即时通信 IM

Demo 专区

产品文档

腾讯云

文档目录

Demo 专区

体验 Demo

最近更新时间：2024-03-12 15:44:00

Demo 体验

Demo 功能

应用内聊天
会话管理
群组管理

用户资料与关系链
离线推送
本地搜索

Demo 及解决方案下载

快速跑通

概述

最近更新时间：2024-05-13 16:35:22

请选择您的开发平台，快速运行腾讯云即时通信 IM Demo：

Android

iOS

Web(React)

Electron

uni-app

Web & H5(Vue)

Unity

UE

React Native

Flutter

用 Flutter 快速集成至您现有原生应用

Android

最近更新时间：2024-05-13 16:39:13

操作步骤

步骤1：创建应用

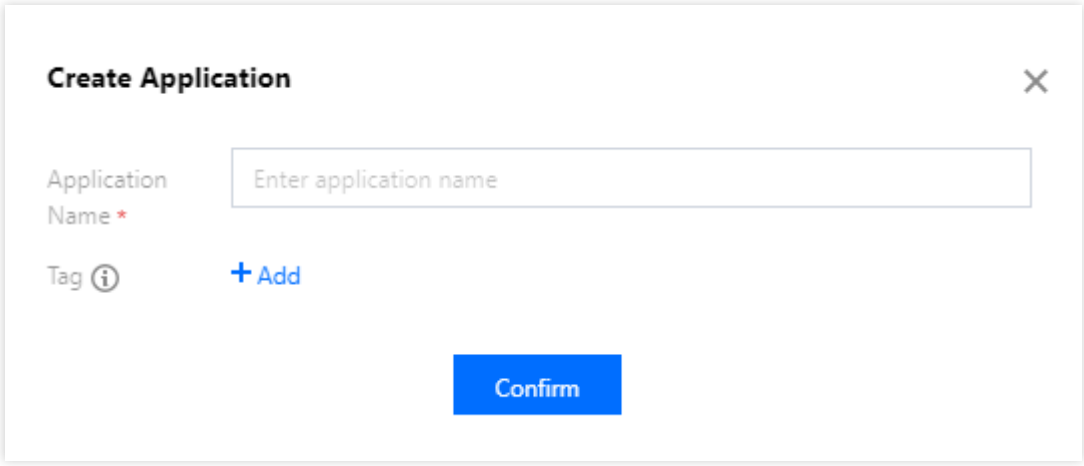
1. 登录 即时通信 IM 控制台。

说明

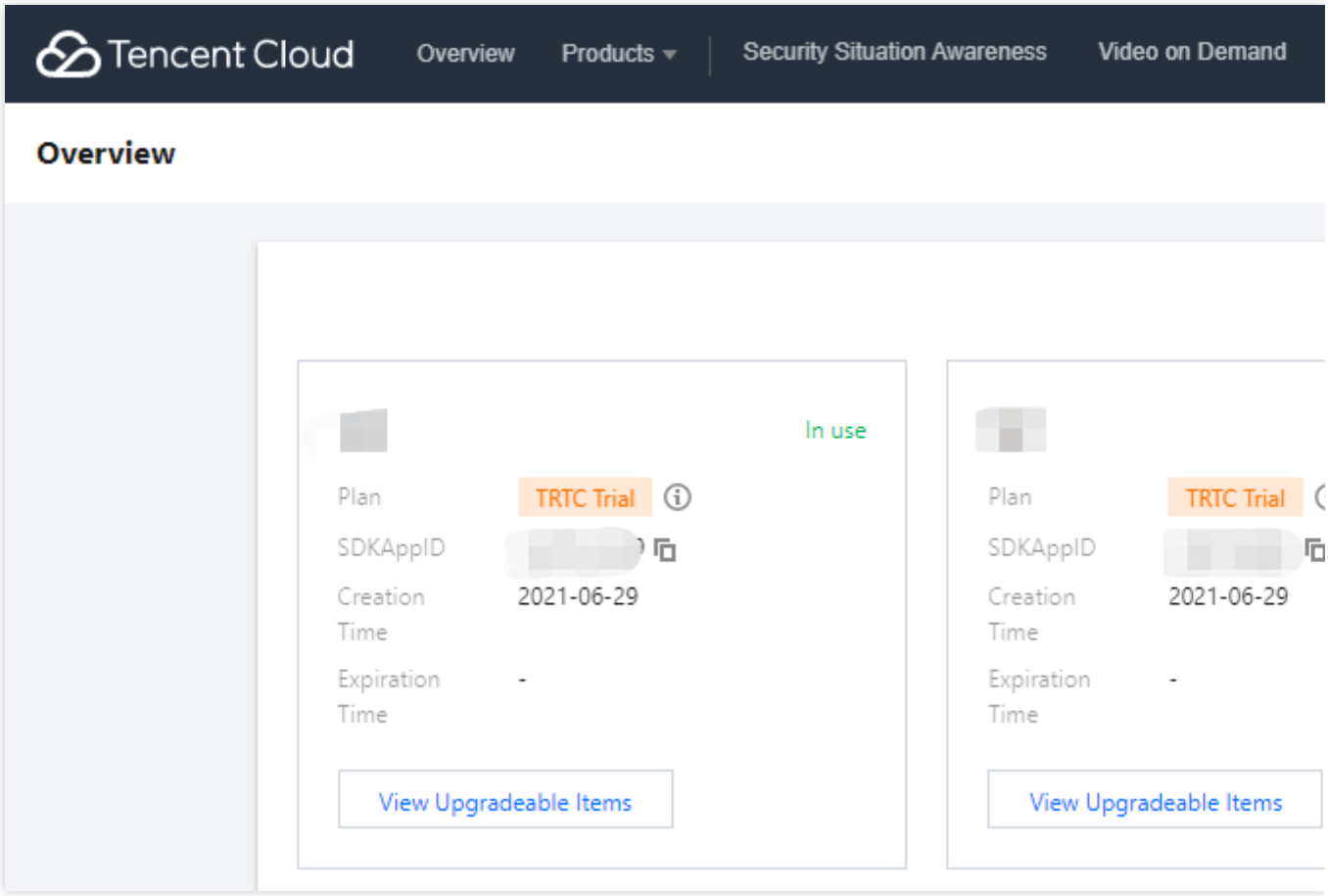
如果您已有应用，请记录其 SDKAppID 并 获取密钥信息。

同一个腾讯云账号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**

2. 单击**创建新应用**，在**创建应用**对话框中输入您的应用名称，单击**确定**。

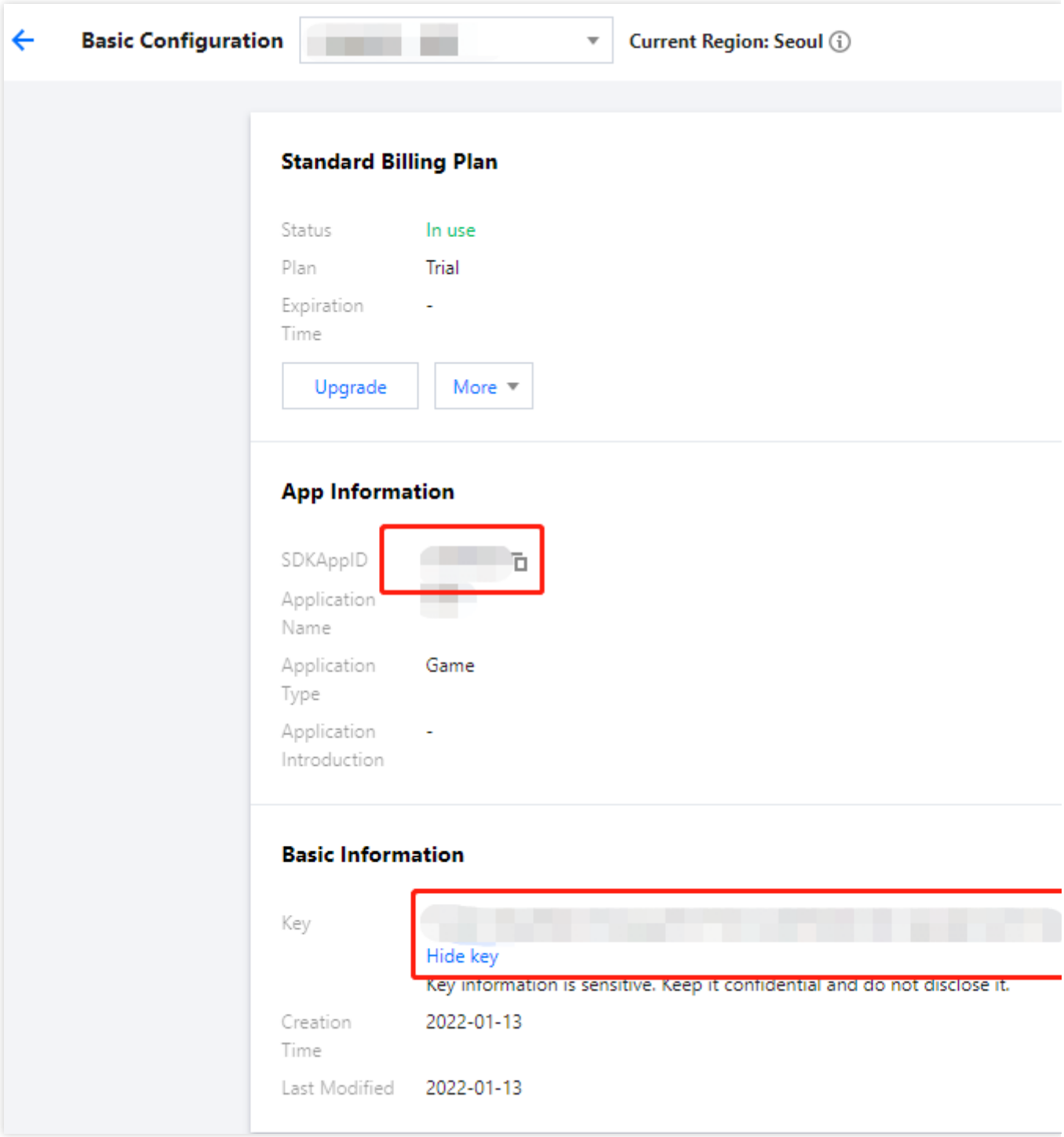


3. 创建完成后，可在控制台总览页查看新建应用的状态、业务版本、SDKAppID、创建时间、标签以及到期时间。请记录 SDKAppID 信息。



步骤2：获取密钥信息

1. 单击目标应用卡片，进入应用的基础配置页面。



2. 在**基本信息**区域，单击**显示密钥**，复制并保存密钥信息。

注意

请妥善保管密钥信息，谨防泄露。

步骤3：下载并配置 Demo 源码

1. 下载即时通信 IM Demo 工程，具体下载地址请参见 SDK 下载。

说明

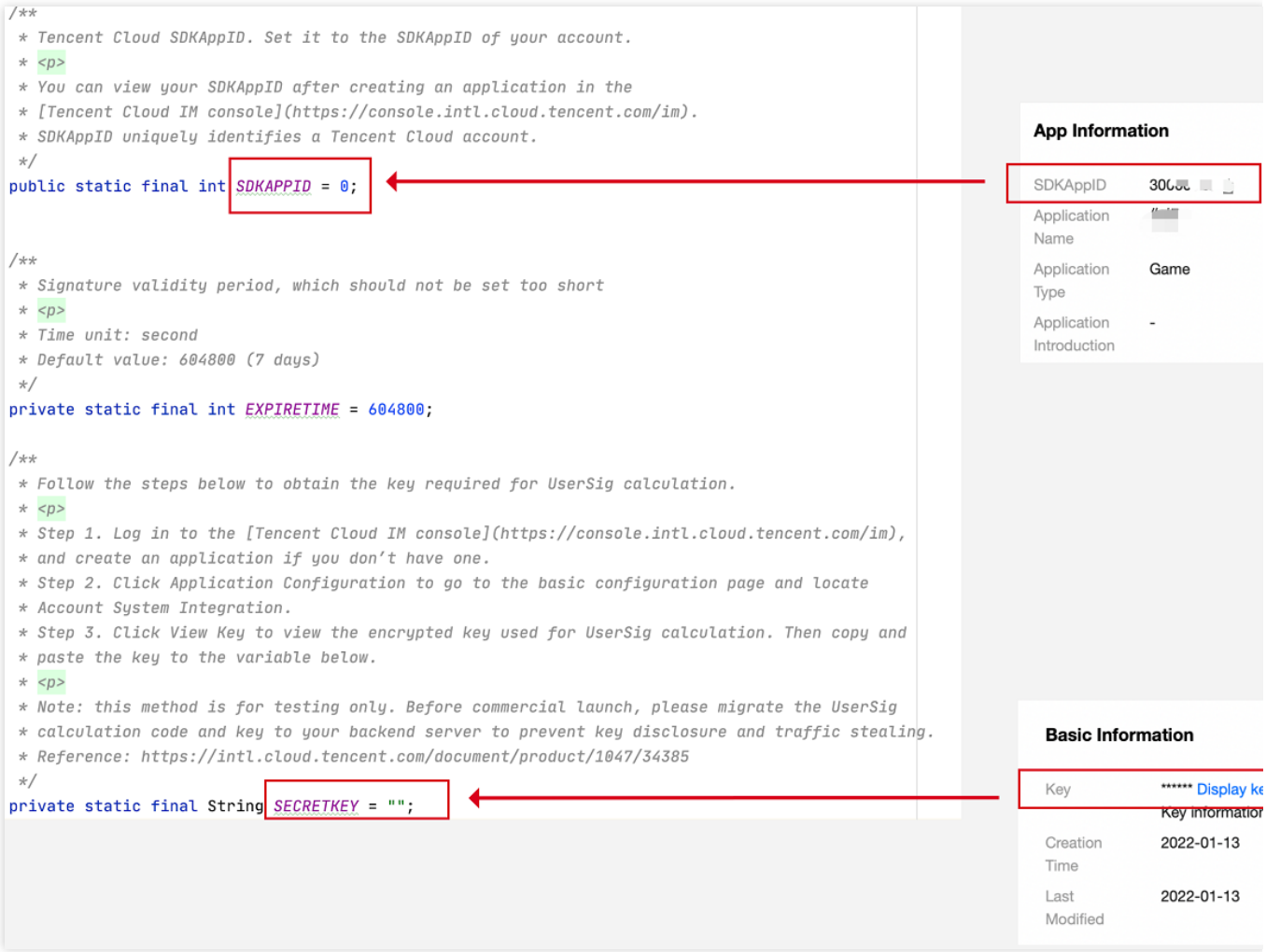
为尊重表情设计版权，下载的 Demo 工程中不包含大表情元素切图，您可以使用自己本地表情包来配置代码。未授权使用 IM Demo 中的表情包可能会构成设计侵权。

打开终端目录的工程，找到对应的 `GenerateTestUserSig` 文件，路径为 `Android/Demo/app/src/main/java/com/tencent/qcloud/tim/demo/signature/GenerateTestUserSig.java`

2. 设置 `GenerateTestUserSig` 文件中的相关参数：

SDKAPPID：请设置为 步骤1 中获取的实际应用 SDKAppID。

SECRETKEY：请设置为 步骤2 中获取的实际密钥信息。



注意

本文提到的获取 UserSig 的方案是在客户端代码中配置 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 Demo 和功能调试**。

正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

步骤4：编译运行

用 Android Studio 导入工程直接编译运行即可。

更多详情可参见 步骤3 克隆的 Demo 工程中对应目录下的 `README.md` 文件。

开发环境要求

Android Studio-Chipmunk

Gradle-6.7.1

Android Gradle Plugin Version-4.2.0

kotlin-gradle-plugin-1.5.31

注意

Demo 默认集成了音视频通话功能，由于该功能依赖的音视频 SDK 暂不支持模拟器，请使用真机调试或者运行 Demo。

iOS

最近更新时间：2024-05-13 16:40:30

本文主要介绍如何快速运行腾讯云即时通信 IM Demo（iOS）。

操作步骤

步骤1：创建应用

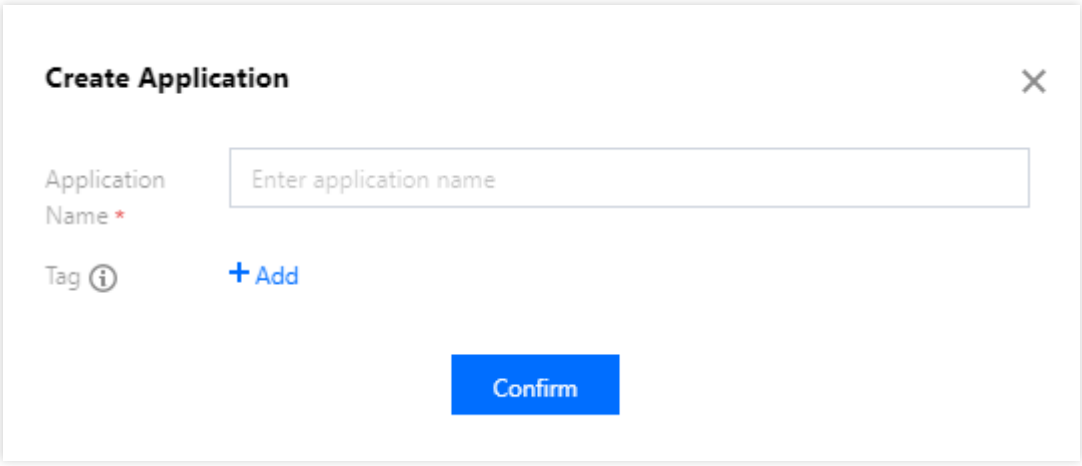
1.1 登录 即时通信 IM 控制台。

说明

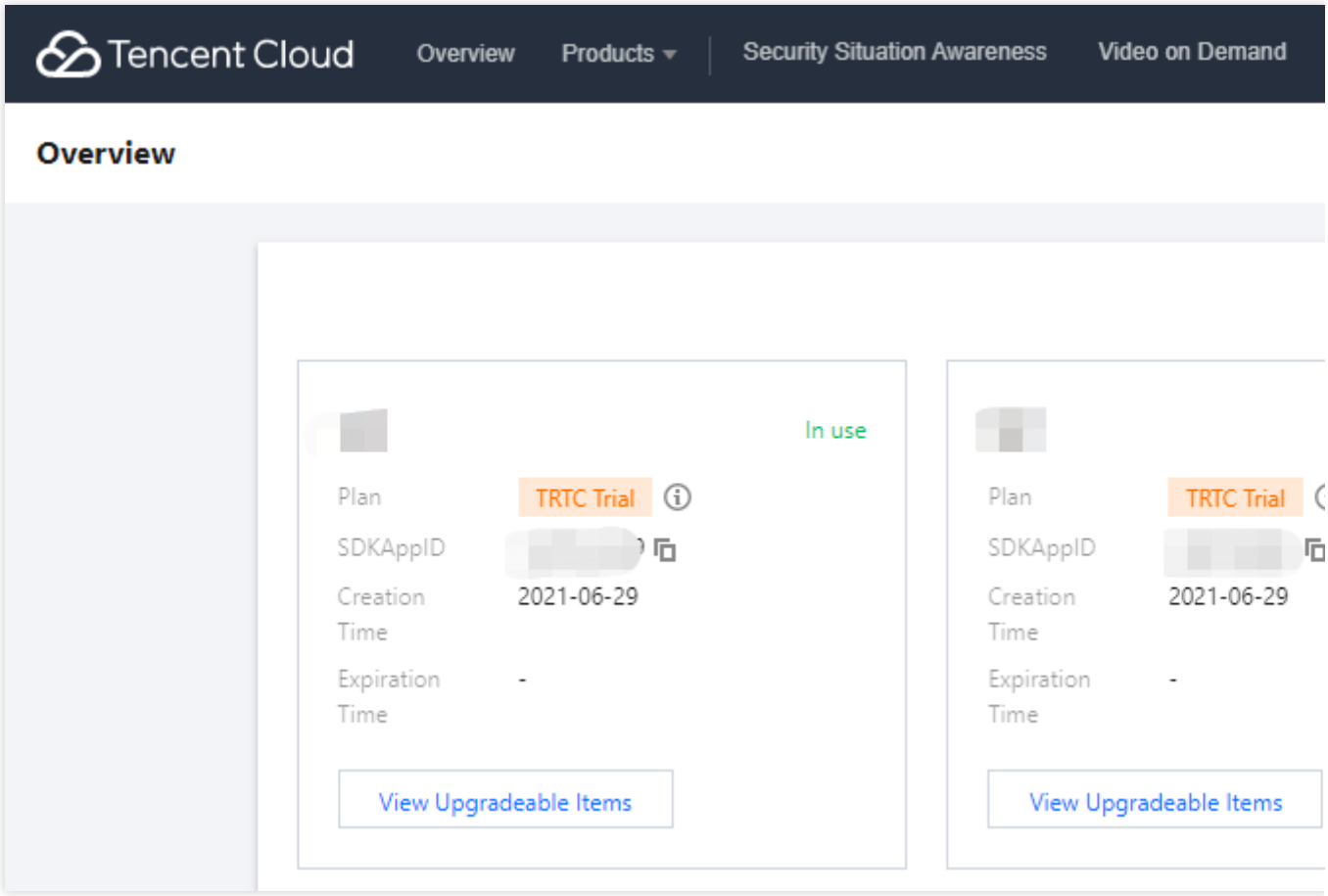
如果您已有应用，请记录其 SDKAppID 并 获取密钥信息。

同一个腾讯云帐号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**

1.2 单击**创建新应用**，在**创建应用**对话框中输入您的应用名称，单击**确定**。

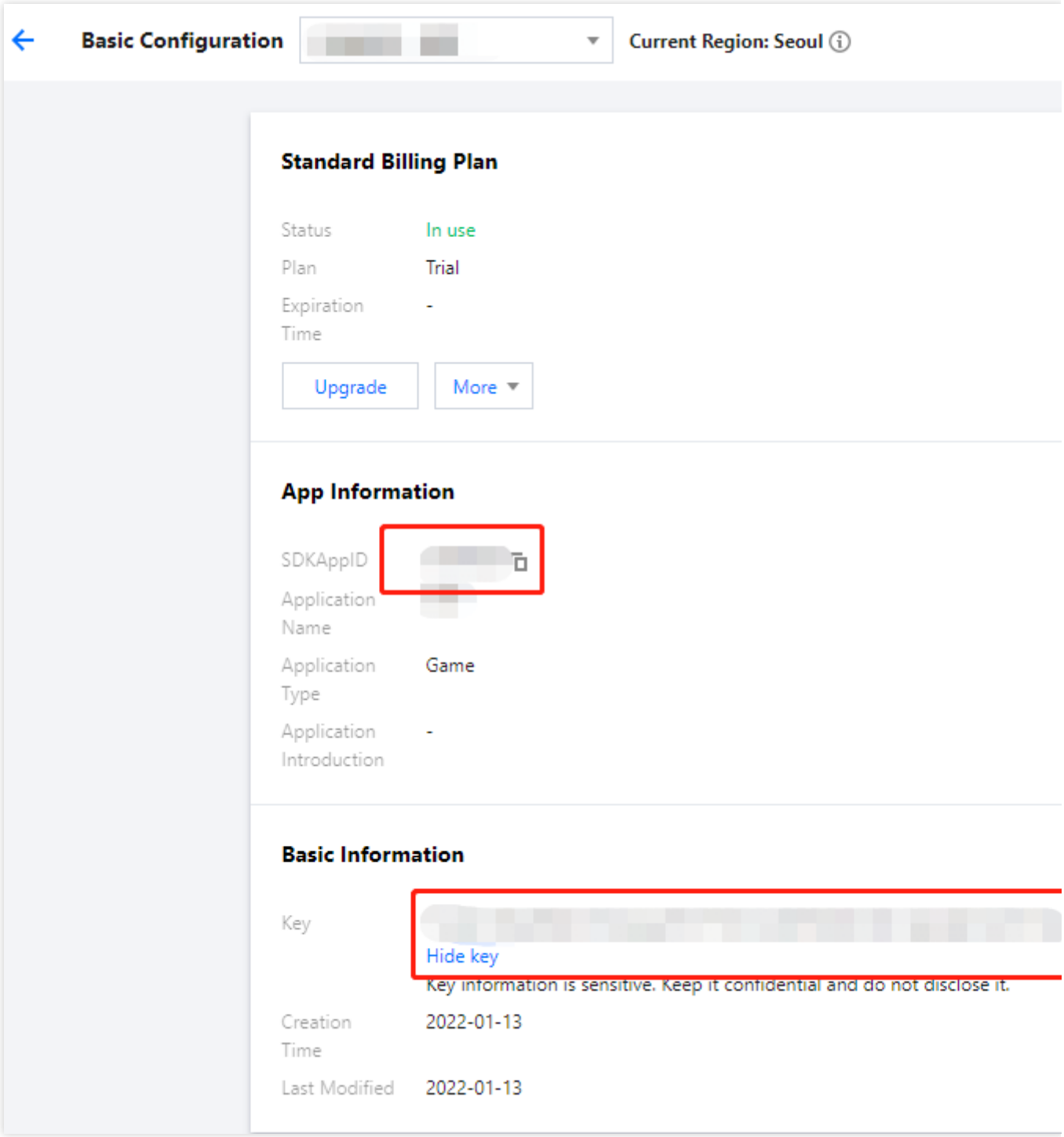


1.3 创建完成后，可在控制台总览页查看新建应用的状态、业务版本、SDKAppID、创建时间、标签以及到期时间。请记录 SDKAppID 信息。



步骤2：获取密钥信息

1.1 单击目标应用卡片，进入应用的基础配置页面。



1.2 在**基本信息**区域，单击**显示密钥**，复制并保存密钥信息。

注意

请妥善保管密钥信息，谨防泄露。

步骤3：下载并配置 Demo 源码

1.1 下载即时通信 IM Demo 工程，具体下载地址请参见 SDK 下载。

说明

为尊重表情设计版权，下载的 Demo 工程中不包含大表情元素切图，您可以使用自己本地表情包来配置代码。未授权使用 IM Demo 中的表情包可能会构成设计侵权。

1.2 打开所属终端目录的工程，找到对应的 `GenerateTestUserSig` 文件。

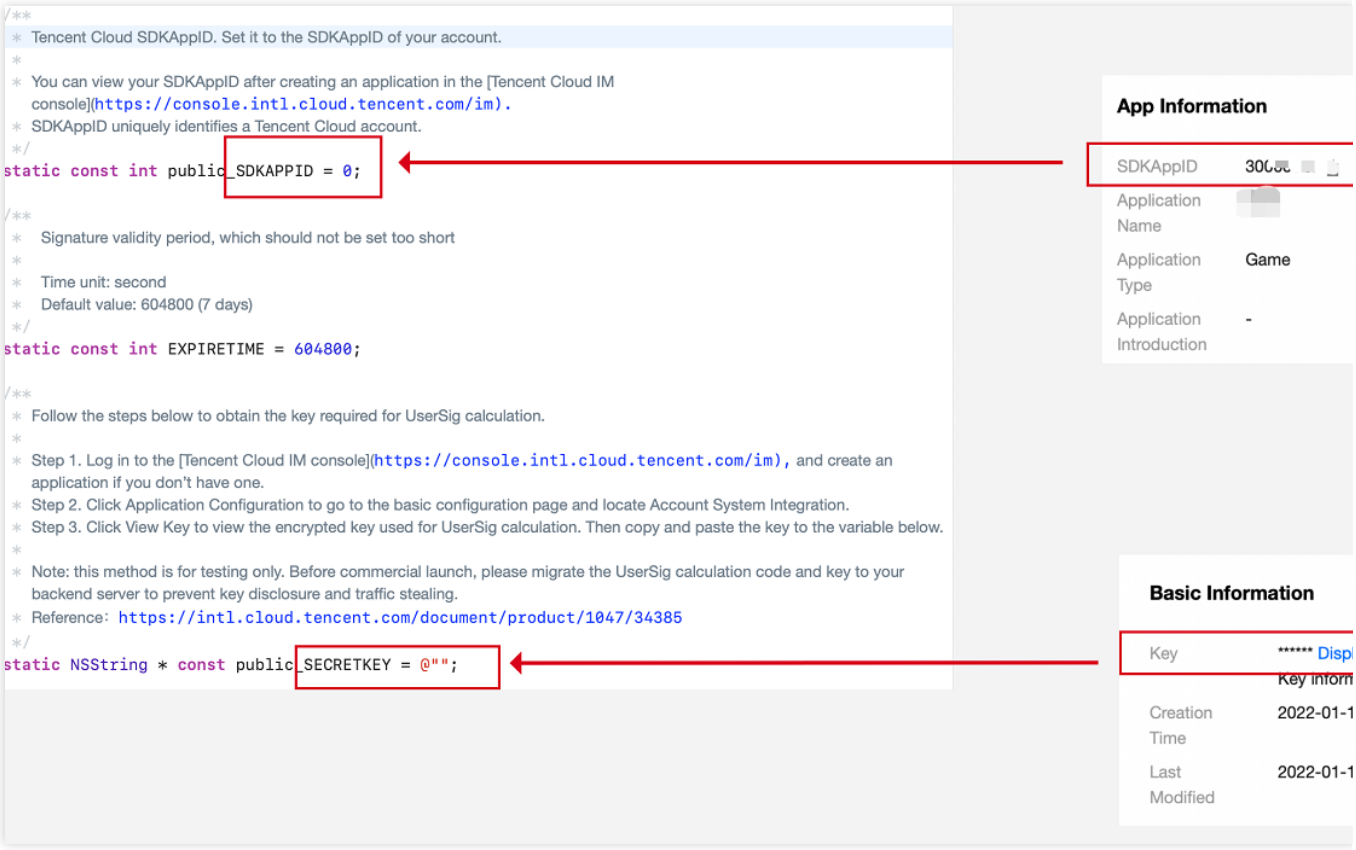
iOS 路径：iOS/Demo/TUIKitDemo/Private/GenerateTestUserSig.h

Mac 路径：Mac/Demo/TUIKitDemo/Debug/GenerateTestUserSig.h

1.3 设置 `GenerateTestUserSig` 文件中的相关参数：

SDKAPPID：请设置为 步骤1 中获取的实际应用 SDKAppID。

SECRETKEY：请设置为 步骤2 中获取的实际密钥信息。



注意

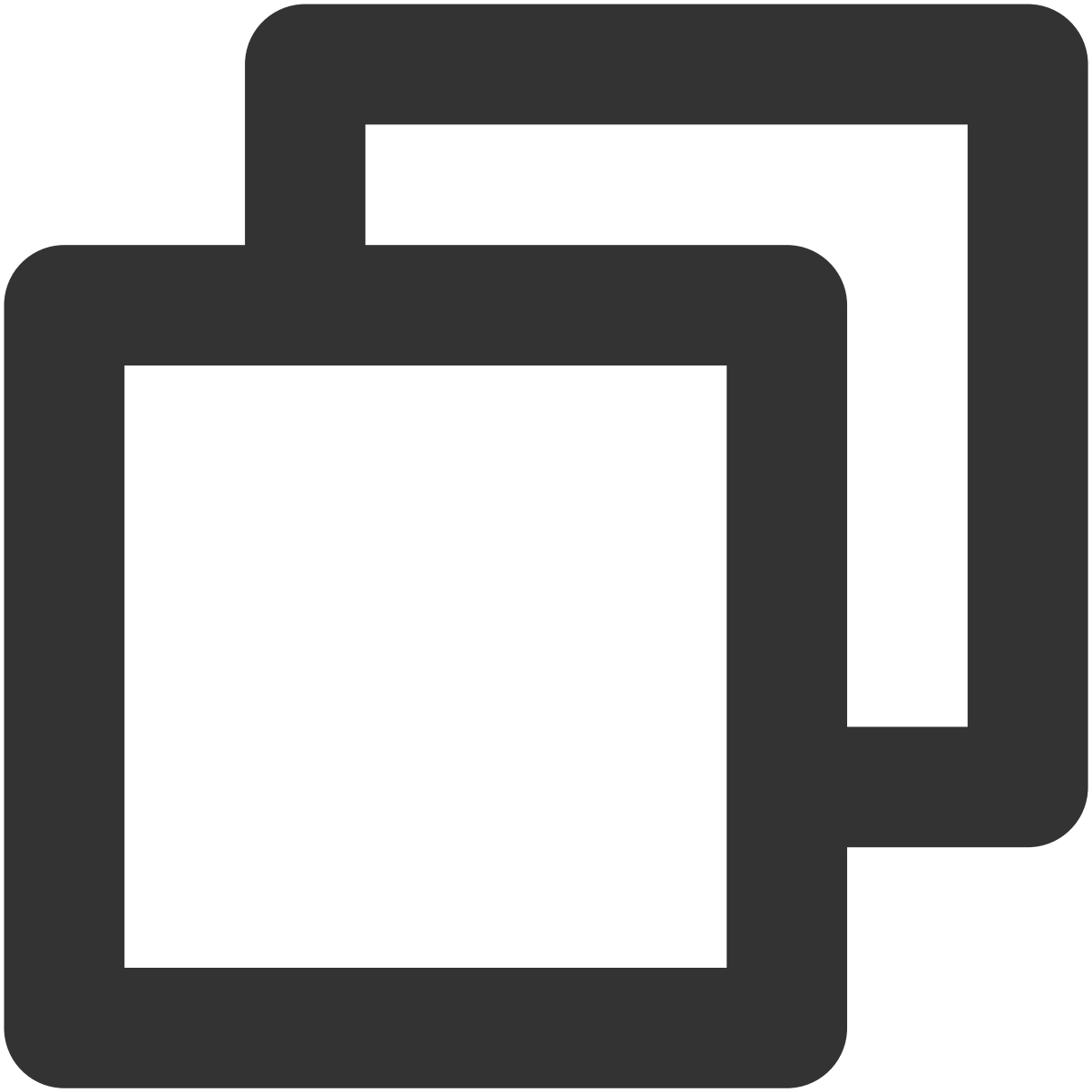
本文提到的获取 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通 Demo 和功能调试**。

正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

步骤4：编译运行

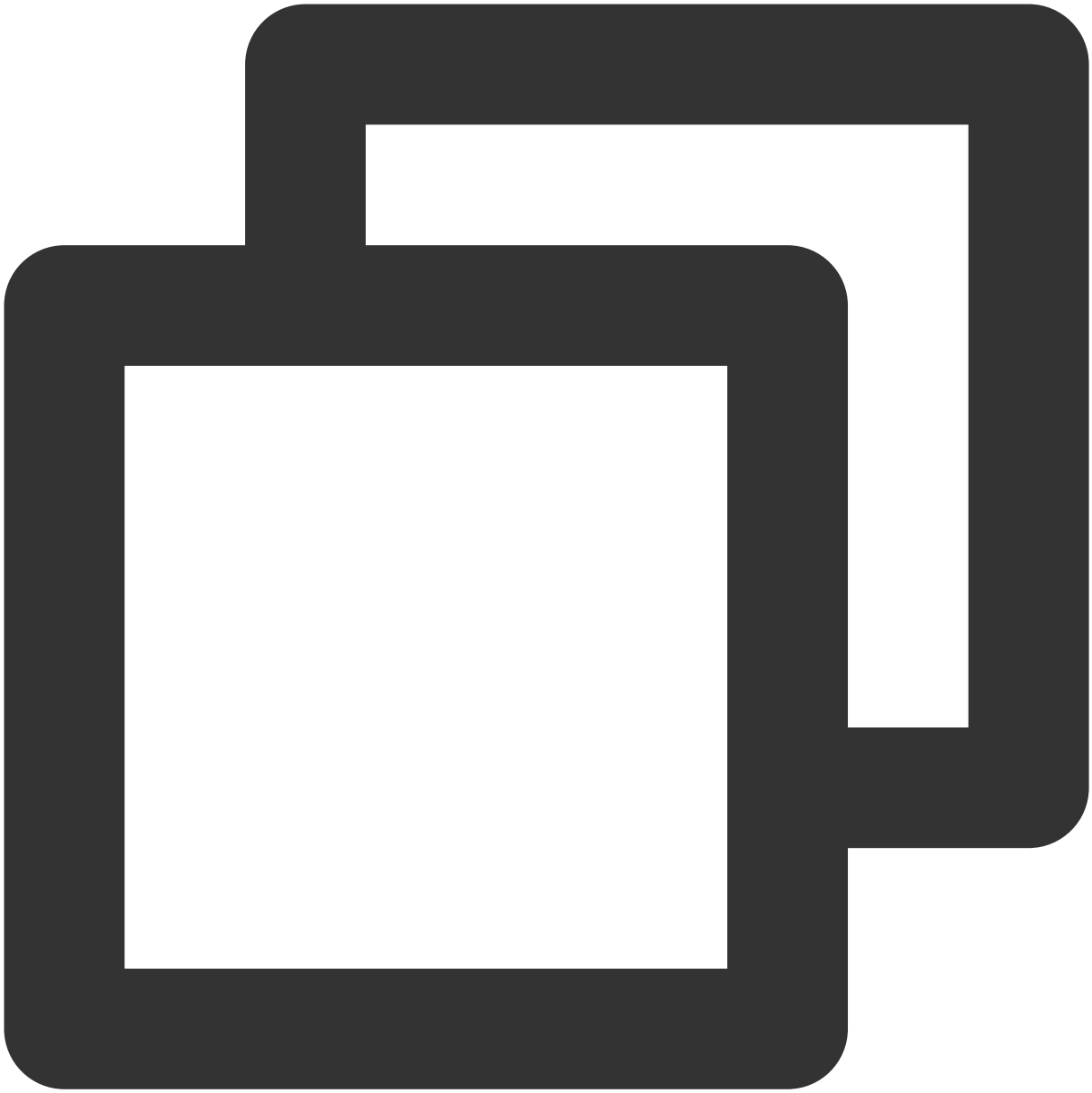
可参见 步骤3 克隆的 Demo 工程中对应目录下的 `README.md` 文件。

1.1 终端执行以下命令，检查 pod 版本。



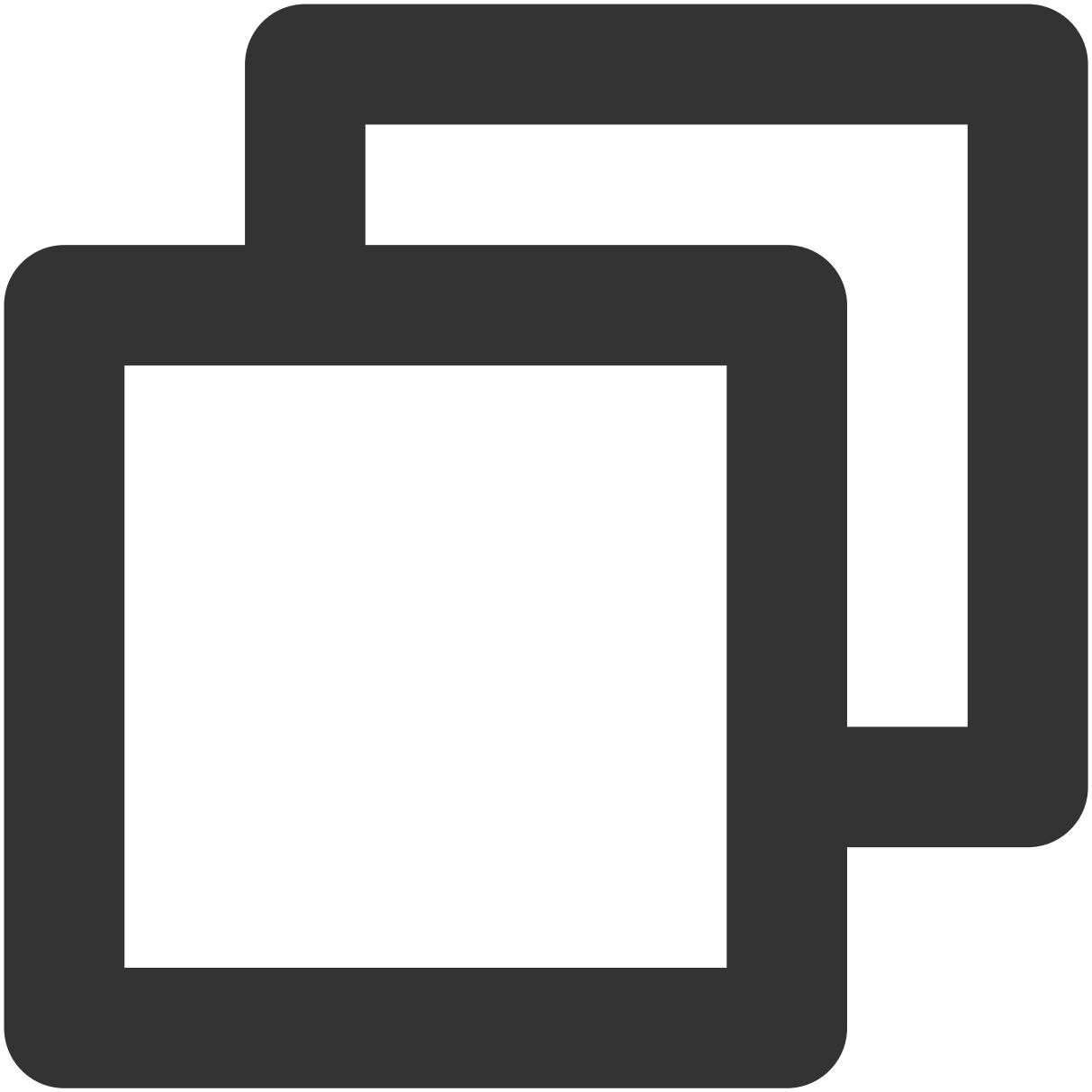
```
pod --version
```

若提示 pod 不存在，或 pod 版本小于 1.7.5，请执行以下命令安装最新 pod。



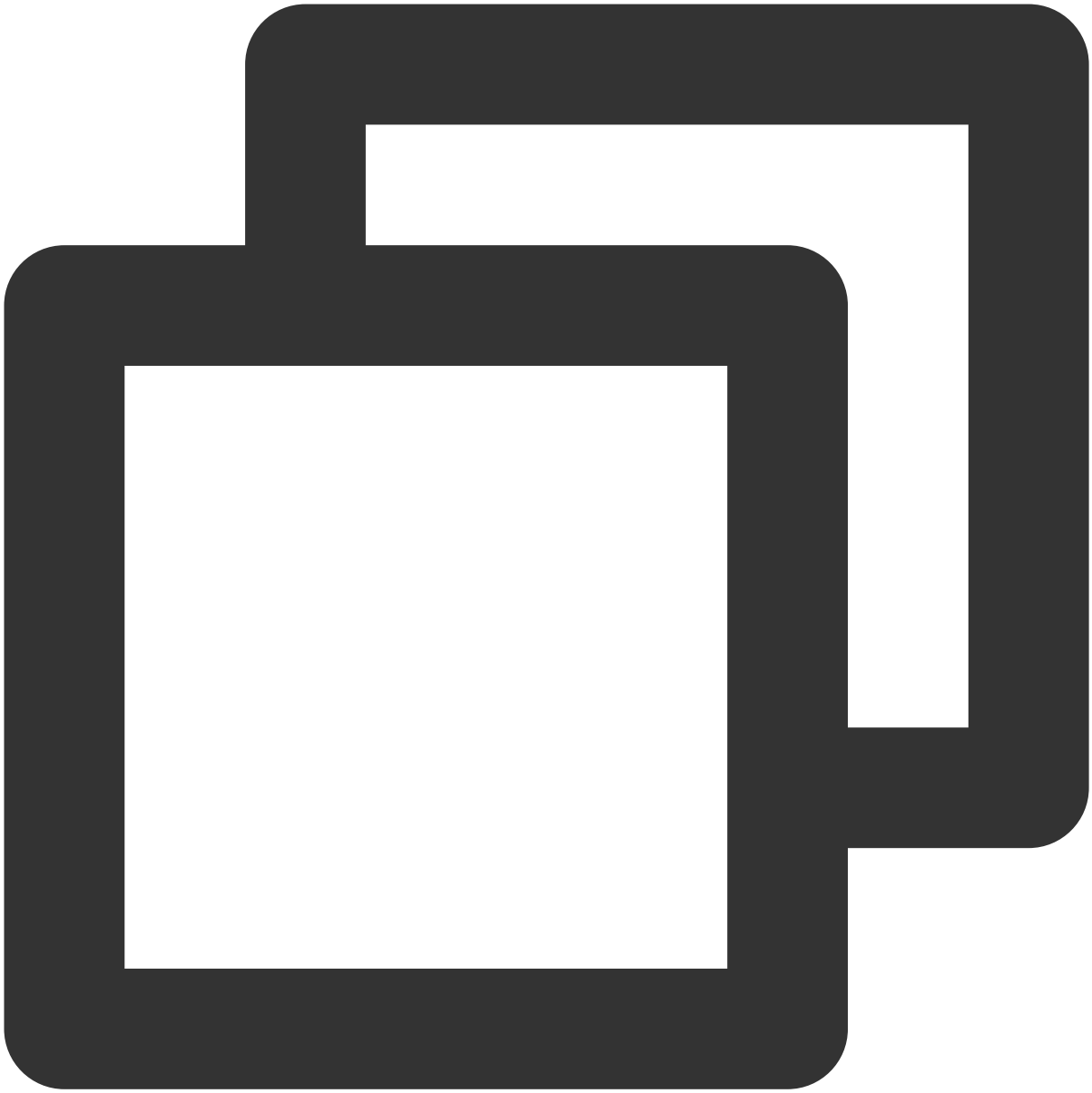
```
//更换 gem 源
gem sources --remove https://rubygems.org/
gem sources --add https://gems.ruby-china.com/
//安装 pod
sudo gem install cocoapods -n /usr/local/bin
//如果安装了多个 Xcode ，请使用下面的命令选择 Xcode 版本（一般选择最新的 Xcode 版本）
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer
//更新 pod 本地库
pod setup
```

1.2 终端执行以下命令，安装依赖库。



```
//iOS
cd iOS/TUIKitDemo
pod install
//Mac
cd Mac/TUIKitDemo
pod install
```

如果安装失败，执行以下命令更新本地的 CocoaPods 仓库列表。



```
pod repo update
```

1.3 编译运行：

iOS 进入 iOS/TUIKitDemo 文件夹，打开 `TUIKitDemo.xcworkspace` 编译运行。

Mac 进入Mac/TUIKitDemo 文件夹，打开 `TUIKitDemo.xcworkspace` 编译运行。

注意

Demo 默认集成了音视频通话功能，由于该功能依赖的音视频 SDK 暂不支持模拟器，请使用真机调试或者运行 Demo。

进阶功能

UI 界面库
开启视频通话

相关文档

价格说明

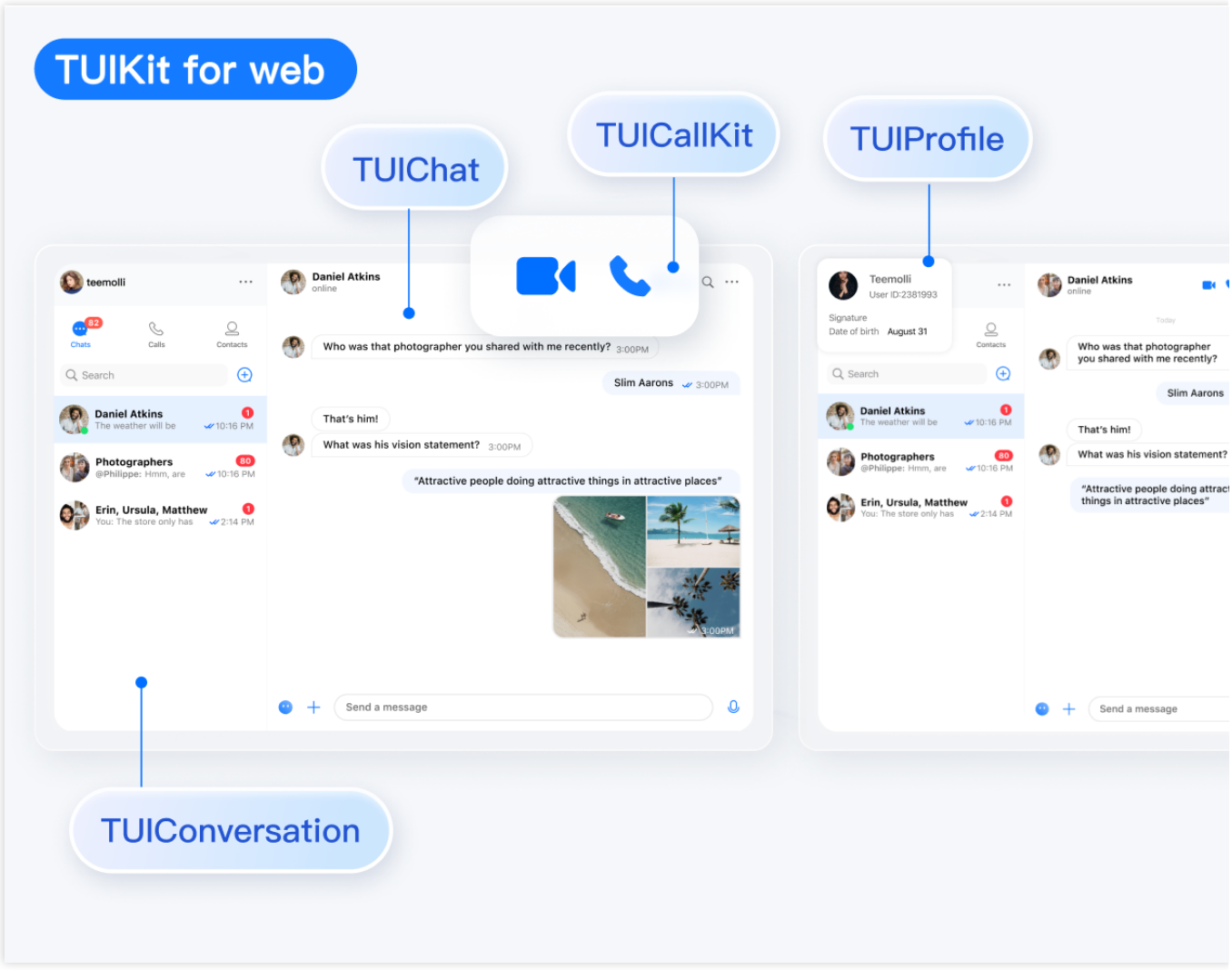
Web React

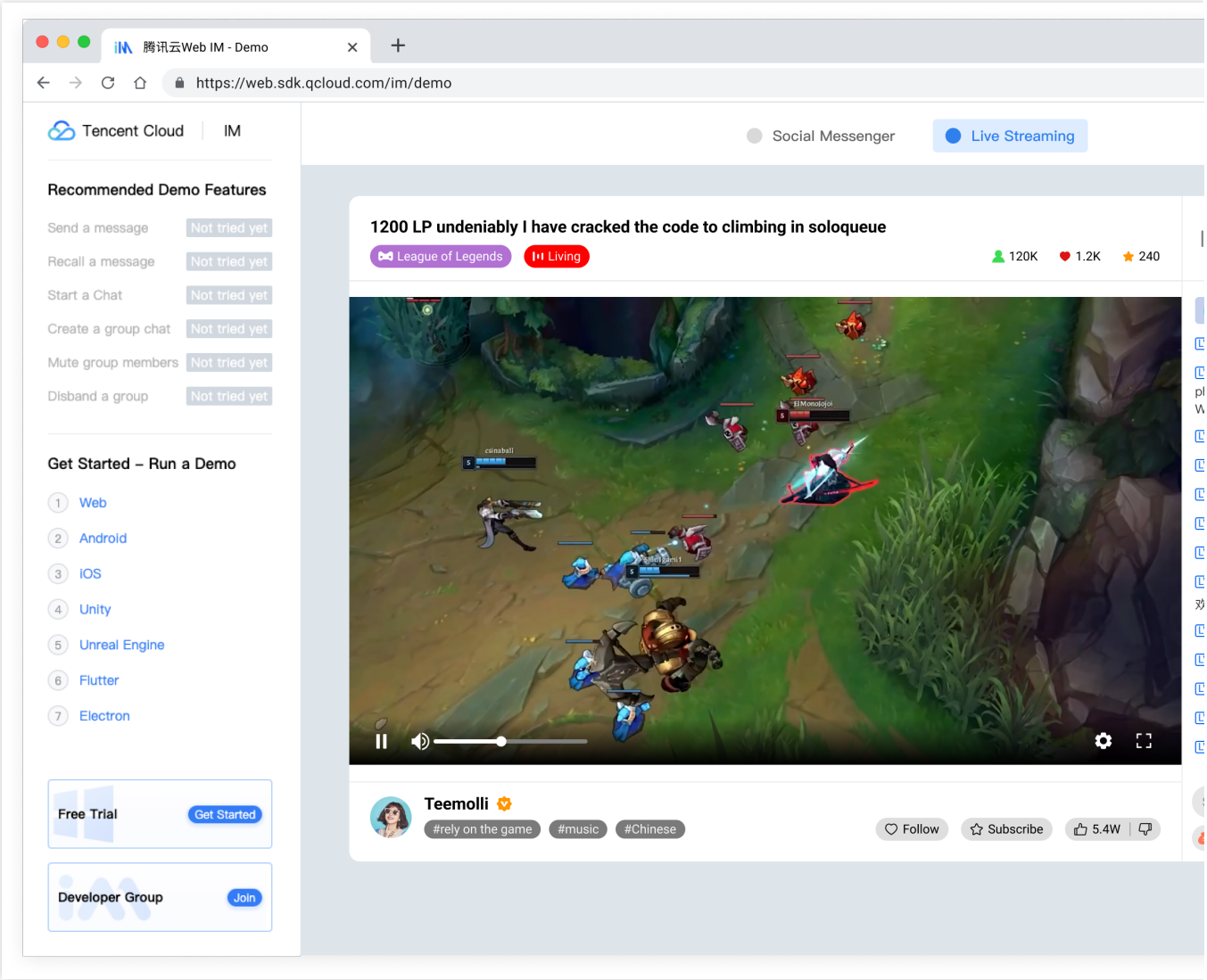
最近更新时间：2024-05-13 16:41:17

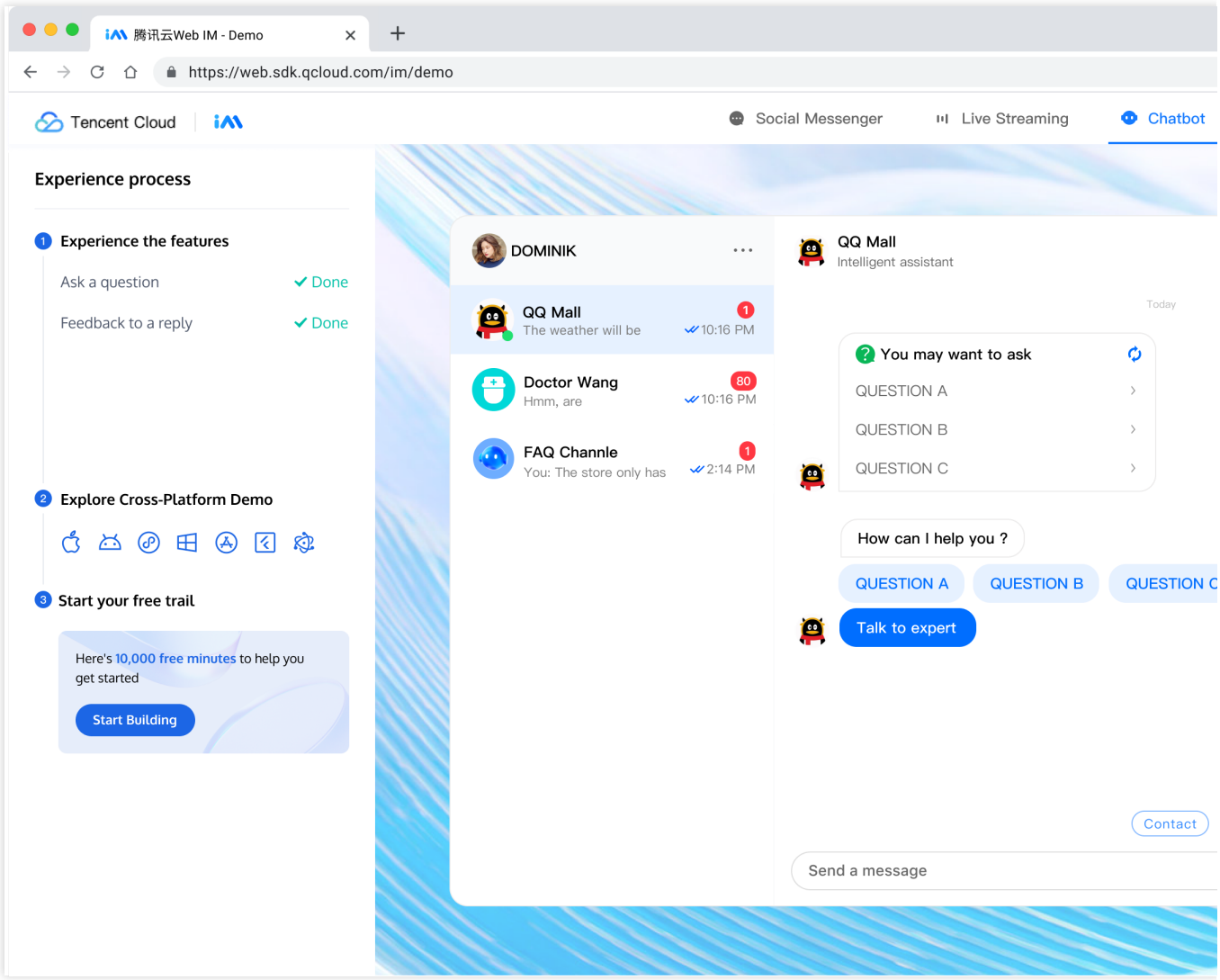
Chat UIKit 是基于腾讯云 IM SDK 的一款 UI 组件库，它提供了一些通用的 UI 组件，包含会话、聊天、关系链、群组、音视频通话等功能。基于 UI 组件您可以像搭积木一样快速搭建起自己的业务逻辑。

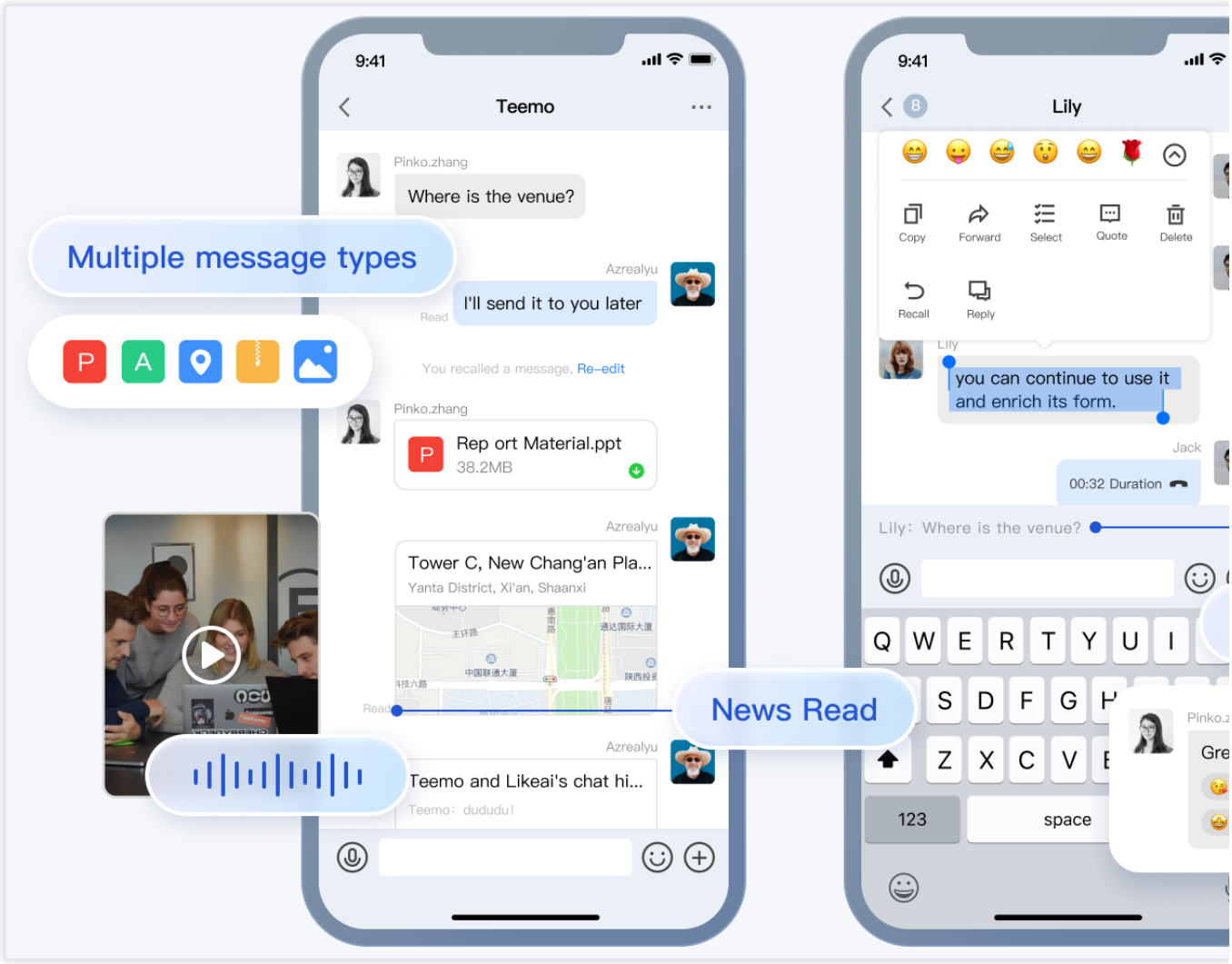
Chat UIKit 中的组件在实现 UI 功能的同时，会调用 IM SDK 相应的接口实现 IM 相关逻辑和数据的处理，因而开发者在使用 Chat UIKit 时只需关注自身业务或个性化扩展即可。

点击体验 Demo。




腾讯云Web IM - Demo
https://web.sdk.qcloud.com/im/demo
Tencent Cloud
IM
Social Messenger
Live Streaming
Recommended Demo Features
Send a message
Recall a message
Start a Chat
Create a group chat
Mute group members
Disband a group
Not tried yet
Get Started – Run a Demo
1 Web
2 Android
3 iOS
4 Unity
5 Unreal Engine
6 Flutter
7 Electron
Free Trial
Get Started
Developer Group
Join
1200 LP undeniably I have cracked the code to climbing in soloqueue
League of Legends
Living
120K
1.2K
240
Teemolli
#rely on the game
#music
#Chinese
Follow
Subscribe
5.4W


腾讯云Web IM - Demo
https://web.sdk.qcloud.com/im/demo
Tencent Cloud
Social Messenger
Live Streaming
Chatbot
Experience process
1 Experience the features
Ask a question
Done
Feedback to a reply
Done
2 Explore Cross-Platform Demo
3 Start your free trail
Here's 10,000 free minutes to help you get started
Start Building
DOMINIK
QQ Mall
The weather will be
10:16 PM
Doctor Wang
Hmm, are
80
10:16 PM
FAQ Channle
You: The store only has
2:14 PM
QQ Mall
Intelligent assistant
Today
You may want to ask
QUESTION A
QUESTION B
QUESTION C
How can I help you ?
QUESTION A
QUESTION B
QUESTION C
Talk to expert
Contact
Send a message

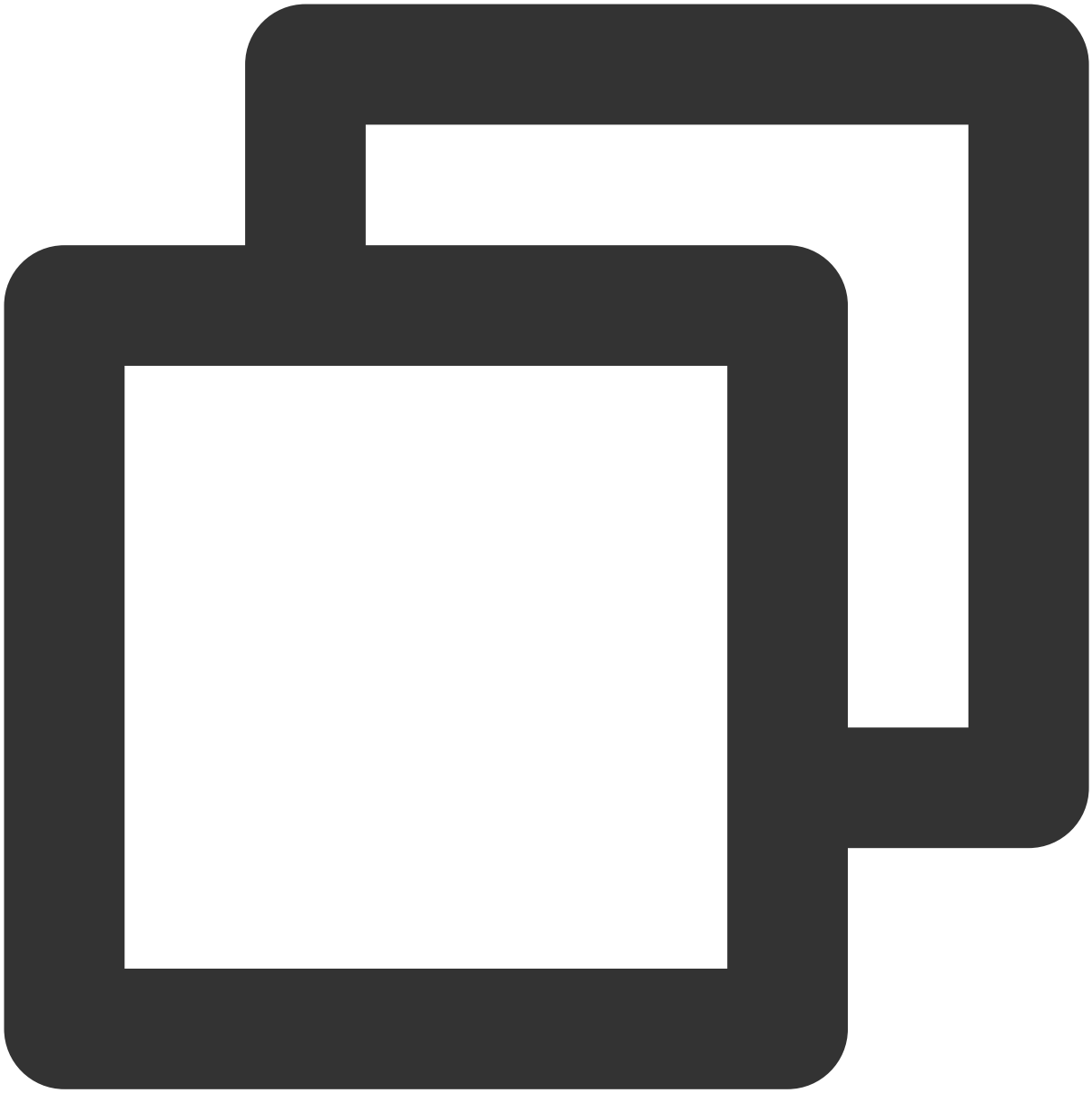


跑通 demo

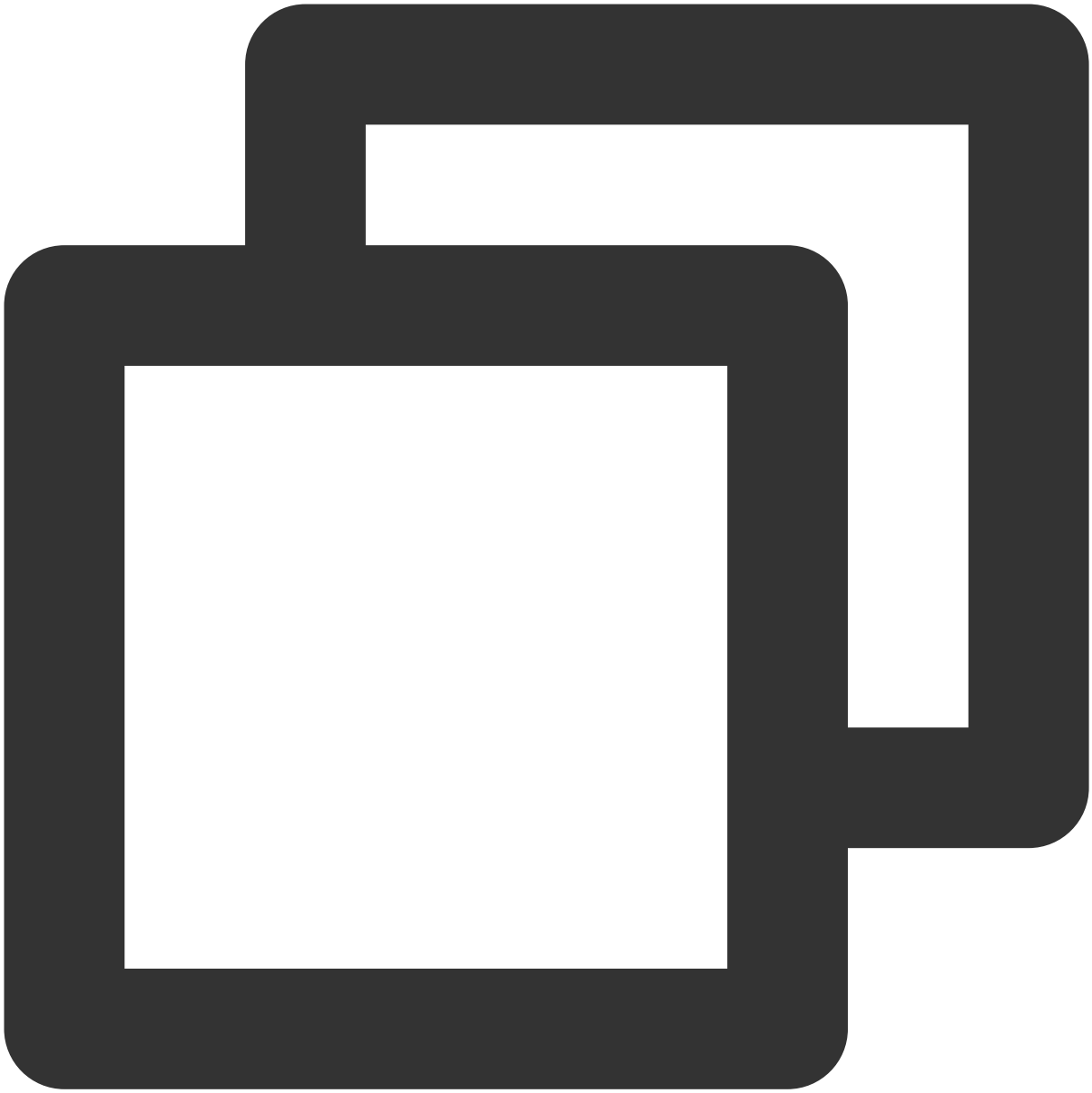
步骤1：下载源码

MacOS

Windows



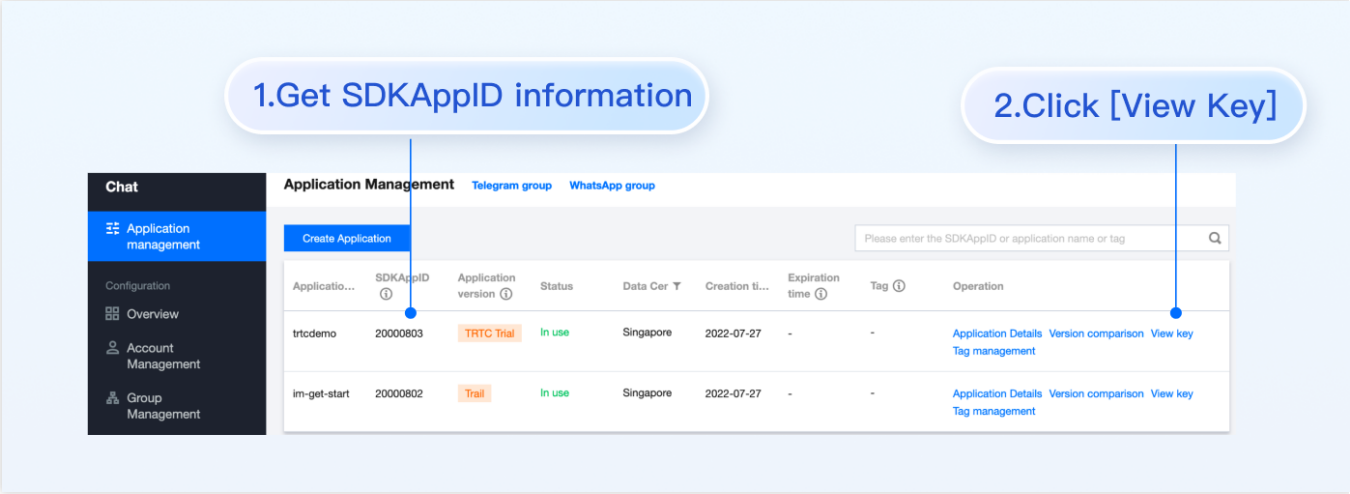
```
# Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
# Go to the project
cd chat-uikit-react
# Install dependencies of the demo
npm install && cd examples/sample-chat && npm install
```

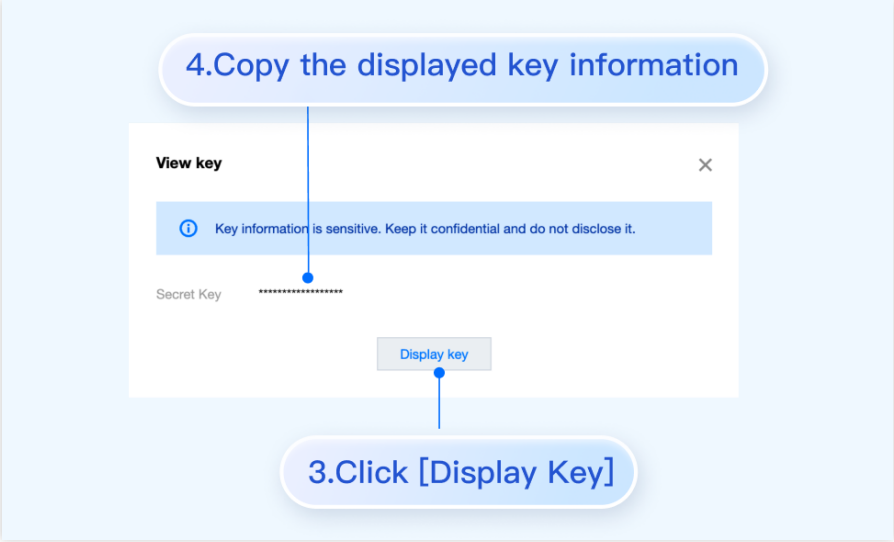


```
# Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
# Go to the project
cd chat-uikit-react
# Install dependencies of the demo
npm install
cd examples/sample-chat
npm install
```

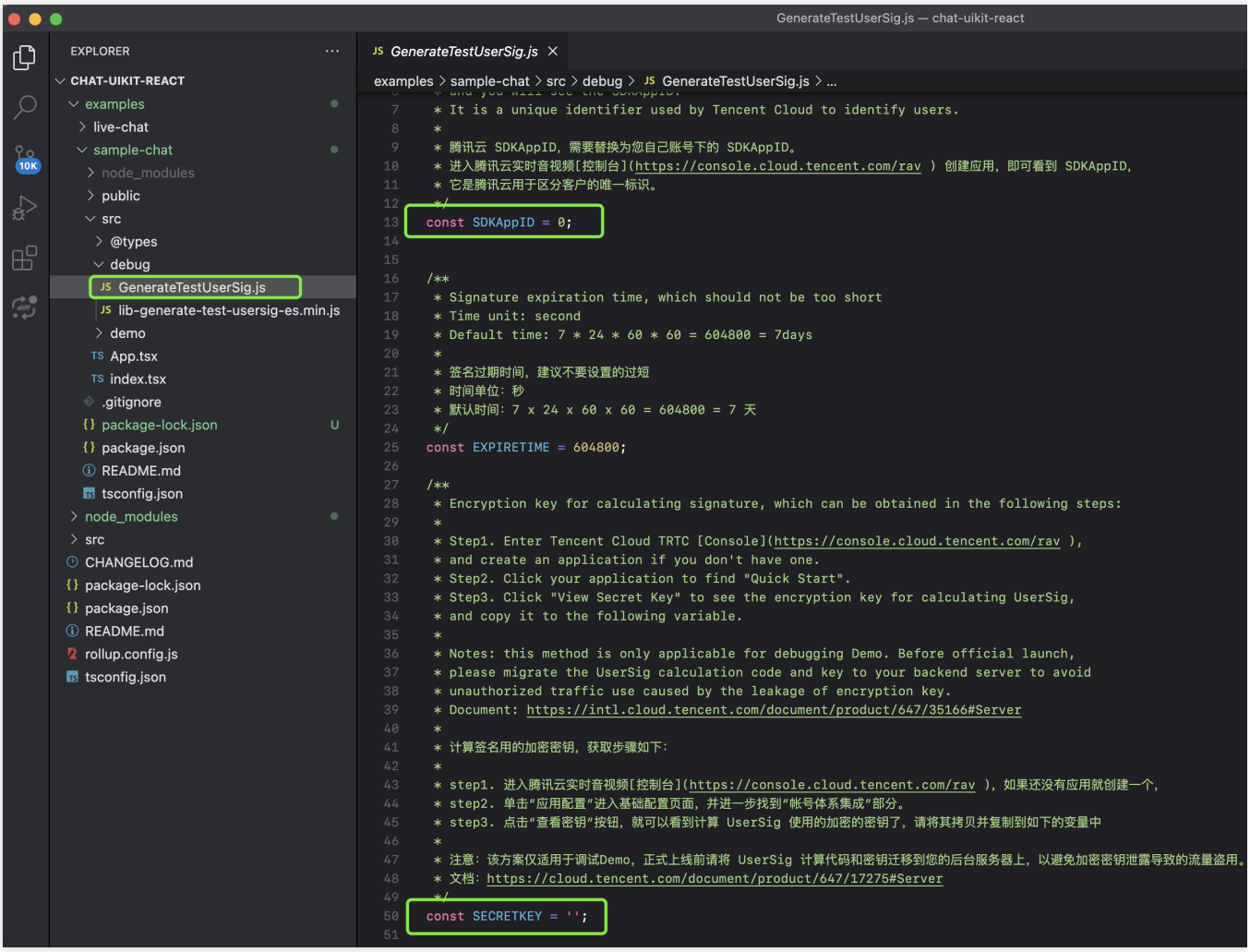
步骤2：配置 demo

1. 打开 `examples/sample-chat` 项目，通过路径 `./examples/sample-chat/src/debug/GenerateTestUserSig.js` 找到 `GenerateTestUserSig.js` 文件。
2. 在 `GenerateTestUserSig.js` 文件中设置 `SDKAPPID` 和 `SECRETKEY` ，其值可以在IM控制台中获取。
点击目标应用卡片，进入其配置页面。
SDKAppID 和 secretKey 等信息，可通过 即时通信 IM 控制台 获取:





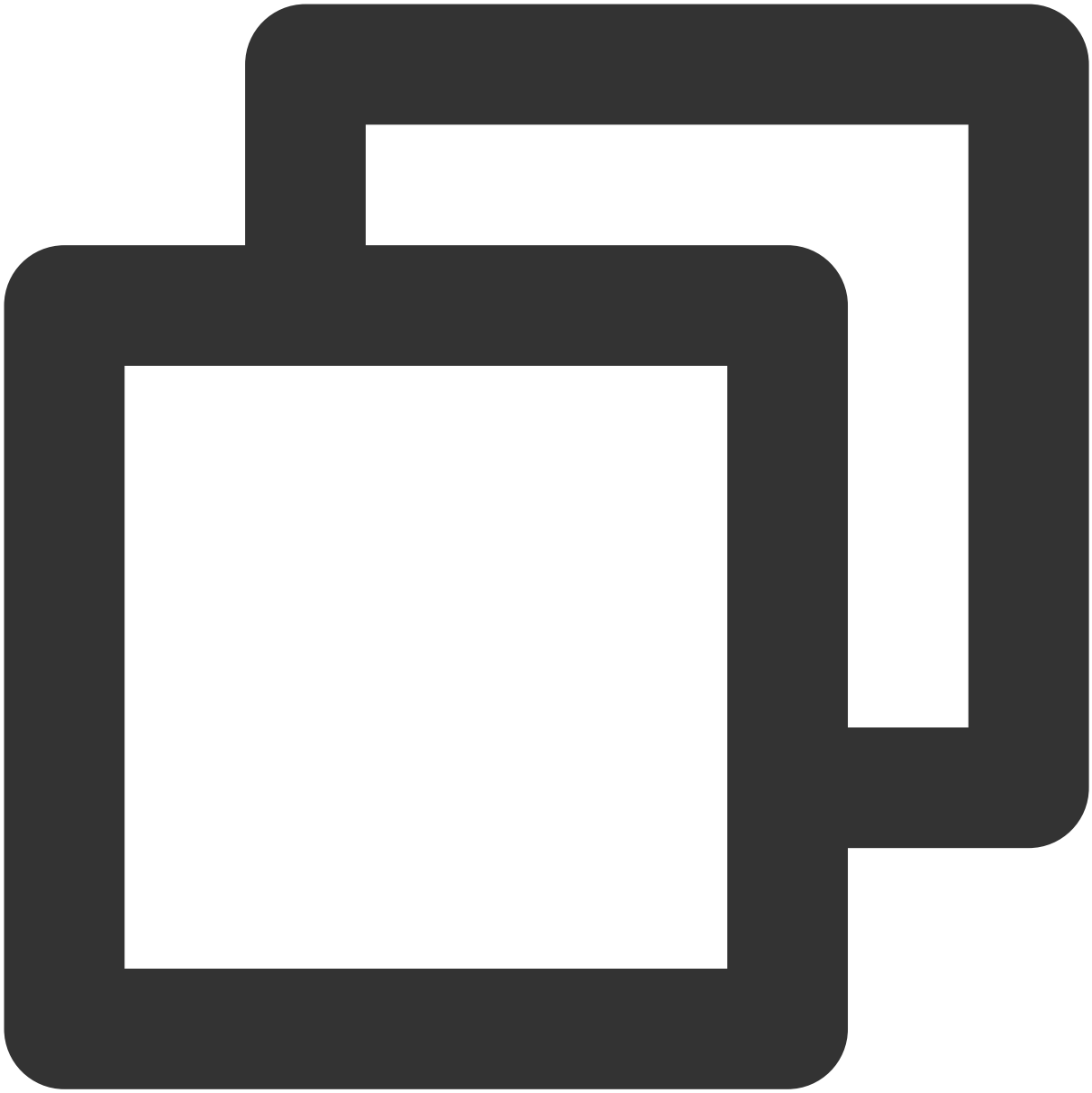
3. 将密钥信息复制并保存到 `./examples/sample-chat/src/debug/GenerateTestUserSig.js` 文件中。



注意：

本文提到的生成 UserSig 的方案是在客户端代码中配置 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 Demo 和功能调试。**正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

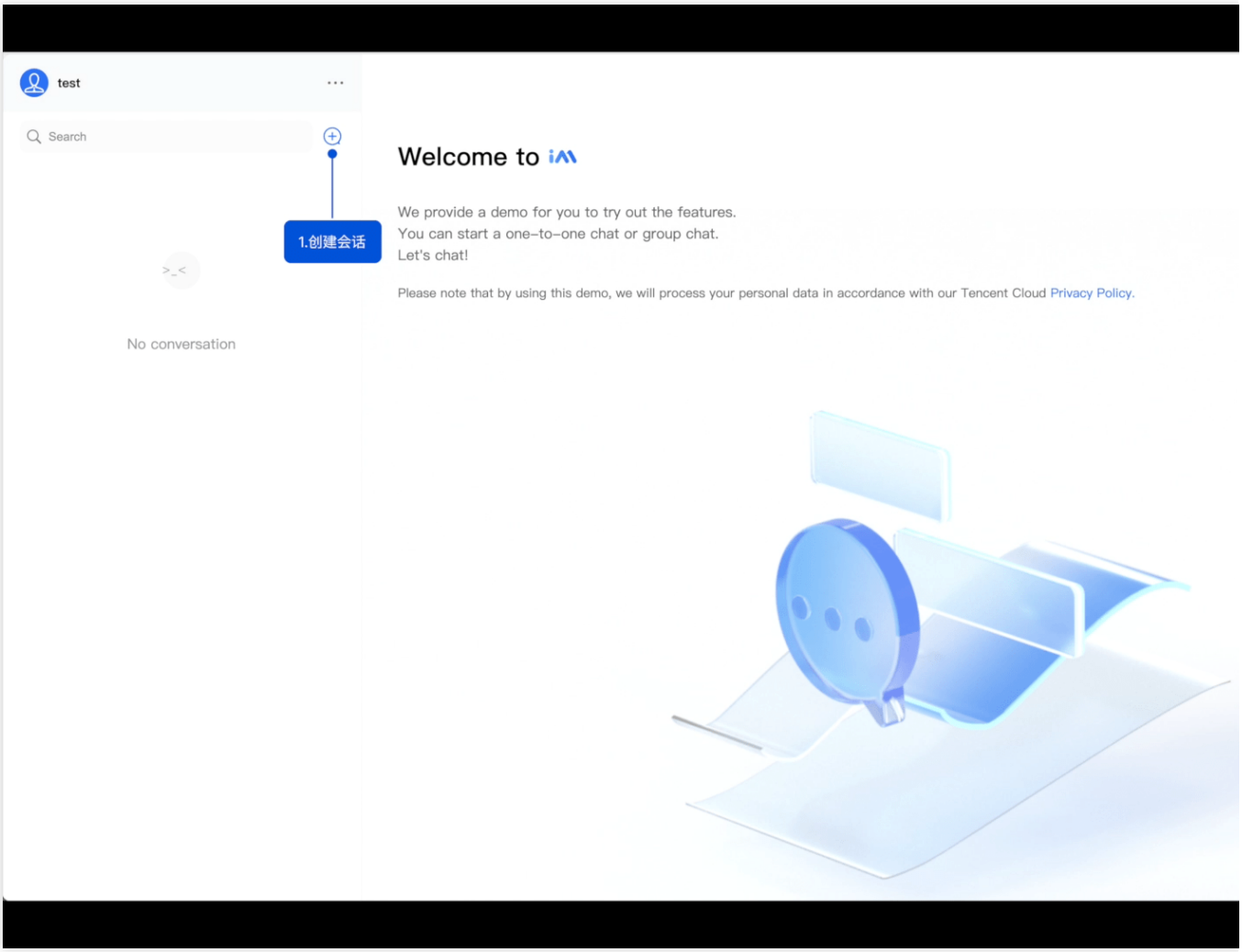
步骤3：启动项目



```
# Launch the project
npm run start
```

步骤4：发送您的第一条消息

1. 项目启动成功后，点击“+”图标，进行创建会话。
2. 在输入框中搜索另一个用户的 userID。
3. 点击用户头像发起会话。
4. 在输入框输入消息，按下"enter"键发送。



集成 chat-uikit-react

如果您想将 chat-uikit-react 集成到您的项目中，请跳转至 含 UI 集成方案/集成基础功能(React) 。

交流与反馈

加入 Telegram 技术交流群组 或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

Electron

最近更新时间：2024-05-13 16:42:09

本文主要介绍如何快速运行腾讯云即时通信 IM Demo（Electron）并了解集成 Electron SDK 的方法。

环境要求

平台	版本
Electron	13.1.5 及以上版本。
Node.js	v14.2.0

支持平台

目前支持 Macos 和 Windows 两个平台。

体验 DEMO

在开始接入前，您可以体验我们的 DEMO ，快速了解腾讯云 IM Electron SDK。

前提条件

您已 注册腾讯云 账号，并完成 实名认证。

操作步骤

步骤1：创建应用

1. 登录 即时通信 IM 控制台。

说明：

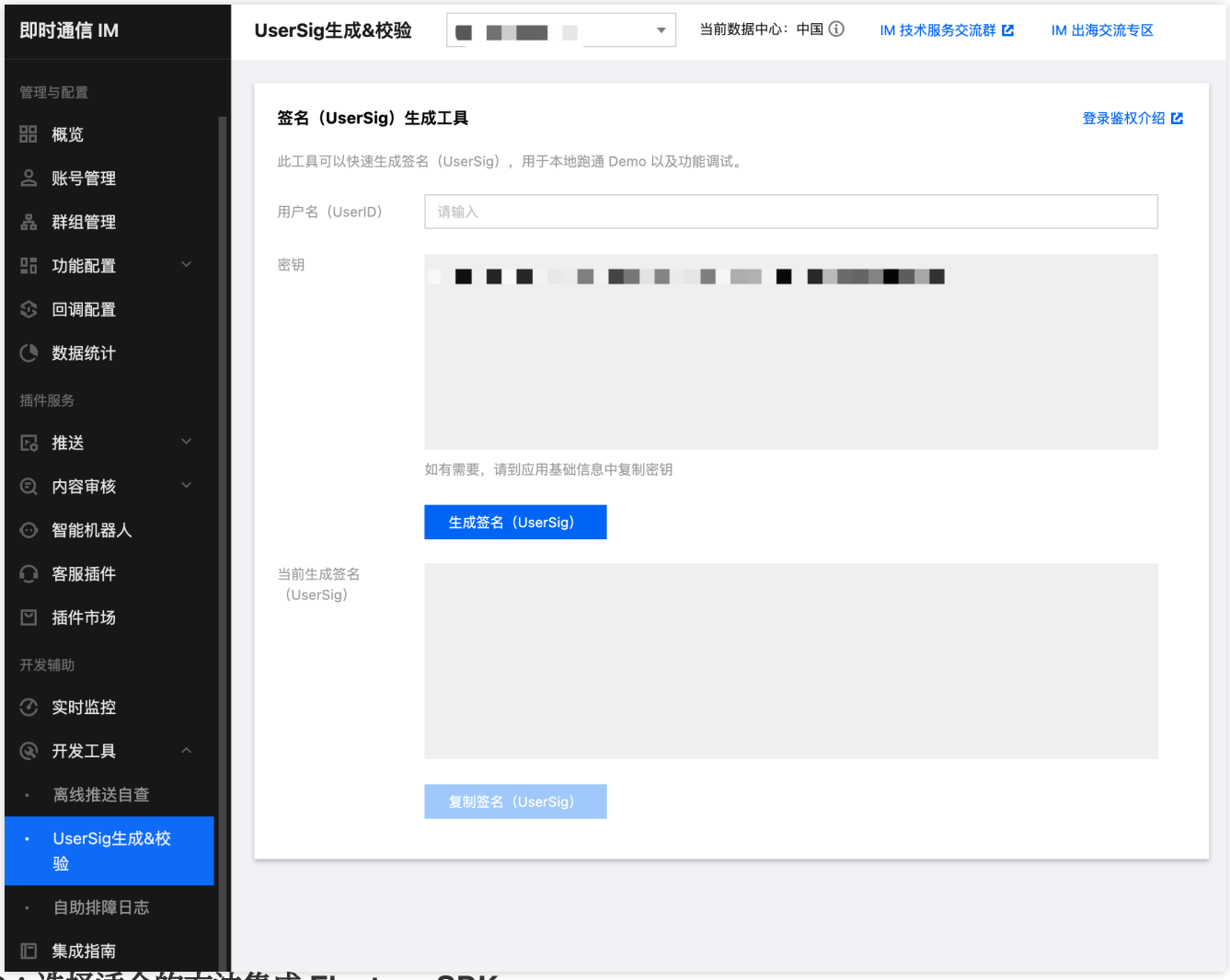
如果您已有应用，请记录其 SDKAppID 并 获取密钥信息。

同一个腾讯云账号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**

2. 单击**创建新应用**，在**创建应用**对话框中输入您的应用名称，单击**确定**。

3. 请保存 SDKAppID 信息。可在控制台总览页查看新建应用的状态、业务版本、SDKAppID、标签、创建时间以及到期时间。

4. 单击创建后的应用，左侧导航栏单击**辅助工具** > **UserSig 生成&校验**，创建一个 UserID 及其对应的 UserSig，复制签名信息，后续登录使用。



步骤2：选择适合的方法集成 Electron SDK

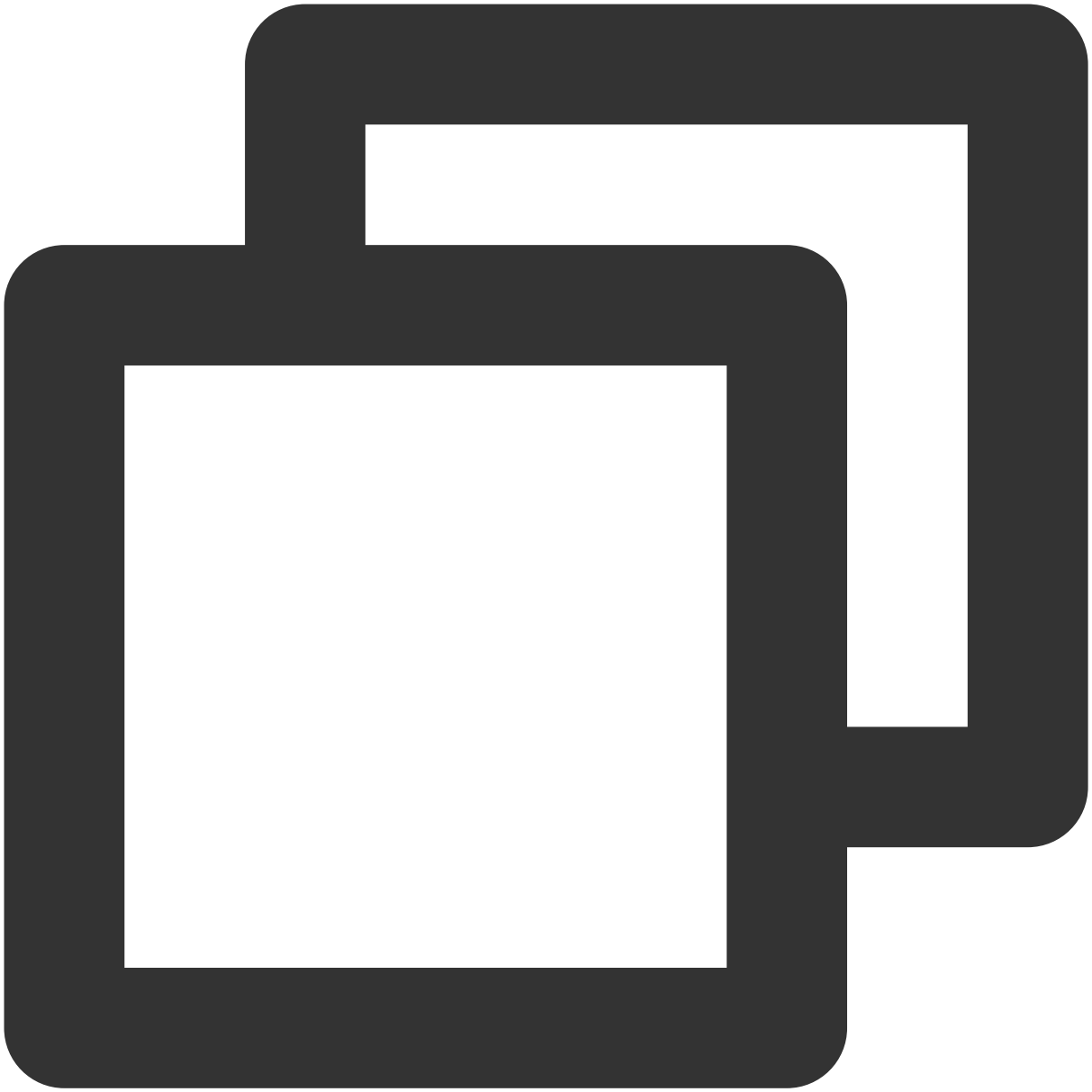
IM 提供了两种方式来即成，您可以选择最合适的方案来即成：

继承方式	适用场景
使用 DEMO	IM Demo包含完整的聊天功能，代码已开源，如果您需要实现聊天类似场景，可以使用 Demo进行二次开发。可立即体验 Demo。
自实现	如果 Demo 不能满足您应用的功能界面需求，可以使用该方法。

为帮助您更好的理解 IM SDK 的各 API，我们还提供了 API 文档。

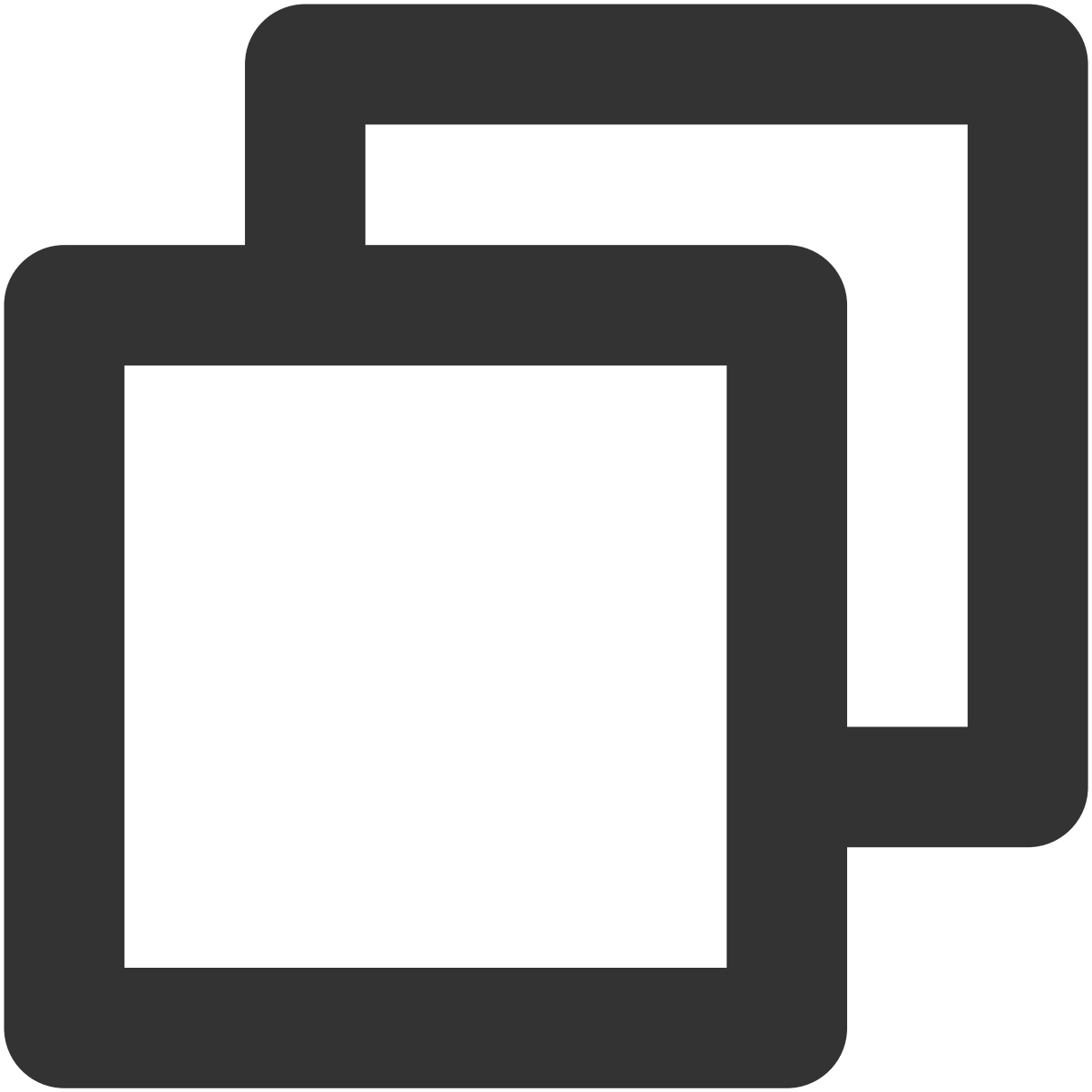
步骤3：使用 Demo

1. 克隆即时通信 IM Electron Demo 源码到本地。



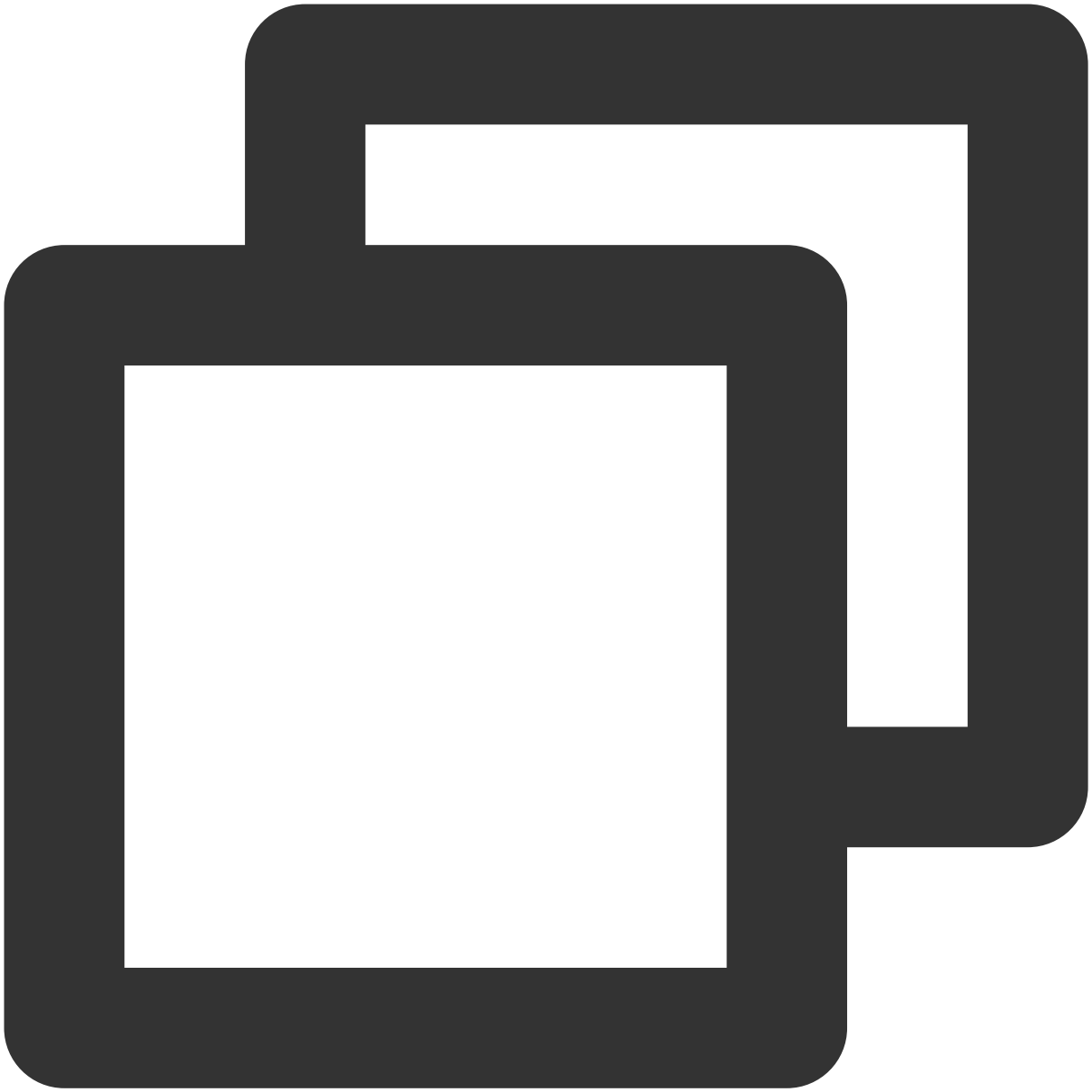
```
git clone https://github.com/TencentCloud/tc-chat-demo-electron.git
```

2. 安装项目依赖。



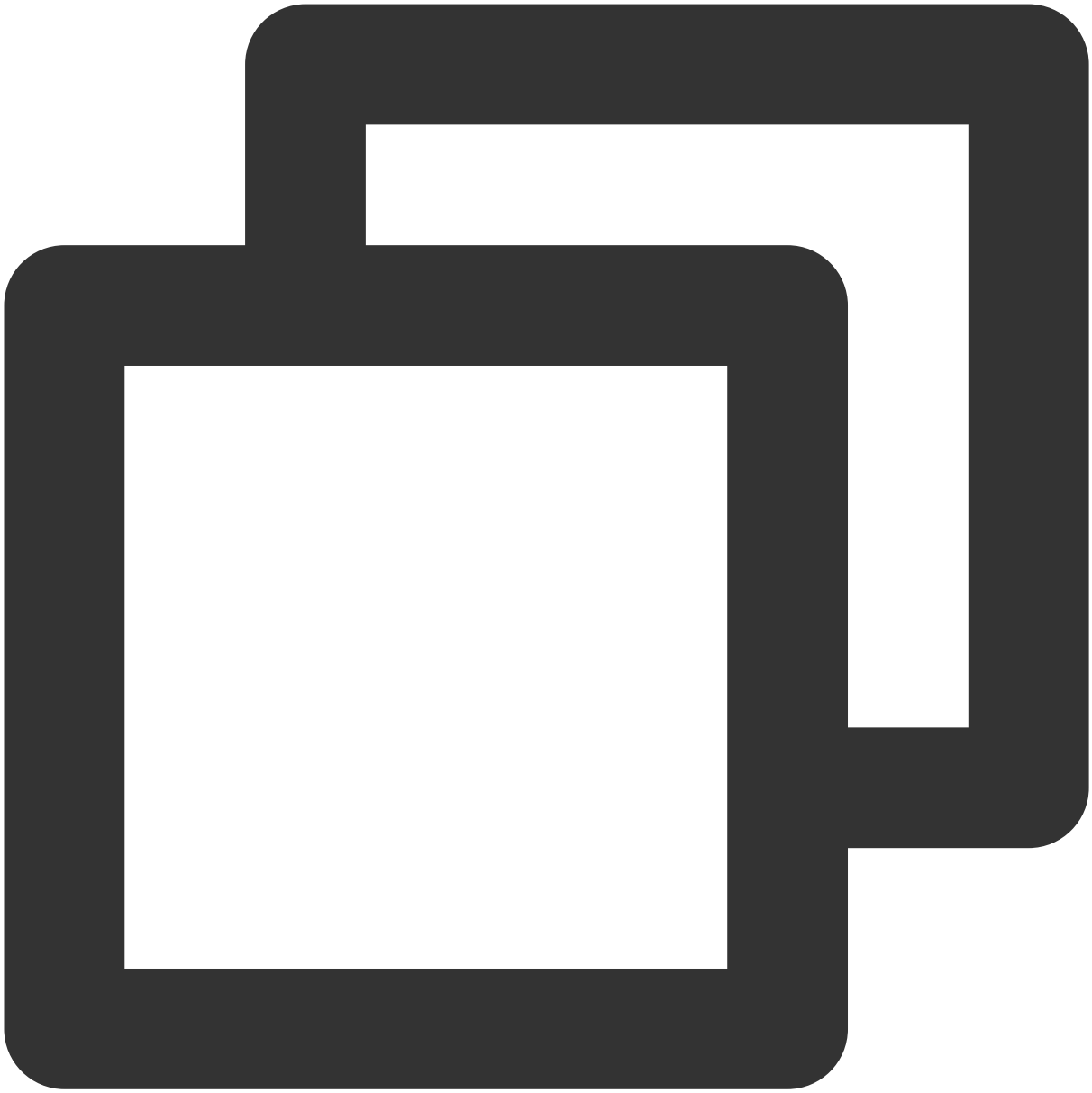
```
// 项目根目录
npm install

// 渲染进程目录
cd src/client
npm install
```

3. 项目运行。



```
// 项目根目录
npm start
```

4. 项目打包。



```
// mac打包
npm run build:mac
// windows打包
npm run build:windows
```

说明：

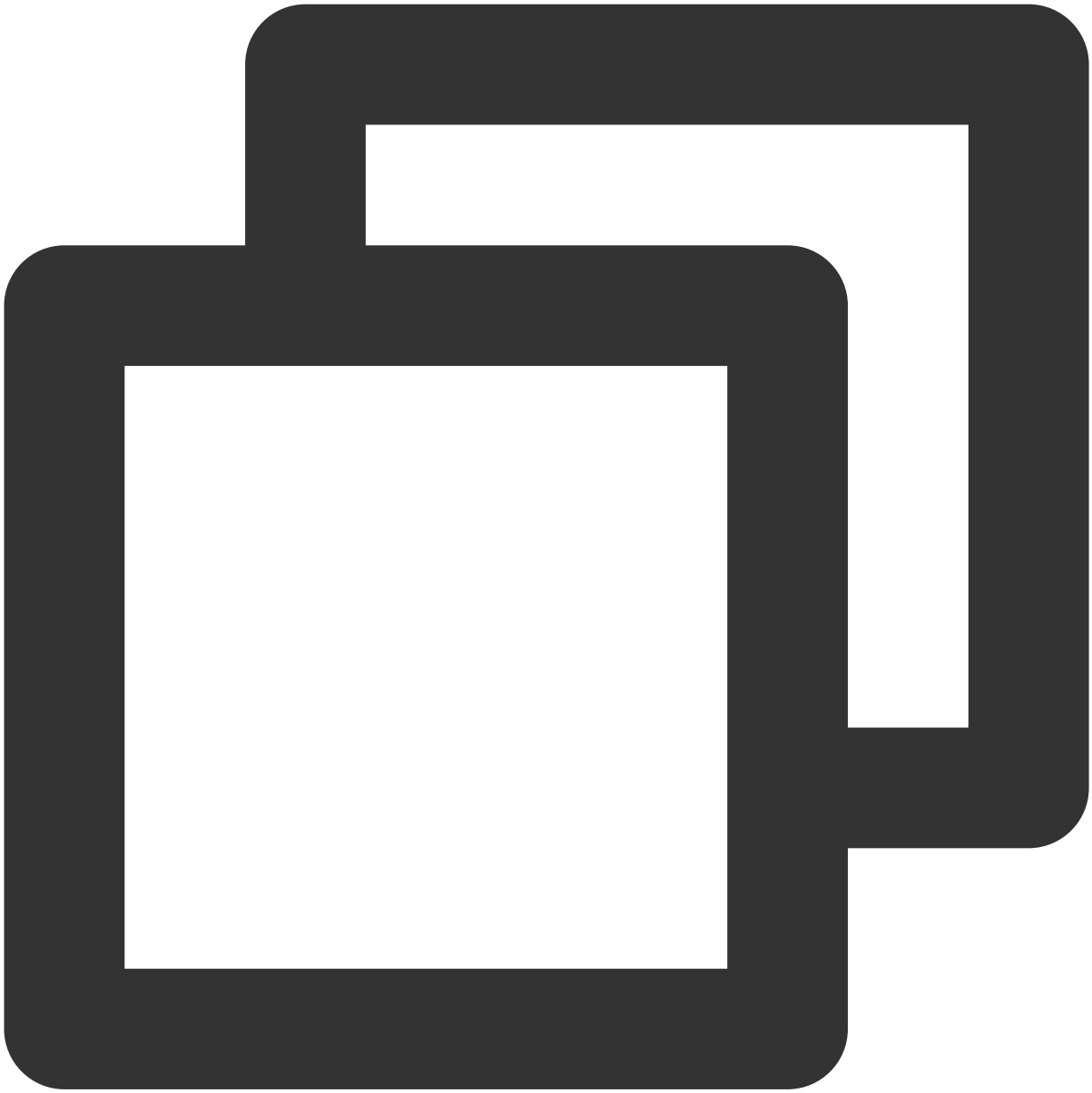
demo 中主进程的目录为 `src/app/main.js` ，渲染进程目录为 `src/client` 。如运行过程出现问题，可优先通过常见问题查找解决。

步骤4：自实现

安装 Electron SDK

使用如下命令，安装 Electron SDK最新版本

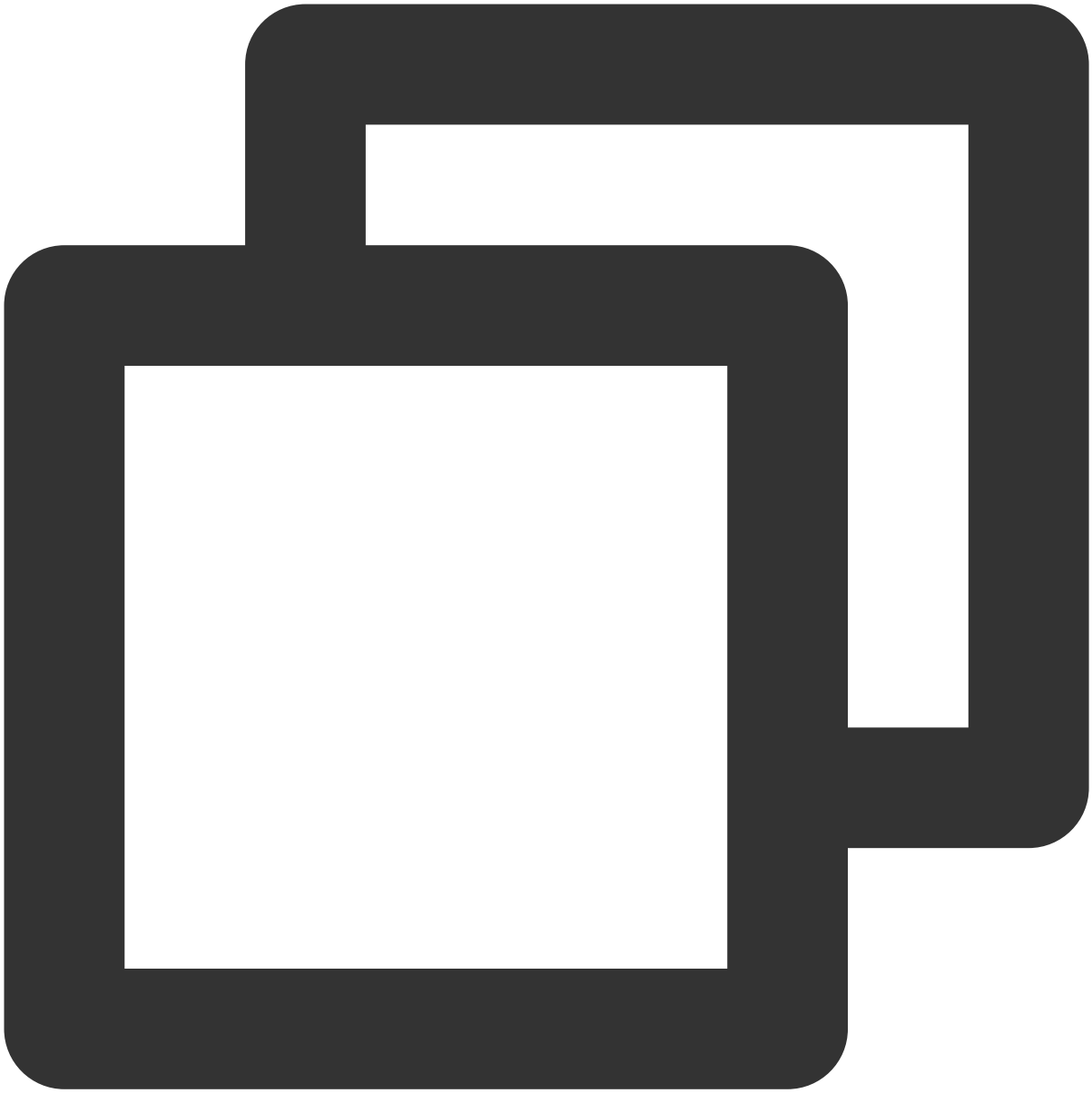
在命令行执行：



```
npm install im_electron_sdk
```

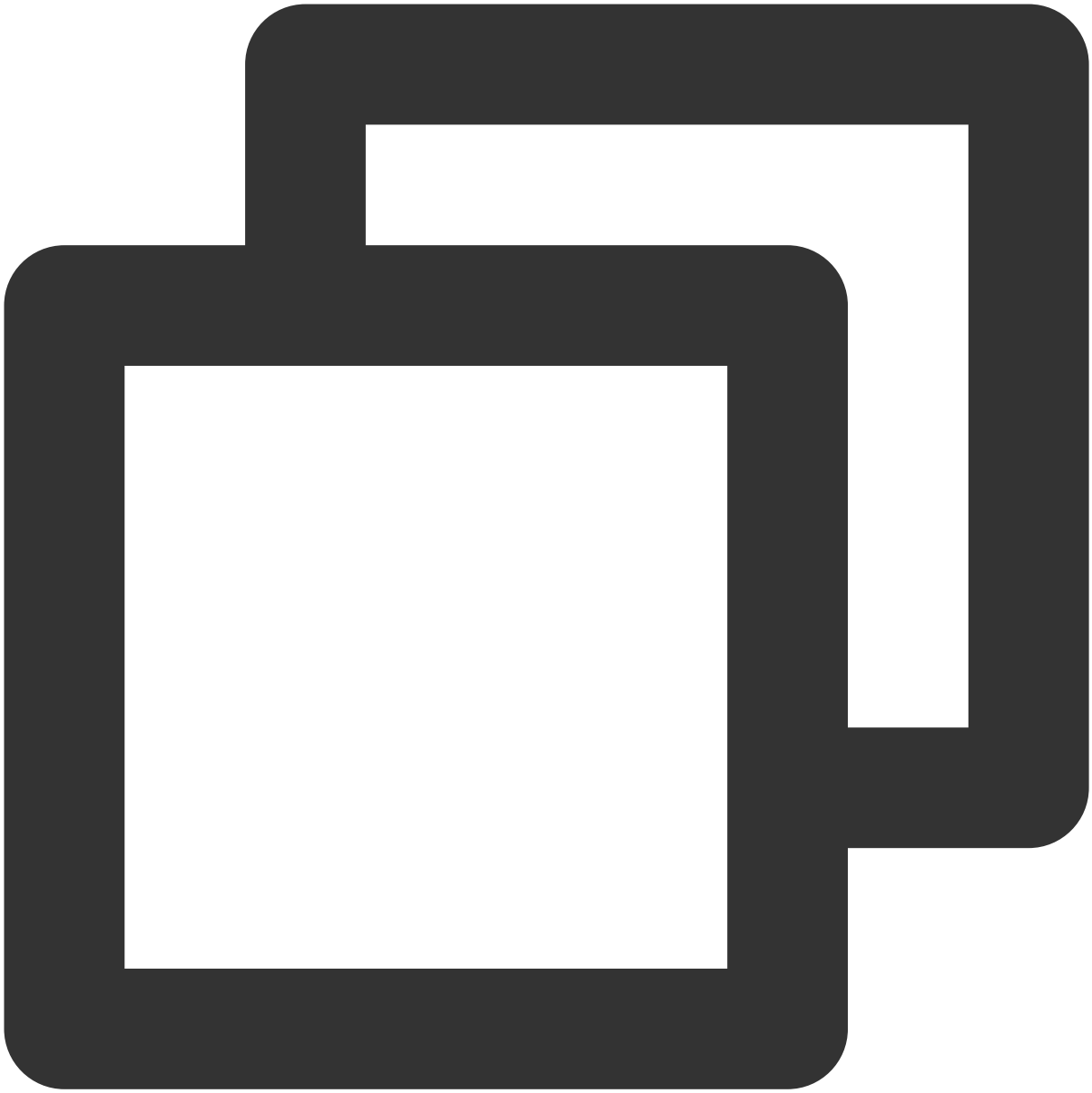
完成 SDK 初始化

1. 在 `TimMain` 中传入您的 `sdkAppID` 。



```
// 主进程
const TimMain = require('im_electron_sdk/dist/main')

const sdkappid = 0;// 可以去腾讯云即时通信IM控制台申请
const tim = new TimMain({
  sdkappid:sdkappid
})
```

2. 调用 `TIMInit` ，完成 SDK 初始化。



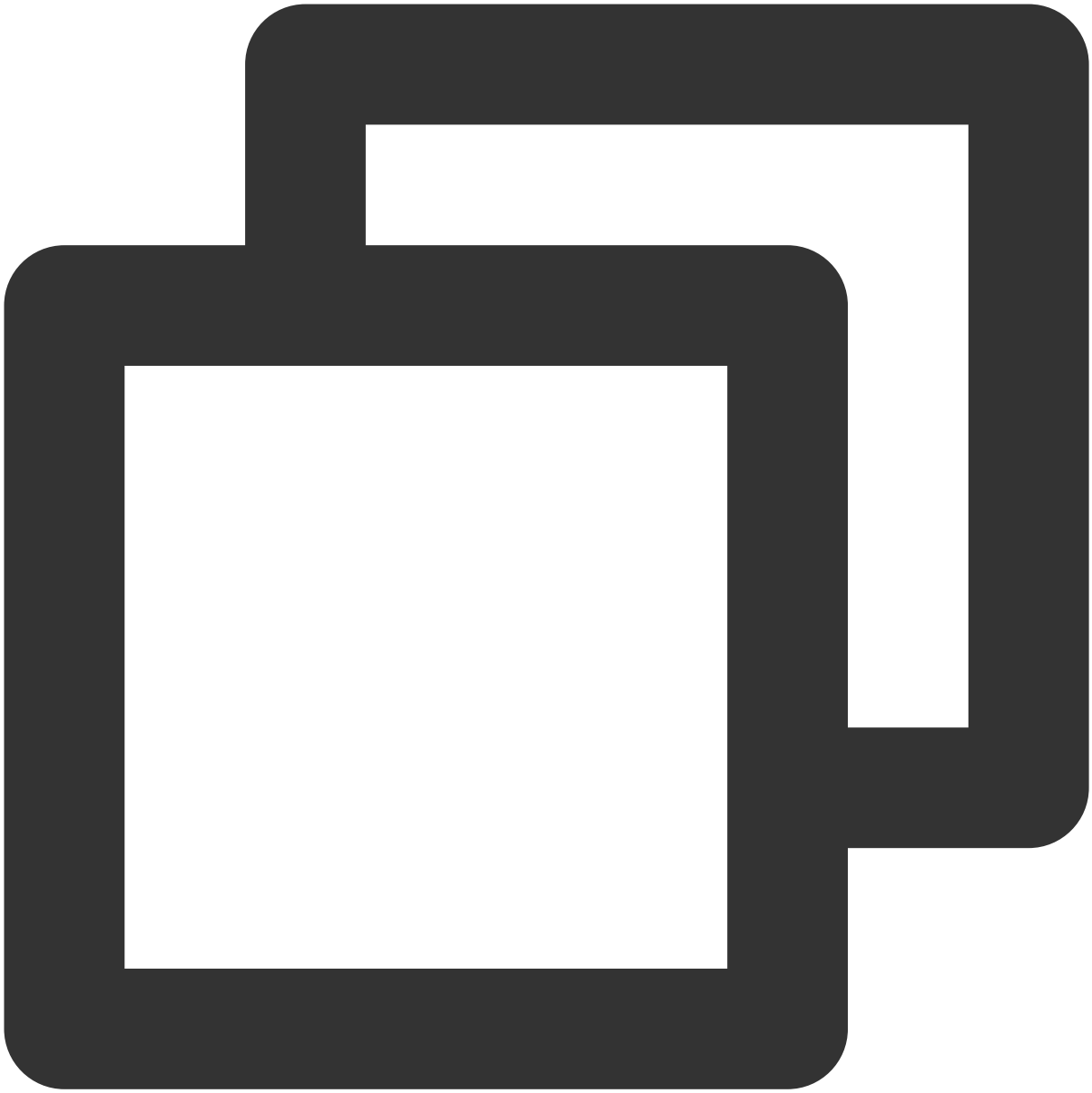
```
//渲染进程
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
// 初始化
timRender.TIMInit()
```

3. 登录测试用户。

此时，您可以使用最开始的时候，在控制台申城的测试账户，完成登录验证。

调用 `timRender.TIMLogin` 方法，登录一个测试用户。

当返回值 `code` 为0时，登录成功。



```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let {code} = await timRender.TIMLogin({
  userID:"userID",
  userSig:"userSig" // 参考userSig生成
})
```

说明：

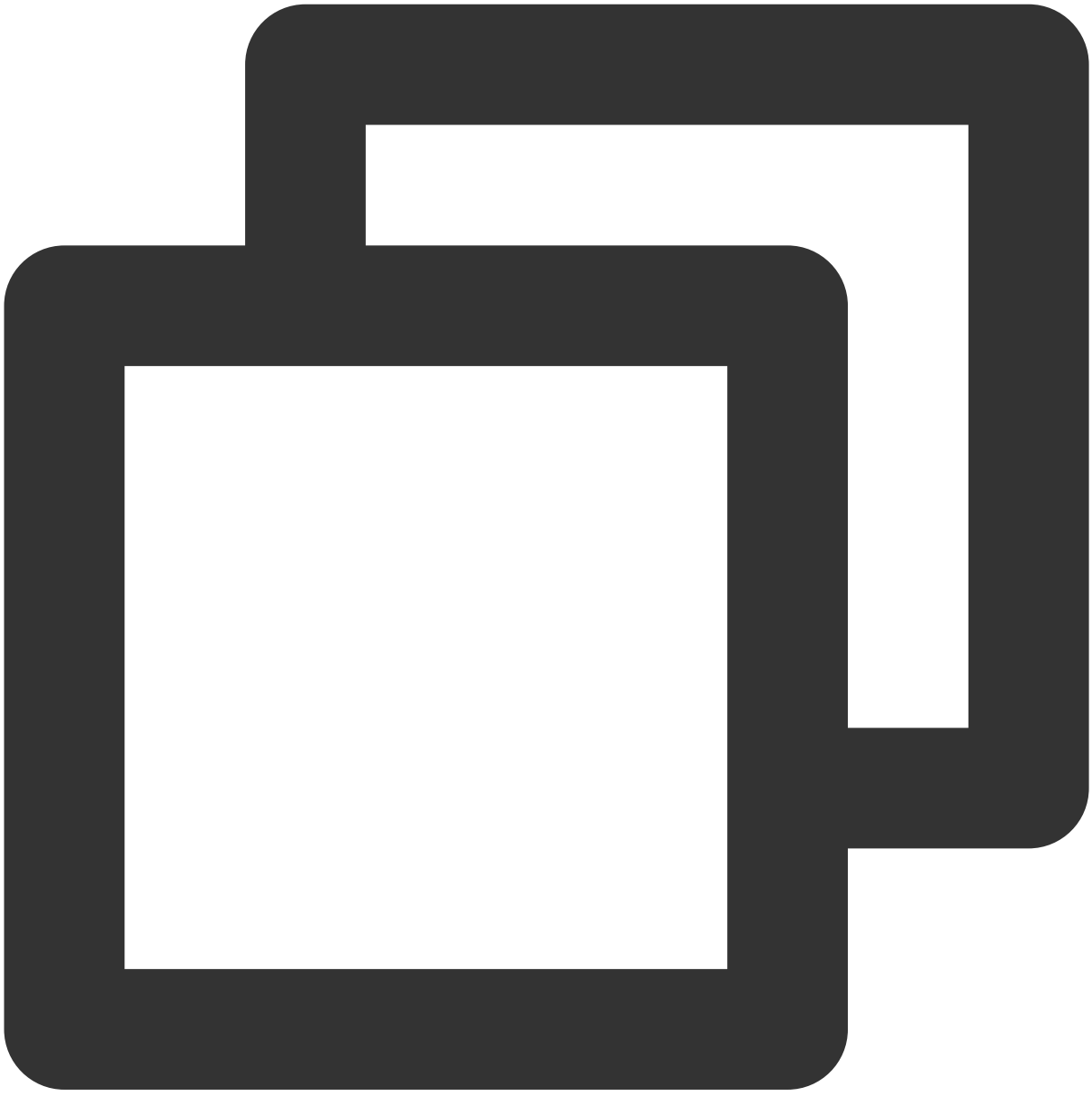
该账户仅限开发测试使用，应用上线前，正确的 `UserSig` 签发方式是将 `UserSig` 的计算代码集成到您的服务端，并提供面向 APP的接口。在需要 `UserSig` 时由您的 APP 向业务服务器发起请求获取动态 `UserSig` 。更

多详情请参见 服务端生成 UserSig。

发送信息

此处以发送文本消息距离， `code` 返回 0 则为消息发送成功。

代码示例：



```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let param:MsgSendMessageParamsV2 = {    // param of TIMMsgSendMessage
    conv_id: "conv_id",
    conv_type: 1,
    params: {
```

```
        message_elem_array: [{
            elem_type: 1,
            text_elem_content:'Hello Tencent!',

        }],
        message_sender: "senderID",
    },
    callback: (data) => {}
  }
let {code} = await timRender.TIMMsgSendMessageV2(param);
```

说明：

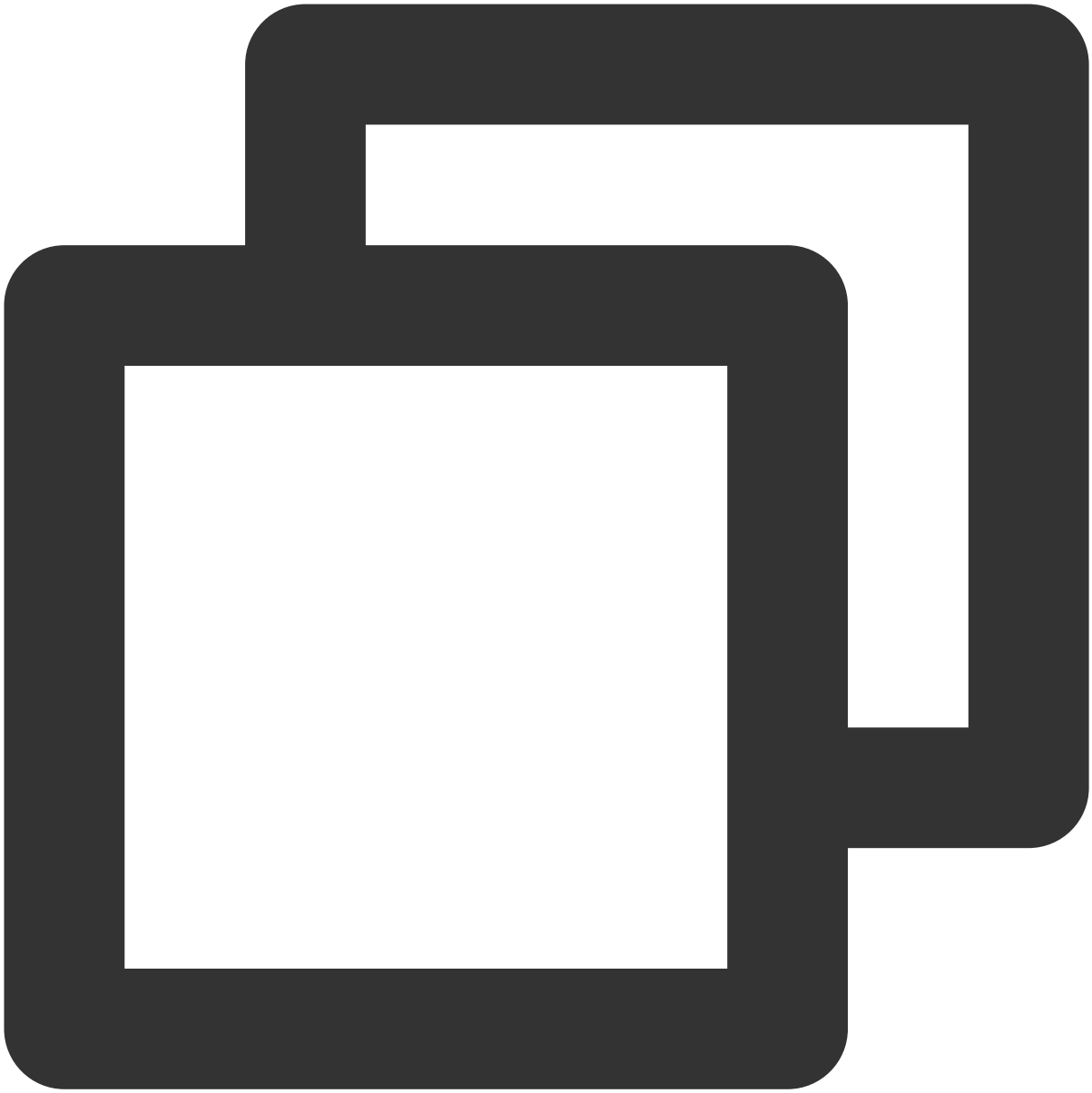
如果发送失败，可能是由于您的 sdkAppID 不支持陌生人发送消息，您可至控制台开启，用于测试。请点击此链接，关闭好友关系链检查。

获取会话列表

在上一个步骤中，完成发送测试消息，现在可登录另一个测试账户，拉取会话列表。

常见应用场景为：

在启动应用程序后立即获取会话列表，然后监听长链接以实时更新会话列表的变化。



```
let param:getConvList = {
            userData:userData,
        }
let data:commonResult<convInfo[]> = await timRenderInstance.TIMConvGetConvList(para
```

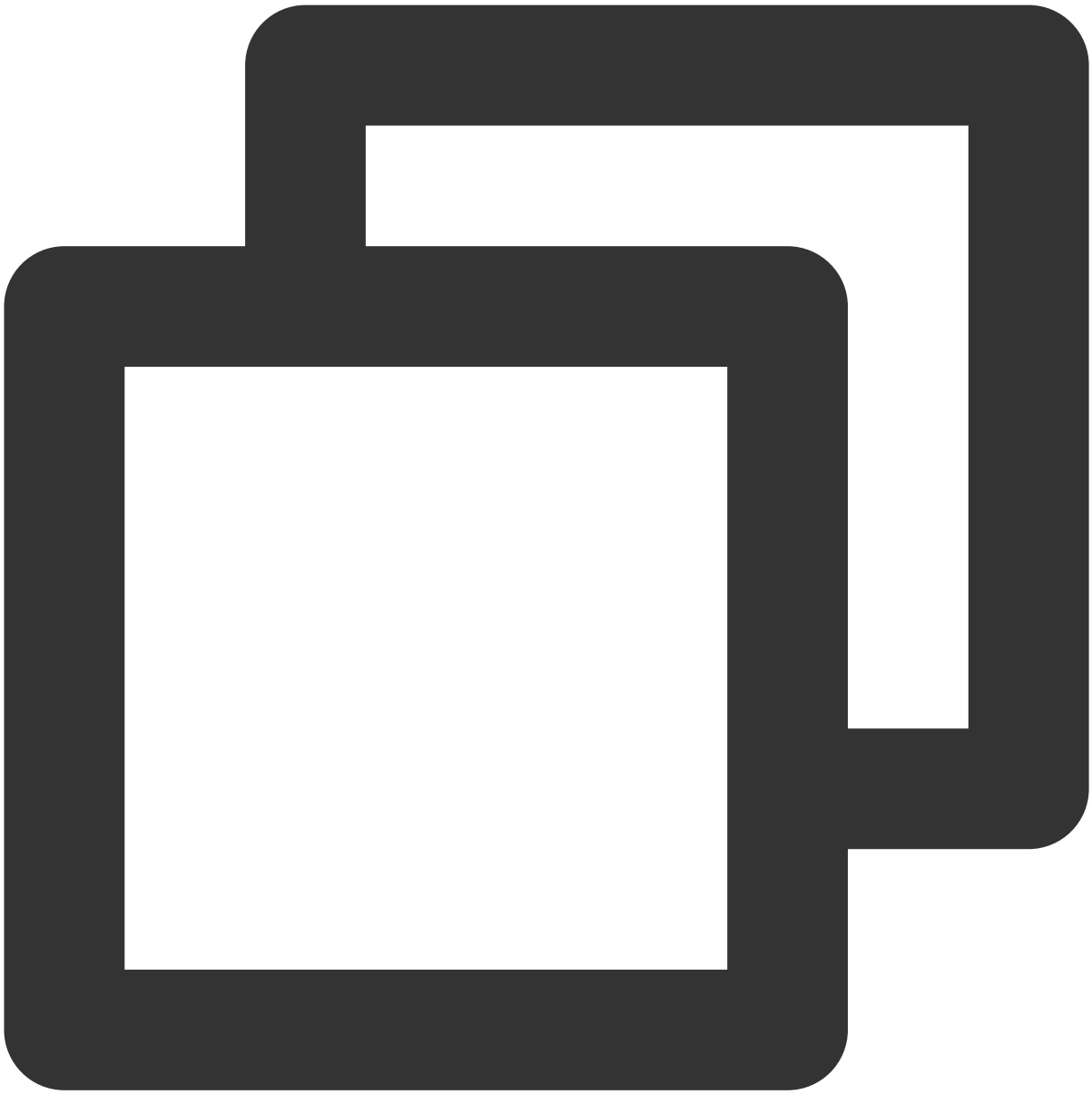
此时，您可以看到您在上一步中，使用另一个测试账号发来的消息的会话。

接收消息

常见应用场景为：

1. 界面进入新的会话后，首先一次性请求一定数量的历史消息，用于展示历史消息列表。
2. 监听长链接，实时接收新的消息，将其添加进历史消息列表中。

一次性请求历史消息列表

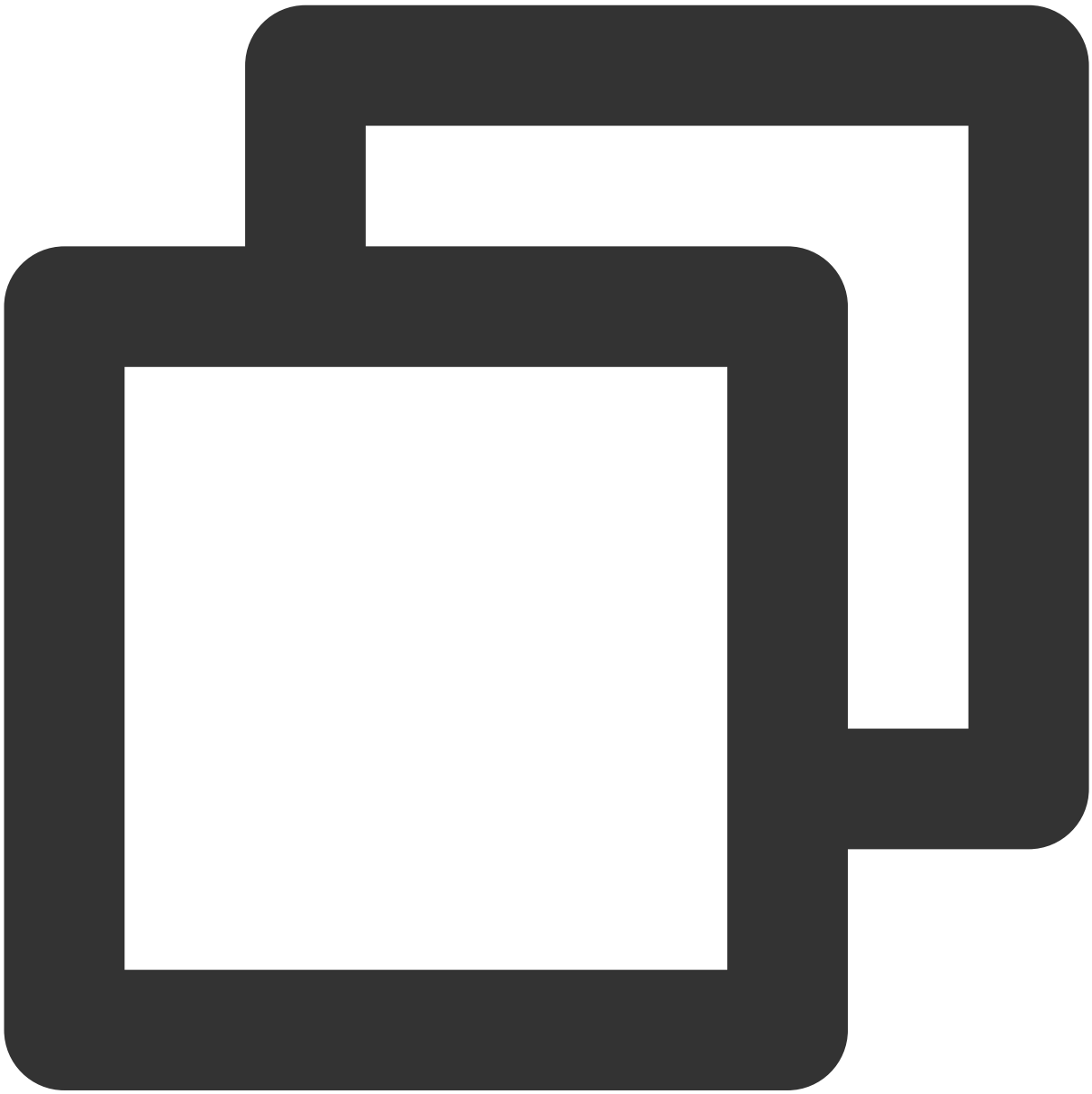


```
let param:MsgGetMsgListParams = {
        conv_id: conv_id,
        conv_type: conv_type,
        params: {
            msg_getmsglist_param_last_msg: msg,
            msg_getmsglist_param_count: 20,
            msg_getmsglist_param_is_remble: true,
        },
        user_data: user_data
```

```
    }
    let msgList:commonResult<Json_value_msg[]> = await timRenderInstance.TIMMsgGetM
```

监听实时获取新消息

绑定 callback 示例代码如下：



```
let param : TIMRecvNewMsgCallbackParams = {
            callback: (...args)=>{},
            user_data: user_data
        }
timRenderInstance.TIMAddRecvNewMsgCallback(param);
```

此时，您已基本完成 IM 模块开发，可以发送接收消息，也可以进入不同的会话。
您可以继续完成 群组，用户资料，关系链，离线推送，本地搜索 等相关功能开发。
详情可查看 API 文档。

常见问题

支持哪些平台？

目前支持 Macos 和 Windows 两个平台。

错误码如何查询？

IM SDK 的 API 层面错误码，请查看 错误码。

安装开发环境问题，出现 `npm ERR! gyp ERR! stack TypeError: Cannot assign to read only property 'cflags' of object '#<Object>'` 错误如何解决？

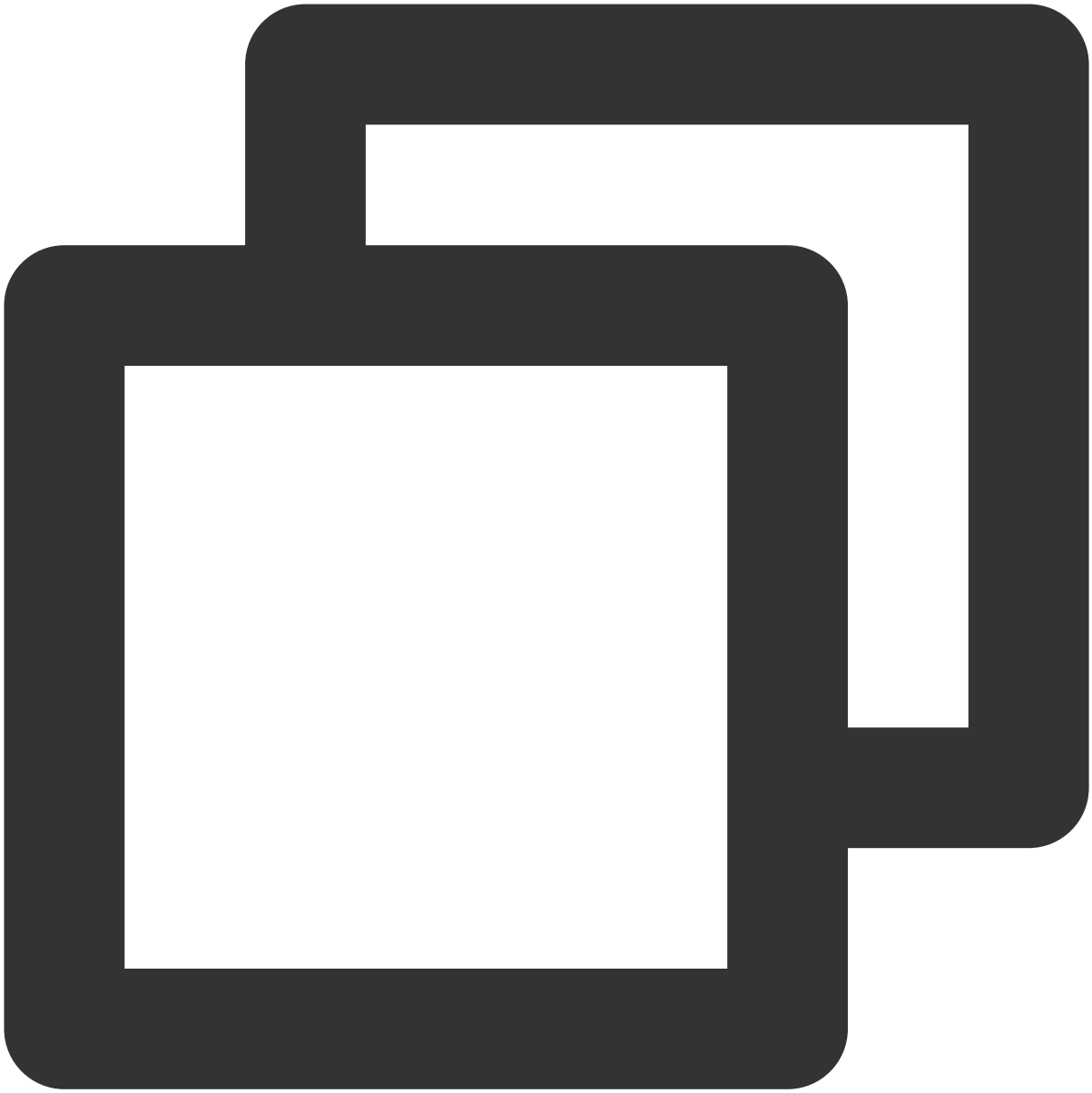
请降低 node 版本，建议使用16.18.1。

安装开发环境问题，出现 `gypgyp ERR!ERR` 错误如何解决？

请参见 gypgyp ERR!ERR!。

执行 `npm install` 出现错误 `npm ERR! Fix the upstream dependency conflict, or retry` ，如何解决？

npmV7之前的版本遇到依赖冲突会忽视依赖冲突，继续进行安装
npmV7版本开始不会自动进行忽略，需要用户手动输入命令
请执行以下命令：



```
npm install --force
```

执行 `npm run start` 出现错误 `Error: error:0308010C:digital envelope routines::unsupported`，如何解决？

请降低node版本，建议使用16.18.1。

Mac 端 Demo 执行 `npm run start` 会出现白屏，如何解决？

Mac 端执行 `npm run start` 会出现白屏，原因是渲染进程的代码还没有 build 完成，主进程打开的3000端口为空页面，当渲染进程代码 build 完成重新刷新窗口后即可解决问题。或者执行 `cd src/client && npm run dev:react` ，`npm run dev:electron` , 分开启动渲染进程和主进程。

`vue-cli-plugin-electron-builder` 构建的项目如何使用 `native modules` ?

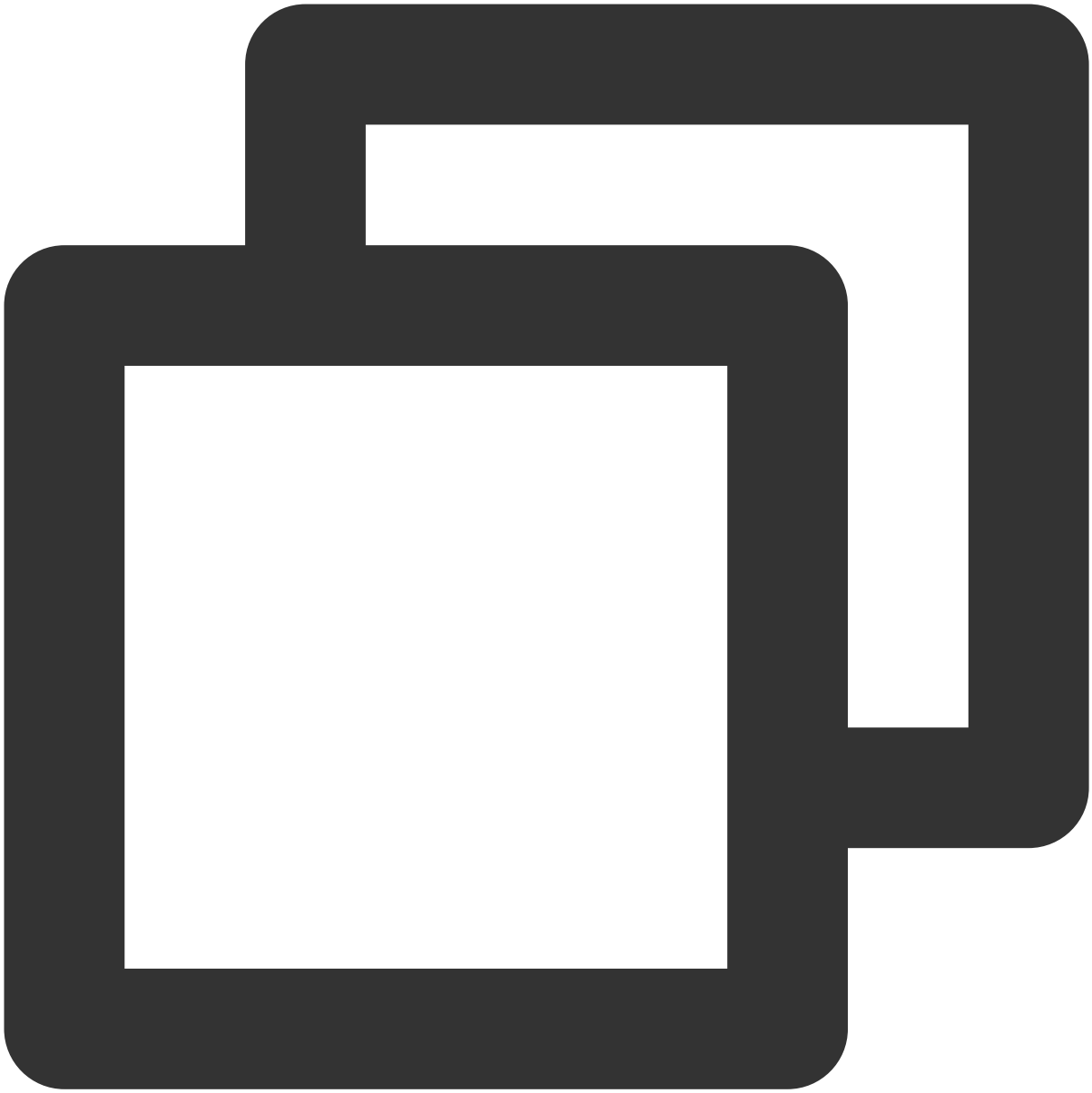
使用 `vue-cli-plugin-electron-builder` 构建的项目使用 `native modules` 请参见 No native build was found for platform = xxx。

用 `webpack` 构建的项目如何使用 `native modules` ?

自己使用webpack 构建的项目使用native modules 请参见 Windows 下常见问题。

出现 `Dynamic Linking Error` ?

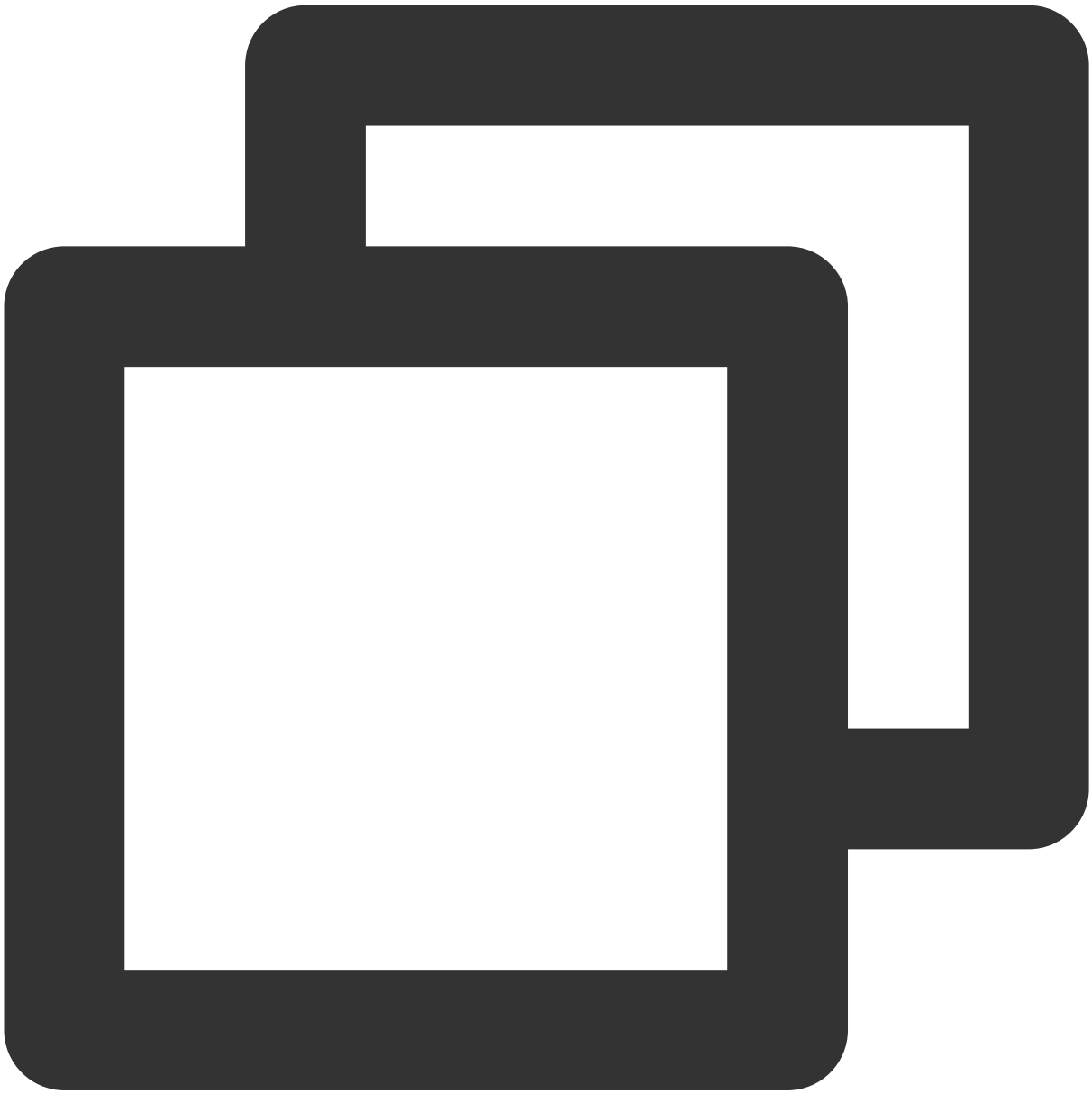
Dynamic Linking Error. electron-builder 配置



```
  extraFiles:[
    {
      "from": "./node_modules/im_electron_sdk/lib/",
      "to": "./Resources",
      "filter": [
        "**/*"
      ]
    }
]
```

使用Electron-vite出现 `__dirname is not defined` ？

由于Electron-vite不支持在渲染进程（renderer) 进行进程间的通信，需要将 IM SDK 写到preload中使用 (主进程的代码正常写到主进程即可)。具体可参考 electron-vite文档。
使用方法相同，可参考文档的实例代码。下面以初始化为例，使用方法如下：



```
// 主进程内容正常写到主进程
// main/index.ts （示例路径）
const TimMain = require('im_electron_sdk/dist/main')
const sdkappid = 0;
const tim = new TimMain({
```

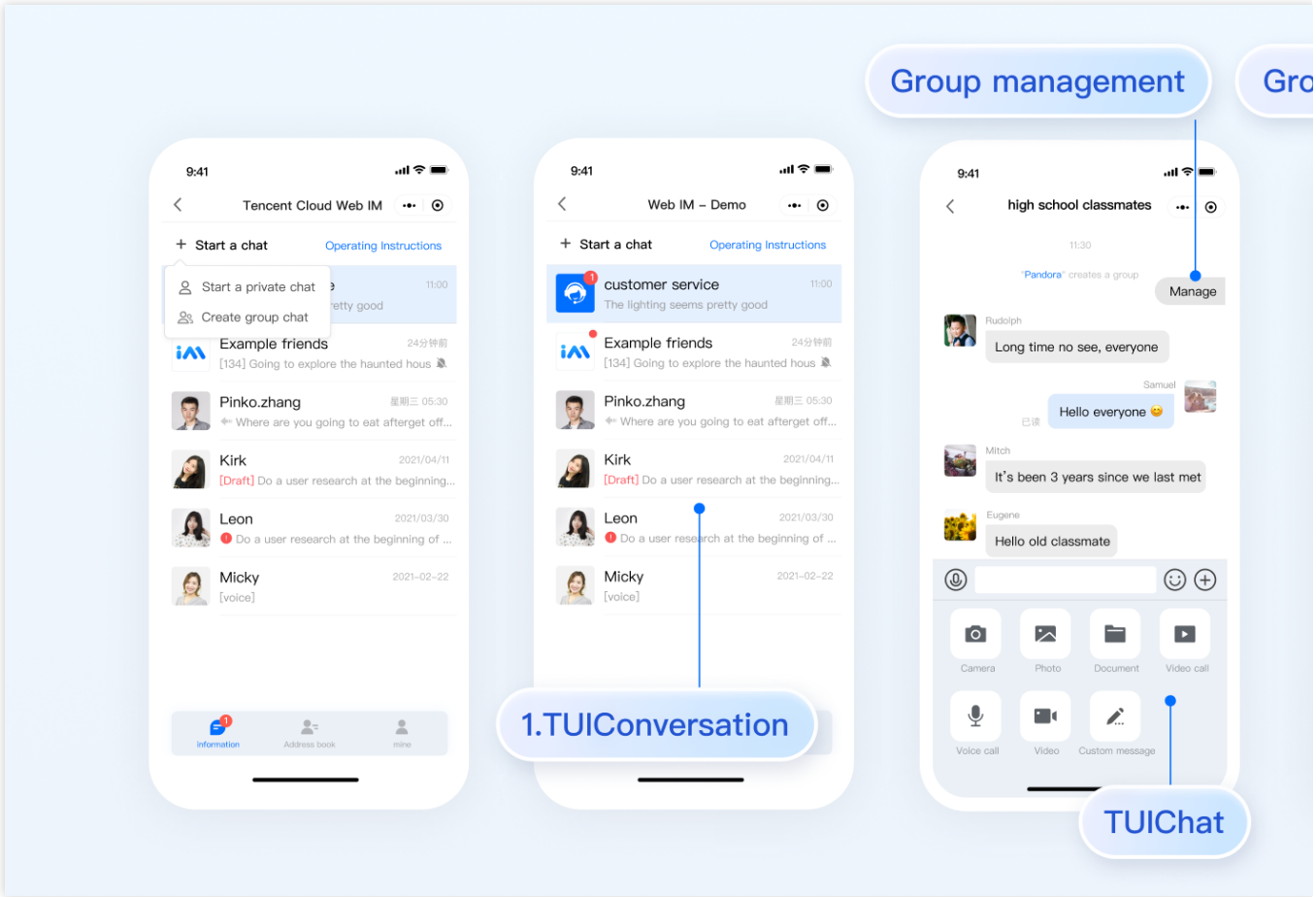
```
  sdkappid:sdkappid
})
// 在 preload 中使用im sdk
// preload/index.ts （示例路径）
import TimRender from 'im_electron_sdk/dist/renderer'
const timRender = new TimRender();
```

uniapp

最近更新时间：2024-05-13 16:43:20

chat-uikit-uniapp 介绍

chat-uikit-uniapp （vue2 /vue3）是基于腾讯云 Chat SDK 的一款 uniapp UI 组件库，它提供了一些通用的 UI 组件，包含会话、聊天、群组等功能。基于这些精心设计的 UI 组件，您可以快速构建优雅的、可靠的、可扩展的 Chat 应用。 chat-uikit-uniapp 界面效果如下图所示：



支持平台

Android
iOS
微信小程序

H5

开发环境要求

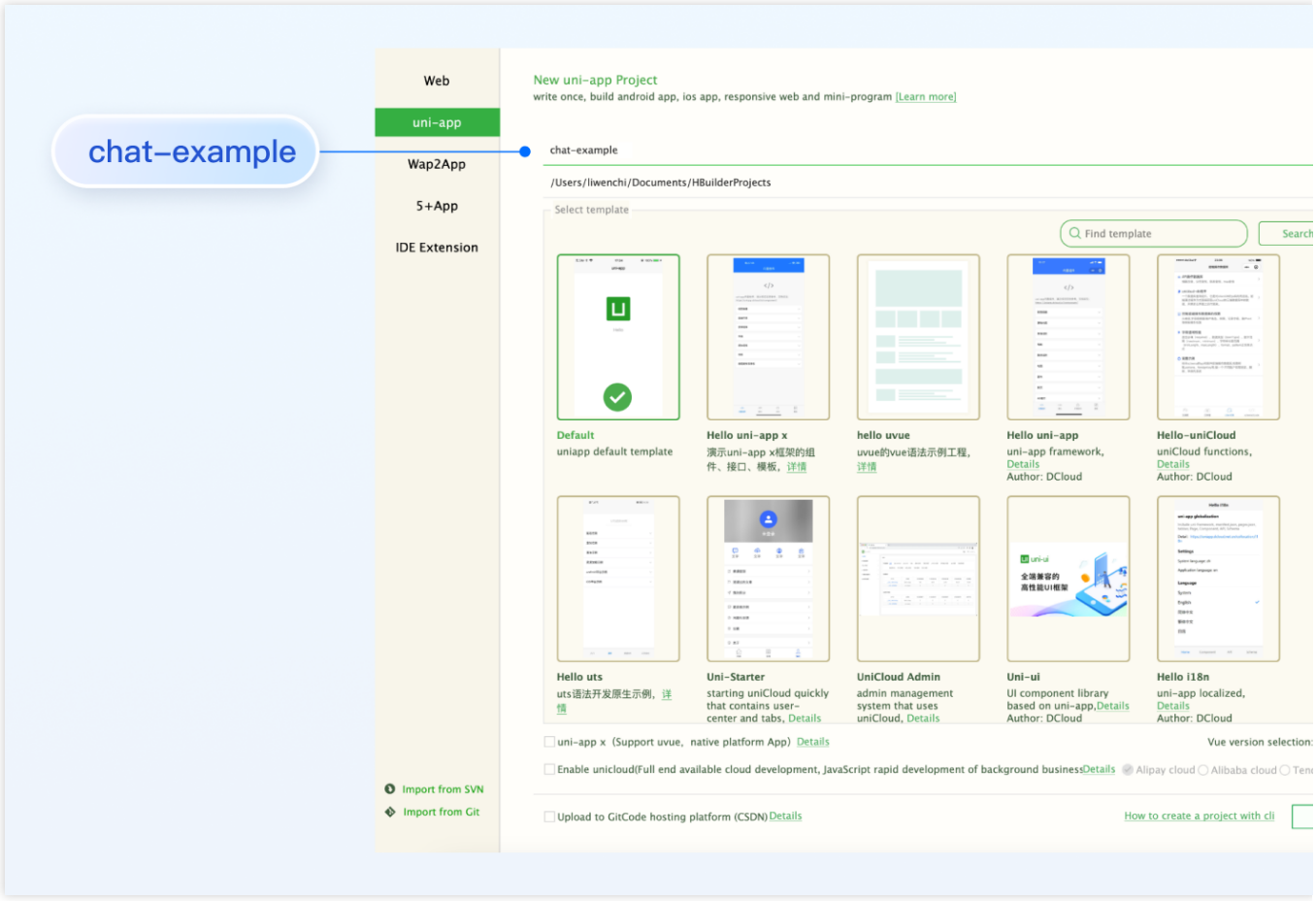
HBuilderX （HBuilderX 版本 >= 3.8.4.20230531）或者升级到新版本
Vue2 / Vue3
sass（sass-loader 版本 ≤ 10.1.1）
node（12.13.0 ≤ node 版本 ≤ 17.0.0, 推荐使用 Node.js 官方 LTS 版本 16.17.0）
npm（版本请与 node 版本匹配）

TUIKit 源码集成

完成以下步骤即可发送您的第一条消息。

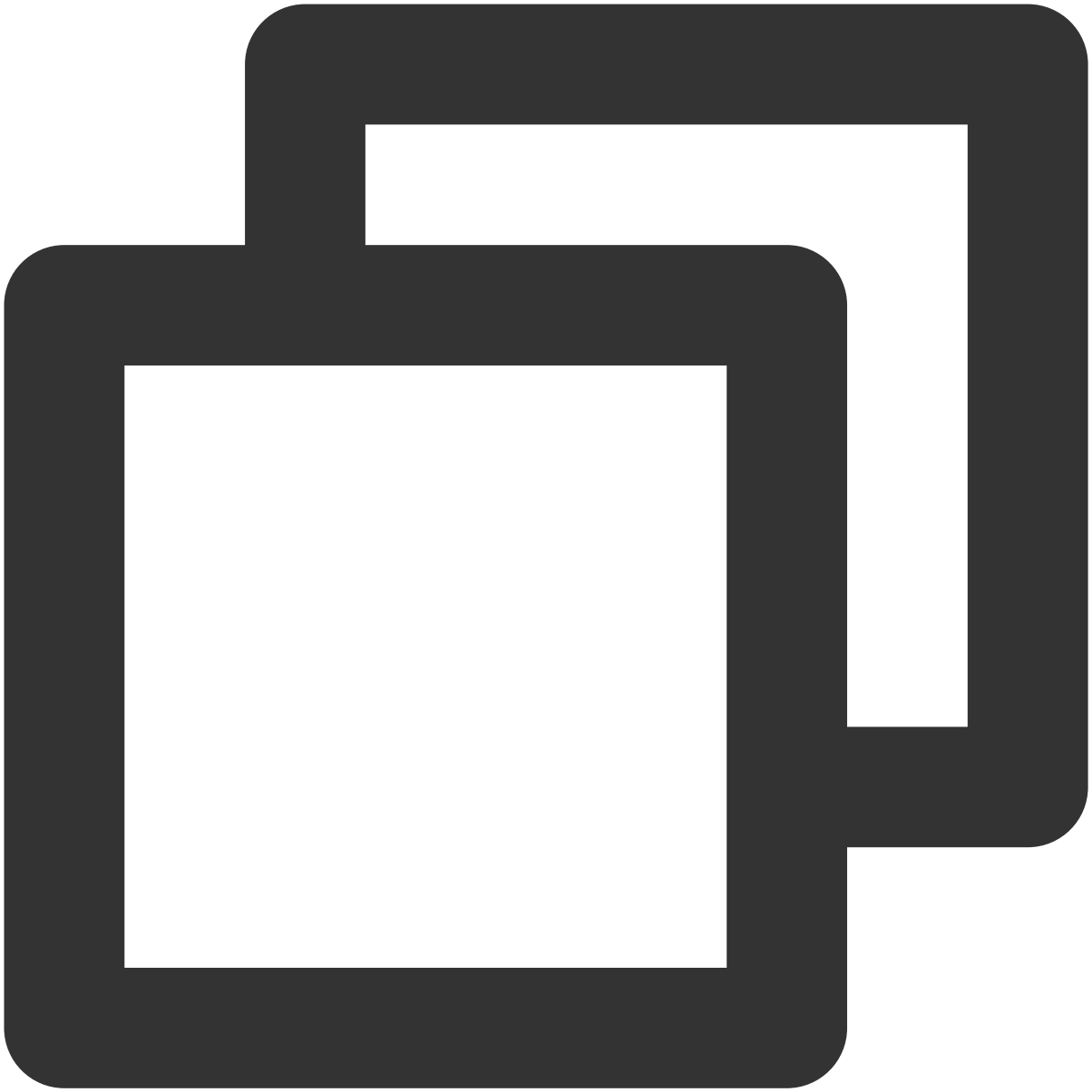
步骤1：创建项目 （已有项目可忽略）

打开 HbuilderX，在菜单栏中选择 “文件-新建-项目”，创建一个名为 `chat-example` 的 uni-app 项目。



步骤2：下载 TUIKit 组件

HBuilderX 不会默认创建 package.json 文件，因此您需要先创建 package.json 文件。请在项目根目录下执行以下命令：



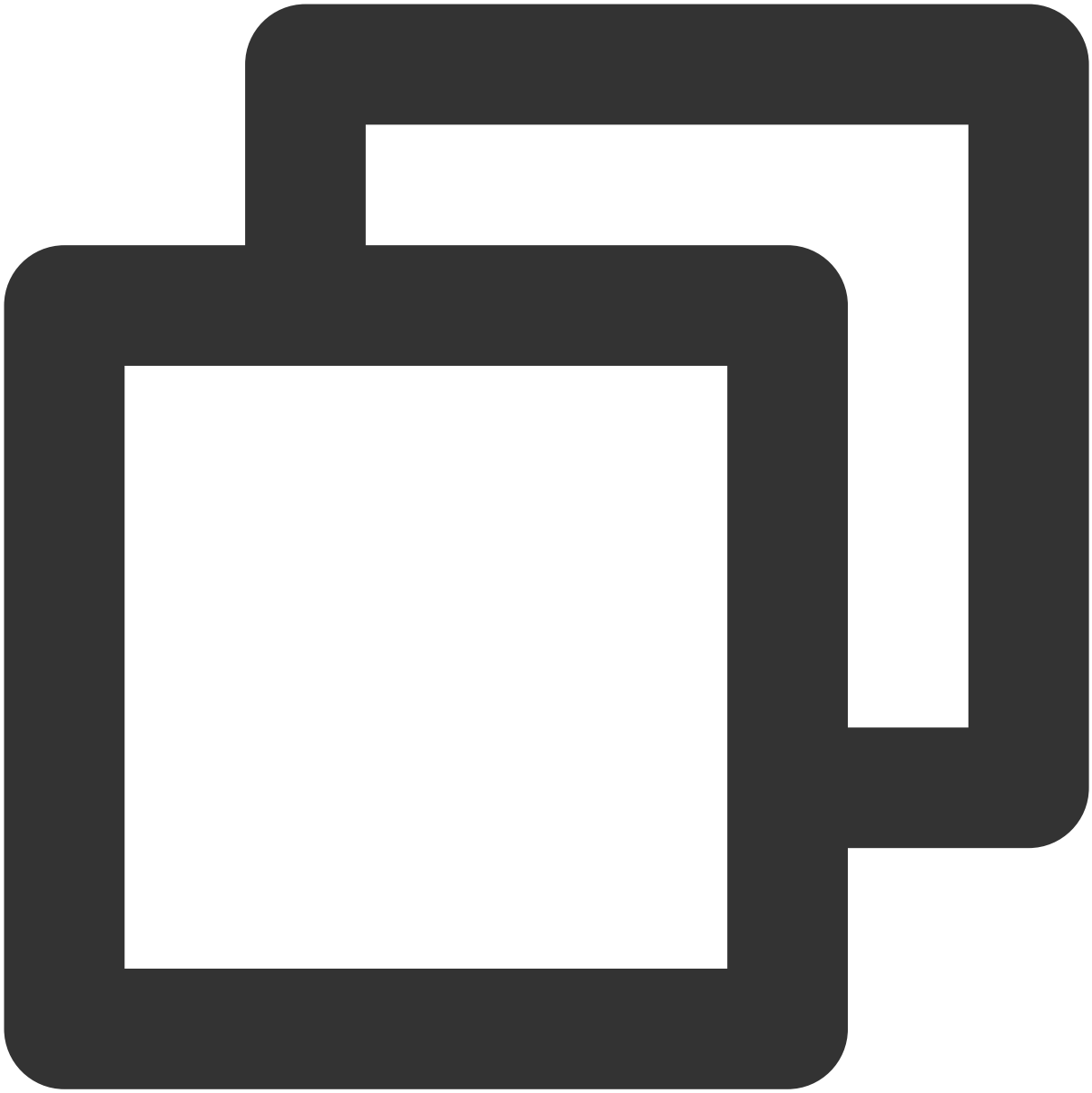
```
npm init -y
```

下载 TUIKit 并拷贝至源码中：

macOS 端

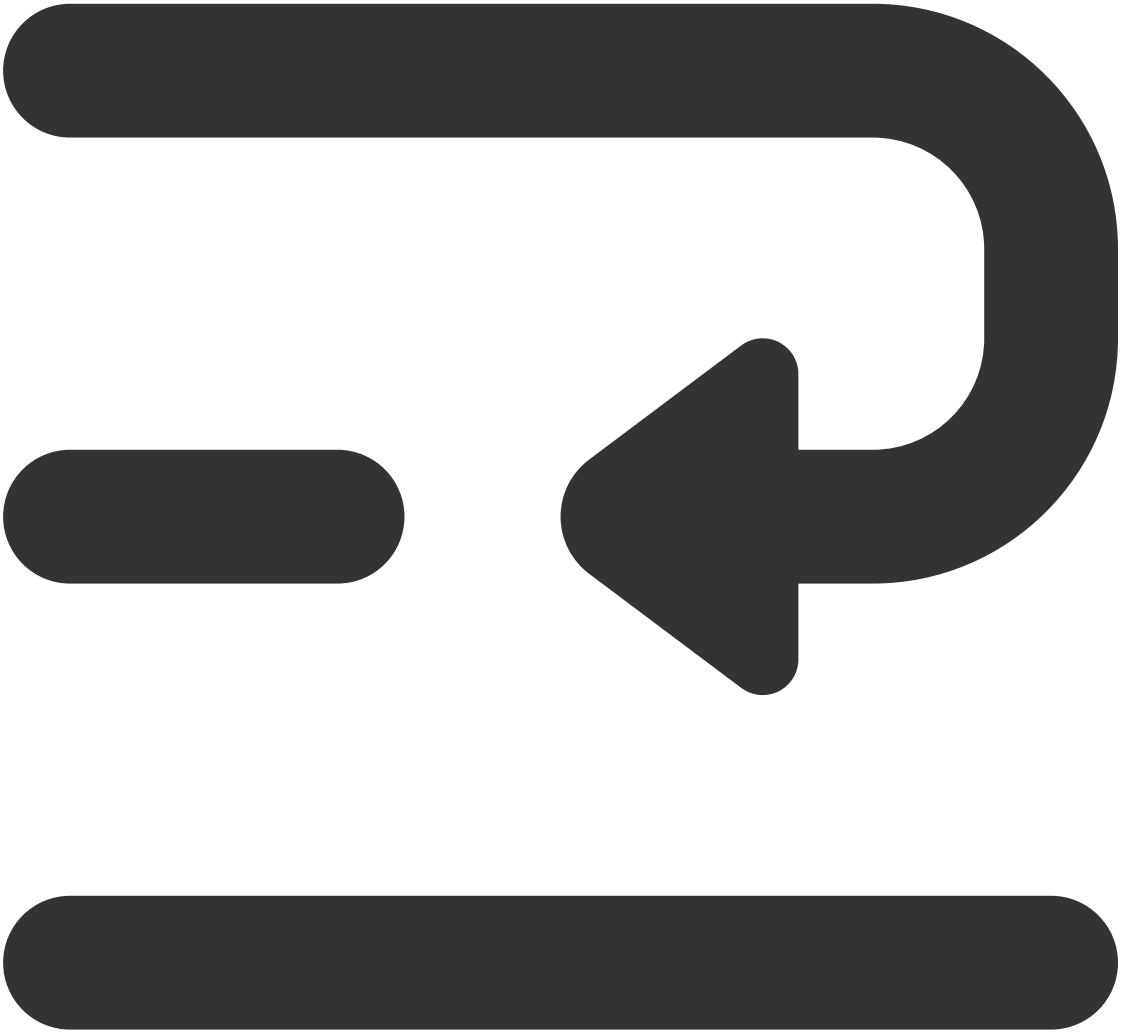
Windows 端

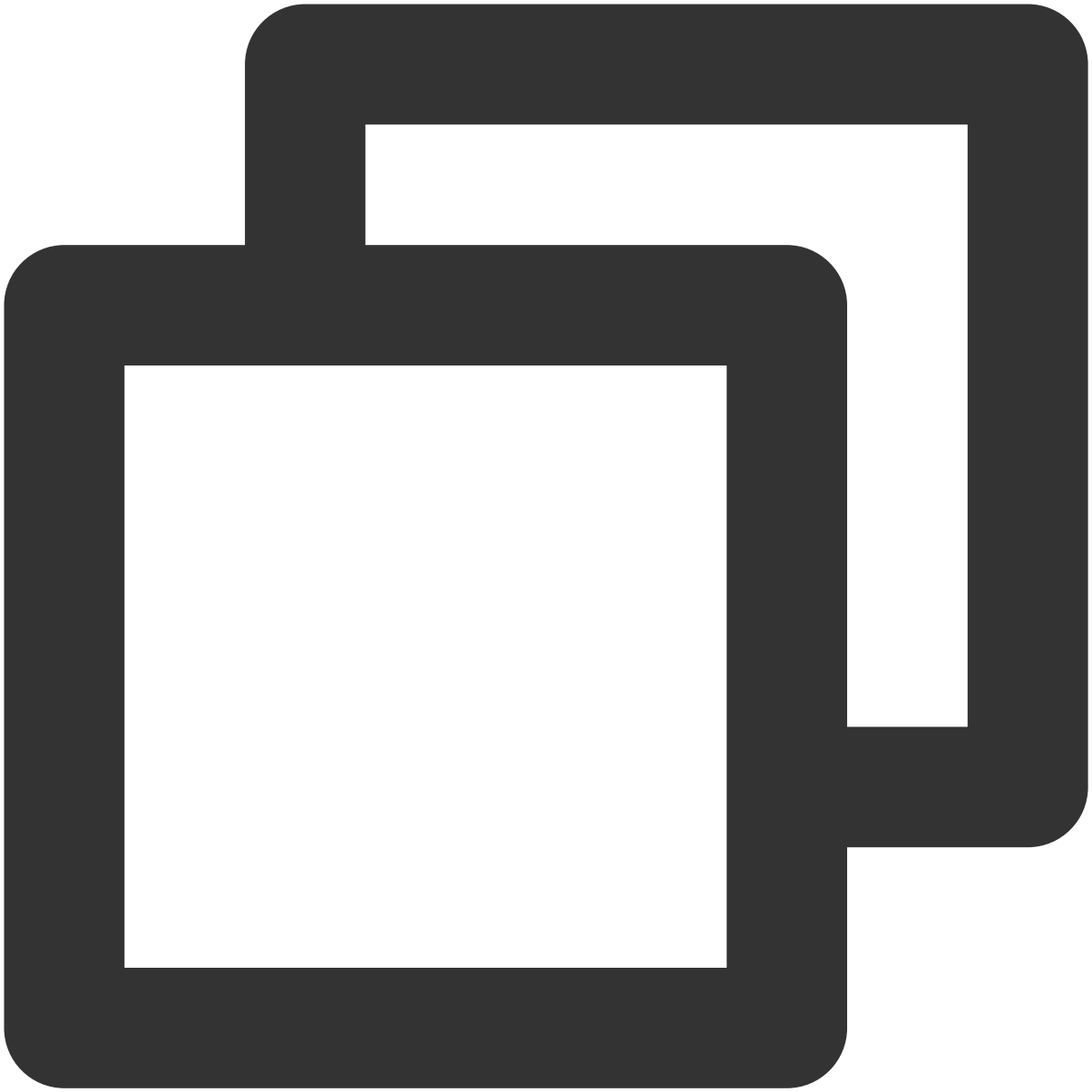
通过 npm 方式下载 TUIKit 组件：



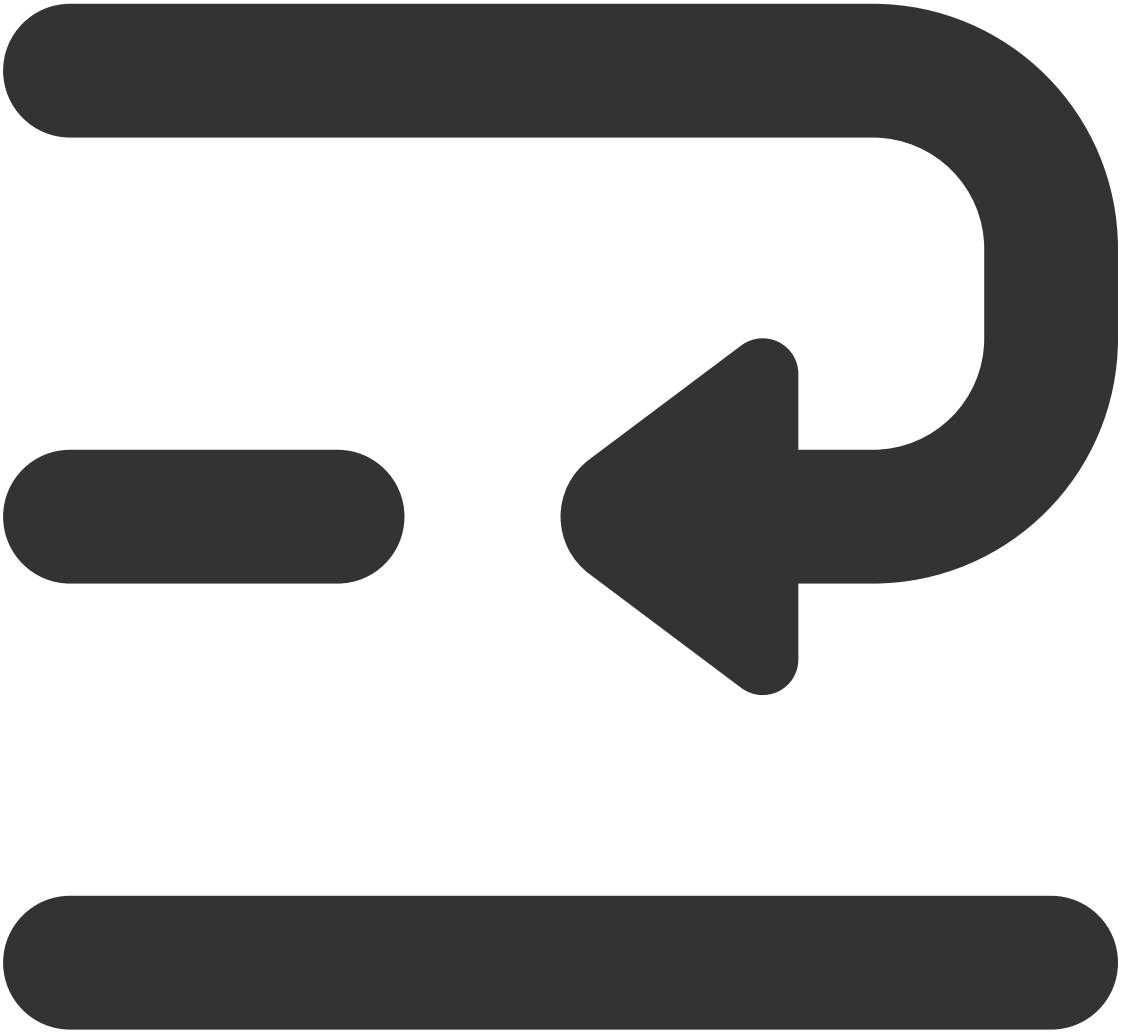
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

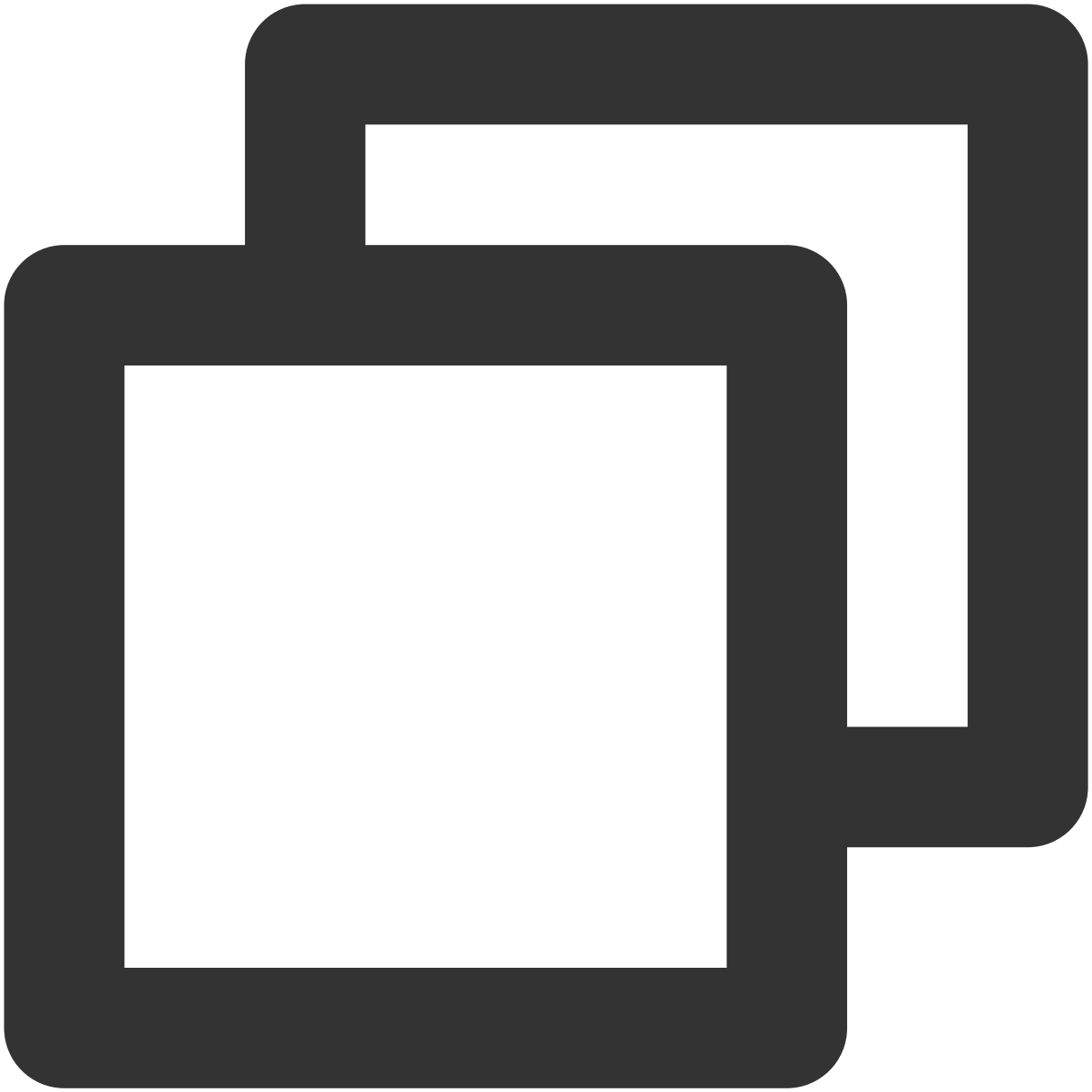
为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 pages 目录下，请在自己的项目根目录下执行以下命令：





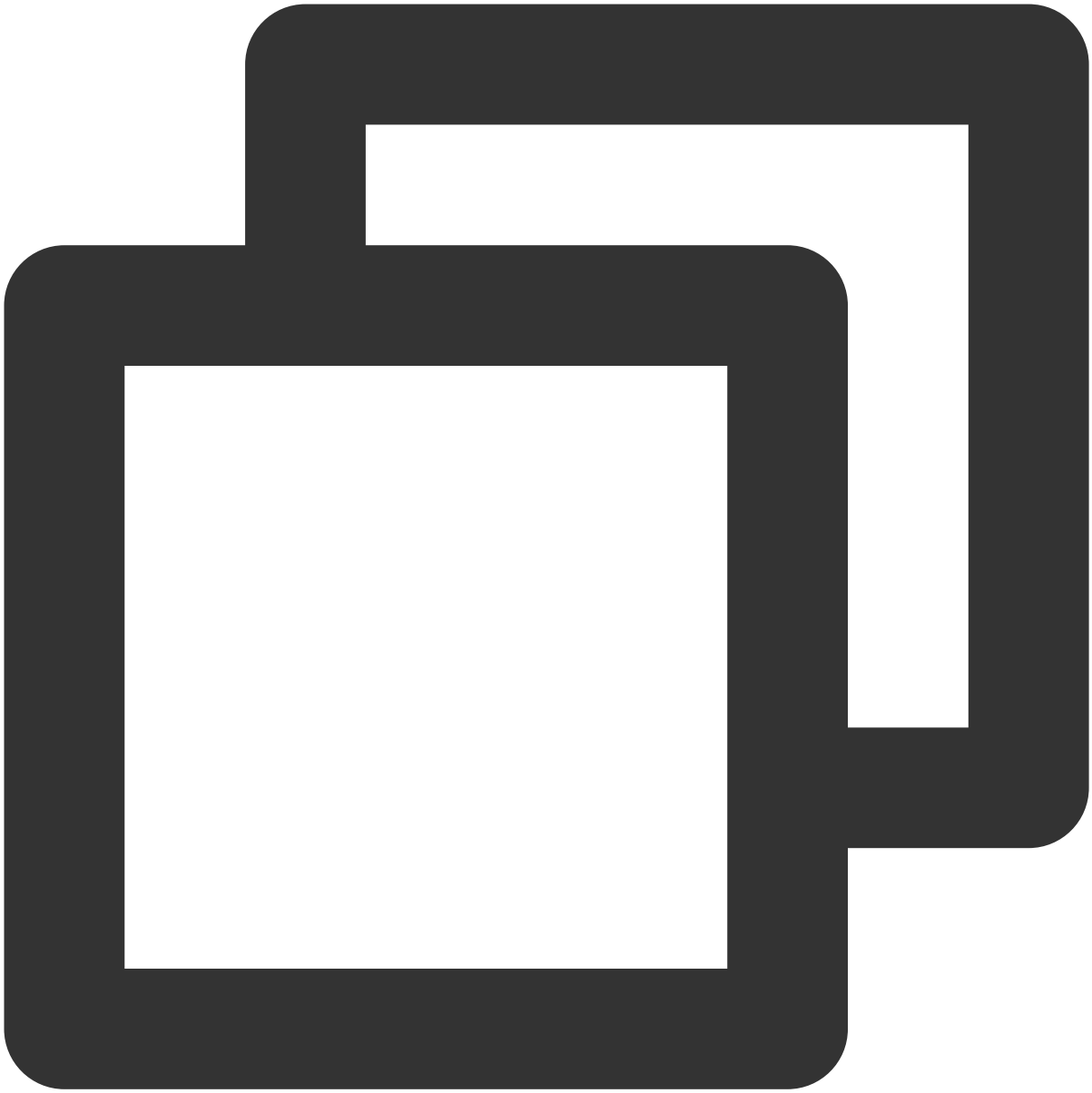
```
mkdir -p ./TUIKit && rsync -av --exclude={'node_modules','package.json','excluded-l
```





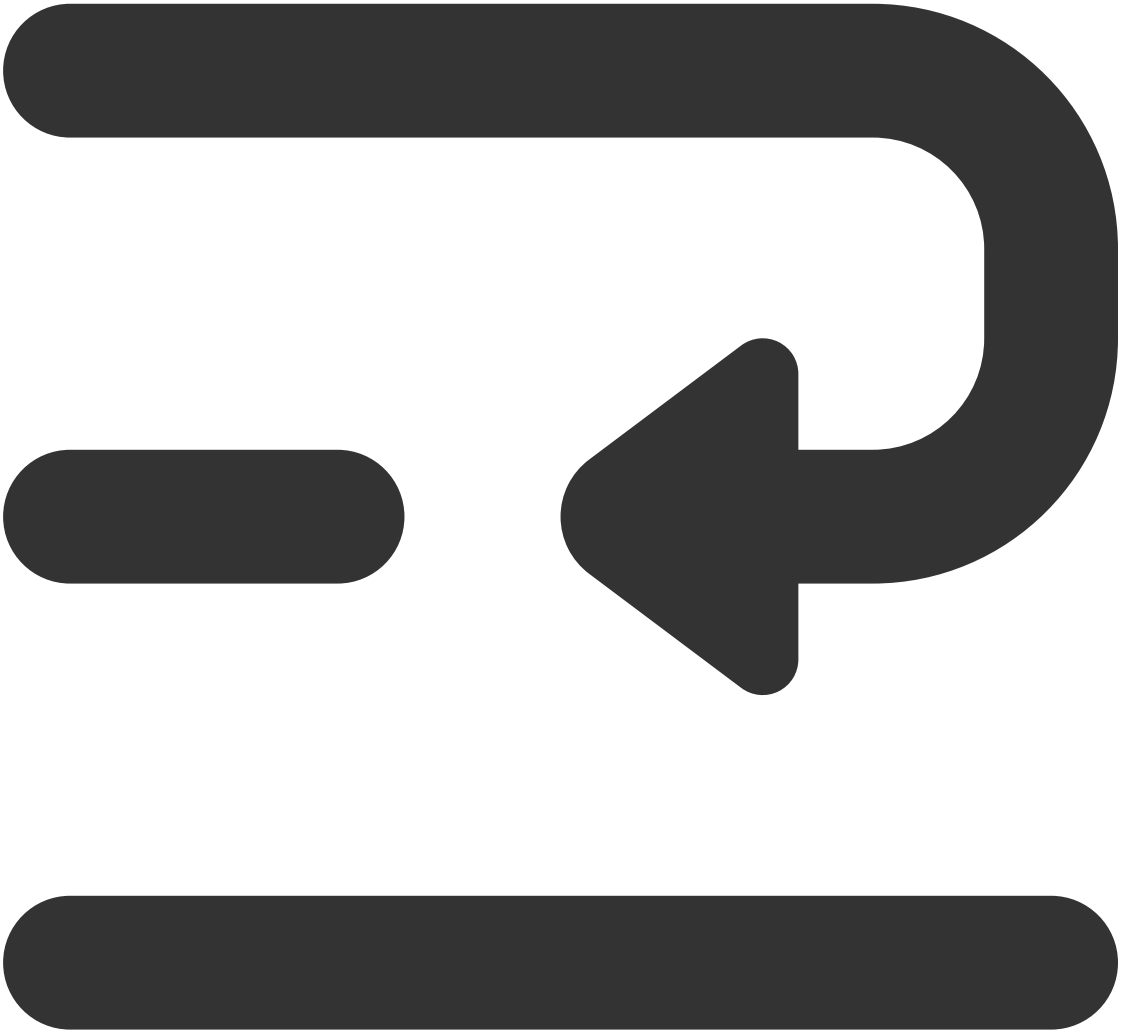
```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av ./node_modules/@tencentc
```

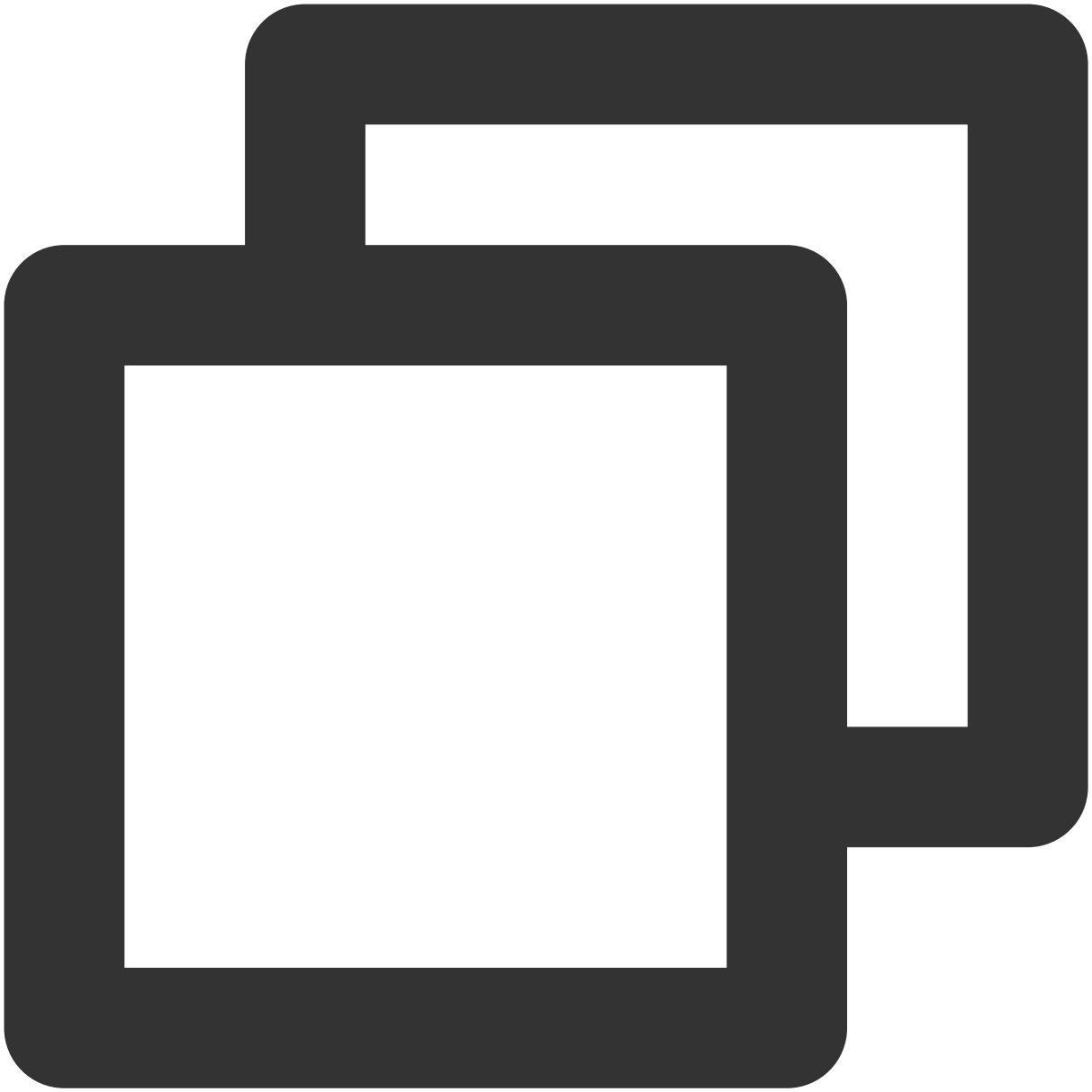
通过 npm 方式下载 TUIKit 组件：



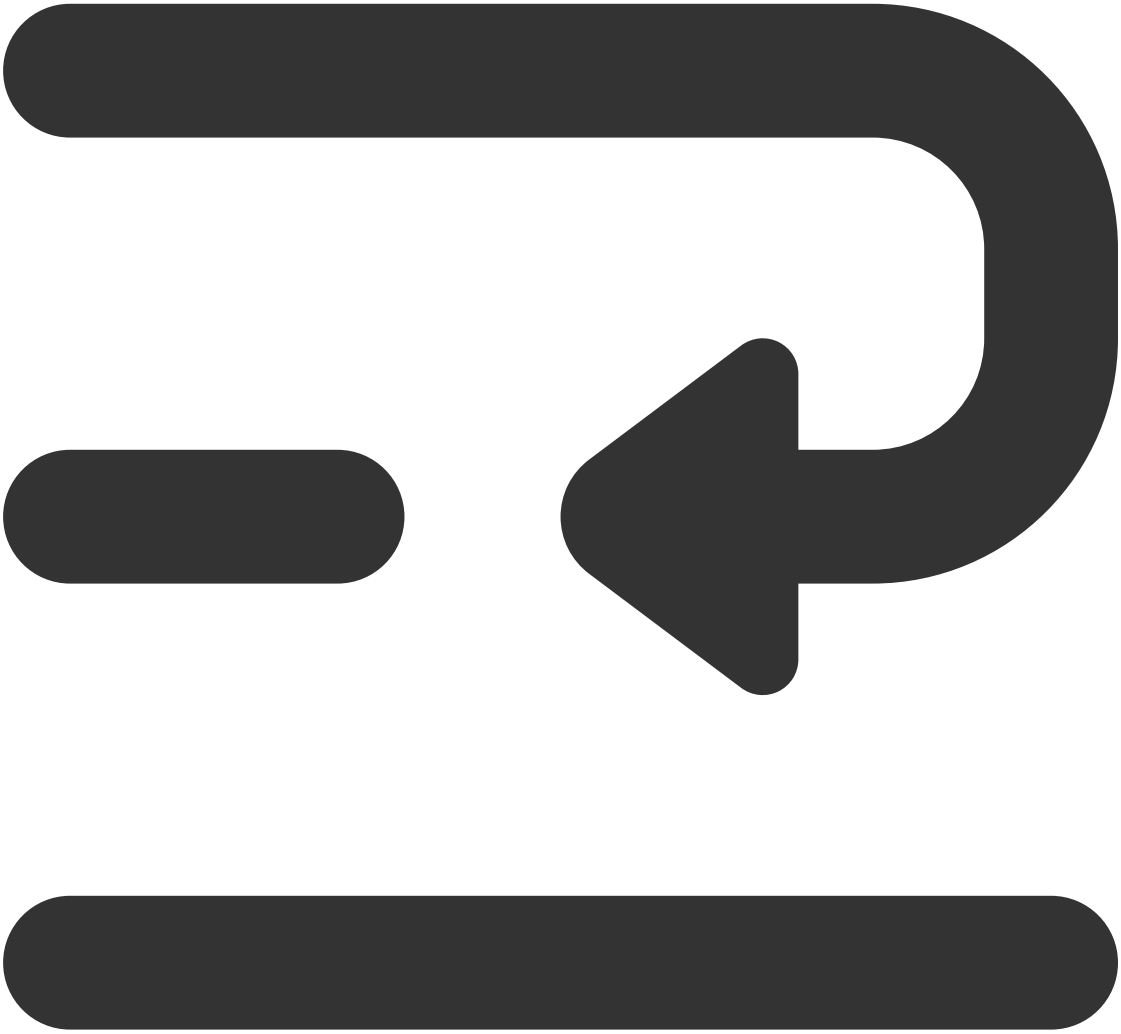
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

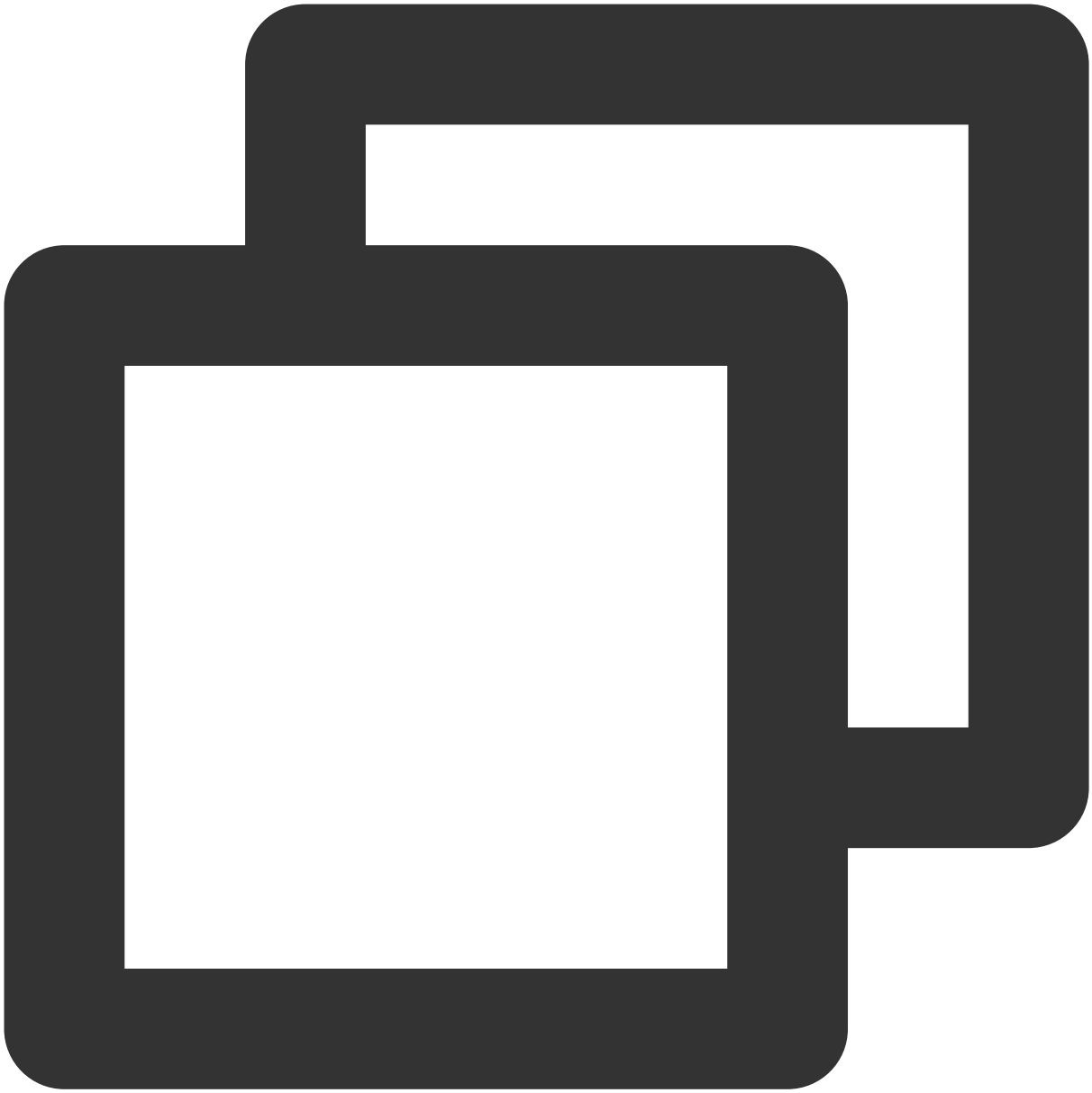
为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 pages 目录下，请在自己的项目根目录下执行以下命令：





```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-uniapp .\\TUIKit /i /e /exclude:.\
```



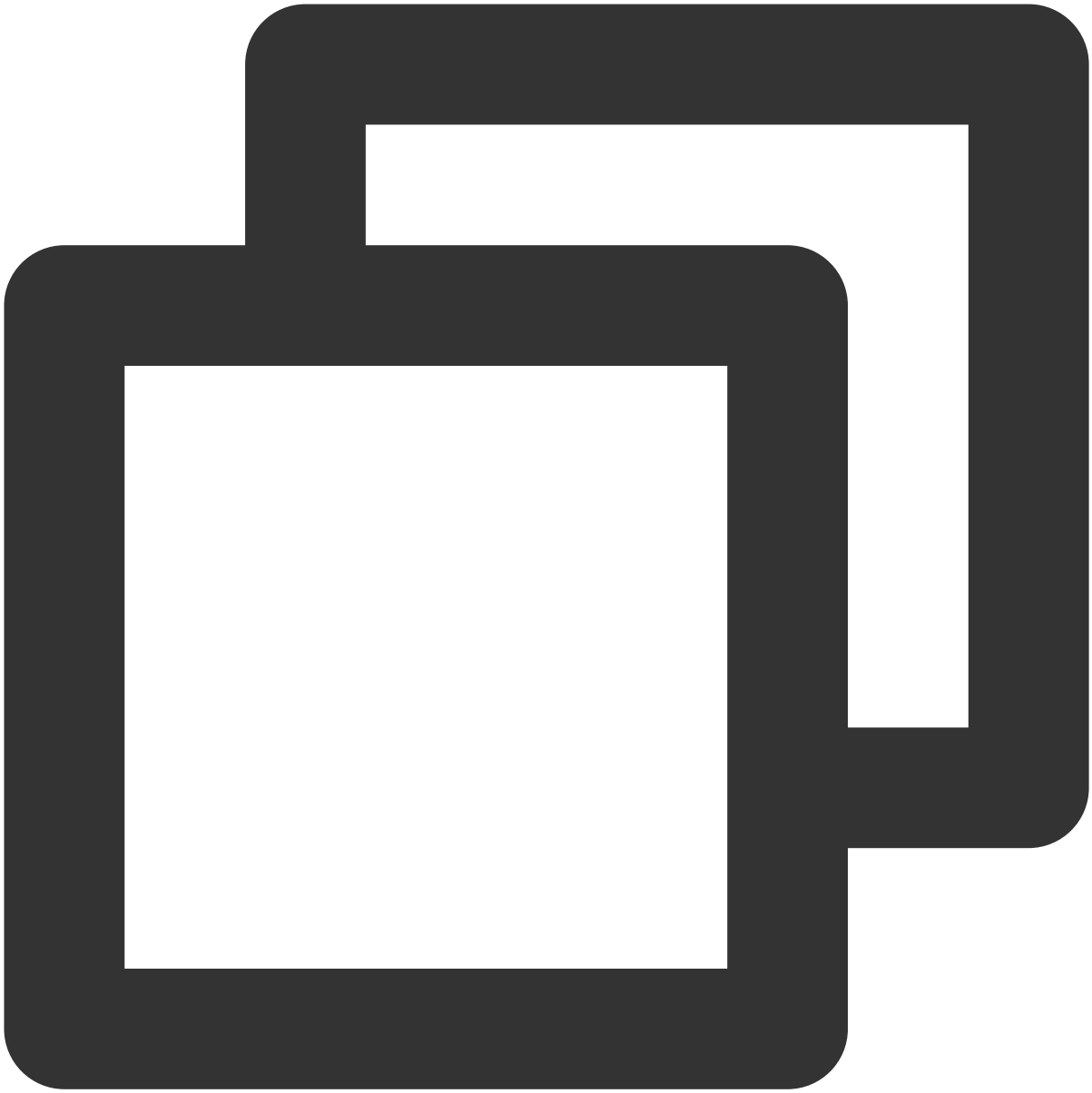


```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin .\\TUIKit\\tui-cu
```

步骤3：引入 TUIKit 组件

1. 工程配置

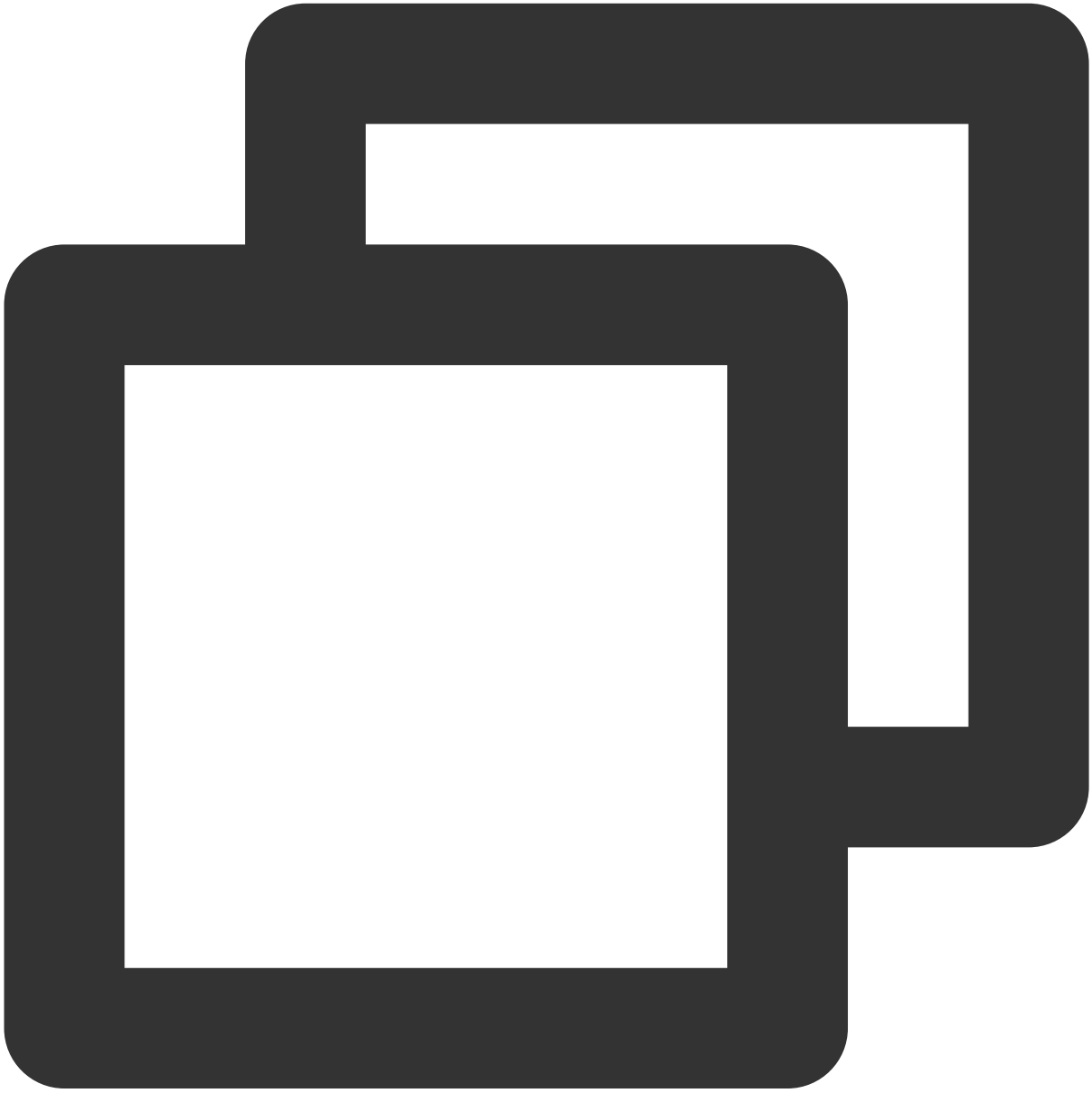
在根目录下创建 vue.config.js （vue3 项目请忽略此步骤）



```
const ScriptSetup = require('unplugin-vue2-script-setup/webpack').default;
module.exports = {
  parallel: false,
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
```

```
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete('fork-ts-checker');
  },
};
```

在 `manifest.json` 文件的源码视图中开启分包配置



```
{
    "mp-weixin": {
        "appid": "",
        "optimization": {
```

```
            "subPackages": true
        }
    },
    "h5": {
        "optimization": {
            "treeShaking": {
                "enable": false
            }
        }
    }
}
```

2. 集成 TUIKIt

注意：

进行集成时，请严格按照以下**四个步骤**进行集成。如果您希望打包小程序，请不要跳过“小程序分包首页”的配置。

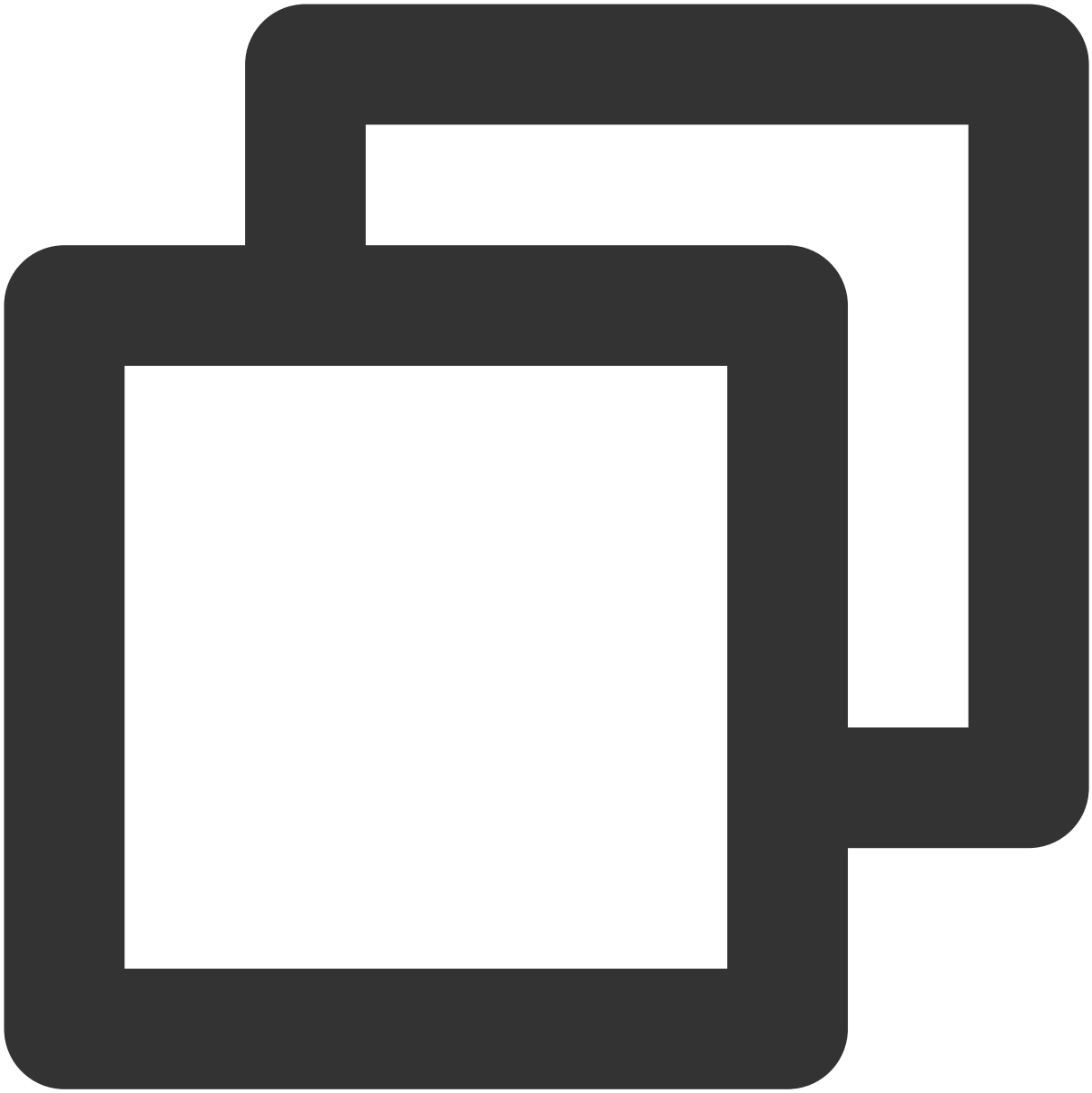
main.js 文件

pages.json 文件

App.vue 文件

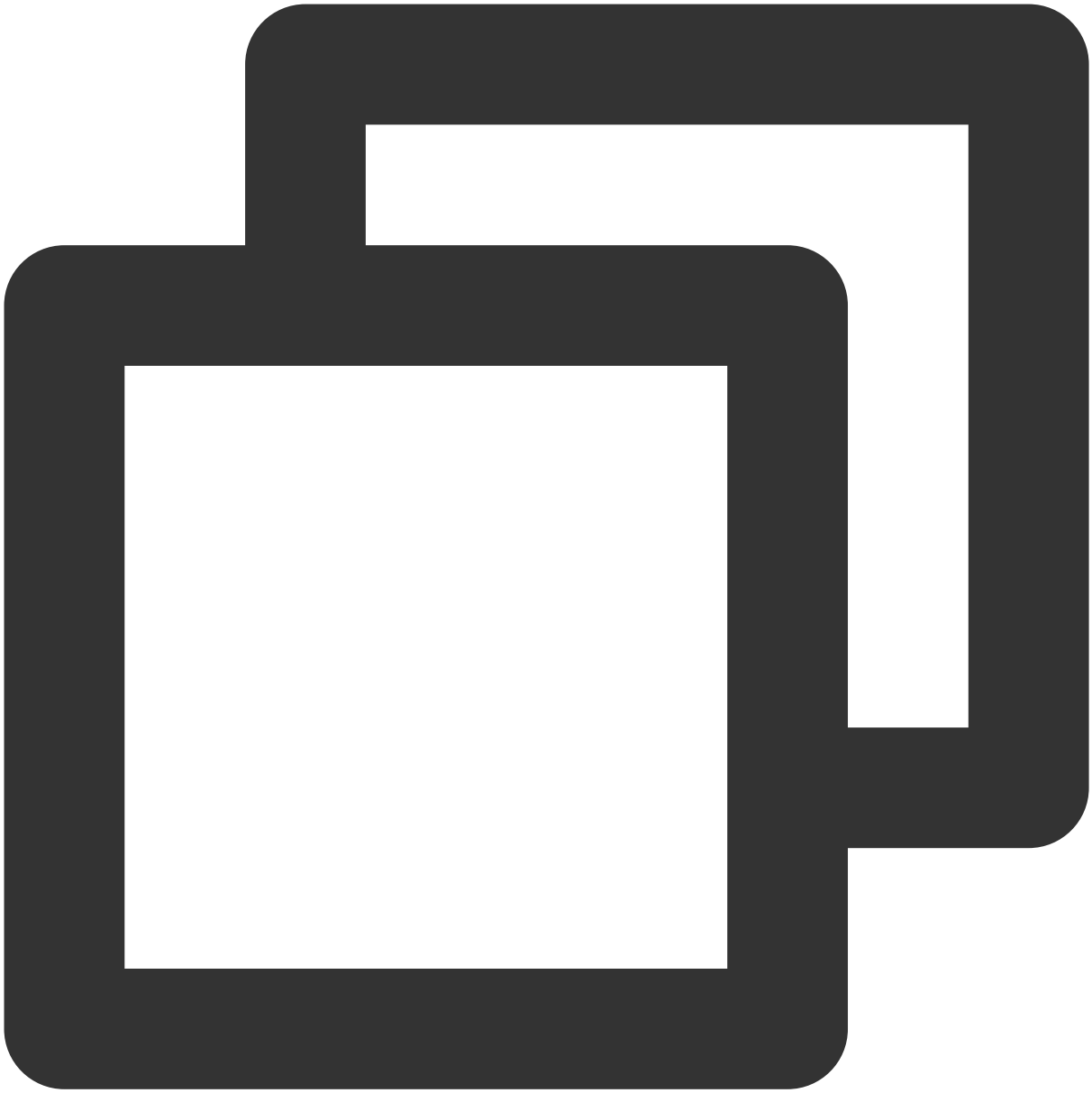
小程序分包首页

请注意，Vue2环境下要使用 `Vue.use(VueCompositionAPI)` ，防止环境变量 `isPC` 等无法使用。



```
// 引入主包依赖
import TencentCloudChat  from "@tencentcloud/chat";
import TUICore from "@tencentcloud/tui-core";

import App from './App';

// #ifndef VUE3
import Vue from 'vue';
import './uni.promisify.adaptor';
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
```

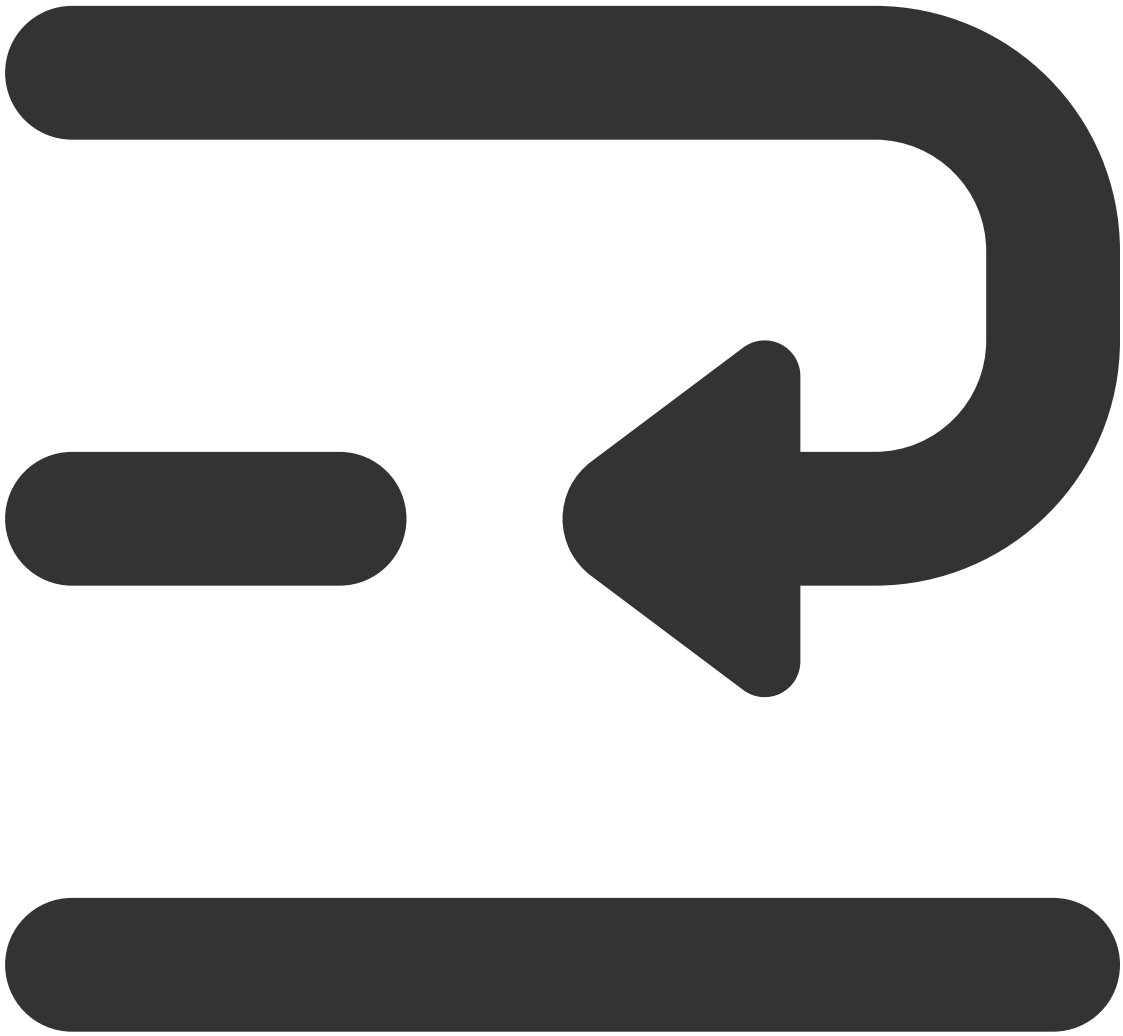
```
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

// #ifdef VUE3
import { createSSRApp } from 'vue';
export function createApp() {
  const app = createSSRApp(App);
  return {
    app,
  };
}
// #endif
```

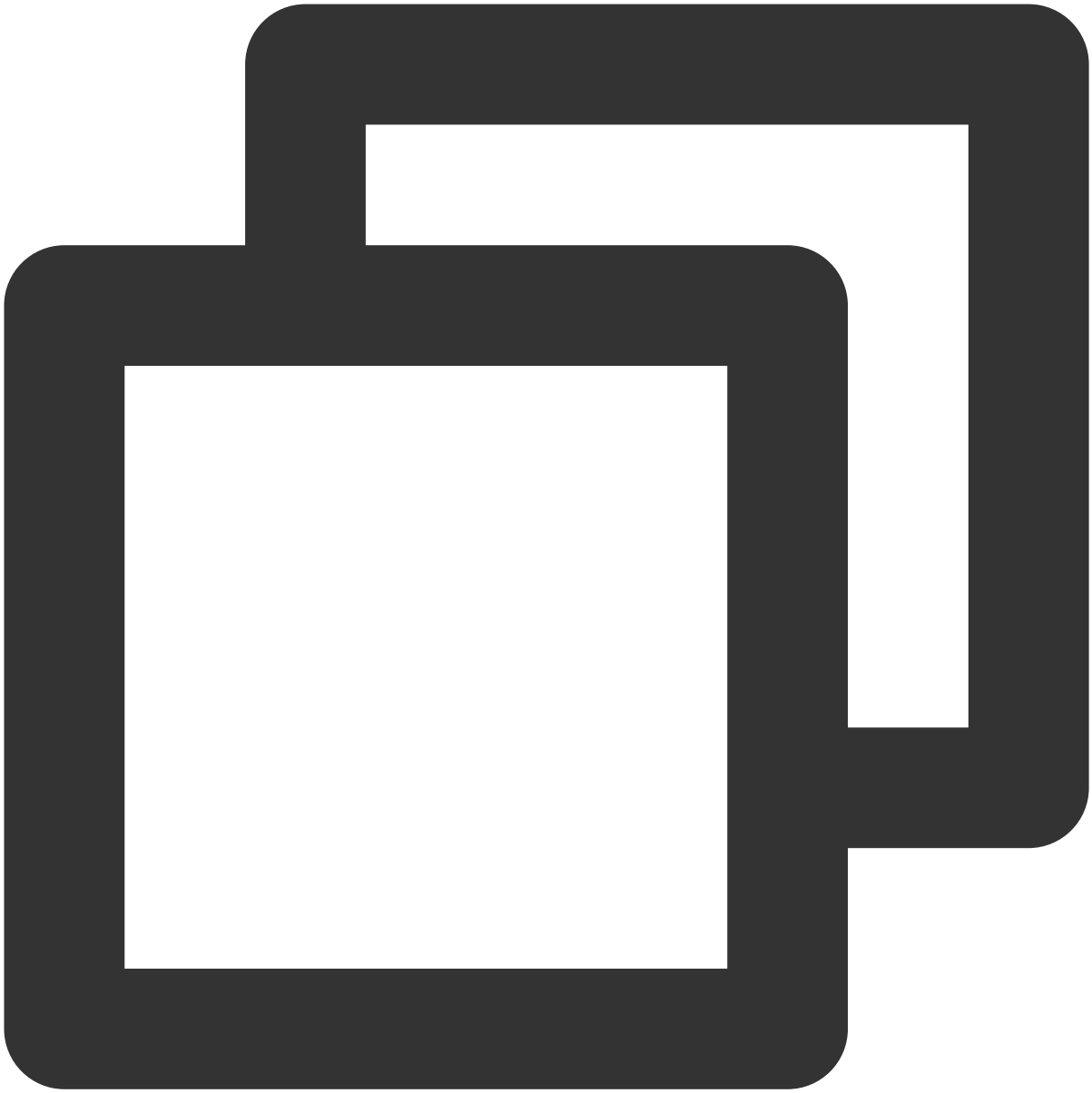


```
{
 "pages": [{
  "path": "pages/index/index" // 您的项目首页
 }],
 "subPackages": [{
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIConversation/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
```

```
    }
   },
   {
    "path": "components/TUIChat/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
    }
   },
    // 集成 chat 组件，必须配置该路径：视频播放
   {
    "path": "components/TUIChat/video-play",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
    }
   },
   {
    "path": "components/TUIChat/web-view",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
    }
   },
   {
    "path": "components/TUIContact/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
    }
   },

   {
    "path": "components/TUIGroup/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
    }
   }
  ]
 }],
 "preloadRule": {
  "TUIKit/components/TUIConversation/index": {
   "network": "all",
   "packages": ["TUIKit"]
  }
 },
 "globalStyle": {
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "uni-app",
  "navigationBarBackgroundColor": "#F8F8F8",
  "backgroundColor": "#F8F8F8"
```

```
  }
}
```





```
<script lang="ts">
// #ifdef APP-PLUS || H5
import { TUIChatKit, genTestUserSig } from "./TUIKit";
import { vueVersion } from "./TUIKit/adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";
// #endif
// 必填信息
const config = {
  userID: "test-user1", // User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
```

```
};
uni.$chat_userID = config.userID;
uni.$chat_SDKAppID = config.SDKAppID;
uni.$chat_secretKey = config.secretKey;

// #ifdef APP-PLUS || H5
uni.$chat_userSig = genTestUserSig(config).userSig;
// TUIChatKit 初始化
TUIChatKit.init();
// #endif
export default {
  onLaunch: function () {
    // #ifdef APP-PLUS || H5
    // TUICore login
    TUILogin.login({
      SDKAppID: uni.$chat_SDKAppID,
      userID: uni.$chat_userID,
      // UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。
      // 该方法仅适合本地跑通 Demo 和功能调试，详情请参见 https://cloud.tencent.com/docume
      userSig: uni.$chat_userSig,
      // 如果您需要发送图片、语音、视频、文件等富媒体消息，请设置为 true
      useUploadPlugin: true,
      // 本地审核可识别、处理不安全、不适宜的内容，为您的产品体验和业务安全保驾护航
      // 此功能为增值服务，请参考：https://cloud.tencent.com/document/product/269/79139
      // 如果您已购买内容审核服务，开启此功能请设置为 true
      useProfanityFilterPlugin: false,
      framework: `vue${vueVersion}` // 当前开发使用框架 vue2 / vue3
    });
    // #endif
  },
 onShow: function() {
      console.log('App Show')
  },
 onHide: function() {
      console.log('App Hide')
  }
};
</script>
<style>
/*每个页面公共css */
uni-page-body,
html,
body,
page {
  width: 100% !important;
  height: 100% !important;
  overflow: hidden;
```

```
}
</style>
```

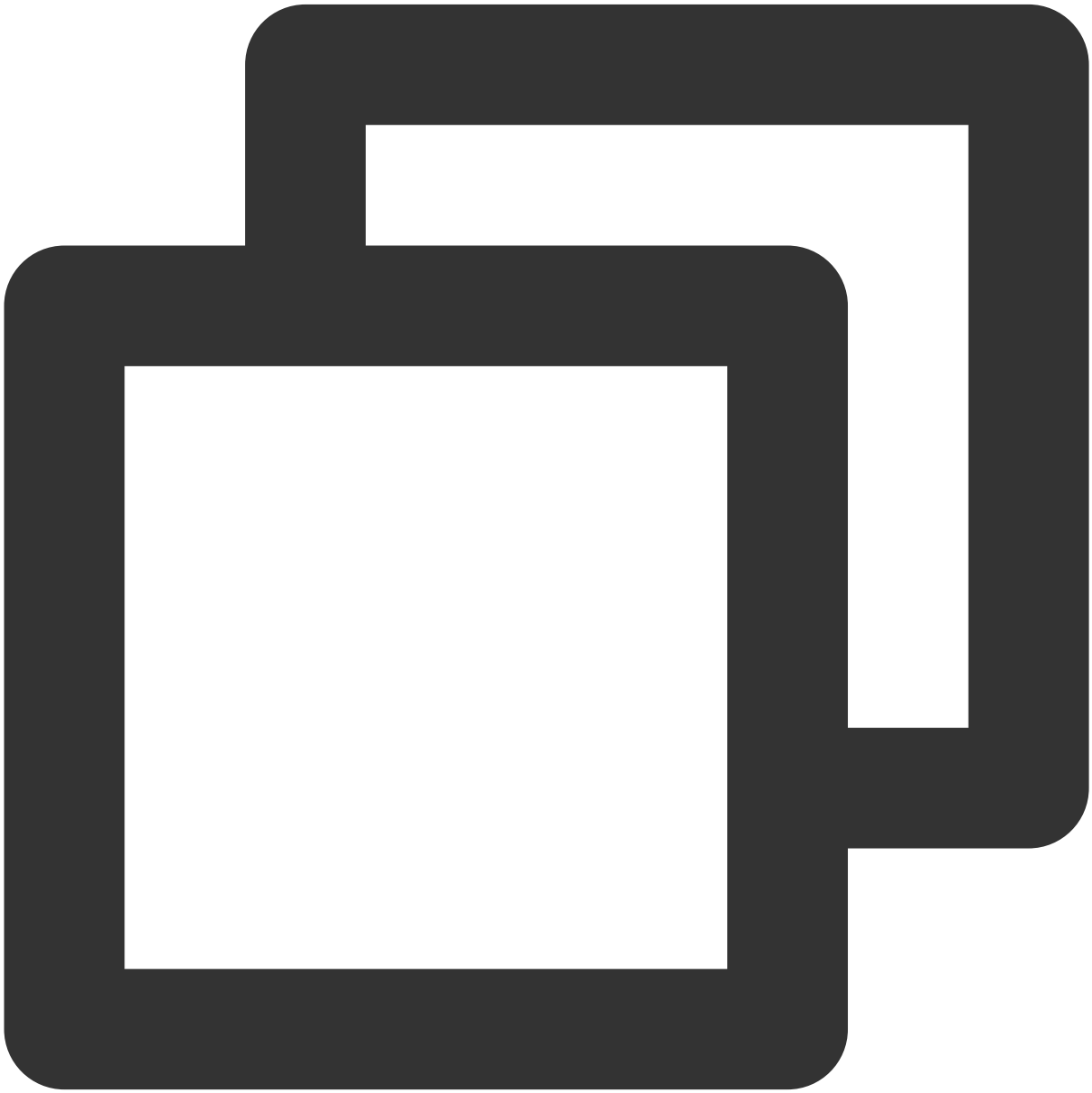
注意：

小程序默认分包集成，需要在 TUIKit 首启动页面完成 login。

如果您不需要打包小程序（如仅构建H5），可忽略"*小程序分包首页*"的配置内容。

示例：TUIKit 分包首屏启动页面为 **TUIConversation** 页面

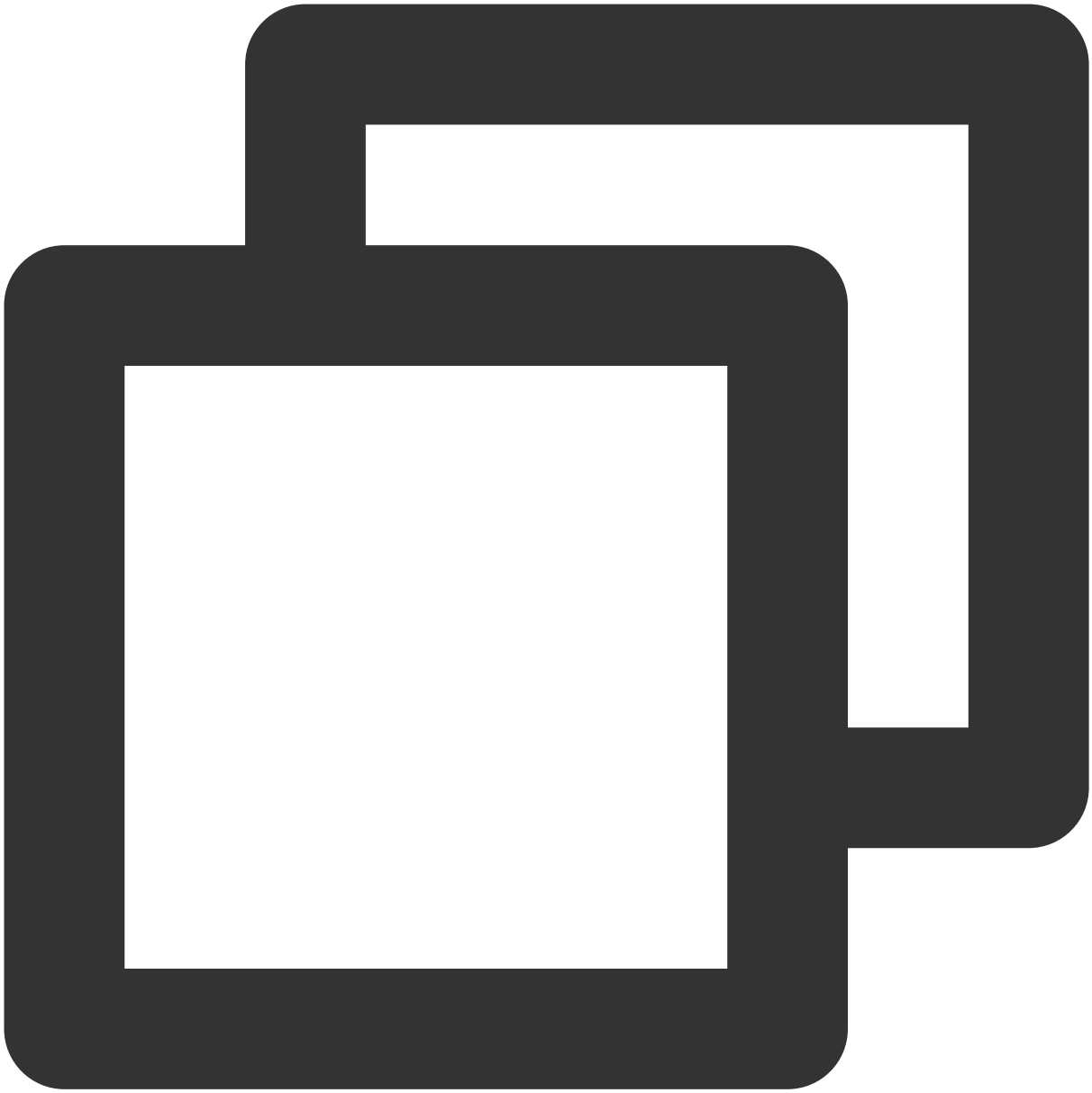
步骤1: 在 TUIKit/components/TUIConversation 文件夹下创建 subPackage-init.ts 文件



```
import { TUIChatKit, genTestUserSig } from "../../index.ts";
```

```
import { vueVersion, onMounted } from "../../adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";

// TUIChatKit 初始化
TUIChatKit.init();
uni.$chat_userSig = genTestUserSig({
        userID: uni.$chat_userID,
        SDKAppID: uni.$chat_SDKAppID,
        secretKey: uni.$chat_secretKey
}).userSig;

// login
TUILogin.login({
  SDKAppID: uni.$chat_SDKAppID,
  userID: uni.$chat_userID,
  // UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。
  // 该方法仅适合本地跑通 Demo 和功能调试，详情请参见 https://cloud.tencent.com/document/p
  userSig: uni.$chat_userSig,
  // 如果您需要发送图片、语音、视频、文件等富媒体消息，请设置为 true
  useUploadPlugin: true,
  // 本地审核可识别、处理不安全、不适宜的内容，为您的产品体验和业务安全保驾护航
  // 此功能为增值服务，请参考：https://cloud.tencent.com/document/product/269/79139
  // 如果您已购买内容审核服务，开启此功能请设置为 true
  useProfanityFilterPlugin: false,
  framework: `vue${vueVersion}` // 当前开发使用框架 vue2 / vue3
}).then(() => {
  uni.showToast({
    title: "login success"
  });
});
```

步骤2: 在 TUIKit/components/TUIConversation/index.vue 中导入



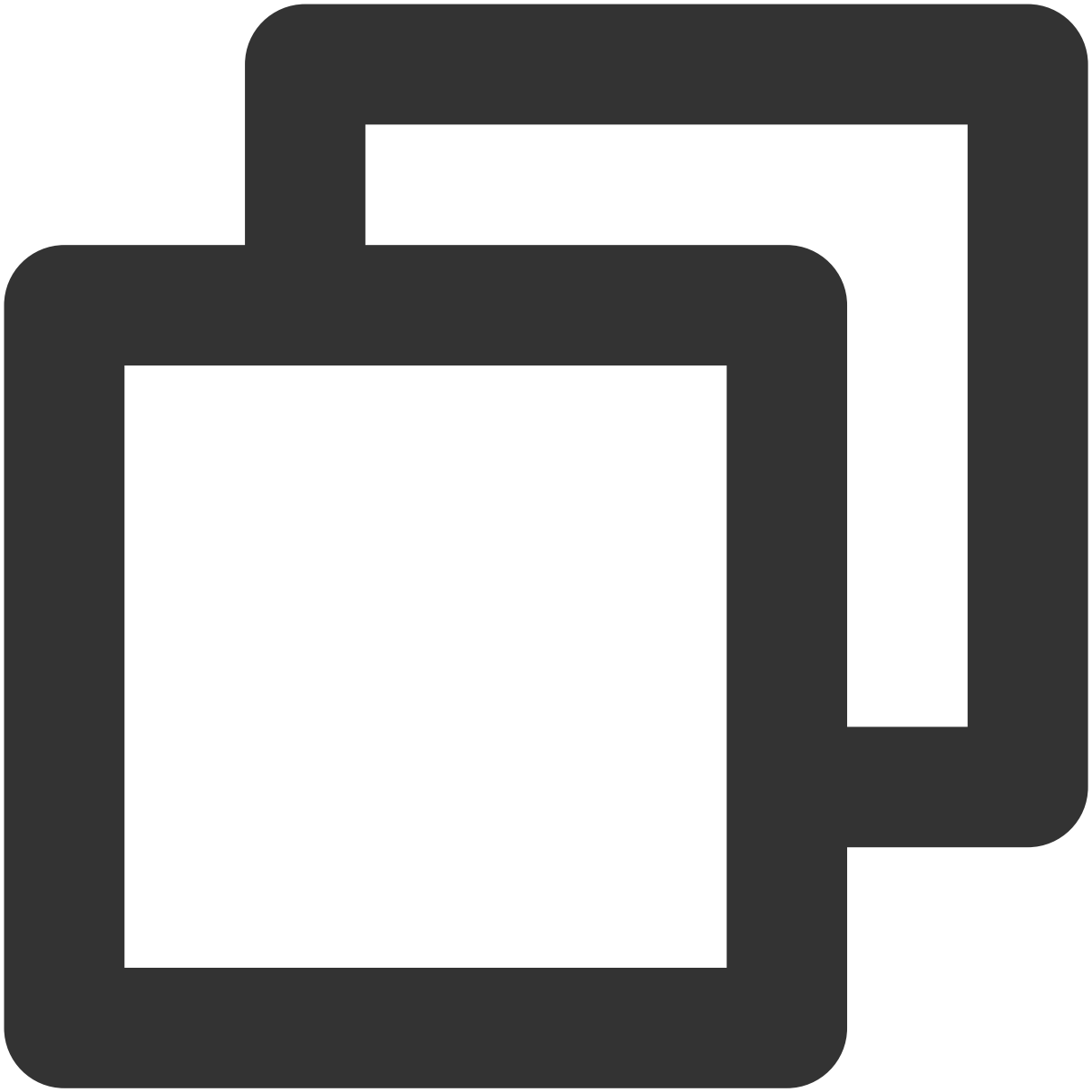
```
// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

如图所示：

```vue
<script lang="ts" setup>
import {
  TUIStore,
  TUIGlobal,
  StoreName,
} from "@tencentcloud/chat-uikit-engine";
import { ref } from "../../adapter-vue";
import ConversationList from "./conversation-list/index.vue";
import ConversationHeader from "./conversation-header/index.vue";
import ConversationNetwork from "./conversation-network/index.vue";
import { onHide } from "@dcloudio/uni-app"; // 该方法只能用在父组件内，子组件内不生效

// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

3. 在项目主包首页中配置 TUIConversation 和 TUIContact 的入口

在 pages/index 文件夹下创建 index.vue 文件

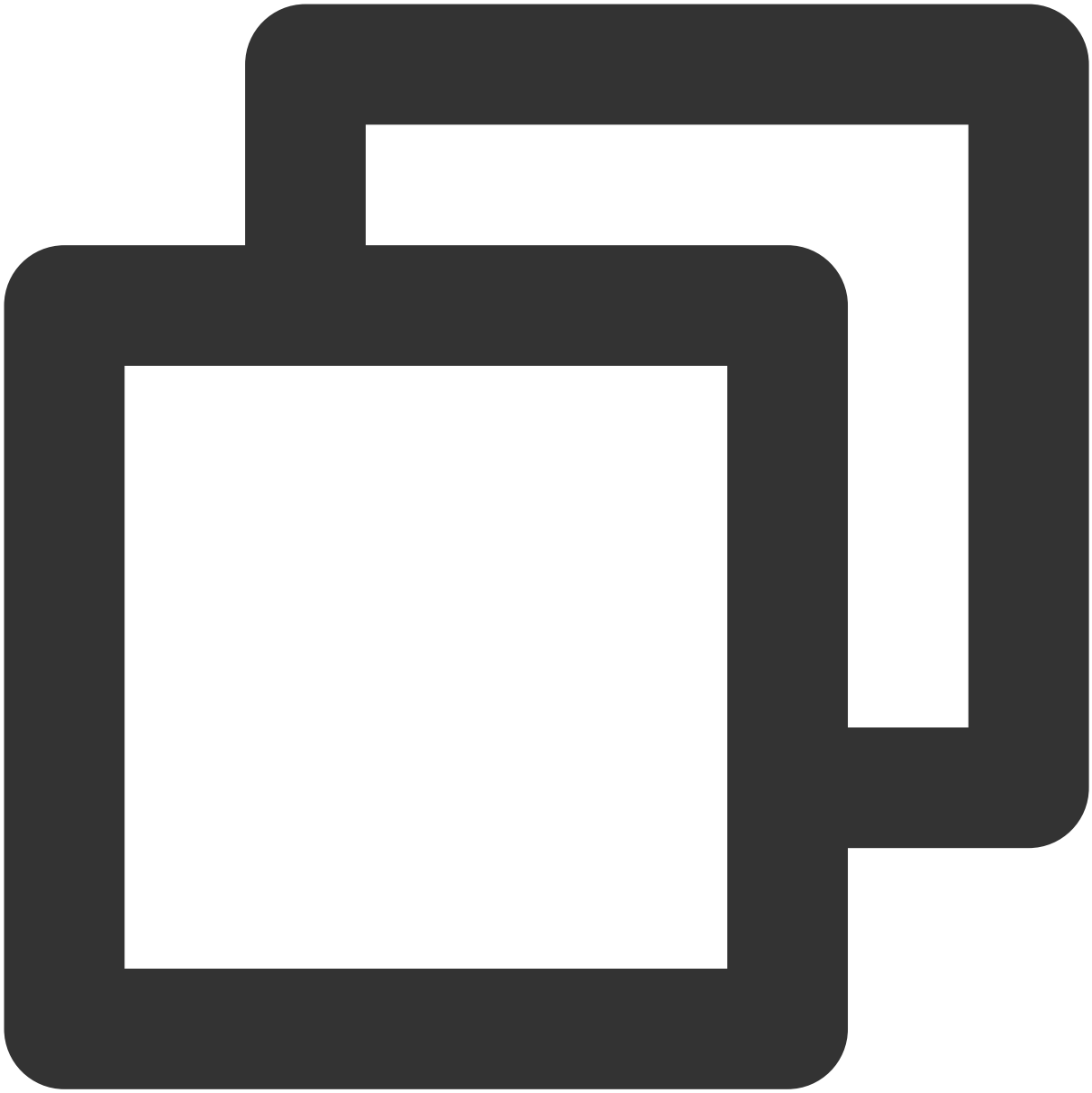


```
<template>
  <div class="index">
        <p class="index-button" @click="openConversation">打开 TUIKit 会话</p>
        <p class="index-button" @click="openContact">打开 TUIKit 联系人</p>
  </div>
</template>
<script>
export default {
methods: {
  // 打开 TUIKit 会话列表
  openConversation() {
```

```
    uni.navigateTo({
          url: "/TUIKit/components/TUIConversation/index",
        });
  },
  // 打开 TUIKit 联系人
  openContact() {
    uni.navigateTo({
          url: "/TUIKit/components/TUIContact/index",
        });
  },
 },
};
</script>
<style lang="scss" scoped>
.index {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  &-button {
    width: 180px;
        padding: 10px 40px;
        color: #fff;
        background-color: #006eff;
        font-size: 16px;
        margin-top: 65px;
        border-radius: 30px;
        text-align: center;
  }
}
</style>
```

步骤4：获取 SDKAppID、secretKey、userID

配置根目录下 App.vue 文件中 `config` 对象的 SDKAppID、secretKey 以及 userID。其中 SDKAppID 、secretKey 可通过 即时通信 IM 控制台获取，userID 可在 即时通信 IM 控制台中创建账户时获取。



```
// 必填信息
const config = {
  userID: "test-user1", // Login User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
};
```

获取 SDKAppID、secretKey

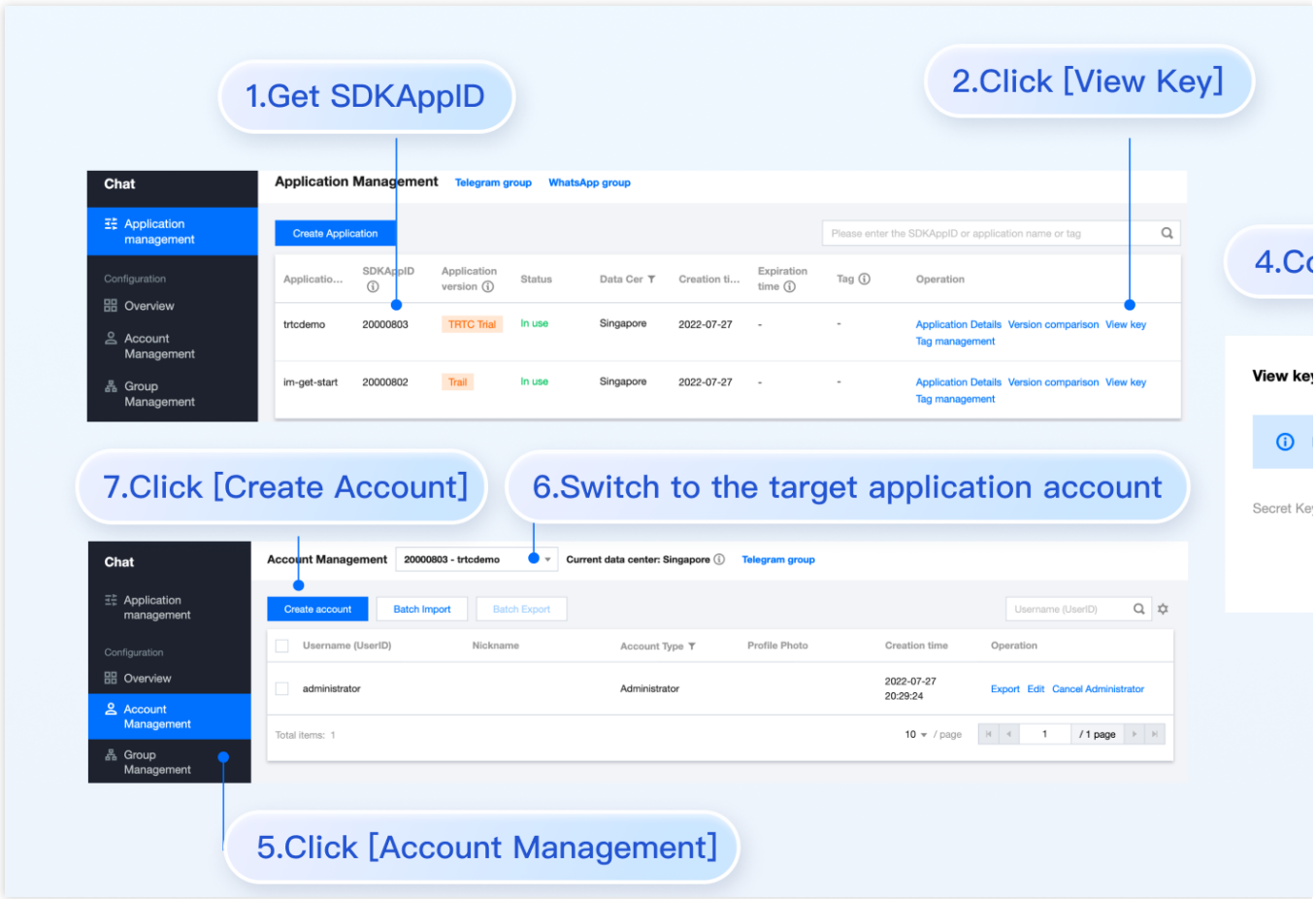
在 即时通信 IM 控制台中的**应用管理**页面下，可以看到您创建的应用，第二列即是 SDKAppID，然后单击操作中的**查看密钥**。网站会弹出查看密钥的对话框，然后单击**显示密钥**，即可查看密钥。

创建 userID 为 test-user1 的账户

点击控制台左侧的**账号管理**，如果您有多个应用，请注意切换至当前应用，然后在当前应用下单击**新建账号**，创建一个 userID 为 `test-user1` 的账号。

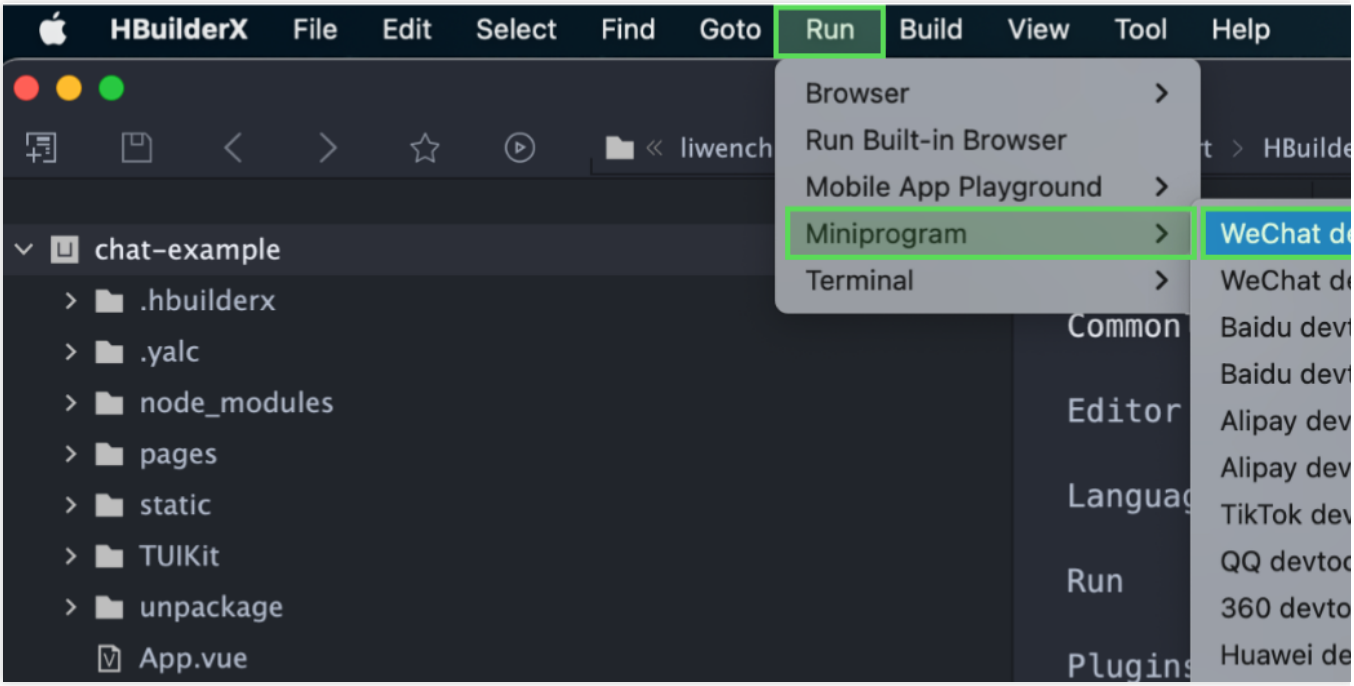
说明：

创建账户的步骤可以跳过，因为 TUIKit 进行登录时，若配置的 userID 不存在，会自动创建账户，此处仅展示如何获取 userID。



步骤5：启动项目

1. 使用 HBuilderX 启动该项目，点击“运行-运行到小程序模拟器-微信开发者工具”。

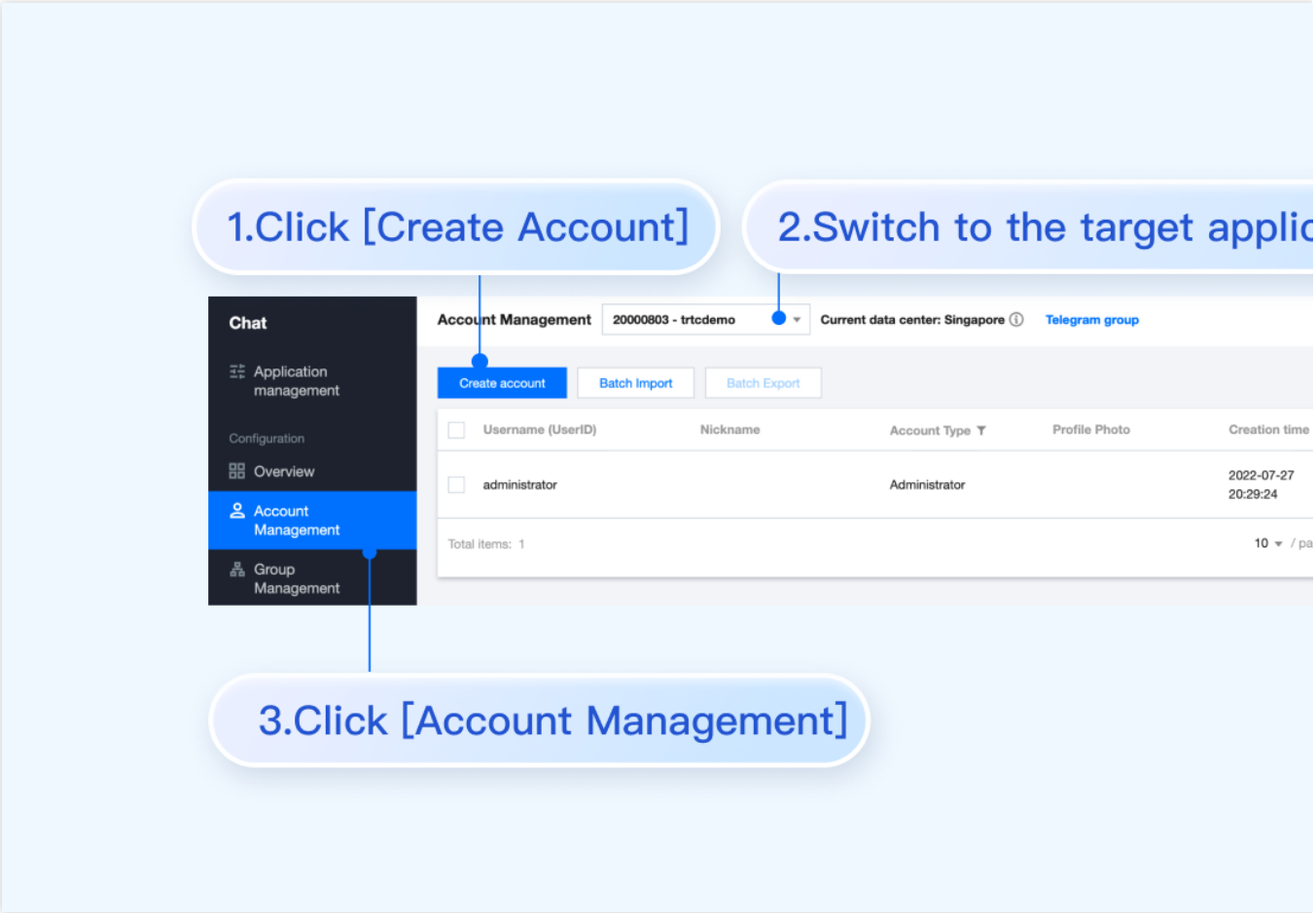


2. 如果 HBuilderX 没有自动拉起微信开发者工具，请使用微信开发者工具手动打开编译后的项目。
使用微信开发者工具打开项目根目录下的 `unpackage/dist/dev/mp-weixin` 即可。

3. 打开项目后，在微信开发者工具 “详情-本地设置” 中勾选 “不校验合法域名、web-view（业务域名）、TLS版本以及 HTTPS 证书”。

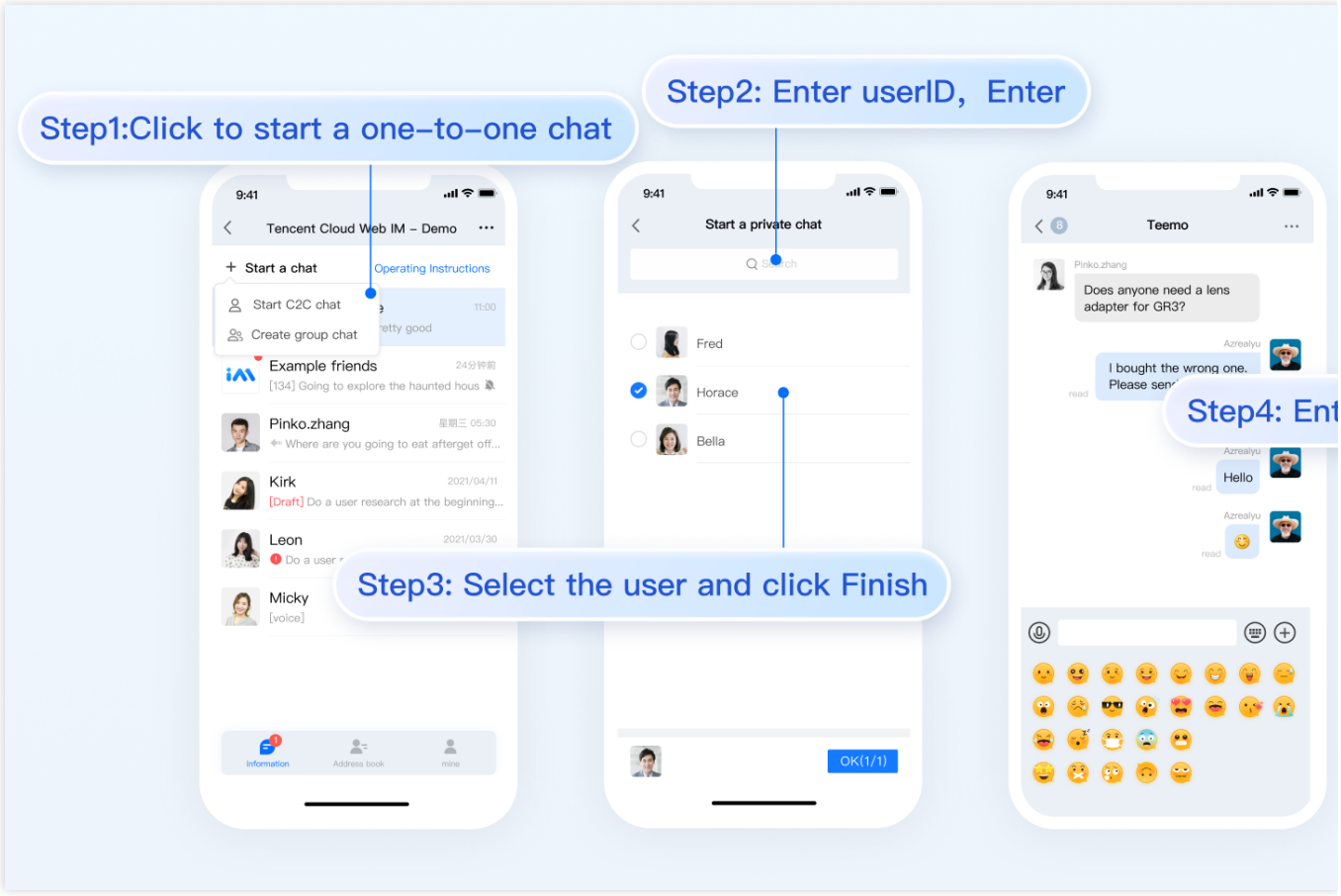
步骤6：发送您的第一条消息

1. 通过 即时通信 IM 控制台创建一个 User 账号
从左侧边栏进入**账号管理**页面，单击**新建账号**并创建一个普通账号 userID：test-user2。



2. 运行项目并发起会话

单击**打开 TUIKit 会话**，搜索用户 userID：test-user2，发送您的第一条消息。



更多高级特性

音视频通话 TUICallKit 插件

说明：

TUIKit 中默认没有集成 TUICallKit 音视频组件，TUICallKit 主要负责语音、视频通话。

如果您需要集成通话功能，可参考以下文档实现。

打包到 APP 请参考： 音视频通话（客户端）

打包到小程序请参考：音视频通话（小程序）

打包到 H5 请参考官：音视频通话（H5）

敬请期待。

TIMPush 离线推送插件

说明

TUIKit 中默认没有集成 TIMPush **离线推送插件**。TIMPush 是腾讯云即时通信 IM Push 插件。目前离线推送支持 Android 和 iOS 平台，设备有：华为、小米、OPPO、vivo、魅族 和 苹果手机。

如果您需要在 APP 中集成离线推送能力，请参见 uni-app 离线推送 实现。

敬请期待。

独立集成 TUIChat 组件

可参考 独立集成 TUIChat 组件 方案

常见问题

更多问题请参见 Uniapp 常见问题。

交流与反馈

点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

参考文档

UIKit （vue2 / vue3）相关：
chat-uikit-uniapp (vue2/vue3) github 源码
chat-uikit-uniapp npm 快速接入
ChatEngine 相关：
ChatEngine API 手册
ChatEngine npm

Web & H5（Vue）

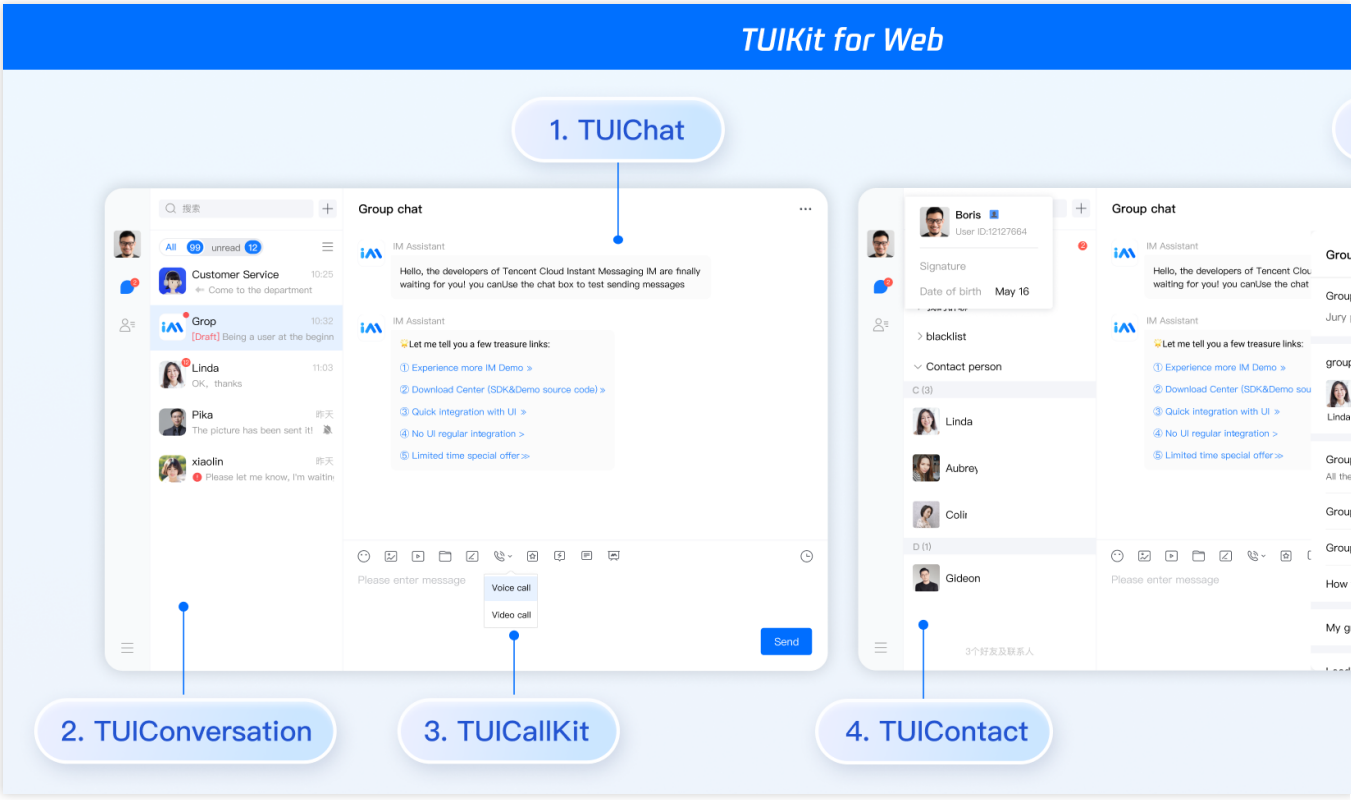
最近更新时间：2024-05-13 16:43:42

TUIKit 介绍

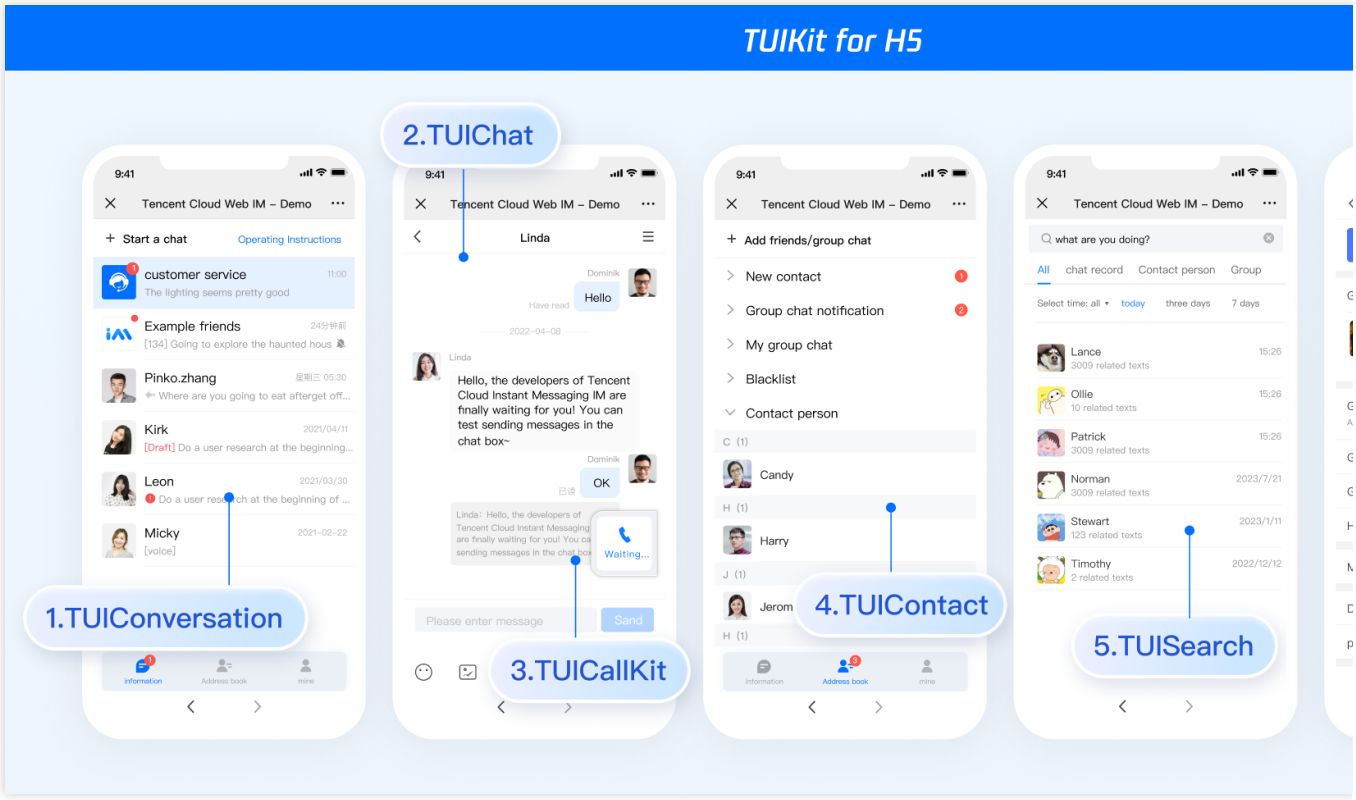
TUIKit 是基于腾讯云 IM SDK 的一款 UI 组件库，它提供了一些通用的 UI 组件，包含会话、聊天、关系链、群组、音视频通话等功能。
基于 UI 组件您可以像搭积木一样快速搭建起自己的业务逻辑。
TUIKit 中的组件在实现 UI 功能的同时，会调用 IM SDK 相应的接口实现 IM 相关逻辑和数据的处理，因而开发者在使用 TUIKit 时只需关注自身业务或个性化扩展即可。

TUIKit 主要功能介绍

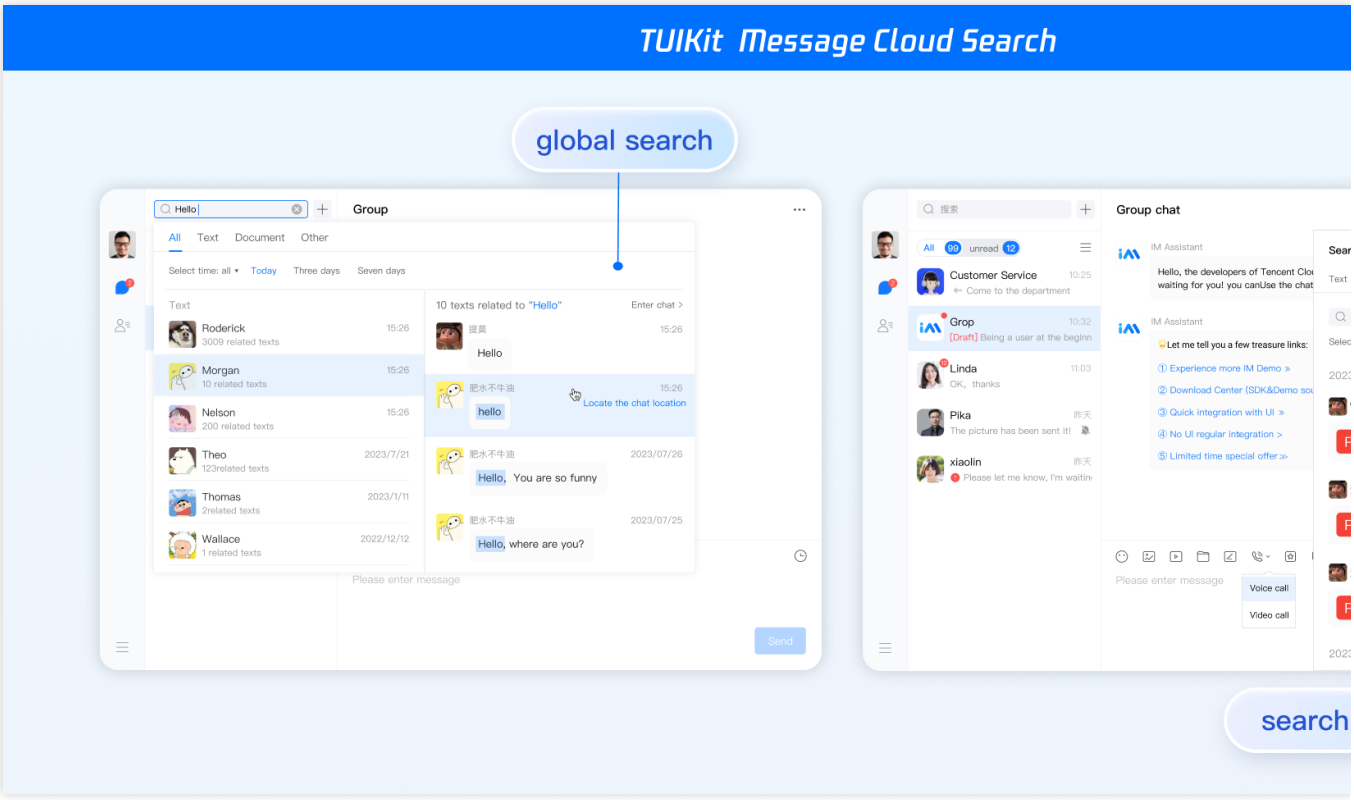
TUIKit 主要分为 TUIChat、TUIConversation、TUIGroup、TUIContact、TUISearch 几个 UI 子组件，每个 UI 组件负责展示不同的内容。
TUIKit Web 端界面效果如下图所示：



TUIKit H5 端界面效果如下图所示：



消息云端搜索功能界面效果如下图所示：



开发环境要求

Vue (全面支持 Vue2 & Vue3 ， 请您在下方接入时选择您所匹配的 Vue 版本接入指引进行接入)
TypeScript （ 如您是 js 项目， 请跳转至 js 工程如何接入 TUIKit 组件? 进行配置 ts 渐进式支持）
sass（sass-loader 版本 ≤ 10.1.1）
node（node.js ≥ 16.0.0）
npm（版本请与 node 版本匹配）

TUIKit 源码集成（ Web & H5 ）

步骤1：创建项目

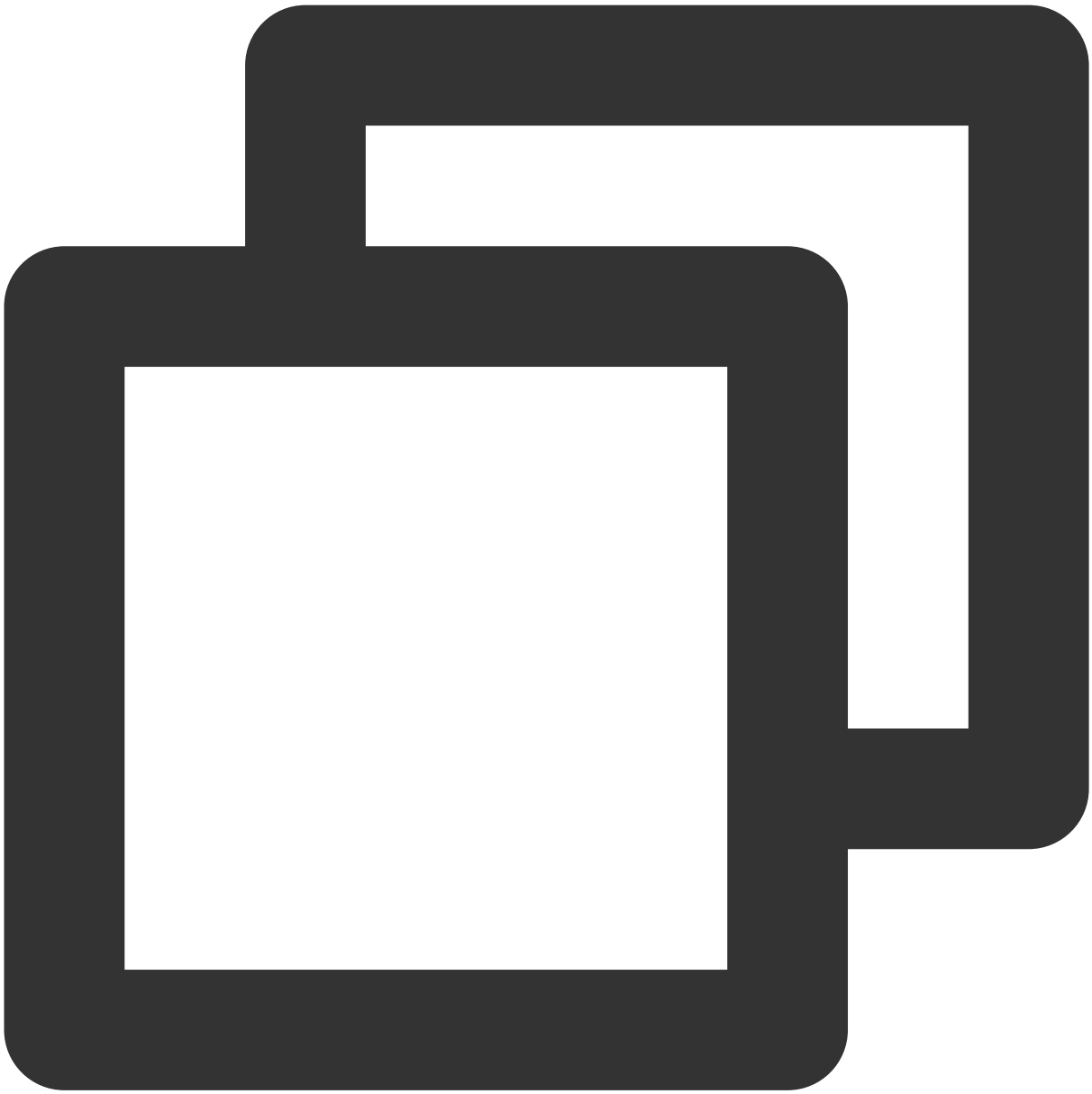
TUIKit 支持使用 webpack 或 vite 创建项目工程，配置 Vue3 / Vue2 + TypeScript + sass。以下是几种项目工程搭建示例：

vue-cli
vite

注意：

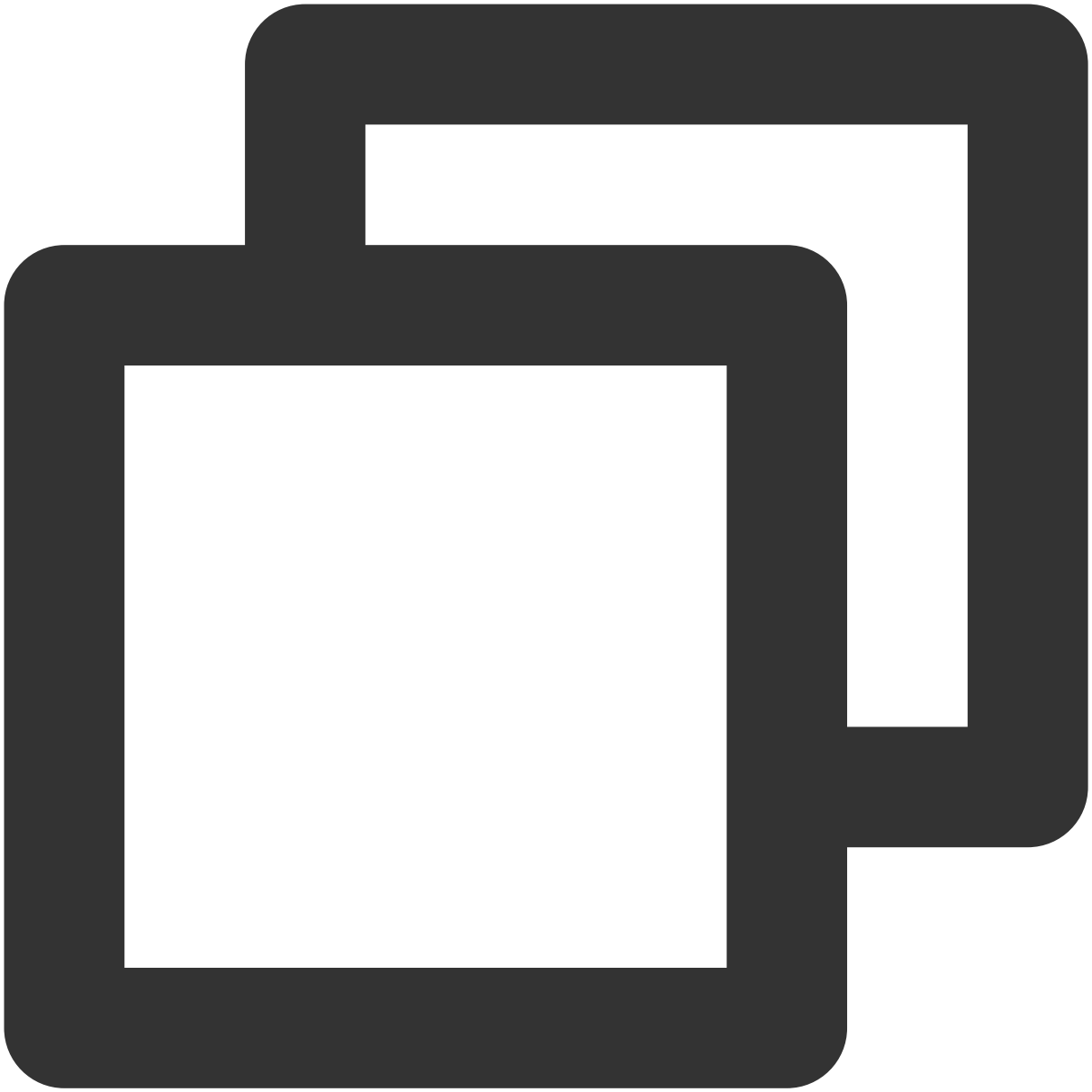
请您务必保证您的 **@vue/cli 版本在 5.0.0 以上**，您可使用以下示例代码升级您的 @vue/cli 版本至 v5.0.8。

使用 vue-cli 方式创建项目， 配置 Vue2 / Vue3 + TypeScript + sass。
如果您尚未安装 vue-cli 或者 vue-cli 版本低于 5.0.0 ，可以在 terminal 或 cmd 中采用如下方式进行安装：



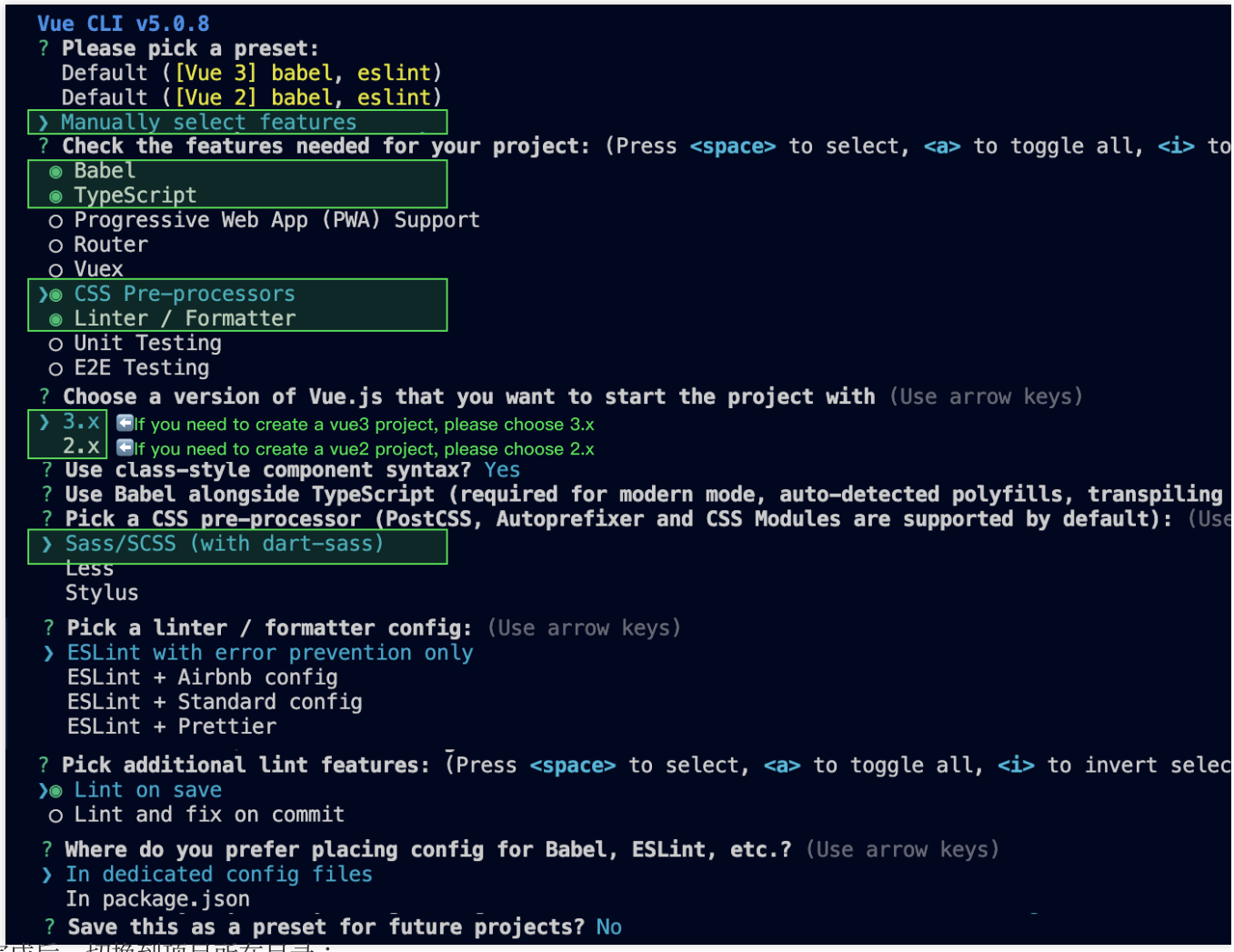
```
npm install -g @vue/cli@5.0.8 sass sass-loader@10.1.1
```

通过 vue-cli 创建项目，并选择下图中所选配置项。

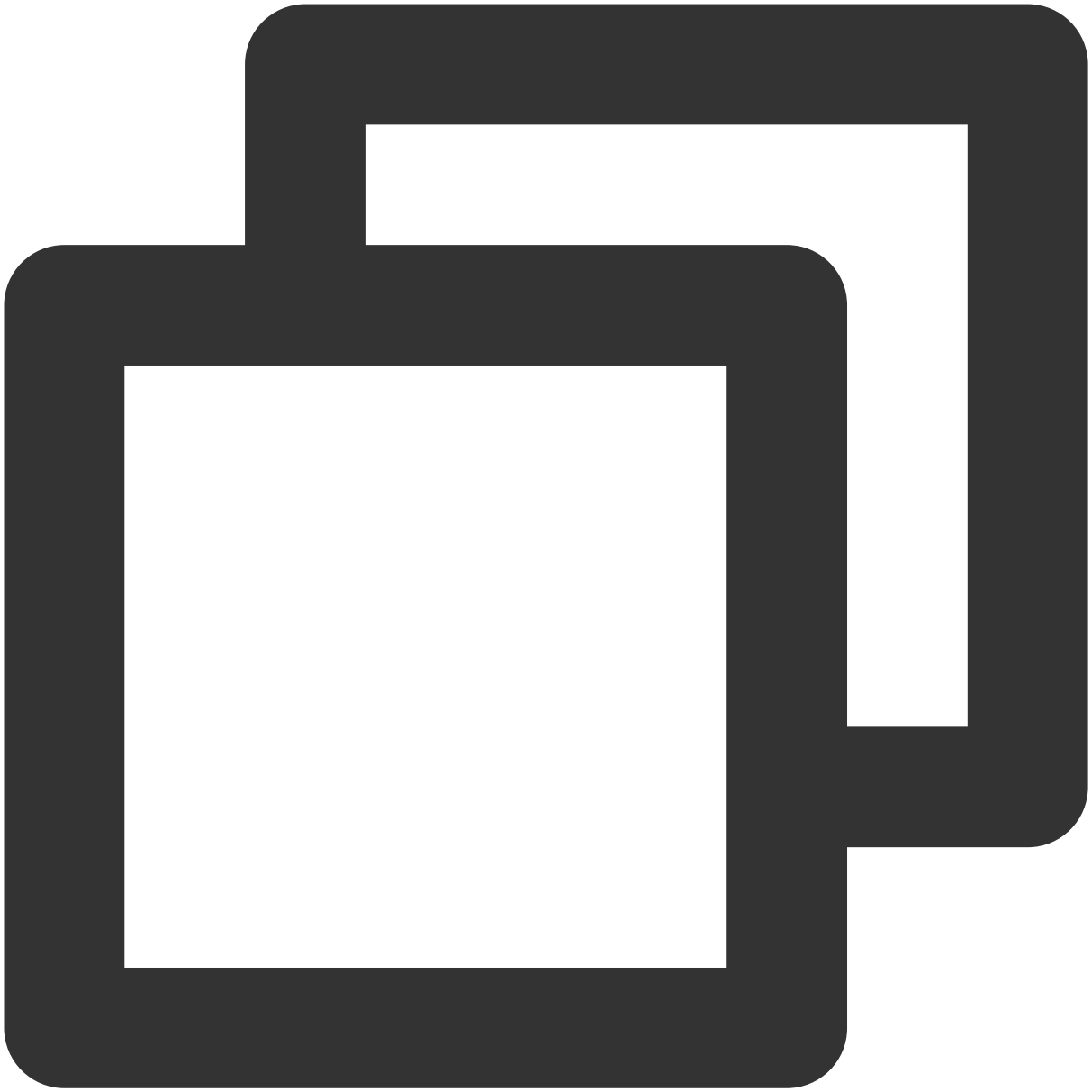


```
vue create chat-example
```

请务必保证按照如下配置选择：



创建完成后，切换到项目所在目录：

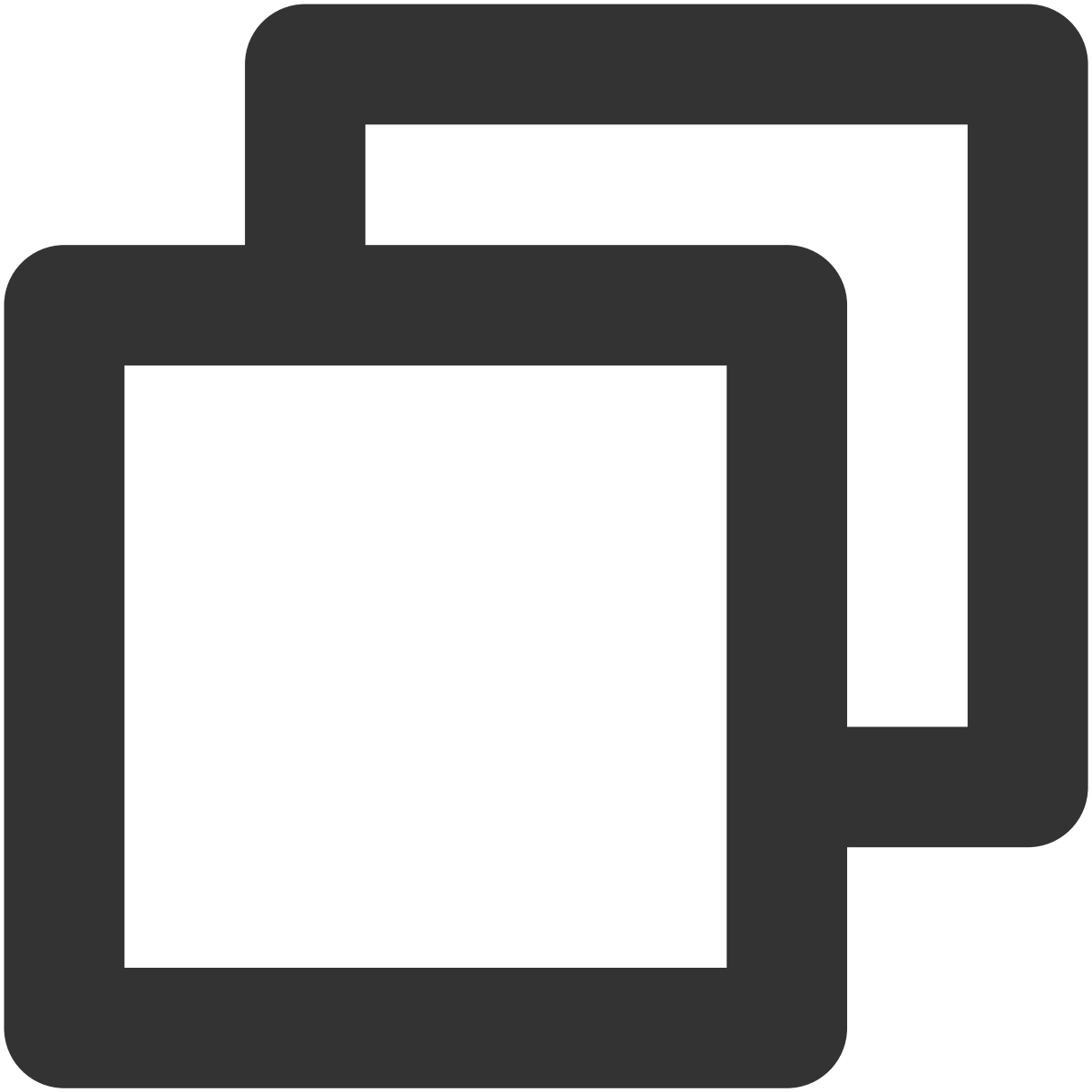


```
cd chat-example
```

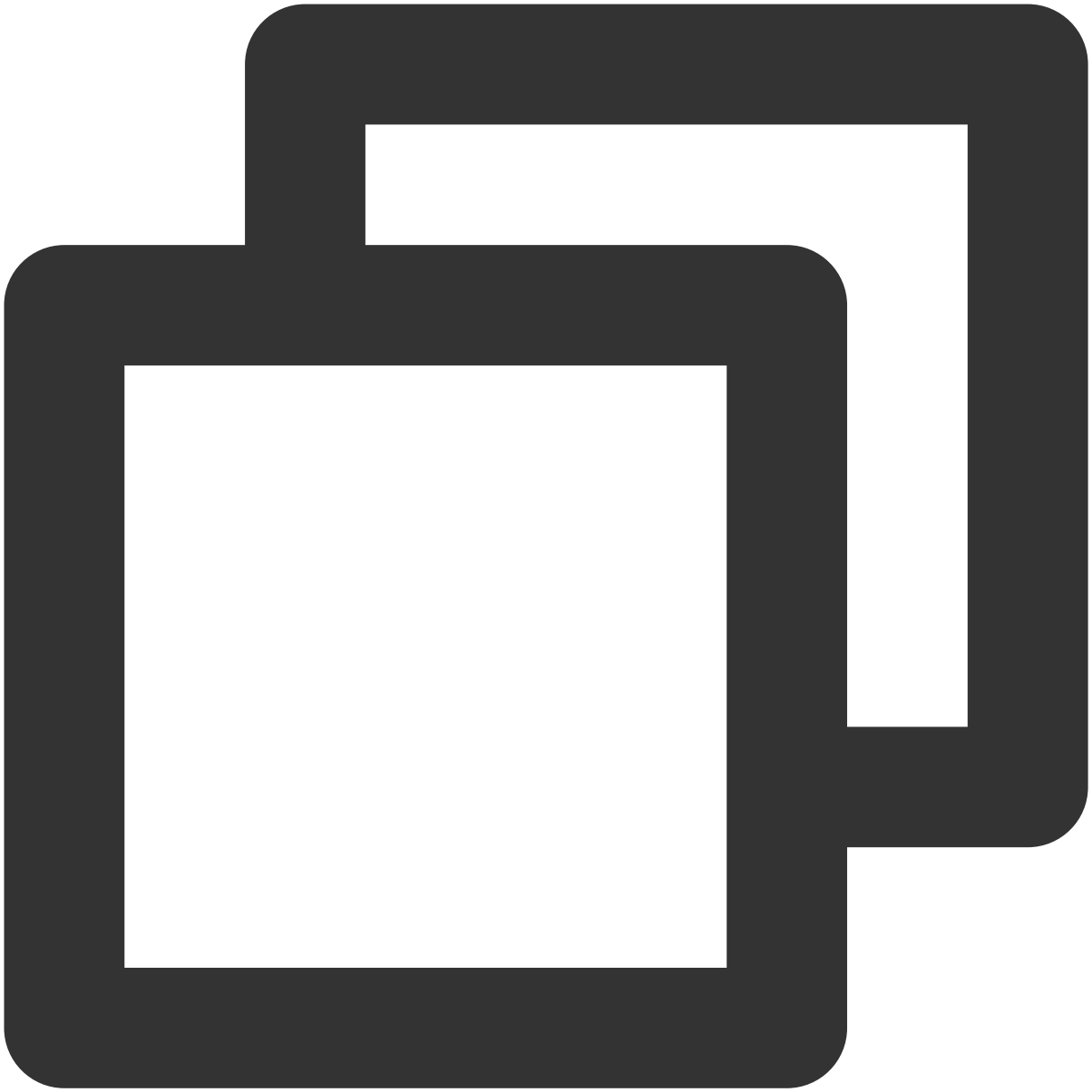
如果您是 vue2 项目，请根据您所使用的 Vue 版本进行以下相应的环境配置， vue3 项目请忽略。

vue2.7

vue2.6及以下



```
npm i vue@2.7.9 vue-template-compiler@2.7.9
```

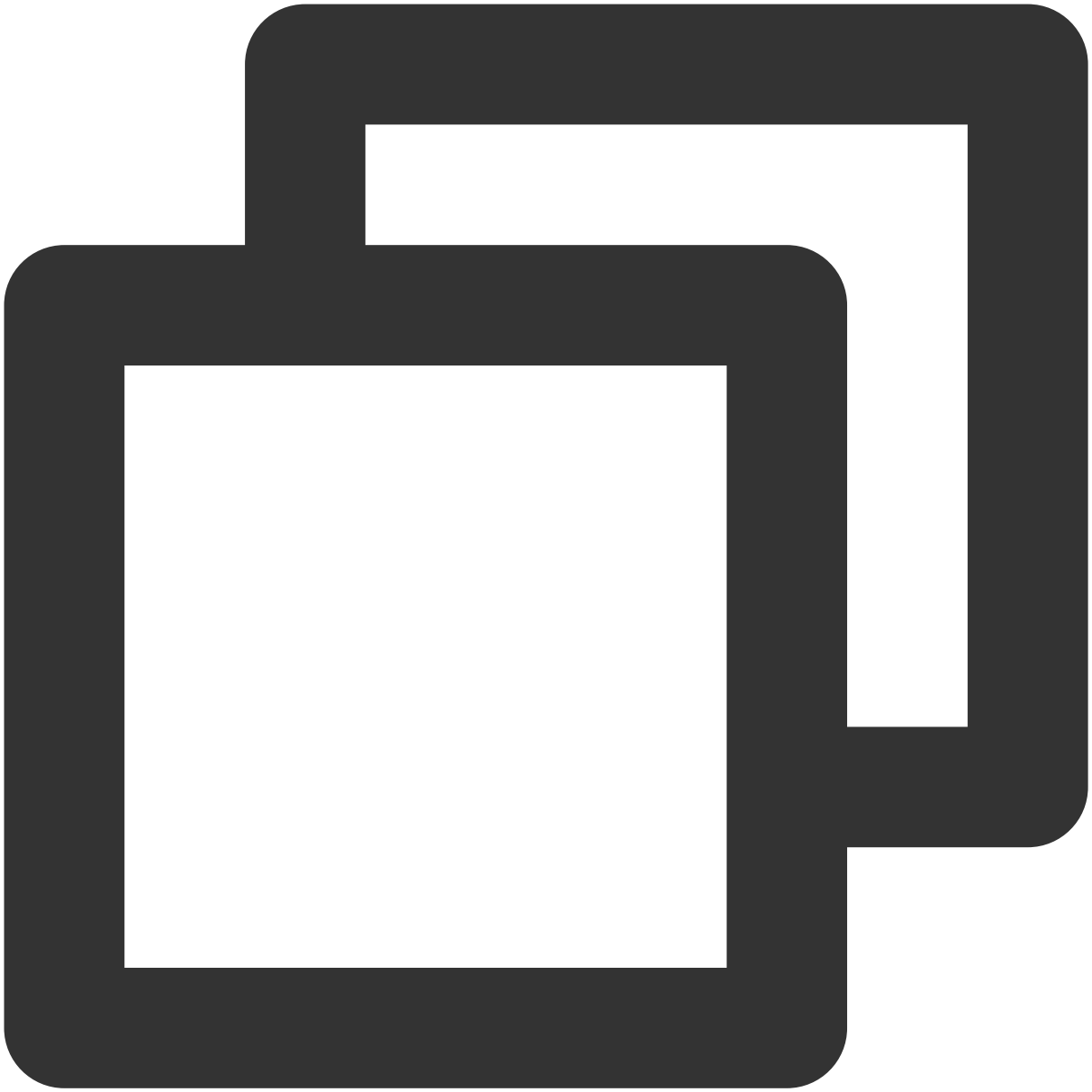


```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

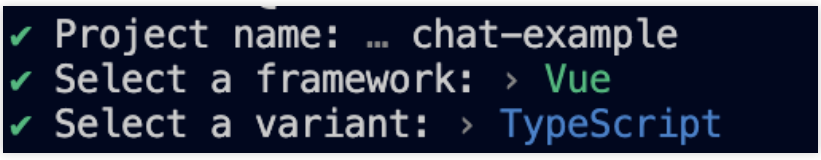
注意：

Vite 需要 **Node.js 版本 18+，20+。**当你的包管理器发出警告时，请注意升级你的 Node 版本，详情请参考 vite 官网。

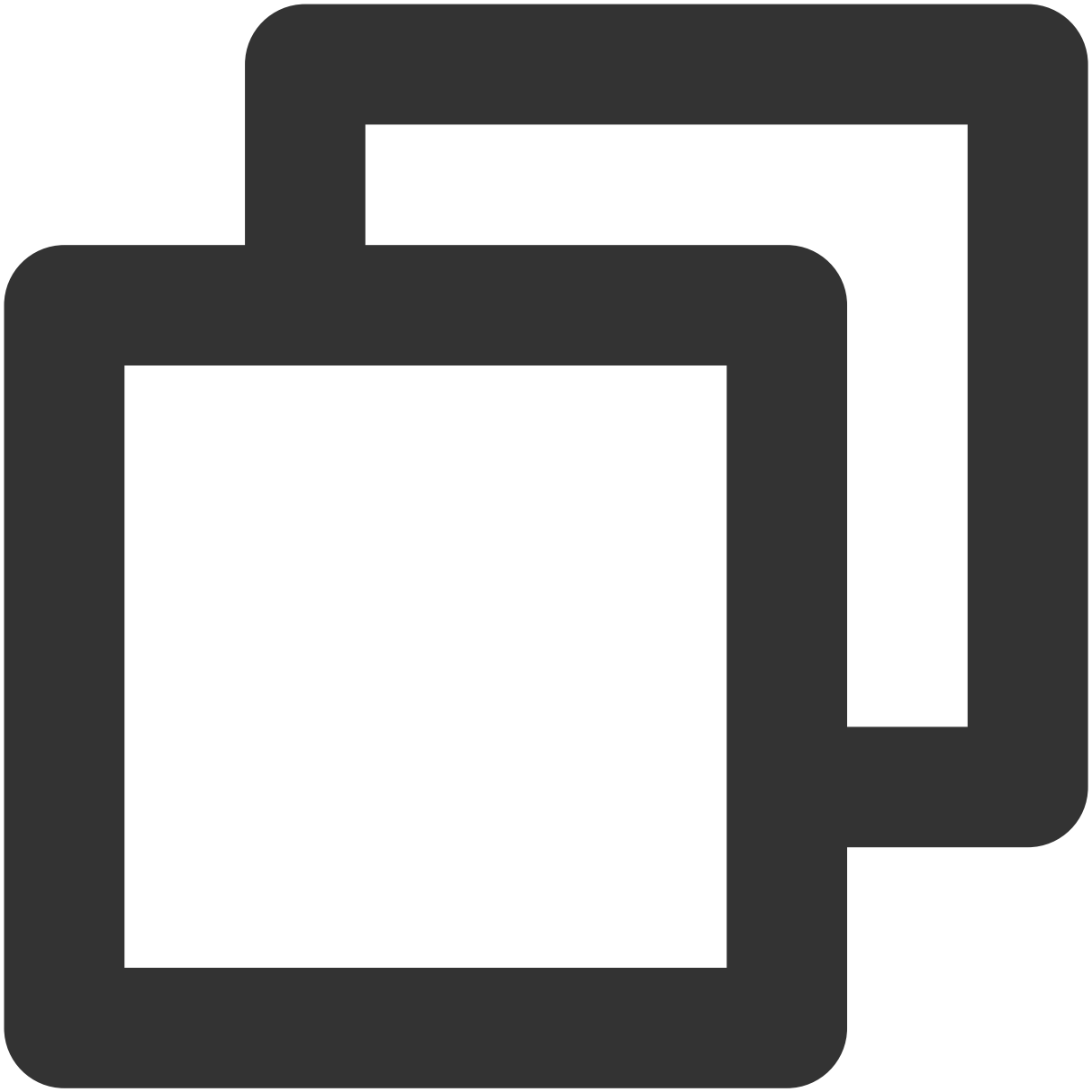
使用 vite 方式创建项目，按照下图选项配置 Vue + TypeScript 。



```
npm create vite@latest
```

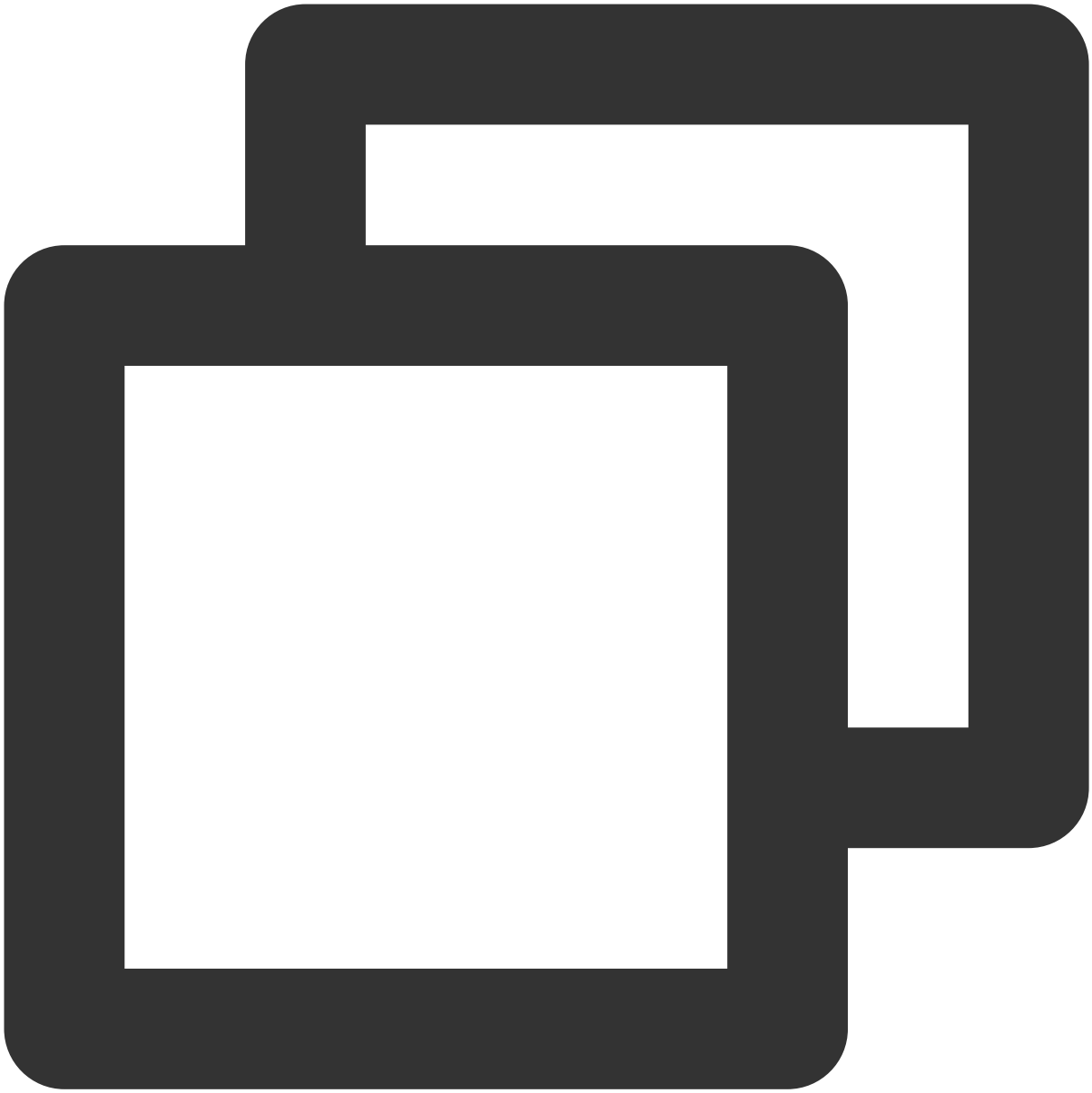


之后切换到项目目录，安装项目依赖：



```
cd chat-example
npm install
```

安装 TUIKit 所需 sass 环境依赖：



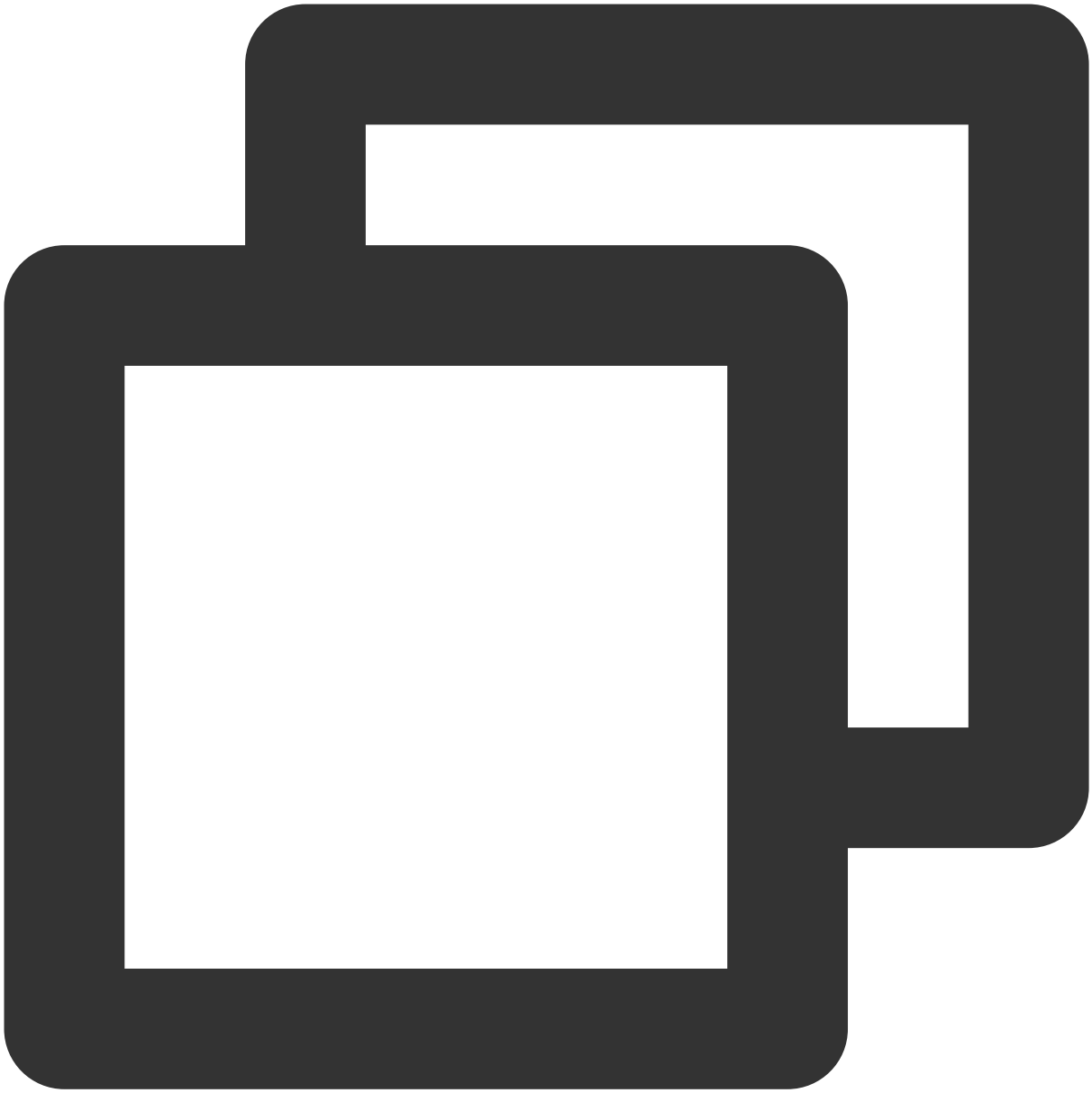
```
npm i -D sass sass-loader
```

步骤2：下载 TUIKit 组件

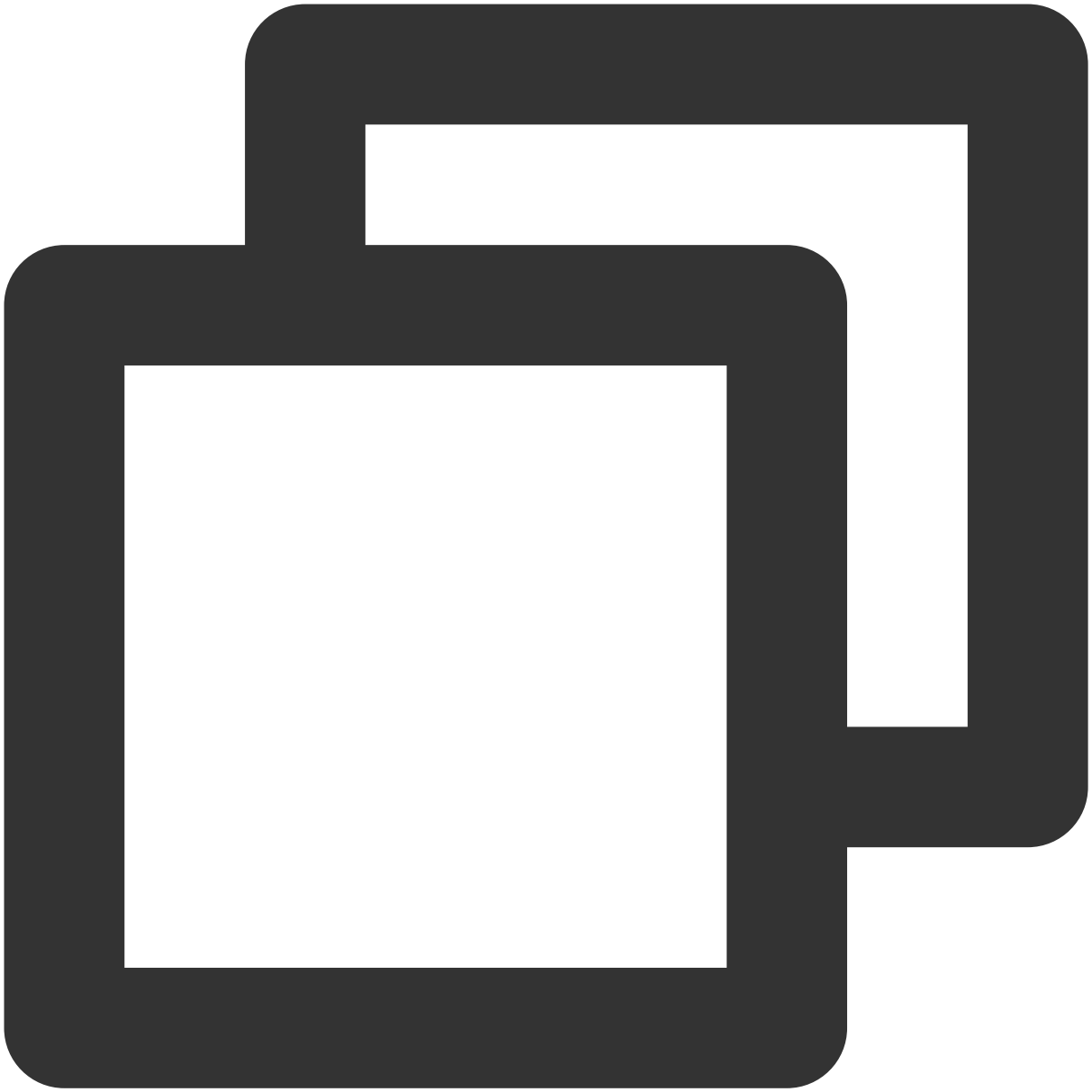
通过 npm 方式下载 TUIKit 组件，为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 src 目录下：

macOS 端

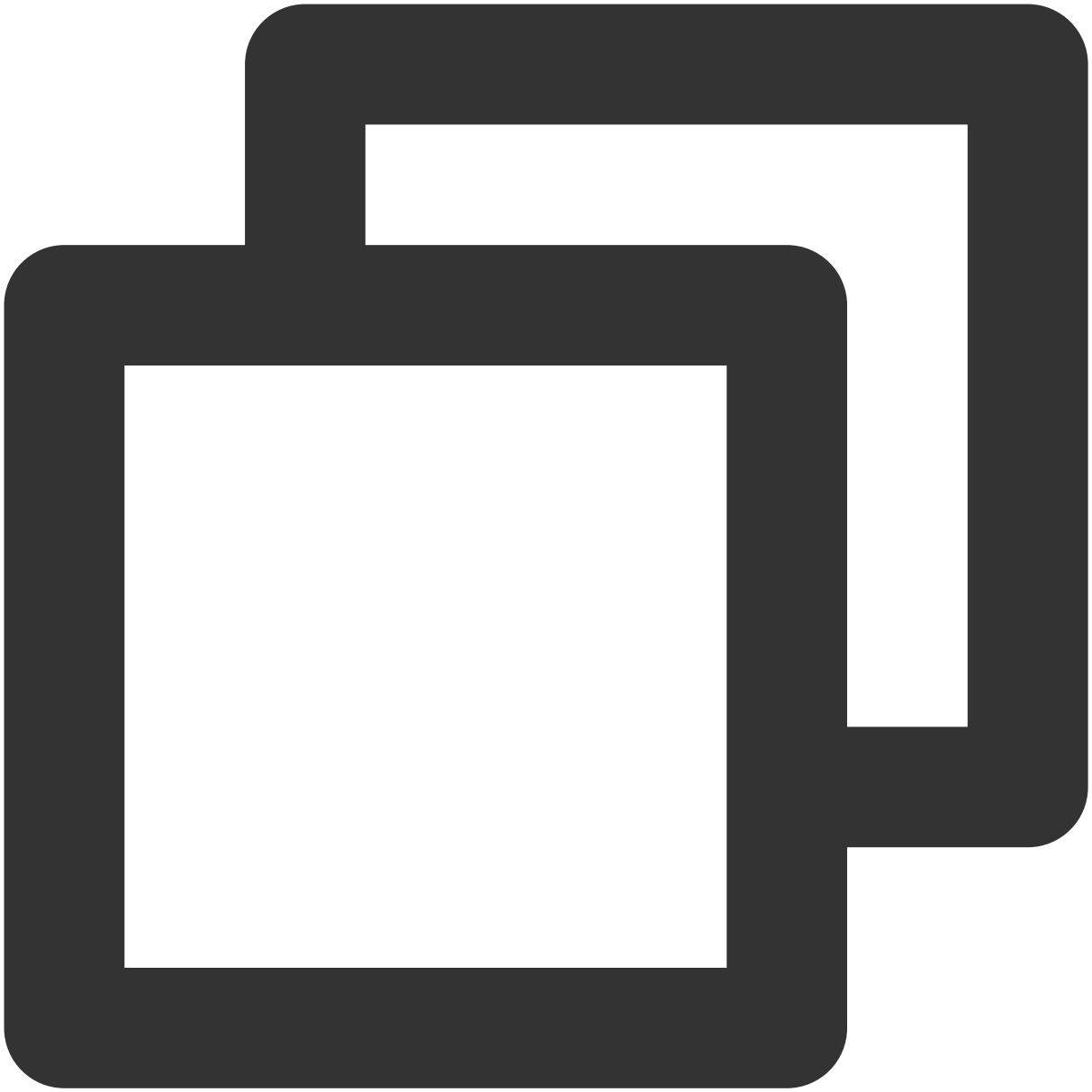
Windows 端



```
npm i @tencentcloud/chat-uikit-vue
mkdir -p ./src/TUIKit && rsync -av --exclude={'node_modules','package.json','exclud
```

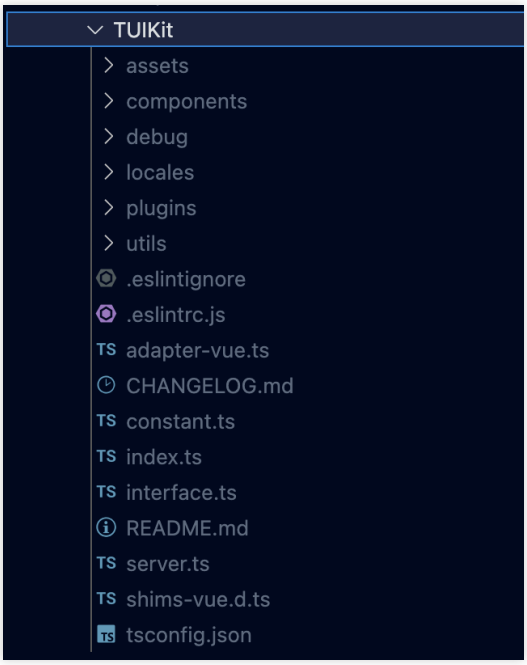


```
npm i @tencentcloud/chat-uikit-vue
```



```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-vue .\\src\\TUIKit /i /e /exclude:
```

成功后目录结构如图所示：

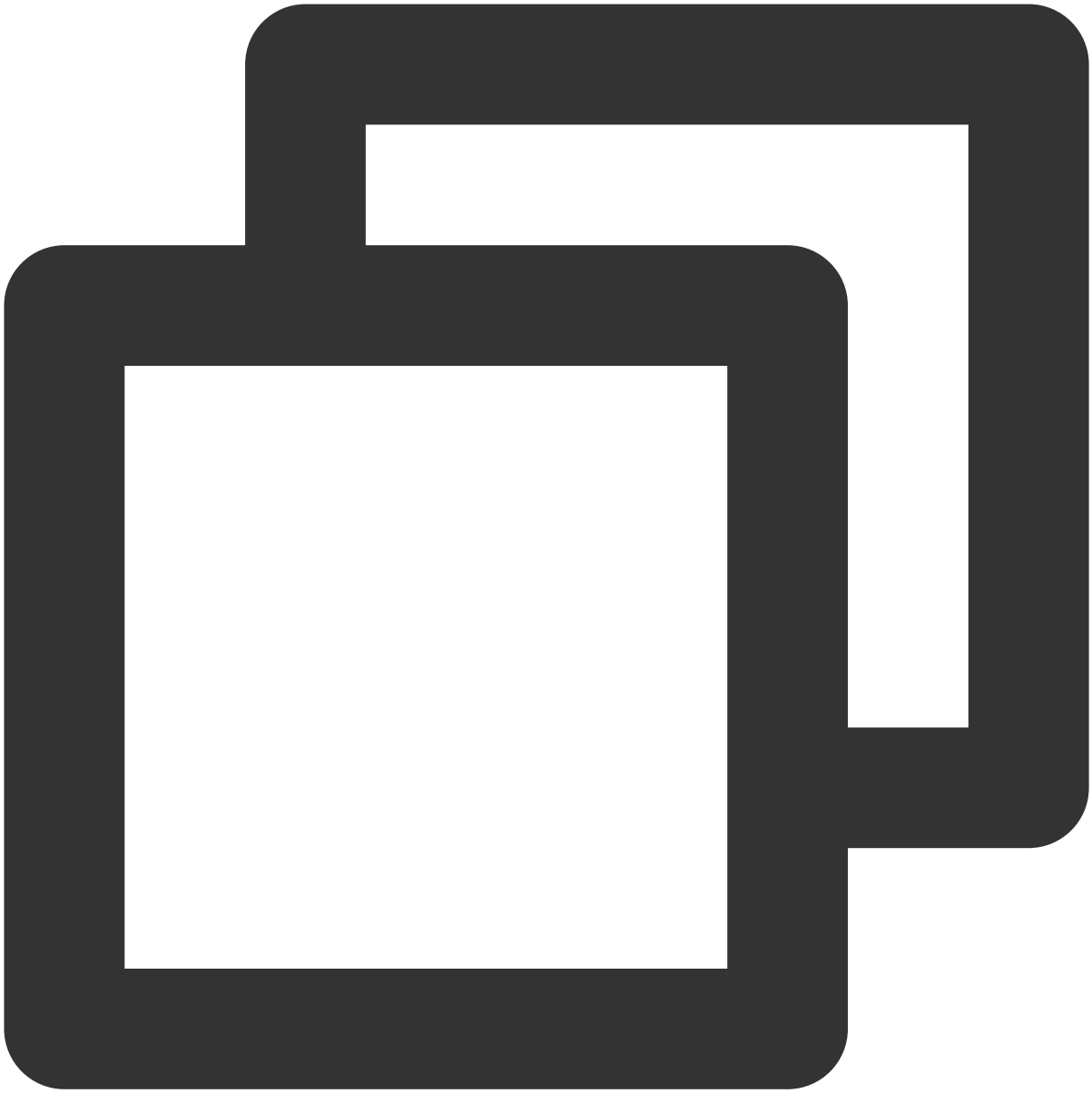


步骤3：引入 TUIKit 组件

3.1 在 `main.ts / main.js` 中，引入 TUIKit，并注册到 Vue 项目实例中

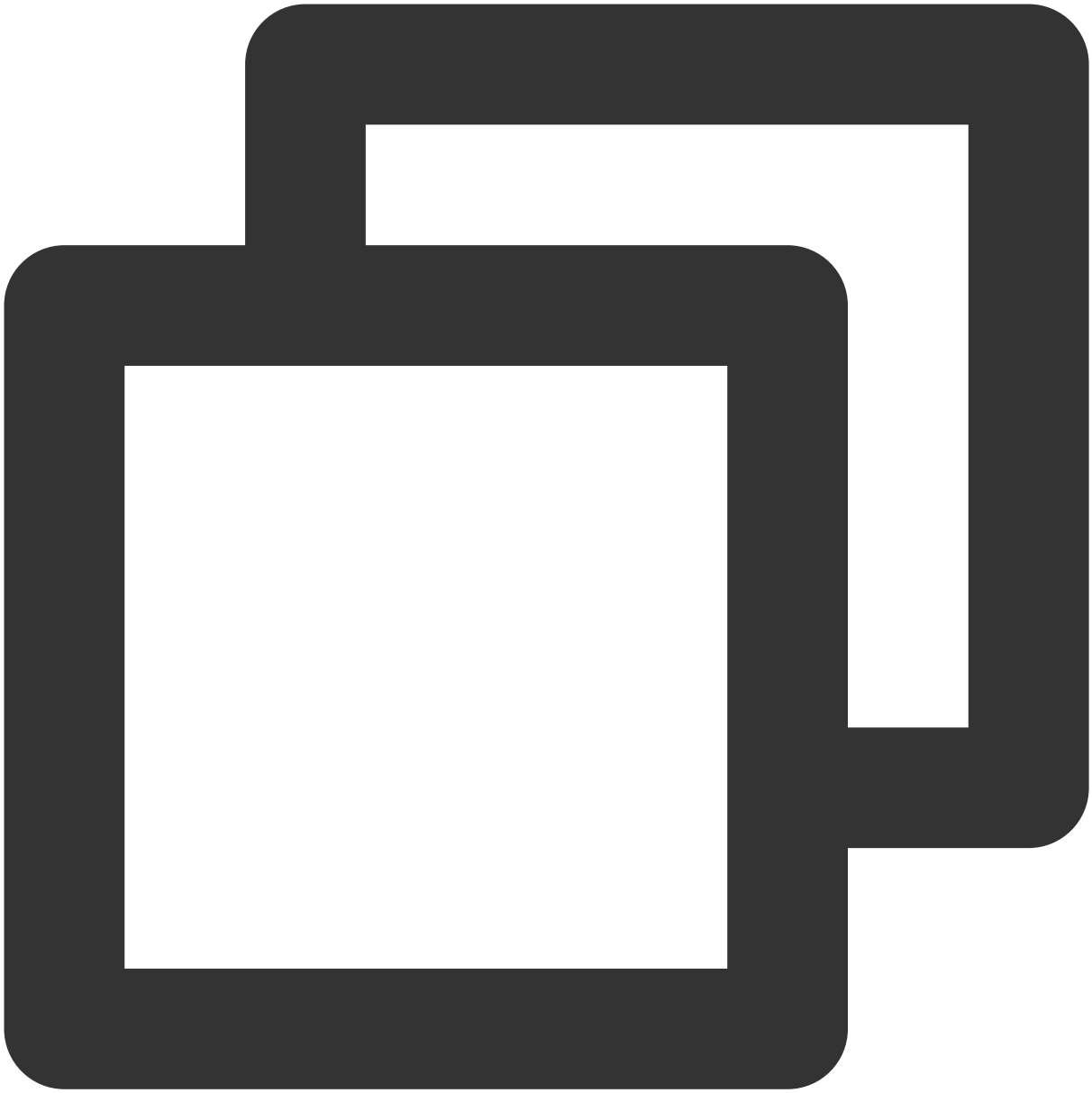
Vue3 版本

Vue2版本



```
import { createApp } from 'vue';
import App from './App.vue';
import { TUIComponents, TUIChatKit, genTestUserSig } from "./TUIKit";
import { TUILogin } from "@tencentcloud/tui-core";
const app = createApp(App);

const SDKAppID = 0; // Your SDKAppID
const secretKey = ""; //Your secretKey
const userID = ""; // User ID

// TUIChatKit add TUIComponents
```

```
TUIChatKit.components(TUIComponents, app);
// TUIChatKit init
TUIChatKit.init();
// TUICore login
TUILogin.login({
  SDKAppID,
  userID,
  // UserSig is a password used to log in to IM. It is the ciphertext obtained afte
  // this method is only suitable for locally running a demo project and feature de
  userSig: genTestUserSig({
    SDKAppID,
    secretKey,
    userID,
  }).userSig,
  useUploadPlugin: true,
  useProfanityFilterPlugin: false,
  framework: "vue3",
});

app.mount("#app");
export { SDKAppID, secretKey };
```



```
import Vue from "vue";
import App from "./App.vue";
import { TUIComponents, TUIChatKit, genTestUserSig } from "./TUIKit";
import { TUILogin } from "@tencentcloud/tui-core";

const SDKAppID = 0; // Your SDKAppID
const secretKey = ""; //Your secretKey
const userID = ""; // User ID

// TUIChatKit add TUIComponents
TUIChatKit.components(TUIComponents, Vue);
```

```
// TUIChatKit init
TUIChatKit.init();
// TUICore login
TUILogin.login({
  SDKAppID,
  userID,
  // UserSig is a password used to log in to IM. It is the ciphertext obtained afte
  // this method is only suitable for locally running a demo project and feature de
  userSig: genTestUserSig({
    SDKAppID,
    secretKey,
    userID,
  }).userSig,
  useUploadPlugin: true,
  // 本地审核可识别、处理不安全、不适宜的内容，为您的产品体验和业务安全保驾护航
  // 此功能为增值服务，请参考：https://cloud.tencent.com/document/product/269/79139
  // 如果您已购买内容审核服务，开启此功能请设置为 true
  useProfanityFilterPlugin: false,
  framework: "vue2",
});
Vue.config.productionTip = false;
  new Vue({
    render: (h) => h(App),
  }).$mount("#app");
export { SDKAppID, secretKey };
```

3.2 在页面中调用 TUIKit 组件

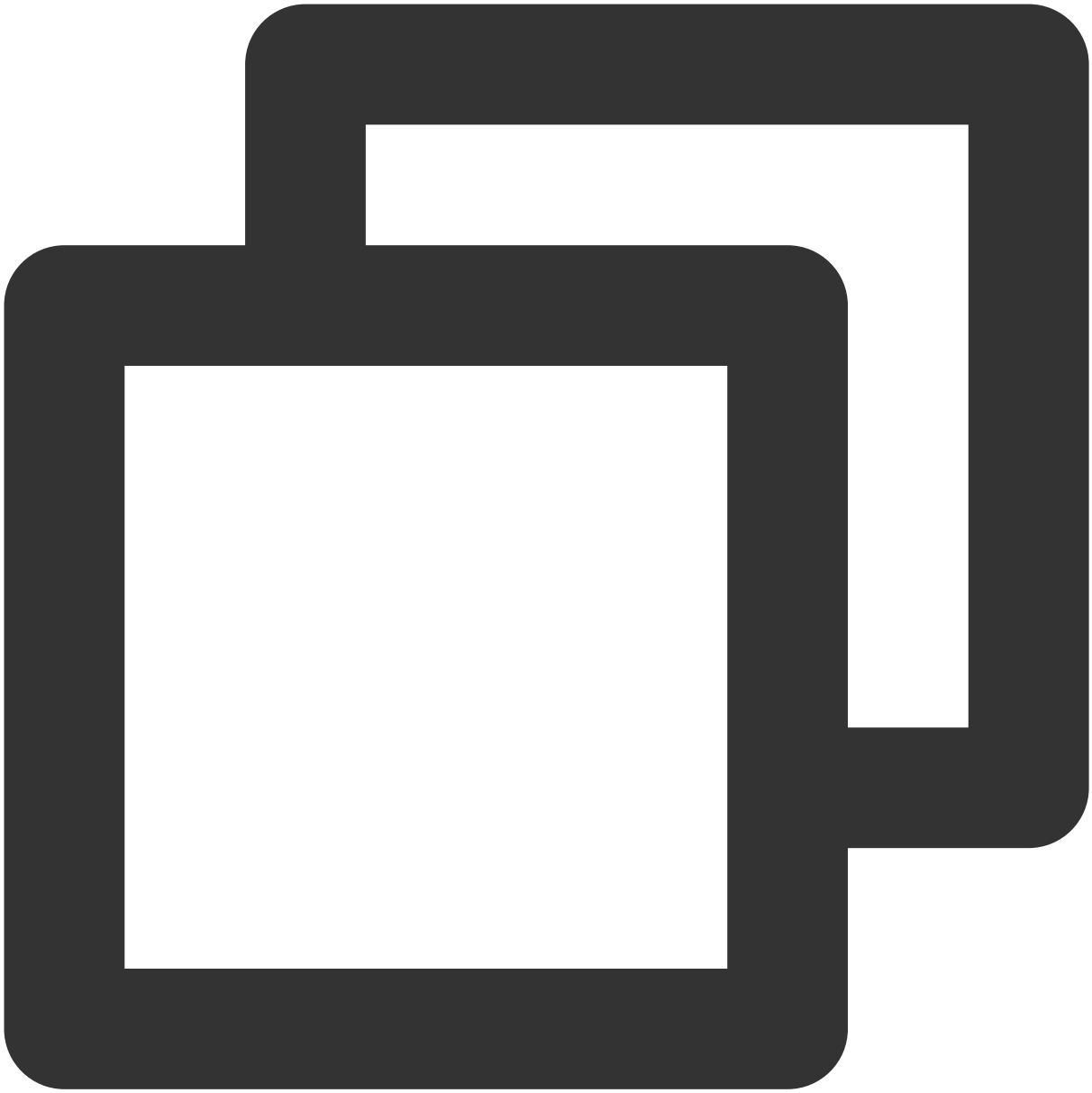
Vue3 版本

Vue2.7 版本

Vue2.6 及以下版本

在需要展示的页面，调用 TUIKit 的组件即可使用。

例如：在 App.vue 页面中，使用 TUIConversation、TUIChat、TUIContact、TUISearch、TUIGroup、TUICallKit 快速搭建聊天界面（以下示例代码同时支持 Web 端与 H5 端）



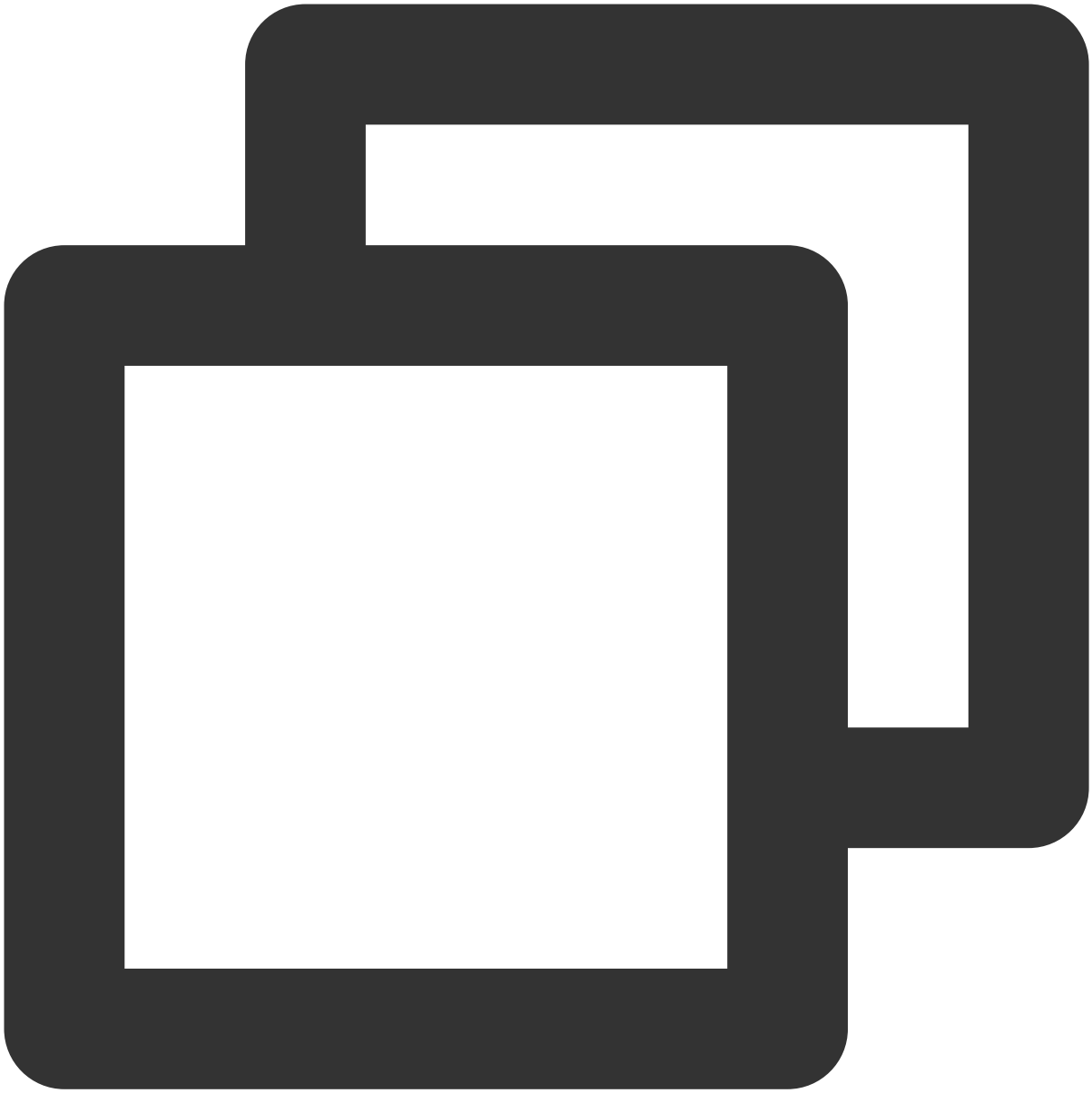
```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar = item.id"
      >
        {{ item.label }}
      </div>
```

```
    </div>
    <div class="TUIKit-main-container">
      <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
        <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
          <TUISearch searchType="global"></TUISearch>
          <TUIConversation></TUIConversation>
        </div>
        <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
          <TUIChat>
            <h1>Welcome Chat</h1>
          </TUIChat>
          <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
          <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
        </div>
        <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
        <TUIContact displayType="selectFriend" />
      </div>
      <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
        <TUIContact
          displayType="contactList"
          @switchConversation="currentNavbar = 'conversation'"
        />
      </div>
      <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
    </div>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
const currentConversationID = ref<string>("");
const currentNavbar = ref<string>("conversation");
const navbarList = [
  {
    id: "conversation",
    label: "Conversation",
  },
  {
    id: "contact",
    label: "Contact",
  },
];
TUIStore.watch(StoreName.CONV, {
  currentConversationID: (id: string) => {
```

```
    currentConversationID.value = id;
  },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

在需要展示的页面，调用 TUIKit 的组件即可使用。

例如：在 App.vue 页面中，使用 TUIConversation、TUIChat、TUIContact、TUISearch、TUIGroup、TUICallKit 快速搭建聊天界面（以下示例代码同时支持 Web 端与 H5 端）

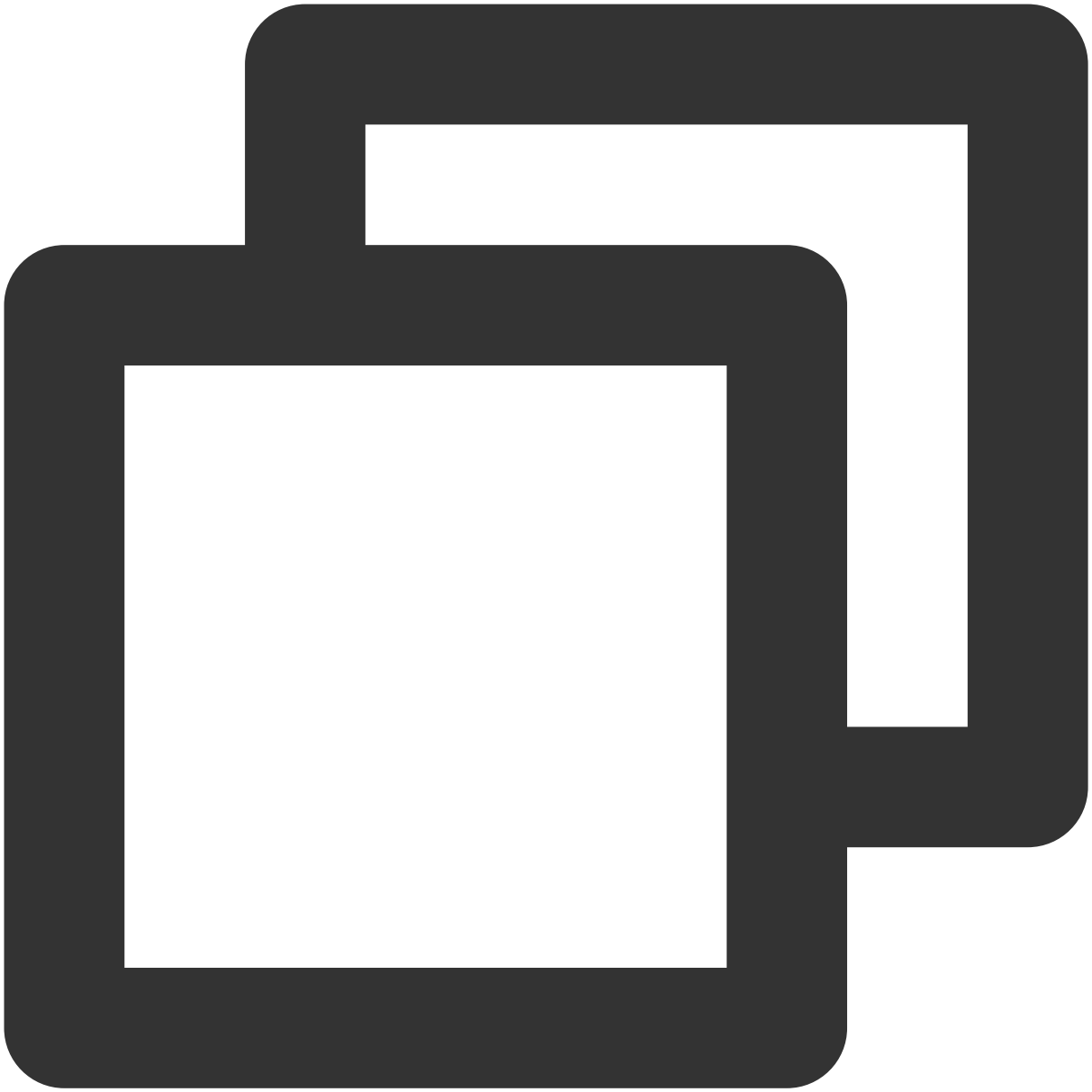


```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar = item.id"
      >
        {{ item.label }}
      </div>
```

```
    </div>
    <div class="TUIKit-main-container">
      <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
        <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
          <TUISearch searchType="global"></TUISearch>
          <TUIConversation></TUIConversation>
        </div>
        <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
          <TUIChat>
            <h1>Let's Chat!</h1>
          </TUIChat>
          <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
          <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
        </div>
        <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
        <TUIContact displayType="selectFriend" />
      </div>
      <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
        <TUIContact
          displayType="contactList"
          @switchConversation="currentNavbar = 'conversation'"
        />
      </div>
      <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
    </div>
  </div>
</template>
<script lang="ts">
import Vue from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue2";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
export default Vue.extend({
  name: "App",
  components: {
    TUISearch,
    TUIGroup,
    TUIConversation,
    TUIChat,
    TUIContact,
    TUICallKit,
  },
  data() {
    return {
      isH5: isH5,
      currentConversationID: "",
```

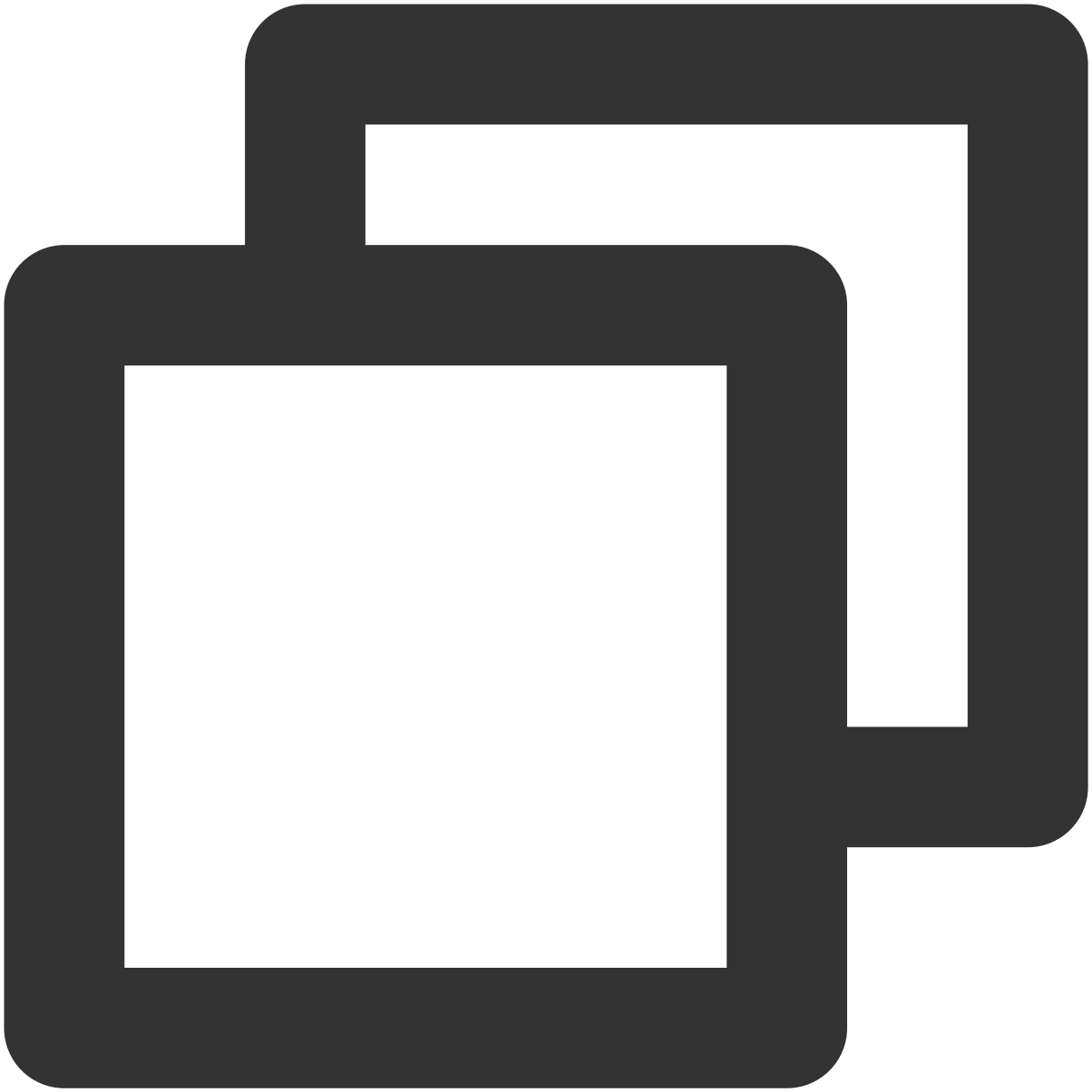
```
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: ",
        },
      ],
    };
  },
  mounted: function () {
    TUIStore.watch(StoreName.CONV, {
      currentConversationID: (id: string) => {
        this.currentConversationID = id;
      },
    });
  },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

1. 安装支持 composition-api 以及 script setup 的相关依赖，以及 vue2.6 相关依赖：



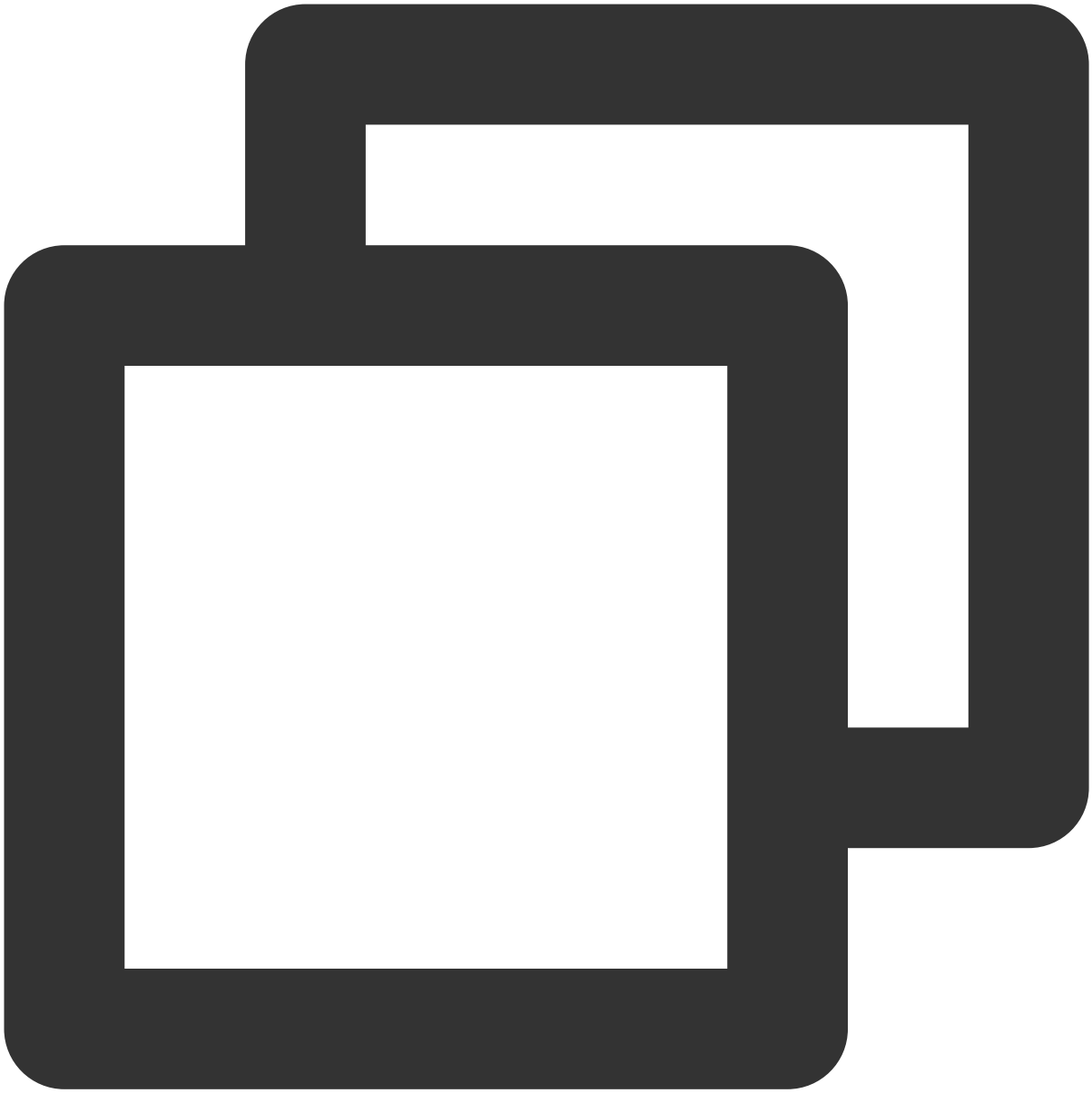
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

2. main.ts 中引入 VueCompositionAPI



```
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
```

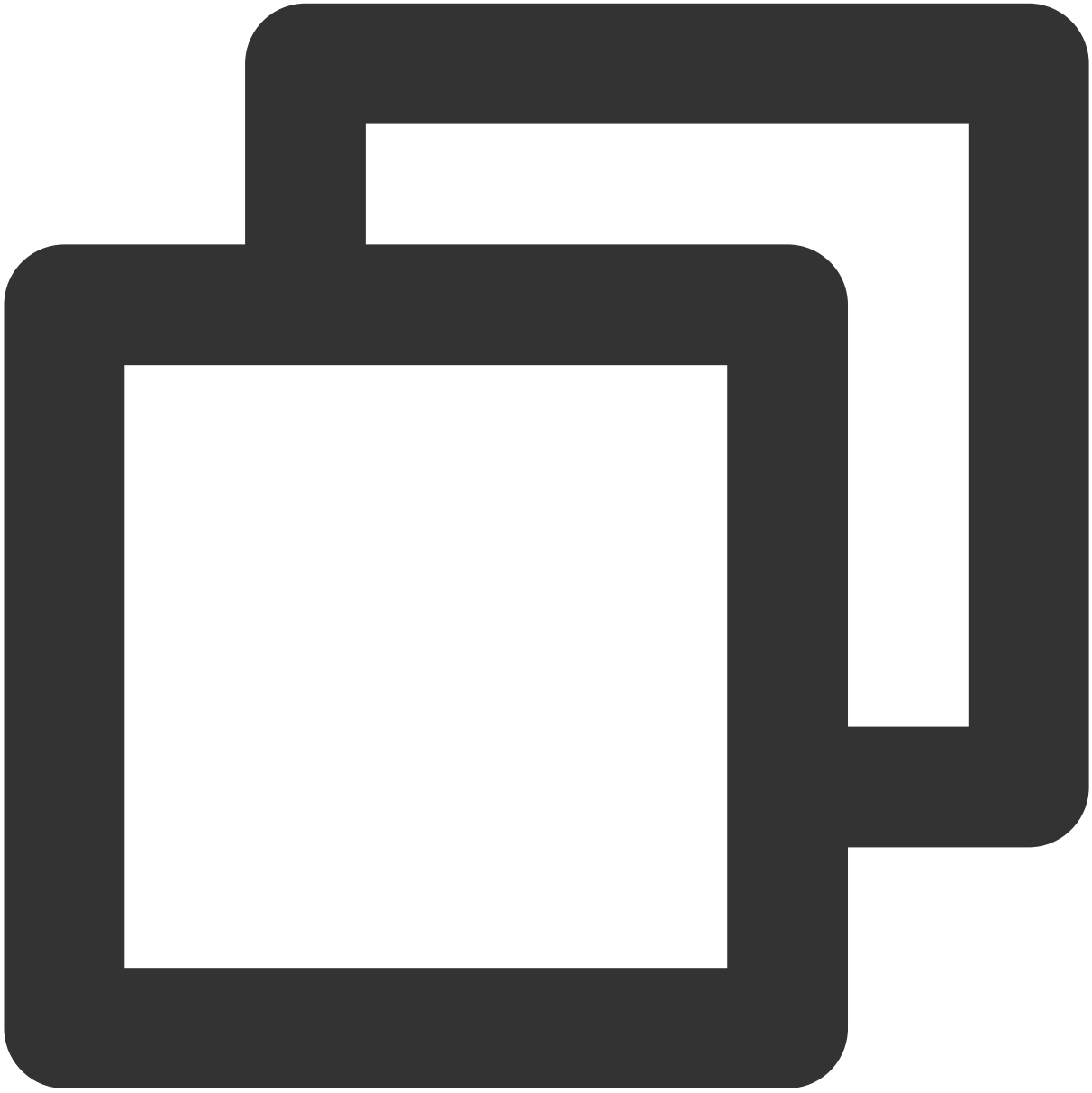
3. 在 vue.config.js 中增加，若没有该文件请新建：



```
const ScriptSetup = require("unplugin-vue2-script-setup/webpack").default;
module.exports = {
  parallel: false, // disable thread-loader, which is not compactible with this plu
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
```

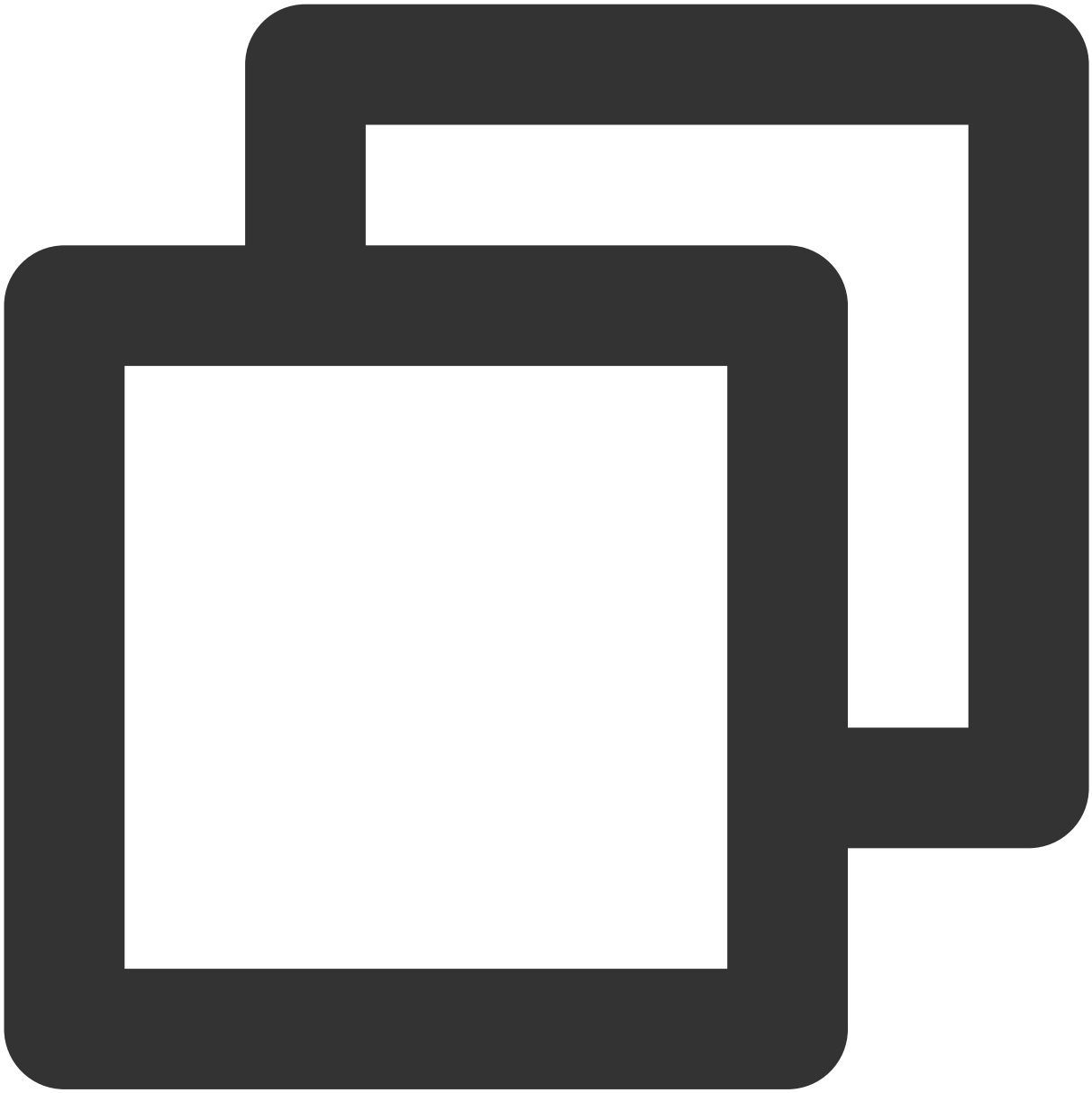
```
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete("fork-ts-checker");
  },
};
```

4. 在 src/TUIKit/adapter-vue.ts 文件最后, 替换导出源：



```
// 初始写法
export * from "vue";
// 替换为
export * from "@vue/composition-api";
```

5. 在需要展示的页面，调用 TUIKit 的组件即可使用。
例如：在 App.vue 页面中，使用 TUIConversation、TUIChat、TUIContact、TUISearch、TUIGroup、TUICallKit 快速搭建聊天界面（以下示例代码同时支持 Web 端与 H5 端）



```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
```

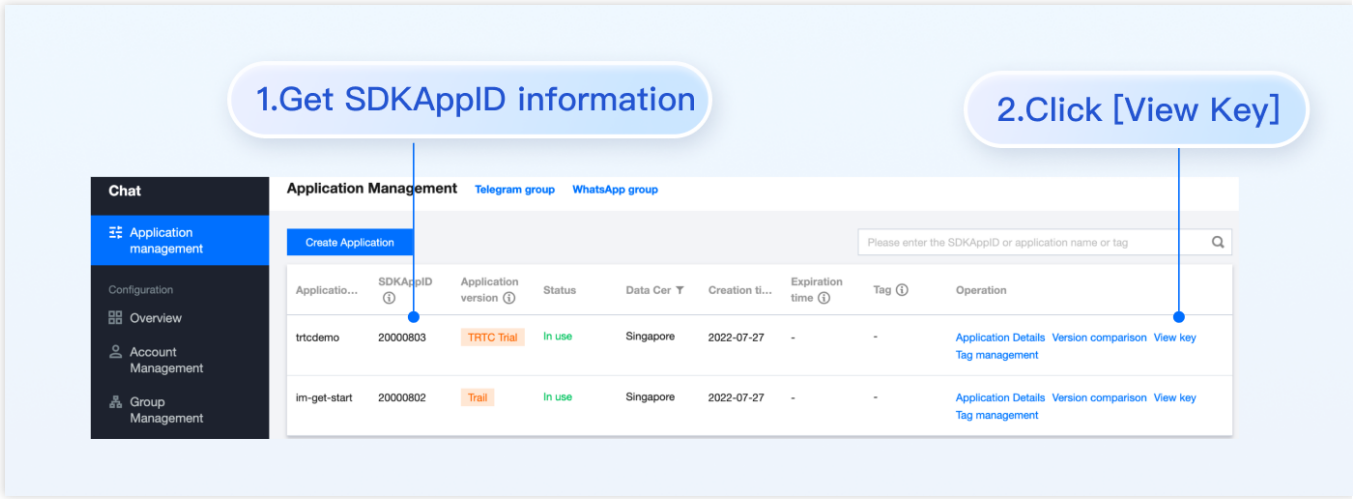
```
        @click="currentNavbar = item.id"
      >
        {{ item.label }}
      </div>
    </div>
    <div class="TUIKit-main-container">
      <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
        <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
          <TUISearch searchType="global"></TUISearch>
          <TUIConversation></TUIConversation>
        </div>
        <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
          <TUIChat>
            <h1>Let's Chat!</h1>
          </TUIChat>
          <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
          <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
        </div>
        <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
        <TUIContact displayType="selectFriend" />
      </div>
      <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
        <TUIContact
          displayType="contactList"
          @switchConversation="currentNavbar = 'conversation'"
        />
      </div>
      <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
    </div>
  </div>
</template>
<script lang="ts">
import Vue from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue2.6";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
export default Vue.extend({
  name: "App",
  components: {
    TUISearch,
    TUIGroup,
    TUIConversation,
    TUIChat,
    TUIContact,
    TUICallKit,
  },
```

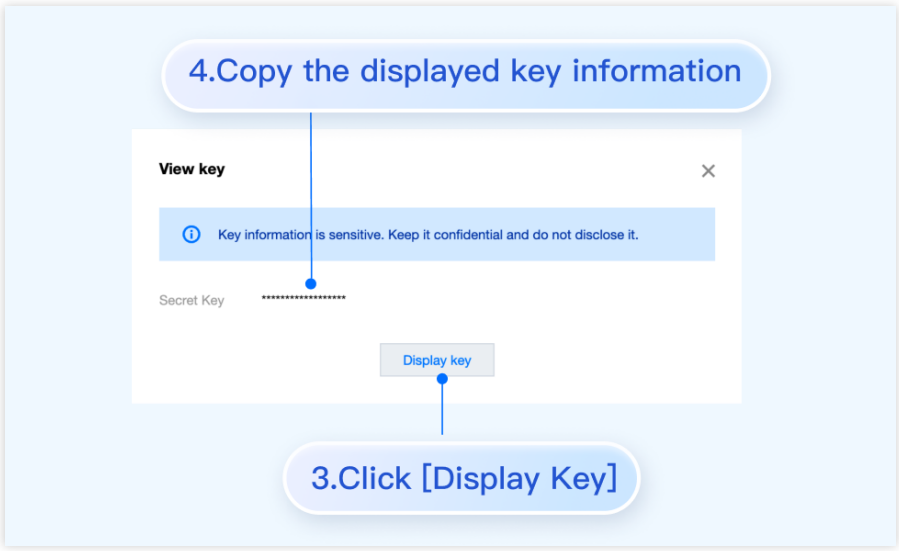
```
  data() {
    return {
      isH5: isH5,
      currentConversationID: "",
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: "Contact",
        },
      ],
    };
  },
  mounted: function () {
    TUIStore.watch(StoreName.CONV, {
      currentConversationID: (id: string) => {
        this.currentConversationID = id;
      },
    });
  },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

步骤4： 获取 SDKAppID 、secretKey 与 userID

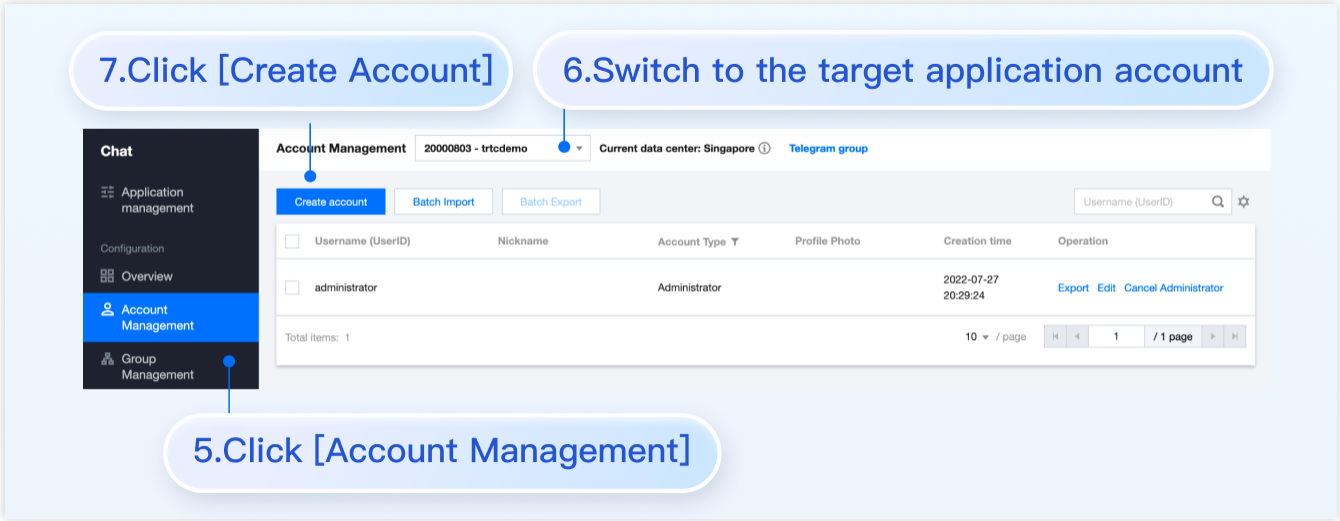
设置 main.ts / main.js 文件示例代码中的相关参数 SDKAppID、secretKey 以及 userID :

SDKAppID 和 secretKey 等信息，可通过 即时通信 IM 控制台 获取:





userID 信息，可单击 即时通信 IM 控制台 > 账号管理，切换至目标应用所在账号，即可创建账号并获取 userID。

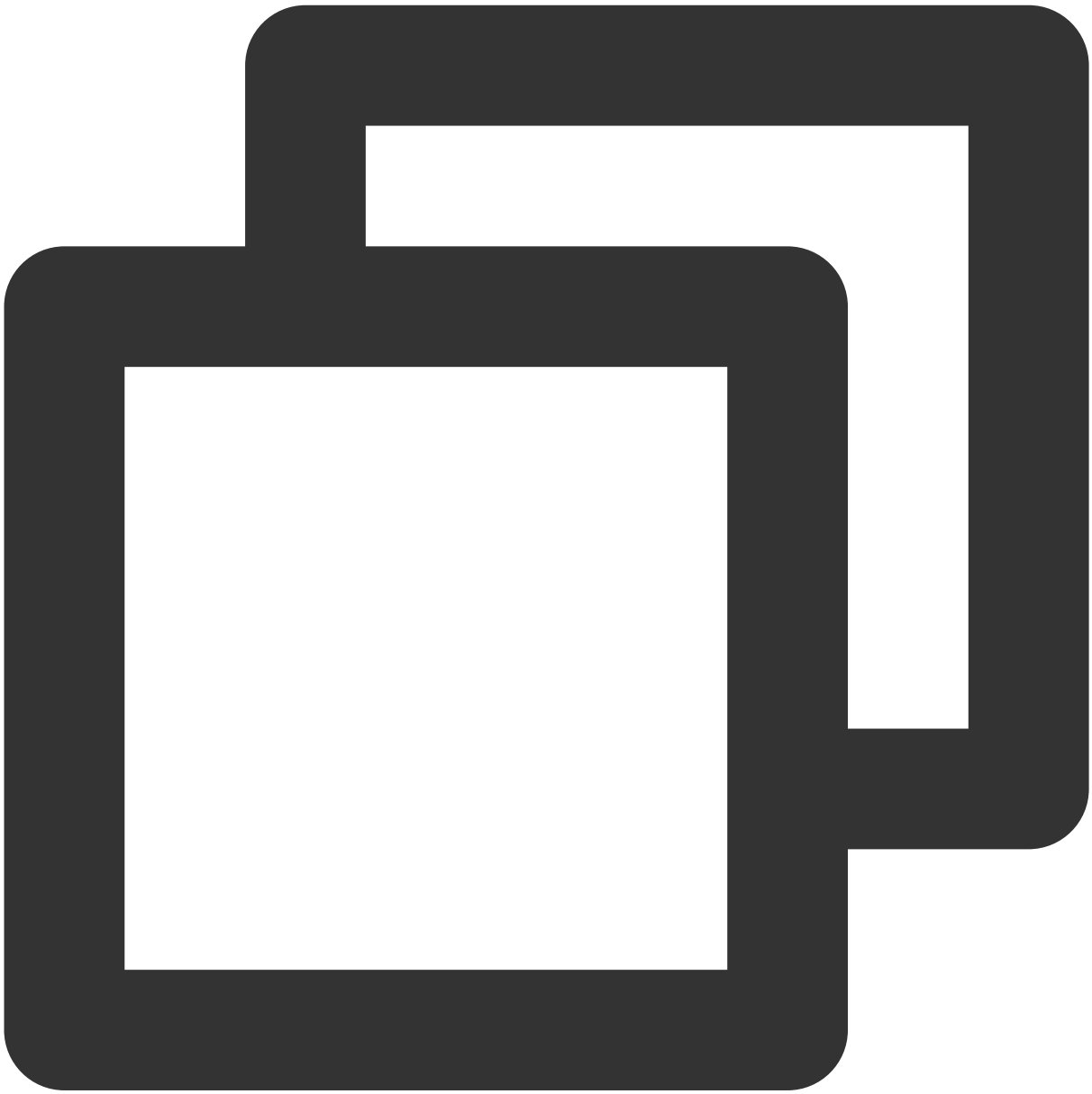


步骤5：启动项目

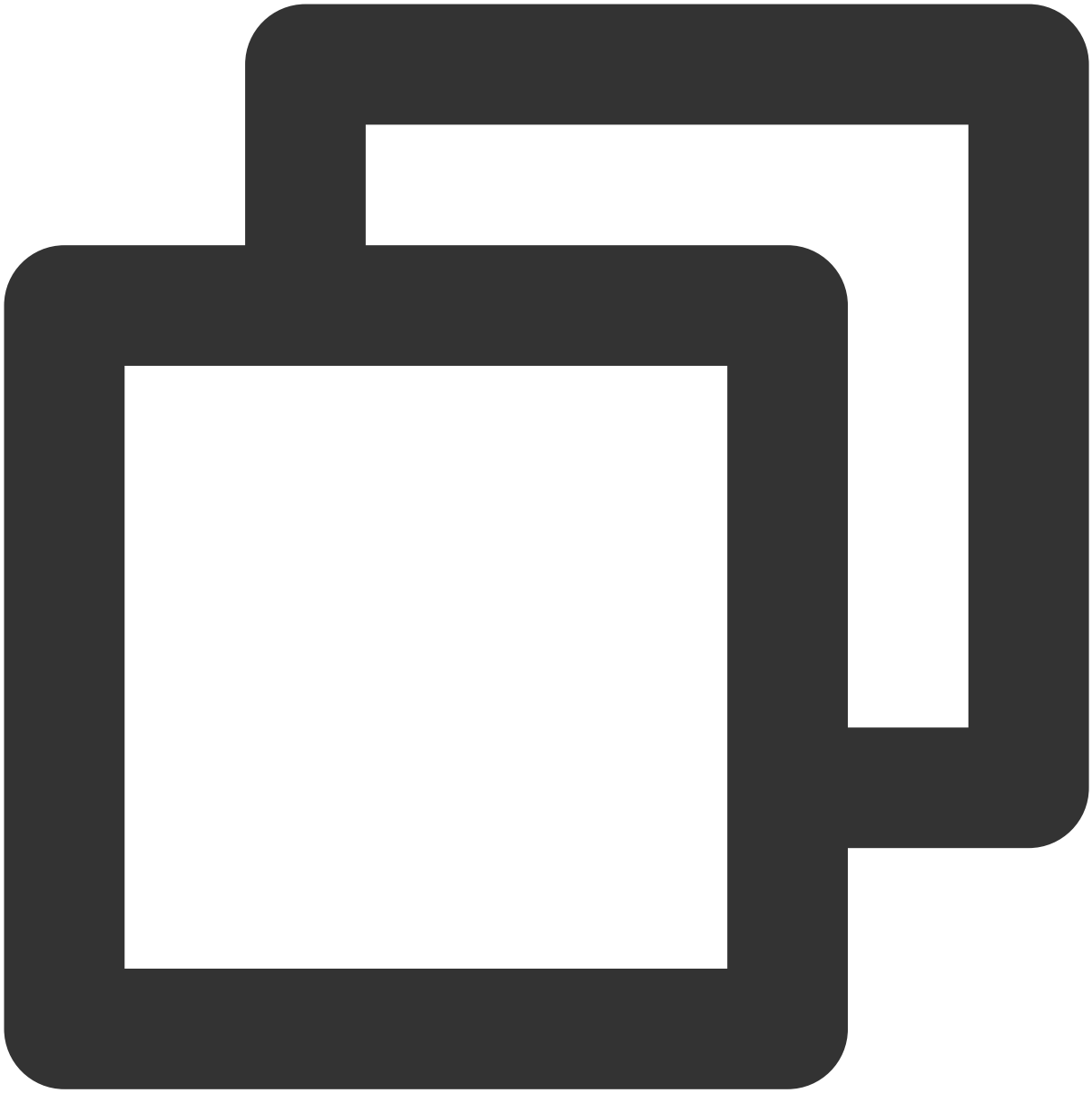
执行以下命令启动项目：

vue-cli

vite



```
npm run serve
```

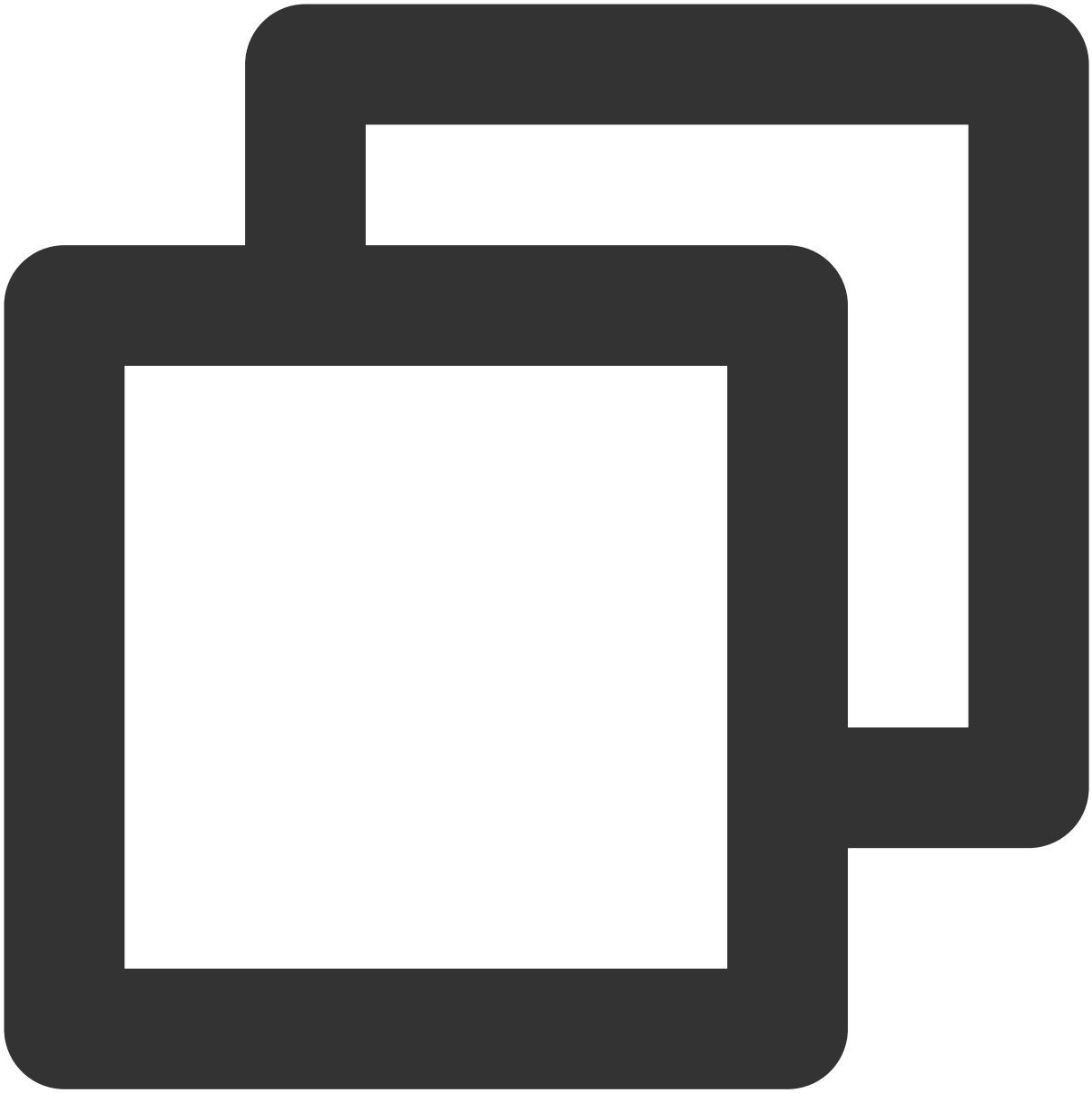


```
npm run dev
```

附加项：切换语言

Web & H5 端 `Vue TUIKit` 默认自带 **简体中文、英语** 语言包，作为界面展示语言。

您可以通过以下方式切换语言。



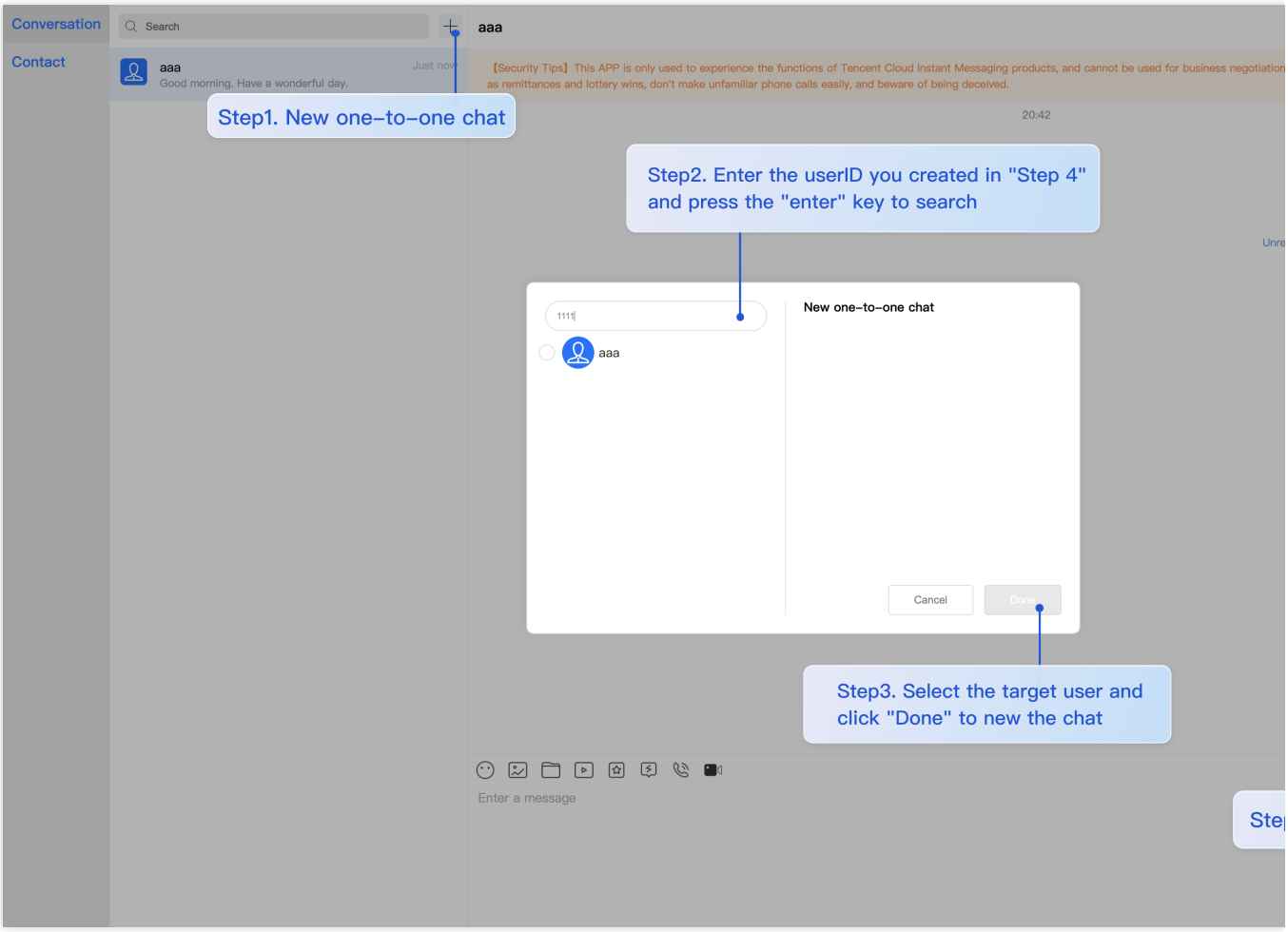
```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

步骤6：发送您的第一条消息

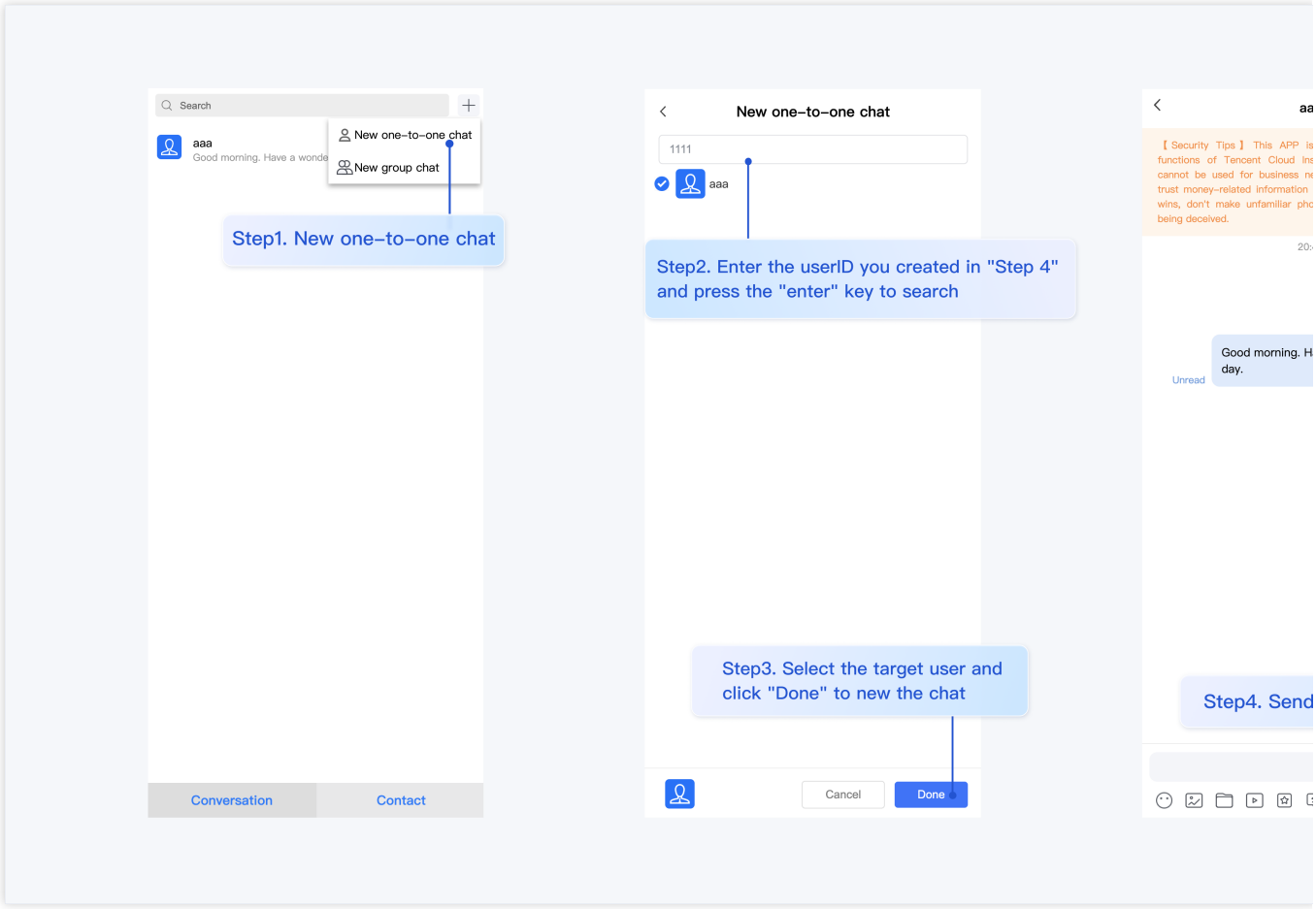
1. 项目启动之后单击左上角**发起单聊**。
2. 进入**发起单聊**弹窗。在搜索栏输入 步骤4 中创建的 userID，选中后单击**完成**。

3. 在输入框中输入消息并单击**发送**。

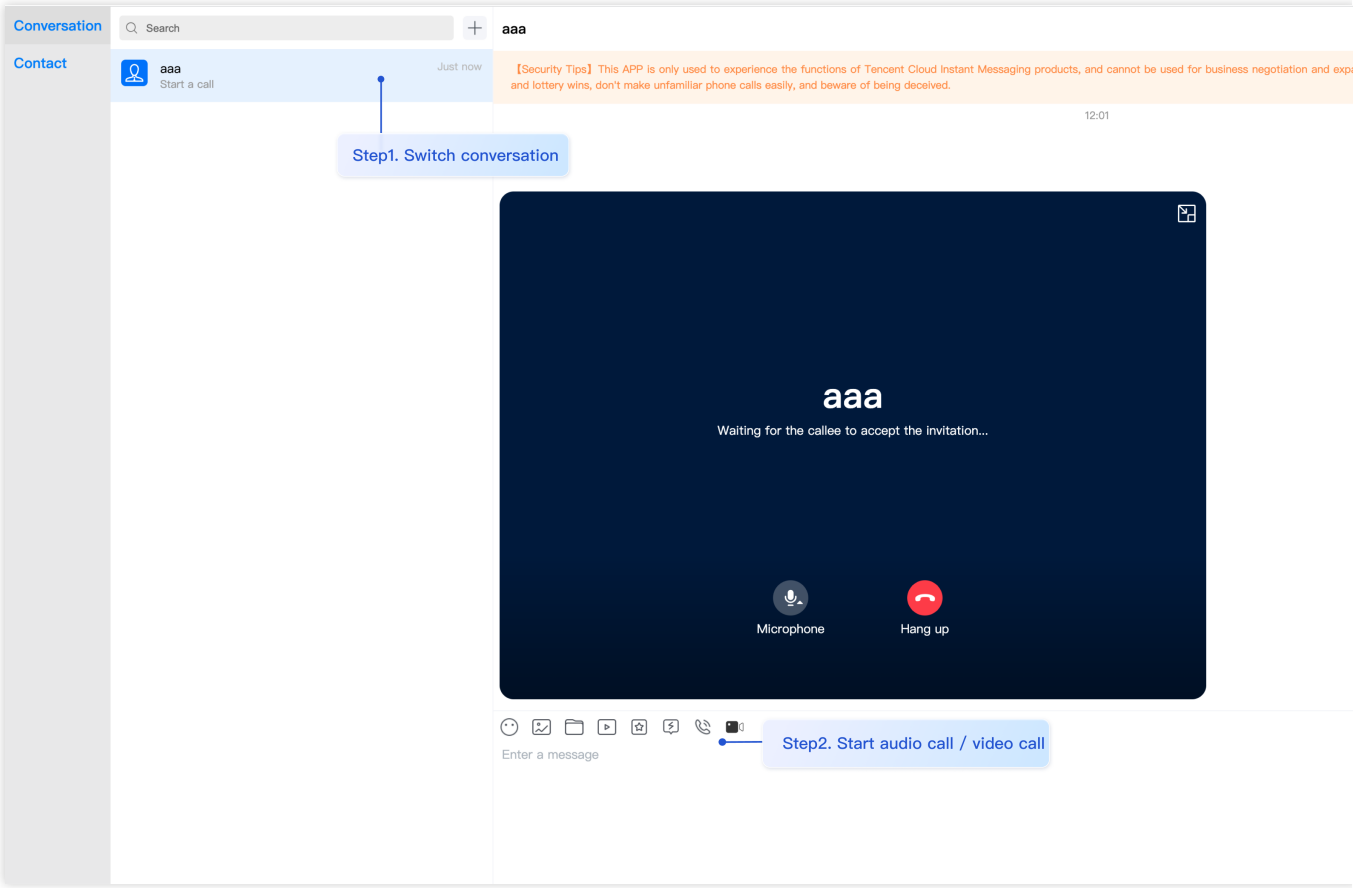
Web 端 “发送您的第一条消息” 具体步骤示例：



H5 端 “发送您的第一条消息” 具体步骤示例：



步骤7：拨打您的第一通电话



常见问题

产品服务类问题

1. 音视频通话能力包未开通？音视频通话发起失败？

请单击 音视频通话 > 常见问题 查看解决方案。

2. 什么是 UserSig？如何生成 UserSig？

UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。

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。

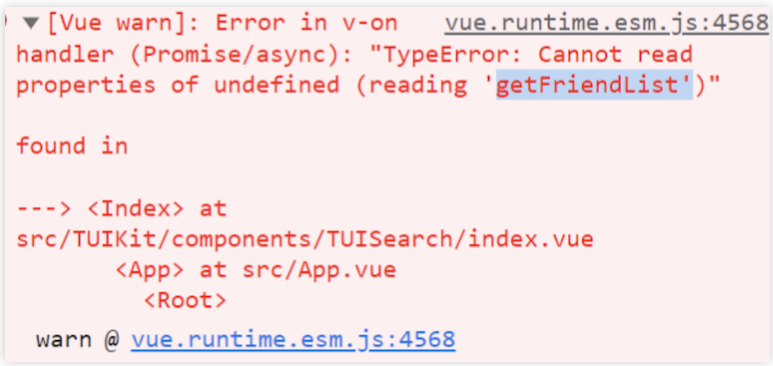
注意

本文示例代码采用的获取 UserSig 的方案是在客户端代码中配置 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**。正确的 UserSig 签发方式请参见上文。

接入报错类问题

1. 运行时报错："TypeError: Cannot read properties of undefined (reading "getFriendList")"

若按照上述步骤接入后，运行时出现以下错误，请您务必**删除 TUIKit 目录下的 node_modules 目录**，以保证 TUIKit 的依赖唯一性，避免 TUIKit 多份依赖造成问题。



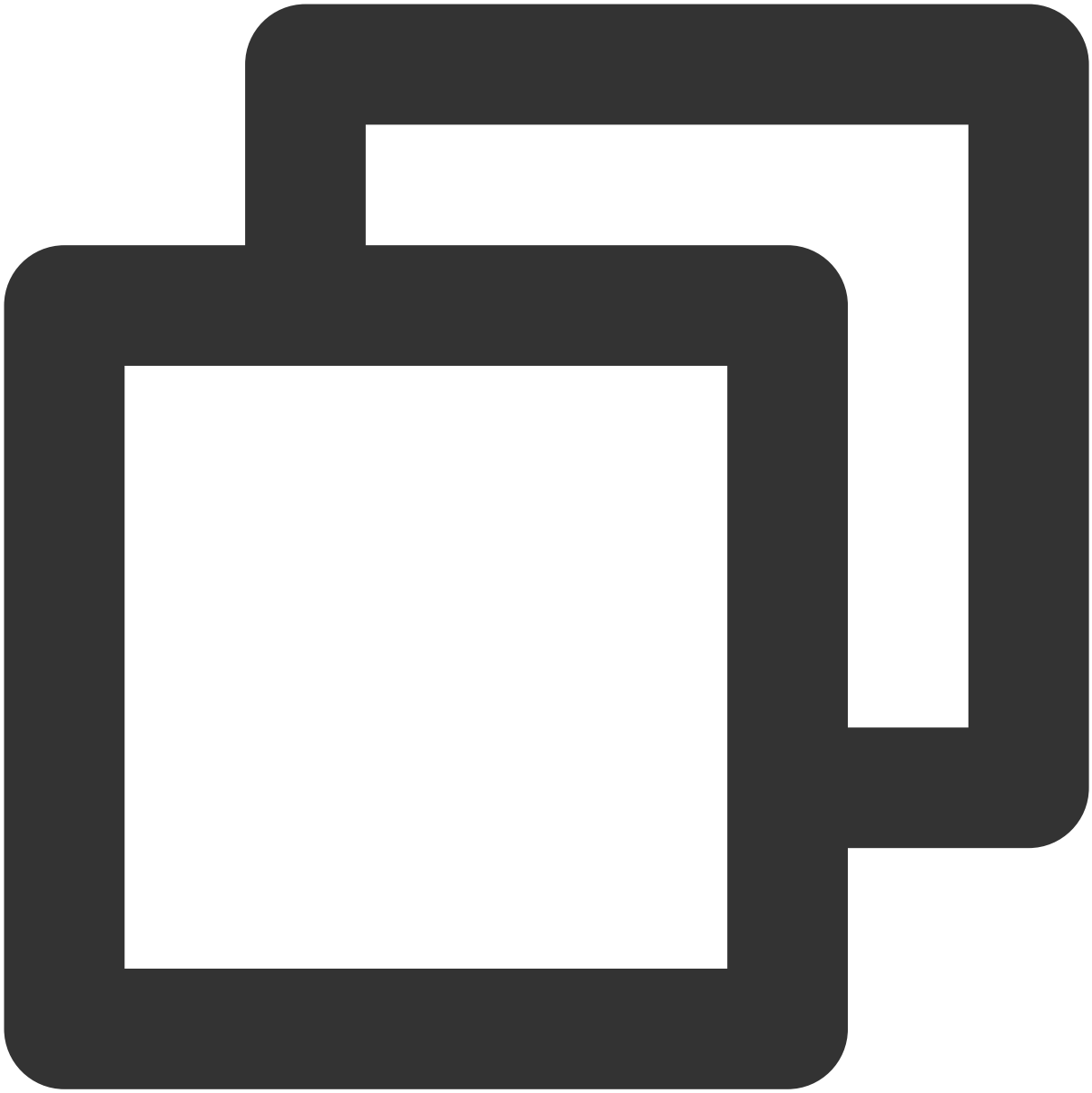
2. js 工程如何接入 TUIKit 组件?

TUIKit 仅支持 ts 环境运行，您可以通过渐进式配置 typescript 来使您项目中已有的 js 代码 与 TUIKit 中 ts 代码共存。

vue-cli

vite

请在您 vue-cli 脚手架创建的工程根目录执行：



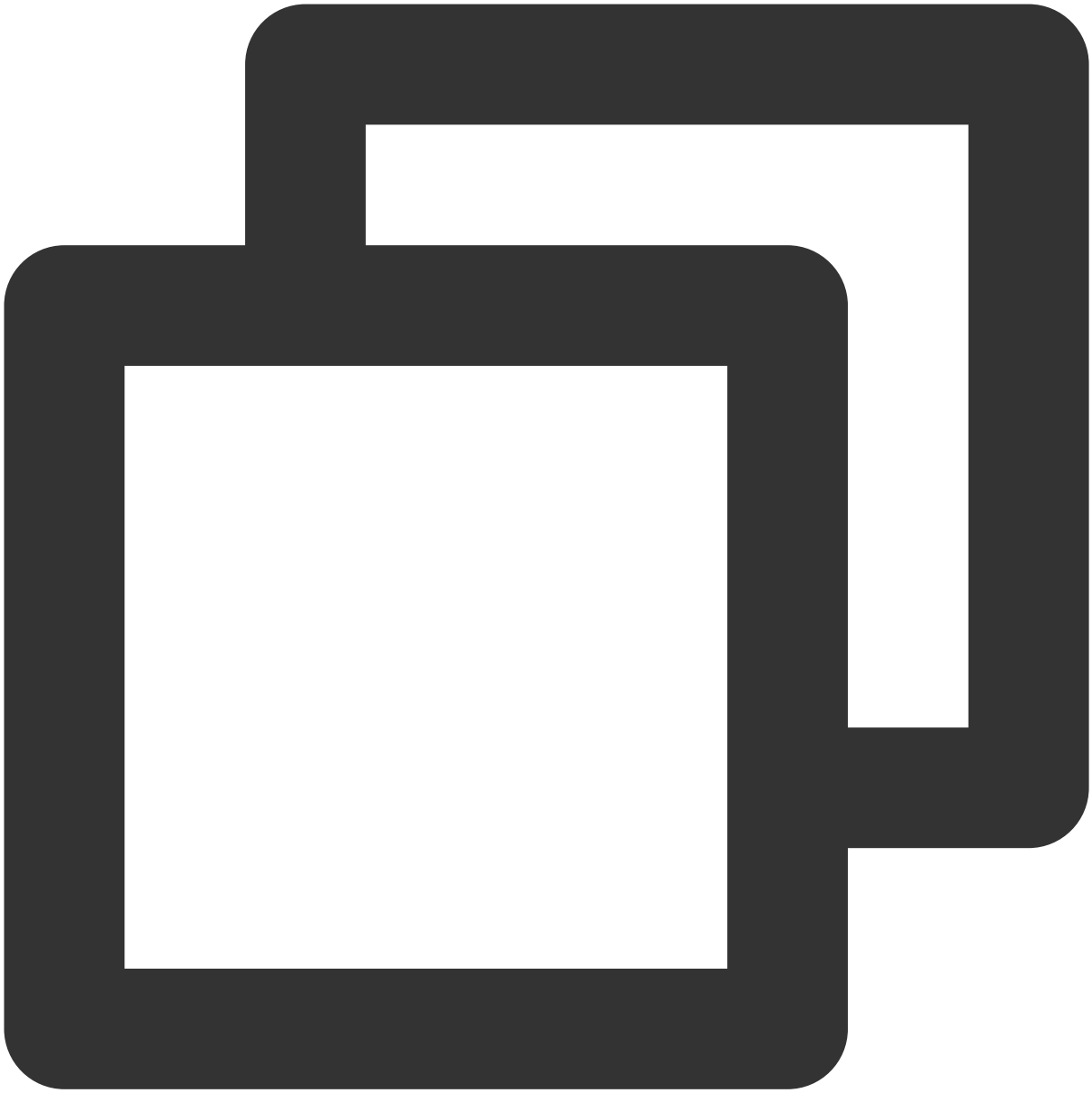
```
vue add typescript
```

之后按照如下进行配置项进行选择（为了保证能同时支持原有 js 代码 与 TUIKit 中 ts 代码，请您务必严格按照以下五个选项进行配置）

```
Run `npm audit` for details.
✔  Successfully installed plugin: @vue/cli-plugin-typescript

? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpi
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? Yes


🚀  Invoking generator for @vue/cli-plugin-typescript...
📦  Installing additional dependencies...
```

完成以上步骤后，请重新运行项目！

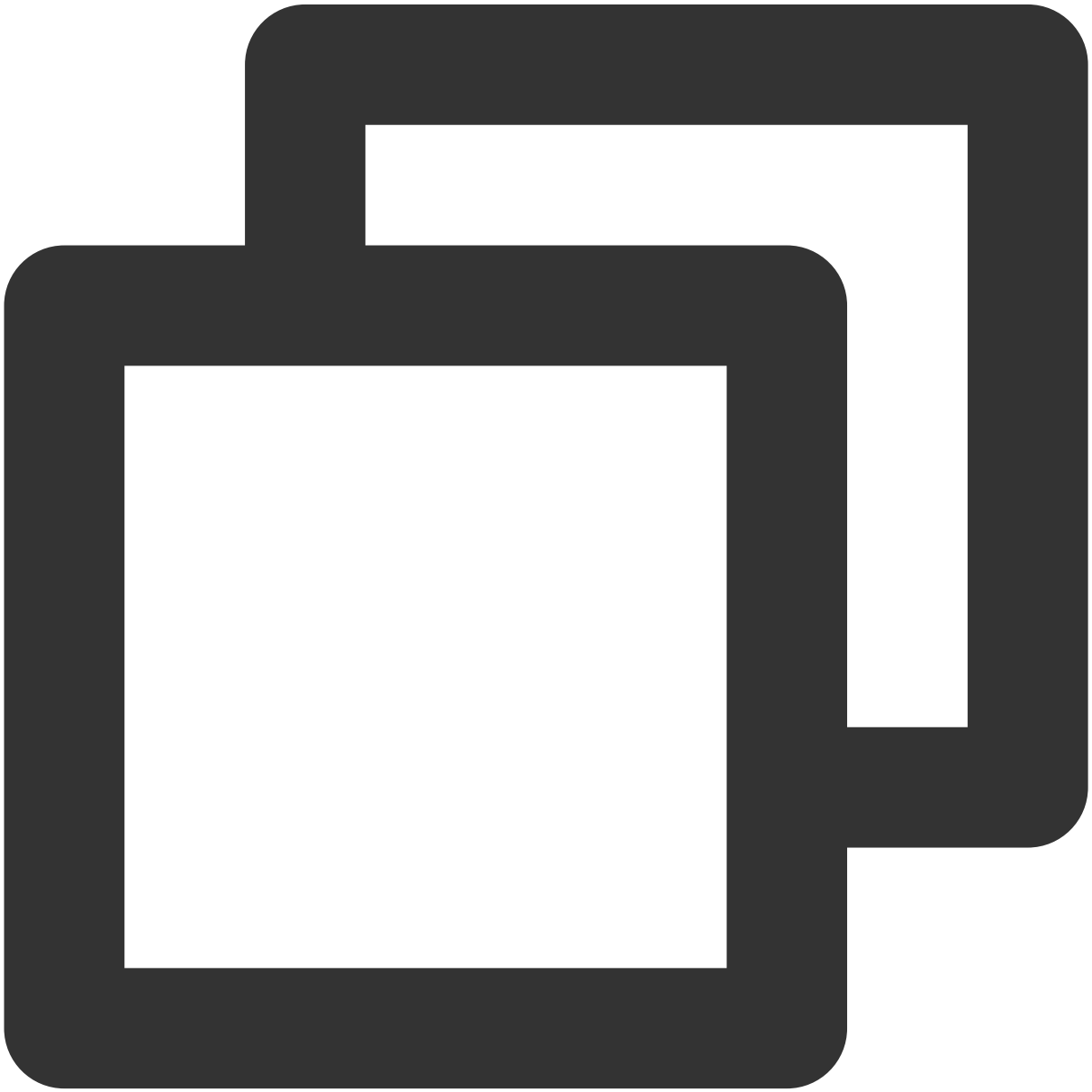
请在您 vite 创建的工程根目录执行：



```
npm install -D typescript
```

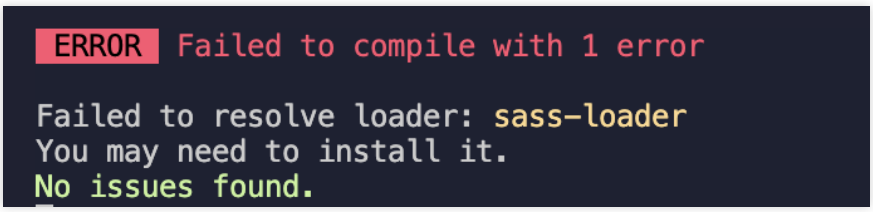
3. 运行时报错：/chat-example/src/TUIKit/components/TUIChat /message-input/message-input-editor.vue .ts（8，23）TS1005: expected.

出现以上报错信息，是因为您安装的 @vue/cli 版本过低导致，**请您务必保证您的 @vue/cli 版本在 5.0.0 及以上**。升级方式如下：

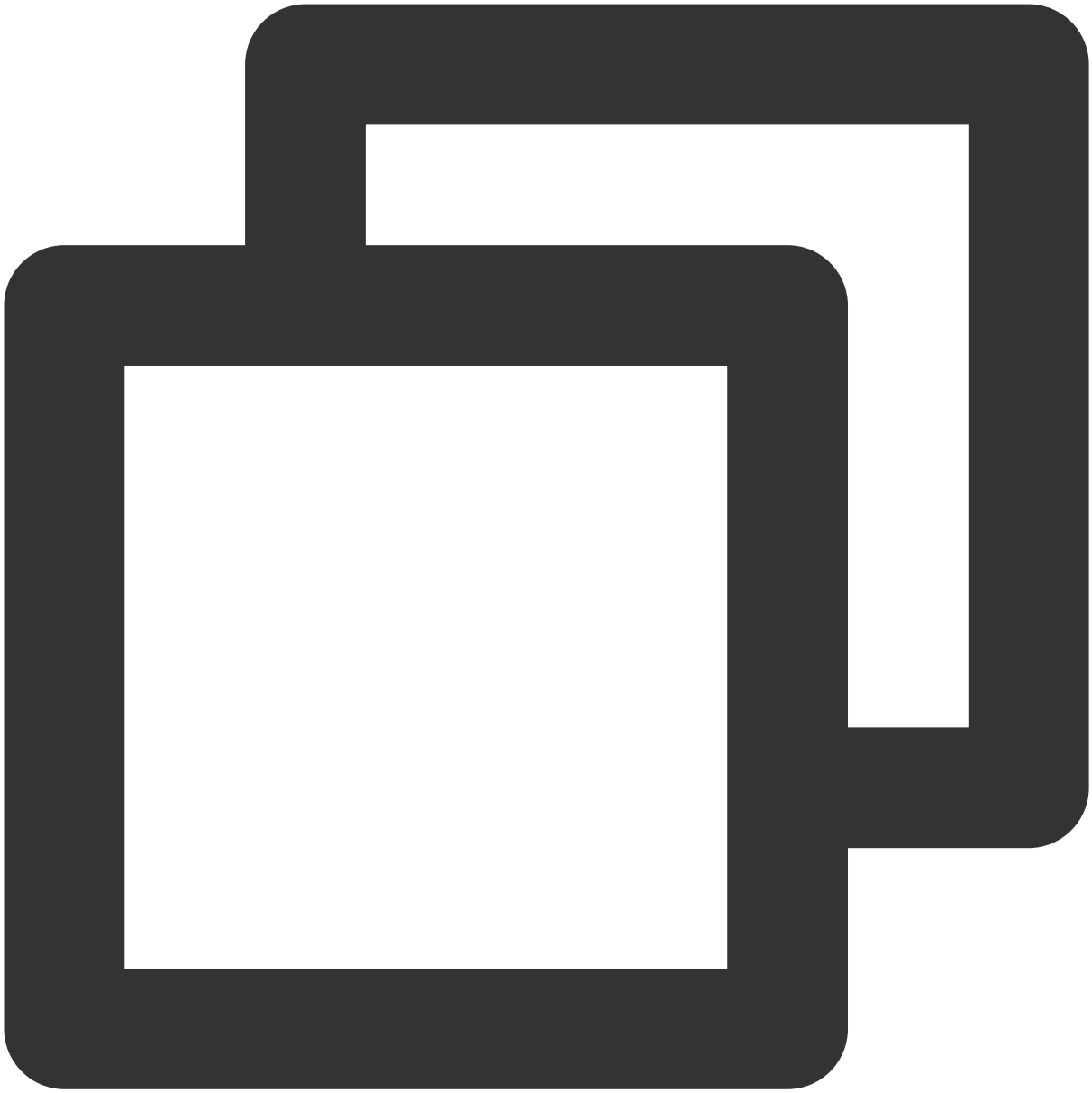


```
npm install -g @vue/cli@5.0.8
```

4. 运行时报错: Failed to resolve loader: sass-loader



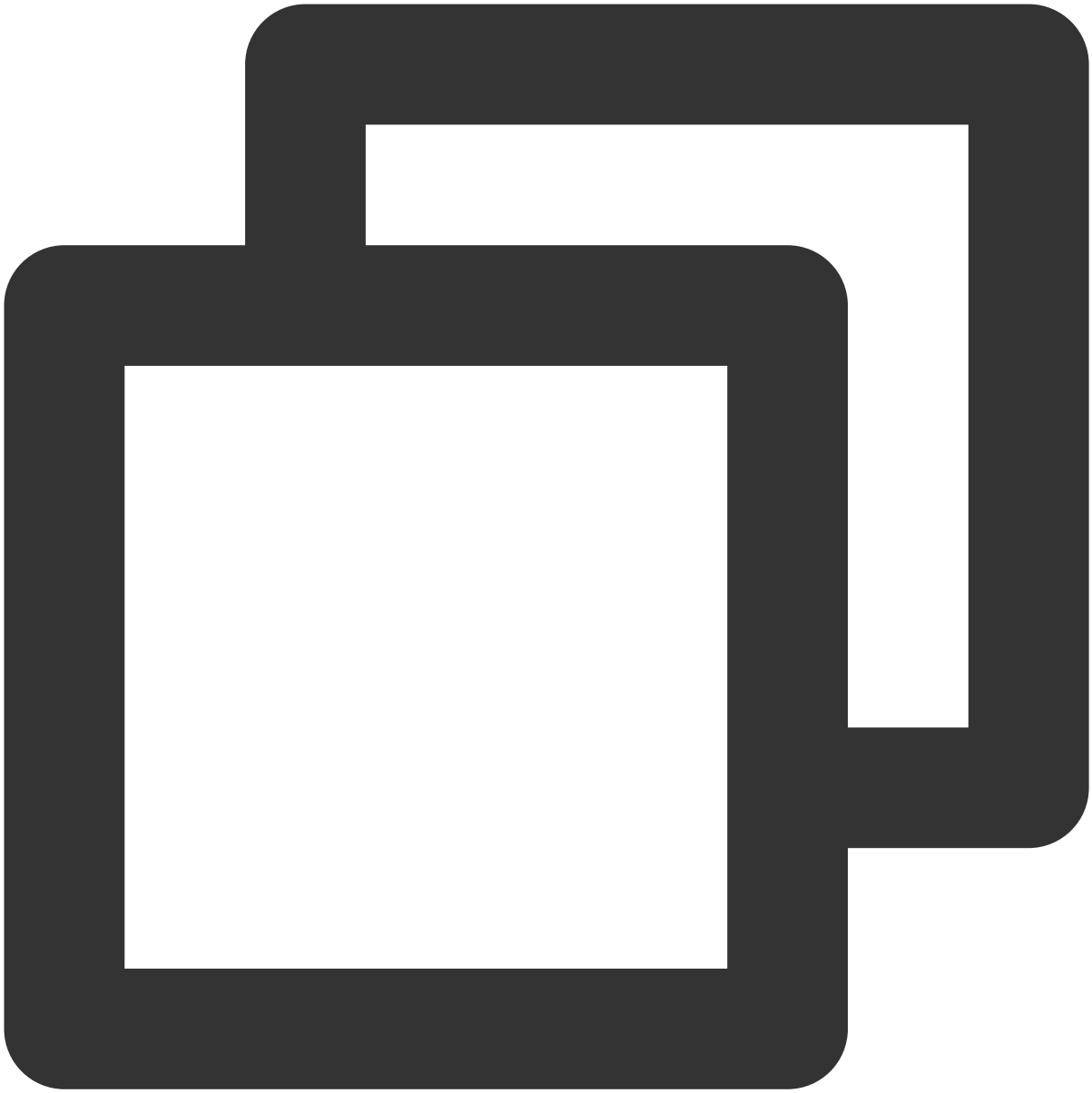
出现以上报错信息，是因为您未安装 `sass` 环境导致，请执行以下命令进行 `sass` 环境安装:



```
npm i -D sass sass-loader@10.1.1
```

5. ESLint 其他报错？

若 chat-uikit-vue 拷贝到 src 目录汇总与您本地项目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：



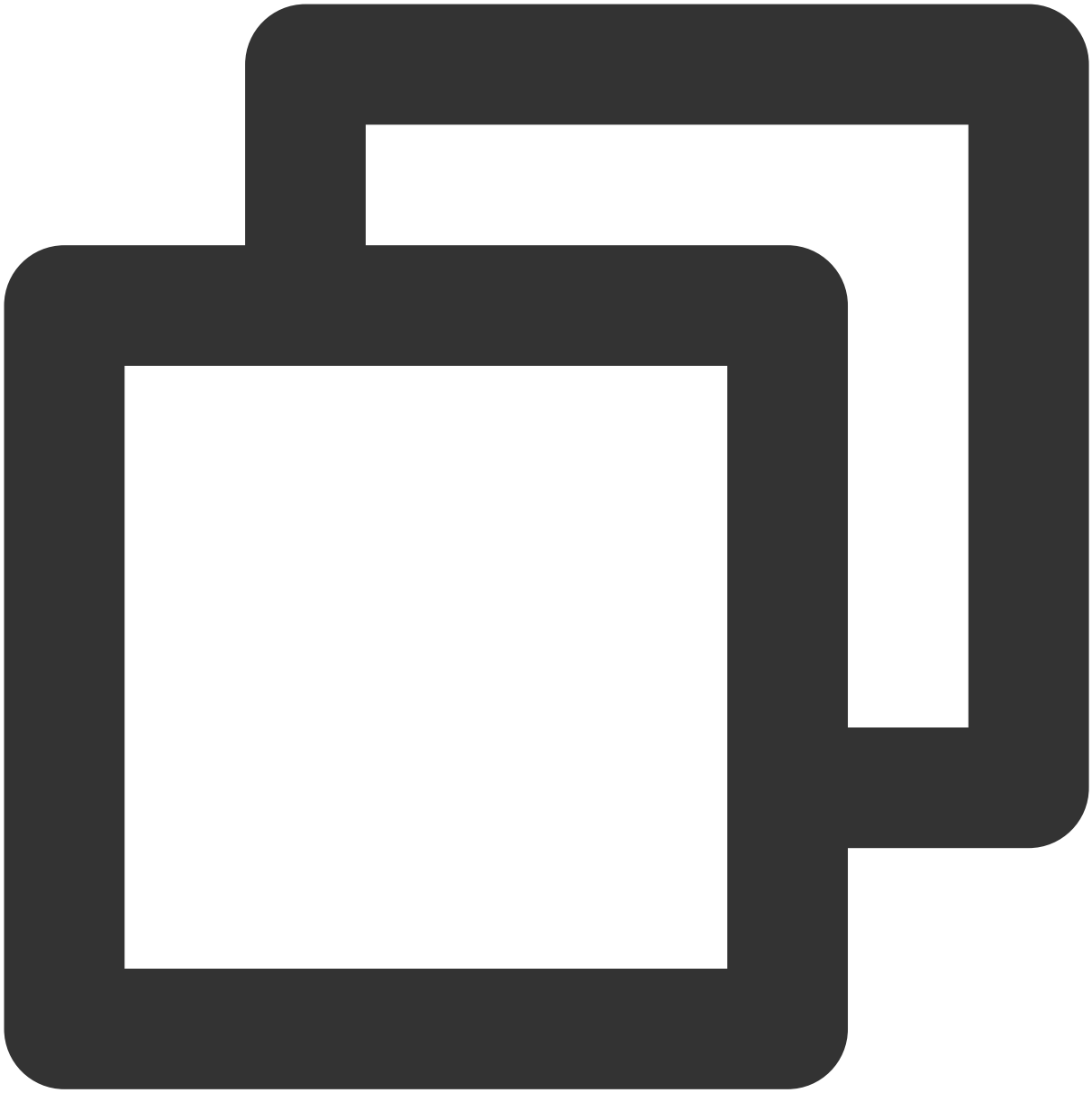
```
# .eslintignore
src/TUIKit
```

6. vue/cli 如何关闭 dev 模式下，webpack 全屏 overlay error 报错信息提示？

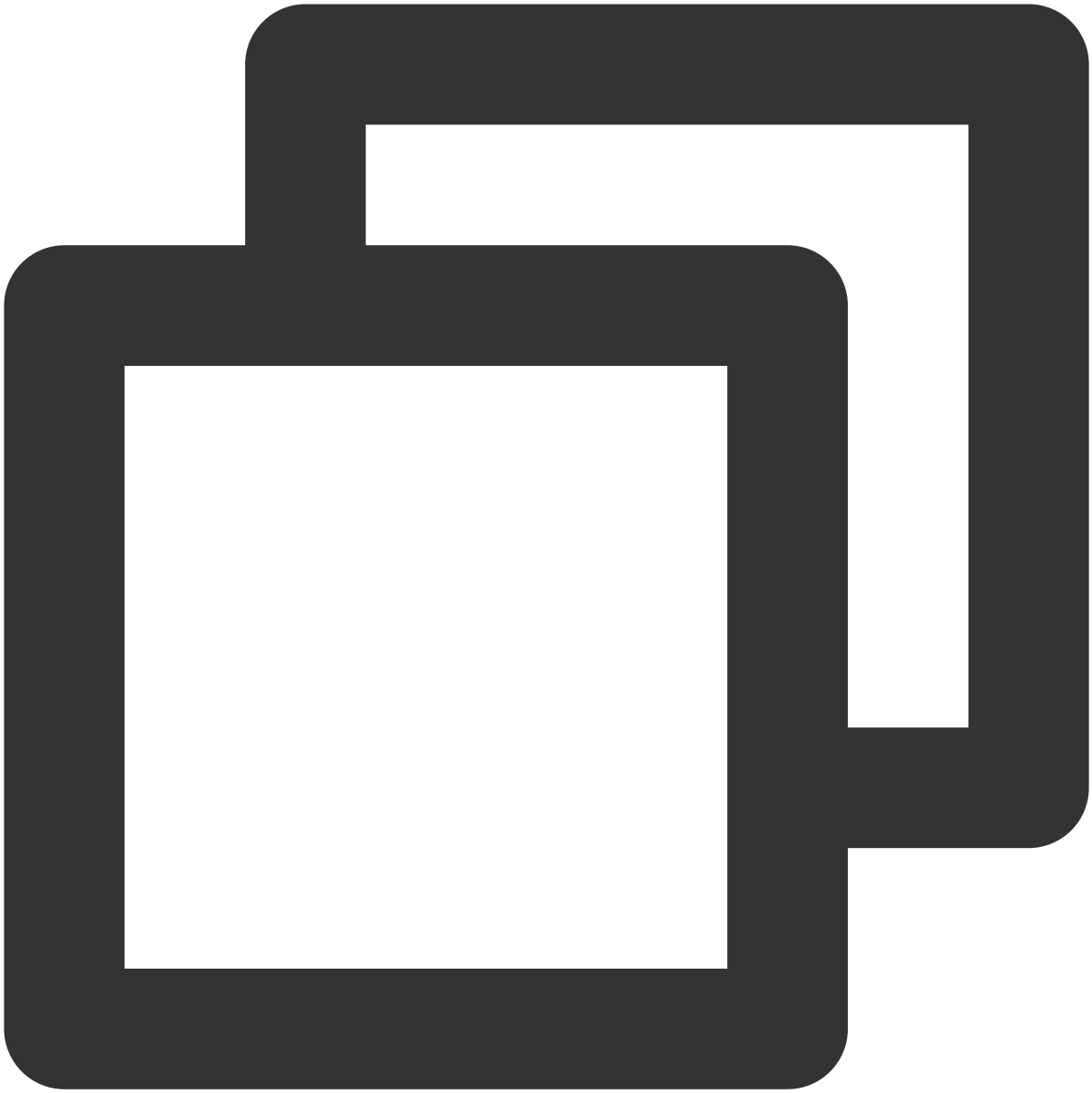
可以在您项目根目录的 vue.config.js 中进行关闭：

webpack4

webpack3



```
module.exports = defineConfig({
  ...
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```

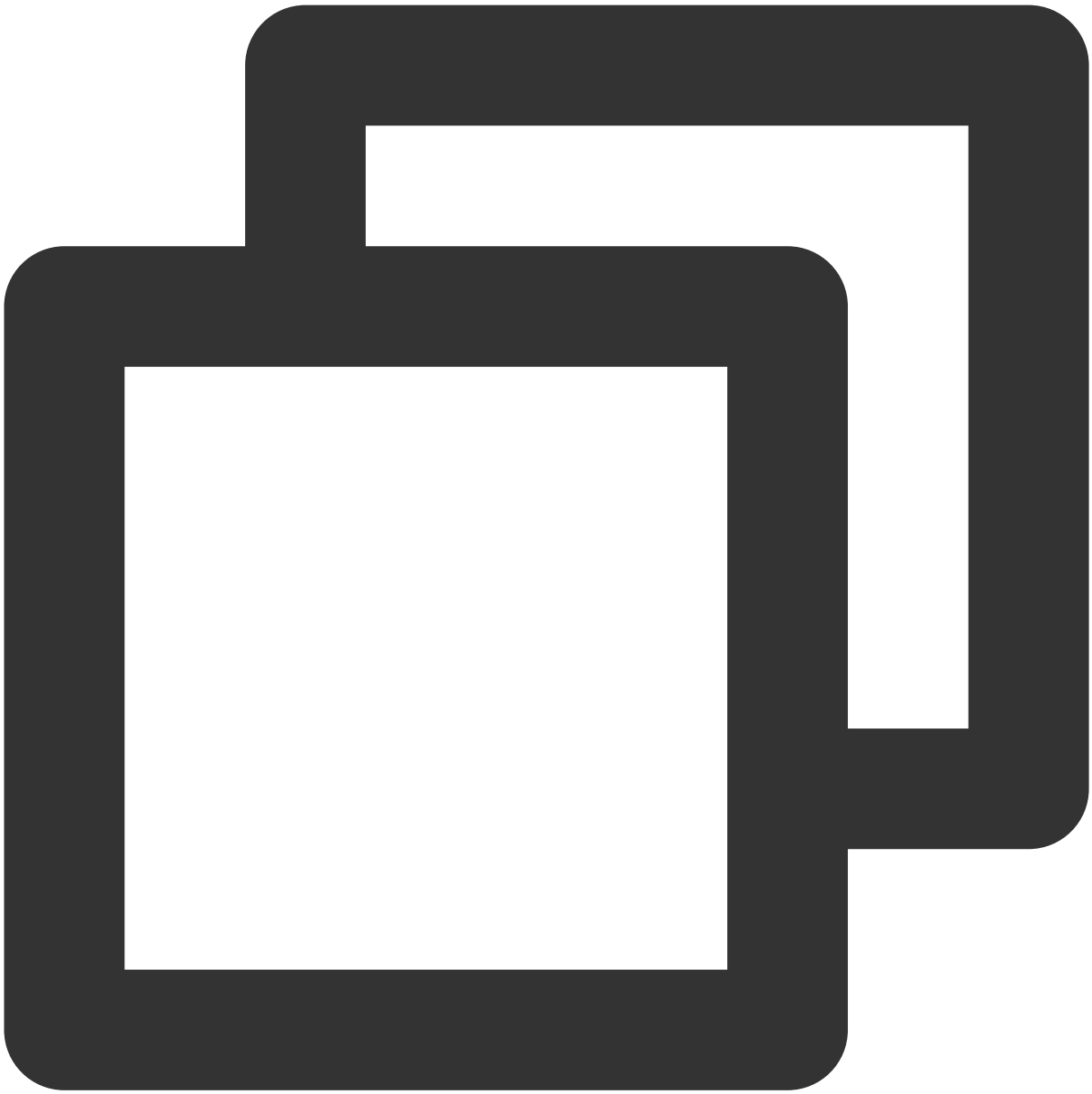


```
module.exports = {
  ...
  devServer: {
    overlay: false,
  },
};
```

7. 出现 Component name "XXXX" should always be multi-word 怎么办？

IM TUIKit web 所使用的 ESLint 版本为 v6.7.2 ，对于模块名的驼峰式格式并不进行严格校验。

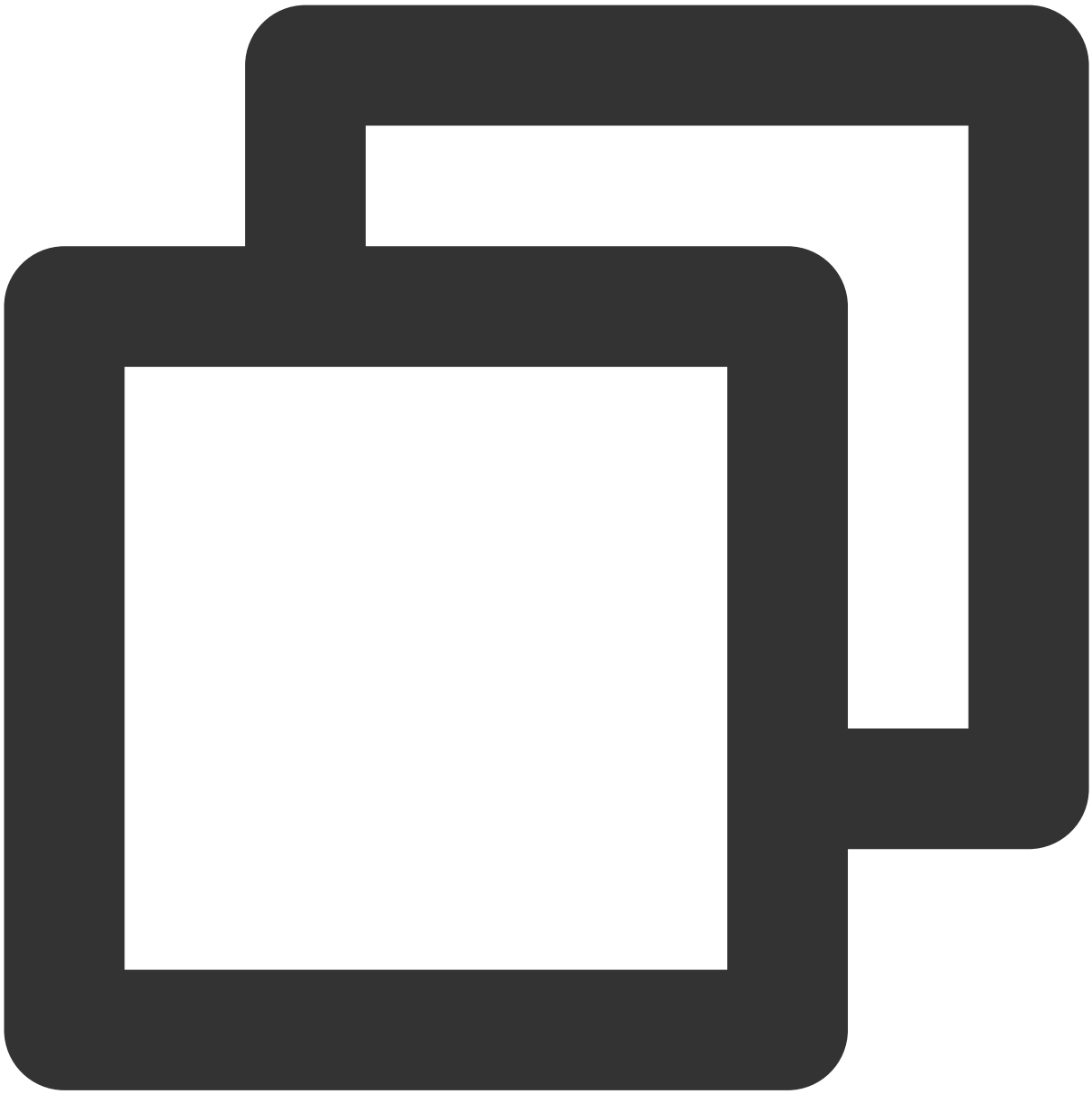
如果您出现此问题，您可以在 .eslintrc.js 文件中进行如下配置：



```
module.exports = {
  ...
  rules: {
      ...
      'vue/multi-word-component-names': 'warn',
  },
};
```

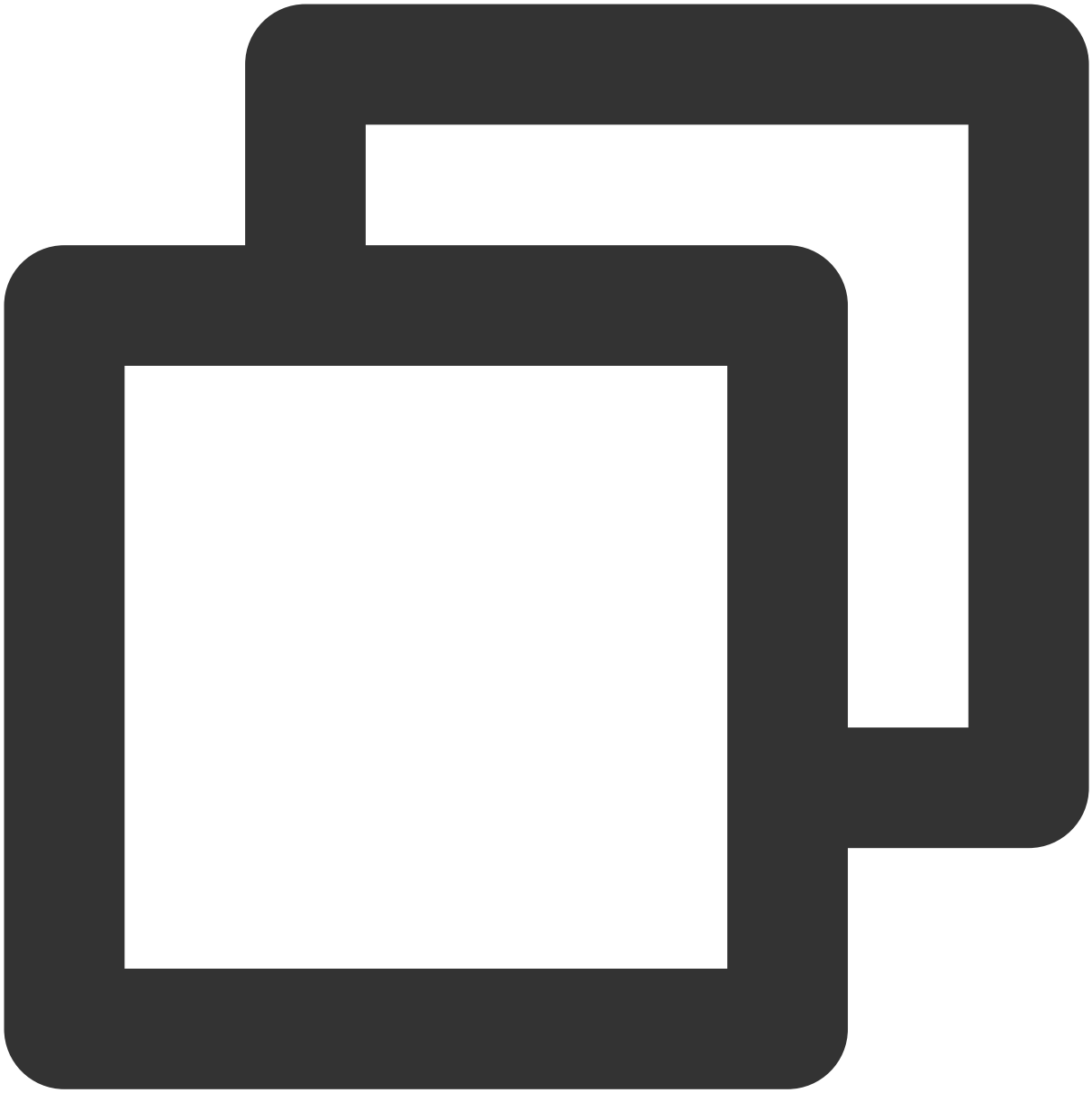
8. 出现 ERESOLVE unable to resolve dependency tree 怎么办？

npm install 的时候如果出现 ERESOLVE unable to resolve dependency tree， 表示依赖安装冲突，可采用以下方式进行安装：



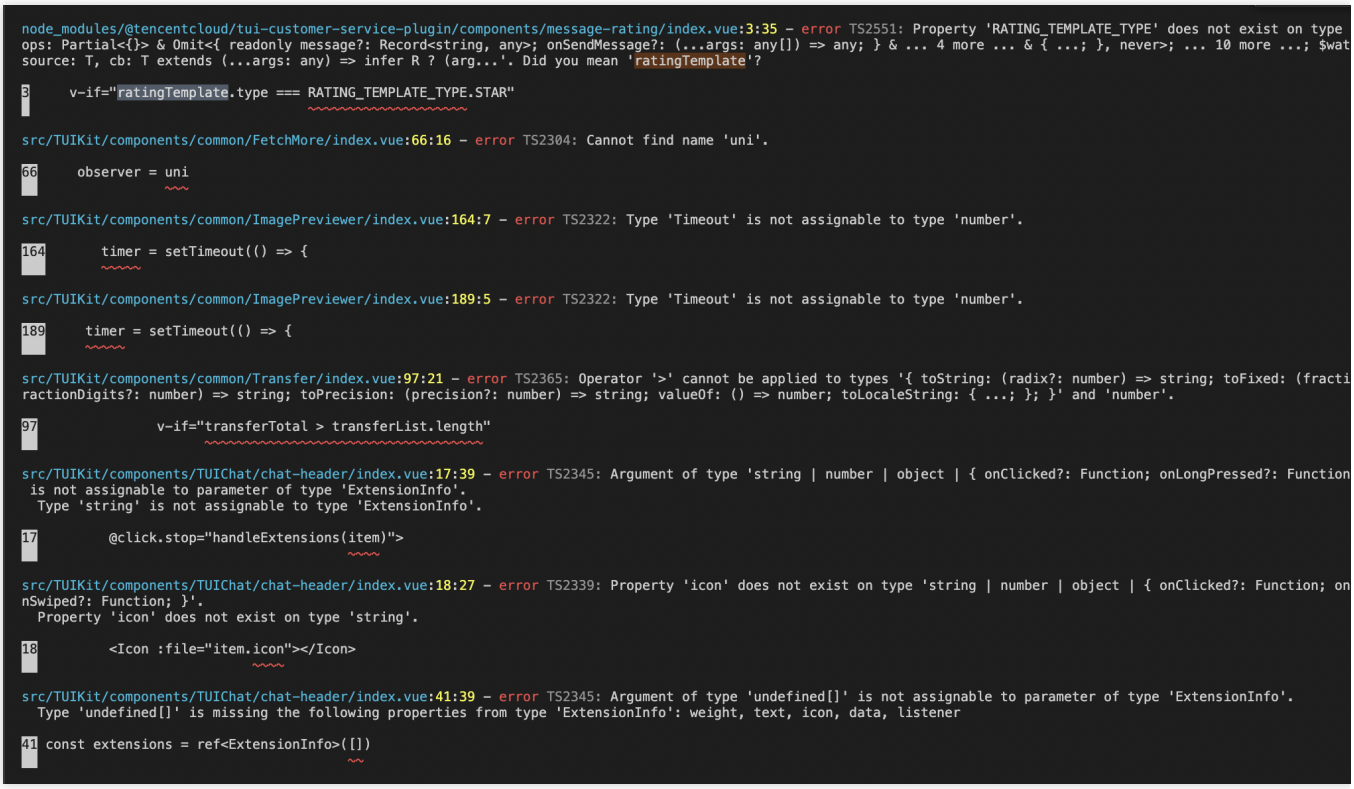
```
npm install --legacy-peer-deps
```

9. 运行报错如下'vue packages version mismatch',如何解决？



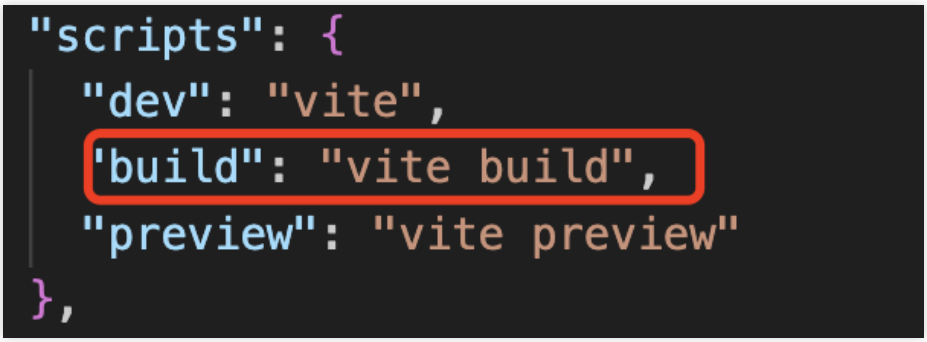
```
// 如果您是 vue2.7 项目，请在您项目根目录执行
npm i vue@2.7.9 vue-template-compiler@2.7.9
// 如果您是 vue2.6 项目，请在您项目根目录执行
npm i vue@2.6.14 vue-template-compiler@2.6.14
```

10. vite 项目 npm run build 之后 ts 报错?



原因：package.json script 下 "build": "vue-tsc && vite build" 中的 vue-tsc 命令导致。

解决方案: 删除 vue-tsc 即可。 "build": "vite build"



交流与反馈

加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

相关文档

Vue2 & Vue3 UIKit 相关：

chat-uikit-vue npm

Vue2 Demo源码及跑通示例

Vue3 Demo源码及跑通示例

Vue2 & Vue3 UIKit 逻辑层: engine 相关

chat-uikit-engine npm 仓库

chat-uikit-engine 接口文档

Unity

最近更新时间：2024-05-13 16:43:59

通过阅读本文，您可以了解集成 Unity SDK 的方法。

环境要求

环境	版本
Unity	2019.4.15f1 及以上版本。
Android	Android Studio 3.5及以上版本，App 要求 Android 4.1及以上版本设备。
iOS	Xcode 11.0及以上版本，请确保您的项目已设置有效的开发者签名。

支持平台

我们致力于打造一套支持 Unity 全平台的即时通信 IM SDK ，帮助您一套代码，全平台运行。

平台	IM SDK
iOS	支持
Android	支持
macOS	支持
Windows	支持
Web	支持，1.8.1+版本起

说明

Web 平台需要简单的几步额外引入，详情请查看本文 第五部分。

前提条件

1.1 您已 注册腾讯云 账号，并完成 实名认证。

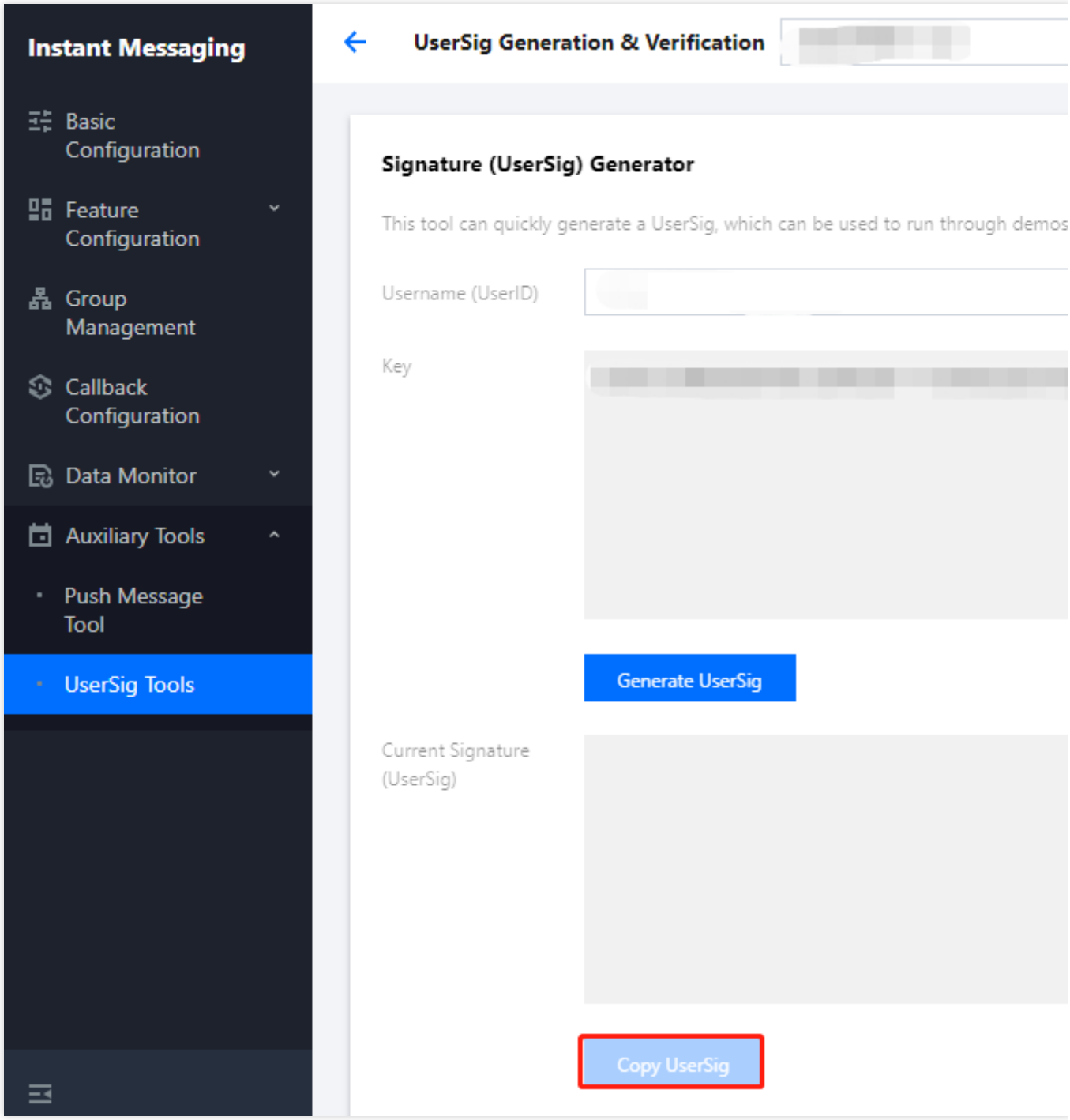
1.2 参照 创建并升级应用 创建应用，并记录好 `SDKAppID` 。

第一部分：创建测试用户

在 IM 控制台 选择您的应用，在左侧导航栏依次单击 **辅助工具 > UserSig 生成&校验**，创建两个 UserID 及其对应的 UserSig，复制 `UserID` 、 `签名（Key）` 、 `UserSig` 这三个，后续登录时会用到。

说明

该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是由服务器端生成，并提供面向 App 的接口，在需要 `UserSig` 时由 App 向业务服务器发起请求获取动态 `UserSig` 。更多详情请参见 服务端生成 UserSig。

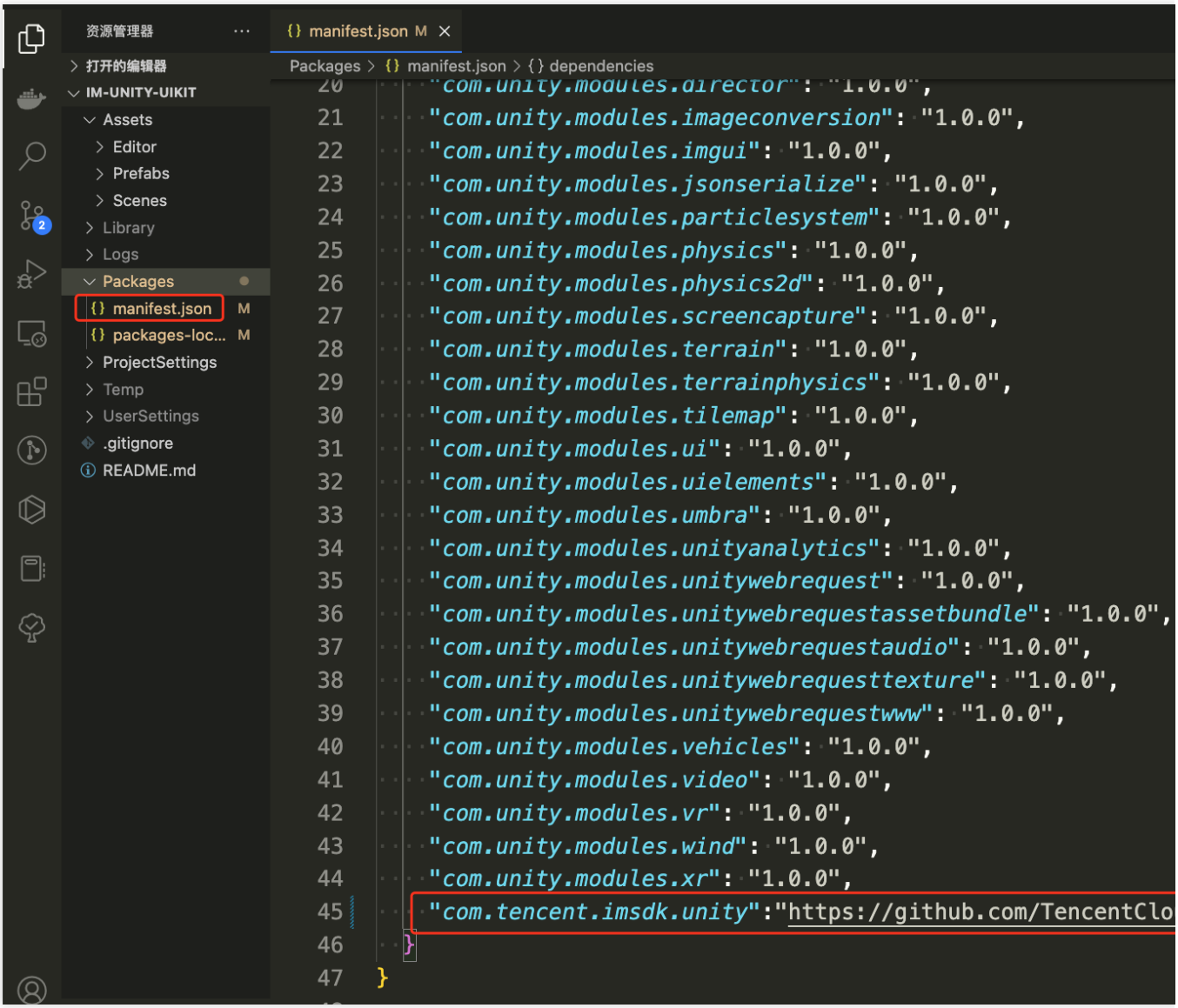


第二部分：集成 IM SDK 进您的 Unity 项目

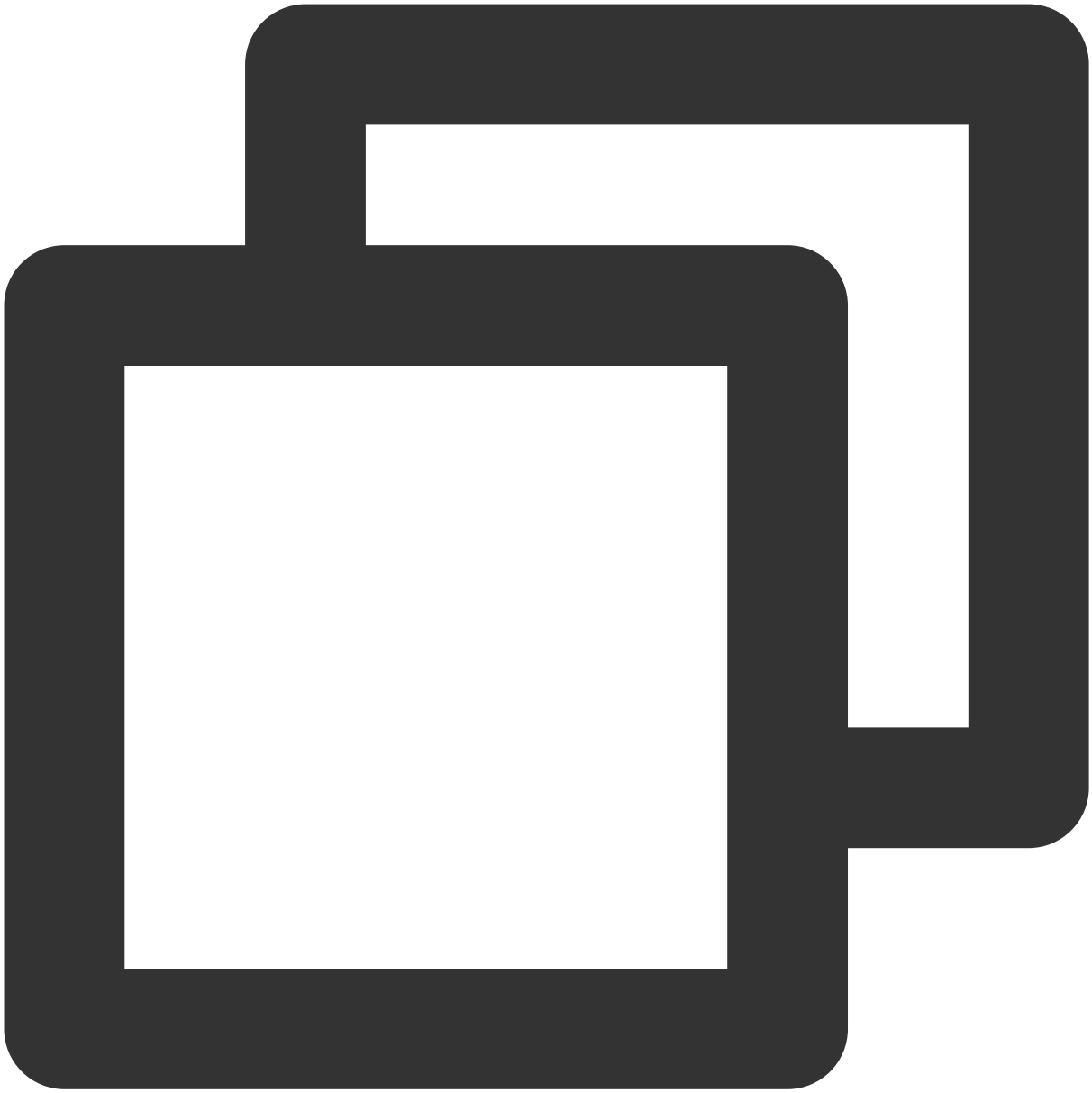
1.1 通过 Unity，创建一个 Unity 项目，并记住项目所在的位置。

或打开一个已有 Unity 项目。

1.2 通过 IDE（如：Visual Studio Code）打开项目：



1.3 根据目录，找到 Packages/manifest.json，并修改依赖如下：

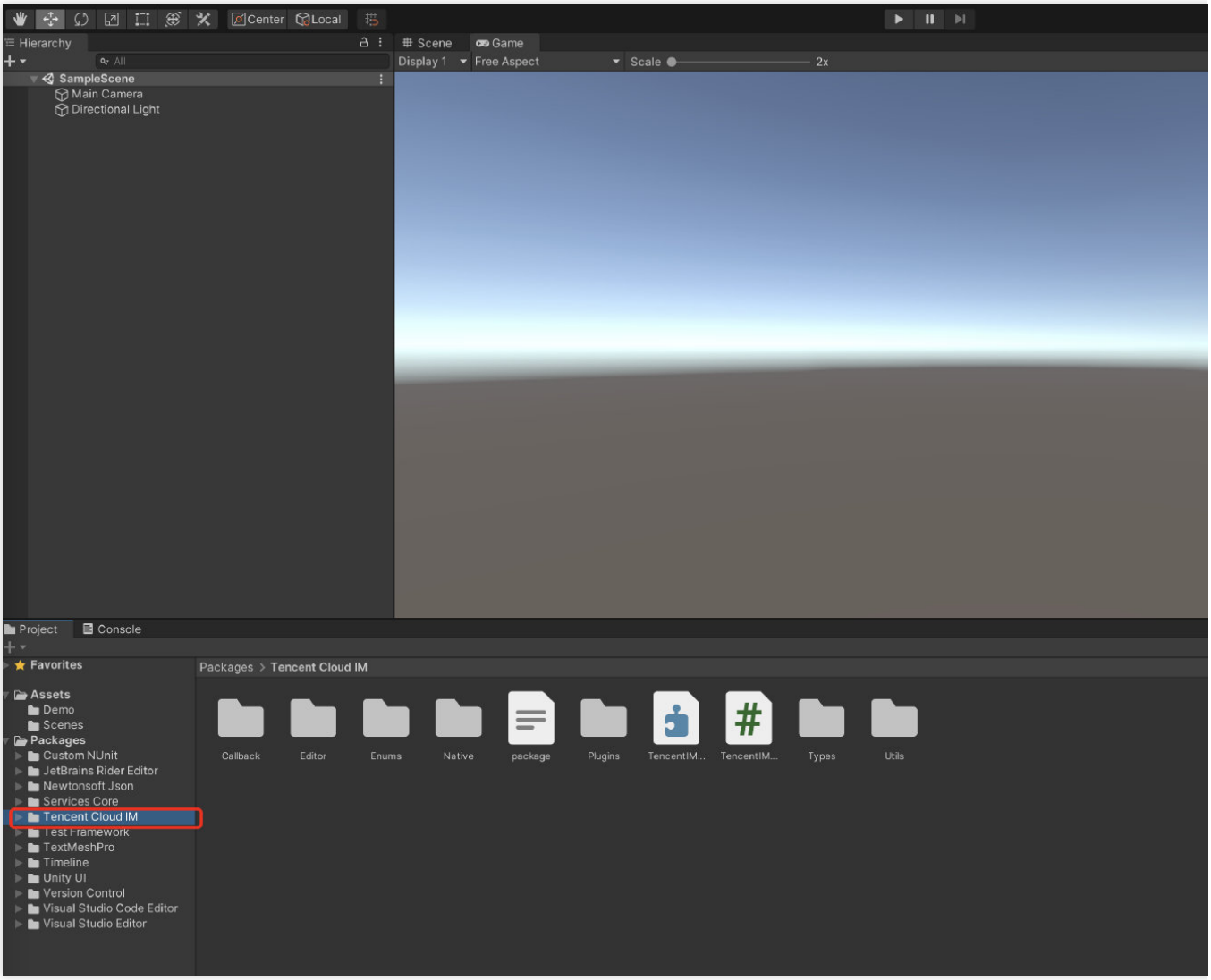


```
{
  "dependencies":{
    "com.tencent.imsdk.unity":"https://github.com/TencentCloud/chat-sdk-unity.git#u
  }
}
```

为帮助您更好的理解 IM SDK 的各 API，我们还提供了 API Example，演示各 API 的调用及监听的触发。

第三部分：加载依赖

在 Unity Editor 中打开项目，等候依赖加载完毕，确认Tencent Cloud IM 已经加载完成。



第四部分：自实现 UI 集成

前提条件

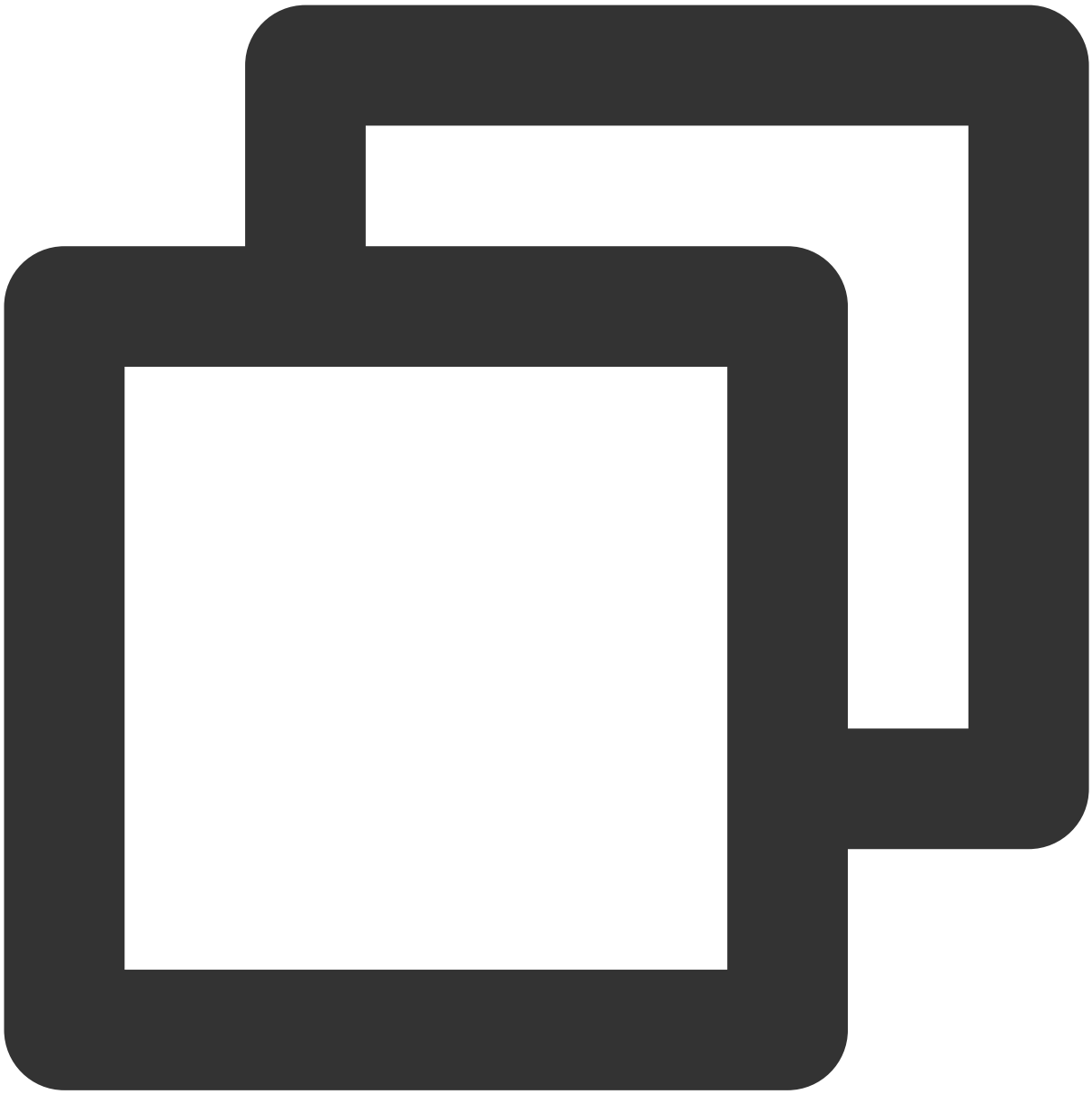
您已经完成创建 Unity 项目，或有可以基于的 Unity 项目，并加载了 Tencent Cloud IM SDK。

完成 SDK 初始化

本节详细文档

调用 `TencentIMSDK.Init` ，完成 SDK 初始化。

将您的 `SDKAppID` 传入。



```
using Com.Tencent.IM.Unity.UIKit;
using com.tencent.imsdk.unity;
using com.tencent.imsdk.unity.types;
using com.tencent.imsdk.unity.enums;
namespace Com.Tencent.IM.Unity.UIKit{
    public static void Init() {
      string SDKAppID = 0; // 从即时通信 IM 控制台获取应用 SDKAppID。
      SdkConfig sdkConfig = new SdkConfig();
```

```
        sdkConfig.sdk_config_config_file_path = Application.persistentDataPath + "/TI

        sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM-L

        TIMResult res = TencentIMSDK.Init(long.Parse(SDKAppID), sdkConfig);
    }
}
```

在 `Init` 后，您可以针对 IM SDK 挂载一些监听，主要包括网络状态及用户信息变更等，详情可参见 该文档。

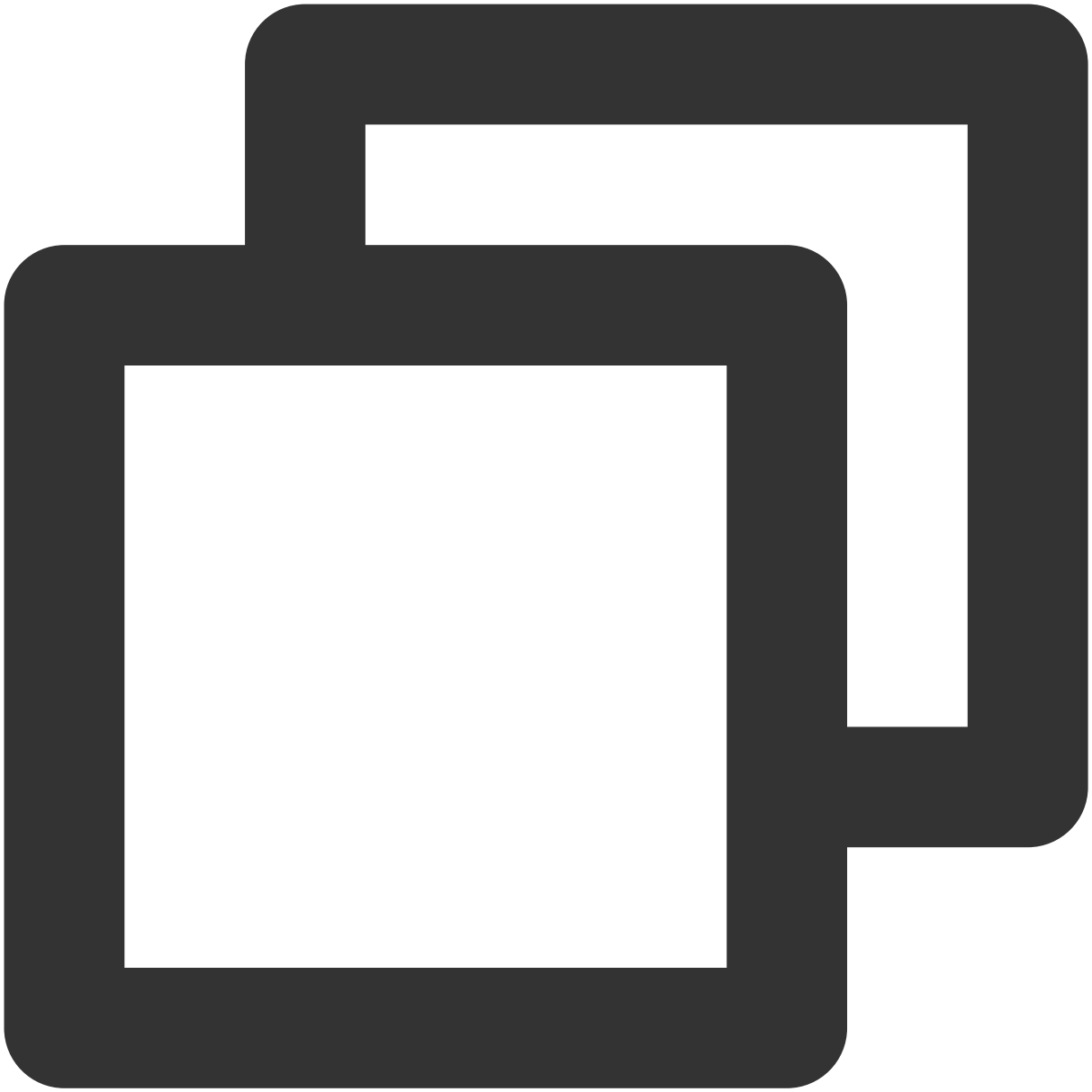
登录测试账户

本节详细文档

此时，您可以使用最开始的时候，在控制台生成的测试账户，完成登录验证。

调用 `TencentIMSDK.Login` 方法，登录一个测试账户。

当返回值 `res.code` 为0时，登录成功。



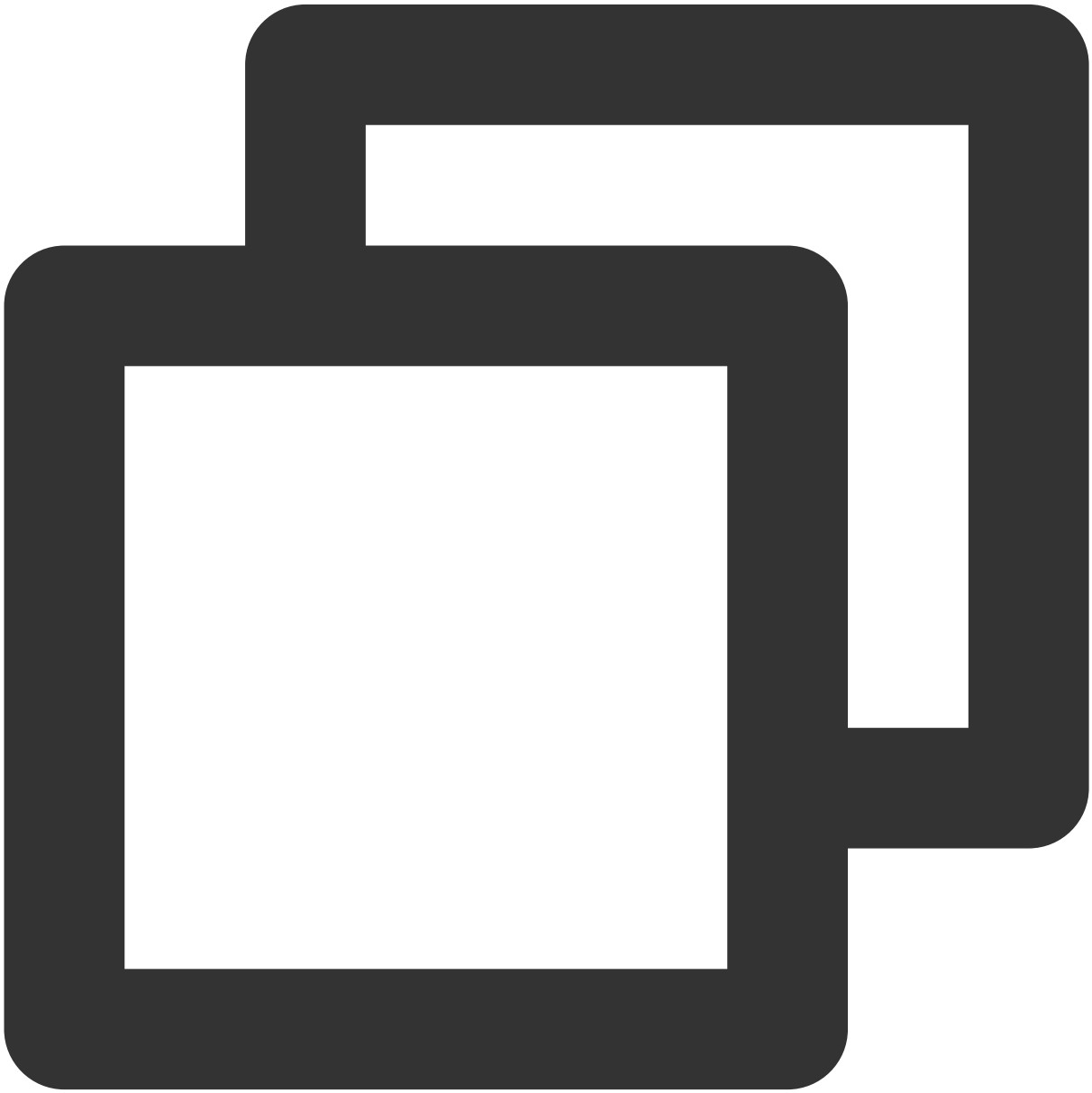
```
public static void Login() {
  if (userid == "" || user_sig == "")
  {
      return;
  }
  TIMResult res = TencentIMSDK.Login(userid, user_sig, (int code, string desc, stri
    // 处理登陆回调逻辑
  });
}
```

说明

该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是将 `UserSig` 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 `UserSig` 时由您的 App 向业务服务器发起请求获取动态 `UserSig` 。更多详情请参见 服务端生成 UserSig。

发送消息

本节详细文档
此处以发送文本消息举例
代码示例：



```
public static void MsgSendMessage() {
```

```
        string conv_id = ""; // c2c 消息会话 ID 为 userID，群消息会话 ID 为 groupID
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // 群消息为TIMConvType.kTIMC
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Text,
              text_elem_content =  "这是一个普通文本消息"
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128); // 承接消息ID

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // 消息发送异步结果
        });
              // 消息发送同步返回的消息ID messageId
}
```

说明

如果发送失败，可能是由于您的 SDKAppID 不支持陌生人发送消息，您可至控制台开启，用于测试。

请单击此链接，关闭好友关系链检查。

获取会话列表

本节详细文档

在上一个步骤中，完成发送测试消息，现在可登录另一个测试账户，拉取会话列表。

获取会话列表的方式有两种：

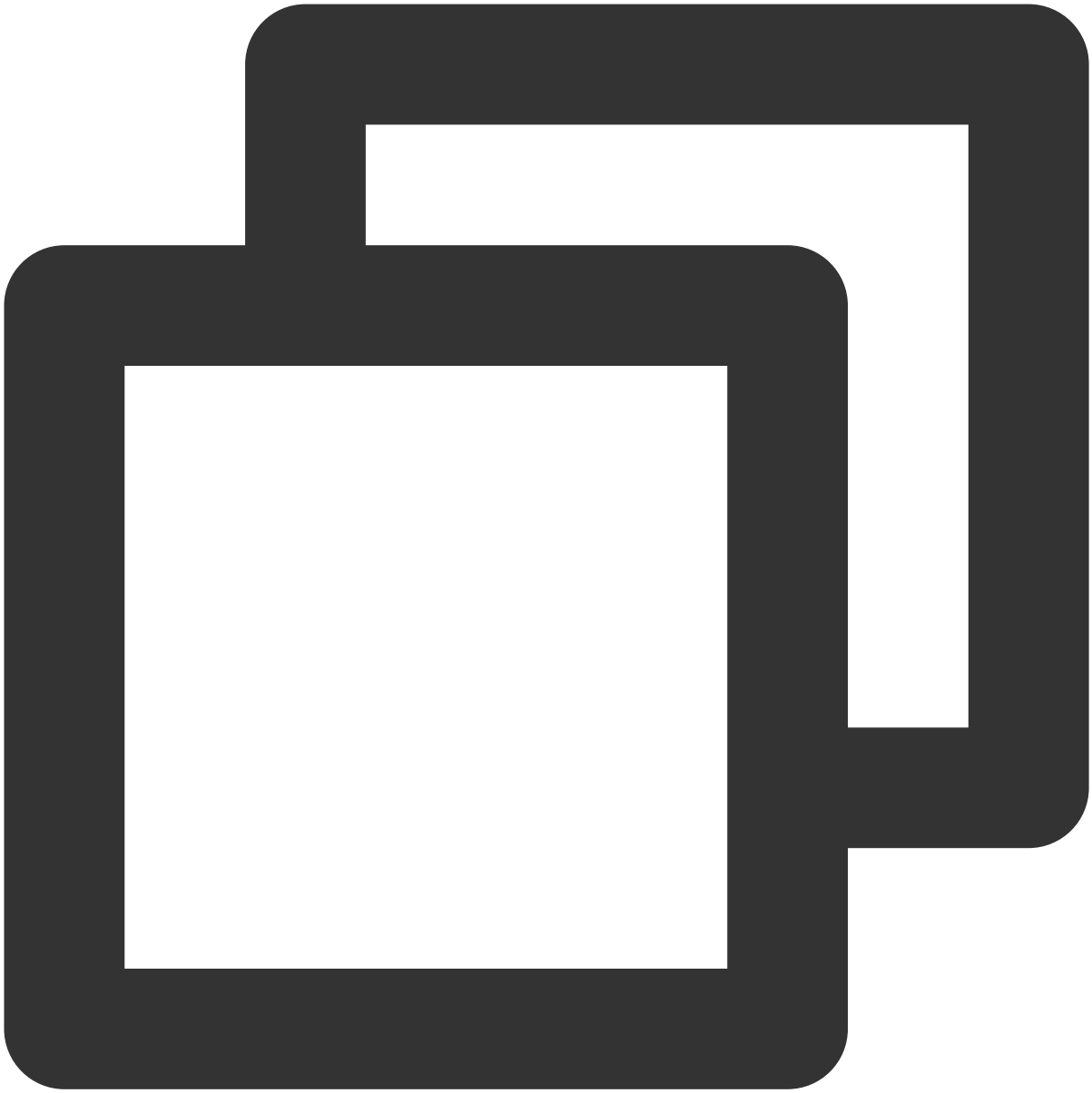
1.1 监听长连接回调，实时更新会话列表。

1.2 请求 API，根据分页一次性获取会话列表。

常见应用场景为：

在启动应用程序后立即获取会话列表，然后监听长连接以实时更新会话列表的变化。

一次性请求会话列表



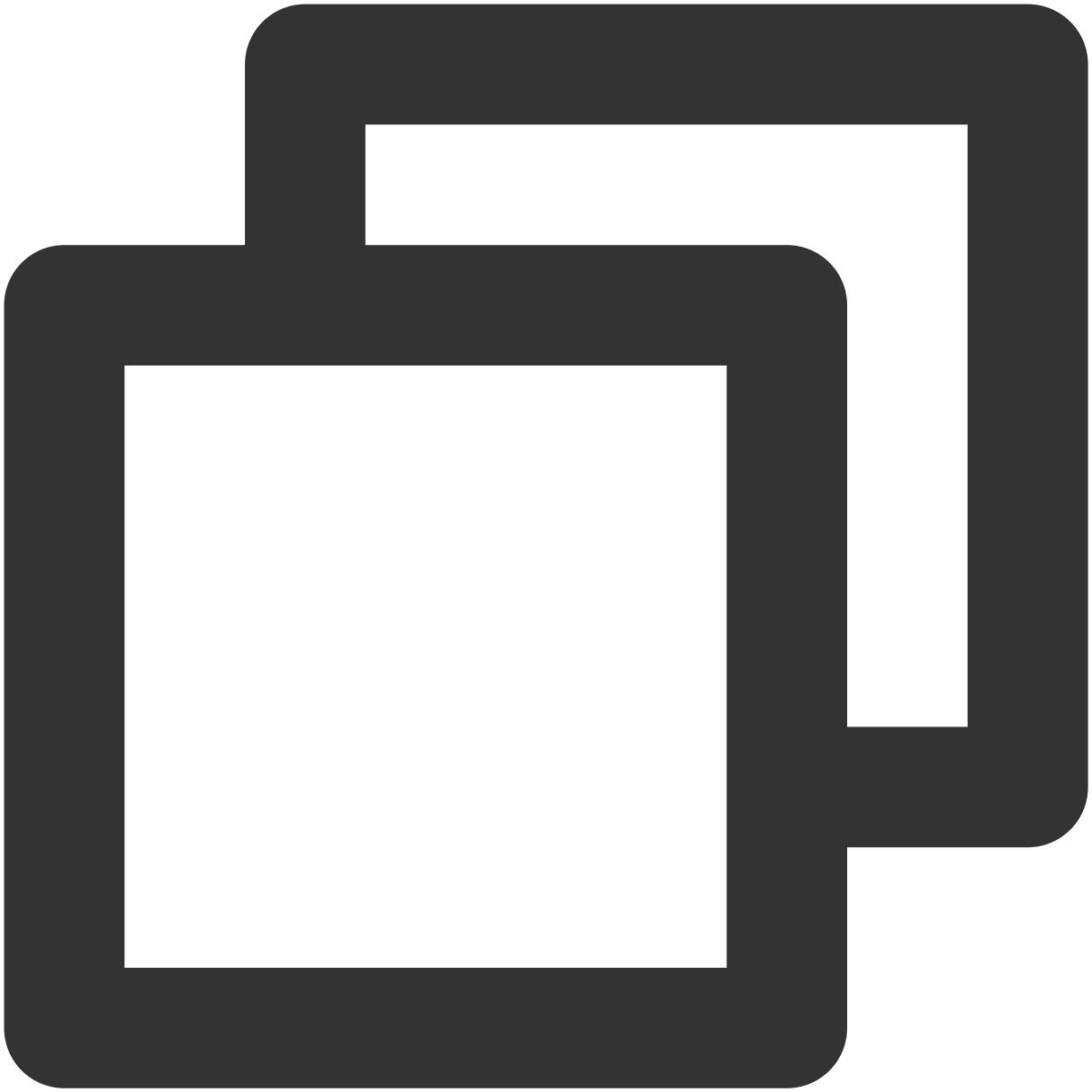
```
TIMResult res = TencentIMSDK.ConvGetConvList((int code, string desc, List<ConvInfo>
 // 处理异步逻辑
});
```

此时，您可以看到您在上一步中，使用另一个测试账号，发来消息的会话。

监听长链接实时获取会话列表

您在此步骤中，需要先在 SDK 上挂载监听，然后处理回调事件，更新 UI。

1.1 挂载监听。



```
TencentIMSDK.SetConvEventCallback((TIMConvEvent conv_event, List<ConvInfo> conv_lis
 // 处理回调逻辑
});
```

1.2 处理回调事件，将最新的会话列表展示在界面上。

接收消息

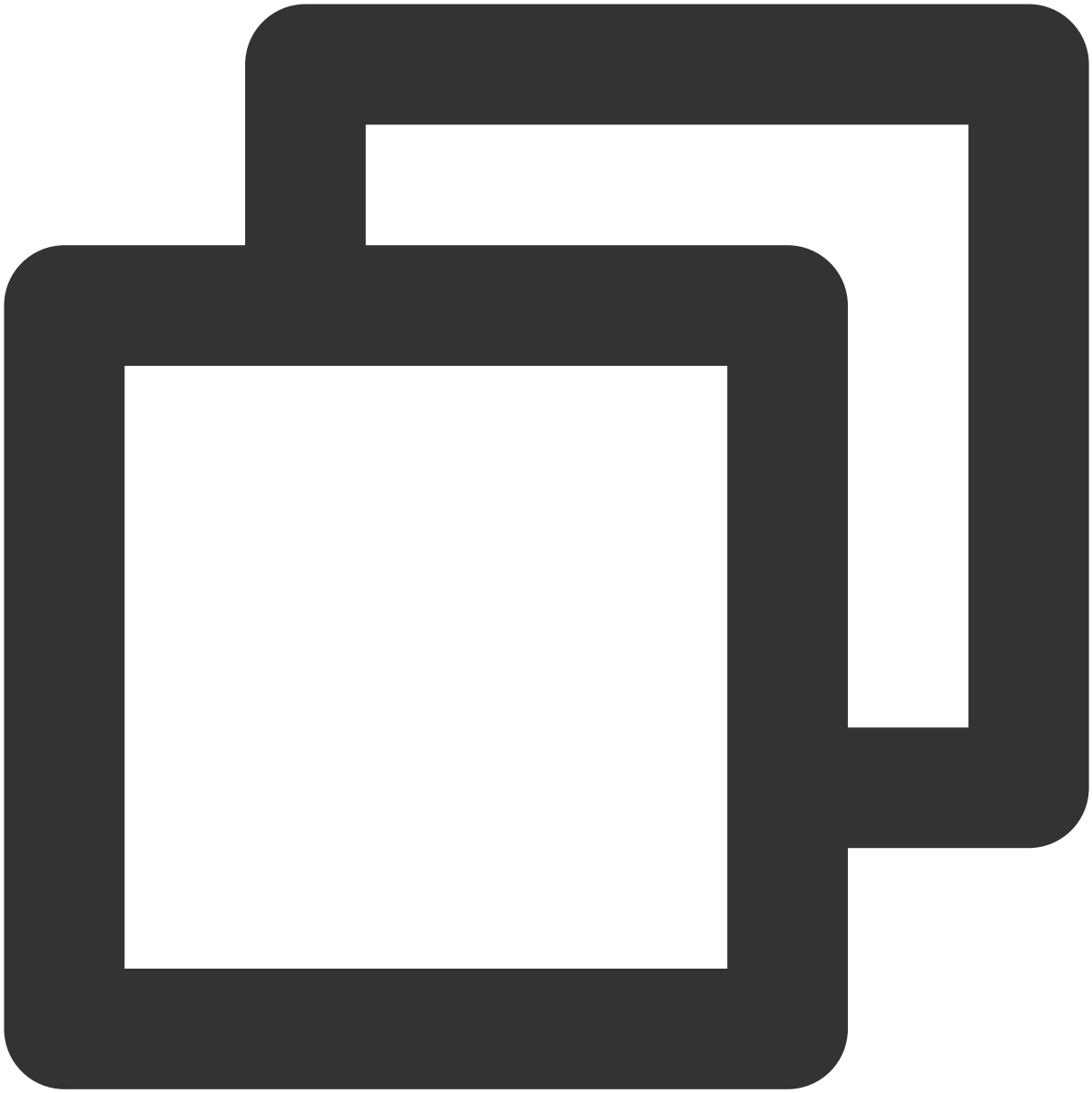
本节详细文档

通过腾讯云 IM SDK 接收消息有两种方式：

1.1 监听长连接回调，实时获取消息变化，更新渲染历史消息列表。
1.2 请求 API，根据分页一次性获取历史消息。
常见应用场景为：
1.1 界面进入新的会话后，首先一次性请求一定数量的历史消息，用于展示历史消息列表。
1.2 监听长链接，实时接收新的消息，将其添加进历史消息列表中。

一次性请求历史消息列表

每页拉取的消息数量不能太大，否则会影响拉取速度。建议此处设置为20左右。
您应该动态记录当前页数，用于下一轮请求。
示例代码如下：



```
// 拉取单聊历史消息
// 首次拉取，不需要填msg_getmsglist_param_last_msg，不填则默认拉取最新消息
Message LastMessage = null;
string LastMessageID = "";
var get_message_list_param = new MsgGetMsgListParam();
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
  // 处理回调逻辑
  List<Message> messages = Utils.FromJson<List<Message>>((string)parameters[1]);
   if (messages.Count > 0){
    LastMessage = messages[messages.Count - 1];
    LastMessageID.text = messages[messages.Count - 1].message_msg_id;
```

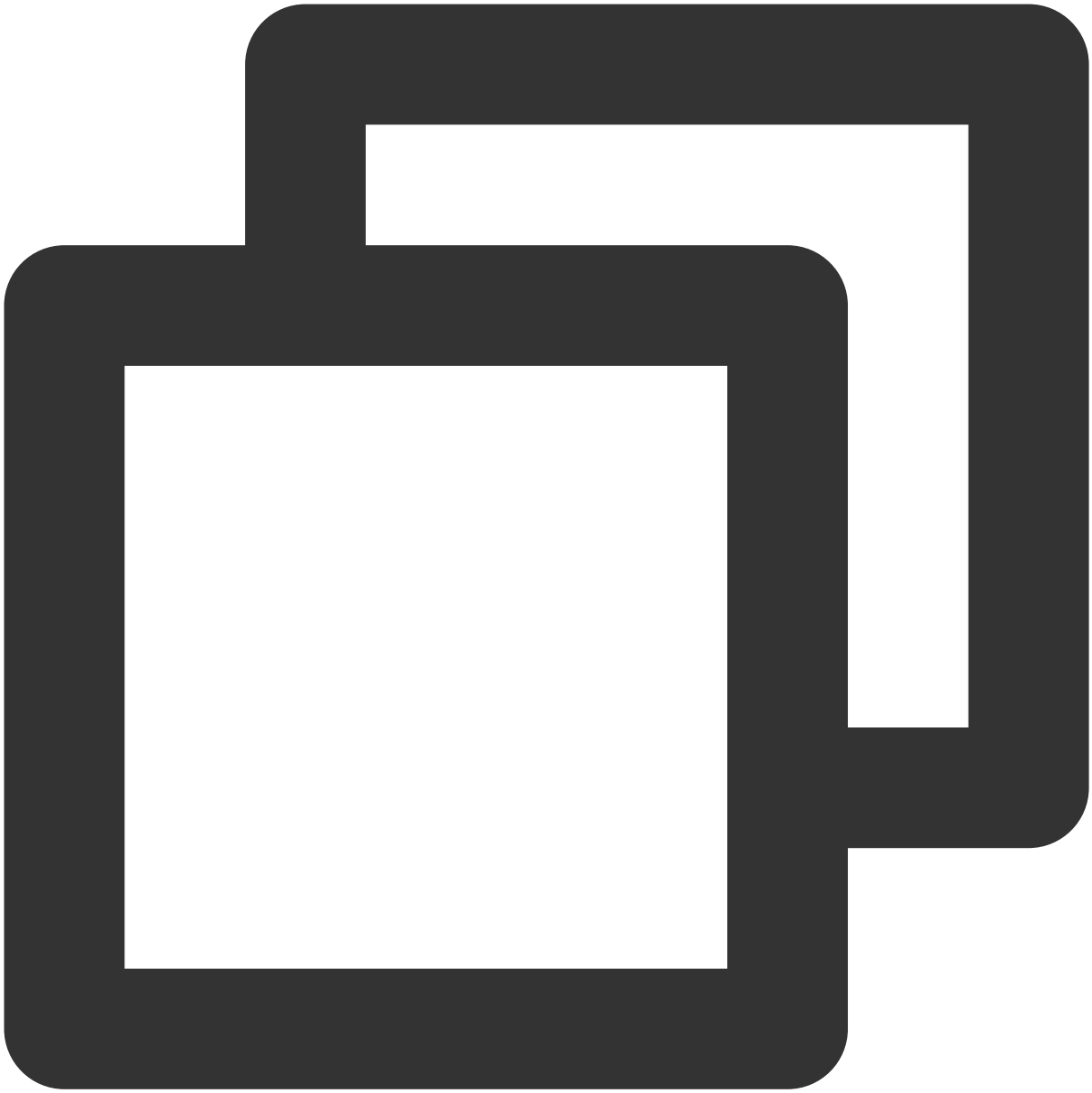
```
  }else {
    LastMessage = null;
    LastMessageID.text = "";
   }

});
// 再次拉取时，msg_getmsglist_param_last_msg 可以使用返回的消息列表中的最后一条消息
var get_message_list_param = new MsgGetMsgListParam
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_Group, get
  // 处理回调逻辑
});
```

监听长链接实时获取新消息

历史消息列表初始化后，新消息来自长链接 `TencentIMSDK.AddRecvNewMsgCallback` 。

`AddRecvNewMsgCallback` 回调被触发后，您可以按需将新消息添加进历史消息列表中。

绑定监听器示例代码如下：



```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> message, string user_data) => {
  // 处理新消息
});
```

此时，您已基本完成 IM 模块开发，可以发送接收消息，也可以进入不同的会话。

您可以继续完成 群组，用户资料，关系链，本地搜索 等相关功能开发。

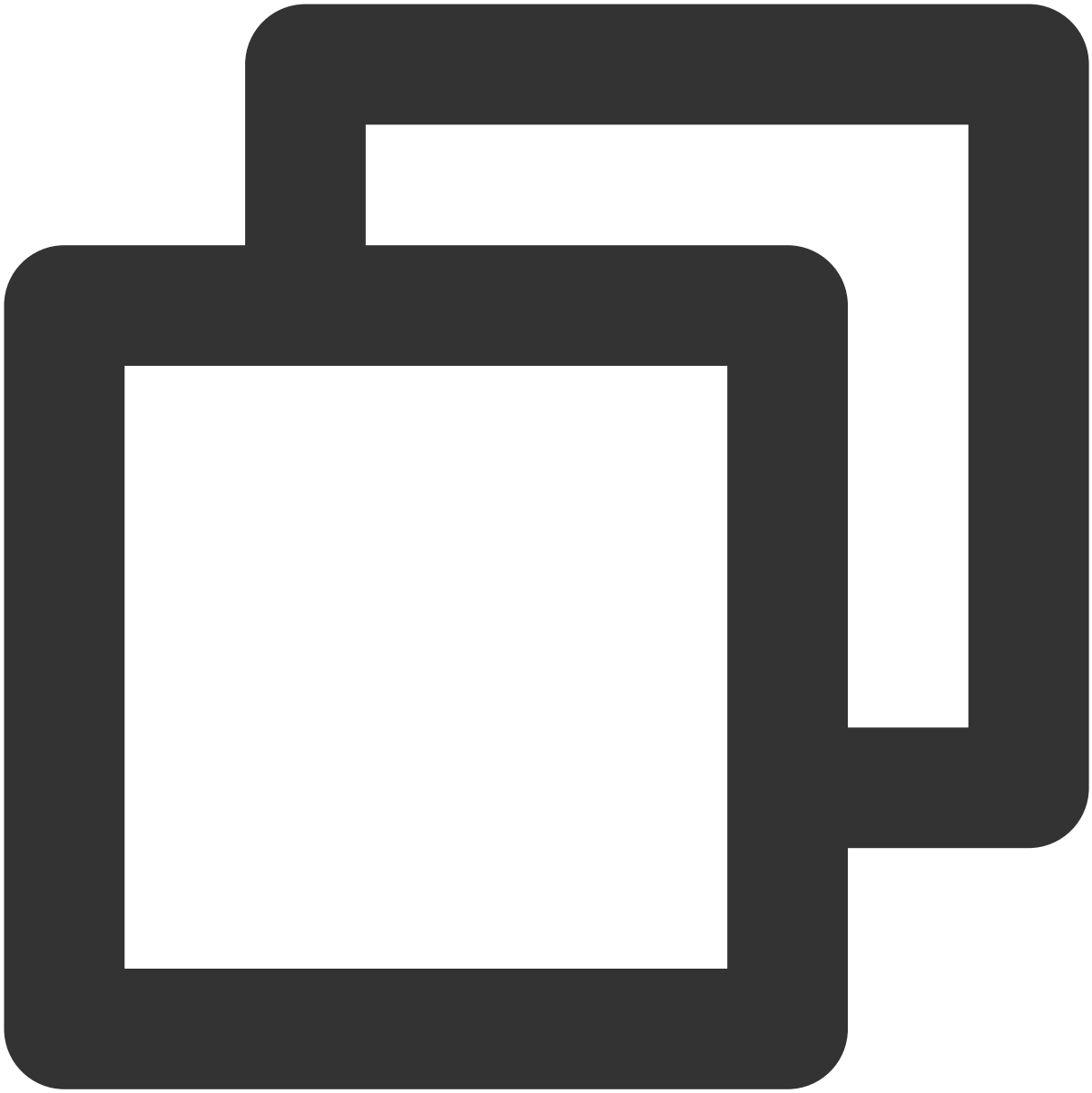
详情可查看 自实现 UI 集成 SDK 文档。

第五部分：#Unity for WebGL 支持

Tencent Cloud IM SDK (Unity 版本) 自 `1.8.1` 版本起支持构建 WebGL。
相比 Android 和 iOS 端，需要一些额外步骤。如下：

引入 JS

从 Npm 下载下方三个JS文件，放置于项目构建 WebGL 产物的文件夹内。
tim-js
tim-js-friendship.js
将此文件重命名成 tim-upload-plugin.js
打开 `index.html` ，并引入这三个JS文件。如下：



```
<script src="./tim-js.js"></script>
<script src="./tim-js-friendship.js"></script>
<script src="./tim-upload-plugin.js"></script>
```

常见问题

支持哪些平台？

目前支持 iOS、Android、Windows Mac 和 WebGL。

Android 单击 Build And Run 报错找不到可用设备？

确保设备没被其他资源占用，或单击 Build 生成 apk 包，再拖动进模拟器里运行。

iOS 第一次运行报错？

按照上面的 Demo 运行配置后，如果报错，可以单击**Product**>**Clean**，清除产物后重新 Build，或者关闭 Xcode 重新打开再次 Build。

2019.04版 Unity，iOS 平台报错？

Library/PackageCache/com.unity.collab-proxy@1.3.9/Editor/UserInterface/Bootstrap.cs(23,20): error CS0117: 'Collab' does not contain a definition for 'ShowChangesWindow'
在 Editor 工具栏单击**Window**>**Package Manager**，将 Unity Collaborate 降级到1.2.16。

2019.04版 Unity，iOS 平台报错？

Library/PackageCache/com.unity.textmeshpro@3.0.1/Scripts/Editor/TMP_PackageUtilities.cs(453,84): error CS0103: The name 'VersionControlSettings' does not exist in the current context
打开源码，把 `|| VersionControlSettings.mode != "Visible Meta Files"` 这部分代码删除即可。

这是 C# 接口吗？如何脱离 Unity 使用？

Unity SDK 是使用 C# 的 SDK，但由于 Unity SDK 包含 Unity 特性，不能直接在纯 C# 的环境下使用。
若需要能在 C# 环境下使用，我们提供单独的 C# SDK nuget 包。使用方法与 Unity SDK 相同，可直接参见 Unity SDK 文档使用。
其中，纯 C# SDK 只支持 PC 端，Unity SDK 支持移动端。

有可以直接使用的 UI 吗？

现在暂不提供 Untiy SDK 和 C# SDK 相应的 UIKit。

错误码如何查询？

IM SDK 的 API 层面错误码，请查看 该文档。

UE

最近更新时间：2024-05-13 16:44:11

本文主要介绍如何快速运行腾讯云即时通信 IM Demo（Unreal Engine）。

说明：

目前支持 Windows、macOS、iOS、Android。

环境要求

建议 Unreal Engine 4.27.1 及以上版本。

开发端	环境
Android	Android Studio 4.0 及以上版本。 Visual Studio 2017 15.6 及以上版本。 只支持真机调试。
iOS & macOS	Xcode 11.0 及以上版本。 OSX 系统版本要求 10.11 及以上版本 。 请确保您的项目已设置有效的开发者签名。
Windows	操作系统：Windows 7 SP1 及以上版本（基于 x86-64 的 64 位操作系统）。 磁盘空间：除安装 IDE 和一些工具之外还应有至少 1.64 GB 的空间。 安装 Visual Studio 2019 。

前提条件

您已 注册腾讯云 账号，并完成 实名认证。

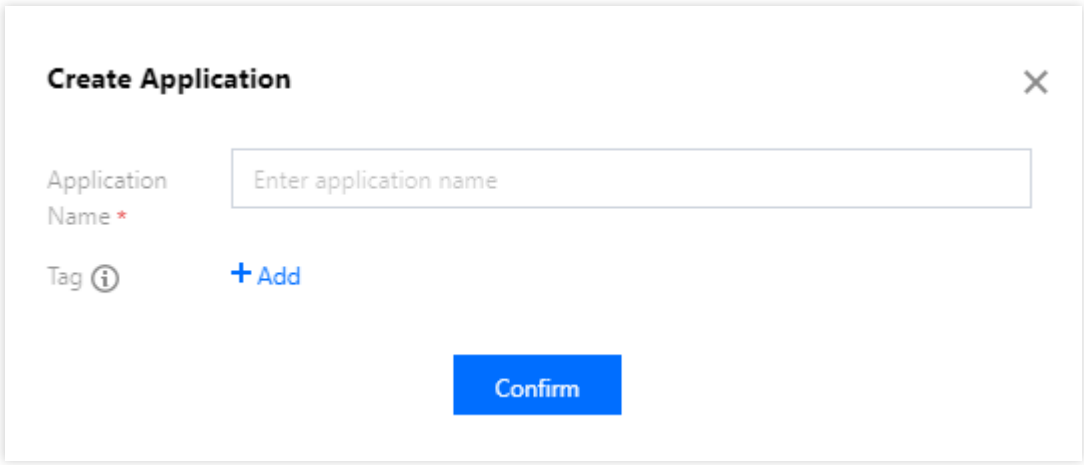
操作步骤

步骤1：创建新的应用

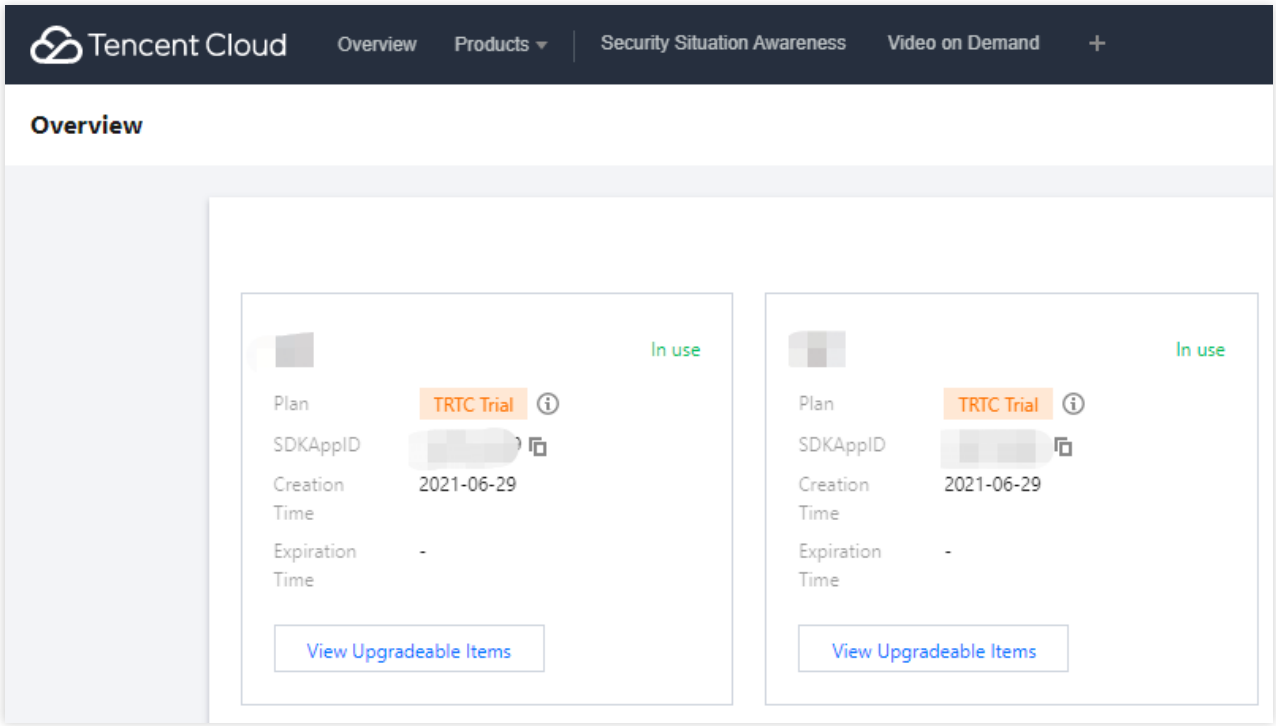
1. 登录 即时通信 IM 控制台。

说明：

如果您已有应用，请记录其 SDKAppID 并 获取密钥信息。
同一个腾讯云帐号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**
2. 单击**创建新应用**，在**创建应用**对话框中输入您的应用名称，单击**确定**。

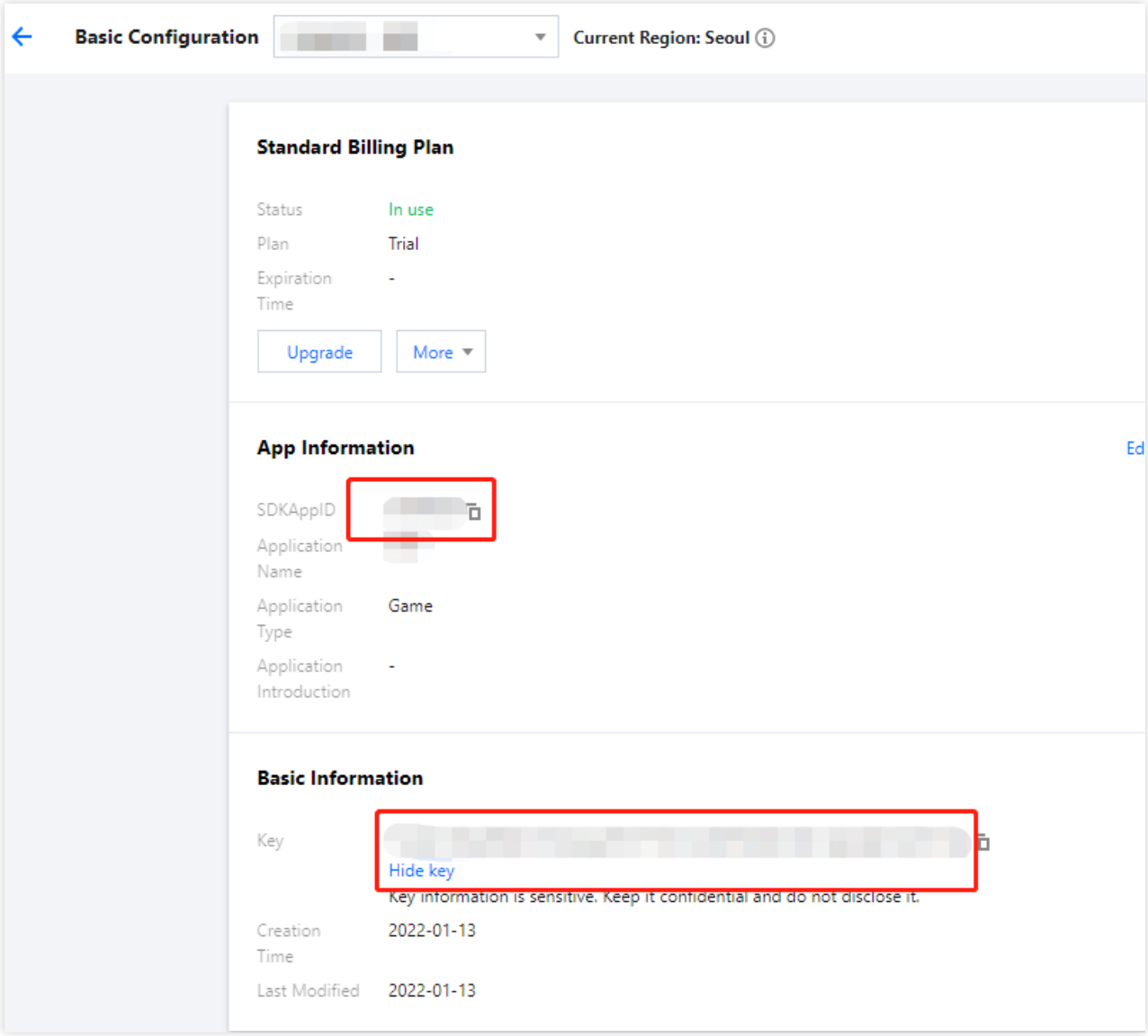


3. 创建完成后，可在控制台总览页查看新建应用的状态、业务版本、SDKAppID、创建时间、标签以及到期时间。请记录 SDKAppID 信息。



步骤2：获取密钥信息

1. 单击目标应用卡片，进入应用的基础配置页面。



2. 在**基本信息**区域，单击**显示密钥**，复制并保存密钥信息。

注意：

请妥善保管密钥信息，谨防泄露。

步骤3：配置 Demo 工程文件

1. 下载即时通信 IM Demo 工程，具体下载地址请参见 Demo 下载。
2. 找到并打开 `/IM_Demo/Source/debug/include/DebugDefs.h` 文件。
3. 设置 `DebugDefs.h` 文件中的相关参数：

SDKAPPID：默认为 0，请设置为实际的 SDKAppID。
SECRETKEY：默认为 ""，请设置为实际的密钥信息。

说明：

本文提到的生成 UserSig 的方案是在客户端代码中配置 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 Demo 和功能调试**。

正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

步骤4：编译打包运行

1. 双击打开 `/IM_Demo/IM_Demo.uproject` 。
2. 编译运行调试：

macOS 端

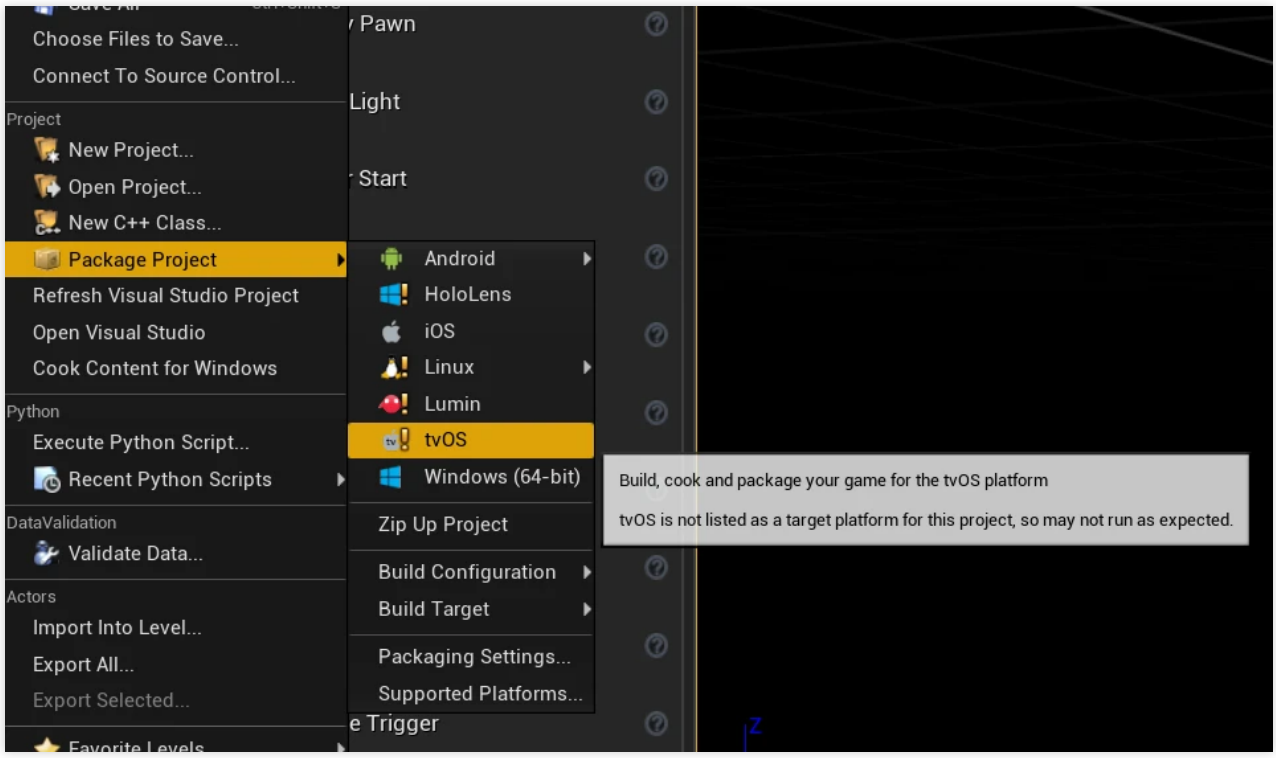
Windows 端

iOS 端

Android 端

File -> Package Project -> Mac

File->Package Project->Windows->Windows(64-bit)



打包项目

File -> Package Project-> iOS

1. 开发调试：详见 Android 快速入门。
2. 打包项目：详见 打包 Android 项目。

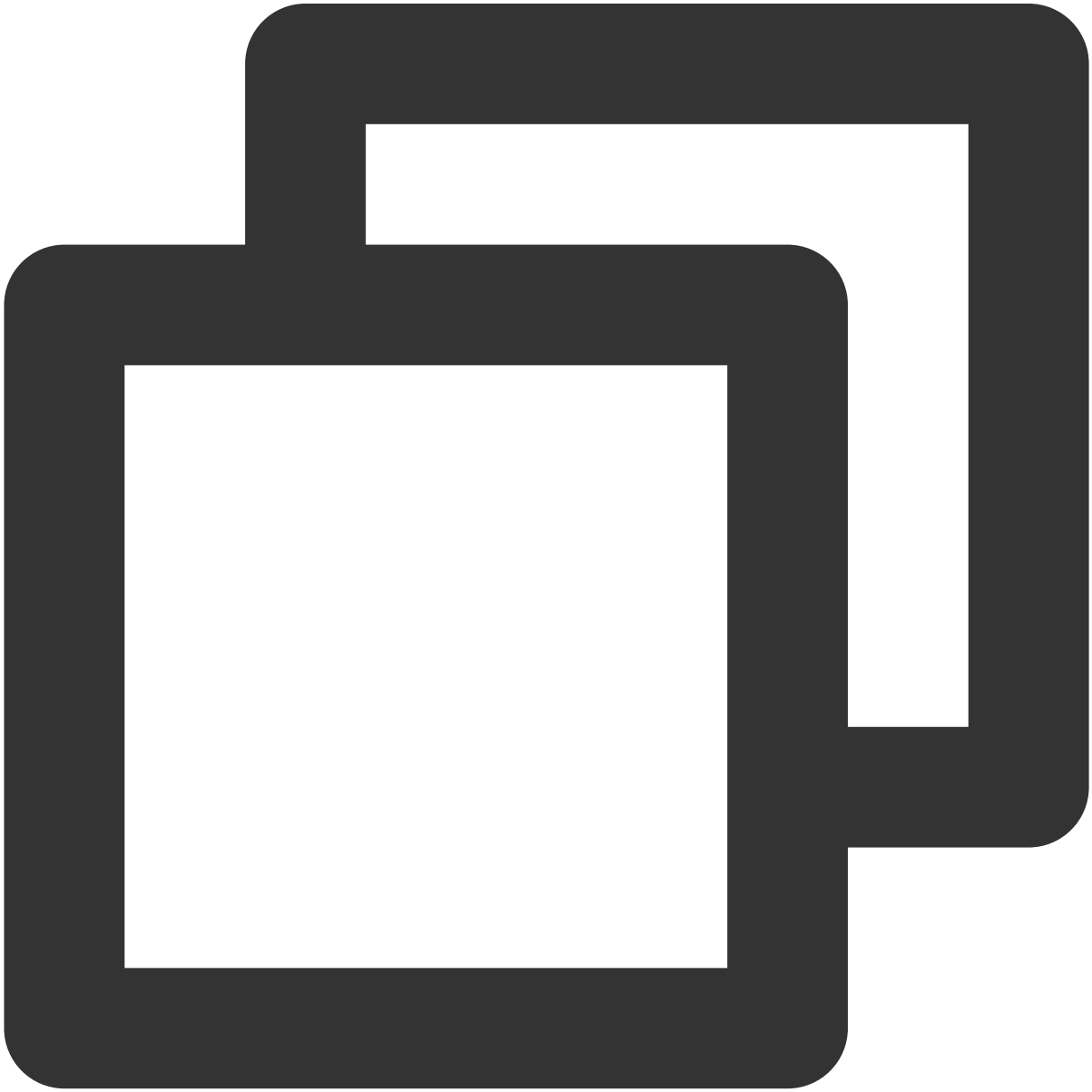
IM Unreal Engine API 文档

更多接口介绍，请参见 API 概览。

常见问题

Android“Attempt to construct staged filesystem reference from absolute path""报错

关闭 UE4 项目，打开 CMD，运行如下命令：



```
adb shell

cd sdcard

ls (you should see the UE4Game directory listed)

rm -r UE4Game
```

重新编译项目。

Flutter

最近更新时间：2024-05-13 16:44:33

通过阅读本文，您可以了解集成 Flutter SDK 的方法。

环境要求

环境	版本
Flutter	IM SDK 最低要求 Flutter 3.0.0 版本，TUIKit 集成组件库最低要求 Flutter 3.16.0 版本。
Android	Android Studio Dolphin | 2021.3.1 及以上版本，App 要求 Android 7.0 及以上版本设备。
iOS	Xcode 12.0 及以上版本，请确保您的项目已设置有效的开发者签名。

支持平台

我们致力于打造一套支持 Flutter 全平台的即时通信 IM SDK 及 TUIKit，帮助您一套代码，全平台运行。

平台	无 UI SDK (tencent_cloud_chat_sdk)	UIKit (tencent_cloud_chat_uikit)	UIKit V2 (tencent_cloud_chat)
iOS	支持	支持	支持
Android	支持	支持	支持
Web	支持，4.1.1+2版本起	支持，0.1.5版本起	*即将支持...*
macOS	支持，4.1.9版本起	支持，2.0.0版本起	支持
Windows	支持，4.1.9版本起	支持，2.0.0版本起	支持
混合开发 （将 Flutter SDK 添加至现有原生应用）	5.0.0版本起支持	1.0.0版本起支持	支持

说明

Web 和 macOS 平台需要简单的几步额外引入，详情请查看本文 拓展更多平台。

前序工作

1. 您已 注册腾讯云，并完成 实名认证。
2. 参照 创建并升级应用 创建应用，并记录好 `SDKAppID` 。
3. 在 IM 控制台 选择您的应用，在左侧导航栏依次点击 **辅助工具->UserSig 生成&校验** ，创建两个 UserID 及其对应的 UserSig，复制 `UserID` 、 `签名（Key）` 、 `UserSig` 这三个，后续登录时会用到。

说明

该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是由服务器端生成，并提供面向 App 的接口，在需要 `UserSig` 时由 App 向业务服务器发起请求获取动态 `UserSig` 。更多详情请参见 服务端生成 UserSig。

选择合适的方案集成 Flutter SDK

IM 提供了三种方式来集成，您可以选择最合适的方案来集成：

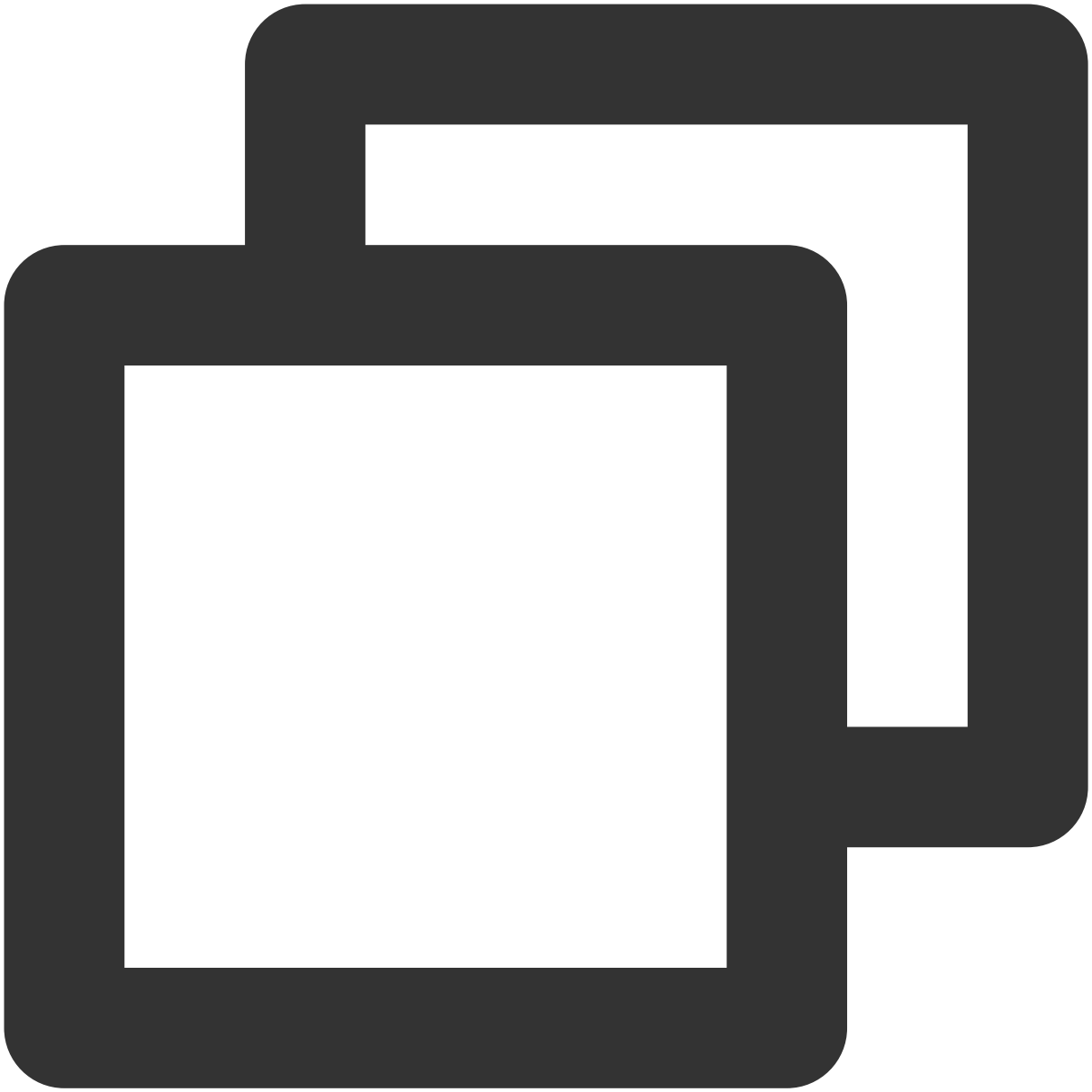
集成方式	适用场景
使用 DEMO 修改	IM Demo 是一个完整的聊天 App，代码已开源，如果您需要实现聊天类似场景，可以使用 Demo 进行二次开发。 可立即 体验 Demo。
含 UI 集成	IM 的 UI 组件库 `UIKit` 提供了通用的 UI 组件，例如会话列表、聊天界面和联系人列表等，开发者可根据实际业务需求通过该组件库快速地搭建自定义 IM 应用。**推荐优先使用该方案**。
自实现 UI 集成	如果 UIKit 不能满足您应用的界面需求，或者您需要比较多的定制，可以使用该方案。

方案一：使用 Demo 修改

本 Demo 程序，由新版 UIKit V2 构建。

跑通 Demo

1. **下载 Demo 源码、安装依赖：**

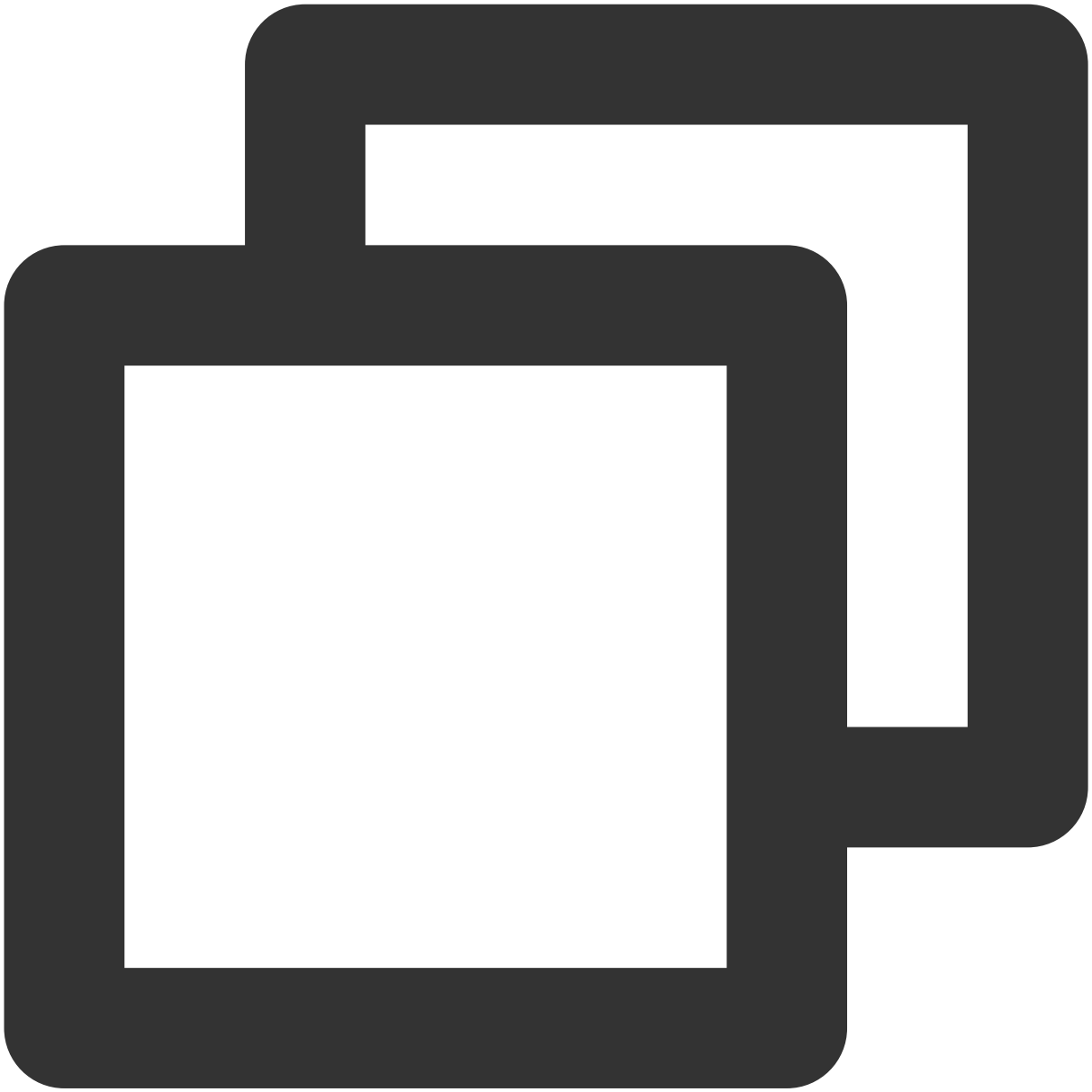


```
# Clone the code
git clone https://github.com/TencentCloud/chat-demo-flutter.git

# Checkout the 'v2' branch
git checkout v2

# Clean the project. Important
flutter clean

# Install dependencies
flutter pub get
```

2. **配置用户信息用于登录。**

打开 `lib/config.dart` , 并且配置 `sdkappid` , `userid` , and `usersig` 从此前步骤获取的。

3. **运行应用.**



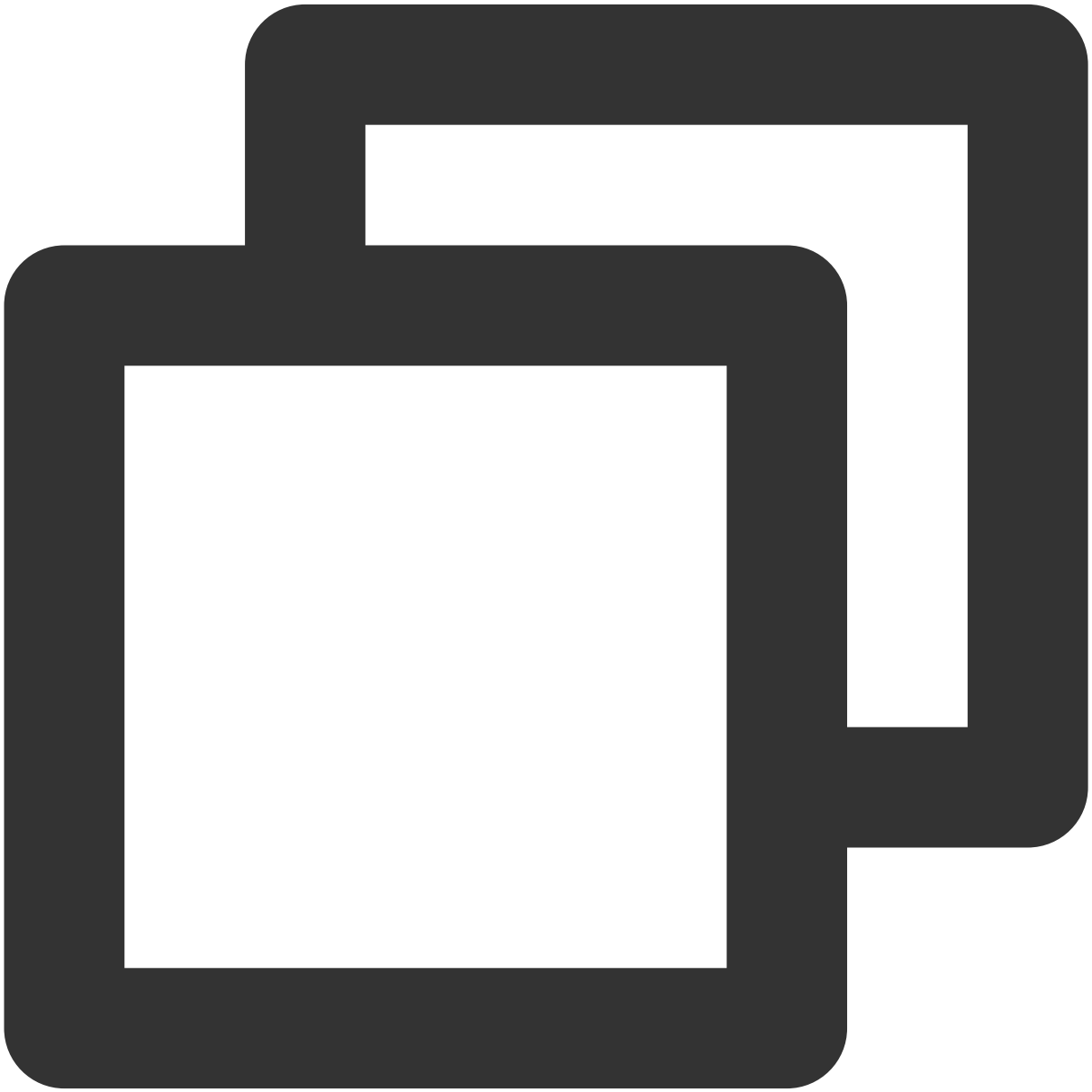
```
flutter run
```

Demo 代码结构概览

文件夹	介绍

lib	程序核心目录
lib/desktop	桌面端适配相关代码
lib/setting	个人中心及设置页面相关代码

`lib` 目录中的 `lib/main.dart` 文件，为本 Demo 程序运行的核心文件。可以看到，有下面几行关键代码。将 `List<Widget> pages` 的组件列表，传入 `bottomNavigation` 进行页面切换。



```
pages = [
  const TencentCloudChatConversation(),
```

```
    const TencentCloudChatContact(),
    TencentCloudChatSettings(
      removeSettings: removeLocalSetting,
      setLoginState: changeLoginState,
    ),
  ];
```

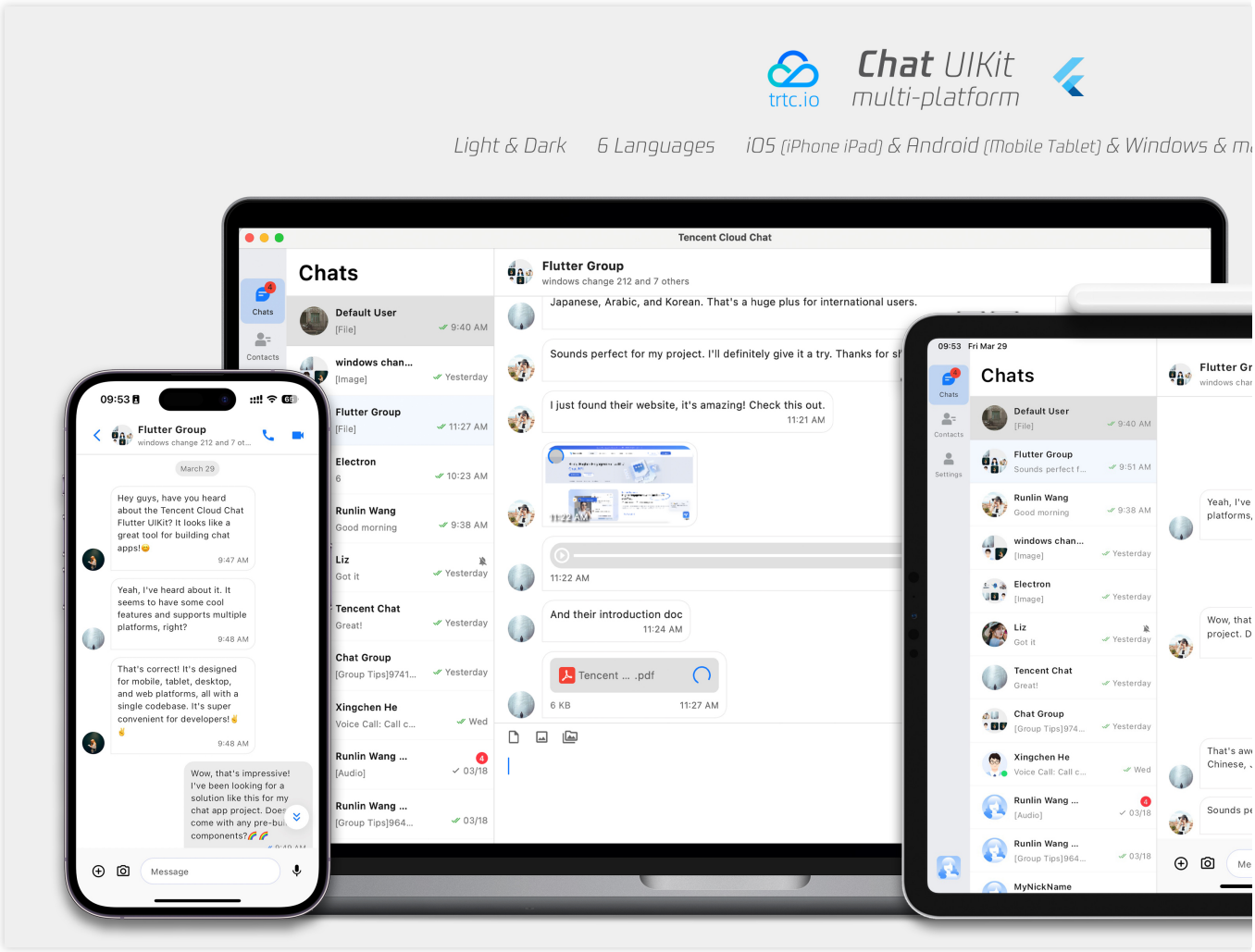
尽管新版 UIKit 有许多组件，但这些组件内置支持互相路由导航及引用关联。即，最简化使用，只需手动使用 `TencentCloudChatConversation` 会话列表组件和 `TencentCloudChatContact` 联系人组件，即可使用到全部 UIKit 关联组件，包括聊天组件，联系人及群组详情组件等。这些组件均可通过我们手动引入的组件，跳转过去。

如果您对于 UIKit 的使用有较大自定义需求，可以查看 UIKit 的快速入门文档。

方案二：含 UI 集成，使用 UIKit 组件库，快速集成聊天能力

我们的全新 Flutter Chat UIKit V2 旨在为开发者提供一套全面的工具，以便轻松创建功能丰富的聊天应用程序。它采用模块化方法构建，让您可以选择所需的组件，同时保持应用程序轻量级和高效。

UIKit 包括许多功能，如会话列表、消息处理、联系人列表、用户和群组资料、搜索功能等。



此处仅简要说明引入使用 UIKit 的方式，详细快速入门文档请参阅此处。

兼容性

我们的 UIKit 支持**手机端**, **平板端**和**桌面端** UI 样式，并兼容 Android、iOS、macOS、Windows 和 Web（将在未来版本中支持）。

它内置支持英语、简体中文、繁体中文、日语、韩语和阿拉伯语（支持阿拉伯RTL界面）以及亮色和暗色外观样式。

要求

Flutter版本：3.16或更高

Dart版本：3.0或更高

开始使用

要开始使用我们的UIKit，首先导入基础包，tencent_cloud_chat。

接下来，从以下列表中导入适合您需求的所需UI组件包：

tencent_cloud_chat_message

tencent_cloud_chat_conversation

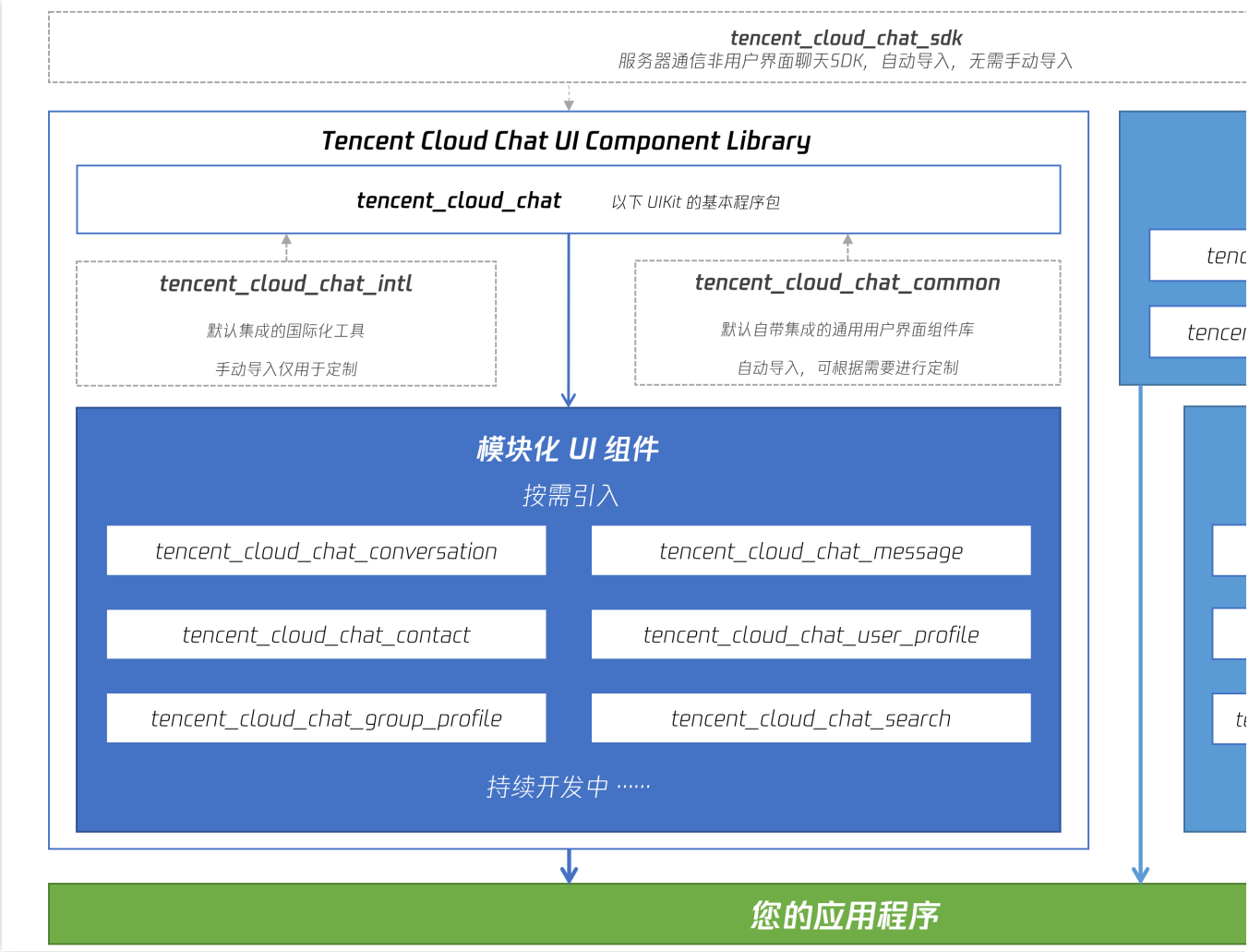
tencent_cloud_chat_contact

tencent_cloud_chat_user_profile

tencent_cloud_chat_group_profile

tencent_cloud_chat_group_search *(In Beta)*

下面展示了我们的UIKit的架构：



集成

有关 Chat UIKit 的集成，请参见 **集成基本功能-Flutter**。

方案三：自实现 UI 集成

前提条件

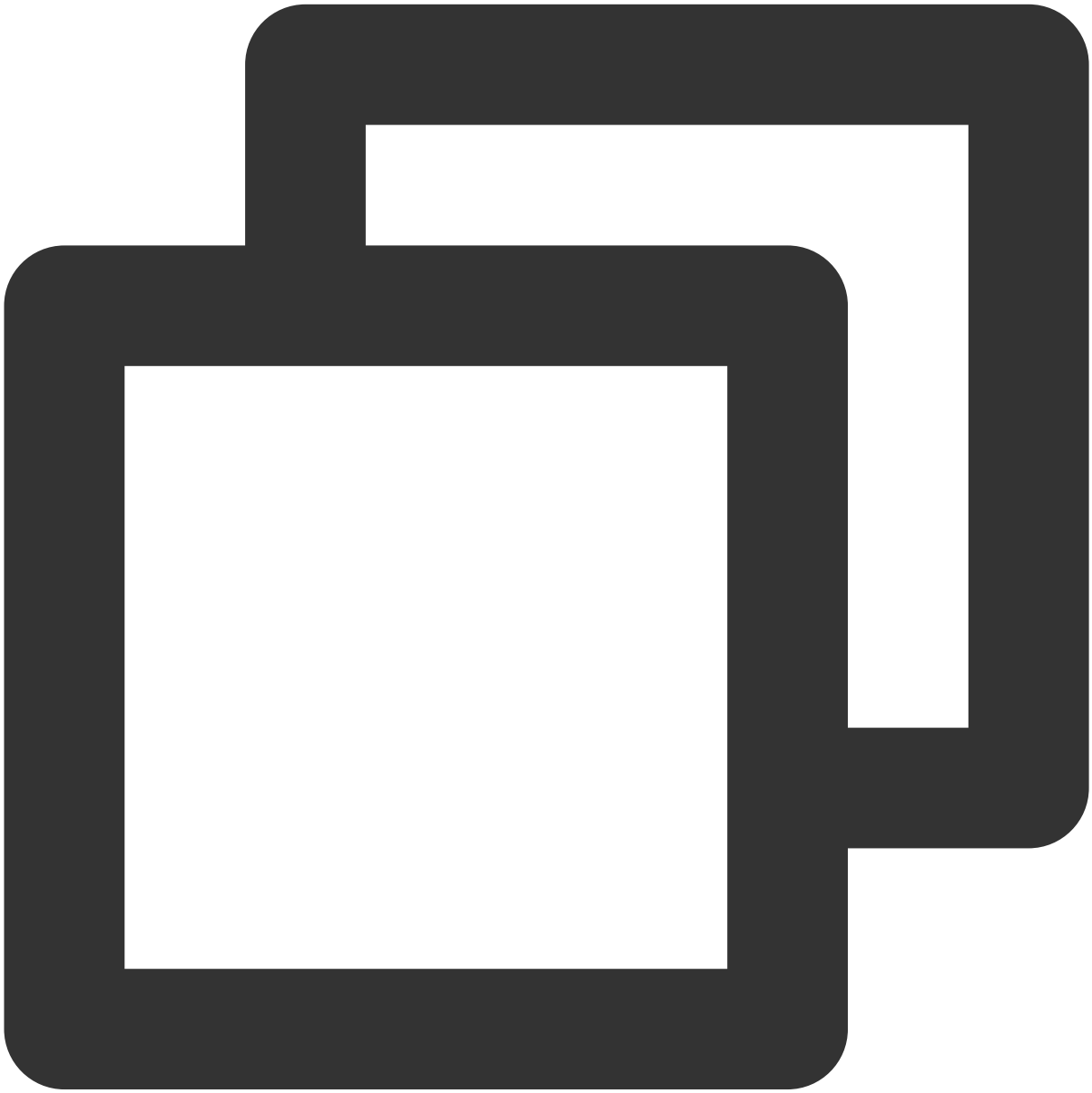
您已经完成创建 Flutter 项目，或有可以基于的 Flutter 项目。

接入步骤

安装 IM SDK

本节详细文档

使用如下命令，安装 Flutter IM SDK 最新版本。



```
flutter pub add tencent_cloud_chat_sdk
```

说明：

如果您的项目还同时需要用于 Web 或 桌面端(macOS、Windows)，一些额外的步骤是需要的，具体情况各自的链

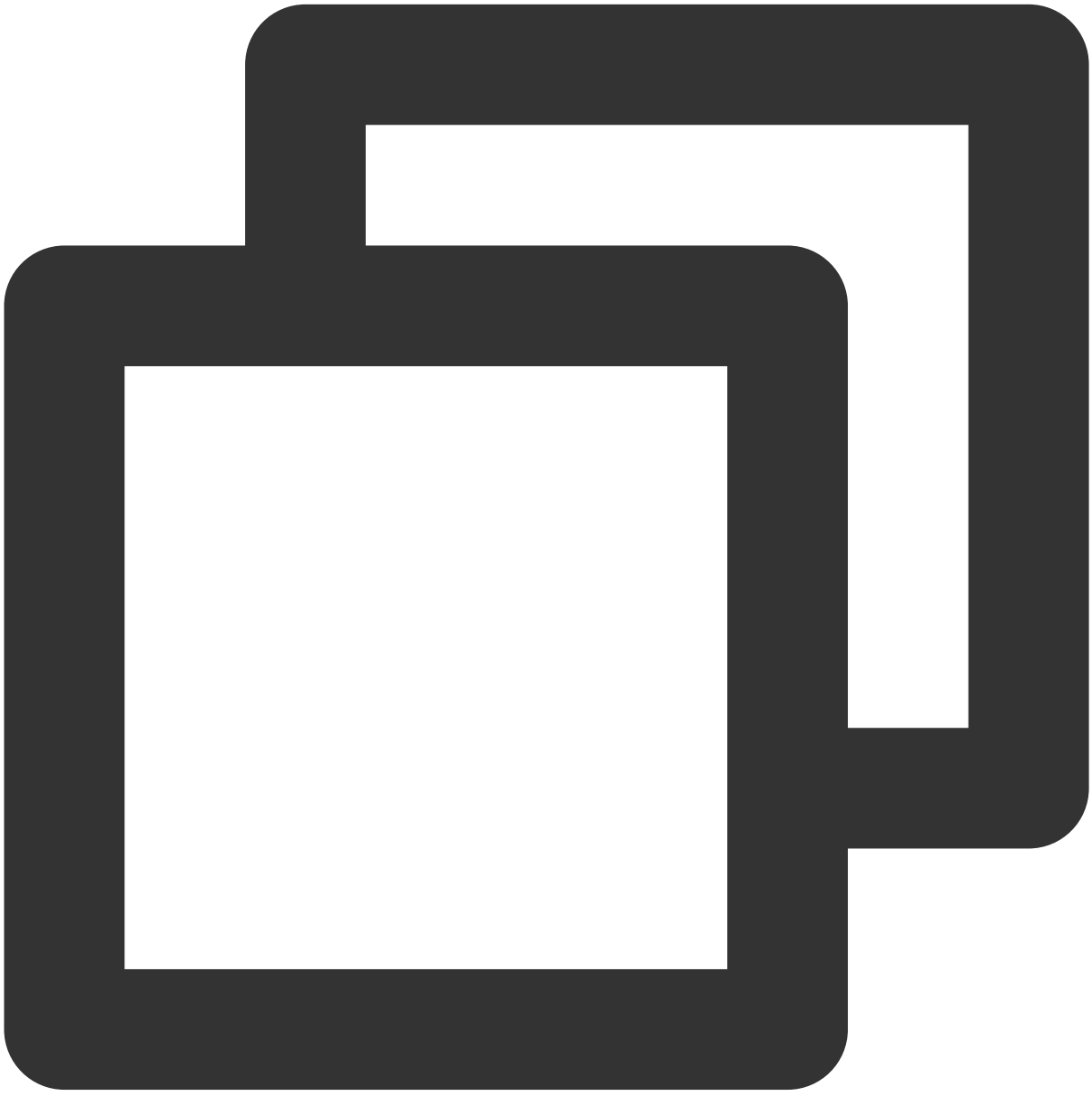
接。

完成 SDK 初始化

本节详细文档

调用 `initSDK` ，完成 SDK 初始化。

将您的 `sdkAppID` 传入。



```
import 'package:tencent_cloud_chat_sdk/enum/V2TimSDKListener.dart';
import 'package:tencent_cloud_chat_sdk/enum/log_level_enum.dart';
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';
```

```
TencentImSDKPlugin.v2TIMManager.initSDK(
  sdkAppID: 0, // Replace 0 with the SDKAppID of your IM application when integrati
  loglevel: LogLevelEnum.V2TIM_LOG_DEBUG, // Log
  listener: V2TimSDKListener(),
);
```

在本步骤，您可以针对 IM SDK 挂载一些监听，主要包括网络状态及用户信息变更等，详情可参见 该文档。

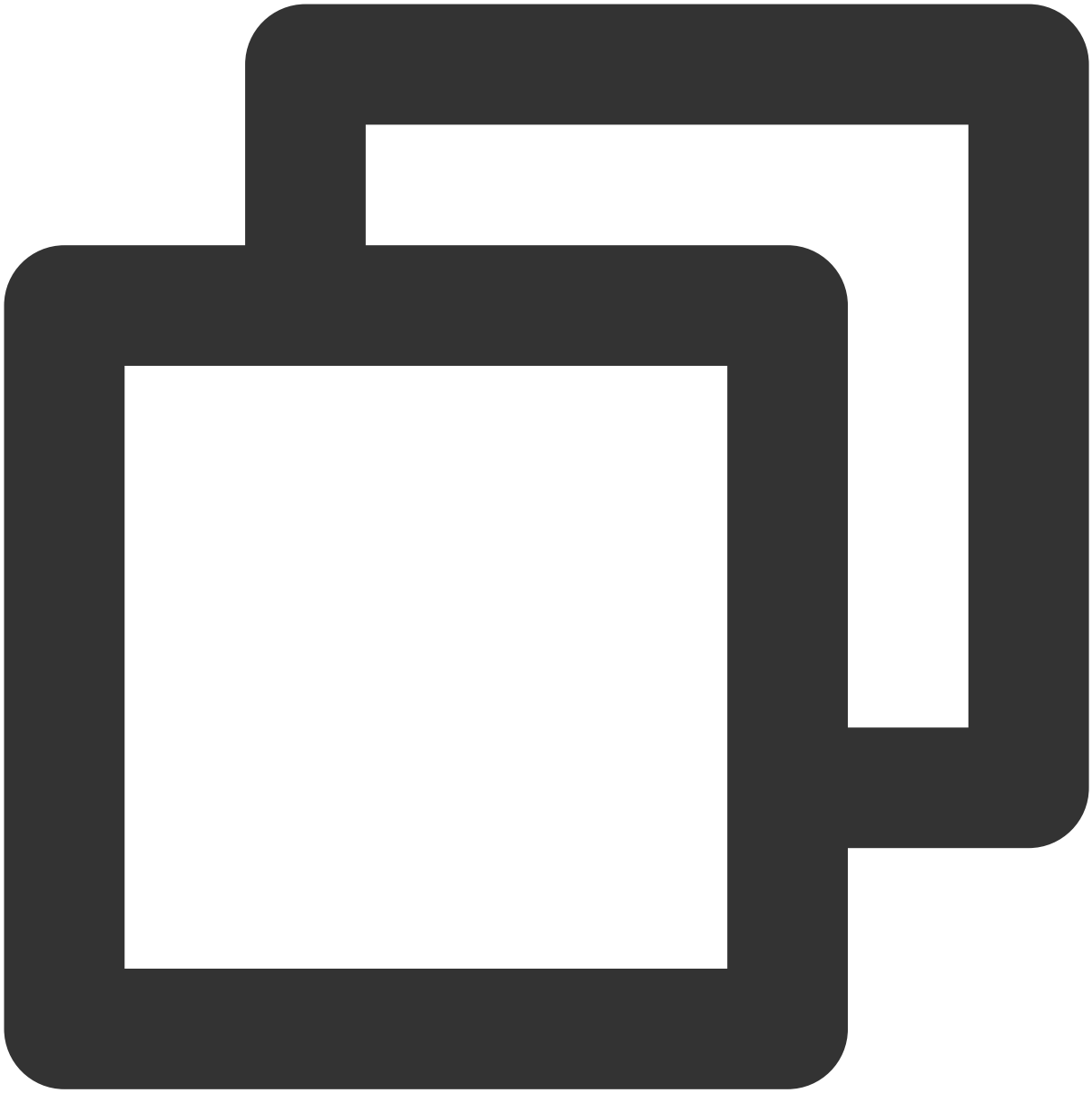
登录测试账户

本节详细文档

此时，您可以使用最开始的时候，在控制台生成的测试账户，完成登录验证。

调用 `TencentImSDKPlugin.v2TIMManager.login` 方法，登录一个测试账户。

当返回值 `res.code` 为0时，登录成功。



```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';
V2TimCallback res = await TencentImSDKPlugin.v2TIMManager.login(
  userID: userID,
  userSig: userSig,
);
```

说明

该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是将 `UserSig` 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 `UserSig` 时由您的 App 向业务服务器发起请求获取动态 `UserSig` 。更多详情请参见 服务端生成 UserSig。

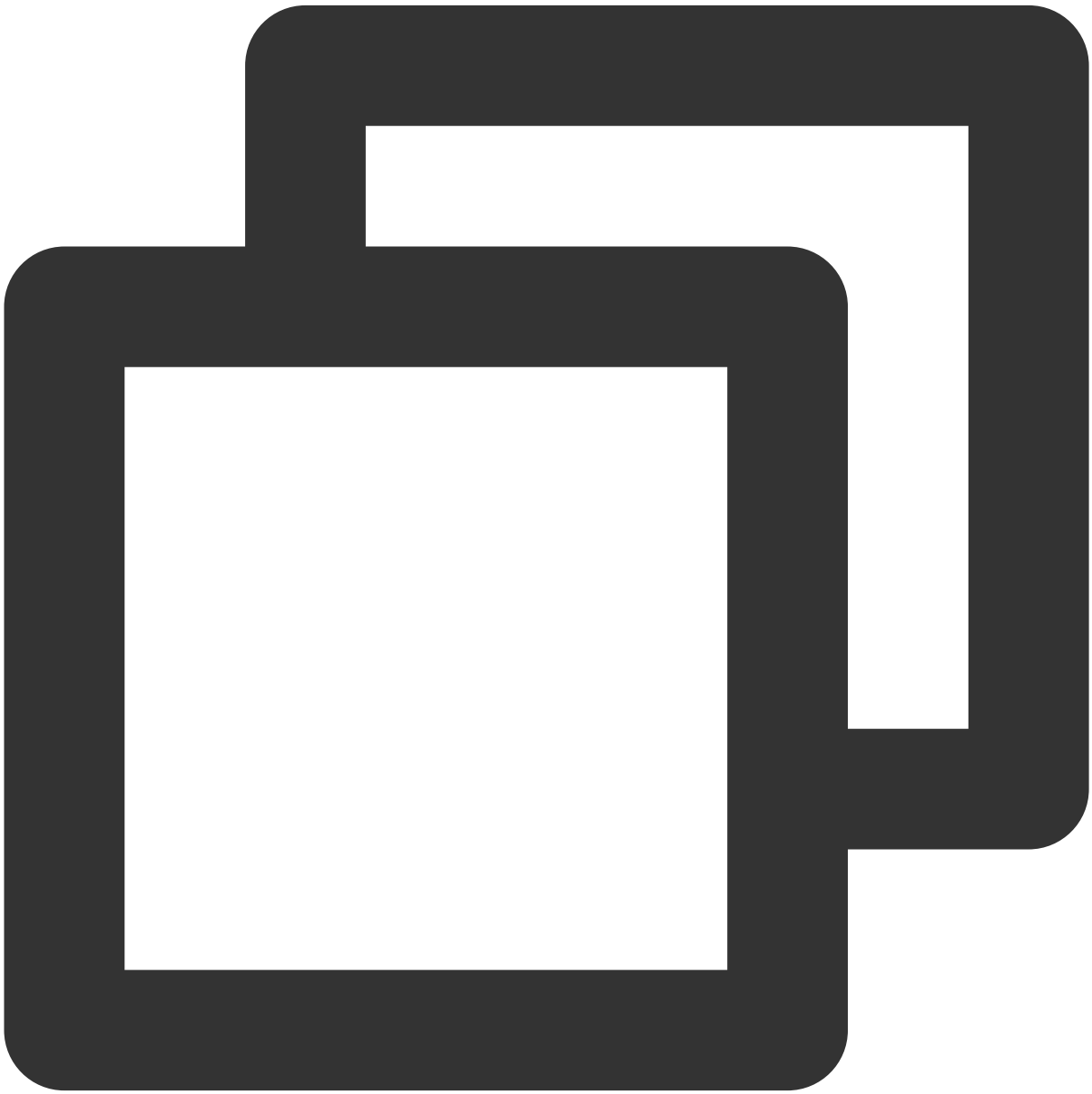
发送消息

本节详细文档

此处以发送文本消息举例，其流程为：

1. 调用 `createTextMessage(String)` 创建一个文本消息。
2. 根据其返回值，拿到消息 ID。
3. 调用 `sendMessage()` 发送该ID的消息。 `receiver` 可填入您此前创建的另一个测试账户 ID。发送单聊消息无需填入 `groupID` 。

代码示例：



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

V2TimValueCallback<V2TimMsgCreateInfoResult> createMessage =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createTextMessage(text: "The text to create");

String id = createMessage.data!.id!; // The message creation ID

V2TimValueCallback<V2TimMessage> res = await TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .sendMessage(
          id: id, // Pass in the message creation ID to
          receiver: "The userID of the destination user",
          groupID: "The groupID of the destination group",
          );
```

说明

如果发送失败，可能是由于您的 sdkAppID 不支持陌生人发送消息，您可至控制台开启，用于测试。

请单击此链接，关闭好友关系链检查。

获取会话列表

本节详细文档

在上一个步骤中，完成发送测试消息，现在可登录另一个测试账户，拉取会话列表。

获取会话列表的方式有两种：

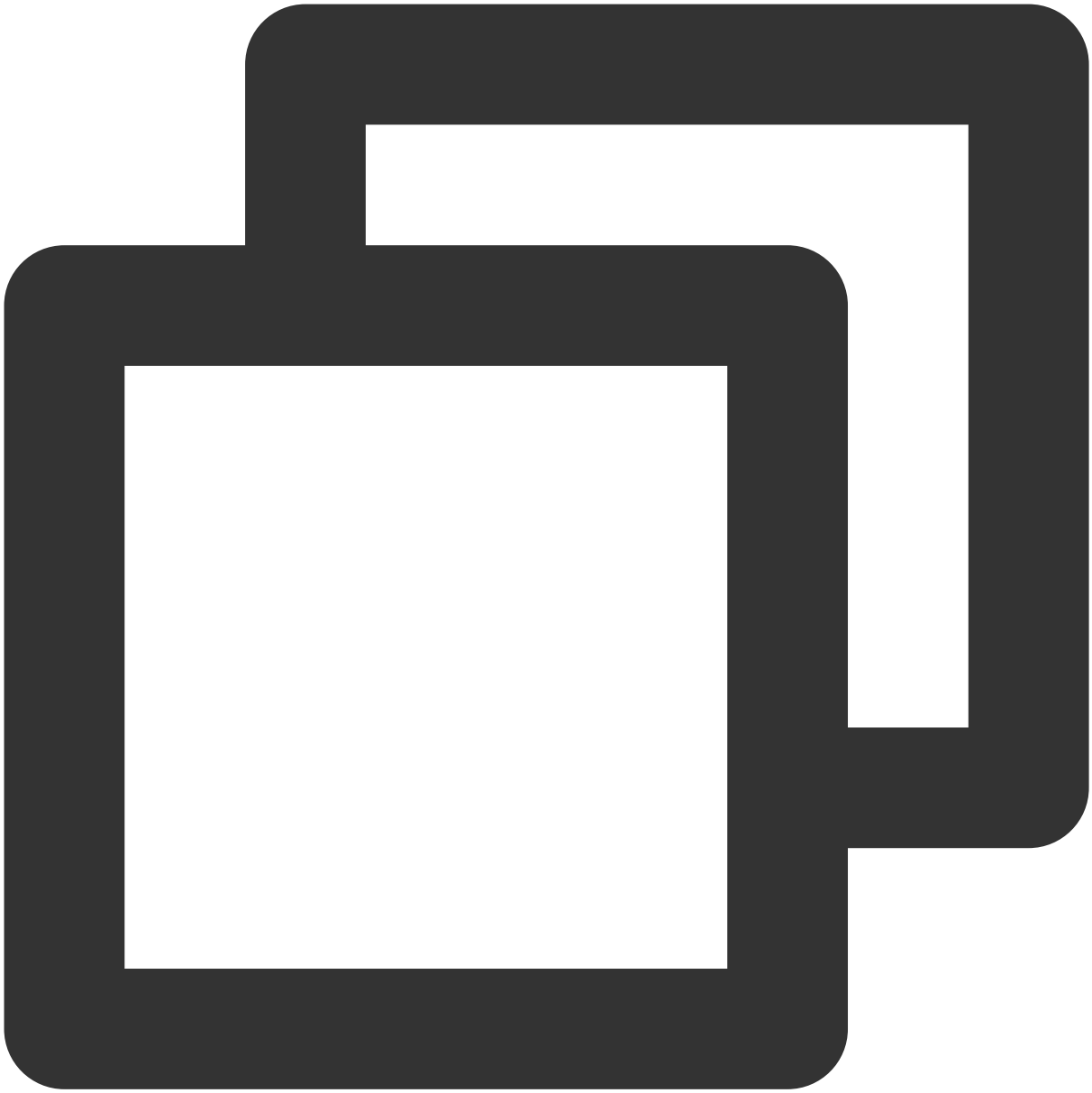
1. 监听长连接回调，实时更新会话列表。
2. 请求 API，根据分页一次性获取会话列表。

常见应用场景为：

在启动应用程序后立即获取会话列表，然后监听长连接以实时更新会话列表的变化。

一次性请求会话列表

为了获取会话列表，需要维护 `nextSeq` ，记录当前位置。



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

String nextSeq = "0";

getConversationList() async {
  V2TimValueCallback<V2TimConversationResult> res = await TencentImSDKPlugin
      .v2TIMManager
      .getConversationManager()
      .getConversationList(nextSeq: nextSeq, count: 10);

  nextSeq = res.data?.nextSeq ?? "0";
```

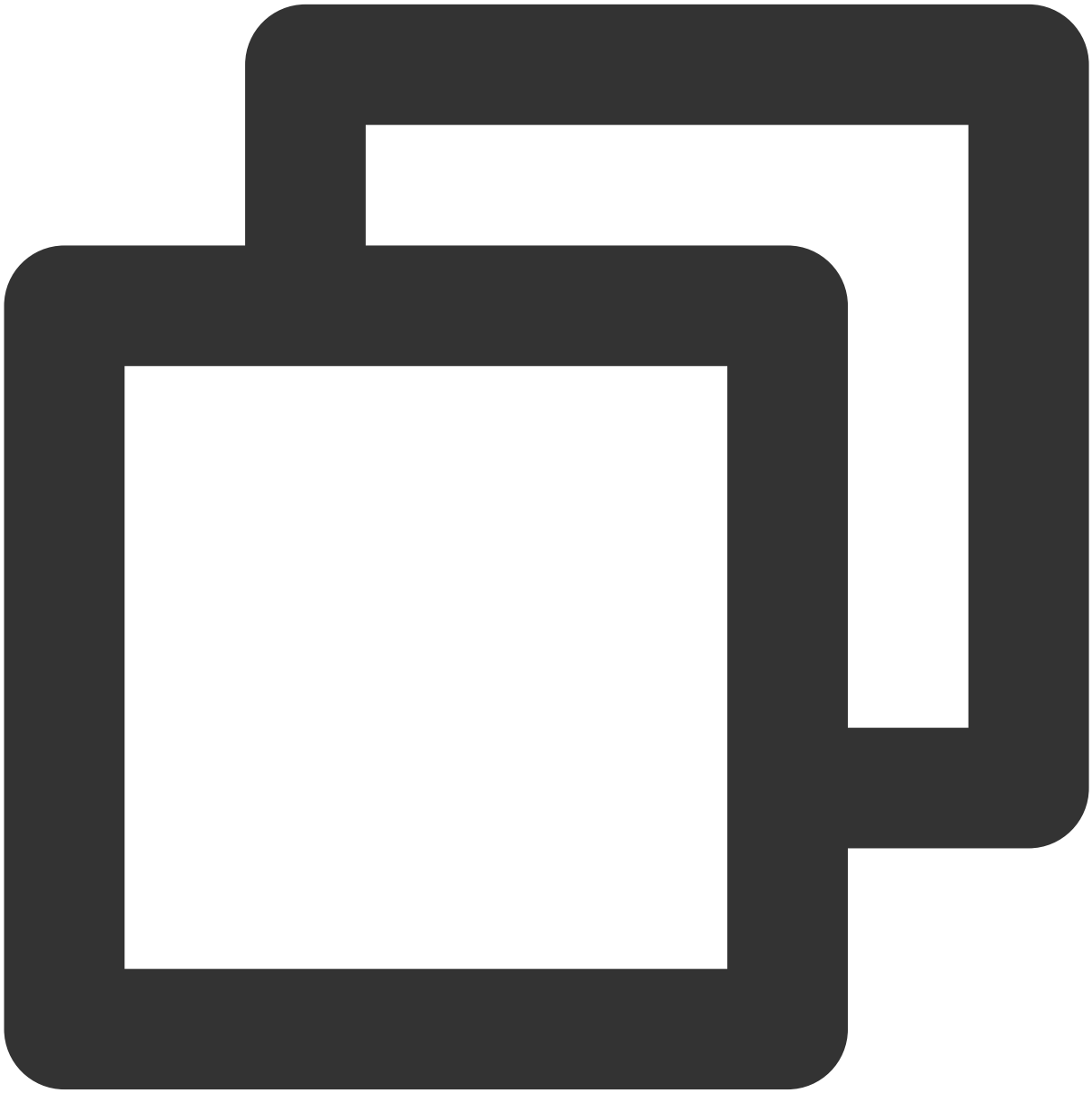
```
}
```

此时，您可以看到您在上一步中，使用另一个测试账号，发来消息的会话。

监听长链接实时获取会话列表

您在此步骤中，需要先在 SDK 上挂载监听，然后处理回调事件，更新 UI。

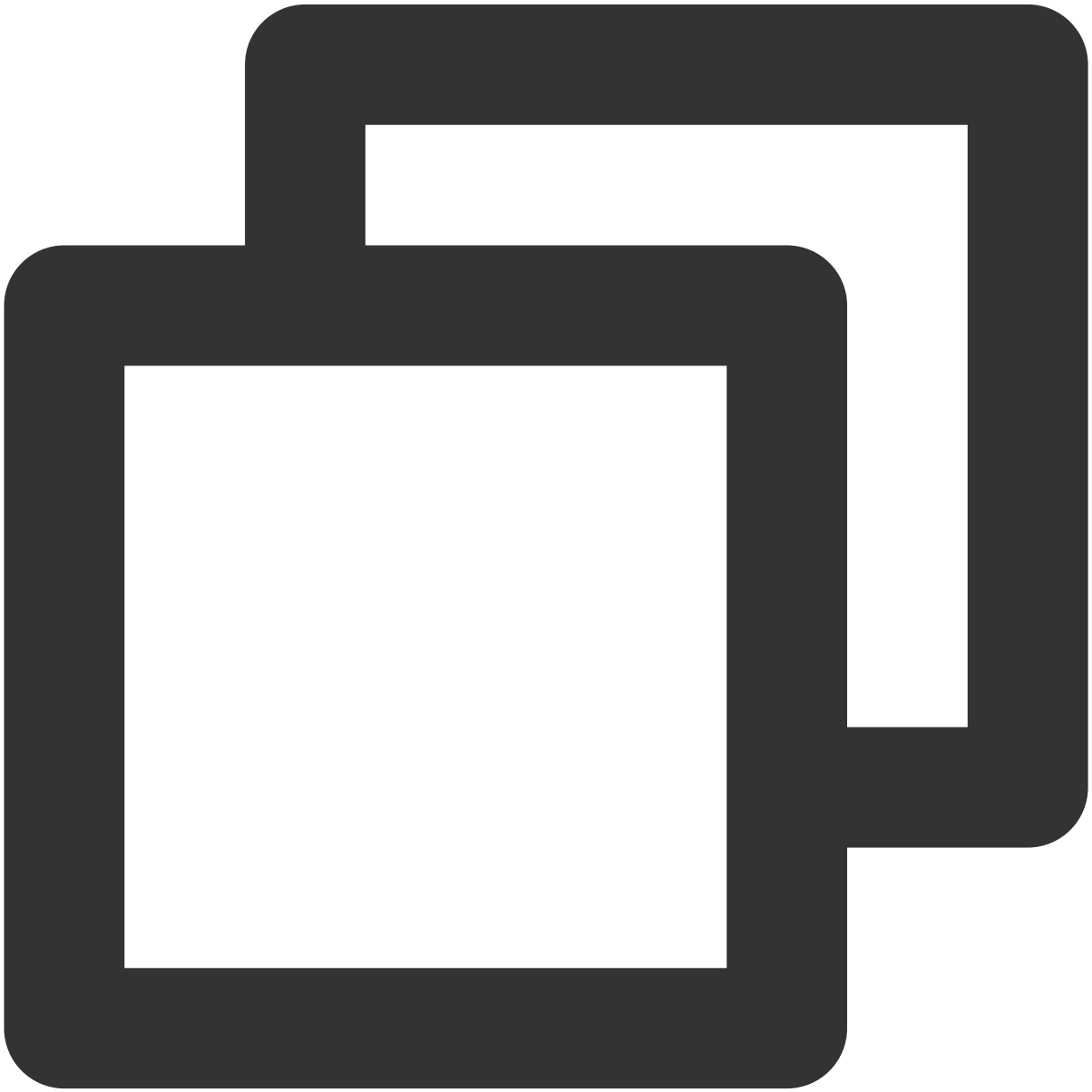
1. 挂载监听。



```
await TencentImSDKPlugin.v2TIMManager
 .getConversationManager()
 .setConversationListener(
```

```
    listener: new V2TimConversationListener(
      onConversationChanged: (List<V2TimConversation> list){
        _onConversationListChanged(list);
  },
      onNewConversation:(List<V2TimConversation> list){
        _onConversationListChanged(list);
  },
```

2. 处理回调事件，将最新的会话列表展示在界面上。



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

```
List<V2TimConversation> _conversationList = [];

_onConversationListChanged(List<V2TimConversation> list) {
  for (int element = 0; element < list.length; element++) {
 int index = _conversationList.indexWhere(
   (item) => item!.conversationID == list[element].conversationID);
 if (index > -1) {
 _conversationList.setAll(index, [list[element]]);
 } else {
 _conversationList.add(list[element]);
 }
  }
```

接收消息

本节详细文档

通过腾讯云 IM Flutter SDK 接收消息有两种方式：

1. 监听长连接回调，实时获取消息变化，更新渲染历史消息列表。
2. 请求 API，根据分页一次性获取历史消息。

常见应用场景为：

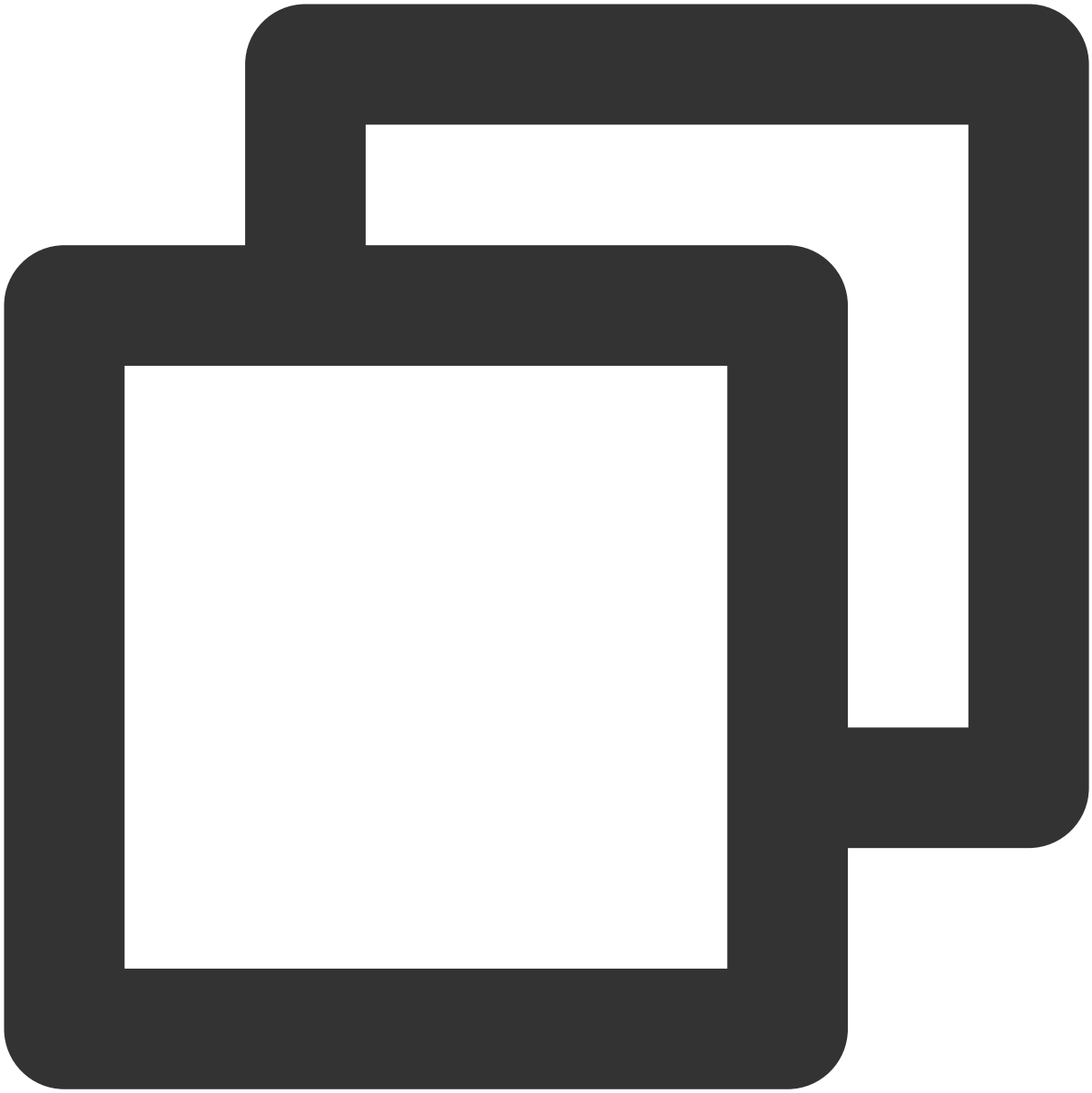
1. 界面进入新的会话后，首先一次性请求一定数量的历史消息，用于展示历史消息列表。
2. 监听长链接，实时接收新的消息，将其添加进历史消息列表中。

一次性请求历史消息列表

每页拉取的消息数量不能太大，否则会影响拉取速度。建议此处设置为20左右。

您应该动态记录当前页数，用于下一轮请求。

示例代码如下：



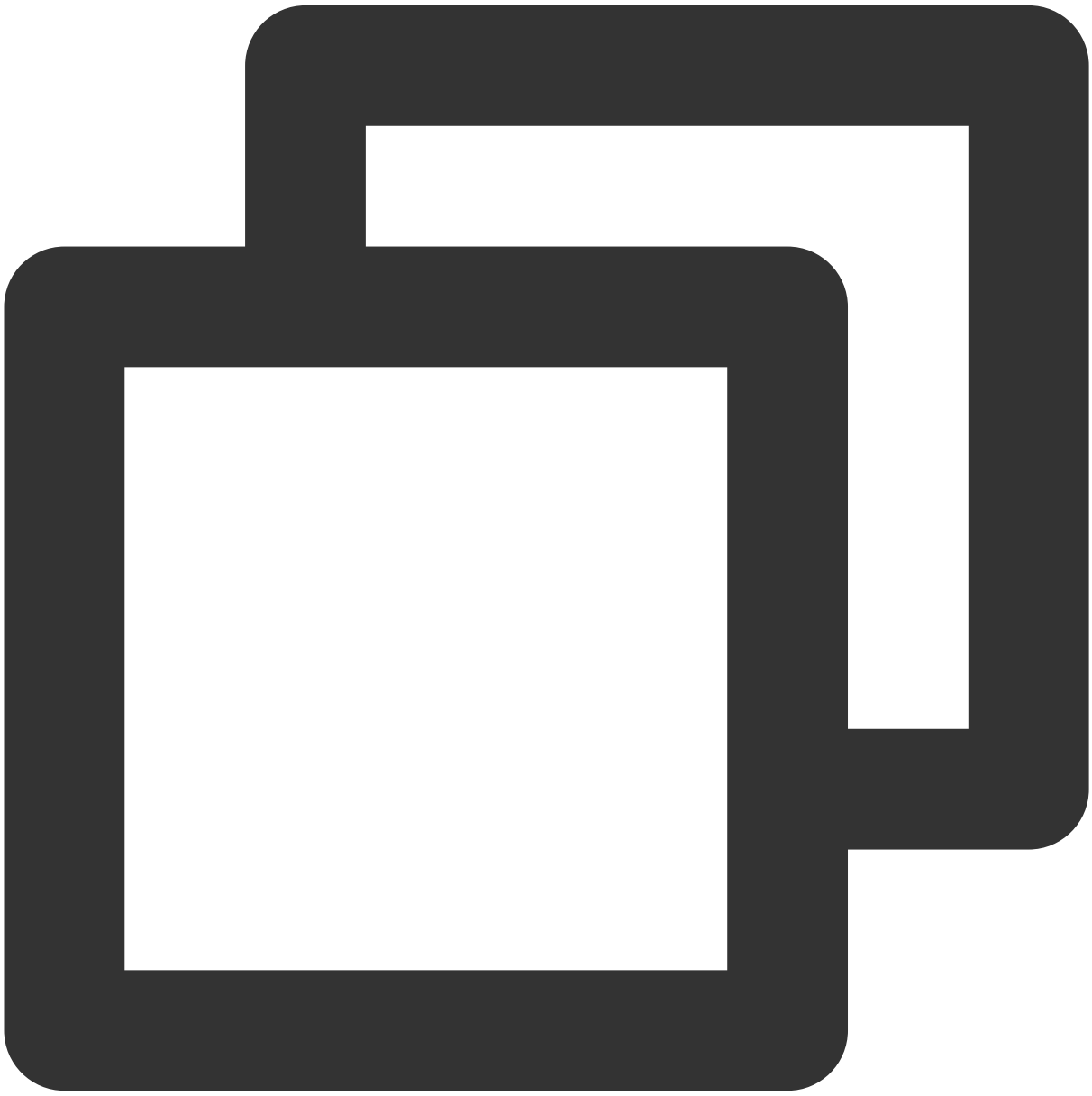
```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';

  V2TimValueCallback<List<V2TimMessage>> res = await TencentImSDKPlugin
      .v2TIMManager
      .getMessageManager()
      .getGroupHistoryMessageList(
        groupID: "groupID",
        count: 20,
        lastMsgID: "",
      );
```

```
    List<V2TimMessage> msgList = res.data ?? [];

    // here you can use msgList to render your message list
      }
```

监听长链接实时获取新消息

历史消息列表初始化后，新消息来自长链接 `V2TimAdvancedMsgListener.onRecvNewMessage` 。`onRecvNewMessage` 回调被触发后，您可以按需将新消息添加进历史消息列表中。

绑定监听器示例代码如下：



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

final adVancesMsgListener = V2TimAdvancedMsgListener(
onRecvNewMessage: (V2TimMessage newMsg) {
  _onReceiveNewMsg(newMsg);
},
/// ... other listeners related to message
);

TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .addAdvancedMsgListener(listener: adVancesMsgListener);
```

此时，您已基本完成 IM 模块开发，可以发送接收消息，也可以进入不同的会话。
您可以继续完成 群组，用户资料，关系链，离线推送，本地搜索 等相关功能开发。
详情可查看 自实现 UI 集成 SDK 文档。

拓展更多平台

腾讯云IM for Flutter 相关SDK默认支持 Android, iOS 和 Windows 平台， 如果您需要拓展更多平台（Web 和 macOS），请参考本部分。

Web

我们的 SDK，TUIKit(tencent_cloud_chat_uikit) 0.1.5版本，无 UI SDK(tencent_cloud_chat_sdk) 4.1.1+2 版本起，可兼容 Web 端。
相比 Android 和 iOS 端，需要一些额外步骤。如下：

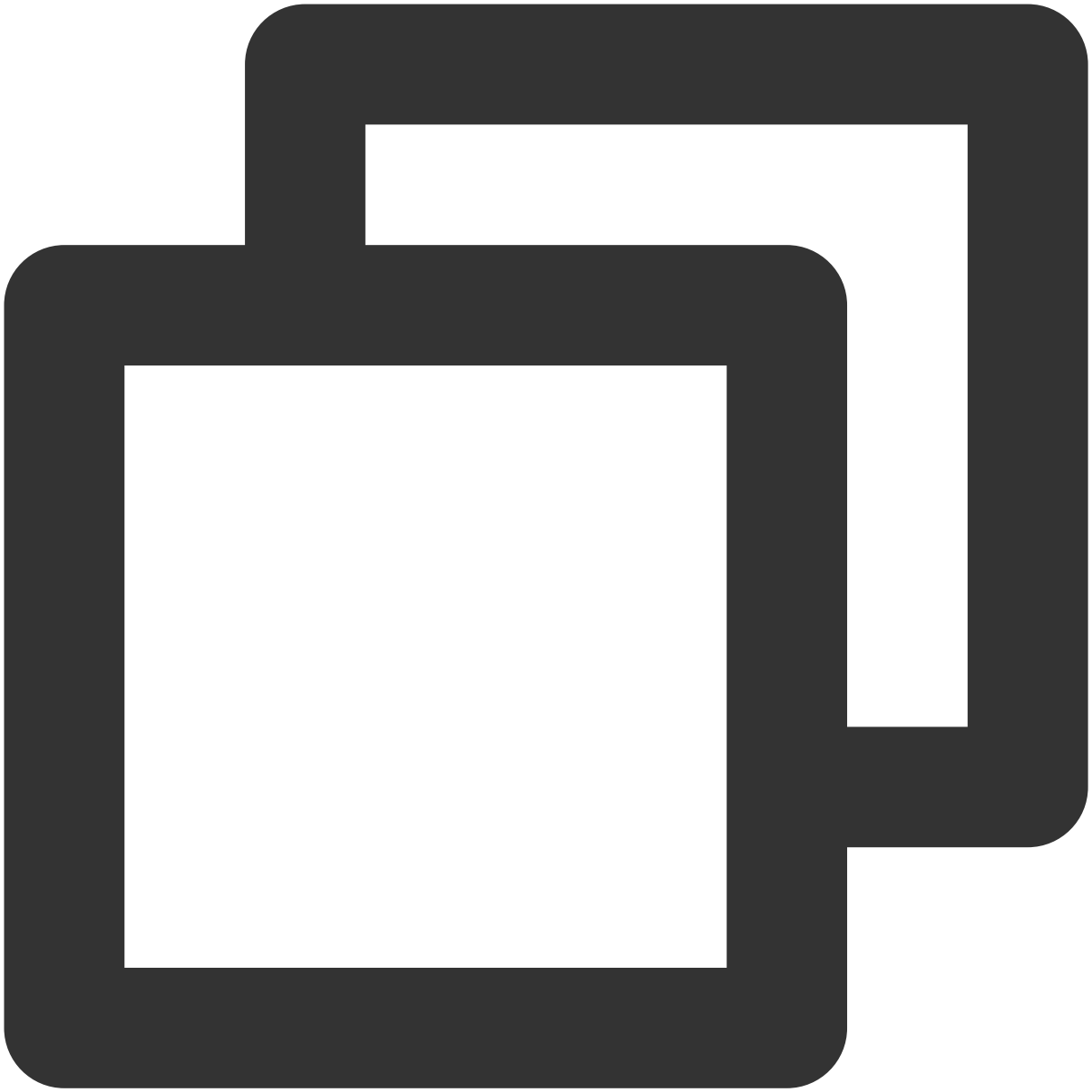
升级 Flutter 3.x 版本

Flutter 3.x 版本 针对 Web 性能做了较多优化，强烈建议您使用其来开发 Flutter Web 项目。

引入 JS

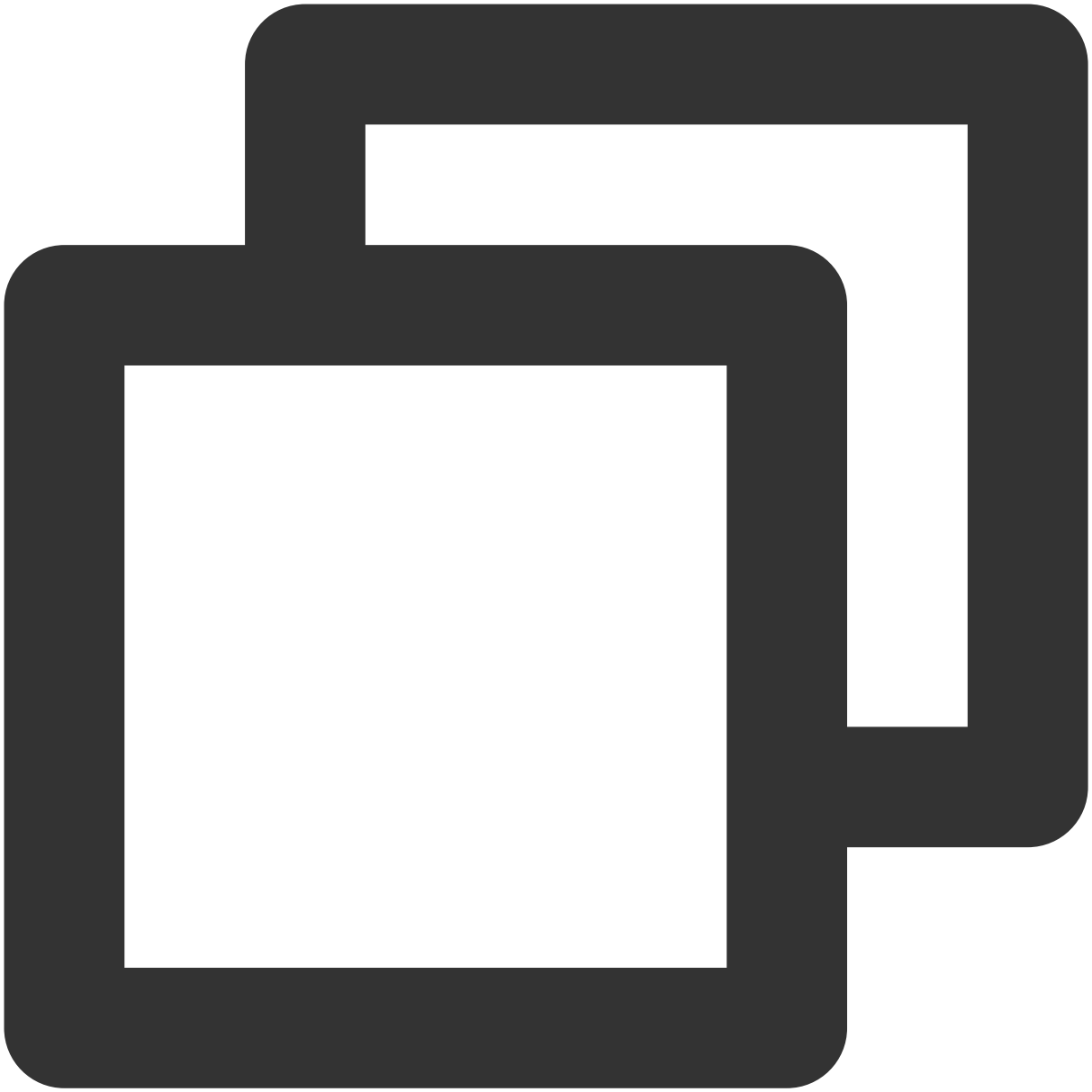
说明

如果您现有的 Flutter 项目不支持 Web，请在项目根目录下运行 `flutter create .` 添加 Web 支持。
进入您项目的 `web/` 目录，使用 `npm` 或 `yarn` 安装相关JS依赖。初始化项目时，根据屏幕指引，进行即可。

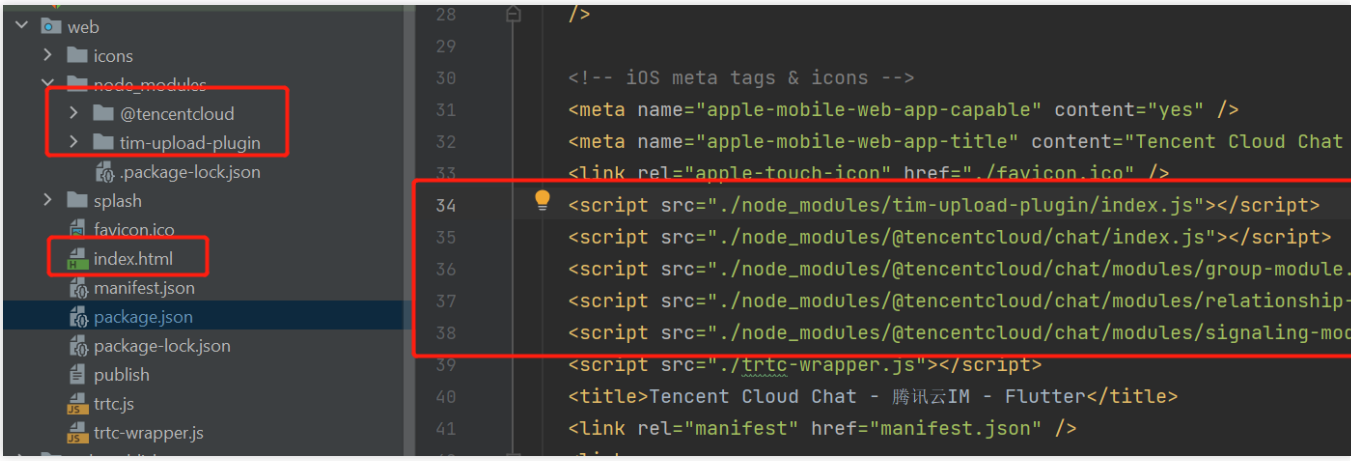


```
cd web

npm init

npm i @tencentcloud/chat

npm i tim-upload-plugin
```

打开 `web/index.html` ，在 `<head> </head>` 间引入这JS文件。如下：



```
<script src="./node_modules/tim-upload-plugin/index.js"></script>
<script src="./node_modules/@tencentcloud/chat/index.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/group-module.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/relationship-module.js"></sc
<script src="./node_modules/@tencentcloud/chat/modules/signaling-module.js"></scrip
```

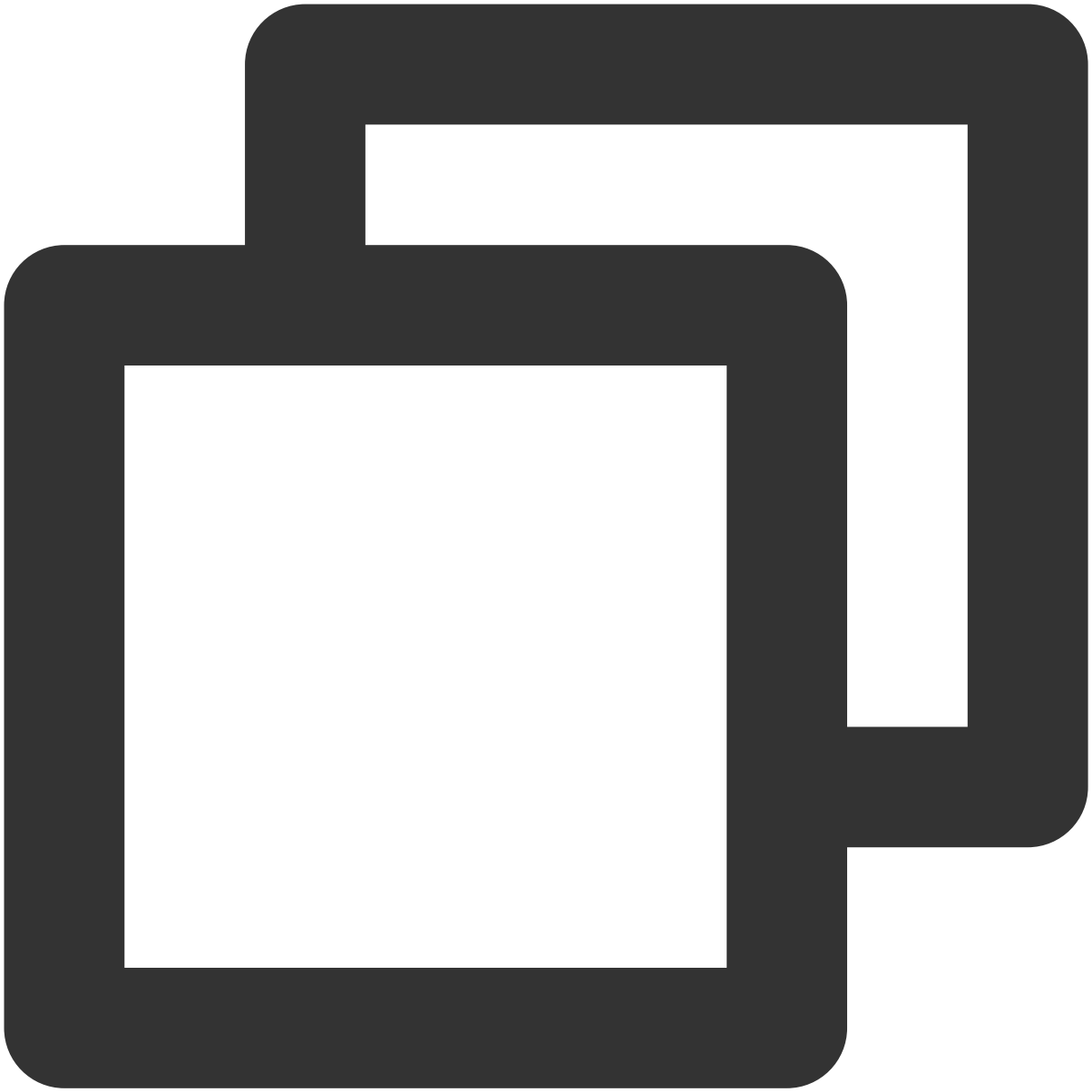


macOS

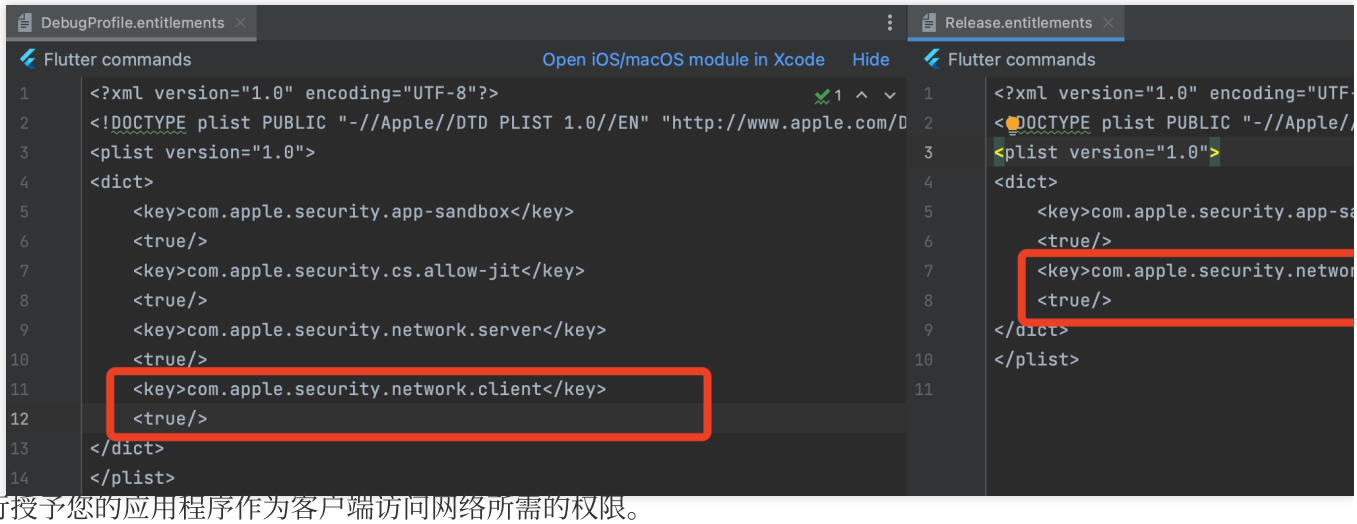
macOS 平台需要其他配置。按照以下步骤配置 macOS 平台：

1. 打开项目中的 `macos/Runner/DebugProfile.entitlements` 和 `macos/Runner/Release.entitlements` 文件。

2. 将以下行添加到每个文件:



```
<key>com.apple.security.network.client</key>
<true/>
```



这些行授予您的应用程序作为客户端访问网络所需的权限。

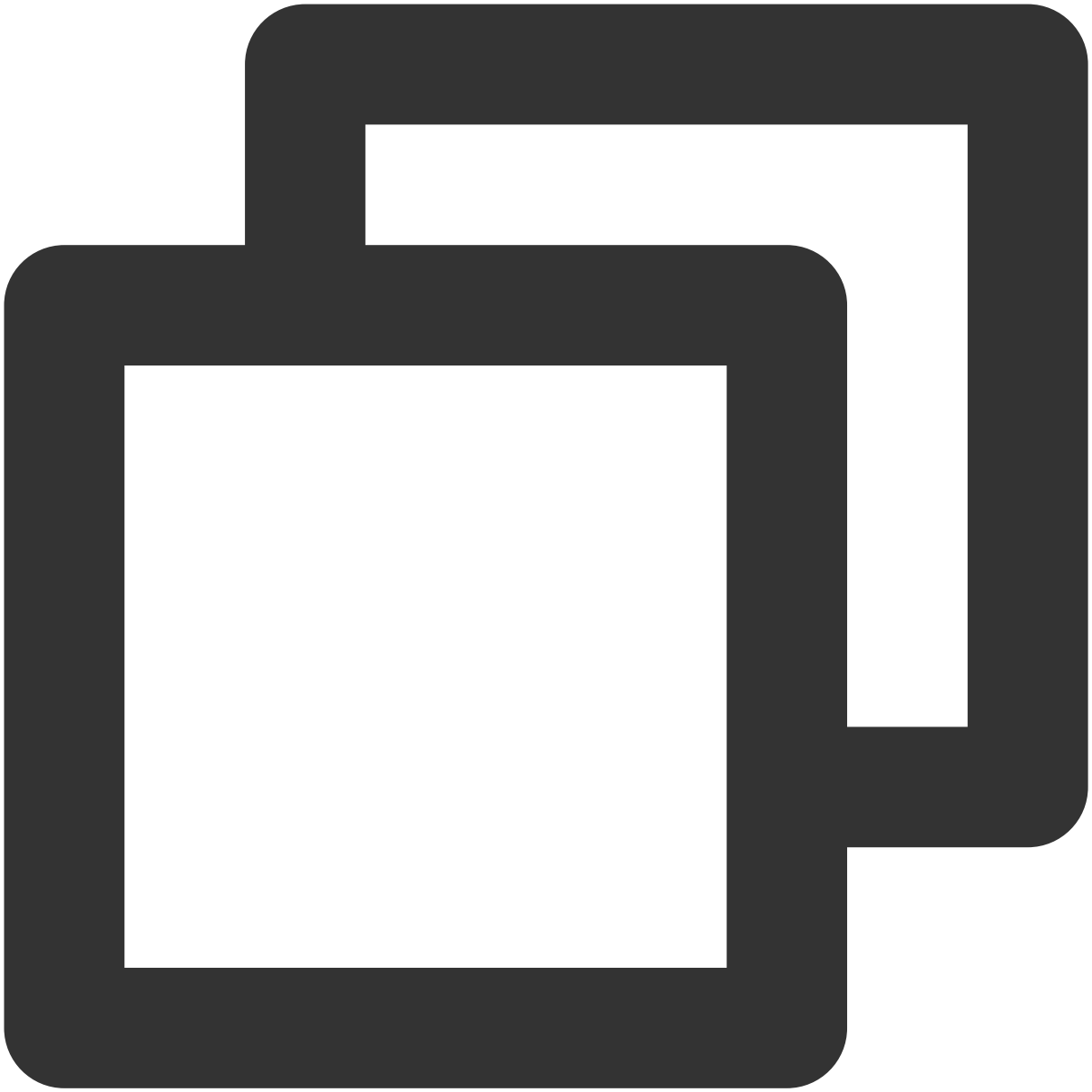
此配置对于确保您的应用程序与 macOS 平台上的后端服务之间的正确通信至关重要。

常见问题

iOS 端 Pods 依赖无法安装成功

尝试方案一：配置运行后，如果报错，可以单击 Product > Clean Build Folder，清除产物后重新 `pod install` 或 `flutter run`

尝试方案二：手动删除 `ios/Pods` 文件夹，及 `ios/Podfile.lock` 文件，并执行如下命令，重新安装依赖

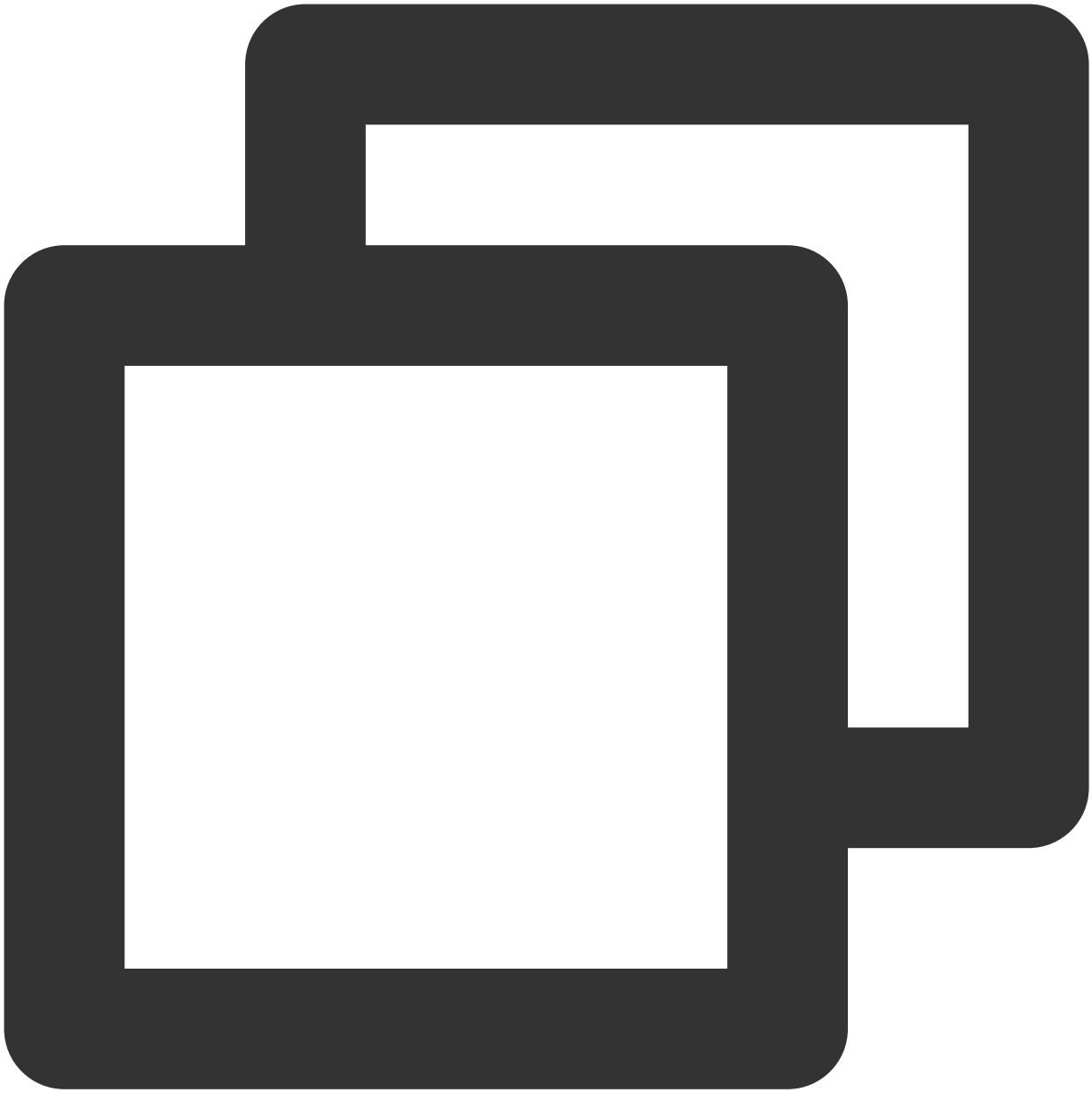


```
cd ios
sudo gem install ffi
pod install --repo-update
```

Flutter 环境问题

如您需得知 Flutter 的环境是否存在问题，请运行 Flutter doctor 检测 Flutter 环境是否装好。

Windows 运行 Demo 报错怎么办？



```
Nuget.exe not found, trying to download or use cached version.
```

这个提示表明您的系统中没有找到 `nuget.exe`，因此程序尝试下载或使用缓存的版本。`nuget.exe` 是 NuGet 包管理器的命令行工具，用于在 `.NET` 项目中管理依赖项。

要解决此问题，您可以手动下载并安装 `nuget.exe`。以下是操作步骤：

1. 访问 NuGet 官方网站的下载页面：https://www.nuget.org/downloads
2. 在 "Command-line" 部分，找到 "Windows x86 Commandline"，点击 "Download" 下载 `nuget.exe`。
3. 将下载的 `nuget.exe` 文件保存到一个合适的位置，例如 `C:\\NuGet`。

4. 将 `nuget.exe` 的路径添加到系统的环境变量 `PATH` 中。这样，您就可以在命令行中全局访问 `nuget.exe` 。

错误码如何查询？

SDK API 层面错误码，请查看 该文档。

UIKit 相关错误及 UI 用户通知相关特有事件, 请查看 该文档.

React Native

最近更新时间：2024-05-13 16:44:50

本文主要介绍如何快速运行腾讯云即时通信 IM Demo（React Native）。

环境要求

平台	版本
React Native	0.63.4 版本以上
Android	Android Studio 3.5 及以上版本，App 要求 Android 4.1 及以上版本设备。
iOS	Xcode 11.0 及以上版本，请确保您的项目已设置有效的开发者签名。

前提条件

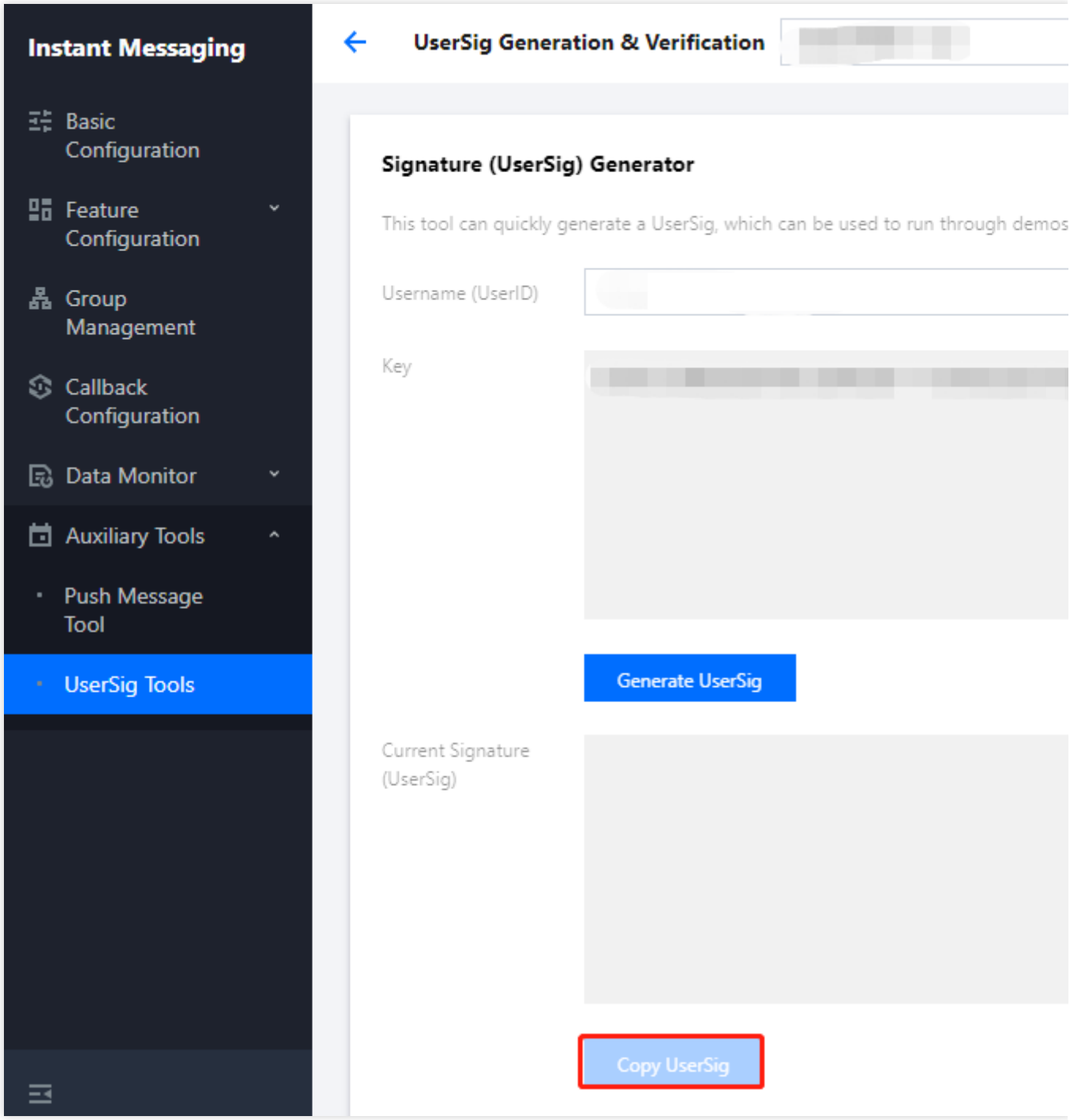
您已 注册腾讯云 帐号，并完成 实名认证。

第一部分：创建测试用户

在 IM 控制台 选择您的应用，在左侧导航栏依次点击 **辅助工具->UserSig 生成&校验**，创建两个 UserID 及其对应的 UserSig，复制 `UserID`、`签名（Key）`、`UserSig` 这三个，后续登录时会用到。

说明：

该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是由服务器端生成，并提供面向 App 的接口，在需要 `UserSig` 时由 App 向业务服务器发起请求获取动态 `UserSig`。更多详情请参见 服务端生成 UserSig。



第二部分：集成 React Native SDK

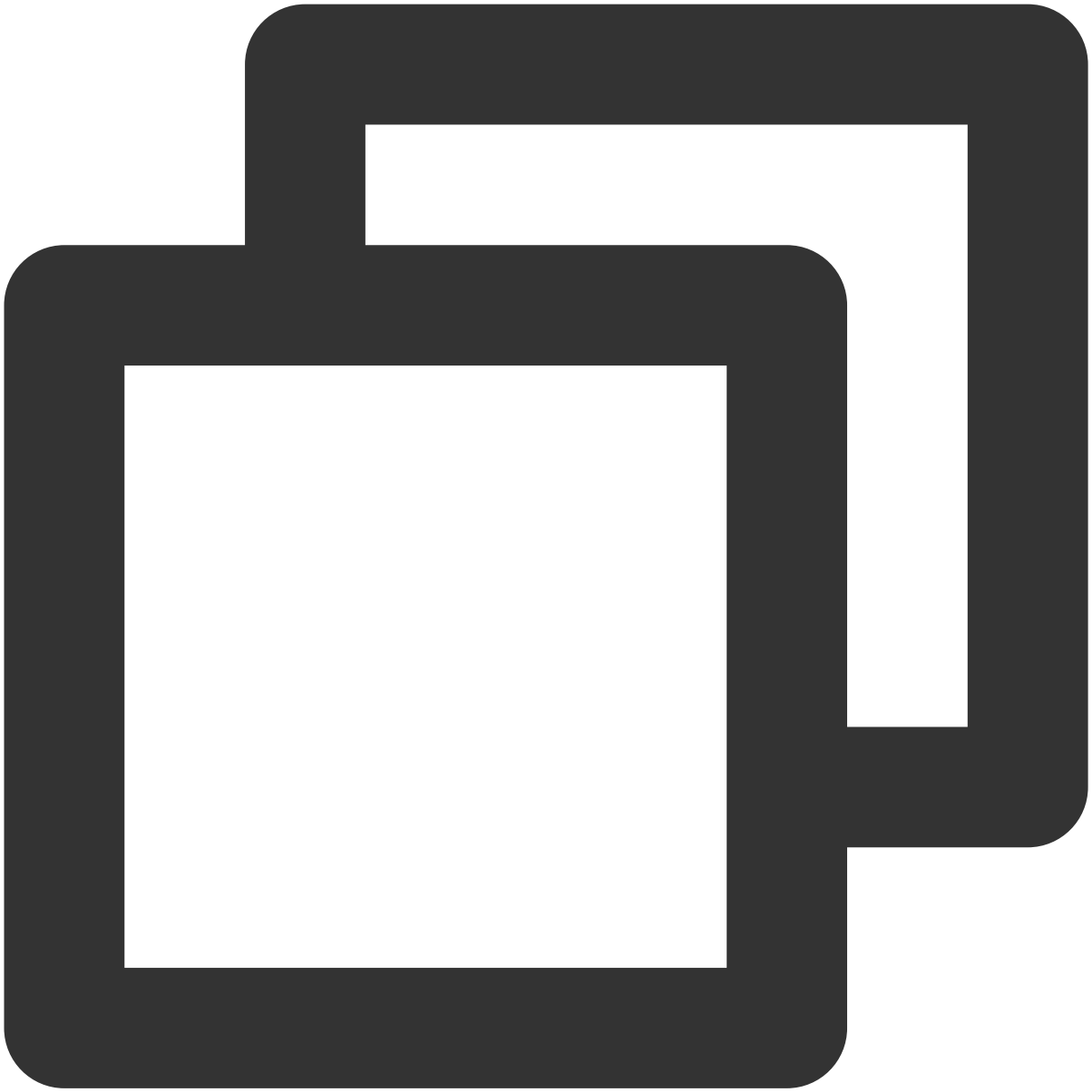
前提条件

您已经完成创建 React Native 项目，或有可以基于的 React Native 项目。

接入步骤

安装 IM SDK

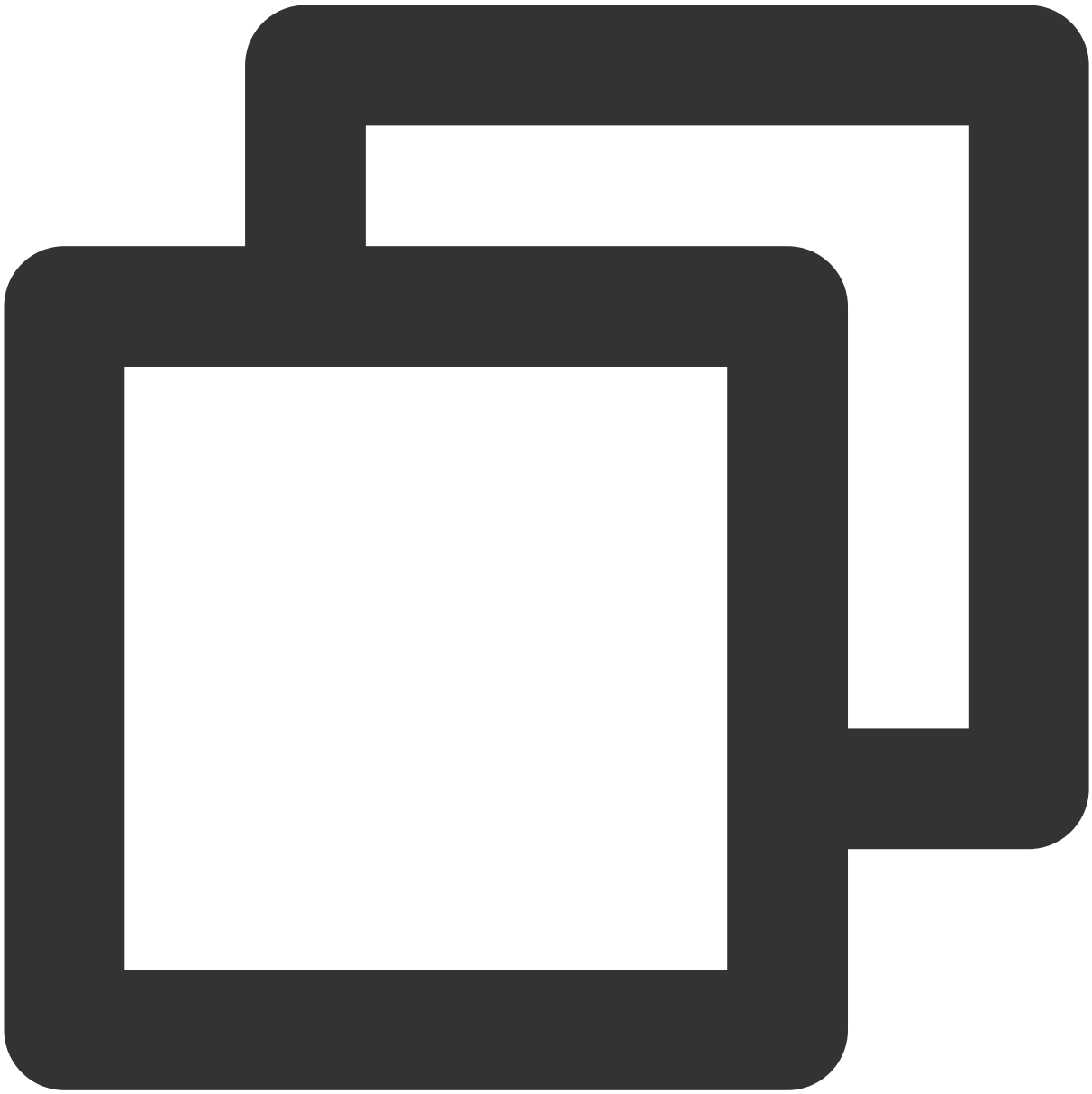
使用如下命令，安装 React Naitve IM SDK 最新版本。在命令行执行：



```
// npm
npm install react-native-tim-js

// yarn
yarn add react-native-tim-js
```

完成 SDK 初始化

调用 `initSDK`，完成 SDK 初始化。将您的 `sdkAppID` 传入。

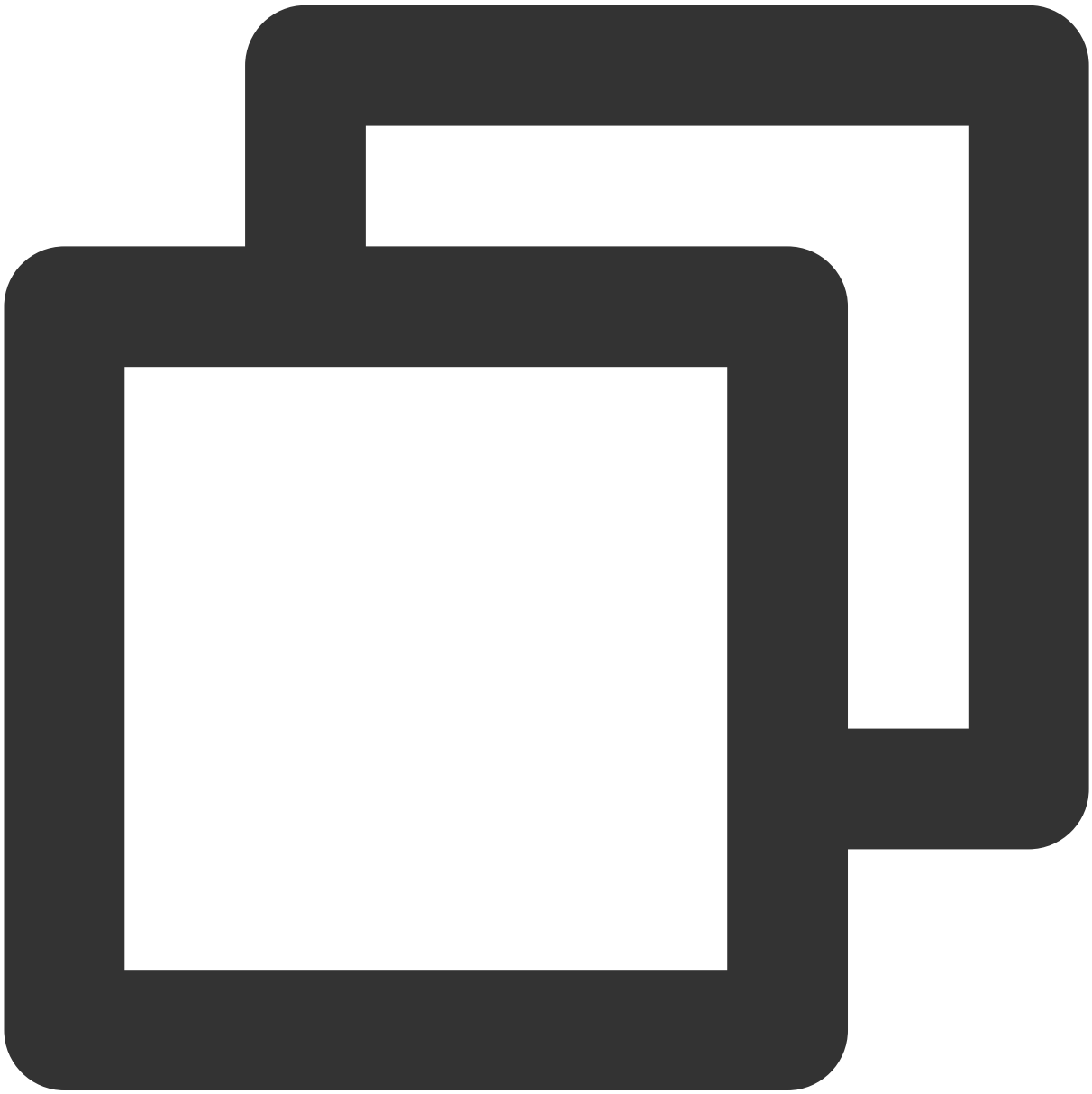


```
import { TencentImSDKPlugin, LogLevelEnum } from 'react-native-tim-js';

TencentImSDKPlugin.v2TIMManager.initSDK(
    sdkAppID: 0, // Replace 0 with the SDKAppID of your IM application when integra
    loglevel: LogLevelEnum.V2TIM_LOG_DEBUG, // Log
    listener: V2TimSDKListener(),
);
```

在本步骤，您可以针对 IM SDK 挂载一些监听，主要包括网络状态及用户信息变更等，详情可参见 该文档。

登录测试账户

1. 此时，您可以使用最开始的时候，在控制台生成的测试账户，完成登录验证。
2. 调用 `TencentImSDKPlugin.v2TIMManager.login` 方法，登录一个测试账户。当返回值 `res.code` 为 0 时，登录成功。



```
import { TencentImSDKPlugin } from 'react-native-tim-js';
 const res = await TencentImSDKPlugin.v2TIMManager.login(
 userID: userID,
 userSig: userSig,
  );
```

说明：

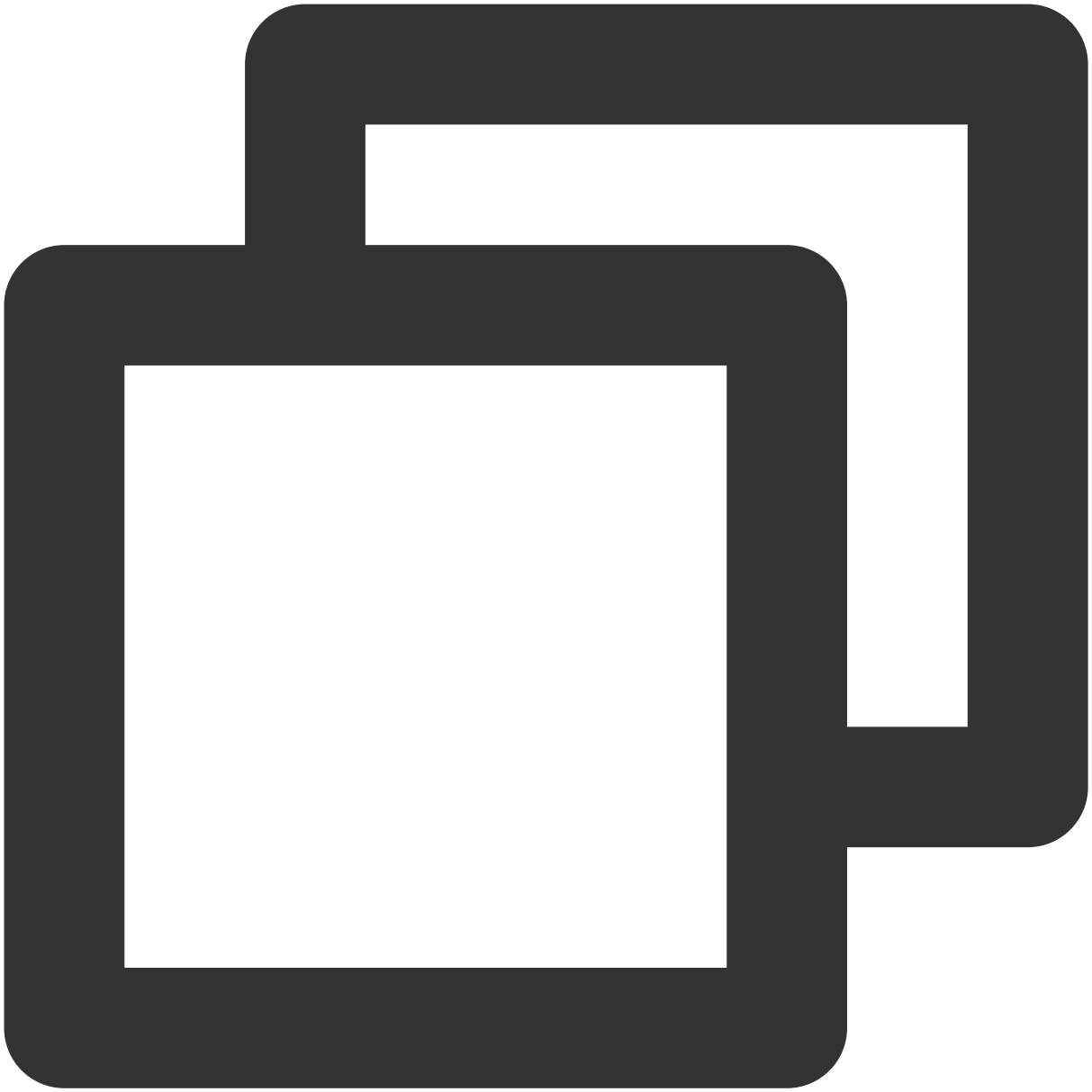
该账户仅限开发测试使用。应用上线前，正确的 `UserSig` 签发方式是将 `UserSig` 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 `UserSig` 时由您的 App 向业务服务器发起请求获取动态 `UserSig` 。更多详情请参见 服务端生成 UserSig。

发送消息

此处以发送文本消息举例，其流程为：

1. 调用 `createTextMessage(String)` 创建一个文本消息。
2. 根据其返回值，拿到消息 ID。
3. 调用 `sendMessage()` 发送该 ID 的消息。 `receiver` 可填入您此前创建的另一个测试账户 ID。发送单聊消息无需填入 `groupID` 。

代码示例：



```
import { TencentImSDKPlugin } from 'react-native-tim-js';

const createMessage =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createTextMessage("The text to create");

const id = createMessage.data!.id!; // The message creation ID

const res = await TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
```

```
        .sendMessage(
            id: id, // Pass in the message creation ID to
            receiver: "The userID of the destination user",
            groupID: "The groupID of the destination group",
            );
```

说明：

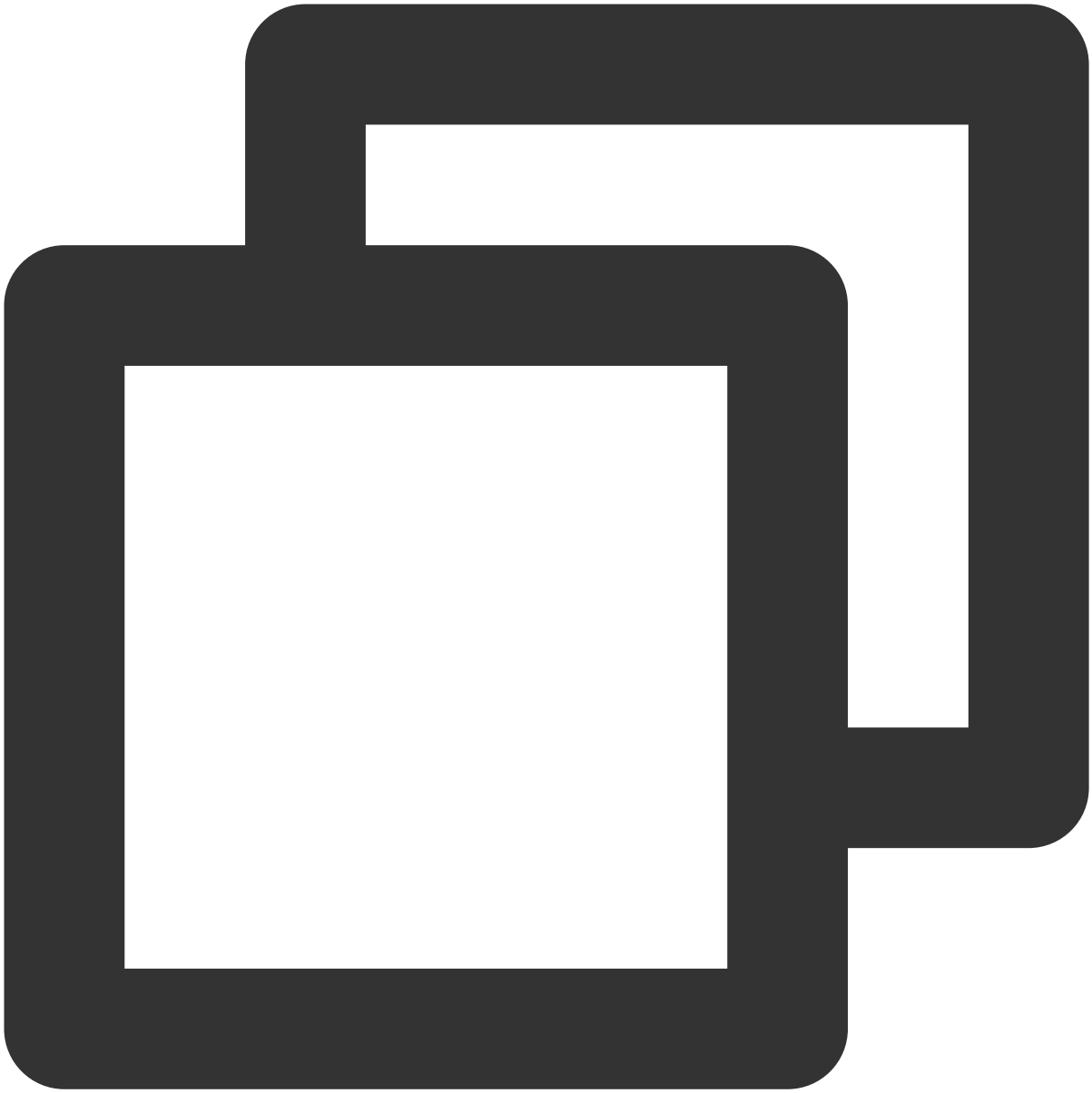
如果发送失败，可能是由于您的 sdkAppID 不支持陌生人发送消息，您可至控制台开启，用于测试。
请单击此链接，关闭好友关系链检查。

获取会话列表

在上一个步骤中，完成发送测试消息，现在可登录另一个测试账户，拉取会话列表。
获取会话列表的方式有两种：
1. 监听长连接回调，实时更新会话列表。
2. 请求 API，根据分页一次性获取会话列表。
常见应用场景为：
在启动应用程序后立即获取会话列表，然后监听长连接以实时更新会话列表的变化。

一次性请求会话列表

为了获取会话列表，需要维护 `nextSeq` ，记录当前位置。



```
import { useState } from "react";
import { TencentImSDKPlugin } from "react-native-tim-js";

const [nextSeq, setNextSeq] = useState<string>("0");

const getConversationList = async () => {
  const count = 10;
  const res = await TencentImSDKPlugin.v2TIMManager
    .getConversationManager()
    .getConversationList(count, nextSeq);
  setNextSeq(res.data?.nextSeq ?? "0");
```

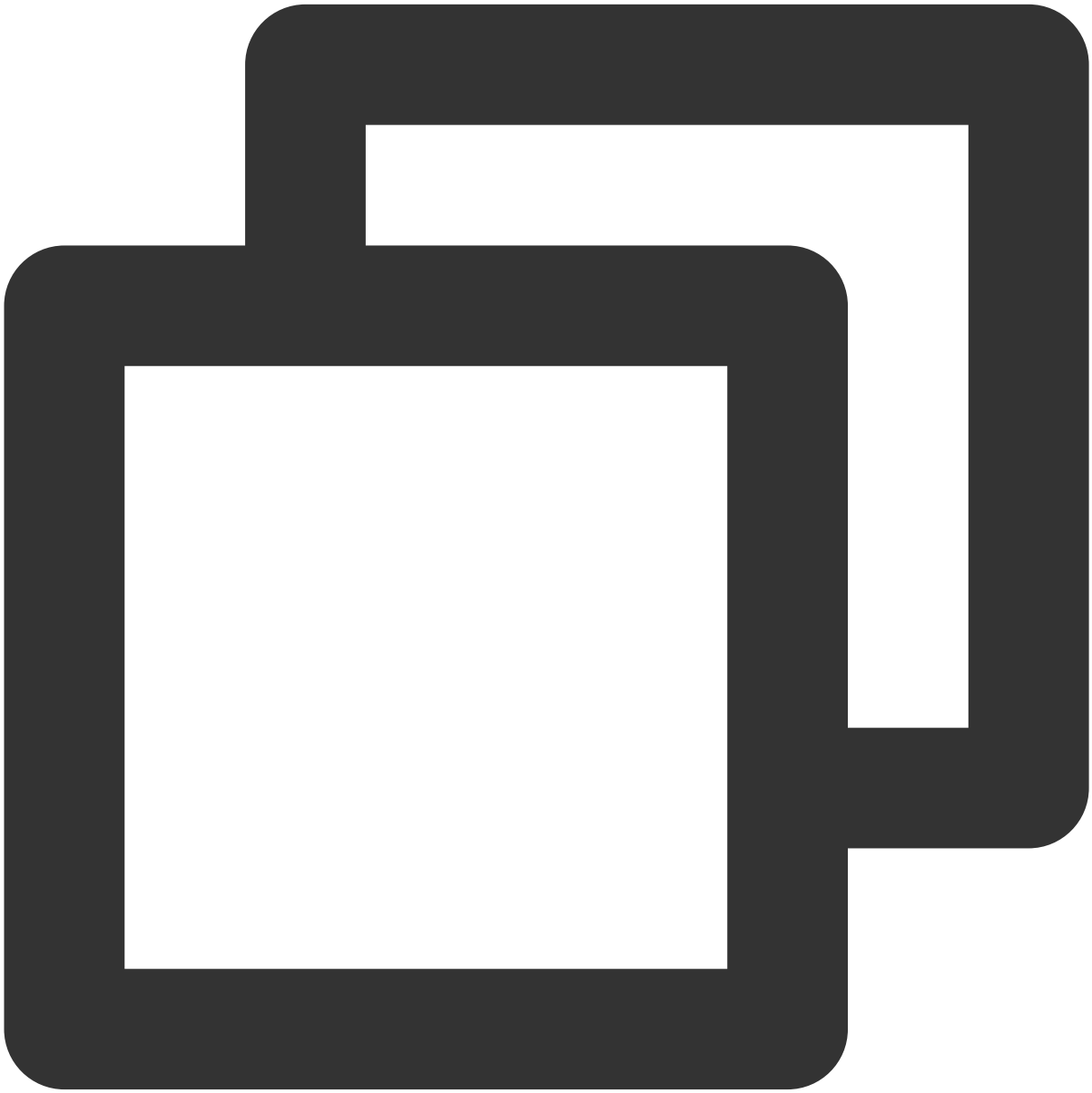
```
};
```

此时，您可以看到您在上一步中，使用另一个测试账号，发来消息的会话。

监听长链接实时获取会话列表

您在此步骤中，需要先在 SDK 上挂载监听，然后处理回调事件，更新 UI。

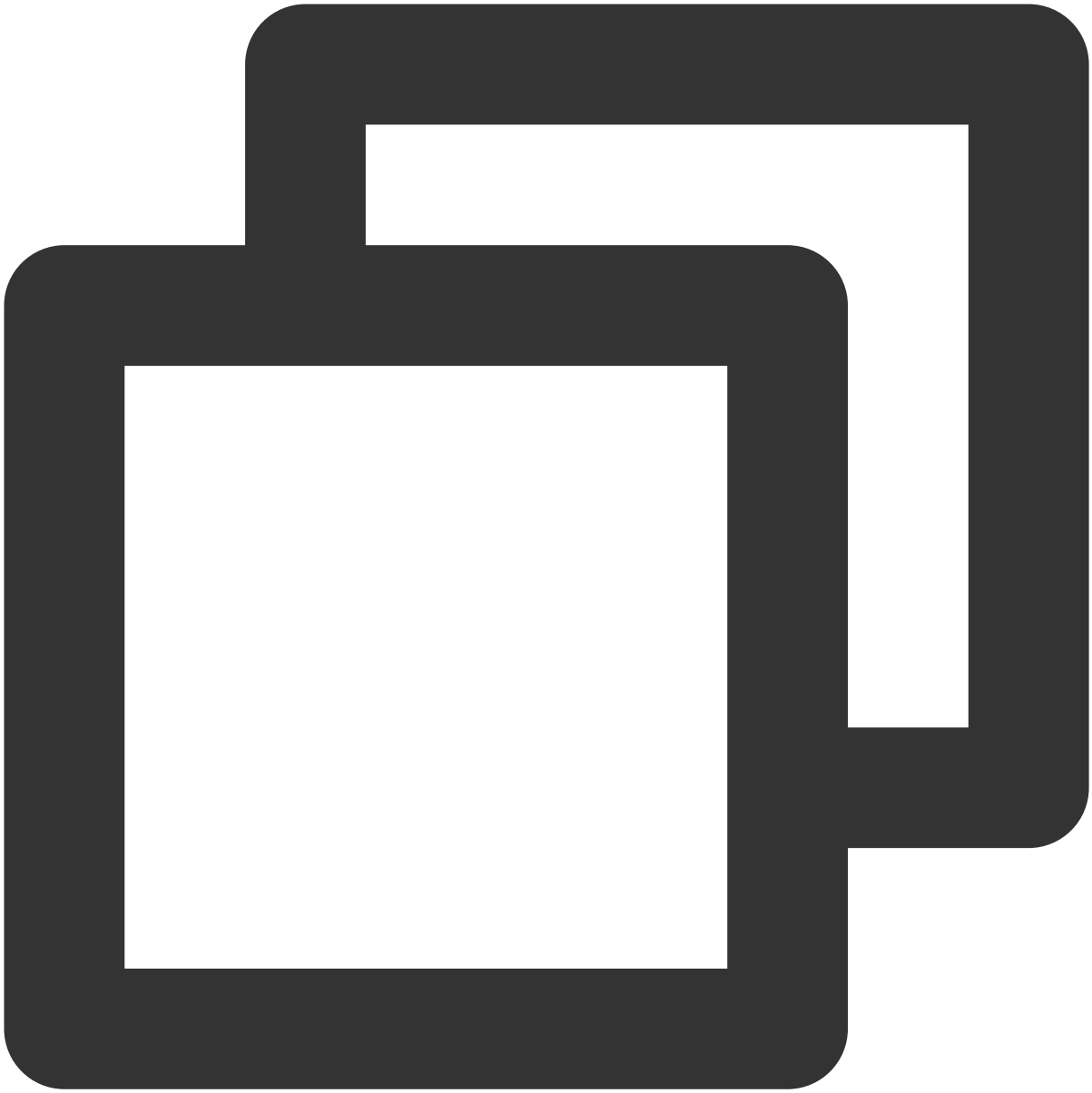
1. 挂载监听。



```
import { TencentImSDKPlugin } from "react-native-tim-js";

const addConversationListener = () => {
```

```
   TencentImSDKPlugin.v2TIMManager
  .getConversationManager()
  .addConversationListener({
    onNewConversation: (conversationList) => {
      // new conversation created callback
      _onConversationListChanged(conversationList);
    },
    onConversationChanged: (conversationList) => {
      // conversation changed callback
      _onConversationListChanged(conversationList);
    },
  });
};
```

2. 处理回调事件，将最新的会话列表展示在界面上。



```
const _onConversationListChanged = (list) => {
  // you can use conversation list to update UI
};
```

接收消息

通过腾讯云 IM React Native SDK 接收消息有两种方式：

1. 监听长连接回调，实时获取消息变化，更新渲染历史消息列表。

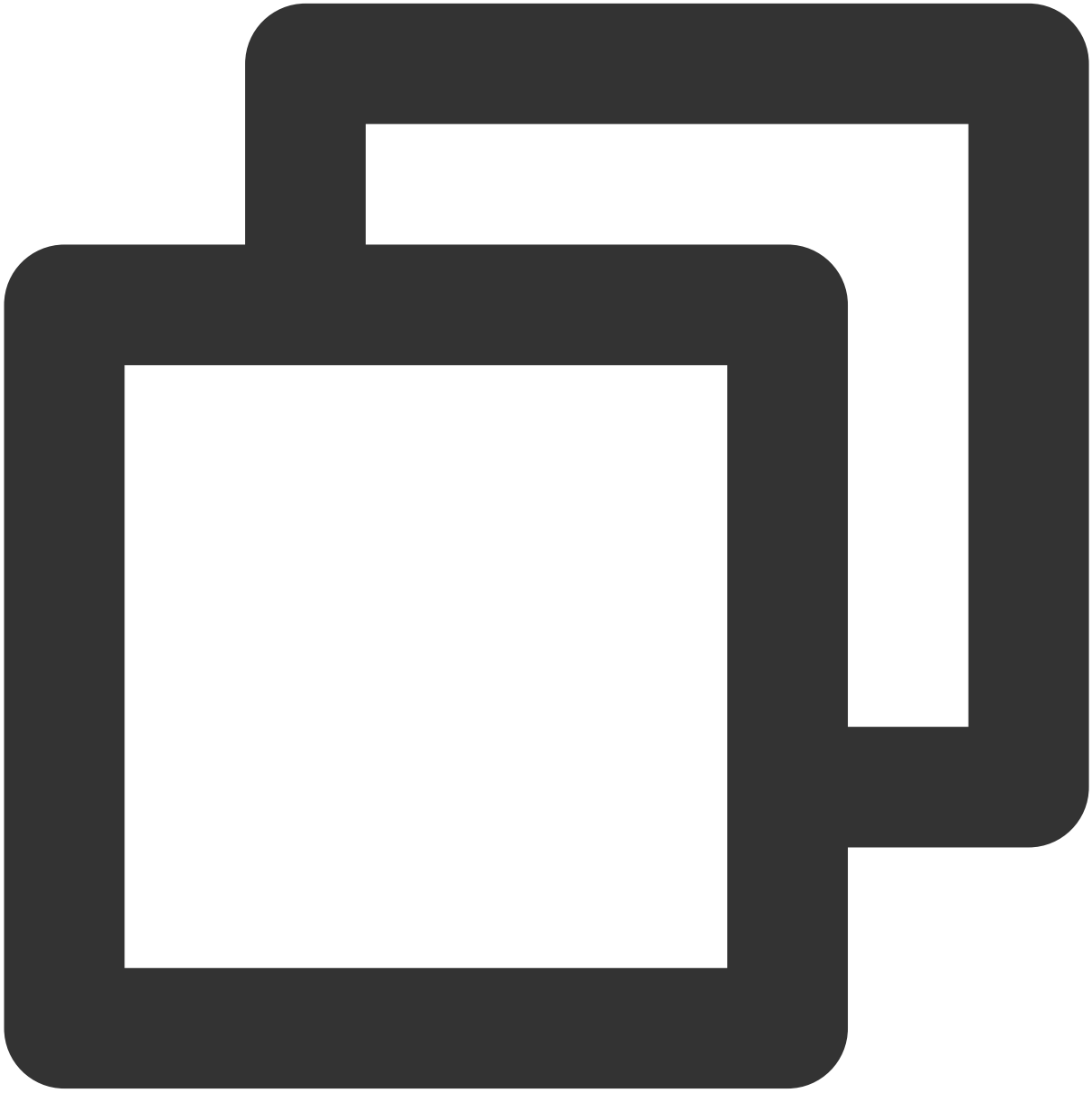
2. 请求 API，根据分页一次性获取历史消息。

常见应用场景为：

1. 界面进入新的会话后，首先一次性请求一定数量的历史消息，用于展示历史消息列表。
2. 监听长链接，实时接收新的消息，将其添加进历史消息列表中。

一次性请求历史消息列表

每页拉取的消息数量不能太大，否则会影响拉取速度。建议此处设置为 20 左右。
您应该动态记录当前页数，用于下一轮请求。
示例代码如下：



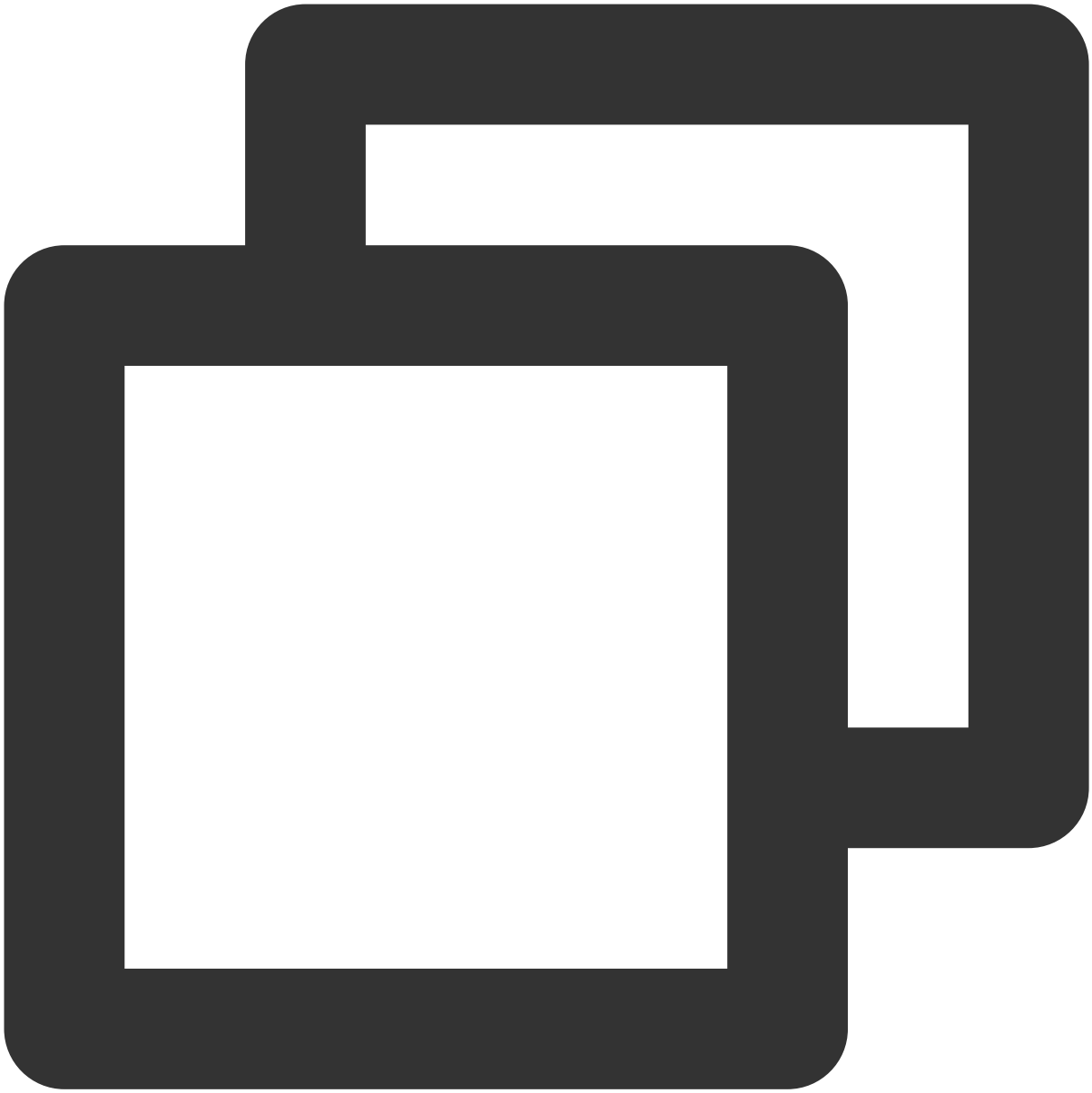
```
import { TencentImSDKPlugin } from "react-native-tim-js";
```

```
const getGroupHistoryMessageList = async () => {
  const groupID = "";
  const count = 20;
  const lastMsgID = "";
  const res = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .getGroupHistoryMessageList(groupID, count, lastMsgID);
  const msgList = res.data ?? [];
  // here you can use msgList to render your message list
};
```

监听长链接实时获取新消息

历史消息列表初始化后，新消息来自长链接 `V2TimAdvancedMsgListener.onRecvNewMessage` 。`onRecvNewMessage` 回调被触发后，您可以按需将新消息添加进历史消息列表中。

绑定监听器示例代码如下：



```
import { TencentImSDKPlugin } from "react-native-tim-js";

const adVancesMsgListener = {
  onRecvNewMessage: (newMsg) => {
    _onReceiveNewMsg(newMsg);
    /// ... other listeners related to message
  },
};

const addAdvancedMsgListener = () => {
  TencentImSDKPlugin.v2TIMManager
```

```
      .getMessageManager()
      .addAdvancedMsgListener(adVancesMsgListener);
  };
```

此时，您已基本完成 IM 模块开发，可以发送接收消息，也可以进入不同的会话。
您可以继续完成群组，用户资料，关系链，离线推送，本地搜索等相关功能开发，详情可参见 SDK API 文档。

常见问题

运行 demo 时出现 `Undefined symbols for architecture x86_64 [duplicate]` 如何解决？

请参见 文档。

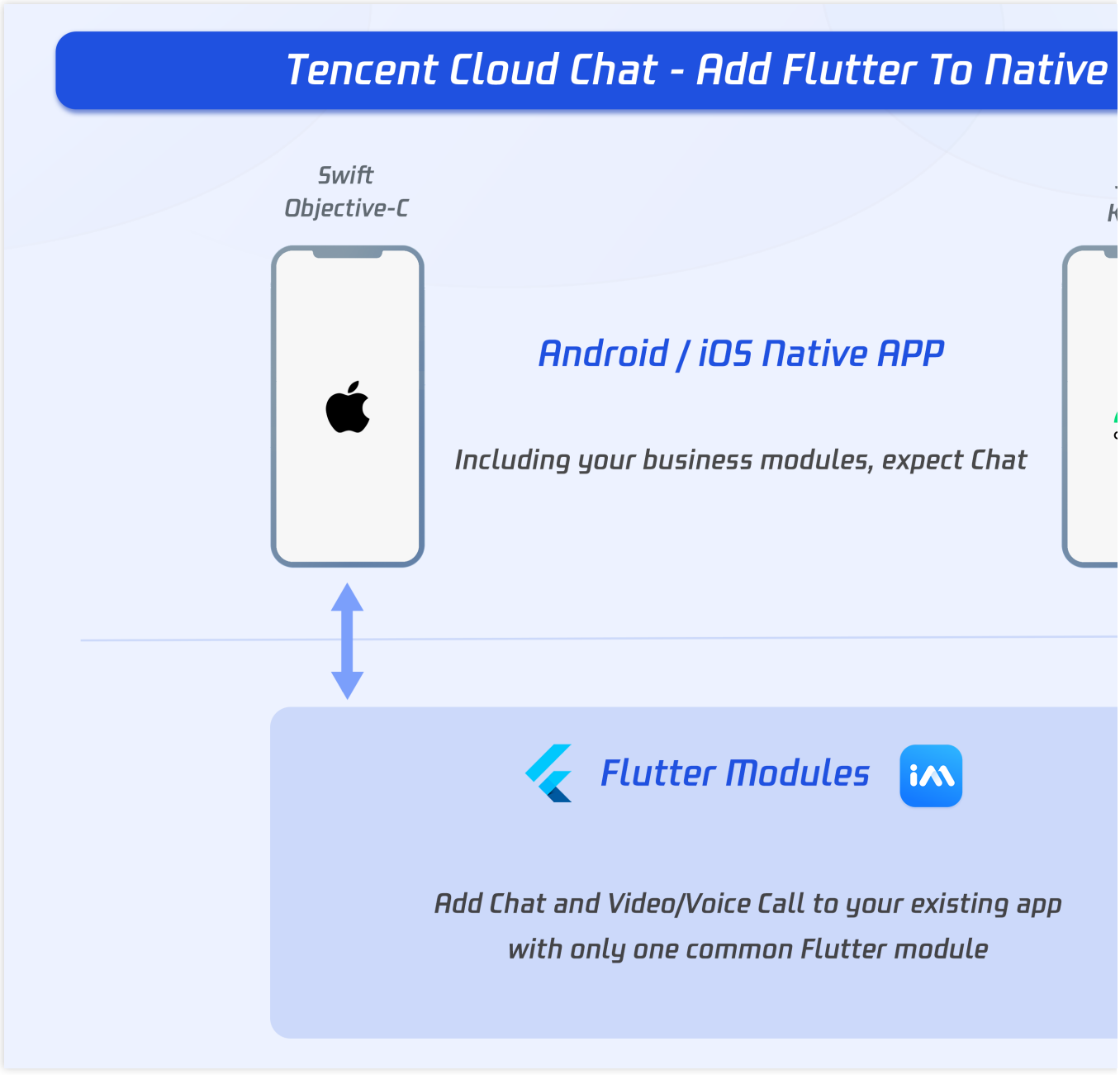
运行 demo 时出现 `Failed to resolve: react-native-0.71.0-rc.0-debug` 如何解决？

请参见 文档。

用 Flutter 快速集成至您现有原生应用

最近更新时间：2024-05-13 16:45:14

通过阅读本文，你可以了解在您现有的 Android / iOS 原生开发项目中，集成腾讯云 IM Flutter 的方法。
有的时候，使用 Flutter 重写您现有的应用程序是不现实的。如果您想在现有 App 中，使用腾讯云 IM 的能力，推荐采用混合开发方案，即将 Flutter 模块，嵌入您的原生开发 App 项目中。
可在很大程度上，降低您的工作量，快速在双端原生 App 中，植入 IM 通信能力。



环境要求

环境	版本
Flutter	SDK 最低要求 Flutter 2.2.0版本，TUIKit 集成组件库最低要求 Flutter 2.10.0 版本。
Android	Android Studio 3.5及以上版本，App 要求 Android 4.1及以上版本设备。
iOS	Xcode 11.0及以上版本，请确保您的项目已设置有效的开发者签名。
腾讯云IM SDK	tencent_im_sdk_plugin 5.0 及以上版本， tim_ui_kit 0.2 及以上版本。

说明：
对于以上的 Demo 项目，源代码可在我们的 GitHub 仓库 中找到，欢迎查阅。

前置知识点

开始之前，您需要了解腾讯云 IM Flutter SDK 及 TUIKit 的用法；及 Flutter-原生混合开发原理。

腾讯云IM

总体入门

在开始前，您首先需要了解腾讯云 IM Flutter 的SDK构成及使用方式。
主要包括两个SDK：无UI版本 及含UI组件库。本文将以 含UI组件库（TUIKit） 为例，介绍混合开发方案。
关于腾讯云IM Flutter详细用法，可从我们的 快速入门文档 看起。

两个模块

腾讯云 IM 主要有两个部分，包括 Chat 聊天模块 和 Call 通话模块。
Chat 聊天模块主要包括消息收发、会话管理、用户关系管理等。
Call 通话模块主要包括音视频通话，包括一对一通话和群组多人通话。

Flutter 混合开发

核心原理是，将 module 形式的Flutter项目，打包成 Native 端的可执行程序，嵌入 Native 项目中。因 Flutter module 可以通用，因此仅需编写一次 Flutter module，即可嵌入 Android/iOS App 中。
当您现有应用需要展示腾讯云IM相关页面时，可加载对应用于承载 Flutter 的 Activity（Android）或 ViewController（iOS）。
当需要两端通信时，如传递当前用户信息，传递音视频通话数据，触发离线推送数据，可采用Method Channel方式进行。触发另一端的方法使用 `invokeMethod` ，监听另一端发来的方法调用使用预挂载的Method Channel监听器。

将 Flutter 模块添加至 Android 项目中

详细学习

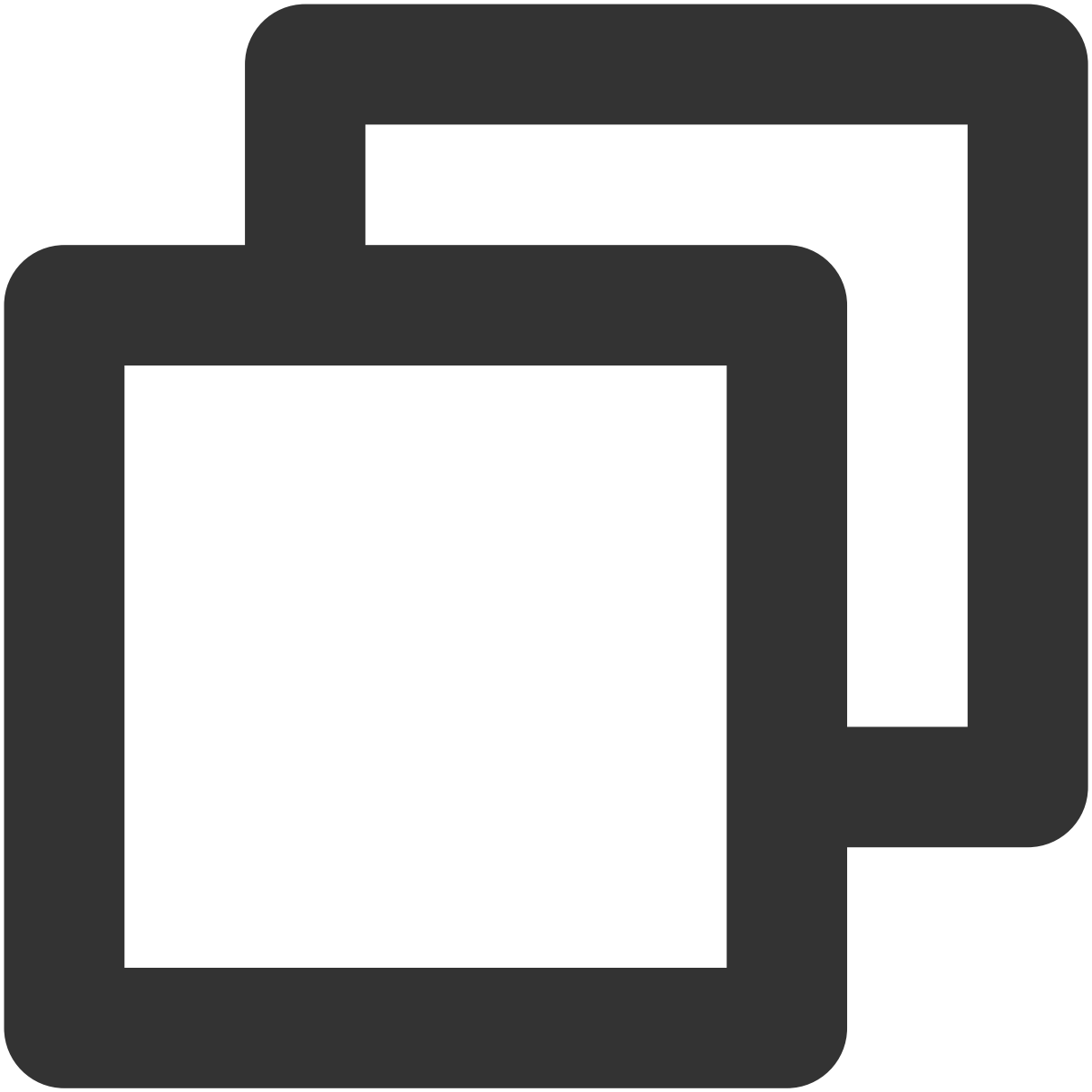
将 Flutter module 添加为 Gradle 中现有应用程序的依赖项。有两种方式可以实现这一点。

Android 方式一：依赖 Android Archive (AAR)

AAR 机制创建通用的 Android AAR 作为打包 Flutter module 的中介。如果您经常构建，它会增加一个构建步骤。该选项将 Flutter 库打包为由 AAR 和 POMS 构件组成的通用本地 Maven 存储库。此选项允许您的团队在不安装 Flutter SDK 的情况下构建主机应用程序。然后，您可以从本地或远程存储库中分发构件。
因此，建议在线上生产环境，使用本方案。

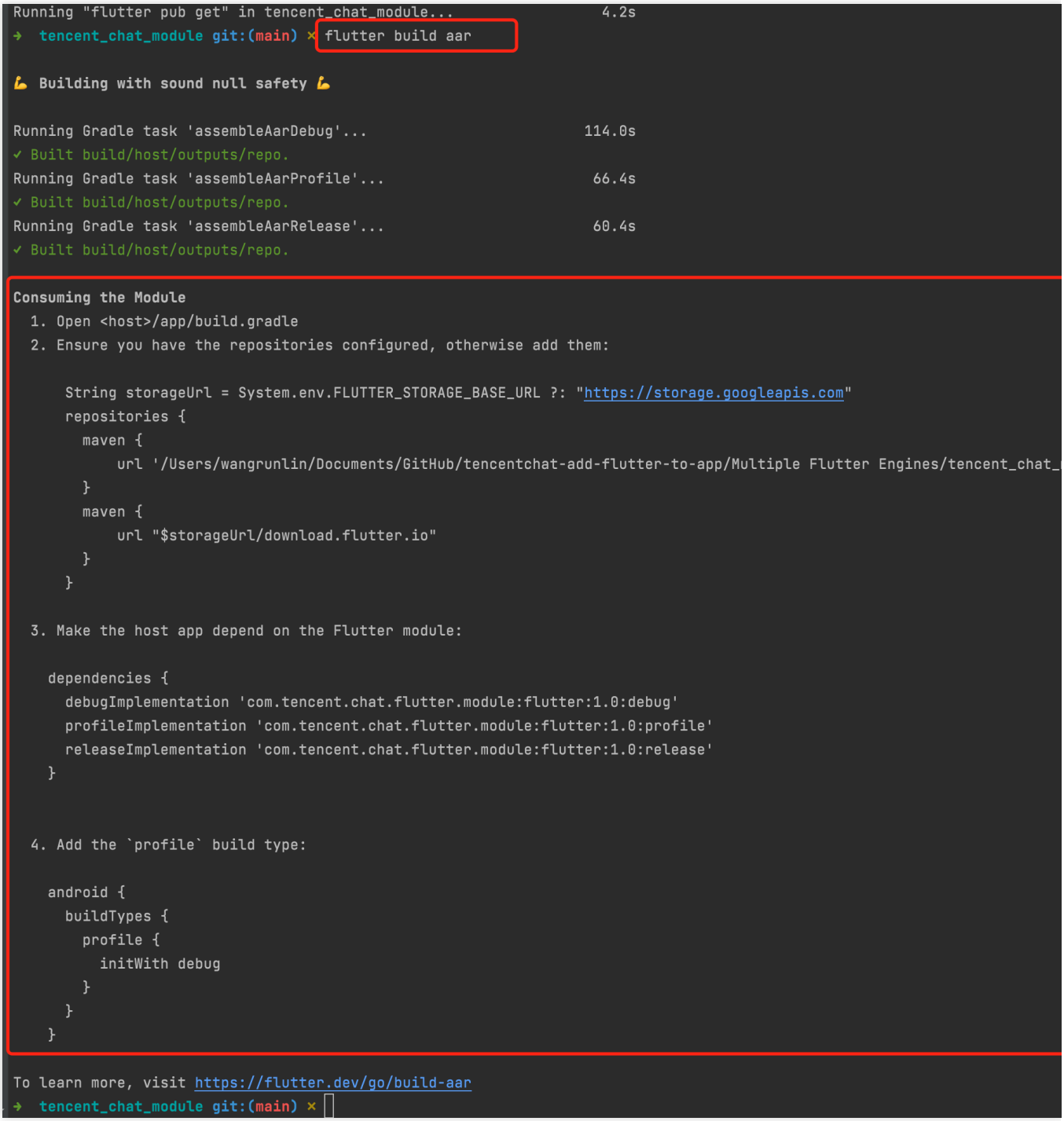
具体步骤:

在您的 Flutter module 中，运行：



```
flutter build aar
```

然后，按照屏幕上的说明进行集成。



您的应用程序现在将 Flutter 模块作为依赖项包括在内。

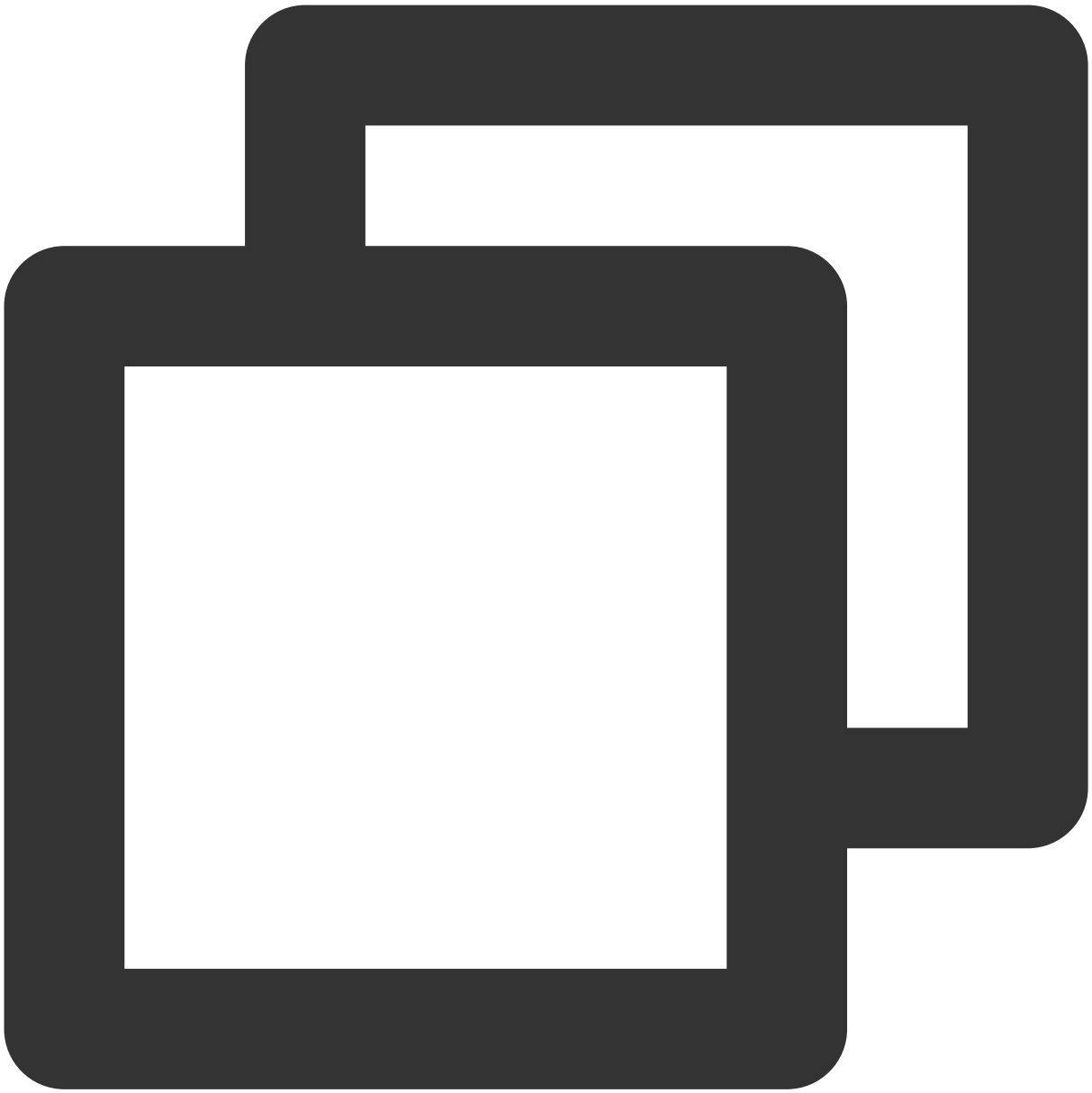
Android 方式二：依赖 Flutter module 源代码

源代码子项目机制是一个方便的一键构建过程，但需要 Flutter SDK。这是 Android Studio IDE 插件使用的机制。此方式可为您的 Android 项目和 Flutter 项目实现一步构建。当您同时处理两个部分并快速迭代时，此选项很方便，但您的团队必须安装Flutter SDK才能构建应用程序。

因此，建议在开发测试环境，使用本方案。

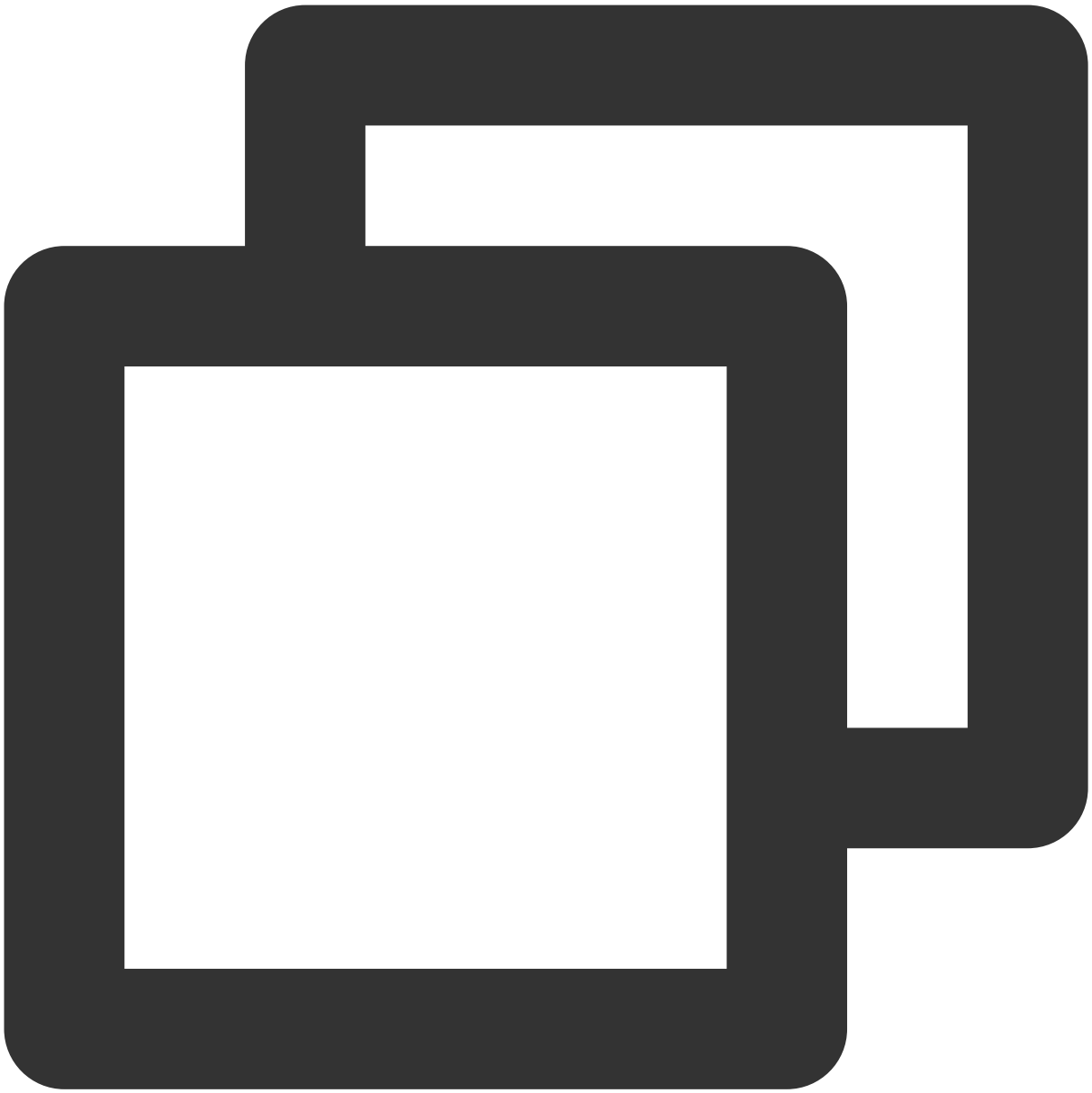
具体步骤:

将 Flutter module 作为一个子项目，添加至宿主 App 的 `settings.gradle` 中：



```
// Include the host app project.
include ':app'                                        // assumed existing content

// Add the following lines to your project
setBinding(new Binding([gradle: this]))
evaluate(new File(
  settingsDir.parentFile,
  'tencent_chat_module/.android/include_flutter.groovy'
))
```

在您应用中的 `app/build.gradle => dependencies` 中引入对 Flutter module 的 `implementation`：



```
dependencies {
  implementation project(':flutter')
}
```

您的应用程序现在将 Flutter模块作为依赖项包括在内。

将 Flutter 模块添加至 iOS 项目中

详细学习

有两种方法可以在现有应用程序中嵌入Flutter。

iOS方式一：嵌入 CocoaPods 和 Flutter SDK 集成

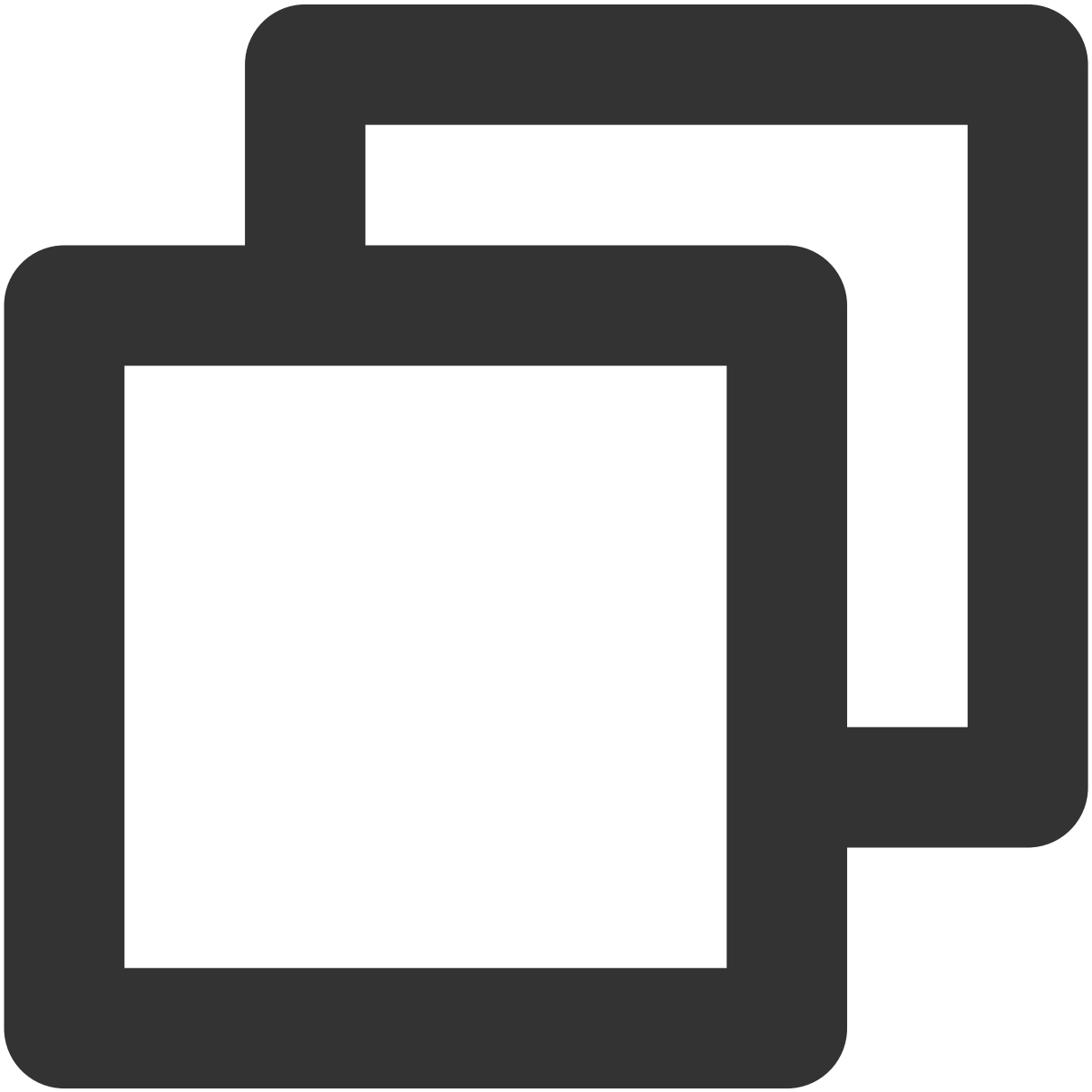
使用 CocoaPods 依赖项管理器并安装 Flutter SDK。这种方法要求每个从事项目工作的开发人员都有一个本地安装的 Flutter SDK 版本。

只需在 Xcode 中构建您的应用程序，即可自动运行脚本来嵌入您的 DART 和插件代码。这允许快速迭代最新版本的颤振模块，而无需在 Xcode 之外运行其他命令。

因此，建议在开发测试环境，使用本方案。

具体步骤:

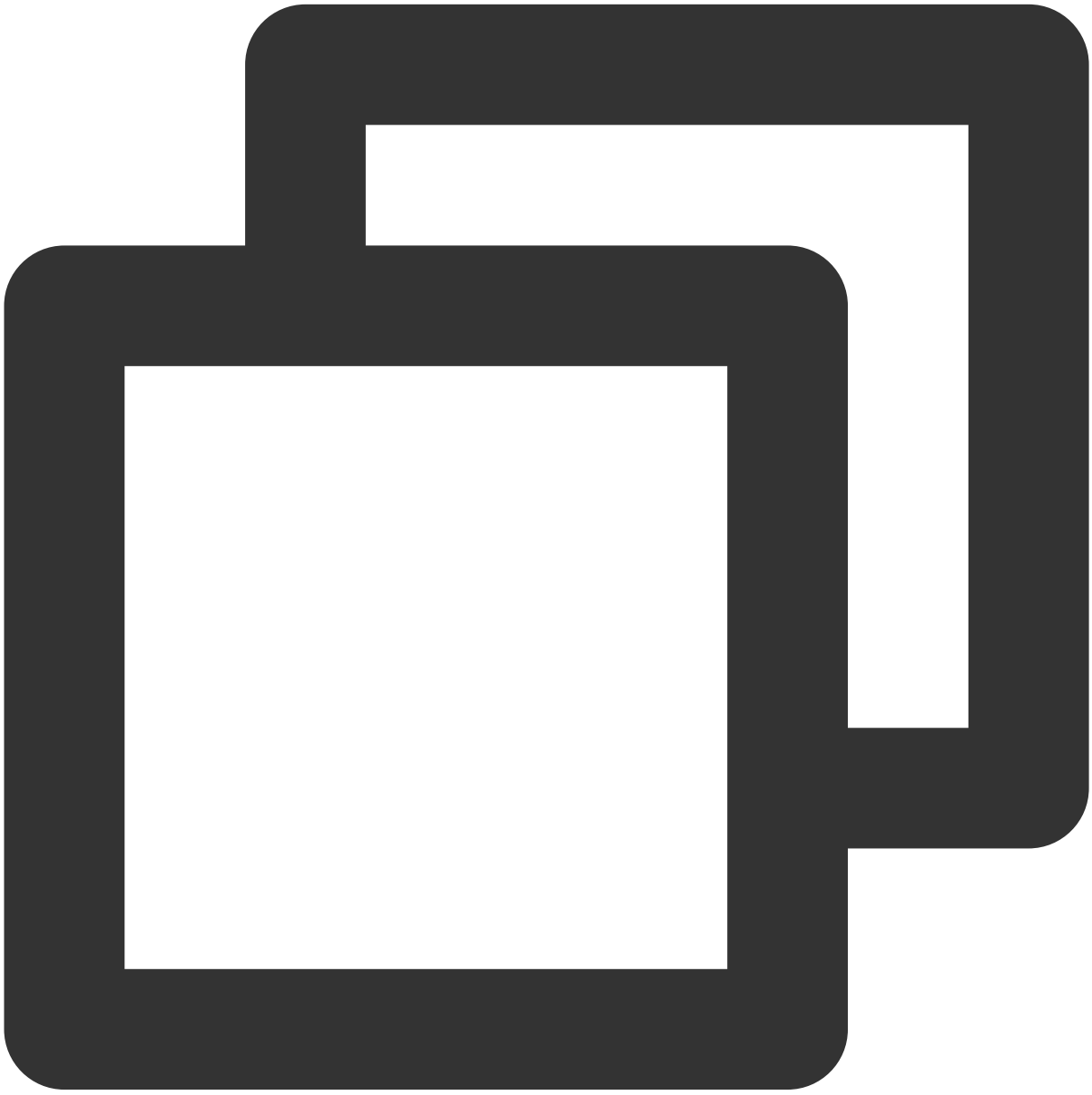
将以下代码添加到 Podfile 中:



```
// 上一步构建的 Flutter Module 的路径
```

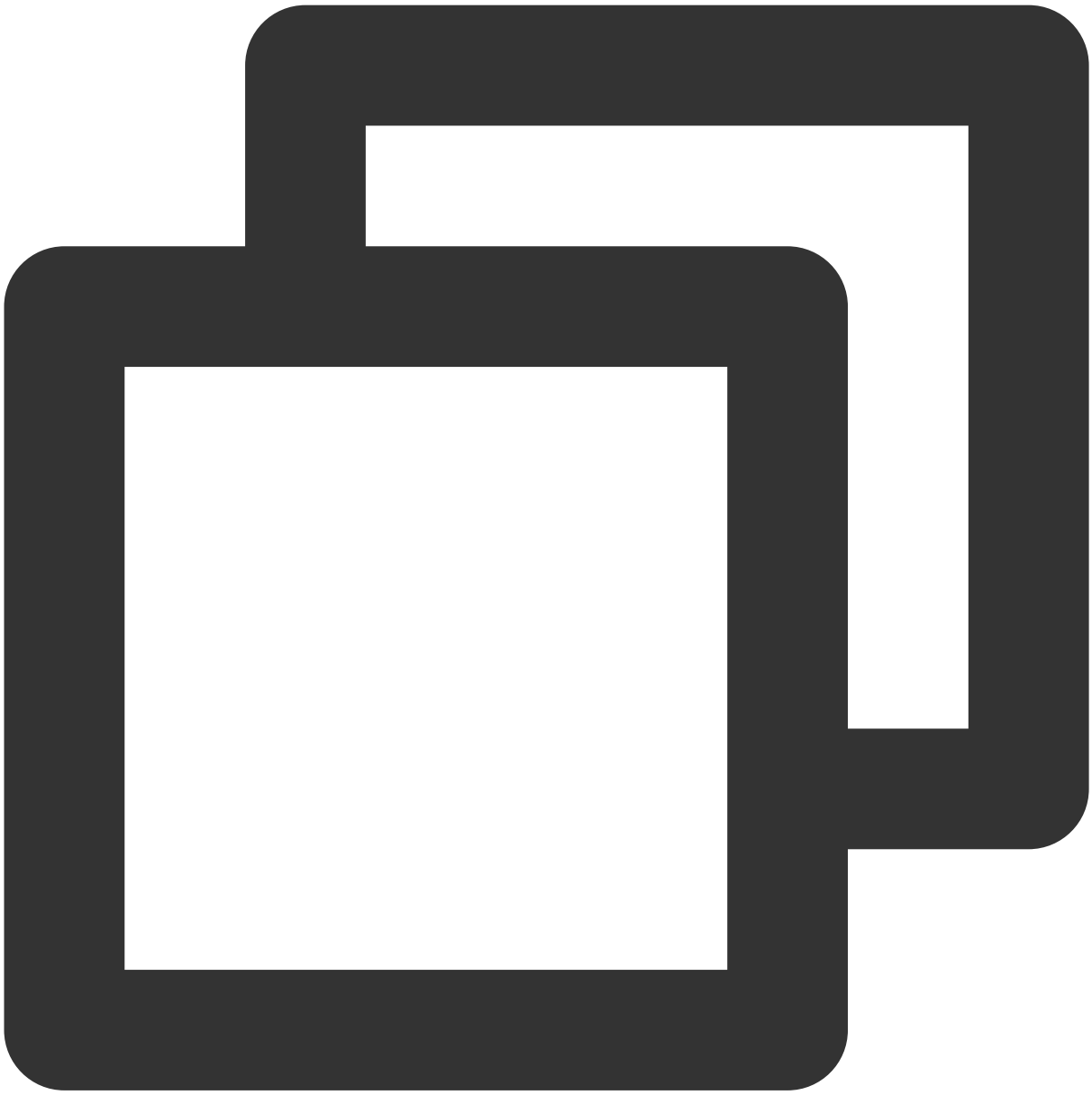
```
flutter_chat_application_path = '../tencent_chat_module'

load File.join(flutter_chat_application_path, '.ios', 'Flutter', 'podhelper.rb')
```

对于每个需要嵌入 Flutter 的 Podfile target，调用 `install_all_flutter_pods(flutter_chat_application_path)` .



```
target 'MyApp' do
  install_all_flutter_pods(flutter_chat_application_path)
end
```

在 Podfile 的 `post_install` 块中，调用 `flutter_post_install(installer)` ，并完成 腾讯云IM TUIKit 所需的权限声明，包括麦克风权限/相机权限/相册权限。



```
post_install do |installer|
  flutter_post_install(installer) if defined?(flutter_post_install)
  installer.pods_project.targets.each do |target|
      flutter_additional_ios_build_settings(target)
      target.build_configurations.each do |config|
            config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
              '$(inherited)',
              'PERMISSION_MICROPHONE=1',
```

```
                'PERMISSION_CAMERA=1',
                'PERMISSION_PHOTOS=1',
              ]
            end
      end
  end
```

执行 `pod install` 。

说明：

在 `tencent_chat_module/pubspec.yaml` 中更改 Flutter 插件依赖时，请在 Flutter Module 目录中运行 `flutter pub get` 以刷新 `podhelper.rb` 脚本读取的插件列表。然后，从您 iOS 应用程序的根目录，再次执行 `pod install` 。

对于 Apple Silicon 芯片 arm64 架构的 Mac 电脑，可能需要执行 `arch -x86_64 pod install --repo-update` 。

`podhelper.rb` 脚本将您的插件 / `Flutter.framework` / `App.framework` 植入您的项目中。

iOS 方式二：在 Xcode 中嵌入 frameworks

为 Flutter 引擎、已编译的 DART 代码和所有 Flutter 插件创建框架。手动嵌入框架，并在 Xcode 中更新现有应用程序的构建设置。

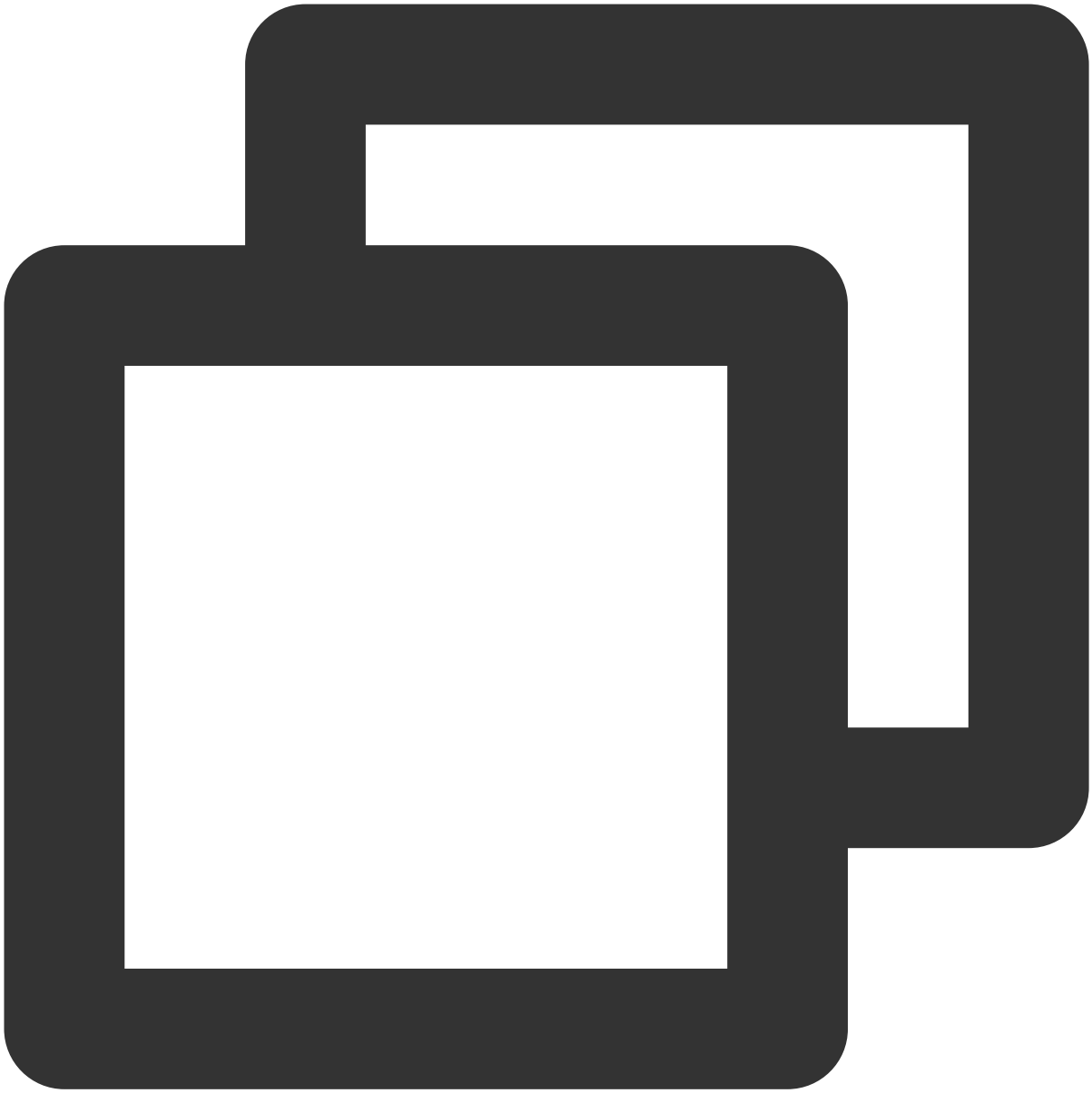
通过手动编辑现有的 Xcode 项目，您可以生成必要的 framework 并将它们嵌入到应用程序中。如果您的团队成员无法在本地安装 Flutter SDK 和CocoaPods，或者如果您不想在现有应用程序中使用 CocoaPods 作为依赖项管理器，则可以这样做。每次你在你的颤动模块中修改代码时，你都必须运行 `flutter build ios-framework` .

因此，建议在线上环境，使用本方案。

具体步骤:

在您的 Flutter module 中，运行如下代码。

下面的示例，假设您想要将 framework 生成到 `some/path/MyApp/Flutter/` .



```
flutter build ios-framework --output=some/path/MyApp/Flutter/
```

在 Xcode 中将生成的 frameworks 集成到你的既有应用中。例如，你可以在 `some/path/MyApp/Flutter/Release/` 目录拖拽 frameworks 到你的应用 target 编译设置的 General > Frameworks, Libraries, and Embedded Content 下，然后在 Embed 下拉列表中选择 “Embed & Sign”。

混合开发选型

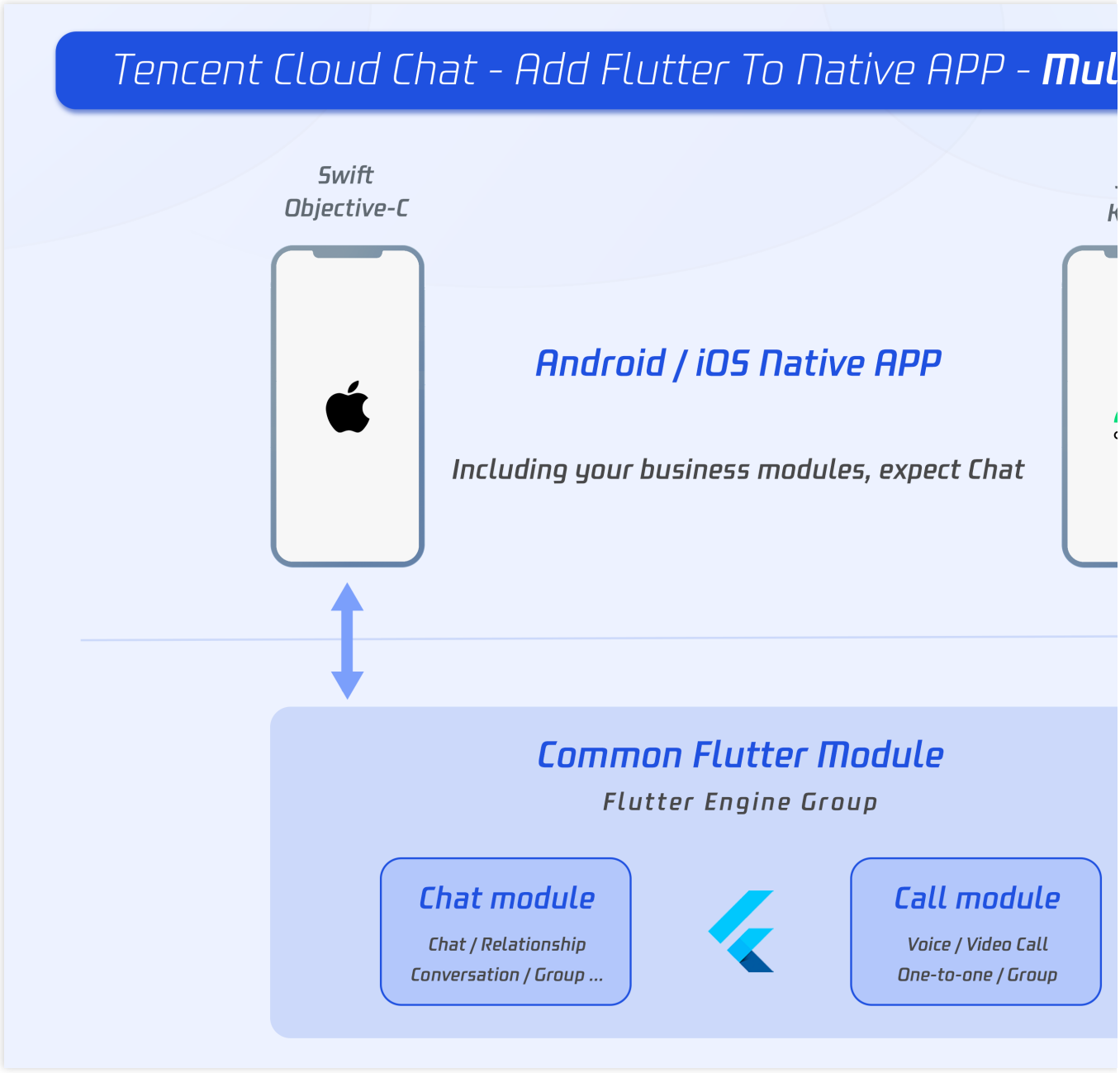
我们推荐您使用 Flutter Module 方式进行混合开发集成。
在 Native 原生项目中，构建 Flutter 引擎，来承载 Flutter 中的 Chat 及 Call 模块。有关两个模块的介绍，请看此处。
对于 Flutter 引擎的创建管理，目前两种方式：单 Flutter 引擎及多 Flutter 引擎。

引擎模式	介绍	优点	缺点	Demo 源码下载
Flutter 单引擎	Chat 模块和 Call 模块在同一个Flutter 引擎中承载。	方便，所有 Flutter 代码统一维护。	由于 Call 插件，在有电话呼入时，需要自动展示来电页面。如果在同一个引擎中，需要强制跳转至 Flutter 所在页面，体验较差。	点击下载
Flutter 多引擎	Chat 模块和 Call 模块分别承载于不同的 Flutter 引擎中，使用 Flutter 引擎组来统一管理这两个引擎。	Call 插件独立存在于一个 Flutter 引擎中，独立页面控制，来电时，直接将该页面弹窗即可，不影响用户当前所在页面，体验较好。	通话模块无法最小化成浮窗形式。	点击下载

此外，我们还提供，将腾讯云 IM Native SDK 与 Flutter SDK 结合使用的方案，适用场景和步骤介绍可查看这里。
Demo源码下载。

方案一：Flutter 多引擎方案【推荐】

本方案中，Chat 和 Call 模块分别独立于不同的 Flutter 引擎。
使用多个 Flutter 引擎的优点是，每个实例都是独立的，并维护其自己的内部导航堆栈、UI 和应用程序状态。这简化了整个应用程序代码的状态保持责任，并提高了模块化能力。



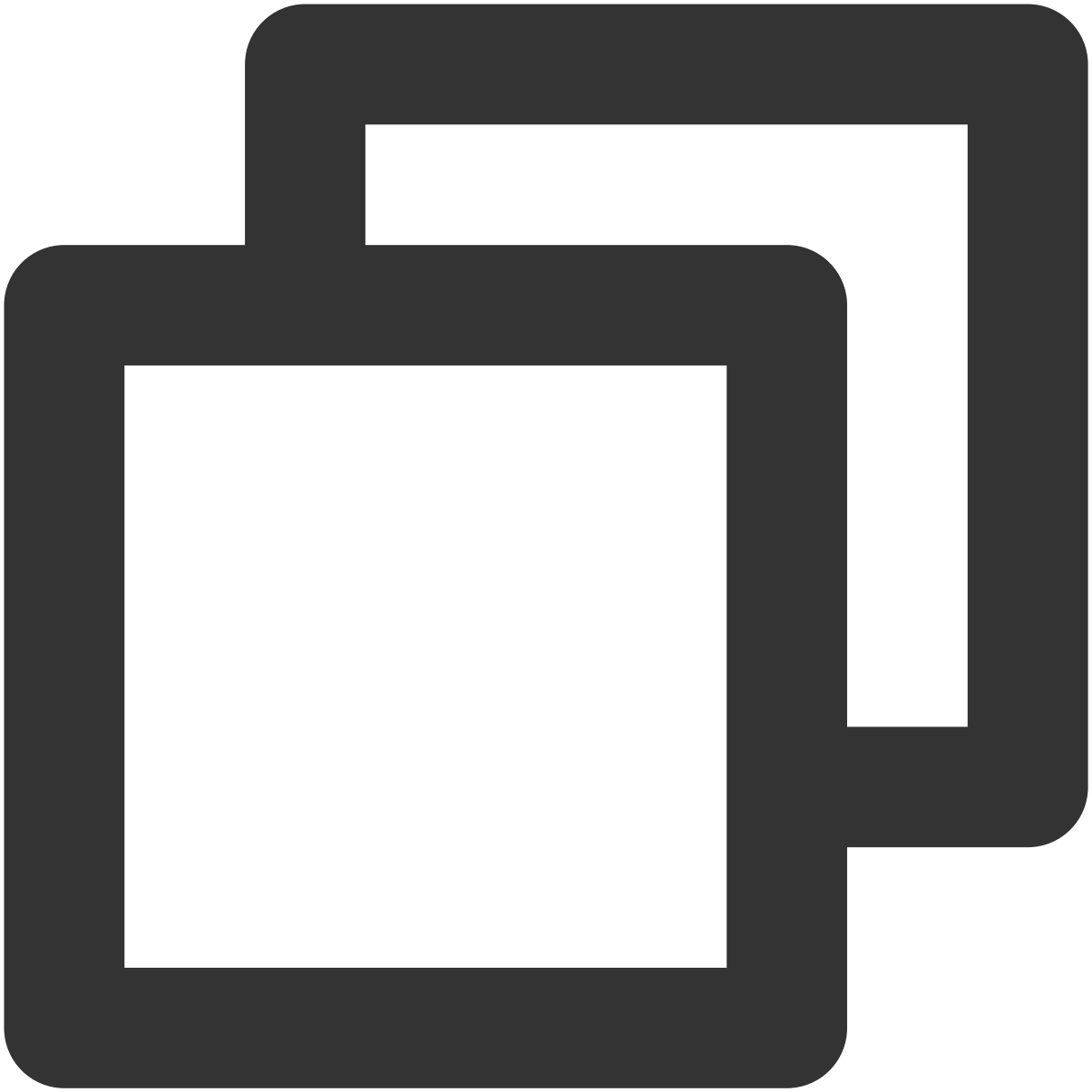
在Android 和iOS上添加多个Flutter引擎，主要基于一个FlutterEngineGroup类(Android API、iOS API)来构造并管理多个FlutterEngine（Flutter引擎）。

在我们的项目中，我们基于一个统一的FlutterEngineGroup，来管理两个FlutterEngine（Flutter引擎），分别用于承载 Chat 和 Calling 模块。

Flutter Module 开发

要将Flutter嵌入到现有应用程序中，请首先创建一个Flutter模块。

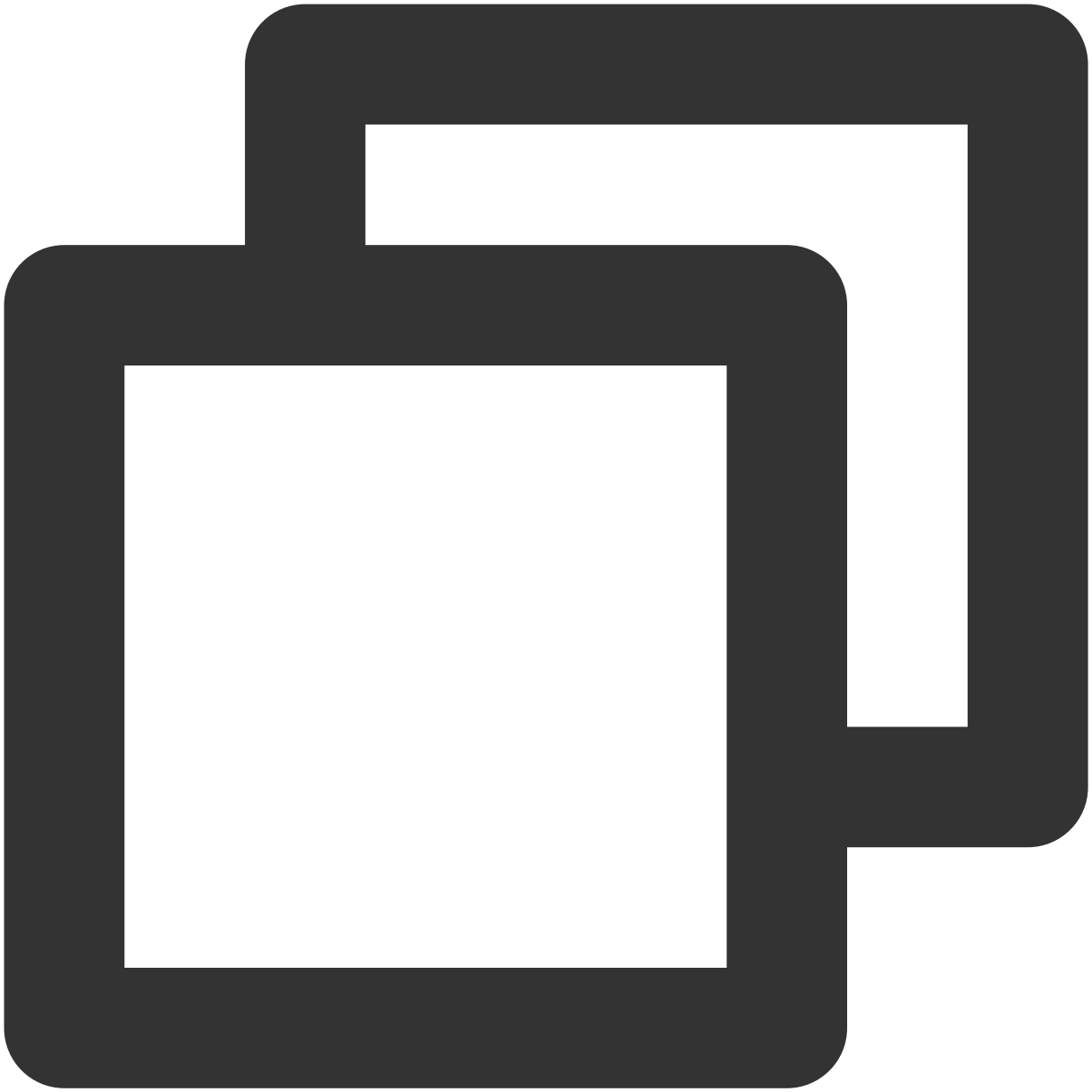
在您项目的根目录外层运行以下内容：



```
cd some/path/
flutter create --template module tencent_chat_module
```

这会在 some/path/tencent_chat_module/ 创建一个 Flutter 模块项目。 在该目录中，您可以运行与在任何其他 Flutter 项目中相同的 Flutter 命令，例如 `flutter run --debug` 或 `flutter build ios` 。 您还可以使用 Flutter 和 Dart 插件在 Android Studio, IntelliJ 或 VS Code 中运行该模块。 该项目在嵌入到现有应用程序之前包含模块的单视图示例版本，这对于测试代码的仅 Flutter 部分很有用。

`tencent_chat_module` 模块目录结构类似于普通的 Flutter 应用程序：



```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

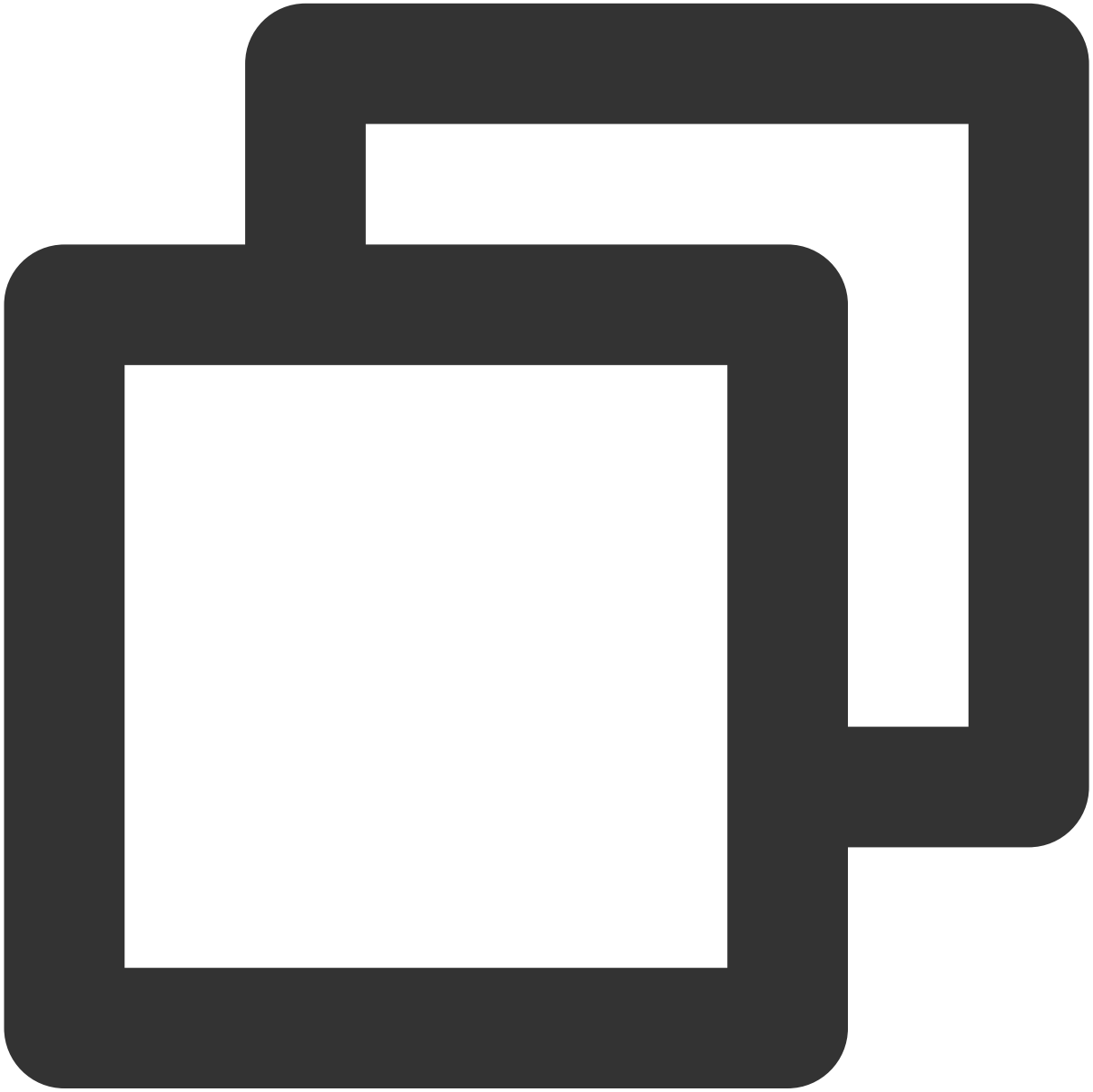
现在，我们可以在 `lib/` 中，编写代码了。

梳理Flutter lib 目录

说明：

以下代码结构，仅供参考，您可根据需要灵活组织，以引入腾讯云IM Flutter。

在 `lib/` 我们创建三个目录， `call` ， `chat` ， `common` 。分别用于放置通话引擎，IM引擎，及通用model类。

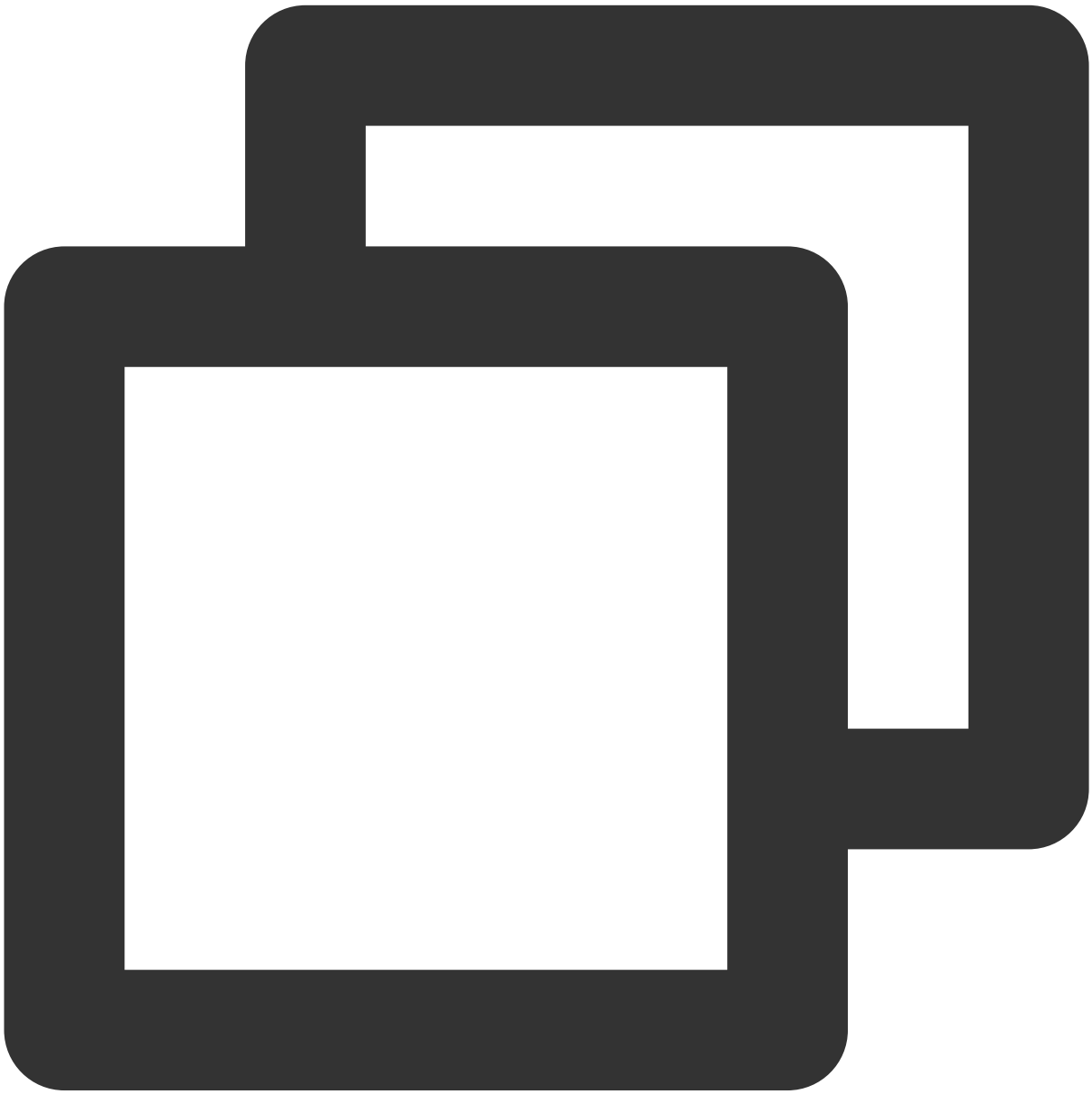


```
tencent_chat_module/
├── lib/
│   └── call/
│   └── chat/
```

```
|   └── common/
```

通用model类模块

新建 `common/common_model.dart` 文件，如下所示，新建两个class，用于定义Flutter与原生应用通信规范。



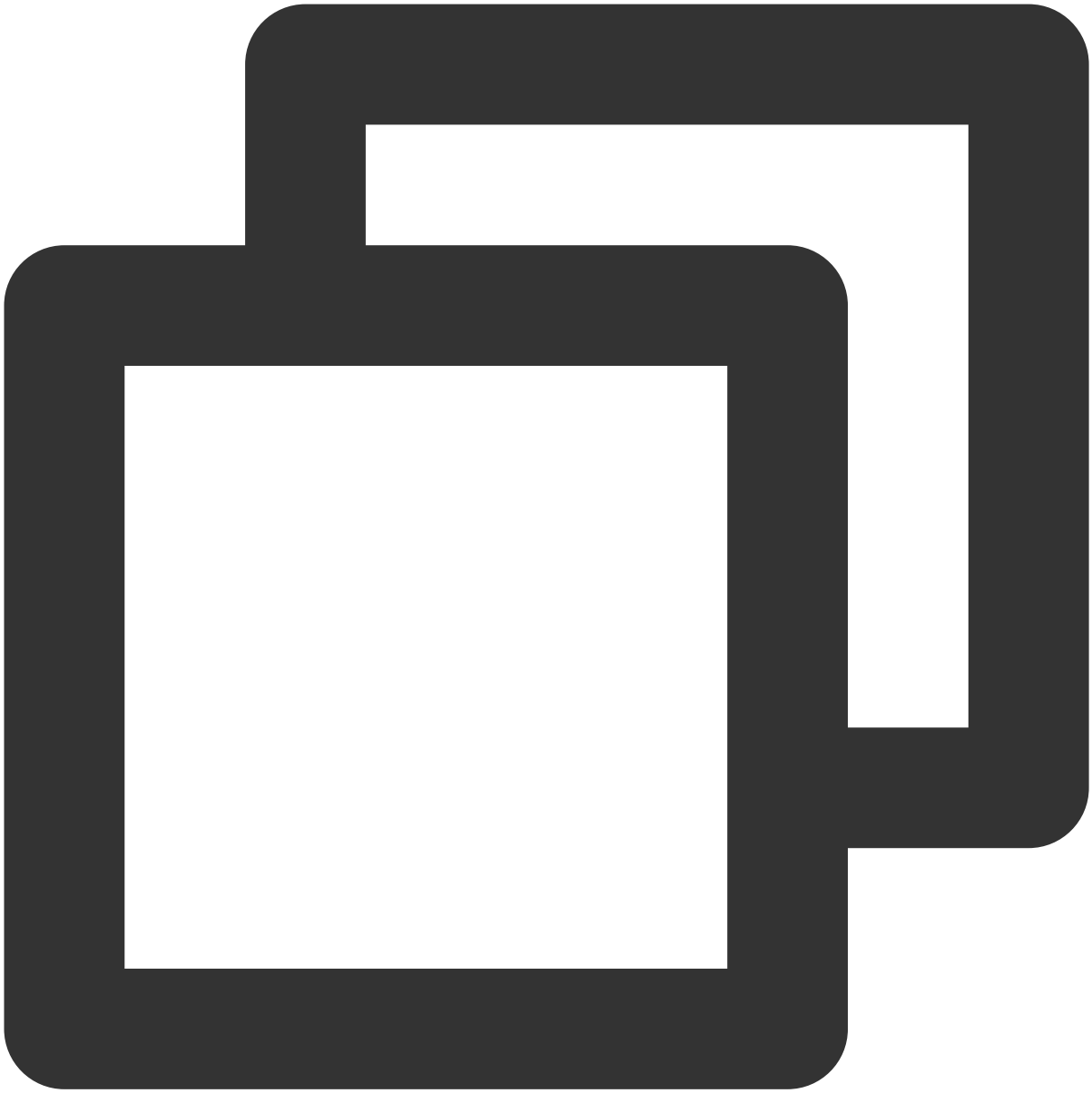
```
class ChatInfo {
  String? sdkappid;
  String? userSig;
  String? userID;
```

```
  ChatInfo.fromJSON(Map<String, dynamic> json) {
    sdkappid = json["sdkappid"].toString();
    userSig = json["userSig"].toString();
    userID = json["userID"].toString();
  }

  Map<String, String> toMap(){
    final Map<String, String> map = {};
    if(sdkappid != null){
      map["sdkappid"] = sdkappid!;
    }
    if(userSig != null){
      map["userSig"] = userSig!;
    }
    if(userID != null){
      map["userID"] = userID!;
    }
    return map;
  }
}

class CallInfo{
  String? userID;
  String? groupID;

  CallInfo();

  CallInfo.fromJSON(Map<String, dynamic> json) {
    groupID = json["groupID"].toString();
    userID = json["userID"].toString();
  }

  Map<String, String> toMap(){
    final Map<String, String> map = {};
    if(userID != null){
      map["userID"] = userID!;
    }
    if(groupID != null){
      map["groupID"] = groupID!;
    }
    return map;
  }
}
```

Chat 模块

首先编写IM引擎。本模块所有代码及文件，均在 `lib/chat` 目录下。

1. 新建全局状态管理Model，名为 `model.dart` 。
该Model用于挂载初始化并管理腾讯云IM Flutter模块，离线推送能力，全局状态管理，维护与Native间通信。
是整个Chat模块的核心。
详细代码可查看Demo源码。重点关注三个部分：
Future _handleMessage(MethodCall call): 动态监听 Native 透传来的事件，包括登录信息及点击推送事件。
Future handleClickNotification(Map<String, dynamic> msg): 点击通知处理事件，来自Native透传，从 Map 中取出数据，跳转至对应的子模块，如某个具体会话。
Future initChat(): 初始化腾讯云IM/登录腾讯云IM/并完成离线推送的初始化及Token上报。该方法使用线程锁机制，保证同时只能执行一个，并在初始化成功后，不重复执行。
说明：
请根据 离线推送接入指引，完成厂商离线推送功能接入，才可正常上报推送Token，使用推送功能。
2. 新建 `chat_main.dart` 文件，用于Chat模块主入口。
该页面也是Flutter Chat模块的首页。
在Demo中，该页面在未登录前为加载状态，登录后展示会话列表。
此外，还需要在这里，完成 `didChangeAppLifecycleState` 监听与前后台切换事件上报，详情请查看离线推送插件文档步骤5。
详细代码可查看Demo源码。
3. 新建 `push.dart` 文件，用于单例管理 离线推送插件 能力。用于获取并上报Token/获取推送权限等操作。详细代码可查看Demo源码。
4. 新建 `conversation.dart` 文件，用于承载TUIKit的会话模块组件 `TIMUIKitConversation` 。详细代码可查看Demo源码。
5. 新建 `chat.dart` 文件，用于承载TUIKit的历史消息列表和发送消息模块组件 `TIMUIKitChat` 。
该页面还有跳转至 Profile 及 Group Profile 页面的能力。
详细代码可查看Demo源码。
6. 新建 `user_profile.dart` 文件，用于承载TUIKit的用户信息及关系链管理模块组件 `TIMUIKitProfile` 。详细代码可查看Demo源码。
7. 新建 `group_profile.dart` 文件，用于承载TUIKit的群信息及群管理模块组件 `TIMUIKitGroupProfile` 。详细代码可查看Demo源码。
此时，Chat模块已开发完成。最终结构如下：



```
tencent_chat_module/
├── lib/
│   └── call/
│       └── chat.dart
│       └── model.dart
│       └── chat_main.dart
│       └── push.dart
│       └── conversation.dart
│       └── user_profile.dart
│       └── group_profile.dart
│   └── chat/
```

```
└── common/
```

Call 模块

该模块用于承载音视频通话能力，该能力由 音视频通话插件 提供。

该模块的核心是，监听收到新的通话邀请时，通过调用Native方法，自动弹出通话页面；并接受 Chat 模块经由Native转发来的通话请求，主动发起通话。

首先编写IM引擎。本模块所有代码及文件，均在 `lib/call` 目录下。

1. 新建全局状态管理Model，名为 `model.dart` 。

该Model用于挂载初始化并管理 音视频通话插件，全局状态管理，维护与Native间通信。

是整个Call模块的核心。

详细代码可查看Demo源码。重点关注两个部分：

_onRtcListener = TUICallingListener(...): 定义了通话事件的监听器，通过 Method Channel 通知Native层，动态控制 Call 模块所属的 ViewController(iOS)/Activity(Android) 的前端展示与否。

Future _handleMessage(MethodCall call): 动态监听 Native 透传来的主动发起通话请求，来自 Call 模块的调用，主动发起通话。

2. 新建 `call_main.dart` 文件，用于Call模块主入口。

该组件用于注入音视频通话插件所需绑定的navigatorKey。

详细代码可查看Demo源码。

配置各个Flutter引擎的入口

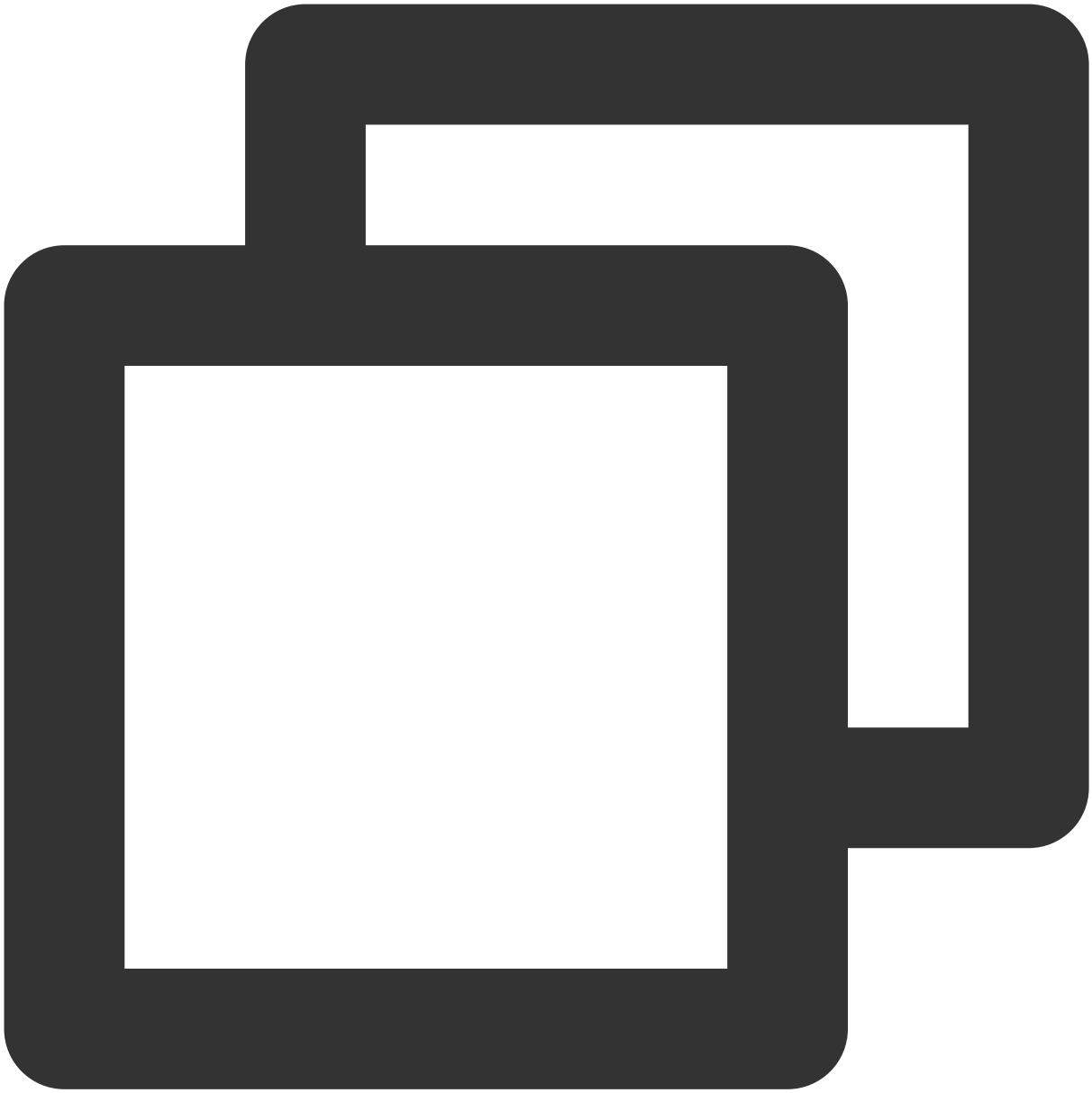
开发完上述三个模块后，现在可完成最终对外暴露的main方法，作为Flutter引擎的入口。

1. 默认入口

打开 `lib/main.dart` 文件，将 `main()` 方法改成一个空 MaterialApp 即可。

该方法作为 Flutter Module 的默认入口，在Flutter多引擎，使用FlutterEngineGroup管理的背景下，如果没有子Flutter Engine不设置任何entry point，这个方法就不会被用到。

例如，在我们的场景中，这个默认 `main()` 方法就没有被用上。

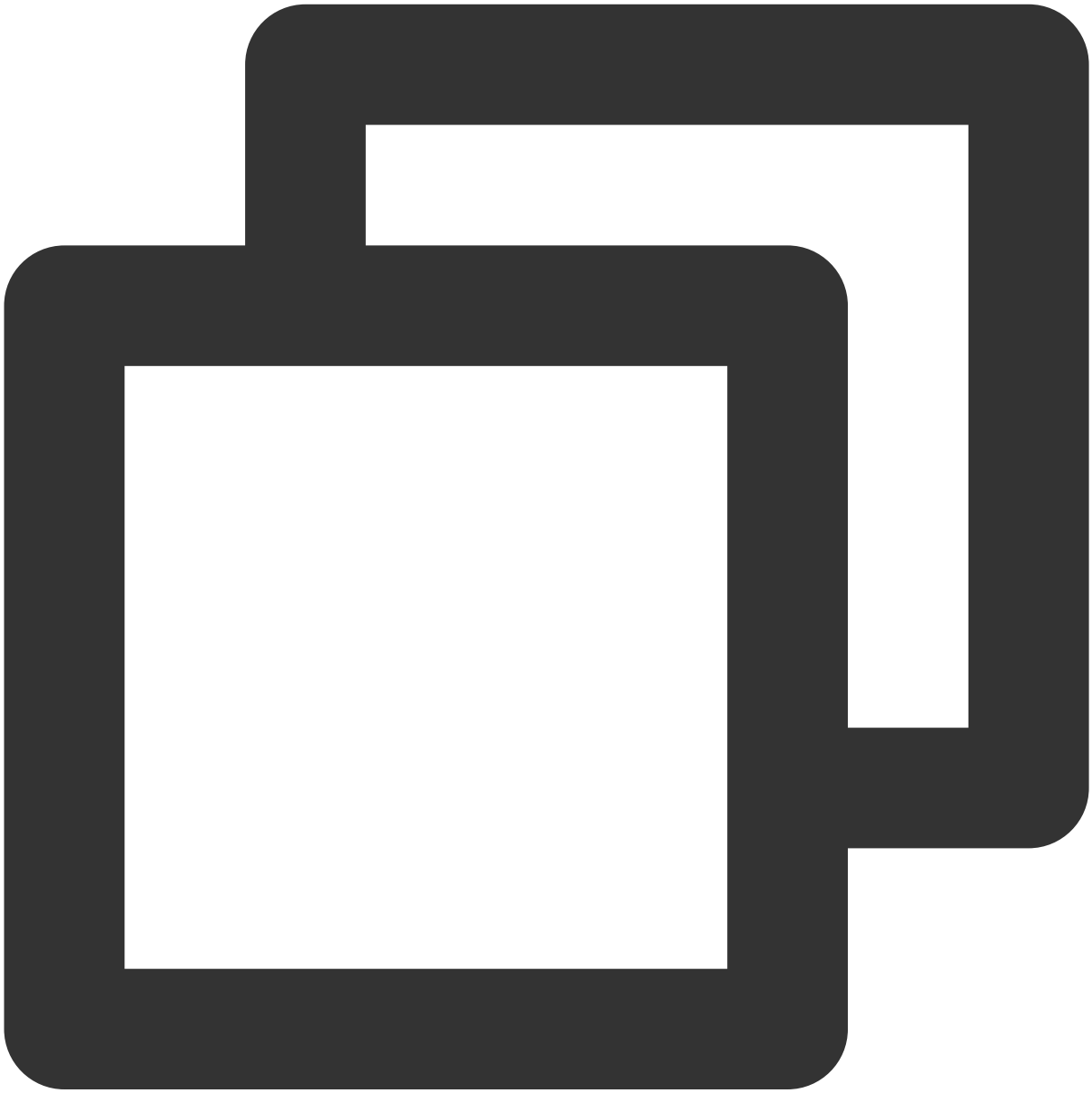


```
void main() {
  WidgetsFlutterBinding.ensureInitialized();

  runApp(MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: Container(),
  ));
}
```

2. 配置 Chat 模块的入口

使用 `@pragma('vm:entry-point')` 注解，将该方法标记为一个 `entry-point` 入口。方法名 `chatMain` 即该入口的名称，在Native中，也使用该名称，创建对应Flutter引擎。

使用全局 `ChangeNotifierProvider` 状态管理，维护 `ChatInfoModel` 数据及业务逻辑。



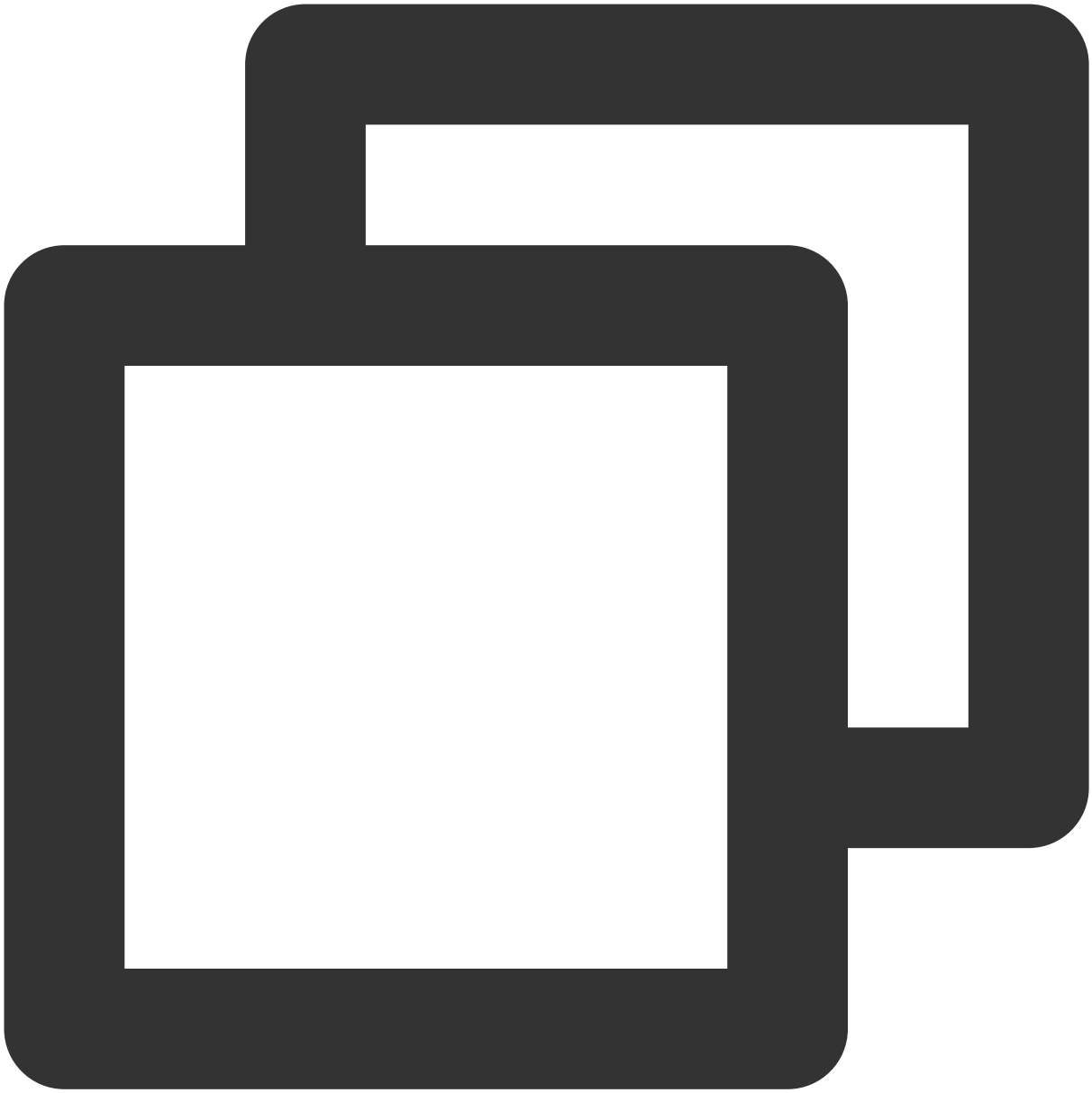
```
@pragma('vm:entry-point')
void chatMain() {
  // This call ensures the Flutter binding has been set up before creating the
  // MethodChannel-based model.
  WidgetsFlutterBinding.ensureInitialized();
```

```
  final model = ChatInfoModel();

  runApp(
    ChangeNotifierProvider.value(
      value: model,
      child: const ChatAPP(),
    ),
  );
}
```

3. 配置 Call 模块的入口

同理，该入口命名为 `callMain` 。

使用全局 `ChangeNotifierProvider` 状态管理，维护 `CallInfoModel` 数据及业务逻辑。



```
@pragma('vm:entry-point')
void callMain() {
  // This call ensures the Flutter binding has been set up before creating the
  // MethodChannel-based model.
  WidgetsFlutterBinding.ensureInitialized();

  final model = CallInfoModel();

  runApp(
    ChangeNotifierProvider.value(
      value: model,
```

```
        child: const CallAPP(),
      ),
    );
  }
```

至此，Flutter Module部分，Dart代码编写完成。

接下来，开始编写 Native 代码。

iOS Native 开发

本文以 Swift 语言为例。

说明：

以下代码结构，仅供参考，您可根据需要灵活组织。

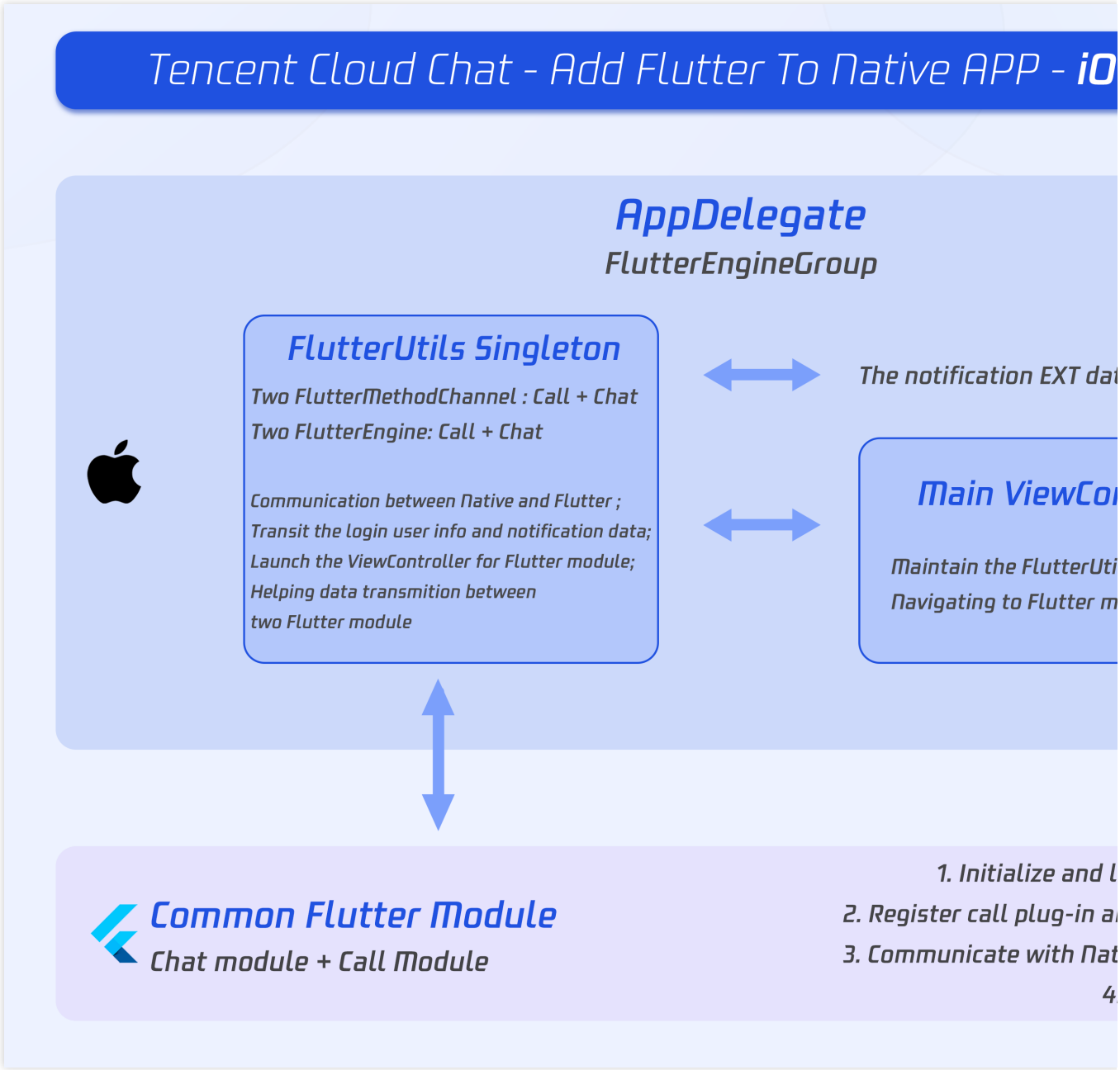
进入您的iOS项目目录。

如果您现有的应用程序，假设叫做 `MyApp` ， 还没有Podfile，请按照CocoaPods入门指南将 `Podfile` 添加到项目中。

引入 Flutter Module

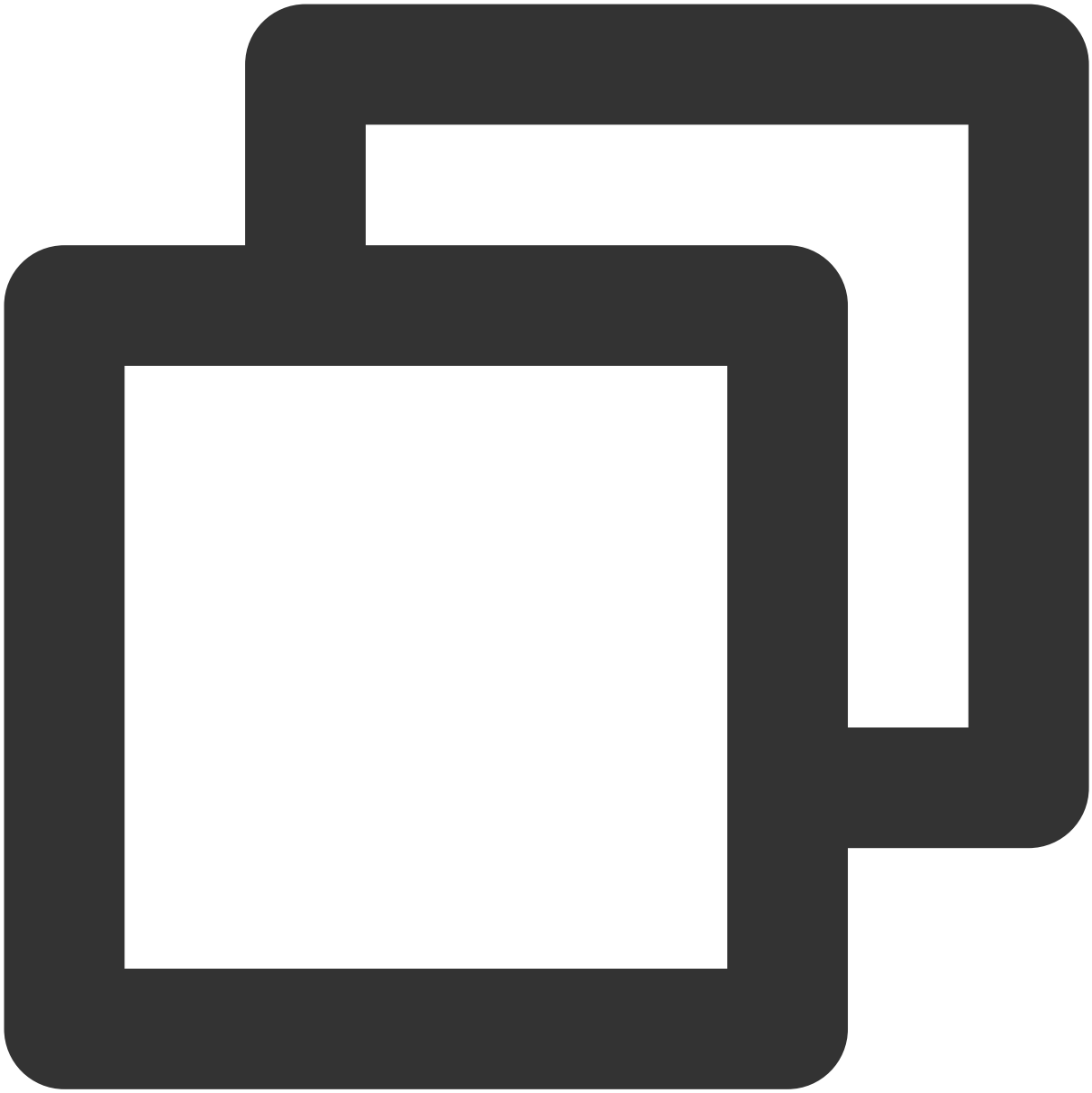
请参考此部分，将Flutter module引入您的原生应用程序中。建议采用方式一。

在 iOS 项目中，管理Flutter引擎



创建一个 `FlutterEngineGroup` （Flutter 引擎组），统一管理多个引擎实例。

在 `AppDelegate.swift` 文件中，添加如下代码：



```
@UIApplicationMain
class AppDelegate: FlutterAppDelegate {
  lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", projec
  ...
}
```

创建一个用于管理Flutter引擎的单例对象。

这个 Swift 单例对象，用于集中管理 Flutter 实例，并方便在项目中各处，直接调用。

Demo代码的逻辑是，使用新的路由，承载Chat的ViewController；Call的ViewController，通过present和dismiss动态弹窗维护。

新建 `FlutterUtils.swift` 文件，编写代码。本部分详细代码，可查看Demo源码。

重点关注：

private override init(): 初始化各 Flutter 引擎实例，注册Method Channel，监听事件。

func reportChatInfo(): 将用户登录信息和SDKAPPID透传至Flutter Module，使Flutter层得以初始化并登录腾讯云IM。

func launchCallFunc(): 用于拉起Call的Flutter页面，可被Call模块收到通话邀请触发，也可被Chat模块主动发起通话触发。

func triggerNotification(msg: String): 将 iOS Native 层收到的离线推送消息点击事件，及其包含的ext信息，以 JSON String形式，透传至 Flutter 层绑定的监听处理事件。用于处理离线推送点击跳转，例如至对应会话。

监听及转发离线推送点击事件

离线推送的初始化/Token上报/点击事件对应的会话跳转处理，已在Flutter Chat模块中进行，因此，Native区域，仅需透传点击通知事件的ext即可。

之所以这么做，是因为点击通知事件已在Native被拦截消费，Flutter层无法直接拿到，必须经由Native转发。

在 `AppDelegate.swift` 文件中，新增如下代码。具体代码，可以参考Demo源码。

```
@UIApplicationMain
class AppDelegate: FlutterAppDelegate {
  lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", project: nil)

  override func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchO

      if #available(iOS 10.0, *) {
        UNUserNotificationCenter.current().delegate = self
      }

      if let remoteNotification = launchOptions?[UIApplication.LaunchOptionsKey.remoteNotification]{
          let notificationExt: [AnyHashable:Any] = remoteNotification as! [AnyHashable:Any]
          let remoteNotificationString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
          FlutterUtils.shared.triggerNotification(msg: remoteNotificationString)
      }

    return super.application(application, didFinishLaunchingWithOptions: launchOptions);
  }

    override func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, wit
        @escaping () -> Void) {
        let notificationExt: Dictionary = response.notification.request.content.userInfo
        let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
        FlutterUtils.shared.triggerNotification(msg: notificationExtString)
    }

}
```

此时，iOS Native层编写完成。

Android Native 开发

本文以 Kotlin 语言为例。

说明：

以下代码结构，仅供参考，您可根据需要灵活组织。

引入 Flutter Module

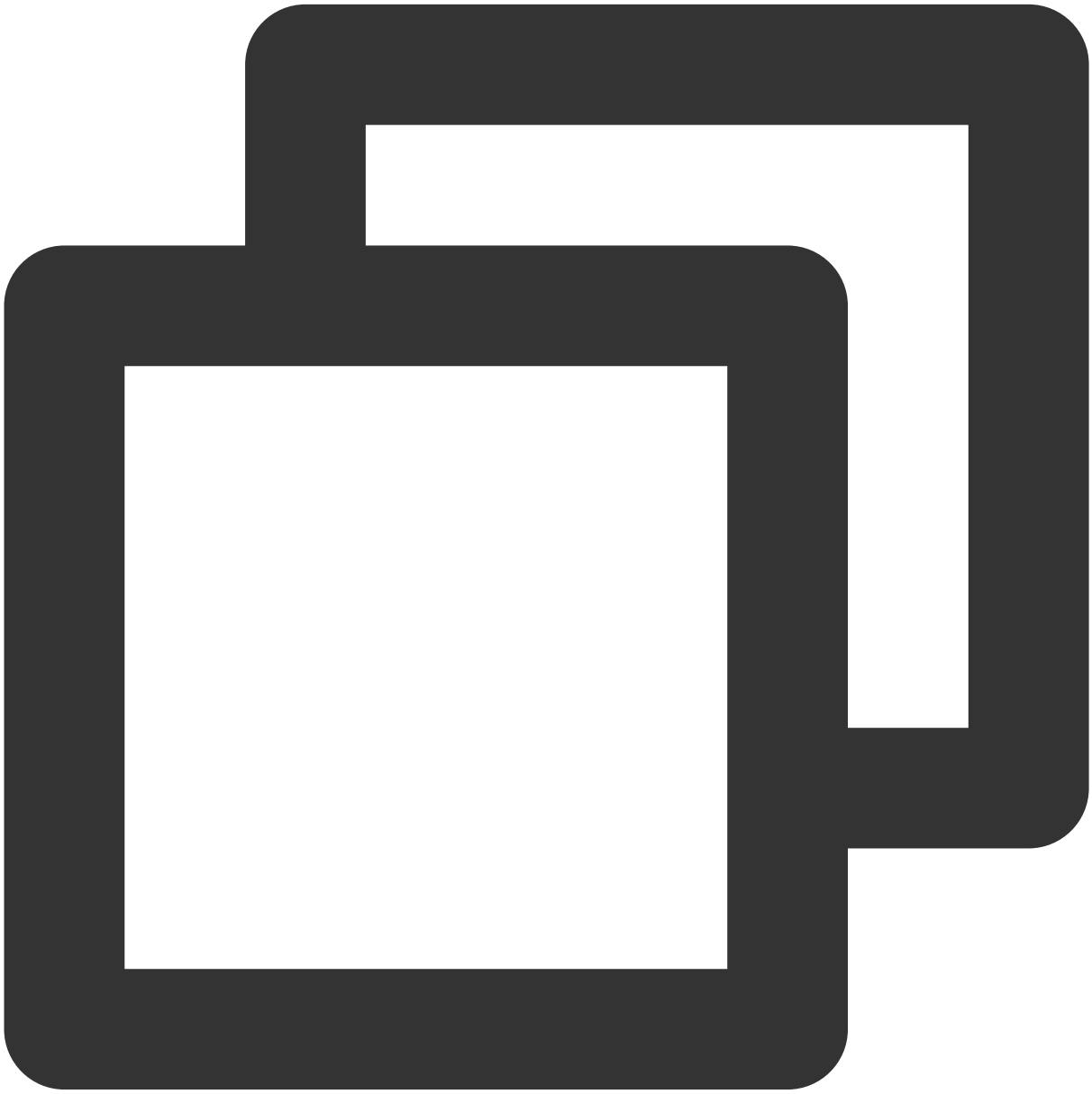
请参考此部分，将Flutter module引入您的原生应用程序中。建议采用方式二。

在 Android 项目中，管理Flutter引擎

创建一个用于管理Flutter引擎的单例对象。

这个 Kotlin 单例对象，用于集中管理 Flutter 实例，并方便在项目中各处，直接调用。

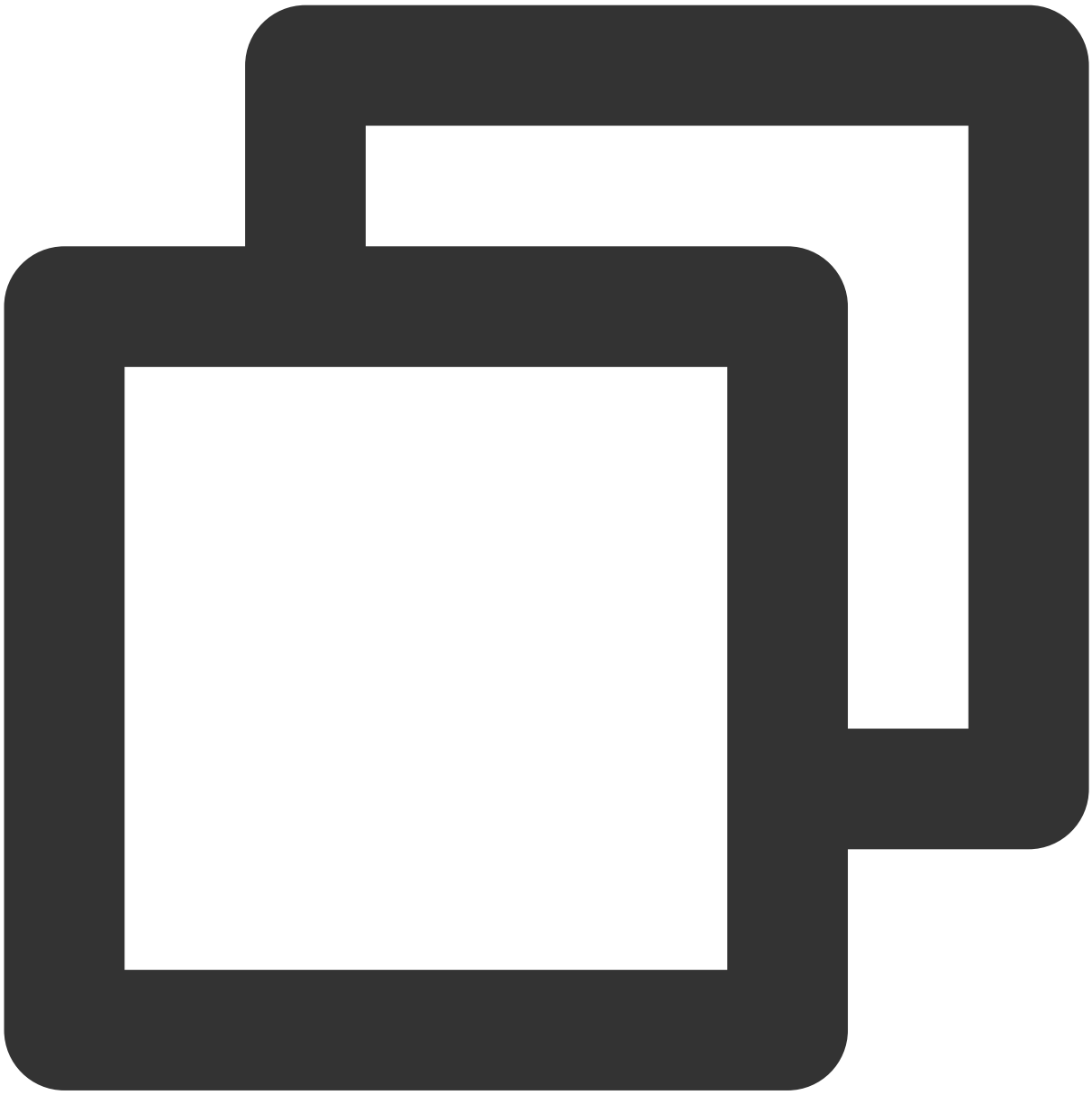
新建 `FlutterUtils.kt` 文件，并定义 `FlutterUtils` 静态类。



```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

创建 `FlutterEngineGroup` （Flutter 引擎组），统一管理多个引擎实例。

在 `FlutterUtils.kt` 文件中，定义一个 `FlutterEngineGroup` ，及配套各个Flutter Engine实例和Method Channel，并在初始化时，将其初始化。



```
lateinit var context : Context
lateinit var flutterEngines: FlutterEngineGroup
private lateinit var chatFlutterEngine:FlutterEngine
private lateinit var callFlutterEngine:FlutterEngine

lateinit var chatMethodChannel: MethodChannel
lateinit var callMethodChannel: MethodChannel
```

```
// 初始化
flutterEngines = FlutterEngineGroup(context)
...
```

继续完成该用于管理Flutter引擎的单例对象。

Demo代码的逻辑是，使用新的路由，承载Chat和Call的Activity。

Chat的Activity，由用户主动进入及退出；Call的Activity，由监听器或主动外呼，自动导航进及返回出。

重点关注：

fun init(): 初始化各 Flutter 引擎实例，注册Method Channel，监听事件。

fun reportChatInfo(): 将用户登录信息和SDKAPPID透传至Flutter Module，使Flutter层得以初始化并登录腾讯云IM。

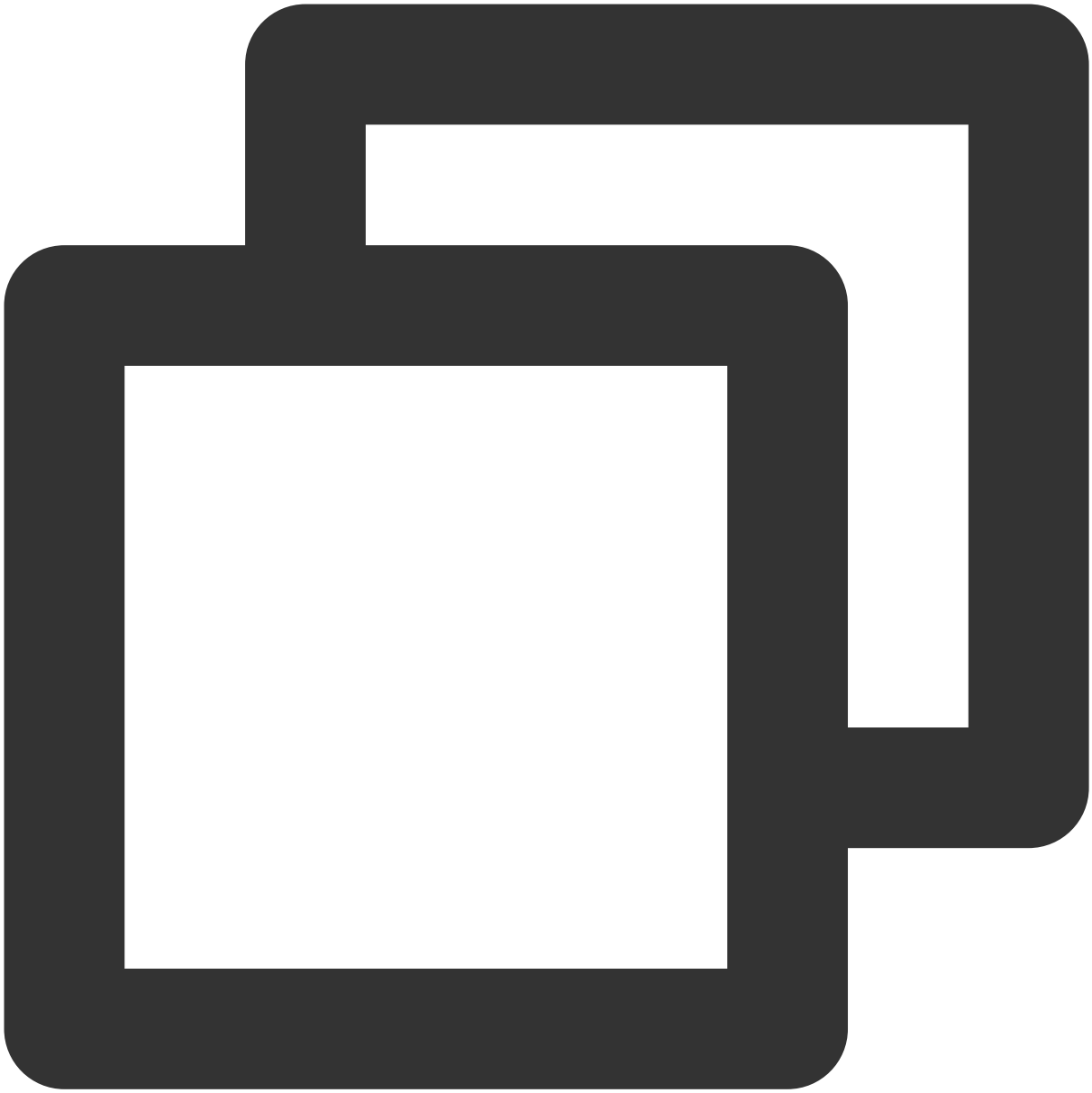
fun launchCallFunc(): 用于拉起Call的Flutter页面，可被Call模块收到通话邀请触发，也可被Chat模块主动发起通话触发。

fun triggerNotification(msg: String): 将 iOS Native 层收到的离线推送消息点击事件，及其包含的ext信息，以 JSON String形式，透传至 Flutter 层绑定的监听处理事件。用于处理离线推送点击跳转，例如至对应会话。

本单例 object 的详细代码，可以参考Demo源码。

在 总入口 `MyApplication` 中，初始化上述对象。

在 `MyApplication.kt` 文件中，将全局context传入单例对象，并执行初始化。



```
class MyApplication : MultiDexApplication() {

    override fun onCreate() {
        super.onCreate()
        FlutterUtils.context = this // new
        FlutterUtils.init()         // new
    }
}
```

监听及转发离线推送点击事件

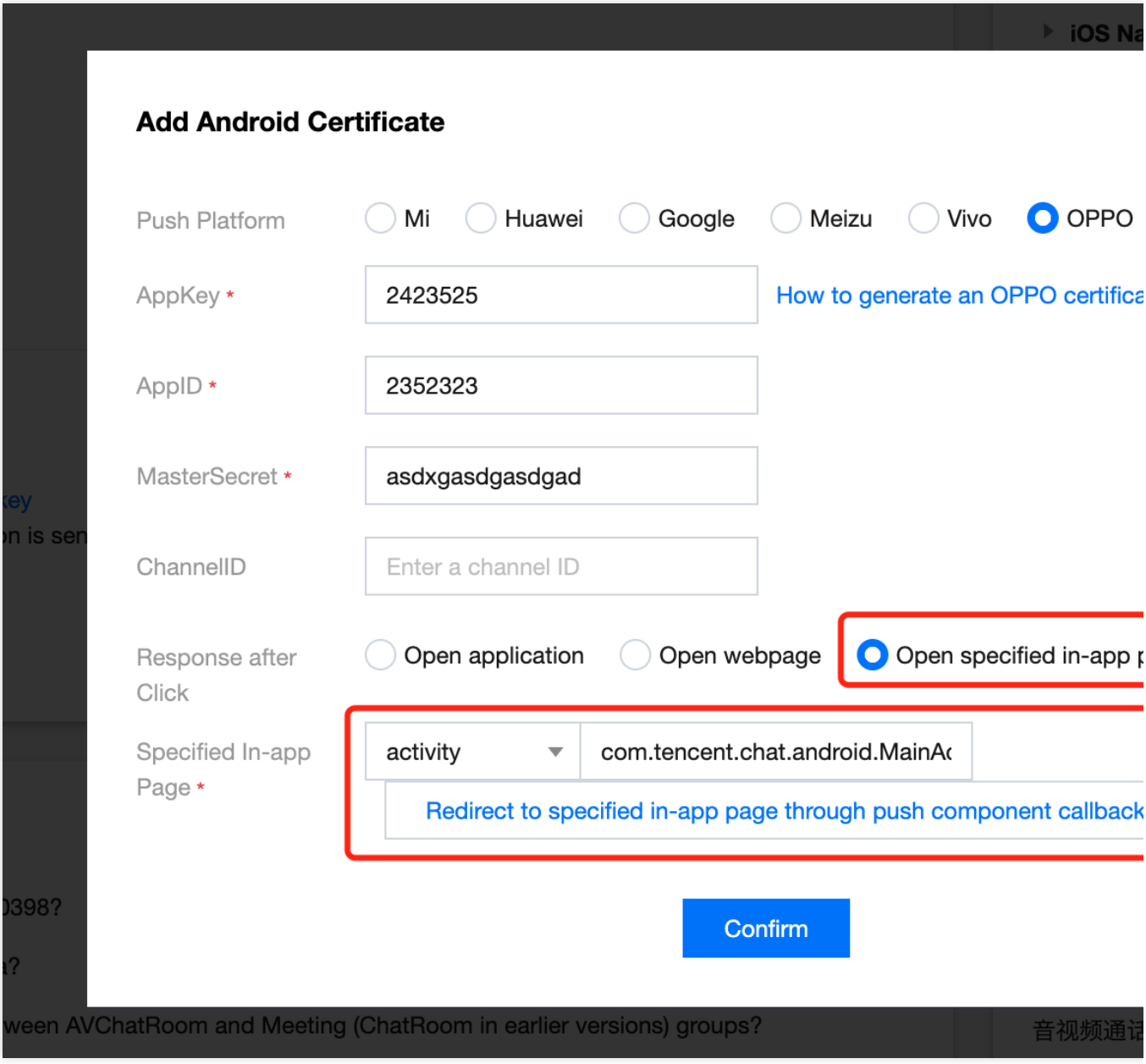
离线推送的初始化/Token上报/点击事件对应的会话跳转处理，已在Flutter Chat模块中进行，因此，Native区域，仅需透传点击通知事件的ext即可。

之所以这么做，是因为点击通知事件已在Native被拦截消费，Flutter层无法直接拿到，必须经由Native转发。

注意：

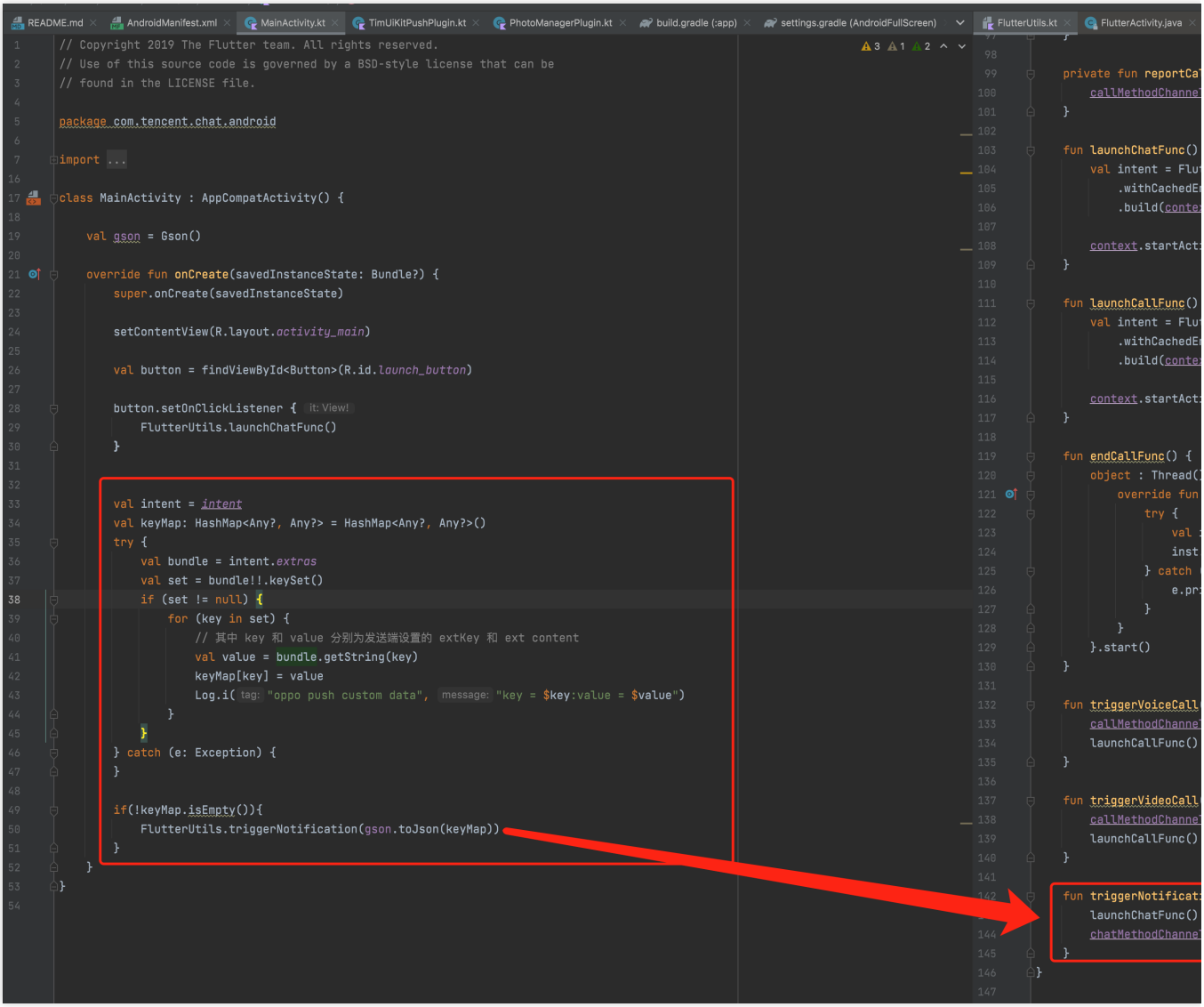
由于不同厂商的离线推送接入步骤不一致，本文以OPPO为例，全部厂商接入方案，可查看本文档.

在腾讯云IM控制台中，新增OPPO的推送证书， `点击后续动作` 选择 `打开应用内指定页面` ， `应用内页面` 以 `Activity` 方式，配置一个用于处理离线推送信息的页面，建议为应用首页。如，我们的Demo配置为： `com.tencent.chat.android.MainActivity` .



在上方控制台配置的用于离线推送的Activity文件中，新增如下代码。

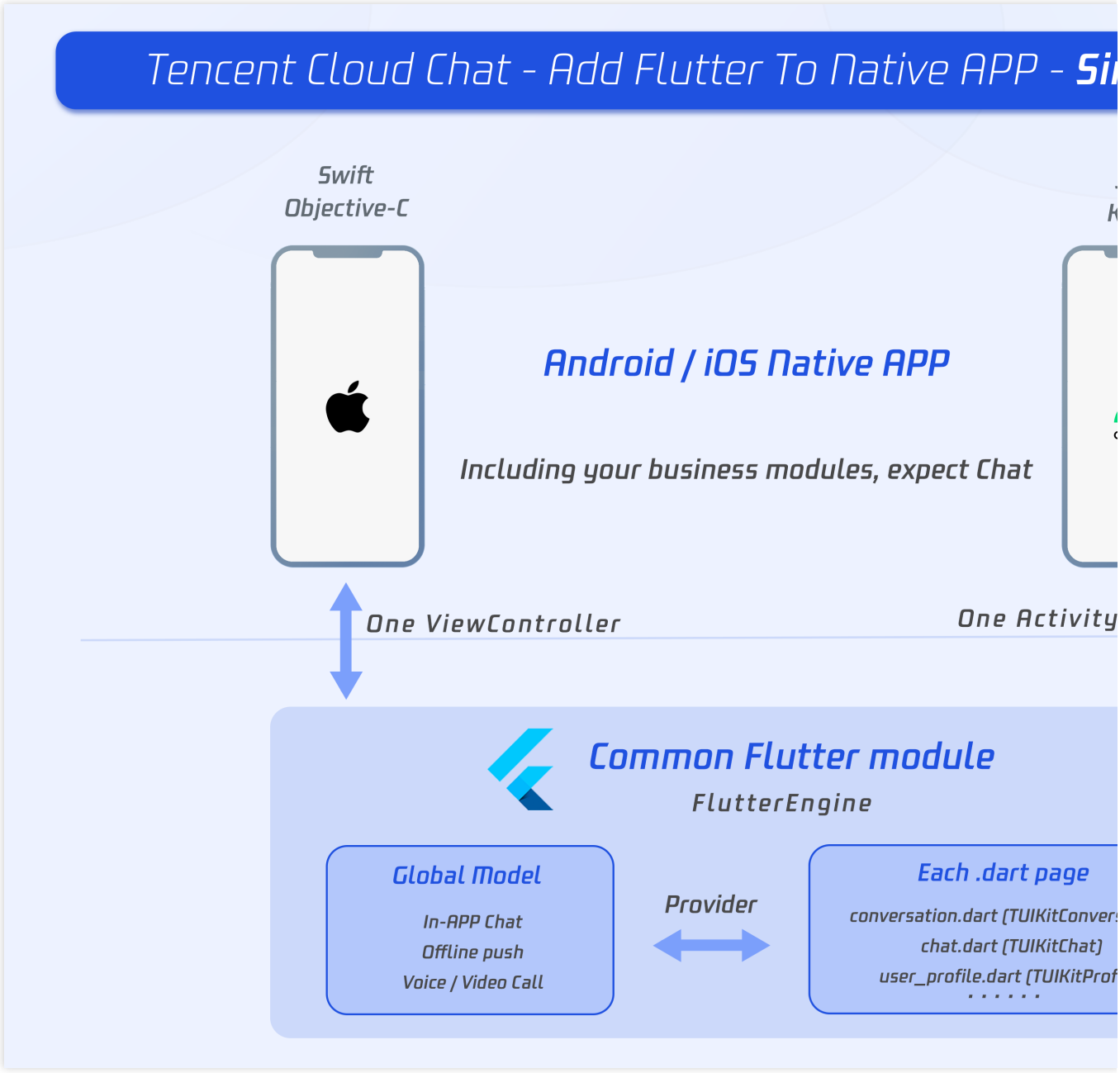
该代码的作用是，当厂商拉起相应Activity时，从Bundle中取出HashMap形式ext信息，触发单例对象中的方法，将这个信息，手动转发至Flutter中。具体代码，可以参考Demo源码。



此时，Android Native层编写完成。

方案二：Flutter 单引擎方案

本方案，将 Chat 模块和 Call 模块，写在同一个 Flutter 引擎实例中。

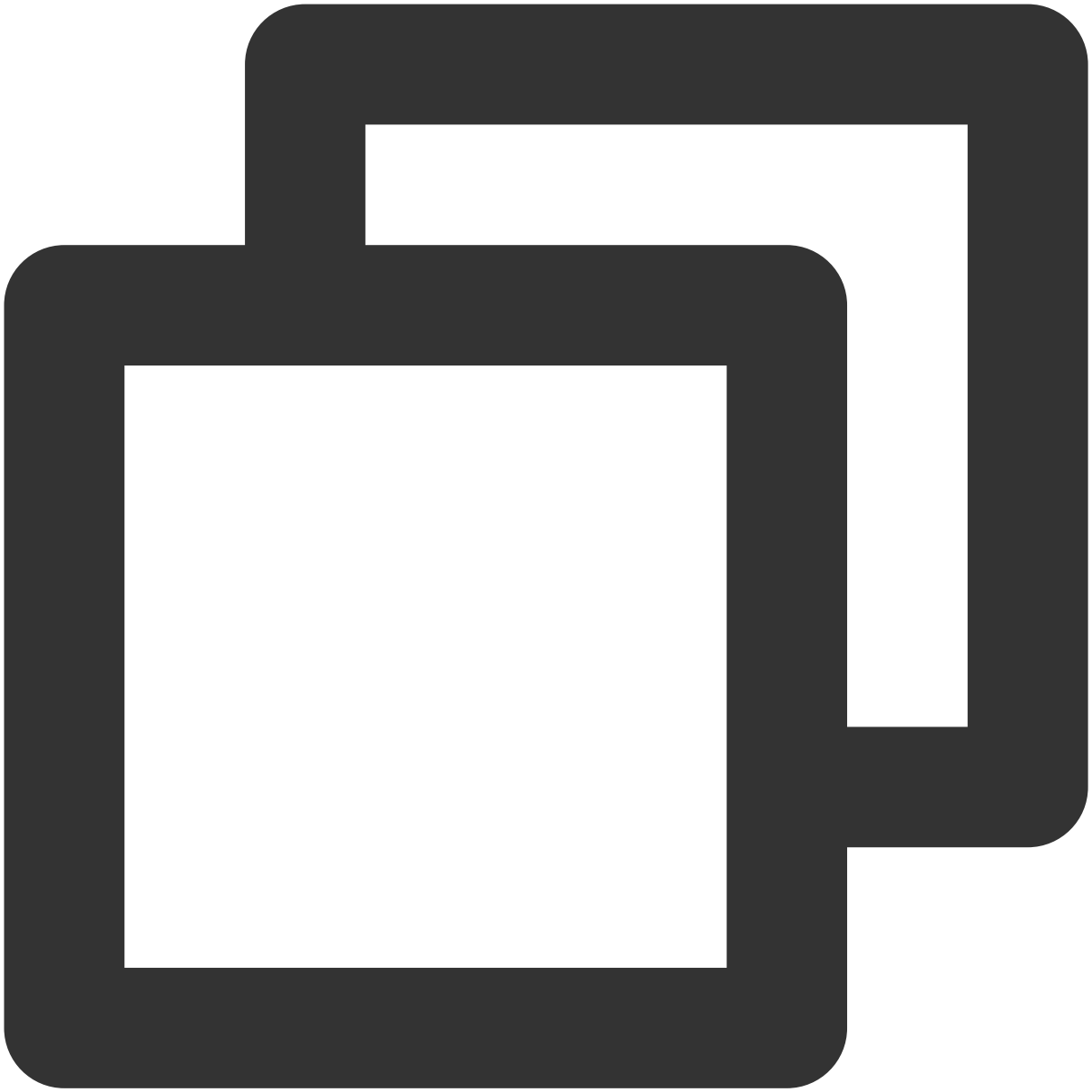


这两个模块只能同时出现同时隐藏，仅需维护一个 Flutter 引擎即可。

Flutter Module 开发

要将 Flutter 嵌入到现有应用程序中，请首先创建一个 Flutter 模块。

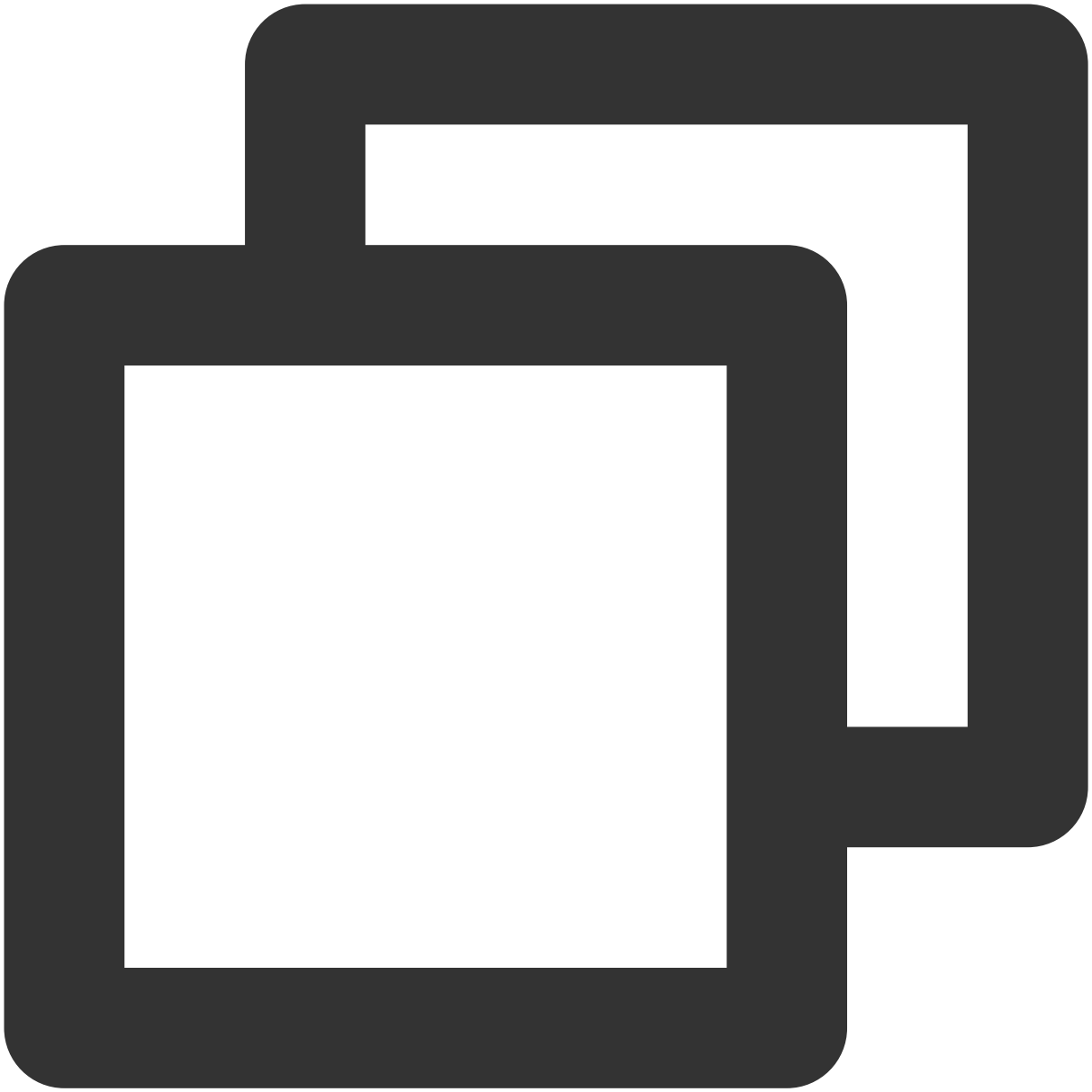
在您项目的根目录外层运行以下内容：



```
cd some/path/
flutter create --template module tencent_chat_module
```

这会在 some/path/tencent_chat_module/ 创建一个 Flutter 模块项目。 在该目录中，您可以运行与在任何其他 Flutter 项目中相同的 Flutter 命令，例如 `flutter run --debug` 或 `flutter build ios` 。 您还可以使用 Flutter 和 Dart 插件在 Android Studio, IntelliJ 或 VS Code 中运行该模块。 该项目在嵌入到现有应用程序之前包含模块的单视图示例版本，这对于测试代码的仅 Flutter 部分很有用。

`tencent_chat_module` 模块目录结构类似于普通的 Flutter 应用程序：



```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

现在，我们可以在 `lib/` 中，编写代码了。

main.dart

修改 `main.dart` 文件，引入TUIKit, 离线推送插件及音视频通话插件。
全局状态，我们的IM SDK及Method Channel与Native通信状态，管理于 `ChatInfoModel` 中。
接收到Native传来的用户信息及SDKAPPID后，调用 `_coreInstance.init()` 及 `_coreInstance.login()` 初始化并登录腾讯云IM。并初始化音视频推送插件及离线推送插件，完成推送Token上报。
说明：
请根据 离线推送接入指引，完成厂商离线推送功能接入，才可正常上报推送Token，使用推送功能。
对于音视频通话插件，需要关注：
监听收到新的通话邀请时，通过调用Native方法，让Native检测用户当前是否在本Flutter模块页面，如果不在，需要强制将前端页面调整至本模块，以展示来电页面。
对于离线推送插件，需要关注：
点击通知处理事件，来自Native透传，从 Map 中取出数据，跳转至对应的子模块，如某个具体会话。
完成首页的制作，在未登录时展示加载动画；登录成功后，展示会话列表页面。
此外，还需要在这里，完成 `didChangeAppLifecycleState` 监听与前后台切换事件上报，详情请查看离线推送插件文档步骤5。
详细代码可查看Demo源码。

其他TUIKit模块引入

1. 新建 `push.dart` 文件，用于单例管理 离线推送插件 能力。用于获取并上报Token/获取推送权限等操作。详细代码可查看Demo源码。
2. 新建 `conversation.dart` 文件，用于承载TUIKit的会话模块组件 `TIMUIKitConversation` 。详细代码可查看Demo源码。
3. 新建 `chat.dart` 文件，用于承载TUIKit的历史消息列表和发送消息模块组件 `TIMUIKitChat` 。
该页面还有跳转至 Profile 及 Group Profile 页面的能力。
详细代码可查看Demo源码。
4. 新建 `user_profile.dart` 文件，用于承载TUIKit的用户信息及关系链管理模块组件 `TIMUIKitProfile` 。详细代码可查看Demo源码。
5. 新建 `group_profile.dart` 文件，用于承载TUIKit的群信息及群管理模块组件 `TIMUIKitGroupProfile` 。详细代码可查看Demo源码。
至此，统一的Flutter Module开发完成。

iOS Native 开发

本文以 Swift 语言为例。
说明：
以下代码结构，仅供参考，您可根据需要灵活组织。
进入您的iOS项目目录。

如果您现有的应用程序，假设叫做 `MyApp` ， 还没有Podfile，请按照CocoaPods入门指南将 `Podfile` 添加到项目中。

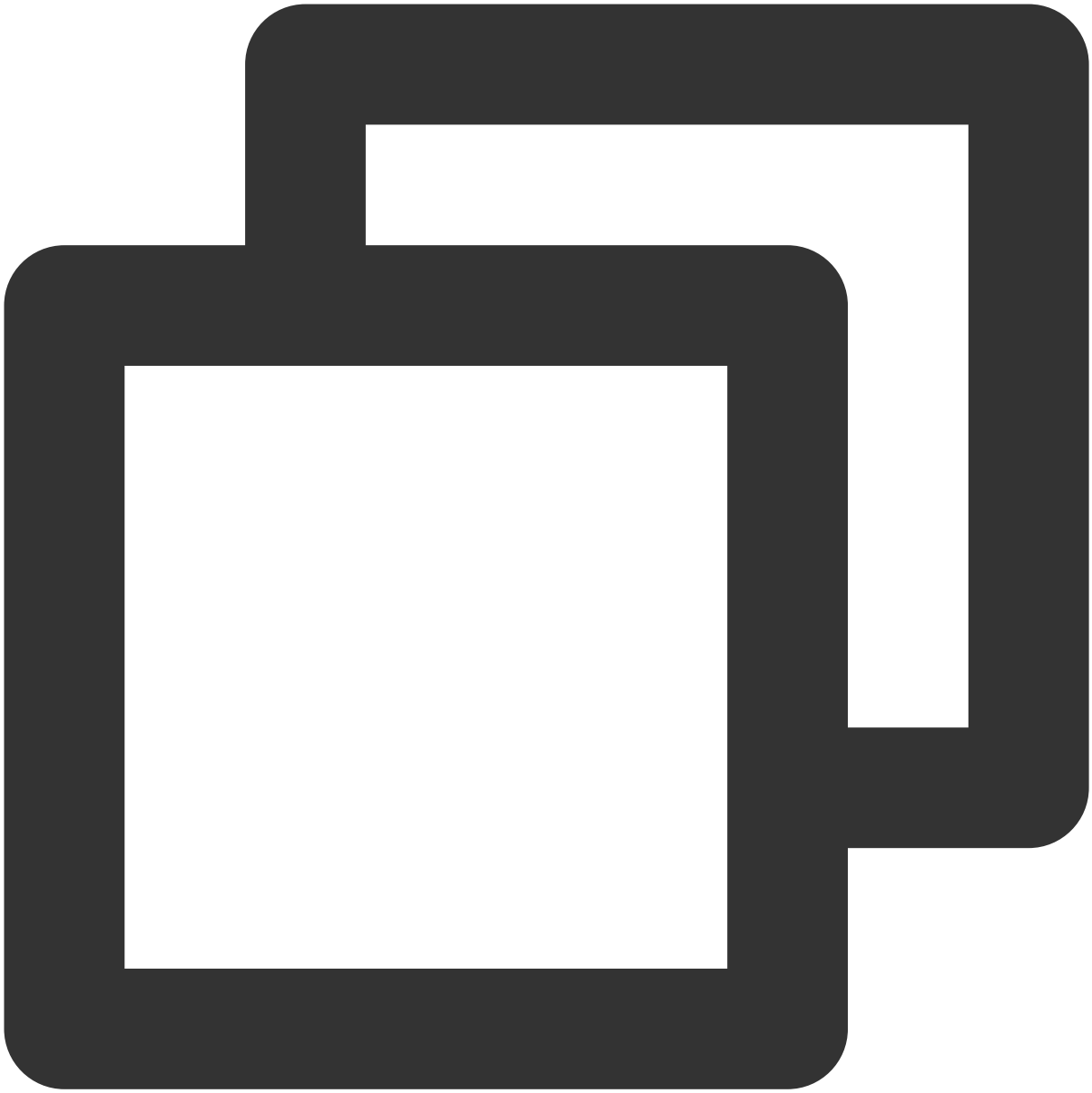
引入 Flutter Module

请参考此部分，将Flutter module引入您的原生应用程序中。建议采用方式一。

在 iOS 项目中，管理Flutter引擎

创建一个FlutterEngine。

创建FlutterEngine的适当位置特定于您的主应用程序入口。作为一个例子，我们演示了如何在 `AppDelegate` 中的app启动时创建一个FlutterEngine，并公开为一个属性。



```
import UIKit
import Flutter
import FlutterPluginRegistrant

@UIApplicationMain
class AppDelegate: FlutterAppDelegate { // More on the FlutterAppDelegate.
  lazy var flutterEngine = FlutterEngine(name: "tencent cloud chat")

  override func application(_ application: UIApplication, didFinishLaunchingWithOpt
    // Runs the default Dart entrypoint with a default Flutter route.
    flutterEngine.run();
```

```
        GeneratedPluginRegistrant.register(with: self.flutterEngine);
        return super.application(application, didFinishLaunchingWithOptions: launchOpti
    }
}
```

创建一个用于管理Flutter引擎的单例对象。

这个 Swift 单例对象，用于集中管理 Flutter Method Channel，并提供一系列与 Flutter Module 通信的方法，方便在项目中各处，直接调用。

这些方法包括：

private override init(): 初始化 Method Channel，并为其绑定事件监听方法。

func reportChatInfo(): 将用户登录信息和SDKAPPID透传至Flutter Module，使Flutter层得以初始化并登录腾讯云IM。

func launchChatFunc(): 拉起或导航至 Flutter Module 所在 ViewController。

func triggerNotification(msg: String): 将 iOS Native 层收到的离线推送消息点击事件，及其包含的ext信息，以 JSON String形式，透传至 Flutter 层绑定的监听处理事件。用于处理离线推送点击跳转，例如至对应会话。

详细代码可查看Demo源码。

监听及转发离线推送点击事件

离线推送的初始化/Token上报/点击事件对应的会话跳转处理，已在Flutter Chat模块中进行，因此，Native区域，仅需透传点击通知事件的ext即可。

之所以这么做，是因为点击通知事件已在Native被拦截消费，Flutter层无法直接拿到，必须经由Native转发。

在 `AppDelegate.swift` 文件中，新增如下代码。具体代码，可以参考Demo源码。

```
@UIApplicationMain
class AppDelegate: FlutterAppDelegate { // More on the FlutterAppDelegate.
  lazy var flutterEngine = FlutterEngine(name: "io.flutter")

  override func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptions

      if #available(iOS 10.0, *) {
        UNUserNotificationCenter.current().delegate = self as? UNUserNotificationCenterDelegate  Conditional cast from 'AppDelegate' to 'ar
      }

      if let remoteNotification = launchOptions?[UIApplication.LaunchOptionsKey.remoteNotification]{
          let notificationExt: [AnyHashable:Any] = remoteNotification as! [AnyHashable:Any]
          let remoteNotificationString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
          FlutterUtils.shared.triggerNotification(msg: remoteNotificationString)
      }

    // Runs the default Dart entrypoint with a default Flutter route.
    flutterEngine.run();
    // Used to connect plugins (only if you have plugins with iOS platform code).
    GeneratedPluginRegistrant.register(with: self.flutterEngine);
    return super.application(application, didFinishLaunchingWithOptions: launchOptions);
  }

    override func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompl
        @escaping () -> Void) {
        let notificationExt: Dictionary = response.notification.request.content.userInfo
        let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
        FlutterUtils.shared.triggerNotification(msg: notificationExtString)
    }

}
```

此时，iOS Native层编写完成。

Android Native 开发

本文以 Kotlin 语言为例。

说明：

以下代码结构，仅供参考，您可根据需要灵活组织。

引入 Flutter Module

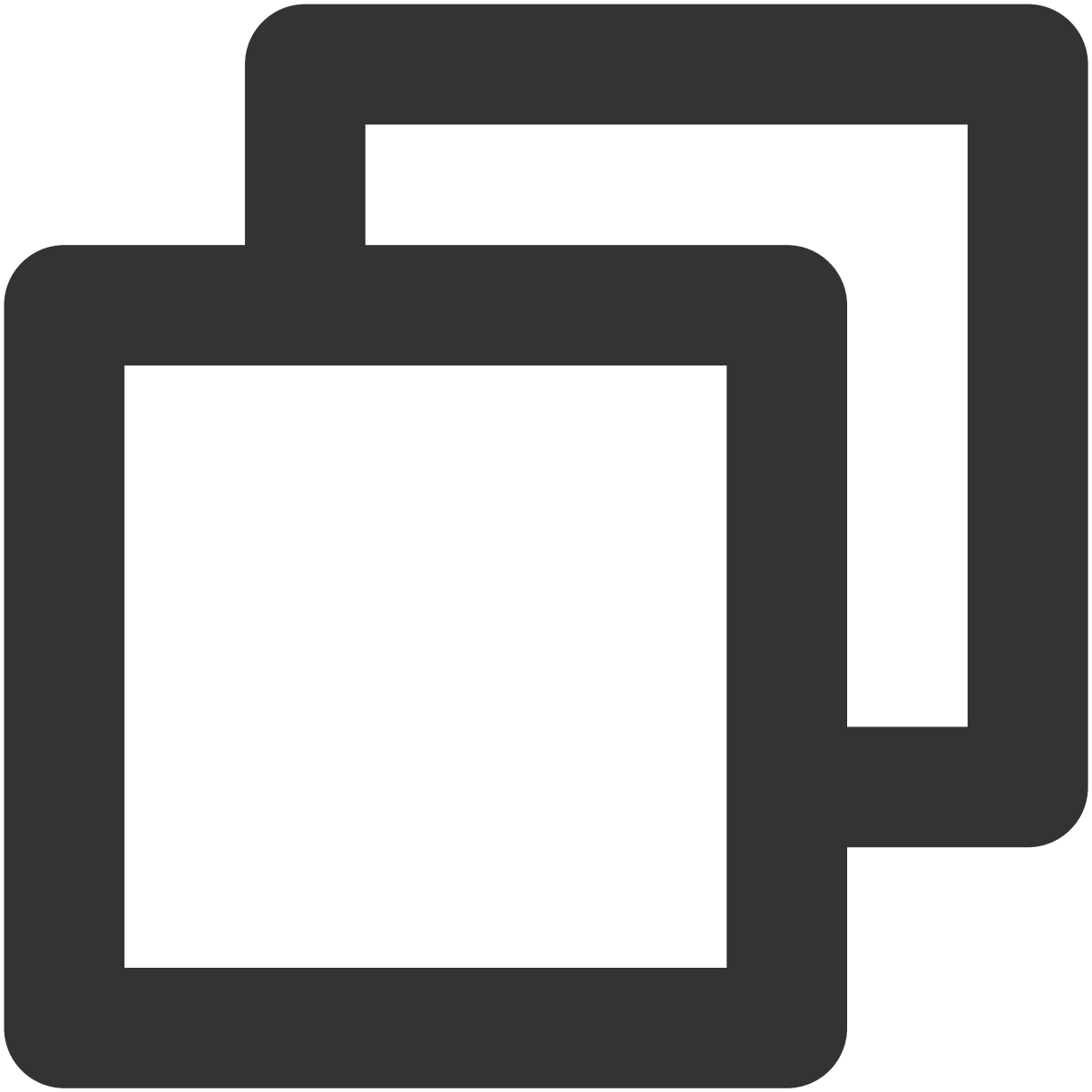
请参考此部分，将Flutter module引入您的原生应用程序中。建议采用方式二。

在 Android 项目中，管理Flutter引擎

创建一个用于管理Flutter引擎的单例对象。

这个 Kotlin 单例对象，用于集中管理 Flutter Method Channel，并提供一系列与 Flutter Module 通信的方法，方便在项目中各处，直接调用。

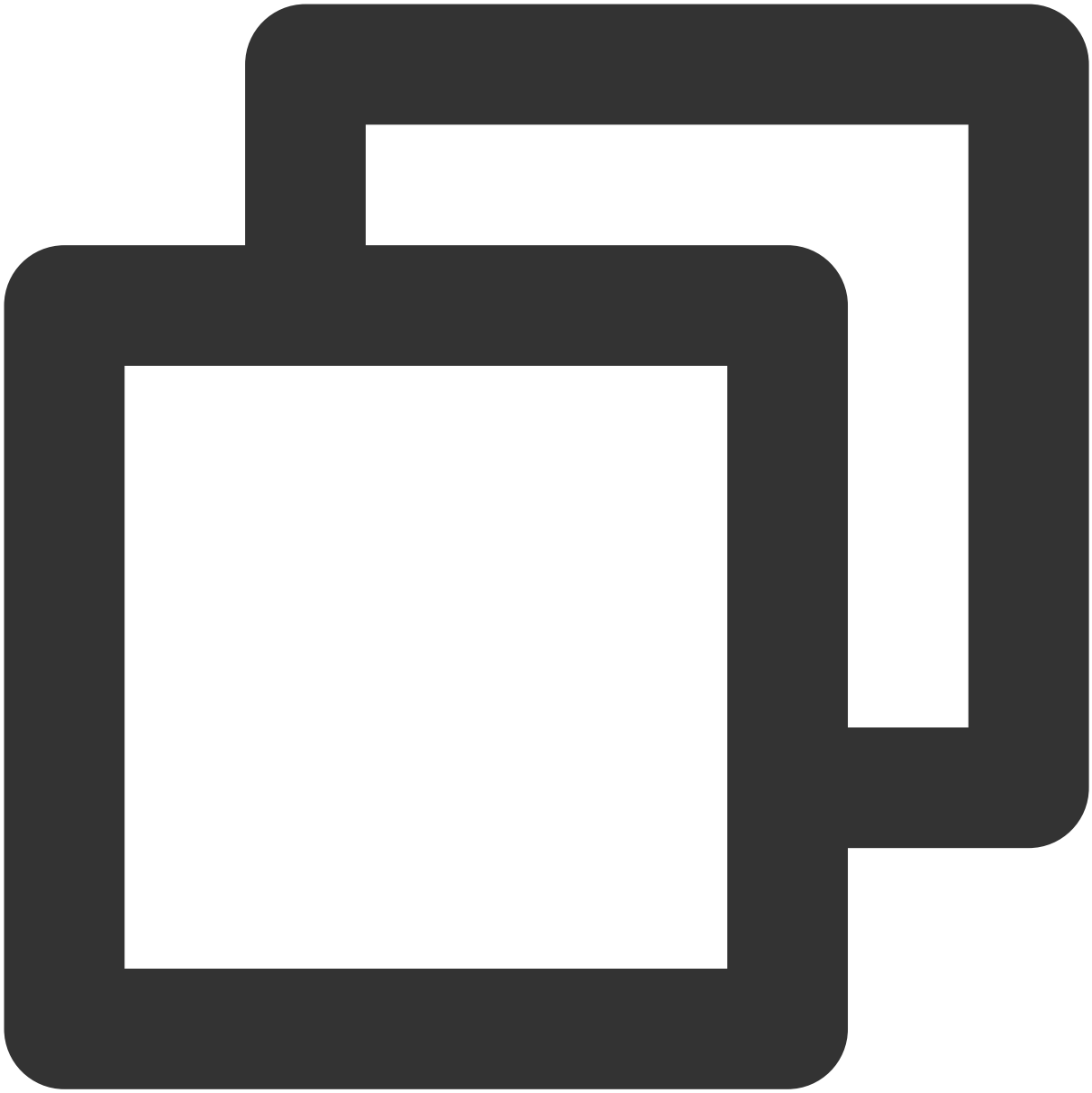
新建 `FlutterUtils.kt` 文件，并定义 `FlutterUtils` 静态类。



```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

创建一个 `FlutterEngine` （Flutter 引擎）。

在 `FlutterUtils.kt` 文件中，定义一个 `FlutterEngine` ，并在初始化时，将其初始化。



```
lateinit var context : Context
private lateinit var flutterEngine:FlutterEngine

// 初始化
flutterEngine = FlutterEngine(context)
```

继续完成该用于管理Flutter引擎的单例对象。

Demo代码的逻辑是，使用新的路由，承载Chat和Call的Activity。

Chat的Activity，由用户主动进入及退出；Call的Activity，由监听器或主动外呼，自动导航进及返回出。

重点关注：

fun init(): 初始化 Method Channel，并为其绑定事件监听方法。

fun reportChatInfo(): 将用户登录信息和SDKAPPID透传至Flutter Module，使Flutter层得以初始化并登录腾讯云IM。

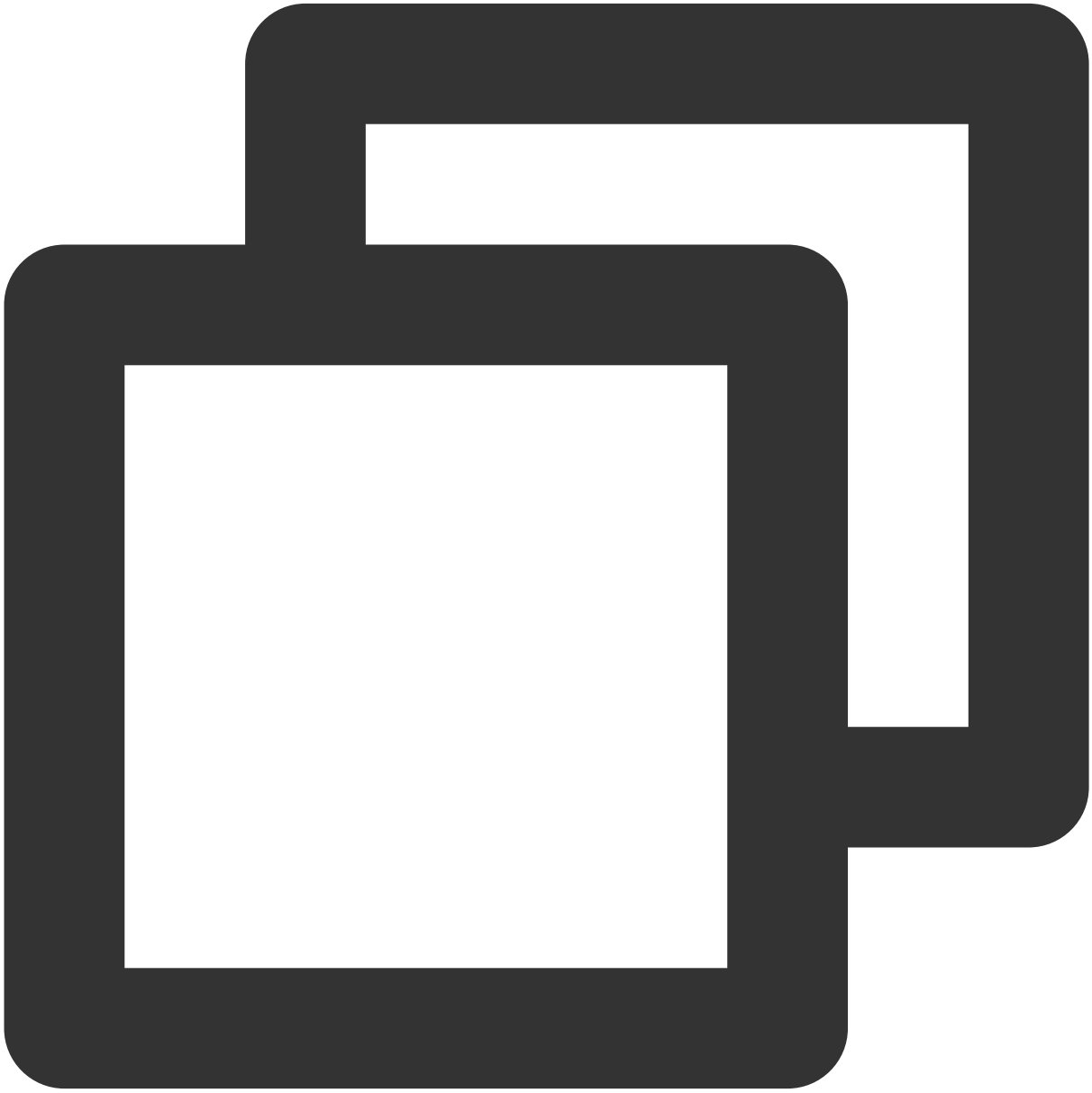
fun launchChatFunc(): 拉起或导航至 Flutter Module 所在 ViewController。

fun triggerNotification(msg: String): 将 iOS Native 层收到的离线推送消息点击事件，及其包含的ext信息，以 JSON String形式，透传至 Flutter 层绑定的监听处理事件。用于处理离线推送点击跳转，例如至对应会话。

本单例 object 的详细代码，可以参考Demo源码。

在 总入口 `MyApplication` 中，初始化上述对象。

在 `MyApplication.kt` 文件中，将全局context传入单例对象，并执行初始化。



```
class MyApplication : MultiDexApplication() {
```

```
    override fun onCreate() {
        super.onCreate()
        FlutterUtils.context = this // new
        FlutterUtils.init()         // new
    }
}
```

监听及转发离线推送点击事件

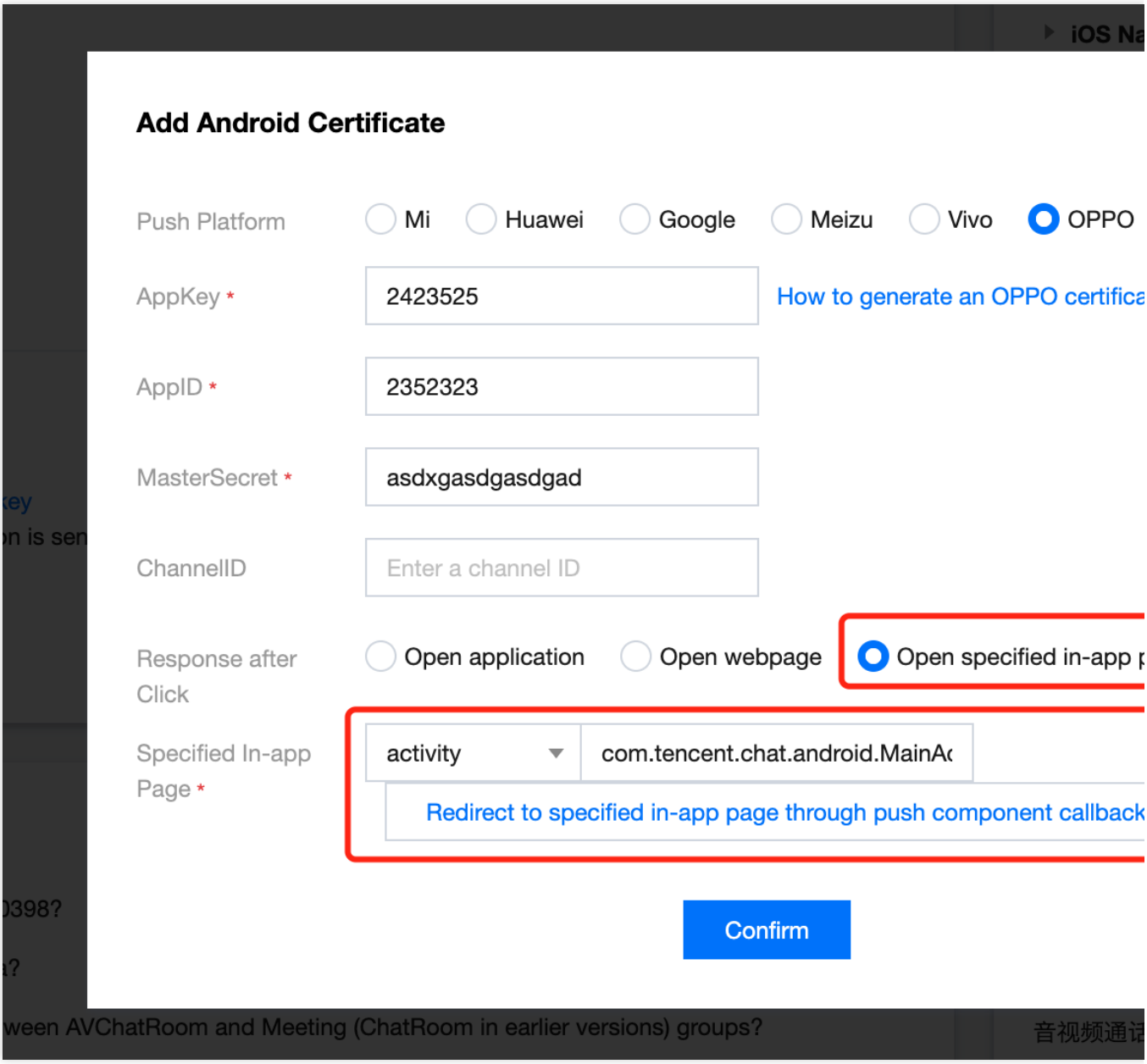
离线推送的初始化/Token上报/点击事件对应的会话跳转处理，已在Flutter Chat模块中进行，因此，Native区域，仅需透传点击通知事件的ext即可。

之所以这么做，是因为点击通知事件已在Native被拦截消费，Flutter层无法直接拿到，必须经由Native转发。

注意：

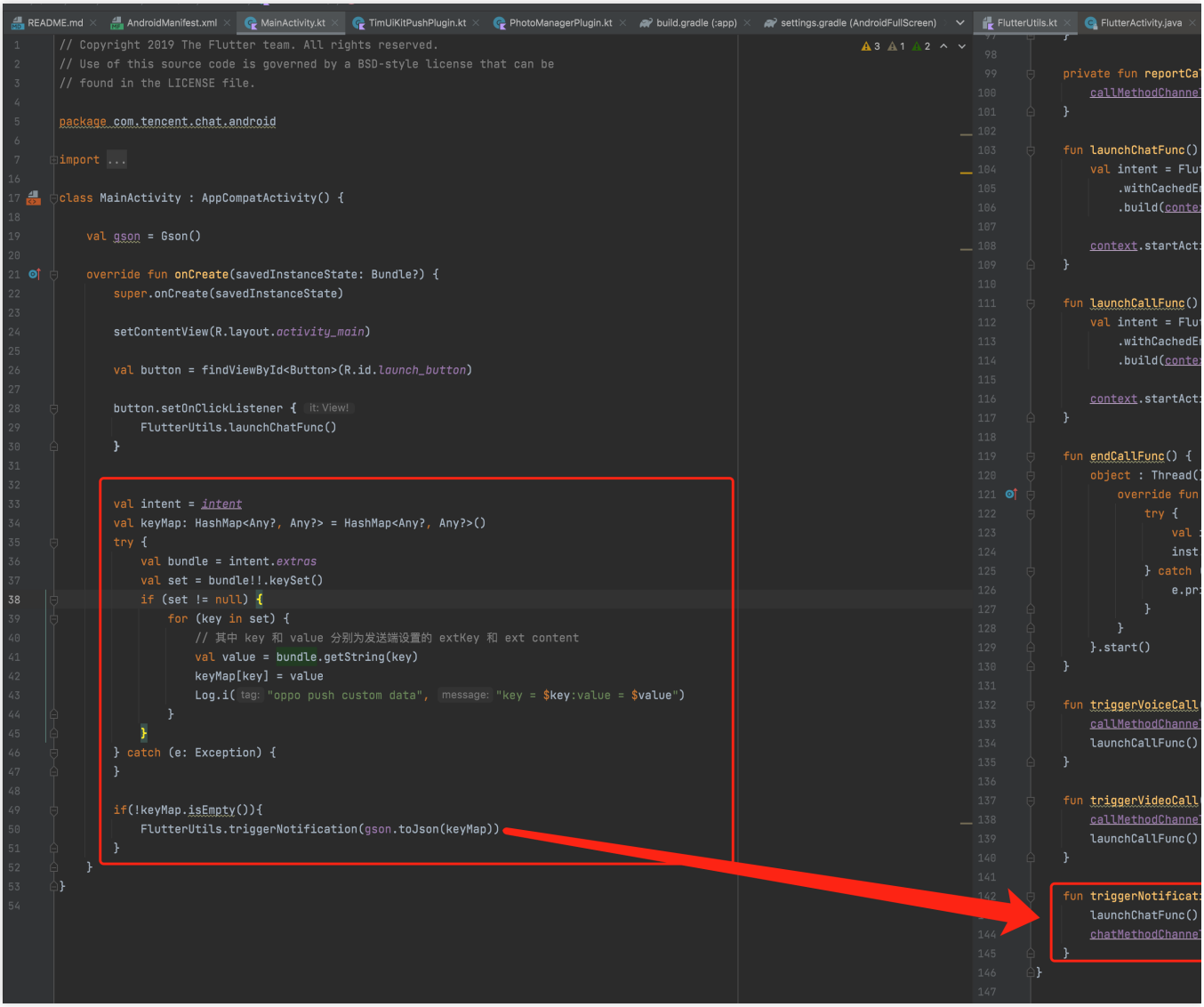
由于不同厂商的离线推送接入步骤不一致，本文以OPPO为例，全部厂商接入方案，可查看本文档.

在腾讯云IM控制台中，新增OPPO的推送证书， `点击后续动作` 选择 `打开应用内指定页面` ， `应用内页面` 以 `Activity` 方式，配置一个用于处理离线推送信息的页面，建议为应用首页。如，我们的Demo配置为： `com.tencent.chat.android.MainActivity` .



在上方控制台配置的用于离线推送的Activity文件中，新增如下代码。

该代码的作用是，当厂商拉起相应Activity时，从Bundle中取出HashMap形式ext信息，触发单例对象中的方法，将这个信息，手动转发至Flutter中。具体代码，可以参考Demo源码。



此时，Android Native层编写完成。

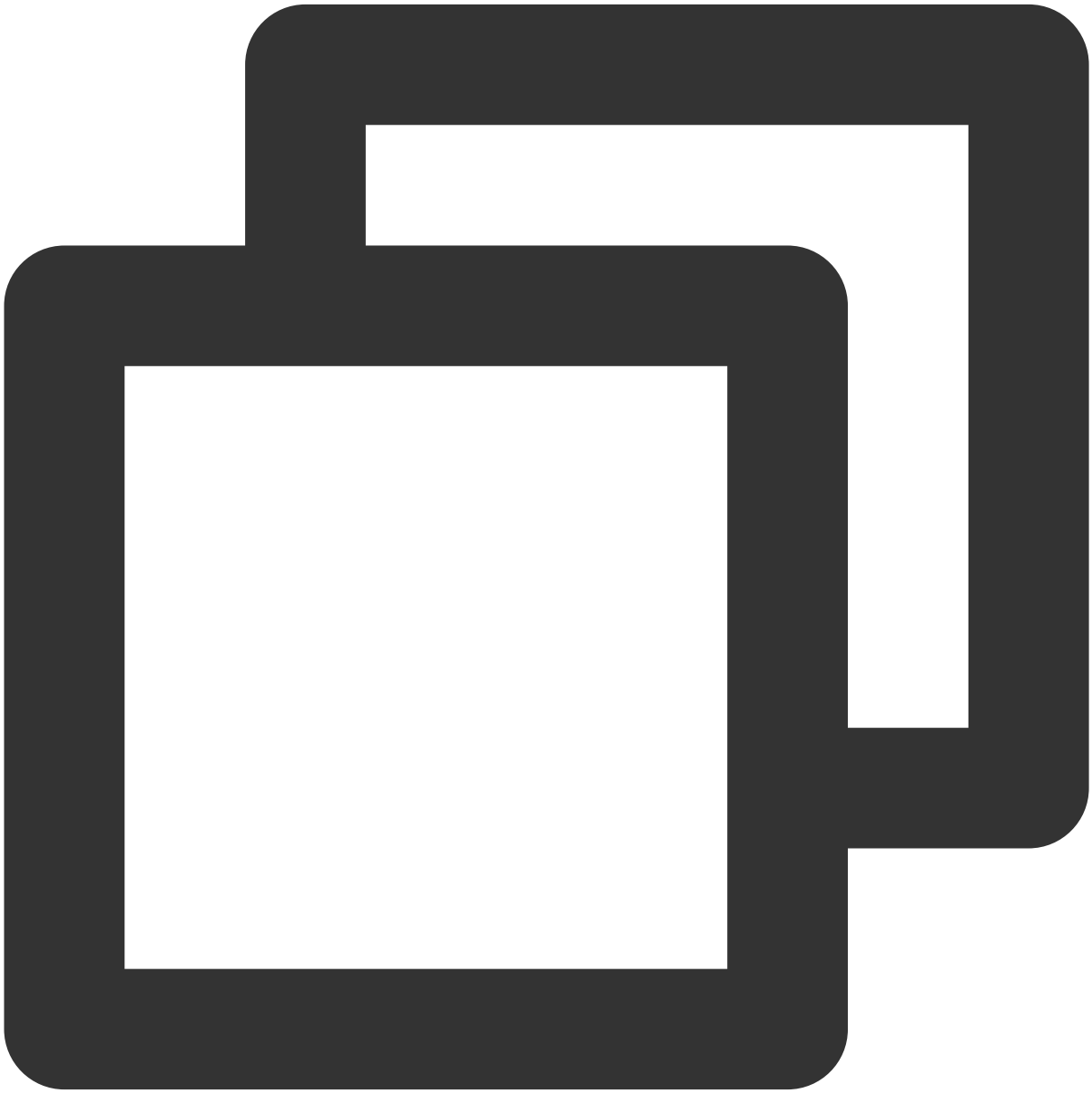
附加方案：在 Native 层，初始化并登录腾讯云IM

有的时候，对于Chat和Call模块能力，您希望对于高频的简单应用场景，能深入嵌入您现有的业务逻辑中。
例如对于游戏场景，在对局内，希望能直接发起会话。
而您的完整功能Chat模块，使用Flutter实现，仅是您APP中一个重要性较低的子模块，因此不希望一上来就启动一个完整的Flutter Module。
这个时候，您可以在Native层调用腾讯云IM Native SDK的初始化及登录方法，此后，便可在您需要的高频简单场景，直接使用腾讯云IM Native SDK，构建 In-App Chat 能力。
说明：

当然，在此种情况下，您也可以选择提前先在 Flutter 初始化并登录腾讯云IM，此时，您将不再需要在 Native 层再次初始化并登录。两端仅需初始化并登录一次，即可在双端都能使用。
由于Flutter SDK已自带Native SDK，您不需要在Native层，再次引入，即可直接使用。

Native初始化并登录

以 iOS Swift 代码为例，演示如何在 Native 层，初始化并登录。



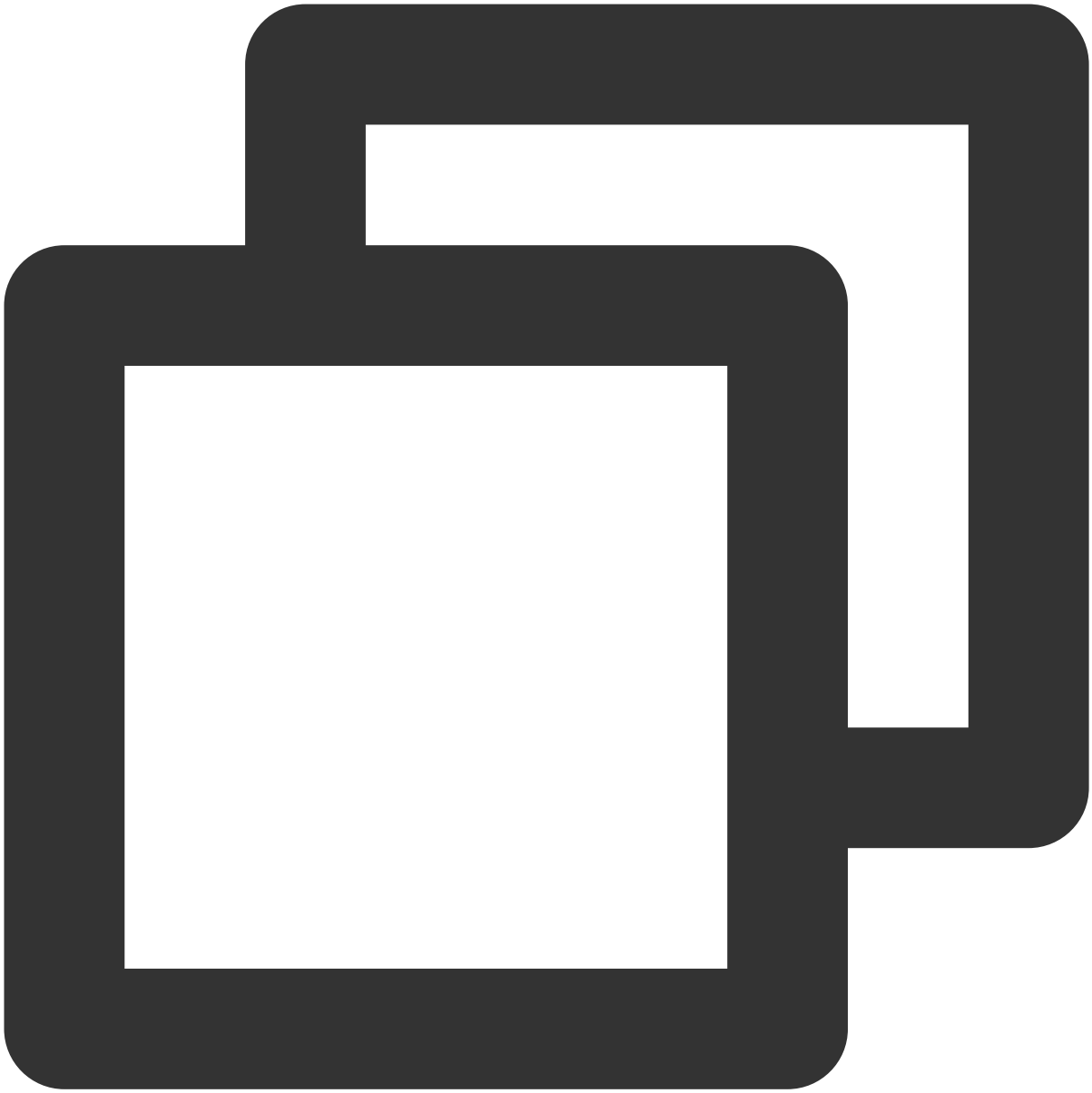
```
import ImSDK_Plus
```

```
func initTencentChat(){
        if(isLoginSuccess == true){
            return
        }
        let data = V2TIMManager.sharedInstance().initSDK( 您的SDKAPPID , config: nil
        if (data == true){
            V2TIMManager.sharedInstance().login(
                chatInfo.userID,
                userSig: chatInfo.userSig,
                succ: {
                    self.isLoginSuccess = true
                    self.reportChatInfo()
                },
                fail: onLoginFailed()
            )
        }
}
```

此后，在 Native 层面，便可直接使用Native SDK，搭建您的业务功能模块。详情可查阅 iOS 快速入门 或 Android 快速入门。

初始化 Flutter TUIKit

如果您已在 Native 层完成初始化并登录，您不需要再次在 Flutter 层再次执行，但需要调用 TUIKit的 `_coreInstance.setDataFromNative()` ，将当前用户信息传入。



```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.setDataFromNative(userId: chatInfo?.userID ?? "");
```

更详细代码，请查阅我们的Demo 源码。

至此，腾讯云IM Flutter - Native 混合开发方式已全部介绍完成。
您可以基于本文档给出的方案，快速在您现有的原生开发 Android/iOS APP 中，使用 Flutter SDK，使用同一套 Flutter代码，快速植入 Chat 和 Call 模块能力。
如果您还有任何疑问，欢迎随时联系我们。
Telegram Group
WhatsApp Group

Reference

1. Integrate a Flutter module into your Android project.
2. Integrate a Flutter module into your iOS project.
3. Adding a Flutter screen to an iOS app.
4. Multiple Flutter screens or views.