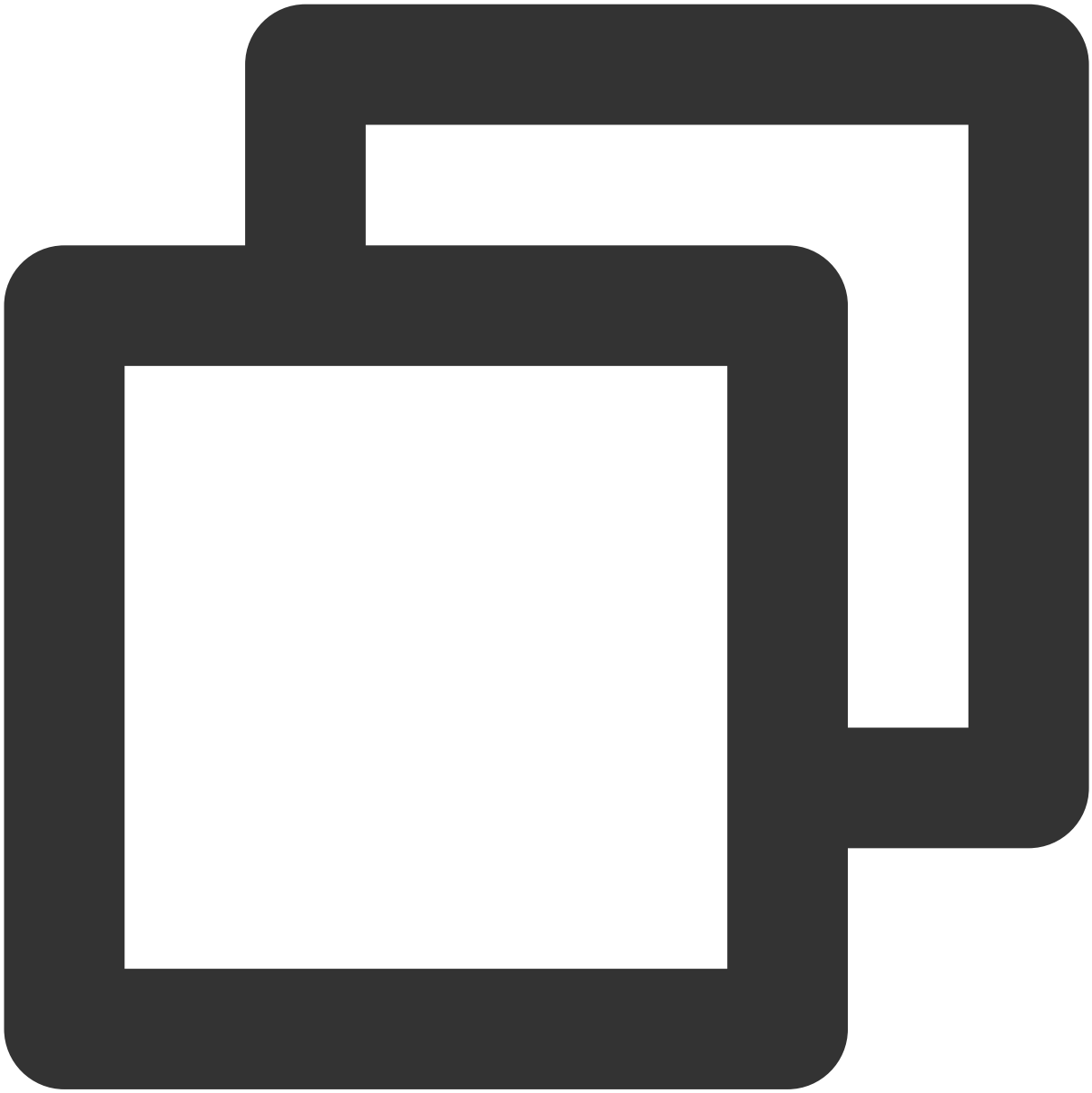
```
……
import TUICore

func steLanguage() {
    TUIGlobalization.setPreferredLanguage("en")
}
```

添加新的语言

第一步：源码集成

1. 在 Github中克隆/下载代码。
2. 在 Application 工程的 Podfile 文件中依赖下载到本地 TUICallKit 源码。



```
target 'TUICallKitApp' do
  use_frameworks!
  ……
  pod 'TUICallKit-Swift/Professional', :path => "Your Download Path/TUICallKit/iOS"

end
```

```
```

3. 执行 `pod update` 命令，更新依赖。

第二步：新增语言包

以西班牙语为例：

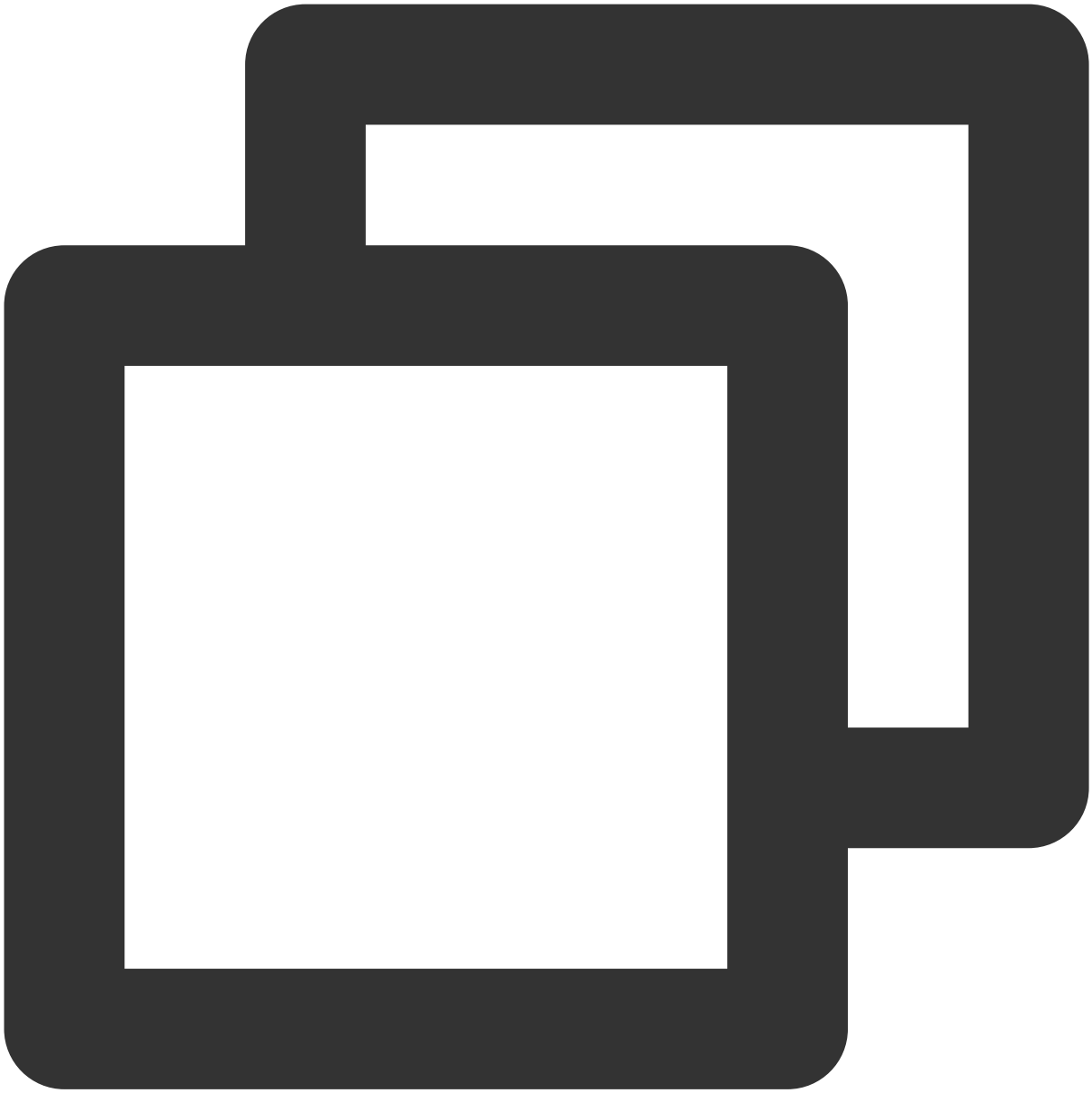
1. 新增西班牙语文件。

进入到 `TUICallKit` 源码文件目录下的 iOS/TUICallKit-Swift/Resources 目录下，新增 `es.lproj/Localized.strings` 文件 。

2. 将 `iOS/TUICallKit-Swift/Resources/en.lproj/Localized.strings` 中的内容复制到新增的 `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` 文件中。

3. 将 `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` 中的英文翻译为西班牙语。

4. 进入到 Application 工程的 Podfile 所在的目录文件夹下执行 `pod install` 命令，更新依赖。



```
  ……
import com.tencent.qcloud.tuicore.TUIThemeManager;

public class MainActivity extends BaseActivity {
    @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      Locale locale = new Locale("es");
      TUIThemeManager.addLanguage("es", locale);
      ……
```

```
        }
        ......
    }
```

Android

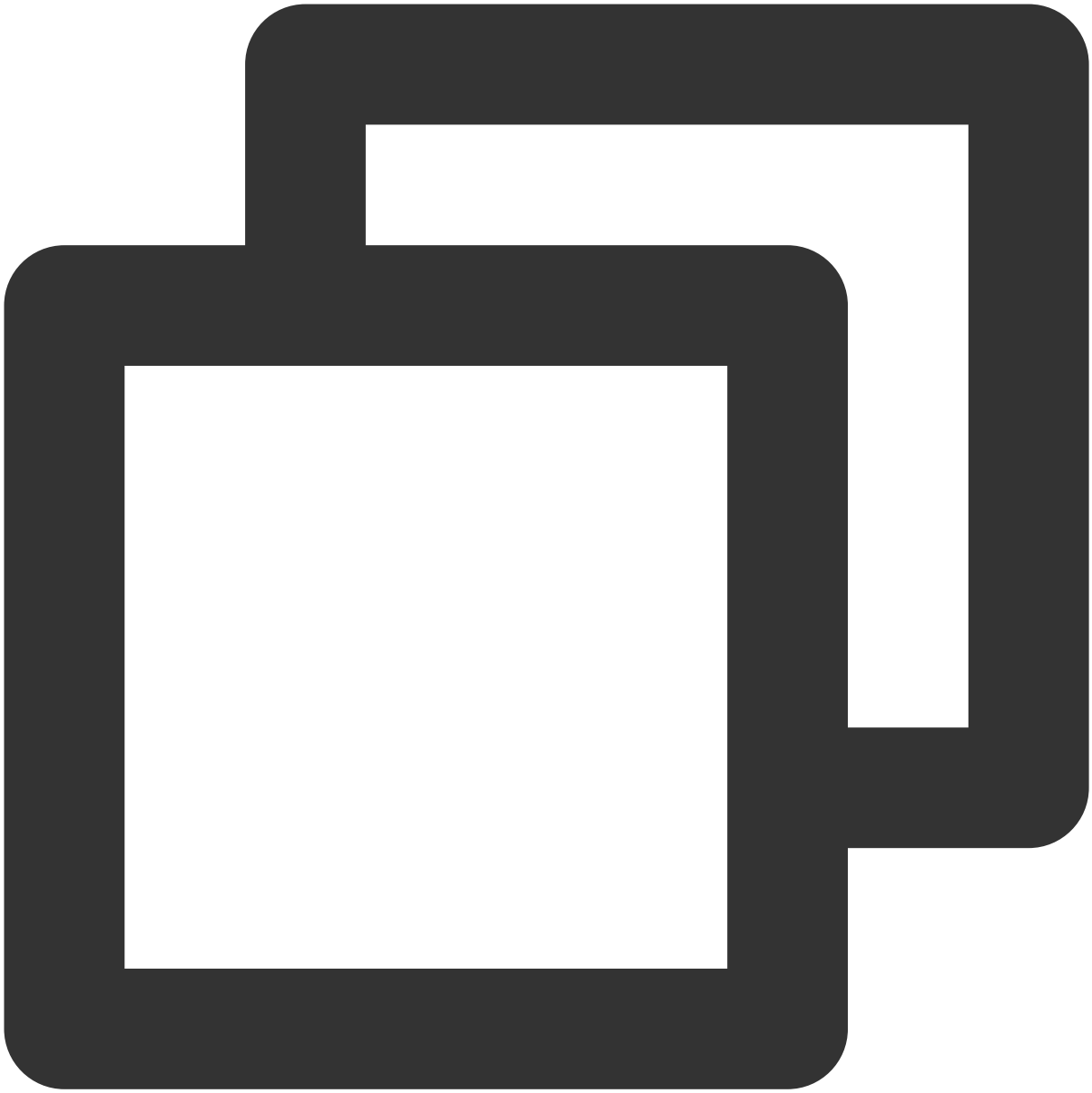
最近更新时间：2024-08-15 11:22:55

支持的语言

目前支持**简体中文、英文、日文和阿拉伯语。**

切换语言

TUICallKit 默认语言与手机系统保持一致 。如果需要切换语言，可以使用 `TUIThemeManager.getInstance().changeLanguage` 切换语言，以切换英文为例：



```
……
import com.tencent.qcloud.tuicore.TUIThemeManager;

public class MainActivity extends BaseActivity {
    @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      TUIThemeManager.getInstance().changeLanguage(getApplicationContext(), "en");
      ……
    }
```

```
        ......
}
```

添加新的语言

第一步：源码集成

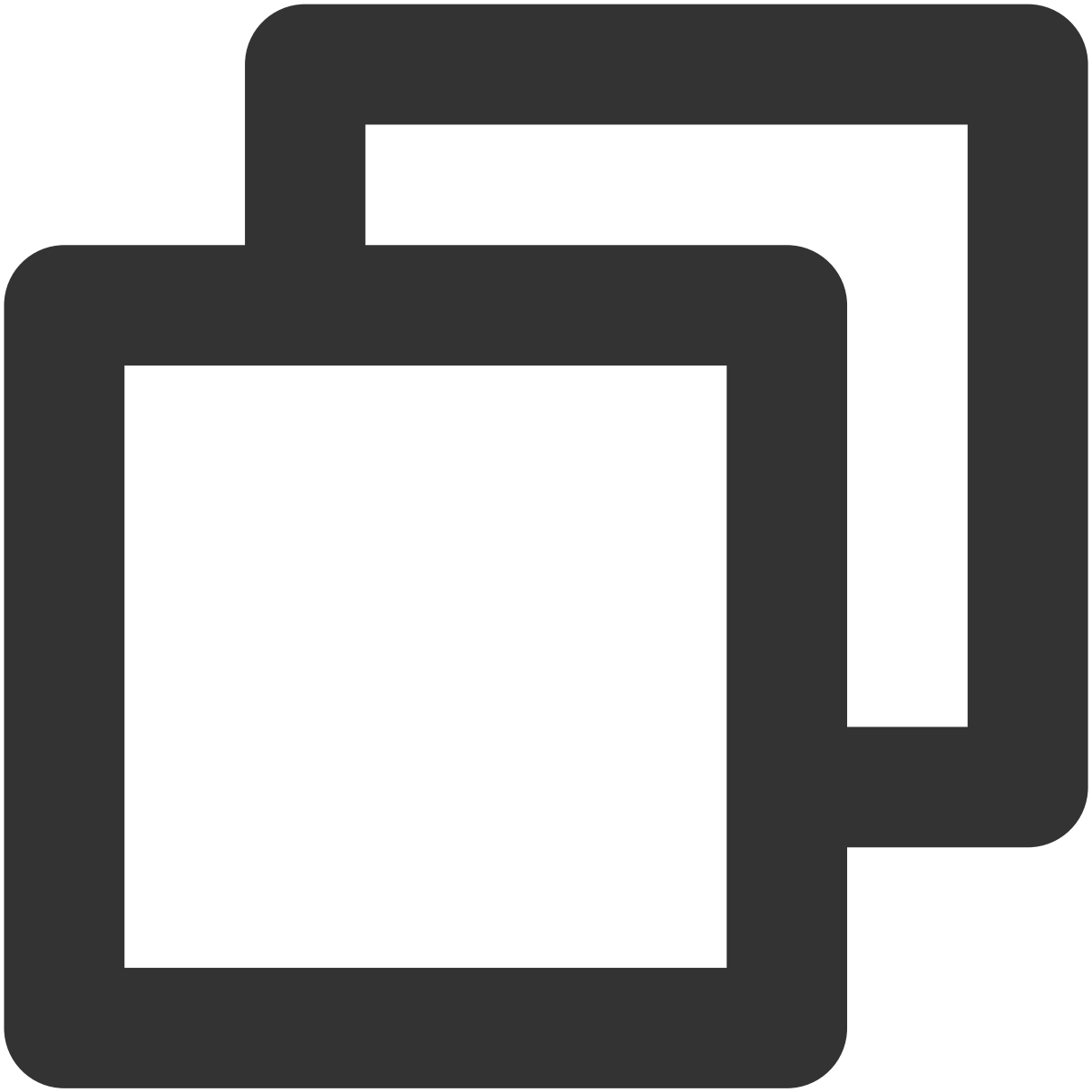
1. 在 Github 中克隆/下载代码，然后拷贝 Android 目录下的 tuicallkit-kt 子目录到您当前工程中的 app 同一级目录中，如下图所示。



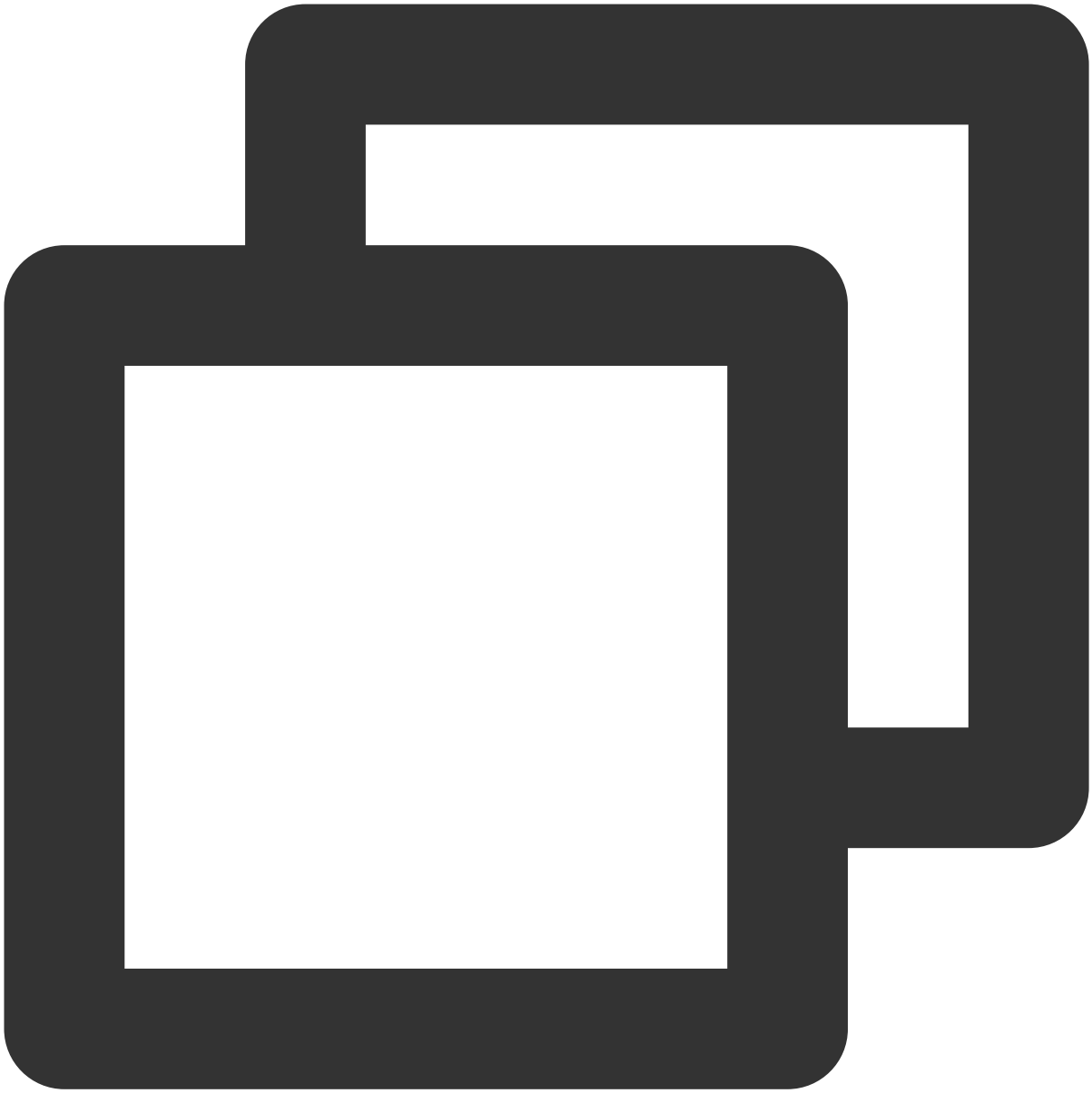
2. 在工程根目录下找到 `settings.gradle.kts（或settings.gradle）` 文件，在其中增加如下代码，导入 `tuicallkit-kt` 组件到项目中。

setting.gradle.kts

settings.gradle



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

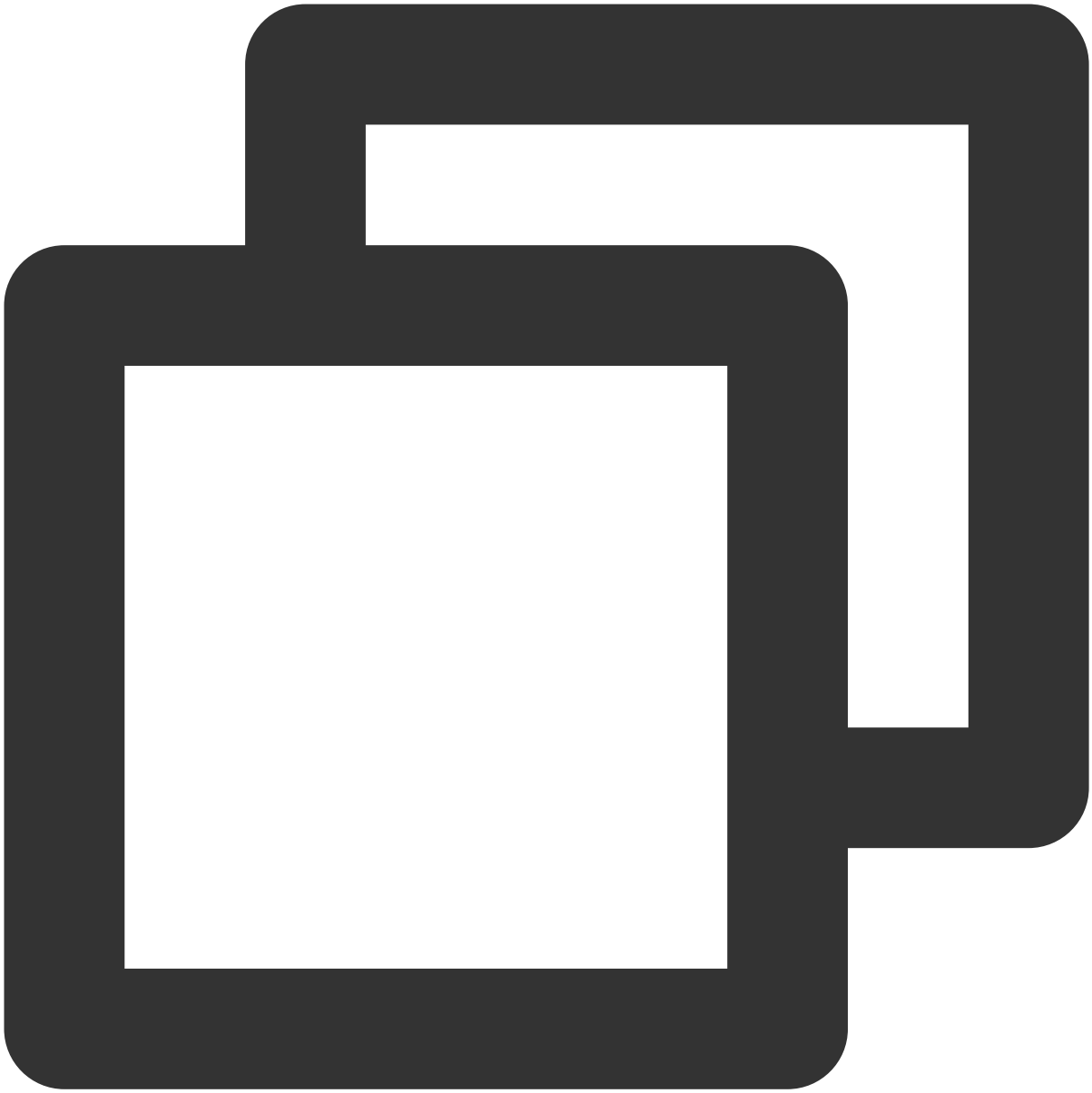


```
include ':tuicallkit-kt'
```

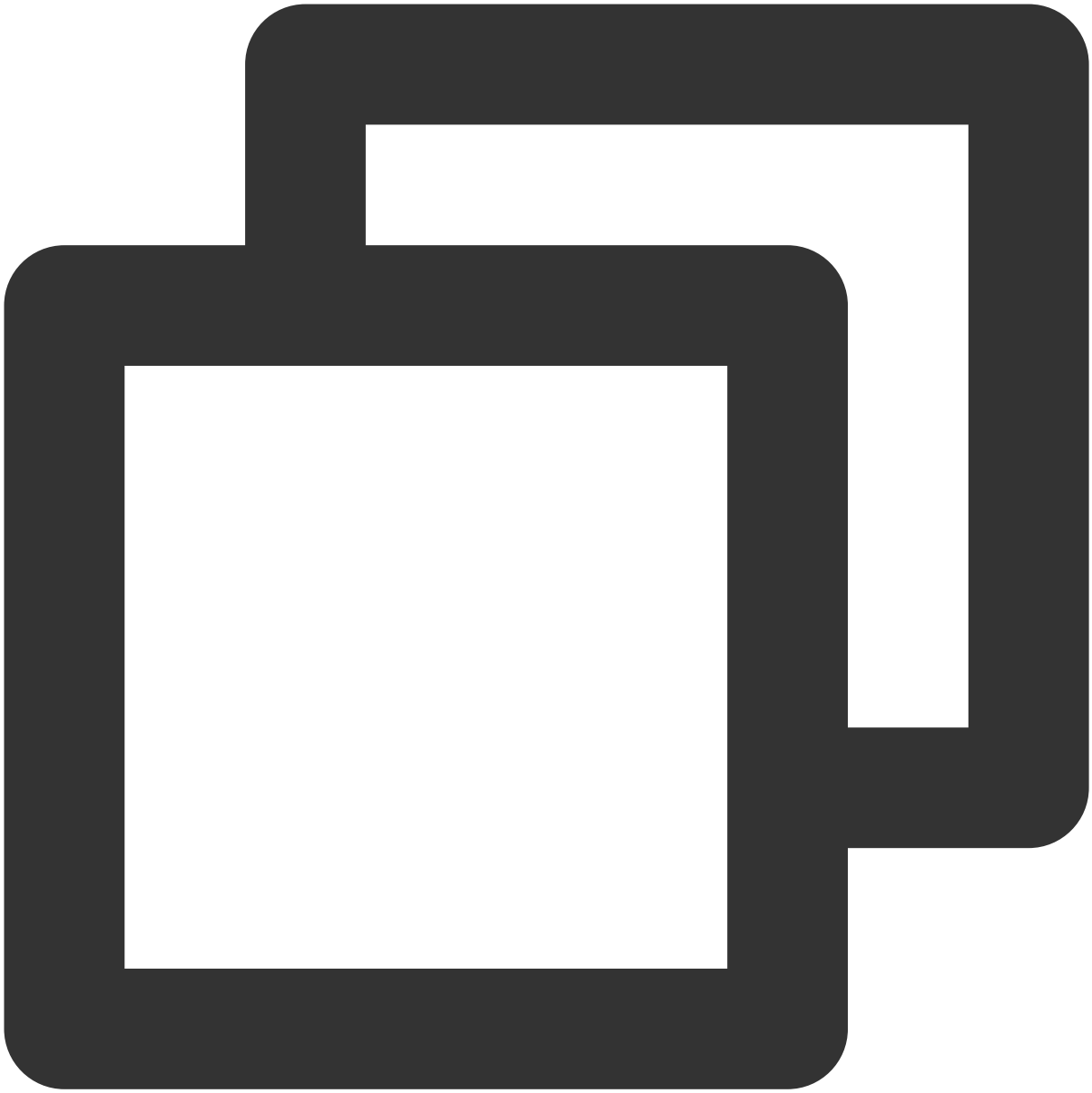
3. 在 app 目录下找到 `build.gradle.kts(或build.gradle)` 文件，在 `dependencies` 中增加如下代码，声明当前 app 对新加入的组件的依赖。

build.gradle.kts

build.gradle



```
dependencies {
    api(project(":tuicallkit-kt"))
}
```



```
dependencies {
    api project(':tuicallkit-kt')
}
```

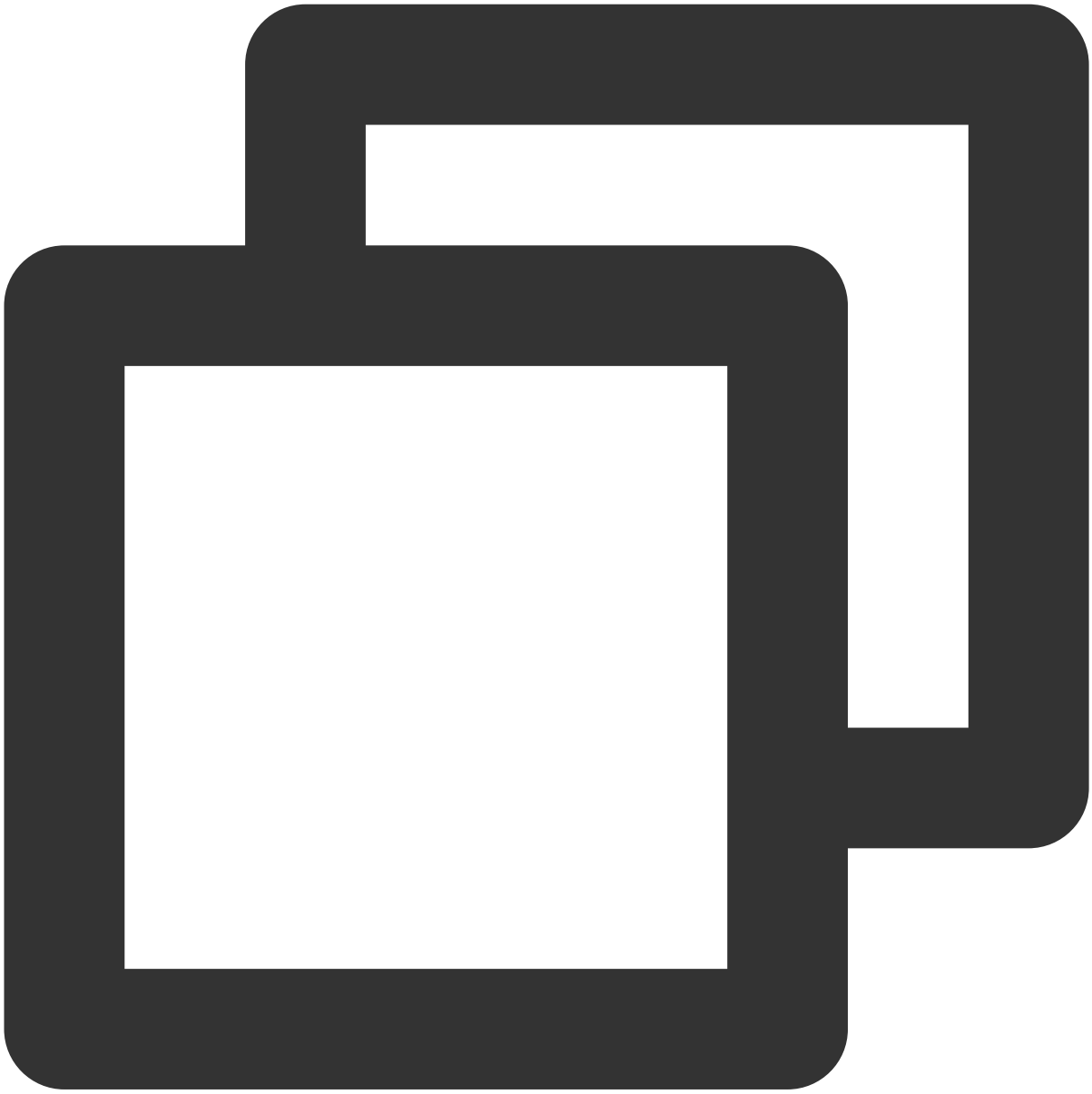
第二步：新增语言包

以西班牙语为例：

1. 新增西班牙语文件。

进入到 `TUICallKit` 源码文件目录下的 `src/main/res` 目录下，新增 `value-es/strings.xml` 文件。

2. 将 `src/main/res/values-en/strings.xml` 中的内容复制到新增的 `src/main/res/values-es/strings.xml` 文件中。
3. 将 `src/main/res/values-es/strings.xml` 中的英文翻译为西班牙语。
4. 新增语言。



```
    ……
import com.tencent.qcloud.tuicore.TUIThemeManager;

public class MainActivity extends BaseActivity {
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Locale locale = new Locale("es");
        TUIThemeManager.addLanguage("es", locale);
        ……
    }
    ……
}
```

Flutter

最近更新时间：2024-08-15 11:22:55

支持的语言

目前支持**简体中文、英文和日文，**默认语言为 **英文 。**

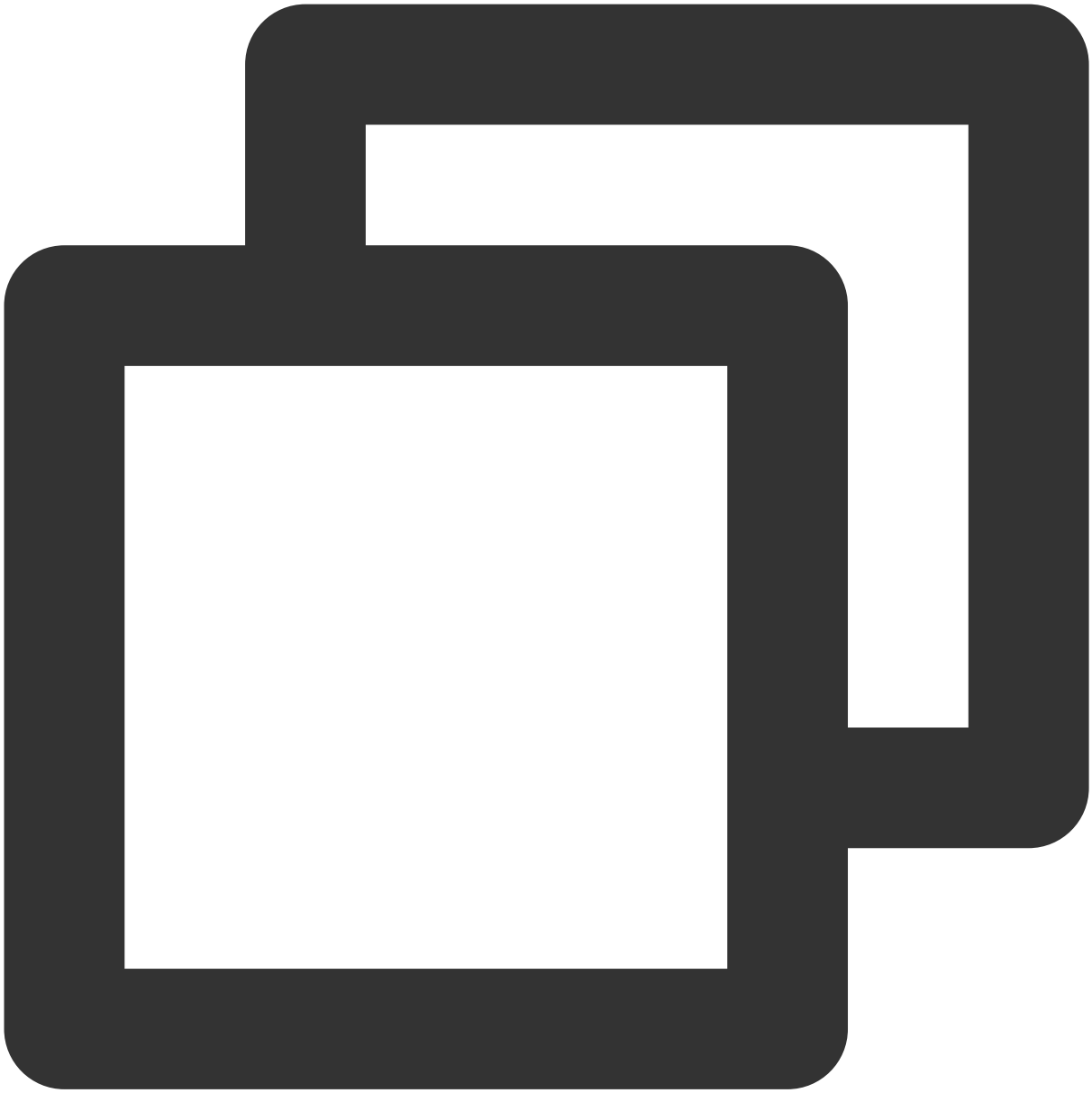
切换语言

`TUICallKit` 不单独提供语言切换的接口， `TUICallKit` 根据当前 `Application` 的 `MaterialApp` （或 CupertinoApp 等风格组件） 使用的语言自适切换，切换 `MaterialApp` （或 CupertinoApp 等风格组件） 使用的语言即可。

添加新的语言

第一步：源码集成

1. 下载源码
进入 `https://pub.dev/packages/tencent_calls_uikit` 下载最新 `TUICallKit` 源码。
2. 依赖本地源码
在 `Application` 工程的 `pubspec.yaml` 文件中修改 `TUICallKit` 为本地依赖：



```
dependencies:
  tencent_calls_uikit:
    path: /TUICallKit 本地路径/
```

第二步：新增语言包

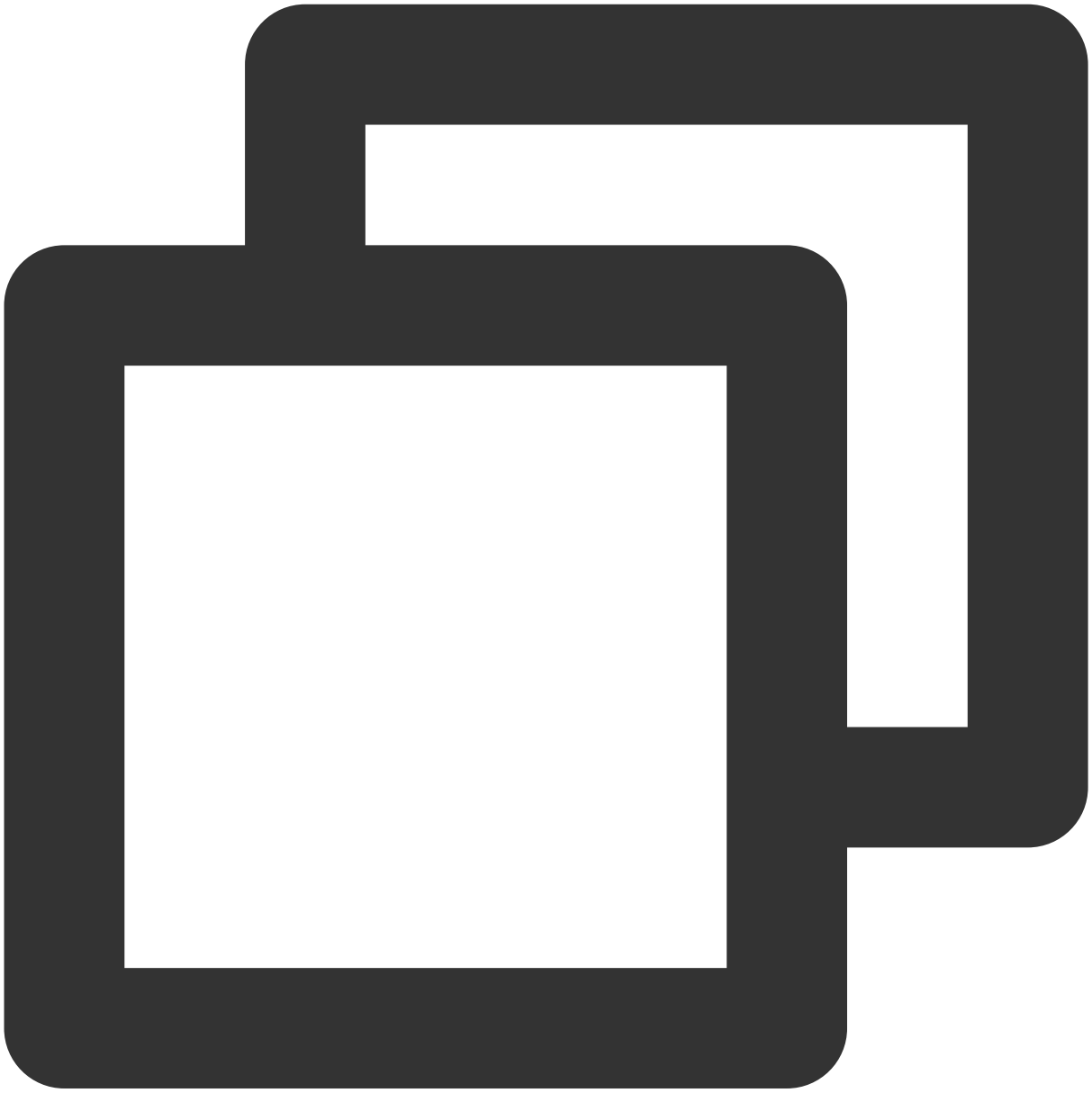
以西班牙语为例：

1. 新增西班牙语文件。

进入到 `TUICallKit` 源码文件目录下的 `lib/src/i18n` 目录下，新增 `strings_es.i18n.json` 。

2. 将 `lib/src/i18n/strings.i18n.json` 中的内容复制到新增的 `lib/src/i18n/strings_es.i18n.json` 文件中。
3. 将 `lib/src/i18n/strings_es.i18n.json` 中的英文翻译为西班牙语。
4. 更新翻译包

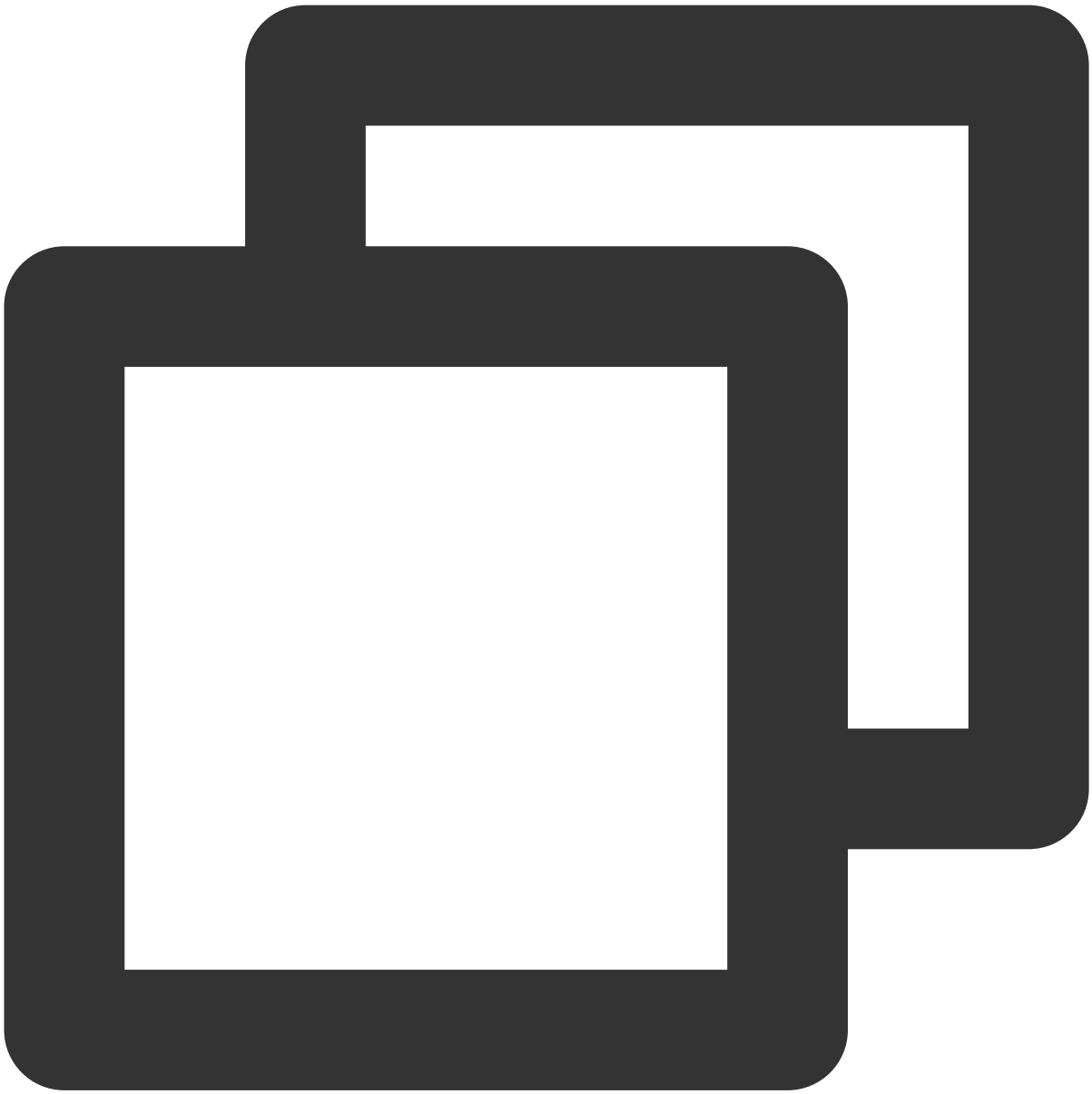
在 `TUICallKit` 源码文件目录下进入命令行工具，执行以下命令更新翻译包：



```
flutter pub add fast_i18n
flutter pub run fast_i18n
```

5. 更新 `TUICallKit` 自适应设置语言方法。

进入到 `lib/src/i18n/i18n_utils.dart` 源码文件中修改 `setLanguage` 方法，修改如下：



```
static setLanguage(Locale currentLocale) {
switch (currentLocale.languageCode) {
  case 'zh':
    {
      CallKitI18nUtils(null, 'zh');
      break;
    }
  case 'en':
```

```
      {
        CallKitI18nUtils(null, 'en');
        break;
      }
    case 'ja':
      {
        CallKitI18nUtils(null, 'ja');
        break;
      }
    // 新增 case 'es'
    case 'es':
      {
        CallKitI18nUtils(null, 'es');
        break;
      }
  }
}
```

uni-app（Android&iOS）

最近更新时间：2024-08-15 11:22:55

支持的语言

目前支持**简体中文、英文、日文和阿拉伯语。**

切换语言

TUICallKit 语言与手机系统保持一致 。

添加新的语言

uni-app（客户端）暂不支持新增语言。

如果您有新增语言的需求，请通过邮件进新反馈： info_rtc@tencent.com 。

服务端 API（TUICallKit）
通话状态回调
通话状态回调

最近更新时间：2024-04-18 16:25:19

为了方便您精细化控制您的通话业务，音视频通话 Call（TUICallKit）提供了通话状态回调。您的业务后台可以通过该回调实时查看用户的通话结果，如未接听、拒接等，您可以据此进行实时数据统计等操作。回调配置方式参见 回调配置 API 列表。

使用条件

仅开通了音视频通话 Call（TUICallKit） **群组通话版**的应用（SDKAppId）可以使用通话状态回调，您也可以开通体验版免费试用，版本说明及开通指引参见 音视频通话能力版本。
目前各平台仅在特定版本的 TUICallKit 可使用通话状态回调，详见下表：

平台/框架	版本号
Android/iOS/Flutter/uni-app（客户端）	≥ 1.7.1
Web	≥ 1.4.6
微信小程序	≥ 1.5.1

注意：
参与通话的所有平台/框架升级到上述新的版本后，才可在控制台上查看到对应的通话信息。

注意事项

要启用回调，必须配置回调 URL，并打开本条命令字的开关，请参见 创建回调配置。
回调的方向是 Callkit 后台向 App 后台发起 HTTP POST 请求。
App 后台在收到回调请求之后，务必校验请求 URL 中的参数 SDKAppID 是否是自己的 SDKAppID。

可能触发该回调的场景

App 用户通过客户端进行通话所产生的动作，例如挂断等。

版权所有：腾讯云计算（北京）有限责任公司

回调的发生时机

通话结束后。

可能发生的回调结果

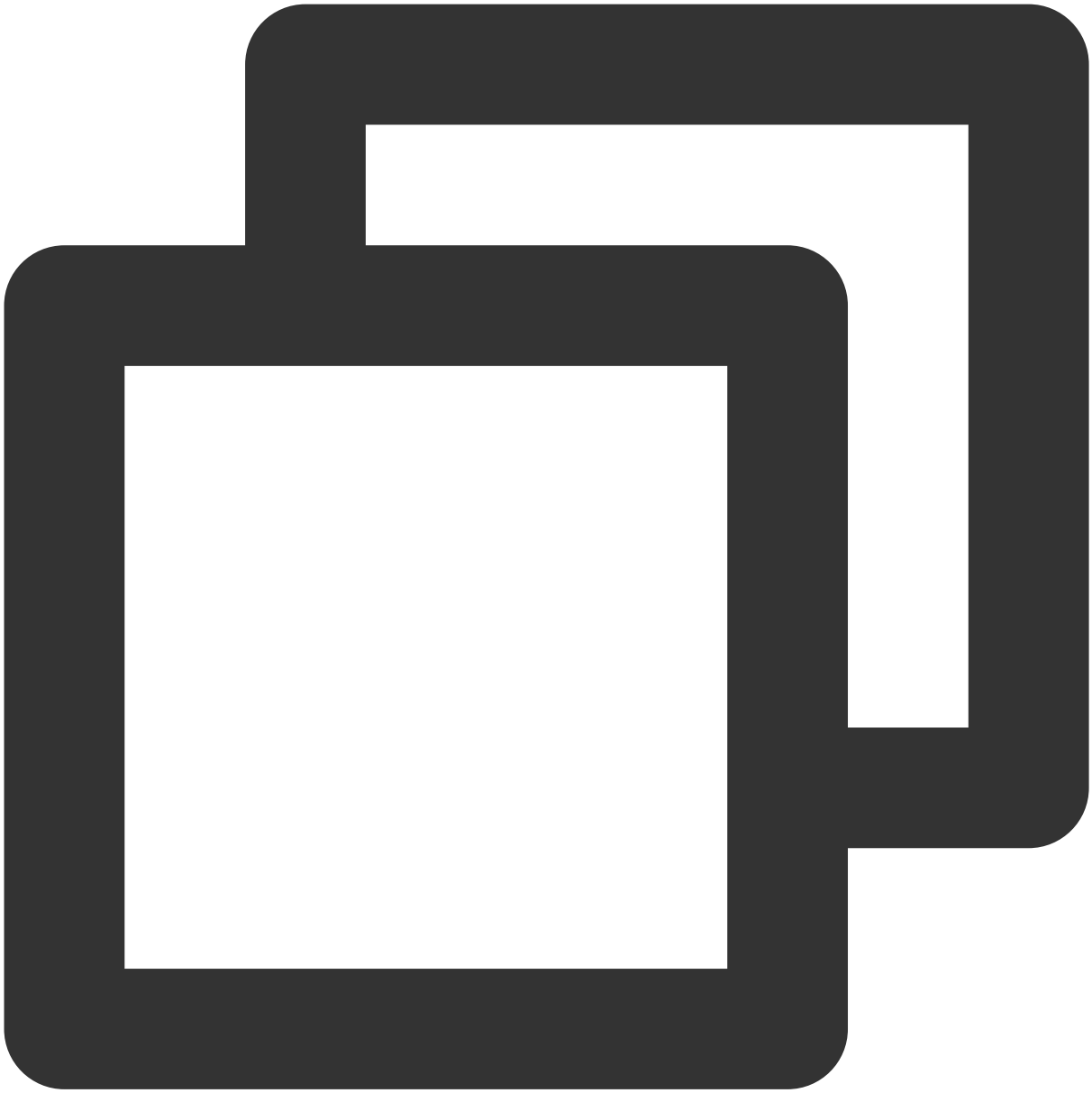
回调状态	结果表示
未接听：被叫方超时未接听	not_answer
拒接：被叫方拒接	reject
忙线：通话忙线	call_busy
取消：主叫方在接通前取消通话	cancel
完成：通话接通并正常结束	normal_end
中断：网络等原因导致通话中断	interrupt

接口说明

请求URL

以下示例中 App 配置的回调 URL 为 `https://www.example.com` 。

示例：



```
$http://www.example.com?sdkappid=$sdkappid&command=$command&contenttype=json&client
```

请求参数说明

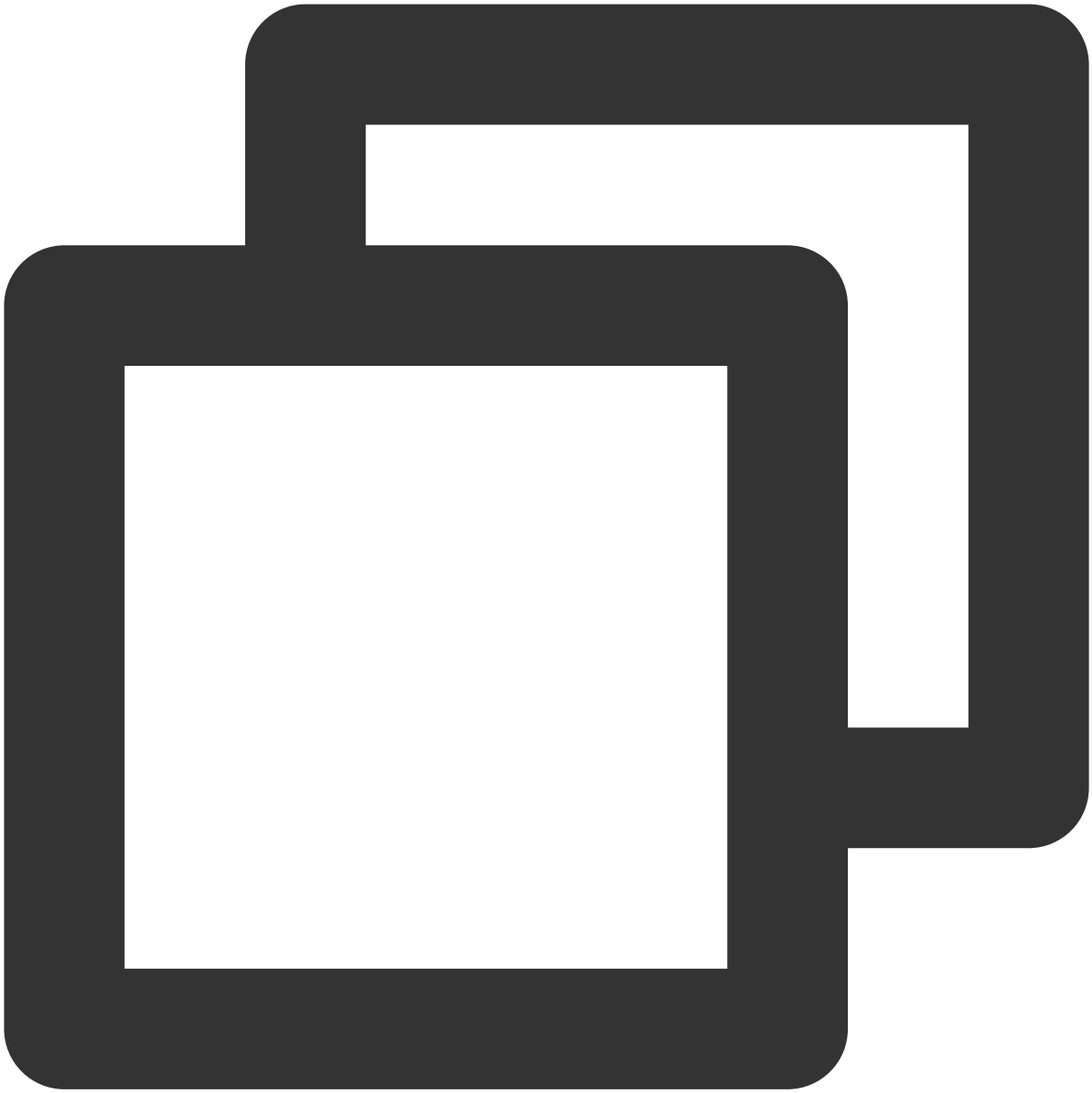
参数	说明
http	请求协议为 HTTPS 或 HTTP，请求方式为 POST
www.example.com	回调 URL

sdkappid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
command	请参见：回调命令字列表
contenttype	固定值为 json
clientip	客户端 IP，格式如：127.0.0.1
optplatform	客户端平台，可能是 iOS、Android、Web、miniProgram

具体的回调内容在 HTTP 请求包体中，参见下文回调示例。

回调示例

回调请求示例：



```
POST /?sdkappid=8888888&command=call_end&contenttype=json&clientip=127.0.0.1&optpla
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
        "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
```

```
    "CallResult": "normal_end",
        "EventTime": 170485688,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

请求包字段说明

字段	类型	说明
UserId	String	操作的用户 ID
RoomId	String	操作的房间 ID
TotalNum	Integer	参与通话人数
MediaType	String	媒体类型： `audio` 音频通话 `video` 视频通话
CallType	String	通话类型： `single` 音频通话 `group` 视频通话
CallId	String	通话唯一 ID
Role	String	角色： `caller` 主叫 userId `callee` 被叫 userId
CallResult	String	通话结果，仅在结束事件中填写，其余为空： `cancel` 取消：主叫方在接通前取消通话 `reject` 拒接：被叫方拒接 `not_answer` 未接听：被叫方超时未接听 `normal_end` 完成：通话接通并正常结束 `call_busy` 忙线：通话忙线 `interrupt` 中断：网络等原因导致通话中断
EventTime	Integer	操作发生的时间戳（秒级）
StartCallTs	Integer	通话发起的时间戳（秒级）只有在normal_end才会回调此值
AcceptTs	Integer	通话接听的时间戳（秒级）只有在normal_end才会回调此值
EndTs	Integer	通话结束的时间戳（秒级）只有在normal_end才会回调此值

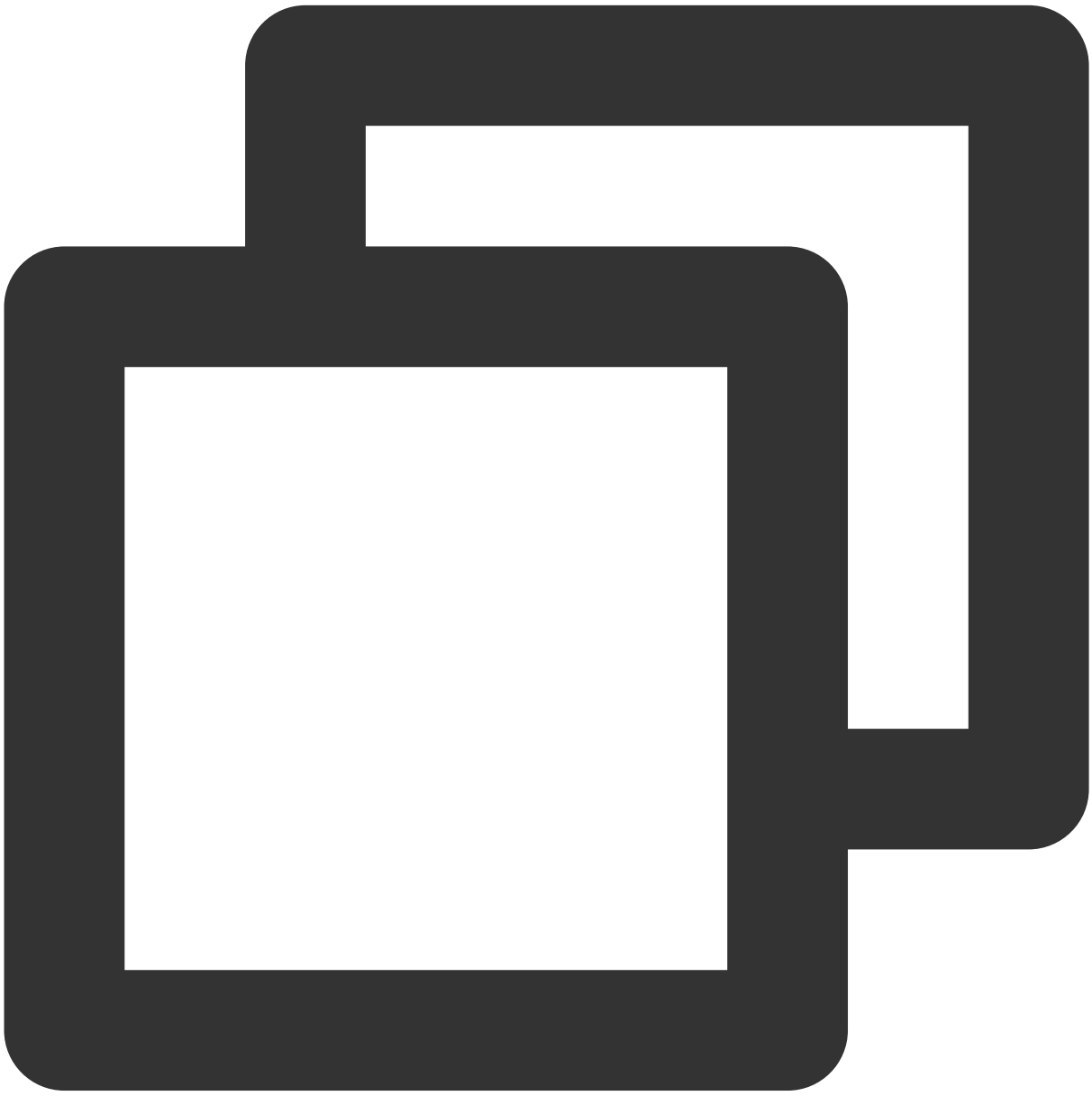
版权所有：腾讯云计算（北京）有限责任公司

说明：

CallResult 字段仅一对一通话可能显示所有类型，群聊仅有 `normal_end` 。

回调应答示例

App 后台同步数据后，发送回调应答包。



```
{
    "ErrorCode": 0,
    "ErrorMessage": "Success"
}
```

应答包字段说明

字段	类型	说明
ErrorCode	Integer	0 为成功，其余为失败
ErrorMessage	String	您的服务器响应可以自定义携带错误信息

注意：

您的服务器在正确接收到请求包时，建议您及时回复正确的应答包，否则会触发系统重试，重试机制如下。

重试机制

触发重试的时机

ErrorCode = 0 视为回调成功，否则会触发重试。

向您的 appServer 发 http 请求报错也会触发重试。

重试间隔

重试间隔为 `0s, 1s, 1s, 2s, 3s, 5s` 若均为成功则放弃。

错误码

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
999	回调不存在
70001	请求参数无效，请检查必填参数是否缺失或填错
70002	UserSig 无效
70003	UserSig 过期
70004	请求用户非超级管理员
70005	请求被限频
未知错误码	系统内部错误，请 提交工单 联系技术人员

通话事件回调

最近更新时间：2024-04-18 16:25:49

为了方便您精细化控制您的通话业务，音视频通话 Call（TUICallKit）提供了通话状态回调。您的业务后台可以通过该回调实时查看用户的通话结果，如未接听、拒接等，您可以据此进行实时数据统计等操作。回调配置方式参见 回调配置 API 列表。

使用条件

仅开通了音视频通话 Call（TUICallKit） **群组通话版**的应用（SDKAppId）可以使用通话状态回调，您也可以开通体验版免费试用，版本说明及开通指引参见 音视频通话能力版本。
目前各平台仅在特定版本的 TUICallKit 可使用通话状态回调，详见下表：

平台/框架	版本号
Android/iOS/Flutter/uni-app（客户端）	≥ 1.7.1
Web	≥ 1.4.6
微信小程序	≥ 1.5.1

注意：
参与通话的所有平台/框架升级到上述新的版本后，才可在控制台上查看到对应的通话信息。

注意事项

要启用回调，必须配置回调 URL，并打开本条命令字的开关，请参见 创建回调配置。
回调的方向是 Callkit 后台向 App 后台发起 HTTP POST 请求。
App 后台在收到回调请求之后，务必校验请求 URL 中的参数 SDKAppID 是否是自己的 SDKAppID。

可能触发该回调的场景

App 用户通过客户端进行通话所产生的动作，例如接听、挂断等。

回调的发生时机

接听、挂断等用户操作后。

可能发生的回调结果

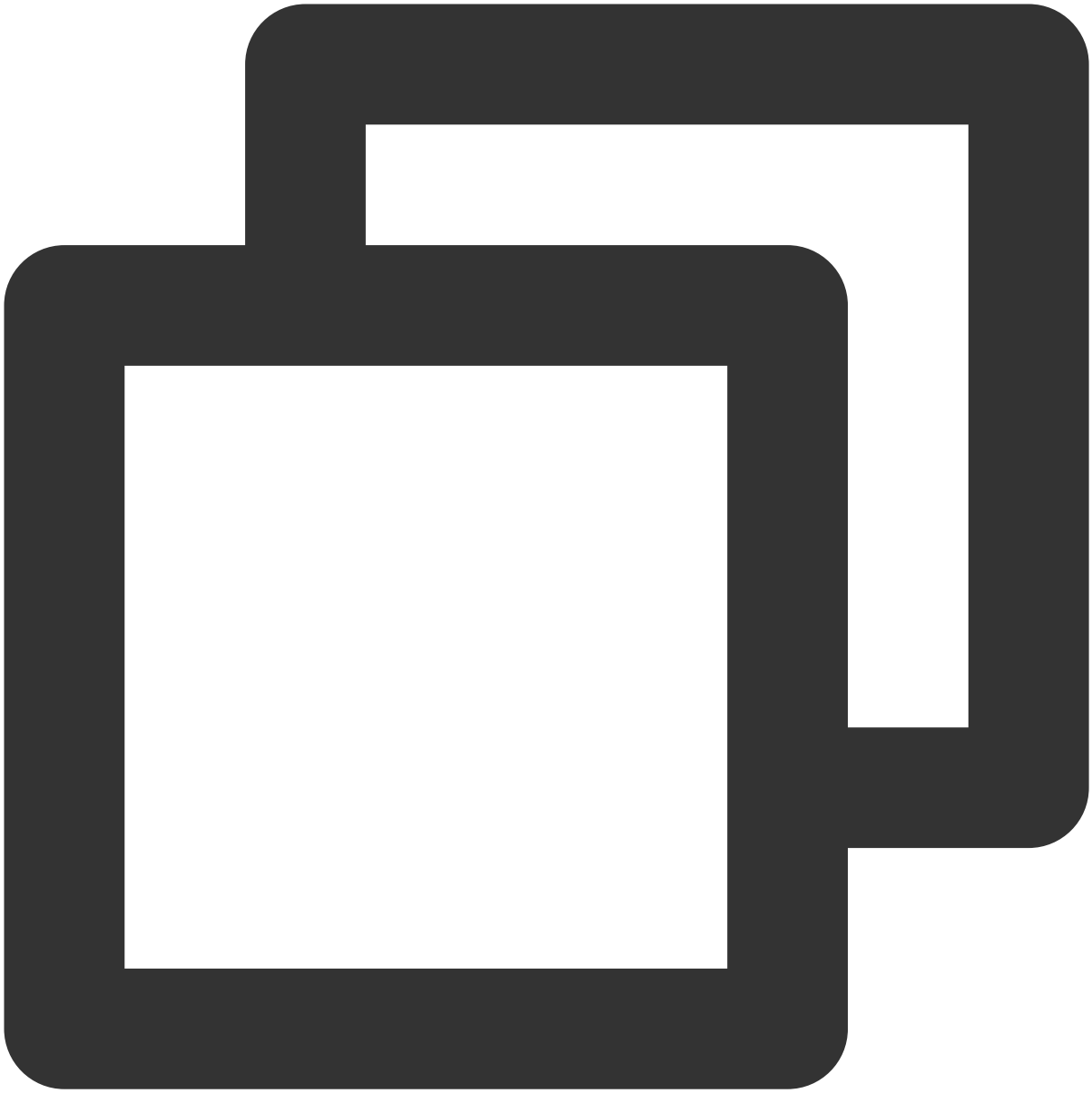
请参见 回调命令字-通话事件回调。

接口说明

请求URL

以下示例中 App 配置的回调 URL 为 `https://www.example.com` 。

示例：



```
$http://www.example.com?sdkappid=$sdkappid&command=$command&contenttype=json&client
```

请求参数说明

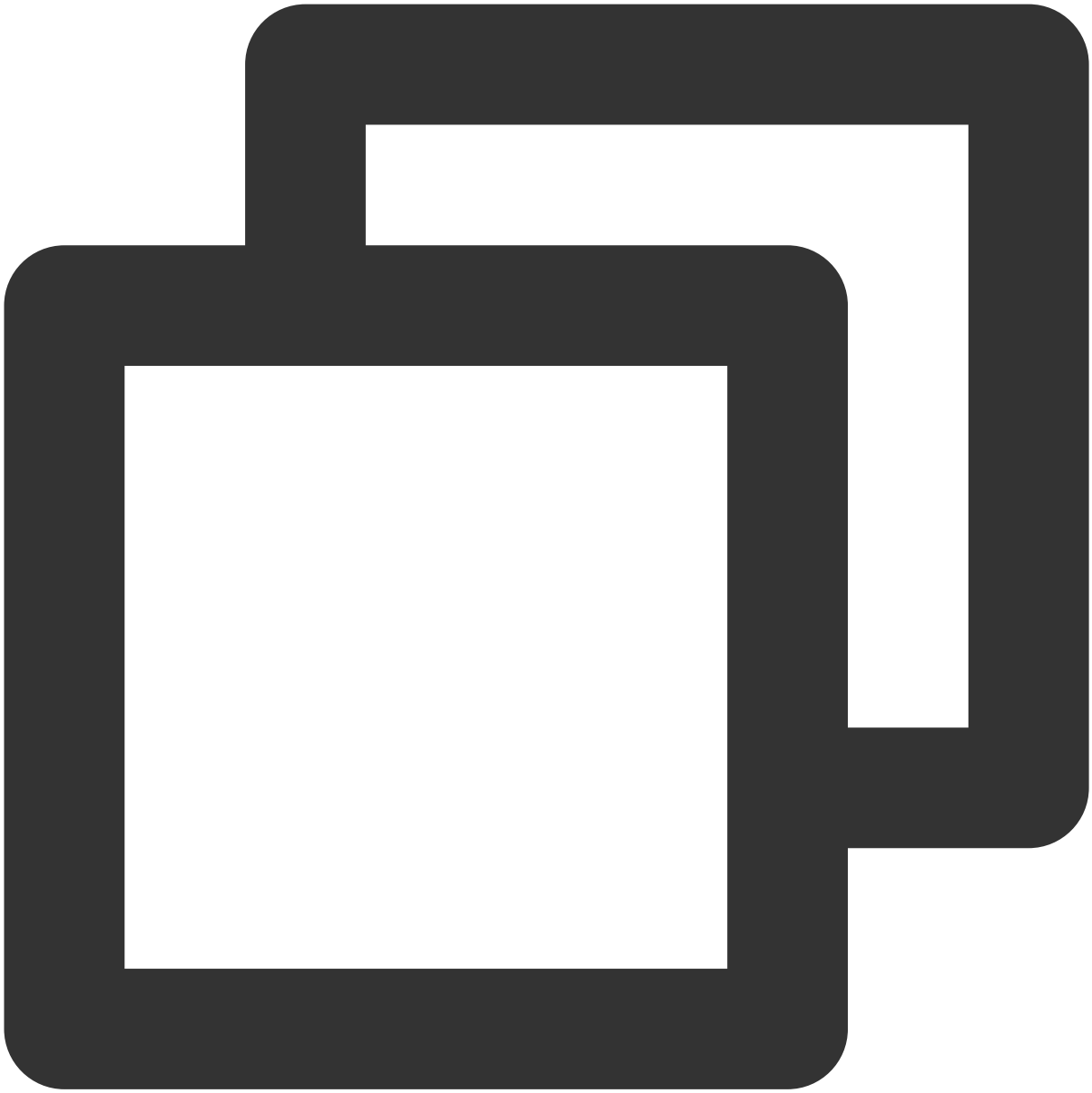
参数	说明
http	请求协议为 HTTPS 或 HTTP，请求方式为 POST
www.example.com	回调 URL

sdkappid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
command	请参见：回调命令字列表
contenttype	固定值为 json
clientip	客户端 IP，格式如：127.0.0.1
optplatform	客户端平台，可能是 iOS、Android、Web、miniProgram

具体的回调内容在 HTTP 请求包体中，参见下文回调示例。

回调示例

回调请求示例：



```
POST /?sdkappid=8888888&command=caller_start_call&contenttype=json&clientip=127.0.0
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
        "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
```

版权所有：腾讯云计算（北京）有限责任公司

```
    "Event": "start_call",
    "CallResult": "",
        "EventTime": 1704695566,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

请求包字段说明

字段	类型	说明
UserId	String	操作的用户 ID
RoomId	String	操作的房间 ID
TotalNum	Integer	参与通话人数
CallType	String	通话类型： `single` 音频通话 `group` 视频通话
CallId	String	通话唯一 ID
Role	String	角色： `caller` 主叫 userId `callee` 被叫 userId
Event	String	通话事件： `start_call` 主叫发起通话 `call_accepted` 主叫接听通话 `call_missed` 主叫未接听 `call_rejected` 主叫拒绝通话 `call_busy` 主叫通话忙线 `cancel_call` 主叫取消通话 `call_failed` 主叫发起通话失败 `call_end` 主叫通话正常结束 `call_interrupted` 主叫通话中断 `receive_call` 被叫接收通话 `accept_call` 被叫接听通话 `not_answer_call` 被叫未应答 `reject_call` 被叫拒接 `ignore_call` 被叫忽略通话 `call_canceled` 被叫取消通话 `call_end` 被叫通话正常结束

版权所有：腾讯云计算（北京）有限责任公司

		`call_interrupted` 被叫通话异常中断 `invite_user` 中途邀请用户 `join_in_group_call` 中途加入通话
StartCallTs	Integer	通话发起的时间戳（秒级）只有在normal_end才会回调此值
AcceptTs	Integer	通话接听的时间戳（秒级）只有在normal_end才会回调此值
EndTs	Integer	通话结束的时间戳（秒级）只有在normal_end才会回调此值

说明：

CallResult 字段仅一对一通话可能显示所有类型，群聊仅有 `normal_end` 。

回调配置

回调配置 API 列表

最近更新时间：2024-04-18 16:29:11

回调信息配置

功能说明	接口
创建回调	v1/callback/set
查询回调	v1/callback/get
更新回调	v1/callback/update
删除回调	v1/callback/delete

回调命令字列表

通话状态回调	通话事件回调
`cancel` 取消：主叫方在接通前取消通话 `reject` 拒接：被叫方拒接 `not_answer` 未接听：被叫方超时未接听 `normal_end` 完成：通话接通并正常结束 `call_busy` 忙线：通话忙线 `interrupt` 中断：网络等原因导致通话中断	`caller_start_call` 主叫发起通话 `caller_call_accepted` 主叫接听通话 `caller_call_missed` 主叫未接听 `caller_call_rejected` 主叫拒绝通话 `caller_call_busy` 主叫通话忙线 `caller_cancel_call` 主叫取消通话 `caller_call_failed` 主叫发起通话失败 `caller_call_end` 主叫通话正常结束 `caller_call_interrupted` 主叫通话中断 `callee_receive_call` 被叫接收通话 `callee_accept_call` 被叫接听通话 `callee_not_answer_call` 被叫未应答 `callee_reject_call` 被叫拒接 `callee_ignore_call` 被叫忽略通话 `callee_call_canceled` 被叫取消通话 `callee_call_end` 被叫通话正常结束 `callee_call_interrupted` 被叫通话异常中断 `invite_user` 中途邀请用户

版权所有：腾讯云计算（北京）有限责任公司

	`join_in_group_call` 中途加入通话

创建回调配置

最近更新时间：2024-07-24 10:20:03

功能说明

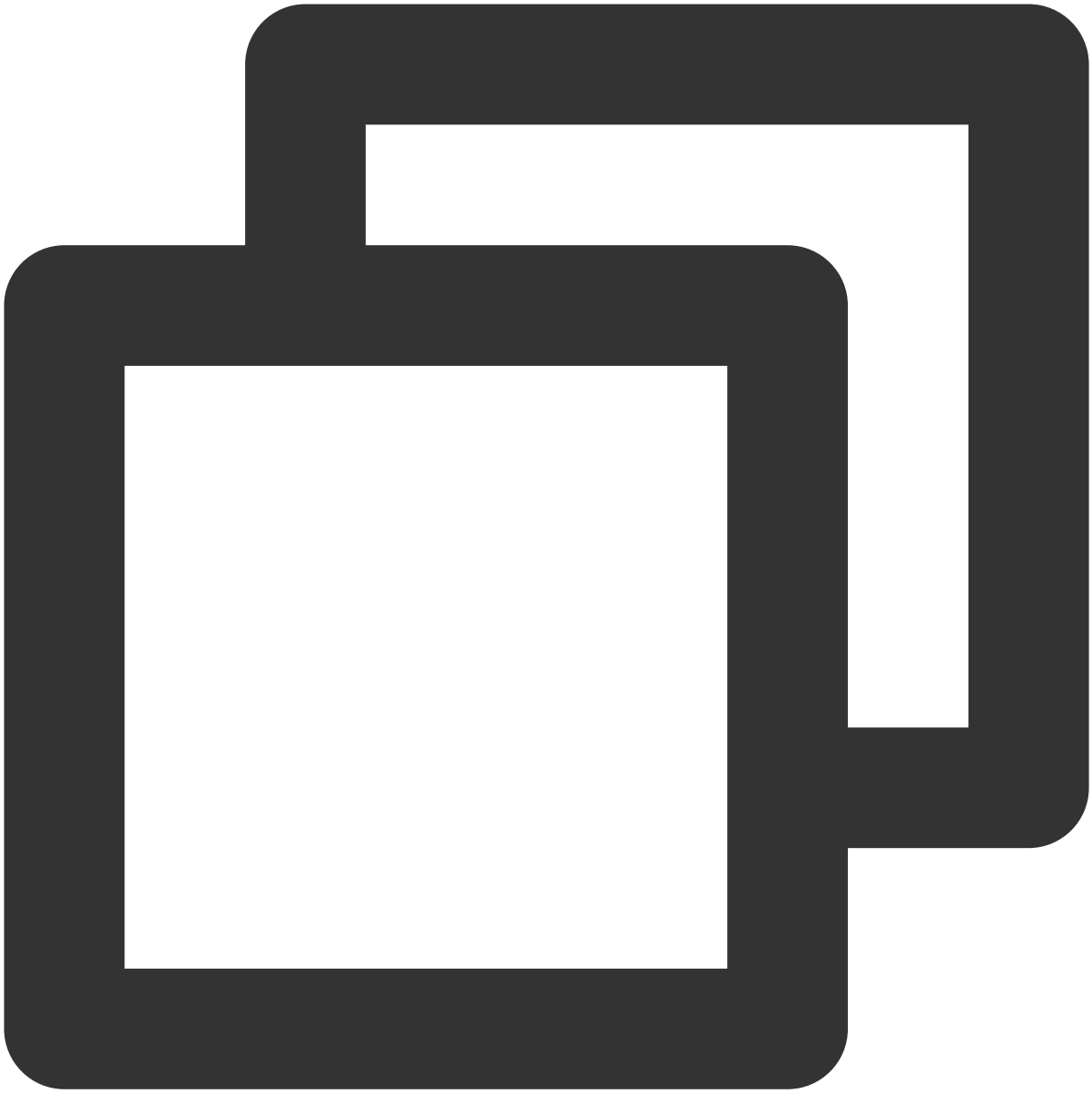
管理员可通过该接口创建回调。

接口说明

注意：

接口多次调用以最后一次的结果为准。

请求URL示列



```
https://xxxxxx/v1/callback/set?sdkappid=88888888&identifier=admin&usersig=xxx&rando
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参考 REST API 简介。

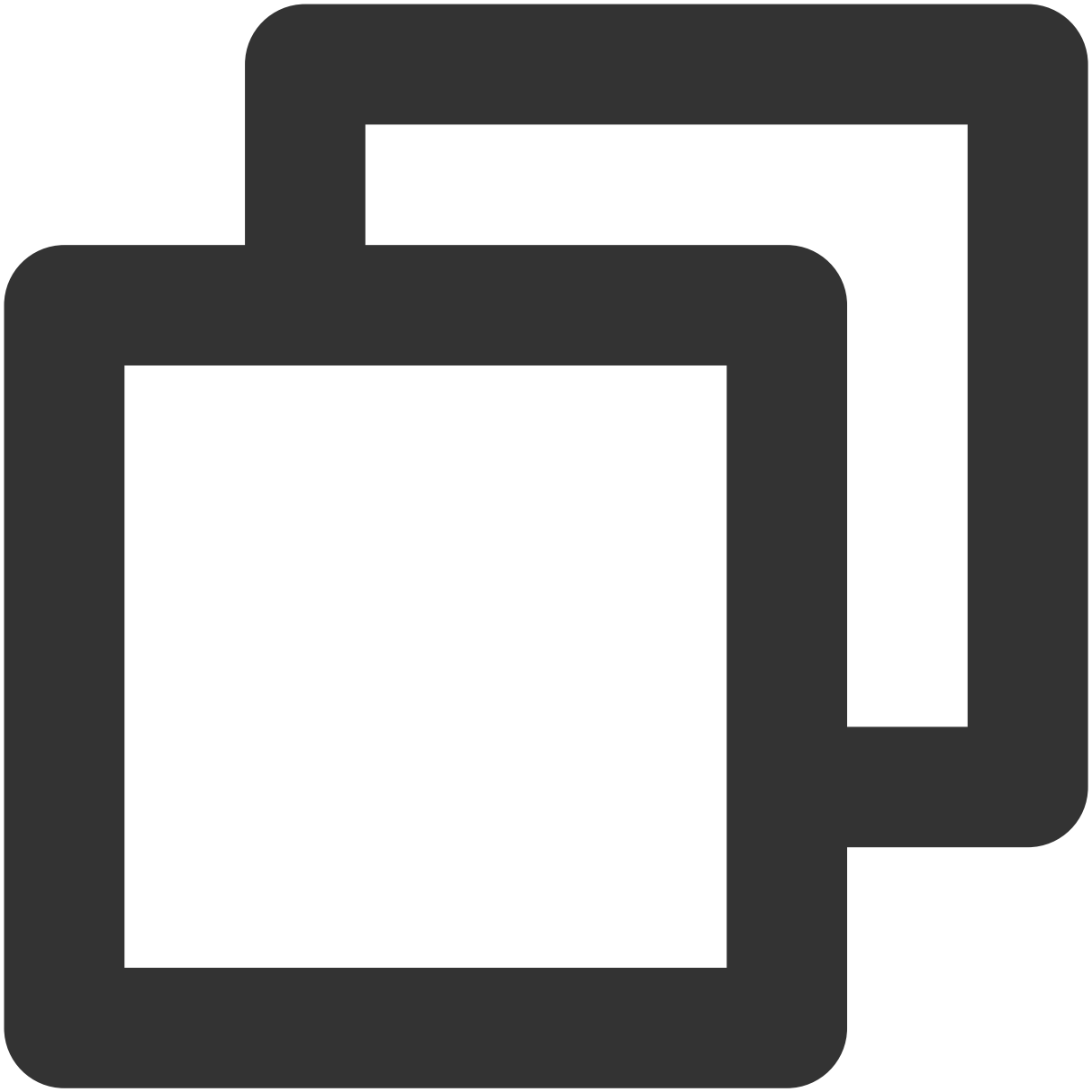
参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名：`callkit-intl.trtc.tencent-cloud.com`

v1/callback/set	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 Chat App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

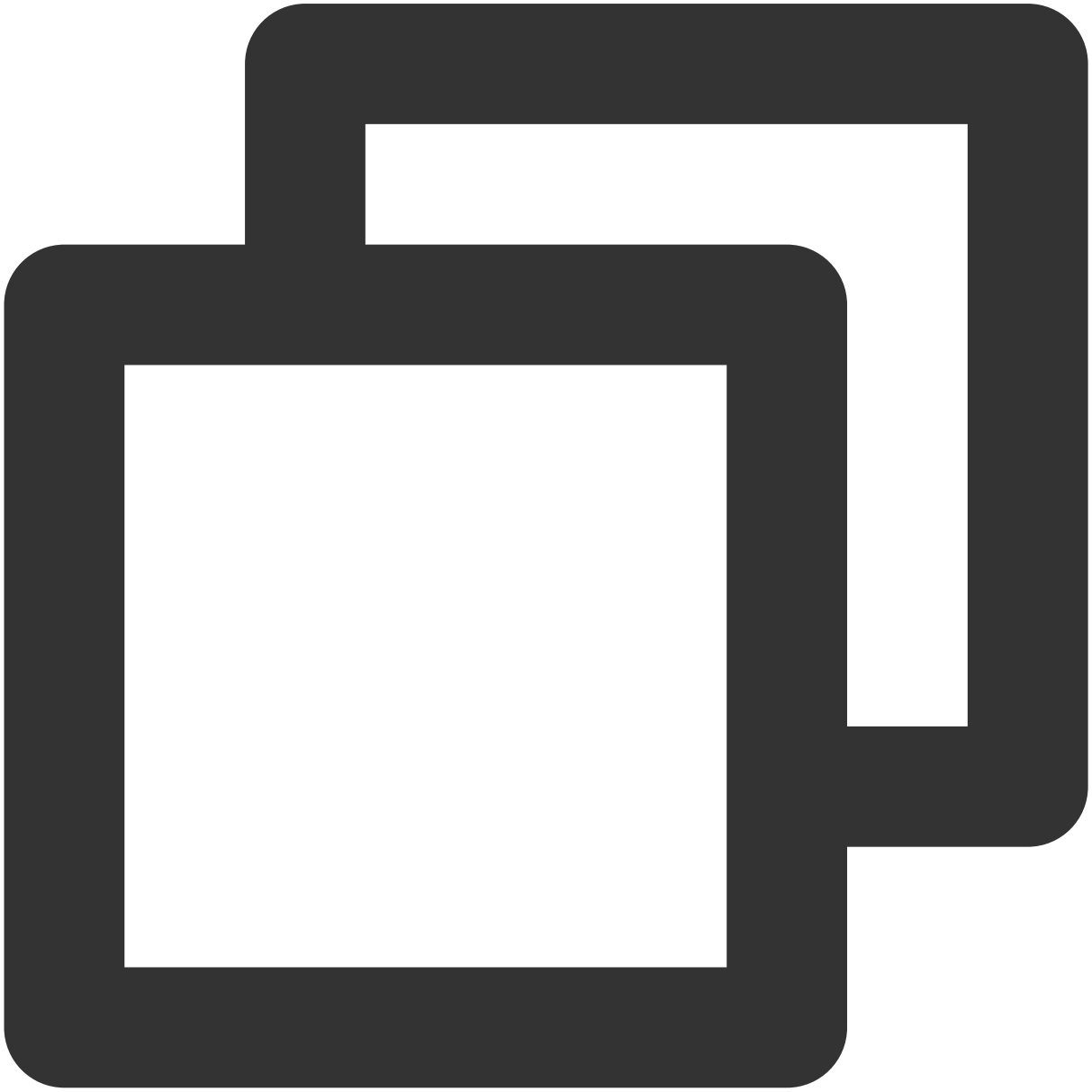


```
{
    "address":"http://www.example.com/callback",
    "actions": [
        "call_busy",
        "normal_end",
        "caller_start_call",
        "invite_user",
        "callee_reject_call"
    ]
}
```

请求字段说明

字段	类型	属性	说明
address	String	必填	回调的地址，需要以 http/https 开头，建议使用更为安全的 https
actions	Array	必填	需要触发回调的情景，列表请参见 回调命令字列表

应答包体示例



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "a1d8543a9b1daef5d0f0c21517a4bc0a",
    "data": "http://www.example.com/callback"
}
```

应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
data	String	配置成功的回调地址

错误码说明

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
70001	回调地址必须 http 或 https 开头

查询回调配置

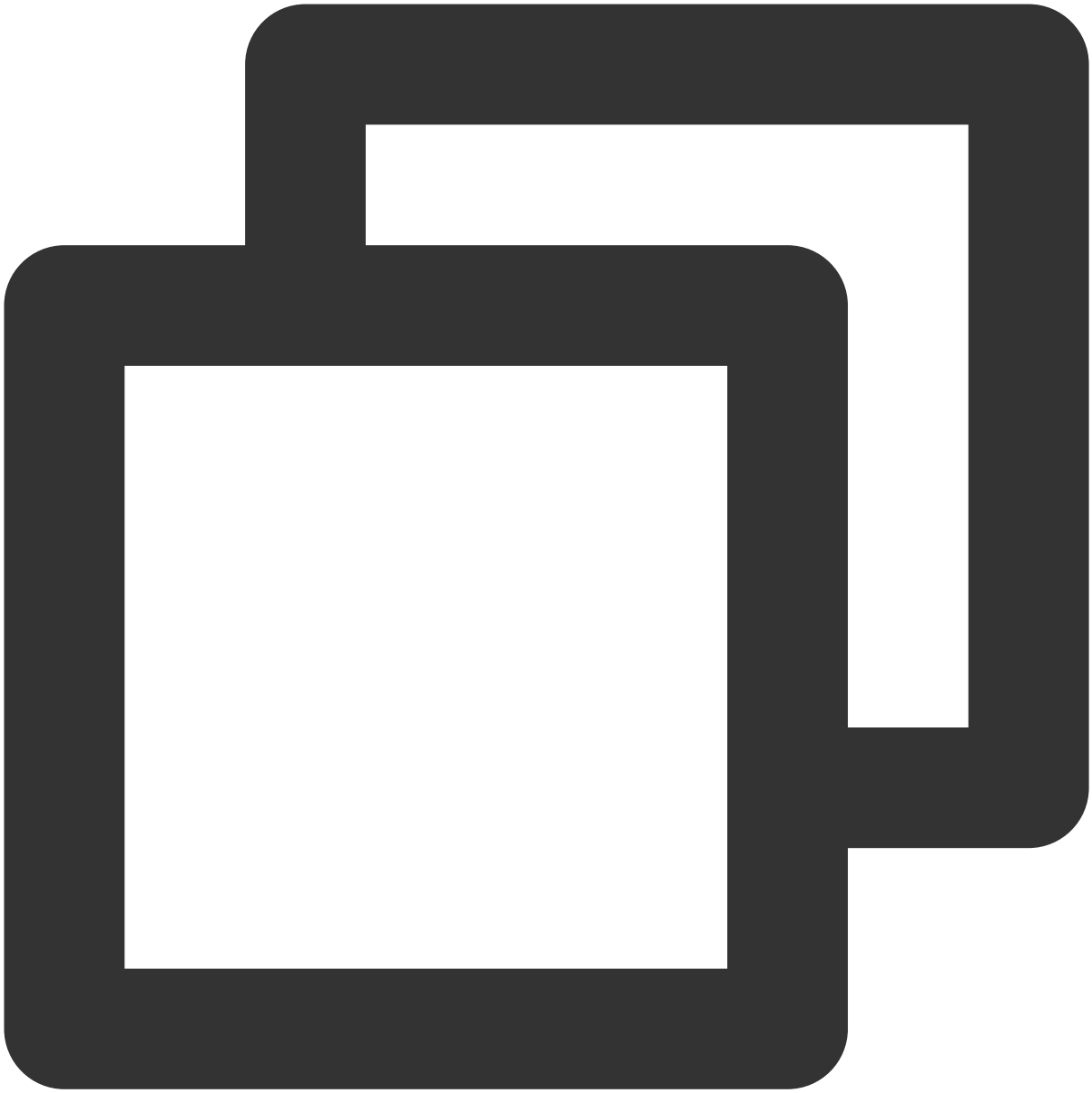
最近更新时间：2024-07-24 10:20:03

功能说明

管理员可通过该接口查询已配置的回调。

接口说明

请求URL示列



```
https://xxxxxx/v1/callback/get?sdkappid=88888888&identifier=admin&usersig=xxx&rando
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参见 REST API 简介。

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： `callkit-intl.trtc.tencent-cloud.com`

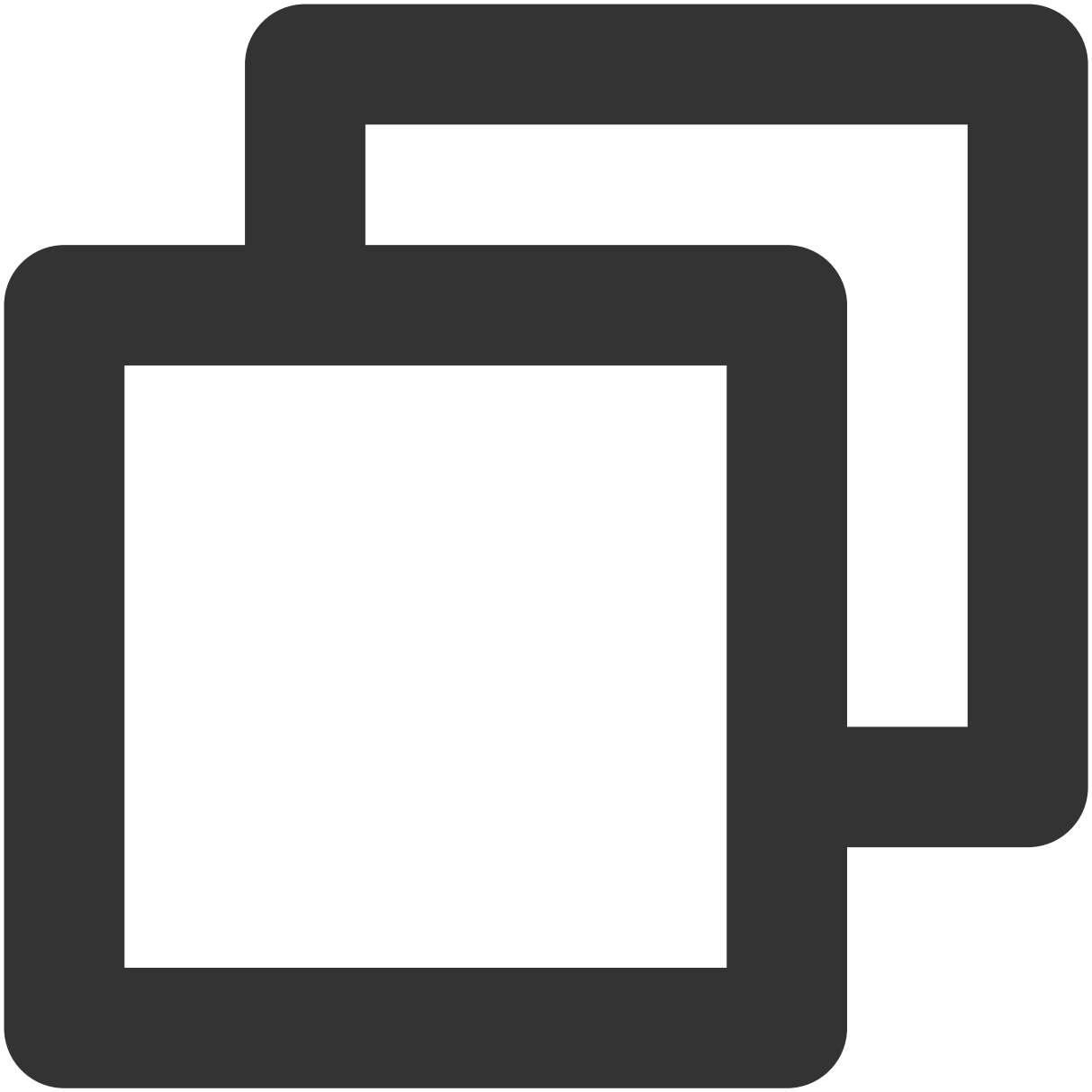
v1/callback/get	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 Chat App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

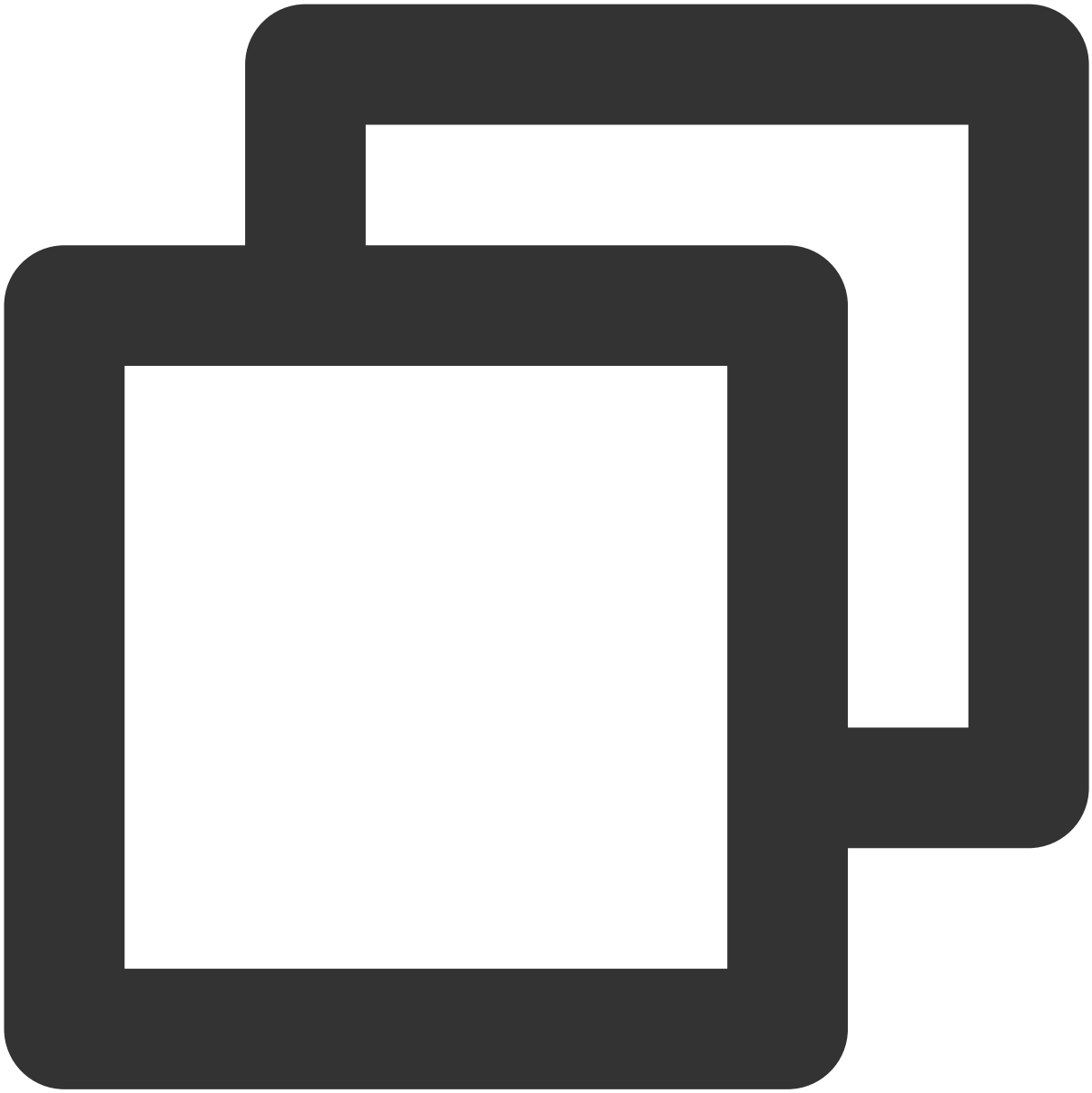
传空即可。



```
{

}
```

应答包体示例



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "355c3c394f0602ed81a13c34999abebb",
    "data": {
        "actions": [
            "call_busy",
            "normal_end"
        ],
        "address": "http://www.example.com/callback"
    }
```

```
}
```

应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
actions	Array	已配置的回调 actions
address	String	已配置的回调地址

错误码说明

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
70001	回调地址必须 http 或 https 开头

更新回调配置

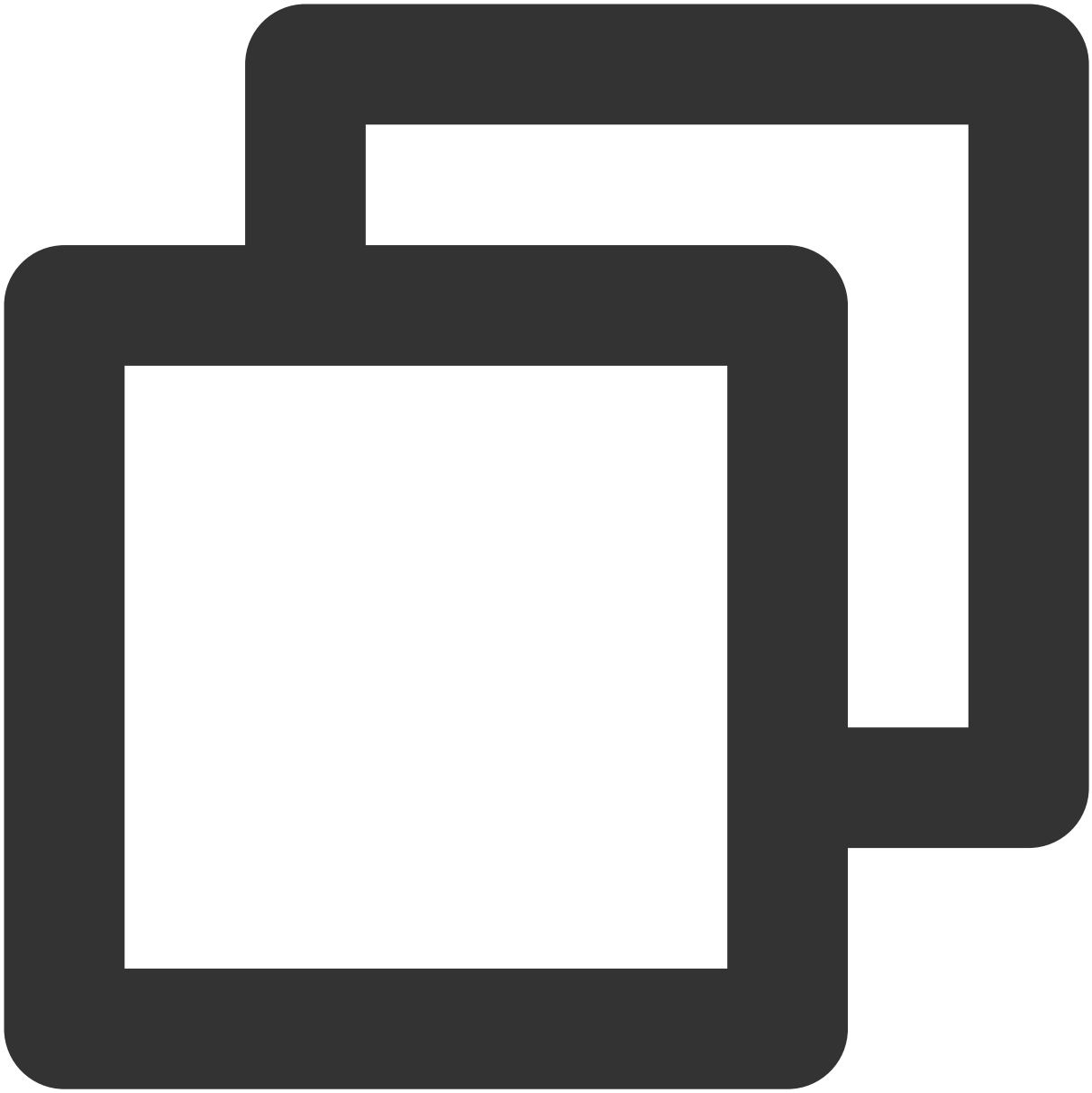
最近更新时间：2024-07-24 10:20:03

功能说明

管理员可通过该接口更新回调。

接口说明

请求URL示列



```
https://xxxxxx/v1/callback/update?sdkappid=88888888&identifier=admin&usersig=xxx&ra
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参见 REST API 简介。

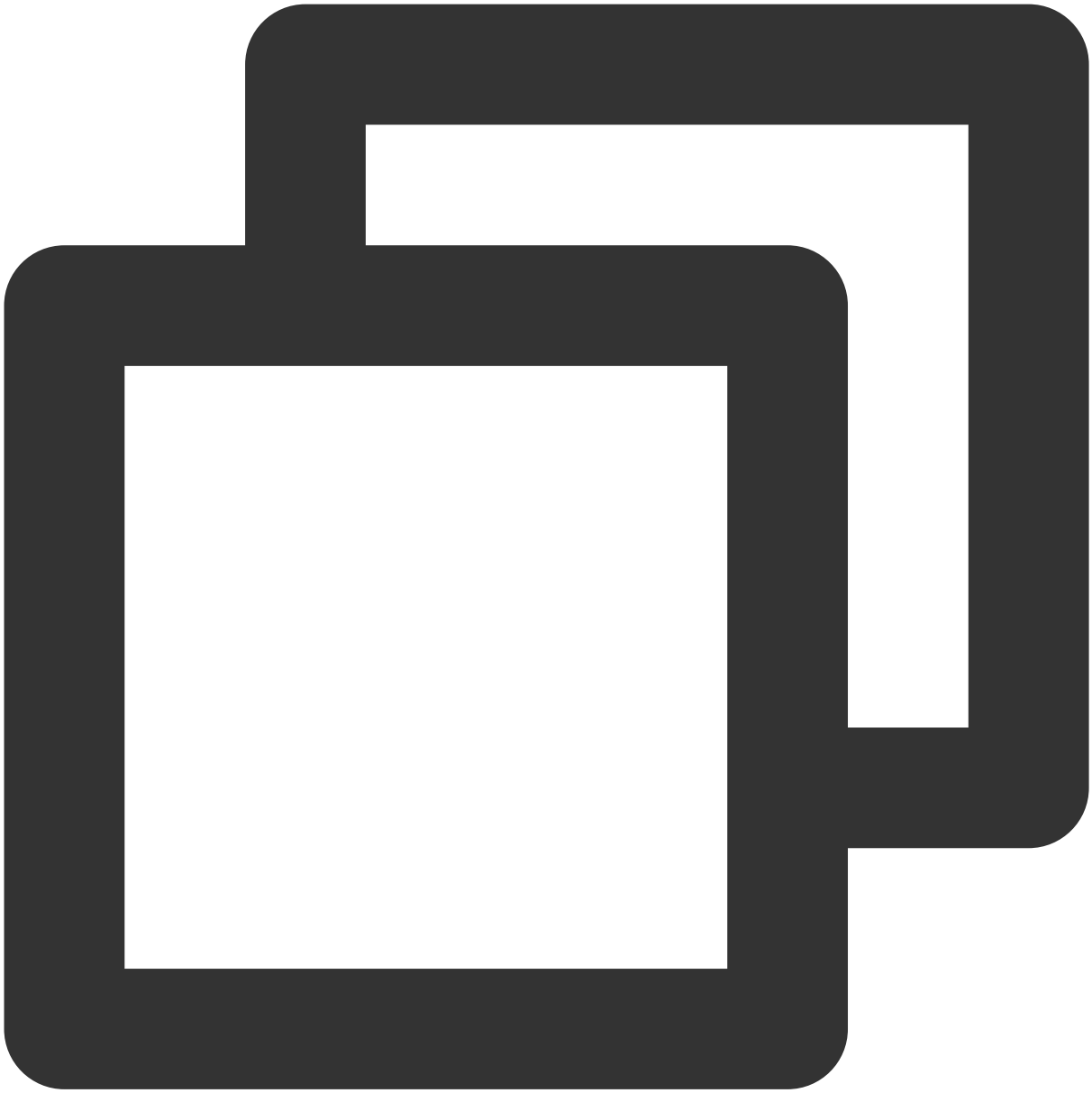
参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名：`callkit-intl.trtc.tencent-cloud.com`

v1/callback/update	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 Chat App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

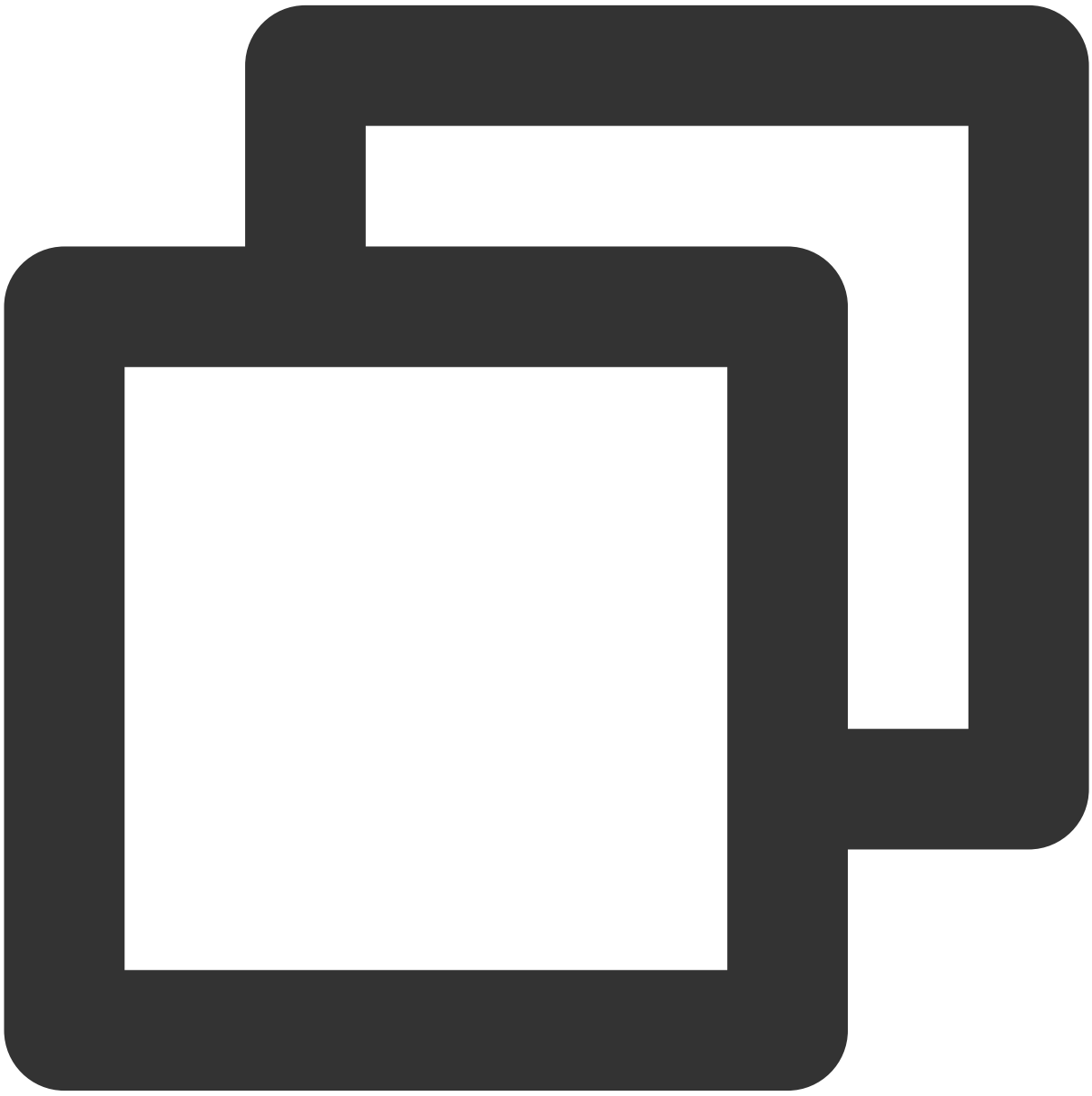


```
{
    "address": "https://www.example2.com/callback",
    "actions": [
        "call_busy",
        "cancel",
        "normal_end"
    ]
}
```

请求字段说明

字段	类型	属性	说明
address	String	必填	回调的地址，需要以 http/https 开头，建议使用更为安全的 https
actions	Array	必填	需要触发回调的情景，列表请参见 回调命令字列表

应答包体示例



```
{
    "errorCode": 0,
```

```
    "errorMessage": "Success",
    "requestId": "5b0fa500064397cad3554506e27e18e1",
    "data": "https://www.example2.com/callback"
}
```

应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
data	String	更新成功的回调地址

错误码说明

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
999	没有配置信息

删除回调配置

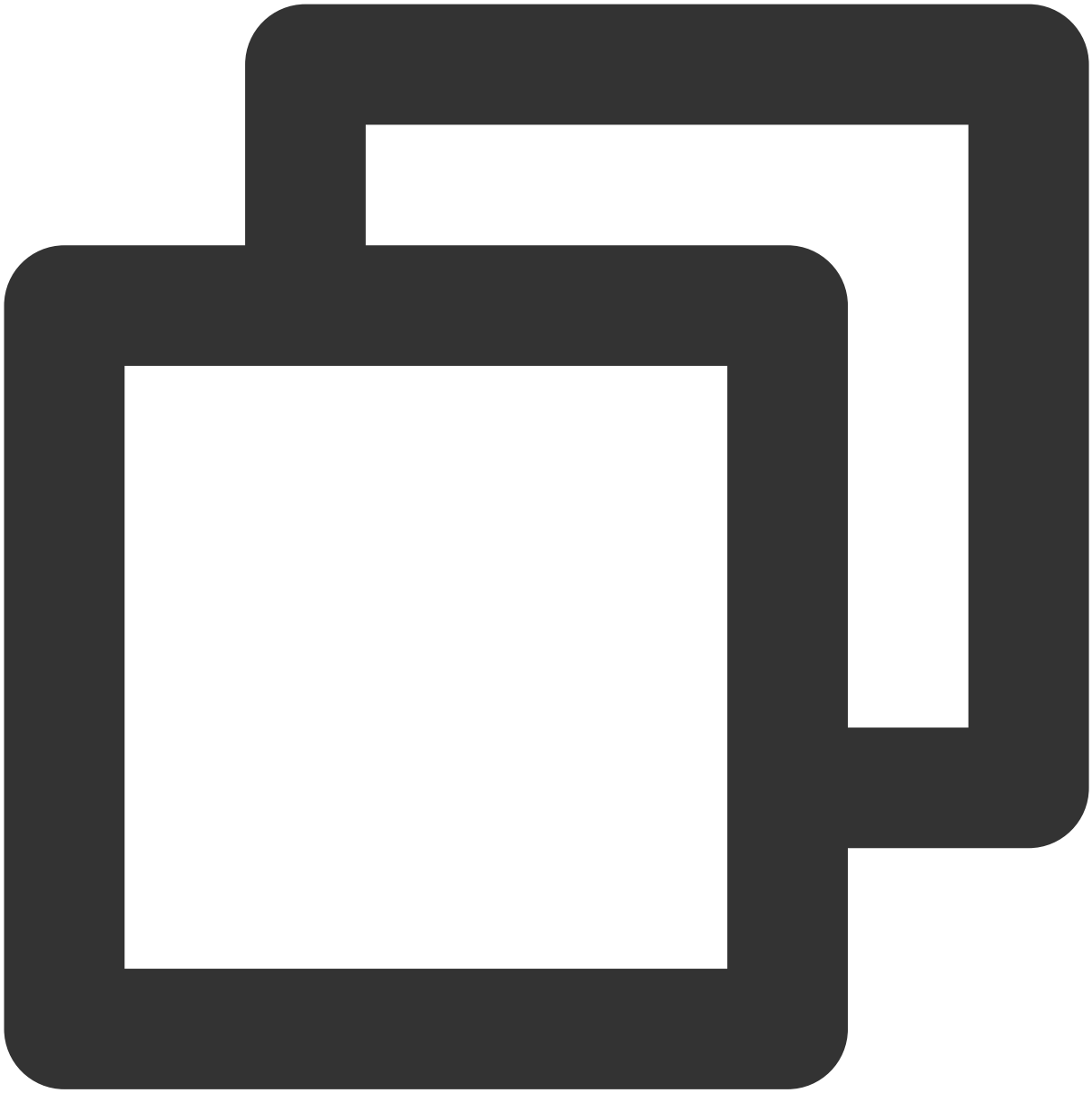
最近更新时间：2024-07-24 10:20:03

功能说明

管理员可通过该接口删除回调。

接口说明

请求URL示列



```
https://xxxxxx/v1/callback/delete?sdkappid=88888888&identifier=admin&usersig=xxx&ra
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参见 REST API 简介。

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： `callkit-intl.trtc.tencent-cloud.com`

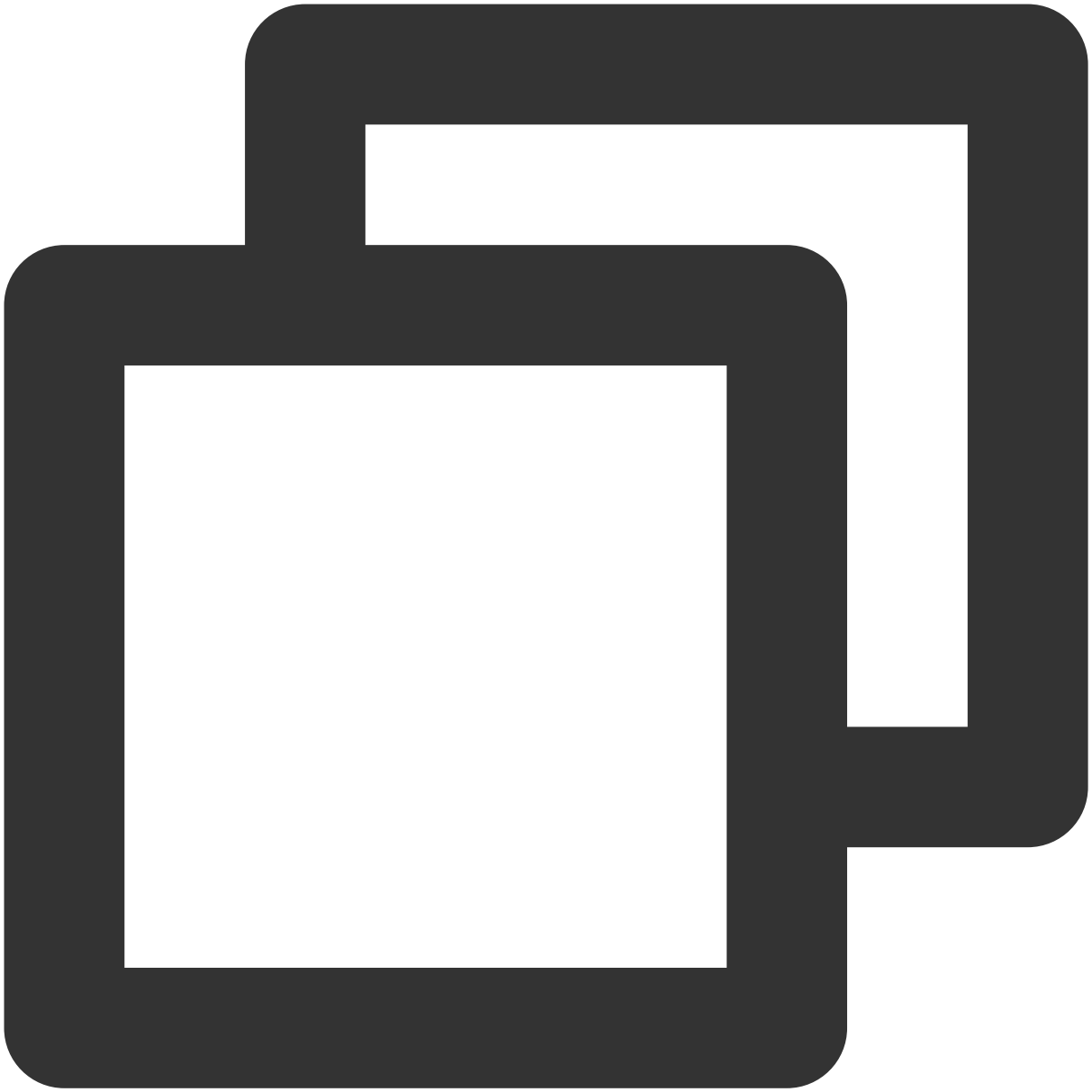
v1/callback/delete	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 Chat App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

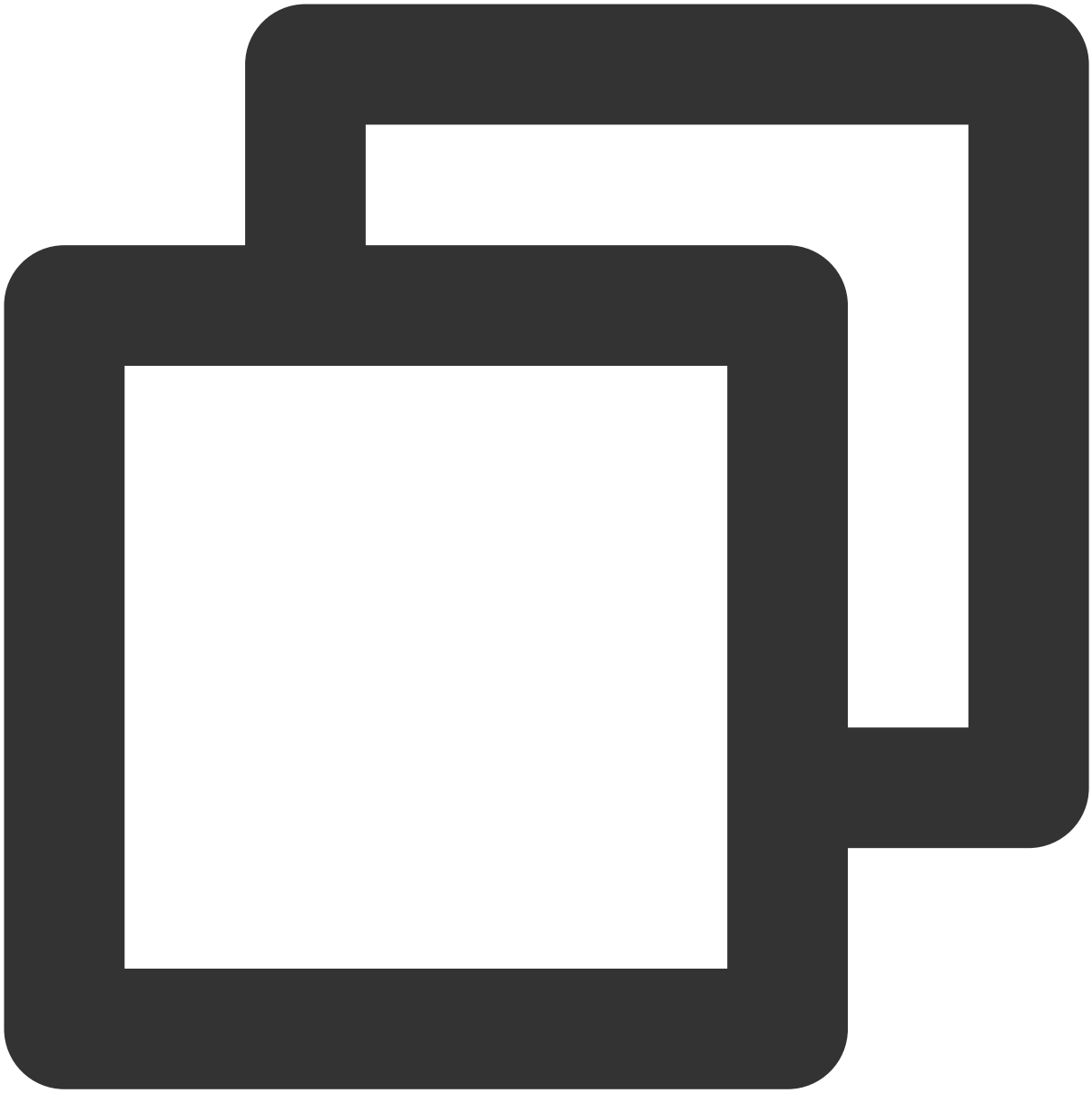
传空即可。



```
{

}
```

应答包体示例



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "5b0fa500064397cad3554506e27e18e1",
    "data": ""
}
```

应答包字段说明

字段	类型	说明

errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
data	String	无数据

错误码说明

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用

REST API

REST API 简介

最近更新时间：2024-07-24 10:20:03

REST API 是音视频通话 Call （TUICallKit）的后台 HTTP 管理接口，可以为开发者提供一套简单的管理入口。目前音视频通话 Call（TUICallKit）支持的 REST API 请参见 REST API 接口列表。
为了安全性，REST API 仅提供 HTTPS 接口。

使用条件

REST API 目前处于内测期，内测期间开通了音视频通话 Call（TUICallKit）**群组通话版**的应用（SDKAppId）可以使用 REST API，您也可以开通体验版免费试用，版本说明及开通指引参见 音视频通话能力版本。
目前各平台仅在特定版本的 TUICallKit 可使用 REST API，详见下表：

平台/框架	版本号
Android/iOS/Flutter/uni-app（客户端）	≥ 1.7.1
Web	≥ 1.4.6
微信小程序	≥ 1.5.1

注意：
参与通话的所有平台/框架升级到上述新的版本后，才可在控制台上查看到对应的通话信息。
内测期 REST API 支持查询近7天数据。

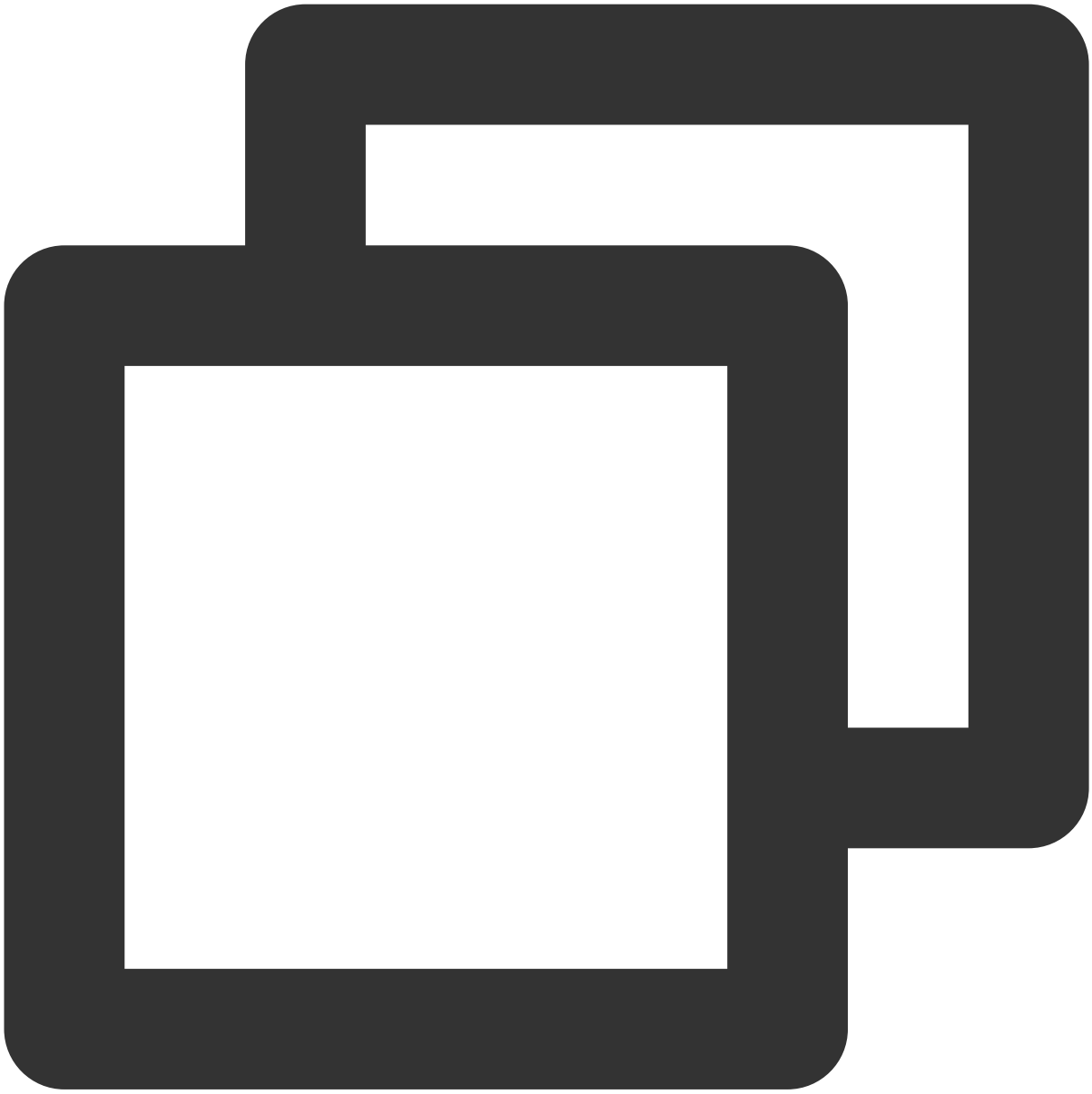
REST API 接口列表

功能说明	接口
通过 callId 获取记录	v1/records/get_record_by_callId
通过条件获取记录	v1/records/get_records_by_filter

调用方法

请求 URL

REST API 的 URL 格式如下：



```
https://xxxxxx/$version/$kind/$command?sdkappid=$SDKAppID&identifier=$identifier&us
```

其中各个参数的含义以及取值如下（参数名称及其取值均区分大小写）：

参数	含义	取值
https	请求协议	请求协议为 HTTPS，请求方式为 POST
xxxxxx	专属域名	

		`callkit-intl.trtc.tencent-cloud.com`
version	协议版本号	固定为 `v1`
kind	管理分类	示列：`v1/records/get_record_by_callId`，其中 records 为 kind
command	命令字，与 kind 组合用来表示具体的业务功能	示列：`v1/records/get_record_by_callId`，其中 get_record_by_callId 为 command
sdkappid	在控制台获取的应用标识	在申请接入时获得
identifier	用户名，调用 REST API 时必须为 App 管理员账号	使用 Chat 的管理员账号
usersig	用户名对应的密码	参见 生成 UserSig
random	标识当前请求的随机数参数	32位无符号整数随机数，取值范围 0 - 4294967295
contenttype	请求格式	固定值为 json

注意：

领取或购买套餐包后会在 Chat 账号系统中创建 administrator 管理员账号，在请求中的 `identifier` 参数中请使用 administrator，如果您在 Chat 账号管理 中取消或删除管理员，请正确指定管理员账号以及对应的 userSig。

App 可以在每次调用 REST API 时都生成管理员账号的 UserSig，亦可生成一个固定的 UserSig 重复使用，但需特别注意 UserSig 的有效期。

在创建房间以及进入房间等操作时，系统会在 Chat 系统中自动导入 Chat 账号，敬请知悉。

HTTP 请求包体格式

REST API 仅支持 POST 方法，其请求包体为 JSON 格式，具体的包体格式参见每个 API 的详细描述。

注意：

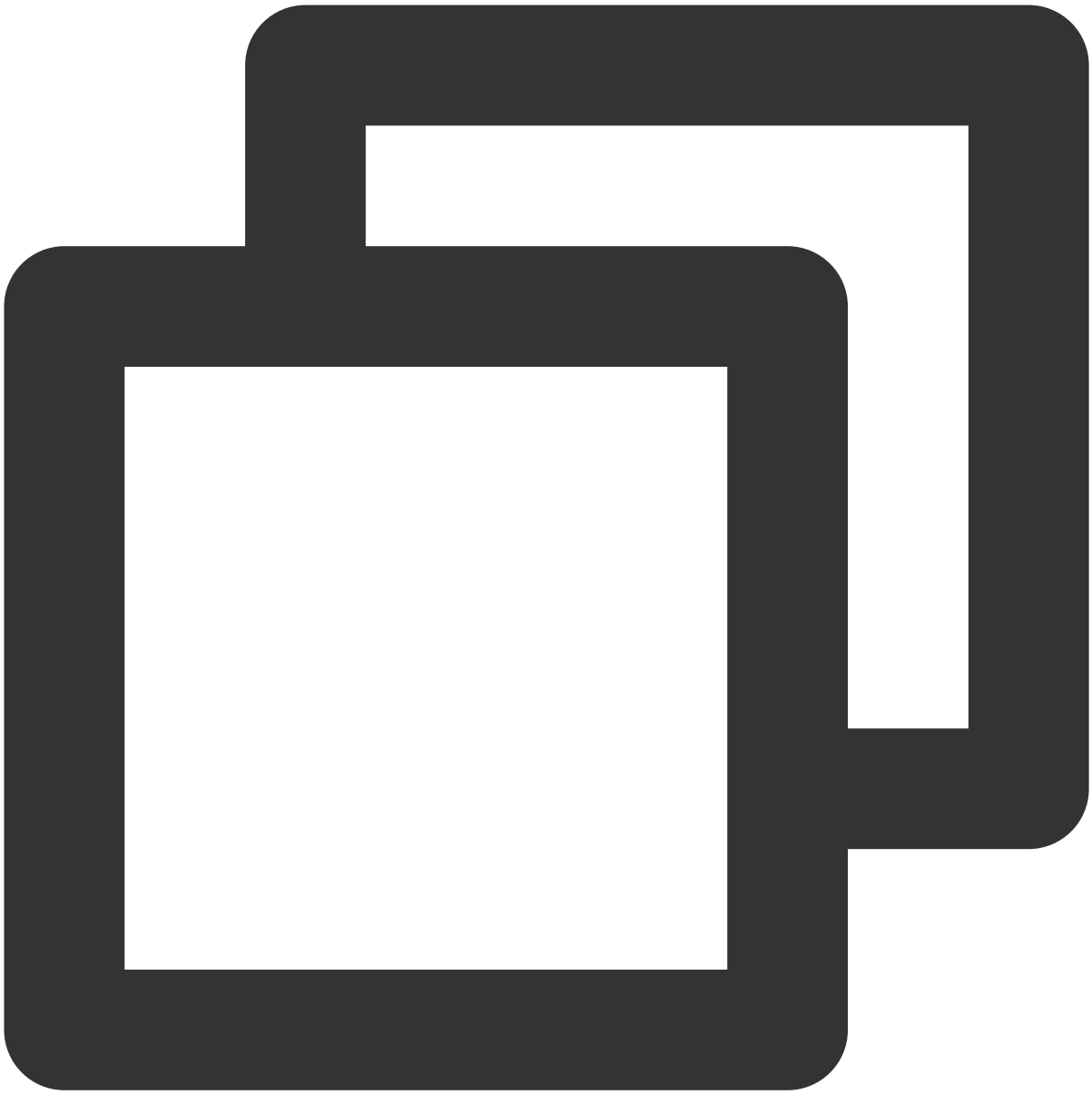
POST 包体不能为空，即使某条协议包体中不用携带任何信息，也必须要携带一个空的 JSON 对象，即 `{}`。

HTTP 返回码

除非发生网络错误（例如502错误），否则 REST API 的调用结果均为200，实际 API 调用的错误码与错误信息在 HTTP 应答包体中返回。

HTTP 应答包体格式

REST API 的应答包体也是 JSON 格式，其格式符合如下特征：



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "1c8960ac38d61be38b6fb219db0182d1",
    "data": {}
}
```

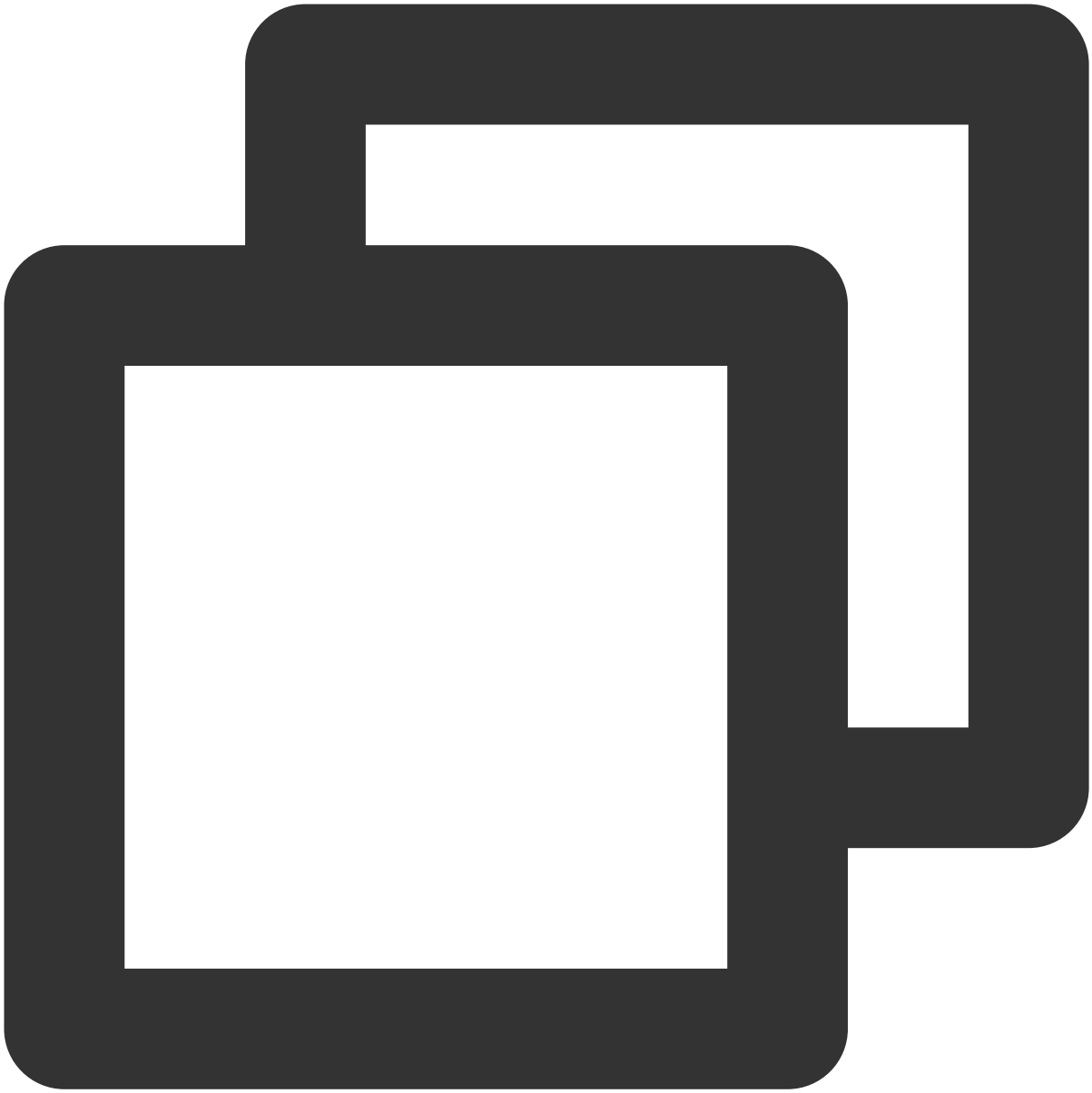
应答包体中必然包含 errorCode、errorMessage、requestId 这三个属性，其含义如下：

字段	类型	说明

errorCode	String	错误码，0为成功，其余为失败
errorMessage	String	错误信息
requestId	Integer	请求唯一标识符

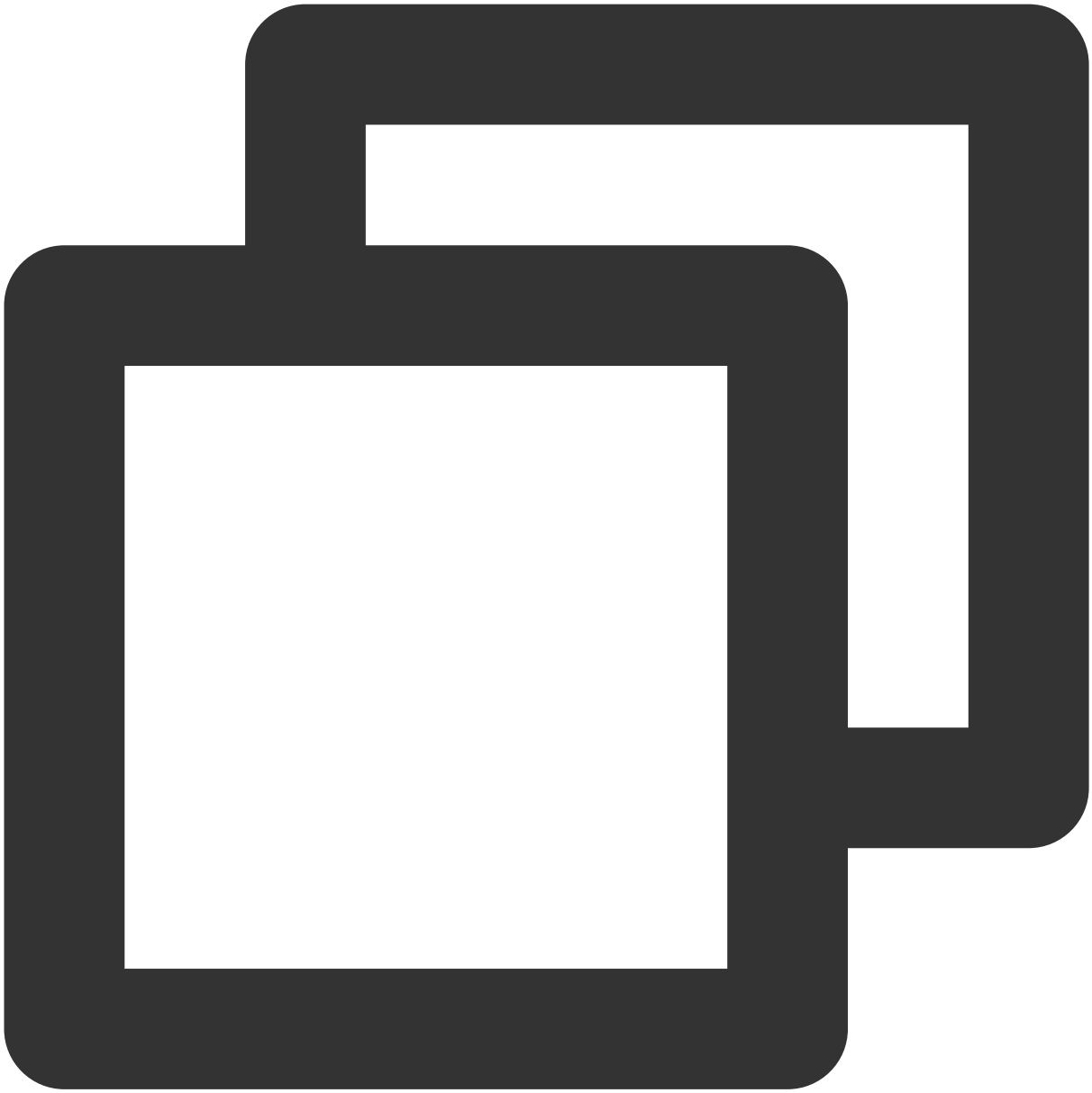
调用示例

以下为通过 REST API 获取指定 callId 的通话记录示例。
HTTPS 请求：



```
POST /records/get_record_by_callId?usersig=xxx&identifier=admin&sdkappid=88888888&r
Host: callkit-intl.trtc.tencent-cloud.com
Content-Length: 36
{
    "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582"
}
```

HTTPS 应答：



```
HTTP/2 200 OK
Date: Fri, 21 Apr 2023 06:06:16 GMT
Content-Length: 112
Connection: keep-alive

{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "9f01db503f2006ad10c45f4f4609e38d",
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
```

```
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```

REST API 公共错误码

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
70001	请求参数无效，请检查必填参数是否缺失或填错
70002	UserSig 无效
70003	UserSig 过期
70004	请求用户非超级管理员
70005	请求被限频
70009	解析请求 body 出错，请检查请求参数类型是否正确
未知错误码	系统内部错误，请 提交工单 联系技术人员

常见问题

REST API 请求有概率超时，收不到任何响应？

您可以从以下几个方面确认：

1. 后台 REST 接口设置的超时时间是 3s，调用方设置的超时时间应该长于 3s。
2. `telnet callkit-intl.trtc.tencent-cloud.com 443` 确认能否连接服务端口。
3. 使用 `curl -I https://callkit-intl.trtc.tencent-cloud.com` 简单测试看状态码是否为200。
4. 确认机器的 dns server 配置是内网 dns server，还是公共 dns server。如果是内网 dns server，请确保 dns server 网络出口和本机器网络出口 IP 所在地域运营商一致。
5. 建议业务调用方使用**长连接+连接池**模式。

通过 callId 获取记录

最近更新时间：2024-07-24 10:20:03

功能说明

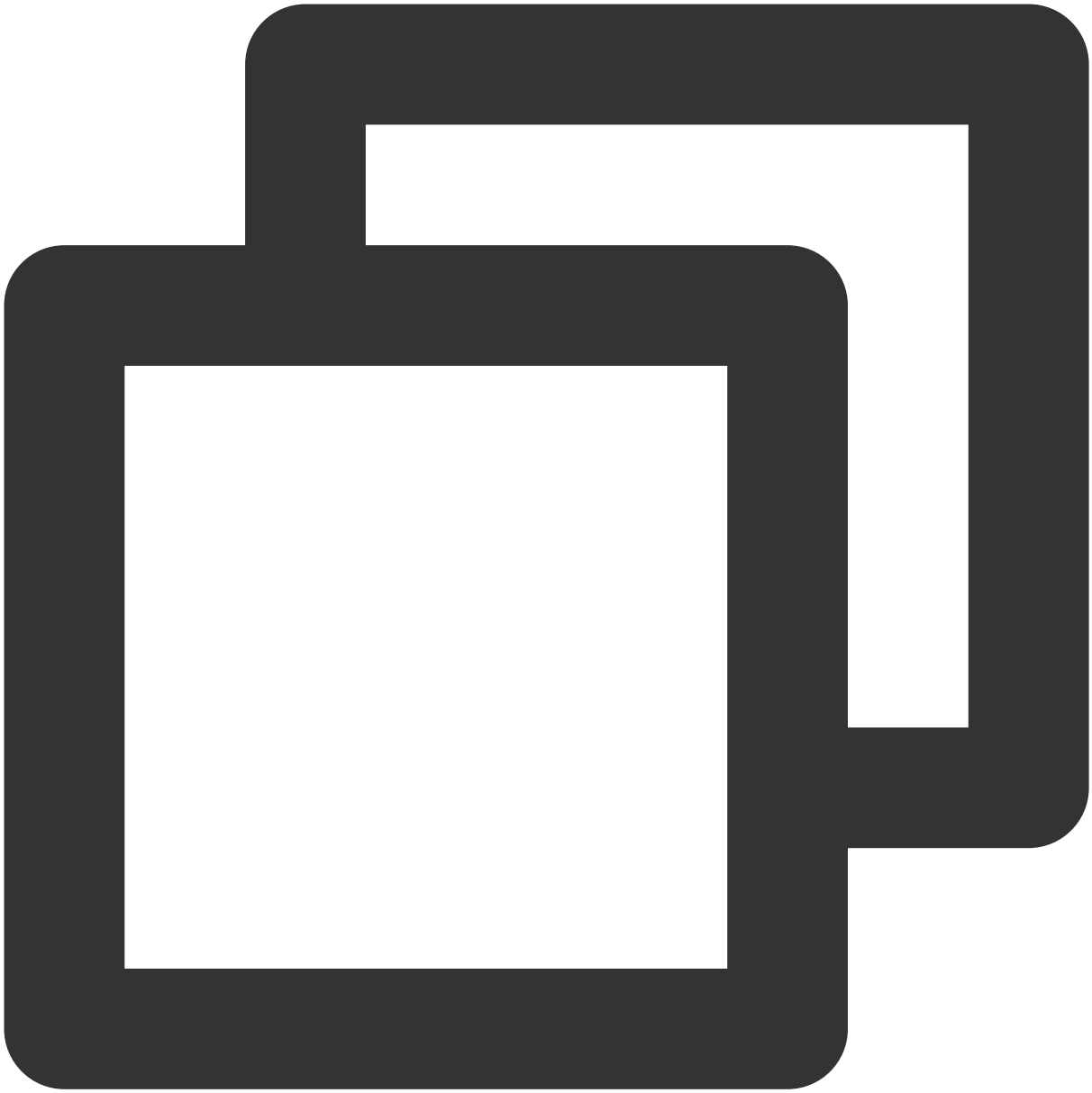
管理员可通过该接口获取指定 callId 的通话记录。

说明：

REST API 目前处于内测期，您可以查询创建时间在7天内的通话数据。

接口调用说明

请求URL示列



```
https://xxxxxx/v1/records/get_record_by_callId?sdkappid=88888888&identifier=admin&u
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参见 REST API 简介。

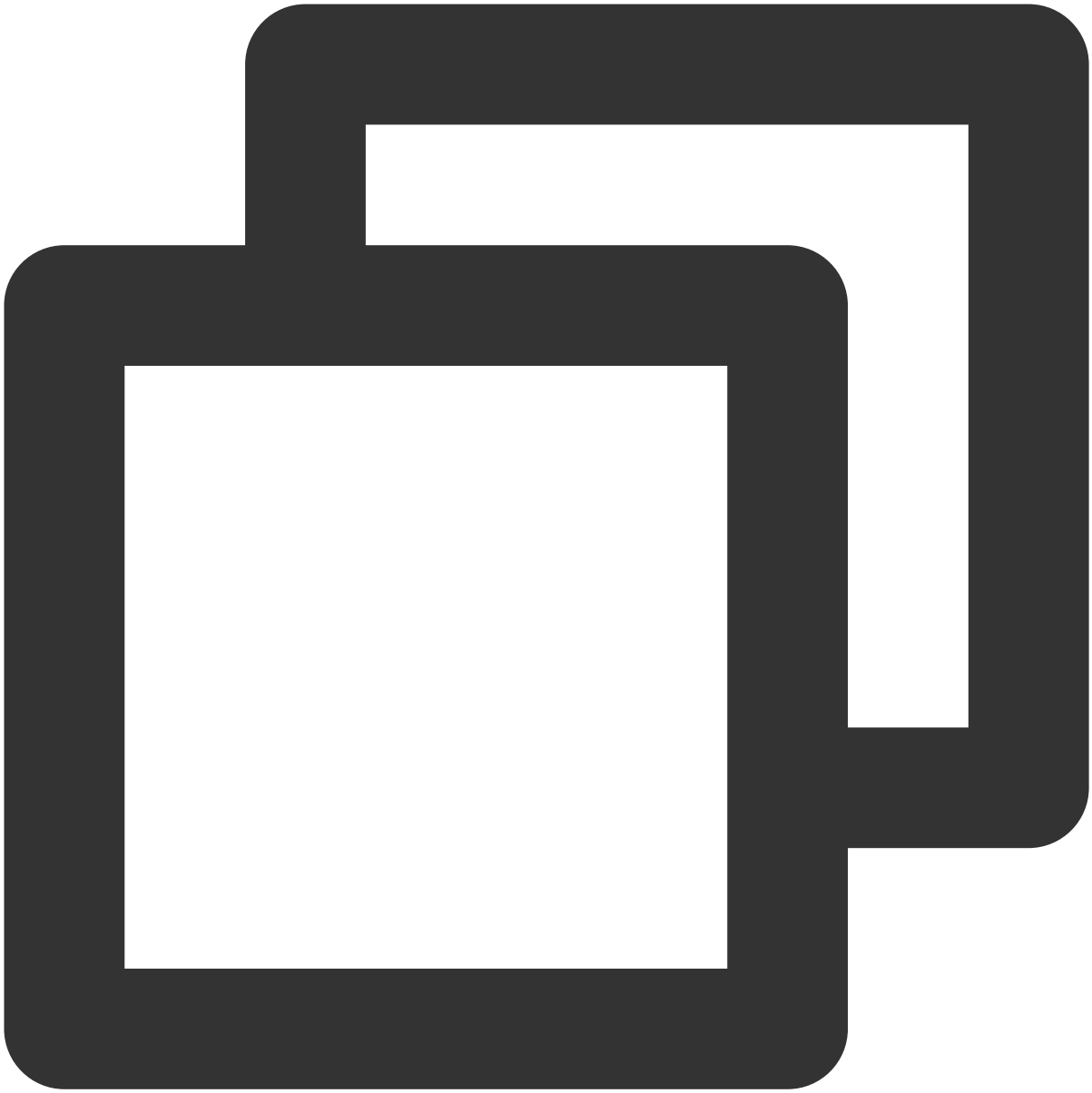
参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名：

	`callkit-intl.trtc.tencent-cloud.com`
v1/records/get_record_by_callId	请求接口
sdkappid	创建应用时控制台分配的 SDKAppID
identifier	使用 Chat 的管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

20次/秒。

请求包示例



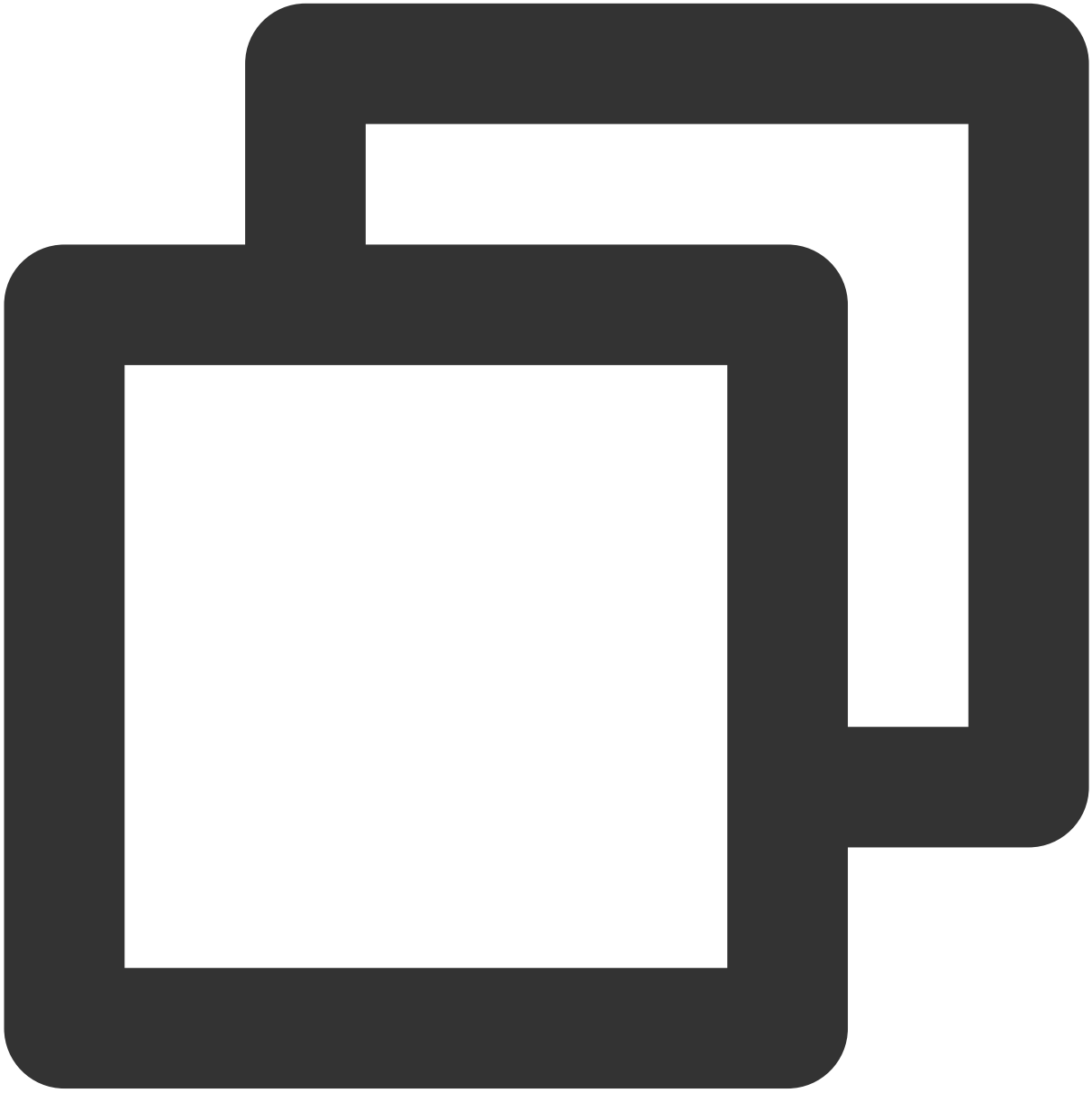
```
{
    "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582"
}
```

请求包字段说明

字段	类型	属性	说明

callId	String	必填	通话唯一 ID

应答包体示例



```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "3fdc344c7a67461c6af3cfc3e77744b5",
```

```
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```

应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
callId	String	通话的唯一 ID
sdkAppId	Integer	您的 sdkAppId
mediaType	String	媒体类型 `video` 视频通话 `audio` 音频通话
roomId	String	通话的房间 ID
startCallTs	Integer	通话发起的时间戳（秒级）
acceptTs	Integer	通话接通的时间戳（秒级）

endTs	Integer	通话结束的时间戳（秒级）
caller	String	主叫 userId
totalUserNumber	Integer	参与通话的总人数
callType	String	通话类型： `single` 一对一通话 `group` 群组通话
callResult	String	通话结果： `cancel` 取消：主叫方在接通前取消通话。 `reject` 拒接：被叫方拒接。 `not_answer` 未接听：被叫方超时未接听。 `normal_end` 完成：通话接通并正常结束。 `call_busy` 忙线：通话忙线。 `interrupt` 中断：网络等原因导致通话中断。
callees	Array	被叫且参与通话的 userId 列表

说明：

callResult 字段仅一对一通话可能显示所有类型，群聊仅有 `normal_end` 。

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。实际的错误码、错误信息是通过应答包体中的 errorCode、errorMessage 来表示的。

公共错误码（70000到79999）请参见 错误码。

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
70011	未找到任何记录
未知错误码	未知错误，请 提交工单 联系技术人员

通过条件获取记录

最近更新时间：2024-07-24 10:20:03

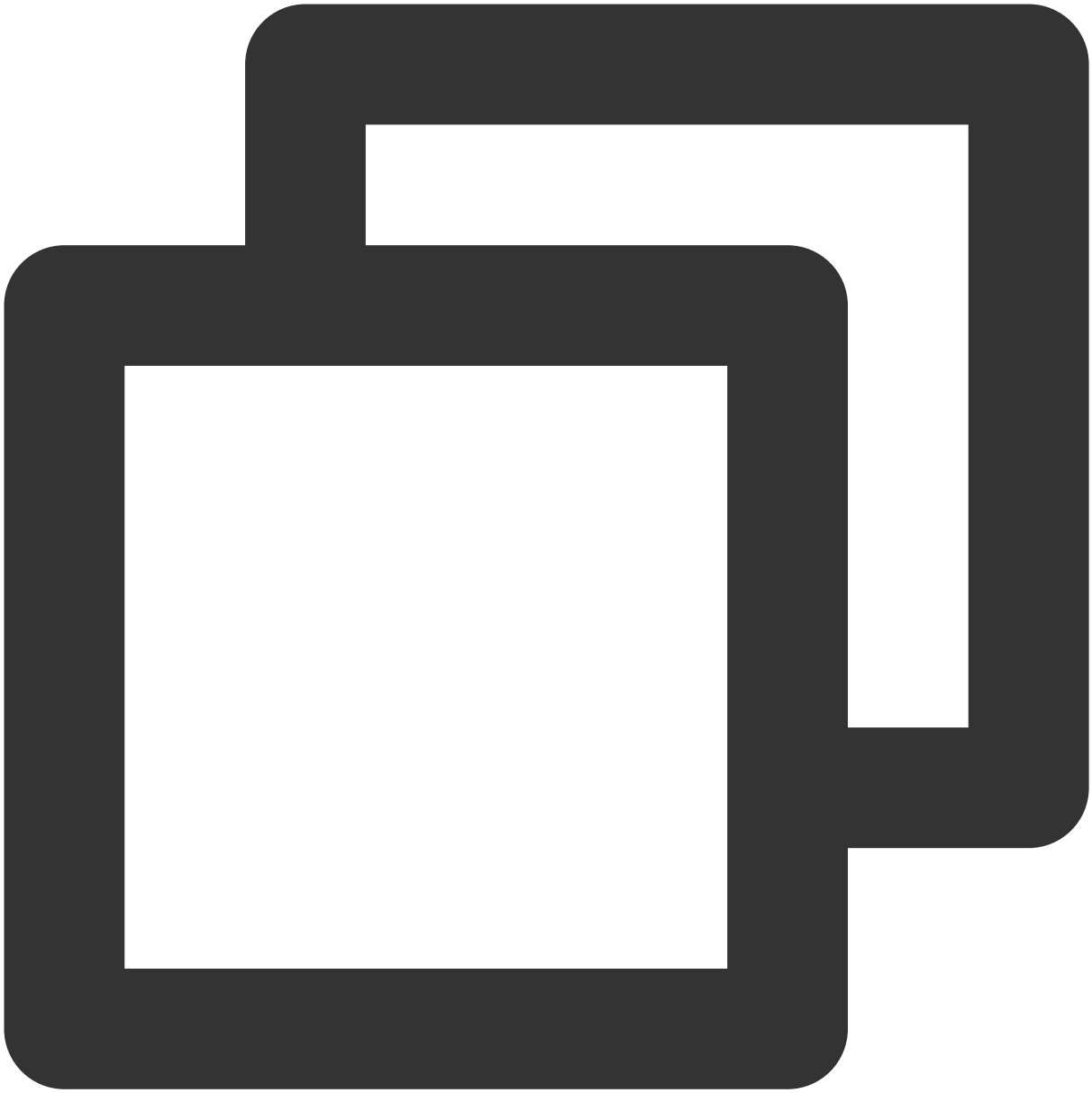
管理员可通过该接口获取指定条件的通话记录。

说明：

REST API 目前处于内测期，您可以查询创建时间在七天内的通话数据。

接口说明

请求URL示列



```
https://xxxxxx/v1/records/get_records_by_filter?sdkappid=88888888&identifier=admin&
```

请求参数说明

下表仅列出调用本接口时涉及修改的参数及其说明，更多参数详情请参见 REST API 简介。

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名：`callkit-intl.trtc.tencent-cloud.com`

v1/records/get_records_by_filter	请求接口
sdkappid	创建应用时控制台分配的 SDKAppID
identifier	使用 Chat 的管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

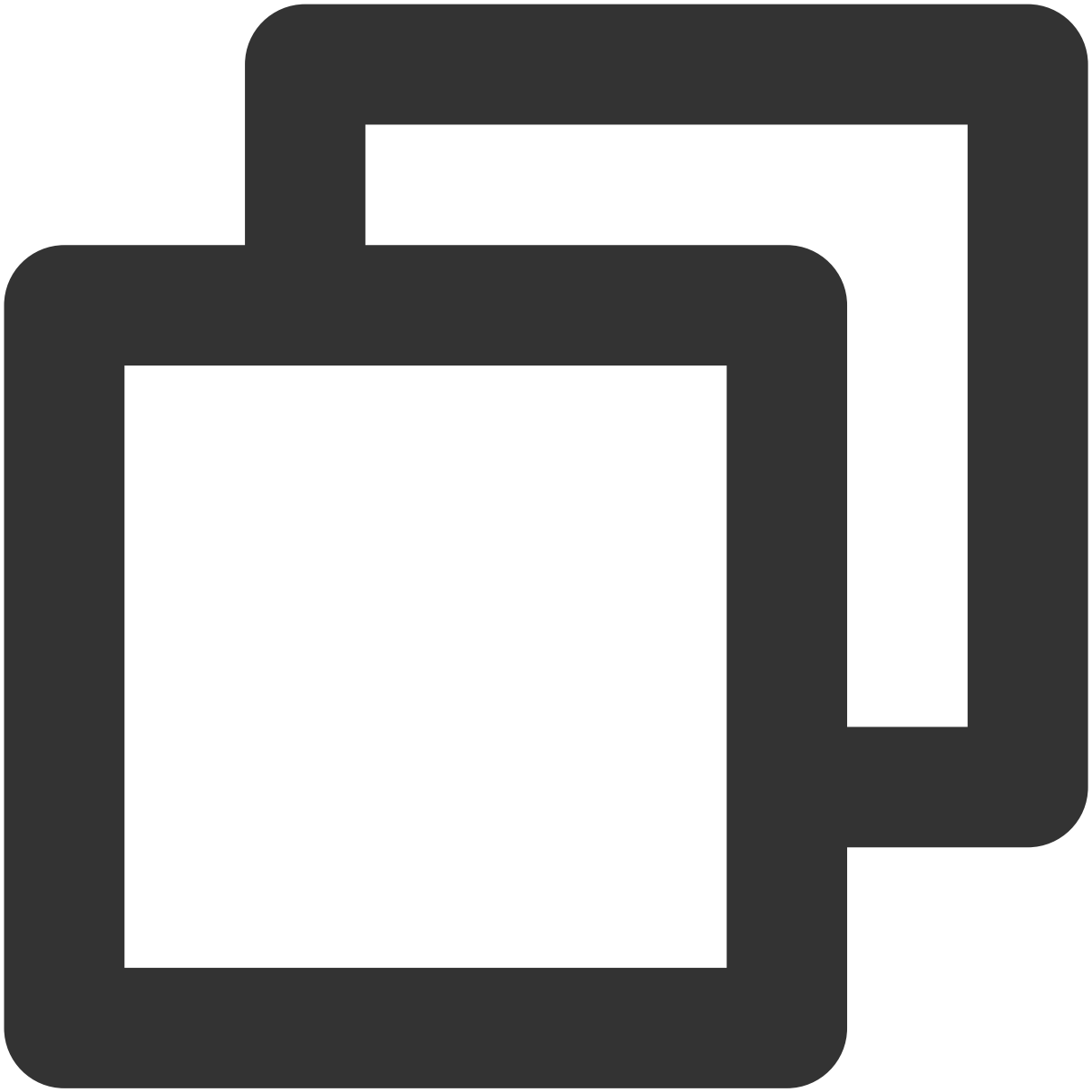
最高调用频率

200次/秒。

请求包示例

基础形式

创建房间

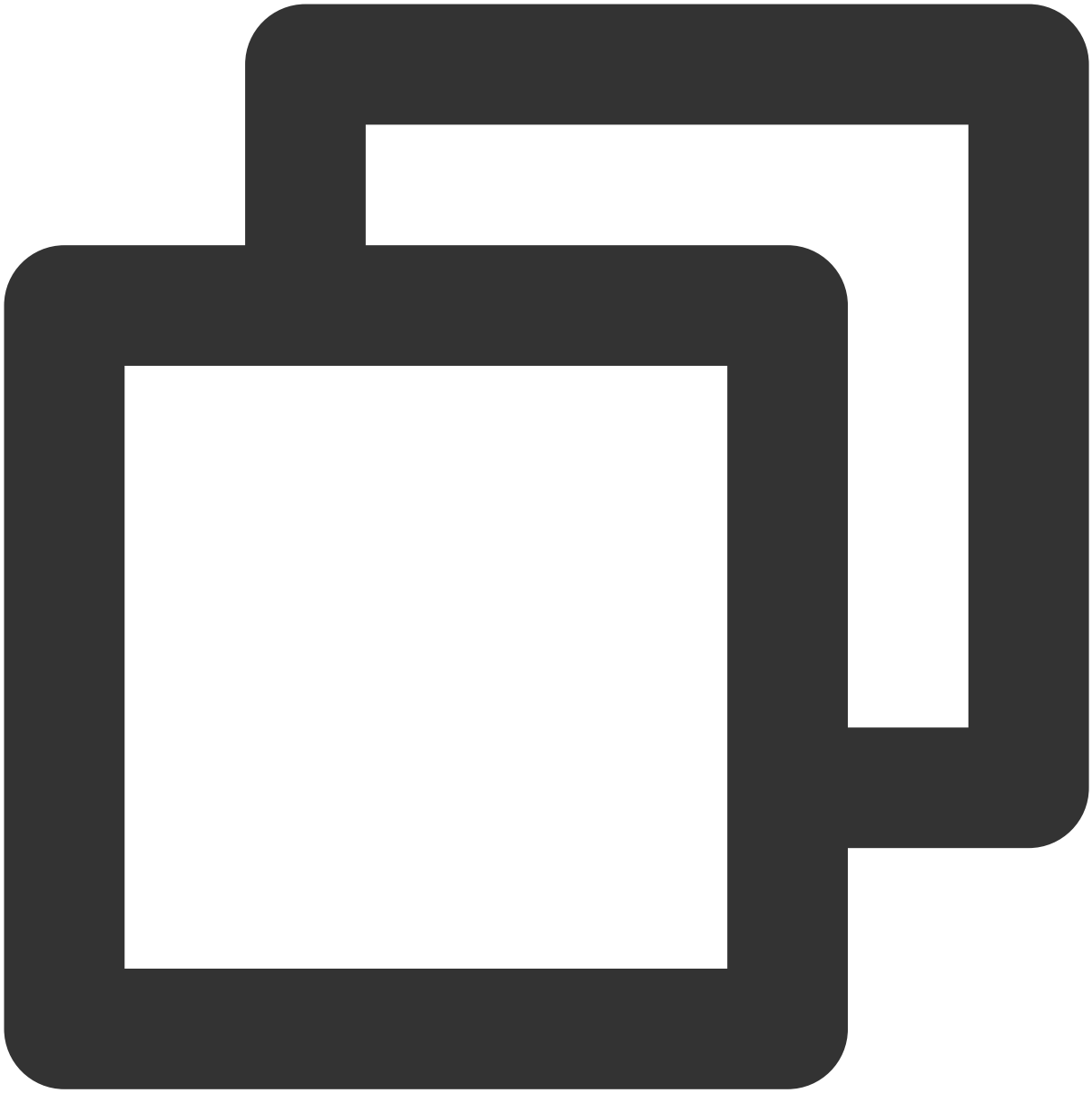


```
{
    "startTimestamp": 1618705638,
    "endTimestamp": 1618705738,
    "callResult": "normal_end",
    "callType": "single",
    "numberPerPage": 20,
    "page": 2
}
```

请求包字段说明

字段	类型	属性	说明
startTimestamp	Integer	必填	通话开始时间戳（秒级）
endTimestamp	Integer	选填	通话结束时间戳（秒级），不填默认为7天
callResult	String	选填	通话结果，不填默认为所有结果
callType	String	选填	通话类型，不填默认为所有类型
numberPerPage	Integer	选填	每页查询数量，默认为：100
page	Integer	选填	查询页数，不填默认为第一页

应答包字段示列




```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "8ba14f64a1d92d38705eec775e1f3deb",
    "data": {
        "totalNum": 3,
        "page": 2,
        "callRecordList": [
            {
                "callId": "2ae7d549-c441-4a9b-87c0-61810fe19581",
                "sdkAppId": 88888888,
```

```
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709003,
            "acceptTs": 1688709003,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
        },
        {
            "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
            "sdkAppId": 88888888,
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709303,
            "acceptTs": 1688709303,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
        },
        {
            "callId": "2ae7d549-c441-4a9b-87c0-61810fe19583",
            "sdkAppId": 88888888,
            "mediaType": "video",
            "roomId": "456",
            "startCallTs": 1688709903,
            "acceptTs": 1688709903,
            "endTs": 1689150030,
            "caller": "123",
            "totalUserNumber": 0,
            "callType": "single",
            "callResult": "offline",
            "callees": [
                "111",
                "123"
            ]
```

```
                }
            ]
        }
    }
```

应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
totalNum	Integer	此次查询的总数量
page	Integer	当 page > 0 时 说明还有数据，将 page+1 可继续请求之后的数据。
callRecordList	Array	请参见：单条通话记录说明

单条通话记录说明

字段	类型	说明
callId	String	通话的唯一 ID
sdkAppId	Integer	您的 sdkAppId
mediaType	String	媒体类型 `video` 视频通话 `audio` 音频通话
roomId	String	通话的房间 ID
startCallTs	Integer	通话发起的时间戳（秒级）
acceptTs	Integer	通话接通的时间戳（秒级）
endTs	Integer	通话结束的时间戳（秒级）
caller	String	主叫 userId
totalUserNumber	Integer	参与通话的总人数
callType	String	通话类型

版权所有：腾讯云计算（北京）有限责任公司

		`single` 一对一通话 `group` 群组通话
callResult	String	通话结果： `cancel` 取消：主叫方在接通前取消通话。 `reject` 拒接：被叫方拒接。 `not_answer` 未接听：被叫方超时未接听。 `normal_end` 完成：通话接通并正常结束。 `call_busy` 忙线：通话忙线。 `interrupt` 中断：网络等原因导致通话中断。
callees	Array	被叫且参与通话的 userId 列表

说明：

callResult 字段仅一对一通话可能显示所有类型，群聊仅有 `normal_end` 。

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。实际的错误码、错误信息是通过应答包体中的 errorCode、errorMessage 来表示的。

公共错误码（70000到79999）请参见 错误码。

错误码	含义说明
0	请求成功
50001	当前应用需要购买 TUICallKit 群组通话版套餐包方可使用
70011	未找到任何记录
未知错误码	未知错误，请 提交工单 联系技术人员

客户端 API（TUICallKit）

Android

API 概览

最近更新时间：2024-07-25 17:49:30

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
enableIncomingBanner	开启/关闭来电横幅显示

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/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（群组通话版套餐支持）

版权所有：腾讯云计算（北京）有限责任公司

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

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

版权所有：腾讯云计算（北京）有限责任公司

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

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2024-07-24 10:20:03

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
enableIncomingBanner	开启/关闭来电横幅显示

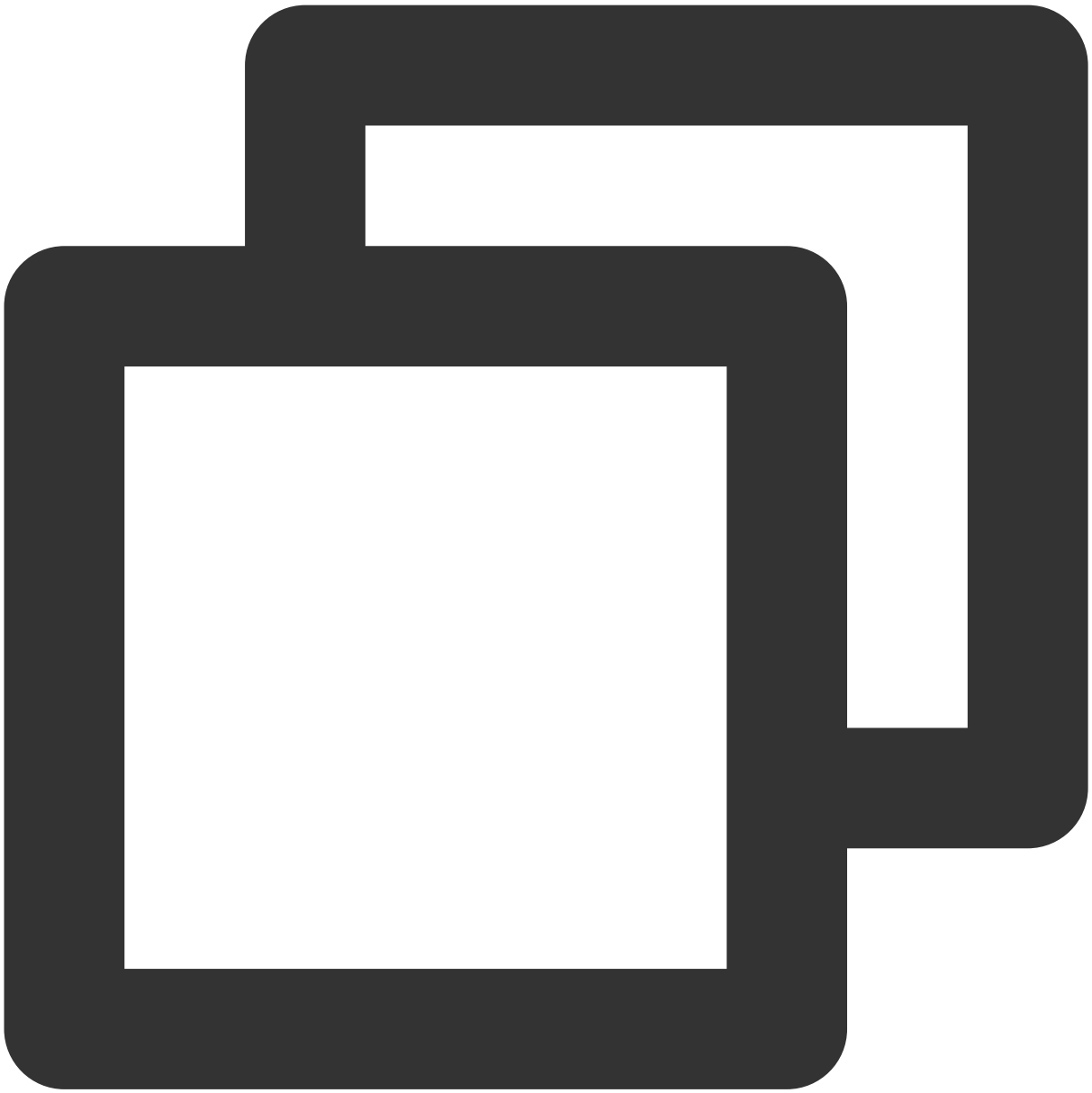
API 详情

createInstance

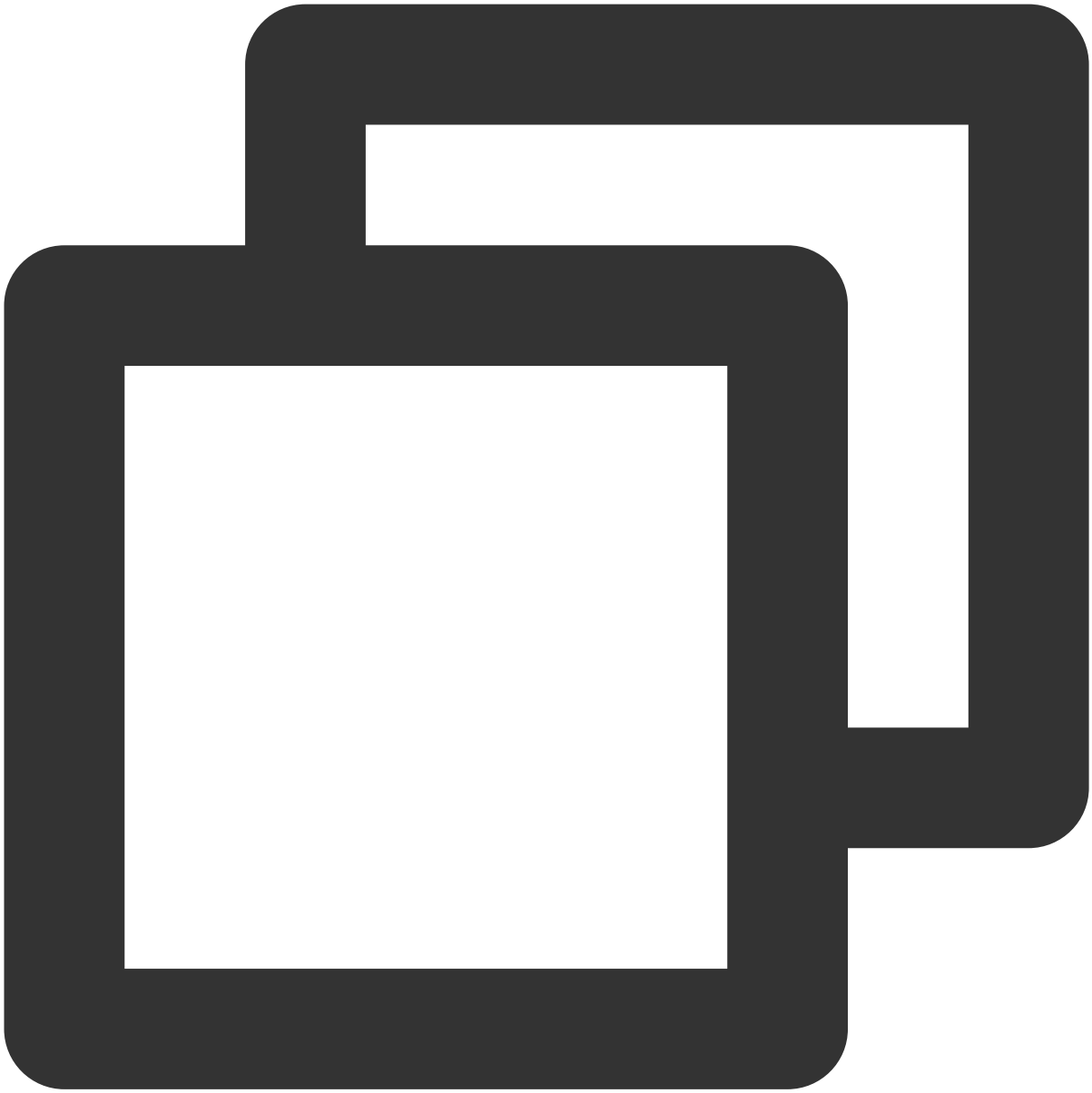
创建 TUICallKit 的单例。

Kotlin

Java



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



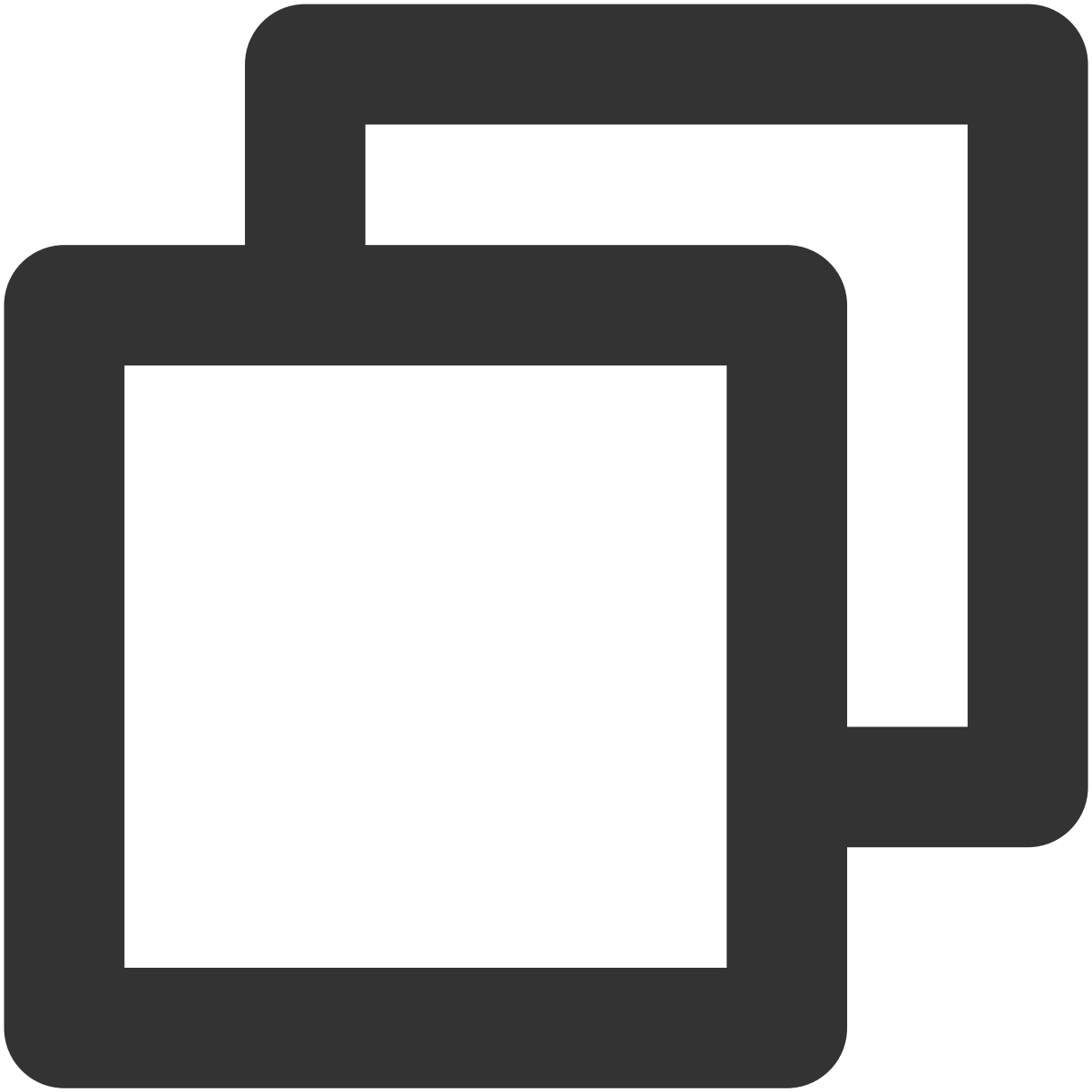
```
TUICallKit createInstance(Context context)
```

setSelfInfo

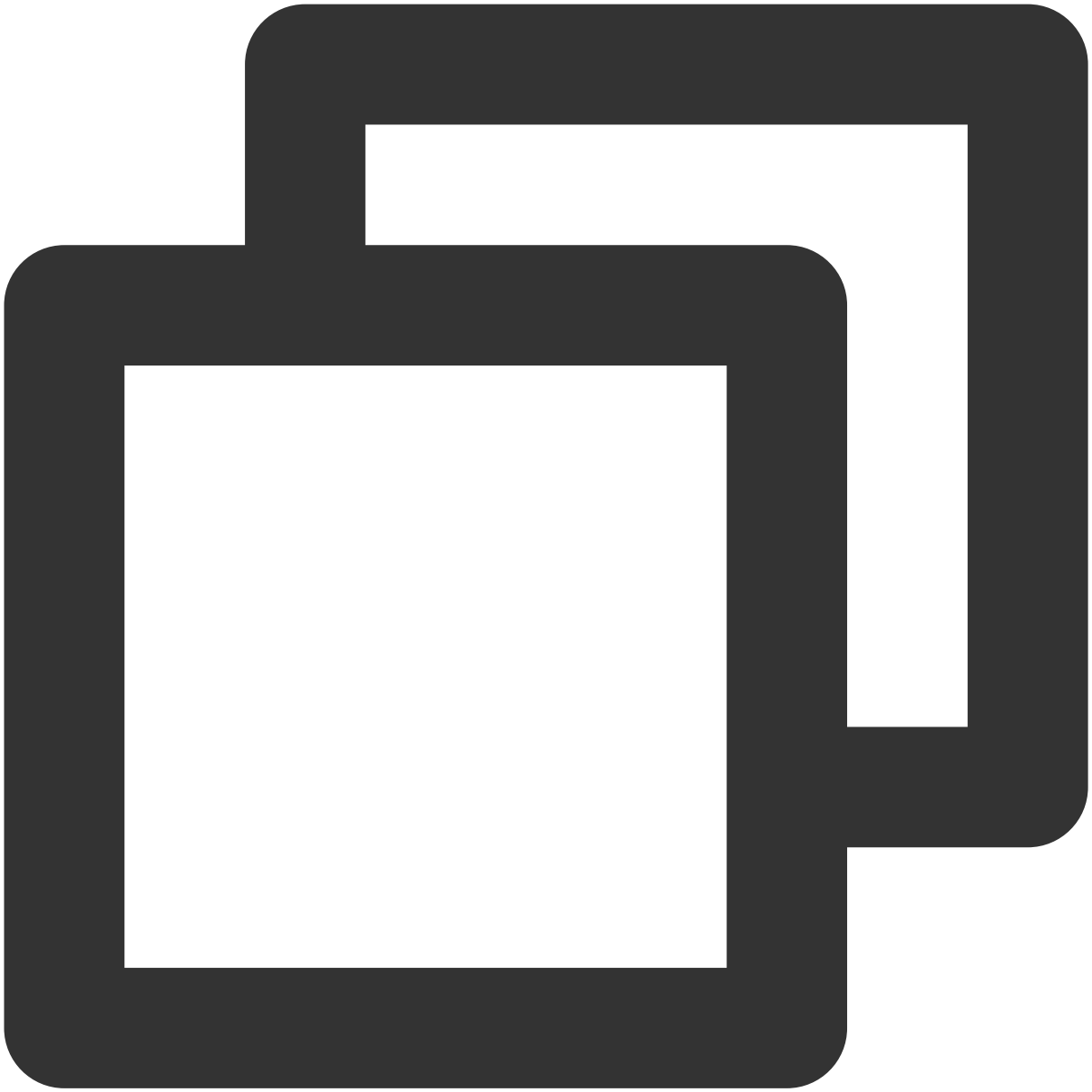
设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

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：

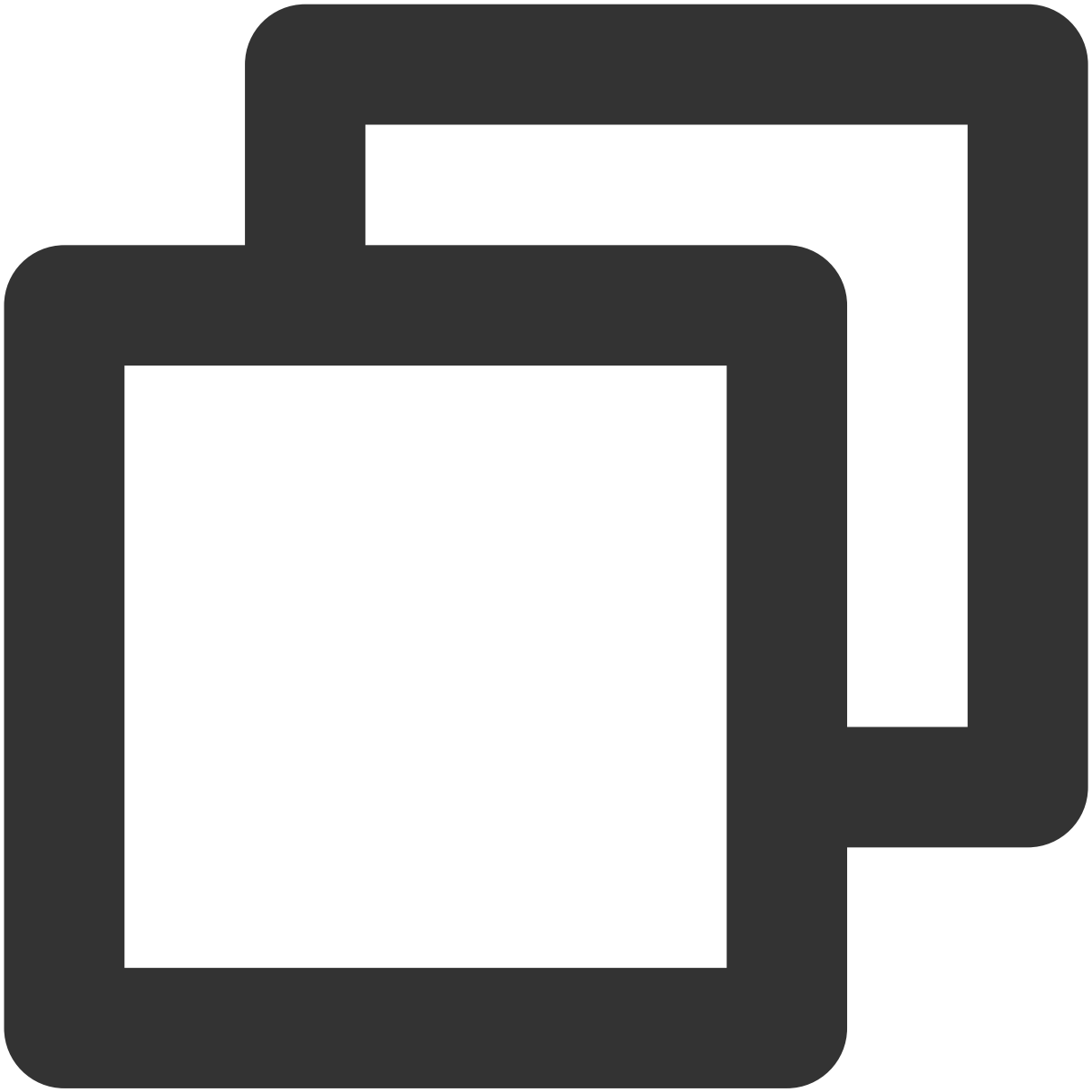
参数	类型	含义
nickname	String	目标用户的昵称
avatar	String	目标用户的头像

call

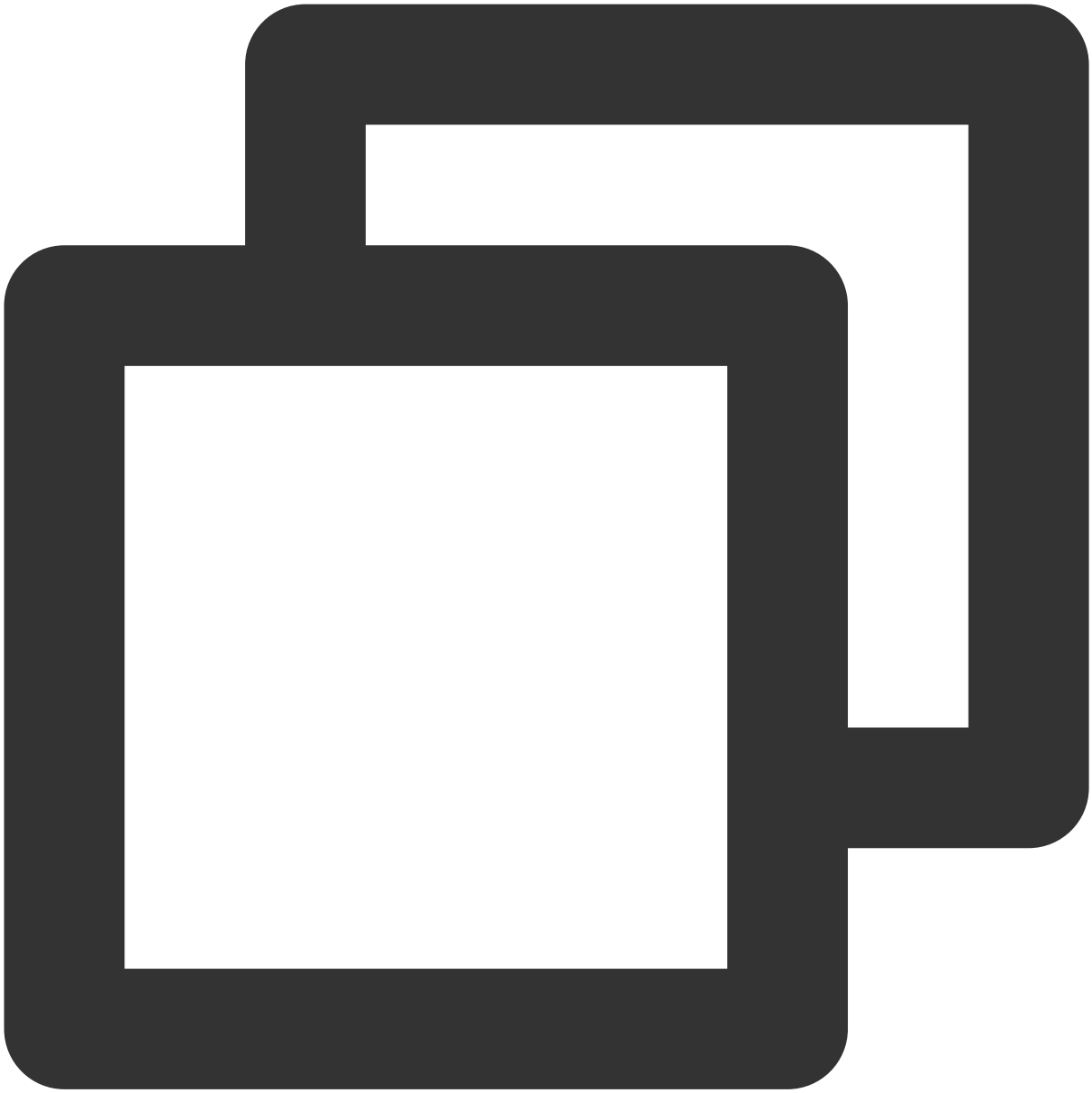
拨打电话（1v1通话）。

Kotlin

Java



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



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

参数如下表所示：

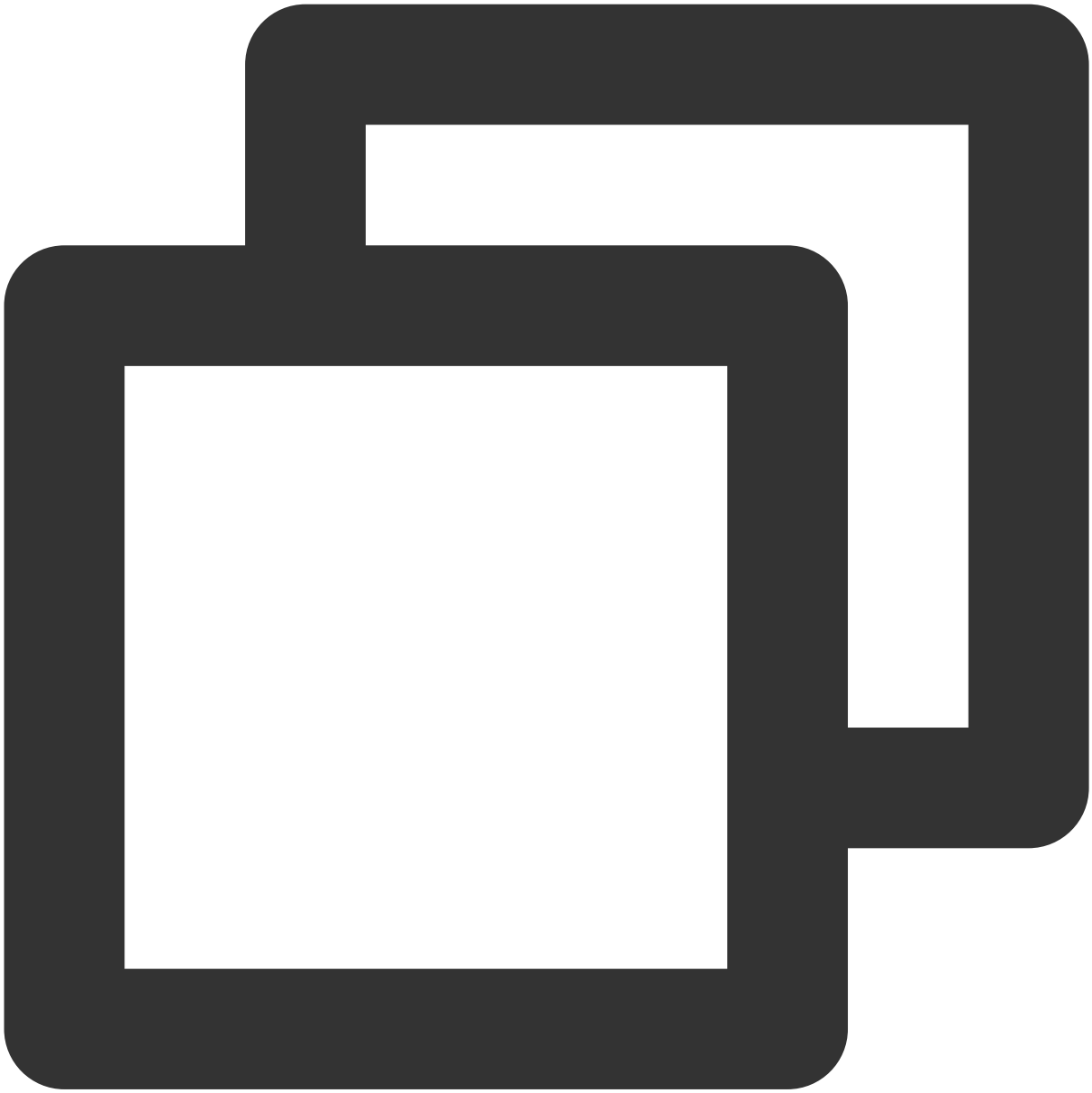
参数	类型	含义
userId	String	目标用户的 userId
callMediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

call

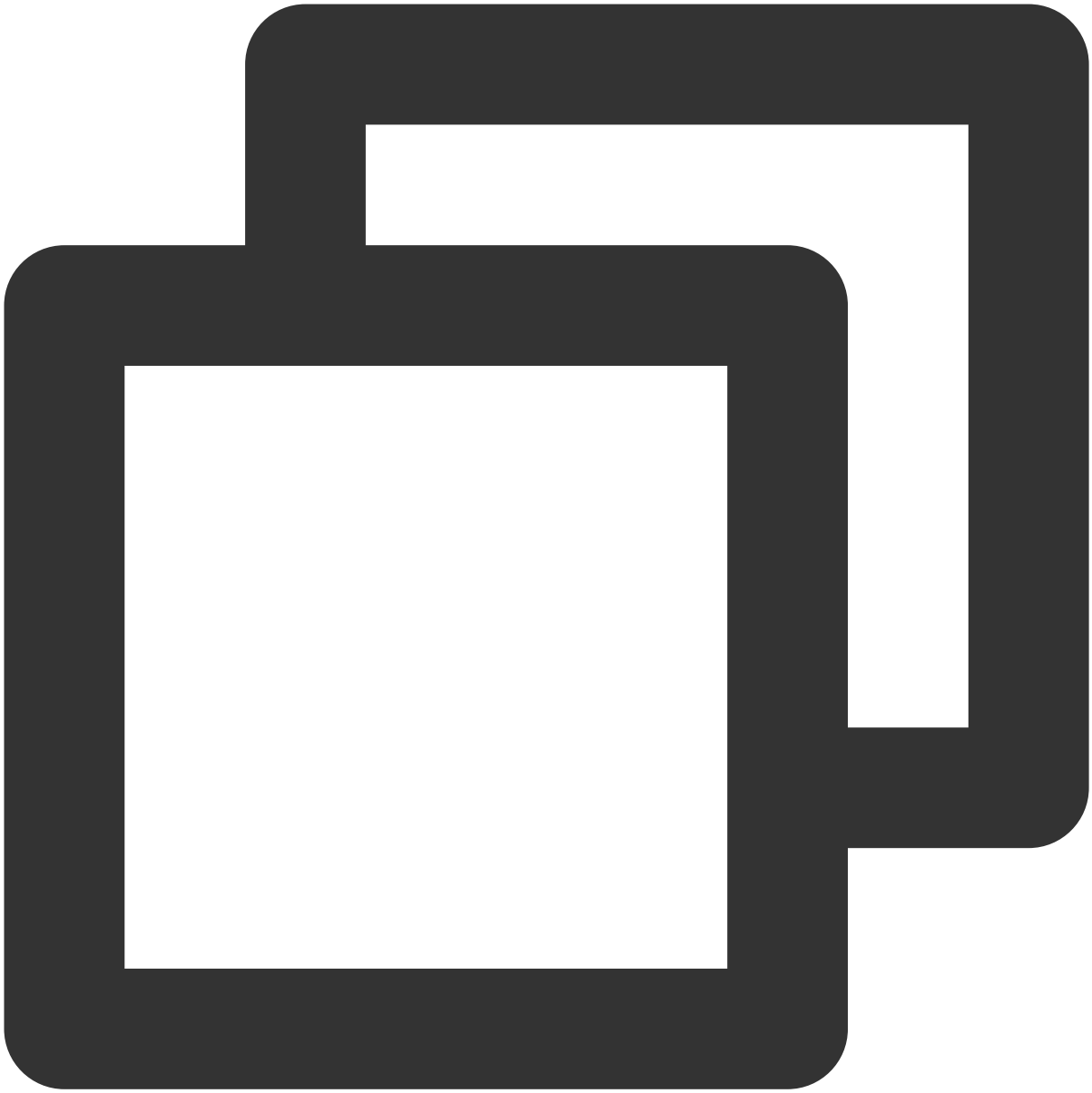
拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。

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。

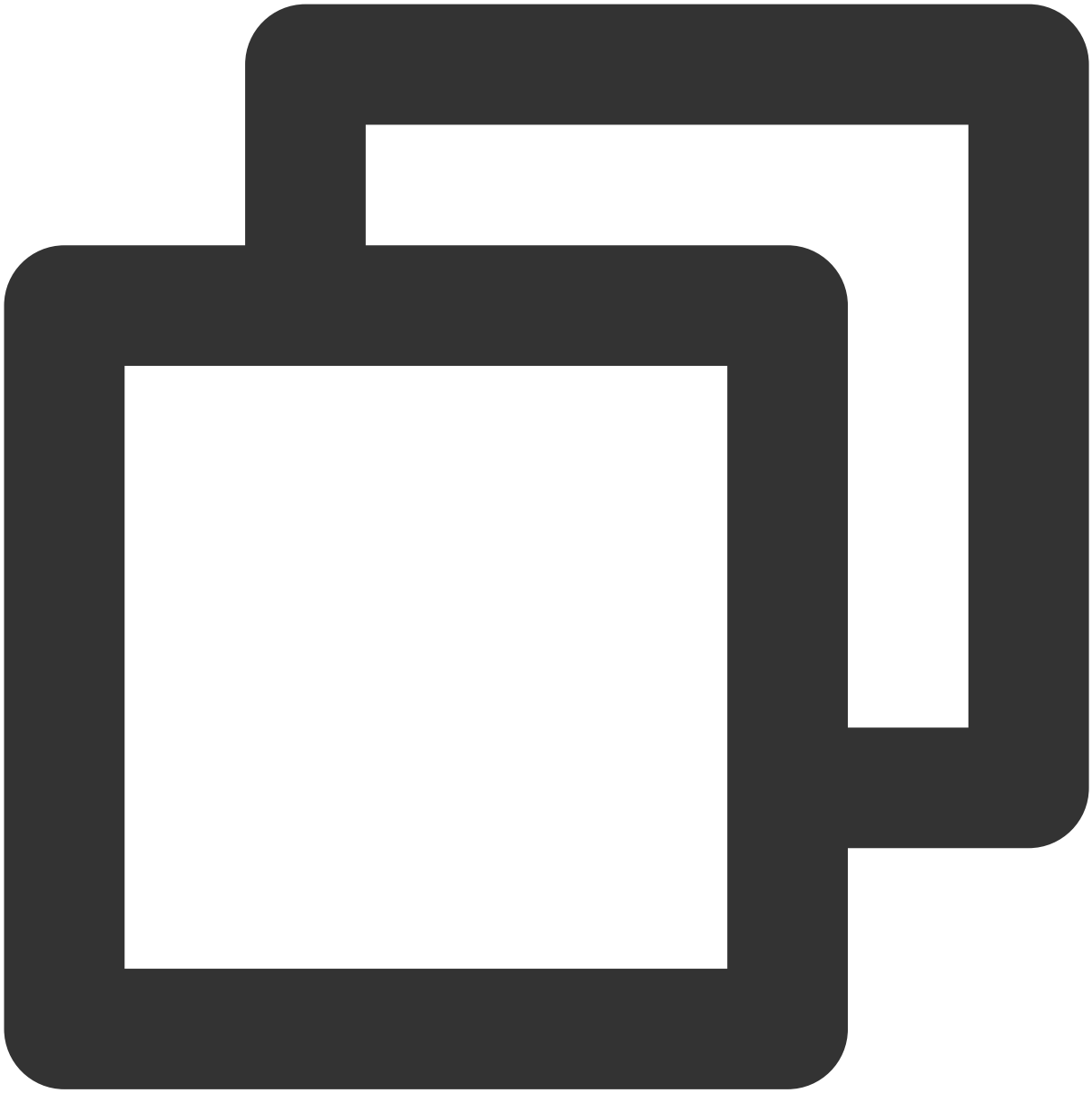
注意

使用群组通话前需要创建 Chat 群组，如果已经创建，请忽略。

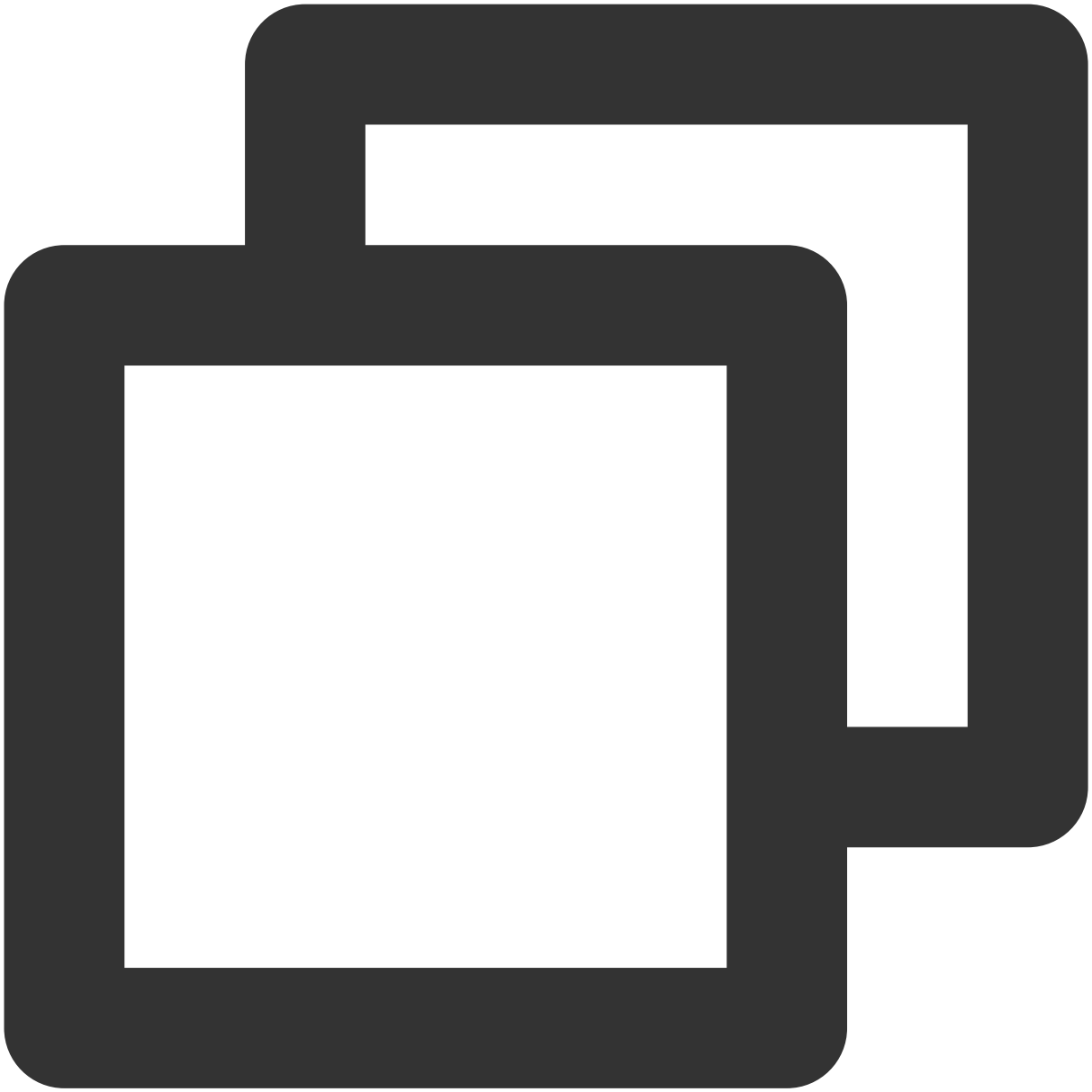
群组的创建详见：Chat 群组管理，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。

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。

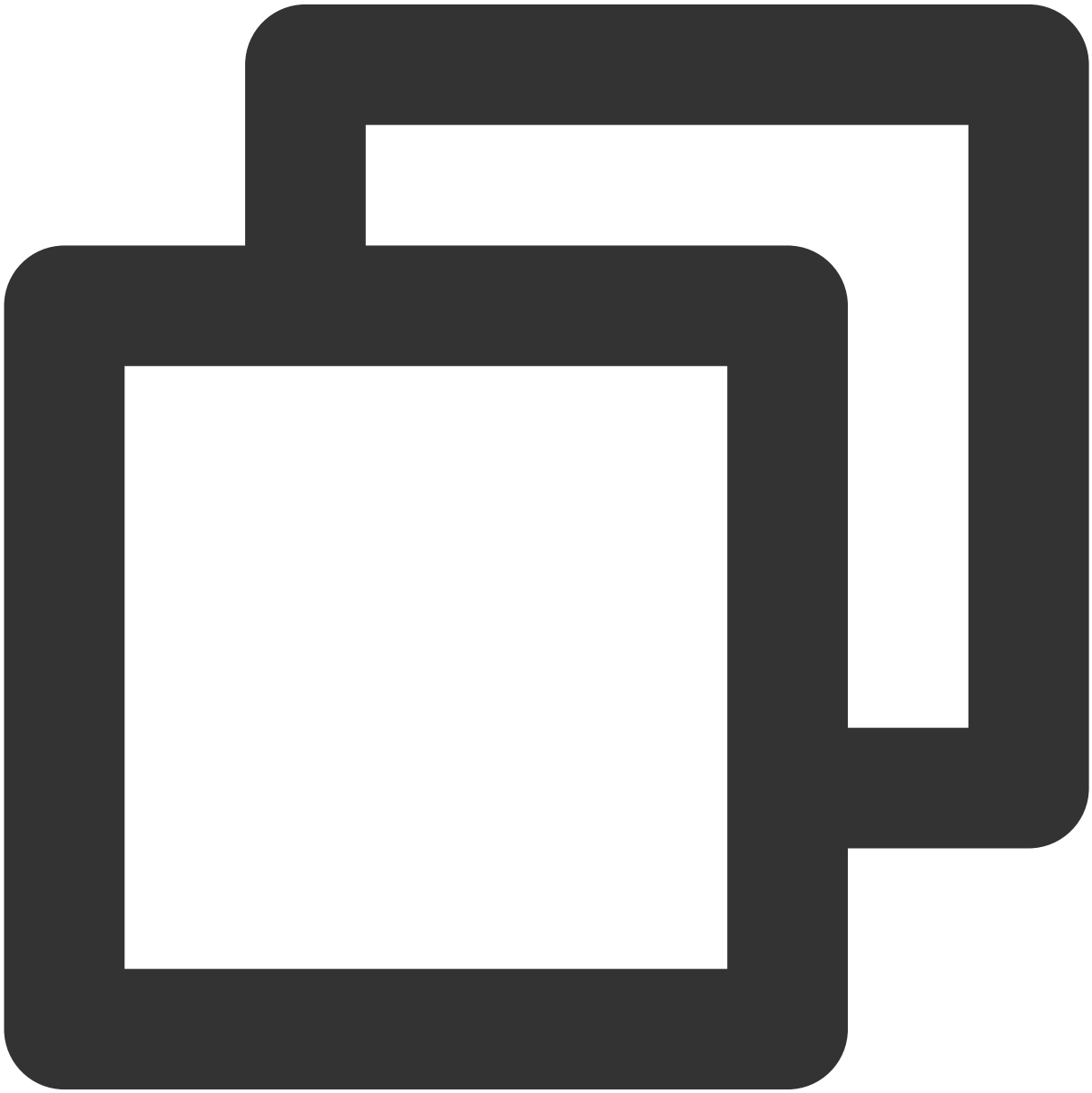
注意

使用群组通话前需要创建 Chat 群组，如果已经创建，请忽略。

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

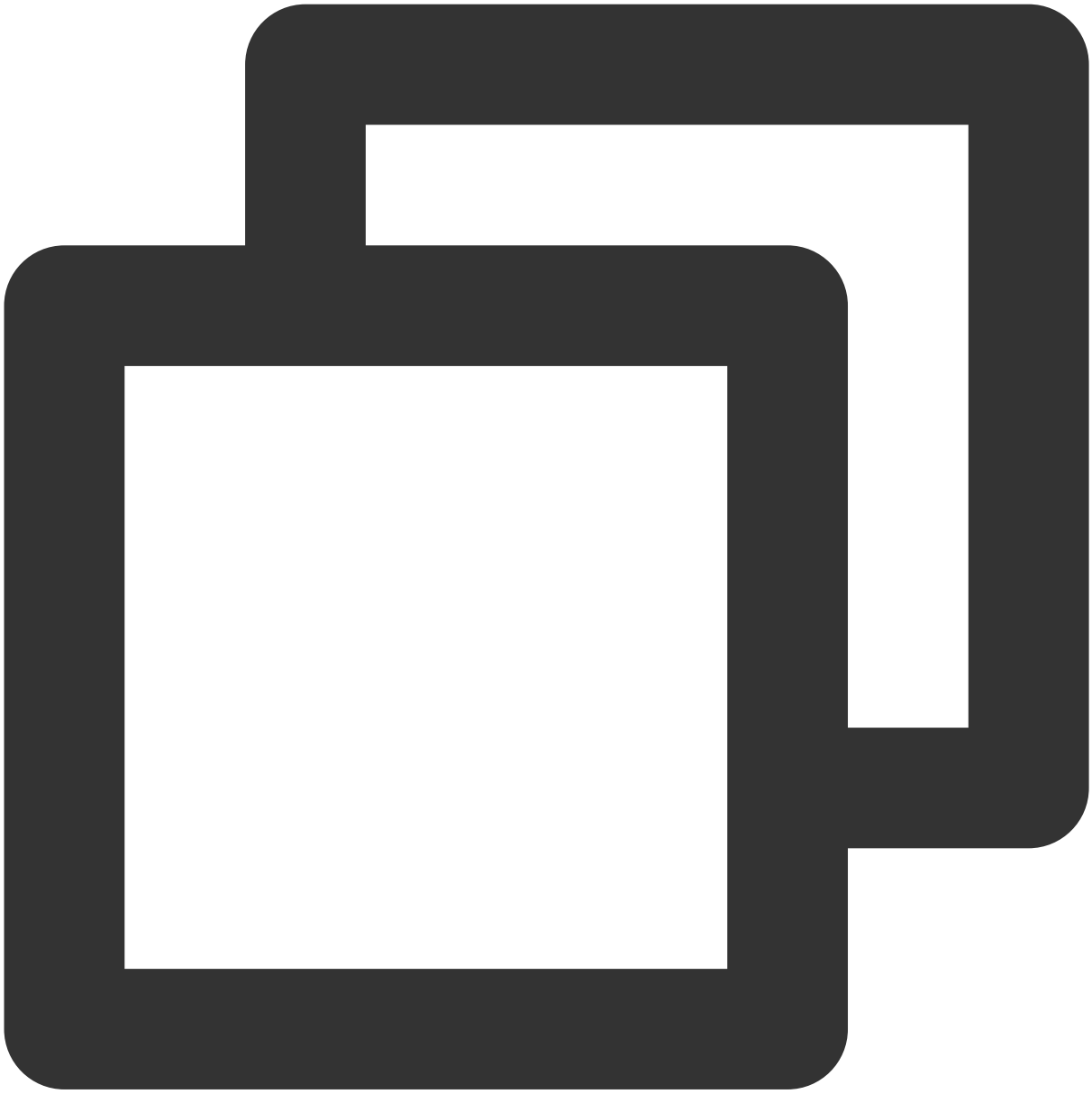
Kotlin

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。

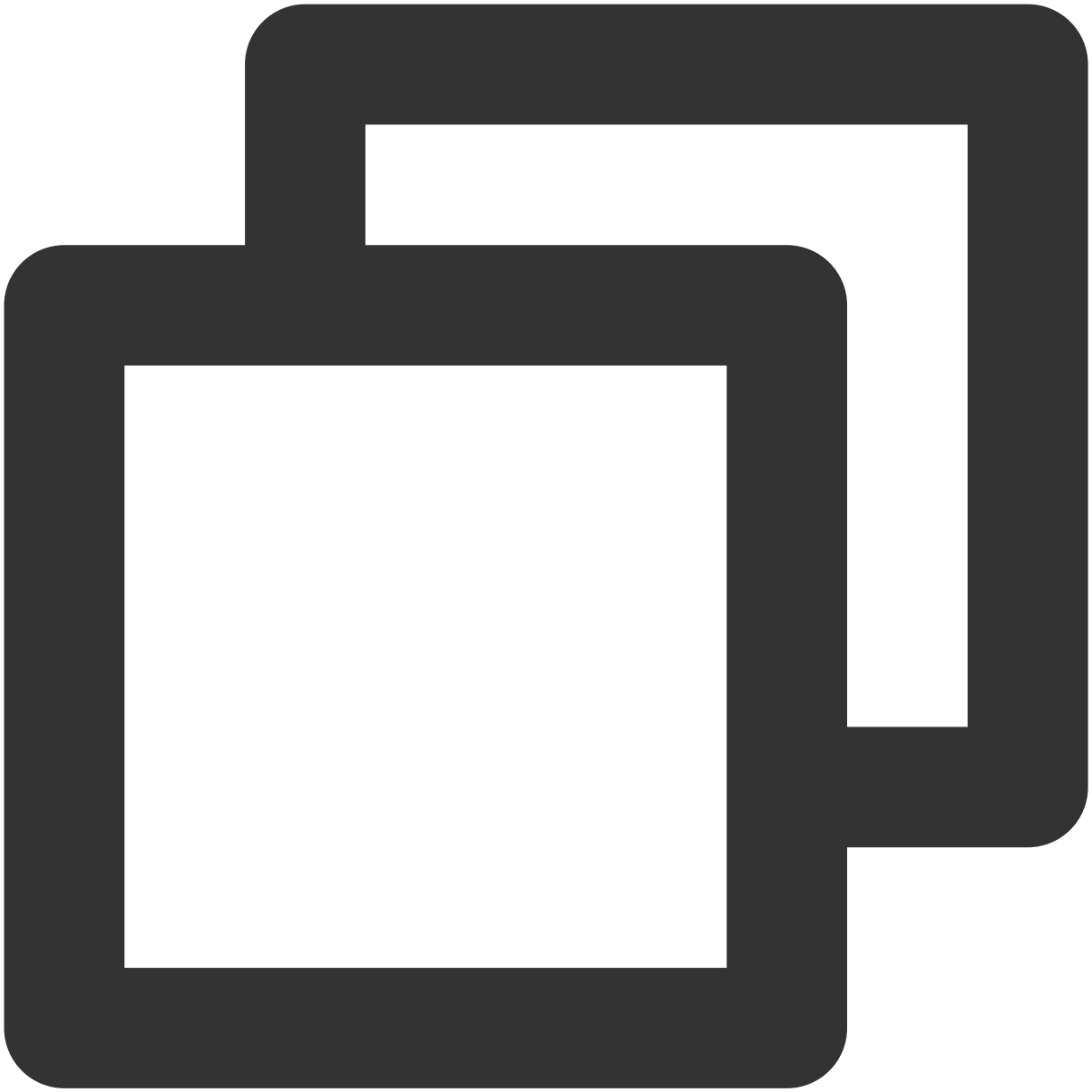
注意

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

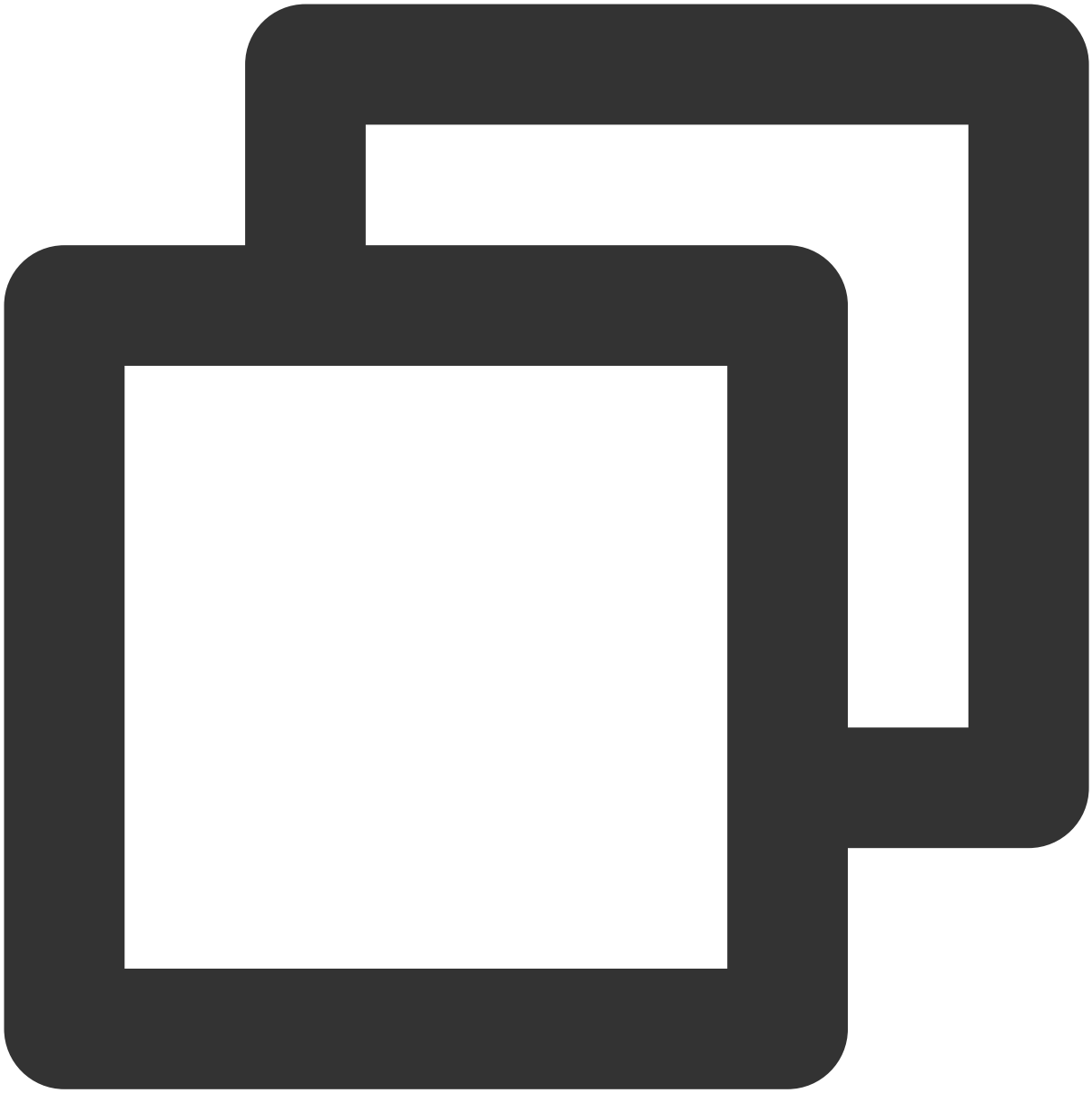
群组的创建详见：Chat 群组管理，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。

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。

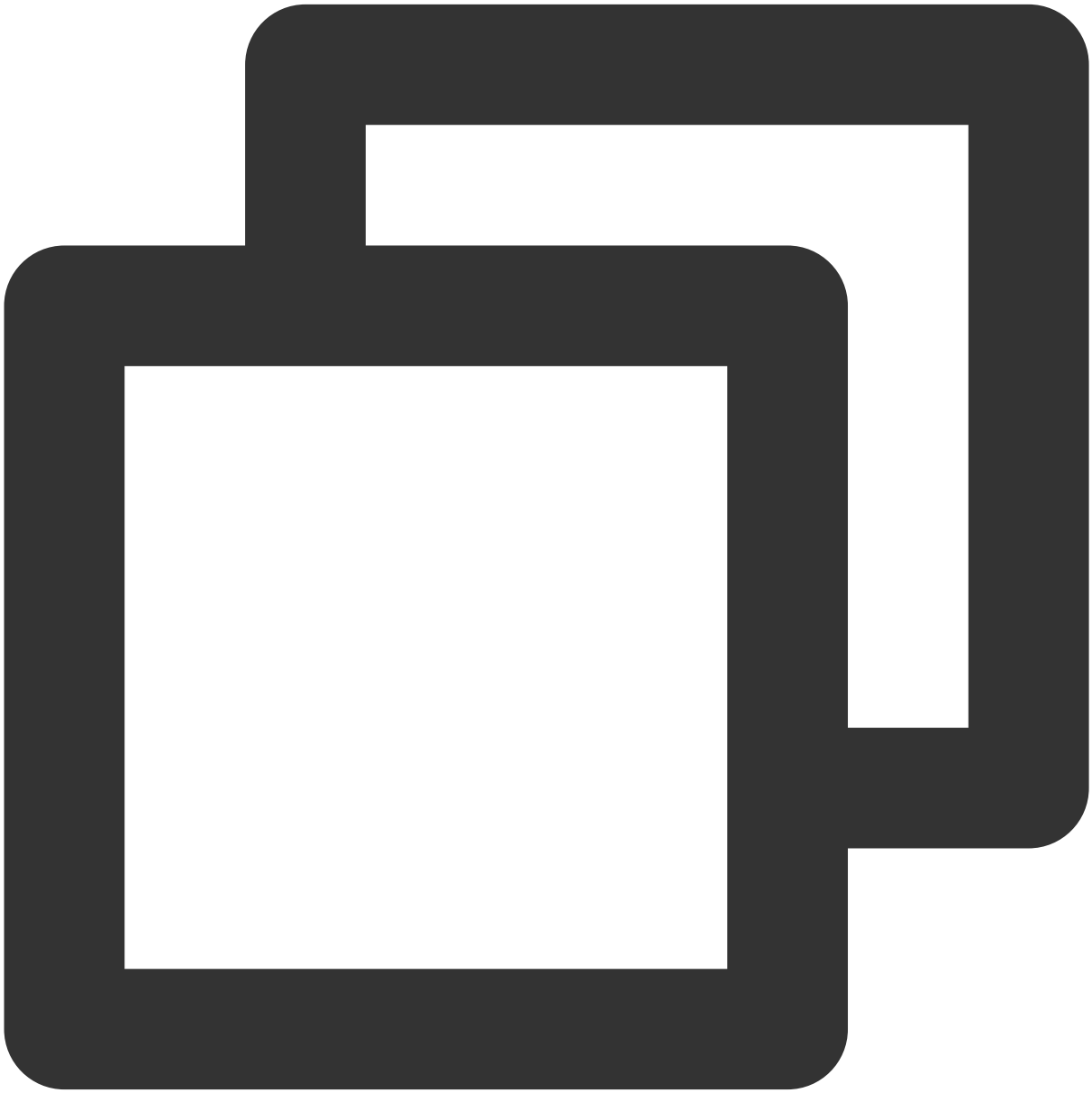
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

铃声设置后与设备绑定，更换用户，铃声依旧会生效。

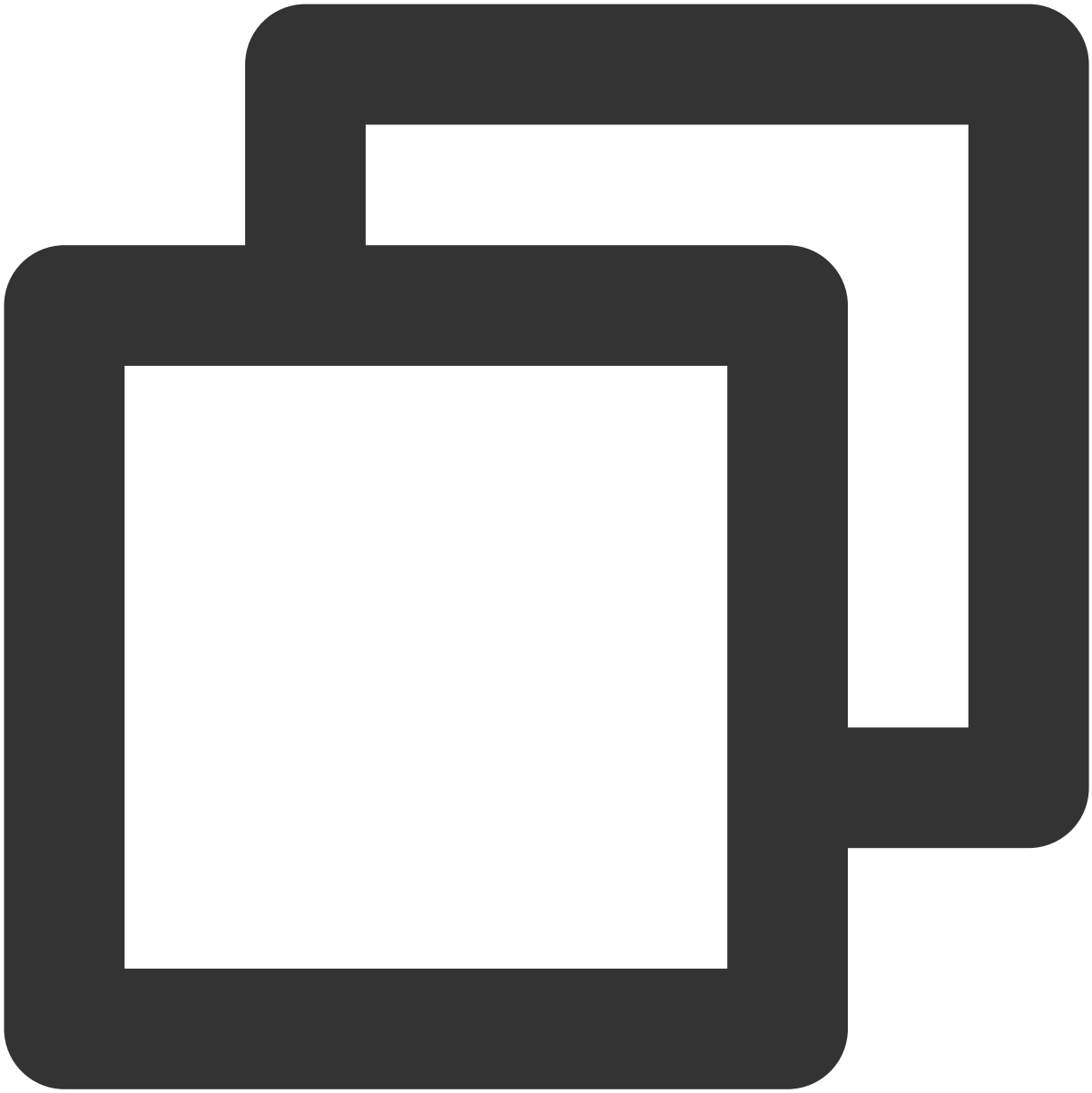
如需恢复默认铃声， `filePath` 传空即可。

Kotlin

Java



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



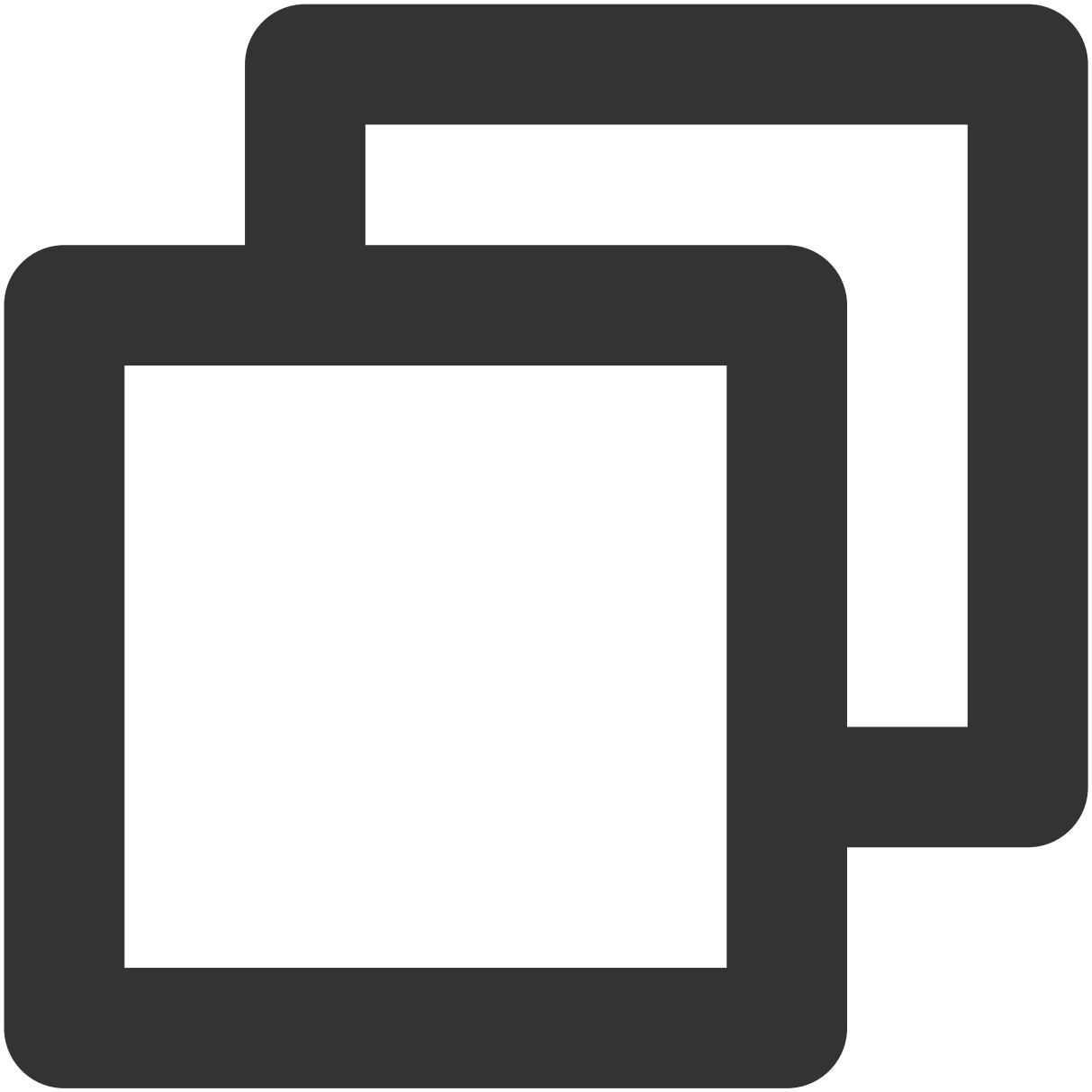
```
void setCallingBell(String filePath);
```

enableMuteMode

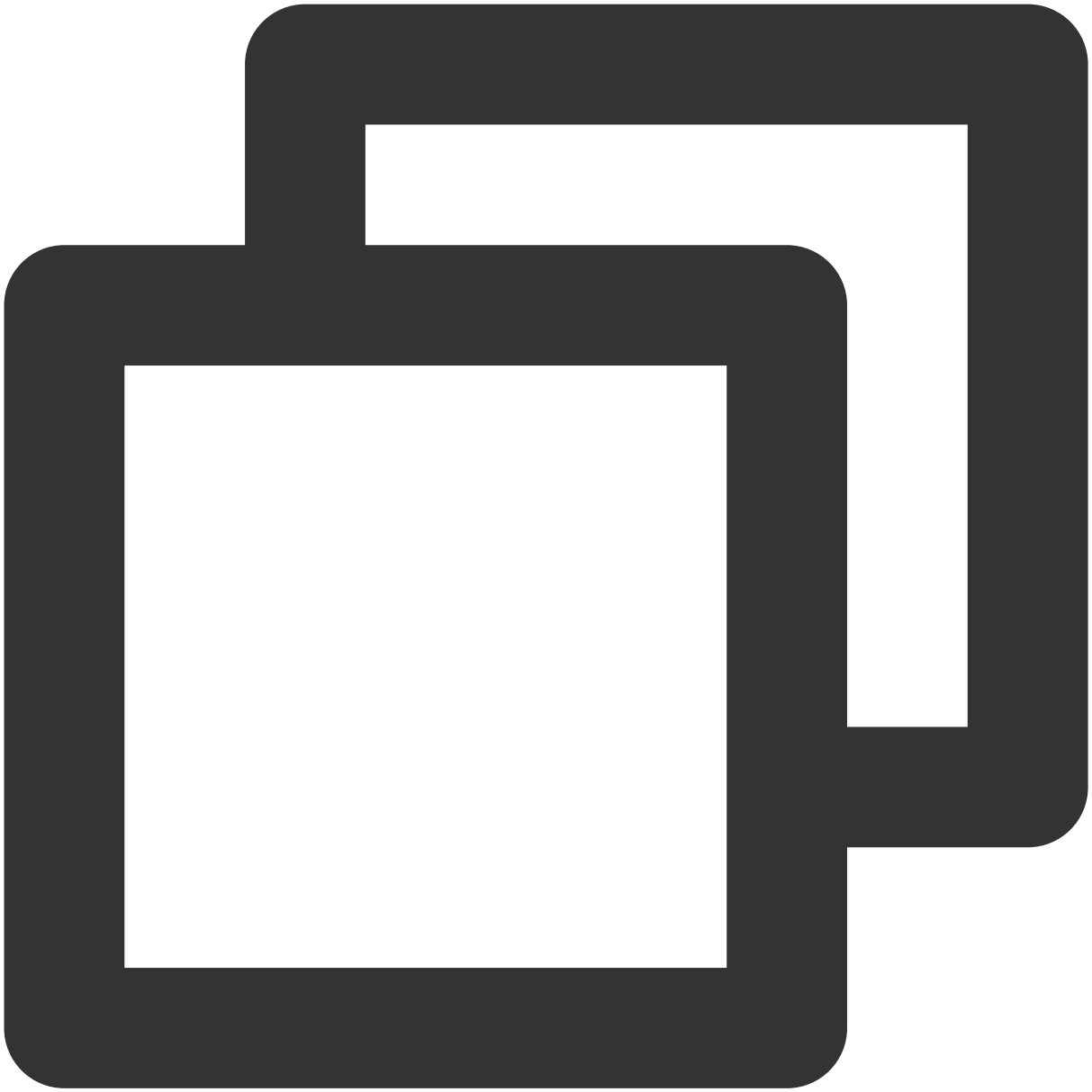
开启后，收到通话请求，不会播放来电铃声。

Kotlin

Java



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



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

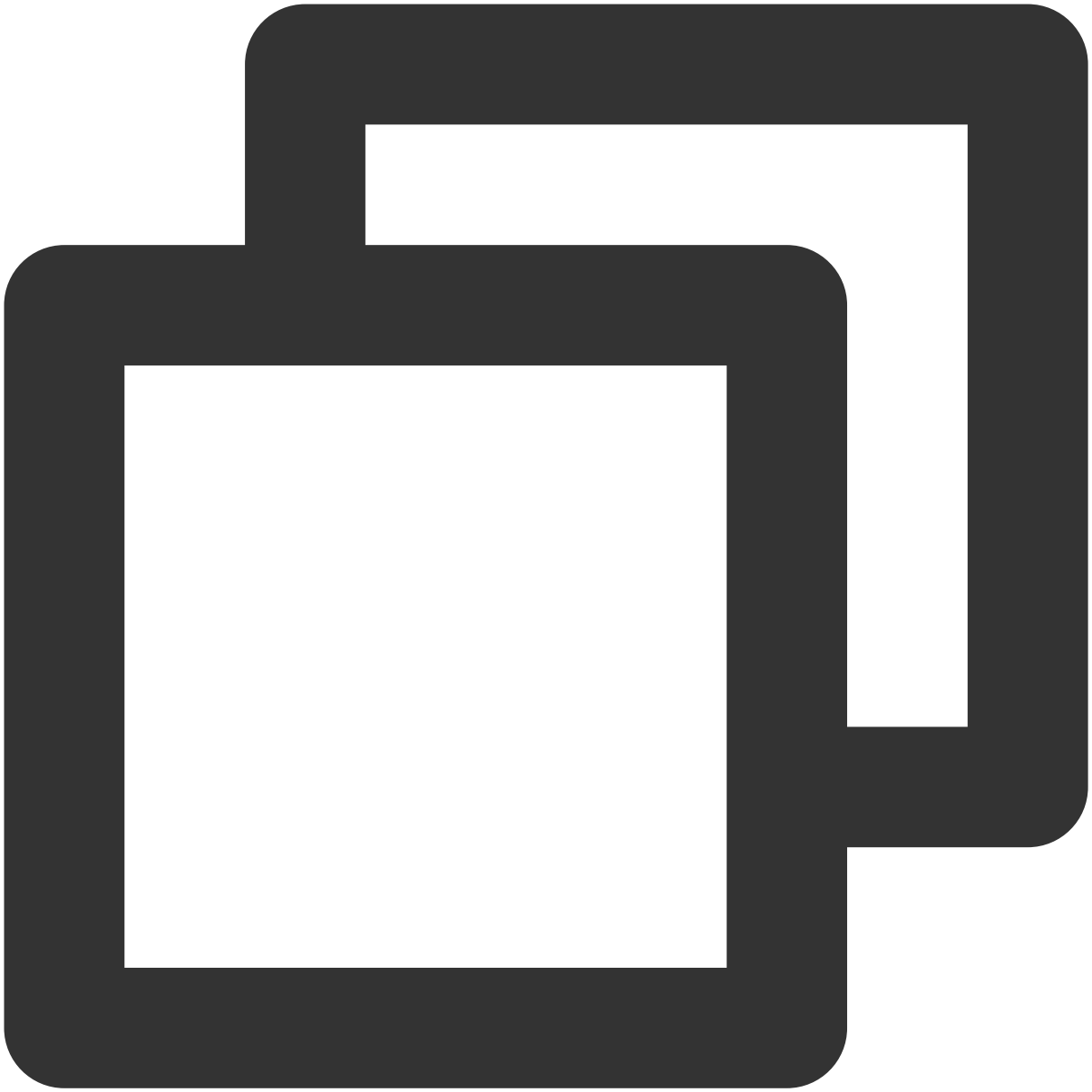
enableFloatWindow

开启/关闭悬浮窗功能。

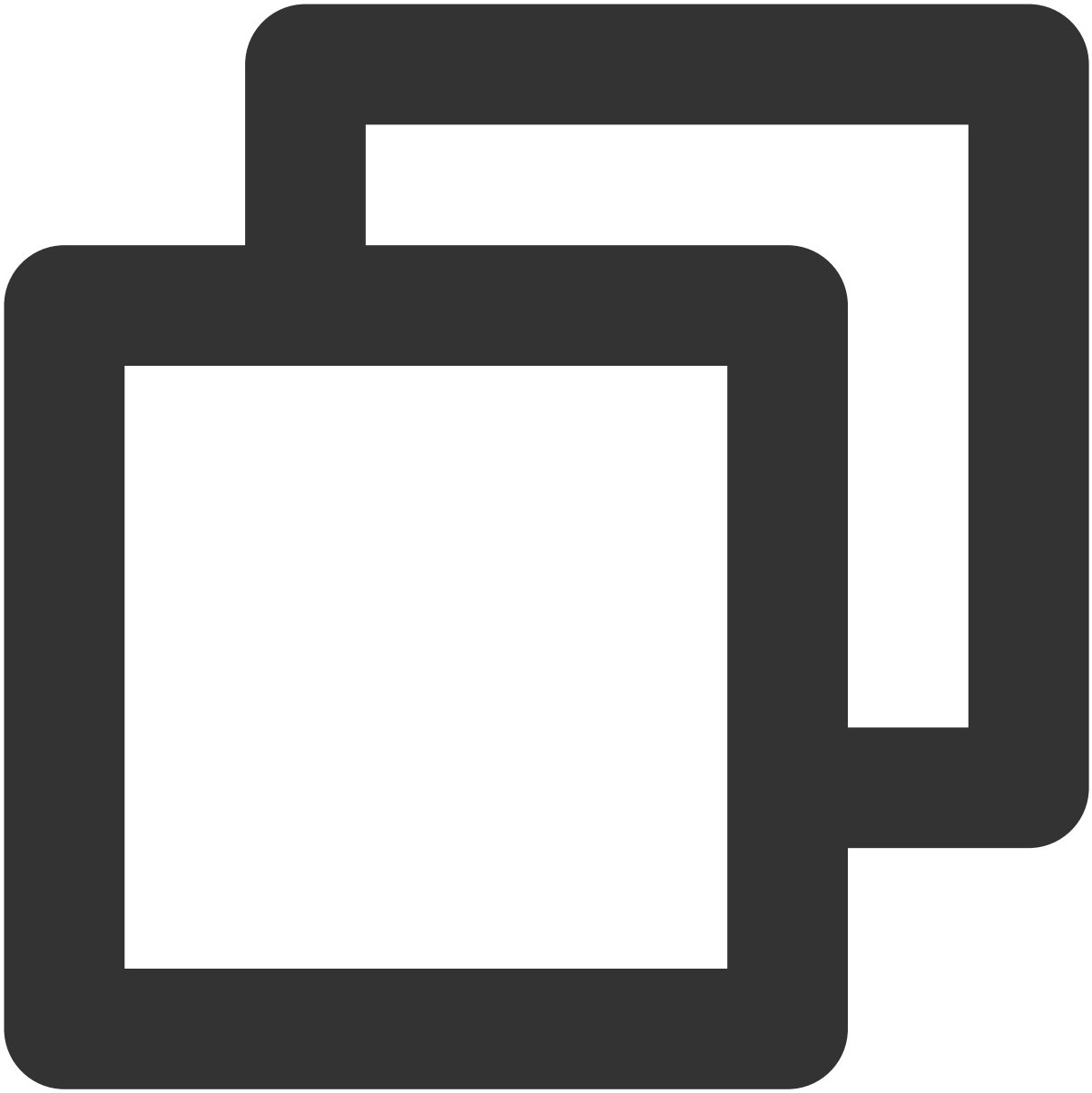
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Kotlin

Java



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

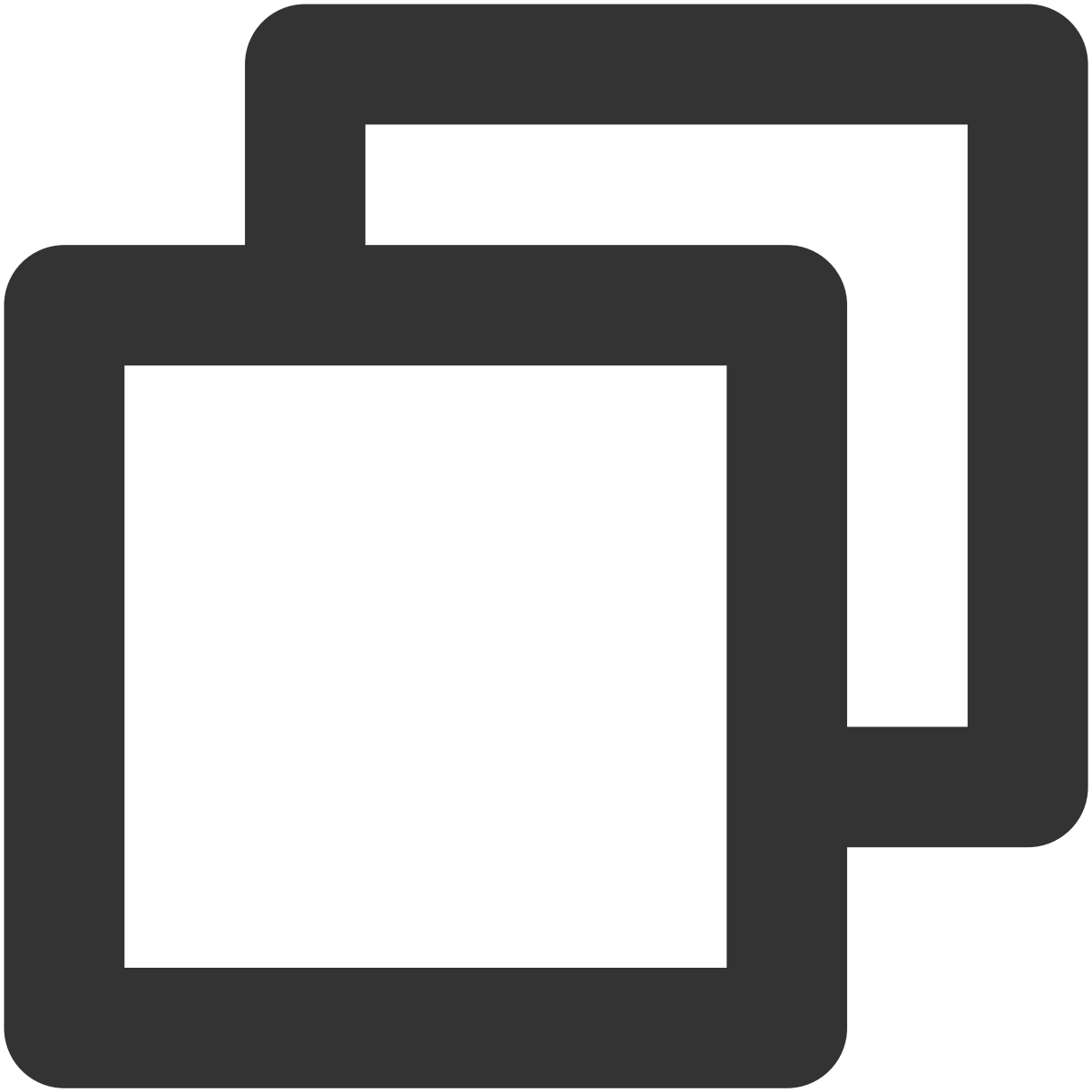


```
void enableFloatWindow(boolean enable);
```

enableIncomingBanner

开启/关闭来电横幅显示。

默认为 `false` ，被叫端收到邀请后默认弹出全屏通话等待界面，开启后先展示一个横幅，然后根据需要拉起全屏通话界面。



```
fun enableIncomingBanner(enable: Boolean)
```

TUICallEngine

最近更新时间：2024-07-25 17:49:30

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

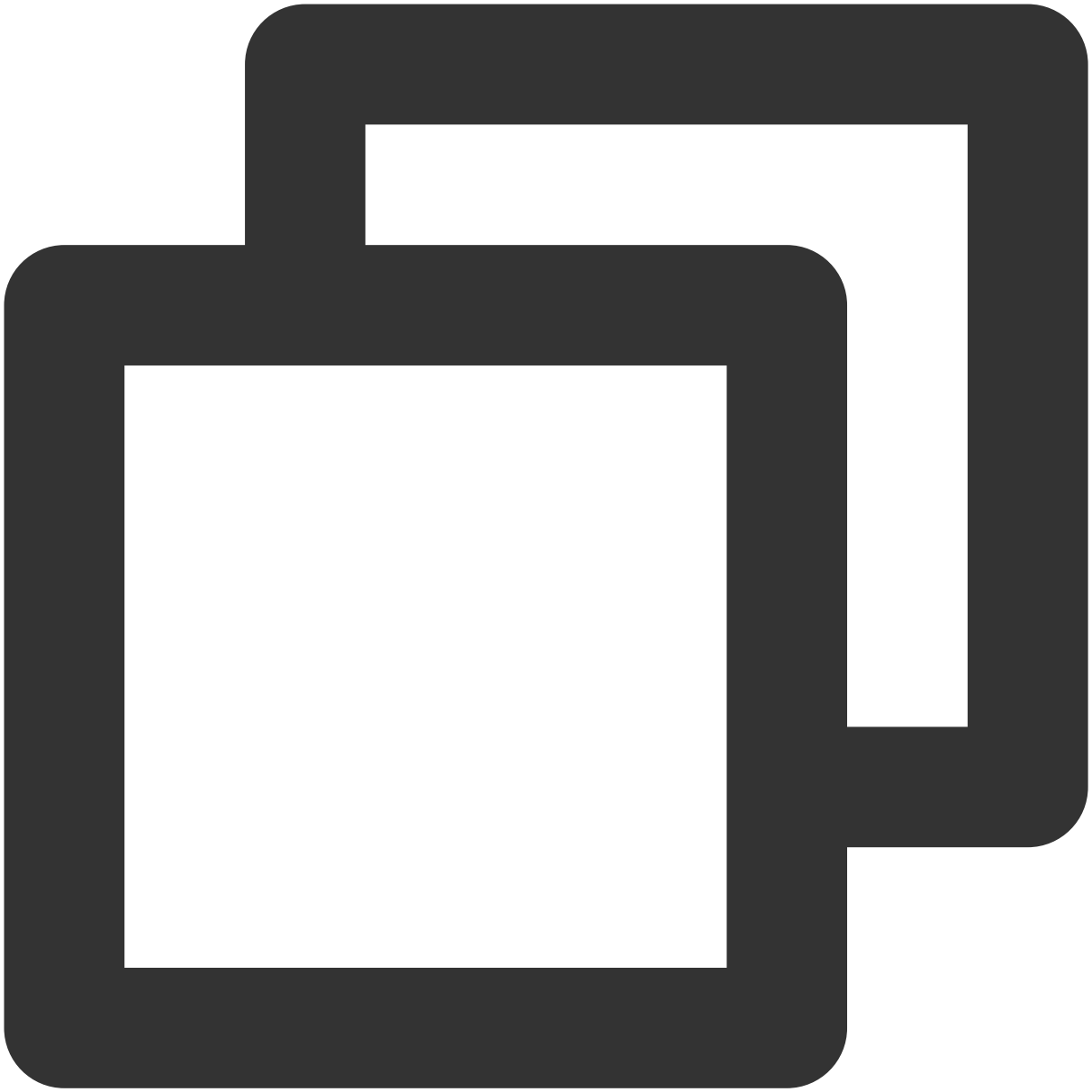
版权所有：腾讯云计算（北京）有限责任公司

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（群组通话版套餐支持）
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

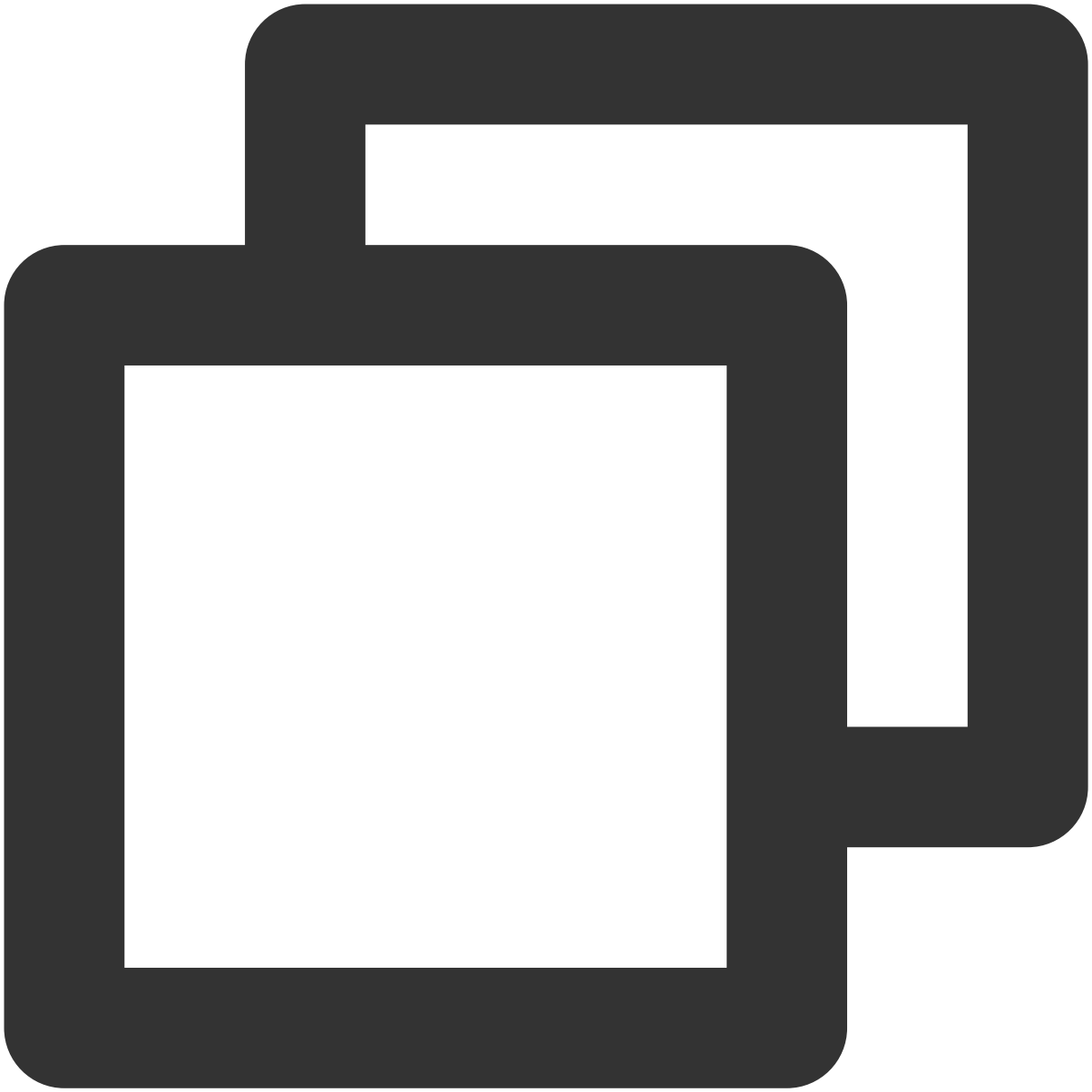
创建 TUICallEngine 的单例。



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

destroyInstance

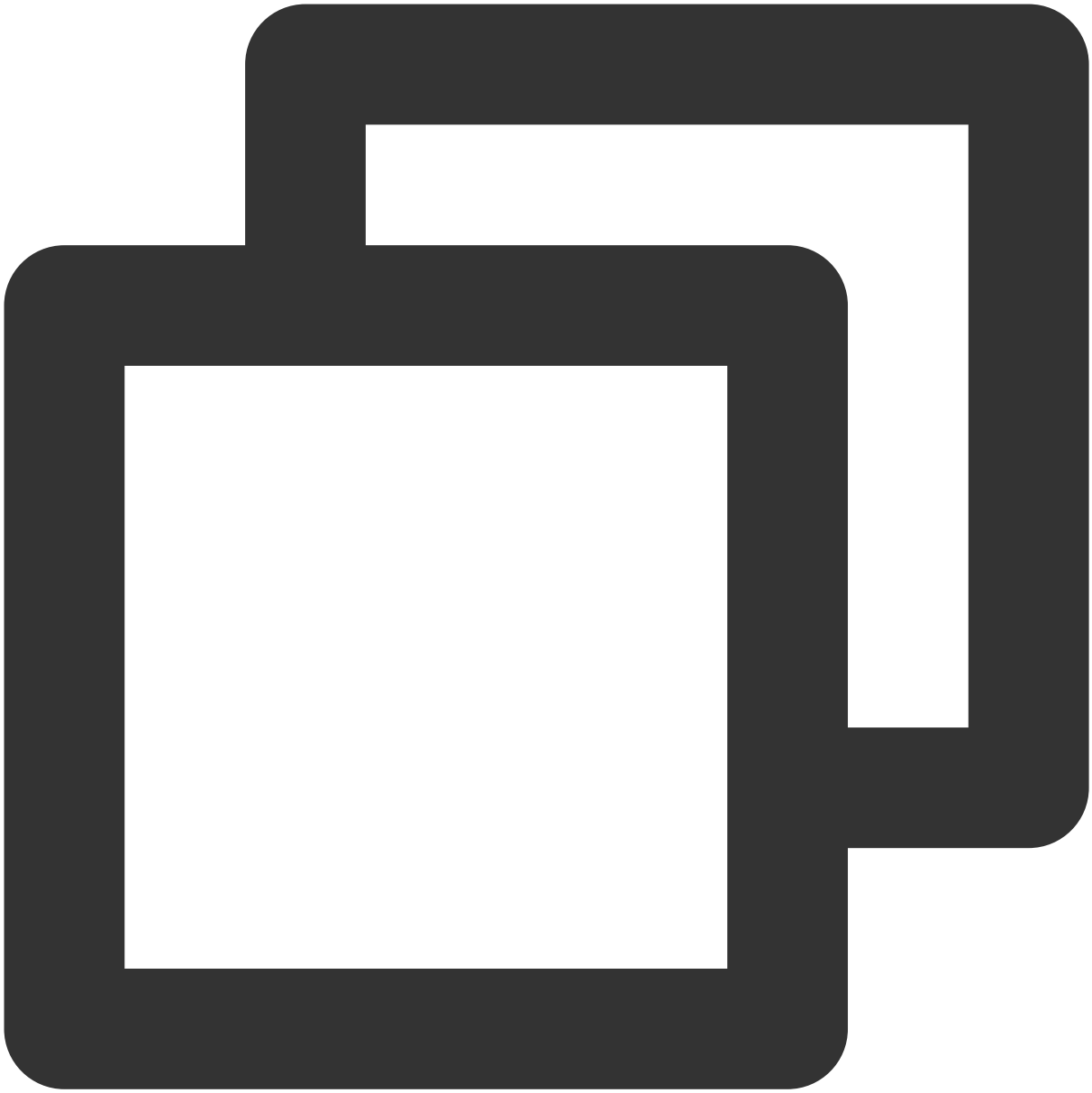
销毁 TUICallEngine 的单例。



```
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
```

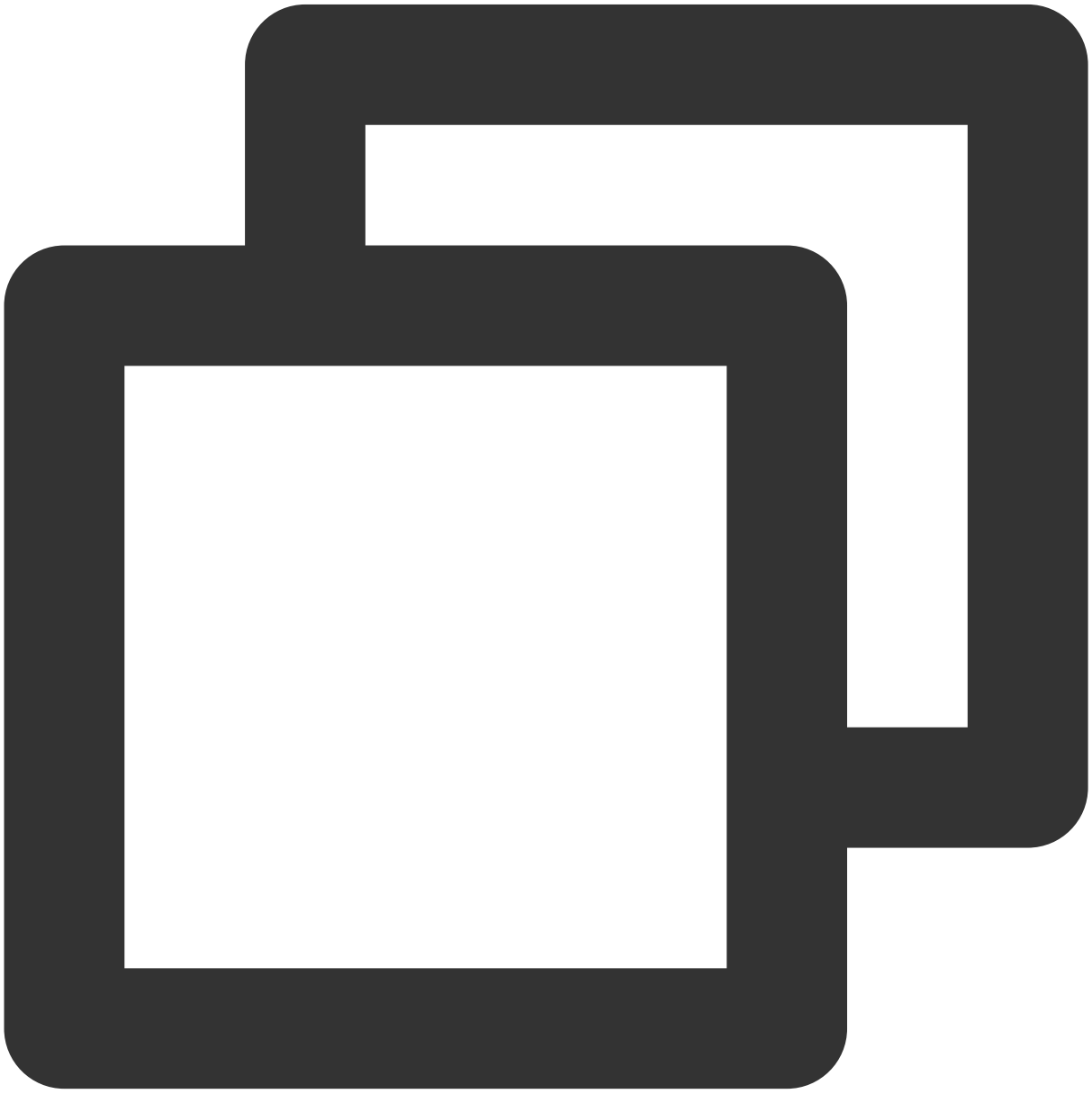
参数如下表所示：

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

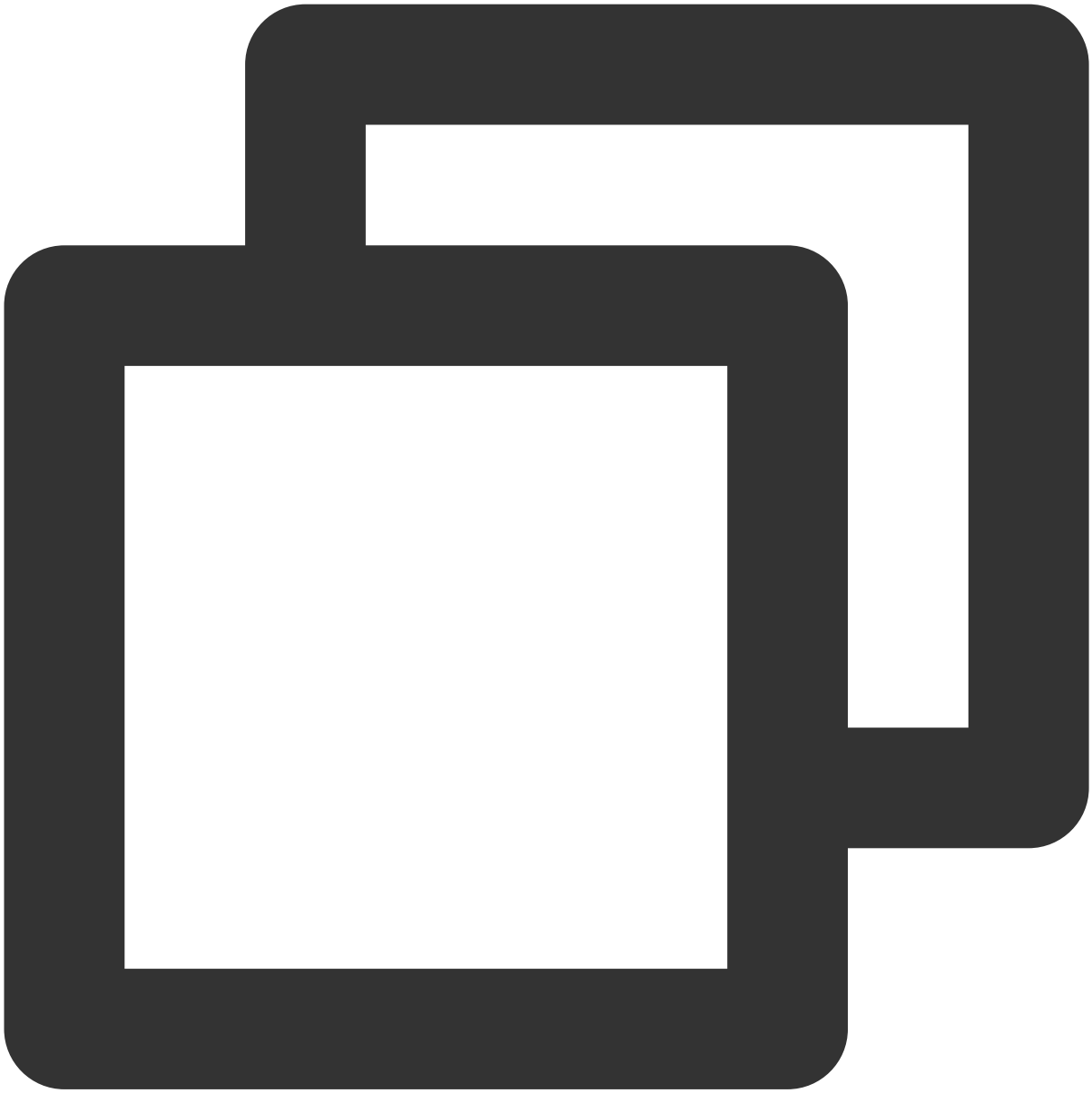
添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。



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

removeObserver

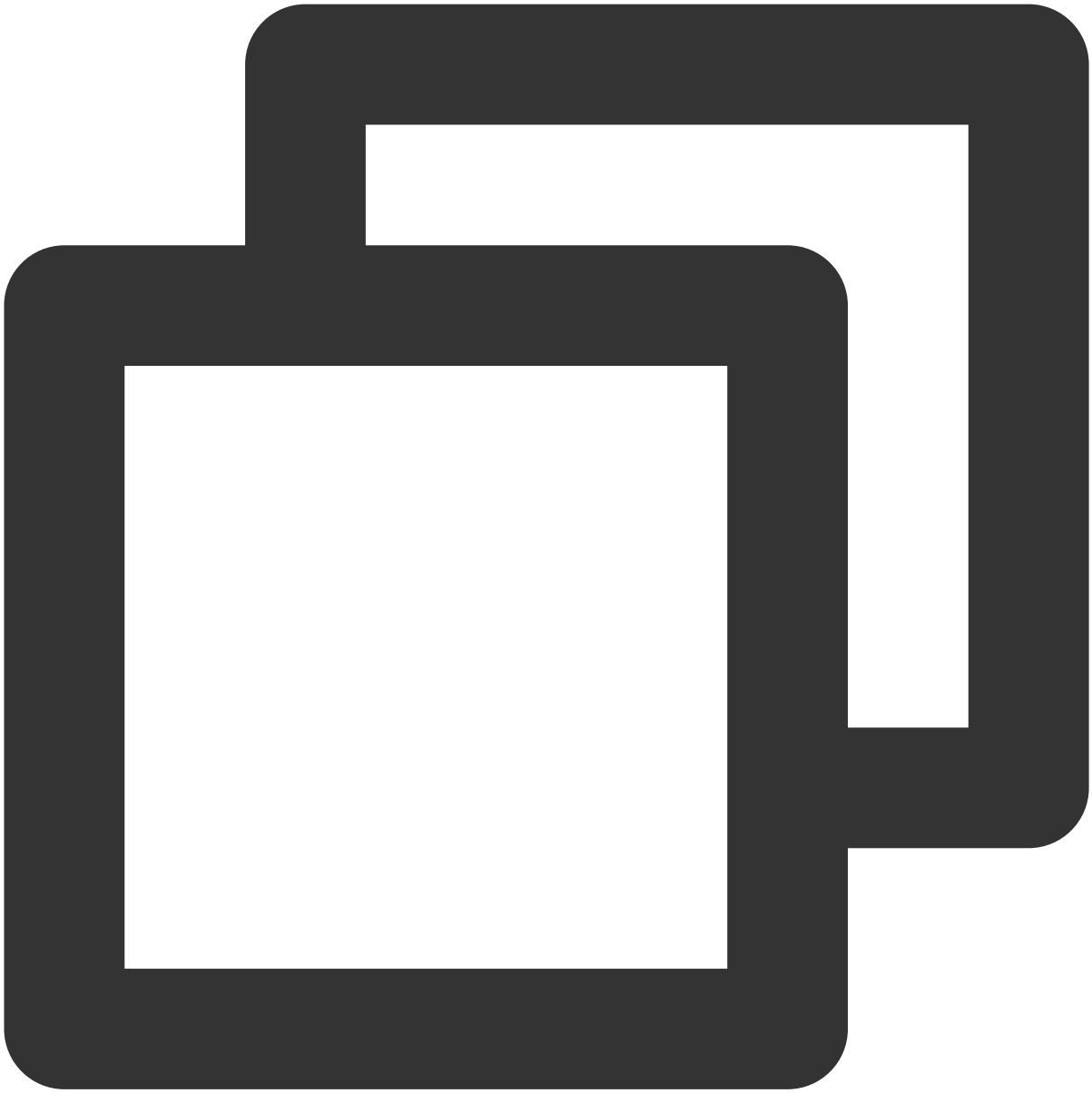
移除回调接口。



```
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

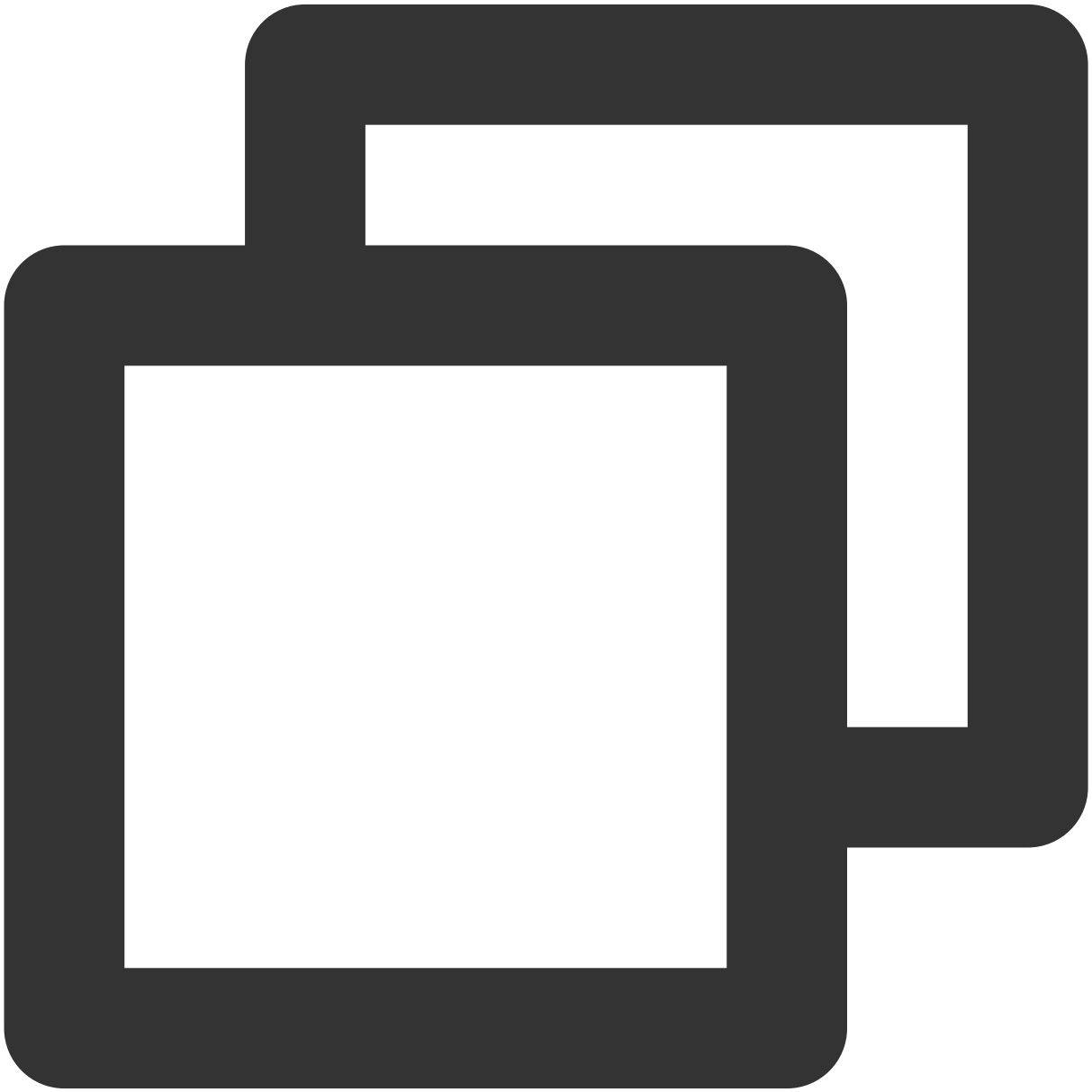
params	TUICallDefine.CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等

groupCall

发起群组通话。

注意：

使用群组通话前需要创建 Chat 群组，如果已经创建，请忽略。



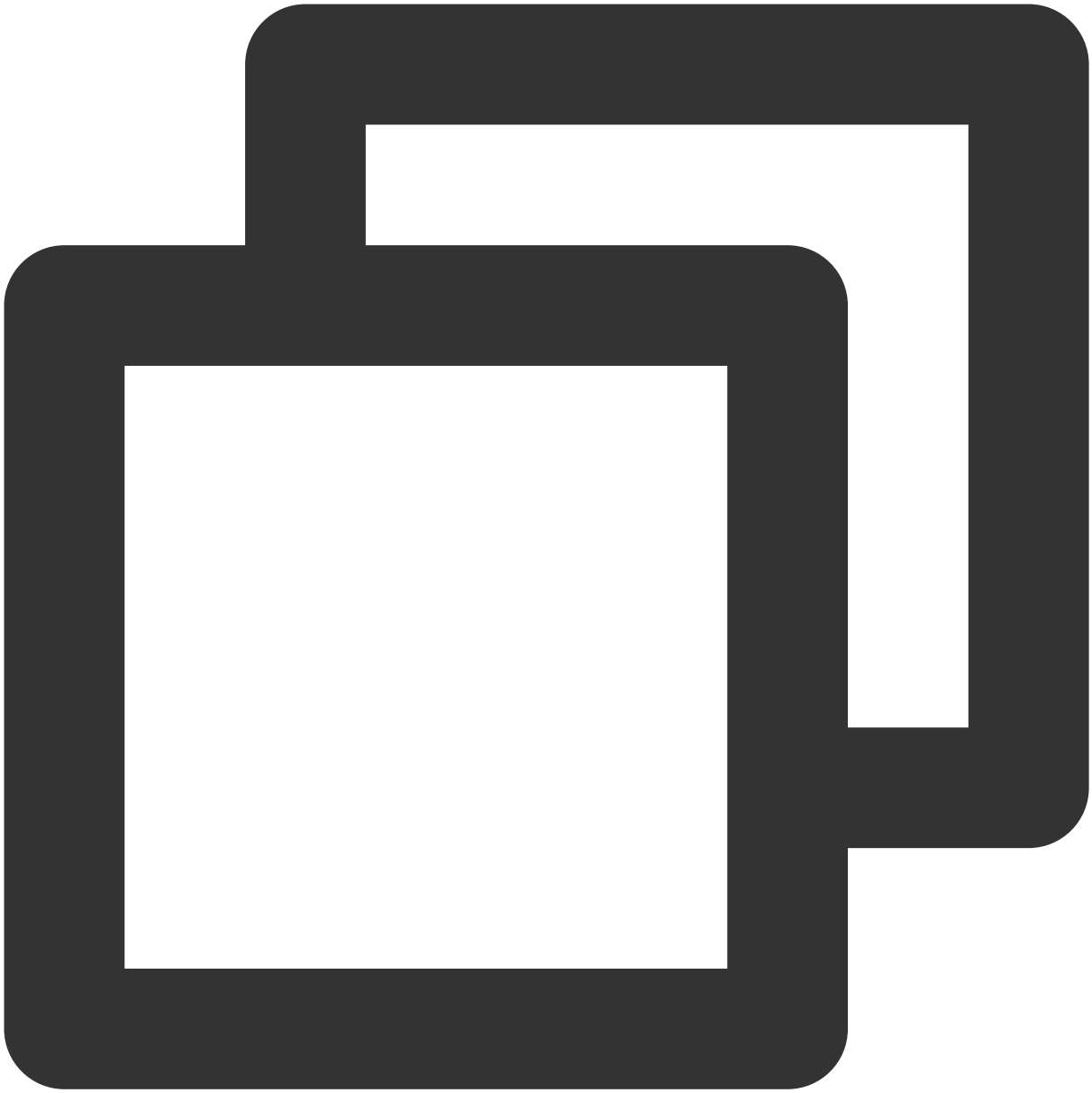
```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType cal
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

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

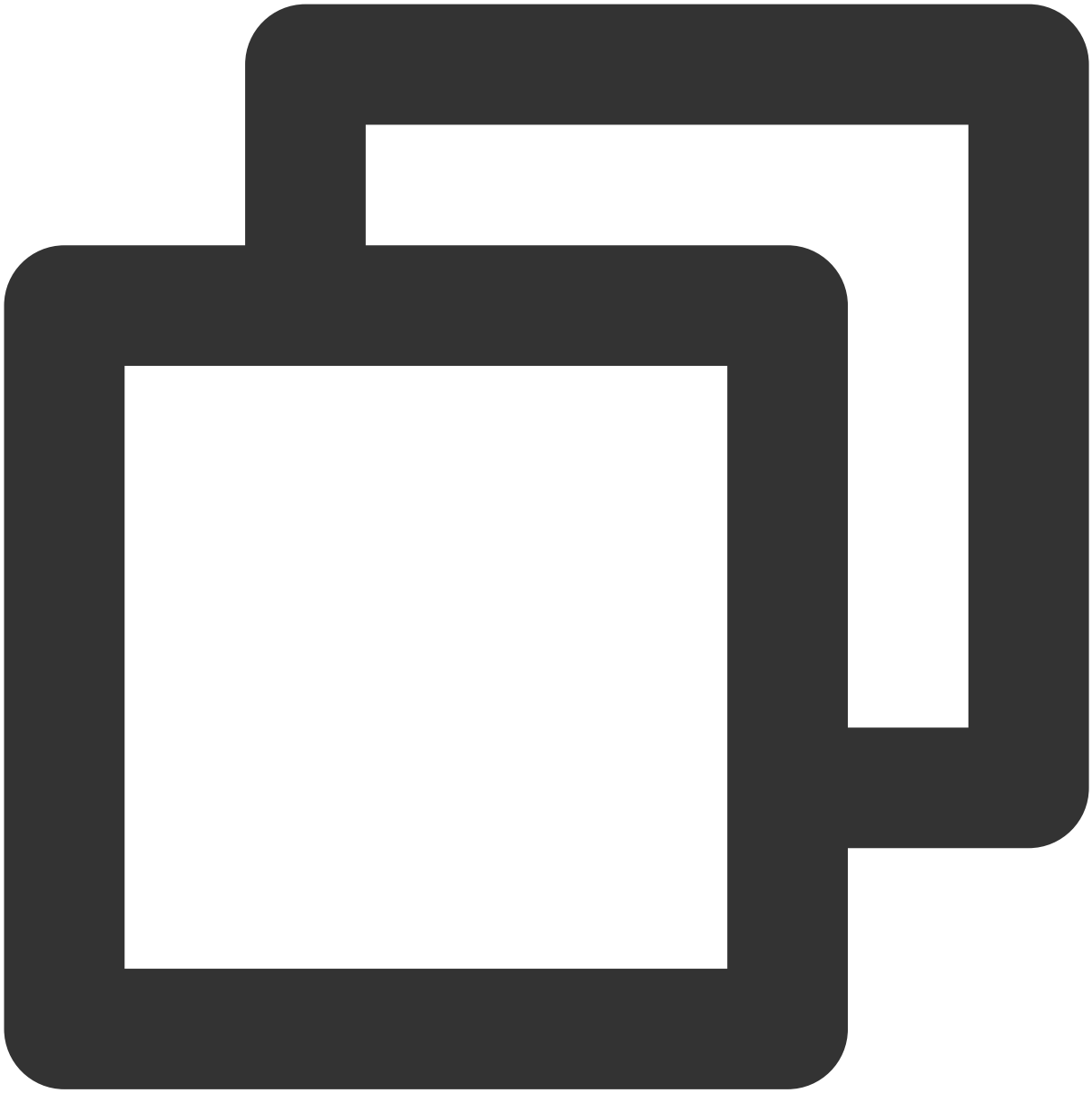
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



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

reject

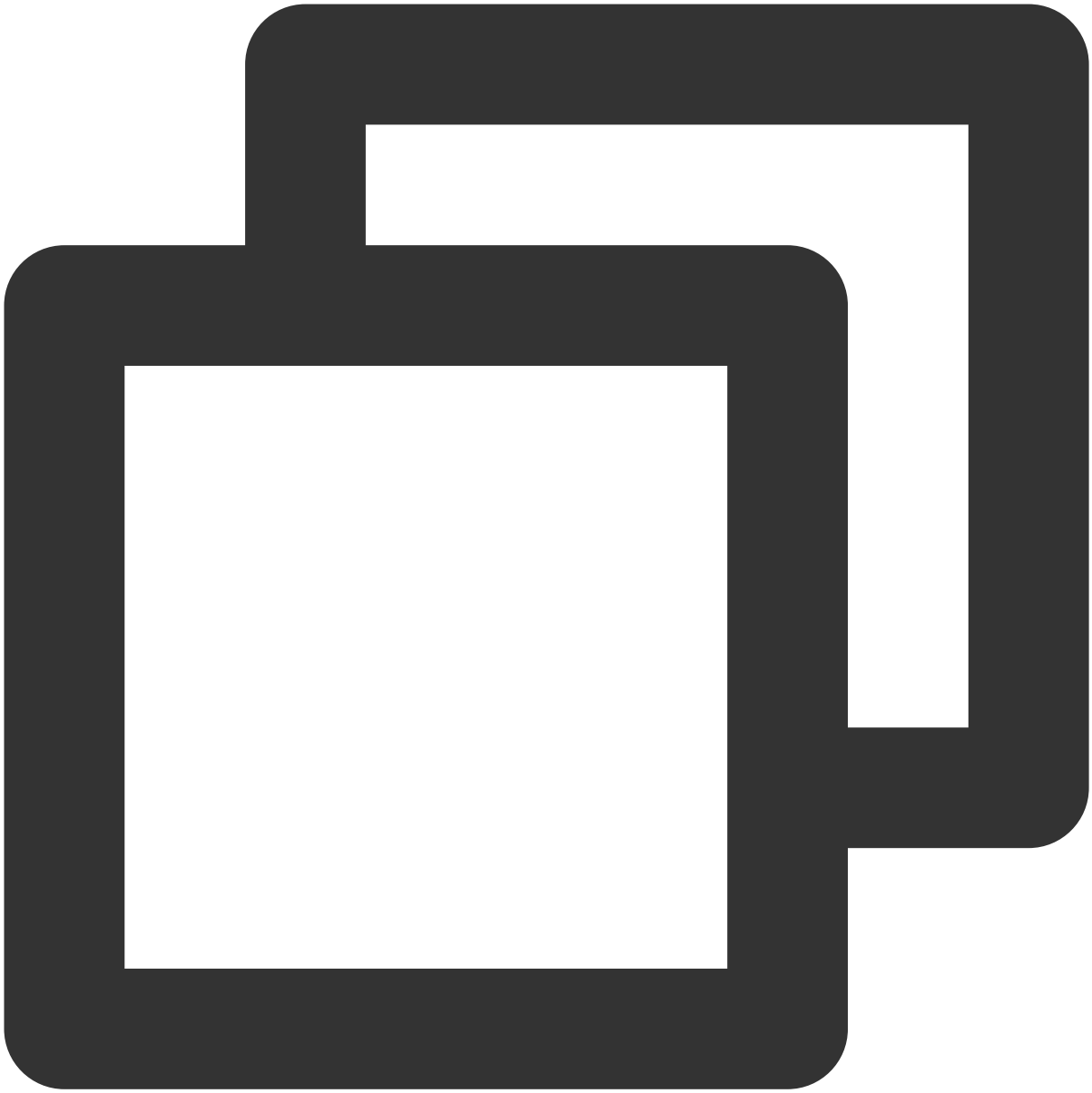
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



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

ignore

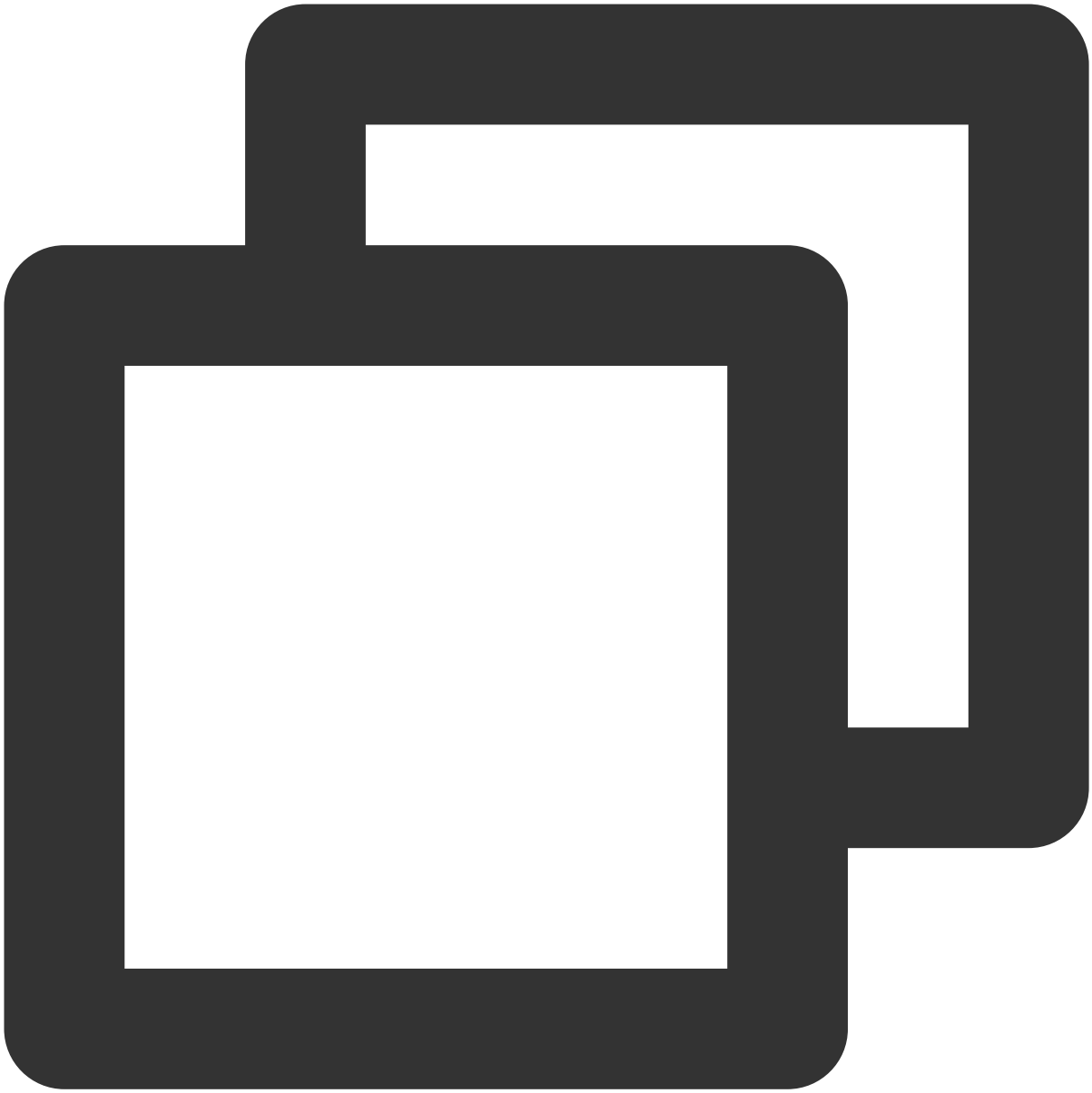
忽略当前通话，当您作为被叫收到 onCallReceived() 的回调时，可以调用该函数忽略来电，此时主叫会收到 onUserLineBusy 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
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。

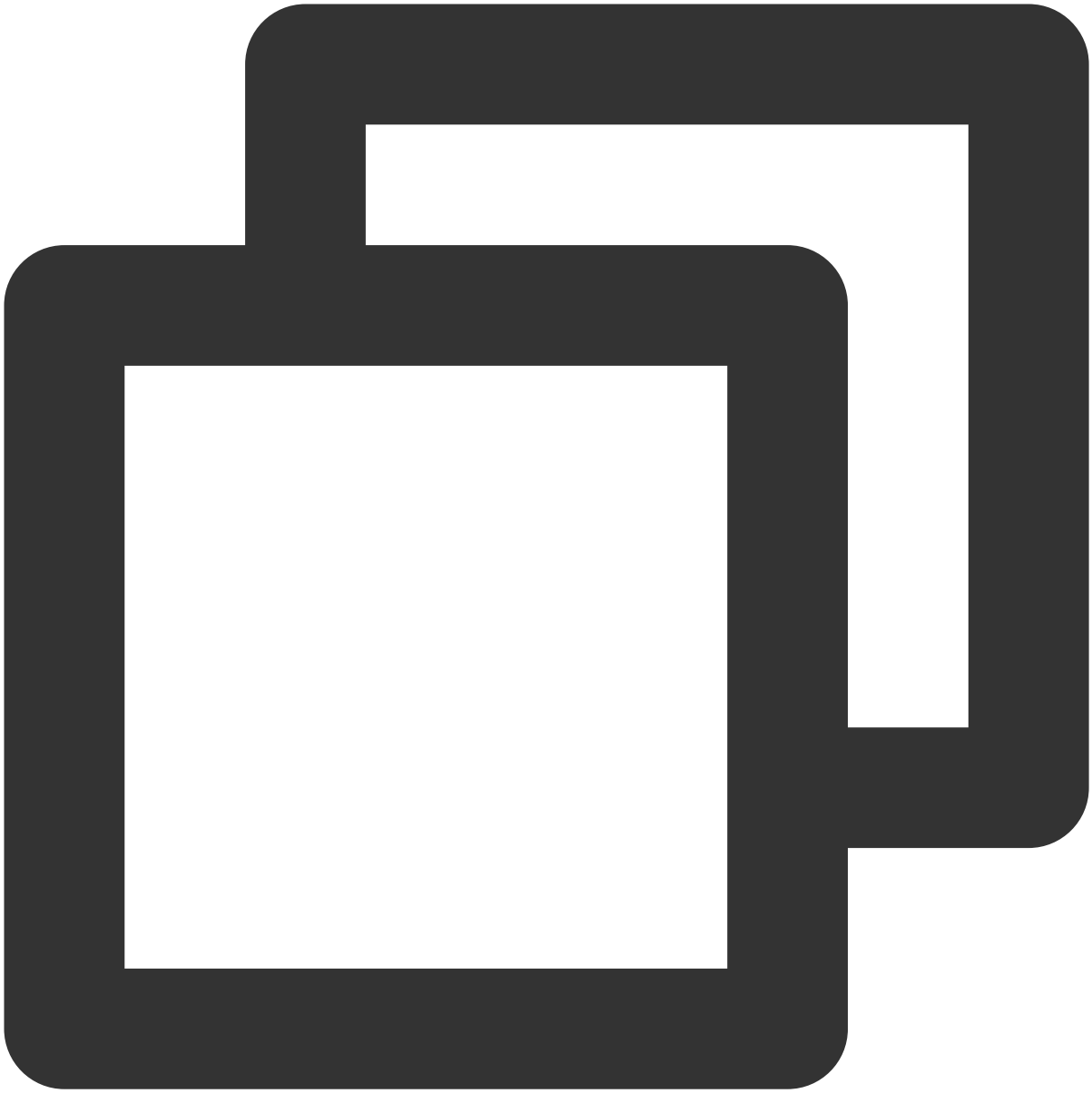


```
void hangup(TUICommonDefine.Callback callback);
```

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);
```

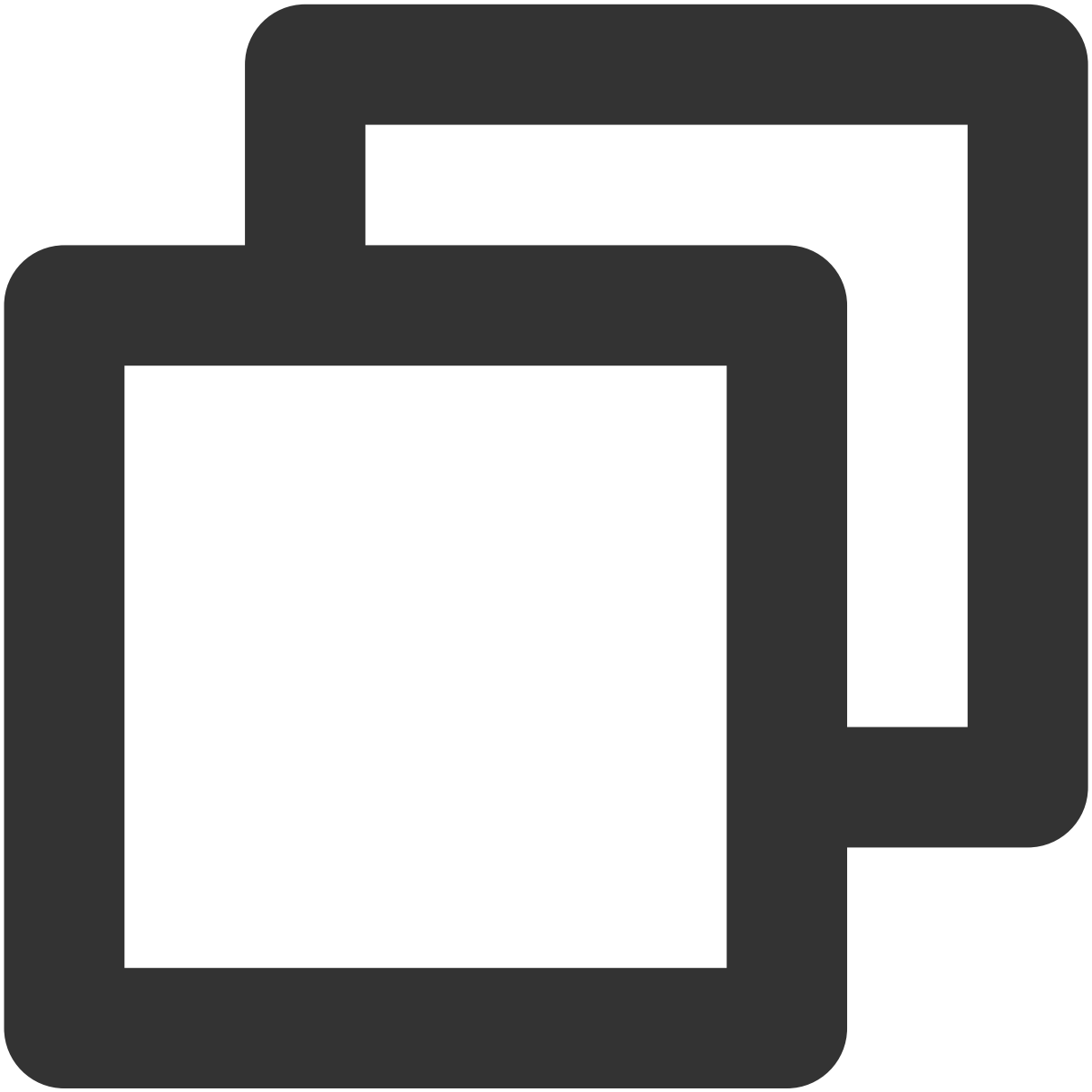
参数如下表所示：

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

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
```

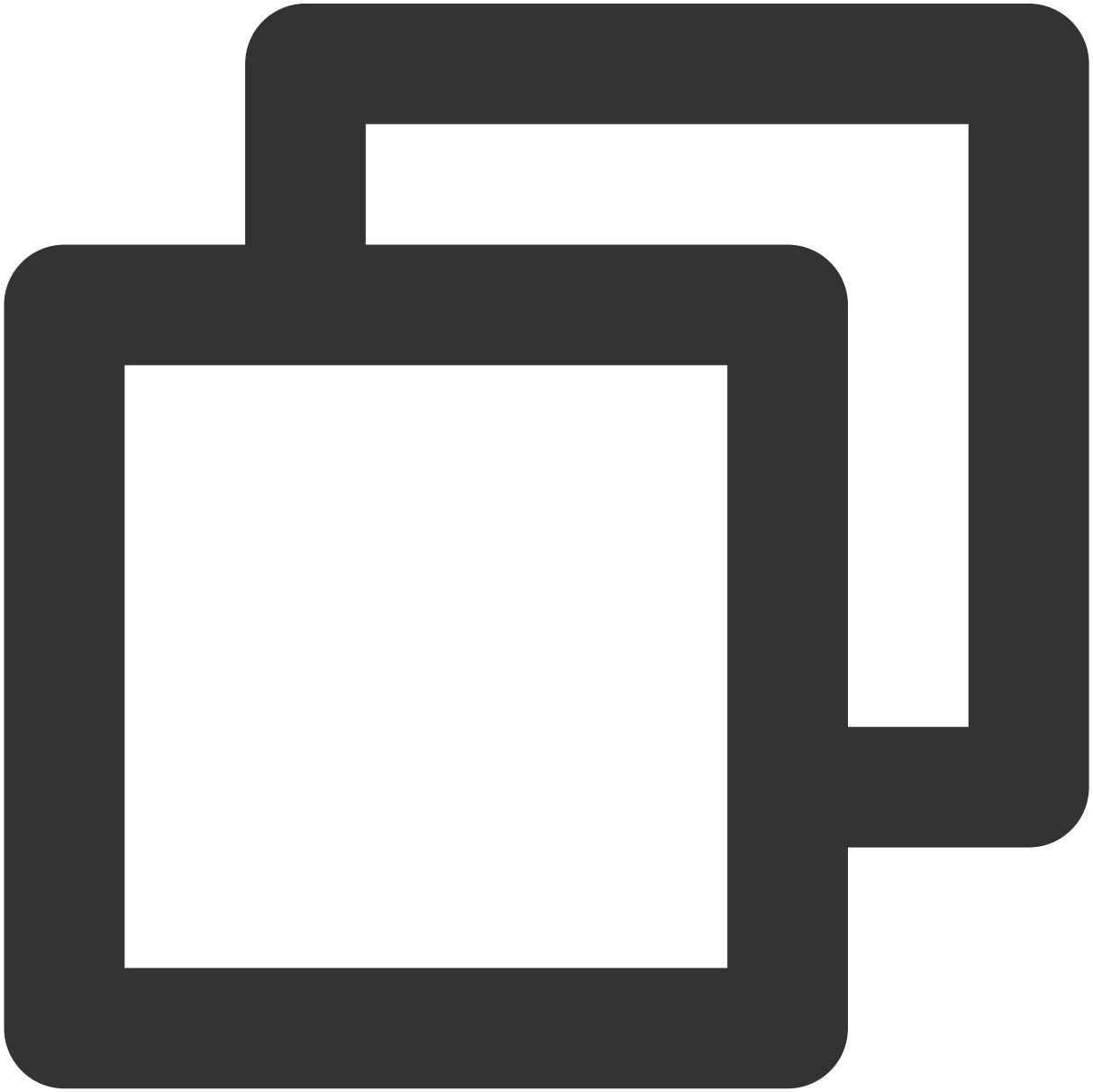
参数如下表所示：

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。



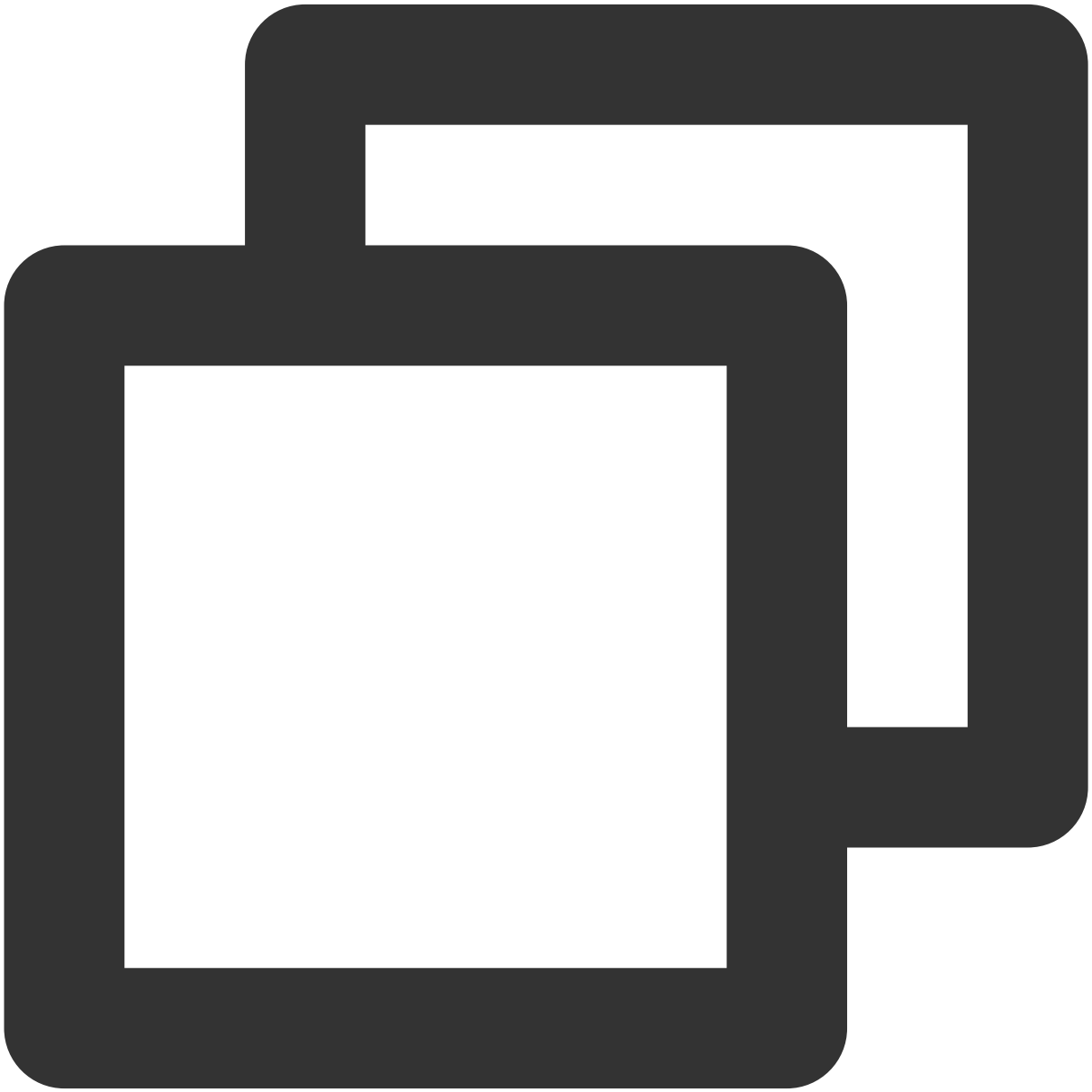
```
void switchCallMediaType(TUICallDefine.MediaType callMediaType);
```

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。



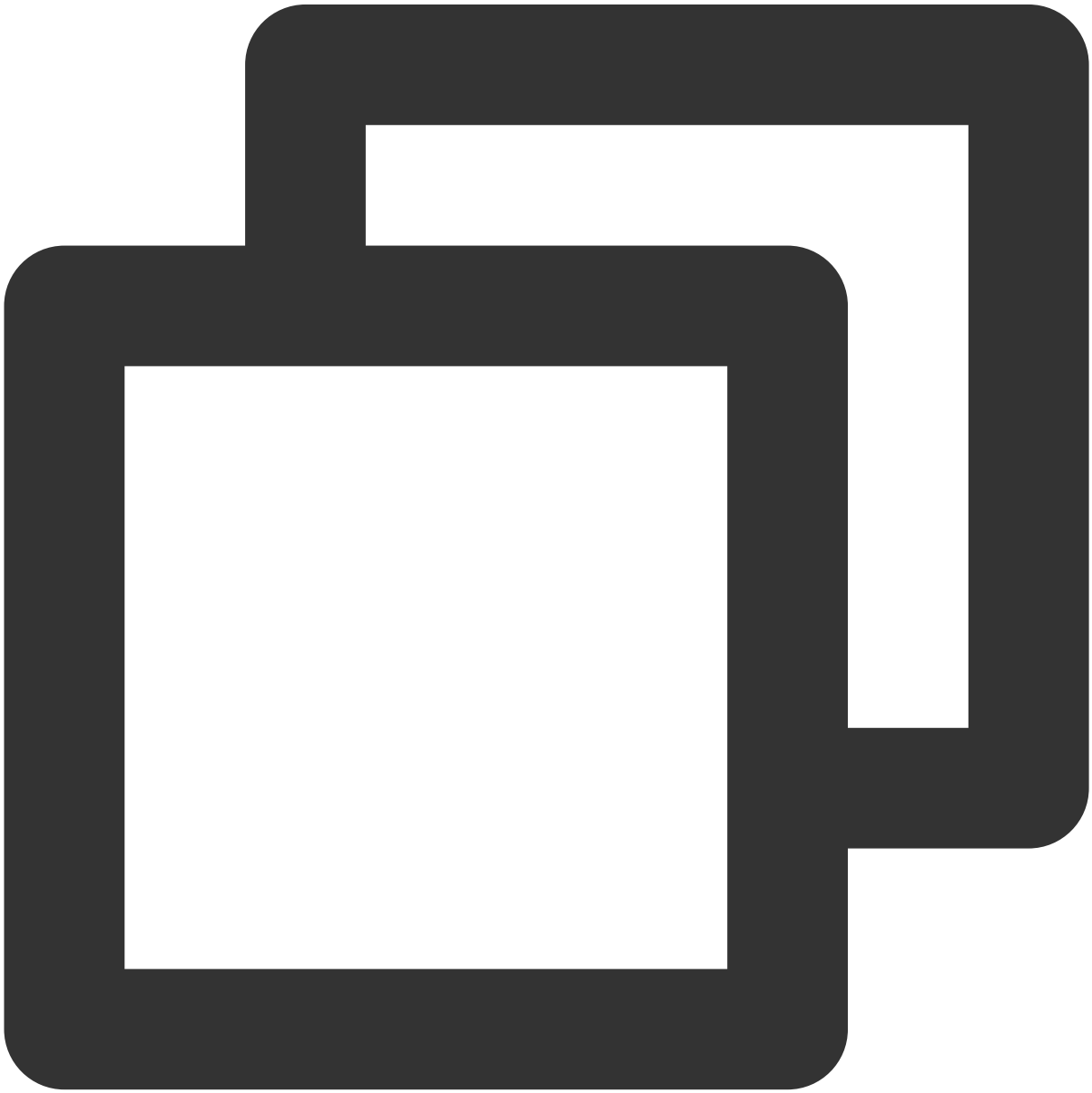
```
void startRemoteView(String userId, TUIVideoView videoView, TUICommonDefine.PlayCal
```

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。



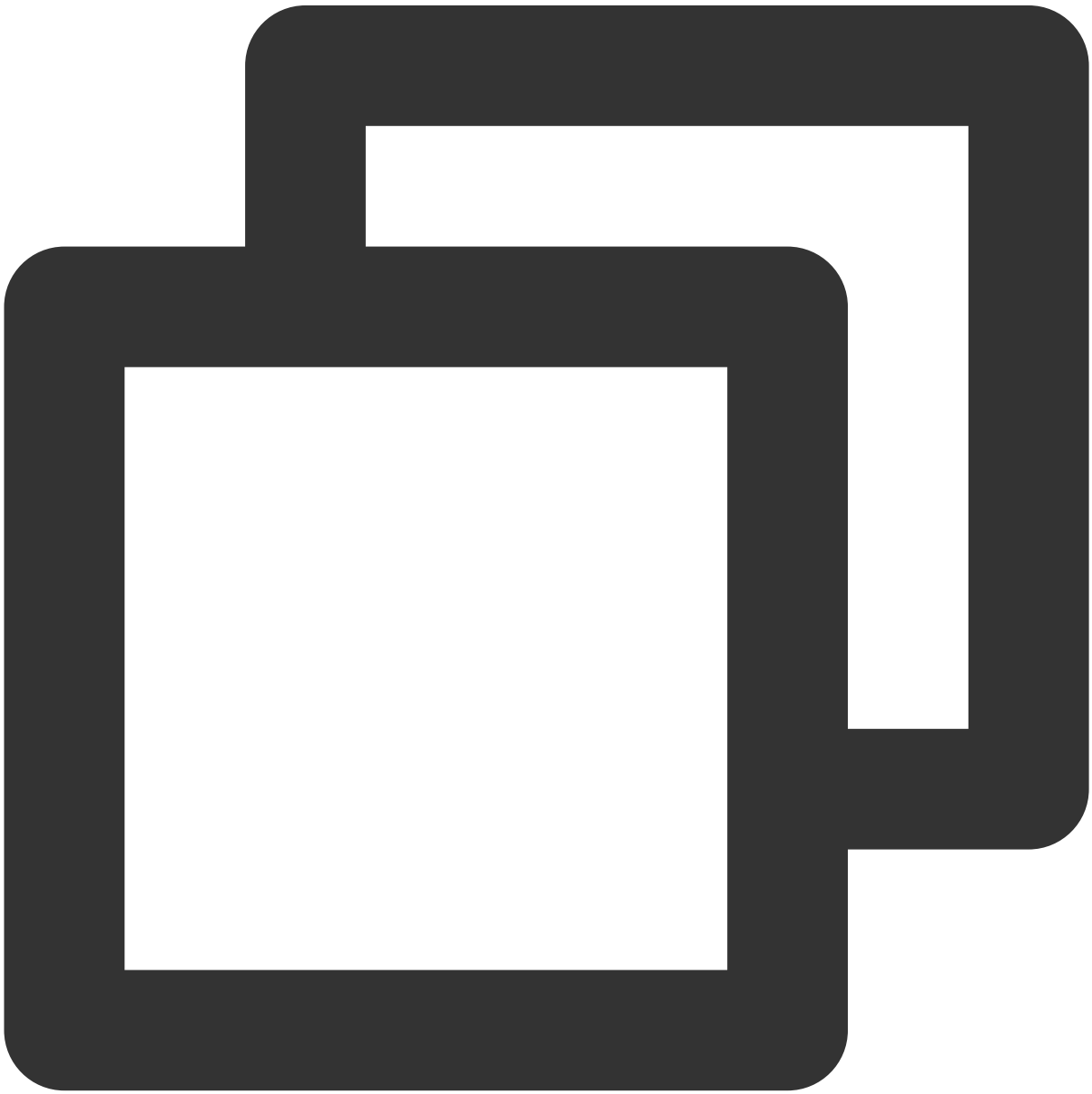
```
void stopRemoteView(String userId);
```

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。



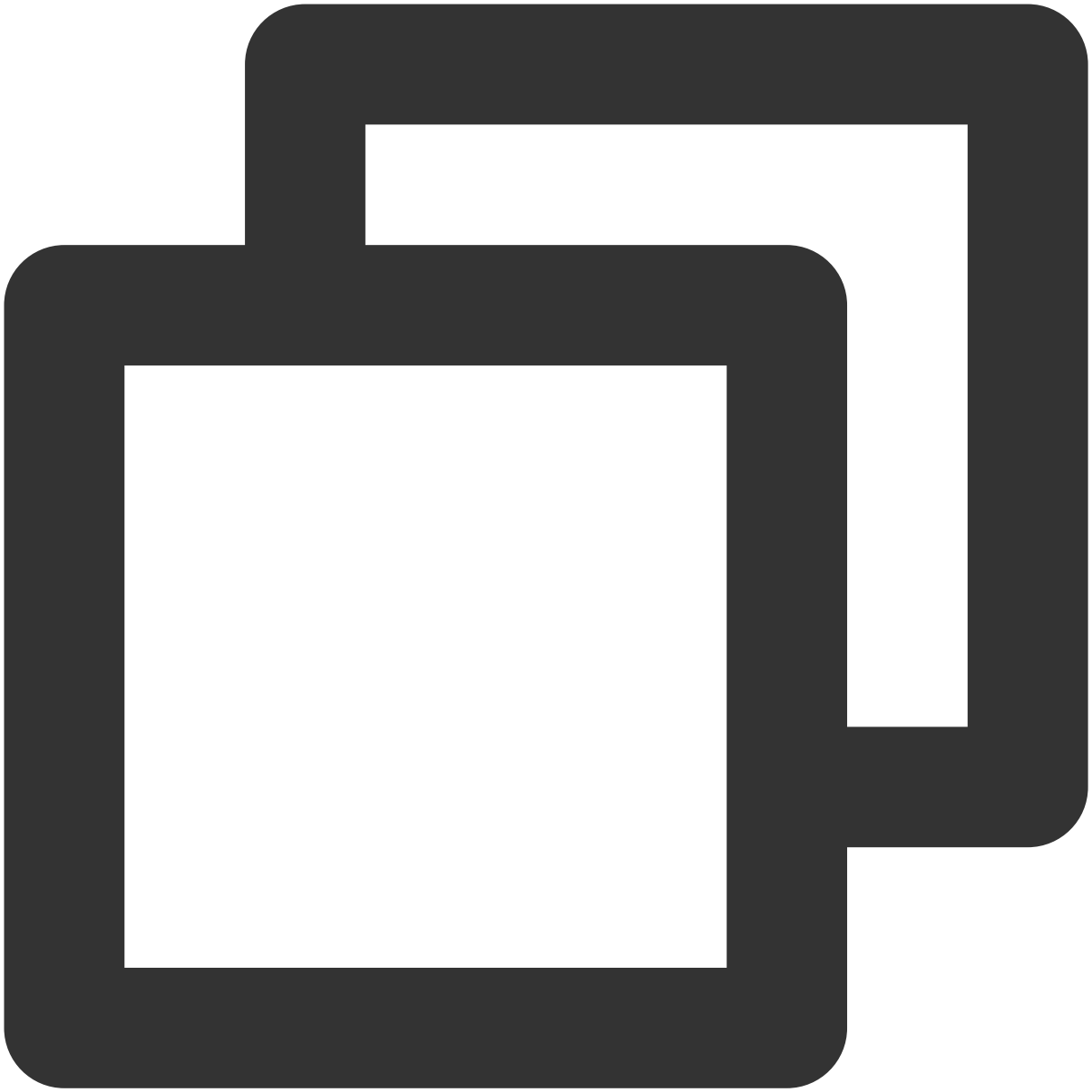
```
void openCamera(TUICommonDefine.Camera camera, TUIVideoView videoView, TUICommonDef
```

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

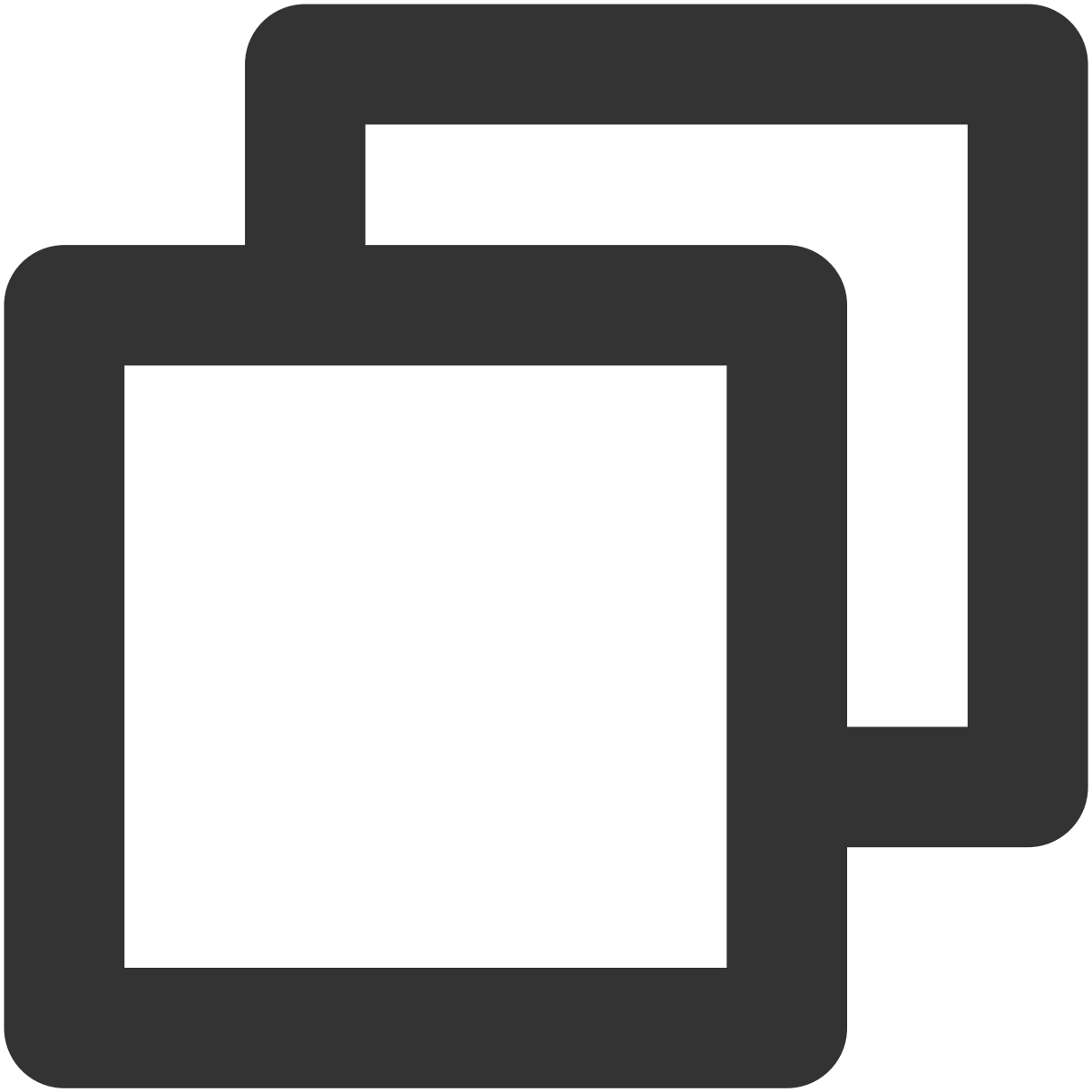
关闭摄像头。



```
void closeCamera();
```

switchCamera

切换前后摄像头。



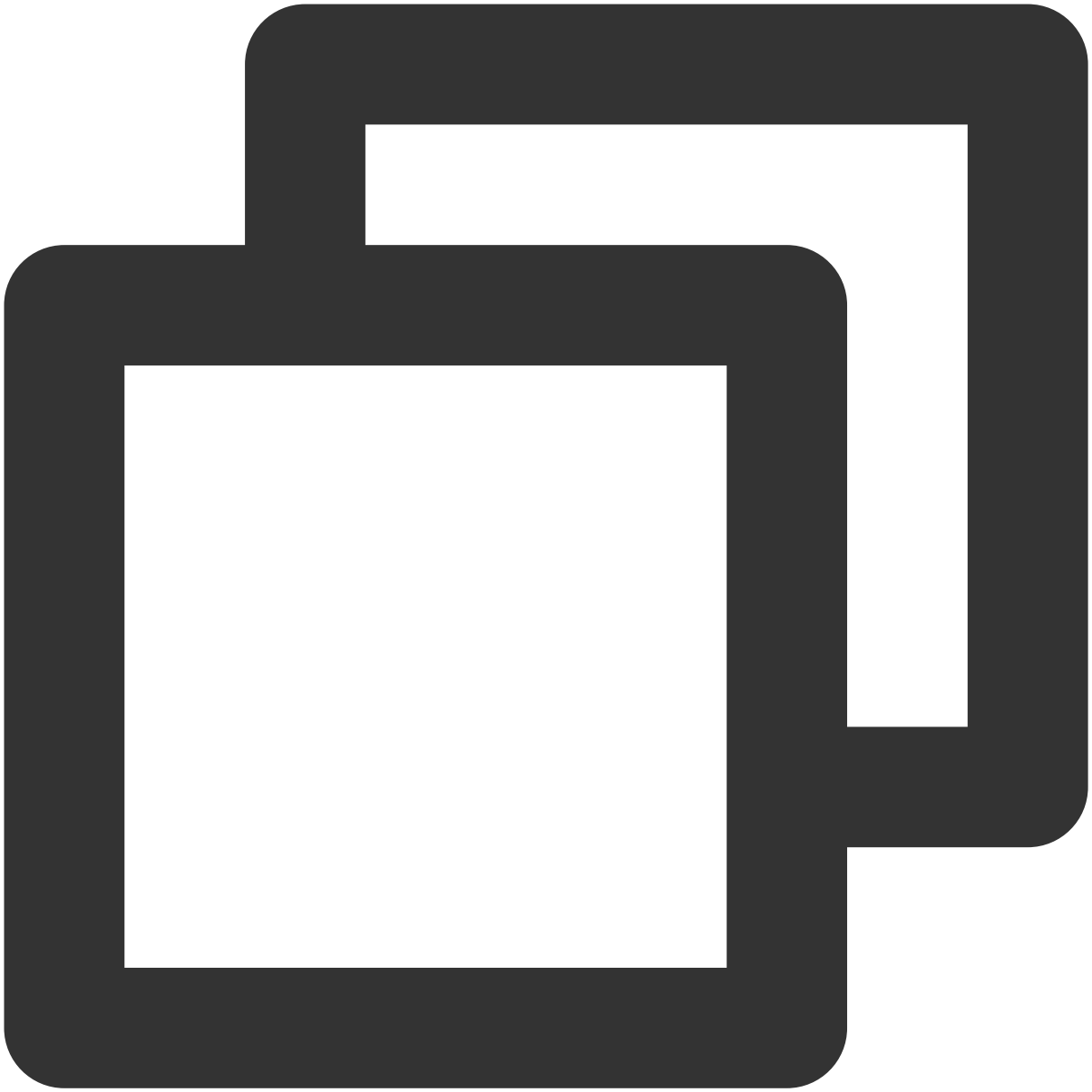
```
void switchCamera(TUICommonDefine.Camera camera);
```

参数如下表所示：

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

openMicrophone

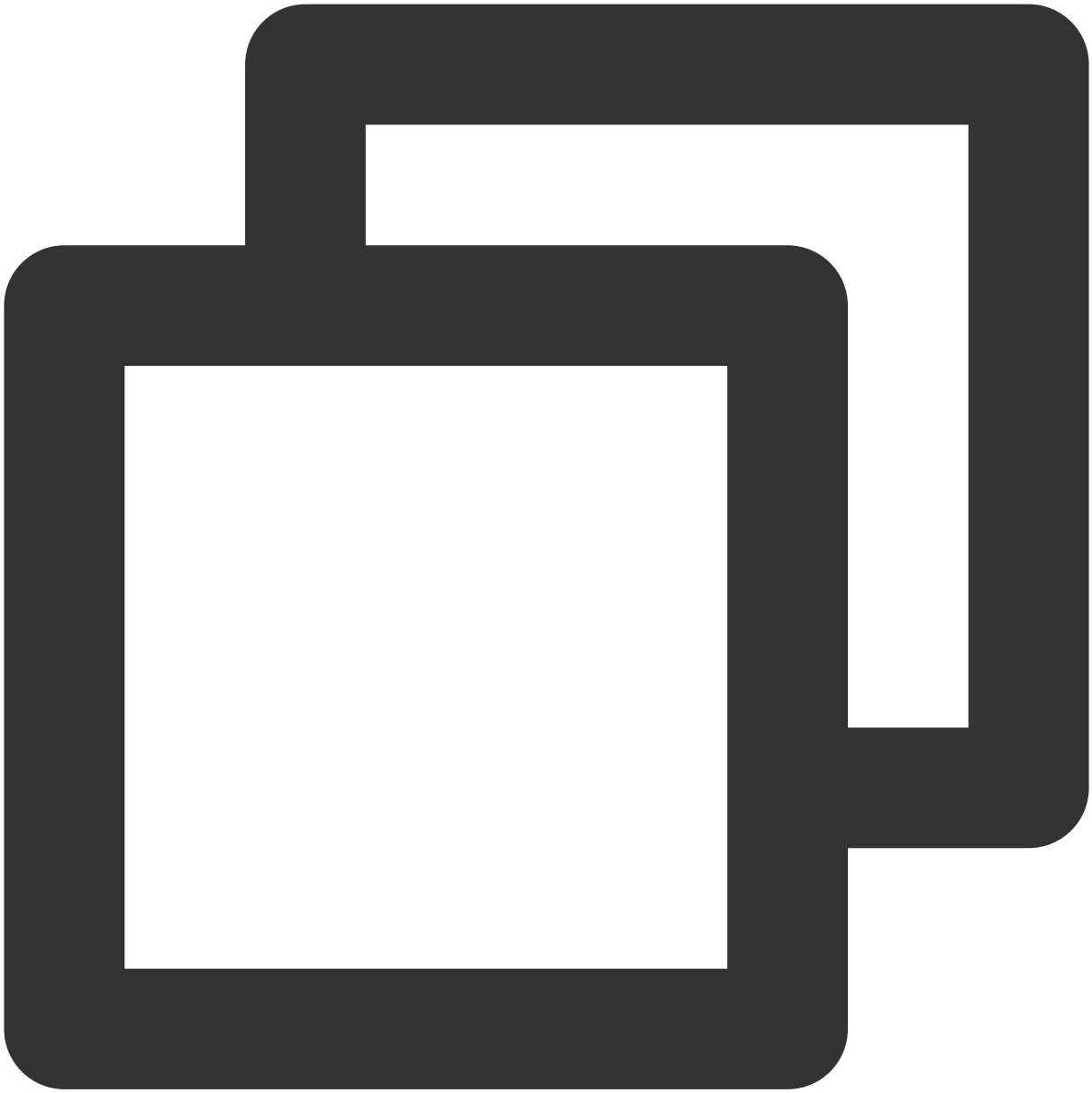
打开麦克风。



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

closeMicrophone

关闭麦克风。

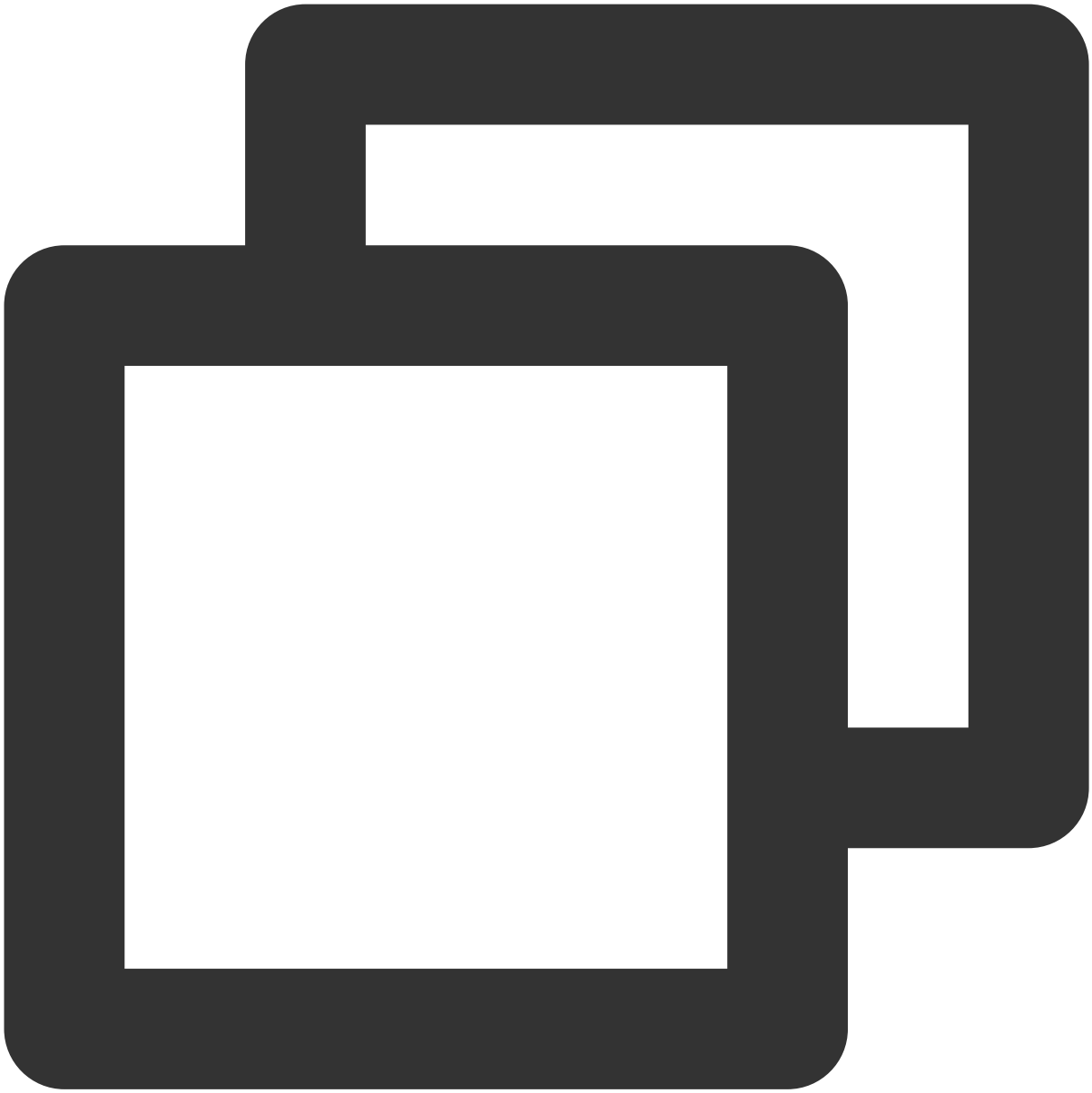


```
void closeMicrophone();
```

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);
```

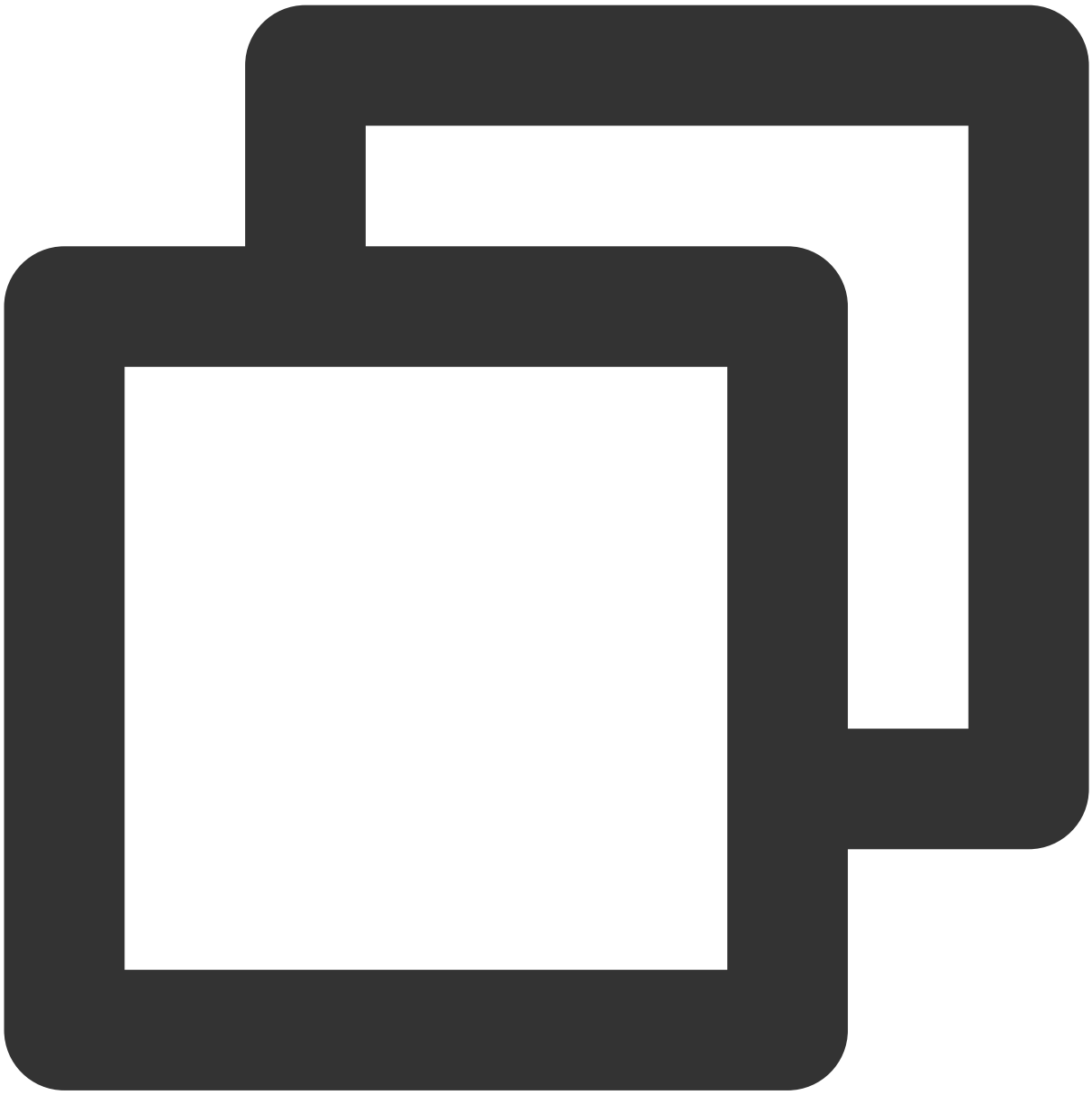
参数如下表所示：

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。



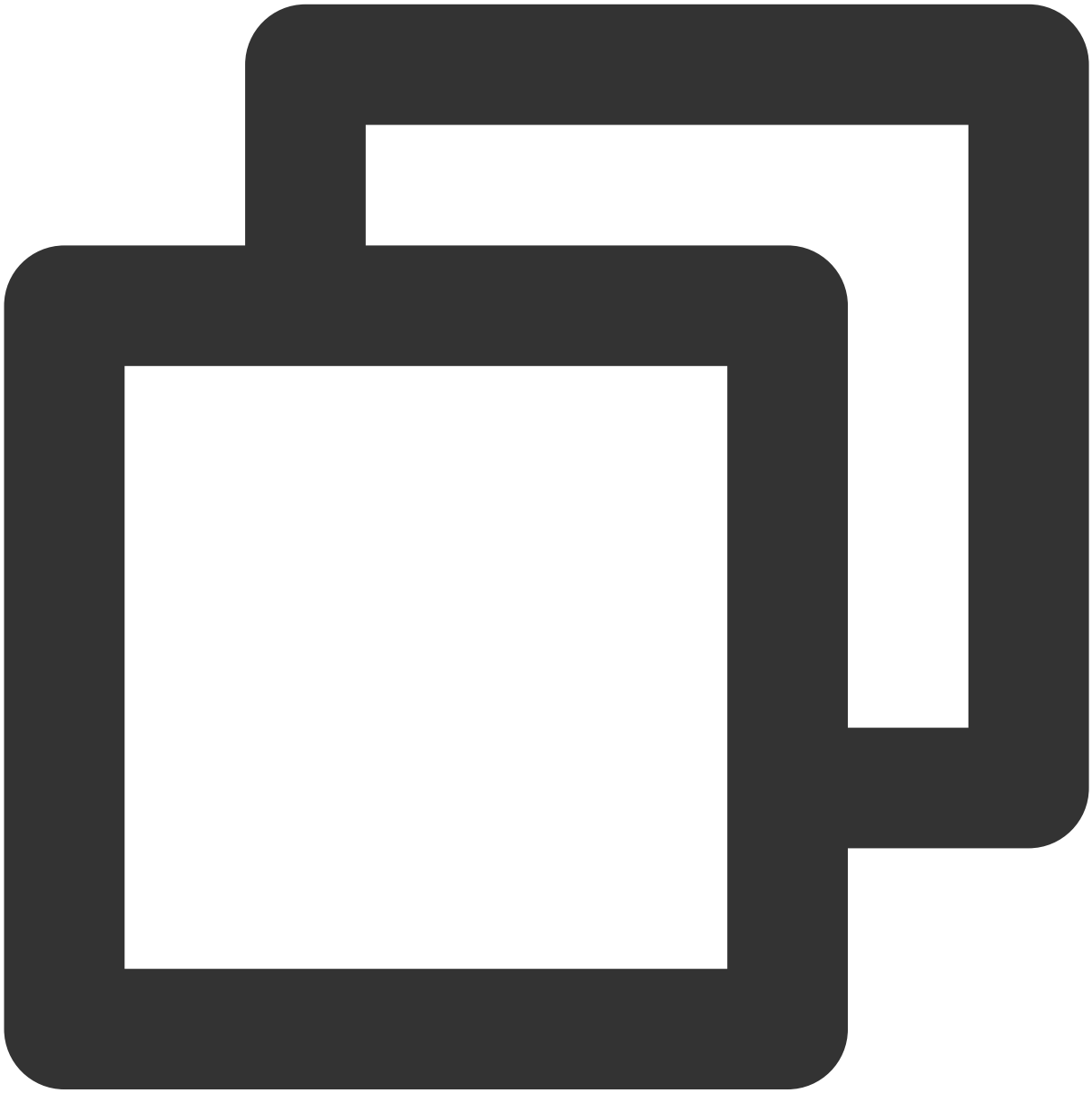
```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

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

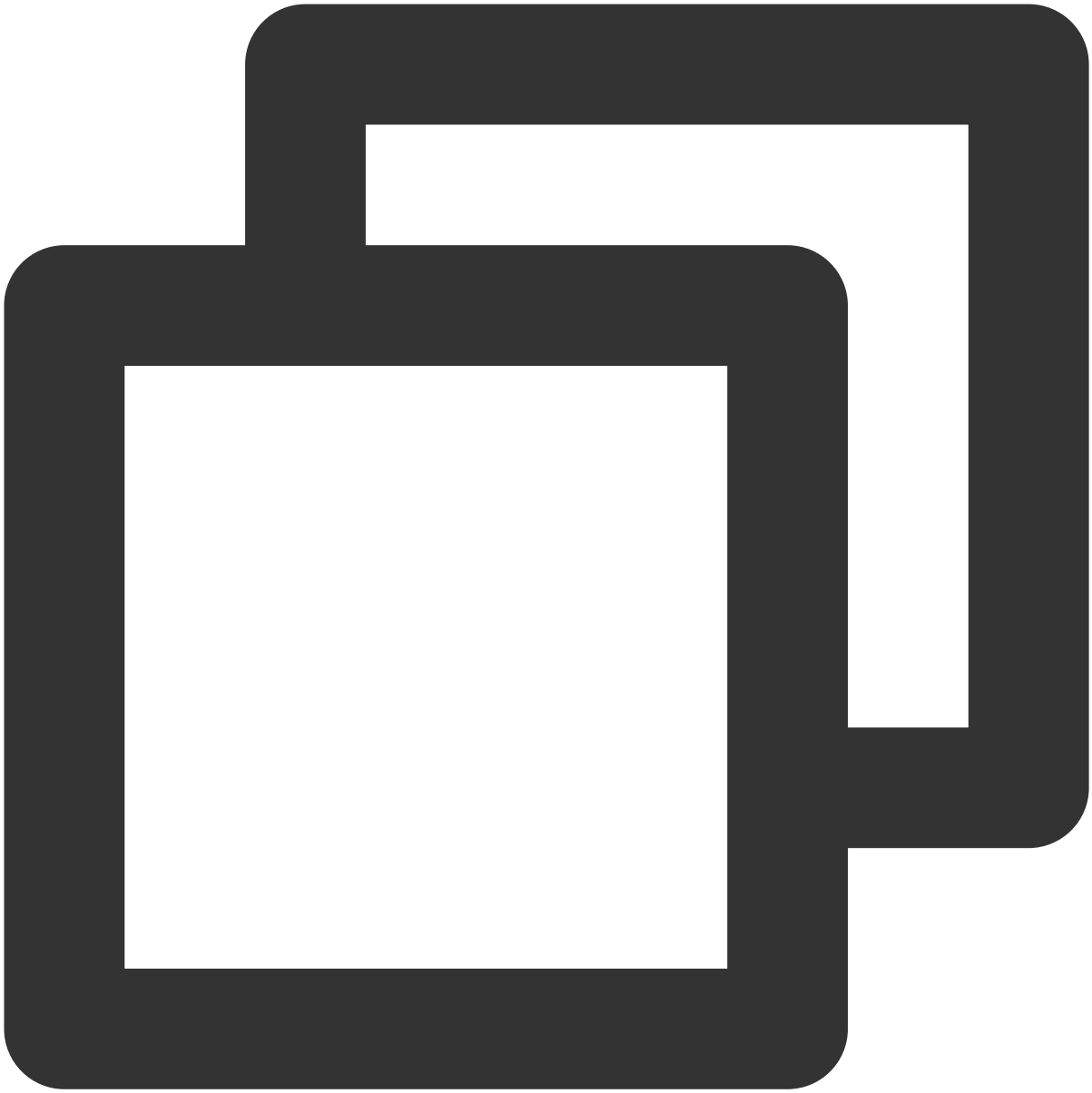
开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐支持）。



```
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,
```

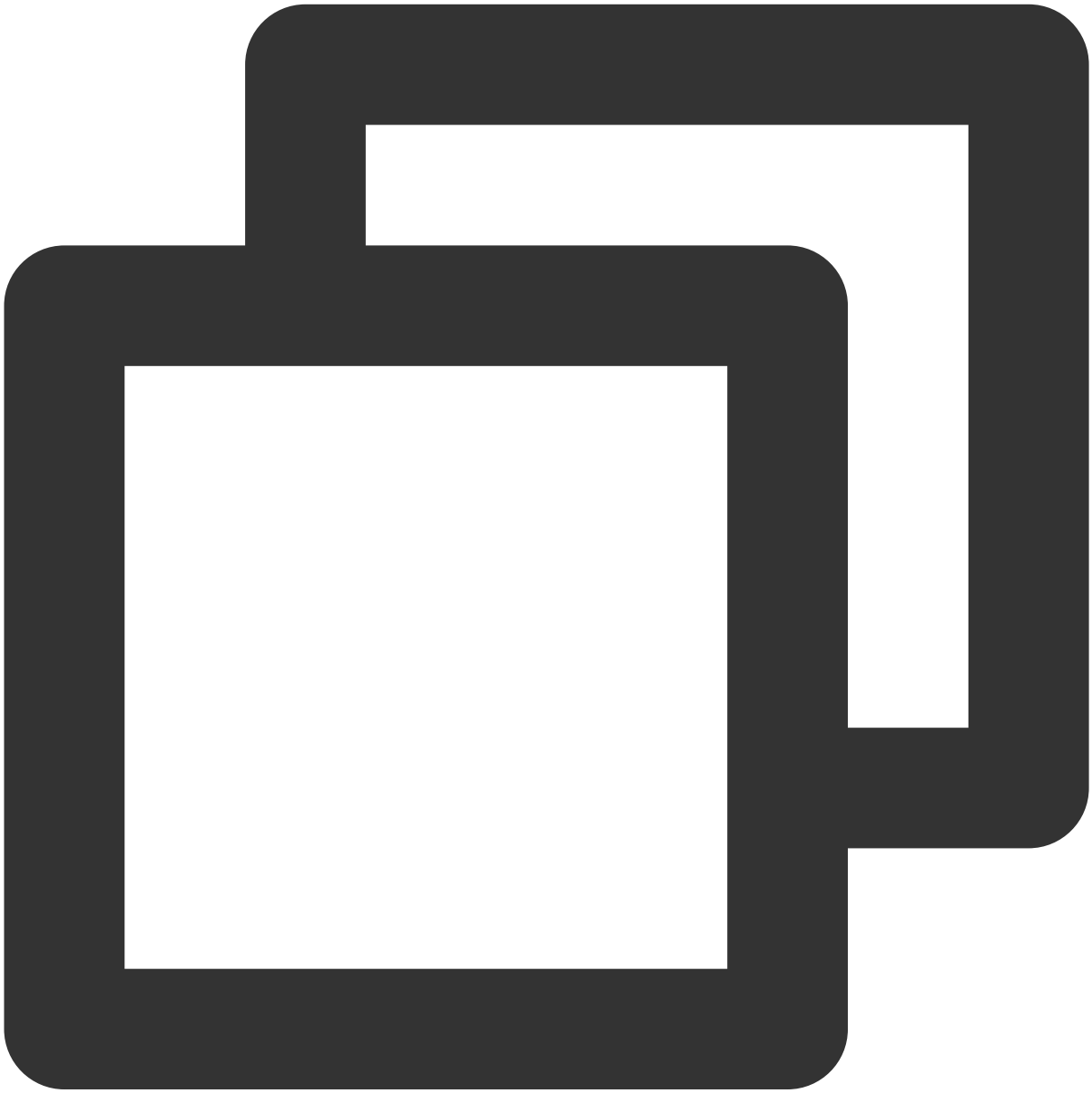
参数如下表所示：

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：

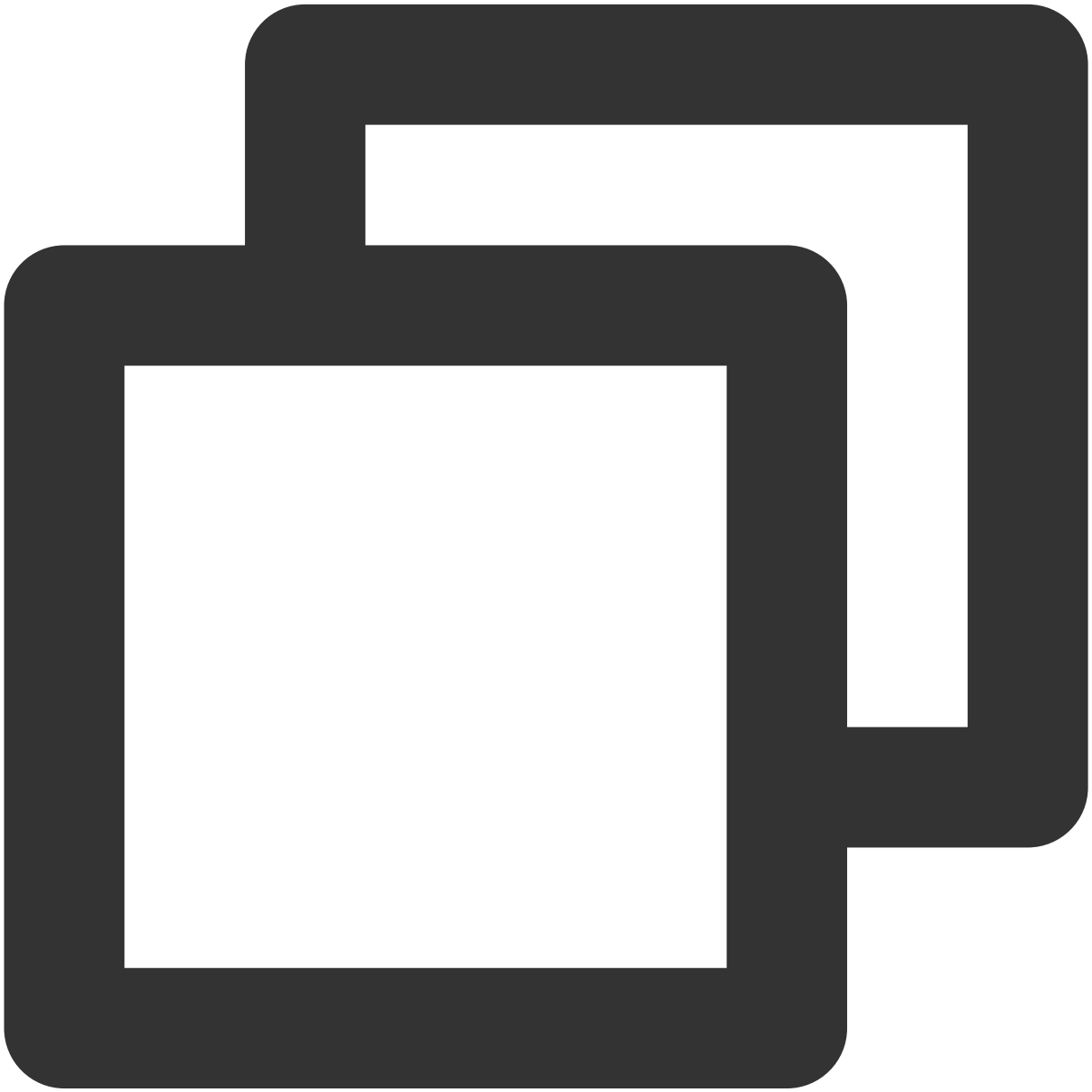
参数	类型	含义
params	TUICommonDefine.VideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高

		比

getTRTCCloudInstance

高级接口。

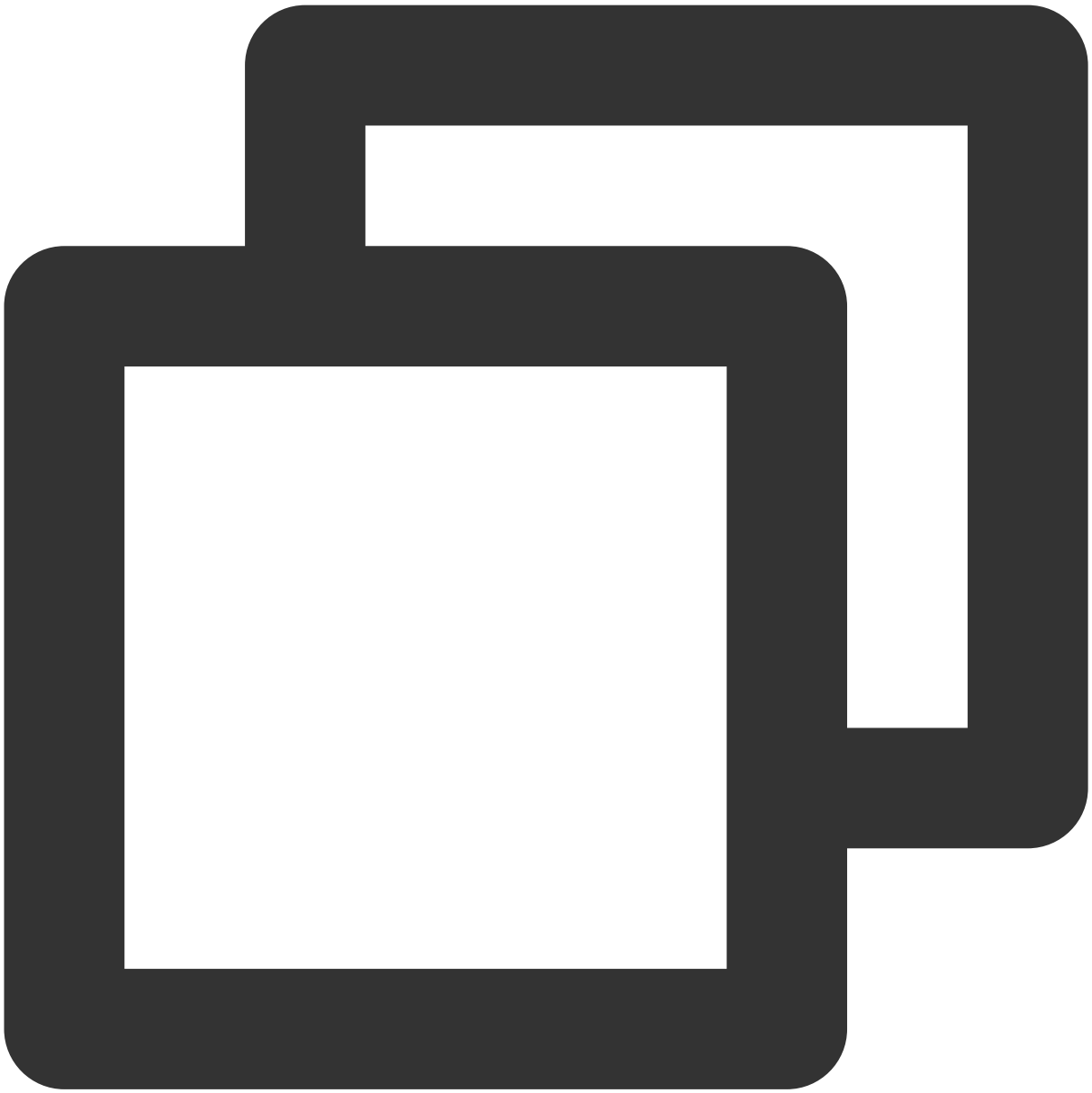
用于接入第三方美颜等。



```
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

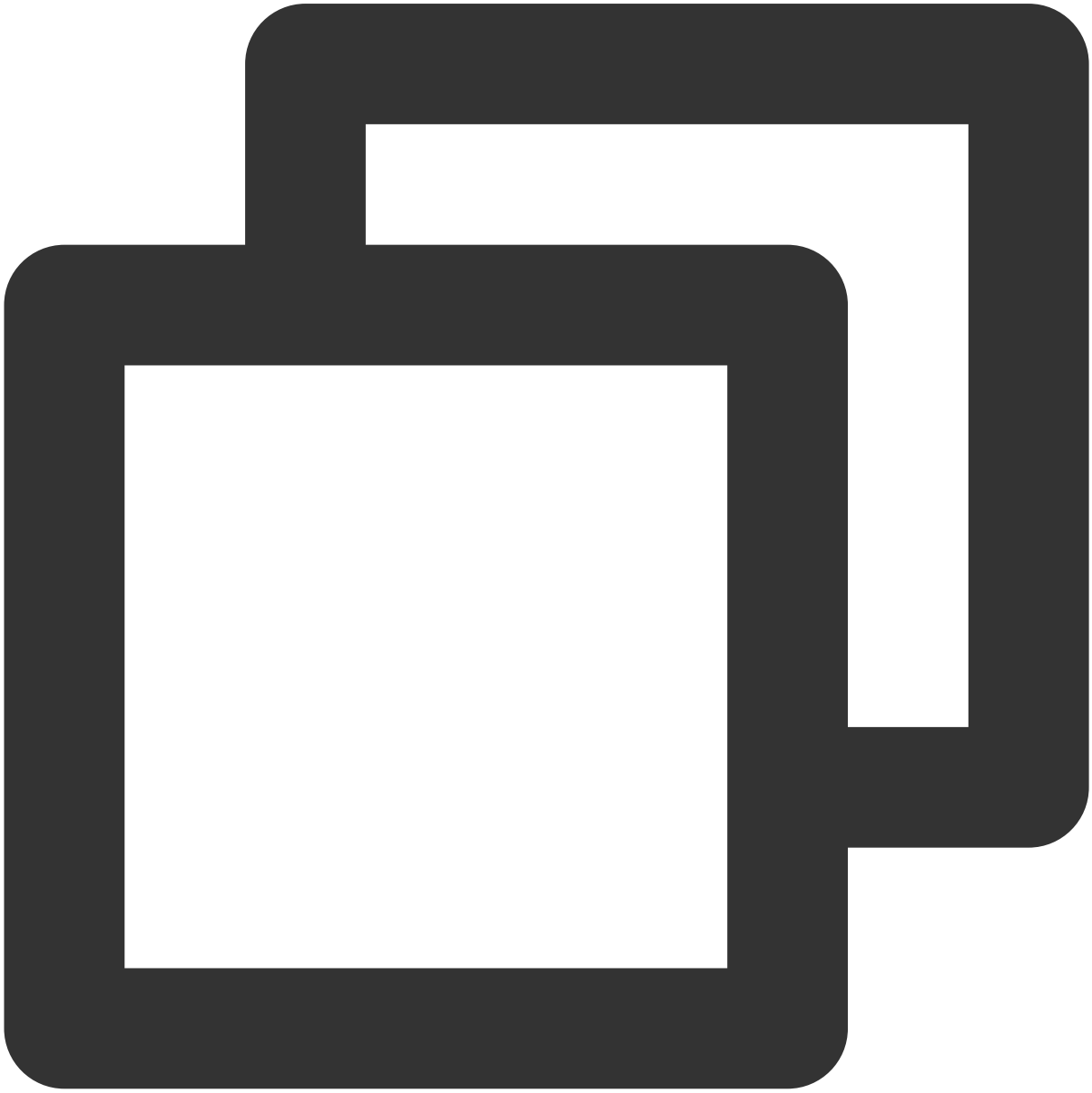
回调事件详情

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。



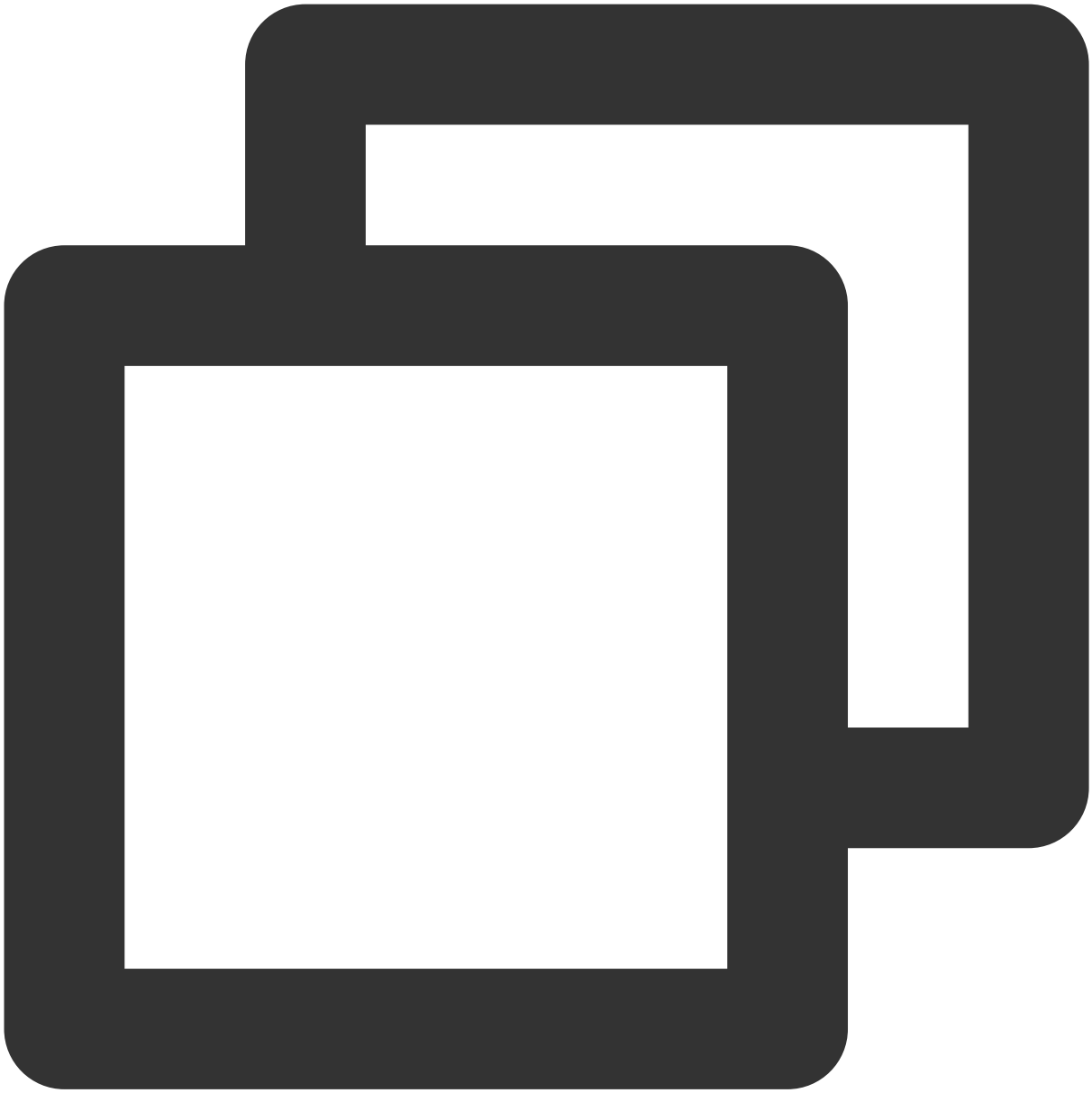
```
void onError(int code, String msg);
```

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。

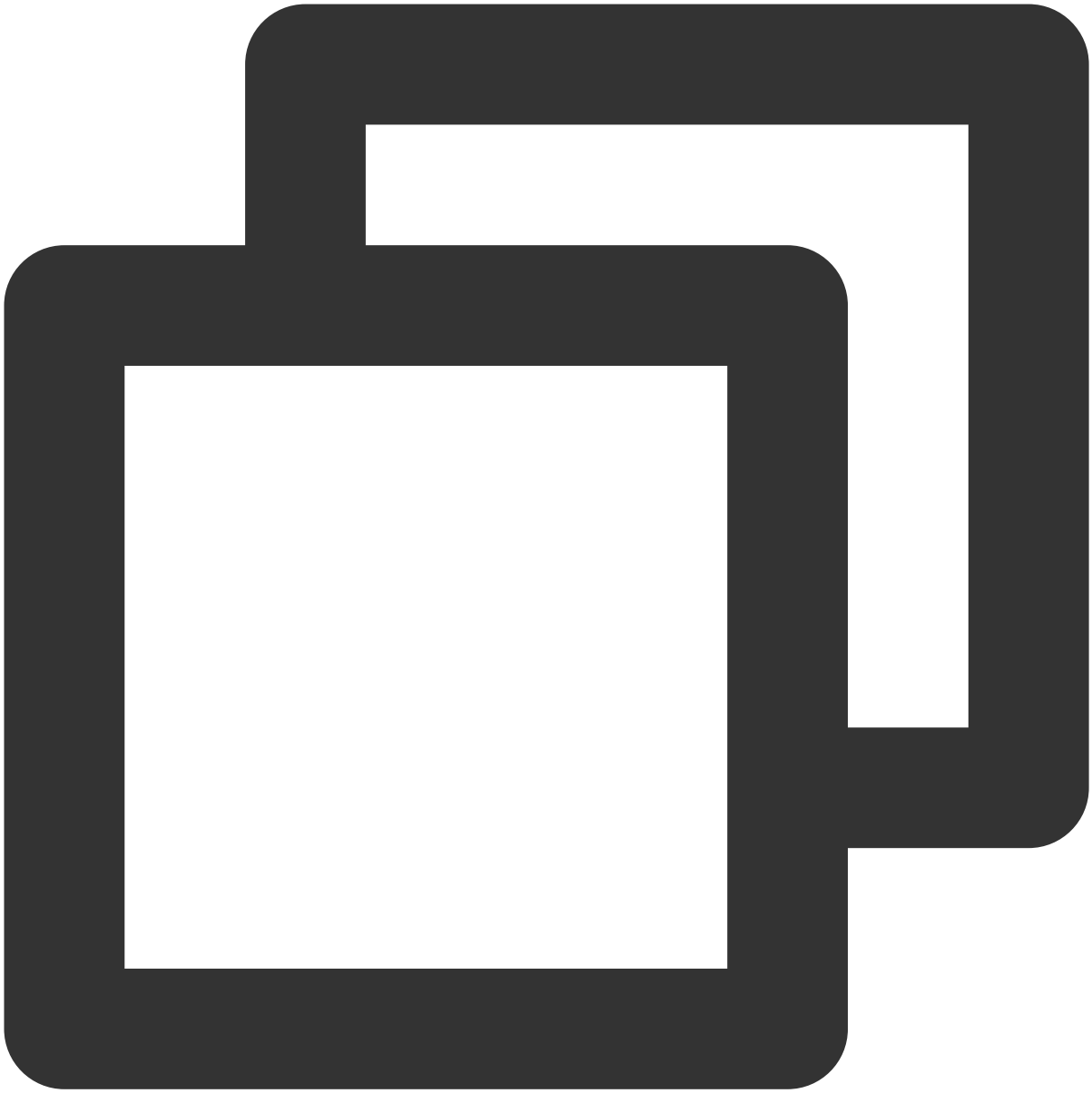
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

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）



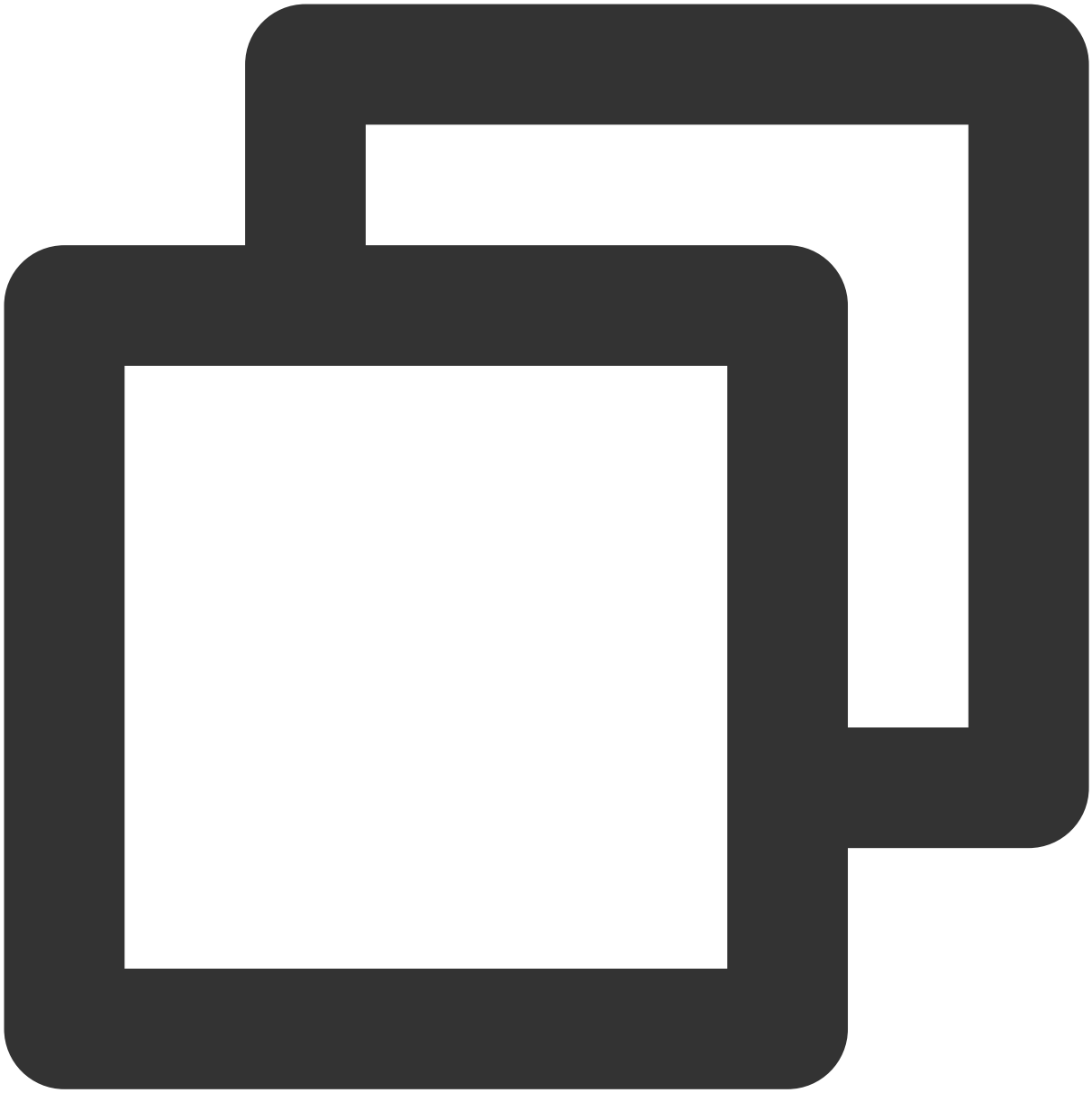
```
void onCallCancelled(String userId);
```

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。



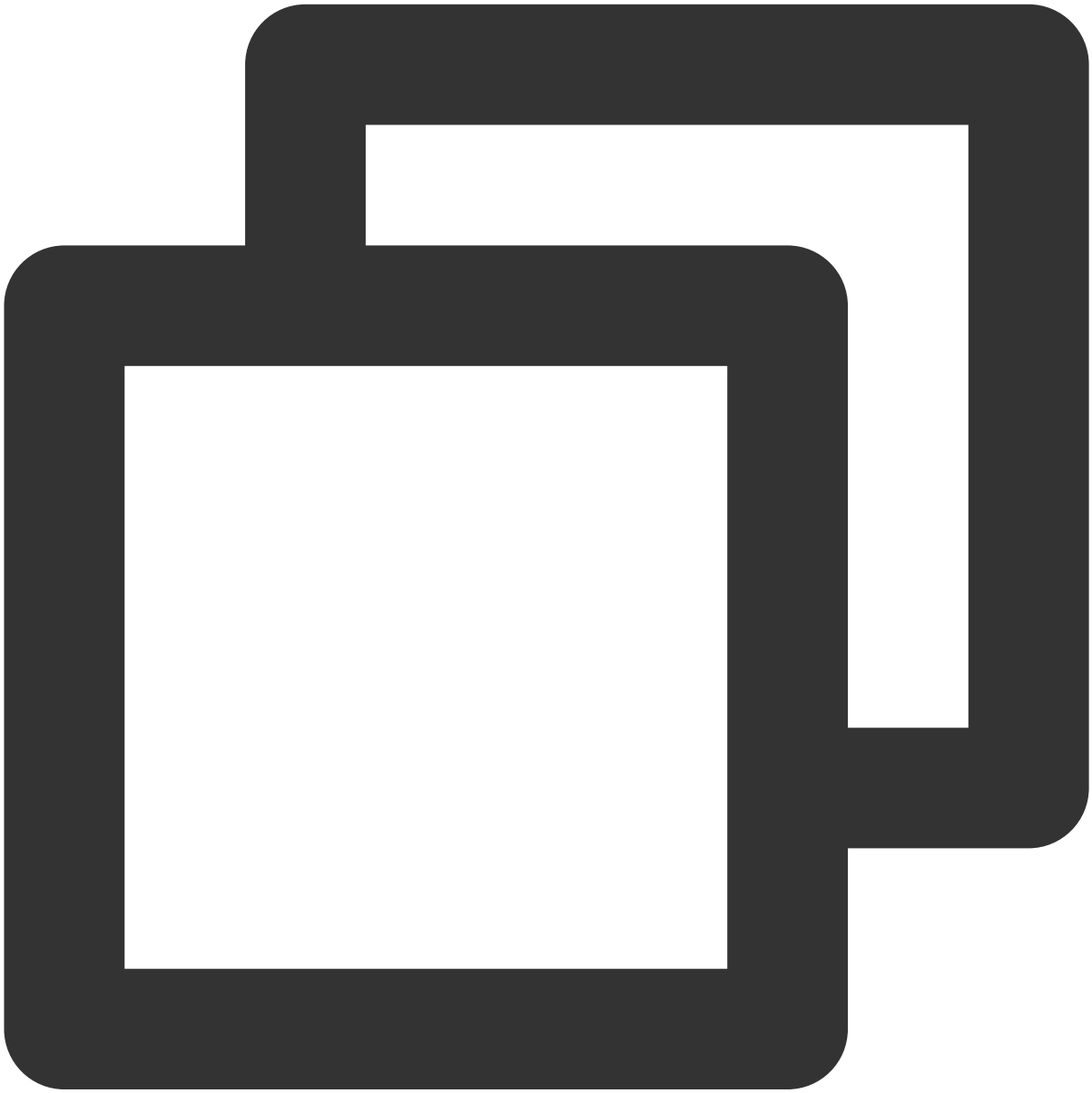
```
void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaTy
```

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

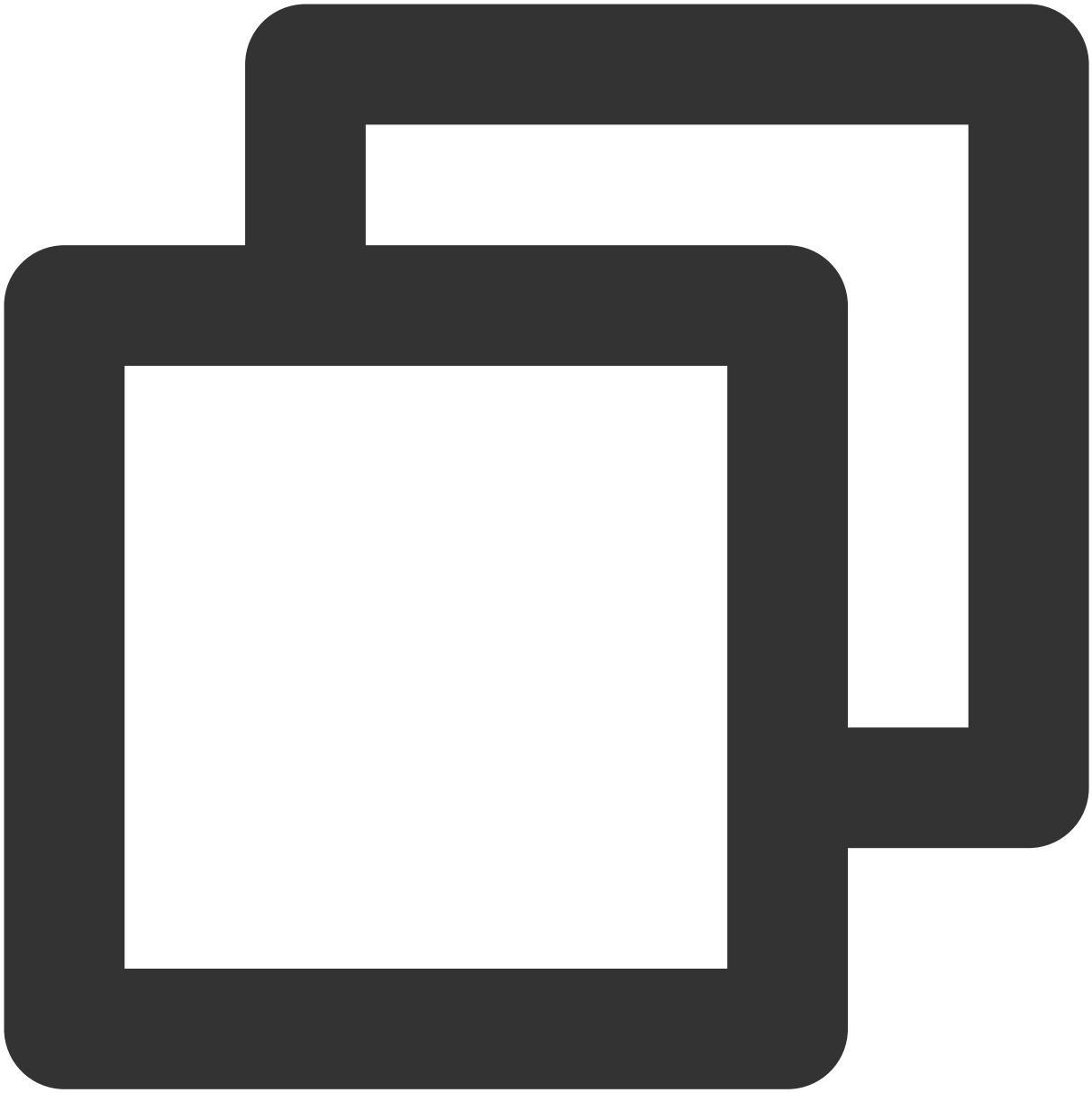
callMediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallDefine.Role	角色，枚举类型：主叫、被叫
totalTime	long	此次通话的时长，单位：秒

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。



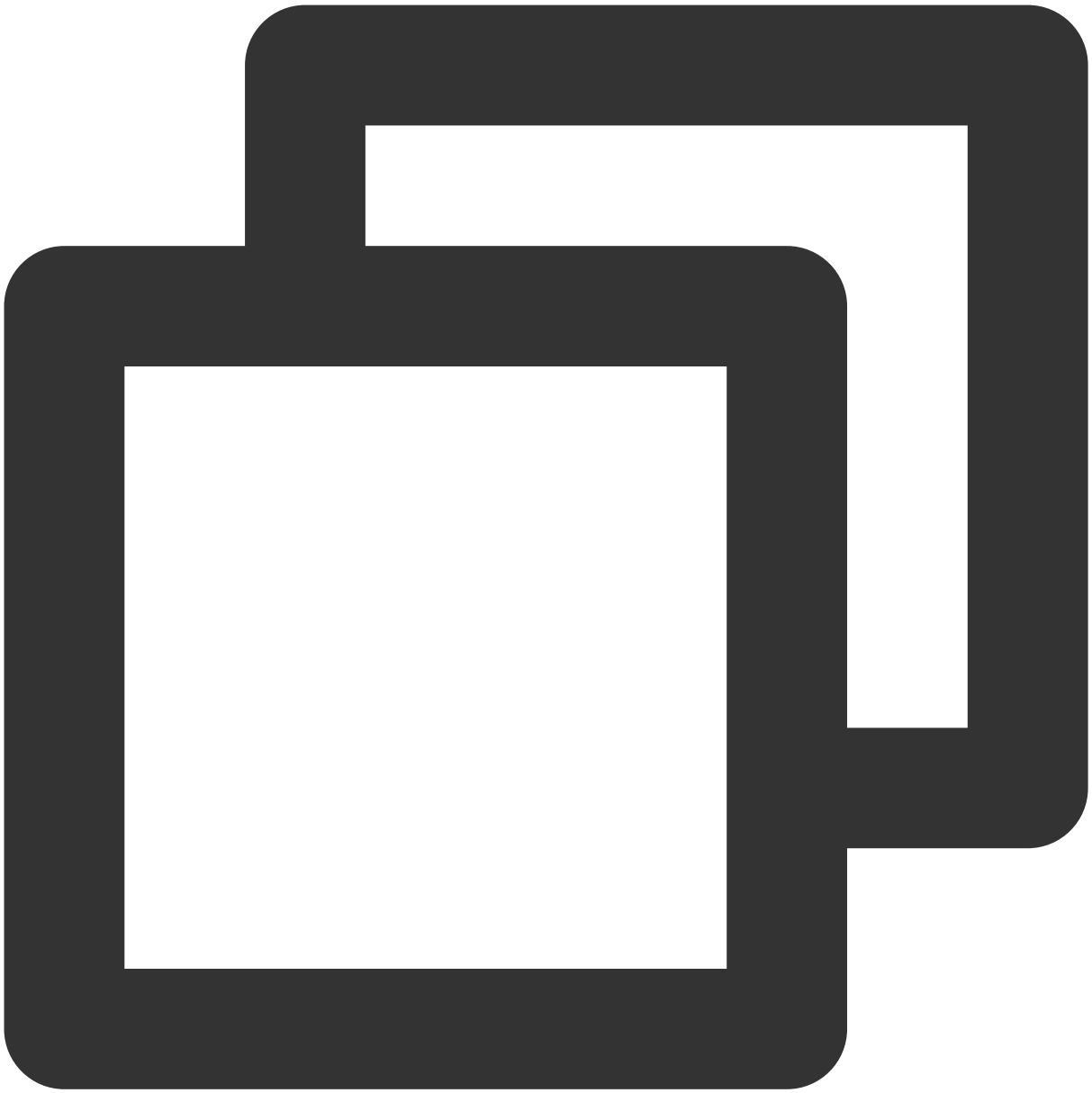
```
void onCallMediaTypeChanged(TUICallDefine.MediaType oldCallMediaType,TUICallDefine.
```

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。



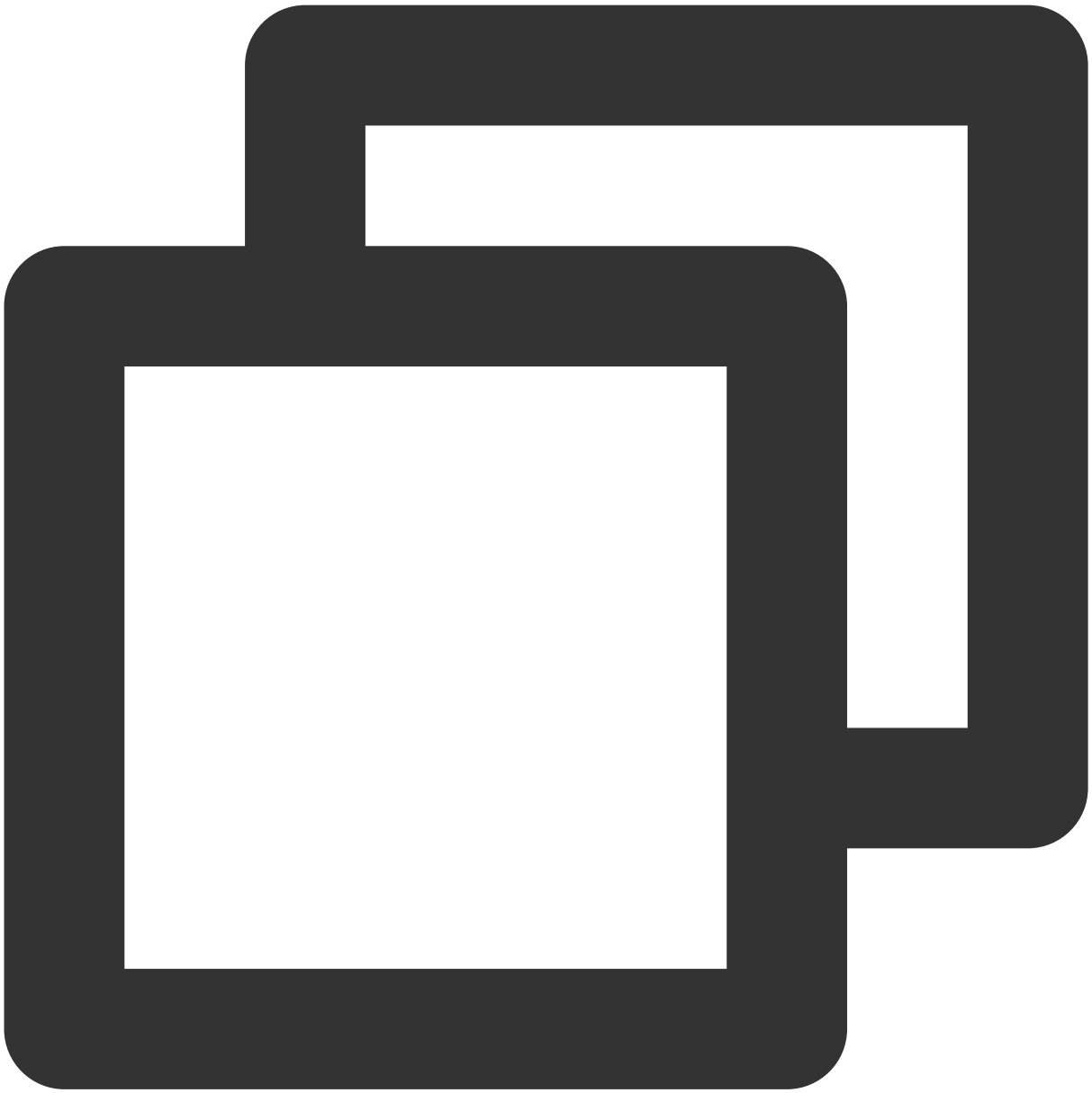
```
void onUserReject(String userId);
```

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。



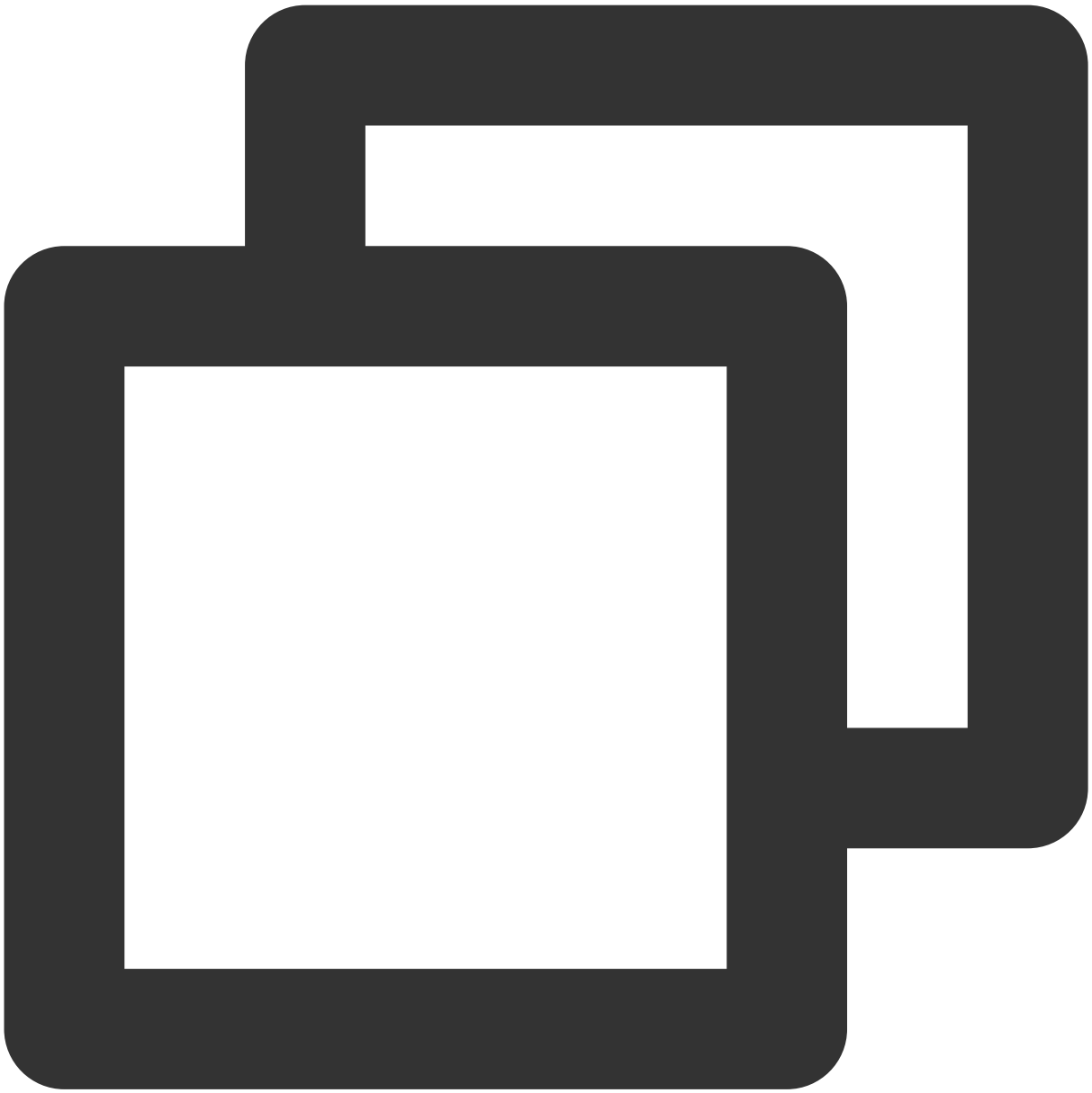
```
void onUserNoResponse(String userId);
```

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。



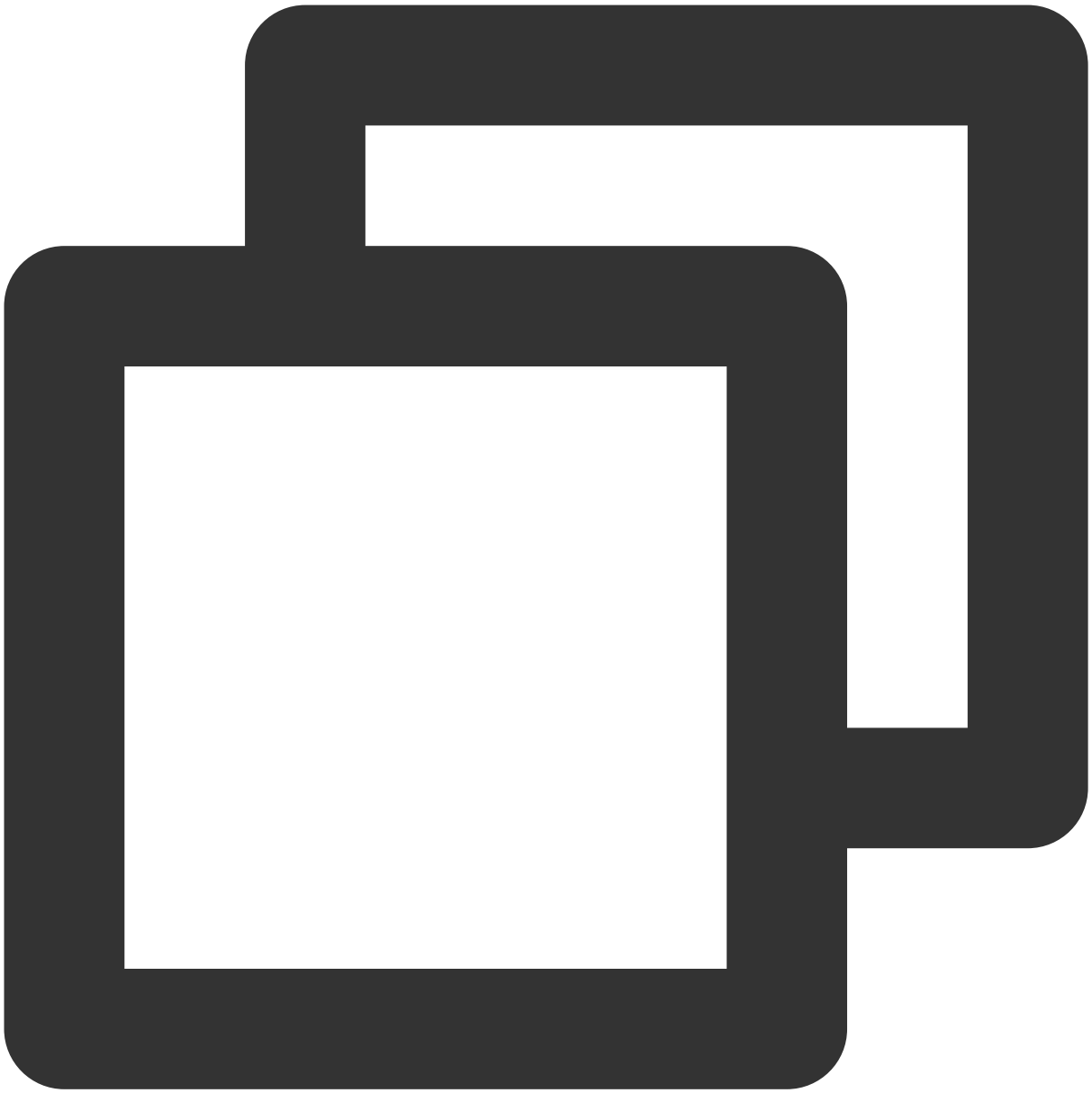
```
void onUserLineBusy(String userId);
```

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。



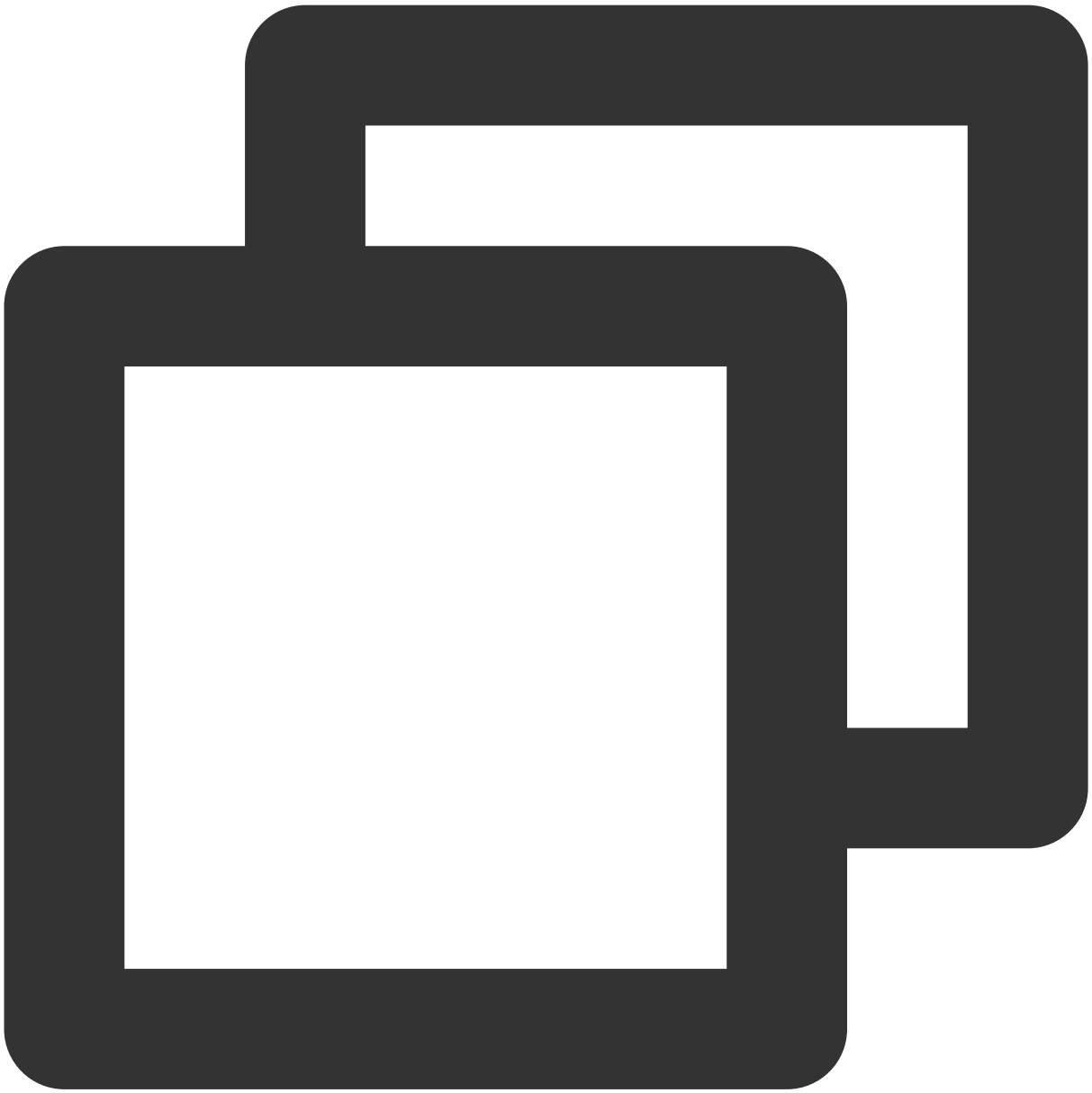
```
void onUserJoin(String userId);
```

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。



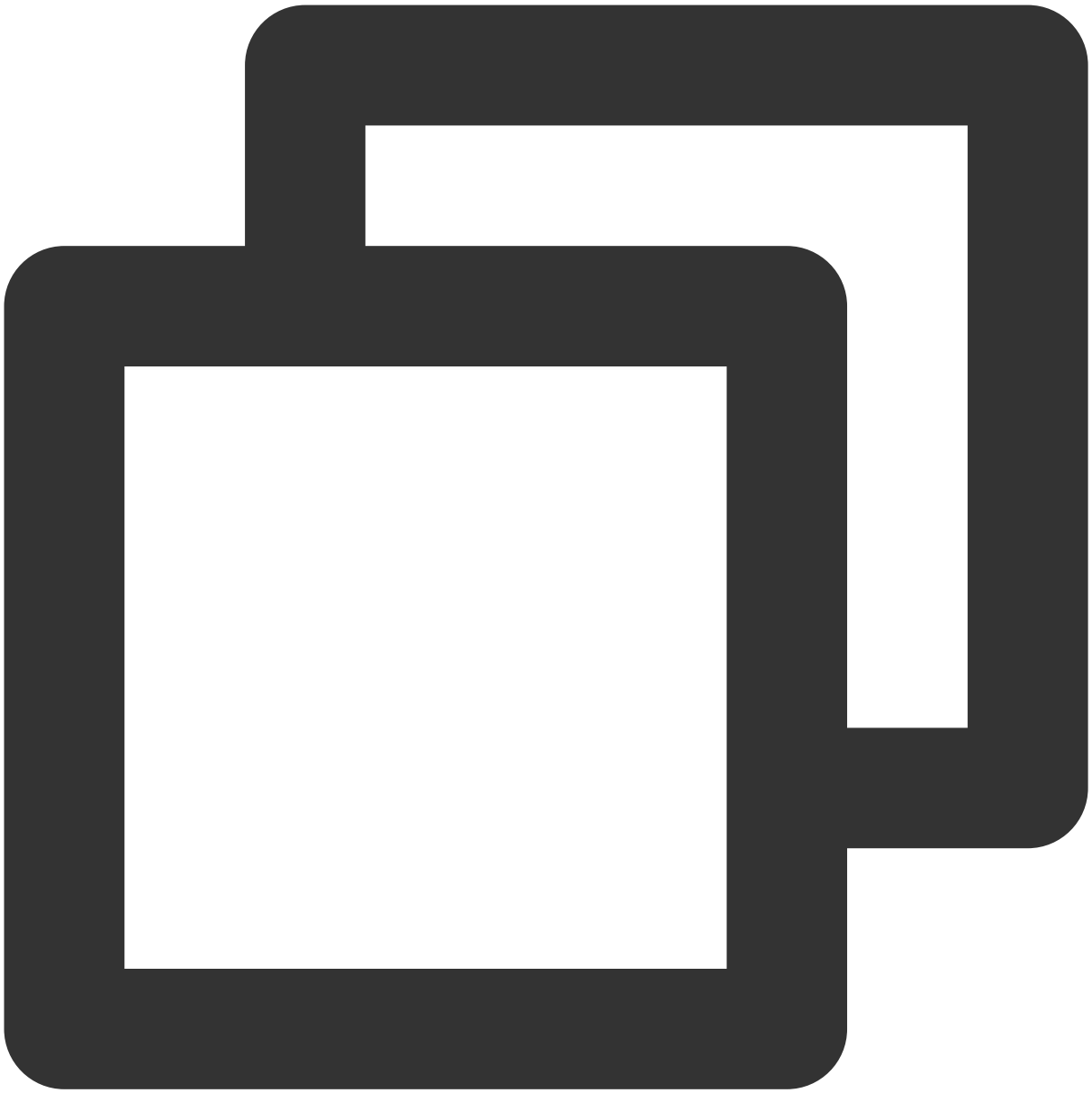
```
void onUserLeave(String userId);
```

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。



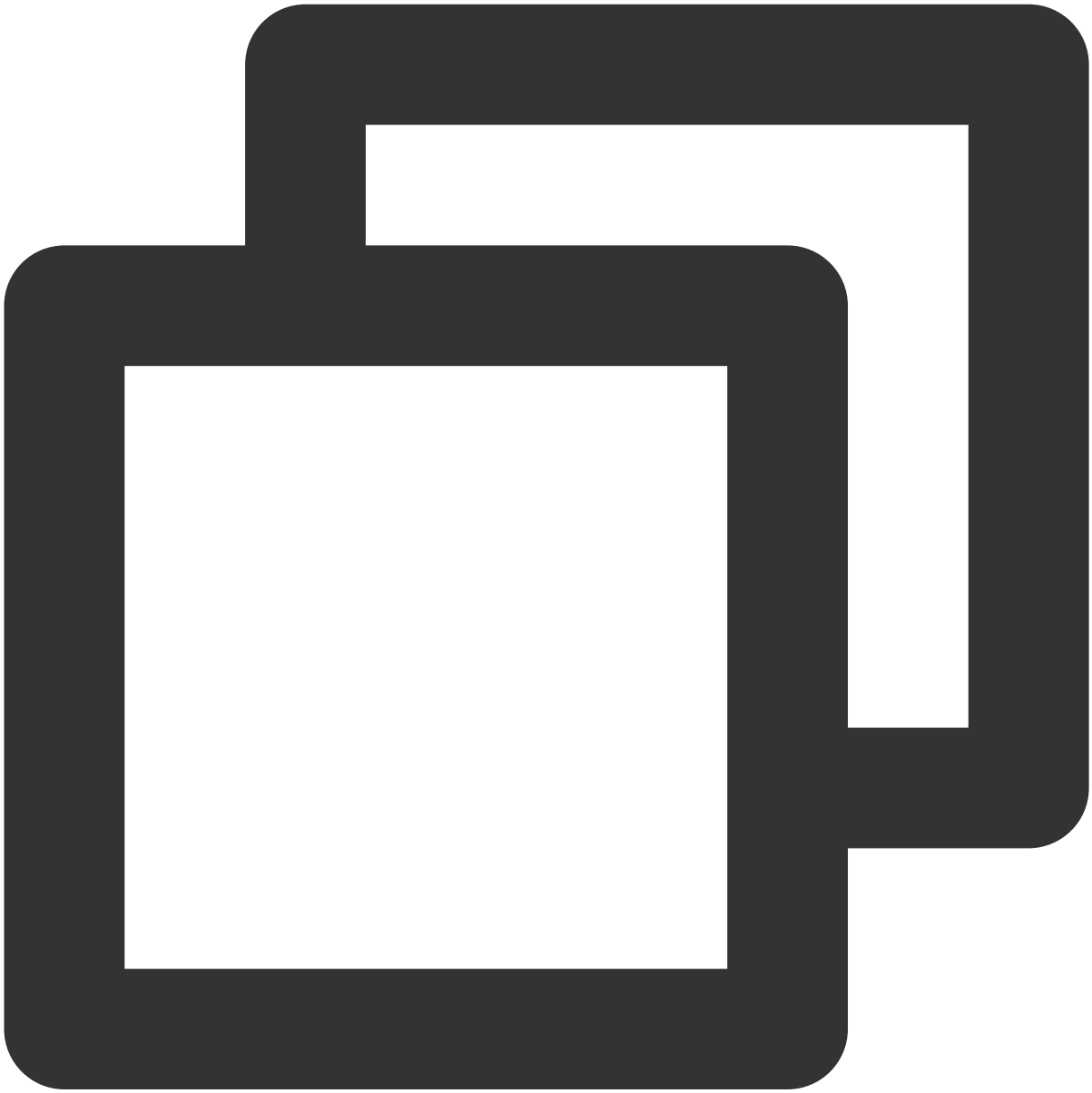
```
void onUserVideoAvailable(String userId, boolean isVideoAvailable);
```

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。



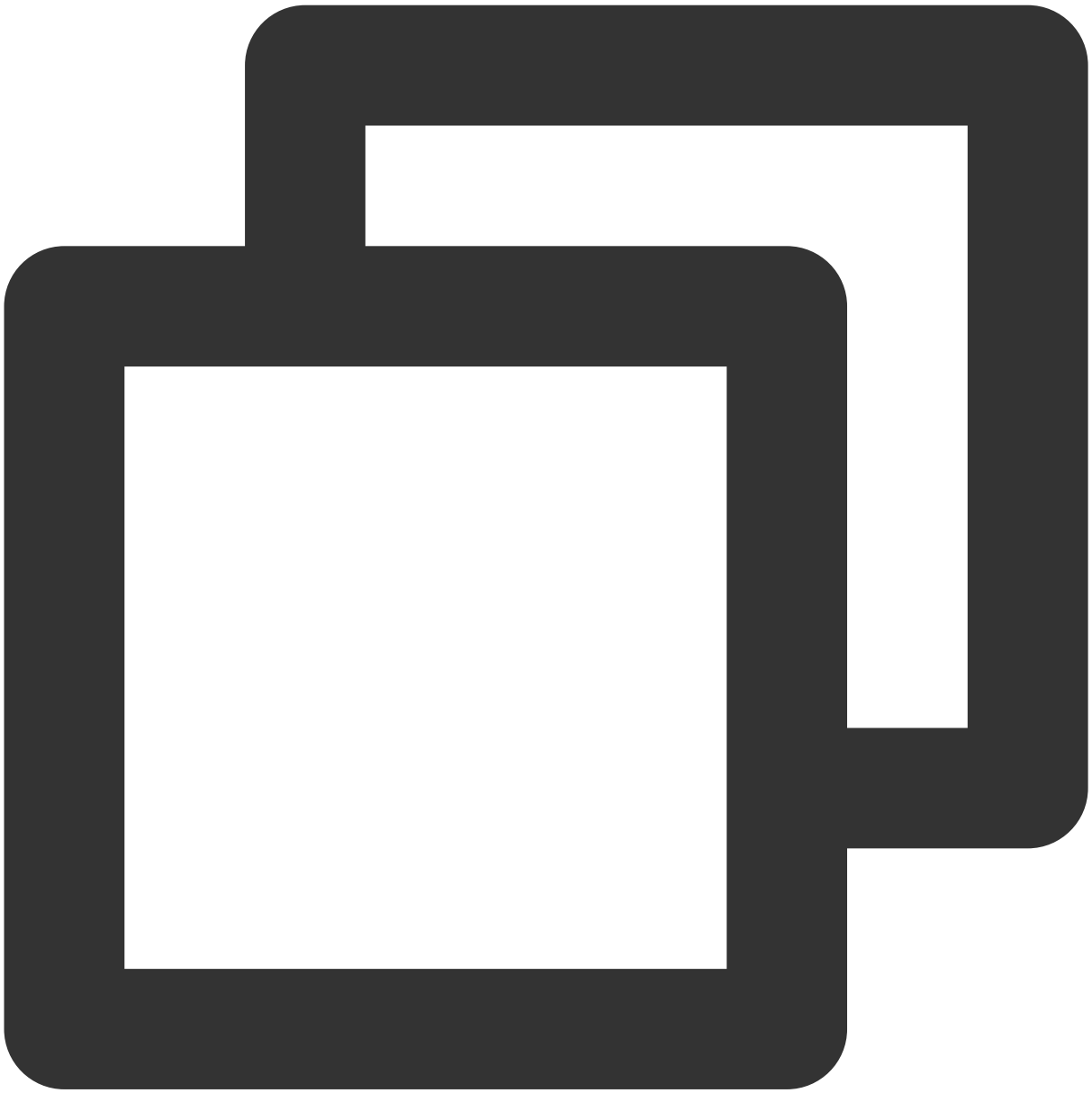
```
void onUserAudioAvailable(String userId, boolean isAudioAvailable);
```

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);
```

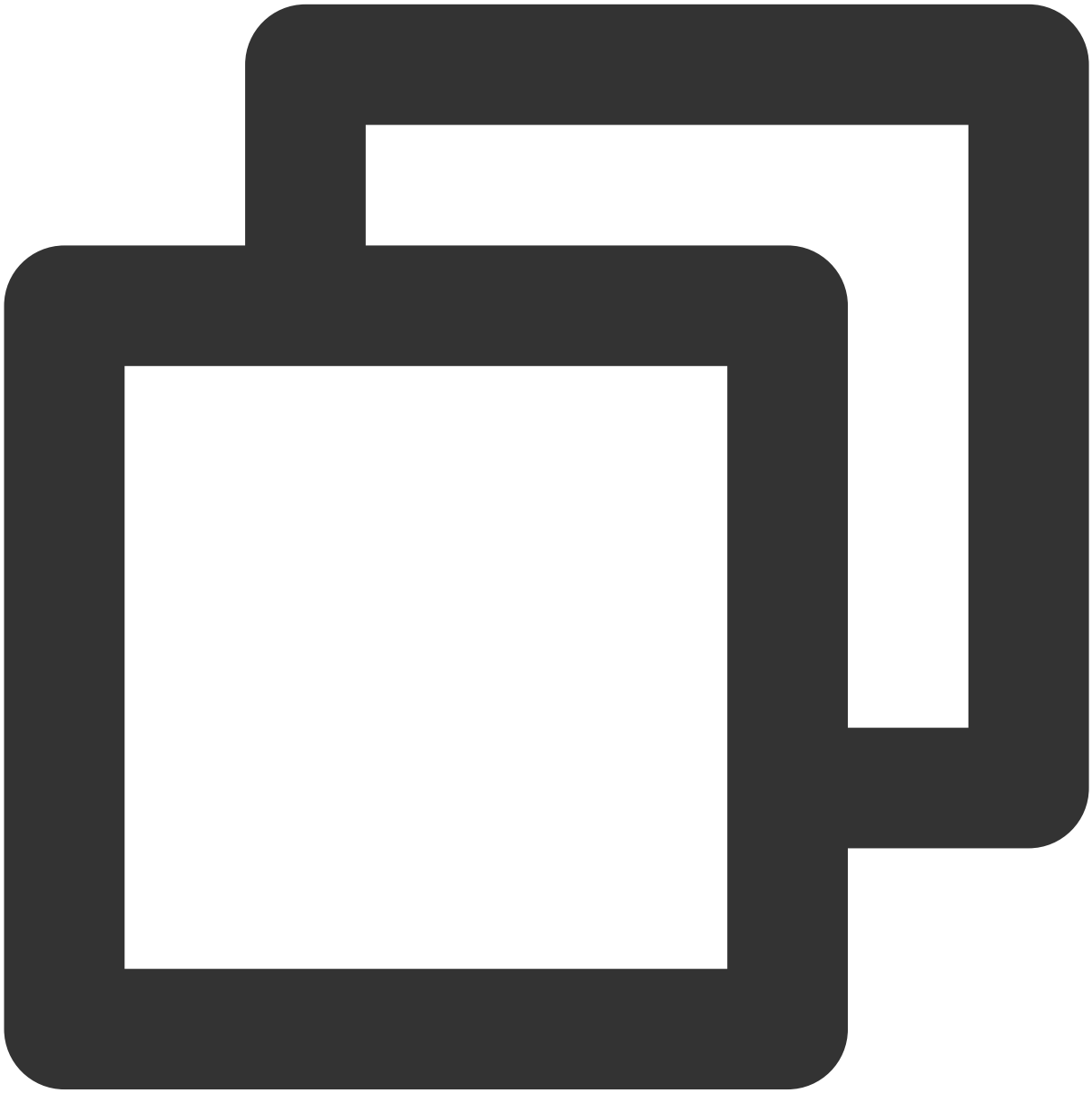
参数如下表所示：

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
```

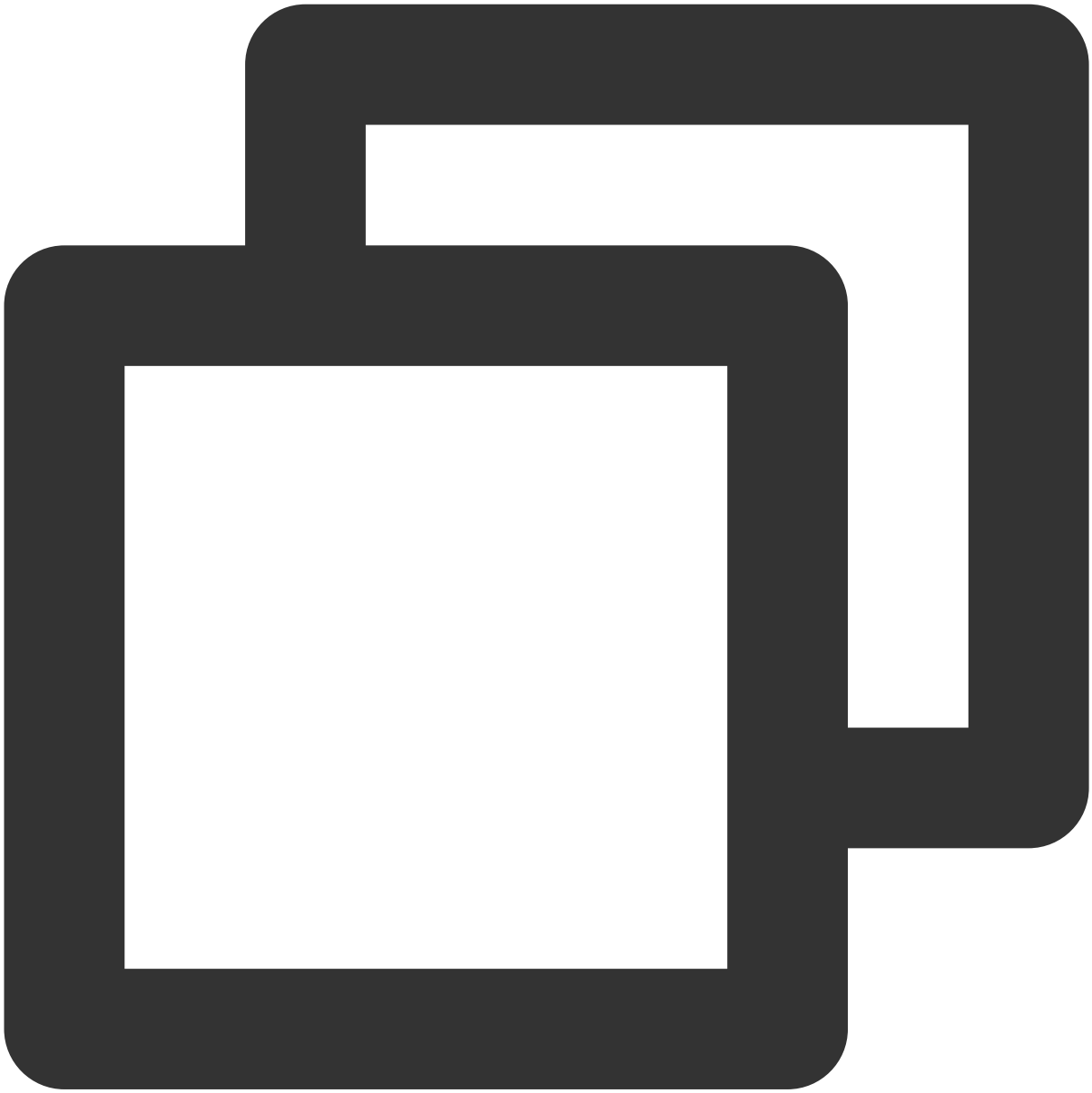
参数如下表所示：

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

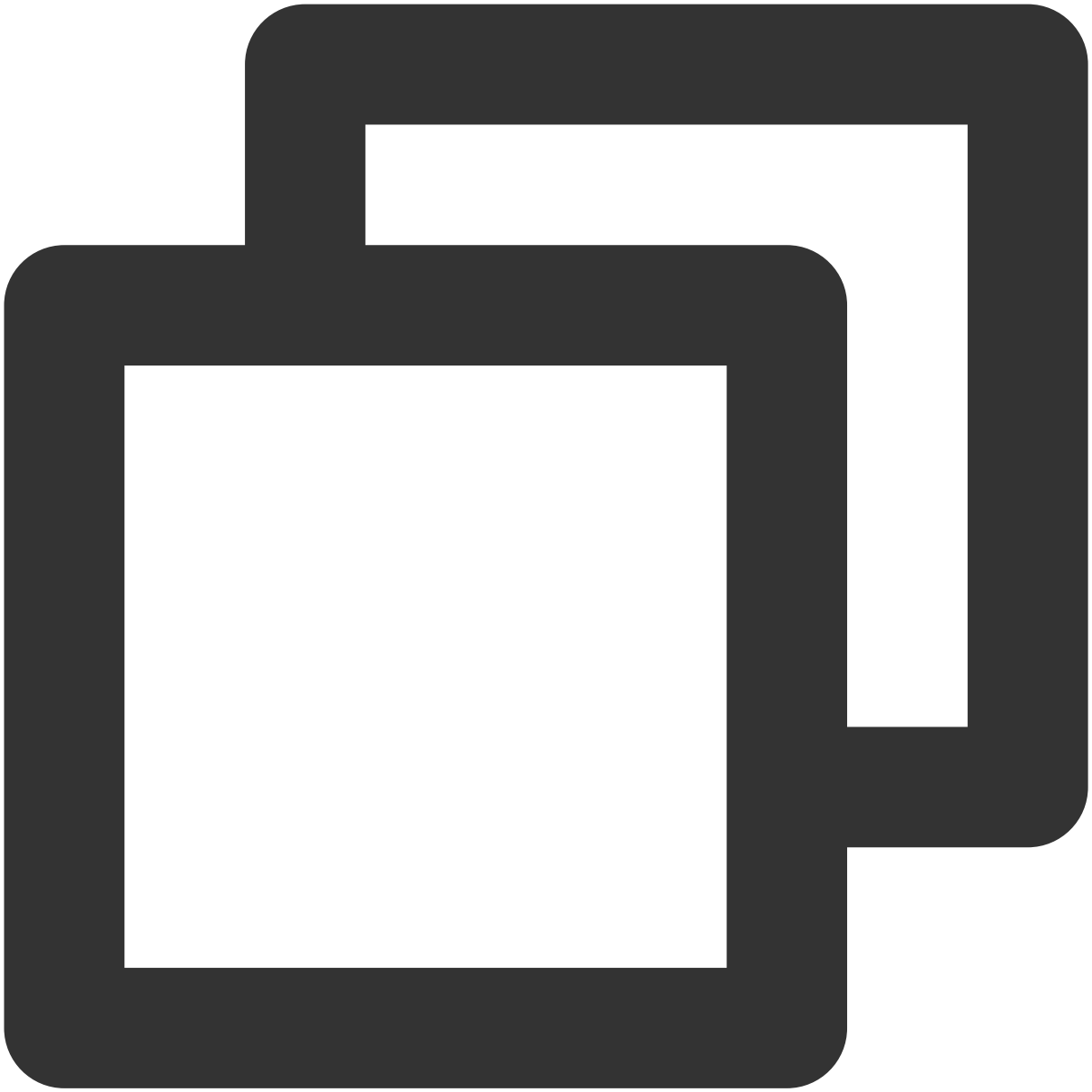
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
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

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：2024-07-25 17:49:30

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
enableIncomingBanner	开启/关闭来电横幅显示

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（群组通话版套餐支持）
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

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2024-07-24 10:20:03

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
enableIncomingBanner	开启/关闭来电横幅显示

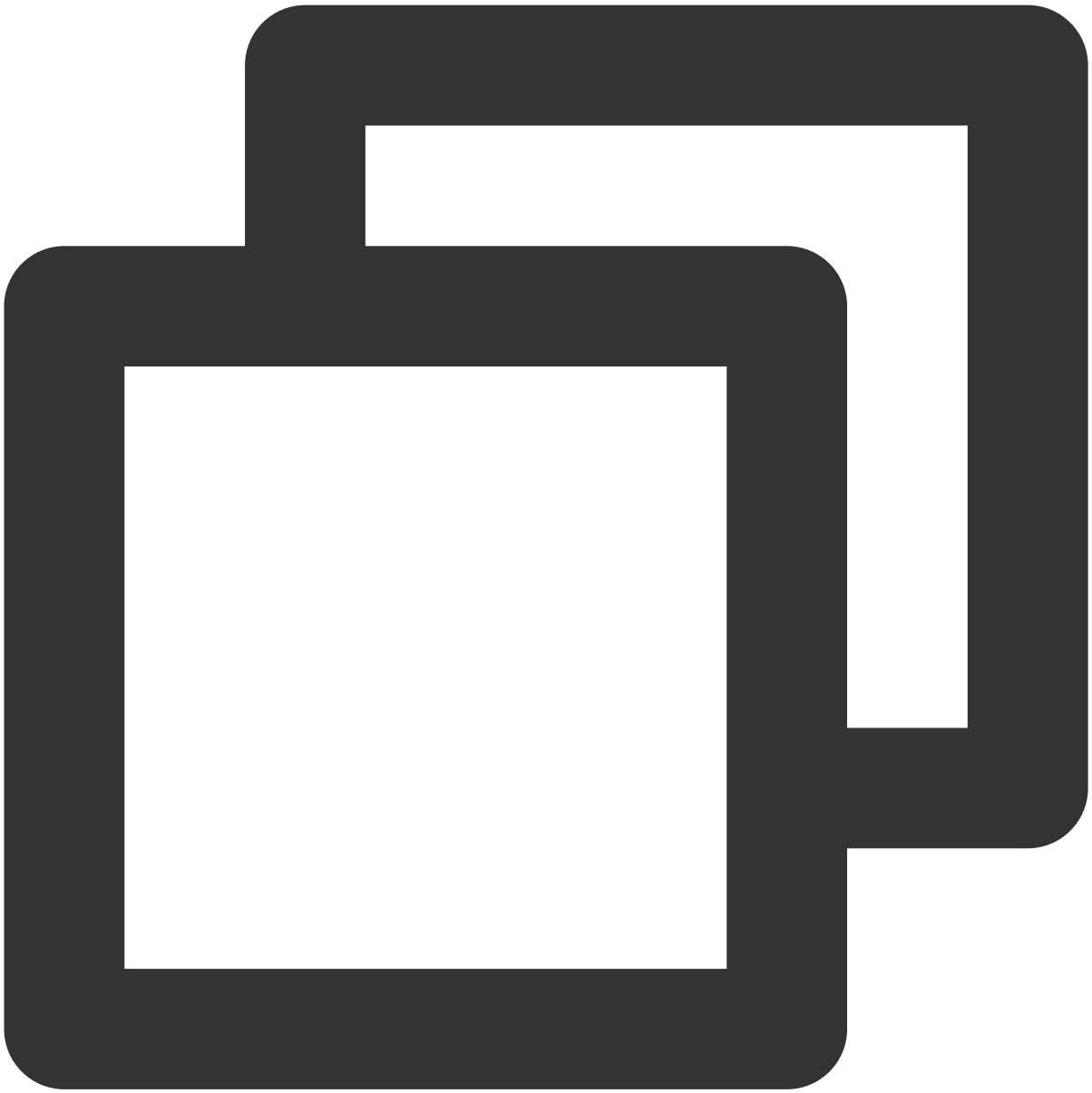
API 详情

createInstance

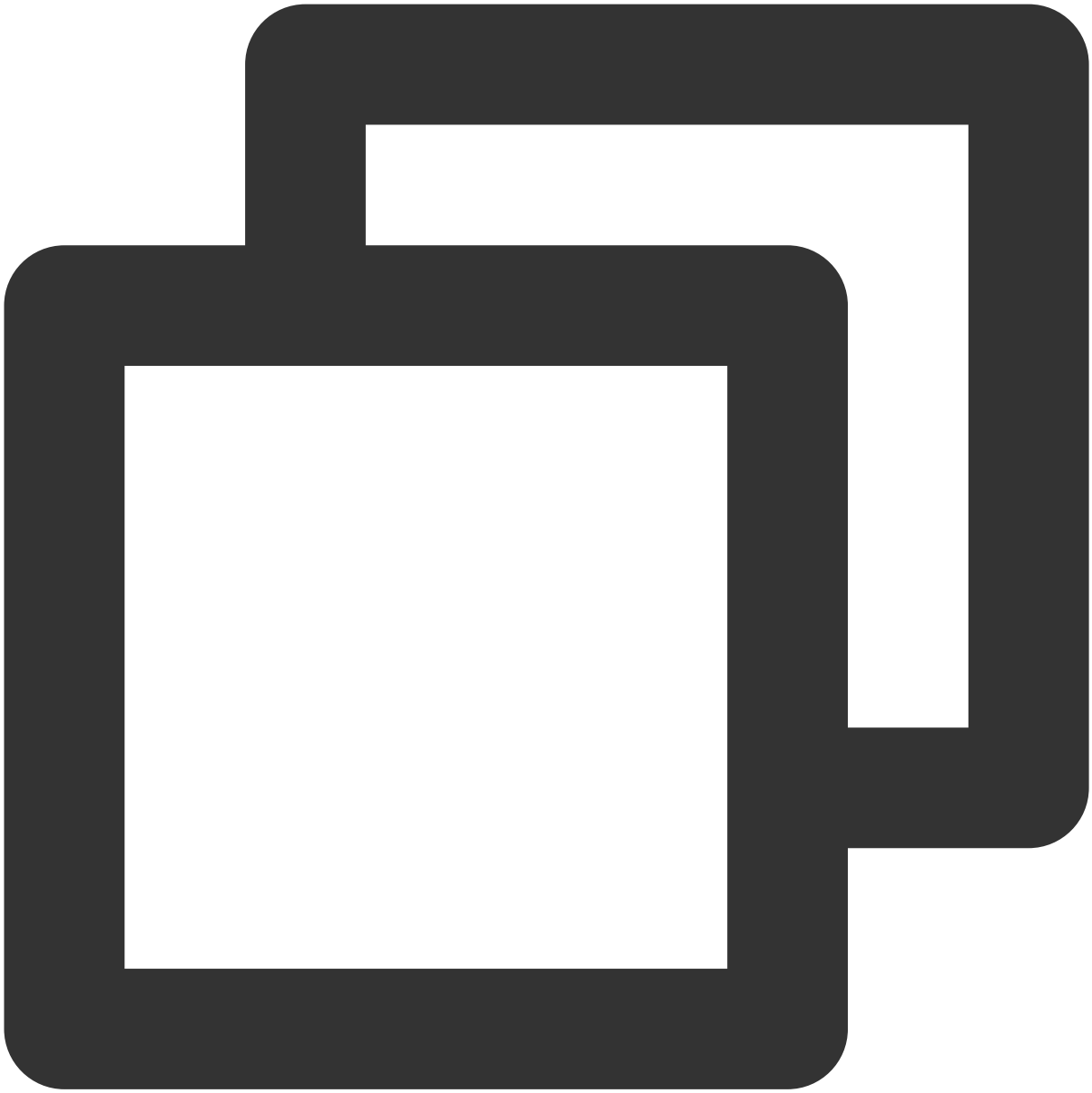
创建 TUICallKit 的单例。

Objective-C

Swift



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



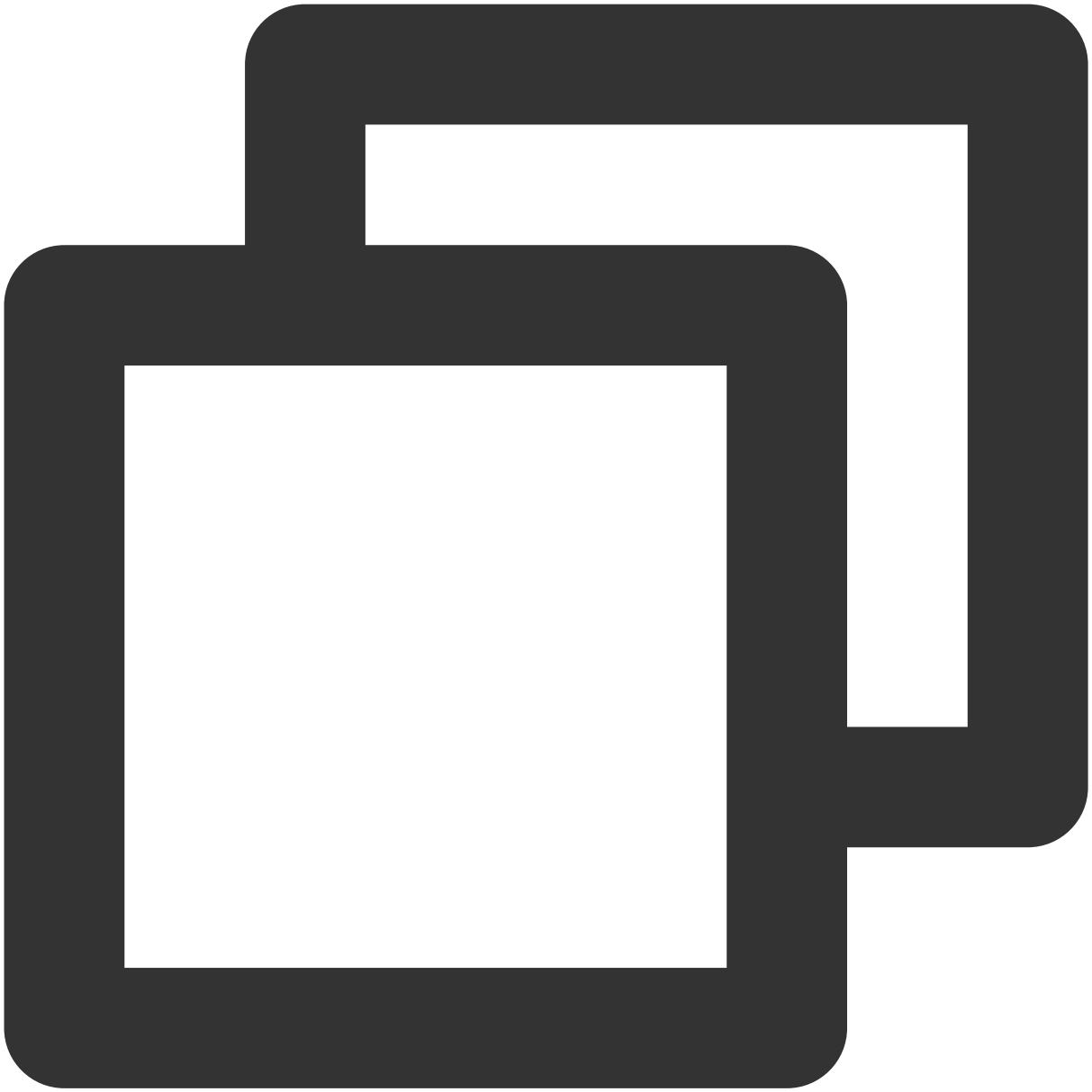
```
public static func createInstance() -> TUICallKit
```

setSelfInfo

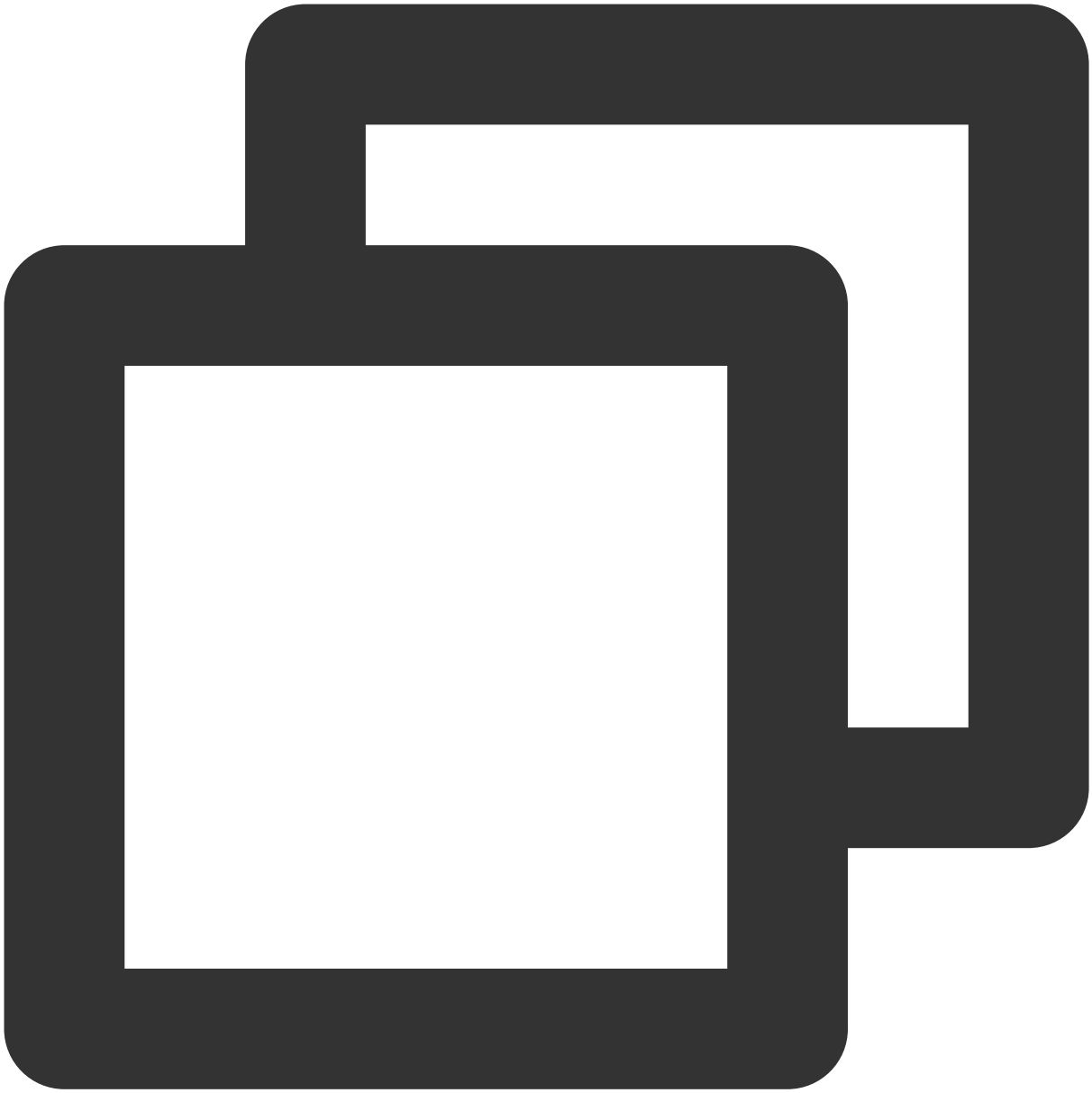
设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

Objective-C

Swift



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



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

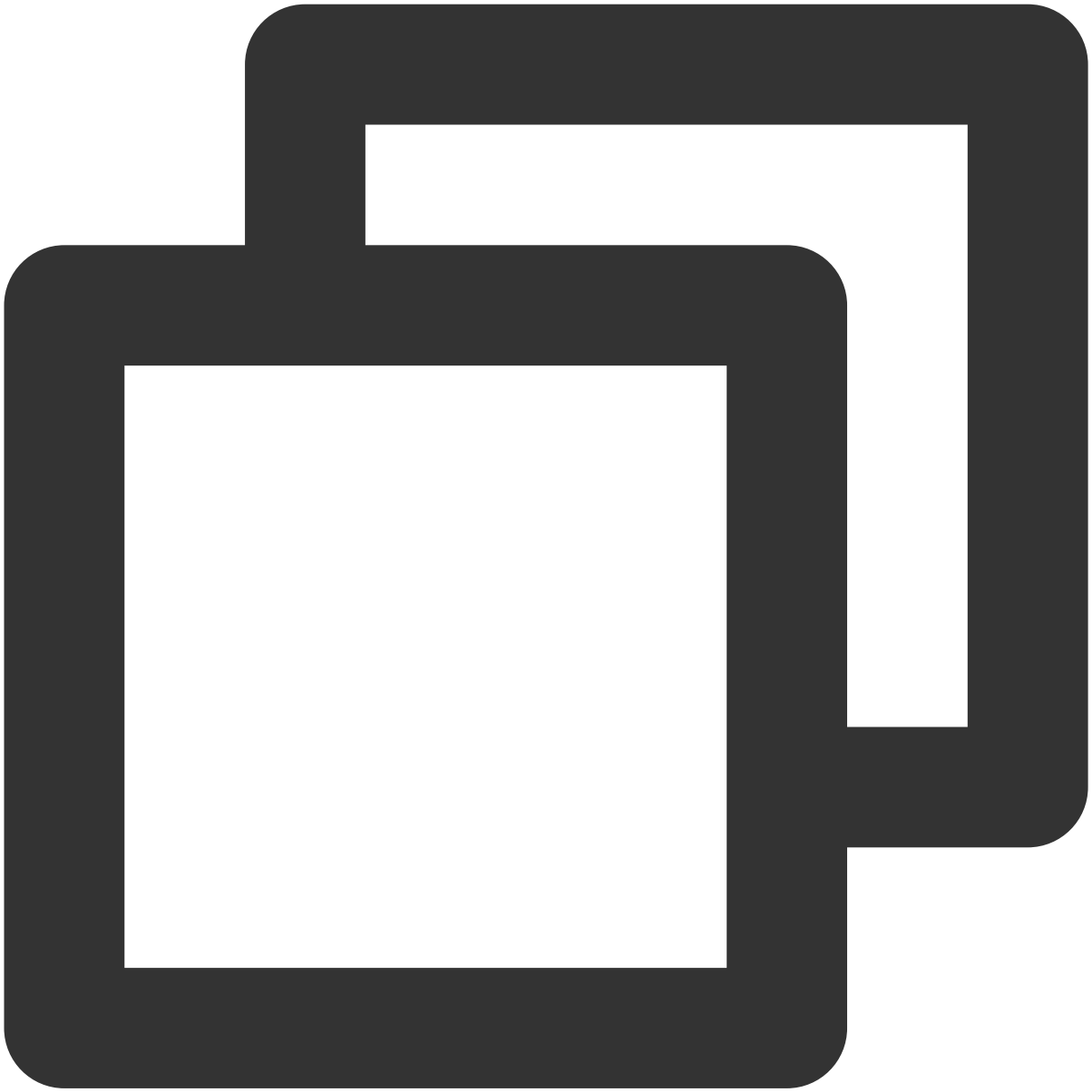
参数	类型	含义
nickName	NSString	目标用户的昵称
avatar	NSString	目标用户的头像

call

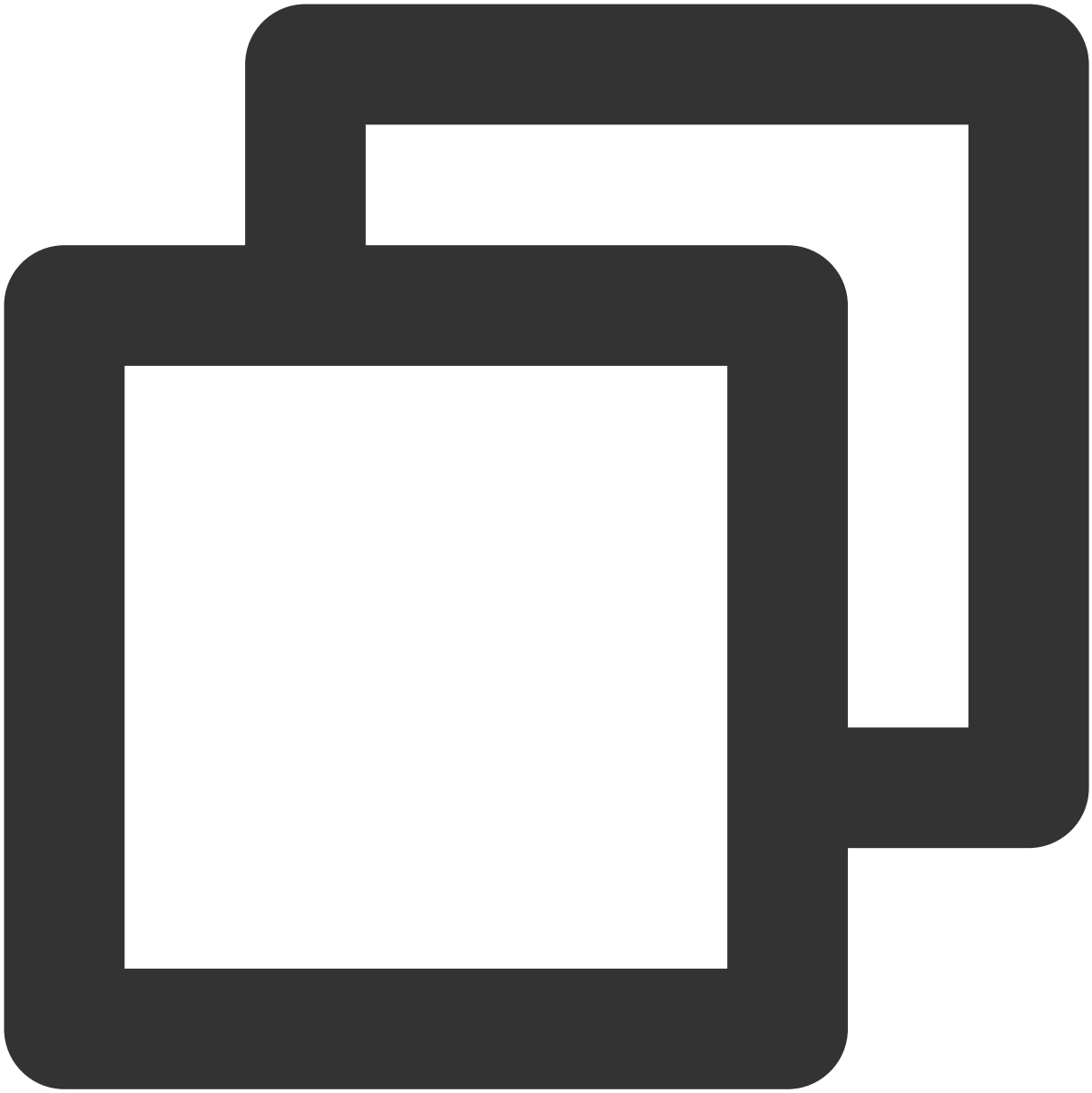
拨打电话（1v1通话）

Objective-C

Swift



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



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

参数如下表所示：

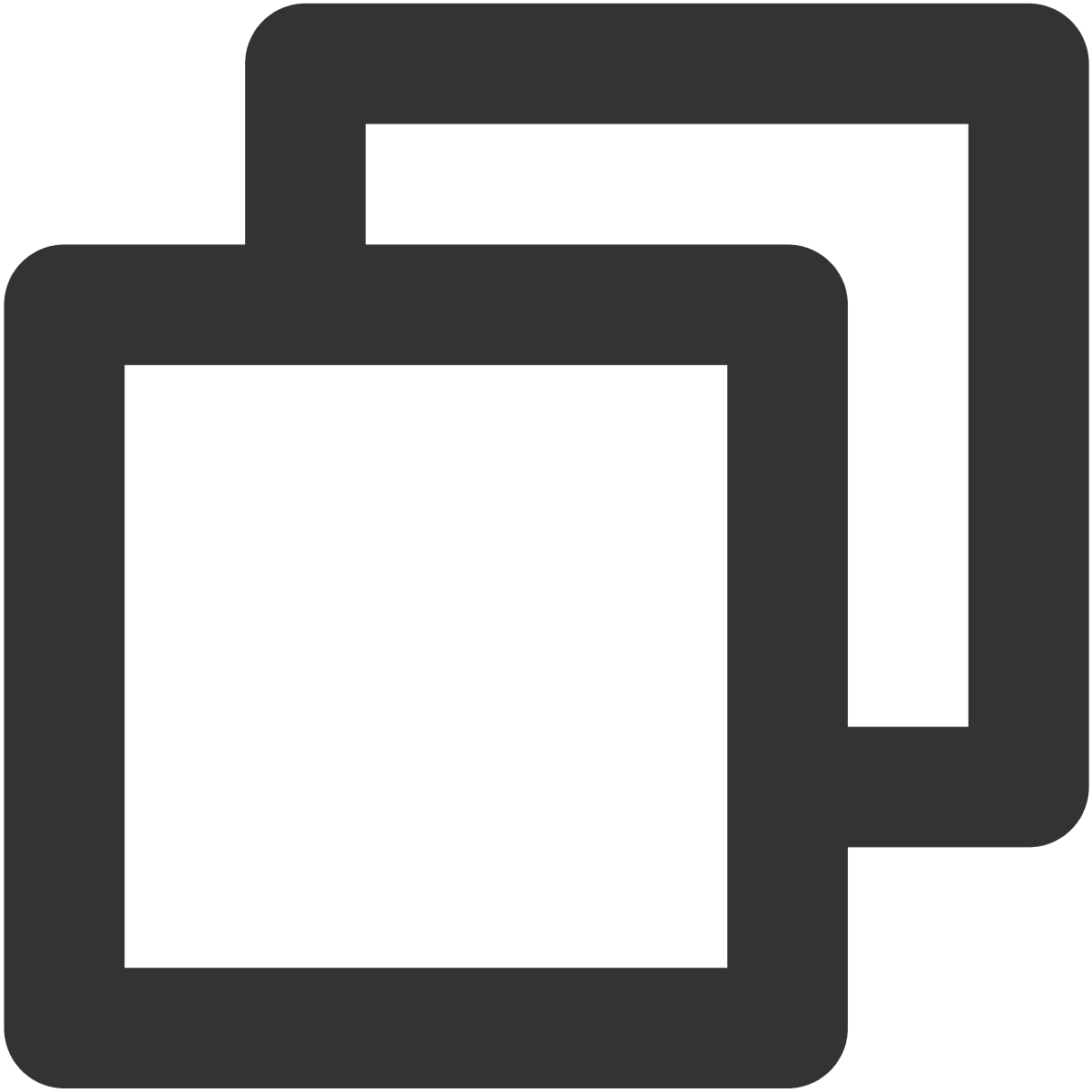
参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

call

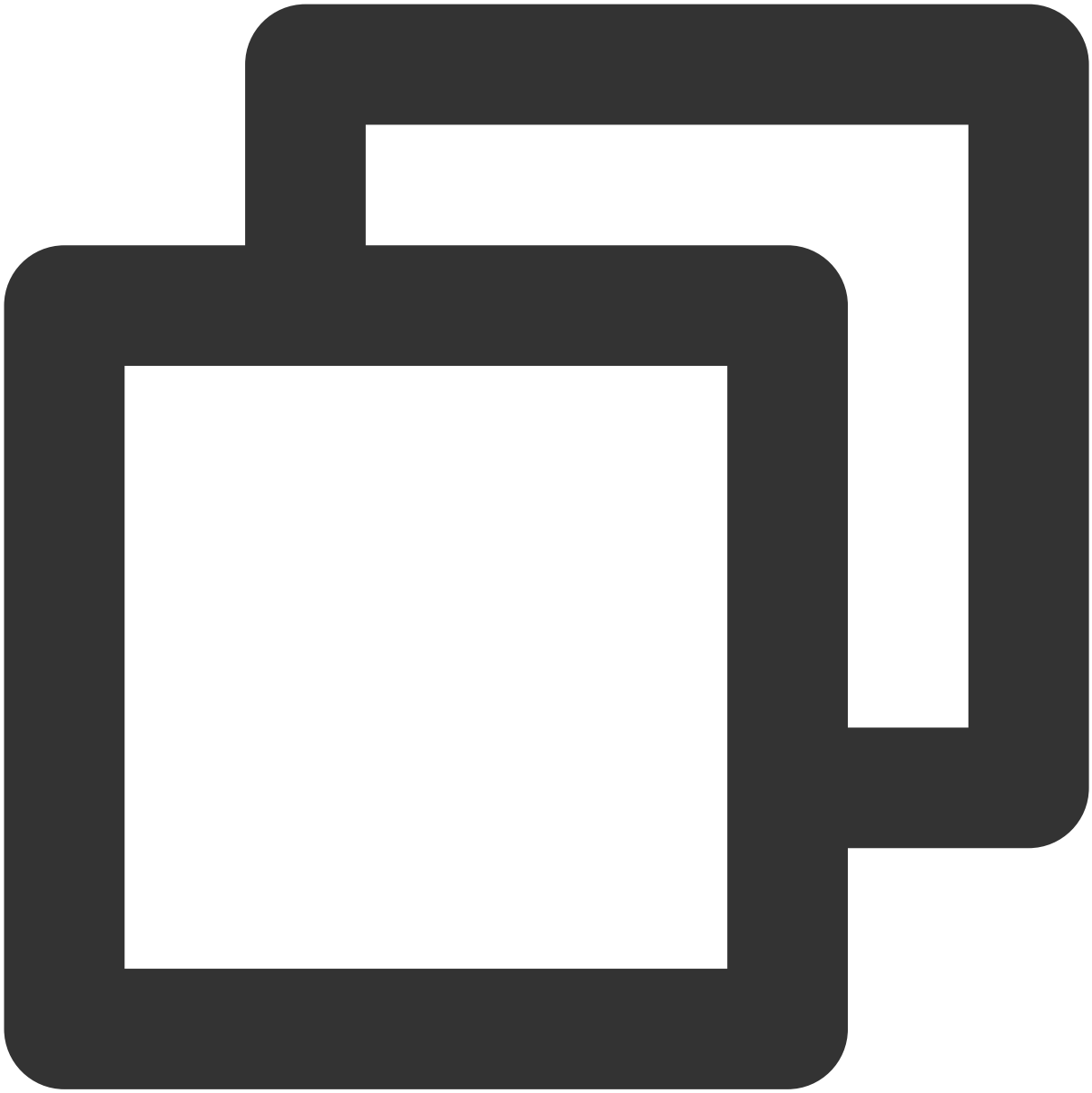
拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。

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

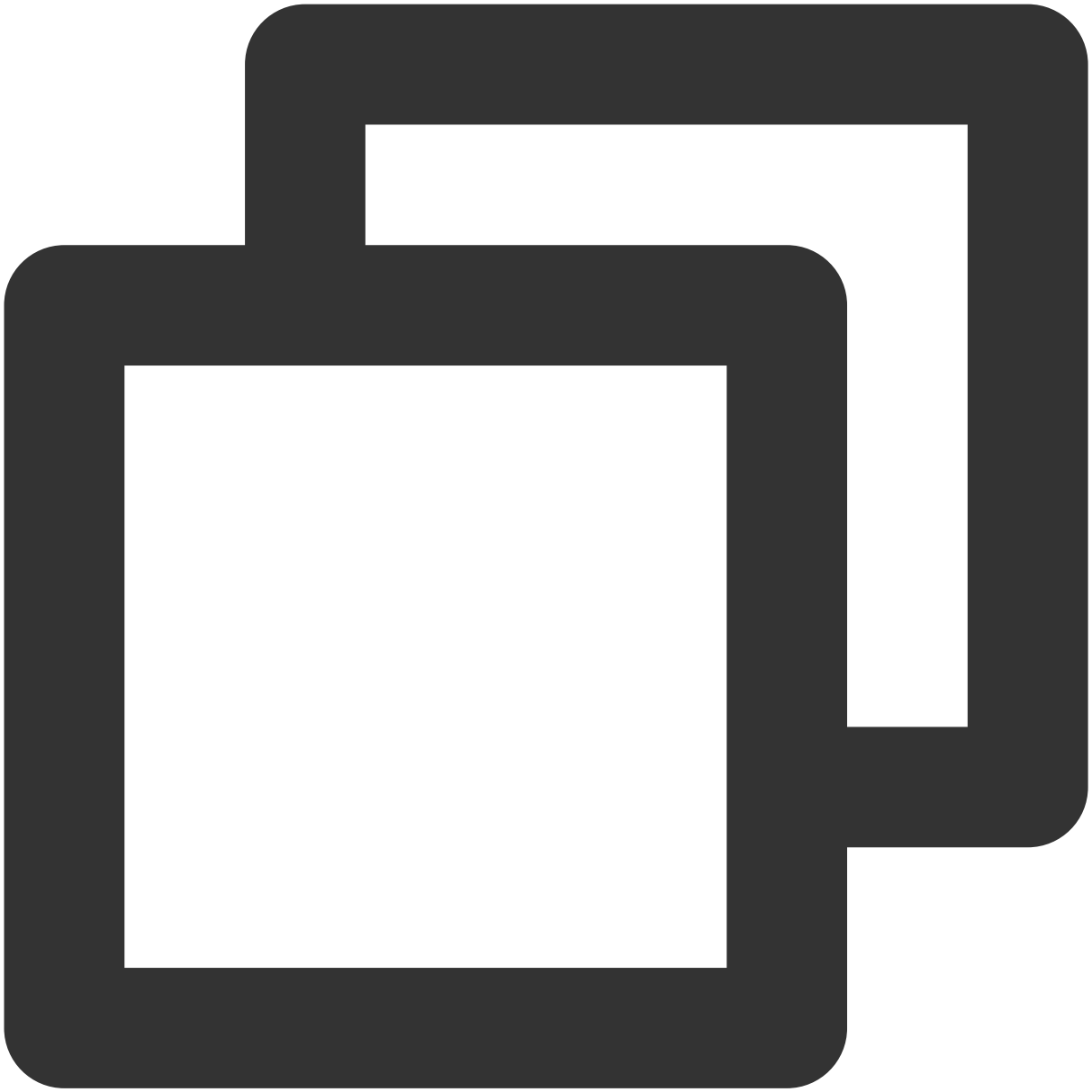
注意：

使用群组通话前需要创建 Chat 群组，如果已经创建，请忽略。

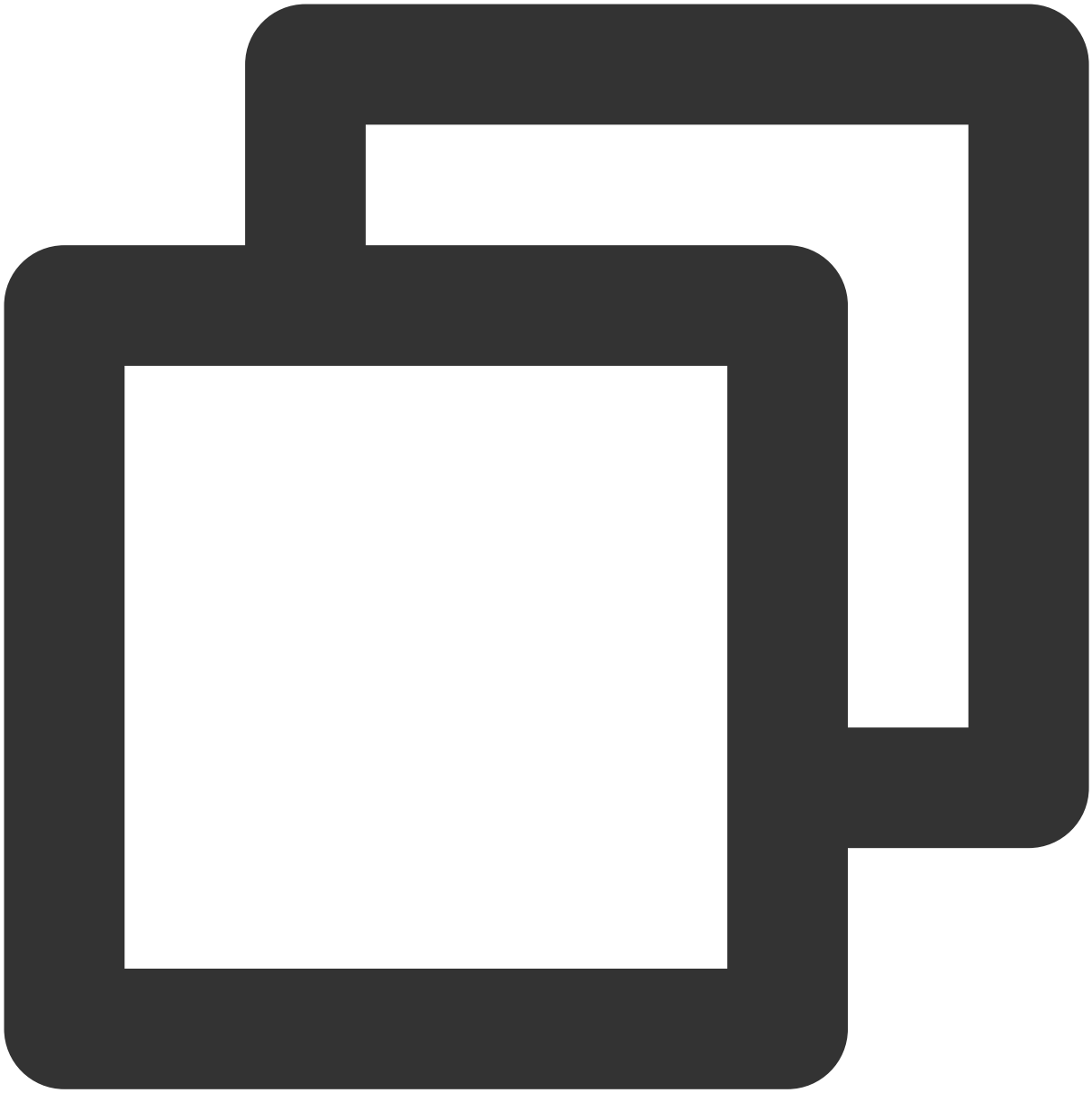
群组的创建详见：Chat 群组管理，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。

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。

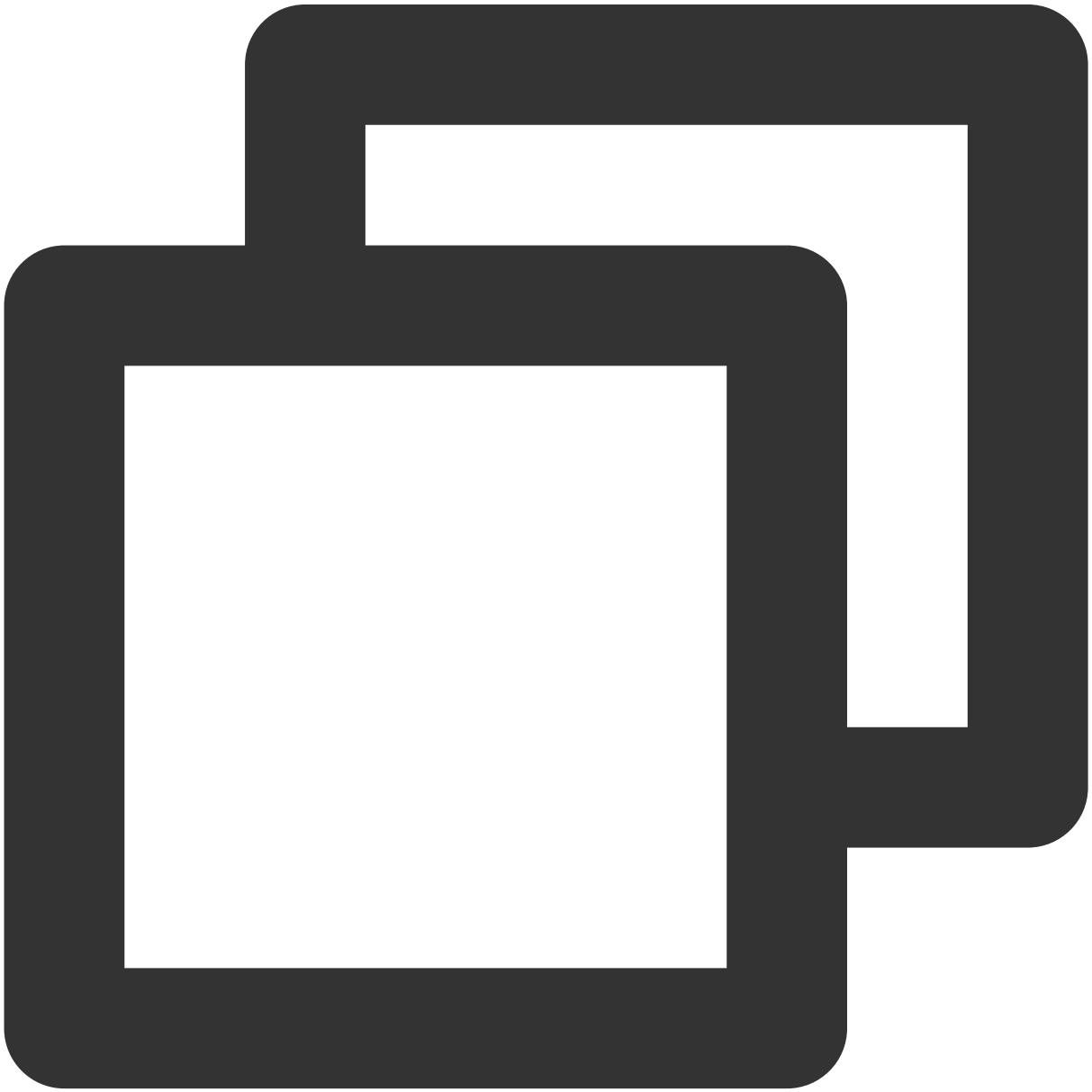
注意：

使用群组通话前需要创建 Chat 群组，如果已经创建，请忽略。

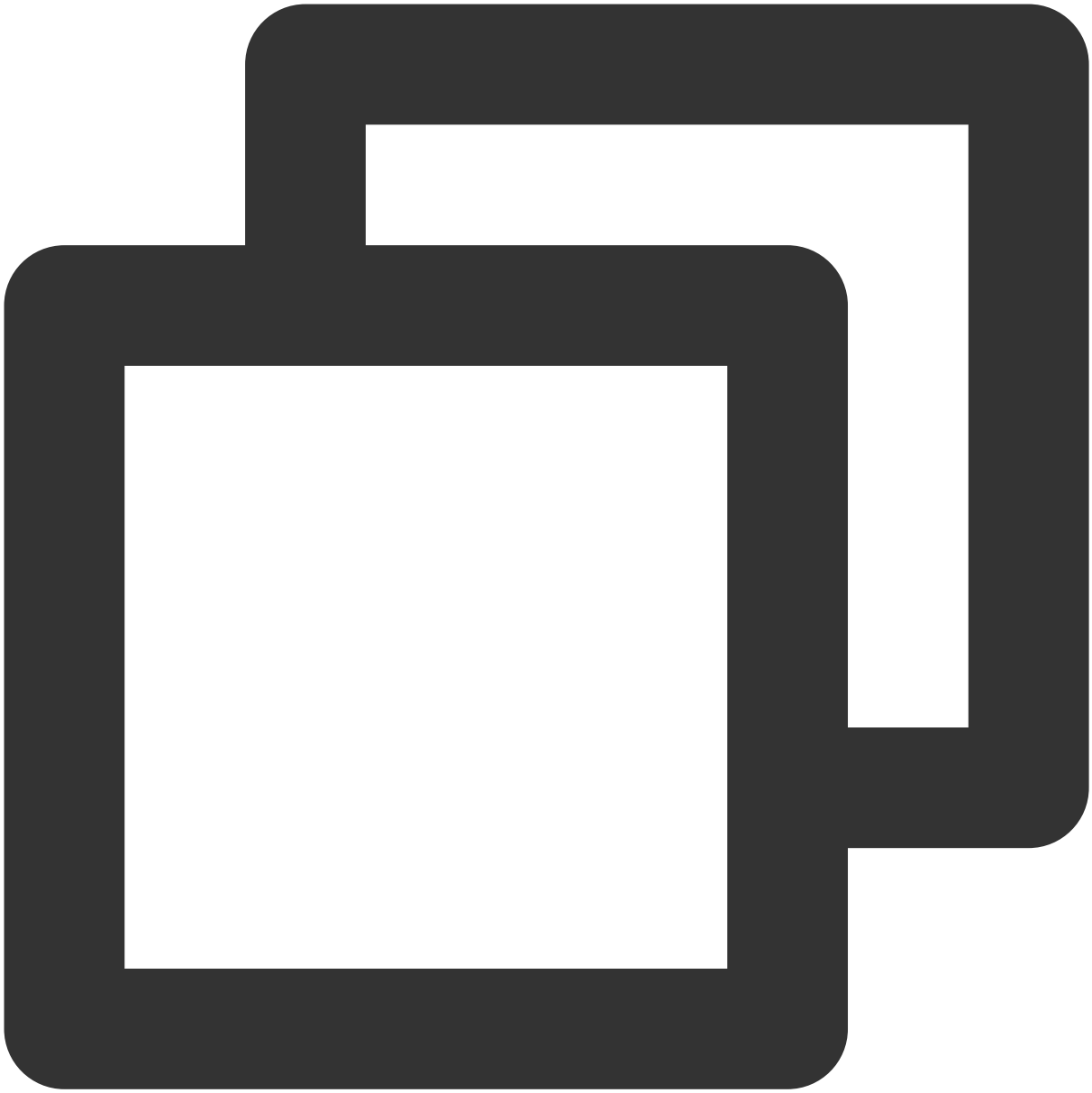
群组的创建详见：Chat 群组管理，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。

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

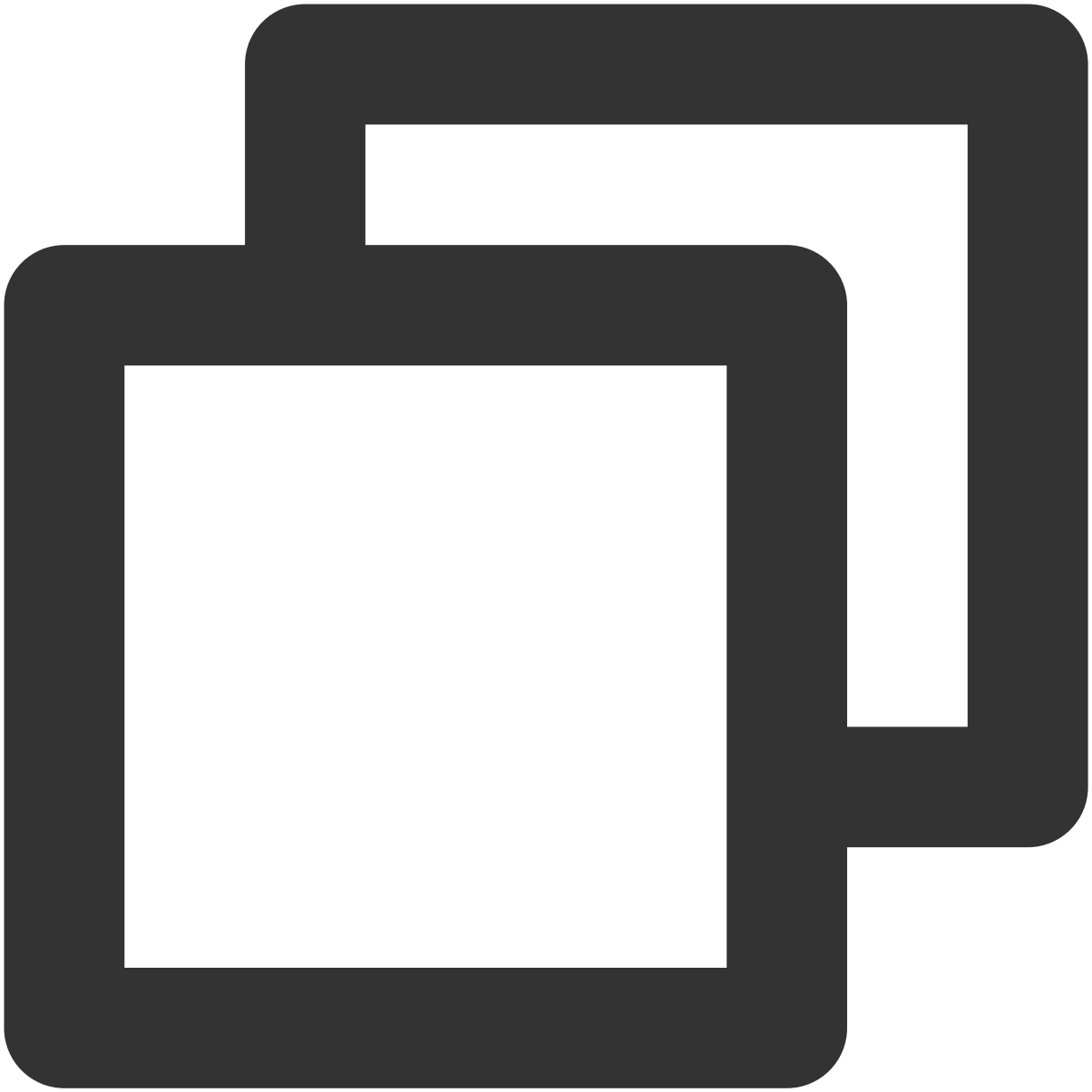
注意：

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

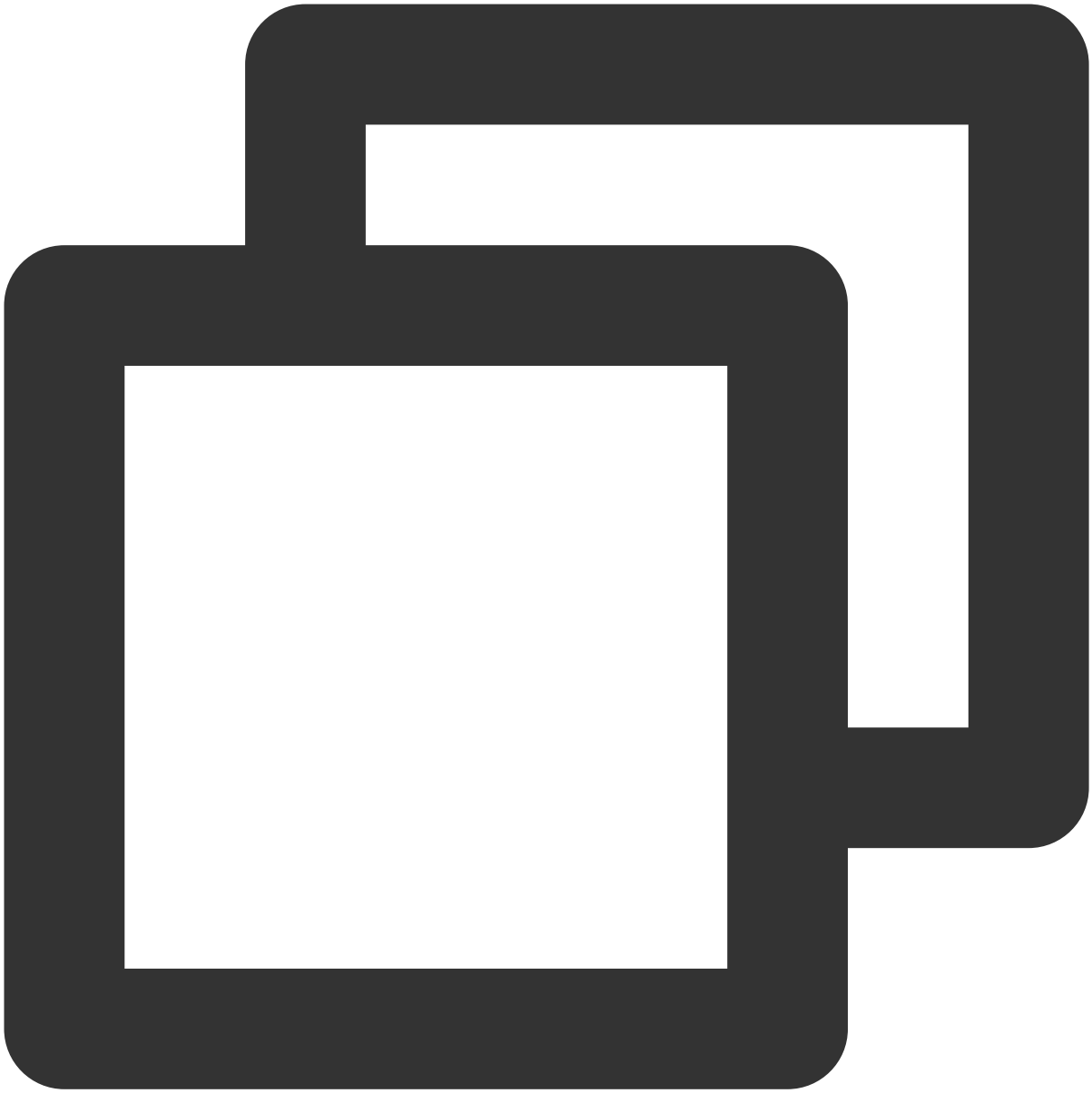
群组的创建详见：Chat 群组管理，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。

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。

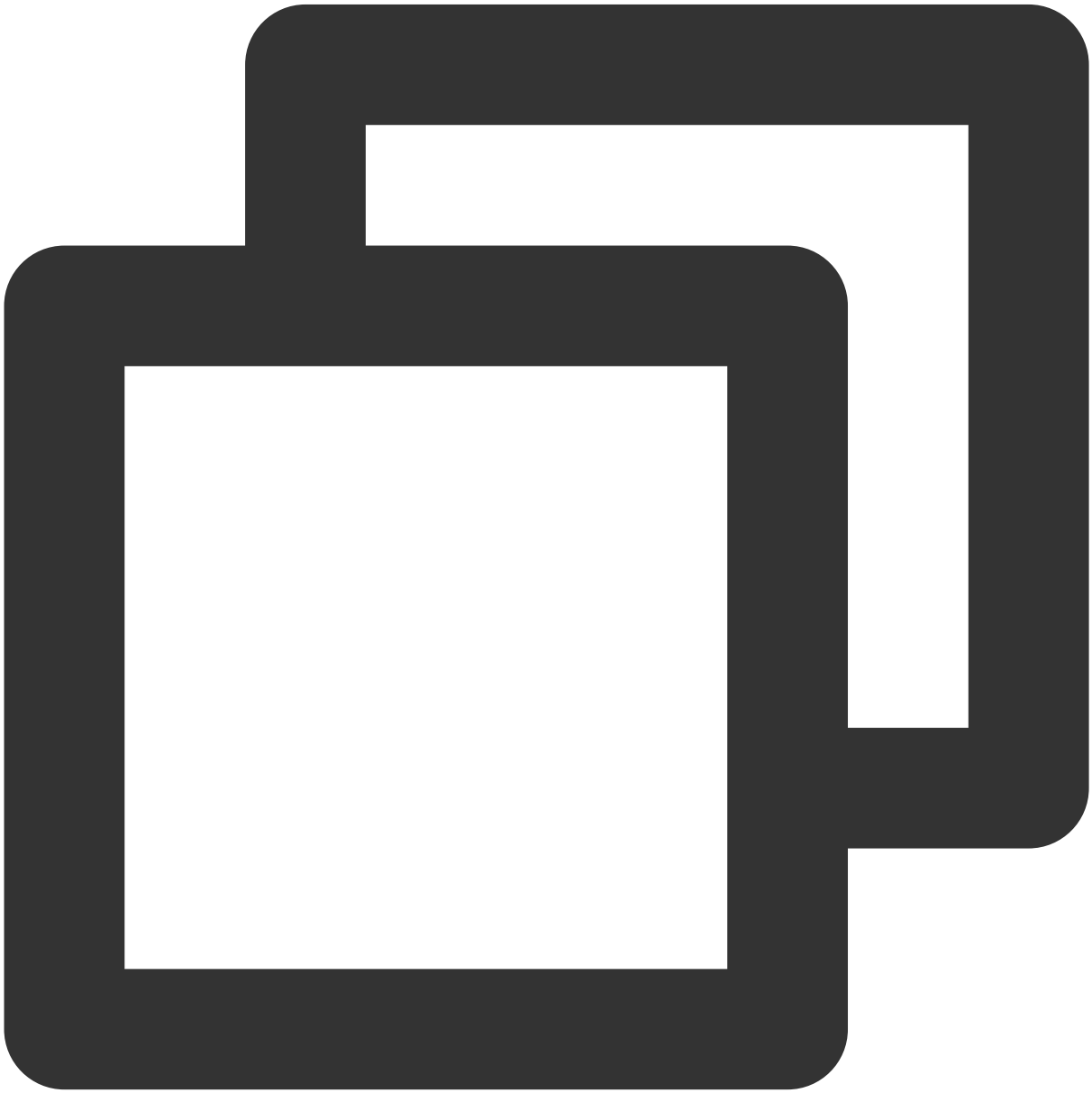
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

铃声设置后与设备绑定，更换用户，铃声依旧会生效。

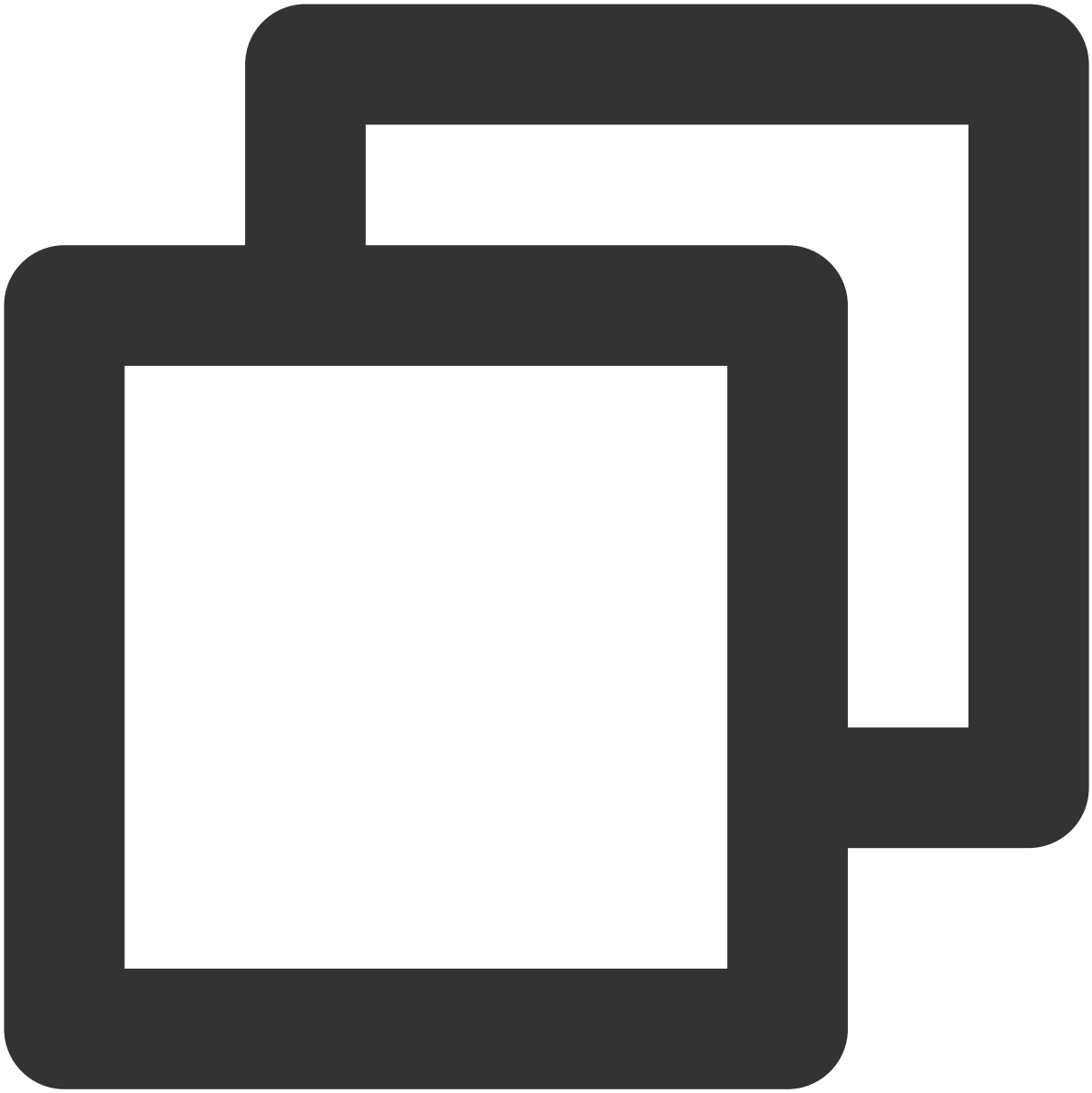
如需恢复默认铃声， `filePath` 传空即可。

Objective-C

Swift



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



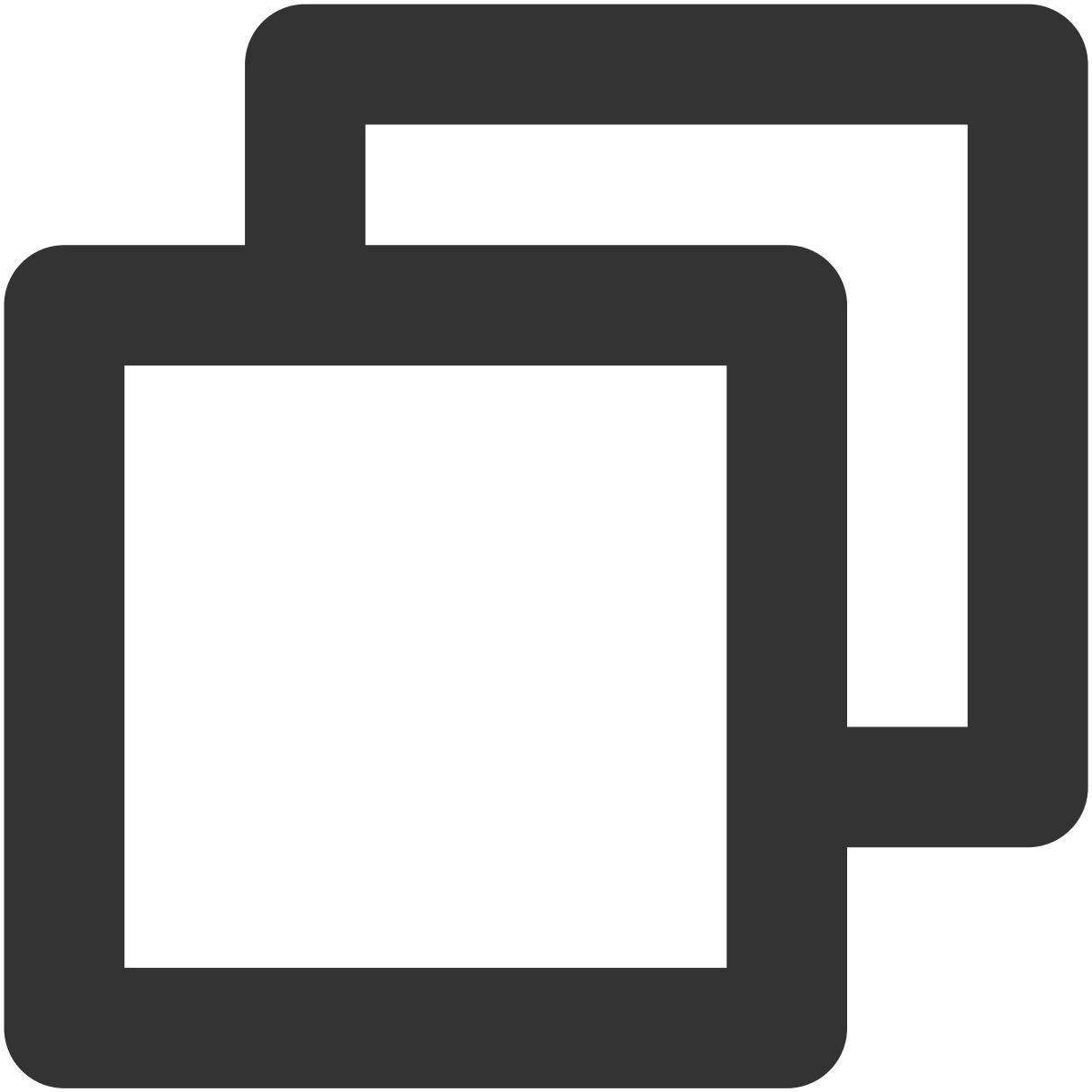
```
public func setCallingBell(filePath: String)
```

enableMuteMode

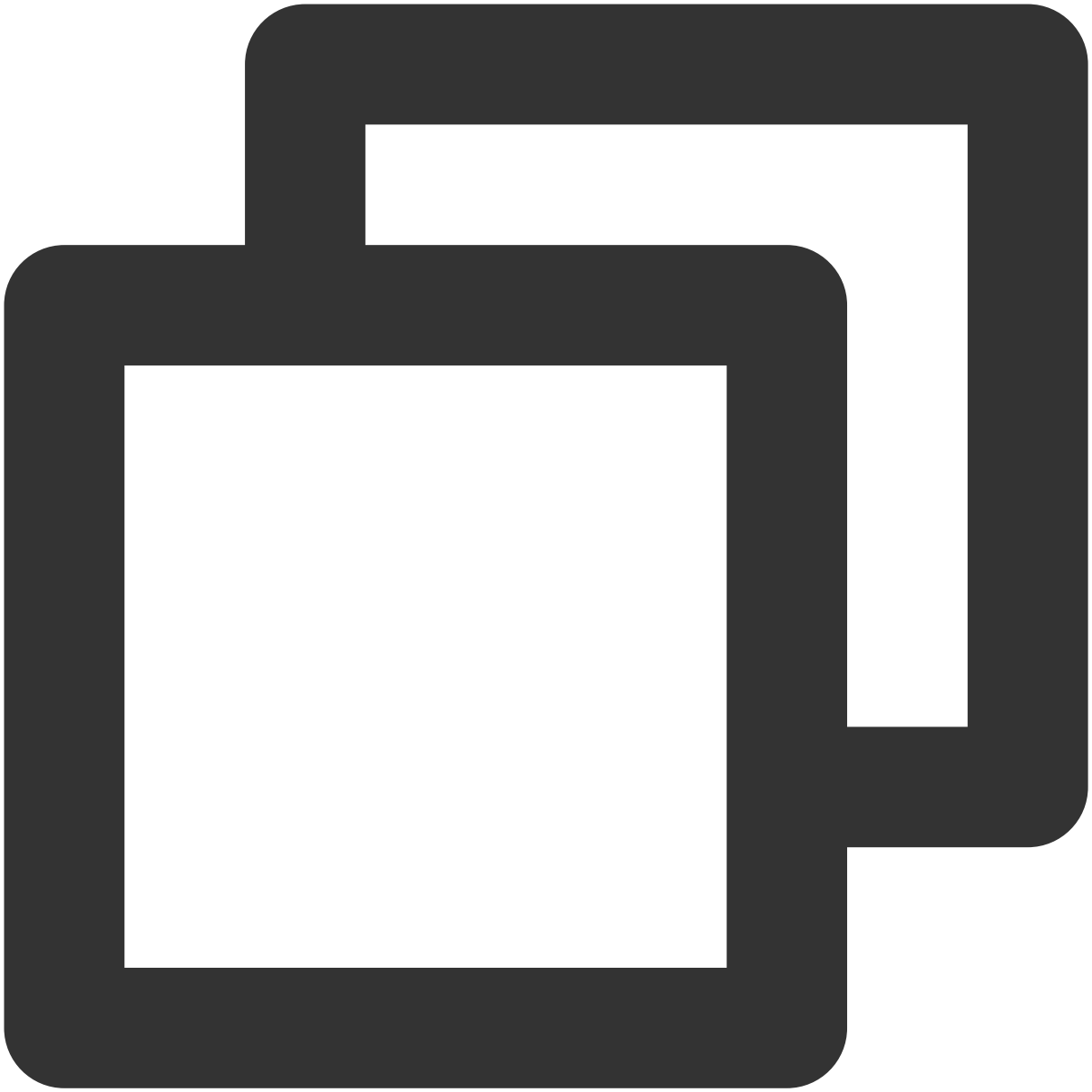
开启/关闭静音模式。

Objective-C

Swift



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



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

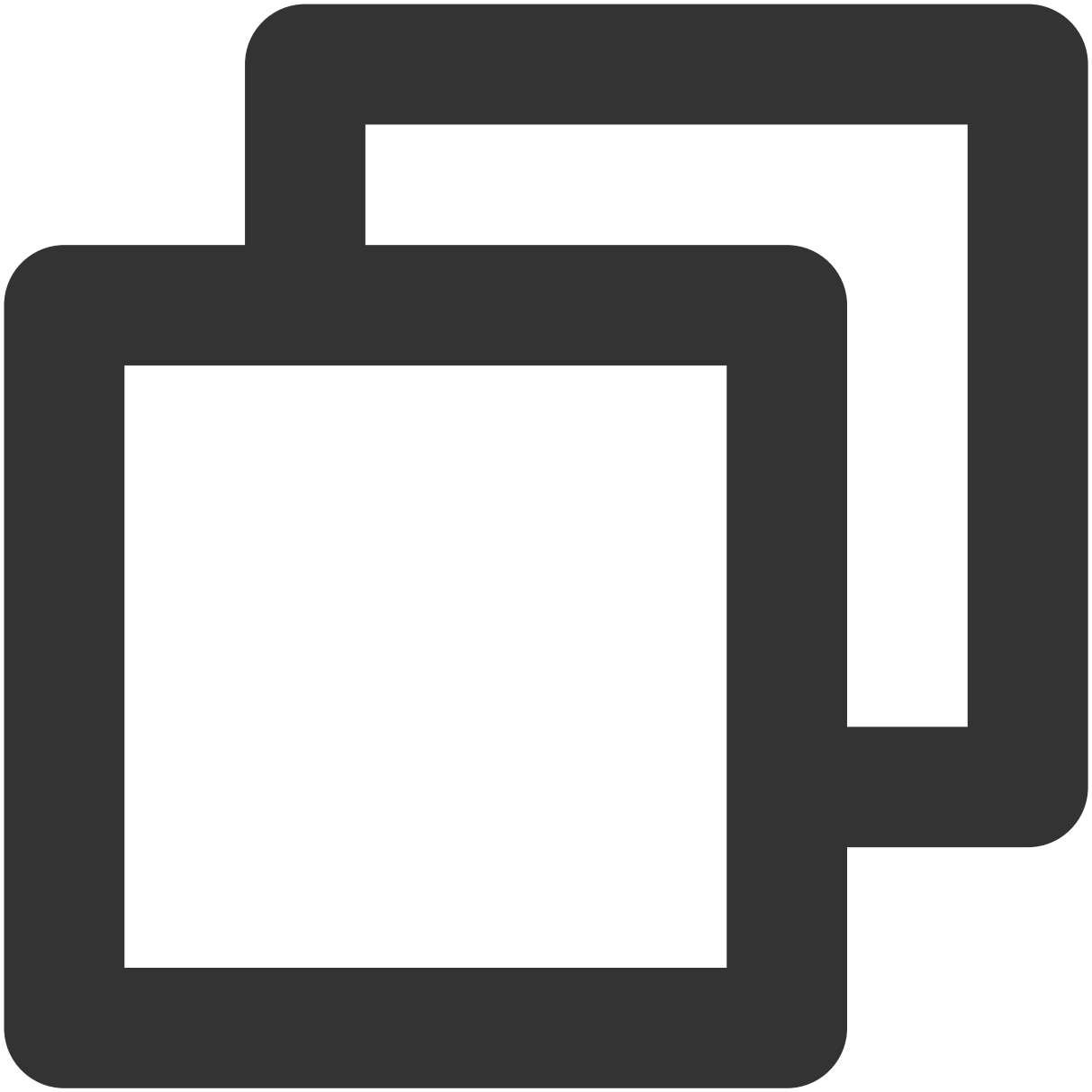
enableFloatWindow

开启/关闭悬浮窗功能。

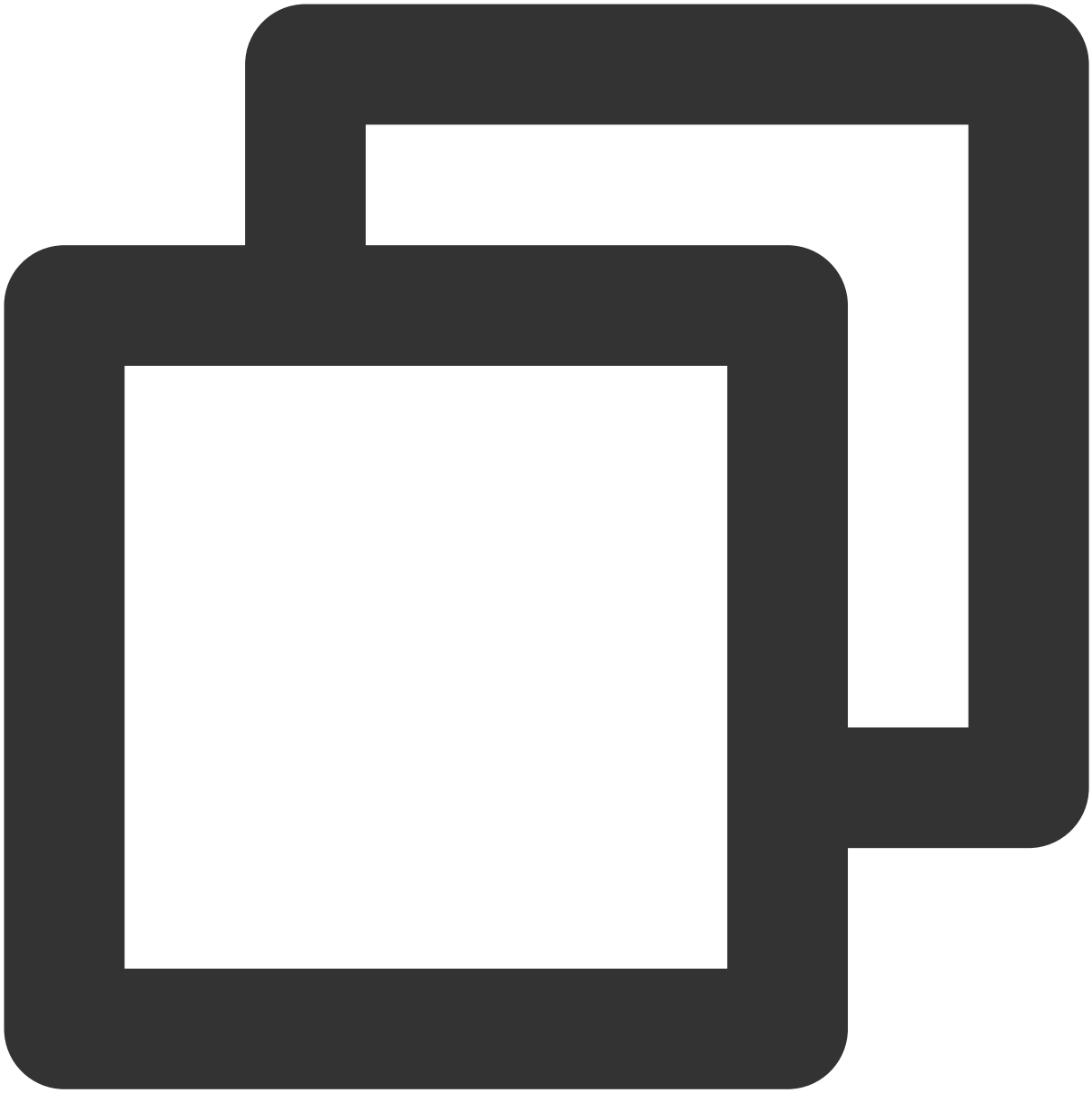
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Objective-C

Swift



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

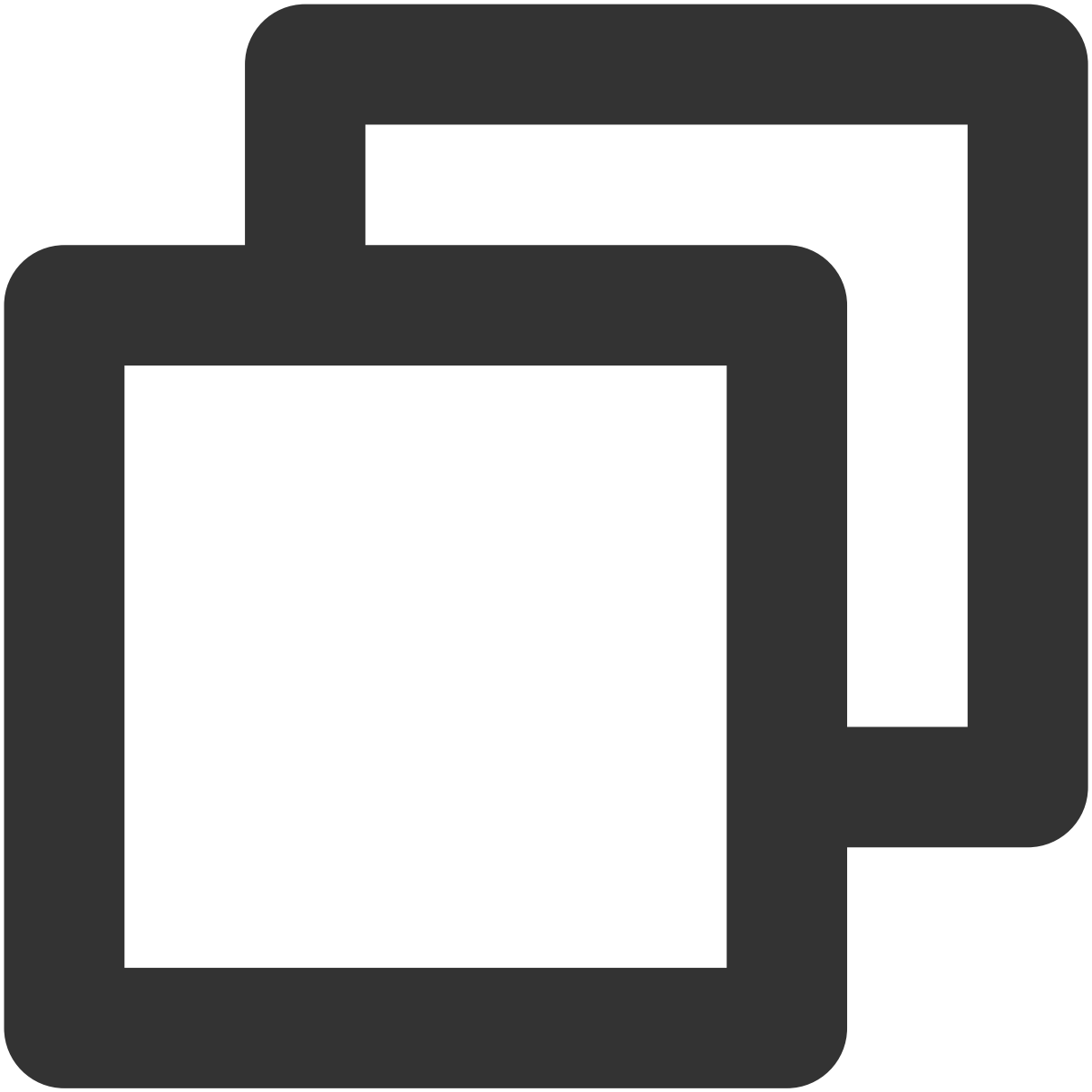


```
public func enableFloatWindow(enable: Bool)
```

enableIncomingBanner

开启/关闭来电横幅显示。

默认为 `false` ，被叫端收到邀请后默认弹出全屏通话等待界面，开启后先展示一个横幅，然后根据需要拉起全屏通话界面。



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

TUICallEngine

最近更新时间：2024-07-25 17:49:30

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

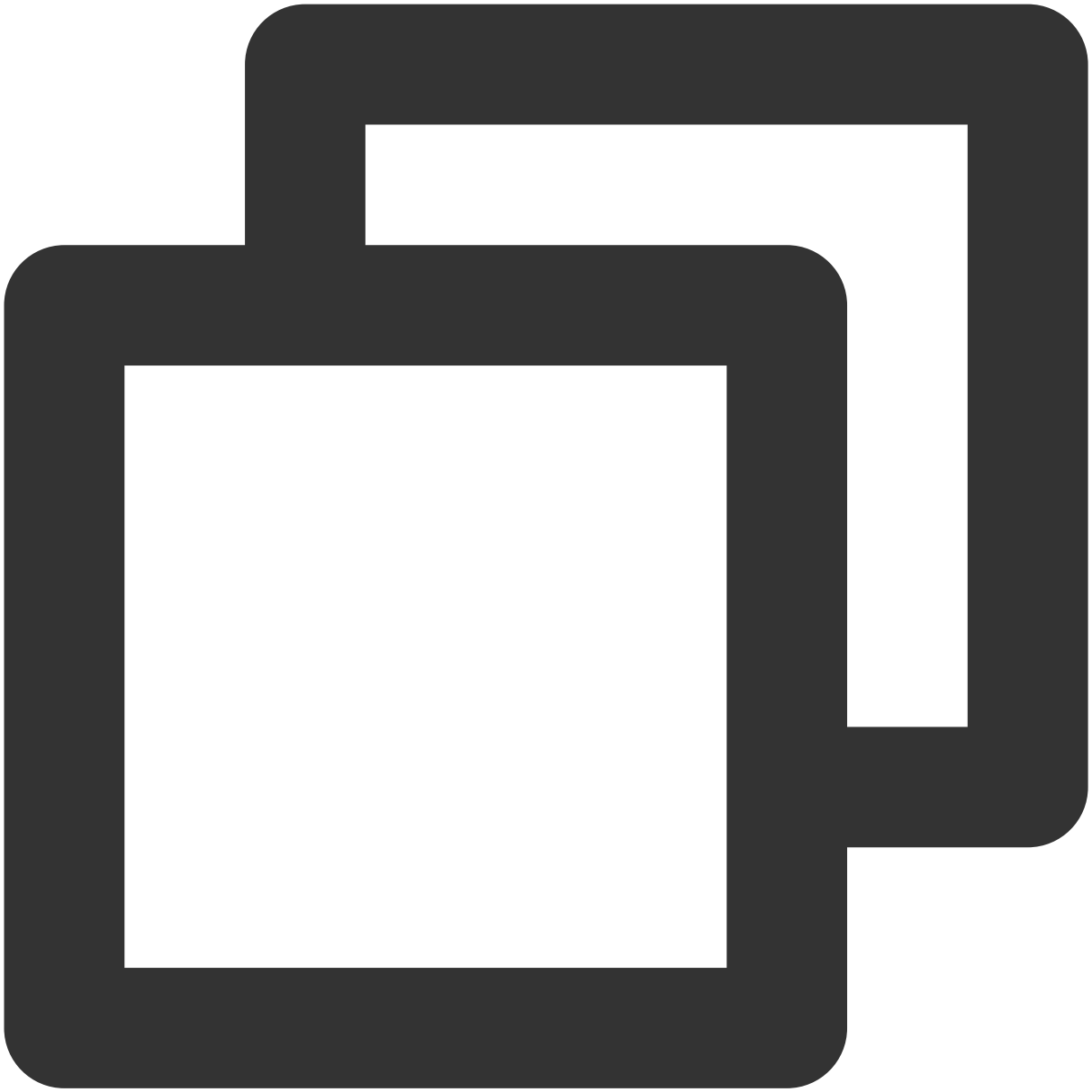
版权所有：腾讯云计算（北京）有限责任公司

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（群组通话版套餐支持）
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

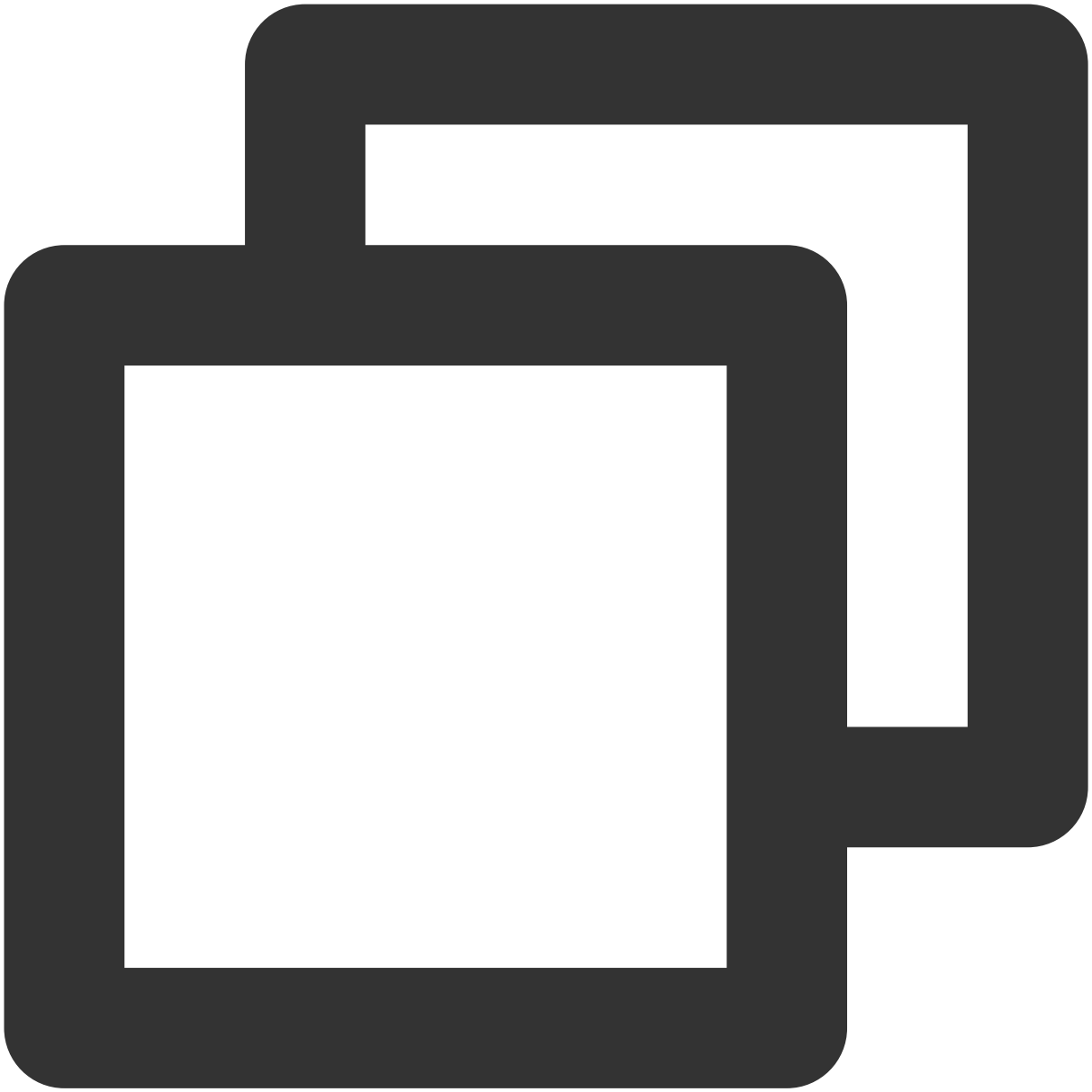
创建 TUICallEngine 的单例。



```
- (TUICallEngine *)createInstance;
```

destroyInstance

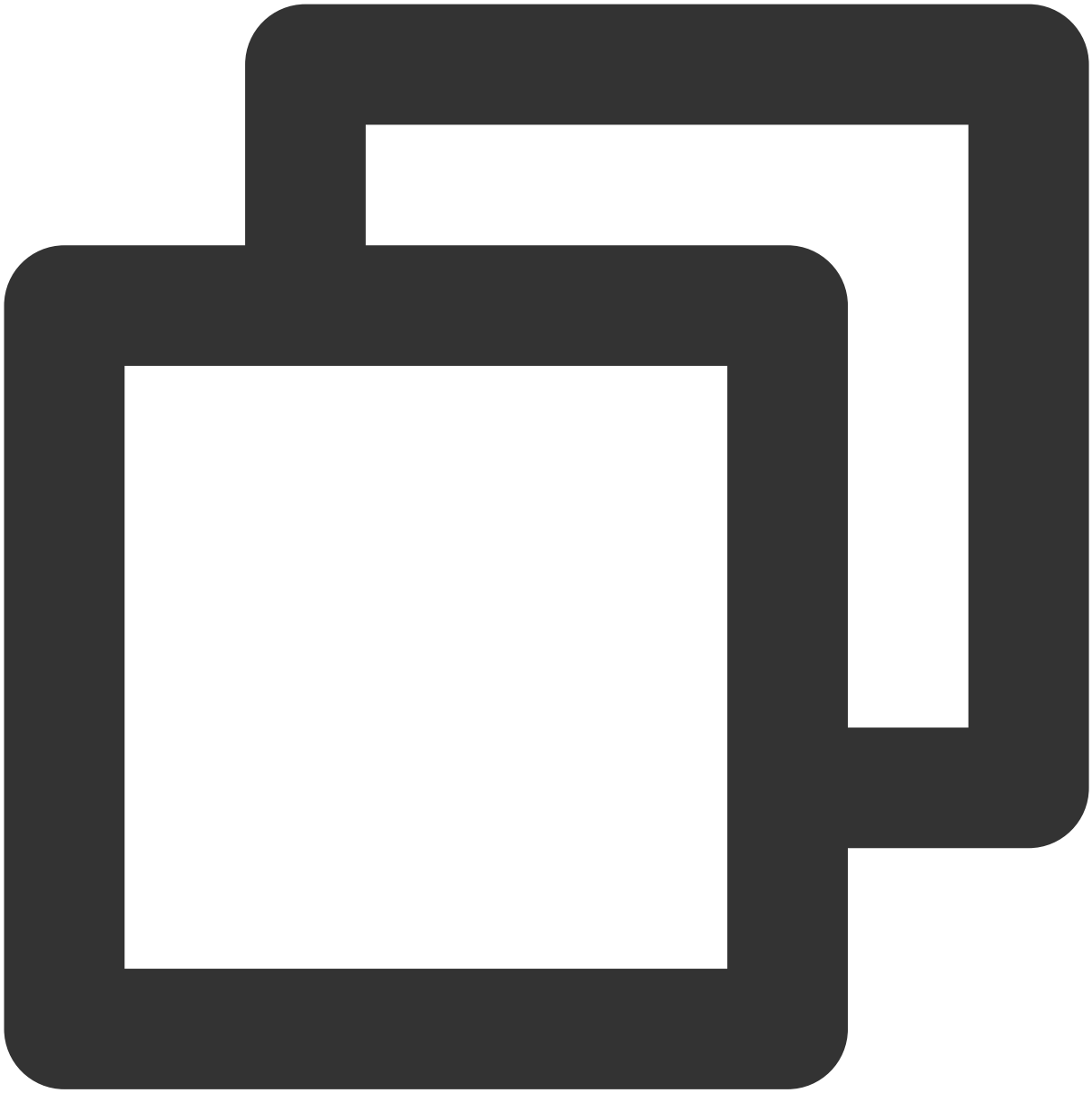
销毁 TUICallEngine 的单例。



```
- (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
```

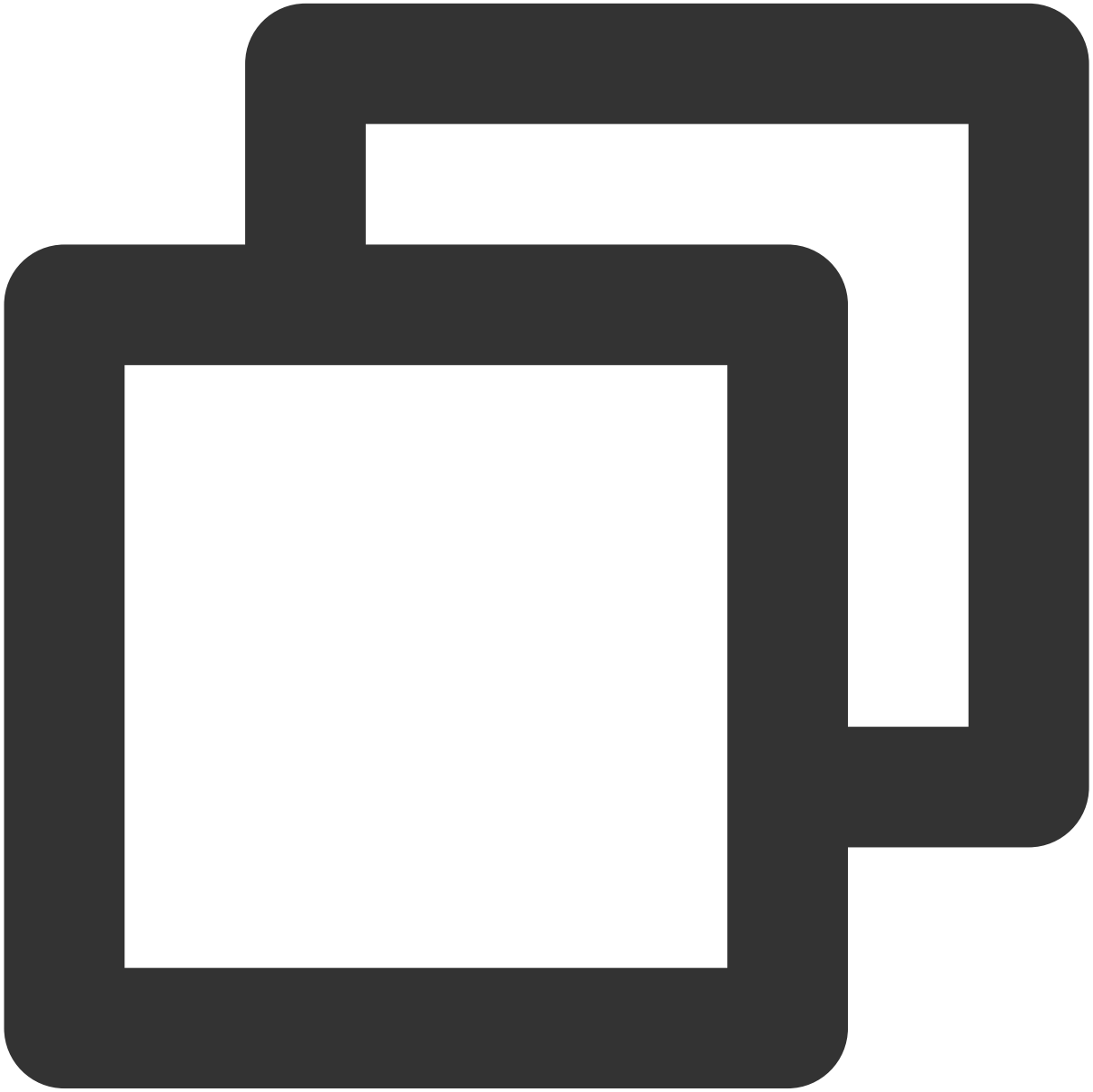
参数如下表所示：

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

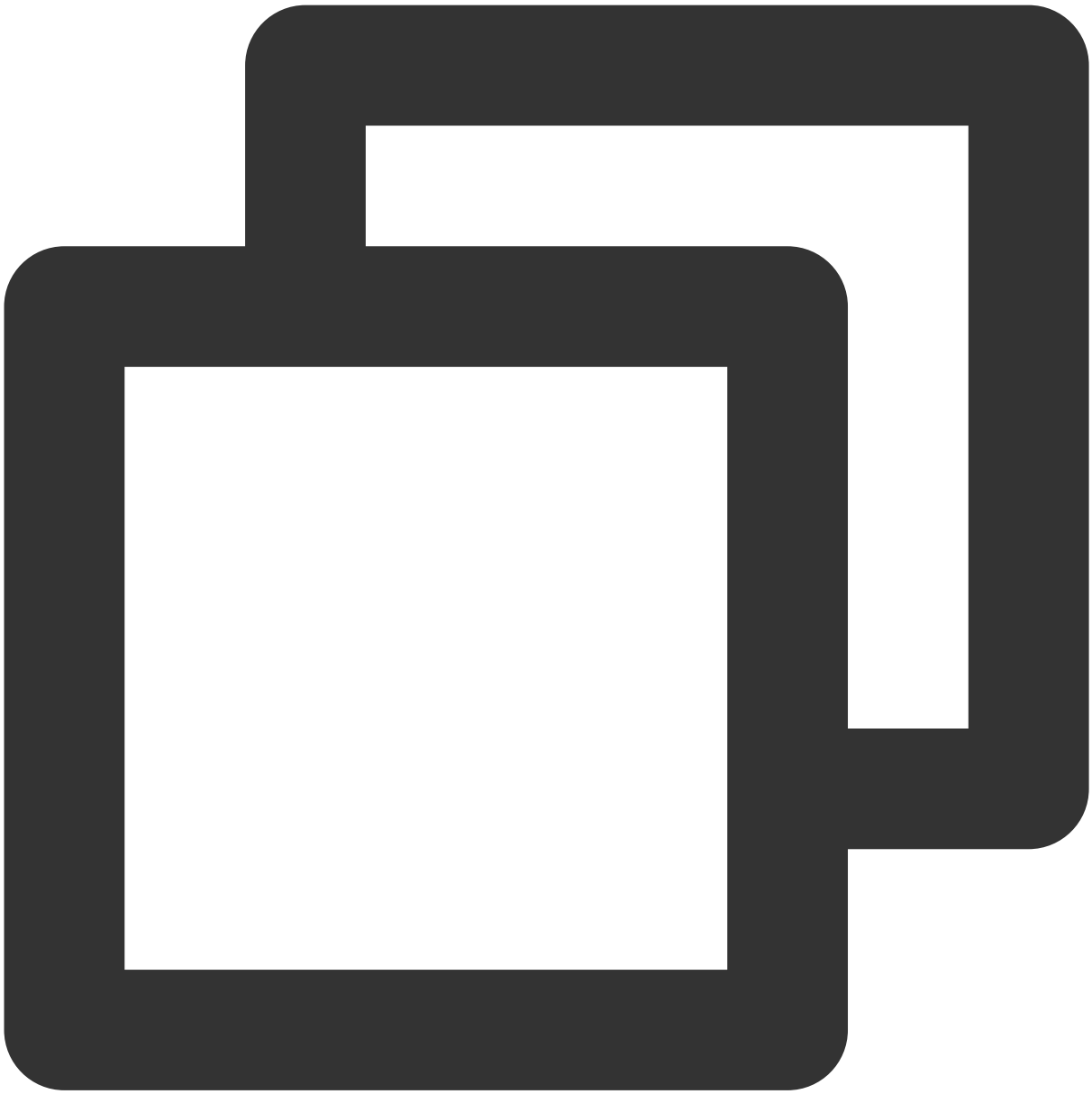
添加回调接口，您可以通过这个接听，监听 TUICallObserver 相关的事件回调。



```
- (void)addObserver:(id<TUICallObserver>)observer;
```

removeObserver

移除回调接口。



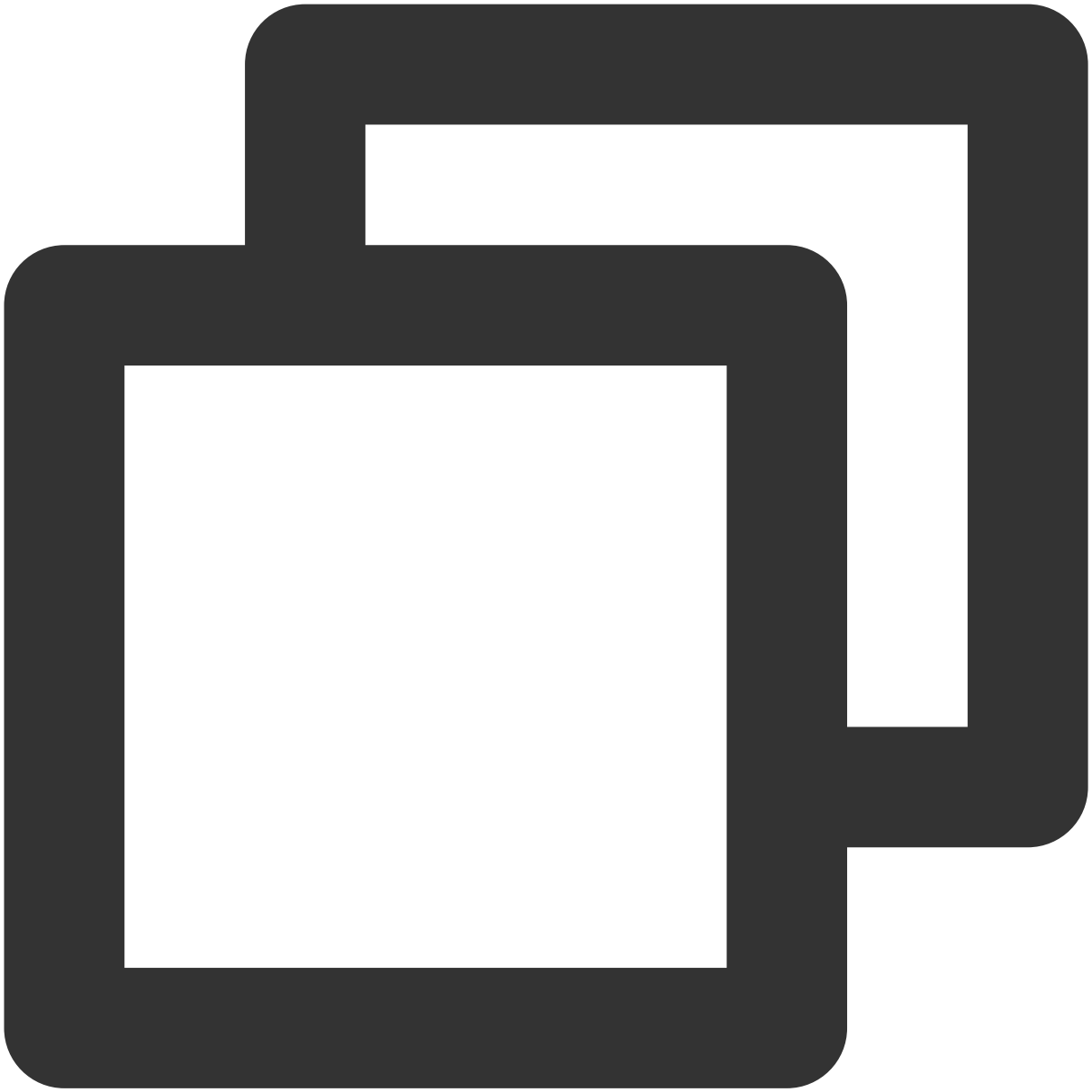
```
- (void)removeObserver:(id<TUICallObserver>)observer;
```

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

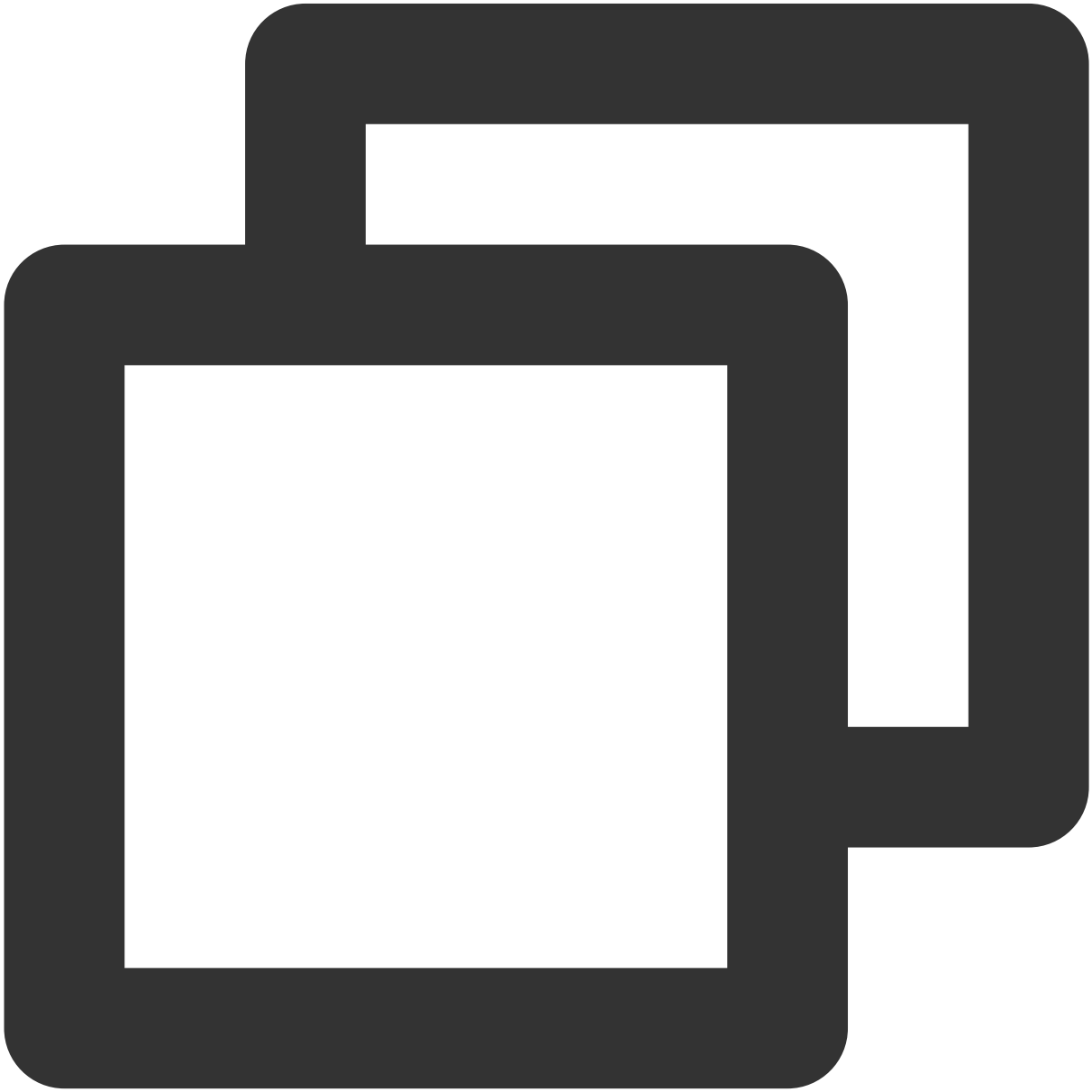
groupCall

发起群组通话。

注意：

使用群组通话前需要创建 Chat 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！

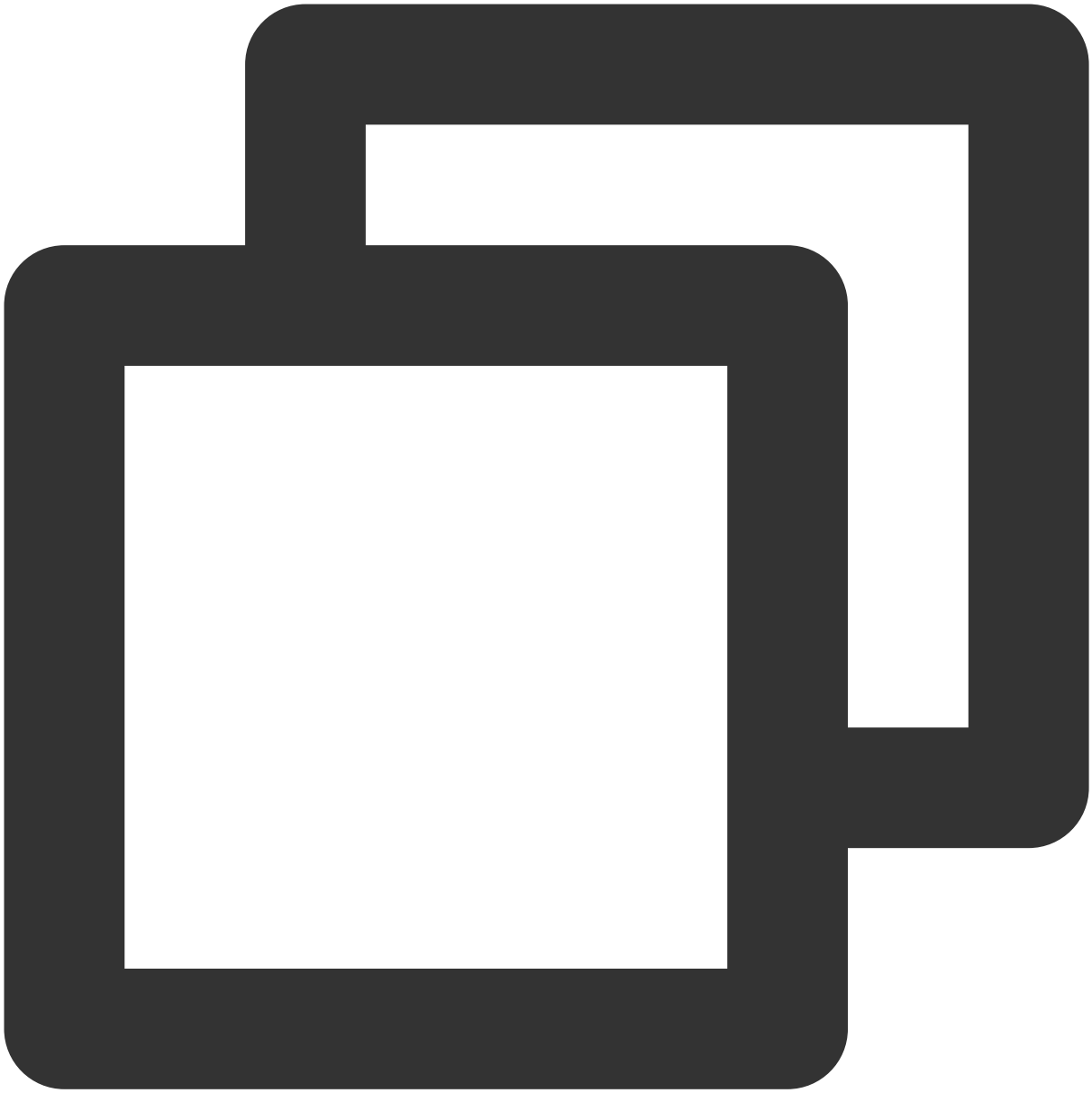


```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray <NSString *> *)userIdList
```

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

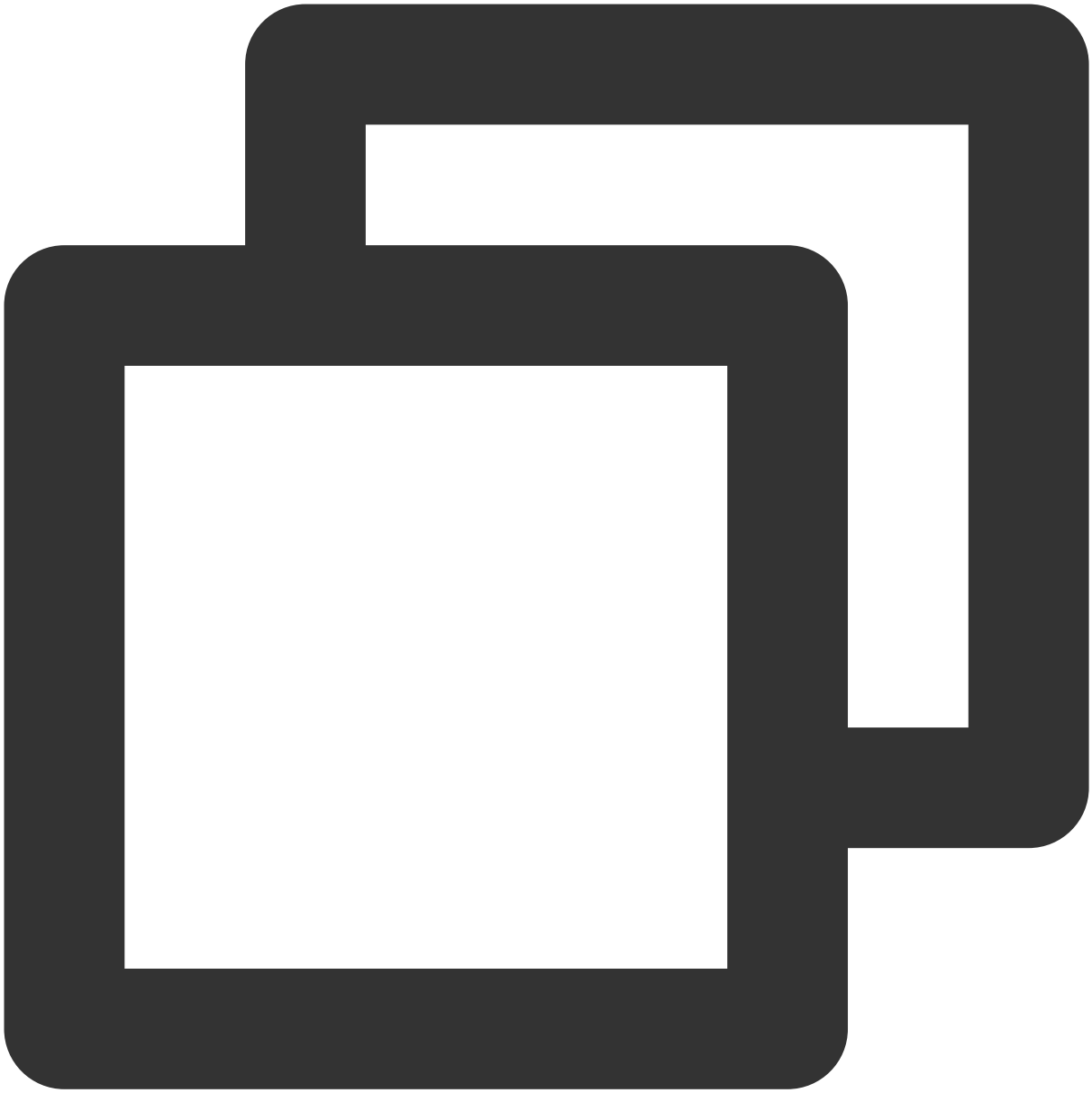
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
- (void)accept:(TUICallSucc)succ fail:(TUICallFail)fail;
```

reject

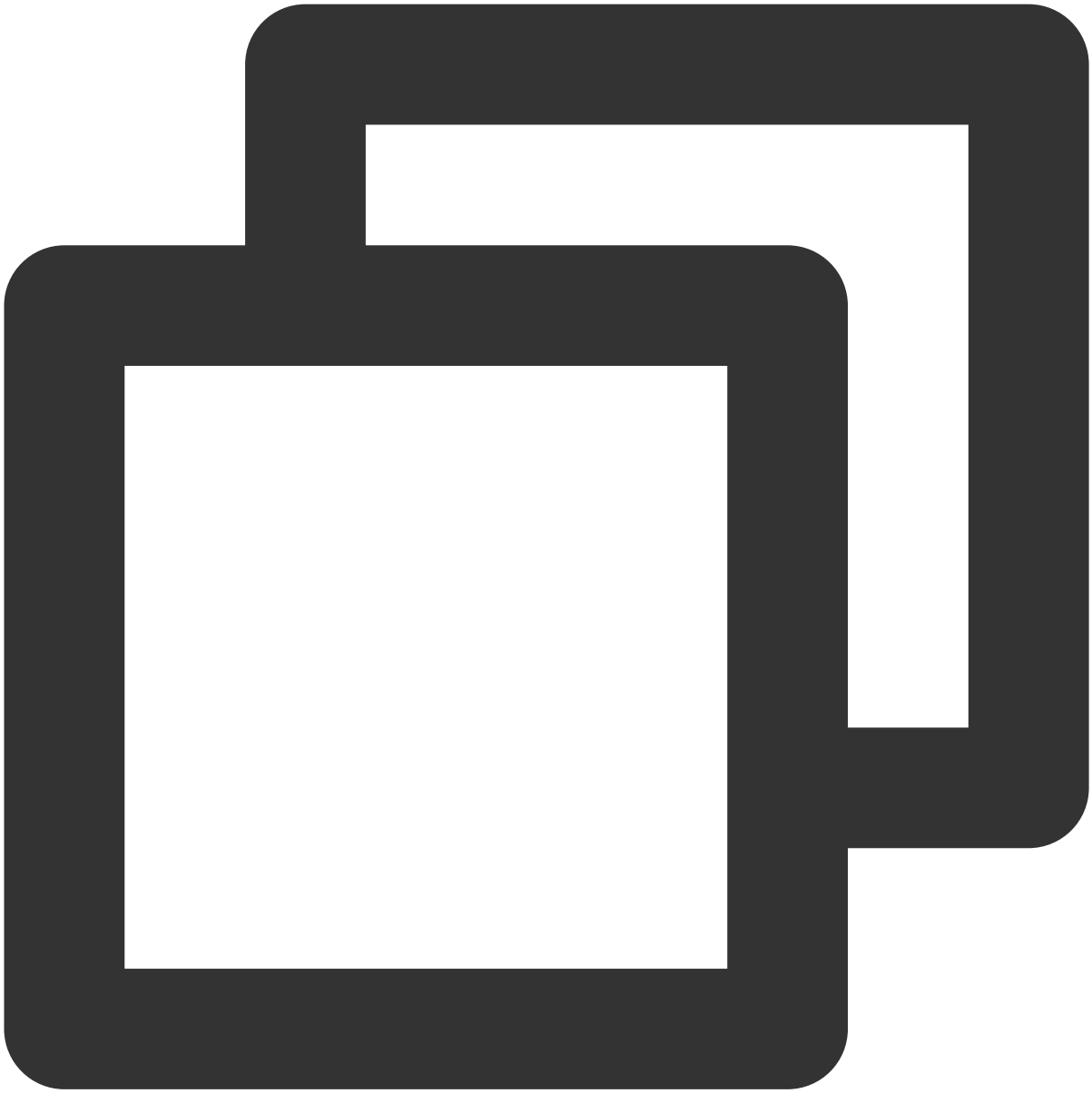
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



```
- (void)reject:(TUICallSucc)succ fail:(TUICallFail)fail;
```

ignore

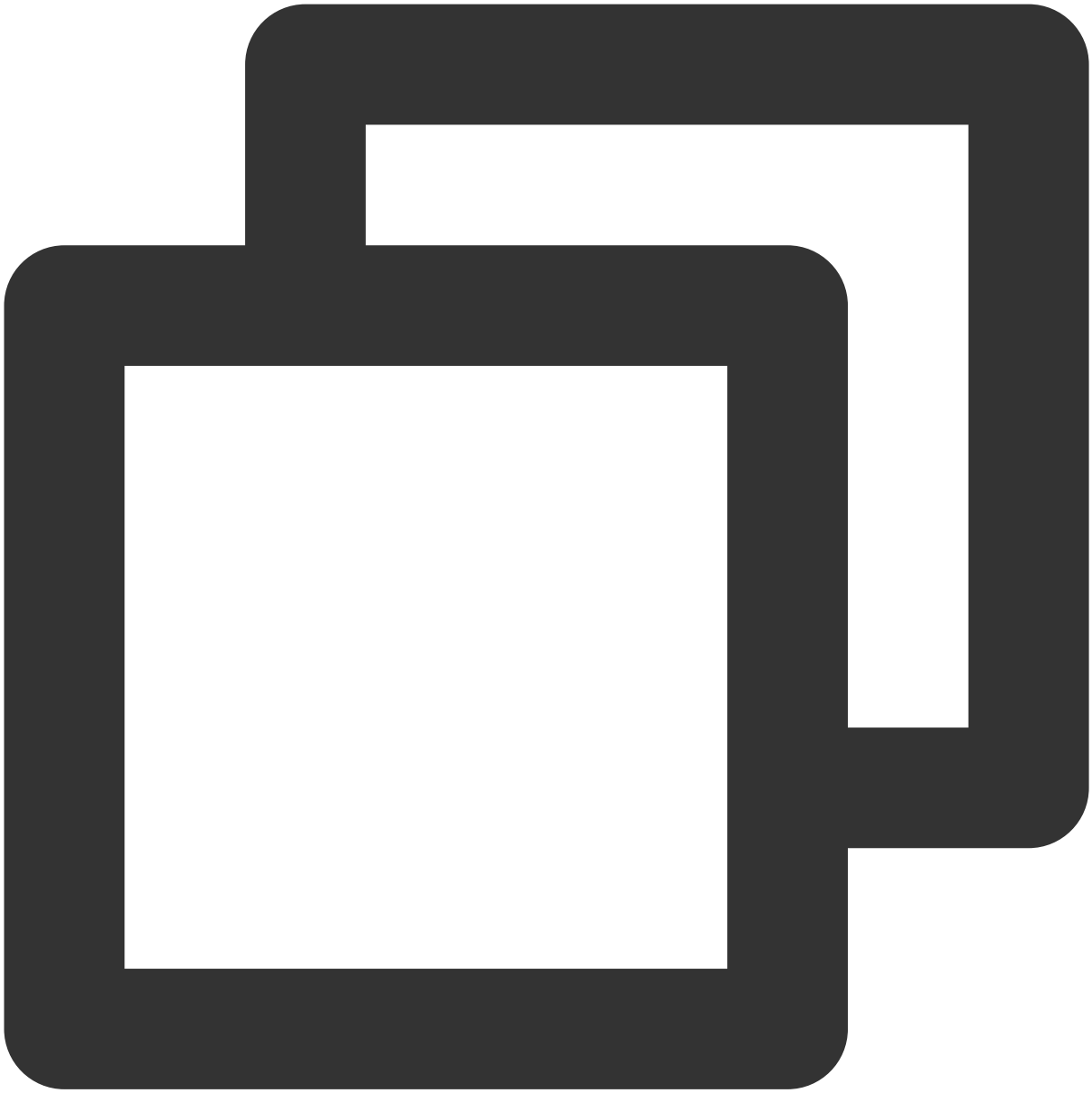
忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
- (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。

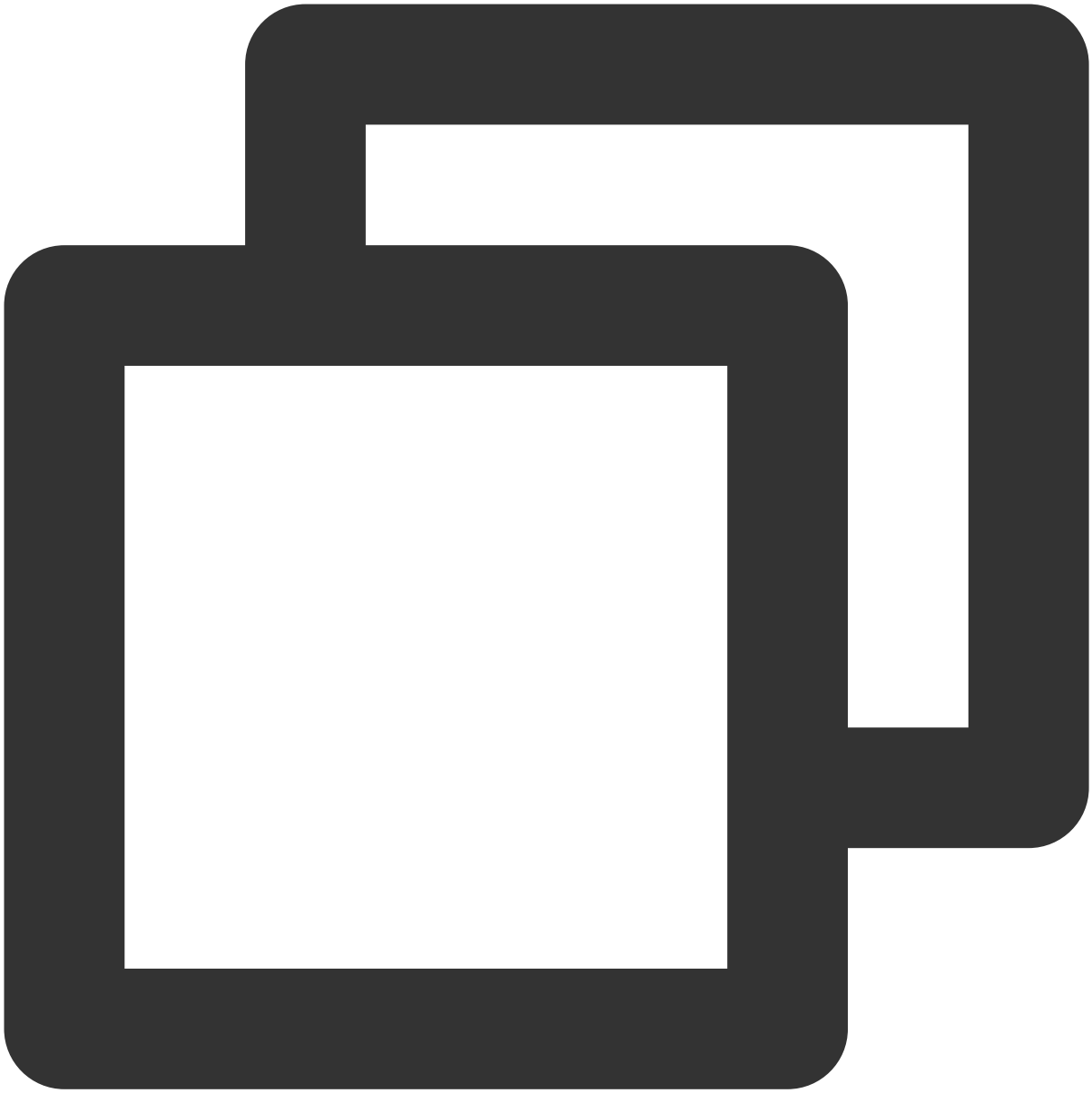


```
- (void)hangup:(TUICallSucc)succ fail:(TUICallFail)fail;
```

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。



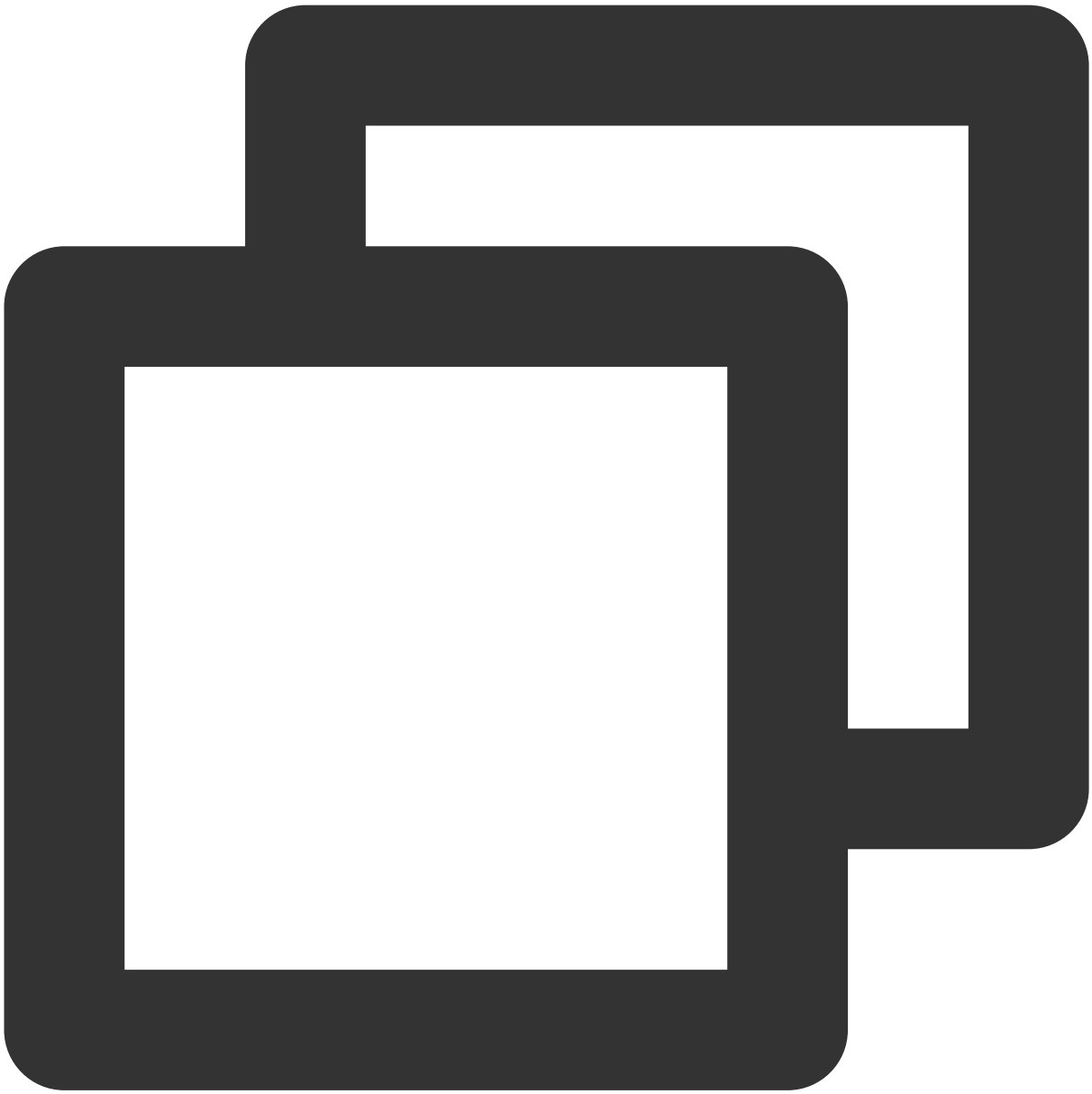
```
- (void)inviteUser:(NSArray<NSString *> *)userIdList params:(TUICallParams *)params
```

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。



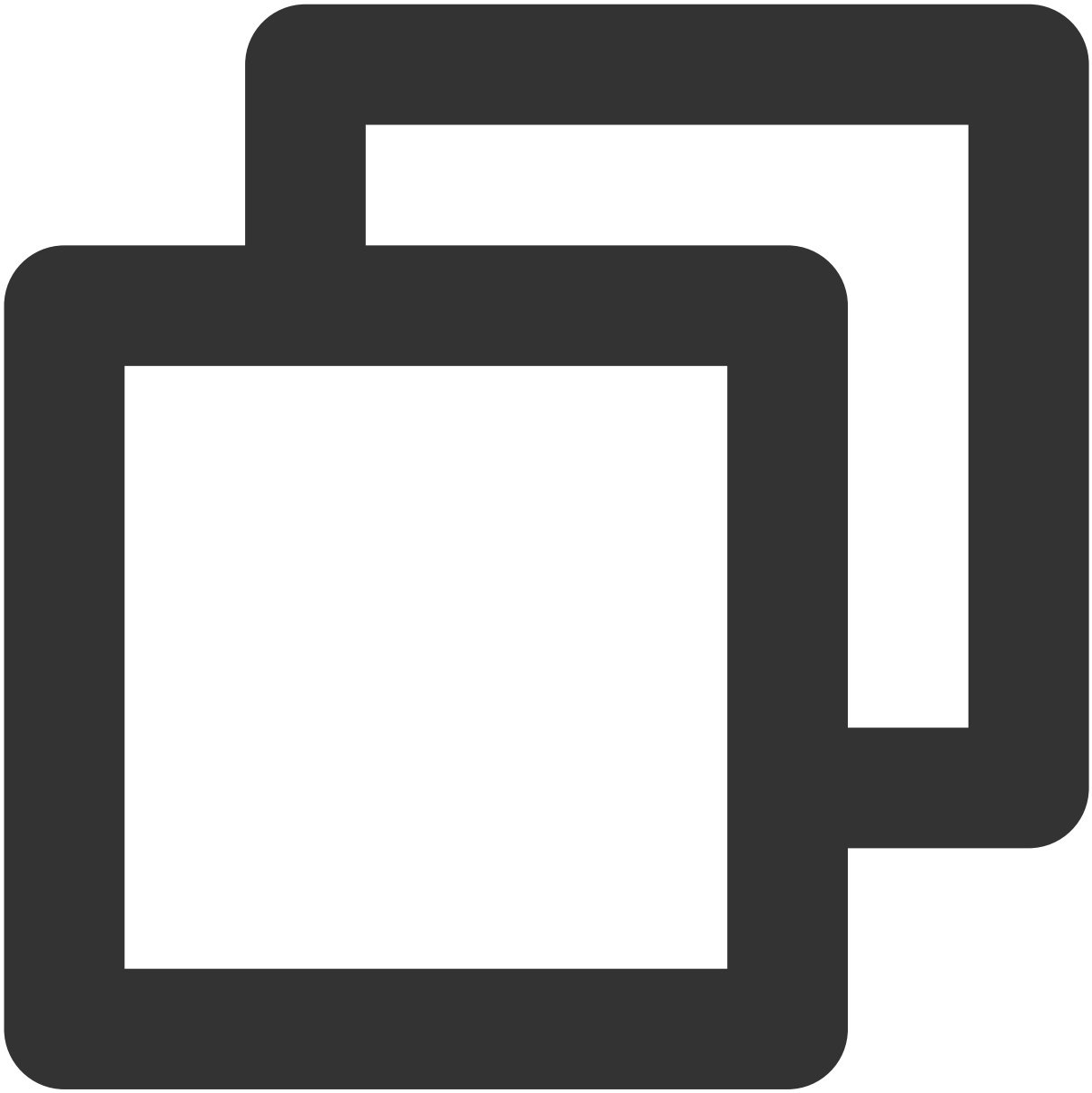
```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaTy
```

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。

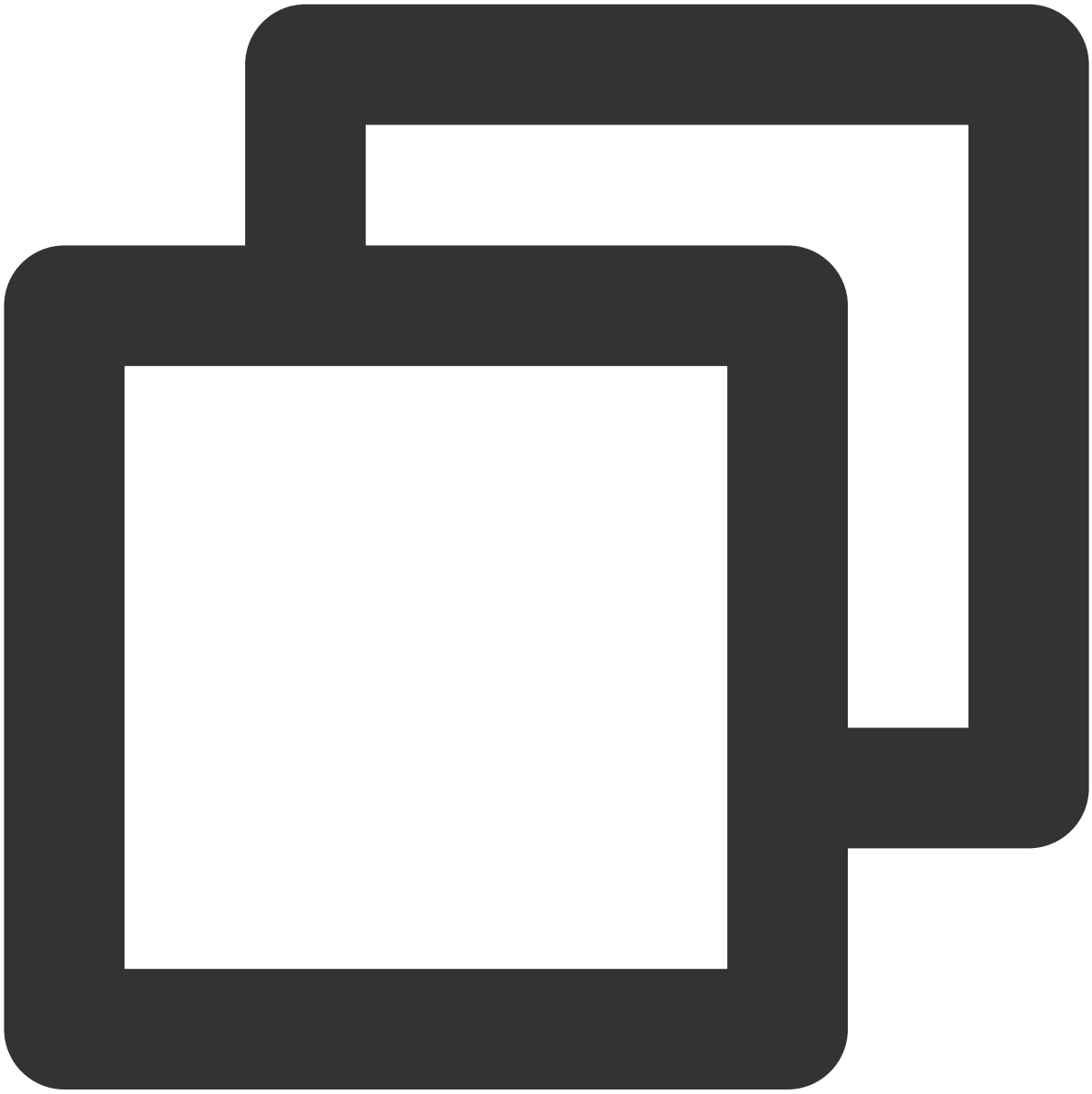


```
- (void)switchCallMediaType:(TUICallMediaType)newType;
```

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。

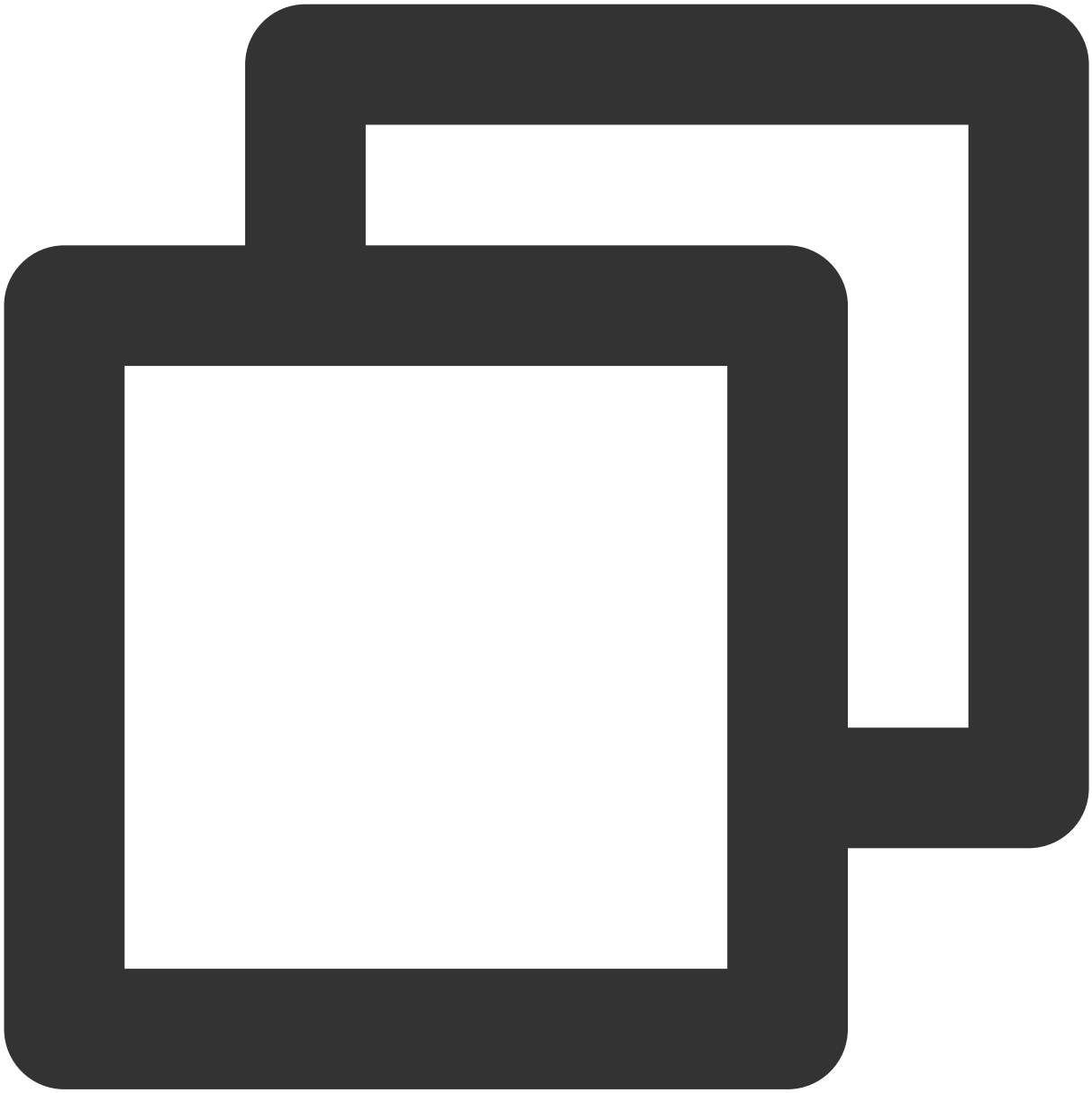


```
- (void)startRemoteView:(NSString *)userId videoView:(TUIVideoView *)videoView onPl
```

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。

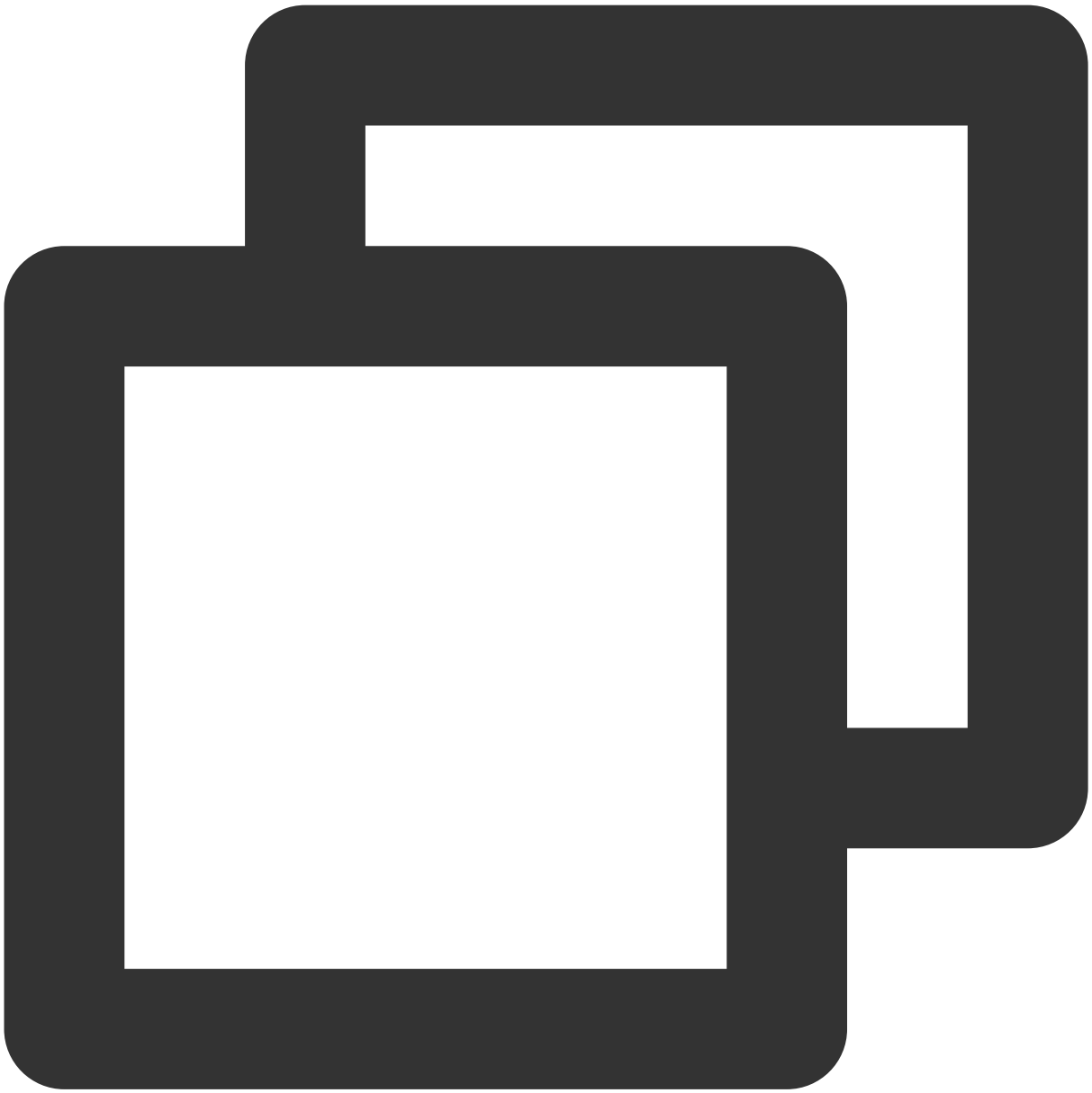


```
- (void)stopRemoteView:(NSString *)userId;
```

参数	类型	含义
userId	NSString	目标用户的 userId

openCamera

开启摄像头。

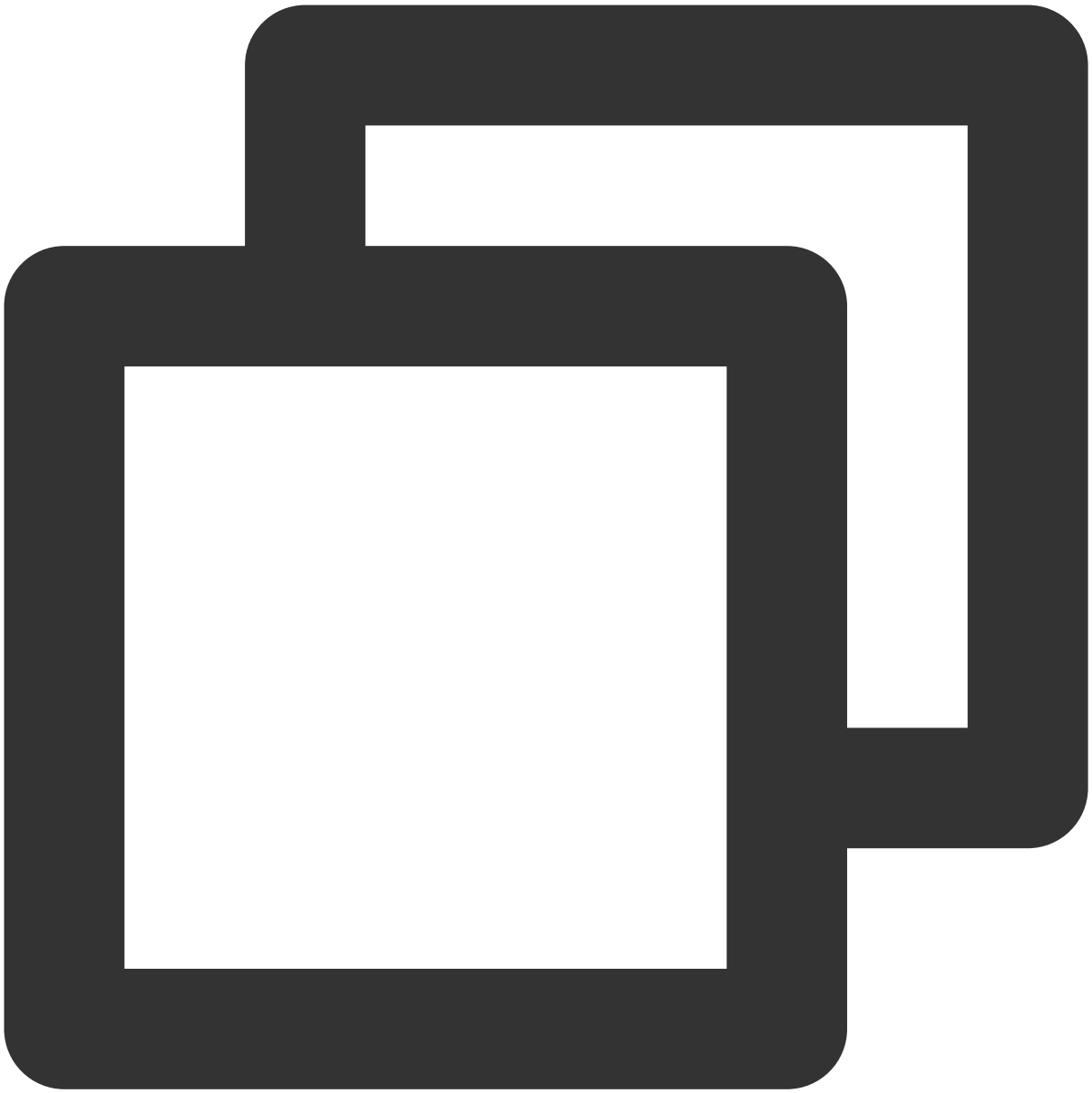


```
- (void)openCamera:(TUICamera)camera videoView:(TUIVideoView *)videoView succ:(TUIC
```

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

closeCamera

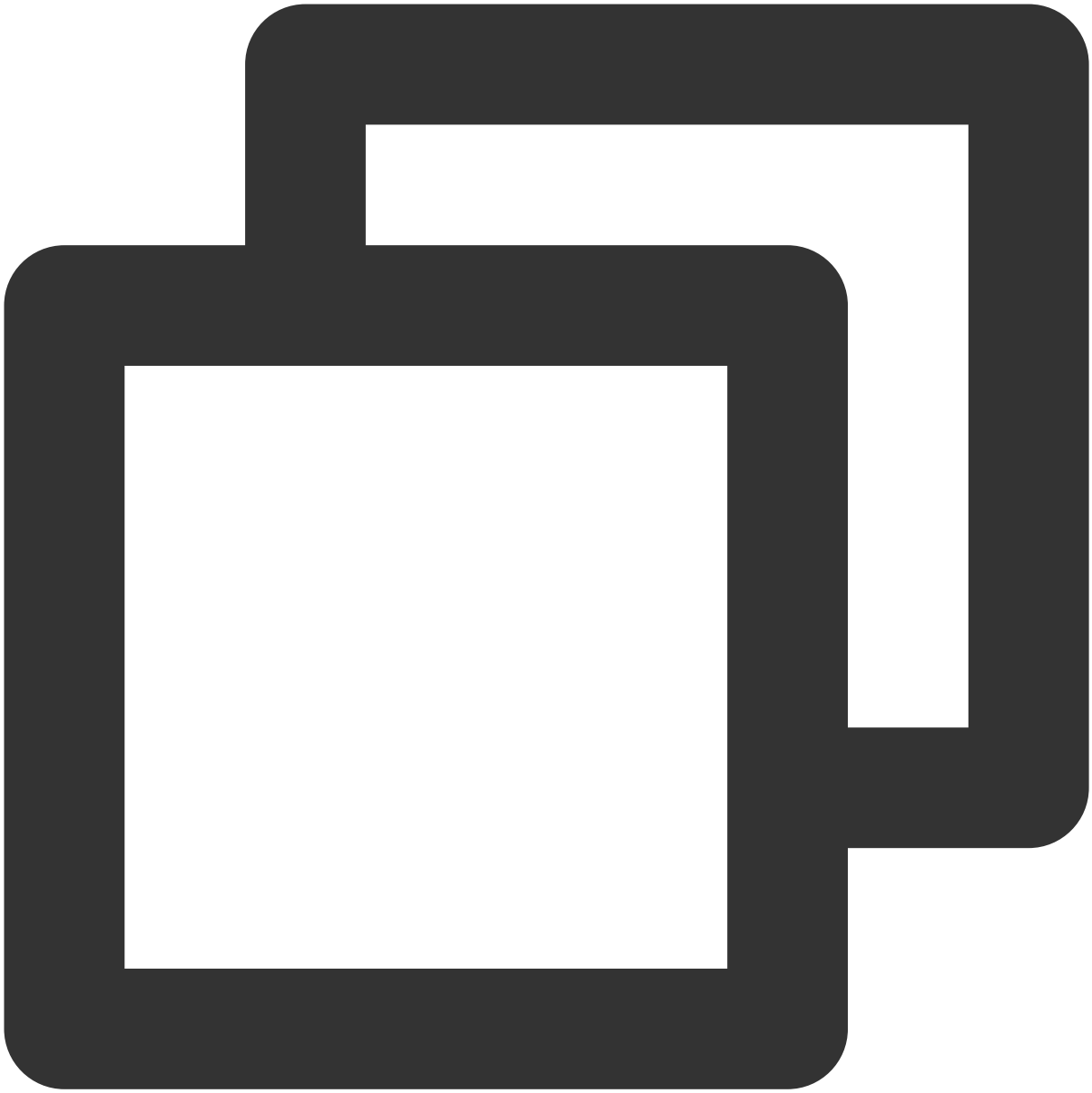
关闭摄像头。



```
- (void)closeCamera;
```

switchCamera

切换前后摄像头。

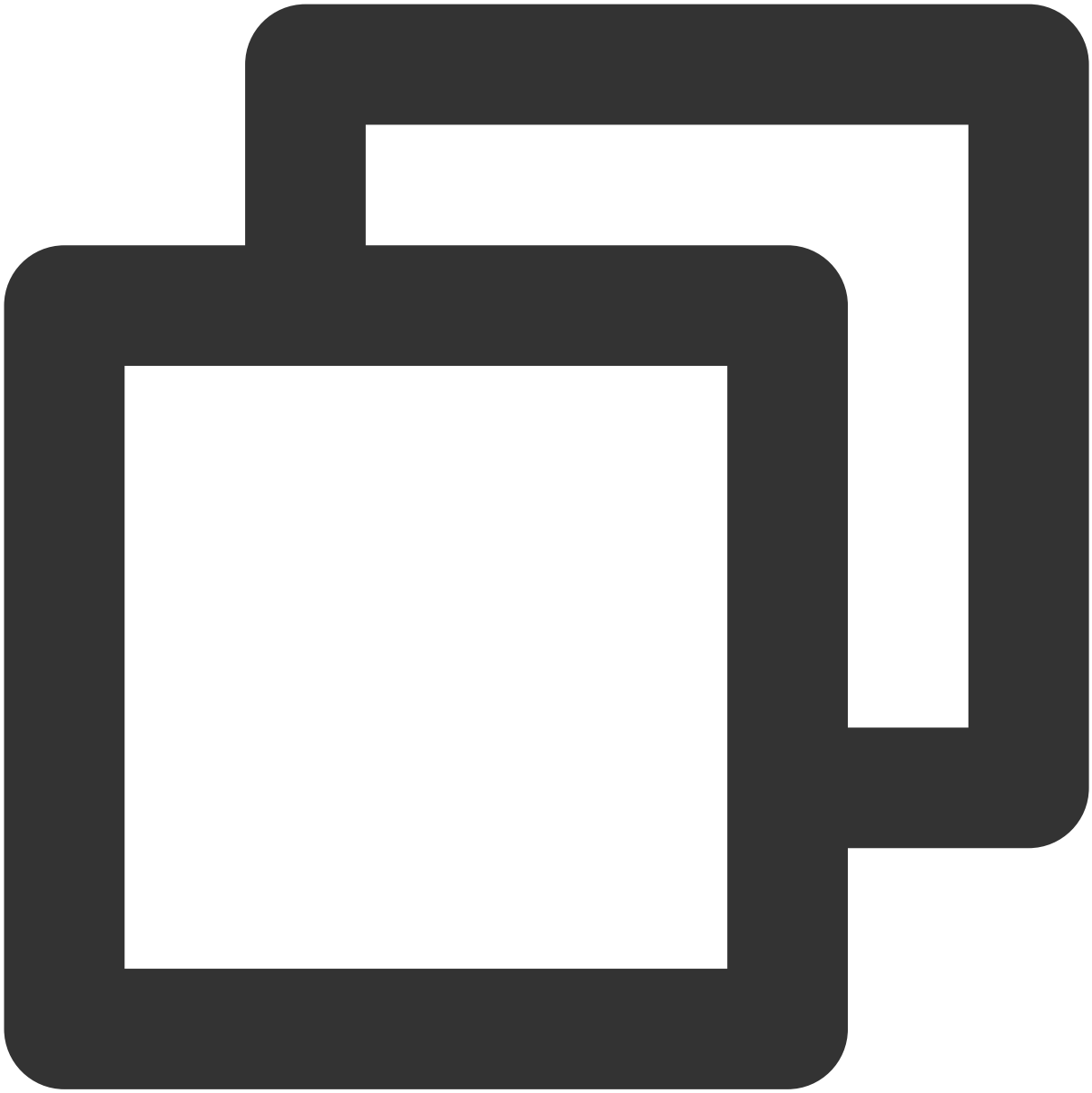


```
- (void)switchCamera:(TUICamera)camera;
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头

openMicrophone

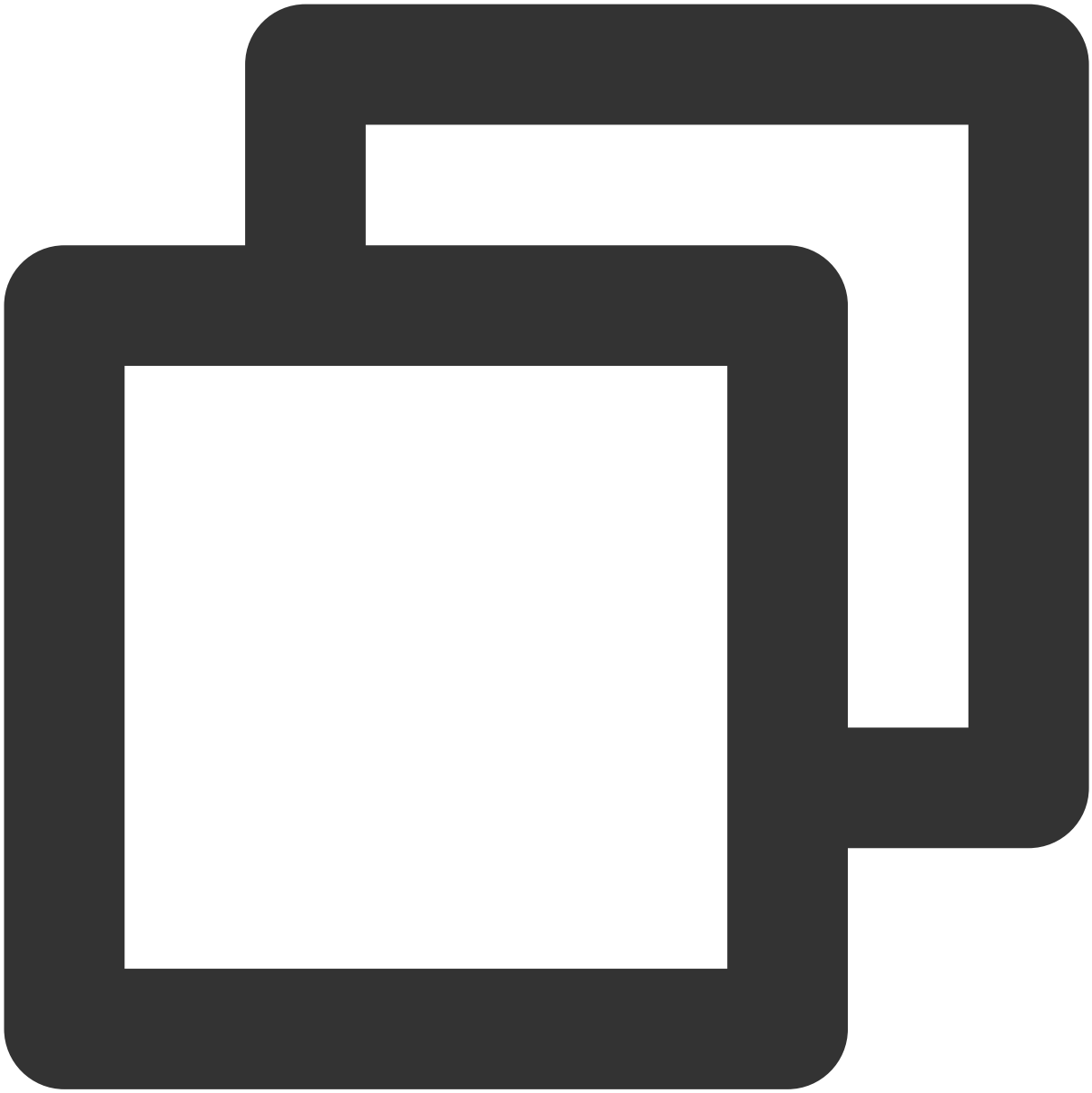
打开麦克风。



```
- (void)openMicrophone:(TUICallSucc)succ fail:(TUICallFail)fail;
```

closeMicrophone

关闭麦克风。

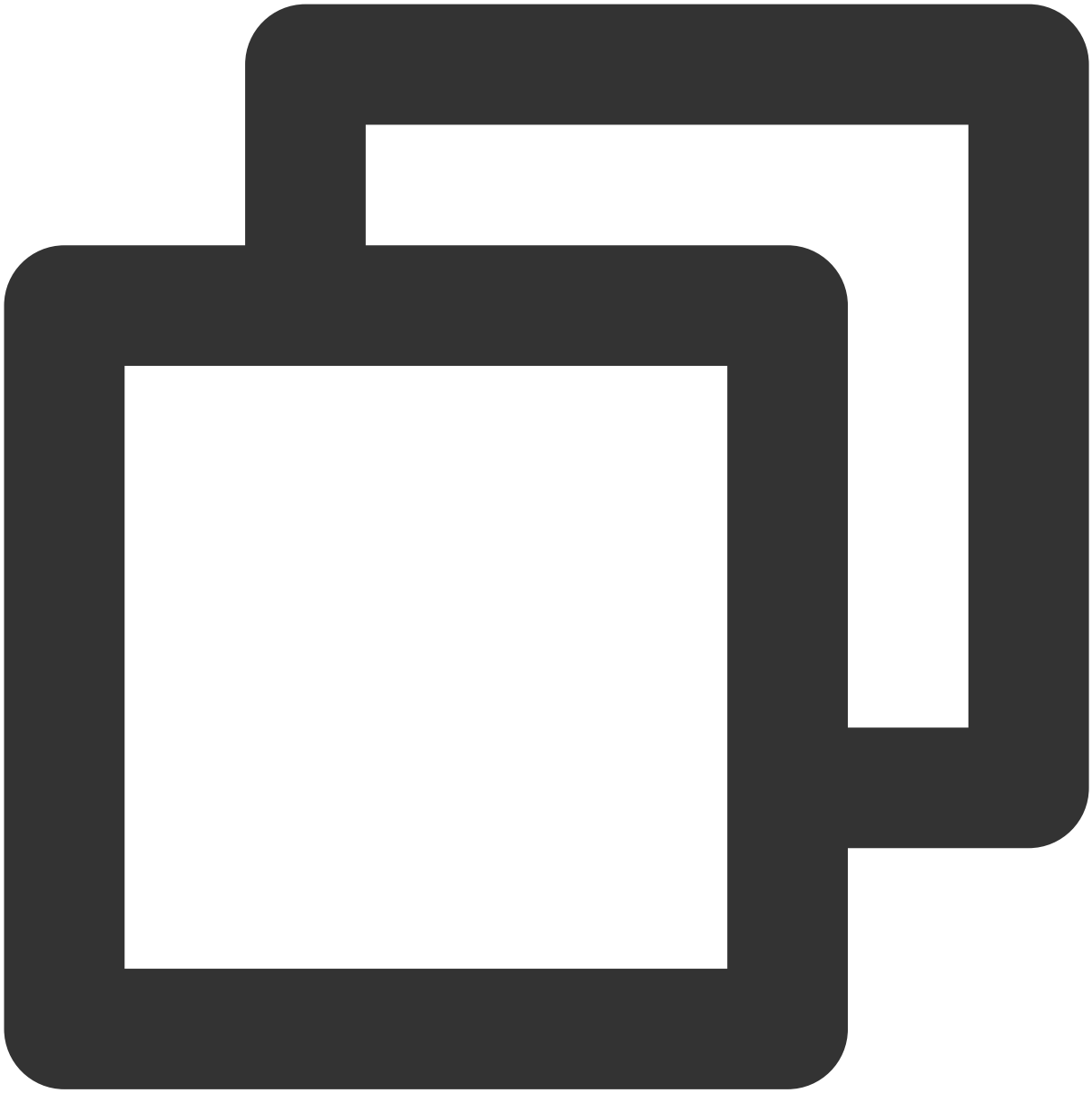


```
- (void)closeMicrophone;
```

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。



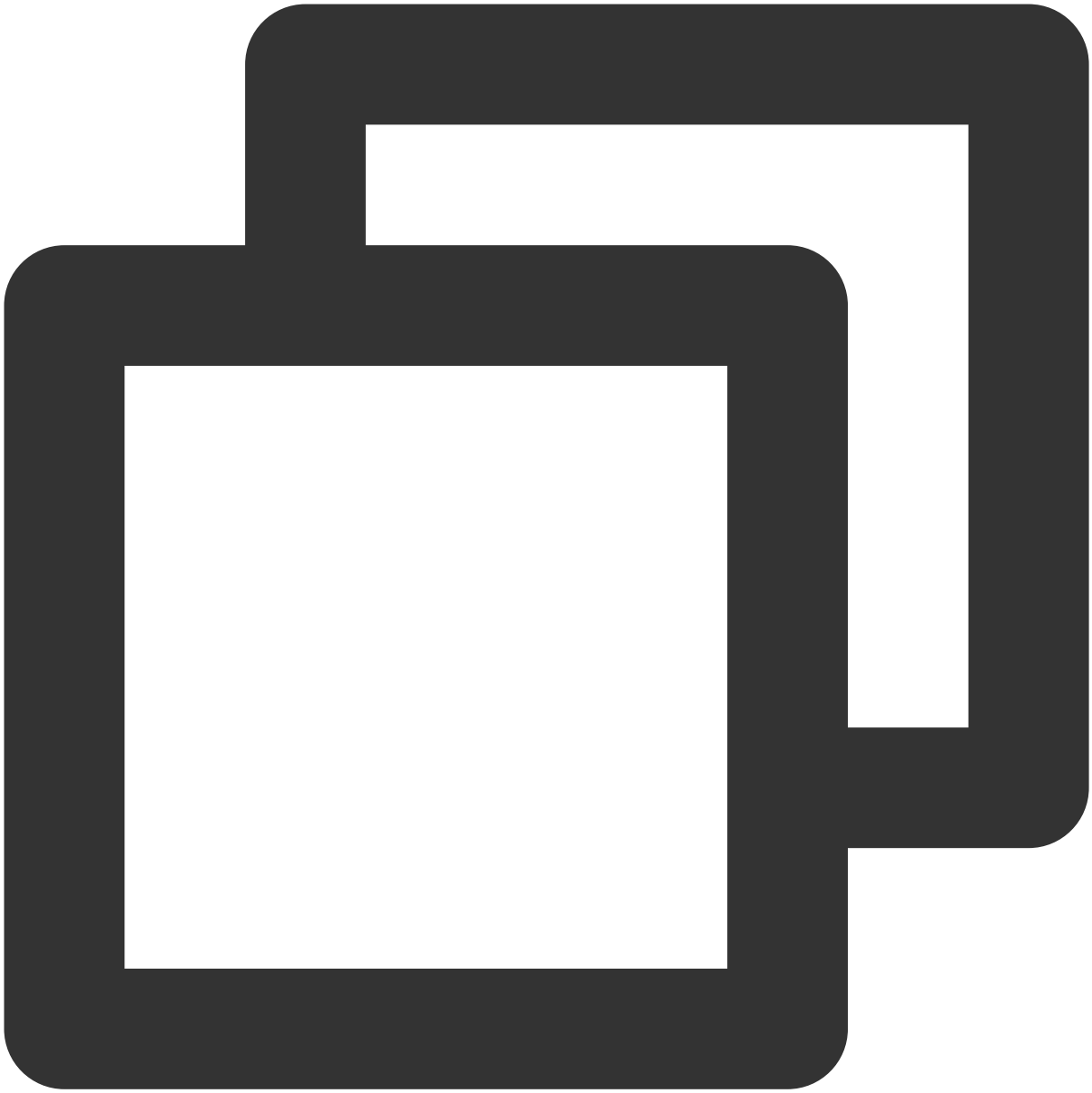
```
- (void)selectAudioPlaybackDevice:(TUIAudioPlaybackDevice)device;
```

参数	类型	含义
device	TUIAudioPlaybackDevice	听筒/扬声器

setSelfInfo

设置用户昵称、头像。

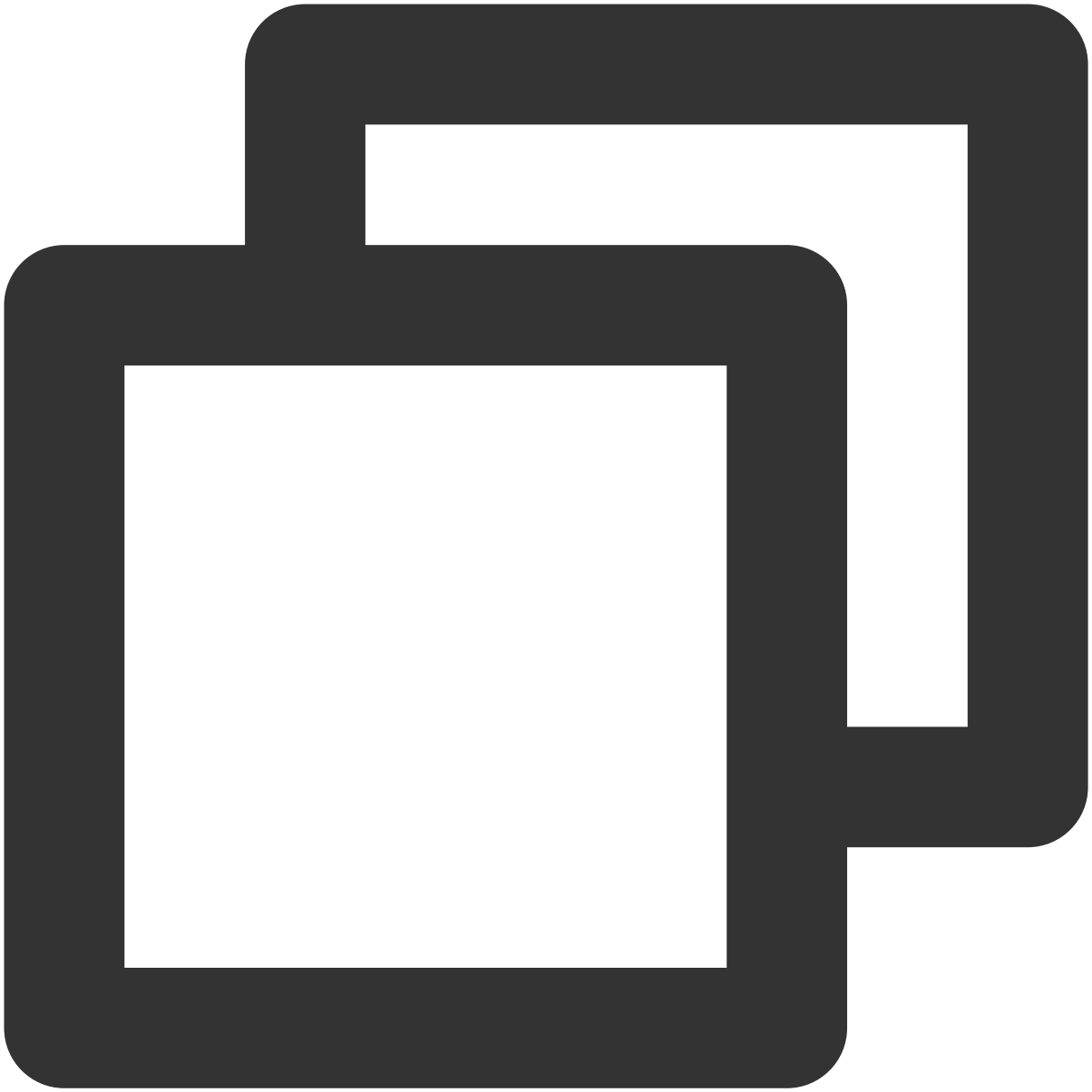
用户昵称不能超过500字节，用户头像必须是 URL 格式。



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

enableMultiDeviceAbility

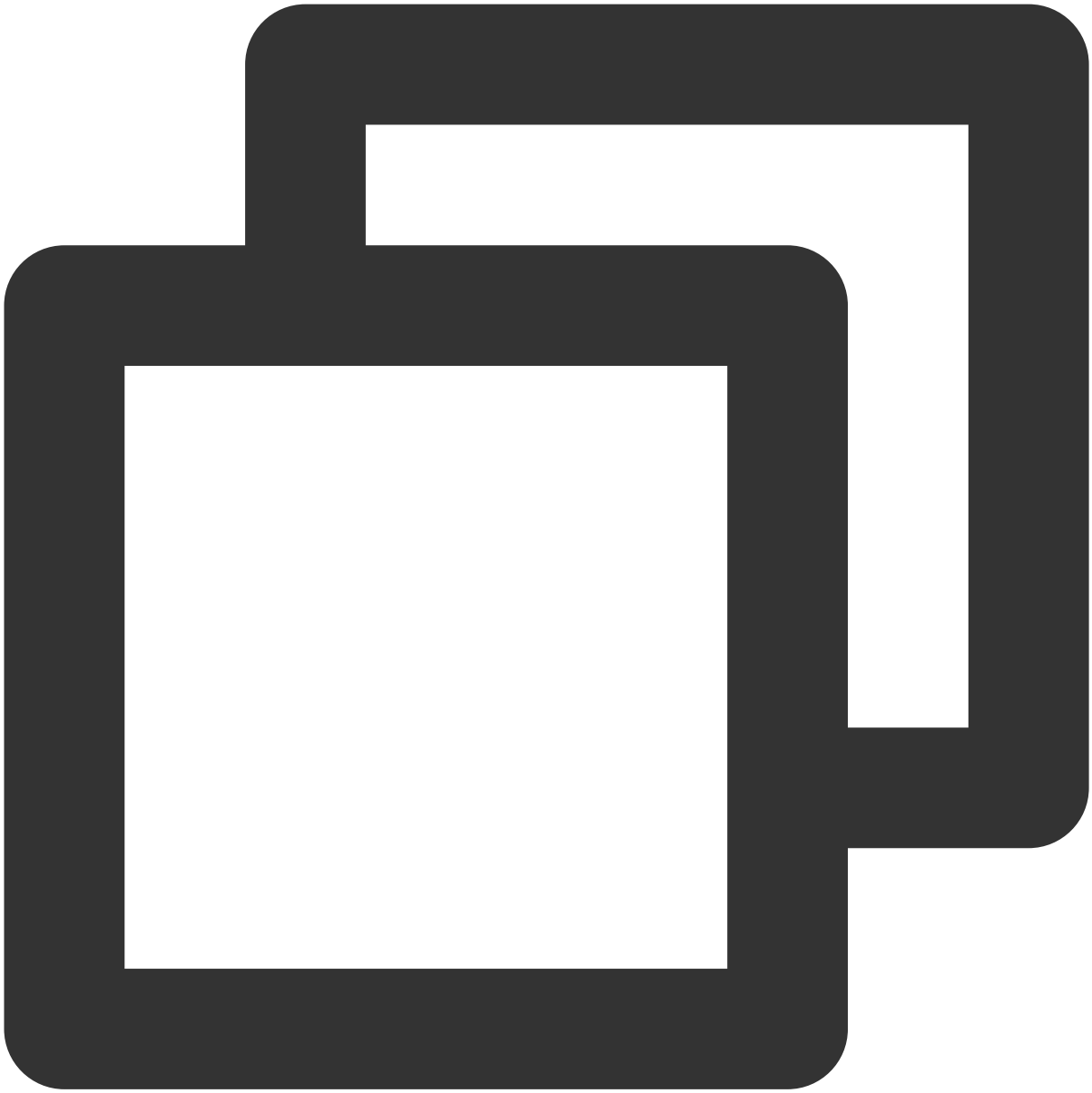
开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐支持）。



```
- (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
```

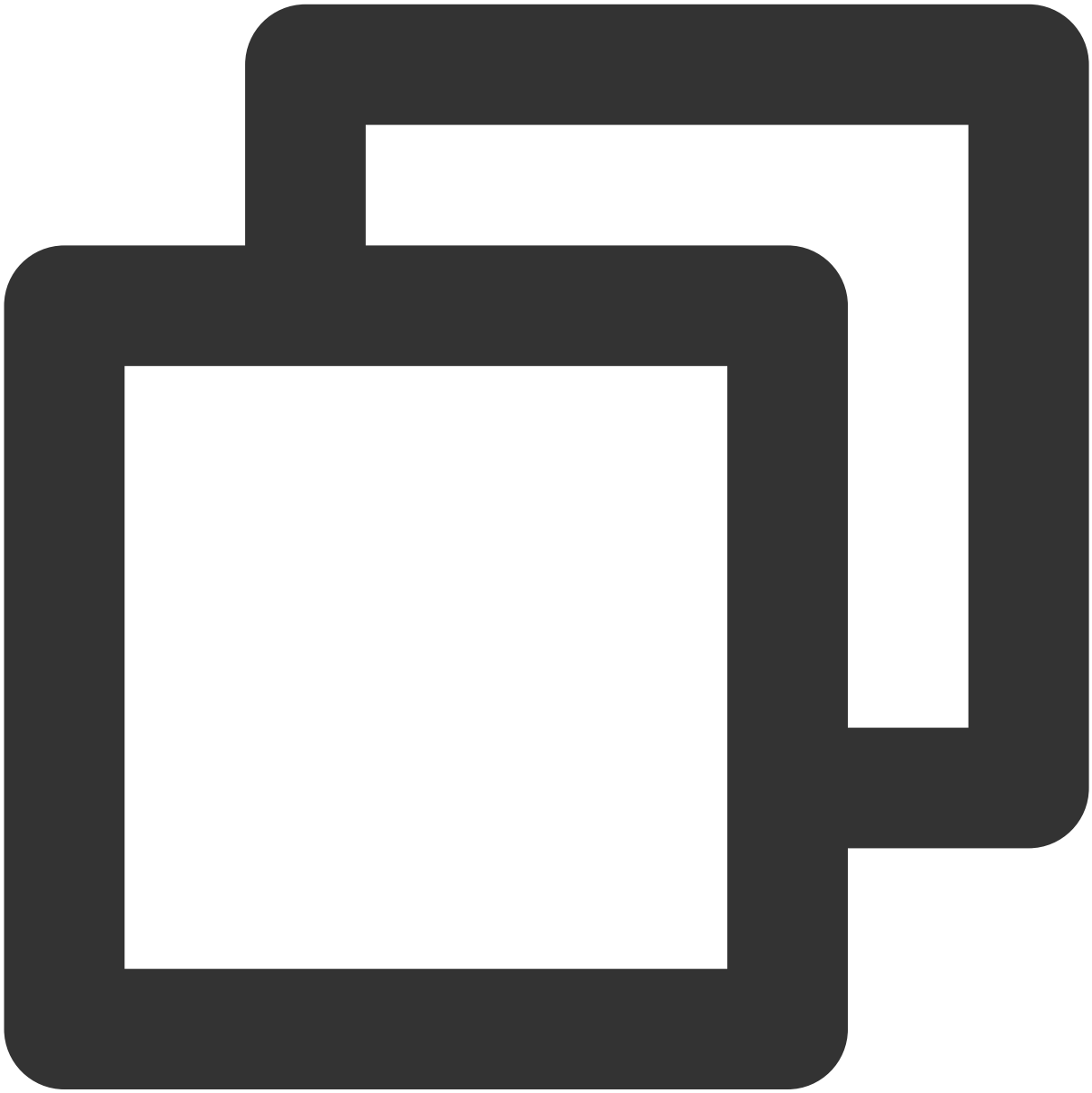
参数如下表所示：

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
```

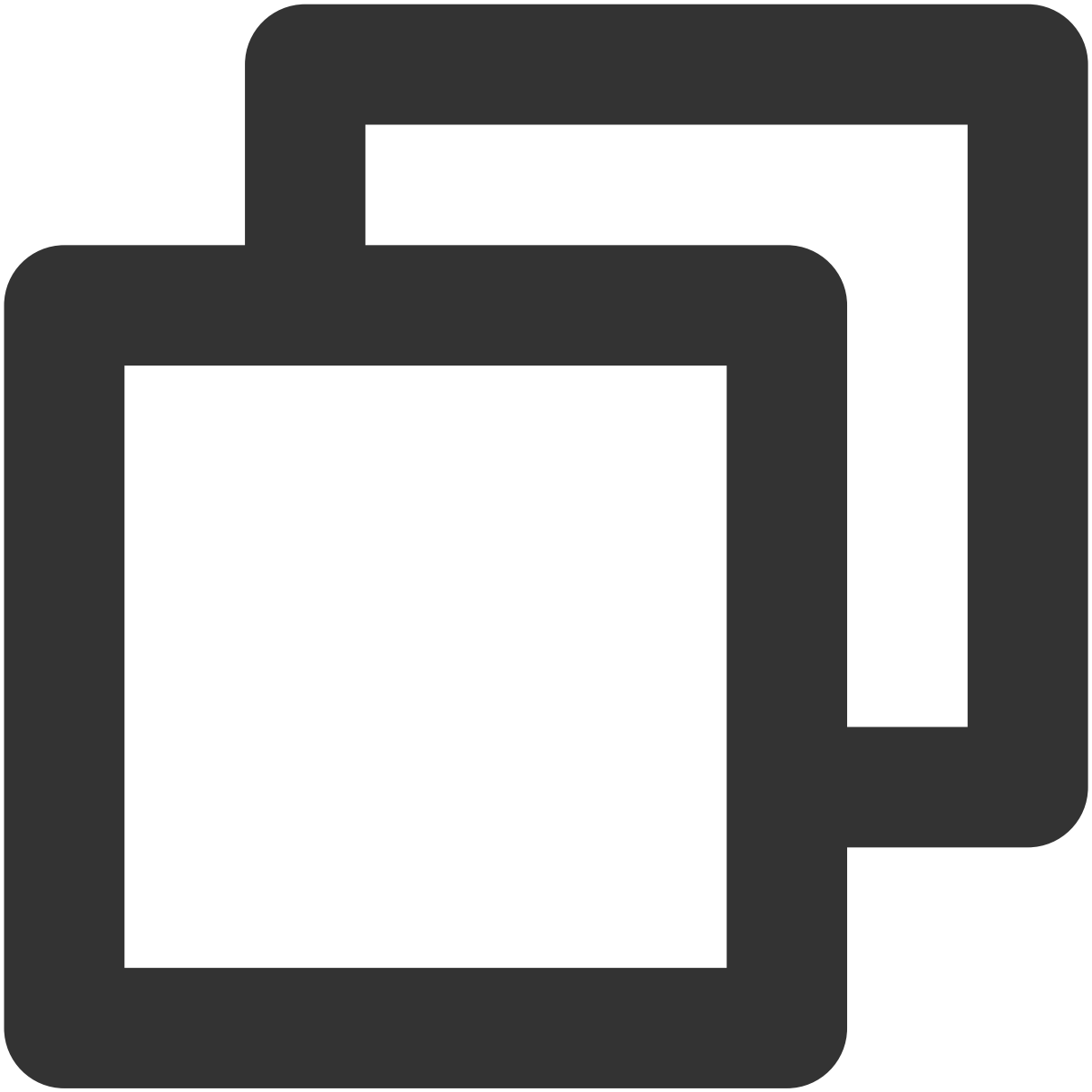
参数如下表所示：

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。

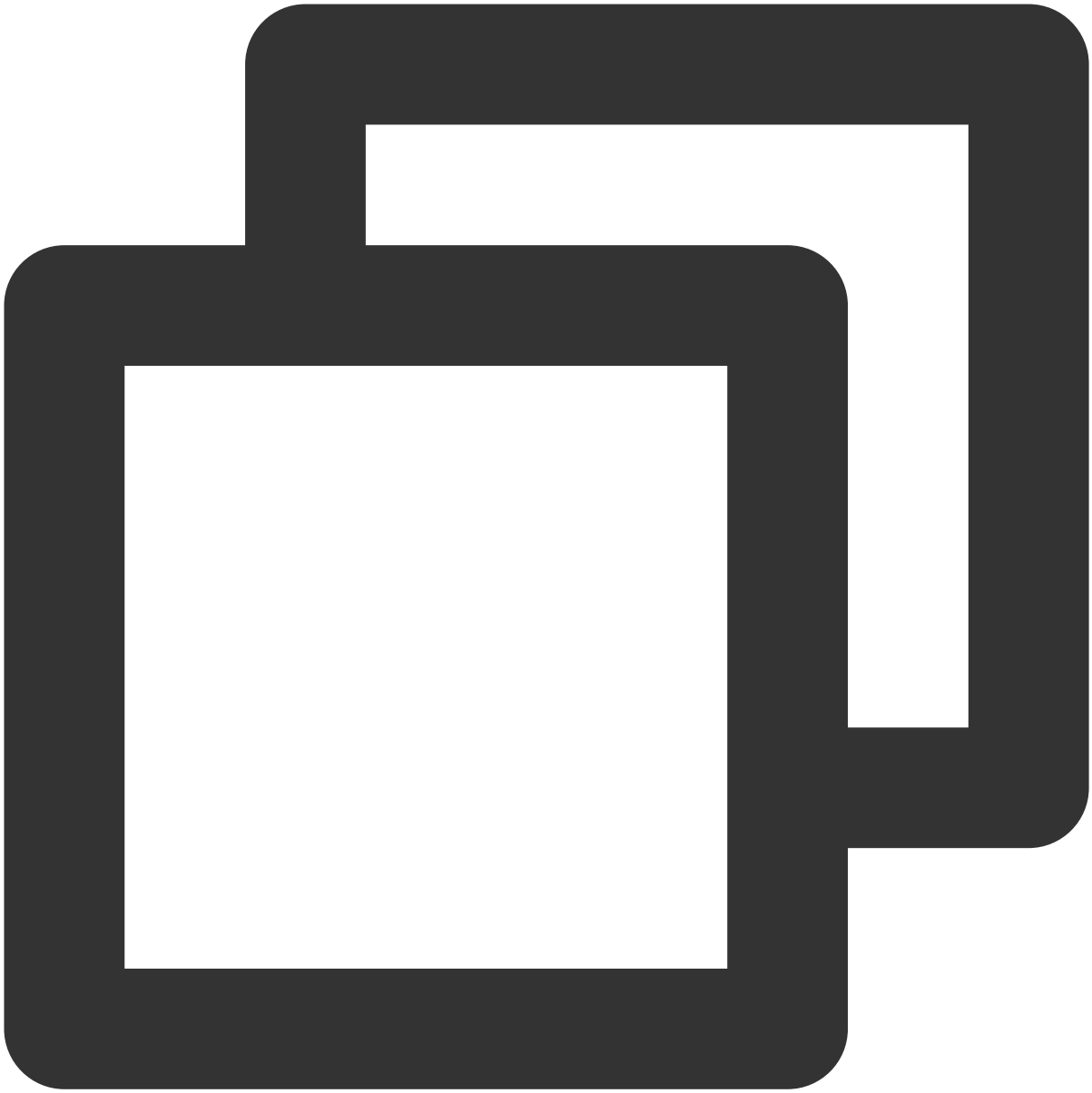
用于接入第三方美颜等。



```
- (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

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	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

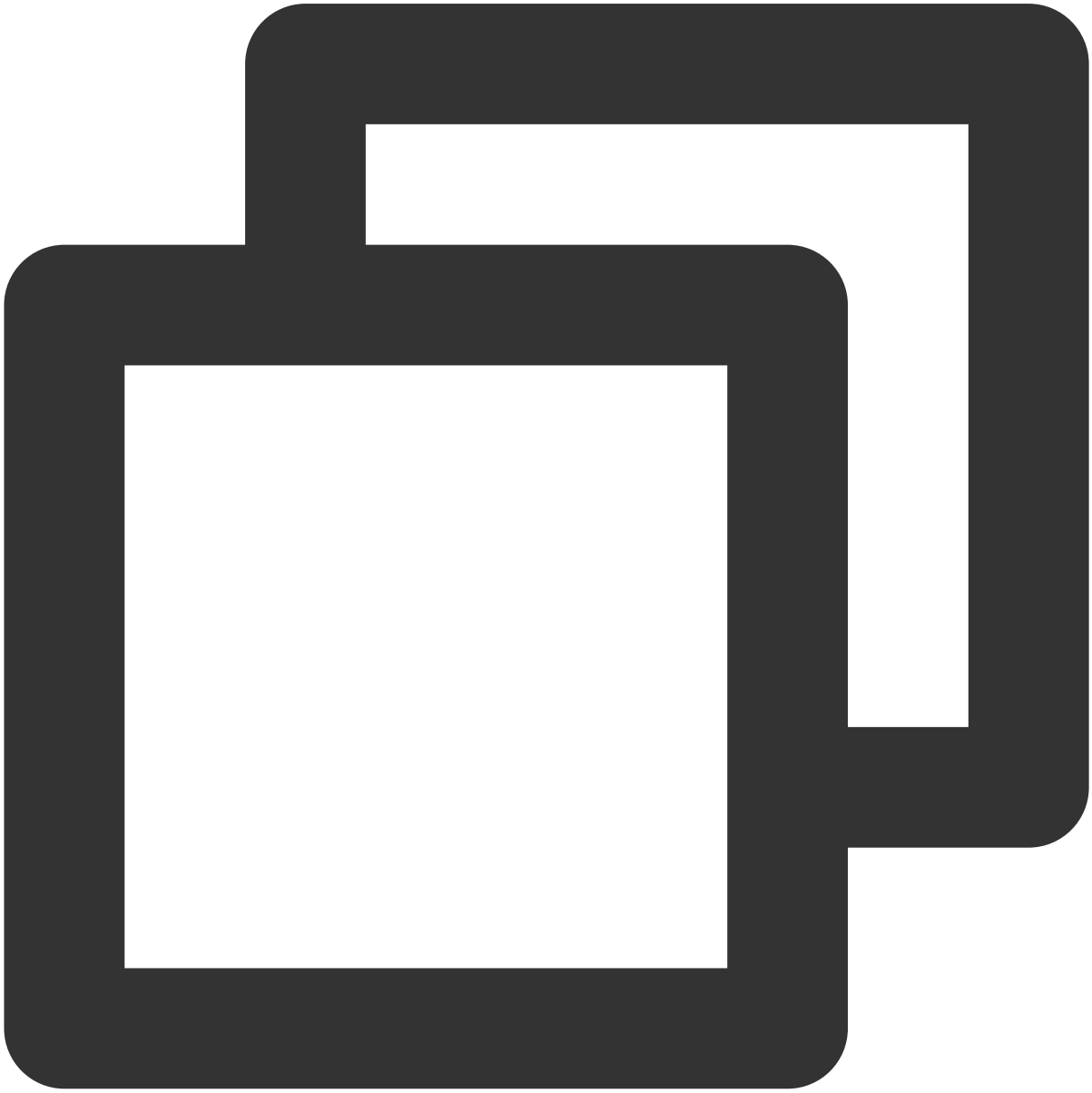
回调事件详情

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。



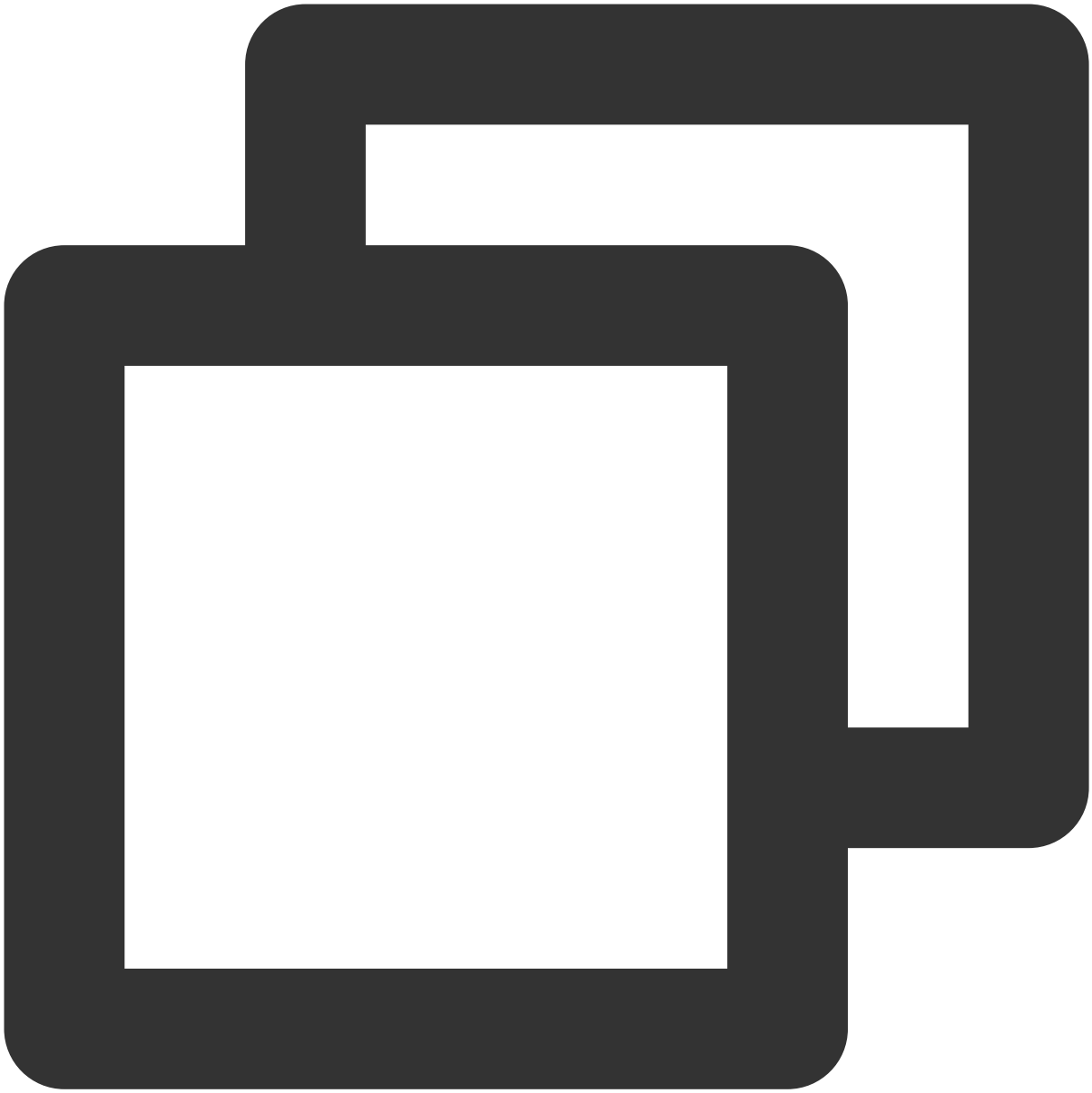
```
- (void)onError:(int)code message:(NSString * _Nullable)message;
```

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。

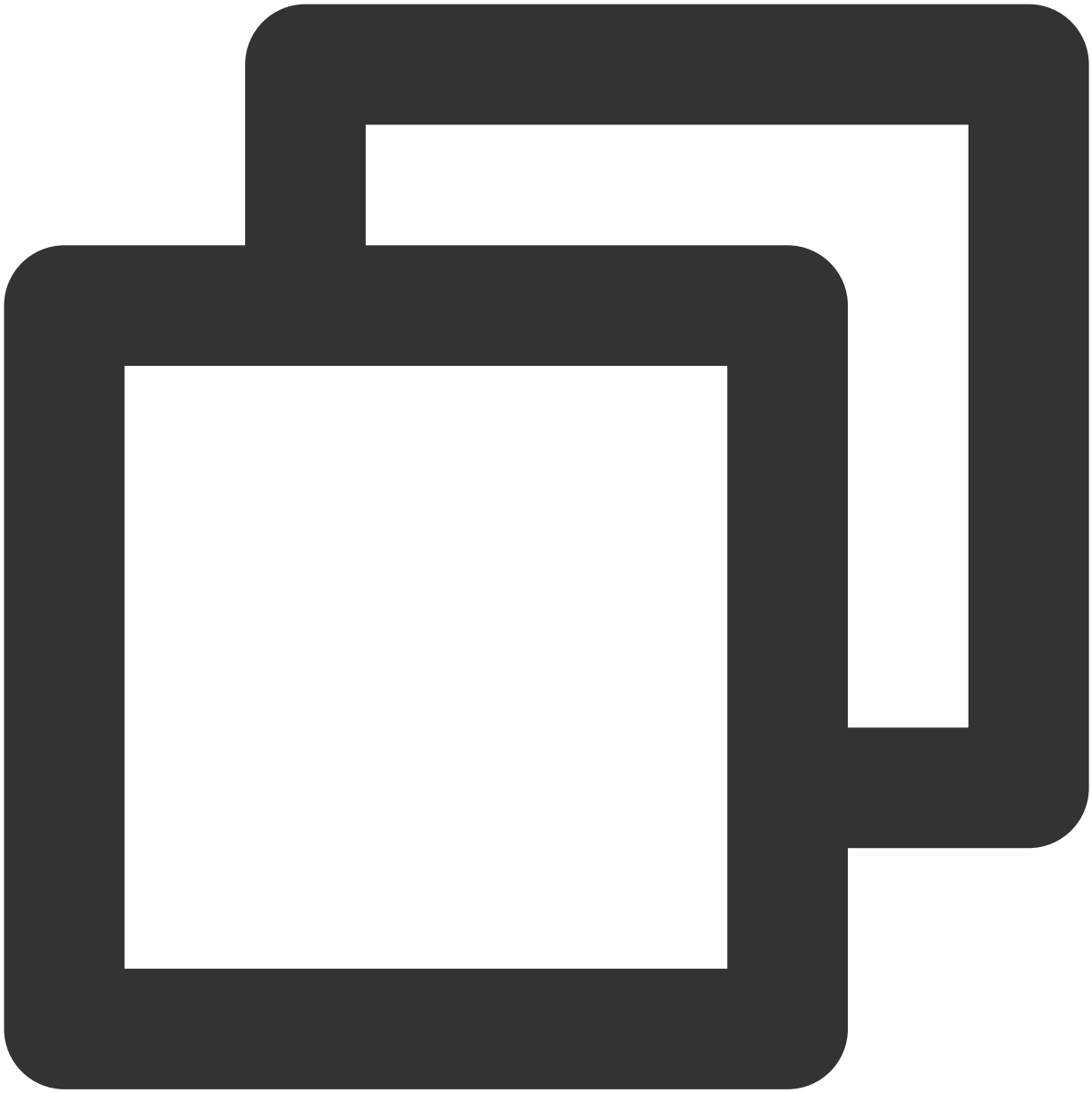
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

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）



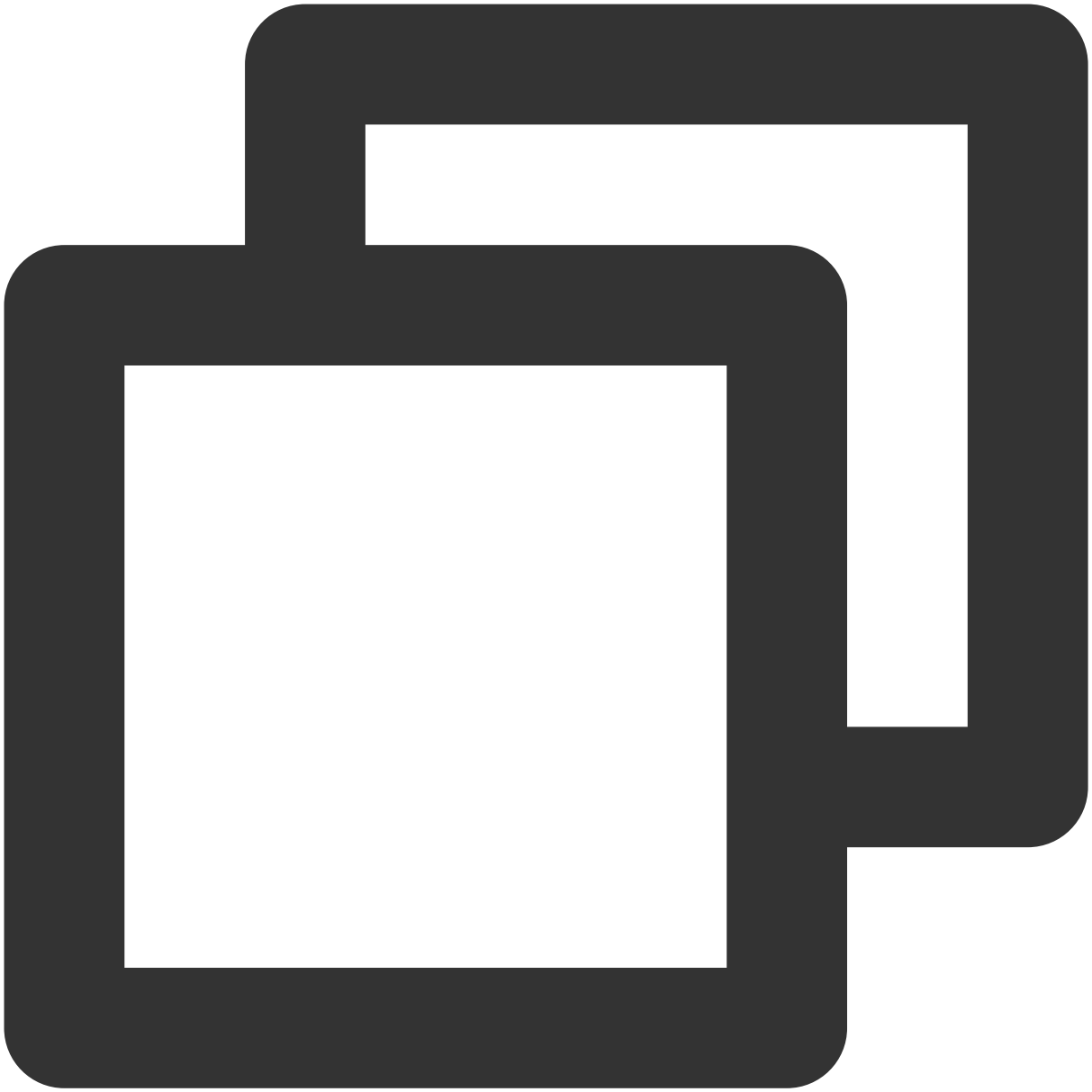
```
- (void)onCallCancelled:(NSString *)callerId;
```

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。



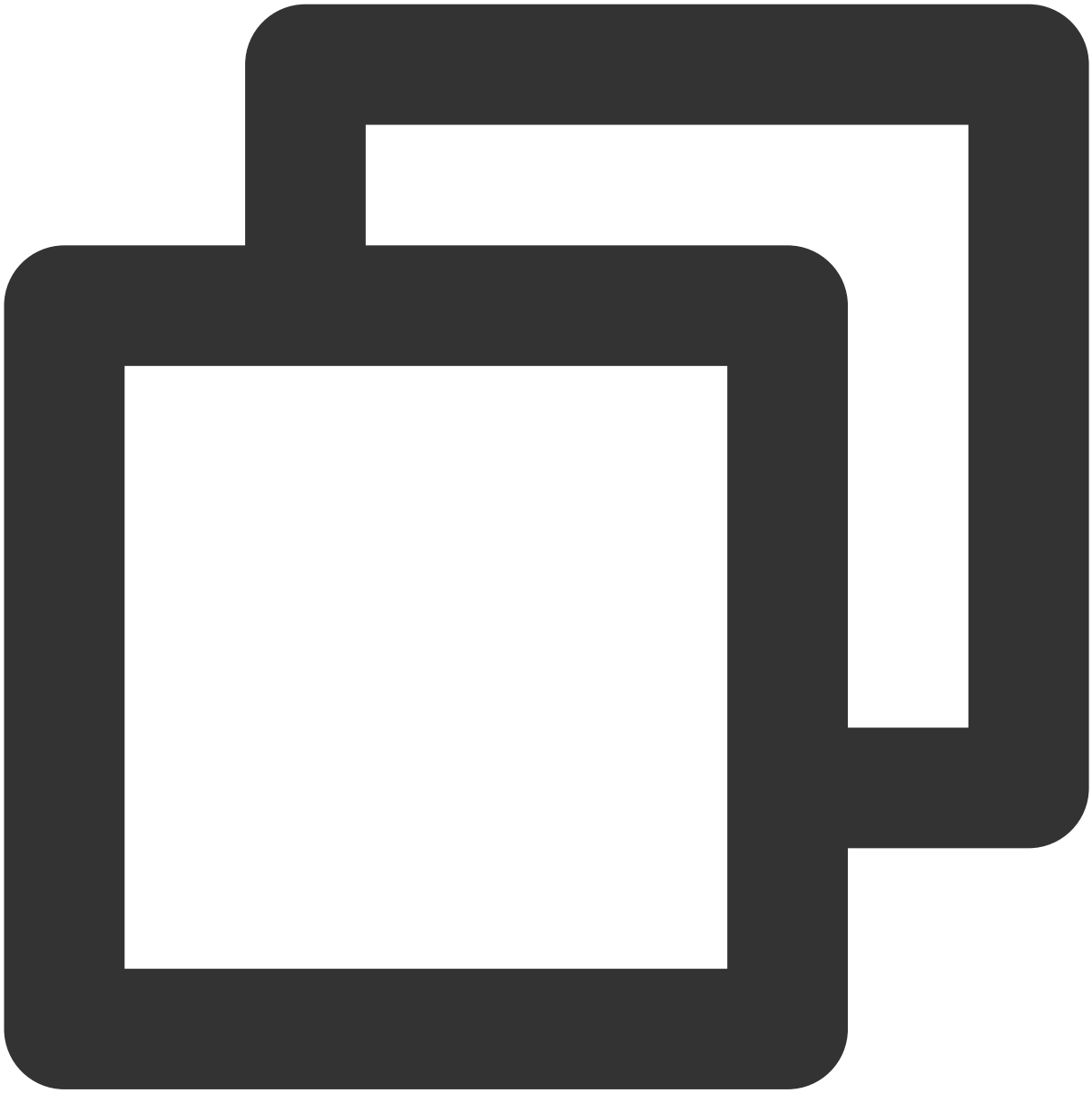
```
- (void)onCallBegin:(TUIRoomId *)roomId callMediaType:(TUICallMediaType)callMediaTy
```

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

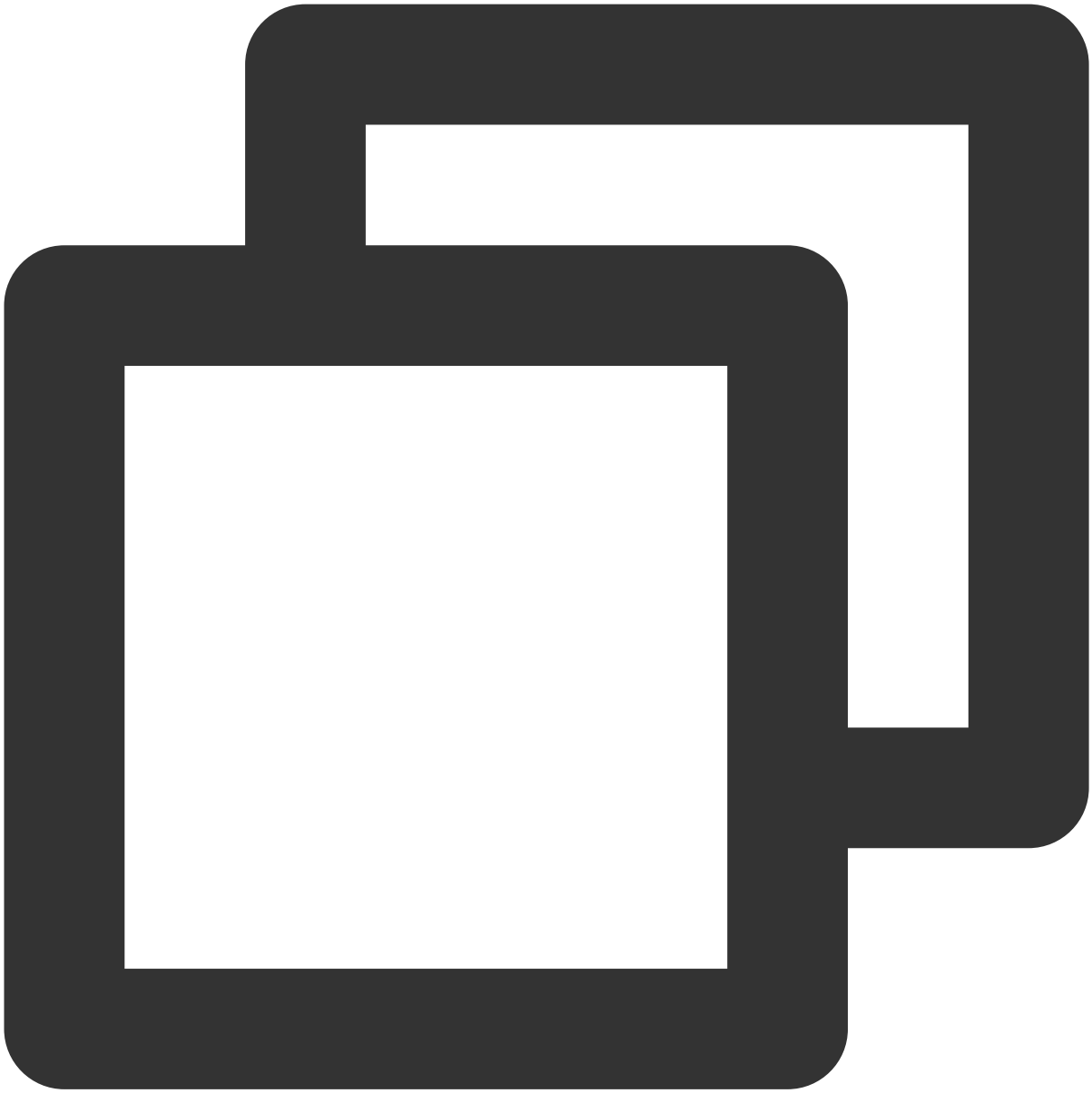
callMediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
callRole	TUICallRole	角色，枚举类型：主叫、被叫
totalTime	float	此次通话的时长

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。



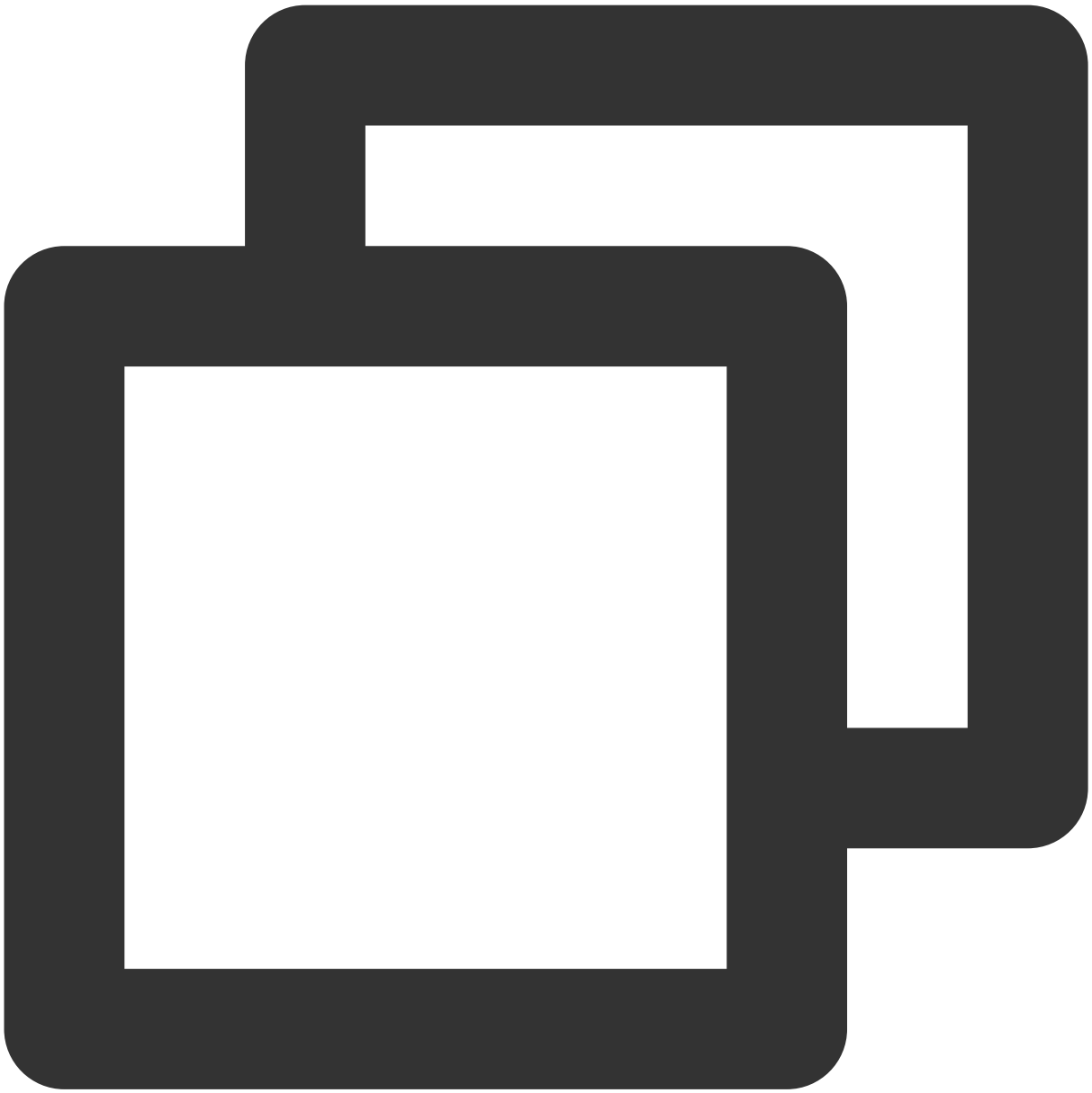
```
- (void)onCallMediaTypeChanged:(TUICallMediaType)oldCallMediaType newCallMediaType:
```

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。



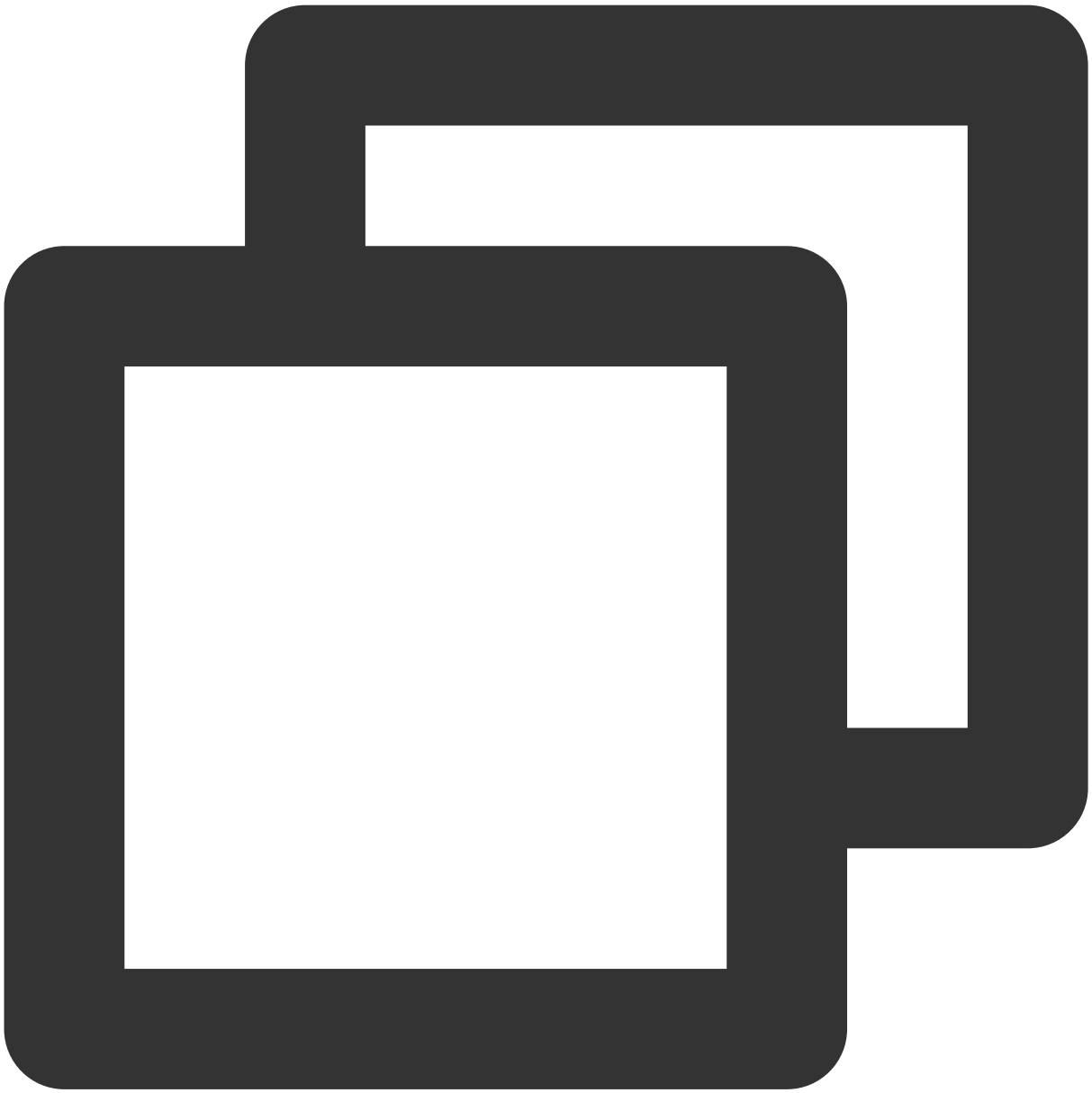
```
- (void)onUserReject:(NSString *)userId;
```

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。



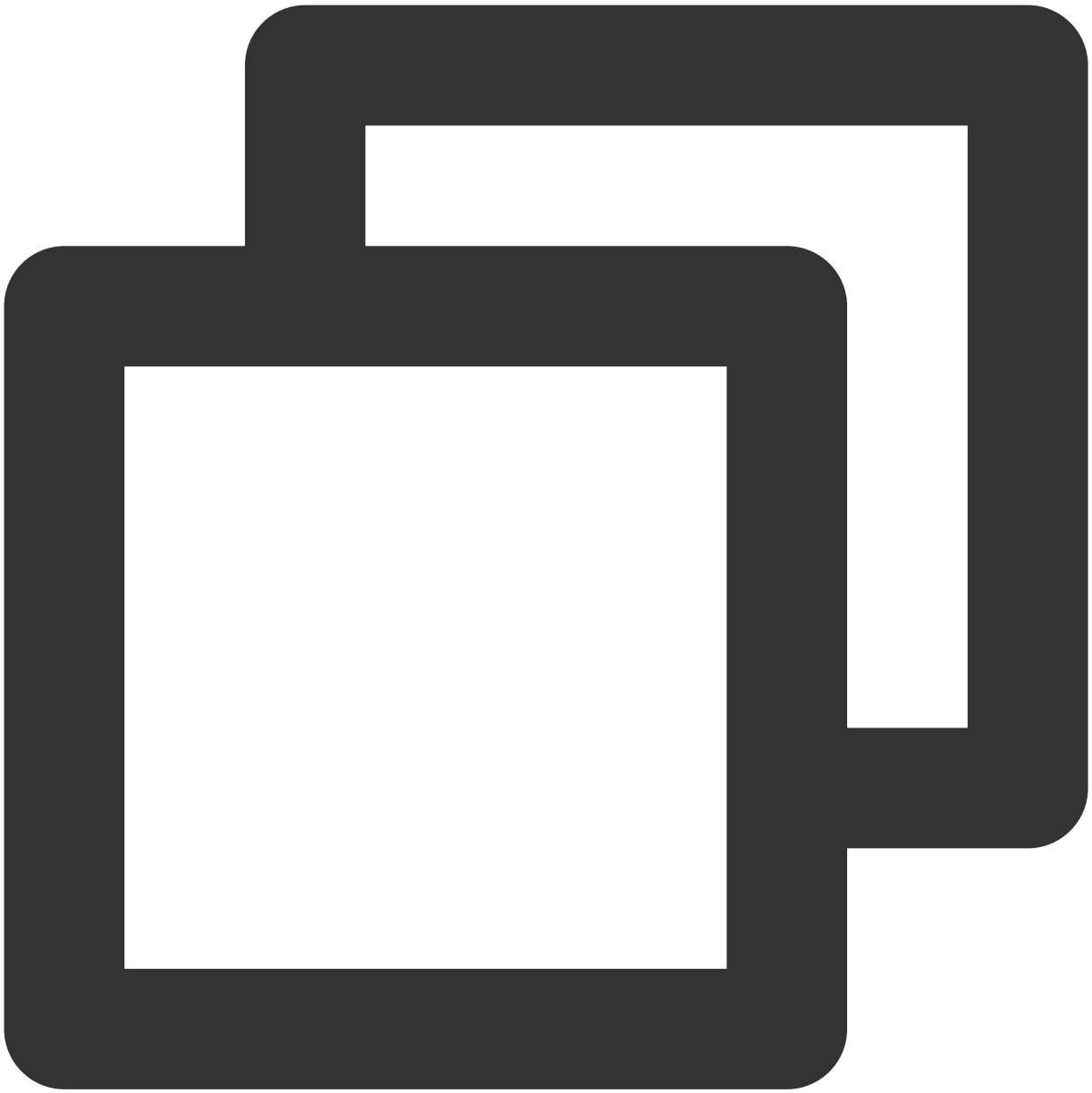
```
- (void)onUserNoResponse:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	无响应用户的 ID

onUserLineBusy

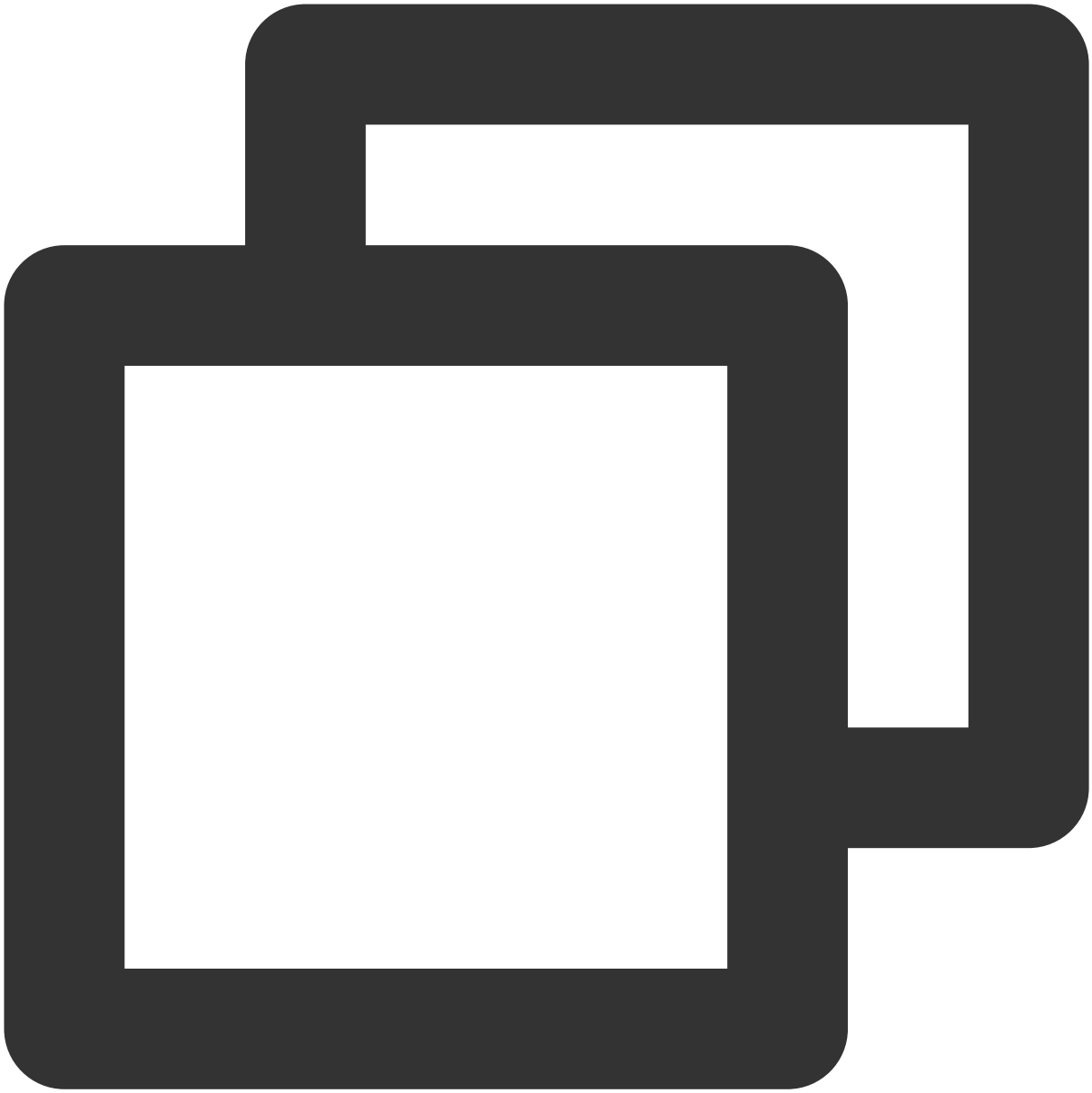
通话忙线回调。



```
- (void)onUserLineBusy:(NSString *)userId;
```

onUserJoin

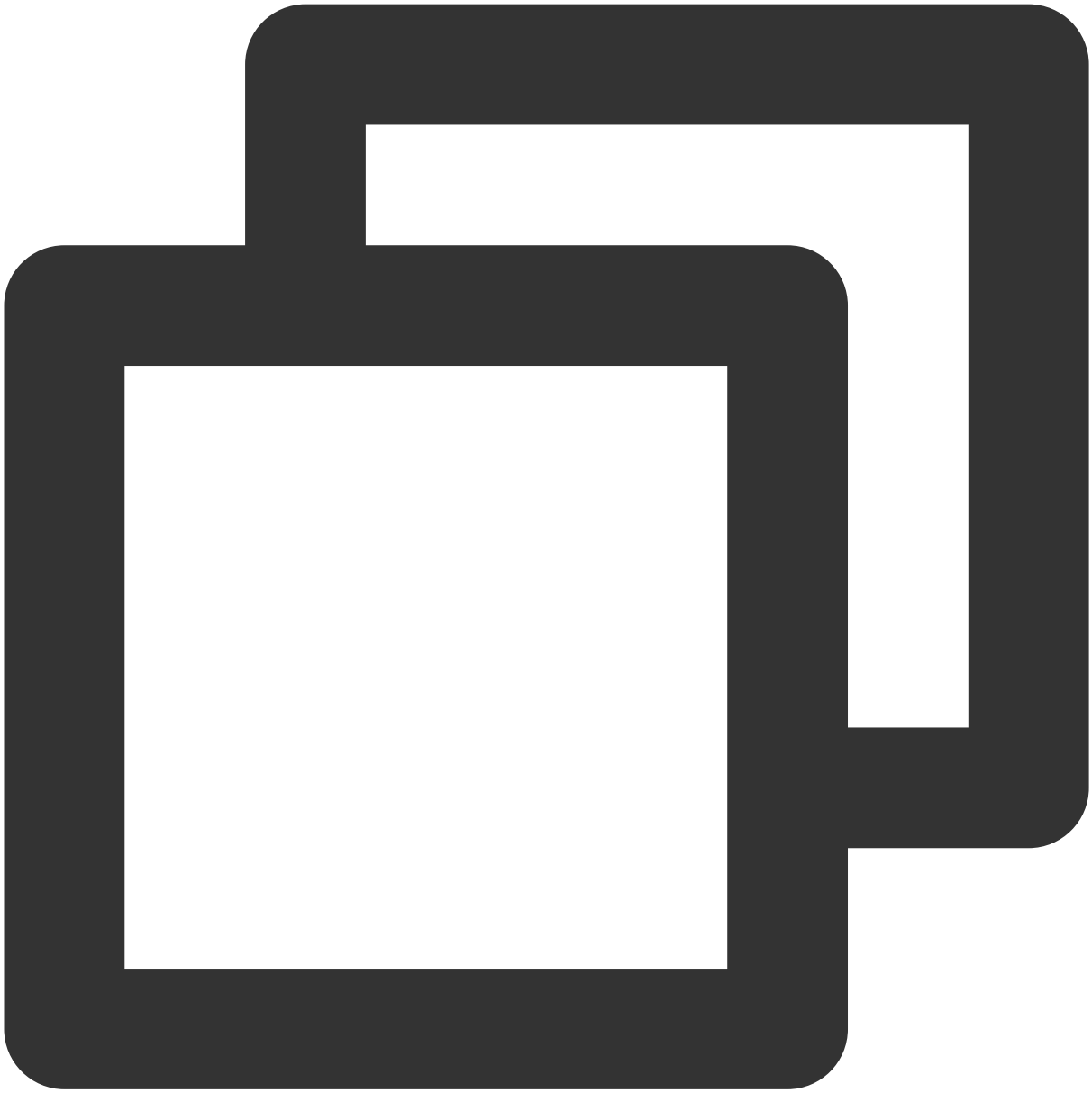
有用户进入此次通话的回调。



```
- (void)onUserJoin:(NSString *)userId;
```

onUserLeave

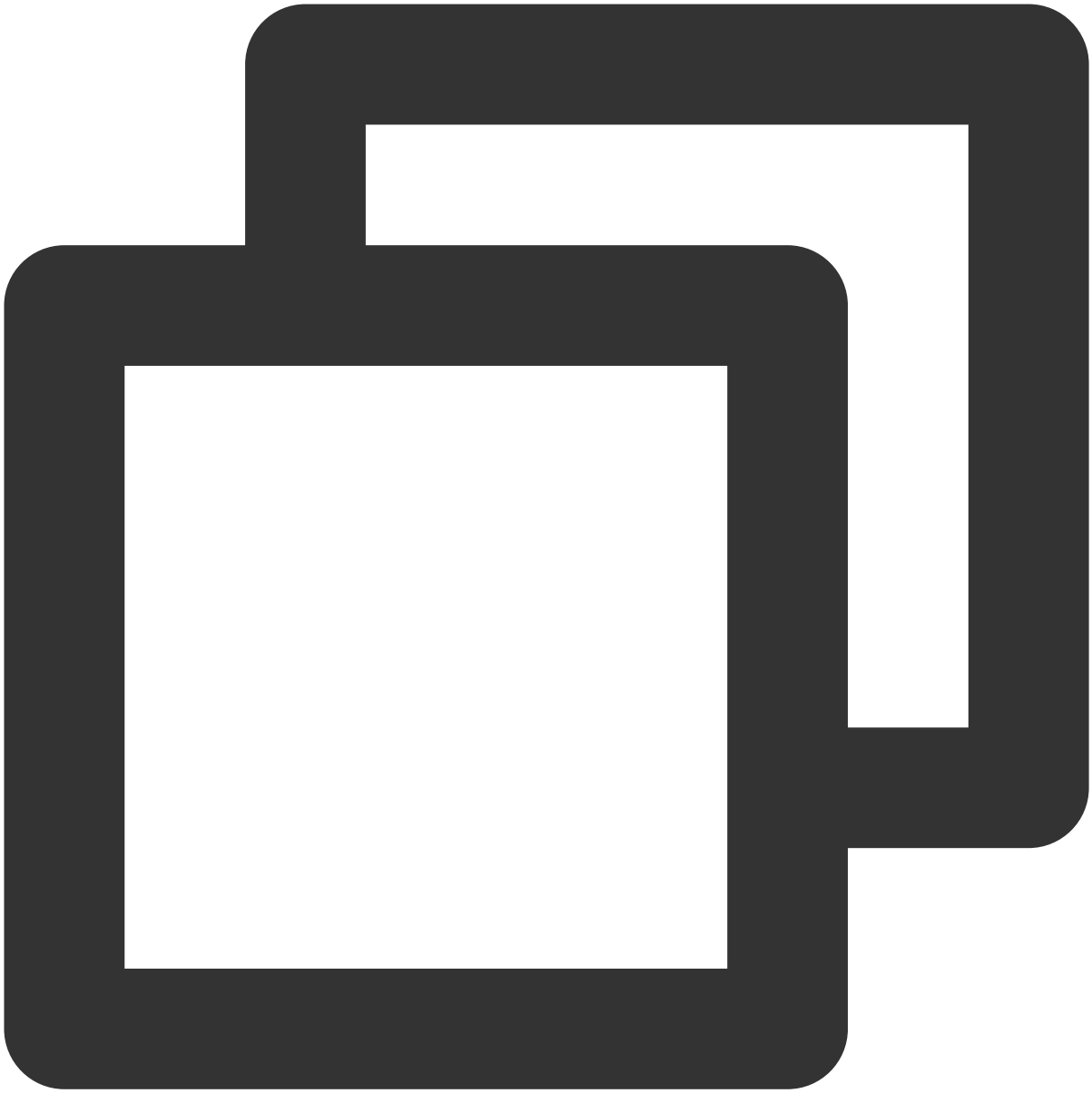
有用户离开此次通话的回调。



```
- (void)onUserLeave:(NSString *)userId;
```

onUserAudioAvailable

用户是否开启音频上行回调。



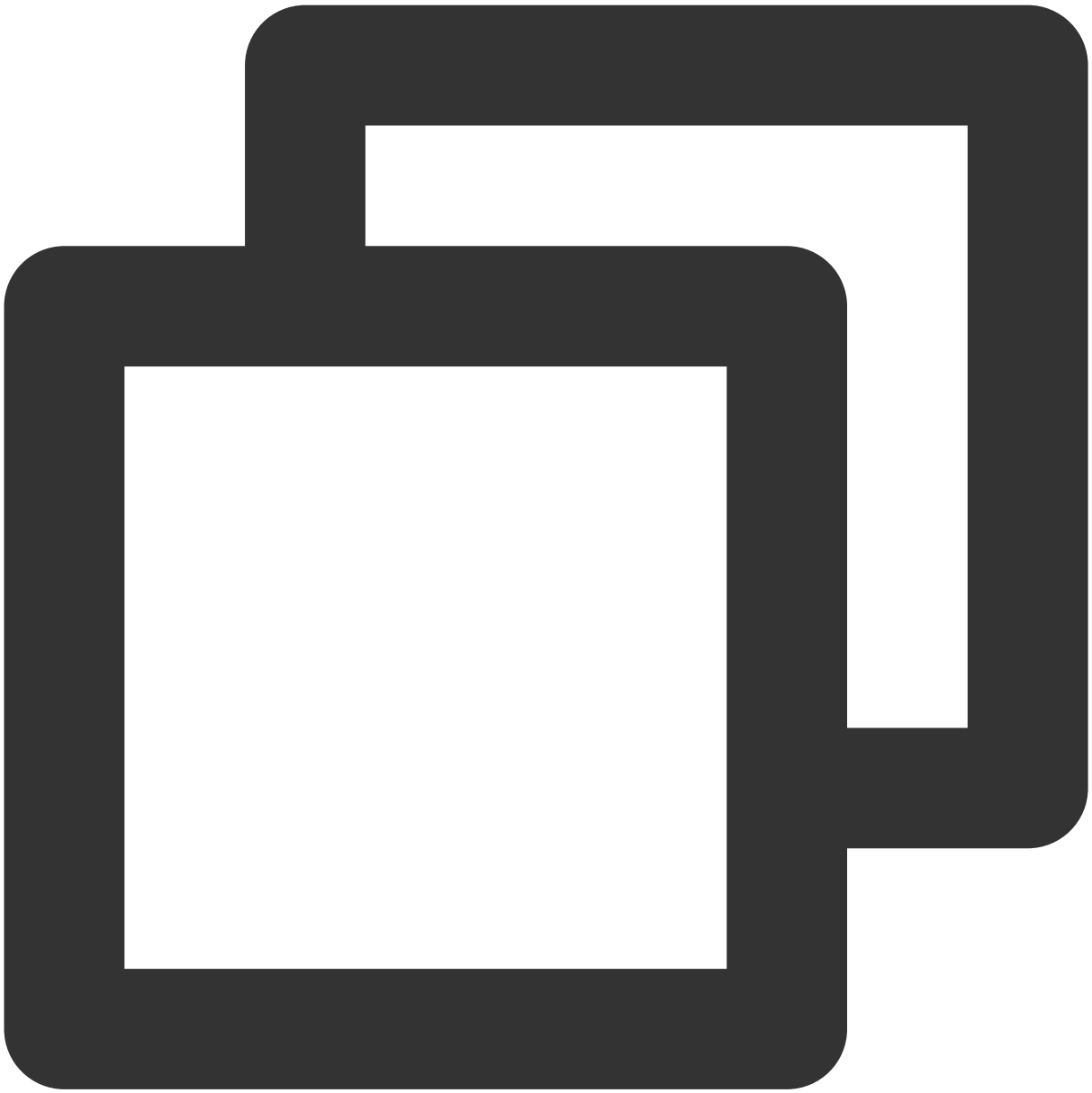
```
- (void)onUserAudioAvailable:(NSString *)userId isAudioAvailable:(BOOL)isAudioAvail
```

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。



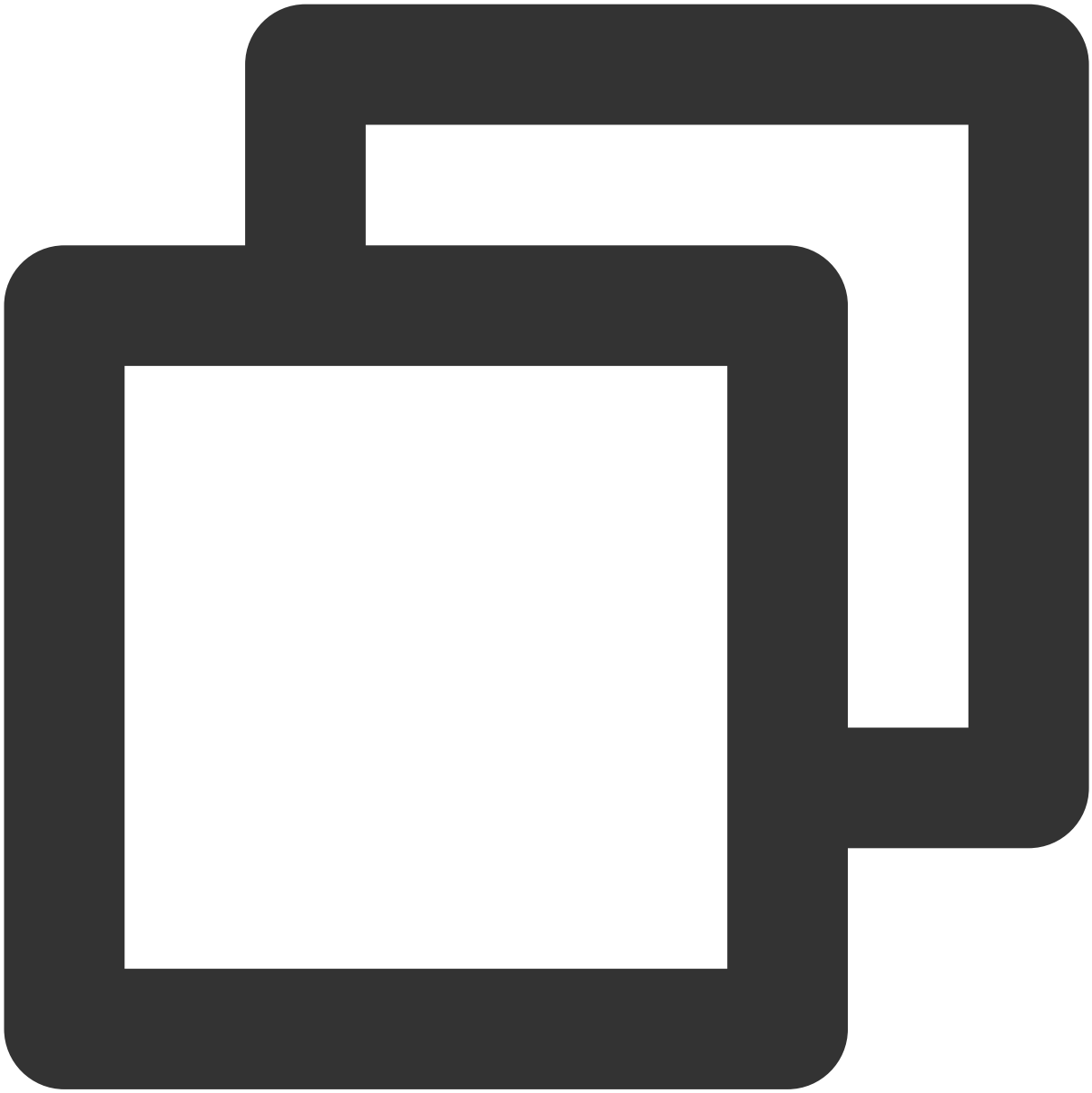
```
- (void)onUserVideoAvailable:(NSString *)userId isVideoAvailable:(BOOL)isVideoAvail
```

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。



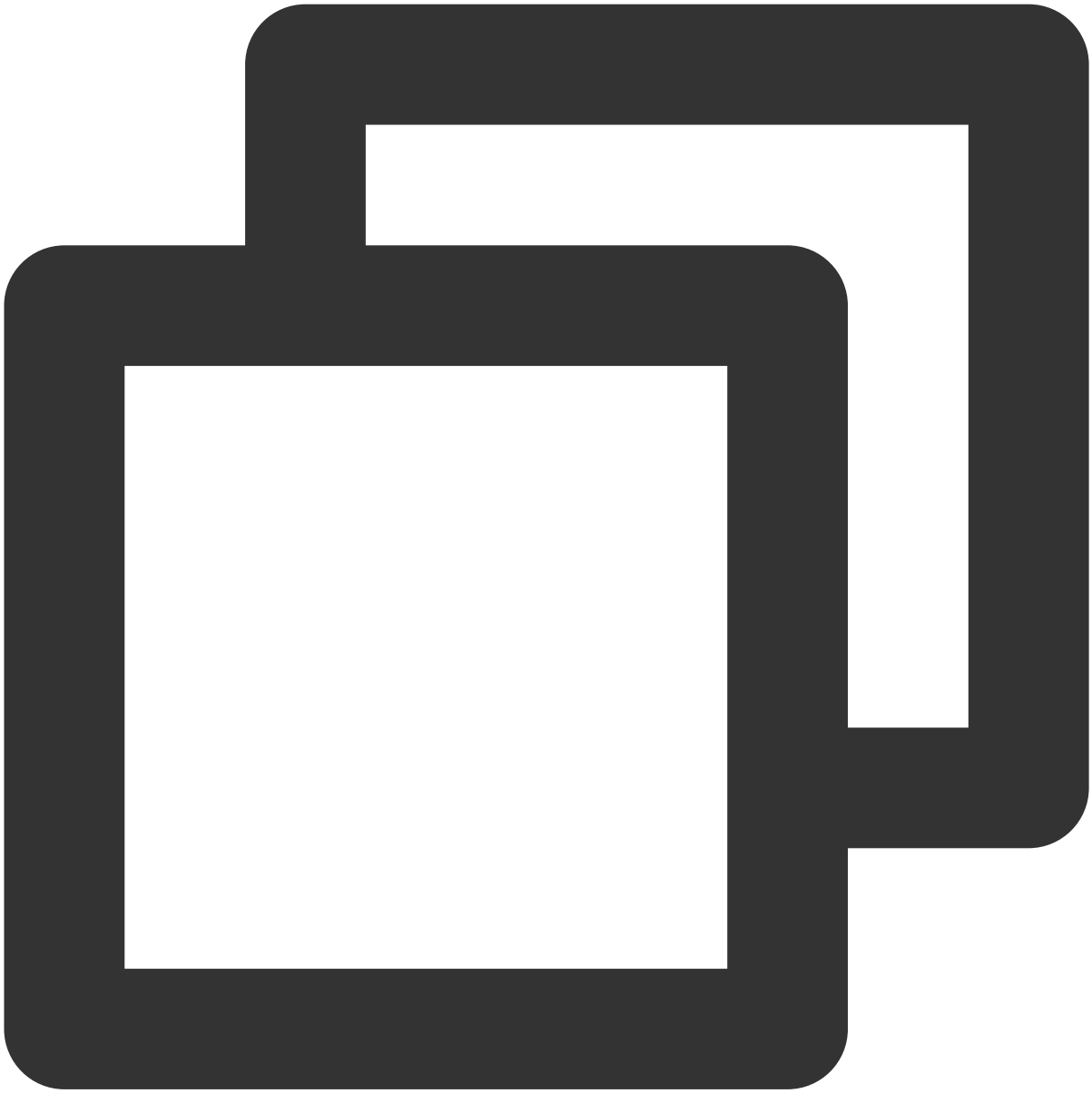
```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap
```

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。



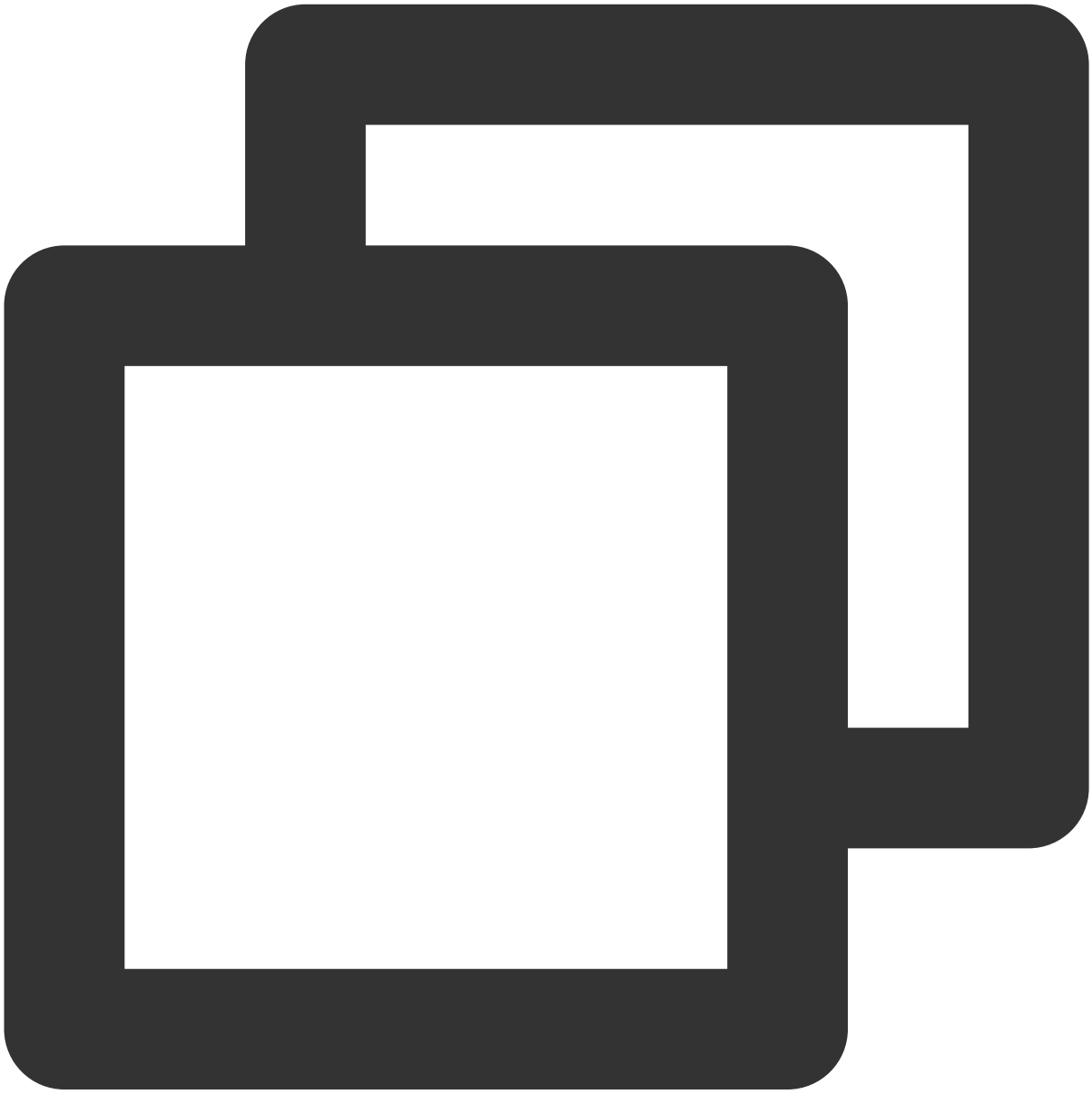
```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQual
```

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

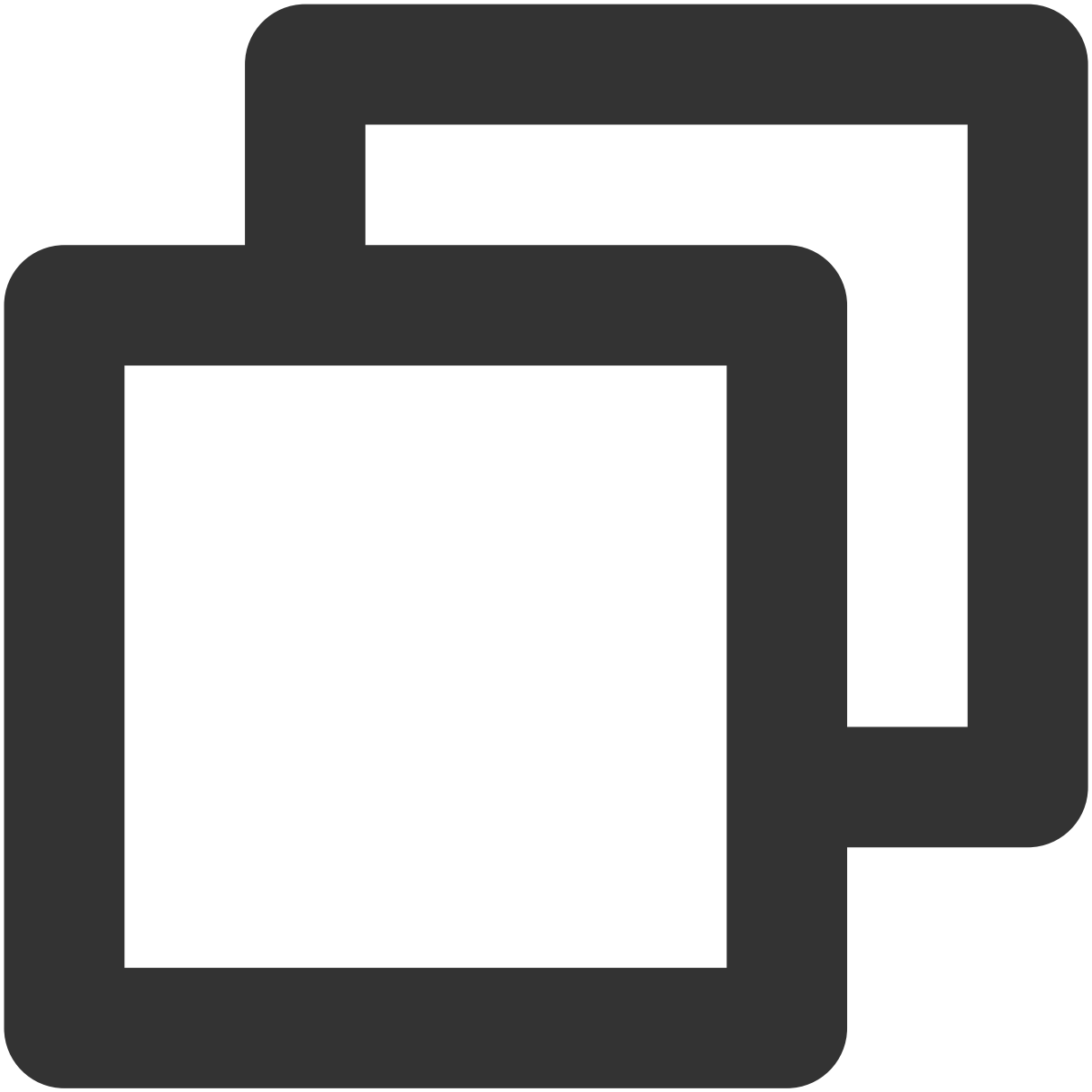
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
- (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：2024-05-29 10:35:43

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，详情配置步骤请参见 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

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：2024-08-09 22:25:01

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。

API	描述
init	初始化 TUICallKit。
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，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，**v3.1.2+ 支持**
setCallingBell	设置自定义来电铃音，**v3.0.0+ 支持**
setSelfInfo	设置用户昵称和头像，**v2.2.0+ 支持**
enableMuteMode	开启/关闭来电铃声，**v3.1.2+ 支持**
enableFloatWindow	开启/关闭悬浮窗功能，**v3.1.0+ 支持**
enableVirtualBackground	开启/关闭模糊背景的功能按钮，**v3.2.4+ 支持**
setLanguage	设置 TUICallKit 组件通话语言
hideFeatureButton	隐藏按钮，**v3.2.9+ 支持**
setLocalViewBackgroundImage	设置本地用户通话界面背景图，**v3.2.9+ 支持**
setRemoteViewBackgroundImage	设置远端用户通话界面背景图，**v3.2.9+ 支持**
setLayoutMode	设置通话界面布局模式，**v3.3.0+ 支持**
setCameraDefaultState	设置摄像头是否默认打开，**v3.3.0+ 支持**
destroyed	销毁 TUICallKit
getTUICallEngineInstance	获取 TUICallEngine 实例，**v2.4.3+ 支持**

版权所有：腾讯云计算（北京）有限责任公司

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
switchCamera	切换前后置摄像头，**注意：仅移动端支持。v3.0.0+ 支持**
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风

版权所有：腾讯云计算（北京）有限责任公司

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
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式。**v2.1.1+ 支持**
setBlurBackground	开关/设置背景虚化，**v3.0.6+ 支持**
setVirtualBackground	开关/设置图片背景虚化，**v3.0.6+ 支持**

事件类型定义

TUICallEvent 是 TUICallEngine 对应的事件类，您可以通过监听如下事件，来监听自己感兴趣的事件。

EVENT	描述
TUICallEvent.ERROR	通话过程中错误事件
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该事件
TUICallEvent.KICKED_OUT	用户重复登录，被踢下线
TUICallEvent.USER_ACCEPT	xxxx 用户接通的事件
TUICallEvent.USER_ENTER	xxxx 用户加入通话的事件
TUICallEvent.USER_LEAVE	xxxx 用户离开通话的事件
TUICallEvent.REJECT	xxxx 用户拒绝通话的事件
TUICallEvent.NO_RESP	xxxx 用户不响应的事件
TUICallEvent.LINE_BUSY	xxxx 用户忙线的事件
TUICallEvent.USER_VIDEO_AVAILABLE	xxxx 用户是否有视频流的事件
TUICallEvent.USER_AUDIO_AVAILABLE	xxxx 用户是否有音频流的事件
TUICallEvent.USER_VOICE_VOLUME	所有用户音量大小的反馈事件
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该事件
TUICallEvent.ON_CALL_BEGIN	通话接通的事件，**v1.4.6+ 支持**

TUICallEvent.INVITED	通话请求的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_RECEIVED**
TUICallEvent.ON_CALL_RECEIVED	通话请求的事件，**v1.4.6+ 支持**
TUICallEvent.CALLING_CANCEL	通话取消的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_CANCELED	通话取消的事件，**v1.4.6+ 支持**
TUICallEvent.CALLING_END	通话结束的事件
TUICallEvent.DEVICED_UPDATED	设备列表更新的事件
TUICallEvent.CALL_TYPE_CHANGED	通话的媒体类型发生改变的事件
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件，**v3.0.7+ 支持**

文档链接

TUICallEngine

TUICallEvent

TUICallKit

最近更新时间：2024-08-09 22:25:01

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，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，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，**v3.1.2+ 支持**
setCallingBell	设置自定义来电铃音，**v3.0.0+ 支持**
setSelfInfo	设置用户昵称和头像，**v2.2.0+ 支持**
enableMuteMode	开启/关闭来电铃声，**v3.1.2+ 支持**
enableFloatWindow	开启/关闭悬浮窗功能，**v3.1.0+ 支持**
enableVirtualBackground	开启/关闭模糊背景的功能按钮，**v3.2.4+ 支持**
setLanguage	设置 TUICallKit 组件通话语言
hideFeatureButton	隐藏按钮，**v3.2.9+ 支持**
setLocalViewBackgroundImage	设置本地用户通话界面背景图，**v3.2.9+ 支持**
setRemoteViewBackgroundImage	设置远端用户通话界面背景图，**v3.2.9+ 支持**
setLayoutMode	设置通话界面布局模式，**v3.3.0+ 支持**

版权所有：腾讯云计算（北京）有限责任公司

setCameraDefaultState	设置摄像头是否默认打开，**v3.3.0+ 支持**
destroyed	销毁 TUICallKit 组件实例
getTUICallEngineInstance	获取 TUICallEngine 实例，**v2.4.3+ 支持**

<TUICallKit/> 属性介绍

属性概览

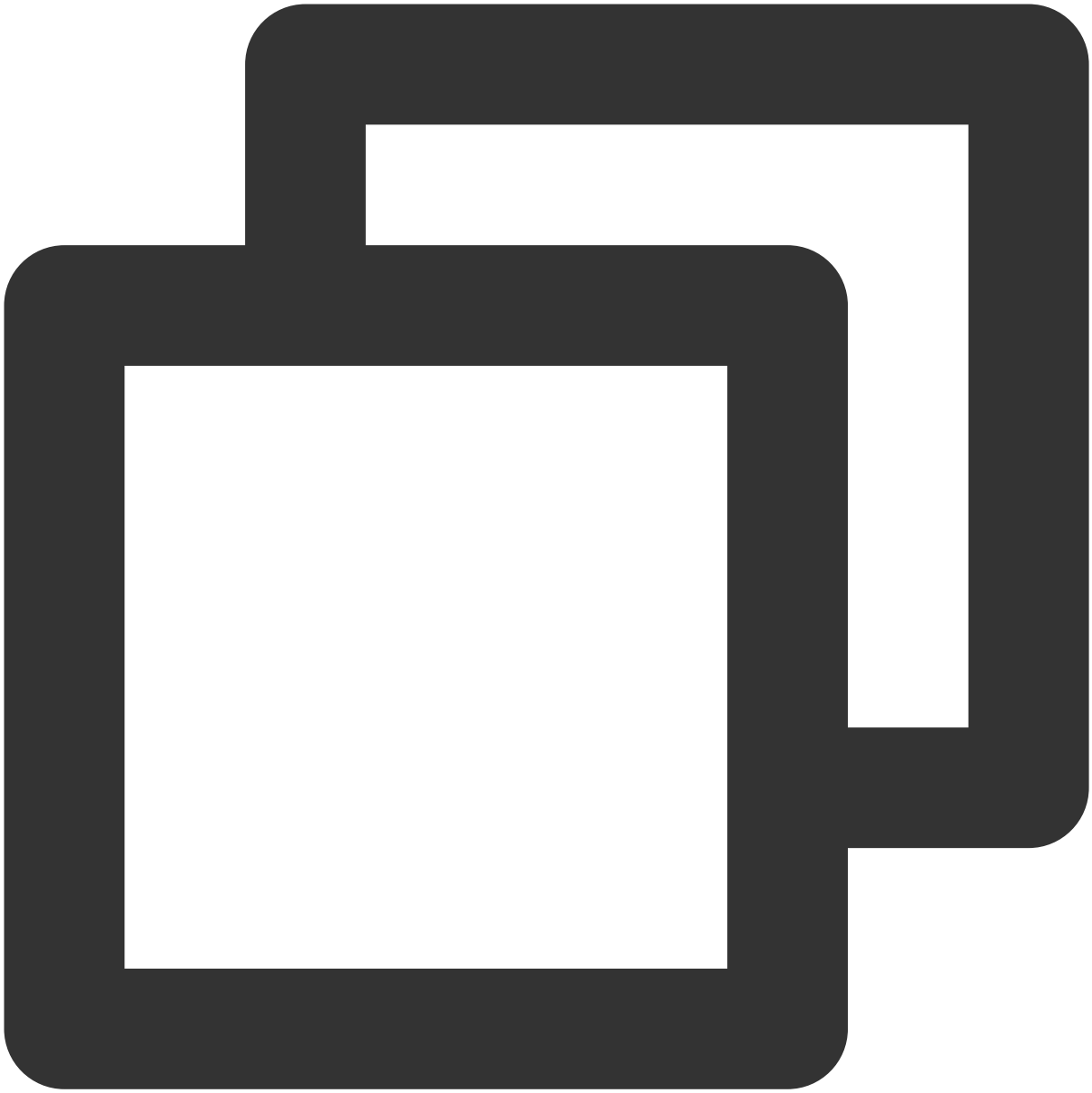
属性	描述	类型	是否必填	默认值
allowedMinimized	是否允许悬浮窗	boolean	否	false
allowedFullScreen	是否允许通话界面全屏	boolean	否	true
videoDisplayMode	通话界面显示模式	VideoDisplayMode	否	VideoDisplayMode.COVER
videoResolution	通话分辨率	VideoResolution	否	VideoResolution.RESOLUTION_480P
beforeCalling	拨打电话前与收到通话邀请前会执行此函数	function(type, error)	否	-
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	function()	否	-

	事件，通话也会自动结束			
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

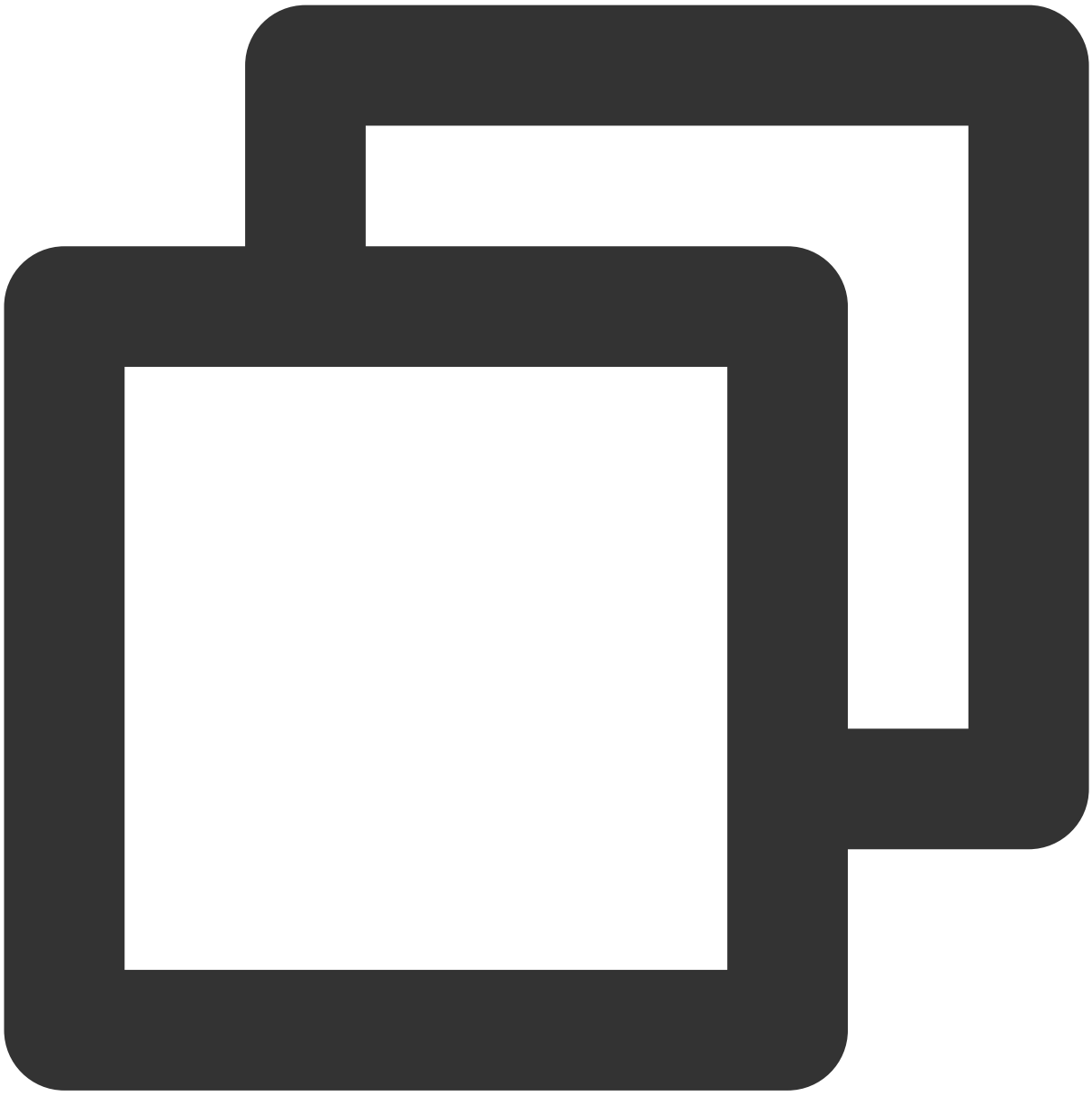


```
import { TUICallKit, VideoDisplayMode, VideoResolution } from "@tencentcloud/call-u

<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");
}
```

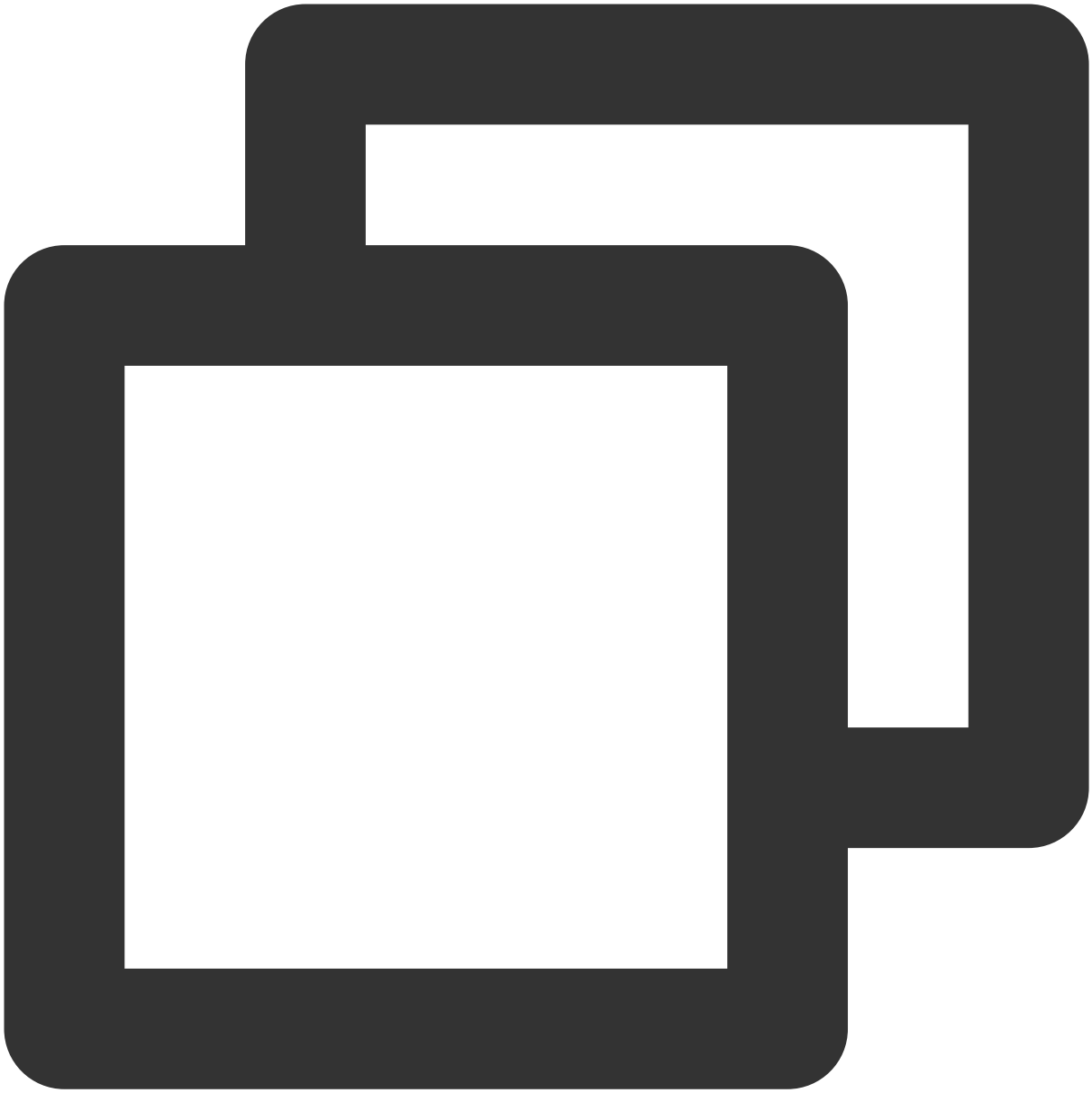


```
<template>
   <TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P"
```

```
    :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



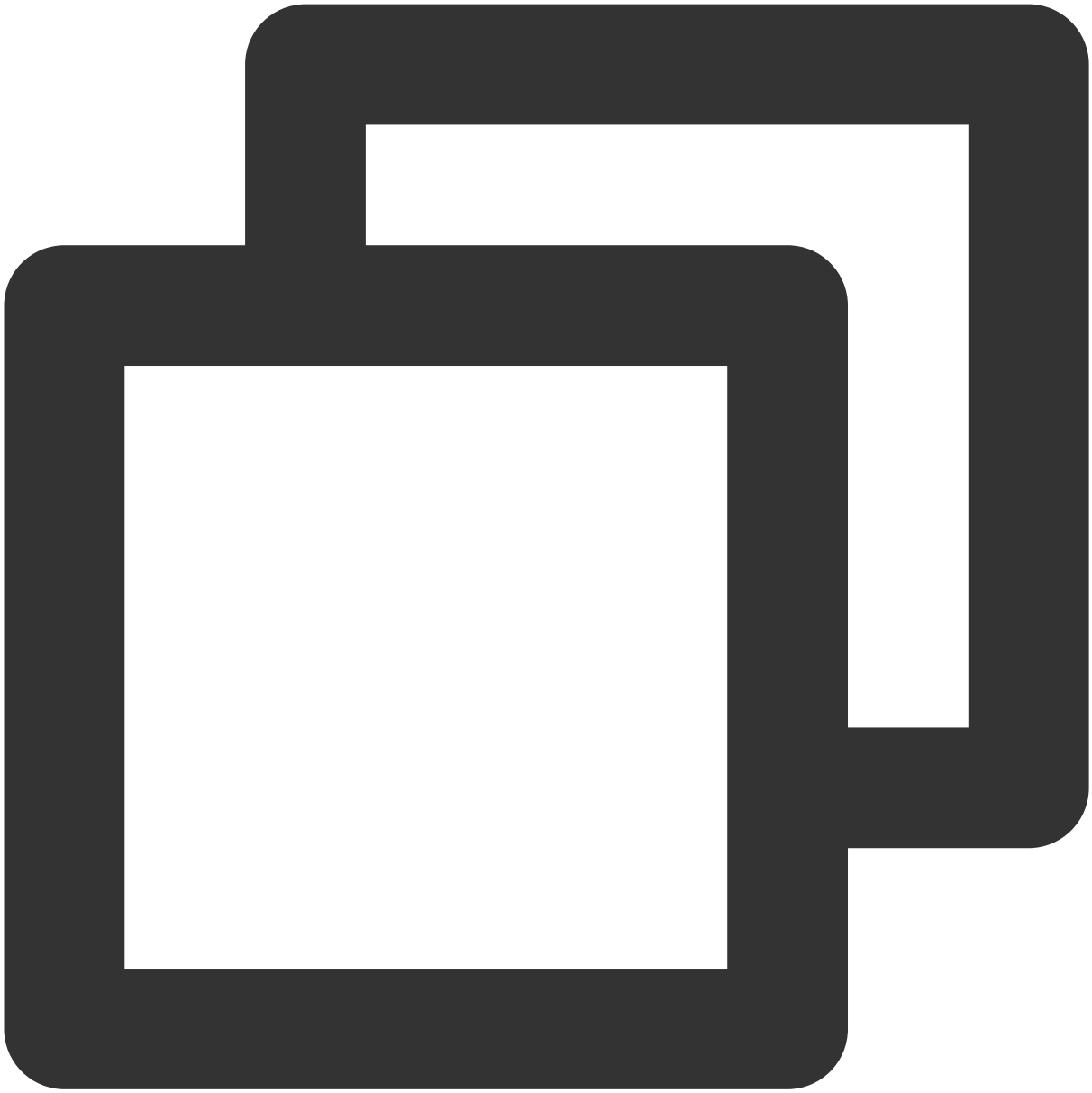
```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
// Replace it with the call-uikit npm package you are currently using
```

init

初始化 TUICallKit。

注意：

需完成 init 初始化，才能使用 call、groupCall 等功能。



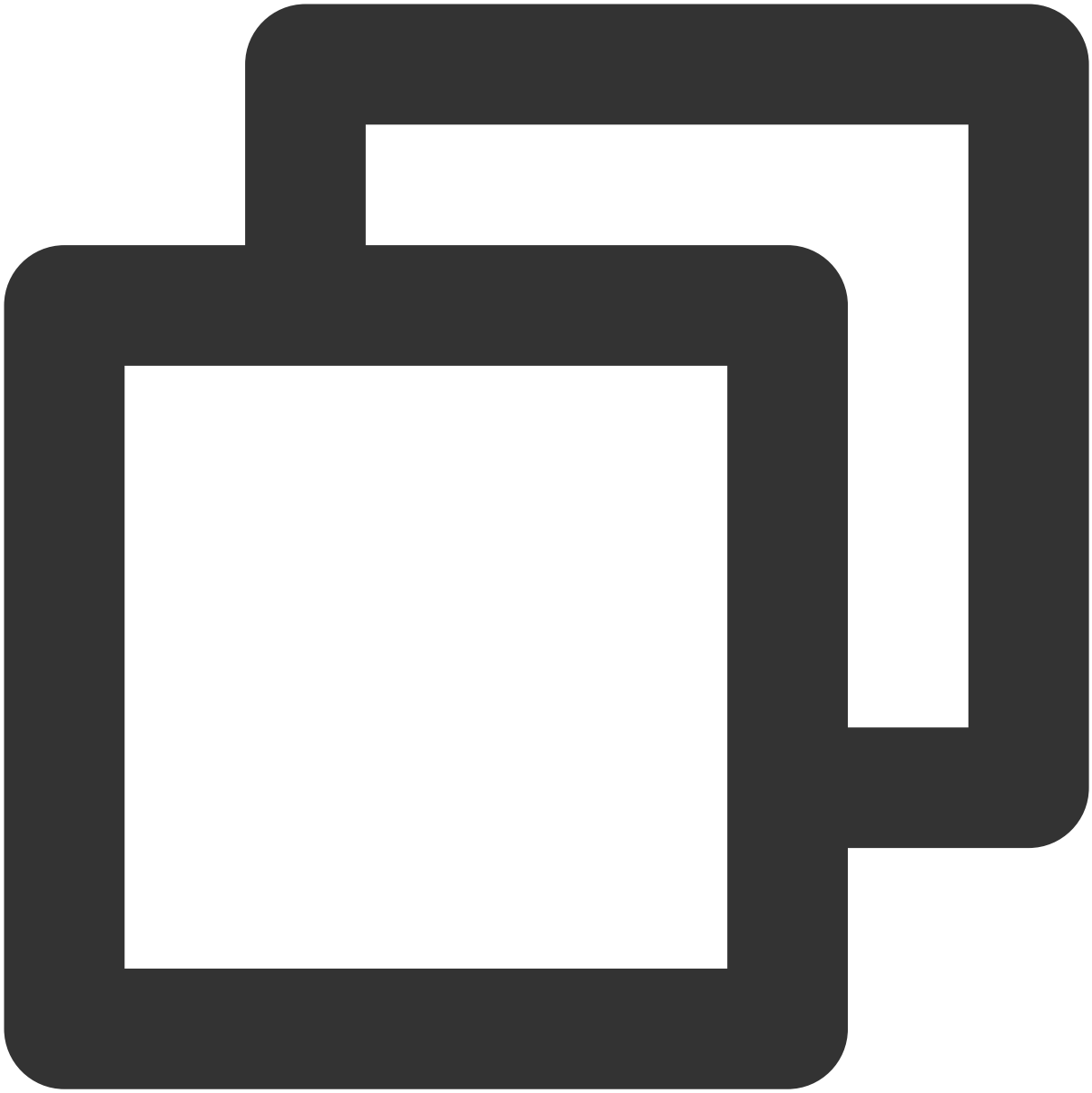
```
try {
  await TUICallKitServer.init({ SDKAppID, userID, userSig });
  // If you already have a tim instance in your project, you need to pass it in her
  // await TUICallKitServer.init({ tim, SDKAppID, userID, userSig});
  console.log("[TUICallKit] Initialization succeeds.");
} catch (error: any) {
  console.error(`[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	应用的 SDKAppID，您可以在实时音视频控制台中找到您的 SDKAppID。具体详见 开通服务
userID	String	是	即用户名，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（_）。 **注意：**不支持同一个 userID 在两台不同的设备上同时进入同一个房间通话，否则会相互干扰。
userSig	String	是	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，获取方式请参考 如何计算 UserSig
tim	TencentCloudChat	否	tim 是 TencentCloudChat SDK 的实例

call

拨打 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。



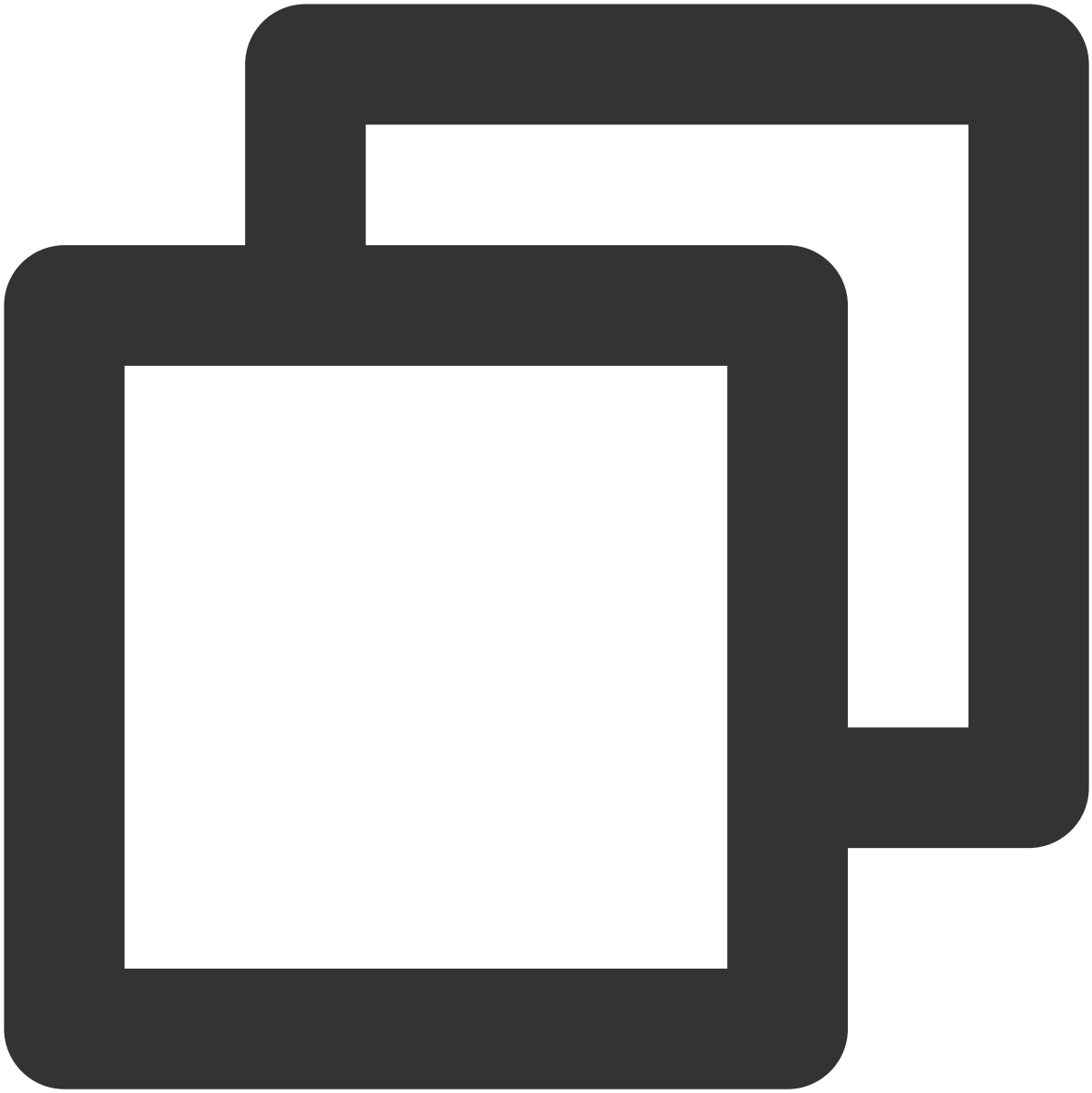
```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.call({
    userID: 'mike', // username being called
    type: TUICallType.VIDEO_CALL,
  });
} catch (error: any) {
  console.error(`[TUICallKit] Call failed. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userID	String	是	被呼叫用户的用户名
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]
strRoomID	String	否	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 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。
timeout	Number	否	通话超时时间，默认：30s，单位：秒。timeout = 0，设置为不超时
userData	String	否	发起通话时自定义扩展字段，被呼叫用户在 ON_CALL_RECEIVED 事件中有该参数
offlinePushInfo	Object	否	自定义离线消息推送参数

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。



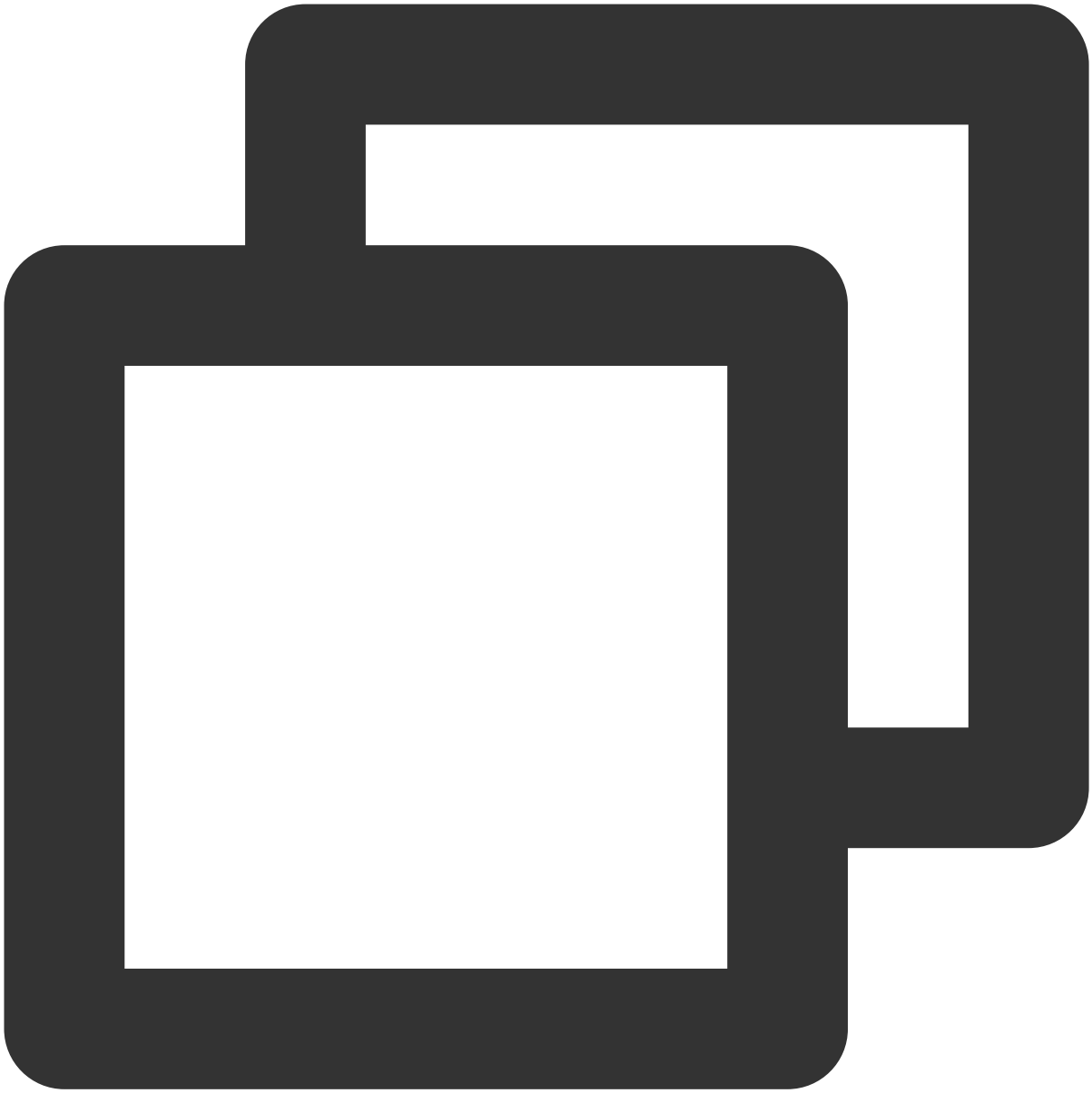
```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.groupCall({
    userIDList: ['jack', 'tom'],
    groupID: 'xxx',
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userIDList	Array<String>	是	被呼叫的用户列表
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	此次群组通话的群 ID，groupID 的创建可参考 chat-createGroup API
roomID	Number	否	数字房间号，范围 [1, 2147483647]
strRoomID	String	否	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 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。
timeout	Number	否	通话超时时间，默认：30s，单位：秒。timeout = 0，设置为不超时
userData	String	否	发起通话时自定义扩展字段，被呼叫用户在 ON_CALL_RECEIVED 事件中有该参数
offlinePushInfo	Object	否	自定义离线消息推送参数

setLanguage

设置语言，目前支持：中文、英文、日文。



```
TUICallKitServer.setLanguage("zh-cn"); // "en" | "zh-cn" | "ja_JP"
```

参数如下表所示：

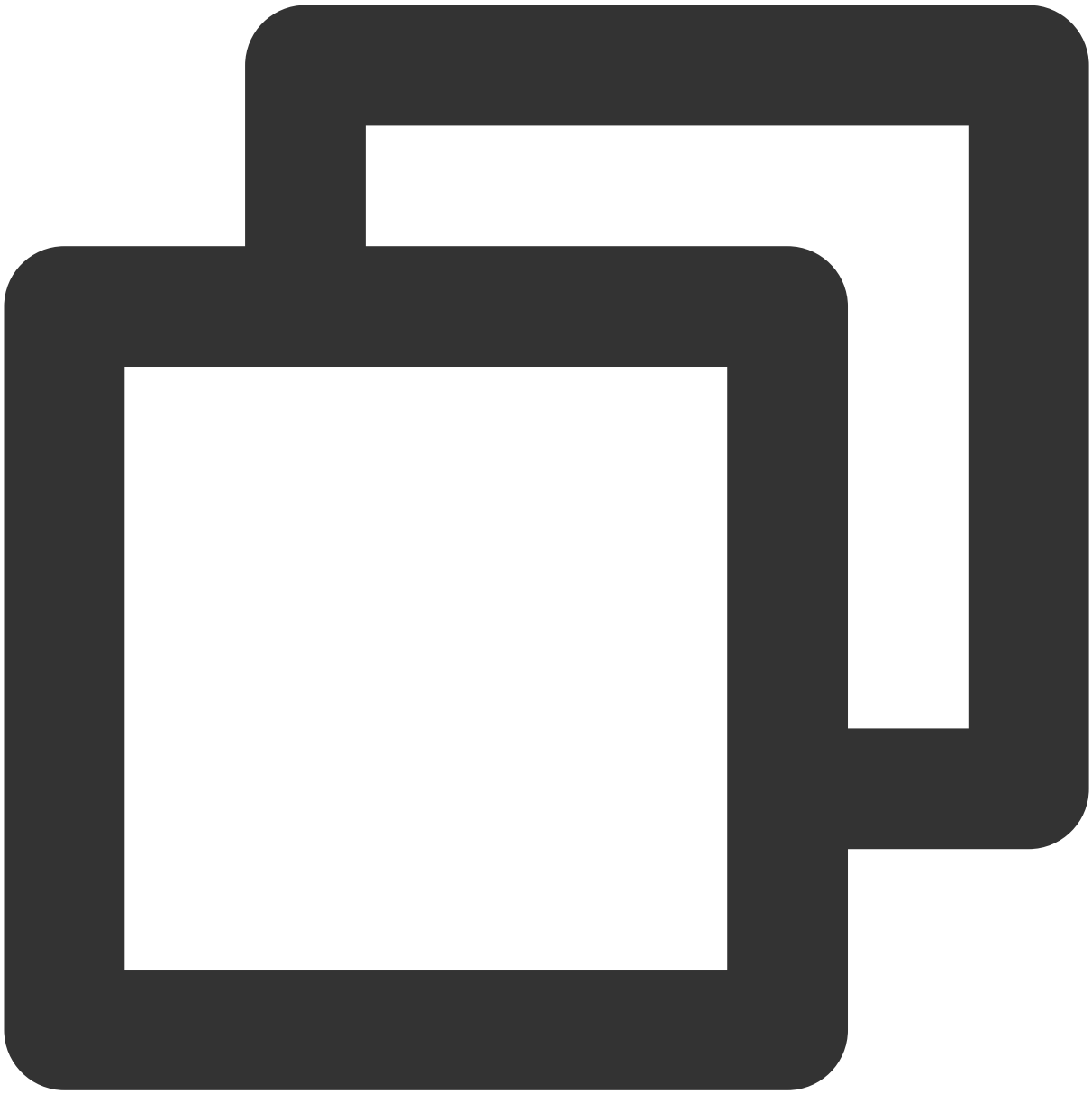
参数	类型	是否必填	含义
lang	String	是	语言类型 `en` 、 `zh-cn` 和 `ja_JP` 。

setSelfInfo

设置自己的昵称和头像。

注意：

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。



```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "xxx", avatar: "http://xxx" });
} catch (error: any) {
  console.error(`[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	自己的昵称
avatar	String	是	自己头像地址

setCallingBell

注意：

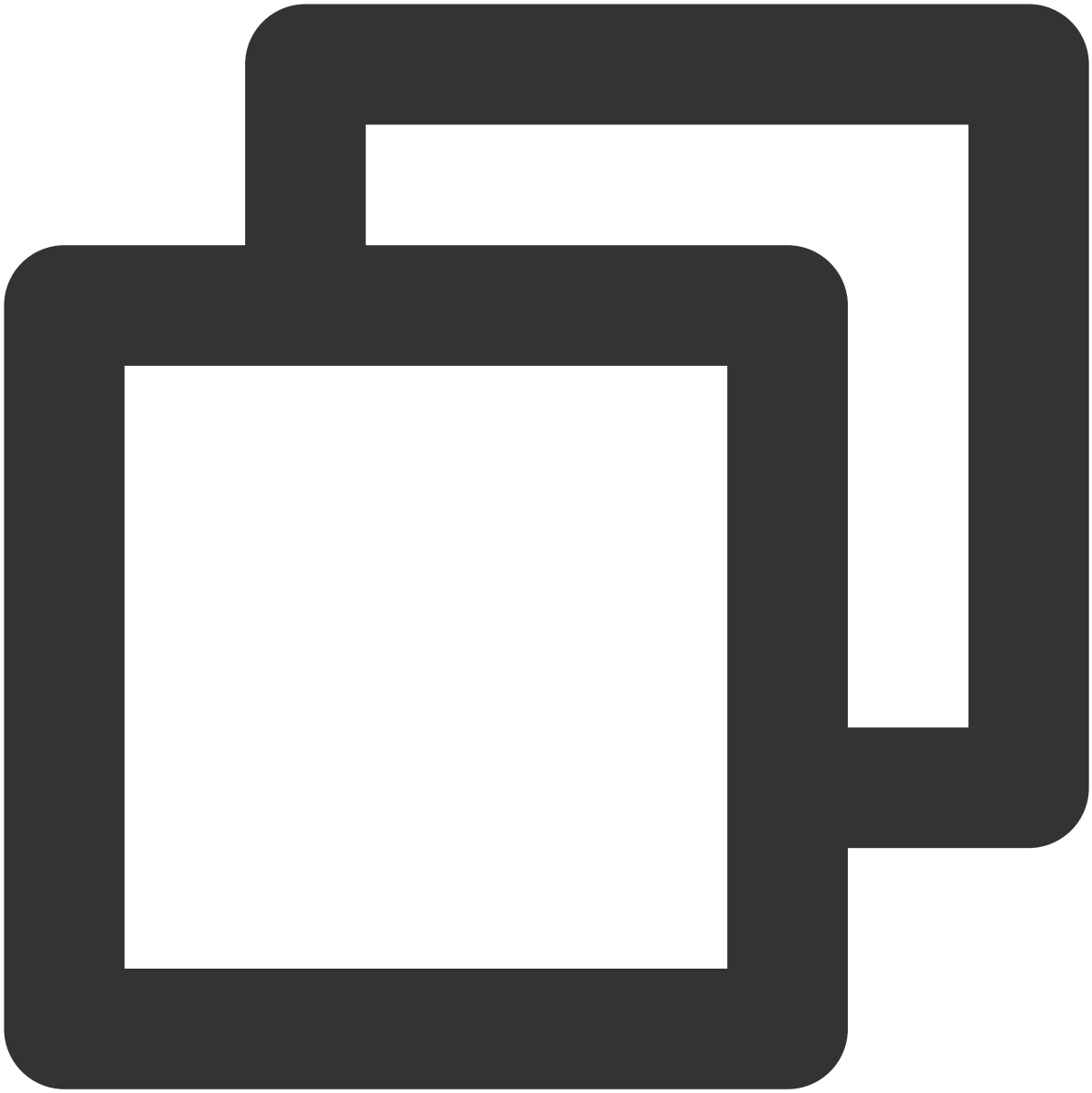
v3.0.0+ 支持。

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 import 方式引入铃声文件。

如需恢复默认铃声，`filePath` 传空即可。



```
import filePath from '../assets/phone_ringing.mp3';
try {
  await TUICallKitServer.setCallingBell(filePath)
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the setCallingBell API. Reason: ${erro
}
```

参数如下表所示：

参数	类型	是否必填	含义

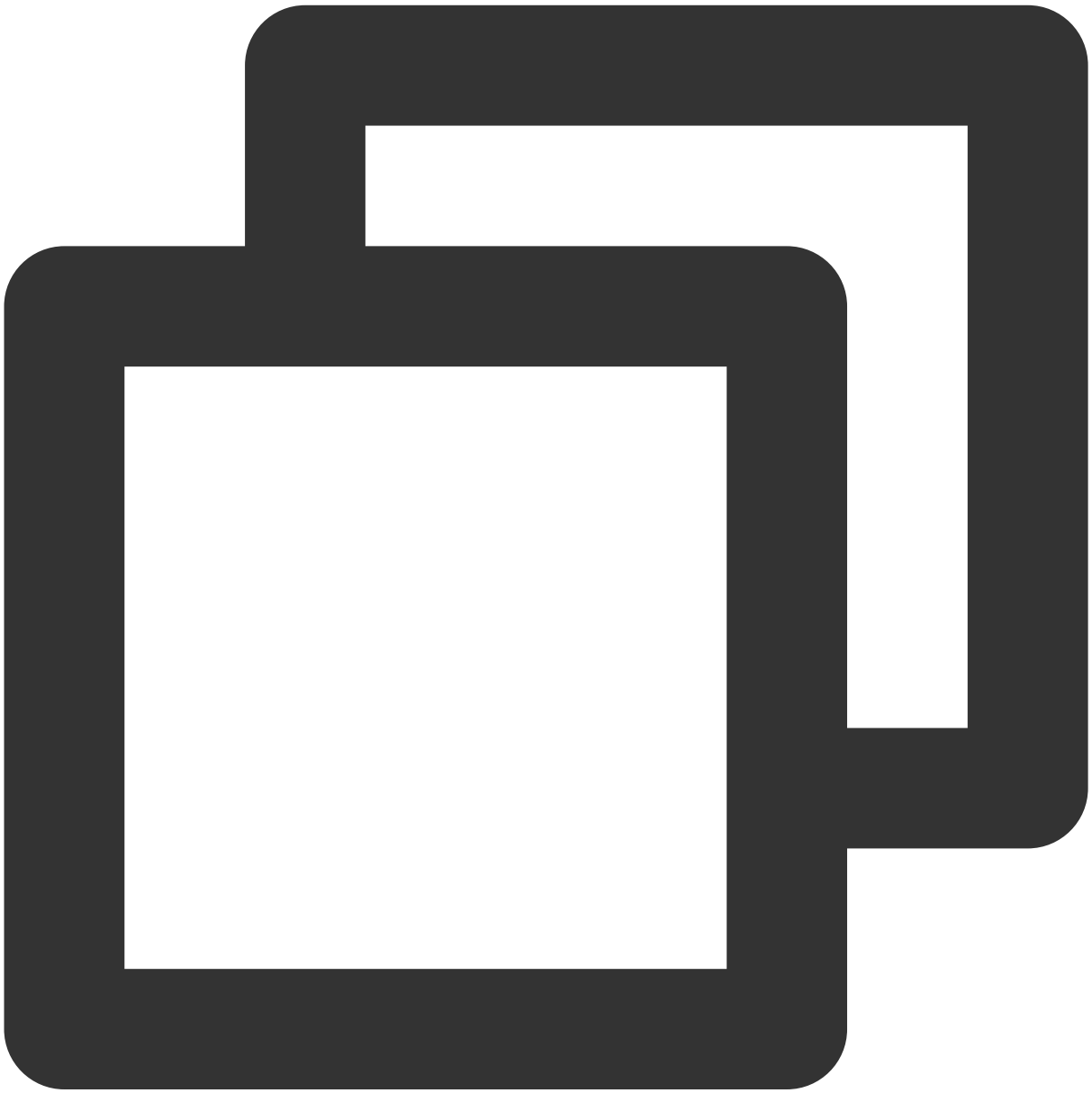
filePath	String	是	铃声文件地址

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。

注意：

v3.1.0+ 支持。



```
try {
  const enable = true;
  await TUICallKitServer.enableFloatWindow(enable);
```

```
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableFloatWindow API. Reason: ${e
}
```

参数如下表所示：

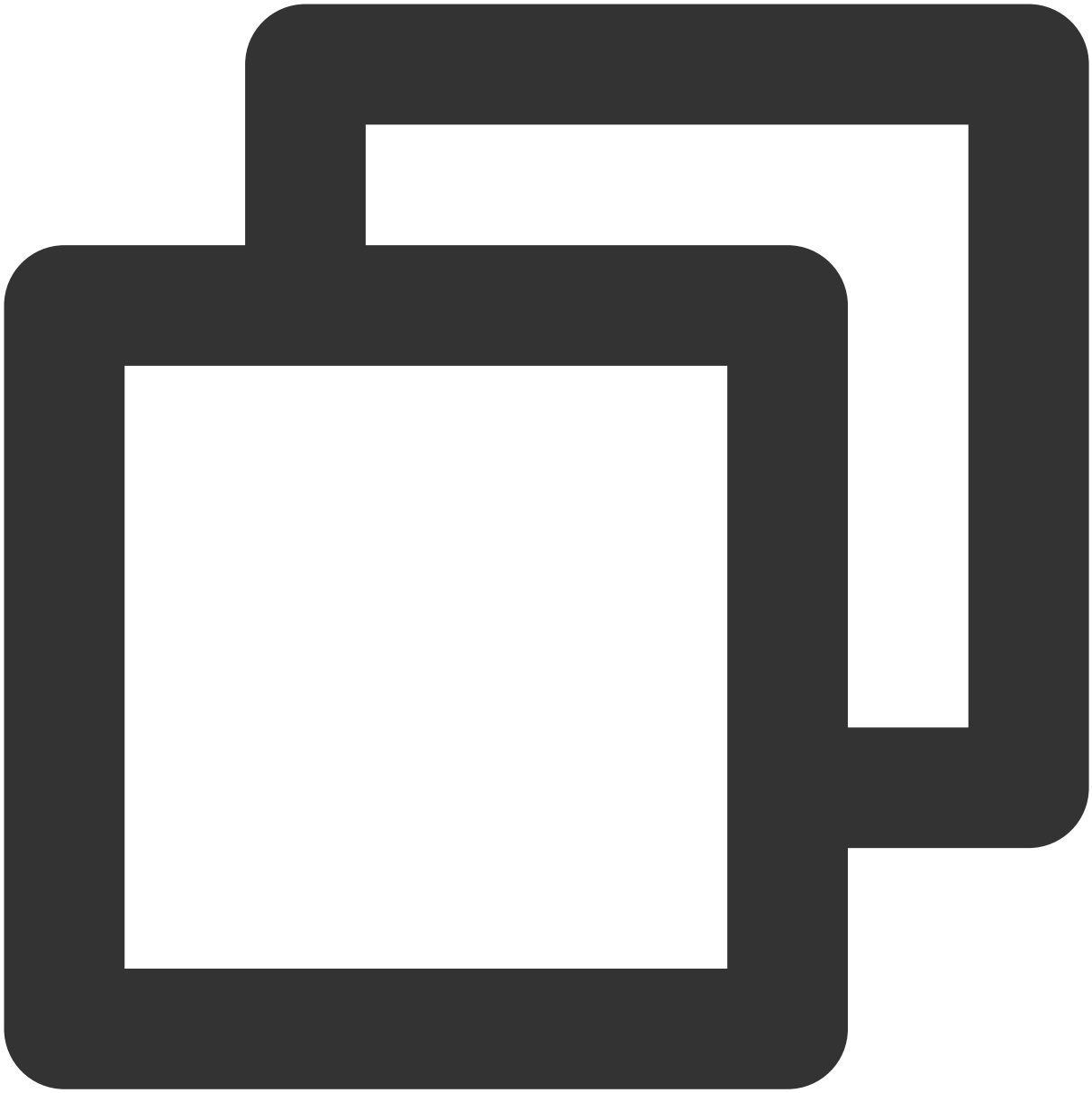
参数	类型	是否必填	含义
enable	Boolean	是	开启/关闭悬浮窗功能

enableMuteMode

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。

注意：

v3.1.2+ 支持



```
try {
  const enable = true;
  await TUICallKitServer.enableMuteMode(enable);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${erro
}
```

参数如下表所示：

参数	类型	是否必填	含义

enable	Boolean	是	开启/关闭来电铃声。默认 false。

joinInGroupCall

加入群组中已有的音视频通话。

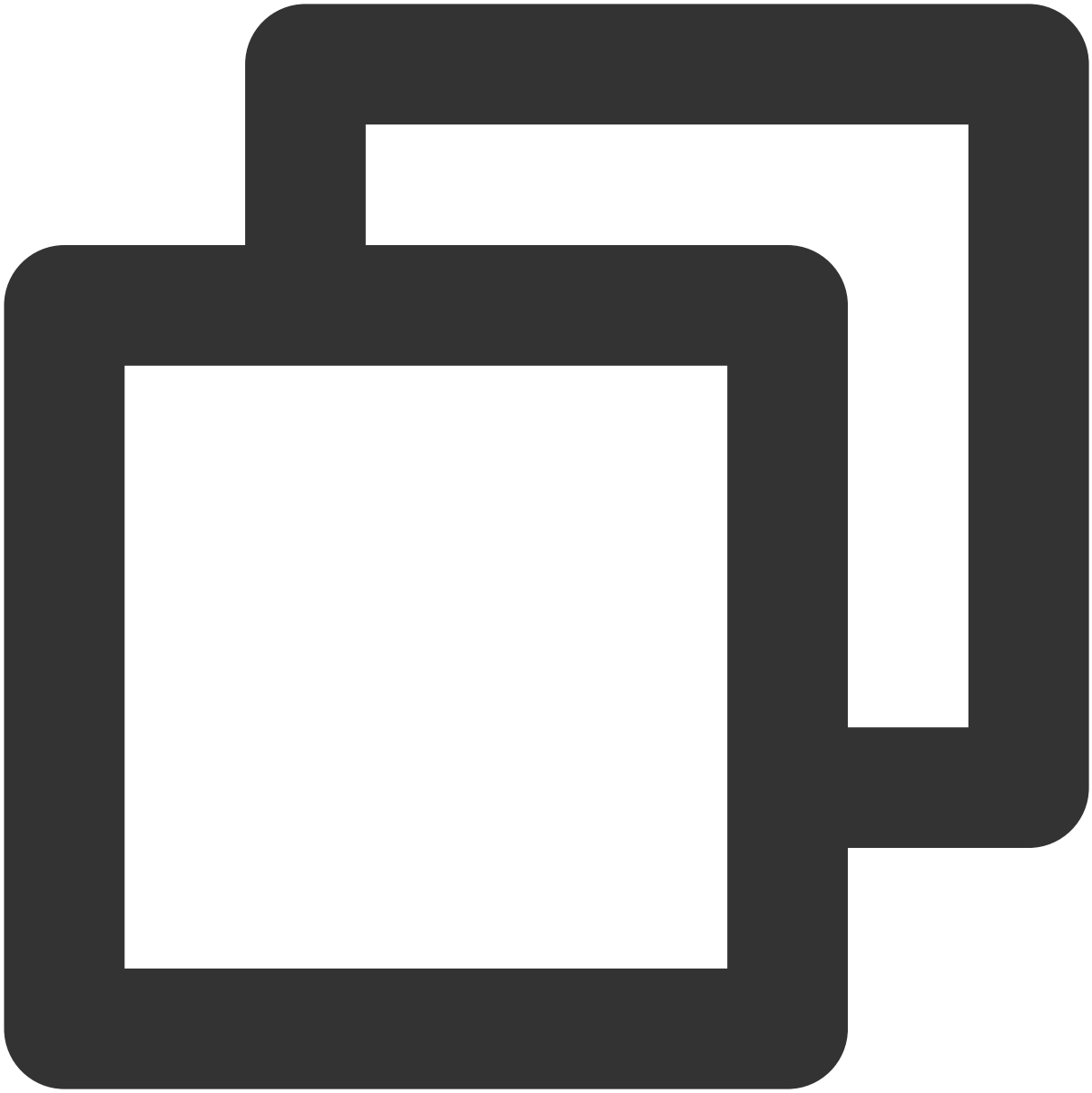
注意：

v3.1.2+ 支持。

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 Chat 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： Chat 群组管理 ，或者您也可以直接使用 Chat TUIKit，一站式集成聊天、通话等场景。



```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
    roomID: 234,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${erro
}
```

版权所有：腾讯云计算（北京）有限责任公司

参数列表：

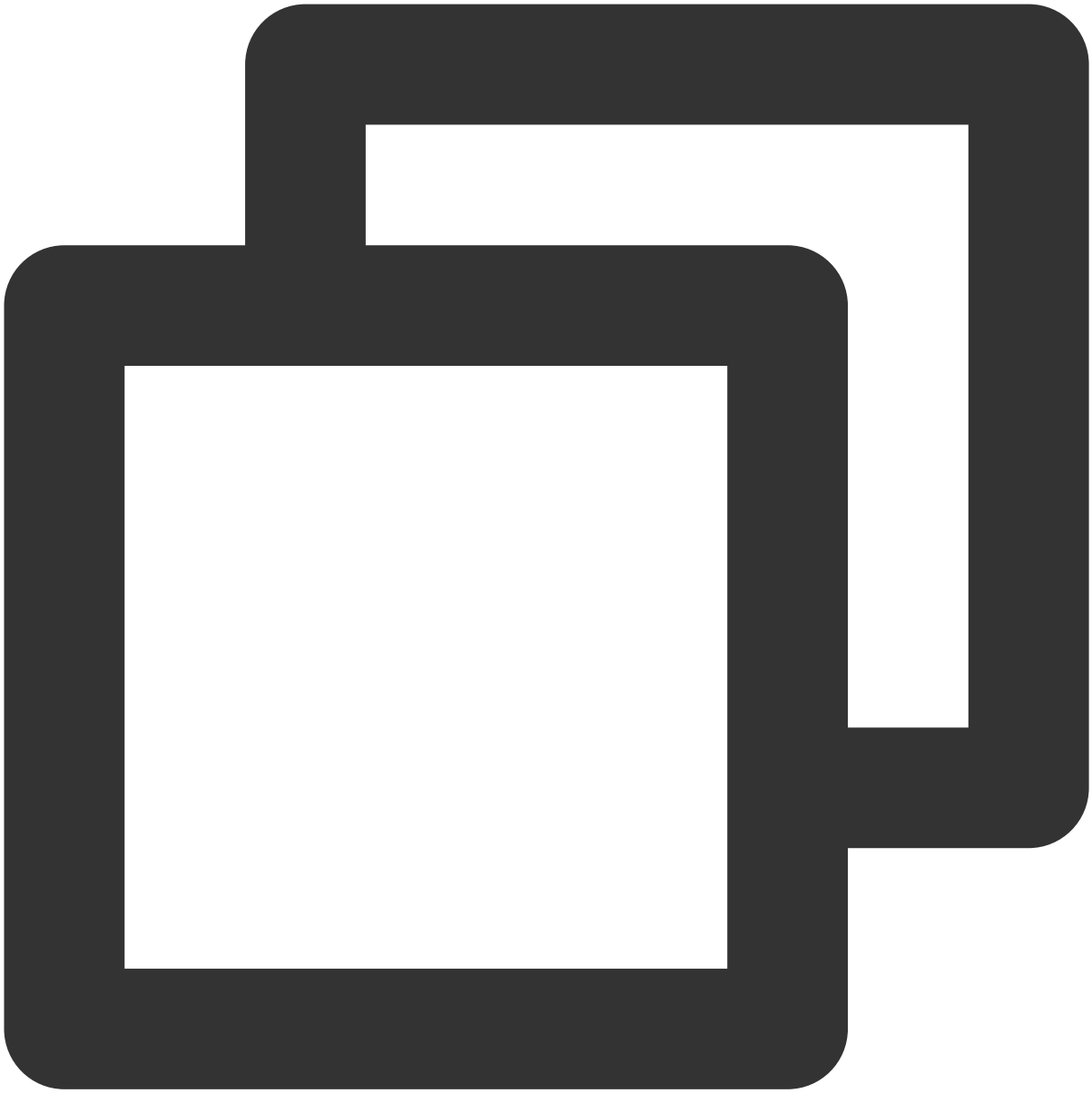
参数	类型	是否必填	含义
type	TUICallType	是	通话的媒体类型，例如视频通话、语音通话
groupID	string	是	此次群组通话的群 ID
roomID	number	是	此次通话的音视频房间 ID

enableVirtualBackground

开启/关闭模糊背景功能。如果想设置图片背景模糊参见 Web。通过调用接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

注意：

v3.2.4+ 支持。



```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
const enable = true;
TUICallKitServer.enableVirtualBackground(enable);
```

参数列表：

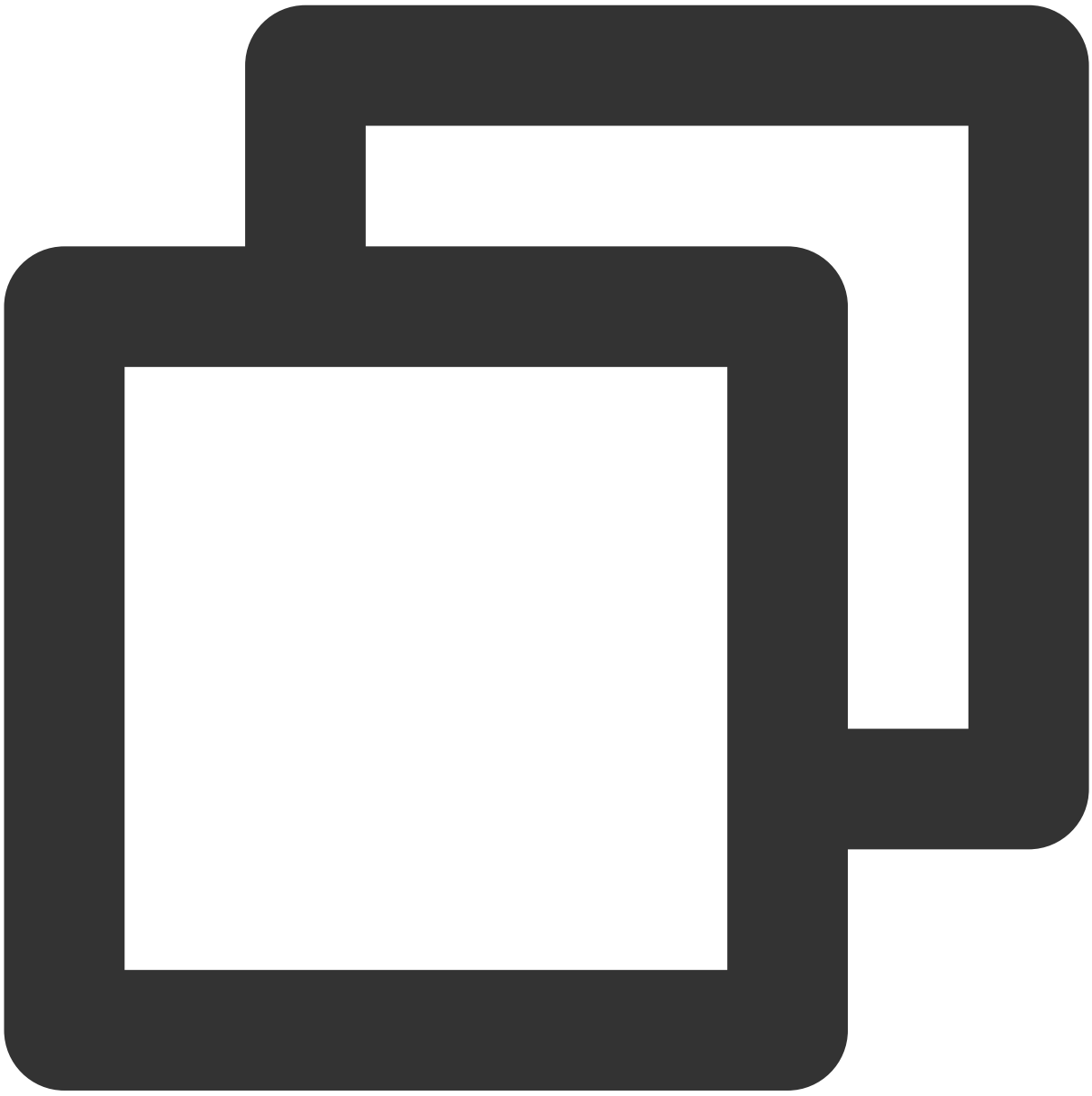
参数	类型	是否必填	含义
enable	boolean	是	enable = true 显示模糊背景按钮 enable = false 不显示模糊背景按钮

hideFeatureButton

隐藏功能按钮，目前仅支持 摄像头，麦克风和切换前后置按钮。

注意：

v3.2.9+ 支持



```
TUICallKitServer.hideFeatureButton(buttonName: FeatureButton);
```

参数列表：

参数	类型	是否必填	含义

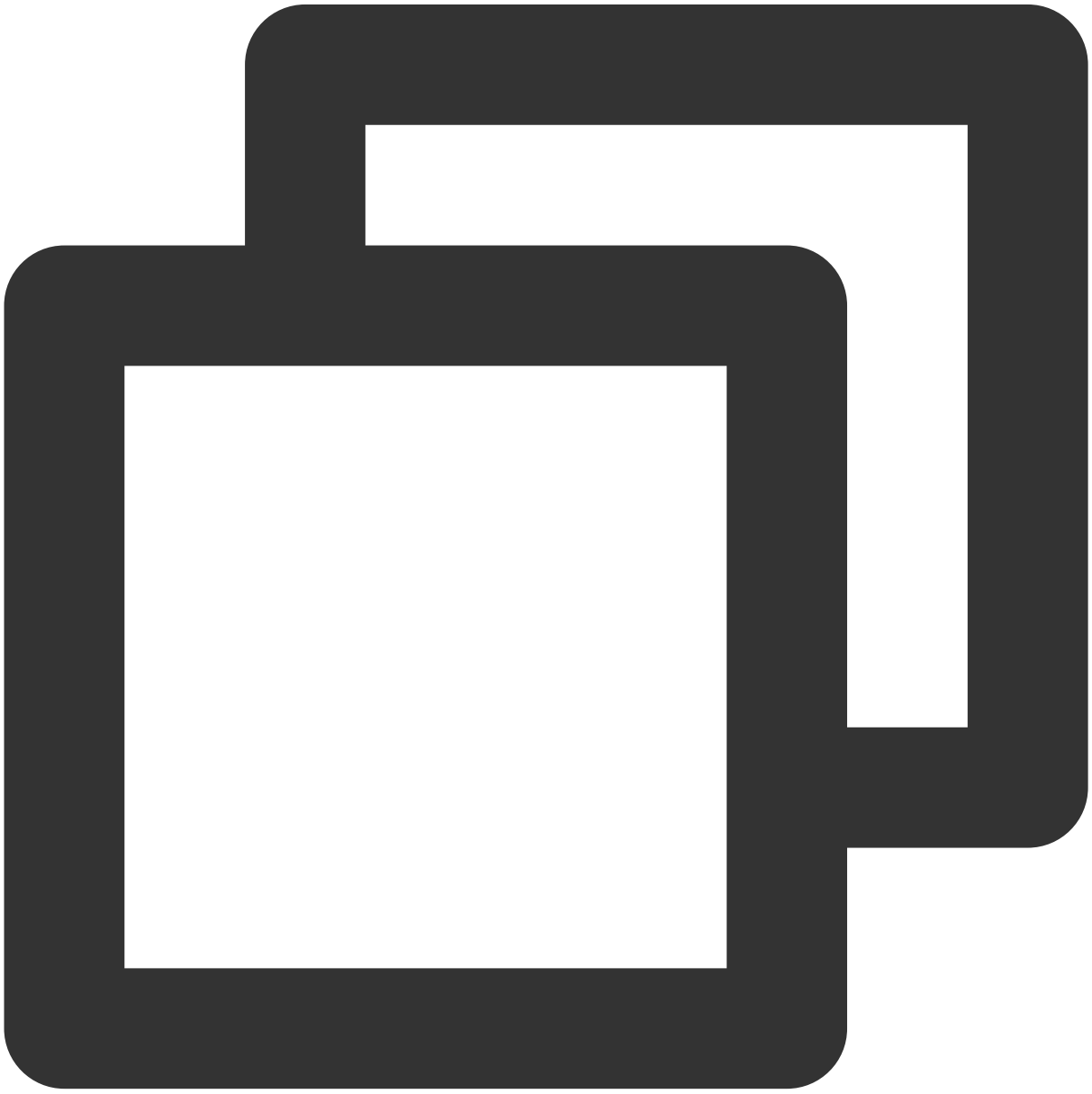
buttonName	FeatureButton	是	按钮名称

setLocalViewBackgroundImage

设置本地用户通话背景。

注意：

v3.2.9+ 支持



```
TUICallKitServer.setLocalViewBackgroundImage(url: string);
```

参数列表：

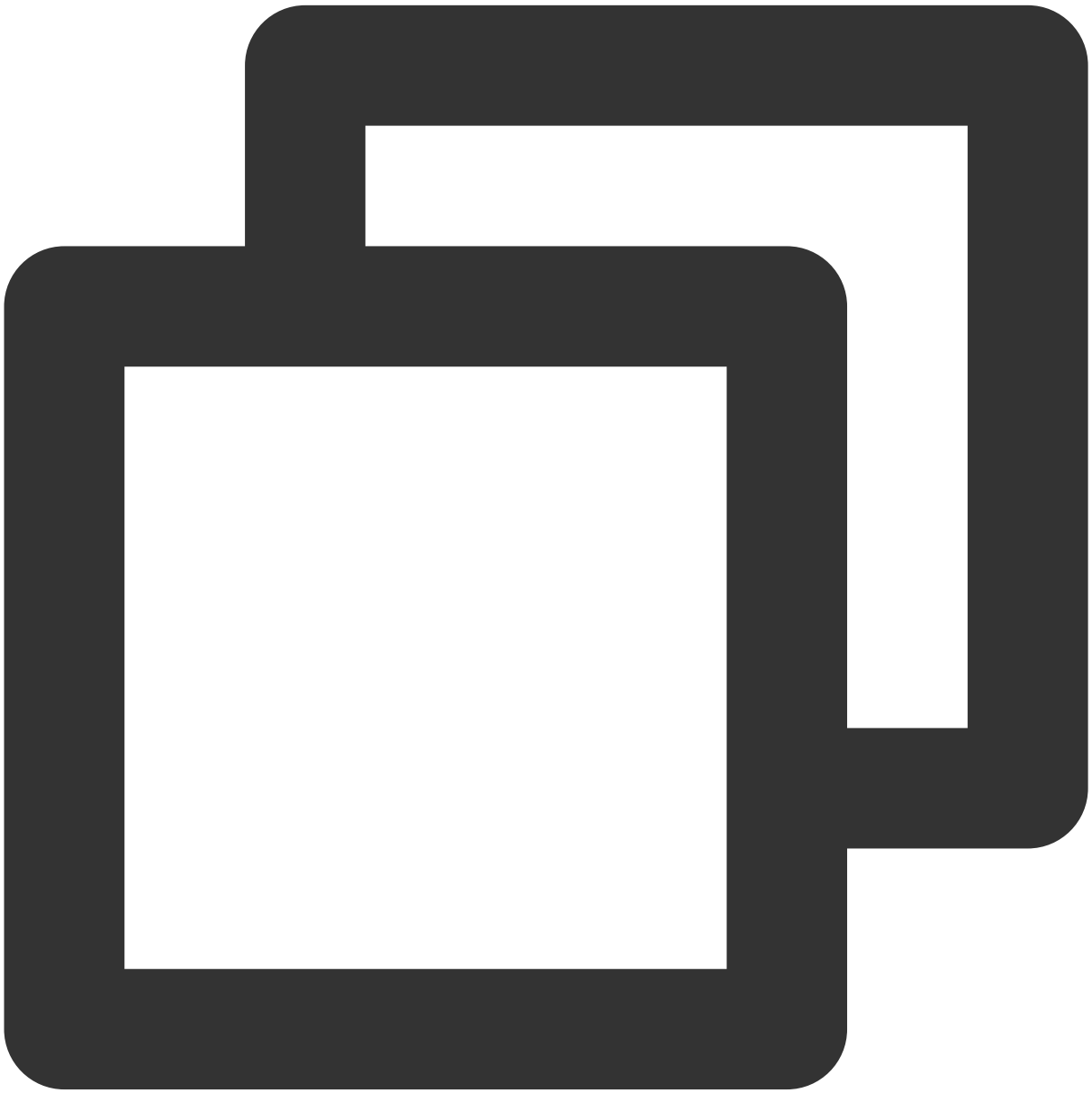
参数	类型	是否必填	含义
url	string	是	图片地址（支持本地路径和网络地址）

setRemoteViewBackgroundImage

设置远端用户通话背景。

注意：

v3.2.9+ 支持



```
TUICallKitServer.setRemoteViewBackgroundImage(userId: string, url: string);
```

参数列表：

参数	类型	是否必填	含义
userId	string	是	远端用户 userId，设置为 '*' 表示对所有远端用户生效
url	string	是	图片地址（支持本地路径和网络地址）

setLayoutMode

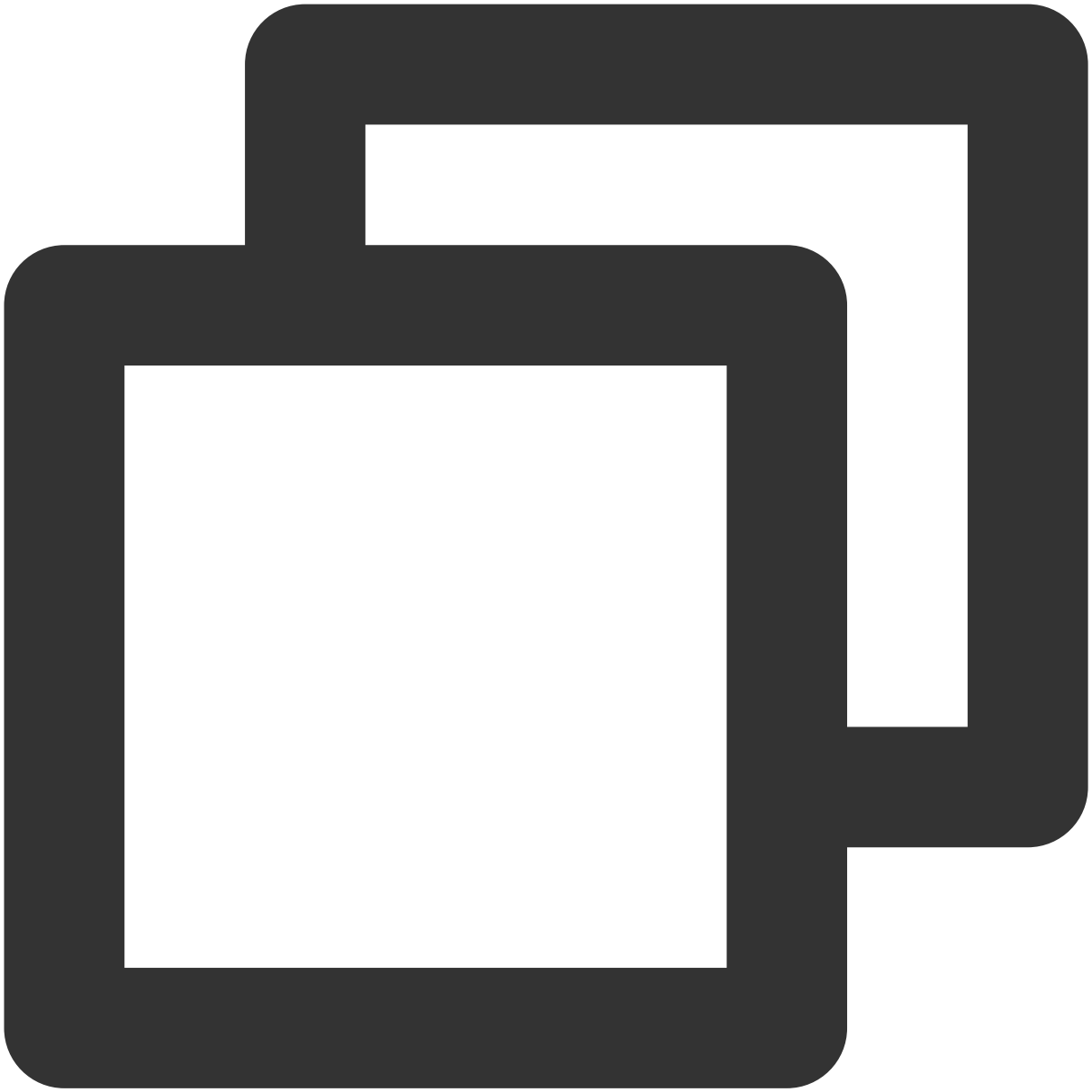
设置通话界面布局模式。

注意：

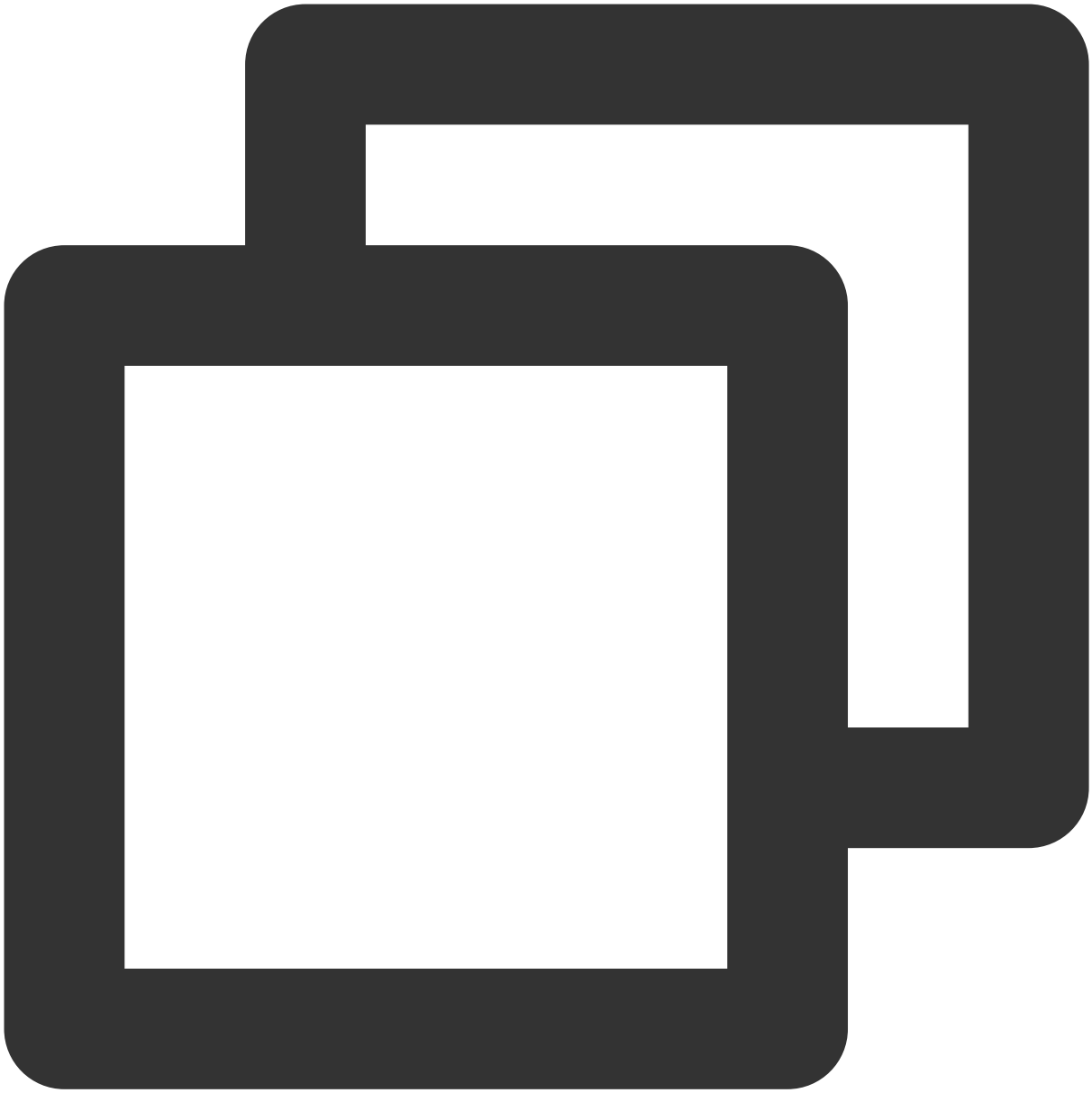
v3.3.0+ 支持。

Vue

React



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-vue";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```



```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-react";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

参数列表：

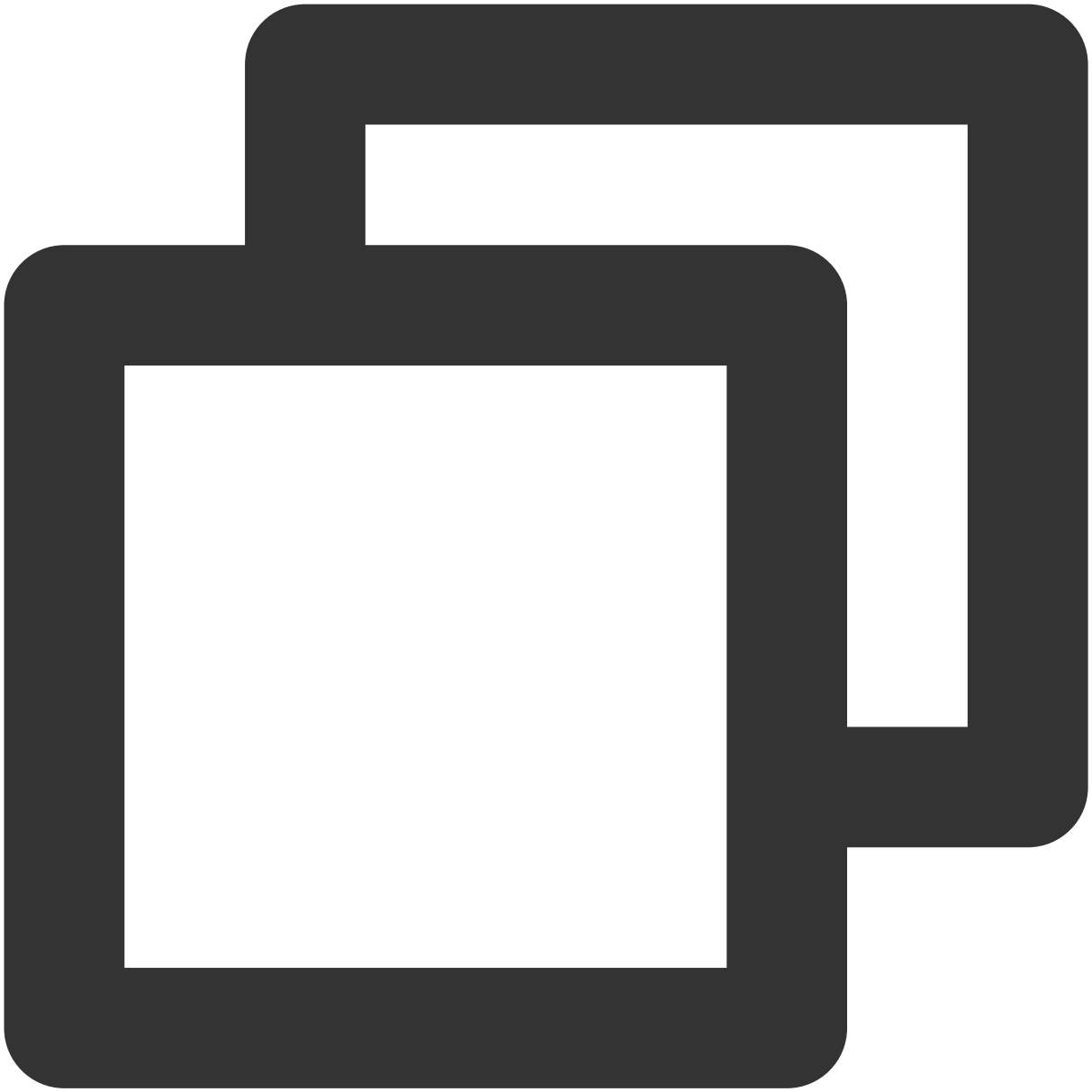
参数	类型	是否必填	含义
layoutMode	LayoutMode	是	用户流的布局模式

setCameraDefaultState

设置摄像头是否默认打开。

注意：

v3.3.0+ 支持。



```
TUICallKitServer.setCameraDefaultState(true);
```

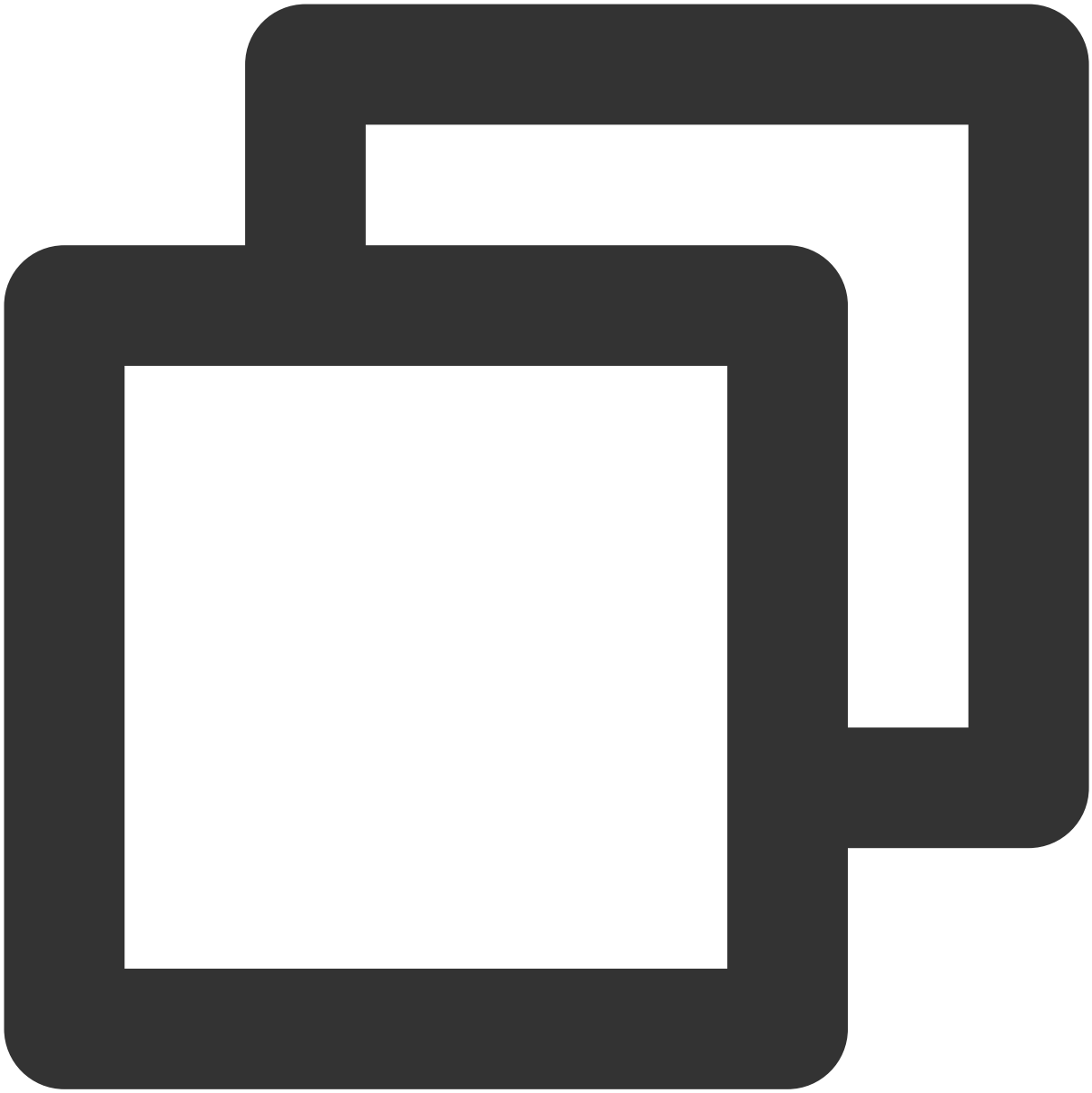
参数列表：

参数	类型	是否必填	含义
isOpen	boolean	是	是否开启摄像头

destroyed

销毁 TUICallKit 实例。

该方法不会自动退出 `tim`，需要手动退出 `tim.logout();`。



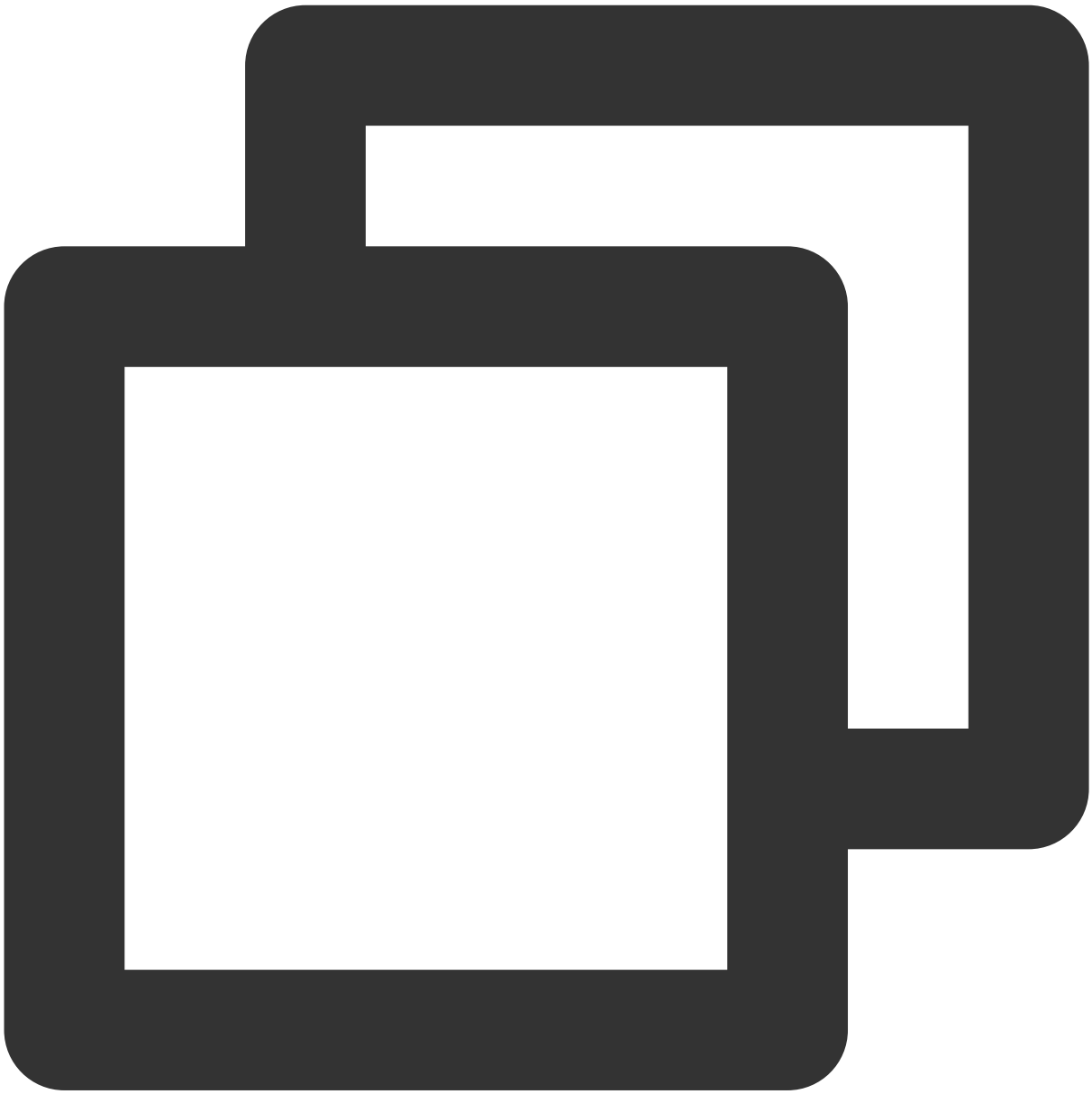
```
try {
  await TUICallKitServer.destroyed();
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the destroyed API. Reason: ${error}`);
}
```

getTUICallEngineInstance

获取 getTUICallEngineInstance 实例。

注意：

v2.4.3+ 支持



```
TUICallKitServer.getTUICallEngineInstance();
```

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。

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

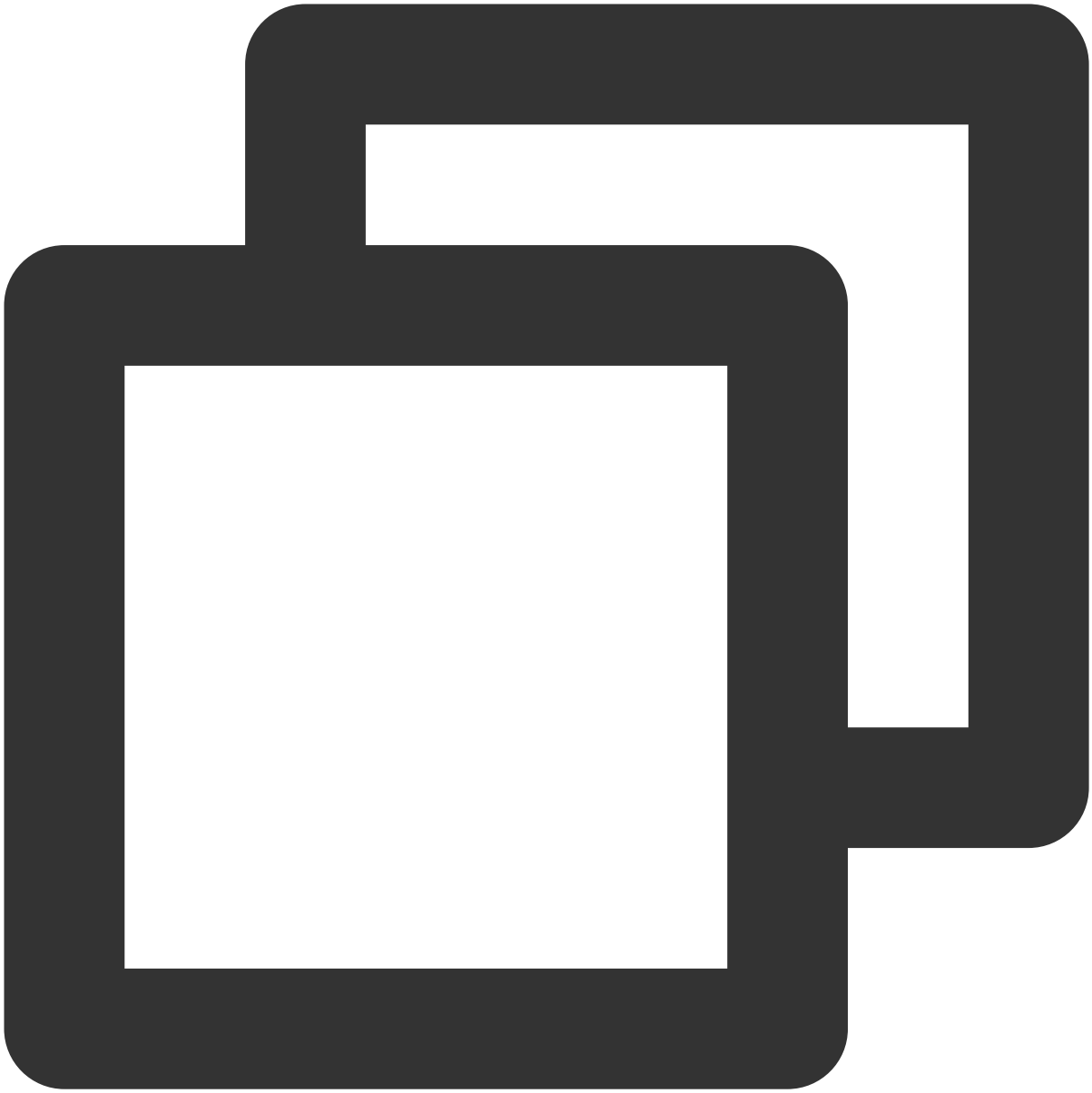
TUICallType 类型	描述
TUICallType.AUDIO_CALL	语音通话
TUICallType.VIDEO_CALL	视频通话

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，可以用于设置 Android Notification 模式 和 VoIP 模式。**默认：**

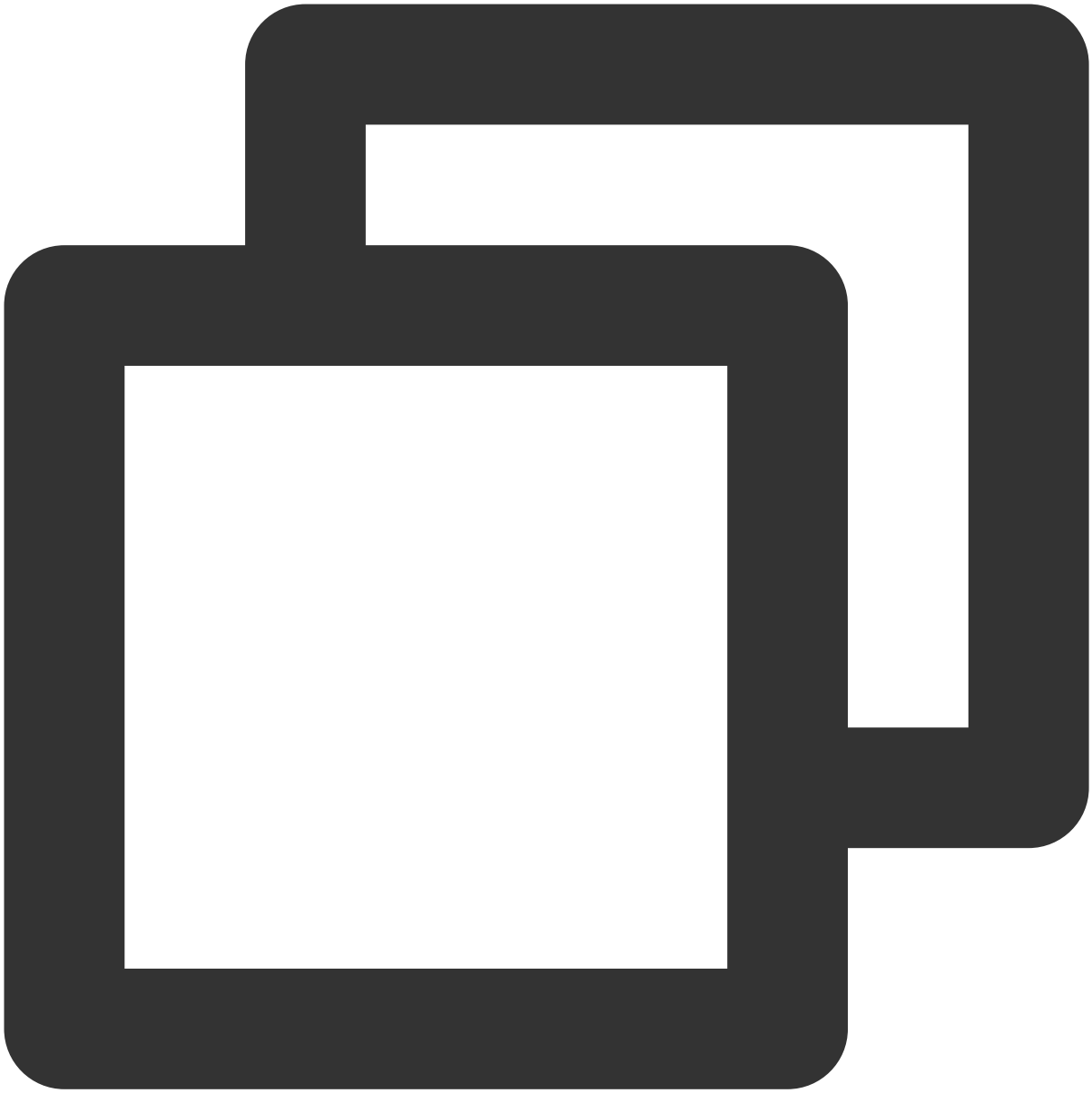
			Notification 模式，会是系统的通知；VoIP 模式是需要传字段。
offlinePushInfo.ignoreIOSBadge	Boolean	否	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加 **v3.3.0+ 支持**
offlinePushInfo.iOSSound	String	否	离线推送声音设置(仅对 iOS 生效)。 **v3.3.0+ 支持**
offlinePushInfo.androidSound	String	否	离线推送声音设置（仅对 Android 生效）。**v3.3.0+ 支持**
offlinePushInfo.androidVIVOClassification	Number	否	VIVO 推送消息分类。**v3.3.0+ 支持**
offlinePushInfo.androidXiaoMiChannelID	String	否	小米手机 8.0 系统及以上的渠道 ID。 **v3.3.0+ 支持**
offlinePushInfo.androidFCMChannelID	String	否	FCM 通道手机 8.0 系统及以上的渠道 ID。 **v3.3.0+ 支持**
offlinePushInfo.androidHuaWeiCategory	String	否	华为推送消息分类。**v3.3.0+ 支持**
offlinePushInfo.isDisablePush	Boolean	否	是否关闭推送（默认开启推送）。**v3.3.0+ 支持**
offlinePushInfo.iOSPushType	Number	否	iOS 离线推送类型。0-普通推送；1-VoIP 推送。默认：0。**v3.3.0+ 支持**

Android Notification 模式



```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, VoIP; 1, notification
  }
};
offlinePushInfo.extension = JSON.stringify(extension);
```

Android VoIP 模式



```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, data; 1, notification
    fcmNotificationType: 1, // 0, TIMPush implementation; 1, business implementatio
  }
};
offlinePushInfo.extension = JSON.stringify(extension);
```

FeatureButton

FeatureButton 类型	描述
FeatureButton.Camera	摄像头按钮
FeatureButton.Microphone	麦克风按钮
FeatureButton.SwitchCamera	切换前后置摄像头按钮
FeatureButton.InviteUser	邀请他人按钮

LayoutMode

LayoutMode 类型	描述
LayoutMode.LocalInLargeView	本地用户在大窗显示
LayoutMode.RemoteInLargeView	远端用户在大窗显示

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2024-05-24 17:39:32

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	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
switchCamera	切花前后置摄像头，**注意：仅移动端支持。v3.0.0+ 支持**
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
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式。**v2.1.1+ 支持**
setBlurBackground	开关/设置背景虚化，**v3.0.6+ 支持**
setVirtualBackground	开关/设置图片背景虚化，**v3.0.6+ 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：2024-06-28 14:38:07

TUICallEvent API 简介

TUICallEvent API 是音视频通话组件的**事件接口**。

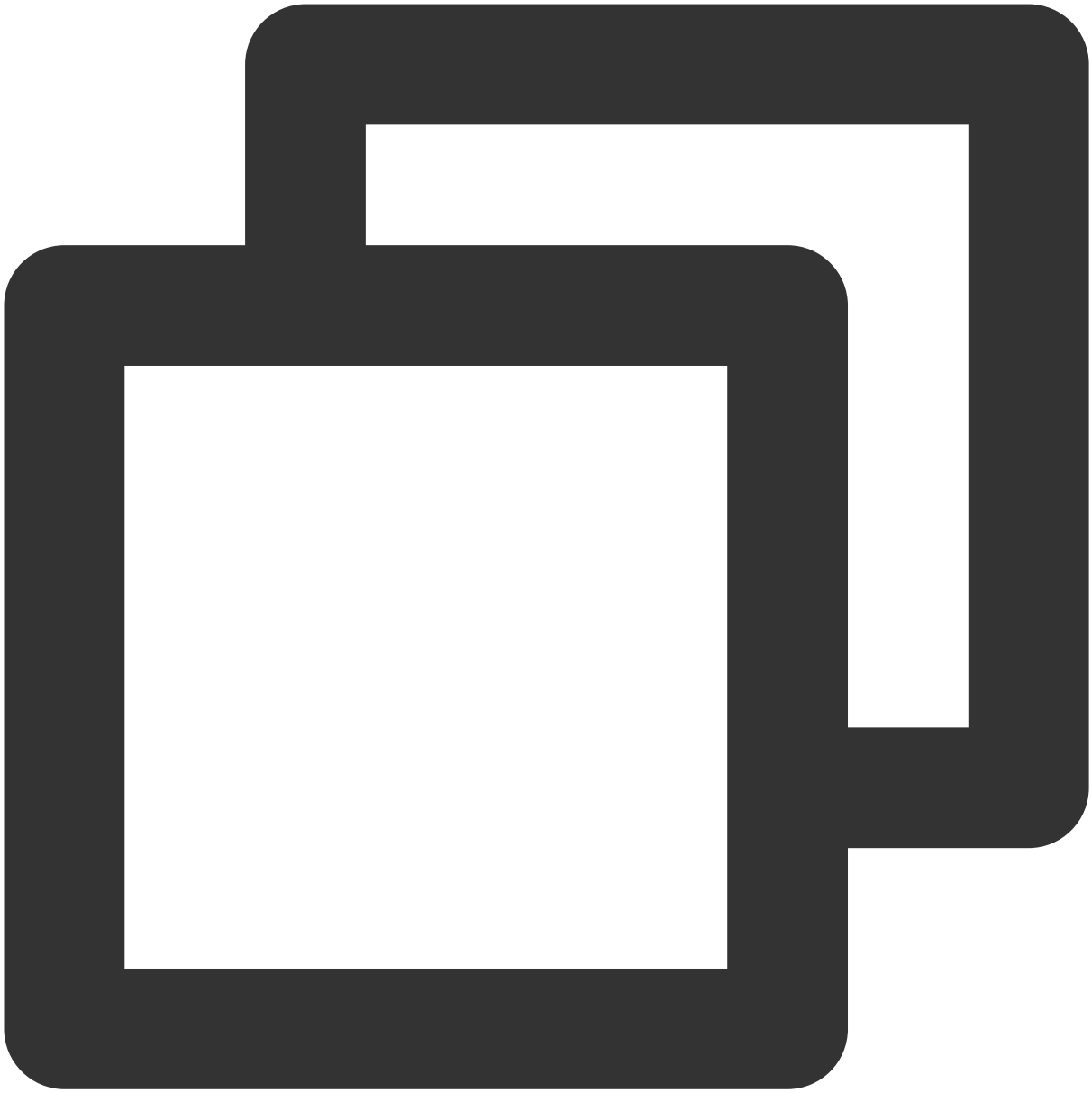
事件列表

EVENT	描述
TUICallEvent.ERROR	通话过程中错误事件
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该事件
TUICallEvent.KICKED_OUT	用户重复登录，被踢下线
TUICallEvent.USER_ACCEPT	xxxx 用户接通的事件
TUICallEvent.USER_ENTER	xxxx 用户加入通话的事件
TUICallEvent.USER_LEAVE	xxxx 用户离开通话的事件
TUICallEvent.REJECT	xxxx 用户拒绝通话的事件
TUICallEvent.NO_RESP	xxxx 用户不响应的事件
TUICallEvent.LINE_BUSY	xxxx 用户忙线的事件
TUICallEvent.USER_VIDEO_AVAILABLE	xxxx 用户是否有视频流的事件
TUICallEvent.USER_AUDIO_AVAILABLE	xxxx 用户是否有音频流的事件
TUICallEvent.USER_VOICE_VOLUME	所有用户音量大小的反馈事件
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该事件
TUICallEvent.ON_CALL_BEGIN	通话接通的事件，**v1.4.6+ 支持**
TUICallEvent.INVITED	通话请求的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_RECEIVED**

TUICallEvent.ON_CALL_RECEIVED	通话请求的事件，**v1.4.6+ 支持**
TUICallEvent.CALLING_CANCEL	通话取消的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_CANCELED	通话取消的事件，**v1.4.6+ 支持**
TUICallEvent.ON_CALL_BEGIN	通话接通时抛出的事件
TUICallEvent.CALLING_END	通话结束的事件
TUICallEvent.DEVICED_UPDATED	设备列表更新的事件
TUICallEvent.CALL_TYPE_CHANGED	通话的媒体类型发生改变的回调
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件，**v3.0.7+ 支持**

ERROR

通话过程中错误事件，可以通过监听该事件，捕获通话过程中的内部错误。



```
let onError = function(error) {
  console.log(error.code, error.msg);
};
tuiCallEngine.on(TUICallEvent.ERROR, onError);
```

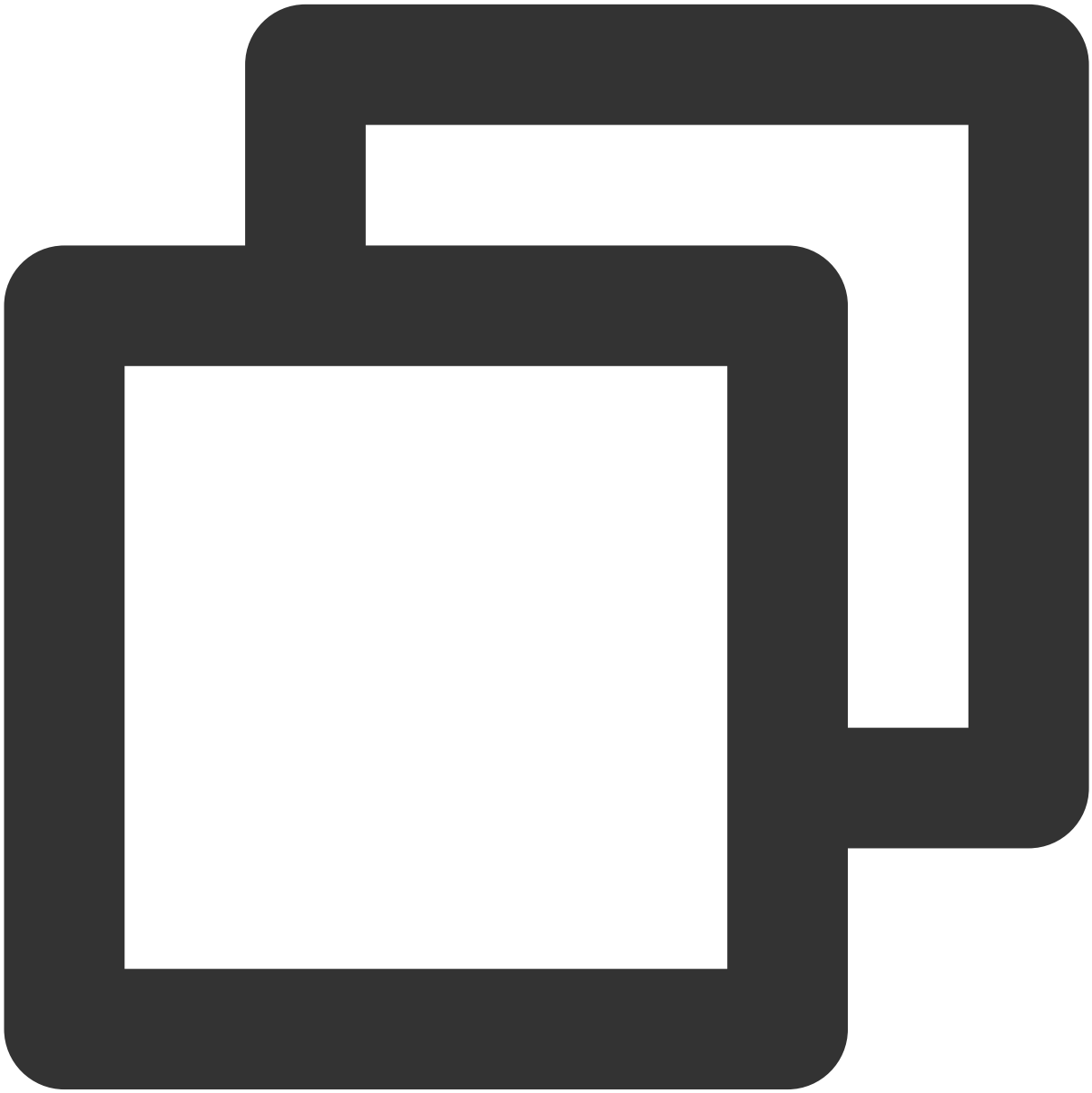
参数如下表所示：

参数	类型	含义
code	Number	错误码

msg	String	错误信息

SDK_READY

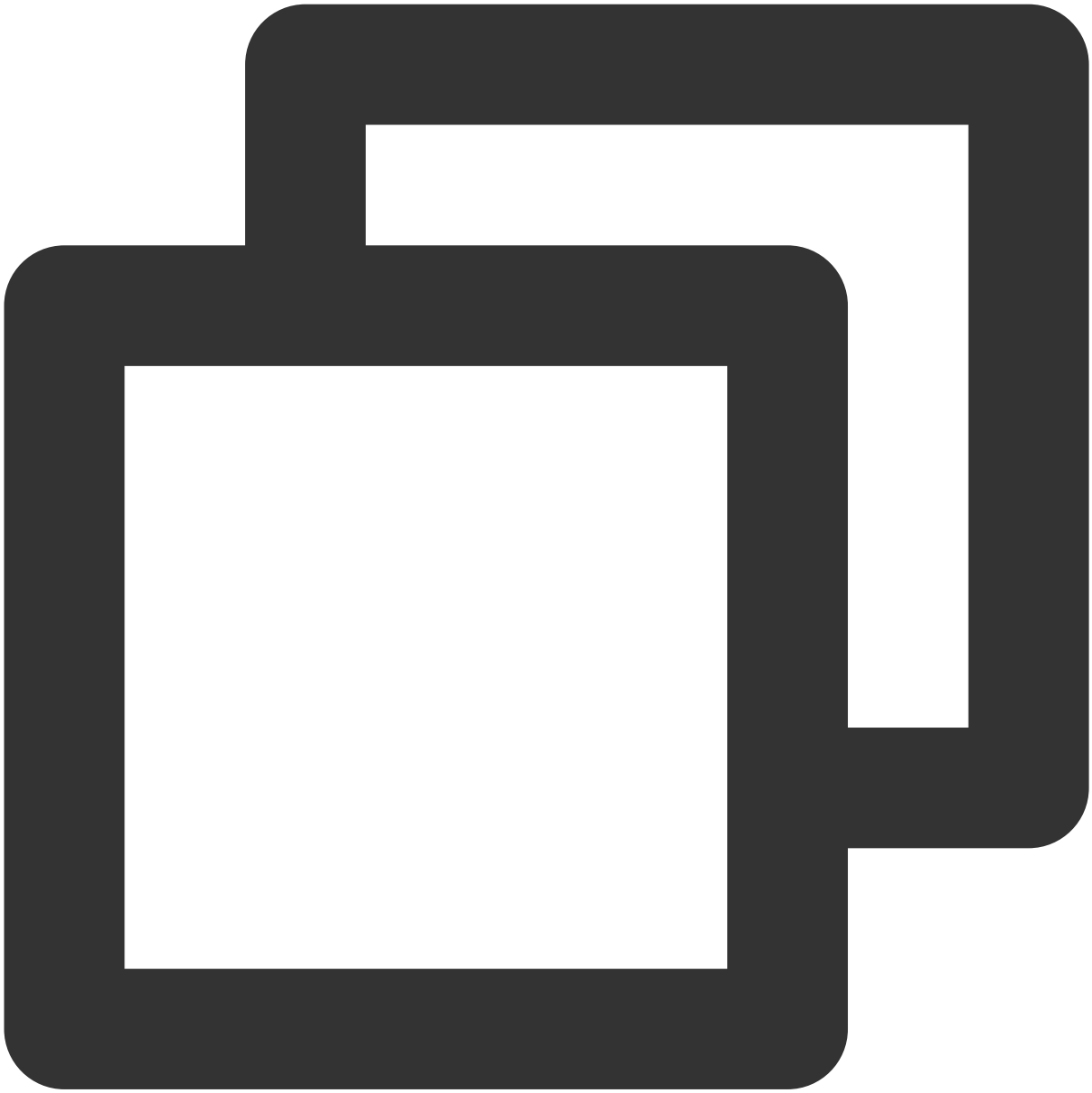
TUICallEngine 依赖 @tencentcloud/chat SDK，登录成功后才会出发 SDK_READY 事件，然后才能使用 SDK 的各项功能。



```
let onSDKReady = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.SDK_READY, onSDKReady);
```

KICKED_OUT

当前用户被踢下线，此时可以 UI 提示用户，并再次重新调用登录。



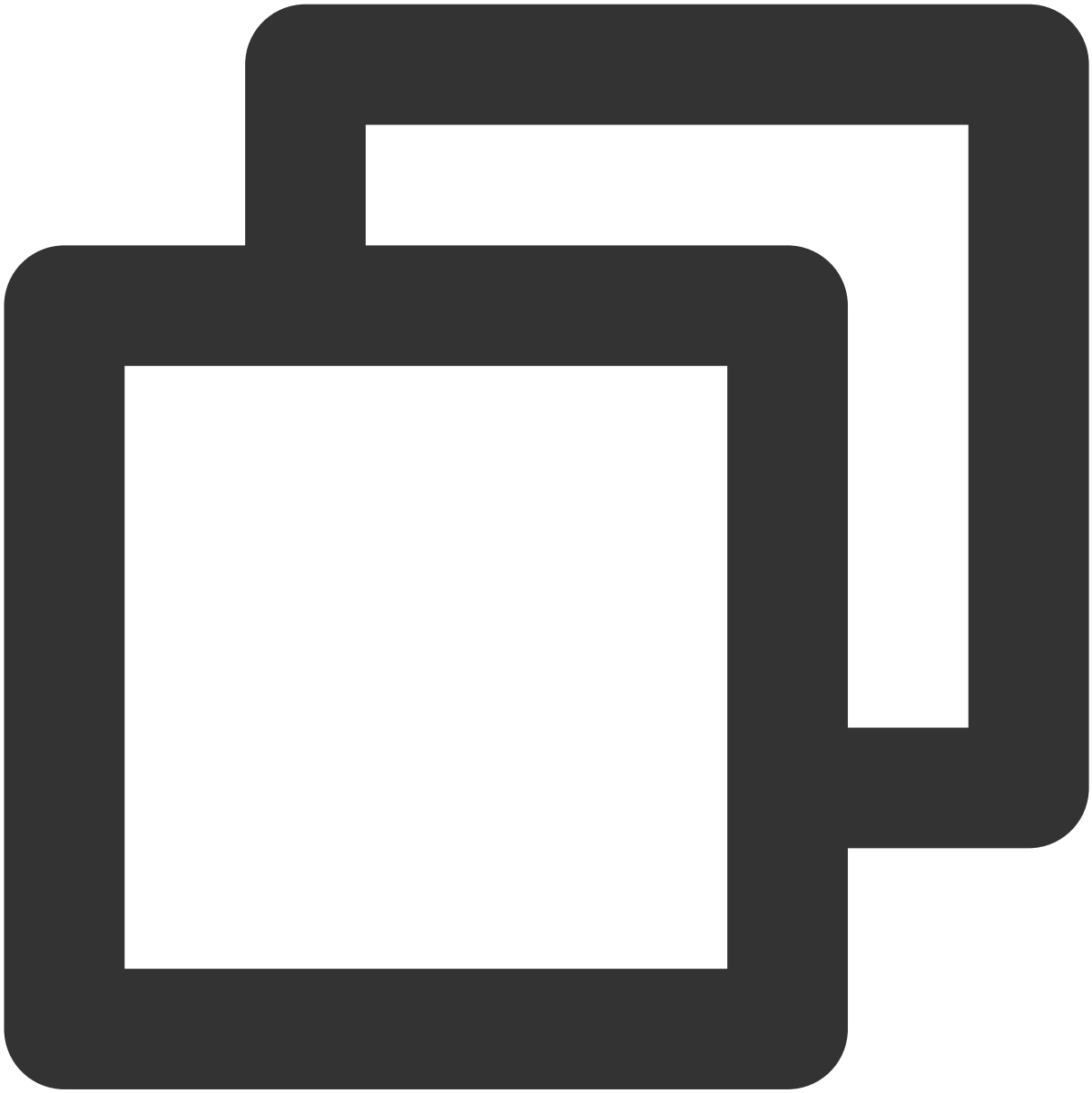
```
let handleOnKickedOut = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.KICKED_OUT, handleOnKickedOut);
```

USER_ACCEPT

如果有用户接通，其他用户都会抛出该事件，userID 就是接通的用户名。

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。



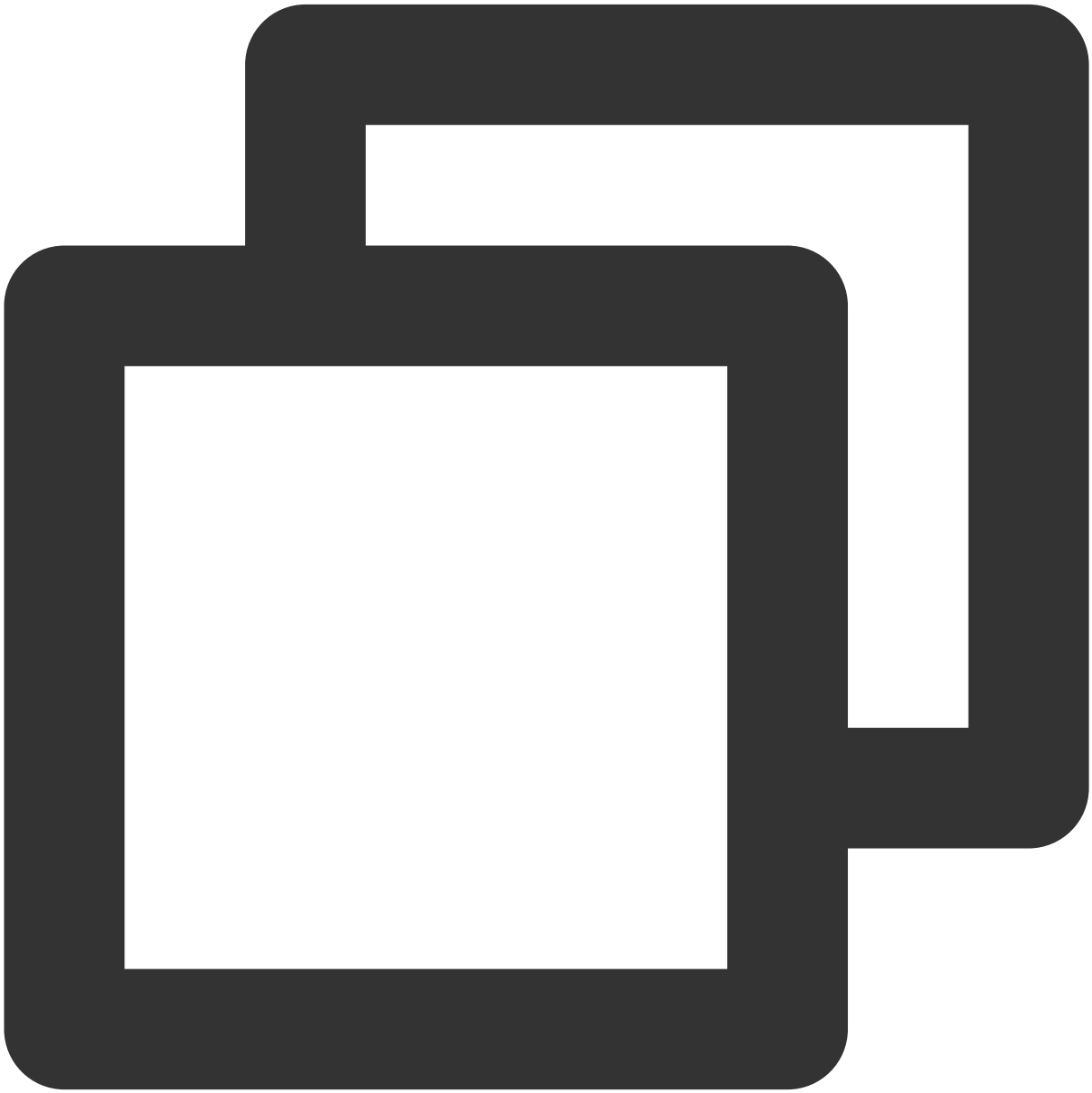
```
let handleUserAccept = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, handleUserAccept);
```

参数如下表所示：

参数	类型	含义
userID	String	接通通话的用户 ID

USER_ENTER

如果有用户进入通话，其他用户都会抛出该事件，userID 就是进入通话的用户名。



```
let handleUserEnter = function(event) {
  console.log(event.userID);
};
```

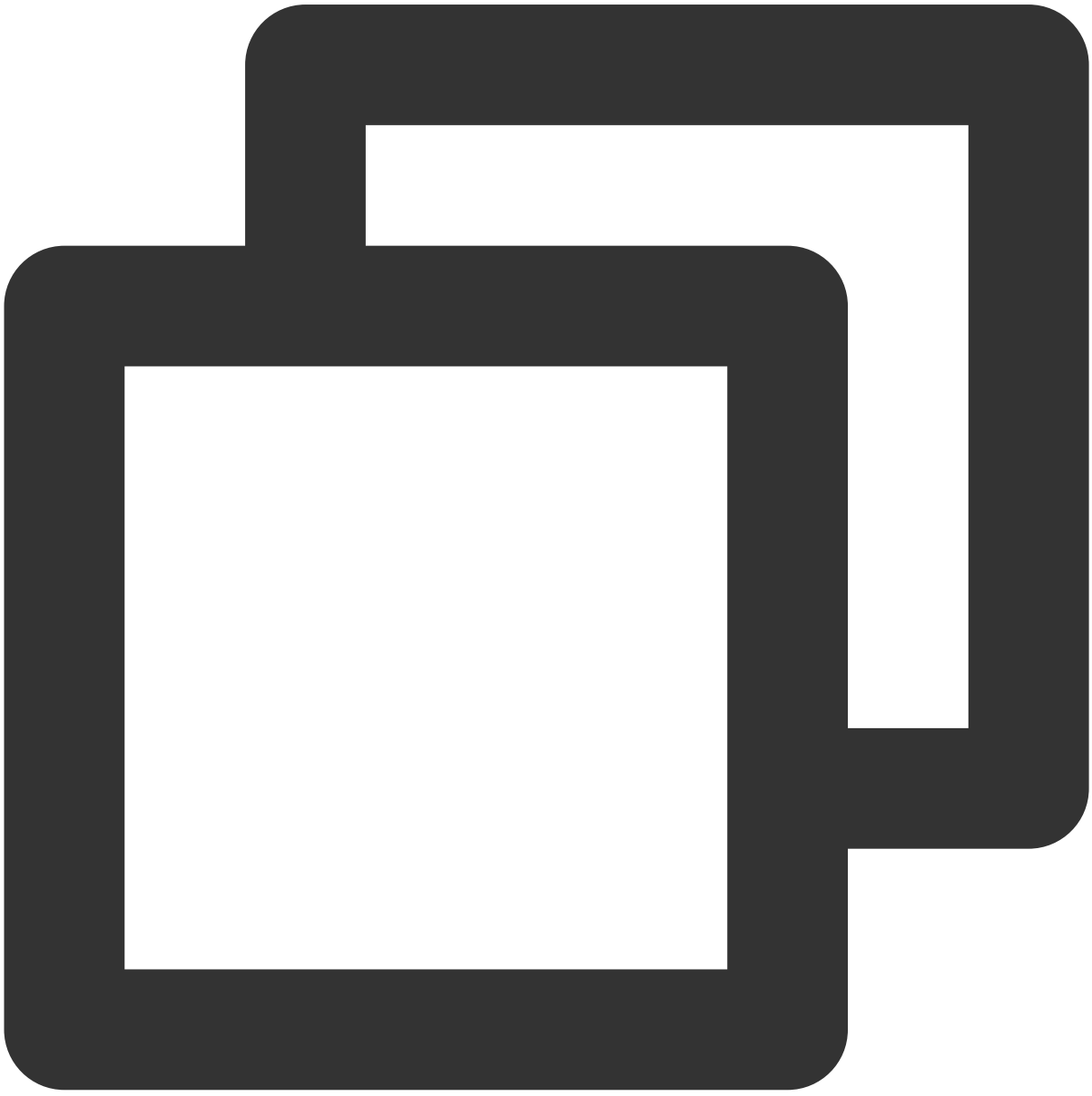
```
tuiCallEngine.on(TUICallEvent.USER_ENTER, handleUserEnter);
```

参数如下表所示：

参数	类型	含义
userID	String	进入通话的用户 ID

USER_LEAVE

有用户离开此次通话时，**通话中**的其他用户都会抛出该事件，userID 就是离开通话的用户名。



```
let handleUserLeave = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_LEAVE, handleUserLeave);
```

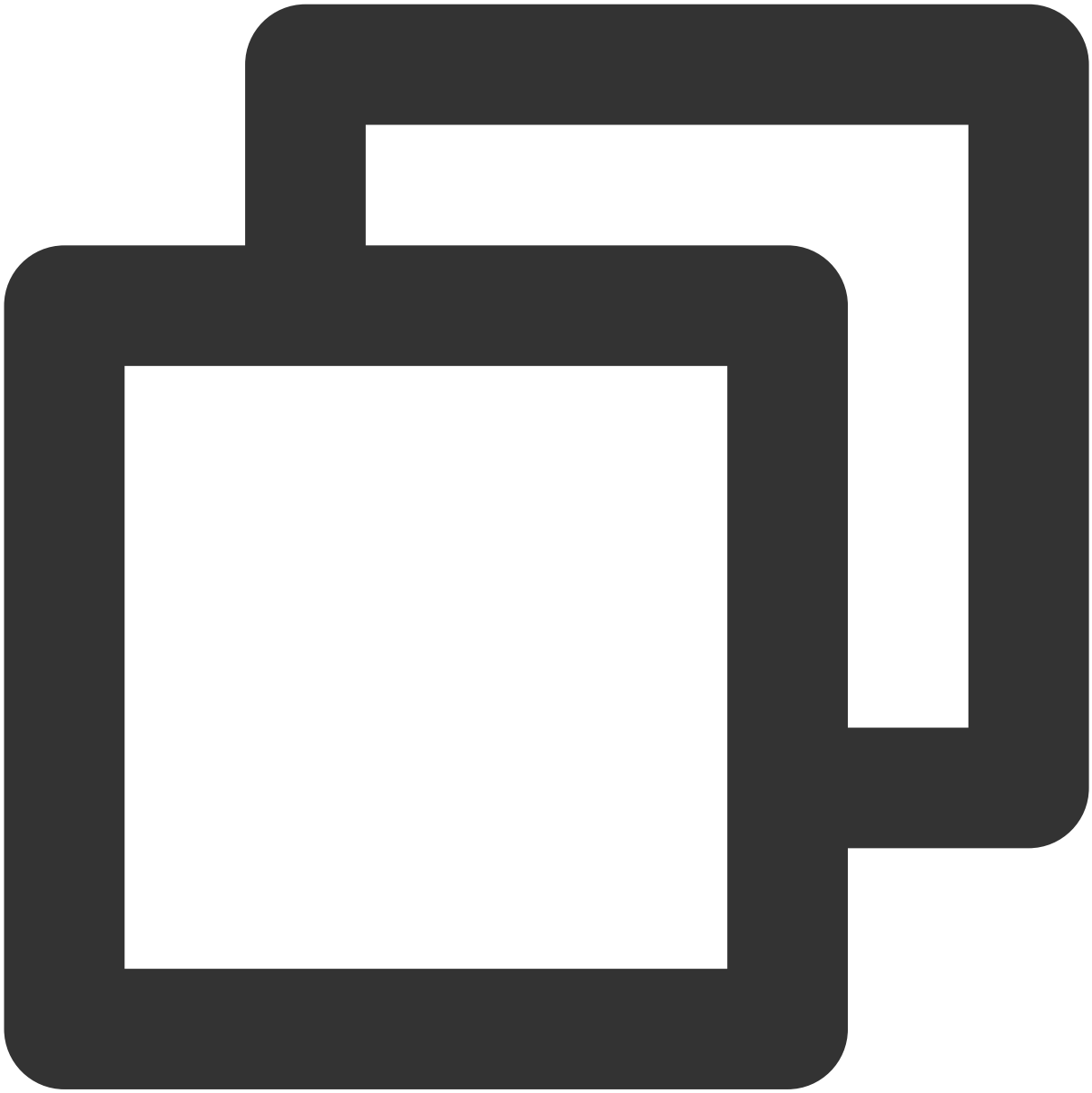
参数如下表所示：

参数	类型	含义
userID	String	离开当前通话的用户 ID

REJECT

通话被拒绝时抛出该事件

1. 在 1v1 通话中，只有主叫方会收到拒绝事件，userID 就是被叫用户名。
2. 在群组通话中，当有被邀请者拒绝通话时，群组通话的其他人都会抛出该事件，userID 就是拒绝通话的用户名。



```
let handleInviteeReject = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.REJECT, handleInviteeReject);
```

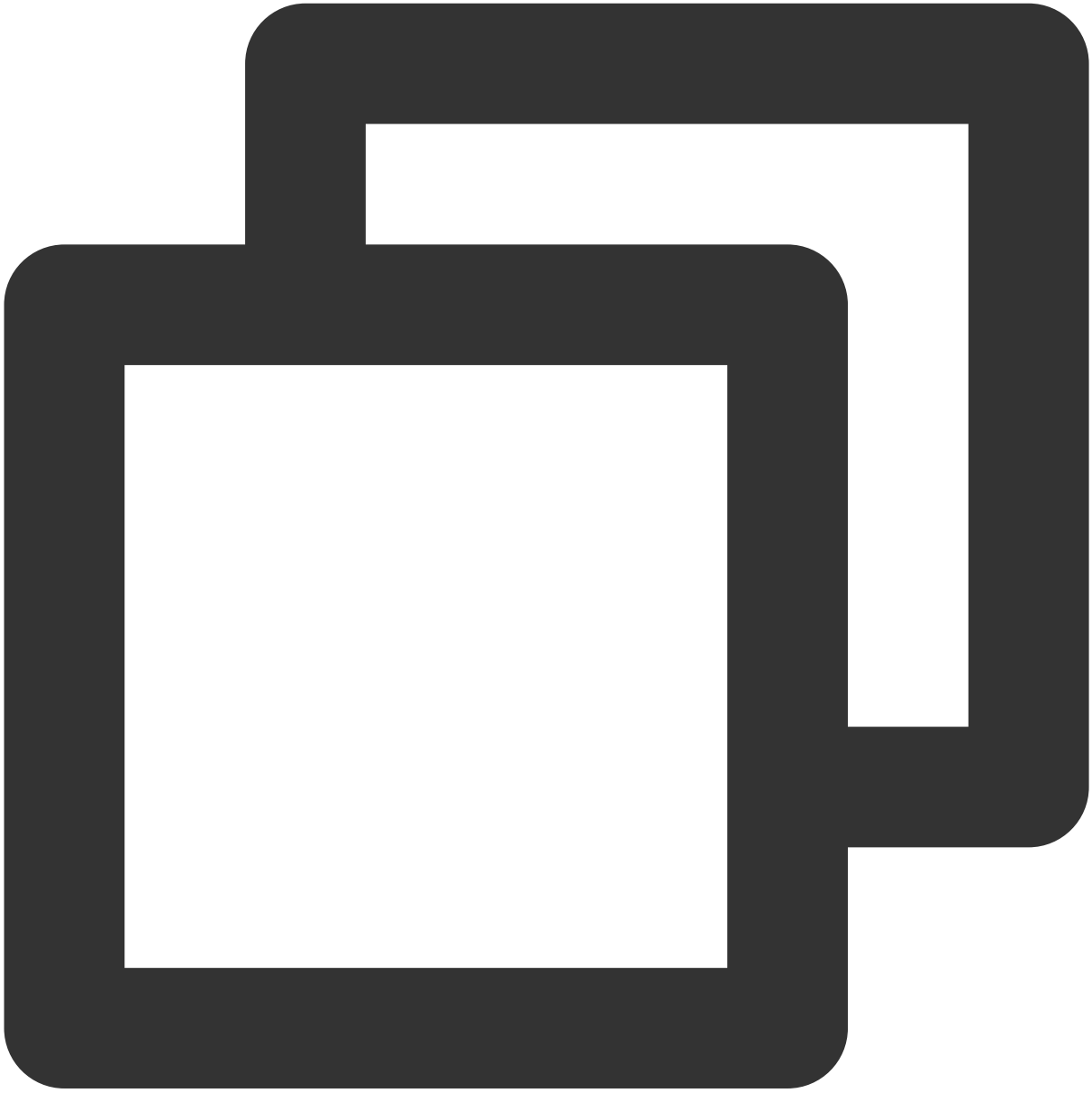
参数如下表所示：

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

被呼叫者没有回应时，其他通话用户都会抛出该事件。
在 1v1 通话中，只有主叫方会抛出无人应答的事件。例如 A 邀请 B，B 不应答，A 抛出该事件。
在群组通话中，当有被邀请者不回应时，群组通话的其他人都会抛出该事件。例如 A 邀请 B、C 进入通话，B 不应答，A、C 均会抛出该事件。



```
let handleNoResponse = function(event) {
console.log(event.sponsor, event.userIDList);
};
tuiCallEngine.on(TUICallEvent.NO_RESP, handleNoResponse);
```

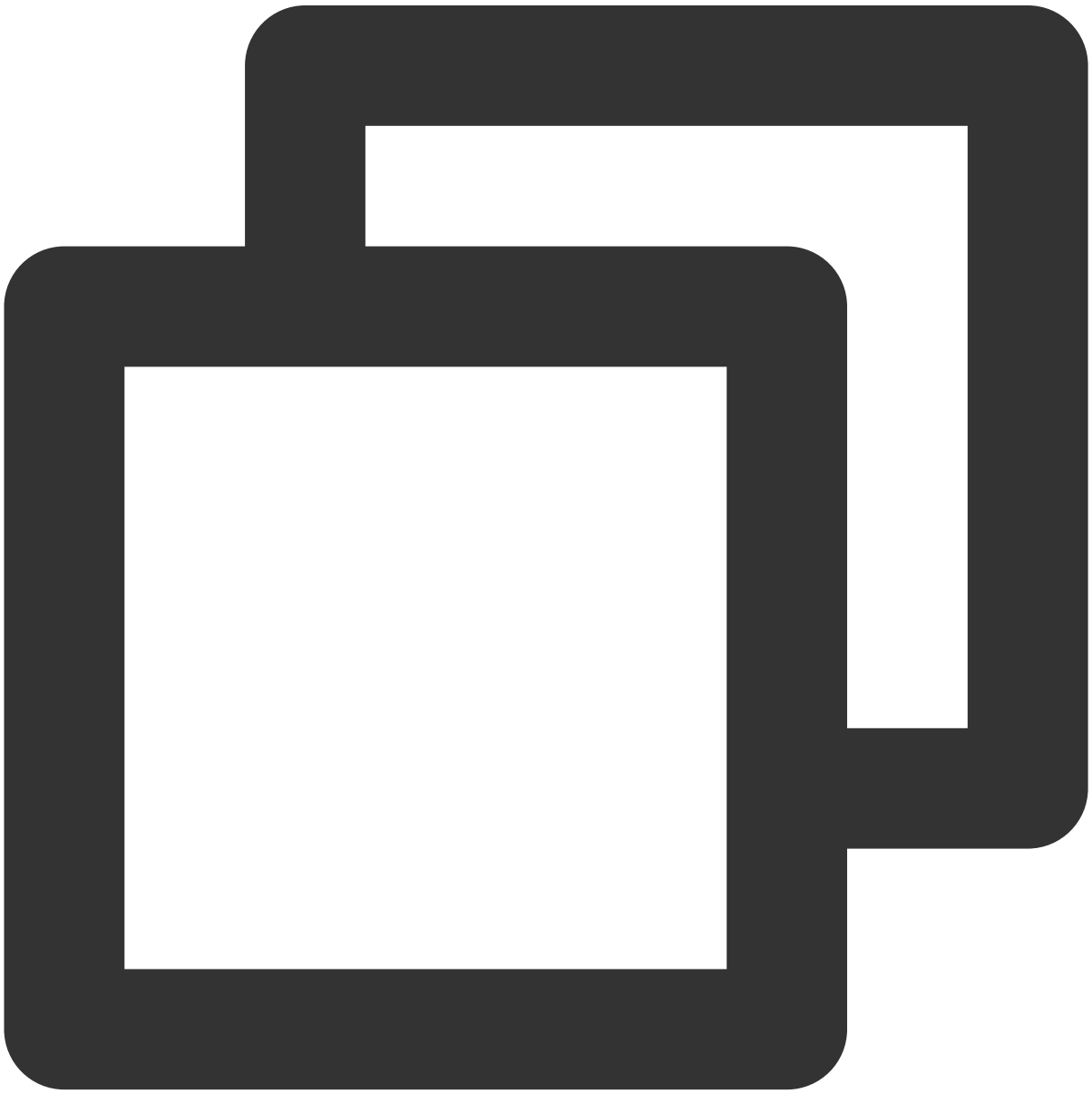
版权所有：腾讯云计算（北京）有限责任公司

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>	无响应的用户 ID 列表

LINE_BUSY

通话忙线事件。例如：B 正在通话时，此时 A 呼叫 B，A 会抛出事件。



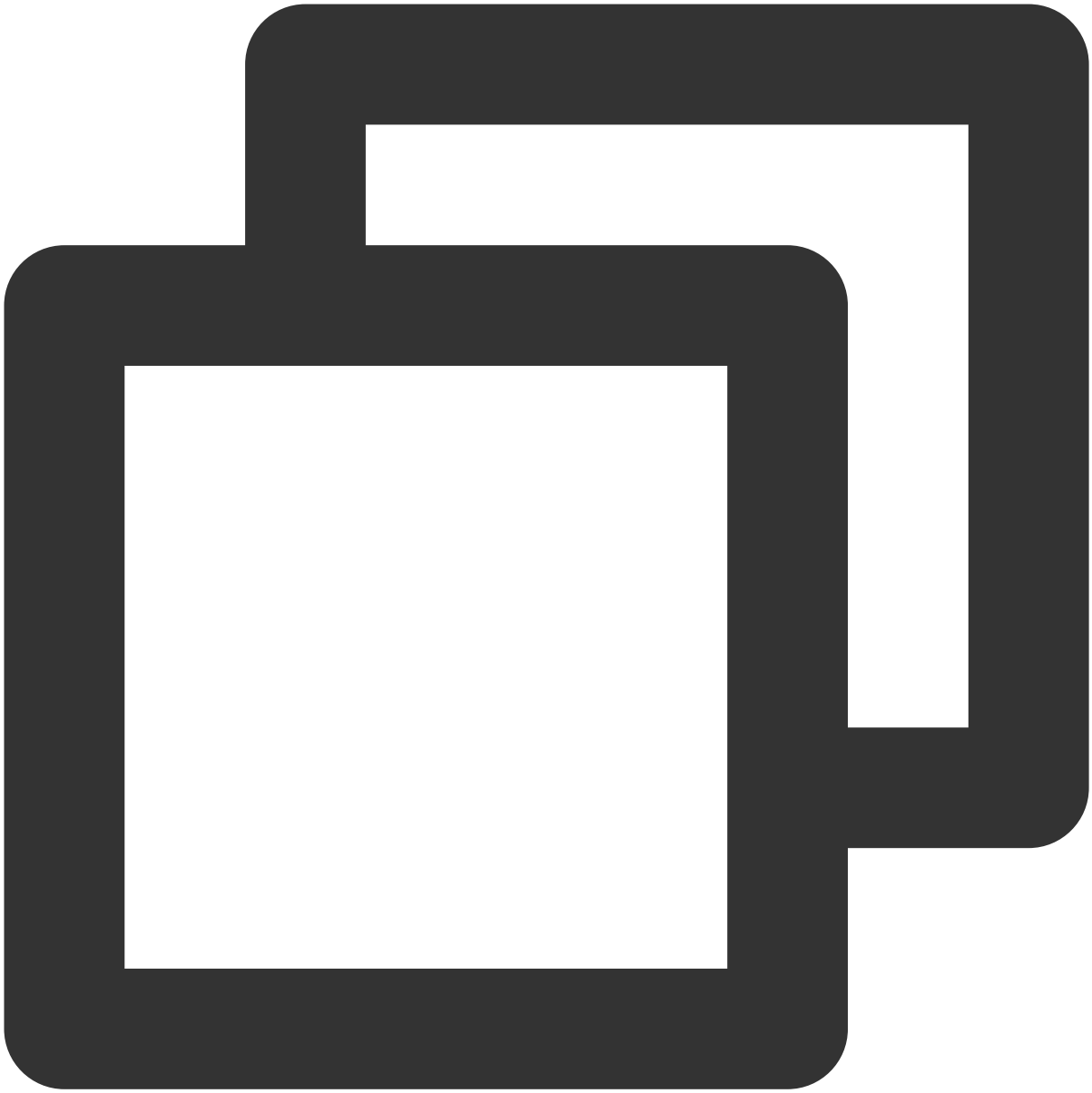
```
let handleLineBusy = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.LINE_BUSY, handleLineBusy);
```

参数如下表所示：

参数	类型	含义
userID	String	忙线用户 ID

USER_VIDEO_AVAILABLE

视频通话中一个用户开启/关闭了摄像头, 通话的其他用户都会抛出该事件。例如：A 和 B 在进行视频通话，A 开/关摄像头，B 会抛出该事件。



```
let handleUserVideoChange = function(event) {
  console.log(event.userID, event.isVideoAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

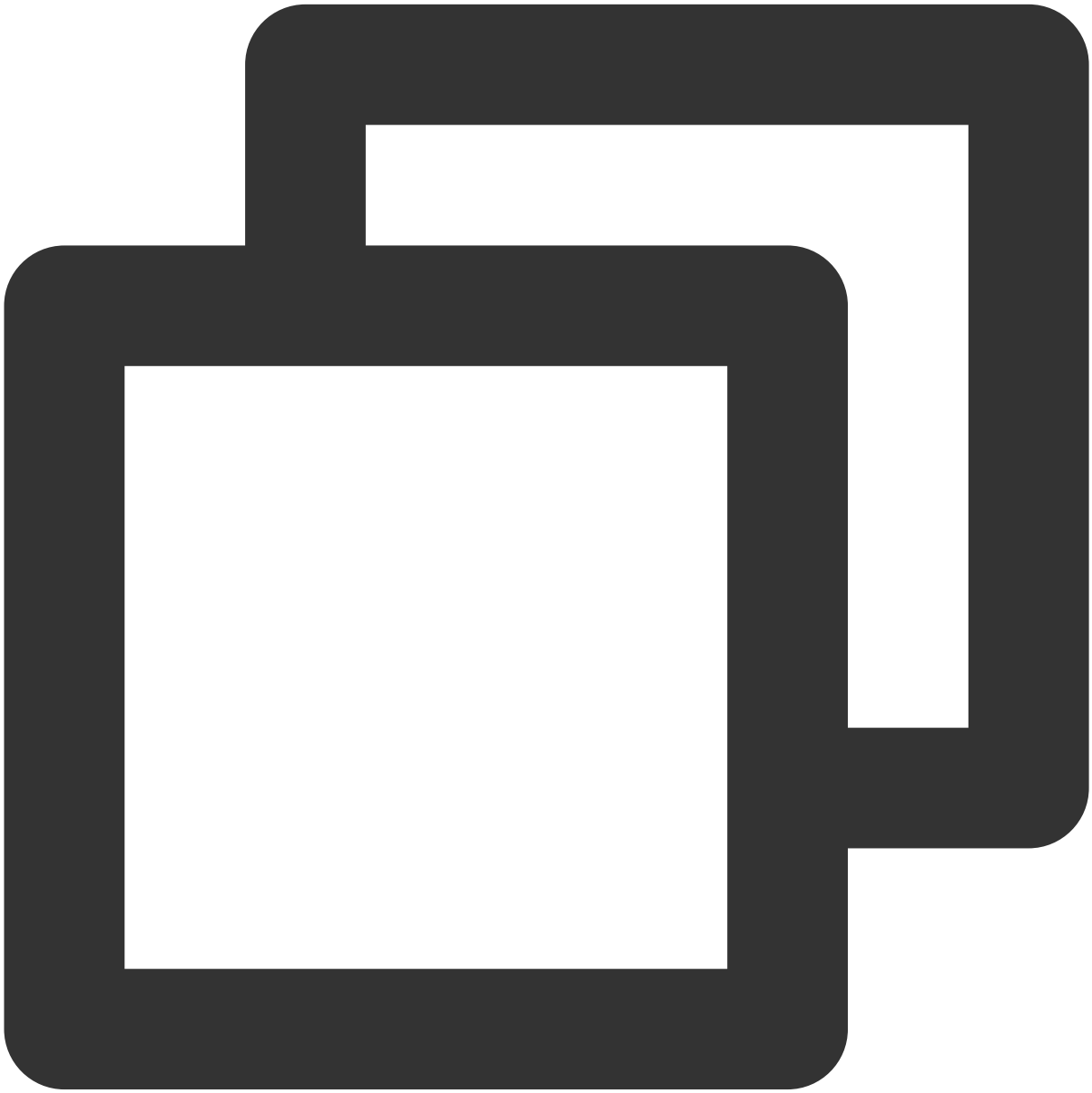
参数如下表所示：

参数	类型	含义
userID	String	开/关摄像头的用户 ID

isVideoAvailable	Boolean	true 用户打开摄像头；false 用户关闭摄像头

USER_AUDIO_AVAILABLE

音视频通话中一个用户开启/关闭了麦克风, 通话的其他用户都会抛出该事件。例如：A 和 B 在进行音视频通话，A 开/关麦克风，B 会抛出该事件。



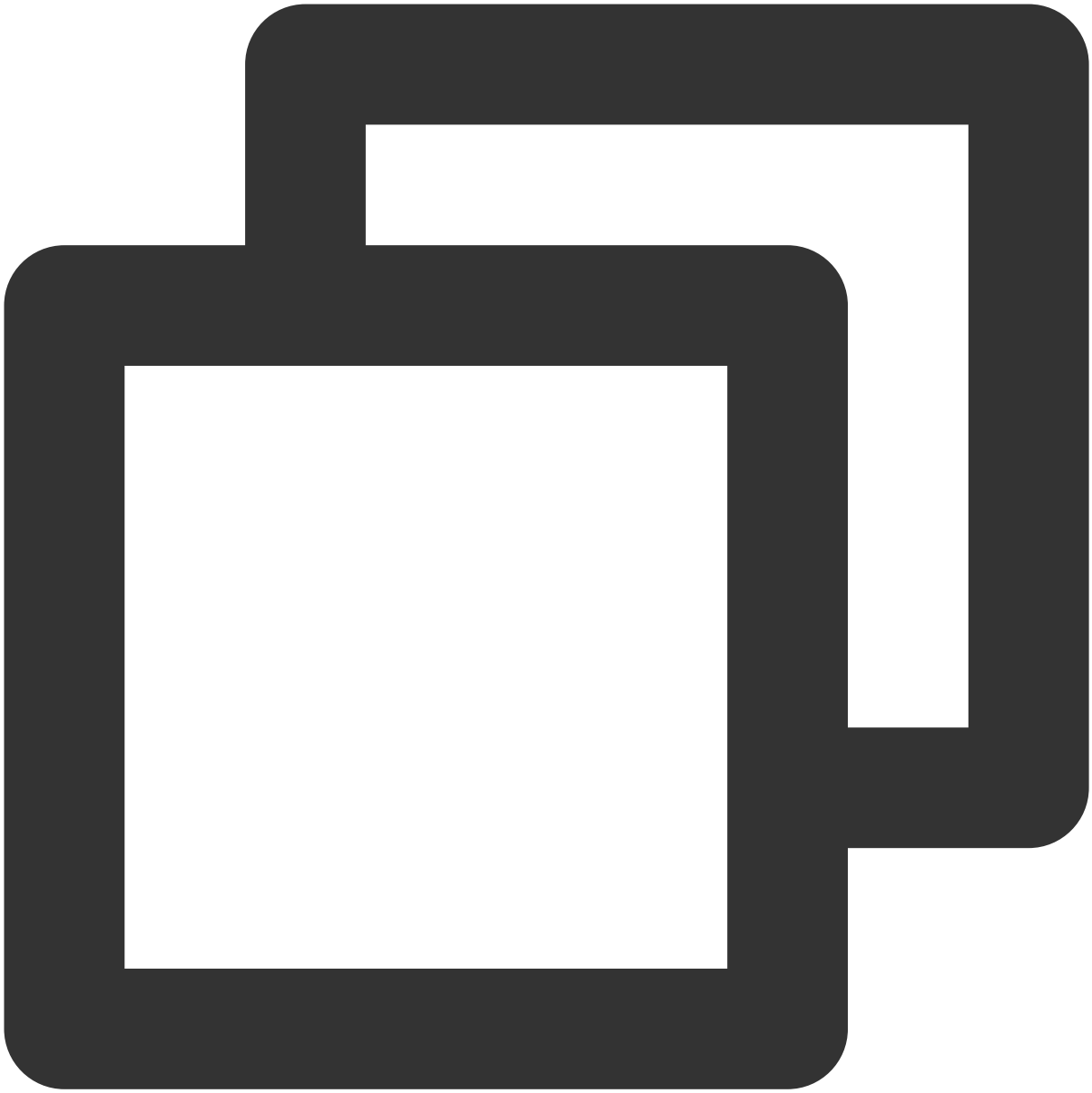
```
let handleUserAudioChange = function(event) {
  console.log(event.userID, event.isAudioAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

参数如下表所示：

参数	类型	含义
userID	String	开/关麦克风的用户 ID
isAudioAvailable	Boolean	true 用户打开麦克风；false 用户关闭麦克风

USER_VOICE_VOLUME

音视频通话中用户音量变化时，通话的其他用户都会抛出该事件。例如：A 和 B 在进行音视频通话，A 音量变化，B 会抛出该事件。



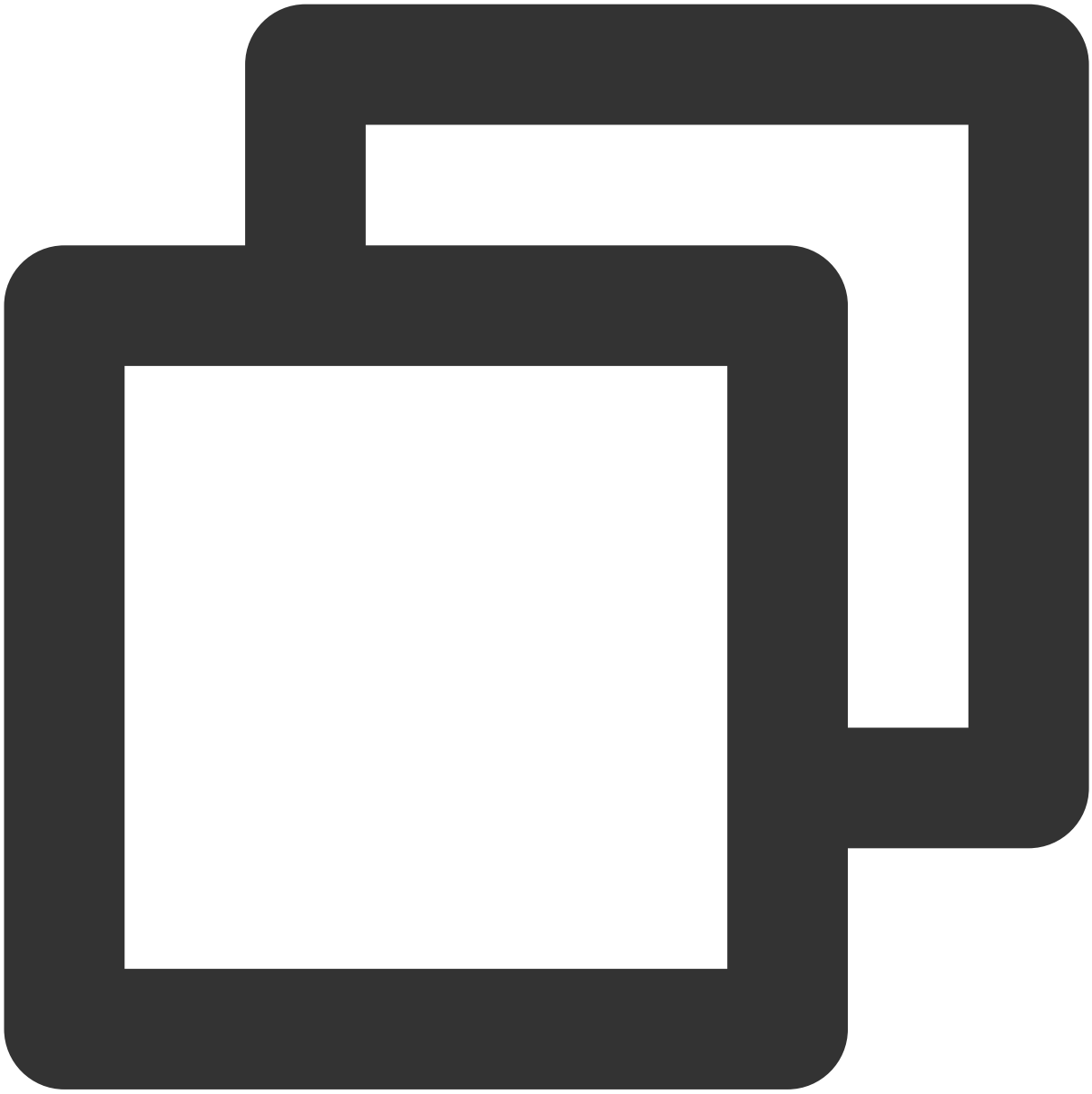
```
let handleUserVoiceVolumeChange = function(event) {
  console.log(event.volumeMap);
};
tuiCallEngine.on(TUICallEvent.USER_VOICE_VOLUME, handleUserVoiceVolumeChange);
```

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，音量范围：[0，100]

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：

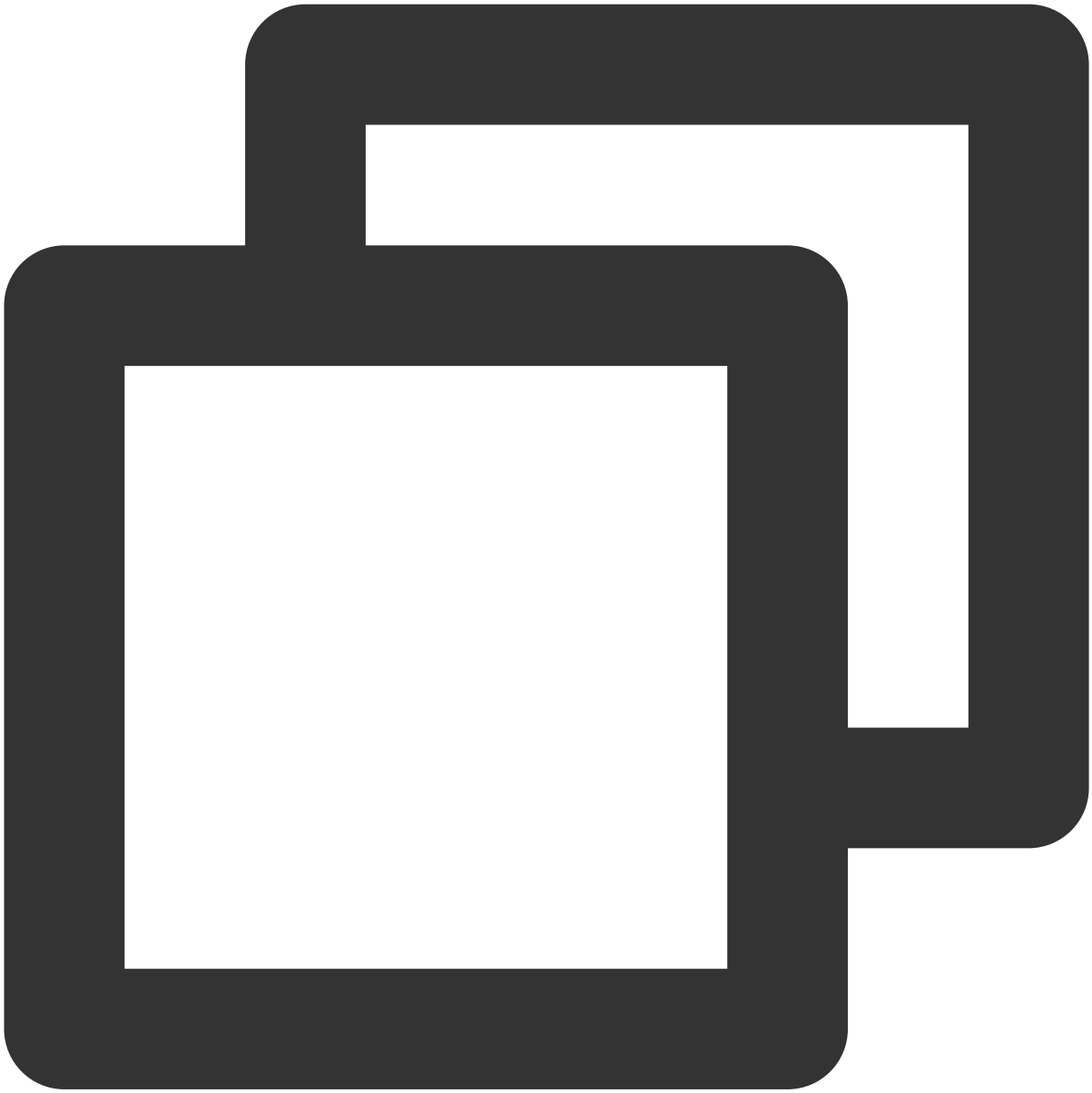
计划后续版本废弃，建议使用：ON_CALL_RECEIVED。

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：

注意：

计划后续版本废弃，建议使用：ON_CALL_CANCELED。

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；

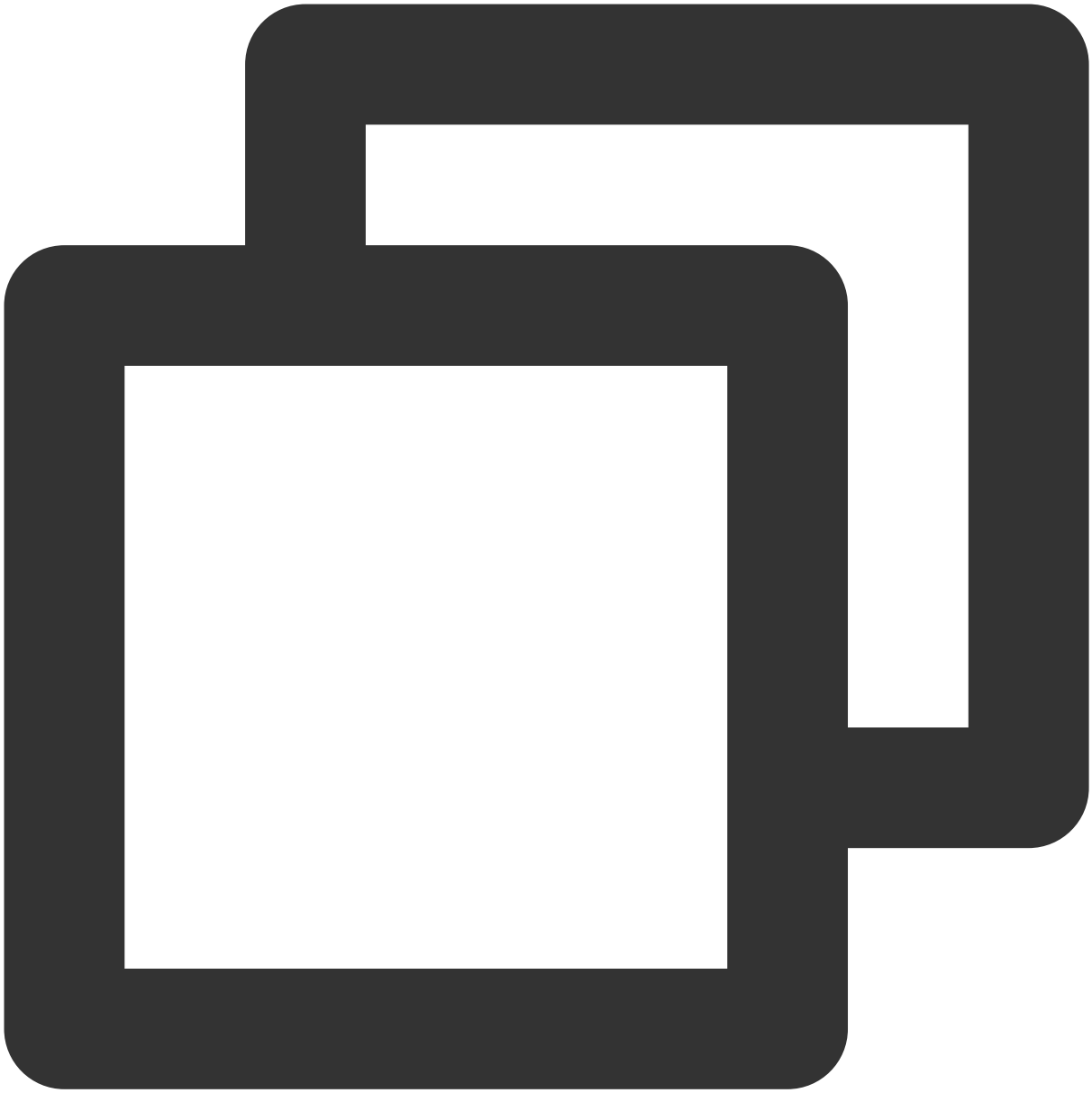
被叫超时：主叫会同时抛出 NO_RESP 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

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

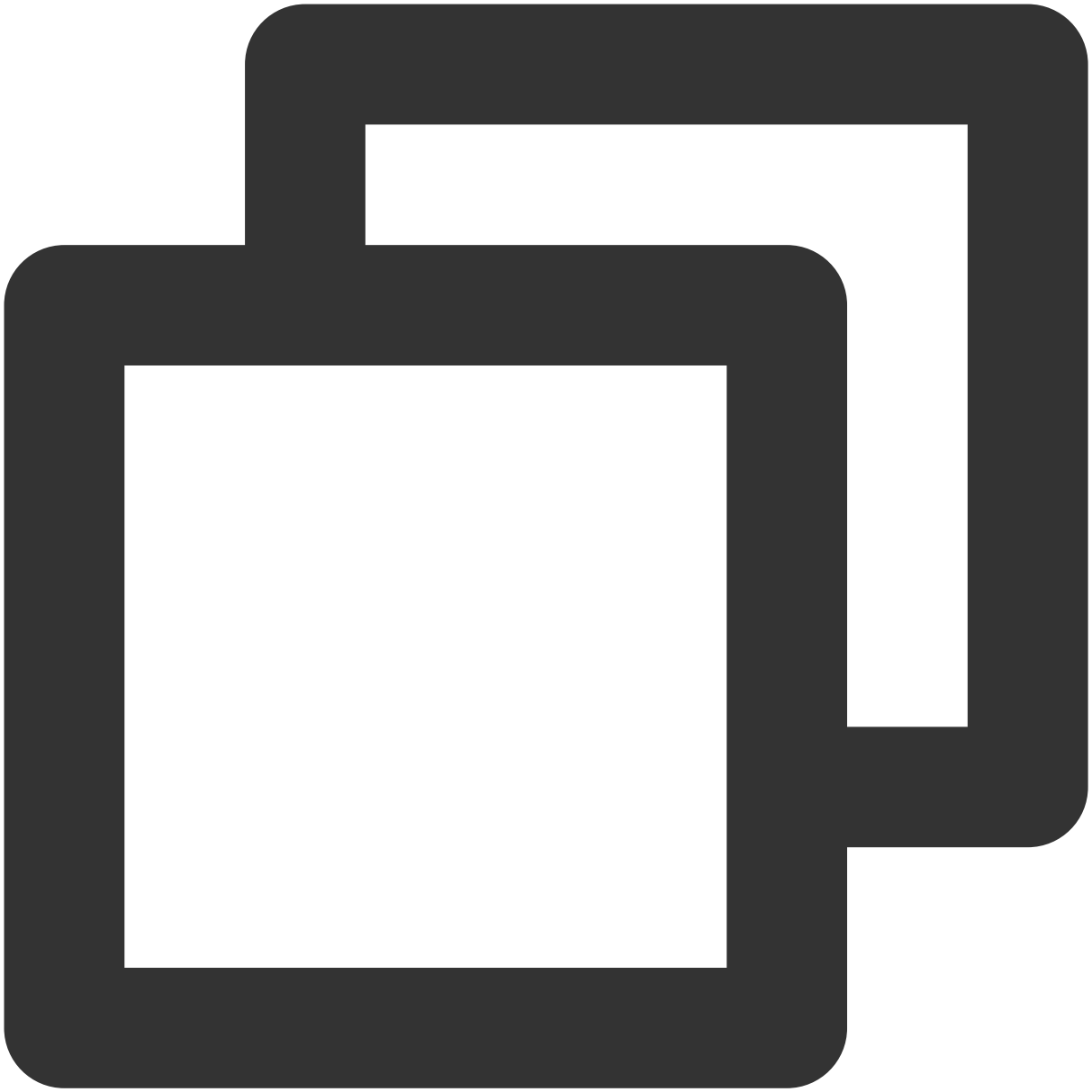
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫、被叫

ON_CALL_BEGIN

表示通话接通，主叫和被叫都会抛出该事件，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);
```

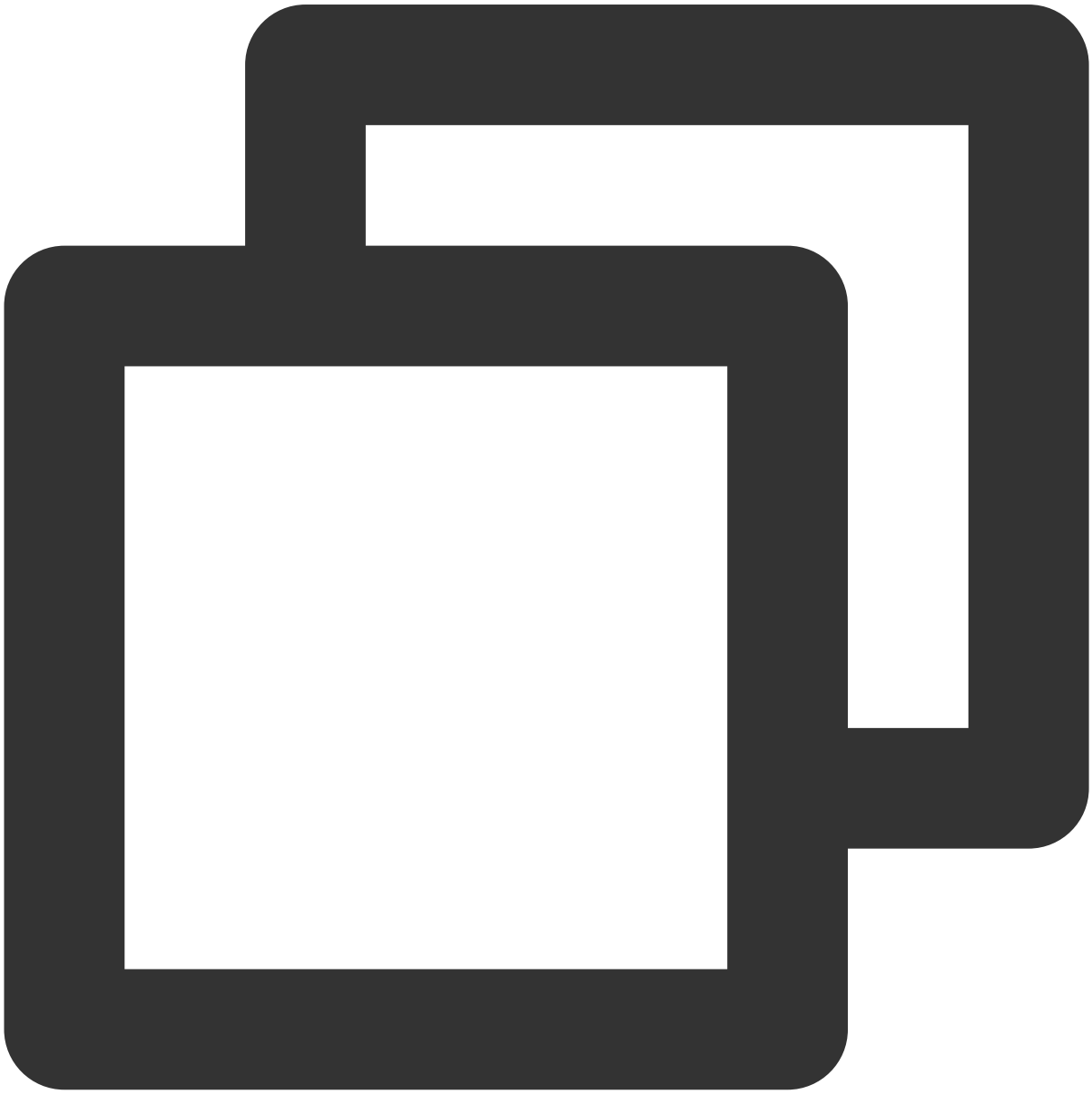
参数如下表所示：

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

通话结束时，主叫和被叫都会抛出该事件。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。



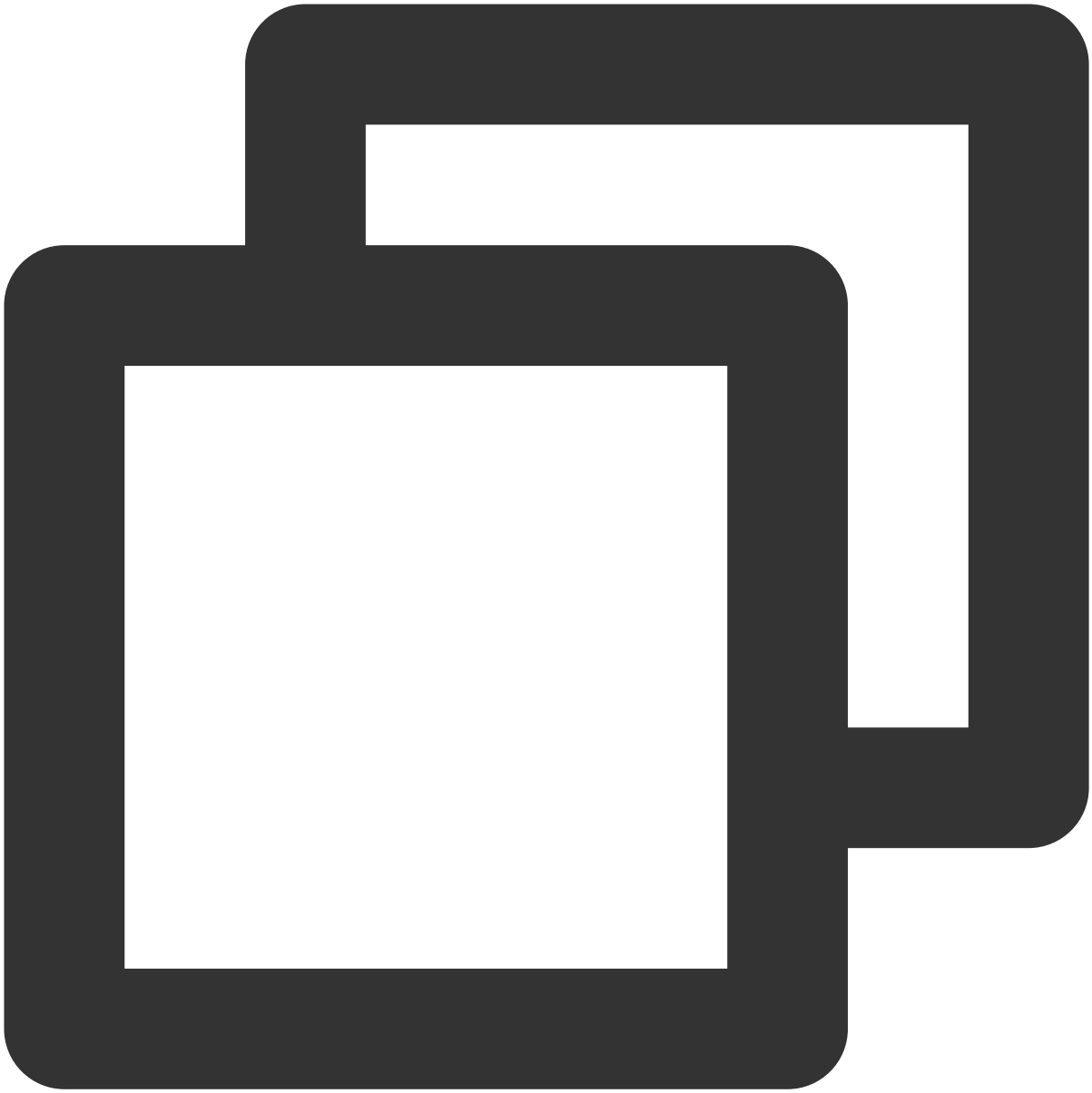
```
let handleCallingEnd = function(event) {
  console.log(event.userID, event.);
};
tuiCallEngine.on(TUICallEvent.CALLING_END, handleCallingEnd);
```

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
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，请使用 totalTime**），单位: 秒

DEVICED_UPDATED

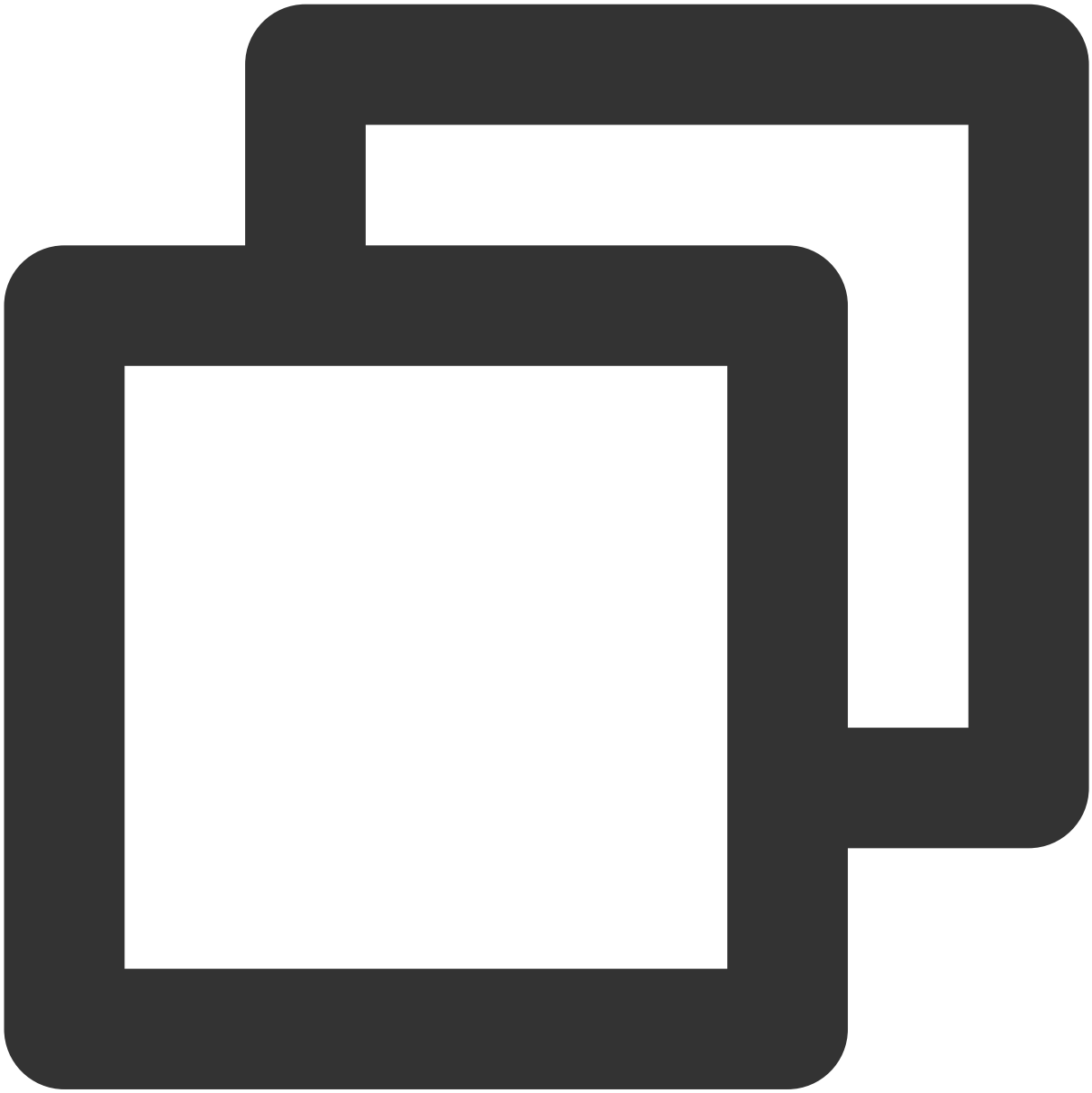
设备列表更新时抛出该事件。



```
let handleDeviceUpdated = function({ microphoneList, cameraList, currentMicrophoneI
  console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID);
};
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

CALL_TYPE_CHANGED

通话类型切换时抛出该事件。



```
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

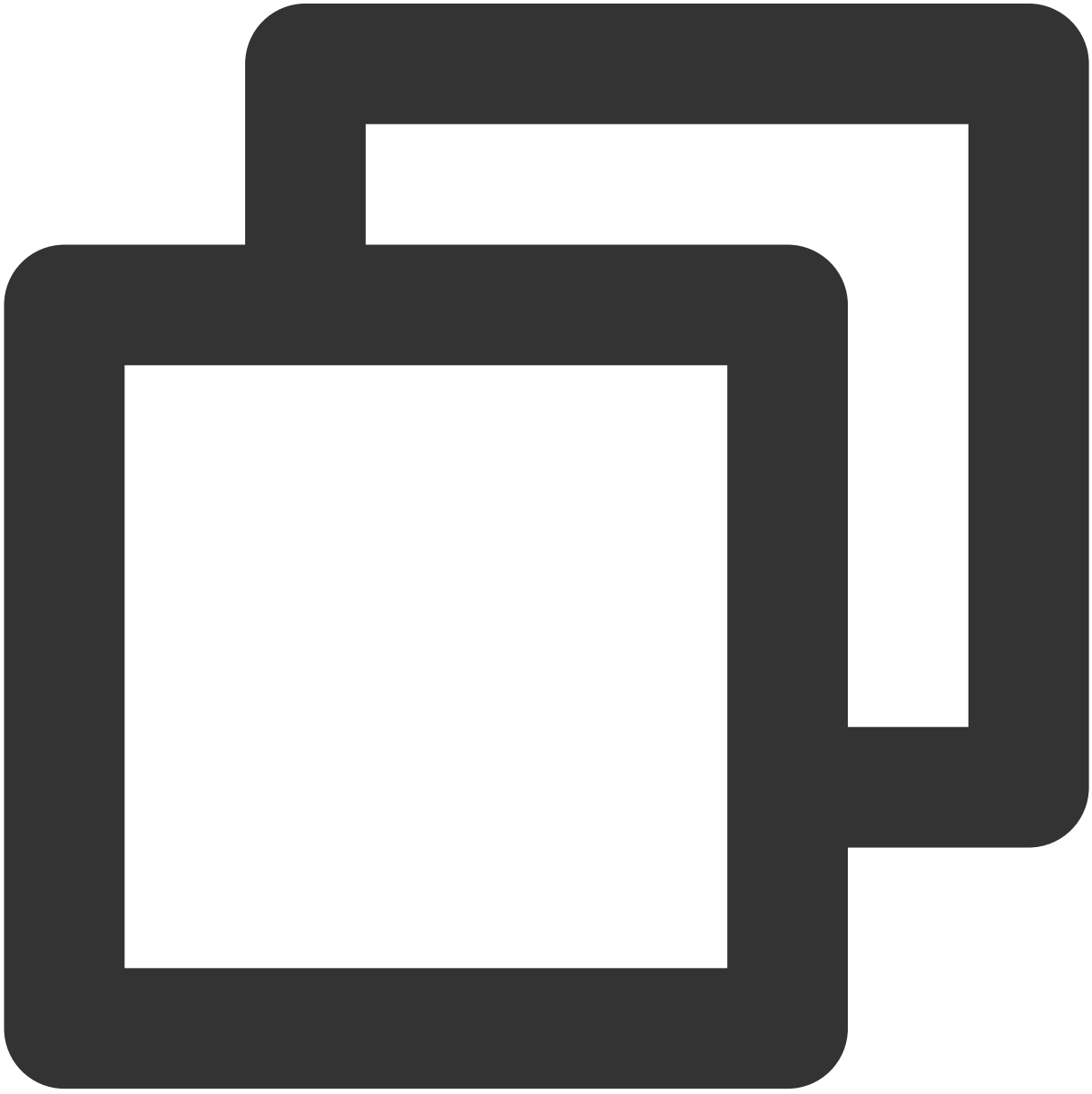
newCallType	Number	新通话类型

ON_USER_NETWORK_QUALITY_CHANGED

所有用户网络质量的事件

注意：

v3.0.7+ 支持。



```
let handleOnUserNetworkQualityChange = function(event) {
  console.log(event.networkQualityList);
};
```

```
tuiCallEngine.on(TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED, handleOnUserNetworkQ
```

参数如下表所示：

参数	类型	含义
networkQualityList	Array<Object>	网络状态，根据 userID 可以获取对应用户当前的网络质量（仅表示本地的上下行）。例如：`networkQualityList: [{ userId: quality }]`。 **网络质量说明：** quality = 0，网络状况未知 quality = 1，网络状况极佳 quality = 2，网络状况较好 quality = 3，网络状况一般 quality = 4，网络状况差 quality = 5，网络状况极差 quality = 6，网络链接已断开

Flutter
API 概览

最近更新时间：2024-05-23 16:24:18

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
enableVirtualBackground	开启/关闭虚拟背景功能

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
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

版权所有：腾讯云计算（北京）有限责任公司

setVirtualBackground	设置虚拟背景图片
setBlurBackground	设置视频模糊效果
setBeautyLevel	设置美颜级别

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：2024-05-23 16:24:18

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）。

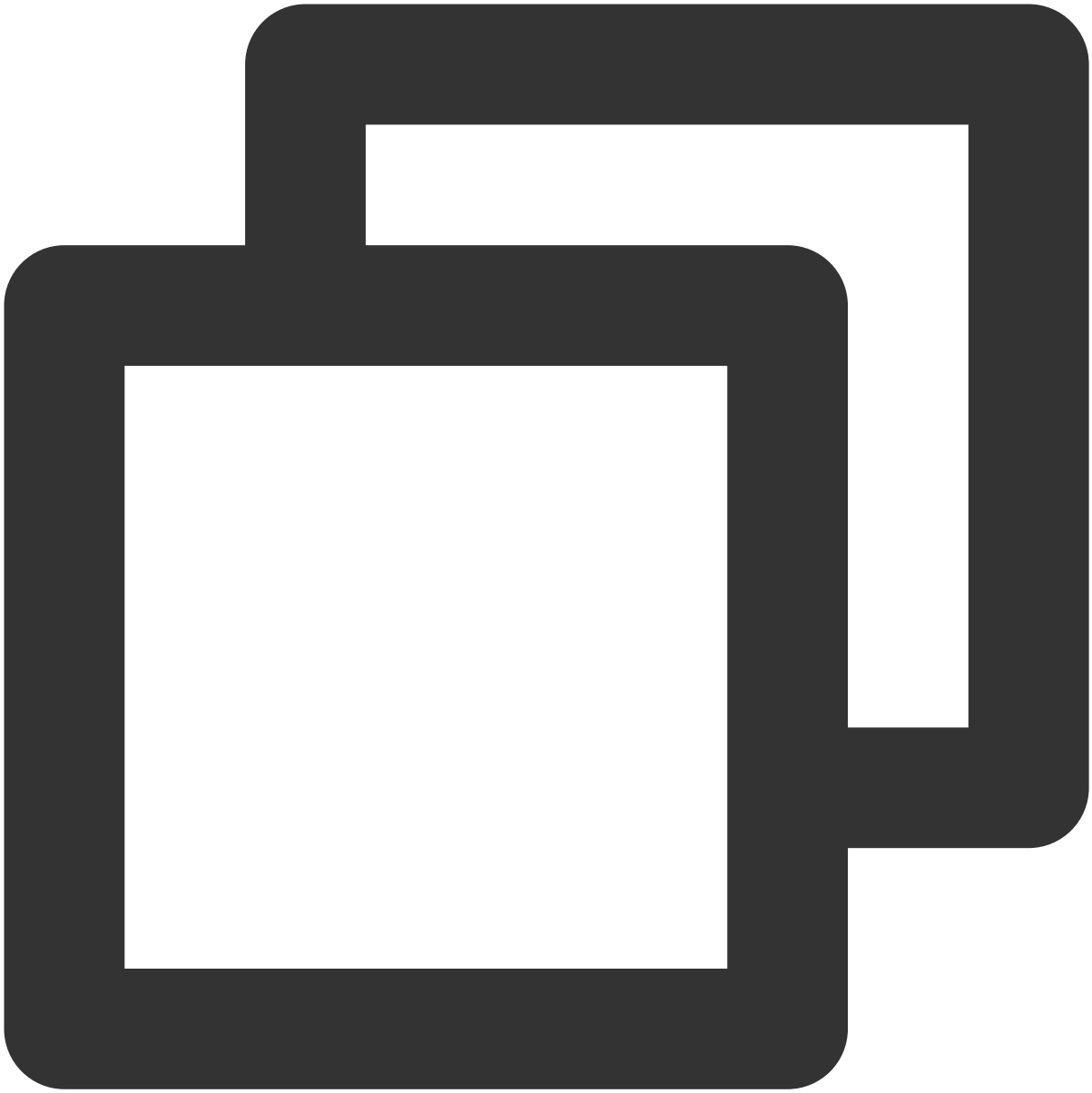
API 概览

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
enableVirtualBackground	开启/关闭虚拟背景功能

API 详情

login

登录。



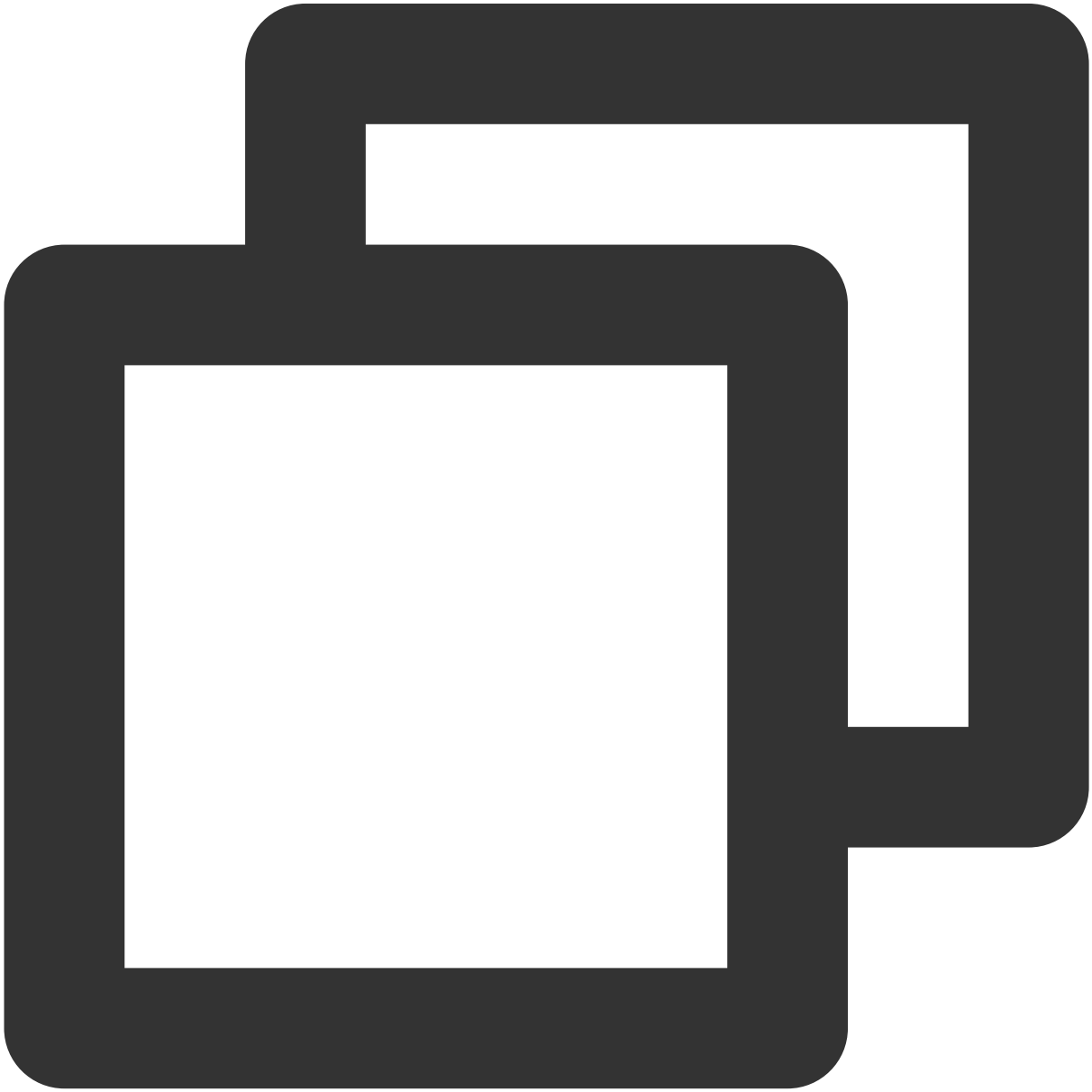
```
Future<TUIResult> login(int sdkAppId, String userId, String userSig);
```

参数	类型	含义
sdkAppId	int	用户 SDKAppID。
userId	String	用户 ID，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	用户签名 userSig。

		userSig 通过使用 SDKSecretKey 对 sdkAppId、userId 等信息进行加密得到（计算方法详见），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。
返回值	TUIResult	包含 code 和 message 信息：code 为空 ("") 表示调用成功；code 不为空 ("") 表示调用失败，失败原因见 message。

logout

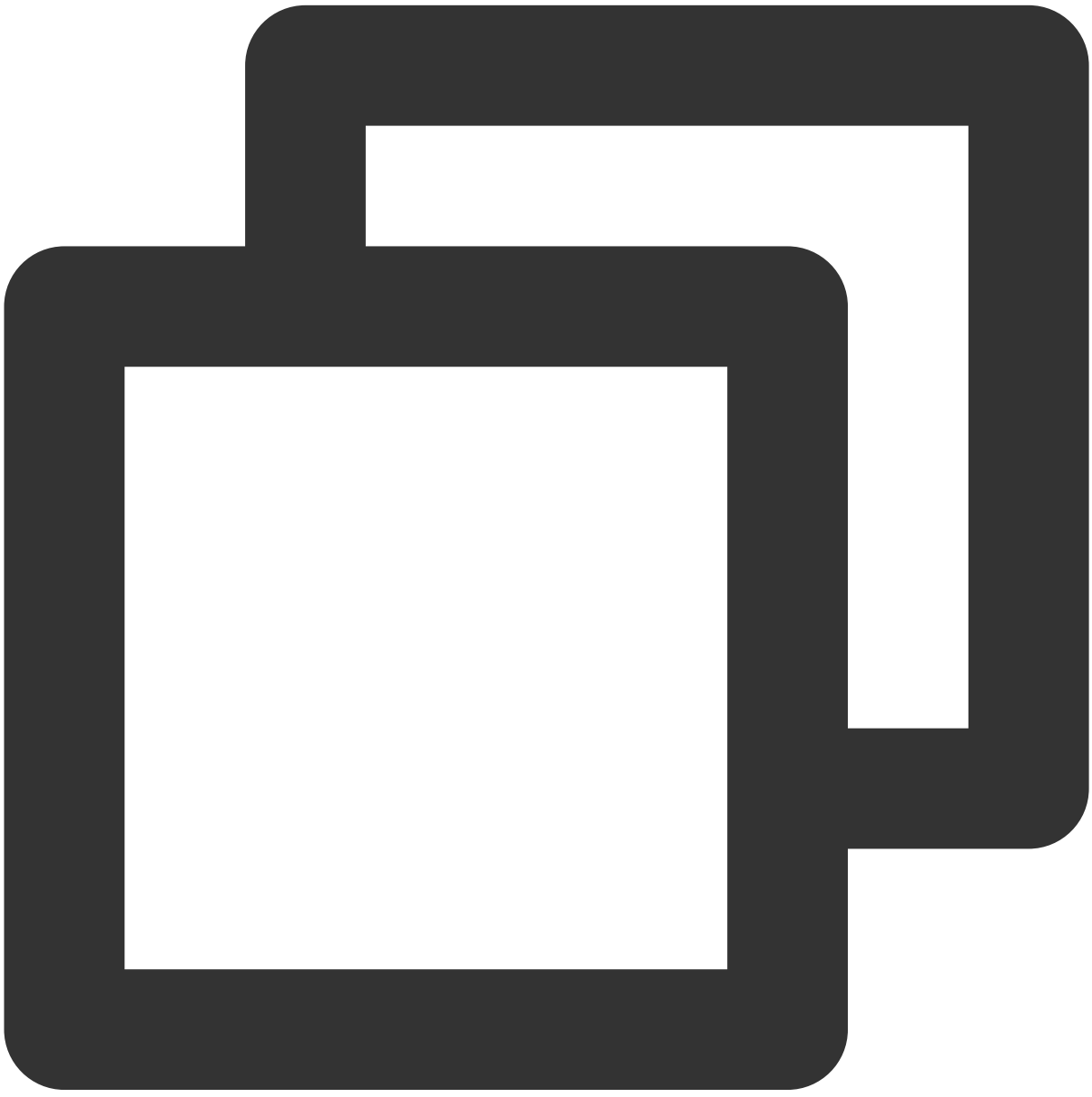
登出。



```
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。



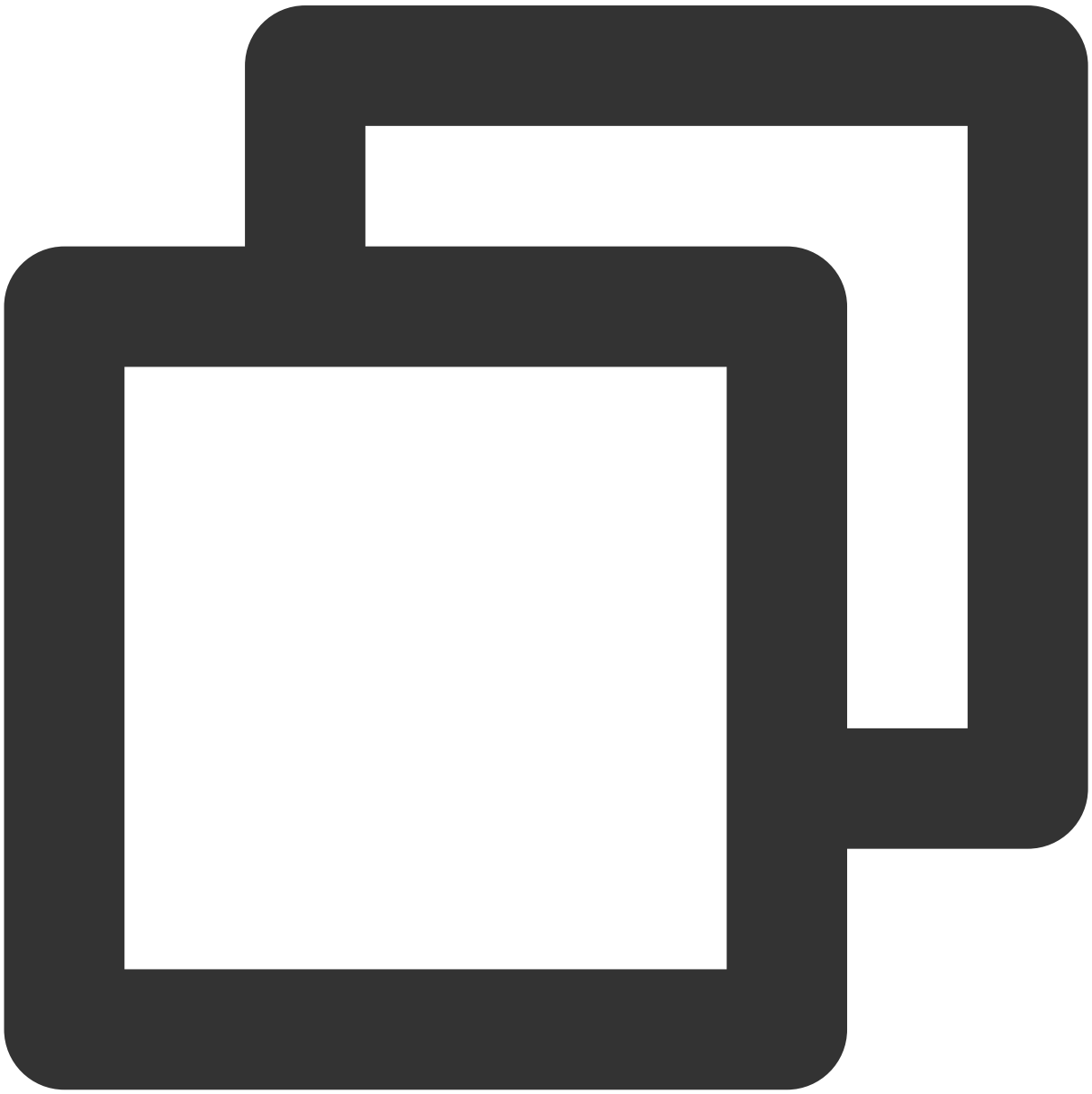
```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

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 为空 ("") 表示调用成功；code 不为空 ("") 表示调用失败，失败原因见 message

call

拨打电话（1v1通话）。



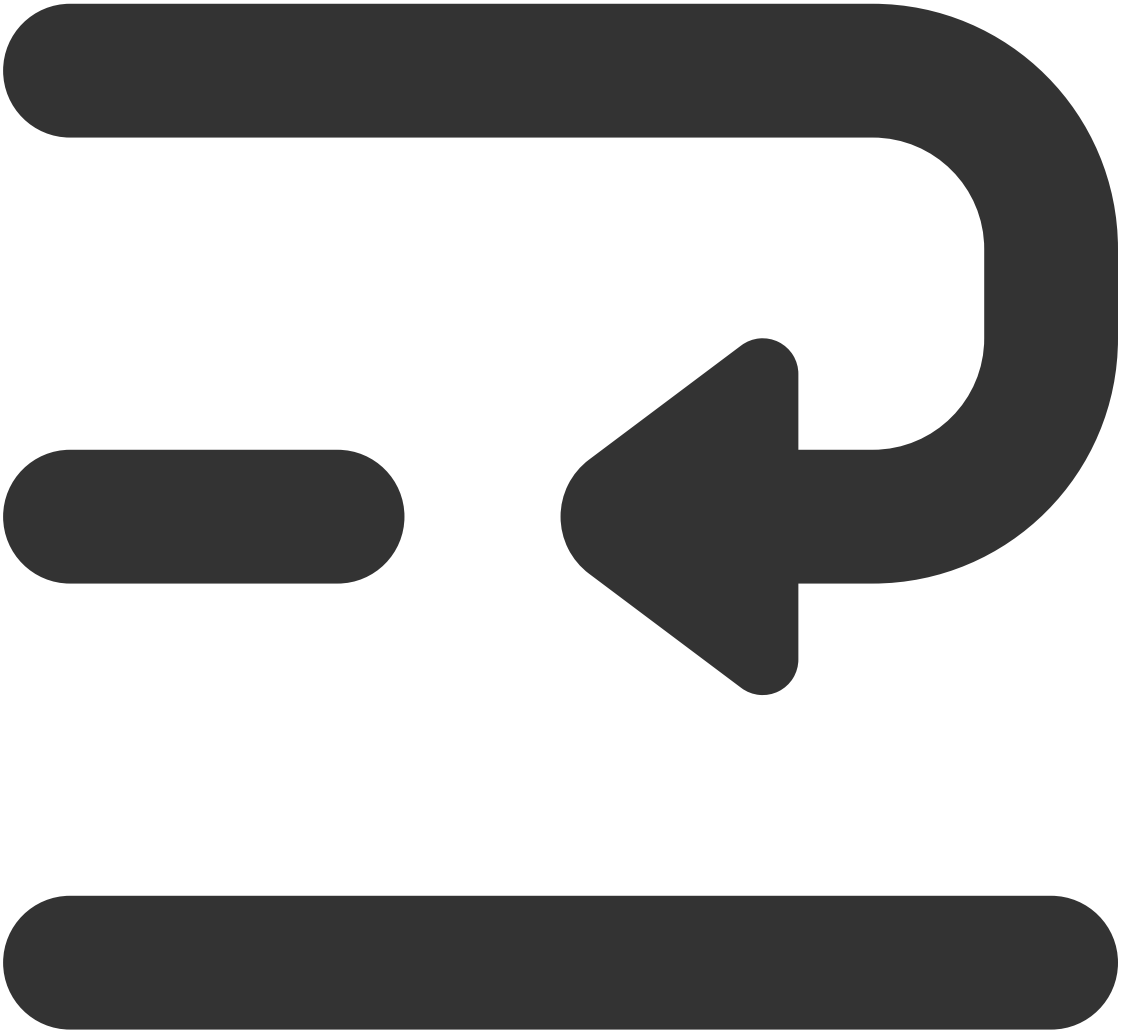
```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? pa
```

参数如下表所示：

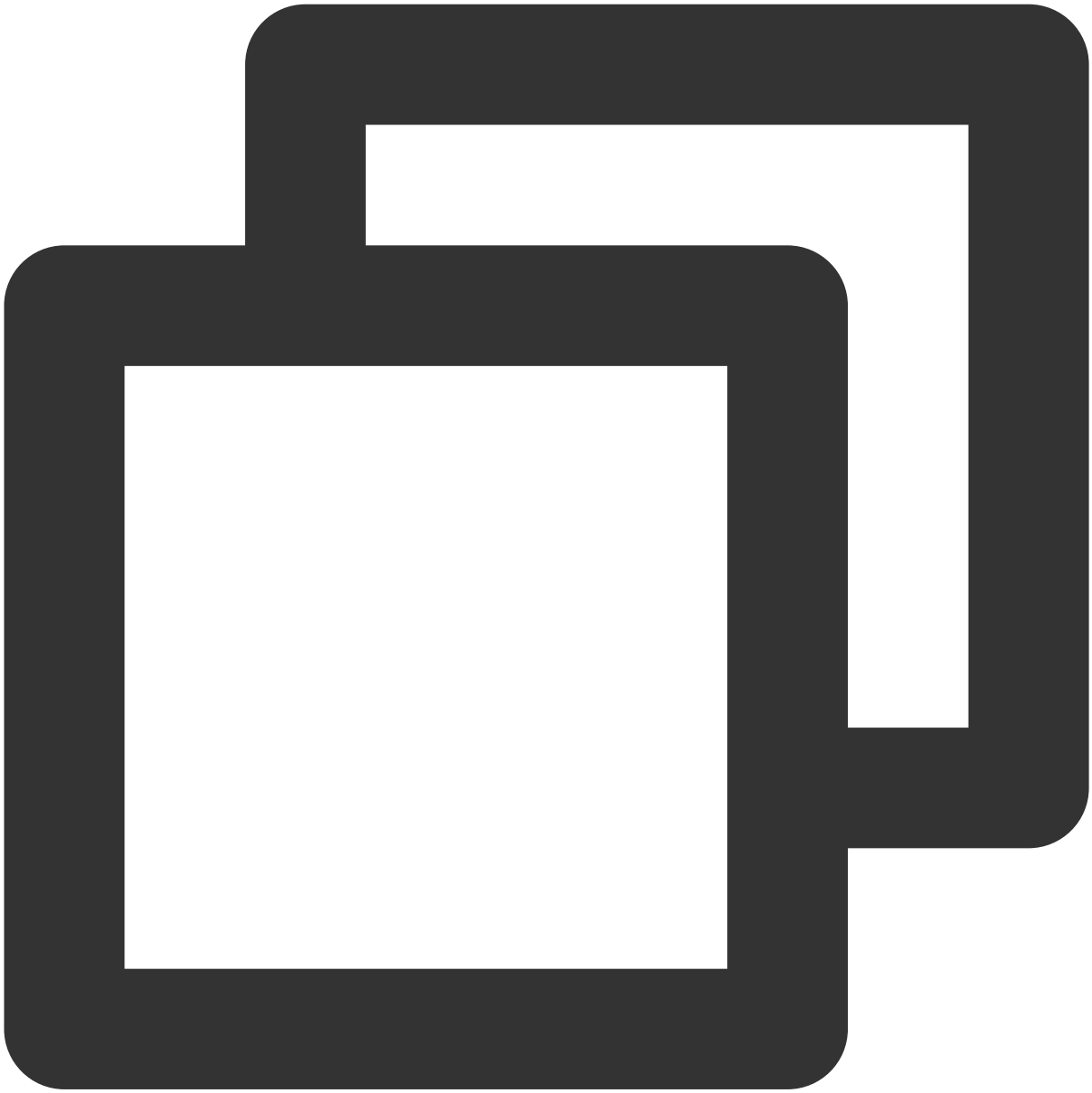
参数	类型	含义
userId	String	目标用户的 userID
callMediaType	TUICallMediaType	通话的媒体类型，比如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`

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。



版权所有：腾讯云计算（北京）有限责任公司

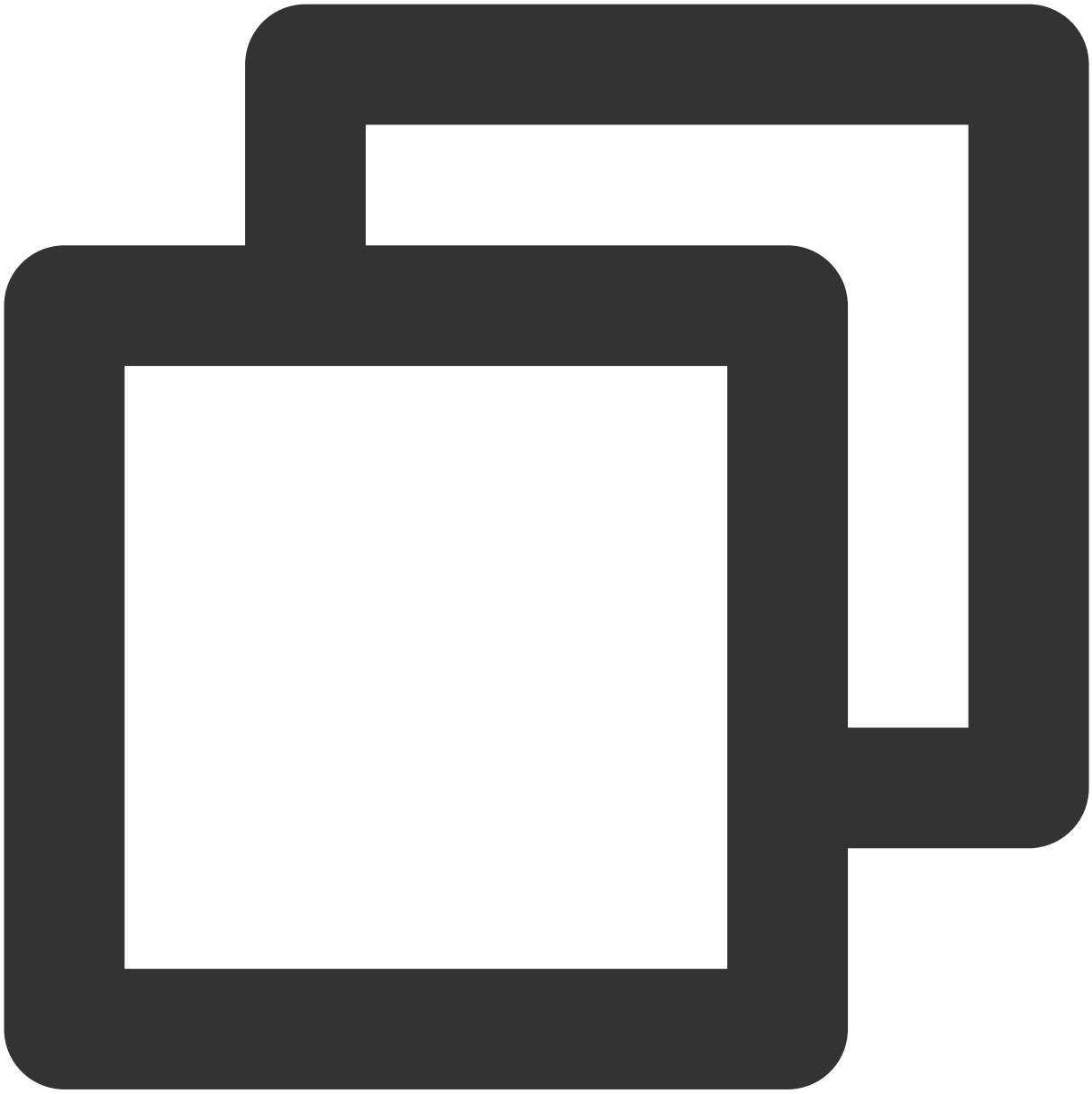


```
Future<void> groupCall(String groupId, List<String> userIdList, TUICallMediaType ca
```

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。



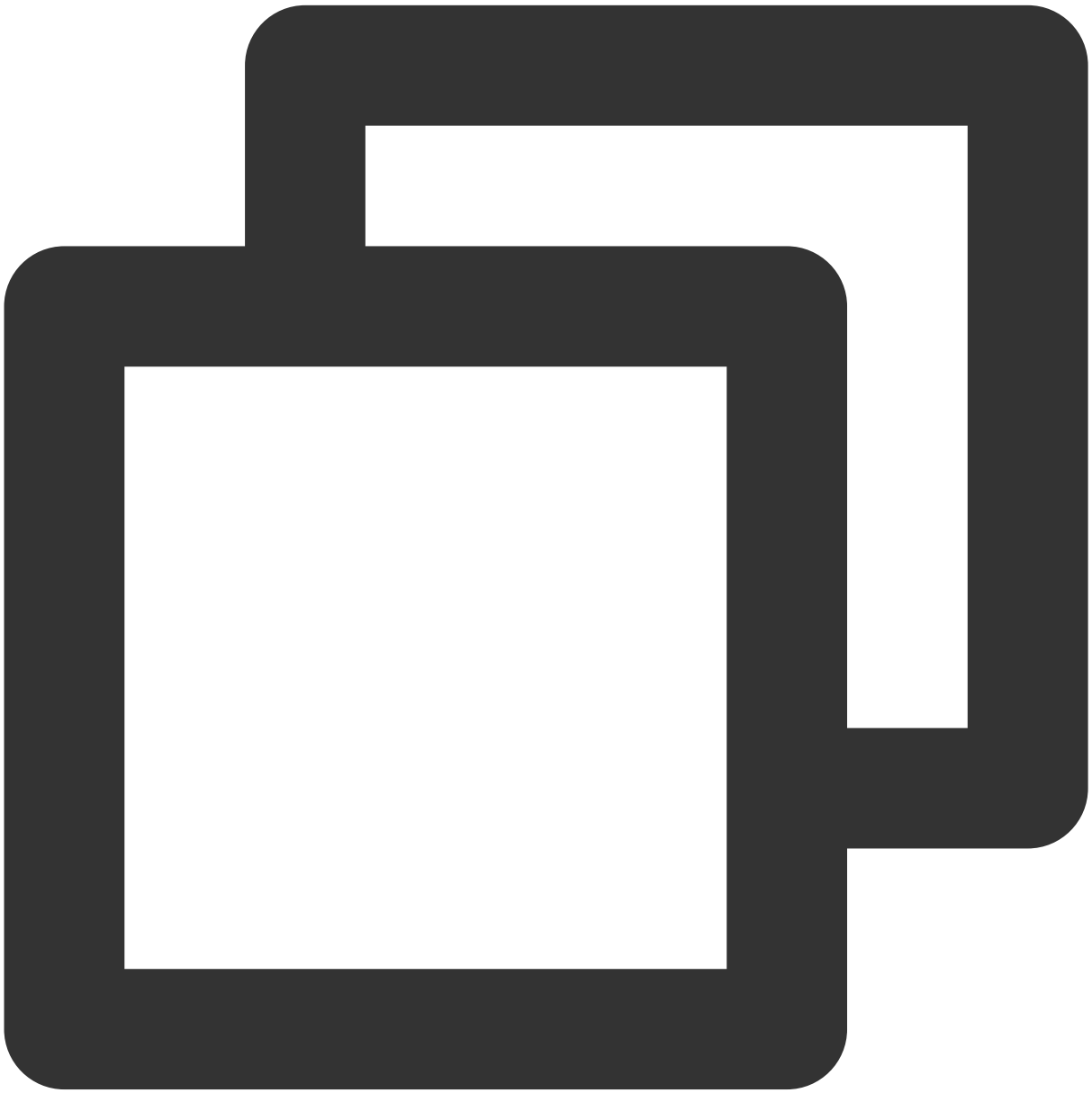
```
Future<void> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType cal
```

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。



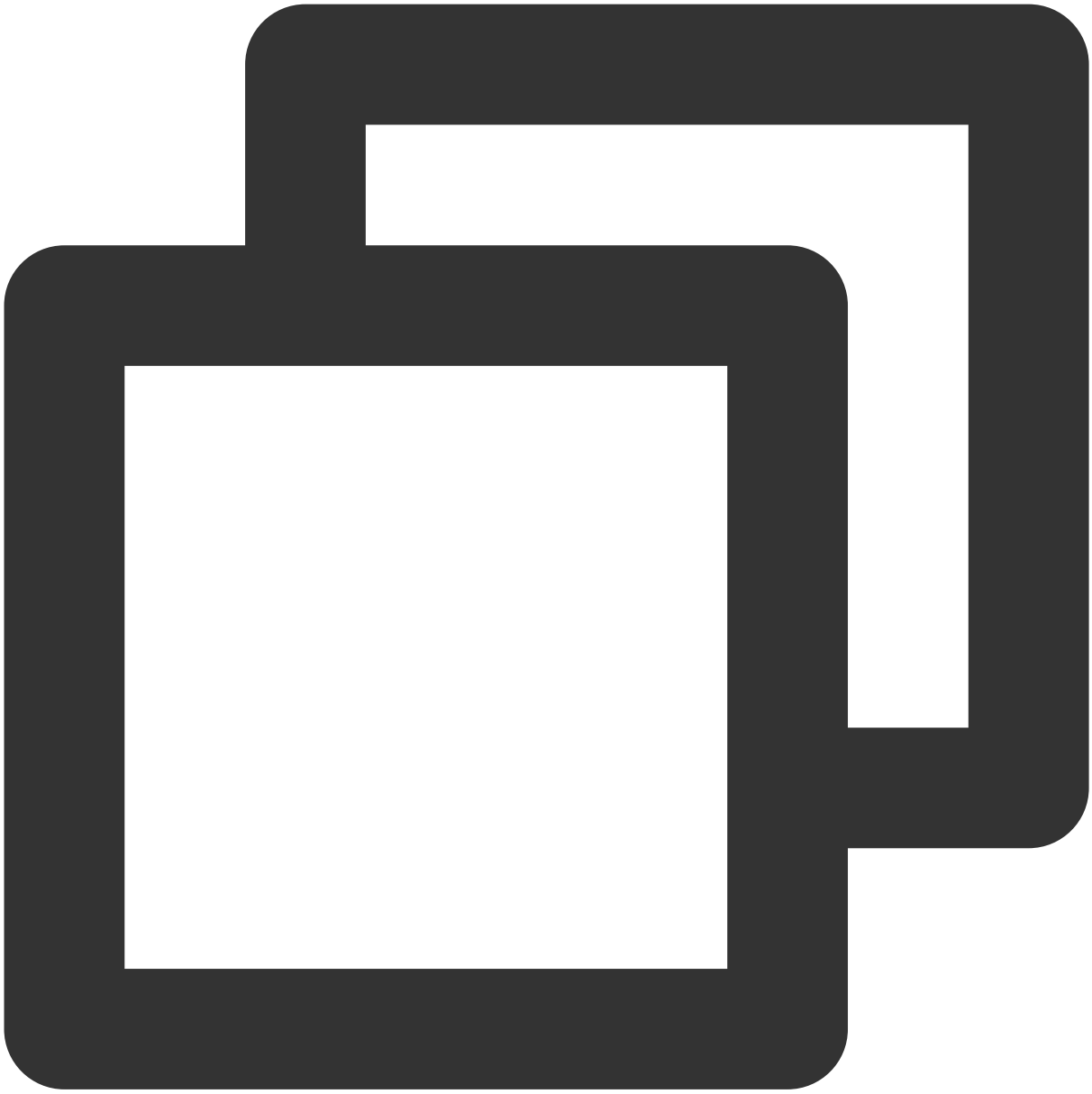
```
Future<void> enableMuteMode(bool enable)
```

参数	类型	含义

enable	bool	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。

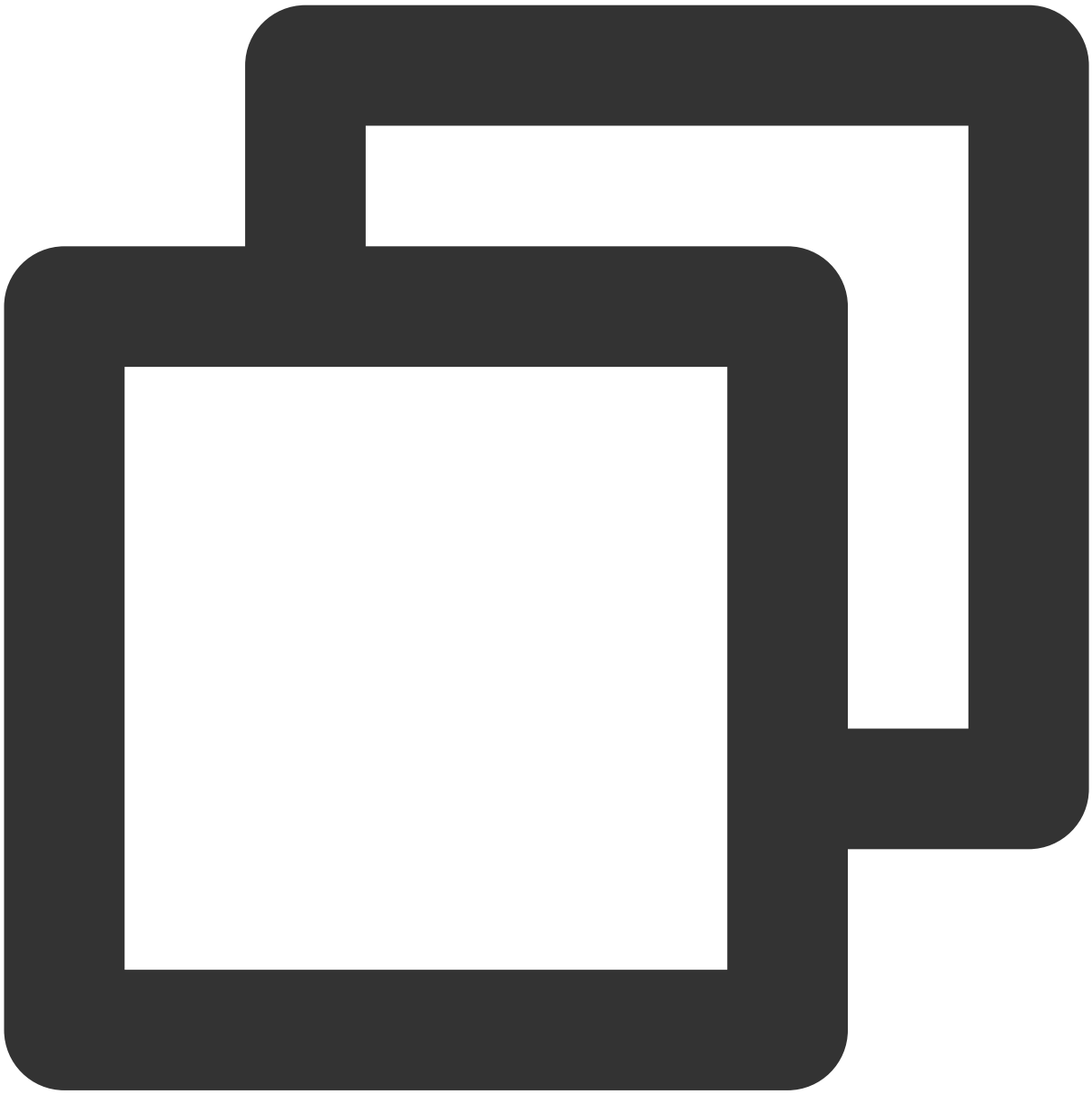


```
Future<void> enableFloatWindow(bool enable)
```

参数	类型	含义
enable	bool	开启、关闭悬浮窗功能；true 表示开启浮窗

setCallingBell

自定义来电铃声。

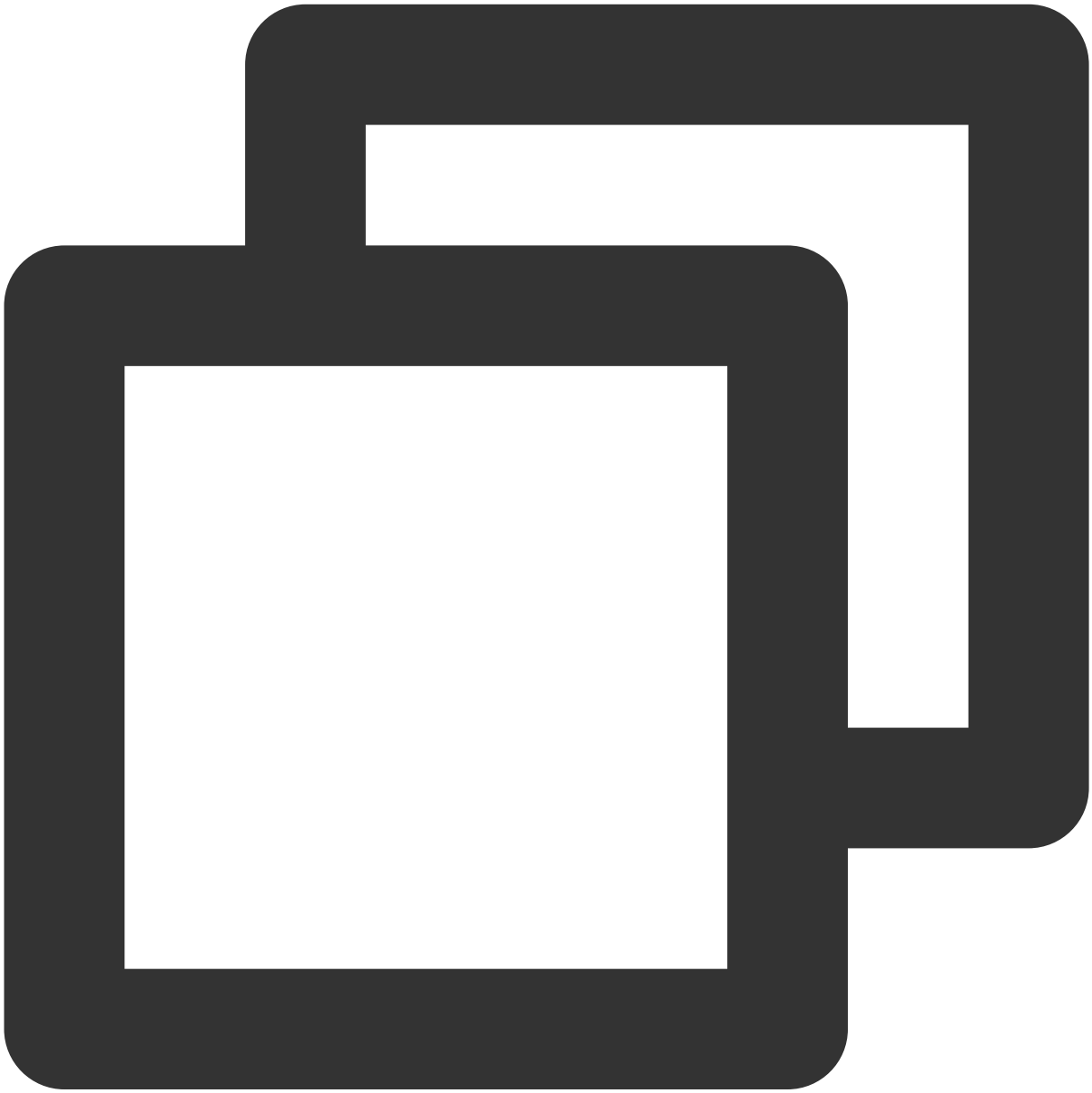


```
Future<void> setCallingBell(String assetName)
```

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。

enableVirtualBackground

开启/关闭虚拟背景功能，开启虚拟背景功能后，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。



```
Future<void> enableVirtualBackground(bool enable)
```

参数	类型	含义
enable	bool	开启、关闭静音；true 表示开启静音

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2024-05-23 16:24:18

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

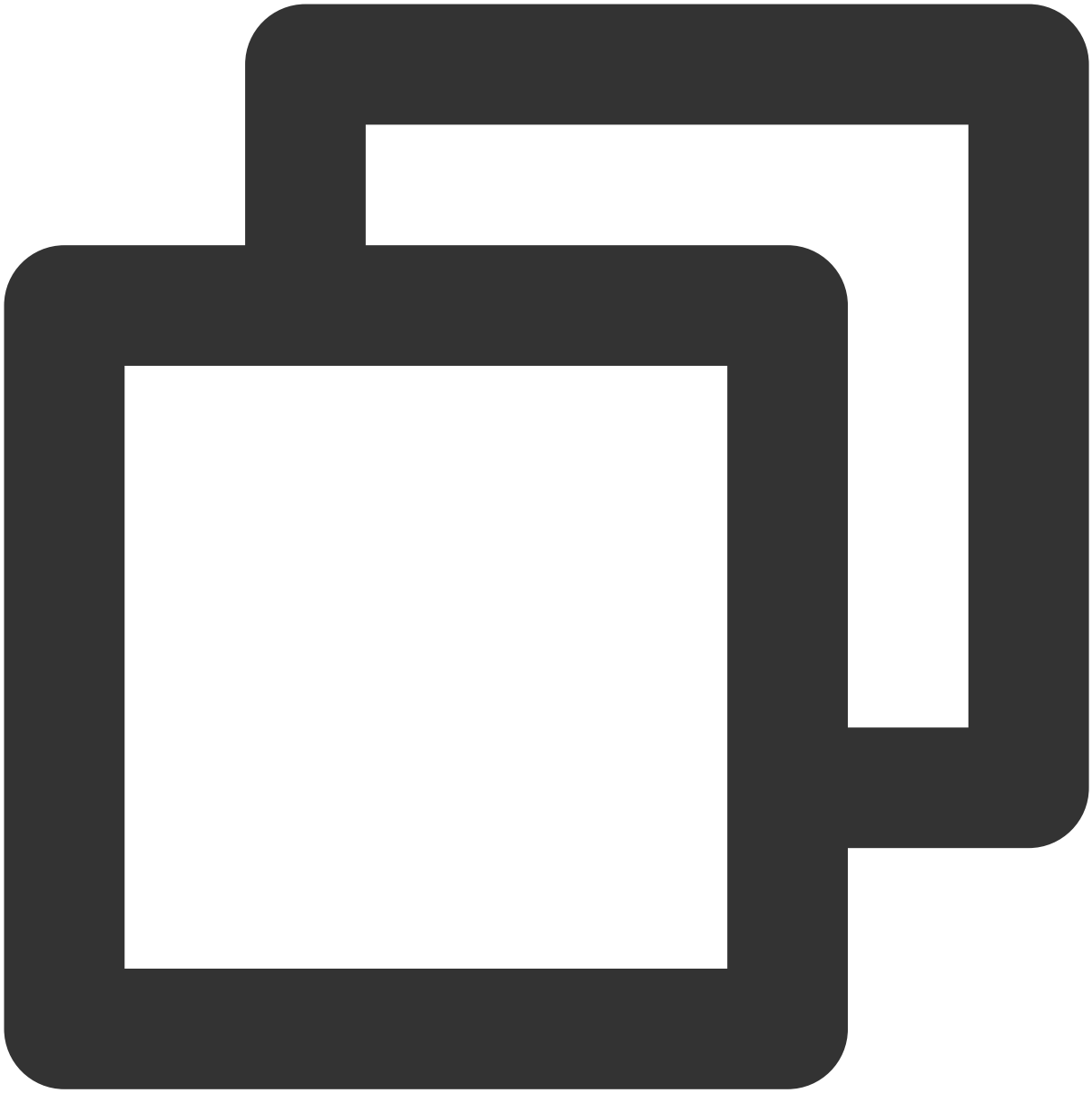
版权所有：腾讯云计算（北京）有限责任公司

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
setBlurBackground	设置视频模糊效果
setVirtualBackground	设置虚拟背景图片
setBeautyLevel	设置美颜级别

API 详情

init

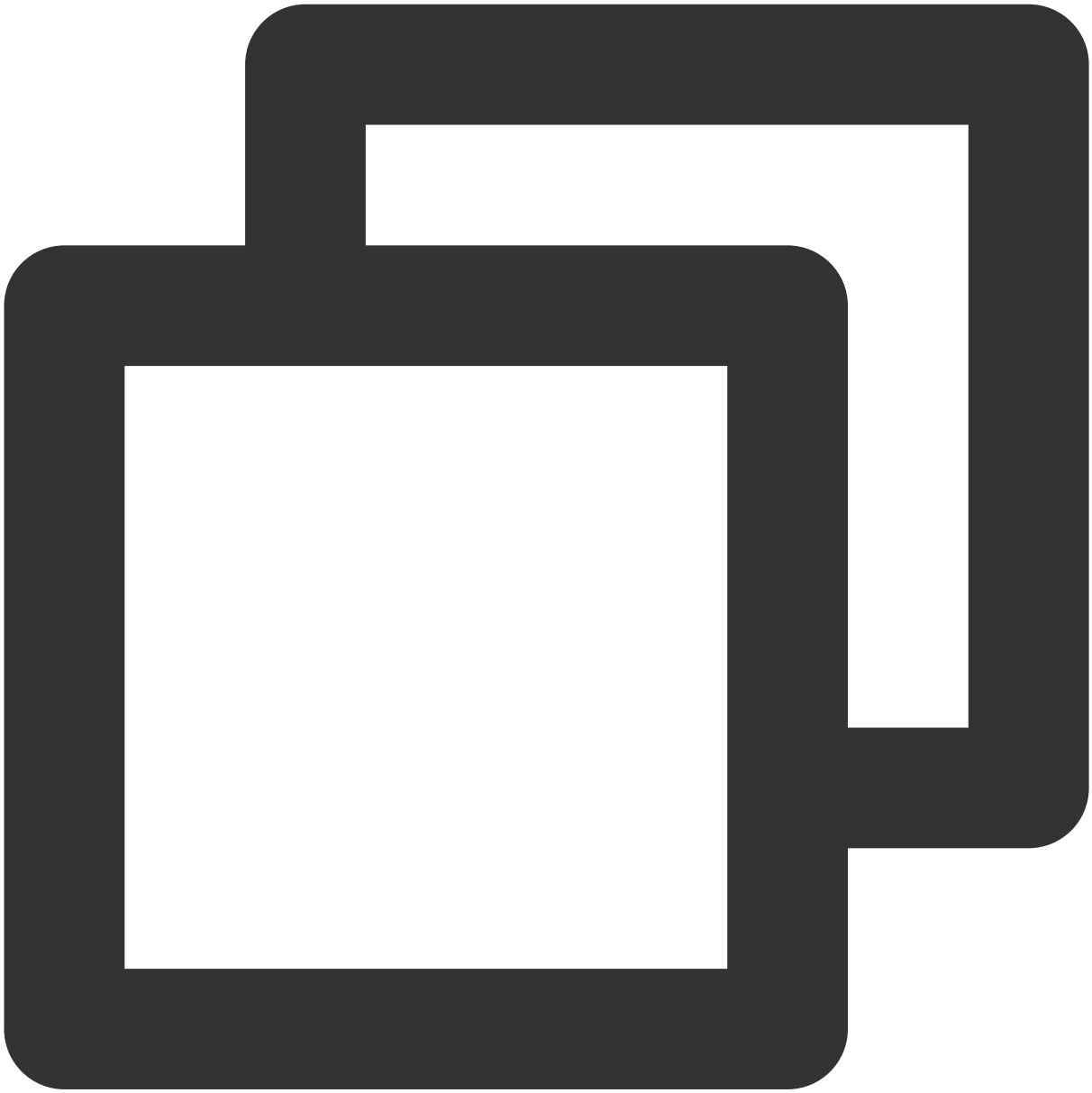
初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。



```
Future<TUIResult> init(int sdkAppID, String userId, String userSig)
```

unInit

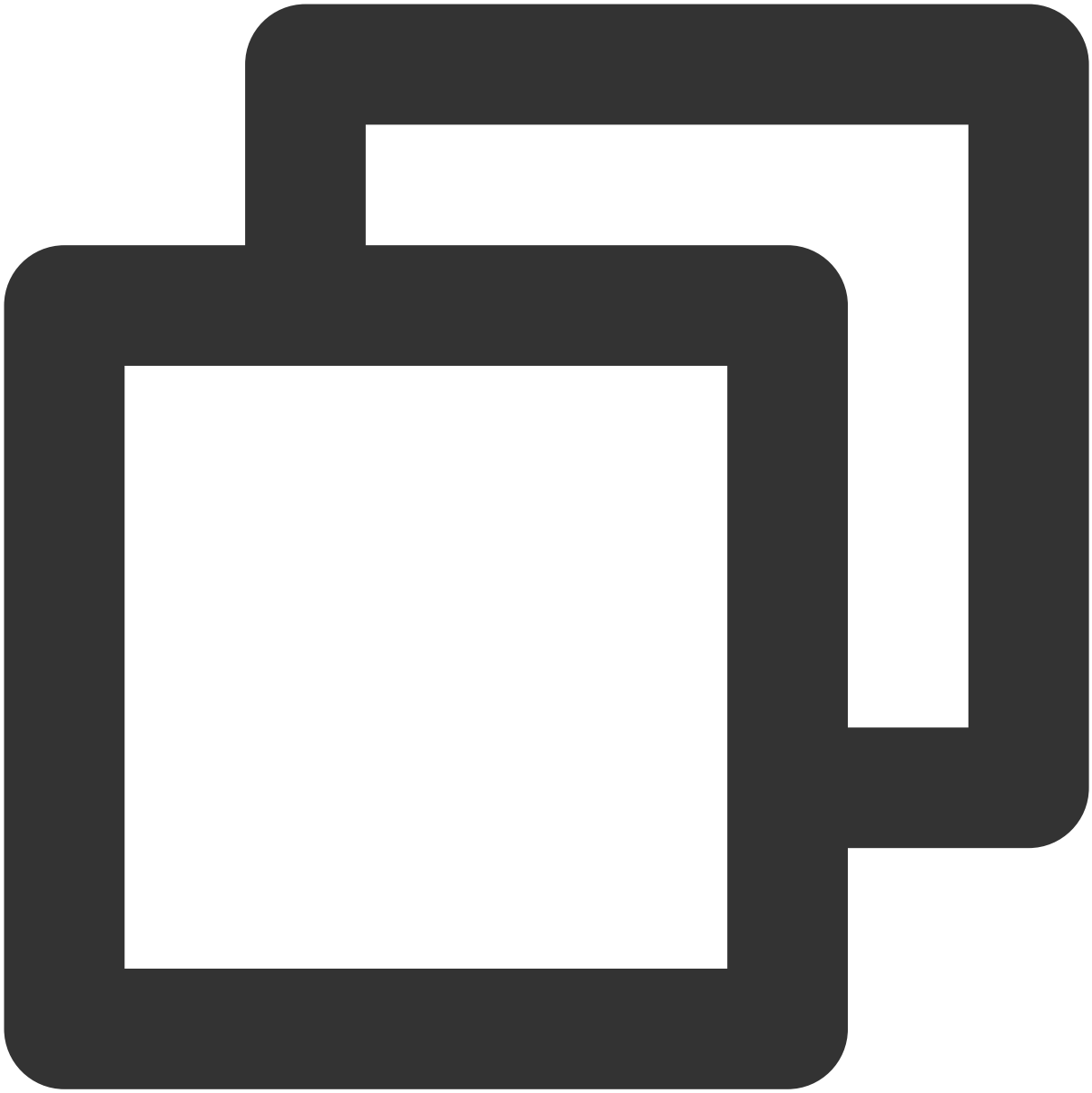
反初始化函数，释放 `TUICallEngine` 使用到的相关资源。



```
Future<TUIResult> unInit()
```

addObserver

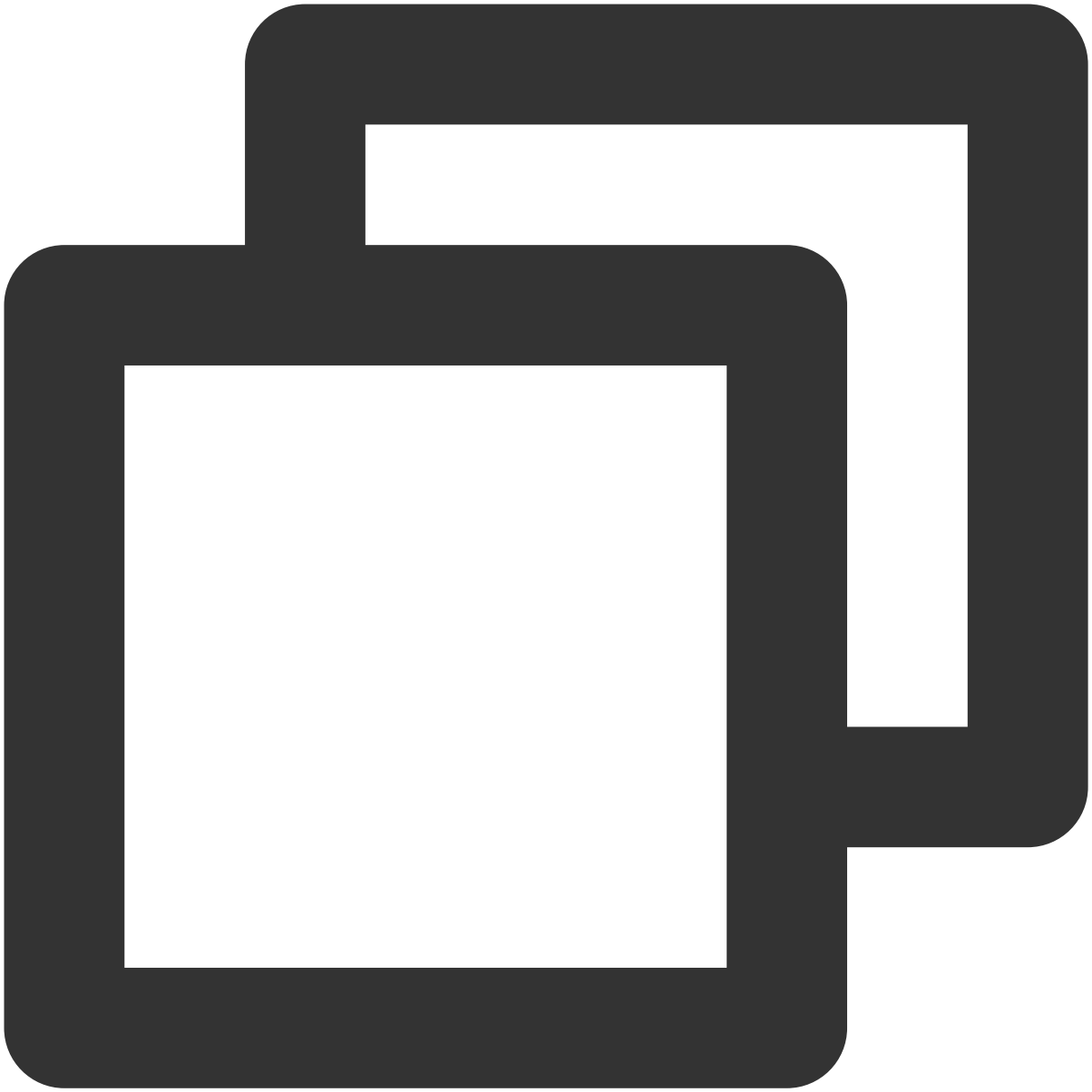
添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。



```
Future<void> addObserver(TUICallObserver observer)
```

removeObserver

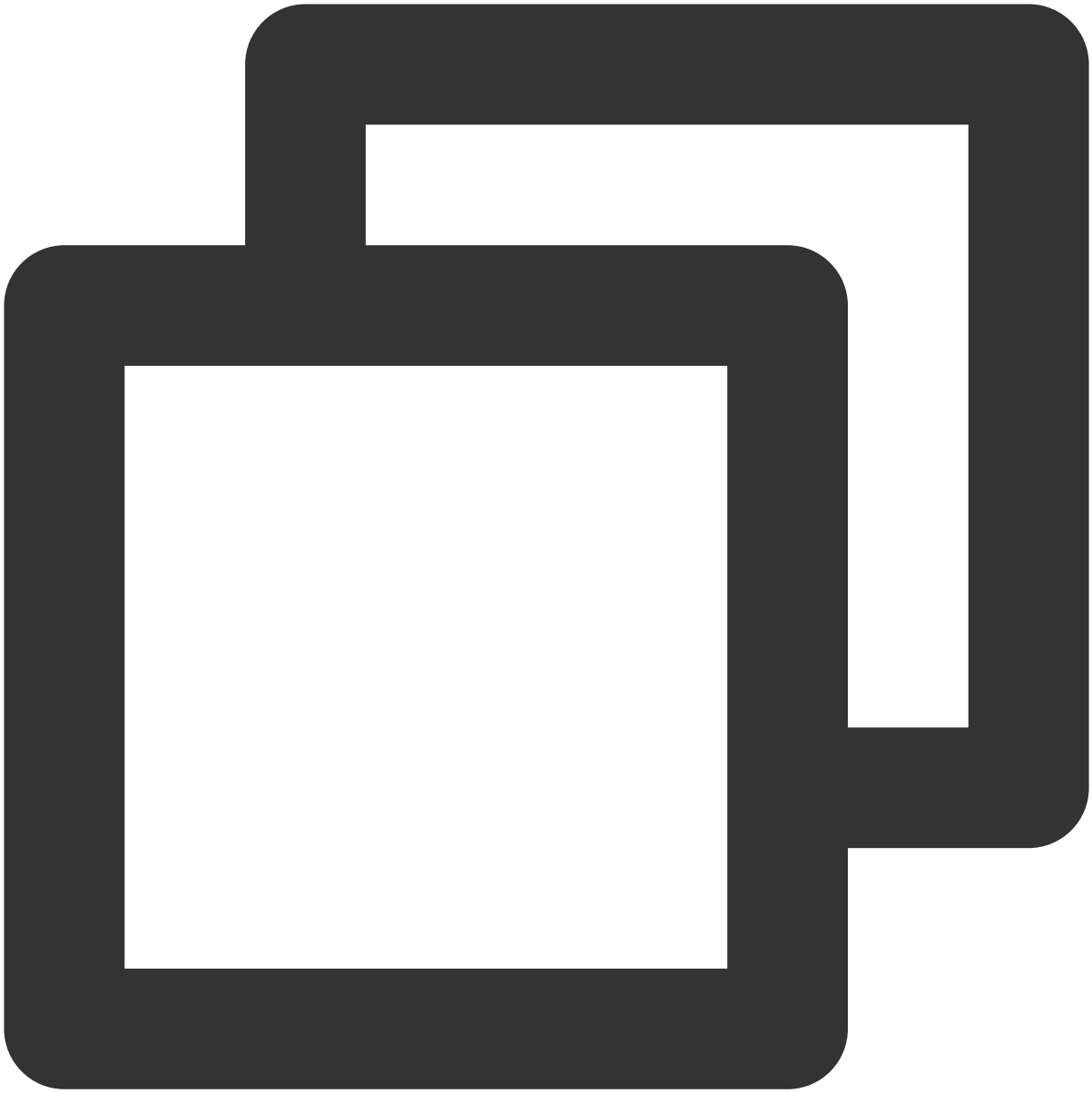
移除回调接口。



```
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

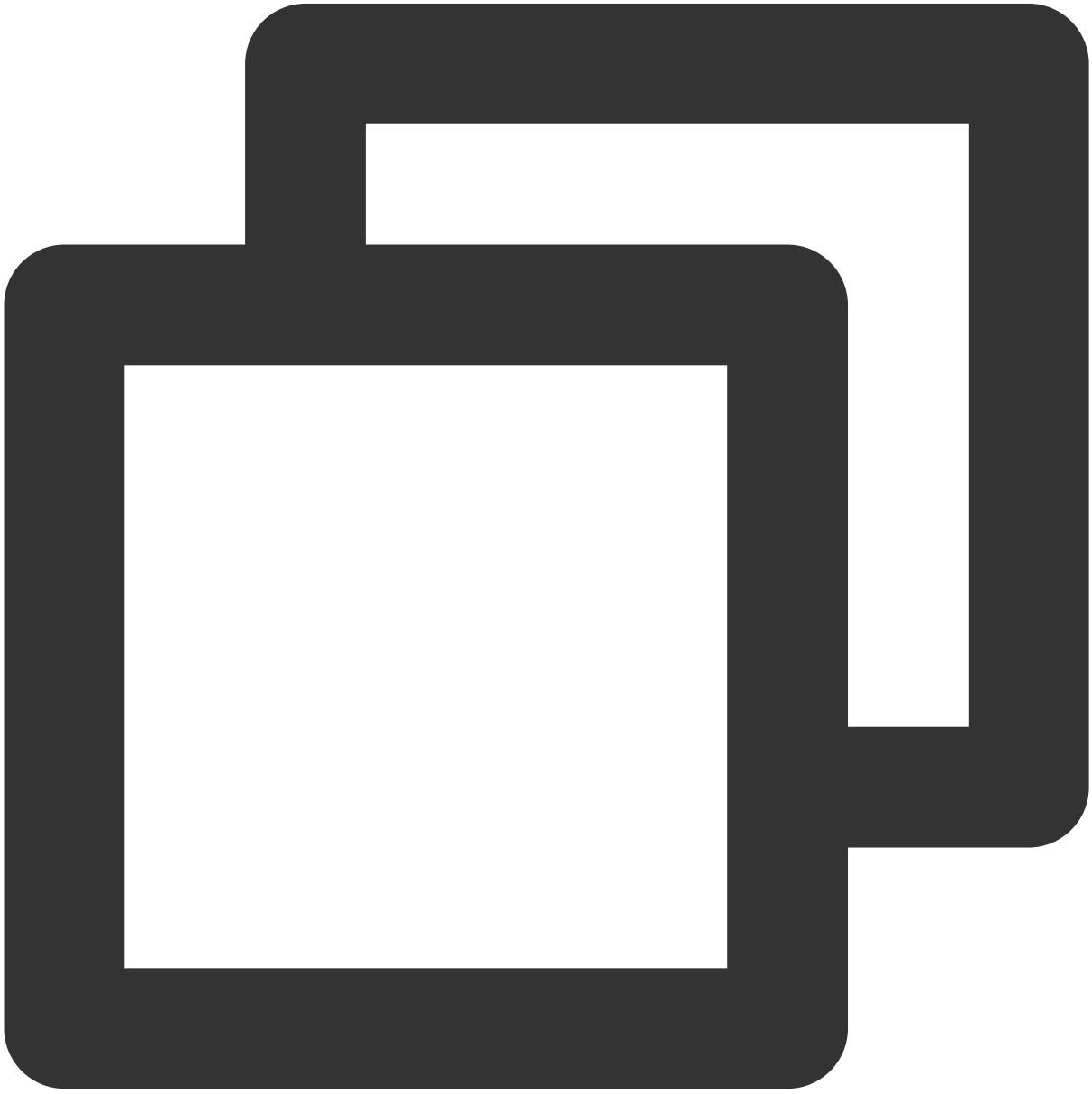
		义内容等

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。

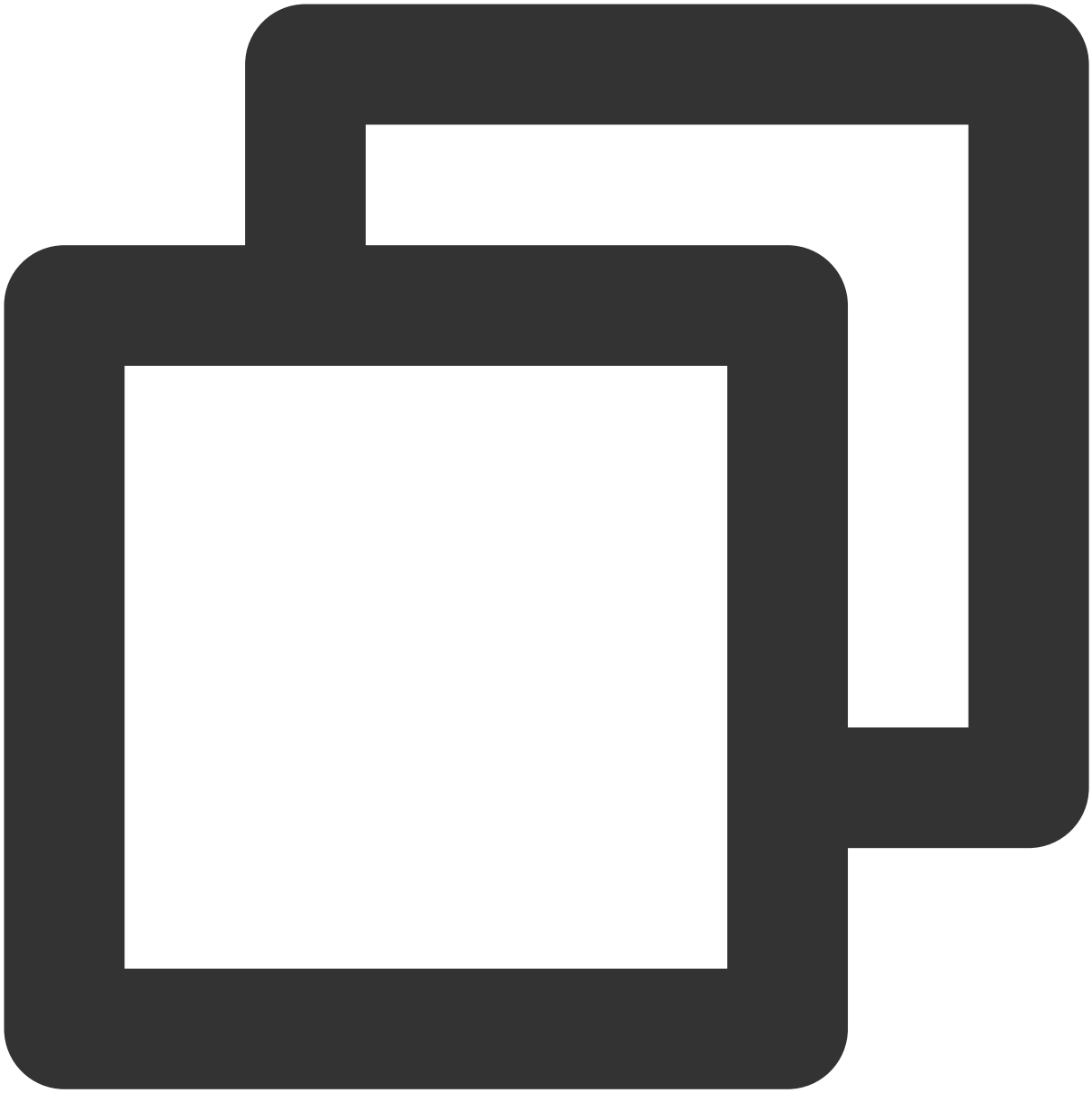


```
Future<TUIResult> groupCall(String groupId, List<String> userIdList, TUICallMediaTy
```


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

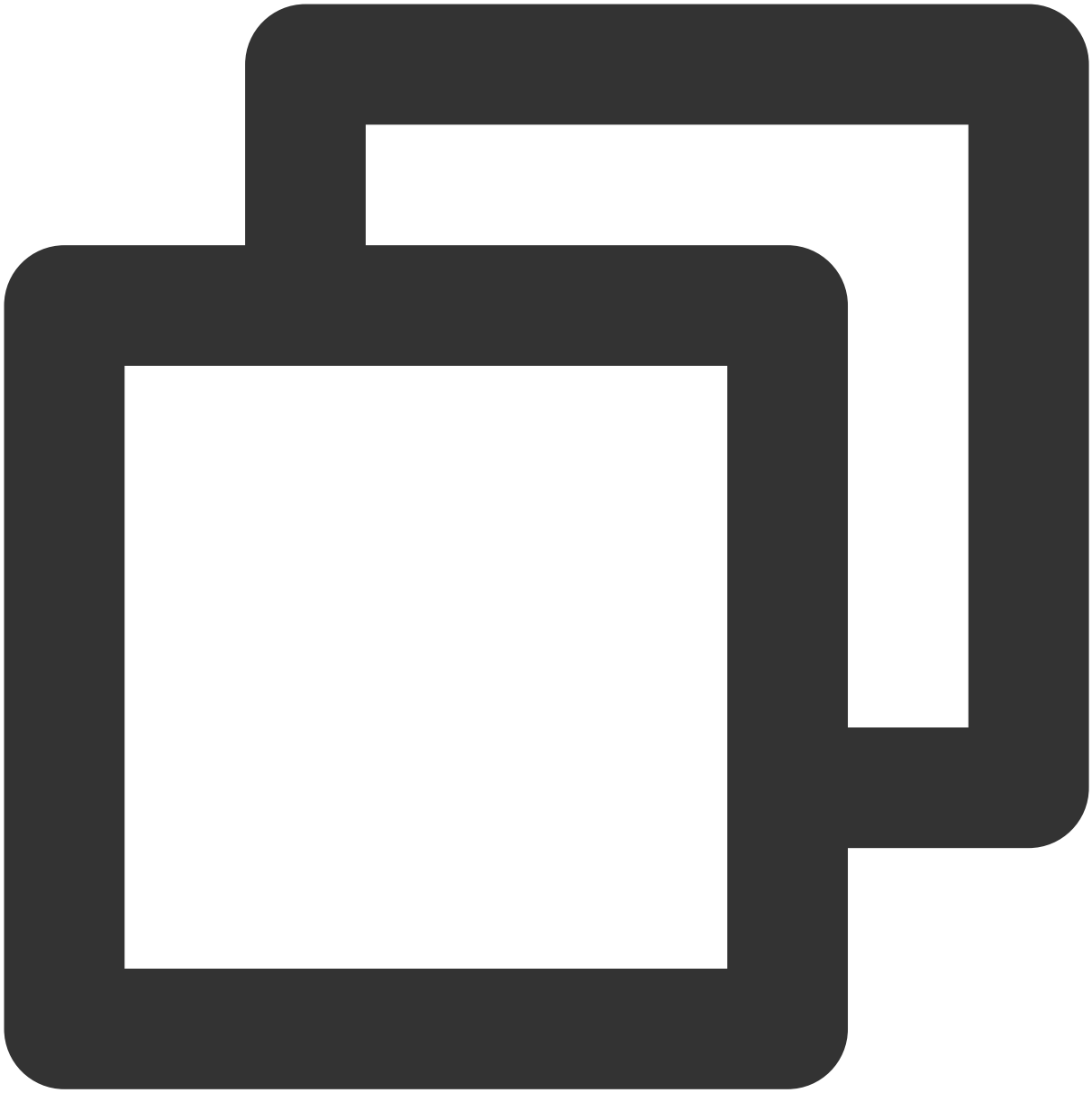
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



```
Future<TUIResult> accept()
```

reject

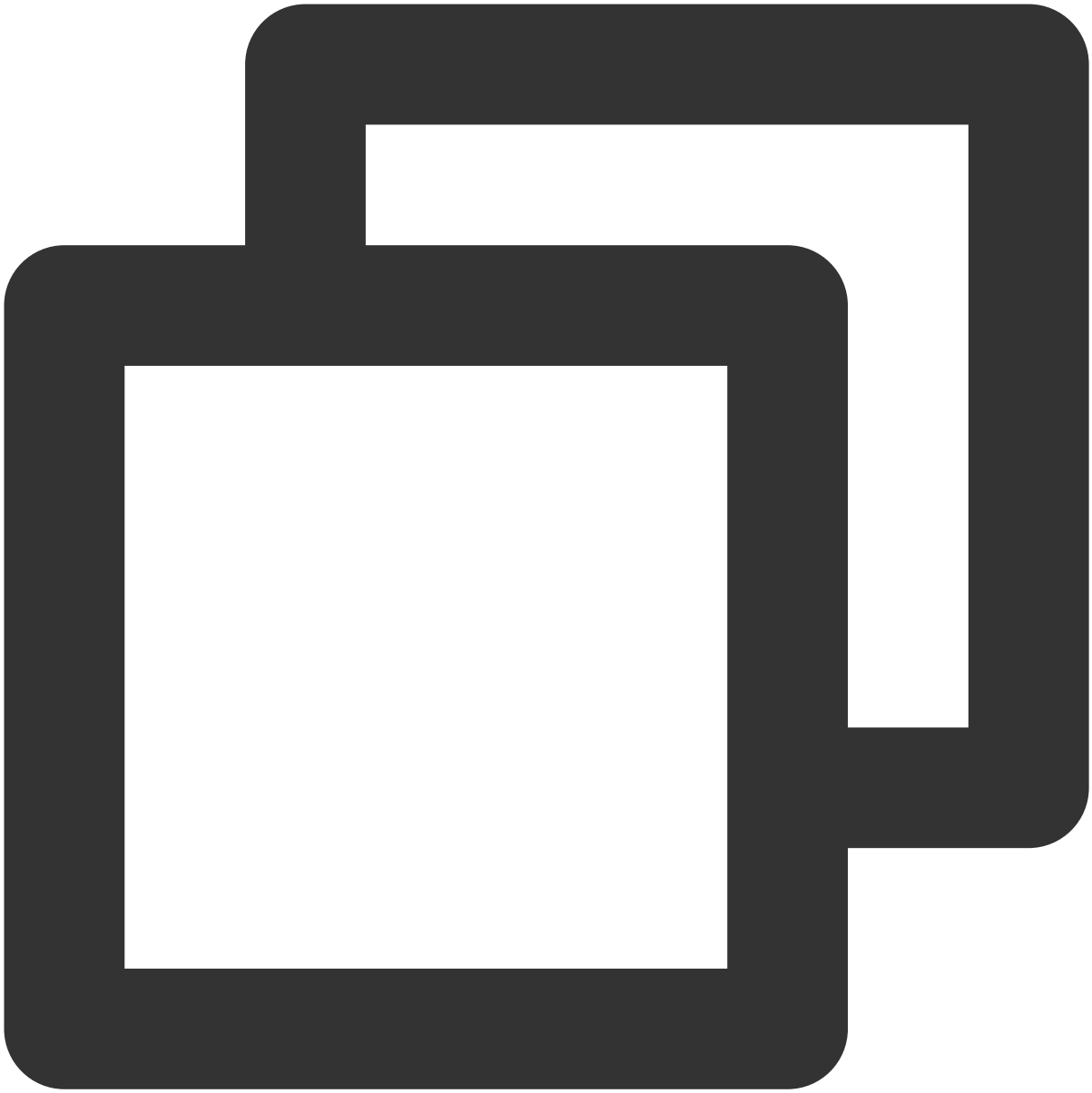
拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。



```
Future<TUIResult> reject()
```

ignore

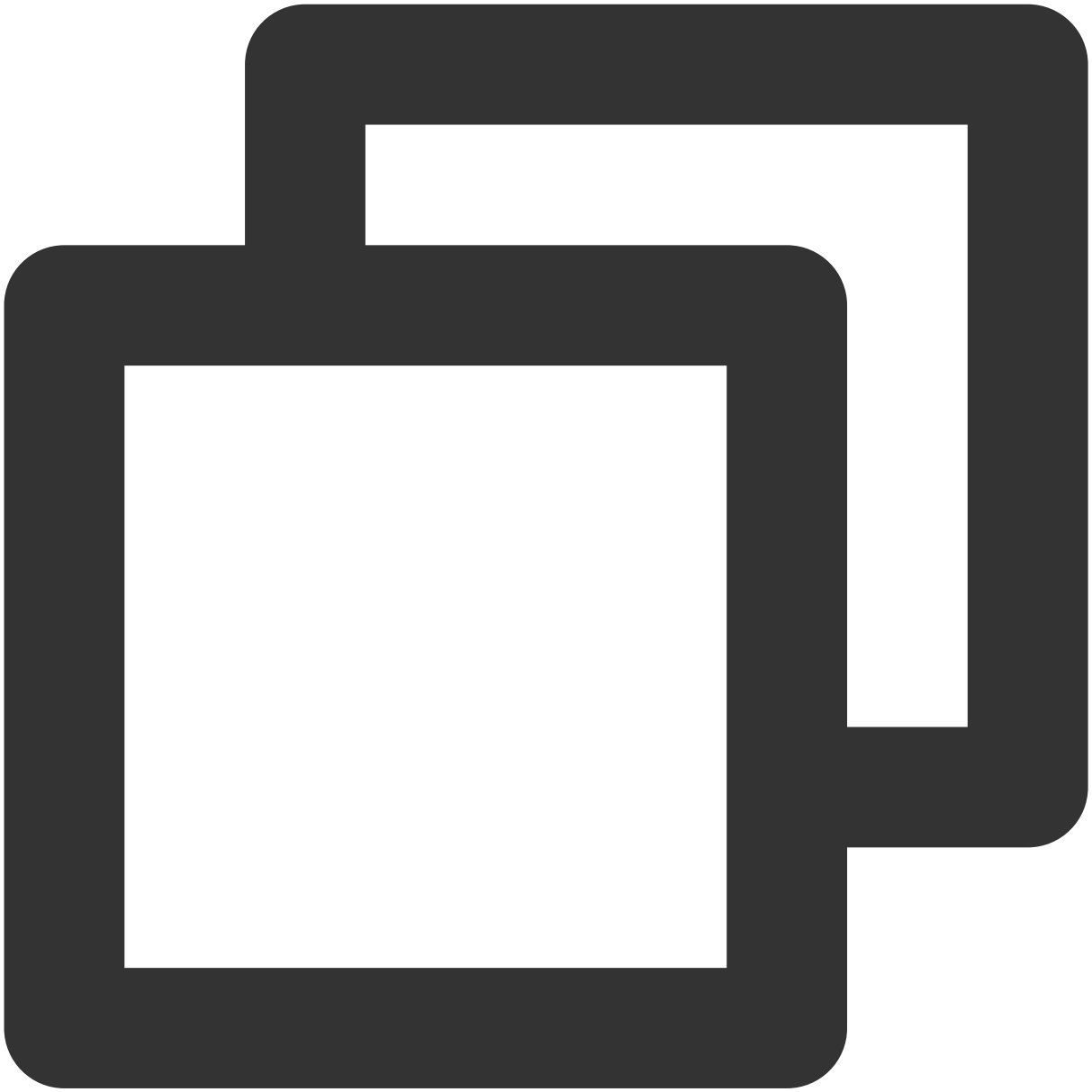
忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。



```
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。

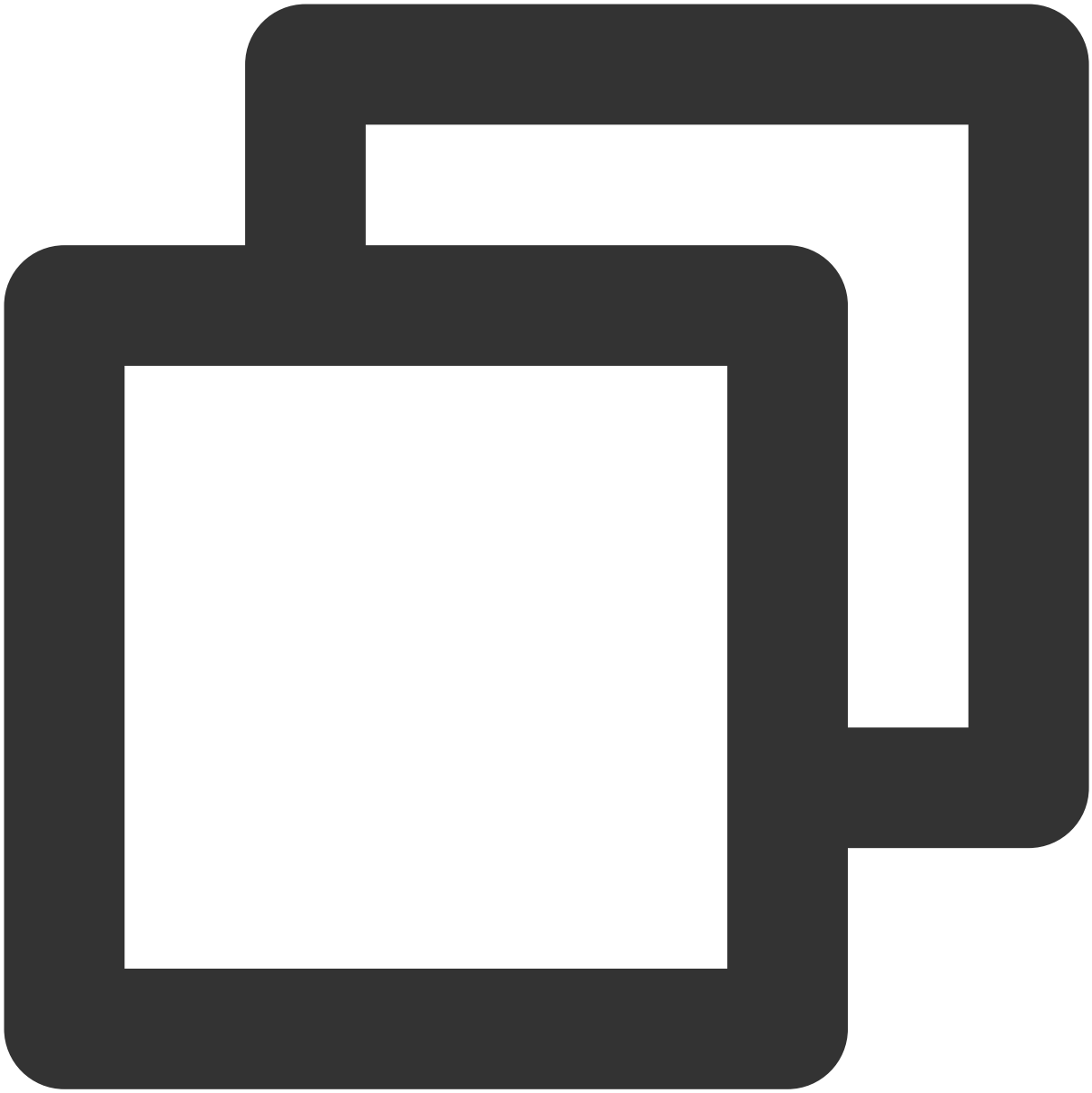


```
Future<TUIResult> hangup()
```

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。



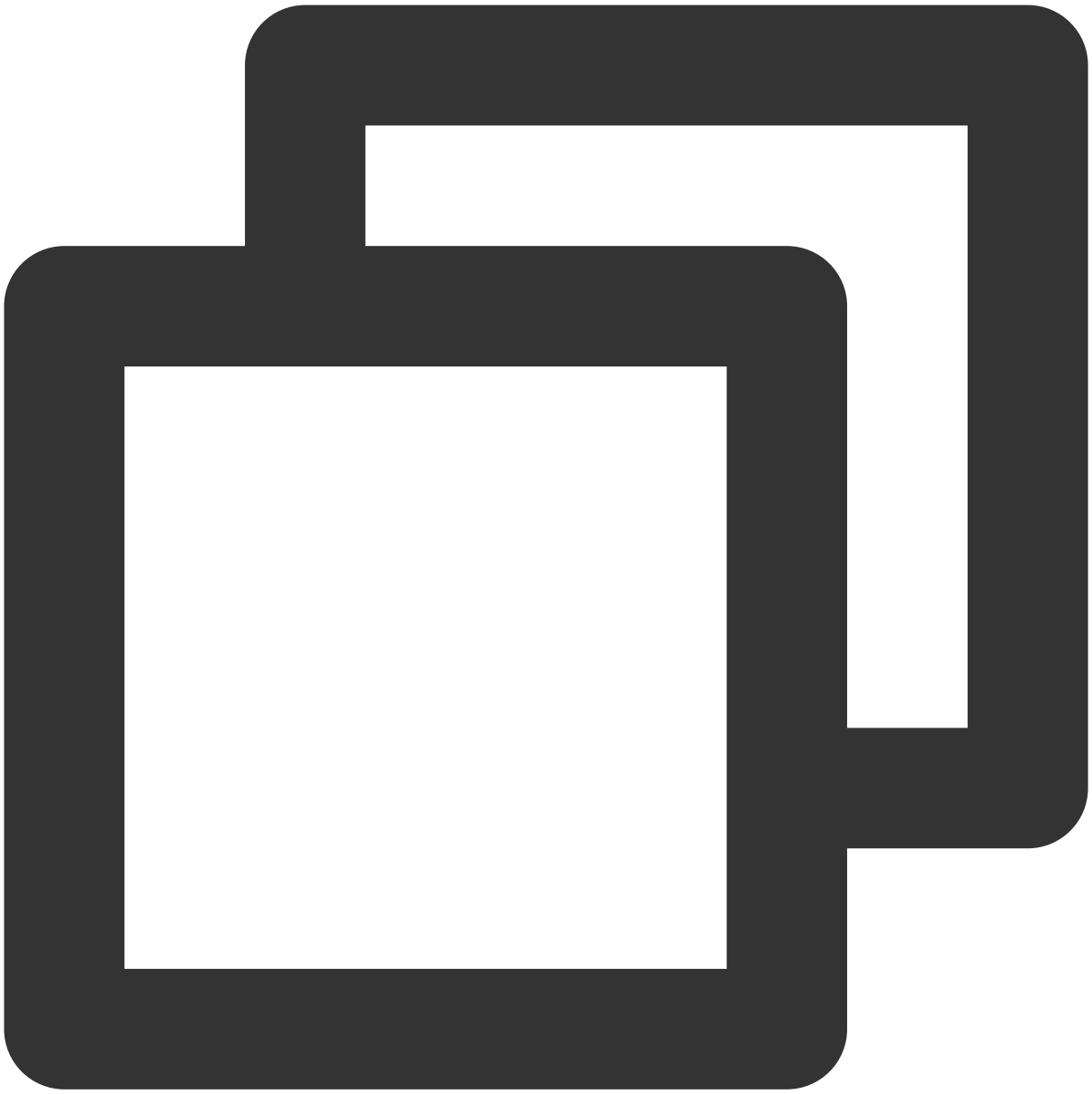
```
Future<void> iniviteUser(List<String> userIdList, TUICallParams params, TUIValueCal
```

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。



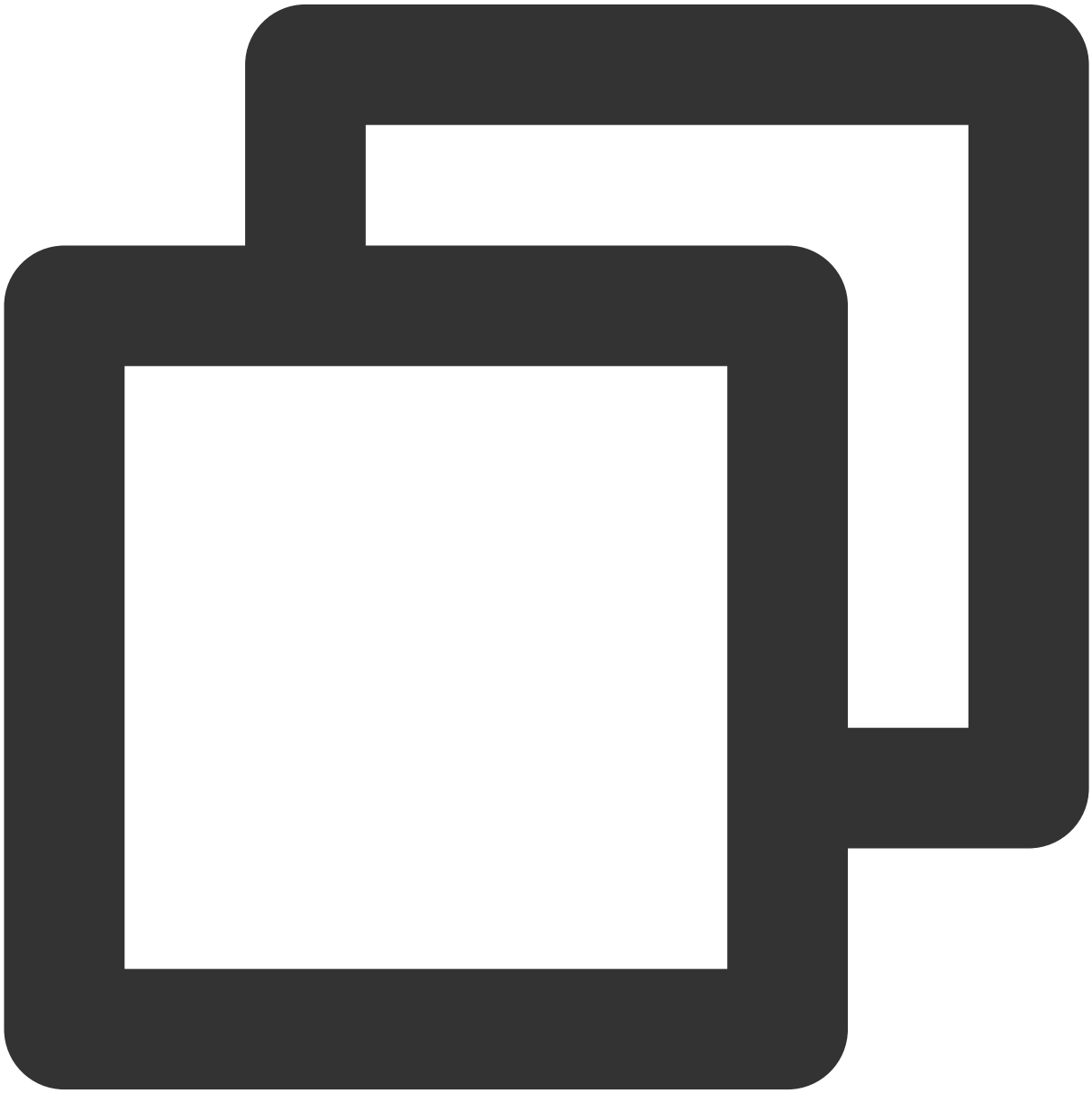
```
Future<TUIResult> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaTyp
```

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。

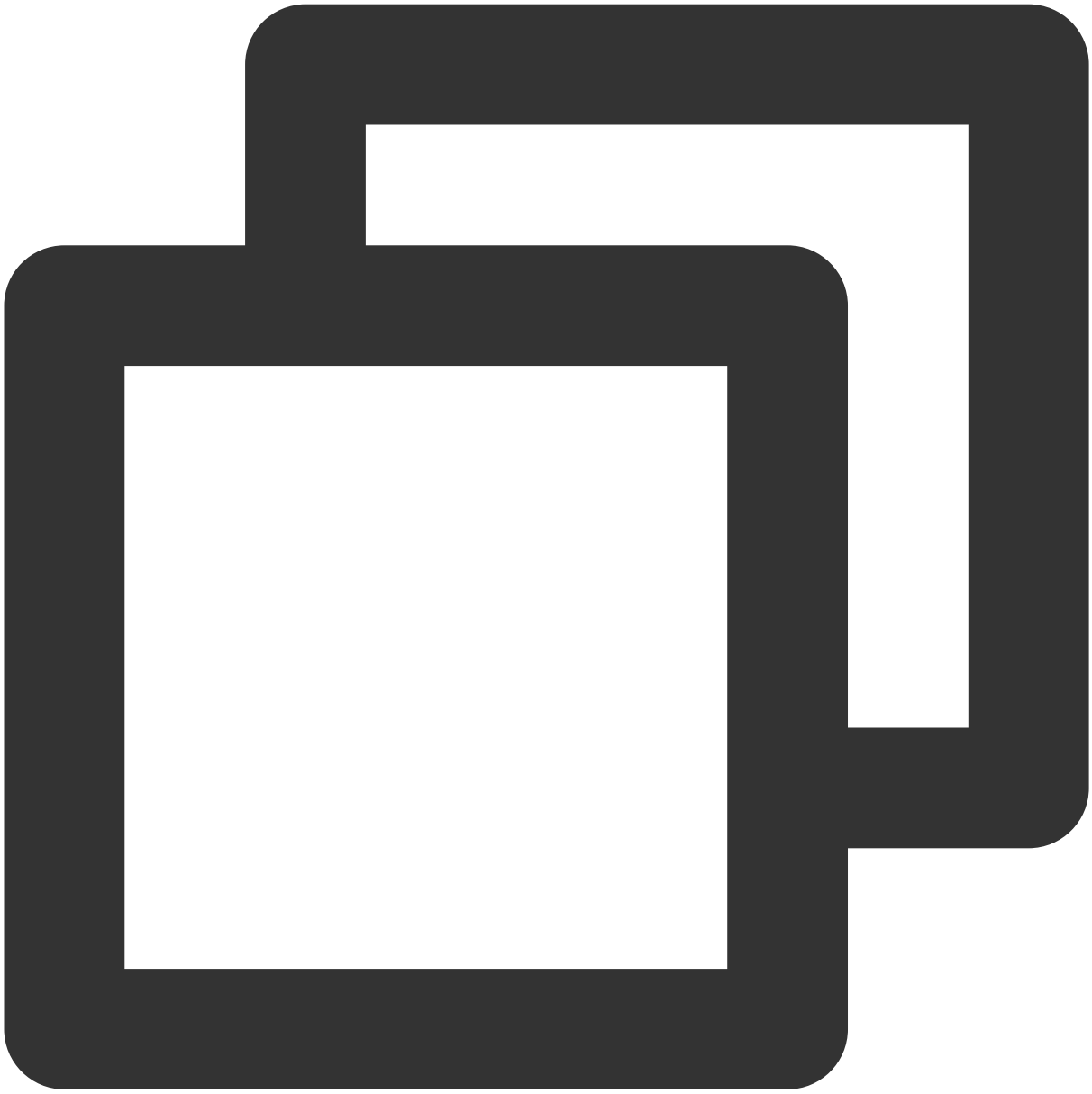


```
Future<void> switchCallMediaType(TUICallMediaType mediaType)
```

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。

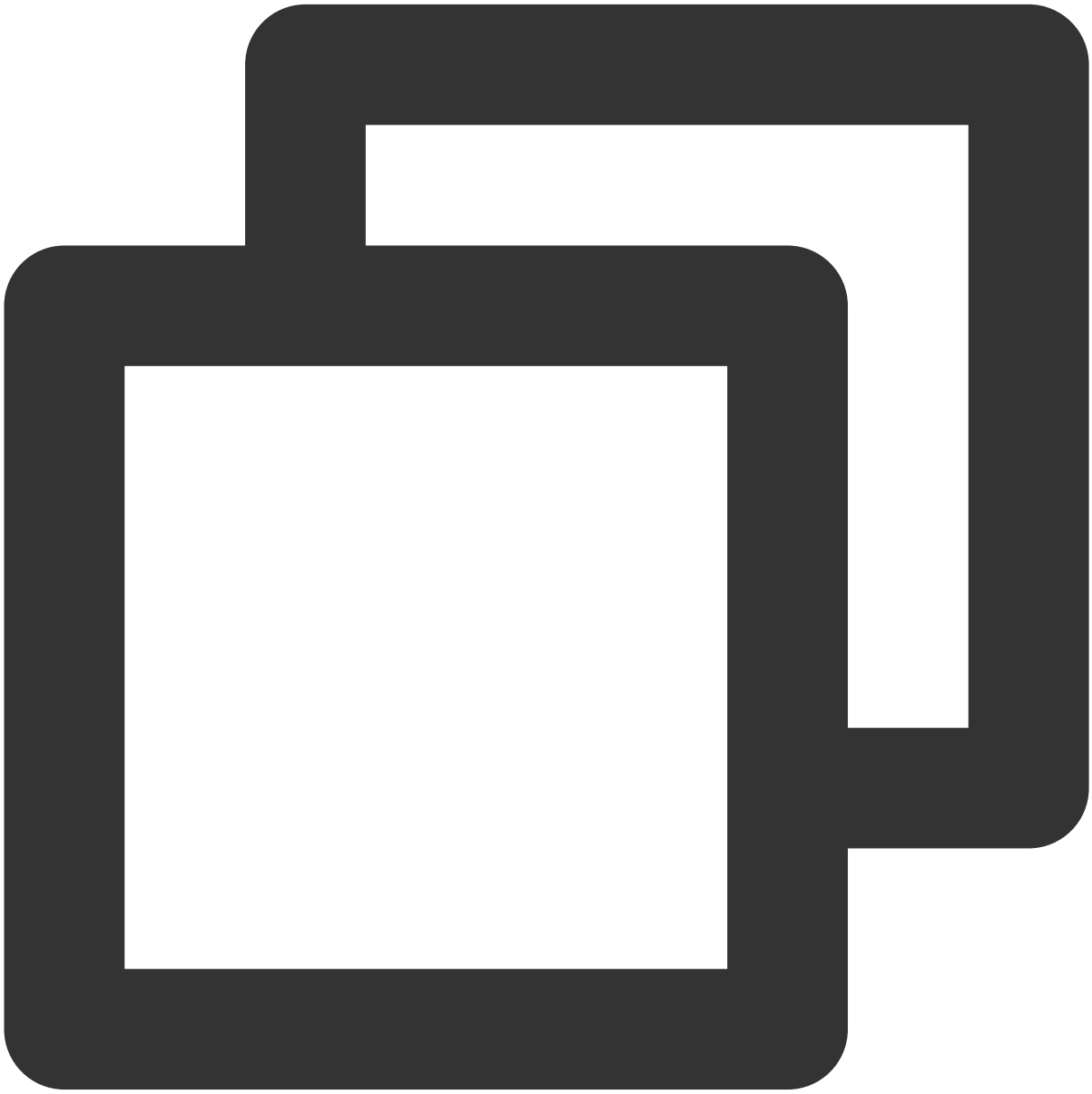


```
Future<void> startRemoteView(String userId, intviewId)
```

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。

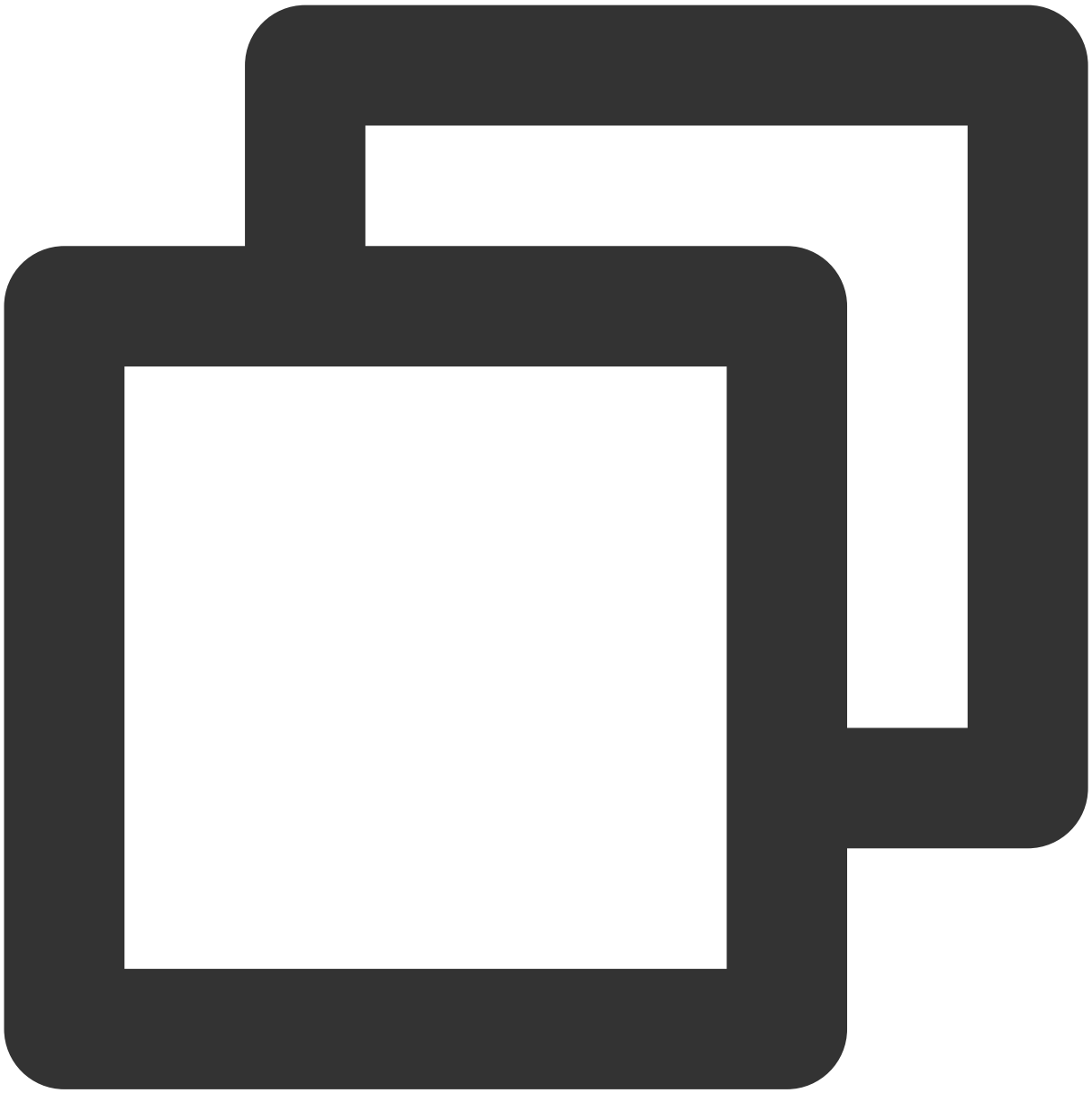


```
Future<void> stopRemoteView(String userId)
```

参数	类型	含义
userId	String	目标用户的 userId

openCamera

开启摄像头。

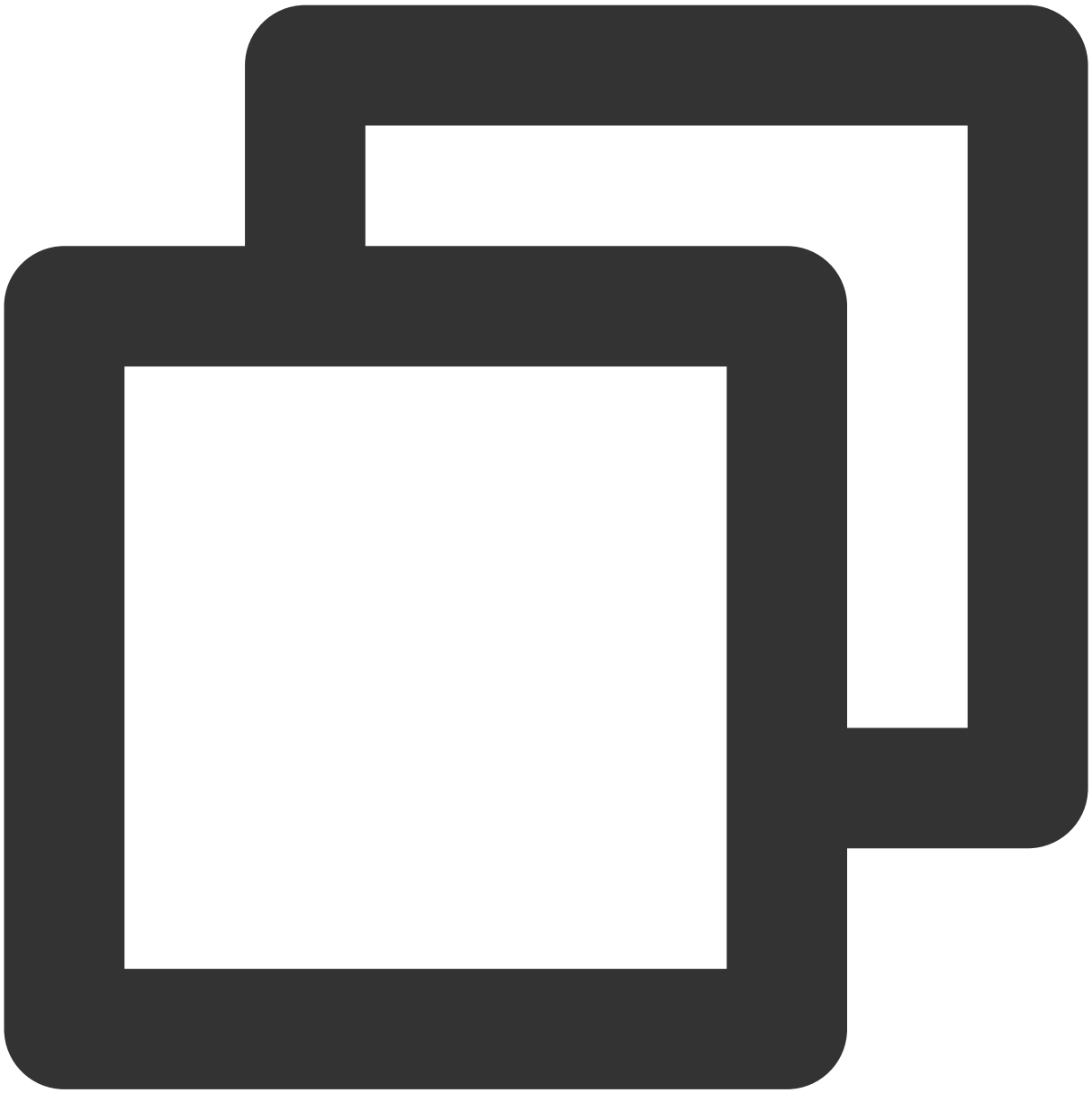


```
Future<TUIResult> openCamera(TUICamera camera, int? viewId)
```

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

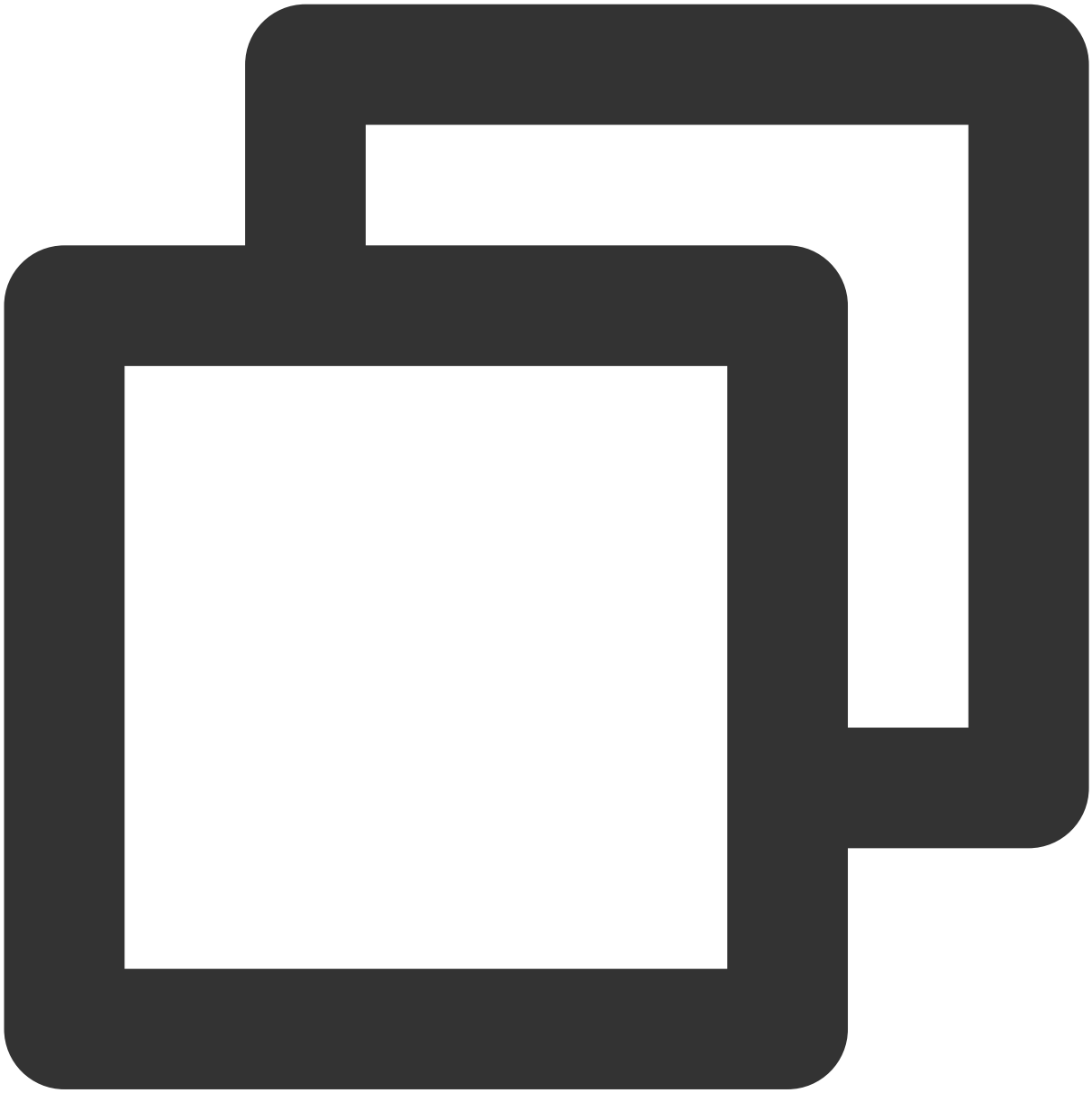
关闭摄像头。



```
Future<void> closeCamera()
```

switchCamera

切换前后摄像头。

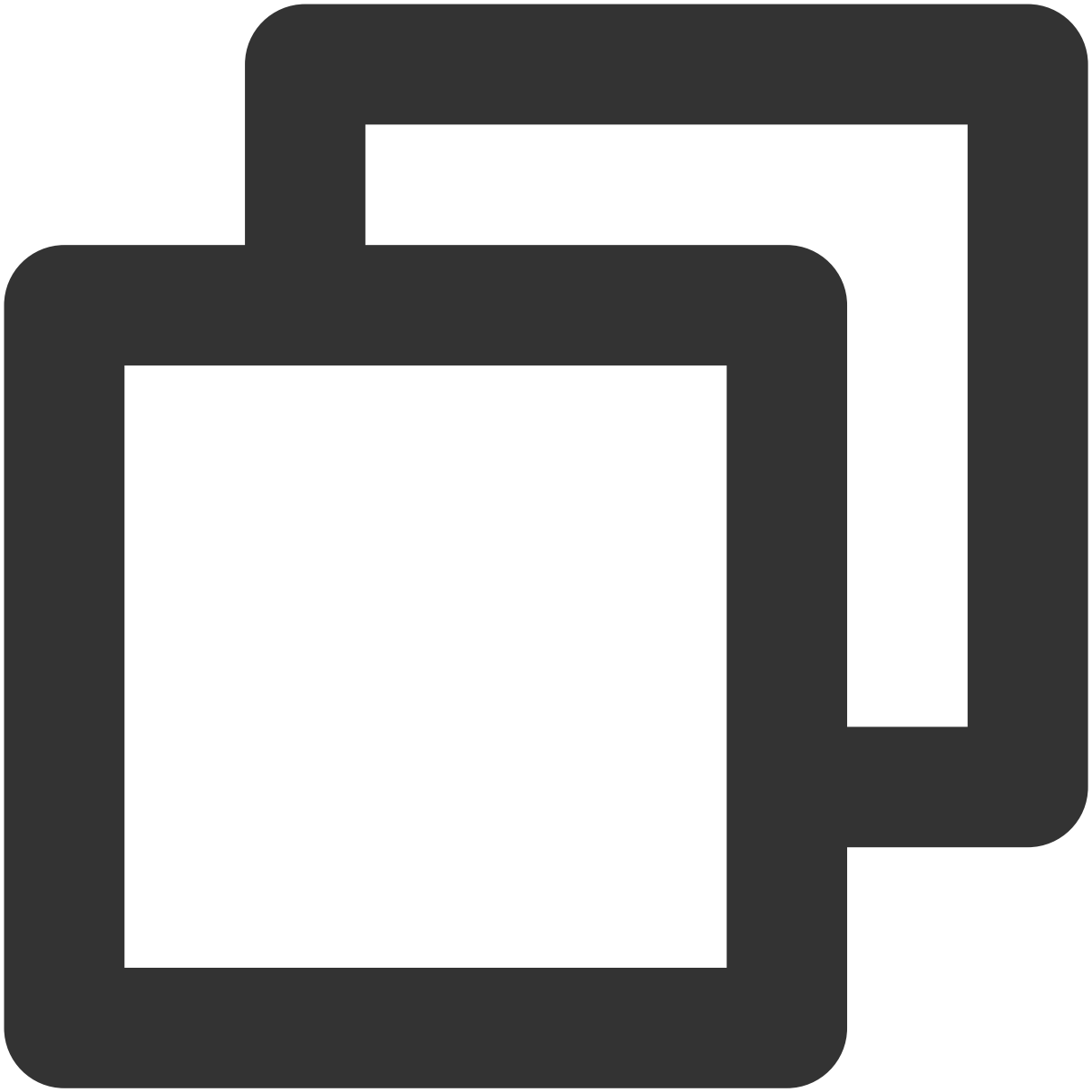


```
Future<void> switchCamera(TUICamera camera)
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头

openMicrophone

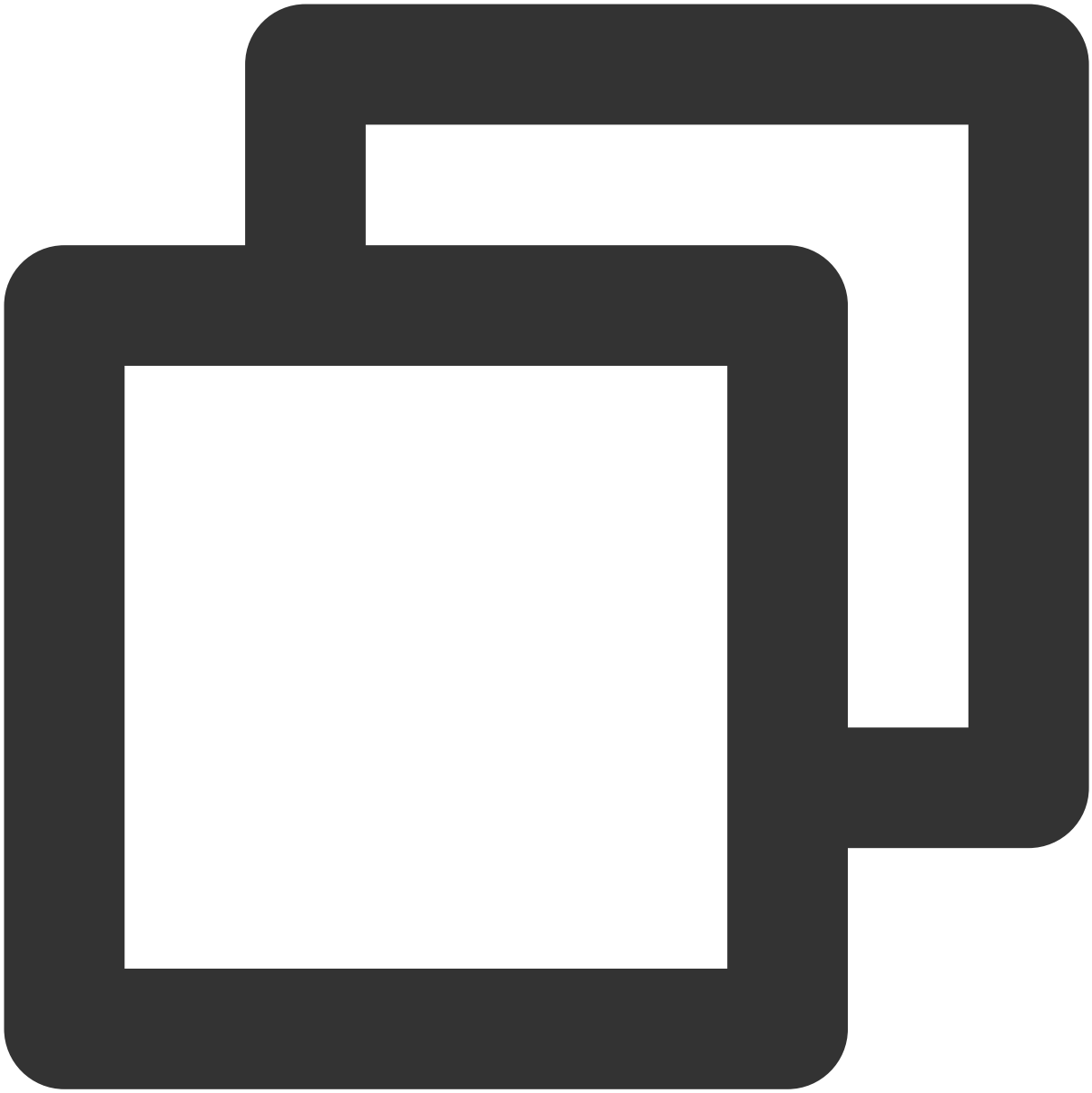
打开麦克风。



```
Future<TUIResult> openMicrophone()
```

closeMicrophone

关闭麦克风。

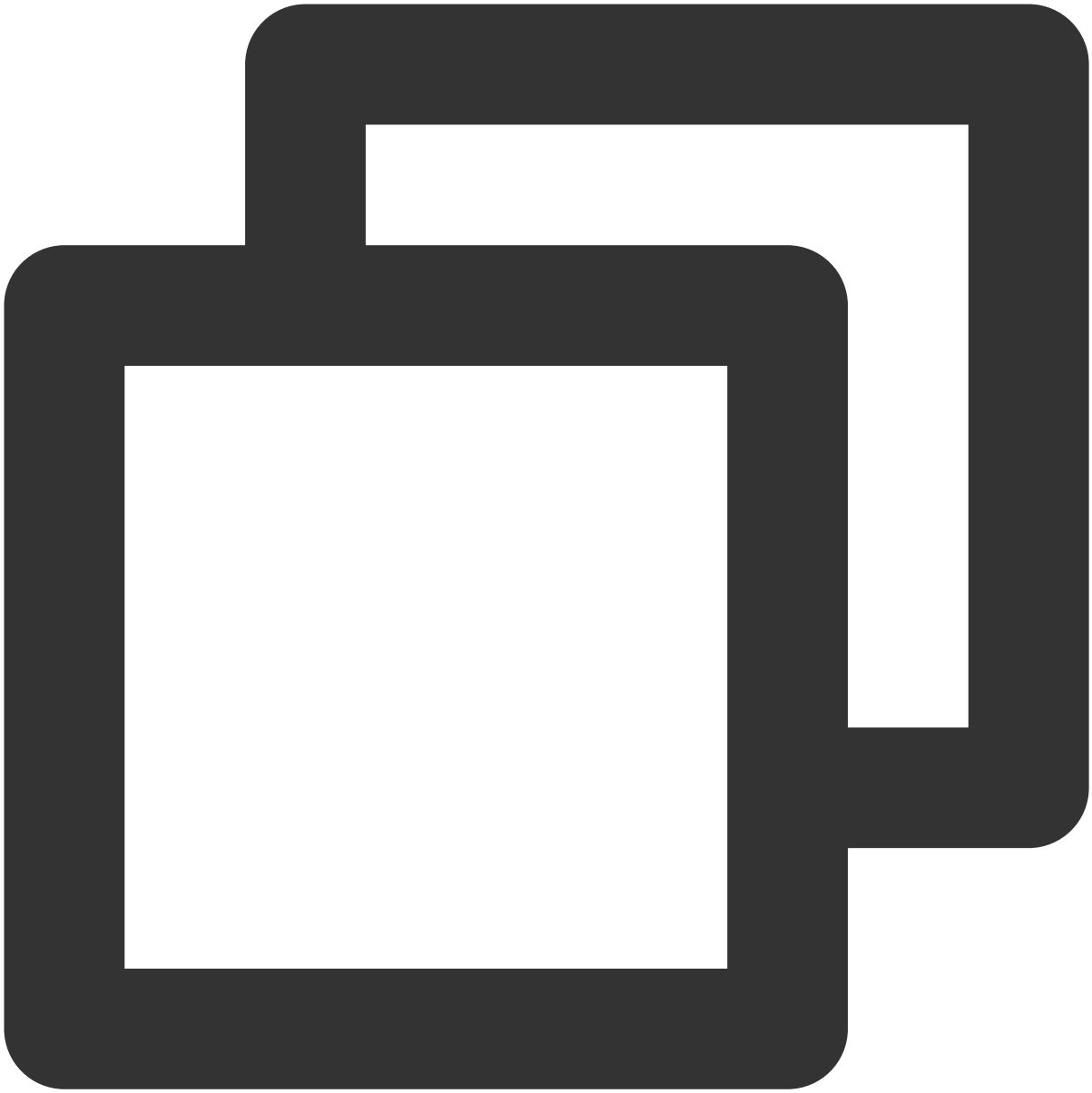


```
Future<void> closeMicrophone()
```

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。



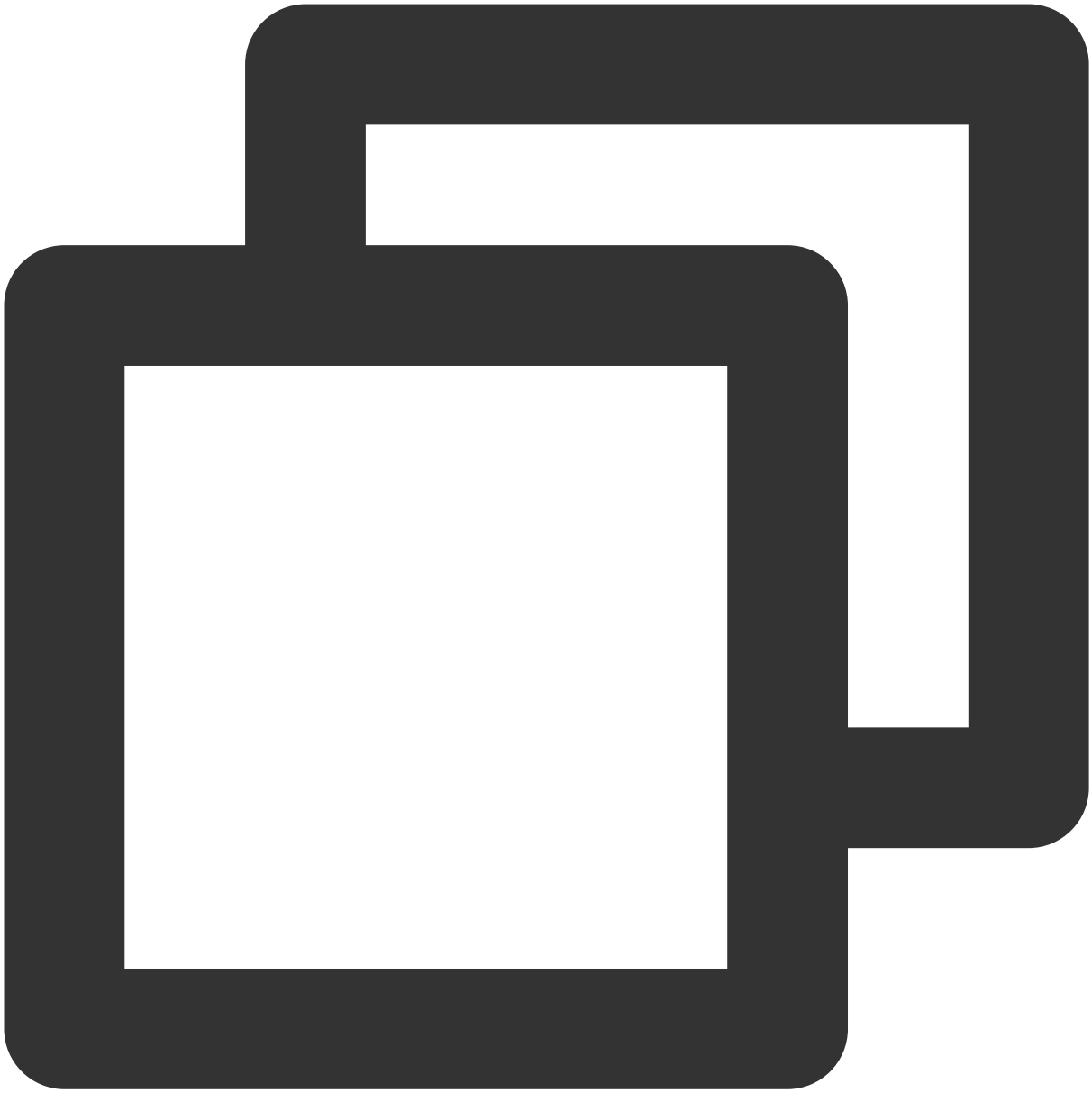
```
Future<void> selectAudioPlaybackDevice(TUIAudioPlaybackDevice device)
```

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。

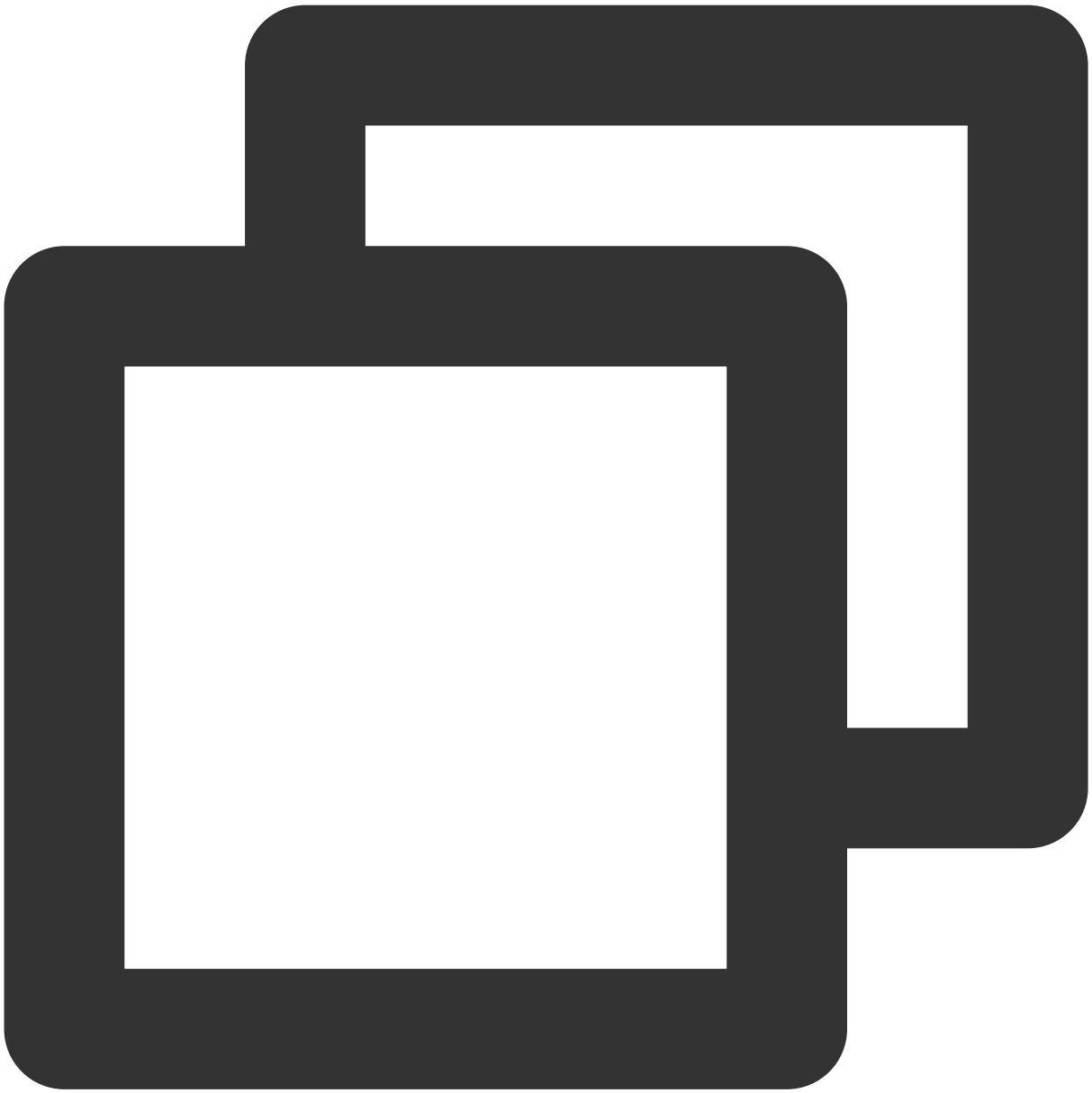


```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

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

enableMultiDeviceAbility

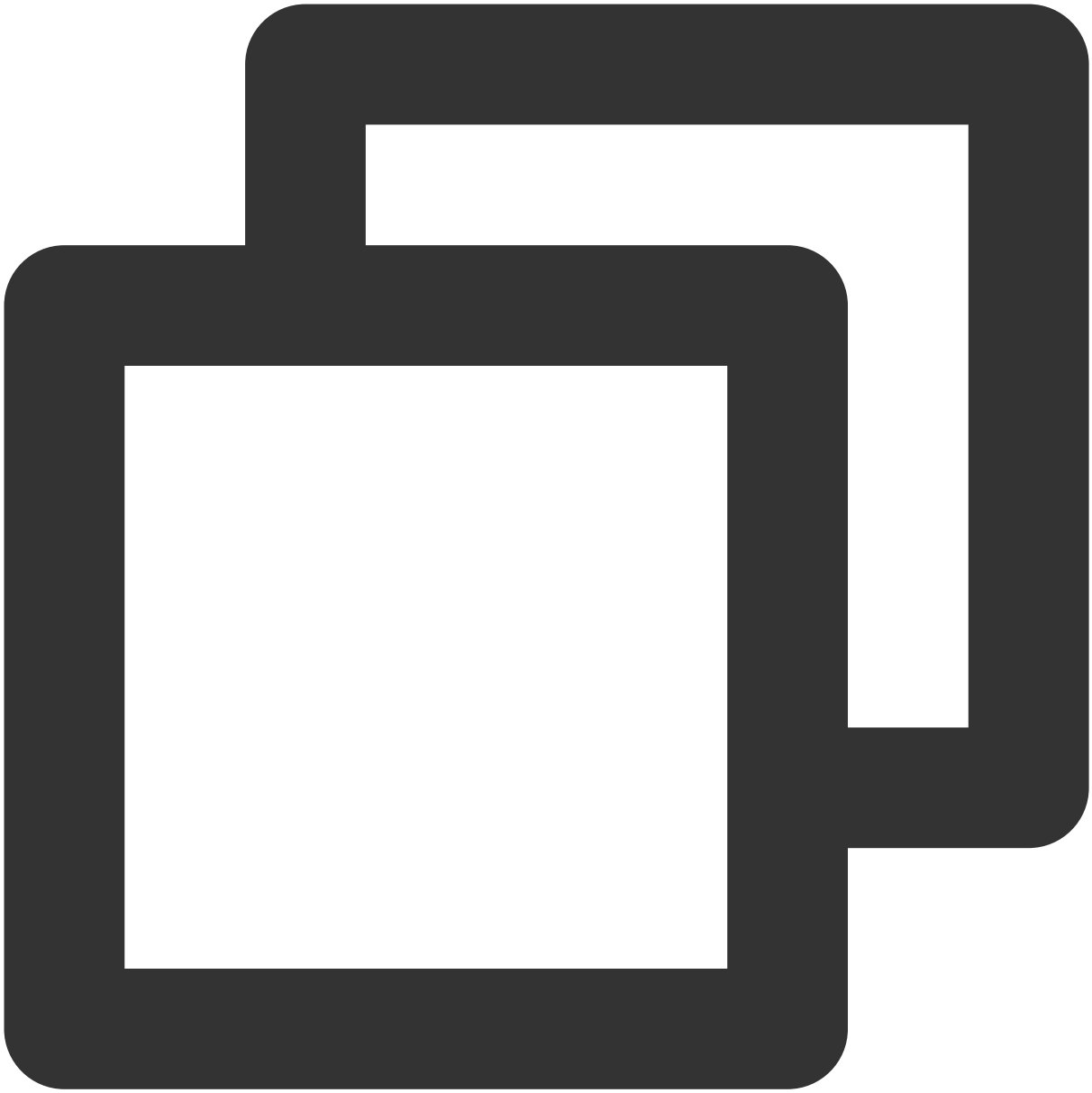
开启/关闭 TUICallEngine 的多设备登录模式 （尊享版套餐支持）。



```
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)
```

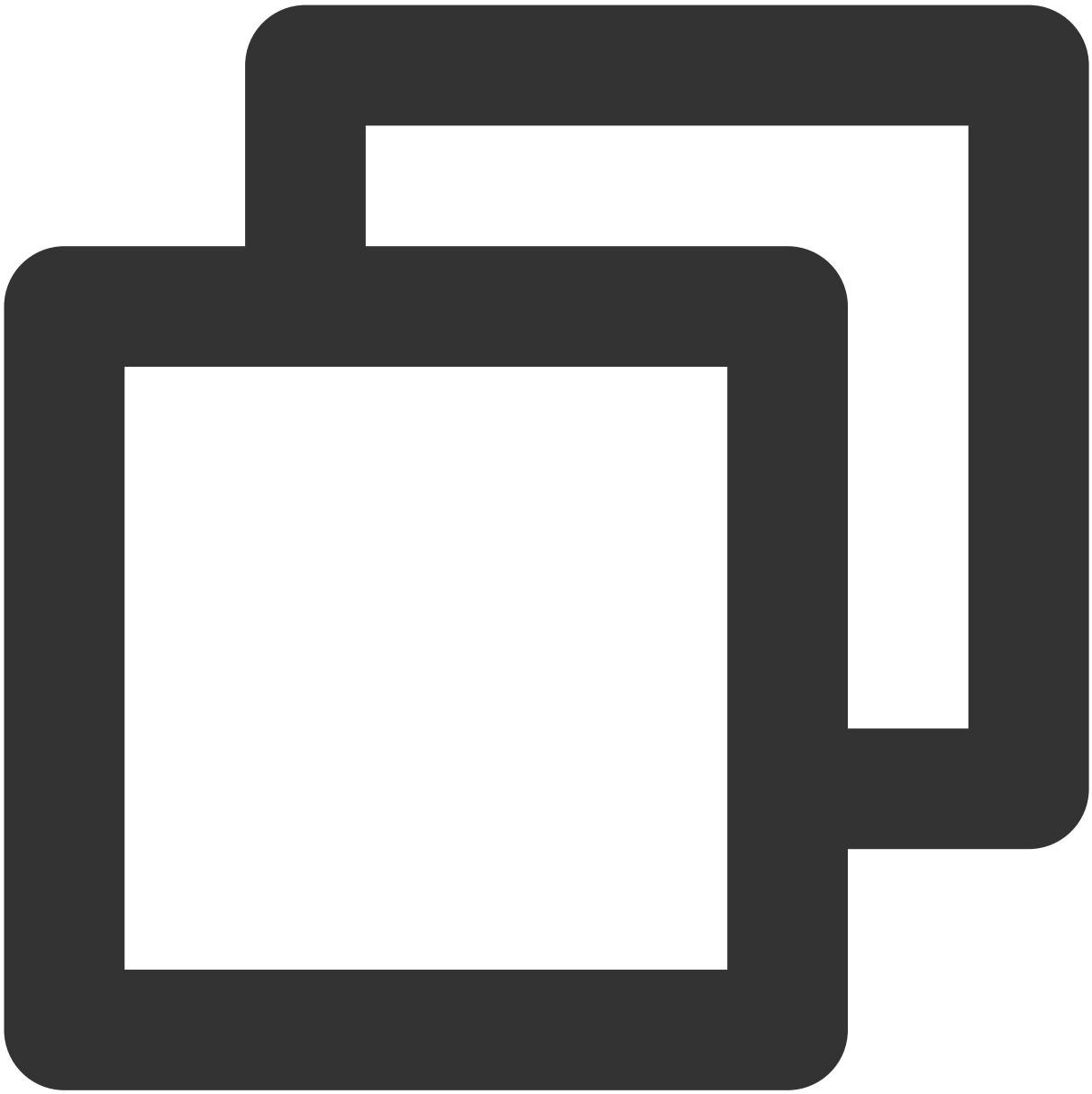
参数如下表所示：

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。



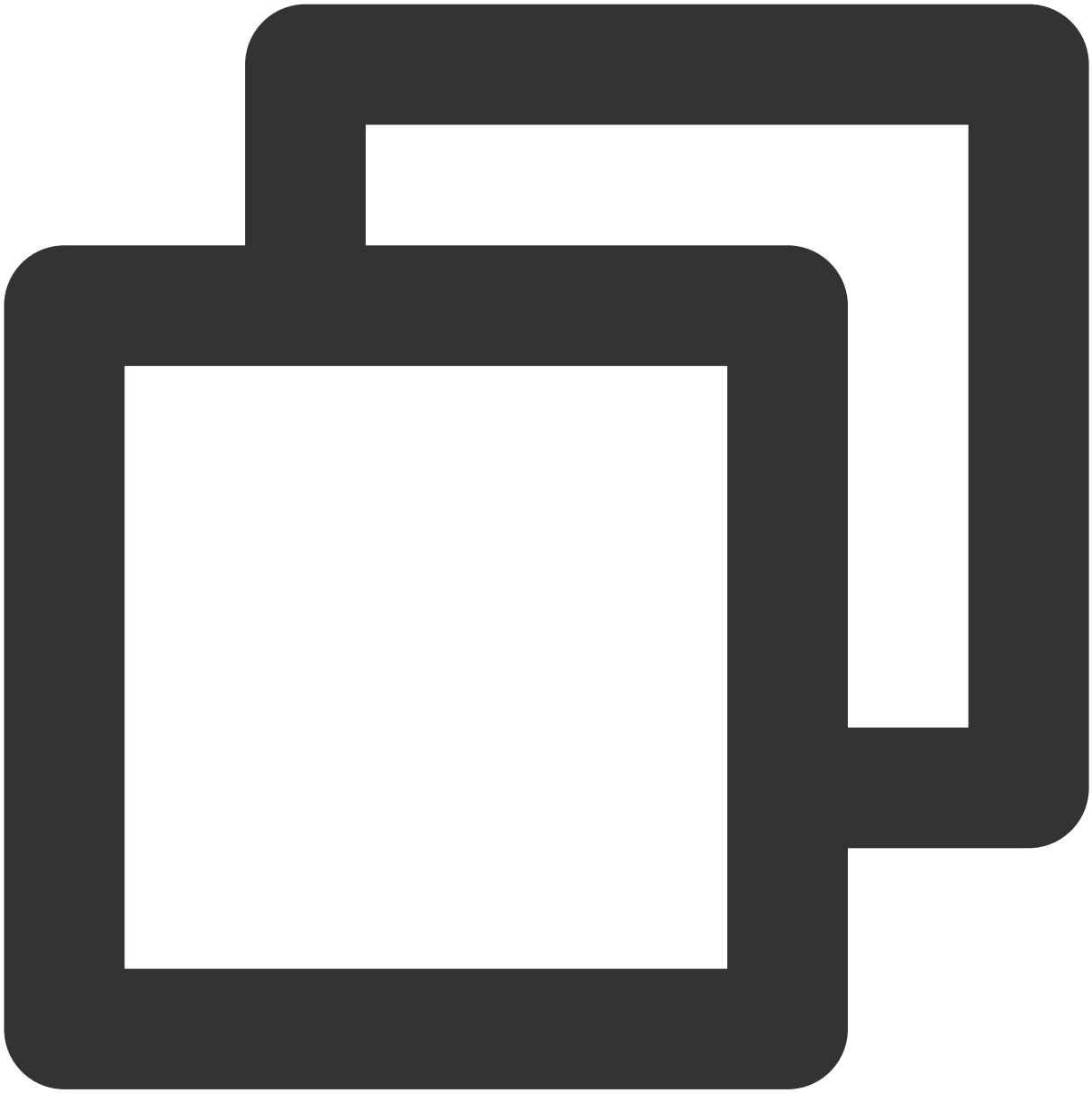
```
Future<TUIResult> setVideoEncoderParams(VideoEncoderParams params)
```

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。



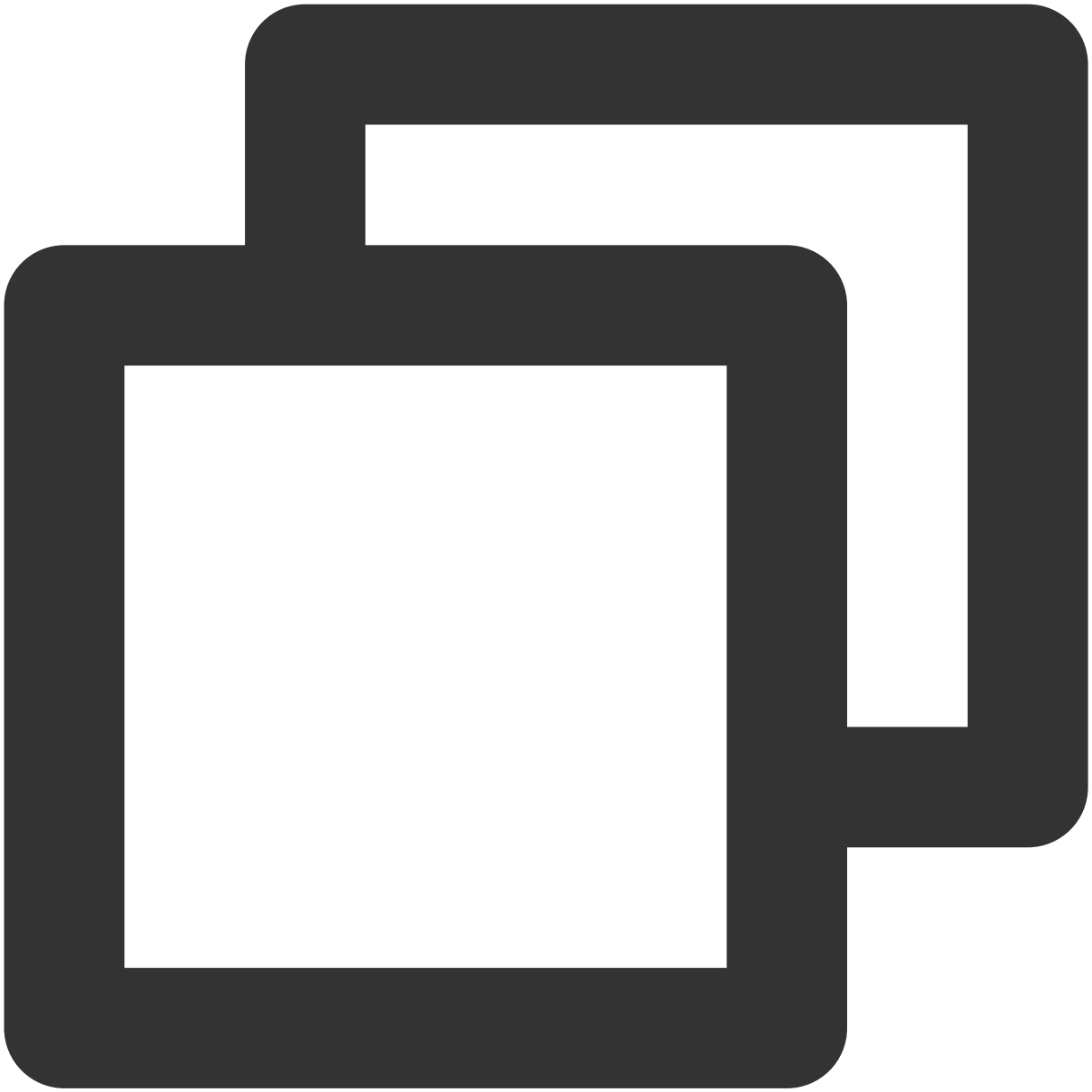
```
Future<void> queryRecentCalls(TUICallRecentCallsFilter filter, TUIValueCallback cal
```

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。



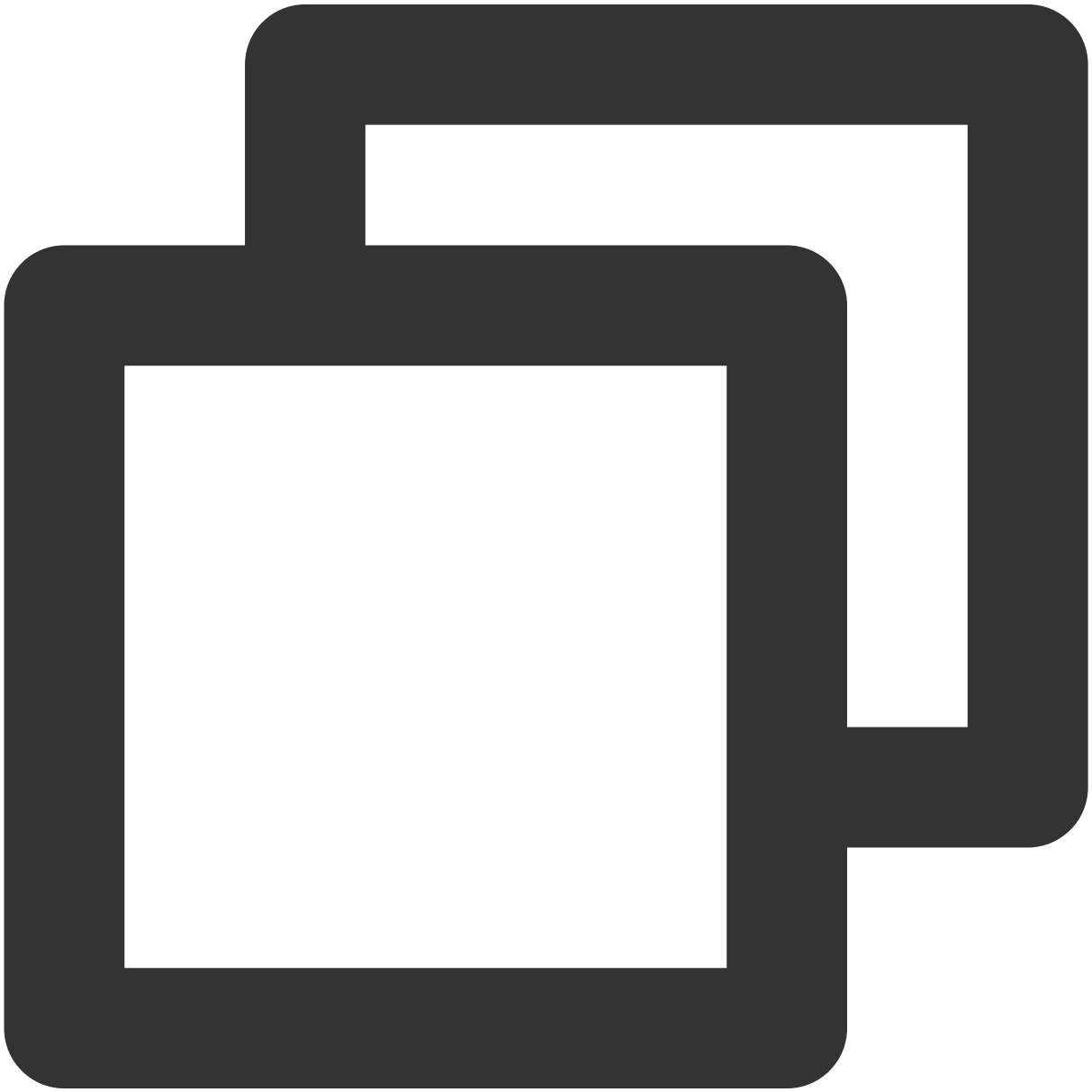
```
Future<void> deleteRecordCalls(List<String> callIdList, TUIValueCallback callback)
```

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。



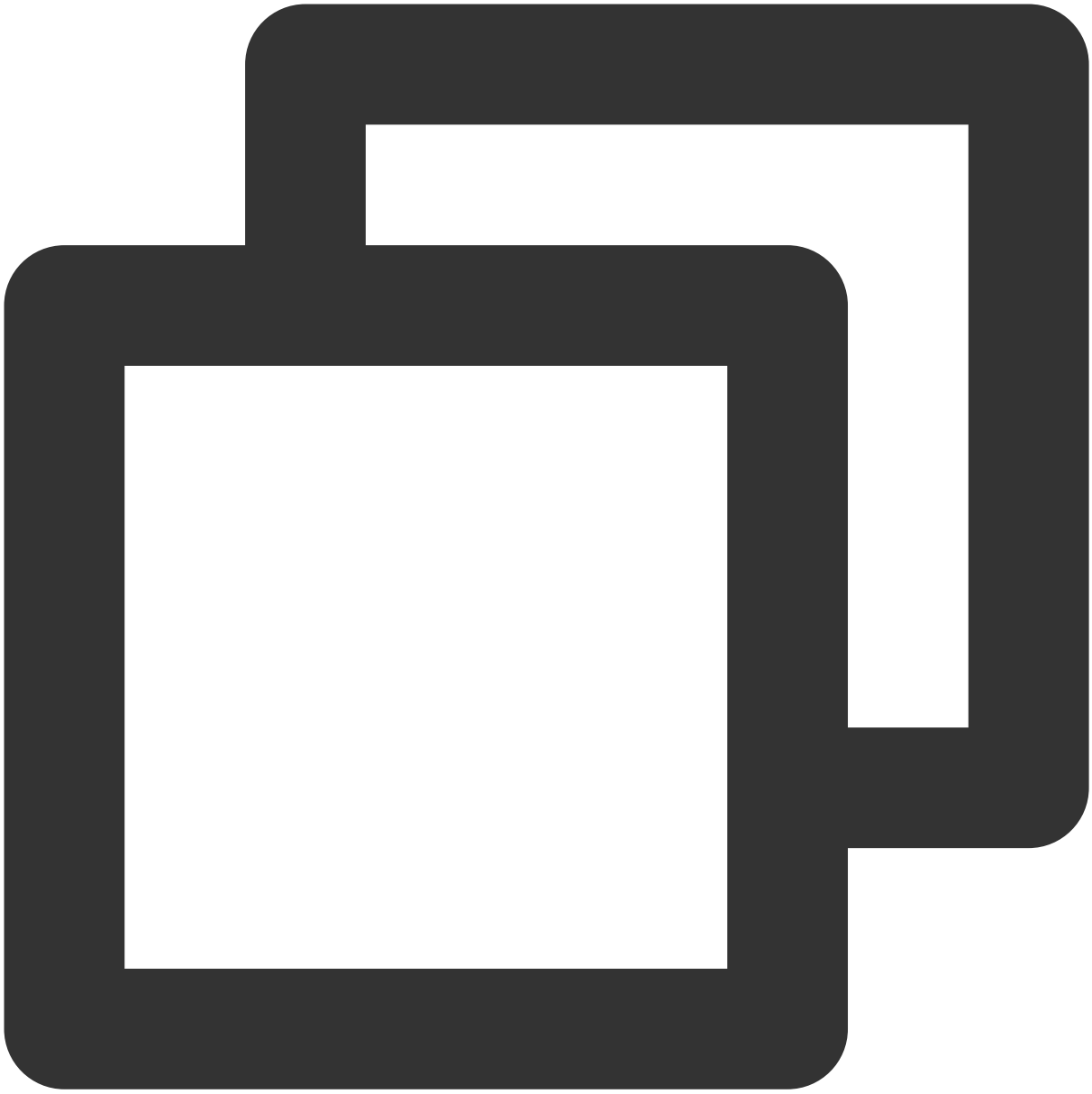
```
Future<TUIResult> setBeautyLevel(double level)
```

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

setBlurBackground

设置视频模糊效果。



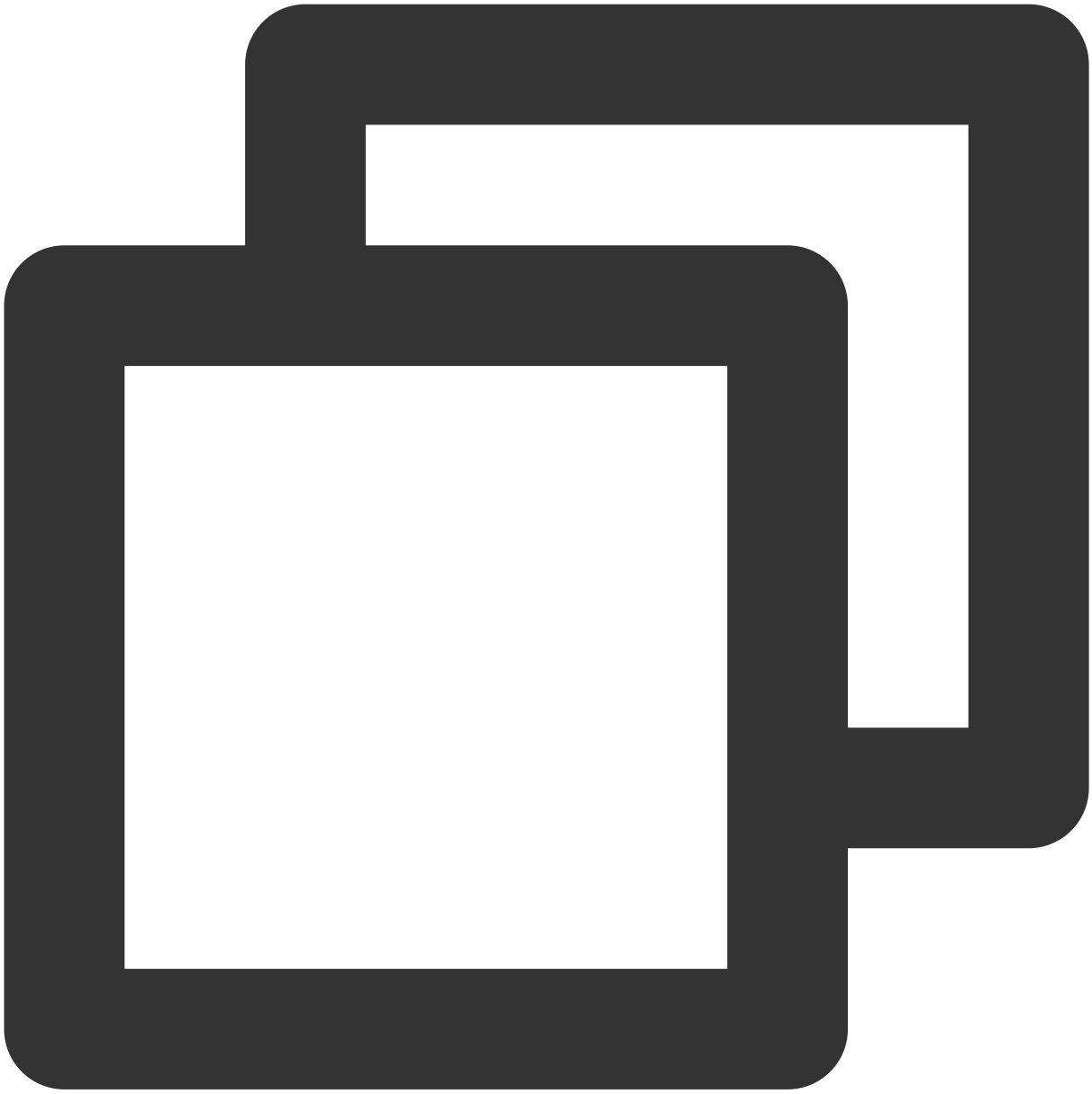
```
void setBlurBackground(int level, Function(int code, String message)? errorCallback
```

参数如下表所示：

参数	类型	含义
level	int	0：关闭，1：低，2：中，3：高。

setVirtualBackground

设置虚拟背景图片。



```
void setVirtualBackground(String imagePath, Function(int code, String message)? err
```

参数	类型	含义
imagePath	String	图片文件名，需要将文件添加至主工程的 assets 资源中。

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

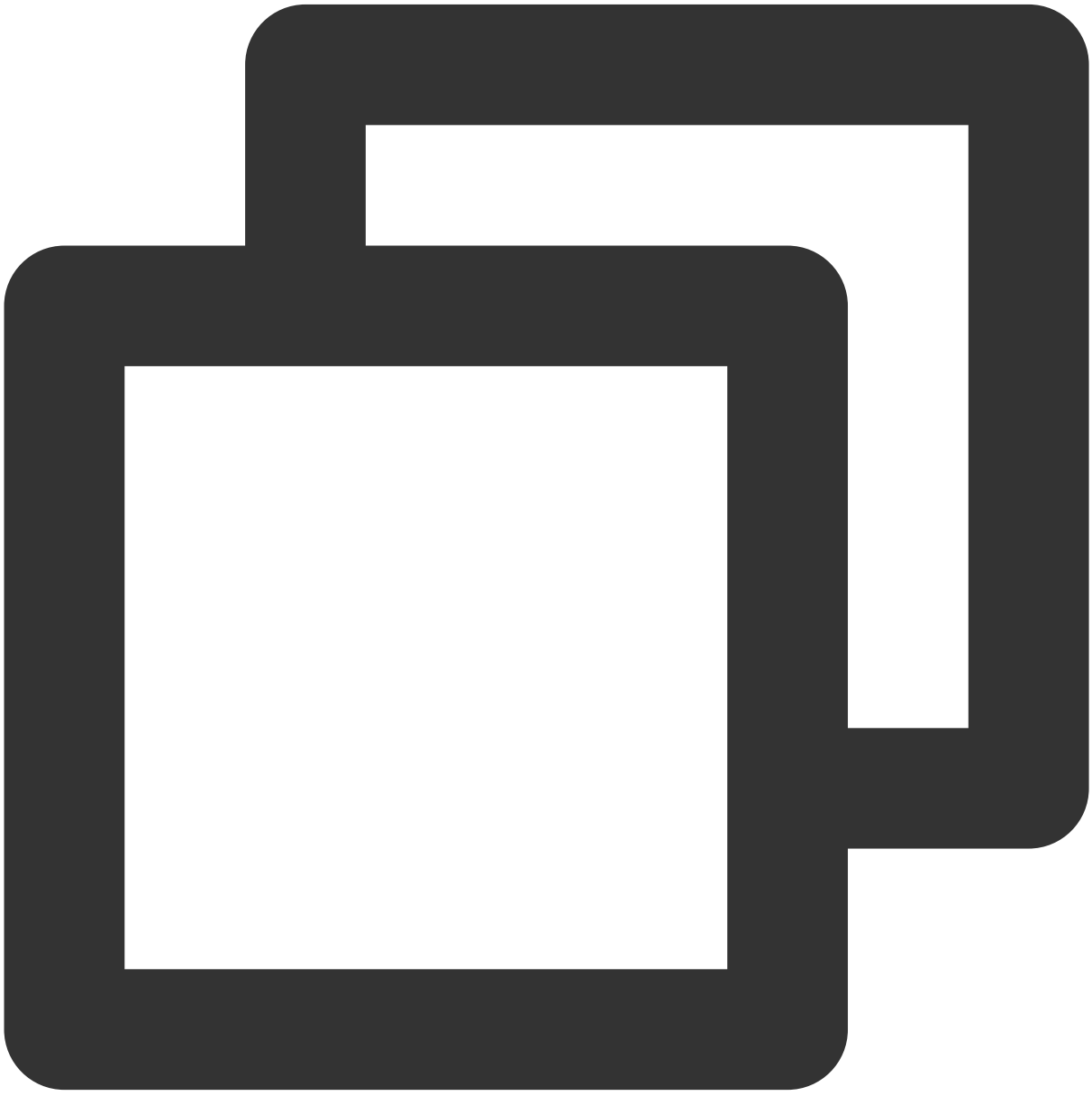
API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable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

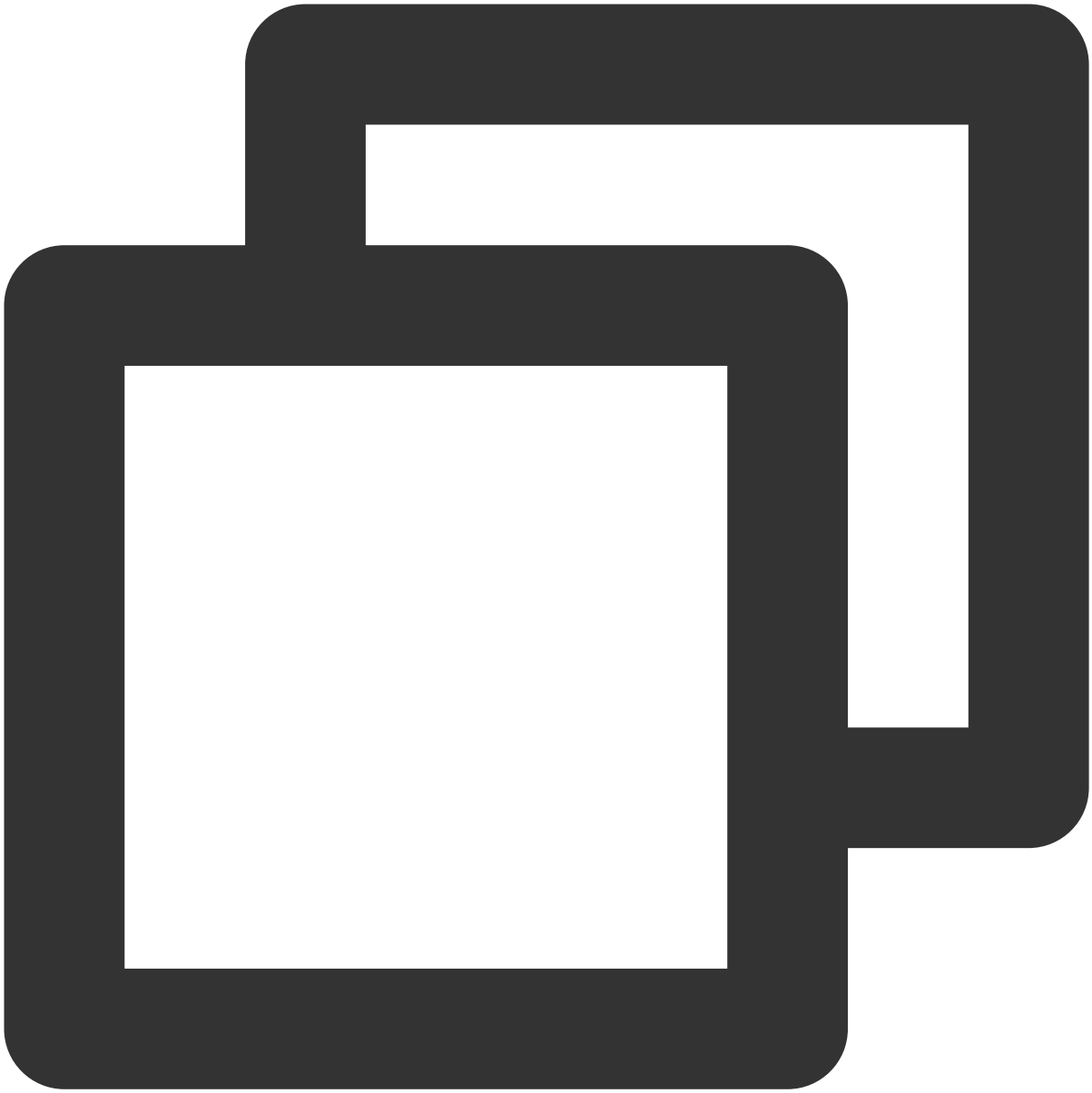
```
    },onCallCancelled: (String callerId) {
      //您的回调处理逻辑
    }, onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole
      //您的回调处理逻辑
    }, onCallEnd: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole ca
      //您的回调处理逻辑
    }, onCallMediaTypeChanged: (TUICallMediaType 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
    }
));
```

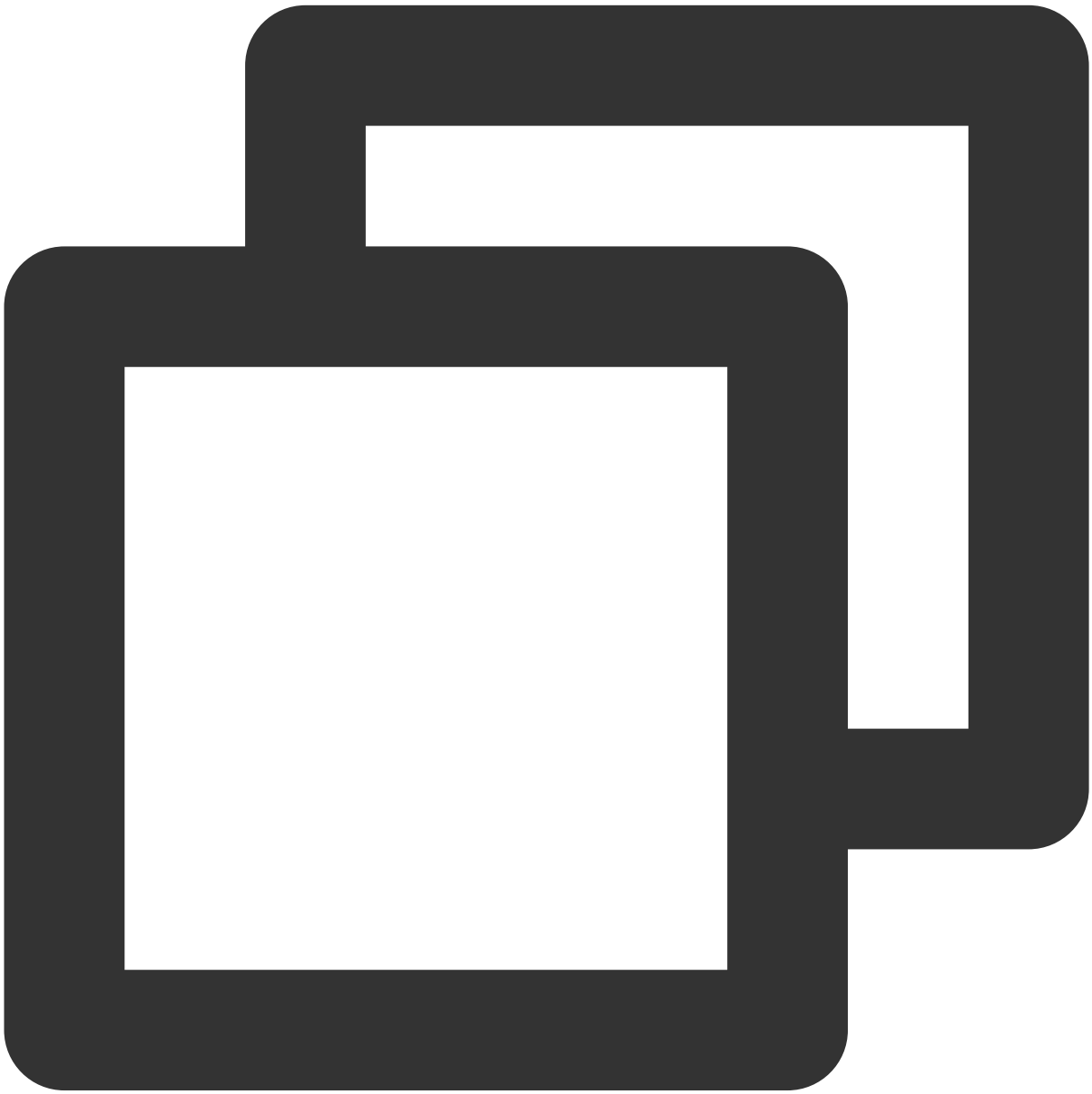
参数如下表所示：

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。

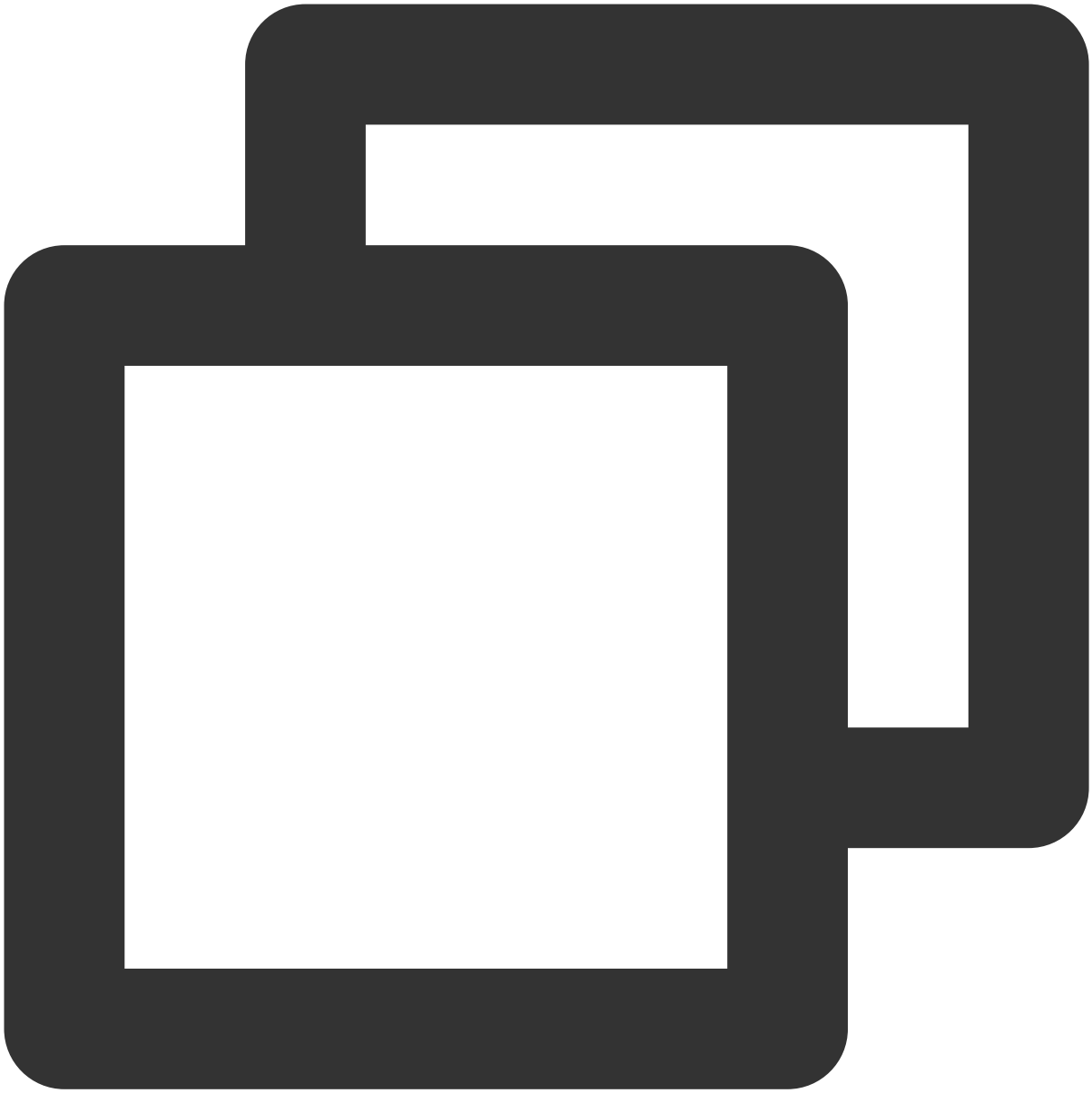
主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）

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）



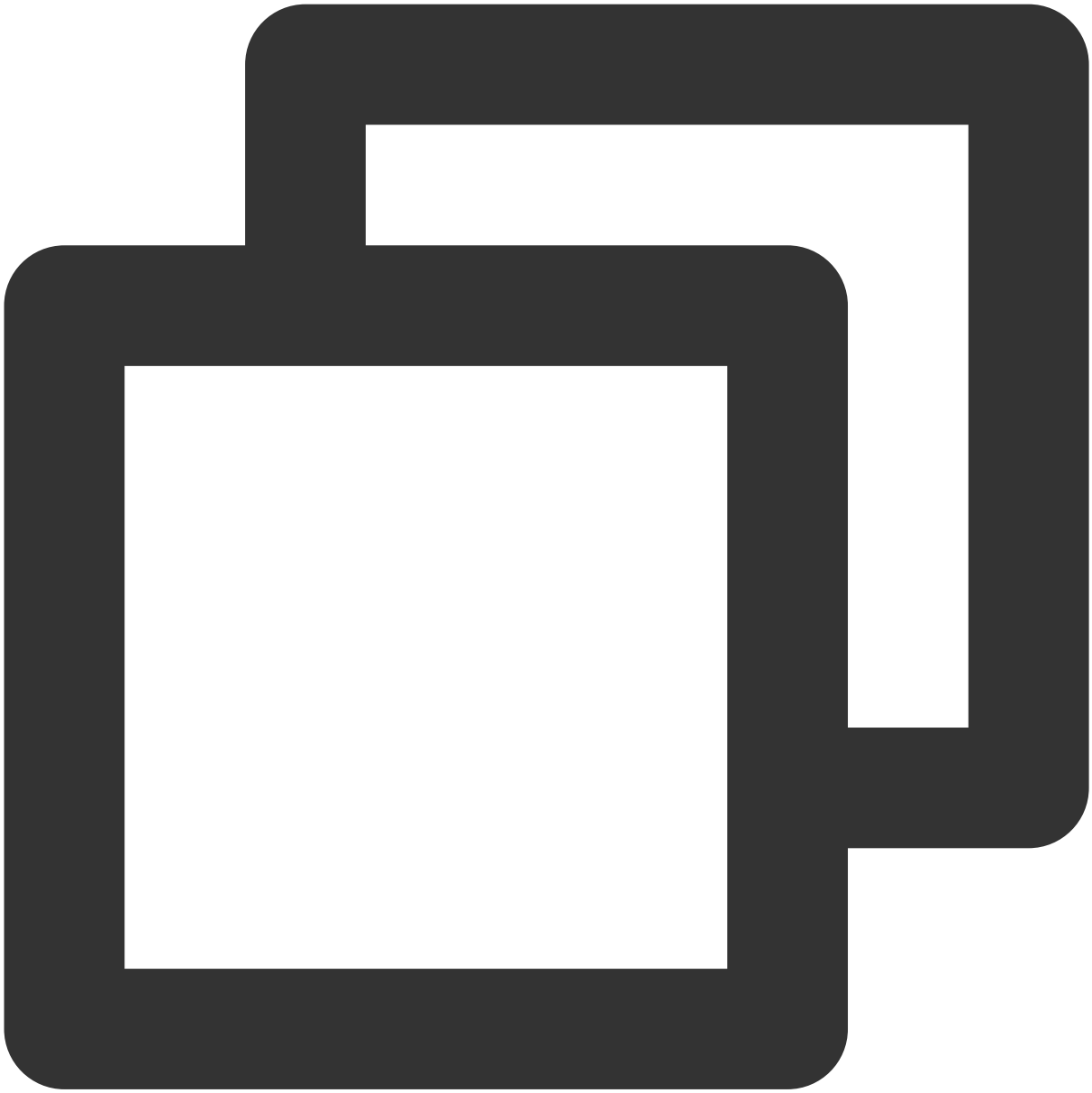
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallCancelled: (String userId) {
      //您的回调处理逻辑
    }
));
```

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。



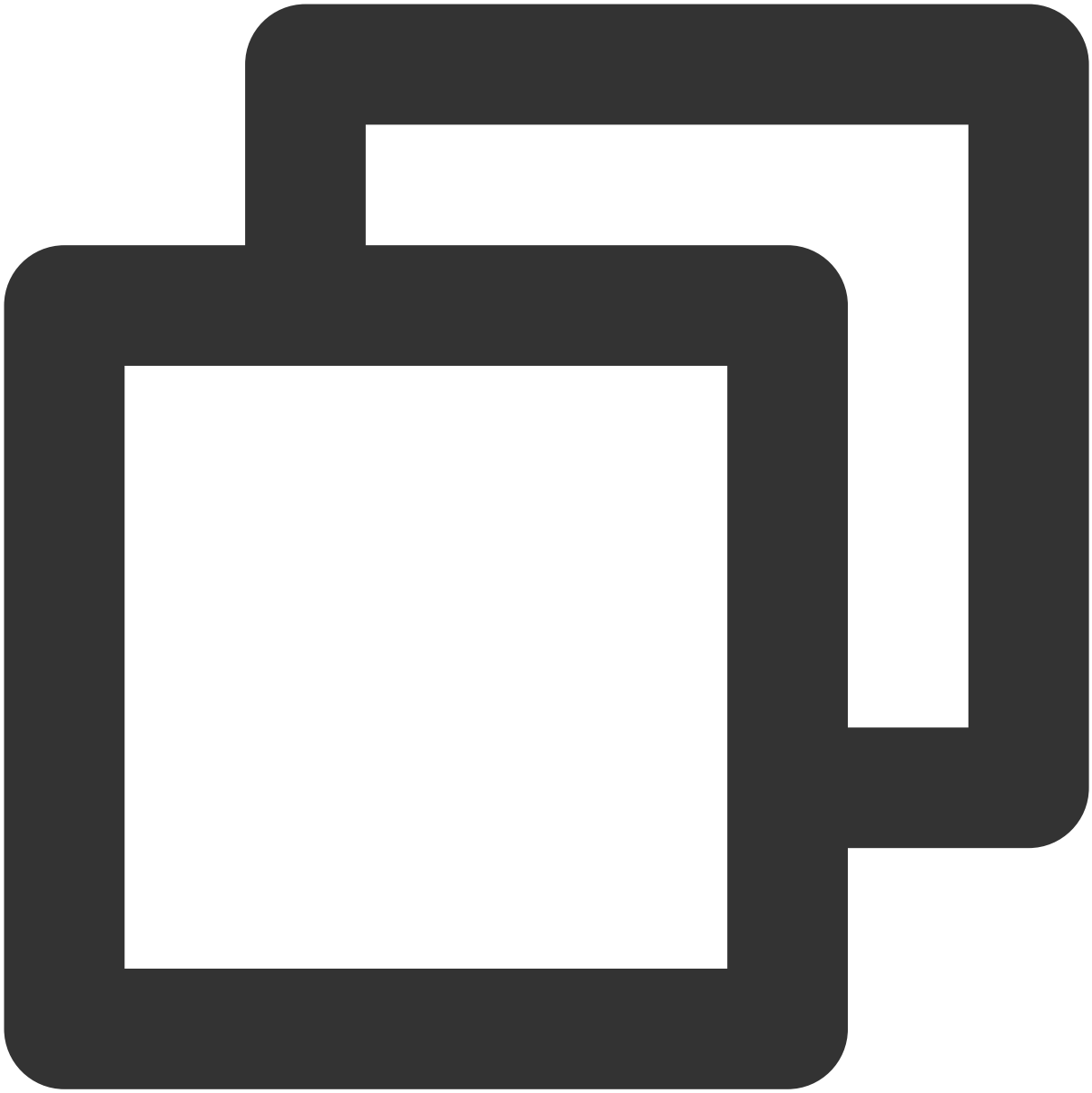
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallBegin: (TUIRoomId roomId, TUICallMediaType callMediaType, TUICallRole cal
      //您的回调处理逻辑
    }
));
```

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

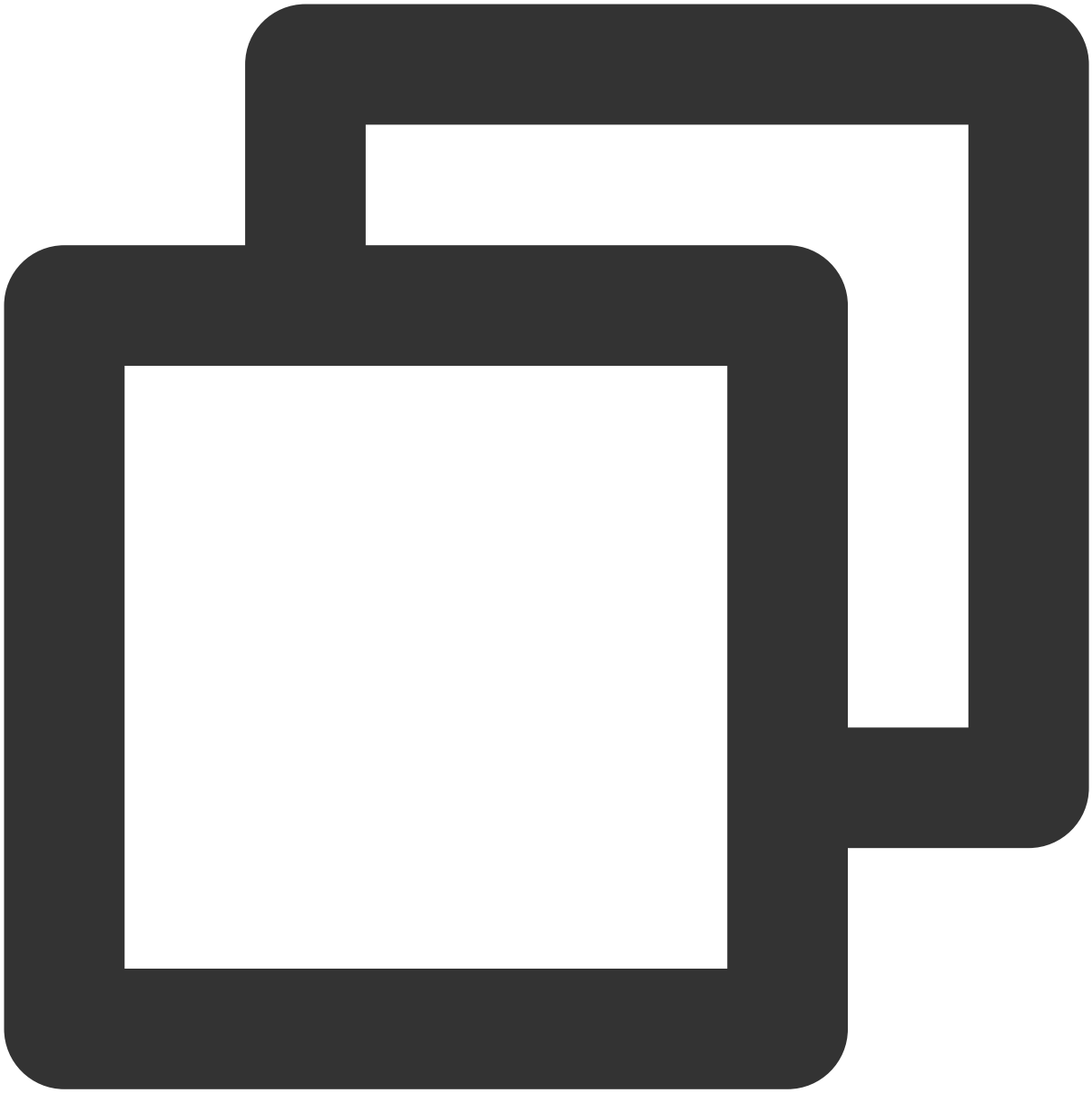
callMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`
callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)
totalTime	double	此次通话的时长，单位 ms

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
    }
));
```

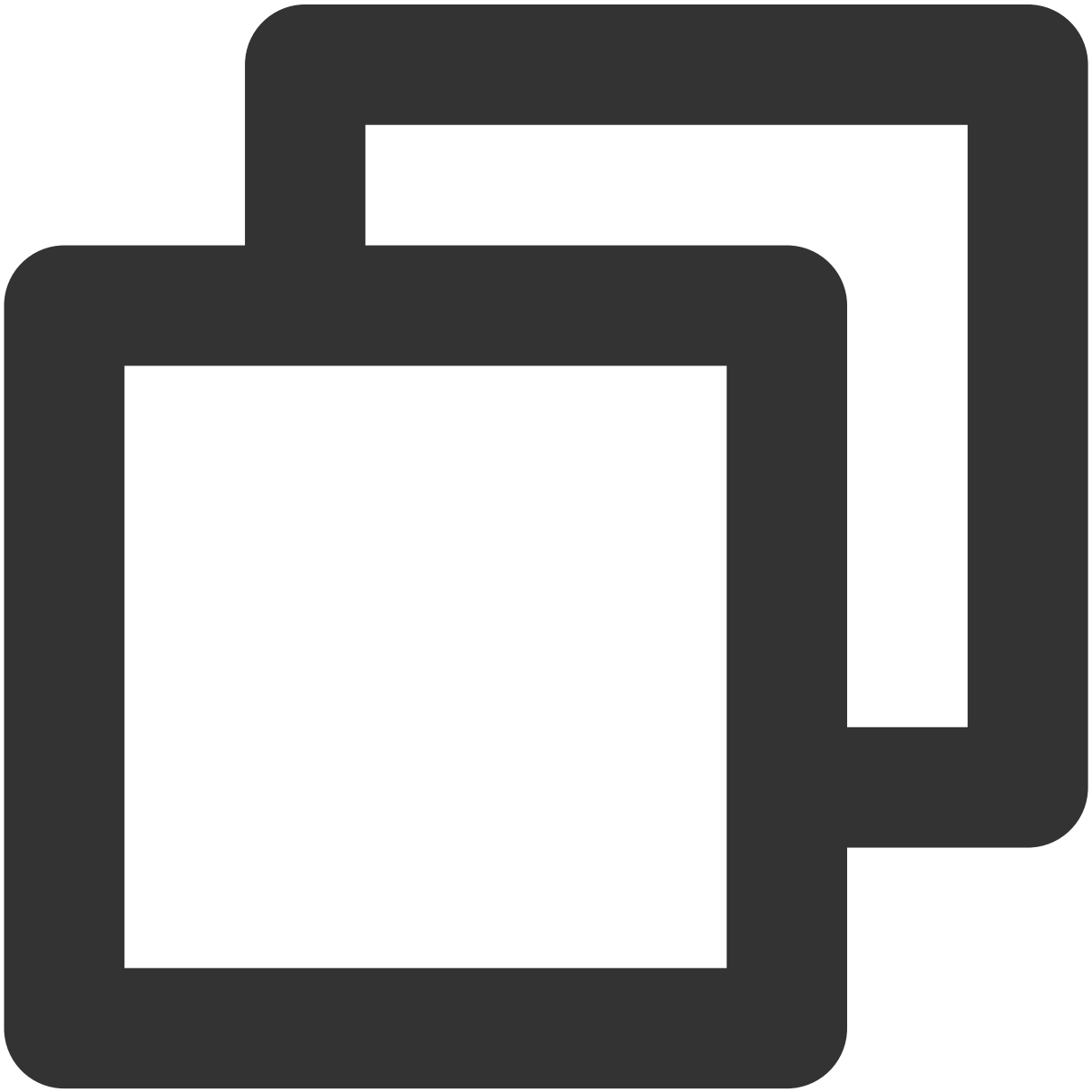
参数如下表所示：

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) {
```

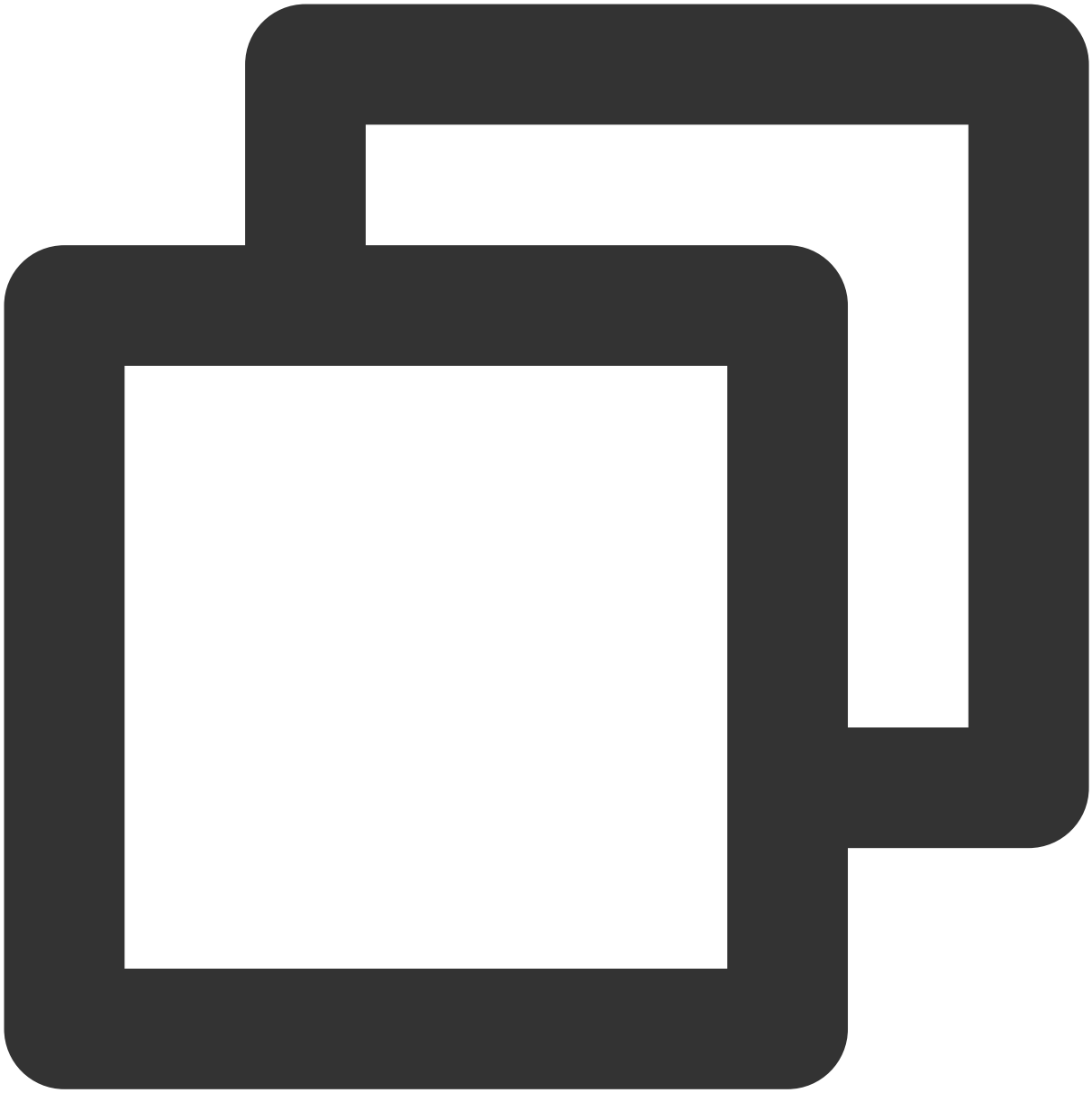
```
        //您的回调处理逻辑
    }
));
```

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。



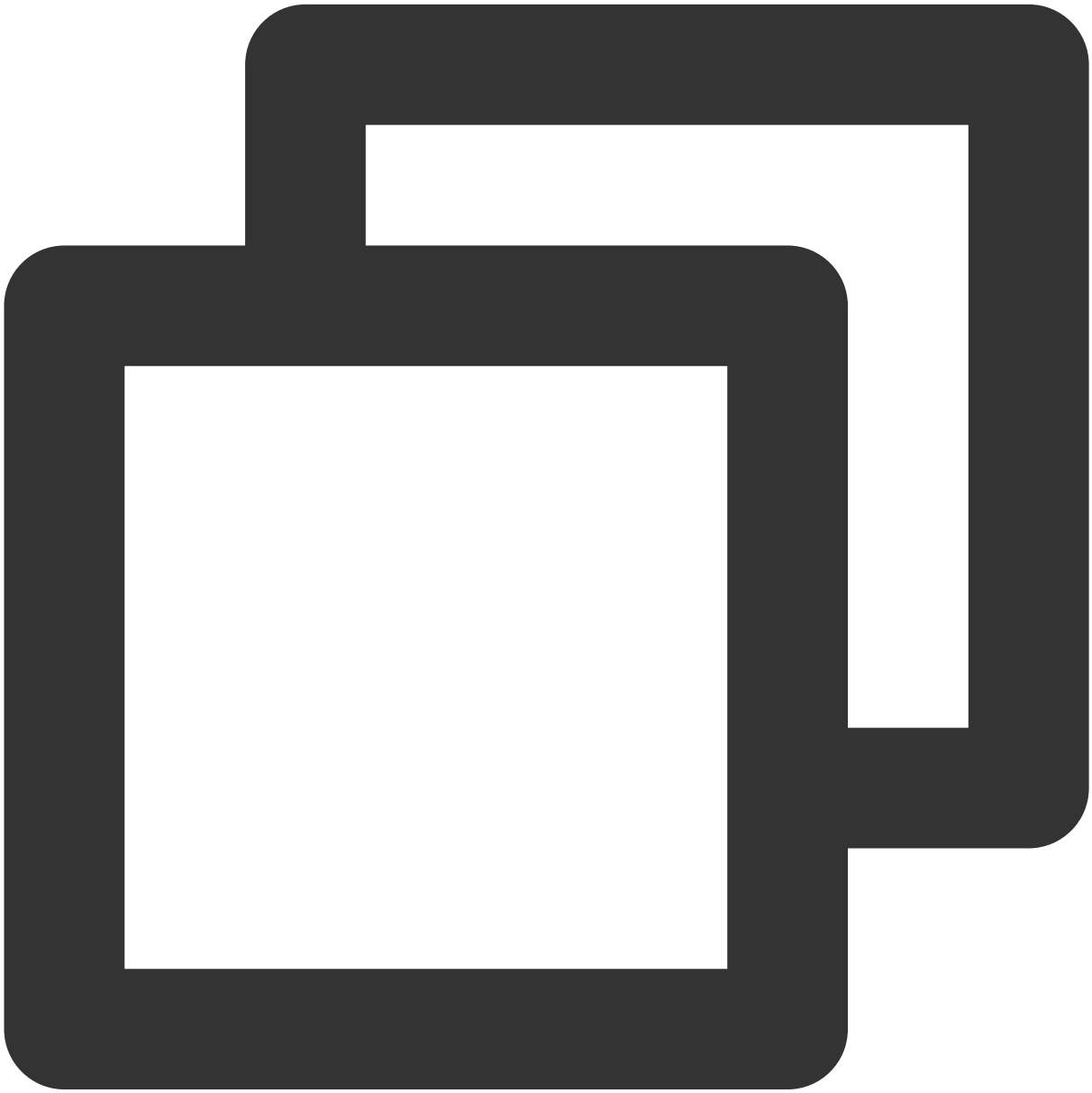
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserNoResponse: (String userId) {
      //您的回调处理逻辑
    }
));
```

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。



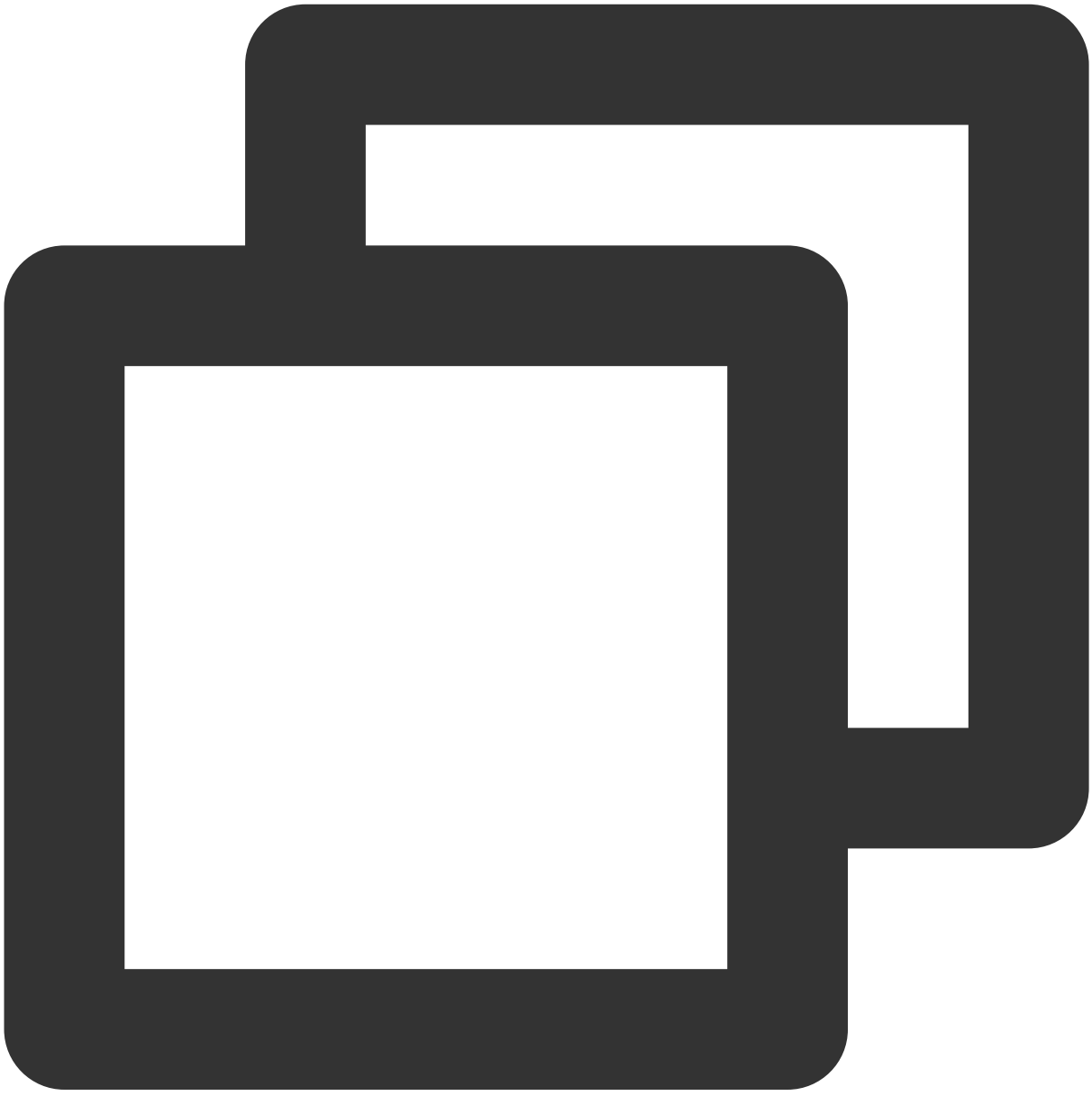
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLineBusy: (String onUserLineBusy) {
      //您的回调处理逻辑
    },
));
```

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
```

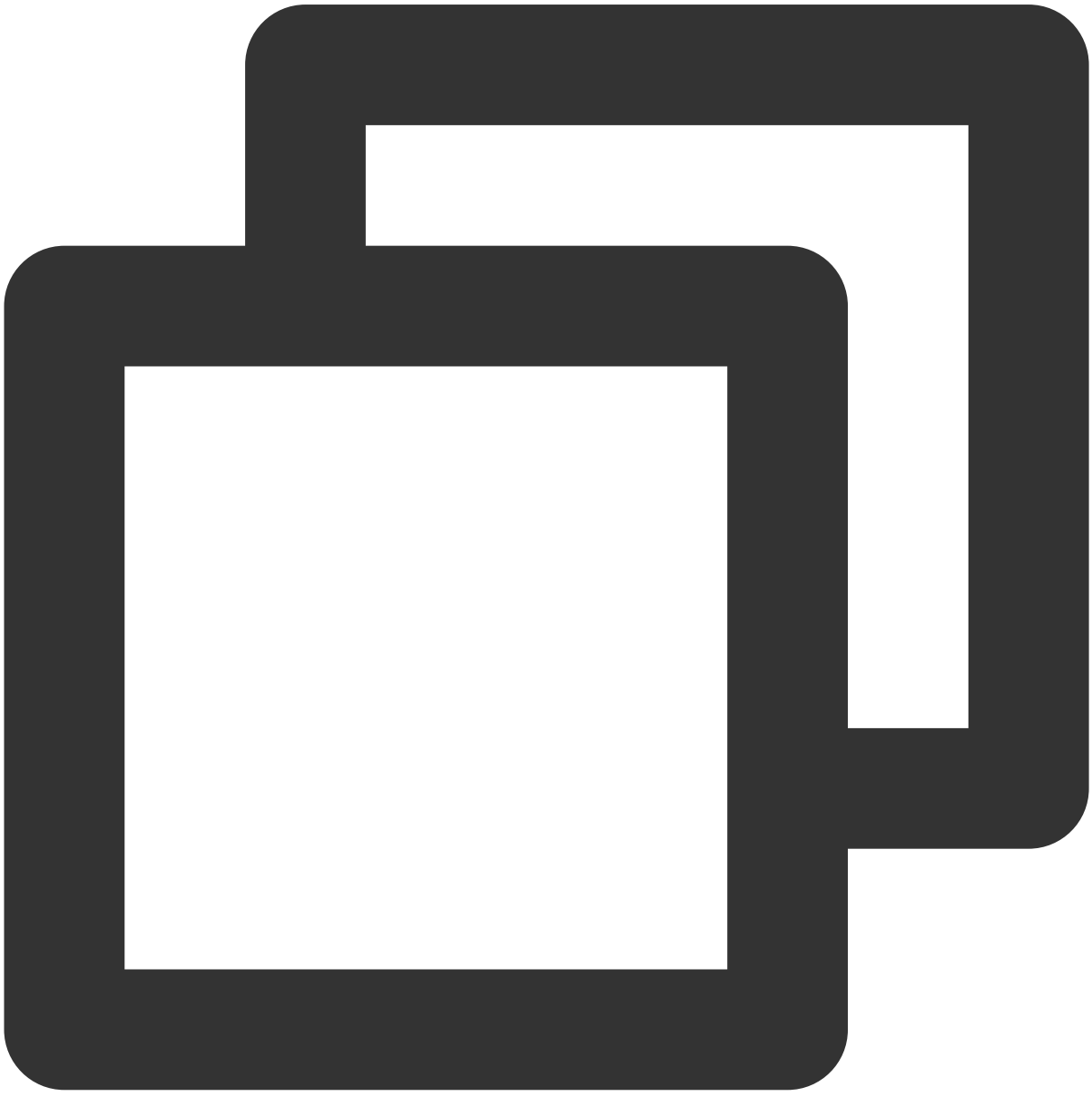
```
    }
));
```

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。



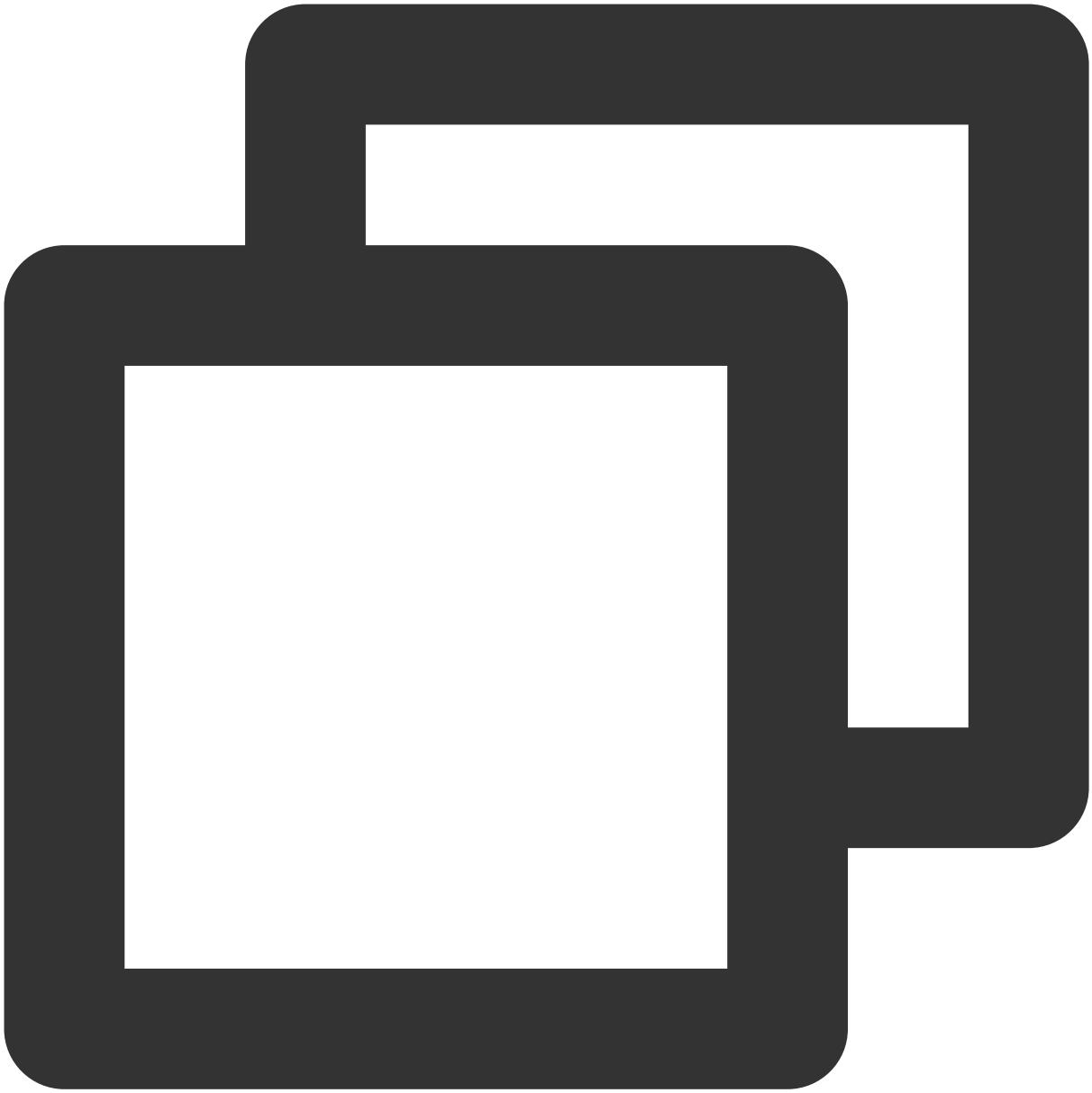
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLeave: (String userId) {
      //您的回调处理逻辑
    }
));
```

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
    }
));
```

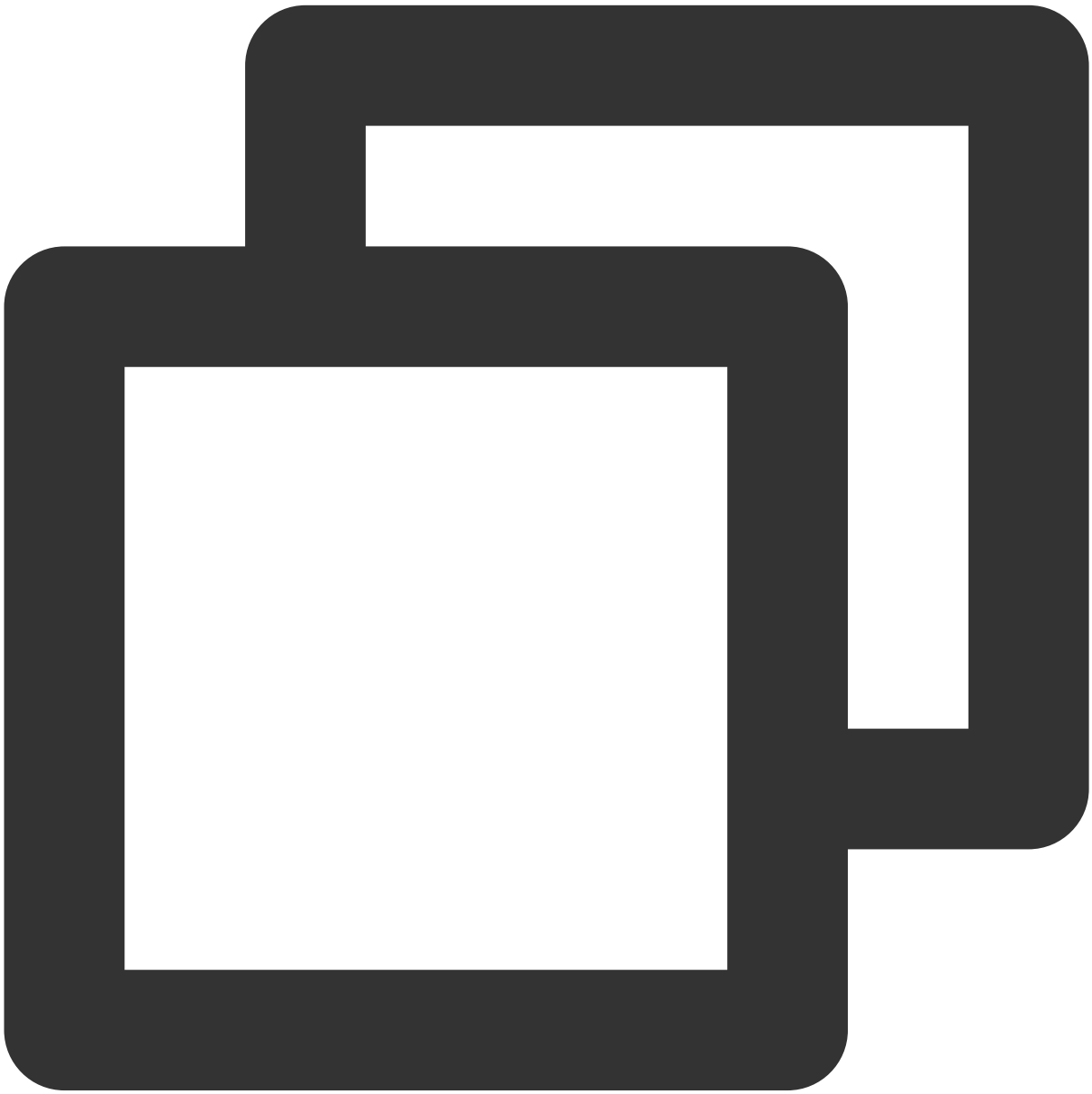
参数如下表所示：

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。



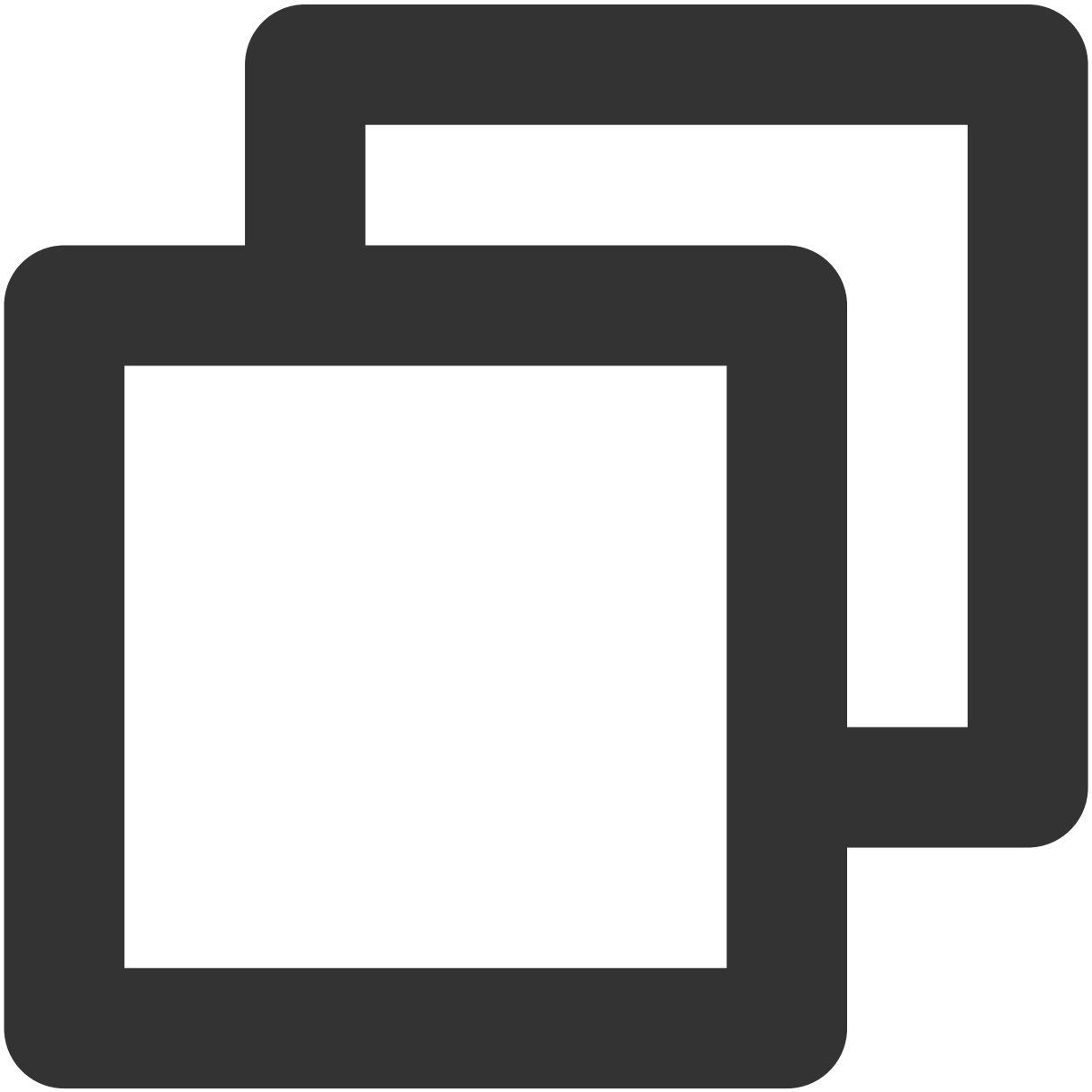
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {
      //您的回调处理逻辑
    }
));
```

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。



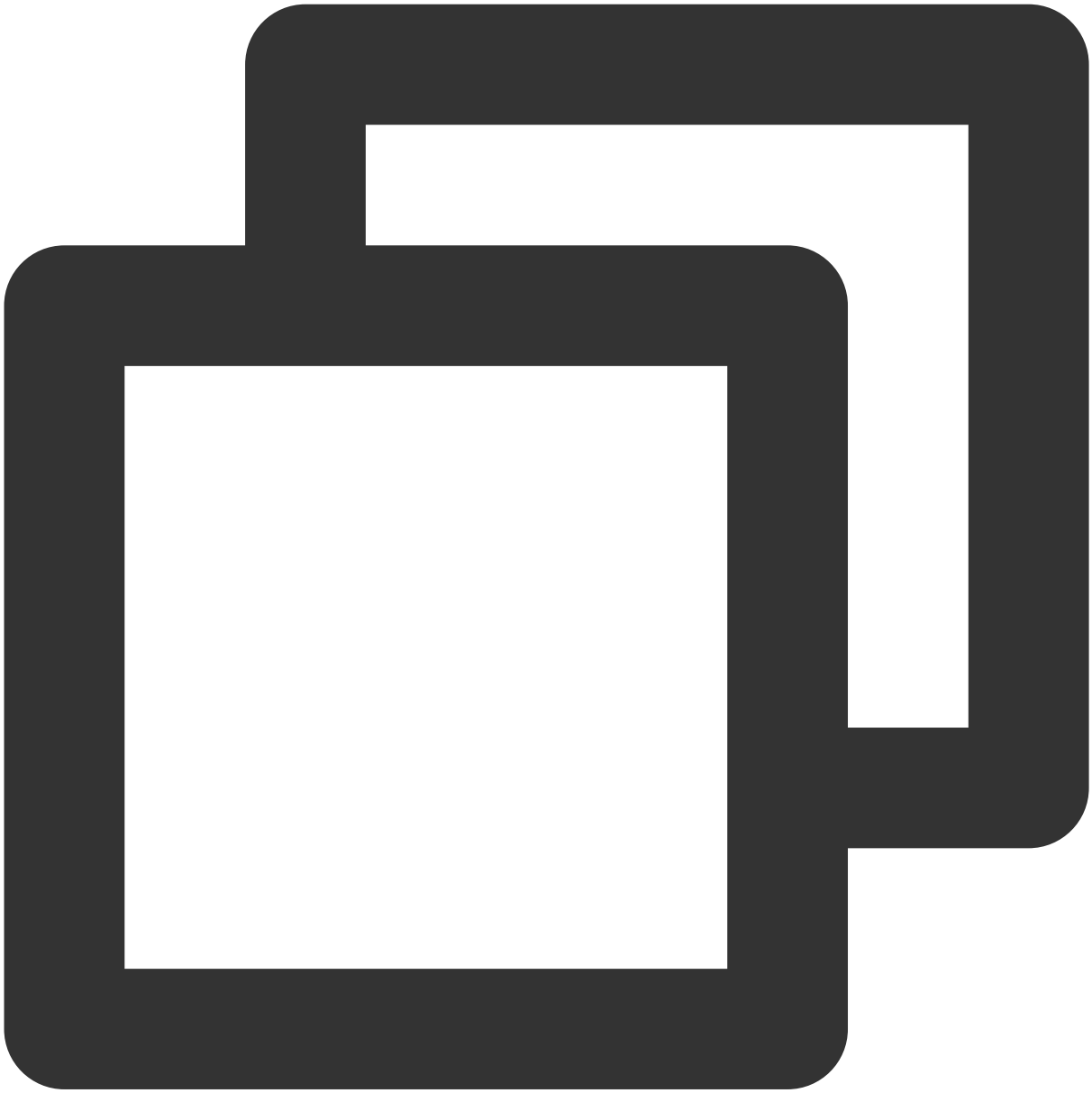
```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {
      //您的回调处理逻辑
    }
));
```

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});
```

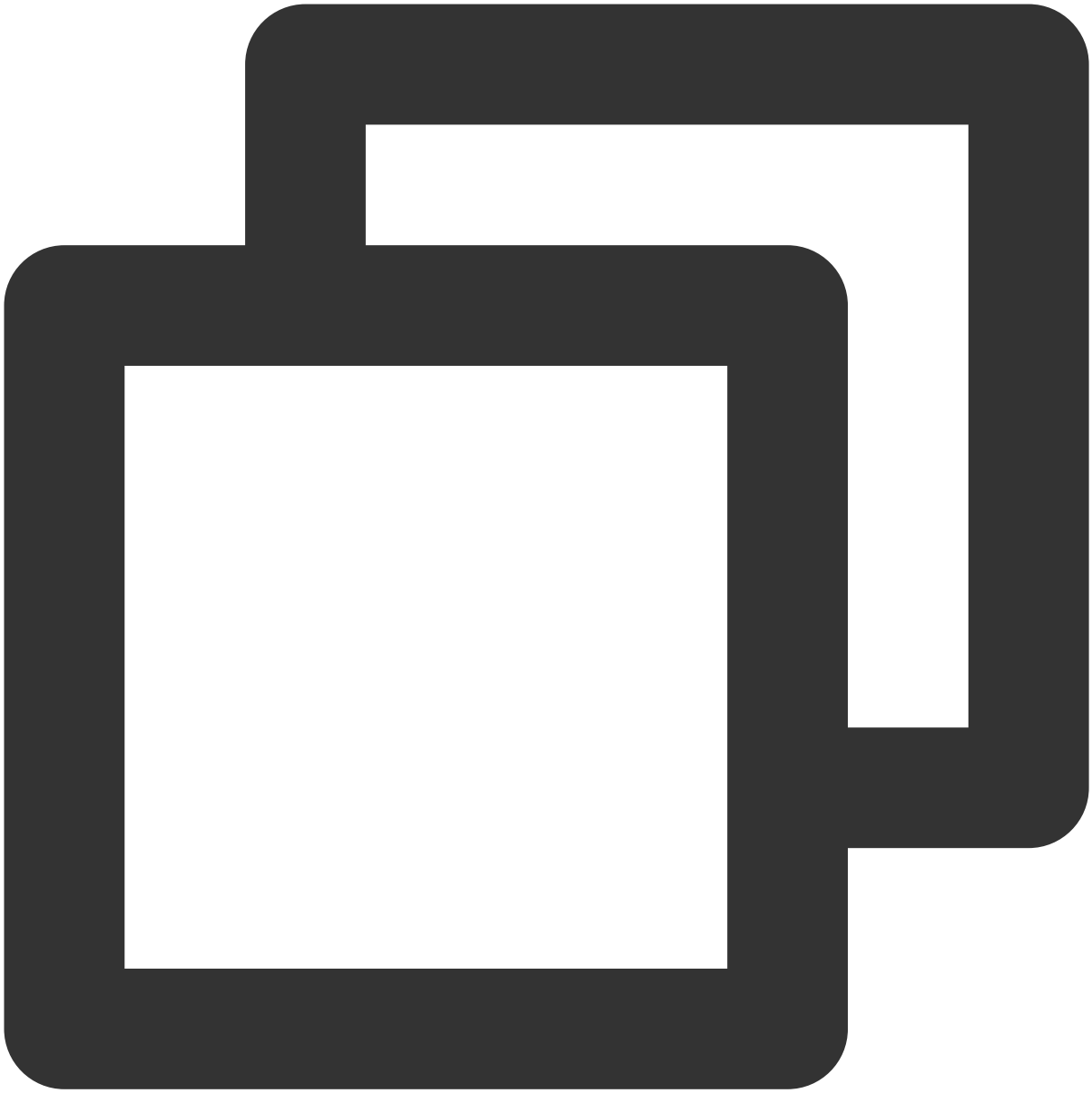
```
}
// TUINetworkQuality的定义如下：
enum TUINetworkQuality {
  unknown,
  excellent,
  good,
  poor,
  bad,
  vBad,
  down
}
```

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

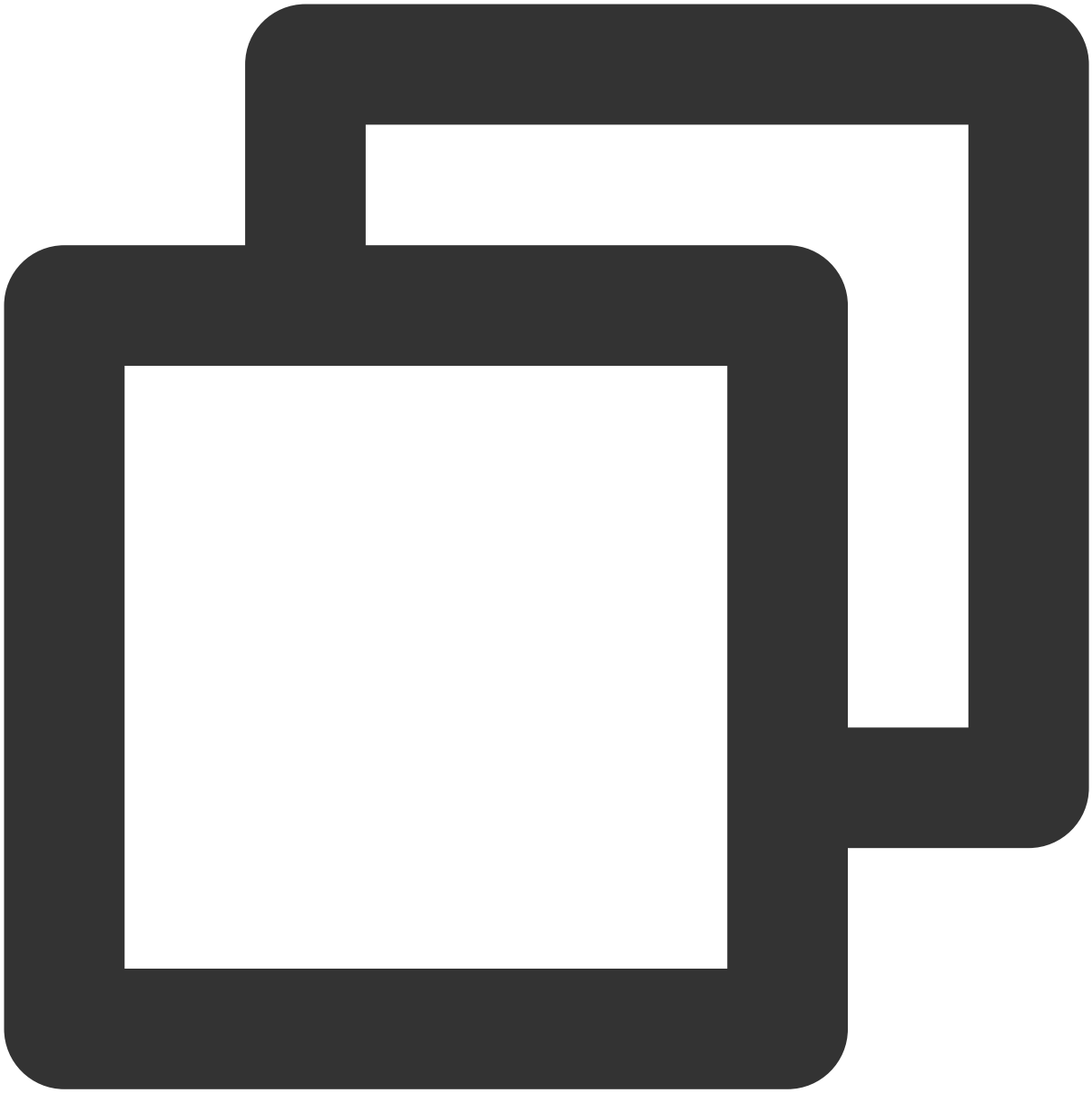
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
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：2024-07-19 14:15:49

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
enableIncomingBanner	开启/关闭来电横幅显示，**v2.3.1+ 支持**

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
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

TUICallKit

最近更新时间：2024-07-19 14:15:49

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。

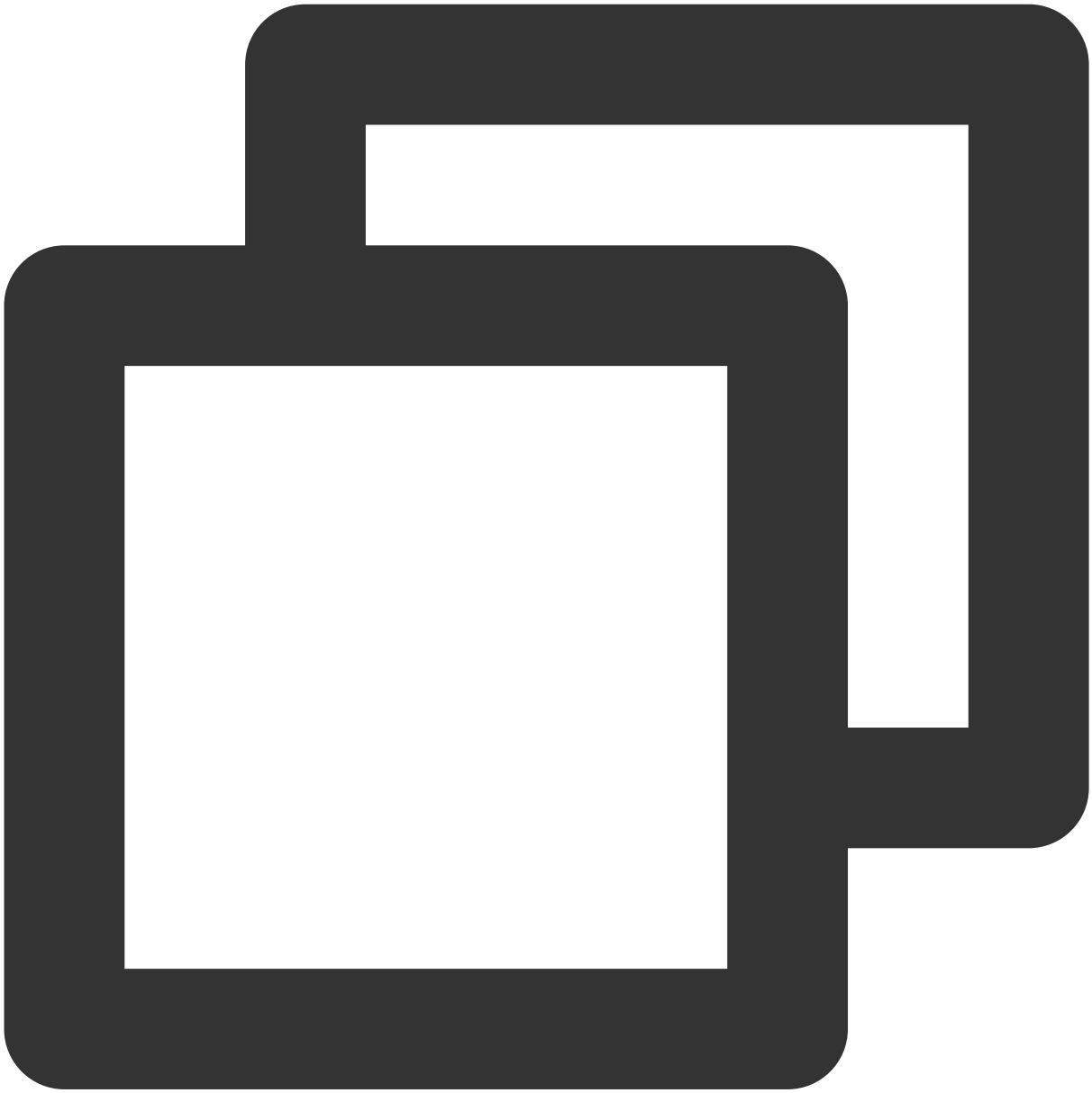
API 概览

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
enableIncomingBanner	开启/关闭来电横幅显示，**v2.3.1+ 支持**

API 详情

login

登录



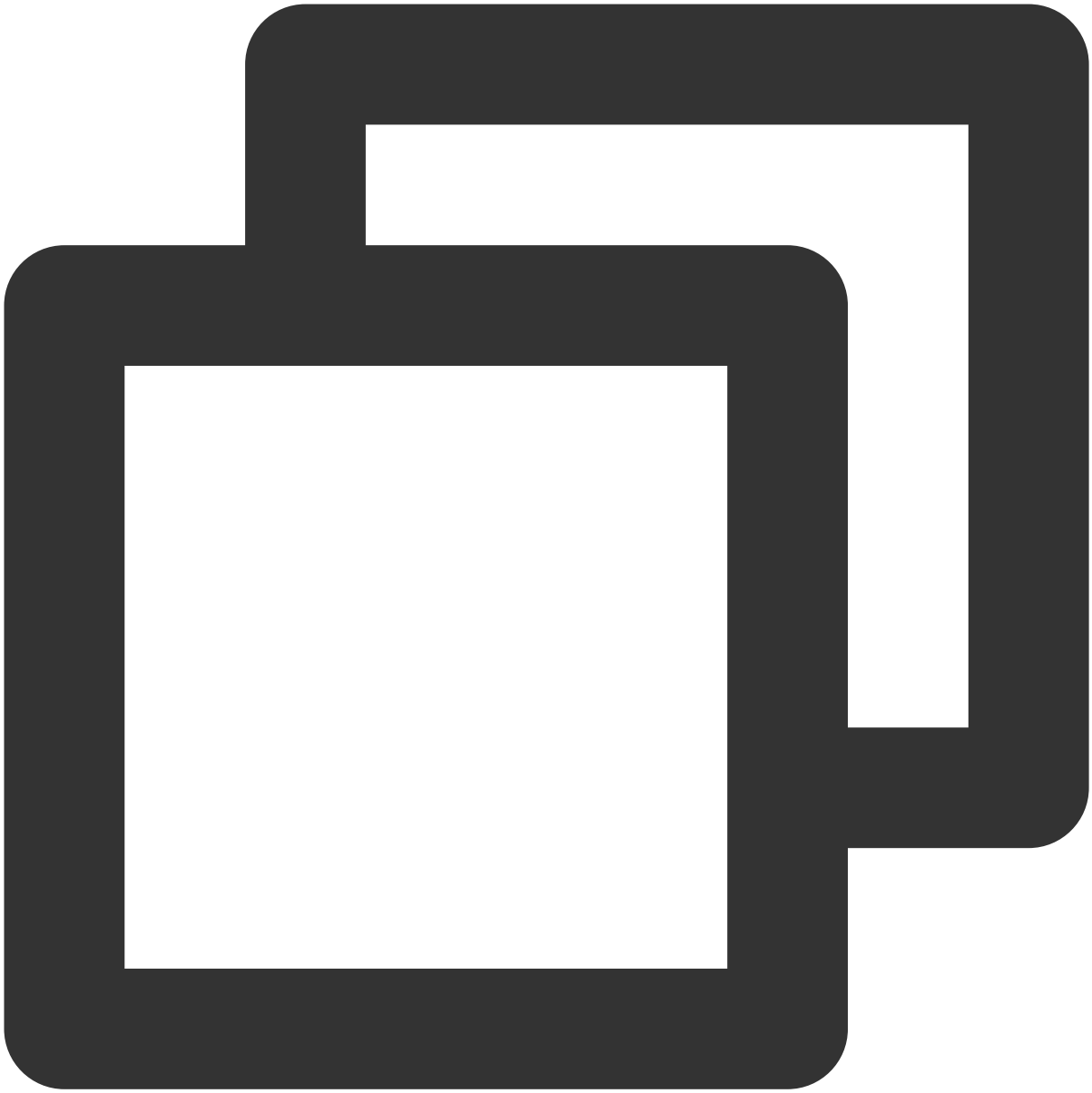
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  SDKAppID: 0,
  userID: 'mike',
  userSig: '',
};
TUICallKit.login(options, (res) => {
  if (res.code === 0) {
    console.log('login success');
  } else {
    console.log(`login failed, error message = ${res.msg}`);
```

```
    }
});
```

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



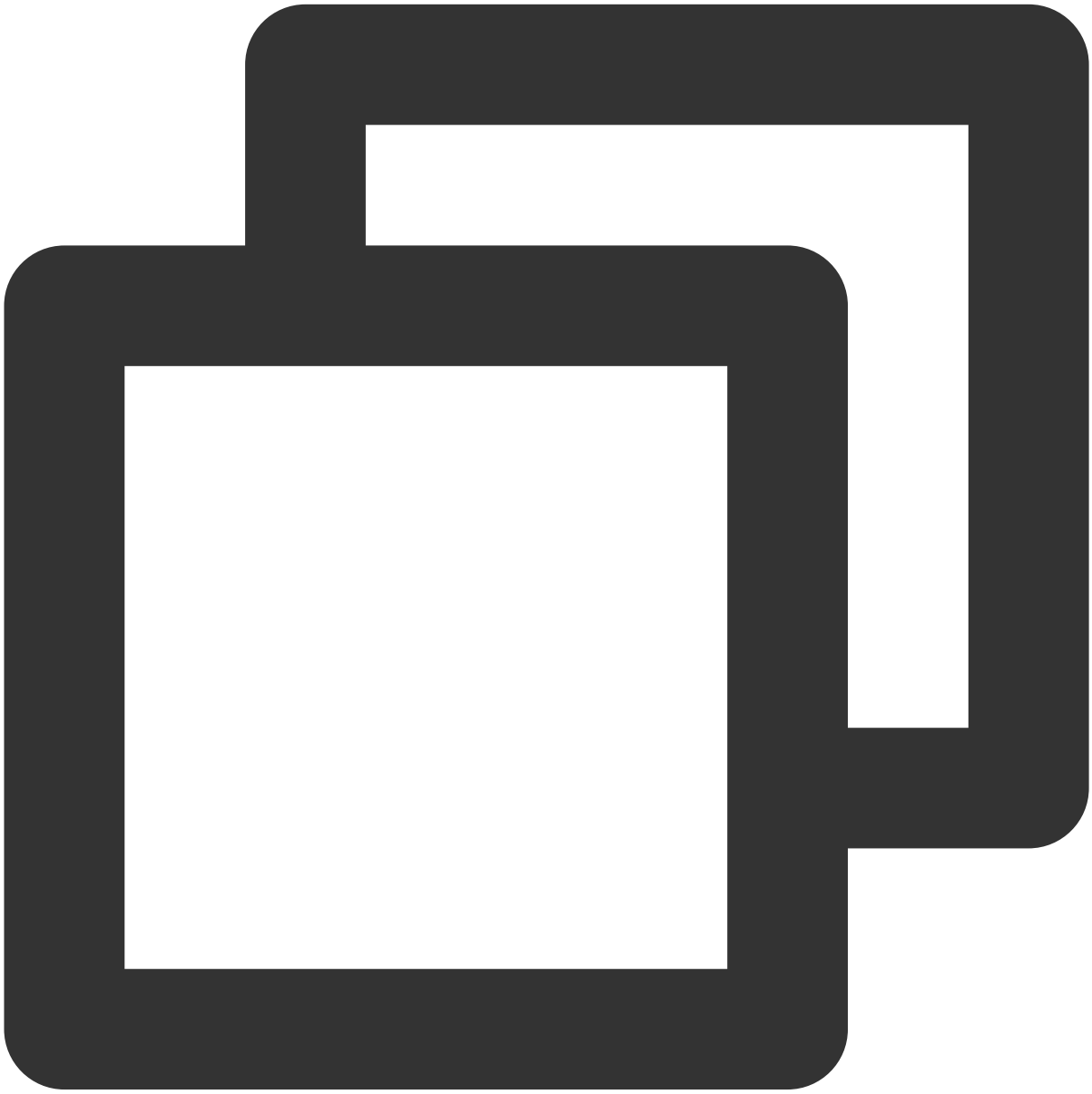
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.logout((res) => {
  if (res.code === 0) {
    console.log('logout success');
  } else {
    console.log(`logout failed, error message = ${res.msg}`);
  }
});
```

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。



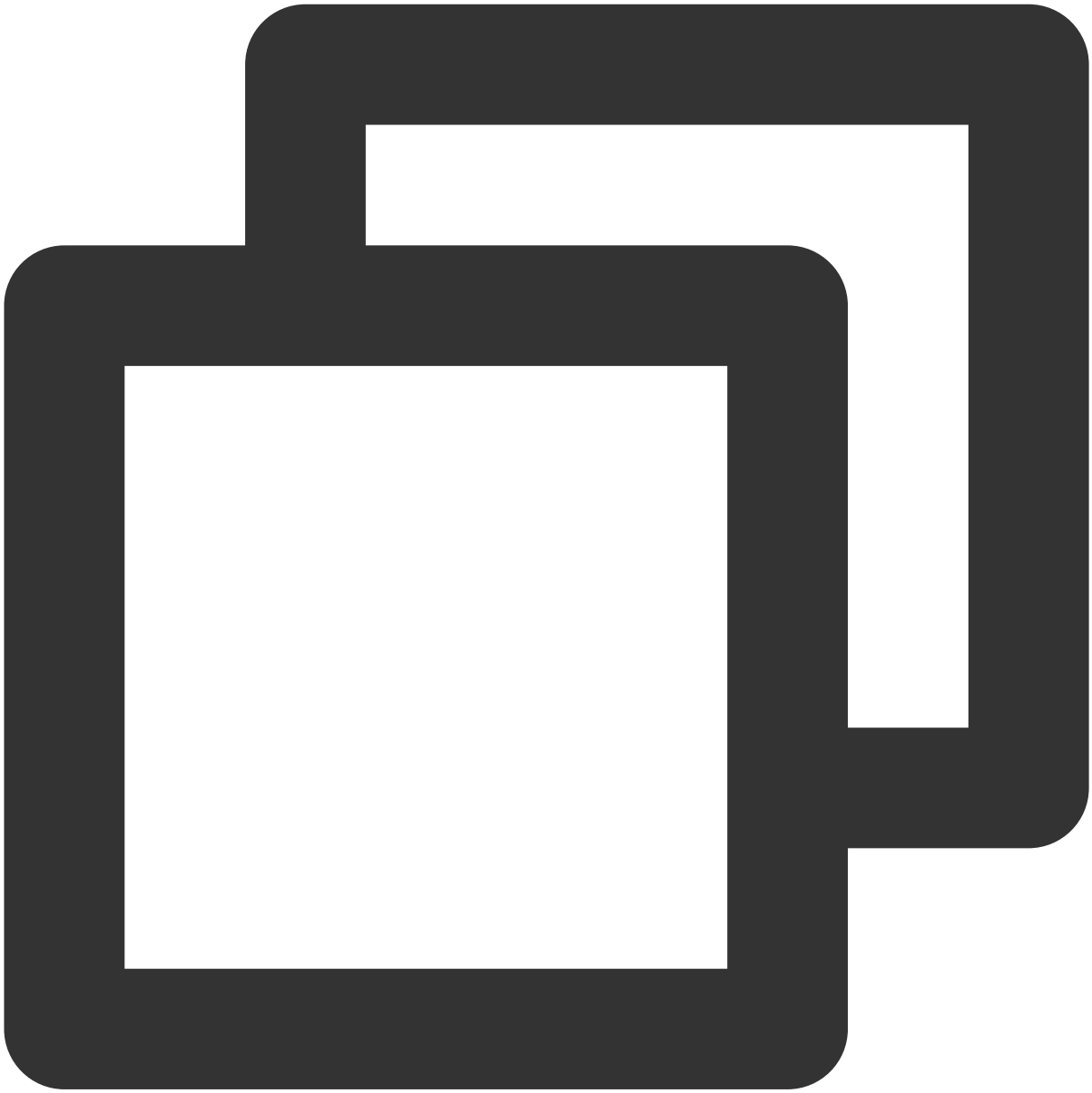
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  nickName: 'jack',
  avatar: 'https:/****/user_avatar.png'
}
```

```
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：

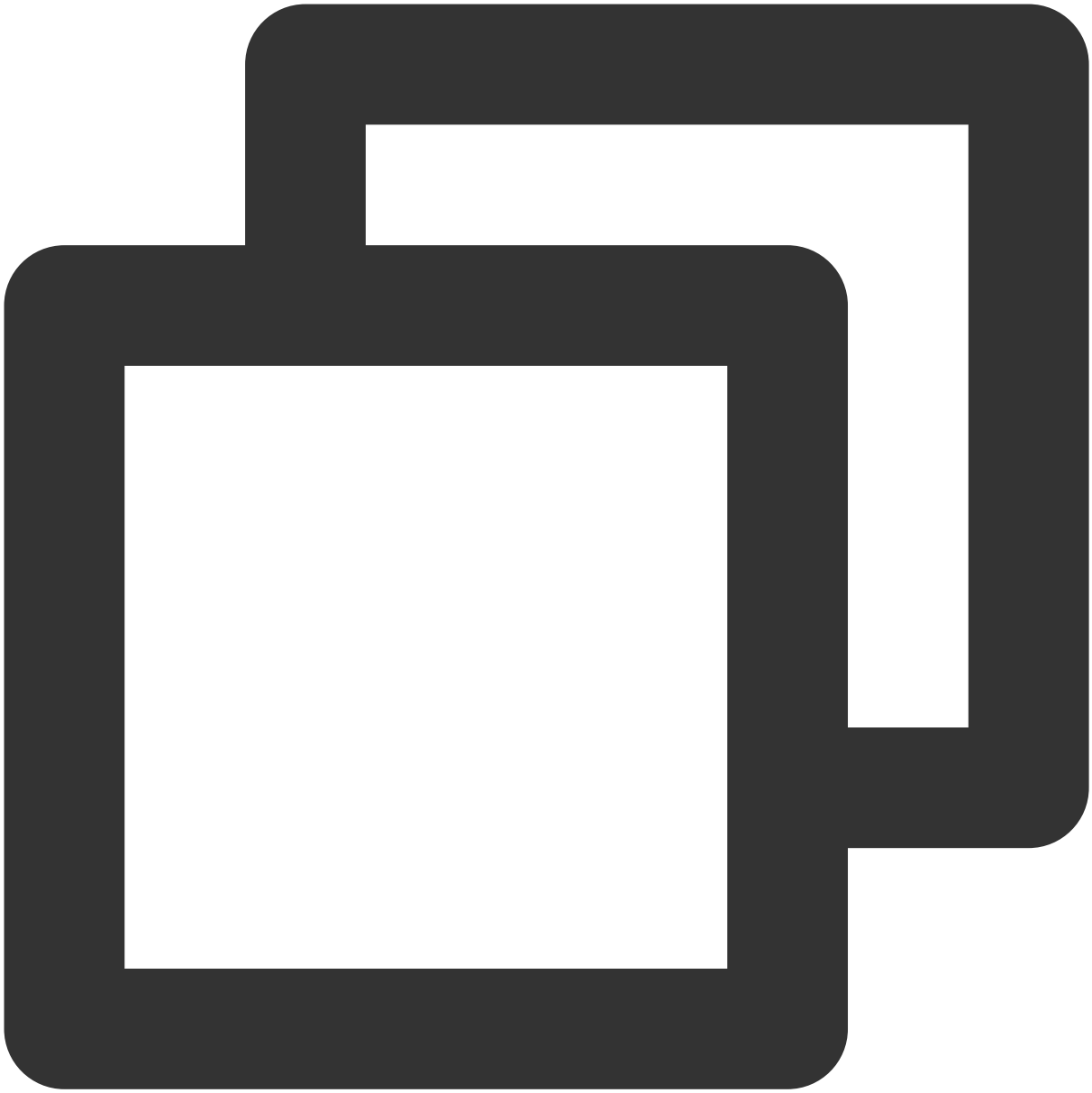
参数	类型	含义
options	Object	初始化参数
options.userID	String	目标用户的 userID
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.roomID	Number	自定义数字房间号。**只要有 roomID，就用的是数字房间号，即使 strRoomID 存在**。
options.strRoomID	String	自定义字符串房间号。**如果想使用字符串房间号，设置 strRoomID 后需要将 roomID = 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。



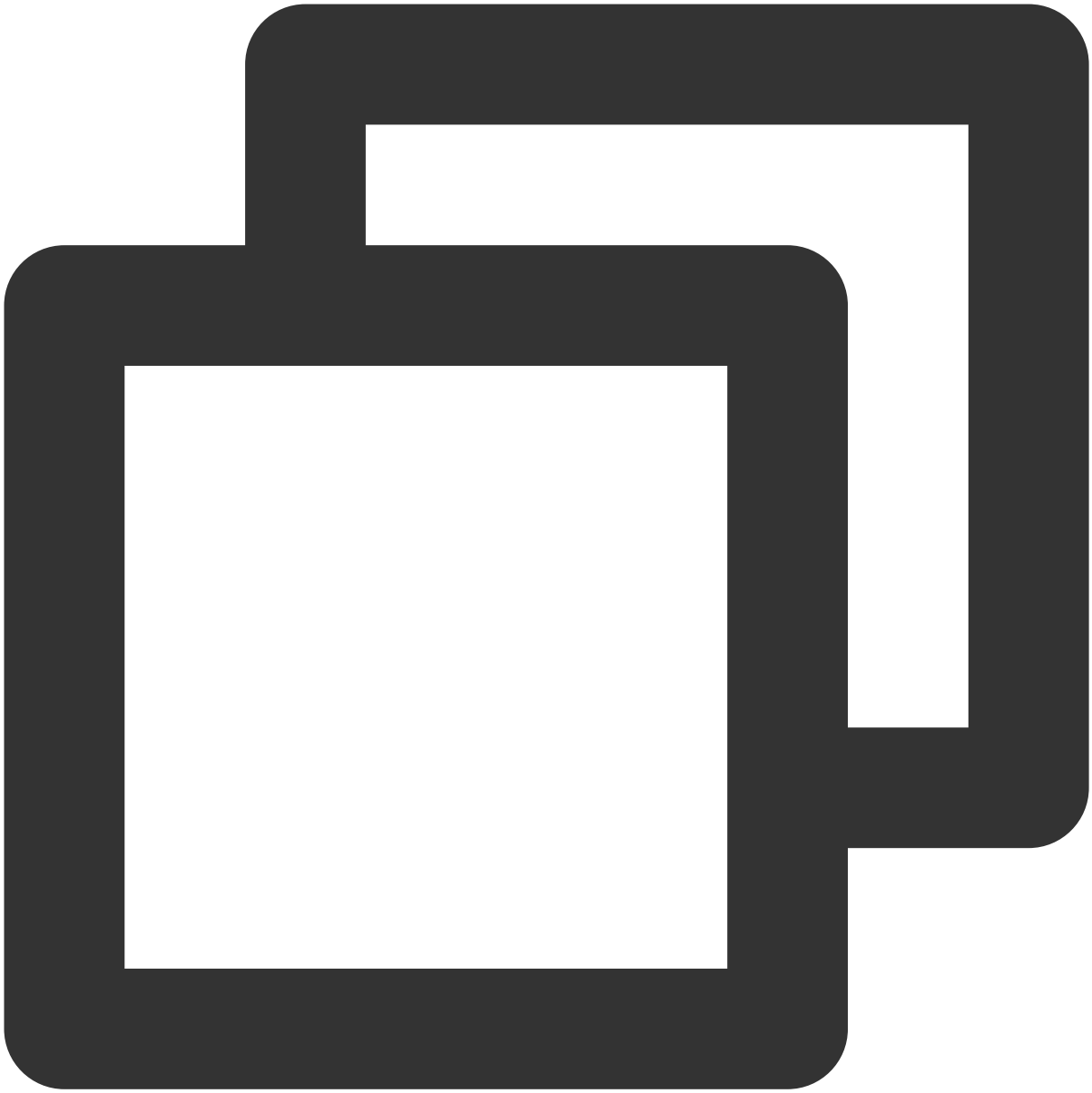
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('call 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。



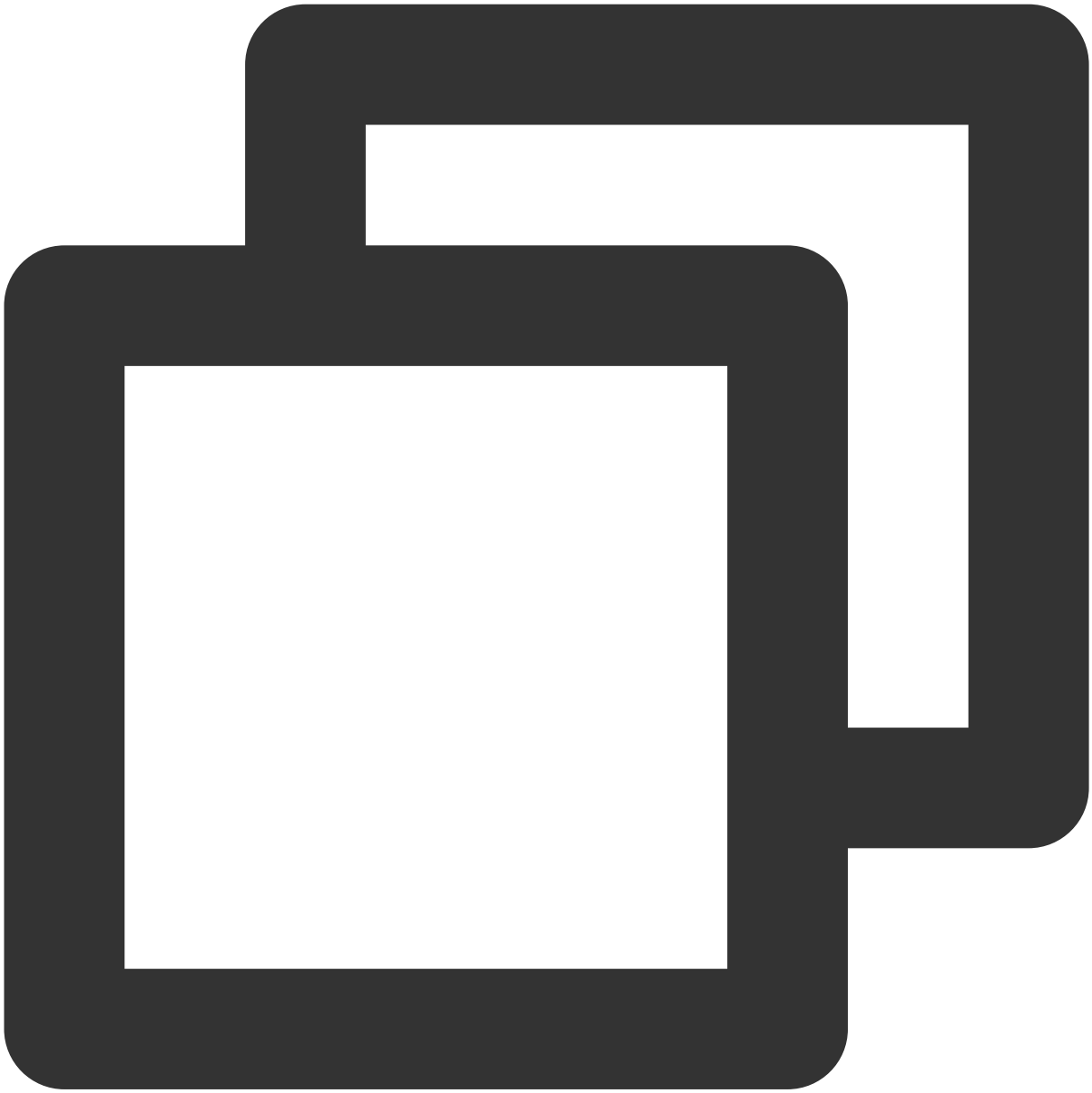
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomID: 9898,
  groupID: 'myGroup',
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall 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;
```

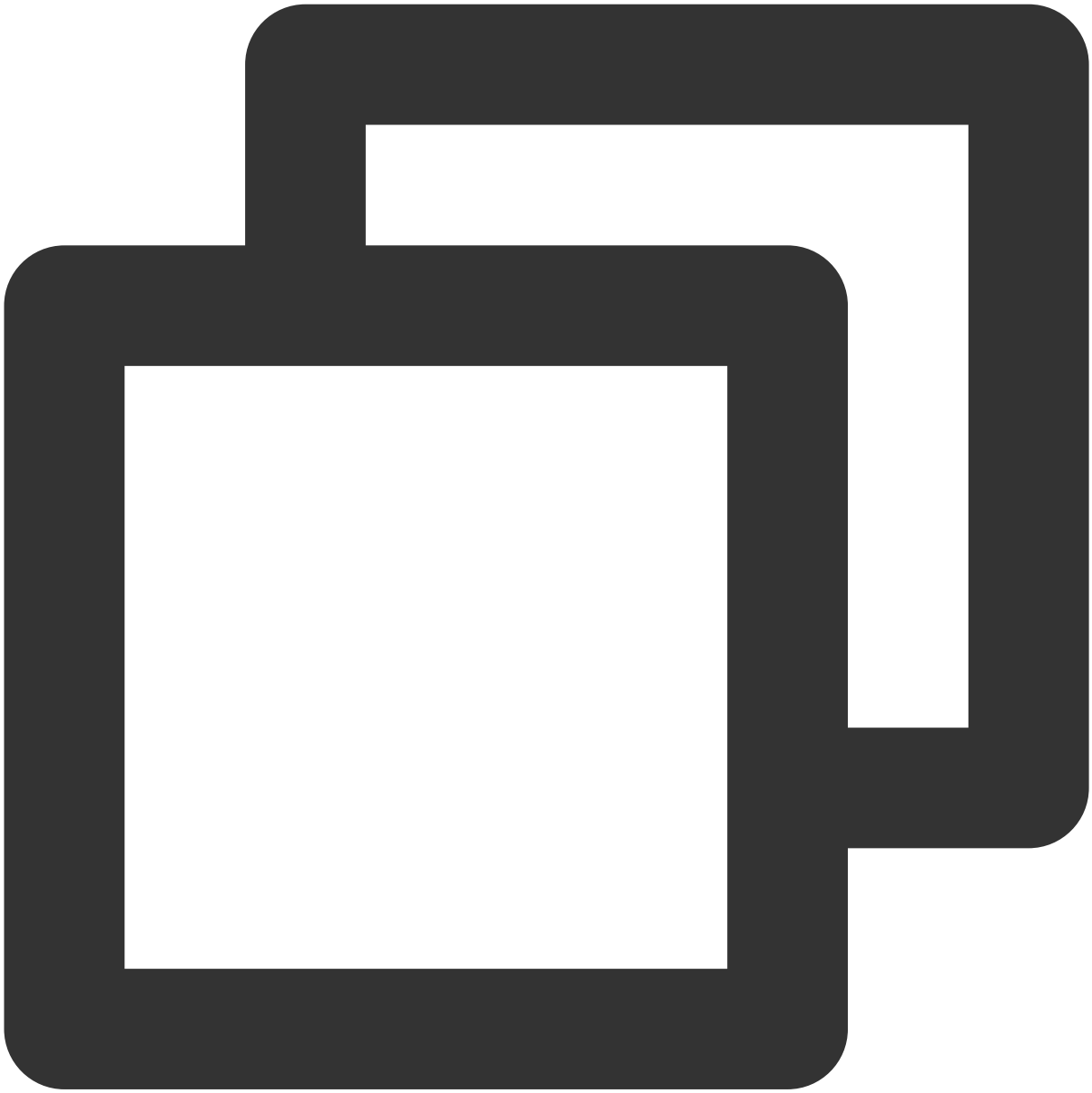
```
        // 【2】相对路径转绝对路径，否则访问不到
        musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);

        // 【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。

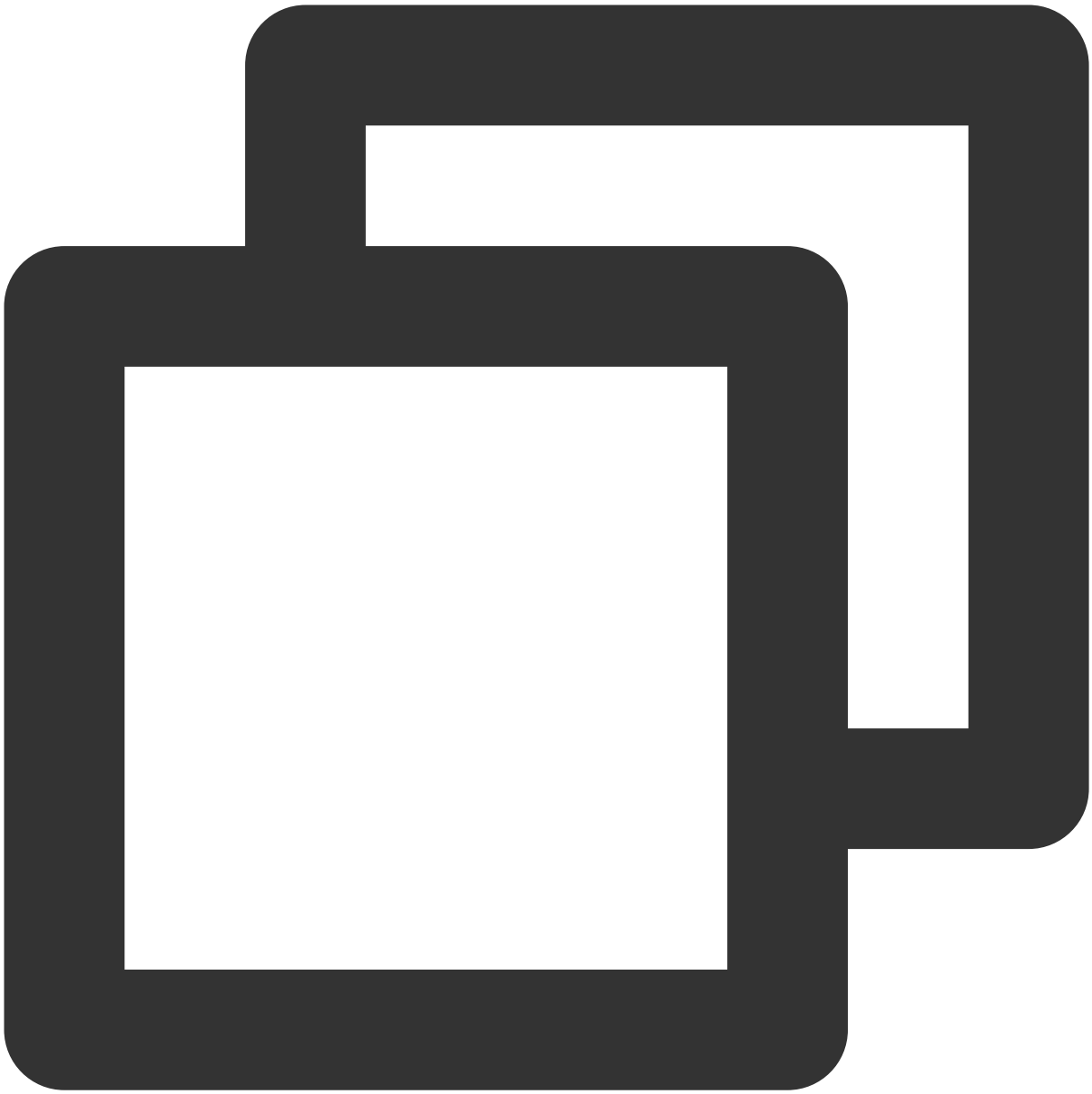


```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableMuteMode(enable);
```

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

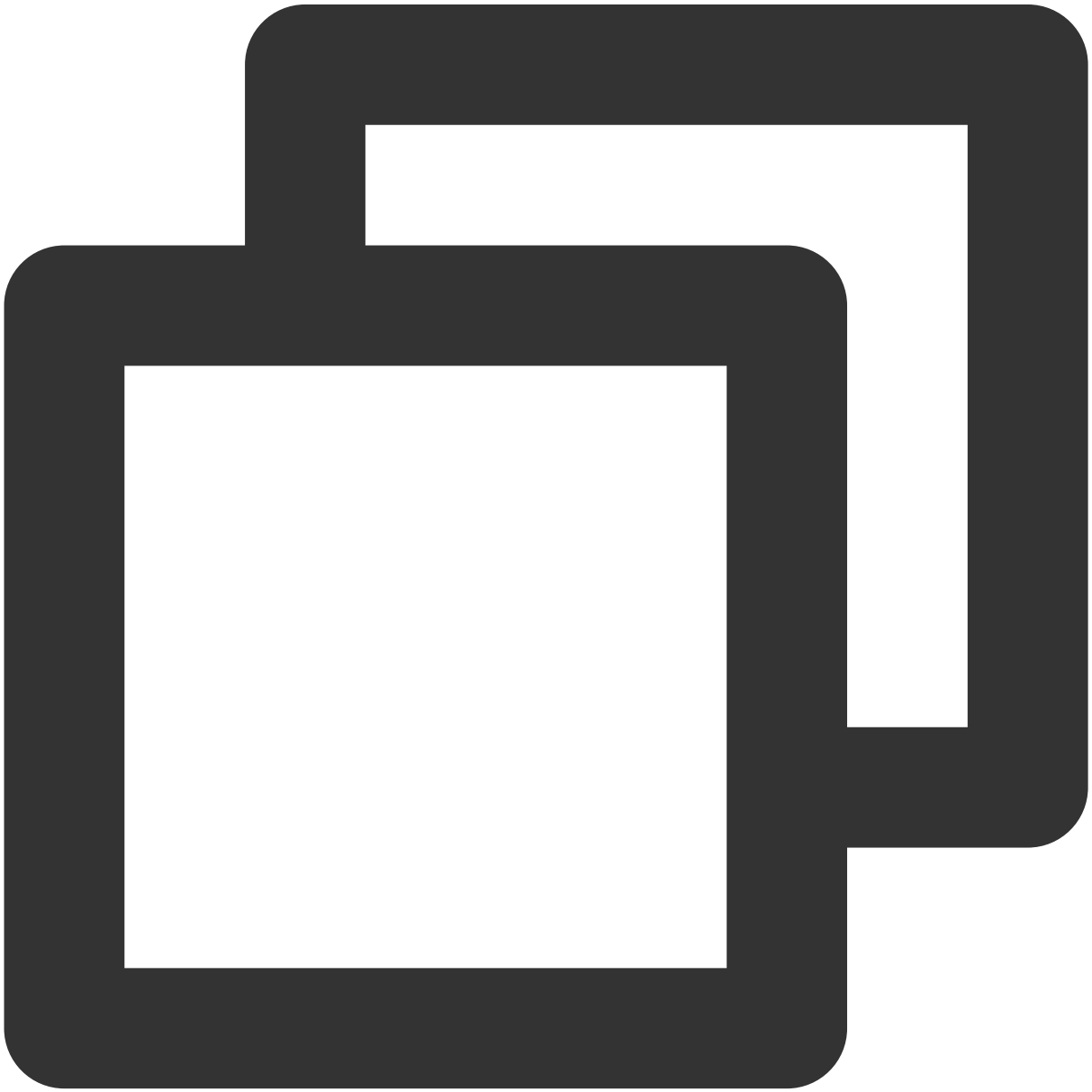
enableIncomingBanner

开启/关闭来电横幅显示。

默认为 `false` ，被叫端收到邀请后默认弹出全屏通话等待界面，开启后先展示一个横幅，然后根据需要拉起全屏通话界面。

注意：

v2.3.1+ 支持。



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

```
TUICallKit.enableIncomingBanner(enable);
```

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2024-03-07 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。

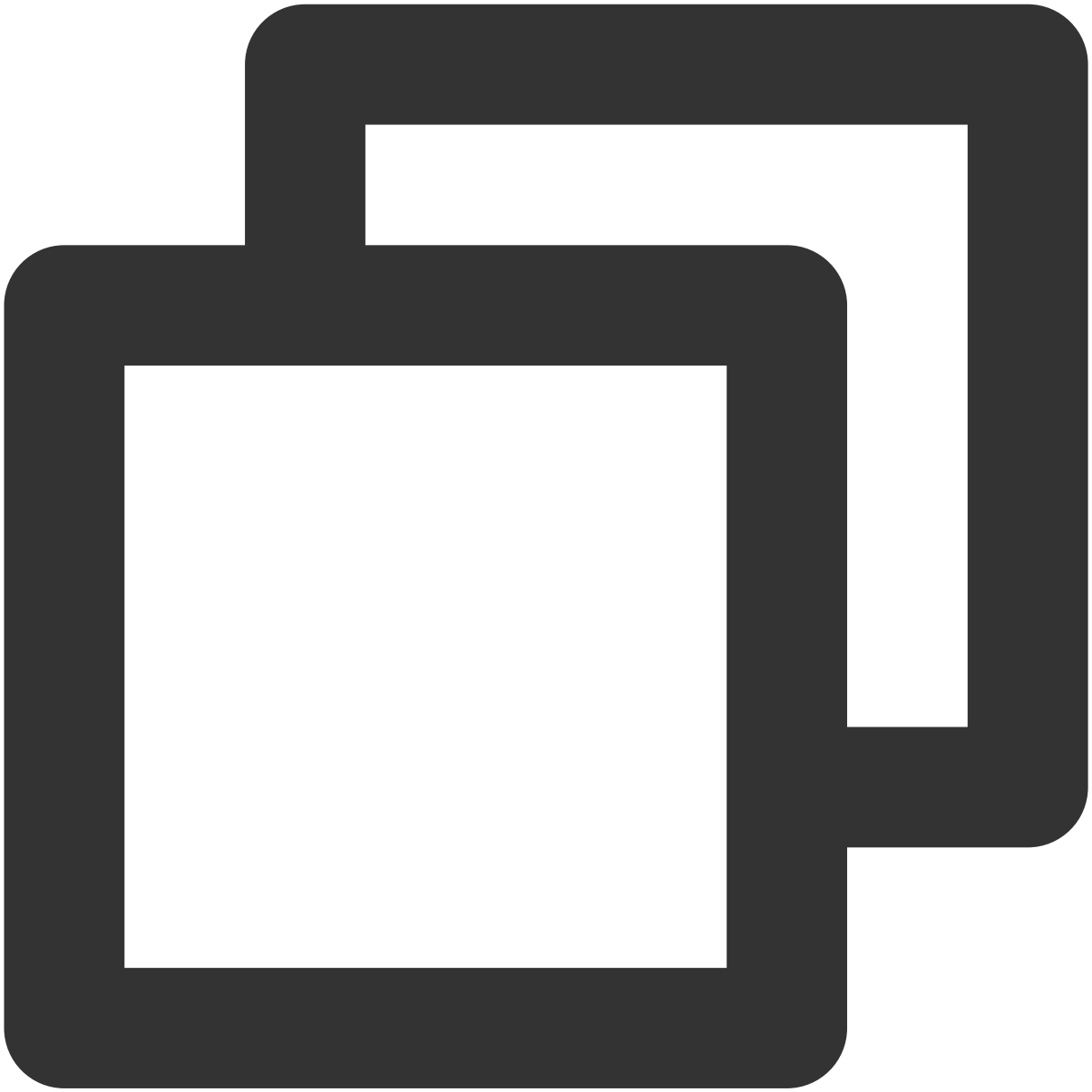
API 概览

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

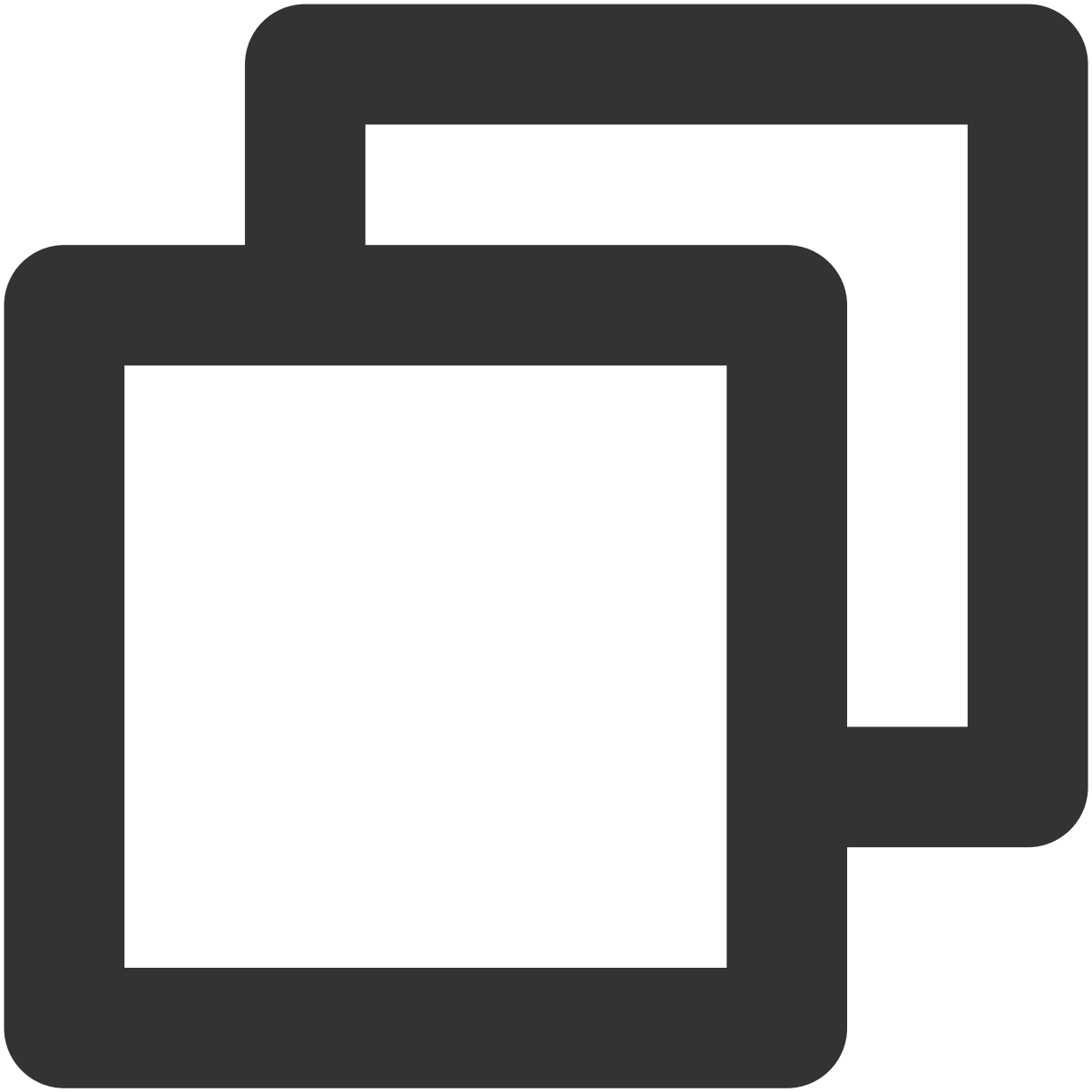
挂断当前通话，当您处于通话中，可以调用该函数结束通话。



```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
TUICallEngine.hangup();
```

accept

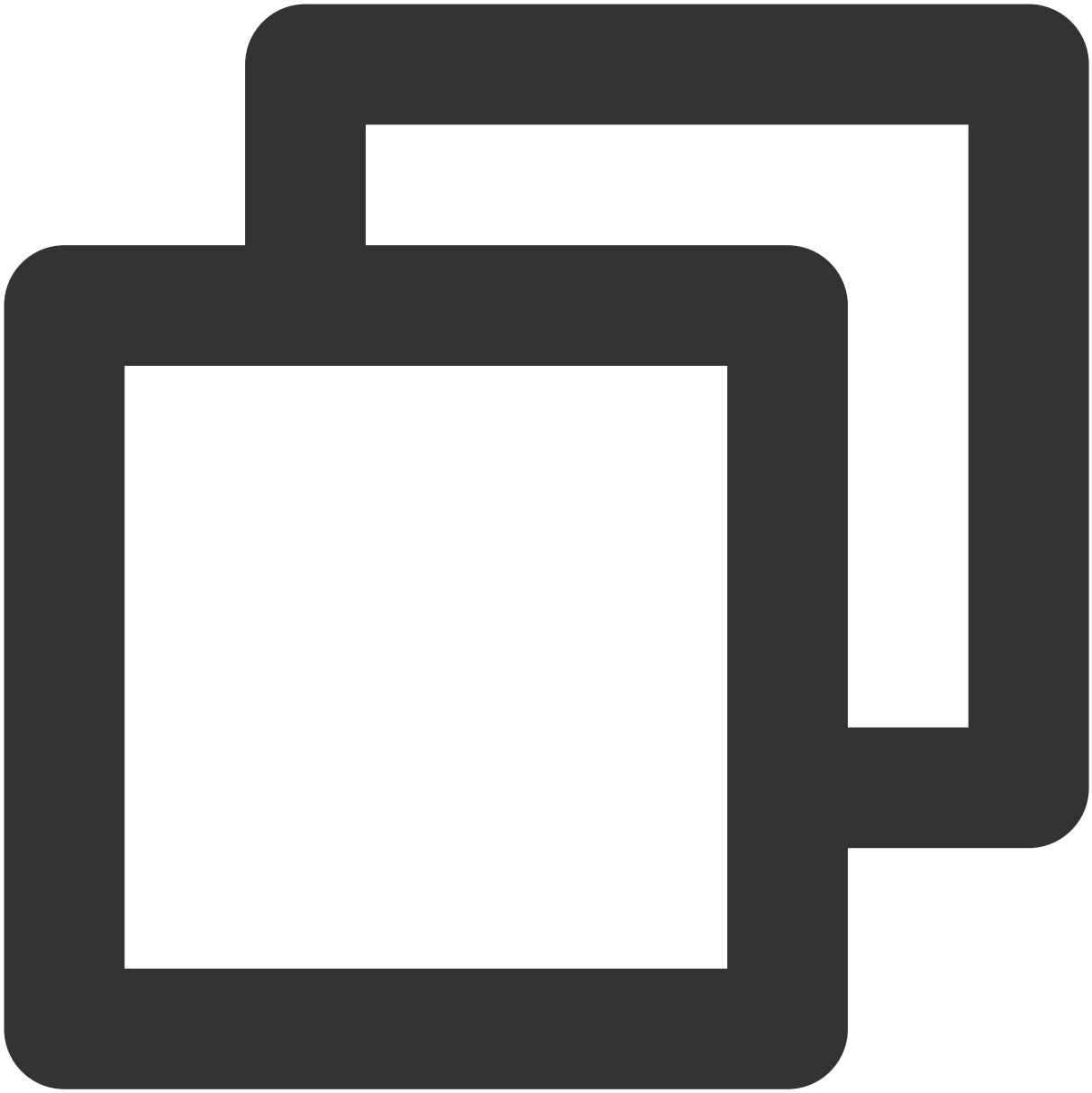
接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。



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

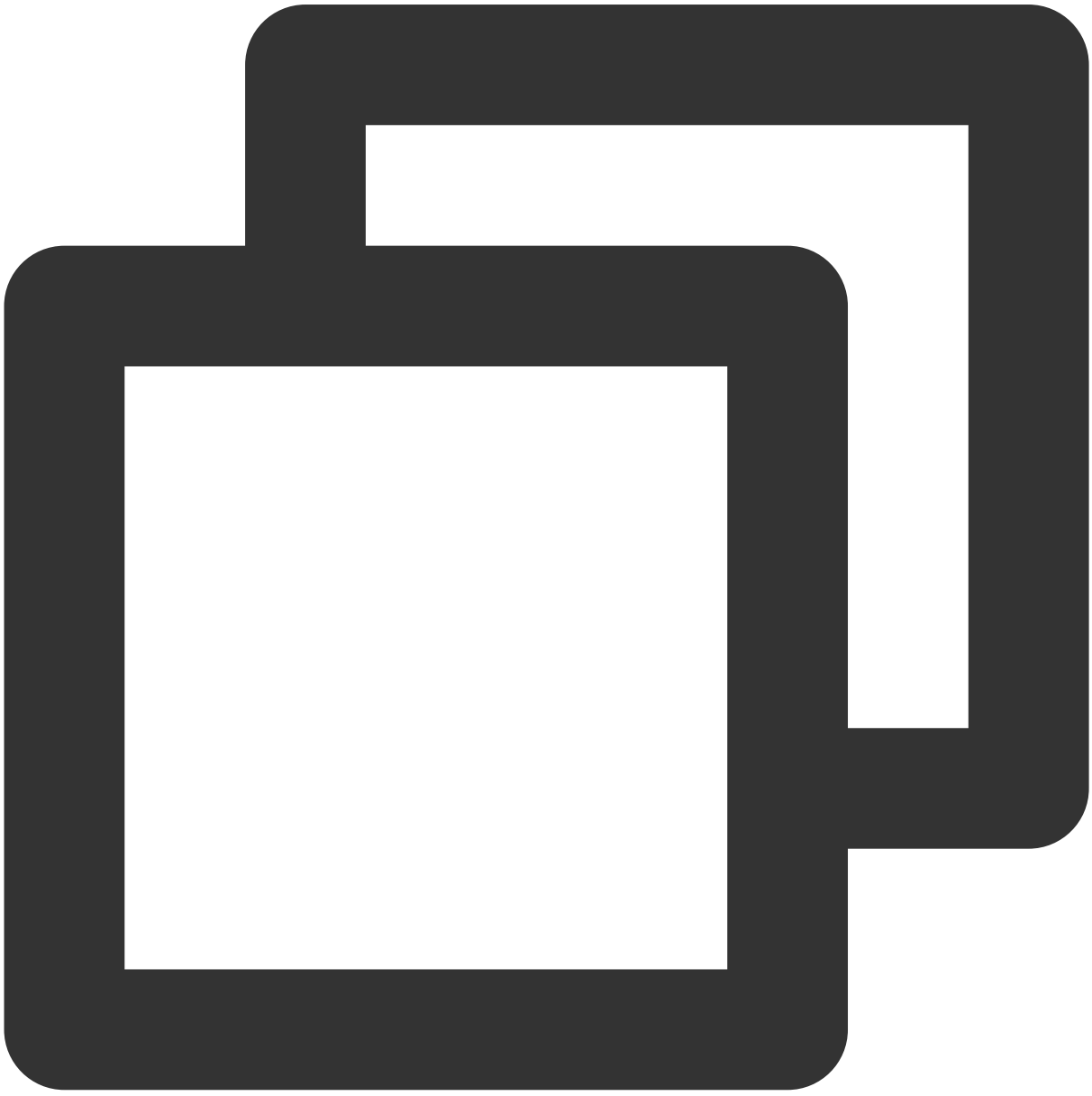
参数如下表所示：

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）

TUICallEvent

最近更新时间：2024-04-30 10:43:45

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

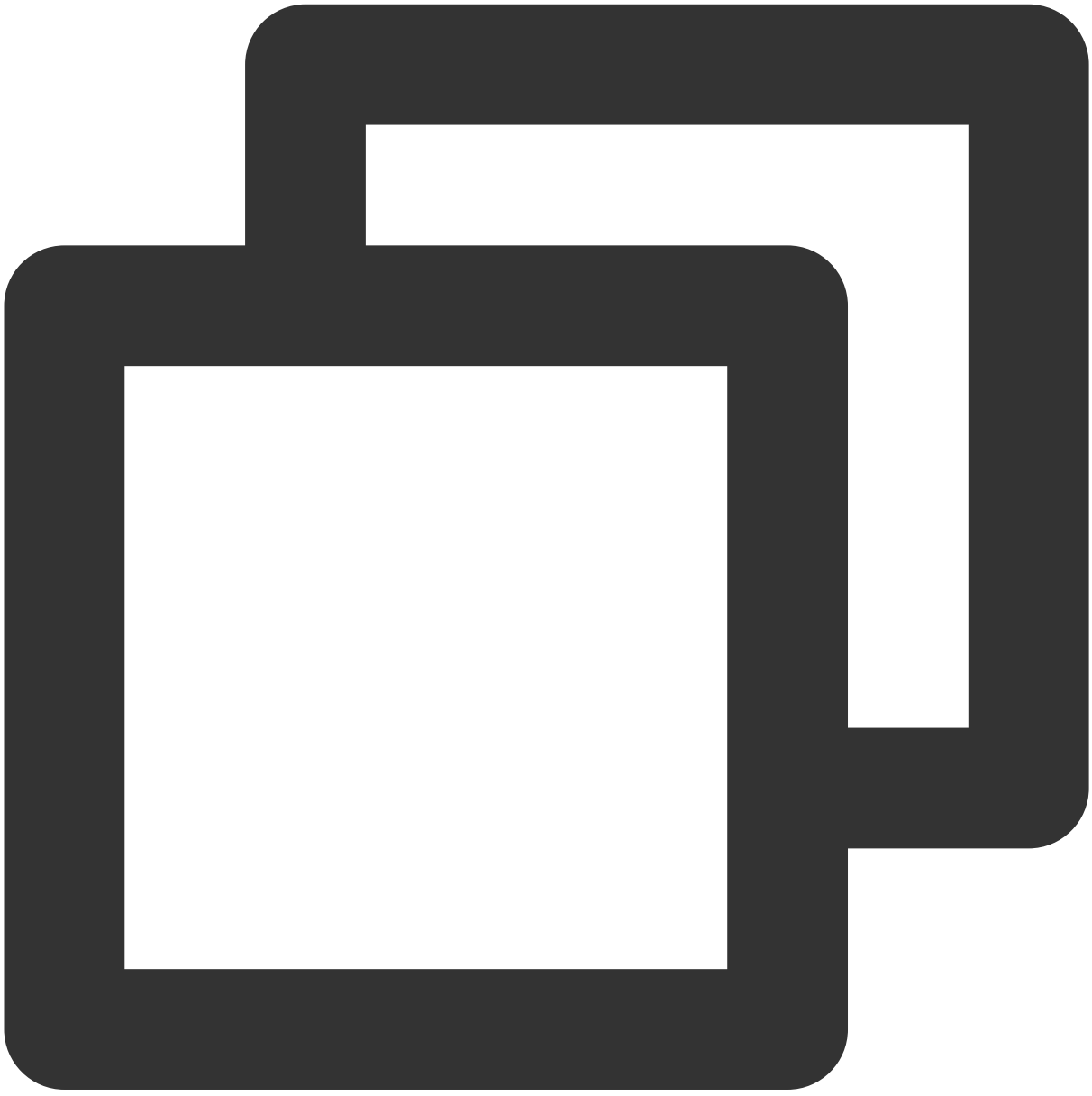
API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable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



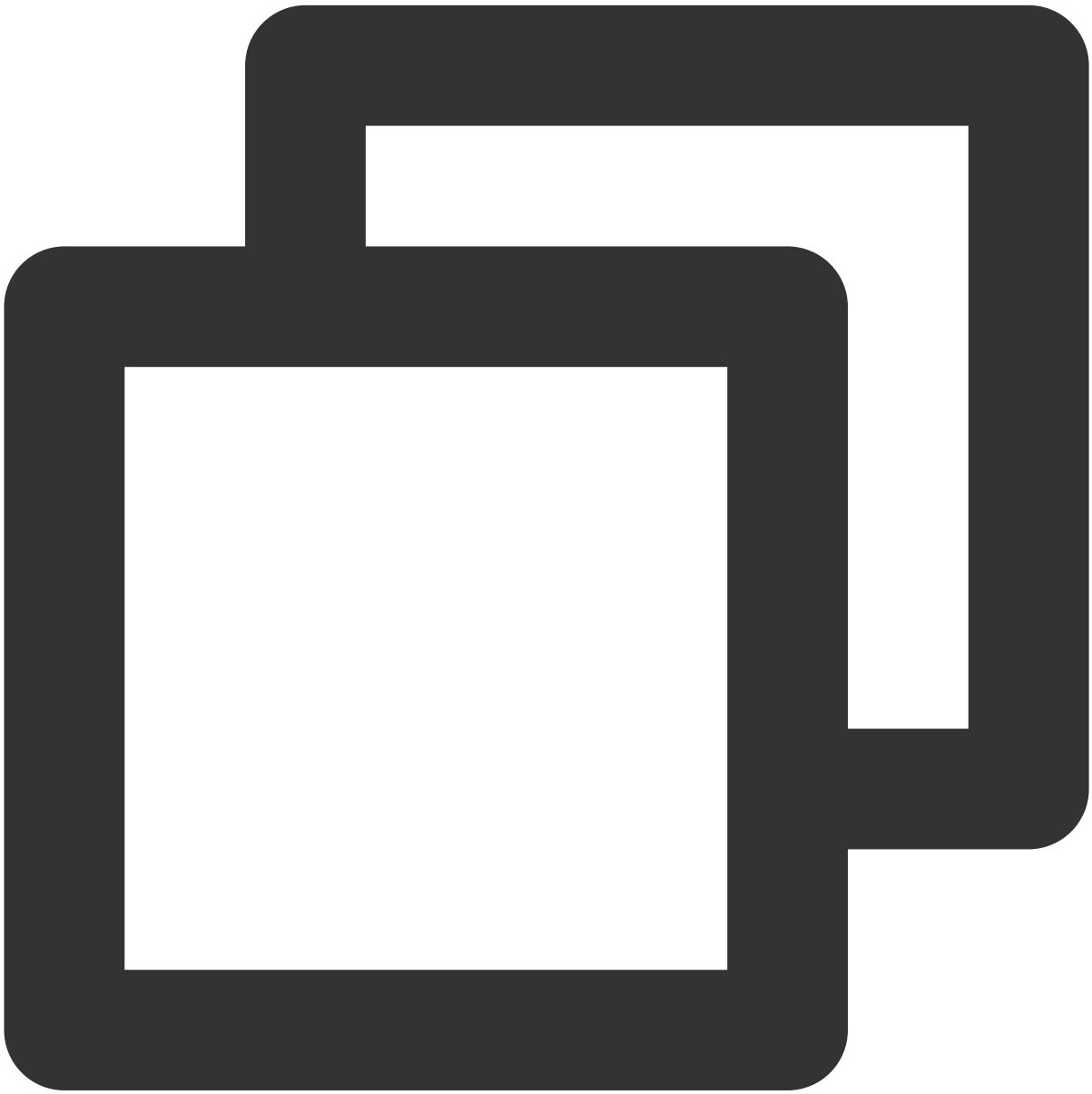
```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngin
```

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。



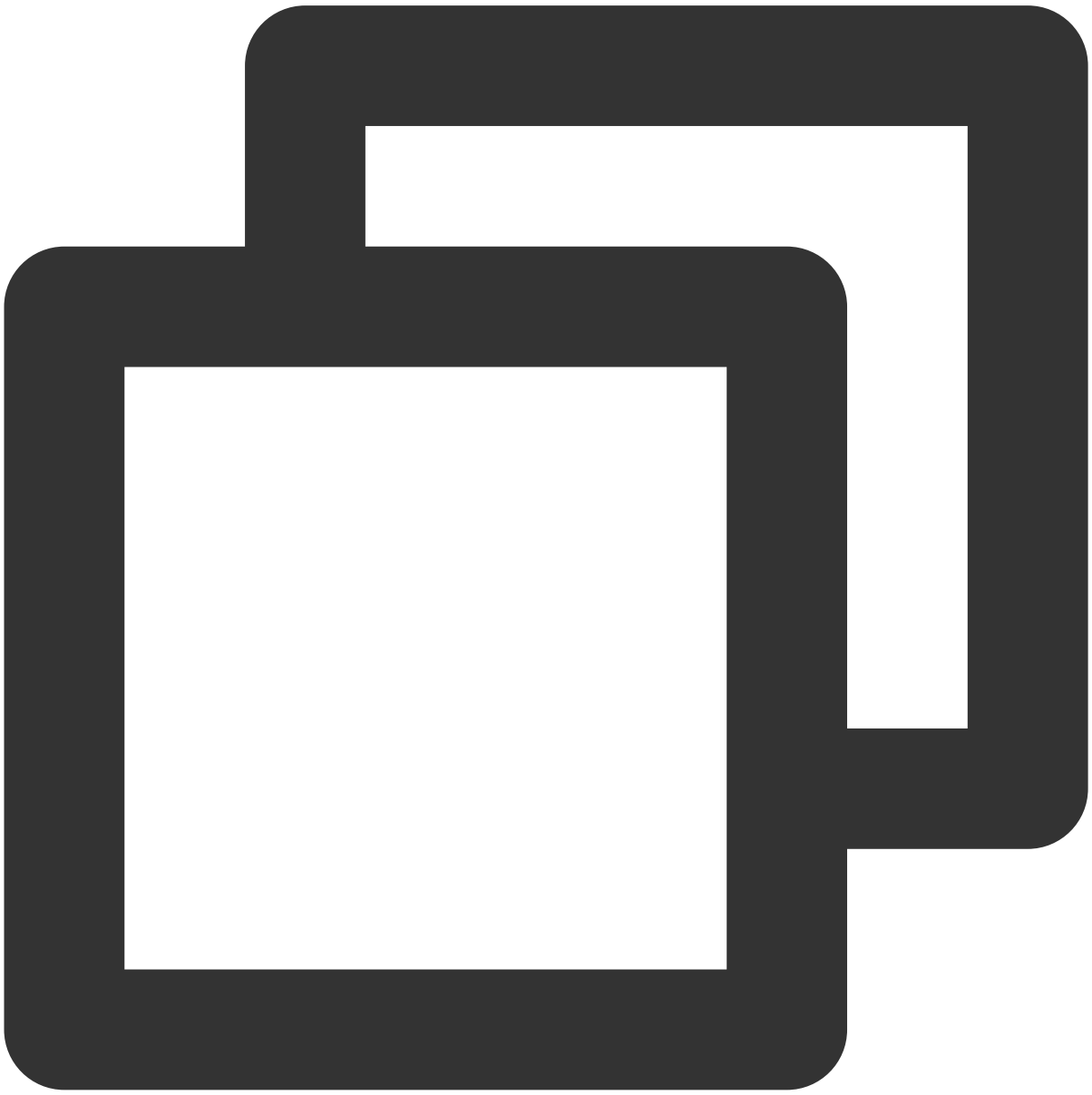
```
TUICallEngine.addEventListener('onError', (res) => {
  console.log('onError', JSON.stringify(res));
});
```

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。



```
TUICallEngine.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**）。

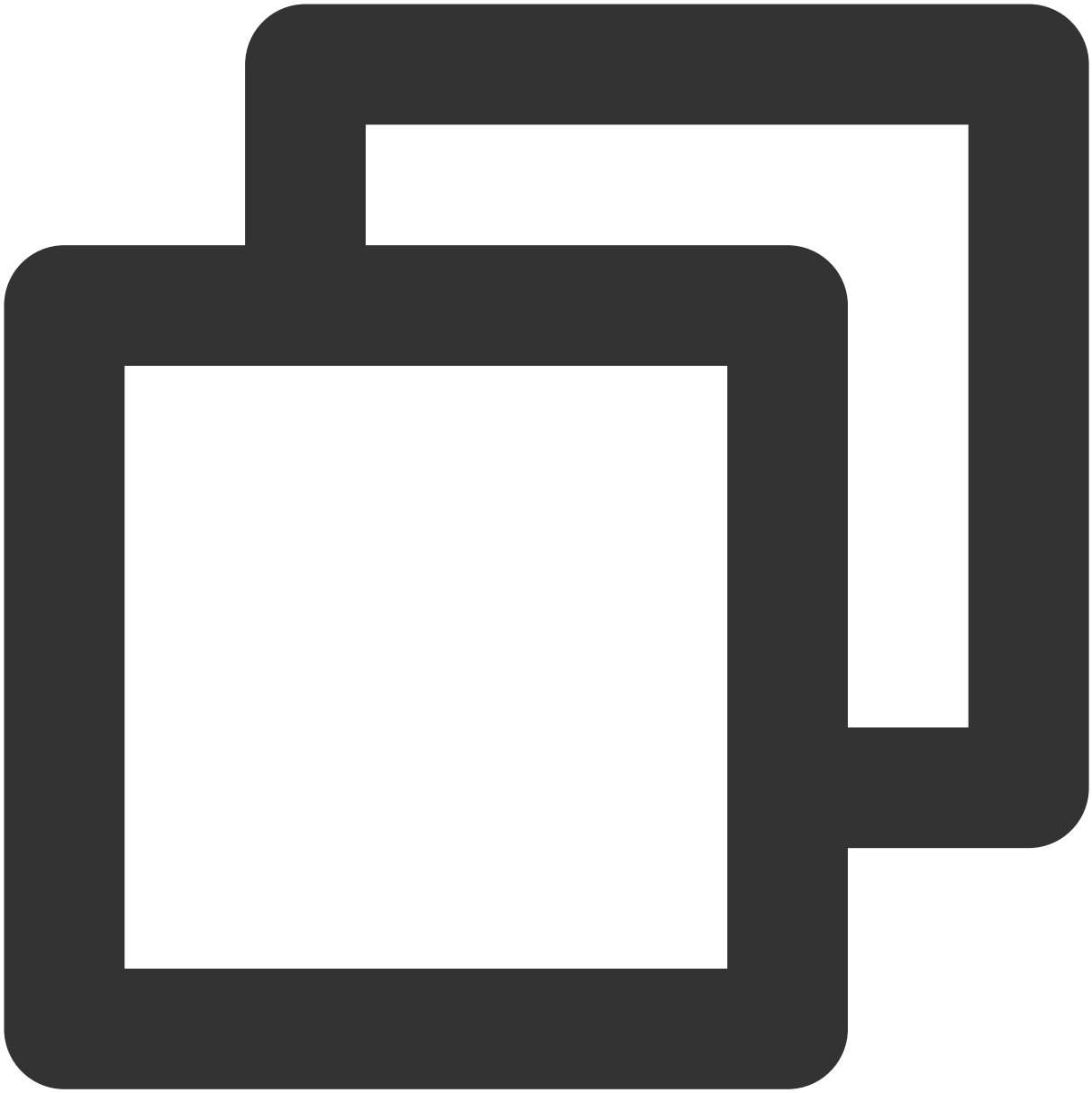
被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）。

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）。

版权所有：腾讯云计算（北京）有限责任公司



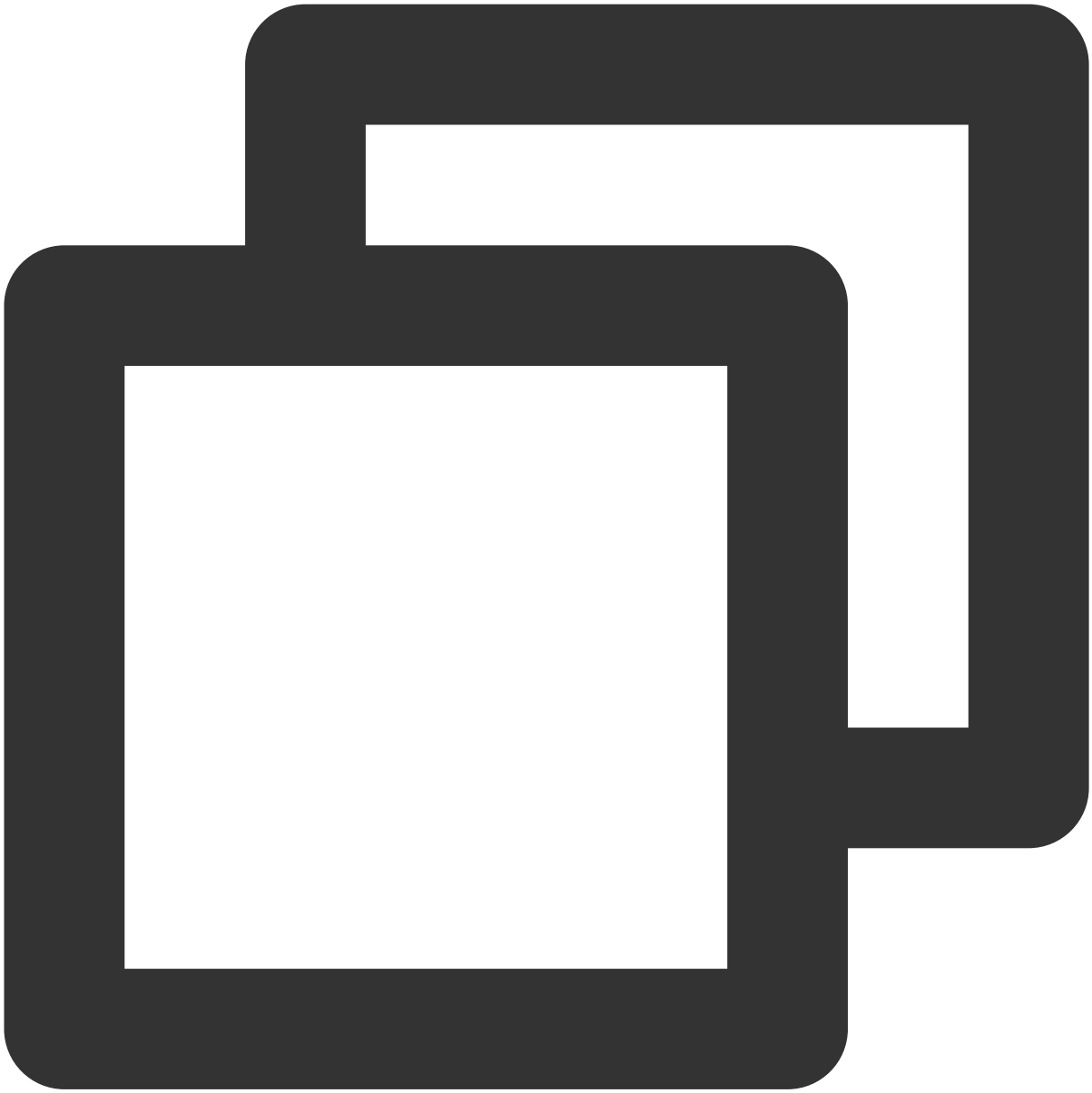
```
TUICallEngine.addEventListener('onCallCancelled', (res) => {
  console.log('onCallCancelled', res);
});
```

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。



```
TUICallEngine.addEventListener('onCallBegin', (res) => {
  console.log('onCallBegin', JSON.stringify(res));
});
```

参数如下表所示：

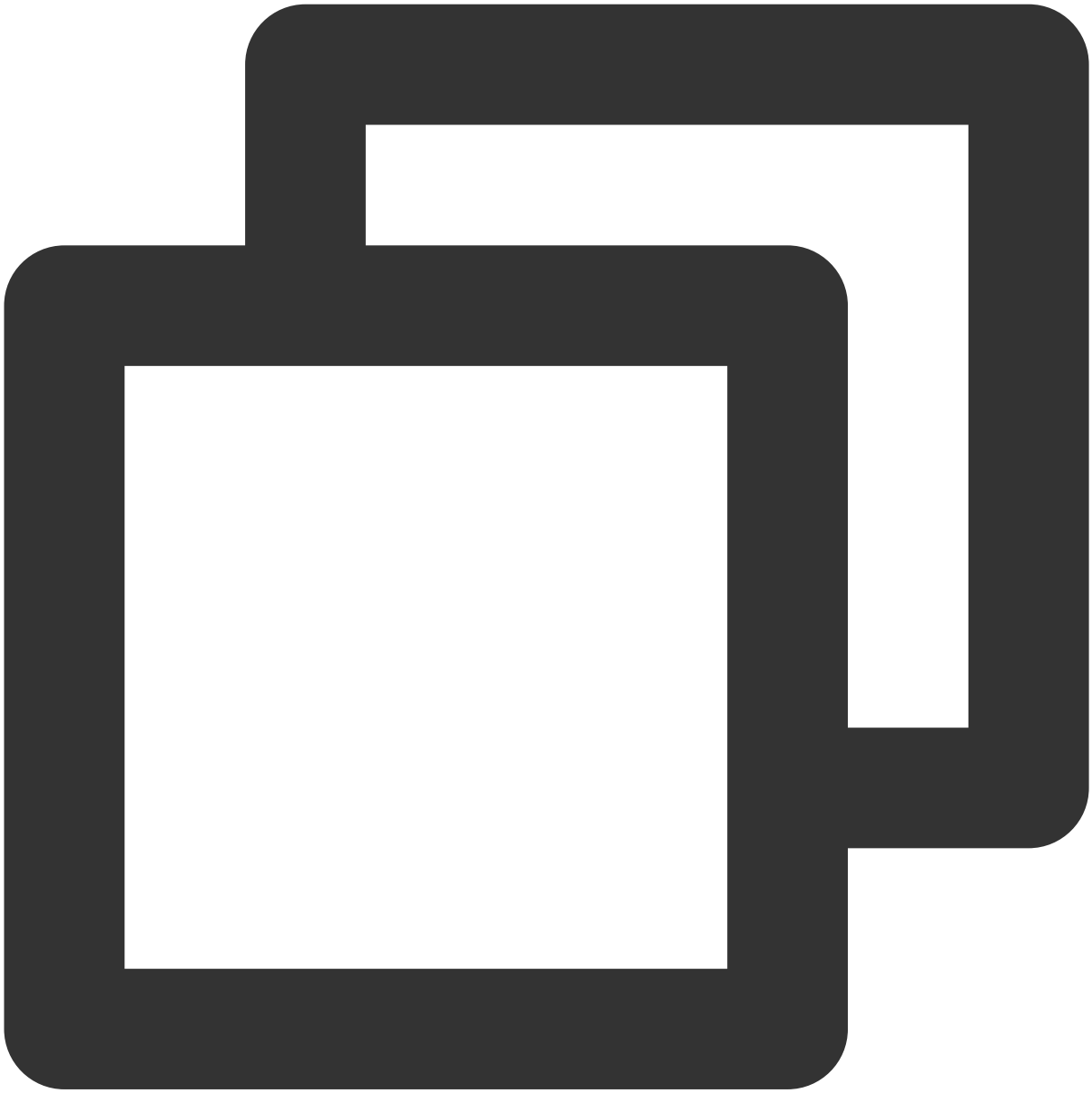
参数	类型	含义

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。

版权所有：腾讯云计算（北京）有限责任公司



```
TUICallEngine.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

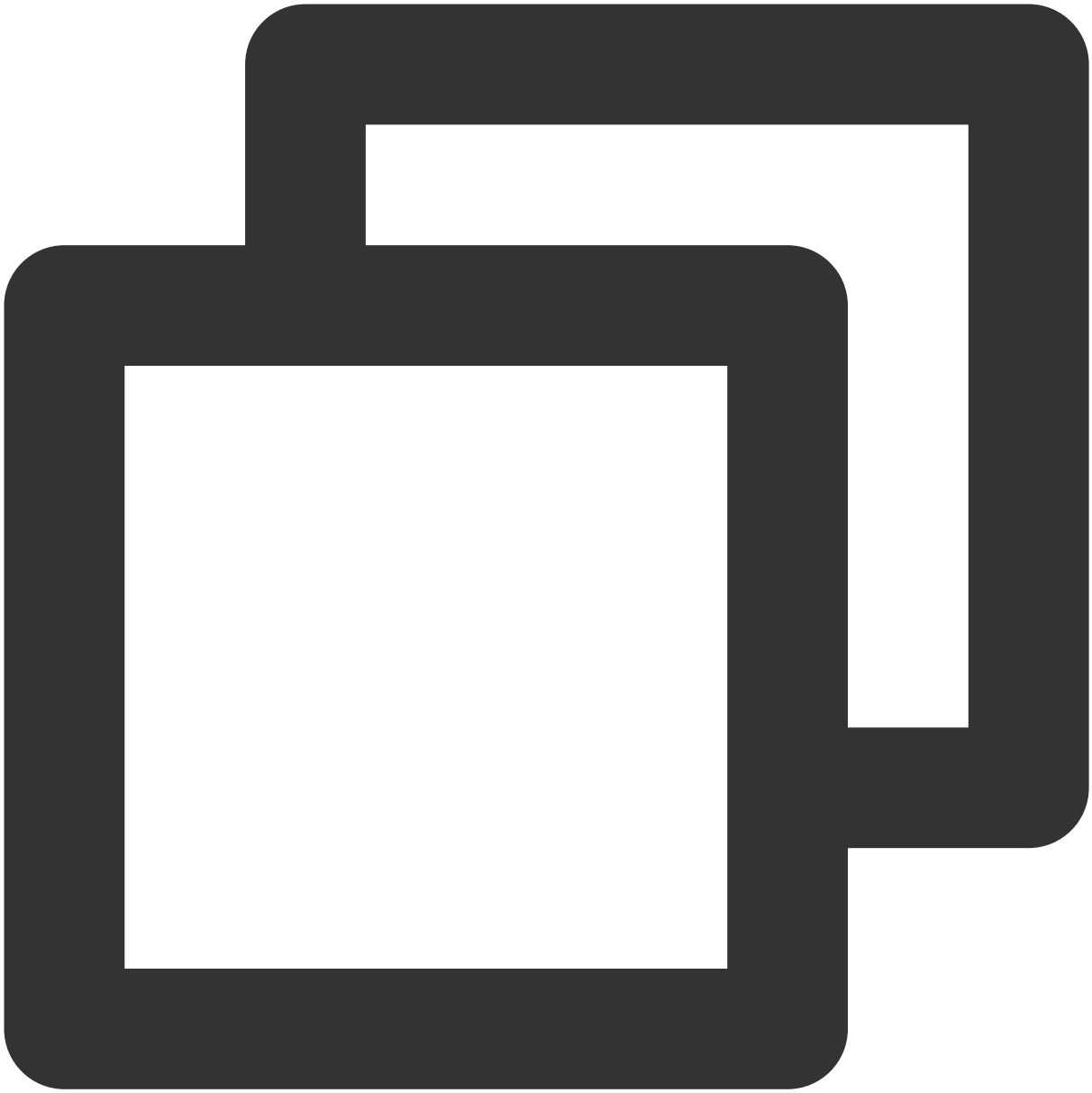
		房间号
res.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)
res.totalTime	Number	此次通话的时长，单位：秒

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。



```
TUICallEngine.addEventListener('onCallMediaTypeChanged', (res) => {
  console.log('onCallMediaTypeChanged', JSON.stringify(res));
});
```

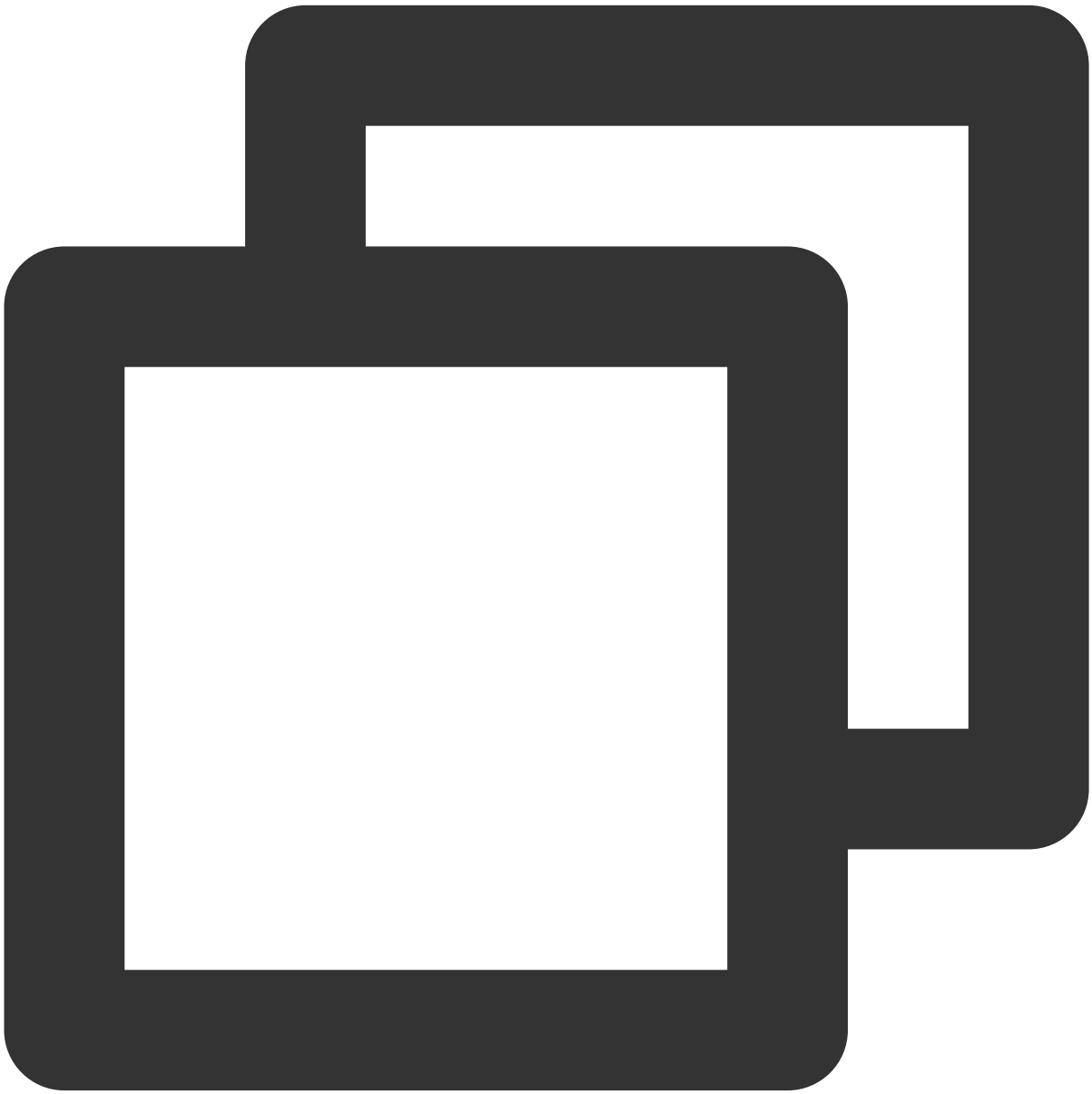
参数如下表所示：

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。



```
TUICallEngine.addEventListener('onUserReject', (res) => {
  console.log('onUserReject', JSON.stringify(res));
```

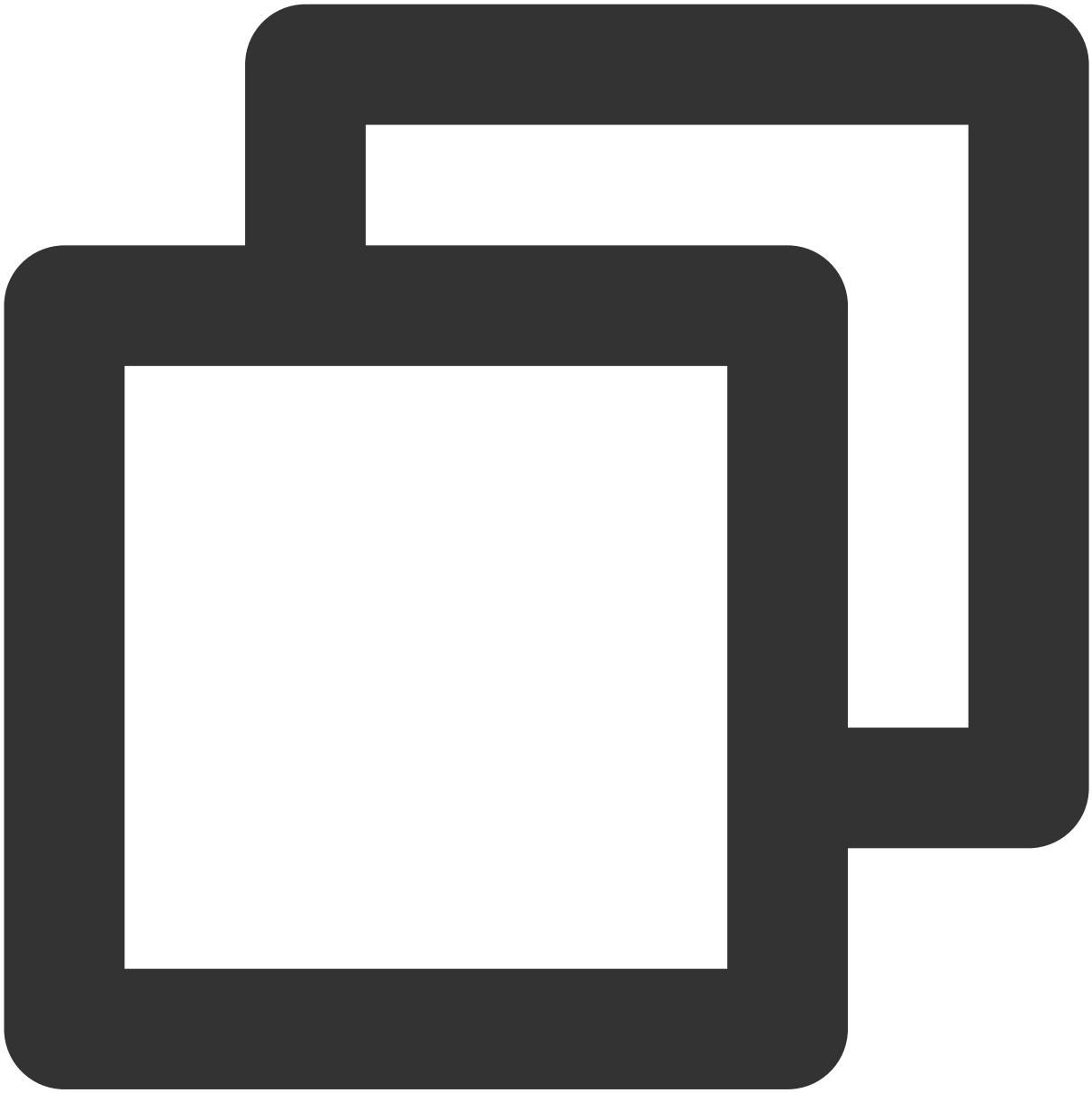
```
});
```

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。



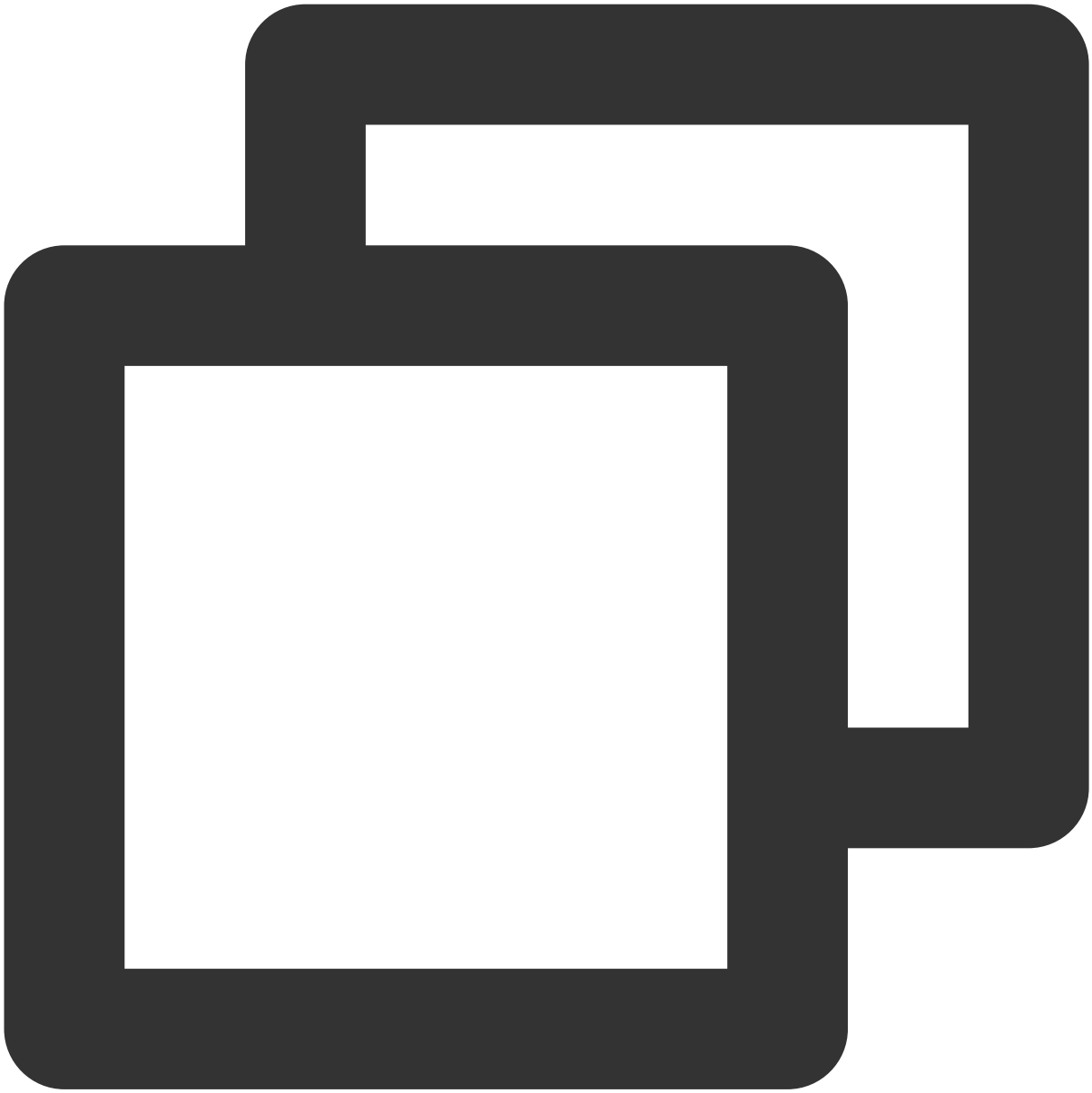
```
TUICallEngine.addEventListener('onUserNoResponse', (res) => {
  console.log('onUserNoResponse', JSON.stringify(res));
});
```

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。



```
TUICallEngine.addEventListener('onUserLineBusy', (res) => {
  console.log('onUserLineBusy', JSON.stringify(res));
});
```

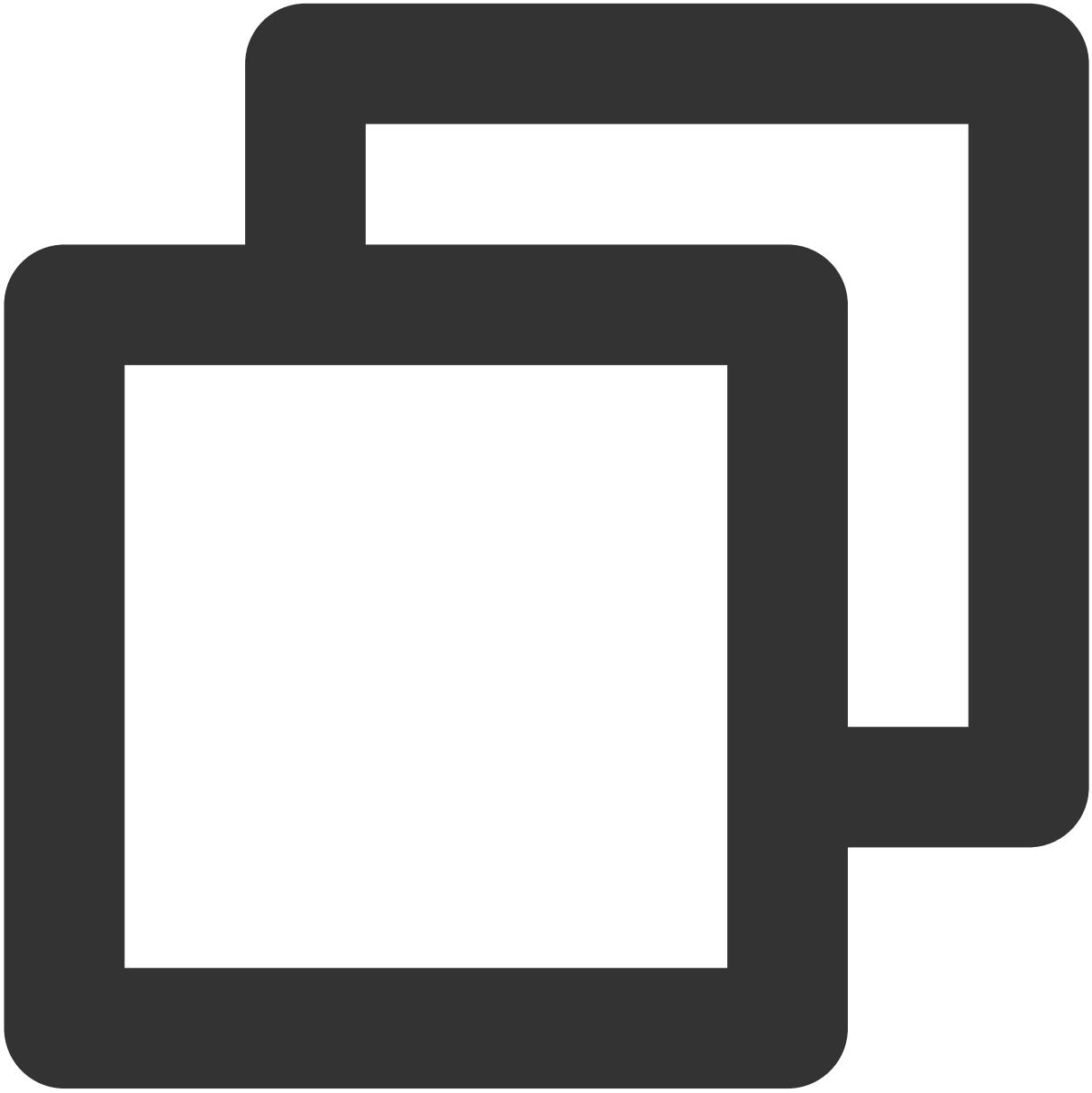
参数如下表所示：

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。



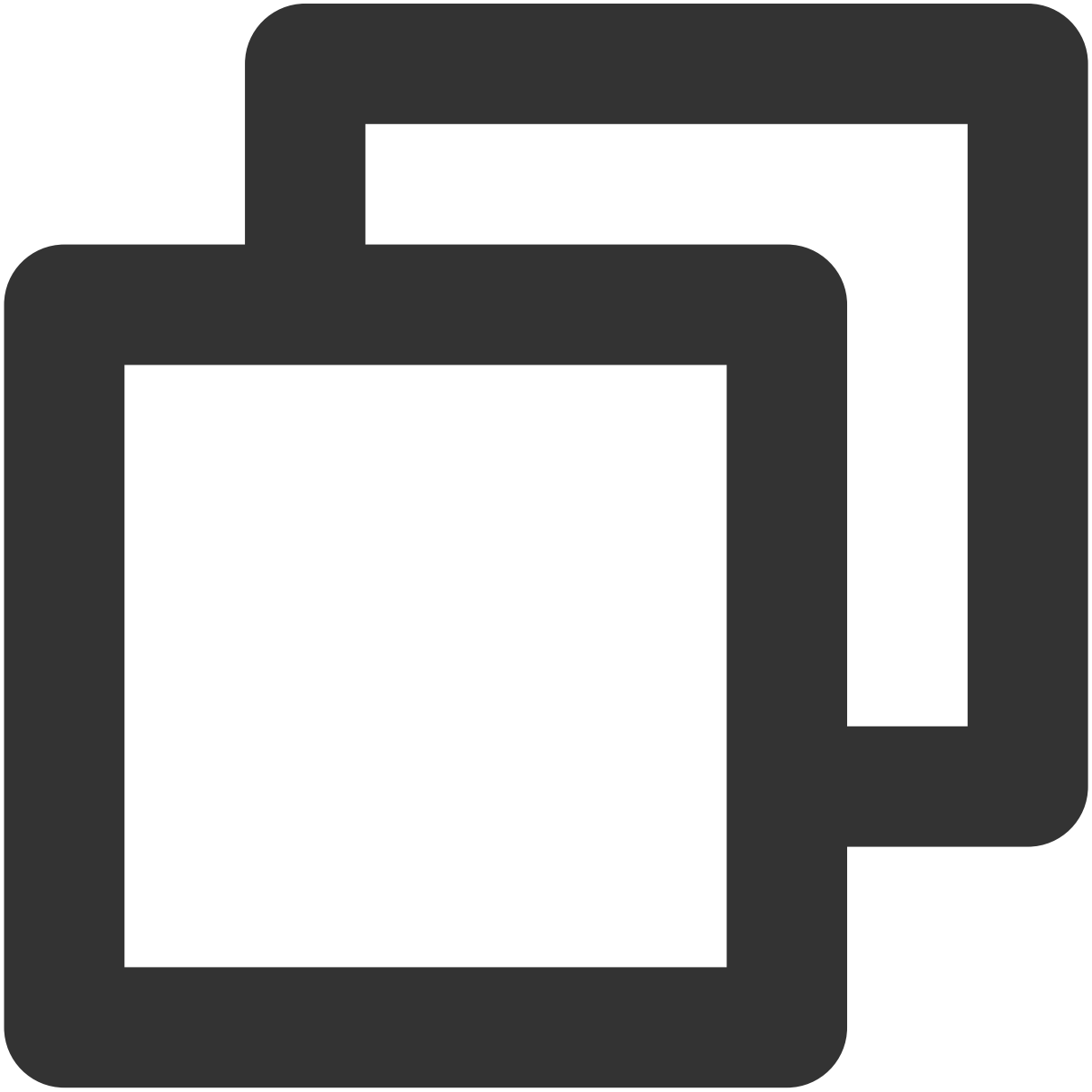
```
TUICallEngine.addEventListener('onUserJoin', (res) => {
  console.log('onUserJoin', JSON.stringify(res));
});
```

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。



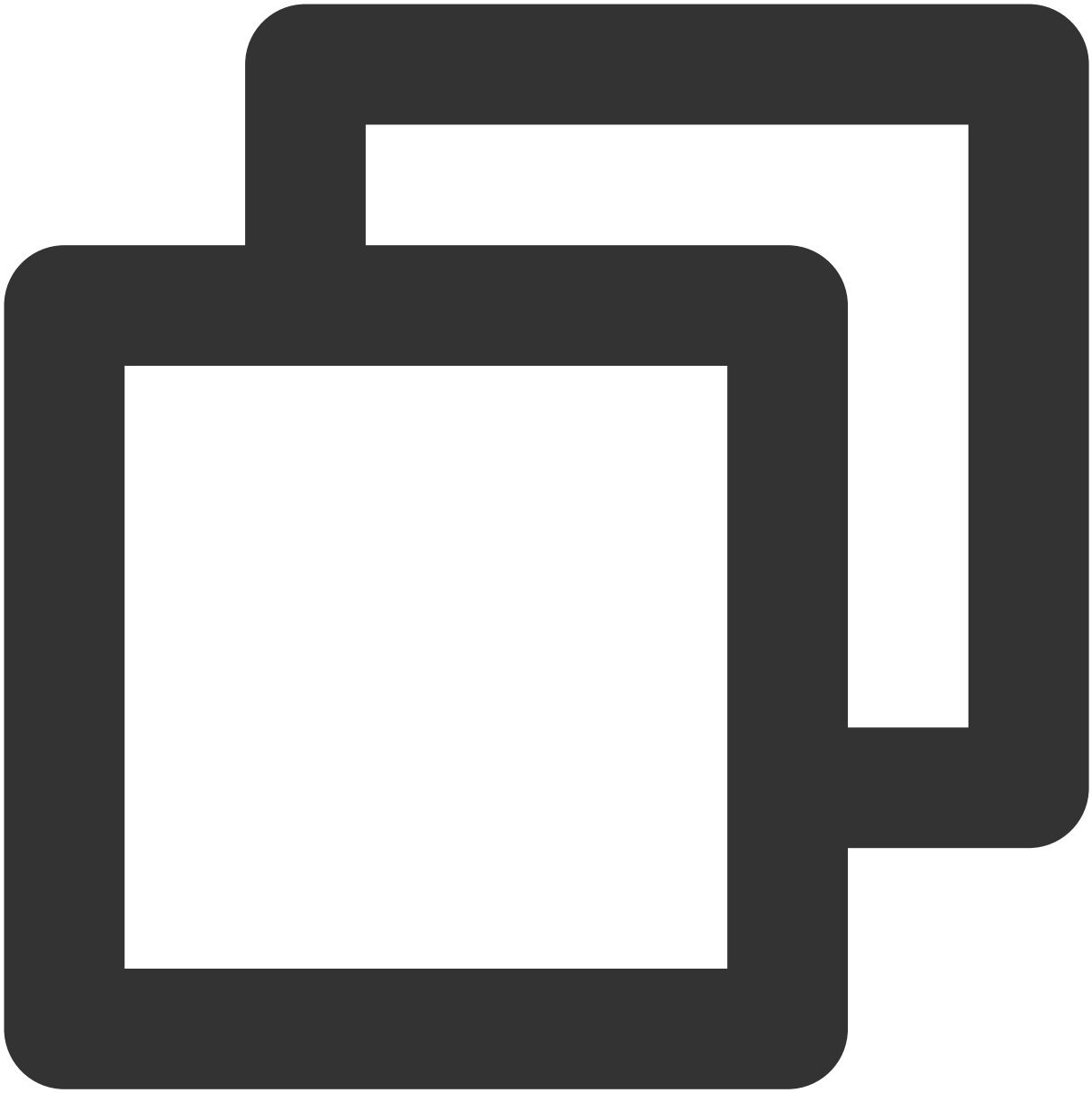
```
TUICallEngine.addEventListener('onUserLeave', (res) => {
  console.log('onUserLeave', JSON.stringify(res));
});
```

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。



```
TUICallEngine.addEventListener('onUserVideoAvailable', (res) => {
  console.log('onUserVideoAvailable', JSON.stringify(res));
});
```

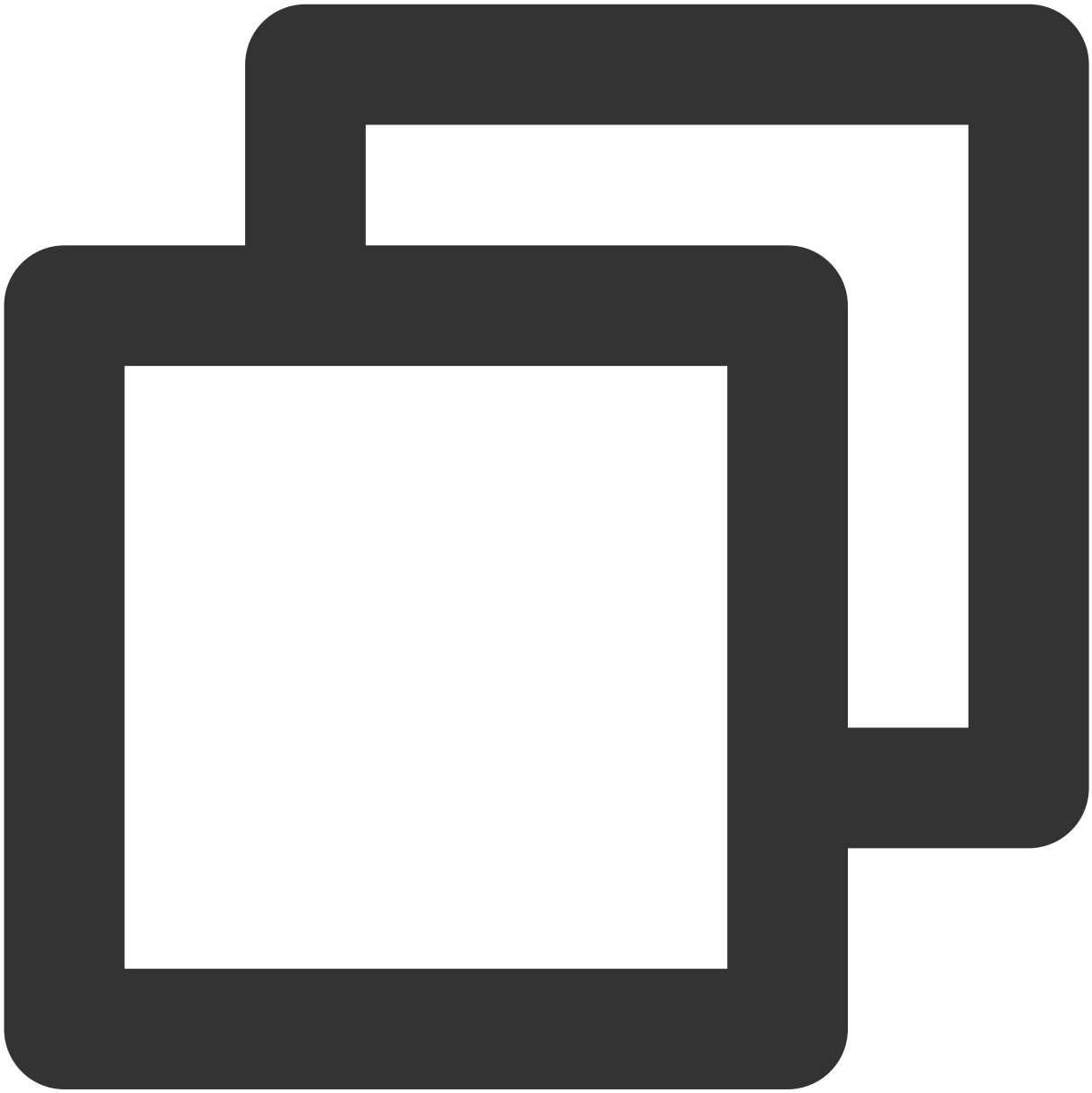
参数如下表所示：

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。



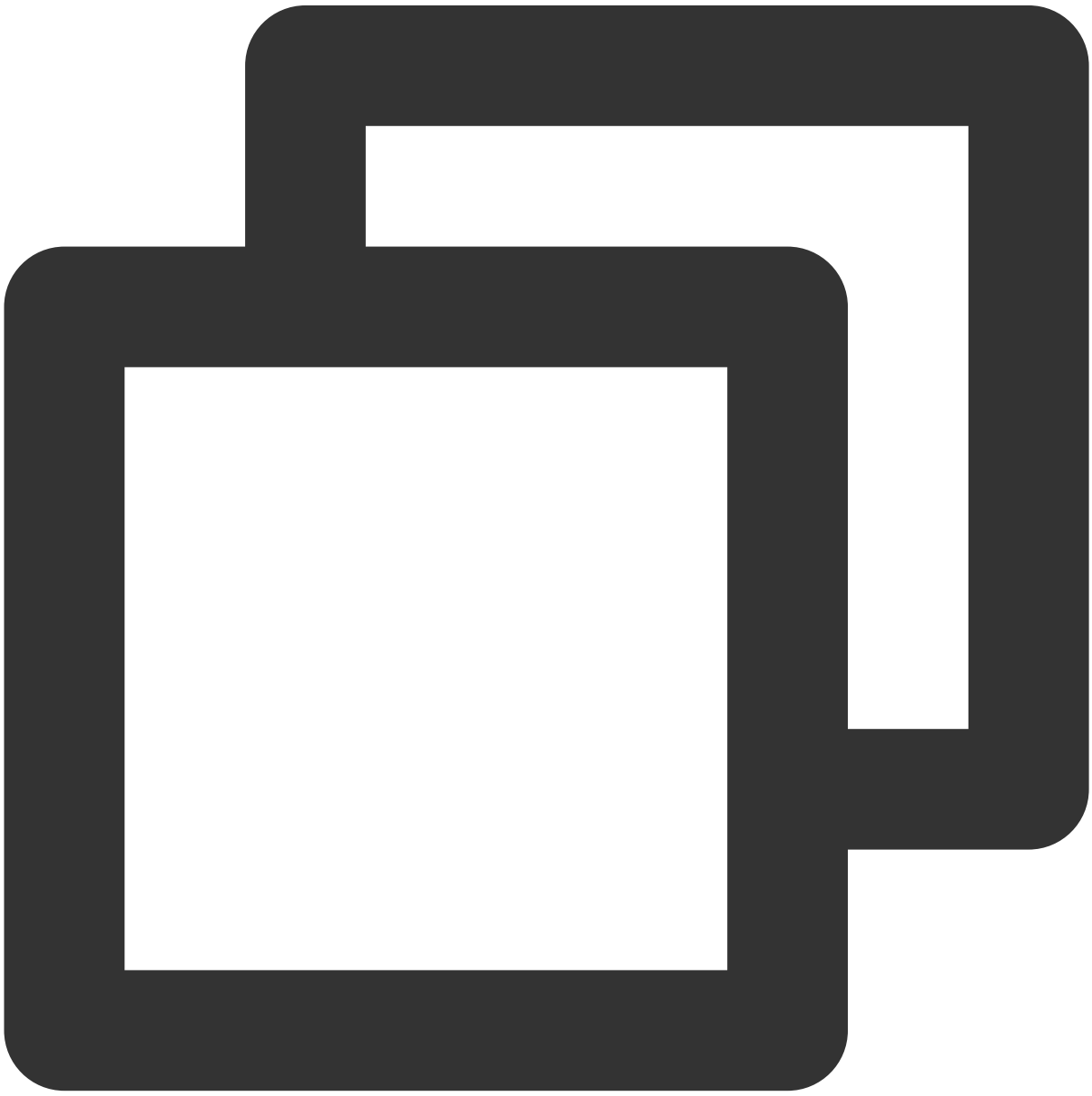
```
TUICallEngine.addEventListener('onUserAudioAvailable', (res) => {
  console.log('onUserAudioAvailable', JSON.stringify(res));
});
```

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。



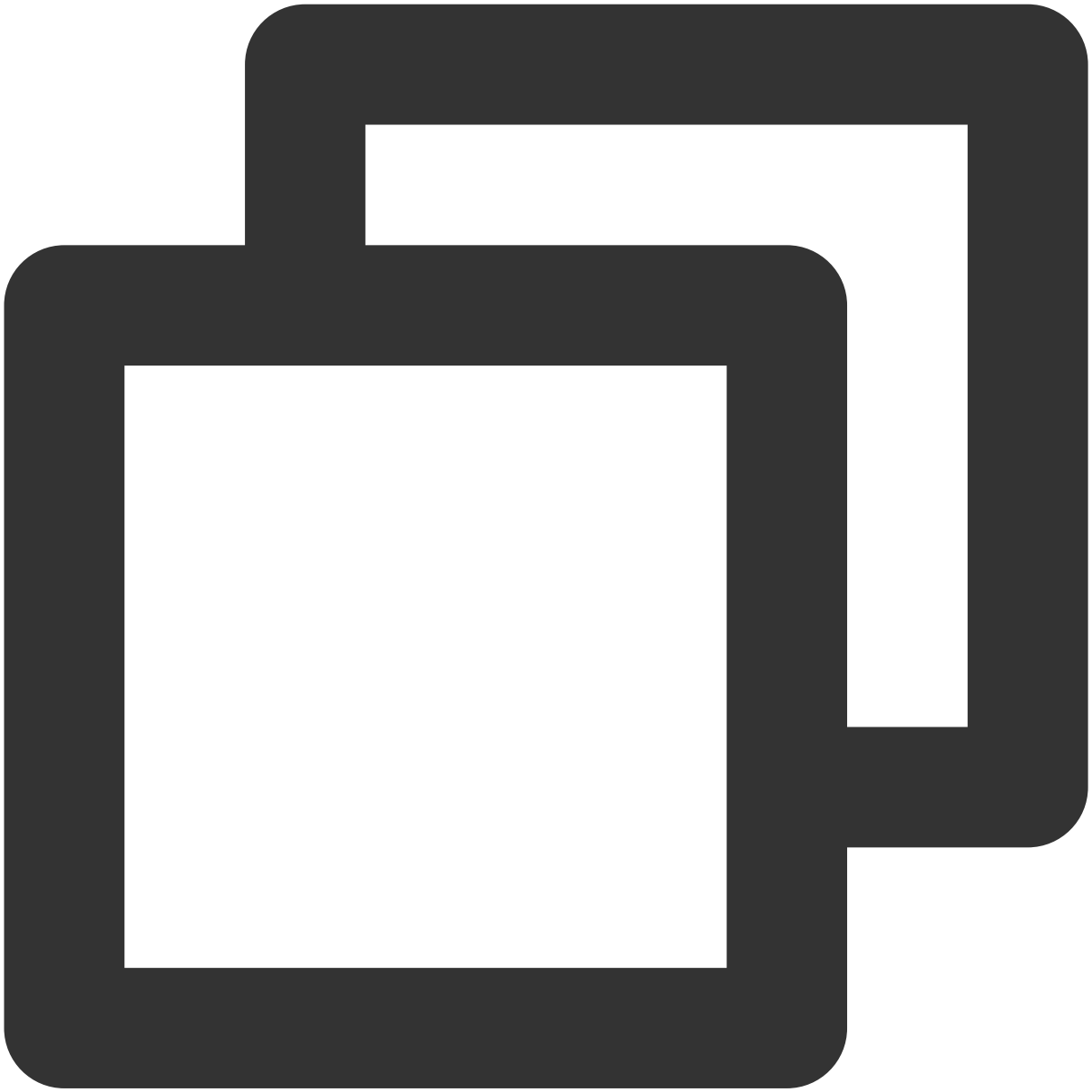
```
TUICallEngine.addEventListener('onUserVoiceVolumeChanged', (res) => {
  console.log('onUserVoiceVolumeChanged', JSON.stringify(res));
});
```

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。



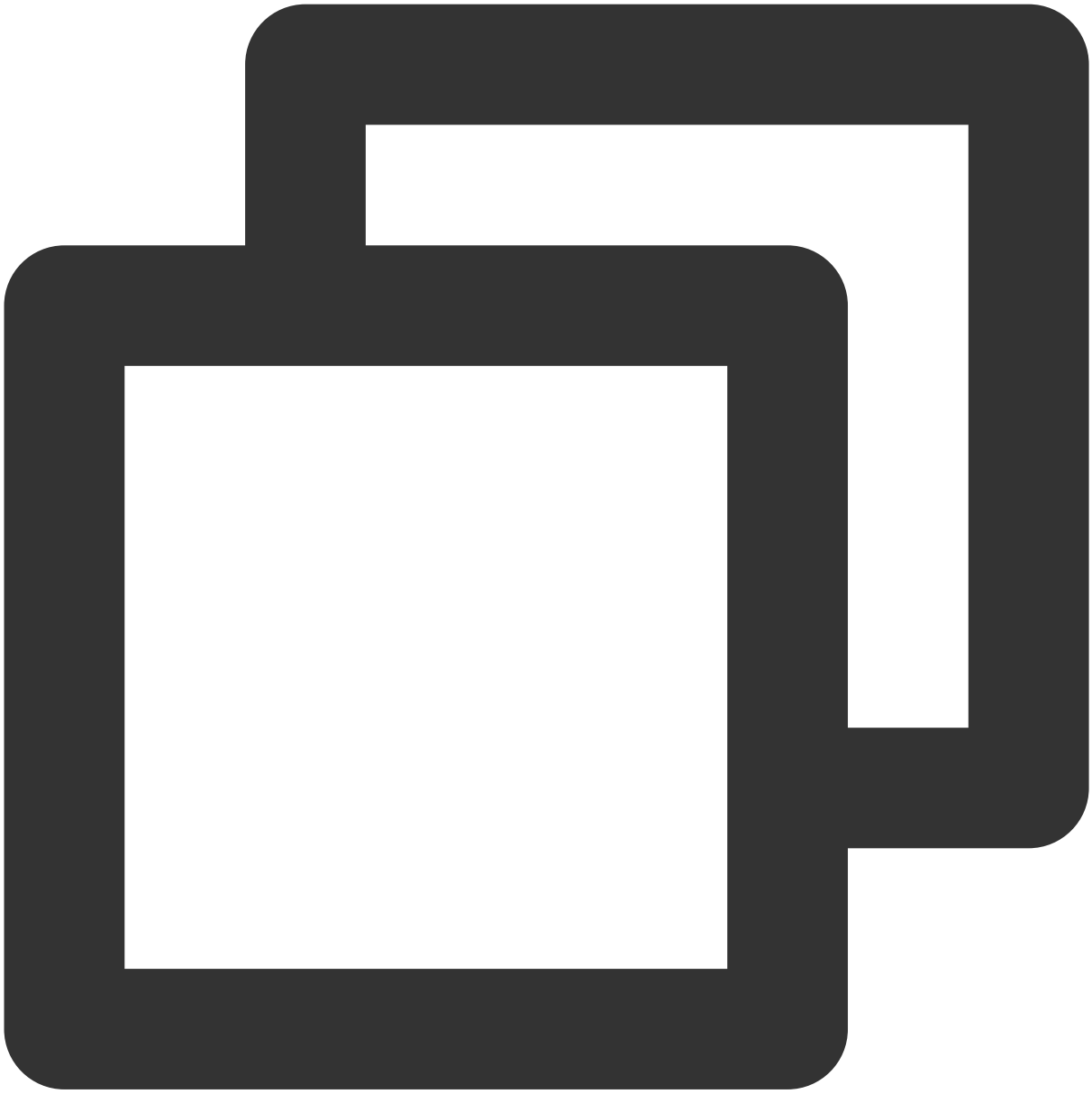
```
TUICallEngine.addEventListener('onUserNetworkQualityChanged', (res) => {
  console.log('onUserNetworkQualityChanged', JSON.stringify(res));
});
```

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

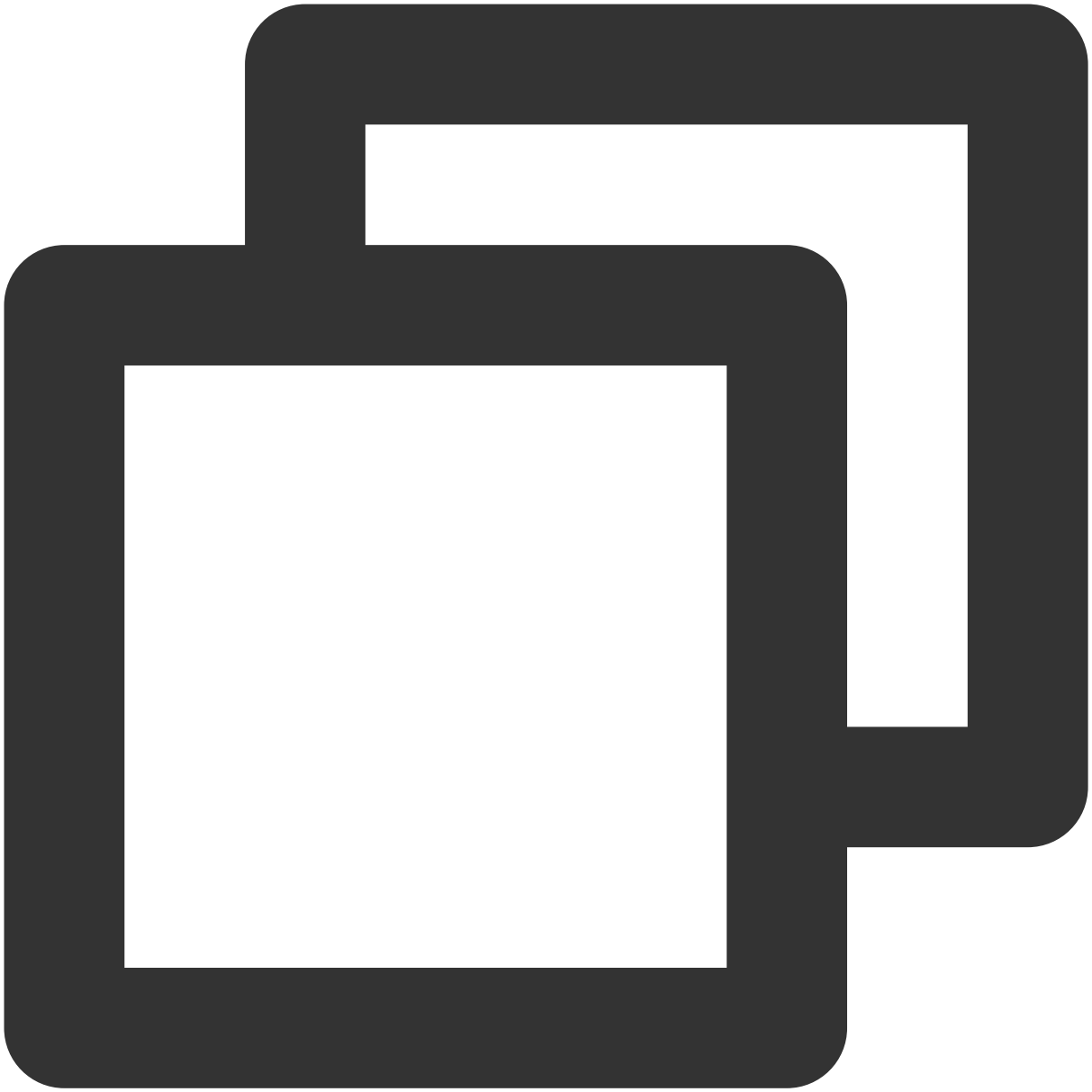
当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。



```
TUICallEngine.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。



```
TUICallEngine.addEventListener('onUserSigExpired', (res) => {
  console.log('onUserSigExpired', JSON.stringify(res));
});
```

错误码表（TUICallkit）

最近更新时间：2024-07-23 18:11:32

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	Chat SDK 版本过低，请升级 Chat 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

Chat 错误码

版权所有：腾讯云计算（北京）有限责任公司

音视频通话使用了腾讯云 Chat 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	Chat 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 Chat SDK 错误码详见：Chat 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：2024-08-09 22:25:01

说明：

TUICallKit Vue3 Github 版本地址：Github TUICallKit Web。

Version 3.3.3 @2024.08.06

新增

新增开启/关闭扬声器功能。

优化

react：优化按钮 loading 效果。
优化铃声实例的创建、销毁时机。
优化关闭摄像头后，翻转摄像头和虚拟背景按钮的交互。
优化用户未 init 时，直接调用 call/groupCall API 时的报错内容。
优化用户未 init 时，在页面引入 TUICallKit 组件，控制台的 setVideoResolution 警告现象。

修复

react：修复开虚拟背景按钮间距。
react：修复昵称没有截断问题。

Version 3.3.2 @2024.07.12

新增

新增支持邀请他人按钮显示隐藏。
网络状态较差时的提示。

修复

设备选择后重新拨打，设备选择列表选中设备与当前设备不符。
设备列表为空时，不显示设备选择列表。
修复中途加入的已知问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 3.3.1 @2024.06.25

修复

修复中途加入功能的一些问题。
删除 debug 文件的注释。

Version 3.3.0 @2024.06.14

新增

新增支持自定义大小窗显示和设置摄像头初始状态功能。
Vue&React：新增支持传入自定义字符串房间号。
React：新增 react TUICallKit 虚拟背景功能。

优化

扩展 offlinePushInfo 参数，支持离线推送声音设置等功能。

修复

修复被叫接通后，立刻自己超时后通话异常问题。
修复群通话被叫等待接听页面远端用户昵称显示问题。

优化

UI 定制化接口增加日志上报。

Version 3.2.9 @2024.05.29

优化

UI 定制化接口增加日志上报。

Version 3.2.8 @2024.05.27

修复

修复 SDK 引入 ref 路径错误问题。

Version 3.2.7 @2024.05.17

新增

新增中途加入 UI 组件。
新增自定义 UI 接口，支持设置通话背景和按钮隐藏。
调整发起通话时的参数校验，支持字符串房间号。

Version 3.2.4 @2024.05.06

新增

视频通话支持背景虚化。

修复

修复 uni-app 打包 web 项目，image 加载 icon 失败问题。
修复群通话切换摄像头按钮问题。
优化按钮点击后，应用卡住，再次点击按钮引起的异常问题。

Version 3.2.3 @2024.04.19

新增

新增群通话支持前后置摄像头切换。

优化

优化 TUICallKit SDK Readme。

Version 3.2.2 @2024.03.25

新增

新增全新的 UI 视效，功能更清晰，体验更好。

优化

优化使用 TUICallKit 的数据上报

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

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）。

版权所有：腾讯云计算（北京）有限责任公司

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。
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。

版权所有：腾讯云计算（北京）有限责任公司

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。

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”。

版权所有：腾讯云计算（北京）有限责任公司

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。

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。
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。

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。

版权所有：腾讯云计算（北京）有限责任公司

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：2024-08-09 20:14:11

Version 2.5.0.1025 @ 2024.8.7

功能优化

Android：优化中途加入的逻辑，避免内存泄露。

缺陷修复

Android&iOS：修复发送群通话邀请给 web 端用户，web 用户收不到通话邀请的问题。
Android：修复群组通话中 A 语音呼叫 B，B 收到推送通知后点击拉起界面，显示是扬声器而不是听筒问题。

Version 2.4.0.970 @ 2024.6.11

功能优化

Android&iOS：弱网时提示用户，增强用户使用体验。
Android：优化被叫端锁屏时的来电策略。
iOS：修复群通话内存异常增长问题。

缺陷修复

Android&iOS：修复主动加入通话界面显示异常问题。

Version 2.3.0.915 @ 2024.4.15

功能优化

Android&iOS：支持在 TUIChat 的群组顶部显示通话状态，并允许群成员主动加入通话。
Android&iOS：优化来电弹出逻辑，默认展示横幅接听框。
Android&iOS：视频通话支持背景虚化。

缺陷修复

Android：修复通话记录编辑界面中，点击删除按钮无响应的问题。
iOS：修复在群通话中点击成员视图时，切换过程中出现重影的问题 。
iOS：修复音视频界面特定场景下不显示的问题。
Android&iOS：修复在呼叫忙线中的用户时，通话结束后缺少提示的问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 2.2.0.860 @ 2024.2.1

功能优化

Android&iOS：全新的UI视效，功能更清晰，体验更好。

缺陷修复

Android&iOS：修复在会议、直播场景下，接听来电后，麦克风和摄像头设备互相占用问题。

Version 2.1.0.810 @ 2023.12.19

功能优化

Android：优化未登录时调用 TUICallKit 接口异常的提示。
Android： 优化针对 Android 14平台(API 34) 的兼容性，详见：Android 14 行为变更。
Android&iOS：优化用户昵称的显示，按照如下顺序显示：用户备注 > 用户昵称 > 用户 Id，默认为 userId。

缺陷修复

iOS：修复群通话头像显示重叠问题。
iOS：修复视频通话时开启悬浮窗发消息再回到通话界面键盘无法收起问题。
iOS：修复视频通话中，先切换摄像头、关闭摄像头，再切换回原摄像头并重新打开时，无法切换和移动摄像头的问题。
Android：修复在阿拉伯语等从右至左布局模式(RTL 模式)下单人视频通话的小窗口无法移动的问题。

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

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。

版权所有：腾讯云计算（北京）有限责任公司

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

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。

版权所有：腾讯云计算（北京）有限责任公司

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 Chat 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。

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()。

版权所有：腾讯云计算（北京）有限责任公司

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：2024-04-28 10:40:58

Version 2.3.2 @2024.04.24

新特性

Android&iOS：优化来电弹出逻辑，默认展示横幅接听框。
Android&iOS：视频通话支持背景虚化。

Version 2.3.1 @2024.04.22

问题修复

修复在非 iOS&Android 平台上出现的 Method Channel 报错问题。

Version 2.3.0 @2024.04.18

新特性

Android&iOS：支持在 tencent_cloud_chat_uikit 的群组顶部显示通话状态，并允许群成员主动加入通话。

功能优化

Android&iOS：优化与 tencent_cloud_chat_uikit 同时使用时的登录方式。

依赖描述

升级 tencent_cloud_uikit_core 版本为1.6.0。
升级依赖客户端 TUICallEngine SDK 版本为 2.3.0.915.

Version 2.2.3 @2024.03.08

依赖描述

升级 tencent_cloud_uikit_core 版本为1.5.2。

Version 2.2.2 @2024.03.07

版权所有：腾讯云计算（北京）有限责任公司

新特性

Android&iOS：TUICallObserver 的 onCallReceived 回调新增用户自定义参数 userData。

Version 2.2.1 @2024.02.06

问题修复

Android：修复了杀死进程后，应用程序接收到了 FCM 推送，引起的应用页面异常问题。

Version 2.2.0 @2024.02.03

新特性

Android：支持 FCM 推送（使用 FCM 推送，需要使用 tencent_cloud_chat_push）。

Version 2.1.1 @2024.01.06

问题修复

Android&iOS：修复了一些 UI 3.0相关的 UI 问题。

Version 2.1.0 @2024.01.04

新特性

Android&iOS：升级为 UI 3.0。

Version 2.0.6 @2023.12.15

问题修复

iOS：修复了 iOS 离线推送信息中缺少某些字段的问题。

Version 2.0.5 @2023.12.10

问题修复

iOS：修复了 VoIP 推送期间进入通话页面时铃声异常以及远程流拉取异常的问题。
Android：修复了在未获得悬浮窗权限的情况下，后台时来电页面显示异常的问题。

Version 2.0.4 @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。

版权所有：腾讯云计算（北京）有限责任公司

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。

版权所有：腾讯云计算（北京）有限责任公司

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。

版权所有：腾讯云计算（北京）有限责任公司

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

缺陷修复

Android：修复通话结束后，`OnCallEnd`事件丢失的问题。

版权所有：腾讯云计算（北京）有限责任公司

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。

版权所有：腾讯云计算（北京）有限责任公司

常见问题（TUICallKit）
iOS

最近更新时间：2024-05-24 18:10:53

错误提示“The package you purchased does not support this ability”？

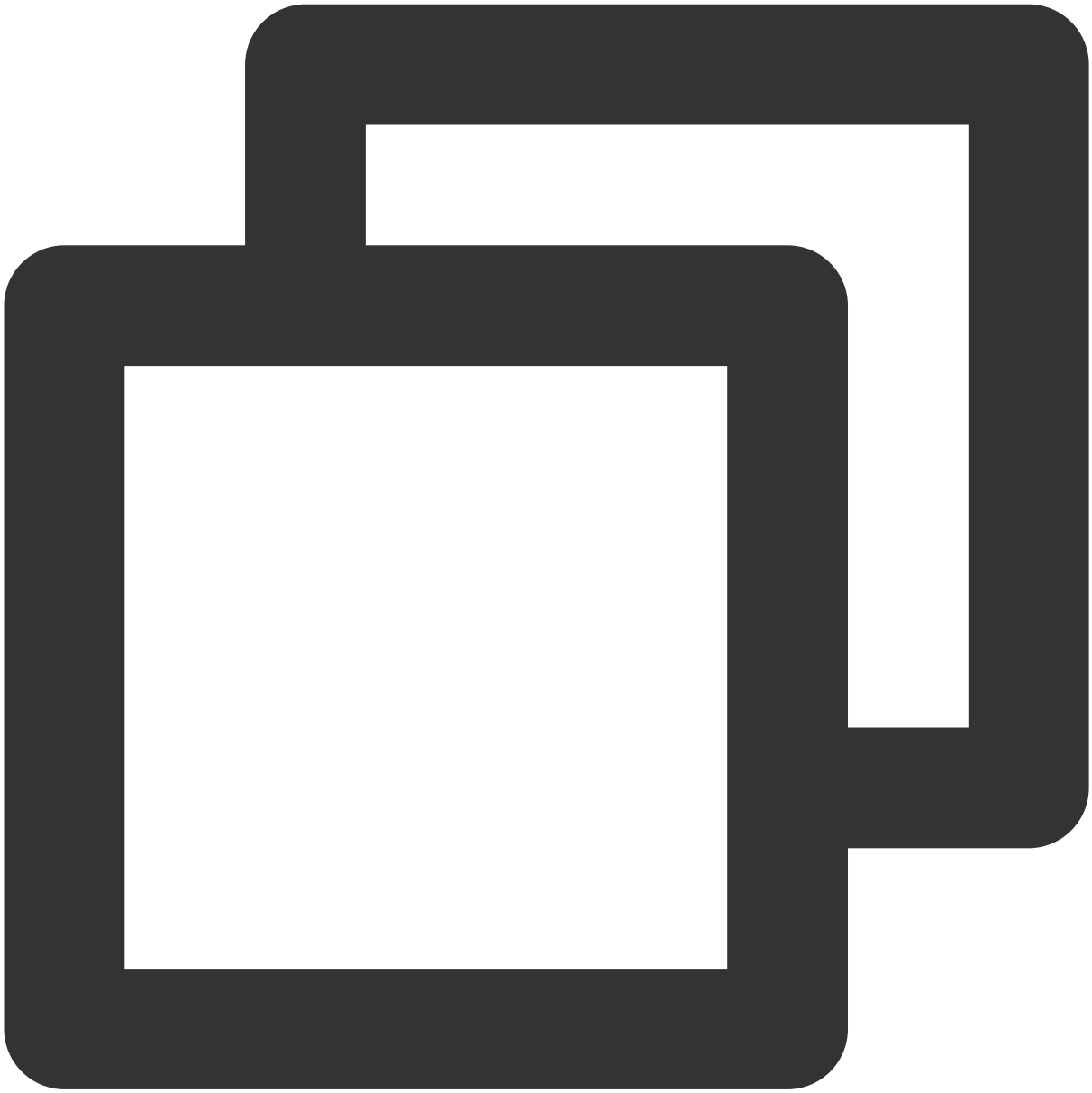
如遇以上错误提示，是由于您当前应用的音视频通话能力包过期或未开通，您可以参考 开通服务，领取或者开通音视频通话能力，进而继续使用TUICallKit 组件。

如何修改 TUICallKit 源码？

使用 `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 `TUICallKit` 文件夹。
2. 单击进入 **Github/TUICallKit** ，选择克隆/下载代码，然后将 **iOS** 目录下的 TUICallKit-Swift 文件夹和 TUICallKit-Swift.podspec 文件拷贝到您在 `步骤1` 创建的 `TUICallKit` 文件夹下。
3. 在您的 `Podfile` 文件中添加以下依赖。

版权所有：腾讯云计算（北京）有限责任公司



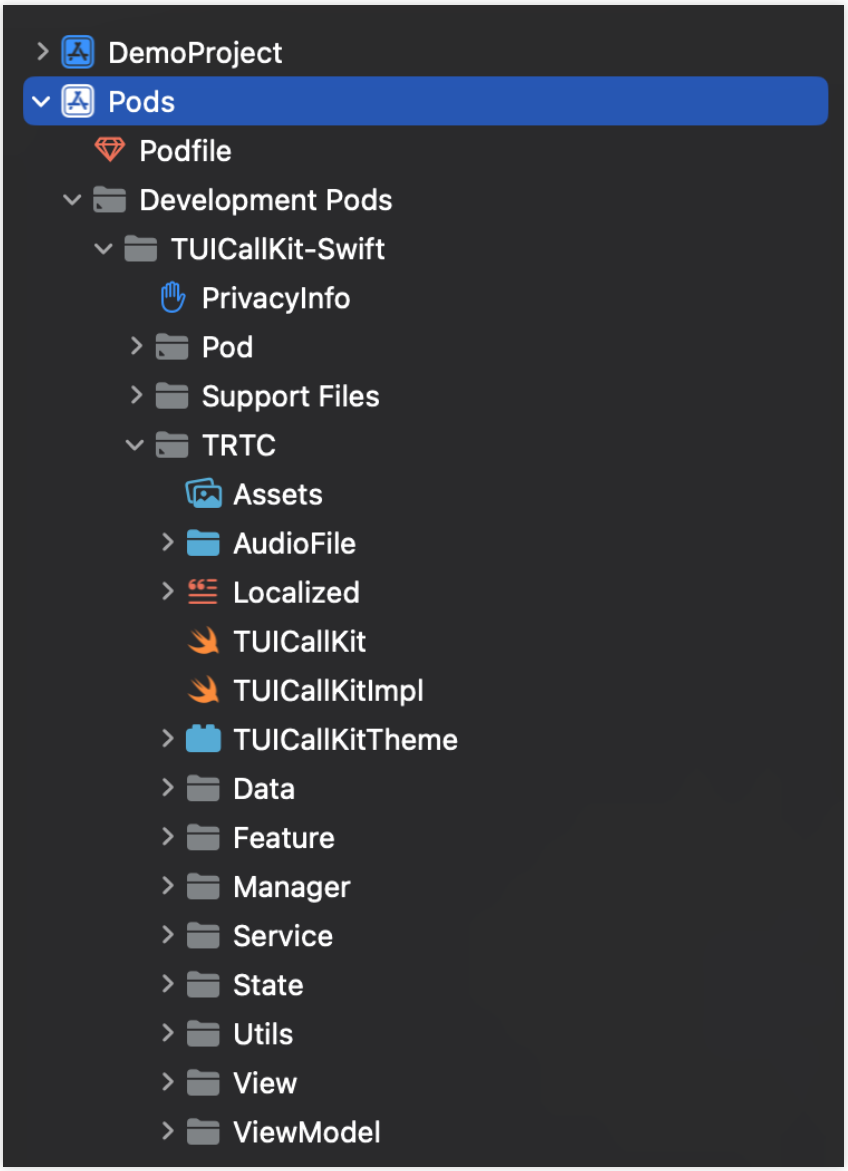
```
# :path => "指向 TUICallKit-Swift.podspec 的相对路径"
pod 'TUICallKit-Swift', :path => "TUICallKit/TUICallKit-Swift.podspec"
```

4. 执行 `pod install` 命令，完成导入。

注意

`TUICallKit-Swift` 文件夹和 `TUICallKit-Swift.podspec` 文件必需在同一目录下。

TUICallKit_Swift 组件集成后效果：

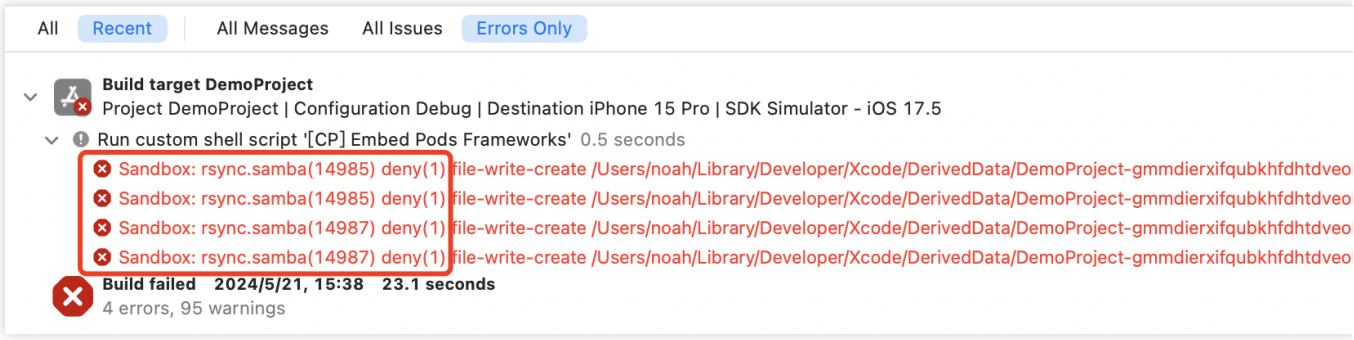


说明

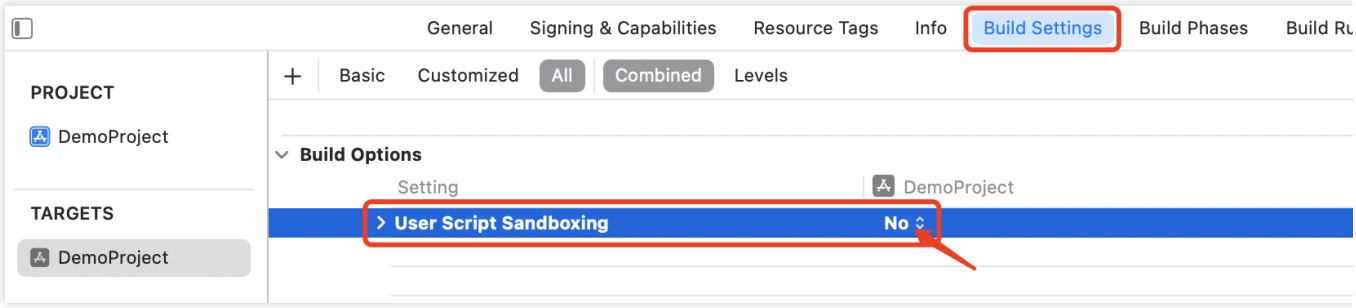
TUICallKit_Swift 组件集成后支持文件夹分层显示，方便您阅读和修改源代码。

Xcode 15 编译报错？

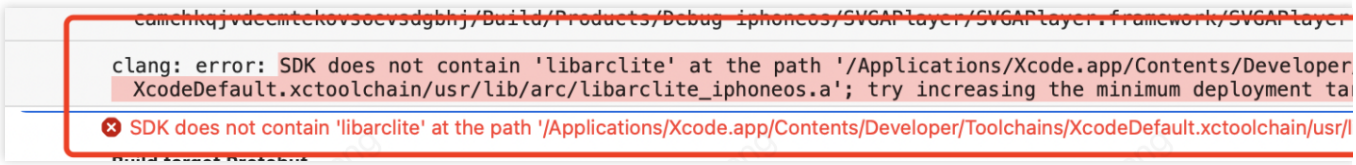
1、出现 Sandbox: rsync 编译报错



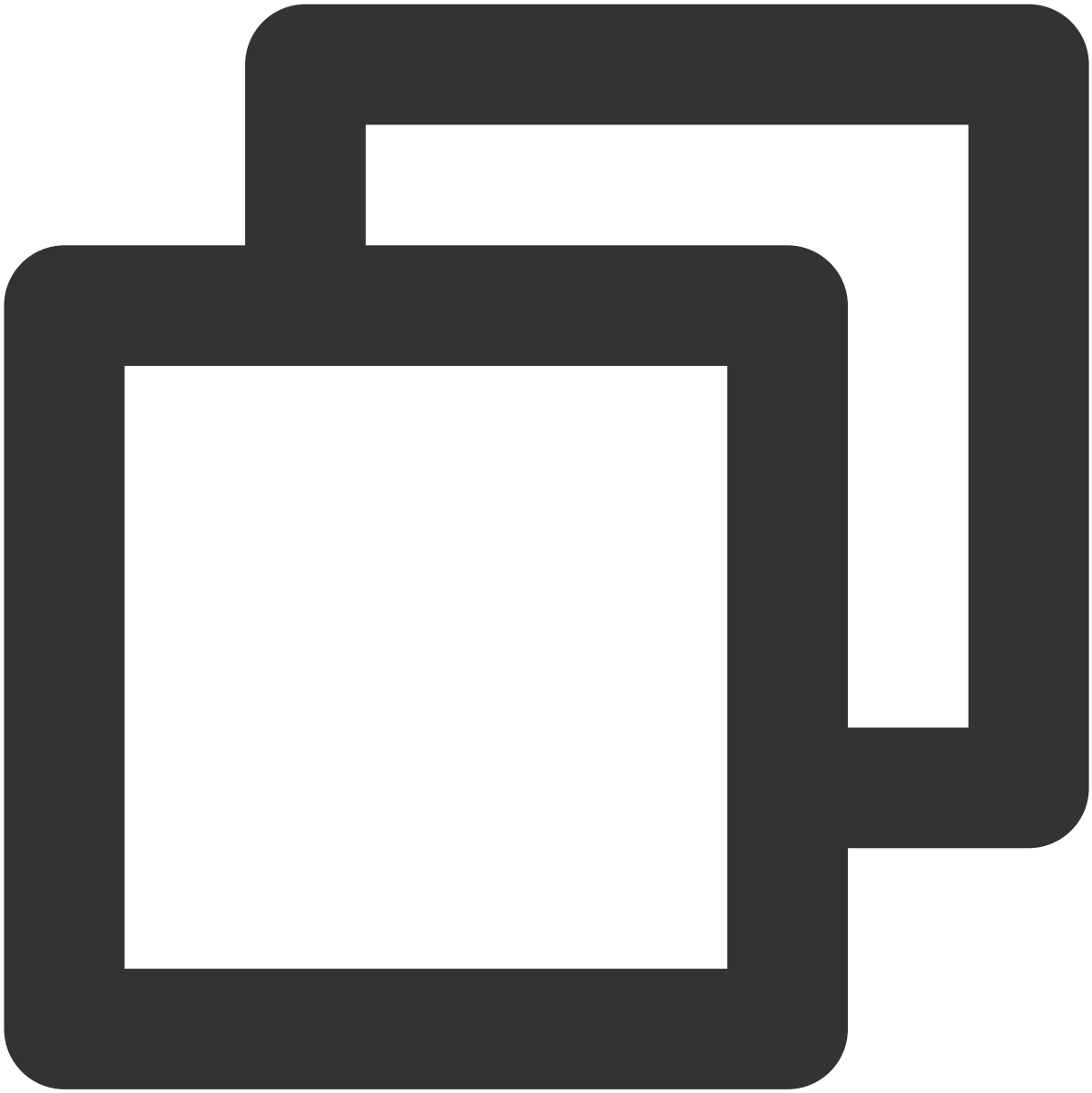
可以在 **Build Settings** 中把 **User Script Sandboxing** 设置为 **NO**



2、出现 SDK does not contain 编译报错



可以在 Podfile 添加如下代码

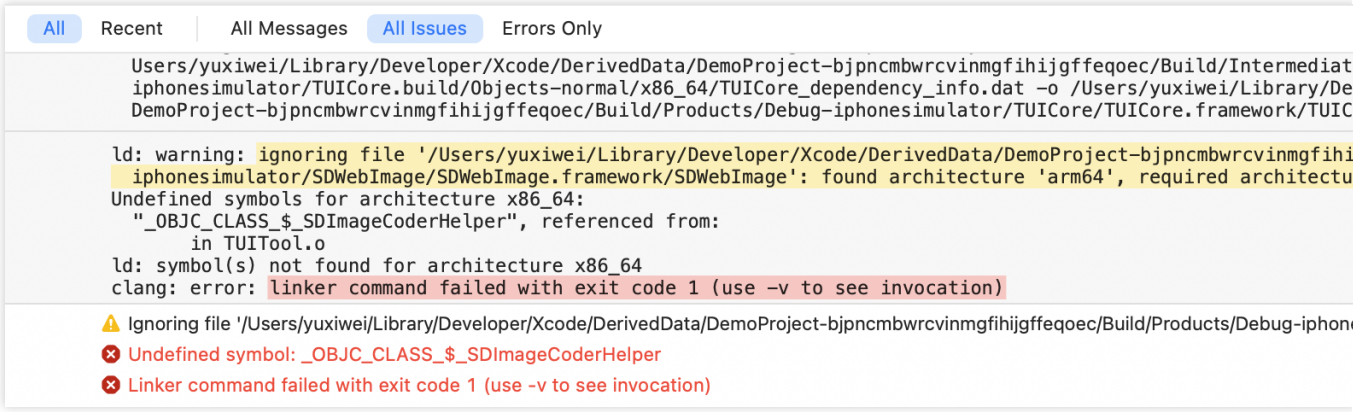


```
# target 'xxxx' do
#  ...
#  pod 'TUICallKit_Swift'
# end

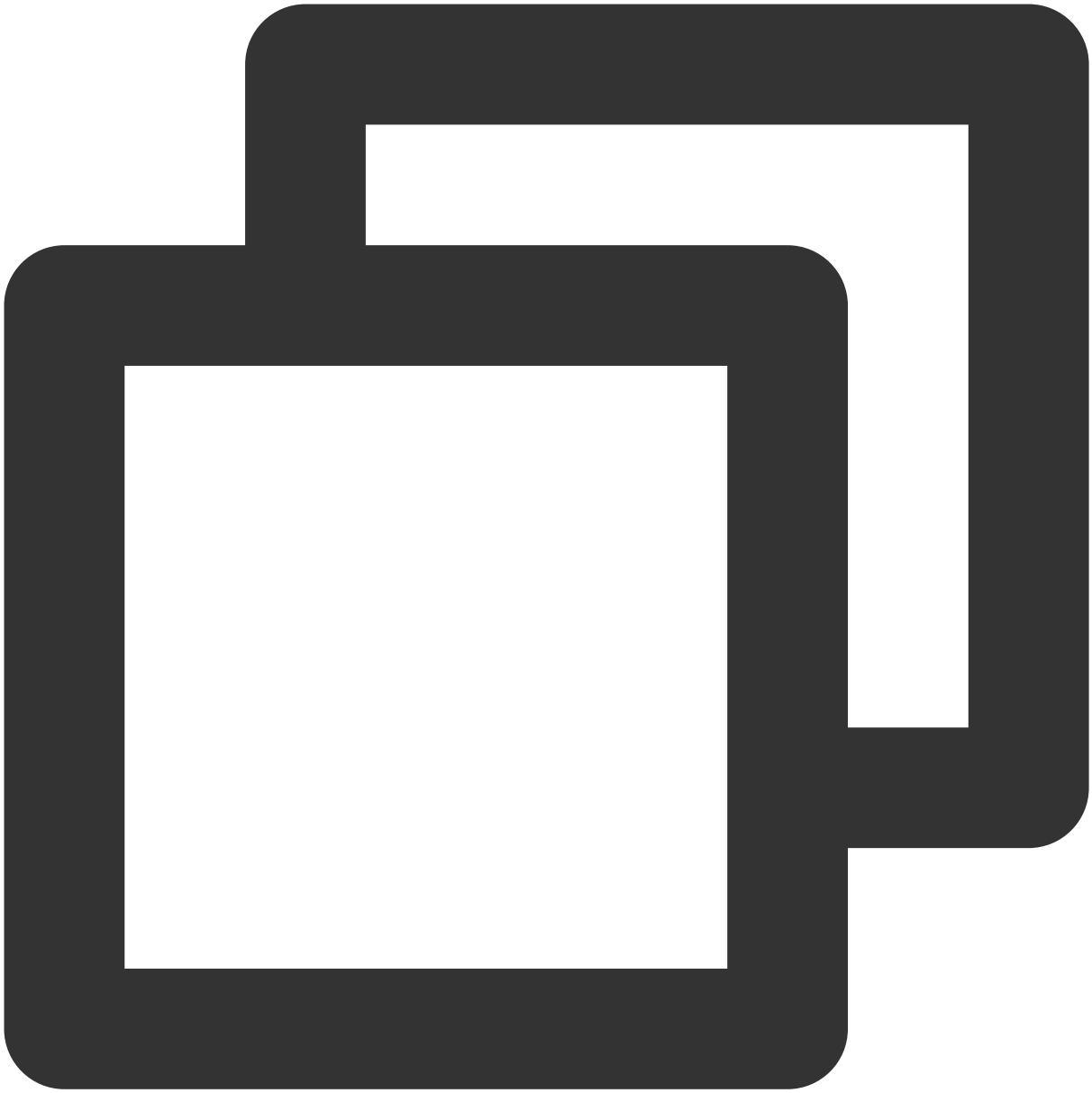
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '13.0'
    end
  end
```

```
end
```

3、如果在 M 系列电脑上运行模拟器，可能会出现 Linker command failed with exit code 1 (use -v to see invocation) 编译报错



可以在 Podfile 添加如下代码



```
# target 'xxxx' do
#  ...
#  pod 'TUICallKit_Swift'
# end

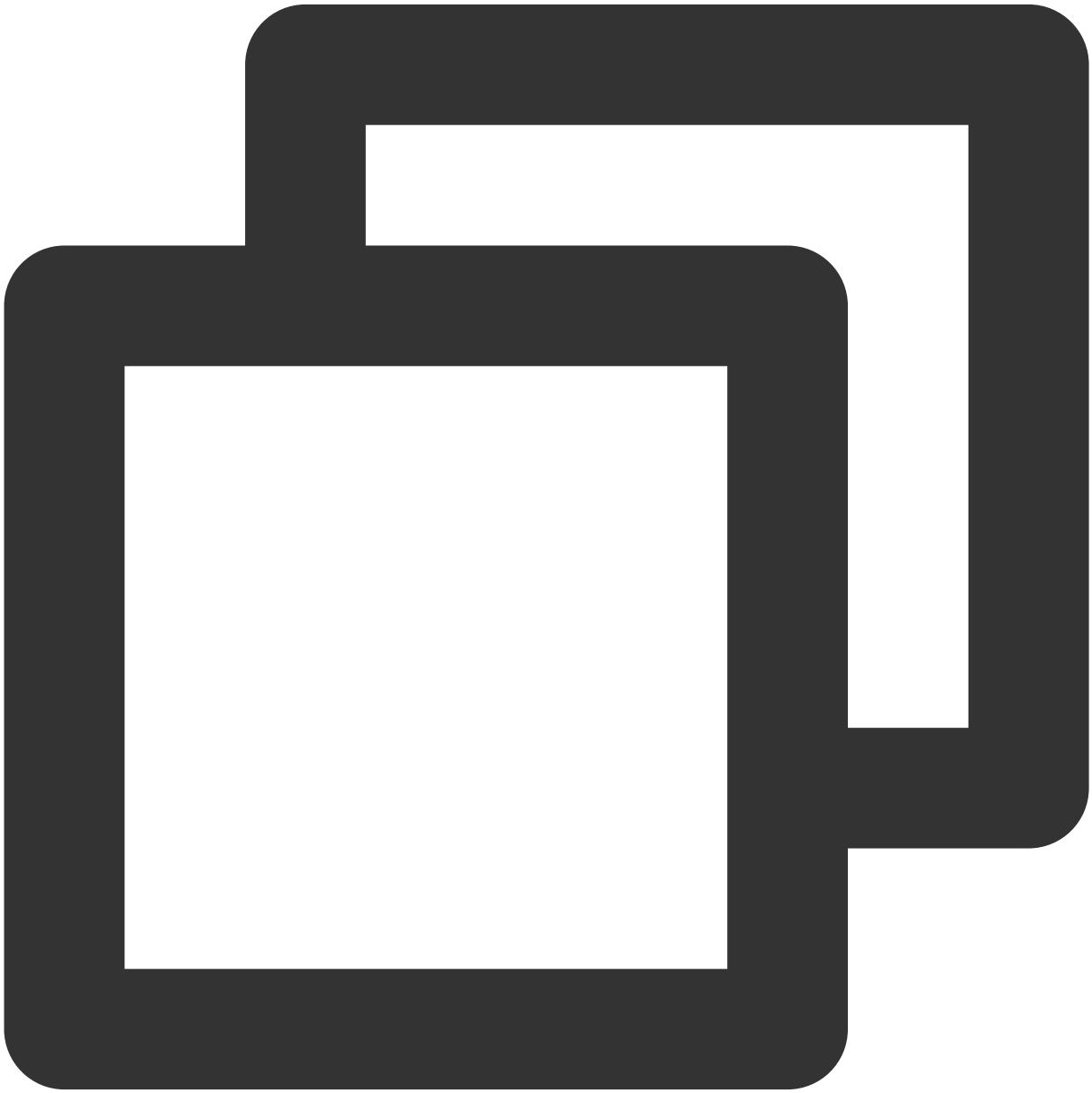
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['EXCLUDED_ARCHS[sdk=iphonesimulator*]'] = "arm64"
    end
  end
```

```
end
```

TUICallKit 和自己集成的音视频库冲突了？

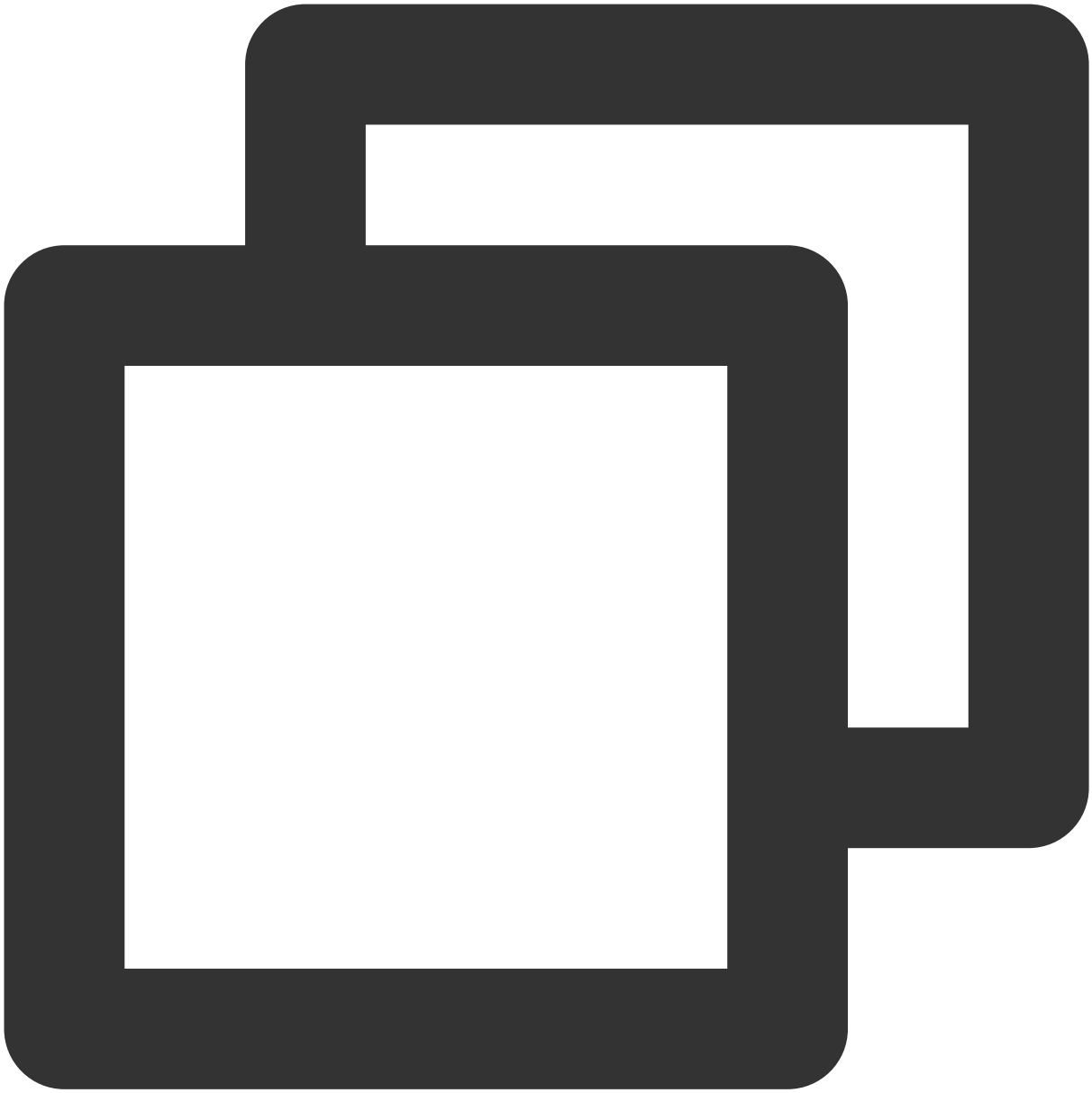
腾讯云的 音视频库 不能同时集成，可能存在符号冲突，可以按照下面的场景处理。

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 'TUICallKit_Swift'
```

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，

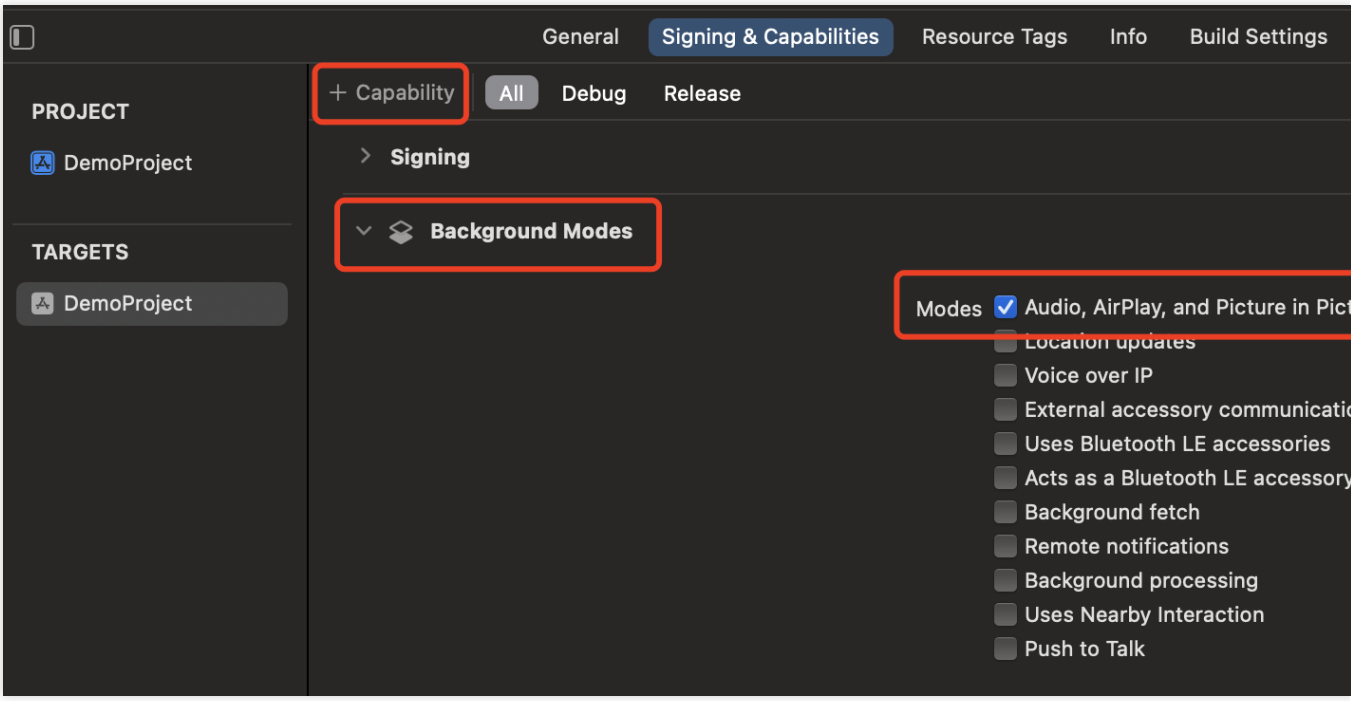


```
pod 'TUICallKit_Swift/Professional'
```

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 TUICallKit_Swift `/Professional` 。

TUICallKit 是否支持后台运行？

支持，如需要进入后台仍然运行相关功能，可选中当前工程项目，在 **Capabilities** 下的 **Background Modes** 模块中勾选 **Audio，AirPlay and Picture in Picture** ，如下图所示：



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： `sandbox的Documents/log`

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：2024-08-15 14:42:54

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，您可以参见 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。

在通话邀请超时时间内，被邀请者如果离线再上线，能否弹出通话界面？

单人通话时，如果在超时时间内上线，会触发来电邀请；群组通话，如果在超时时间内上线，会拉起未处理的20条群消息，如果存在通话邀请，则触发来电邀请。在不同的版本上来电的显示策略有所不同（详见下方：被叫端来电显示策略）。

应用在后台时，不能自动将通话界面拉取到前台

1. TUICallKit 2.3 及以上版本调整了被叫端的来电显示策略，见下方：被叫端来电显示策略。
2. TUICallKit 2.3 版本之前，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。

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。如何开启权限，详见下方：相关权限开启。

3. 如果遇到以下场景拉不起通话界面，原因是：应用的启动堆栈变化，导致 CallKitActivity 界面被遮挡移除了。

场景一：接通后退到后台，点击桌面图标进入应用，原通话界面消失；

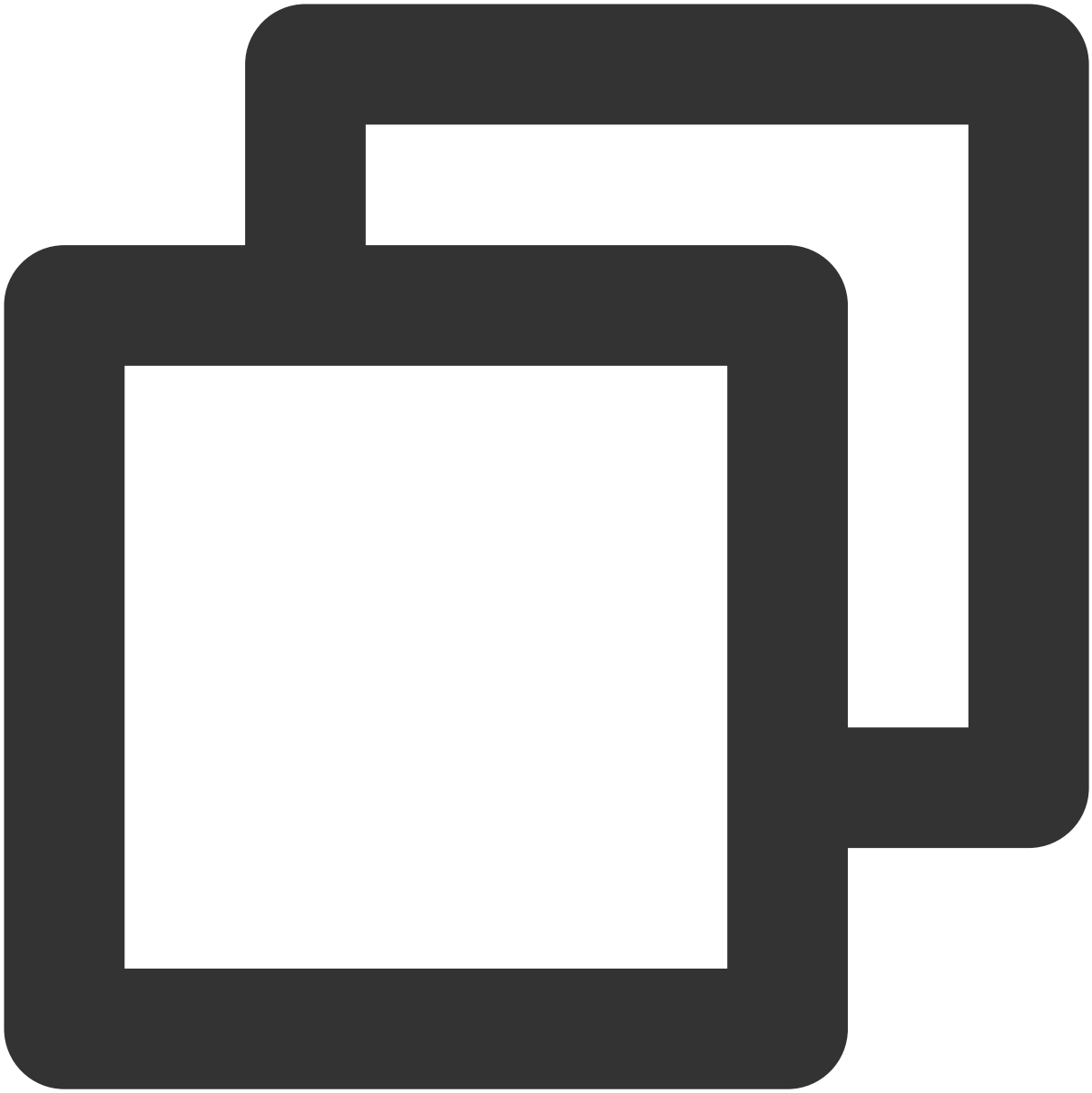
场景二：应用在后台时，开启 banner 的情况下，收到通知 View，不点击 View，点击桌面图标进入应用，拉不起通话界面且通知消失；

场景三：应用在后台时，收到离线推送，不点击推送通知，点击桌面图标进入应用，无法拉起通话界面或通话界面闪了一下；

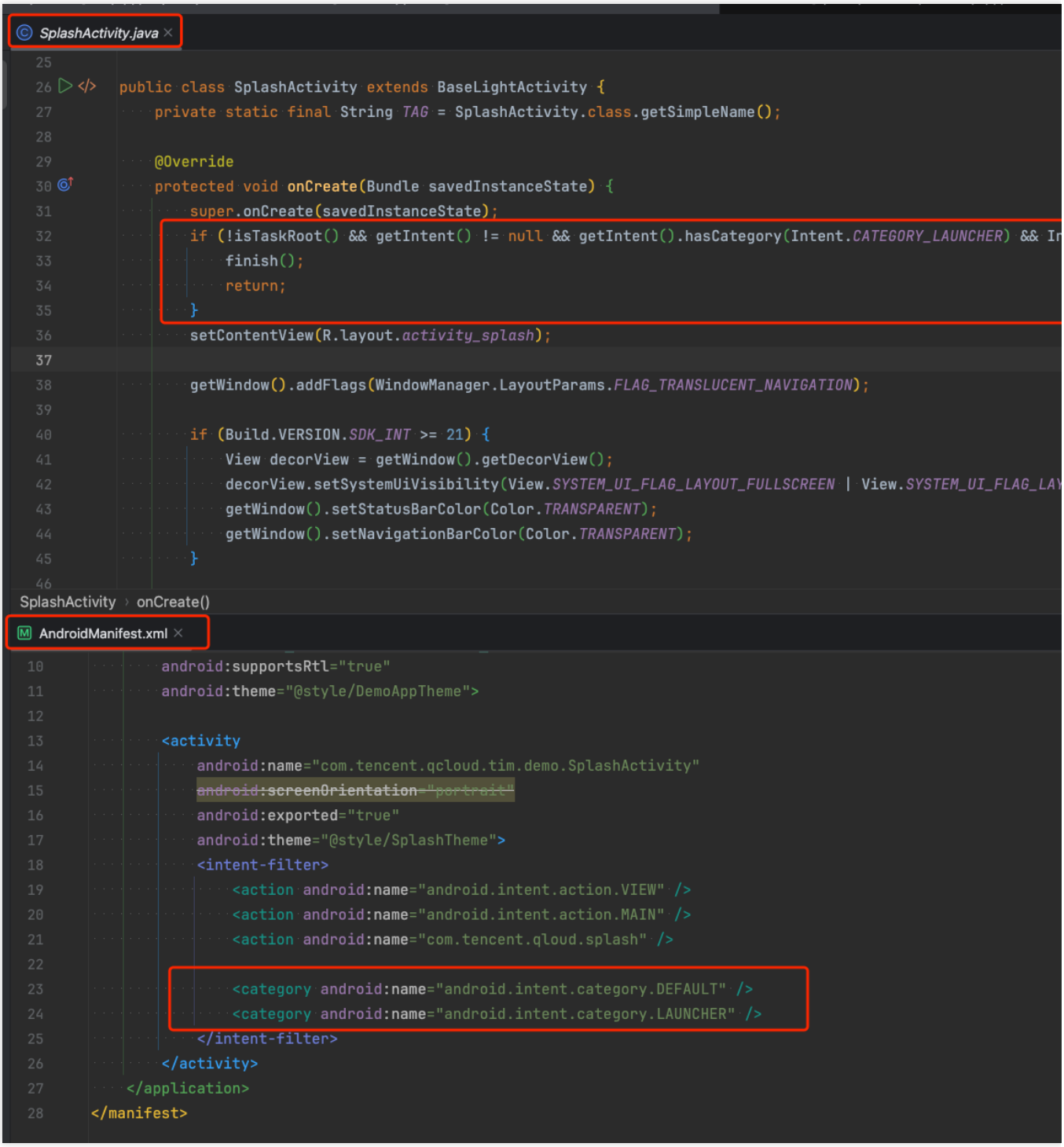
以上几种情况，需要在您自己业务的默认启动的主 Activity 中添加以下代码，每个应用默认启动的 Activity 都不一样，详见 AndroidManifest.xml 配置，以目前大部分应用的启动页 SplashActivity 为例：

添加代码

具体添加位置



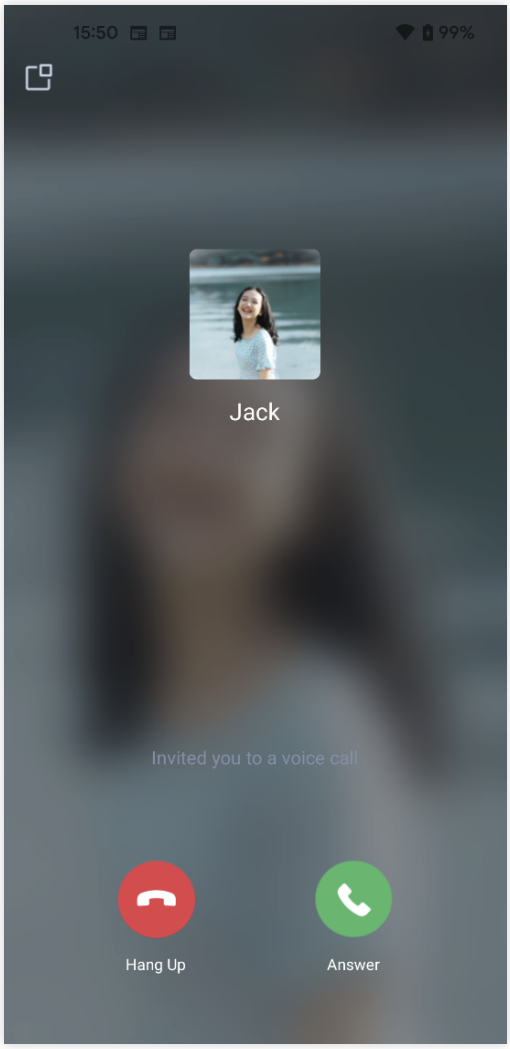
```
if (!isTaskRoot() && getIntent() != null && getIntent().hasCategory(Intent.CATEGORY
        && Intent.ACTION_MAIN.equals(getIntent().getAction())) {
    finish();
    return;
}
```



被叫端来电显示策略

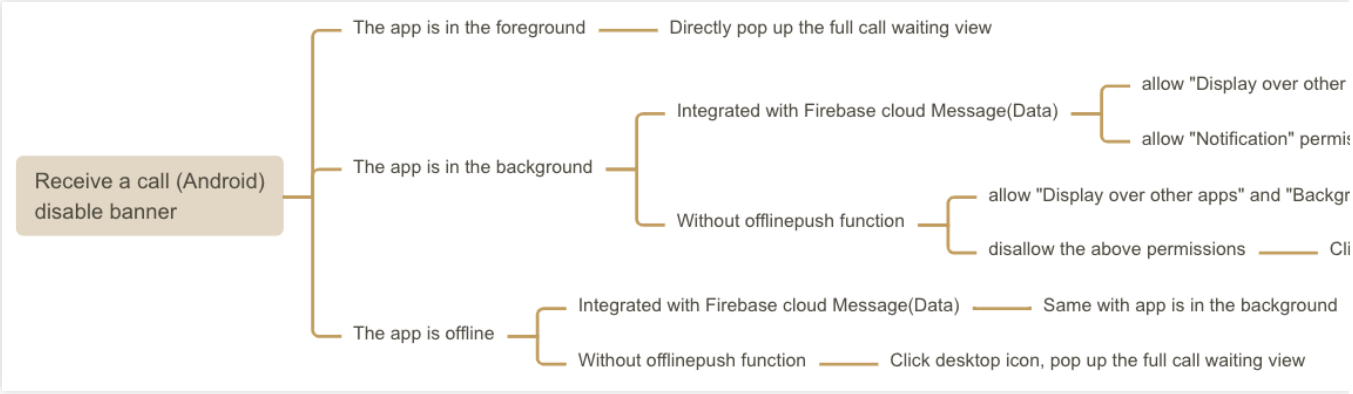
为使 TUICallKit 适应不同的业务需求，增加产品特色，提升用户的使用体验，TUICallKit **2.3 及以上版本**（发布日志），优化收到来电后的通话页面显示弹出策略，详情如下所示：

全屏显示通话等待界面	横幅显示来电请求

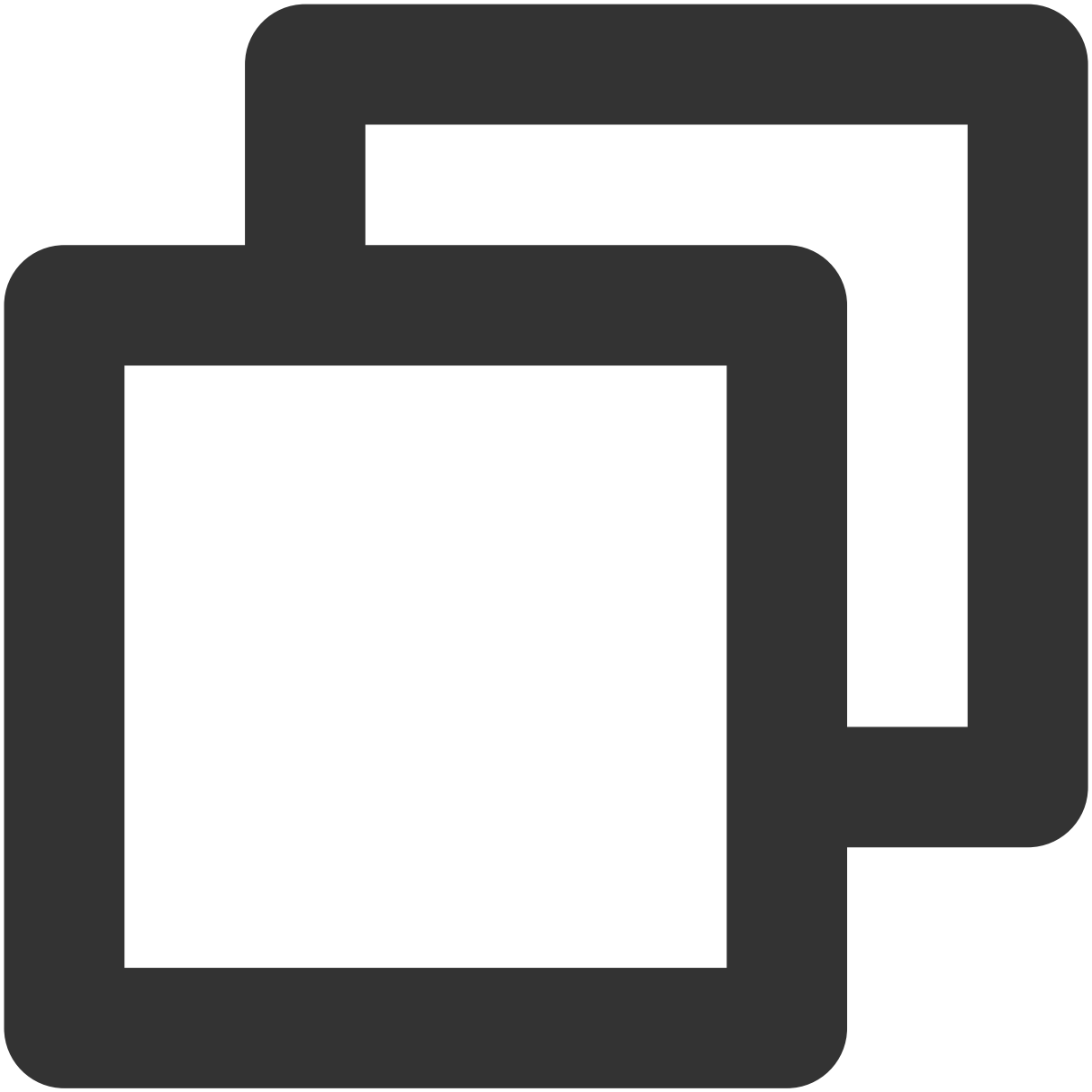




1. 如果您想要被叫端收到邀请时，尽量去拉起全屏通话界面，那么您可以集成 tuicallkit-kt 代码。默认情况下，被叫端的来电显示策略如下：

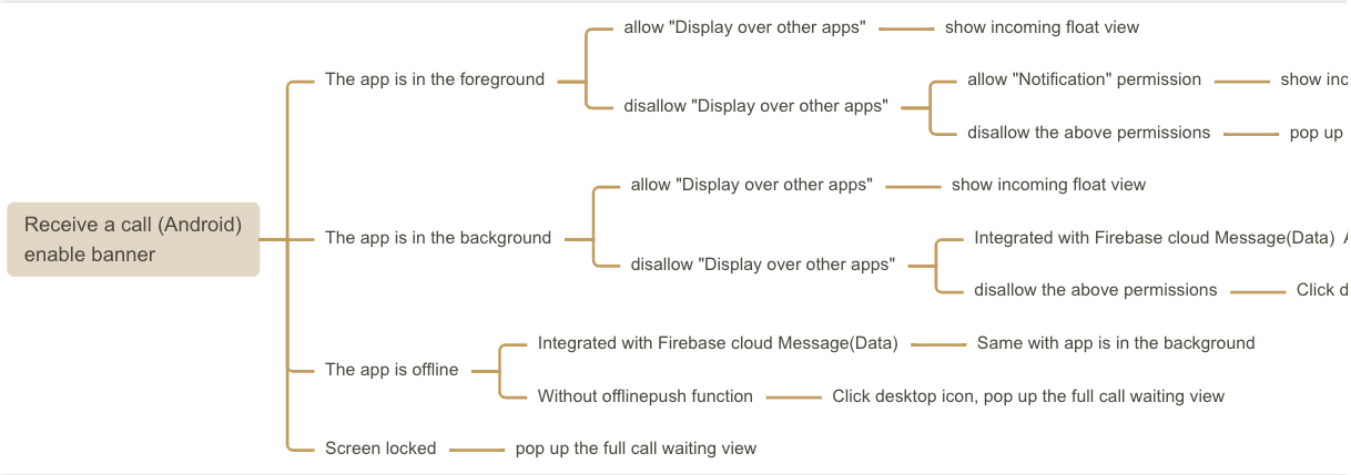


2. 如果您想在被叫端收到通话的时候，先展示一个横幅，然后根据需要拉起全屏通话界面，那么您可以调用以下接口开启该功能。



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

开启后，被叫端的来电策略如下所示，只要开启**悬浮窗权限，**就能尽可能的展示来电横幅。



说明

相关权限如何开启，详见下方：相关权限开启。

如果没有按照上述策略显示来电界面，请过滤 `onCallReceived` 日志，检查是否收到通话邀请。

相关权限开启

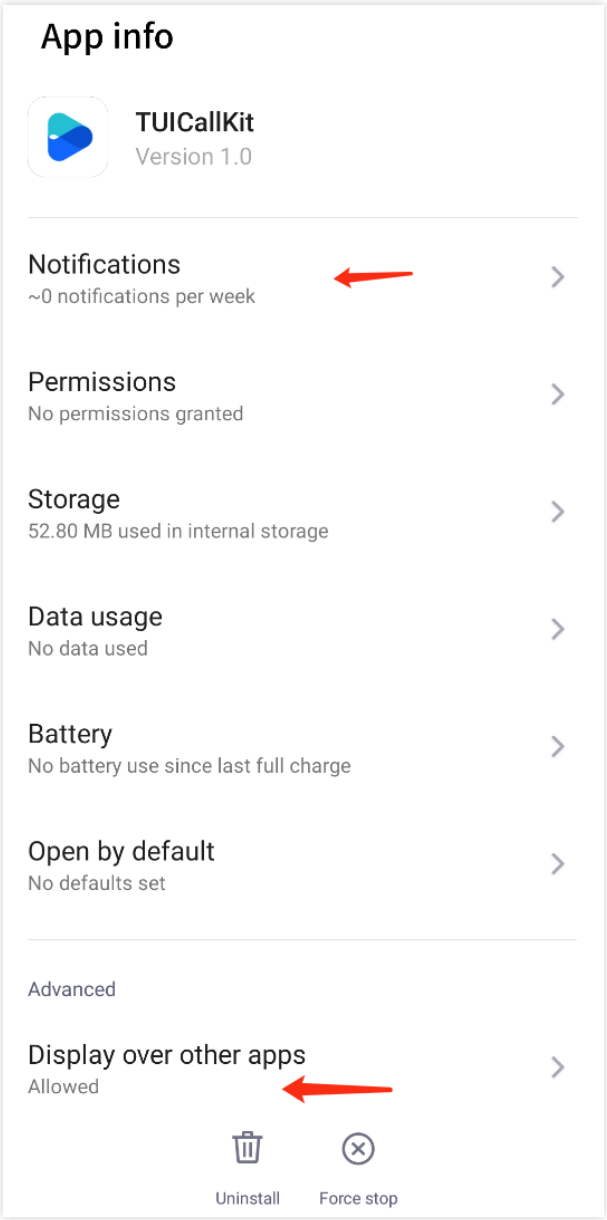
为实现良好的通话体验，建议您在应用中开启“通知”权限、“显示在其他应用上层（悬浮窗）”以及“后台拉起界面”权限，具体方法如下：

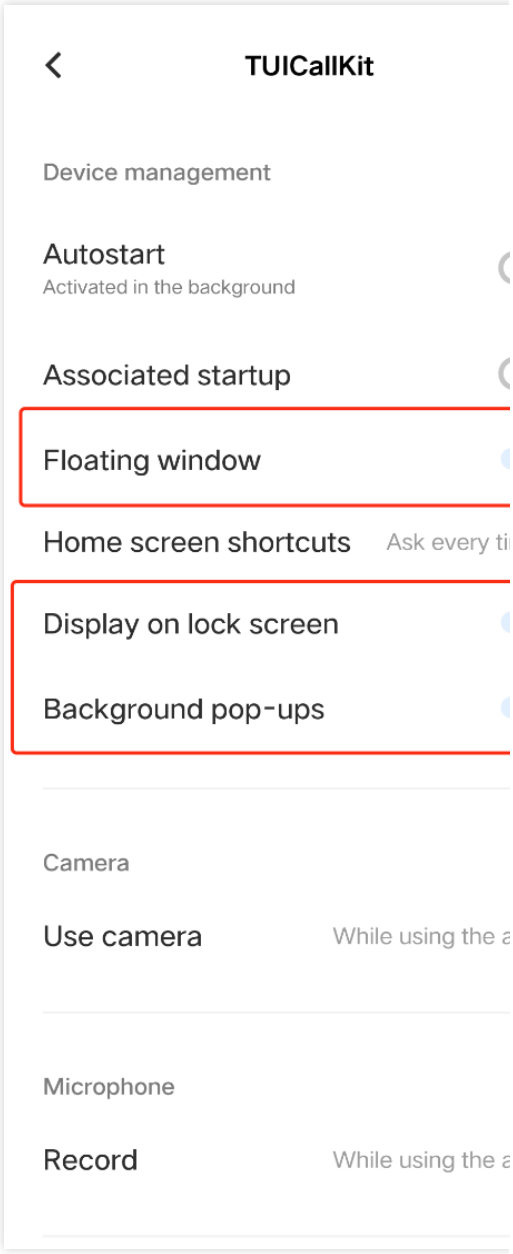
手动开启

代码指引

安装应用后，您可以长按应用图标，选择“应用信息”，然后开启“通知”权限、“显示在其他应用上层”以及“后台拉起界面”权限。或者您可以到 `手机->系统设置->应用管理->应用` 中手动开启上述权限。

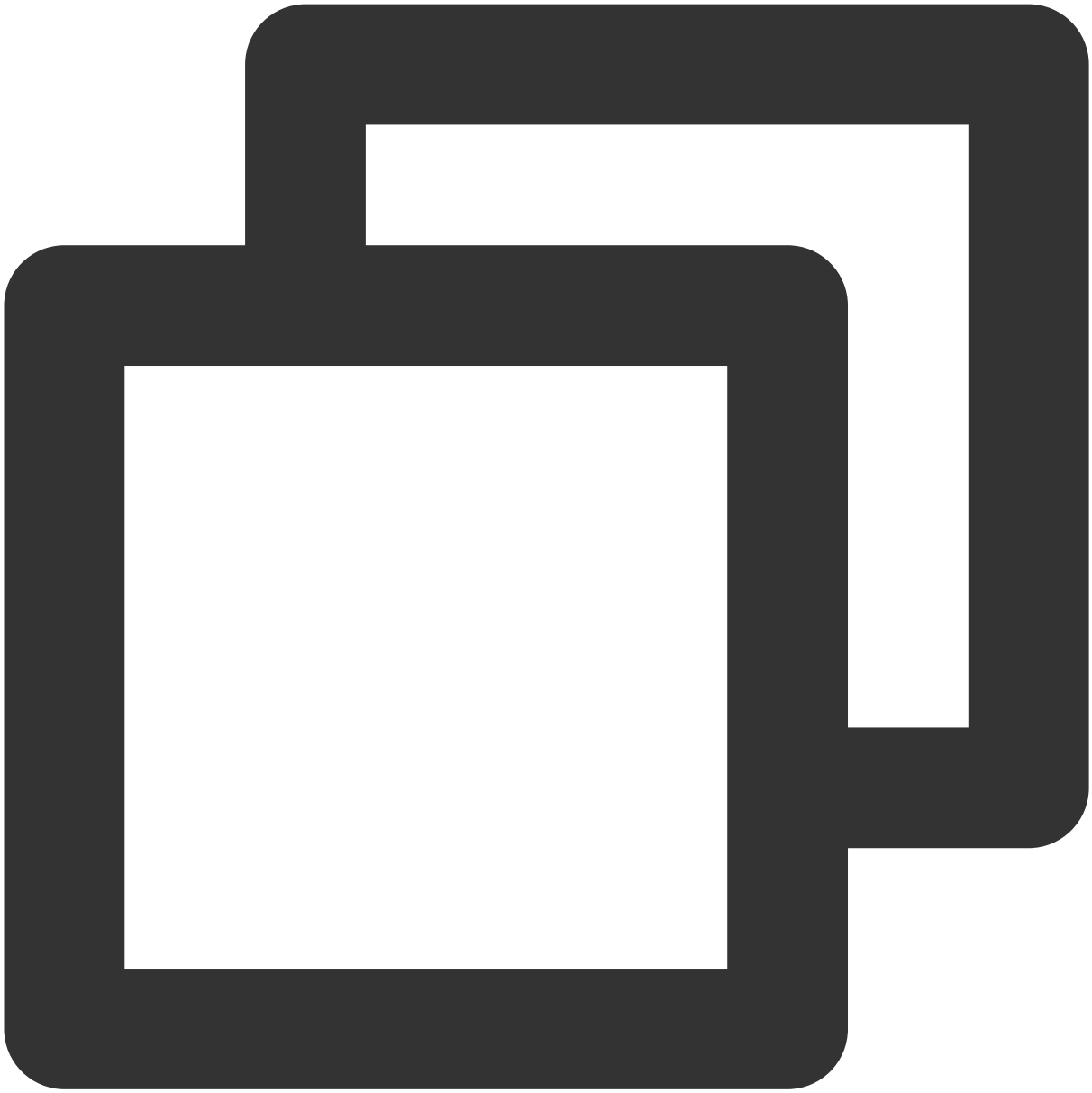
Pixel 4a	VIVO





通知权限：便于展示推送通知：请参考通知运行时权限 和 请求运行时权限 根据业务需求自行实现。
悬浮窗权限：用于展示自定义的来电横幅，以及通话悬浮窗。
后台拉起界面权限：用于当应用在后台时拉起界面（例如：vivo手机）。

版权所有：腾讯云计算（北京）有限责任公司



```
fun requestPermission(context: Context?) {
    //In TUICallKit,Please open both OverlayWindows and Background pop-ups permissi
    PermissionRequester.newInstance(
```

Web

最近更新时间：2024-06-27 16:54:10

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？

请参见购买链接 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。

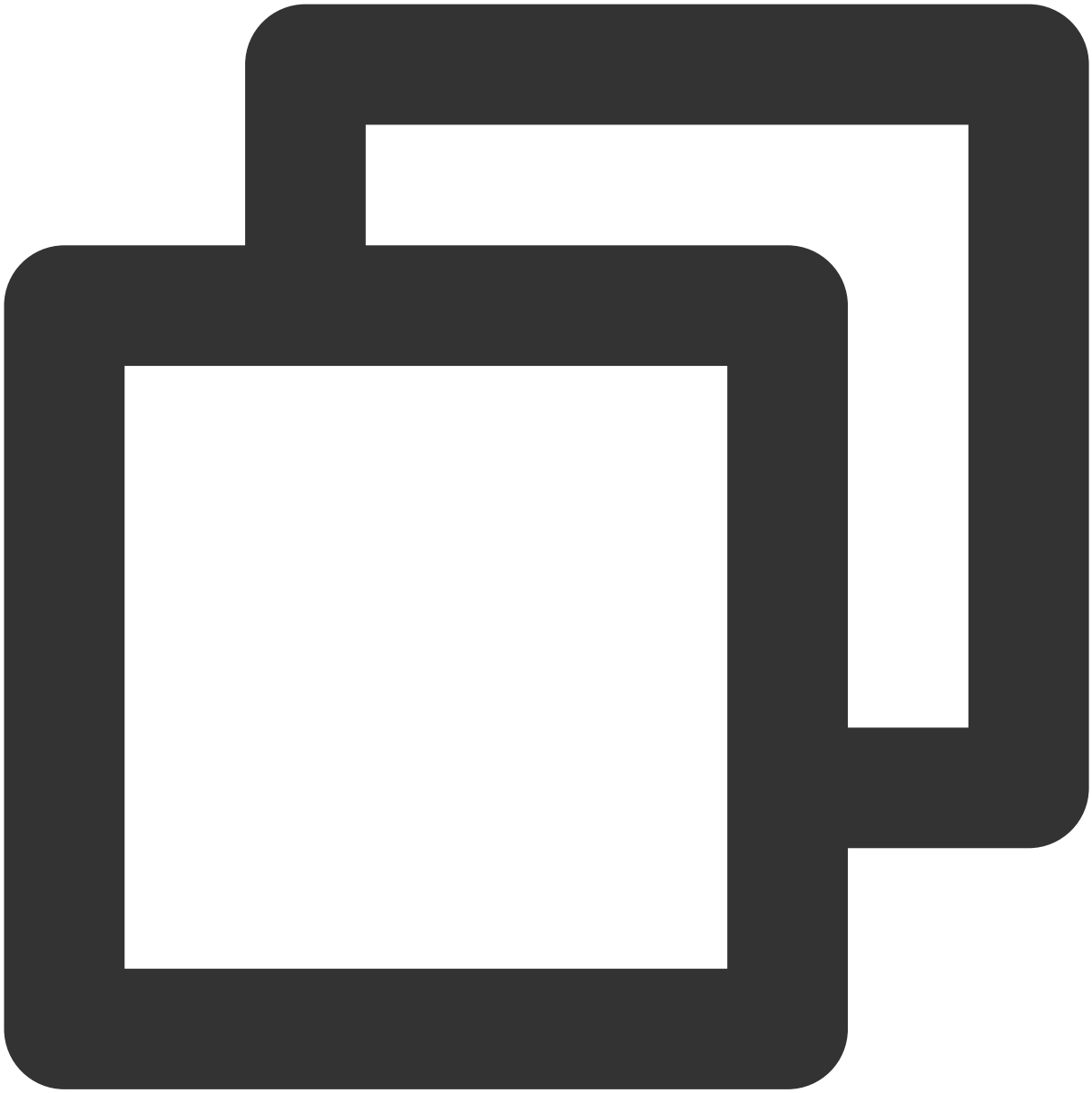
方式一：控制台获取，参见 获取临时 userSig。

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：



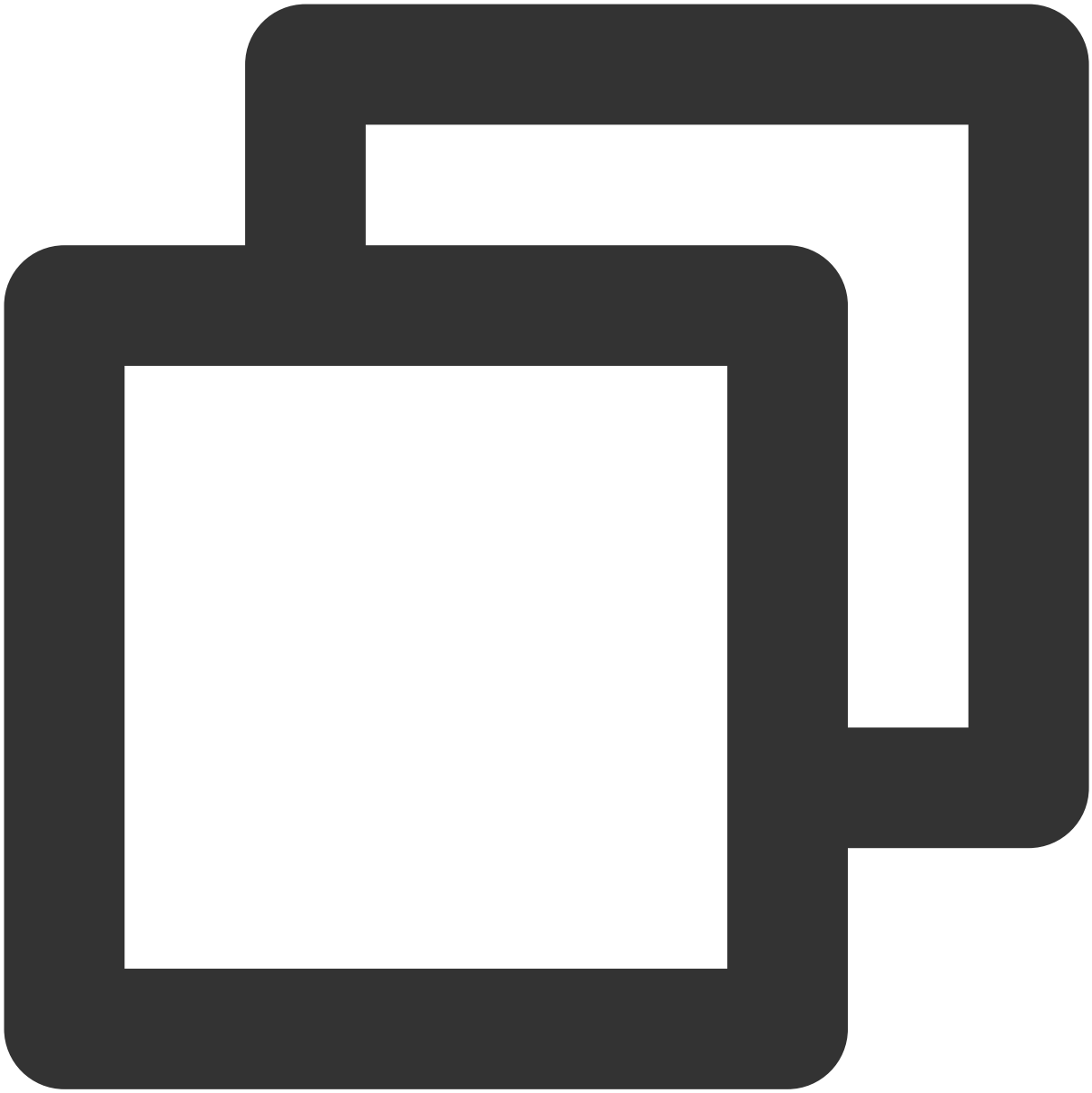
```
import { genTestUserSig } from "./debug/GenerateTestUserSig-es.js";
const { userSig } = genTestUserSig({ userID: "Alice", SDKAppID: 0, SecretKey: "YOUT
```

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 Chat from "@tencentcloud/chat";   // npm i @tencentcloud/chat

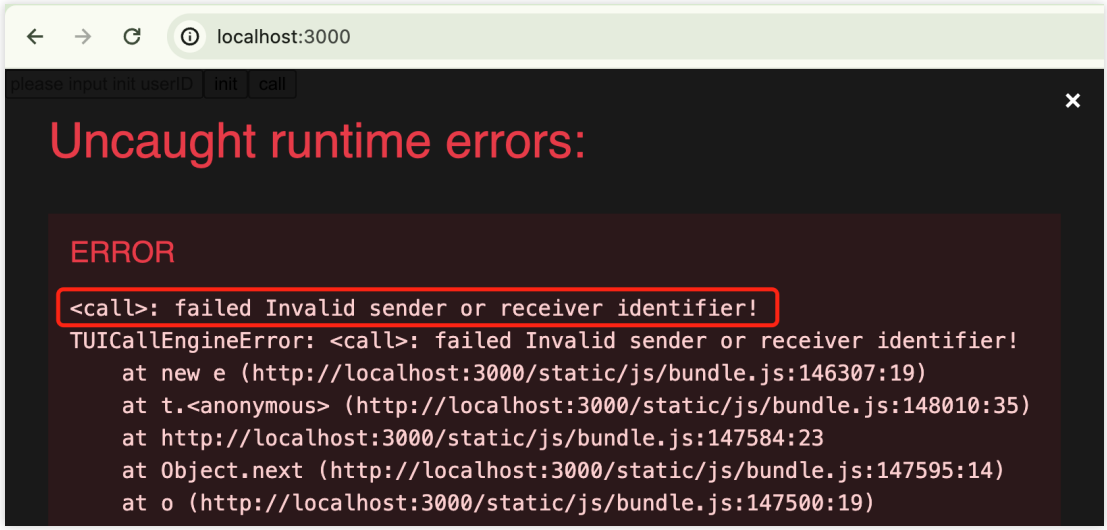
const userIDList: string[] = ['user1', 'user2'];
async function createGroupID() {
  const chat = Chat.create({ SDKAppID });
  const memberList: any[] = userIDList.map(userId => ({ userID: userId }));
  const res = await chat.createGroup({
    type: Chat.TYPES.GRP_PUBLIC,
    name: 'WebSDK',
    memberList
  });
```

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

如何创建 userID？

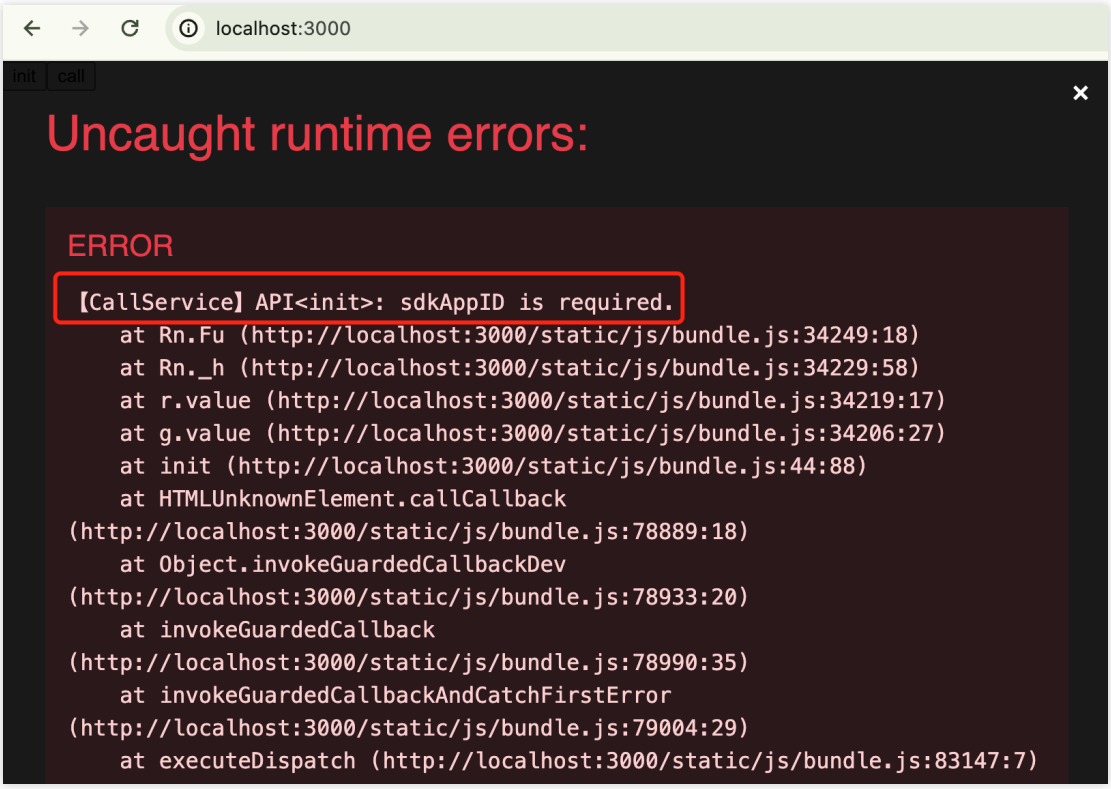
用户的唯一标识符，由您定义，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
通过 userID 与 userSig 登录过一次，会默认创建该用户。
通过 Tencent RTC 控制台进行创建和获取。

Error <call>: failed Invalid sender or receiver identifier？



如果出现这个错误，是因为您 call 的 userID 不存在，需要确保该 userID 已经登录过一次。详见如何创建 userID。

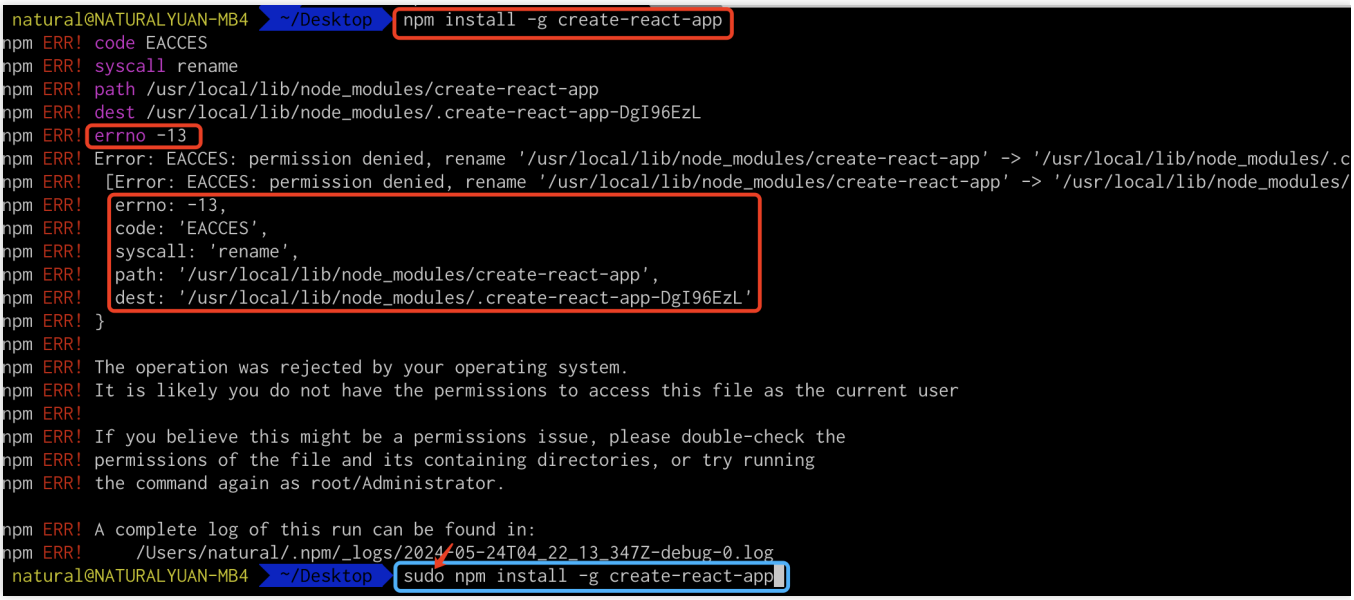
Error 【CallService】API<init>: sdkAppID is required？



如果出现这个错误，是因为你在 TUICallKitServer.init/ `GenerateTestUserSig.genTestUserSig` 时没有填写 SDKAppID 这个信息。

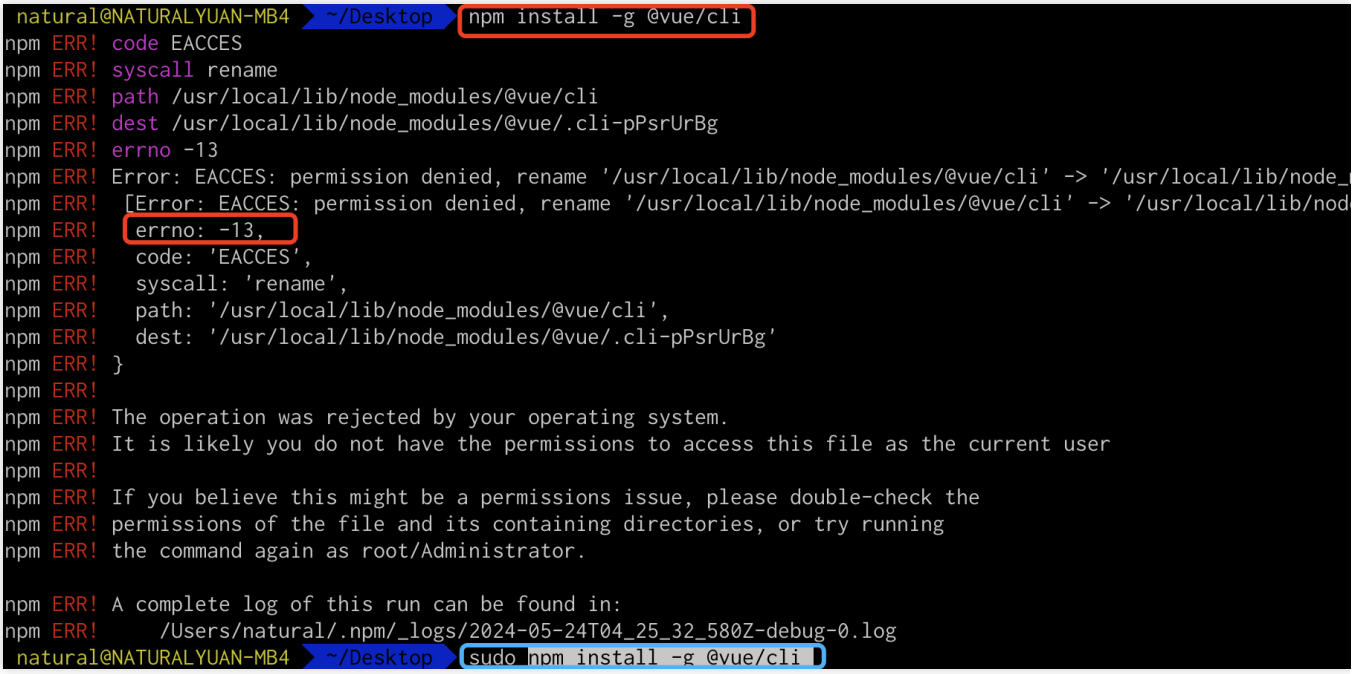
请从Tencent RTC 控制台获取并填写。

npm install -g create-react-app 报错 errno -13？



如果出现这个错误，是因为当前用户没有权限在全局安装脚手架，请使用 `sudo npm install -g create-react-app` 。

npm install -g @vue/cli 包错 errno -13？



如果出现这个错误，是因为当前用户没有权限在全局安装脚手架，请使用 `sudo npm install -g @vue/cli` 。

全平台

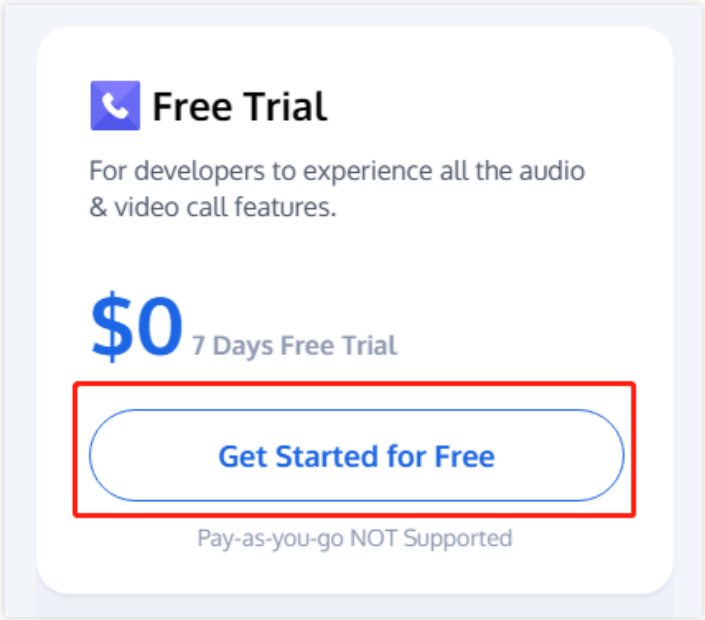
最近更新时间：2024-07-23 18:11:32

音视频通话功能出现了未开通的错误提示？

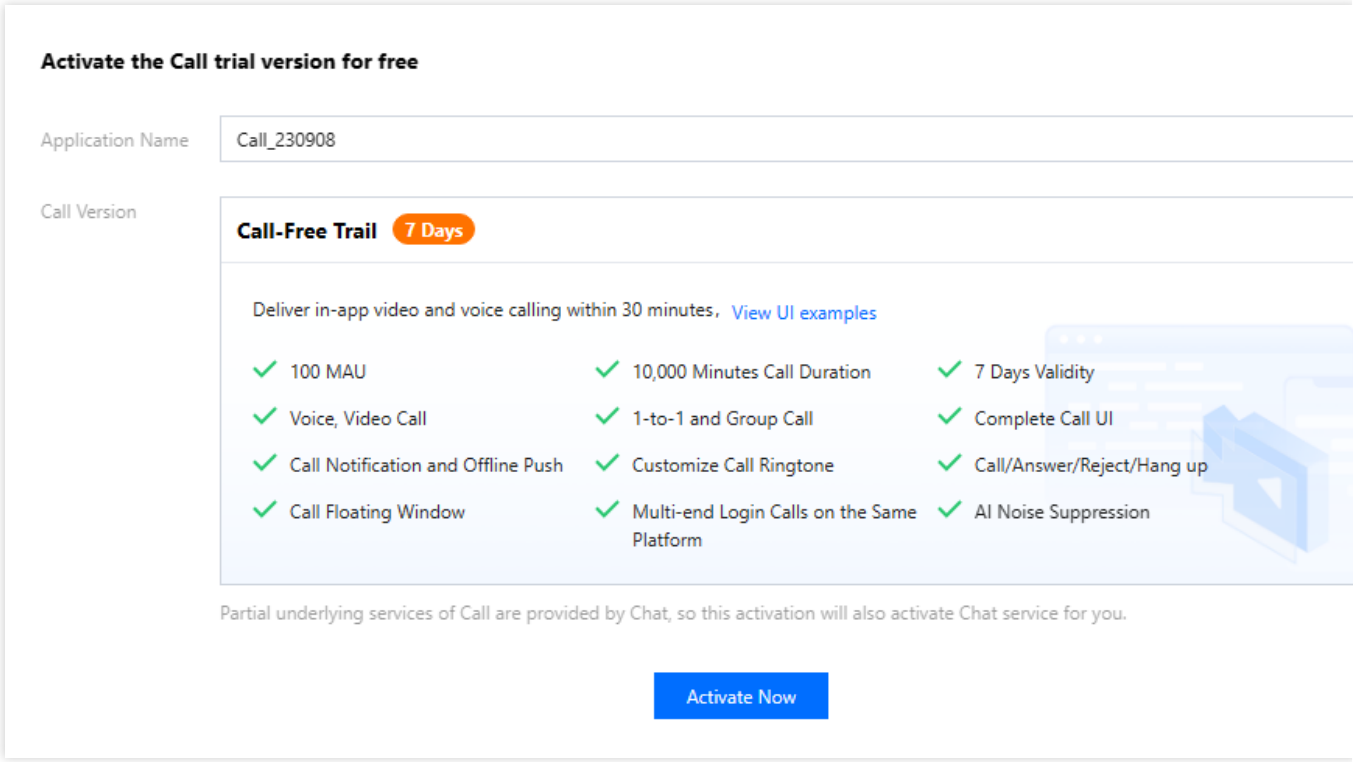
您可以选择体验版（7 天试用）或者购买正式版。

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。

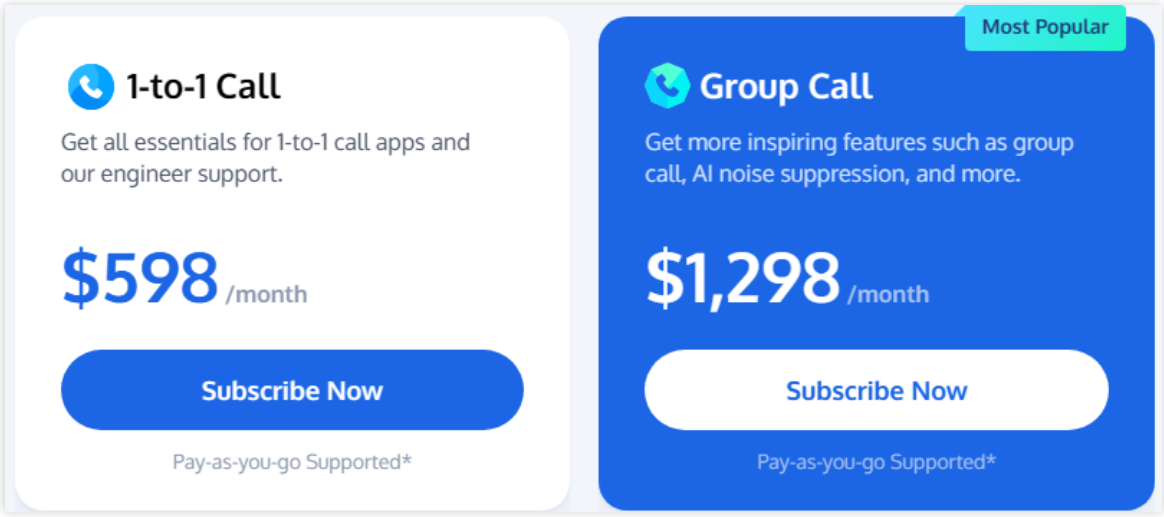


注意：

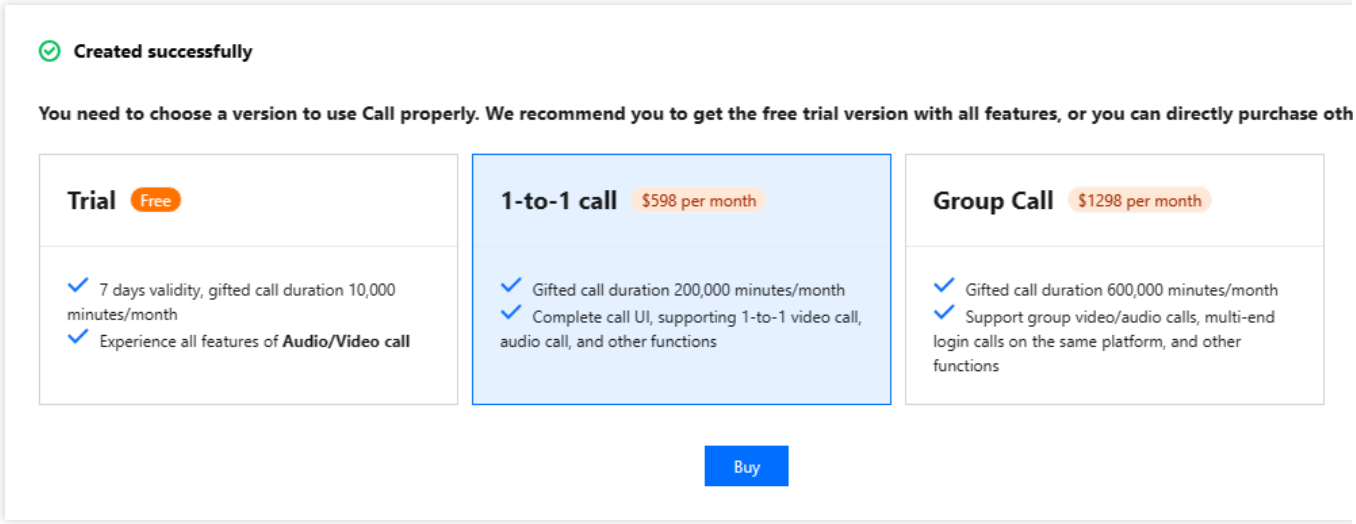
Call的部分底层服务是由Chat提供的，因此此激活也将为您激活Chat服务。

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 Chat，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 Chat SDK，只使用 TRTC？

不可以，TUIKit 全系组件都使用了腾讯云 Chat 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 Chat 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。

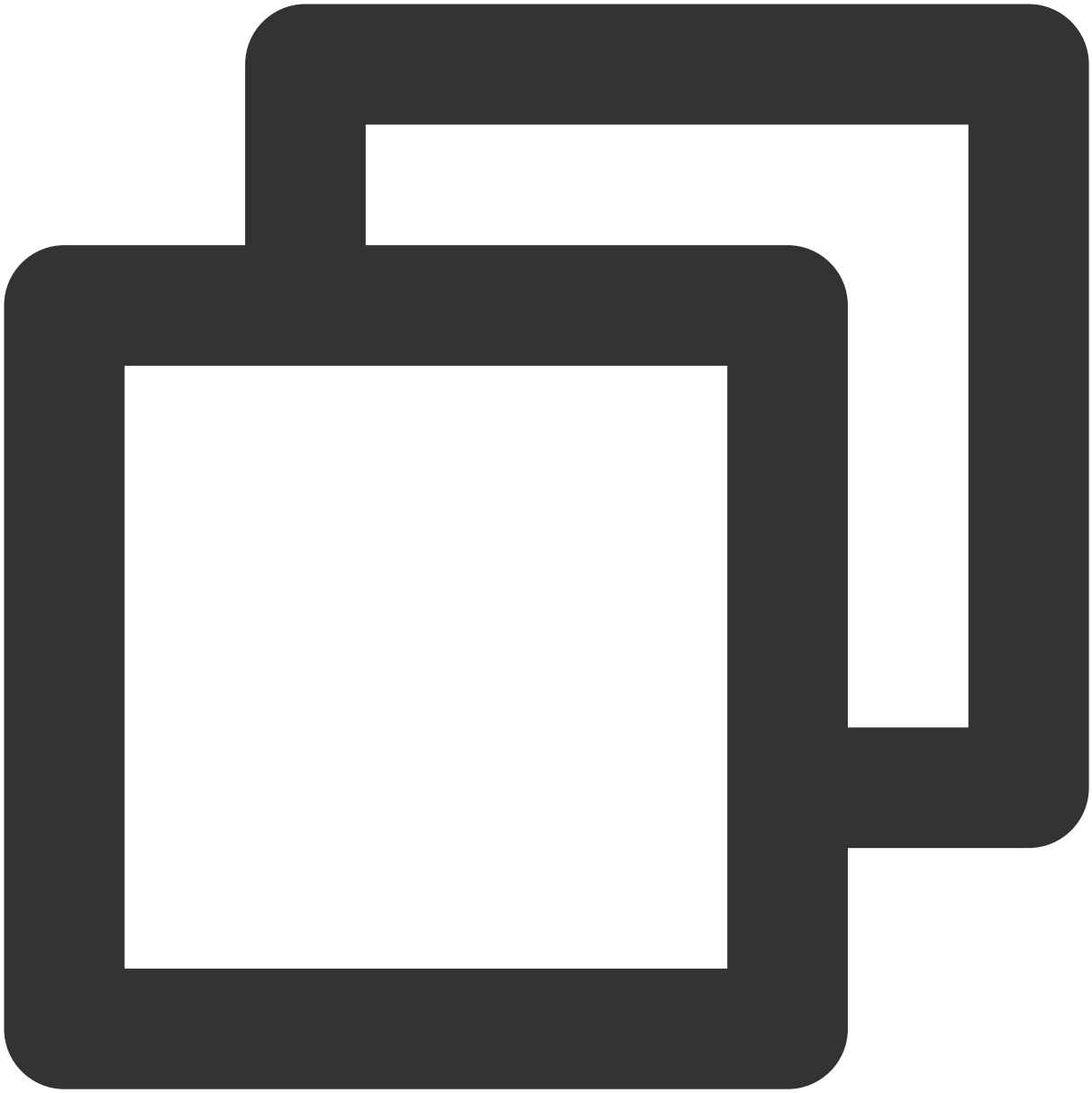
版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-08-15 14:42:54

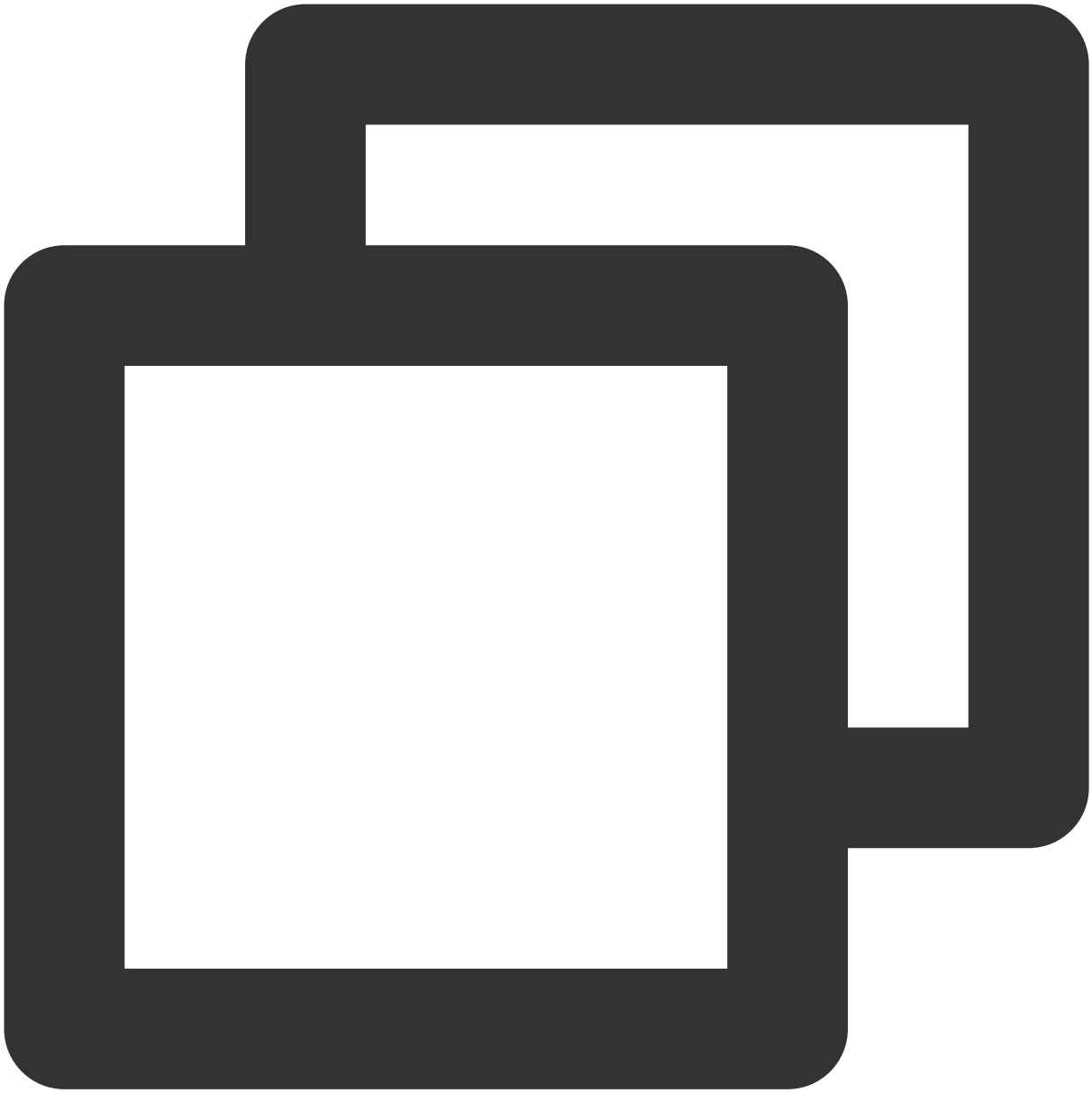
1、同时集成 tencent_calls_uikit 和 tencent_trtc_cloud，或同时集成 tencent_calls_uikit 和 live_flutter_plugin，出现符号冲突报错，怎么解决？

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：

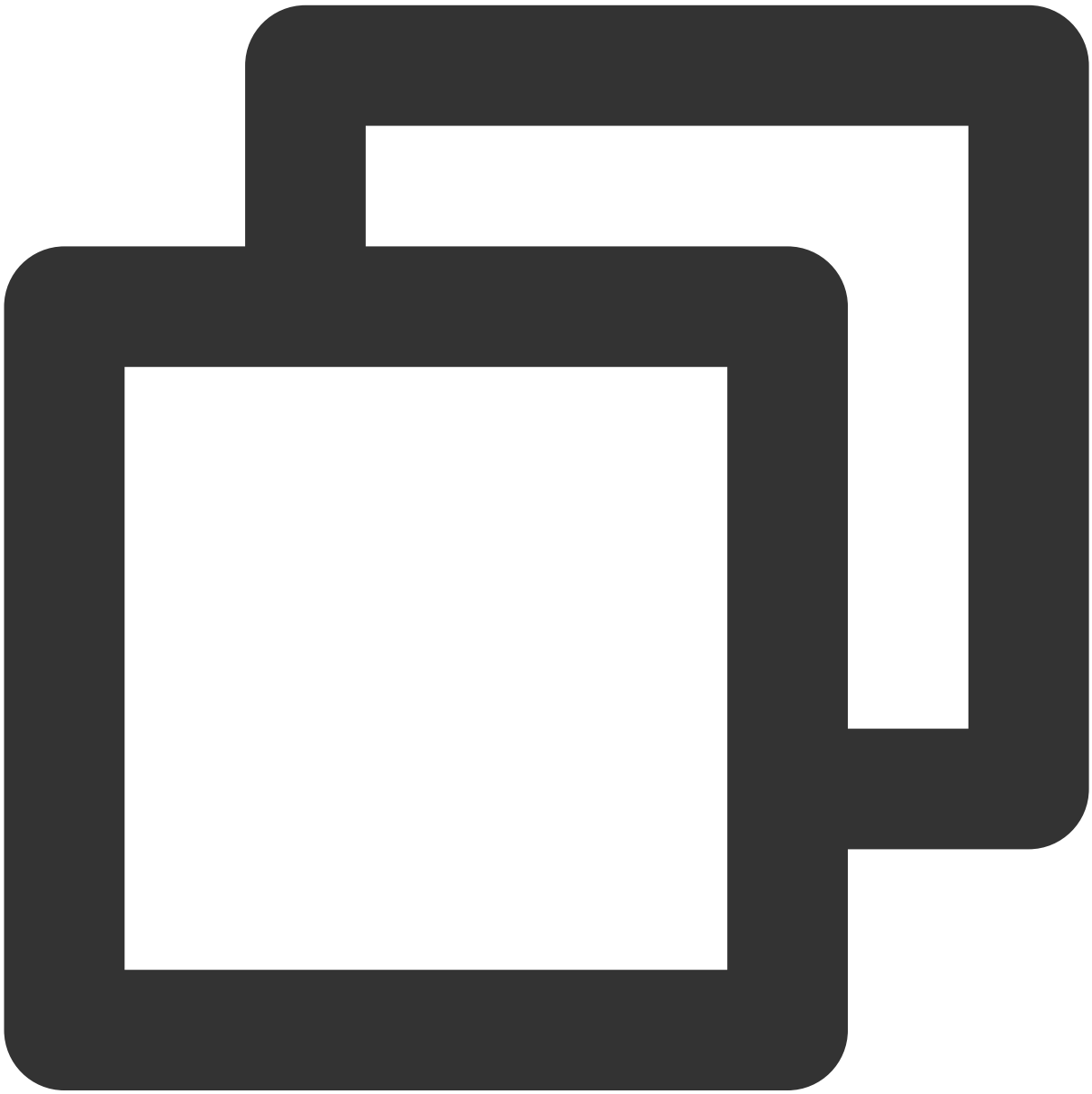


```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xc
```

这个问题原因是您使用的 tencent_calls_uikit 和 tencent_trtc_cloud 分别依赖于 TRTC Android SDK 的专业版和精简版，这个问题我们已经在最新的版本中解决，只需要您将 tencent_calls_uikit 和 tencent_trtc_cloud。

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：

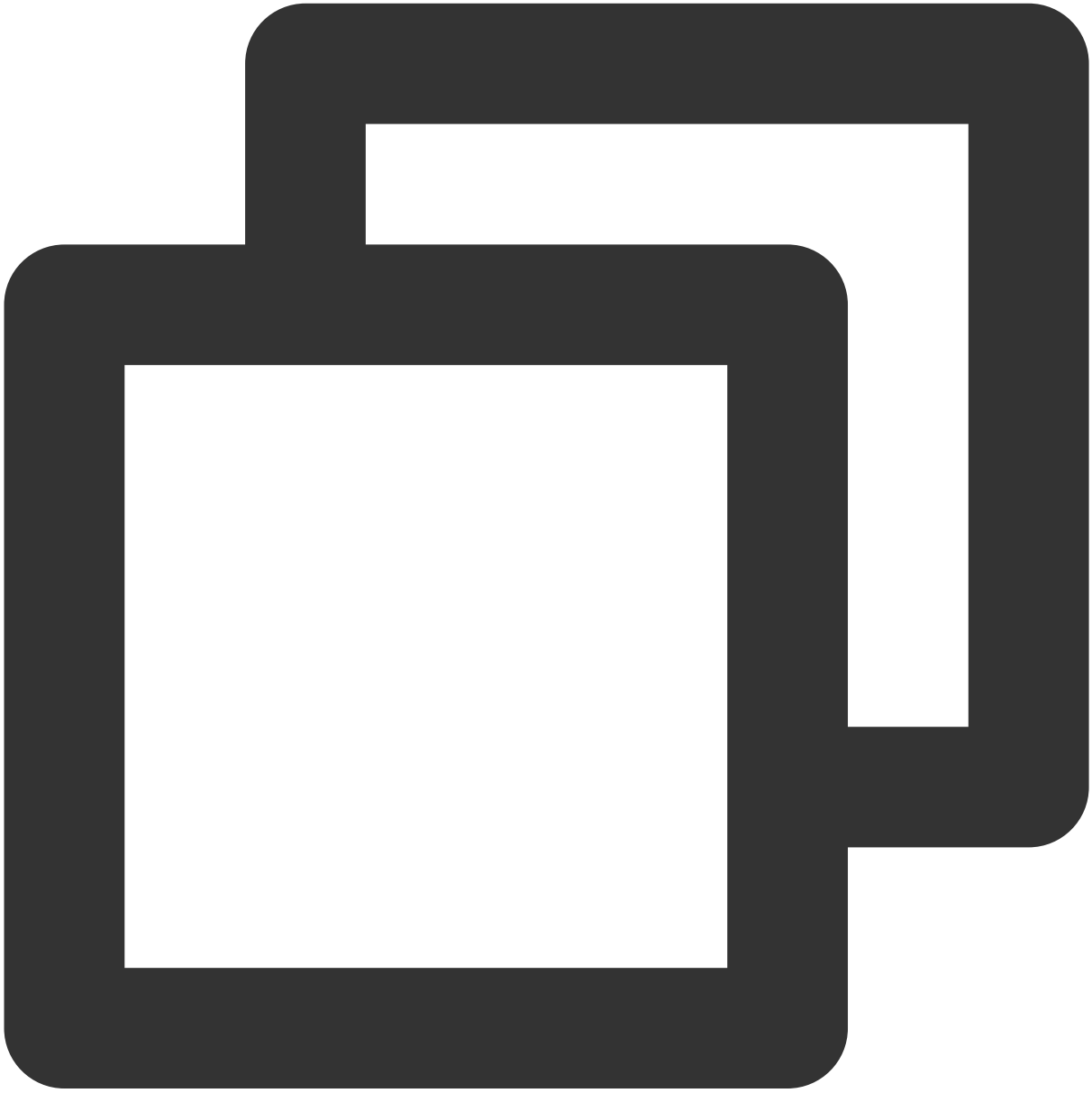


```
-keep class com.tencent.** { *; }
```

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：

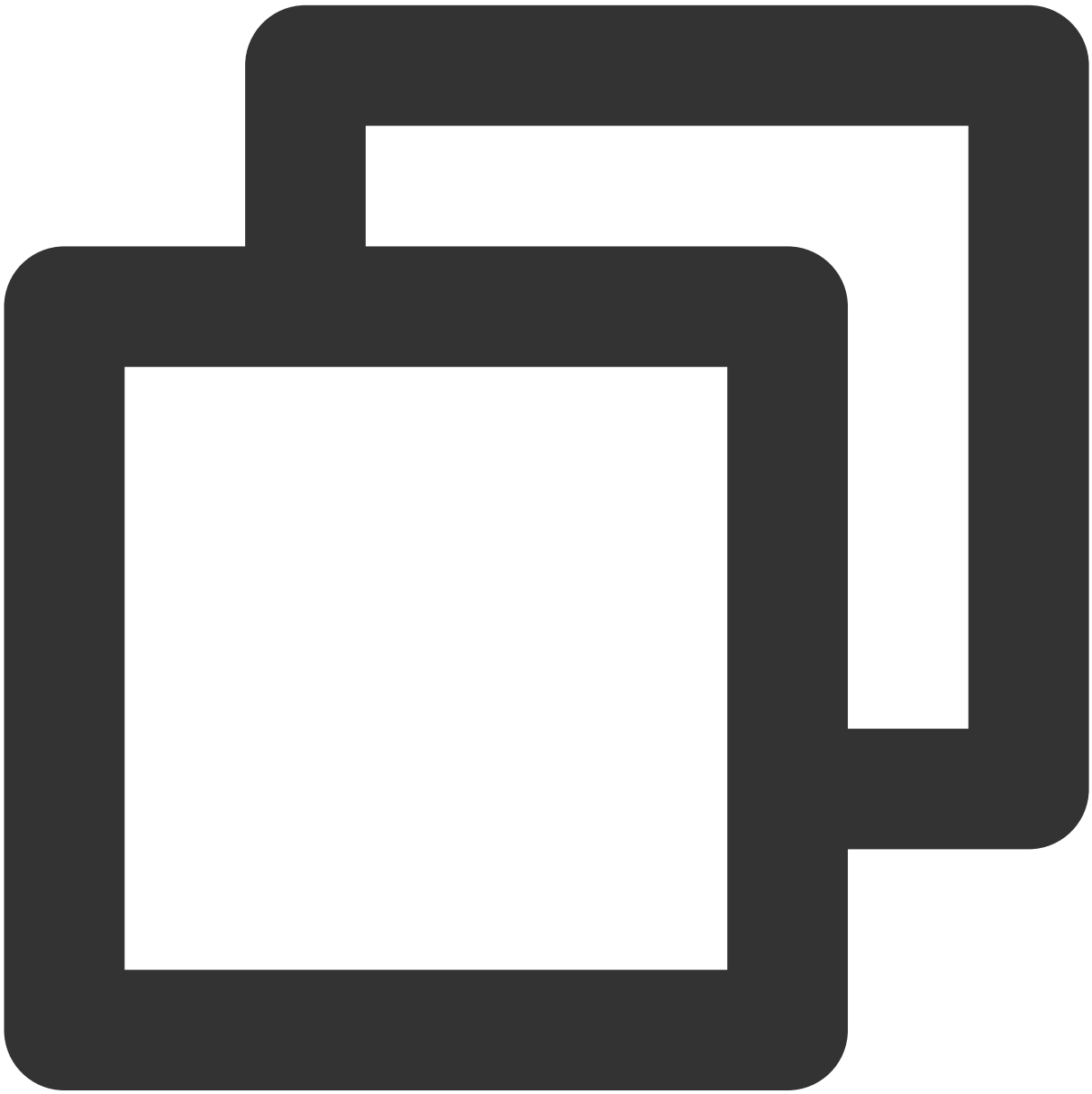
1. 将 `navigatorObservers` 添加到 `MateriaApp` 。 目的是在收到呼叫邀请时导航到 TUICallKit 页面。示例代码如下：



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

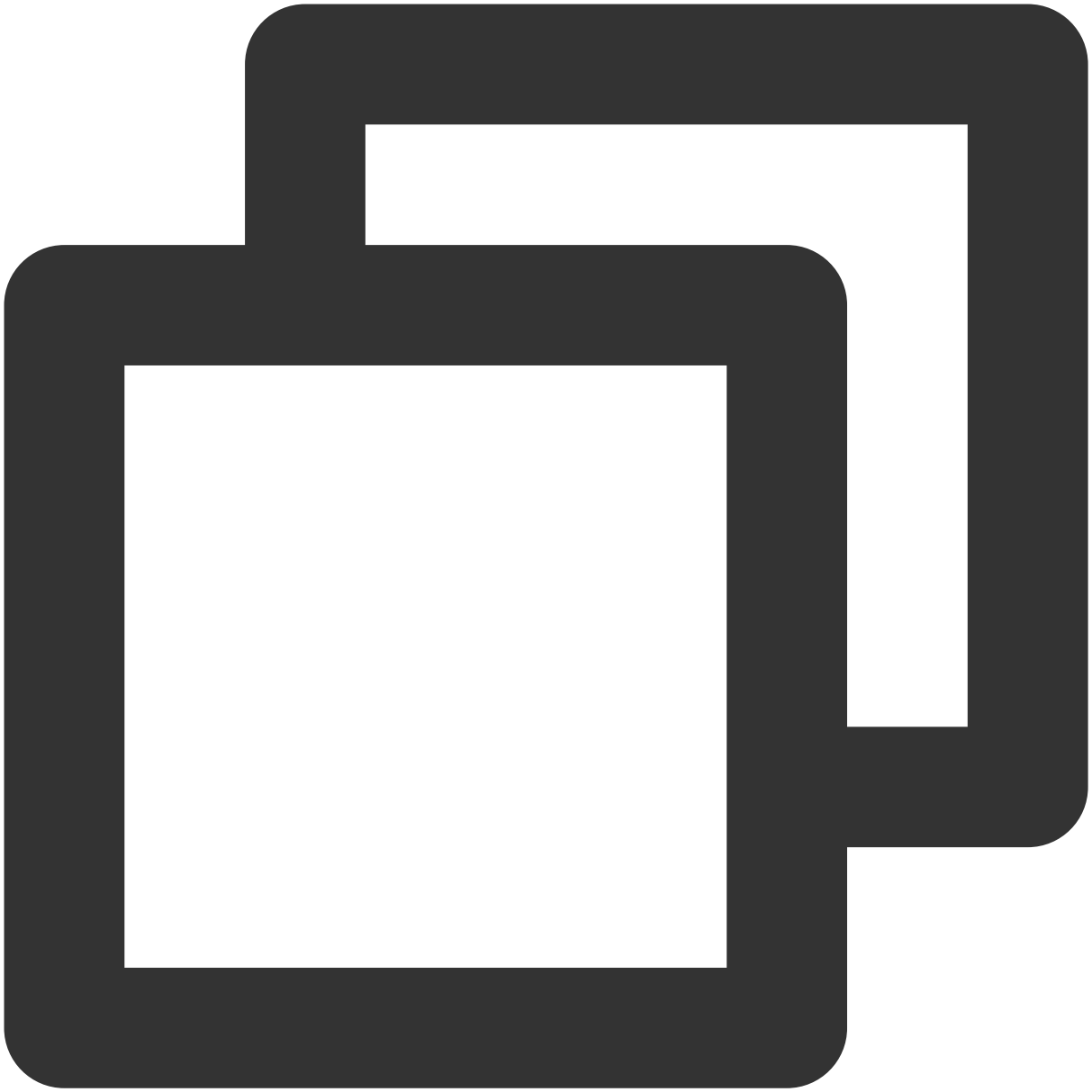
MaterialApp (
      navigatorObservers：[TUICallKit.navigatorObserver],
      ...
)
```

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。

版权所有：腾讯云计算（北京）有限责任公司