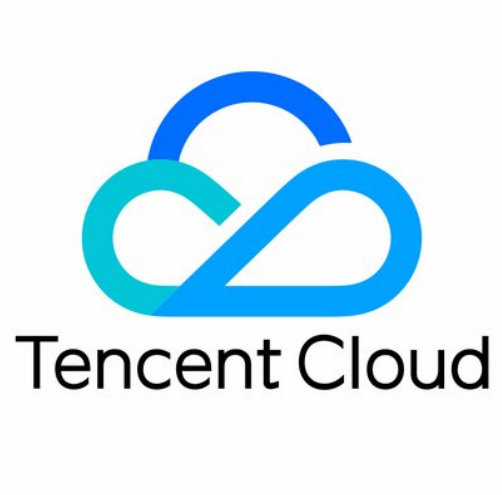
Chat

クイックスタート

製品ドキュメント



カタログ：

クイックスタート
クイックスタート（Android）

最終更新日：：2024-04-11 16:20:42

操作手順

手順1：アプリケーションの作成

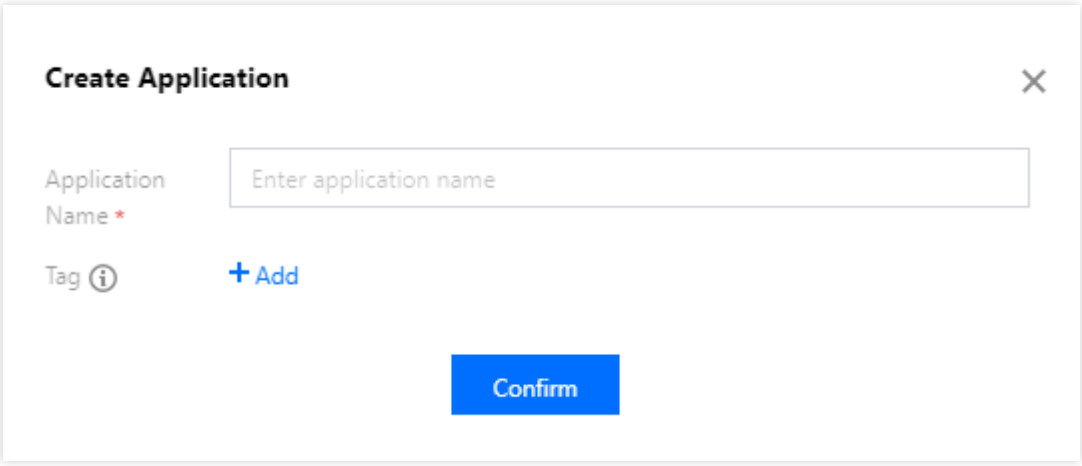
1. IMコンソールにログインします。

説明：

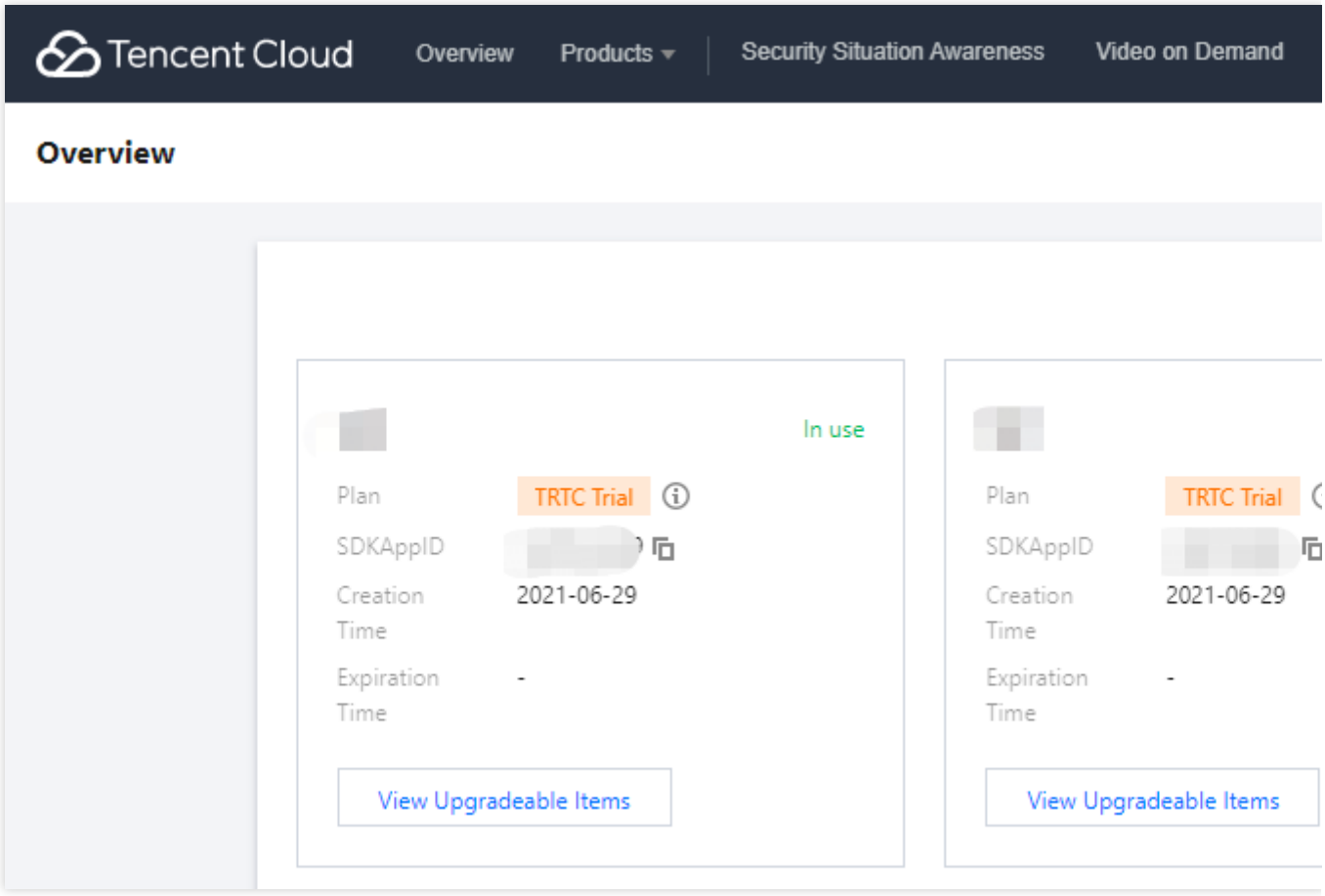
アプリケーションをすでに保有している場合は、そのSDKAppIDを記録してからキー情報の取得を実行してください。

同じTencent Cloudのアカウントで、最大300個のIMアプリケーションを作成することができます。すでにアプリケーションが300個ある場合は、使用する必要のないアプリケーションを使用停止して削除すると、新しいアプリケーションを作成することができます。**アプリケーションを削除した後、そのSDKAppIDに対応するすべてのデータとサービスは失われます。慎重に操作を行ってください。**

2. **新しいアプリケーションの作成**をクリックし、**アプリケーションの作成**のダイアログボックスにアプリケーション名を入力し、**OK**をクリックします。

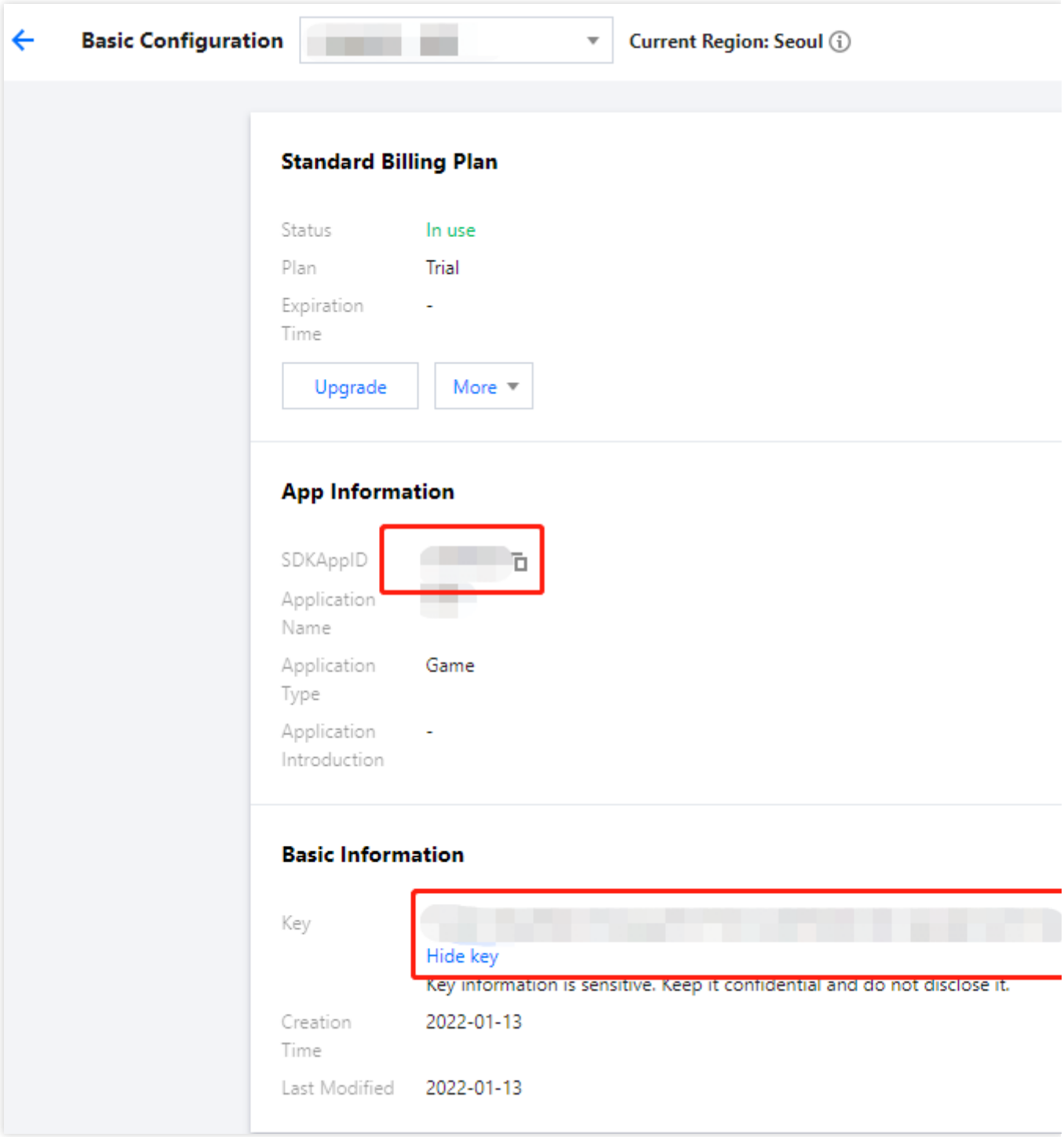


3. 作成が完了すると、コンソールの概要ページで、作成したアプリケーションのステータス、サービスバージョン、SDKAppID、作成時間、タグおよび有効期限を確認できます。SDKAppID情報を記録してください。



ステップ2：キー情報の取得

1. 対象のアプリケーションカードをクリックし、アプリケーションの基本設定画面に移動します。



2. **基本情報**セクションで、**表示キー**をクリックし、キー情報をコピーして保存します。

ご注意：

キー情報を適切に保管して、漏えいしないようにしてください。

ステップ3： Demo ソースコードのダウンロードおよび設定

1. IM Demoプロジェクトをダウンロードします。具体的なダウンロードアドレスについては、SDKダウンロードをご参照ください。

説明：

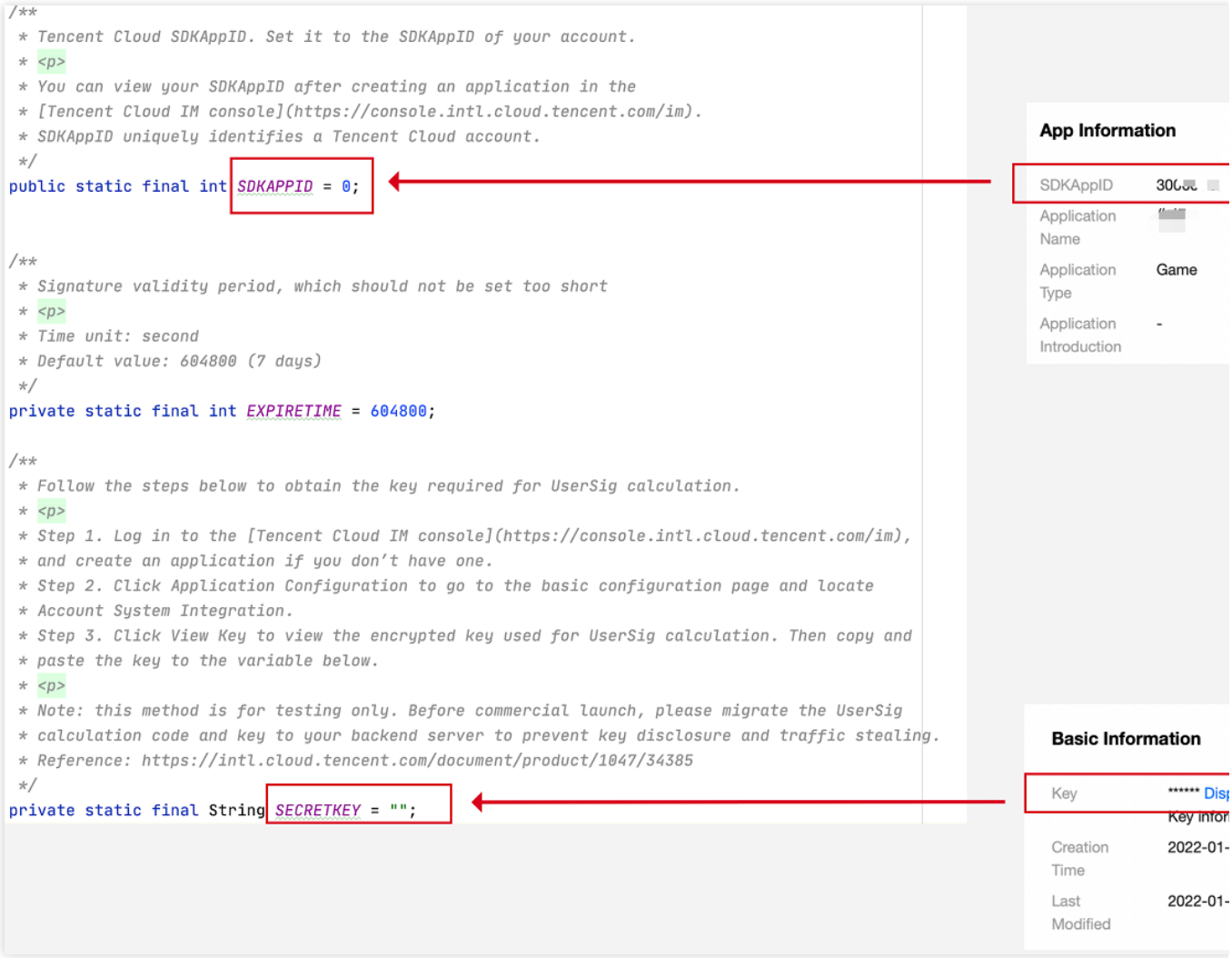
顔絵文字デザインの著作権を尊重するため、ダウンロードするDemoプロジェクトには大きな顔絵文字要素の切り取りが含まれておらず、自身のローカル顔絵文字パッケージを使用してコードを設定できます。IM Demoでの顔絵文字パッケージの不正使用は意匠権の侵害に当たる可能性があります。

2. 端末ディレクトリのプロジェクトを開き、対応する `GenerateTestUserSig` ファイルを見付けます。パスは「Android/Demo/app/src/main/java/com/tencent/qcloud/tim/demo/signature/GenerateTestUserSig.java」です。

3. `GenerateTestUserSig` ファイルの関連パラメータを設定します：

SDKAPPID：手順1で取得した実際のアプリケーションSDKAppIDを設定してください。

SECRETKEY： 手順2で取得した実際のキー情報を設定してください。



ご注意：

ここで言及するUserSigの取得方法は、クライアントコードにSECRETKEYを設定しますが、この手法のSECRETKEYは逆コンパイルによって逆クラッキングされやすく、キーがいったん漏洩すると、攻撃者はTencent Cloudトラフィックを盗用できるようになります。そのため**この手法は、ローカルのDemoクイックスタートおよび機能デバッグにのみ適しています**。

正しい UserSigの発行方法は、UserSig の計算コードをお客様のサーバーに統合して、App向けのポートを用意

し、UserSig を必要とするときは、App から業務サーバーにリクエストを出して、ダイナミック UserSigを取得することです。より詳細な内容については、 サーバーでのUserSig生成をご参照ください。

手順4：コンパイルと実行

Android Studioを用いてプロジェクトをインポートし、直接コンパイルして実行します。
詳細については、ステップ3 でクローンしたDemoプロジェクトの対応ディレクトリにある `README.md` ファイルをご参照ください。

開発環境要件

Android Studio-Chipmunk

Gradle-6.7.1

Android Gradle Plugin Version-4.2.0

kotlin-gradle-plugin-1.5.31

ご注意：

Demoデフォルトでオーディオビデオ通話機能が統合されています。この機能が依存するAVのSDKは現在、シミュレーターをサポートしていないため、実際のマシンを使用してデバッグするかまたはDemoを実行してください。

クイックスタート（iOS）

最終更新日：：2024-04-11 16:20:42

ここでは、主にTencent Cloud IM Demo(iOS)を素早く実行する方法について説明します。

操作手順

手順1：アプリケーションの作成

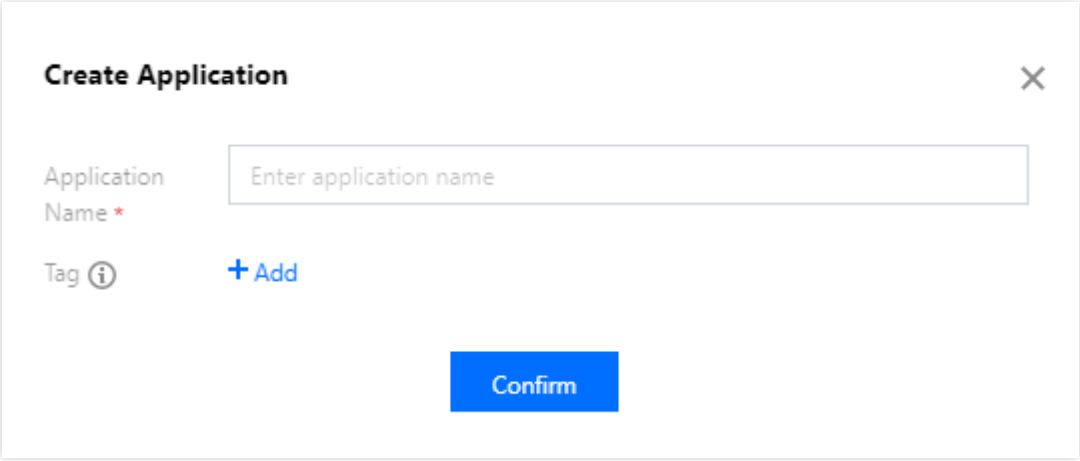
1. IMコンソールにログインします。

説明：

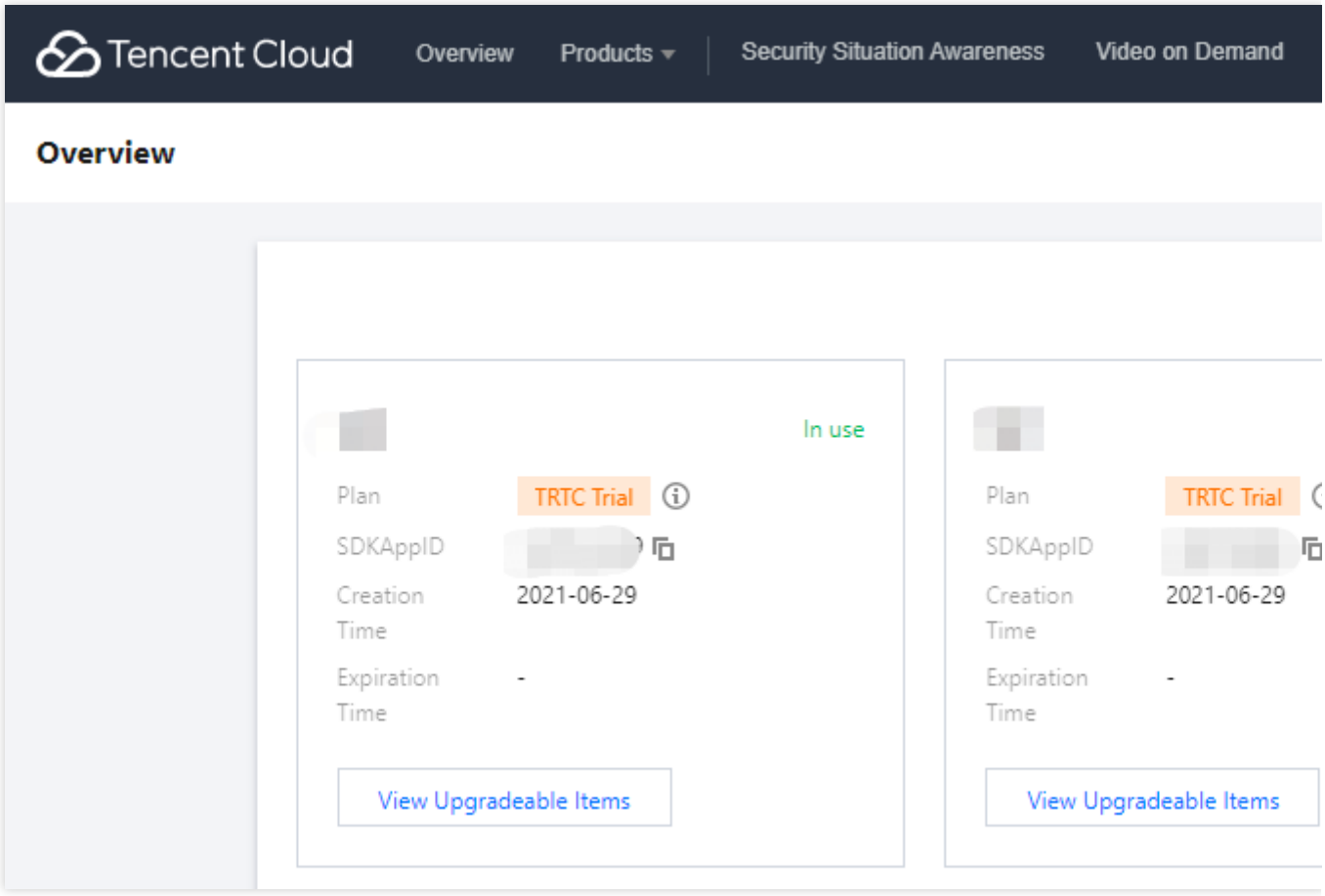
アプリケーションをすでに所有している場合は、そのSDKAppIDを記録してからキー情報の取得を実行してください。

同じTencent Cloudのアカウントで、最大300個のIMアプリケーションを作成することができます。すでにアプリケーションが300個ある場合は、使用する必要のないアプリケーションを使用停止して削除すると、新しいアプリケーションを作成することができます。**アプリケーションを削除した後、そのSDKAppIDに対応するすべてのデータとサービスは失われます。慎重に操作を行ってください。**

2. **新しいアプリケーションの作成**をクリックし、**アプリケーションの作成**のダイアログボックスにアプリケーション名を入力し、**OK**をクリックします。

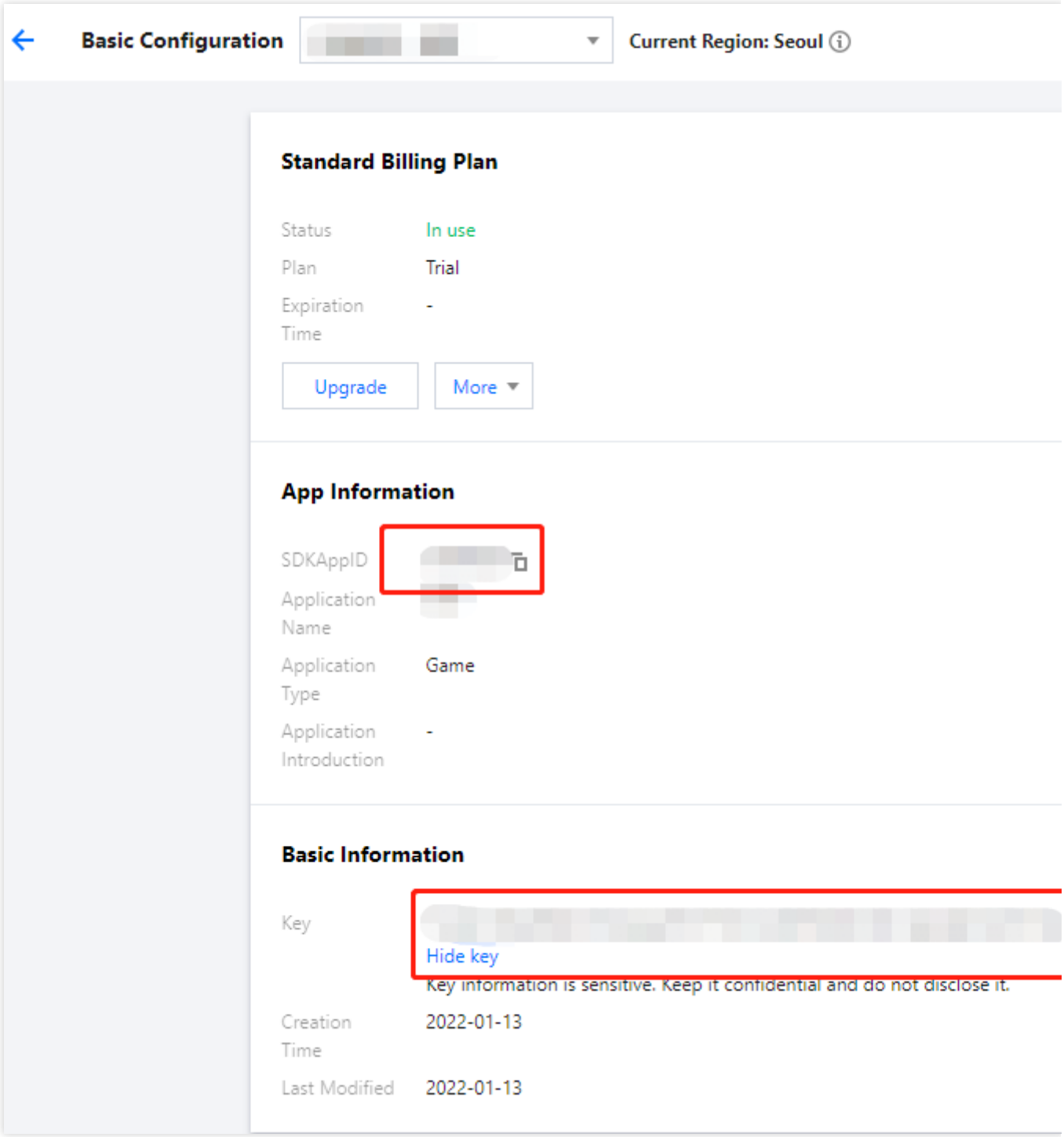


3. 作成が完了すると、コンソールの概要ページで、作成したアプリケーションのステータス、サービスバージョン、SDKAppID、作成時間、タグおよび有効期限を確認できます。SDKAppID情報を記録してください。



ステップ2：キー情報の取得

1. 対象のアプリケーションカードをクリックし、アプリケーションの基本設定画面に移動します。



2. **基本情報**セクションで、**表示キー**をクリックし、キー情報をコピーして保存します。

ご注意：

キー情報を適切に保管して、漏洩しないようにしてください。

ステップ3： Demo ソースコードのダウンロードおよび設定

1. IM Demoプロジェクトをダウンロードします。具体的なダウンロードアドレスについては、SDKダウンロードをご参照ください。

説明：

顔絵文字デザインの著作権を尊重するため、ダウンロードするDemoプロジェクトには大きな顔絵文字要素の切り取りが含まれておらず、自身のローカル顔絵文字パッケージを使用してコードを設定できます。IM Demoでの顔絵文字パッケージの不正使用は意匠権の侵害に当たる場合があります。

2. 所属する端末ディレクトリのプロジェクトを開き、対応する `GenerateTestUserSig` ファイルを見付けます。

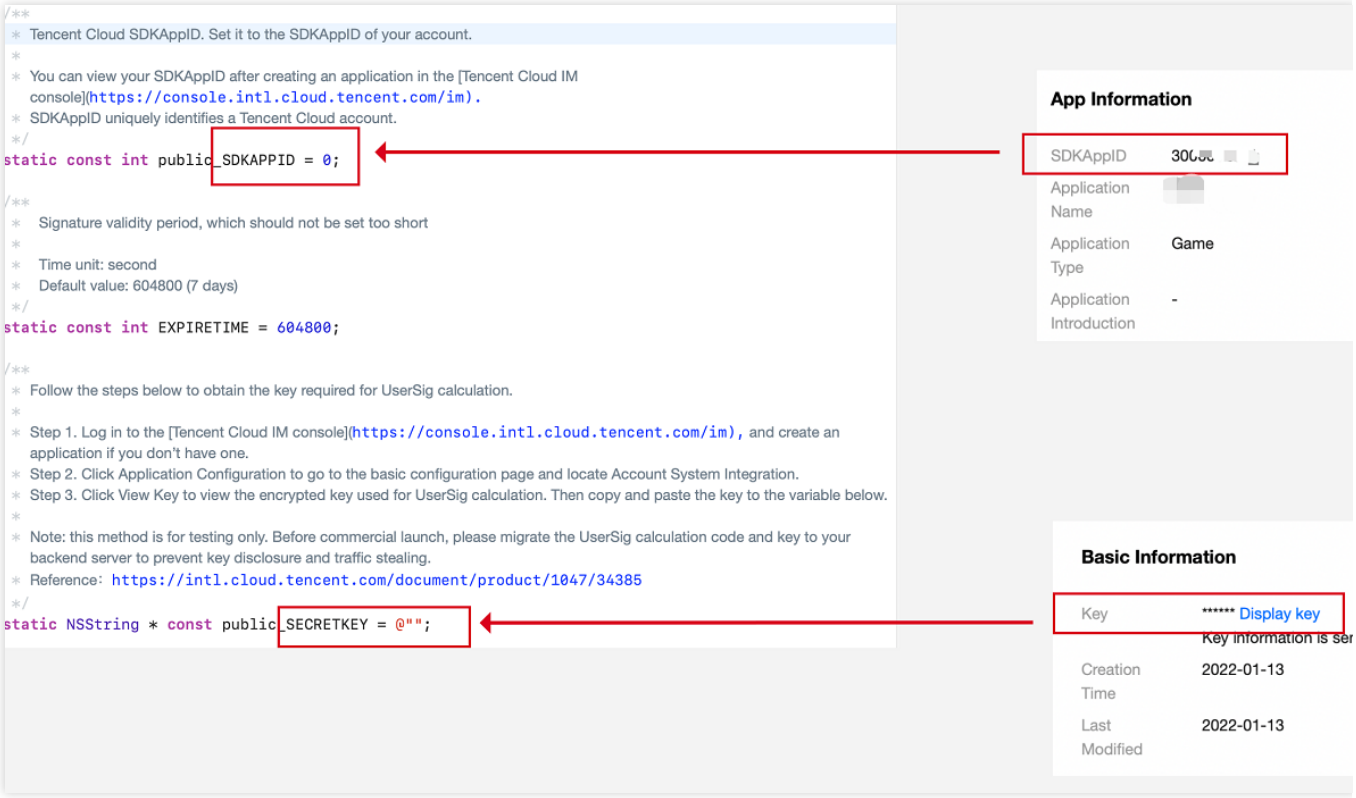
iOS パス：iOS/Demo/TUIKitDemo/Private/GenerateTestUserSig.h

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

3. `GenerateTestUserSig` ファイルの関連パラメータを設定します。

SDKAPPID：手順1で取得した実際のアプリケーションSDKAppIDを設定してください。

SECRETKEY： ステップ2で取得した実際のキー情報を設定してください。



ご注意：

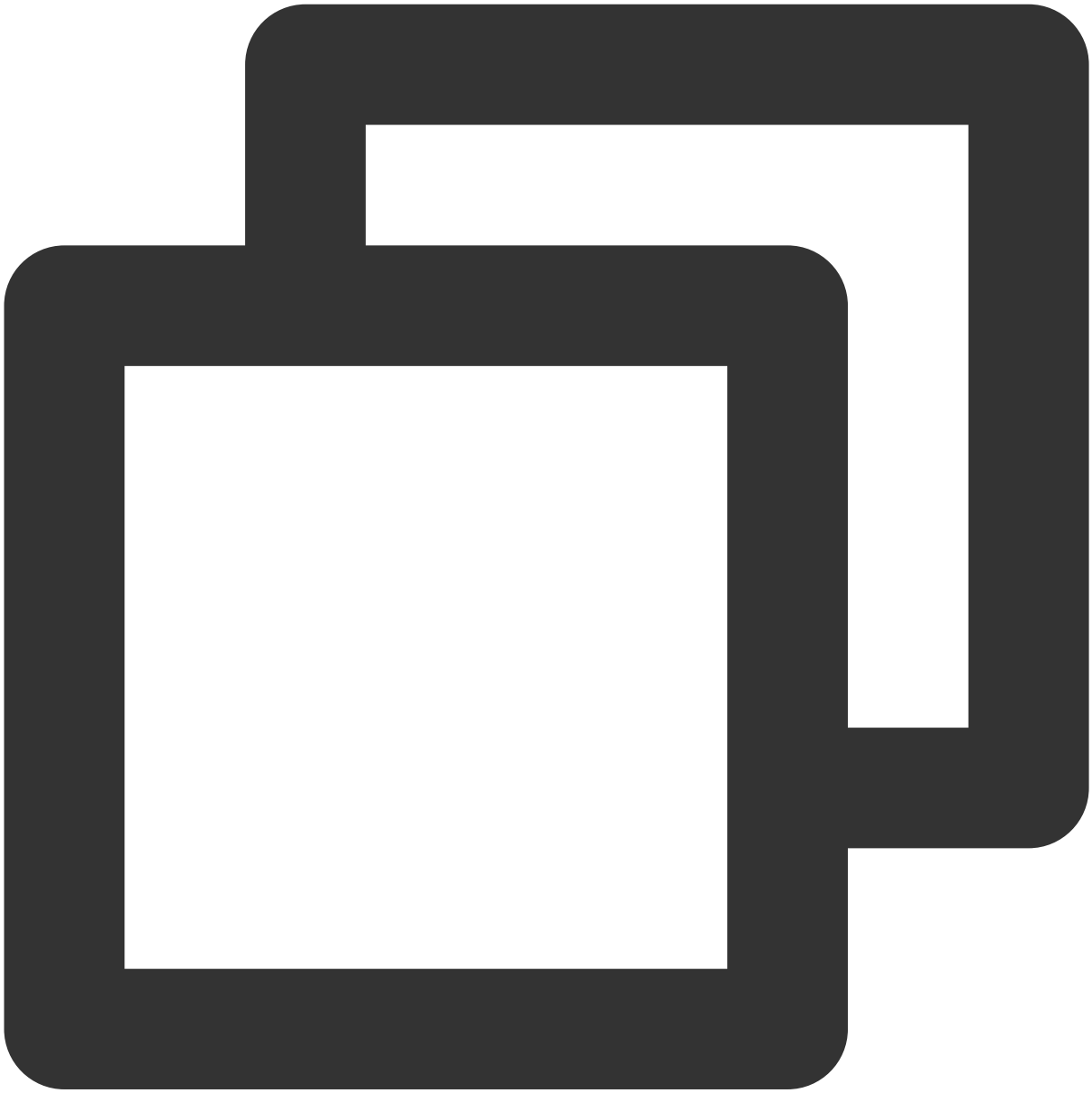
ここで言及するUserSigの取得方法は、クライアントコードにSECRETKEYを設定しますが、この手法のSECRETKEYは逆コンパイルによって逆クラッキングされやすく、キーがいったん漏洩すると、攻撃者はTencent Cloudトラフィックを盗用できるようになります。そのため**この手法は、ローカルのDemoクイックスタートおよび機能デバッグにのみ適しています**。

UserSigの正しい発行方法は、UserSigの計算コードをサーバーに統合し、Appのインターフェース向けに提供します。 UserSigが必要なときは、Appから業務サーバーにリクエストを送信してダイナミックUserSigを取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

手順4：コンパイル実行

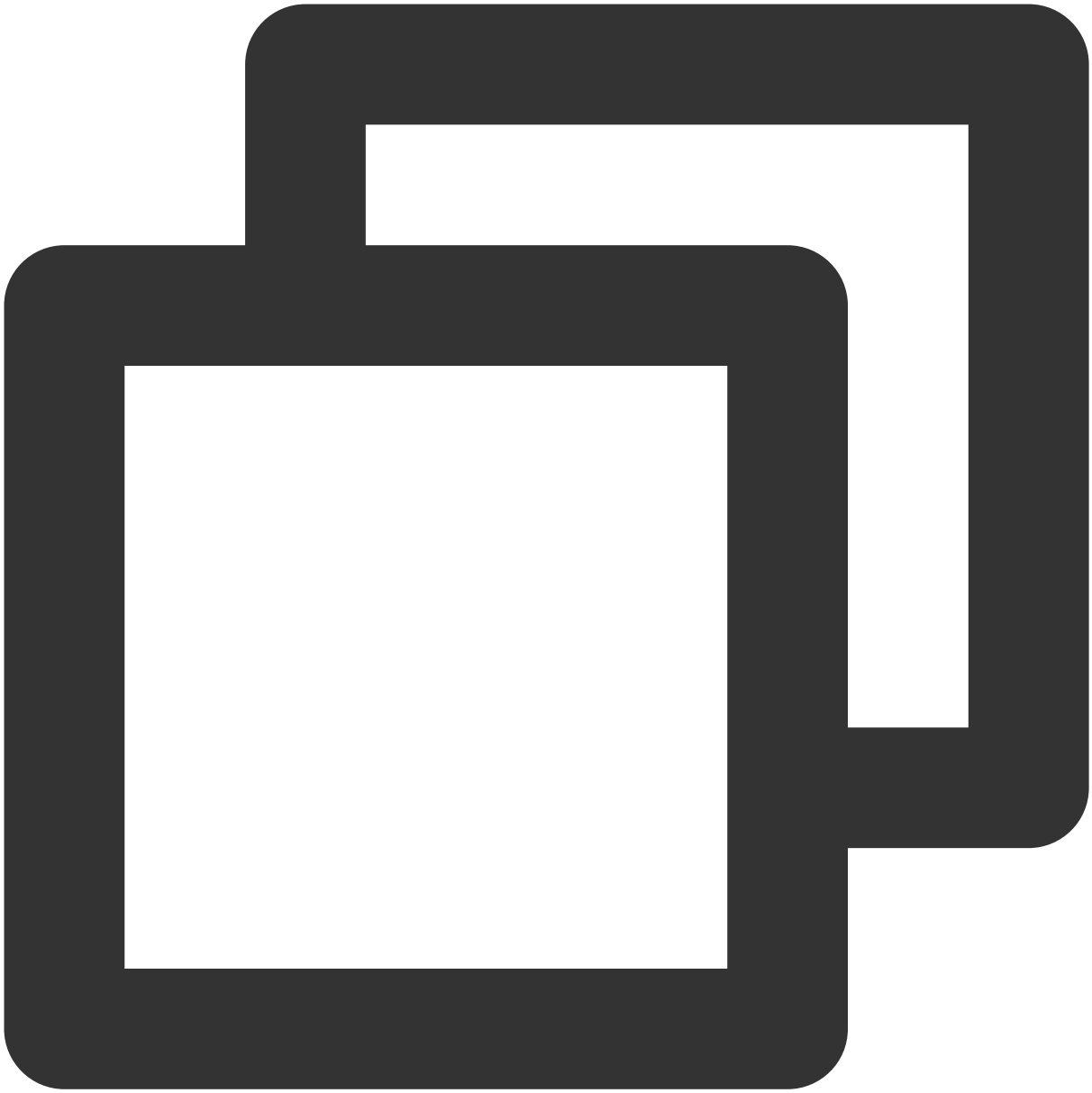
ステップ3 でクローンしたDemoプロジェクトの対応ディレクトリにある `README.md` ファイルをご参照ください。

1. 端末で次のコマンドを実行して、podのバージョンをチェックします。



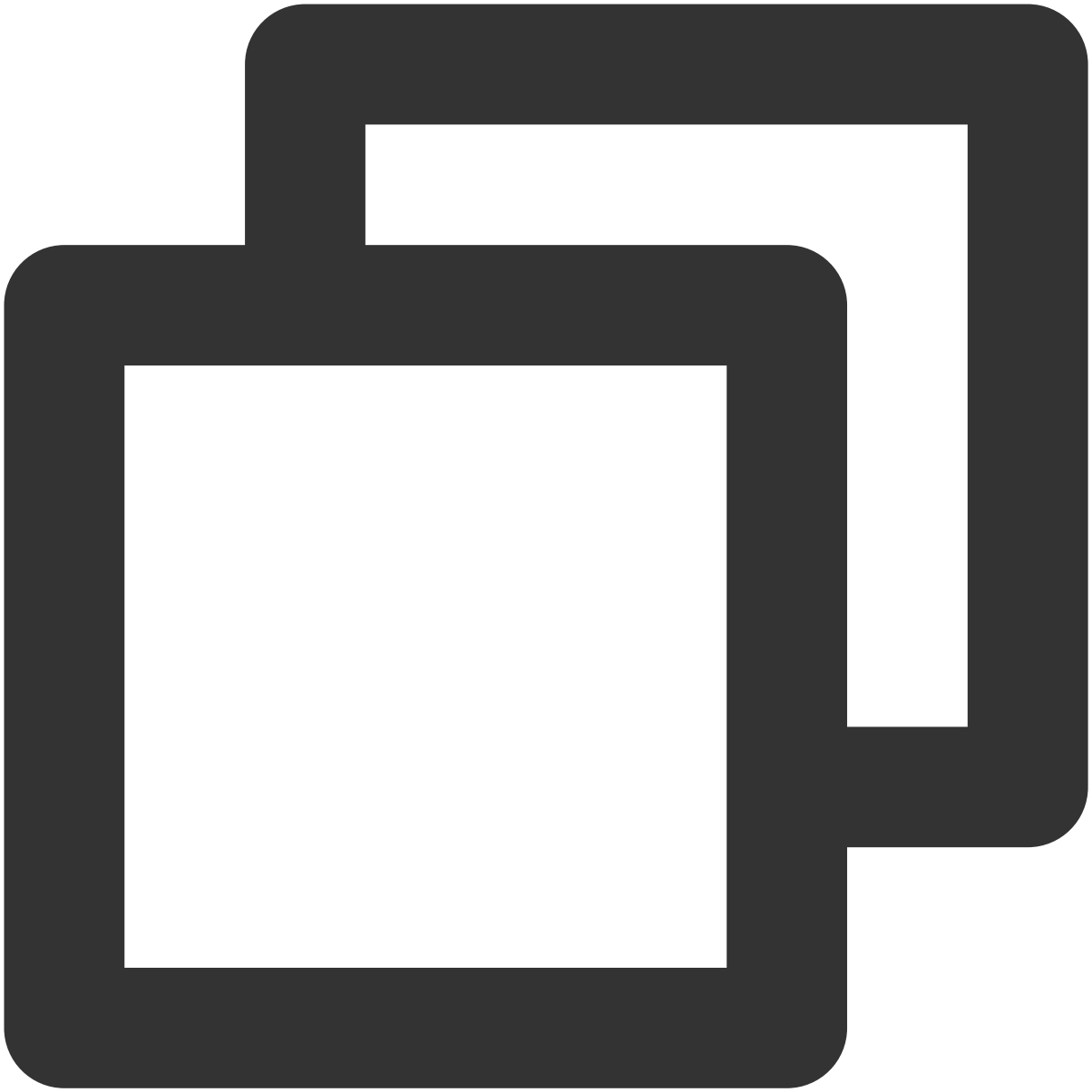
```
pod --version
```

podが存在しない、またはpodのバージョンが1.7.5未満であるというプロンプトが表示された場合は、次のコマンドを実行して最新のpodをインストールします。



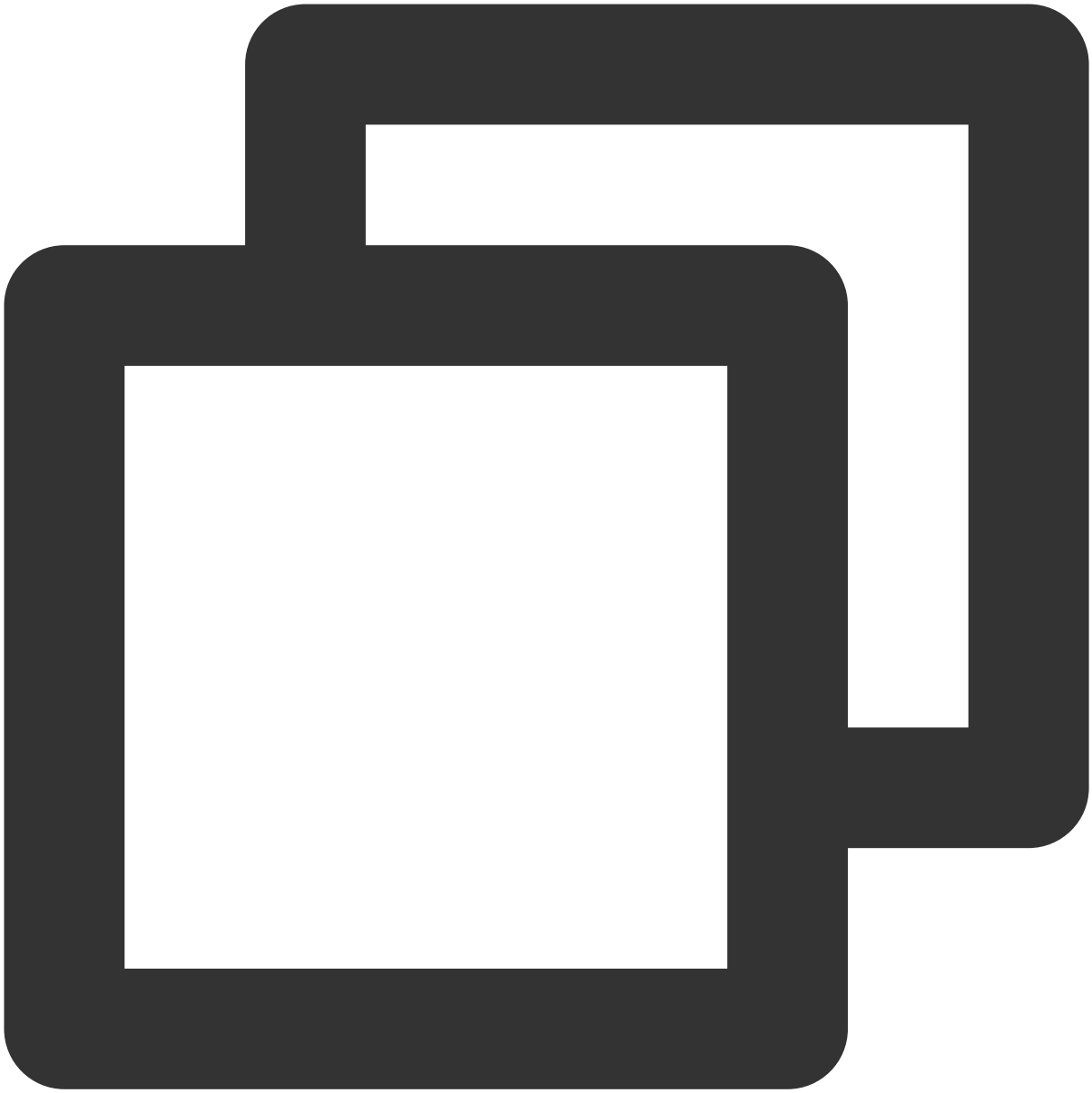
```
//gemソースの変更
gem sources --remove https://rubygems.org/
gem sources --add https://gems.ruby-china.com/
//podのインストール
sudo gem install cocoapods -n /usr/local/bin
//複数のXcodeをインストールする場合は、次のコマンドを使用してXcodeのバージョンを選択してください
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer
//podローカルライブラリの更新
pod setup
```

2. 端末で次のコマンドを実行して、依存ライブラリをインストールします。



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

インストールに失敗した場合は、次のコマンドを実行してローカルのCocoaPodsリポジトリリストを更新します。



```
pod repo update
```

3. コンパイルして実行します。

iOSでiOS/TUIKitDemoフォルダに移動し、 `TUIKitDemo.xcworkspace` を開き、コンパイルして実行します。

Macで入Mac/TUIKitDemoフォルダに移動し、 `TUIKitDemo.xcworkspace` を開き、コンパイルして実行します。

ご注意：

Demoデフォルトでオーディオビデオ通話機能が統合されています。この機能が依存するAVのSDKは現在、シミュレーターをサポートしていないため、実際のマシンを使用してデバッグするかまたはDemoを実行してください。

高度な機能

UIライブラリ
ビデオ通話の起動

関連ドキュメント

料金説明

クイックスタート（Web）

最終更新日：：2024-04-11 16:20:42

Chat UIKit React

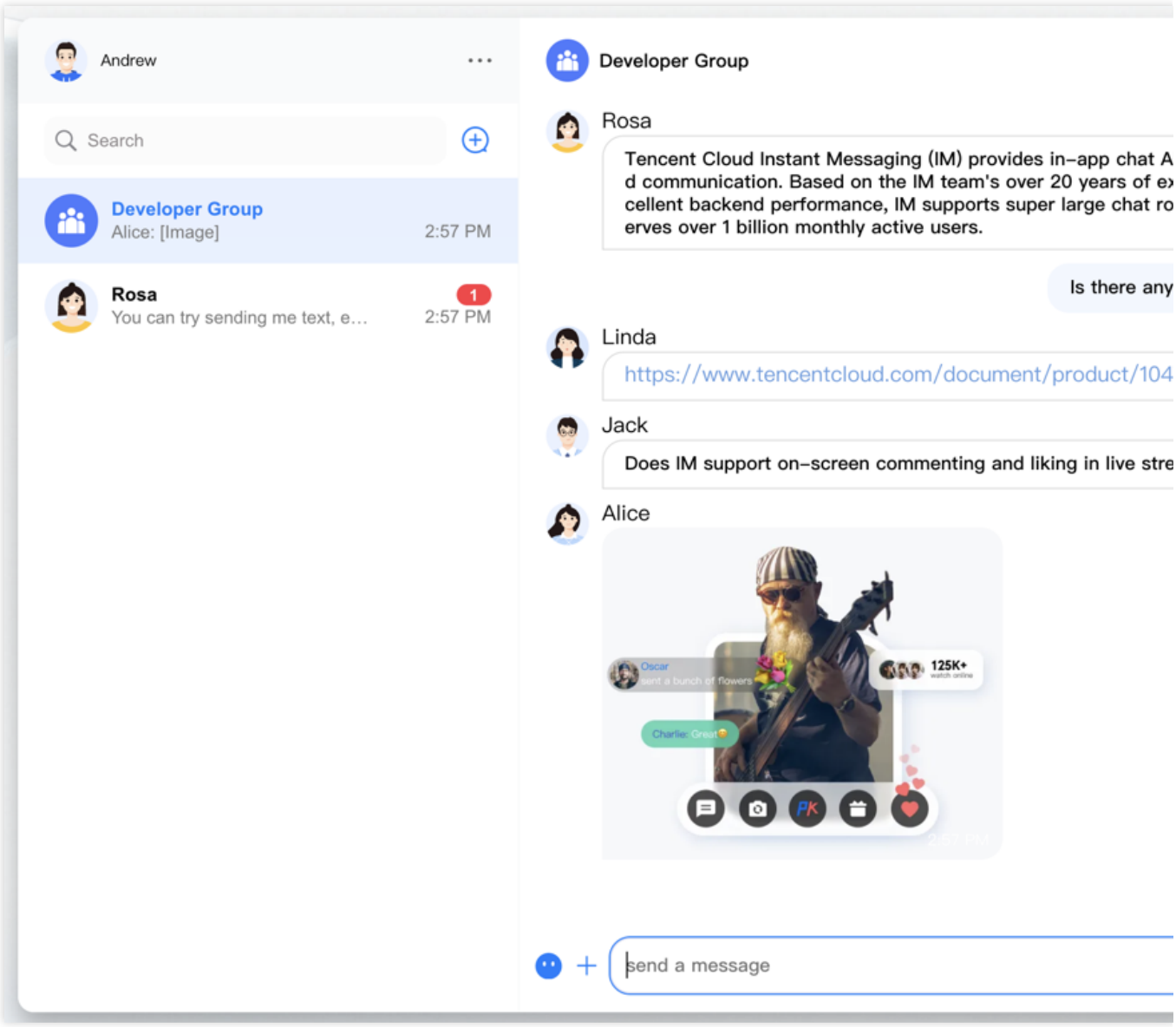
説明：

Chat UIKitはTencent Cloud IM SDKをベースにしたUIコンポーネントライブラリであり、汎用UIコンポーネント及び会話、チャット、リレーションシップチェーン、グループ、オーディオビデオ通話などの機能を提供します。UIコンポーネントを使用すれば、自分の業務ロジックを積み木を組み合わせるように素早く構築できます。

Chat UIKitが提供したコンポーネントは、UI機能を実装すると同時に、IM SDKに応じてインスタンスを呼出して、IMに関するロジックとデータ処理も実装します。そのため、開発者がChat UIKitを使用する時、自分の業務とカスタム拡張だけに集中するだけです。

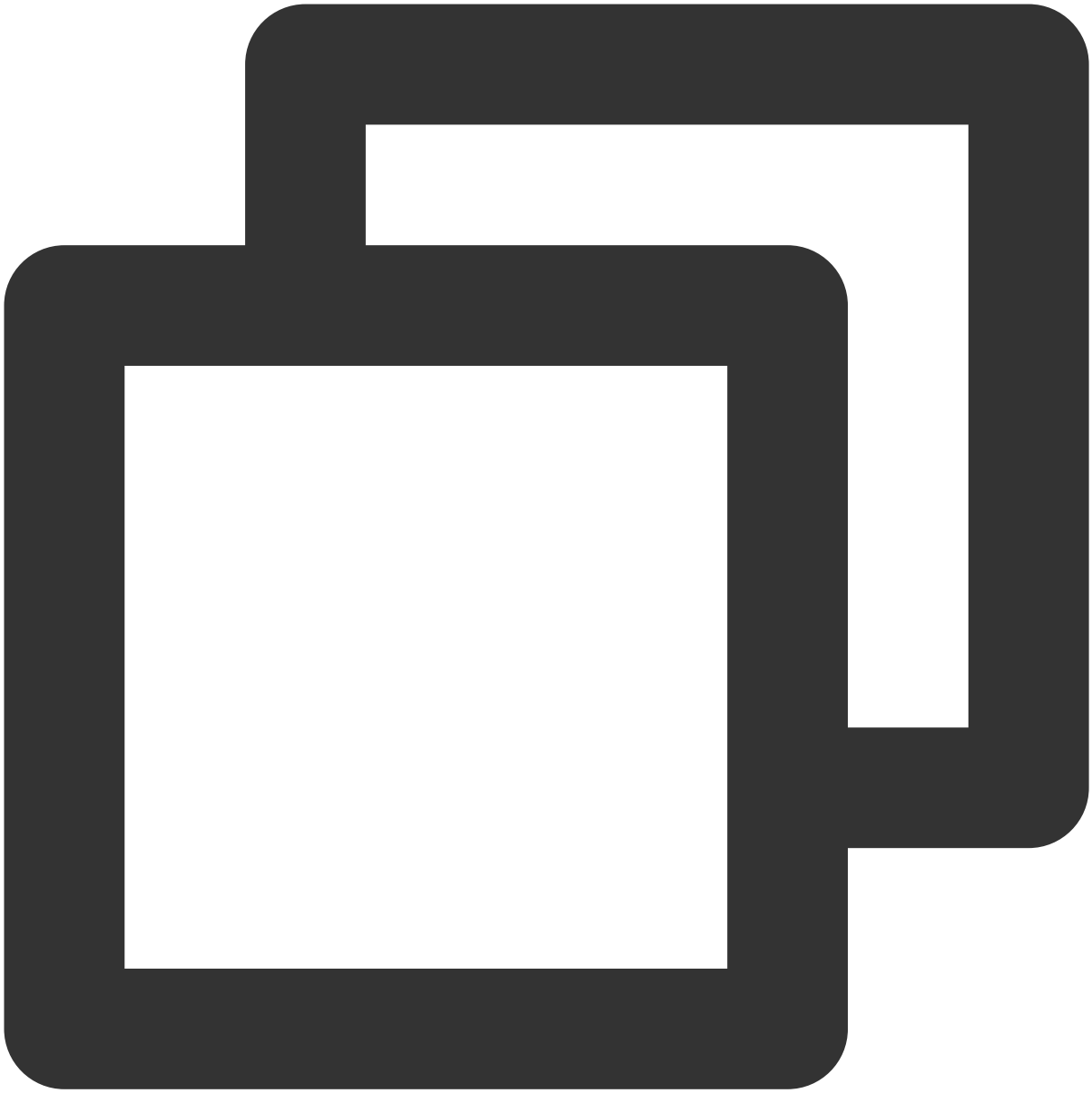
Example App

チャット機能をデモするプログラムをご用意しておりますので、公式サイトでこれらのdemoを参照できます。また、GitHubで関連のオープンソースコードも公開しています。



Demoのクイックスタート

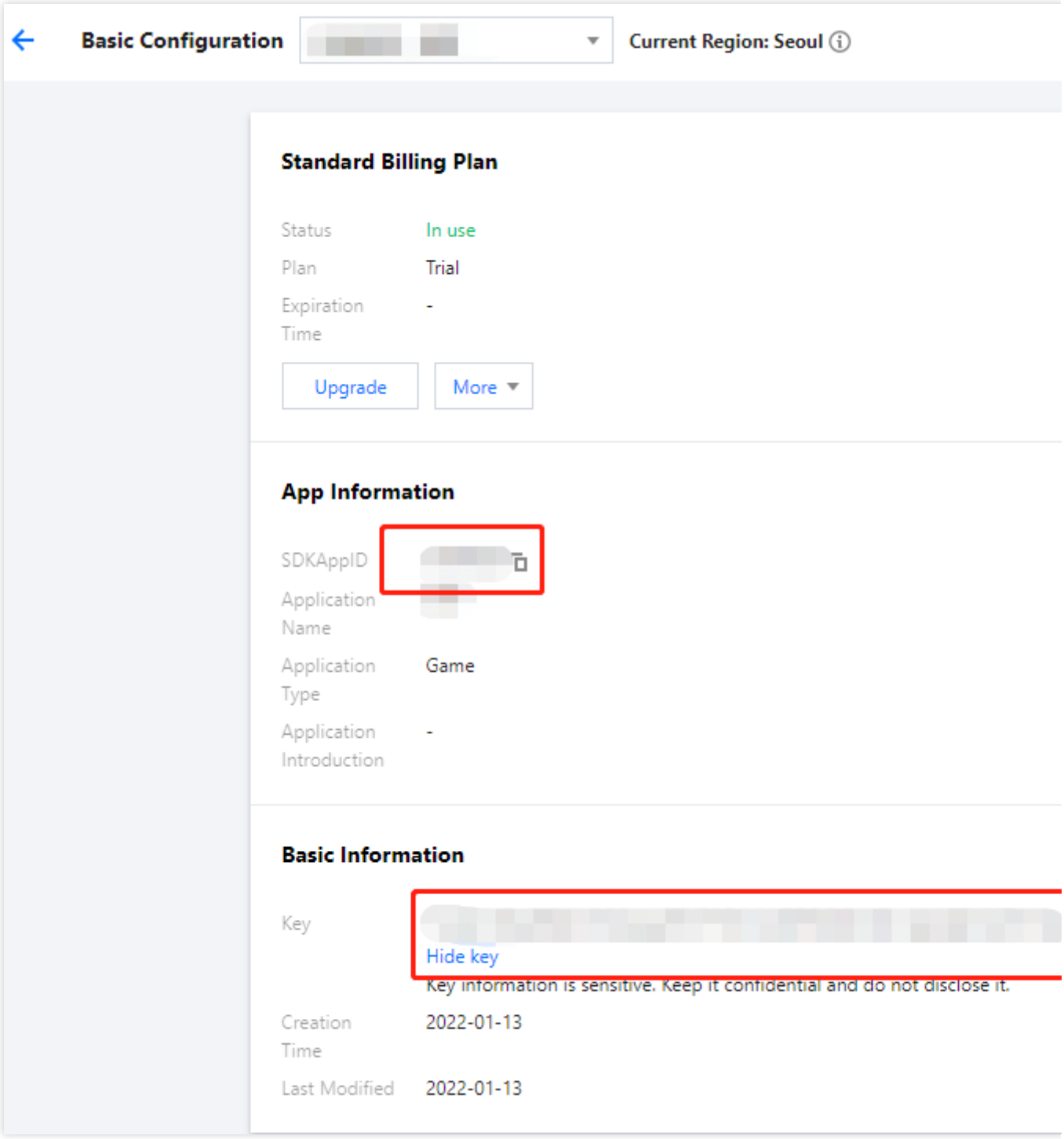
ステップ1：ソースコードのダウンロード



```
# 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
```

ステップ2：demoの設定

1. `examples/sample-chat` プロジェクトを開き、 `./examples/sample-chat/src/debug/GenerateTestUserSig.js` パスで `GenerateTestUserSig.js` ファイルに進みます。
2. `GenerateTestUserSig.js` ファイルで `SDKAPPID` と `SECRETKEY` を設定します。具体的な値はIMコンソールから取得してください。対象アプリケーションカードをクリックし、設定ページに進みます。

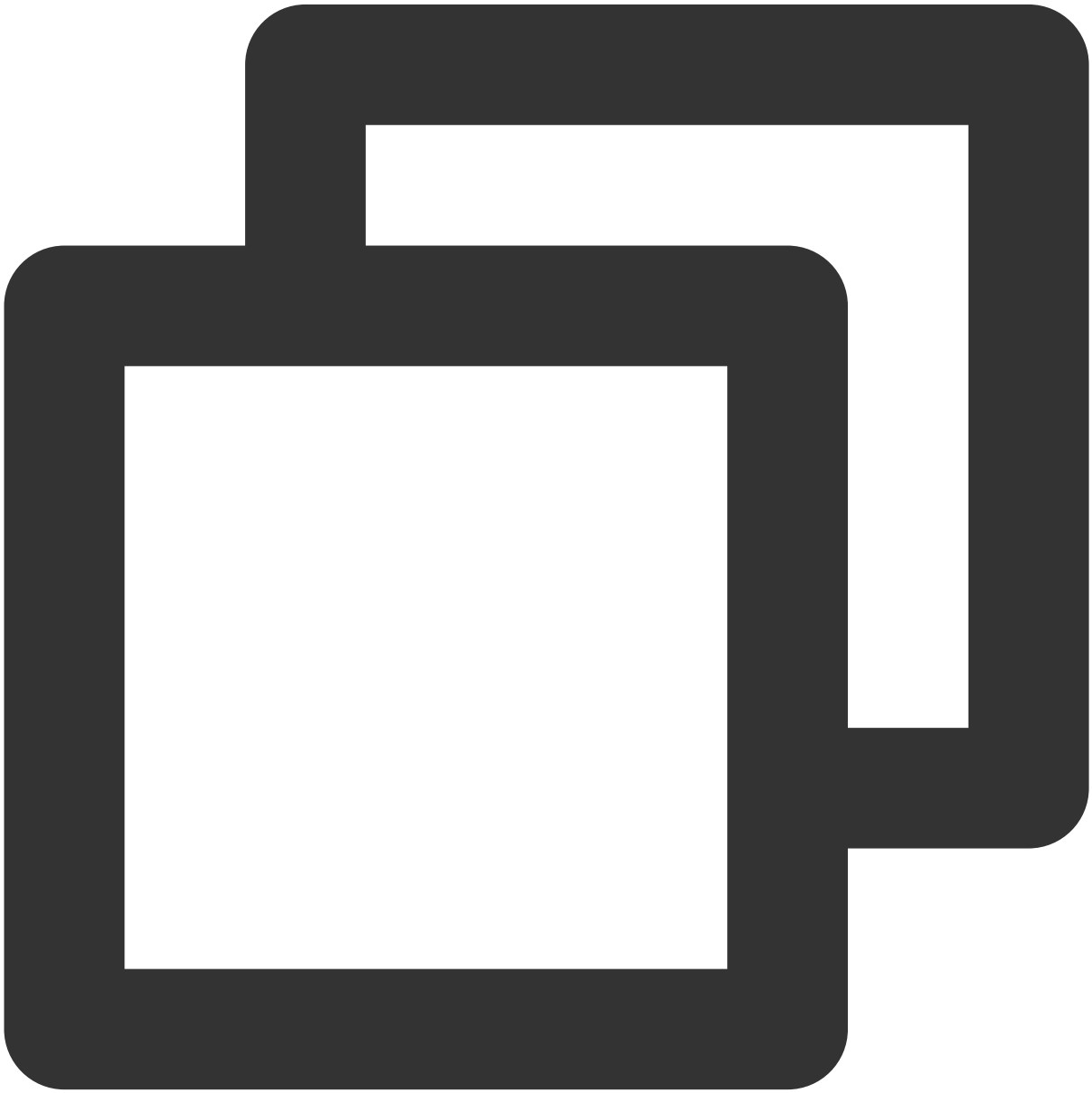


3. **Basic Information**エリアで**Display key**をクリックし、キーの情報をコピーしてGenerateTestUserSigファイルに保存します。

ご注意：

ここで説明するUserSigの作成は、クライアントのソースコードでSECRETKEYを設定する方法を使用します。この方法では、逆コンパイルによってSECRETKEYがハックされる可能性が大きいです。キーが漏洩すると、攻撃者はご利用中のTencent Cloudのトラフィックを盗用できます。そのため**この方法は、ローカルでのDemoクイックスタートおよび機能デバッグにのみ適します**。
UserSigの正しい発行方法として、UserSigの演算コードをご利用中のサーバーに集約し、Appのインターフェースを提供し、UserSigを必要とする場合、ご利用中のAppから業務サーバーにリクエストを送信し動的UserSigの取得を要求します。詳しくは、サーバーでのUserSig新規作成をご参照ください。

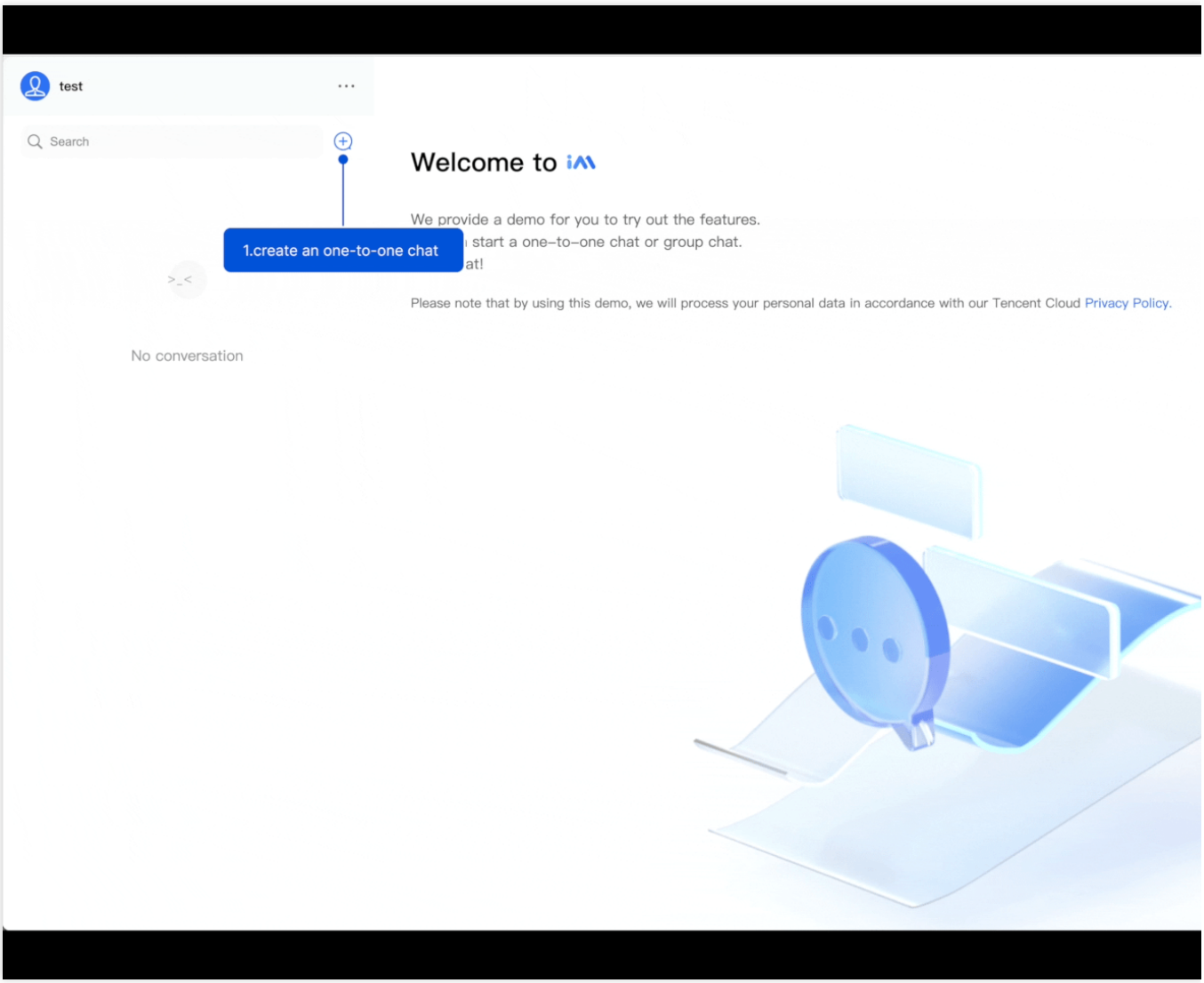
ステップ3：プロジェクトの起動



```
# Launch the project
$ cd examples/sample-chat
$ npm run start
```

ステップ4:初メッセージの送信

1. プロジェクトを正常に起動した後、「+」アイコンをクリックし、会話を作成します。
2. 入力ボックスで他のユーザーのuserIDを検索します。
3. ユーザープロファイルフォトをクリックして会話を開始します。
4. 入力ボックスにメッセージを入力し、「enter」キーを押して送信します。



Quick Links

Demo App

Client API

Free Demos

FAQ

GitHub Source

Generating UserSig

クイックスタート（Electron）

最終更新日：：2024-04-11 16:20:42

このドキュメントでは、Tencent CloudのIM Demo（Electron）を速やかに実行する方法と、Electron SDKを統合する方法について説明します。

環境要件

プラットフォーム	バージョン
Electron	13.1.5以上のバージョン。
Node.js	v14.2.0

サポートするプラットフォーム

現在、MacosとWindows両方のプラットフォームをサポートしています。

体験DEMO

導入を開始する前に、DEMOをご利用いただくと、Tencent Cloud IM Electron SDKを簡単にご覧いただけます。

前提条件

Tencent Cloudアカウントの登録を行い、実名認証が完了済みであること。

操作手順

手順1：アプリケーションの作成

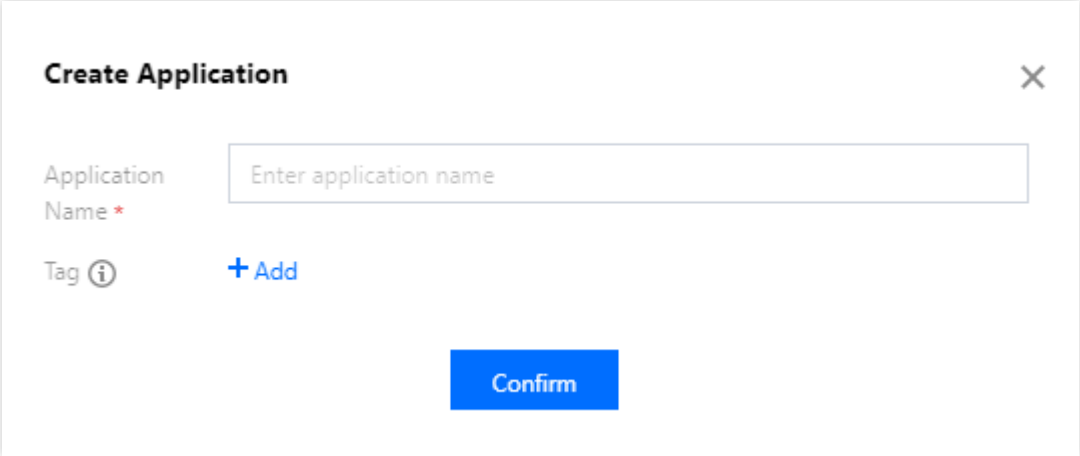
1. IMコンソールにログインします。

説明：

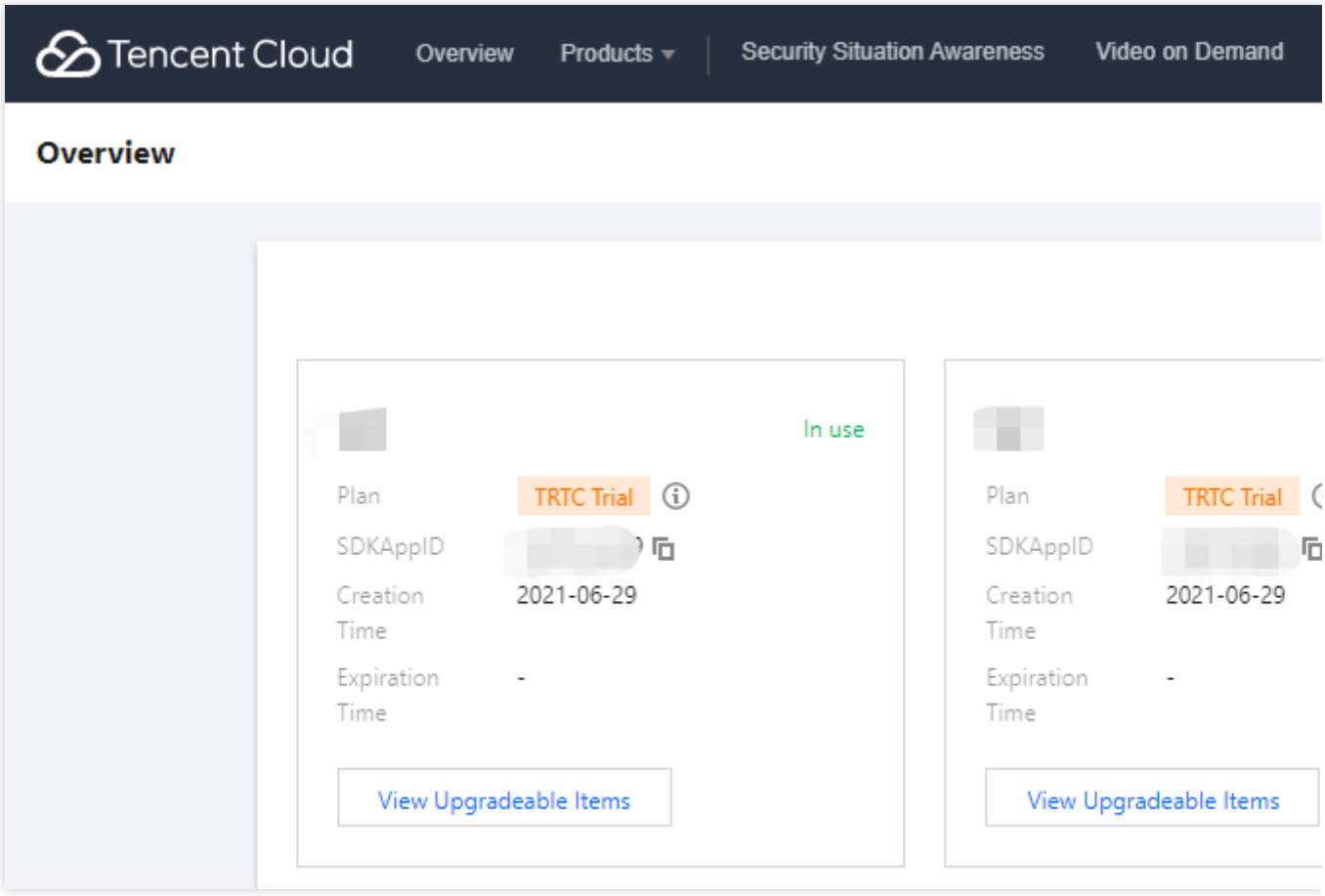
アプリケーションをすでに保有している場合は、そのSDKAppIDを記録してからキー情報の取得を実行してください。

同じTencent Cloudのアカウントで、最大300個のIMアプリケーションを作成することができます。すでにアプリケーションが300個ある場合は、使用する必要のないアプリケーションを使用停止して削除した後、新しいアプリケーションを作成することができます。**アプリケーションが削除された後、そのSDKAppIDに対応するすべてのデータとサービスは失われます。慎重に操作を行ってください。**

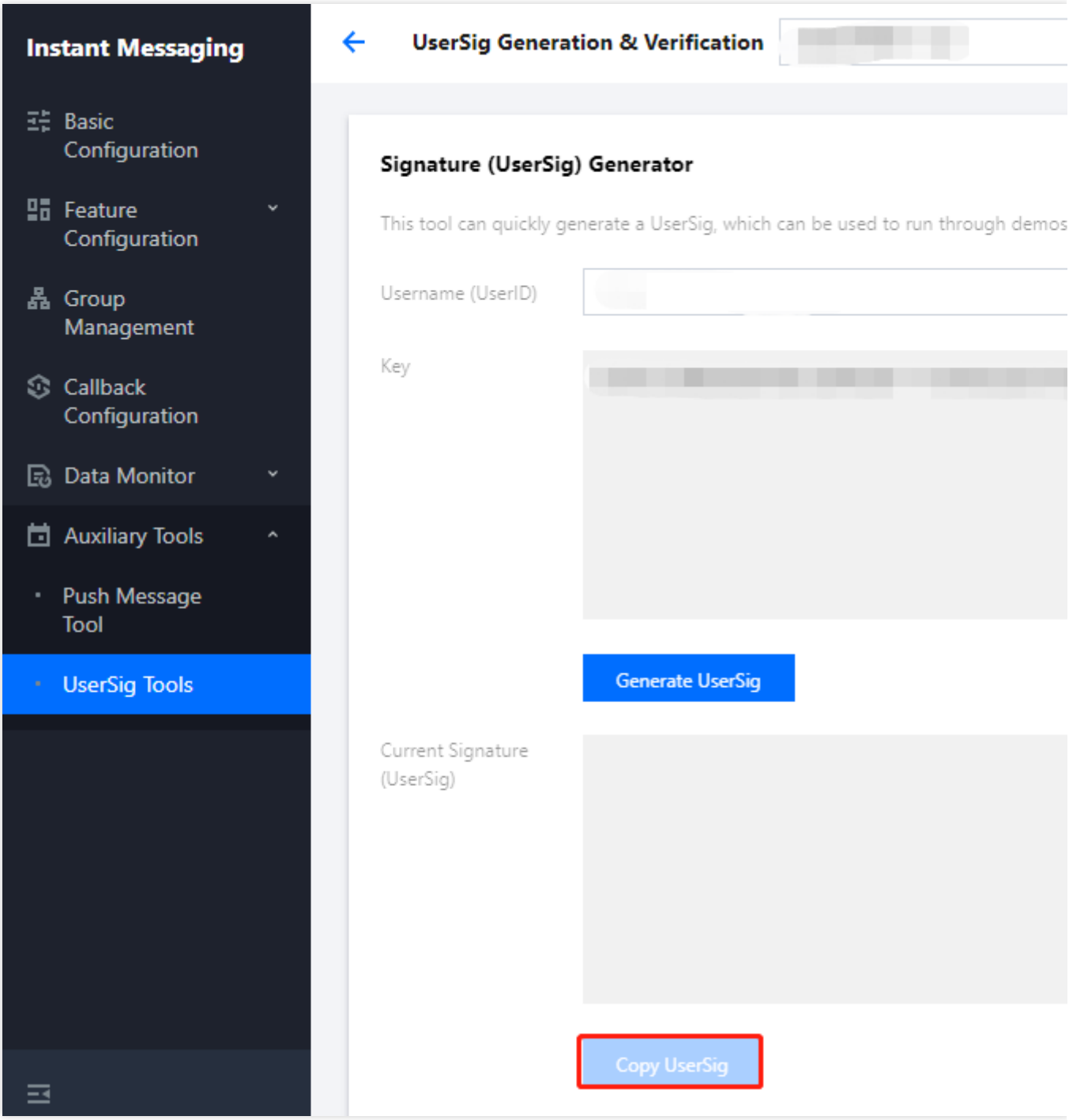
2. **新しいアプリケーションの作成**をクリックし、**アプリケーションの作成**のダイアログボックスにアプリケーション名を入力し、**OK**をクリックします。



3. SDKAppID情報を保存してください。コンソールの概要ページで、作成したアプリケーションのステータス、サービスバージョン、SDKAppID、タグ、作成時間、有効期限を確認できます。



4. 作成後のアプリケーションをクリックし、左側のナビゲーションバーで**支援ツール>UserSig発行&チェック**をクリックすると、UserIDおよびそれに対応するUserSigが作成されます。署名情報をコピーし、後でログインして使用します。



ステップ2：Electron SDKの統合に適した方法の選択

IMは2種類の統合方法を提供しており、最適な方法を選択して統合を行うことができます。

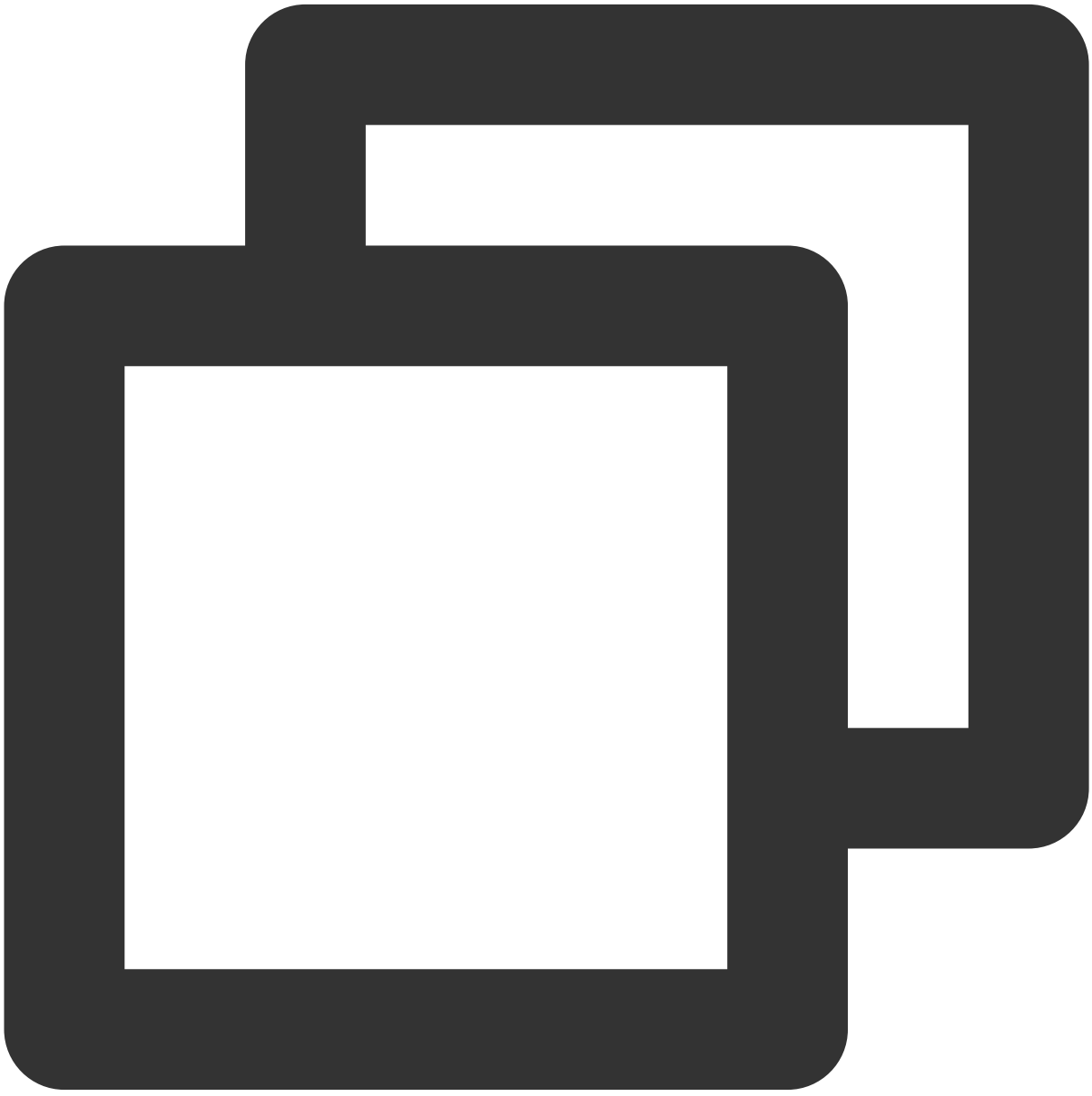
継承方式	適用シナリオ
DEMOの使用	IM Demoは完全なチャット機能を含み、コードはオープンソース化されています。チャットライクなユースケースを実装したい場合は、Demoを使用して二次開発を行うことができます。今すぐDemo体験が可能です。

自己実装	この方法は、Demoがアプリケーションの機能インターフェース要件を満たしていない場合に使用できます。

IM SDKのAPIの理解を深めるために、APIドキュメントも用意されています。

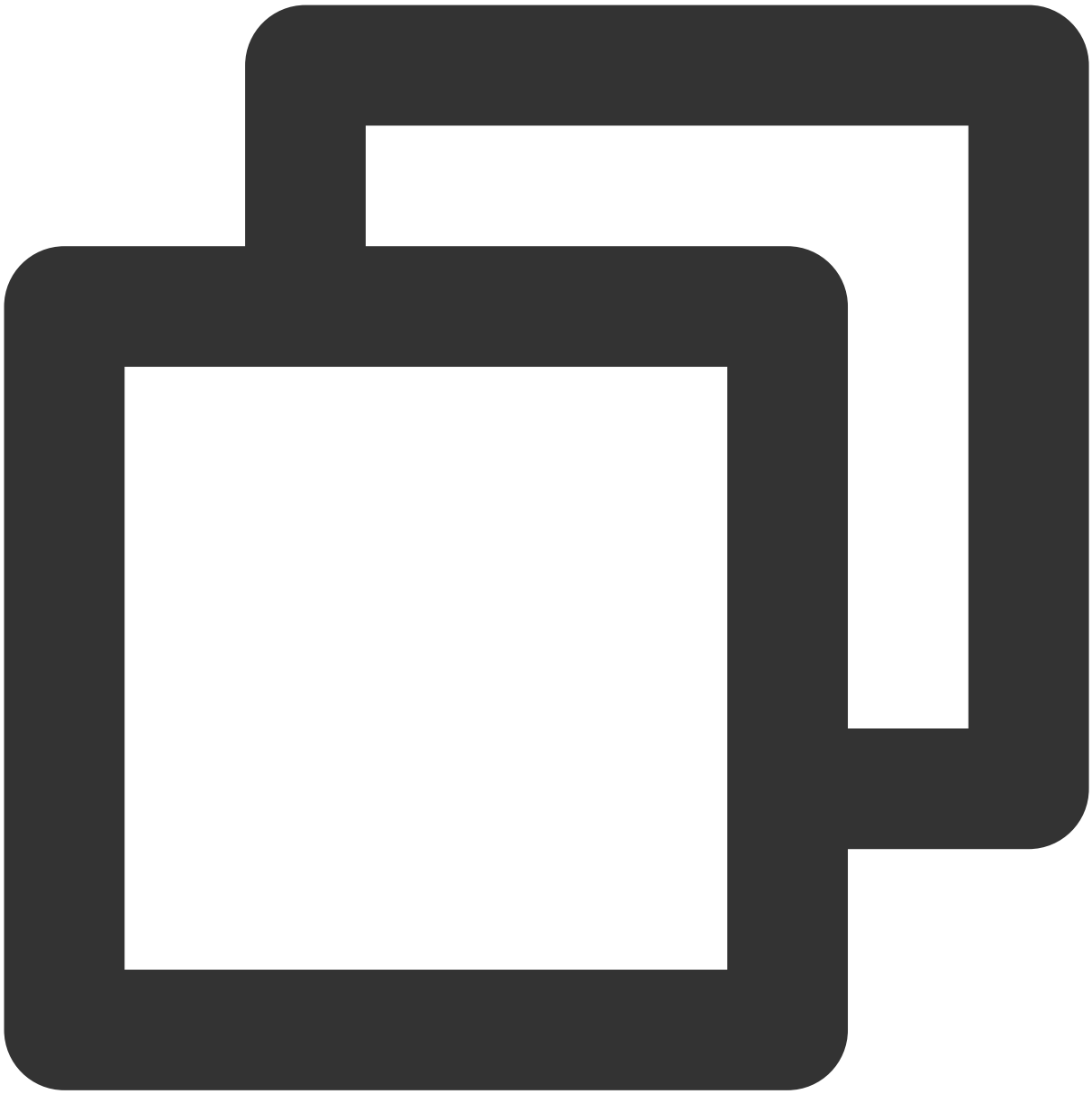
ステップ3：Demoの使用

1. Instant Messaging IM Electron Demoソースコードをローカルにクローンします。



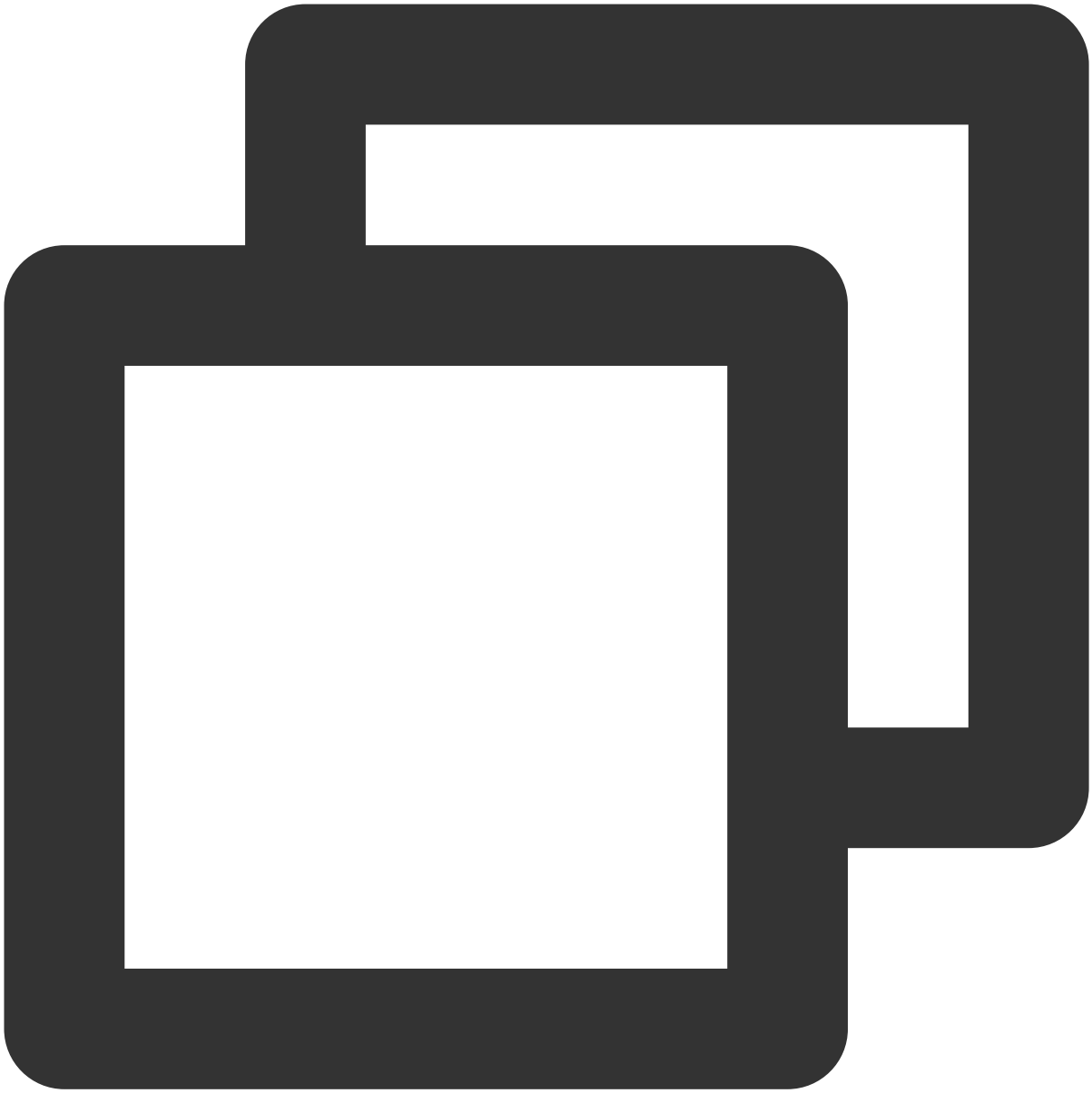
```
git clone https://github.com/TencentCloud/tc-chat-demo-electron.git
```

2. プロジェクトの依存関係をインストールします。



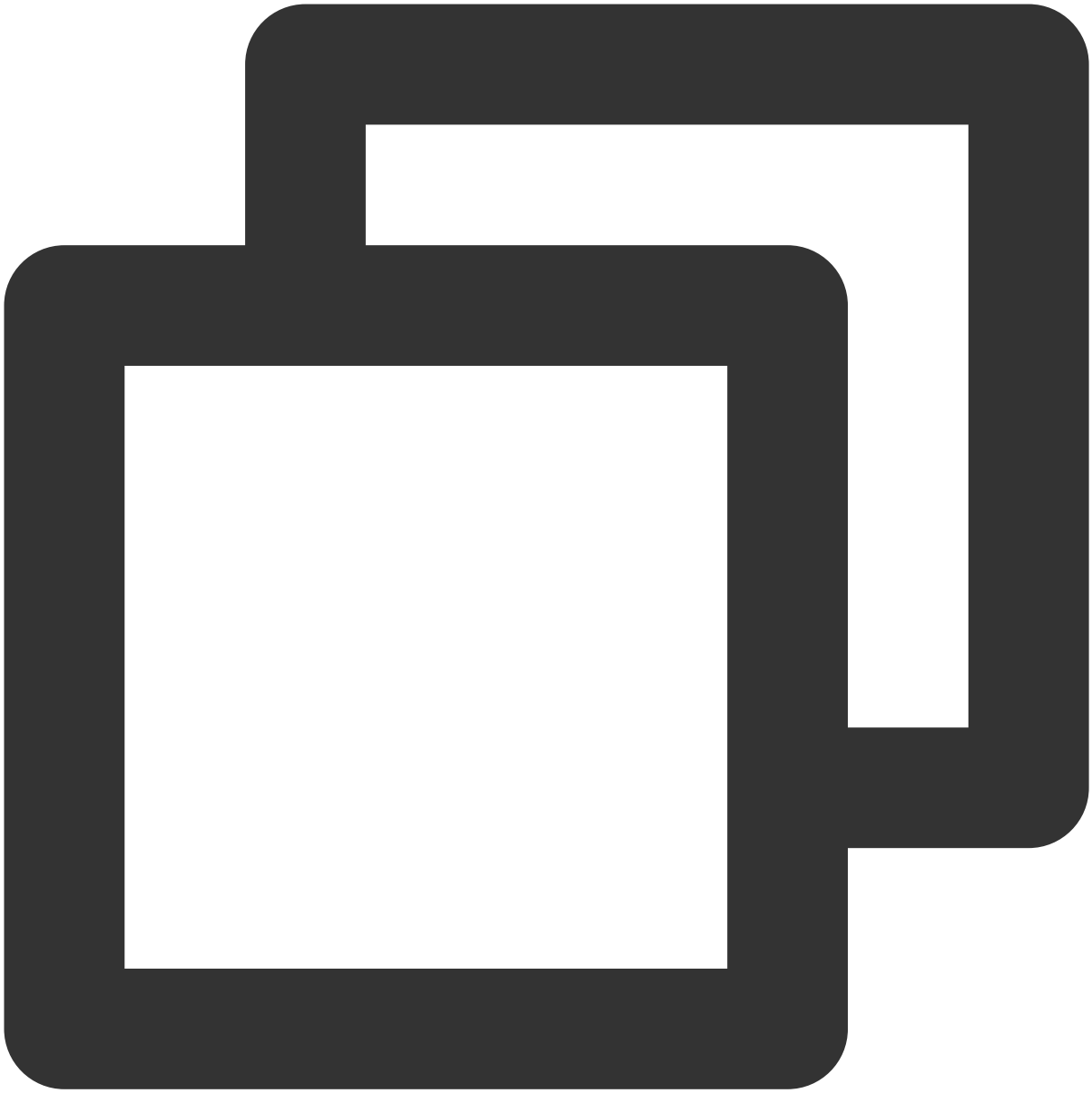
```
// プロジェクトのルートディレクトリ
npm install

// レンダリングプロセスディレクトリ
cd src/client
npm install
```

3. プロジェクトを実行します。



```
// プロジェクトのルートディレクトリ
npm start
```

4. プロジェクトをパッケージ化します。



```
// macパッケージ化
npm run build:mac
// windowsパッケージ化
npm run build:windows
```

説明：

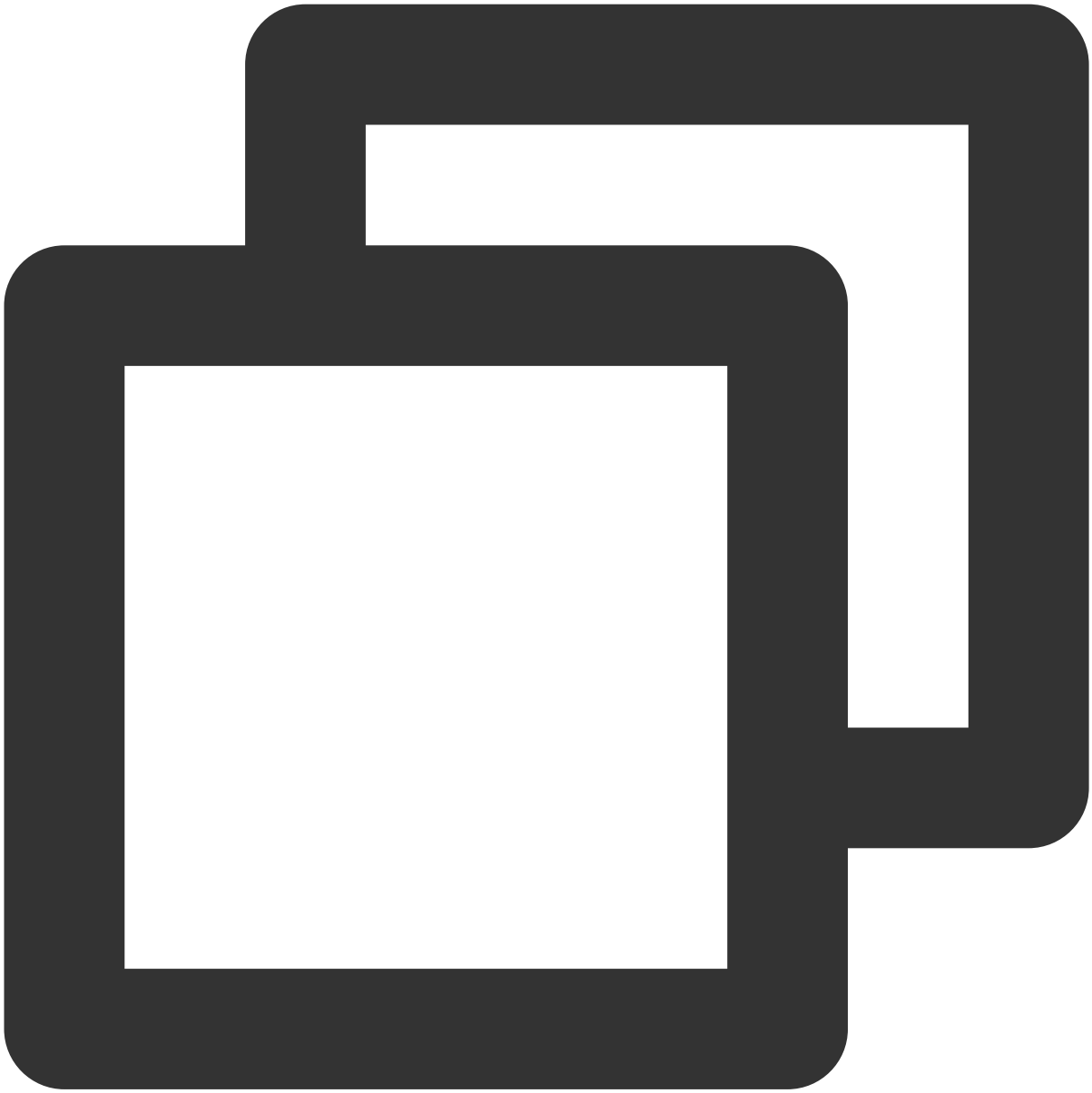
demo内のメインプロセスのディレクトリは `src/app/main.js` であり、レンダリングプロセスのディレクトリは `src/client` です。実行中に問題が発生した場合は、FAQを使用して解決することができます。

ステップ4：自己実装

Electron SDKのインストール

次のコマンドを使用して、最新バージョンのElectron SDKをインストールします。

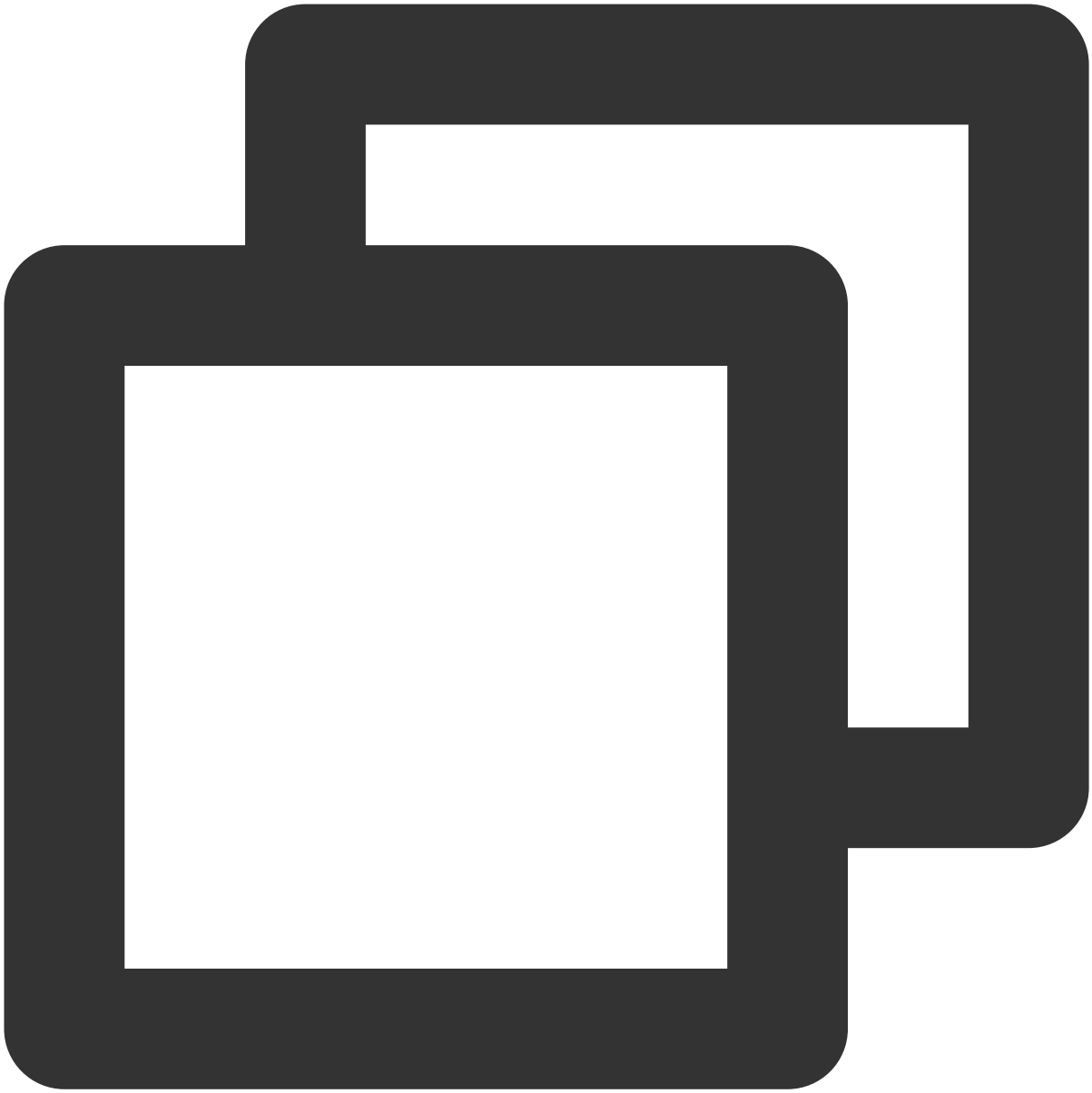
コマンドラインでの実行：



```
npm install im_electron_sdk
```

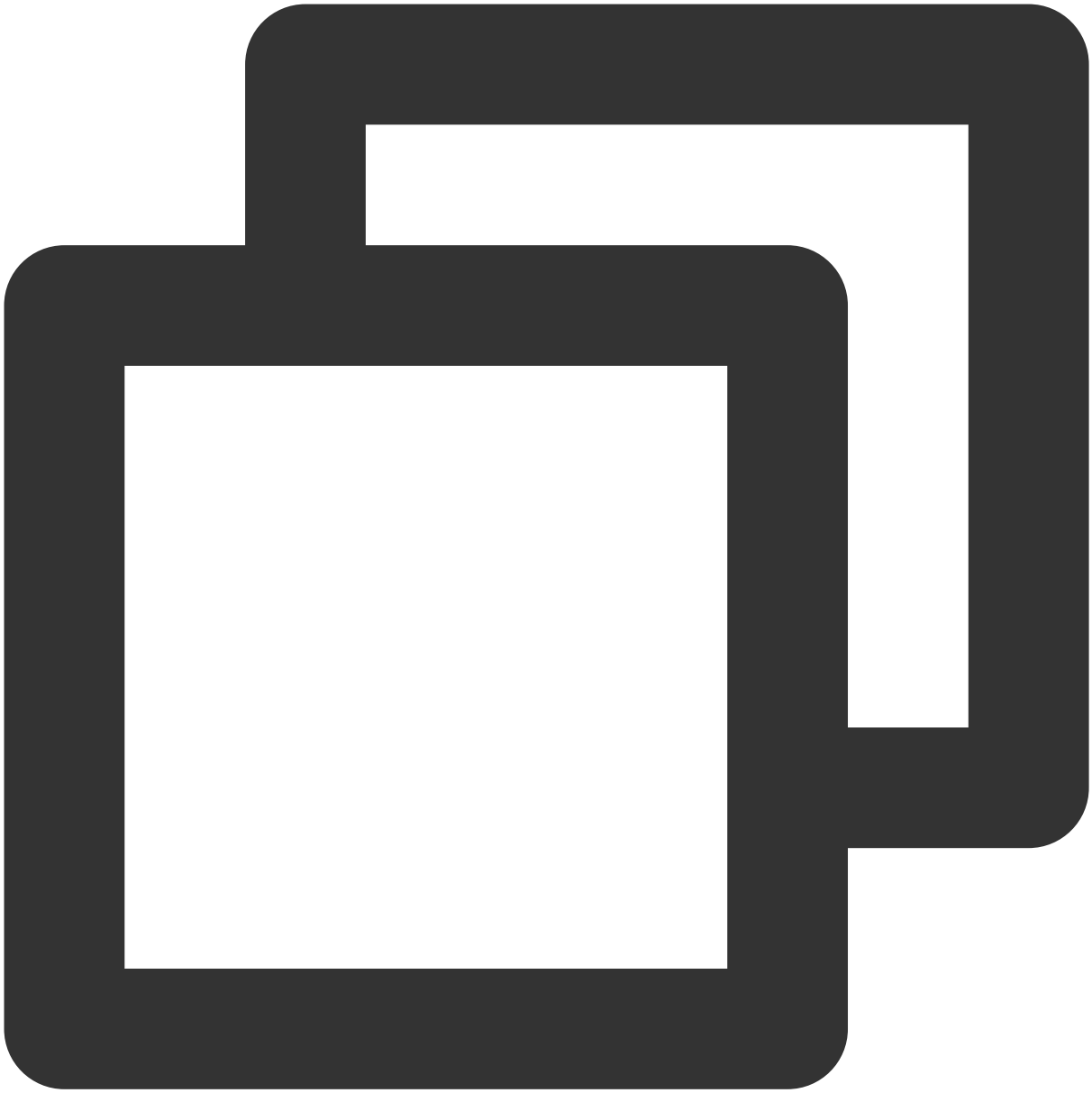
SDKの初期化完了

1. `sdkAppID` を `TimMain` に渡します。



```
//  メインプロセス
const TimMain = require('im_electron_sdk/dist/main')

const sdkappid=0;// Tencent Cloud  IMコンソールで申し込みます。
const tim = new TimMain({
  sdkappid:sdkappid
})
```

2. `TIMInit` を呼び出してSDKの初期化を完了します。

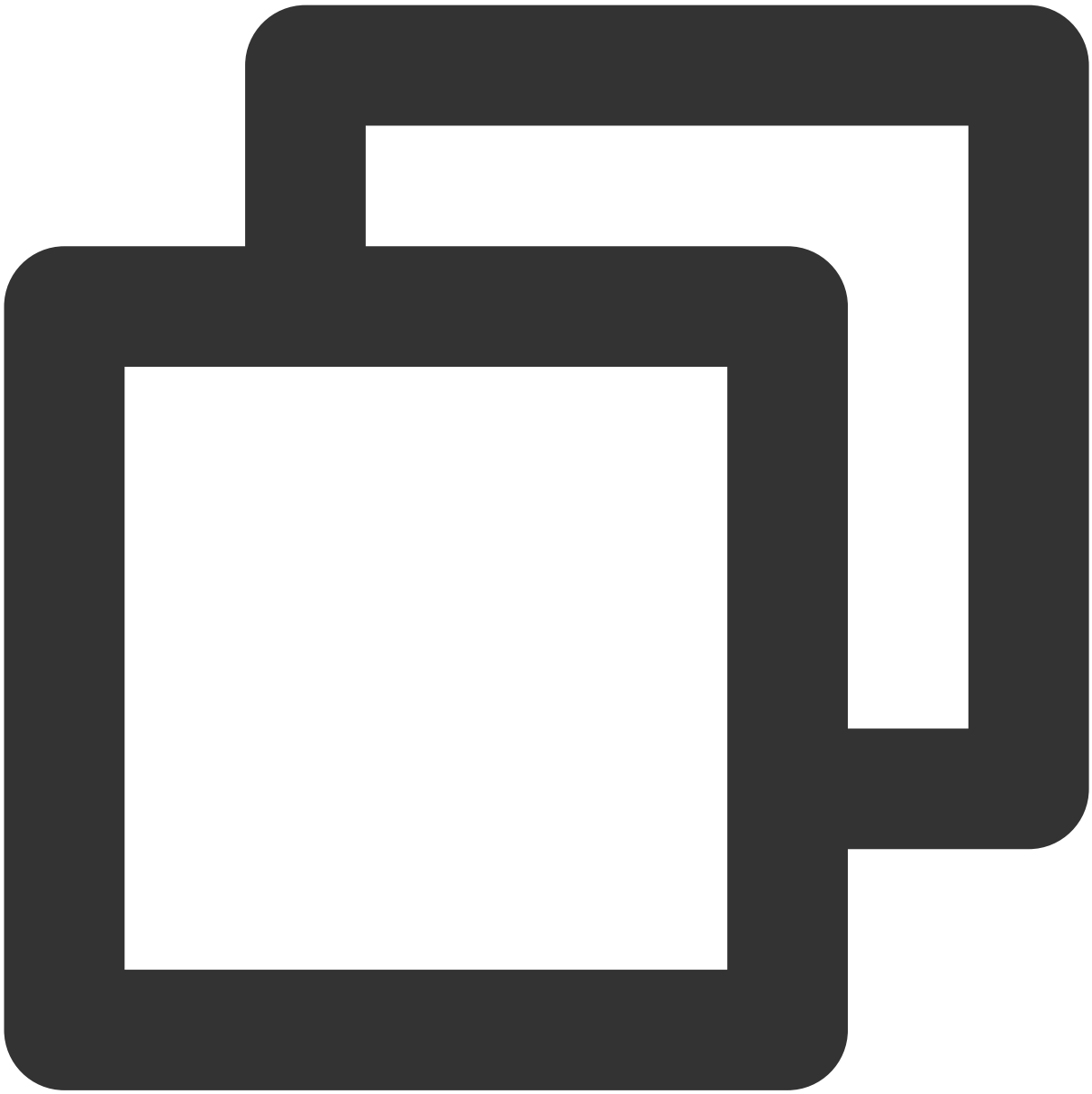


```
// レンダリングプロセス
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
// 初期化
timRender.TIMInit()
```

3. テストユーザーにログインします。

この時点で、最初にコンソール上で作成したテストアカウントを使用して、ログイン検証を完了することが可能です。

`timRender.TIMLogin` メソッドを呼び出し、テストユーザとしてログインします。
戻り値 `code` が0の場合、ログインは成功です。



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

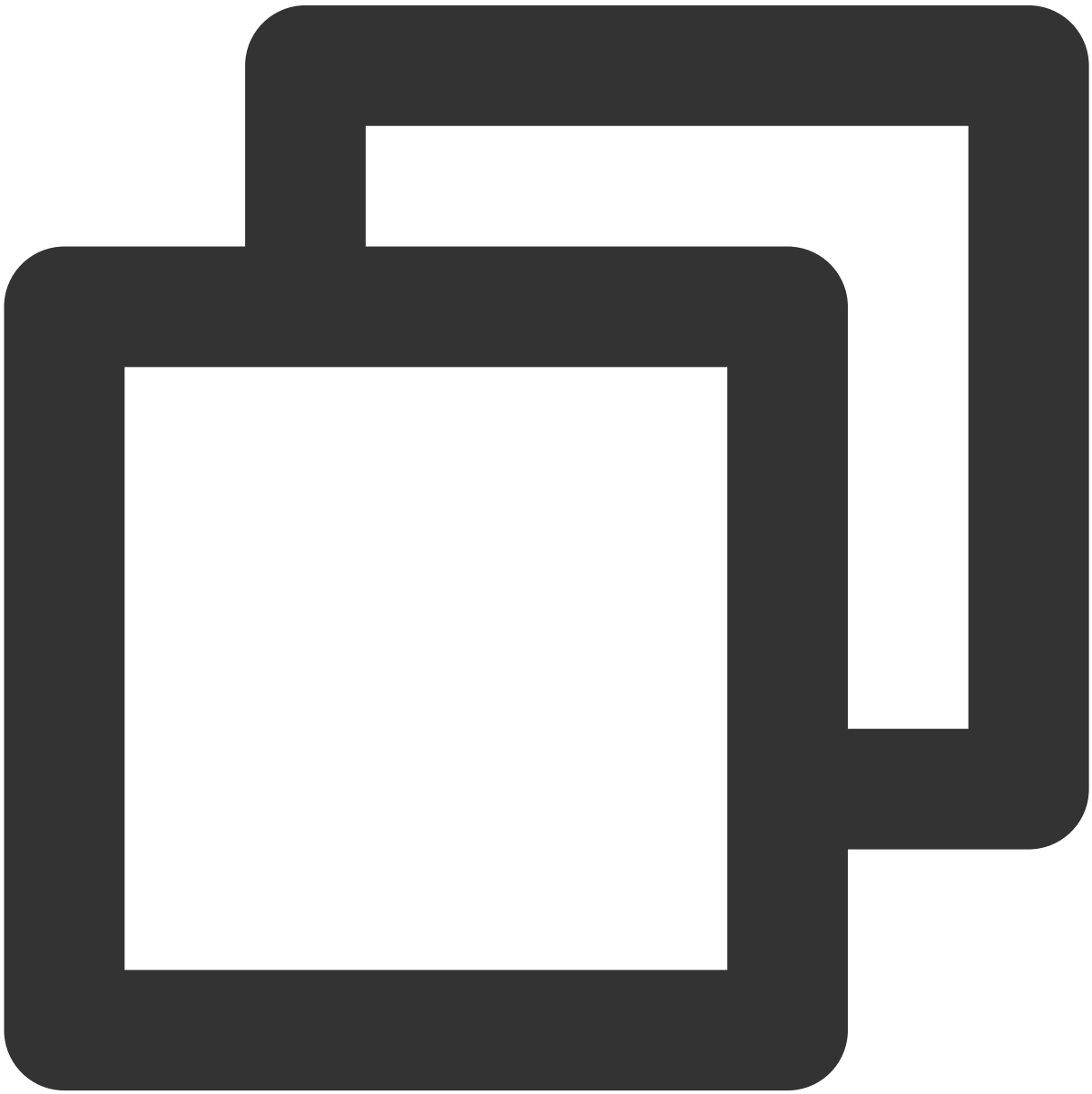
説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、 `UserSig` の計算コードをサーバーに統合し、Appのインターフェースを提供することです。 `UserSig` が必要なときは、Appから業務サーバーに対し、動的に `UserSig` を取得するリクエストを送信します。詳細についてはサーバーでのUserSig新規作成をご参照ください。

メッセージ送信

ここでは、テキストメッセージの送信を例として、 `code` が0を返すとメッセージ送信が成功したことを意味します。

サンプルコード：



```
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);
```

説明：

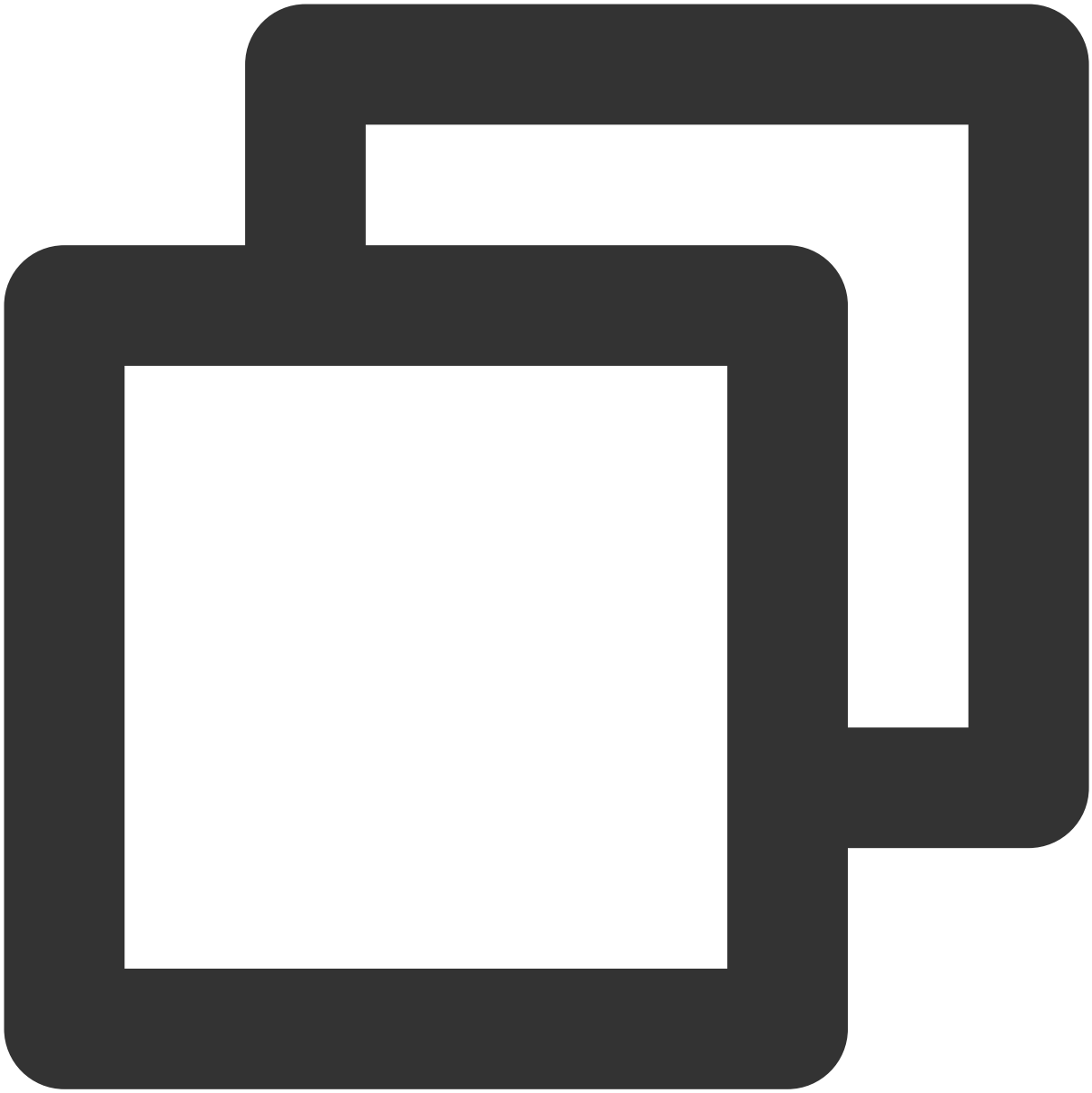
送信に失敗した場合は、sdkAppIDが知らない人宛てのメッセージ送信をサポートしていないことが原因の可能性があります。コンソールで有効にしてからテストに使用してください。 このリンクをクリックして、フレンドリレーションシップチェーンのチェックを無効にしてください。

セッションリストの取得

前の手順でテストメッセージの送信が完了しましたので、別のテストアカウントでログインし、セッションリストをプルできるようになりました。

一般的なユースケースは次のようなものです：

アプリケーションの起動後すぐにセッションリストを取得し、その後は長時間リンクを監視して、セッションリストの変更をリアルタイムに更新します。



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

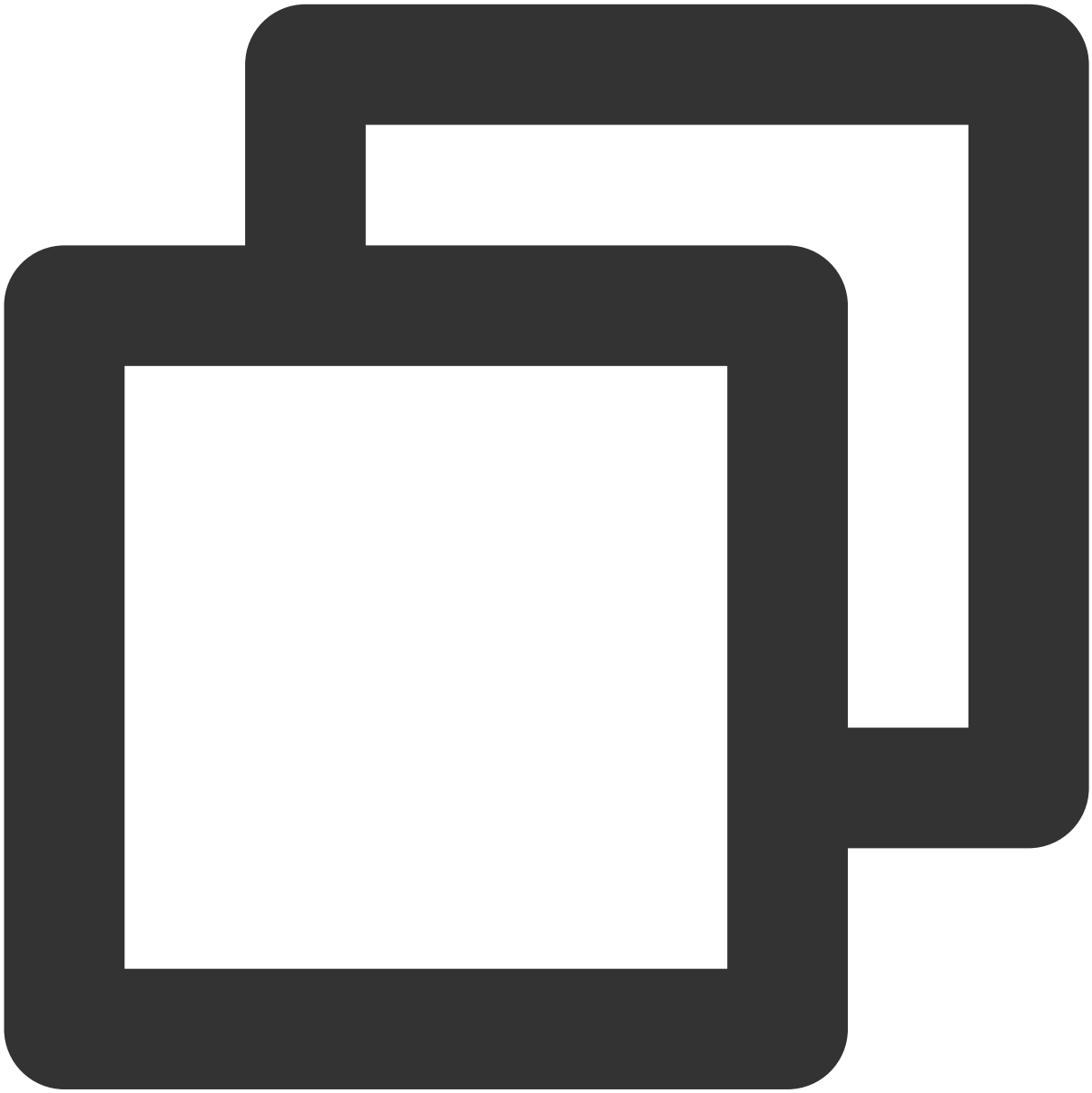
この時点で、前の手順で別のテストアカウントを使用して送信したメッセージのセッションを見ることができるようになりました。

メッセージの受信

一般的なユースケースは次のようなものです：

1. インターフェースが新しいセッションに入ると、まず一定量のメッセージ履歴を一括でリクエストし、メッセージ履歴リストの表示に用います。
2. 長時間接続を監視し、新しいメッセージをリアルタイムに受信してメッセージ履歴リストに追加します。

メッセージ履歴リストの一括リクエスト

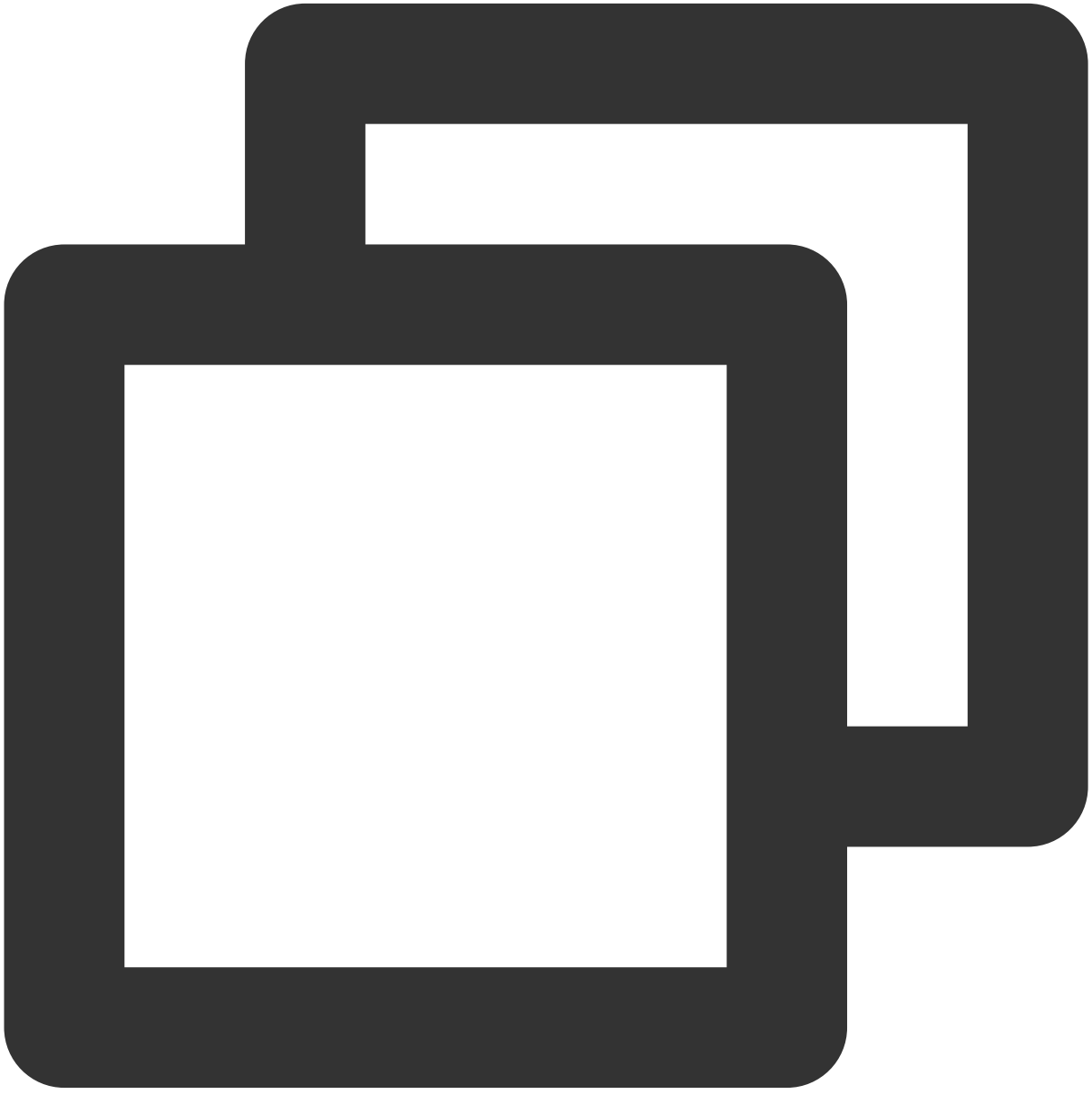


```
let param:MsgGetMsgListParams = {
        conv_id: conv_id,
        conv_type: conv_type,
        params: {
            msg_getmsglist_param_last_msg: msg,
            msg_getmsglist_param_count: 20,
```

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

監視による新メッセージのリアルタイム取得

callbackをバインドするサンプルコードは次のとおりです：



```
let param : TIMRecvNewMsgCallbackParams = {
            callback: (...args)=>{},
```

```
            user_data: user_data
        }
timRenderInstance.TIMAddRecvNewMsgCallback(param);
```

この時点で、IMモジュールの開発は基本的に完了し、メッセージの送受信や様々なセッションに入ることが可能になりました。
続いて、グループ、ユーザープロファイル、リレーションシップチェーン、オフラインプッシュ、ローカル検索などの関連機能の開発を完了することができます。
詳細については、APIドキュメントをご参照ください。

よくあるご質問

サポートされているプラットフォームはどれですか？

現在、MacosとWindows両方のプラットフォームをサポートしています。

エラーコードのクエリー方法

IM SDKのAPIレベルのエラーコードについては、エラーコードをご確認ください。

開発環境インストール問題で、`npm ERR！gyp ERR！stack TypeError：Cannot assign to read only property'cflags'of object'#<Object>'` エラーが発生した場合の解決方法は。

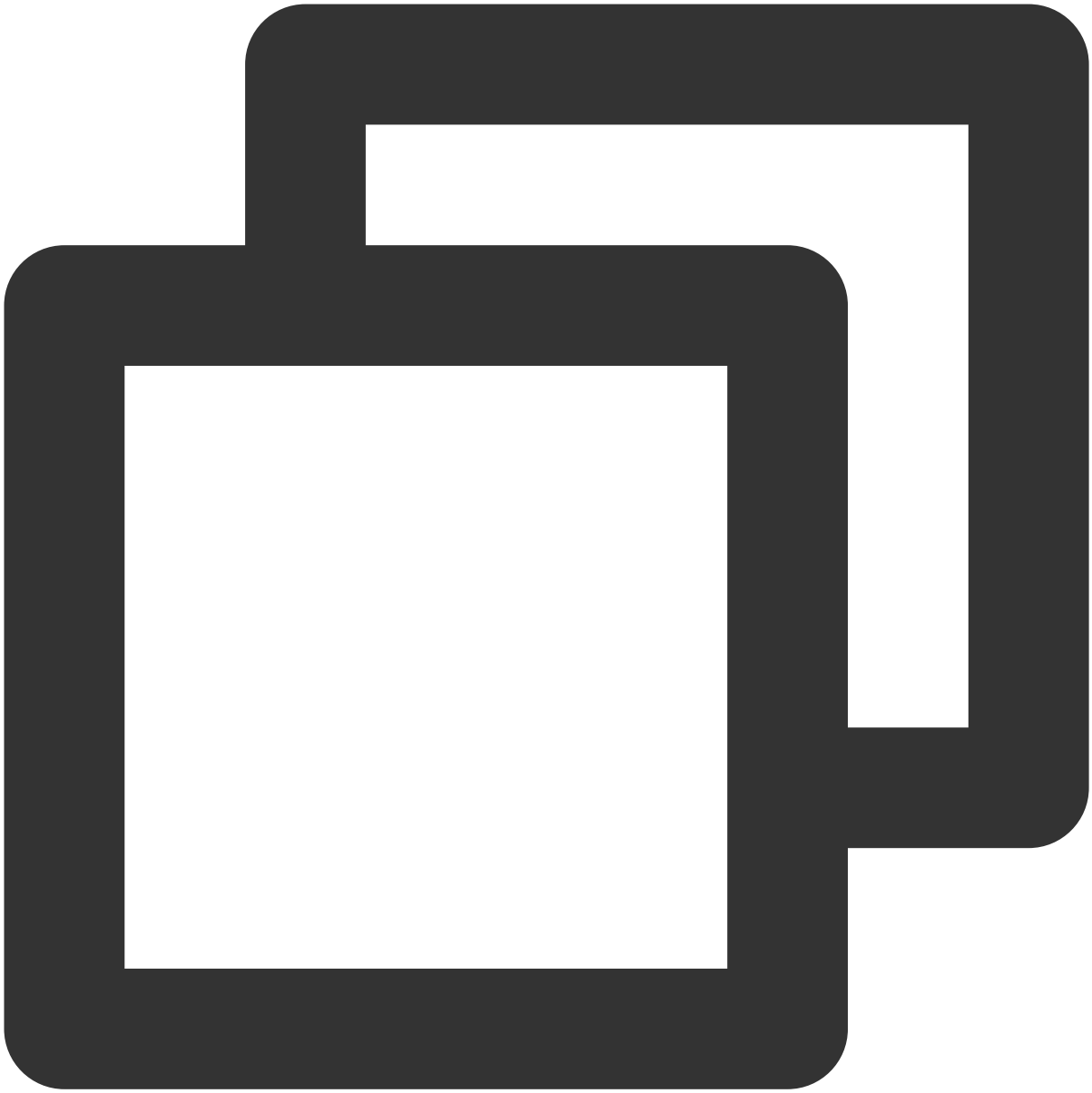
nodeのバージョンを下げてください。16.18.1をお勧めします。

開発環境インストールの問題で、`gypgyp ERR!ERR` エラーが発生した場合の解決方法は。

gypgyp ERR!ERR! をご参照ください。

`npm install` を実行すると `npm ERR!  Fix the upstream dependency conflict,or retry` のエラーが発生した場合の解決方法

npmV7より前のバージョンで依存性の競合が発生すると、依存性の競合は無視され、インストールが続行されます。
npmV7リリースからは自動的に無視されず、ユーザーが手動でコマンドを入力する必要があります。
以下のコマンドを実行してください：



```
npm install --force
```

`npm run start` を実行すると `Error：error:0308010C:digital envelope routines::unsupported` のエラーが発生します。どうすればいいですか。

nodeのバージョンを下げてください。16.18.1をお勧めします。

Mac側のDemoで `npm run start` を実行すると白い画面が表示される場合の解決方法は。

Macで `npm run start` を実行すると白い画面が表示される原因は、レンダリングプロセスコードのbuildが完了しておらず、メインプロセスによって開かれた3000ポートがブランクページであるためです。レンダリングプロセスコードのbuildが完了した後にウインドウを更新することで、この問題を解決できます。または `cd src/client && npm run dev:react` , `npm run dev:electron` を実行し、レンダリングプロセスとメインプロセスを別々に開始します。

`vue-cli-plugin-electron-builder` で構築されたプロジェクトは `native modules` をどのように使用しますか？

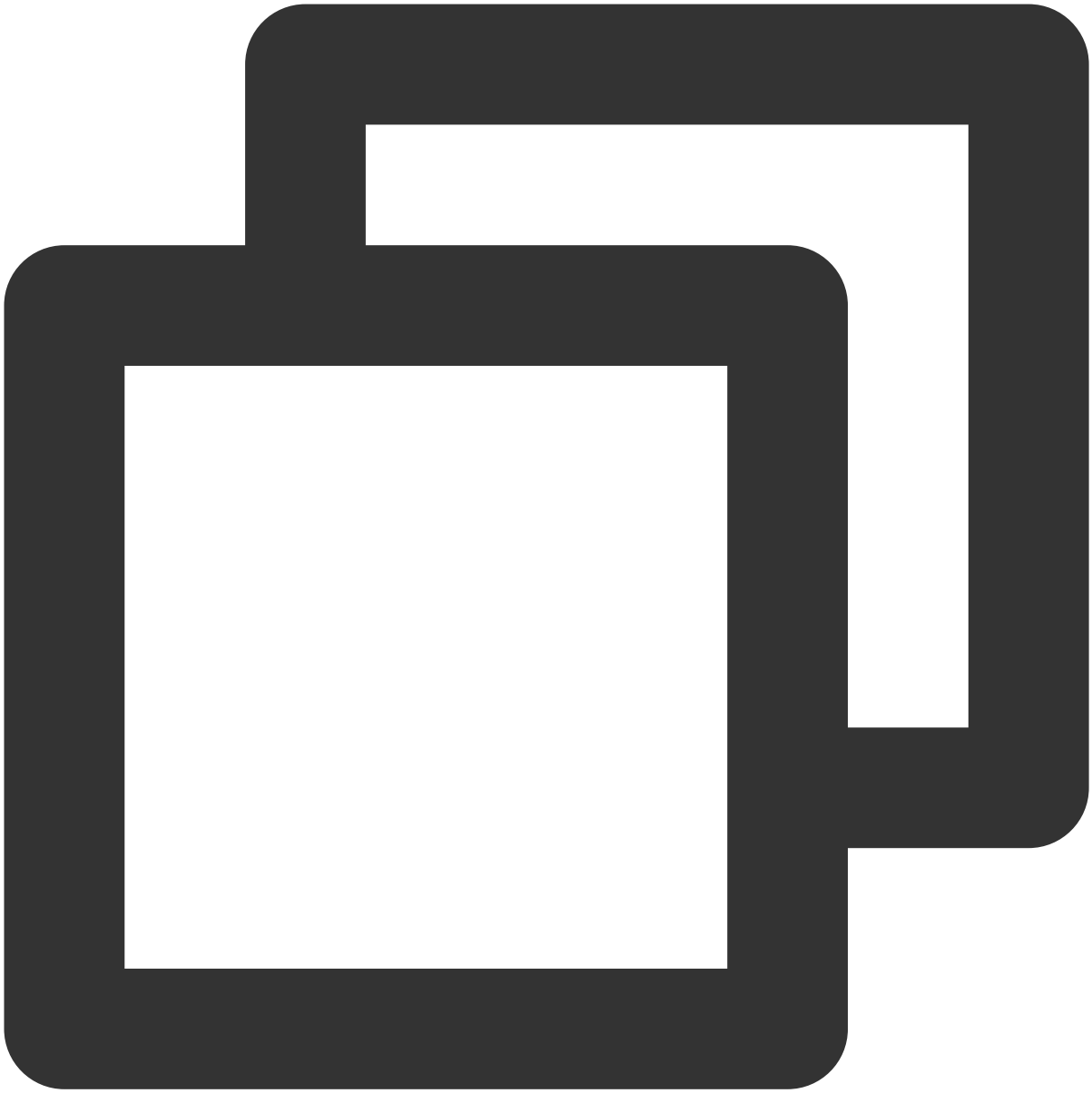
`vue-cli-plugin-electron-builder` を使用して構築されたプロジェクトで `native modules` を使用するには、No native build was found for platform=xxxをご参照ください。

`webpack` で構築されたプロジェクトはどのように `native modules` を使用しますか？

webpackを使用して独自に構築されたプロジェクトでnative modulesを使用するには、WindowsのFAQをご参照ください。

`Dynamic Linking Error` が表示されますか？

Dynamic Linking Error. electron-builderの設定



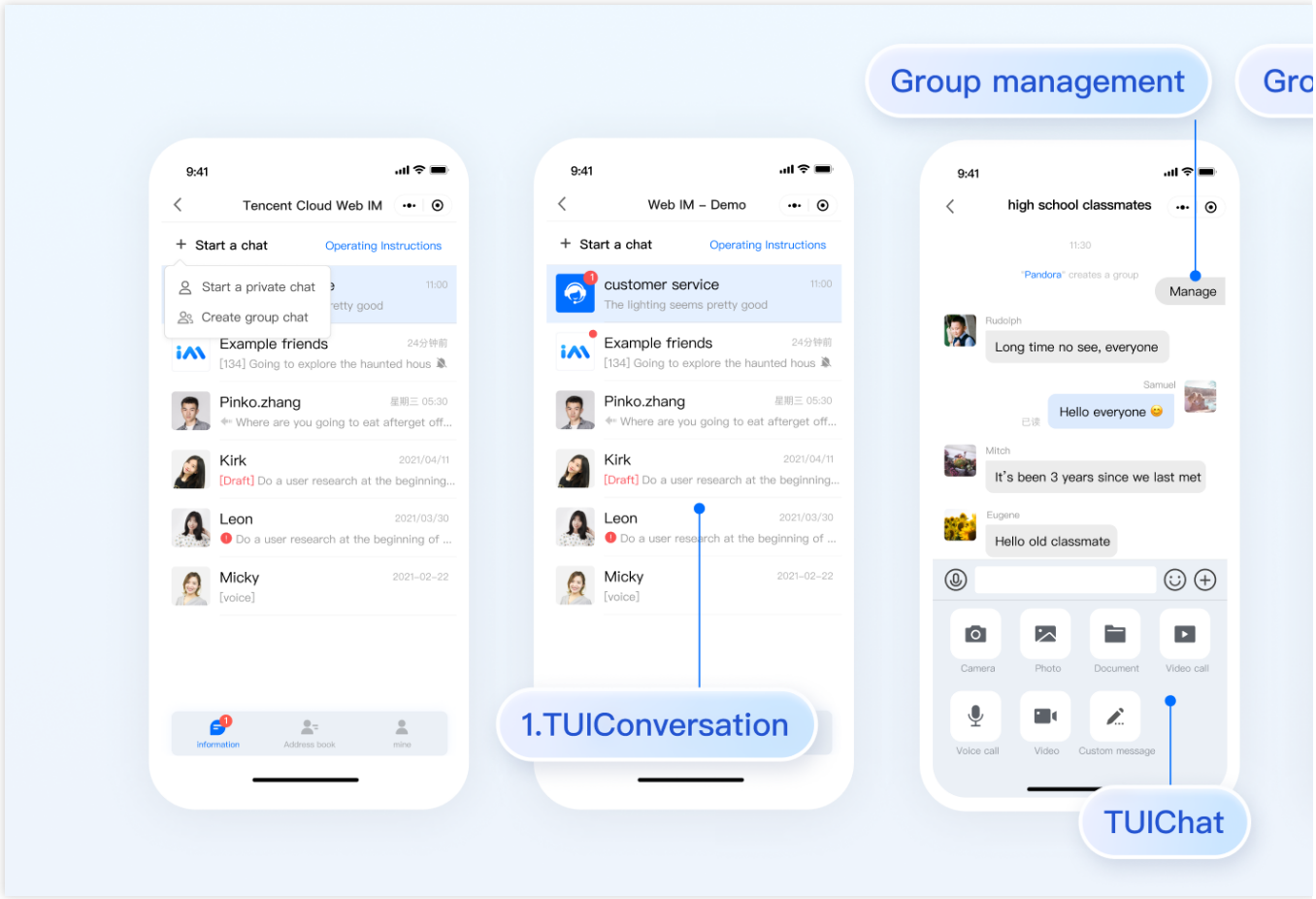
```
 extraFiles:[
  {
    "from": "./node_modules/im_electron_sdk/lib/",
    "to": "./Resources",
    "filter": [
      "**/*"
    ]
  }
]
```

uniapp

最終更新日：：2024-02-01 11:12:57

Introduction to chat-uikit-uniapp

chat-uikit-uniapp (vue2 /vue3) is a uniapp UI component library based on Tencent Cloud Chat SDK. It provides universally used UI components that include Conversation, Chat, and Group components. Leveraging these meticulously crafted UI components, you can quickly construct an elegant, reliable, and scalable Chat application. The interface of chat-uikit-uniapp is as demonstrated in the image below:



Supported Platform

Android

iOS

WeChat Mini Program
H5

Environment Requirements

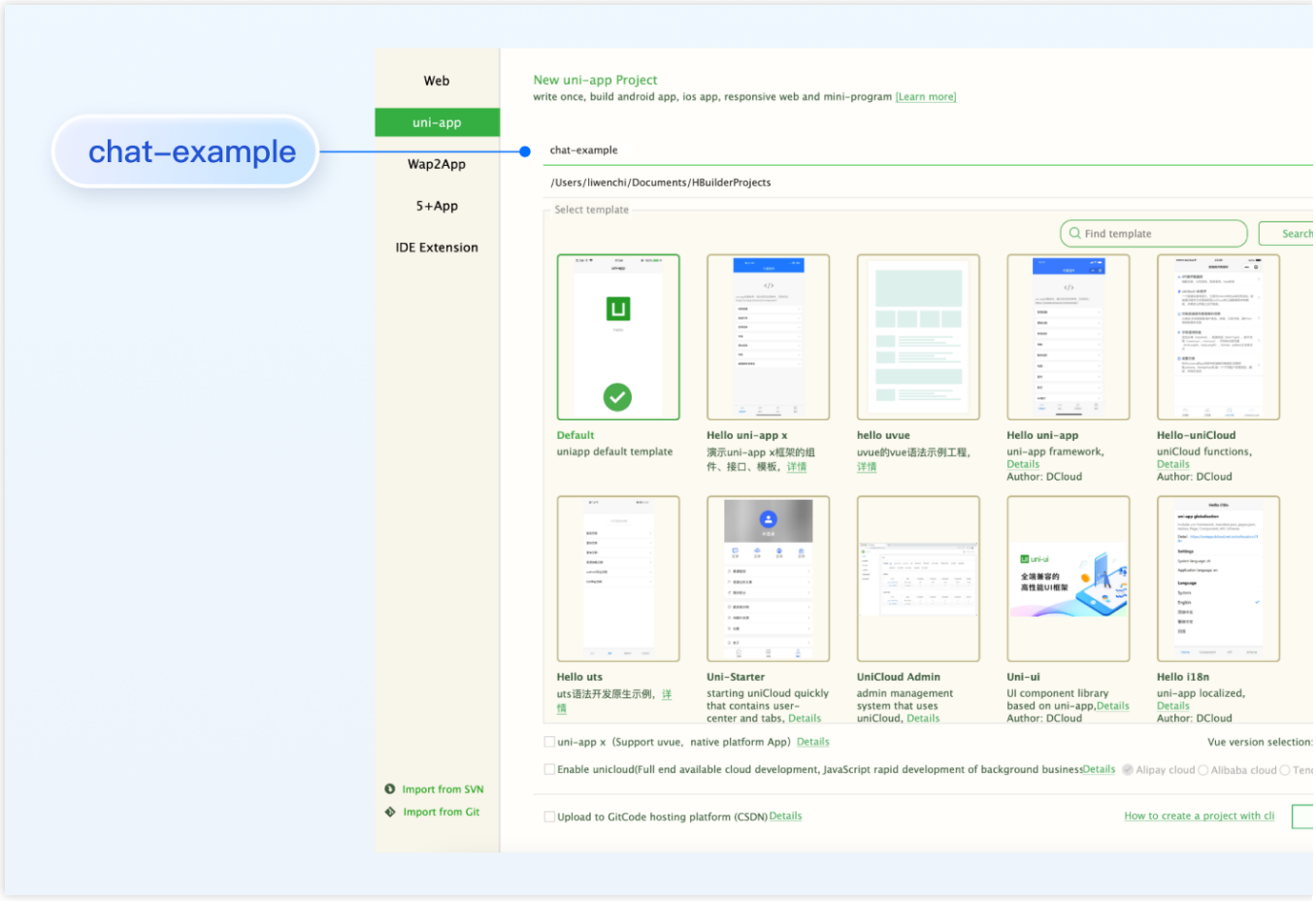
HBuilderX (HBuilderX Version >= 3.8.4.20230531) or upgrade to the newest version
Vue2 / Vue3
Sass (sass-loader version ≤ 10.1.1)
Node (12.13.0 ≤ node version ≤ 17.0.0. The official LTS version 16.17.0 of Node.js is recommended.)
npm (use a version that matches the Node version in use)

TUIKit Source Code Integration

Follow the steps below to send your inaugural message.

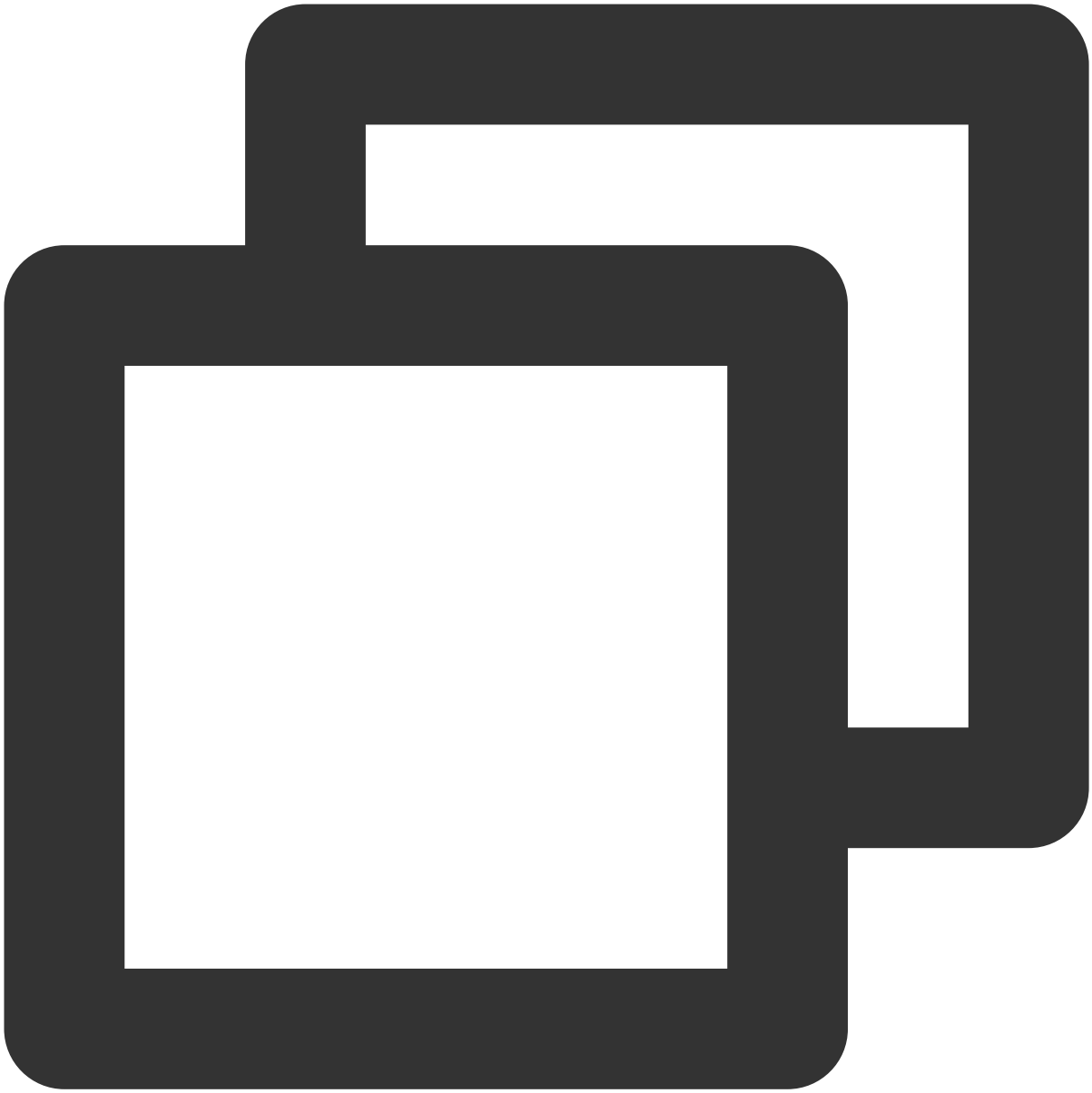
Step 1: create a project (ignore this step if already has project)

Launch HbuilderX, select "File-New-Project" in the menu bar, and create a uni-app project named `chat-example` .



Step 2. Download the TUIKit component

Since HBuilderX does not create package.json files by default, you need to proactively create one. Execute the following command in the root directory of the project:



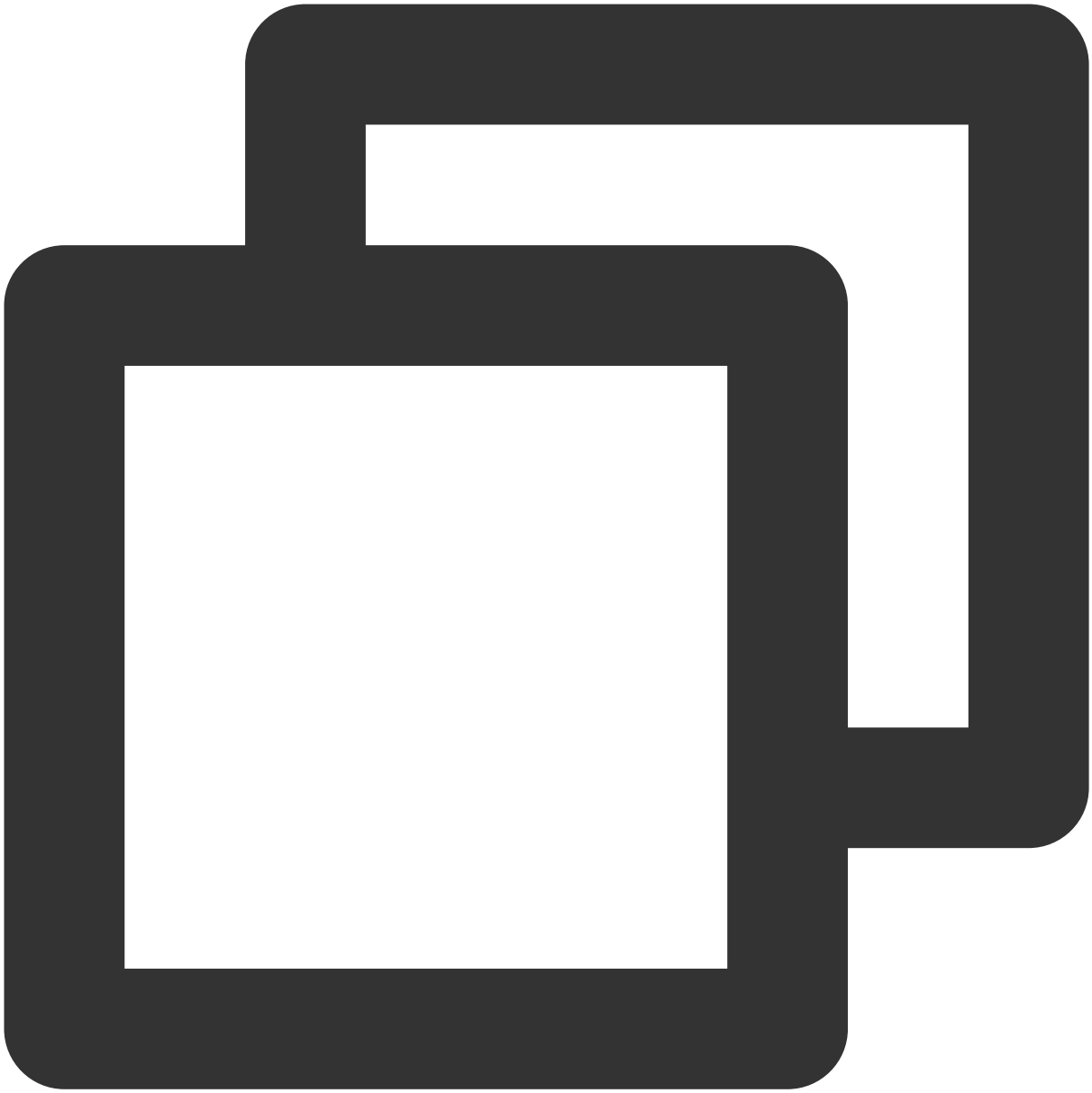
```
npm init -y
```

Download TUIKit and copy it to the source code:

macOS

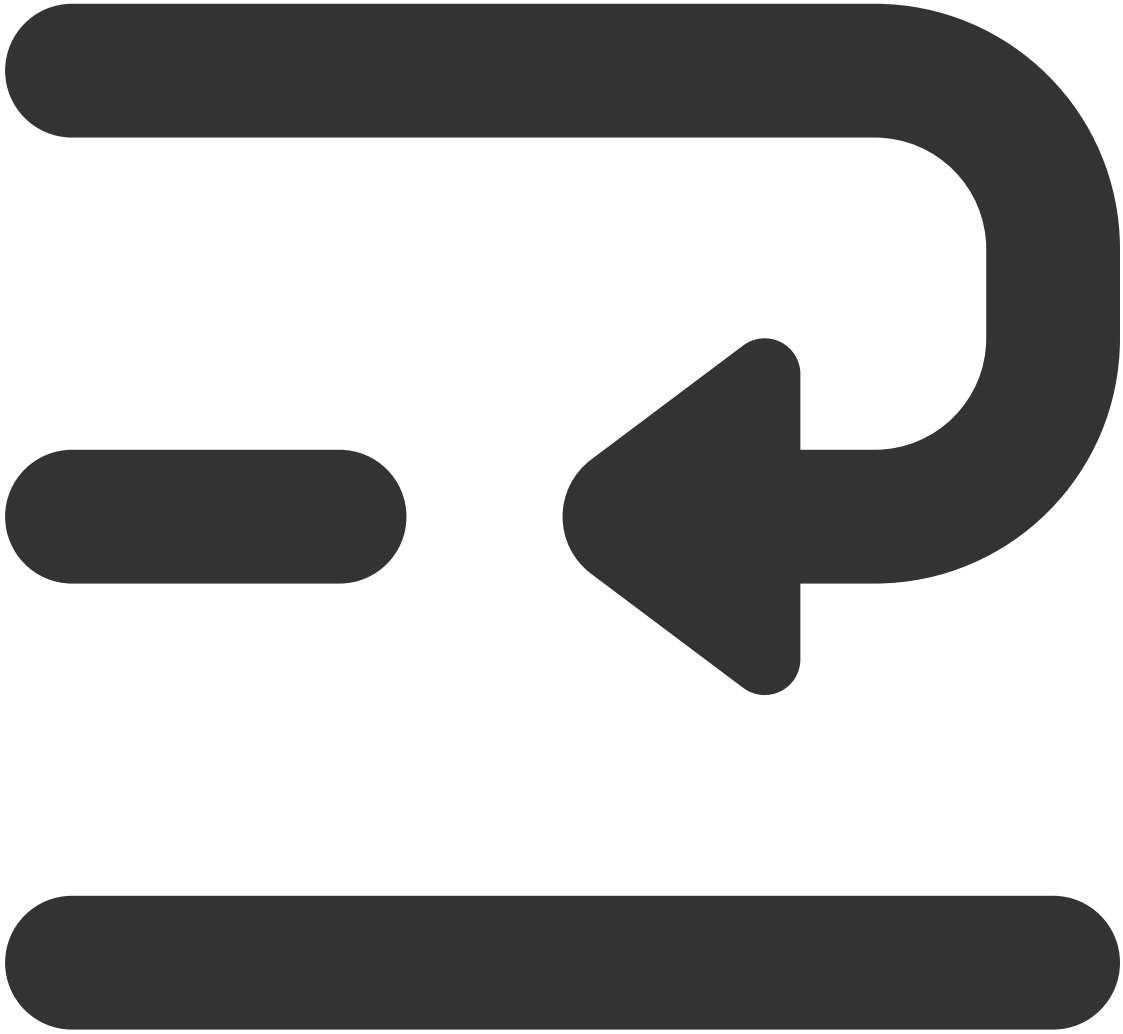
Windows

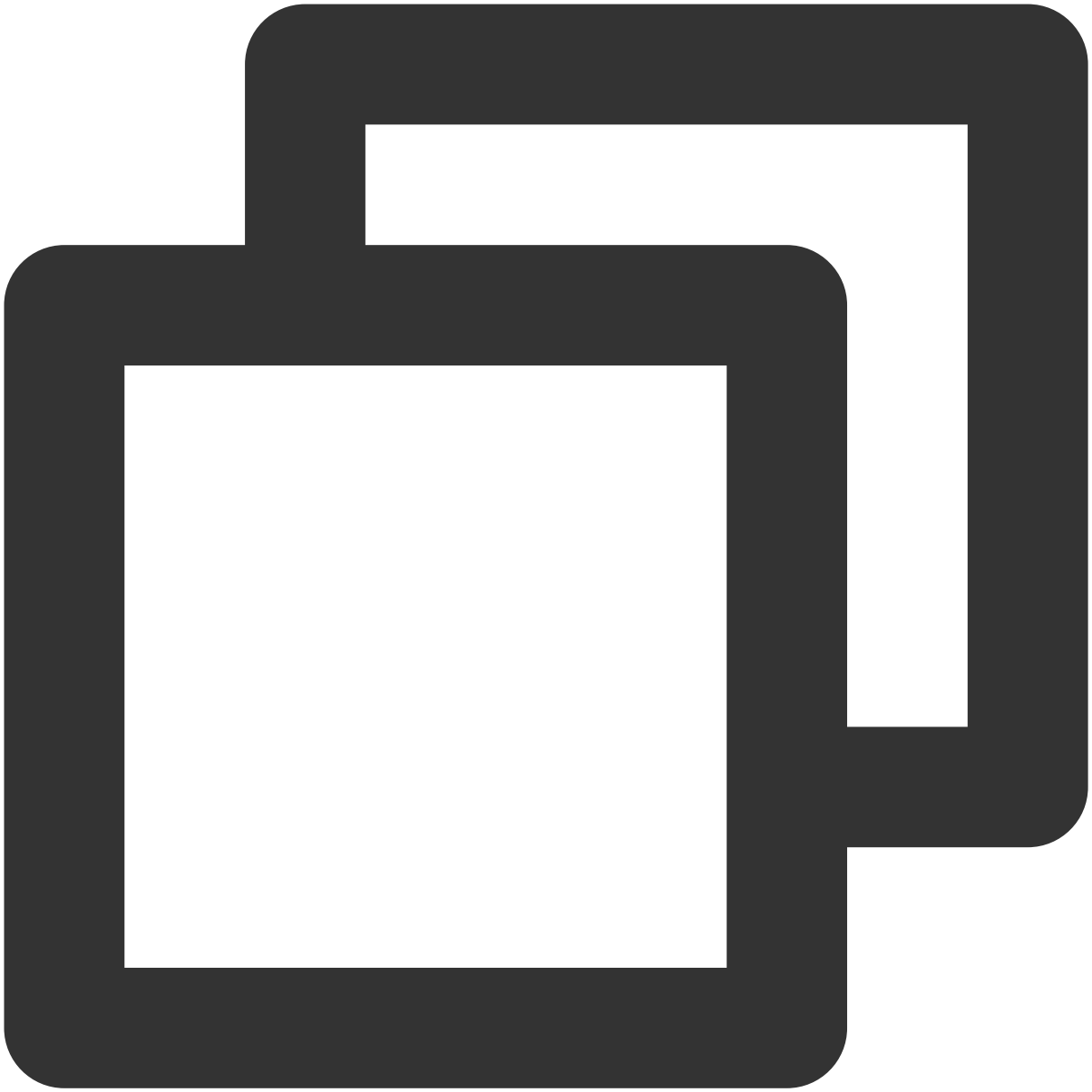
Download the TUIKit component using the npm method:



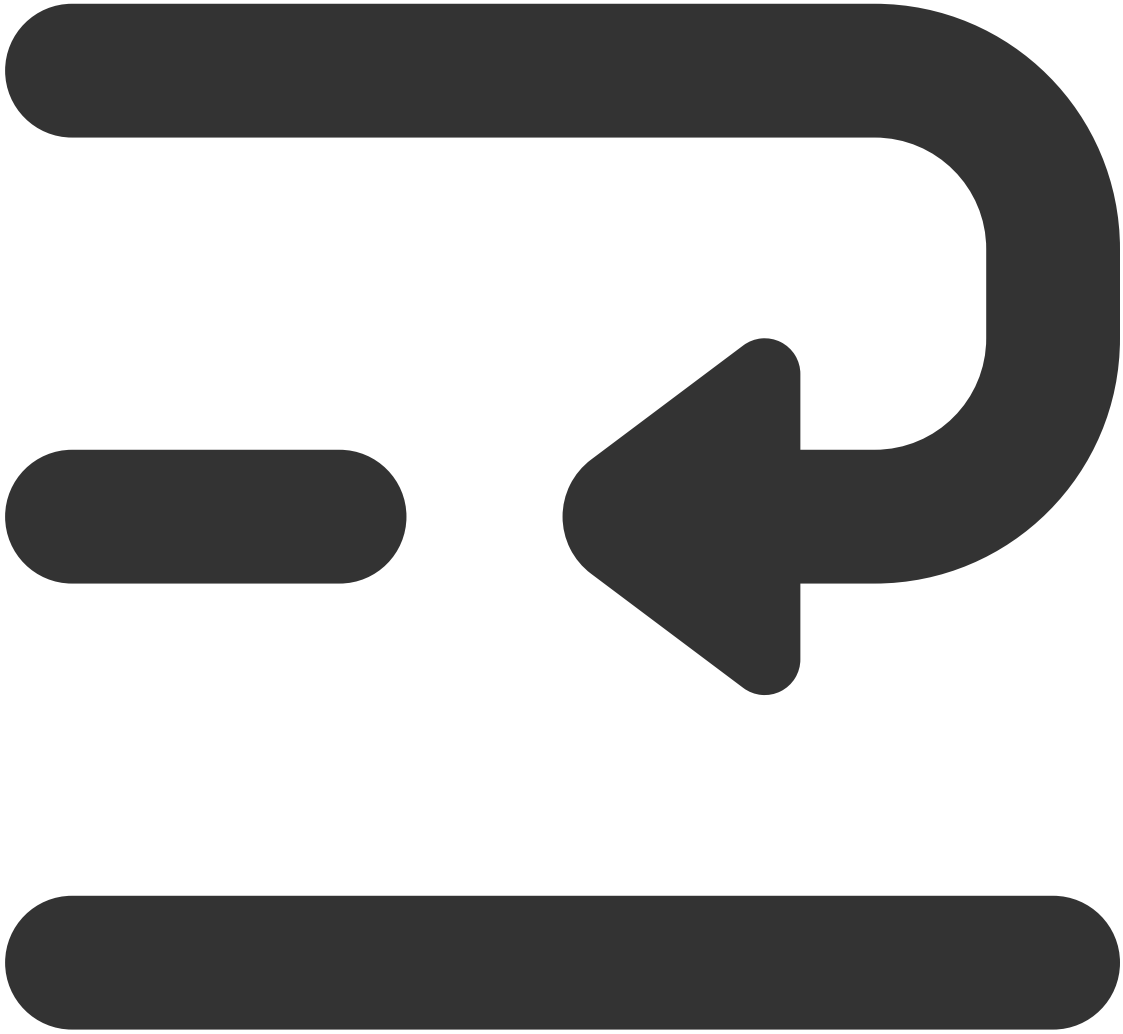
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

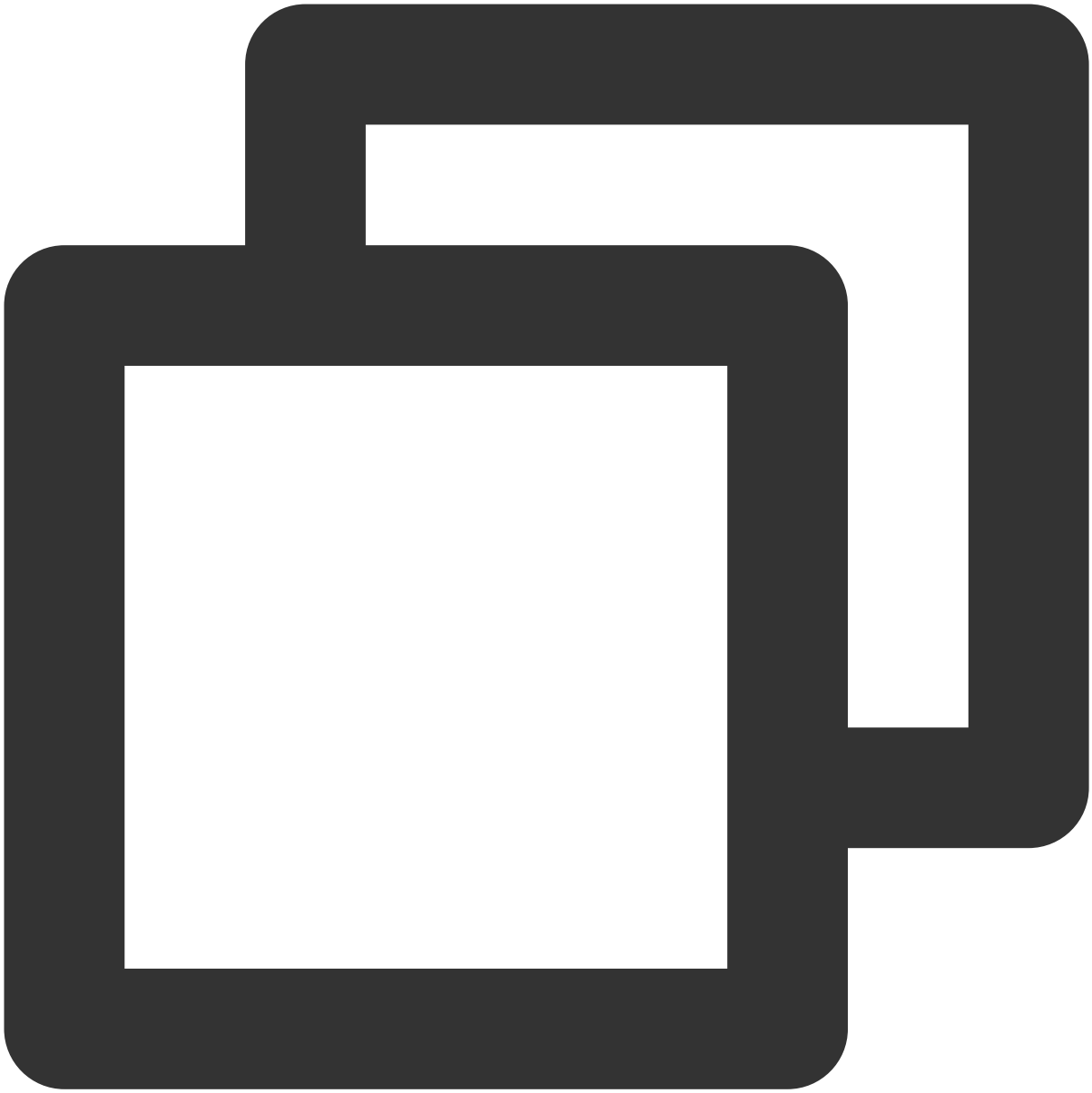
For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:





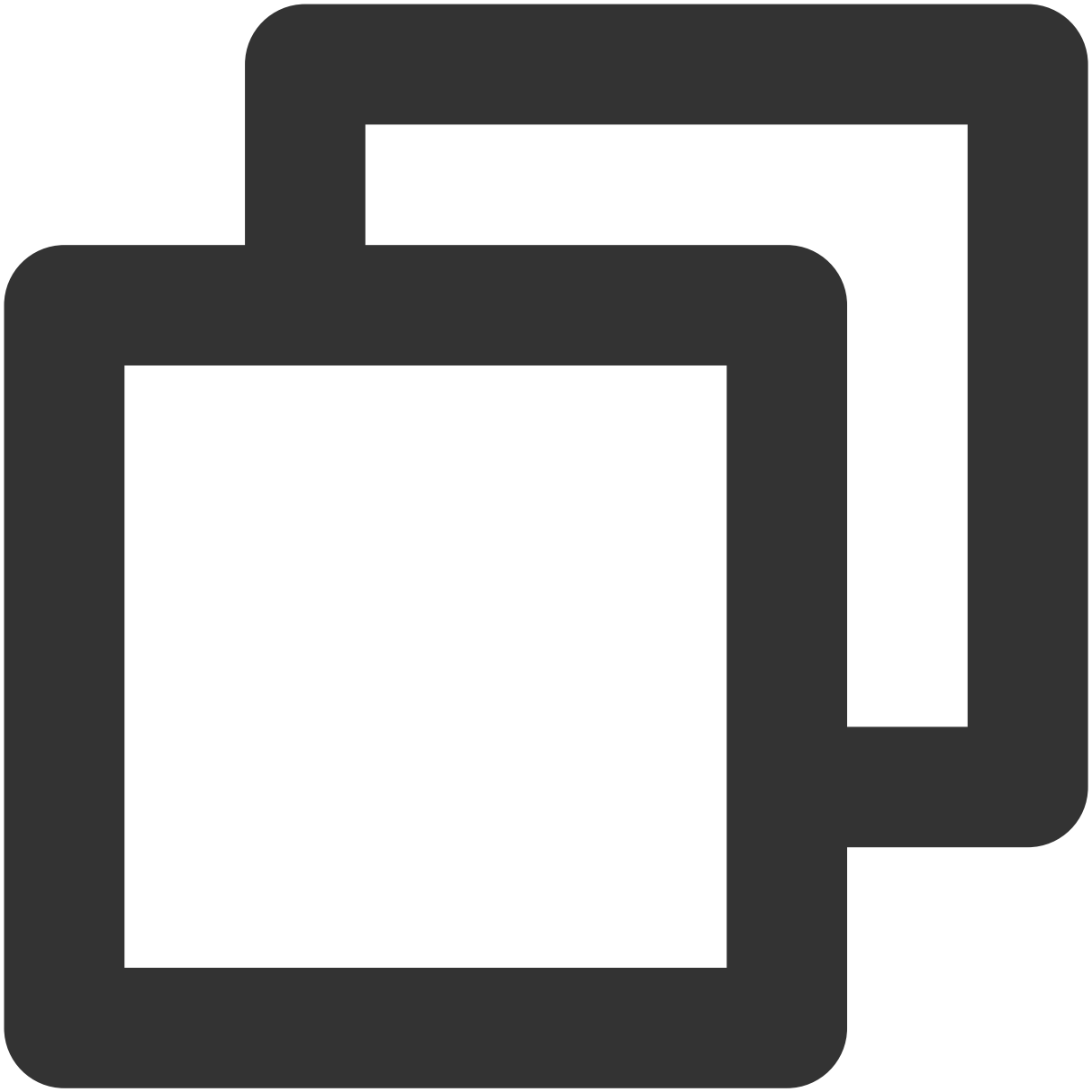
```
mkdir -p ./TUIKit && rsync -av --exclude={'node_modules','package.json','excluded-l
```





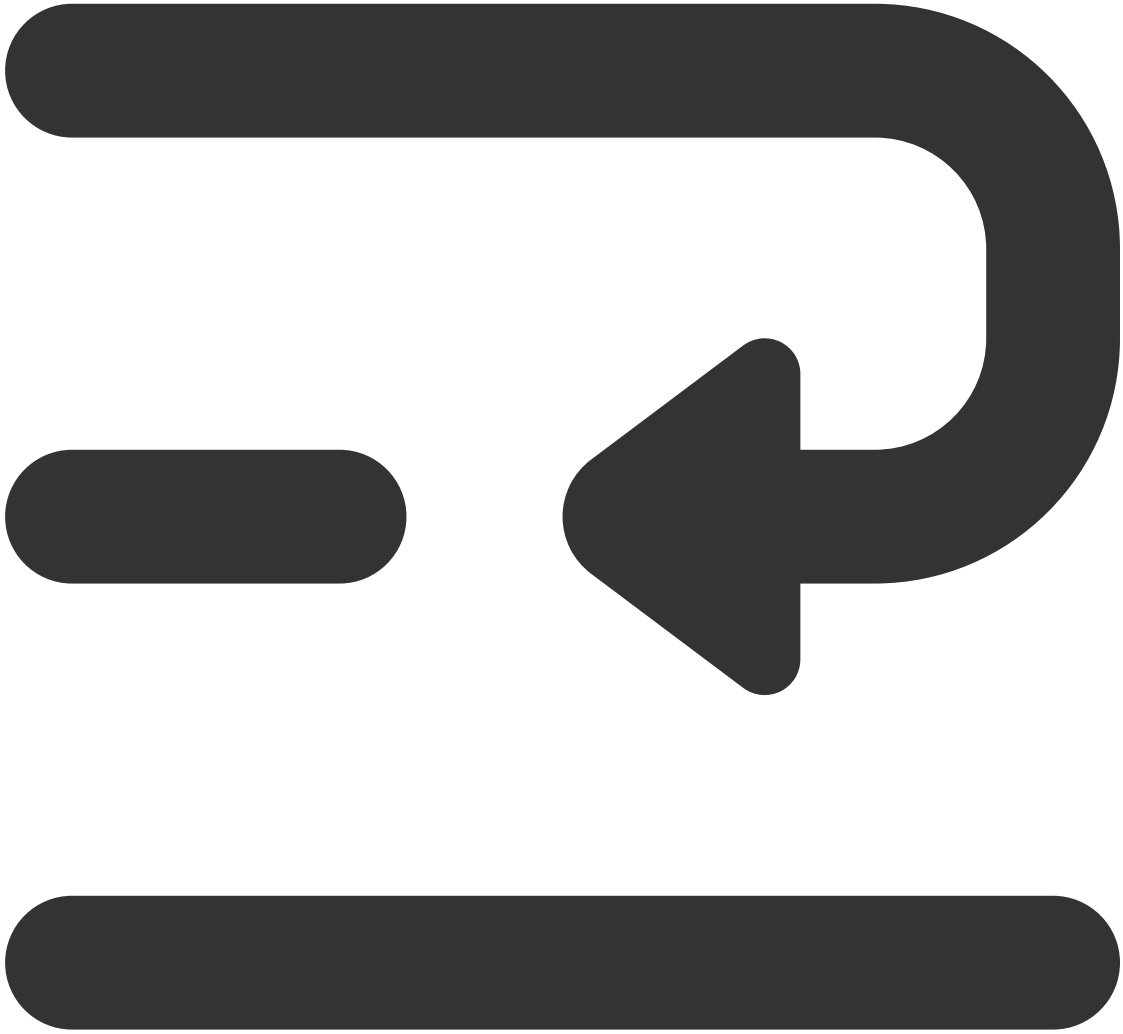
```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av ./node_modules/@tencentc
```

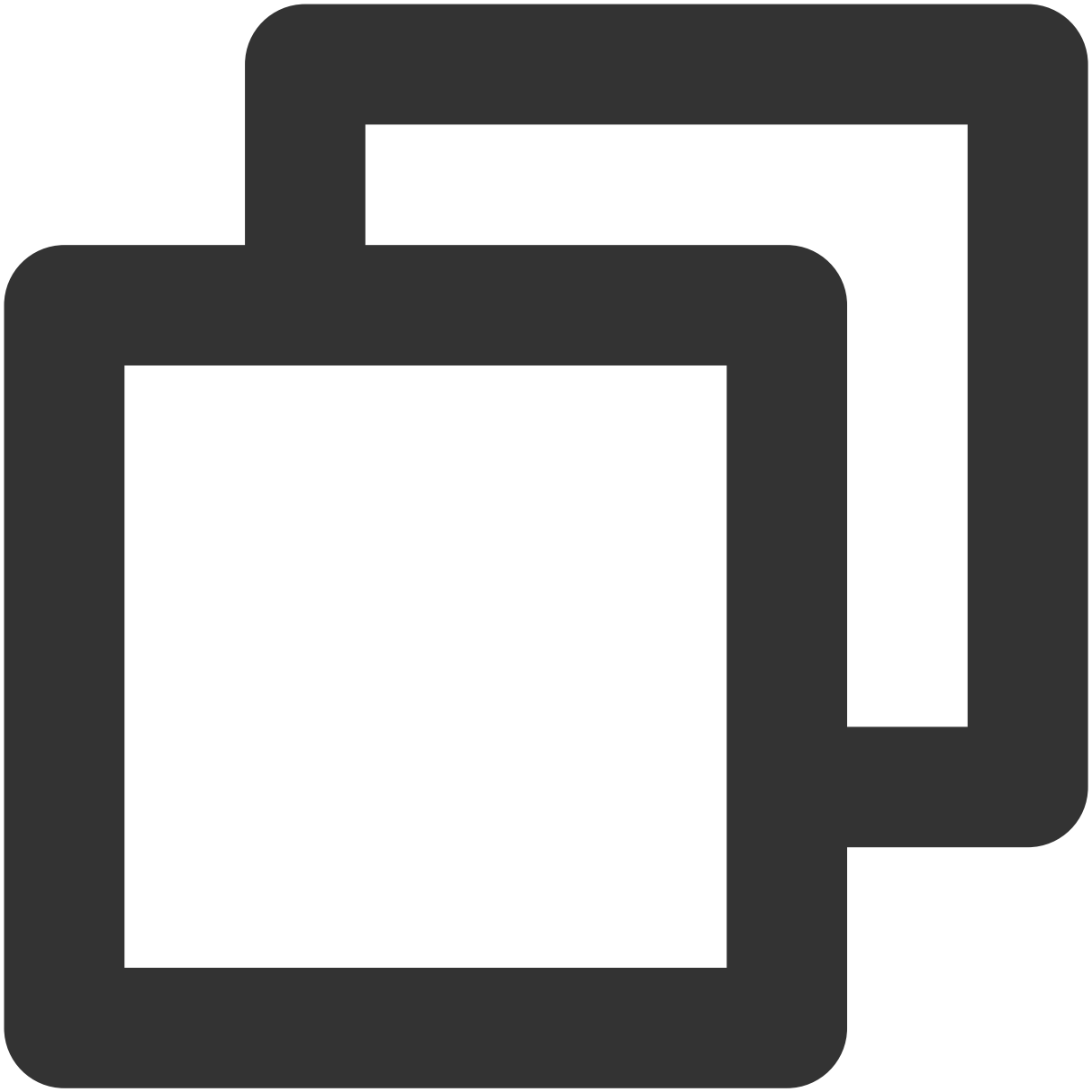
Download the TUIKit component using the npm method:



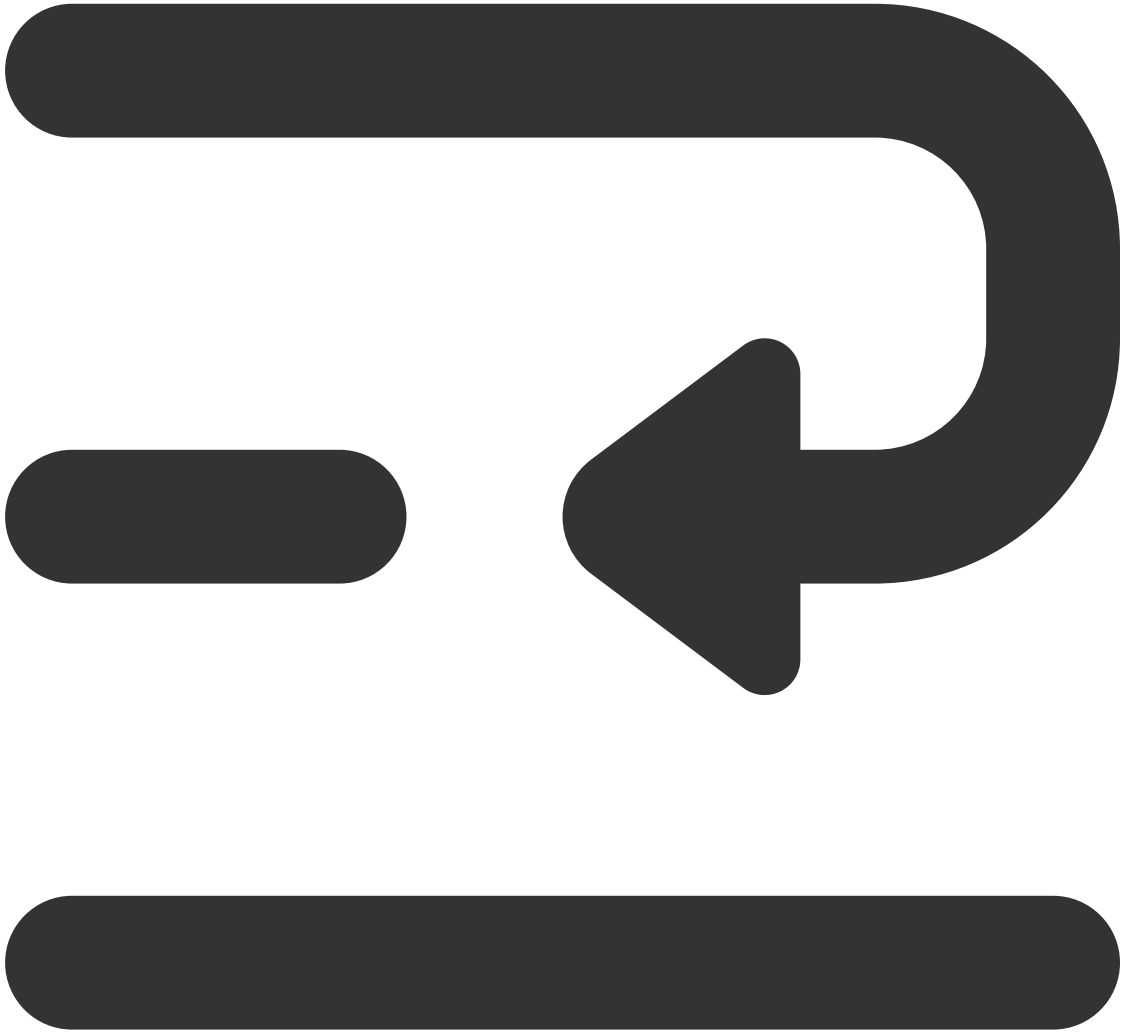
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

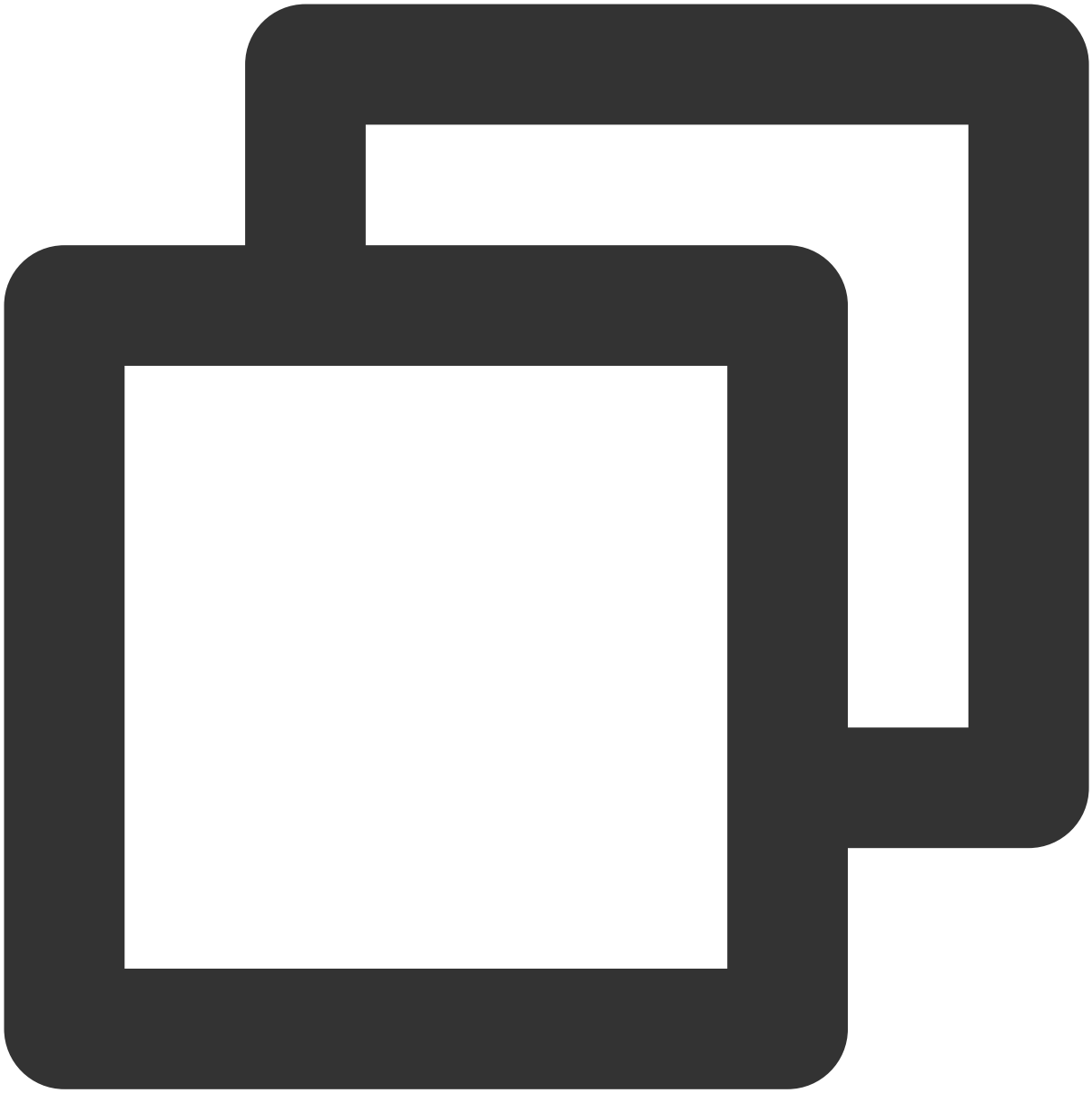
For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:





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



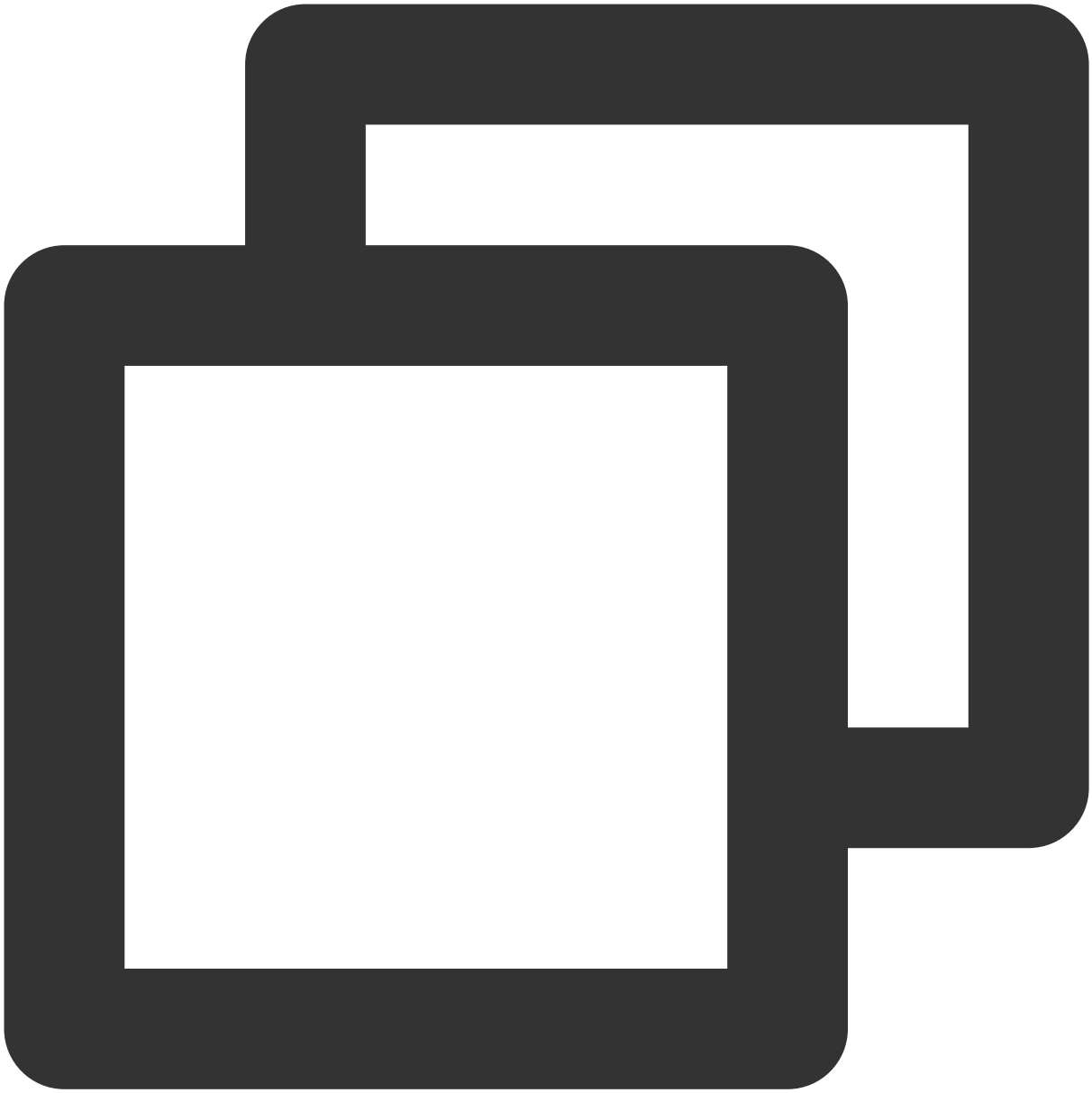


```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin .\\TUIKit\\tui-cu
```

Step 3: Incorporate the TUIKit component

1. Project Configuration

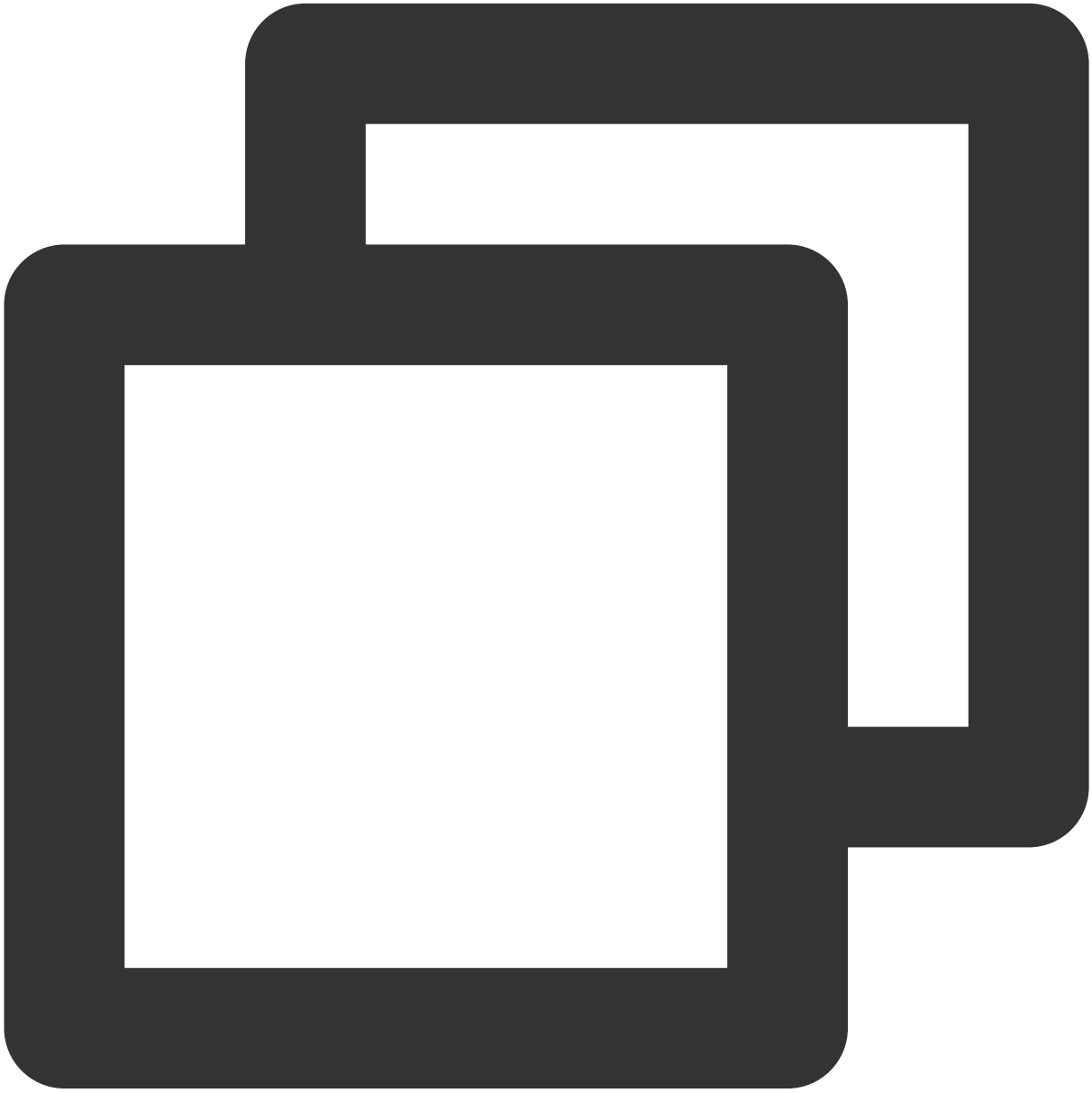
In the root directory, create vue.config.js (For Vue3 projects, please disregard this part).



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

Activate the split package configuration in the source code view of the `manifest.json` file



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

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

2. Merge TUIKIt

Note:

Pursue the integration stringently in **Four Steps**. If you wish to package a Mini Program, please do not bypass the configuration of the "Home page of Mini Program Sub-package".

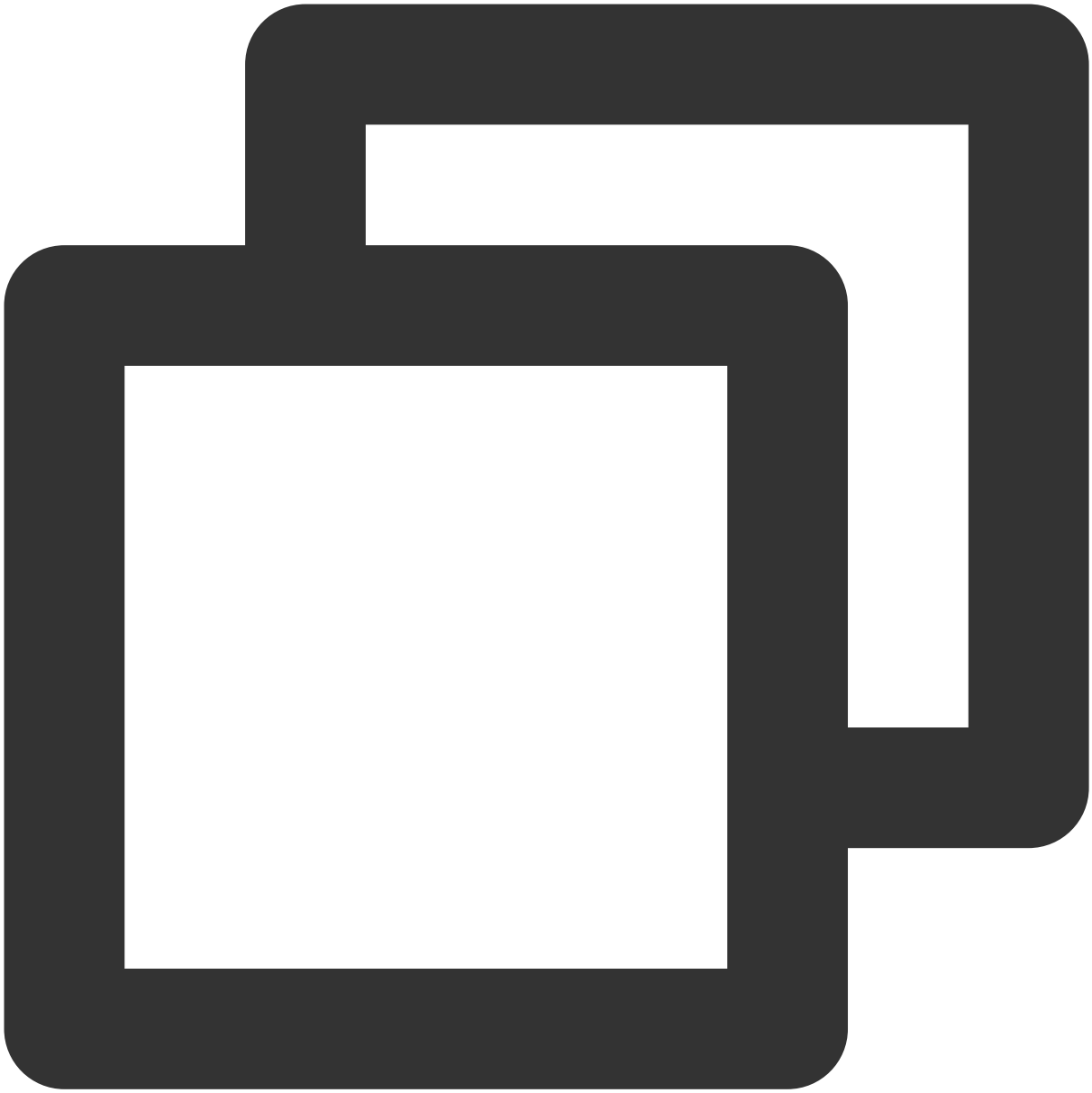
main.js file

pages.json file

App.vue file

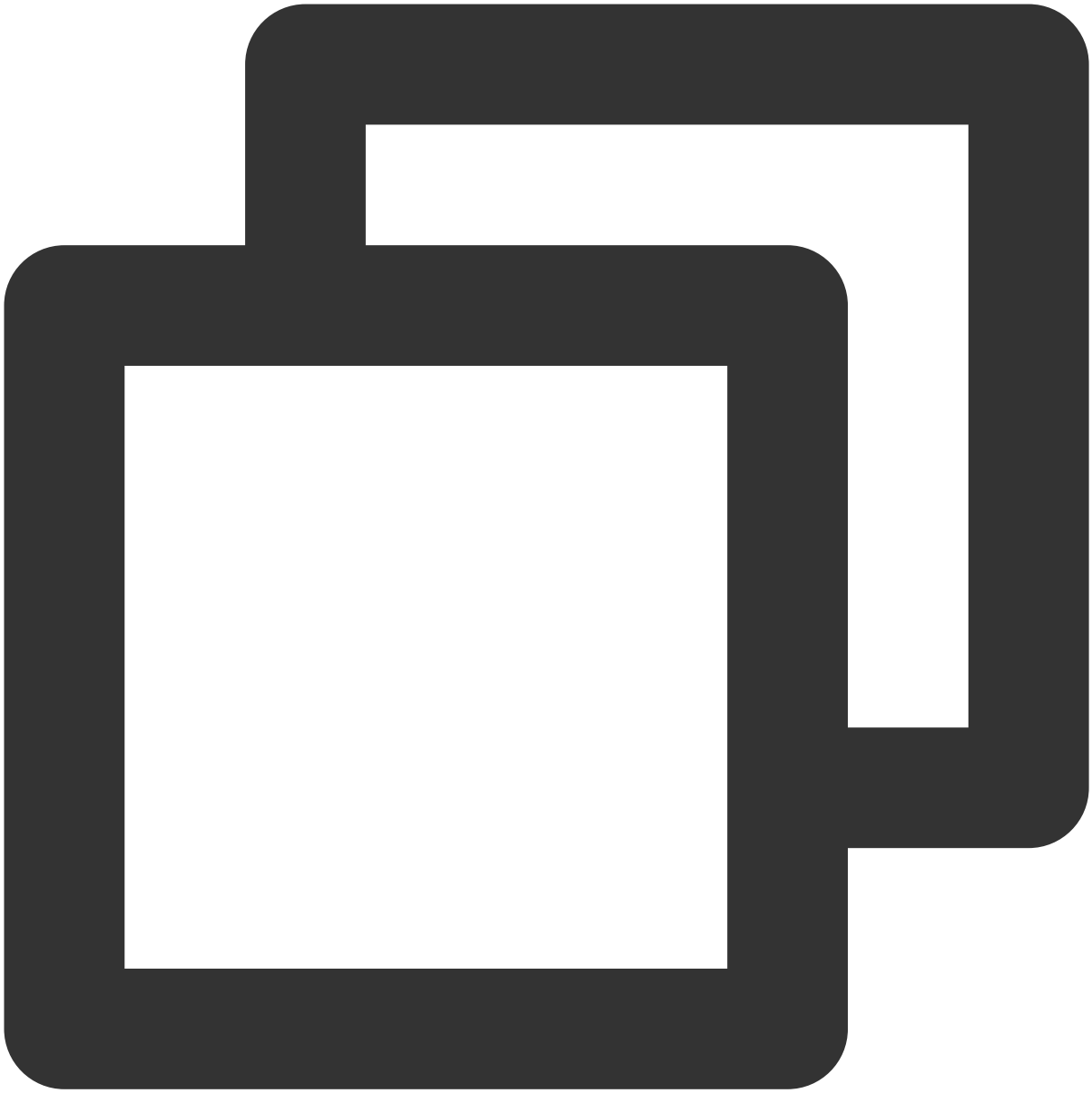
Mini Program Sub-package Home Page

Pay heed, under Vue2 environment, make use of `Vue.use(VueCompositionAPI)` , to prevent inability to use environment variables such as `isPC` .



```
// Introduce the main package dependency
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);
```

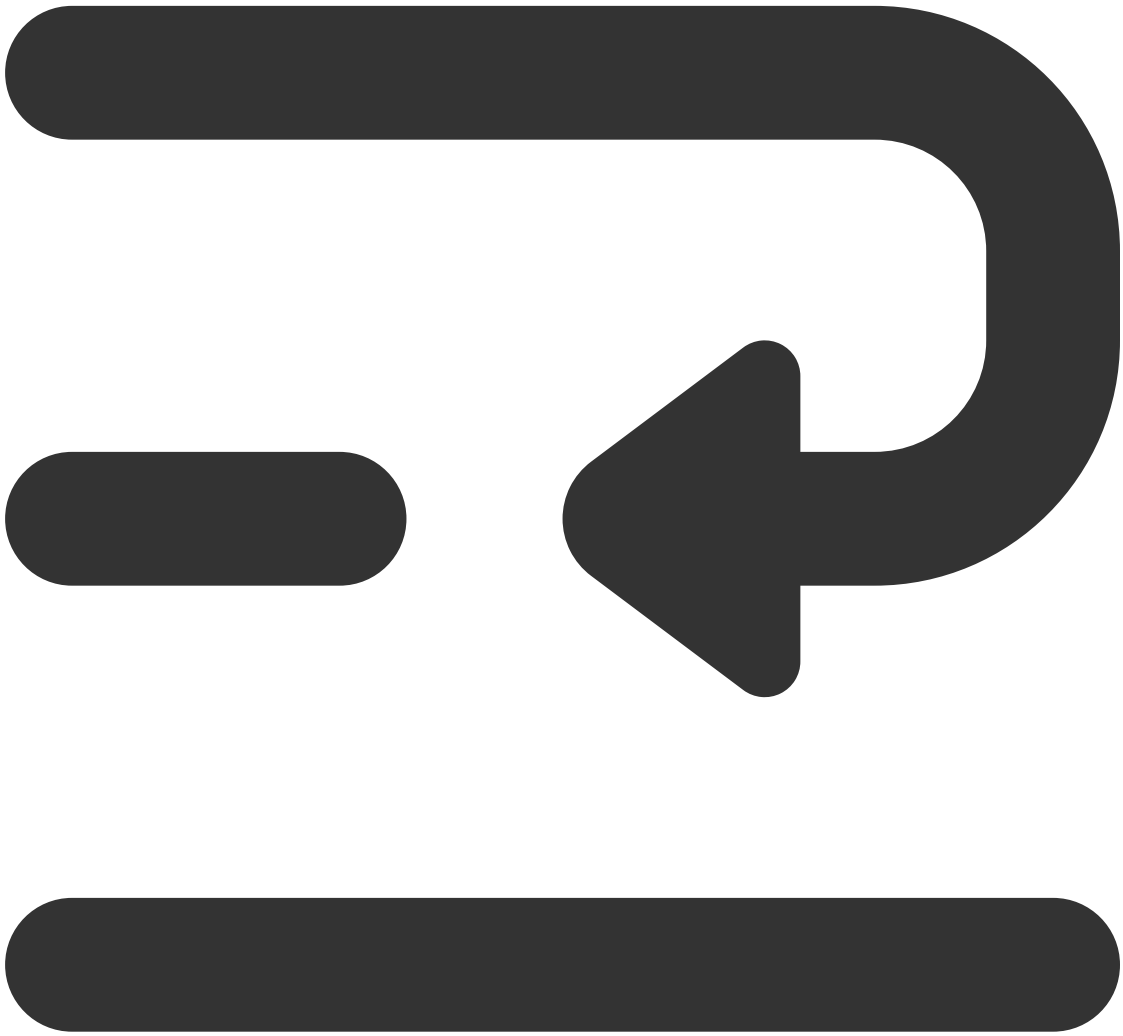
```
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

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

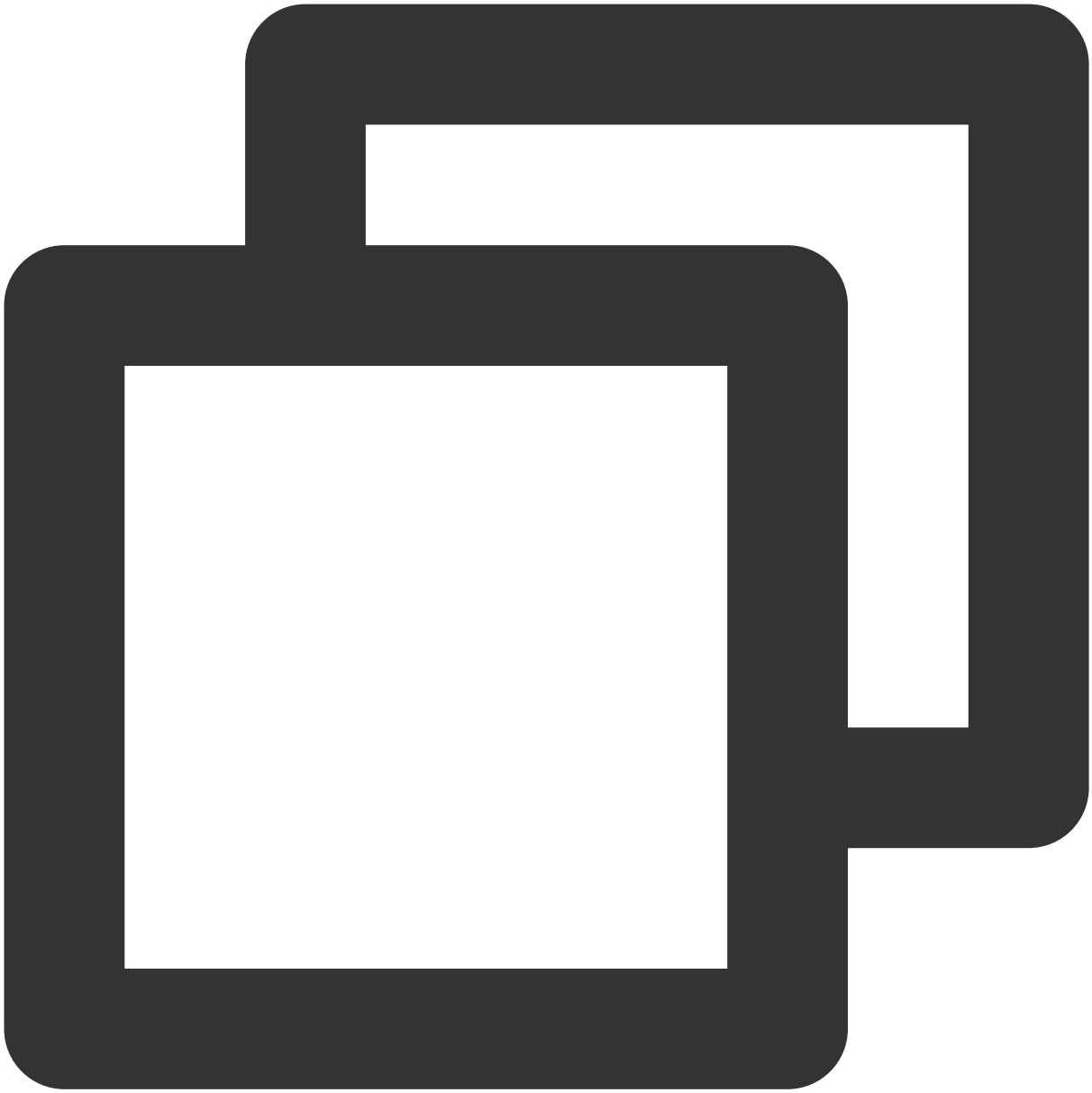


```
{
 "pages": [{
  "path": "pages/index/index" // Your project's homepage
 }],
 "subPackages": [{
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIConversation/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
```

```
     }
    },
    {
     "path": "components/TUIChat/index",
     "style": {
      "navigationBarTitleText": "Tencent Cloud IM"
     }
    },
     // To integrate the chat component, this path must be configured: video playbac
    {
     "path": "components/TUIChat/video-play",
     "style": {
      "navigationBarTitleText": "Tencent Cloud IM"
     }
    },
    {
     "path": "components/TUIChat/web-view",
     "style": {
      "navigationBarTitleText": "Tencent Cloud IM"
     }
    },
    {
     "path": "components/TUIContact/index",
     "style": {
      "navigationBarTitleText": "Tencent Cloud IM"
     }
    },

    {
     "path": "components/TUIGroup/index",
     "style": {
      "navigationBarTitleText": "Tencent Cloud 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
// Mandatory information
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.$chat_userSig = genTestUserSig(config).userSig;
// Initialization of TUIChatKit
TUIChatKit.init();
// #endif
export default {
  onLaunch: function () {
    // #ifdef APP-PLUS || H5
    // TUICore login
    TUILogin.login({
      SDKAppID: uni.$chat_SDKAppID,
      userID: uni.$chat_userID,
      // A UserSig is a cipher for users to sign in to Instant Messaging - it is es
      // This method is only suitable for running demos locally and debugging featu
      userSig: uni.$chat_userSig,
      // Should you require to transmit imagery, audio, video, files, and other for
      useUploadPlugin: true,
      // Local audit can identify and handle unsuitable and unsafe content to effec
      // This feature is an added service, please refer to: https://cloud.tencent.c
      // If you've purchased the content review service, please enable this feature
      useProfanityFilterPlugin: false,
      framework: `vue${vueVersion}` // Current development framework in use: vue2 /
    });
    // #endif
  },
 onShow: function() {
      console.log('App Show')
  },
 onHide: function() {
      console.log('App Hide')
  }
};
</script>
<style>
/*Common CSS for each page*/
uni-page-body,
html,
body,
page {
  width: 100% !important;
  height: 100% !important;
  overflow: hidden;
```

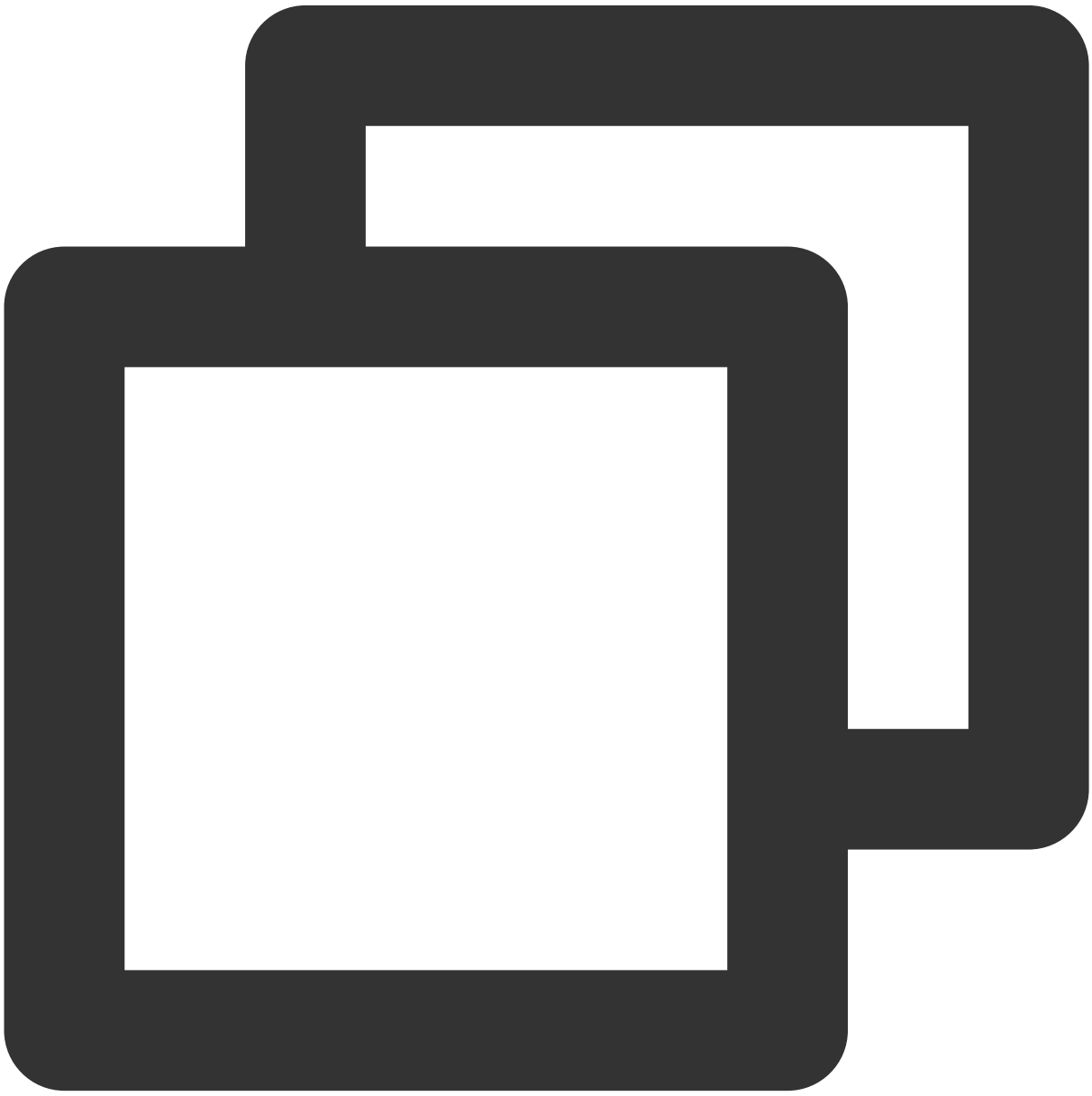
```
}
</style>
```

Note:

The mini program integrates by default in a subpackage. The login must be completed on the TUIKit startup page. If you do not require the packaging of mini-programs (for instance, building H5 only), you can disregard the configuration content of "*Mini Program Split Package Homepage*".

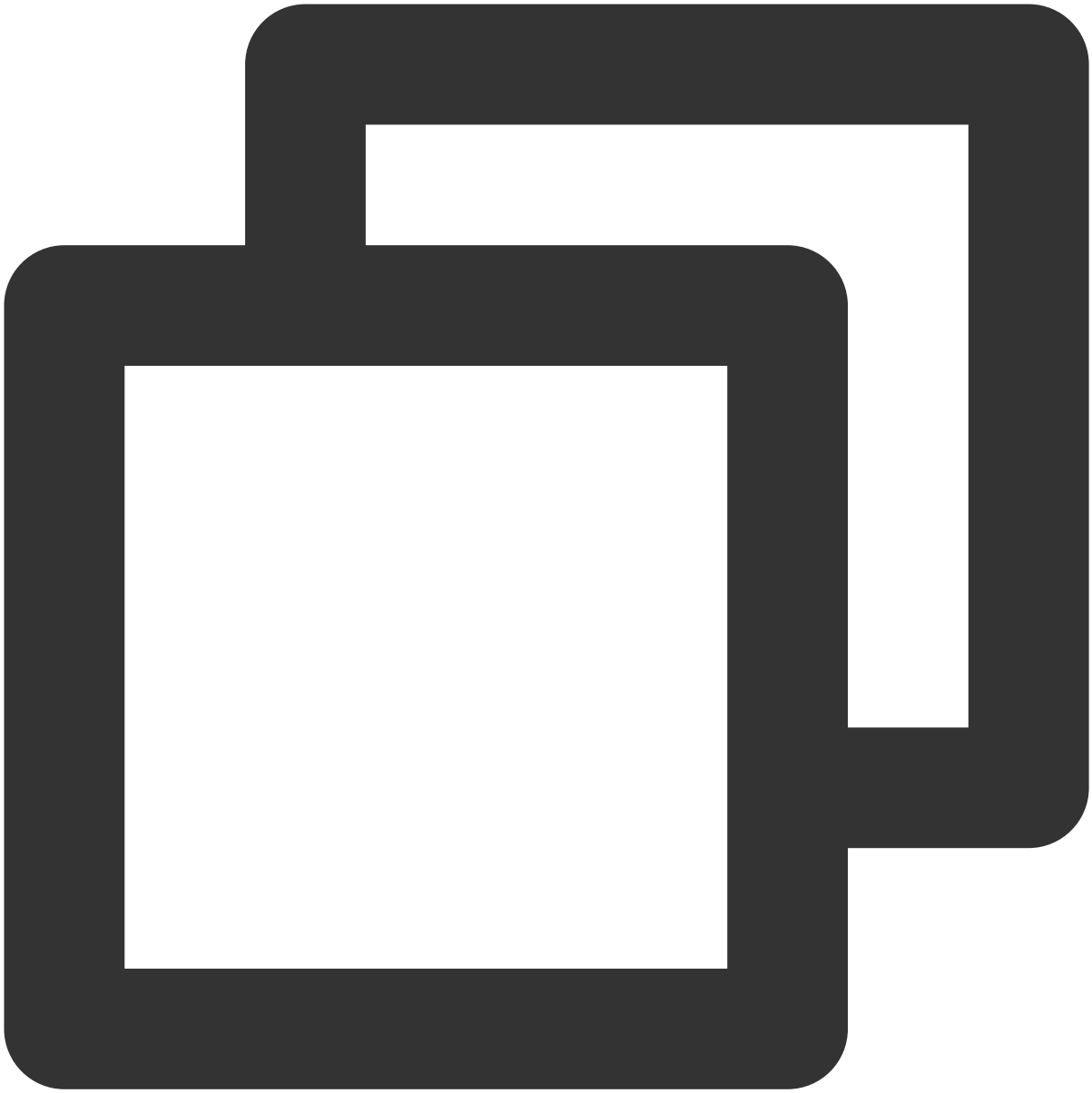
Example: The TUIKit sub-package first screen launch page is the **TUIConversation** page

Step 1: Create a subPackage-init.ts file under the TUIKit/components/TUIConversation directory



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

// Initialization of TUIChatKit
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 is the cipher for users to sign in to Instant Messaging, essentially b
  // This method is only suitable for running Demo locally and debugging functions.
  userSig: uni.$chat_userSig,
  // Should you require to send image, voice, video, file and other rich media mess
  useUploadPlugin: true,
  // Local review can successfully identify and handle inappropriate and unsafe con
  // This functionality is a value-added service, please refer to: https://cloud.te
  // If you have purchased the content review service, to activate this feature ple
  useProfanityFilterPlugin: false,
  framework: `vue${vueVersion}` // Current development uses framework vue2 / vue3
}).then(() => {
  uni.showToast({
    title: "login success"
  });
});
```

Step 2: Import within TUIKit/components/TUIConversation/index.vue



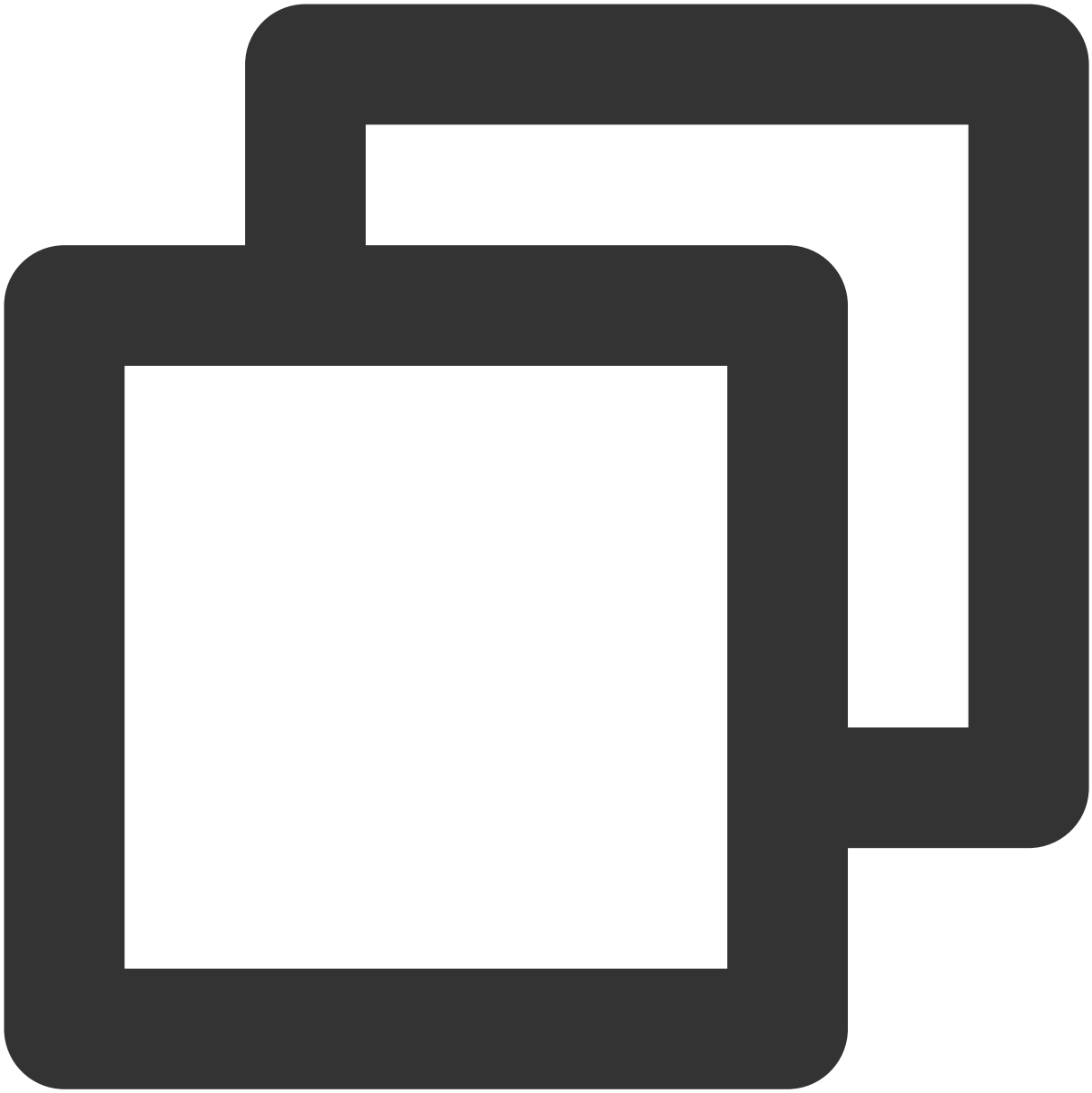
```
// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

See the following figure:

```
<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. Configuring the entry points of TUIConversation and TUIContact on the main package homepage of the project

Create an index.vue file under the pages/index folder

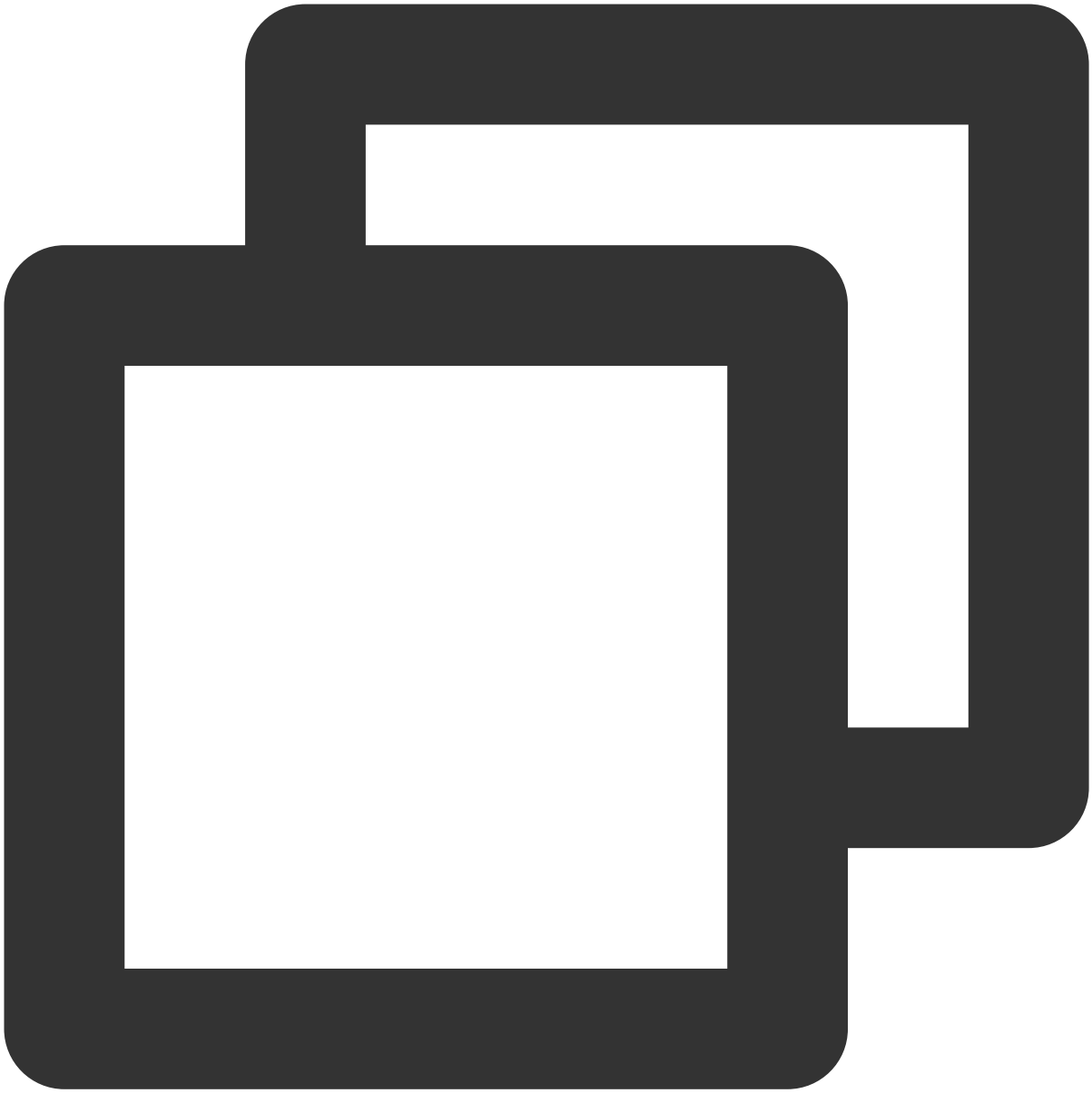


```
<template>
  <div class="index">
        <p class="index-button" @click="openConversation">Open TUIKit Conversation<
        <p class="index-button" @click="openContact">Open TUIKit Contacts</p>
  </div>
</template>
<script>
export default {
methods: {
  // Open the TUIKit session list
  openConversation() {
```

```
    uni.navigateTo({
          url: "/TUIKit/components/TUIConversation/index",
        });
  },
  // Accessing TUIKit Contacts
  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>
```

Step 4: Gain access to SDKAppID, secretKey, and userID

Within the App.vue file in the root directory of configuration, find `config` object's SDKAppID, secretKey, and userID. The SDKAppID and secretKey can be accessed through the Instant Messaging Console, and the userID can be accessed when creating an account in the Instant Messaging Console.



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

access SDKAppID,secretKey

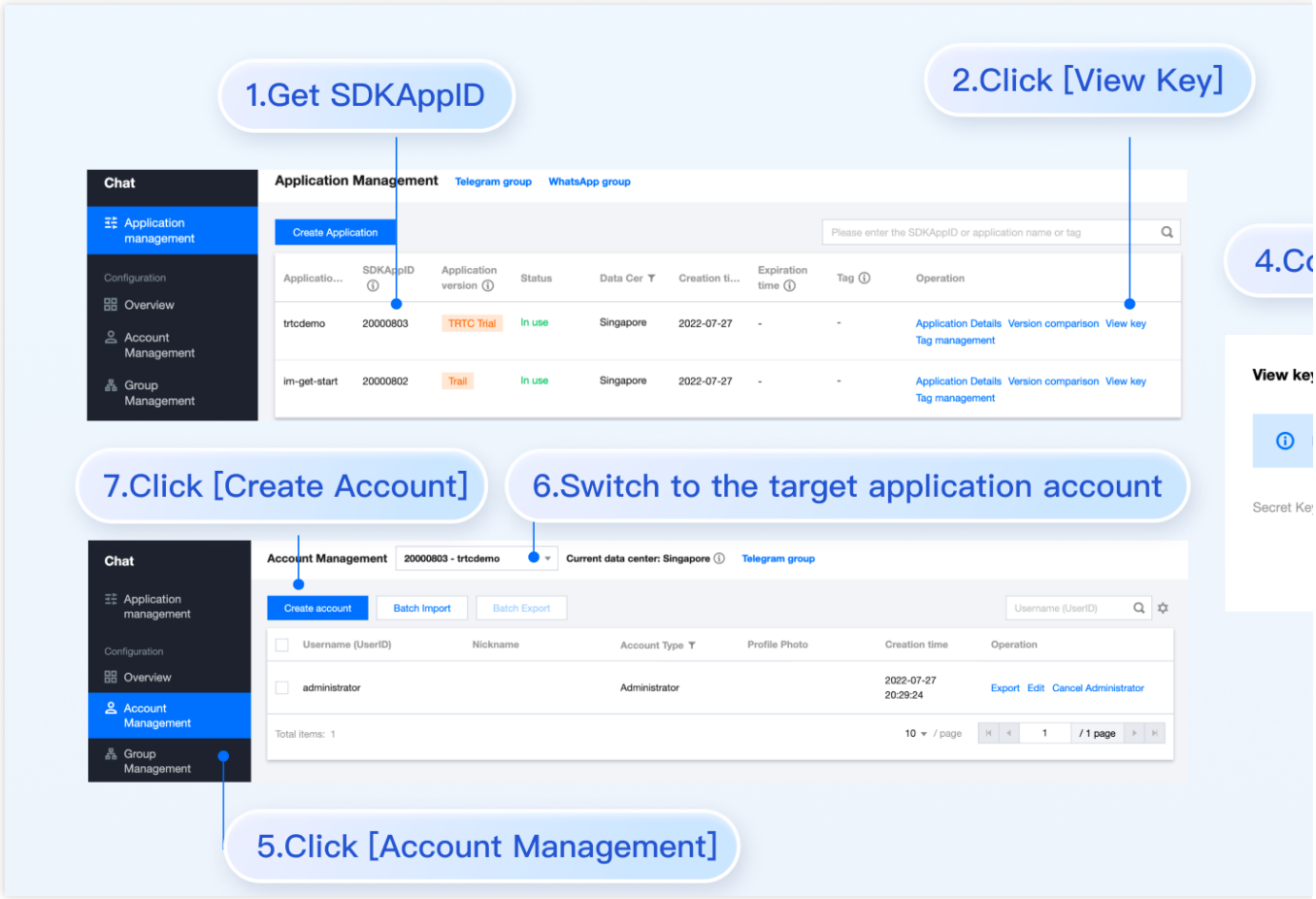
In the Instant Messaging Console under the **application management** page, you can see the applications you have created. The SDKAppID is in the second column. Then click on the **peekKey** in the operation options. A dialogue box will appear on the website for the peekKey, and by clicking on the **Show Key**, the peekKey will be revealed.

Create an account with `userID` as `test-user1`

Click on **Account Management** on the left side of the console. If you have multiple applications, ensure to switch to your current application. Then, under the current application, click **Create new account** to create an account with a userID of `test-user1`.

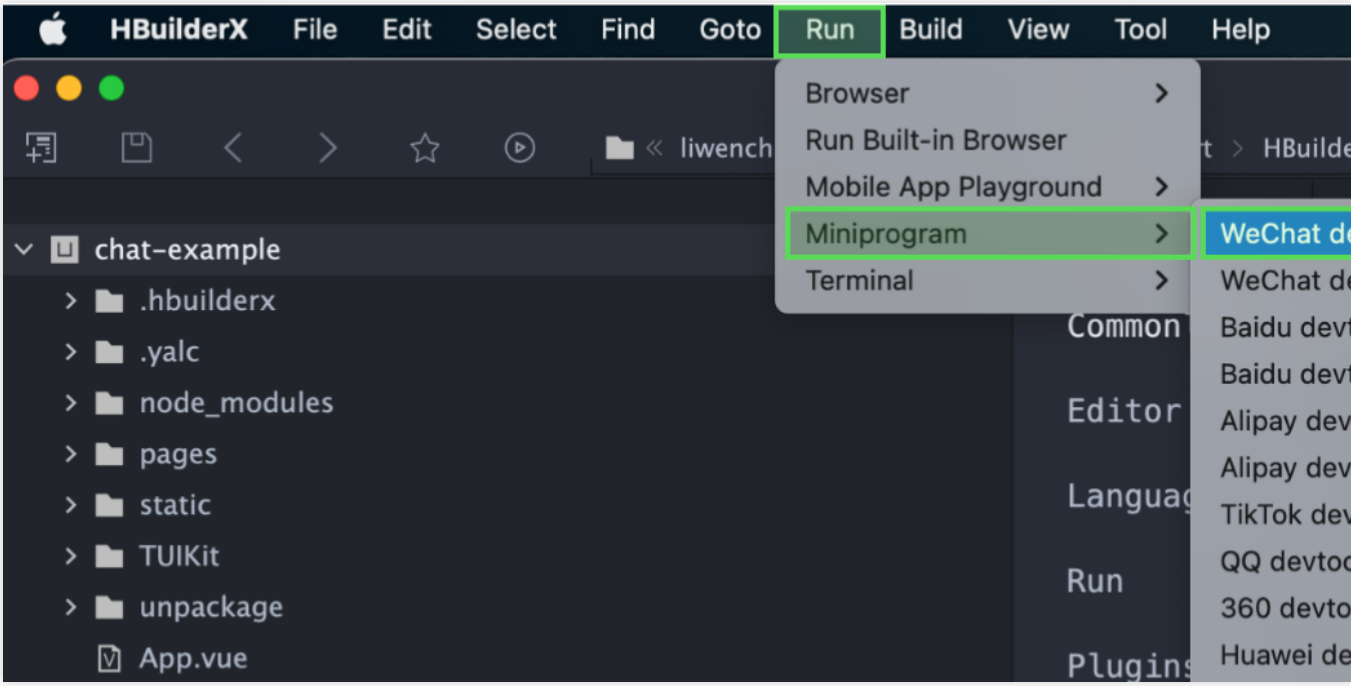
Note:

The step of creating an account can be circumvented as TUIKit will auto-generate an account during the sign in process if the configuration's userID does not exist. This only demonstrates how to access the userID.



Step 5. Launch the project

1. Launch the project using HBuilderX, then click on "Run - Run to Mini Program Simulator - WeChat Developer Tools".



2. Should HBuilderX fail to automatically activate the WeChat Developer Toolkit, kindly use the toolkit to manually open the compiled project.

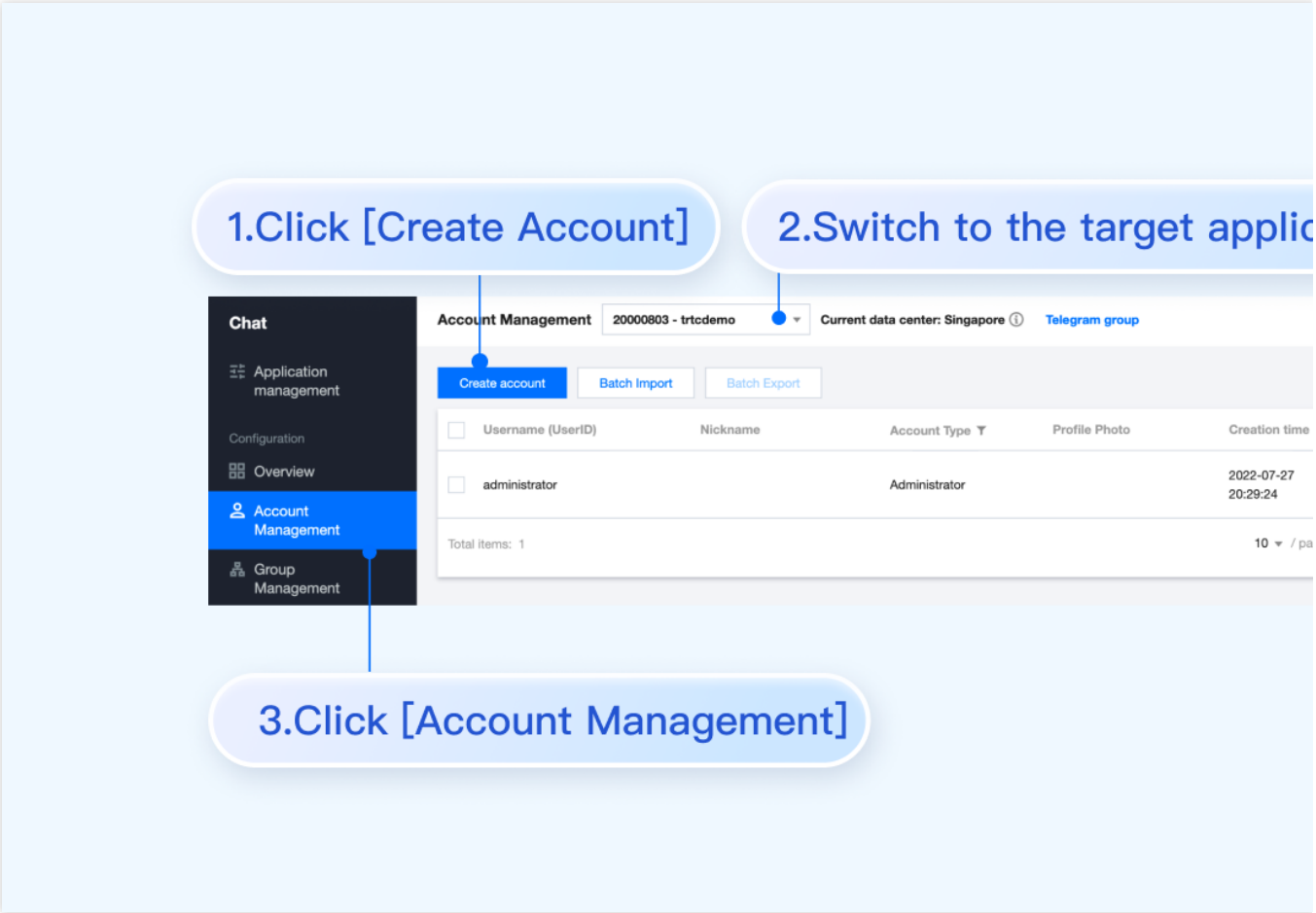
Open the `unpackage/dist/dev/mp-weixin` under the project root directory using the WeChat Developer Tool.

3. After opening the project, check the "Do not verify valid domain, web-view (business domain), TLS version, and HTTPS certificate" in "Details-Local Setting" of WeChat Developer Tools.

Step 6. Send your first message

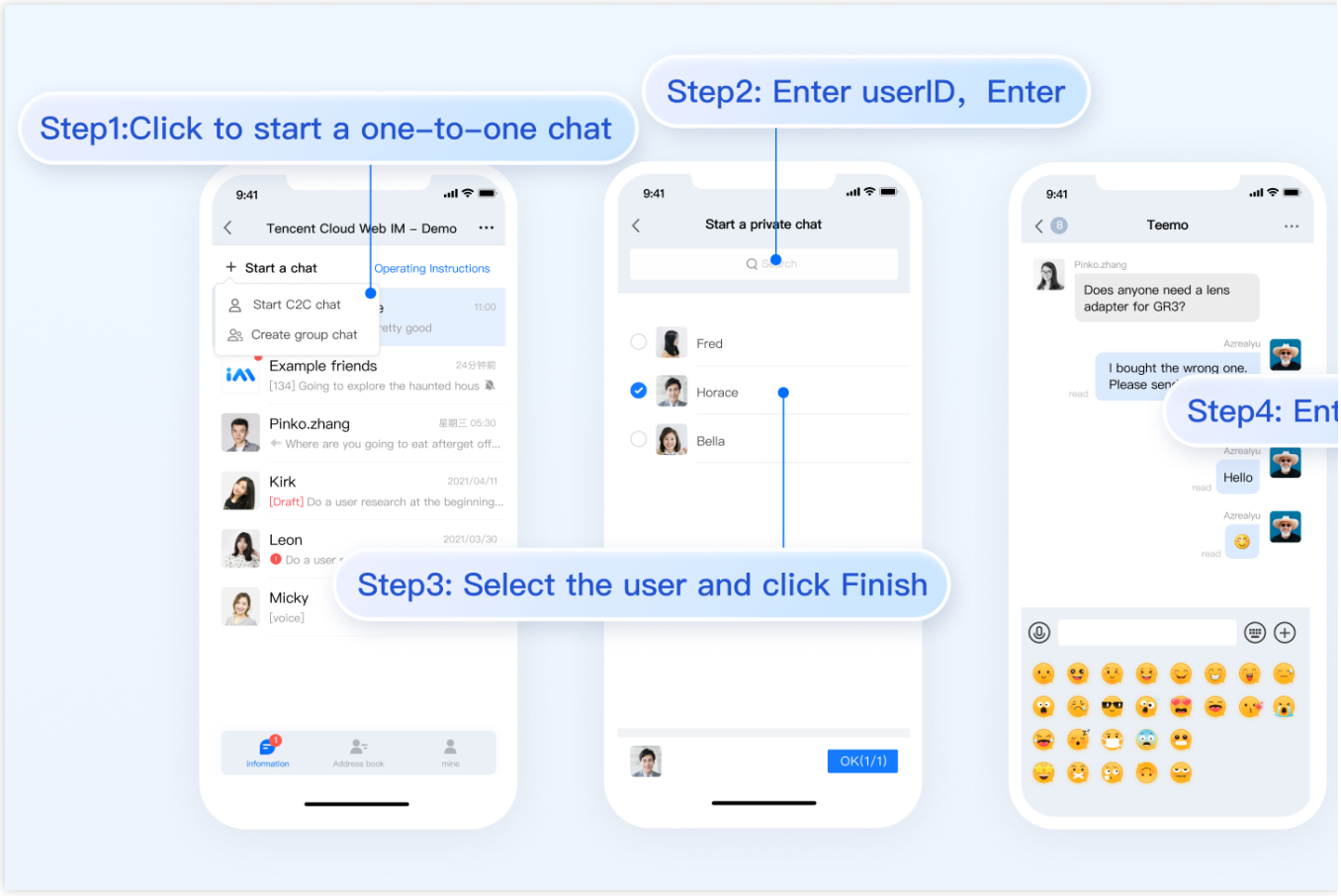
1. Create a User account through the Instant Messaging Console

Navigate to the **Account Management** page from the left sidebar, and click on **New Account** to create a regular account with userID:test-user2.



2. Run project and create conversation.

click **to open TUIKit conversation** , search for user userID:test-user2, and send your first message.



Additional Advanced Features

Audio-Visual Communication TUICallKit Plugin

Note:

The TUICallKit audio/video component is not integrated by default in TUIKit,TUICallKit primarily handles voice and video calls.

Should you need to integrate call functionalities, kindly refer to the following documents for guidelines.

For packaging into APP, refer to: Audio/Video Calling (Client)

For packaging to Miniprogram, please refer to: Video Calls(Miniprogram)

For packaging into HTML5, please refer to the official documentation: Audio and Video Calls (HTML5)

Please stay tuned.

TIMPush Offline Push Plugin

Indication

By default, TUIKit does not integrate the TIMPush **offline push plugin**.TIMPush is Tencent Cloud's Instant Messaging Push Plugin. Currently, offline push supports Android and iOS platforms, and devices include: Huawei,

Xiaomi, OPPO, Vivo, Meizu, and Apple phones.
Should you require the integration of offline push capabilities within your APP, kindly refer to the implementation of uni-app offline push.
Please stay tuned.

Individually integrate TUIChat component

Consider Independent Integration of TUIChat Component as a Solution

FAQs

For additional inquiries, please refer to Uniapp FAQ.

Exchange and Feedback

Click here to join the IM community, where you'll receive support from experienced engineers to help overcome your challenges.

Reference Documentation

Related to UIKit (vue2 / vue3):
Source code of chat-uikit-uniapp (vue2/vue3) on GitHub
Rapid Incorporation of chat-uikit-uniapp npm
Regarding ChatEngine:
ChatEngine API Manual
ChatEngine npm

Web & H5（Vue）

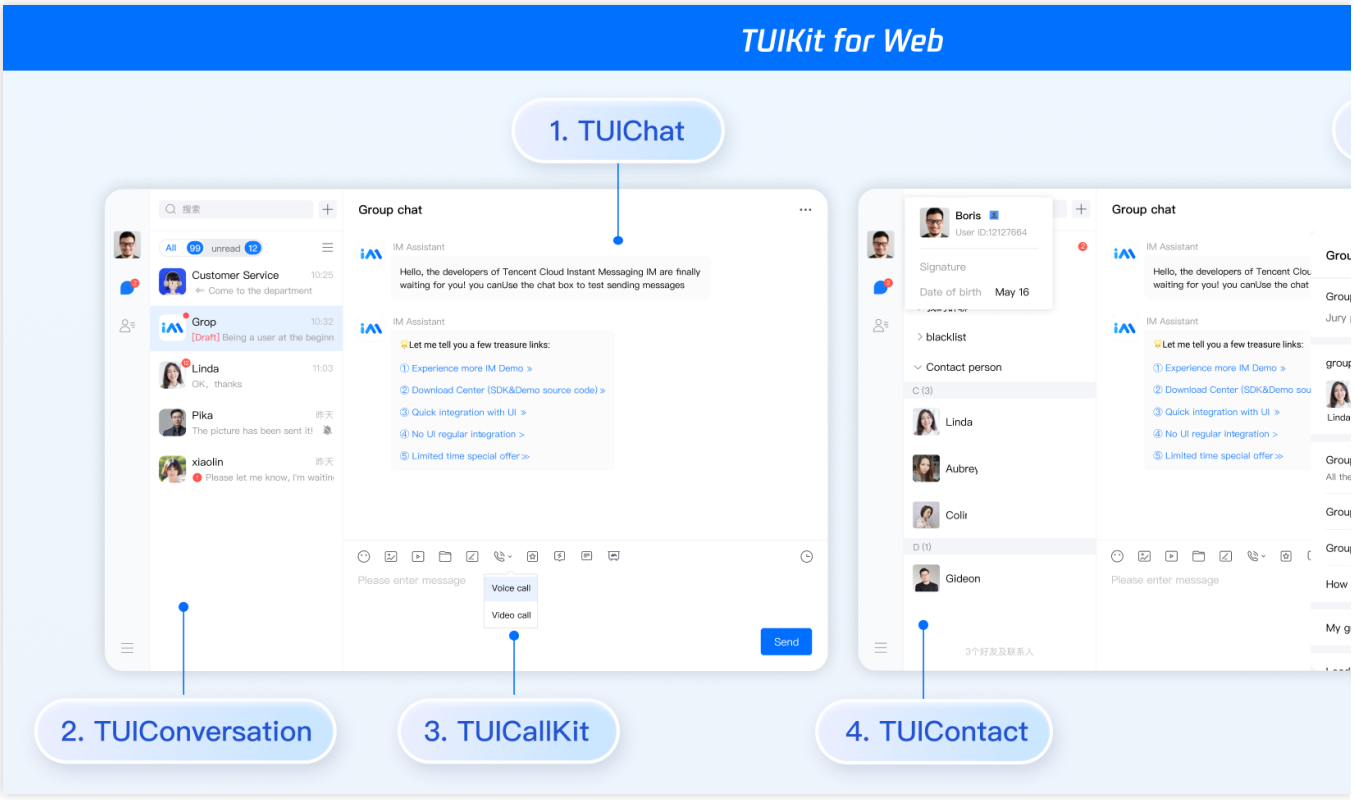
最終更新日：：2024-01-31 17:47:49

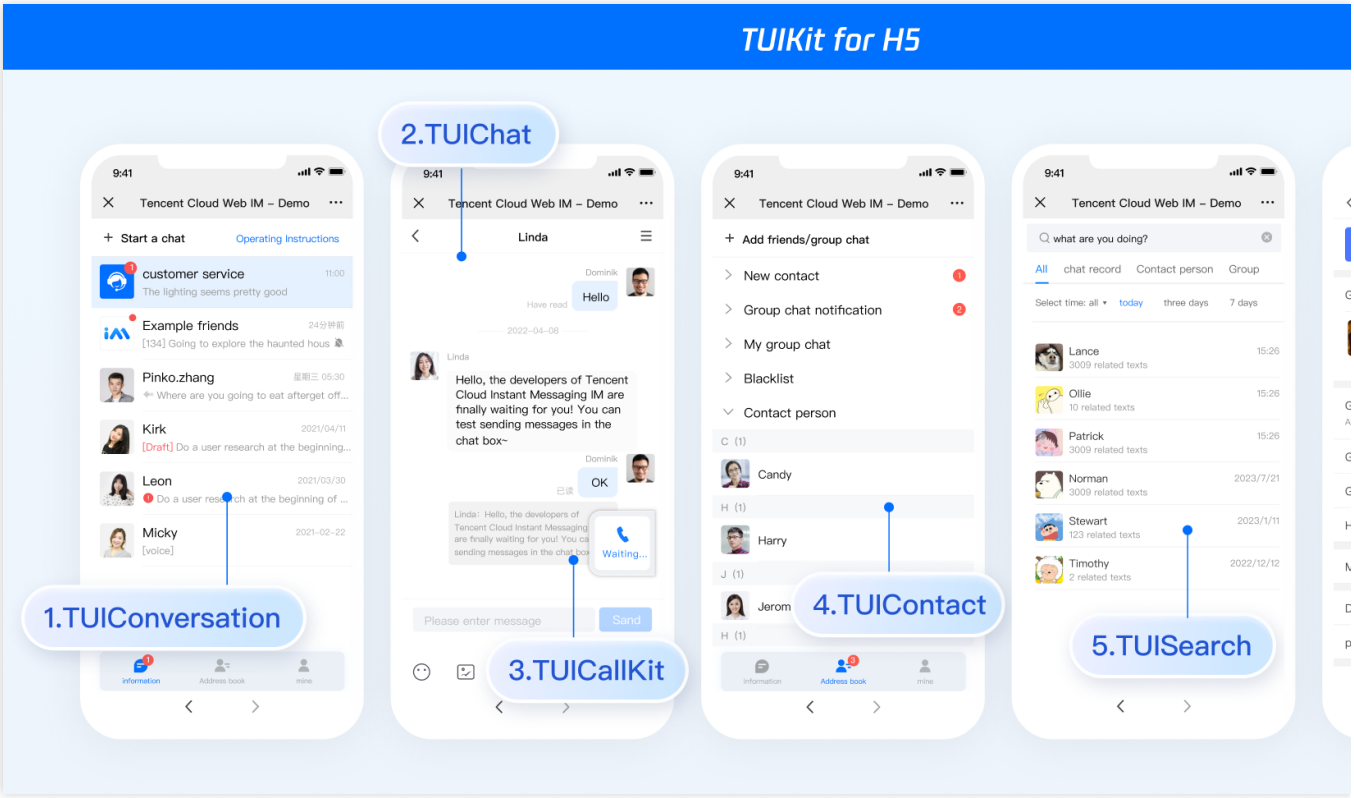
TUIKit Overview

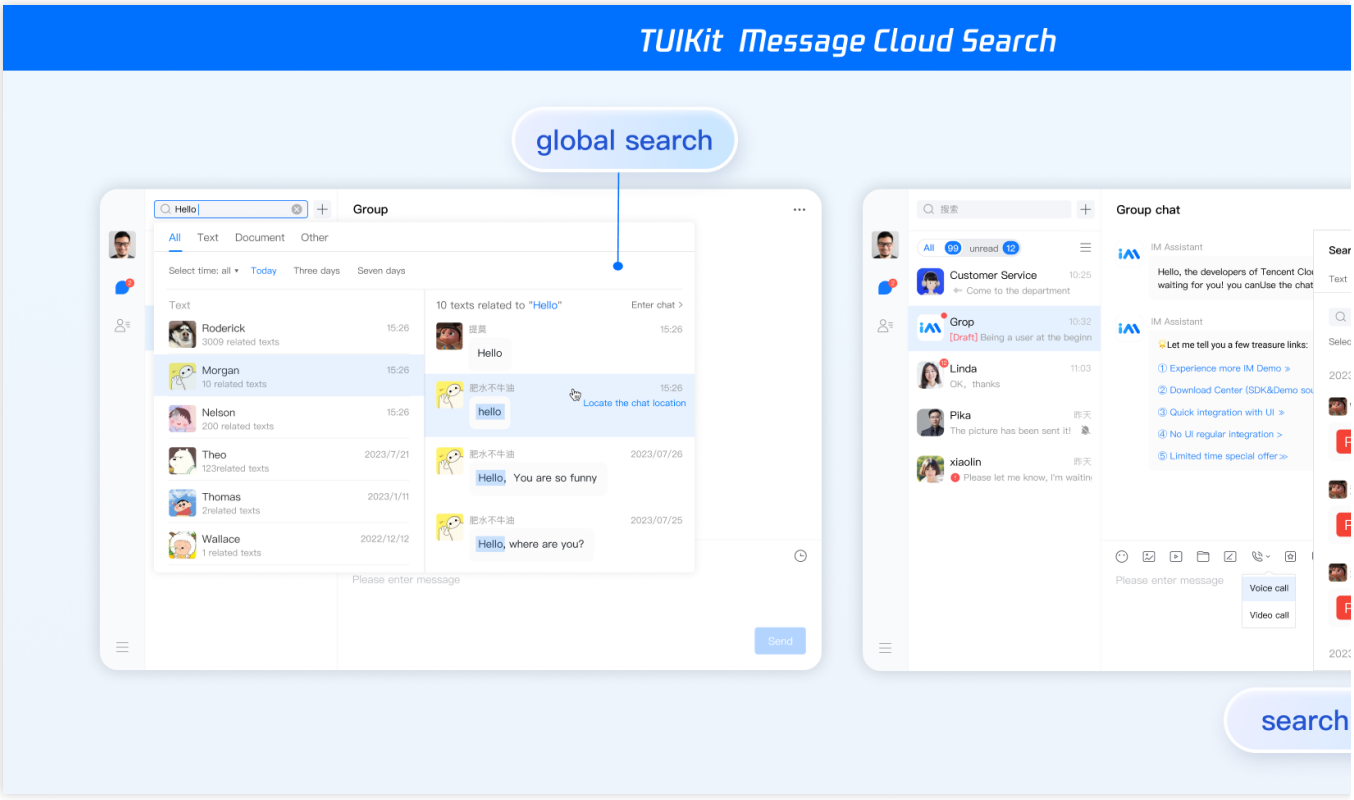
TUIKit is a UI component library based on Tencent Cloud Chat SDK. It provides universal UI components, including conversations, chats, relationship chains, groups, and audio/video calls and other features.
With these UI components, you can quickly build your own in-app chat.
While implementing UI functions, the components in TUIKit will call the corresponding interfaces of the Chat SDK to implement Chat-related logic and data processing. Therefore, developers only need to focus on their own business or personalized expansion when using TUIKit.

TUIKit Components

TUIKit is mainly divided into several UI sub-components: TUIChat, TUIConversation, TUIGroup, TUIContact, and TUISearch.
Each UI component is responsible for displaying different content.







Environment Requirements

Vue (Fully compatible with both Vue2 & Vue3. While incorporating below, please select the Vue version guide that matches your needs)

TypeScript (Should your project be based on JavaScript, please proceed to JS project integrate to set up a progressive support for TypeScript)

Sass (sass-loader ≤ 10.1.1)

node(node.js ≥ 16.0.0)

npm (use a version that matches the Node version in use)

Integration of TUIKit (Web & H5)

Step 1. Create a project

TUIKit supports creating a project structure using webpack or vite, configured with Vue3 / Vue2 + TypeScript + sass. Below are a few examples of how to construct your project:

vue-cli

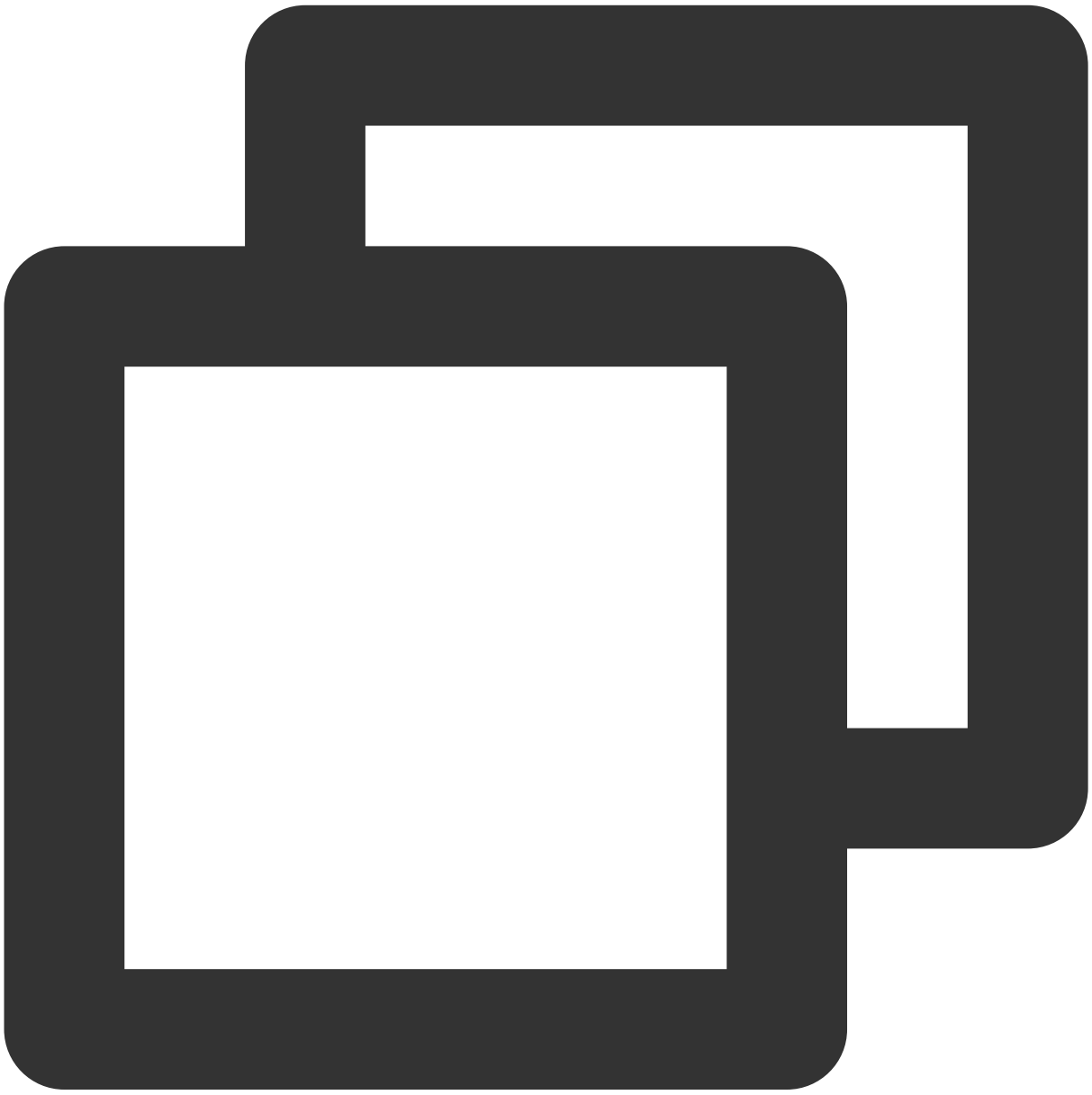
vite

Please Note:

Please make sure you have **@vue/cli version 5.0.0 or above** . The following sample code can be used to upgrade your @vue/cli version to v5.0.8.

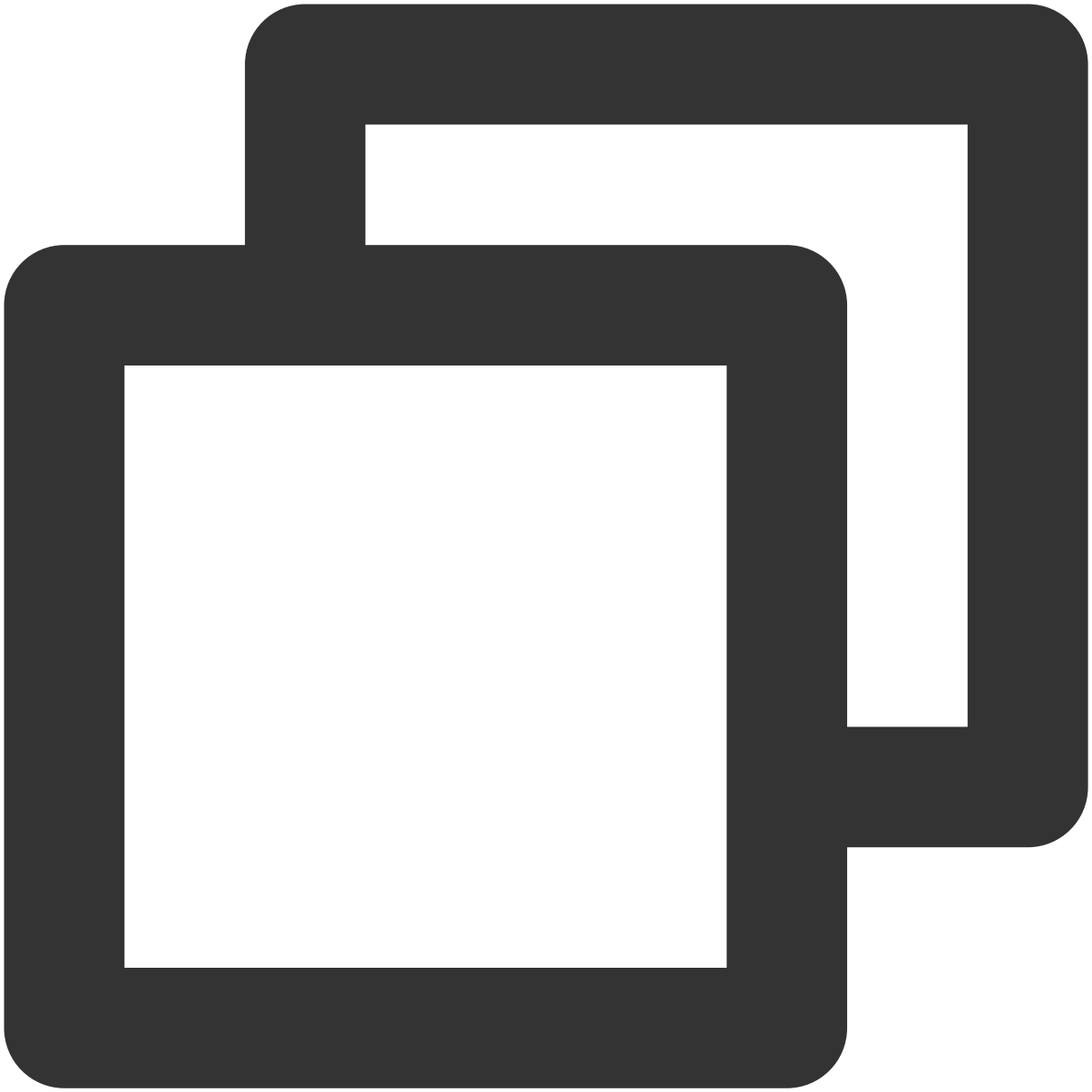
Establish a project using Vue CLI, with configuration set to Vue2/Vue3 + TypeScript + Sass/SCSS.

If Vue CLI is not yet installed, or the version is below 5.0.0, you can use the following method for installation via Terminal or CMD:



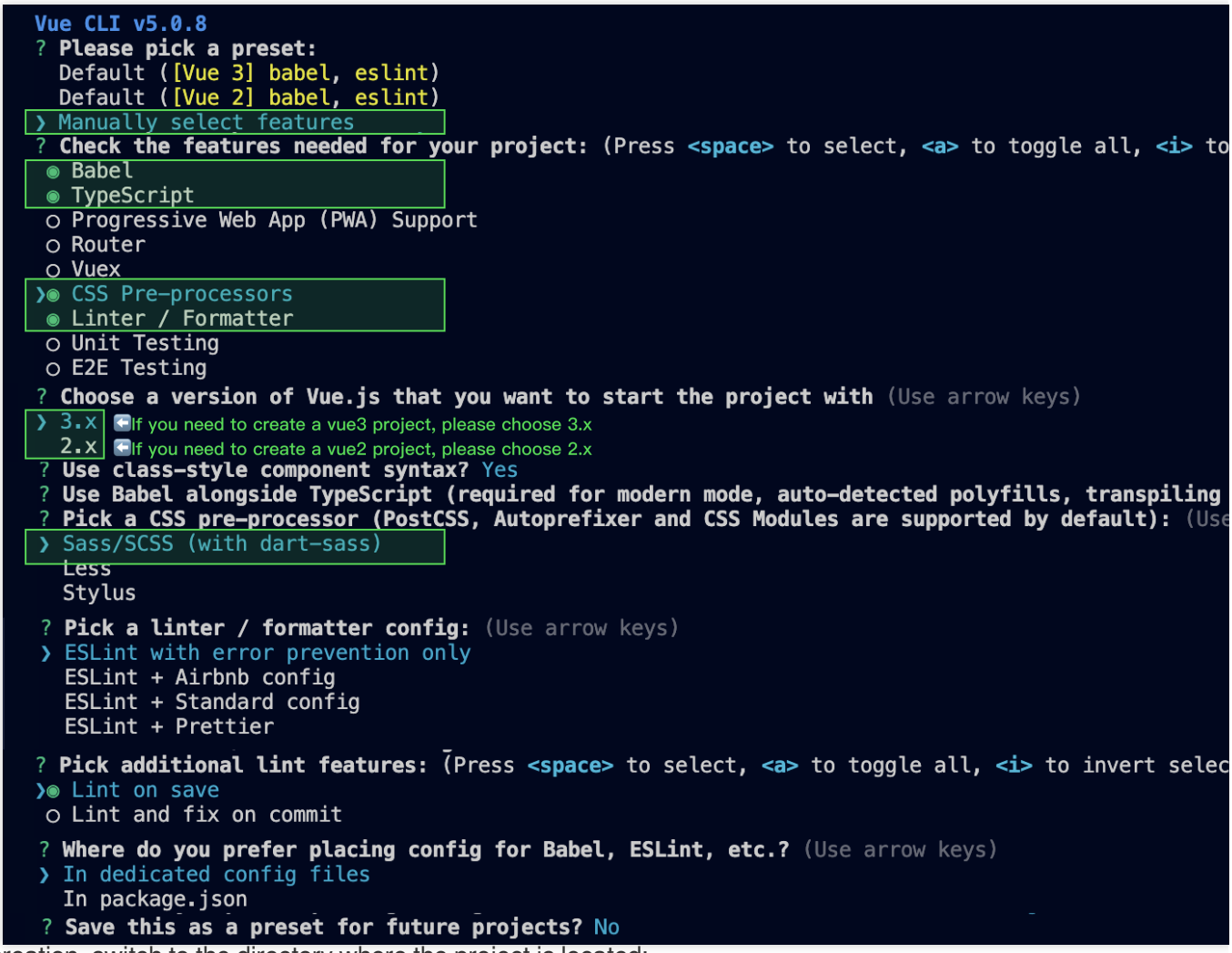
```
npm install -g @vue/cli@5.0.8 sass sass-loader@10.1.1
```

Create a project through Vue CLI and select the configuration items depicted below.

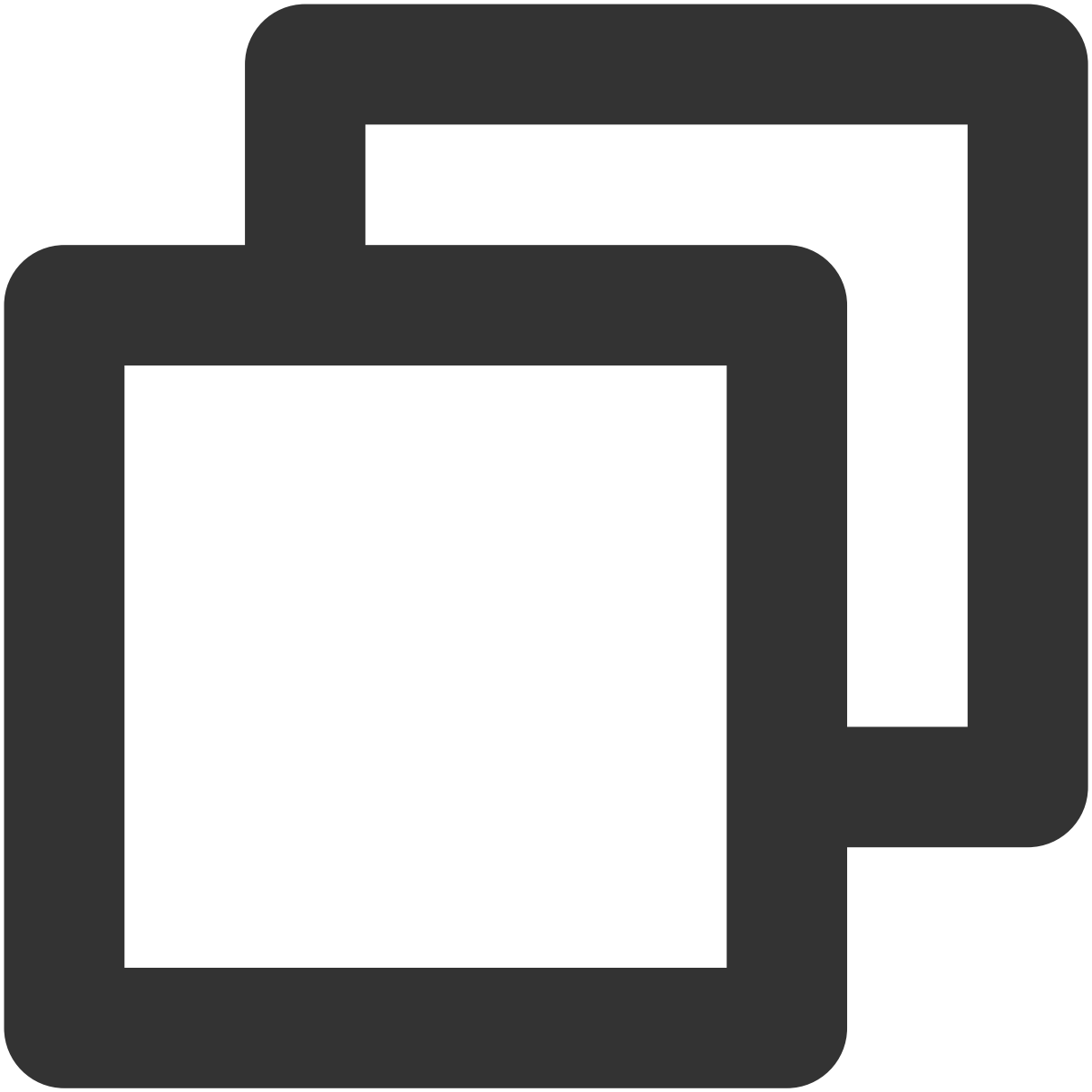


```
vue create chat-example
```

Please make sure to select according to the following configuration:



After creation, switch to the directory where the project is located:



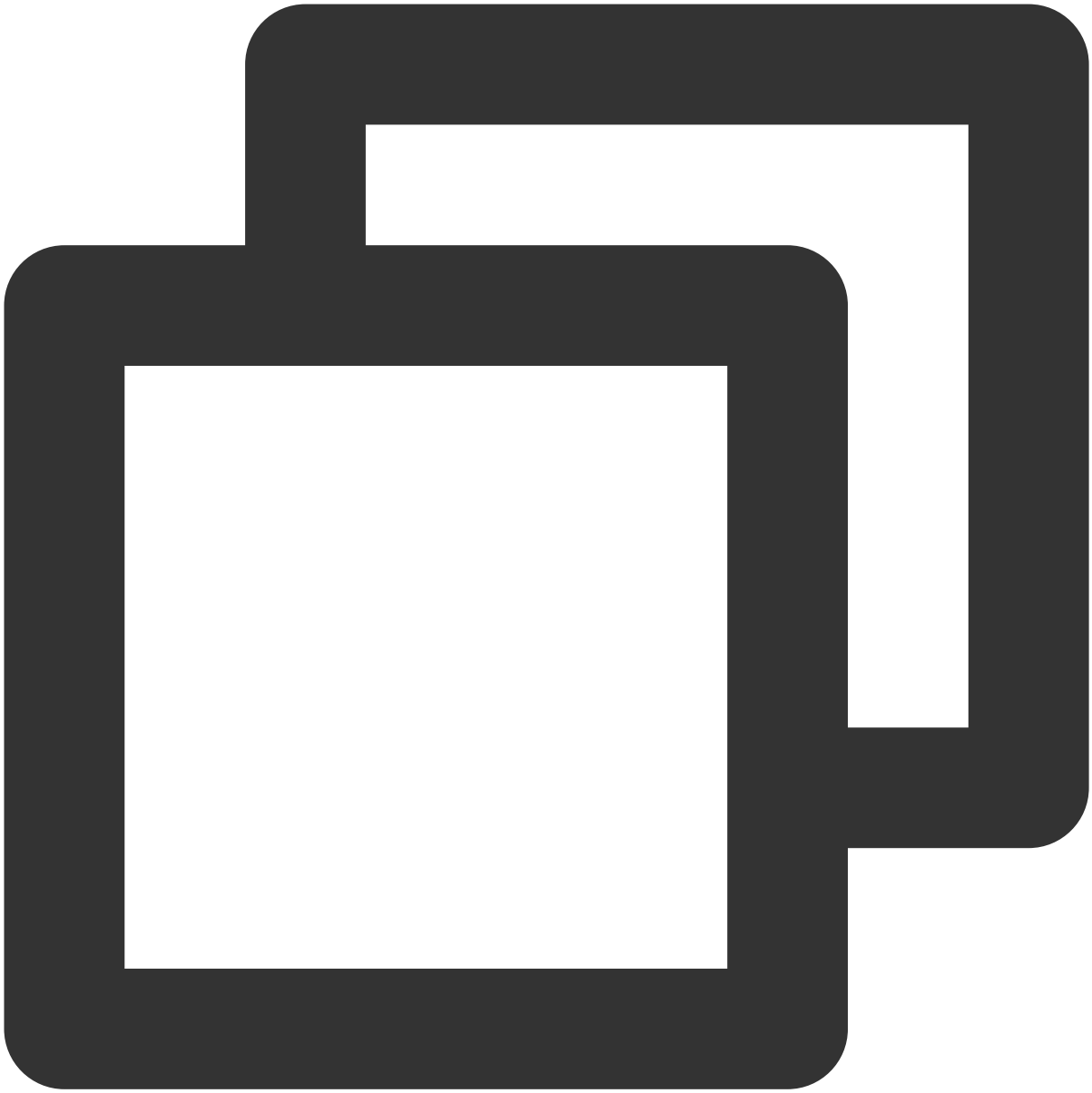
```
cd chat-example
```

If you are a vue2 project, please make the following corresponding environment configurations based on the Vue version you are using.

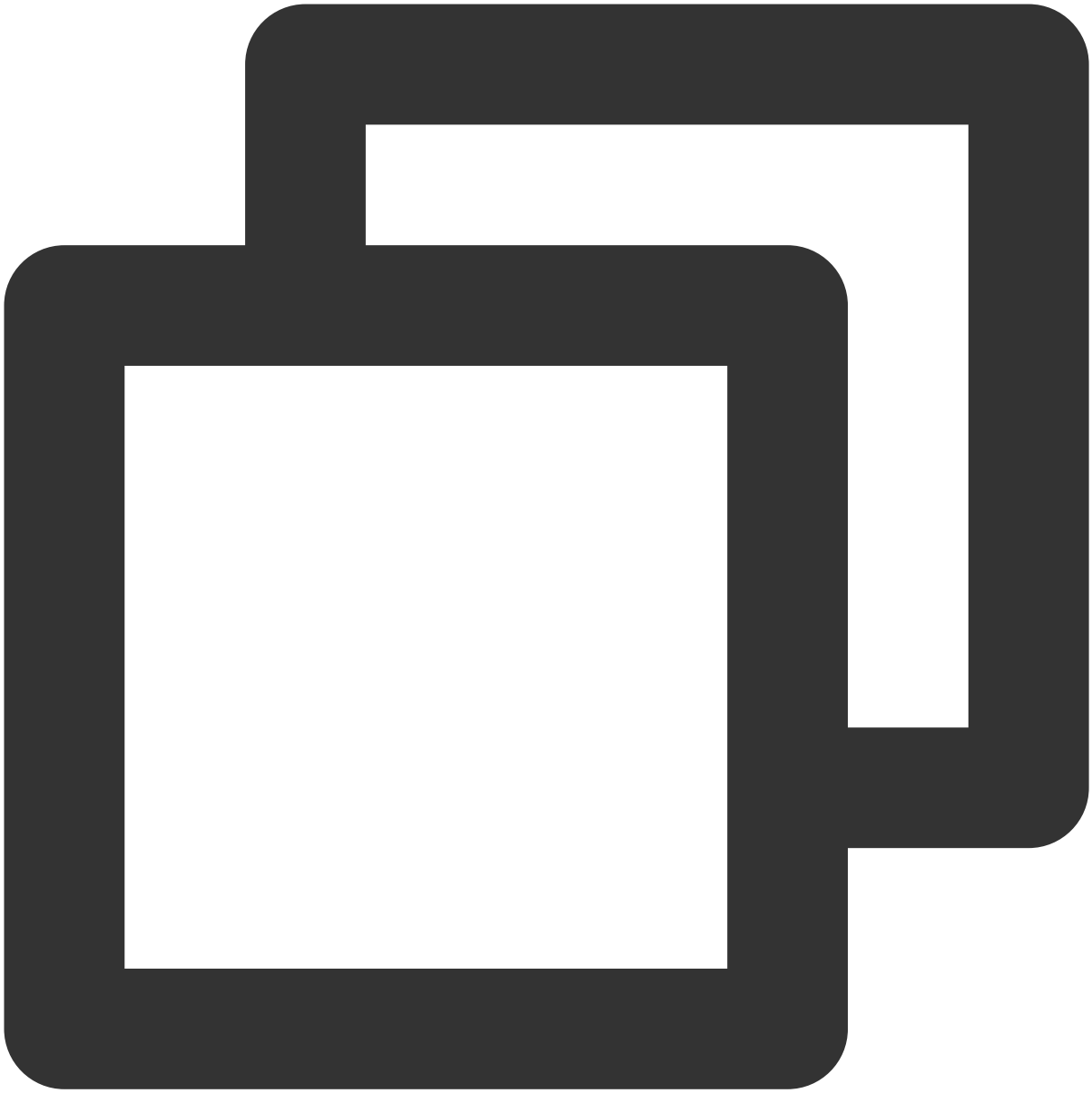
IIf you are a vue2 project, please ignore.

vue2.7

Vue 2.6 and below



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

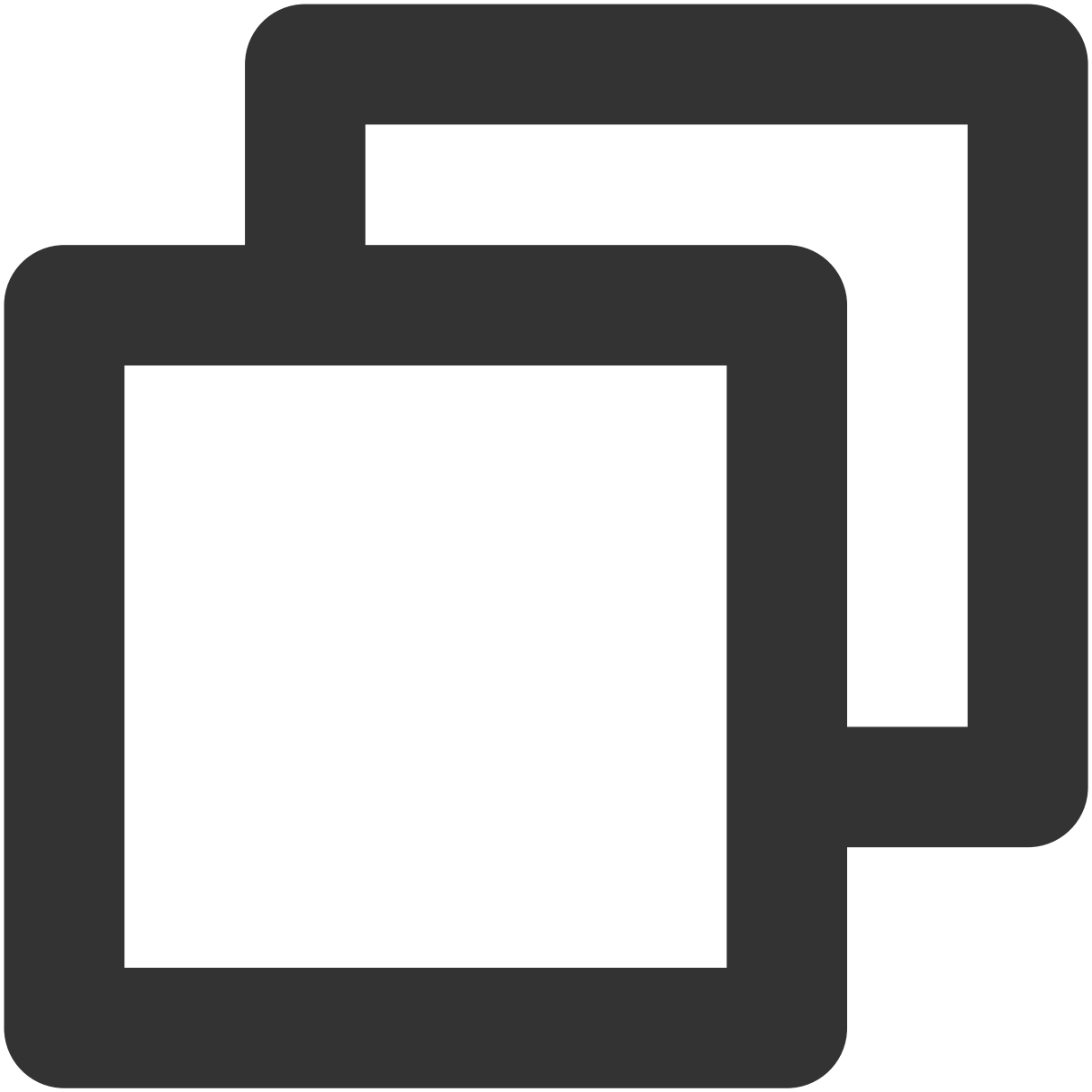


```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

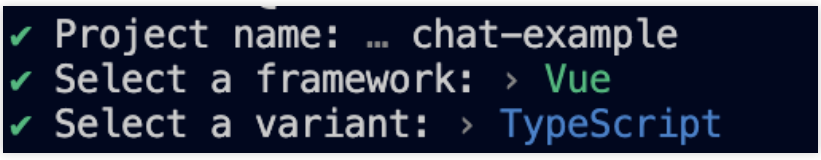
Please Note:

Vite requires **Node.js versions 18+, 20+.** Pay attention to upgrade your Node version when your package manager issues a warning, for more details refer to Vite official website.

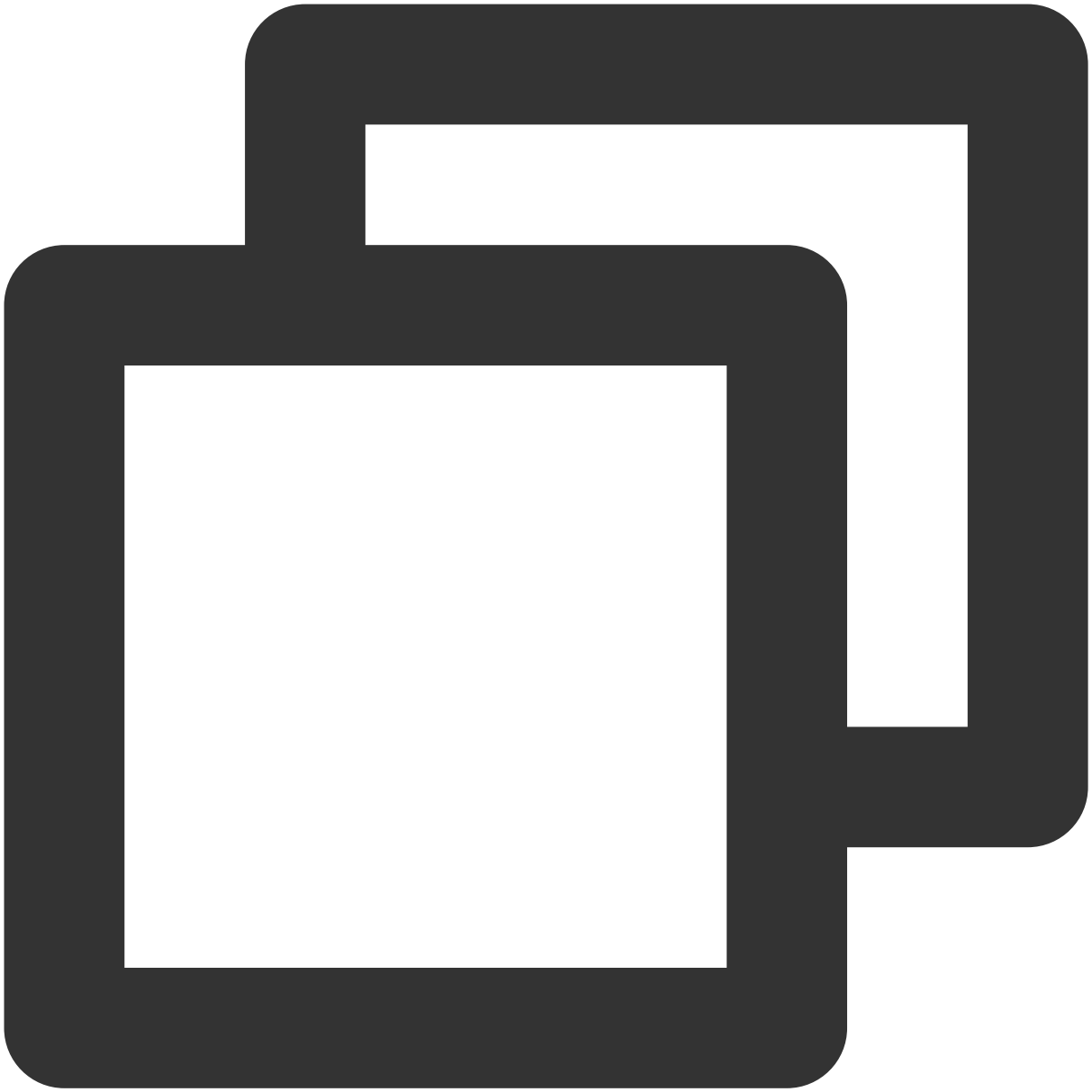
Create a project using Vite, configure Vue + TypeScript according to the options in the picture below.



```
npm create vite@latest
```

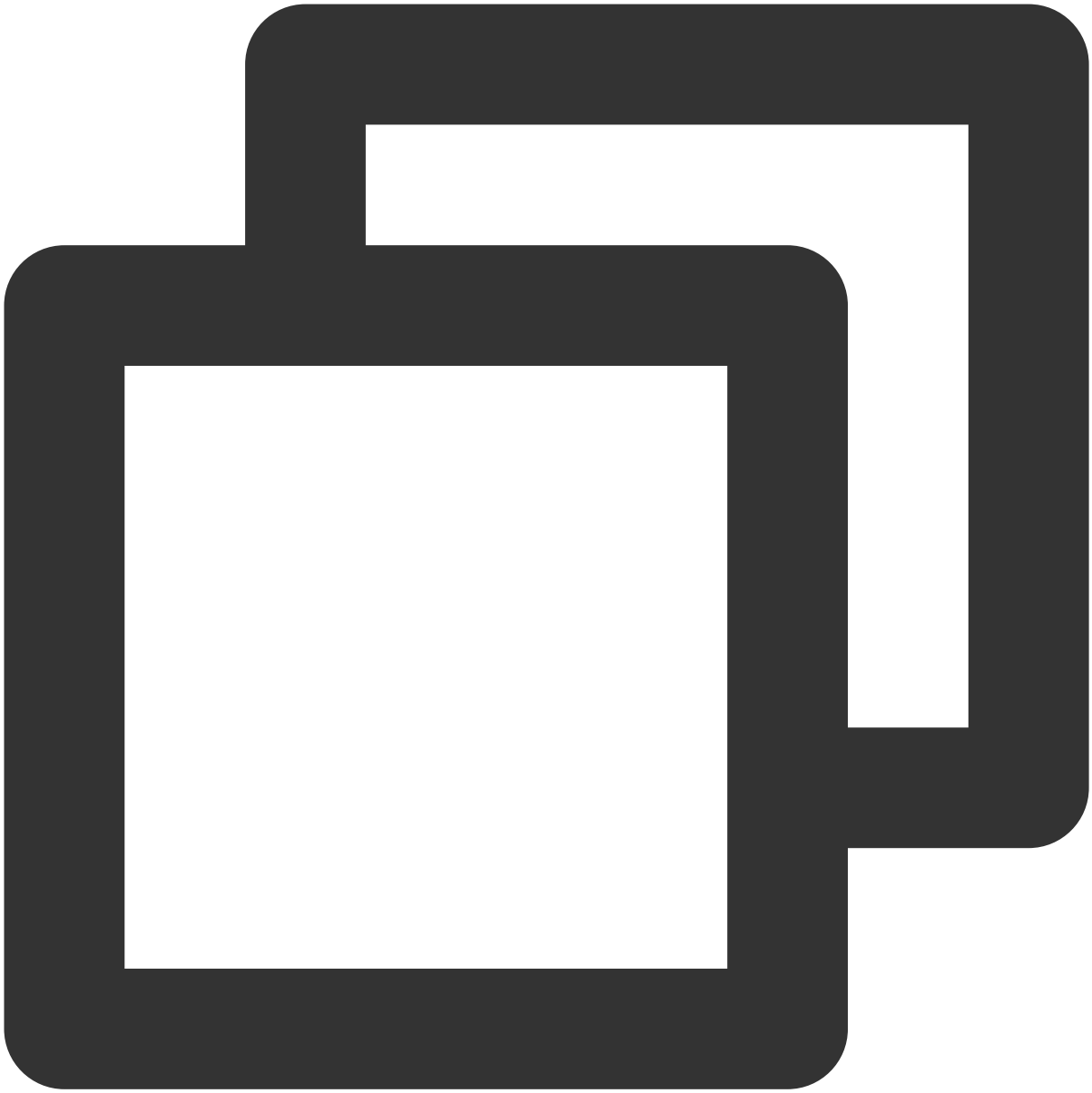


Then, switch to the project directory, and install the project dependencies:



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

Install the sass environment dependency required for TUIKit:



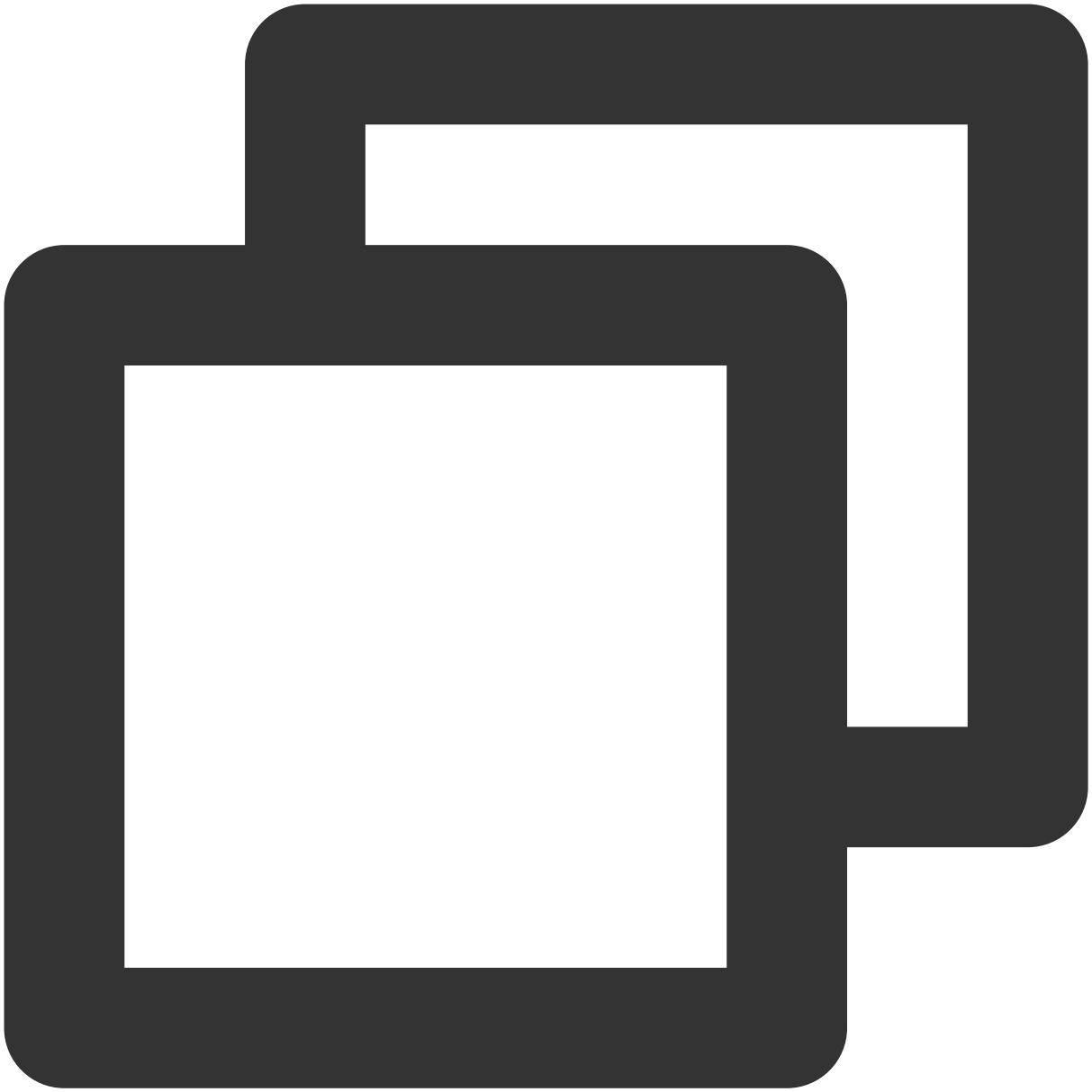
```
npm i -D sass sass-loader
```

Step 2. Download the TUIKit component

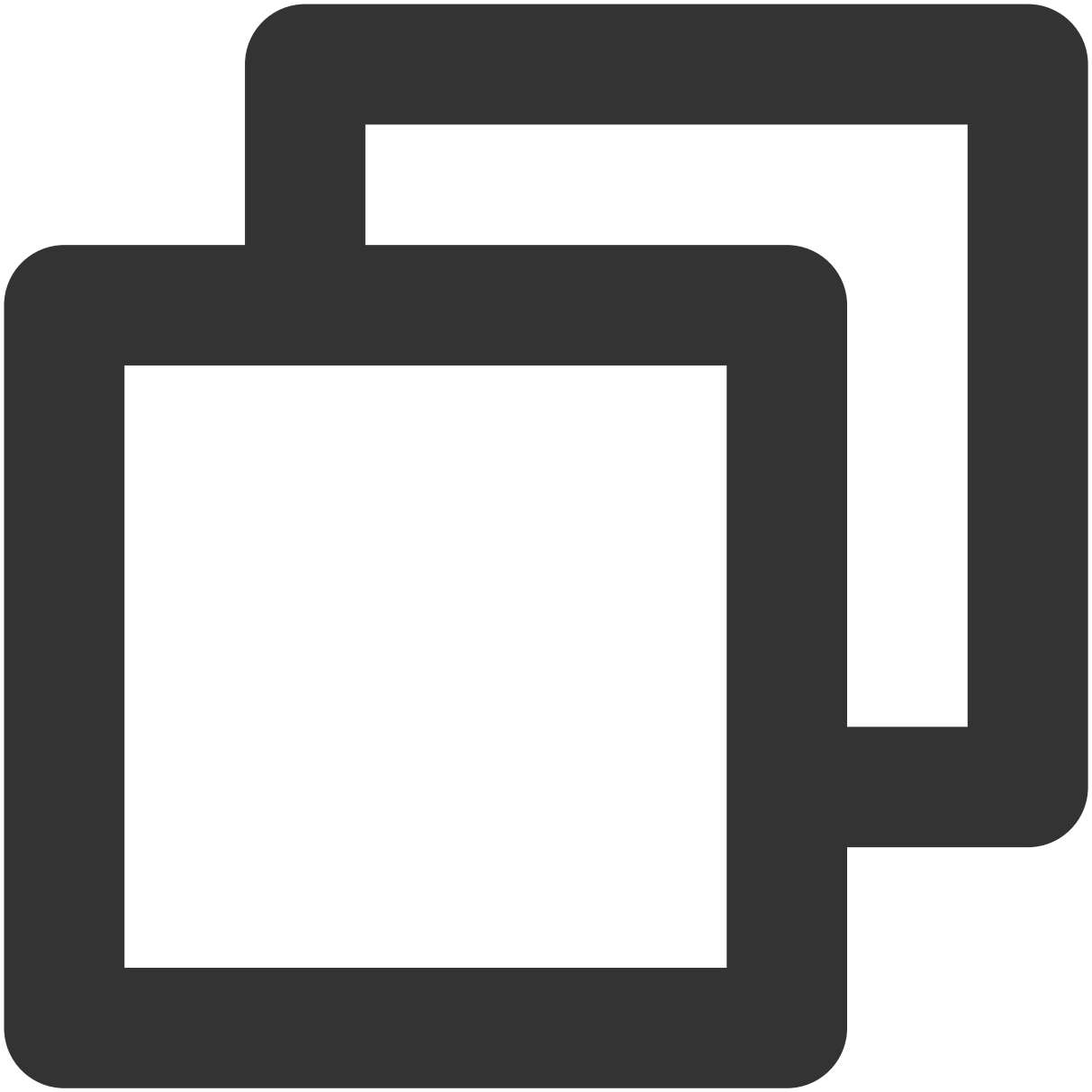
Download the TUIKit component through npm. To facilitate your subsequent expansion, it is recommended that you copy the TUIKit component to the src directory of your project:

macOS

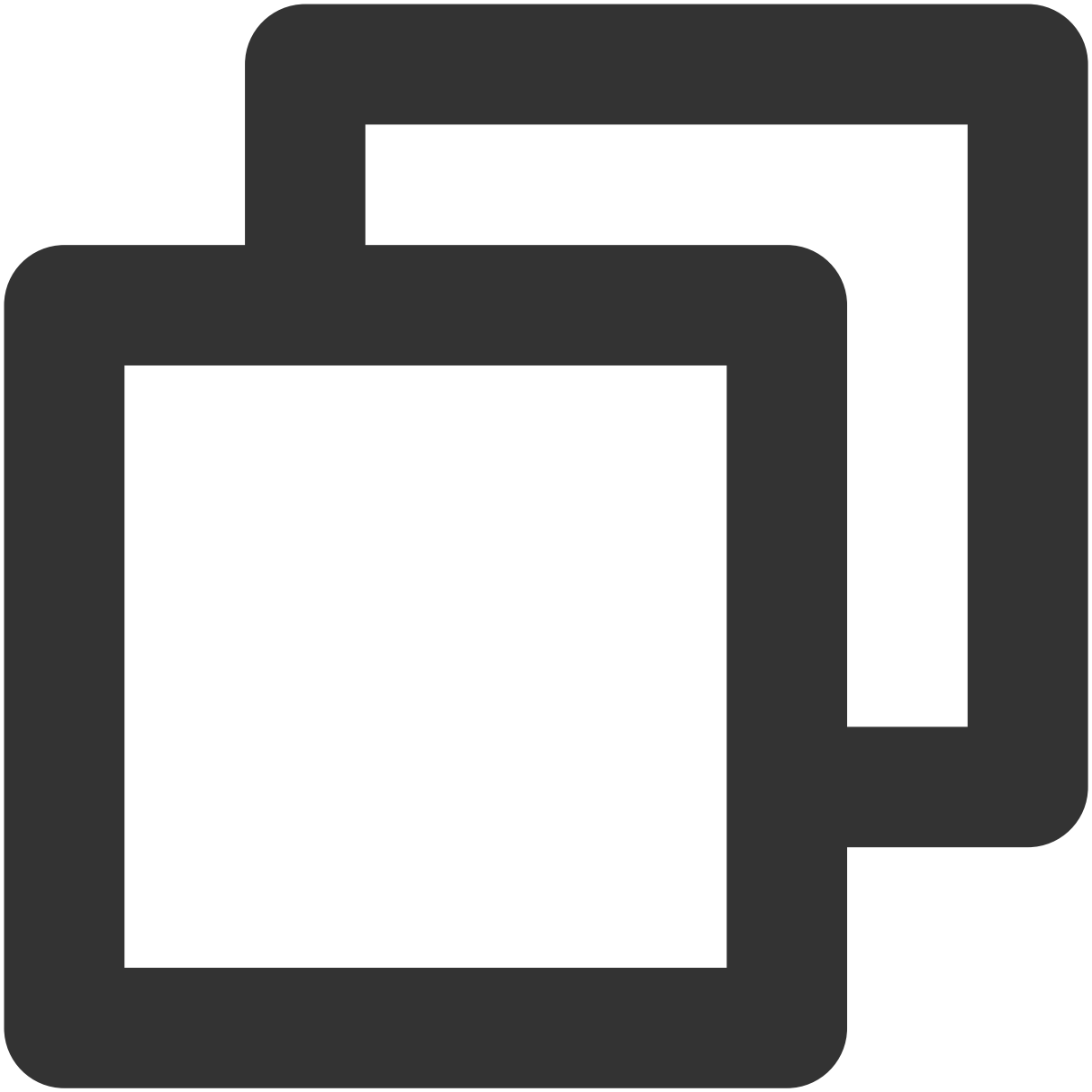
Windows



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

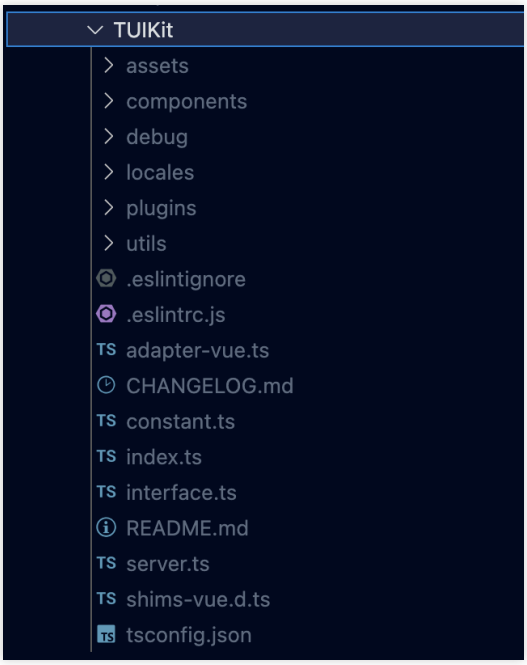


```
npm i @tencentcloud/chat-uikit-vue
```



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

Upon successful completion, the directory structure is depicted as follows:

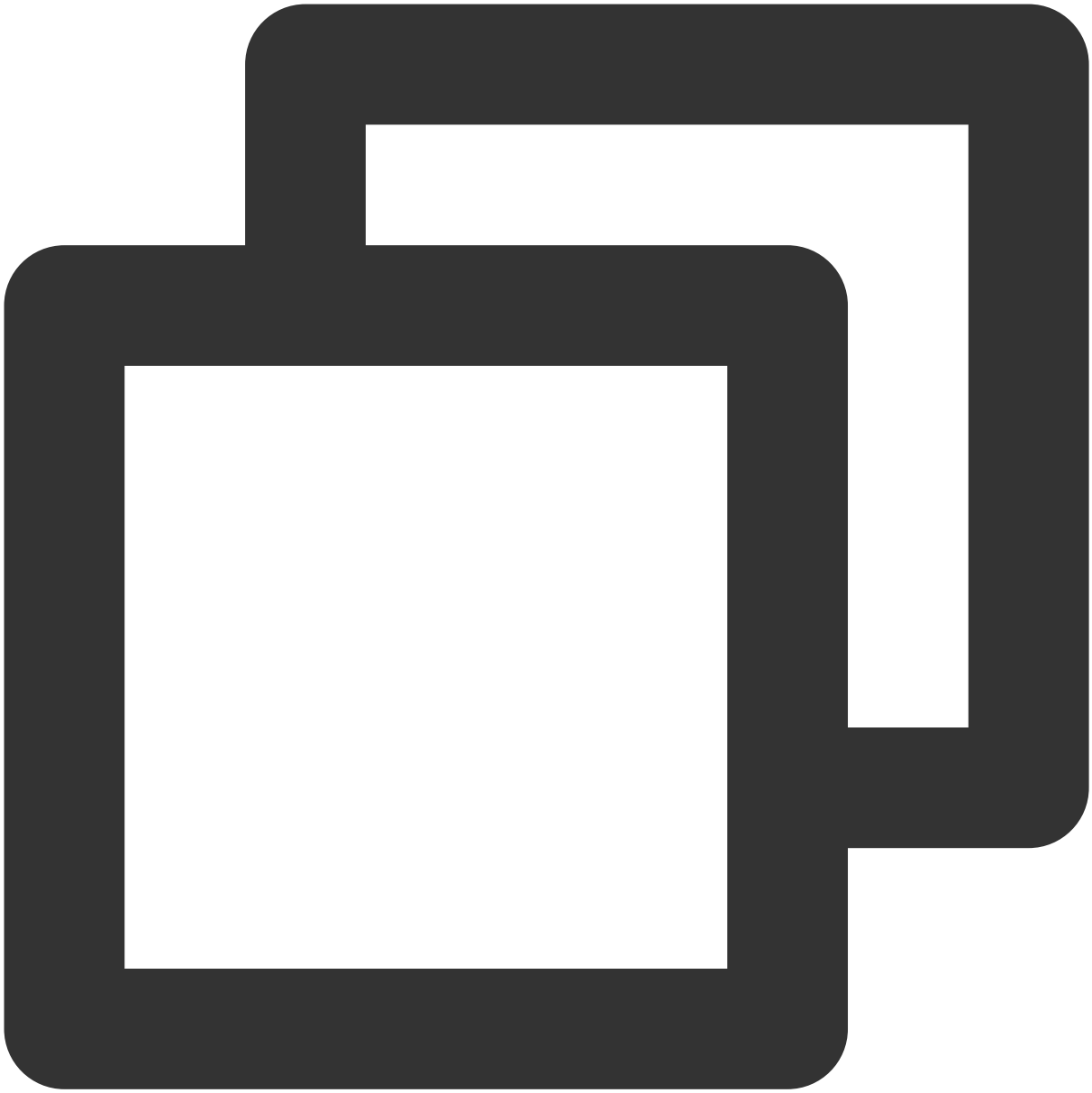


Step 3: Import TUIKit Component

3.1 Import TUIKit in `main.ts / main.js` and register it into the Vue project instance

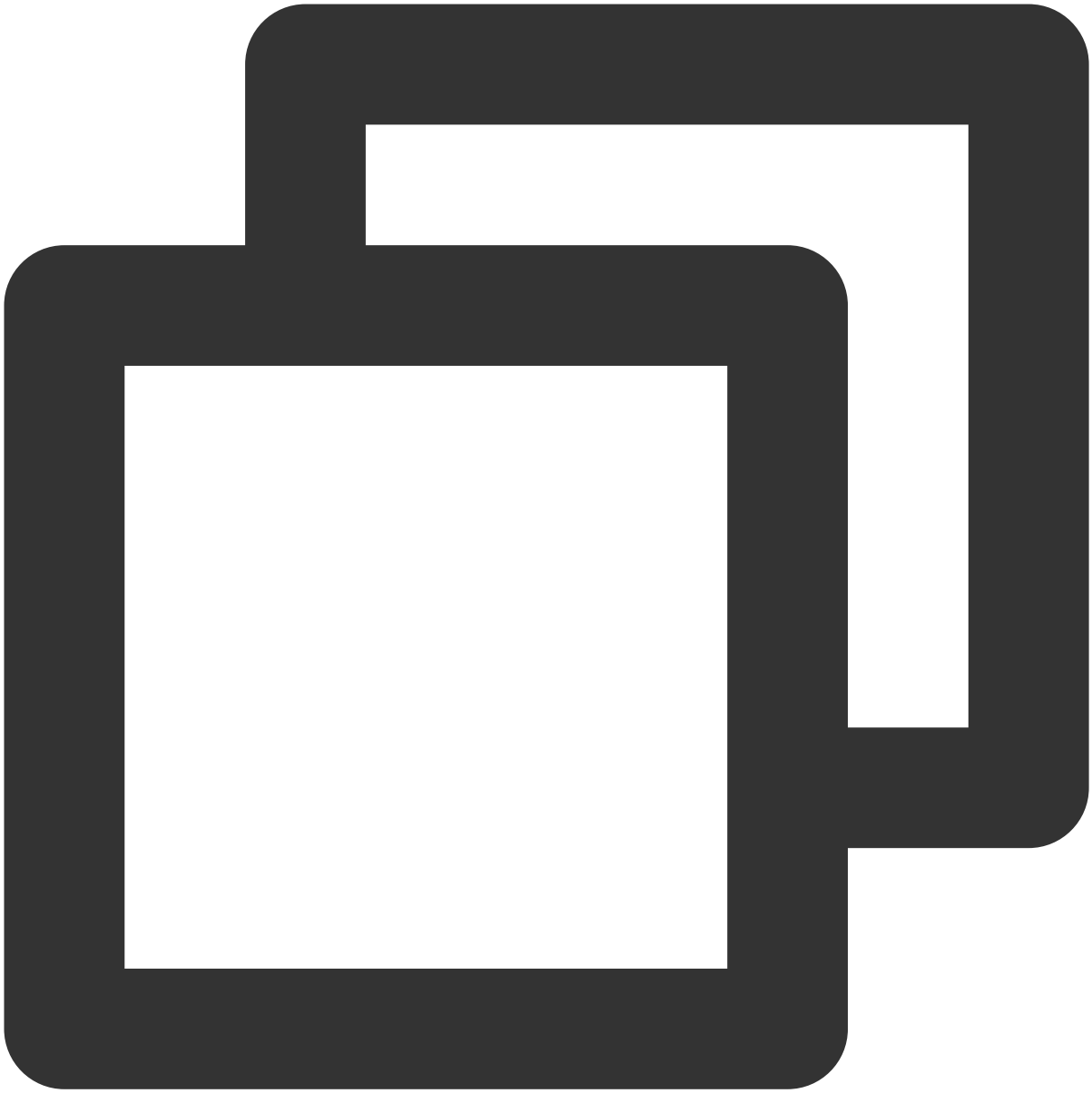
Vue3

Vue2



```
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,
  // Local review can identify and handle unsafe and inappropriate content, safegua
  // This function is a value-added service, for reference please visit: https://cl
  // If you have purchased the content review service, please set this feature to `
  useProfanityFilterPlugin: false,
  framework: "vue2",
});
Vue.config.productionTip = false;
  new Vue({
    render: (h) => h(App),
  }).$mount("#app");
export { SDKAppID, secretKey };
```

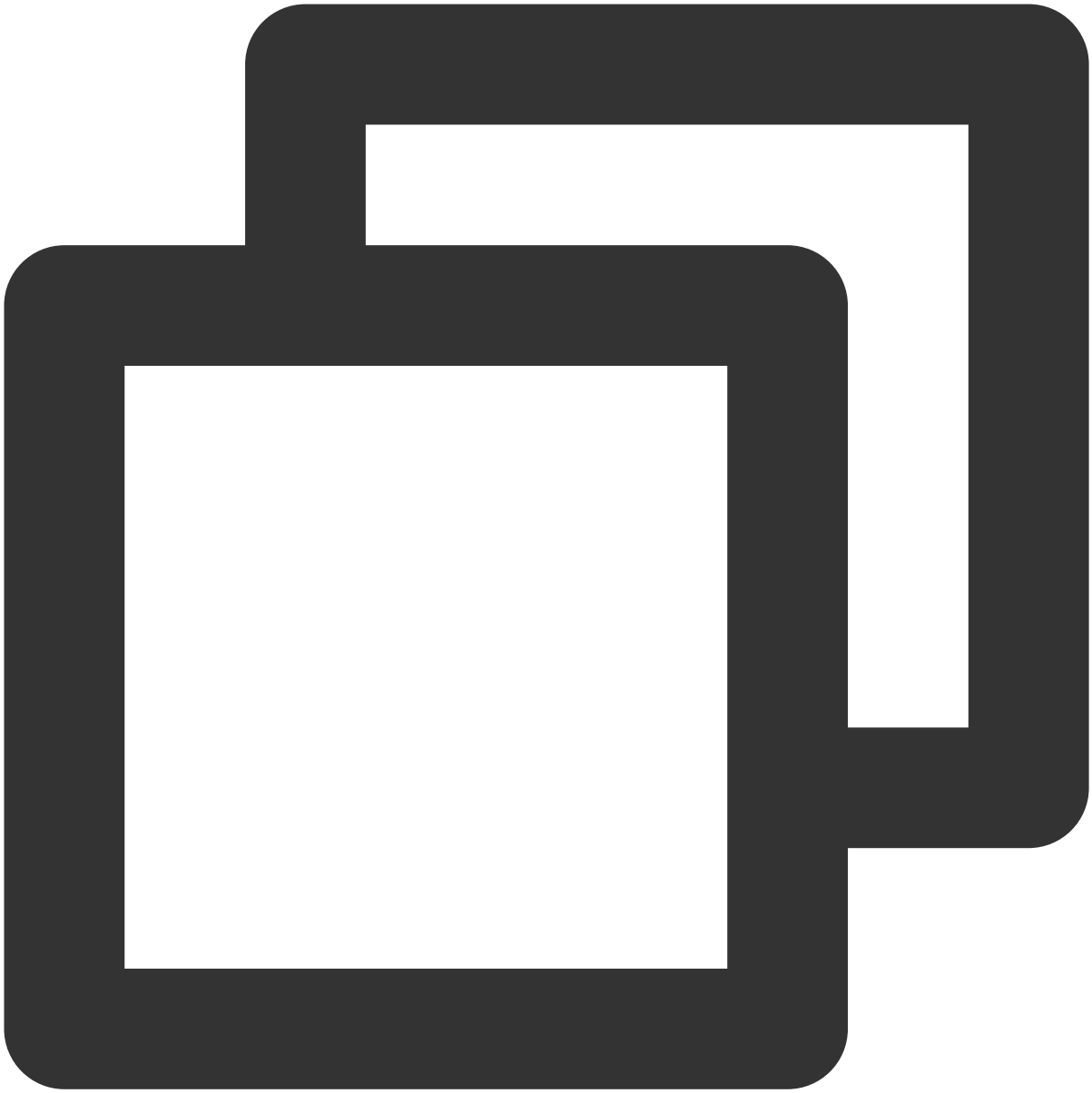
3.2 Implement UIKit to your web app

Vue3

Vue 2.7

Vue2.6 or lower versions

For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).

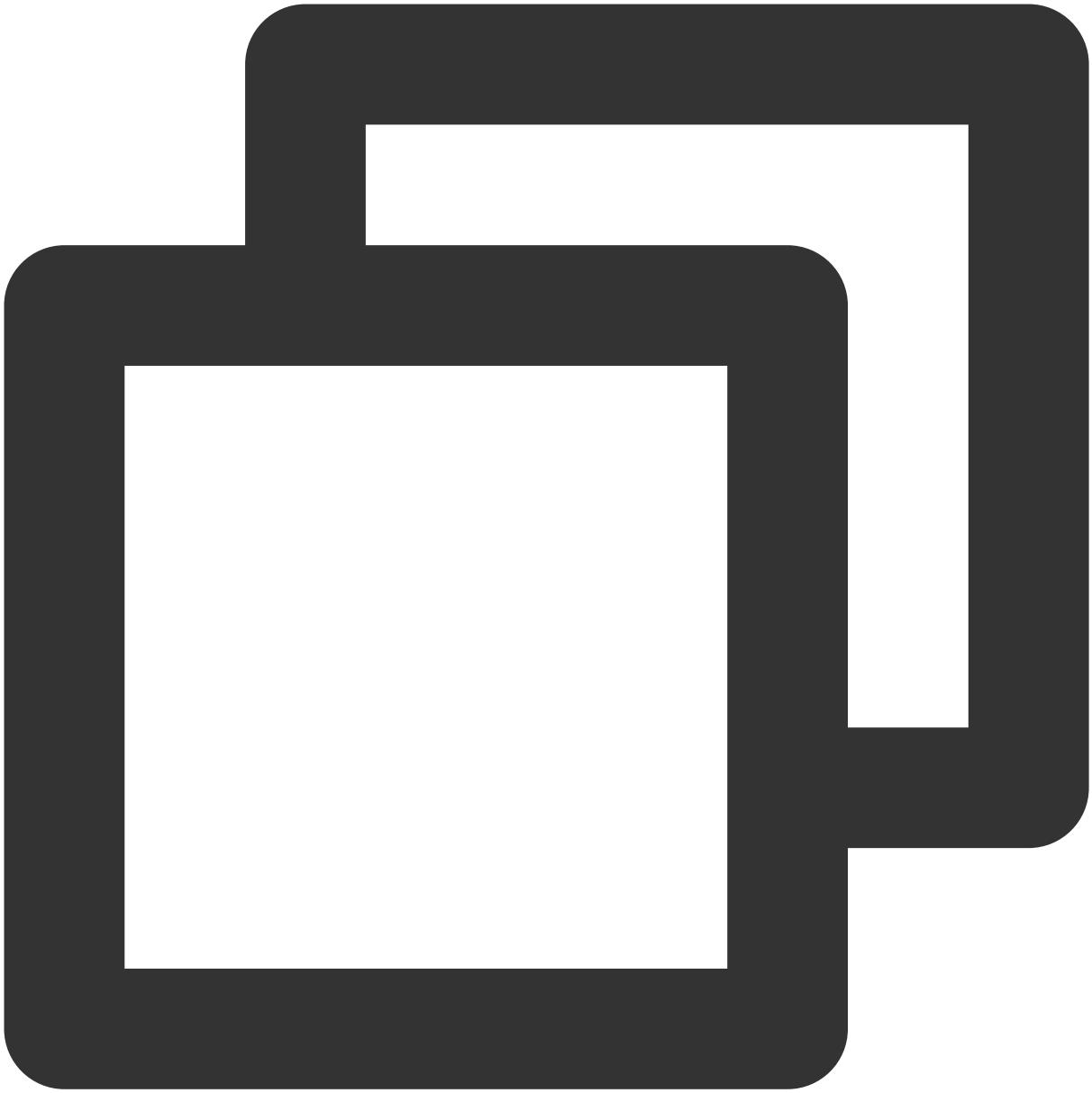


```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar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

For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).

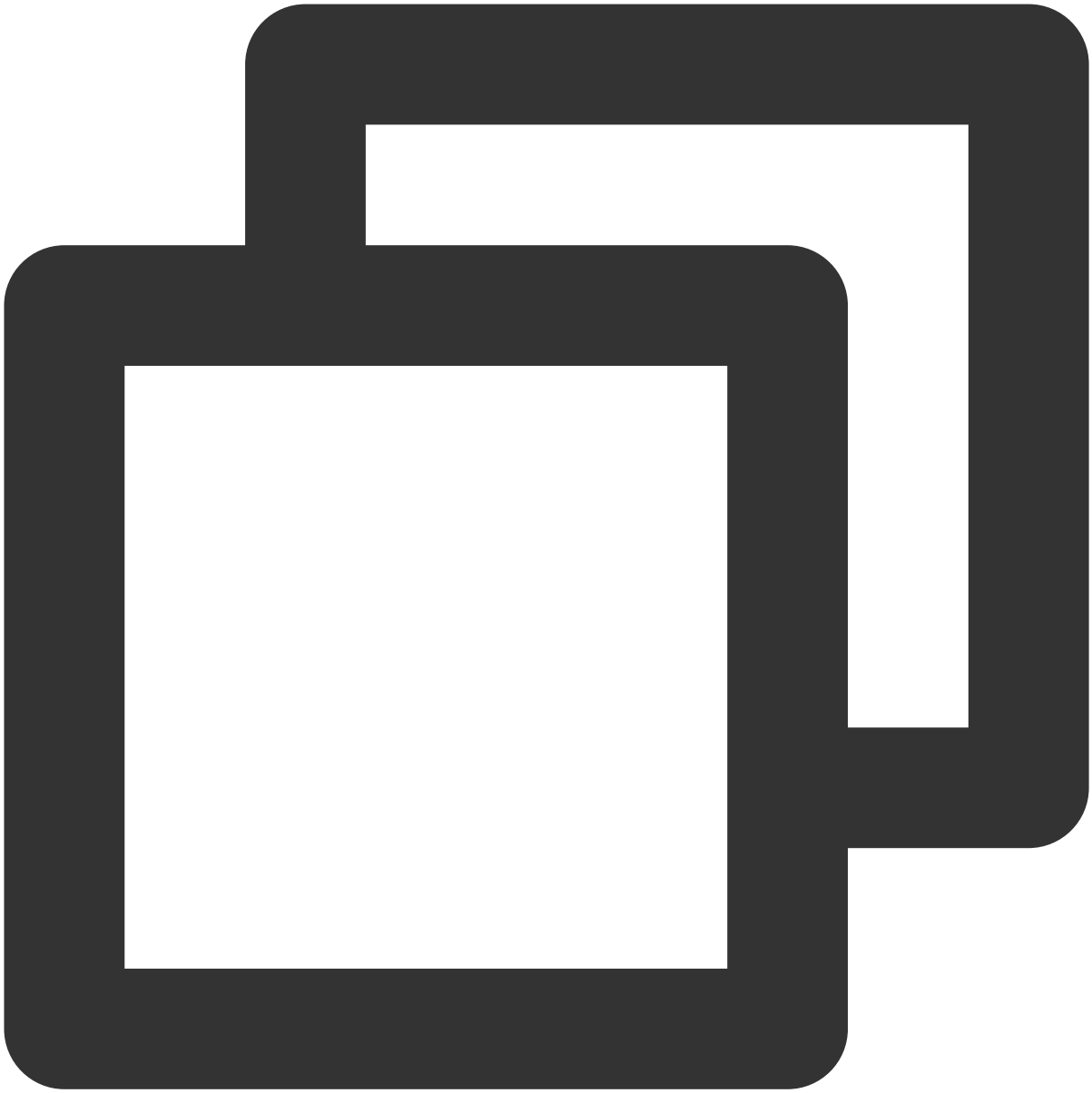


```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar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

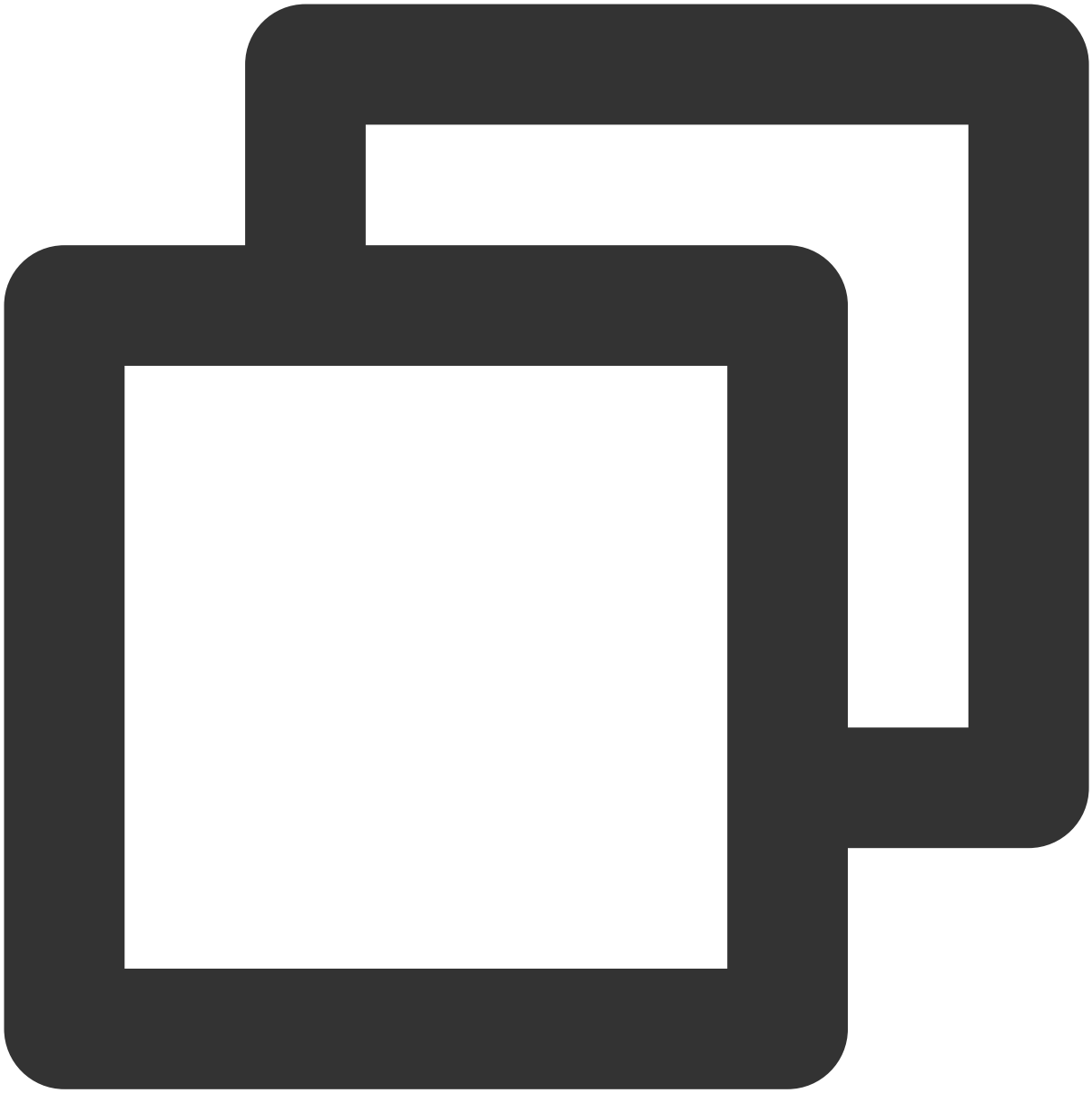
```
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: ",
        },
      ],
    };
  },
  mounted: function () {
    TUIStore.watch(StoreName.CONV, {
      currentConversationID: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

1. Install the relevant dependencies supporting `composition-api` , `script setup` , and `vue2.6`



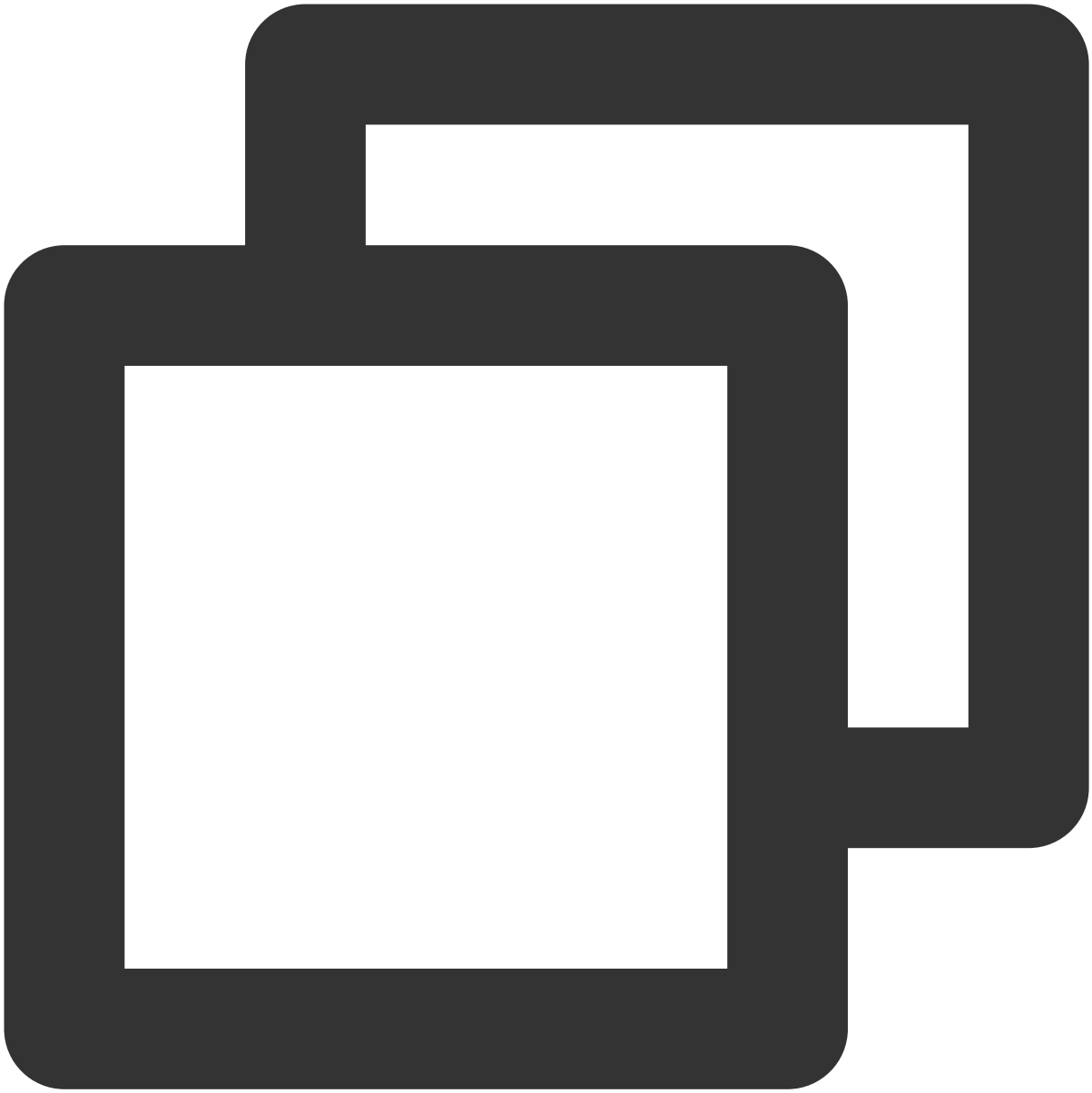
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

2. Import `VueCompositionAPI` in `main.ts`



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

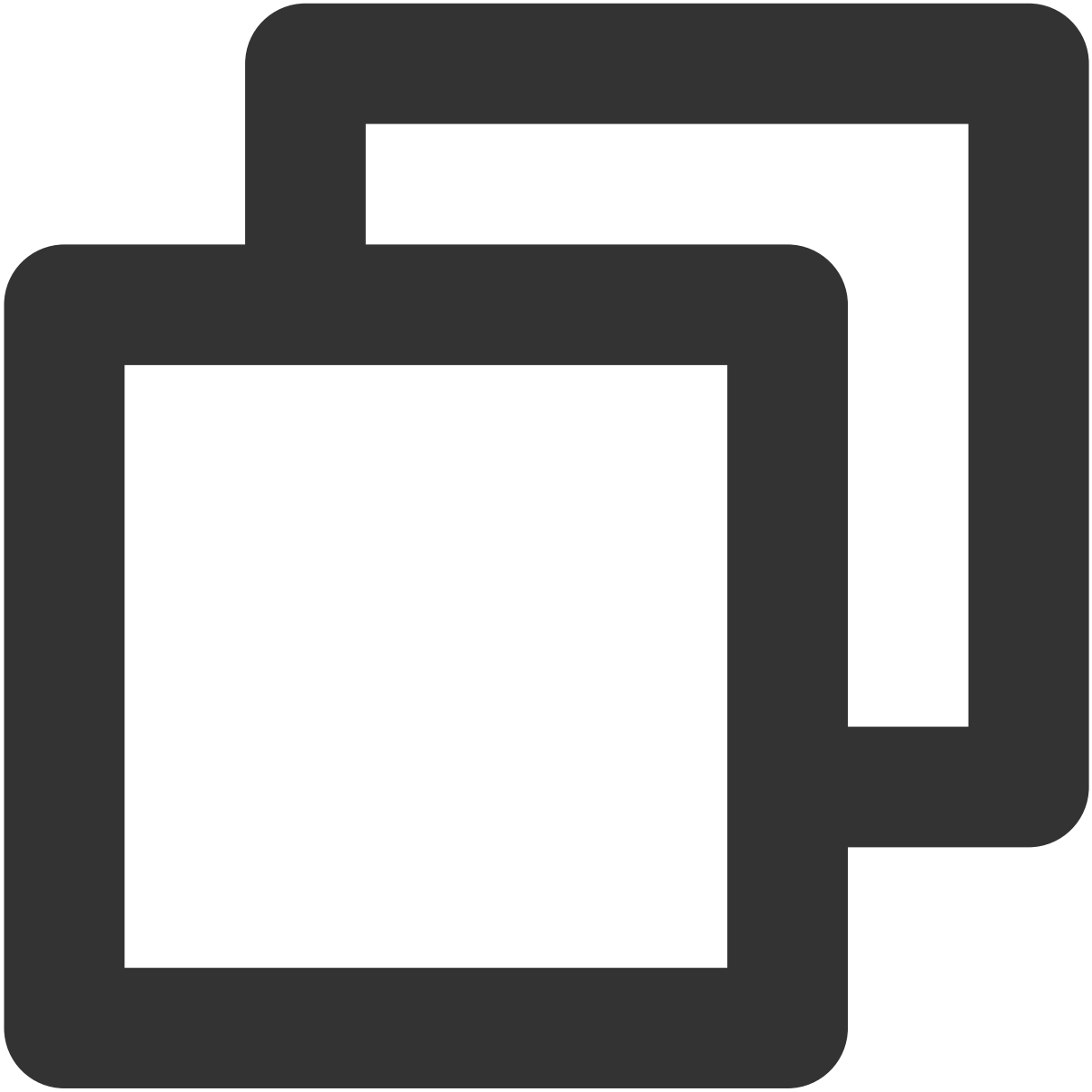
3. Add the following pattern to `vue.config.js` , if the file doesn't exist, please create one:



```
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) {
```

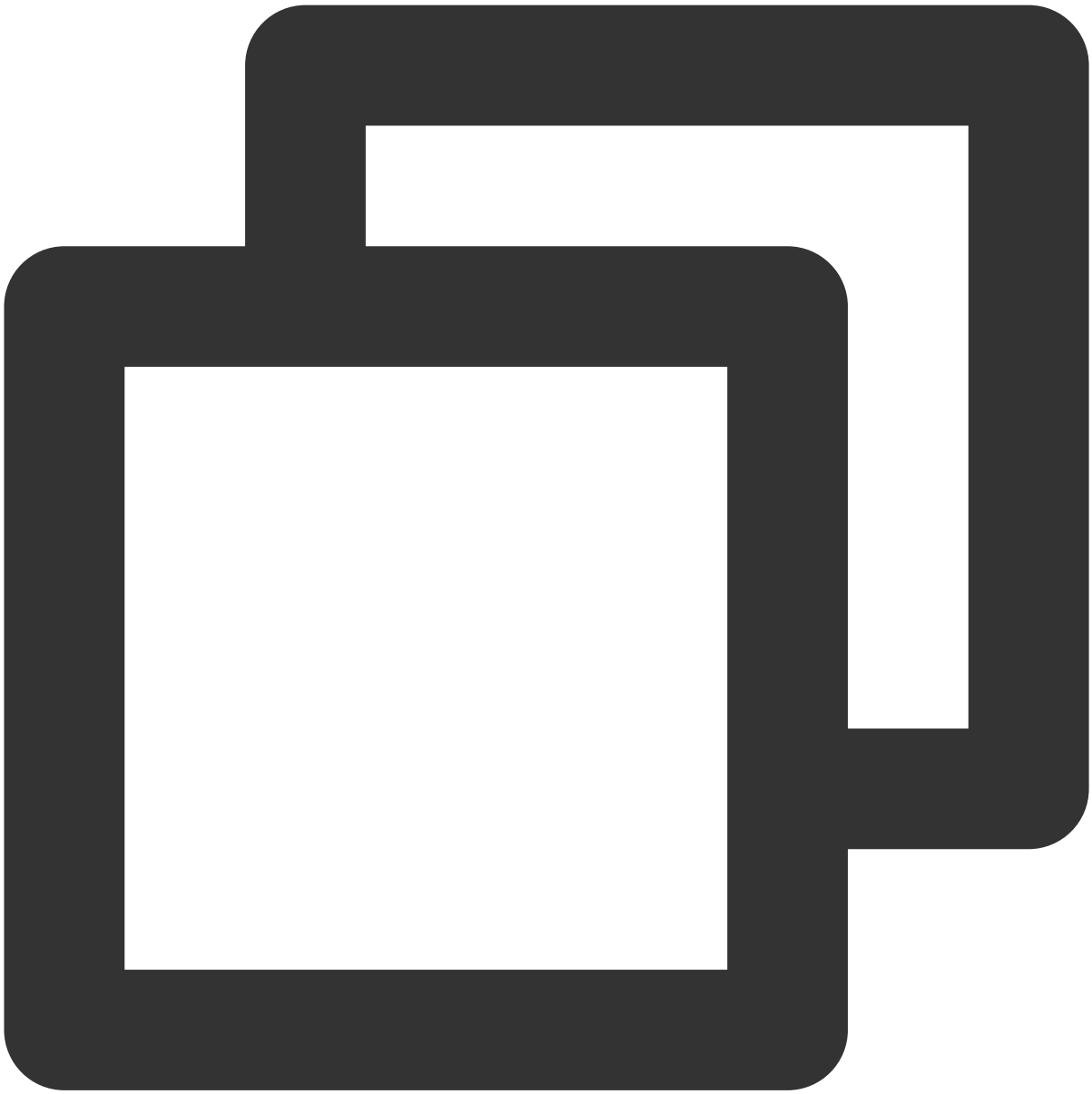
```
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete("fork-ts-checker");
  },
};
```

4. Replace the export source at the end of the `src/TUIKit/adapter-vue.ts` file:



```
// Initial script
export * from "vue";
// Replace with
export * from "@vue/composition-api";
```

5. For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).



```
<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"
```

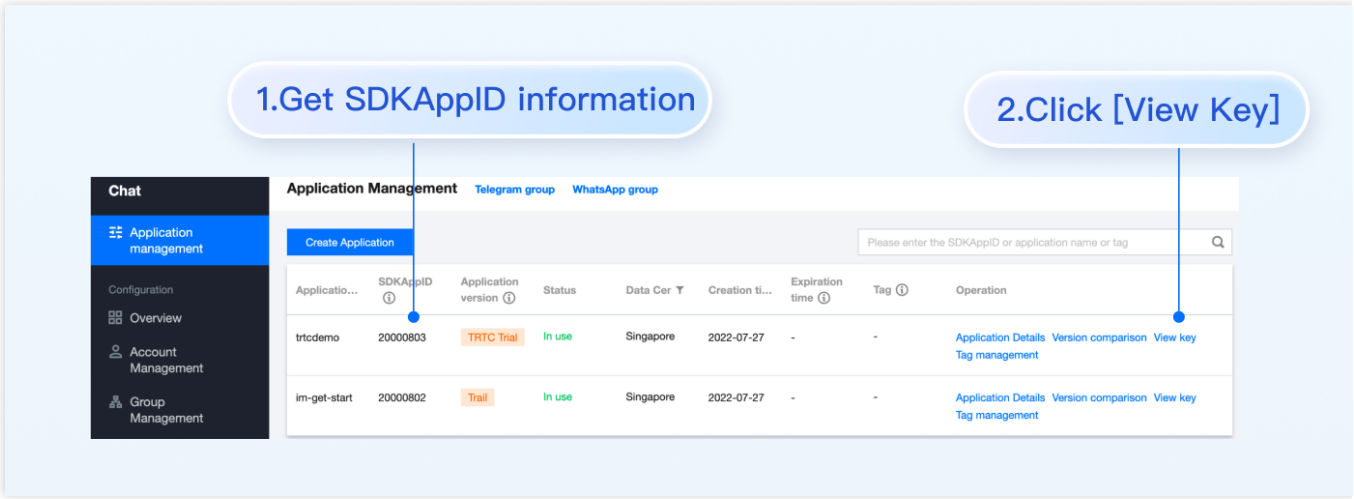
```
      >
        {{ item.label }}
      </div>
    </div>
    <div class="TUIKit-main-container">
      <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
        <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
          <TUISearch searchType="global"></TUISearch>
          <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() {
```

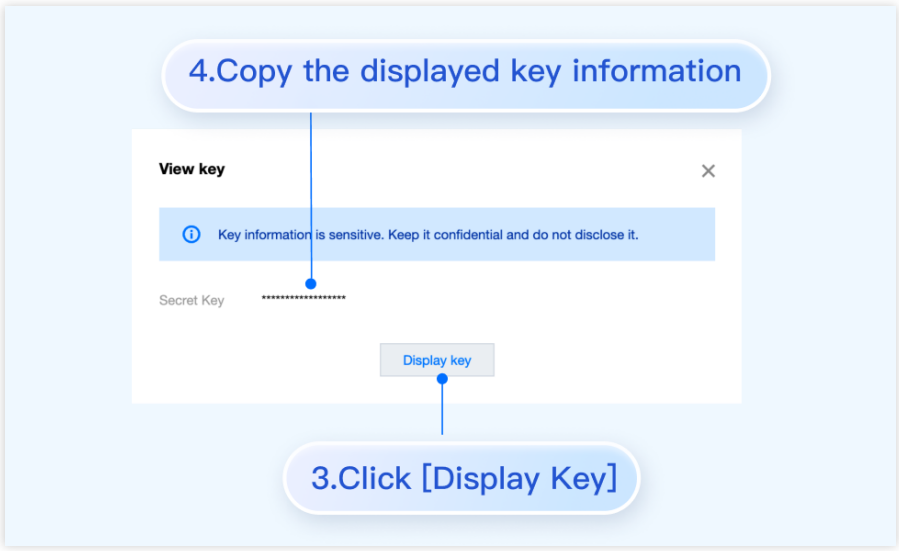
```
    return {
      isH5: isH5,
      currentConversationID: "",
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: "Contact",
        },
      ],
    };
  },
  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>
```

Step 4: Secure SDKAppID, secretKey, and userID

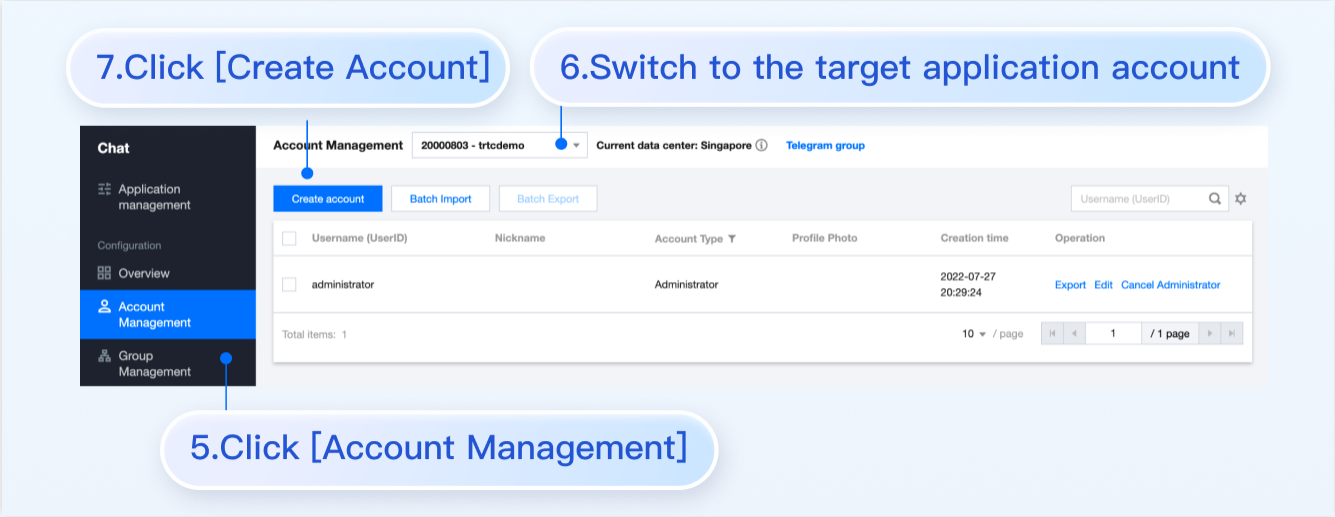
Set the relevant parameters `SDKAppID`, `secretKey`, and `userID` in the example code of the `main.ts / main.js` file:

`SDKAppID` and `SecretKey` can be accessed by the Instant Messaging console:





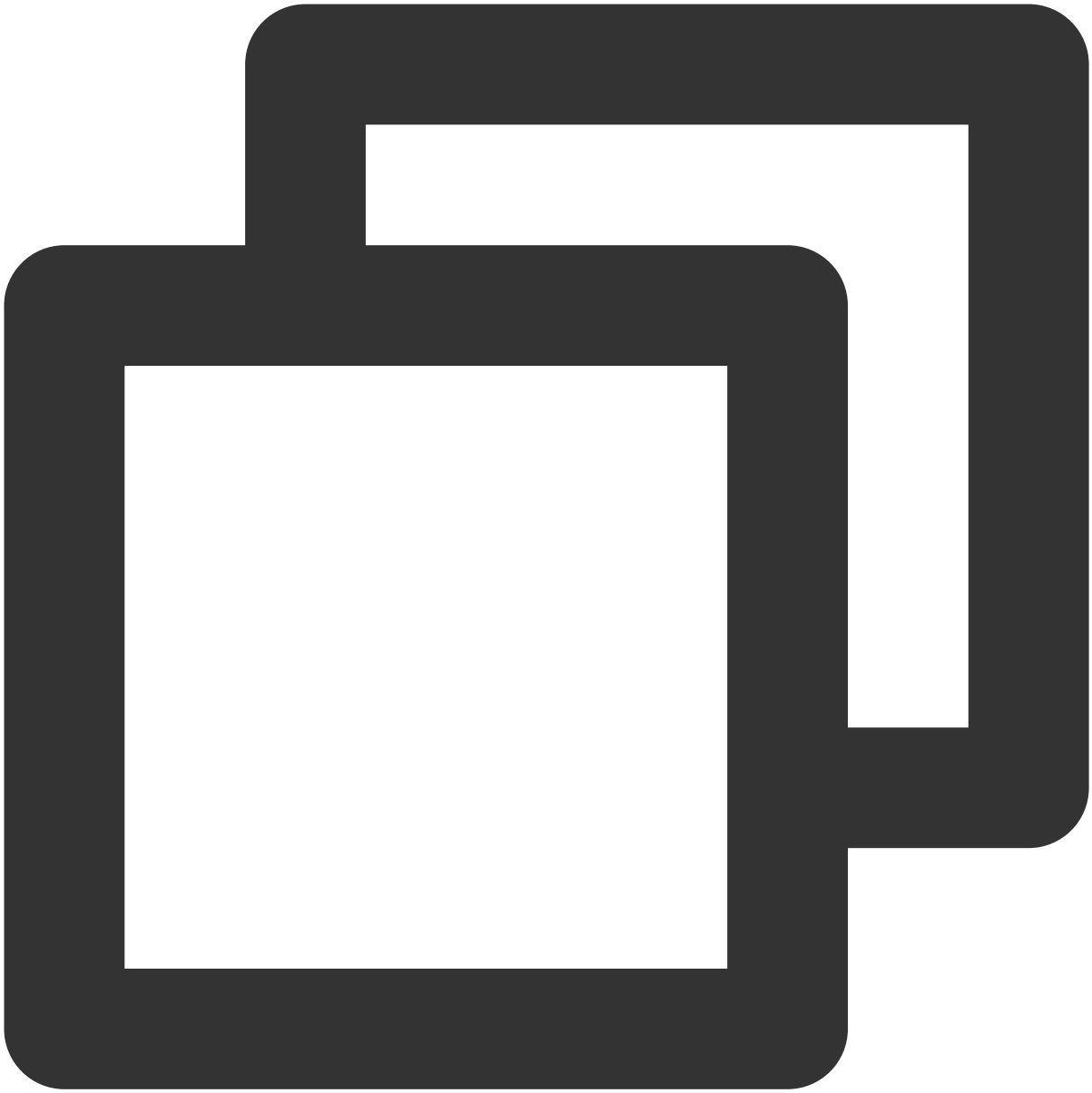
`UserID` can be accessed by the Instant Messaging Console > Account Management . Switch to the target application account to create an account and get the userID.



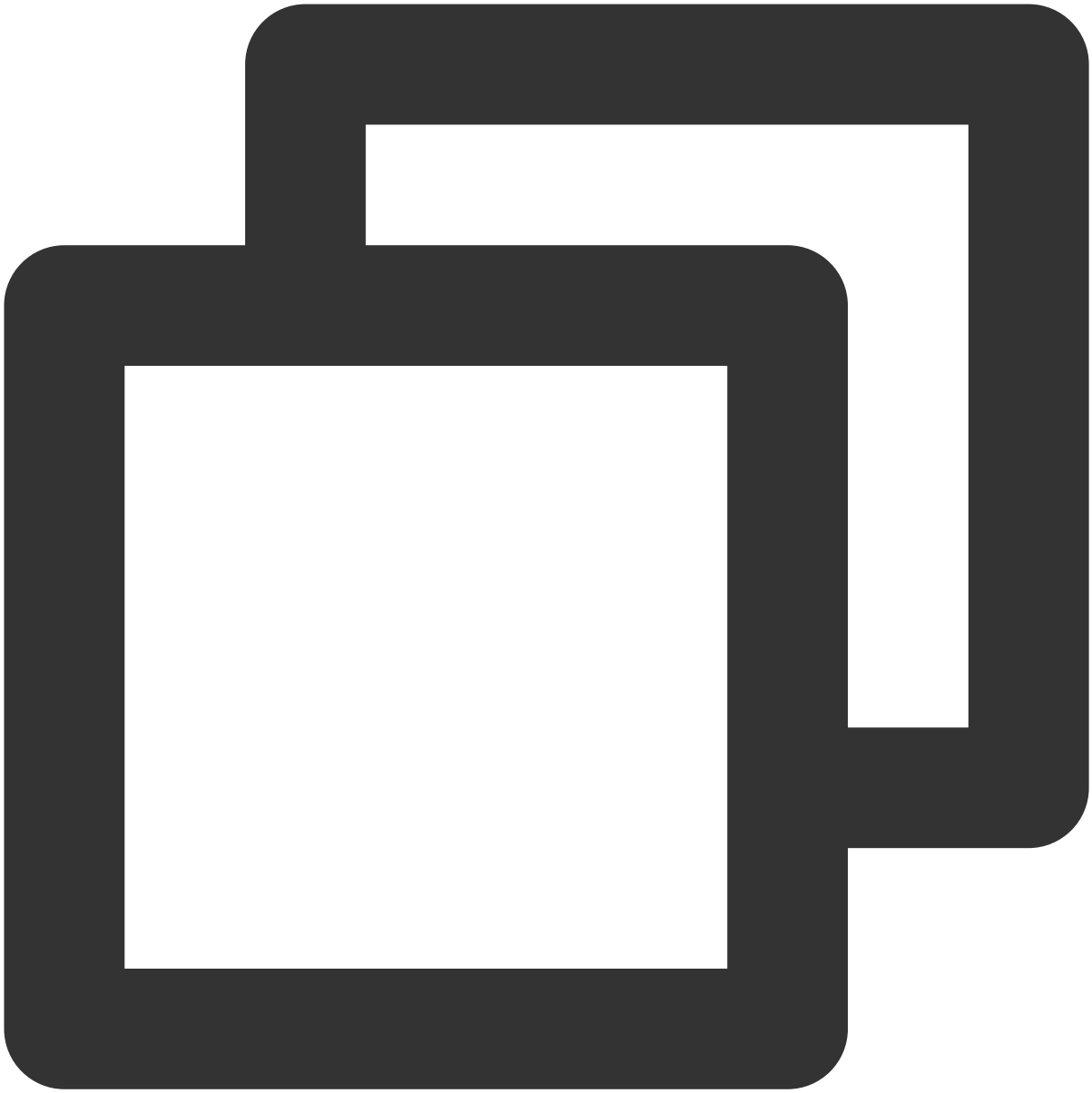
Step 5. Launch the project

vue-cli

vite



```
npm run serve
```

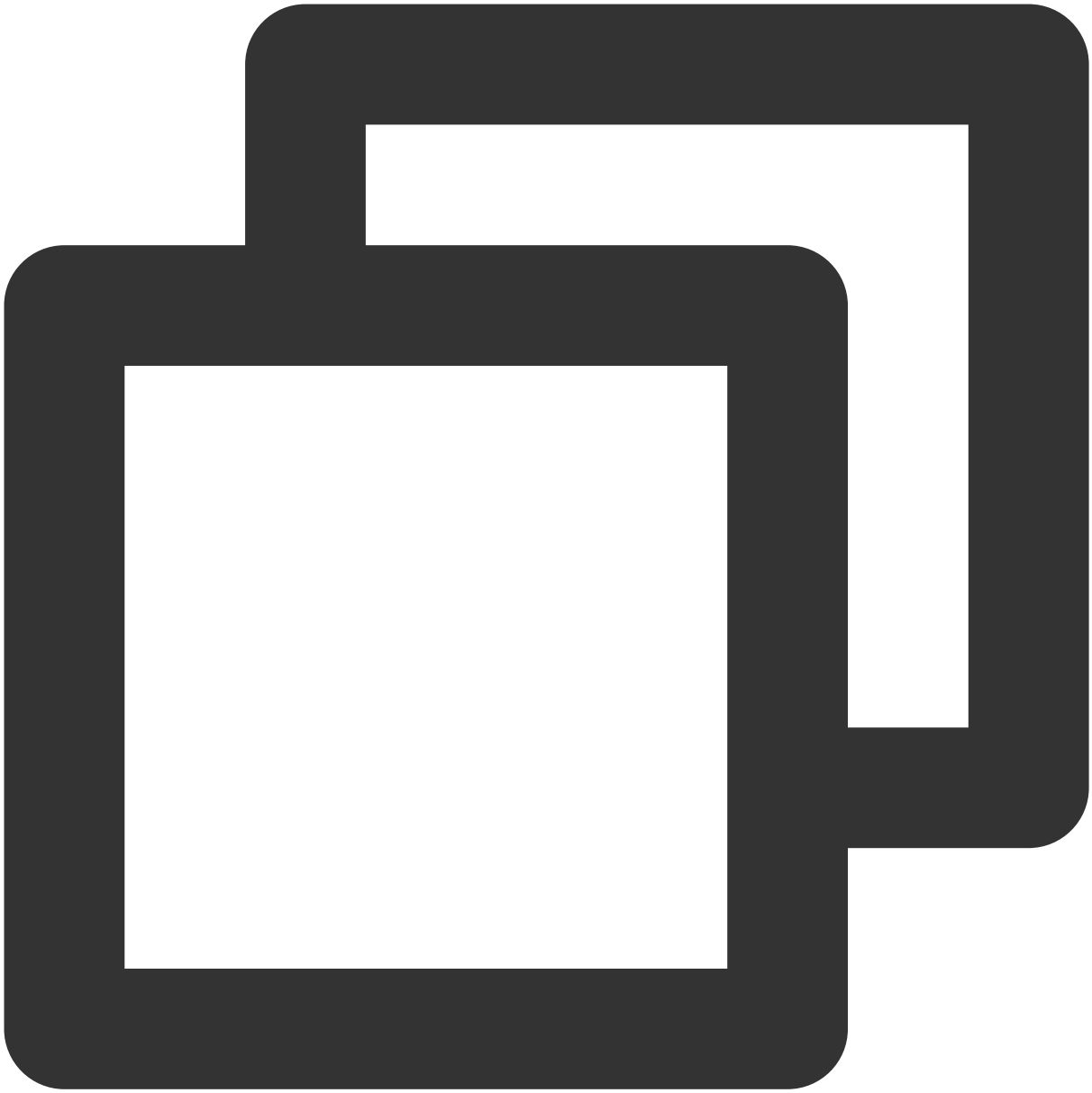


```
npm run dev
```

Additional item: Switching languages

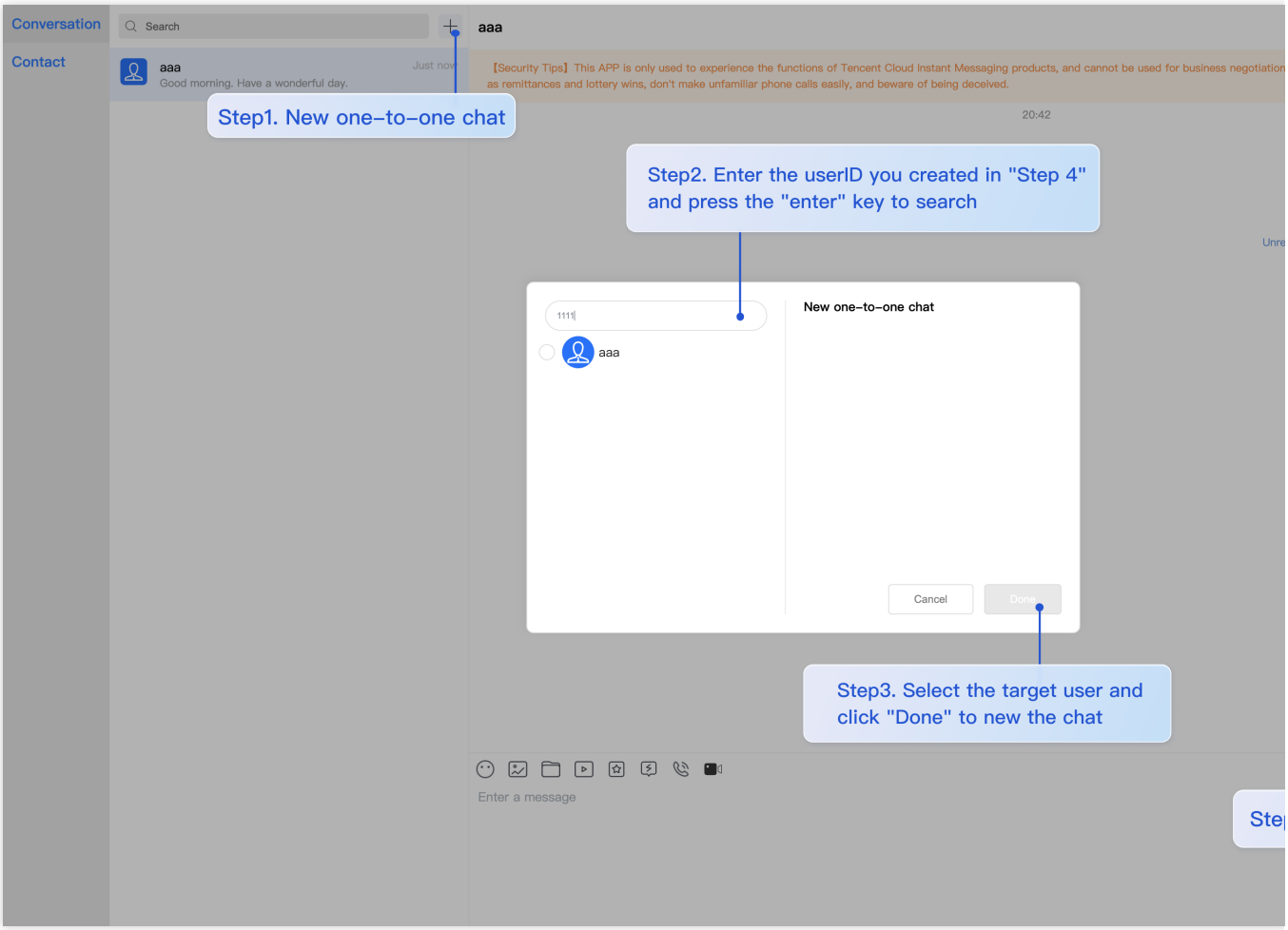
The `Vue TUIKit` comes with default **Simplified Chinese, English** language packages that serve as the interface display language.

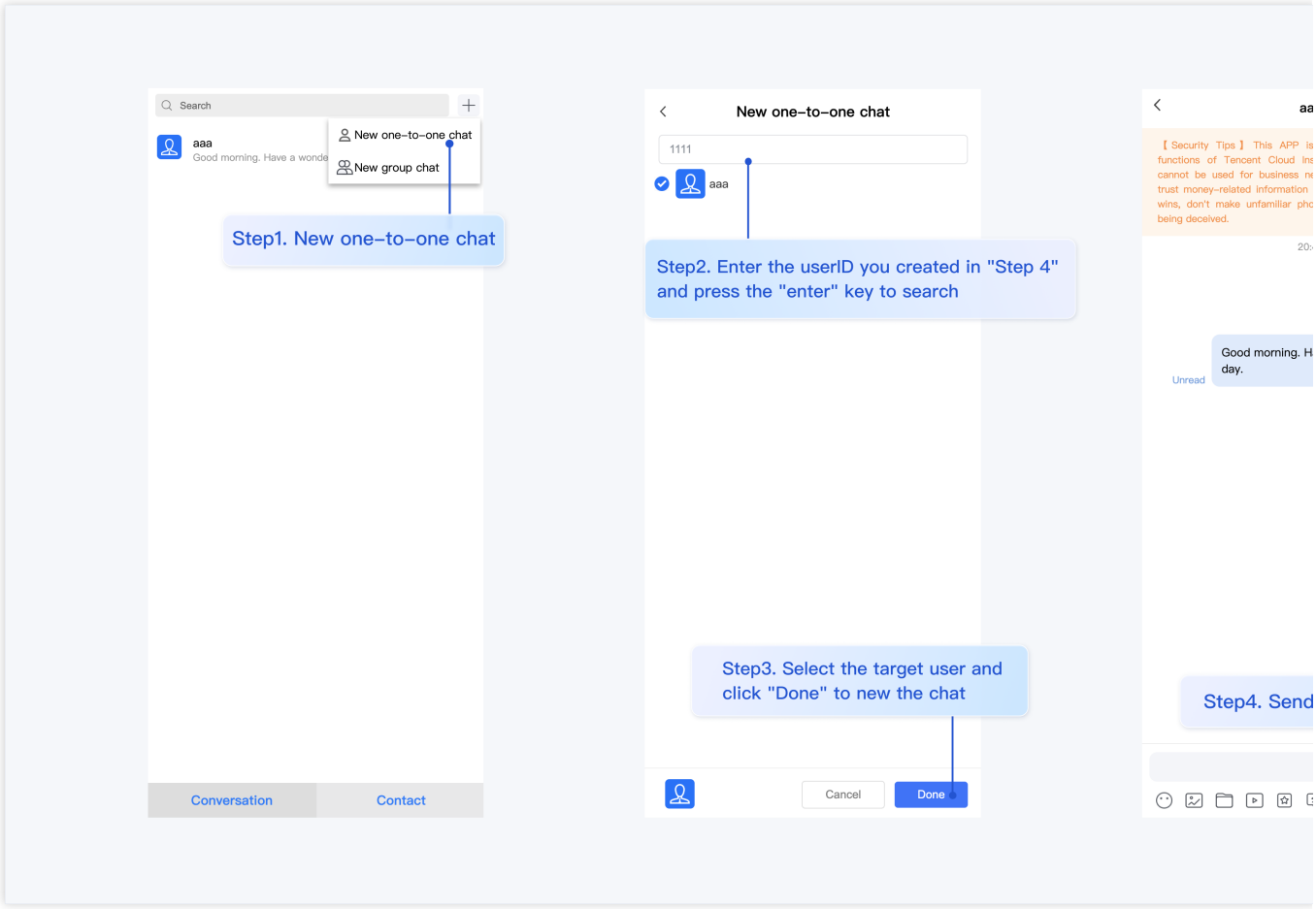
You may switch languages through the following methods.



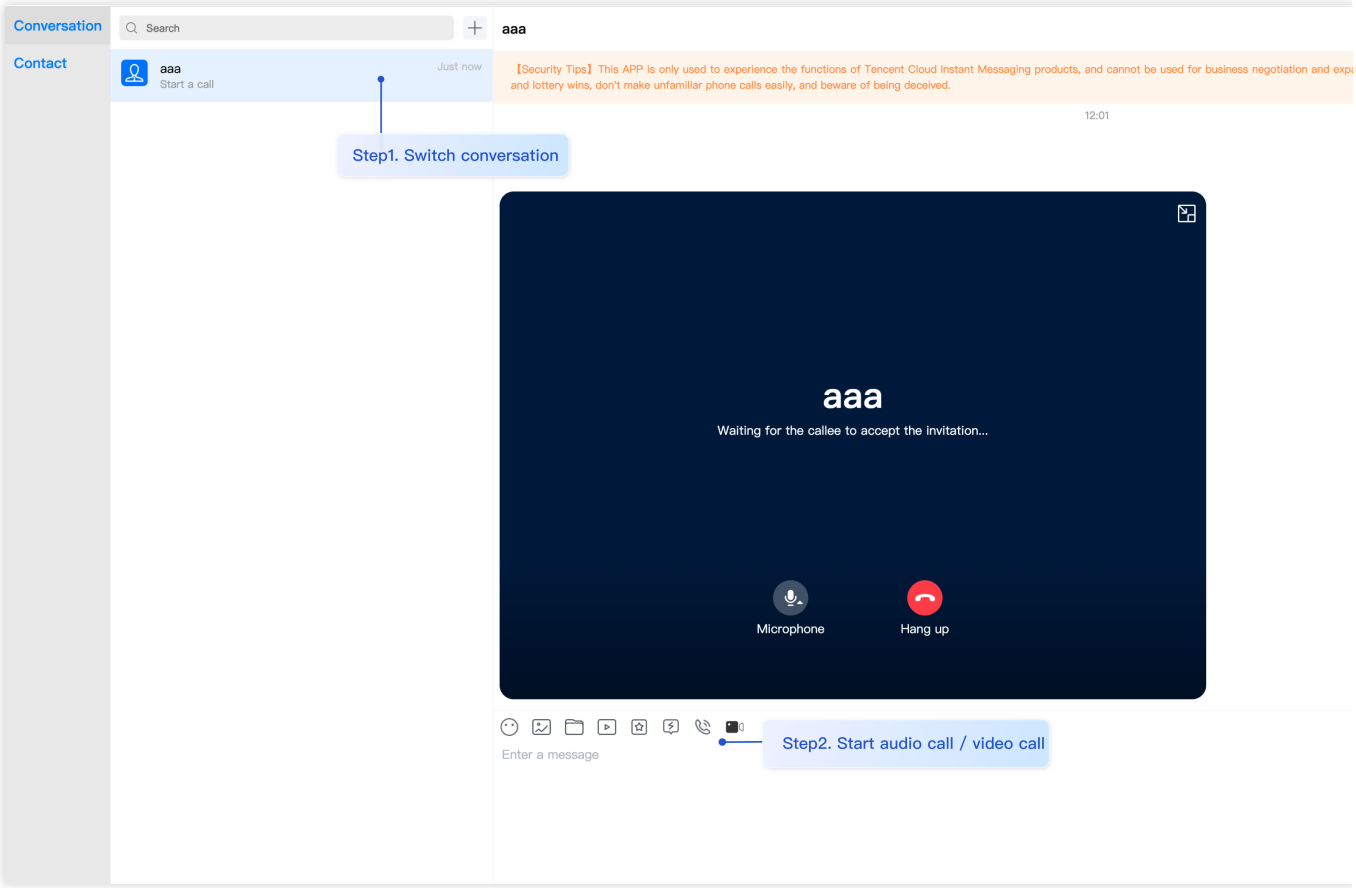
```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

Step 6. Send your first message





Step 7: Make your first phone call



FAQs

Product Service FAQs

1. The Audio/Video Call Capability package is not activated? Failure to initiate the Audio/Video Call?

Please click Audio/Video Call > Frequently Asked Questions to view the solutions.

2. What is UserSig? How is UserSig generated?

A UserSig is a password with which you can log in to use IM service. It is the ciphertext generated by encrypting information such as userID.

The issuance of UserSig is achieved by integrating the calculation code for UserSig into your server-side, whilst providing an interface designed for your project. Whenever UserSig is required, your project could request the operational server for a dynamic UserSig. For further information, please refer to Generating UserSig on the server-side.

Caution

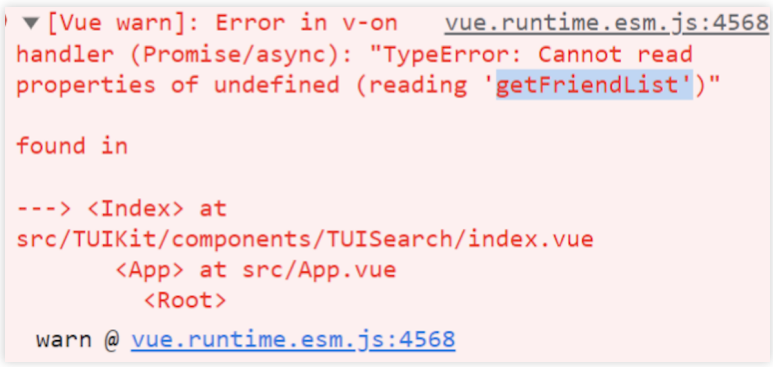
The method to obtain UserSig demonstrated in this document utilizes the configuration of a SECRETKEY within the client-side code. Within this procedure, the SECRETKEY is notably vulnerable to decompilation and reverse-

engineering. Should your SECRETKEY be leaked, malefactors could potentially exploit your Tencent Cloud traffic. Therefore, **this technique is only appropriate for local operation and functional debugging**. For the correct method of issuing UserSig, please refer to the earlier text.

Connection Errors FAQs

1. Runtime error: "TypeError: Cannot read properties of undefined (reading "getFriendList")"

If the following errors occur during runtime after connecting as per the steps outlined above, it is imperative that you **delete the node_modules directory under the TUIKit folder** to ensure the uniqueness of TUIKit's dependencies, preventing issues caused by multiple copies of dependencies.



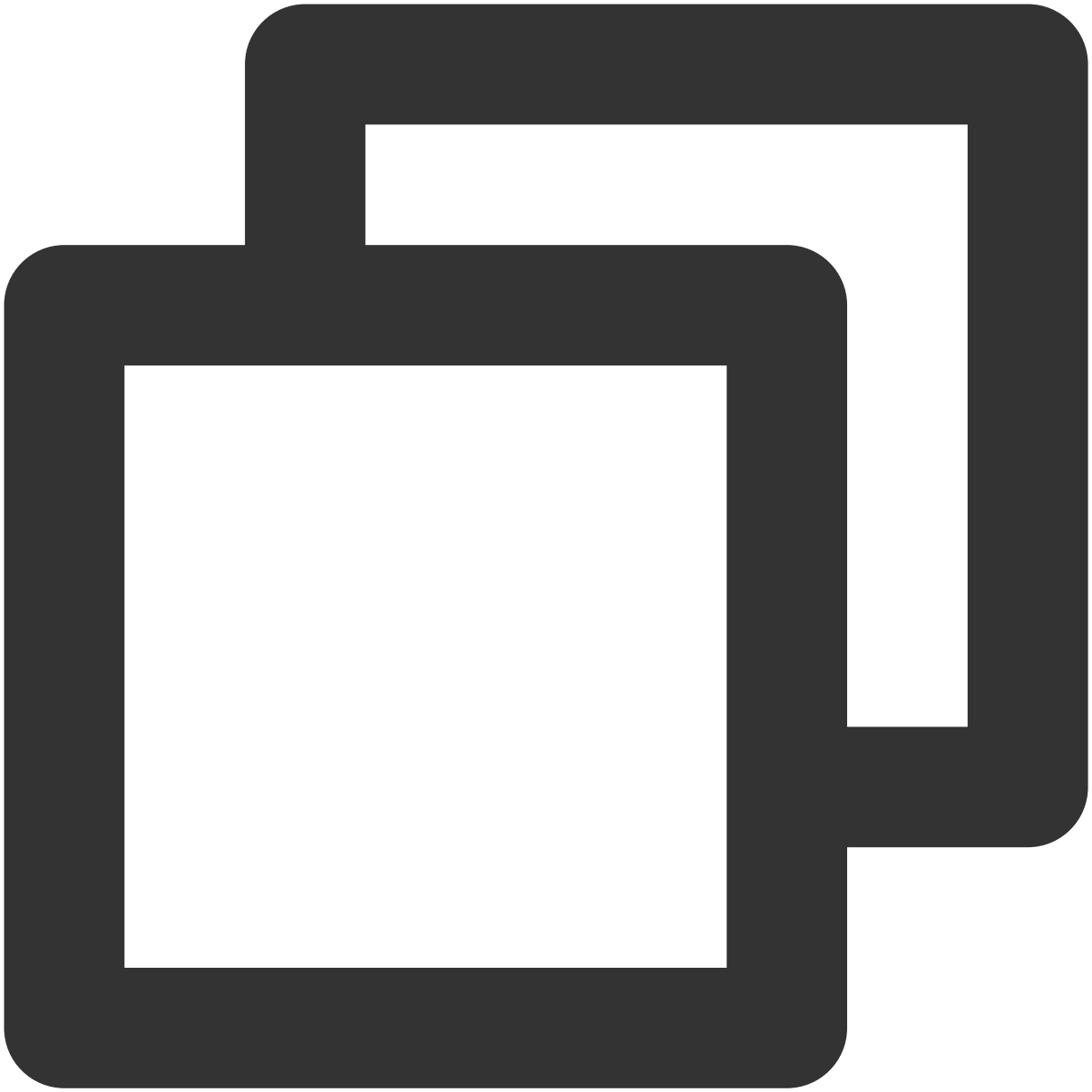
2. How does a JS project integrate the TUIKit component?

TUIKit exclusively supports the TS environment for operation. You can enable the coexistence of existing JS code in your project with the TS code in TUIKit through progressive configuration of TypeScript.

vue-cli

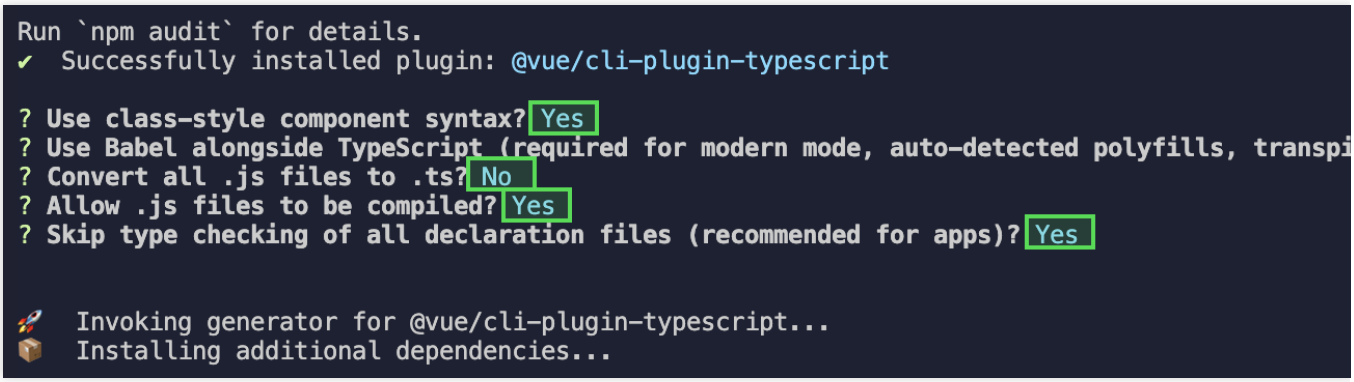
vite

Please execute the following in the root directory of your engineering project created by the Vue CLI scaffold:



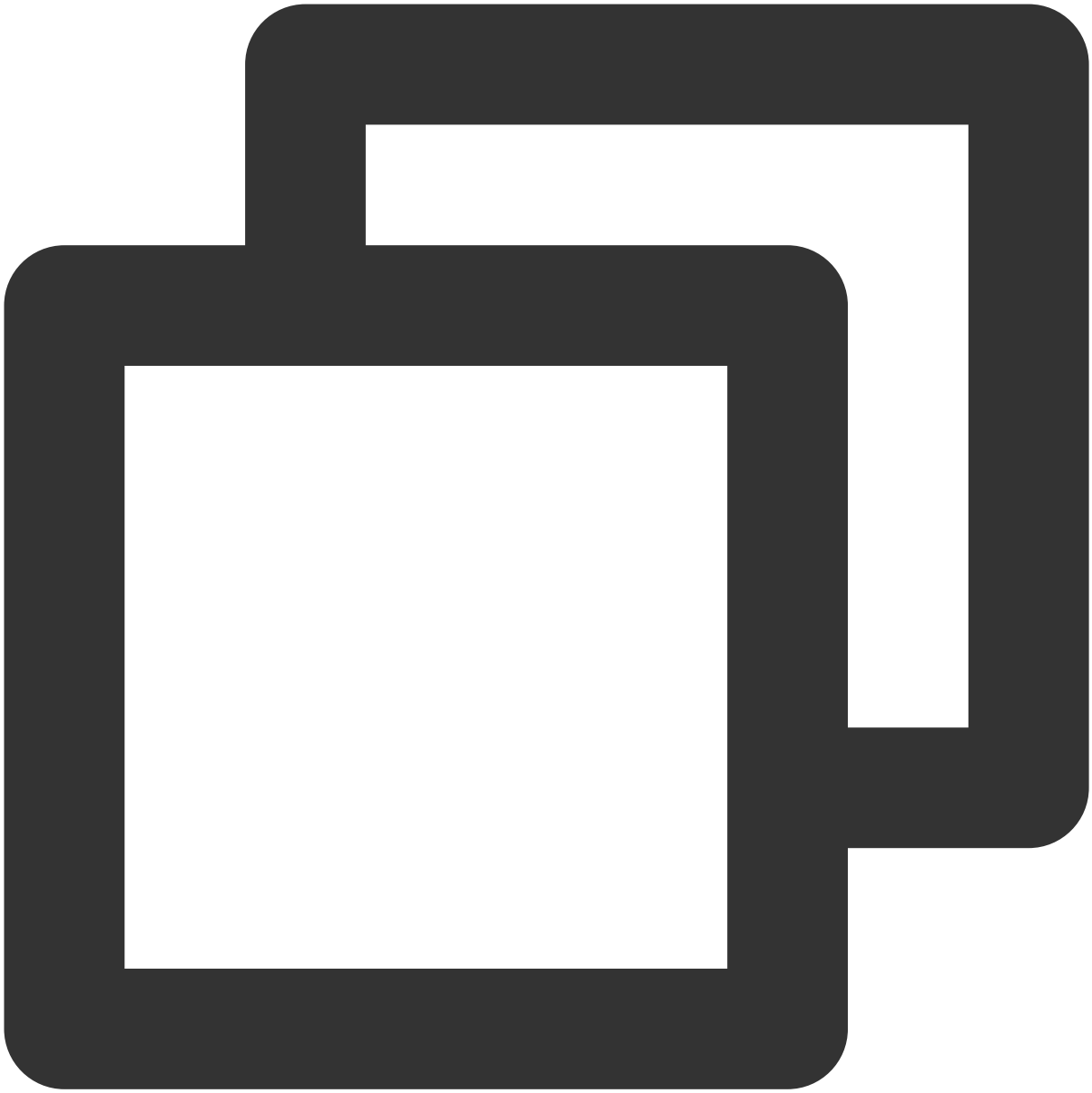
```
vue add typescript
```

Subsequently, please make selections in accordance with the following configuration options. To assure that we can support both the existing js code and the ts code within TUIKit, it is imperative that you strictly adhere to the five options presented below.



Once these steps are completed, please rerun the project!

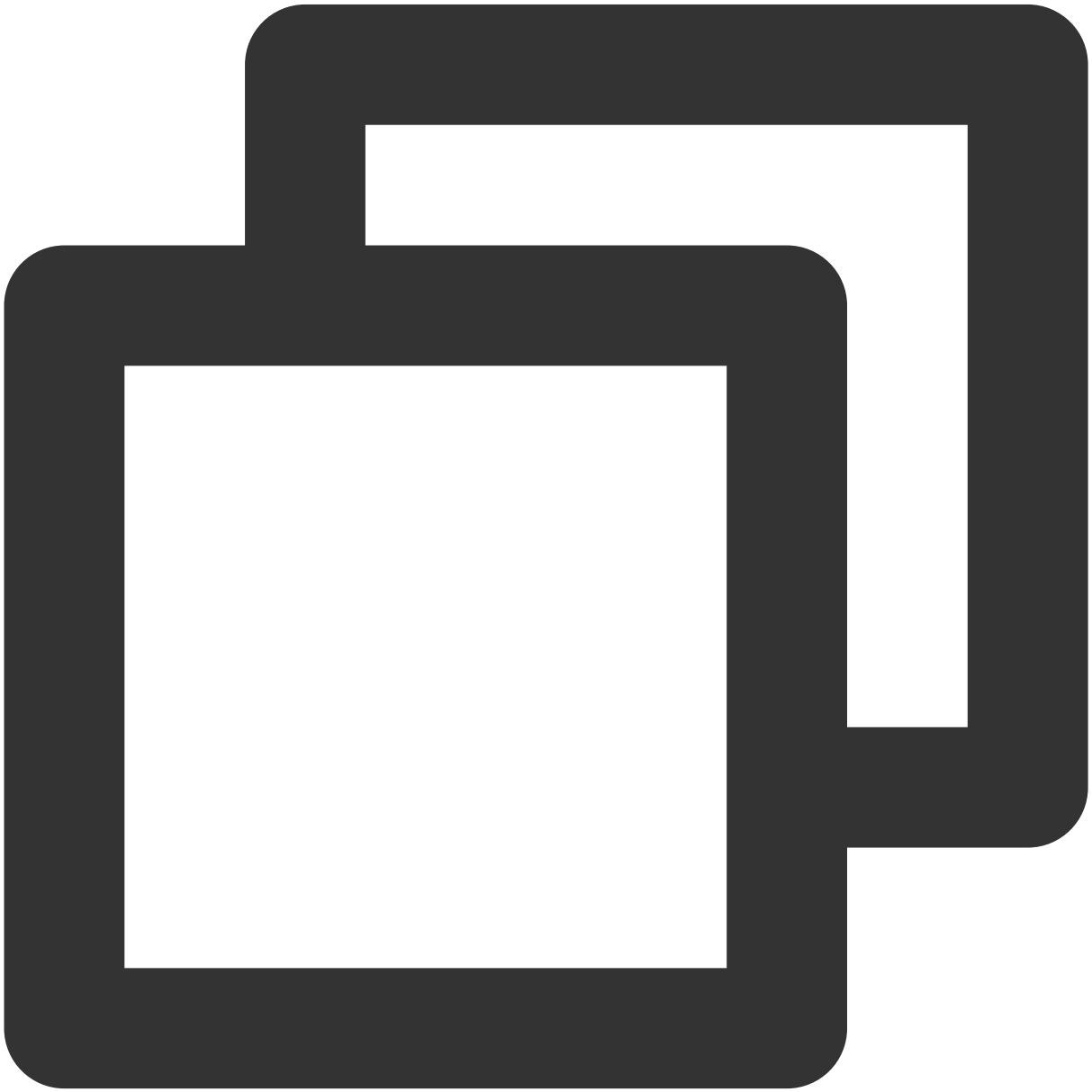
Please execute the following command in your project's root directory created with vite:



```
npm install -D typescript
```

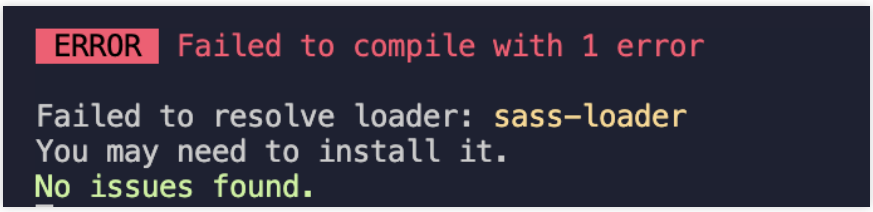
3. Runtime error reported: /chat-example/src/TUIKit/components/TUIChat/message-input/message-input-editor.vue .ts(8,23)TS1005: expected.

The above error message appears because your installed @vue/cli version is too low. **You must ensure that your @vue/cli version is 5.0.0 or higher**. The upgrade method is as follows:

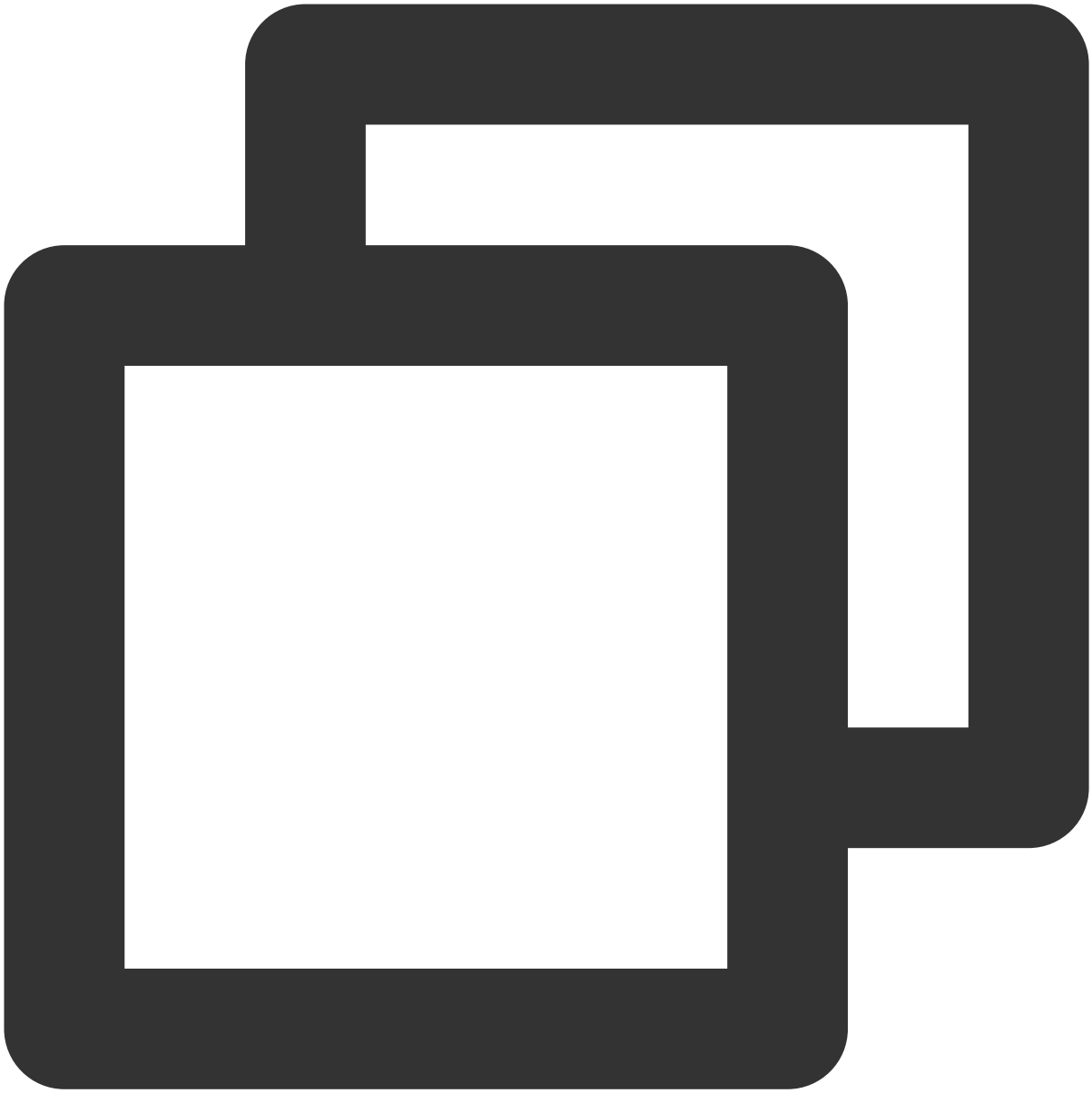


```
npm install -g @vue/cli@5.0.8
```

4. Runtime error: Failed to resolve loader: sass-loader



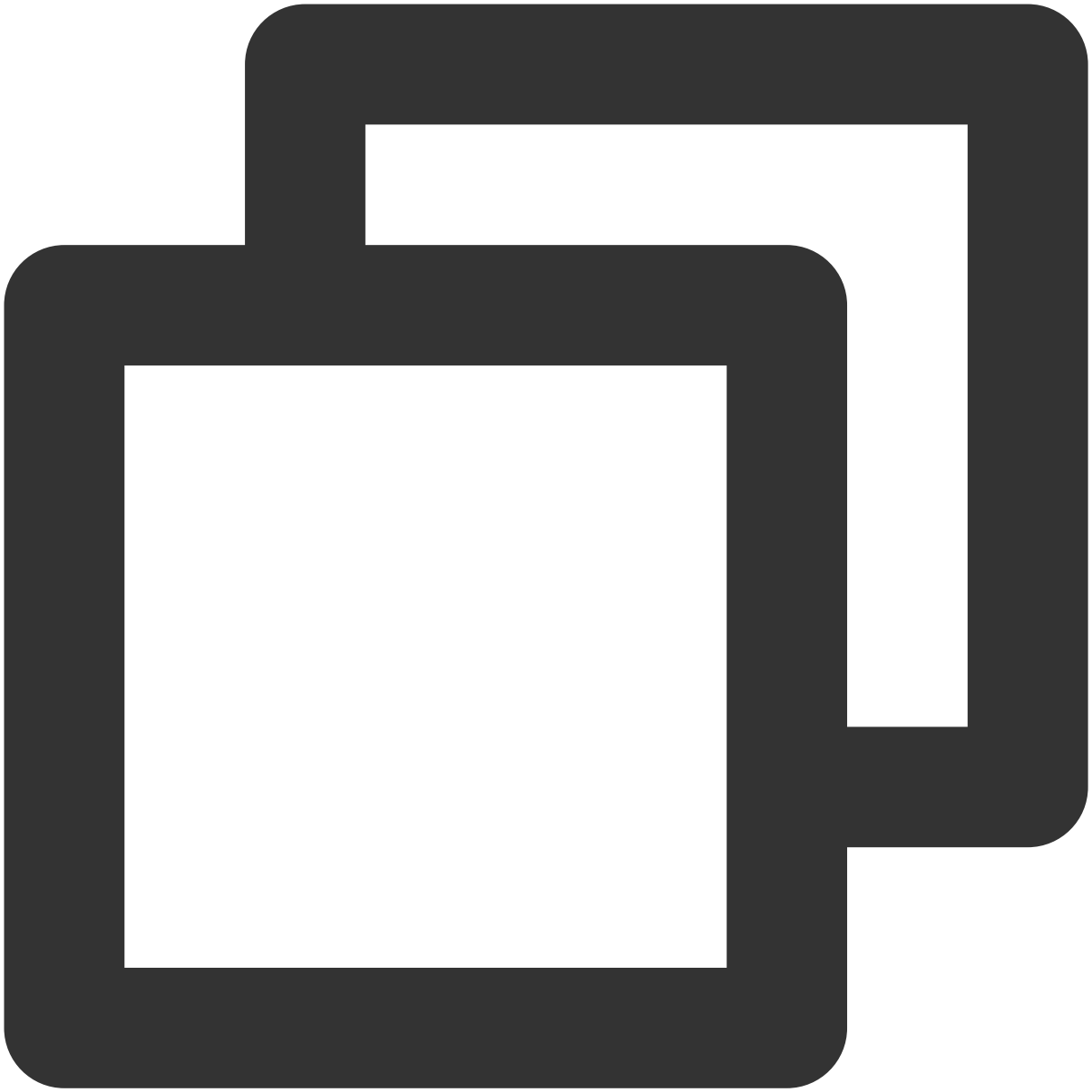
The above error message appears because the `sass` environment is not installed on your machine, please run the following command to install the `sass` environment:



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

5. Other ESLint errors?

If copying chat-uikit-vue to the src directory results in error due to inconsistency with your local project code style, you may ignore this component directory. This can be achieved by adding .eslintignore file to the project root directory:



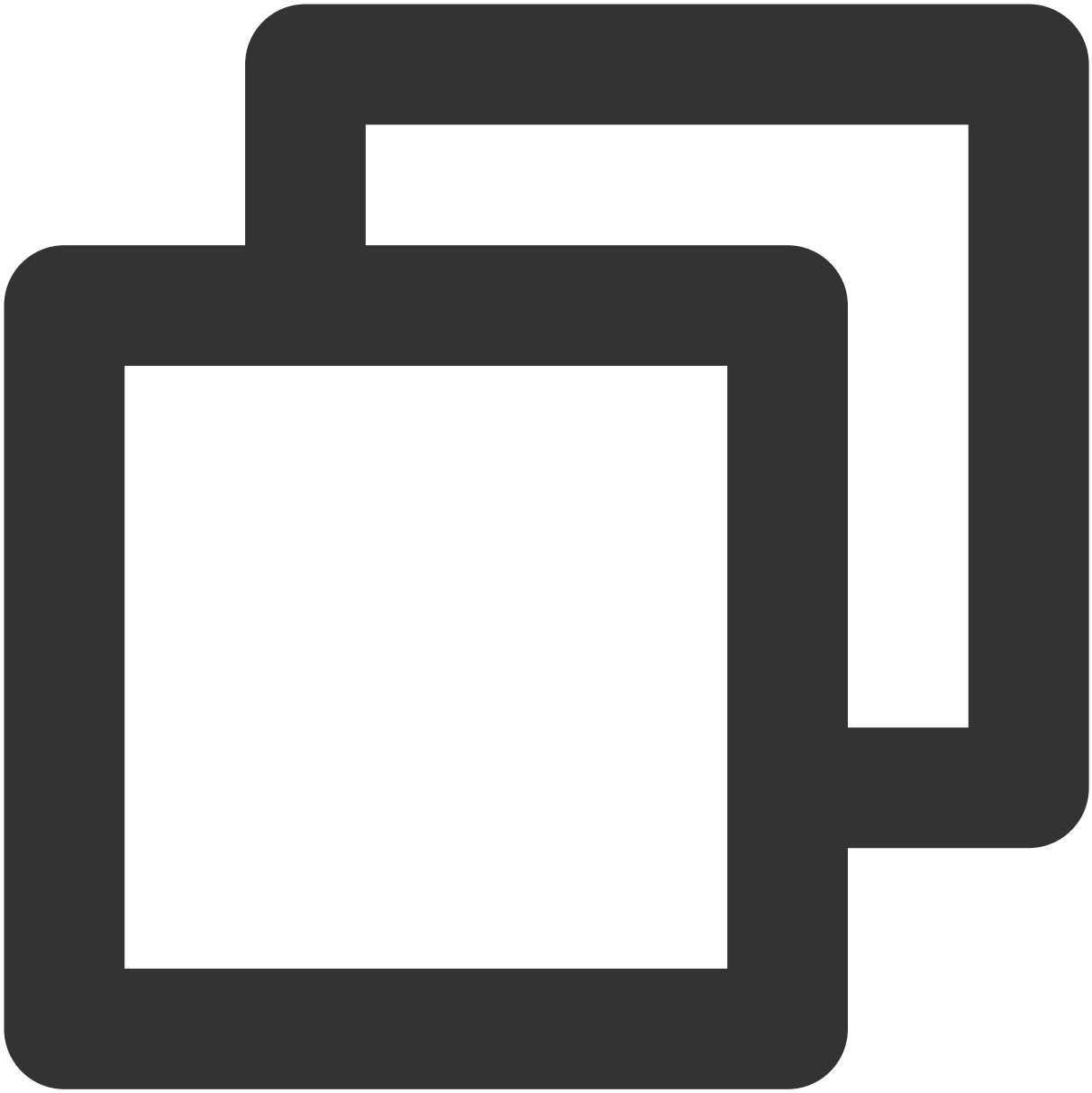
```
# .eslintignore
src/TUIKit
```

6. How to disable the full screen overlay error message prompt of webpack in dev mode in vue/cli?

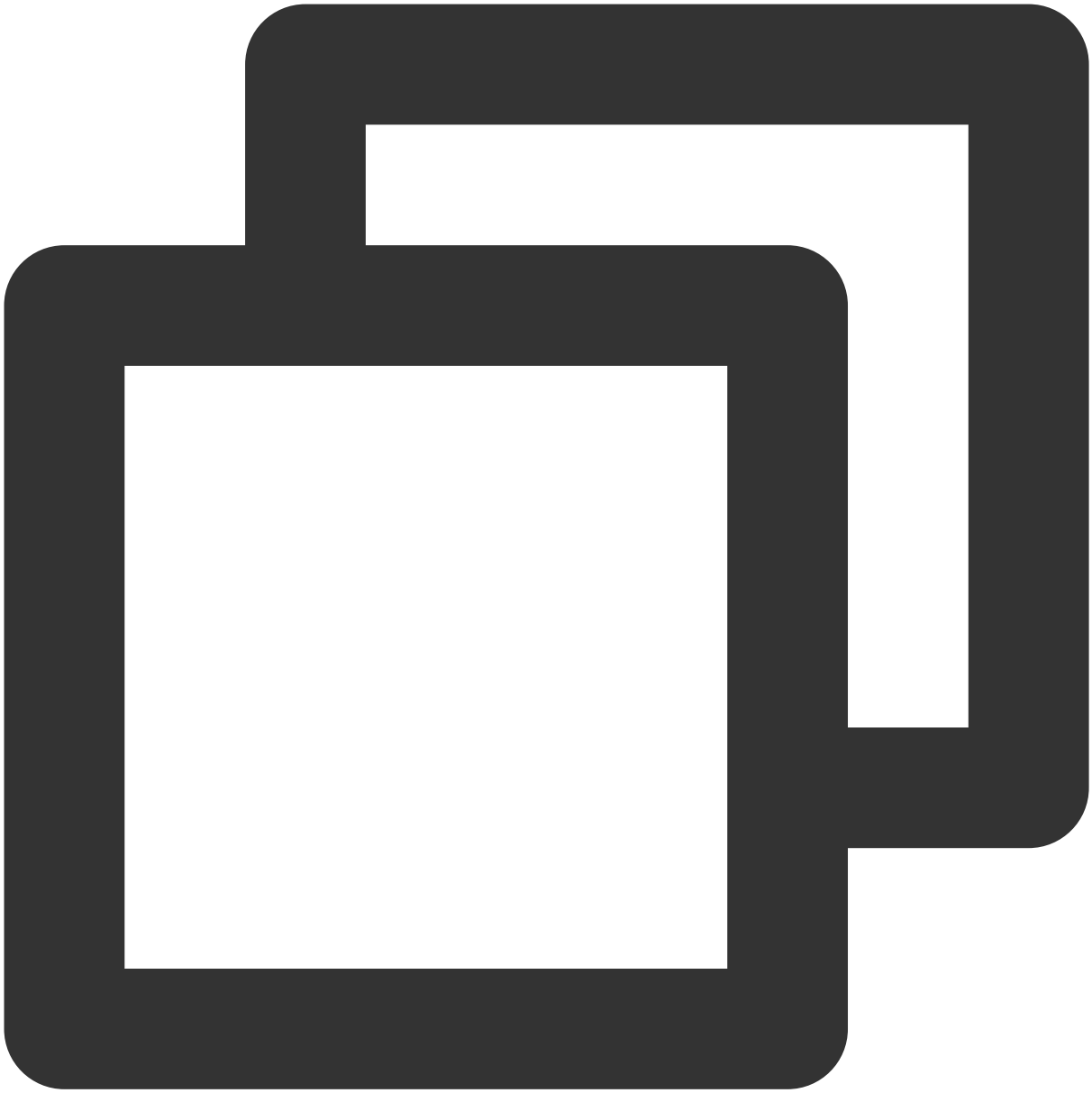
You can disable it in the vue.config.js file at the root directory of your project:

webpack4

webpack3



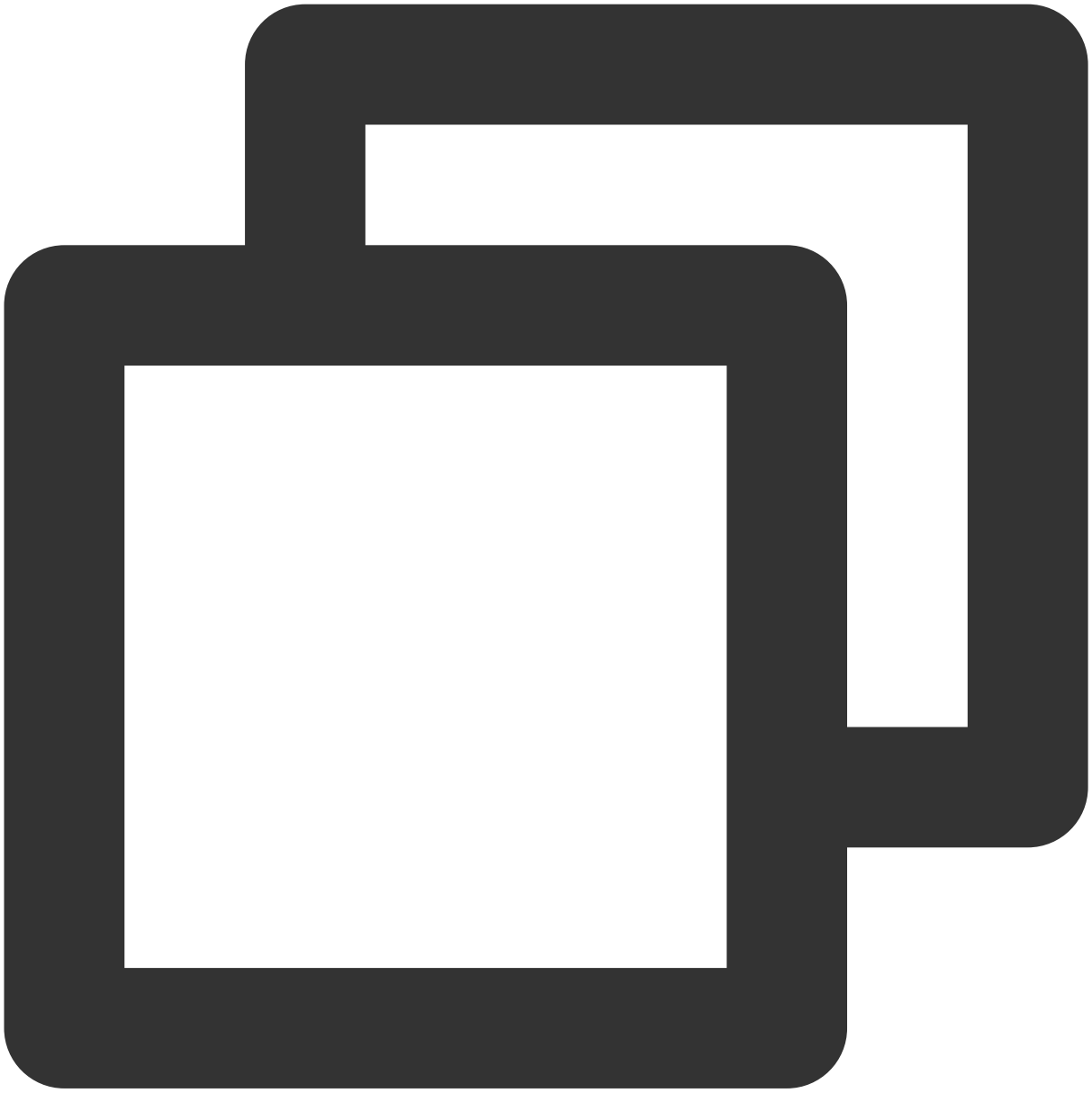
```
module.exports = defineConfig({
  ...
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```



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

7. What to do when encountering 'Component name "XXXX" should always be multi-word'?

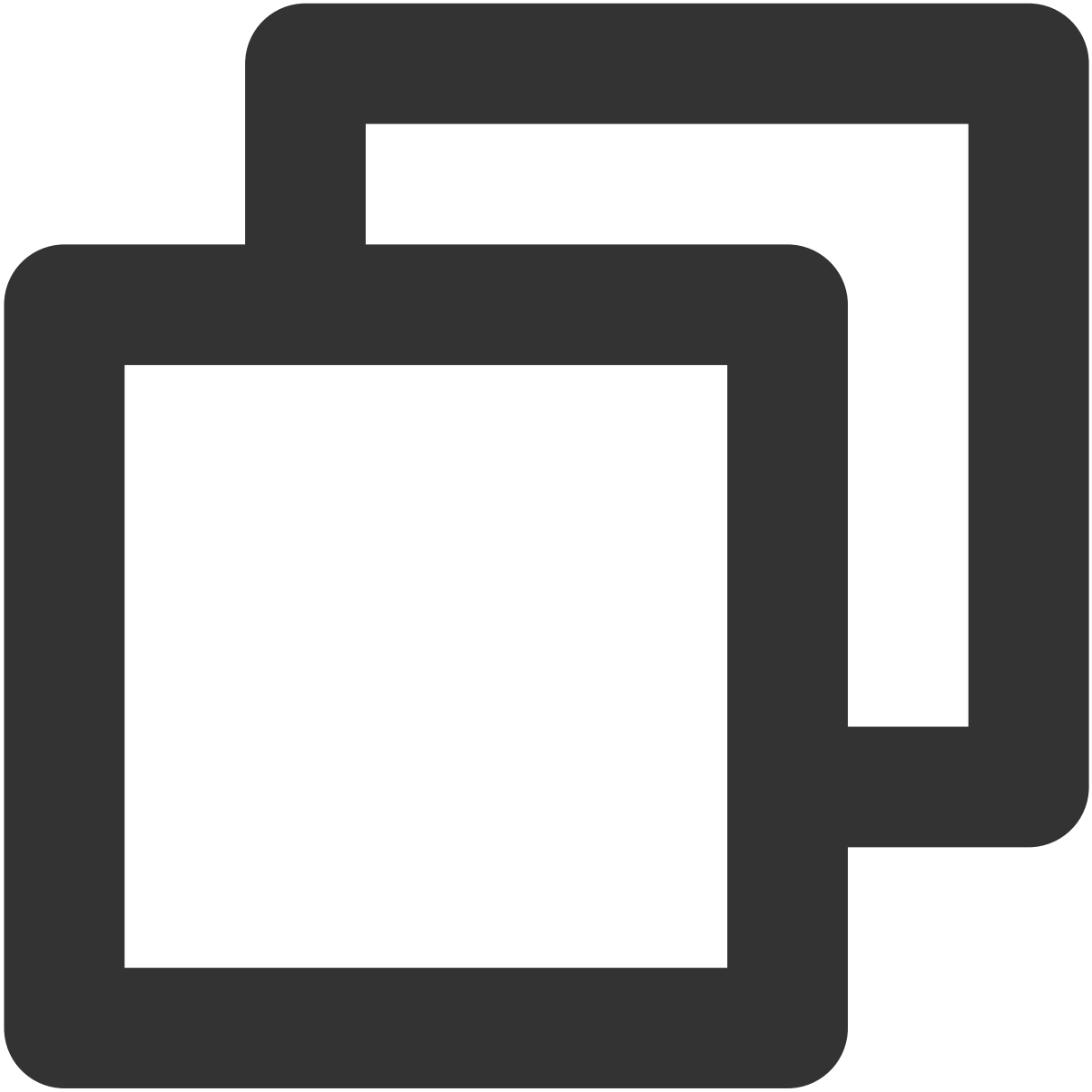
The version of ESLint utilized in IM TUIKit web is v6.7.2, which does not rigidly verify the camelCase format for module names.
Should you encounter this dilemma, you may configure as follows in the .eslintrc.js file:



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

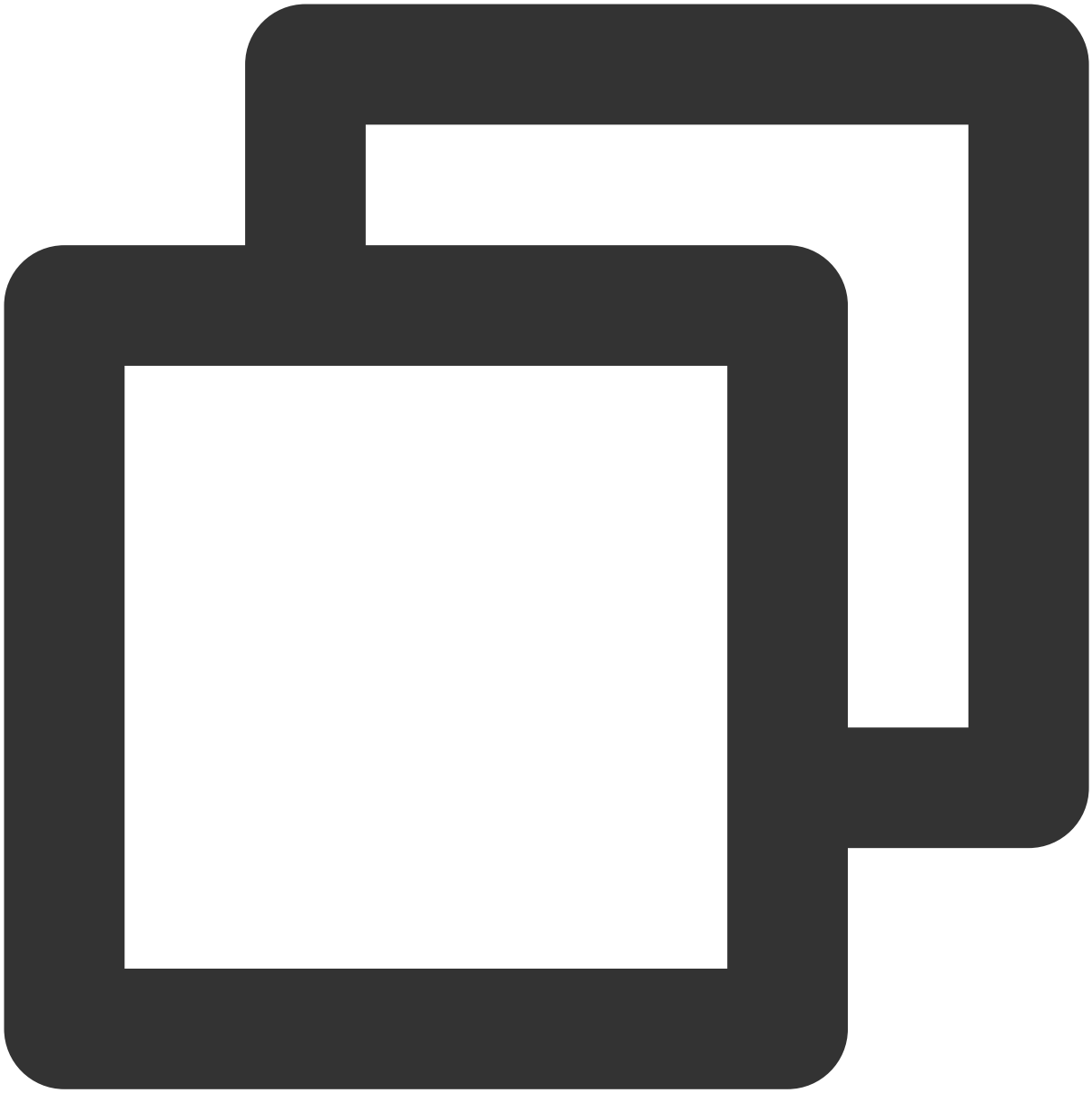
8. What should I do if I encounter ERESOLVE unable to resolve dependency tree?

If ERESOLVE unable to resolve dependency tree appears when npm install is run, it indicates a conflict in dependency installation. The following method can be adopted for installation:



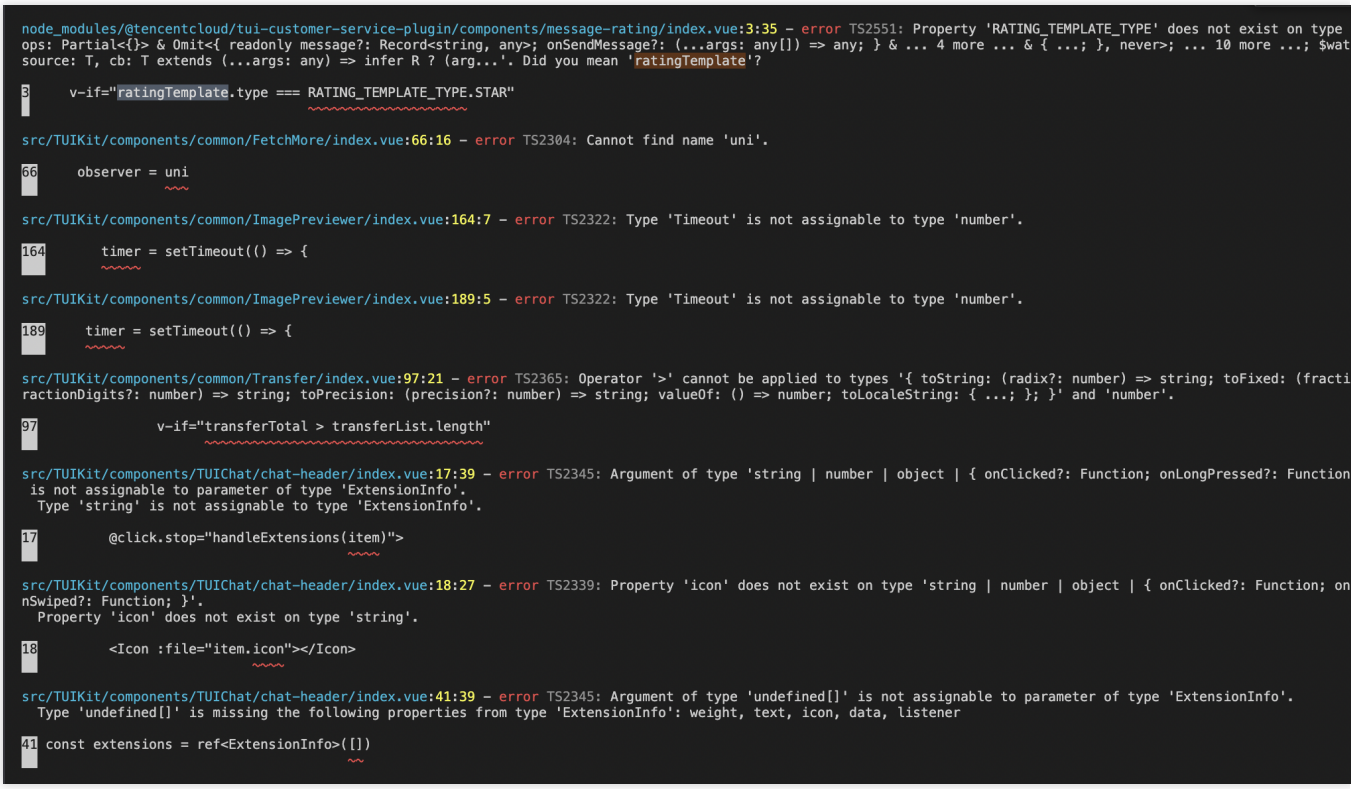
```
npm install --legacy-peer-deps
```

9. How might one address the error message, 'vue packages version mismatch' occurring during execution?



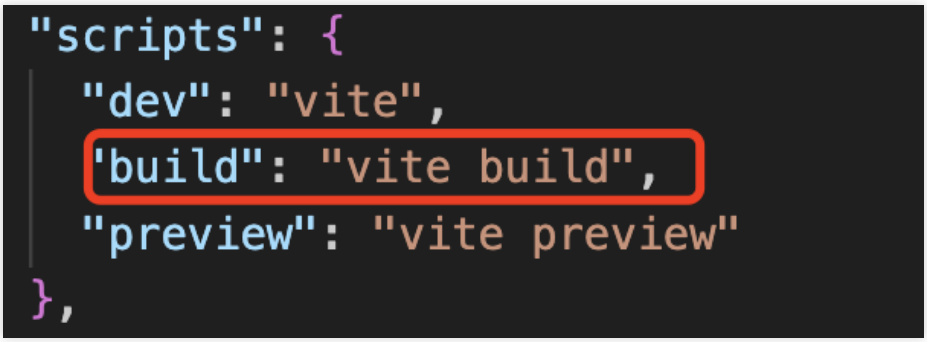
```
// If you are using a vue2.7 project, please execute in your project root directory
npm i vue@2.7.9 vue-template-compiler@2.7.9
// If you have a Vue2.6 project, please execute in your project's root directory
npm i vue@2.6.14 vue-template-compiler@2.6.14
```

10. Why does TypeScript report an error after npm run build in a Vite project?



Reason: It's led by the vue-tsc command in "build": "vue-tsc && vite build" under package.json script.

Solution: Simply remove vue-tsc. "build": "vite build"



Contact Us

Join the Telegram technical discussion group or WhatsApp discussion group, enjoy the support of professional engineers, and solve your difficulties.

Documentation

Related to Vue2 & Vue3 UIKit:

chat-uikit-vue npm
Vue2 Demo Source Code and Running Example
Vue3 Demo Source Code and Running Example

Vue2 & Vue3 UIKit logic layer: engine

chat-uikit-engine npm
chat-uikit-engine interface

クイックスタート（Unity）

最終更新日：：2024-04-11 16:20:42

このドキュメントを読むことで、Unity SDKを統合する方法を学ぶことができます。

環境要件

環境	バージョン
Unity	2019.4.15f1以降のバージョン。
Android	Android Studio 3.5以降のバージョン。AppはAndroid 4.1以降のバージョンのデバイスが必要です。
iOS	Xcode 11.0以降のバージョン。プロジェクトに有効な開発者署名を設定済みであることを確認してください。

サポートするプラットフォーム

Unityのすべてのプラットフォームをサポートする IM SDKの構築に取り組み、1つのコードセットですべてのプラットフォームで実行することを支援します。

プラットフォーム	IM SDK
iOS	サポートあり
Android	サポートあり
macOS	サポートあり
Windows	サポートあり
Web	サポートあり、1.8.1以降のバージョン

説明：
Webプラットフォームは、いくつかの追加手順を簡単に導入する必要があります。詳細については、このドキュメントのその5をご確認ください。

前提条件

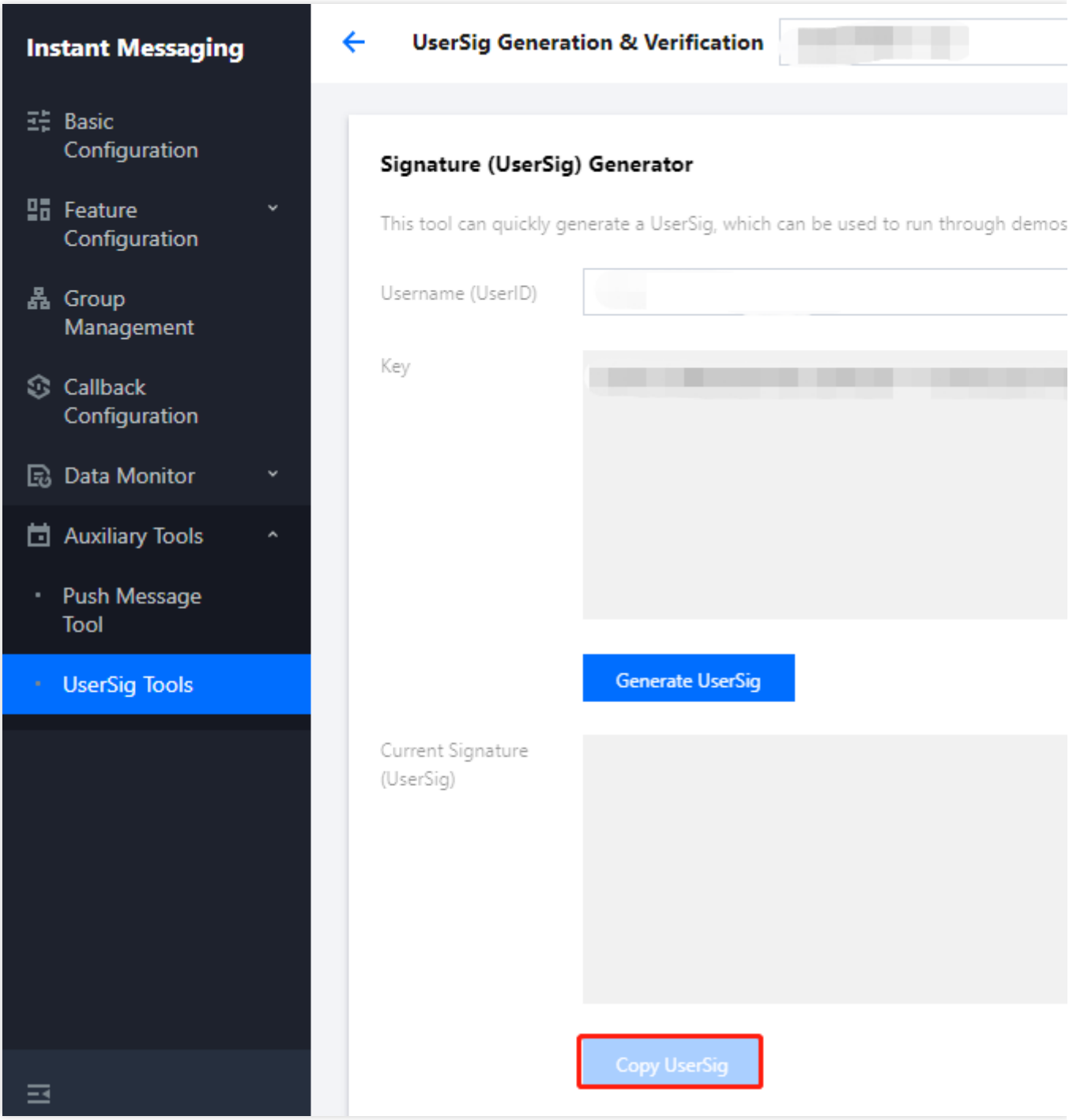
1. Tencent Cloudアカウントの登録を行い、実名認証が完了していること。
2. アプリケーションの作成とアップグレードを参照してアプリケーションを作成し、`SDKAppID` を記録していること。

その1：テストユーザーの作成

IMコンソールでご自分のアプリケーションを選択し、左側ナビゲーションバーで**支援ツール->UserSig生成&検証**の順にクリックし、2つのUserIDおよびそれに対応するUserSigを作成します。`UserID` 、 `署名（Key）` 、 `UserSig` の3つをコピーし、その後のログインで使用します。

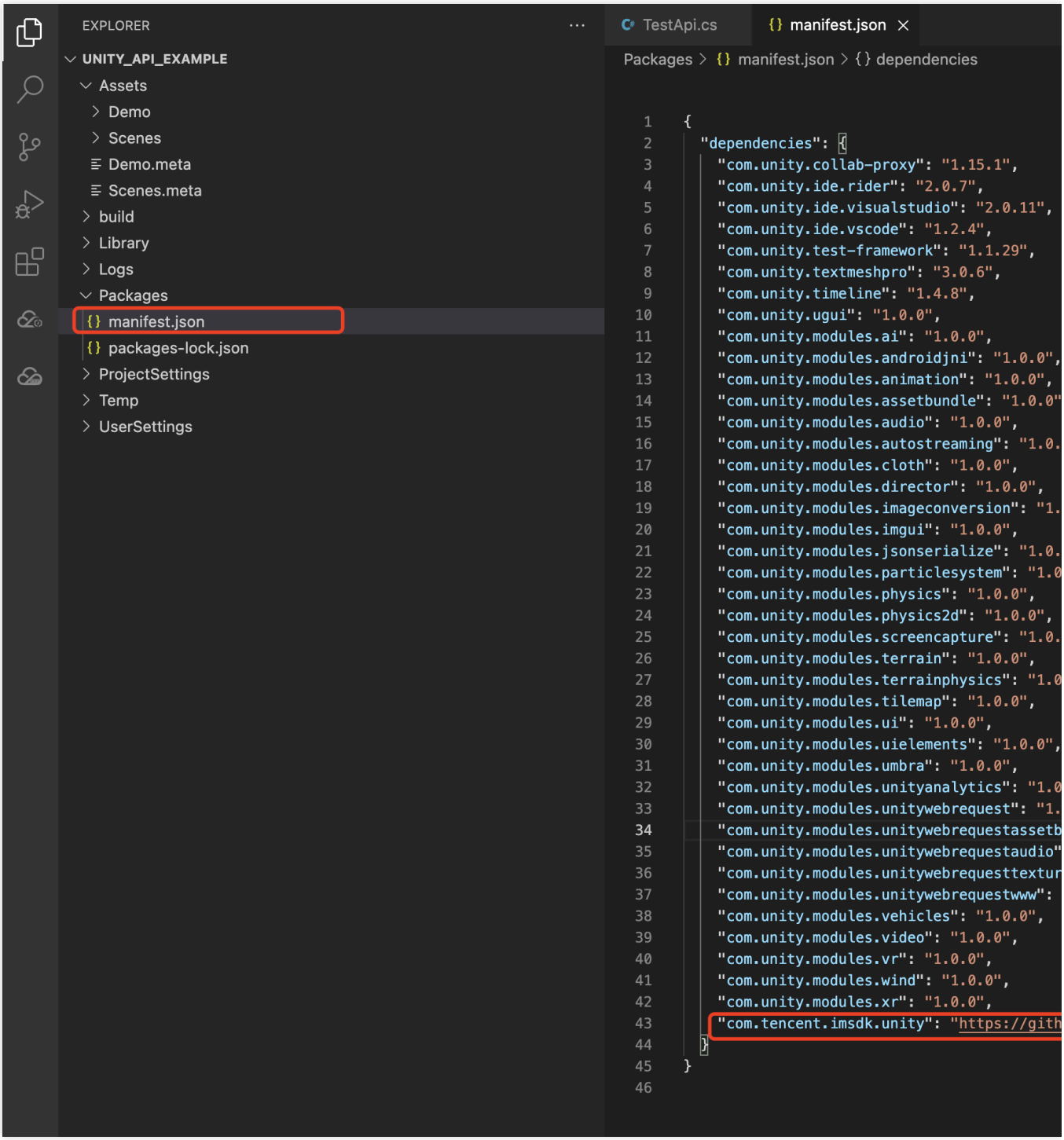
説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、サーバーで生成し、Appのインターフェース向けに提供する方法となります。 `UserSig` が必要なときは、Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

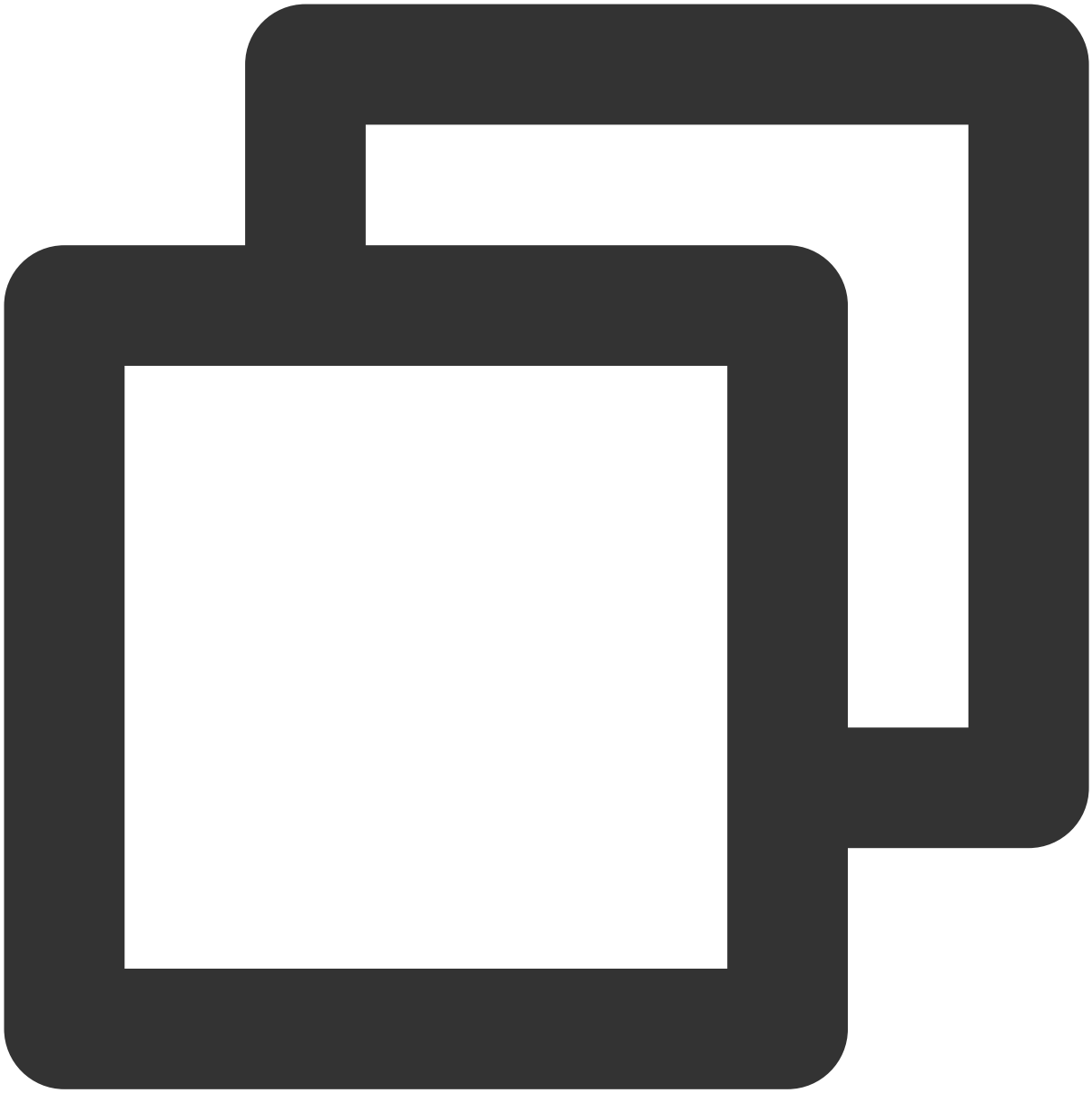


その2：UnityプロジェクトへのIM SDK統合

1. UnityからUnityプロジェクトを作成し、プロジェクトのある場所を記録しておきます。
または、既存のUnityプロジェクトを開きます。
2. IDE（例：Visual Studio Code）からプロジェクトを開きます。



3. ディレクトリに基づいてPackages/manifest.jsonを見つけて次のように依存を修正します：

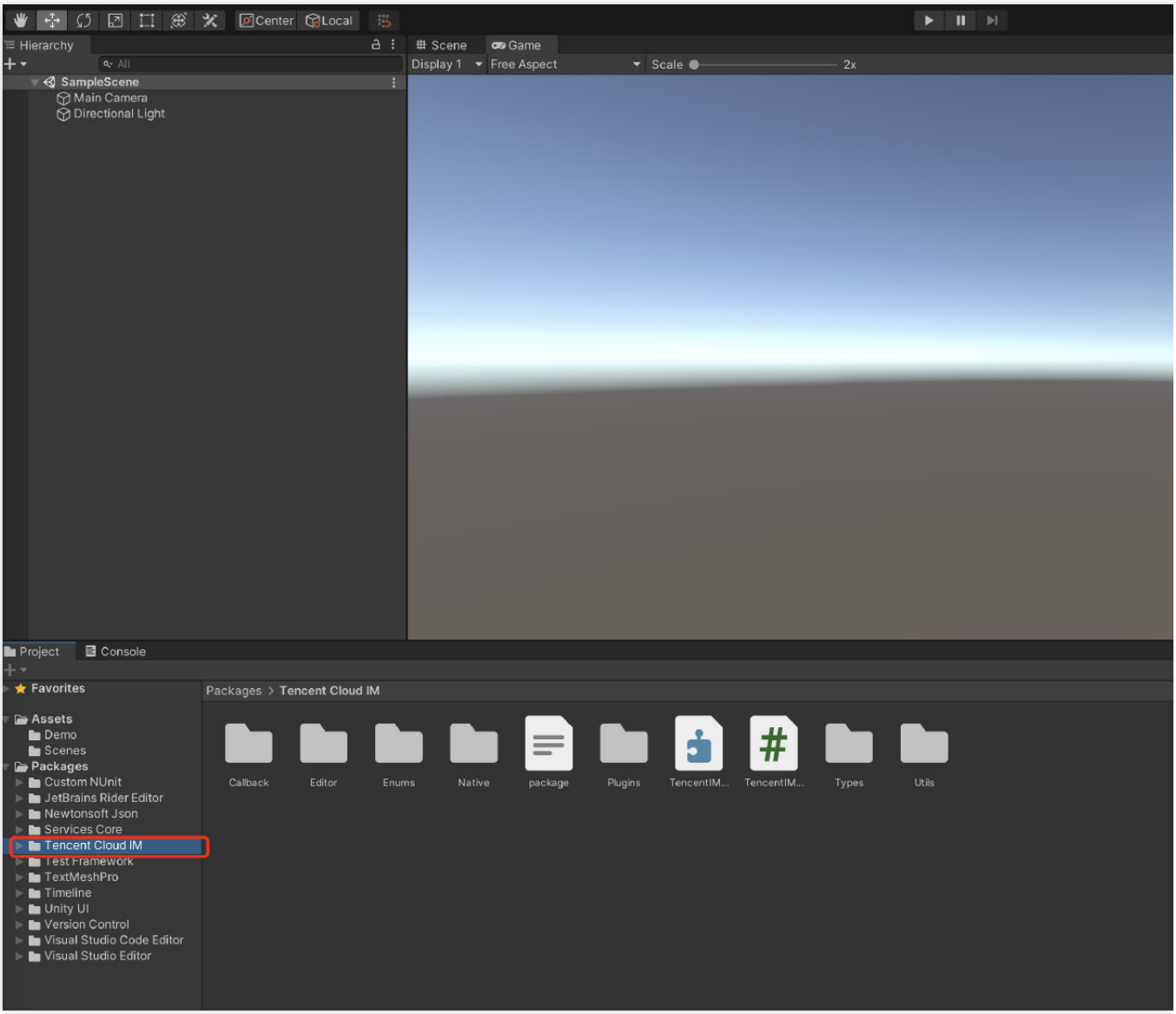


```
{
  "dependencies":{
    "com.tencent.imsdk.unity":"https://github.com/TencentCloud/chat-sdk-unity.git#u
  }
}
```

IM SDKの各APIをよりよく理解させるために、API Exampleをさらに提供して、各APIの呼び出しと監視のトリガーを示します。

その3：ロードの依存性

Unity Editorでプロジェクトを開き、依存のロードが完了してから、Tencent Cloud IMのロードが完了したことを確認します。



その5：UI統合の自己実装

前提条件

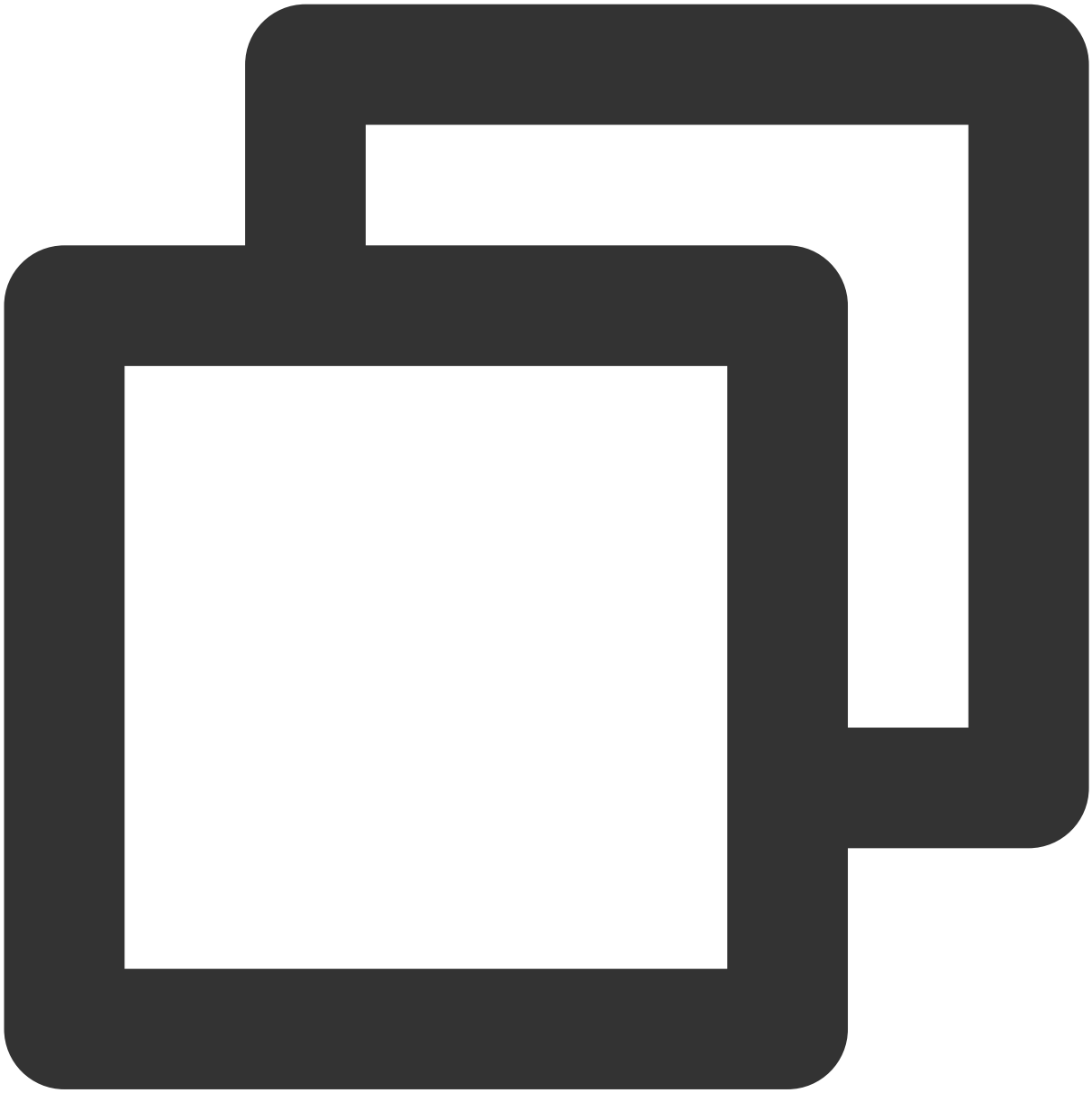
Unityプロジェクトの作成が完了したか、基盤となるUnityプロジェクトが作成されて、Tencent Cloud IM SDKがロードされました。

SDKの初期化完了

このセグメントの詳細なドキュメント

`TencentIMSDK.Init` を呼び出して、SDK初期化を完了します。

お客様の `SDKAppID` を渡します。



```
public static void Init() {
  int SDKAppID = 0; // IMコンソールからアプリケーションSDKAppIDを取得します。
  SdkConfig sdkConfig = new SdkConfig();

  sdkConfig.sdk_config_config_file_path = Application.persistentDataPath + "/TIM-Co

  sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM-Log";
```

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

`Init` の後、主にネットワークの状態やユーザー情報の変更など、IM SDK用のいくつかの監視をマウントできます。詳細については、このドキュメントをご参照ください。

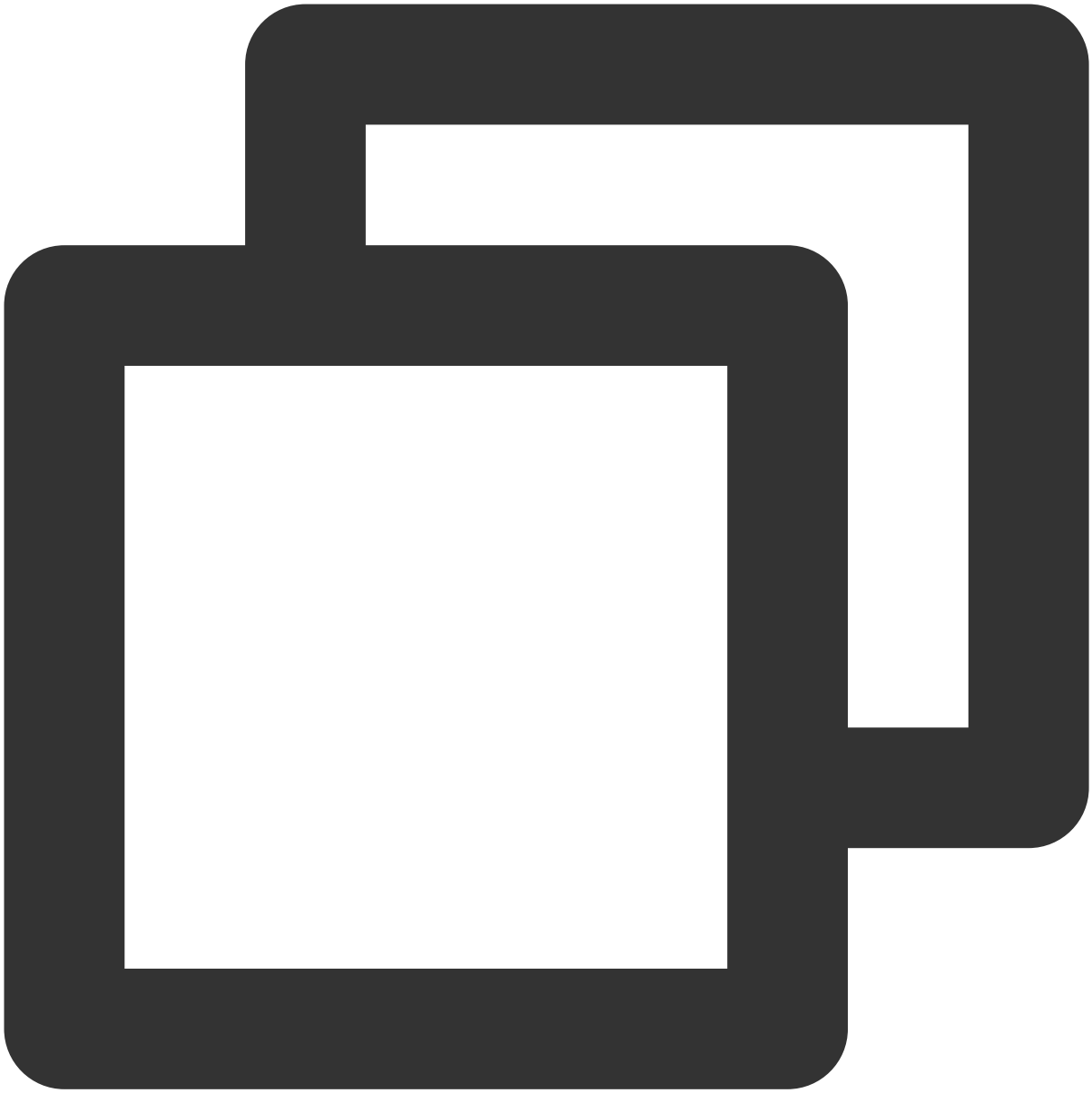
テストアカウントのログイン

このセグメントの詳細なドキュメント

この時点で、最初にコンソール上で作成したテストアカウントを使用して、ログイン検証を完了することが可能です。

`TencentIMSDK.Login` メソッドを呼び出して、テストアカウントにログインします。

戻り値 `res.code` が0の場合、ログインは成功です。



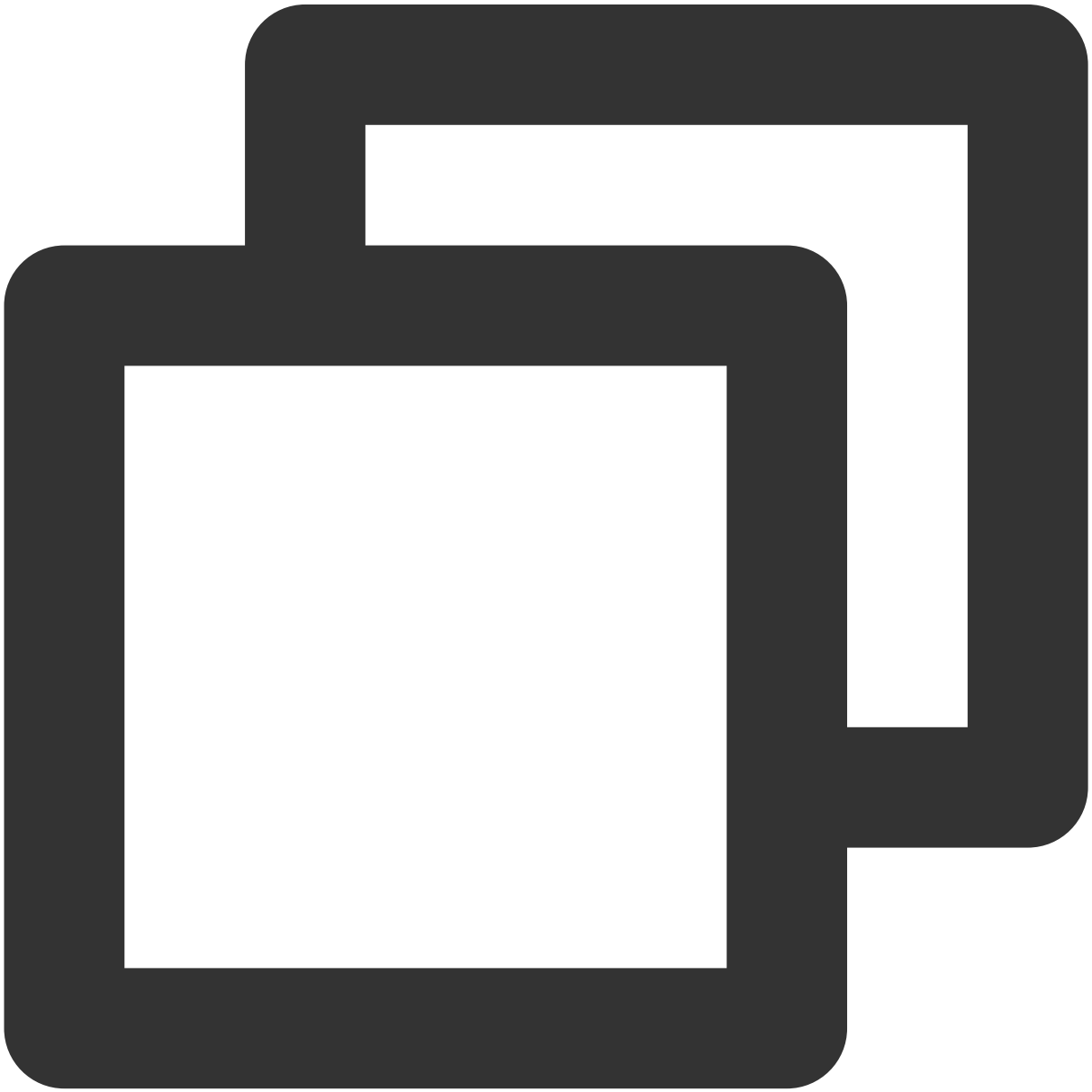
```
public static void Login() {
  if (userid == "" || user_sig == "")
  {
      return;
  }
  TIMResult res = TencentIMSDK.Login(userid, user_sig, (int code, string desc, stri
    // ログインコールバックロジックを処理します
  });
}
```

説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、 `UserSig` の計算コードをサーバーに統合し、Appのインターフェース向けに提供する方法となります。 `UserSig` が必要なときは、Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

メッセージの送信

このセグメントの詳細なドキュメント
ここでテキストメッセージの送信を例に取ります
サンプルコード：



```
public static void MsgSendMessage() {
        string conv_id = ""; // c2cメッセージセッションIDはuserID、グループメッセージセッシ
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // グループメッセージはTIMConv
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Text,
              text_elem_content =  "これは通常のテキストメッセージです"
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // メッセージ送信非同期結果
        });
              // 同期に送信されたメッセージから返されたメッセージID messageId
}
```

説明：

送信に失敗した場合、sdkAppIDは知らない人から送信されたメッセージをサポートしていない可能性があります。コンソールで有効にしてからテストに使用されます。

このリンクをクリックして、フレンドリレーションシップチェーンのチェックを無効にしてください。

セッションリストの取得

このセグメントの詳細なドキュメント

前の手順でテストメッセージの送信が完了しましたので、別のテストアカウントでログインし、セッションリストを取得できるようになりました。

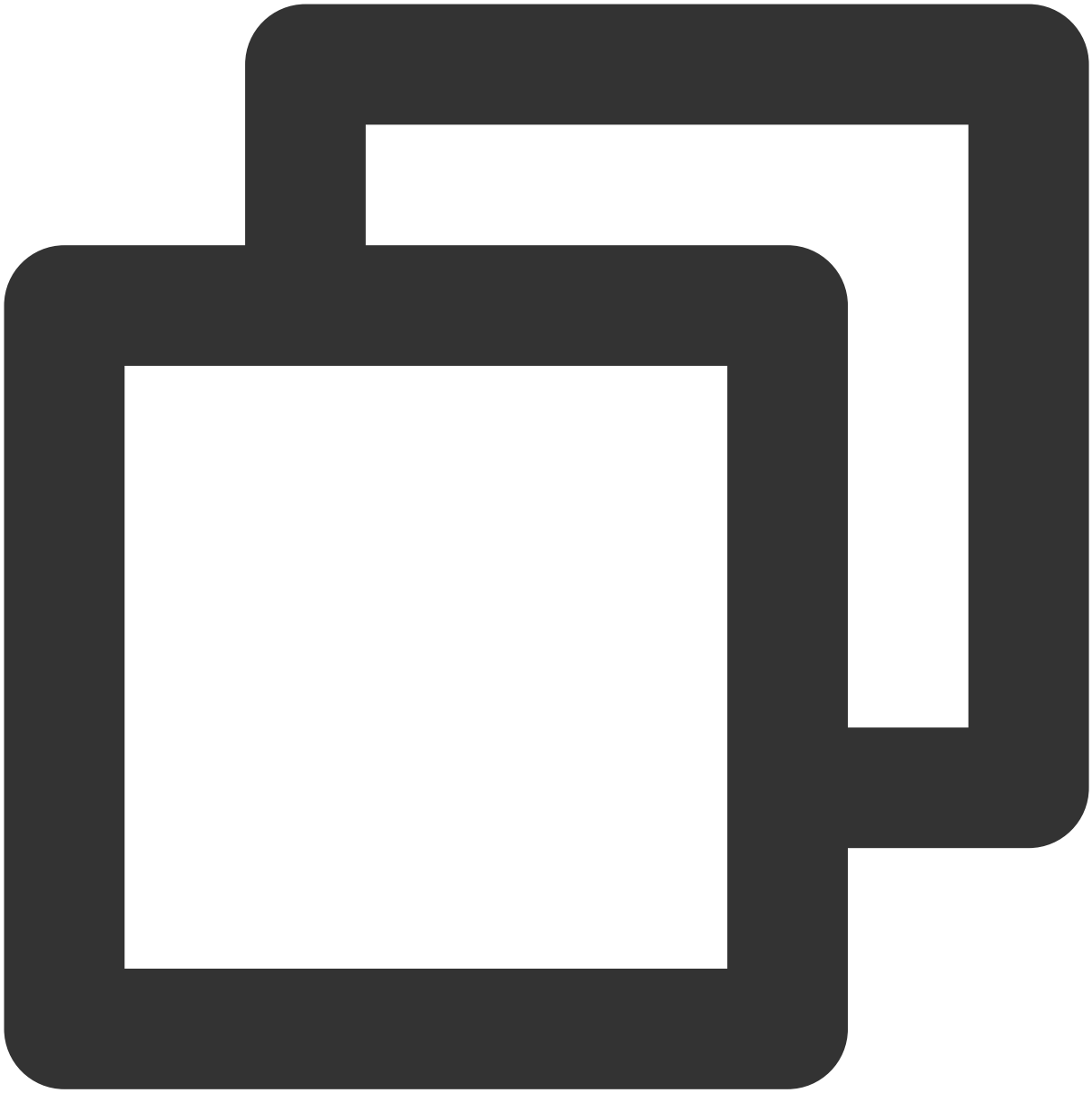
セッションリストの取得には2つの方法があります：

1. 長時間接続コールバックを監視し、セッションリストをリアルタイムに更新します。
2. APIをリクエストし、ページごとに一括でセッションリストを取得します。

一般的なユースケースは次のとおりです：

アプリケーションの起動後すぐにセッションリストを取得し、その後は長時間接続を監視して、セッションリストの変更をリアルタイムに更新します。

セッションリストの一括リクエスト



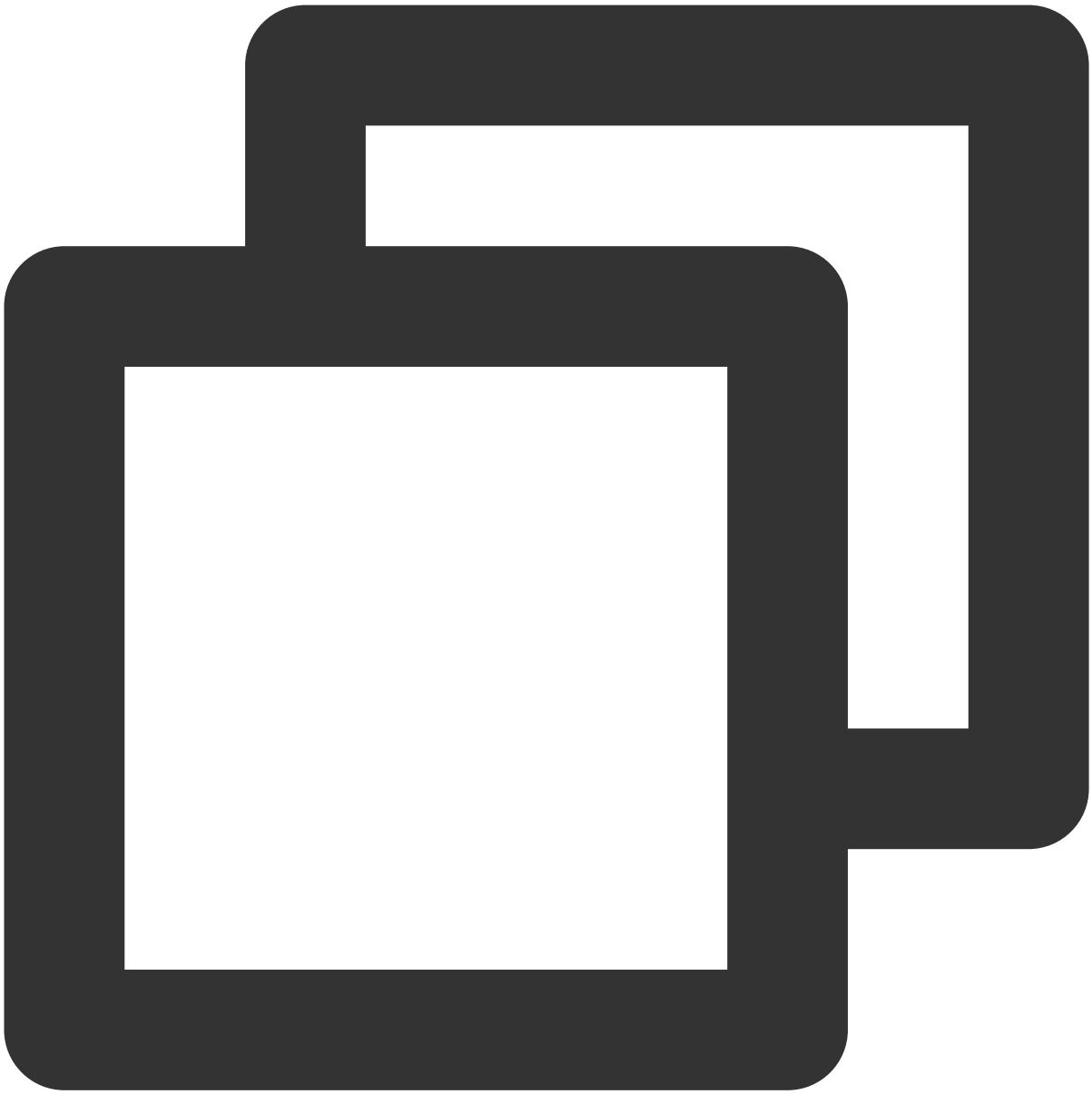
```
TIMResult res = TencentIMSDK.ConvGetConvList((int code, string desc, List<ConvInfo>
 // 非同期ロジックを処理します
});
```

この時点で、前の手順で別のテストアカウントを使用して送信したメッセージのセッションを見ることができるようになりました。

長時間接続のリッスンによるセッションリストのリアルタイム取得

この手順では、先にSDKにリッスンをマウントしてからコールバックイベントを処理し、UIを更新する必要があります。

1. 監視をマウントします。



```
TencentIMSDK.SetConvEventCallback((TIMConvEvent conv_event, List<ConvInfo> conv_lis
 // コールバックロジックを処理します
});
```

2. コールバックイベントを処理し、最新のセッションリストをインターフェース上に表示します。

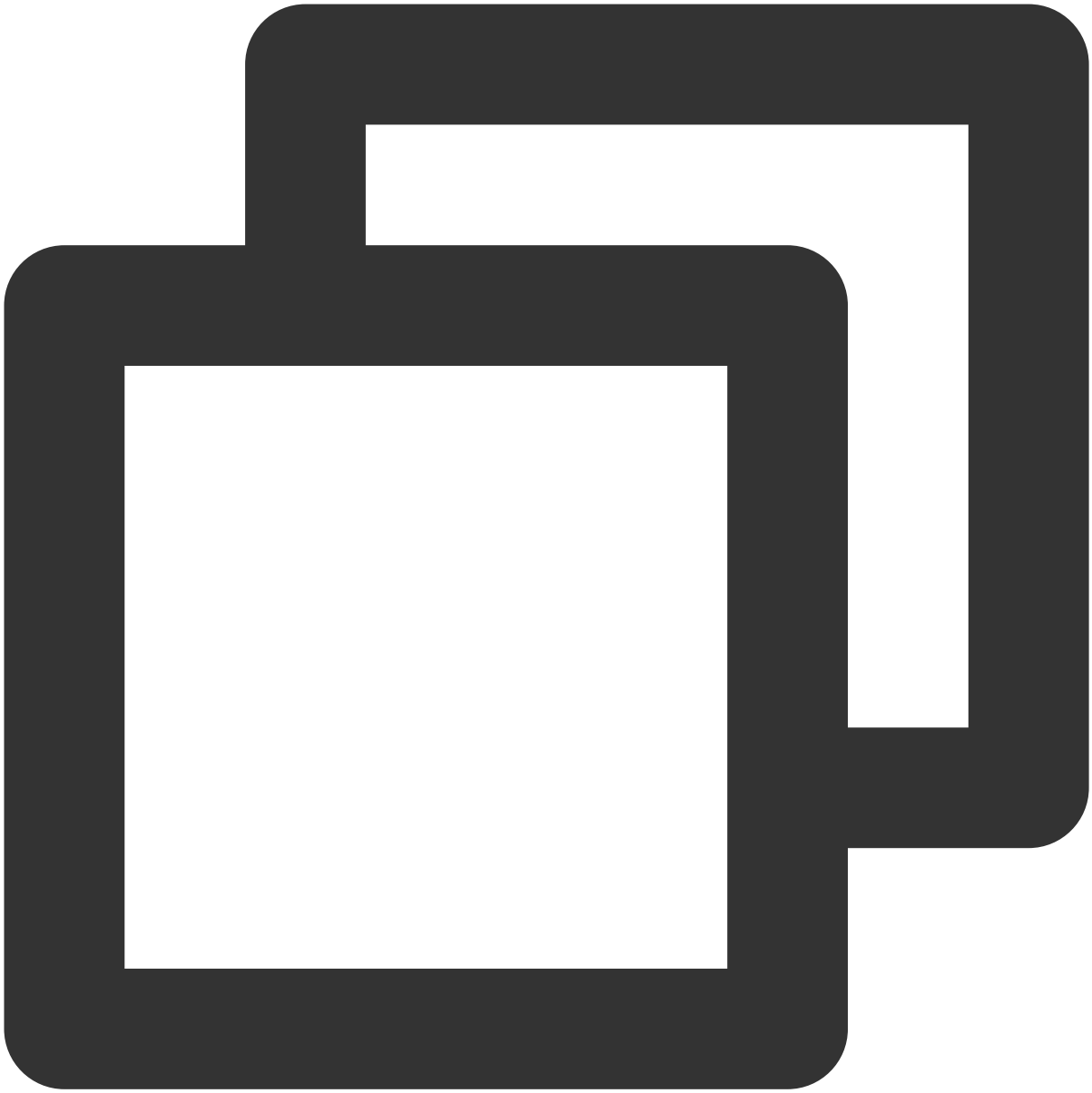
メッセージの受信

このセグメントの詳細なドキュメント

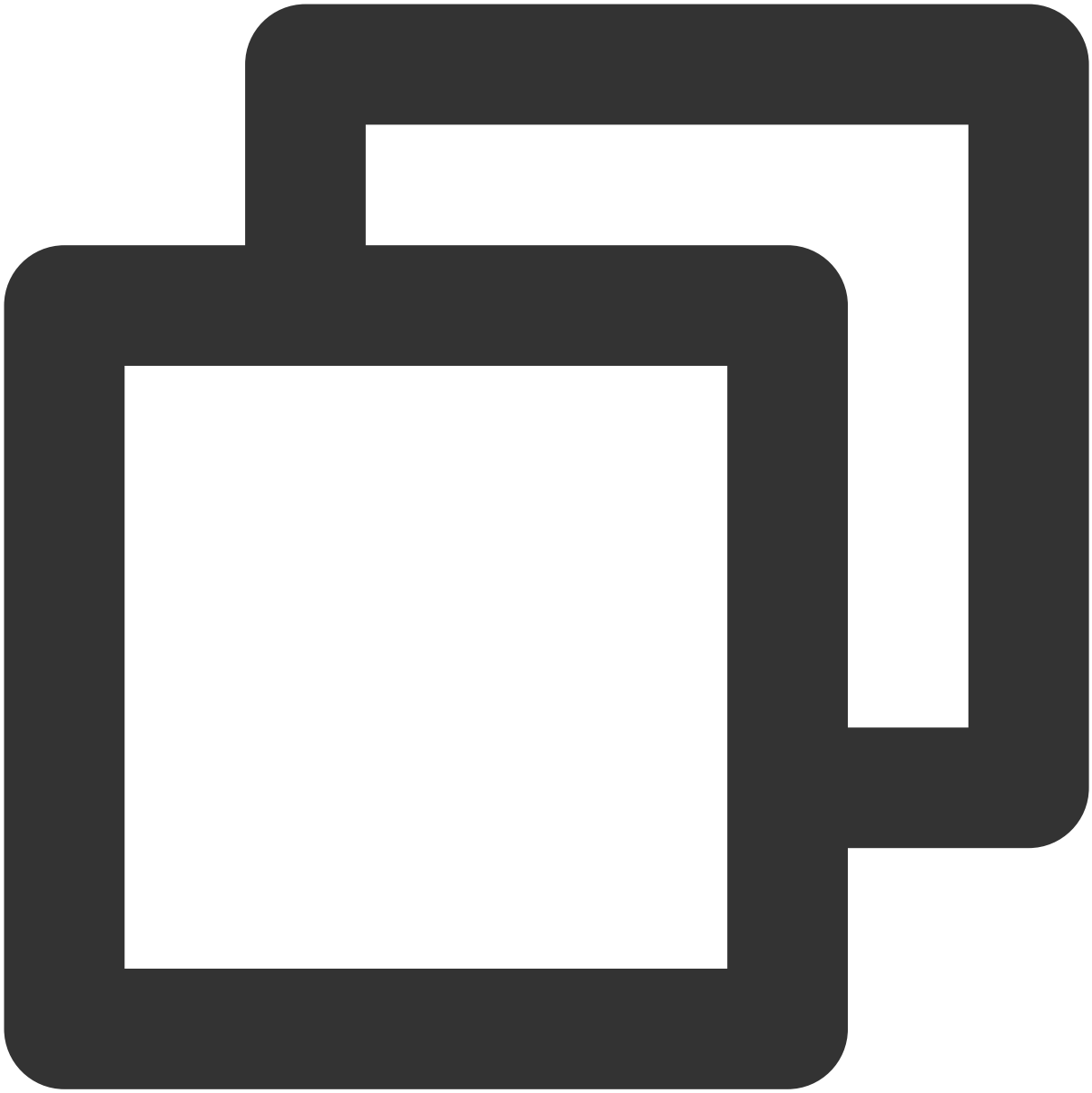
Tencent Cloud IM SDKによるメッセージの受信には2つの方法があります：
1. 長時間接続コールバックを監視し、メッセージの変更をリアルタイムに取得し、メッセージ履歴リストを更新してレンダリングします。
2. APIをリクエストし、ページごとに一括でメッセージ履歴を取得します。
一般的なユースケースは次のとおりです：
1. インターフェースが新しいセッションに入ると、まず一定量のメッセージ履歴を一括でリクエストし、メッセージ履歴リストの表示に用います。
2. 長時間接続を監視し、新しいメッセージをリアルタイムに受信してメッセージ履歴リストに追加します。

メッセージ履歴リストの一括リクエスト

ページごとに取得するメッセージ数が多すぎると取得速度に影響しますので、多すぎてはなりません。20通前後に設定することをお勧めします。
次のリクエストの際に使用できるよう、現在のページ数を動的に記録しなければなりません。
サンプルコードは次のとおりです：



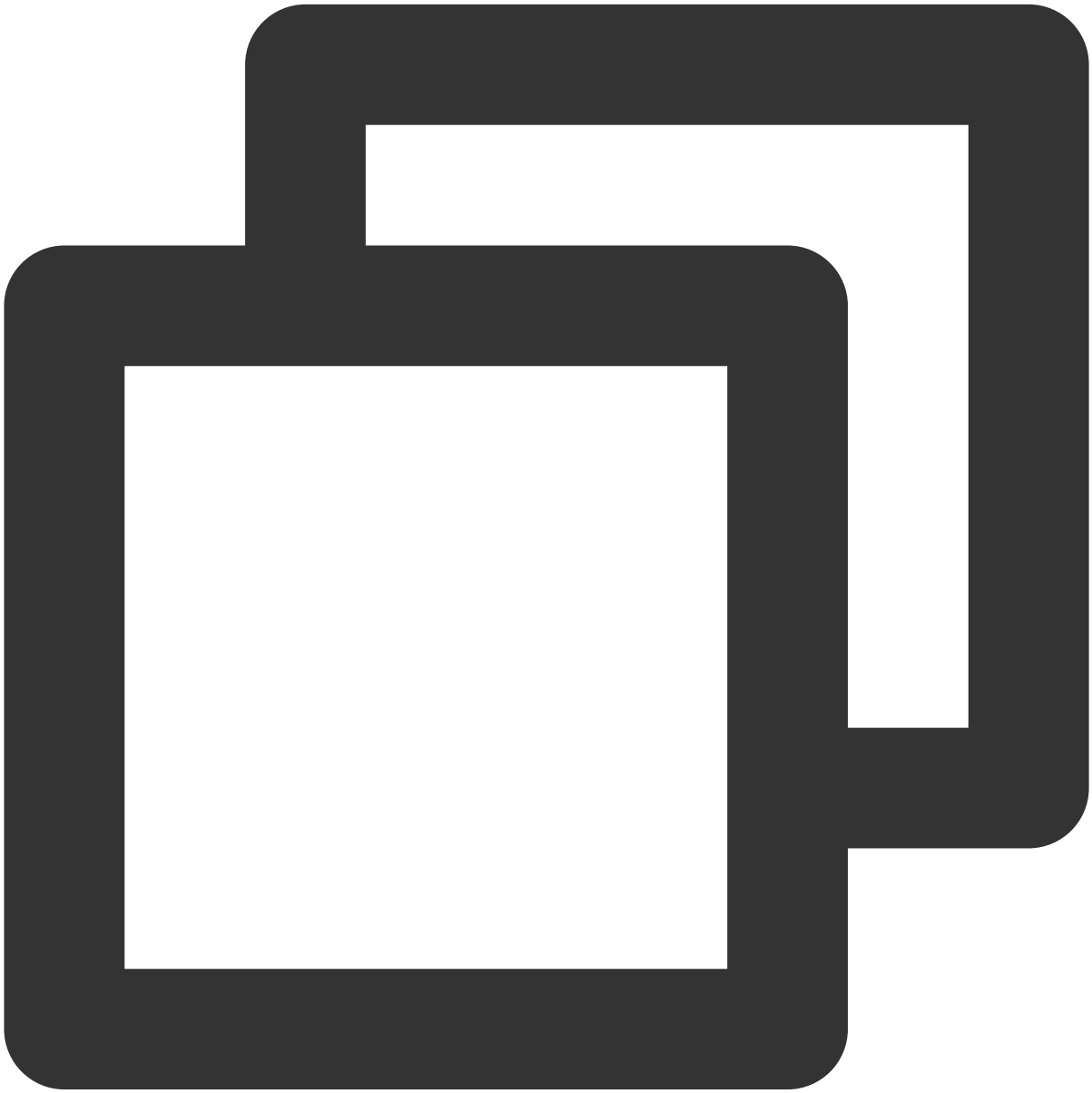
```
// シングルチャットメッセージ履歴の取得
// 初めて取得する場合は、msg_getmsglist_param_last_msgをnullに設定します
// 再度取得する場合は、msg_getmsglist_param_last_msgは、返されたメッセージリストの最後のメッセ
var get_message_list_param = new MsgGetMsgListParam
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
  // コールバックロジックを処理します
});
```



```
// シングルチャットメッセージ履歴を取得します
// 初めて取得する場合は、msg_getmsglist_param_last_msgをnullに設定します
// 再度取得する場合は、msg_getmsglist_param_last_msgは、返されたメッセージリストの最後のメッセ
var get_message_list_param = new MsgGetMsgListParam
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_Group, get
  // コールバックロジックを処理します
});
```

長時間接続の監視による新メッセージのリアルタイム取得

メッセージ履歴リストを初期化すると、新メッセージは `TencentIMSDK.AddRecvNewMsgCallback` 長時間接続から送信されます。
`AddRecvNewMsgCallback` コールバックがトリガーされると、必要に応じて、メッセージ履歴リストに新しいメッセージを追加できます。
モニターをバインドするサンプルコードは次のとおりです



```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> message, string user_data) => {
  // 新メッセージを処理します
});
```

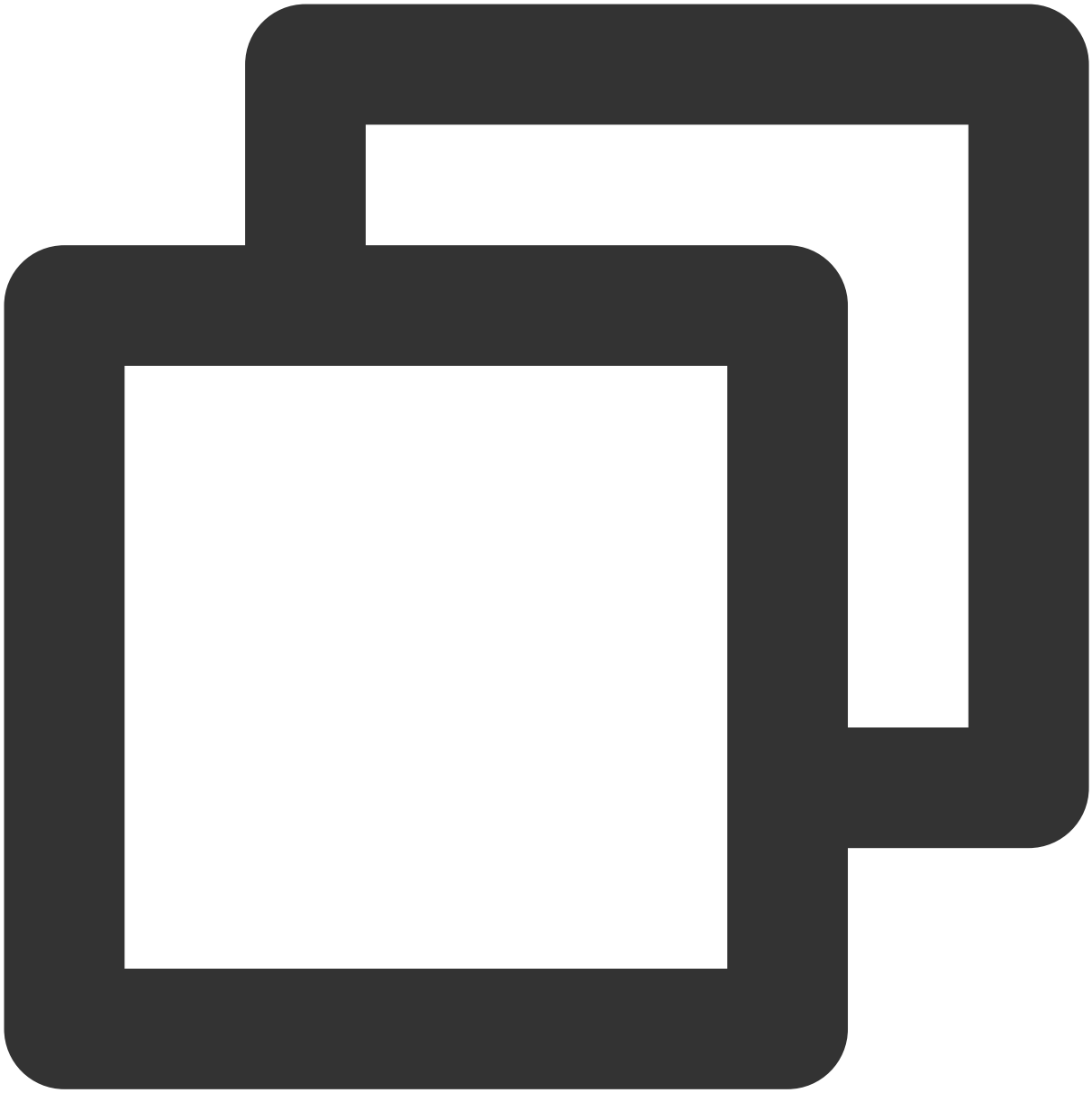
この時点で、IMモジュールの開発は基本的に完了し、メッセージの送受信や様々なセッションに入ることが可能になりました。
続いて、グループ、ユーザー個人情報、リレーションシップチェーン、ローカル検索などの関連機能の開発を完了することができます。
詳細については、SDKドキュメントのUI統合の自己実装をご確認ください。

その5：#Unity for WebGLのへのサポート

Tencent Cloud IM SDK (Unityバージョン)は、バージョン `1.8.1` からWebGLの構築をサポートします。
AndroidとiOS端末と比べると、いくつかの追加手順が必要です。次のとおりです：

JSの導入インポート

GitHubから次の3つのJSファイルをダウンロードし、プロジェクトビルドWebGL製品のフォルダーに配置します。
tim-js
tim-js-friendship.js
このファイルの名前をtim-upload-plugin.jsに変更 `index.html` を開き、この3つのJSファイルを導入します。次のとおりです：



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

よくあるご質問

サポートされているプラットフォームはどれですか？

現時点ではiOS、Android、Windows MacおよびWebGLをサポートしています。

AndroidでBuild And Runをクリックすると、使用できるデバイスが見つからないというエラーが発生します。

デバイスが他のリソースに使用されないことを保証するか、またはBuildをクリックしてapkパッケージを生成してから、エミュレーター内にドラッグして実行します。

iOSで初回実行時にエラーが発生します。

上のDemoで設定を実行した後もエラーが発生する場合、**Product>Clean**をクリックし、プロダクトを削除してから改めてBuildするか、またはXcodeをオフにして改めて開いて再度Buildします。

2019.04バージョンのUnity、iOSプラットフォームでエラーが発生します。

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'
Editorツールバーで**Window>Package Manager**をクリックし、Unity Collaborateを1.2.16にダウングレードします。

2019.04バージョンのUnity、iOSプラットフォームでエラーが発生します。

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
ソースコードを開き、`|| VersionControlSettings.mode != "Visible Meta Files"` の部分のコードを削除すれば完了です。

エラーコードのクエリー方法

IM SDKのAPIレベルのエラーコードについては、このドキュメントをご確認ください。

クイックスタート（UE）

最終更新日：：2024-04-11 16:20:42

ここでは、主にTencent Cloud IM Demo（Unreal Engine）を素早く実行する方法について説明します。

説明：

現時点ではWindows、macOS、iOS、Androidをサポートしています。

環境要件

Unreal Engine 4.27.1およびそれ以降のバージョンを推奨します。

開発端末	環境
Android	Android Studio 4.0およびそれ以降のバージョン。 Visual Studio 2017 15.6およびそれ以降のバージョン。 実機デバッグのみサポートしています。
iOS & macOS	Xcode 11.0およびそれ以降のバージョン。 OSXシステムバージョンは10.11およびそれ以降のバージョン。 プロジェクトに有効な開発者署名を設定済みであることを確認してください。
Windows	OS：Windows 7 SP1およびそれ以降のバージョン（x86-64ベースの64ビットOS）。 ディスク容量：IDEおよびいくつかのツールのインストールの他、少なくとも1.64GBの空きがある必要があります。 Visual Studio 2019のインストール。

前提条件

Tencent Cloudアカウントの登録を行い、実名認証が完了済みであること。

操作手順

手順1：アプリケーションの新規作成

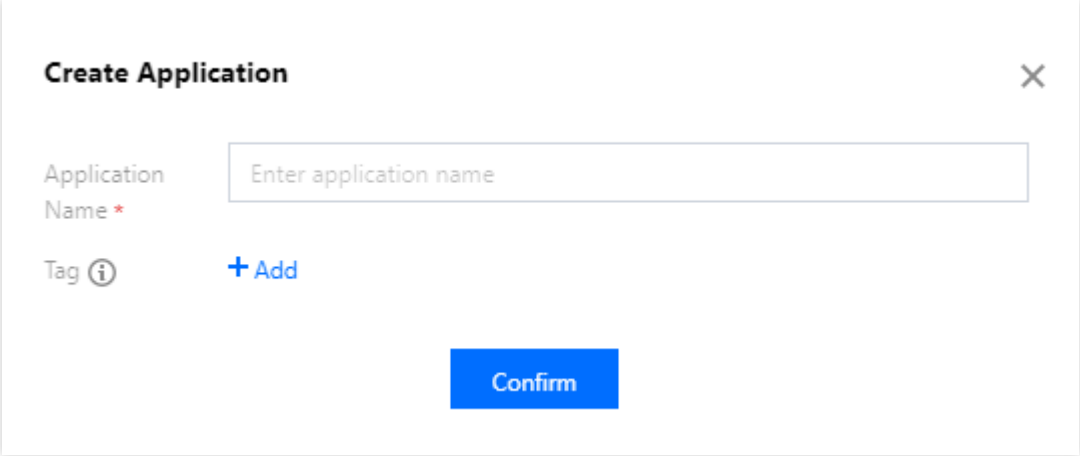
1. IMコンソールにログインします。

説明：

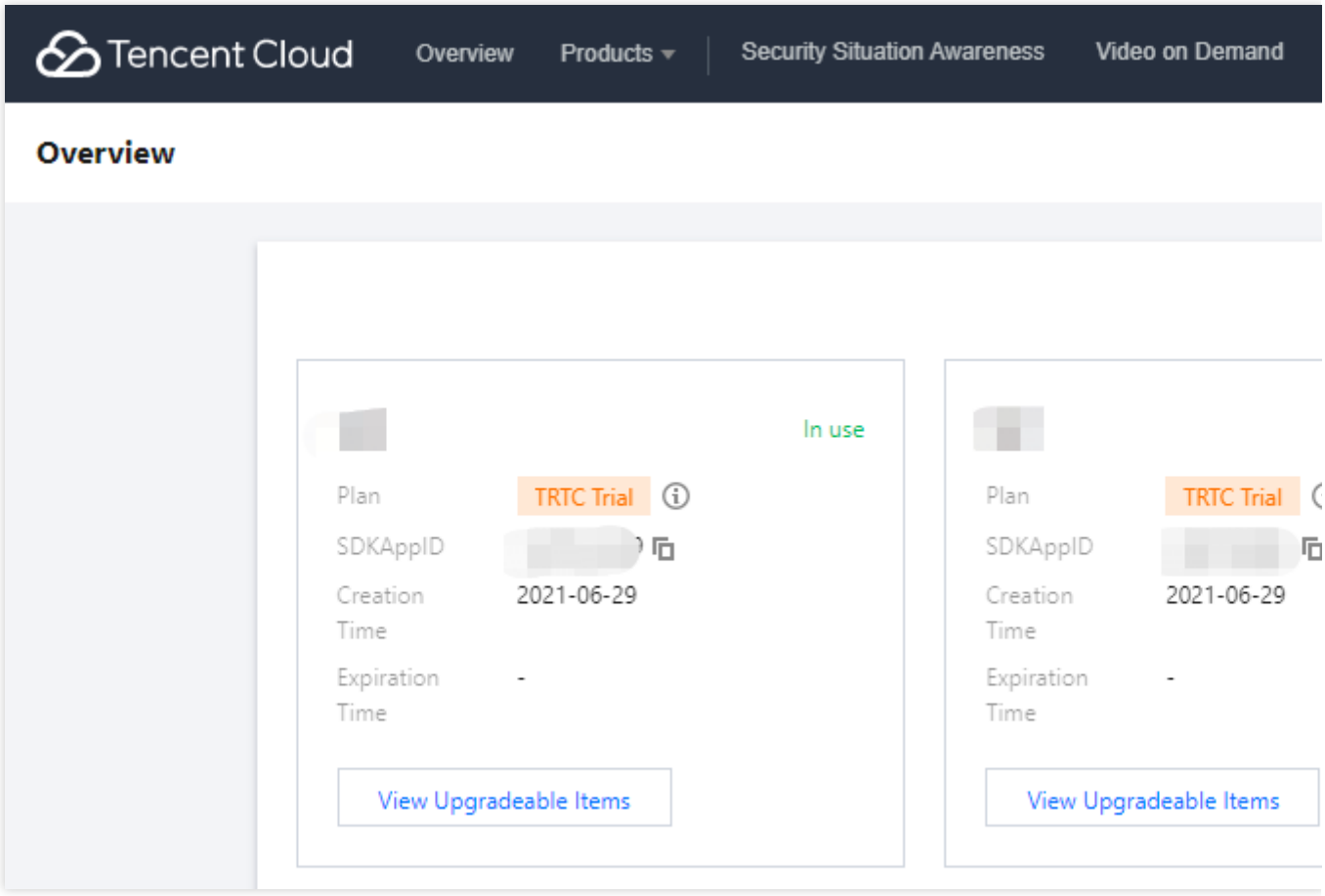
アプリケーションをすでに保有している場合は、そのSDKAppIDを記録してからキー情報の取得を実行してください。

同じTencent Cloudのアカウントで、最大300個のIMアプリケーションを作成することができます。すでにアプリケーションが300個ある場合は、使用する必要のないアプリケーションを使用停止して削除すると、新しいアプリケーションを作成することができます。**アプリケーションを削除した後、そのSDKAppIDに対応するすべてのデータとサービスは失われます。慎重に操作を行ってください。**

2. **新しいアプリケーションの作成**をクリックし、**アプリケーションの作成**のダイアログボックスにアプリケーション名を入力し、**OK**をクリックします。

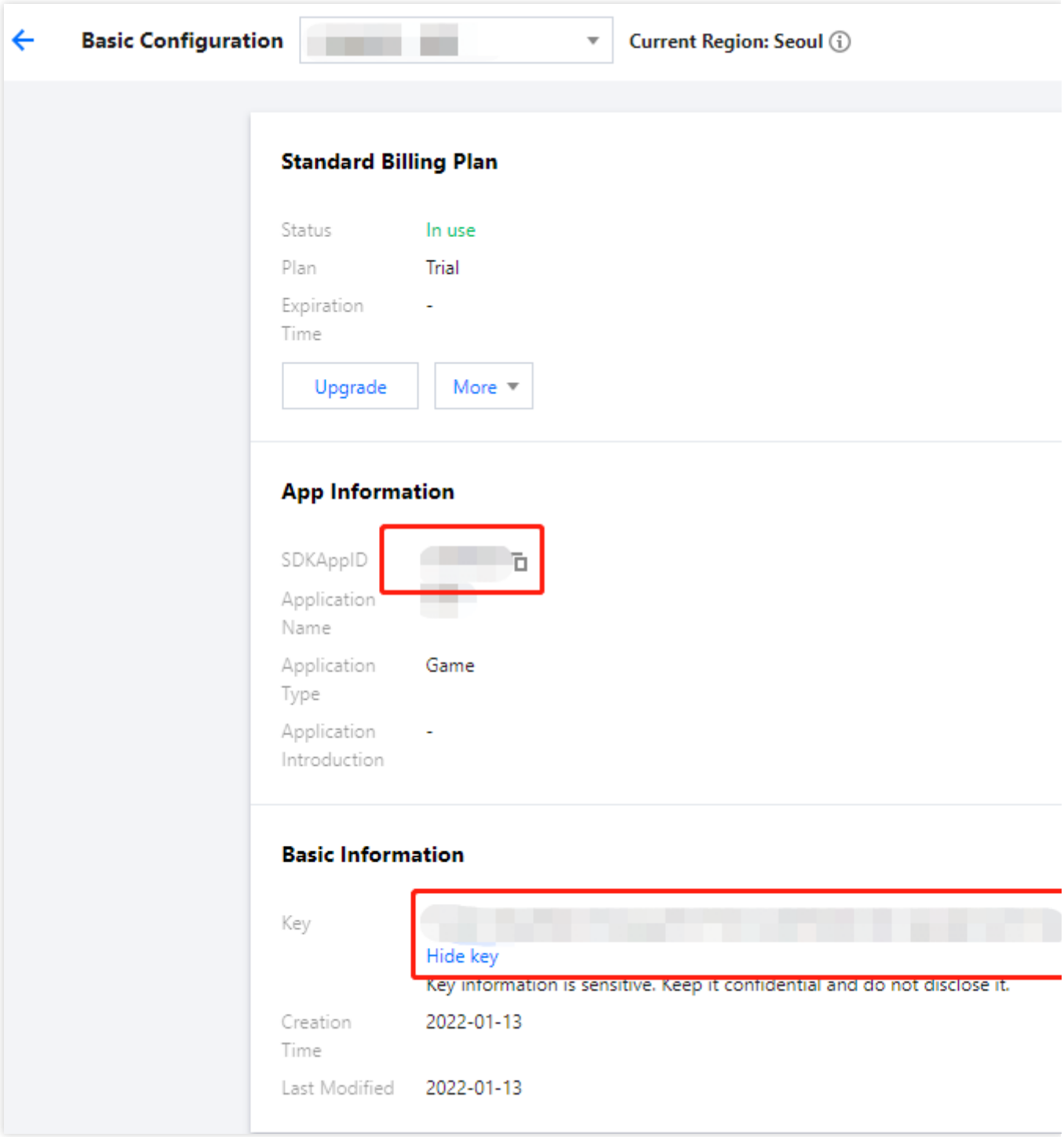


3. 作成が完了すると、コンソールの概要ページで、作成したアプリケーションのステータス、サービスバージョン、SDKAppID、作成時間、タグおよび有効期限を確認できます。SDKAppID情報を記録してください。



ステップ2：キー情報の取得

1. 対象のアプリケーションカードをクリックし、アプリケーションの基本設定画面に移動します。



2. **基本情報**セクションで、**表示キー**をクリックし、キー情報をコピーして保存します。

ご注意：

キー情報を適切に保管して、漏えいしないようにしてください。

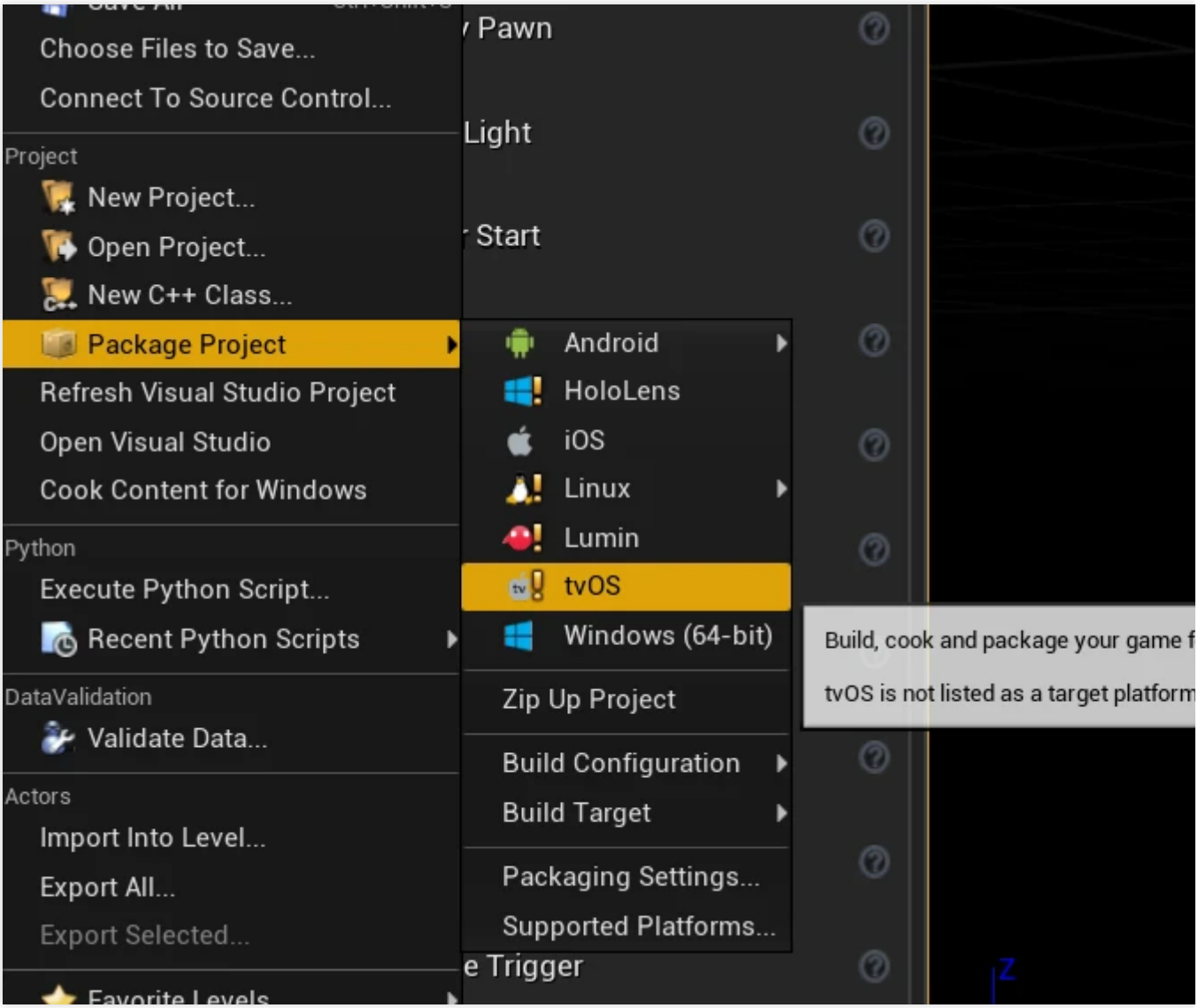
ステップ3：Demo プログラムファイルの設定

1. IM Demoプログラムをダウンロードします。ダウンロードアドレスはDemoダウンロードをご参照ください（ご不明な点がございましたら、QQグループにご参加の上、764231117にお問い合わせください）。
2. `/IM_Demo/Source/debug/include/DebugDefs.h` ファイルを見つけて開きます。

3. `DebugDefs.h` のファイルの関連パラメータを設定します。
SDKAPPID：デフォルトは0。実際のSDKAppIDを設定してください。
SECRETKEY：デフォルトは""。実際のキー情報を設定してください。
説明：
ここで言及したUserSigの新規作成ソリューションでは、クライアントコードでSECRETKEYを設定します。この手法のうちSECRETKEYは逆コンパイルによって逆向きにクラッキングされやすく、キーがいったん漏洩すると、攻撃者はTencent Cloudトラフィックを盗用できるようになります。そのため**この手法は、ローカルのDemoクイックスタートおよび機能デバッグにのみ適合します**。
正しい UserSigの発行方法は、 UserSig の計算コードをお客様のサーバーに統合して、App向けのインタフェースを用意し、UserSig を必要とするときは、App から業務サーバーにリクエストを出して、ダイナミックUserSigを取得することです。より詳細な内容については、 サーバーでのUserSig新規作成をご参照ください。

手順4：コンパイルとパッケージ化の実行

1. `/IM_Demo/IM_Demo.uproject` をダブルクリックして開いてください。
2. コンパイルを実行し、デバッグを行います。
macOS 端末
Windows 端末
iOS 端末
Android 端末
File -> Package Project -> Mac
File->Package Project->Windows->Windows(64-bit)



プロジェクトをパッケージ化します

File -> Package Project-> iOS

1. 開発とデバッグ：Androidクイックスタートをご参照ください。
2. プロジェクトのパッケージ化：Androidプロジェクトのパッケージ化をご参照ください。

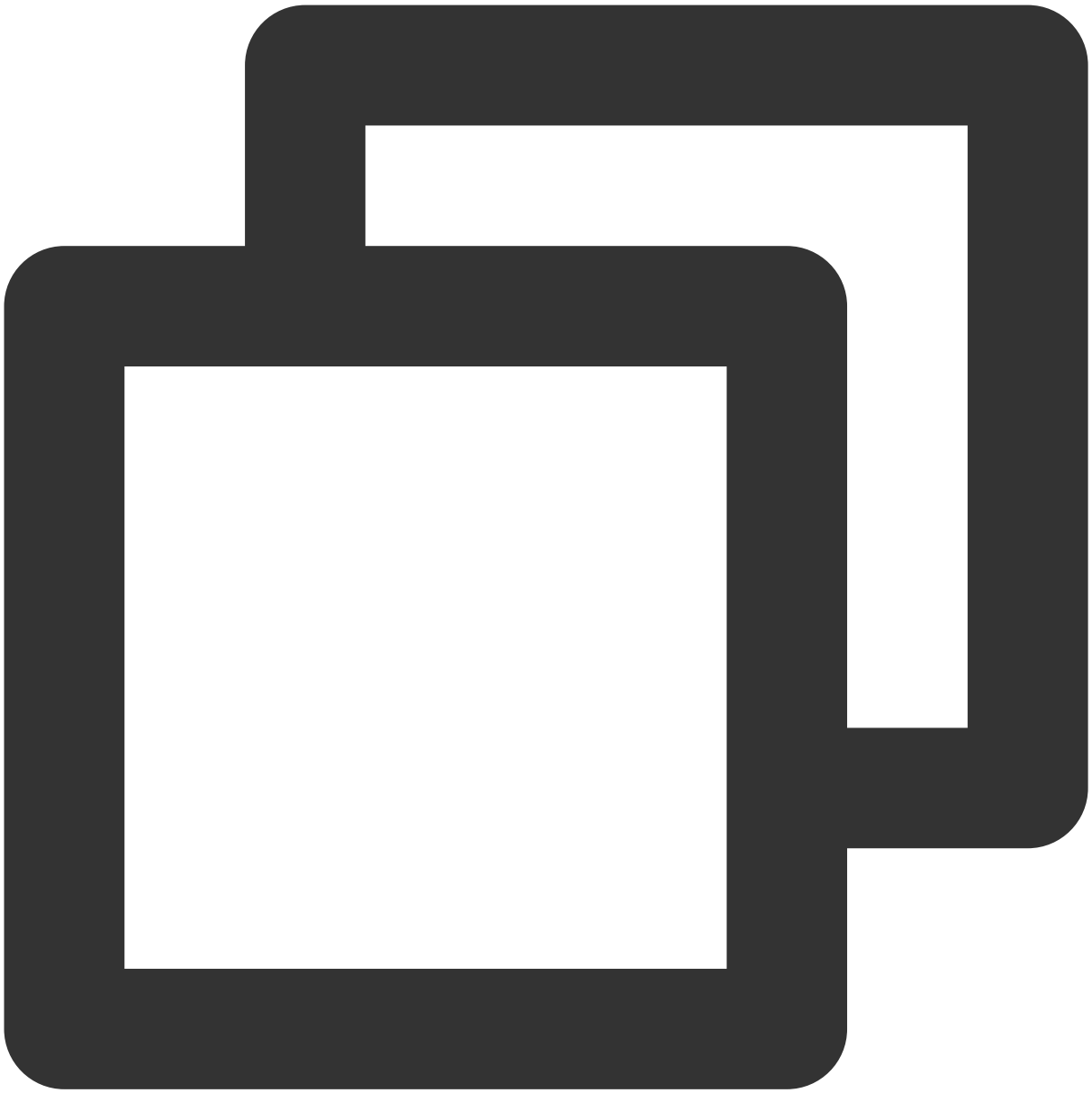
IM Unreal Engine APIドキュメント

その他のインターフェースについては、API概要をご参照ください。

よくあるご質問

Android“Attempt to construct staged filesystem reference from absolute path""”エラー

UE4プロジェクトを閉じて、CMDを開き、次のコマンドを実行します。



```
adb shell

cd sdcard

ls (you should see the UE4Game directory listed)

rm -r UE4Game
```

プロジェクトを再コンパイルします

クイックスタート（Flutter）

最終更新日：：2024-04-11 16:20:42

ここでは、Flutter SDKの統合方法についてご説明します。

環境要件

環境	バージョン
Flutter	IM SDKの最低要件はFlutter 2.2.0バージョン、TUIKit統合コンポーネントリポジトリの最低要件はFlutter 2.10.0バージョンです。
Android	Android Studio 3.5以降のバージョン。Appの要件はAndroid 4.1以降のバージョンのデバイスです。
iOS	Xcode 11.0以降のバージョン。プロジェクトに有効な開発者署名を設定済みであることを確認してください。

サポートするプラットフォーム

Flutterのすべてのプラットフォームをサポートする IM SDKとTUIKitの構築に取り組み、1つのコードセットですべてのプラットフォームで実行することを支援します。

プラットフォーム	UI SDK (tencent_cloud_chat_sdk) なし	UIおよび基本業務ロジックTUIKit (tencent_cloud_chat_uikit)を含めます
iOS	対応可	対応可
Android	対応可	対応可
Web	対応可、4.1.1+2以降のバージョンは対応可	対応可、0.1.5とそれ以降のバージョンは対応可
macOS	対応可、4.1.9以降のバージョンは対応可	近日リリース
Windows	対応可、4.1.9以降のバージョンは対応可	近日リリース
混合開発（Flutter SDKを既存のネイティブアプリケーションに追加しま	5.0.0以降のバージョンは対応可	1.0.0とそれ以降のバージョンは対応可

す）		

説明：
Web/macOS/Windowsプラットフォームでは、わずかなステップで簡単に導入する必要があります。詳細については、この記事のその他のプラットフォームを展開をご参照ください。

Demo体験

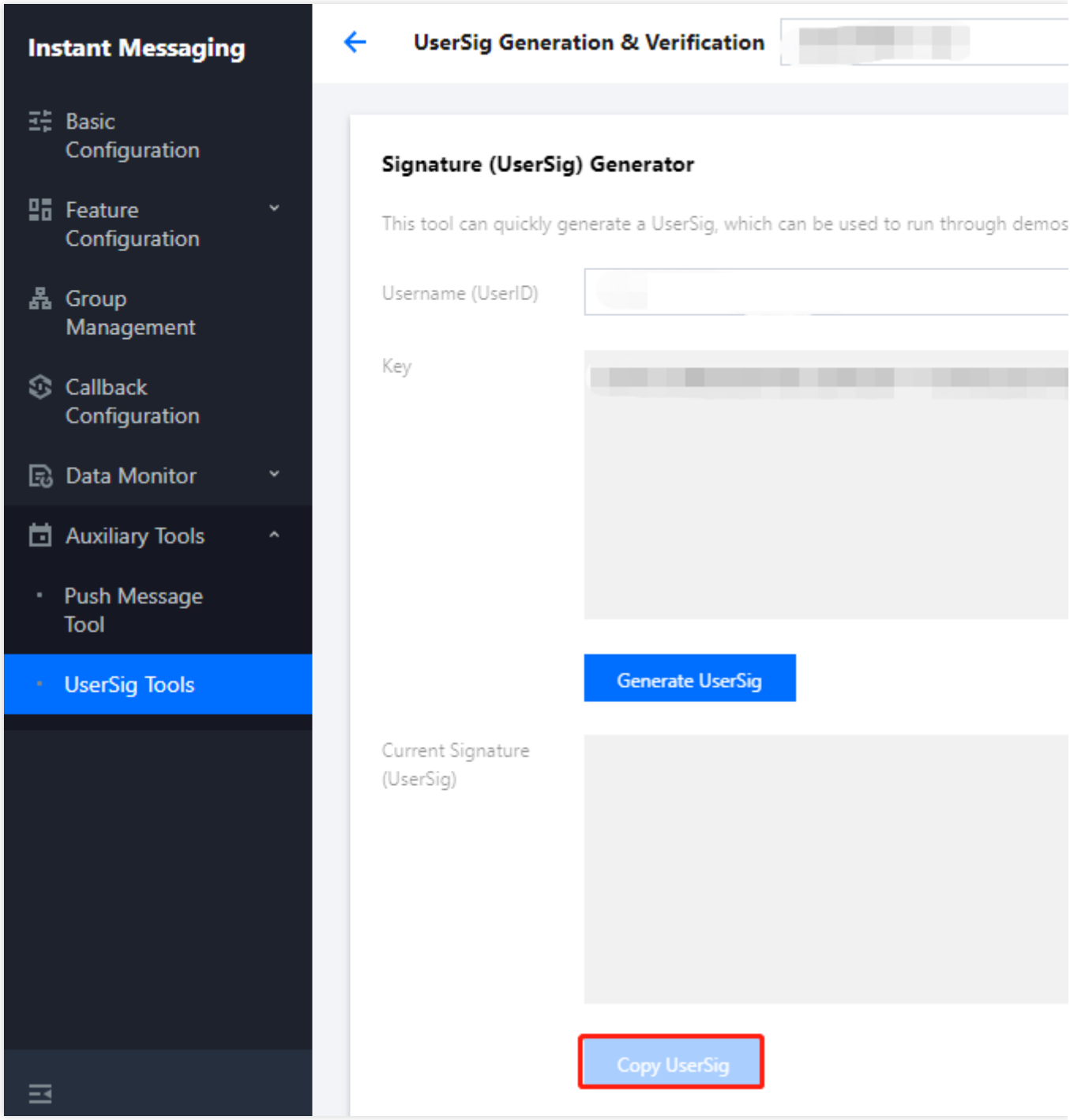
アクセス前に、DEMOを体験して、Tencent Cloud IM FlutterクロスプラットフォームSDKとTUIKit機能をすばやく理解できます。
以下の各バージョンのDEMOは、すべてTUIKitを同じFlutterプロジェクトに導入して制作・パッケージ化しています。Desktop(macOS/Windows)プラットフォームは、SDKでサポートされています。DEMOは近日リリースされます。

モバイル端末 APP	WEB - H5
iOS/Android APP、プラットフォームのダウンロードを自動的に判断します	携帯電話でコードをスキャンしてオンラインWeb版DEMOを体験します

予備作業

1. Tencent Cloudアカウントの登録を行い、実名認証が完了していること。
2. アプリケーションの作成とアップグレードを参照してアプリケーションを作成し、`SDKAppID`を記録していること。

3. IMコンソールでご自分のアプリケーションを選択し、左側ナビゲーションバーで**支援ツール->UserSig生成&検証**の順にクリックし、2つのUserIDおよびそれに対応するUserSigを作成します。`UserID`、署名`（Key）`、`UserSig`の3つをコピーし、その後のログインで使用します。



説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の`UserSig`の正しい発行方法は、サーバーで生成し、Appのインターフェース向けに提供する方法となります。`UserSig`が必要なときは、

Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

適切な方法を選択してFlutter SDKを統合

IMは3種類の統合方法を提供しており、最適な方法を選択して統合を行うことができます。

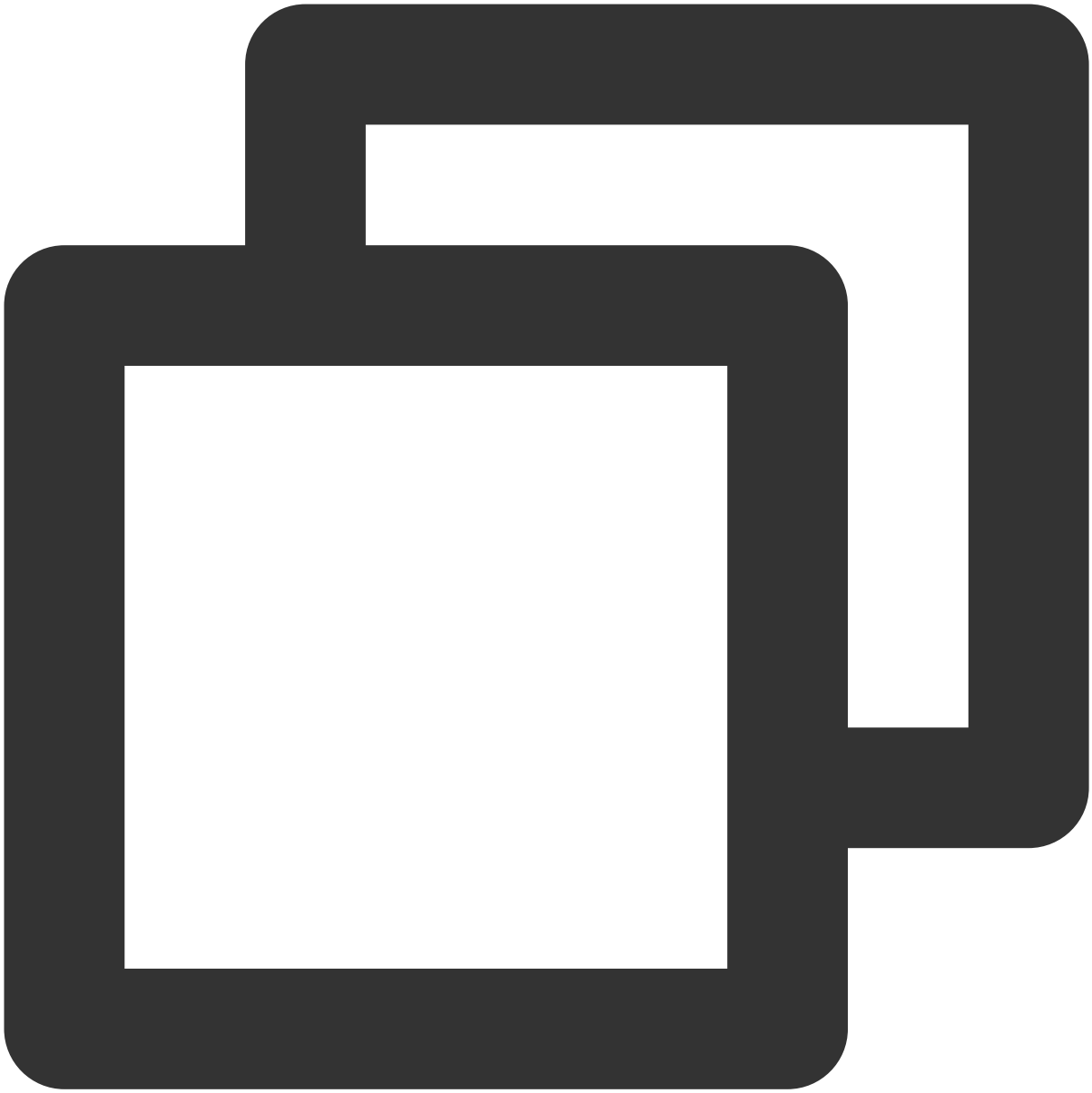
統合方法	適用シナリオ
DEMOによる修正	IM Demoは完全なチャットAppであり、コードはオープンソース化されています。チャットライクなユースケースを実装したい場合は、Demoを使用して二次開発を行うことができます。今すぐDemo体験が可能です。
UIを含む統合	IMのUIコンポーネントリポジトリ `TUIKit` は、セッションリスト、チャットインターフェース、連絡先リストなどの共通のUIコンポーネントを提供しています。開発者は実際の業務ニーズに応じて、このコンポーネントリポジトリによってカスタムIMアプリケーションをスピーディーにビルドすることができます。**この方法を優先して用いることを推奨します**。
UI統合を自身で実装	アプリケーションインターフェースのニーズがTUIKitでは満たせない場合、または多くのカスタマイズが必要な場合は、この方法を用いることができます。

IM SDKの各APIについてより深くご理解いただけるよう、各APIの呼び出しおよび監視のトリガーのデモンストレーションを行うAPI Exampleも提供しています。

ソリューション1：Demoによる修正

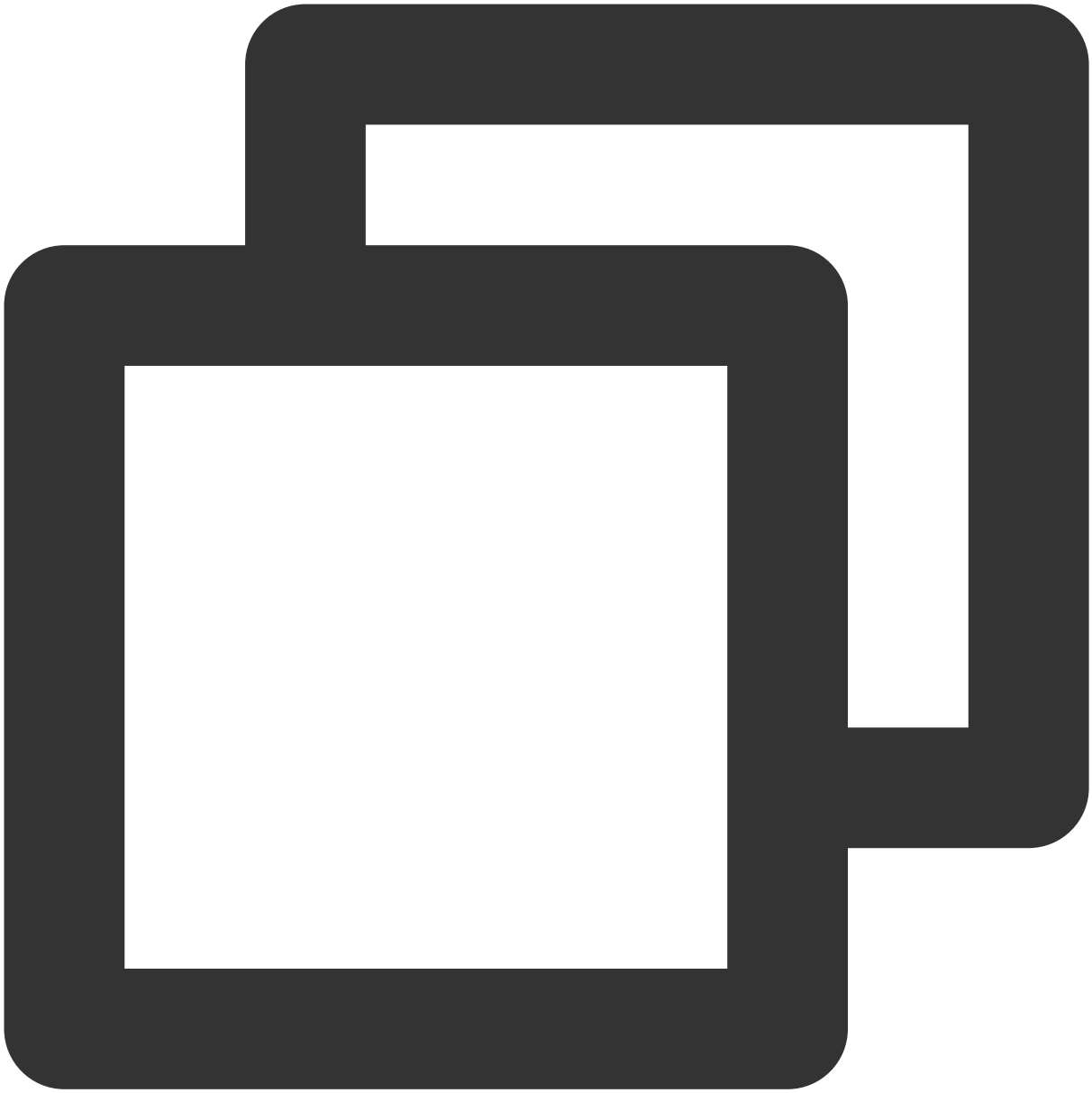
Demoクイックスタート

1. Demoソースコードのダウンロード、依存関係のインストール：



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

# Install dependencies
flutter pub get
```

2. Demoプロジェクトの実行：



```
#demoプロジェクトを起動します。SDK_APPID、KEYの2つのパラメータを置き換えてください
flutter run --dart-define=SDK_APPID={YOUR_SDKAPPID} --dart-define=ISPRODUCT_ENV=fal
```

説明：

`--dart-define=SDK_APPID={YOUR_SDKAPPID}` の中の `{YOUR_SDKAPPID}` をご自分のアプリケーションのSDKAppIDに置き換える必要があります。

`--dart-define=ISPRODUCT_ENV=false` は開発環境か本番環境かを判断します。開発環境の場合、falseとしてください。

`--dart-define=KEY={YOUR_KEY}` の中の `{YOUR_KEY}` を、その1：テストユーザーの作成の `キー（Key）` 情報に置き換える必要があります。

IDEを使用して実行することも可能です：（オプションの手順）

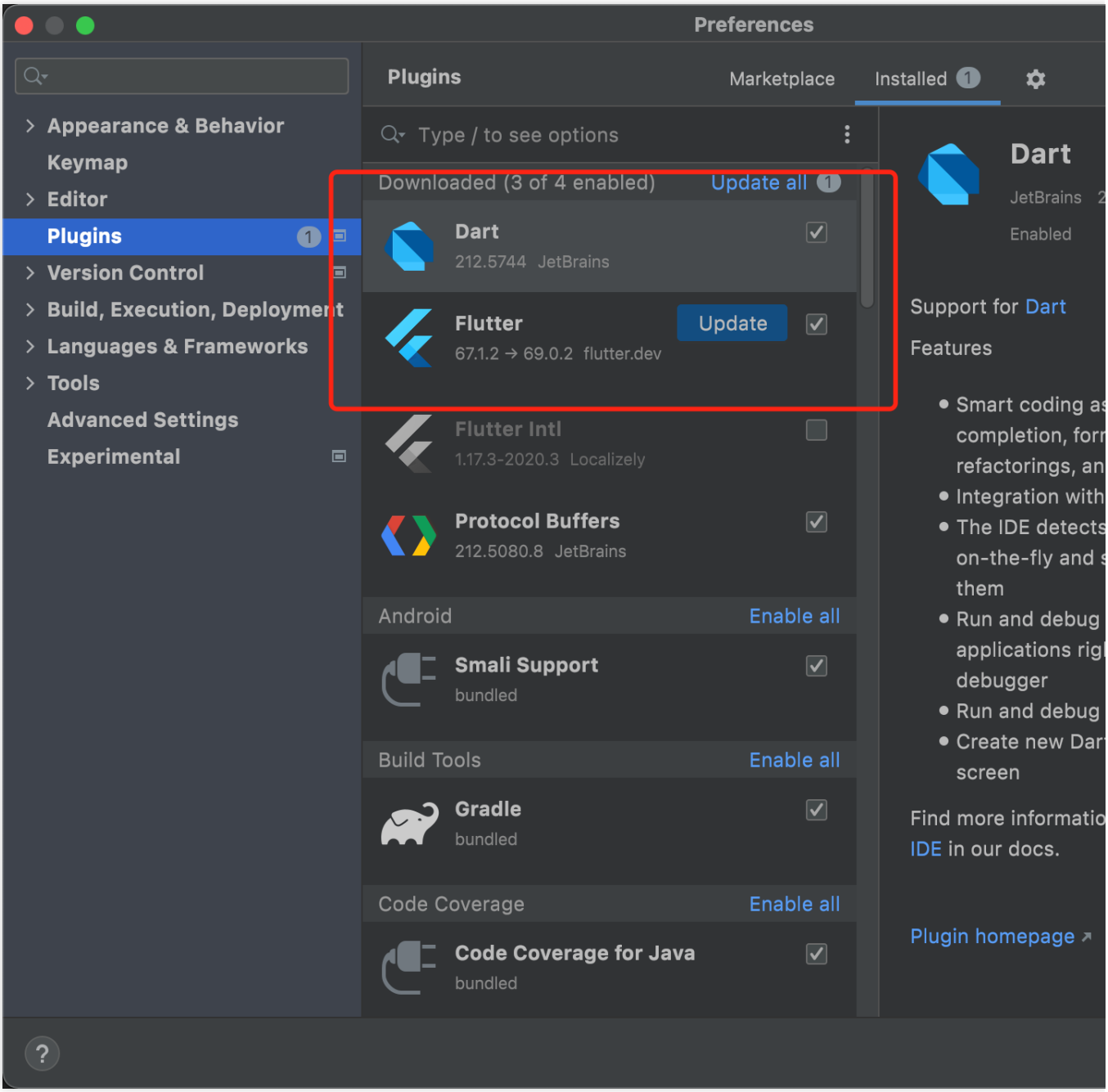
Androidプラットフォーム

iOSプラットフォーム

1. Android Studioでは、FlutterとDartプラグインをインストールします。

Macプラットフォーム：プラグイン設定を開く（v3.6.3.0以降のシステムでPreferences > Pluginsを開く） => Flutterプラグインを選択してインストールをクリックする => Dartプラグインをインストールするといったプロンプトが表示されると、Yesをクリックする => 再起動といったプロンプトが表示されると、Restartをクリックする。

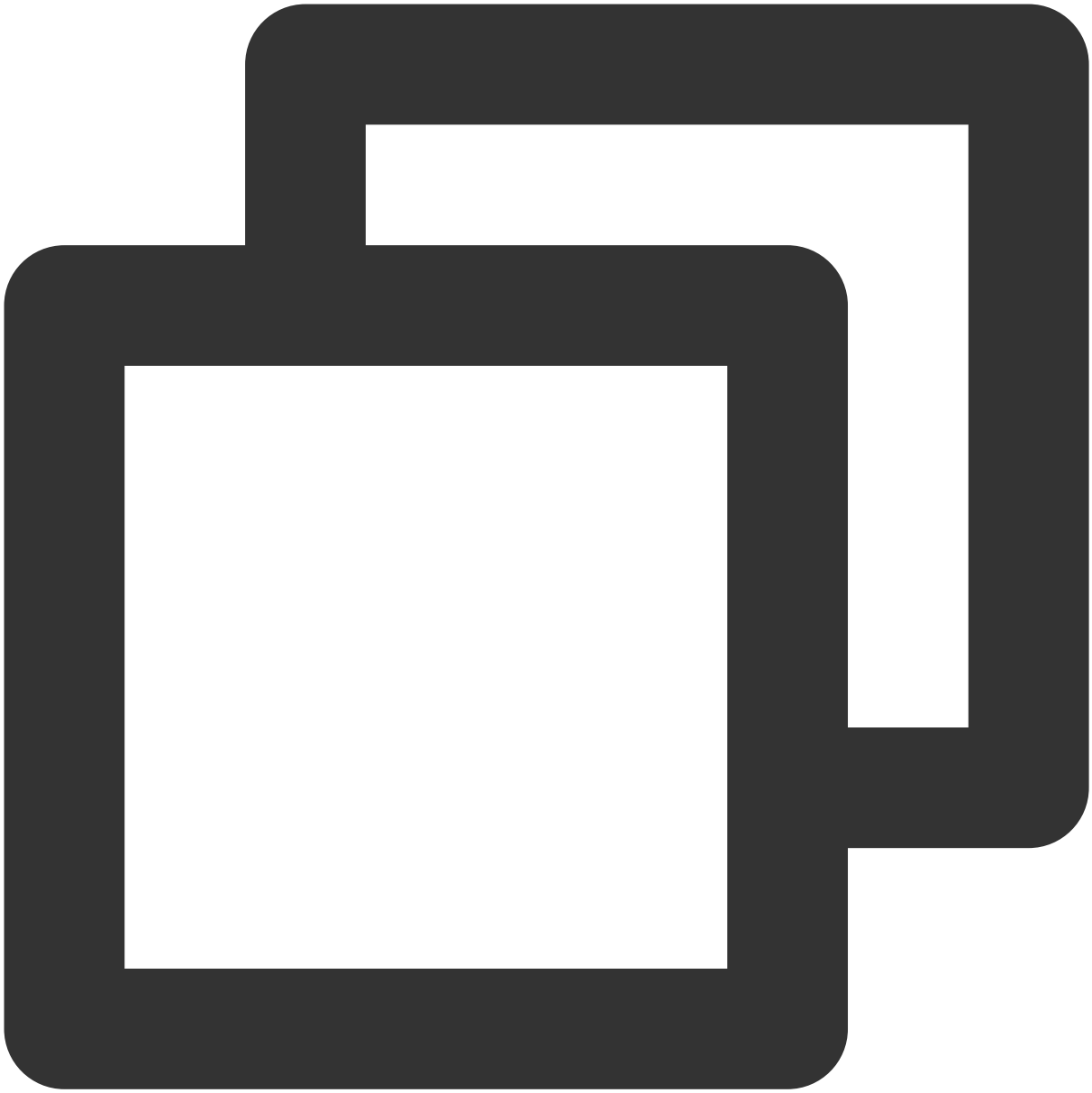
LinuxまたはWindowsプラットフォーム：プラグイン設定を開く（File > Settings > Pluginsにある） => Marketplace（拡張ストア）を選択し、Flutter pluginを選択してから、Install（インストール）をクリックする。



2. プロジェクトを開いて、依存関係を取得します。

Android Studioでは、 `im-flutter-uikit` ディレクトリを開きます。

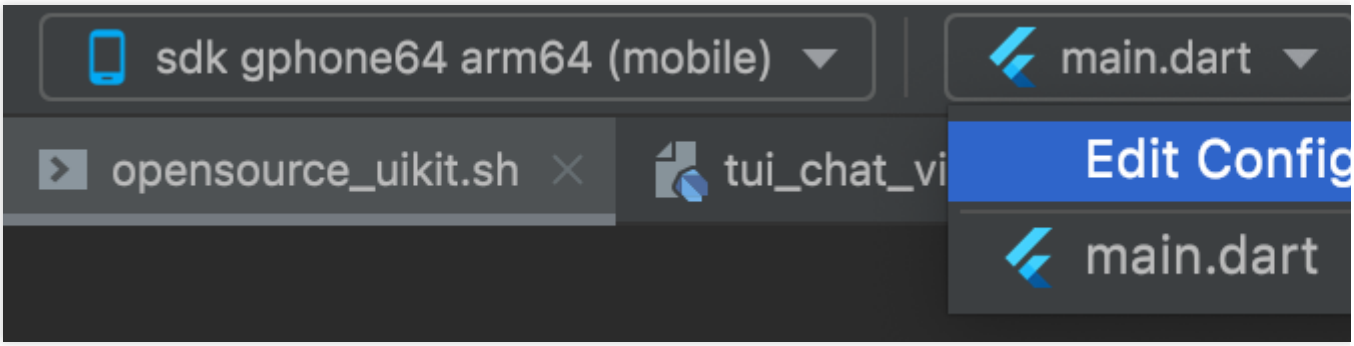
このパスでコマンドを実行して、依存関係を実行します。



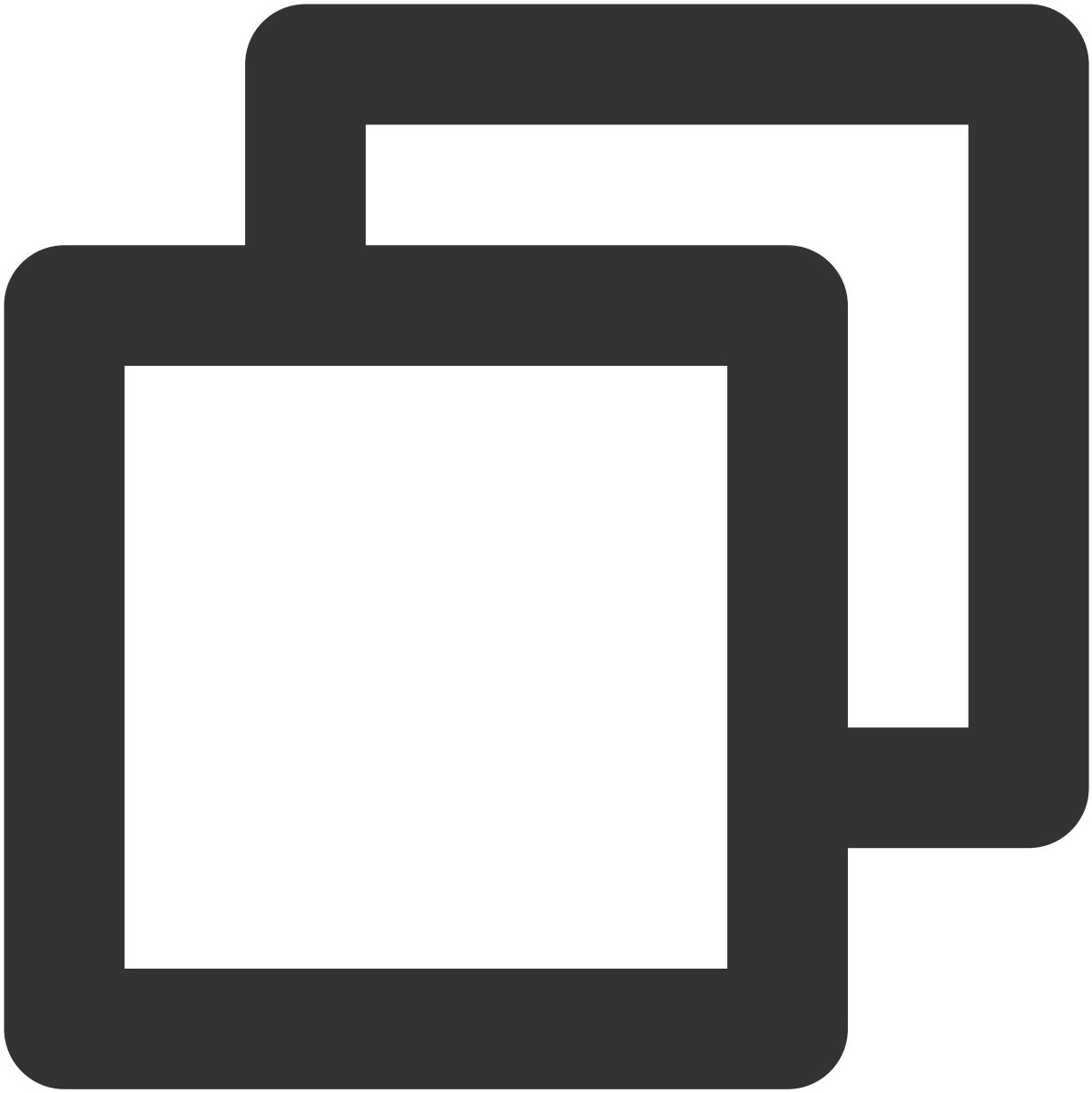
```
flutter pub get
```

3. 環境変数の設定

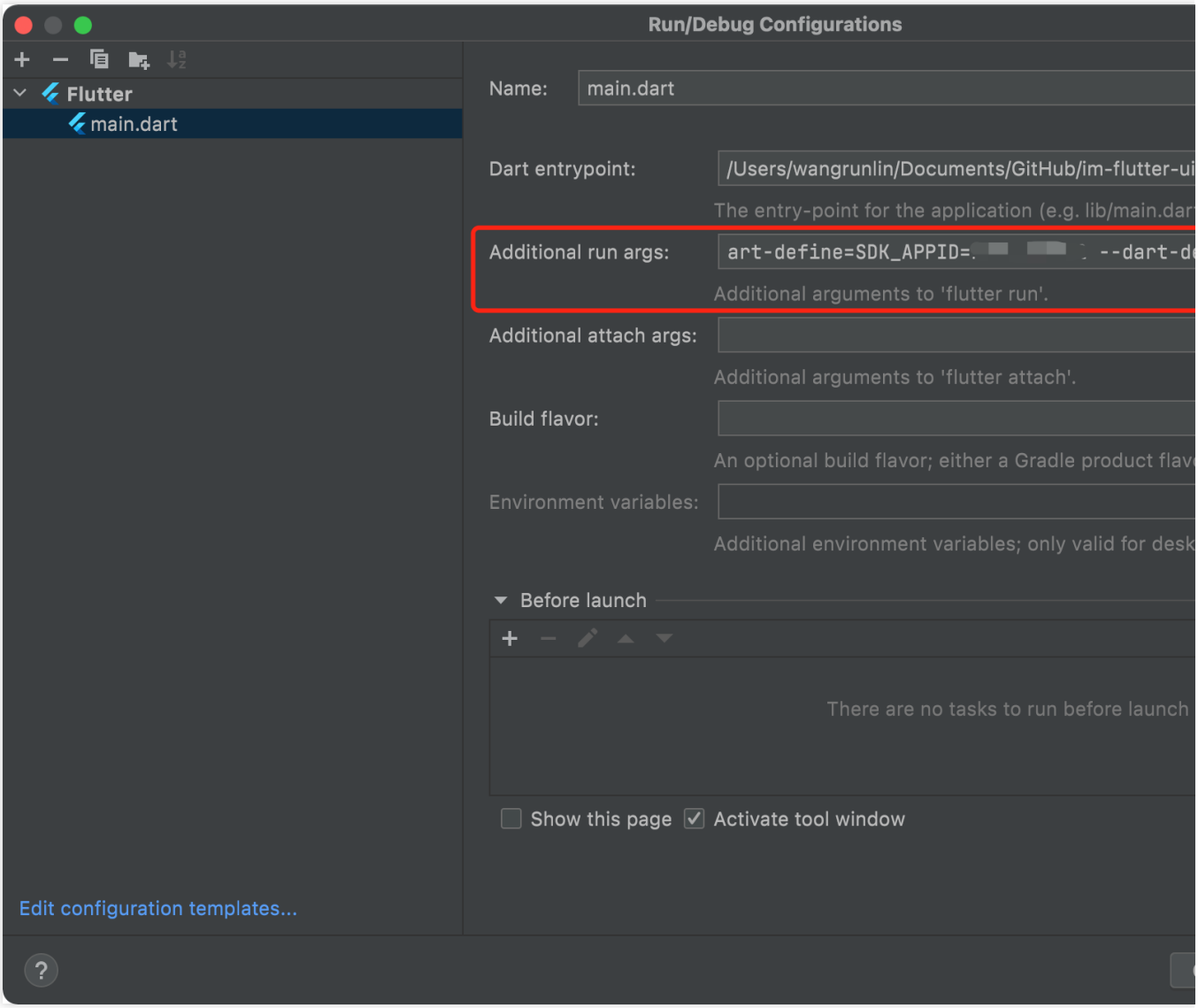
右上隅の実行ボタンの横で、 `main.dart` にマウスをhoverし、 `Edit Configurations` を設定します。



ポップアップ表示されたウィンドウでは、 `Additional run args` を設定し、環境変数（SDKAPPIDなどの情報）を入力します。例：

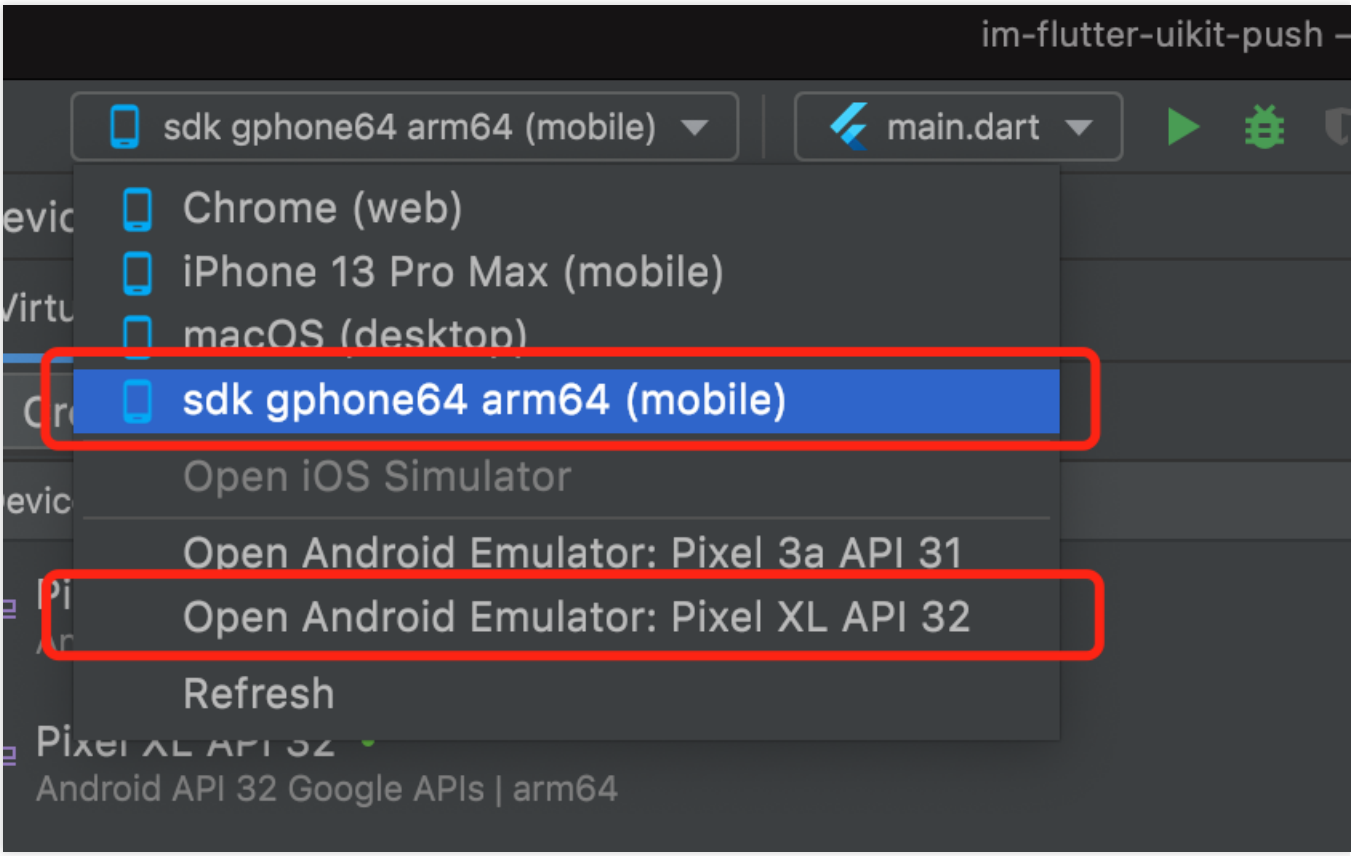


```
# SDK_APPID、KEYの2つのパラメータを置き換えてください
--dart-define=SDK_APPID={YOUR_SDKAPPID} --dart-define=ISPRODUCT_ENV=false --dart-de
```

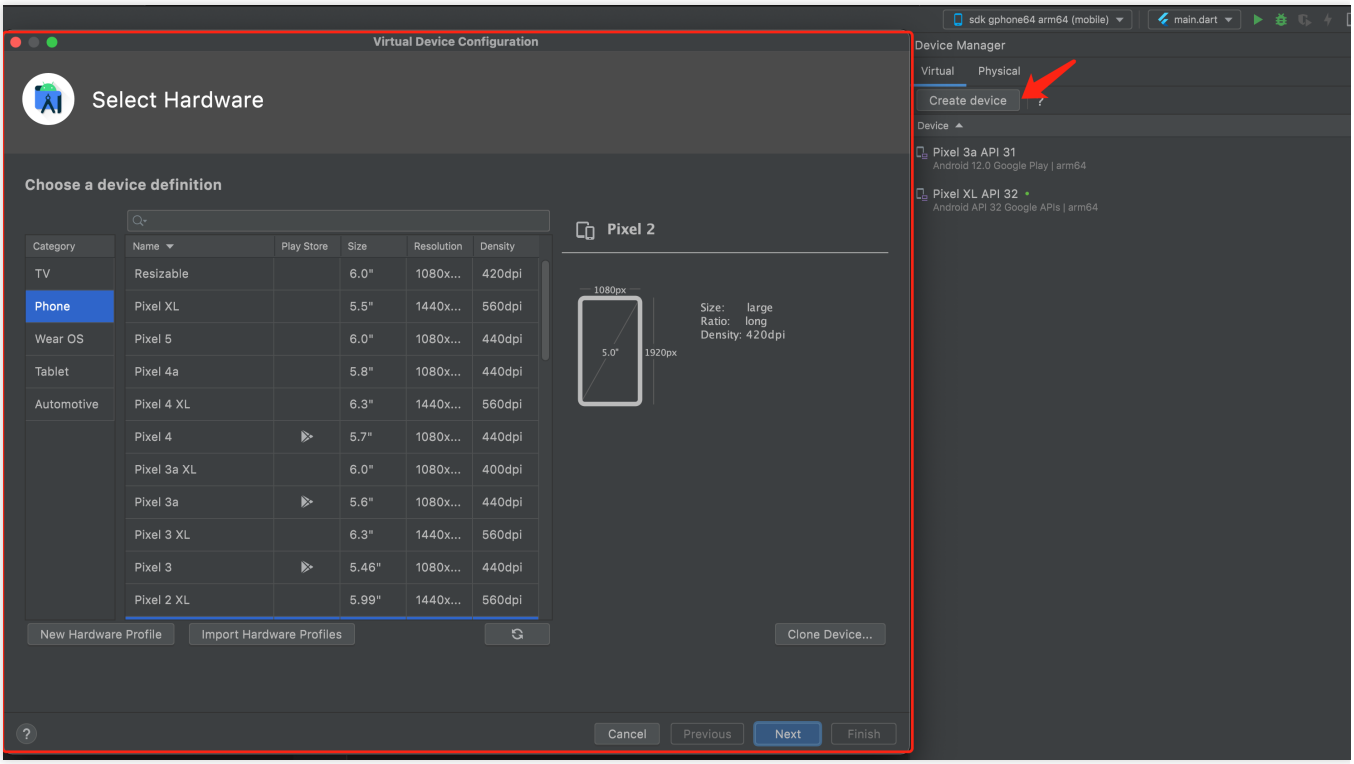


4. Androidシミュレータを作成します。

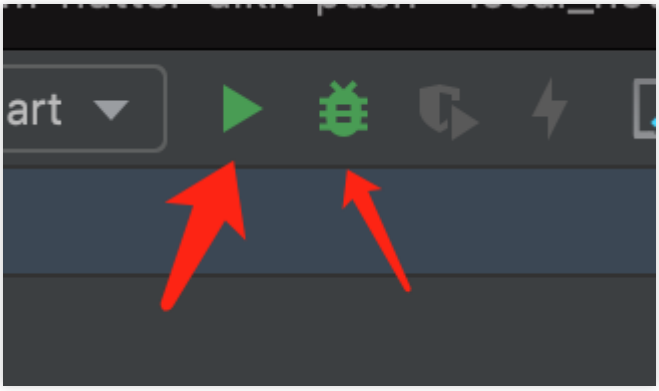
インストールしたばかりのシミュレータを起動し、それを選択します。



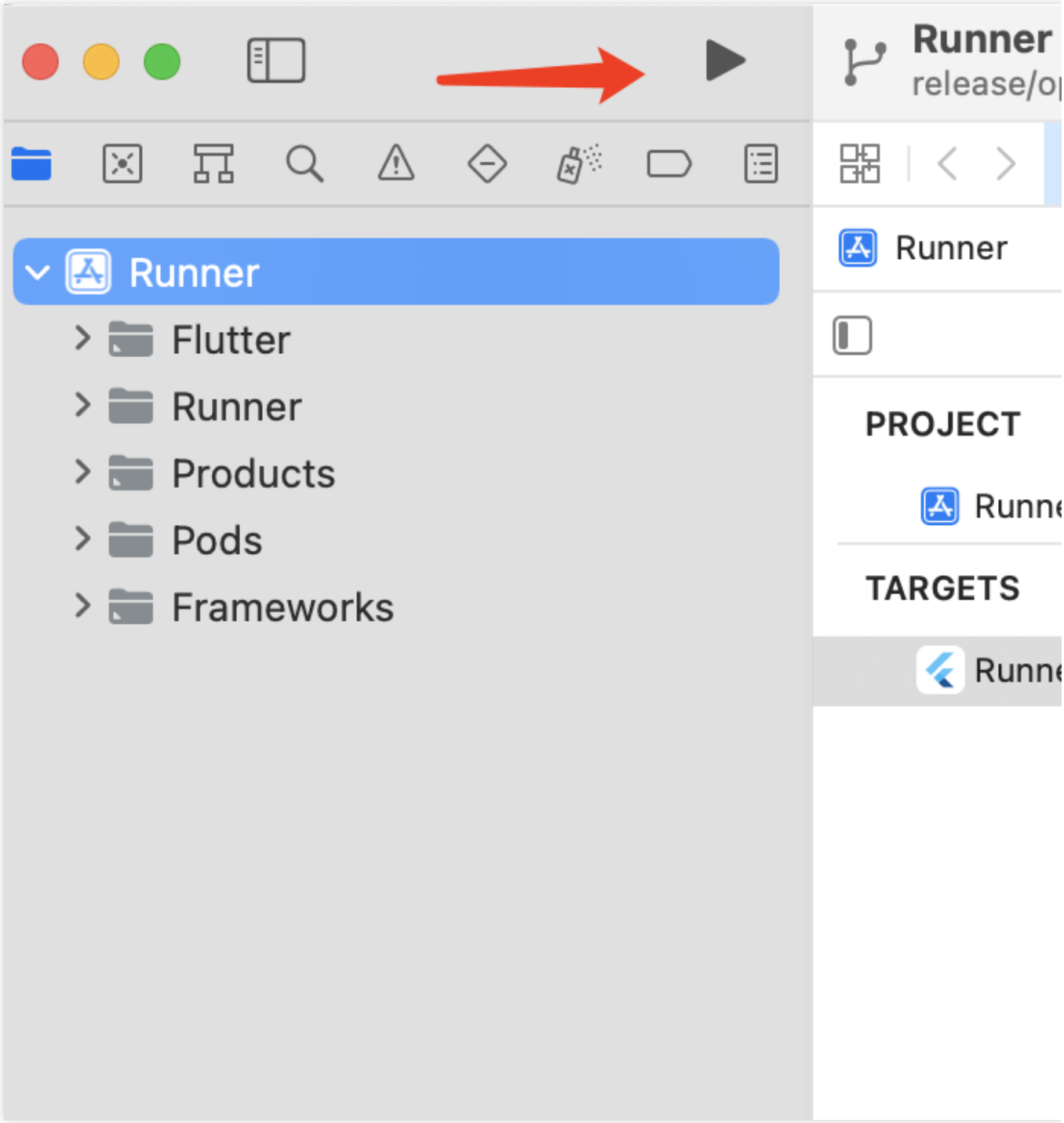
画面の右上隅にあるDevice Managerをクリックしてcreate devicesを完了し、シミュレータを作成します。Google FCMプッシュ機能を使用するには、できるだけGoogle Play Storeをサポートするデバイスをインストールすることをお勧めします。



5. プロジェクトを実行します。
必要に応じて、下図左側のRunまたは右側のDebugをクリックしてプロジェクトを実行します。



説明：
UIは部分的に調整され更新される可能性があります。最新バージョンを基準としてください。
1. Xcodeでは、 `im-flutter-uikit/ios` ディレクトリを開きます。
2. iPhoneの実機に接続して、**Build And Run**をクリックします。iOSプロジェクトのコンパイルが終了すると、新しいウィンドウにXcodeプロジェクトが表示されます。
3. iOSプロセスを開き、メインTargetのSigning & Capabilities（Apple開発者アカウントが必要）を設定すると、プロジェクトをiPhoneの実機で実行可能になります。
4. プロジェクトを起動し、実機でDemoのデバッグを実行します。



Demoコードの構造の概要

説明：

DemoのUIおよび業務ロジックの部分には、Flutter TUIKitを使用しています。Demo層自体は、Appのビルド、ナビゲーションリダイレクトの処理およびインスタンス化されたTUIKit内の各コンポーネントの呼び出しのみに用いられます。

フォルダ	説明

lib	プログラムのコアディレクトリ
lib/i18n	国際化関連コードです。ここでの国際化には、TUIKit自体の機能の国際化や多言語対応は含まれません。必要に応じてインポートできます
lib/src	プロジェクトのメインディレクトリ
lib/src/pages	このDemoのいくつかの重点ナビゲーションページです。プロジェクトの初期化完了後、`app.dart` はアニメーションをロードして表示し、ログインセッションを判断して、ユーザーを `login.dart` または `home_page.dart` に誘導します。ユーザーのログイン後、ログイン情報は `shared_preference` プラグインを通じてローカルに保存されます。その後、毎回のアプリケーション起動の際、ローカルに保存されたログイン情報を発見すると、自動的にその情報を使用してログインし、情報がない場合やログインに失敗した場合はログインページに誘導します。自動ログイン中、ユーザーはまだ `app.dart` で、ロードしたアニメーションを見ることができます。 `home_page.dart` にはボトムTabが含まれ、このDemoの4つの主要機能ページの切り替えを担います。
lib/utils	いくつかのツール関数クラス

基本的に、 `lib/src` 内の各dartファイルはTUIKitコンポーネントをインポートしており、ファイル内でコンポーネントをインスタンス化すると、ページレンダリングが可能になります。
主要なファイルは次のとおりです：

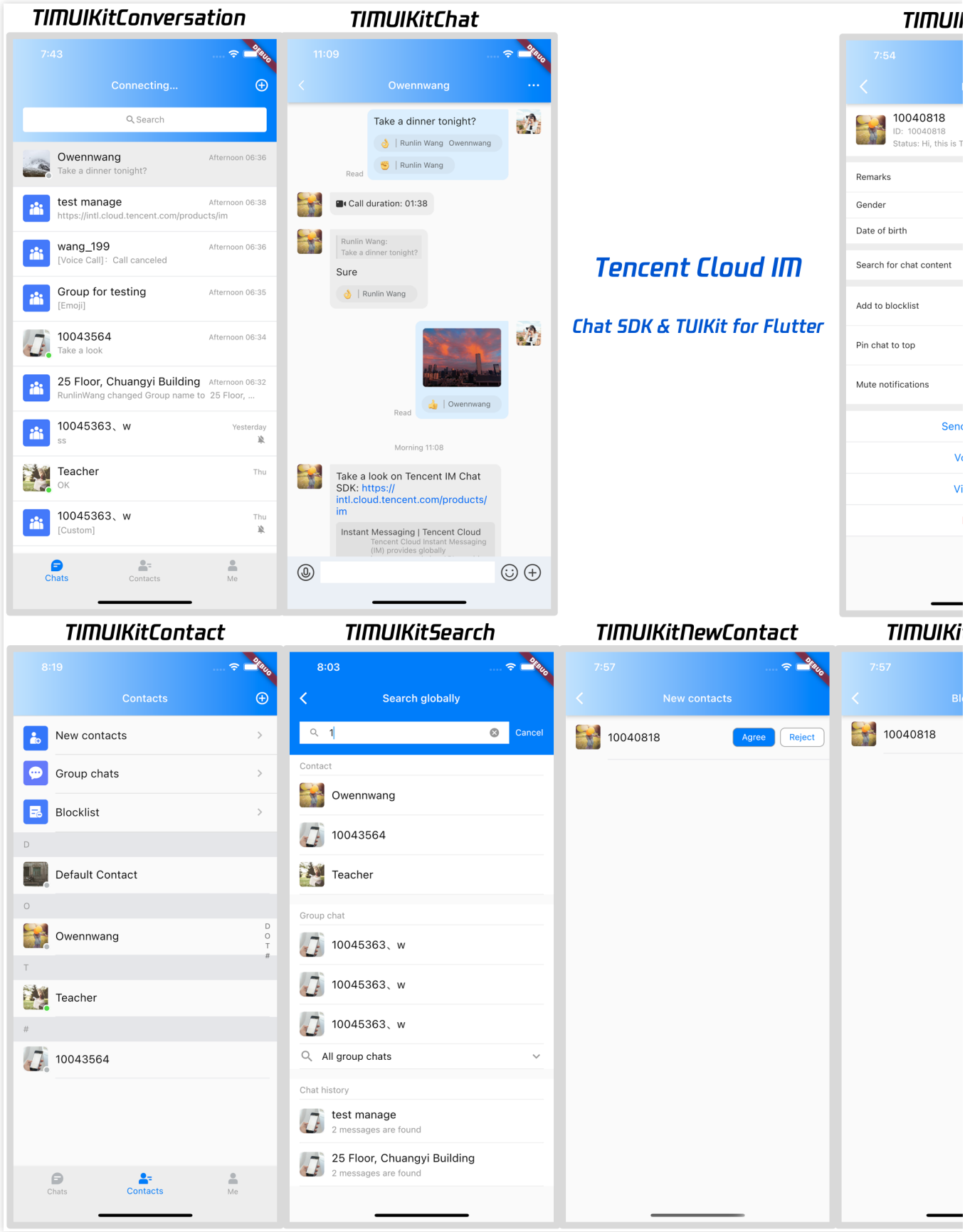
lib/srcの主要なファイル	ファイルの説明
add_friend.dart	フレンド追加申請ページです。 `TIMUIKitAddFriend` コンポーネントを使用します
add_group.dart	グループ参加申請ページです。 `TIMUIKitAddGroup` コンポーネントを使用します
blacklist.dart	ブラックリスト一覧ページです。 `TIMUIKitBlackList` コンポーネントを使用します
chat.dart	メインチャットページです。TUIKitのフルセットのチャット機能、 `TIMUIKitChat` コンポーネントを使用します
chatv2.dart	メインチャットページです。アトミック機能、 `TIMUIKitChat` コンポーネントを使用します
contact.dart	連絡先ページです。 `TIMUIKitContact` コンポーネントを使用します
conversation.dart	セッションリストインターフェースです。 `TIMUIKitConversation` コンポーネントを使用します
create_group.dart	グループチャット開始ページです。Demoのみで実装でき、コンポーネントは

	使用しません
group_application_list.dart	グループ参加申請リストページです。`TIMUIKitGroupApplicationList` コンポーネントを使用します
group_list.dart	グループリストページです。`TIMUIKitGroup` コンポーネントを使用します
group_profile.dart	グループプロファイルおよびグループ管理ページです。`TIMUIKitGroupProfile` コンポーネントを使用します
newContact.dart	連絡先フレンド申請ページです。`TIMUIKitNewContact` コンポーネントを使用します
routes.dart	Demoのルートです。ログインページ `login.dart` またはホームページ `home_page.dart` にナビゲートします。
search.dart	グローバル検索およびセッション内検索ページです。`TIMUIKitSearch`（グローバル検索）および `TIMUIKitSearchMsgDetail`（セッション内検索）コンポーネントを使用します
user_profile.dart	ユーザー情報およびリレーションシップチェーン保守ページです。`TIMUIKitProfile` コンポーネントを使用します

大部分のTUIKitコンポーネントはナビゲーションリダイレクトのメソッドに渡す必要があります。そのため、Demo層での `Navigator` の処理が必要です。
Demoの説明は以上です。Demoを直接変更して二次開発を行うか、またはこれを参照し、業務ニーズに合わせて実装することができます。

ソリューション2：UIを含む統合、TUIKitコンポーネントリポジトリの使用、IM機能埋め込みを半日で完了

TUIKitはTencent Cloud IM SDKのUIコンポーネントリポジトリであり、セッションリスト、チャットインターフェース、連絡先リストなどの共通のUIコンポーネントを提供しています。開発者は実際の業務ニーズに応じて、このコンポーネントリポジトリによってカスタムIMアプリケーションをすばやくビルドすることができます。
TUIKitのグラフィックとテキストの説明をご参照ください。
この部分は、TUIKitの使用について簡単に説明します。入門ガイドの詳細については、TUIKit統合基本機能をご参照ください。


TIMUIKitConversation
TIMUIKitChat
TIMUI
Tencent Cloud IM
Chat SDK & TUIKit for Flutter
TIMUIKitContact
TIMUIKitSearch
TIMUIKitNewContact
TIMUIKi

前提条件

Flutterプロジェクトの作成が完了している、またはベースになるFlutterプロジェクトがすでにあること。

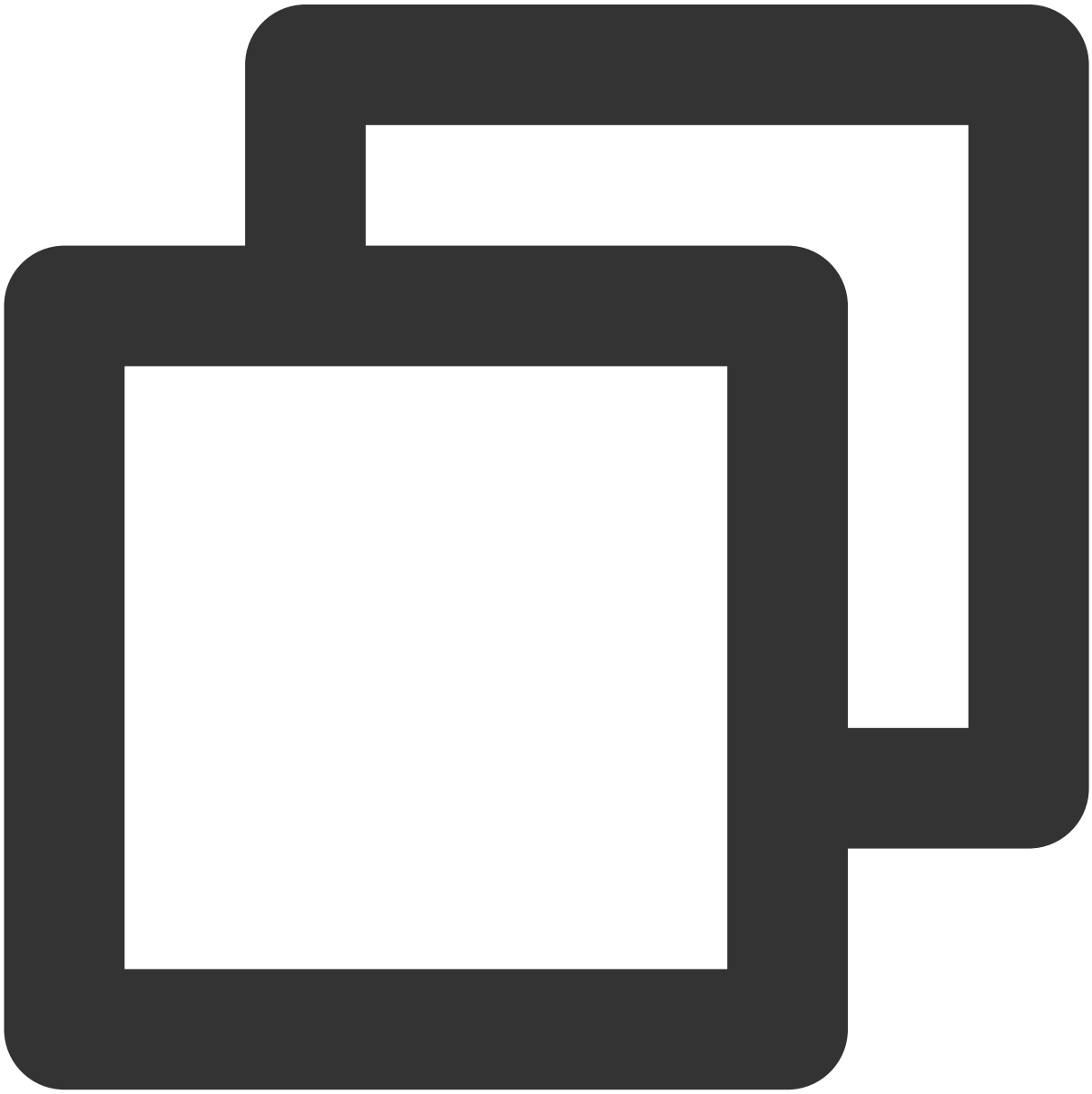
統合の手順

権限の設定

TUIKitの実行には、撮影/アルバム/録音/ネットワークなどの権限が必要なため、関連の機能を正常に使用するにはNativeファイルに手動でステートメントを作成する必要があります。

Android

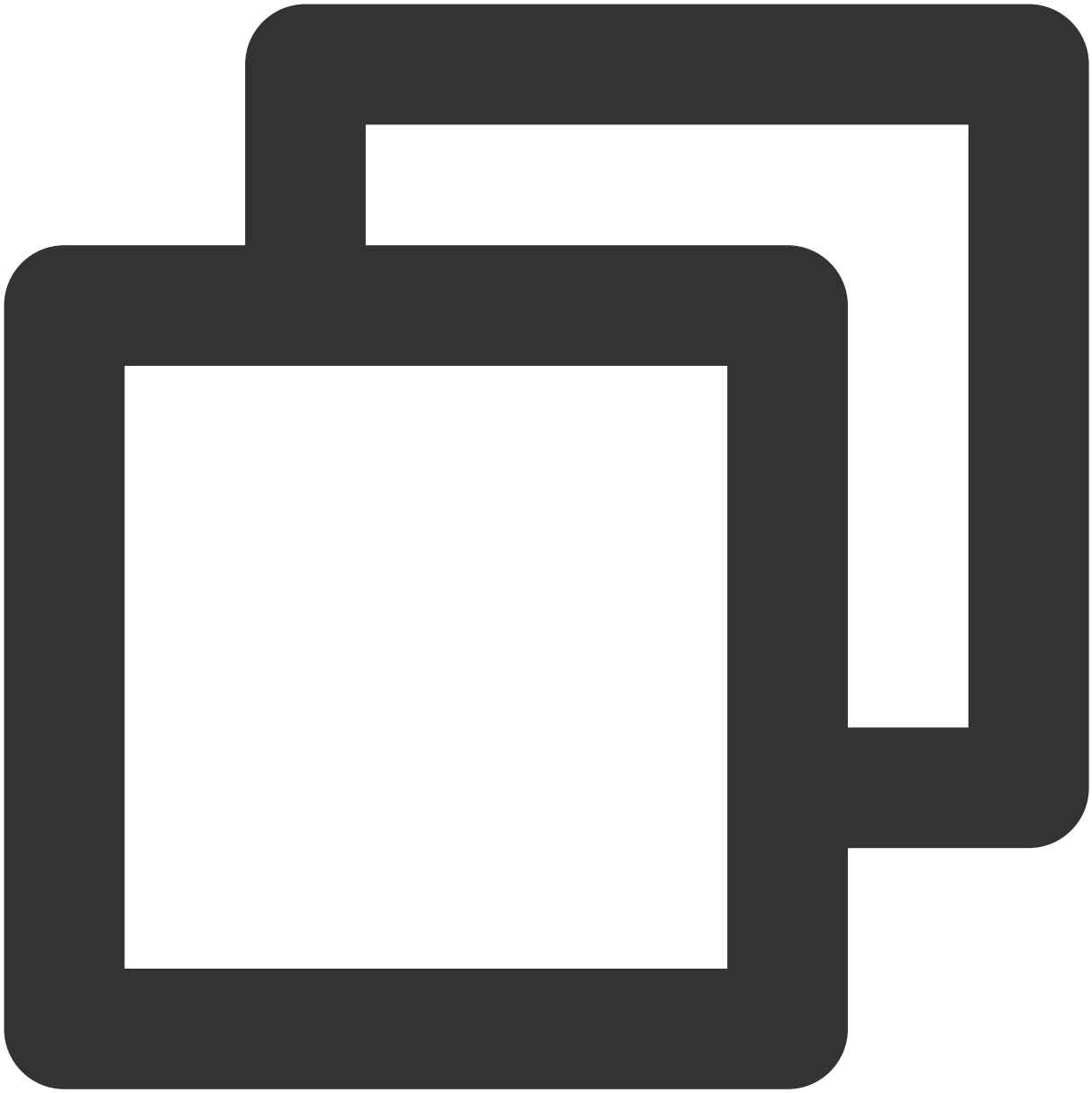
`android/app/src/main/AndroidManifest.xml` を開き、 `<manifest></manifest>` に次の権限を追加します。



```
    <uses-permission
        android:name="android.permission.INTERNET"/>
    <uses-permission
        android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission
        android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
        android:name="android.permission.VIBRATE"/>
    <uses-permission
        android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission
        android:name="android.permission.CAMERA"/>
```

iOS

`ios/Podfile` を開き、ファイルの末尾に次の権限コードを追加します。



```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
          config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
            '$(inherited)',
            'PERMISSION_MICROPHONE=1',
            'PERMISSION_CAMERA=1',
            'PERMISSION_PHOTOS=1',
          ]
        end
```

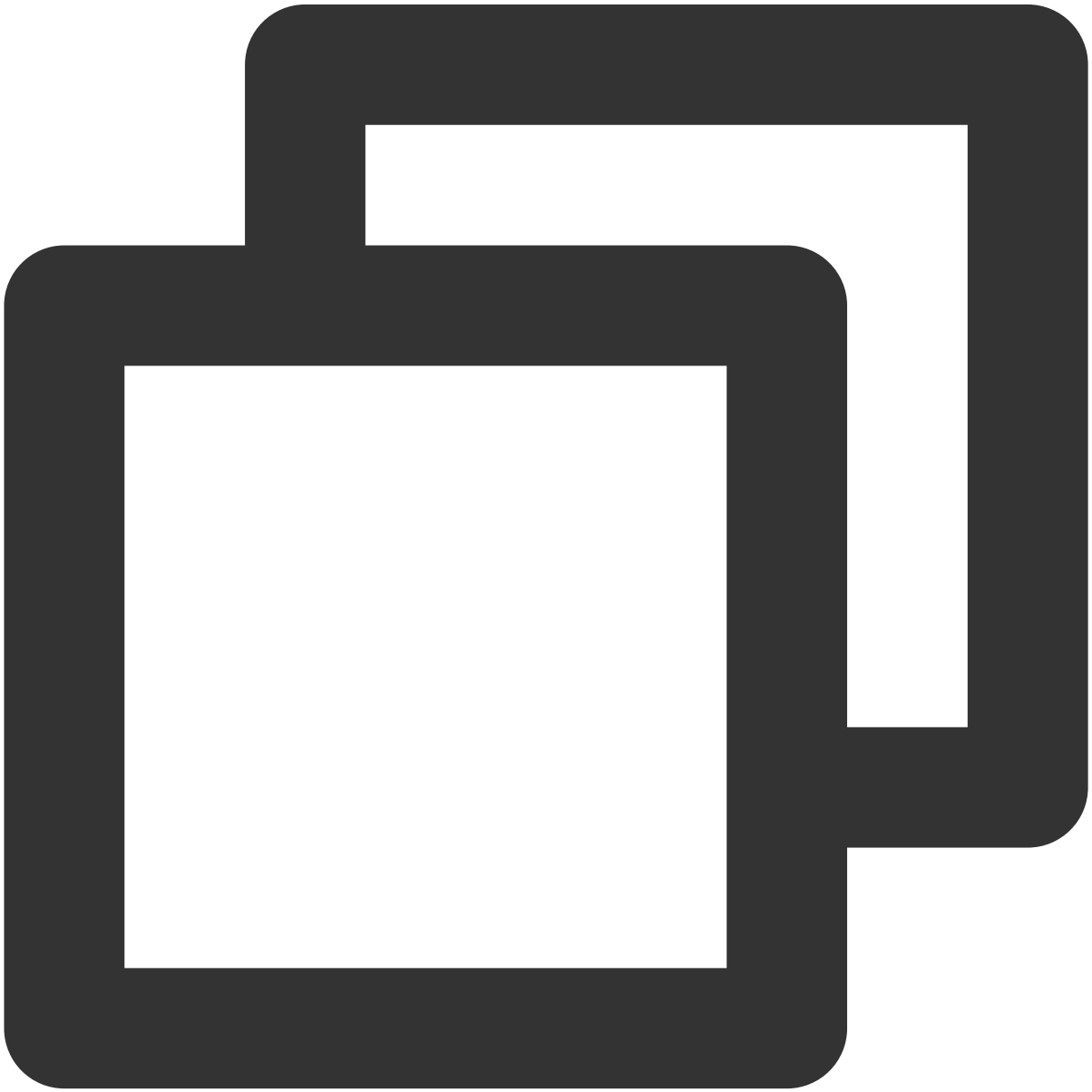
```
    end
  end
```

説明：

プッシュ機能を使用する必要がある場合は、プッシュに関する権限も追加する必要があります。詳細については、Flutterメーカーメッセージプッシュプラグイン統合ガイドをご確認ください。

IM TUIkitのインストール

TUIkitにはIM SDKが含まれているため、 `tencent_cloud_chat_uikit` をインストールすれば、基本のIM SDKをインストールする必要はありません。

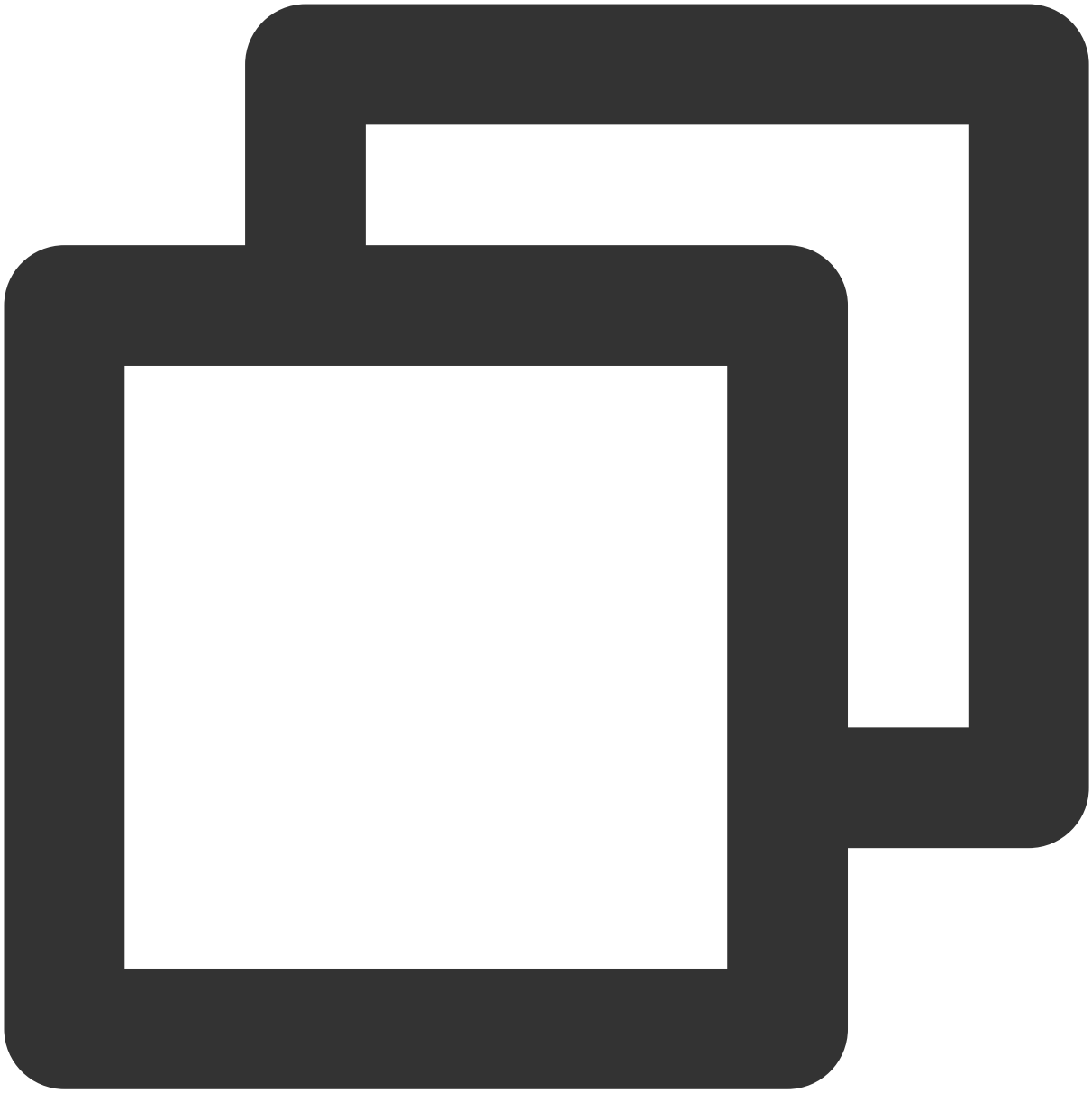


```
#コマンドラインでの実行：
flutter pub add tencent_cloud_chat_uikit
```

プロジェクトでWebをサポートする必要がある場合は、次の手順を実行する前に、Web互換性セクションを参照して、JSファイルを導入してください。

初期化

1. アプリケーションを起動する際にTUIKitを初期化します。
2. `TIMUIKitCore.getInstance()` を実行してから、初期化関数 `init()` を呼び出し、 `sdkAppID` を渡すよう、必ず確認してください。
3. APIエラーメッセージと、ユーザーに思い出させるための提案を取得しやすくするために、ここでonTUIKitCallbackListenerのリッスンをマウントすることをお勧めします。詳細については、この部分を参照してください。



```
/// main.dart
import 'package:tencent_cloud_chat_uikit/tencent_cloud_chat_uikit.dart';

final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
  @override
 void initState() {
   _coreInstance.init(
     sdkAppID: 0, // Replace 0 with the SDKAppID of your IM application when integr
     // language: LanguageEnum.en, // インターフェース言語の設定です。設定されていない場合は
     loglevel: LogLevelEnum.V2TIM_LOG_DEBUG,
     onTUIKitCallbackListener:  (TIMCallback callbackValue){}, // [設定をお勧めします。
```

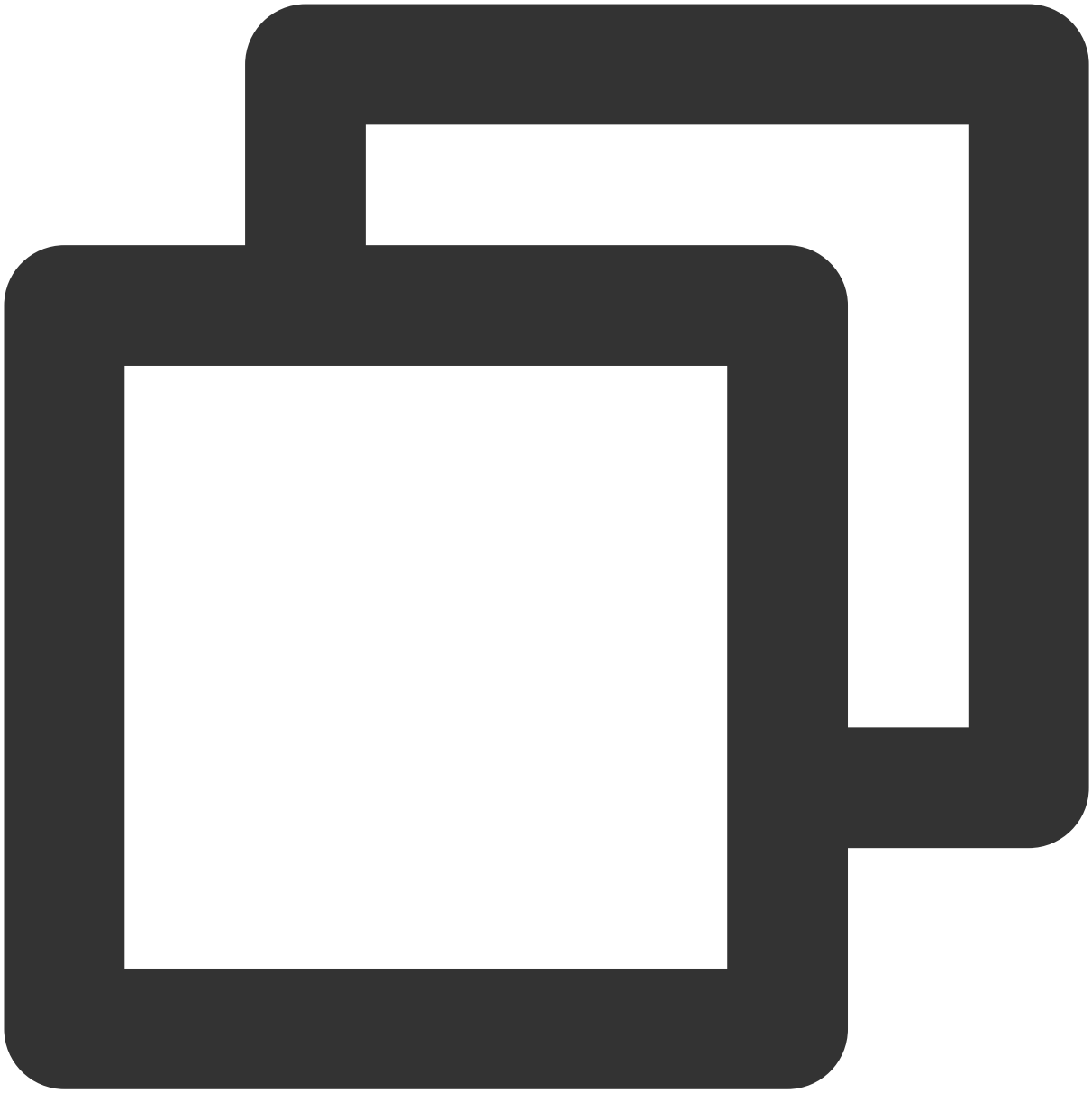
```
      listener: V2TimSDKListener());
    super.initState();
  }
}
```

説明：

それ以降のステップは、このステップでawaitの初期化が完了した後にのみ実行できます。

テストユーザーのログイン

1. この時点で、最初にコンソール上で作成したテストアカウントを使用して、ログイン検証を完了することが可能です。
2. `_coreInstance.login` メソッドを呼び出して、テストアカウントにログインします。



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

final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.login(userID: userID, userSig: userSig);
```

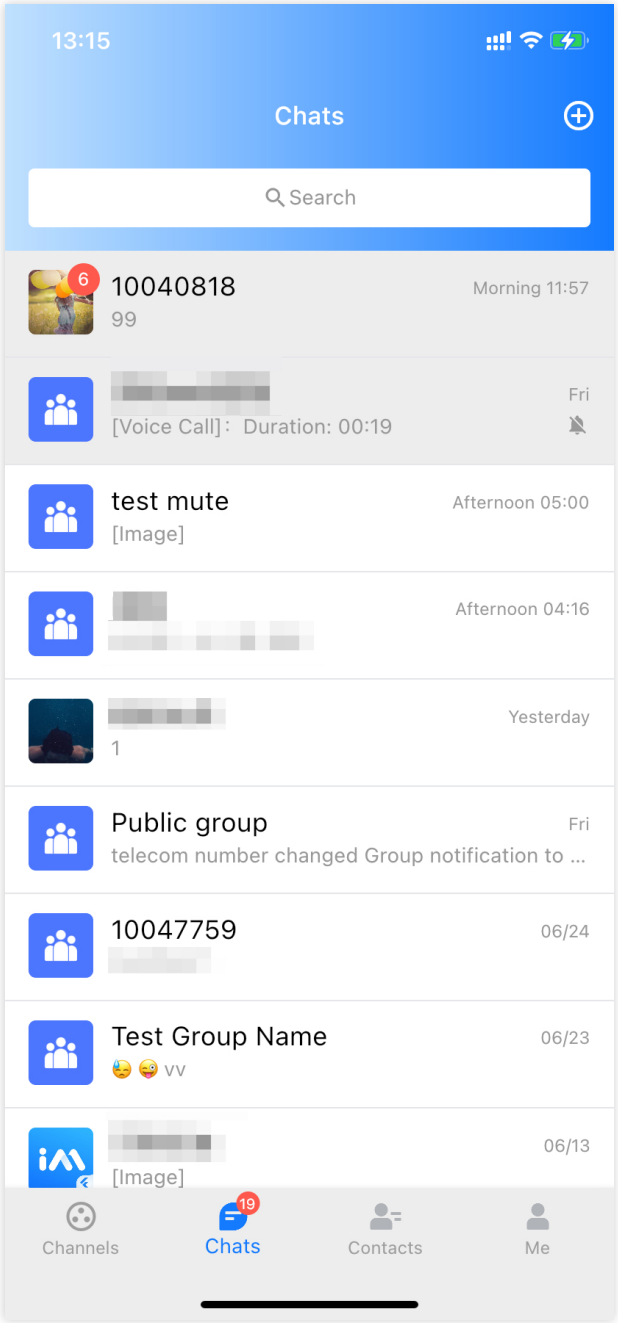
説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、サーバーで生成し、Appのインターフェース向けに提供する方法となります。 `UserSig` が必要なときは、

Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

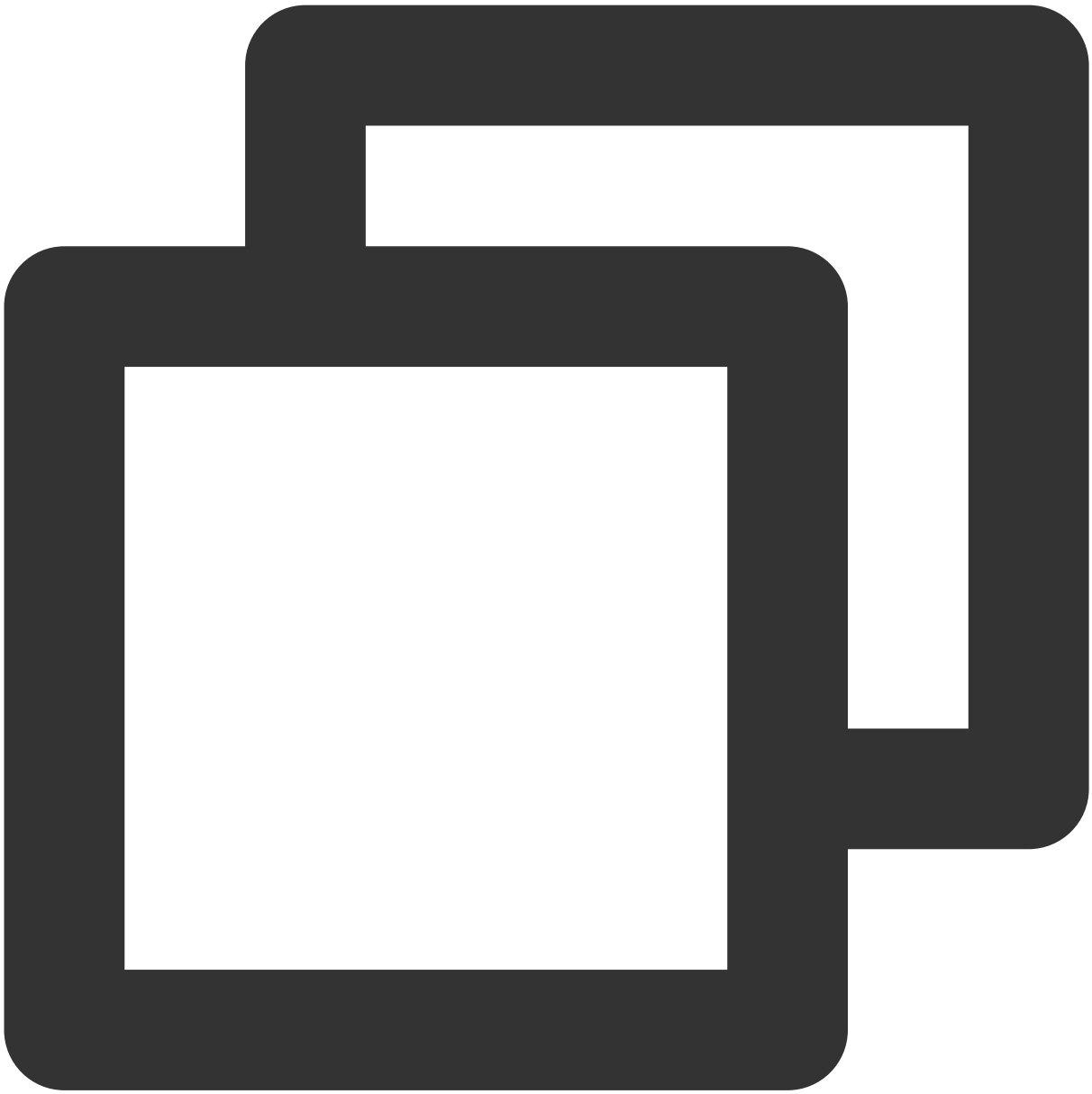
実装：セッションリストページ

セッションリストをIM機能のトップページにし、チャット記録のあるすべてのユーザーとのセッションおよびグループチャットセッションをカバーすることができます。



`Conversation` クラスを作成してください。 `body` では、 `TIMUIKitConversation` コンポーネントを使用してセッションリストをレンダリングします。

具体的なセッションチャットページにリダイレクトするためのナビゲーションに用いる、 `onTapItem` イベントの処理関数を渡すだけです。 `Chat` クラスに関しては、次の手順で説明します。

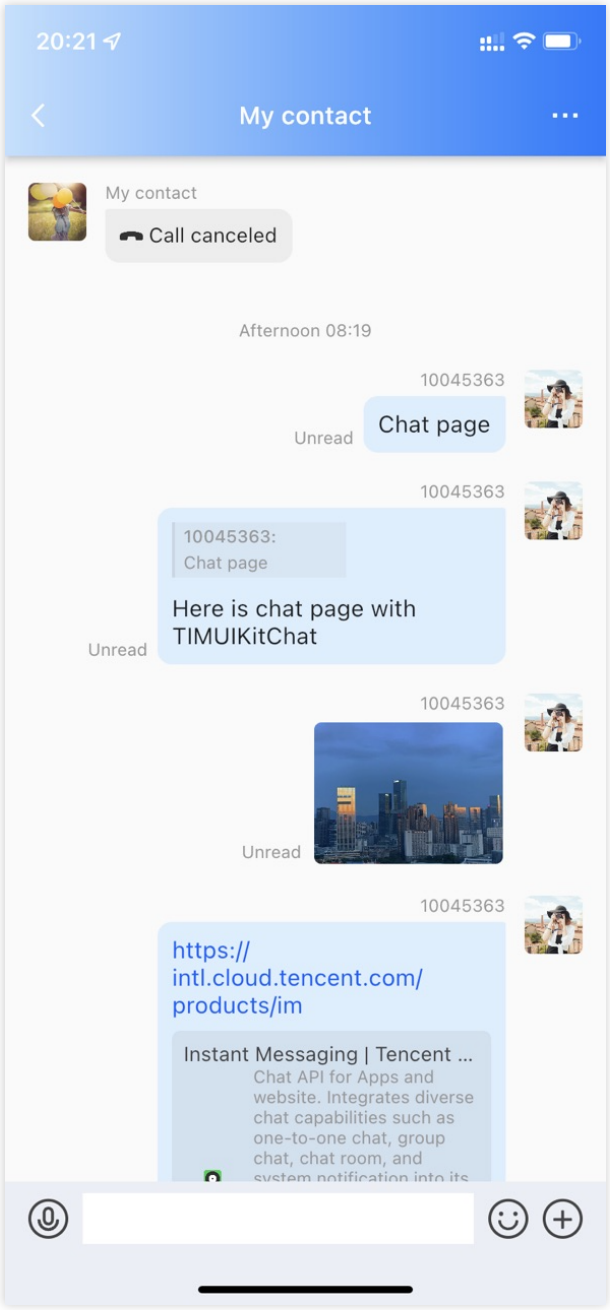


```
import 'package:flutter/material.dart';
import 'package:tencent_cloud_chat_uikit/tencent_cloud_chat_uikit.dart';

class Conversation extends StatelessWidget {
const Conversation({Key? key}) : super(key: key);
@override
Widget build(BuildContext context) {
return Scaffold(
```

```
    appBar: AppBar(
      title: const Text(
        "Message",
        style: TextStyle(color: Colors.black),
      ),
    ),
    body: TIMUIKitConversation(
      onTapItem: (selectedConv) {
        Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => Chat(
                selectedConversation: selectedConv,
              ),
            ));
      },
    ),
  );
  }
  }
```

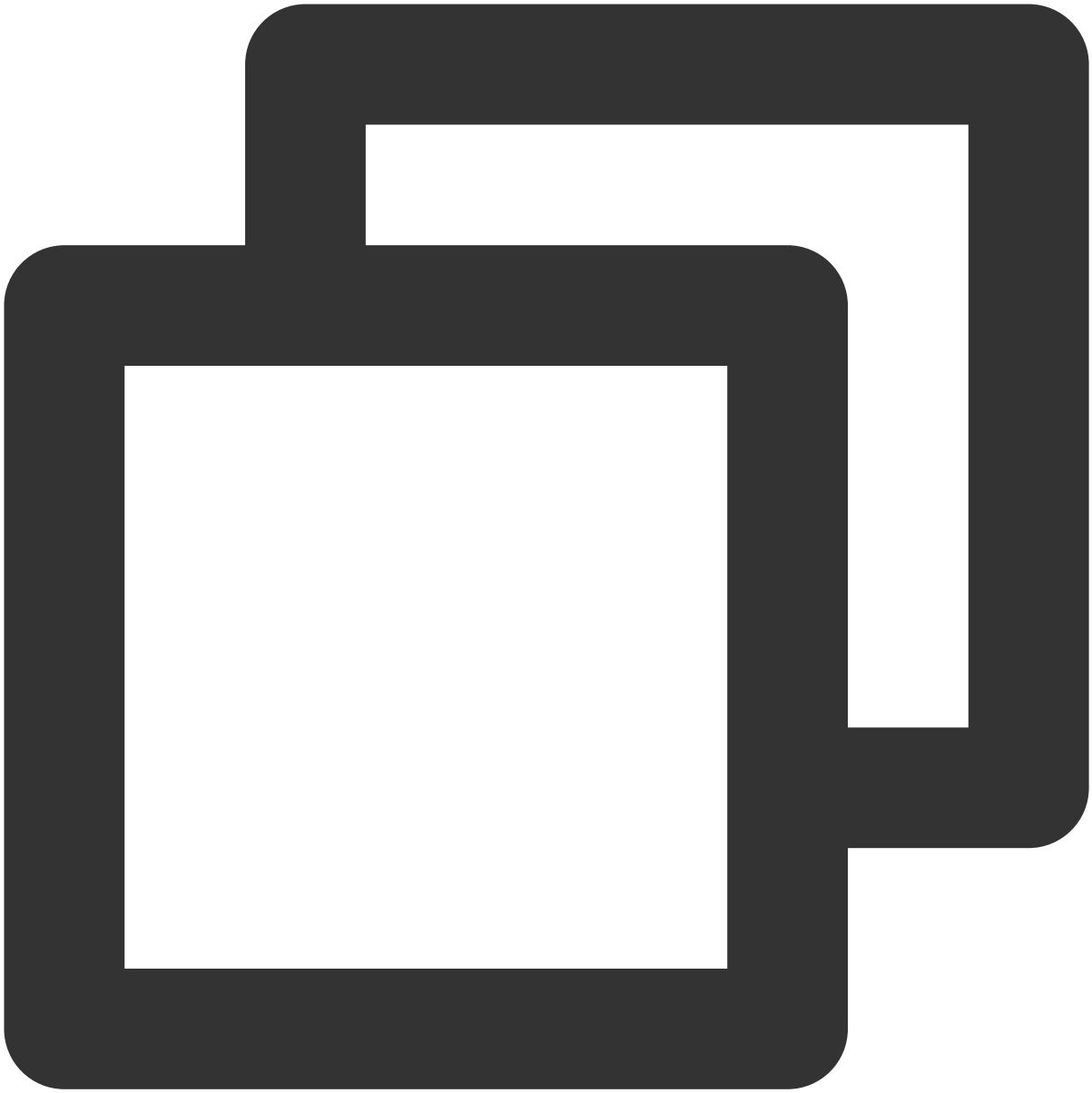
実装：セッションチャットページ

このページはトップのメインチャット履歴と、ボトムのメッセージ送信モジュールで構成されます。



`Chat` クラスを作成してください。 `body` では、 `TIMUIKitChat` コンポーネントを使用して、チャットページをレンダリングします。

連絡先の詳細情報ページへのリダイレクトに用いる、 `onTapAvatar` イベントの処理関数を渡すとよいでしょう。 `UserProfile` クラスに関しては、次の手順で説明します。

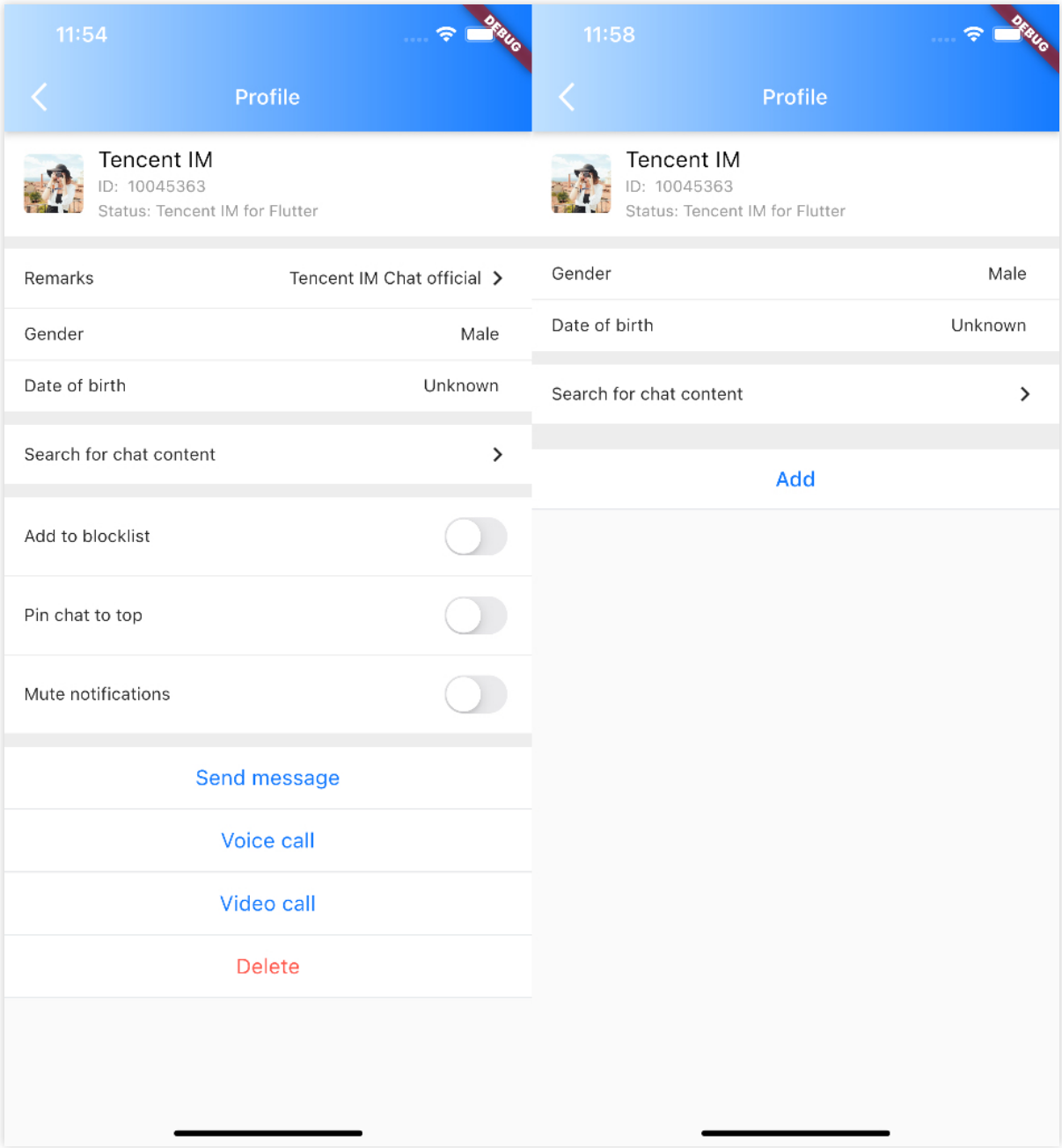


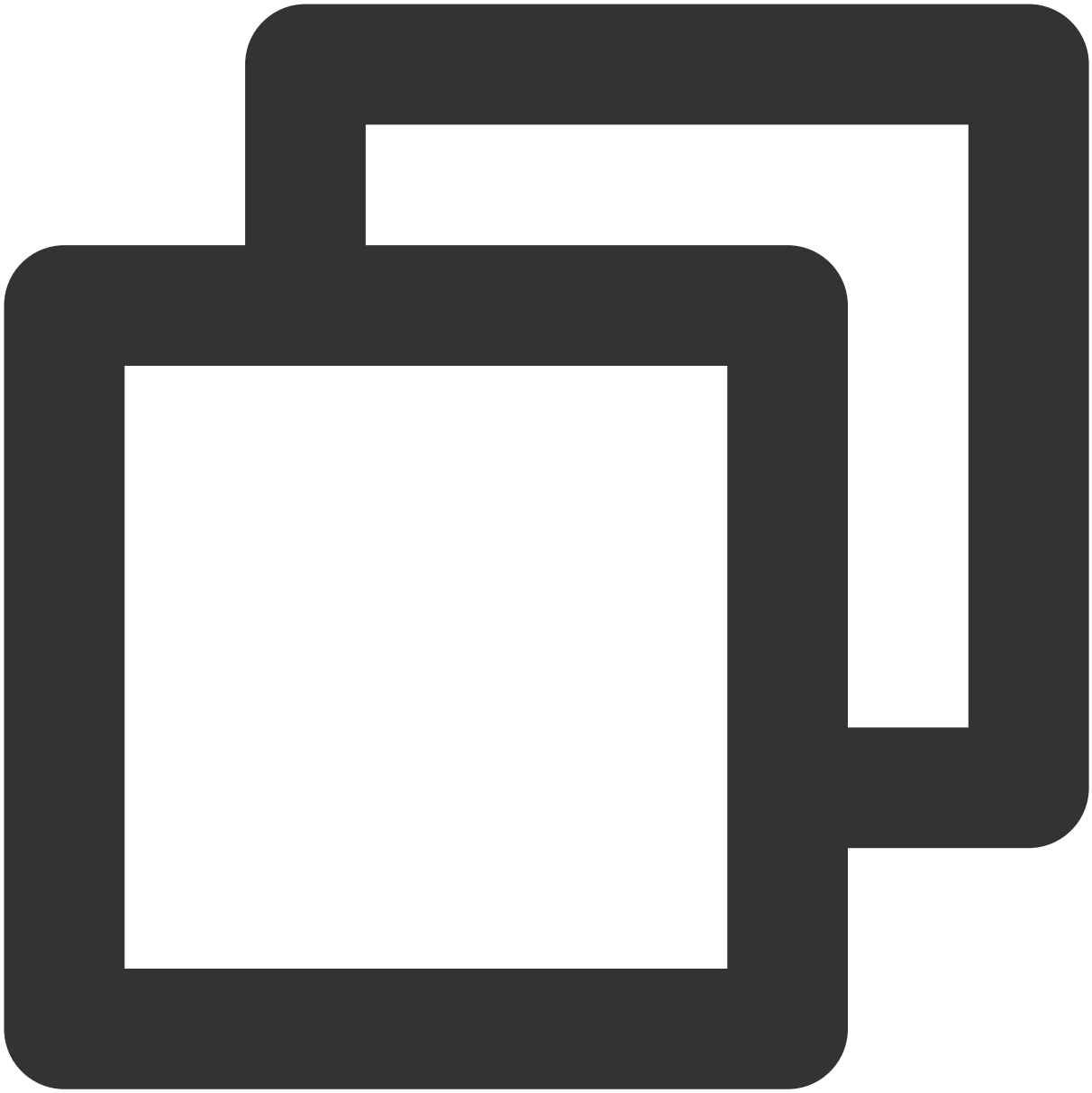
```
import 'package:flutter/material.dart';
import 'package:tencent_cloud_chat_uikit/tencent_cloud_chat_uikit.dart';

class Chat extends StatelessWidget {
final V2TimConversation selectedConversation;
const Chat({Key? key, required this.selectedConversation}) : super(key: key);
String? _getConvID() {
return selectedConversation.type == 1
    ? selectedConversation.userID
    : selectedConversation.groupID;
}
```

```
@override
Widget build(BuildContext context) {
return TIMUIKitChat(
  conversationID: _getConvID() ?? '', // groupID or UserID
  conversationType: selectedConversation.type ?? 1, // Conversation type
  conversationShowName: selectedConversation.showName ?? "", // Conversation displa
  onTapAvatar: (_) {
        Navigator.push(
            context,
            MaterialPageRoute(
             builder: (context) => UserProfile(userID: userID),
        ));
  }, // Callback for the clicking of the message sender profile photo. This callbac
);
}
```

実装：ユーザー詳細ページ

このページはデフォルトで、 `userID` を渡すだけで、フレンドかどうかに基づいて自動的にユーザー詳細ページを生成できるものです。
`UserProfile` クラスを作成してください。 `body` では、 `TIMUIKitProfile` コンポーネントを使用して、ユーザーの詳細およびリレーションシップチェーンページをレンダリングします。
説明：
このページをカスタマイズしたい場合は、 `profileWidgetBuilder` でカスタマイズが必要なprofileを渡すことを優先し、 `profileWidgetsOrder` に合わせて垂直方向の配置順序を決定します。満足できない場合は、 `builder` を使用できます。





```
import 'package:flutter/material.dart';
import 'package:tencent_cloud_chat_uikit/tencent_cloud_chat_uikit.dart';

class UserProfile extends StatelessWidget {
    final String userID;
    const UserProfile({required this.userID, Key? key}) : super(key: key);

    @override
    Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
        title: const Text(
          "Message",
          style: TextStyle(color: Colors.black),
        ),
      ),
      body: TIMUIKitProfile(
           userID: widget.userID,
      ),
     );
   }
}
```

この時点で、アプリケーションはメッセージの送受信、フレンドシップの管理、ユーザー詳細情報の表示、セッションリストの表示ができるようになっています。

その他の機能

引き続き以下のTUIKitプラグインを使用して、完全なIM機能をスピーディーに実装することができます。
TIMUIKitContact：連絡先リストページです。
TIMUIKitGroupProfile：グループ情報ページです。使用方法は `TIMUIKitProfile` と基本的に同じです。
TIMUIKitGroup：グループリストインターフェースです。
TIMUIKitBlackList：ブラックリストインターフェースです。
TIMUIKitNewContact：連絡先（友達）申請リストです。小さな赤い点を外部に表示する必要がある場合は、監視を自動的にマウントする `TIMUIKitUnreadCount` 小さな赤い点コンポーネントを使用できます。
ローカル検索： `TIMUIKitSearch` グローバル検索コンポーネントです。連絡先/グループ/チャット記録のグローバル検索をサポートし、 `TIMUIKitSearchMsgDetail` を使用して特定のセッションでチャット記録を検索することもサポートします。2つのモードは、 `conversation` が渡されるかどうかによって異なります。
UIコンポーネントの概要は、このグラフィックとテキストの概要または詳細のドキュメントをご参照ください。

ソリューション3：UI統合の自己実装

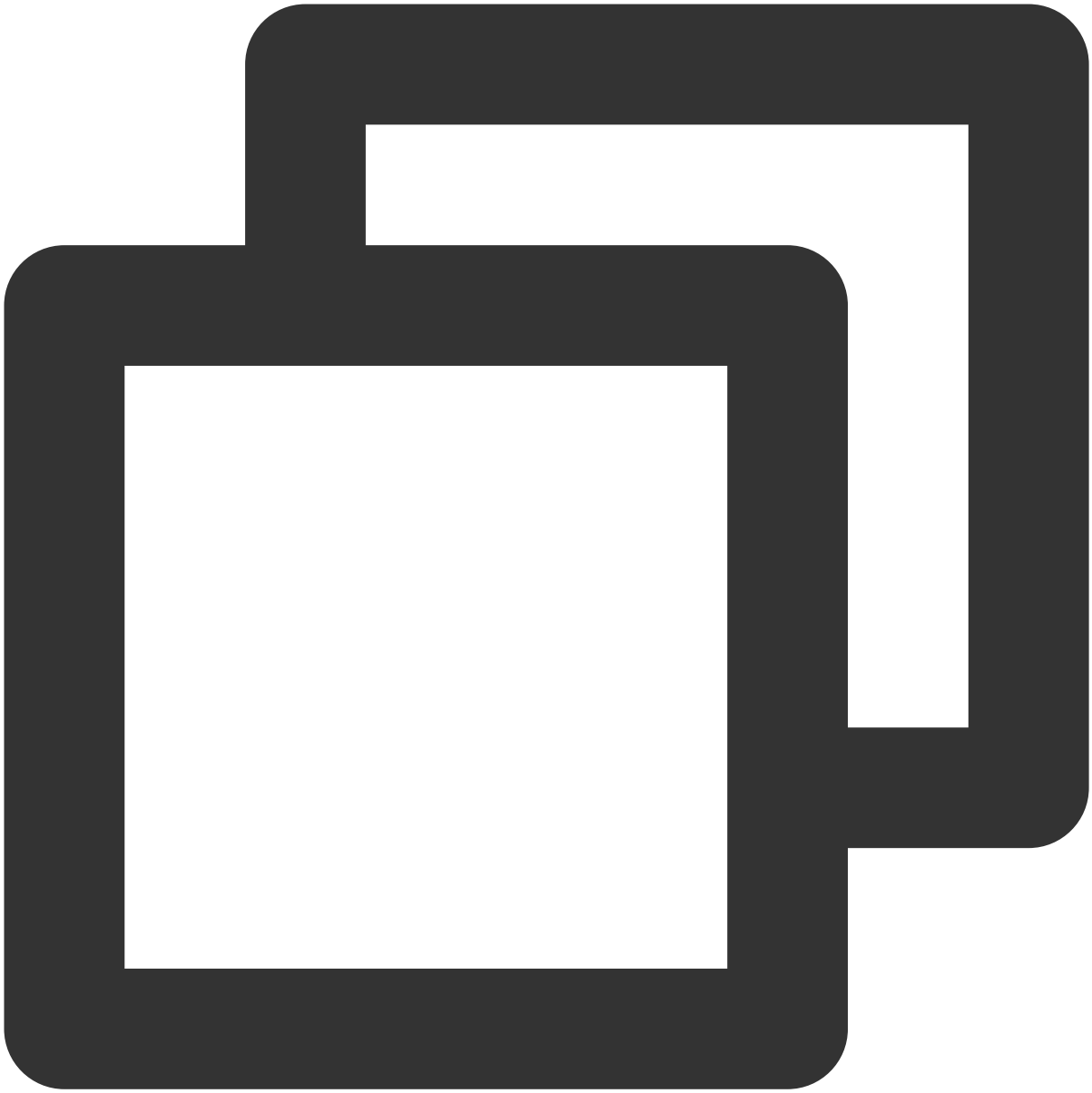
前提条件

Flutterプロジェクトの作成が完了している、またはベースになるFlutterプロジェクトがすでにあること。

統合の手順

IM SDKのインストール

このセグメントの詳細なドキュメント
次のコマンドを使用して、Flutter IM SDKの最新バージョンをインストールします。
コマンドラインでの実行：



```
flutter pub add tencent_cloud_chat_sdk
```

説明：

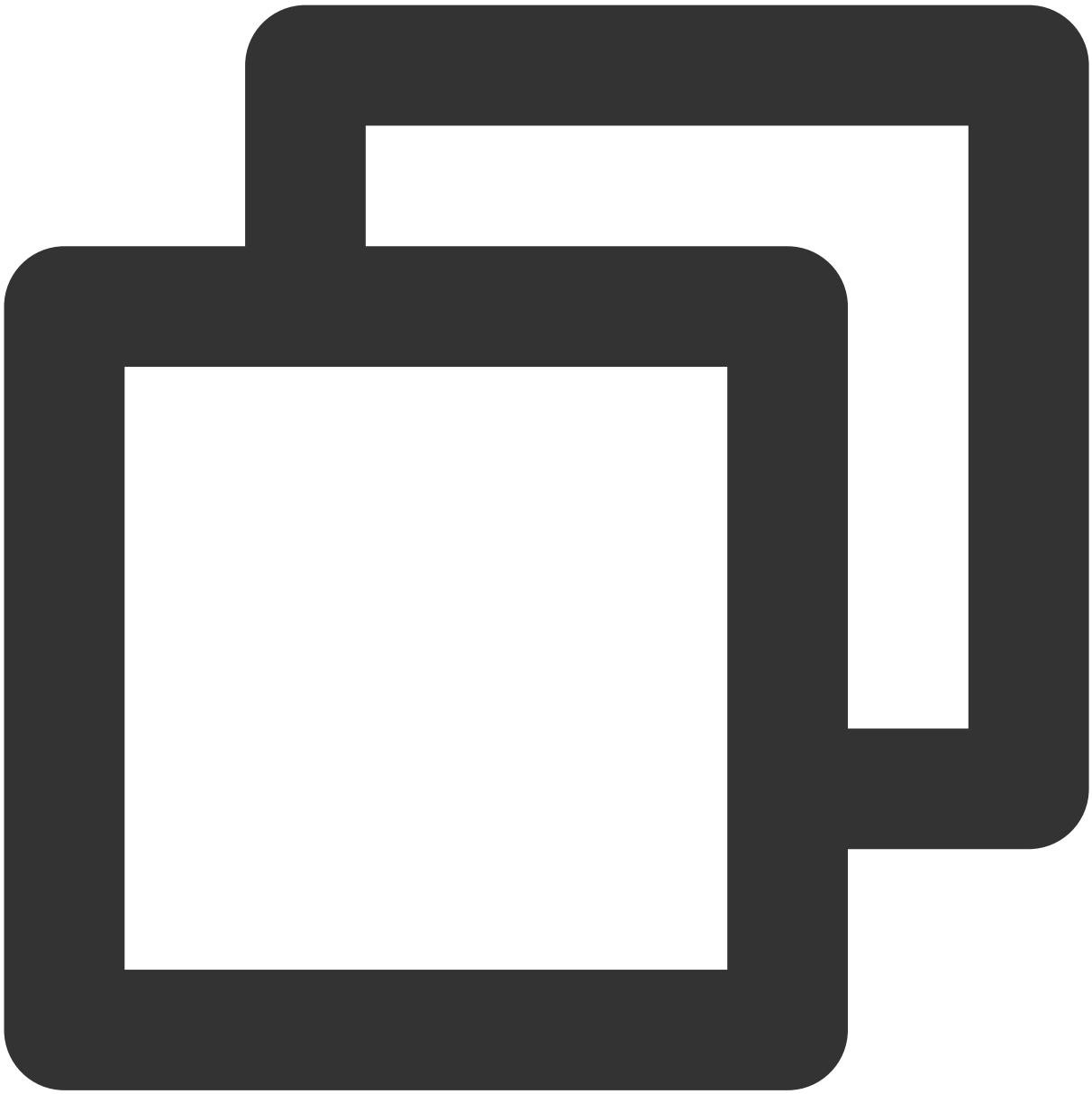
プロジェクトがWebまたはデスクトップ端末(macOS、Windows)も対象とする場合は、いくつかの追加手順が必要です。詳細については、それぞれのリンクをご参照ください。

SDKの初期化完了

このセグメントの詳細なドキュメント

`initSDK` を呼び出して、SDKの初期化を完了します。

お客様の `sdkAppID` を渡します。



```
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(),
);
```

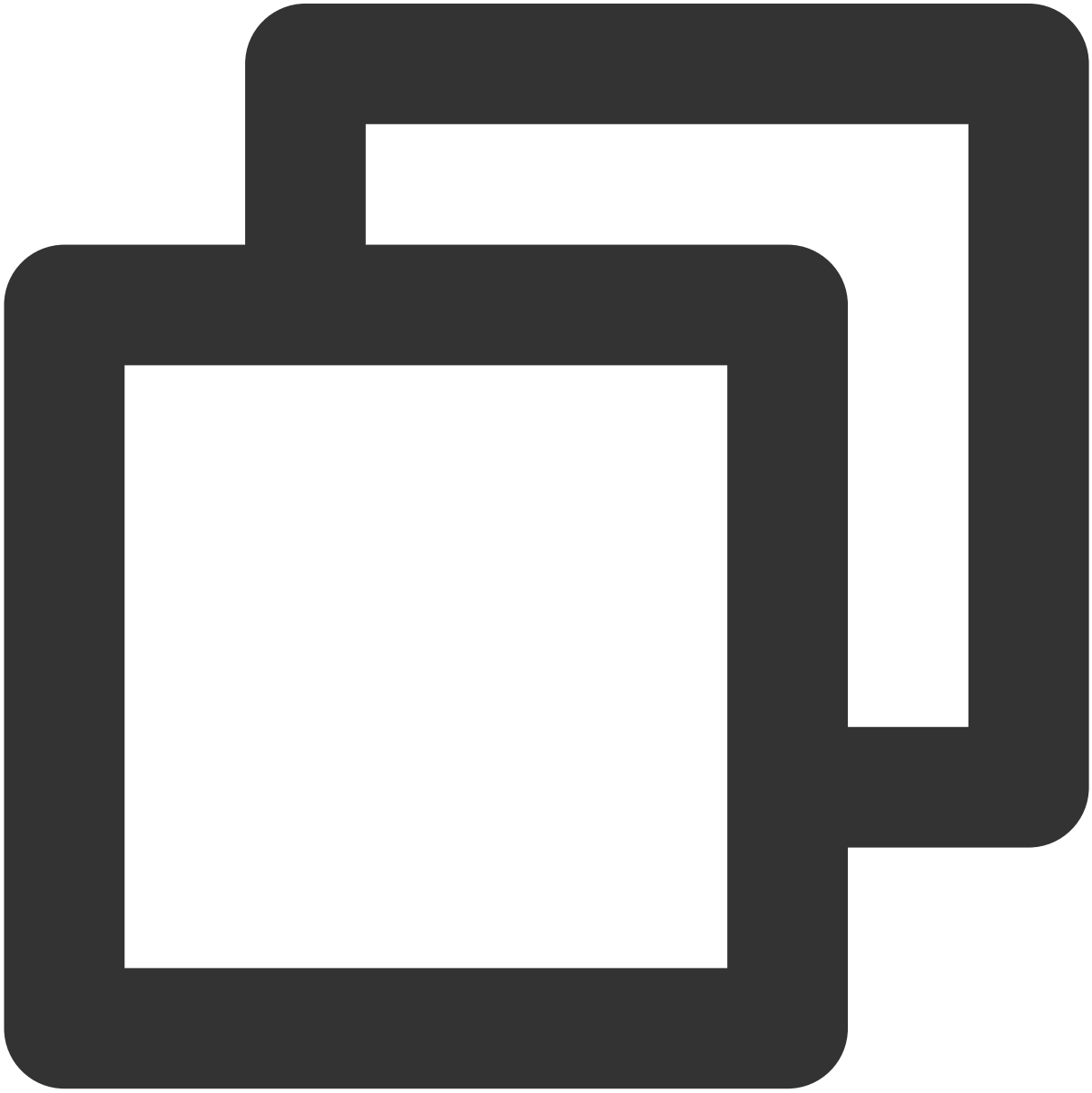
このステップでは、IM SDKに対して、ネットワーク状態やユーザー情報の変更などを含むいくつかの監視をマウントできます。詳細については、このドキュメントをご参照ください。

テストユーザーのログイン

このセグメントの詳細なドキュメント

この時点で、最初にコンソール上で作成したテストアカウントを使用して、ログイン検証を完了することが可能です。

`TencentImSDKPlugin.v2TIMManager.login` メソッドを呼び出し、テストアカウントにログインします。戻り値 `res.code` が0の場合、ログインは成功です。



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

説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、 `UserSig` の計算コードをサーバーに統合し、Appのインターフェース向けに提供する方法となります。 `UserSig` が必要なときは、Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

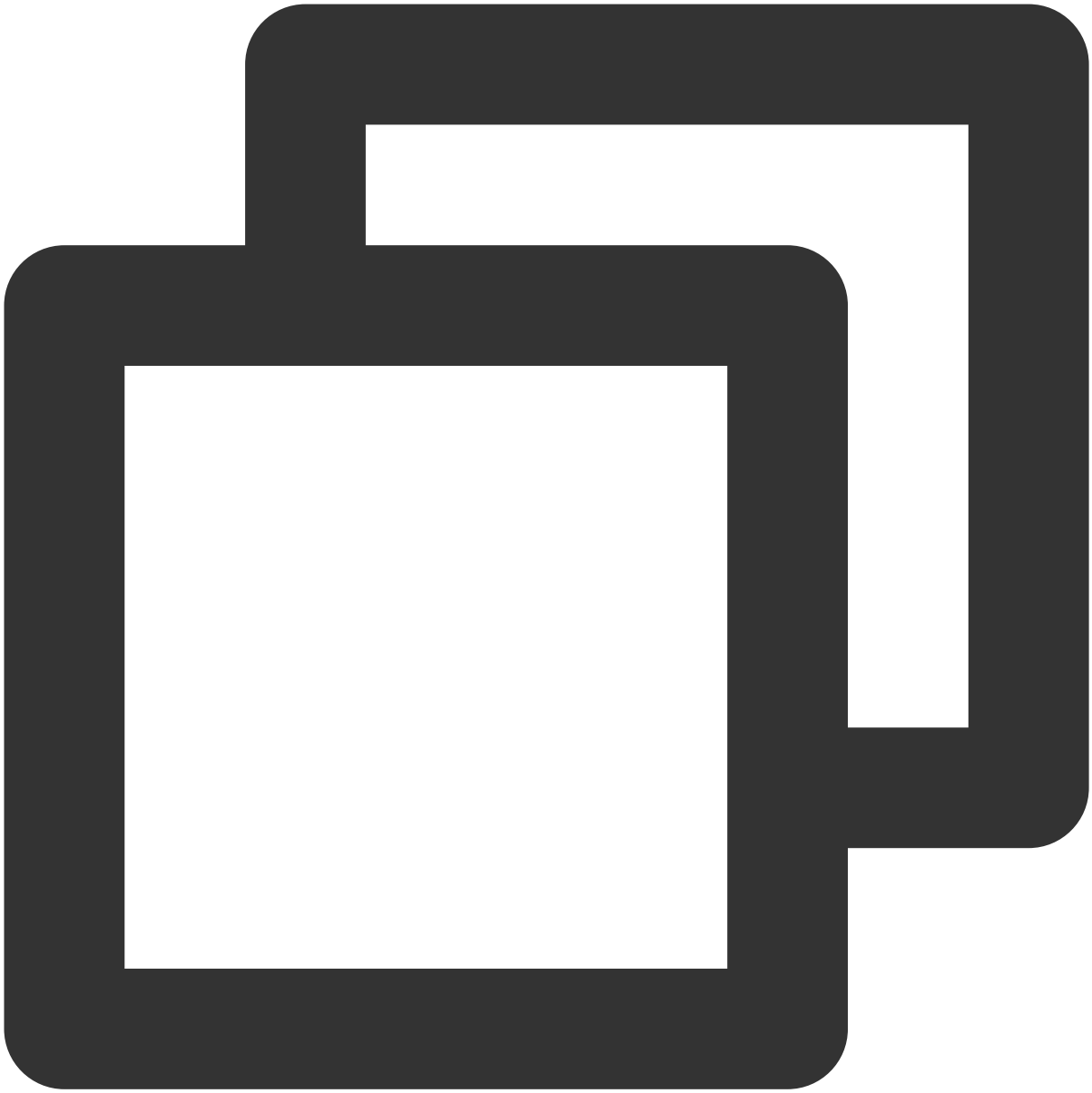
メッセージの送信

このセグメントの詳細なドキュメント

ここでは、テキストメッセージの送信を例に挙げます。そのフローは次のとおりです：

1. `createTextMessage(String)` を呼び出してテキストメッセージを作成します。
2. 戻り値に基づいて、メッセージIDを取得します。
3. `sendMessage()` を呼び出して、このIDのメッセージを送信します。 `receiver` には、その前に作成した別のテストアカウントIDを入力できます。シングルチャットメッセージを送信する場合は `groupID` の入力は不要です。

サンプルコード：



```
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()
```

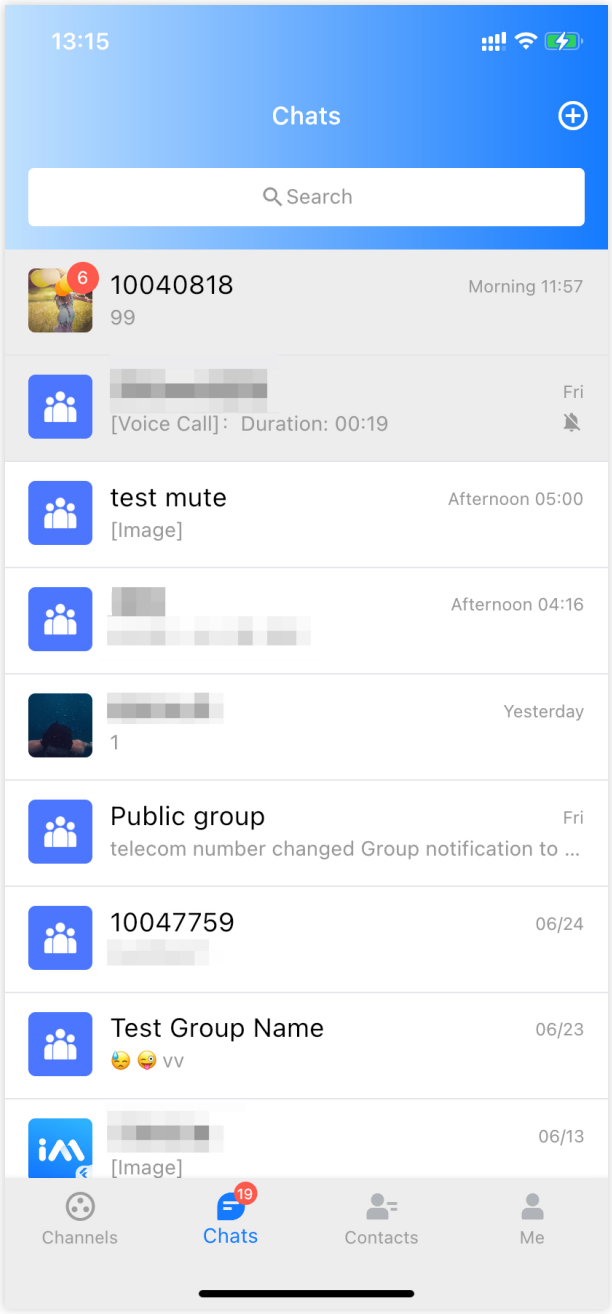
```
        .sendMessage(
            id: id, // Pass in the message creation ID to
            receiver: "The userID of the destination user",
            groupID: "The groupID of the destination group",
            );
```

説明：

送信に失敗した場合は、sdkAppIDが知らない人宛てのメッセージ送信をサポートしていないことが原因の可能性があります。コンソールで有効にしてからテストに使用してください。
このリンクをクリックして、フレンドリレーションシップチェーンのチェックを無効にしてください。

セッションリストの取得

このセグメントの詳細なドキュメント
前の手順でテストメッセージの送信が完了しましたので、別のテストアカウントでログインし、セッションリストをプルできるようになりました。



セッションリストの取得には2つの方法があります：

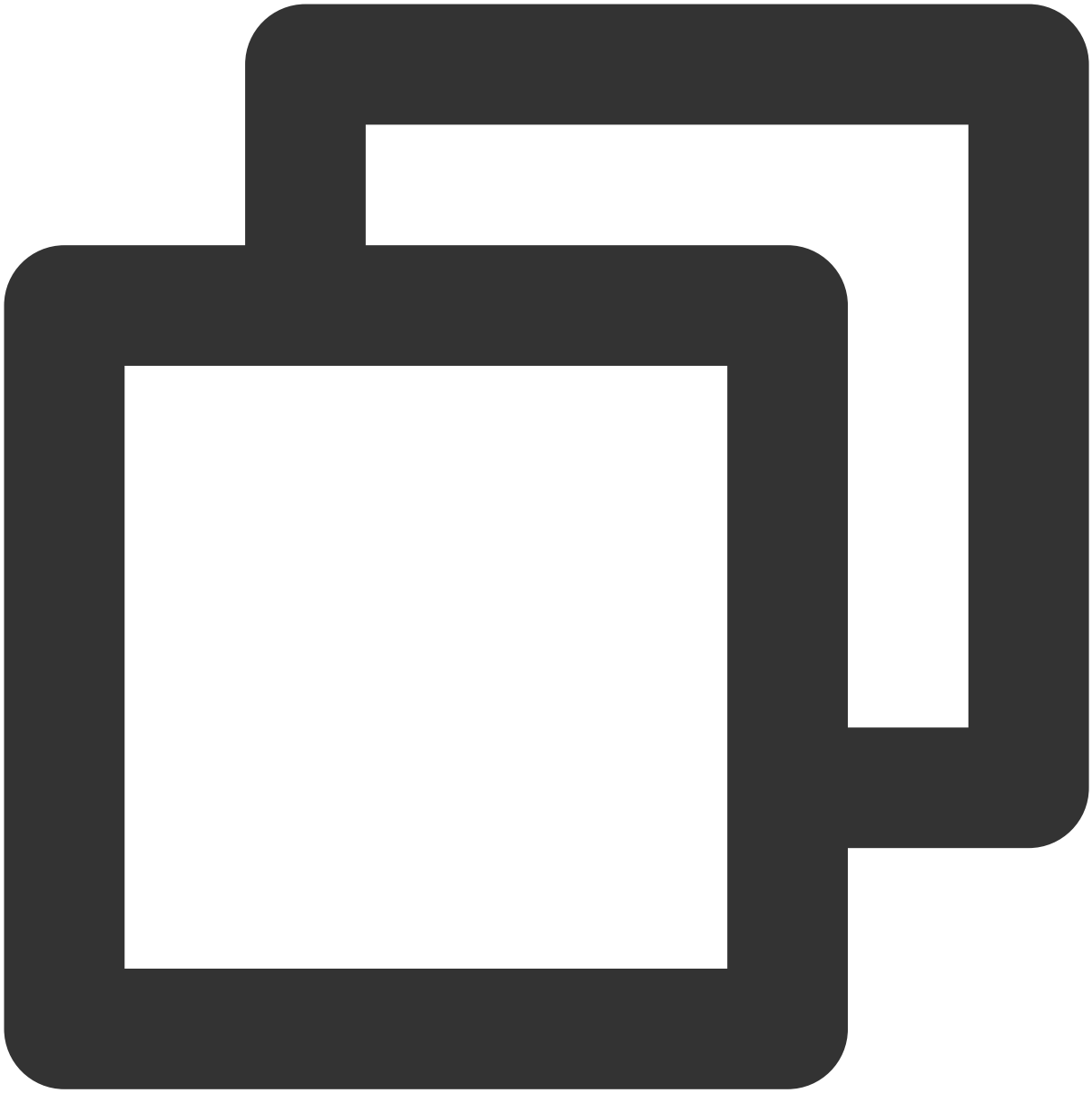
1. 長時間接続コールバックを監視し、セッションリストをリアルタイムに更新します。
2. APIをリクエストし、ページごとに一括でセッションリストを取得します。

一般的なユースケースは次のようなものがあります：

アプリケーションの起動後すぐにセッションリストを取得し、その後は長時間接続を監視して、セッションリストの変更をリアルタイムに更新します。

セッションリストの一括リクエスト

セッションリストを取得するためには `nextSeq` を保守し、現在の位置を記録する必要があります。



```
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";
```

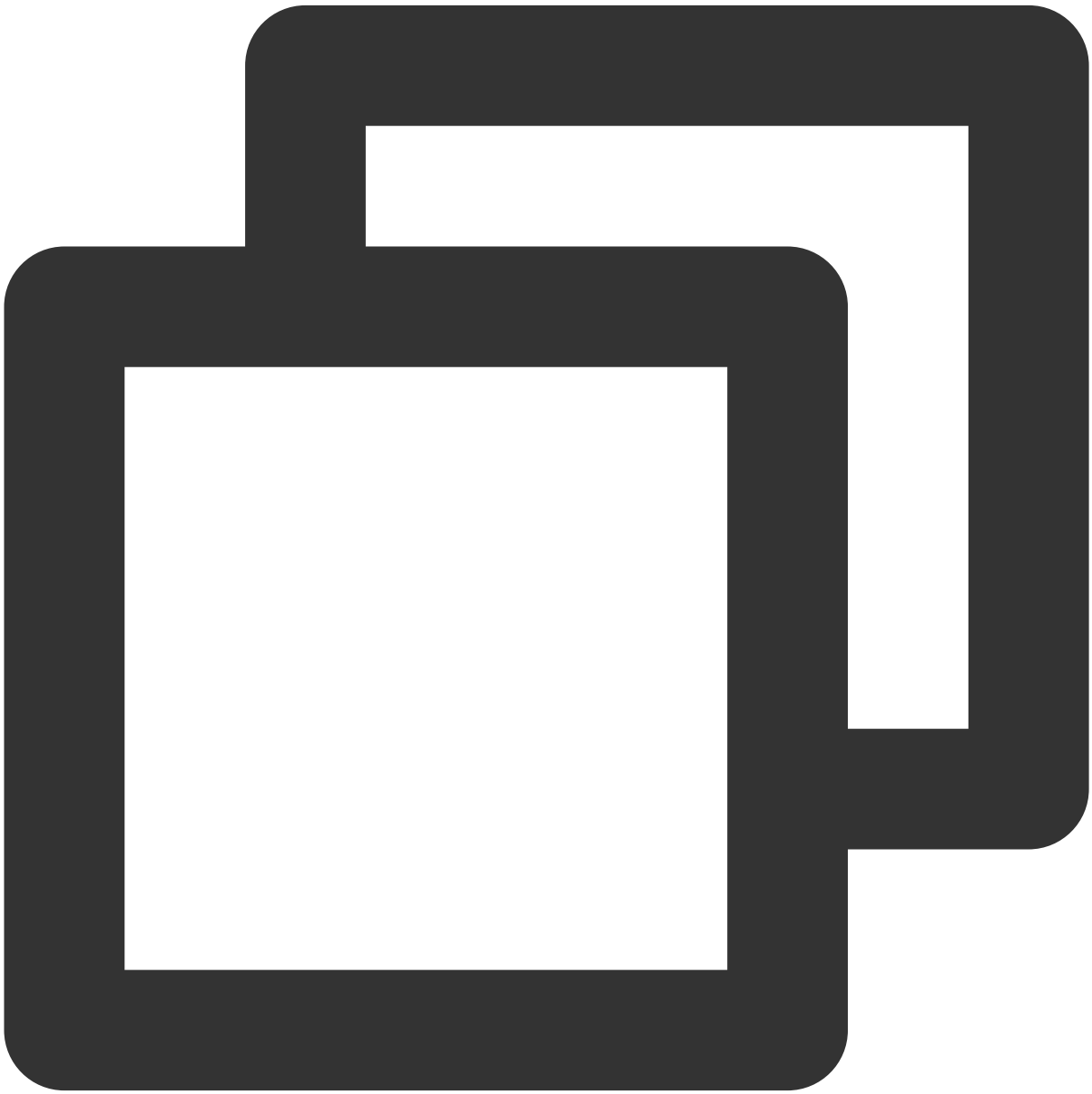
```
}
```

この時点で、前の手順で別のテストアカウントを使用して送信したメッセージのセッションを見ることができるようになりました。

長時間接続の監視によるセッションリストのリアルタイム取得

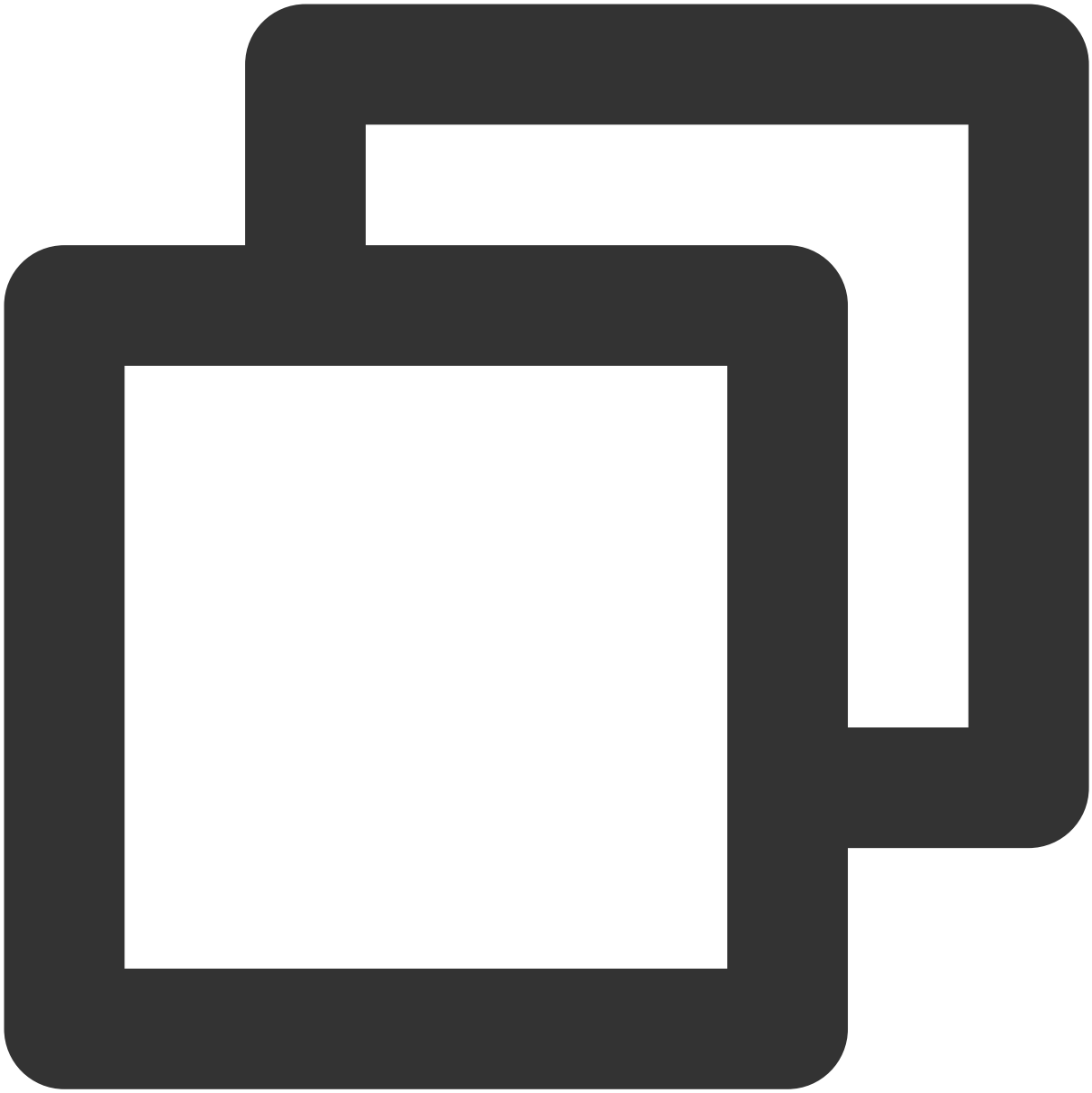
この手順では、先にSDKにリスナーをマウントしてからコールバックイベントを処理し、UIを更新する必要があります。

1. 監視をマウントします。



```
await TencentImSDKPlugin.v2TIMManager
   .getConversationManager()
   .setConversationListener(
     listener: new V2TimConversationListener(
       onConversationChanged: (List<V2TimConversation> list){
         _onConversationListChanged(list);
 },
       onNewConversation:(List<V2TimConversation> list){
         _onConversationListChanged(list);
 },
```

2. コールバックイベントを処理し、最新のセッションリストをインターフェース上に表示します。



```
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]);
  }
   }
```

メッセージの受信

このセグメントの詳細なドキュメント

Tencent Cloud IM Fflitter SDKによるメッセージの受信には2つの方法があります。

1. 長時間接続コールバックを監視し、メッセージの変更をリアルタイムに取得し、メッセージ履歴リストを更新してレンダリングします。

2. APIをリクエストし、ページごとに一括でメッセージ履歴を取得します。

一般的なユースケースは次のようなものです。

1. インターフェースが新しいセッションに入ると、まず一定量のメッセージ履歴を一括でリクエストし、メッセージ履歴リストの表示に用います。

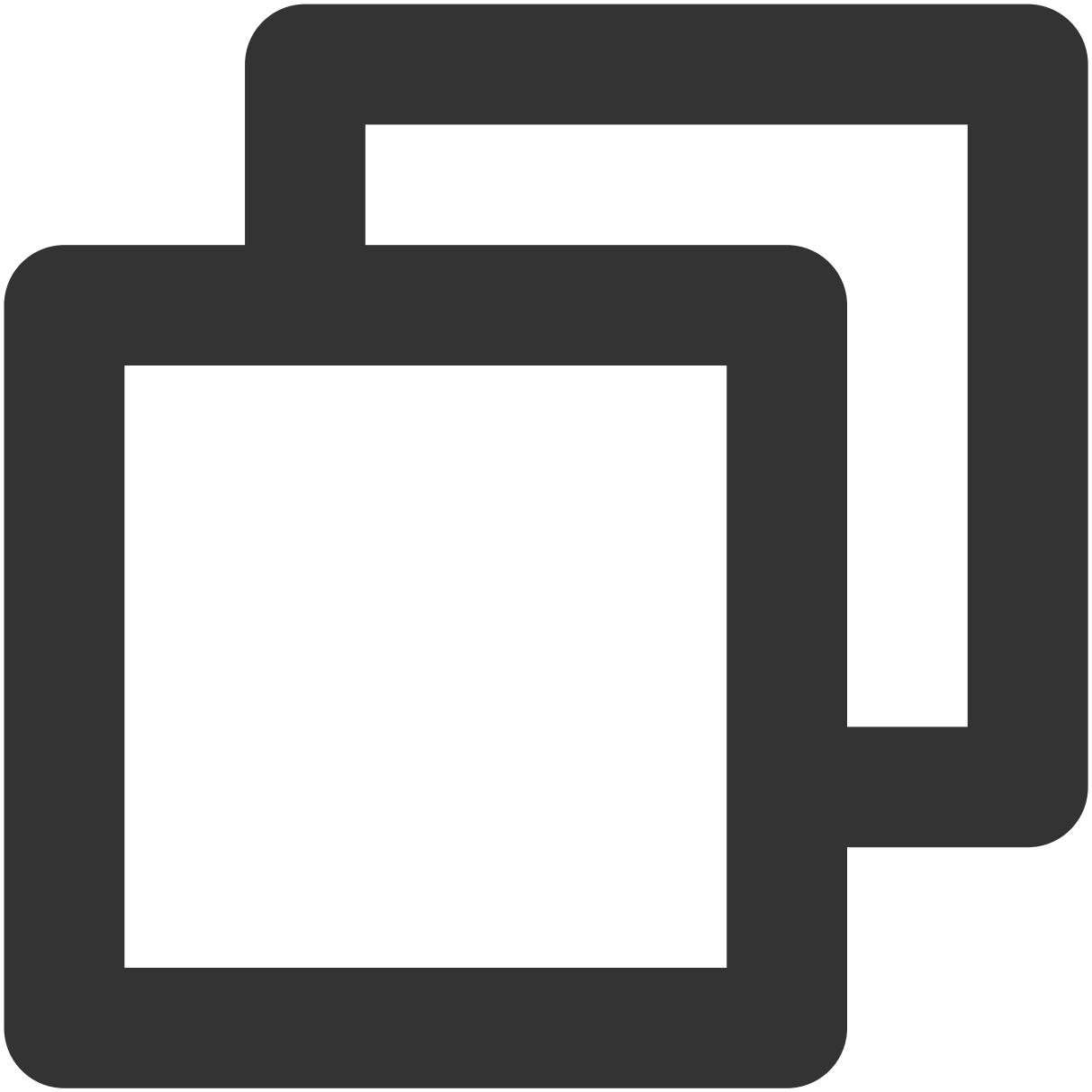
2. 長時間接続を監視し、新しいメッセージをリアルタイムに受信してメッセージ履歴リストに追加します。

メッセージ履歴リストの一括リクエスト

ページごとに取得するメッセージ数が多すぎると取得速度に影響しますので、多すぎてはなりません。20通前後に設定することをお勧めします。

次のリクエストの際に使用できるよう、現在のページ数を動的に記録しなければなりません。

サンプルコードは次のとおりです：



```
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: "",
      );
```

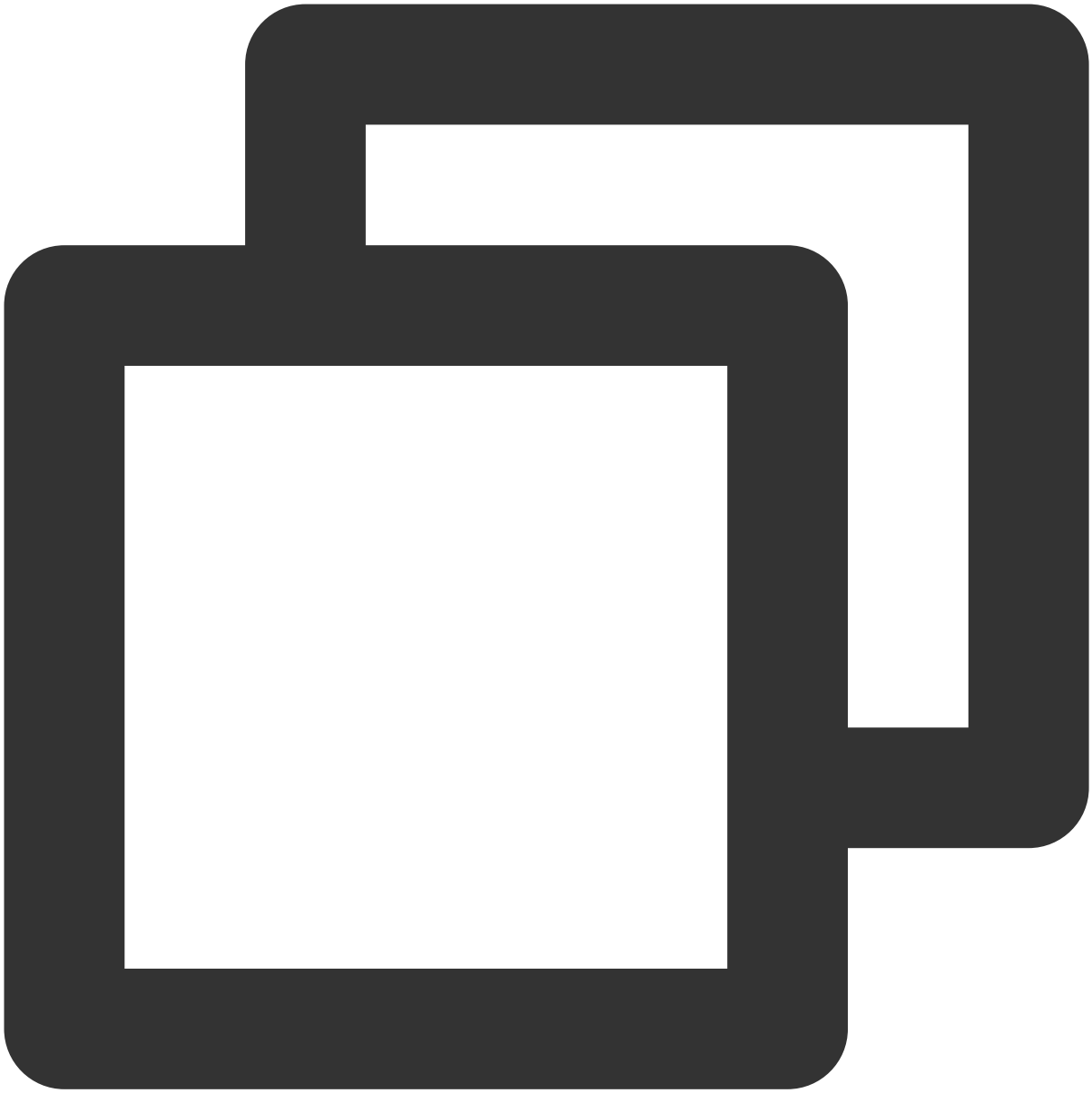
```
    List<V2TimMessage> msgList = res.data ?? [];

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

長時間接続の監視による新メッセージのリアルタイム取得

メッセージ履歴リストを初期化すると、新しいメッセージは長時間接続 `V2TimAdvancedMsgListener.onRecvNewMessage` から受信します。

`onRecvNewMessage` コールバックがトリガーされると、必要に応じて新しいメッセージをメッセージ履歴リストに追加することができます。

リスナーをバインドするサンプルコードは次のとおりです：



```
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);
```

この時点で、IMモジュールの開発は基本的に完了し、メッセージの送受信や様々なセッションに入ることが可能になりました。
続いて、グループ、ユーザー個人情報、リレーションシップチェーン、オフラインプッシュ、ローカル検索などの関連機能の開発を完了することができます。
詳細については、UI統合SDKの自己実装ドキュメントをご確認ください。

高度な機能統合

その他のプラグインを使用してFlutter IM使用体験を強化

SDKとTUIKitの基本的な機能に加えて、IM機能を強化するのに役立つ4つのオプションのプラグインも提供しています。
メッセージプッシュプラグイン：メーカーのネイティブオフラインプッシュ機能およびオンラインプッシュ機能をサポートし、その他の業務メッセージのプッシュもサポートし、メッセージの到達率を向上させることができます。
統合コードについては、Demoをご参照ください。
説明：
良いアイデアや提案があれば、気軽にお問い合わせください。

その他のプラットフォームを展開

Tencent Cloud IM for Flutter関連SDKは、デフォルトでAndroid / iOSプラットフォームをサポートします。その他のプラットフォーム(Web/Desktop)を展開するには、この部分をご参照ください。

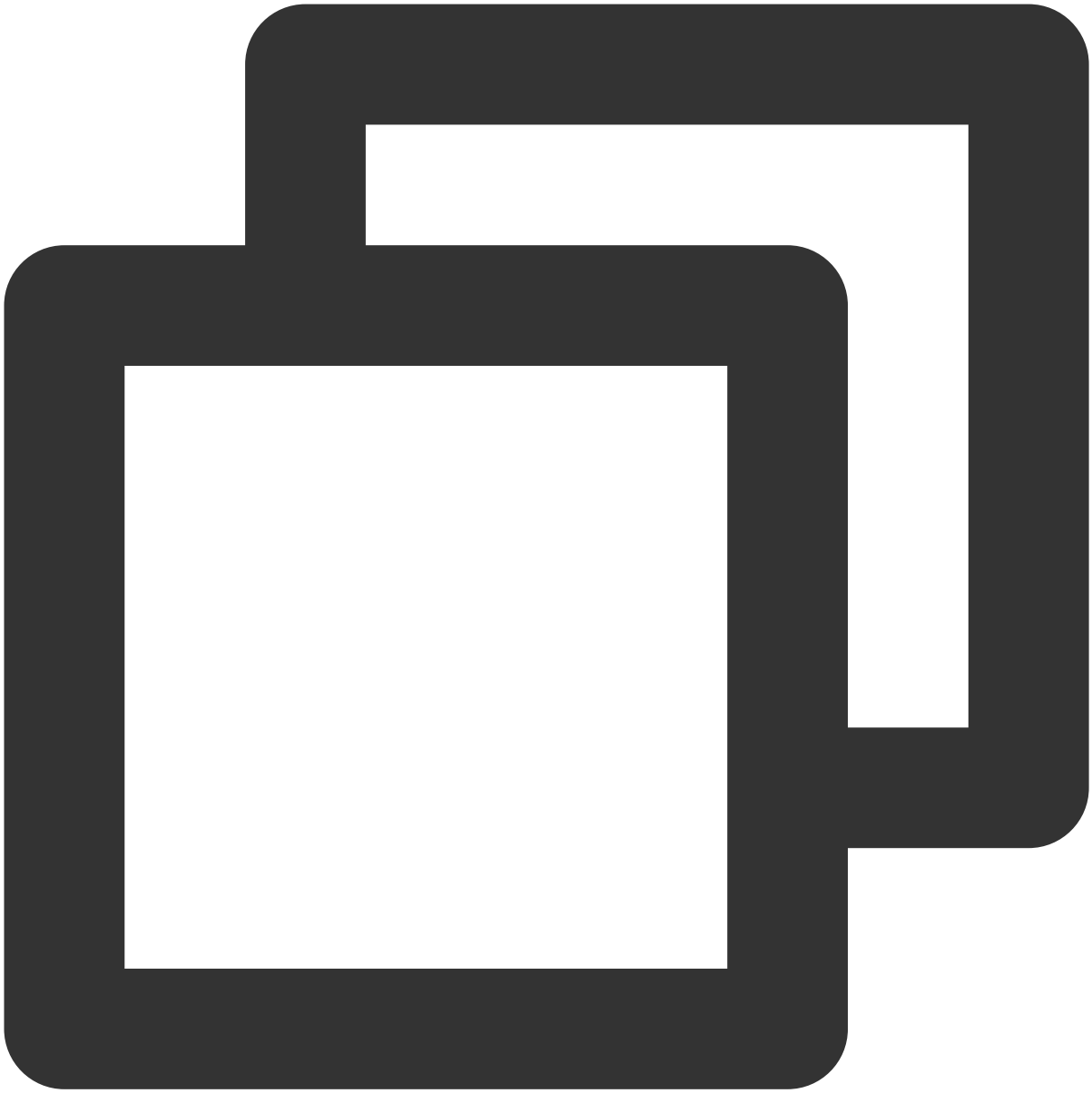
Flutter for Webサポート

SDK、TUIKit(tencent_cloud_chat_uikit) 0.1.5バージョンは、UIなし SDK(tencent_cloud_chat_sdk) 4.1.1+2とそれ以降のバージョンは、Webとの完全な互換性があります。
AndroidとiOS端末と比べて、いくつかの追加手順が必要です。次のとおりです：

Flutter 3.xバージョンをアップグレード

Flutter 3.xバージョンは、Web性能をさらに最適化します。これを使用してFlutter Webプロジェクトを開発することを強くお勧めします。

Flutter for Webを導入してSDKを追加

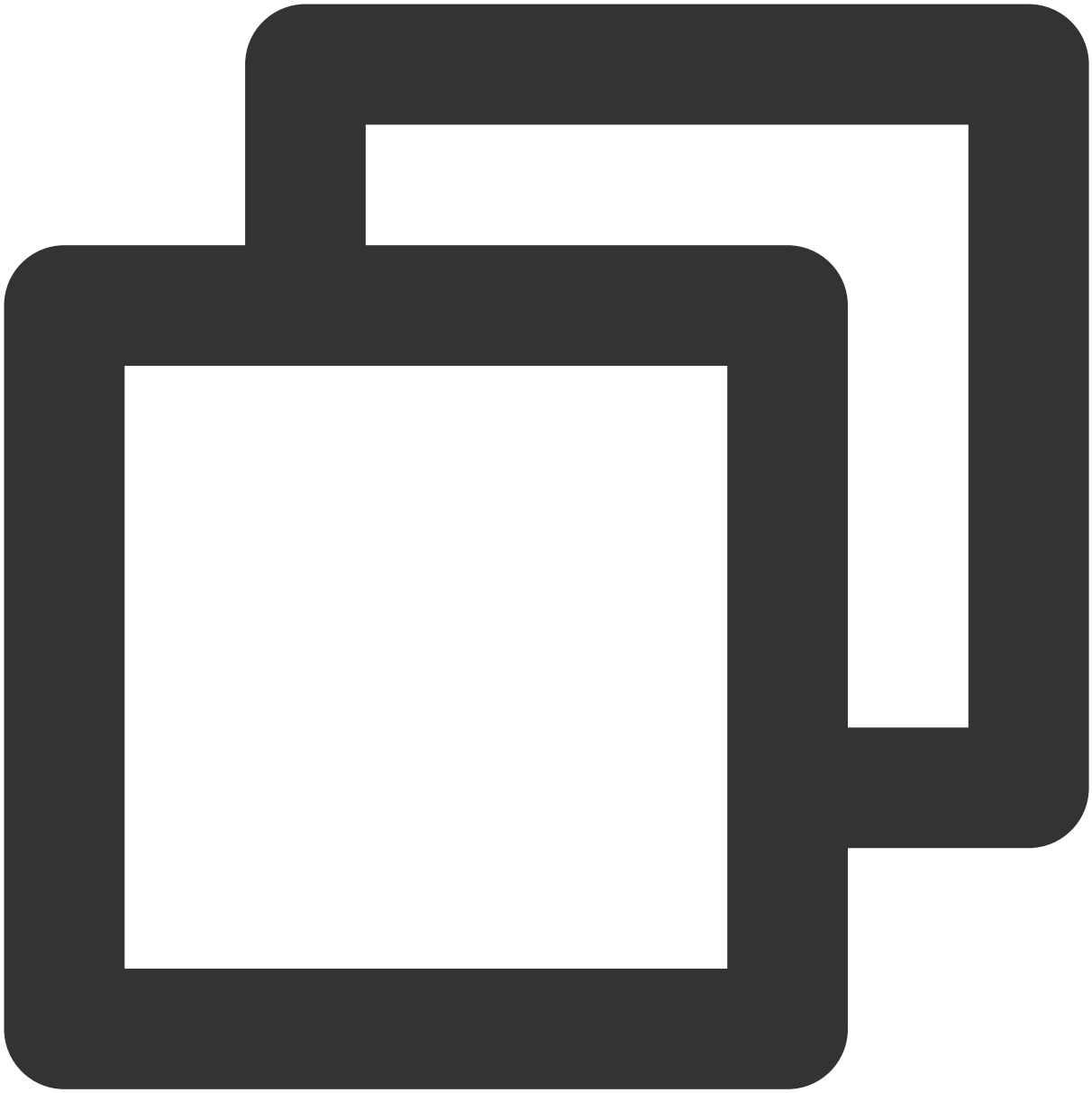


```
flutter pub add tencent_im_sdk_plugin_web
```

JSを導入

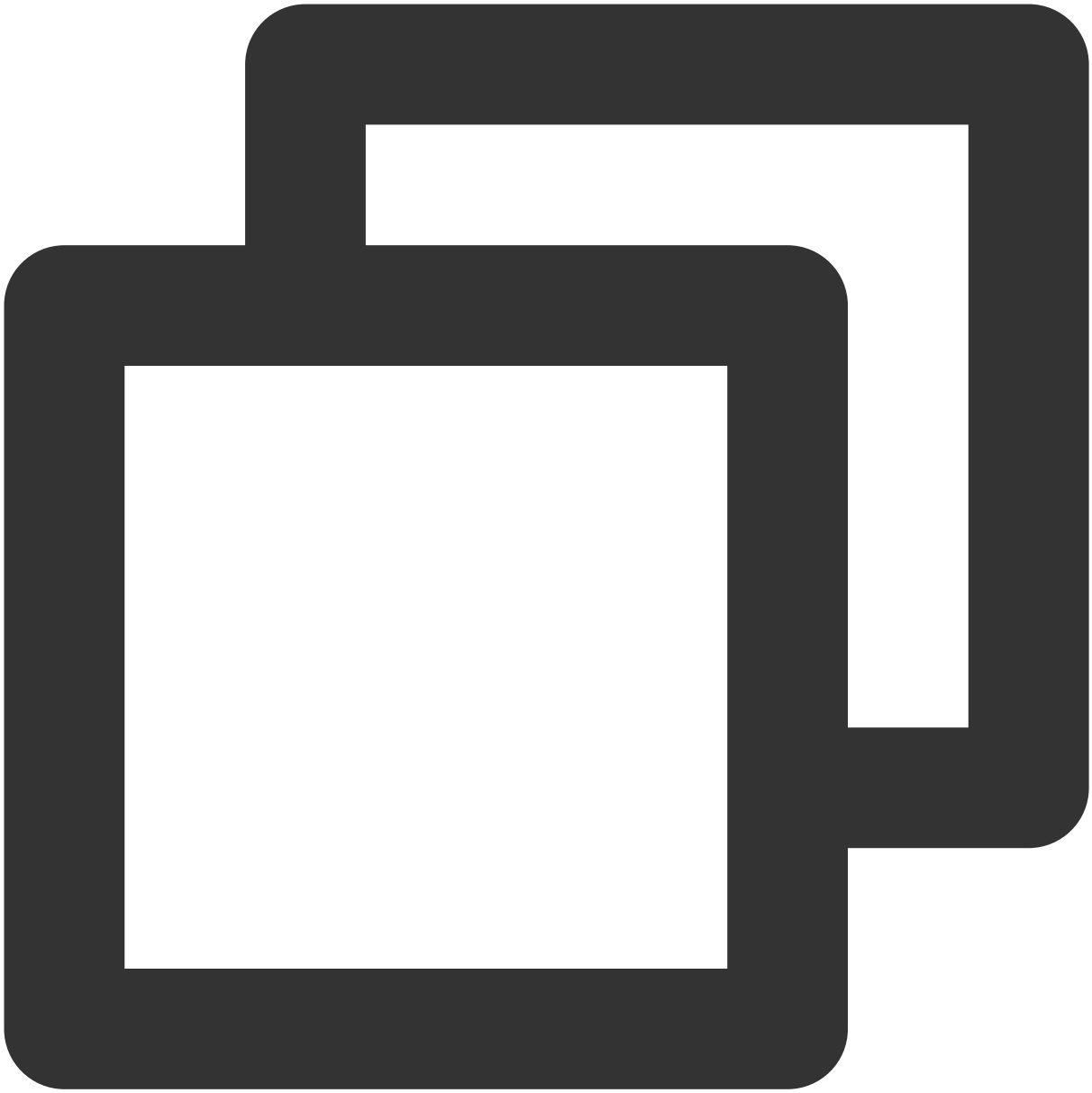
説明：

既存のFlutterプロジェクトがWebをサポートしない場合は、ルートディレクトリで `flutter create .` を実行してWebサポートを追加してください。
プロジェクトの `web/` に進み、 `npm` または `yarn` を使用して関連するJSの依存関係をインストールします。
プロジェクトを初期化するときは、画面指示に従って進めてください。

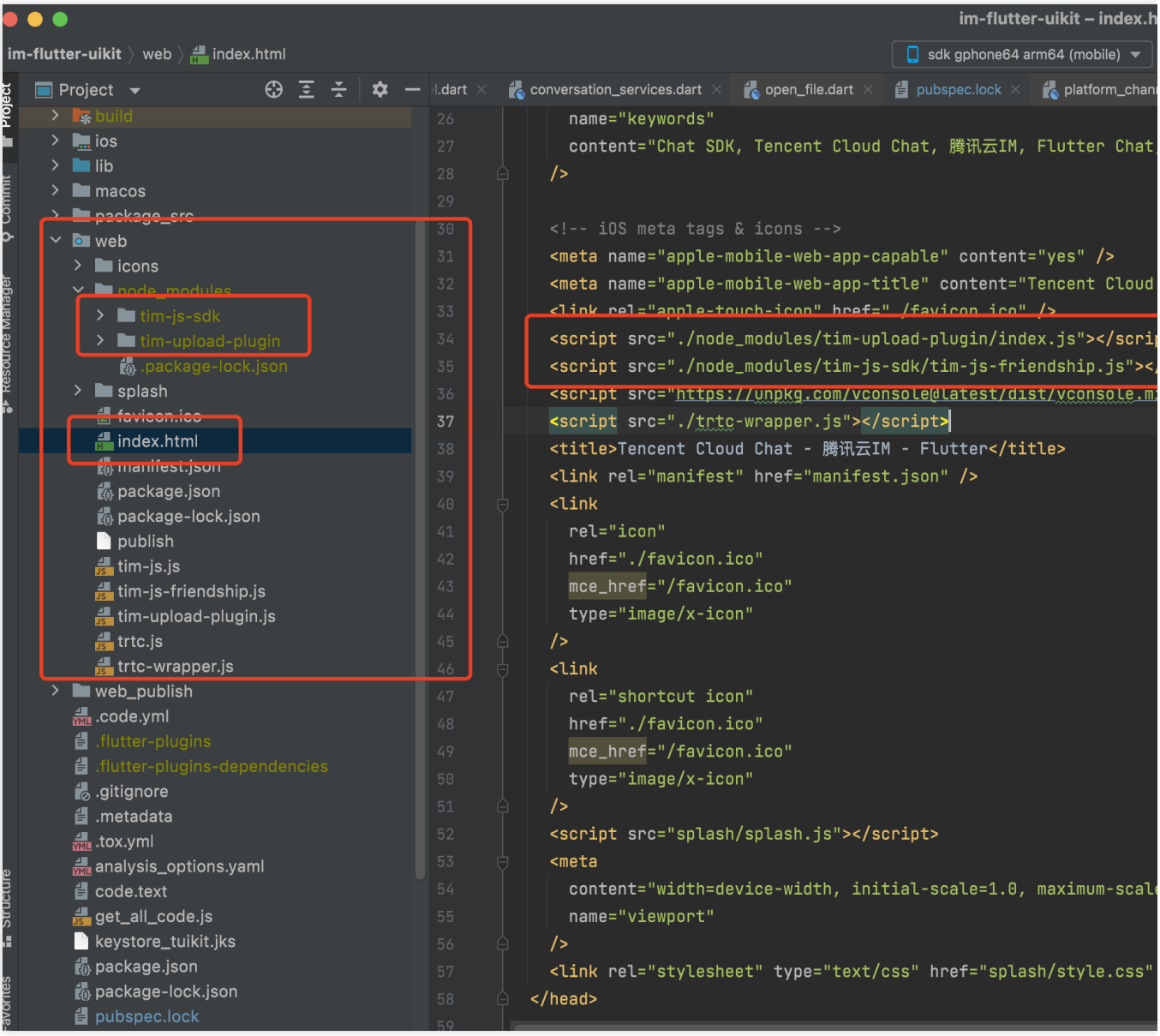


```
cd web

npm init

npm i tim-js-sdk

npm i tim-upload-plugin
```

`web/index.html` を開き、 `<head> </head>` にJSファイルを導入します。次のとおりです。



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



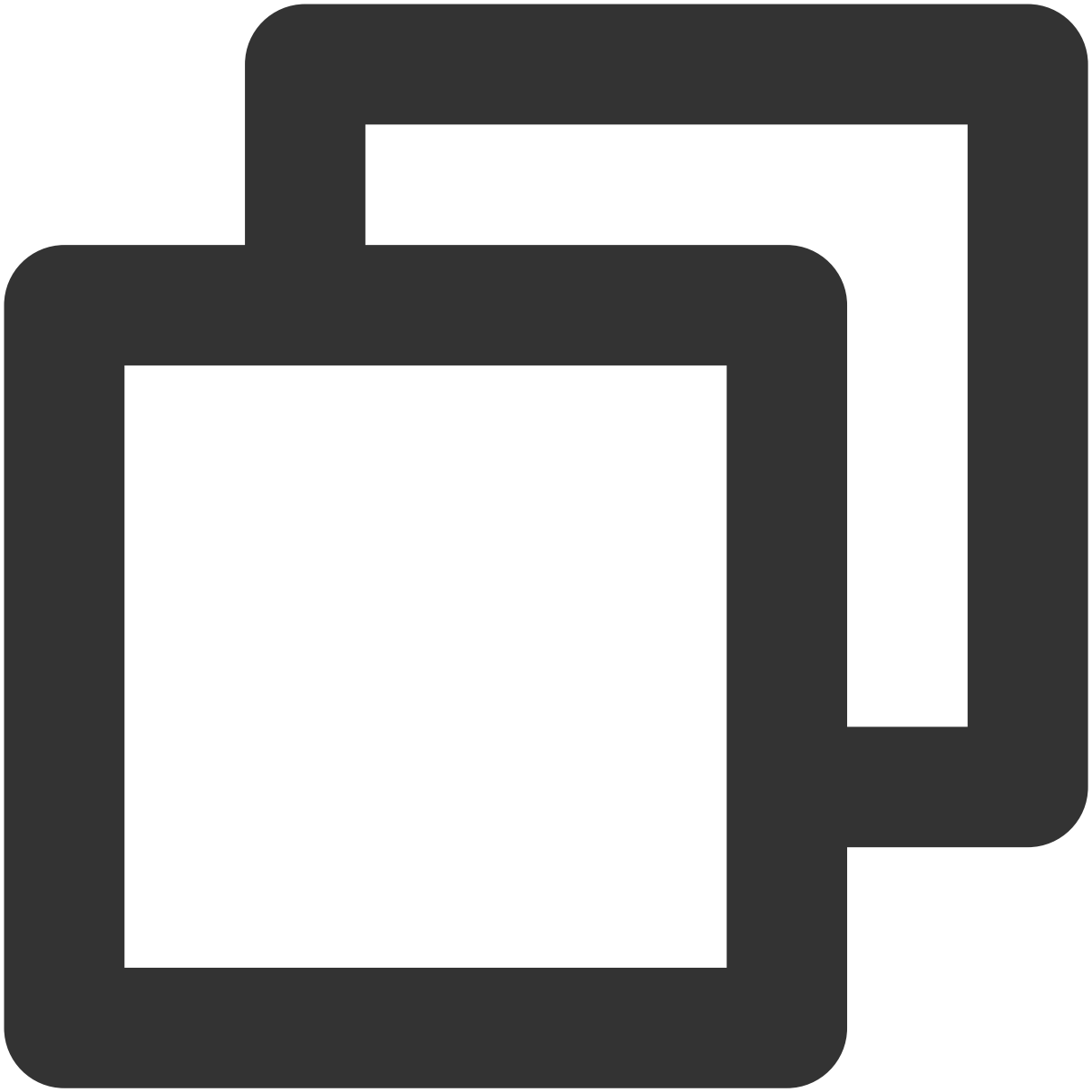
Flutter for Desktop(PC) サポート

UIなしSDK(tencent_cloud_chat_sdk) 4.1.9とそれ以降のバージョンは、macOS、Windows端末との完全な互換性があります。AndroidとiOS端末と比べて、いくつかの追加手順が必要です。次のとおりです：

Flutter 3.xバージョンをアップグレード

Flutter 3.0以降のバージョンにのみ、desktop端末のパッケージが適用されるため、使用する必要がある場合は、Flutter 3.xバージョンにアップグレードしてください。

Flutter for Desktopを導入してSDKを追加

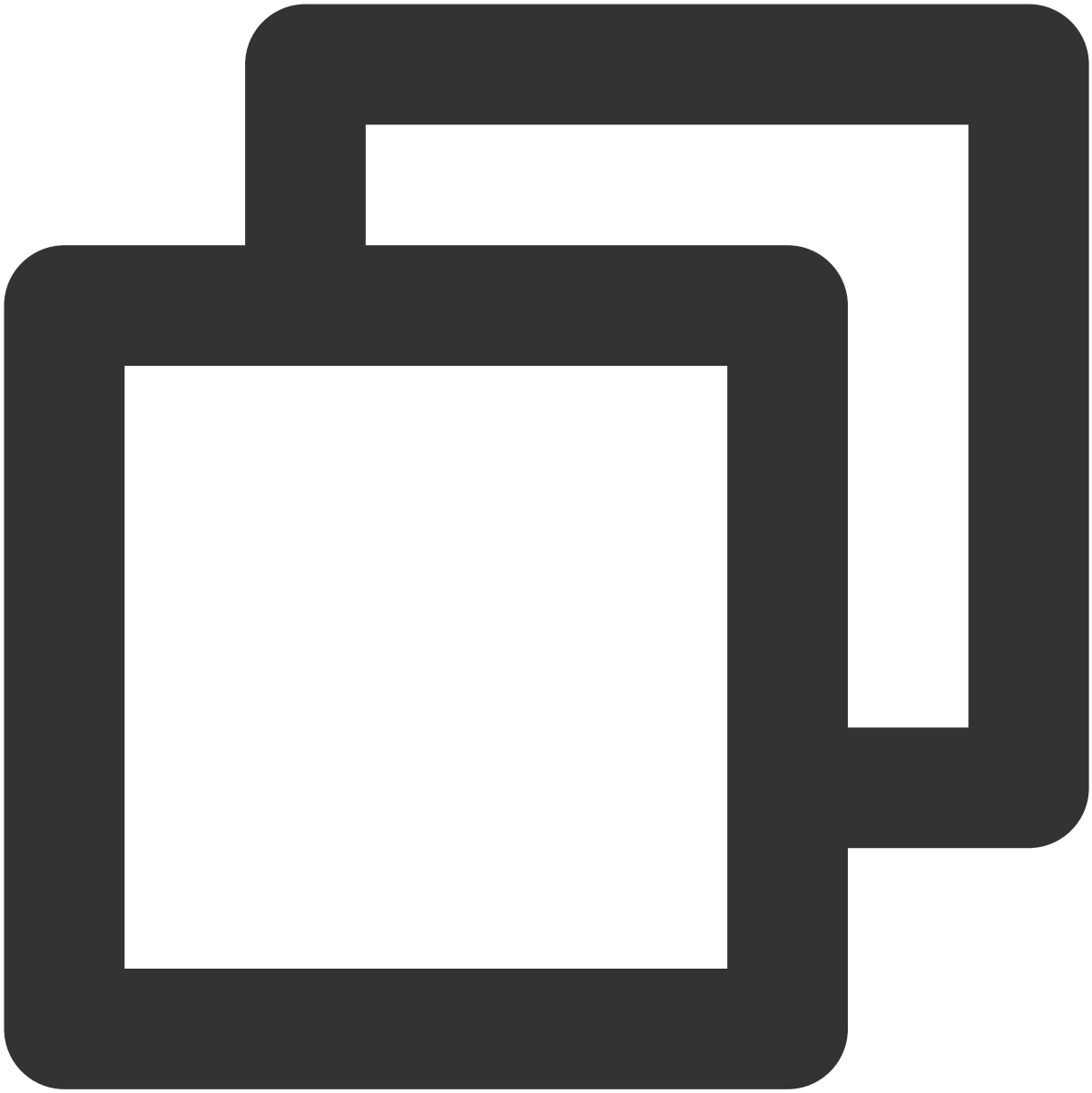


```
flutter pub add tencent_im_sdk_plugin_desktop
```

macOS修正

`macos/Runner/DebugProfile.entitlements` ファイルを開きます。

`<dict></dict>` に次の `key-value` キーと値のペアを追加します。

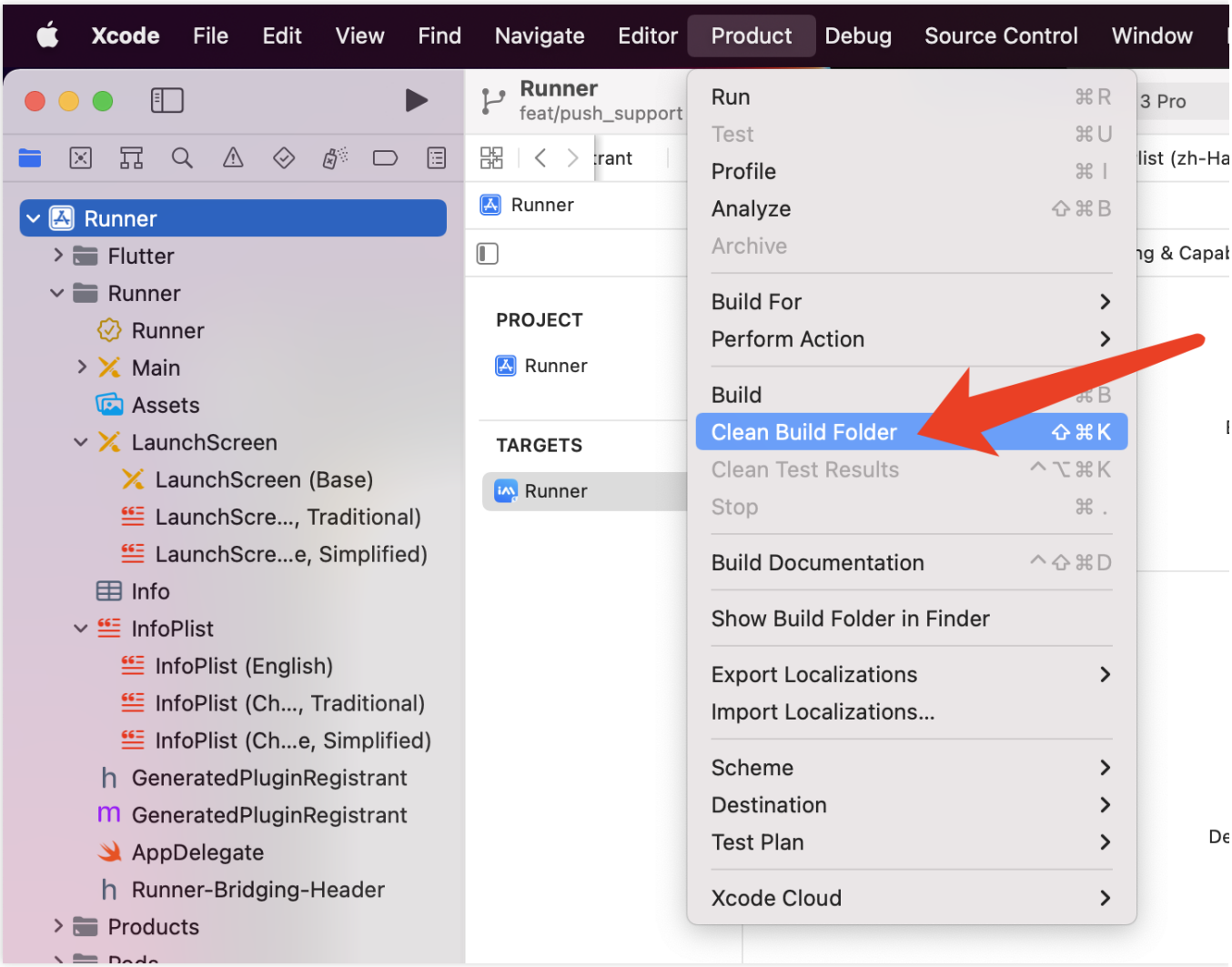


```
<key>com.apple.security.app-sandbox</key>
<false/>
```

よくあるご質問

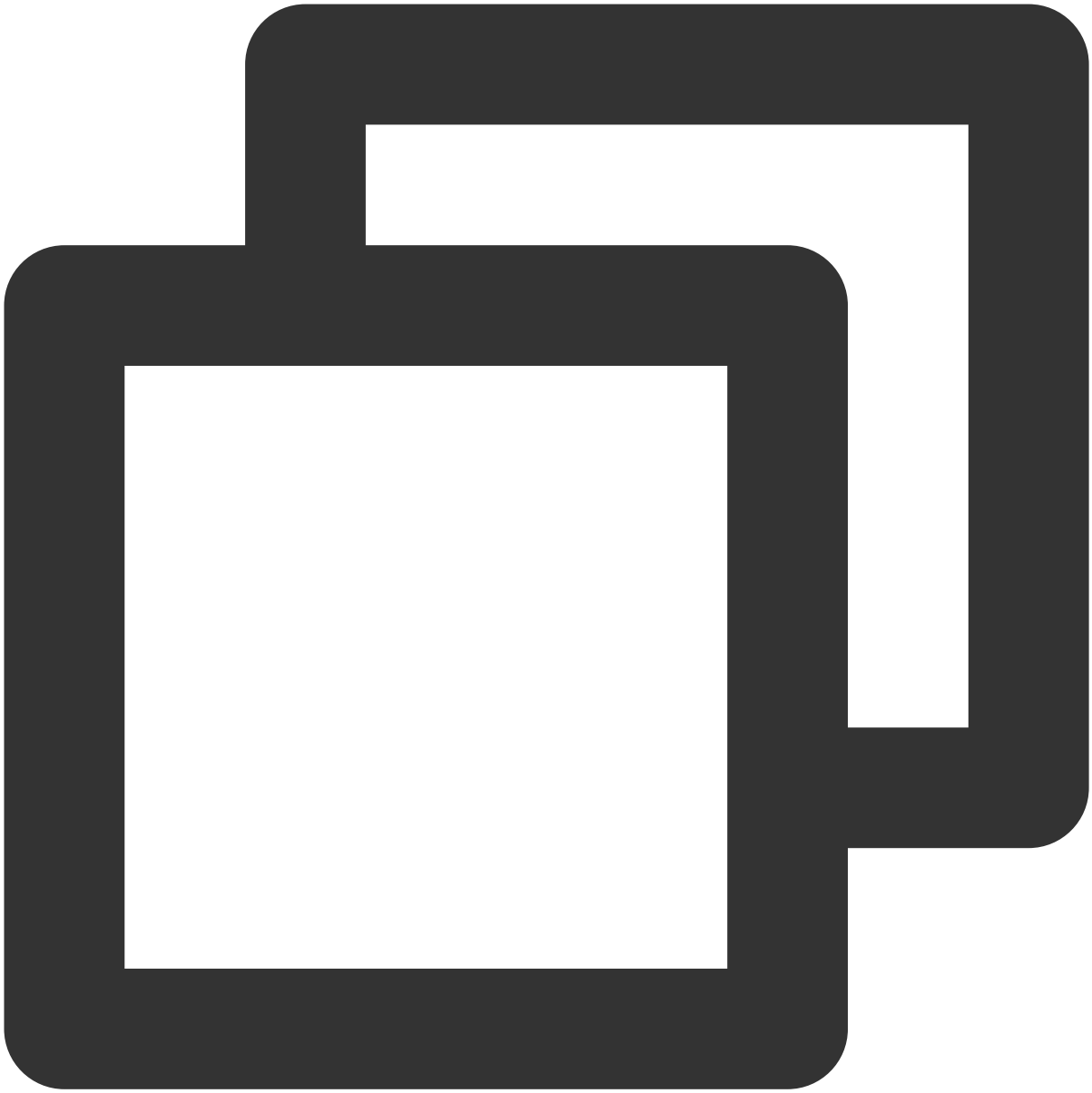
iOS側のPod依存関係を正常にインストールできません

試行ソリューション1：設定を実行した後、エラーが発生した場合は、**Product > Clean Build Folder**をクリックし、生成物を削除してから再度 `pod install` または `flutter run` を行うことができます



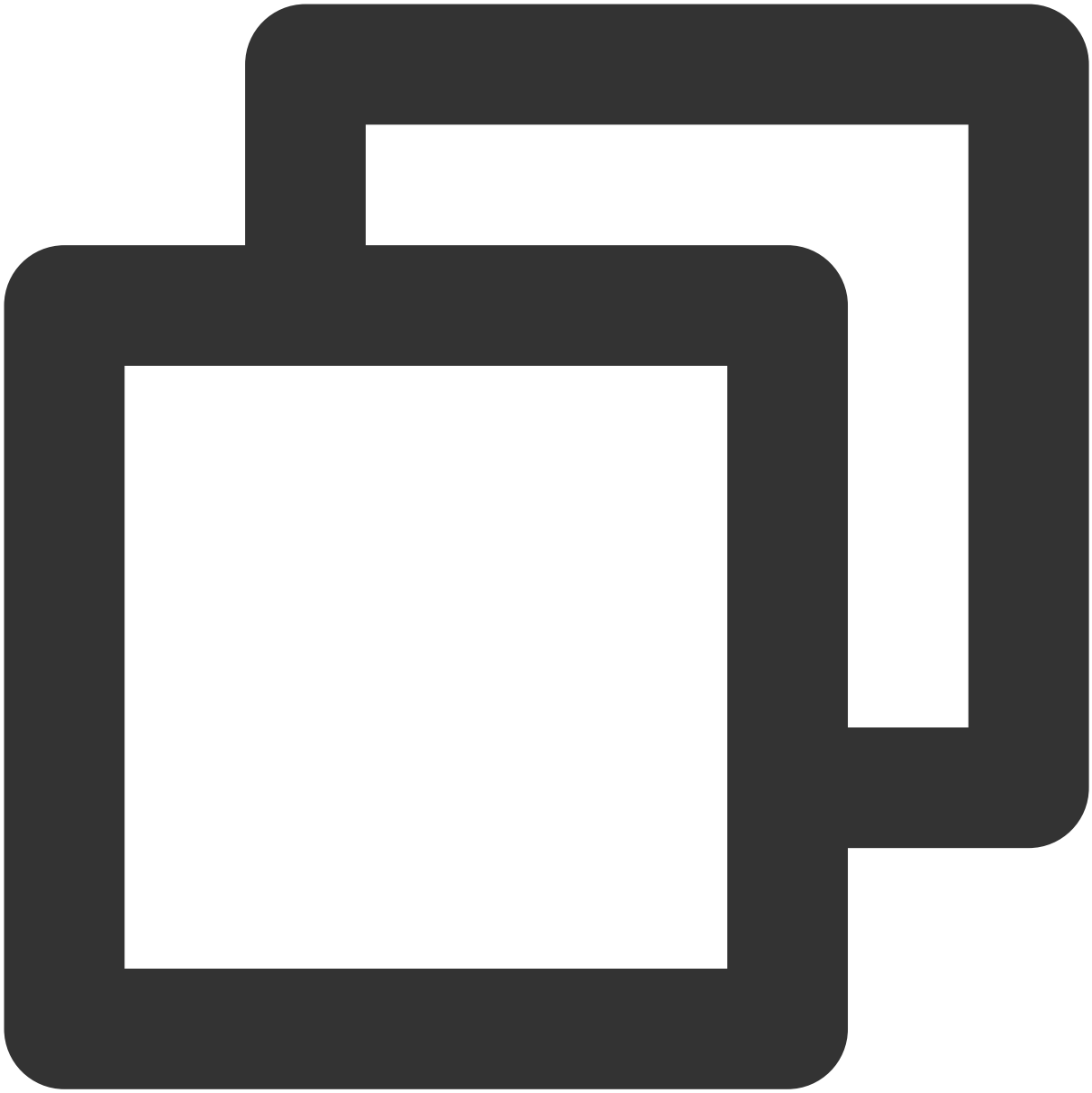
試行ソリューション2： `ios/Pods` フォルダおよび `ios/Podfile.lock` ファイルを手動で削除し、次のコマンドを実行して、依存関係を再インストールします

1. M1など、新規Apple SiliconのMacデバイスを搭載します。



```
cd ios
sudo arch -x86_64 gem install ffi
arch -x86_64 pod install --repo-update
```

2. 古いIntelチップのMacデバイスを搭載します。



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

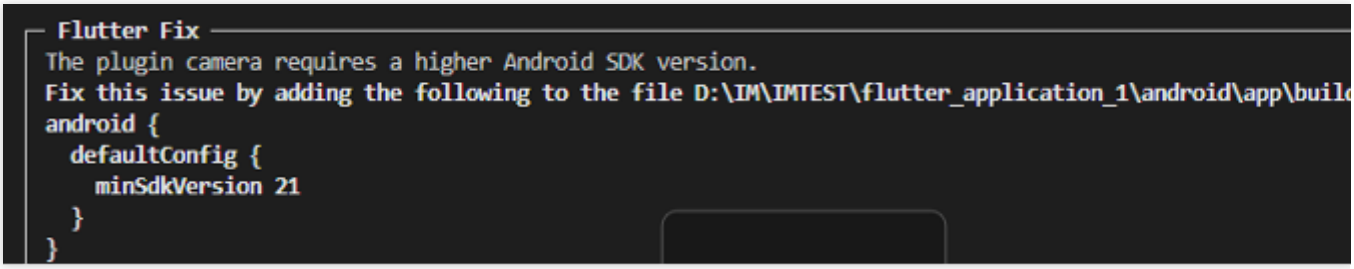
Apple Watchを装着した際、 実機デバッグでiOSのエラーが発生します

Apple Watchを機内モードに設定し、iPhoneのBluetooth機能を `設定 => Bluetooth` で完全にオフにしてください。
Xcodeを再起動し（オンになっている場合）、再び `flutter run` を実行します。

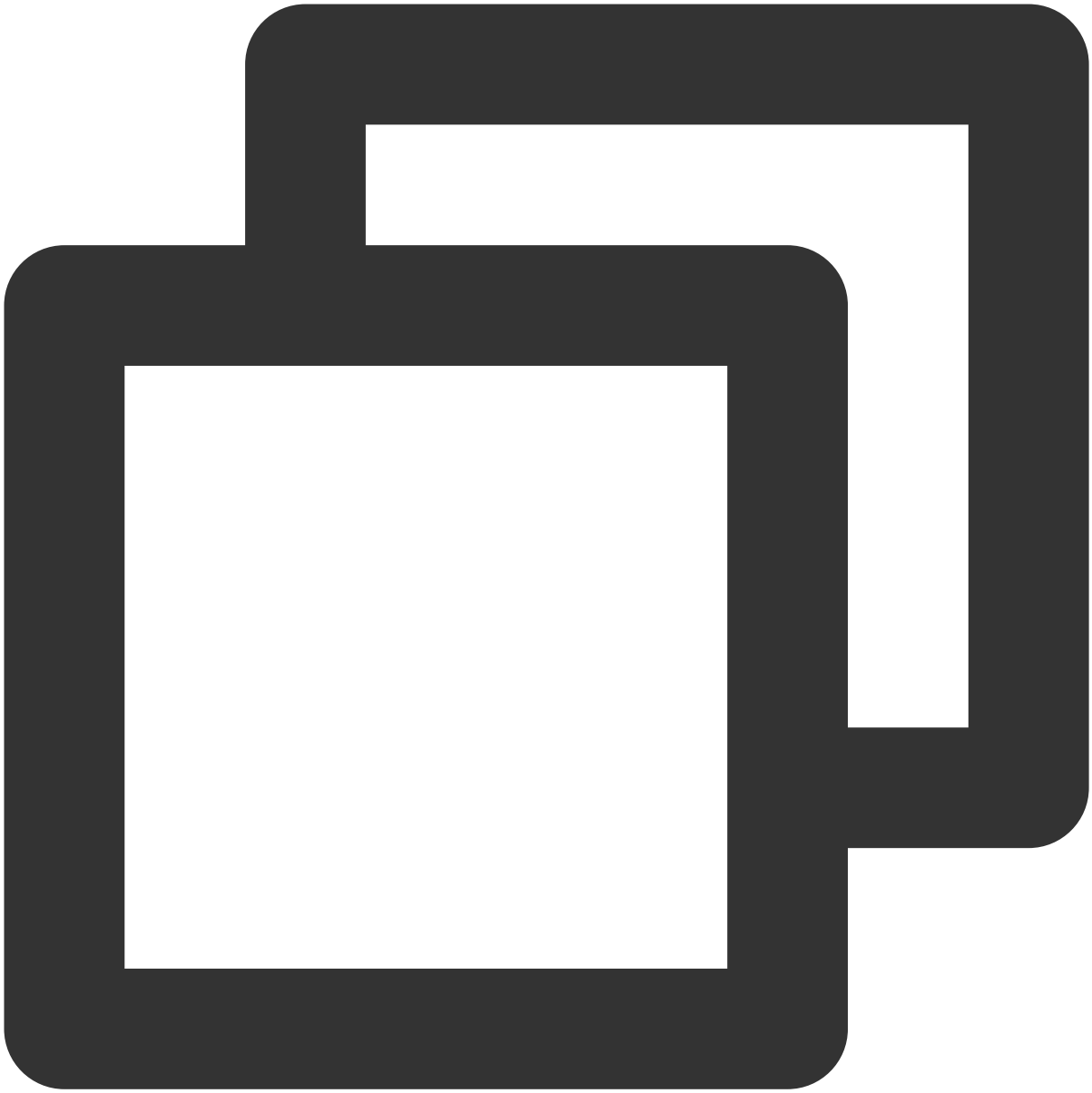
Flutter環境の問題

Flutterの環境に問題のないことを確認する場合、Flutter doctorを実行してFlutter環境が適切にインストールされているかチェックしてください。

Flutterを使用して自動生成したプロジェクトをTUIKitにインポートし、Android端末で実行するとエラーが発生します

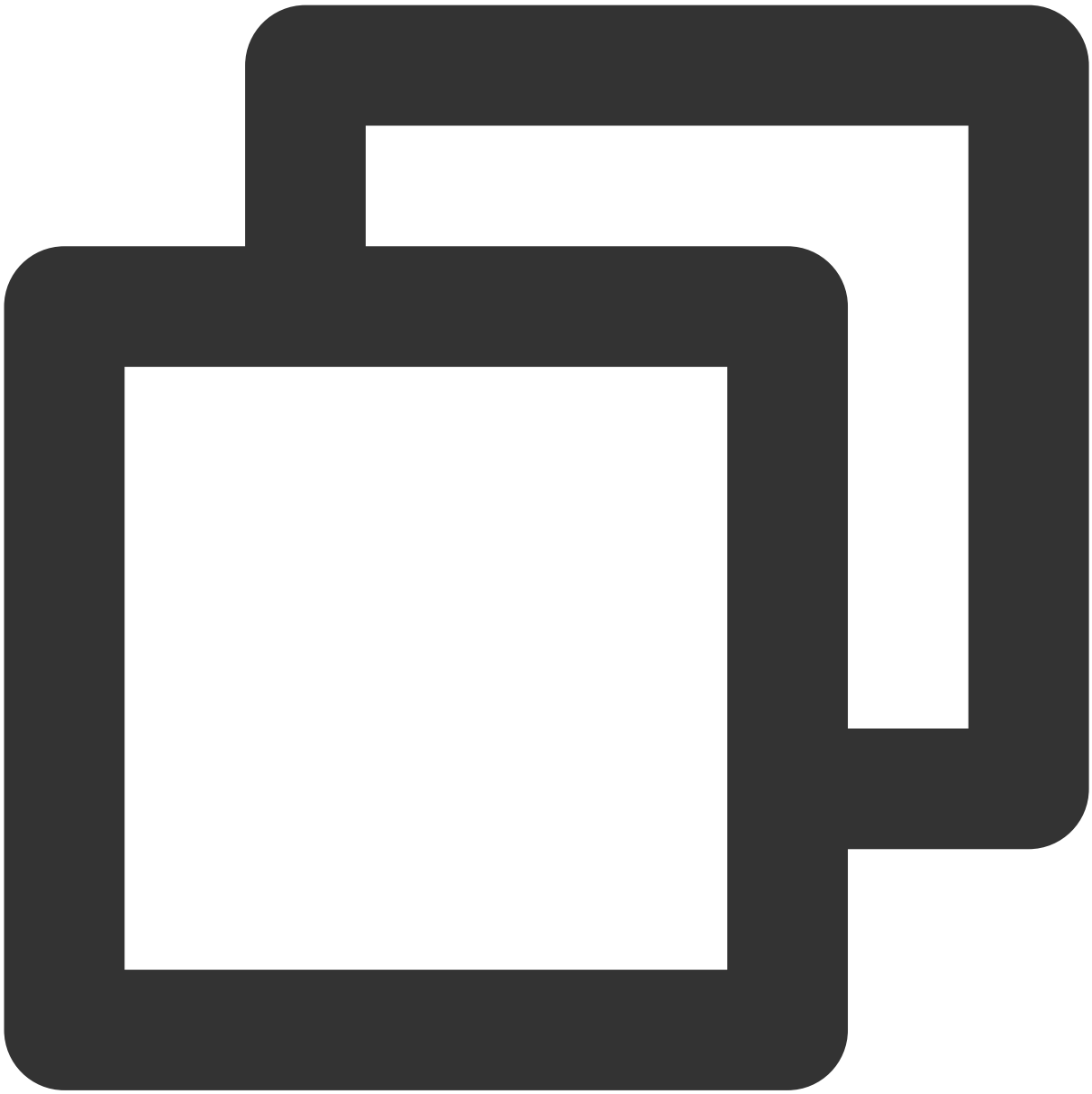


1. `android\\app\\src\\main\\AndroidManifest.xml` を開き、下記に従って、 `xmlns:tools="http://schemas.android.com/tools"` / `android:label="@string/android_label"` および `tools:replace="android:label"` への入力を補完します。



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="Android端末のパッケージ名に置き換える"
 xmlns:tools="http://schemas.android.com/tools">
 <application
     android:label="@string/android_label"
     tools:replace="android:label"
     android:icon="@mipmap/ic_launcher" // iconパスを指定する
     android:usesCleartextTraffic="true"
     android:requestLegacyExternalStorage="true">
```

2. `android\\app\\build.gradle` を開き、 `defaultConfig` の `minSdkVersion` および `targetSdkVersion` への入力を補完します。



```
defaultConfig {
  applicationId "" // Android端末のパッケージ名に置き換える
  minSdkVersion 21
  targetSdkVersion 30
}
```

エラーコードのクエリー方法

IM SDKのAPIレベルのエラーコードについては、このドキュメントをご確認ください。
TUIKitのシナリオコードは、インターフェースポップアップ表示に使用されます。onTUIKitCallbackListener監視を通じて取得します。すべてのシナリオコードリストについては、このドキュメントをご確認ください。

クイックスタート （React Native）

最終更新日：：2024-04-11 16:20:42

ここでは、主にTencent CloudのInstant Messaging（IM）Demo（React Native）を素早く実行する方法をご紹介します。

環境要件

	バージョン
React Native	0.63.4バージョン以降
Android	Android Studio 3.5以降のバージョン。Appの要件はAndroid 4.1以降のバージョンのデバイスです。
iOS	Xcode 11.0以降のバージョン。プロジェクトに有効な開発者署名を設定済みであることを確認してください。

前提条件

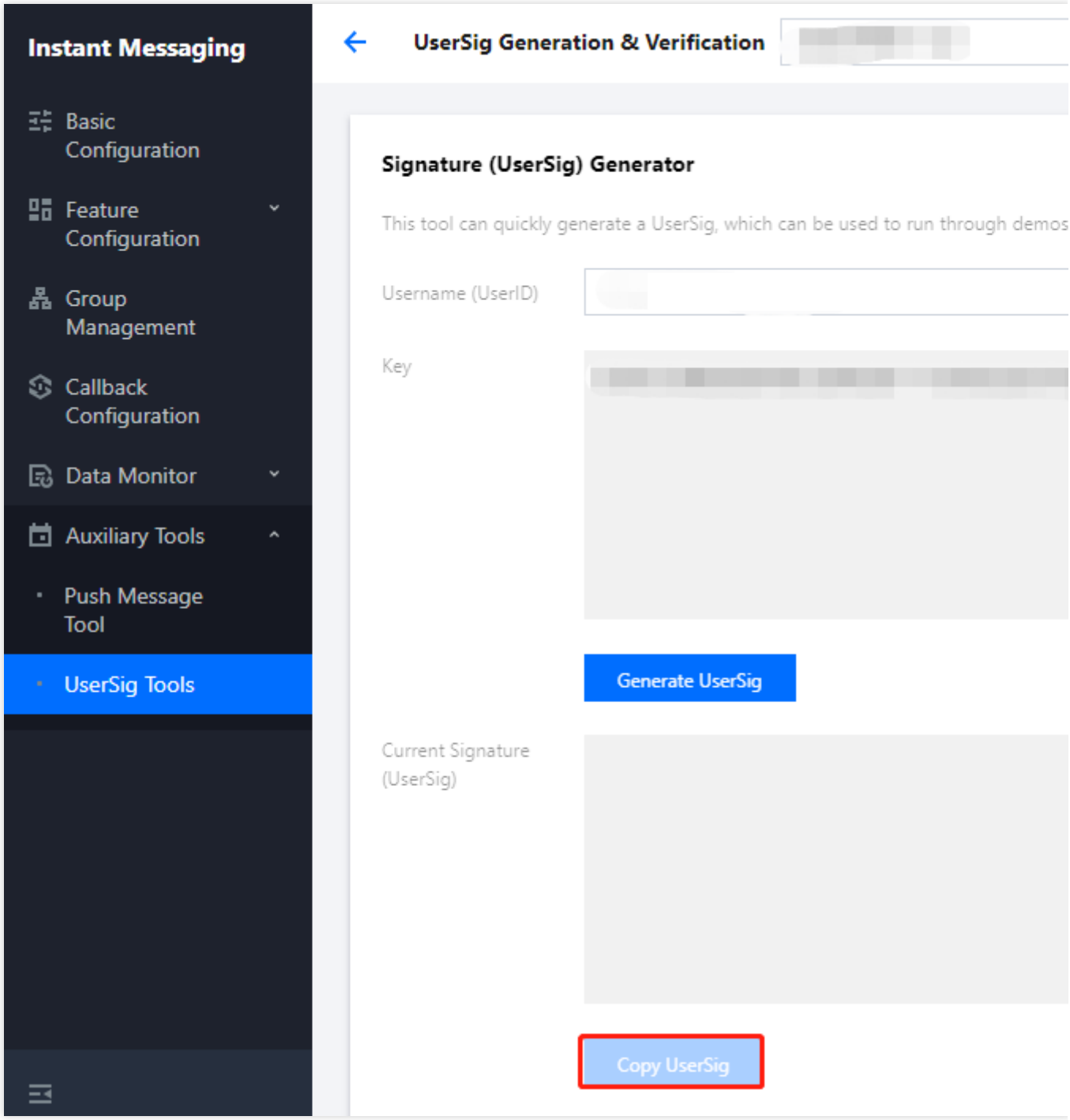
Tencent Cloudアカウントの登録を行い、実名認証が完了済みであること。

その1：テストユーザーの作成

IMコンソールでご自分のアプリケーションを選択し、左側ナビゲーションバーで**支援ツール->UserSig生成&検証**の順にクリックし、2つのUserIDおよびそれに対応するUserSigを作成します。`UserID`、`署名（Key）`、`UserSig`の3つをコピーし、その後のログインで使用します。

説明：

このアカウントは開発テストのみに使用します。アプリケーションのリリース前の`UserSig`の正しい発行方法は、サーバーで生成し、Appのインターフェース向けに提供する方法となります。`UserSig`が必要なときは、Appから業務サーバーにリクエストを送信し動的に`UserSig`を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。



その2：React Native SDKの統合

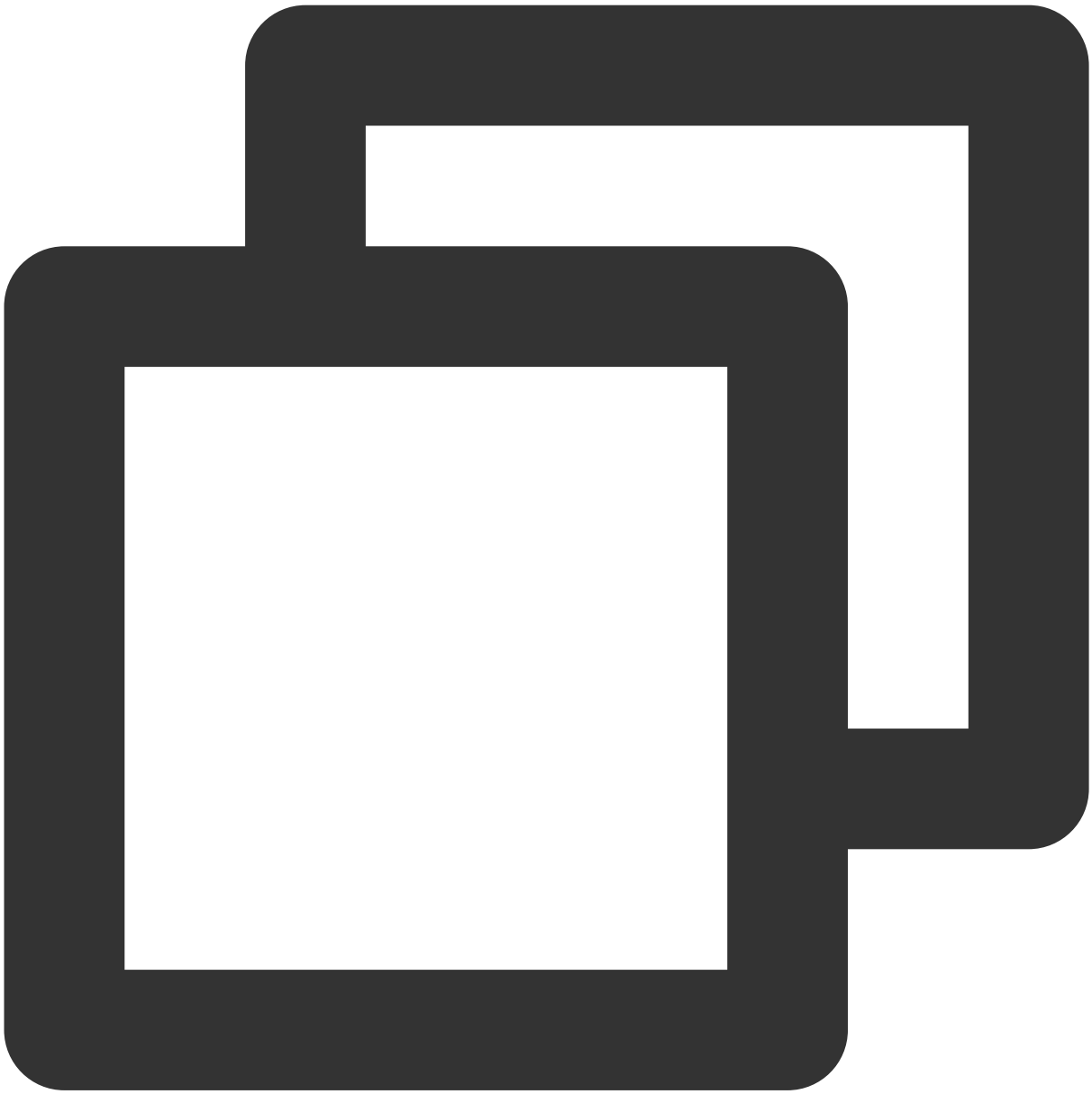
前提条件

React Nativeプロジェクトの作成が完了しているか、またはベースになるReact Nativeプロジェクトがすでにあること。

統合の手順

IM SDKのインストール

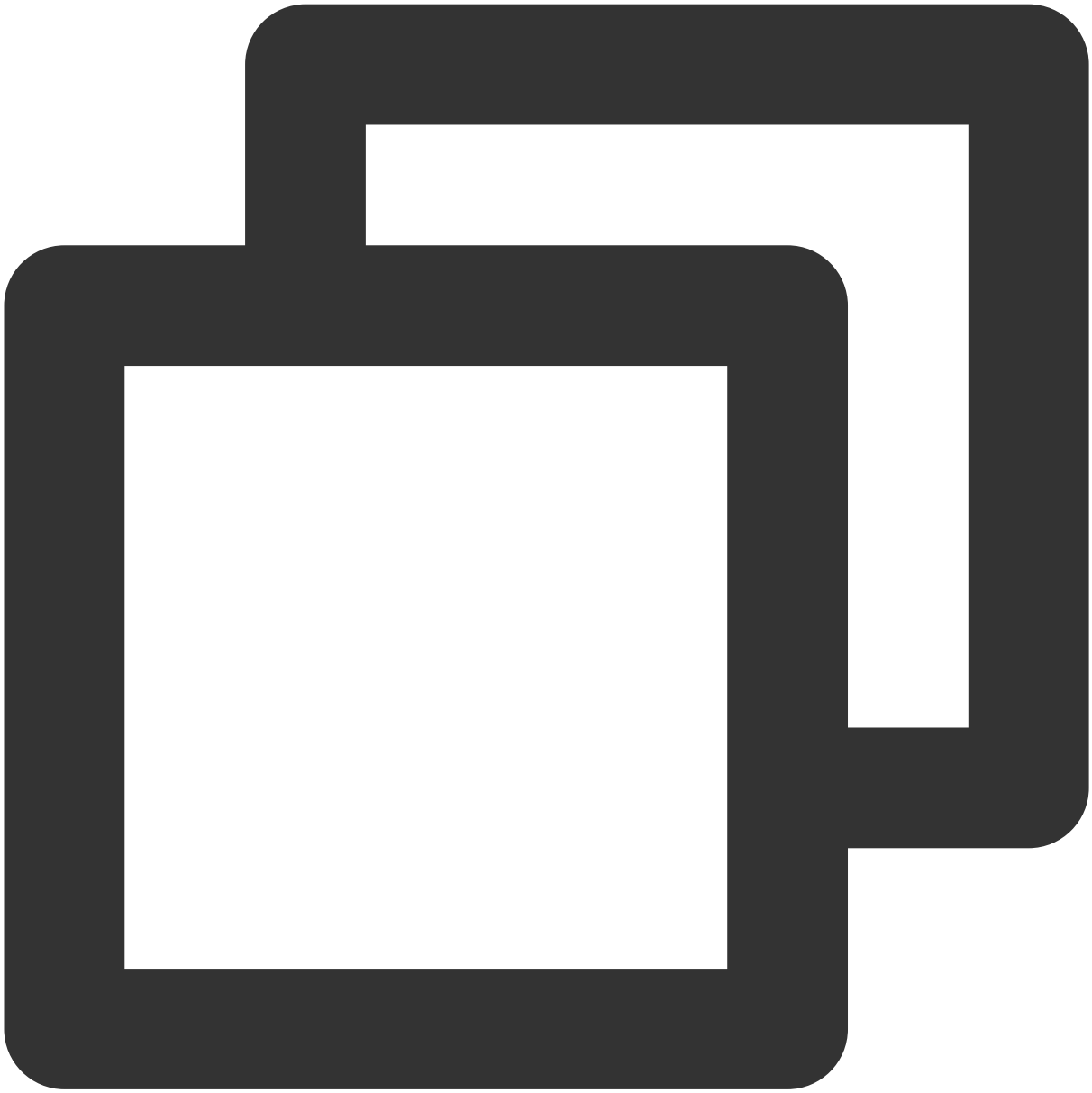
次のコマンドを使用して、React Native IM SDKの最新バージョンをインストールします。コマンドラインで次を実行します。



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

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

SDKの初期化完了

`initSDK` を呼び出して、SDKの初期化を完了します。 `sdkAppID` を渡します。

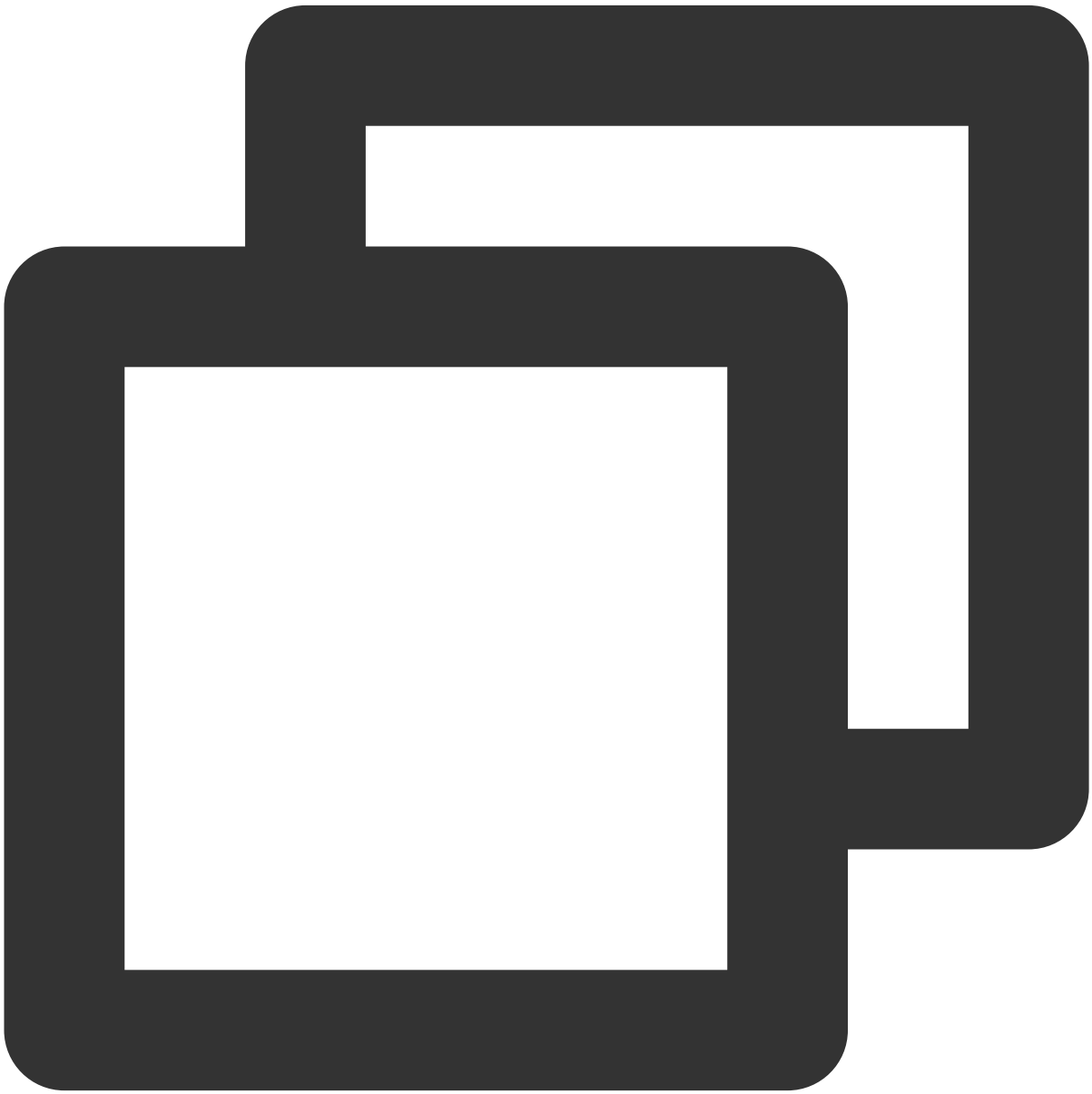


```
import { TencentImSDKPlugin, LogLevelEnum } from 'react-native-tim-js';

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

この手順ではIM SDKにいくつかのリスナーをマウントすることができます。これには主に、ネットワーク状態およびユーザー情報の変更などが含まれます。詳細については、このドキュメントをご参照ください。

テストユーザーのログイン

1. この時点で、最初にコンソール上で作成したテストアカウントを使用して、ログイン検証を完了することが可能です。
2. `TencentImSDKPlugin.v2TIMManager.login` メソッドを呼び出し、テストアカウントにログインします。戻り値 `res.code` が0であれば、ログインは成功です。



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

```
const res = await TencentImSDKPlugin.v2TIMManager.login(
userID: userID,
userSig: userSig,
 );
```

説明：

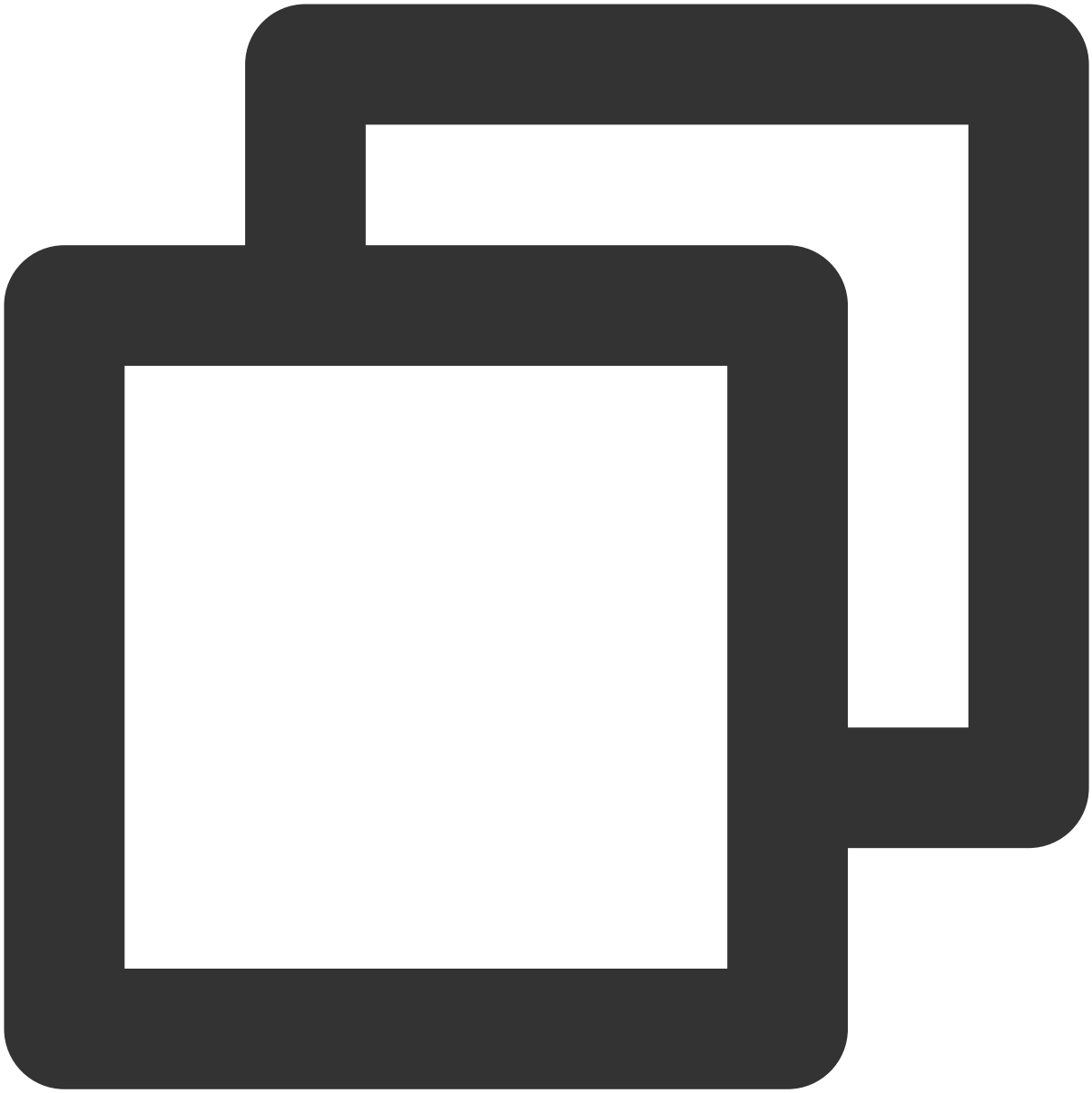
このアカウントは開発テストのみに使用します。アプリケーションのリリース前の `UserSig` の正しい発行方法は、 `UserSig` の計算コードをサーバーに統合し、Appのインターフェース向けに提供する方法となります。 `UserSig` が必要なときは、Appから業務サーバーにリクエストを送信し動的に `UserSig` を取得します。詳細はサーバーでのUserSig新規作成をご参照ください。

メッセージの送信

ここでは、テキストメッセージの送信を例に挙げます。そのフローは次のとおりです：

1. `createTextMessage(String)` を呼び出してテキストメッセージを作成します。
2. 戻り値に基づいて、メッセージIDを取得します。
3. `sendMessage()` を呼び出して、このIDのメッセージを送信します。 `receiver` には、その前に作成した別のテストアカウントIDを入力できます。シングルチャットメッセージを送信する場合は `groupID` の入力は不要です。

サンプルコード：



```
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",
            );
```

説明：

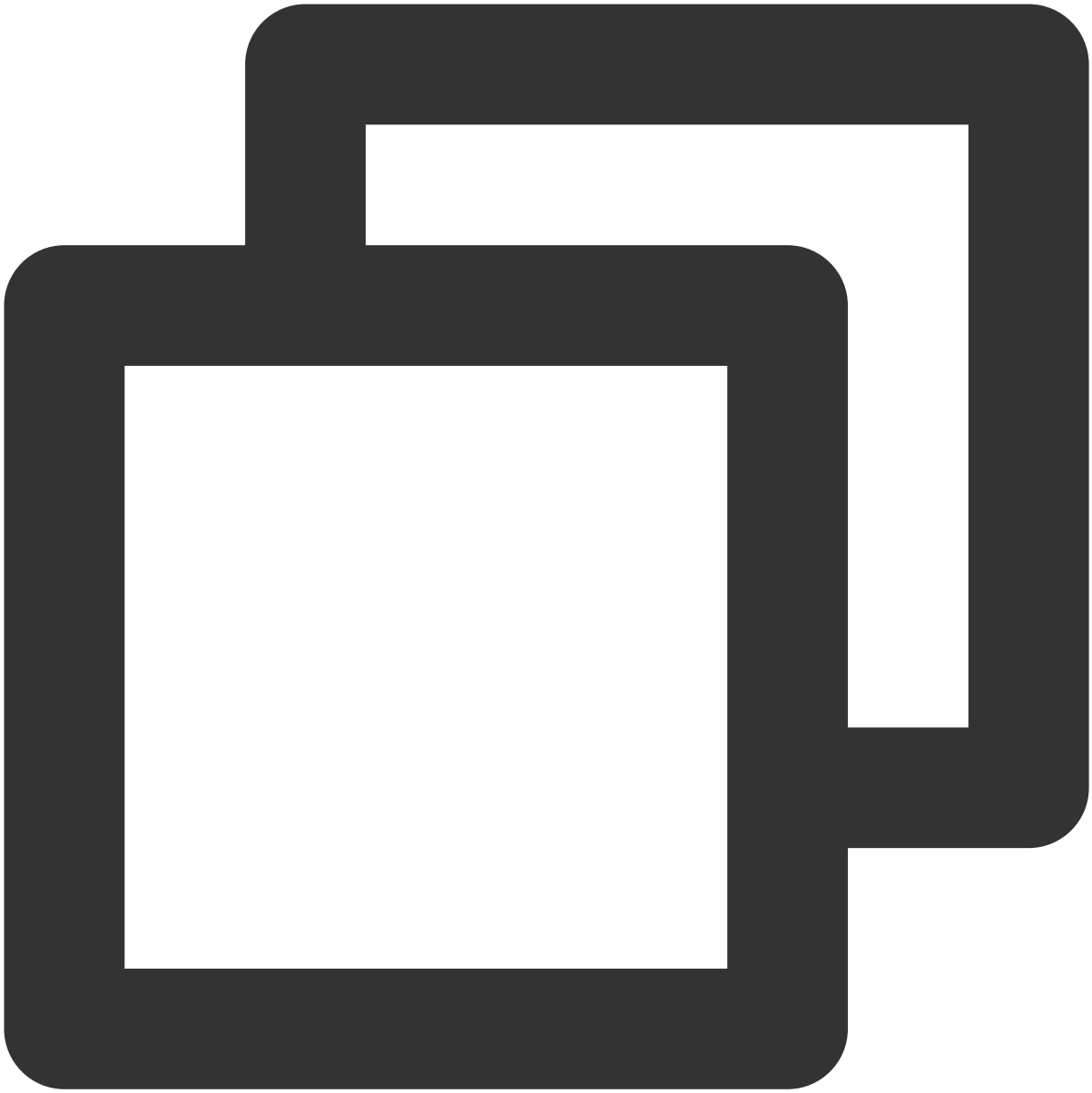
送信に失敗した場合は、sdkAppIDが知らない人宛てのメッセージ送信をサポートしていないことが原因の可能性があります。コンソールで有効にしてからテストに使用してください。
このリンクをクリックして、フレンドリレーションシップチェーンのチェックを無効にしてください。

セッションリストの取得

前の手順でテストメッセージの送信が完了しましたので、別のテストアカウントでログインし、セッションリストをプルできるようになりました。
セッションリストの取得には次の2つの方法があります：
1. 長時間接続コールバックを監視し、セッションリストをリアルタイムに更新します。
2. APIをリクエストし、ページごとに一括でセッションリストを取得します。
一般的なユースケースは次のようなものです：
アプリケーションの起動後すぐにセッションリストを取得し、その後は長時間接続を監視して、セッションリストの変更をリアルタイムに更新します。

セッションリストの一括リクエスト

セッションリストを取得するためには `nextSeq` を保守し、現在の位置を記録する必要があります。



```
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");
```

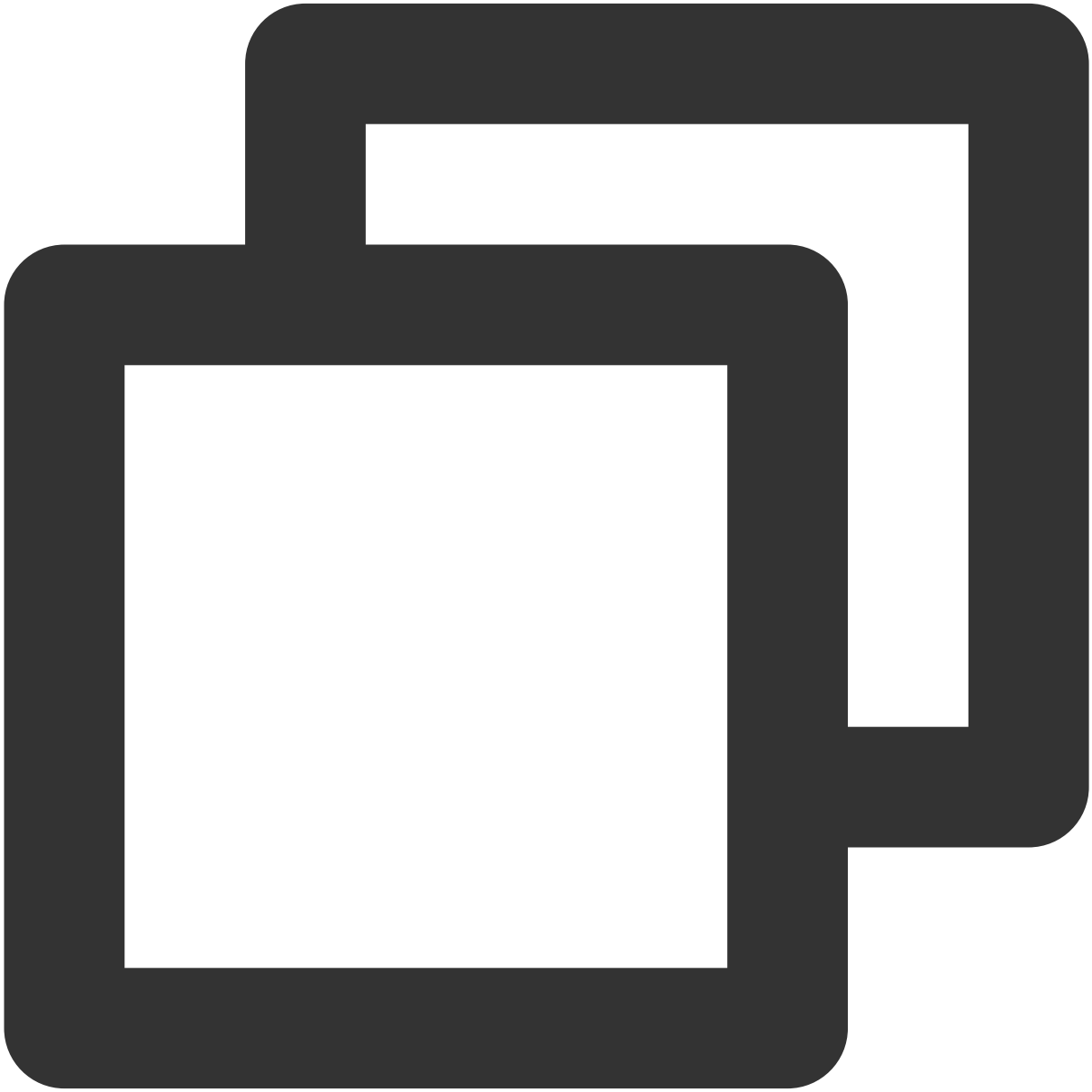
```
};
```

この時点で、前の手順で別のテストアカウントを使用して送信したメッセージのセッションを見ることができるようになりました。

長時間接続の監視によるセッションリストのリアルタイム取得

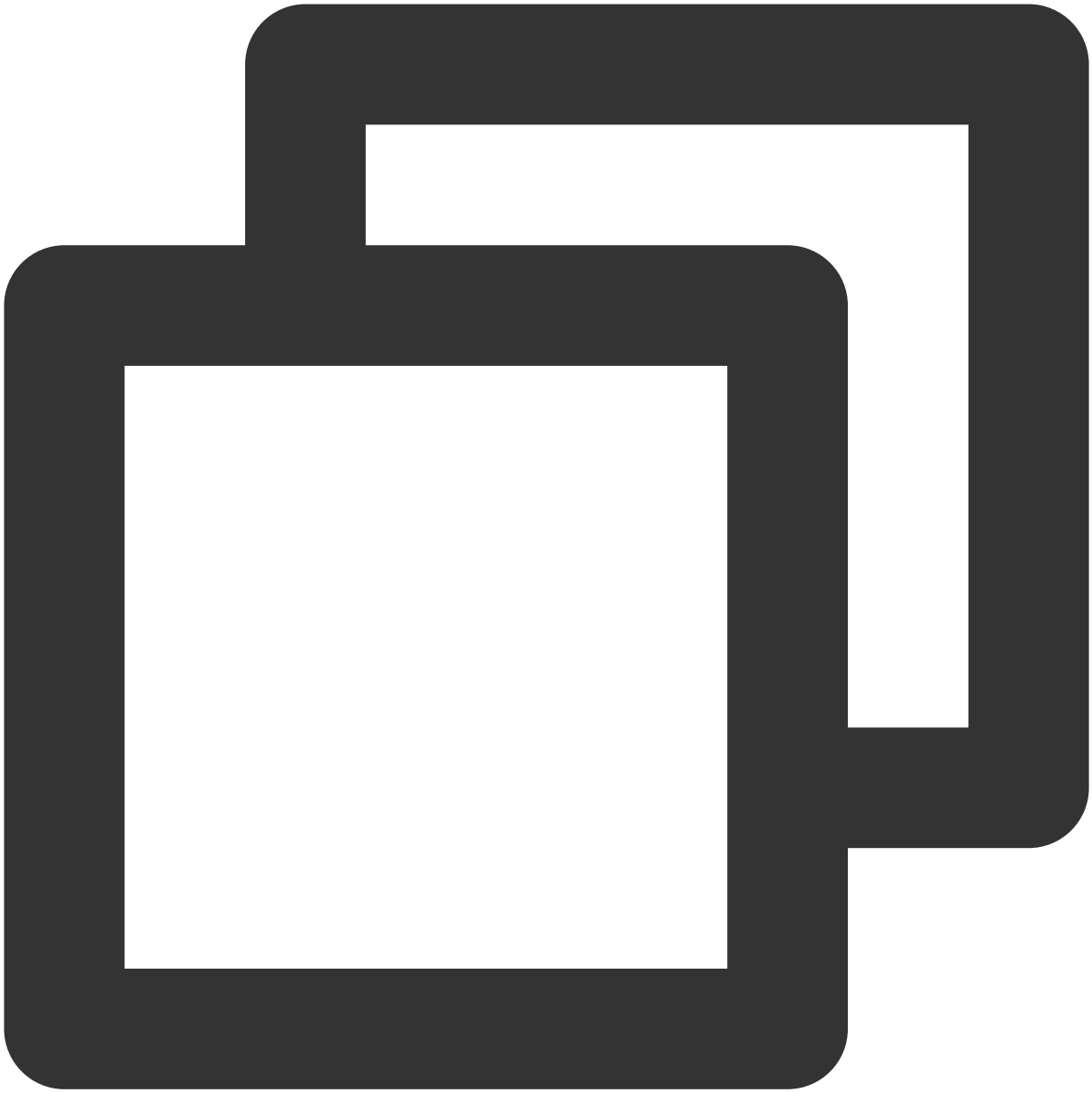
この手順では、先にSDKにリスナーをマウントしてからコールバックイベントを処理し、UIを更新する必要があります。

1. リスナーをマウントします。



```
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. コールバックイベントを処理し、最新のセッションリストをインターフェース上に表示します。



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

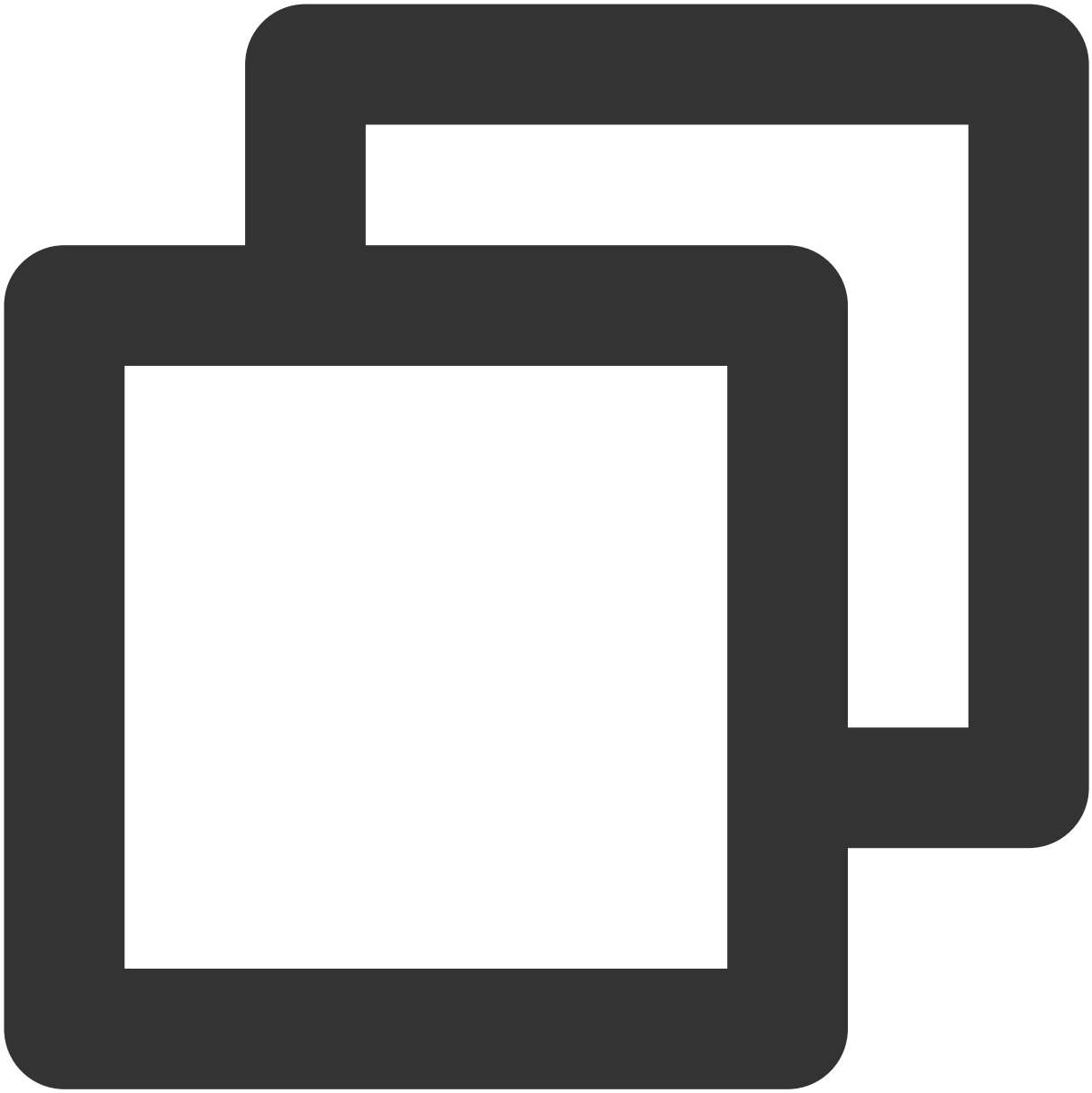
メッセージの受信

Tencent Cloud IM React Native SDKによるメッセージの受信には2つの方法があります。
1. 長時間接続コールバックを監視し、メッセージの変更をリアルタイムに取得し、メッセージ履歴リストを更新してレンダリングします。
2. APIをリクエストし、ページごとに一括でメッセージ履歴を取得します。

一般的なユースケースは次のようなものです。
1. インターフェースが新しいセッションに入ると、まず一定量のメッセージ履歴を一括でリクエストし、メッセージ履歴リストの表示に用います。
2. 長時間接続を監視し、新しいメッセージをリアルタイムに受信してメッセージ履歴リストに追加します。

メッセージ履歴リストの一括リクエスト

ページごとに取得するメッセージ数が多すぎると取得速度に影響しますので、多すぎてはなりません。20通前後に設定することをお勧めします。
次のリクエストの際に使用できるよう、現在のページ数を動的に記録しなければなりません。
サンプルコードは次のとおりです：



```
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
```

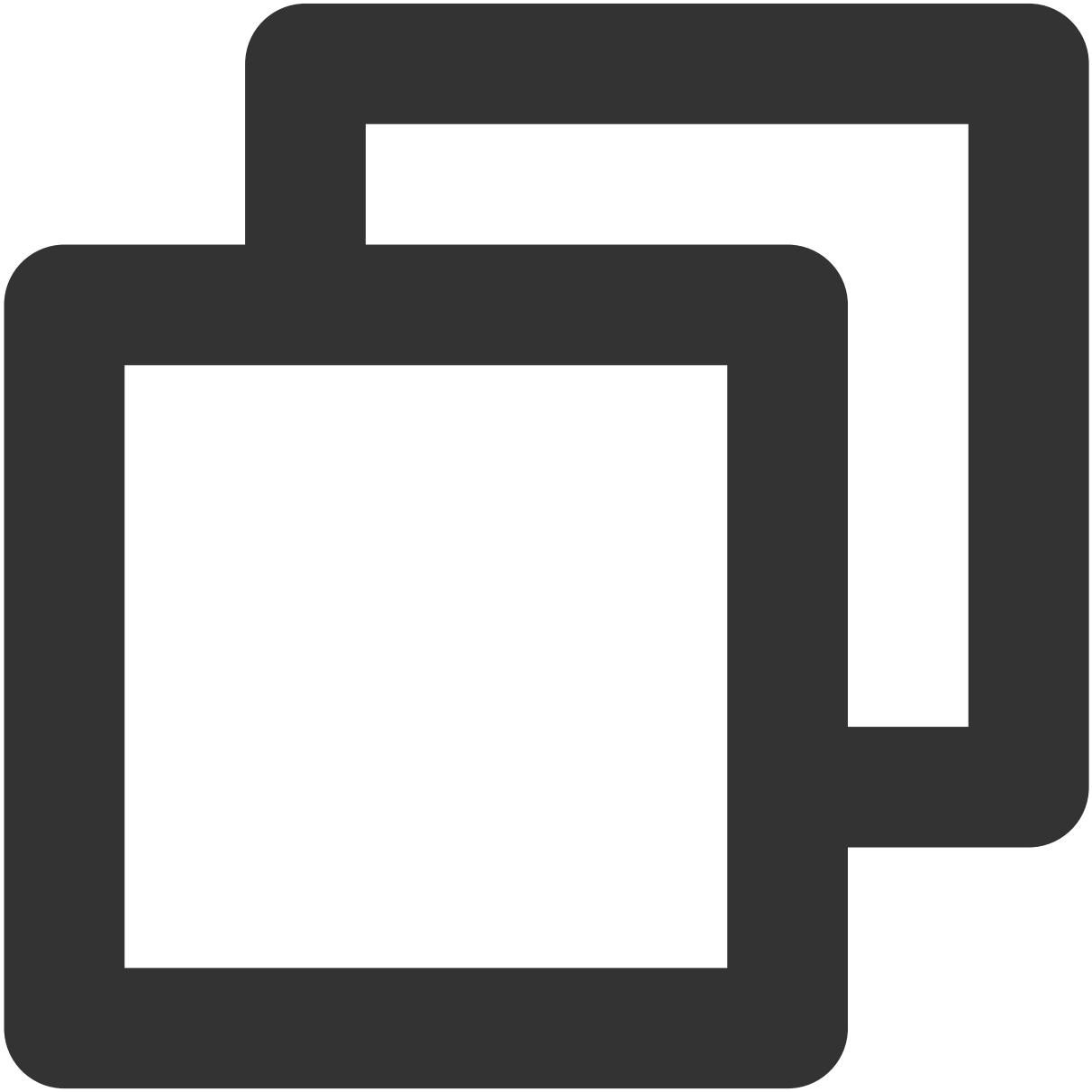
```
};
```

長時間接続の監視による新メッセージのリアルタイム取得

メッセージ履歴リストを初期化すると、新しいメッセージは長時間接続 `V2TimAdvancedMsgListener.onRecvNewMessage` から受信します。

`onRecvNewMessage` コールバックがトリガーされると、必要に応じて新しいメッセージをメッセージ履歴リストに追加することができます。

リスナーをバインドするサンプルコードは次のとおりです：



```
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);
};
```

この時点で、IMモジュールの開発は基本的に完了し、メッセージの送受信や様々なセッションに入ることが可能になりました。

続いて、グループ、ユーザープロファイル、リレーションシップチェーン、オフラインプッシュ、ローカル検索などの関連機能の開発を完了することができます。詳細については、SDK APIドキュメントをご参照ください。

よくあるご質問

demoを実行すると `Undefined symbols for architecture x86_64 [duplicate]` が表示されます。どうすればいいですか？

ドキュメントをご参照ください。

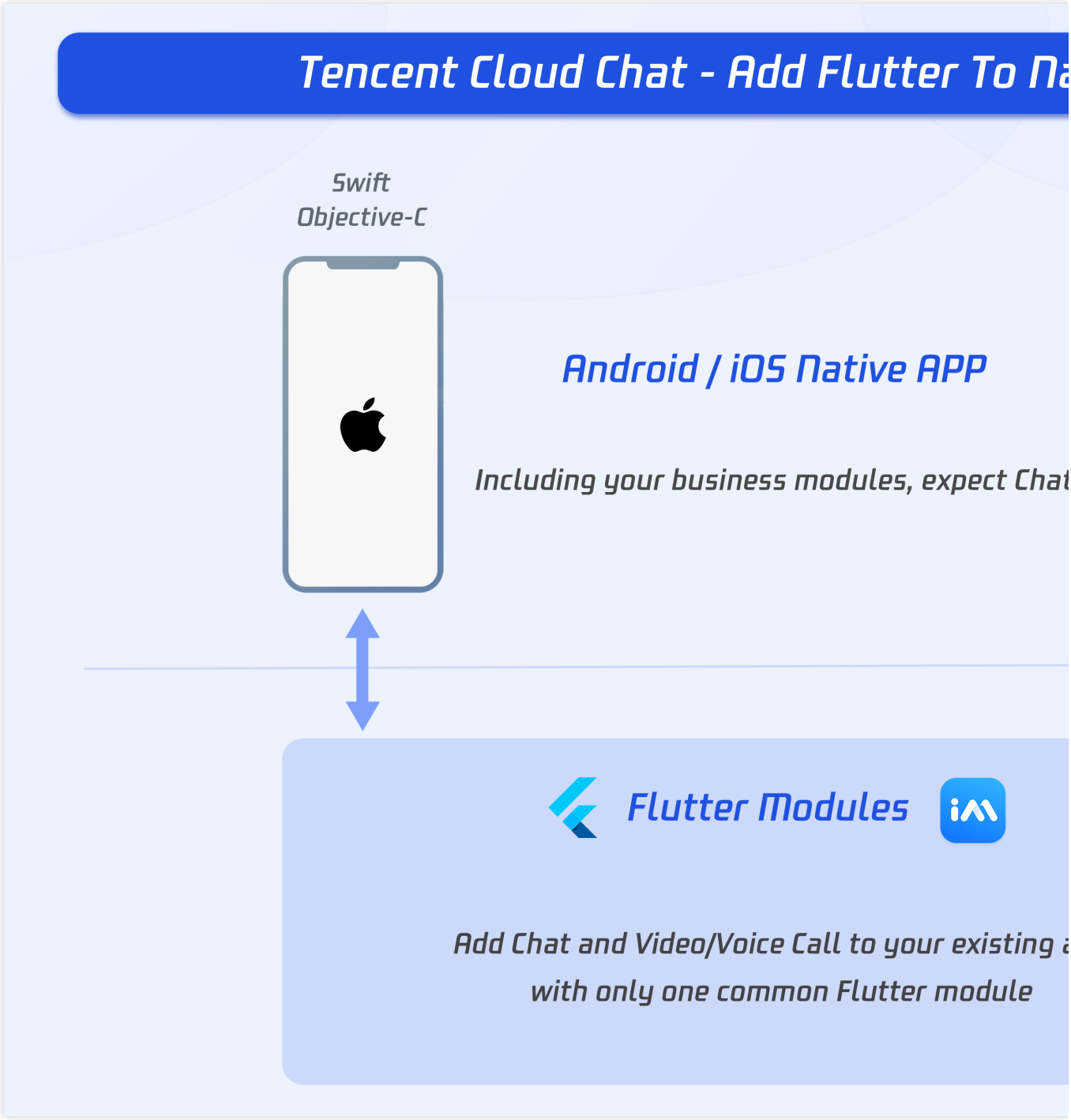
demoを実行すると `Failed to resolve: react-native-0.71.0-rc.0-debug` が表示されます。どうすればいいですか？

ドキュメントをご参照ください。

クイックスタート（Flutterをお客様のアプリケーションに速やかに統合）

最終更新日：：2024-04-11 16:20:42

このドキュメントを読むと、既存のAndroid/iOSネイティブ開発プロジェクトに、Tencent Cloud IM Flutterを統合する方法を知ることができます。
Flutterを使って既存のアプリケーションを書き換えるのは現実的ではない場合もあります。既存のAPPで、Tencent Cloud IMの機能を使用したい場合は、Flutterモジュールをネイティブ開発APPプロジェクトに組み込むハイブリッド開発スキームをお勧めします。
作業量を大幅に削減し、IM通信機能をデュアルエンドのネイティブAPPに速やかに組み込むことができます。



環境要件

環境	バージョン
Flutter	SDKの最低要件はFlutter 2.2.0バージョン、TUIKit統合コンポーネントリポジトリの最低要件はFlutter 2.10.0バージョンです。

Android	Android Studio 3.5以降のバージョン。Appの要件はAndroid 4.1以降のバージョンのデバイスです。
iOS	Xcode 11.0以降のバージョン。プロジェクトに有効な開発者署名を設定済みであることを確認してください。
Tencent Cloud IM SDK	tencent_im_sdk_plugin 5.0以降、 tim_ui_kit 0.2以降。

説明：
上記のDemoプロジェクトのソースコードは、GitHubウェアハウスにありますので、ぜひご利用ください。

始める前に知るべきこと

始める前に、Tencent CloudのIM Flutter SDKとTUIKitの使用方法、そしてFlutter-ネイティブハイブリッド開発原理を理解しておく必要があります。

Tencent Cloud IM

入門全般

始める前に、Tencent Cloud IM FlutterのSDK構成と使用方法を理解しておく必要があります。
主に、UIなしバージョンとUIありコンポーネントライブラリの2つのSDKがあります。このドキュメントでは、UIありコンポーネントライブラリ（TUIKit）を例に、ハイブリッド開発スキームを紹介します。
Tencent CloudのIM Flutterの詳細な使用方法については、クイックスタートドキュメントをご参照ください。

2つのモジュール

Tencent Cloud IMには主にChatチャットモジュールとCall通話モジュールの2つの部分があります。
Chatモジュールには主にメッセージ受送信、セッション管理、ユーザー関係管理などが含まれます。
Call通話モジュールは主に、一対一通話とグループでの複数人トークを含むオーディオ・ビデオ通話があります。

Flutterハイブリッド開発

核となる原理は、module形式のFlutterプロジェクトを、Native側の実行可能プログラムとしてパッケージ化し、Nativeプロジェクトに組み込むことです。Flutter moduleは共通化されているため、Flutter moduleを1回書くだけでAndroid/iOSアプリに埋め込むことができます。
既存のアプリケーションでTencent Cloud IM関連ページを表示する必要がある場合は、Flutterを搭載するためのActivity (Android)またはViewController (iOS)を読み込むことができます。
Method Channelは、現在のユーザ情報の配信、オディオ/ビデオ通話データの配信、オフラインプッシュデータのトリガなど、双方向通信が必要な場合に使用できます。他側のメソッドをトリガするには `invokeMethod` を使

用し、他側からのメソッド呼び出しをリッスンするにはプリマウントされたMethod Channel Listenerを使用します。

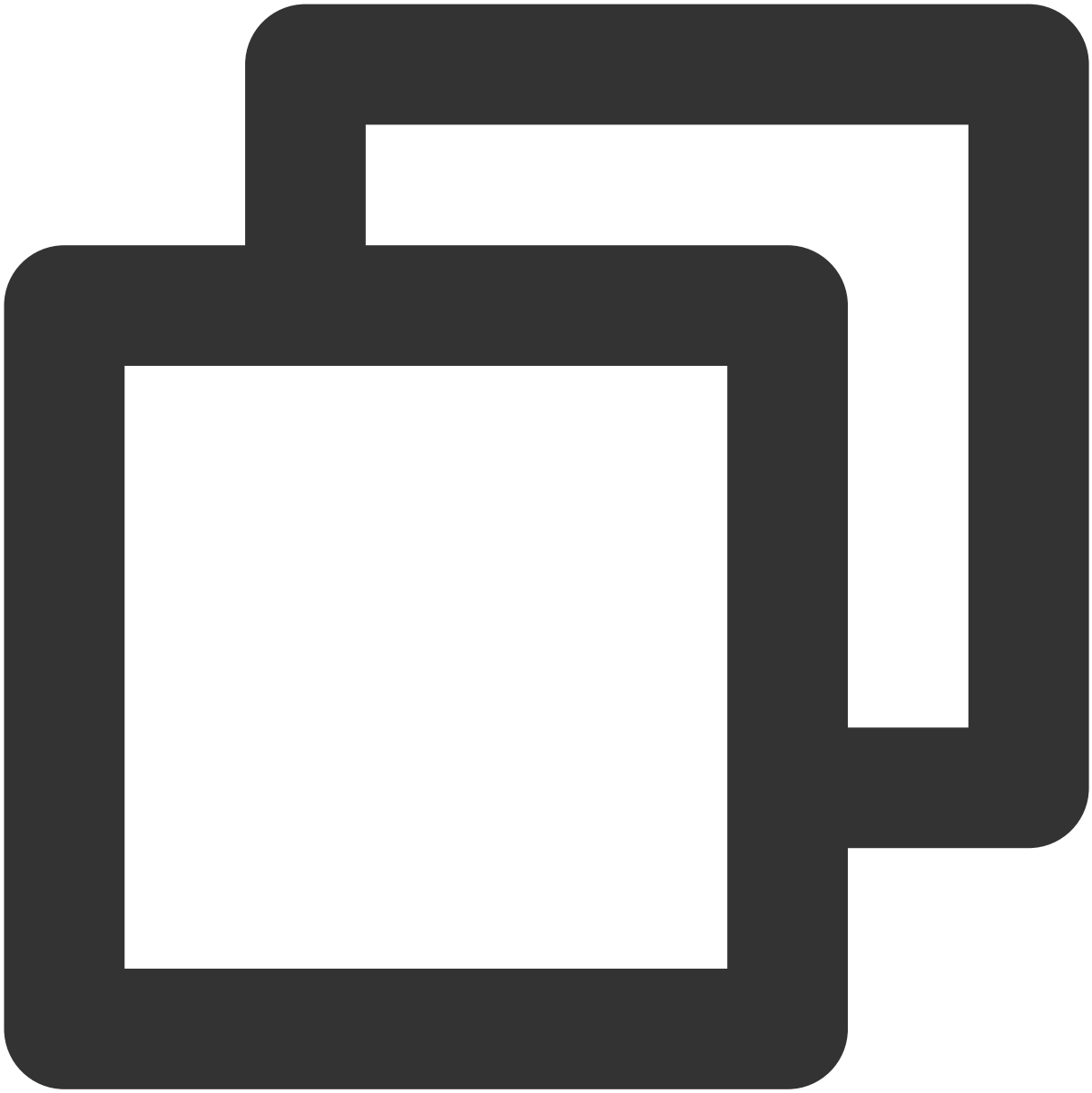
AndroidプロジェクトにFlutterモジュールを追加

詳しく学ぶ

Gradle内の既存のアプリケーションの依存関係としてFlutter moduleを追加します。これを実現するには2つの方式があります。

Android方式その1：Android Archive (AAR)への依存

AARメカニズムは、Flutter moduleのパッケージングのための媒介として汎用Android AARを作成します。頻繁にビルドする場合は、ビルド手順が1つ追加されます。
このオプションは、Flutterライブラリを、AARおよびPOMS Widgetで構成される汎用のローカルMavenリポジトリとしてパッケージ化します。このオプションにより、あなたのチームがFlutter SDKをインストールせずにホストアプリケーションを構築できます。その後、ローカルまたはリモートリポジトリからWidgetを配布できます。
そのため、オンラインの本番環境でこのソリューションを使用することをお勧めします。
具体的な手順：
Flutter moduleで次の操作を実行します：



```
flutter build aar
```

次に、画面の指示に従って統合を行います。

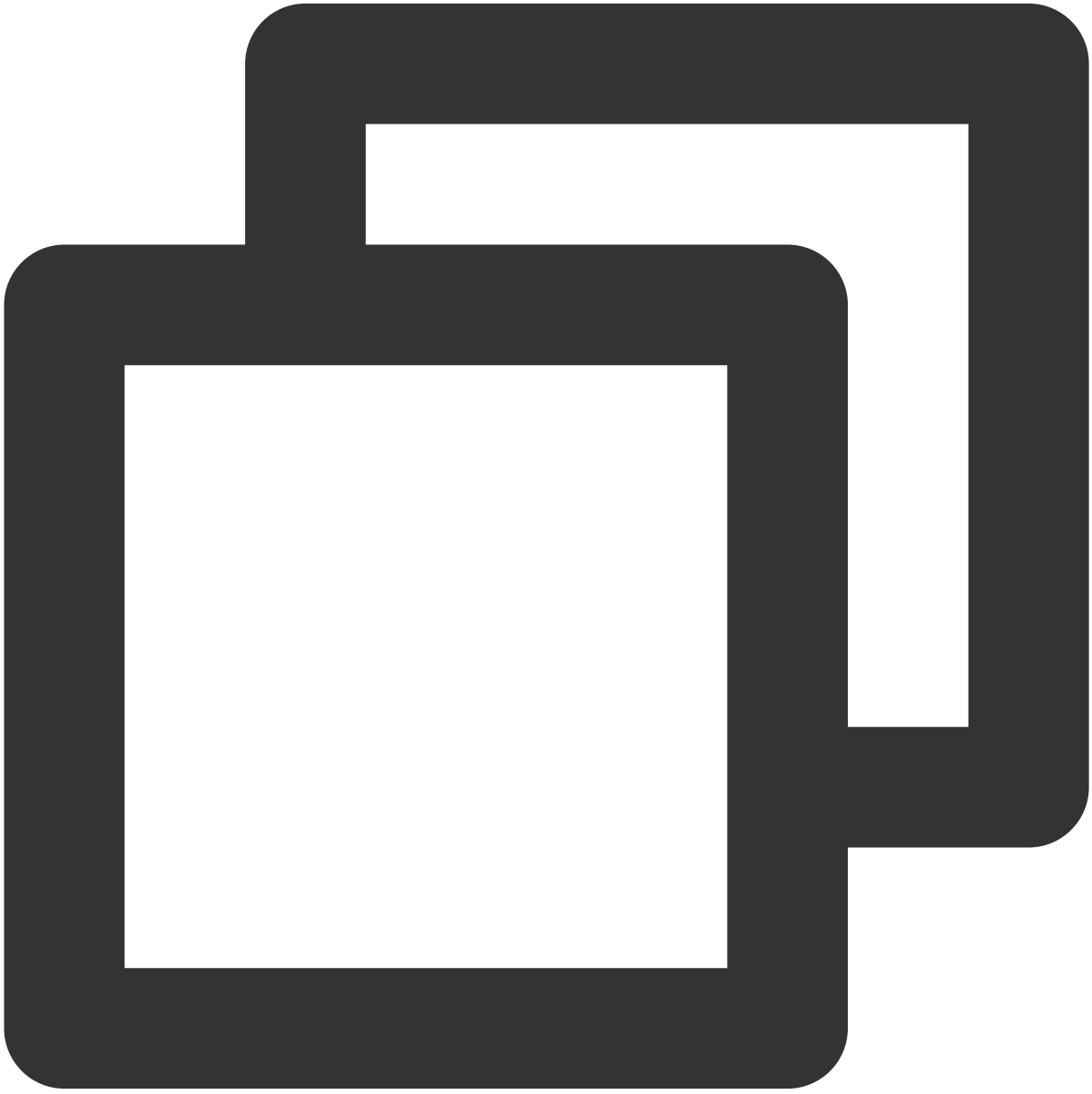
```
Running "flutter pub get" in tencent_chat_module...                 4.2s
➜  tencent_chat_module git:(main) ✗ flutter build aar

💪 Building with sound null safety 💪

Running Gradle task 'assembleAarDebug'...                         114.0s
✓ Built build/host/outputs/repo.
Running Gradle task 'assembleAarProfile'...                        66.4s
✓ Built build/host/outputs/repo.
Running Gradle task 'assembleAarRelease'...                        60.4s
✓ Built build/host/outputs/repo.

Consuming the Module
  1. Open <host>/app/build.gradle
  2. Ensure you have the repositories configured, otherwise add them:

      String storageUrl = System.env.FLUTTER_STORAGE_BASE_URL ?: "https://storage.googleapis.com"
      repositories {
        maven {
            url '/Users/wangrunlin/Documents/GitHub/tencentchat-add-flutter-to-app/Multiple Flutter Engines/ten
        }
        maven {
            url "$storageUrl/download.flutter.io"
        }
      }

  3. Make the host app depend on the Flutter module:

    dependencies {
      debugImplementation 'com.tencent.chat.flutter.module:flutter:1.0:debug'
      profileImplementation 'com.tencent.chat.flutter.module:flutter:1.0:profile'
      releaseImplementation 'com.tencent.chat.flutter.module:flutter:1.0:release'
    }


  4. Add the `profile` build type:

    android {
      buildTypes {
        profile {
          initWith debug
        }
      }
    }

To learn more, visit https://flutter.dev/go/build-aar
➜  tencent_chat_module git:(main) ✗
```

あなたのアプリケーションには依存関係としてFlutterモジュールが含まれるようになりました。

Android方式その2：Flutter moduleのソースコードへの依存

ソースコードのサブプロジェクトメカニズムは、簡単なワンタッチのビルドプロセスですが、Flutter SDKが必要です。これはAndroid Studio IDEプラグインで使われている仕組みです。

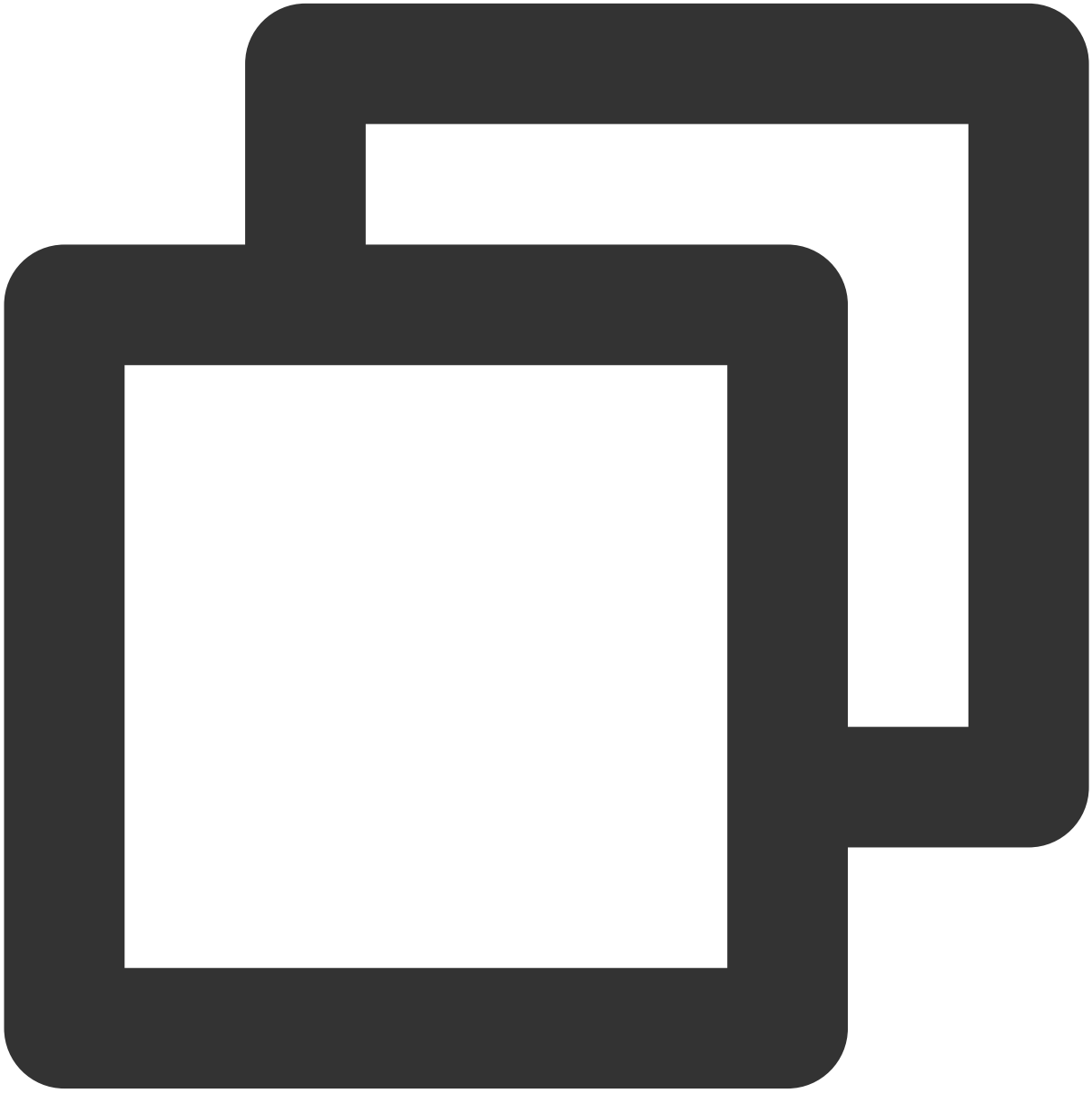
この方式は、あなたのAndroidプロジェクトとFlutterプロジェクトの両方を1ステップで構築できます。このオプションは、2つの部分を同時に処理して高速でイテレーションする場合に便利ですが、あなたのチームがアプリケーションを構築するためのFlutter SDKをインストールする必要があります。
そのため、開発テスト環境では、本ソリューションを用いることをお勧めします。
具体的な手順：
Flutter moduleをサブプロジェクトとして、ホストAPPの `settings.gradle` に追加します。



```
// Include the host app project.
include ':app'                                  // assumed existing content
setBinding(new Binding([gradle: this]))                                // new
```

```
evaluate(new File(                                                    // new
  settingsDir.parentFile,                                             // new
  'tencent_chat_module/.android/include_flutter.groovy'               // new
))                                                                    // new
```

お客様のアプリケーションの `app/build.gradle => dependencies` に、Flutter moduleに対する `implementation` を導入する：



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

あなたのアプリケーションには依存関係としてFlutterモジュールが含まれるようになりました。

iOSプロジェクトにFlutterモジュールを追加

詳しく学ぶ

既存のアプリケーションにFlutterを埋め込む方法は2つあります。

iOS方式その1：CocoaPodsとFlutter SDKを埋め込む

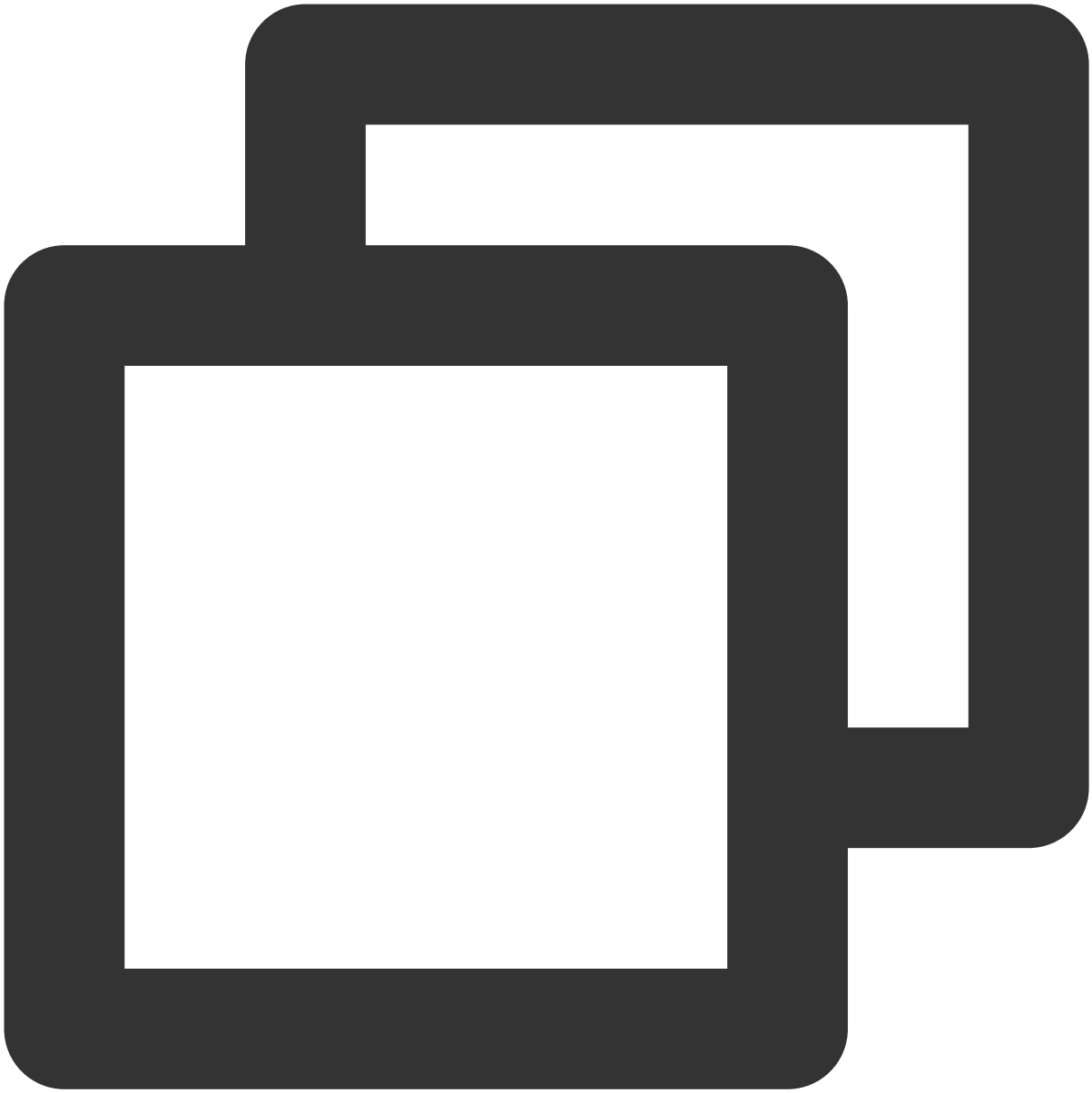
CocoaPods依存関係マネージャを使用し、Flutter SDKをインストールします。この方法では、プロジェクトに携わるすべての開発者がFlutter SDKのバージョンをローカルにインストールする必要があります。

アプリケーションをXcodeにビルドするだけで、DARTとプラグインコードを埋め込むスクリプトが自動的に実行されます。これにより、Xcode以外で他のコマンドを実行することなく、最新バージョンのFlutterモジュールを高速でイテレーションすることができます。

そのため、開発テスト環境では、本ソリューションを用いることをお勧めします。

具体的な手順：

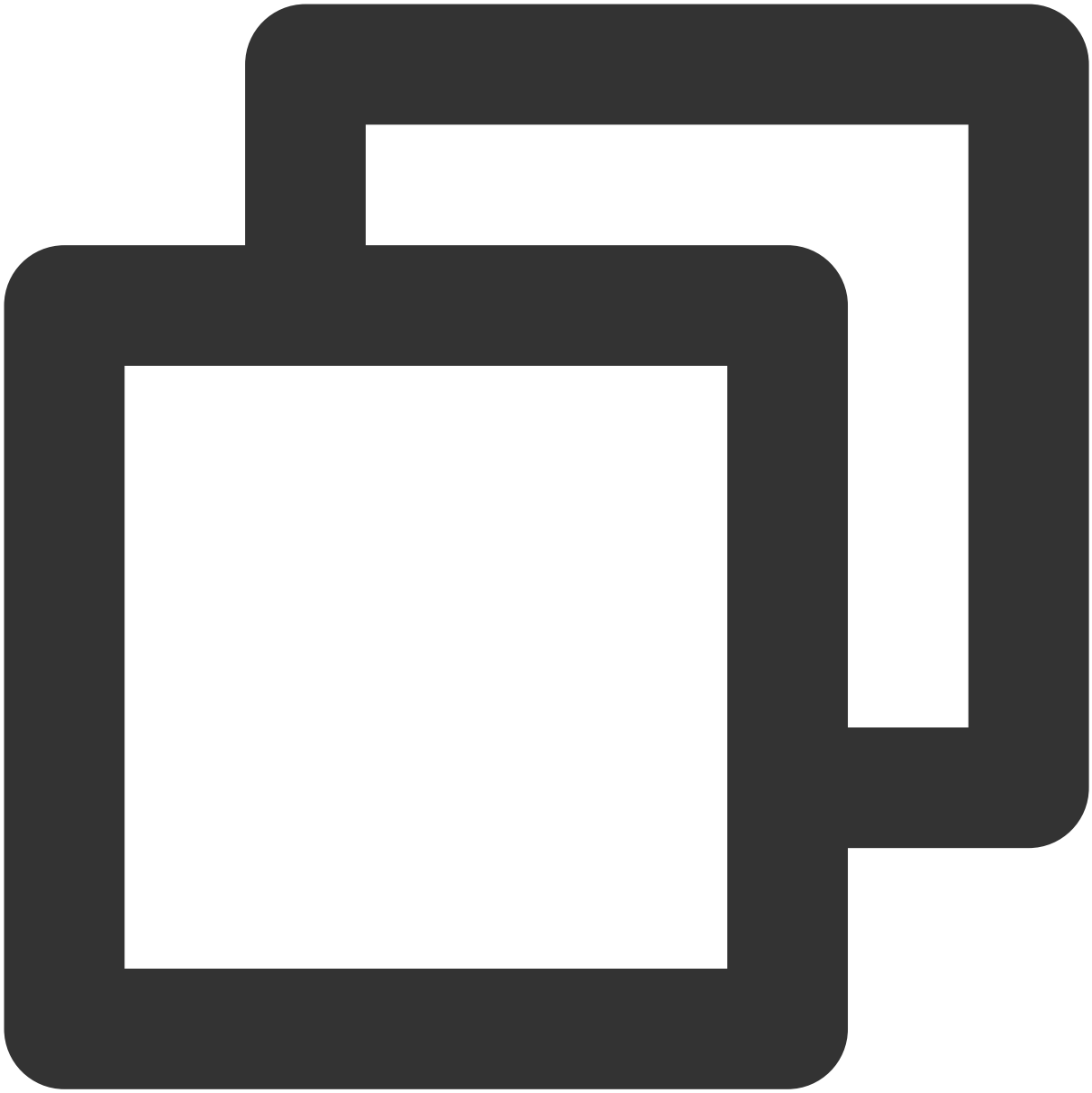
次のコードをPodfileに追加します：



```
// 前のステップで構築したFlutter Moduleのパス
flutter_chat_application_path = '../tencent_chat_module'

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

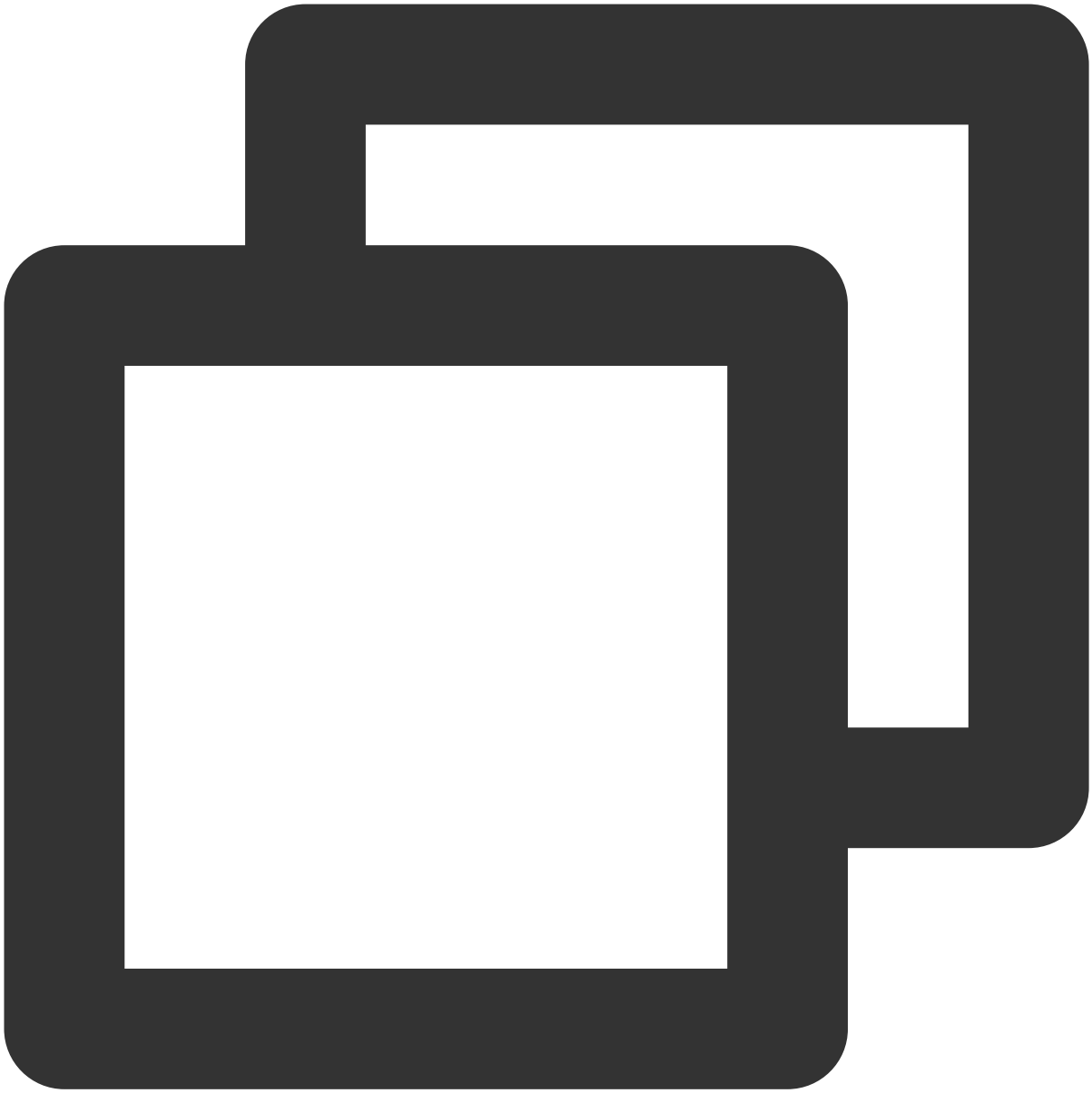
Flutterを組み込むPodfile targetに対

し、 `install_all_flutter_pods(flutter_chat_application_path)` を呼び出します。



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

Podfileの `post_install` ブロックで `flutter_post_install(installer)` を呼び出し、Tencent Cloud IM TUIKitに必要な権限宣言（マイク/カメラ/アルバムの権限を含む）を行います。



```
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
```

`pod install` を実行します。

説明：

`tencent_chat_module/pubspec.yaml` でFlutterプラグインの依存関係を変更する場合は、Flutter　Module目ディレクトリで `flutter pub get` を実行して、 `podhelper.rb` スクリプトによって読み込まれたプラグインのリストを更新してください。次に、iOSアプリケーションのルートディレクトリから「pod install」を再度実行します。

Apple Siliconチップarm64アーキテクチャのMacでは、 `arch-x86_64 pod install--repo-update` を実行する場合があります。

`podhelper.rb` スクリプトは、あなたのプラグイン/ `Flutter.framework` / `App.framework` を該当するプロジェクトに移植します。

iOS方式その2：Xcodeにframeworksを埋め込む

Flutterエンジン、コンパイルされたDARTコード、およびすべてのFlutterプラグインのために必要なフレームワークを作成します。フレームワークを手動で埋め込み、Xcodeで既存のアプリケーションのビルド設定を更新します。

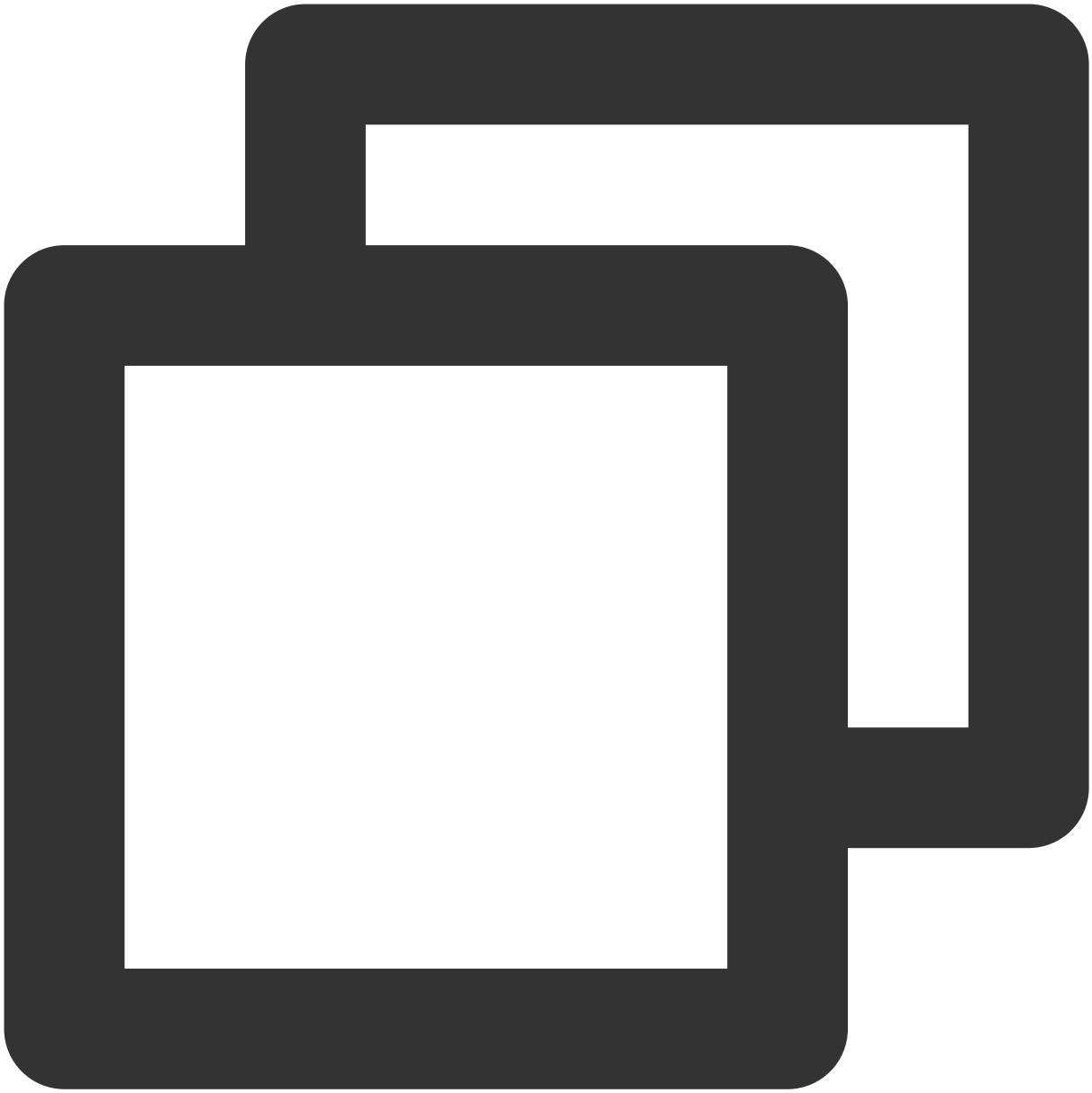
既存のXcodeプロジェクトを手動で編集することで、必要なframeworkを生成してアプリケーションに埋め込むことができます。あなたのチームメンバーがFlutter SDKとCocoaPodsをローカルにインストールできない場合や、既存のアプリケーションでCocoaPodsを依存関係マネージャとして使用したくない場合に、このように行うことができます。Flutterモジュールでコードを変更するたびに、 `flutter buildios-framework` .を実行しなければなりません。

そこで、オンライン環境でこのソリューションを用いることをお勧めします。

具体的な手順：

Flutter moduleで、次のコードを実行します。

次の例では、frameworkを `some/path/MyApp/Flutter/` に生成するとします。



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

生成されたframeworksをXcodeで既存のアプリケーションに統合します。例えば、`some/path/MyApp/Flutter/Release/`ディレクトリで、frameworksをあなたのアプリケーションのtargetコンパイル設定の「General ＞ Frameworks,Libraries,and Embedded Content」の下にドラッグし、「Embed」ドロップダウンリストから「Embed&Sign」を選択することができます。

ハイブリッド開発オプション

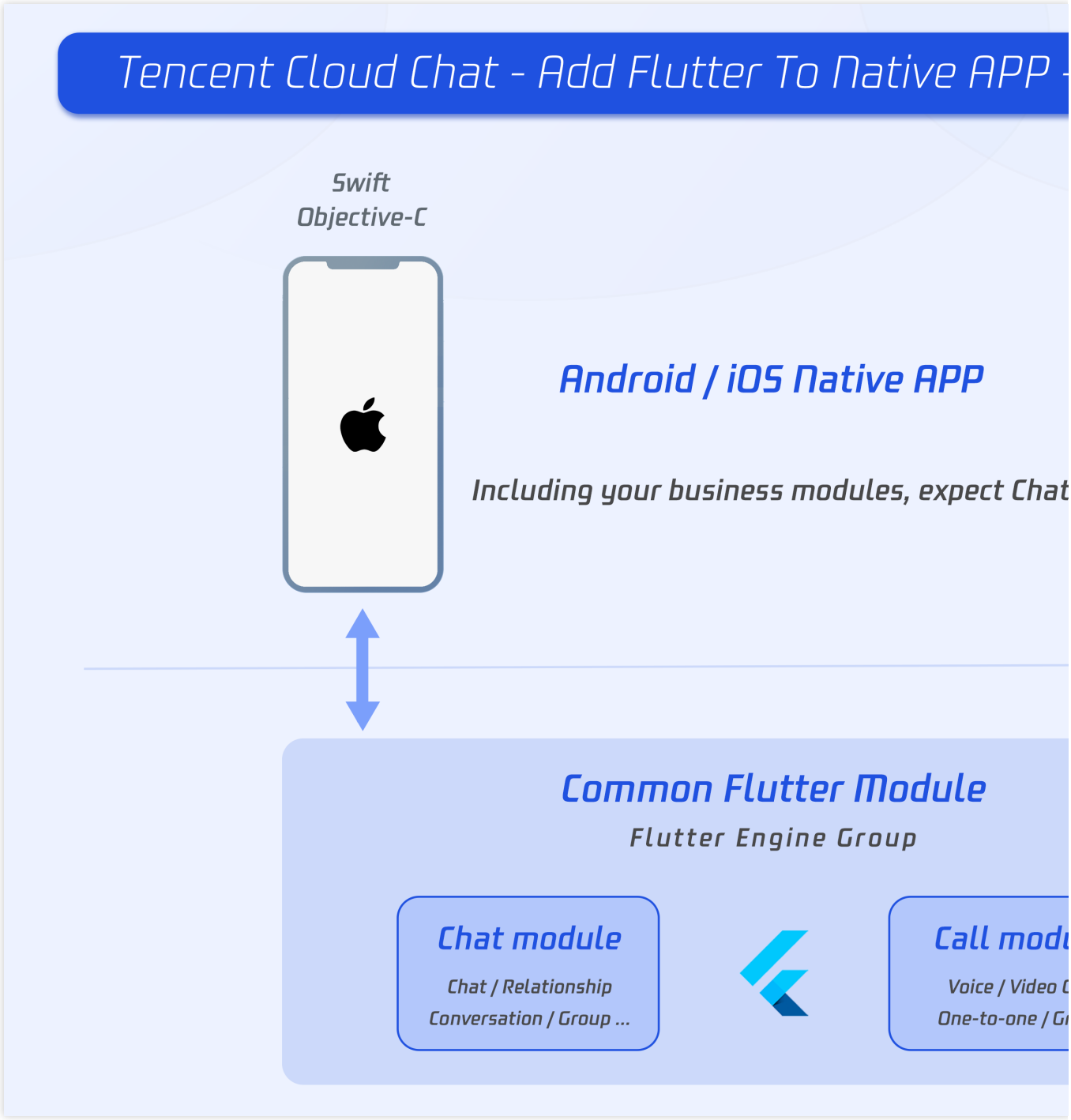
ハイブリッド開発の統合を行うためにFlutter Module方式を使用することをお勧めします。
Nativeプロジェクトでは、Flutterエンジンを構築し、FlutterのChatおよびCallモジュールを搭載します。2つのモジュールについては、こちらをご参照ください。
Flutterエンジンの作成管理には、現在、単一のFlutterエンジンと複数のFlutterエンジンの2つの方法があります。

エンジンモード	紹介	メリット	デメリット	Demoソースのダウンロード
Flutter単一エンジン	ChatモジュールとCallモジュールは、同じFlutterエンジンで搭載されます。	すべてのFlutterコードの統一的なメンテナンスに便利です。	Callプラグインにより、電話の着信があった場合には、自動的に着信ページを表示する必要があります。同じエンジンでは、Flutterのあるページに強制的にジャンプする必要があり、体験がよくない。	クリックしてダウンロードする
Flutterマルチエンジン	ChatモジュールとCallモジュールはそれぞれ異なるFlutterエンジンに搭載され、Flutterエンジングループを使用して両エンジンを一元的に管理します。	Callプラグインは独立して1つのFlutterエンジンの中に存在し、個別ページで制御されます。着信時に、直接このページをポップアップすることができ、ユーザーの現在のページに影響しないで、比較的好ましい体験を得られます。	通話モジュールをフローティングウインドーに最小化することはできません。	クリックしてダウンロードする

また、Tencent Cloud IM Native SDKとFlutter SDKを組み合わせて使用するソリューションも用意されています。
ユースケースと手順の説明についてはこちらをご参照ください。Demoソースのダウンロード。

ソリューション1：Flutterマルチエンジンソリューション【推奨】

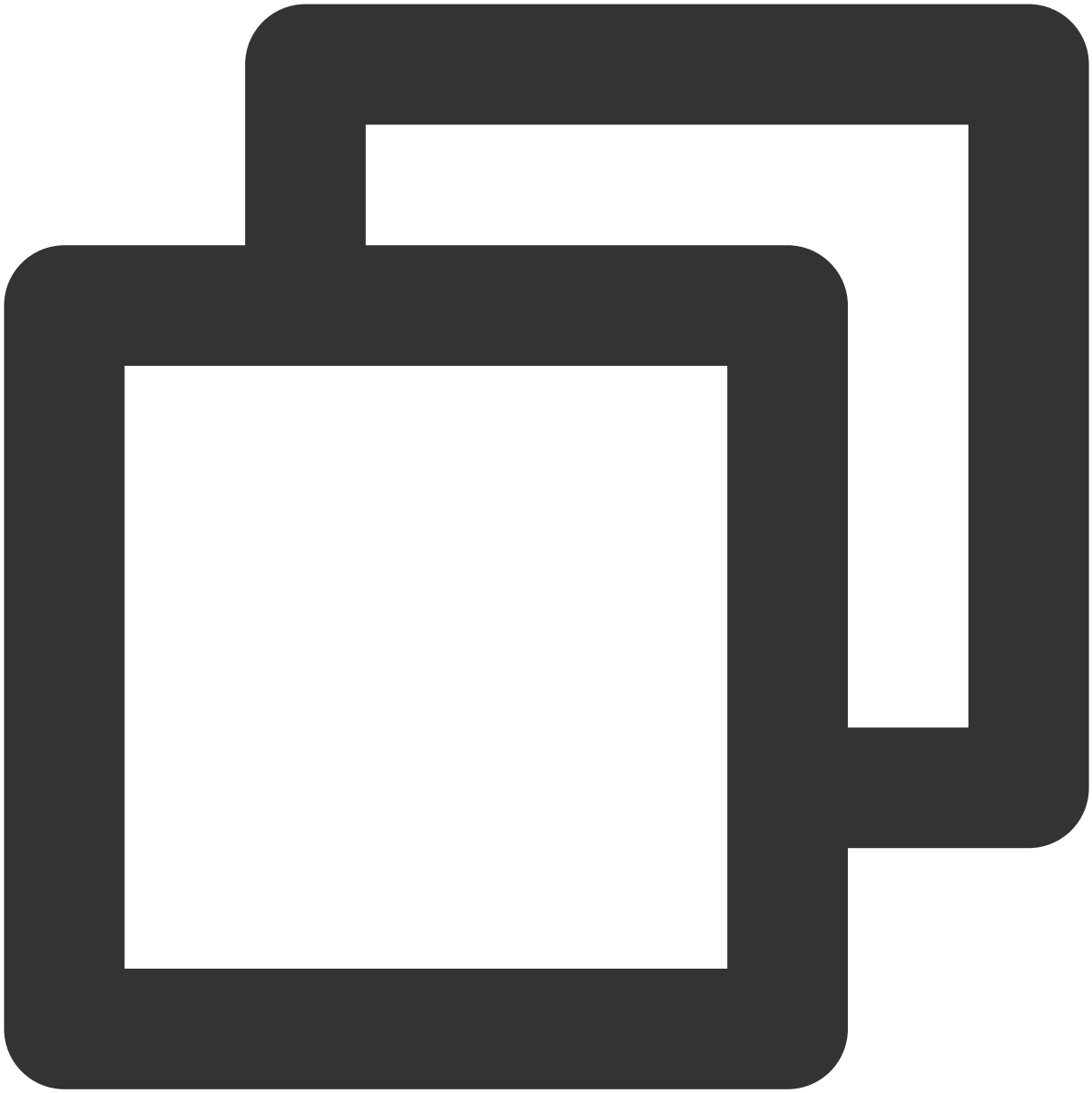
このソリューションでは、ChatモジュールとCallモジュールはそれぞれ異なるFlutterエンジンから独立しています。
複数のFlutterエンジンを使用する利点は、各インスタンスが独立したもので、独自の内部ナビゲーションスタック、UI、およびアプリケーションステータスを維持することです。これにより、アプリケーションコード全体の状態維持責任が合理化され、モジュール機能が向上します。



AndroidとiOSに複数のFlutterエンジンを追加し、主に1つのFlutterEngineGroupクラス（Android API、iOS API）に基づいて複数のFlutterEngine（Flutterエンジン）を構築・管理します。
このプロジェクトでは、統合されたFlutterEngineGroupに基づいて、ChatモジュールとCallingモジュールを搭載する2つのFlutterEngine（Flutterエンジン）を管理しています。

Flutter Moduleの開発

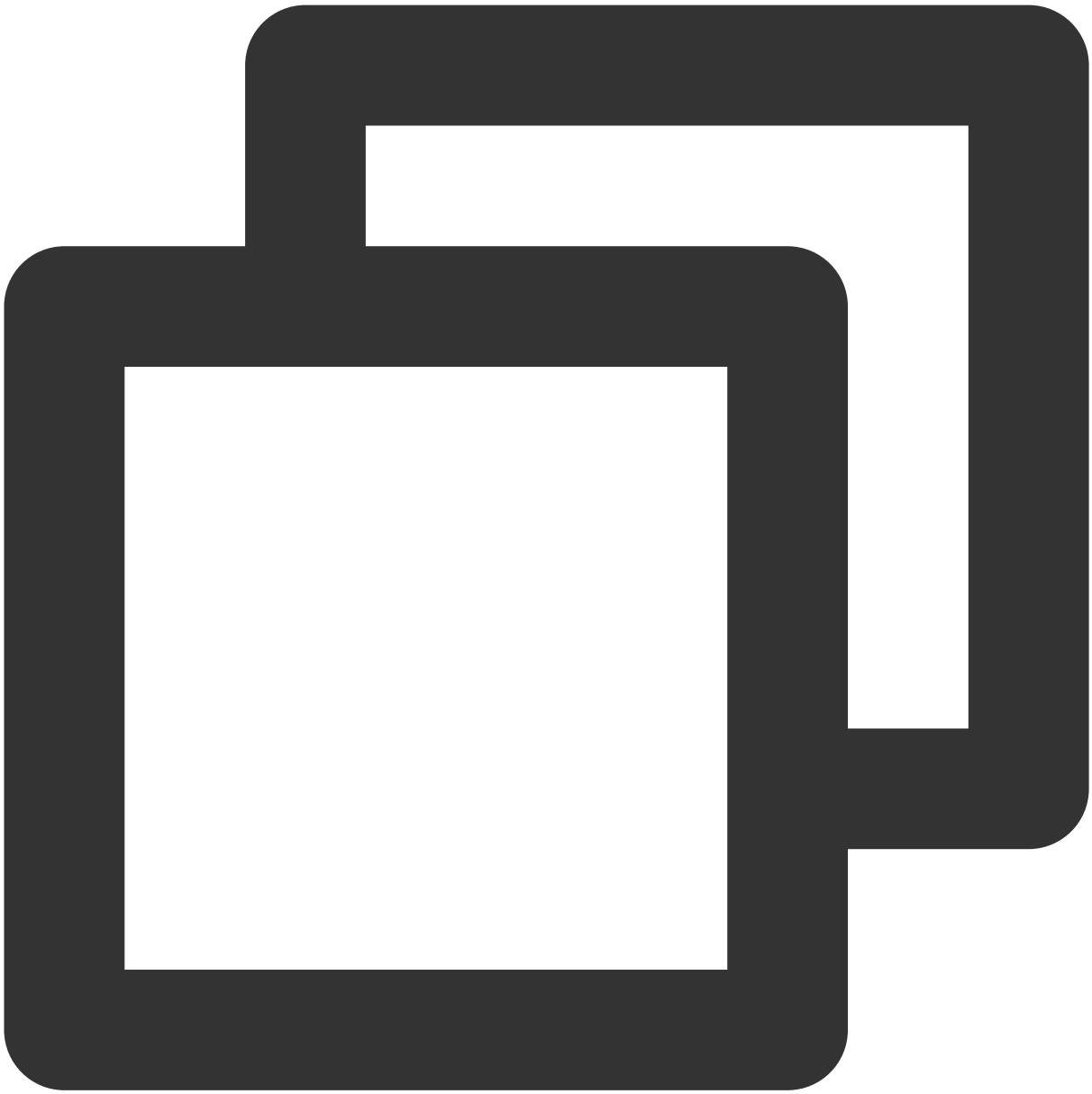
既存のアプリケーションにFlutterを埋め込むには、まずFlutterモジュールを作成します。
プロジェクトのルートディレクトリの外層で次を実行します。



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

これにより、some/path/tencent_chat_module/にFlutterモジュールプロジェクトが作成されます。このディレクトリでは、 `flutter run--debug` や `flutter buildios` など、他のFlutterプロジェクトと同じFlutterコマンドを実行できます。FlutterおよびDartプラグインを使用して、Android Studio、IntelliJまたはVS Codeでこのモジュールを実行することもできます。このプロジェクトには、既存のアプリケーションに埋め込む前にモジュールの単一ビューのサンプルバージョンが含まれています。これは、コードのFlutterのみの部分をテストするのに有効です。

`tencent_chat_module` モジュールディレクトリ構造は、通常のFlutterアプリケーションに似ています：

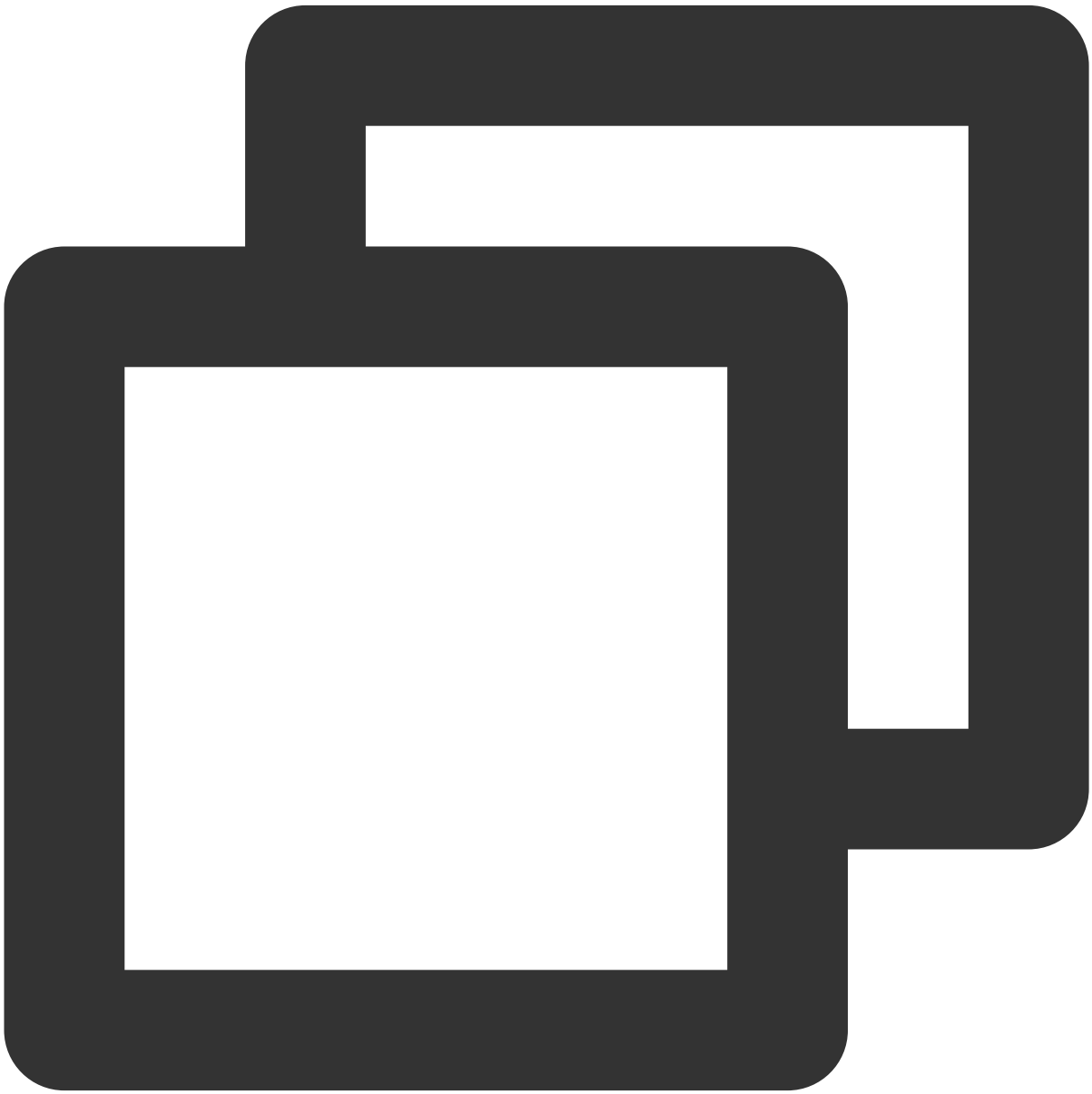


```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

これで `lib/` に、コードを書くことができるようになりました。

Flutter libディレクトリを整理

説明：
以下のコード構造は参考までに記載されています。必要に応じてTencent Cloud IM Flutterを導入するために柔軟に構成することができます。
`lib/` で `call` , `chat` , `common` という3つのディレクトリを作成します。それぞれ通話エンジン、IMエンジン、汎用modelクラスを置くために用いられます。

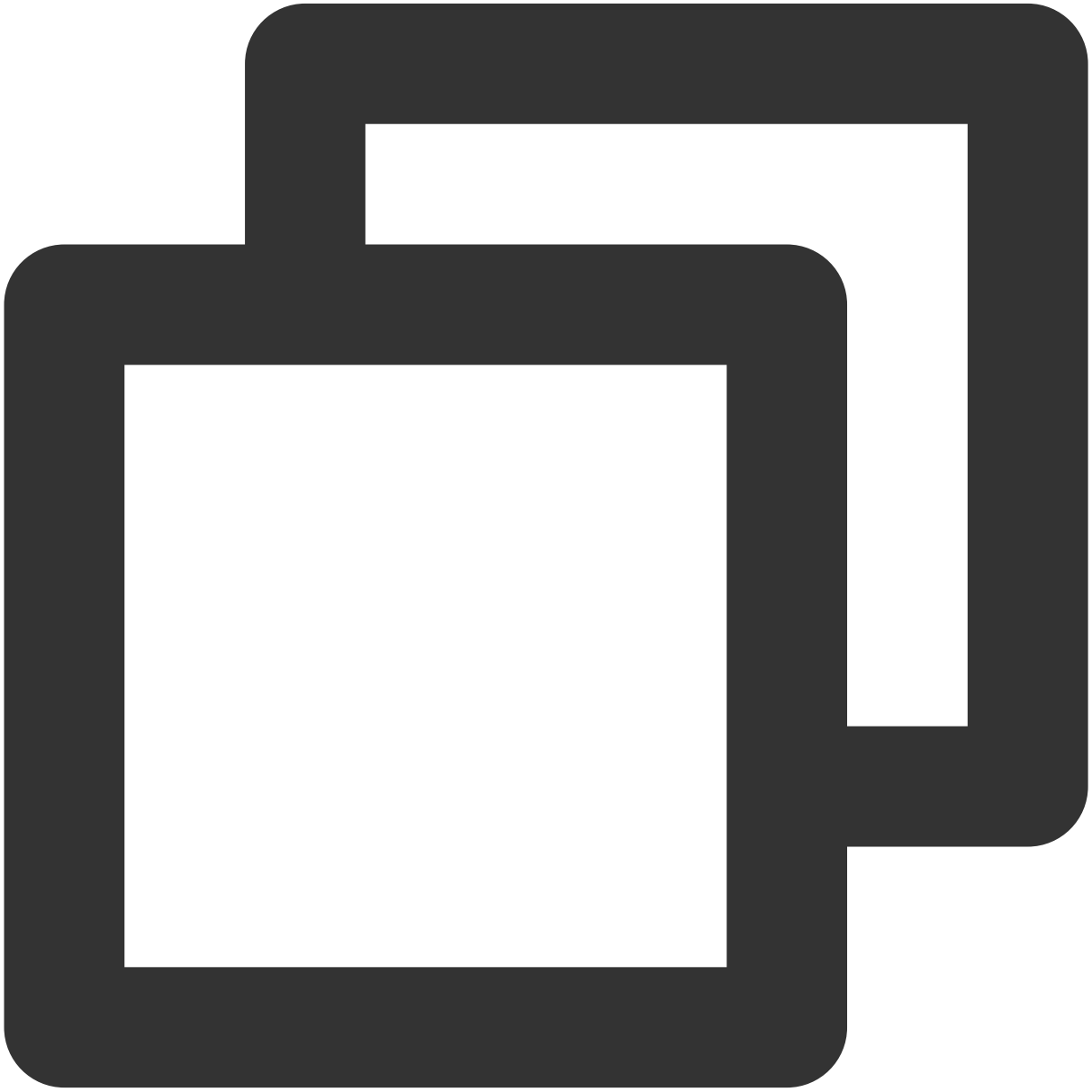


```
tencent_chat_module/
```

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

汎用modelクラスモジュール

以下に示すように、`common/common_model.dart` ファイルを新規作成し、Flutterとネイティブアプリケーションの通信仕様を定義する2つのclassを作成します。



```
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モジュール

まずIMエンジンを作成します。このモジュールのすべてのコードとファイルは `lib/chat` ディレクトリにあります。

1. model.dartという名前のグローバル状態管理Modelを新規作成します。

このModelはTencent Cloud IM Flutterモジュールのマウント、初期化、管理、オフラインプッシュ機能、グローバル状態管理、Nativeとの間の通信の維持を行うために使用されます。

Chatモジュール全体の中心となるものです。

詳細コードについてはDemoソースコードをご参照ください。次の3つの部分に重点を置きます：

Future_handleMessage(MethodCall call)：ログイン情報やクリックプッシュイベントなど、Nativeからのイベントを動的に監視します。

Future handleClickNotification(Map<String, dynamic>　msg)：通知をクリックしてイベントを処理します。Nativeからのパススルー、Mapからのデータの取り出し、特定のセッションなど、対応するサブモジュールへジャンプします。

Future<void> initChat()：Tencent Cloud IMを初期化/Tencent Cloud IM/にログイン/オフラインプッシュの初期化とToken送信を完了します。このメゾットはスレッドロック機構を使用し、同時に1つしか実行できなく、初期化が成功した後、繰り返し実行しないことを保証します。

説明：

オフラインプッシュの導入ガイドラインに従って、ベンダーのオフラインプッシュ機能の導入を完了してから、プッシュTokenを正常に送信し、プッシュ機能を使用してください。

2. Chatモジュールのメインエントリー用に新しいchat_main.dartファイルを作成します。

このページは、Flutter Chatモジュールの最初のページでもあります。

Demoで、このページはログインする前に読み込み状態であり、ログインするとセッションリストが表示されます。

また、ここでdidChangeAppLifecycleState監視とフォアグラウンドバックグラウンド切り替えイベントの送信を行う必要があります。詳細については、オフラインプッシュプラグインのドキュメント手順5をご参照ください。

詳細コードについて、Demoソースコードをご参照ください。

3. オフラインプッシュプラグイン機能を単一のインスタンスで管理するために、新しい `push.dart` ファイルを作成します。Tokenの取得・送信/プッシュ権限の取得などの操作に使用されます。詳細コードについてはDemoソースコードをご参照ください。

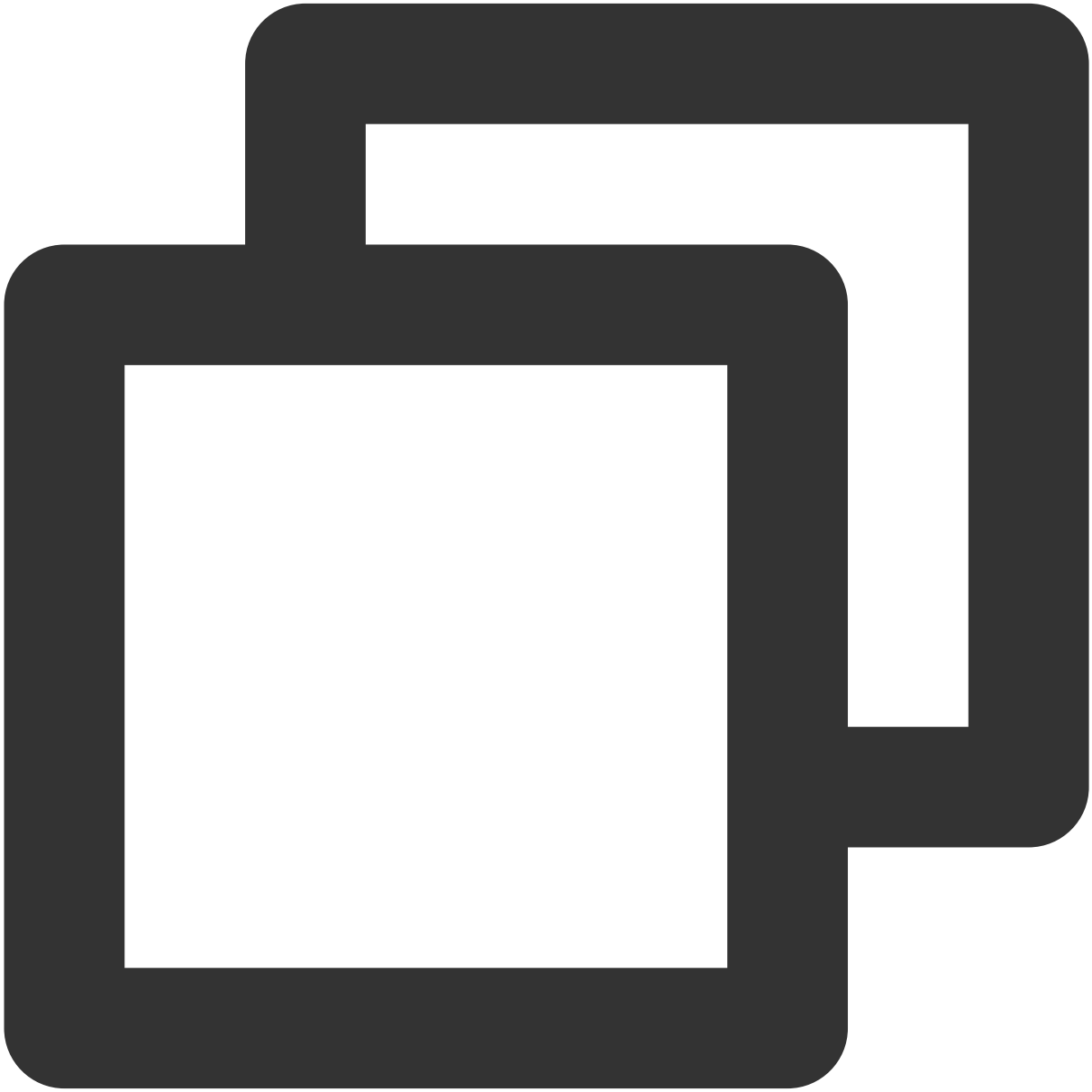
4. TUIKitのセッションモジュールコンポーネント `TIMUIKitConversation` を搭載するための新しい `conversation.dart` ファイルを作成します。詳細コードについてはDemoソースコードをご参照ください。

5. TUIKitの履歴メッセージリストを記載し、メッセージモジュールコンポーネント `TIMUIKitChat` を送信するための新しい `chat.dart` ファイルを作成します。

　このページには、「Profile」および「Group Profile」ページにジャンプする機能もあります。

　詳細コードについてはDemoソースコードをご参照ください。

6. TUIKitのユーザー情報とリレーショナルチェーン管理モジュールコンポーネント `TIMUIKitProfile` を記載するための新しい `user_profile.dart` ファイルを作成します。詳細コードについてはDemoソースコードをご参照ください。
7. TUIKitのグループ情報とグループ管理モジュールコンポーネント `TIMUIKitGroupProfile` を搭載するための `group_profile.dart` ファイルを新規作成します。詳細コードについてはDemoソースコードをご参照ください。
これで、Chatモジュールの開発が完了しました。最終的な構造は次のとおりです：



```
tencent_chat_module/
├── lib/
```

```
|   └── call/
|       └── chat.dart
|       └── model.dart
|       └── chat_main.dart
|       └── push.dart
|       └── conversation.dart
|       └── user_profile.dart
|       └── group_profile.dart
|   └── chat/
|   └── common/
```

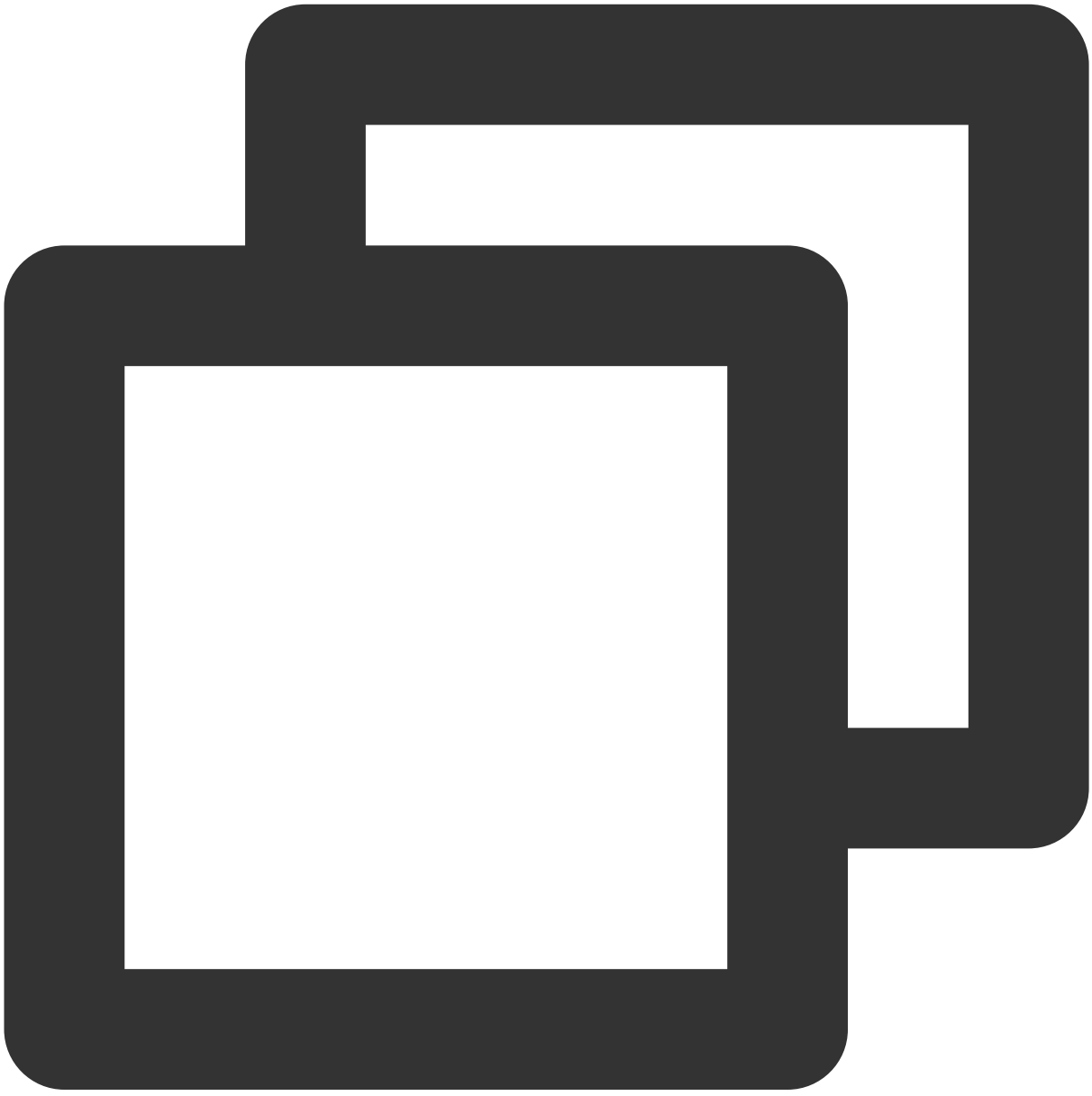
Callモジュール

このモジュールは、オディオビデオ通話プラグインによって提供されるオディオビデオ通話機能を搭載するために使用されます。
このモジュールの中核は、リスナーが新たな通話の招待を受けると、Nativeメソッドを呼び出すことで、自動的に通話ページをポップアップするとともに、Native経由で転送されたChatモジュールからの通話要求を受け、自発的に通話を開始します。
まずIMエンジンを作成します。このモジュールのすべてのコードとファイルは `lib/call` ディレクトリにあります。
1. model.dartという名前のグローバル状態管理Modelを新規作成します。
このModelはオディオビデオ通話プラグインのマウント、初期化、管理、グローバル状態管理、Nativeとの間の通信維持を行います。
Callモジュール全体のコアです。
詳細コードについてはDemoソースコードをご参照ください。次の2つの部分に重点を置きます。
_onRtcListener=TUICallingListener(...)：通話イベントを定義するリスナーは、Method Channelを介してNative層に通知し、Callモジュールが属するViewController(iOS)/Activity(Android)のフロントエンドを表示するかどうかを動的に制御します。
Future_handleMessage(MethodCall call)：Nativeからのアクティブな通話開始要求、Callモジュールからの呼び出しを動的に監視し、自主的に通話を開始します。
2. Callモジュールのメインエントリー用に新しい `call_main.dart` ファイルを作成します。
このコンポーネントは、オディオ・ビデオ通話プラグインのバインドに必要なnavigatorKeyを注入するために使用されます。
詳細コードについてはDemoソースコードをご参照ください。

各Flutterエンジンのエントリーを設定

上記の3つのモジュールを開発した後、Flutterエンジンへのエントリーとして、最終的に外部に露出するmainメソッドを完成させます。
1. デフォルトのエントリー
`lib/main.dart` ファイルを開き、 `main()` メソッドを空のMaterialAppに変更します。

このメソッドは、Flutter Moduleへのデフォルトのエントリーとして、Flutterマルチエンジン、FlutterEngineGroupで管理されている背景で、サブFlutter Engineがなければ何もentry pointが設定されない限り、このメソッドは利用されません。
例えば、このデフォルトの `main()` メソッドは、私たちのシナリオでは使用されていません。



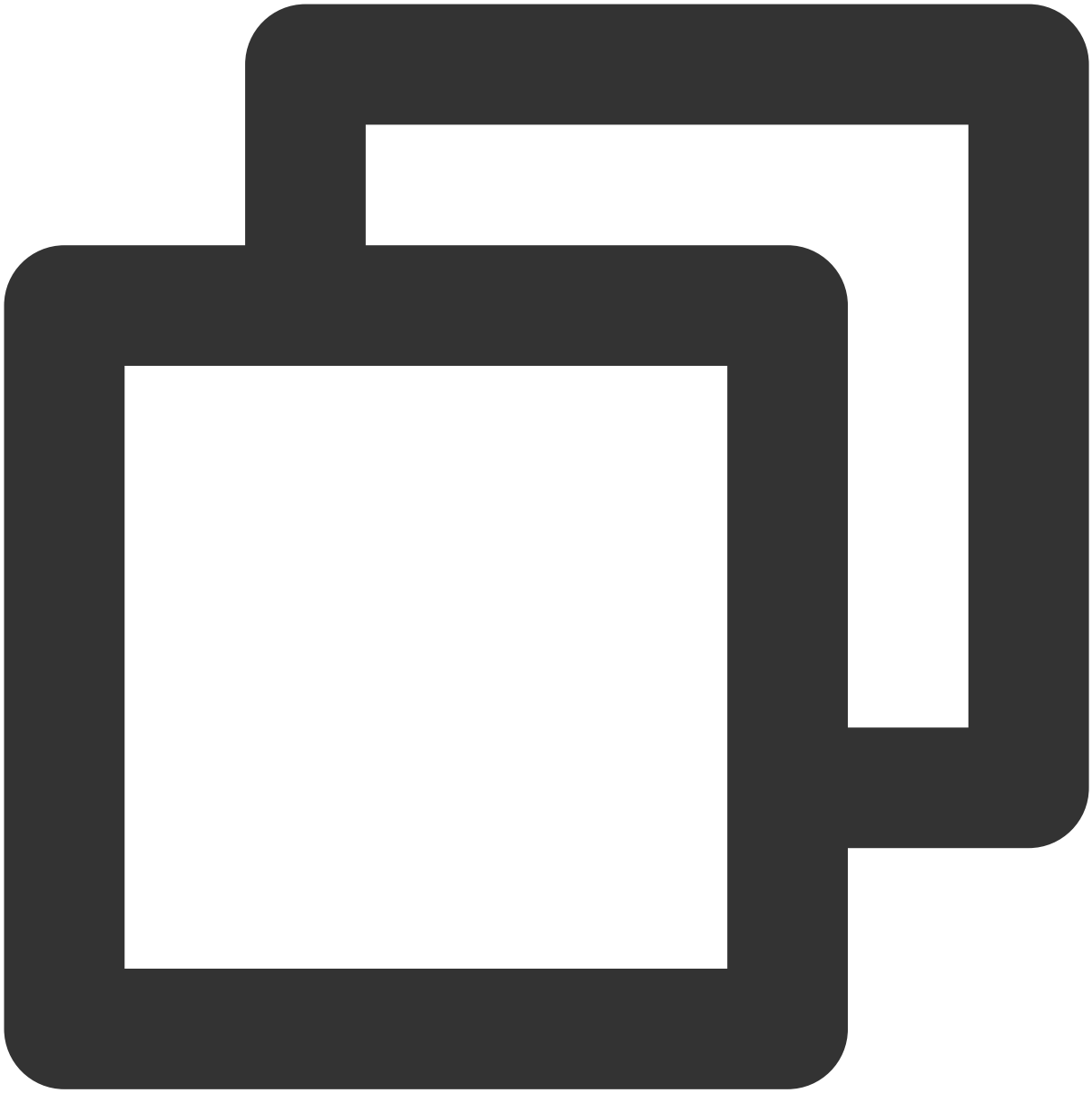
```
void main() {
  WidgetsFlutterBinding.ensureInitialized();

  runApp(MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
```

```
        primarySwatch: Colors.blue,
      ),
      home: Container(),
    ));
  }
```

2. Chatモジュールのエントリーを設定

`@pragma('vm:entry-point')` コメントを使用して、このメソッドを `entry-point` としてマークします。メソッド名 `chatMain` は、このエントリの名前であり、Nativeでは、対応するFlutterエンジンを作成するためにも使用されます。

グローバルな `ChangeNotifierProvider` 状態管理を使用して、 `ChatInfoModel` データとビジネスロジックを維持します。



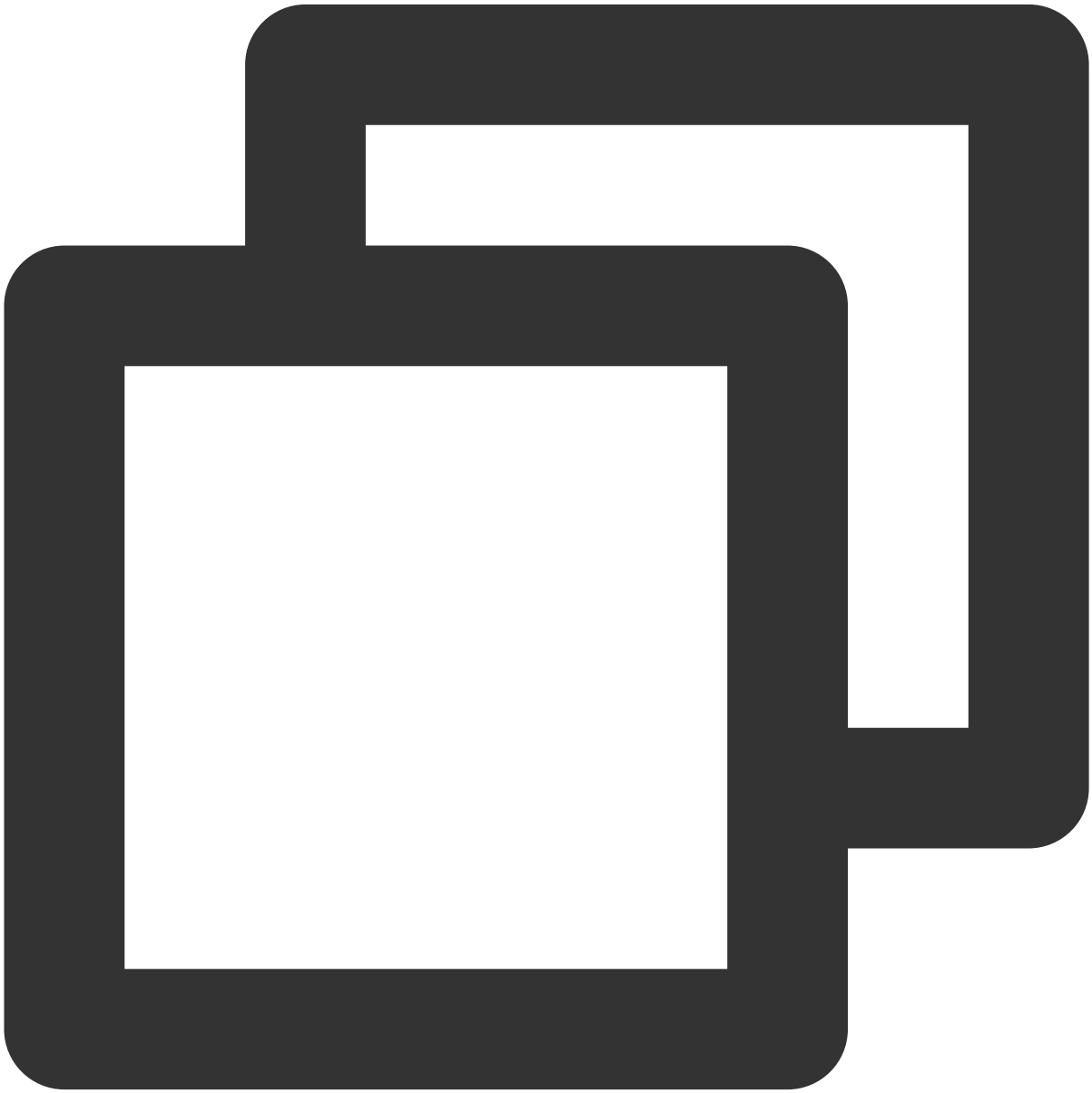
```
@pragma('vm:entry-point')
void chatMain() {
  // This call ensures the Flutter binding has been set up before creating the
  // MethodChannel-based model.
  WidgetsFlutterBinding.ensureInitialized();

  final model = ChatInfoModel();

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

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

3. Callモジュールのエントリーを設定

同様に、このエントリーは `callMain` という名前が付けられています。

グローバルな `ChangeNotifierProvider` 状態管理を使用して、 `CallInfoModel` データとビジネスロジックを維持します。



```
@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(),
    ),
  );
}
```

これでFlutter Moduleの部分のDartコードが作成されました。
次に、Nativeコードの作成を開始します。

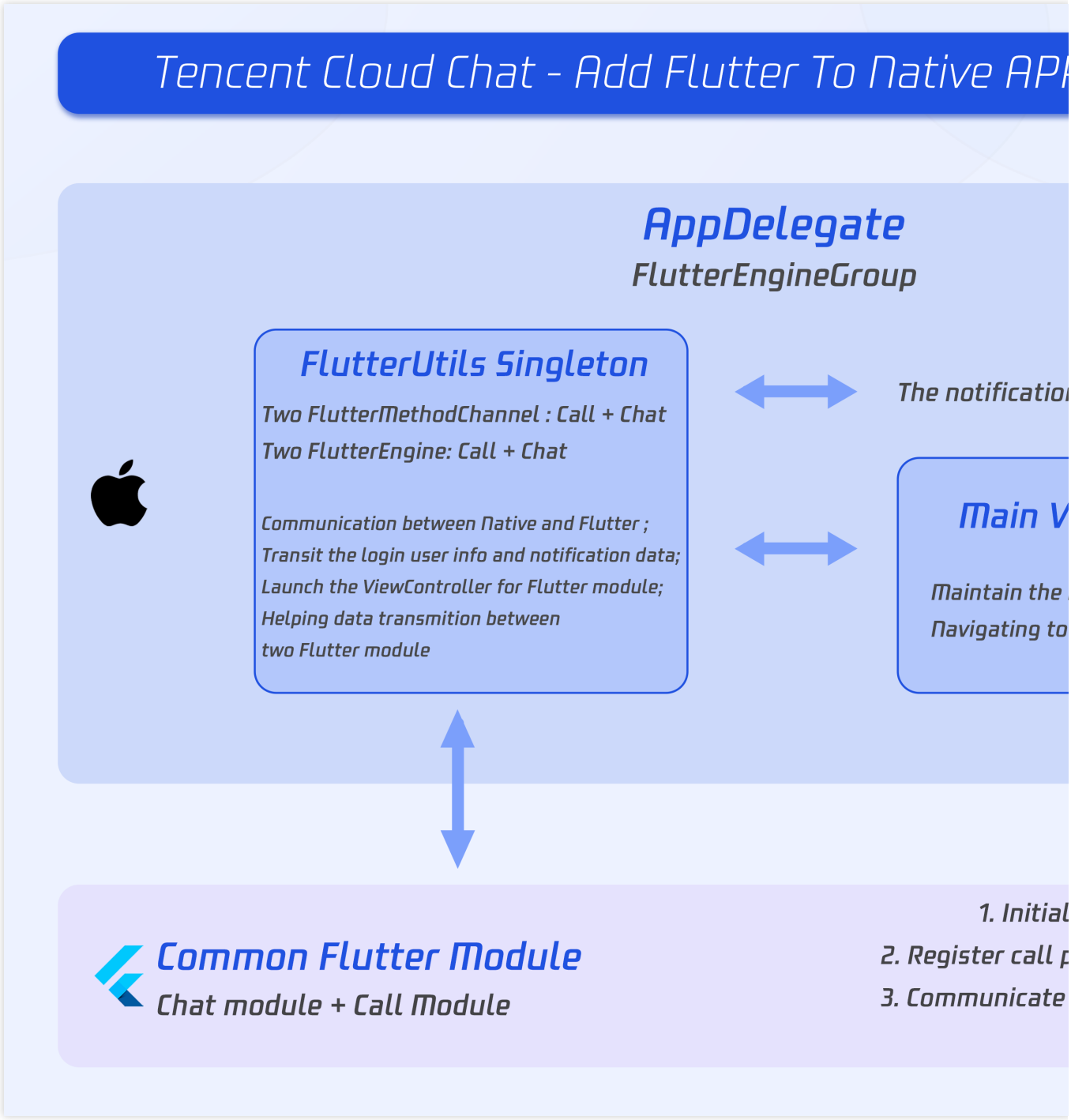
iOS Native開発

このドキュメントでは、Swift言語を例にします。
説明：
次のコード構造は参照用です。必要に応じて柔軟に設定できます。
iOSのプロジェクトディレクトリに移動します。
もし既存のアプリケーションが `MyApp` とし、Podfileがなければ、CocoaPods入門ガイドラインを参照して `Podfile` をプロジェクトに追加してください。

Flutter Moduleの導入

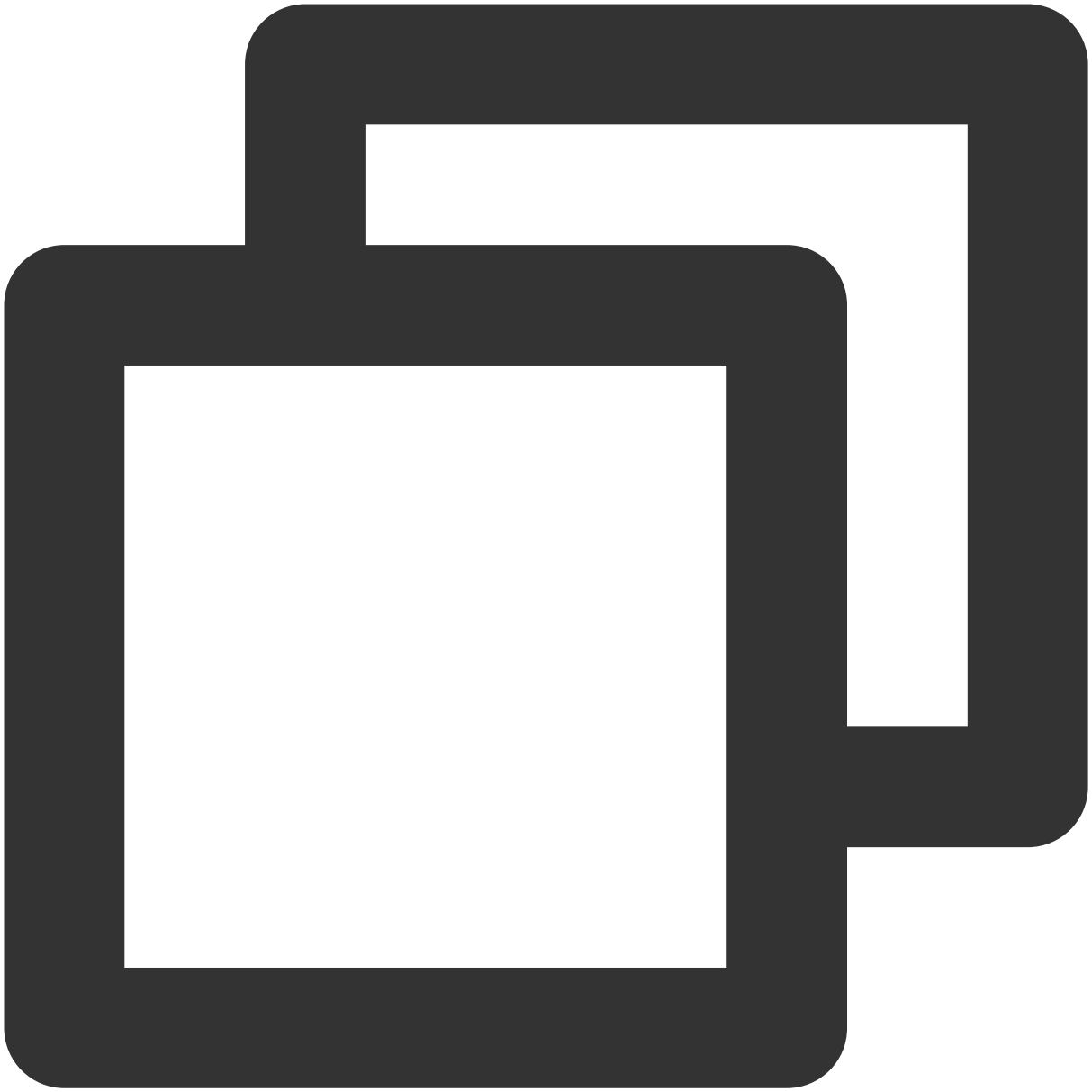
ネイティブアプリケーションにFlutter moduleを導入するには、この部分をご参照ください。方式1を使用することをお勧めします。

iOSプロジェクトで、Flutterエンジンを管理



複数のエンジンインスタンスを統合管理する `FlutterEngineGroup` を作成します。

`AppDelegate.swift` ファイルに、次のコードを追加します：



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

Flutterエンジンを管理する単一のインスタンスオブジェクトを作成します。

このSwift単一のインスタンスオブジェクトは、Flutterインスタンスの一元管理に使用され、プロジェクト内のあらゆる場所で直接呼び出すことができます。

Demoコードのロジックは、新しいルーティングを使用し、ChatのViewControllerを搭載します。CallのViewControllerは、presentとdismissの動的ウィンドウでメンテナンスを行います。
新しい `FlutterUtils.swift` ファイルを作成し、コードを書きます。この部分の詳細コードについてDemoソースコードをご参照ください。
重点は次のとおりです：
private override init()：Flutterエンジンの各インスタンスを初期化し、Method Channelを登録し、イベントを監視します。
func reportChatInfo()：Flutter層を初期化してTencent Cloud IMにログインできるよう、ユーザーログイン情報とSDKAPPIDをFlutter Moduleにパススルーします。
func launchCallFunc()：CallのFlutterページを引き上げるために使用されます。Callモジュールが通話招待を受けるときにトリガーされ、またはChatモジュールが自主的な通話を開始するときにトリガーされます。
func triggerNotification(msg: String)：iOS　Native層が受信したオフラインプッシュメッセージクリックイベントとそれに含まれるext情報を、JSON String形式でFlutter層にバインドされた監視処理イベントに送信します。例えば、対応するセッションへのオフラインプッシュクリックジャンプを処理するために使用されます。
オフラインプッシュクリックイベントの監視と転送
オフラインプッシュの初期化/Token送信/クリックイベントに対するセッションジャンプ処理は、Flutter Chatモジュールで行われているので、Native領域はクリック通知イベントのextをパススルーするだけでよいです。
これは、クリック通知イベントがNativeでブロックされて消費されているため、Flutter層が直接取得することができず、Native経由で転送しなければならないからです。
`AppDelegate.swift` ファイルに、次のコードを追加します。具体的なコードについてはDemoソースコードをご参照ください。

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

  override func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicat

      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: UNNotificationRe
        @escaping () -> Void) {
        let notificationExt: Dictionary = response.notification.request.content.userInfo
        let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
        FlutterUtils.shared.triggerNotification(msg: notificationExtString)
    }

}
```

この時点で、iOS Native層の作成が完了しました。

Android Native開発

このドキュメントはKotlin言語を例にします。

説明：

次のコード構造は参照用です。必要に応じて柔軟に設定できます。

Flutter Moduleの導入

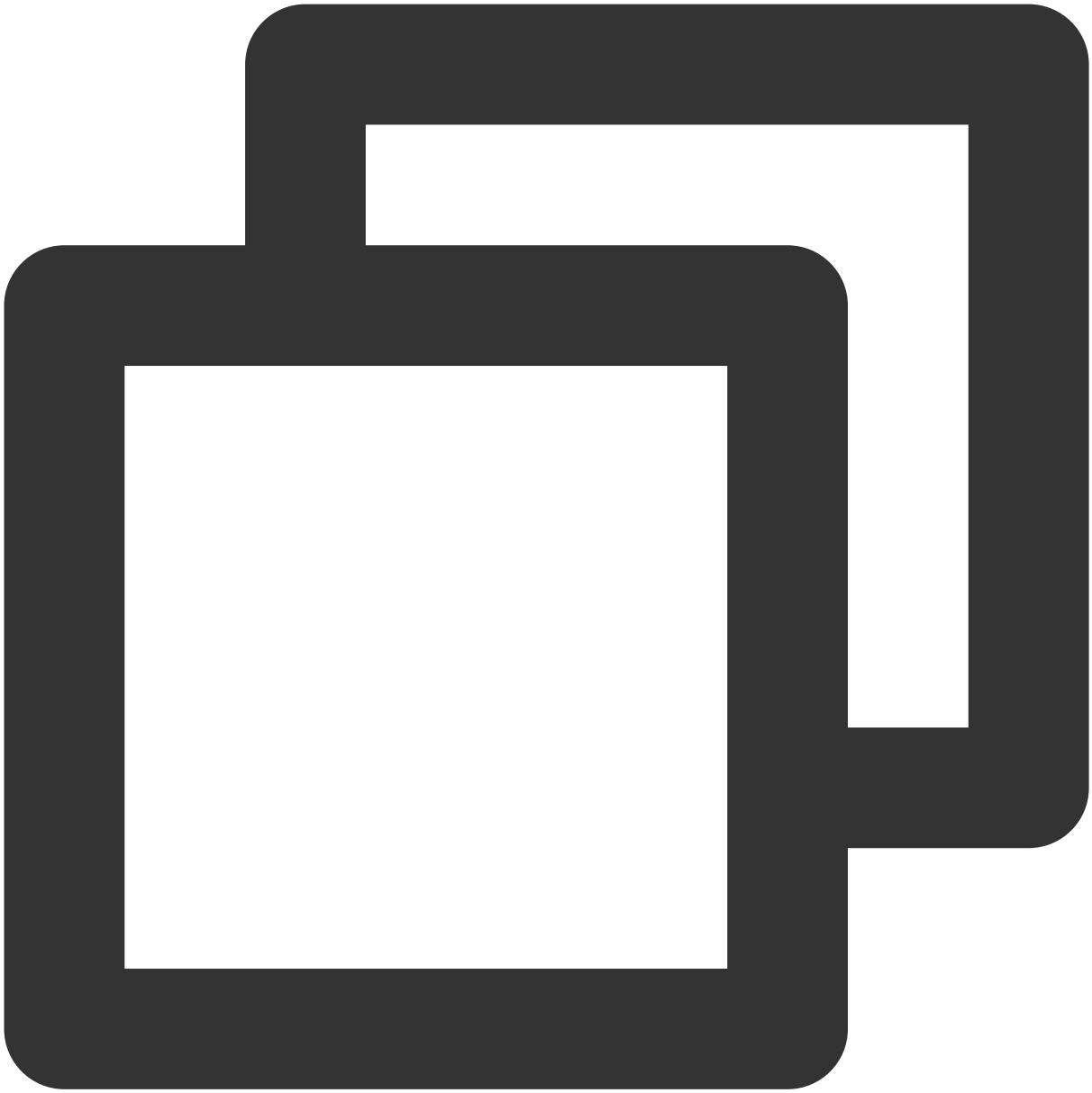
ネイティブアプリケーションにFlutter moduleを導入するには、[この部分]（#android）をご参照ください。方式2を使用することをお勧めします。

AndroidプロジェクトでFlutterエンジンを管理

Flutterエンジンを管理する単一のインスタンスオブジェクトを作成します。

このKotlin単一のインスタンスオブジェクトは、Flutterインスタンスの一元管理に使用され、プロジェクト内のあらゆる場所で直接呼び出すことができます。

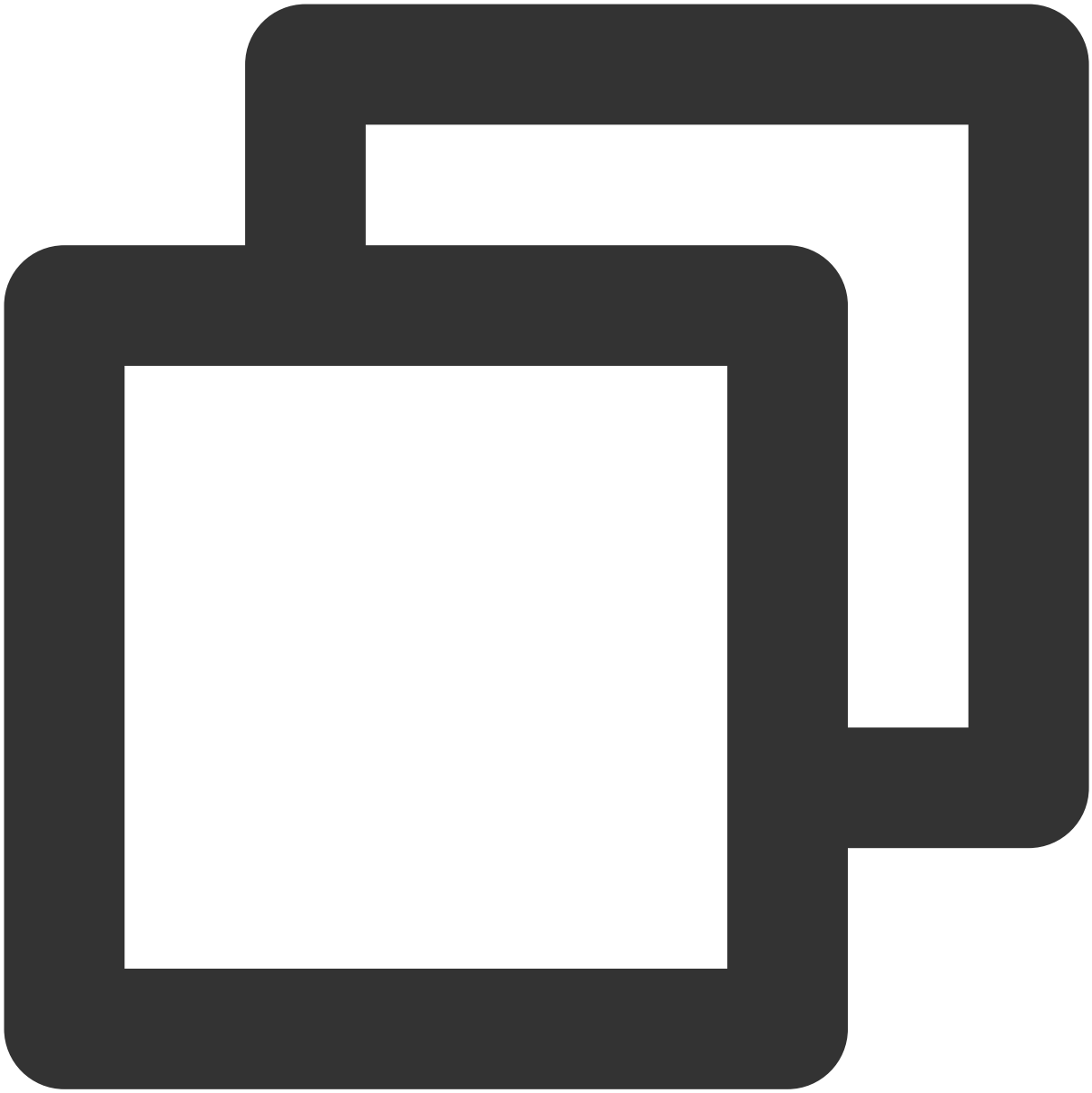
新しい `FlutterUtils.kt` ファイルを作成し、 `FlutterUtils` 静的クラスを定義します。



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

「FlutterEngineGroup」(Flutterエンジングループ)を作成して、複数のエンジンインスタンスを統合管理します。

「FlutterUtils.kt」ファイルで、「FlutterEngineGroup」を定義し、各Flutter Engineインスタンスと Method Channelをセットにして、初期化時に初期化します。



```
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

// 初期化
flutterEngines = FlutterEngineGroup(context)
...
```

Flutterエンジンを管理するために使用される単一のオブジェクトを完成させます。

Demoコードのロジックは新しいルーティングを使用して、ChatとCallのActivityを搭載することです。

ChatのActivityはユーザーが自主的に入ったり退出したりします。CallのActivityは、リスナーまたはアクティブな外部発呼で自動的にナビゲーションにより入ったり退出します。

重点は次のとおりです：

fun init()：Flutterエンジンの各インスタンスを初期化し、Method Channelを登録し、イベントを受信します。

fun reportChatInfo()：Flutter層を初期化してTencent Cloud IMにログインできるよう、ユーザーログイン情報とSDKAPPIDをFlutter Moduleにパススルーします。

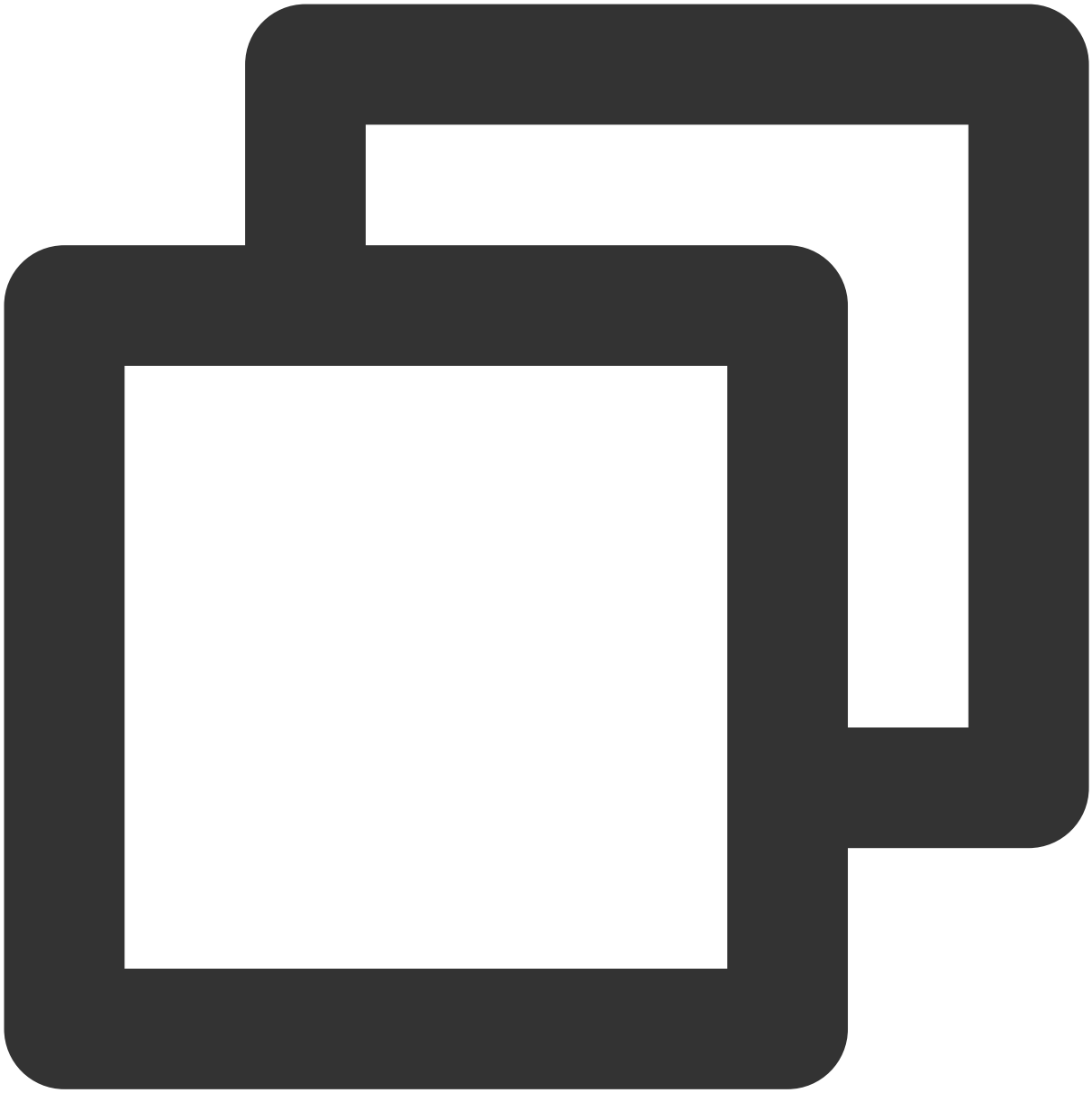
fun launchCallFunc()：CallのFlutterページを引き上げるために使用されます。Callモジュールが通話招待を受けるときにトリガーされ、またはChatモジュールが自主的な通話を開始するときにトリガーされます。

fun triggerNotification(msg: String)：iOS　Native層が受信したオフラインプッシュメッセージクリックイベントとそれに含まれるext情報を、JSON String形式でFlutter層にバインドされた監視処理イベントに送信します。例えば、対応するセッションへのオフラインプッシュクリックジャンプを処理するために使用されます。

この単一インスタンスobjectの詳細コードについては、Demoソースコードをご参照ください。

マスターエントリー `MyApplication` で、上記のオブジェクトを初期化します

`MyApplication.kt` ファイルで単一インスタンスオブジェクトにグロバールcontextを渡し、初期化を実行します。



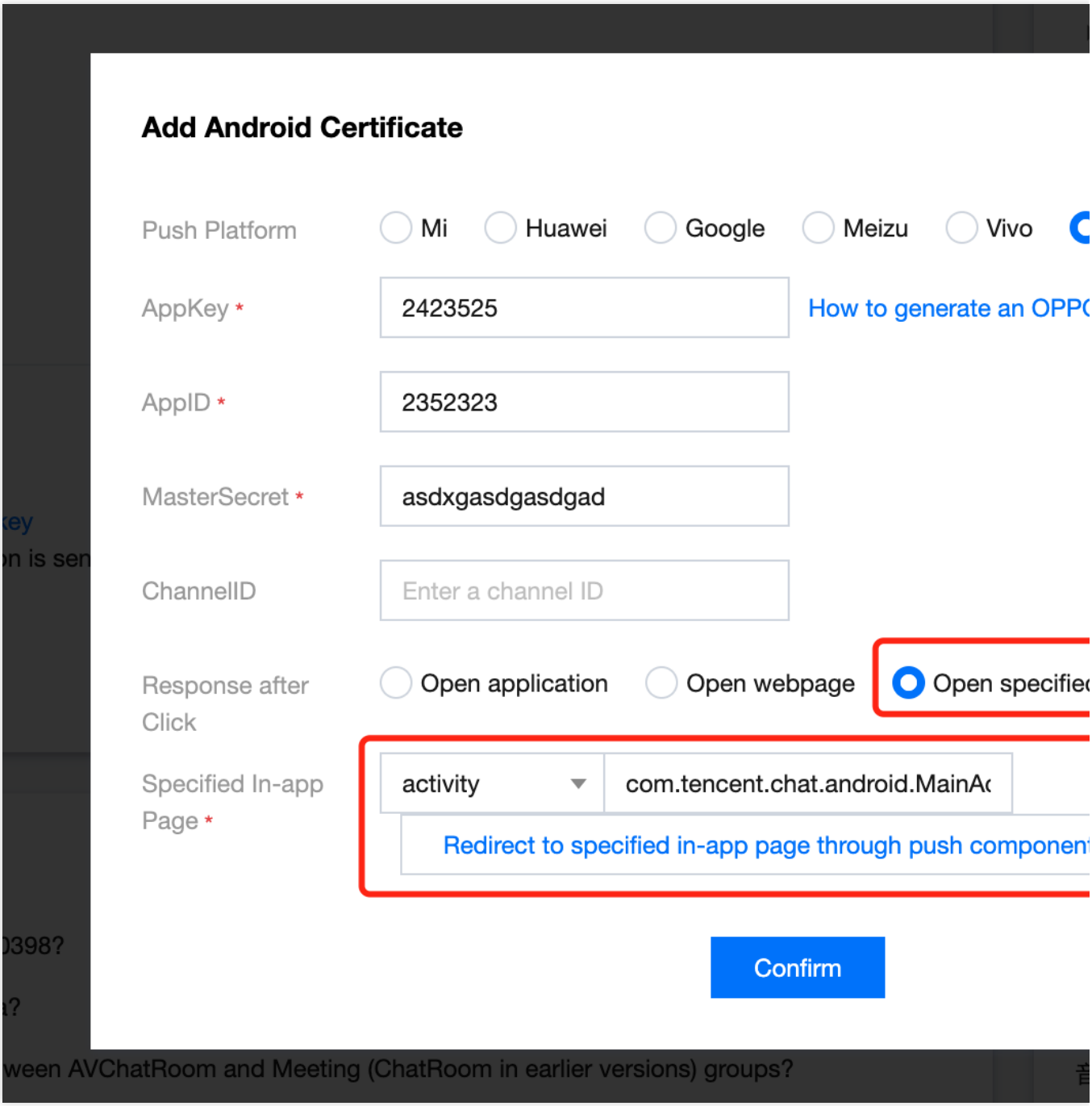
```
class MyApplication : MultiDexApplication() {

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

オフラインプッシュクリックイベントの監視と転送

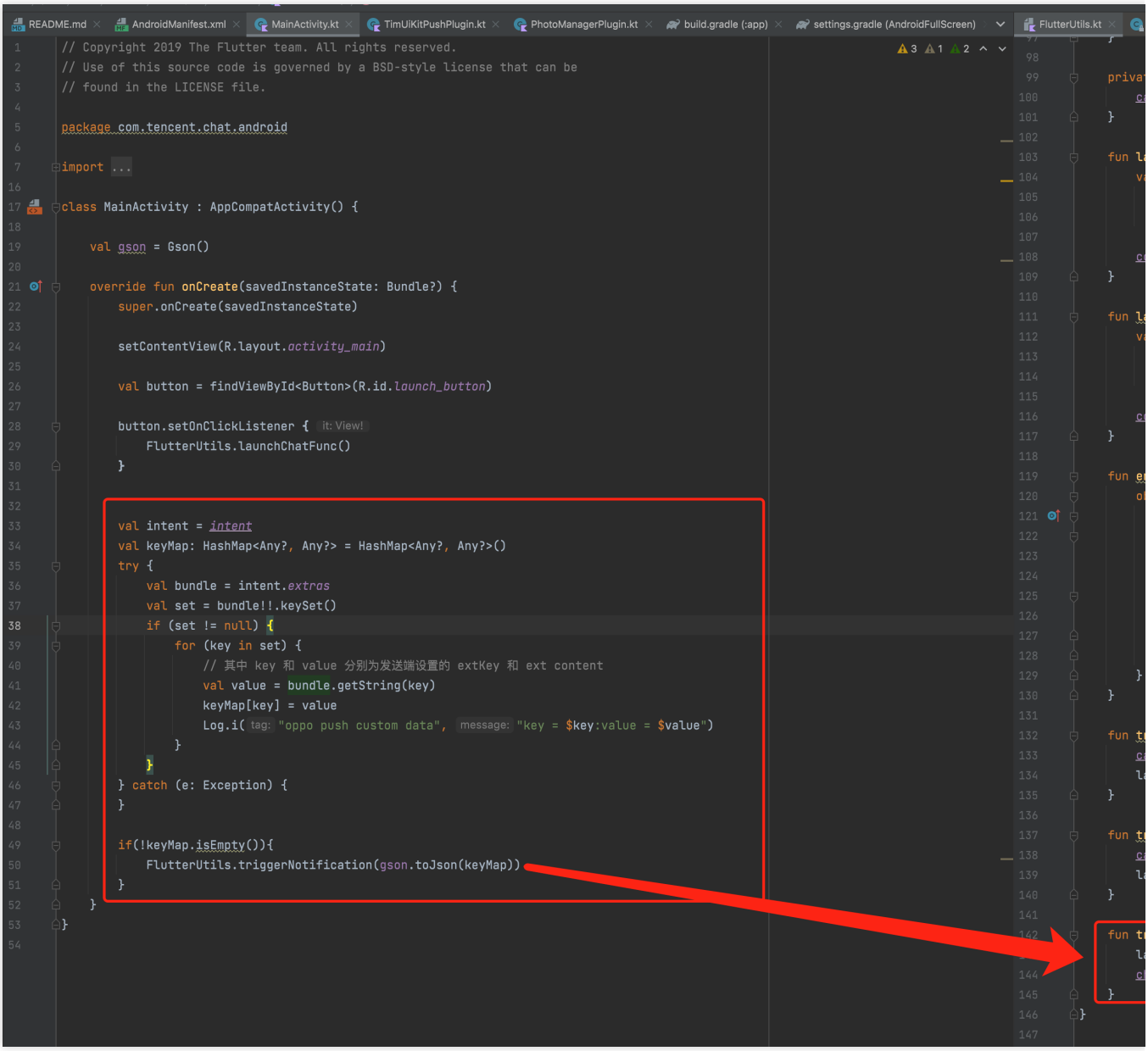
オフラインプッシュの初期化/Token送信/クリックイベントに対するセッションジャンプ処理は、Flutter Chatモジュールで行われているので、Native領域はクリック通知イベントのextをパススルーするだけでです。
これは、クリック通知イベントがNativeでブロックされて消費されているため、Flutter層が直接取得することができず、Native経由で転送しなければならないからです。

ご注意：

オフラインプッシュの導入手順はベンダーによって異なるため、このドキュメントではOPPOを例にします。すべてのベンダーの導入方法についてはこのドキュメントをご参照ください。
Tencent Cloud IMコンソールでは、OPPOのプッシュ証明書が追加され、`次のアクションをクリック`して`アプリケーション内の指定されたページを開く`を選択し、`アプリケーション内のページ`で`Activity`方式を用いて、オフラインプッシュ情報を処理するためのページを設定します。アプリケーションの最初のページにすることをお勧めします。例えば、Demoは`com.tencent.chat.android.MainActivity`に設定されます。



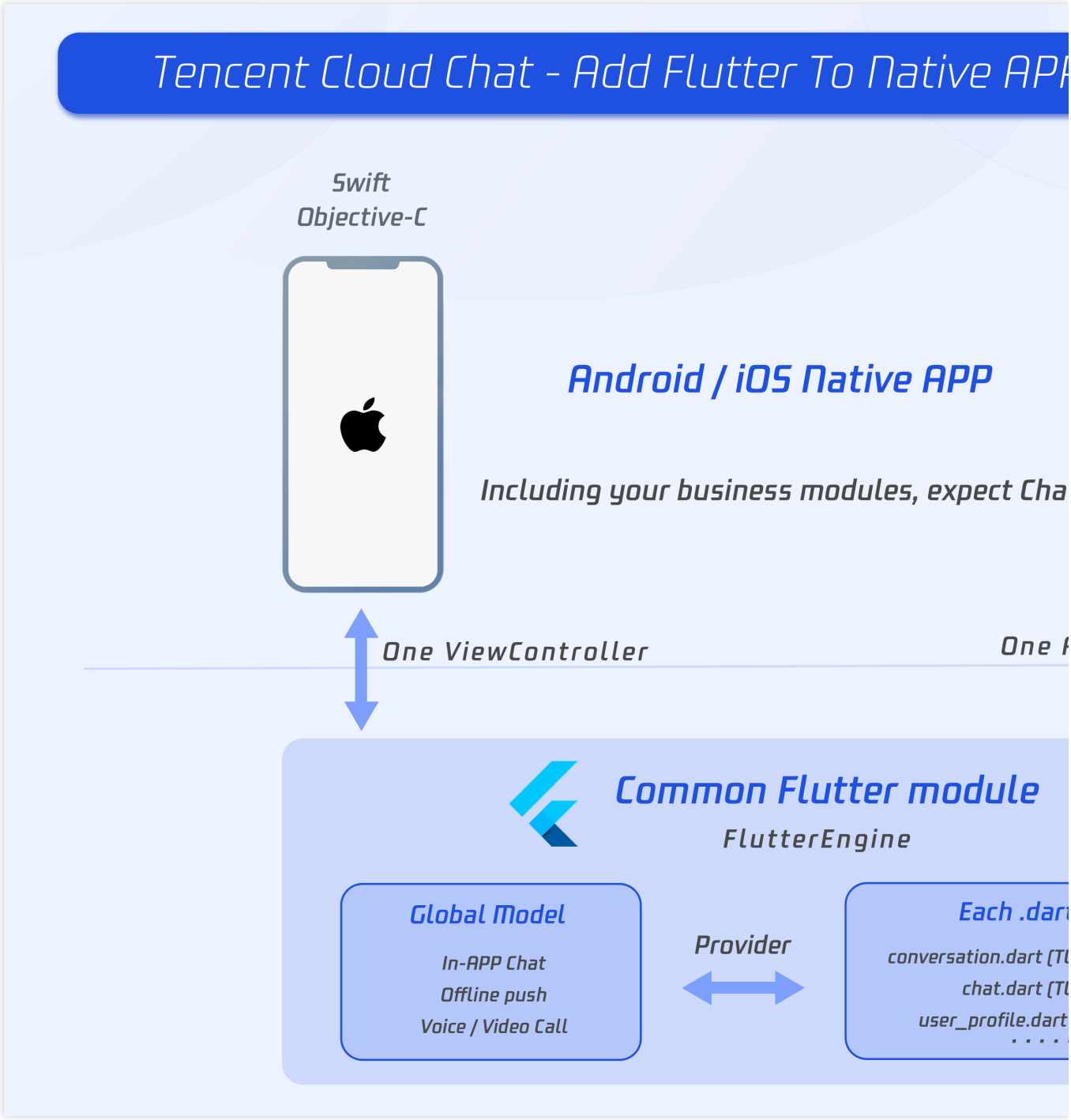
上のコンソールで設定されたオフラインプッシュ用のActivityファイルに、次のコードを追加します。
このコードの役割は、ベンダーが該当するActivityを引き上げると、BundleからHashMap形式のextメッセージを取り出し、単一のインスタンスオブジェクト中のメソッドをトリガーし、そのメッセージを手動でFlutterに転送することです。具体的なコードについてはDemoソースコードをご参照ください。



この時点で、Android Native層の作成が完了しました。

ソリューション2：Flutter単一エンジンソリューション

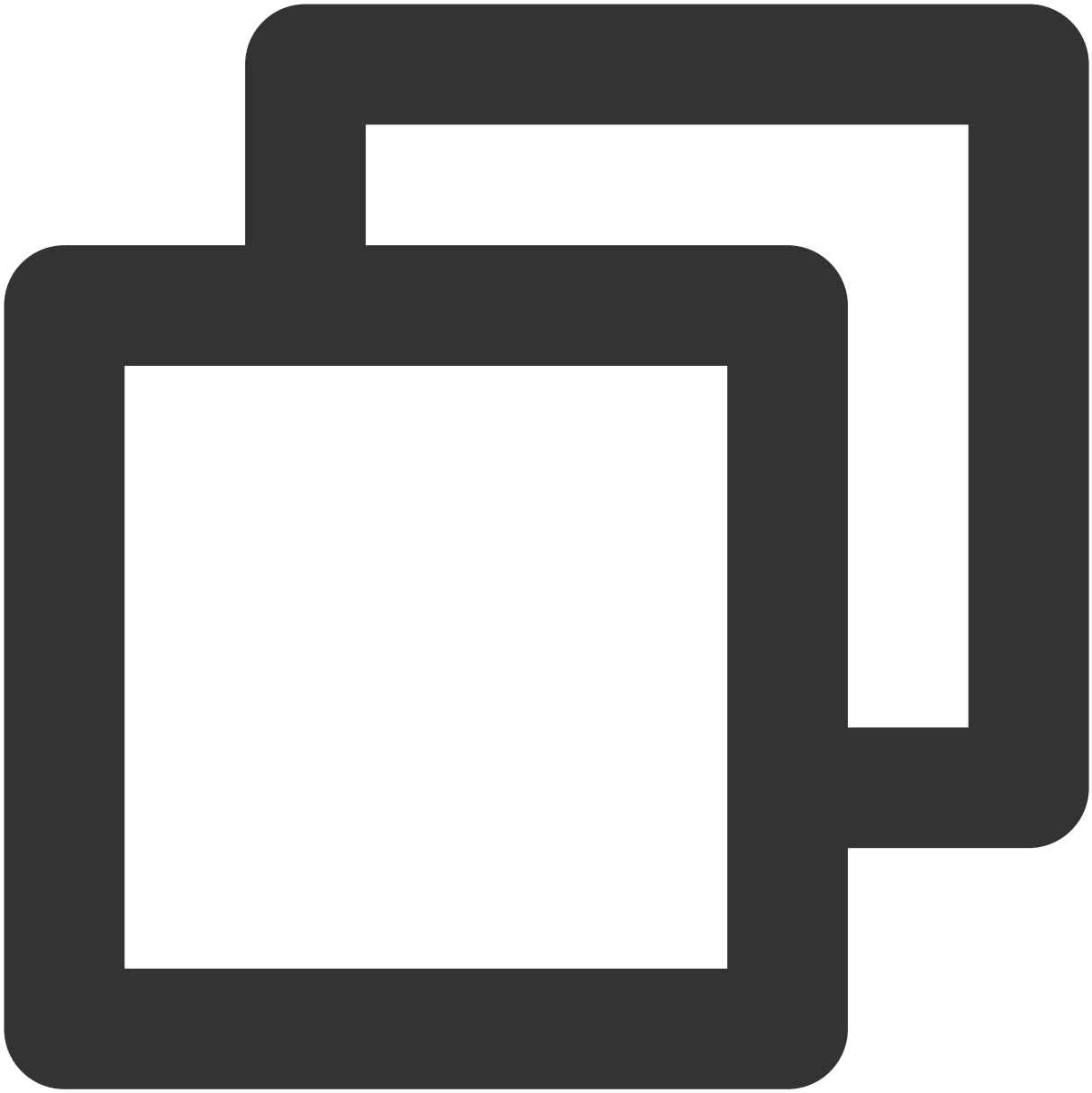
このソリューションは、ChatモジュールとCallモジュールを、同じFlutterエンジンインスタンスに記述します。



この2つのモジュールは、Flutterエンジンを維持するだけで、同時に表示および非表示にすることができます。

Flutter Moduleの開発

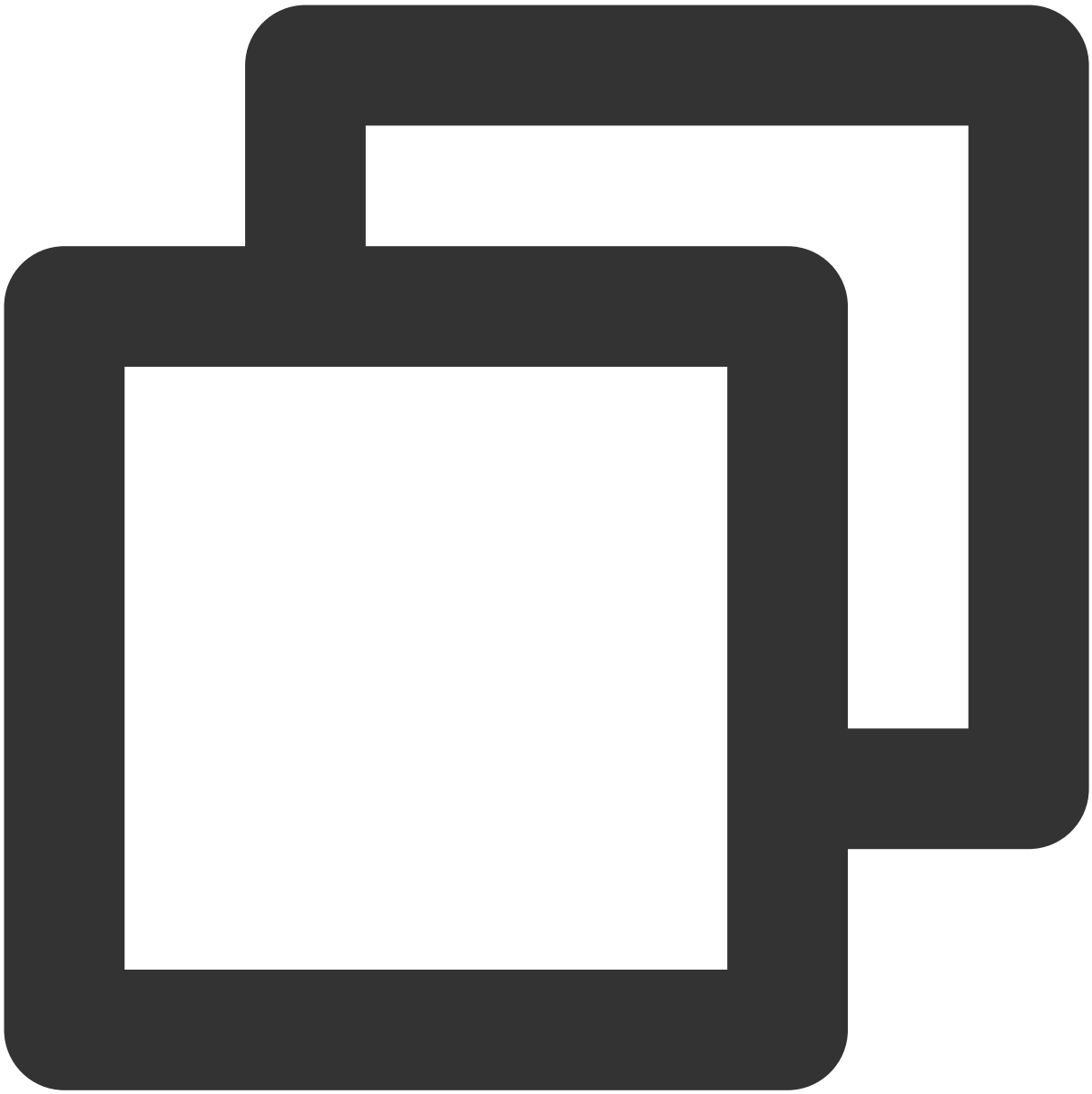
既存のアプリケーションにFlutterを埋め込むには、まずFlutterモジュールを作成します。
プロジェクトのルートディレクトリの外層で次を実行します。



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

これにより、some/path/tencent_chat_module/にFlutterモジュールプロジェクトが作成されます。このディレクトリでは、 `flutter run--debug` や `flutter buildios` など、他のFlutterプロジェクトと同じFlutterコマンドを実行できます。FlutterおよびDartプラグインを使用して、Android Studio、IntelliJまたはVS Codeでこのモジュールを実行することもできます。このプロジェクトには、既存のアプリケーションに埋め込む前にモジュールの単一ビューのサンプルバージョンが含まれています。これは、コードのFlutterのみの部分をテストするのに役立ちます。

`tencent_chat_module` モジュールディレクトリ構造は、通常のFlutterアプリケーションに似ています。



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

これで `lib/` に、コードを書くことができるようになりました。

main.dart

`main.dart` ファイルを変更して、TUIKit、オフラインプッシュプラグインおよびオディオビデオ通話プラグインを導入します。
グローバル状態、我々のIM SDKおよびMethod ChannelとNativeが通信している状態は、 `ChatInfoModel` で管理されています。
Nativeからユーザ情報とSDKAPPIDを受信したら、 `_coreInstance.init()` および `_coreInstance.login()` を呼び出してTencent Cloud IMの初期化・ログインを行います。また、オーディオビデオプッシュプラグインおよびオフラインプッシュプラグインを初期化し、プッシュTokenの送信を完了します。
説明：
オフラインプッシュの導入ガイドラインに従って、ベンダーのオフラインプッシュ機能の導入を完了してから、プッシュTokenを正常に送信し、プッシュ機能を使用してください。
オディオビデオ通話プラグインの場合は、次の点に注意してください：
リスナーで新しい通話招待を受信したときにNativeメソッドを呼び出すと、Nativeはユーザーが現在Flutterモジュールのページにいるかどうかを検出します。いない場合は、フロントエンドのページを強制的にこのモジュールに調整して着信ページを表示する必要があります。
オフラインプッシュプラグインの場合は、次の点に注意してください：
通知をクリックしてイベントを処理します。Nativeからのパススルー、Mapからのデータの取り出し、特定のセッションなど、対応するサブモジュールへジャンプします。
トップページの制作を完了し、ログインしていないときにロードしたアニメーションを表示します。ログインに成功すると、セッション一覧ページが表示されます。
また、ここで `didChangeAppLifecycleState` の監視とフォアグラウンドバックグラウンド切り替えイベントの送信を完了する必要があります。詳細はオフラインプッシュプラグインドキュメント手順5をご参照ください。
詳細コードについてはDemoソースコードをご参照ください。

他のTUIKitモジュールの導入

1. 新しい `push.dart` ファイルを作成し、オフラインプッシュプラグイン機能を単一のインスタンスで管理します。Tokenの取得・送信/プッシュ権限の取得などの操作に使用されます。詳細コードについてはDemoソースコードをご参照ください。
2. TUIKitのセッションモジュールコンポーネント `TIMUIKitConversation` を搭載するための新しい `conversation.dart` ファイルを作成します。詳細コードについてはDemoソースコードをご参照ください。
3. TUIKitの履歴メッセージリストを記載し、メッセージモジュールコンポーネント `TIMUIKitChat` を送信するための新しい `chat.dart` ファイルを作成します。
   このページには、「Profile」および「Group Profile」ページにジャンプする機能もあります。
   詳細コードについてはDemoソースコードをご参照ください。

4. TUIKitのユーザー情報およびリレーショナルチェーン管理モジュールコンポーネント `TIMUIKitProfile` を搭載するための新しい `user_profile.dart` ファイルを作成します。詳細コードについてはDemoソースコードをご参照ください。
5. TUIKitのグループ情報とグループ管理モジュールコンポーネント `TIMUIKitGroupProfile` を搭載する `group_profile.dart` ファイルを新規作成します。詳細コードについてはDemoソースコードをご参照ください。
これで統一されたFlutter Moduleの開発が完了しました。

iOS Native開発

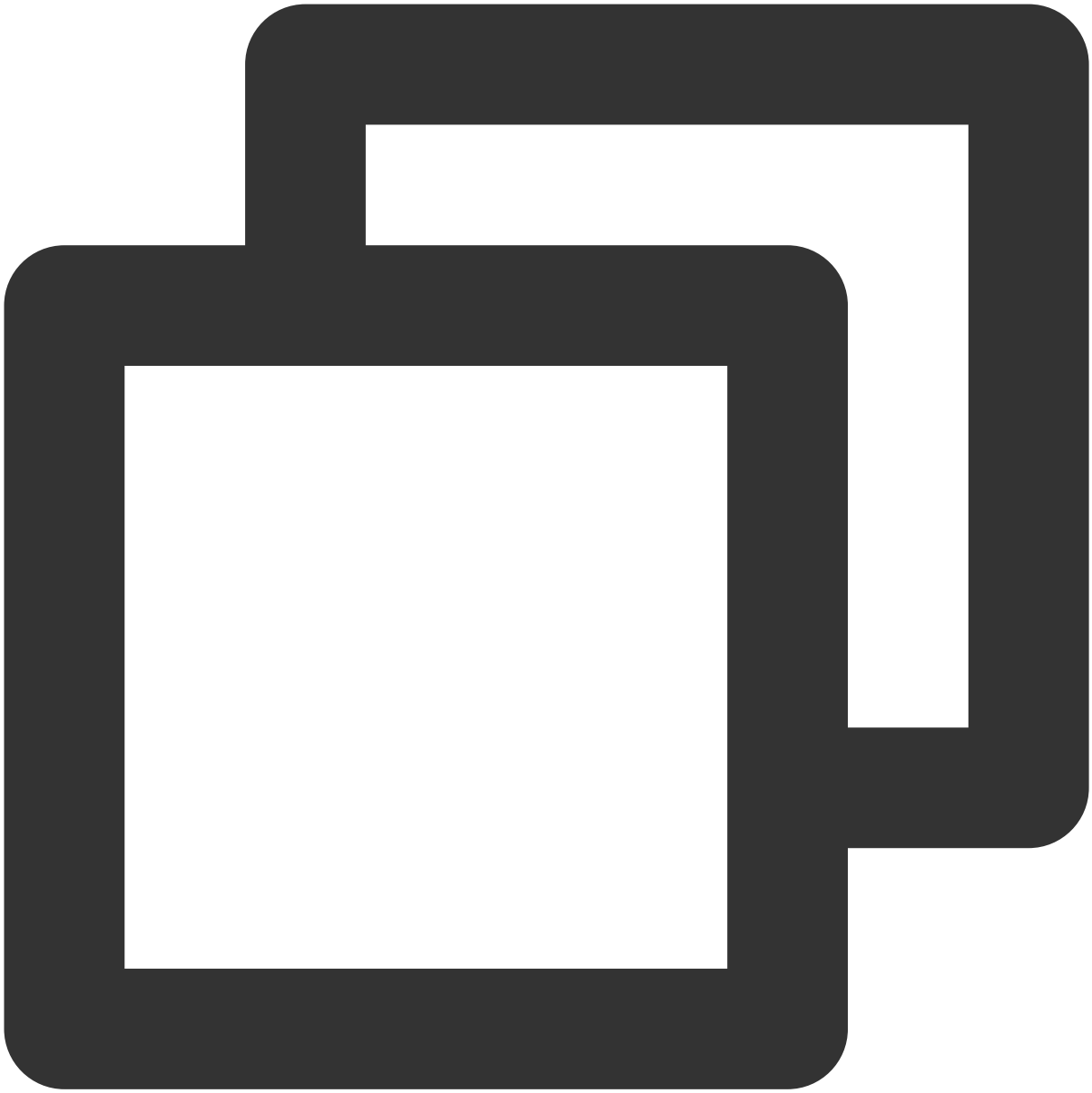
このドキュメントでは、Swift言語を例にします。
説明：
次のコード構造は参照用です。必要に応じて柔軟に設定できます。
iOSのプロジェクトディレクトリに移動します。
もしあなたの既存のアプリケーションが `MyApp` とし、Podfileがなければ、CocoaPods入門ガイドラインを参照して `Podfile` をプロジェクトに追加してください。

Flutter Moduleの導入

ネイティブアプリケーションにFlutter moduleを導入するには、この部分をご参照ください。方式1を使用することをお勧めします。

iOSプロジェクトで、Flutterエンジンを管理

FlutterEngineを作成します。
FlutterEngineを作成する場所は、メインアプリケーションのエントリーに固有です。例として、 `AppDelegate` 内のappの起動時にFlutterEngineを作成し、プロパティとして公開する方法を示しました。



```
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
  }
}
```

Flutterエンジンを管理する単一のインスタンスオブジェクトを作成します。

このSwift単一インスタンスオブジェクトは、Flutter Method Channelを一元管理し、Flutter Moduleと通信するための一連のメソッドを提供し、プロジェクト内のあらゆる場所で直接呼び出すことができます。

次のようなメソッドがあります：

private override init()：Method Channelを初期化し、イベントリスニングメソッドをバインドします。

func reportChatInfo()：Flutter層を初期化してTencent Cloud IMにログインできるよう、ユーザーログイン情報とSDKAPPIDをFlutter Moduleにパススルーします。

func launchChatFunc()：Flutter Moduleが置かれるViewControllerを引き上げるか、ナビゲーションします。

func triggerNotification(msg: String)：iOS　Native層が受信したオフラインプッシュメッセージクリックイベントとそれに含まれるext情報を、JSON String形式でFlutter層にバインドされた監視処理イベントに送信します。例えば、対応するセッションへのオフラインプッシュクリックジャンプを処理するために使用されます。

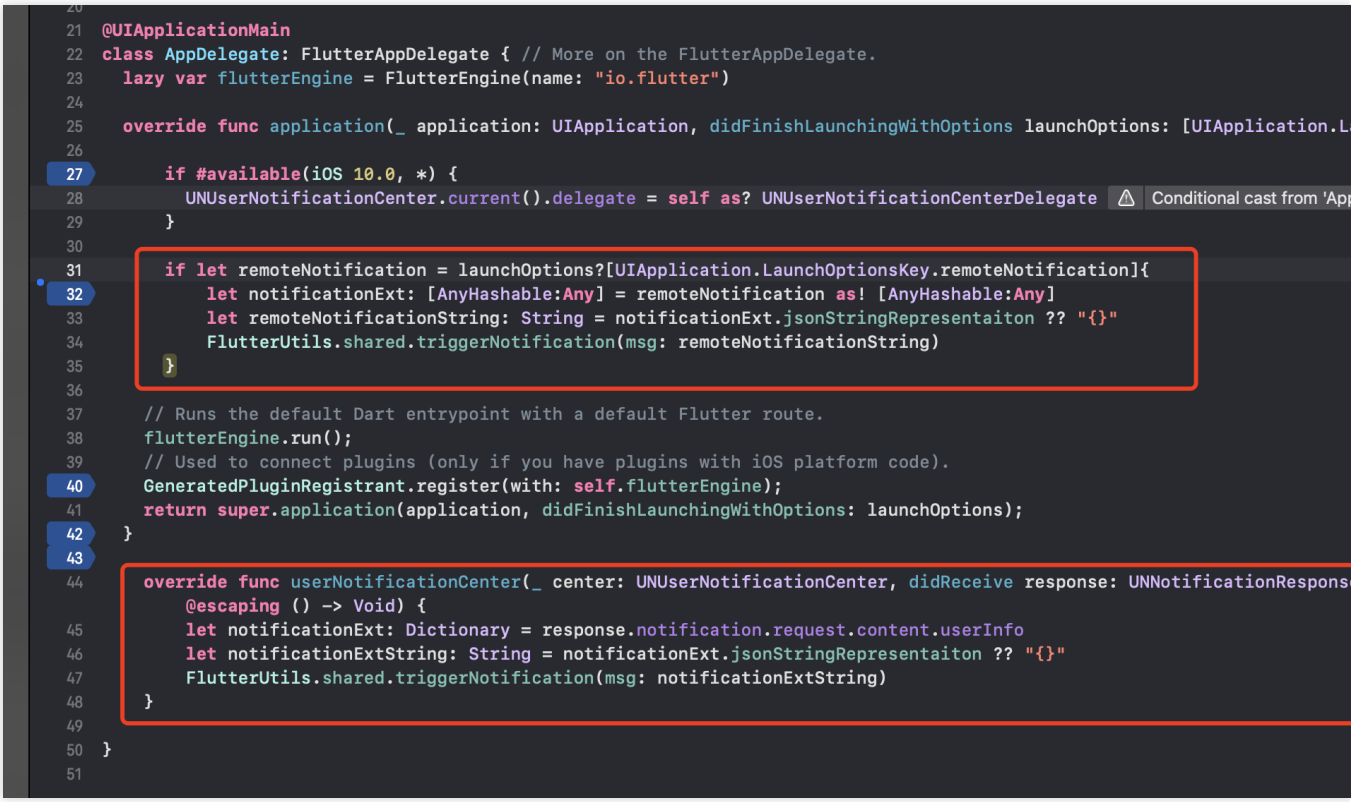
詳細コードについてはDemoソースコードをご参照ください。

オフラインプッシュクリックイベントの監視と転送

オフラインプッシュの初期化/Token送信/クリックイベントに対するセッションジャンプ処理は、Flutter Chatモジュールで行われているので、Native領域はクリック通知イベントのextをパススルーするだけでよい。

これは、クリック通知イベントがNativeでブロックされて消費されているため、Flutter層が直接取得することができず、Native経由で転送しなければならないからです。

`AppDelegate.swift` ファイルに、次のコードを追加します。具体的なコードについてはDemoソースコードをご参照ください。



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

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

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

      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: UNNotificationRespons
        @escaping () -> Void) {
        let notificationExt: Dictionary = response.notification.request.content.userInfo
        let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
        FlutterUtils.shared.triggerNotification(msg: notificationExtString)
    }

}
```

この時点で、iOS Native層の作成が完了しました。

Android Native開発

このドキュメントはKotlin言語を例にします。

説明：

次のコード構造は参照用です。必要に応じて柔軟に設定できます。

Flutter Moduleの導入

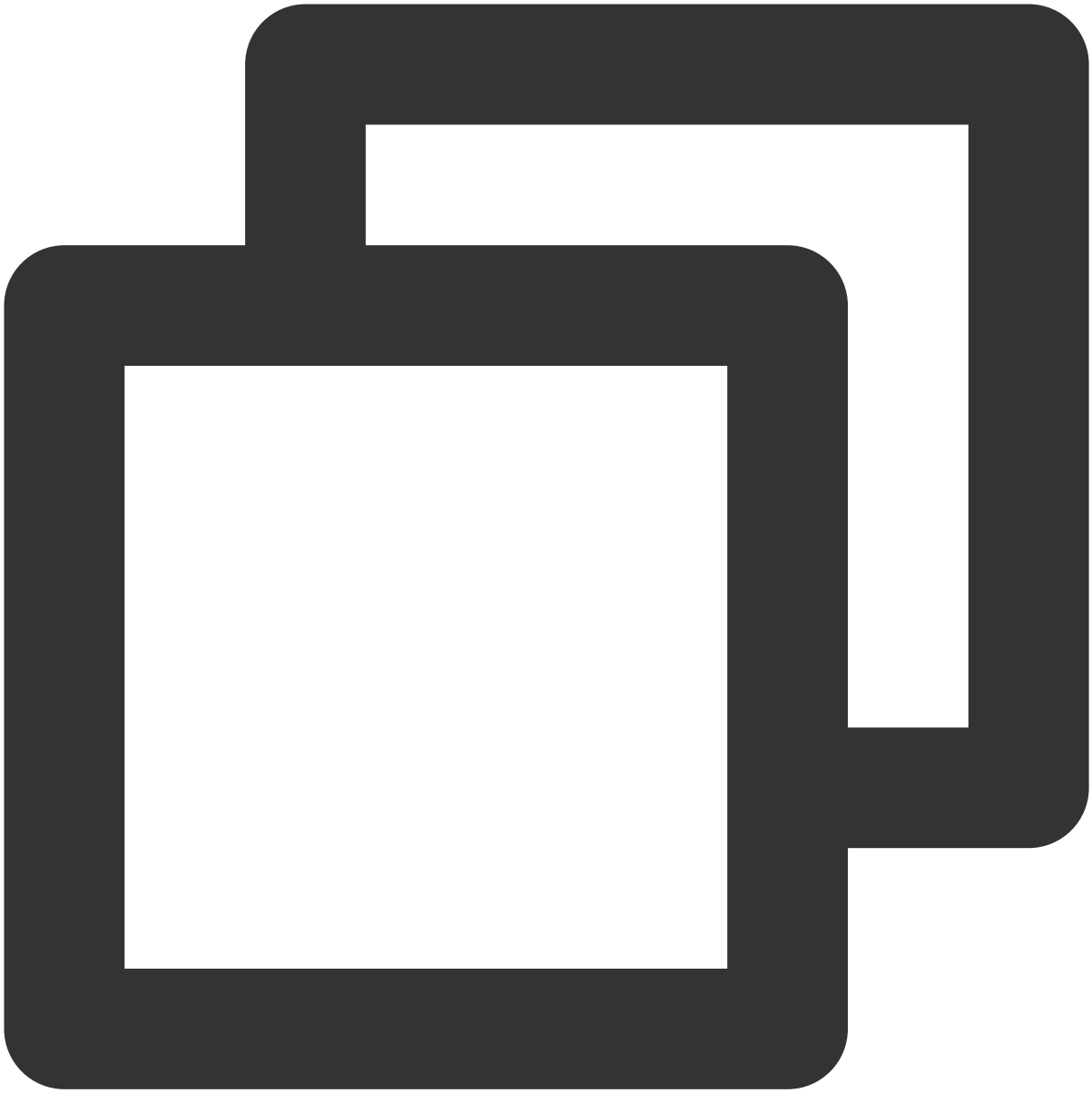
ネイティブアプリケーションにFlutter moduleを導入するには、[この部分]（#android）をご参照ください。方式2を使用することをお勧めします。

AndroidプロジェクトでFlutterエンジンを管理

Flutterエンジンを管理する単一のインスタンスオブジェクトを作成します。

このKotlinの単一インスタンスオブジェクトは、Flutter Method Channelの一元管理に使用され、Flutter Moduleと通信するための一連のメソッドを提供し、プロジェクト内のあらゆる場所で直接呼び出すことができます。

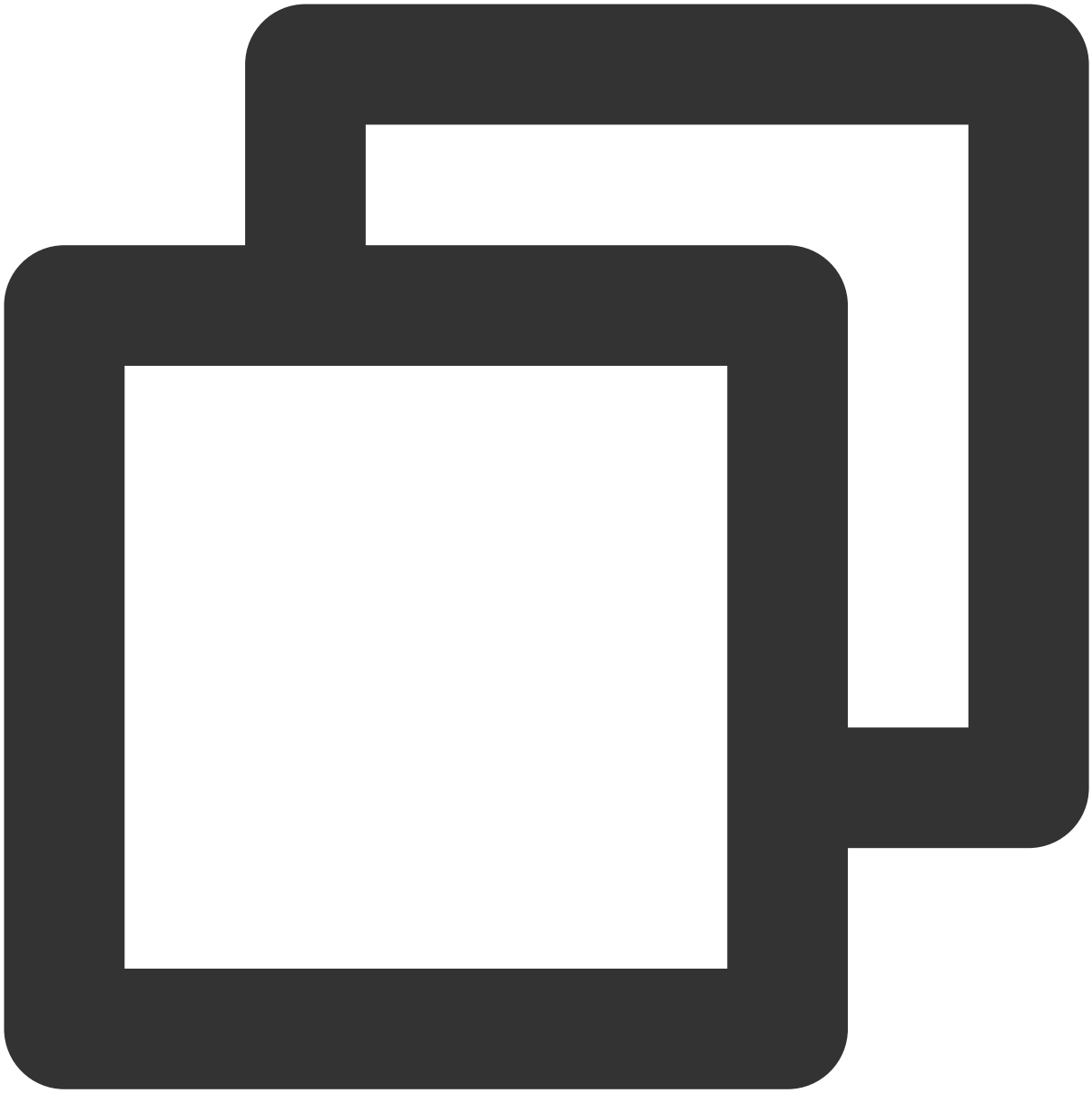
新しい `FlutterUtils.kt` ファイルを作成し、 `FlutterUtils` 静的クラスを定義します。



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

`FlutterEngine` (Flutterエンジン)を作成します。

`FlutterUtils.kt` ファイルに`FlutterEngine'を定義し、初期化時に初期化します。



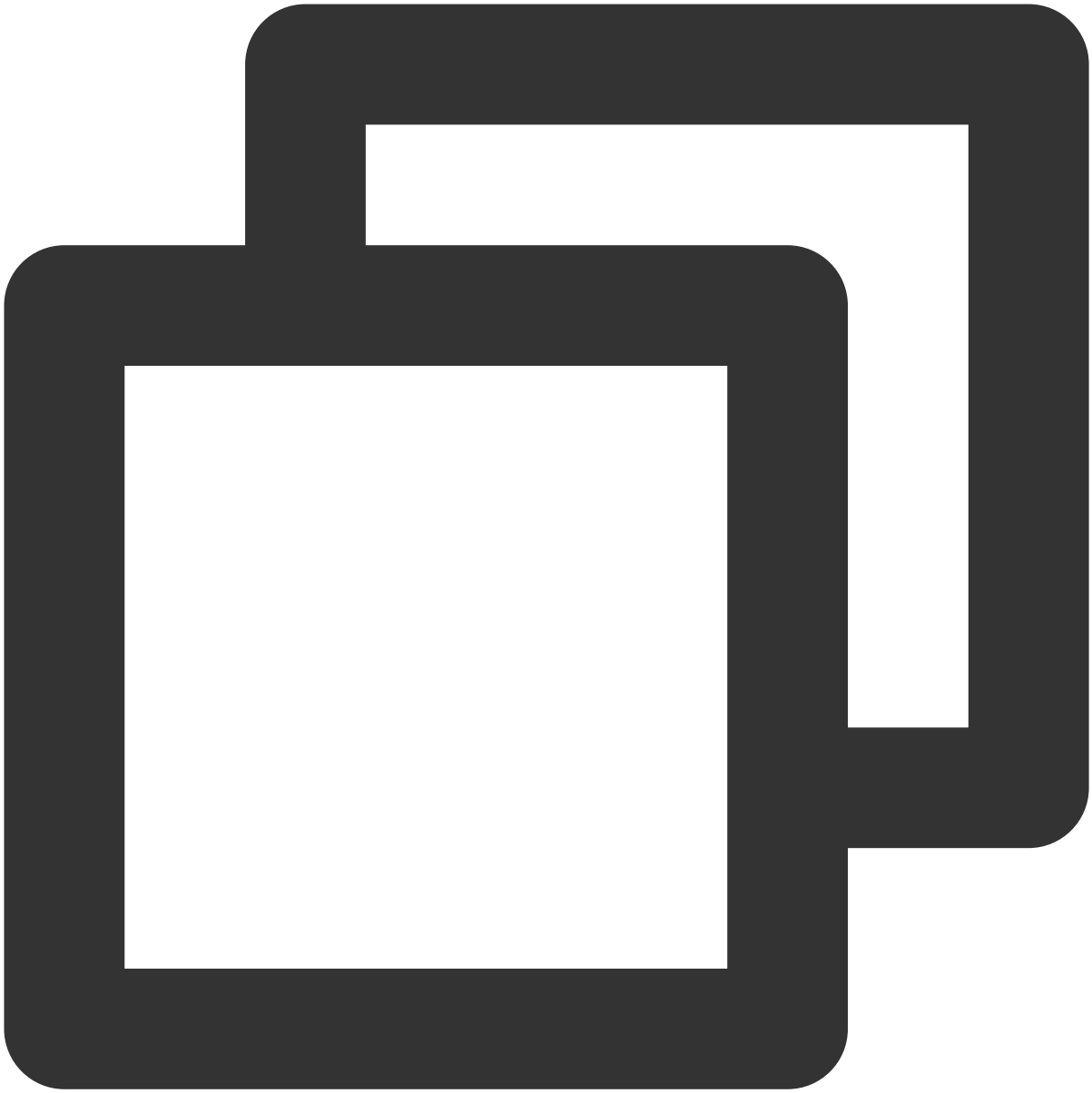
```
lateinit var context : Context
private lateinit var flutterEngine:FlutterEngine

// 初期化
flutterEngine = FlutterEngine(context)
```

Flutterエンジンを管理するために使用される単一のオブジェクトを完成させます。

Demoコードのロジックは新しいルーティングを使用して、ChatとCallのActivityを搭載することです。

ChatのActivityはユーザーが自主的に入ったり退出したりします。CallのActivityは、リスナーまたはアクティブな外部発呼で自動的にナビゲーションにより入ったり退出します。

重点に置くところは次のとおりです。
fun init()：Method Channelを初期化し、イベントリスニングメソッドをバインドします。
fun reportChatInfo()：Flutter層を初期化してTencent Cloud IMにログインできるよう、ユーザーログイン情報とSDKAPPIDをFlutter Moduleにパススルーします。
fun launchChatFunc()：Flutter Moduleが置かれるViewControllerに引き上げるか、ナビゲーションします。
fun triggerNotification(msg: String)：iOS　Native層が受信したオフラインプッシュメッセージクリックイベントとそれに含まれるext情報を、JSON String形式でFlutter層にバインドされた監視処理イベントに送信します。例えば、対応するセッションへのオフラインプッシュクリックジャンプを処理するために使用されます。
この単一インスタンスobjectの詳細コードについては、Demoソースコードをご参照ください。
マスターエントリー `MyApplication` で、上記のオブジェクトを初期化します
`MyApplication.kt` ファイルで単一インスタンスオブジェクトにグロバールcontextを渡し、初期化を実行します。



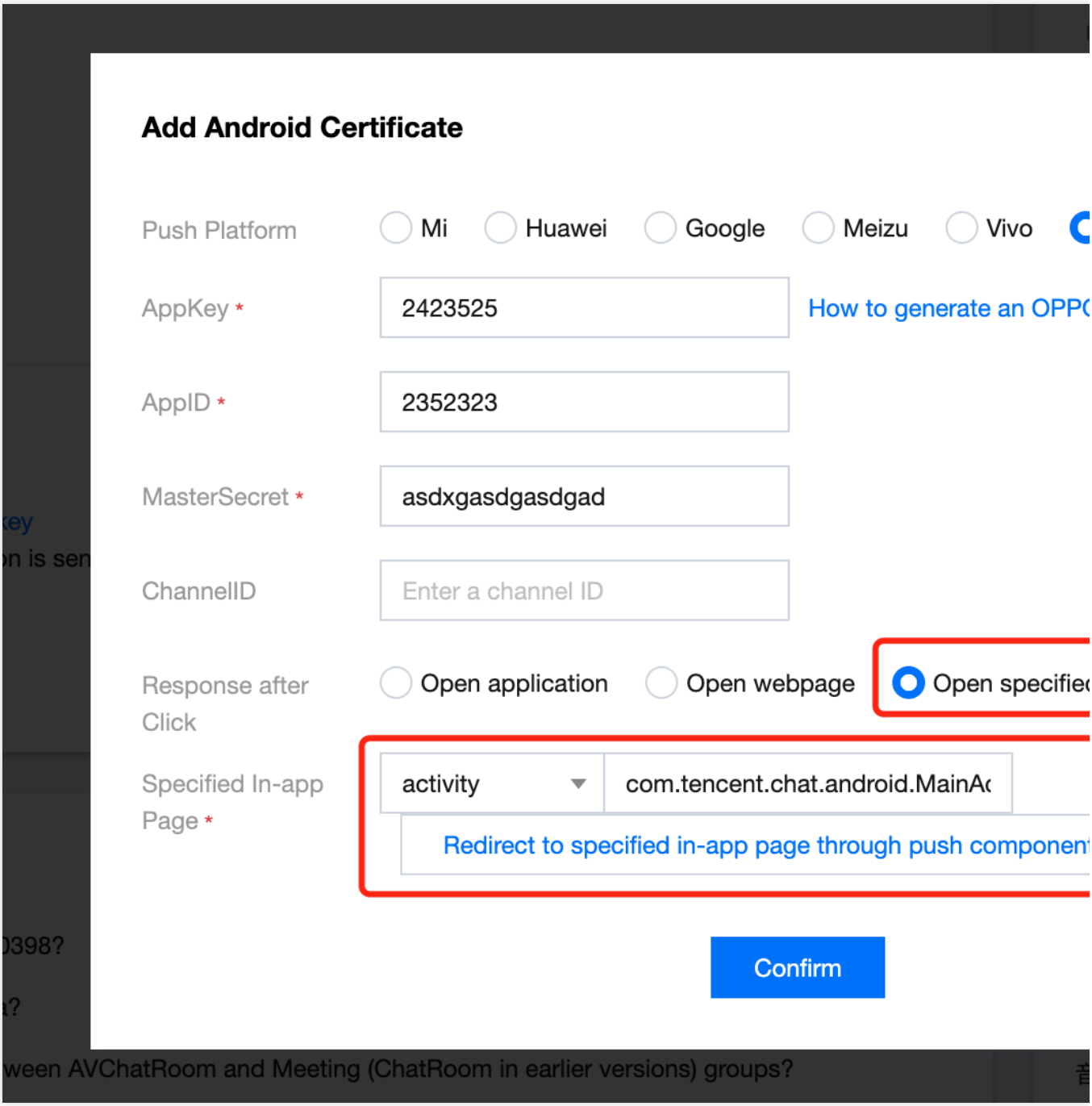
```
class MyApplication : MultiDexApplication() {

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

オフラインプッシュクリックイベントの監視と転送

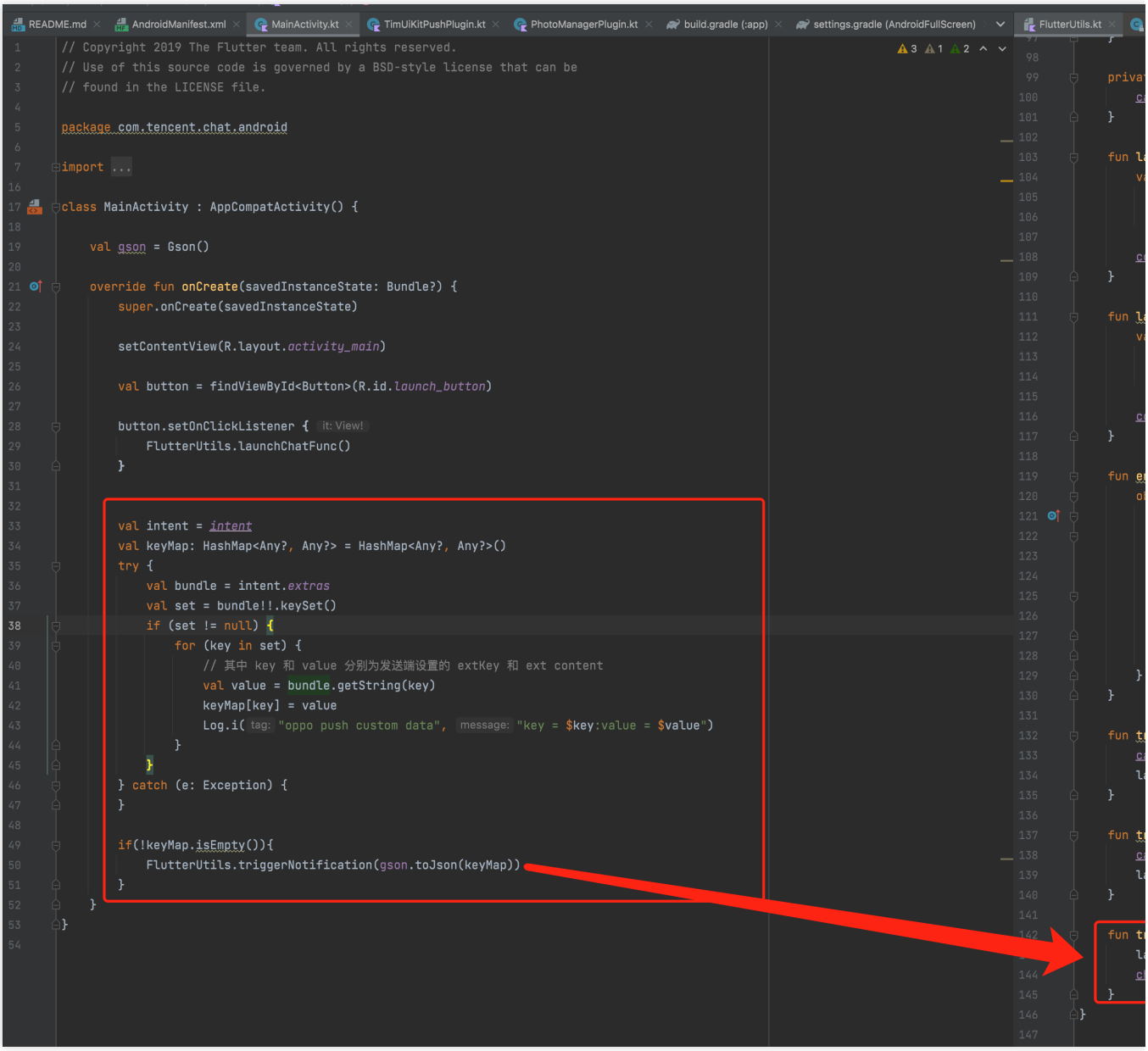
オフラインプッシュの初期化/Token送信/クリックイベントに対するセッションジャンプ処理は、Flutter Chatモジュールで行われているので、Native領域はクリック通知イベントのextをパススルーするだけでよいです。
これは、クリック通知イベントがNativeでブロックされて消費されているため、Flutter層が直接取得することができず、Native経由で転送しなければならないからです。

ご注意：

オフラインプッシュの導入手順はベンダーによって異なるため、このドキュメントではOPPOを例にします。すべてのベンダーの導入方法についてはこのドキュメントをご参照ください。
Tencent Cloud IMコンソールでは、OPPOのプッシュ証明書が追加され、`次のアクションをクリック`して`アプリケーション内の指定されたページを開く`を選択し、`アプリケーション内のページ`で`Activity`方式を用いて、オフラインプッシュ情報を処理するためのページを設定します。アプリケーションの最初のページにすることをお勧めします。例えば、Demoは`com.tencent.chat.android.MainActivity`に設定されます。



上のコンソールで設定されたオフラインプッシュ用のActivityファイルに、次のコードを追加します。
このコードの役割は、ベンダーが該当するActivityを引き上げると、BundleからHashMap形式のextメッセージを取り出し、単一のインスタンスオブジェクト中のメソッドをトリガーし、そのメッセージを手動でFlutterに転送することです。具体的なコードについてはDemoソースコードをご参照ください。



この時点で、Android Native層の作成が完了しました。

追加ソリューション：Native層でTencent Cloud IMの初期化・ログインを行います

ChatとCallのモジュール機能については、高頻度の単純なユースケースを既存のビジネスロジックに深く組み込みたい場合があります。
例えば、ゲームシーンについて、対局内で、直接セッションを立ち上げてほしい。
Flutterを使用して実装された完全な機能Chatモジュールは、APPの重要度の低いサブモジュールであるだけなので、最初から完全なFlutter Moduleを開始することは望ましくありません。

この時点で、Native層からTencent Cloud IM Native SDKの初期化およびログインメソッドを呼び出すことができます。その後、必要な高頻度で簡単なシナリオで、Tencent Cloud IM Native SDKを直接使用してIn-App Chat機能を構築することができます。

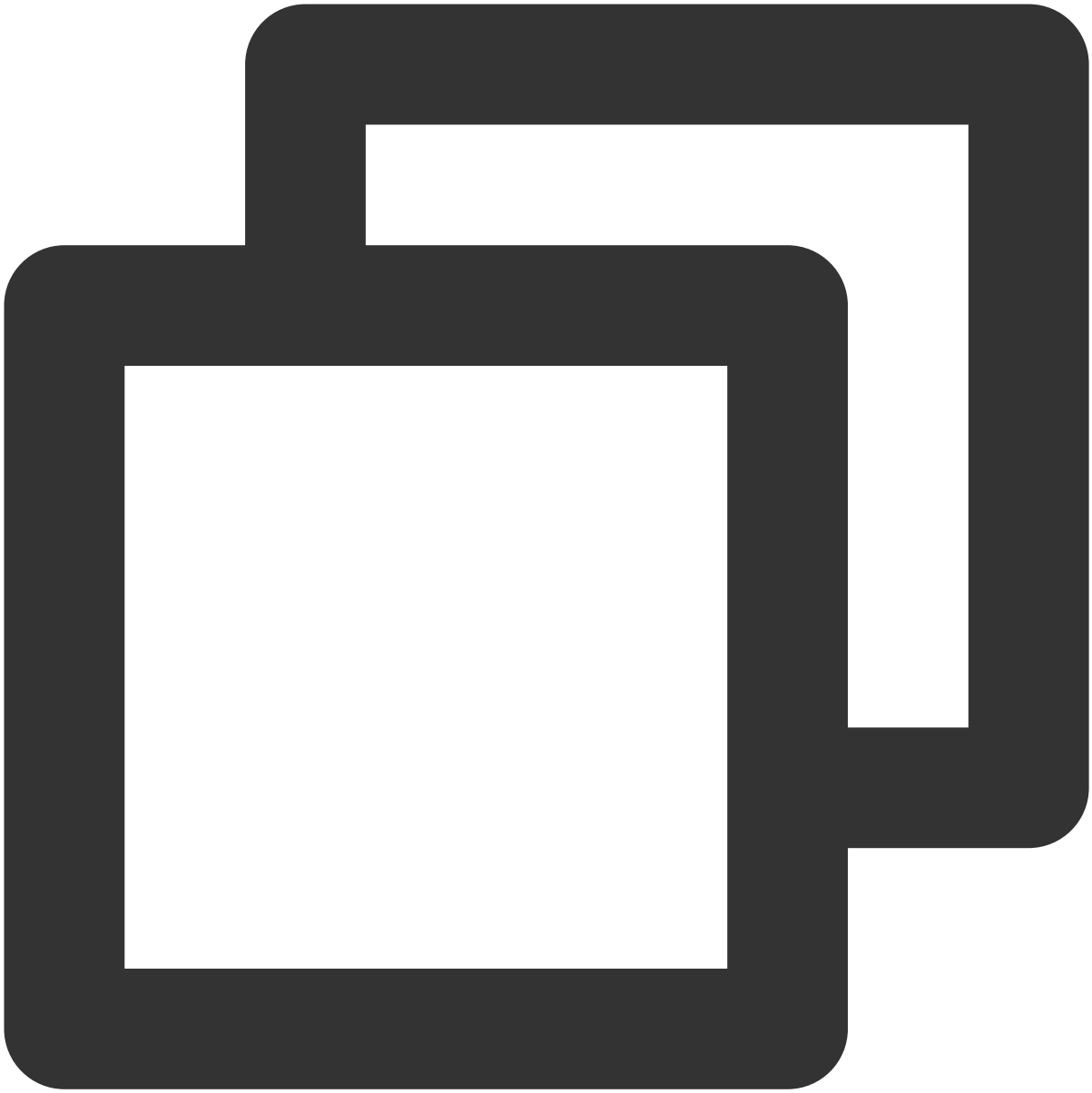
説明：

もちろん、その場合は事前にFlutterでTencent Cloud IMを初期化してログインするという選択肢もありますが、その時点でNative層で再度初期化してログインする必要はなくなります。両方とも一度初期化してログインするだけで、両方のエンドで使用できるようになります。

Flutter SDKにはNative SDKが組み込まれているため、Native層で再度導入する必要はなく、直接使用できます。

Native初期化とログイン

iOS Swiftコードを例に、Native層で初期化してログインする方法を示します。



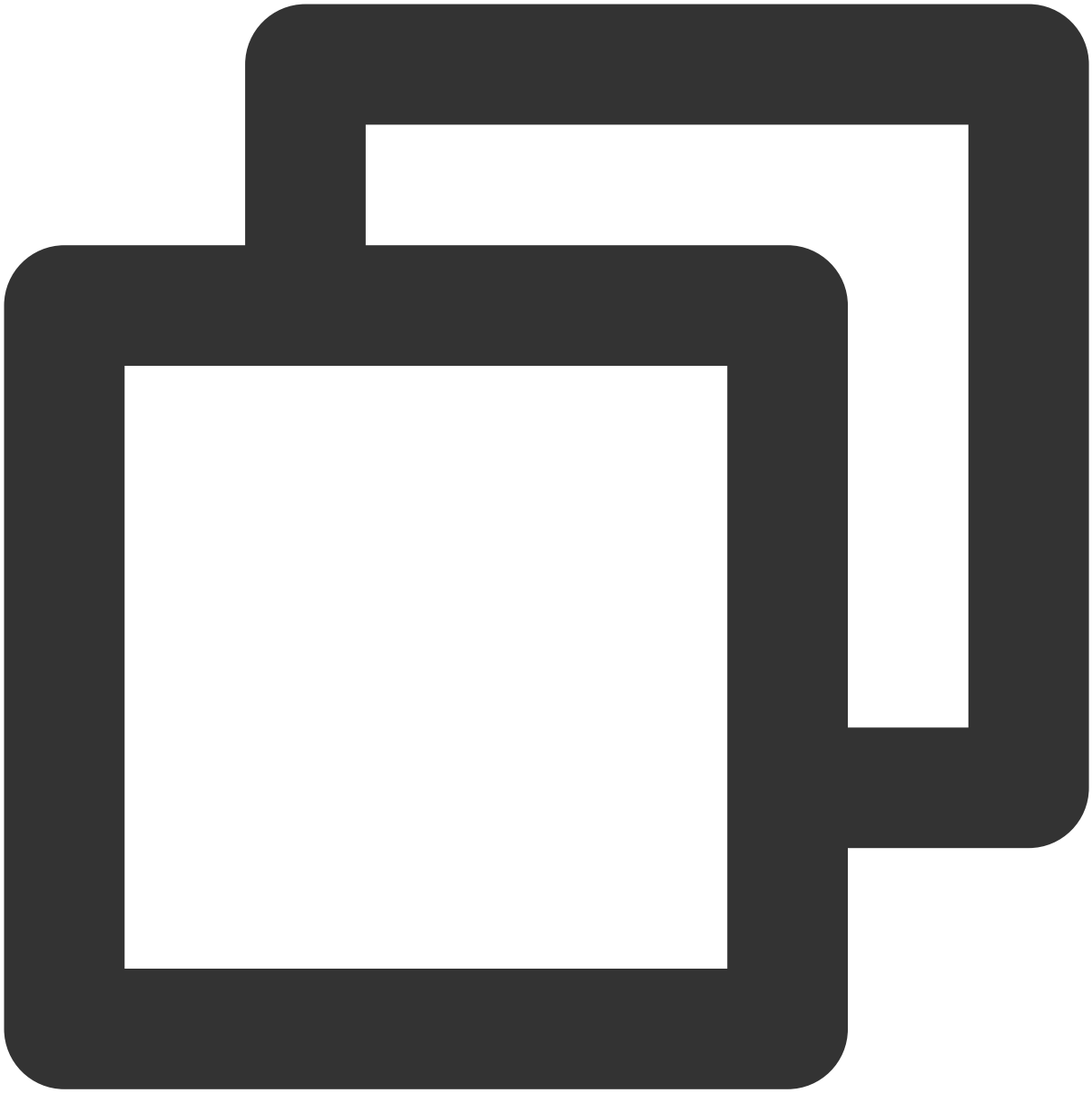
```
import ImSDK_Plus


func initTencentChat(){
        if(isLoginSuccess == true){
            return
        }
        let data = V2TIMManager.sharedInstance().initSDK(あなたのSDKAPPID , config: 
        if (data == true){
            V2TIMManager.sharedInstance().login(
                chatInfo.userID,
```

```
                userSig: chatInfo.userSig,
                succ: {
                    self.isLoginSuccess = true
                    self.reportChatInfo()
                },
                fail: onLoginFailed()
            )
        }
}
```

Native層で、Native SDKを直接使用してビジネス機能モジュールを構築できます。詳細についてはiOSクイックスタートまたはAndroidクイックスタートをご参照ください。

Flutter TUIKitの初期化

Native層で初期化・ログインを実行した場合、Flutter層で再度実行する必要はありませんが、TUIKitの `_coreInstance.setDataFromNative()` を呼び出して、現在のユーザ情報を渡す必要があります。



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

コードの詳細については、Demoソースをご参照ください。

これにより、Tencent Cloud　IM Flutter-Nativeハイブリッド開発方式のすべてを紹介しました。
このドキュメントに記載されたソリューションにより、既存のネイティブ開発Android/iOSアプリで、Flutter SDKを使用して、同じFlutterコードセットを使用して、ChatとCallモジュール機能を迅速に移植することができます。
ご不明の点があれば、いつでもお気軽にお問い合わせをしてください。
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.