即时通信 IM

聊天互动（含 UI）

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

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

【商标声明】

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

【服务声明】

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

文档目录

聊天互动（含 UI）
TUIKit 界面库
Android & iOS
Web
H5
Flutter
快速开始
Android
iOS
Flutter
集成 TUIKit
Android
iOS
Web & H5（react)
Web & H5 （Vue）
Unity
ReactNative
Flutter
uniapp
仅集成聊天
Android
iOS
Web（Vue）
React
uni-app
Flutter
构建基础界面
聊天界面
Android
iOS
会话列表
Android
iOS
联系人界面
Android

版权所有：腾讯云计算（北京）有限责任公司

- iOS
- 添加联系人
  - Android
  - iOS
- 创建群组
  - Android
  - iOS
- 音视频通话功能
  - Android
  - iOS

修改界面主题
- Android
- iOS
- Flutter

设置界面风格
- Android
- iOS
- Web(Vue)
- H5(Vue)
- Flutter

实现本地搜索
- Android
- iOS
- Web & H5 & Uniapp（Vue）
- Flutter

接入离线推送
- Android
- iOS
- Flutter

用户在线状态
- Android
- iOS
- Web & H5 & Uniapp (Vue)
- Flutter

对方正在输入
- Android
- iOS
- Web & H5 & Uniapp (Vue)

- Flutter
- 消息已读回执
  - Android
  - iOS
  - Web & H5 & Uniapp（Vue）
  - Flutter
- 消息表情回应
  - Android
  - iOS
  - Web & H5 & Uniapp（Vue）
  - Flutter
- 消息引用
  - Android & iOS
  - Web & H5 & Uniapp（Vue）
  - Flutter
- 文本消息翻译
  - Android & iOS
  - Web & H5 & Uniapp（Vue）
  - Flutter
- 语音转文字
  - Android & iOS
  - Web & H5 & Uniapp（Vue）
- 国际化界面语言
  - Android
  - iOS
  - Web & H5 (Vue)
  - React
  - Flutter
- 添加自定义消息
  - Android
  - iOS
  - Web & H5 & Uniapp (Vue)
  - Flutter
  - ReactNative
- 添加自定义表情
  - Android
  - iOS
  - Web & H5 & Uniapp

版权所有：腾讯云计算（北京）有限责任公司

Flutter

聊天互动（含 UI）
TUIKit 界面库
Android & iOS

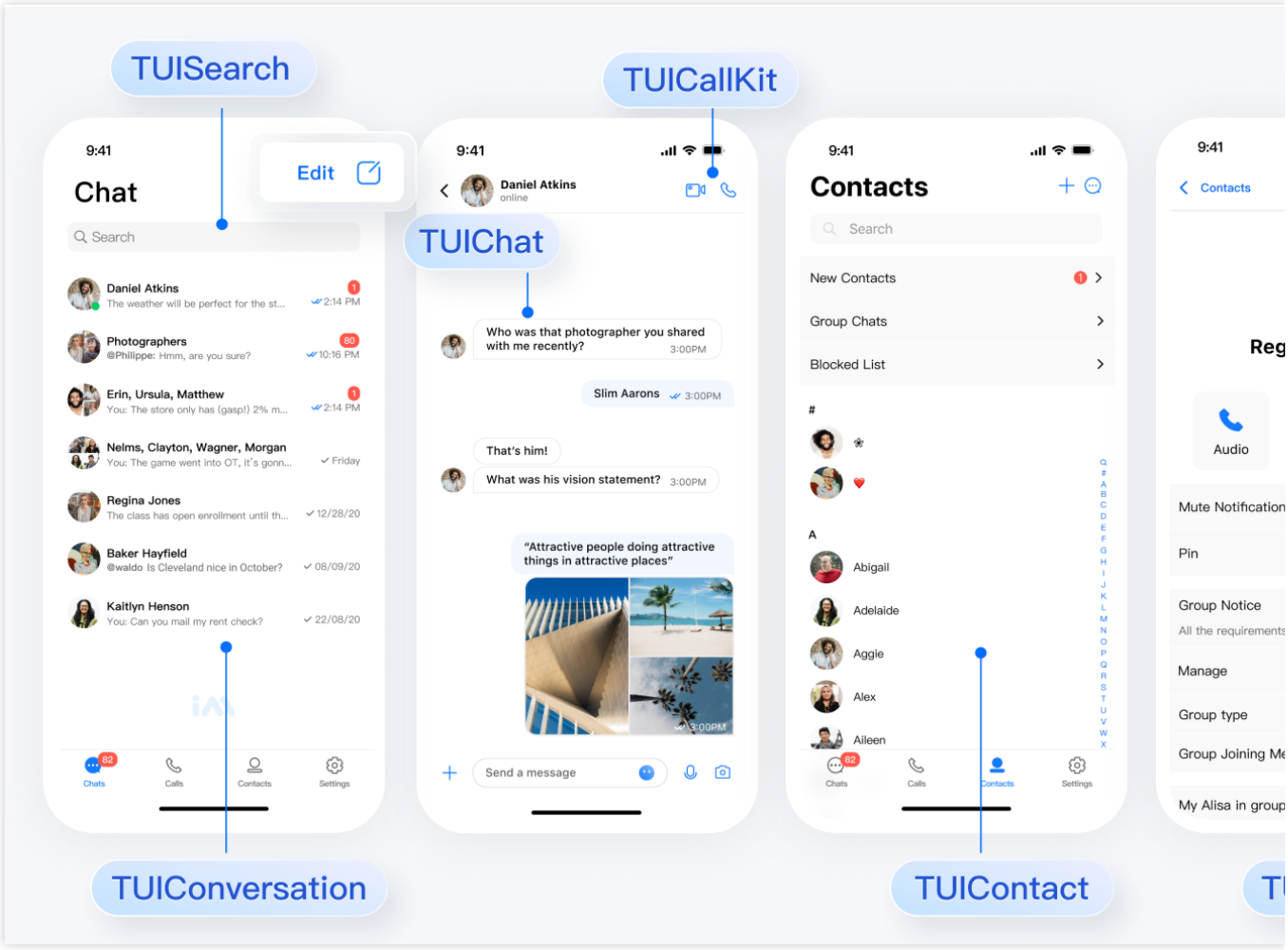
最近更新时间：2024-03-26 15:27:48

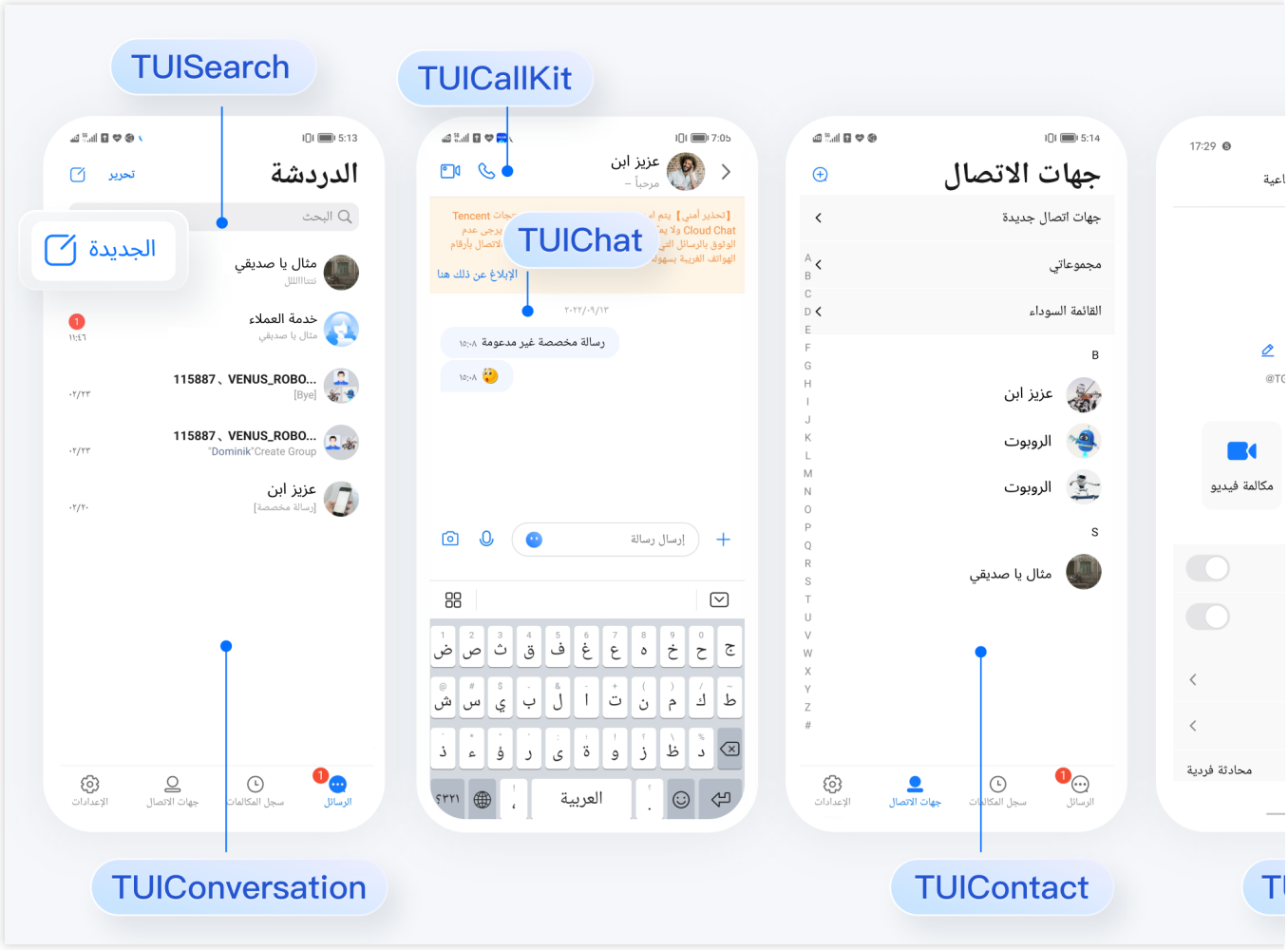
TUIKit 介绍

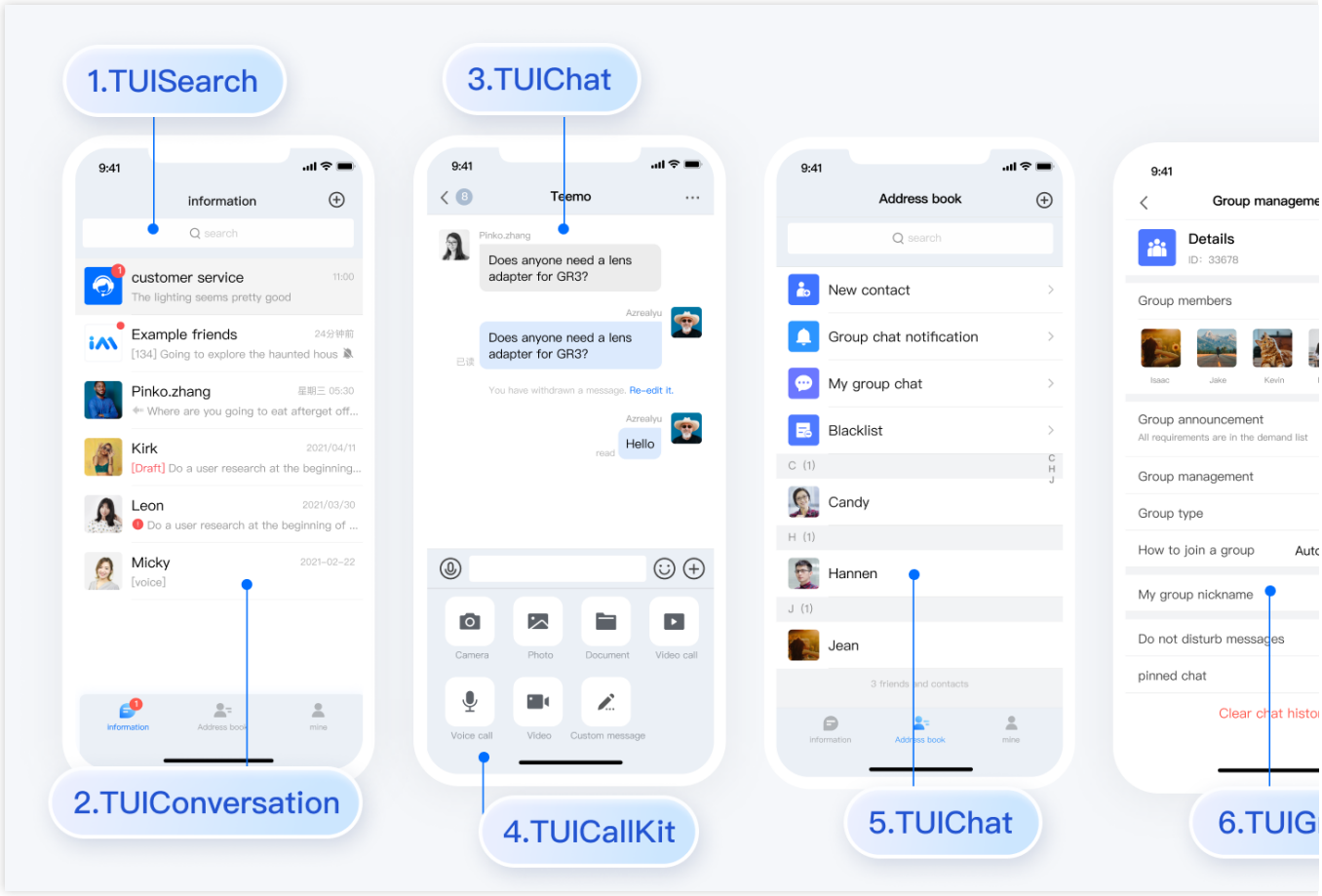
TUIKit 是基于腾讯云 IM SDK 的一款 UI 组件库，它提供了一些通用的 UI 组件，包含会话、聊天、搜索、关系链、群组、音视频通话等功能。
基于 UI 组件您可以像搭积木一样快速搭建起自己的业务逻辑。
TUIKit 中的组件在实现 UI 功能的同时，会调用 IM SDK 相应的接口实现 IM 相关逻辑和数据的处理，因而开发者在使用 TUIKit 时只需关注自身业务或个性化扩展即可。
TUIKit 从 6.9.3557 版本开始新增了全新的简约版 UI 组件，之前版本 UI 组件依旧保留，我们称之为经典版 UI 组件，您可以根据需求自由选择经典版或简约版 UI 组件。
TUIKit 从 7.5.4852 版本开始新增了 RTL 语言（文字方向从右到左的语言，比如阿拉伯语、希伯来语等）支持，当应用内语言为 RTL 语言时，TUIKit 会自动切换到 RTL 样式；同时内置语言新增了阿拉伯语。

TUIKit 主要功能介绍

TUIKit 主要分为 TUISearch、TUIConversation、TUIChat、TUICallKit、TUIContact、TUIGroup 和 TUIOfflinePush 几个 UI 子组件，每个 UI 组件负责展示不同的内容。
界面效果如下图所示：
简约版
RTL语言
经典版


TUISearch
TUICallKit
TUIChat
TUIConversation
TUIContact
9:41
Chat
Edit
Search
Daniel Atkins
The weather will be perfect for the st...
2:14 PM
Photographers
@Philippe: Hmm, are you sure?
10:16 PM
Erin, Ursula, Matthew
You: The store only has (gasp!) 2% m...
2:14 PM
Nelms, Clayton, Wagner, Morgan
You: The game went into OT, it's gonn...
Friday
Regina Jones
The class has open enrollment until th...
12/28/20
Baker Hayfield
@waldo Is Cleveland nice in October?
08/09/20
Kaitlyn Henson
You: Can you mail my rent check?
22/08/20
Chats
Calls
Contacts
Settings
Daniel Atkins
online
Who was that photographer you shared with me recently?
3:00PM
Slim Aarons
3:00PM
That's him!
What was his vision statement?
3:00PM
"Attractive people doing attractive things in attractive places"
Send a message
Contacts
Search
New Contacts
Group Chats
Blocked List
Abigail
Adelaide
Aggie
Alex
Aileen
Audio
Mute Notification
Pin
Group Notice
Manage
Group type
Group Joining Me
My Alisa in group


TUISearch
TUICallKit
TUIChat
TUIConversation
TUIContact
الدردشة
تحرير
البحث
الجديدة
مثال يا صديقي
خدمة العملاء
115887، VENUS_ROBO...
115887، VENUS_ROBO...
عزيز ابن
الإعدادات
جهات الاتصال
سجل المكالمات
الرسائل
عزيز ابن
مرحباً -
إرسال رسالة
العربية
جهات الاتصال
جهات اتصال جديدة
مجموعاتي
القائمة السوداء
عزيز ابن
الروبوت
الروبوت
مثال يا صديقي
مكالمة فيديو
محادثة فردية



TUIChat 重点功能介绍

TUIChat 主要负责消息界面的展示。您还可以利用它直接发送不同类型的消息、对消息长按点赞/回复/引用、查询消息已读回执详情等。

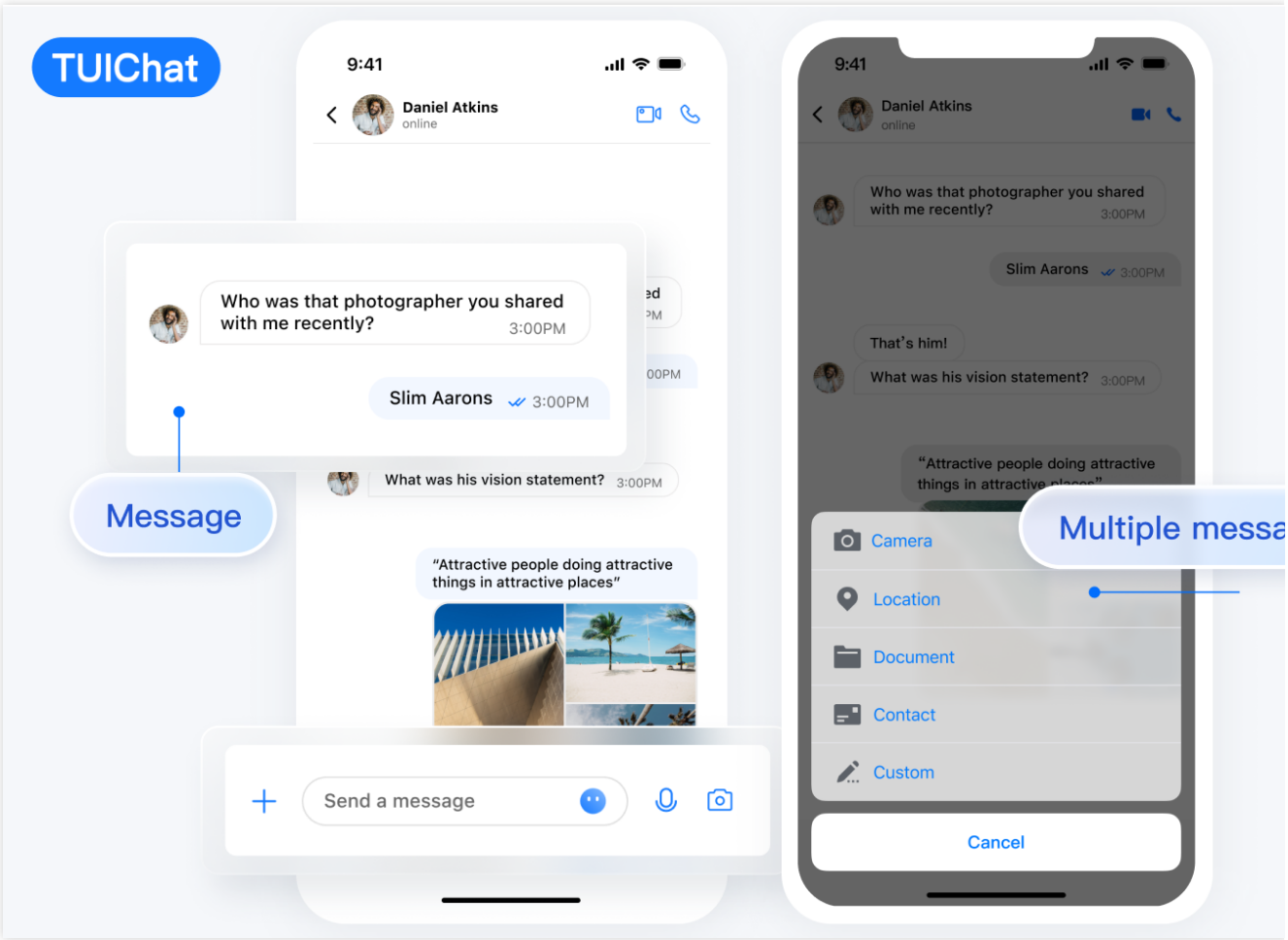
界面效果如下图所示：

简约版

RTL语言

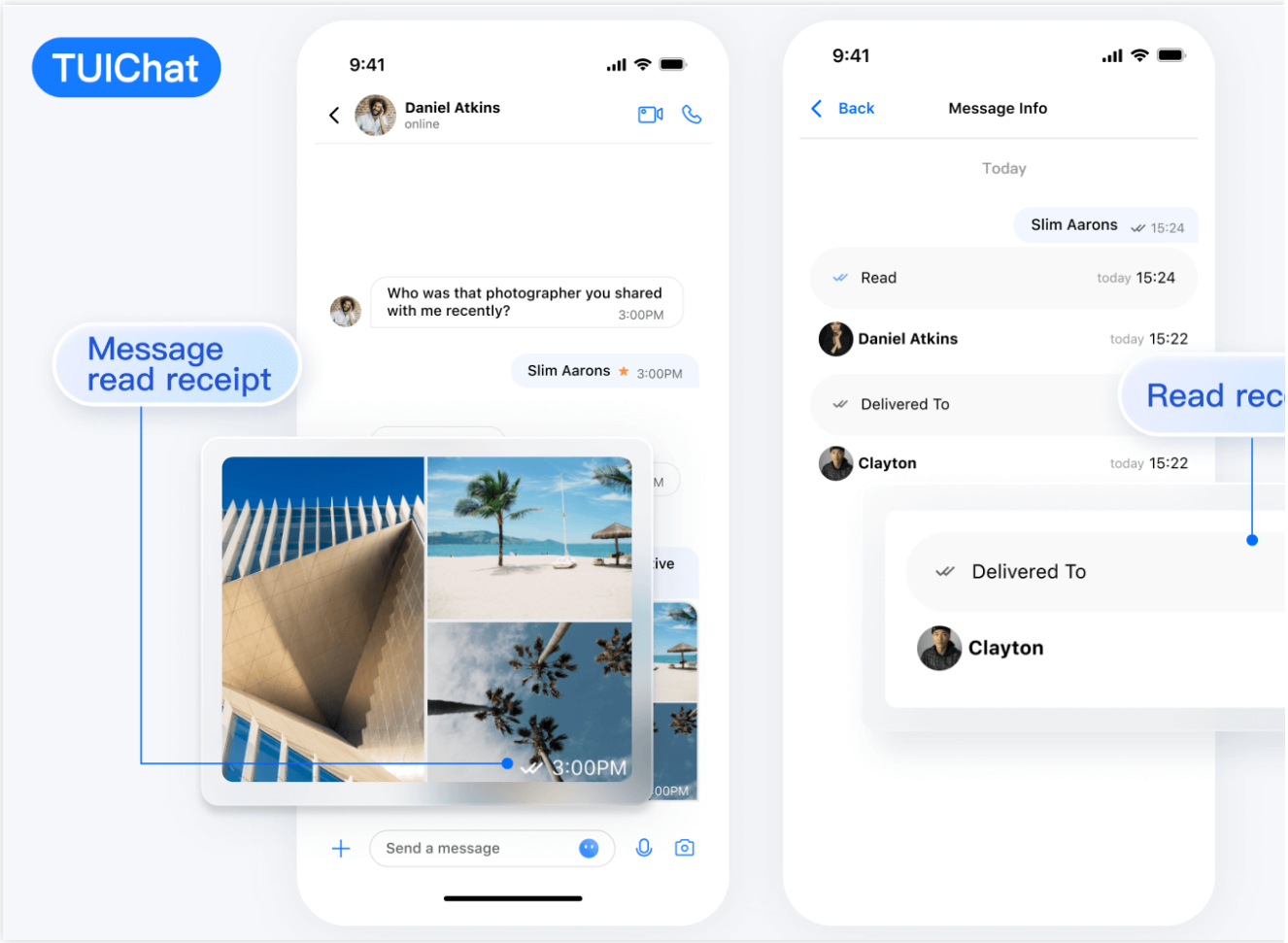
经典版

消息界面 | 发送多种类型消息


TUIChat
9:41
Daniel Atkins
online
Who was that photographer you shared with me recently?
3:00PM
Slim Aarons
3:00PM
What was his vision statement?
3:00PM
"Attractive people doing attractive things in attractive places"
Message
Send a message
Camera
Location
Document
Contact
Custom
Cancel
That's him!

消息点赞 | 回复

消息已读回执 | 已读回执详情

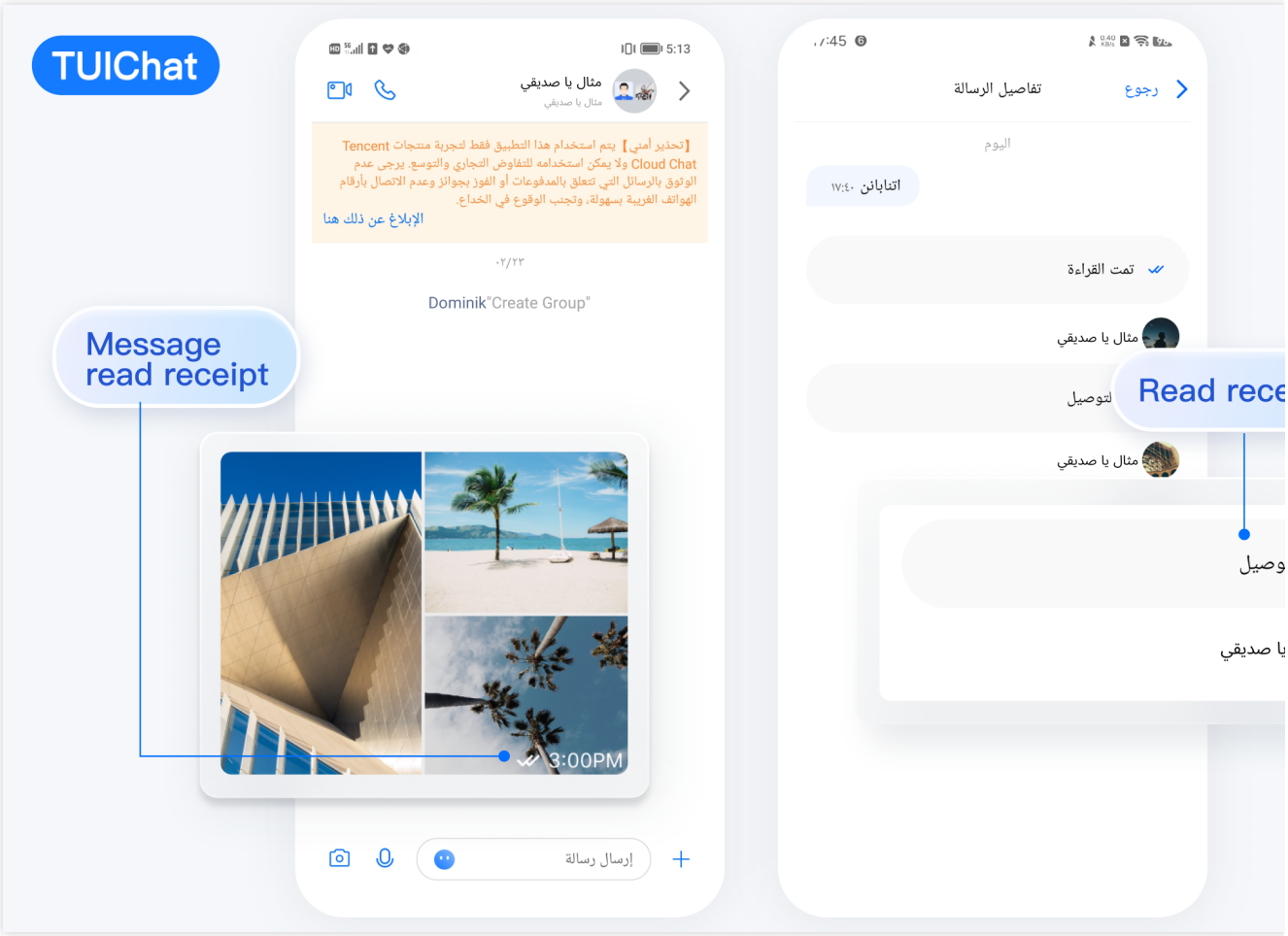


消息界面 | 发送多种类型消息

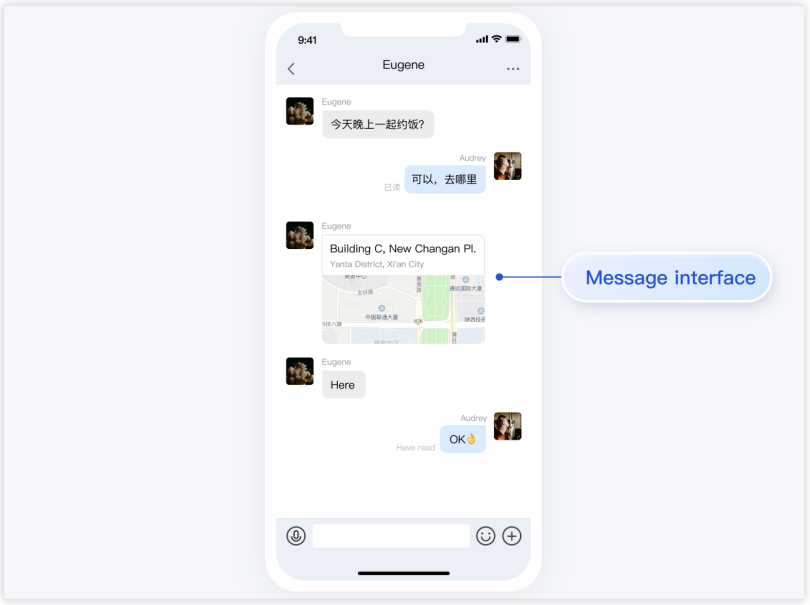
TUIChat
Message
Multiple messa
إرسال رسالة
صورة
التقاط صورة
تصوير فيديو
ملف
رسالة مخصصة
إلغاء
العربية

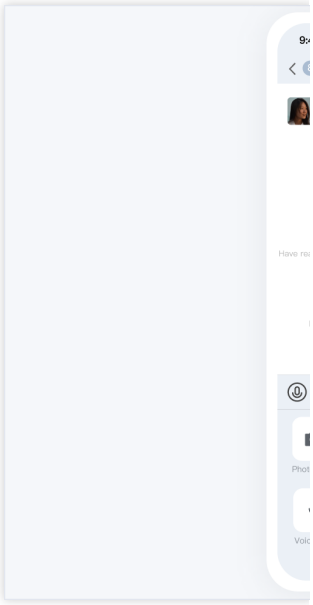
消息点赞 | 回复

消息已读回执 | 已读回执详情



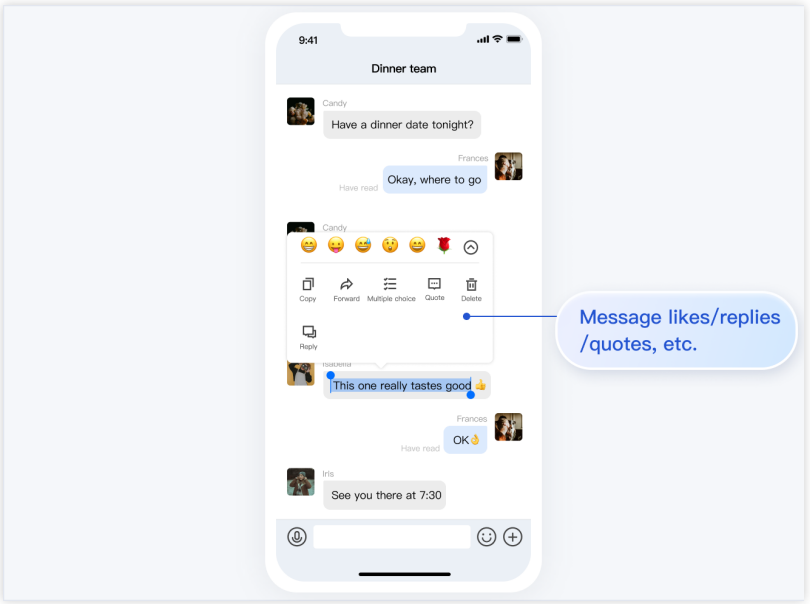
消息界面	发送多种类型消息

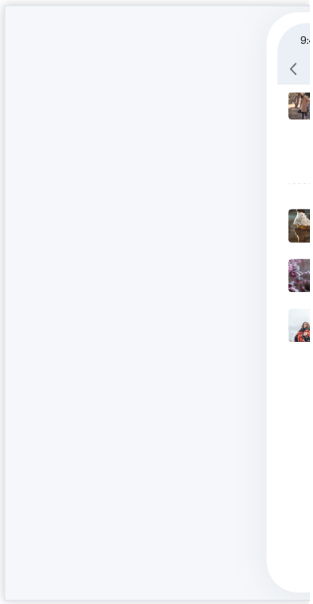




消息点赞/回复/引用

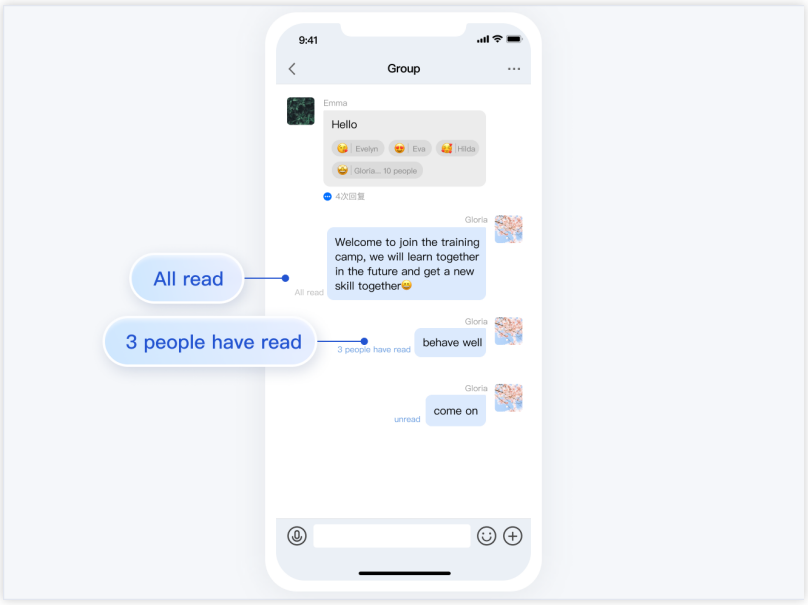
消息回复详情

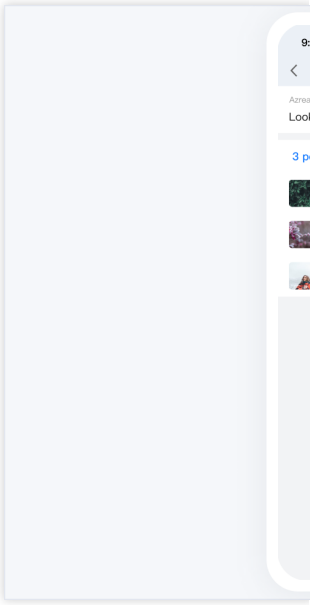




消息已读回执

已读回执详情





TUIContact 重点功能介绍

TUIContact 主要负责联系人的展示、权限设置等。

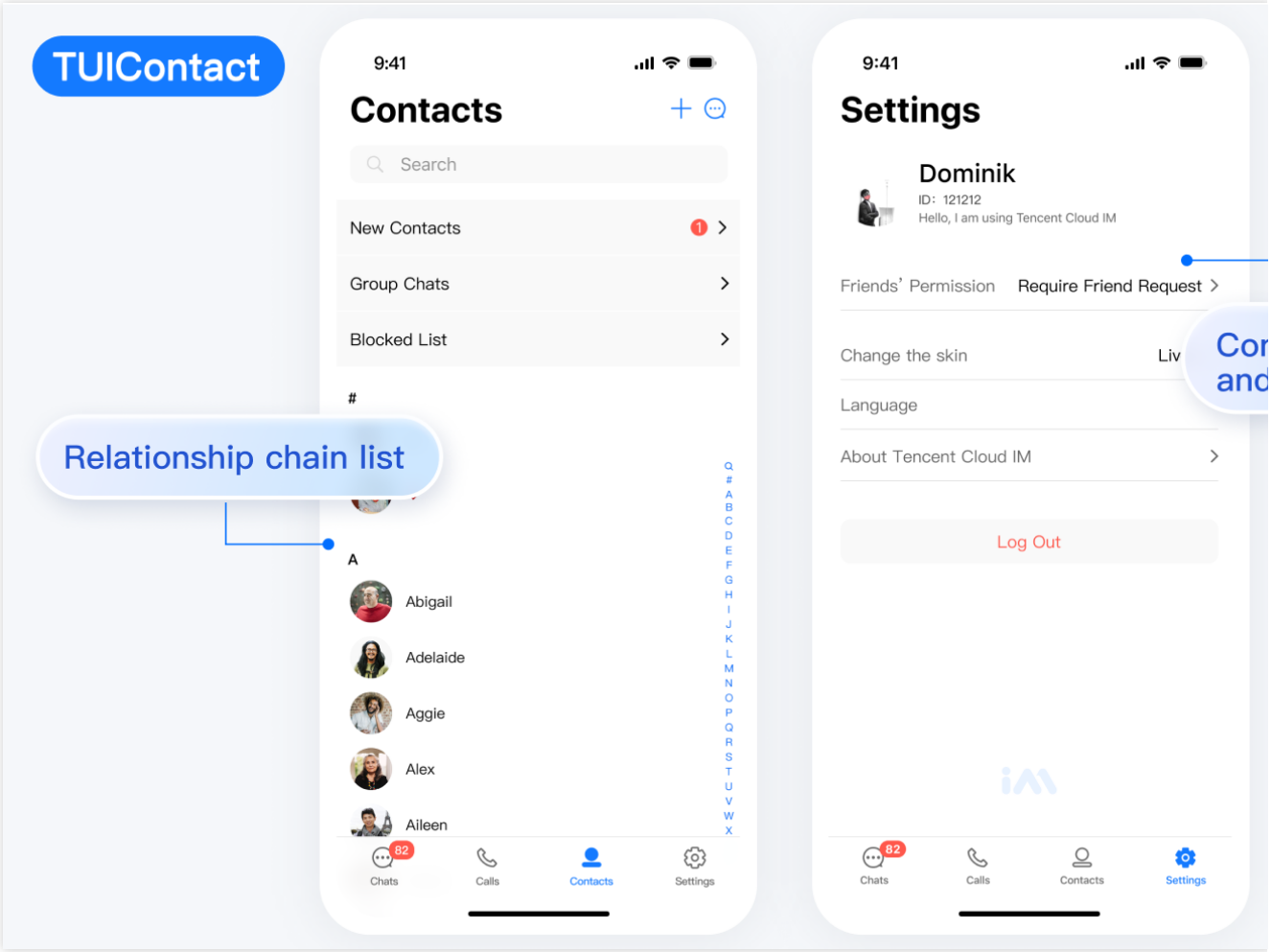
界面效果如下图所示：

简约版

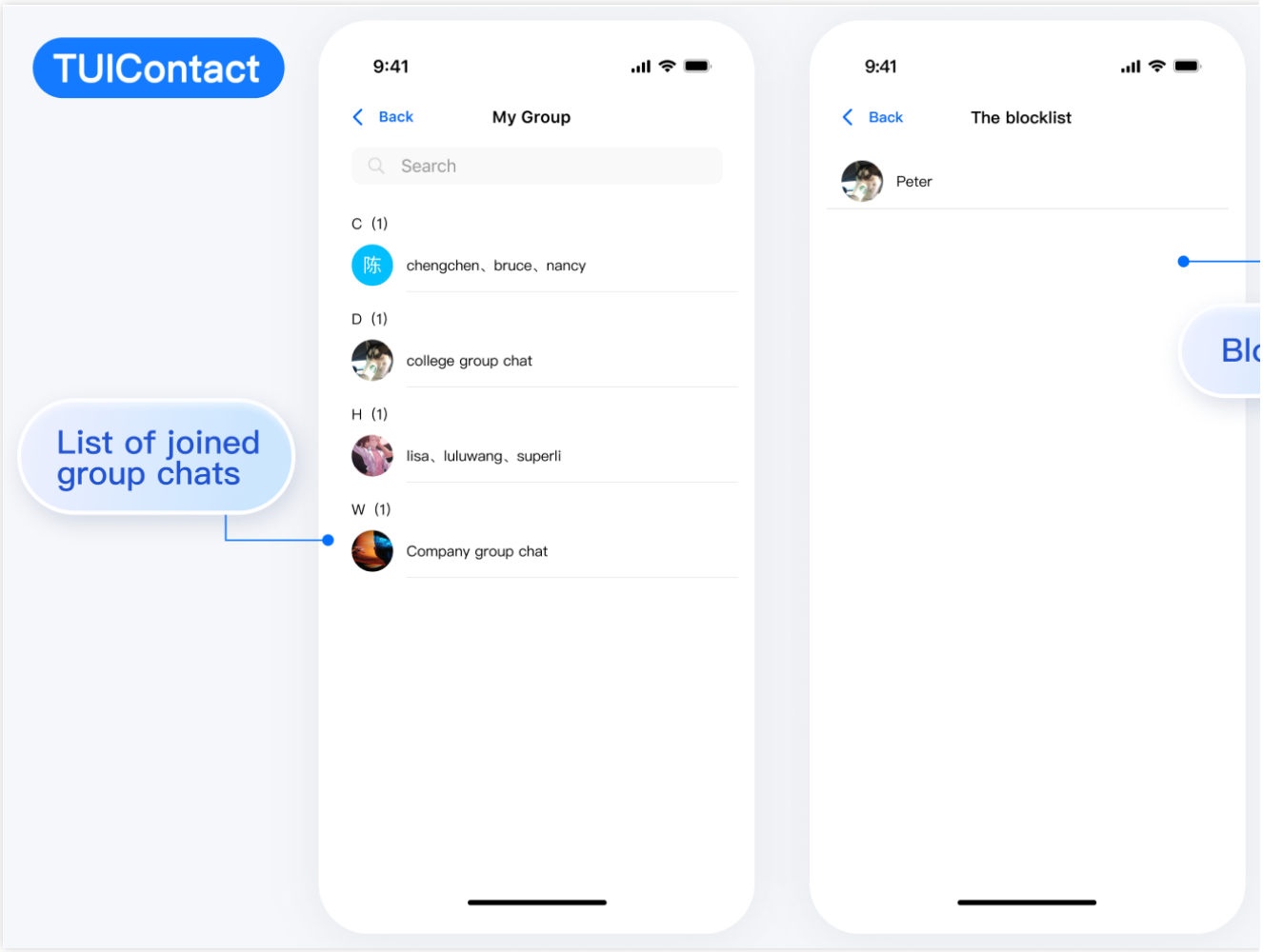
RTL语言

经典版

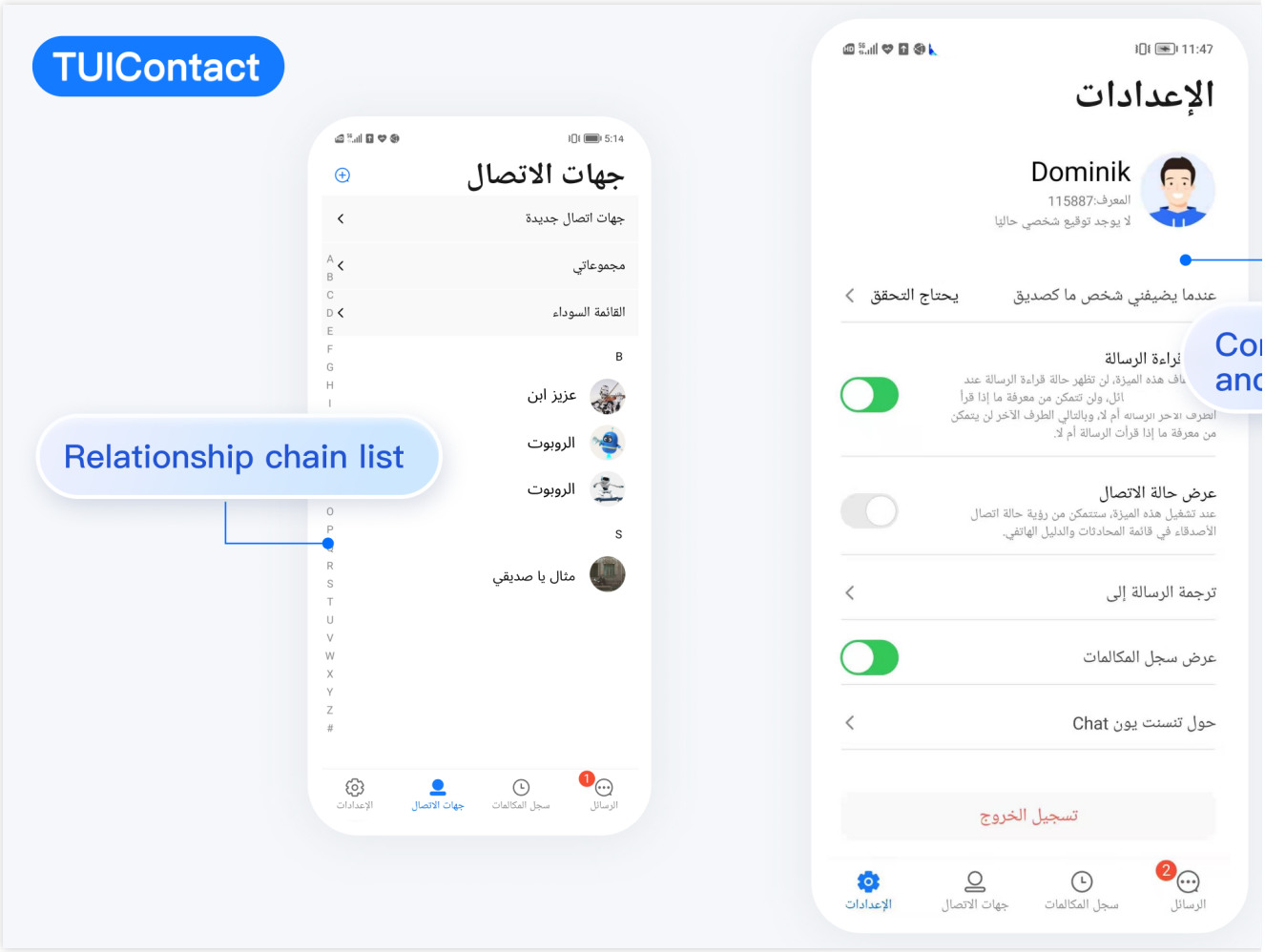
关系链列表 | 联系人资料及管理



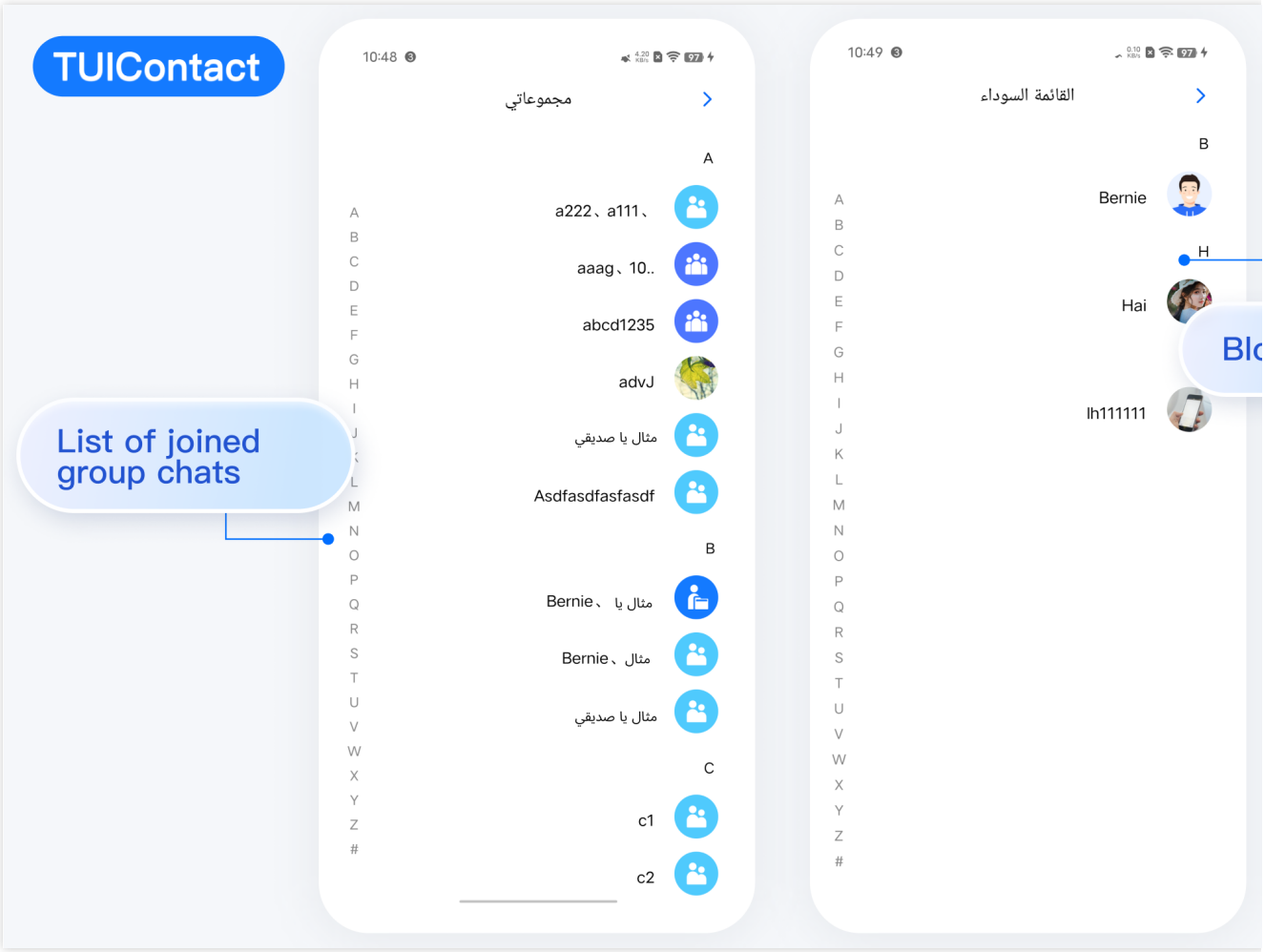
参与的群聊列表 | 黑名单列表



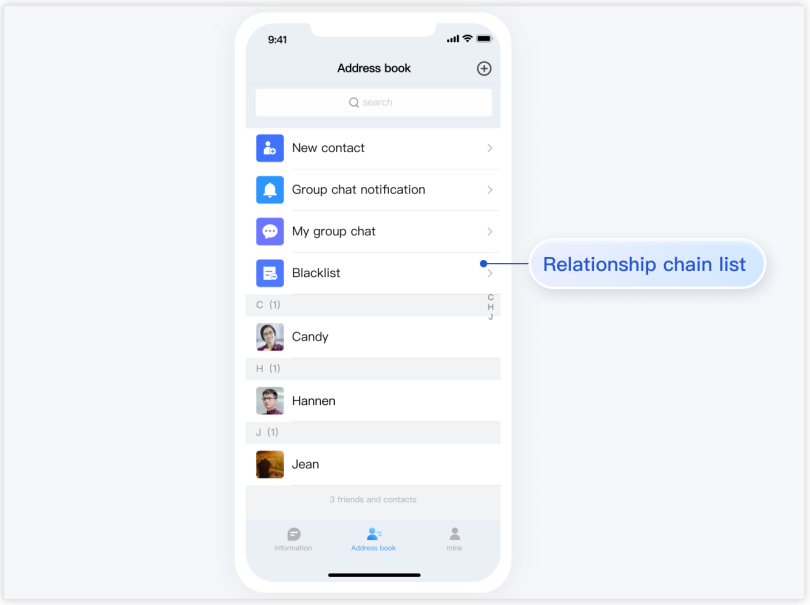
关系链列表 | 联系人资料及管理

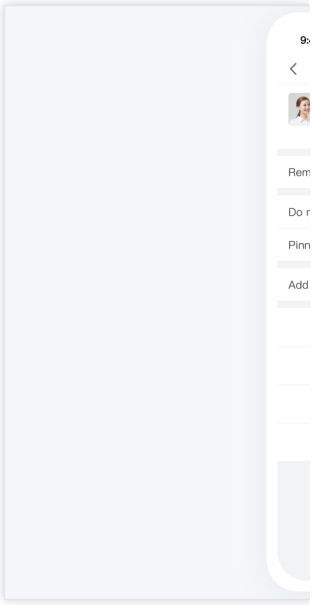


参与的群聊列表 | 黑名单列表

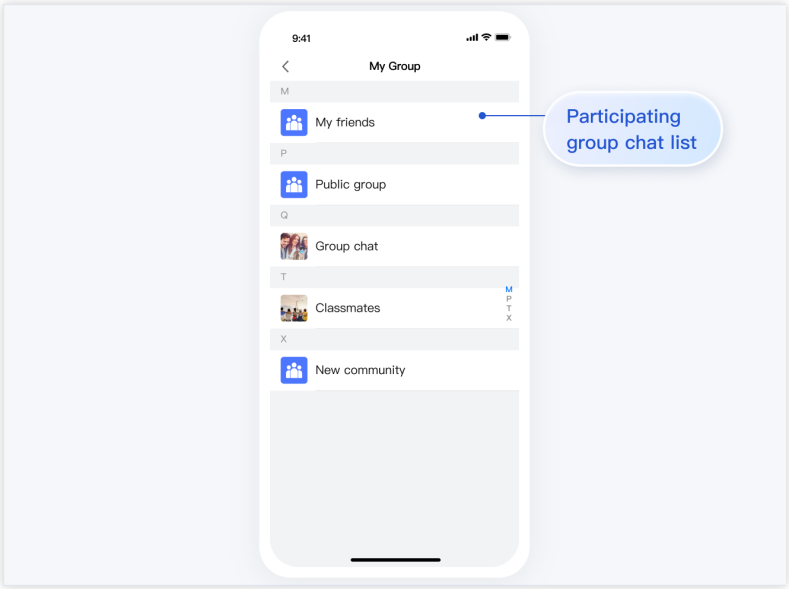


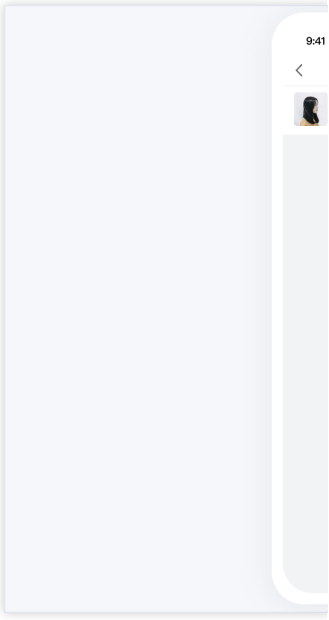
关系链列表	联系人资料及管理





参与的群聊列表	黑名单列表





TUIConversation 重点功能介绍

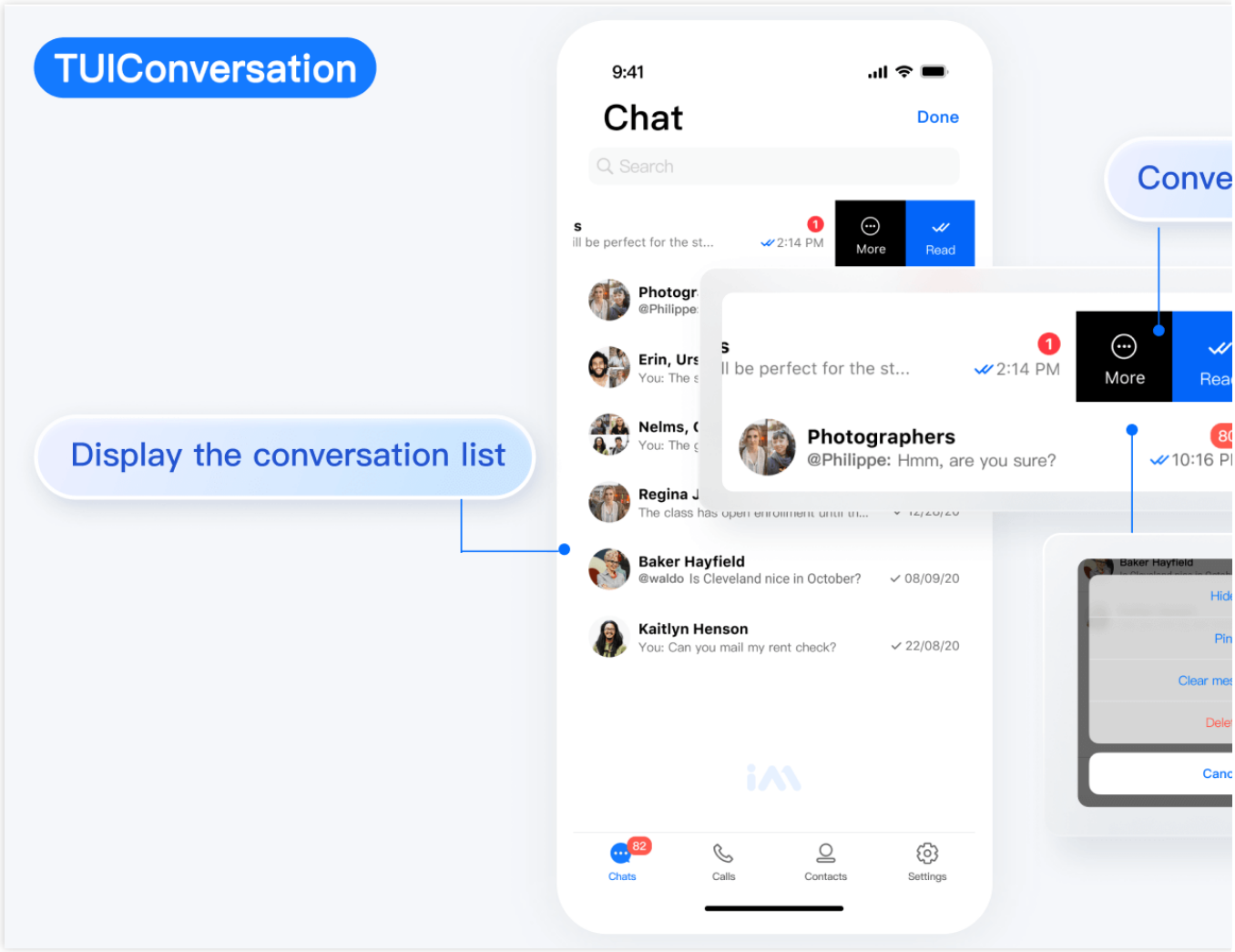
TUIConversation 主要负责会话列表的展示和编辑。

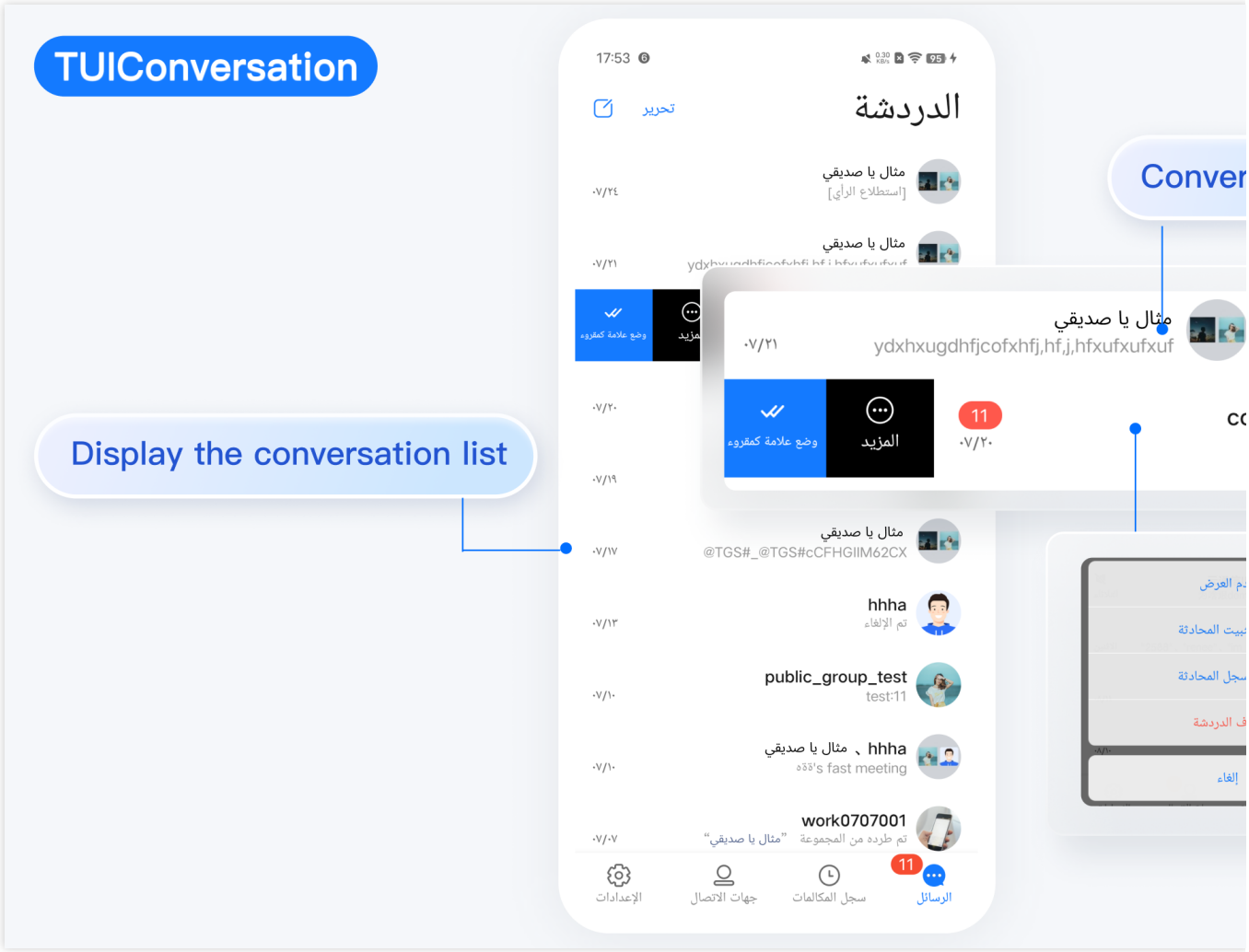
界面效果如下图所示：

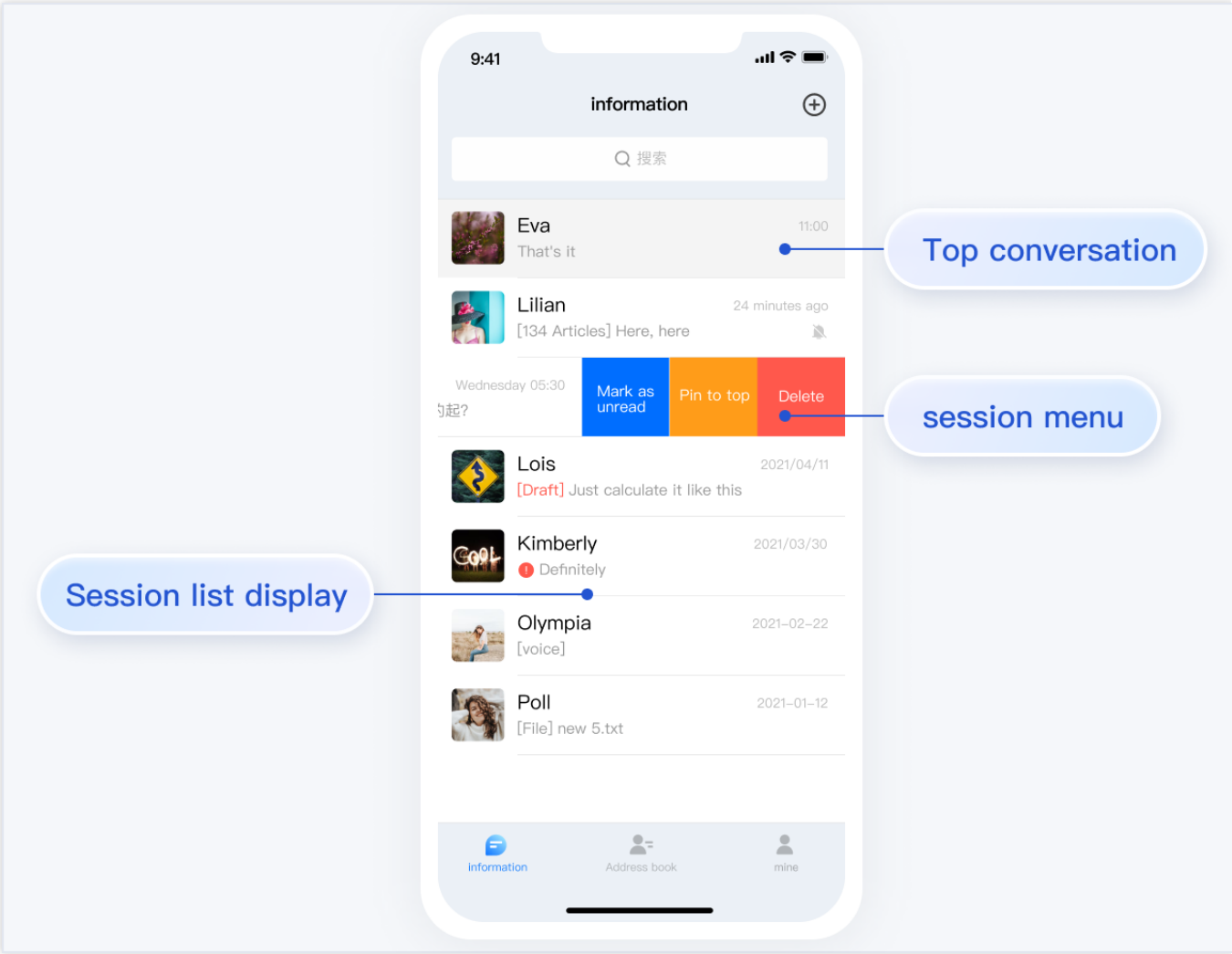
简约版

RTL语言

经典版




TUIConversation
Display the conversation list
17:53
الدردشة
تحرير
مثال يا صديقي
[استطلاع الرأي]
٠٧/٢٤
مثال يا صديقي
٠٧/٢١
وضع علامة كمقروء
مزيد
٠٧/٢٠
٠٧/١٩
مثال يا صديقي
@TGS#_@TGS#cCFHGIIM62CX
٠٧/١٧
hhha
تم الإلغاء
٠٧/١٣
public_group_test
test:11
٠٧/١٠
hhha ، مثال يا صديقي
's fast meeting
٠٧/١٠
work0707001
تم طرده من المجموعة "مثال يا صديقي"
٠٧/٠٧
11
الإعدادات
جهات الاتصال
سجل المكالمات
الرسائل
Conver
مثال يا صديقي
ydxhxugdhfjcofxhfj,hf,j,hfxufxufxuf
٠٧/٢١
وضع علامة كمقروء
المزيد
11
٠٧/٢٠
cc



TUIGroup 重点功能介绍

TUIGroup 主要负责群资料、群成员、群组权限的管理。

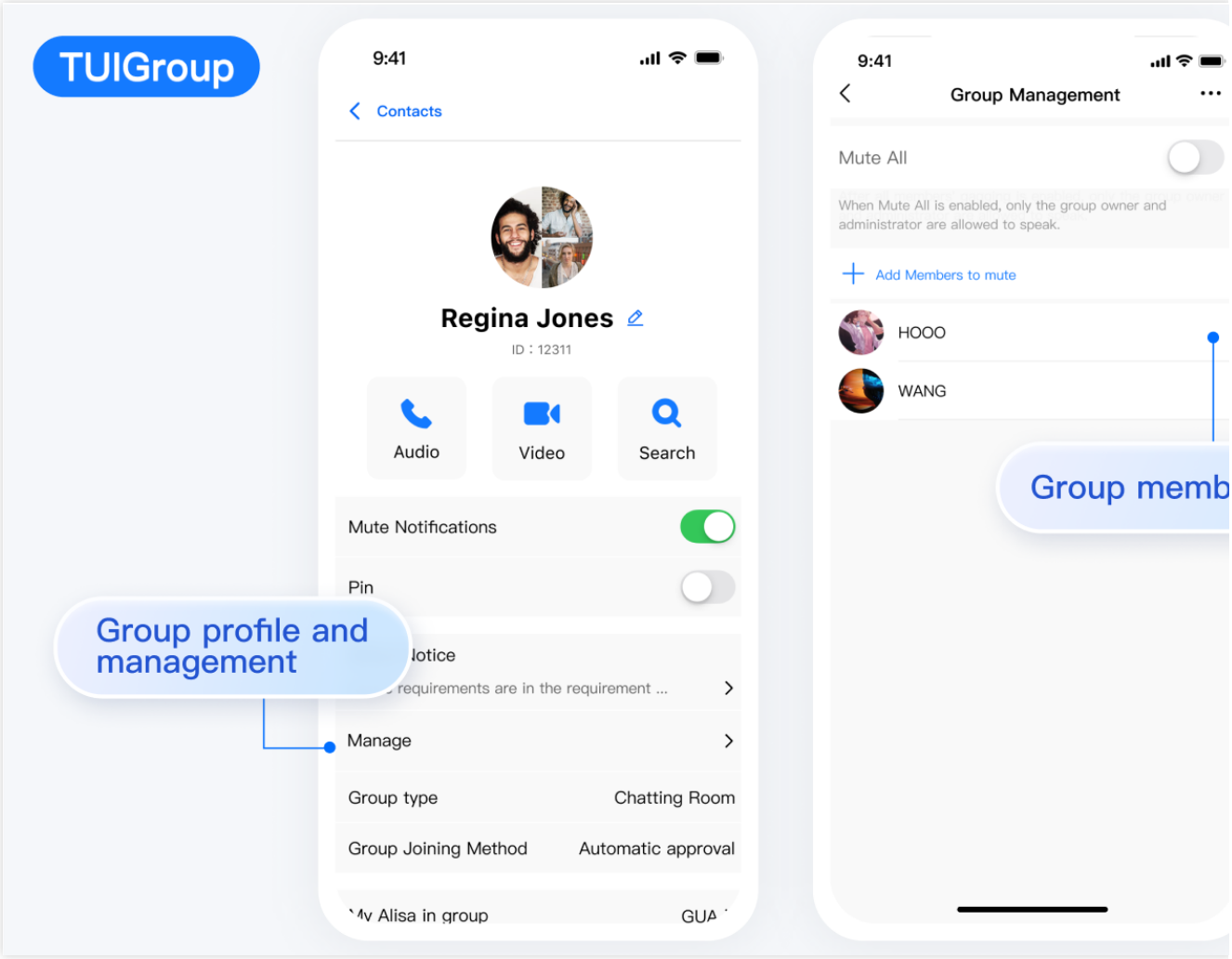
界面效果如下图所示：

简约版

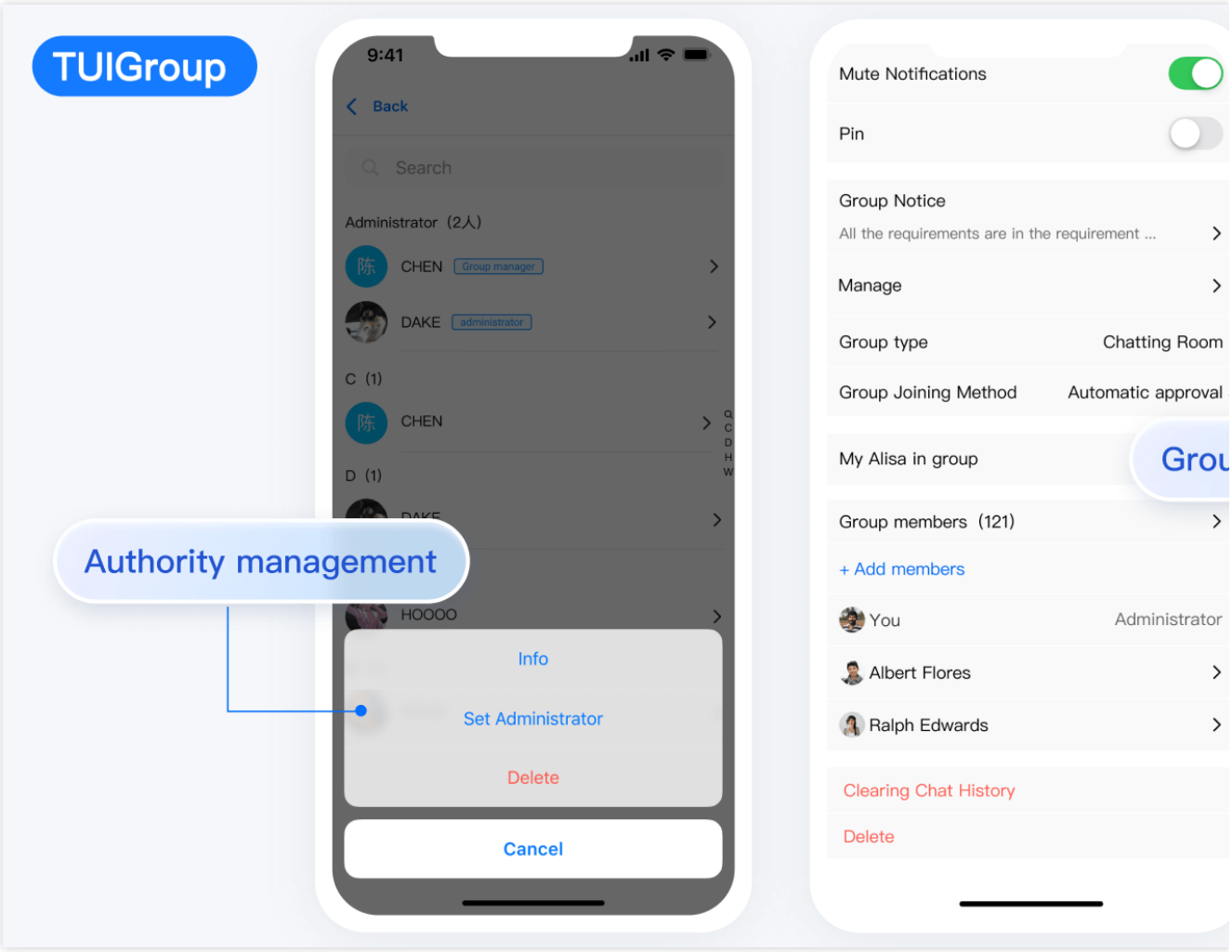
RTL语言

经典版

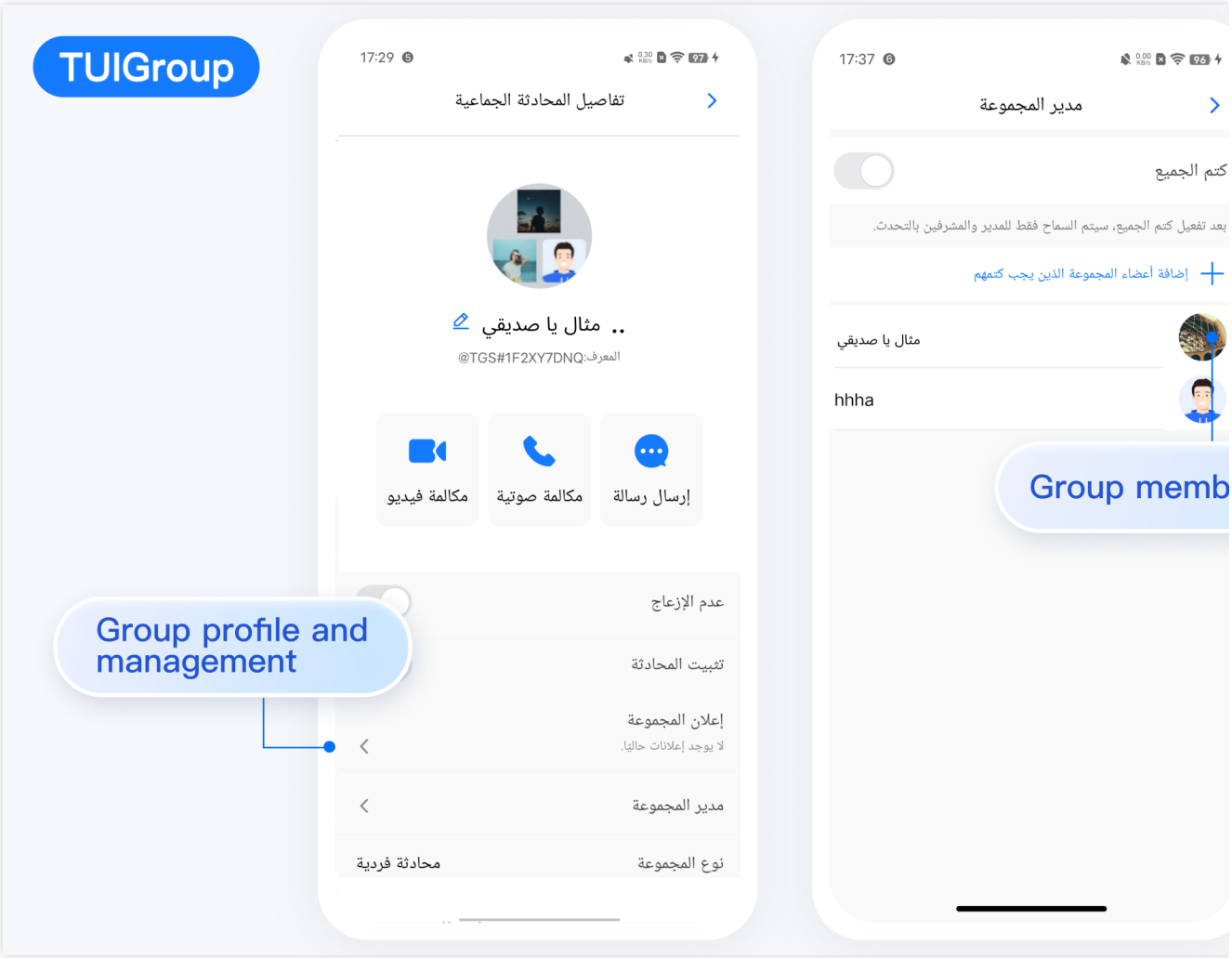
群资料及管理 | 群成员管理


TUIGroup
9:41
Contacts
Regina Jones
ID：12311
Audio
Video
Search
Mute Notifications
Pin
Group profile and management
Manage
Group type
Chatting Room
Group Joining Method
Automatic approval
9:41
Group Management
Mute All
When Mute All is enabled, only the group owner and administrator are allowed to speak.
Add Members to mute
HOOO
WANG
Group memb

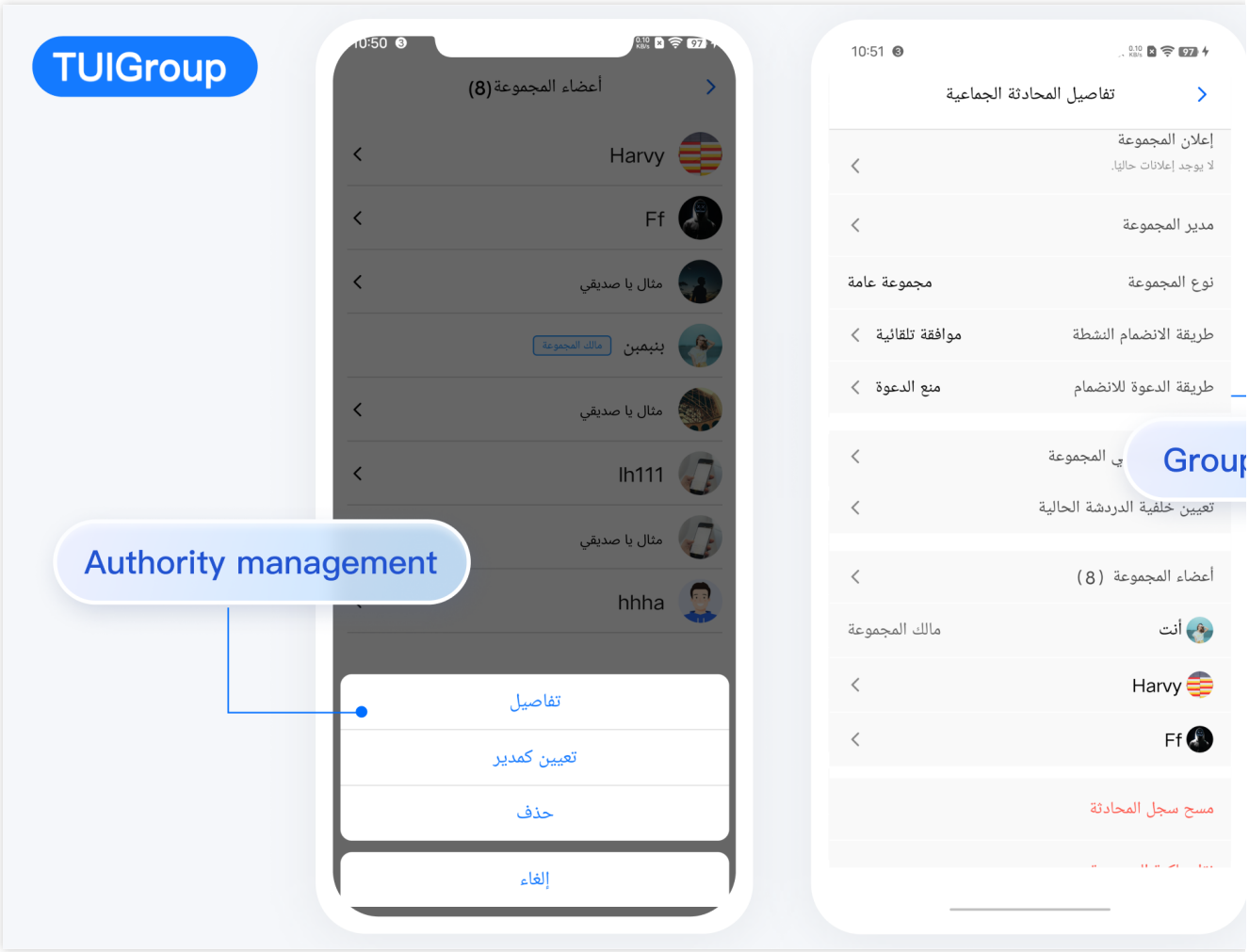
加群方式管理 | 权限管理



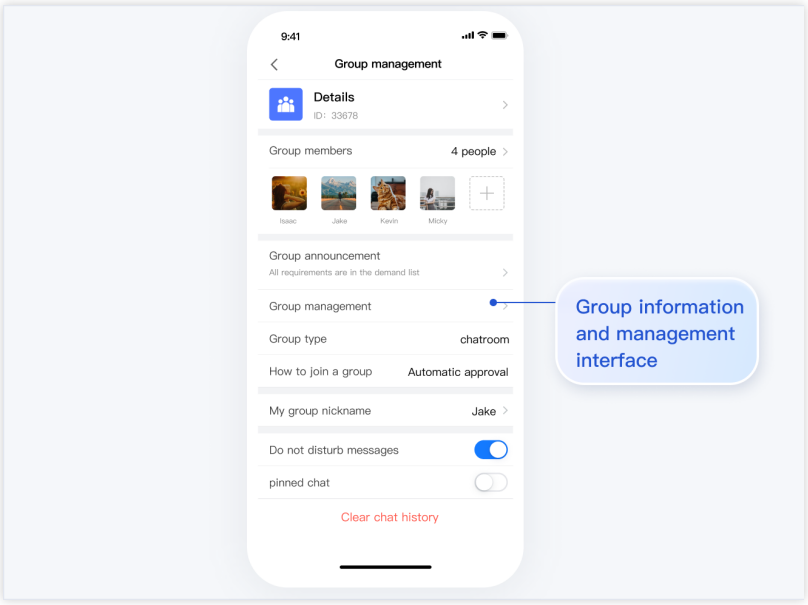
群资料及管理 | 群成员管理

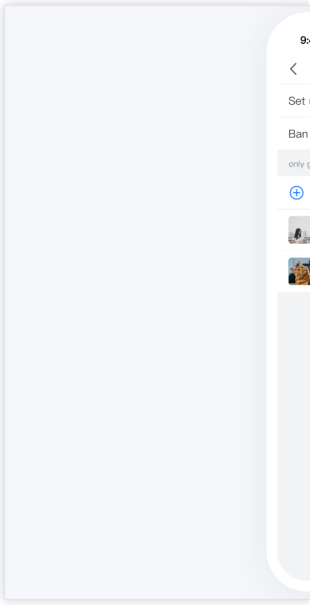

TUIGroup
17:29
تفاصيل المحادثة الجماعية
مثال يا صديقي ..
@TGS#1F2XY7DNQ:المعرف
مكالمة فيديو
مكالمة صوتية
إرسال رسالة
عدم الإزعاج
تثبيت المحادثة
إعلان المجموعة
لا يوجد إعلانات حاليًا.
مدير المجموعة
نوع المجموعة
محادثة فردية
Group profile and management
17:37
مدير المجموعة
كتم الجميع
بعد تفعيل كتم الجميع، سيتم السماح فقط للمدير والمشرفين بالتحدث.
إضافة أعضاء المجموعة الذين يجب كتمهم
مثال يا صديقي
hhha
Group memb

加群方式管理 | 权限管理



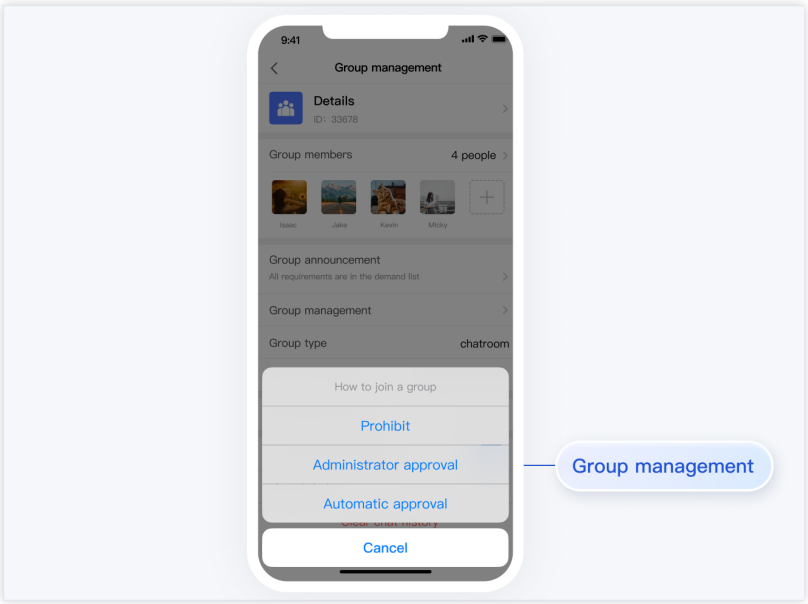
群资料及管理	群成员管理

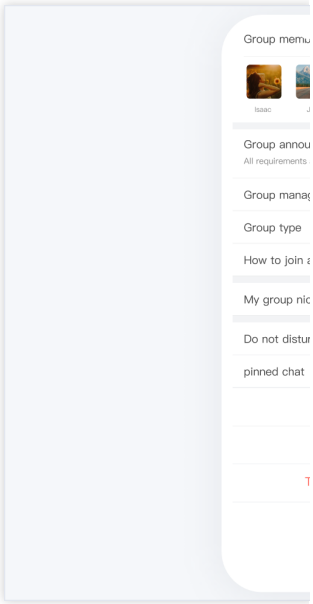




加群方式管理

权限管理





TUISearch 重点功能介绍

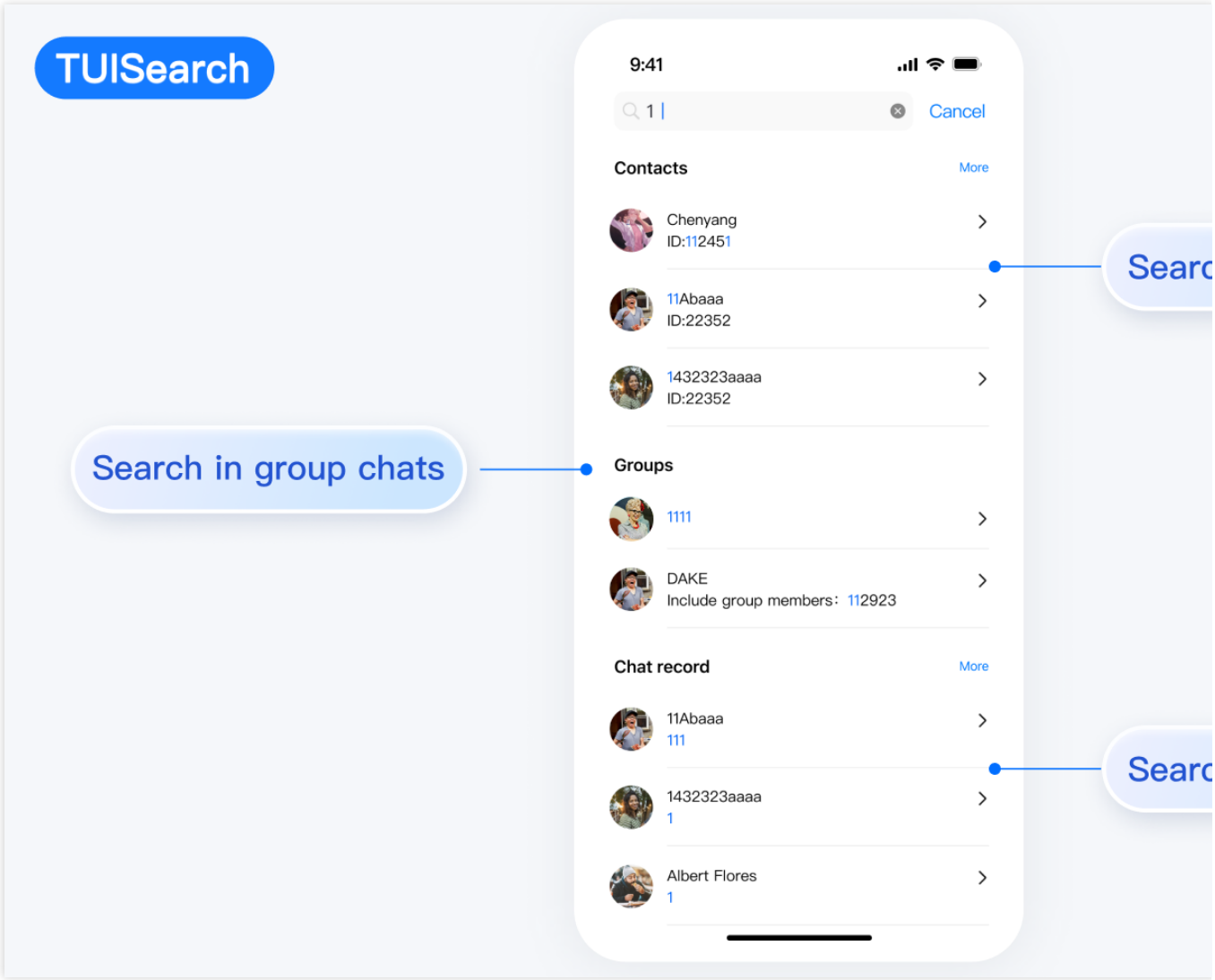
TUISearch 主要负责本地搜索，支持搜索联系人、群聊、聊天记录。

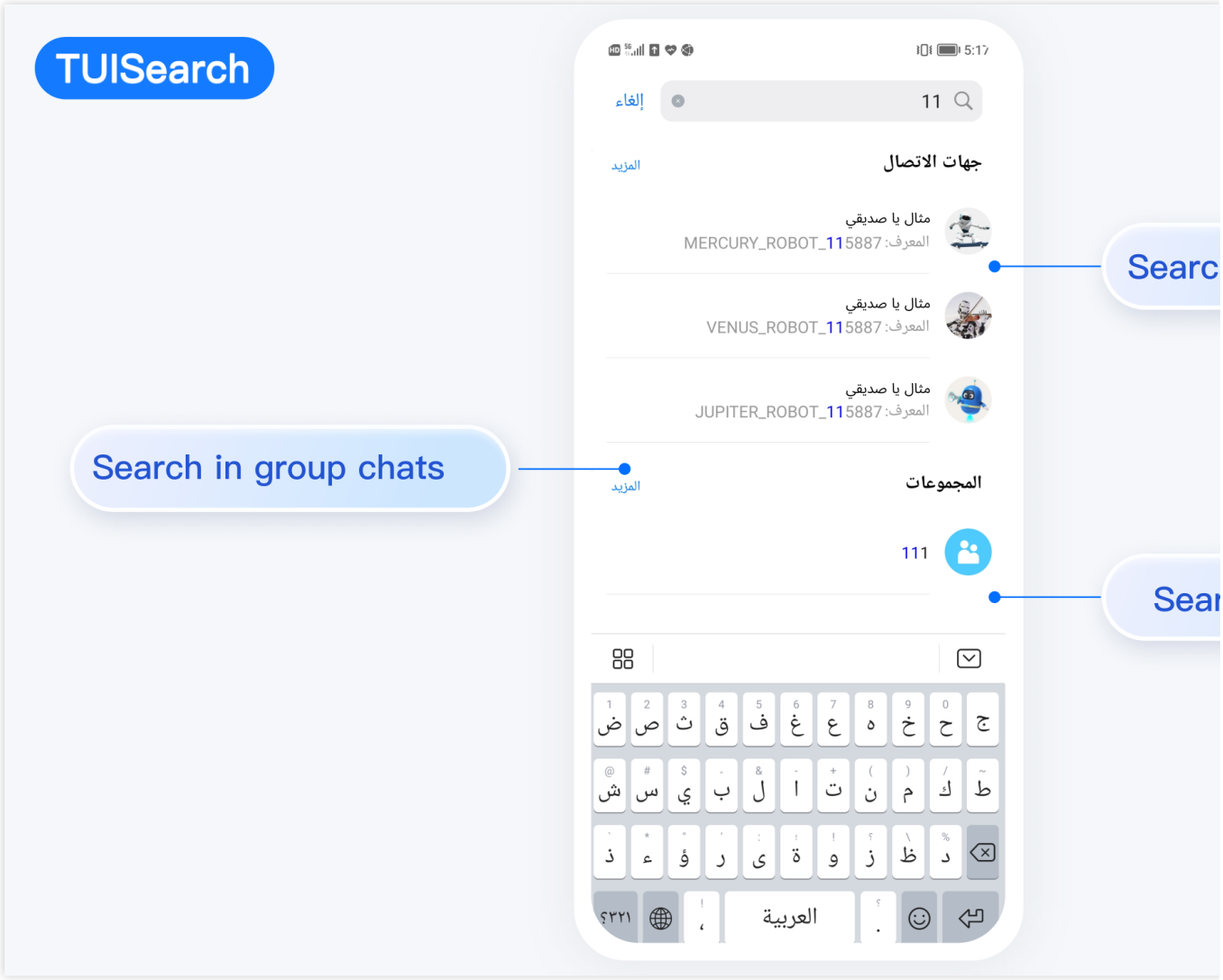
界面效果如下图所示：

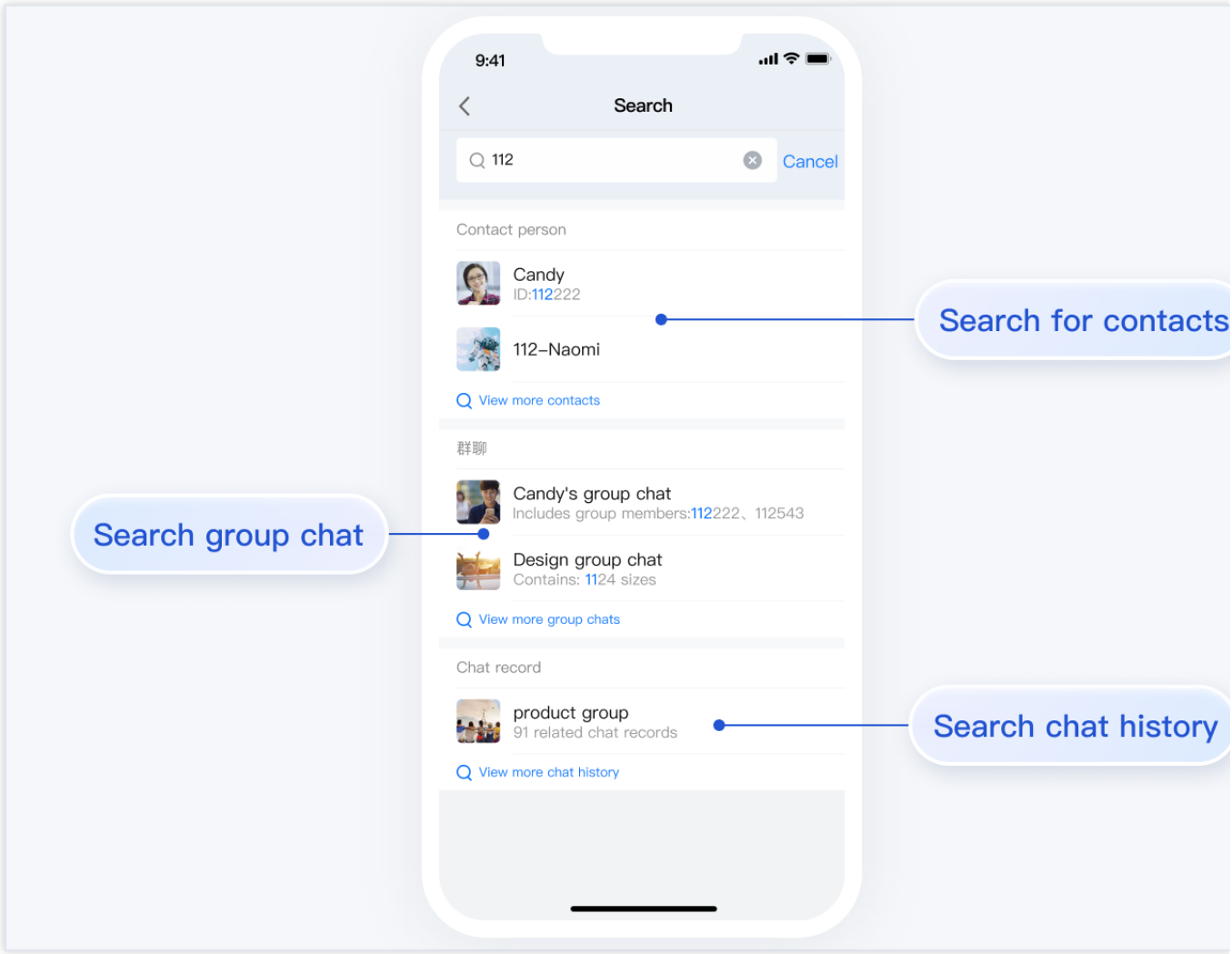
简约版

RTL语言

经典版




TUISearch
Search in group chats
Searc
Sear
إلغاء
11
جهات الاتصال
المزيد
مثال يا صديقي
المعرف: MERCURY_ROBOT_115887
مثال يا صديقي
المعرف: VENUS_ROBOT_115887
مثال يا صديقي
المعرف: JUPITER_ROBOT_115887
المجموعات
المزيد
111
العربية



TUICallKit 重点功能介绍

TUICallKit 主要负责语音、视频通话。

单聊通话示意图：

视频通话	语音通话

群聊通话示意图：

视频通话	语音通话

版权所有：腾讯云计算（北京）有限责任公司

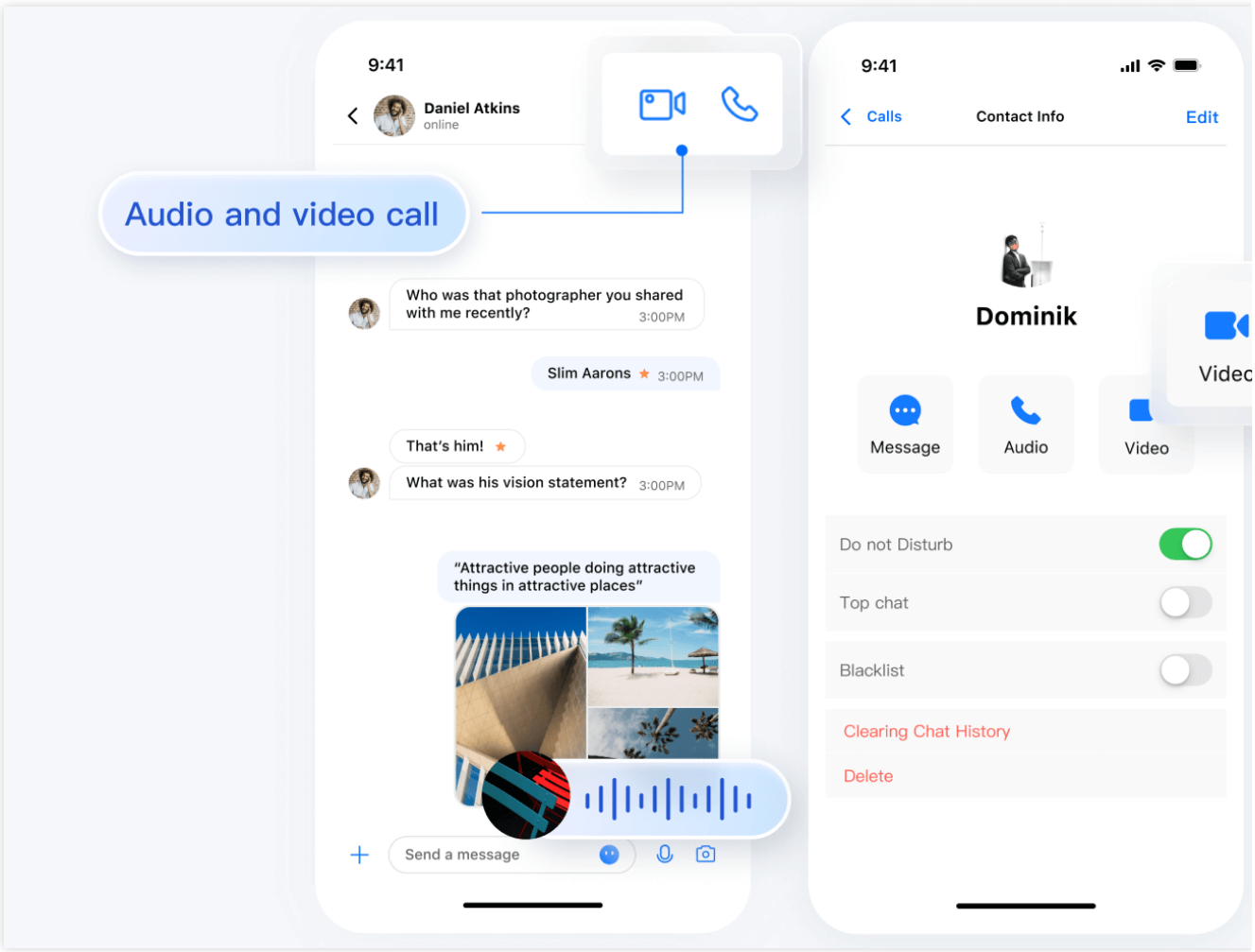
如果您集成了 TUIChat、TUIContact 及 TUICallKit，您可以在 TUIChat 消息页、TUIContact 个人资料页启动语音、视频通话。

界面效果如下图所示：

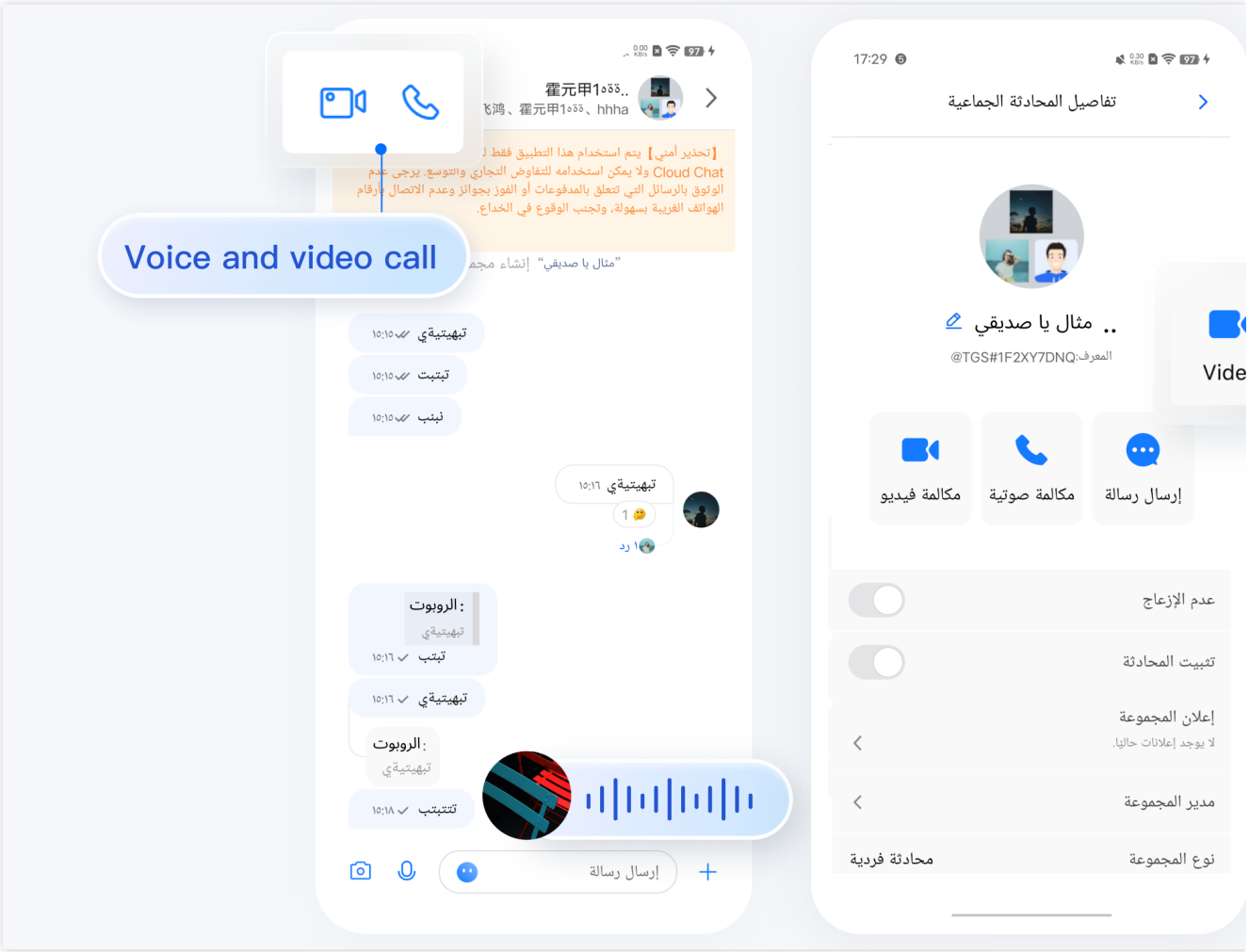
简约版

RTL语言

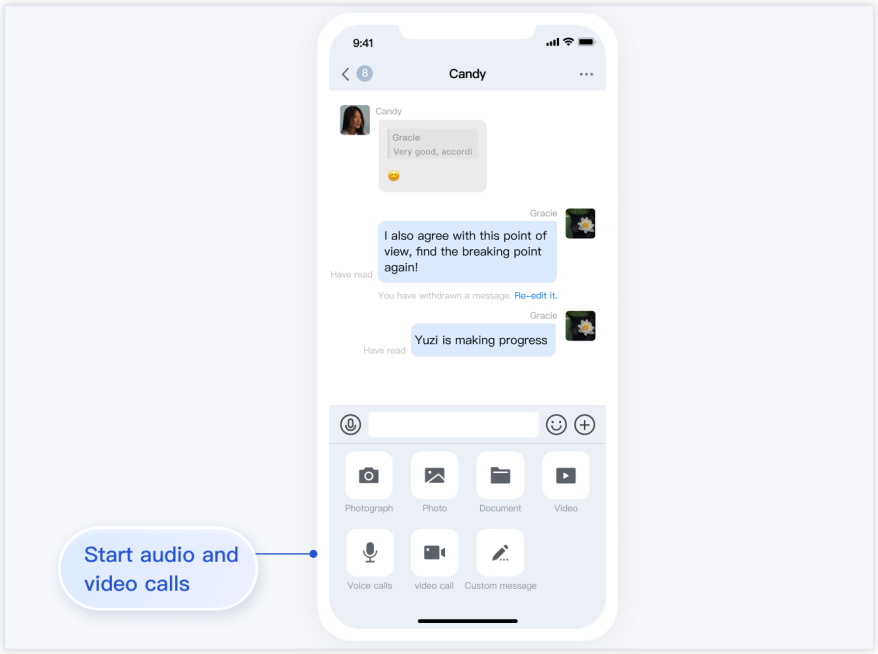
经典版

消息页启动 | 资料页启动


消息页启动 | 资料页启动


Voice and video call
تفاصيل المحادثة الجماعية
مثال يا صديقي ..
@TGS#1F2XY7DNQ:المعرف
مكالمة فيديو
مكالمة صوتية
إرسال رسالة
عدم الإزعاج
تثبيت المحادثة
إعلان المجموعة
لا يوجد إعلانات حاليا.
مدير المجموعة
نوع المجموعة
محادثة فردية
إرسال رسالة

消息页启动	资料页启动



TUIOfflinePush 重点功能介绍

TUIOfflinePush 主要负责离线推送消息展示。

离线推送效果如下图所示：

TUIOfflinePush
9:41
09:34
06/28 Wednesday
Grope
Let's meet at the meeting later
5 mins ago
Nicole
[Image]Please compare the adjusted ...
20 mins ago
James
[Video call]
1 hour ago
Swipe up to unlock
Video call
Group
Image

Web

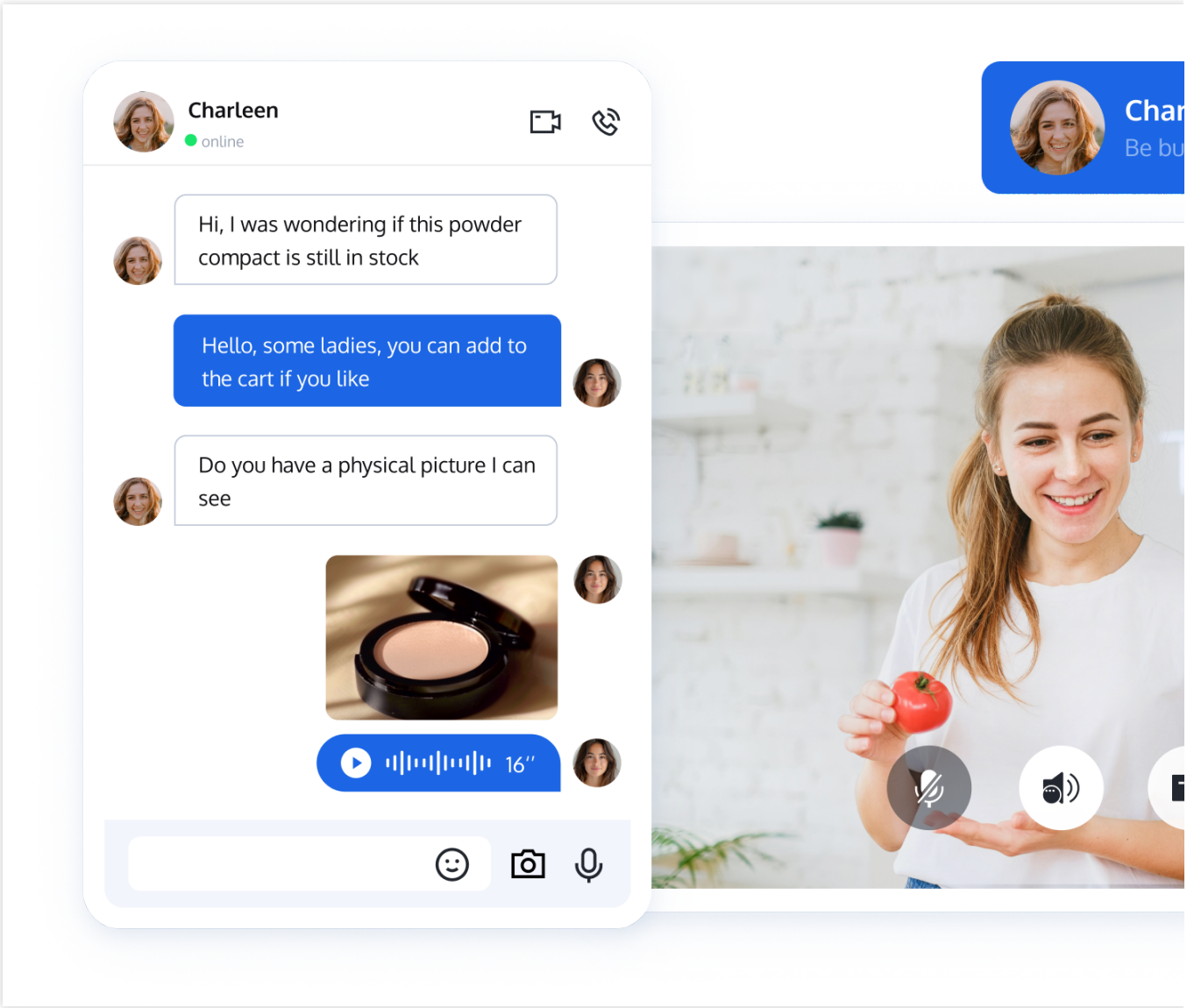
最近更新时间：2024-08-14 16:15:46

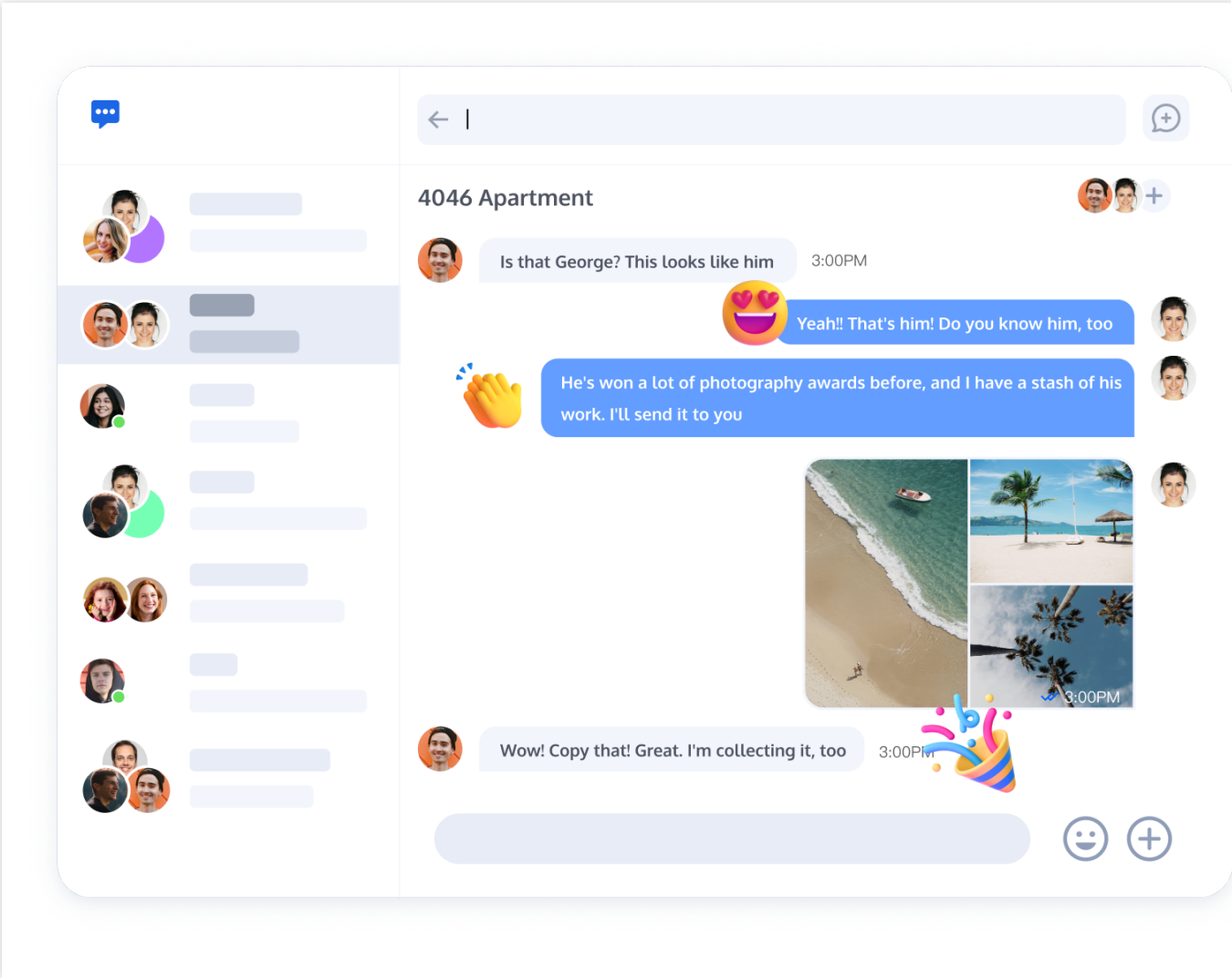
Chat 提供多平台聊天 API、 UI 组件、服务端 API 和 webhook，使您能够在十分钟内构建一个功能齐全的聊天应用程序。
您可以直接体验下面的 Chat UIKit Demo。同时，您可以通过 **体验沙箱** 快速体验在线代码实现。

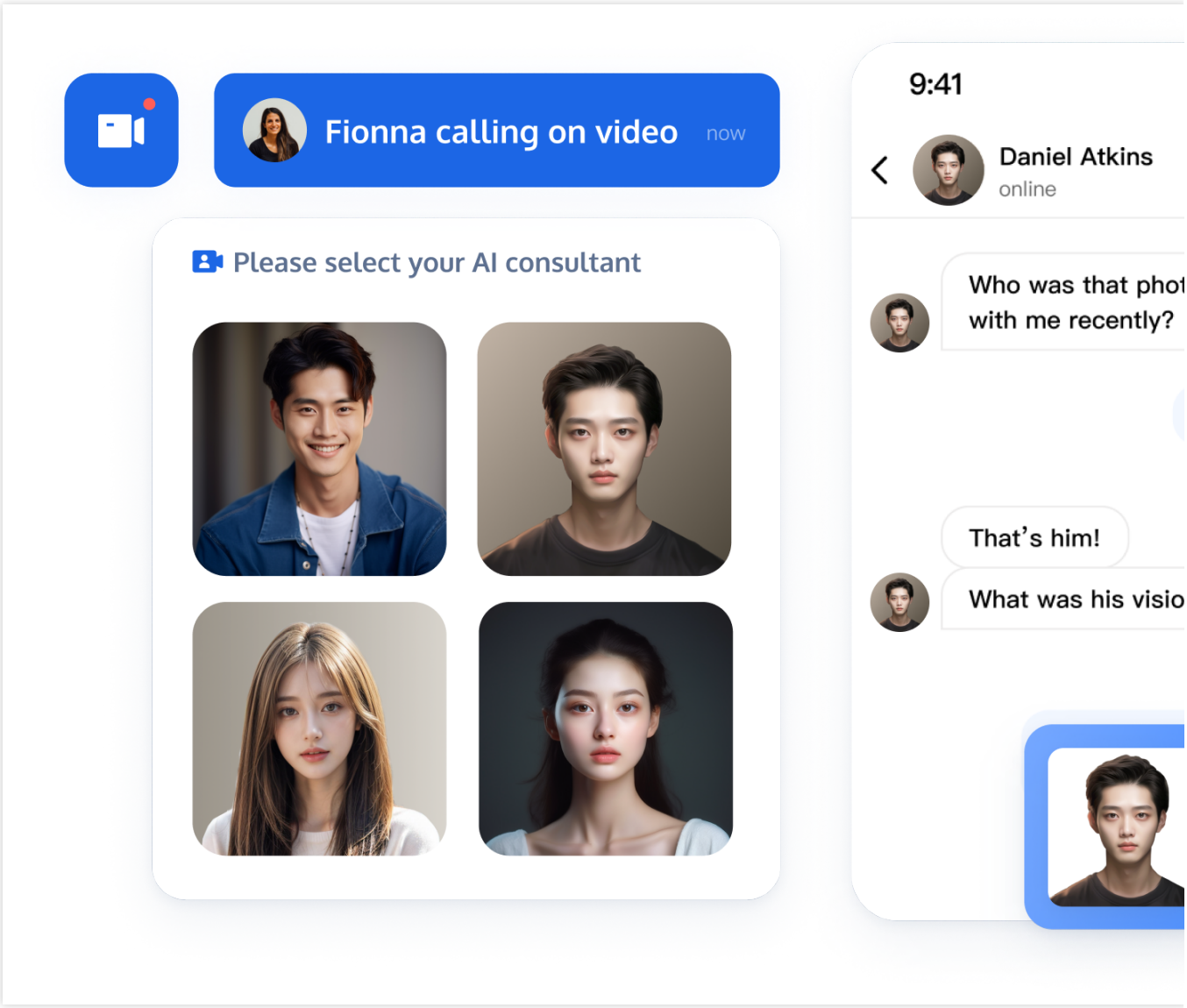
应用场景

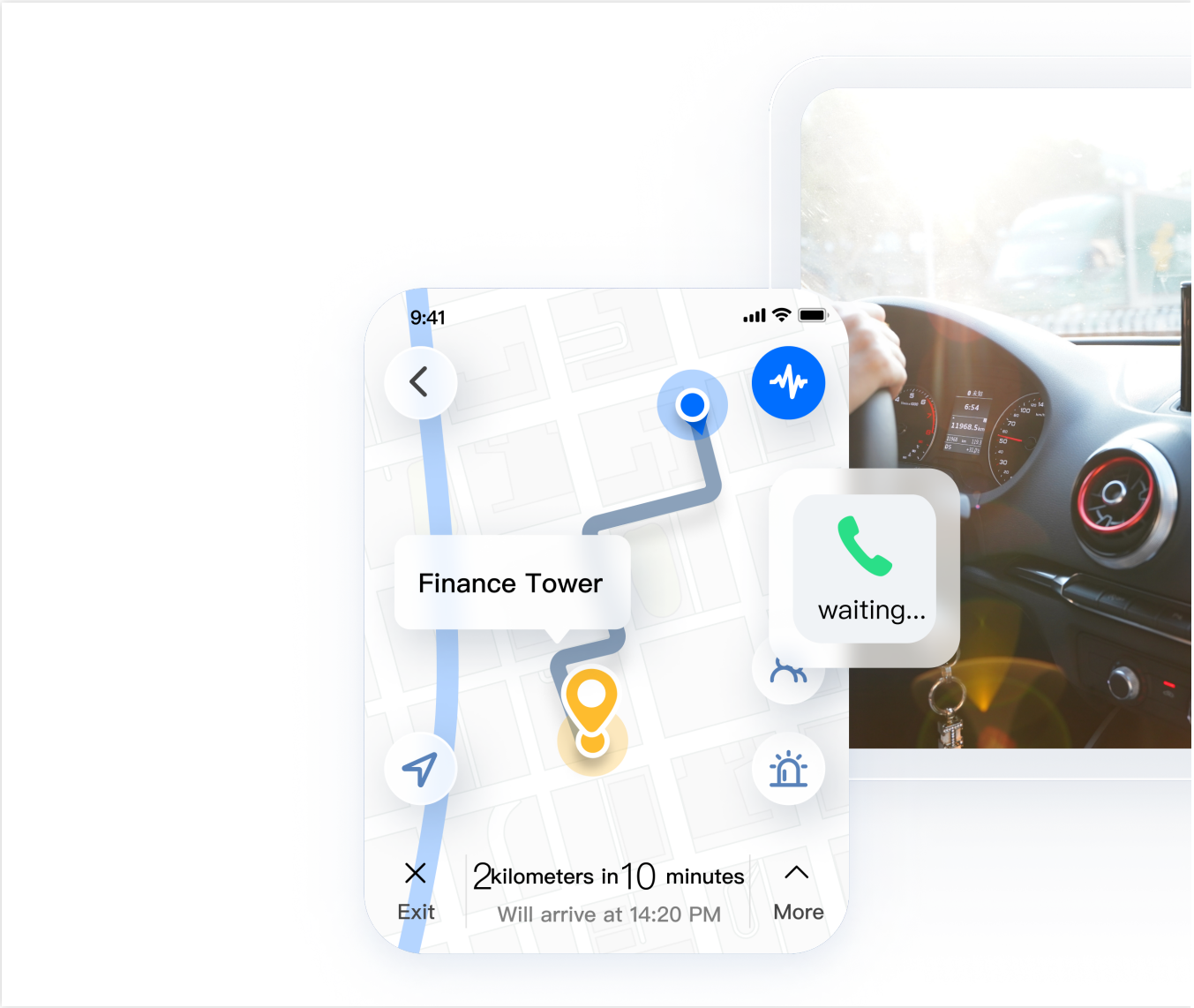
直播互动
零售电商
社交娱乐
对话式 AI
点播服务
游戏开黑
在线教育
医疗服务

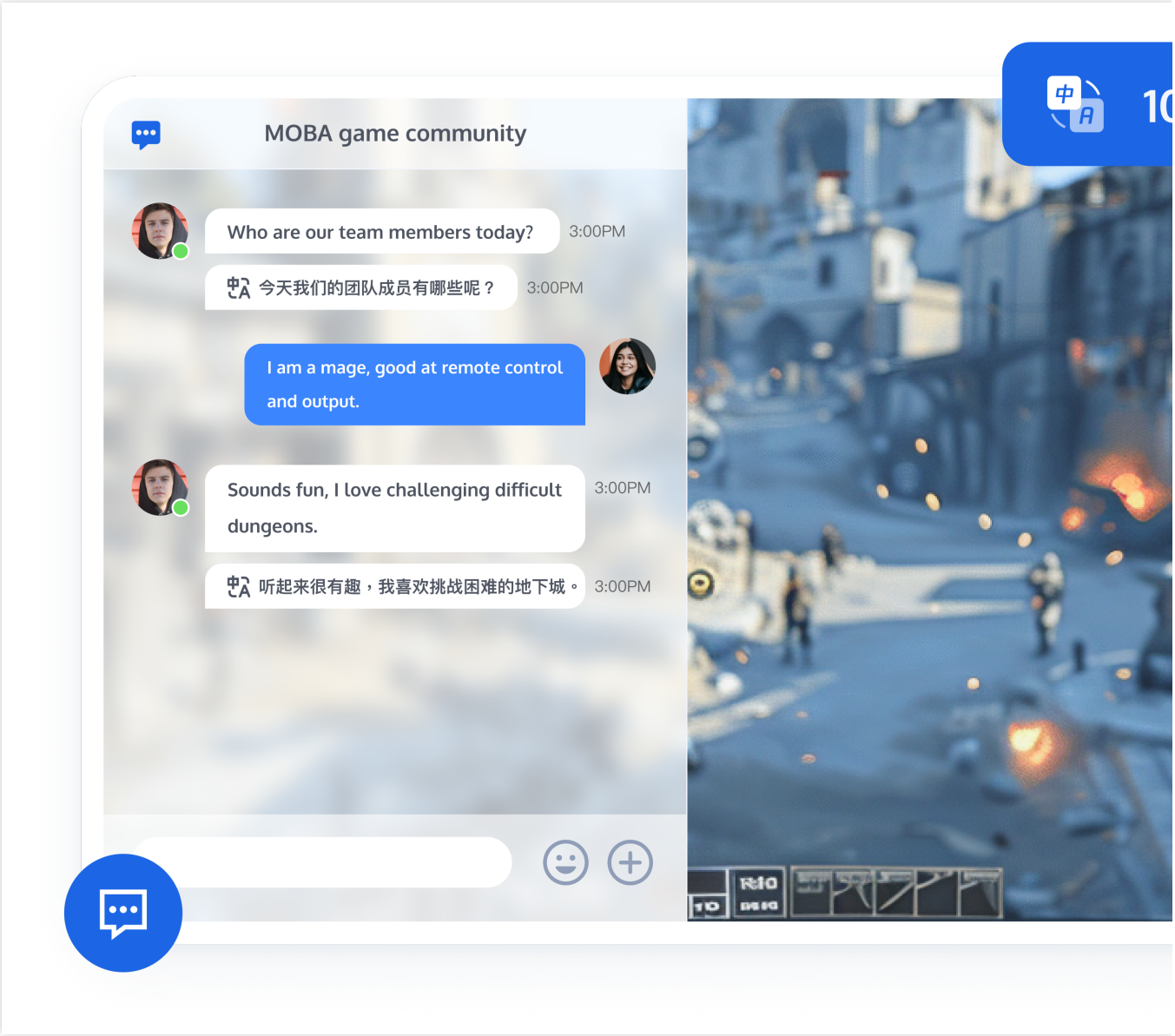
320
Emma
99
Sophia
0
Charlotte
0
Ava
0
Ryan
8
9
Spektral0...
Sent a Sports car
water and caught a cod !
99 Nightbot : EbilDoggies cast their line in
water and caught a cod !
16 Ghost_Beretta Enter the room
65 ANCHOR Spektral0_0 : modCheck
Let's talk

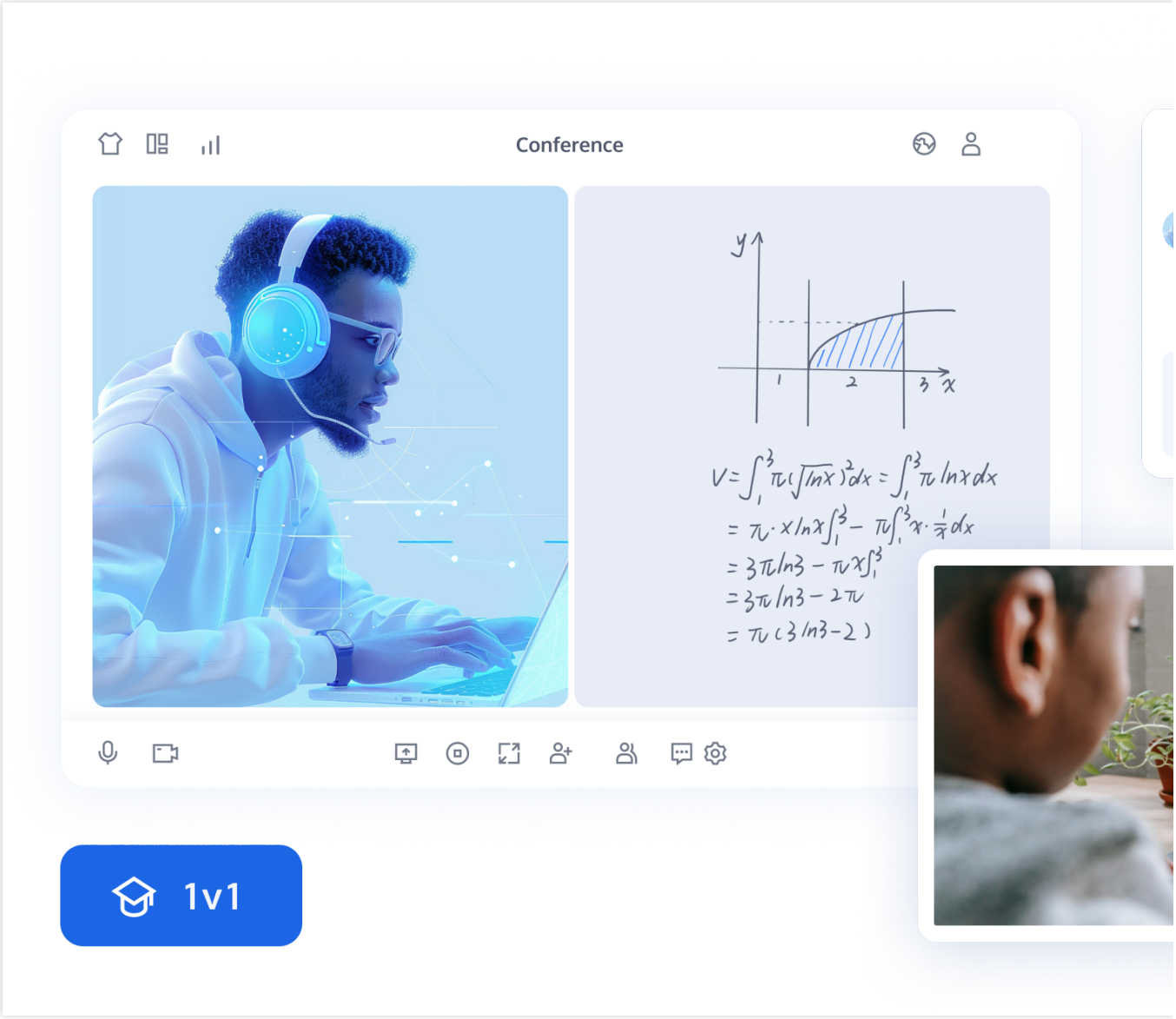

Charleen
online
Hi, I was wondering if this powder compact is still in stock
Hello, some ladies, you can add to the cart if you like
Do you have a physical picture I can see
16''


4046 Apartment
Is that George? This looks like him
3:00PM
Yeah!! That's him! Do you know him, too
He's won a lot of photography awards before, and I have a stash of his work. I'll send it to you
3:00PM
Wow! Copy that! Great. I'm collecting it, too
3:00PM


Fionna calling on video
now
Please select your AI consultant
9:41
Daniel Atkins
online
Who was that phot
with me recently?
That's him!
What was his visio


9:41
Finance Tower
waiting...
Exit
2kilometers in10 minutes
Will arrive at 14:20 PM
More


MOBA game community
Who are our team members today?
3:00PM
今天我们的团队成员有哪些呢？
3:00PM
I am a mage, good at remote control and output.
Sounds fun, I love challenging difficult dungeons.
3:00PM
听起来很有趣，我喜欢挑战困难的地下城。
3:00PM


Conference
1v1

UIKit

Chat

Conversation

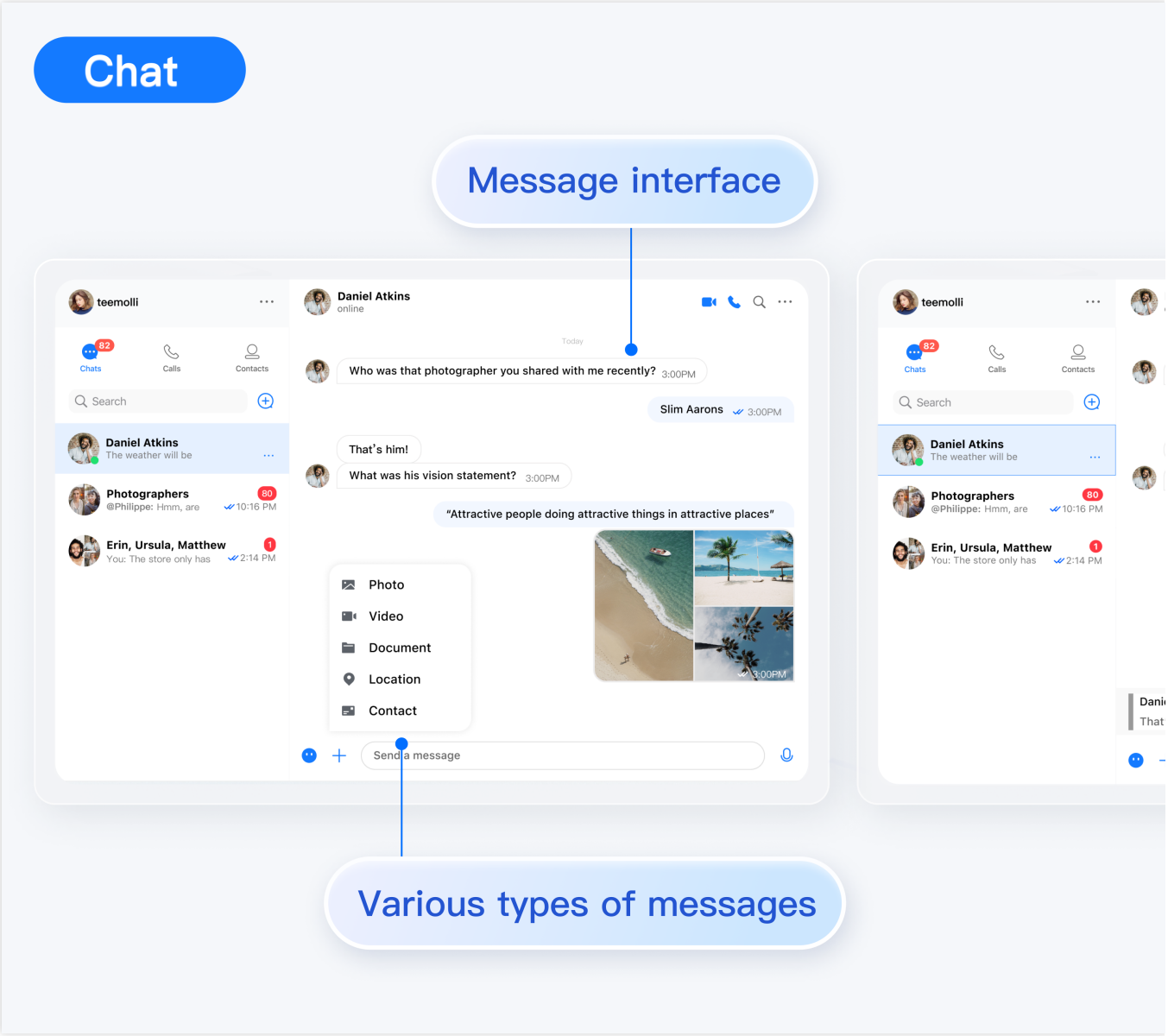
Group

Contact

Profile

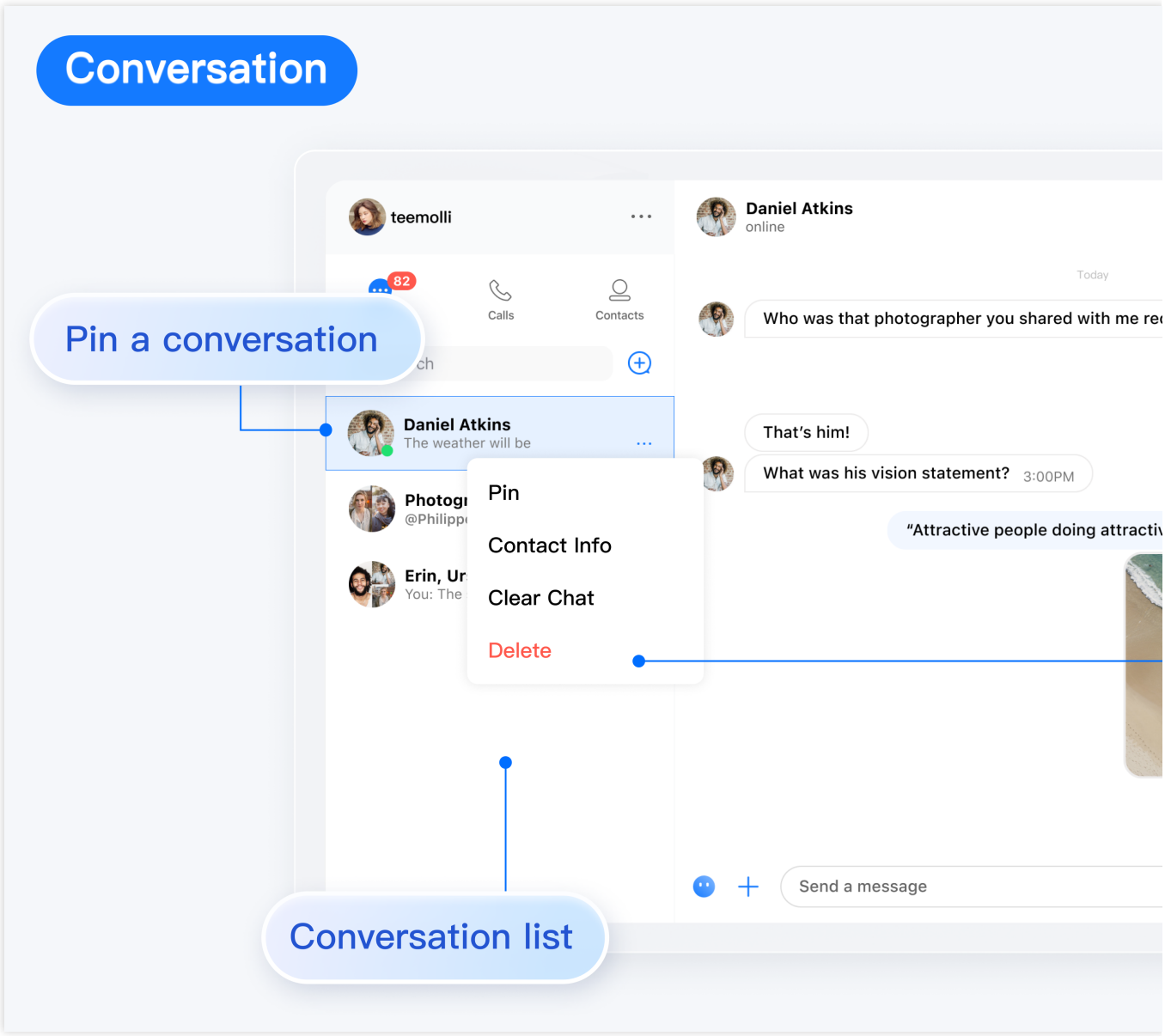
Chat 组件主要负责消息界面的展示。您可以利用它直接发送不同类型的消息，支持文字/表情/图片/视频/文件/自定义等消息类型，同时支持消息转发/撤回/引用、查询、已读回执等功能。

版权所有：腾讯云计算（北京）有限责任公司

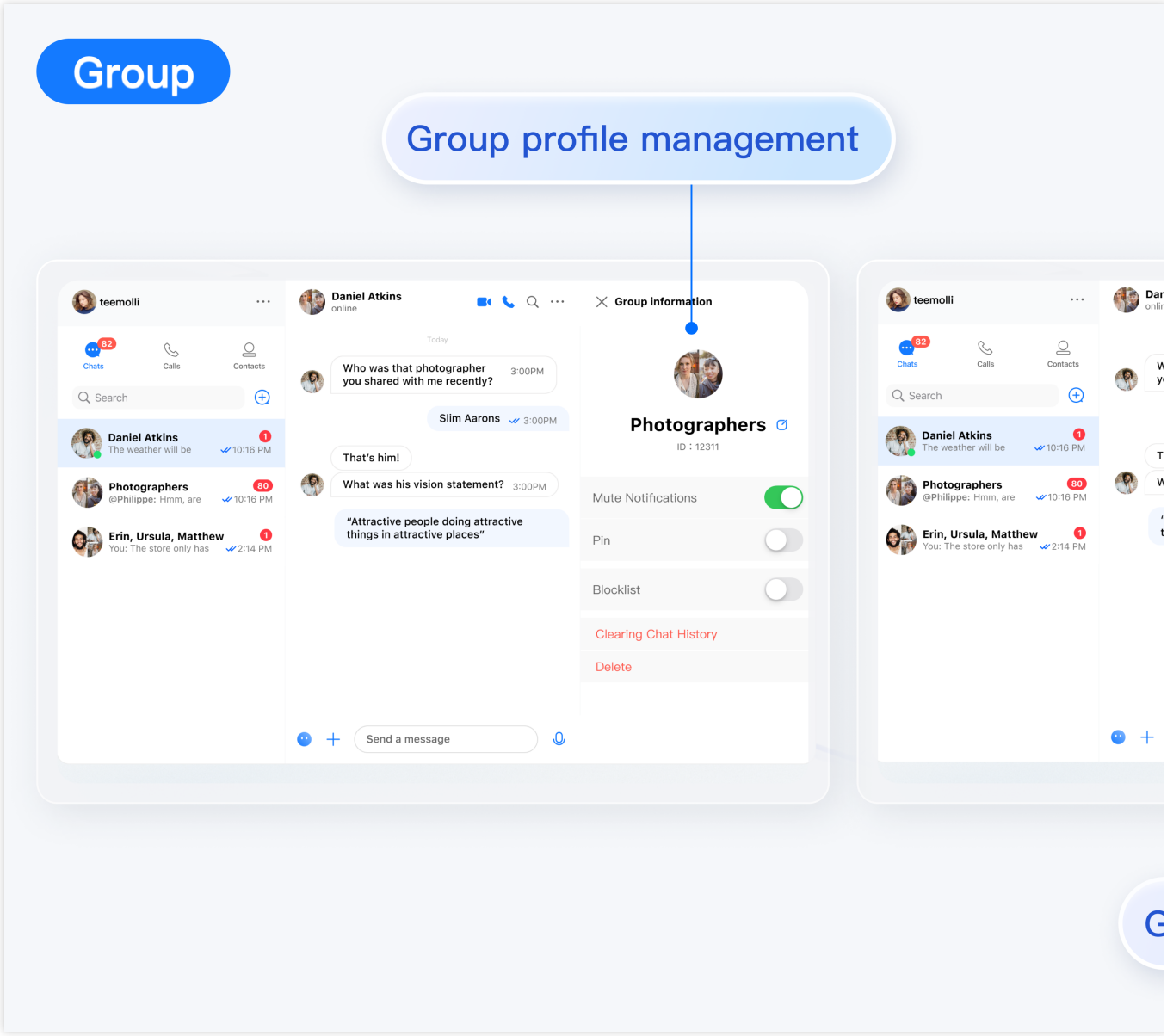


Conversation 组件主要负责会话列表的展示和编辑，包含会话置顶、会话删除等功能。

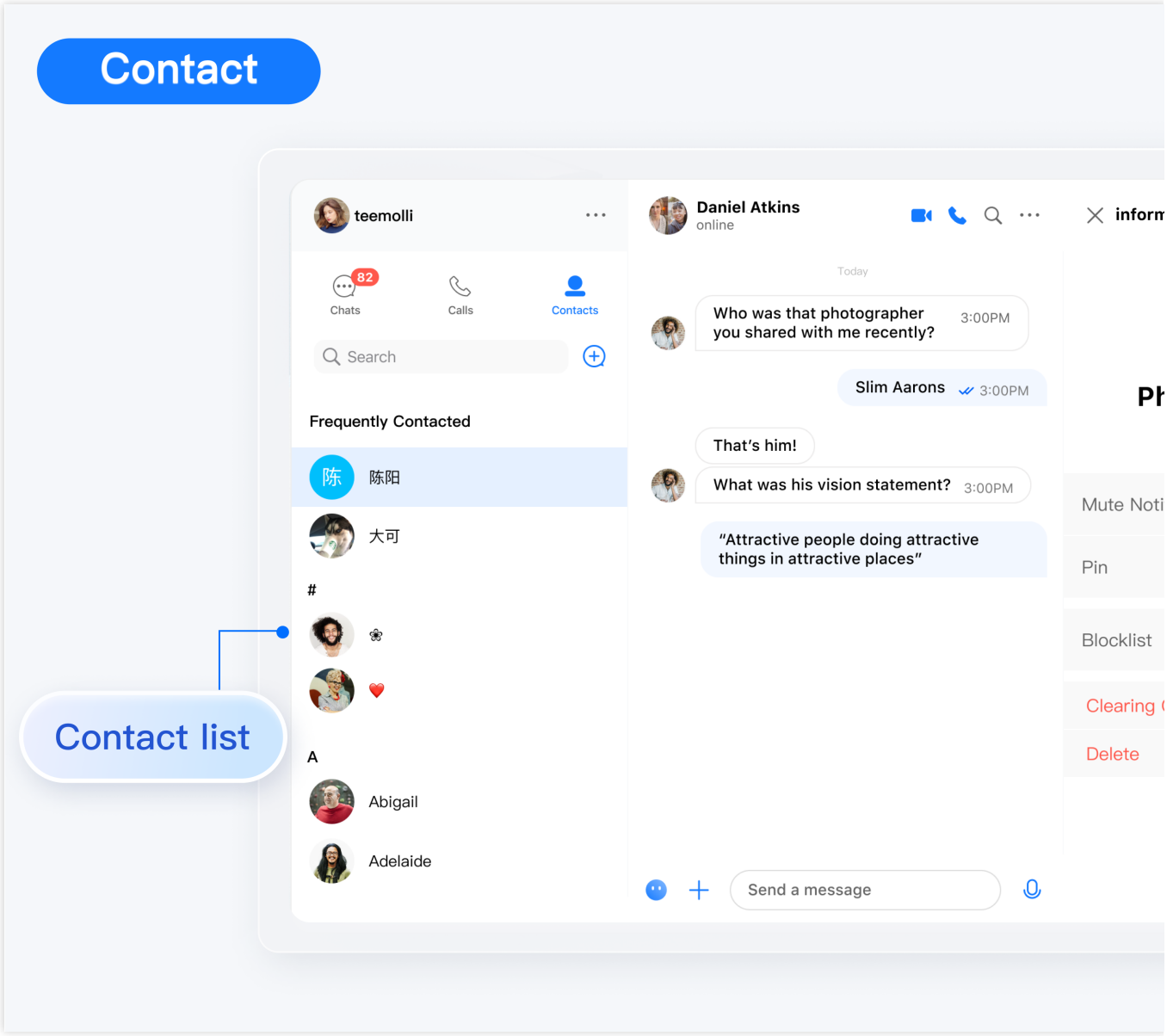
版权所有：腾讯云计算（北京）有限责任公司



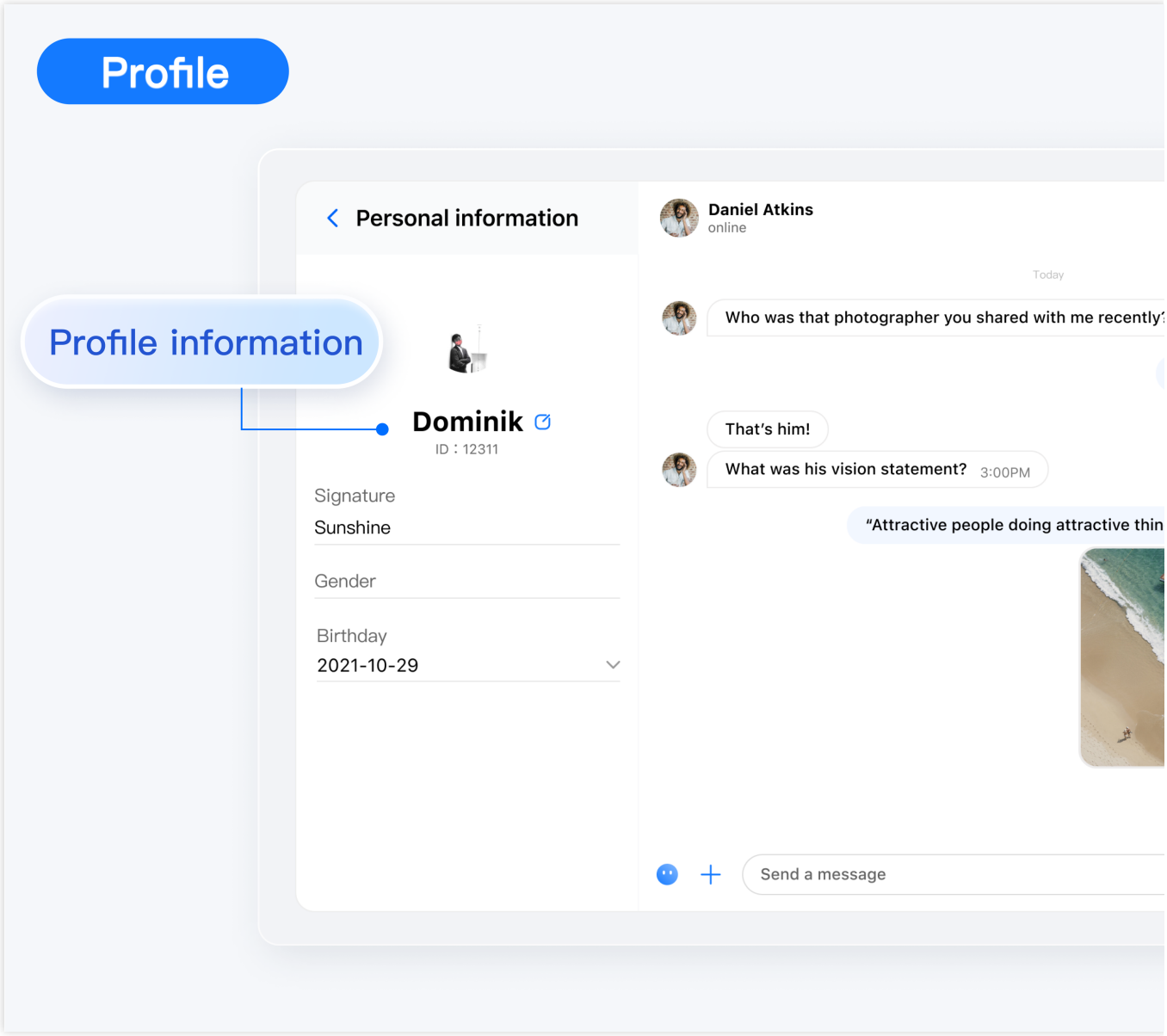
Group 组件主要负责群资料、群成员、群组权限的管理。



Contact 组件主要负责关系链展示以及创建会话等操作。



Profile 组件主要负责用户资料的管理。



平台	React	Vue	Android	iOS and macOS	Flutter
是否支持					
体验 Demo	点击体验	点击体验			点击体验

跑通 Github Demo	**React**	**Vue**	**Android**	**iOS and macOS**	**Flutter**
快速集成 UI	**React**	**Vue**	**Android**	**iOS and macOS**	**Flutter**

SDK

平台	Web	Android	iOS and macOS	Flutter	React Native	Windows	Unity	Unreal Engine
是否支持								
集成 SDK	**JavaScript**	**Android**	**iOS macOS**	**Flutter**	**React Native**	**Windows**	**Unity**	**Unreal Engine**

产品能力

消息	数据统计	群管理	用户资料与关系链
单聊 群聊 推送 表情回应 合并消息 撤回消息 消息回复 文件分享 消息搜索 多端同步消息 已读回执	数据看版 用户活跃度 消息总量 发消息用户数 最高在线人数 用户注册数 当前在线人数 数据导出	群资料配置 成员禁言 @群成员 群成员管理 设置管理员 未读计数 踢人	添加好友 用户搜索 黑名单 用户资料设置 好友申请 好友备注 好友分组

版权所有：腾讯云计算（北京）有限责任公司

未读计数 免打扰 消息翻译 对方正在输入			

交流与反馈

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

H5

最近更新时间：2024-05-31 14:59:51

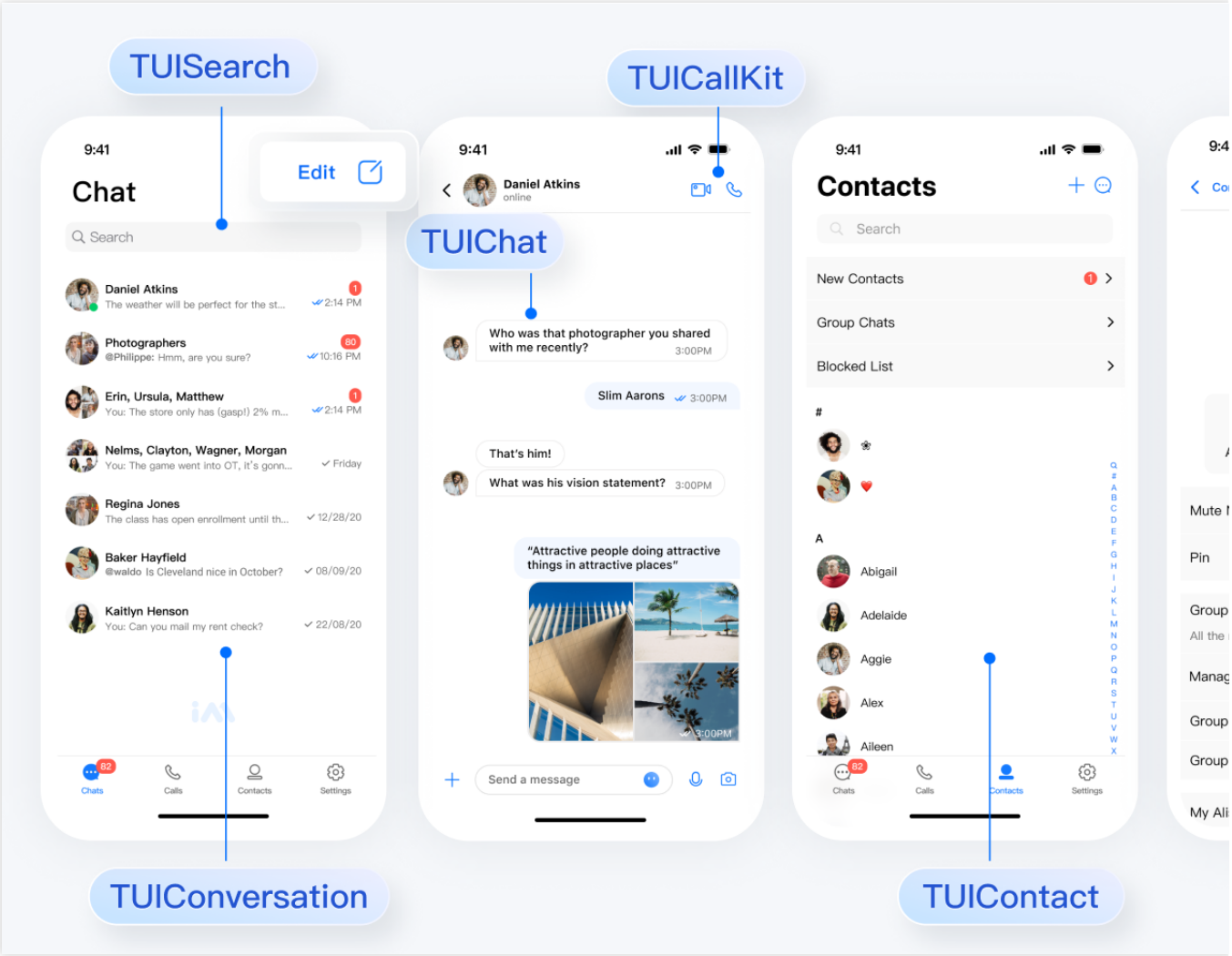
TUIKit 介绍

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

TUIKit 主要功能介绍

TUIKit 主要分为 TUIChat、TUIConversation、TUIGroup、TUIContact、TUIProfile几个 UI 子组件，每个 UI 组件负责展示不同的内容。

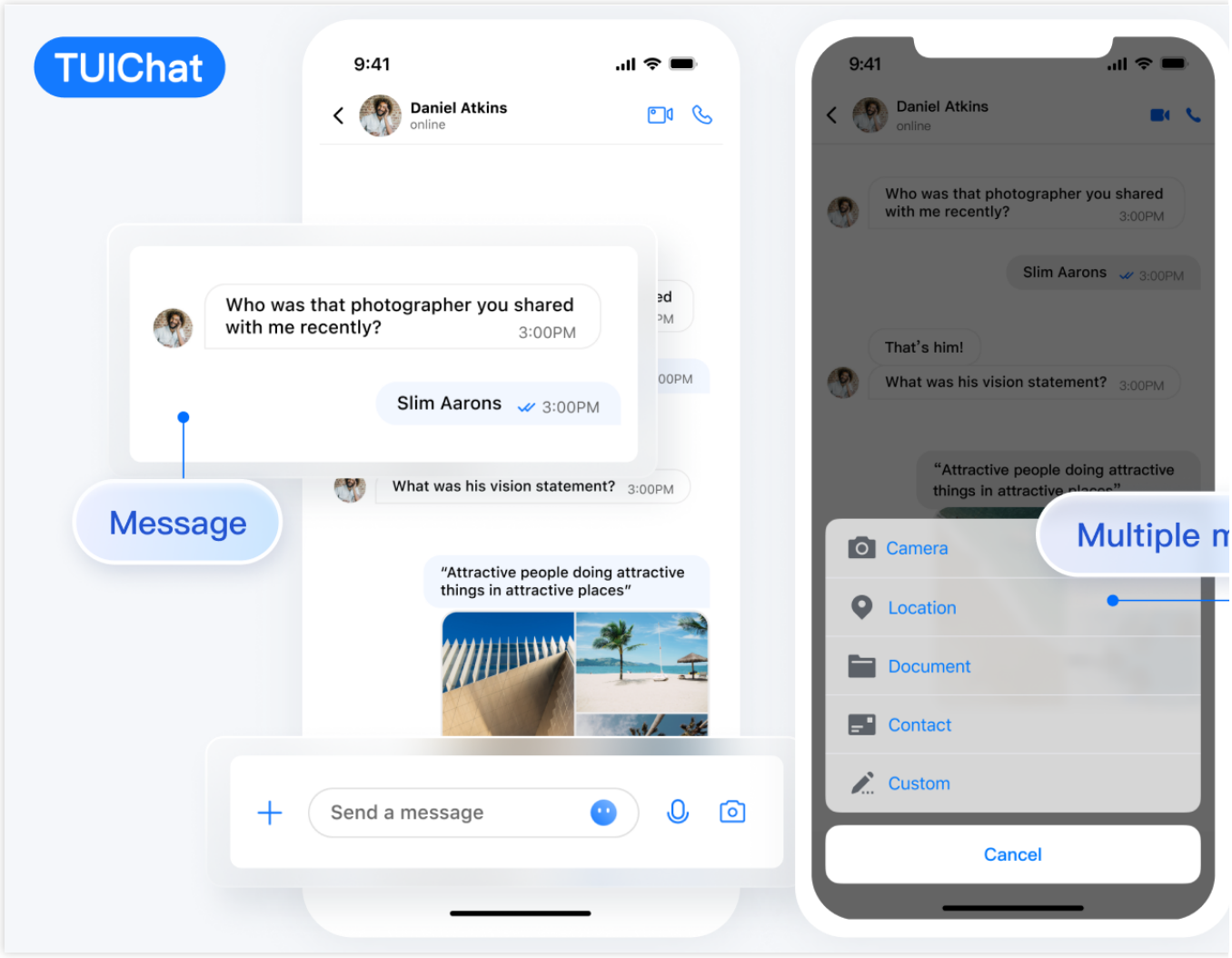
版权所有：腾讯云计算（北京）有限责任公司



TUIChat 重点功能介绍

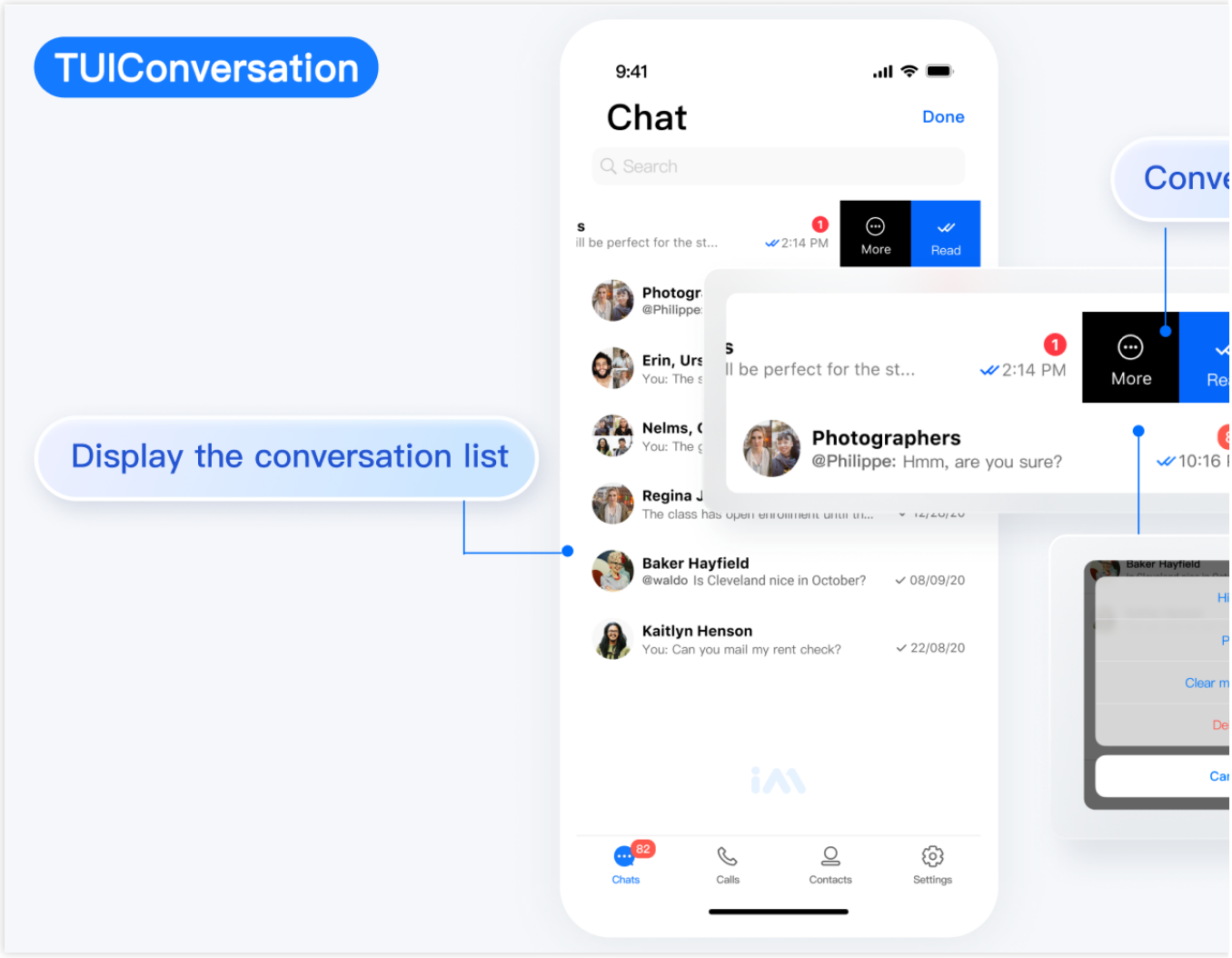
TUIChat 主要负责消息界面的展示。您可以利用它直接发送不同类型的消息，支持文字/表情/图片/视频/文件/自定义等消息类型，同时您还可以利用它对消息长按进行转发/撤回/引用、查询消息已读回执详情等。

版权所有：腾讯云计算（北京）有限责任公司



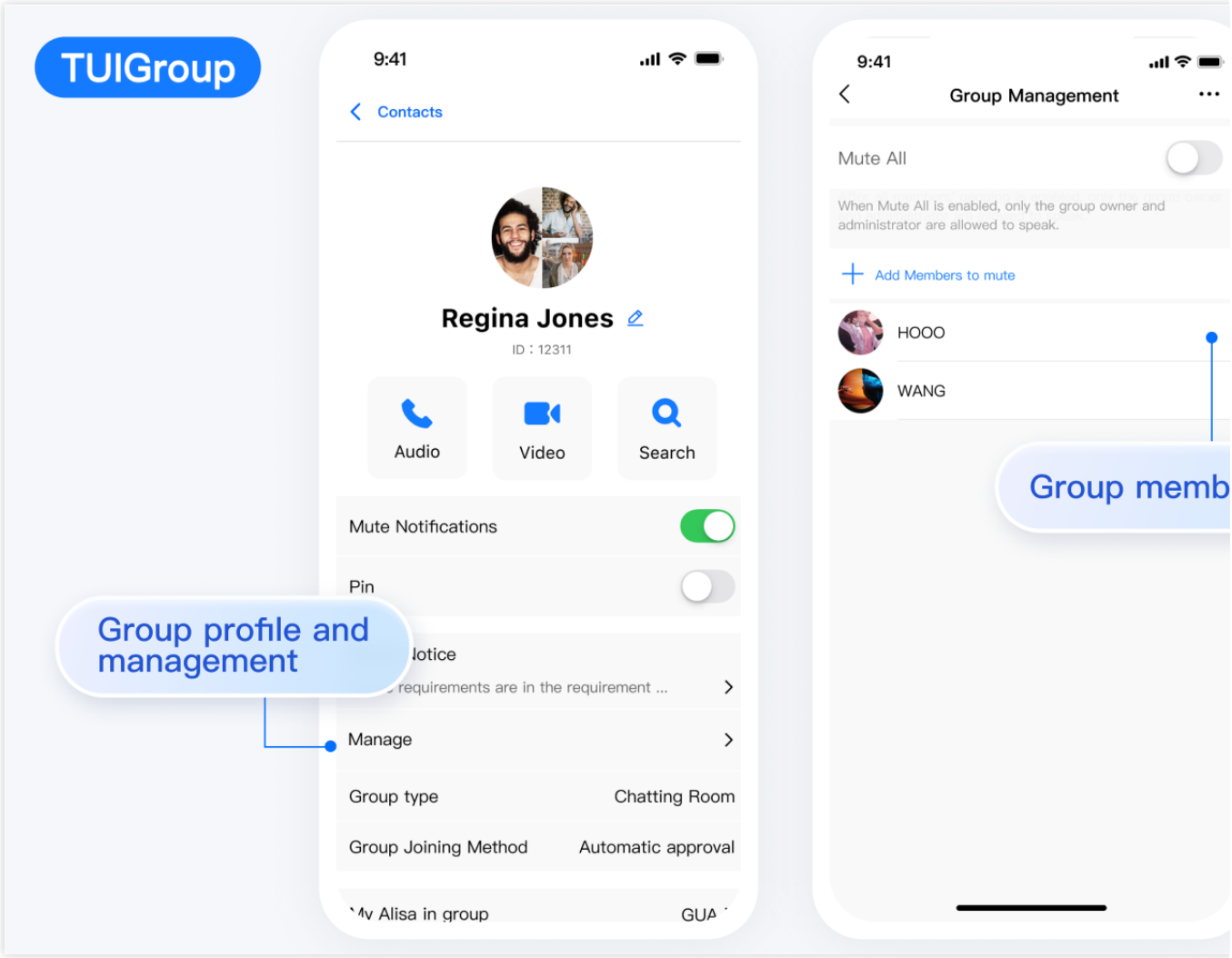
TUIConversation 重点功能介绍

TUIConversation 主要负责会话列表的展示和编辑，包括会话置顶、会话消息免打扰、会话删除等功能。



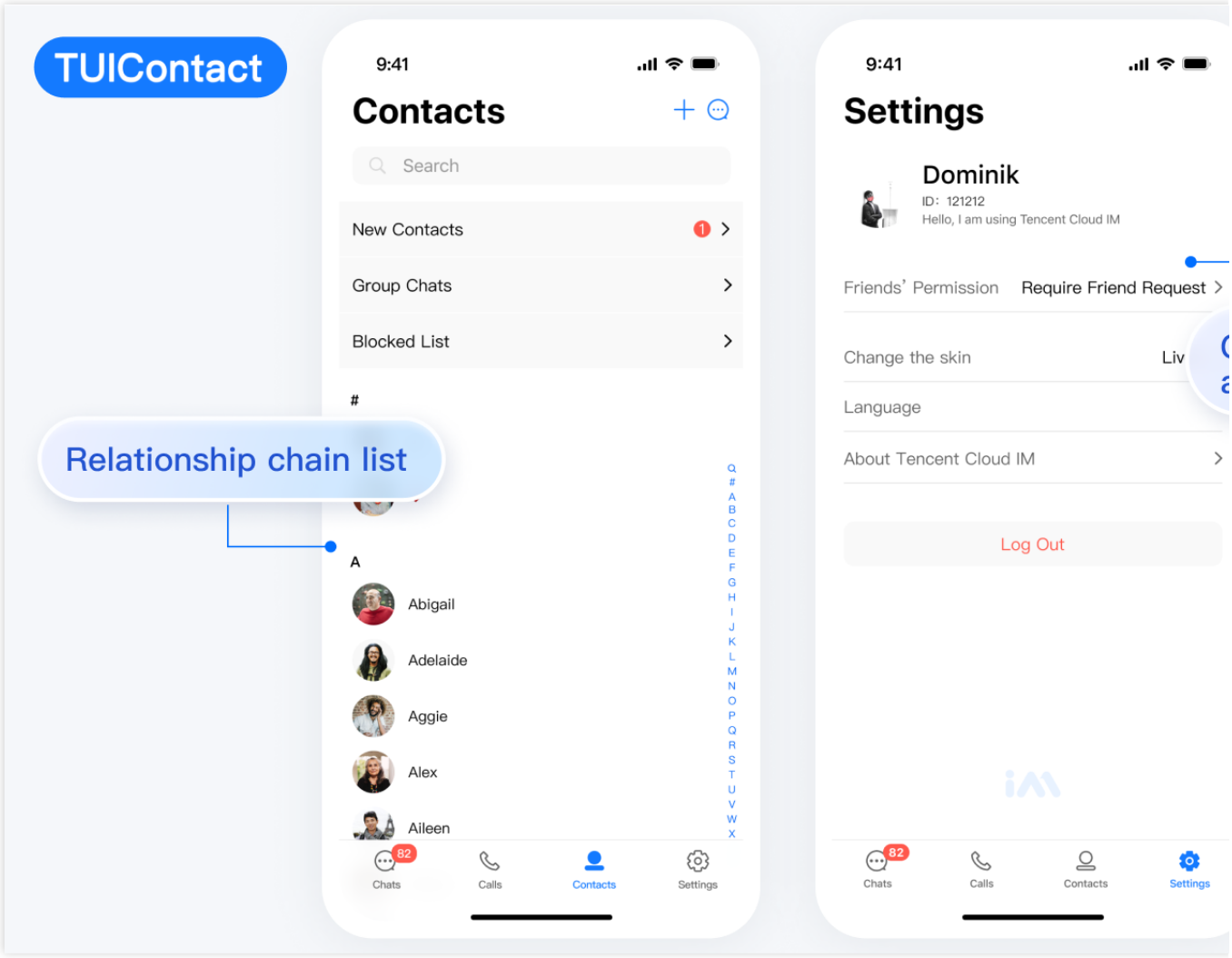
TUIGroup 重点功能介绍

TUIGroup 主要负责群资料、群成员、群组权限的管理。



TUIContact 重点功能介绍

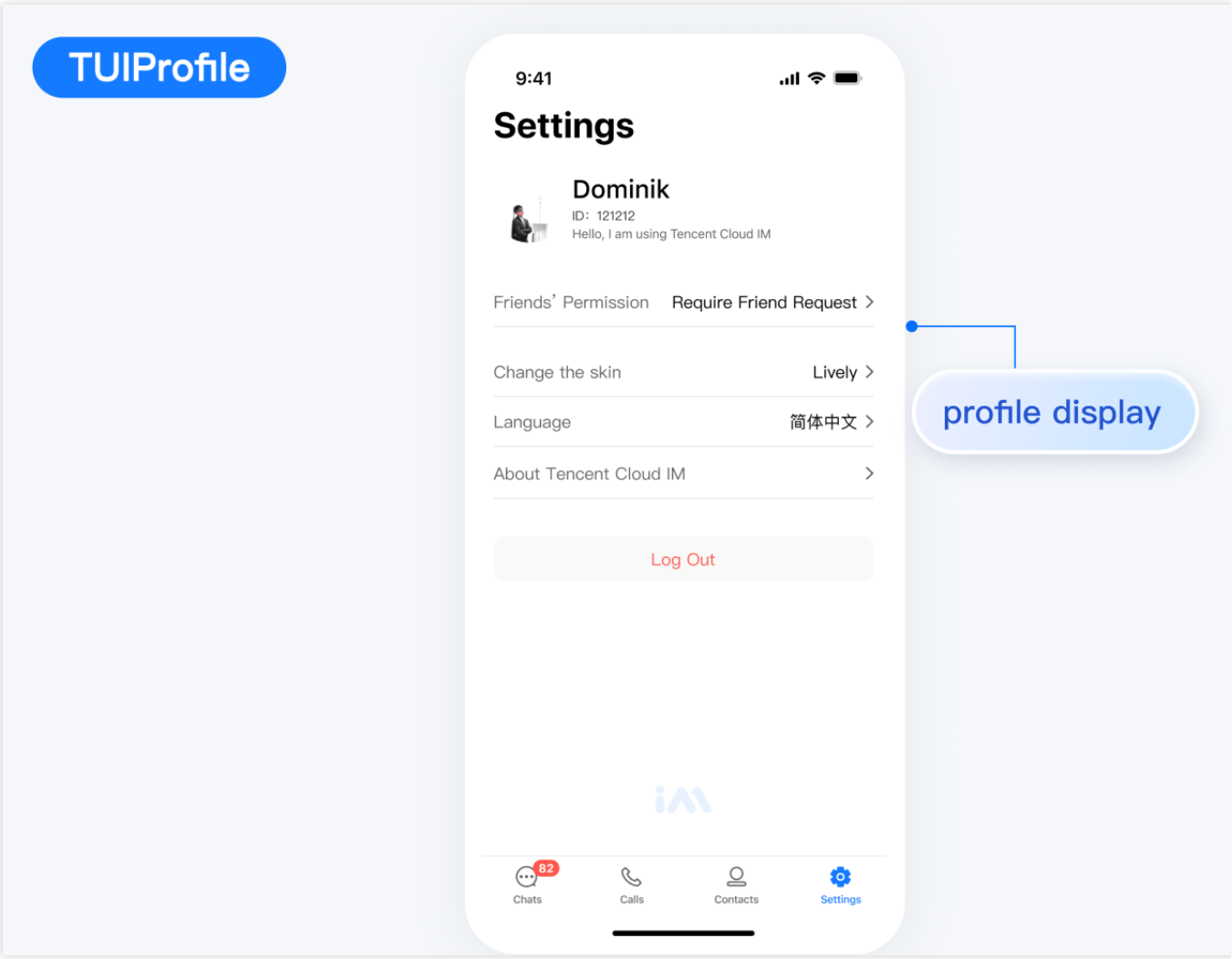
TUIContact 主要负责联系人的展示、权限设置等。



TUIProfile 重点功能介绍

TUIProfile 主要负责用户资料的管理。

版权所有：腾讯云计算（北京）有限责任公司

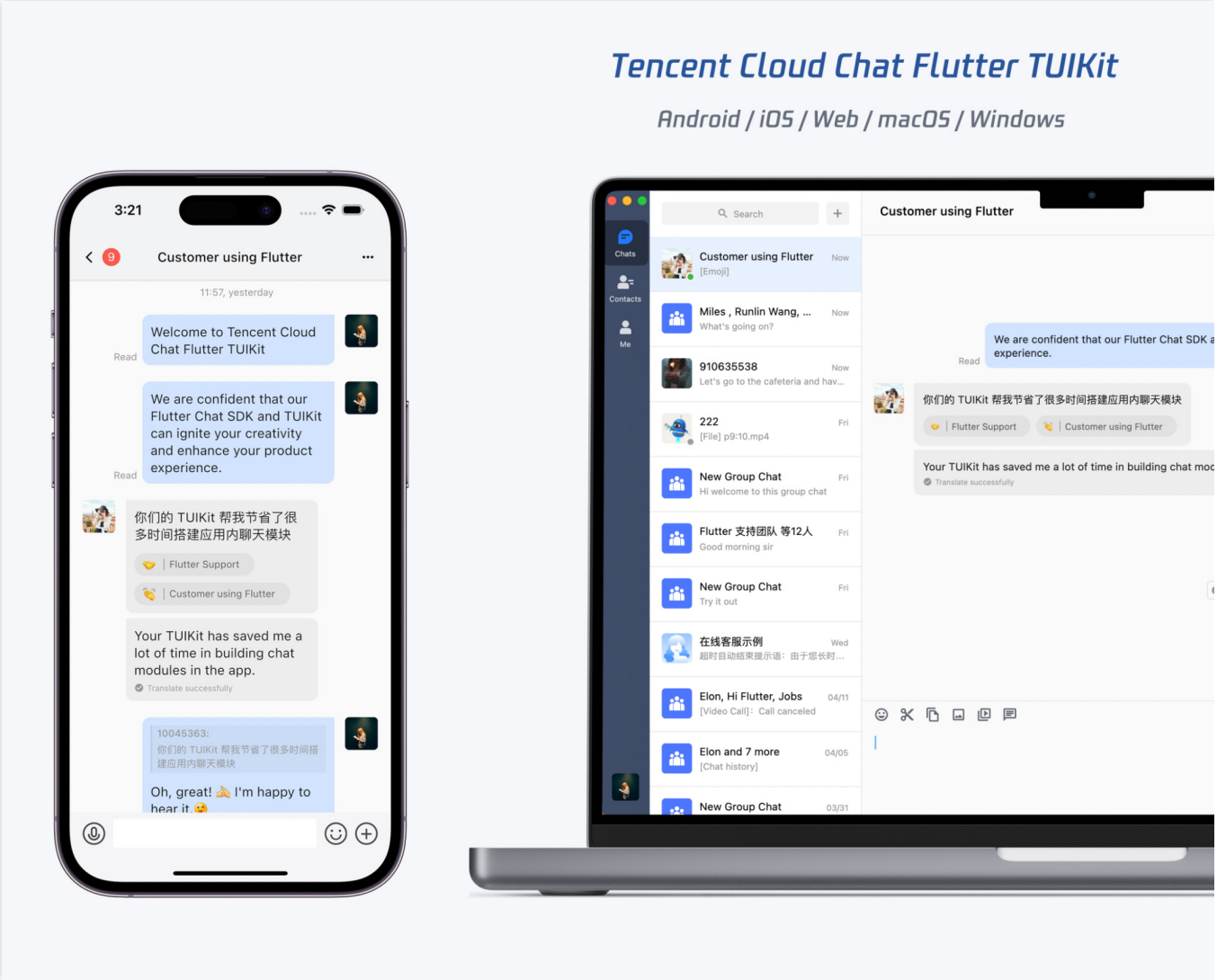

TUIProfile
9:41
Settings
Dominik
ID: 121212
Hello, I am using Tencent Cloud IM
Friends' Permission
Require Friend Request
Change the skin
Lively
Language
简体中文
About Tencent Cloud IM
Log Out
82
Chats
Calls
Contacts
Settings
profile display

Flutter

最近更新时间：2023-06-14 15:52:52

TUIKit 介绍

TUIKit 是基于 腾讯云 IM SDK 开发的一款即时通讯 UI 组件库，包括会话、聊天、搜索、关系链、群组、音视频通话等能力。通过 TUIKit，可快速通过**一套代码一次开发**，集成包含 UI 界面的**移动及桌面全平台即时通讯应用**。
TUIKit 简化了基于 腾讯云 IM SDK 的应用开发过程。它不仅能助您快速实现 UI 功能，也支持调用 腾讯云 IM SDK 相应的接口实现即时通讯业务逻辑和数据处理。因此，您在使用 TUIKit 时仅需关注自身业务或个性化扩展。



支持平台

说明：

TUIKit 拥有自适应 UI 界面，可同时用于 **移动端 及 桌面端。**并支持以下所有平台，针对不同平台有不同的特色能力。

方便您**一套代码开发全平台应用**。

iOS / Android / Web (二维码) / macOS / Windows / 混合开发（将 Flutter IM SDK 添加至现有原生应用）

单击跳转至对应平台 Demo 下载体验。以上各端 Demo 均由**同一项目代码**引入 TUIKit 开发编译打包而成。

TUIKit 主要功能介绍

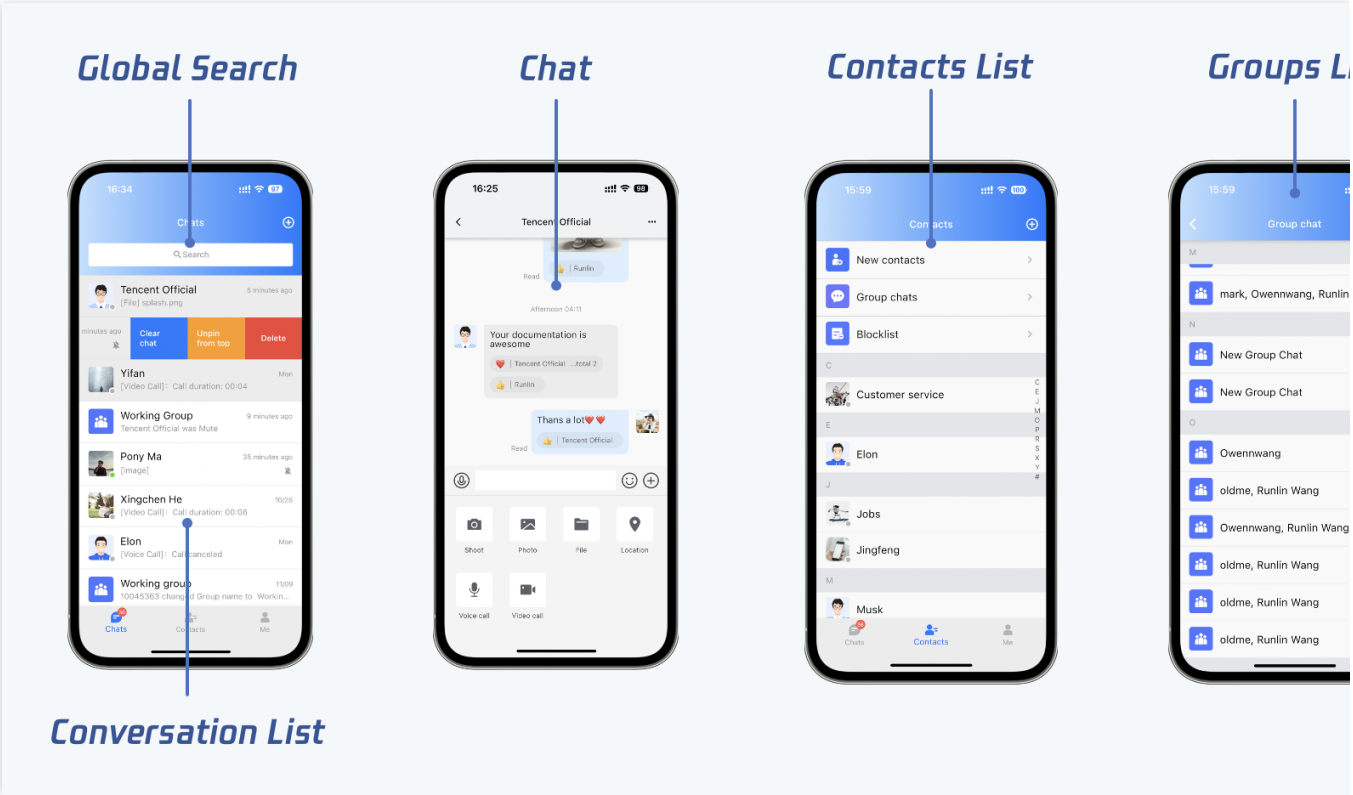
TUIKit 按照功能主要分为 **聊天、会话列表、关系链管理、用户或群组资料、搜索、音视频通话** 等多个类型的 UI 组件，每个类型的 UI 组件负责承载不同的功能模块。

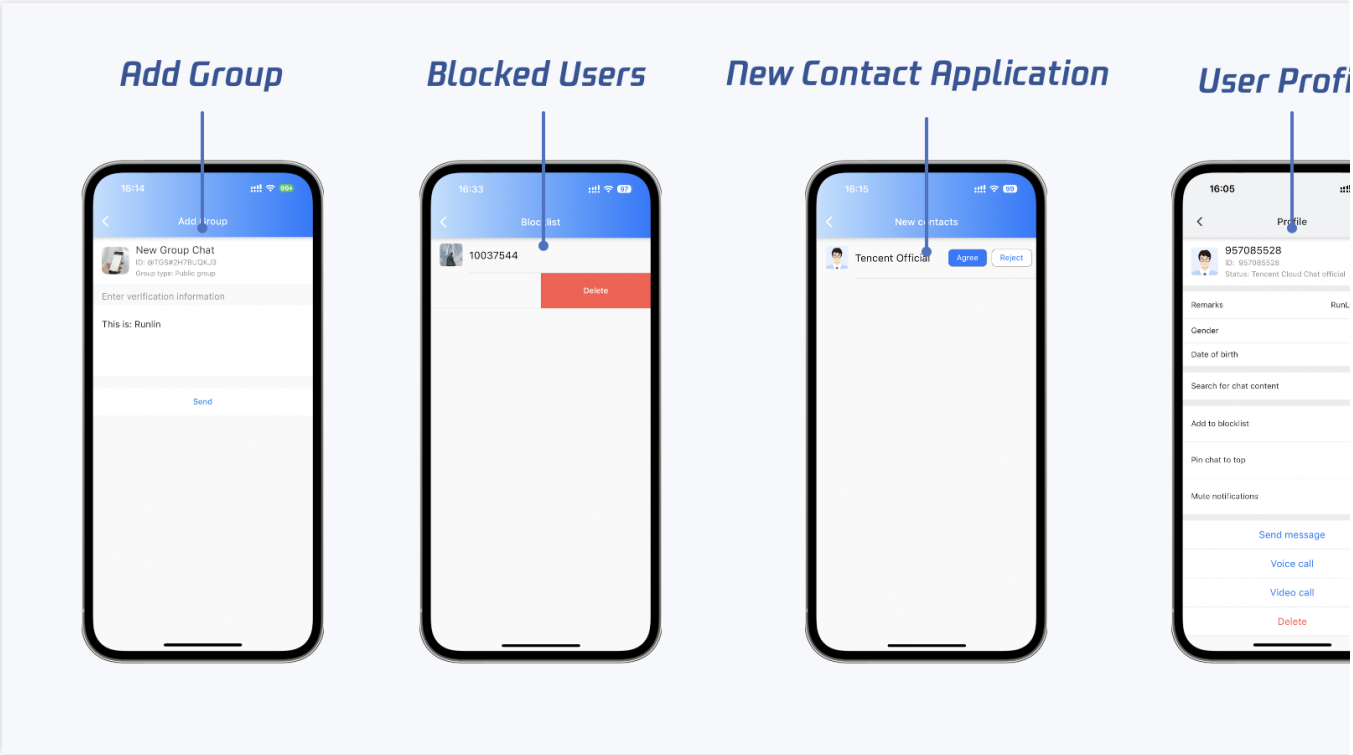
每个组件在移动端和桌面端的用法一致，由 TUIKit 内部做平台自适应。

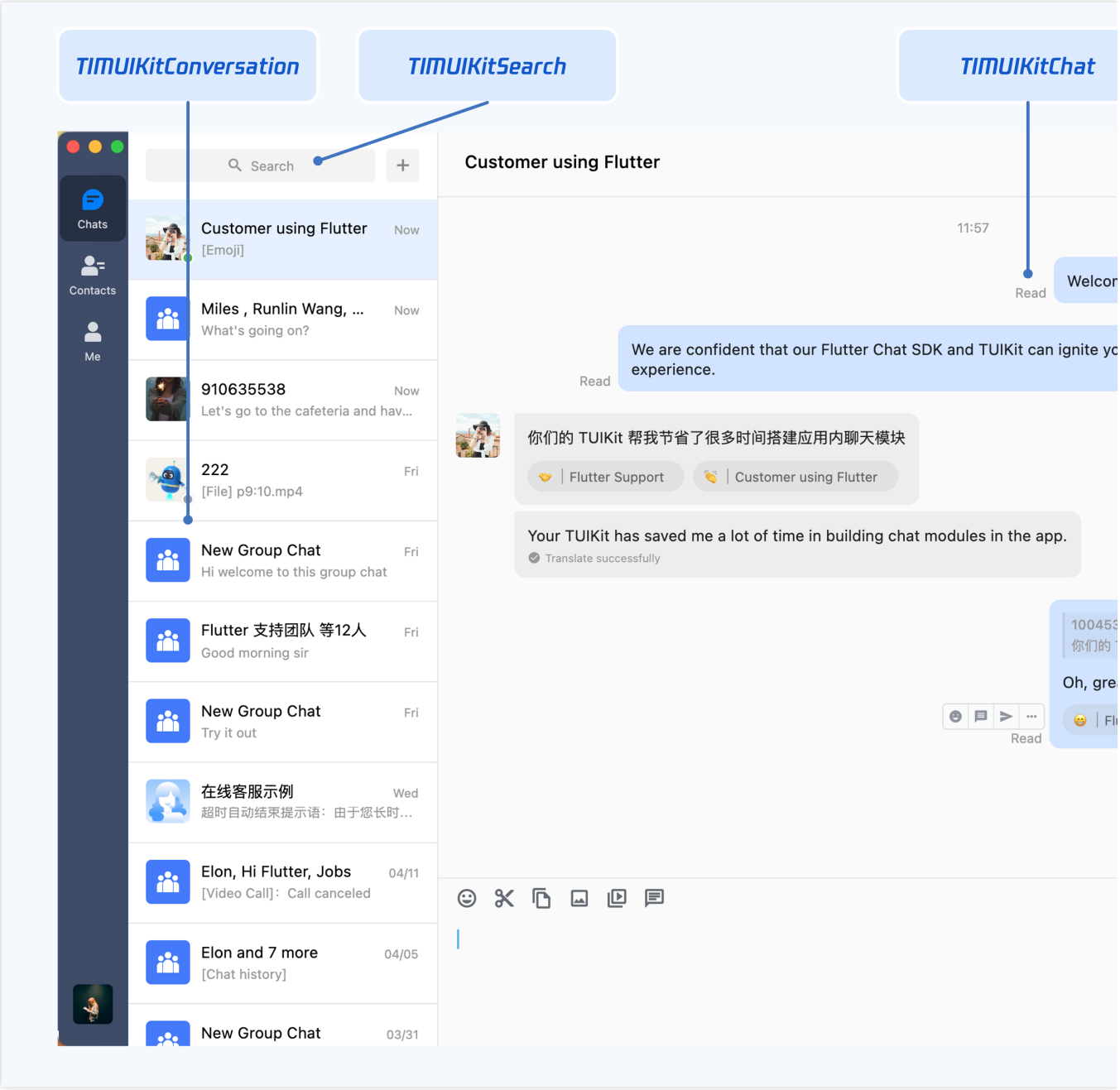
界面效果如下图所示：

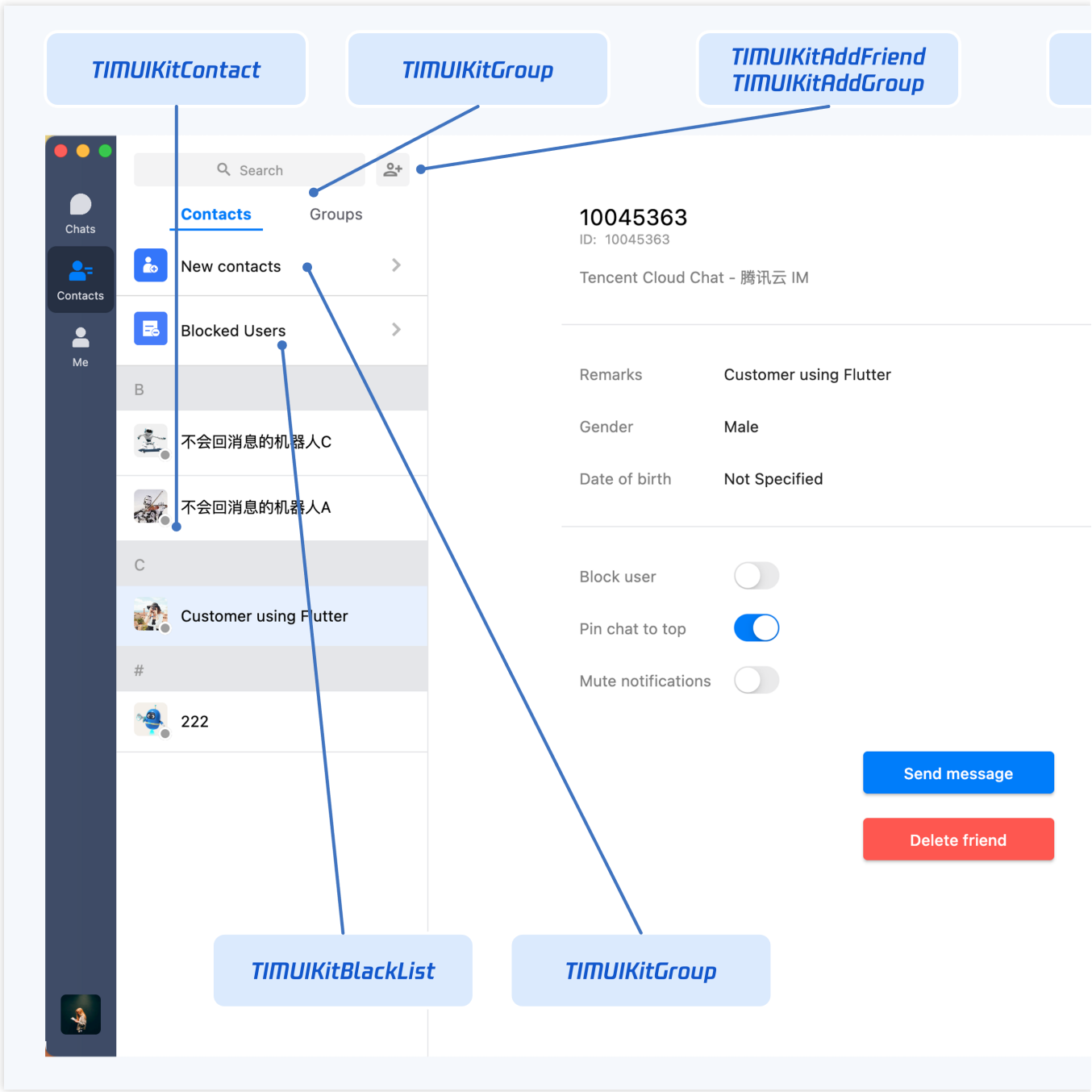
移动端

桌面端




Add Group
Blocked Users
New Contact Application
User Profi
New Group Chat
Enter verification information
This is: Runlin
Send
10037544
Delete
Tencent Official
Agree
Reject
957085528
Remarks
Gender
Date of birth
Search for chat content
Add to blocklist
Pin chat to top
Mute notifications
Send message
Voice call
Video call
Delete


TIMUIKitConversation
TIMUIKitSearch
TIMUIKitChat
Search
Chats
Contacts
Me
Customer using Flutter
[Emoji]
Now
Miles , Runlin Wang, ...
What's going on?
910635538
Let's go to the cafeteria and hav...
222
[File] p9:10.mp4
Fri
New Group Chat
Hi welcome to this group chat
Flutter 支持团队 等12人
Good morning sir
New Group Chat
Try it out
在线客服示例
超时自动结束提示语：由于您长时...
Wed
Elon, Hi Flutter, Jobs
[Video Call]：Call canceled
04/11
Elon and 7 more
[Chat history]
04/05
New Group Chat
03/31
Customer using Flutter
11:57
Read
We are confident that our Flutter Chat SDK and TUIKit can ignite yo
experience.
你们的 TUIKit 帮我节省了很多时间搭建应用内聊天模块
Flutter Support
Customer using Flutter
Your TUIKit has saved me a lot of time in building chat modules in the app.
Translate successfully
Oh, gre
Read



消息收发聊天组件

TIMUIKitChat 主要负责消息界面的展示。您可以利用它直接发送不同类型的消息，进行消息表情回应 / 回复 / 引用，并查看消息已读回执详情等操作。

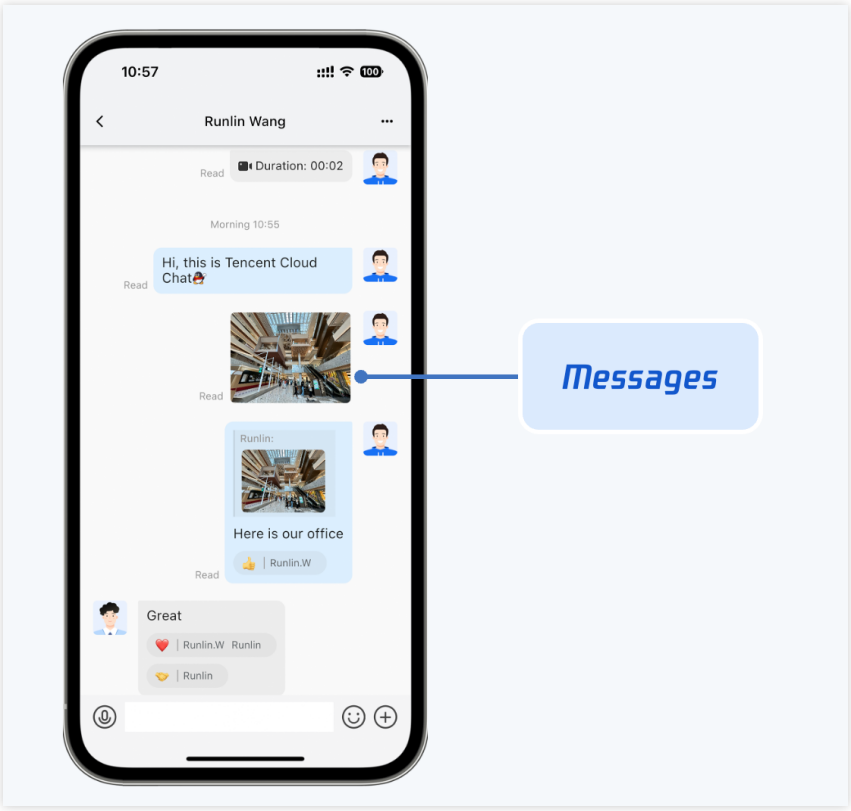
在桌面端，还包括 文件拖入发送 / 截图 / 粘贴图片发送 / 在目录中打开文件 等桌面端特色能力。

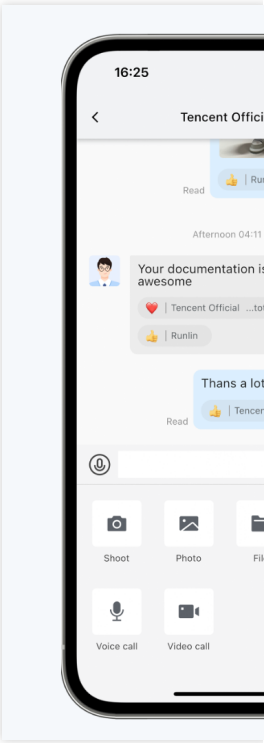
界面效果如下图所示：

移动端

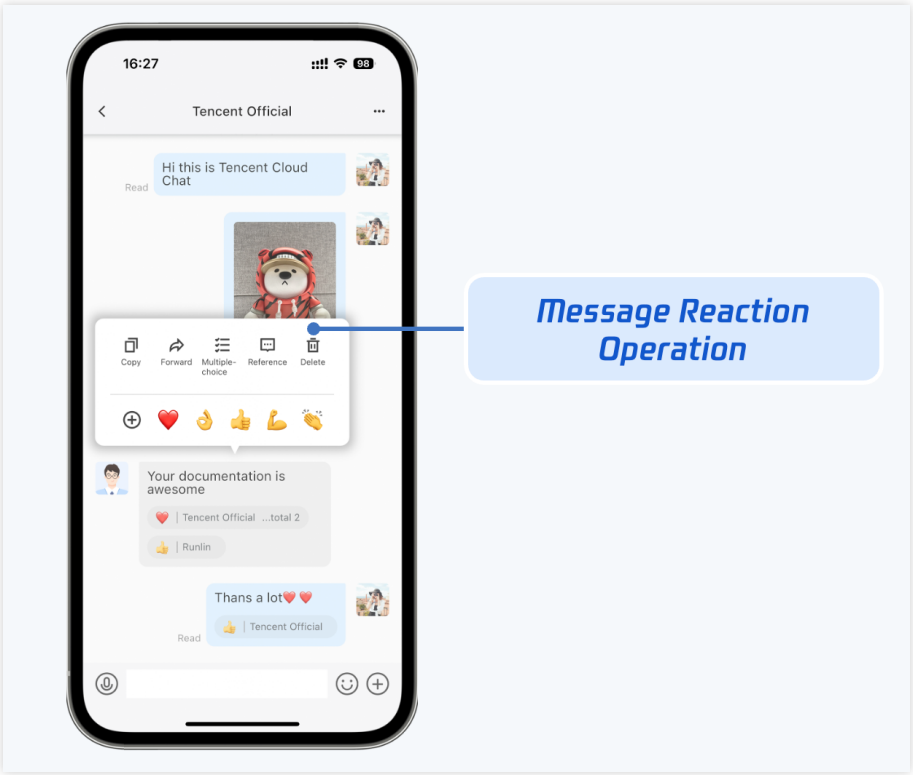
桌面端

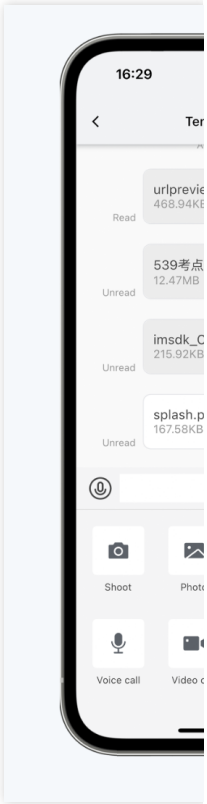
消息界面	发送多种类型消息



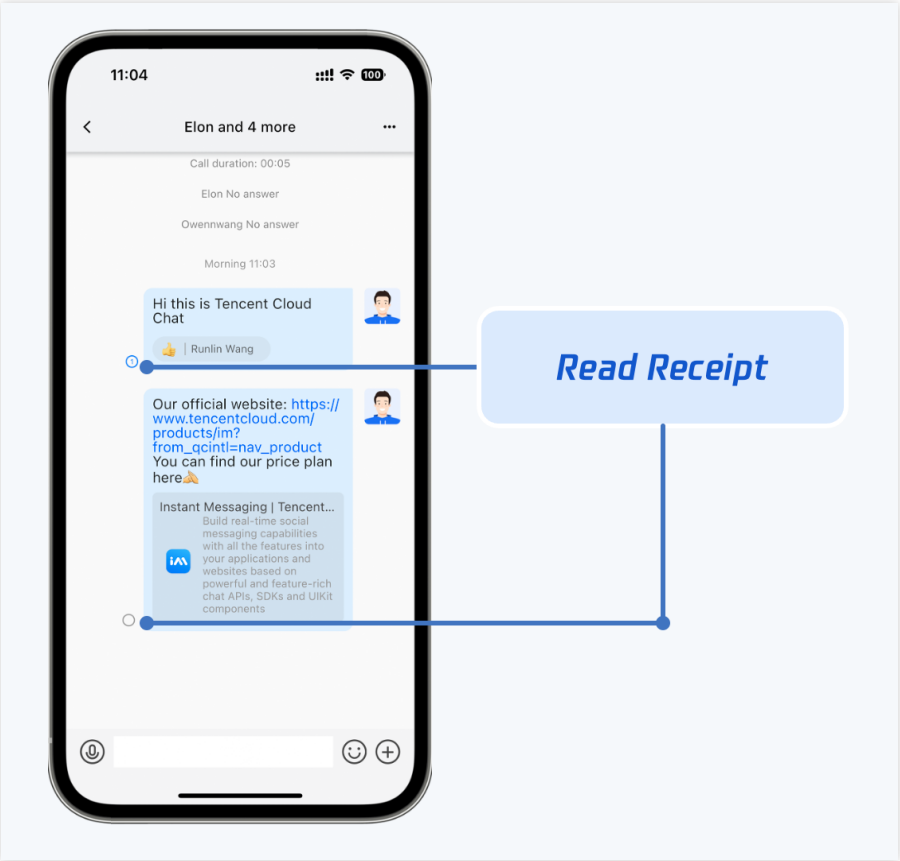


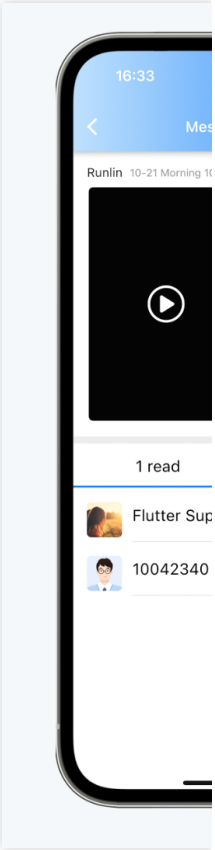
表情回应/回复/引用	文件自动匹配icon



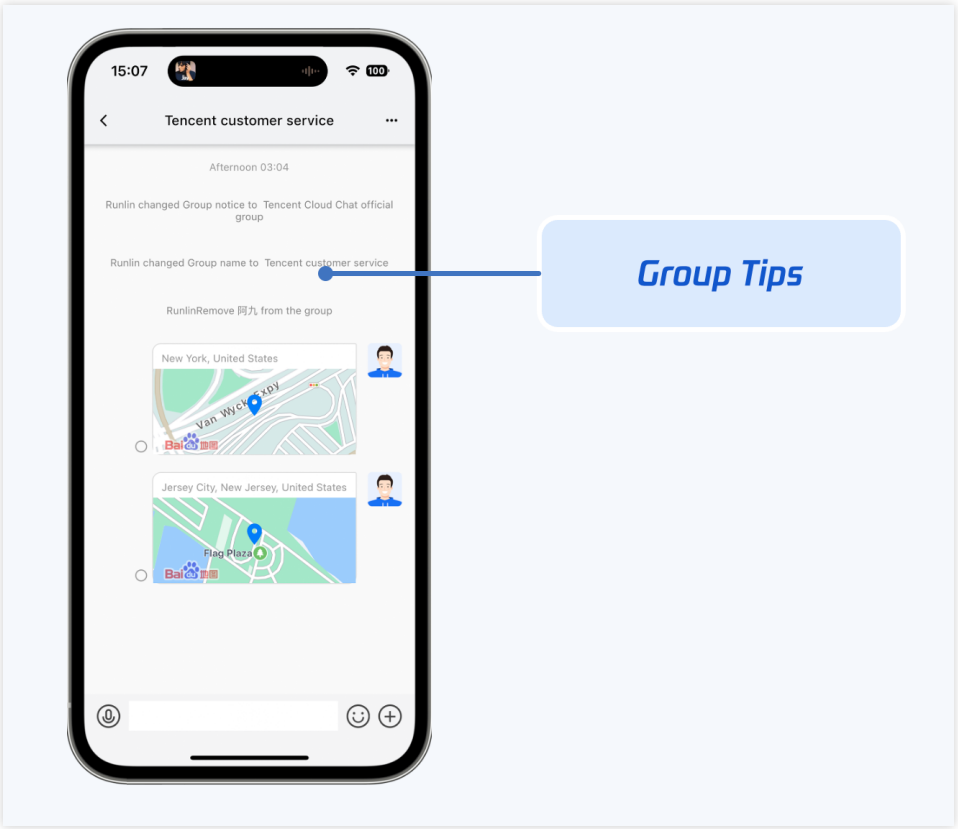


消息已读回执	已读回执详情

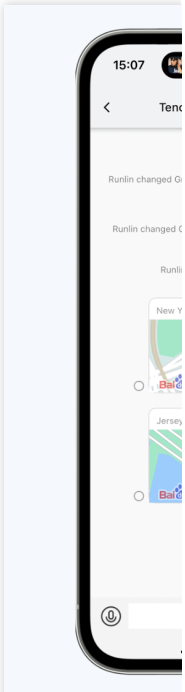


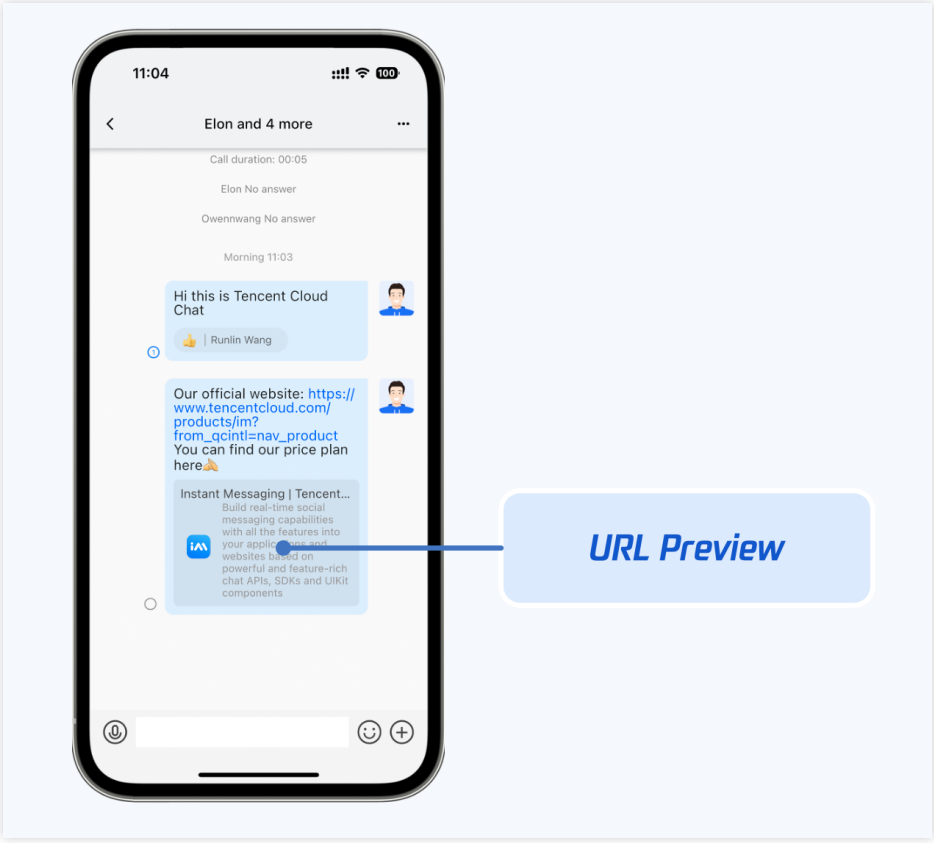


群 tips 消息	入群申请审批



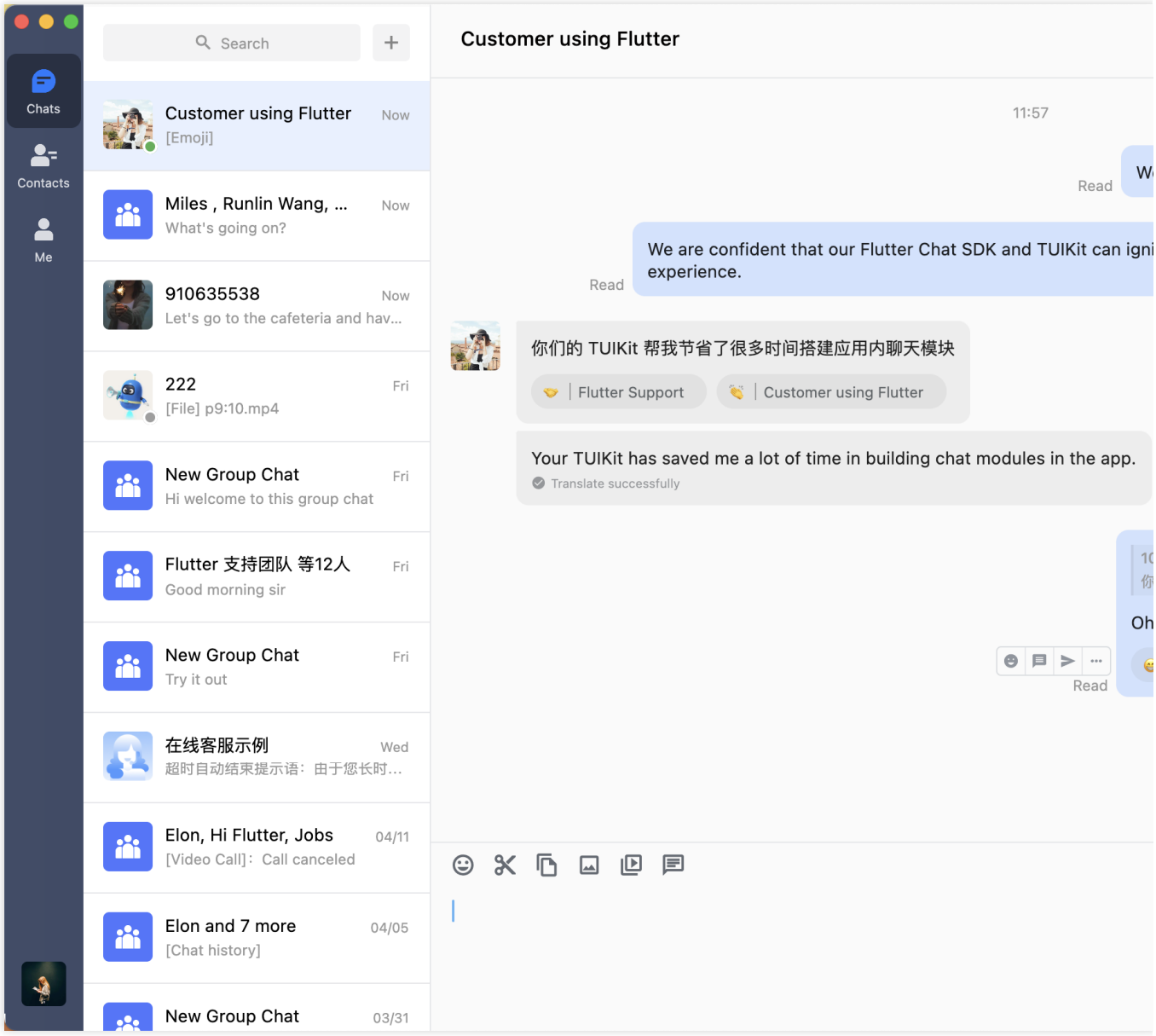
链接解析预览	地理位置消息





消息界面整体如图所示，包含了桌面端交互常见的消息收发能力。

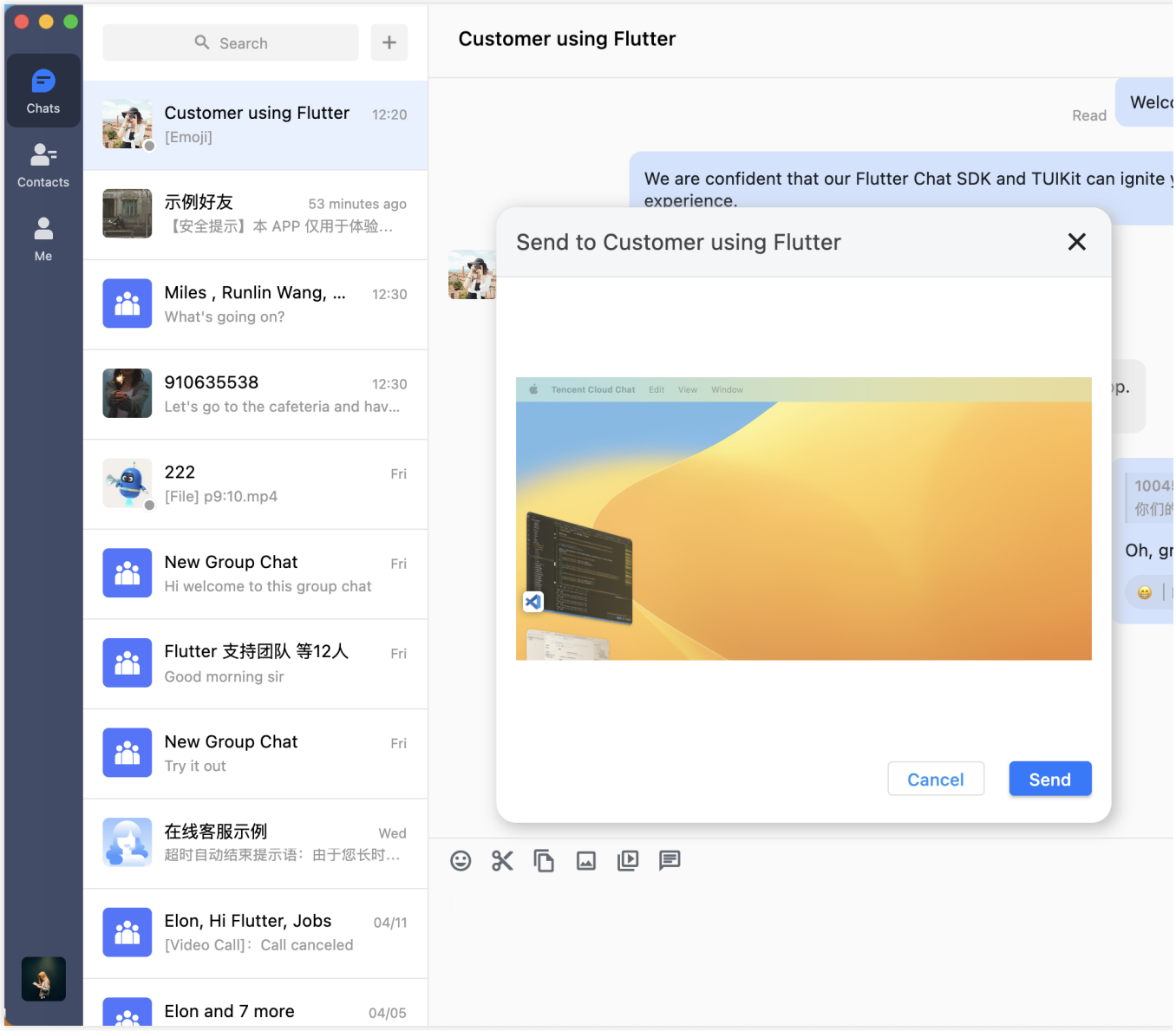
版权所有：腾讯云计算（北京）有限责任公司



除 **移动端Tab** 展示的功能外，桌面端还包括一些额外的功能，如下方所述。

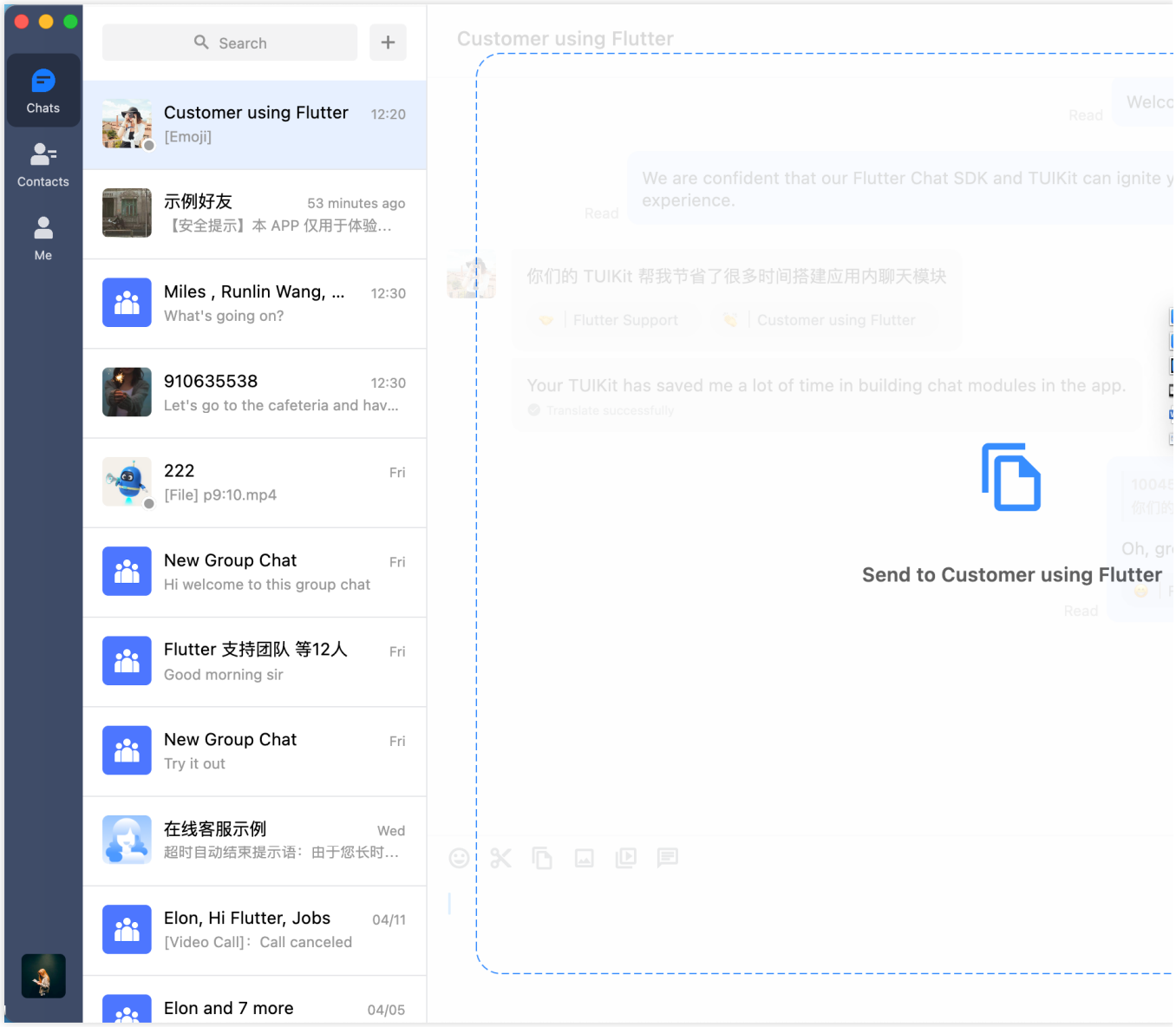
截图 或 粘贴图片至发送区域 直接发送图片。

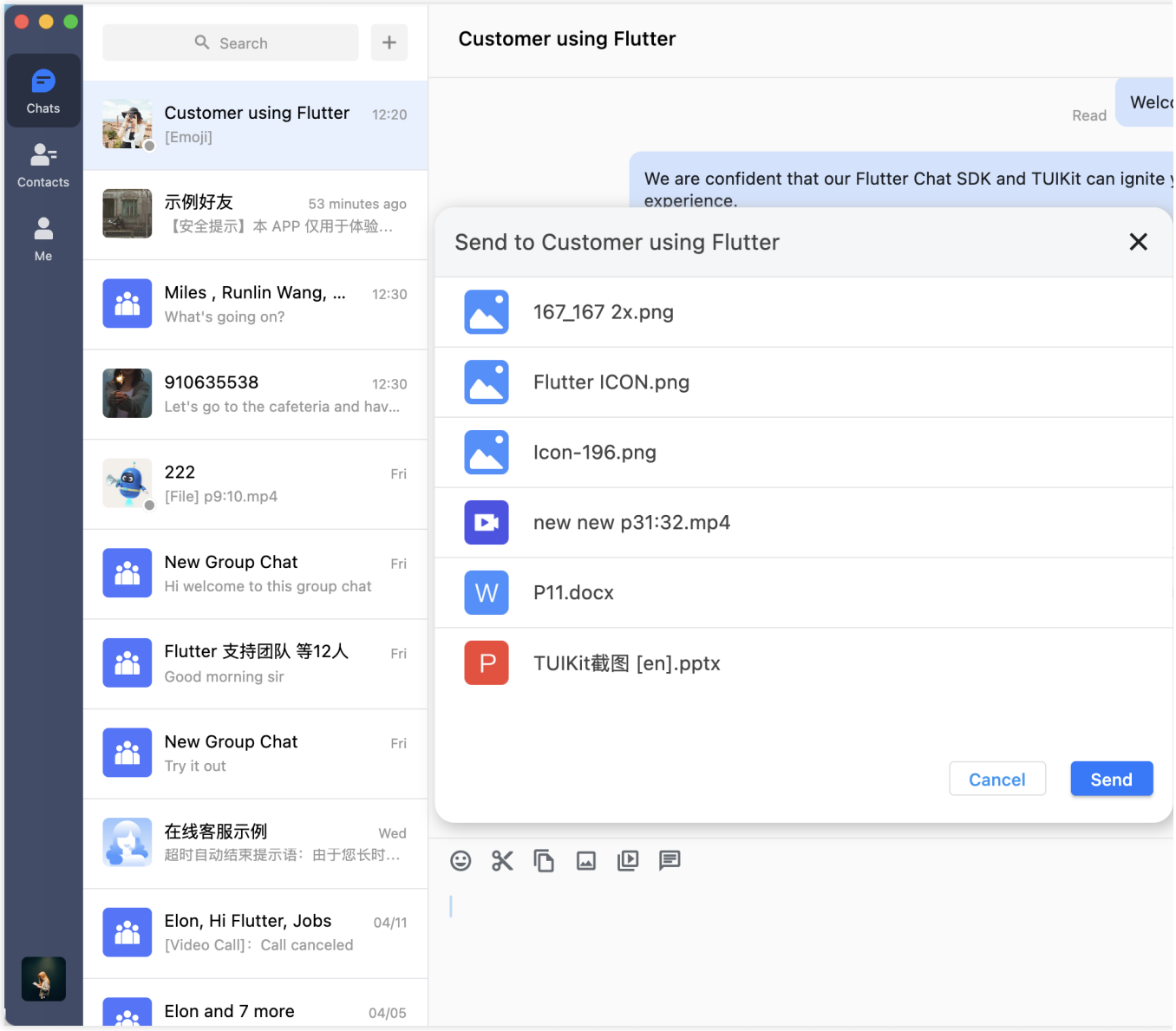
版权所有：腾讯云计算（北京）有限责任公司



拖入多个文件直接发送。

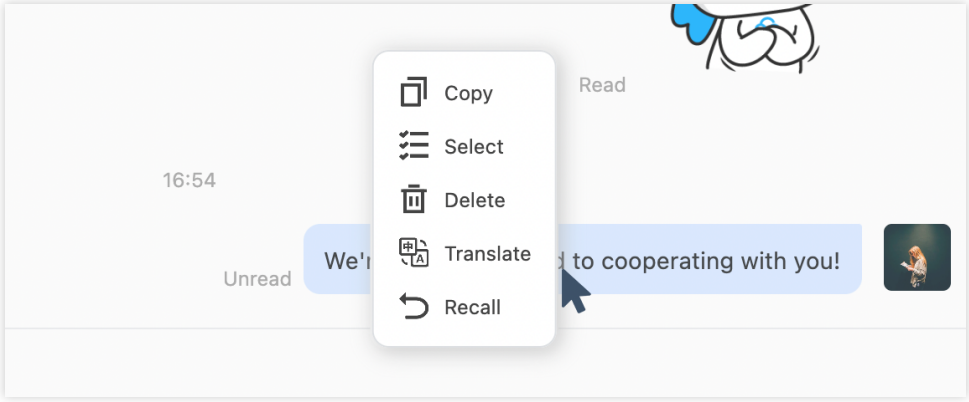
版权所有：腾讯云计算（北京）有限责任公司


Search
Chats
Contacts
Me
Customer using Flutter 12:20
[Emoji]
示例好友 53 minutes ago
【安全提示】本 APP 仅用于体验...
Miles , Runlin Wang, ... 12:30
What's going on?
910635538 12:30
Let's go to the cafeteria and hav...
222 Fri
[File] p9:10.mp4
New Group Chat Fri
Hi welcome to this group chat
Flutter 支持团队 等12人 Fri
Good morning sir
New Group Chat Fri
Try it out
在线客服示例 Wed
超时自动结束提示语：由于您长时...
Elon, Hi Flutter, Jobs 04/11
[Video Call]：Call canceled
Elon and 7 more 04/05
Customer using Flutter
We are confident that our Flutter Chat SDK and TUIKit can ignite y
experience.
你们的 TUIKit 帮我节省了很多时间搭建应用内聊天模块
Flutter Support
Customer using Flutter
Your TUIKit has saved me a lot of time in building chat modules in the app.
Translate successfully
Send to Customer using Flutter

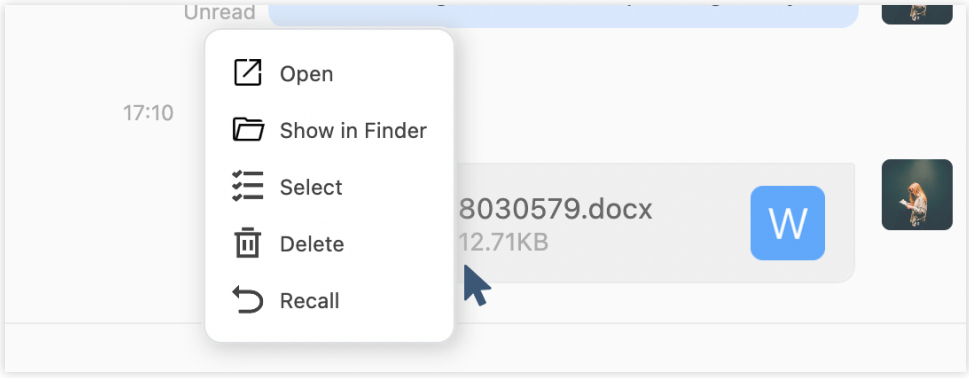


消息上，鼠标 Hover 菜单。包含 表情回应 / 消息回复 / 转发 及更多能力。

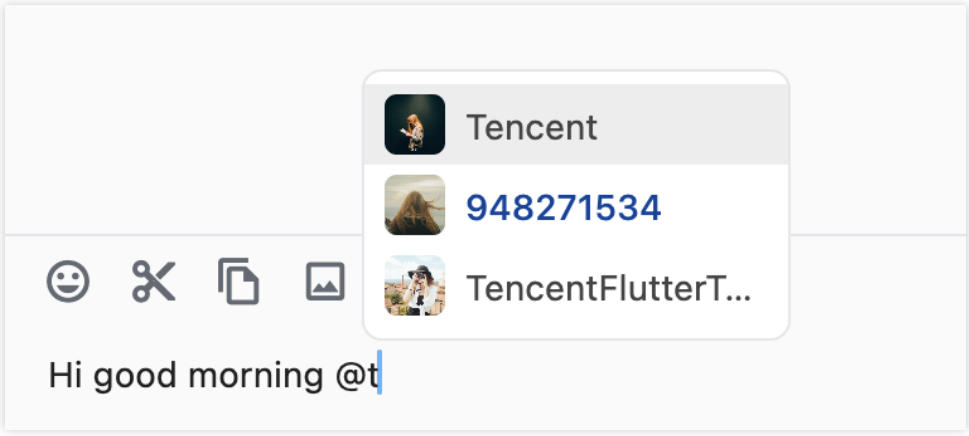
消息上，鼠标右键菜单。包含更多消息操作，如 复制 / 多选 / 删除 / 翻译 / 撤回 等。



文件消息支持在右键菜单中 直接打开 或 打开所在目录。也可点击消息 item 直接打开。

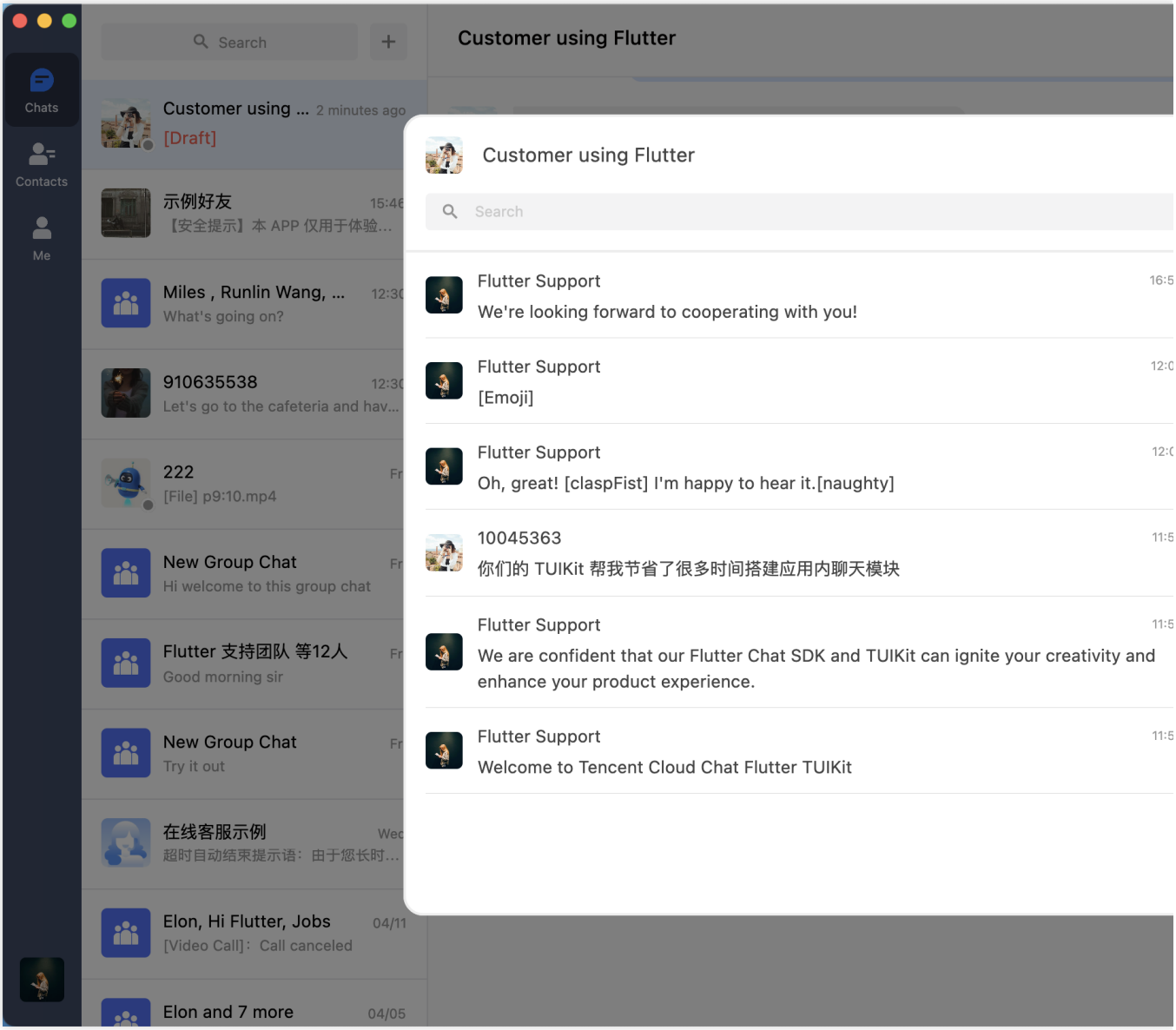


群聊中 @ 其他群成员。在群成员选择面板中，可渐进式输入搜索并选择群成员。被 @ 的群成员可收到特别提醒。



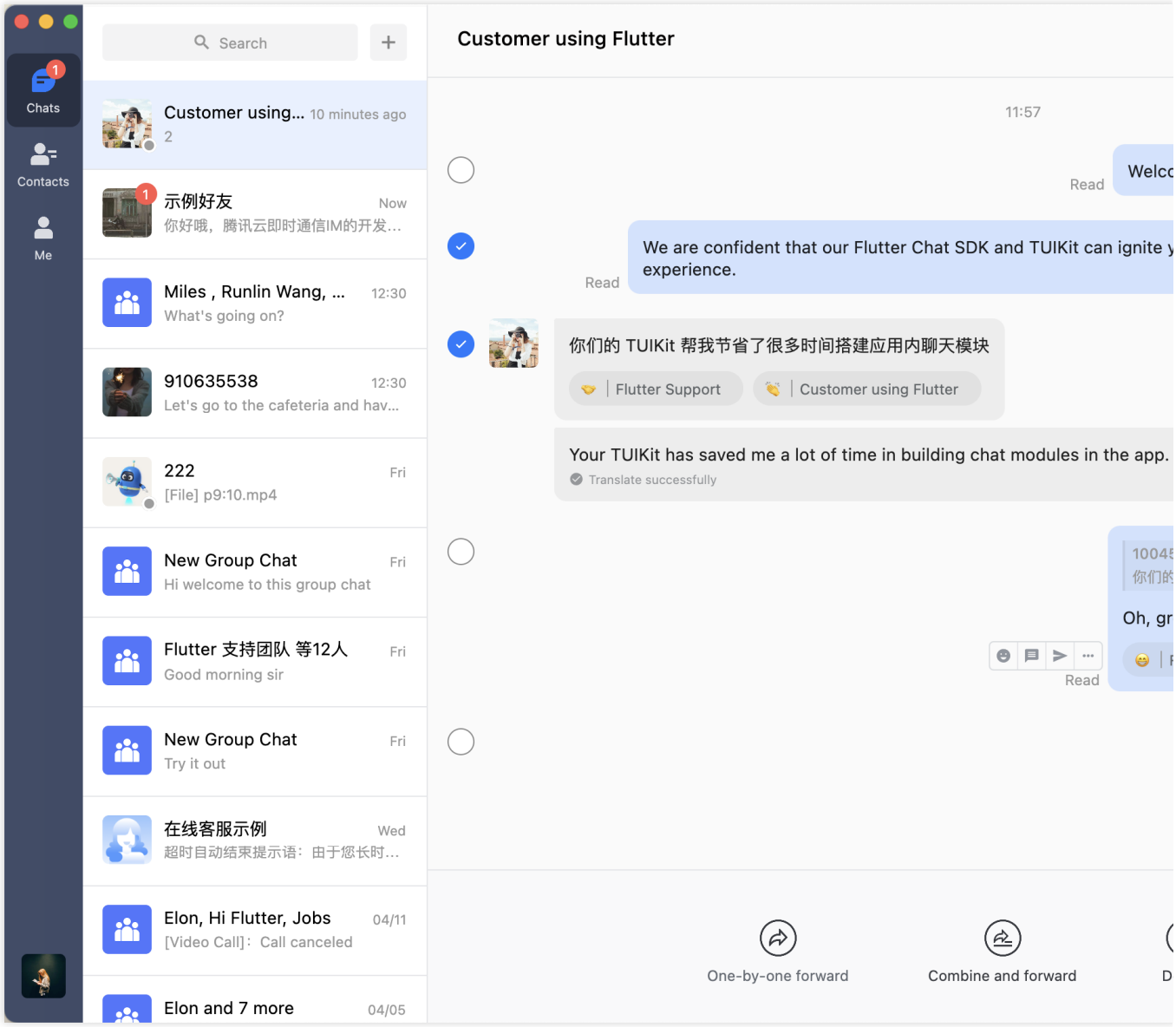
历史消息盒子。支持关键词搜索。

版权所有：腾讯云计算（北京）有限责任公司



消息多选。

版权所有：腾讯云计算（北京）有限责任公司



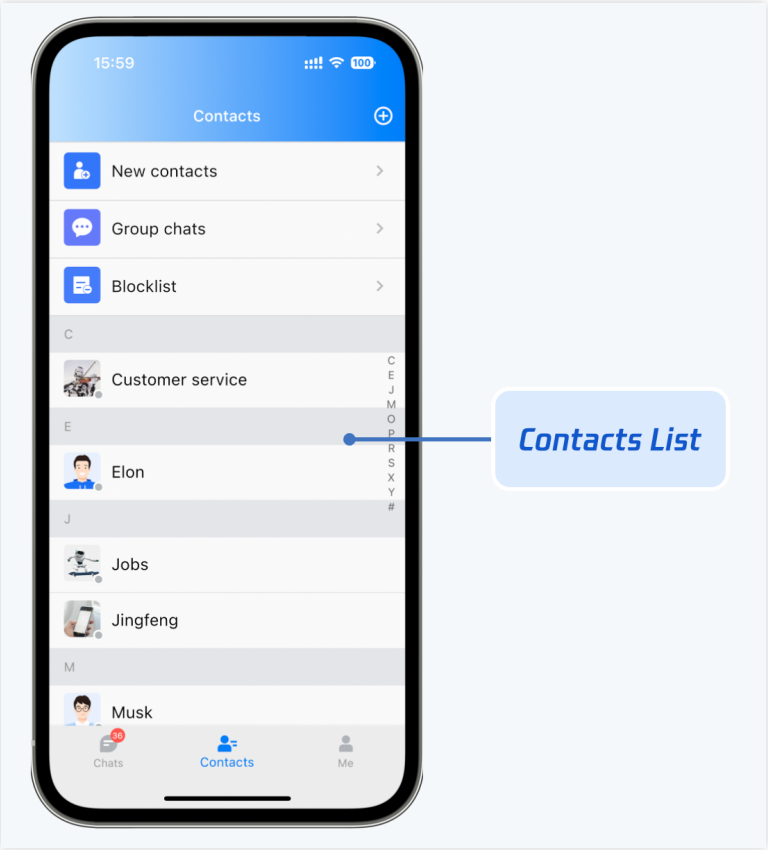
关系链系列组件

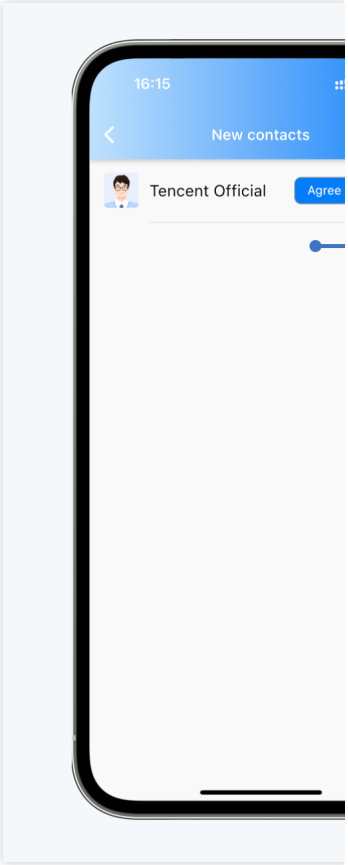
关系链主要负责当前用户的联系人、群聊、黑名单的信息展示。界面效果如下图所示：

移动端

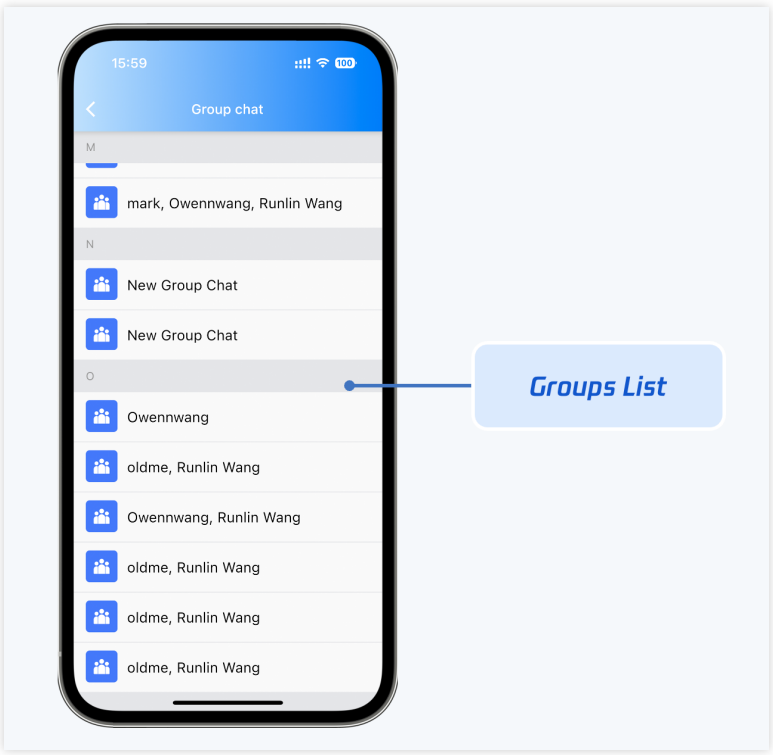
桌面端

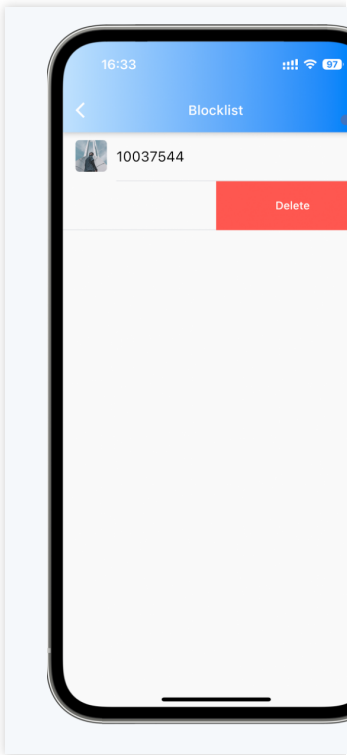
联系人列表(TIMUIKitContact)	好友申请列表(TIMUIKitNewContac



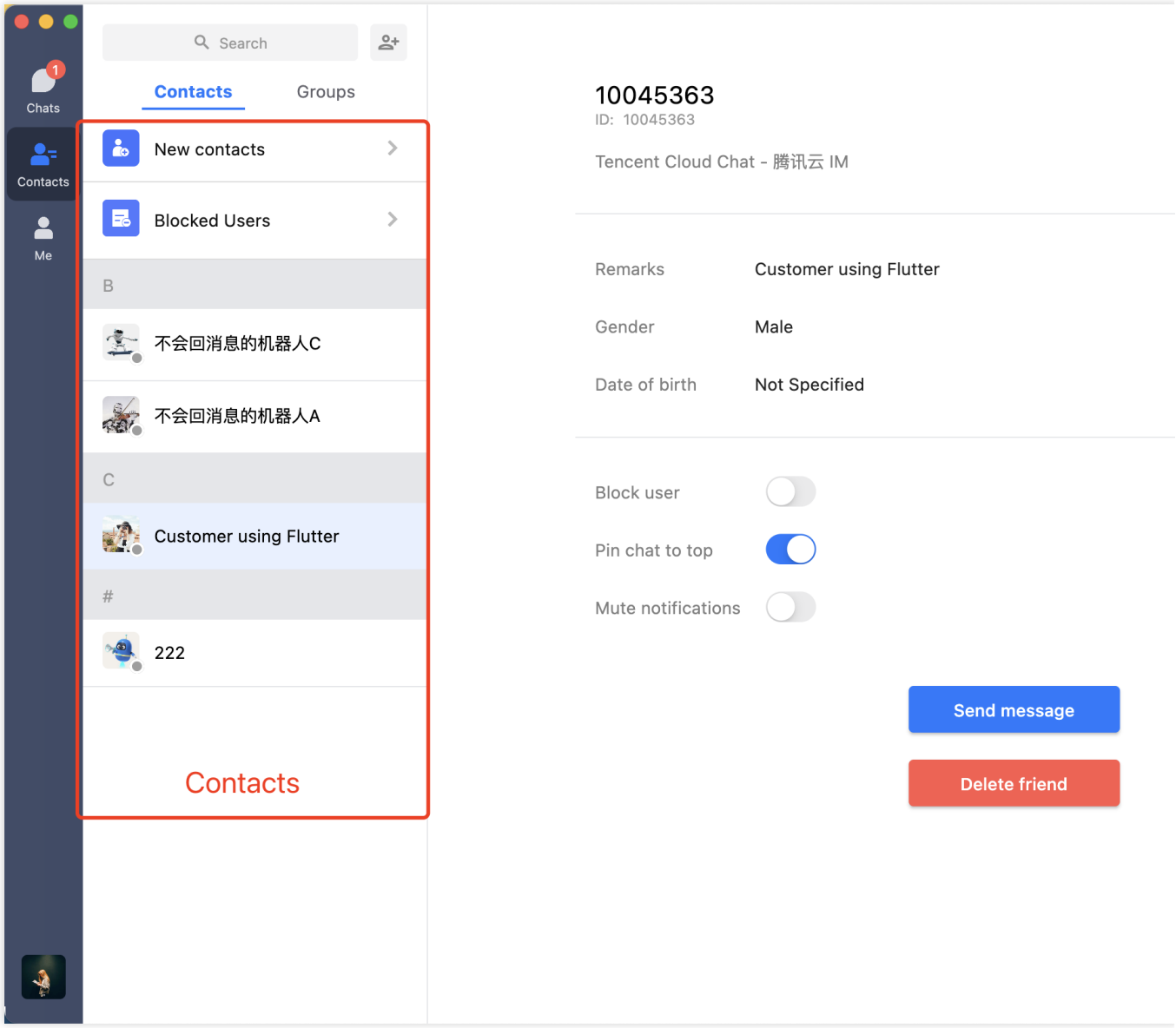


参与的群聊列表(TIMUIKitGroup)	黑名单列表(TIMUIKitBlackList)

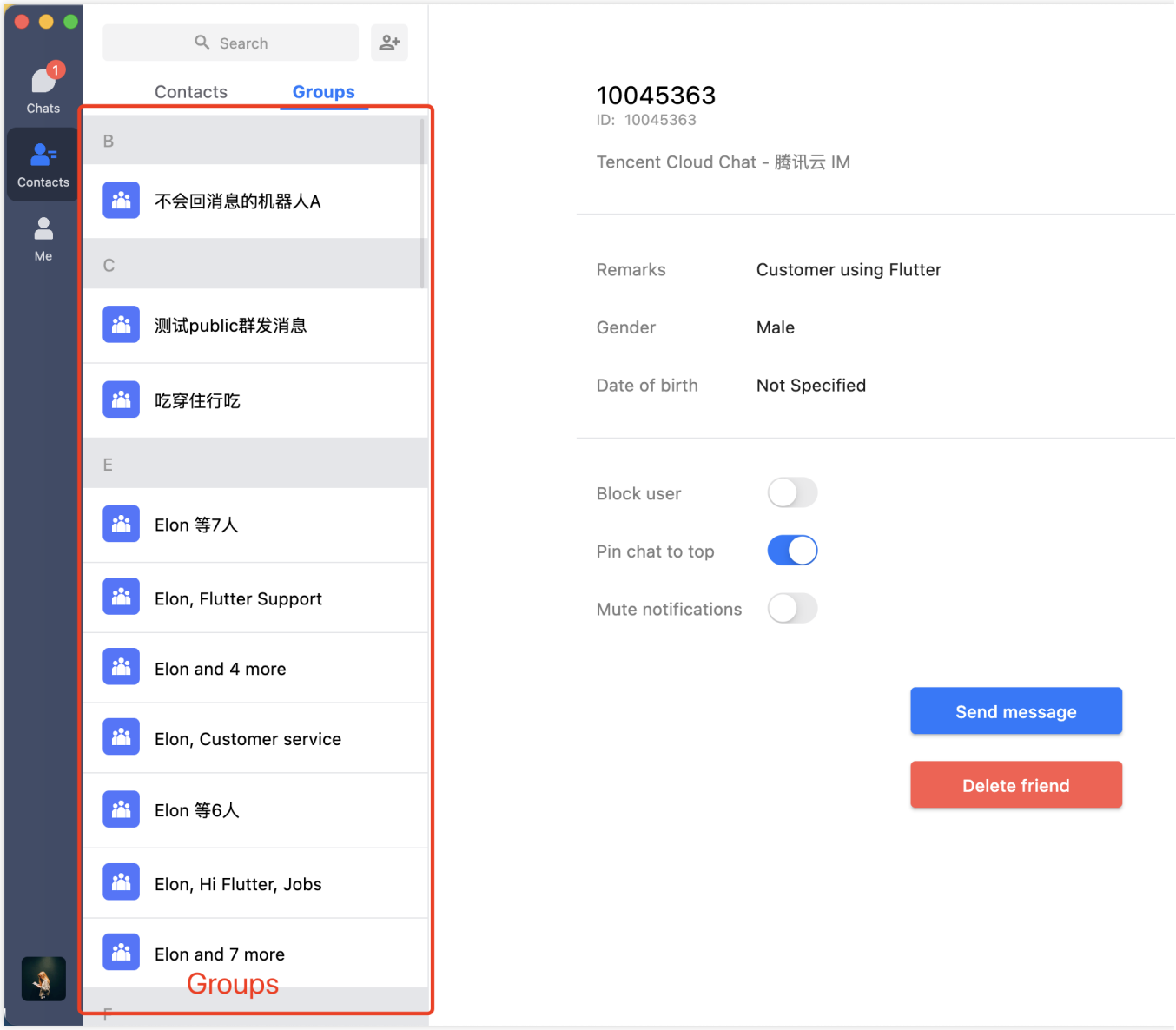




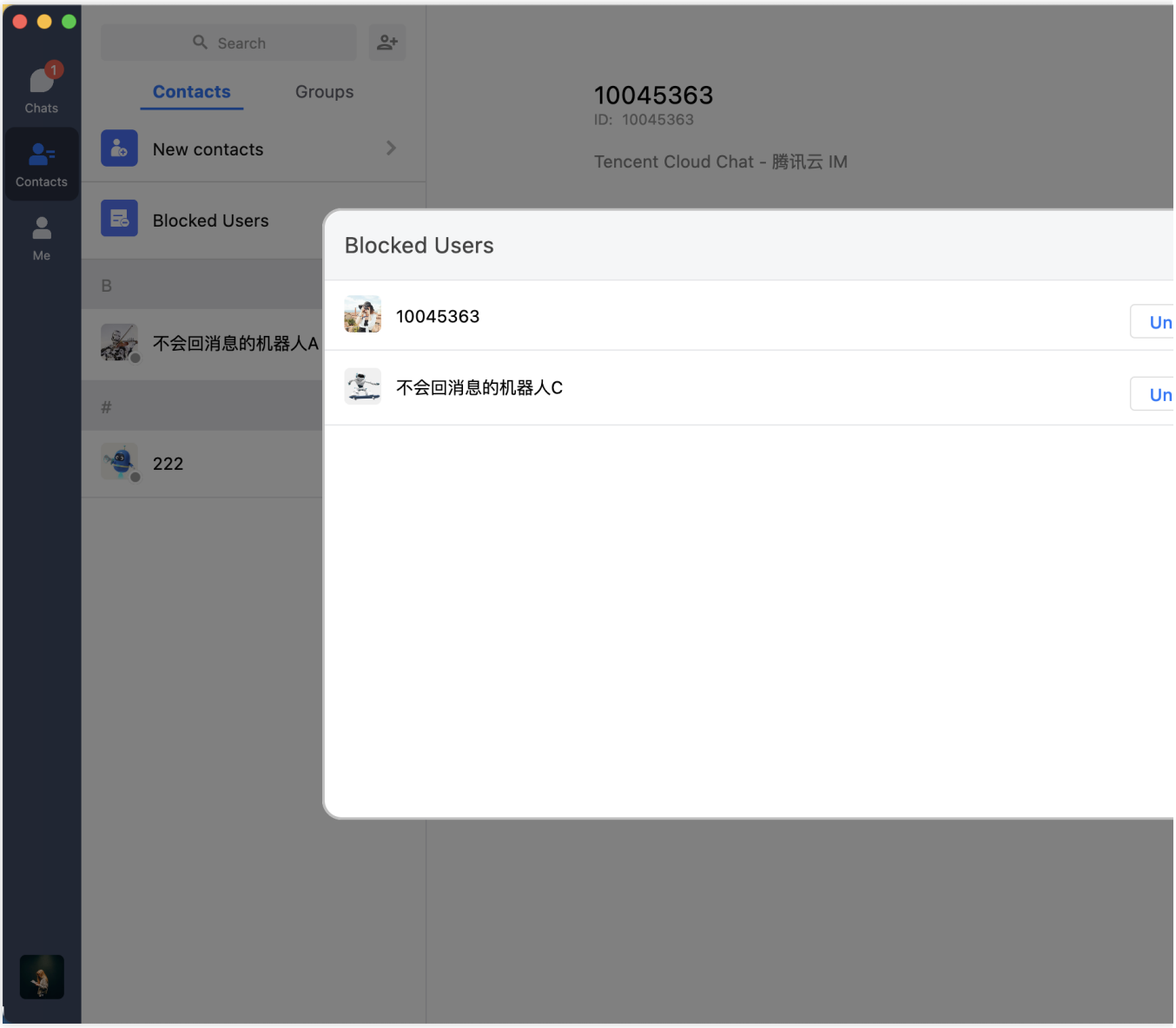
联系人列表 - TIMTUIKitContact



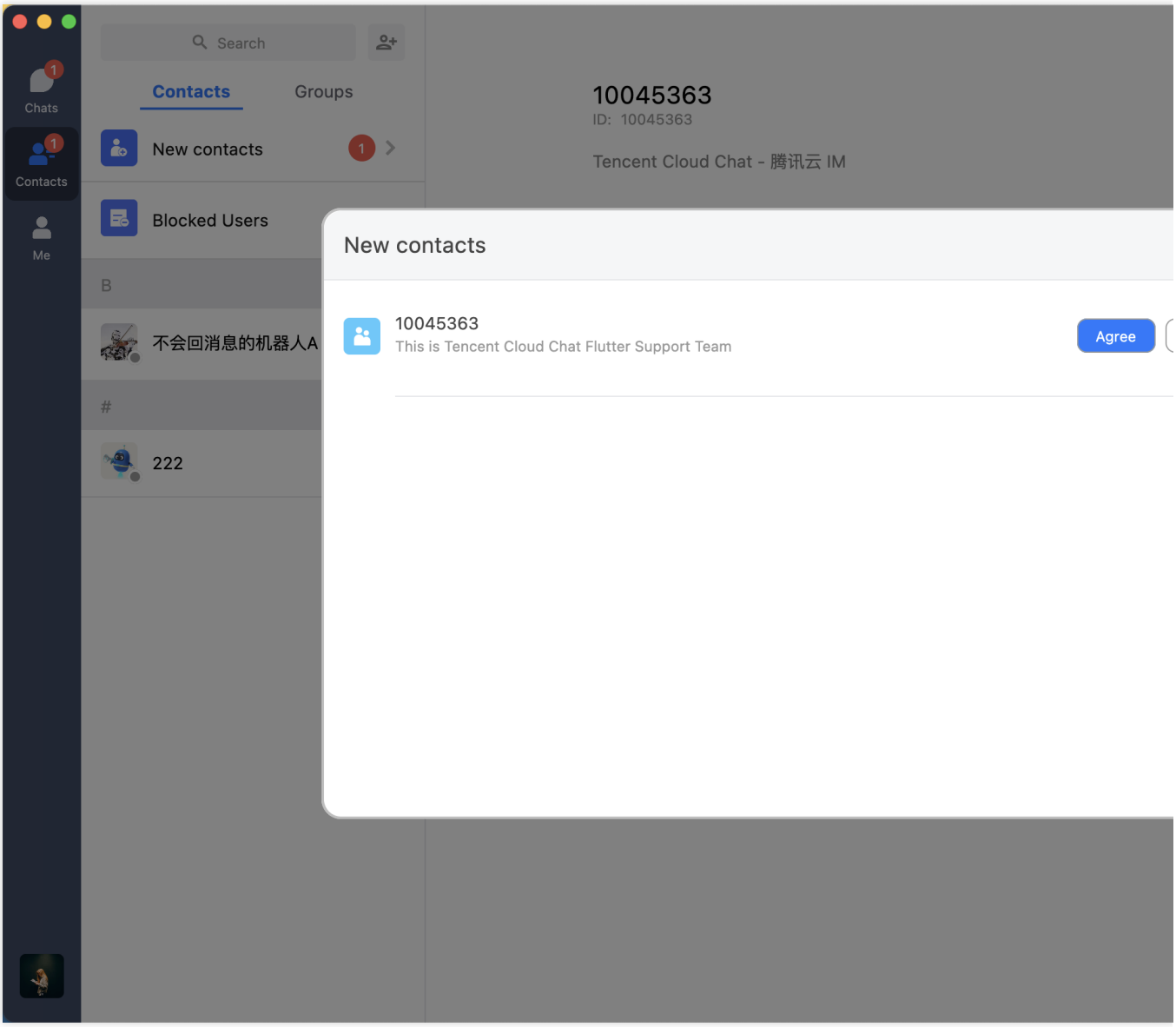
群组列表 - TIMUIKitGroup



黑名单列表 - TIMUIKitBlackList



好友申请列表 - TIMUIKitNewContact

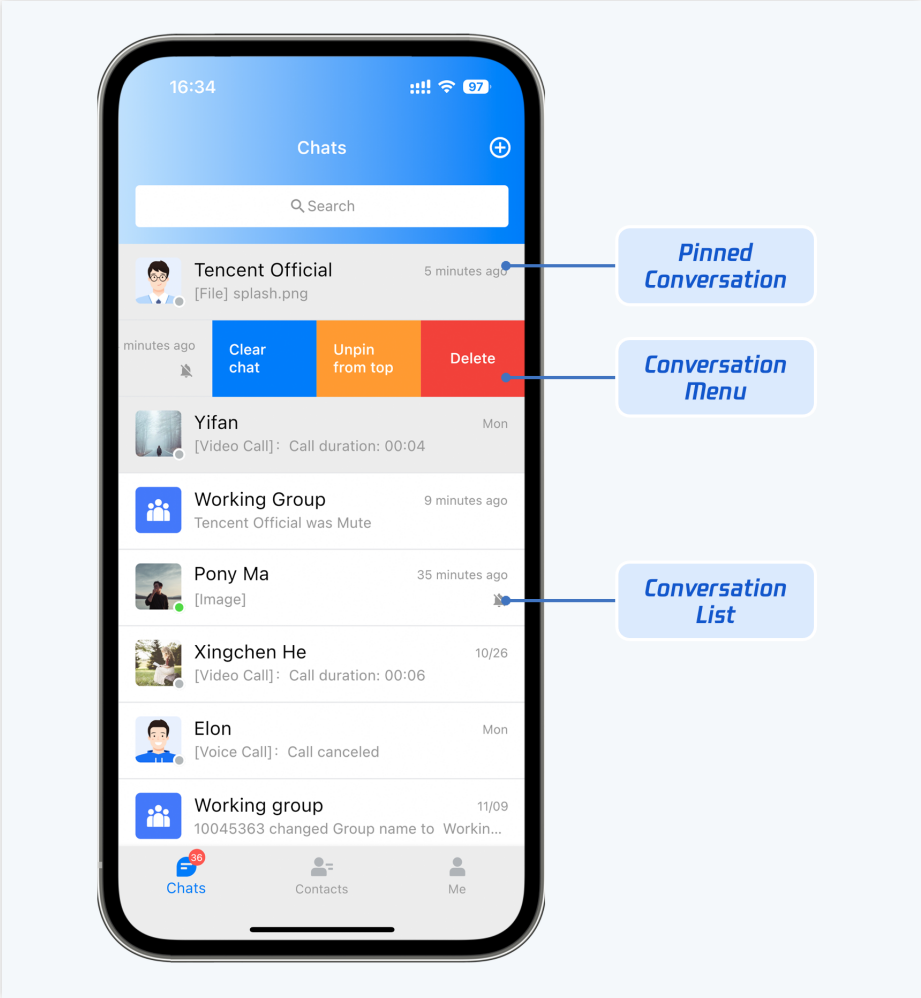


会话列表组件

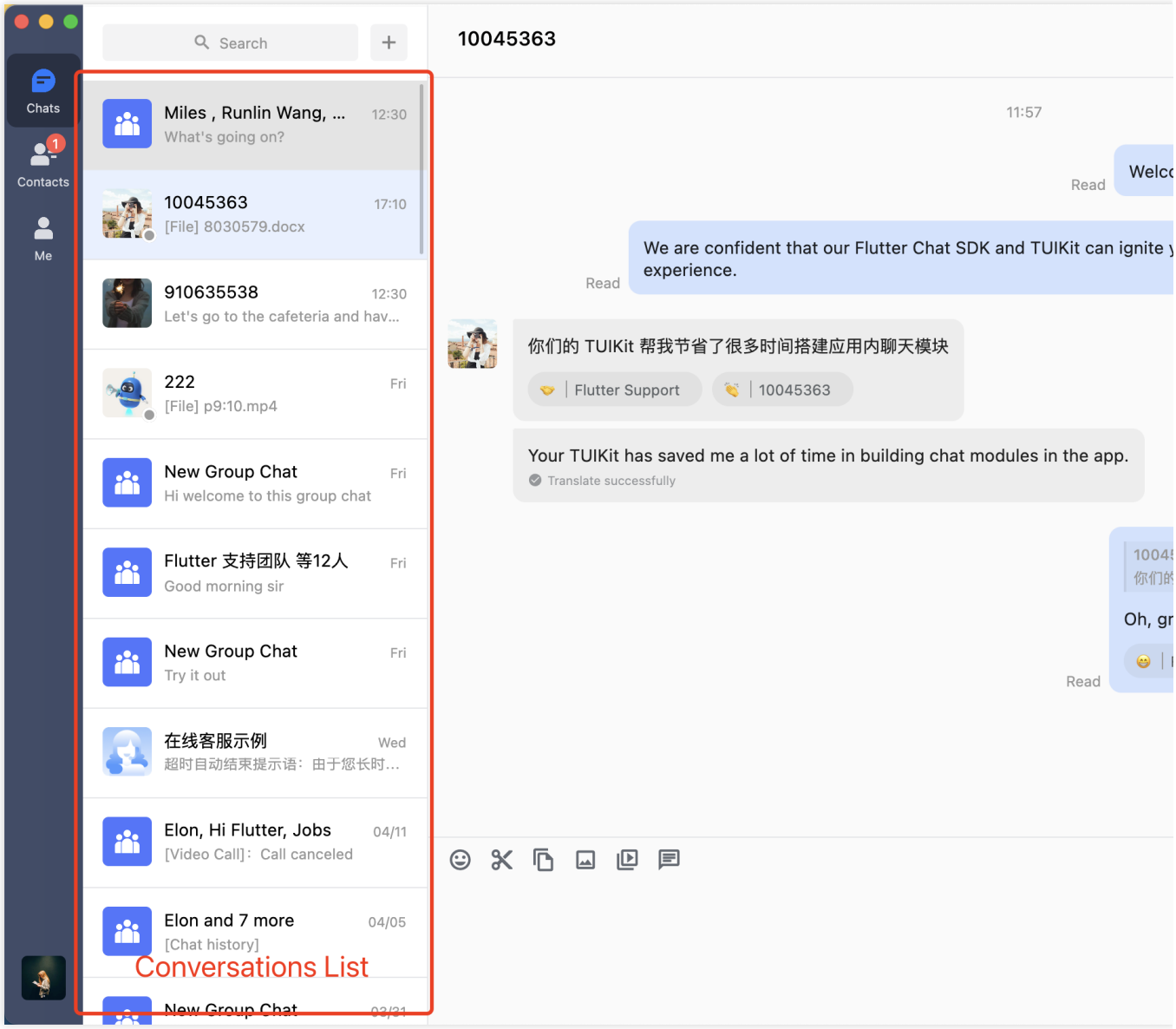
TIMUIKitConversation 主要负责会话列表的展示和编辑。界面效果如下图所示：

移动端

桌面端

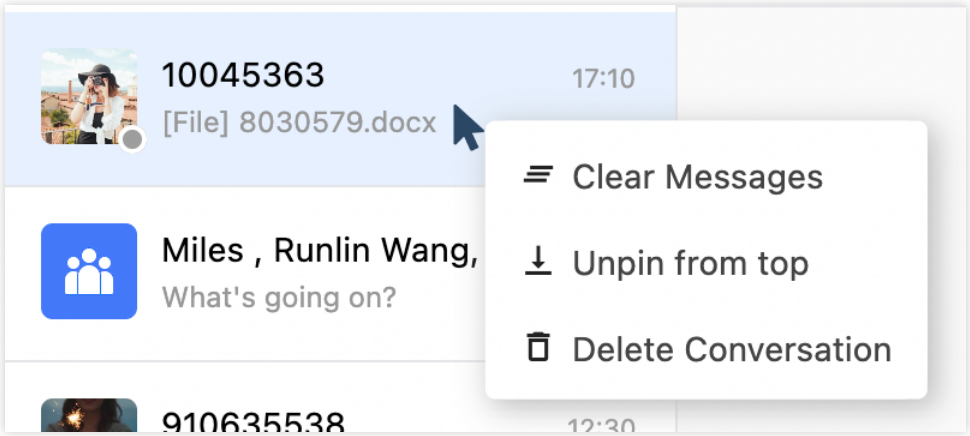


会话列表。当前选中会话 / 置顶的会话 / 未选中会话 分别用不同颜色展示。



右键操作菜单。包含 清空所有消息 / 会话置顶 / 删除会话 等能力。

版权所有：腾讯云计算（北京）有限责任公司

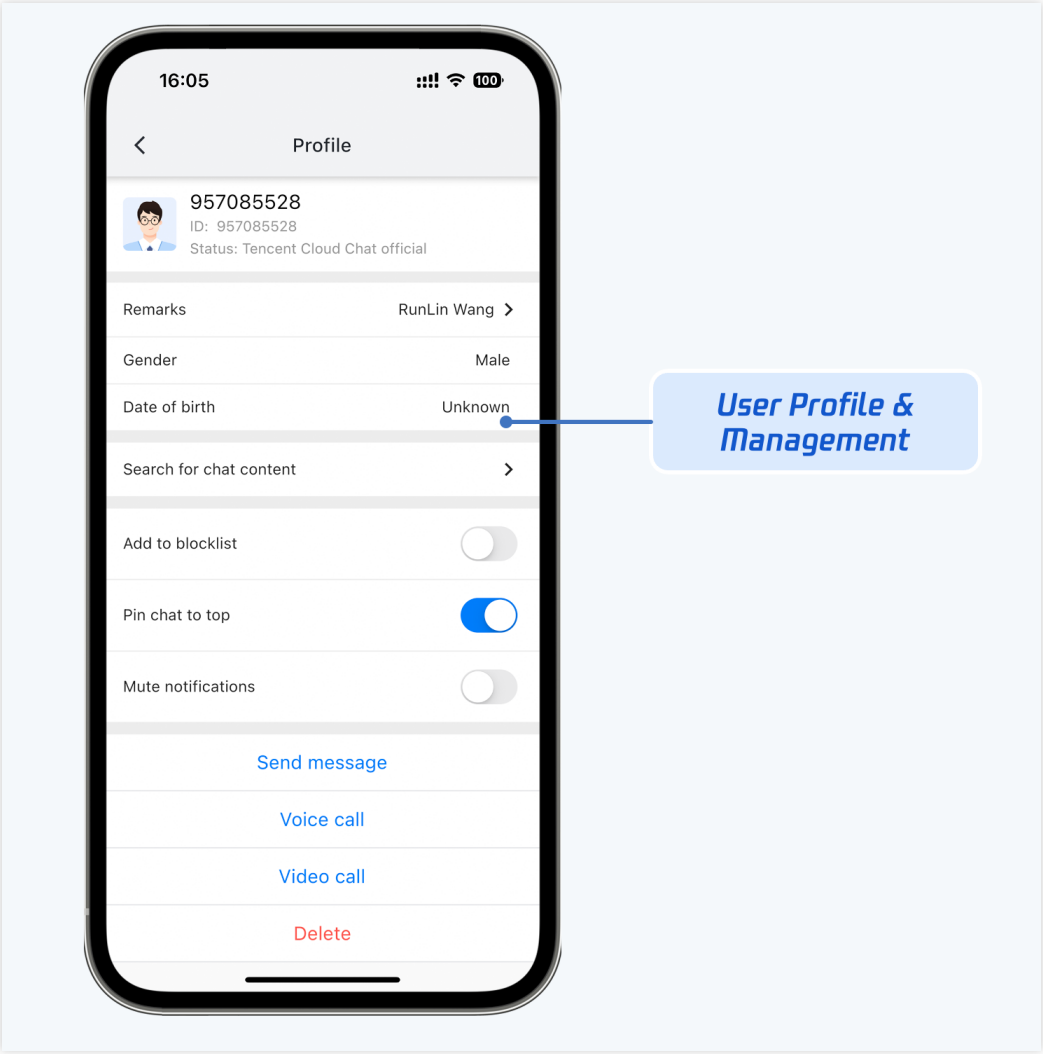


用户资料与管理组件

TIMUIKitProfile 主要负责联系人的资料展示与管理。界面效果如下图所示：

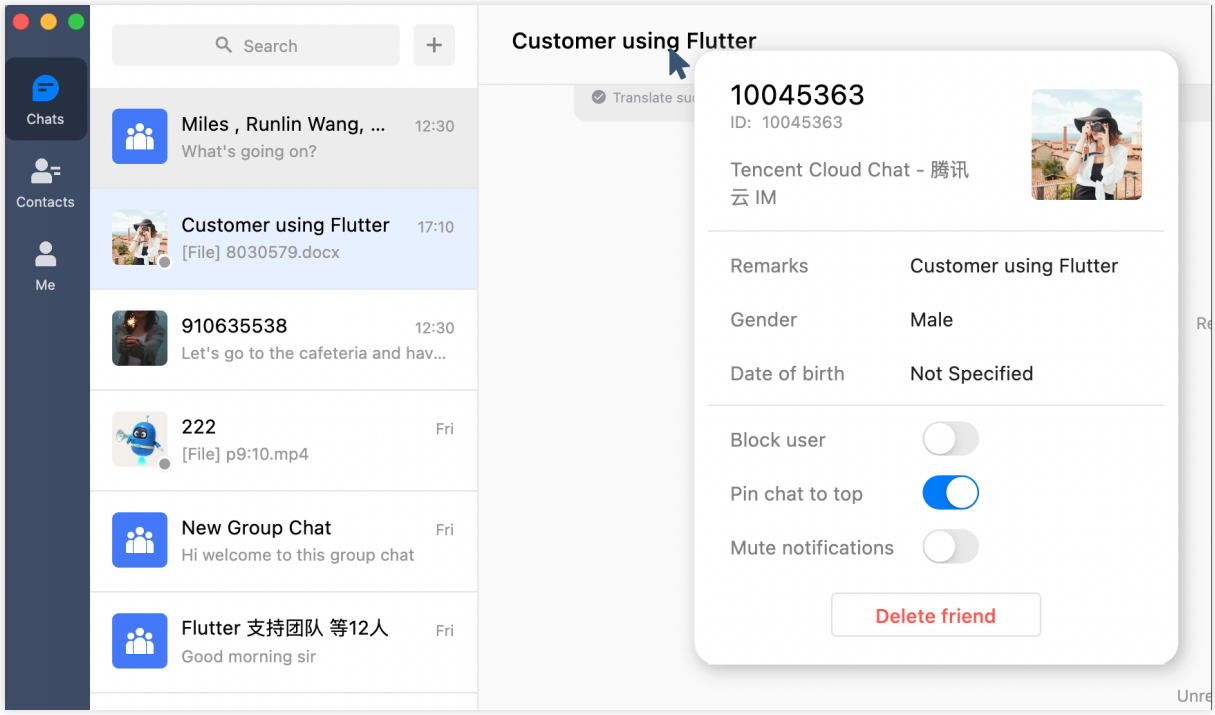
移动端

桌面端

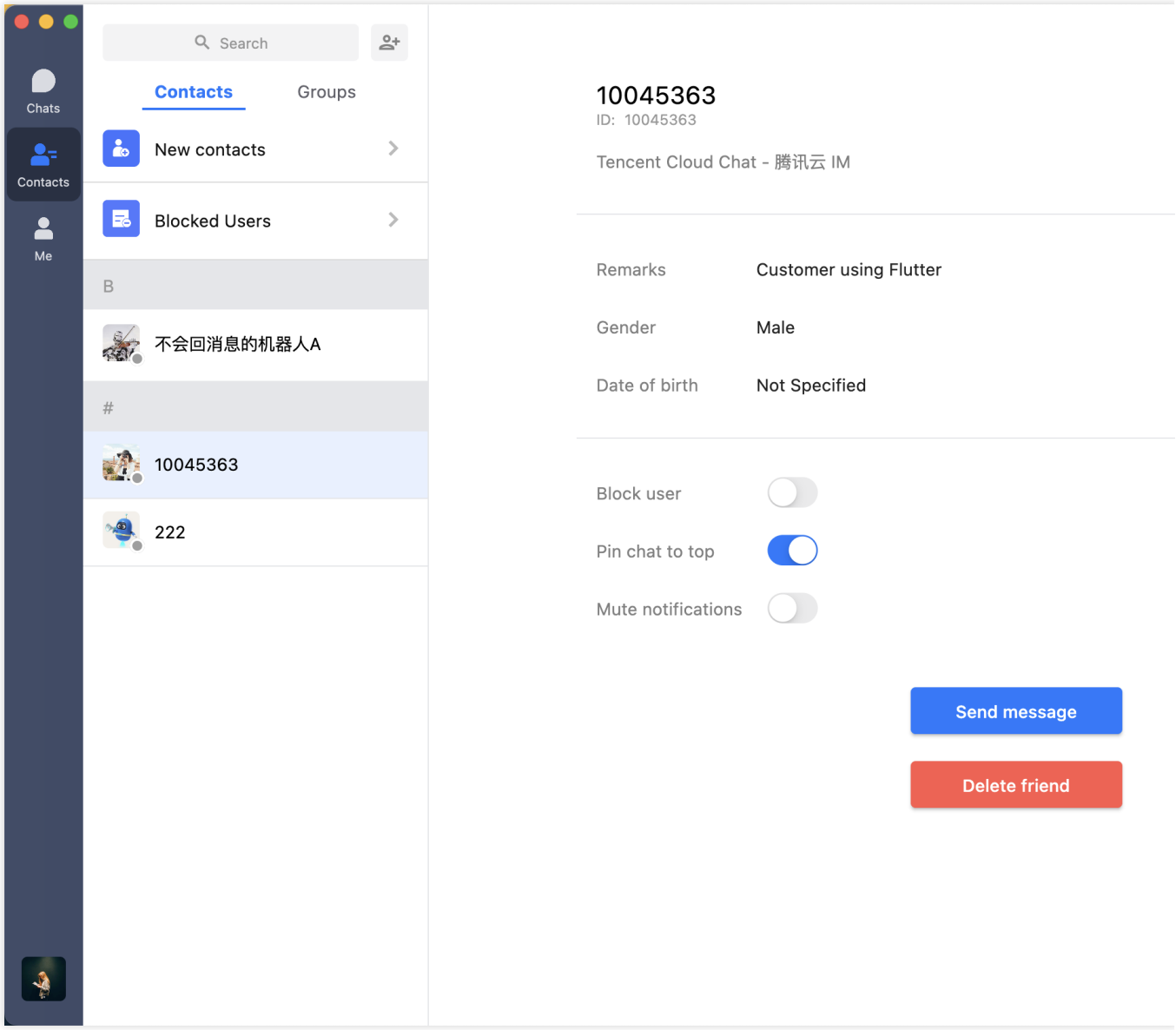


TIMUIKitProfile 组件在桌面端有两种展示方式布局，小卡片展示 及 联系人详情页。分别用于不同场景。

小卡片展示。用于 单聊标题点击 / 群聊其他成员头像点击 等场景。


Search
Chats
Contacts
Me
Miles , Runlin Wang, ...
12:30
What's going on?
Customer using Flutter
17:10
[File] 8030579.docx
910635538
12:30
Let's go to the cafeteria and hav...
222
Fri
[File] p9:10.mp4
New Group Chat
Fri
Hi welcome to this group chat
Flutter 支持团队 等12人
Fri
Good morning sir
Customer using Flutter
10045363
ID: 10045363
Tencent Cloud Chat - 腾讯云 IM
Remarks
Customer using Flutter
Gender
Male
Date of birth
Not Specified
Block user
Pin chat to top
Mute notifications
Delete friend

联系人详情页。



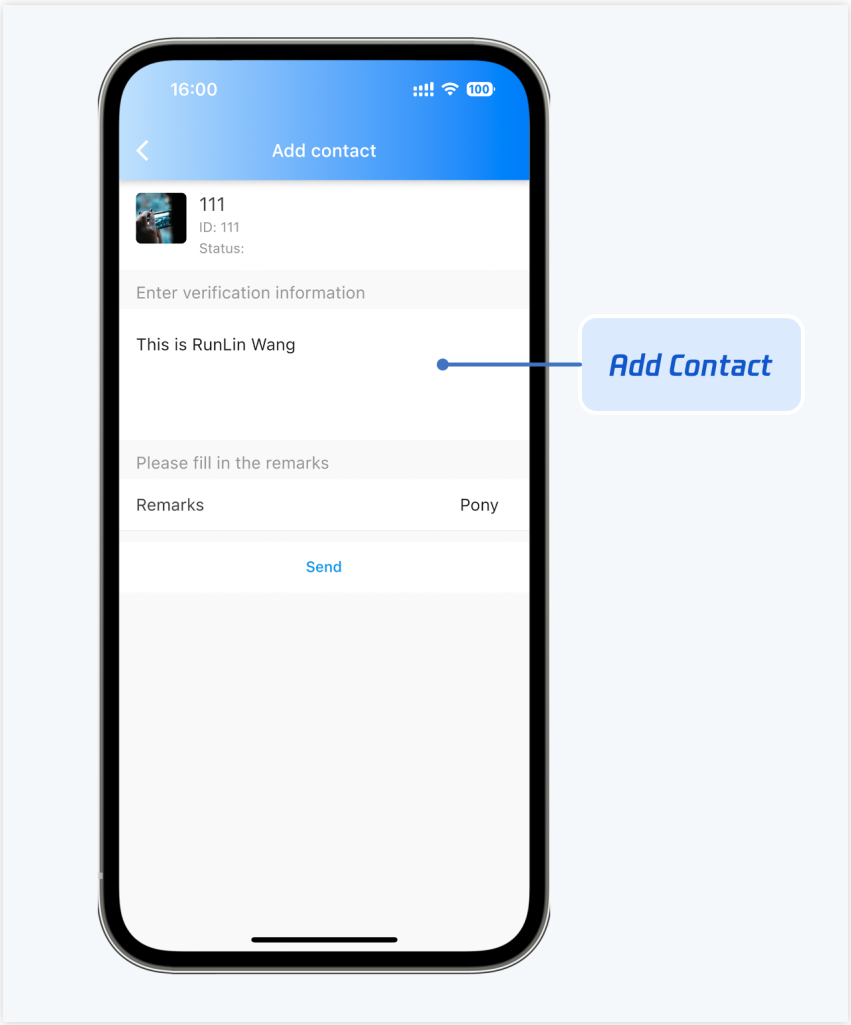
添加好友与群组系列组件

TIMUIKitAddFriend 为添加好友页面。**TIMUIKitAddGroup** 为添加群组页面。界面效果如下图所示：

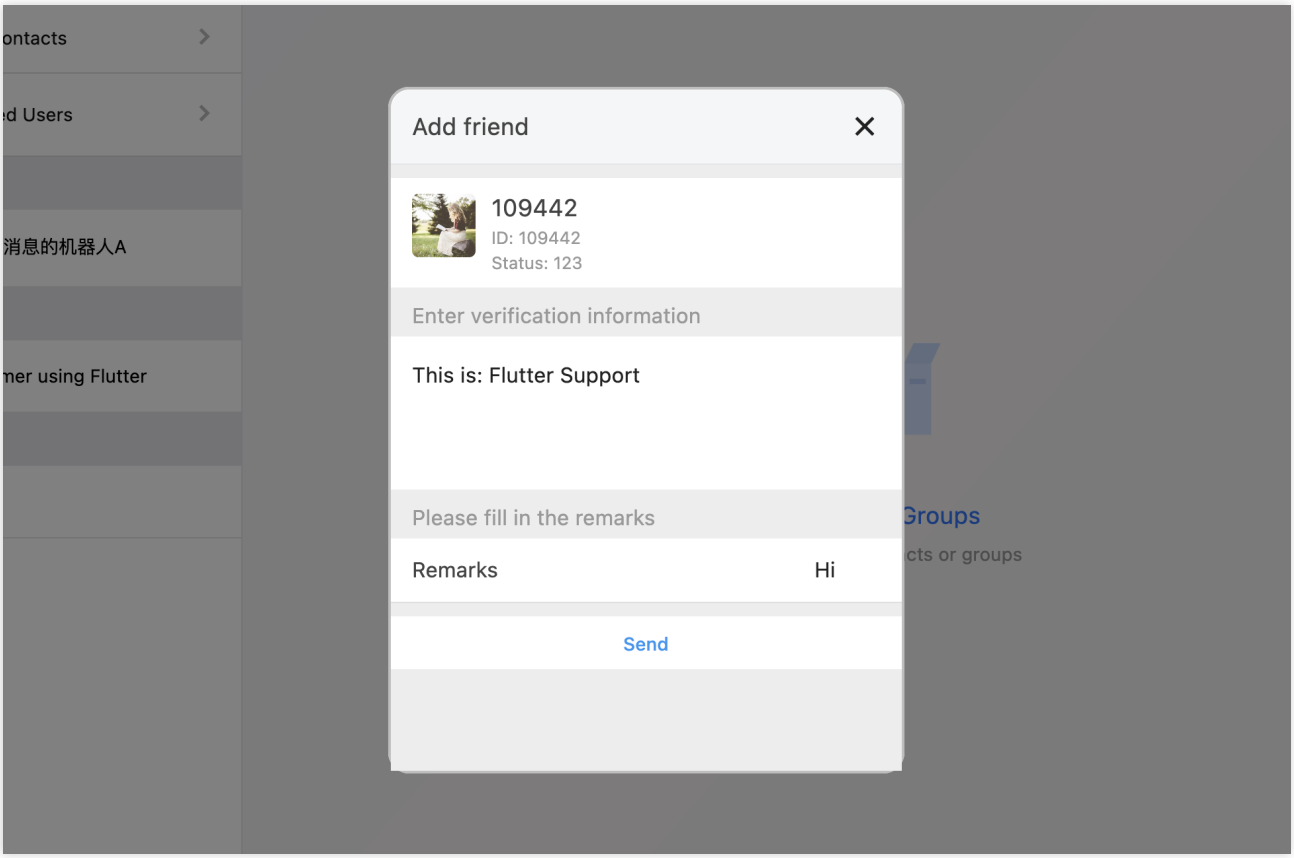
移动端

桌面端

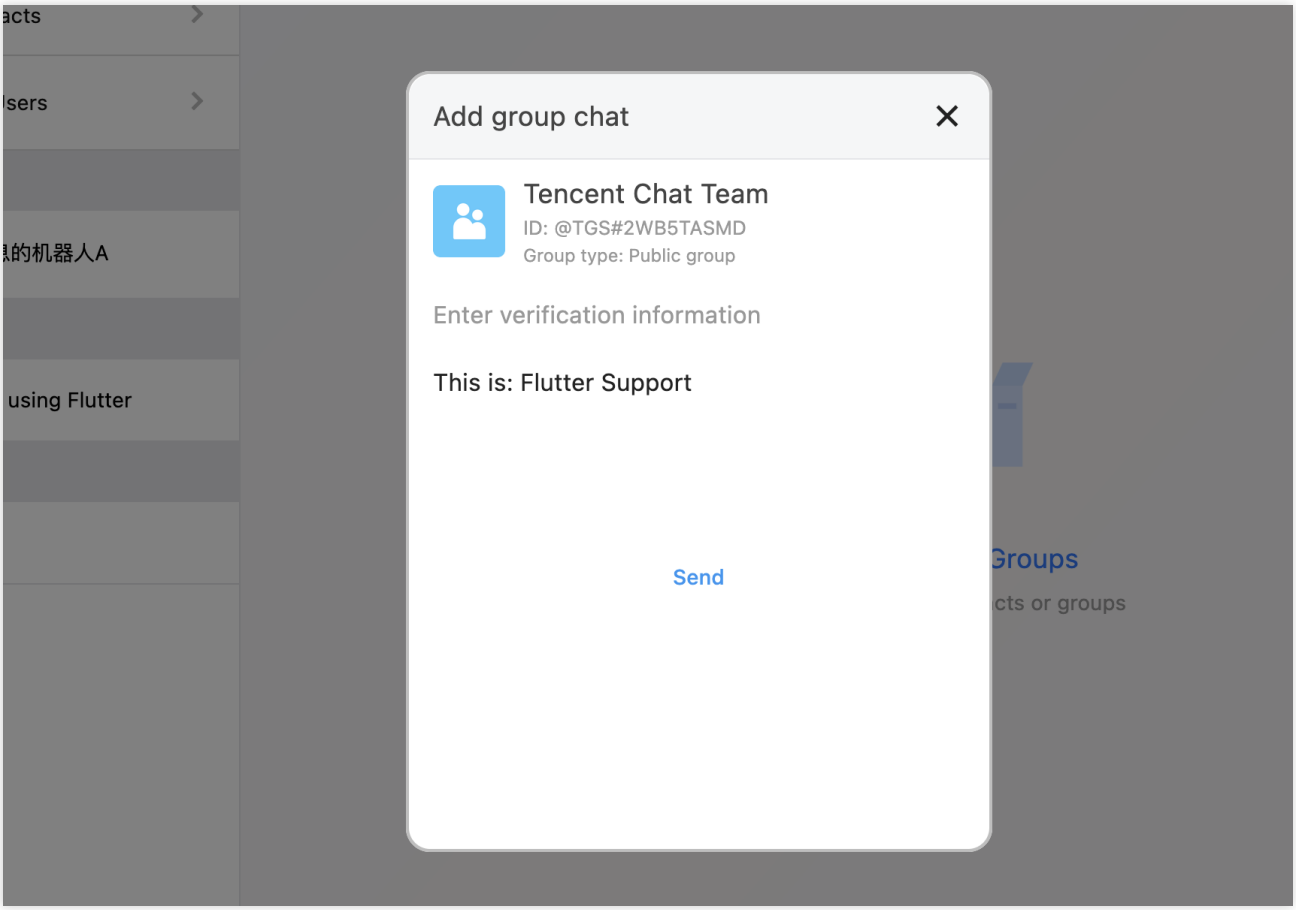
添加好友页面(TIMUIKitAddFriend)	添加群组页面(TIMUIKitAdd



添加好友 - TIMUIKitAddFriend



添加群组 - TIMUIKitAddGroup



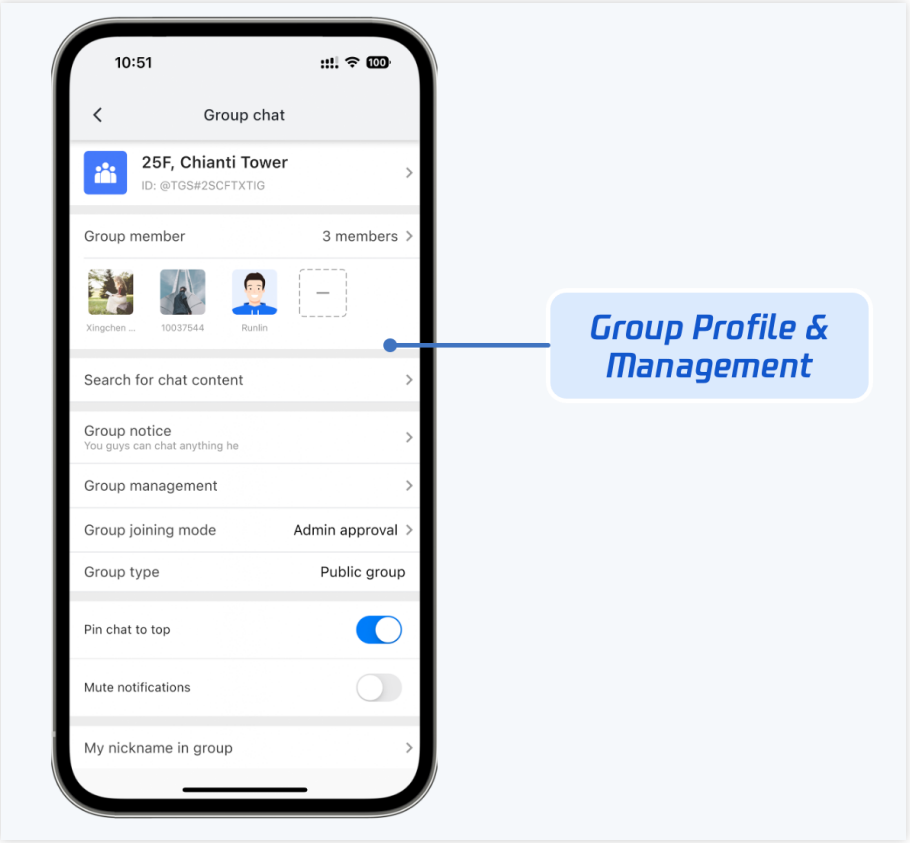
群资料与管理组件

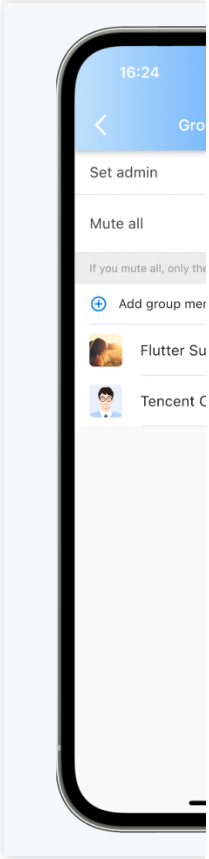
TIMUIKitGroupProfile 主要负责群资料、群成员、群组权限的展示与管理。界面效果如下图所示：

移动端

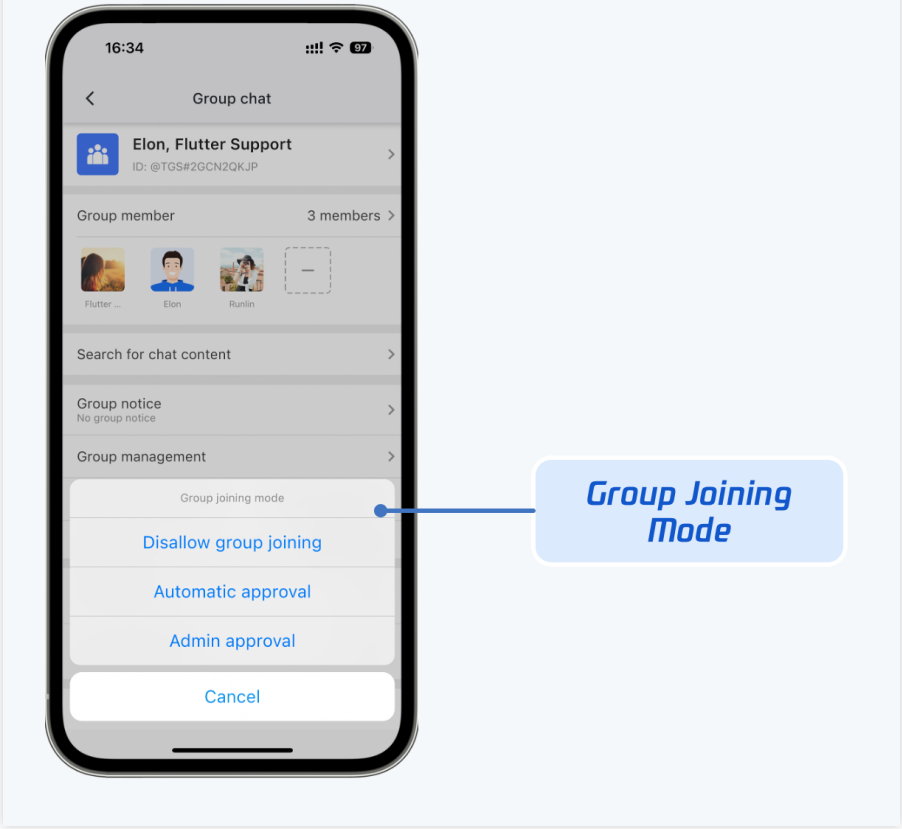
桌面端

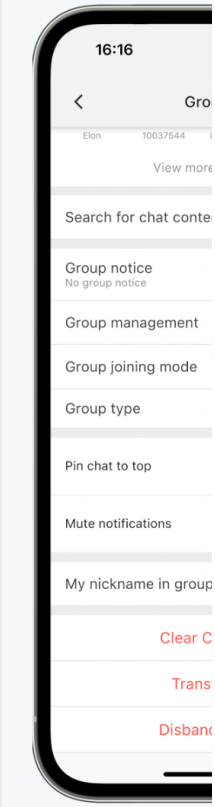
群资料及管理	群成员管理


10:51
Group chat
25F, Chianti Tower
ID: @TGS#2SCFTXTIG
Group member
3 members
Search for chat content
Group notice
You guys can chat anything he
Group management
Group joining mode
Admin approval
Group type
Public group
Pin chat to top
Mute notifications
My nickname in group
Group Profile & Management


16:24
Set admin
Mute all
Add group mer
Flutter Su
Tencent C

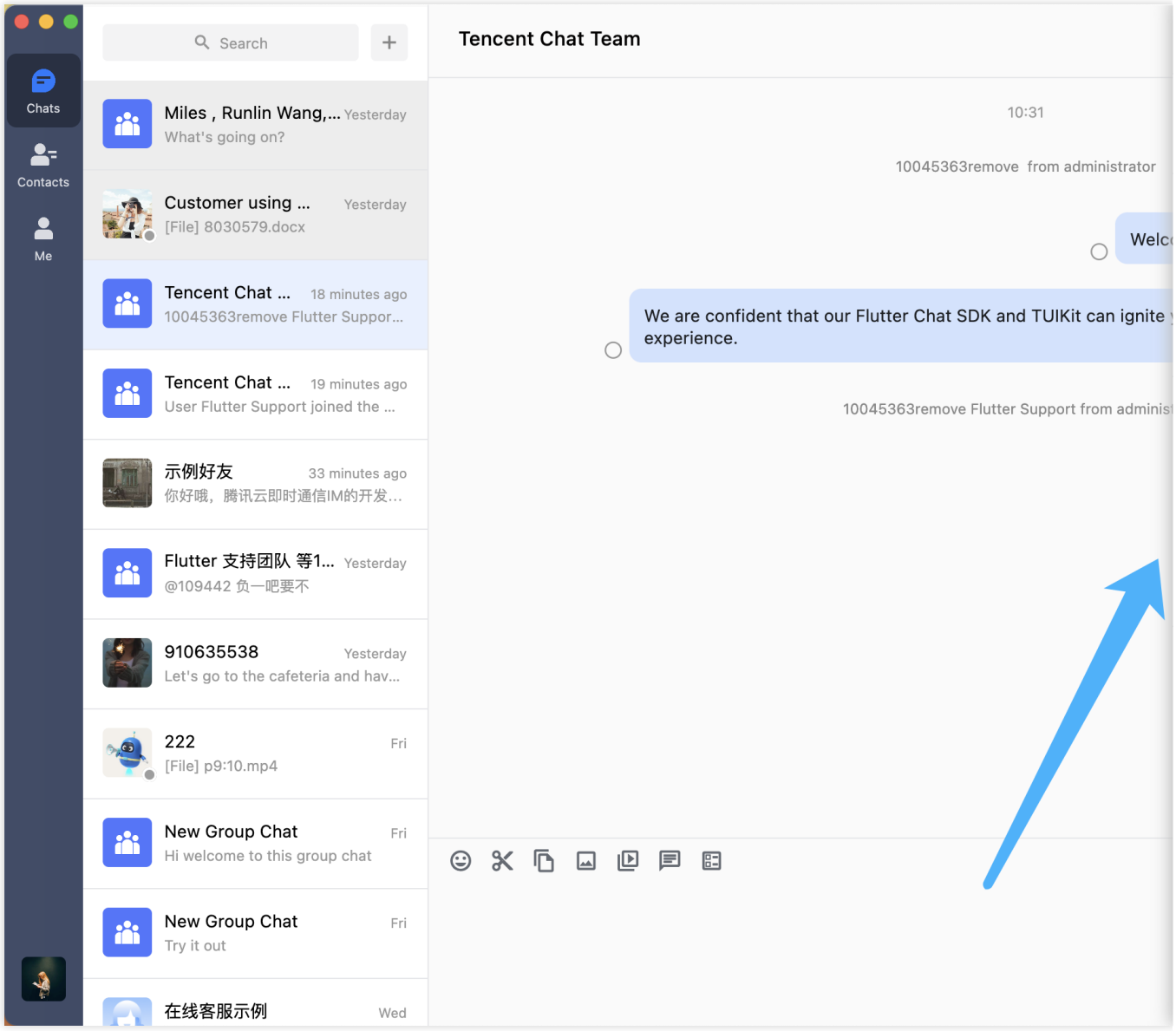
加群方式管理	群操作



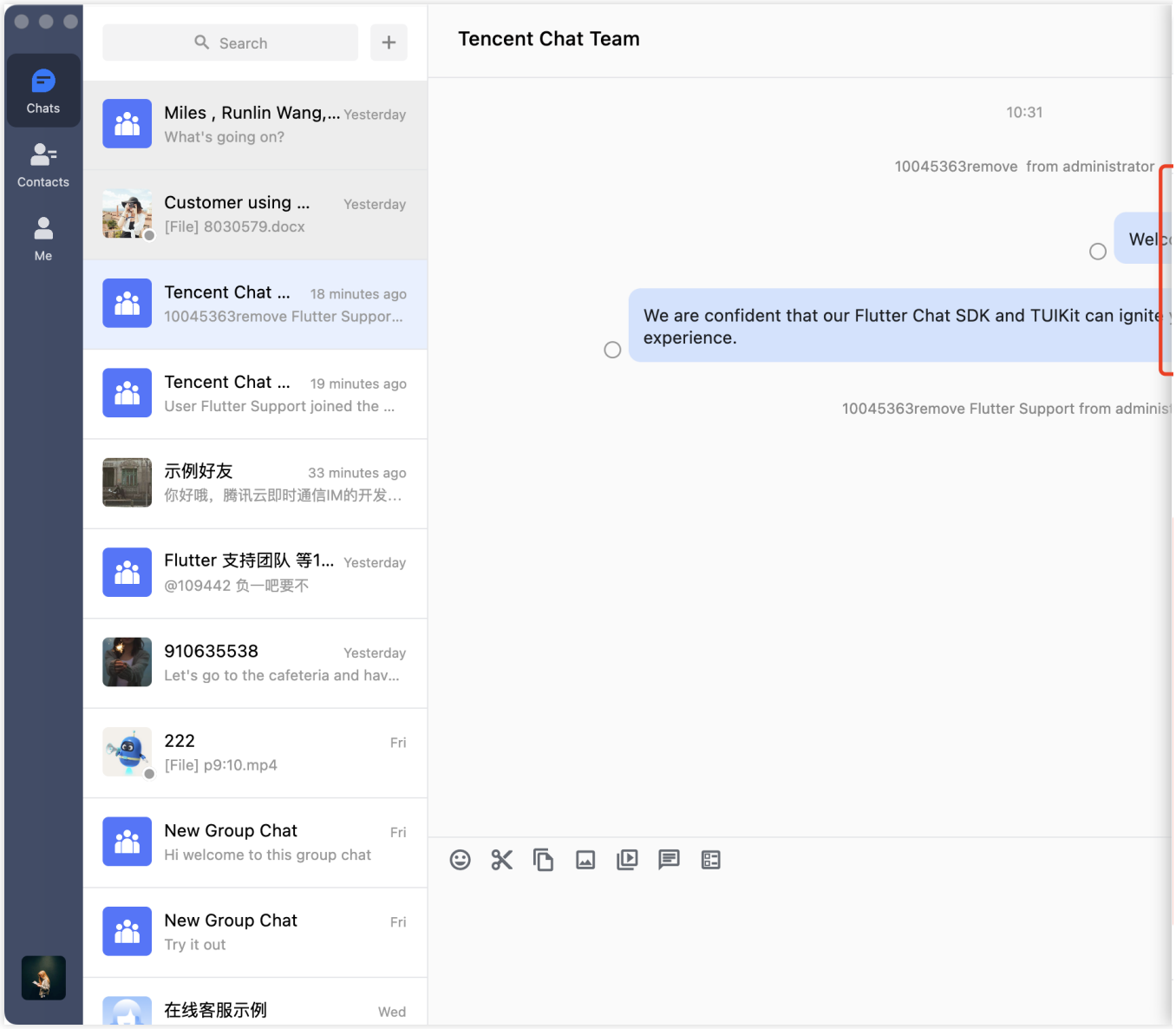


群资料及管理。可展示在群聊右侧，和移动端功能保持一致，只是界面 UI 不同。

版权所有：腾讯云计算（北京）有限责任公司

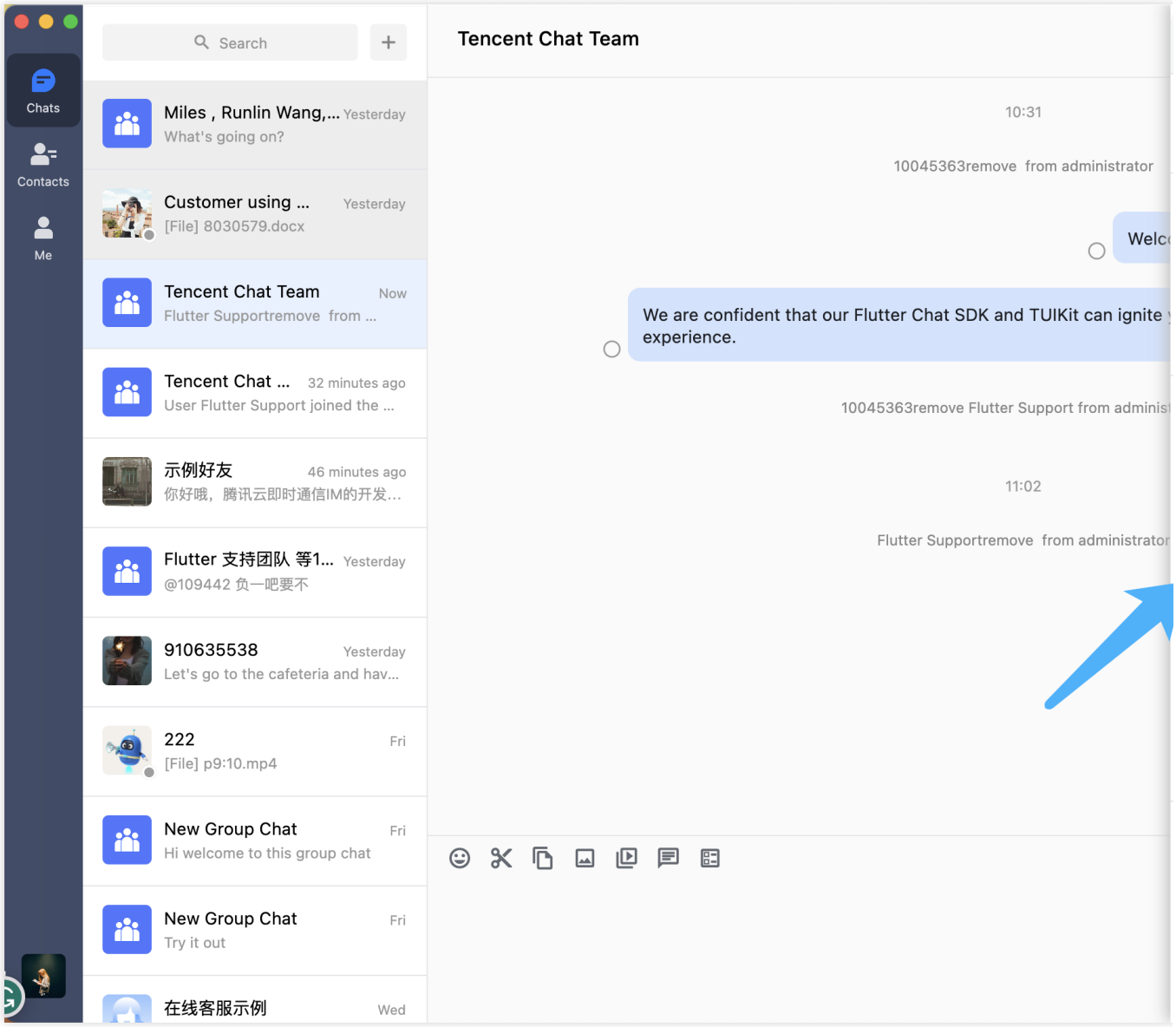


群成员管理。可在群成员卡片中查看全部群成员，并操作群成员增减。点开群成员管理面板后，可配置 管理员 / 群成员禁言 / 全员禁言 等信息。



群公告编辑。点击群公告卡片，即可将其变为输入框，进行群公告编辑并发布。

版权所有：腾讯云计算（北京）有限责任公司



本地搜索系列组件

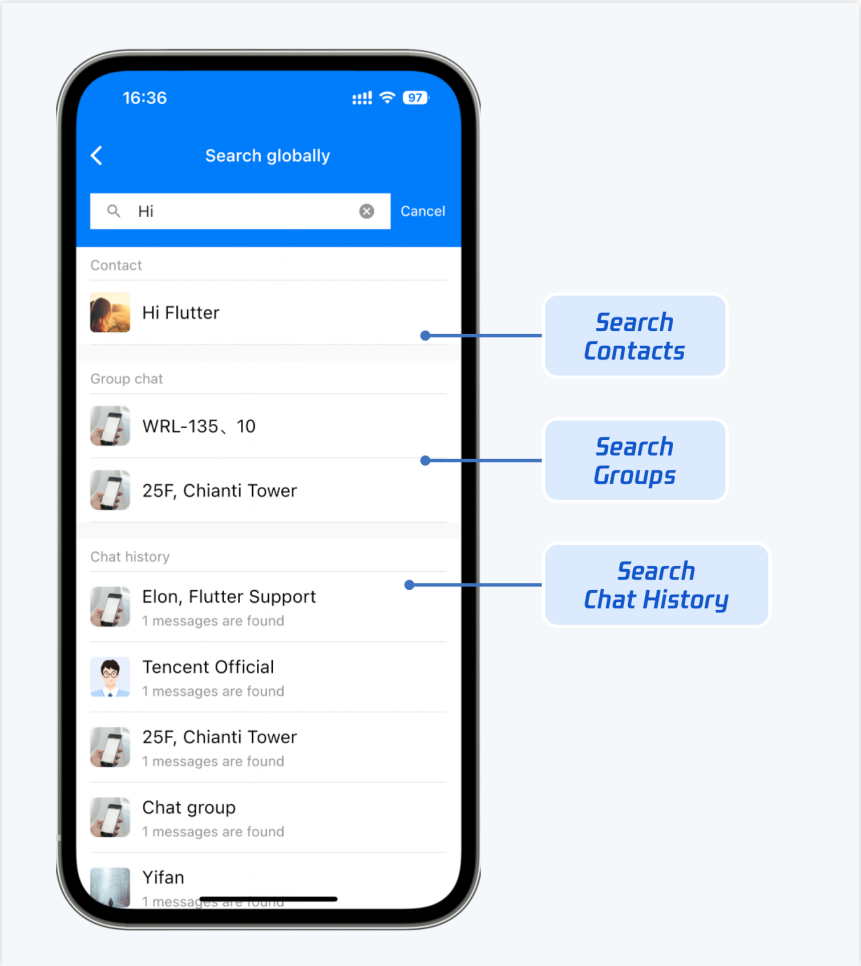
本地搜索能力，共包括两个不同的组件 **TIMUIKitSearch** 和 **TIMUIKitSearchMsgDetail**。

TIMUIKitSearch 为本地全局搜索，支持同时搜索联系人、群聊及聊天记录；**TIMUIKitSearchMsgDetail** 为会话内聊天历史记录搜索。

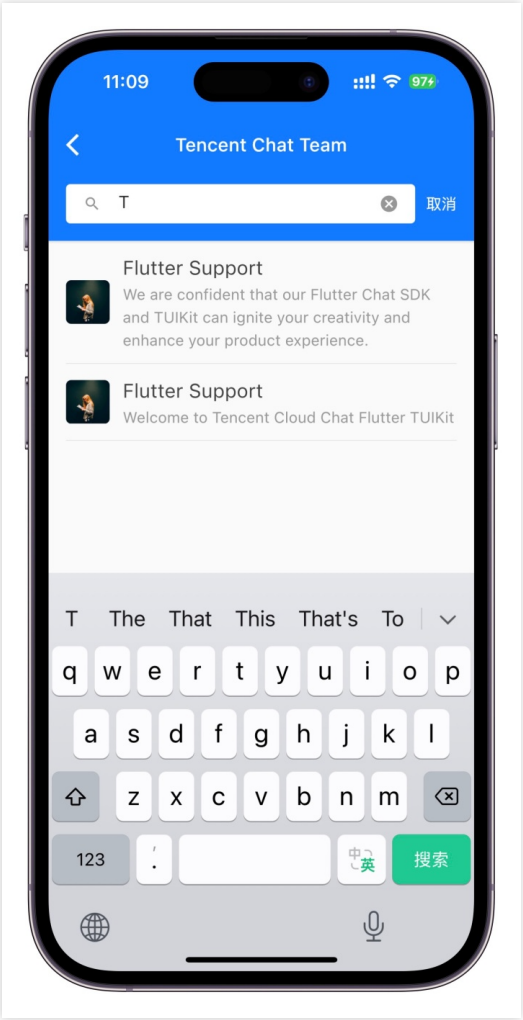
移动端

桌面端

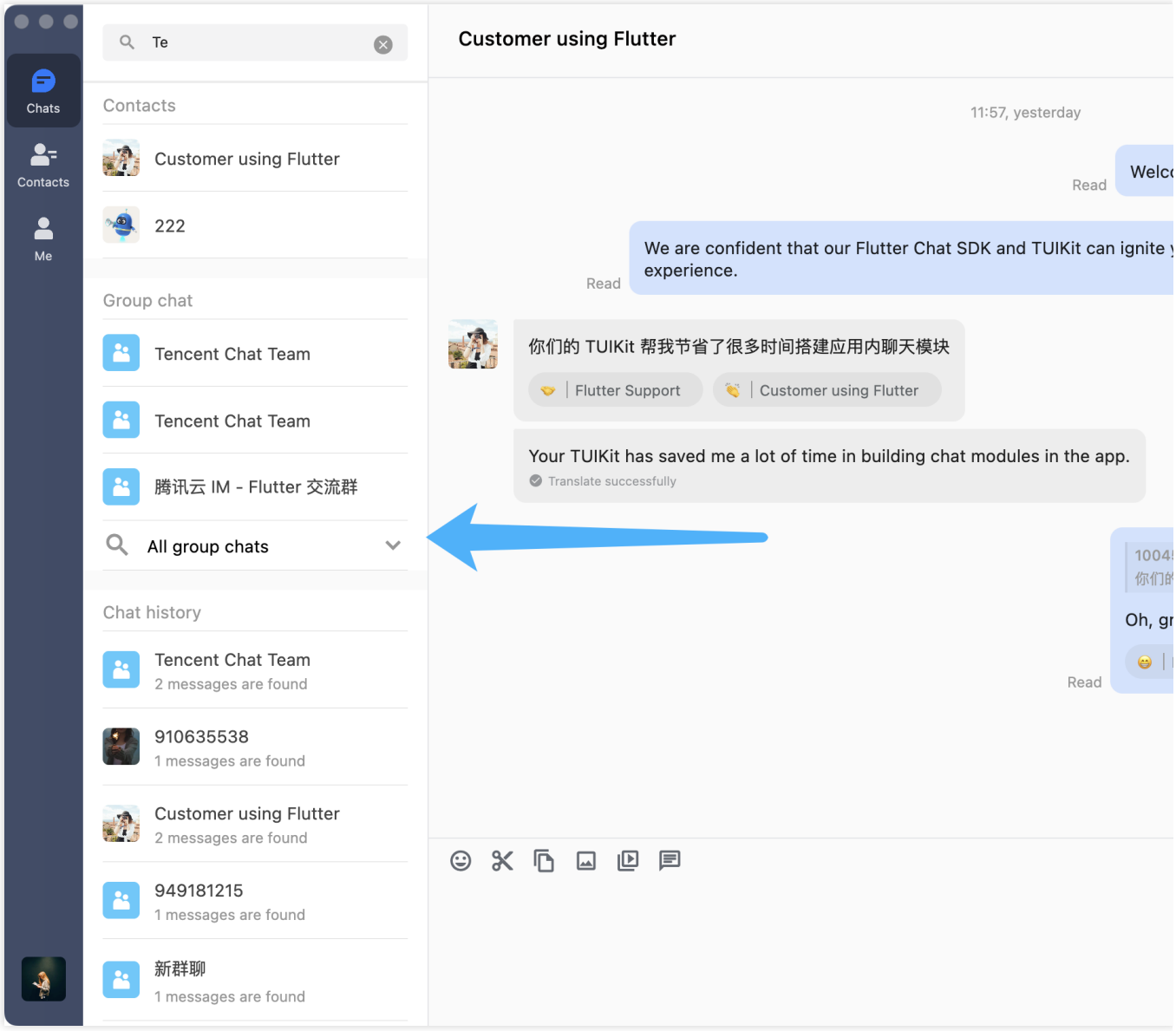
全局搜索 - TIMUIKitSearch



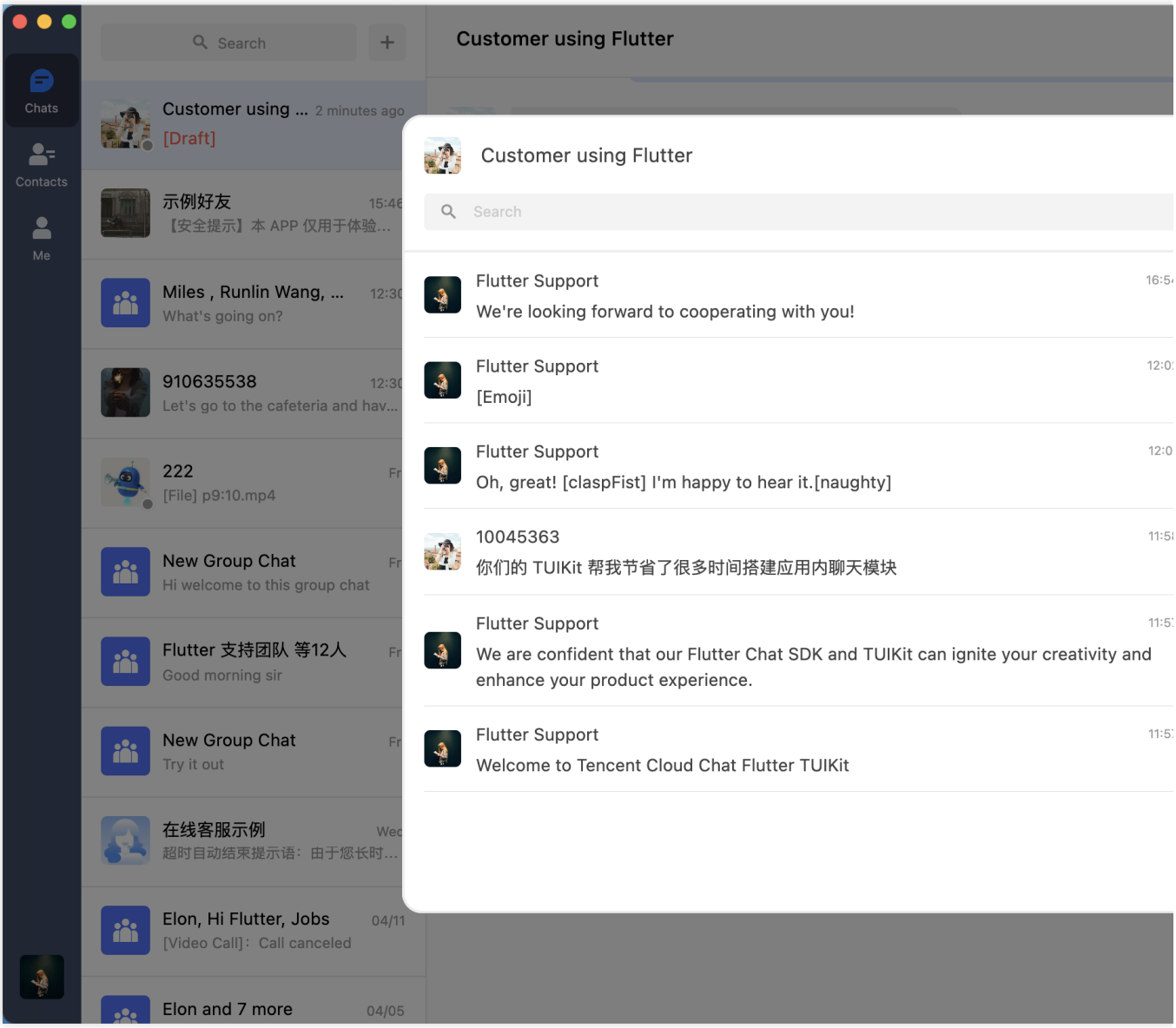
会话内搜索 - TIMUIKitSearchMsgDetail



全局搜索 - TIMUIKitSearch



会话内搜索 - TIMUIKitSearchMsgDetail



音视频通话

TUICallKit插件 用于语音、视频通话。目前仅支持移动端。

消息推送

您可通过我们的 Flutter 推送插件 集成消息推送能力，涵盖离线推送与在线推送。消息推送效果如下图所示：

Tue 22 26°
14:34
Tencent Chat
Show less
Tencent Chat
Video message
now
Tencent Chat
File message
now
Tencent Chat
Location message
now
Tencent Chat
invites you to a video call
now
Tencent Chat
invites you to a voice call
now
Tencent Chat
Hi this is Tencent Cloud Chat
now
Offline Push Notification

快速开始

Android

最近更新时间：2024-06-24 17:05:38

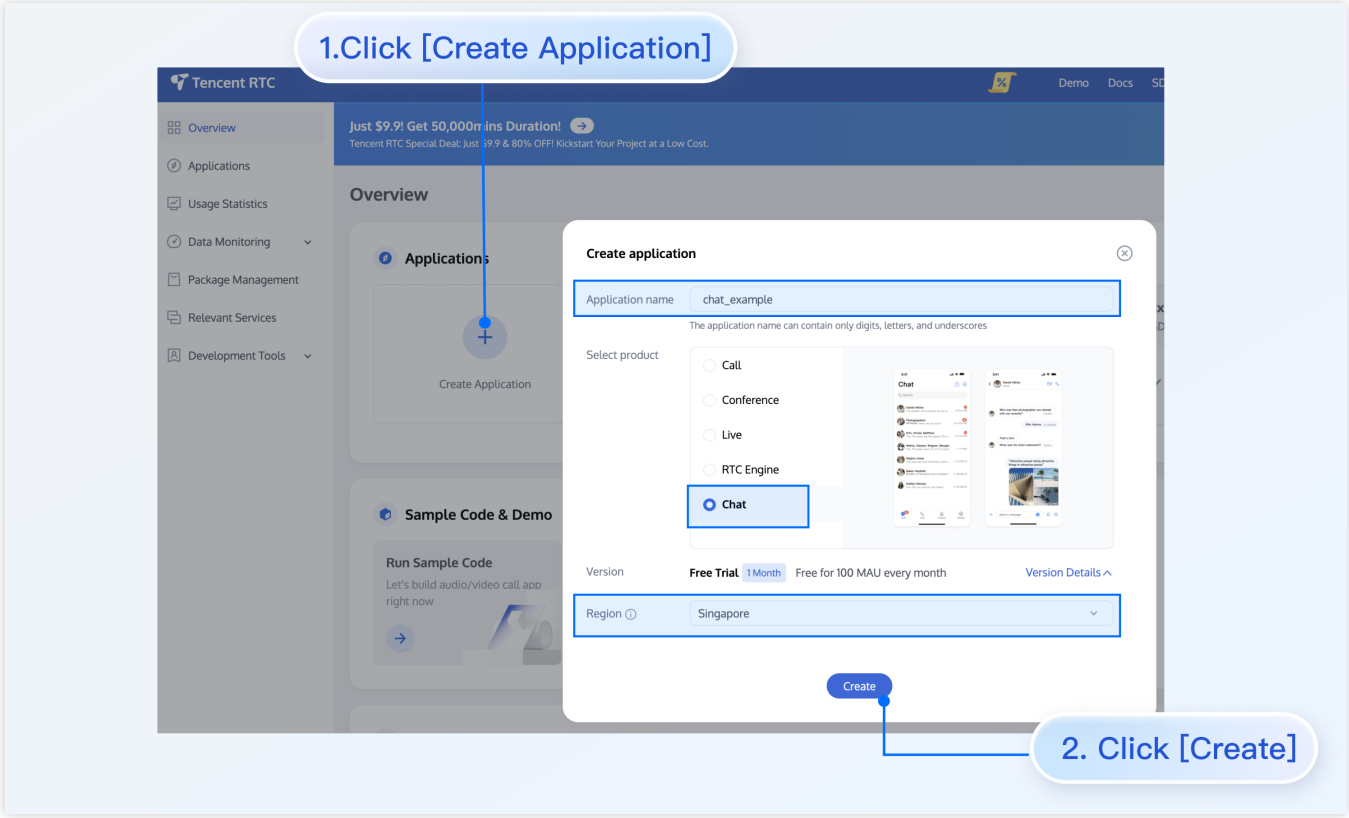
本文会引导您集成 `TUIKit` 或 `TUIChat` 并成功发送第一条消息。

开发环境要求

Android Studio-Giraffe
Gradle-7.2
Android Gradle Plugin Version-7.0.0
kotlin-gradle-plugin-1.5.31

创建应用

在集成 TUIKit 之前，您需要先去控制台创建一个新的 Chat 应用，步骤如下：
1. 注册控制台账号。
2. 进入 `Application`，点击 `Create application`，弹出 application 信息填写框。
3. 填写 Application name，选择 product 为 Chat，选择合适的 Region。
操作完成后，您会在 `My Applications` 列表中看到刚才创建的 application。
注意：
请记录下该 application 的 SDKAppID，后续步骤会使用到。另外，请严格保管好 SDKSecretKey，不要透露给无关人员。
操作步骤图示如下：



创建用户账号

创建 application 只能保证您可以正常初始化 SDK。如果要成功发消息，您还需要在 application 中创建用户账号。创建账号方式有很多，例如直接在控制台创建，或者通过 API 在客户端注册，您可以选择任意一种合适的方式。

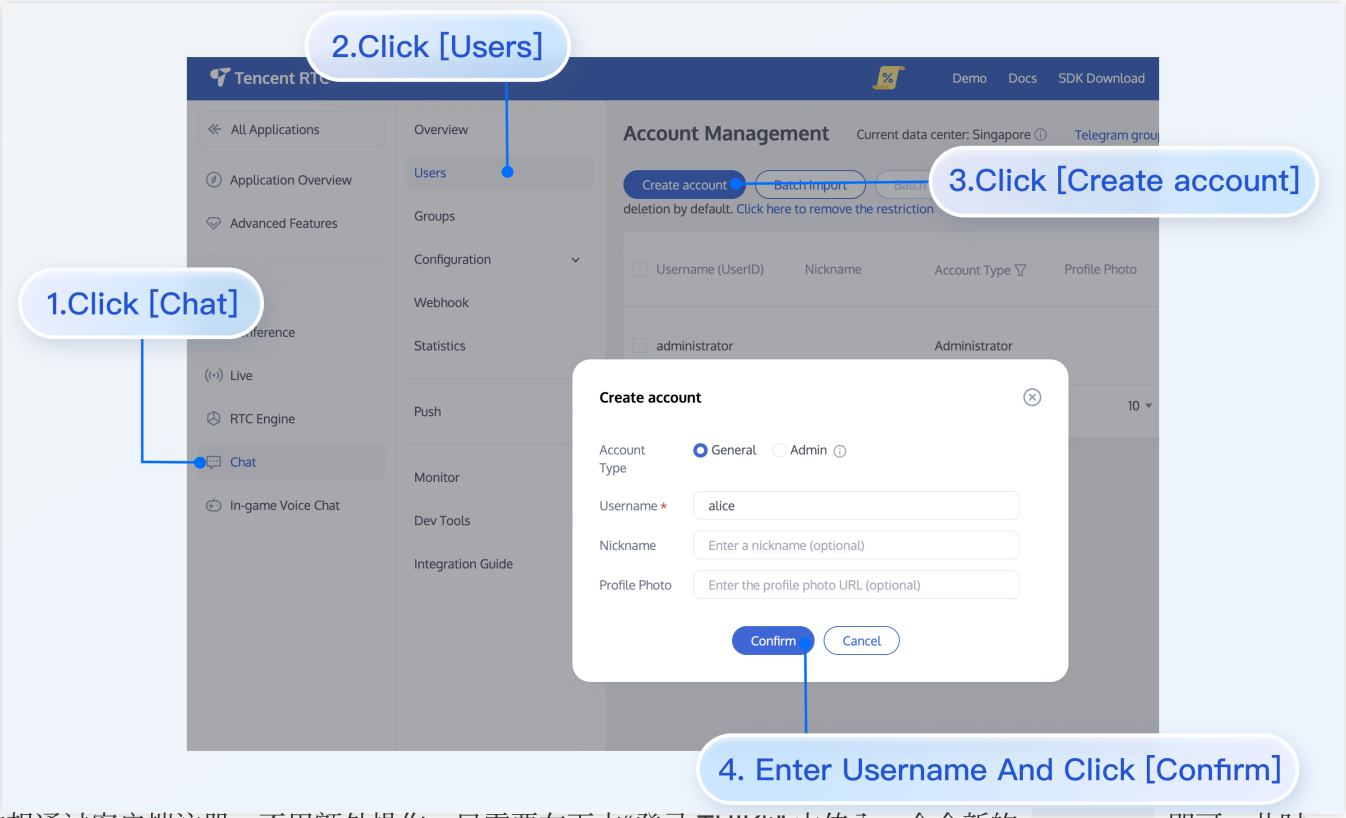
注意：

发消息至少是两个用户之间进行，因此您在此环节至少要创建 2 个账号。请记录下这 2 个账号的 userID，后续步骤会使用到。

如果您想在控制台创建，步骤如下：

1. 单击进入您上面创建的 application，会在左侧边栏看到 Chat 产品入口，单击进入。
2. 进入 Chat 产品子页面后，点击 `Users`，进入用户管理页面。
3. 单击 `Create account`，弹出创建账号信息填写框。如果只是普通成员，我们建议您选择 `General` 类型。虽然 `Nickname` 不是必填项，我们依然建议您设置。如果界面上不方便展示 `userID`，您可以通过 `Nickname` 识别出不同用户。

图示如下：



如果你想通过客户端注册，不用额外操作，只需要在下文“登录 TUIKit” 中传入一个全新的 `userID` 即可，此时 TUIKit 会自动为您注册该 `userID` 。

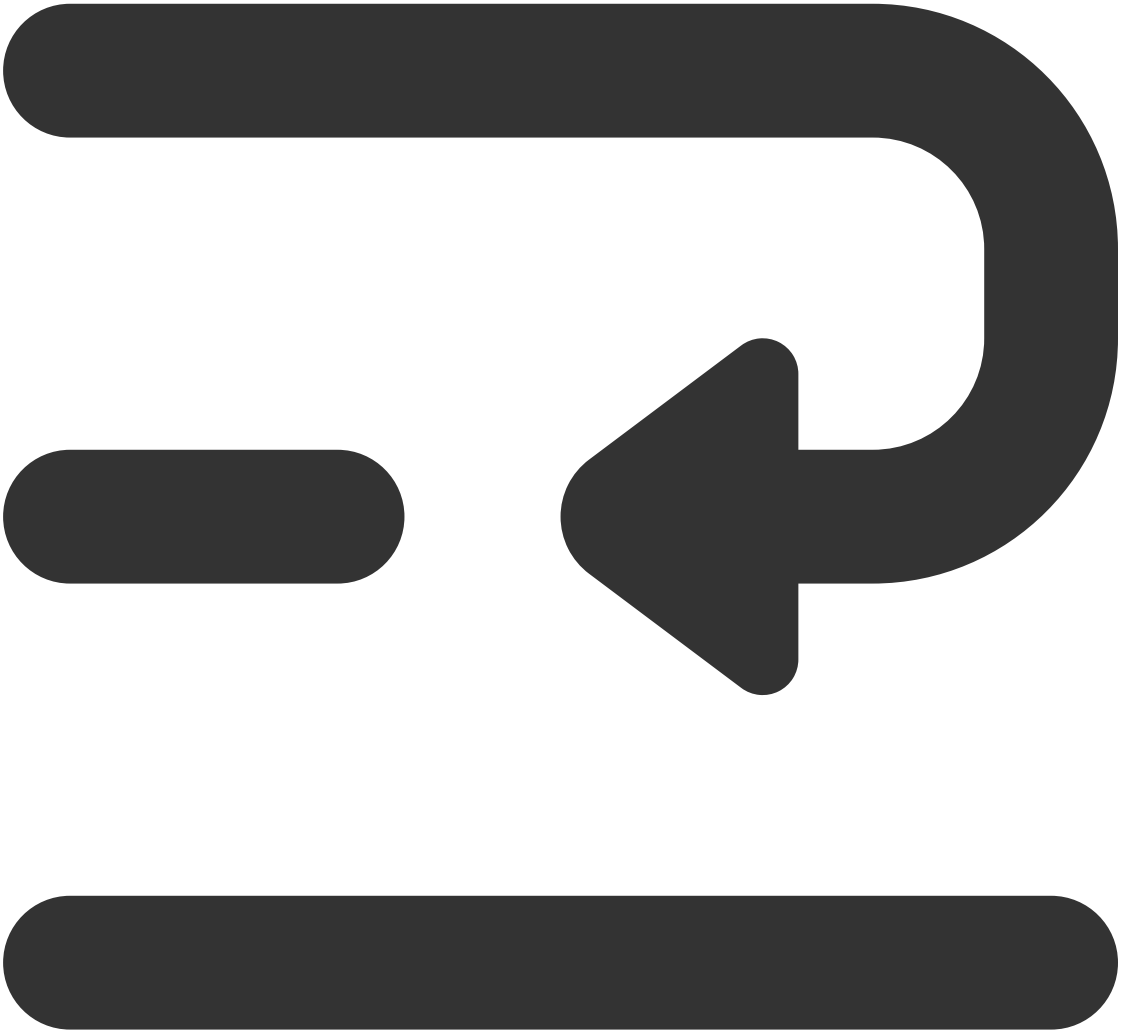
集成 TUIKit

聊天互动中发送消息的功能是由 `TUIChat` 实现的，您至少要集成 `TUIChat` 才能正常收发消息，其他的组件，例如 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 等，您可以按需集成。
如果您需要多个 UI 组件，可以集成 TUIKit，请参见文档：集成 TUIKit。
如果您只需要集成 TUIChat，请参见文档：仅集成聊天。

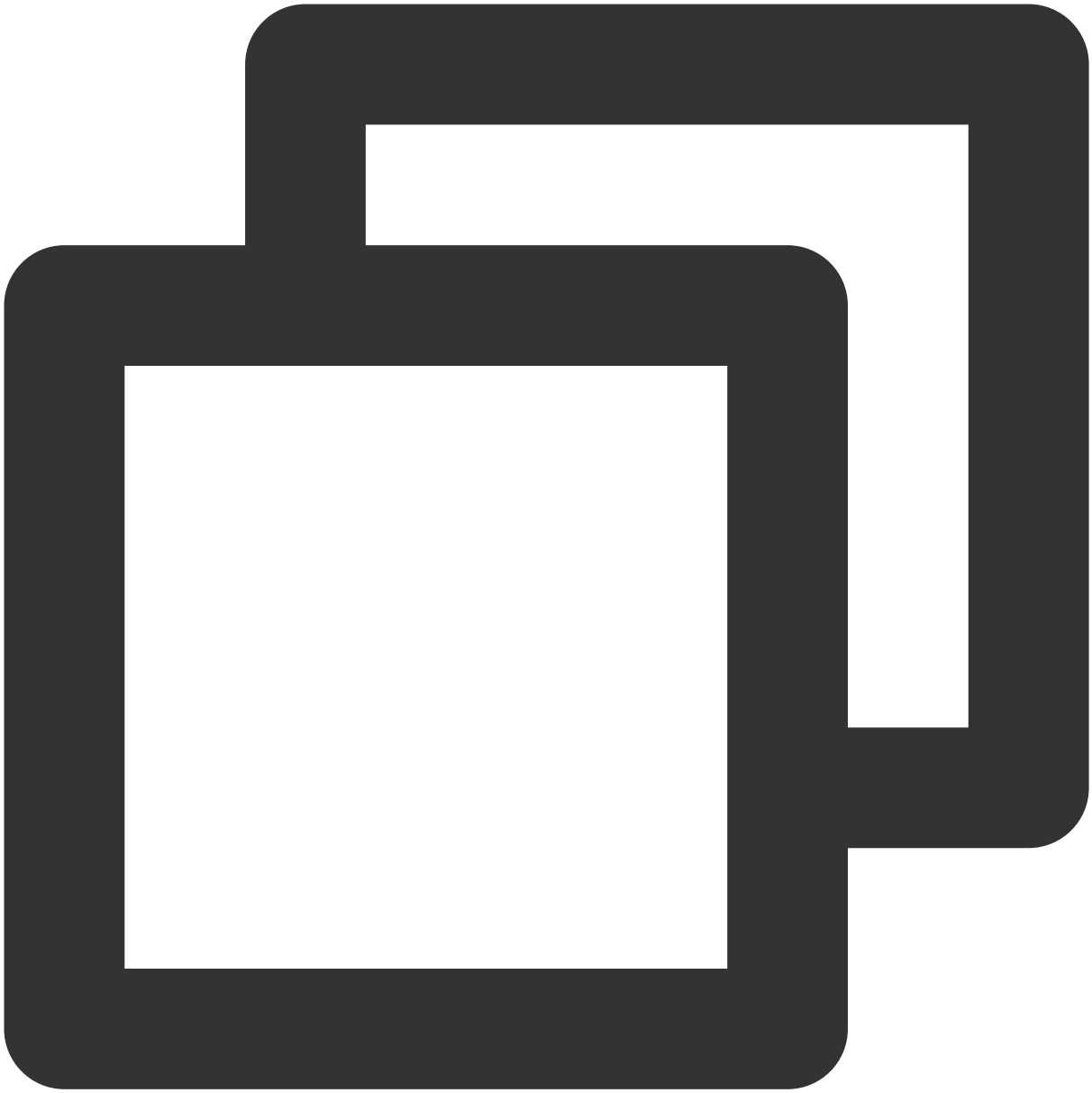
登录 TUIKit

使用 TUIKit 组件里的功能都需要登录，由 TUILogin 提供登录接口，如下：

版权所有：腾讯云计算（北京）有限责任公司



版权所有：腾讯云计算（北京）有限责任公司



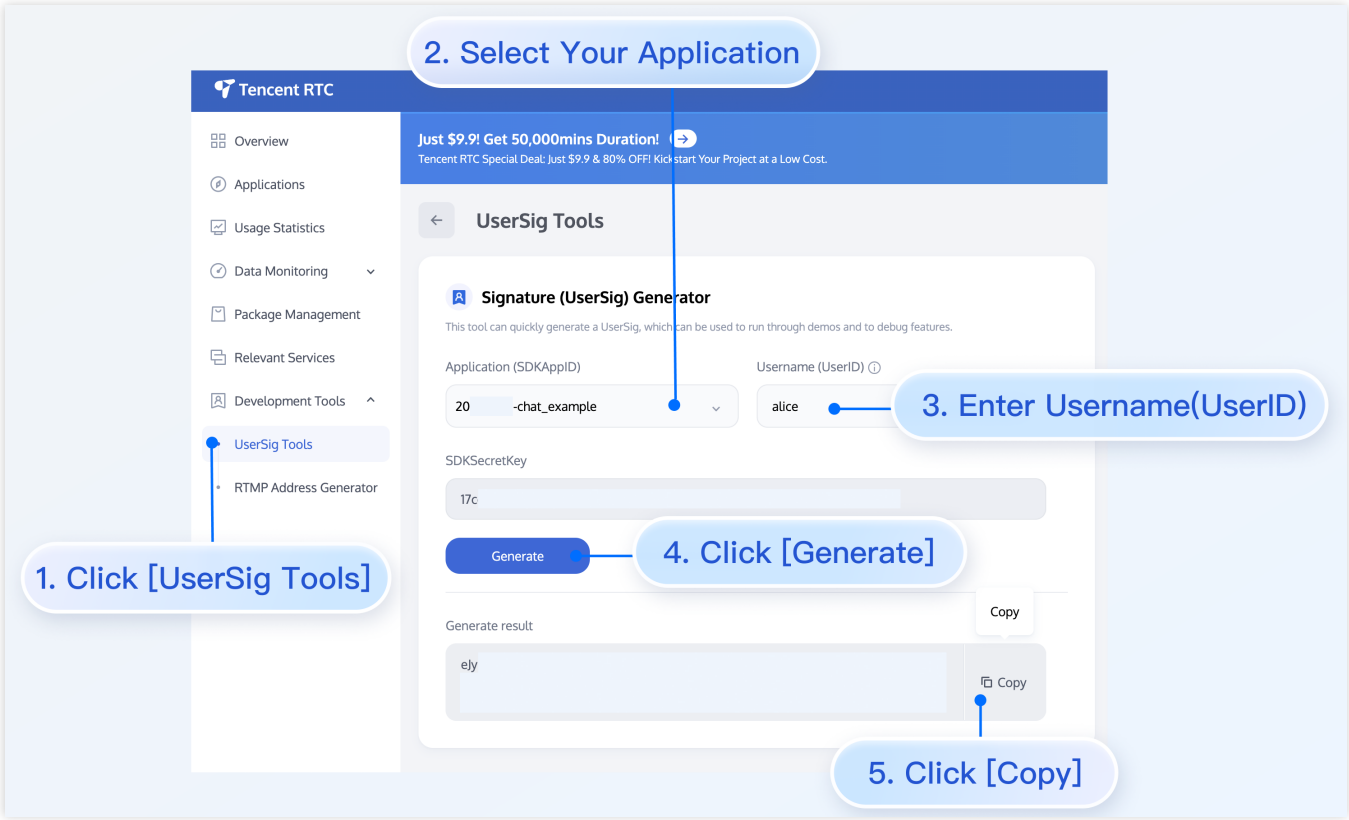
```
// API location: TUICore/TUILogin.java
// Called when login is clicked on the user UI
TUILogin.login(context, sdkAppID, userID, userSig, new TUICallback() {
    @Override
    public void onSuccess() {
    }

    @Override
    public void onError(final int code, final String desc) {
    }
});
```

该接口要求输入 3 个参数：

sdkAppID，新创建应用的 SDKAppID，已在上文步骤中获取到。

userID，user1 的 userID，已在上文步骤中获取到，注意不是用户的 NickName。

userSig，user1 的 userSig，可使用控制台提供的开发工具实时生成，路径：主页 > `Development Tools` > `UserSig Tools` > `Signature (UserSig) Generator` ，图示如下：



跳转聊天界面

为了实现发消息的目标，接下来要做的是：

1. 使用上述注册的账号之一（下文称之为 user1）登录 TUIKit，此时 user1 上线了。

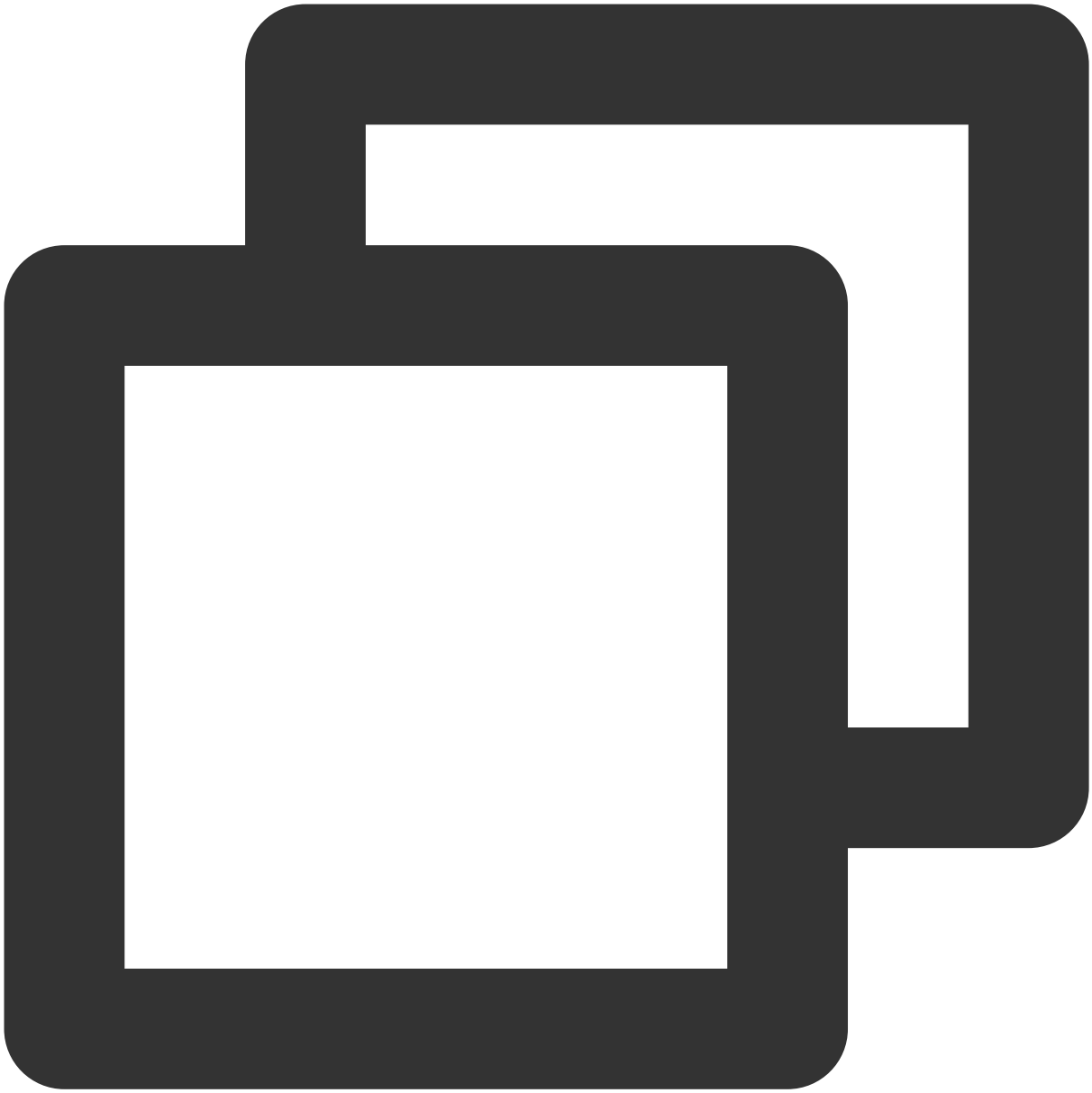
2. user1 给另一个账号（下文称之为 user2）发消息，user2 可以不登录，跟 user1 可以没有任何好友关系。

说明：

此处讲解的是登录 user1 后给 user2 发消息。如果您希望 user1 和 user2 能聊天互动，需要使用同样的步骤登录 user2 并进入与 user1 的聊天界面。

您可以在 user1 登录成功的回调里跳转到聊天界面，就可以给 user2 发消息了。

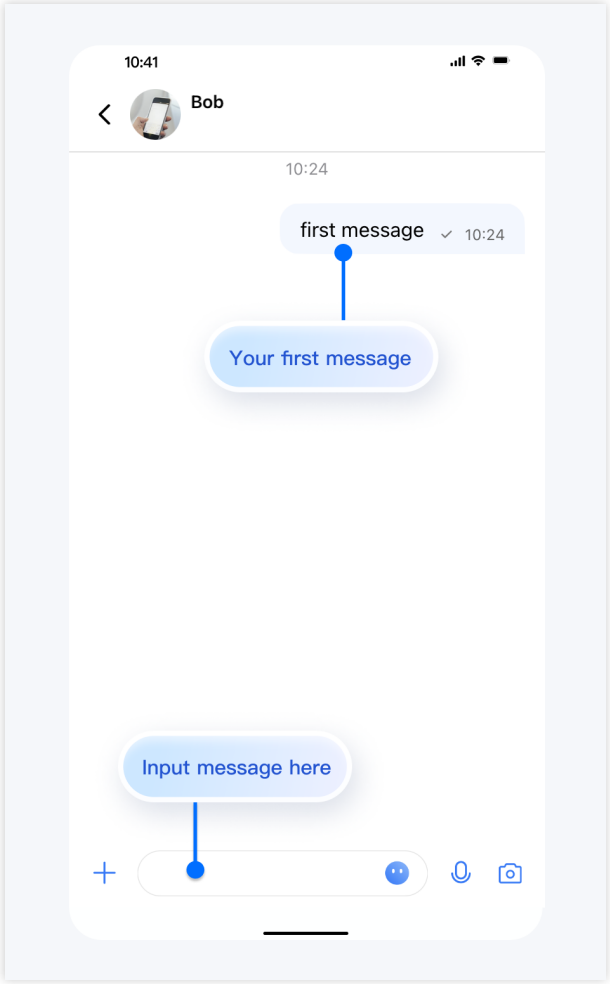
示例代码如下，其中 chatID 需要传入聊天对象 user2 的 userID。



```
Intent intent = new Intent(context, TUIC2CChatMinimalistActivity.class);
intent.putExtra(TUIConstants.TUIChat.CHAT_ID, "chatID");
intent.putExtra(TUIConstants.TUIChat.CHAT_TYPE, V2TIMConversation.V2TIM_C2C);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(intent);
```

发送第一条消息

操作完上述步骤，您可以跳转到如下的聊天界面。快点手动点击输入框，发送您的第一条消息吧：



联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

iOS

最近更新时间：2024-06-24 17:06:14

本文会引导您集成 `TUIKit` 或 `TUIChat` 并成功发送第一条消息。

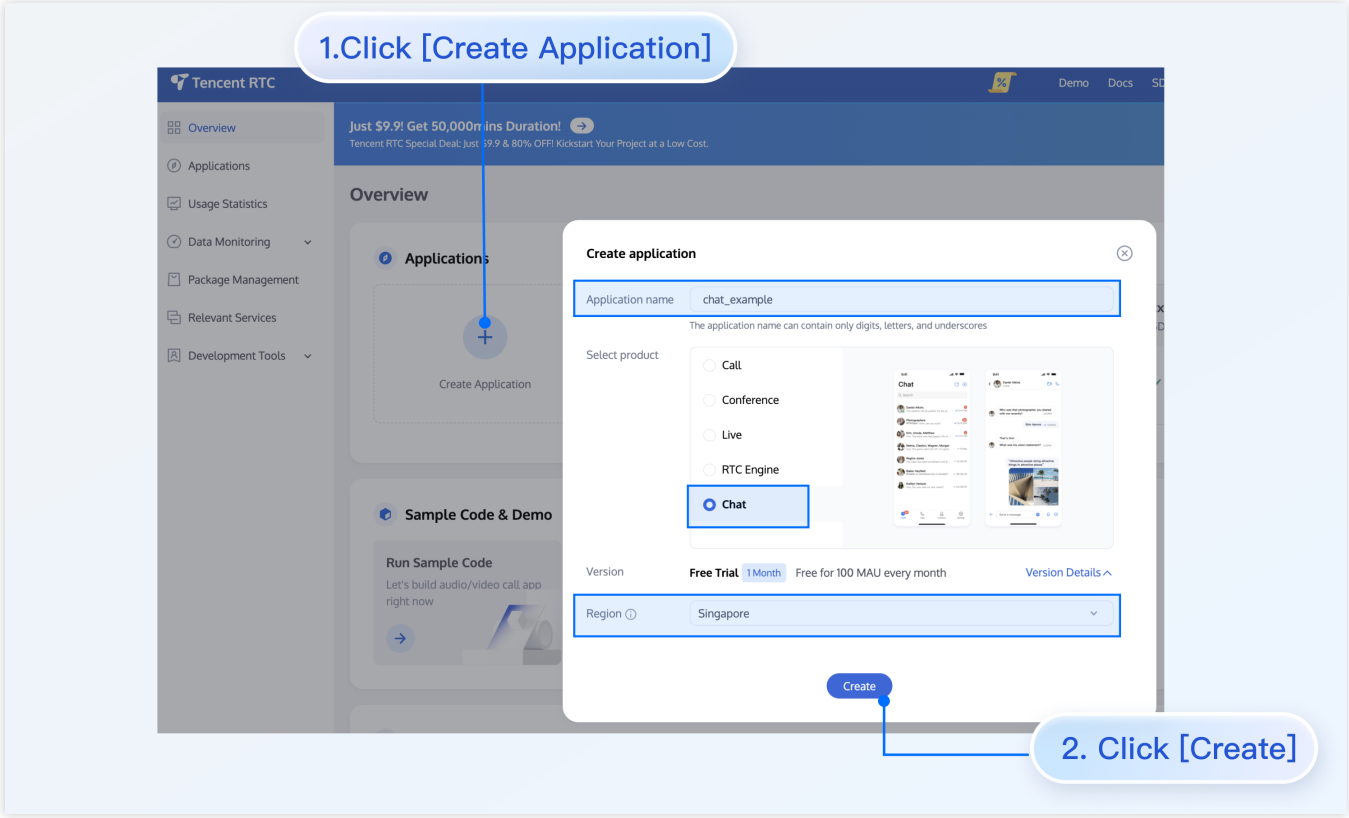
开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

创建应用

在集成 TUIKit 之前，您需要先去控制台创建一个新的 Chat 应用，步骤如下：
1. 注册控制台账号。
2. 进入 `Application`，单击 `Create application`，弹出 application 信息填写框。
3. 填写 Application name，选择 product 为 Chat，选择合适的 Region。
操作完成后，您会在 `My Applications` 列表中看到刚才创建的 application。

注意：
请记录下该 application 的 SDKAppID，后续步骤会使用到。另外，请严格保管好 SDKSecretKey，不要透露给无关人员。
操作步骤图示如下：



创建用户账号

创建 application 只能保证您可以正常初始化 SDK。如果要成功发消息，您还需要在 application 中创建用户账号。创建账号方式有很多，例如直接在控制台创建，或者通过 API 在客户端注册，您可以选择任意一种合适的方式。

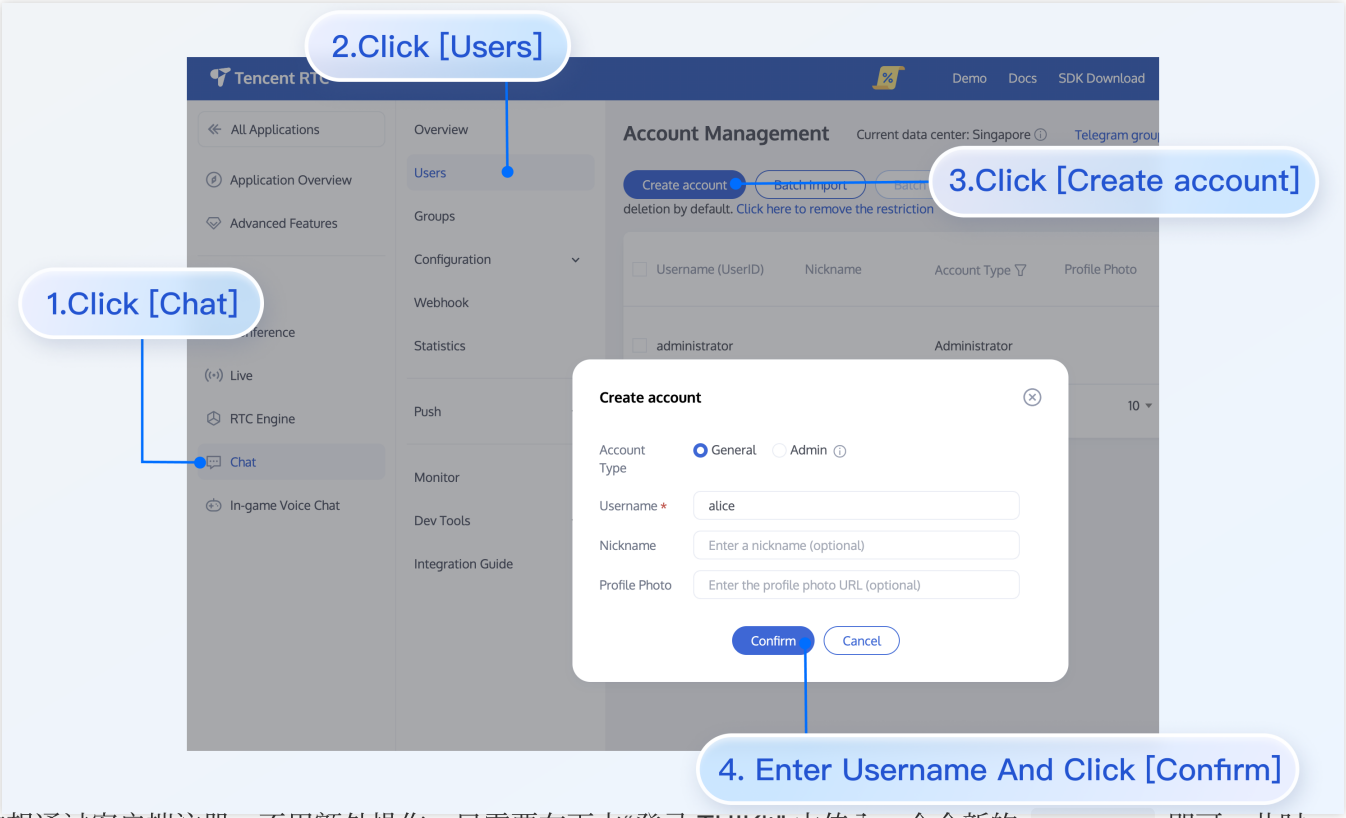
注意：

发消息至少是两个用户之间进行，因此您在此环节至少要创建 2 个账号。请记录下这 2 个账号的 userID，后续步骤会使用到。

如果您想在控制台创建，步骤如下：

1. 单击进入您上面创建的 application，会在左侧边栏看到 Chat 产品入口，单击进入。
2. 进入 Chat 产品子页面后，单击 `Users`，进入用户管理页面。
3. 单击 `Create account`，弹出创建账号信息填写框。如果只是普通成员，我们建议您选择 `General` 类型。虽然 `Nickname` 不是必填项，我们依然建议您设置。如果界面上不方便展示 `userID`，您可以通过 `Nickname` 识别出不同用户。

图示如下：



如果你想通过客户端注册，不用额外操作，只需要在下文“登录 TUIKit” 中传入一个全新的 `userID` 即可，此时 TUIKit 会自动为您注册该 `userID` 。

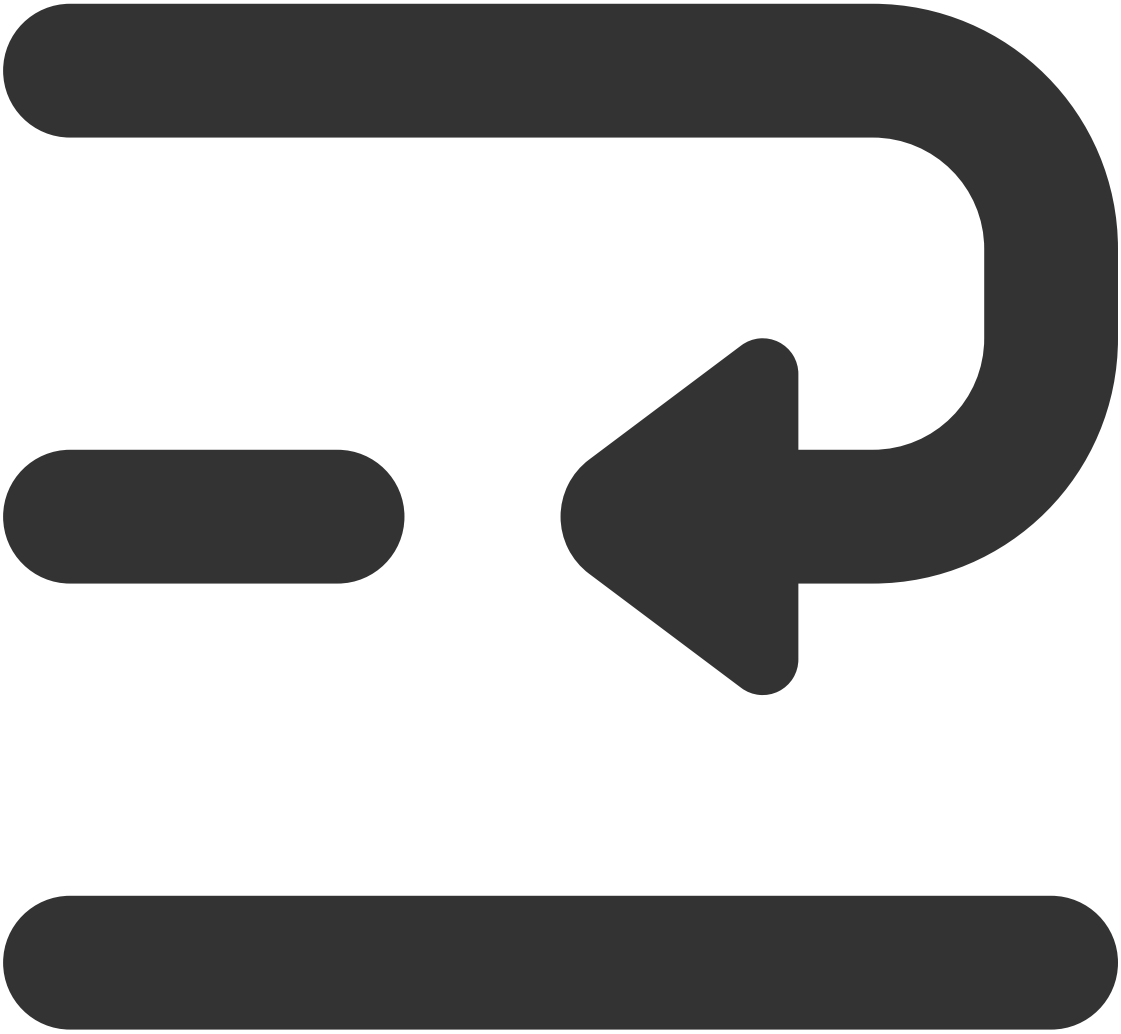
集成 TUIKit

聊天互动中发送消息的功能是由 `TUIChat` 实现的，您至少要集成 `TUIChat` 才能正常收发消息，其他的组件，例如 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 等，您可以按需集成。
如果您需要多个 UI 组件，可以集成 TUIKit，请参见文档：集成 TUIKit。
如果您只需要集成 TUIChat，请参见文档：仅集成聊天。

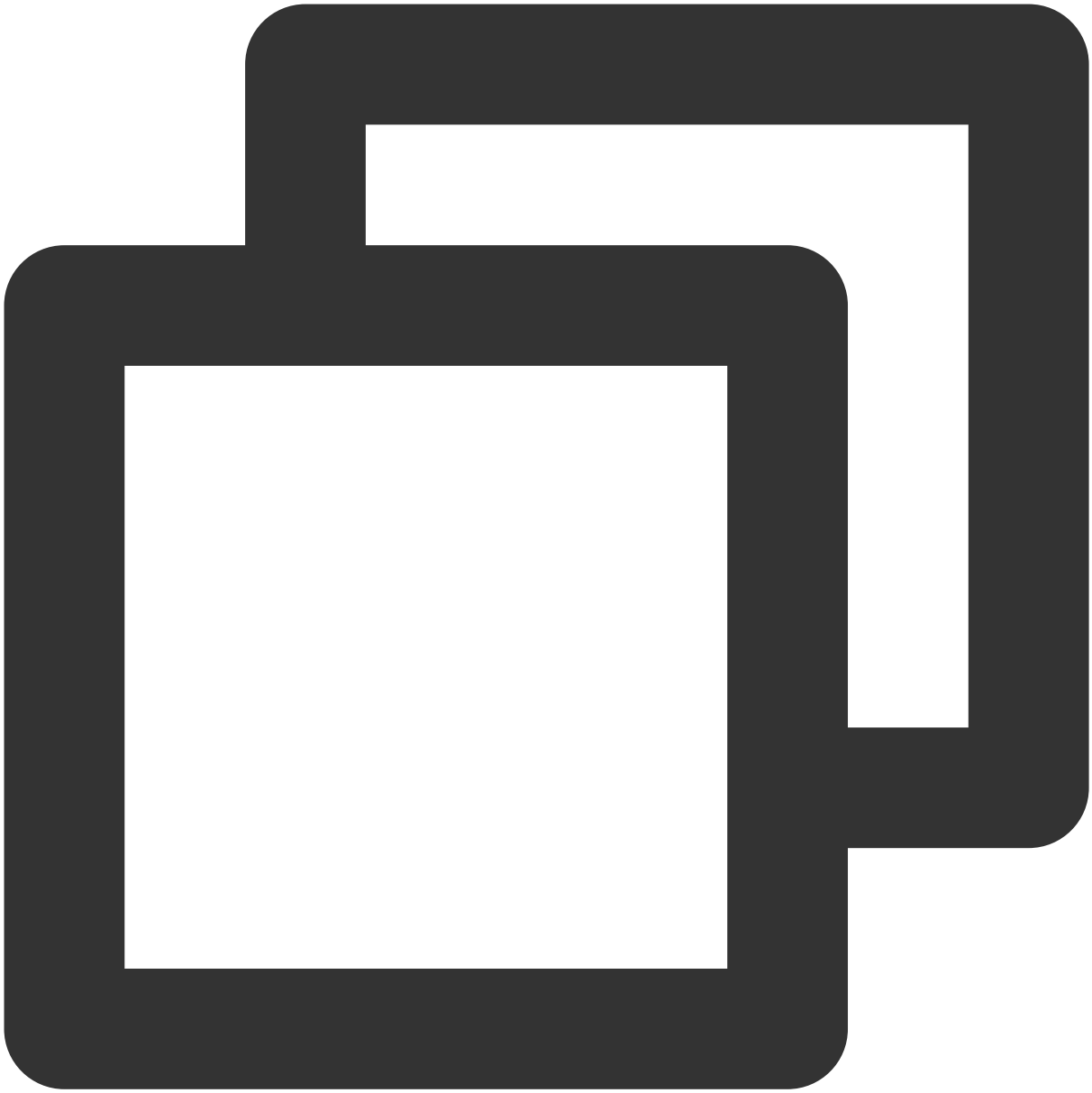
登录 TUIKit

使用 TUIKit 组件里的功能都需要登录，由 TUILogin 提供登录接口，如下：

版权所有：腾讯云计算（北京）有限责任公司



版权所有：腾讯云计算（北京）有限责任公司



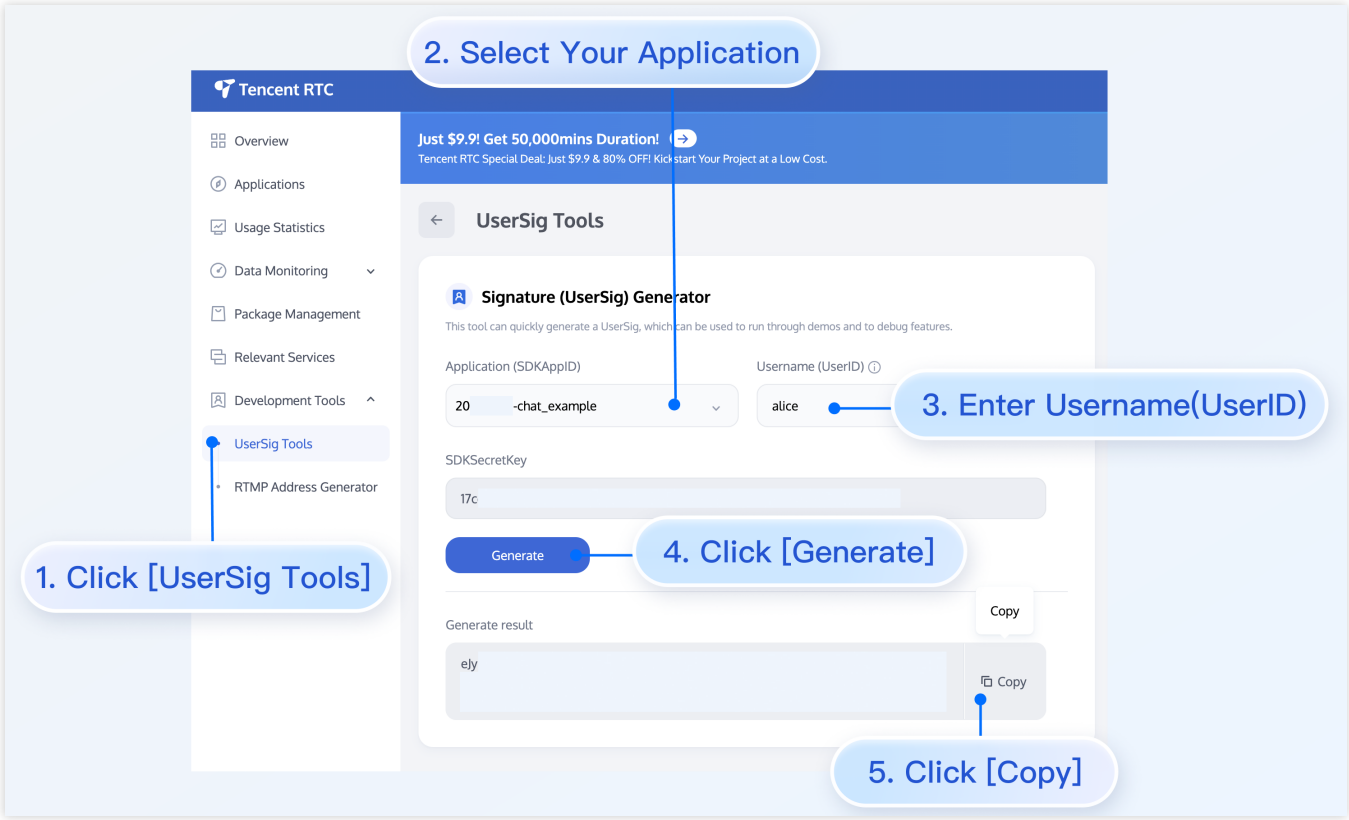
```
// API location: TUICore/TUILogin.h
+ (void)login:(int)sdkAppID userID:(NSString *)userID userSig:(NSString *)userSig s
```

该接口要求输入 3 个参数：

sdkAppID，新创建应用的 SDKAppID，已在上文步骤中获取到。

userID，user1 的 userID，已在上文步骤中获取到，注意不是用户的 NickName。

userSig，user1 的 userSig，可使用控制台提供的开发工具实时生成，路径：主页 > `Development Tools` > `UserSig Tools` > `Signature (UserSig) Generator` ，图示如下：



跳转聊天界面

为了实现发消息的目标，接下来要做的是：

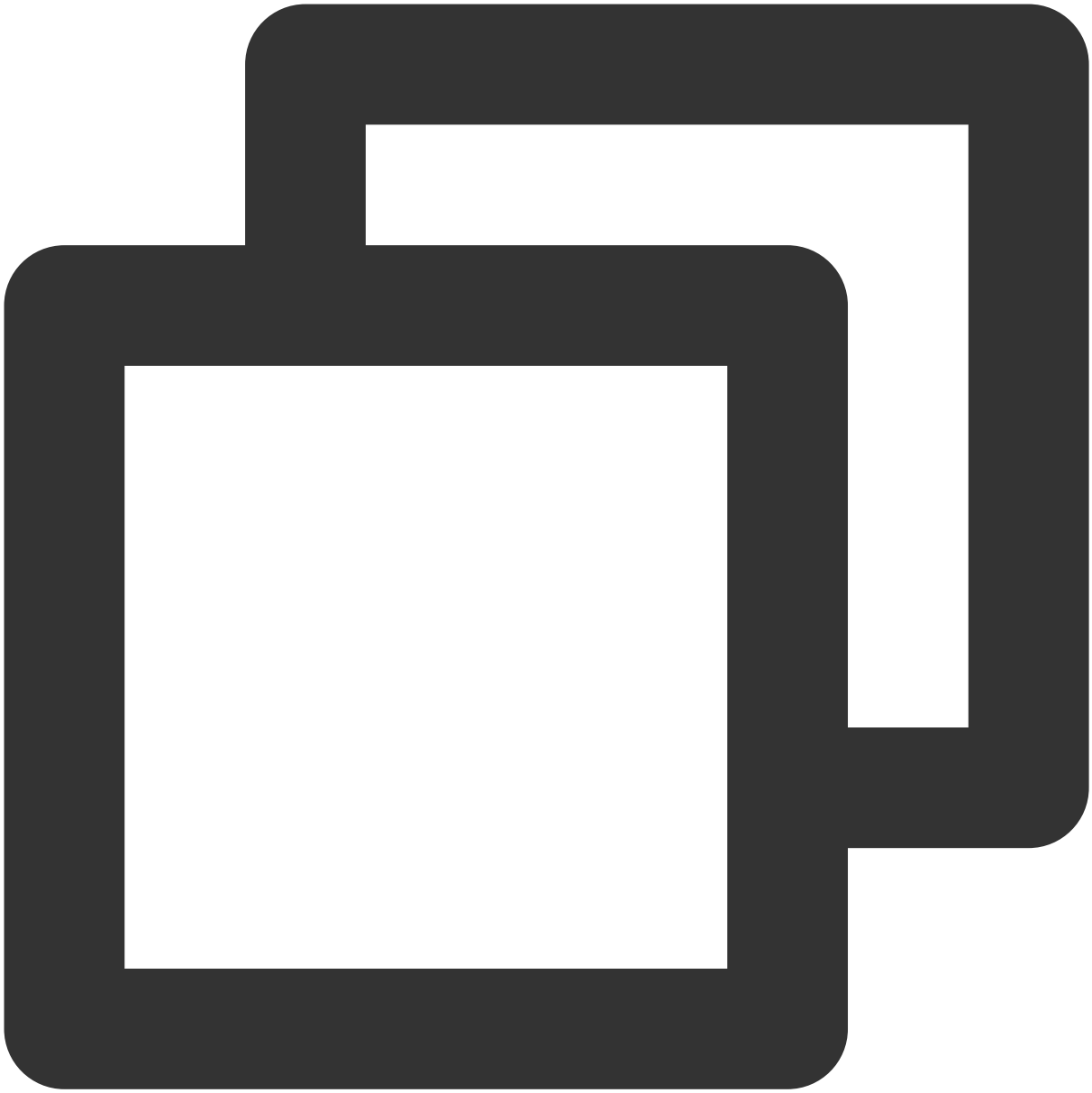
1. 使用上述注册的账号之一（下文称之为 user1）登录 TUIKit，此时 user1 上线了。
2. user1 给另一个账号（下文称之为 user2）发消息，user2 可以不登录，跟 user1 可以没有任何好友关系。

说明：

此处讲解的是登录 user1 后给 user2 发消息。如果您希望 user1 和 user2 能聊天互动，需要使用同样的步骤登录 user2 并进入与 user1 的聊天界面。

您可以在 user1 登录成功的回调里跳转或者嵌套聊天界面，就可以给 user2 发消息了。

示例代码如下，其中 userID 需要传入聊天对象 user2 的 userID。

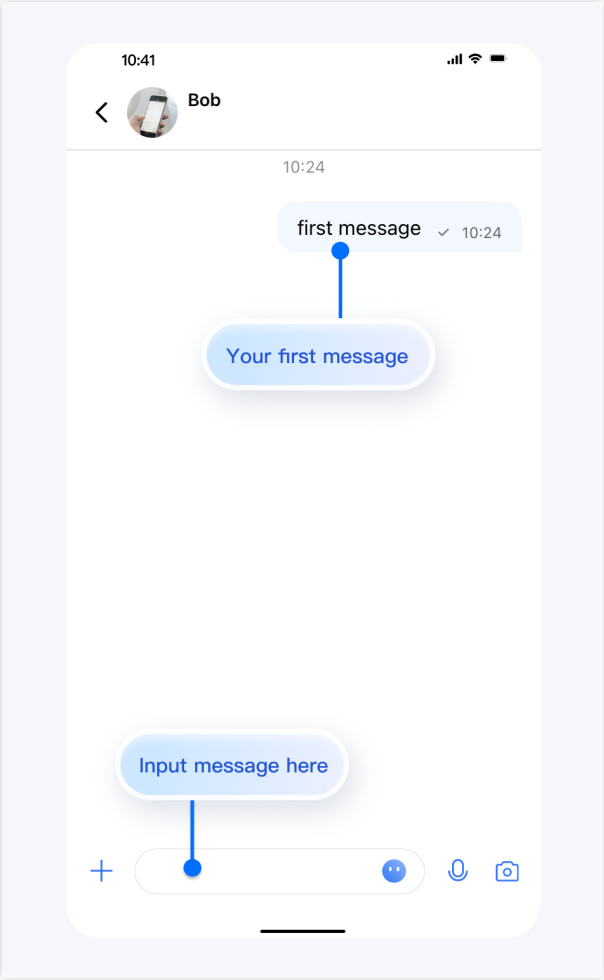


```
// Pass userID for 1v1 conversation.
- (void)pushToChatViewController:(NSString *)groupID userID:(NSString *)userID {
    // Create conversationData.
    TUIChatConversationModel *conversationData = [[TUIChatConversationModel alloc]
    conversationData.userID = userID;

    // Create c2c chatVC.
    TUIBaseChatViewController_Minimalist *chatVC = [[TUIC2CChatViewController_Minim
    chatVC.conversationData = conversationData;

    // Option 1: navigate to chatVC.
```

```
    [self.navigationController pushViewController:chatVC animated:YES];
    // Option 2: add chatVC as a childVC to your parent VC.
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}
```

发送第一条消息

操作完上述步骤，您可以跳转到如下的聊天界面。快点手动单击输入框，发送您的第一条消息吧：



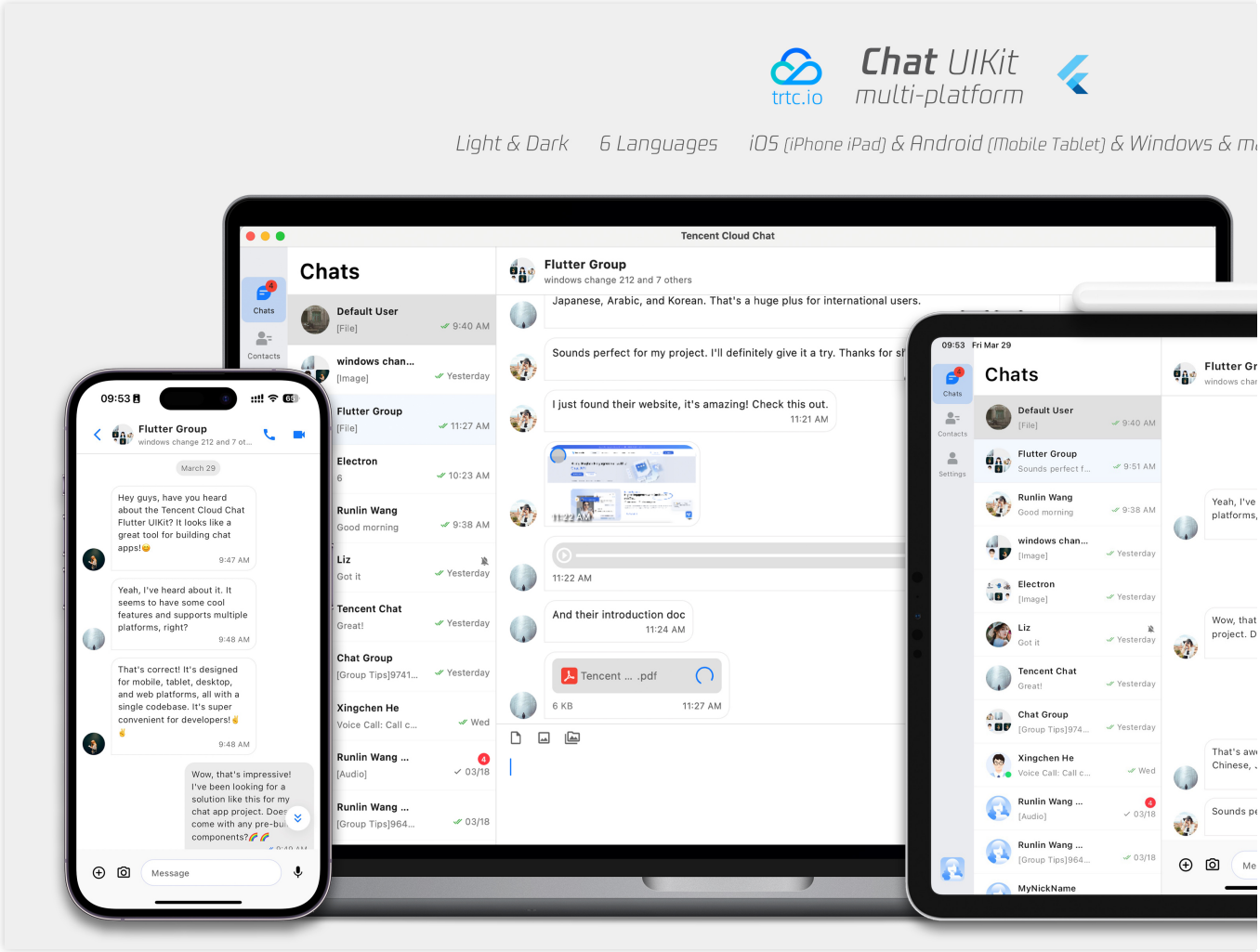
联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Flutter

最近更新时间：2024-07-30 15:48:43

Flutter Chat UIKit 旨在为开发者提供一整套工具，轻松创建功能丰富的聊天应用程序。
它采用模块化方法构建，允许您选择所需的组件，同时保持应用程序轻量级和高效。
UIKit 包括多种功能，例如会话列表、消息处理、联系人列表、用户和群组资料、搜索功能等。



特点

1. **个性化外观**：内置**深色和浅色**模式，UIKit 提供多种**主题和外观定制**选项，以满足您的业务需求。
2. **多平台兼容性**：适应性单一代码库确保了在不同平台上的兼容性，包括**移动**设备（iOS/Android）、**平板电脑**（iPad和Android平板电脑）、**Web**浏览器和**桌面**环境（Windows/macOS）。
3. **本地化支持**：开发时支持英语和其他语言选项，包括阿拉伯语、日语、韩语、简体中文和繁体中文。国际化功能确保本地化界面语言，并支持自定义和**补充语言**，阿拉伯语支持 **RTL** UI。

版权所有：腾讯云计算（北京）有限责任公司

4. **性能提升**：UIKit 提供了更好的消息列表**性能**、**内存使用**和精确的消息定位功能，适用于大量消息和导航到旧消息的场景。
5. **高级功能**：UIKit 拥有众多高级功能，包括连续语音消息播放、增强的多媒体和文件消息体验以及直观的左右滑动多媒体消息预览。
6. **优化用户体验**：丰富的**动画**、**触觉反馈**和**精美的界面**等细节优化有助于提升用户体验。新功能例如网格样式头像、重新设计的转发面板、群组成员选择器和重新设计的长按消息菜单进一步丰富了体验。
7. **模块化设计**：组件按模块化包组织，允许选择性导入并减少不必要的膨胀。每个包支持**内置导航过渡**，通过自动处理过渡（例如会话和消息之间的过渡）简化开发和集成。
8. **开发者友好的方法**：更统一、标准化的组件参数设计，更清晰的代码命名规范和详细的注释，结合选择**全局或实例级配置**管理的灵活性，使开发更加简单高效。

快速开始

要求

Flutter 版本：3.19或更高
Dart 版本：3.0或更高

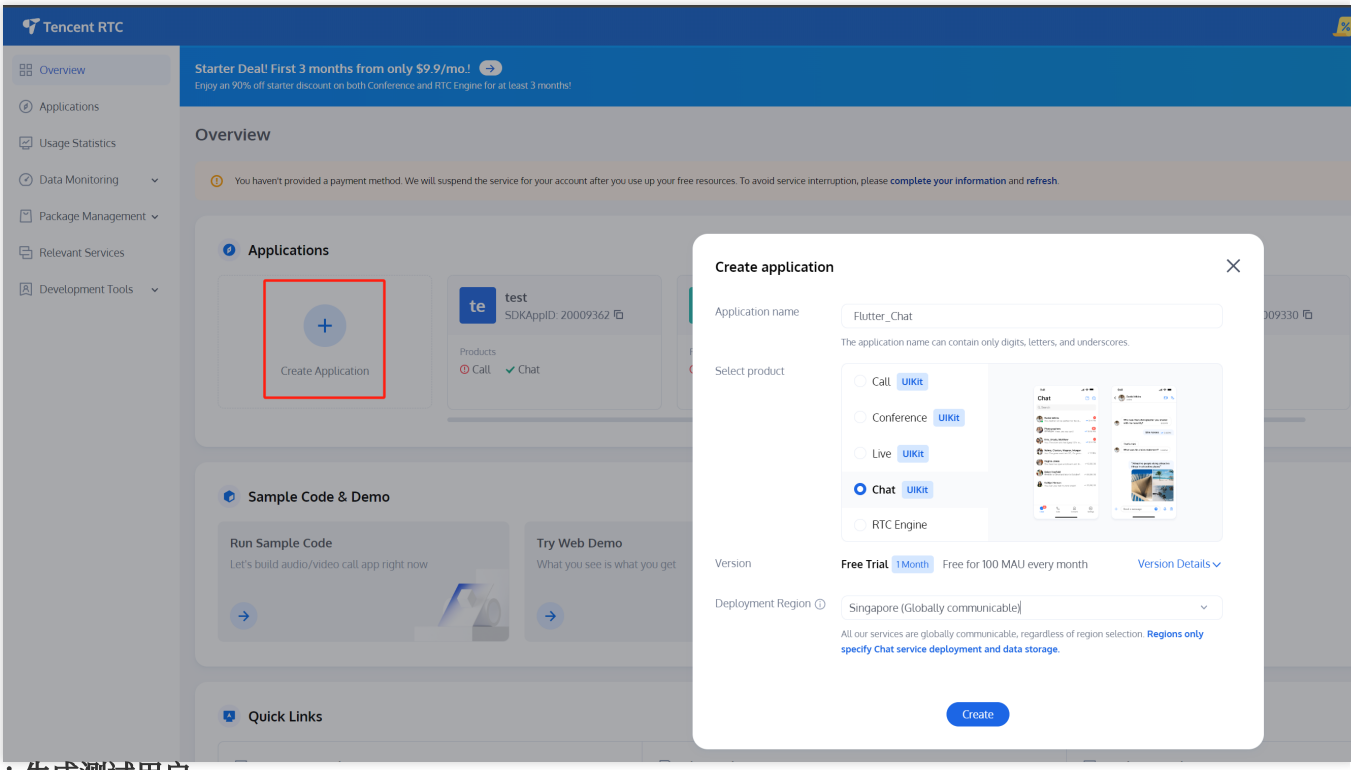
在控制台中设置应用程序

步骤1：创建账户

访问 控制台 并按照提示创建账户。

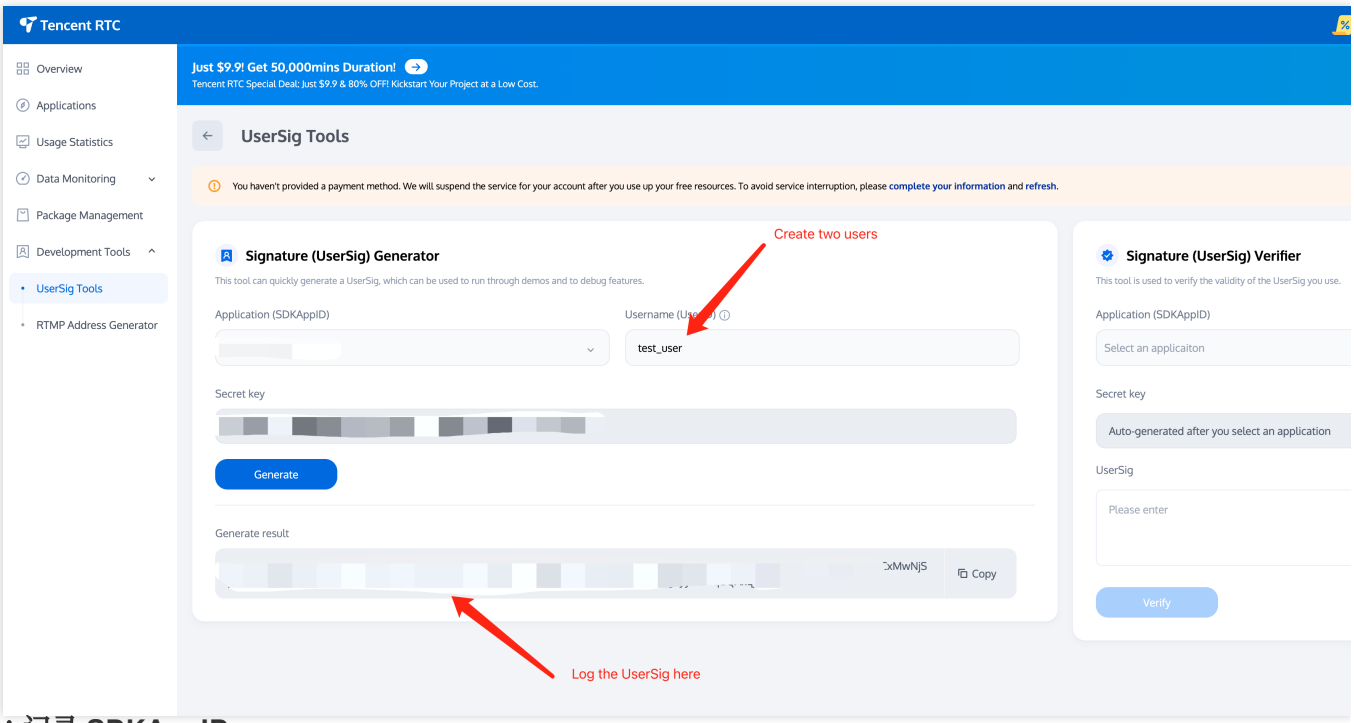
步骤2：开始免费试用

在首页上创建一个应用程序并开始免费试用。



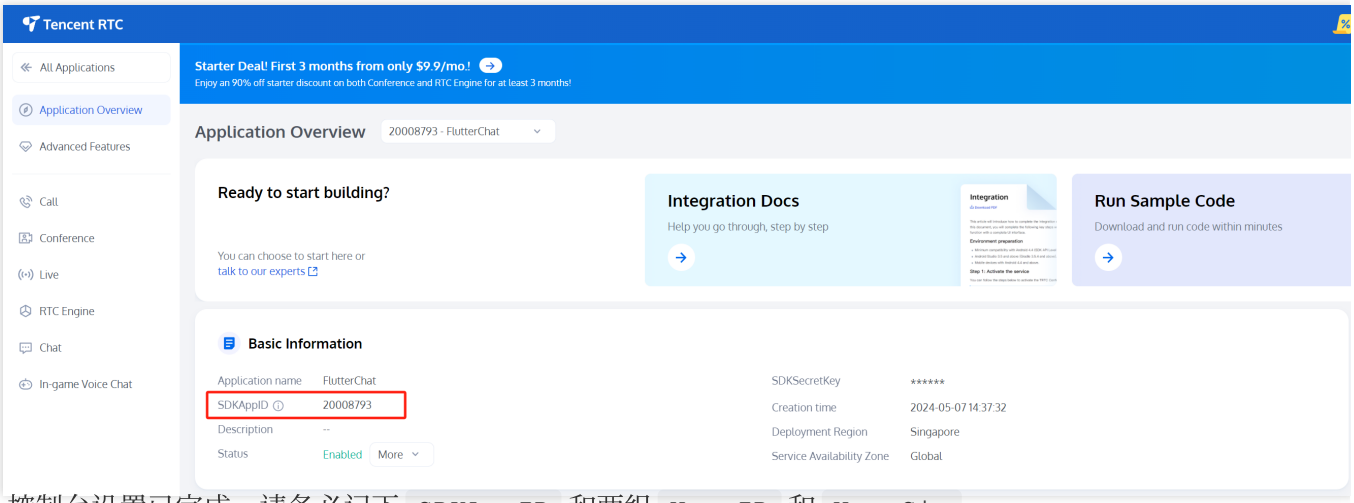
步骤3：生成测试用户

在 账户管理上创建两个用户（测试帐户）。然后使用 UserSig 工具为它们创建相应的 UserSigs，记下 UserSigs 以备后用。



步骤3：记录 SDKAppID

转到应用程序，选择您刚刚创建的应用程序，然后导航到相应的 应用程序概述 以找到您的 SDKAppID。



此时，控制台设置已完成。请务必记下 `SDKAppID` 和两组 `UserID` 和 `UserSig` 。

编码

说明：

本指南仅提供与Flutter Chat UIKit集成的**最简化概述**。

有关详细完整的集成过程，请参阅本指南：详细集成指南。

首先，最好准备好一个Flutter项目，或者创建一个新的项目以充分体验本教程。我们建议按照创建新的Flutter项目的步骤进行操作。

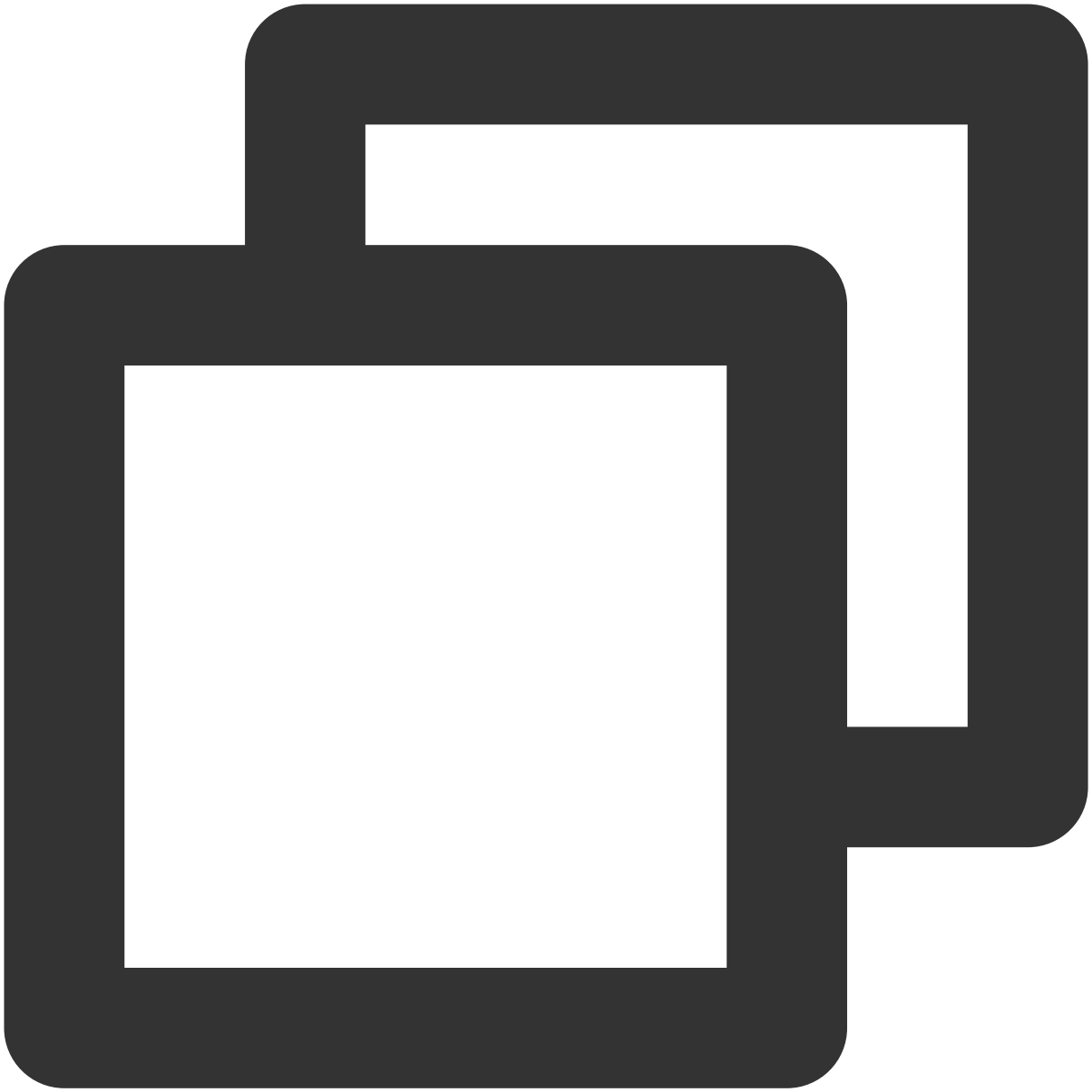
此外，接下来的步骤涉及客户端项目和代码操作。为了增强您的理解，您可以参考稍后展示的简化集成项目的源代码，该项目可在GitHub仓库上找到。

注意：此源代码仓库仅用于展示简化集成目的，供以下教程使用。

如果您对探索功能齐全、具有广泛功能、高级功能和定制选项的应用程序感兴趣，请查看此仓库。

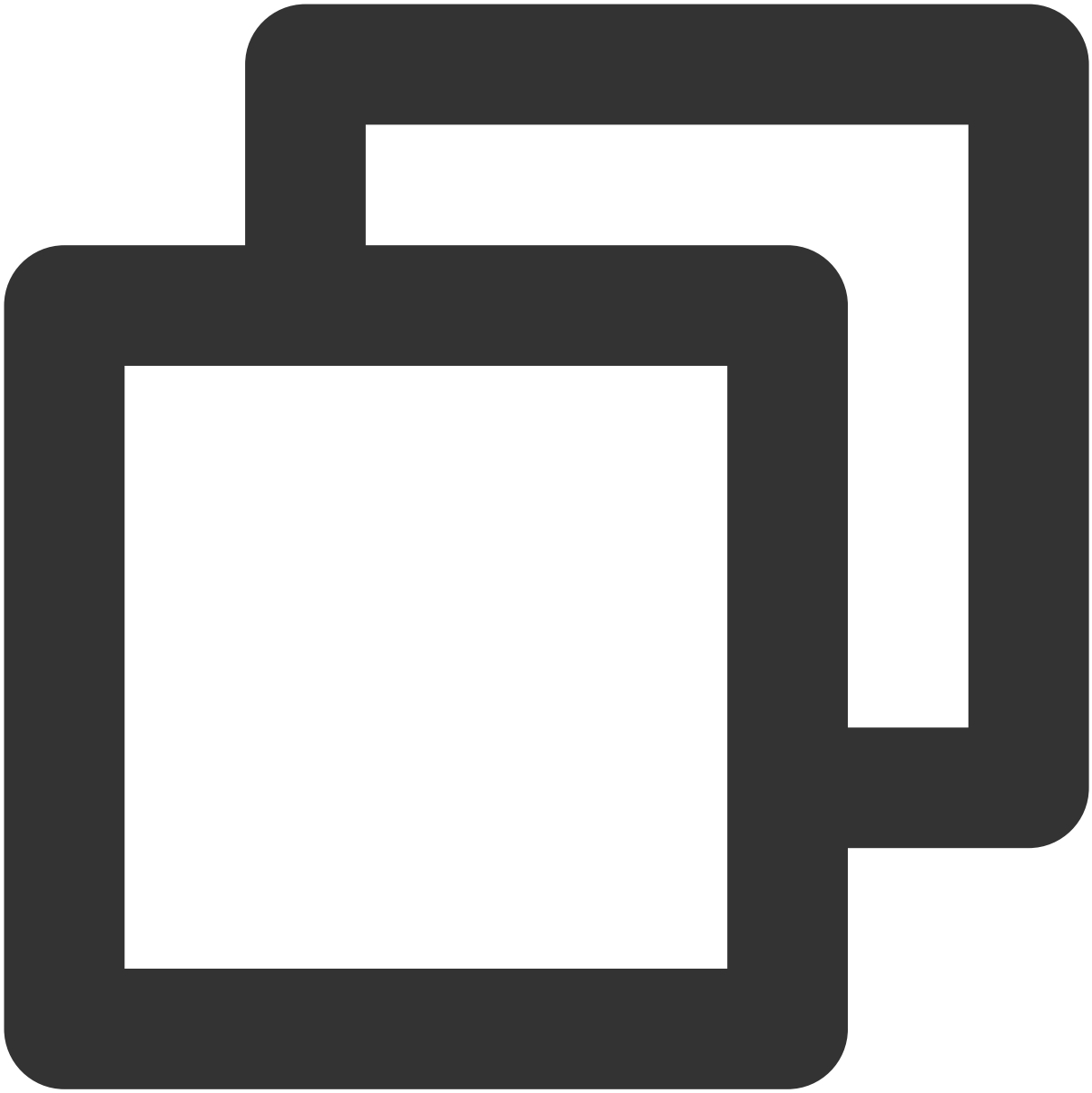
步骤1. 导入包

首先，导入基础包tencent_cloud_chat。



```
flutter pub add tencent_cloud_chat
```

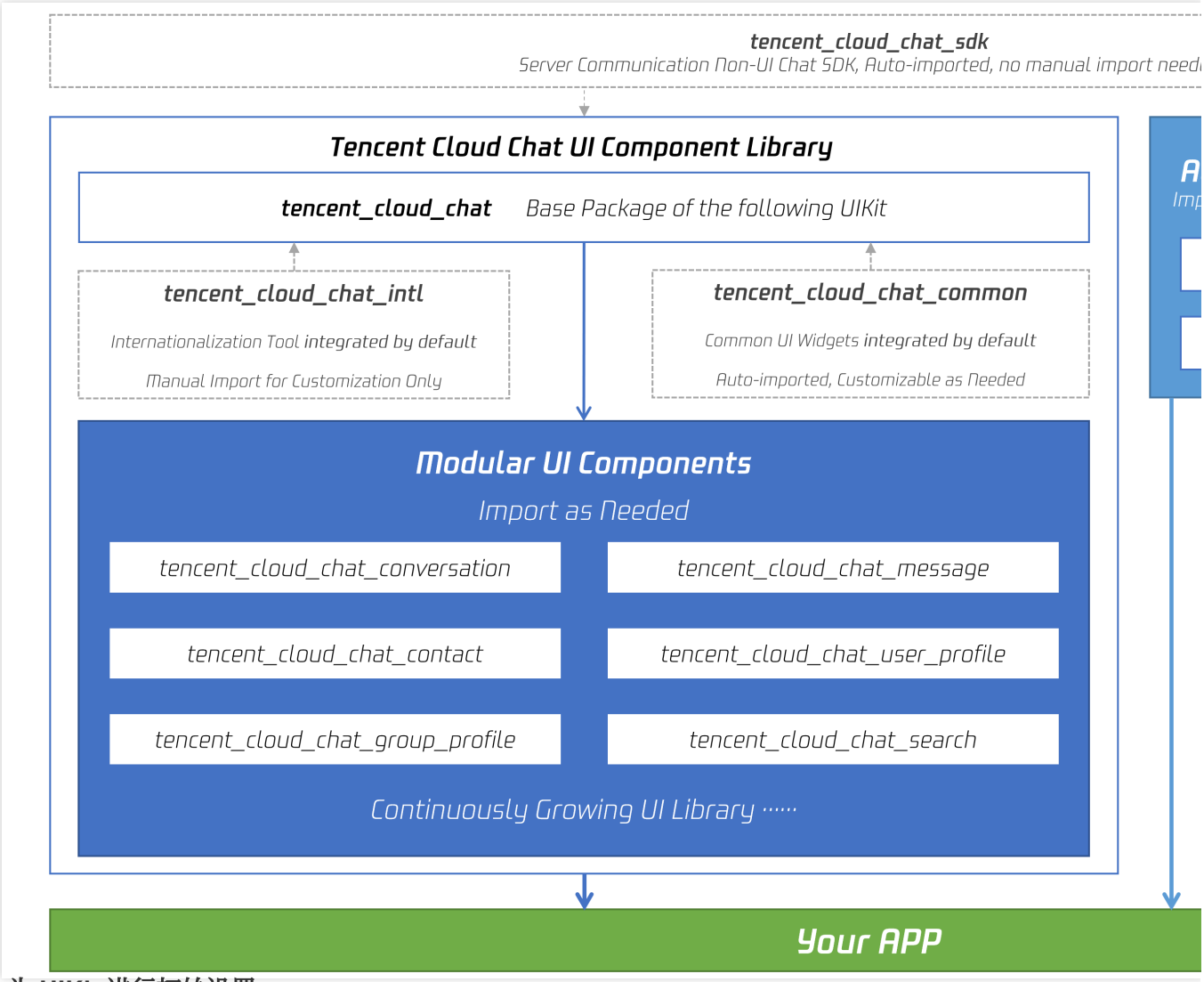
接下来，根据您的需求导入 UI 组件包：



```
flutter pub add tencent_cloud_chat_message
flutter pub add tencent_cloud_chat_conversation
flutter pub add tencent_cloud_chat_contact
flutter pub add tencent_cloud_chat_user_profile
flutter pub add tencent_cloud_chat_group_profile
```

为了演示目的，我们建议导入所有组件包。然而，在实际项目中，您可以根据具体需求导入包。

Flutter Chat UIKit 的架构如下所示：



步骤2. 为 UIKit 进行初始设置

在开始使用每个模块化包 UI 组件之前，请按照以下初始设置步骤操作：

全局配置

用 `TencentCloudChatMaterialApp` 替换项目中的 `MaterialApp` 。

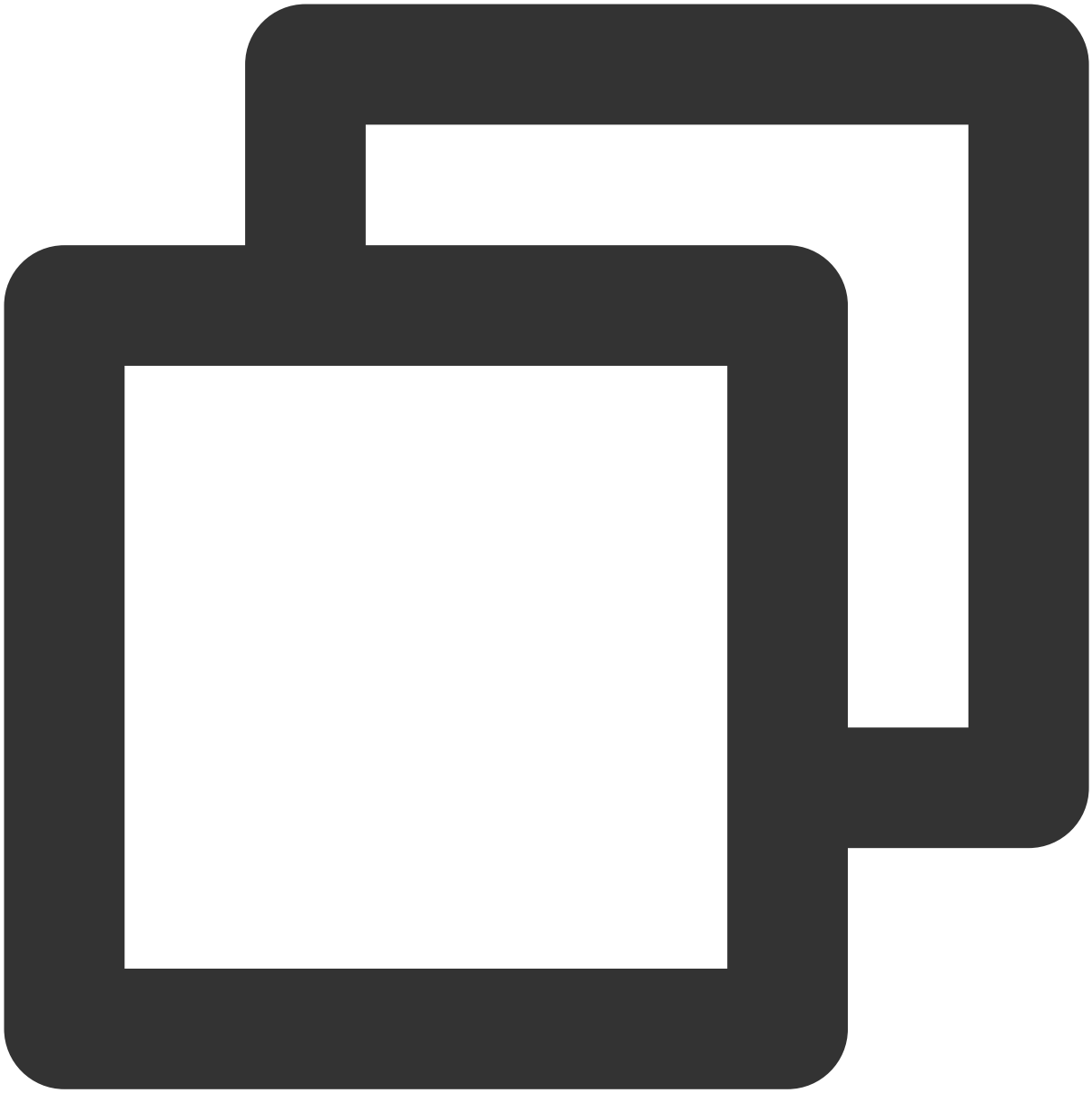
这可以自动管理和配置语言、主题（包括 material3）、主题模式和其他设置。如果您喜欢手动配置，请参见 手动实现UIKit的全局配置。

初始化和登录

调用 `TencentCloudChat.controller.initUIKit` 进行初始化和登录。

传入您的腾讯云聊天应用的 `sdkAppID` 、 `userID` 和 `userSig` 。同时，在 `usedComponentsRegister` 列表中声明每个子模块 UI 包的注册。

版权所有：腾讯云计算（北京）有限责任公司



```
TencentCloudChat.controller.initUIKit(
  options: const TencentCloudChatInitOptions(
    sdkAppID: , /// [必需]：您的腾讯云聊天应用的SDKAppID
    userID: , /// [必需]：已登录用户的userID
    userSig: , /// [必需]：已登录用户的userSig
  ),
  components: const TencentCloudChatInitComponentsRelated( /// [必需]：模块化UI组件相关
    usedComponentsRegister: [
      /// [必需]：聊天UIKit中使用的组件的注册函数列表。
          TencentCloudChatConversationManager.register,
          TencentCloudChatMessageManager.register,
```

```
            TencentCloudChatUserProfileManager.register,
            TencentCloudChatGroupProfileManager.register,
            TencentCloudChatContactManager.register,
        ],
    ),
);
```

全局配置完成后，我们现在可以深入了解模块化 UI 组件的使用。让我们探讨它们如何增强您的聊天应用程序体验。

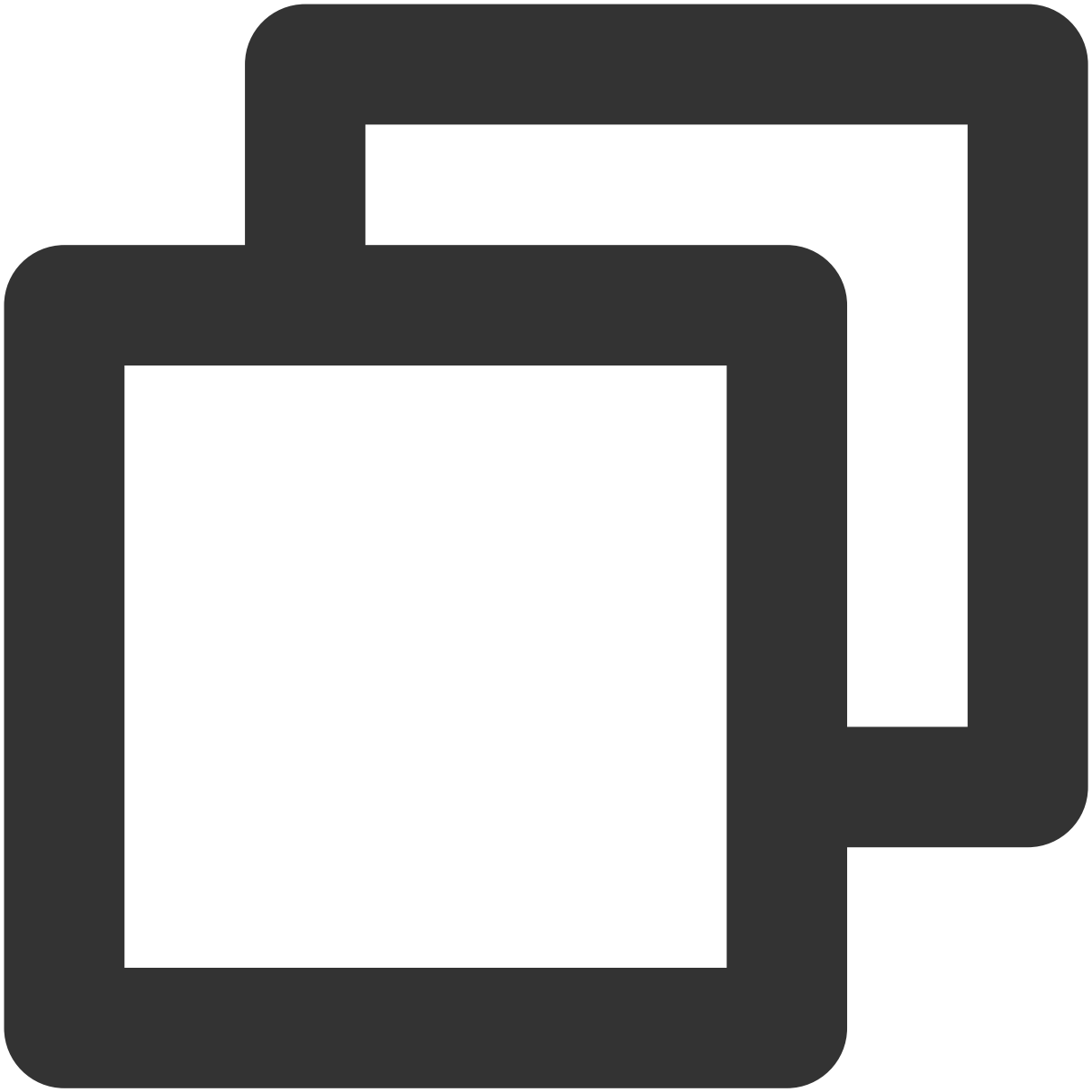
步骤3. 集成模块化UI组件

在大多数使用场景中，您需要手动实例化并将 `TencentCloudChatConversation` 和 `TencentCloudChatContact` 组件添加到一个小部件中（如果需要）。

其他组件会根据用户操作自动导航。

在本教程中，我们将使用 `bottomNavigationBar` 来管理页面，并在 `TencentCloudChatConversation` 和 `TencentCloudChatContact` 组件之间切换。

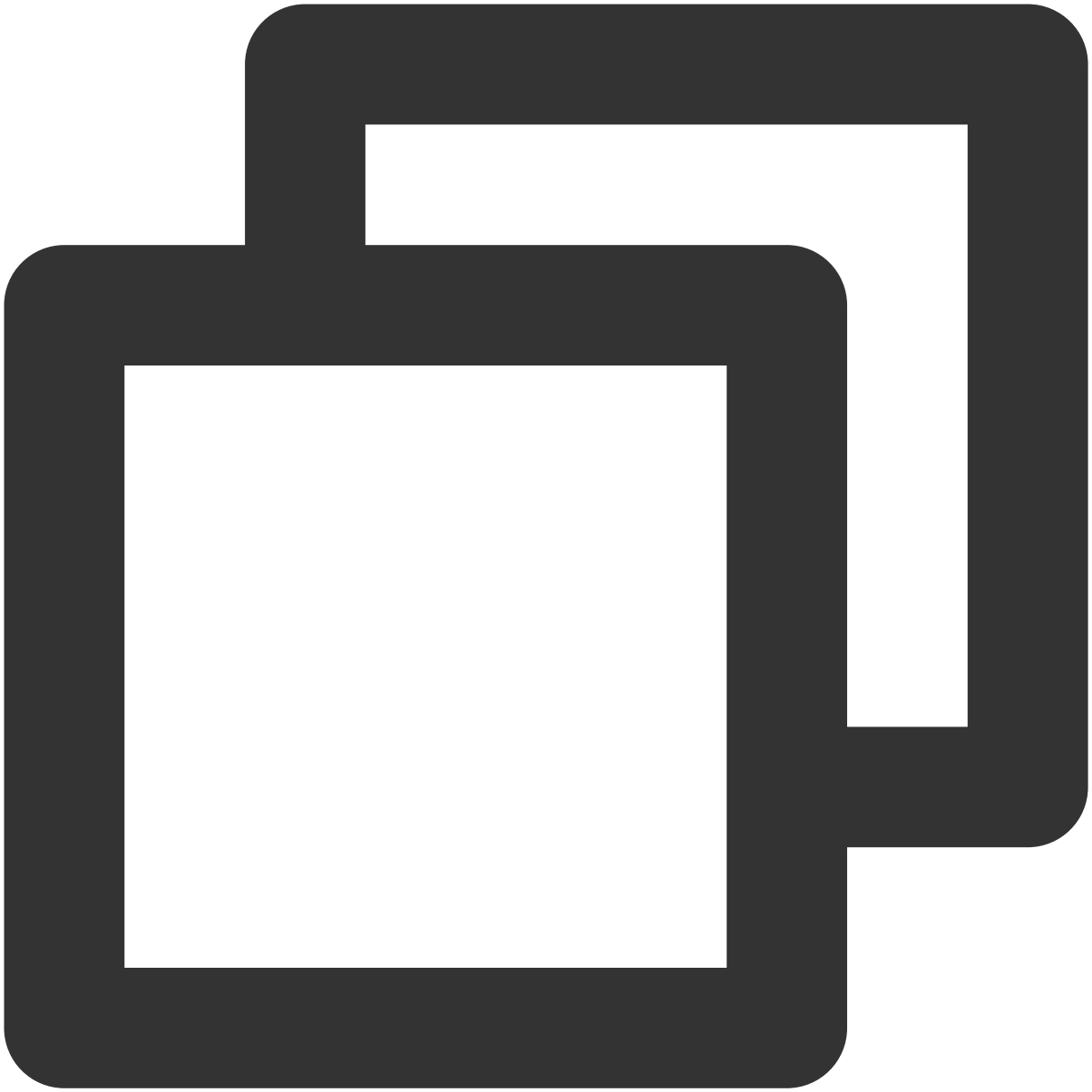
首先，声明一个 `currentIndex` 变量和一个 `List<Widget> pages` 数组，以表示当前选择的组件并存储组件实例。



```
List<Widget> pages = [];
int currentIndex = 0;
```

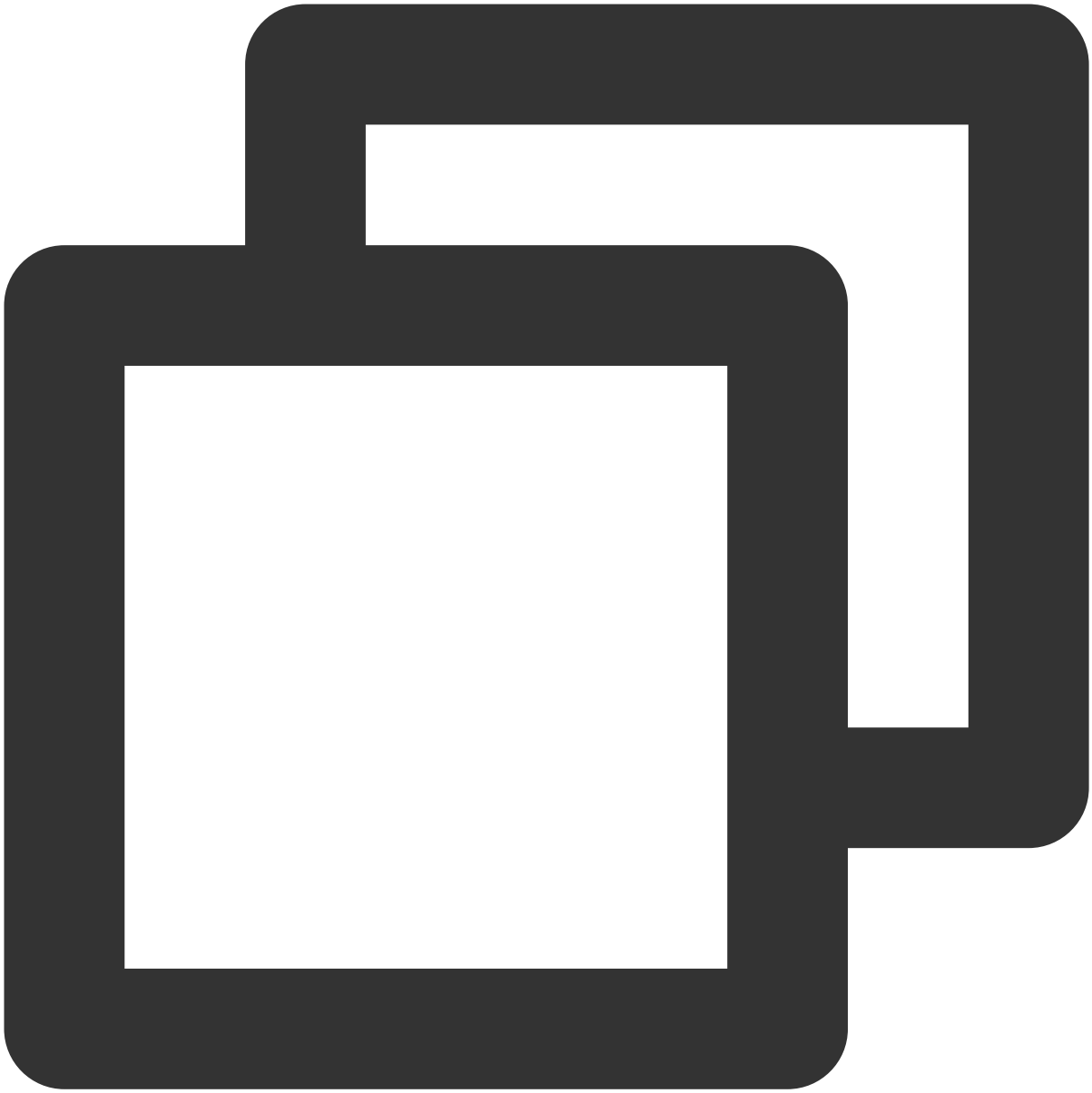
将实例存储在 `pages` 数组中。

版权所有：腾讯云计算（北京）有限责任公司



```
pages = [
      const TencentCloudChatConversation(),
      const TencentCloudChatContact(),
    ];
```

最后，修改 `build` 方法如下：



```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      bottomNavigationBar: BottomNavigationBar(
        type: BottomNavigationBarType.fixed,
        currentIndex: currentIndex,
        onTap: (index) async {
          if (index != currentIndex) {
            setState(
                  () {
                currentIndex = index;
```

```
                },
              );
            }
          },
          items: const [
            BottomNavigationBarItem(
                icon: Icon(Icons.chat_bubble_outline), label: "Chats"),
            BottomNavigationBarItem(
                icon: Icon(Icons.contacts), label: "Contacts"),
          ],
        ),
        body: pages[currentIndex],
      );
    }
```

就是这样！您已成功集成了组件。

回顾一下，您可以在 GitHub 仓库 上查看简化集成代码。

步骤4. 体验 Flutter Chat UIKit 的实际效果

现在，让我们运行项目并体验 Flutter Chat UIKit。

使用在 `initUIKit` 方法中创建的第一个测试账户登录并启动应用。

首先运行 `flutter run` 。

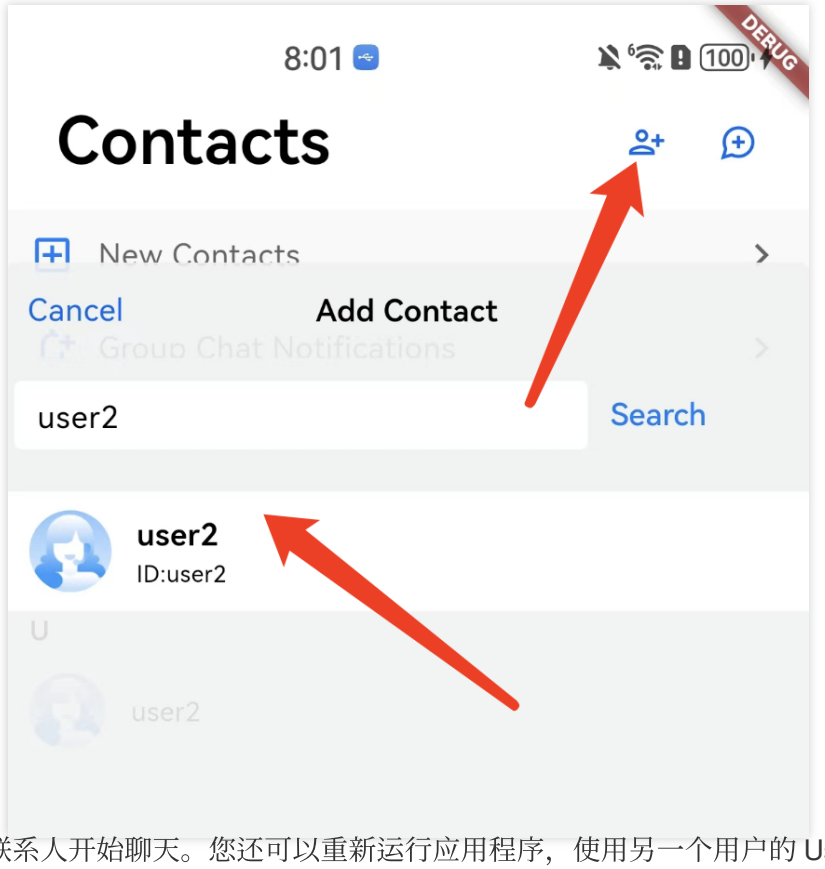
注意：

如果您遇到 iOS 无法运行 或 Android SDK 版本不匹配 的问题，请参考我们的 示例应用程序仓库 并选择合适的版本配置。

成功进入应用程序后，您将看到会话和联系人页面，底部可以在它们之间切换。

然而，还没有对话可以测试。

别担心！切换到联系人页面，点击右上角的“添加联系人”，然后将另一个测试账户添加为联系人。您现在可以在联系人列表中看到另一个账户。



点击联系人开始聊天。您还可以重新运行应用程序，使用另一个用户的 UserID 登录，并体验互相发送消息。

总之，我们现在已经完成了整个简化集成过程。感谢您体验腾讯云 Flutter Chat UIKit 的强大功能。

有关详细集成、配置和高级用法的更多信息，请参见本指南：详细集成指南。

集成 TUIKit

Android

最近更新时间：2024-08-14 10:31:18

本文将介绍如何集成 `TUIKit` 组件。

说明：

从 5.7.1435 版本开始，TUIKit 支持模块化集成，支持了经典版 UI，您可以根据自己的需求集成所需模块。

从 6.9.3557 版本开始，TUIKit 新增了全新的简约版 UI。

您可以根据需求自由选择经典版或简约版 UI 组件。如果您还不了解各个界面库的效果，可以查阅文档 TUIKit 界面库介绍。

开发环境要求

Android Studio-Giraffe

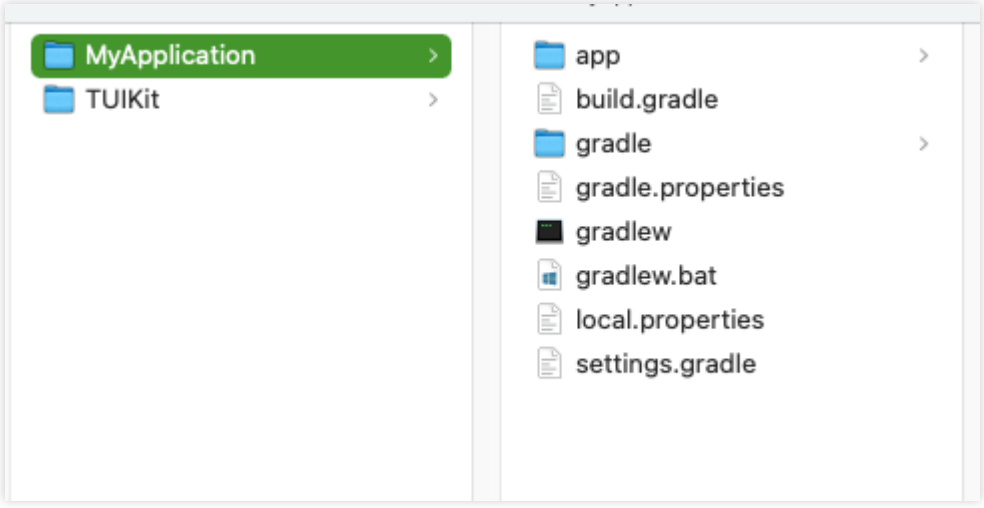
Gradle-7.2

Android Gradle Plugin Version-7.0.0

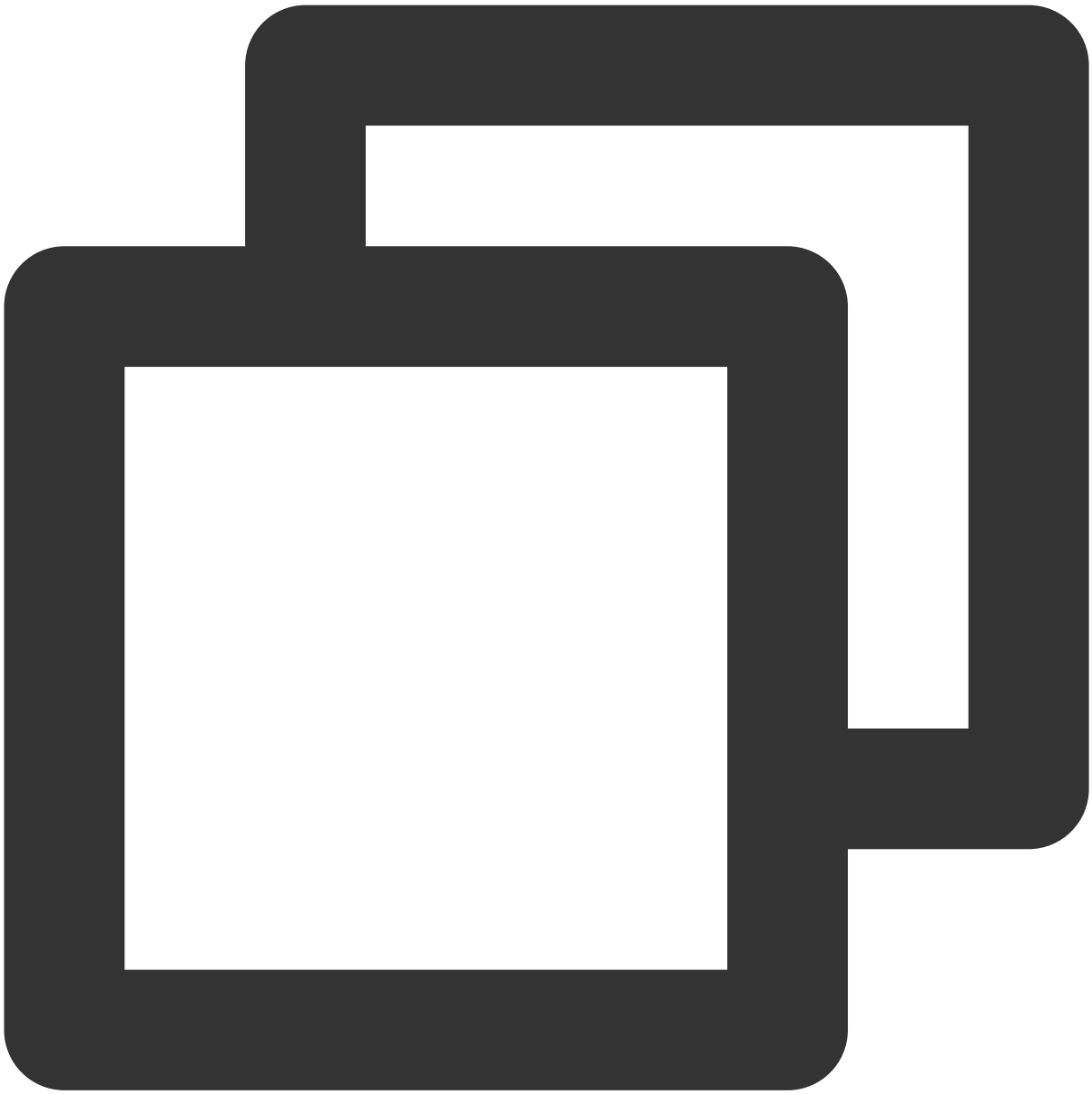
kotlin-gradle-plugin-1.5.31

源码集成

1. 从 GitHub 下载 TUIKit 源码。使 TUIKit 文件夹跟自己的工程文件夹同级，例如：



2. 根据实际业务需求在 settings.gradle 中添加对应的 TUI 组件。TUI 组件之间相互独立，添加或删除均不影响工程编译。



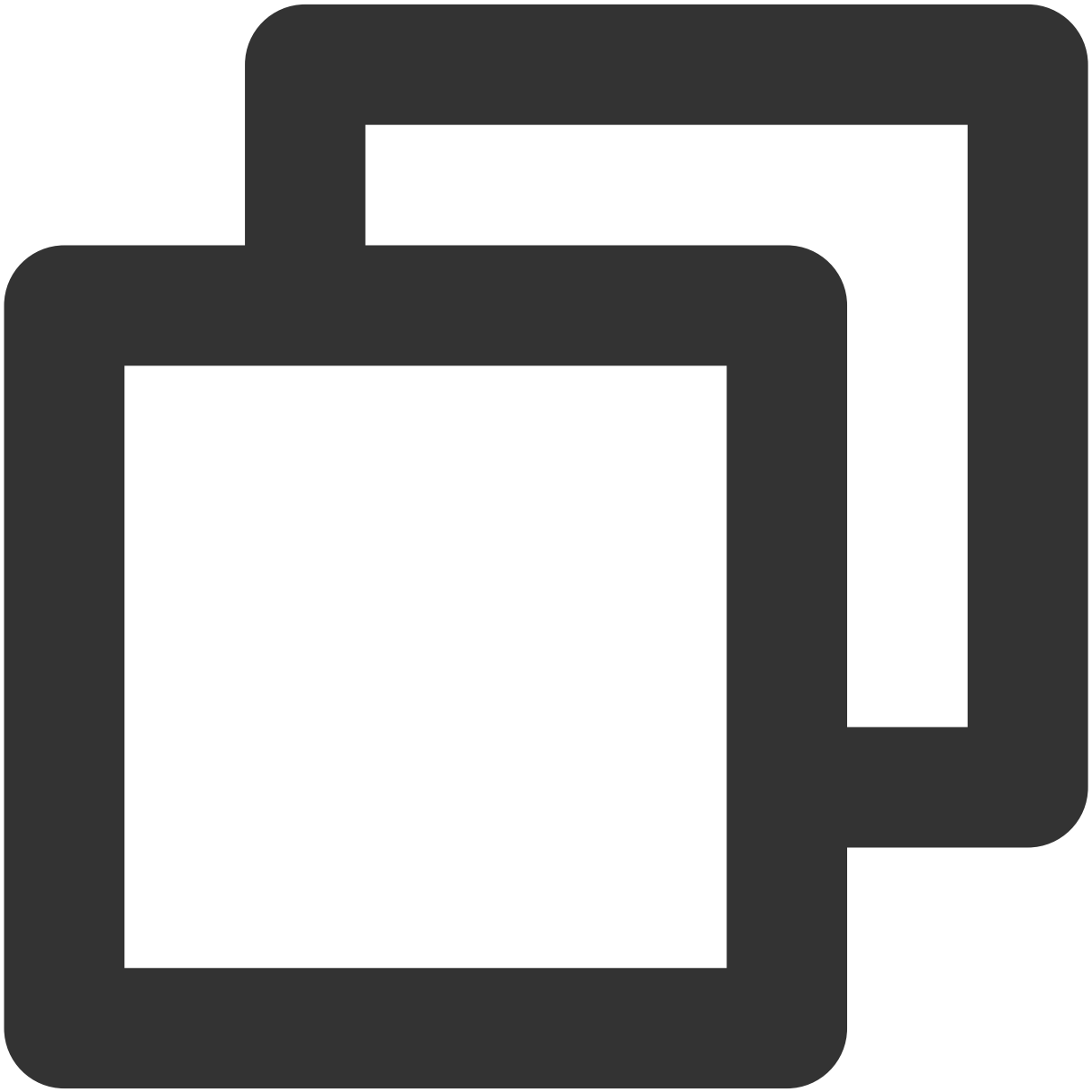
```
// Include the upper-layer app module
include ':app'

// Include the internal communication module (required module)
include ':tuicore'
project(':tuicore').projectDir = new File(settingsDir, '../TUIKit/TUICore/tuicore')

// Include the common module of IM component (required module)
```

```
include ':timcommon'
project(':timcommon').projectDir = new File(settingsDir, '../TUIKit/TIMCommon/timco

// Include the chat feature module (basic feature module)
include ':tuichat'
project(':tuichat').projectDir = new File(settingsDir, '../TUIKit/TUIChat/tuichat')

// Include the relationship chain feature module (basic feature module)
include ':tuicontact'
project(':tuicontact').projectDir = new File(settingsDir, '../TUIKit/TUIContact/tui

// Include the conversation list feature module (basic feature module)
include ':tuiconversation'
project(':tuiconversation').projectDir = new File(settingsDir, '../TUIKit/TUIConver

// Include the search feature module (To use this module, you need to purchase the
include ':tuisearch'
project(':tuisearch').projectDir = new File(settingsDir, '../TUIKit/TUISearch/tuise

// Include the group feature module
include ':tuigroup'
project(':tuigroup').projectDir = new File(settingsDir, '../TUIKit/TUIGroup/tuigrou

// Include the community topic feature module (To use this module, you need to purc
include ':tuicommunity'
project(':tuicommunity').projectDir = new File(settingsDir, '../TUIKit/TUICommunity

// Include the audio/video call feature module
include ':tuicallkit'
project(':tuicallkit').projectDir = new File(settingsDir, '../TUIKit/TUICallKit/tui

// Include the video conference module
include ':tuiroomkit'
project(':tuiroomkit').projectDir = new File(settingsDir, '../TUIKit/TUIRoomKit/tui

// Include speech-to-text plugin, supported from version 7.5
include ':tuivoicetotextplugin'
project(':tuivoicetotextplugin').projectDir = new File(settingsDir, '../TUIKit/TUIV

// Include customer service plugin, supported from version 7.6
include ':tuicustomerserviceplugin'
project(':tuicustomerserviceplugin').projectDir = new File(settingsDir, '../TUIKit/

// Include chatbot plugin, supported from version 7.7
include ':tuichatbotplugin'
project(':tuichatbotplugin').projectDir = new File(settingsDir, '../TUIKit/TUIChatB
```

版权所有：腾讯云计算（北京）有限责任公司

```
// Include chat message translation plugin, supported from version 7.2 (Value-added
include ':tuitranslationplugin'
project(':tuitranslationplugin').projectDir = new File(settingsDir, '../TUIKit/TUIT

// Include emoji reaction plugin, supported from version 7.8 (To use this module, y
include ':tuiemojiplugin'
project(':tuiemojiplugin').projectDir = new File(settingsDir, '../TUIKit/TUIEmojiPl
```

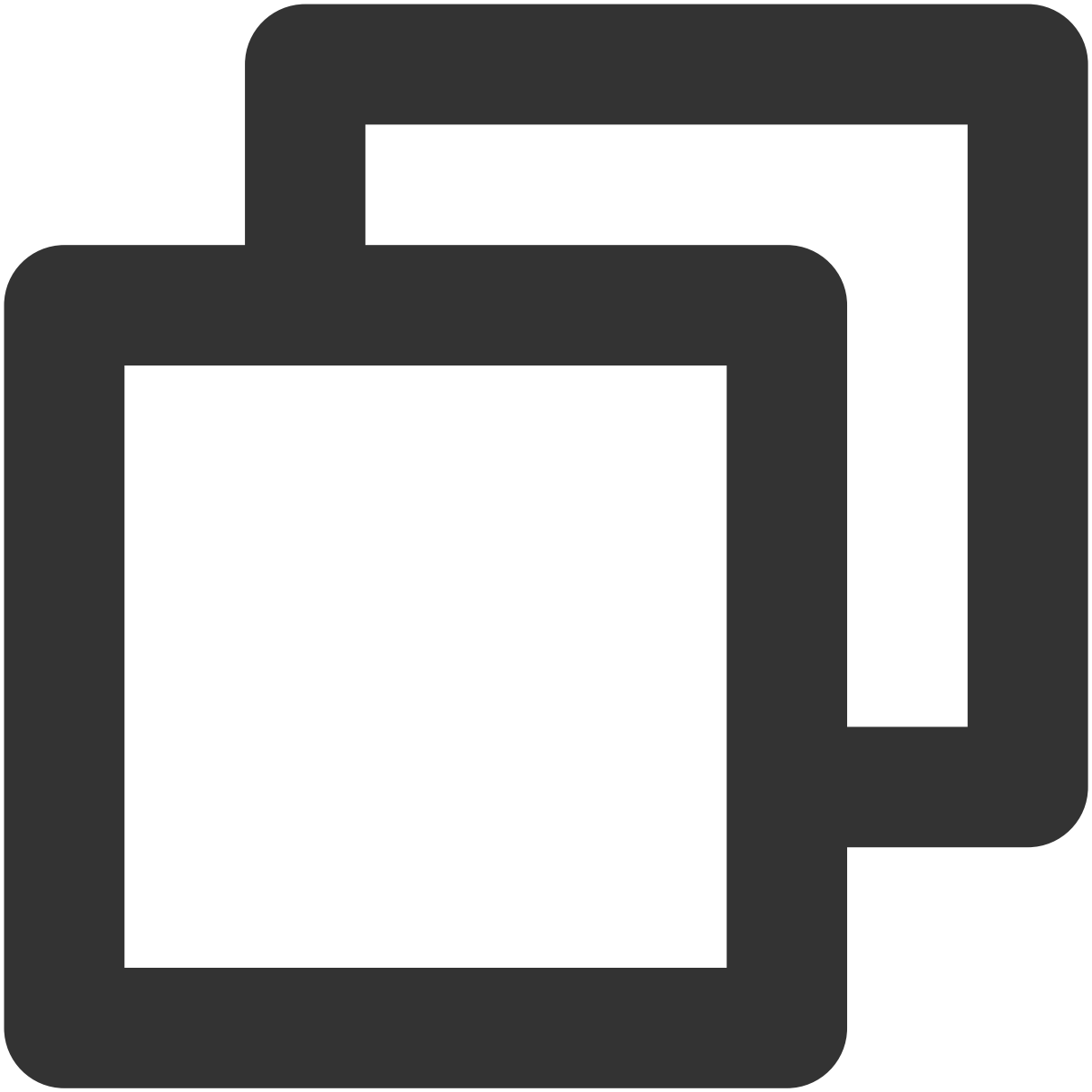
3. 在 APP 的 build.gradle 中添加：



```
dependencies {
 api project(':tuiconversation')
```

```
    api project(':tuicontact')
    api project(':tuichat')
    api project(':tuisearch')
    api project(':tuigroup')
    api project(':tuicommunity')
    api project(':tuicallkit')
    api project(':tuiroomkit')
    // Integrate speech-to-text plugin, supported from version 7.5
    api project(':tuivoicetotextplugin')
    // Integrate customer service plugin, supported from version 7.6
    api project(':tuicustomerserviceplugin')
    // Integrate chatbot plugin, supported from version 7.7
    api project(':tuichatbotplugin')
    // Integrate translation plugin, supported from version 7.2 (Value-added feature a
    api project(':tuitranslationplugin')
    // Integrate emoji reaction plugin, supported from version 7.8 (To use this module
    api project(':tuiemojiplugin')
    // Integrate group chain plugin, supported from version 7.1
    api 'com.tencent.imsdk:tuigroupnote-plugin:8.1.6103'
    // Integrate group voting plugin, supported from version 7.1
    api 'com.tencent.imsdk:tuipoll-plugin:8.1.6103'
    // Integrate session grouping plugin, supported from version 7.3
    api 'com.tencent.imsdk:tuiconversationgroup-plugin:8.1.6103'
    // Integrate session tagging plugin, supported from version 7.3
    api 'com.tencent.imsdk:tuiconversationmark-plugin:8.1.6103'
    // Integrate message push plugin, supported from version 7.6
    api 'com.tencent.timpush:timpush:8.1.6103'
    // Integrate the corresponding manufacturer's push package as needed
    api 'com.tencent.timpush:fcm:8.1.6103'
    api 'com.tencent.timpush:xiaomi:8.1.6103'
    api 'com.tencent.timpush:meizu:8.1.6103'
    api 'com.tencent.timpush:oppo:8.1.6103'
    api 'com.tencent.timpush:vivo:8.1.6103'
    api 'com.tencent.timpush:huawei:8.1.6103'
    api 'com.tencent.timpush:honor:8.1.6103'
}
```

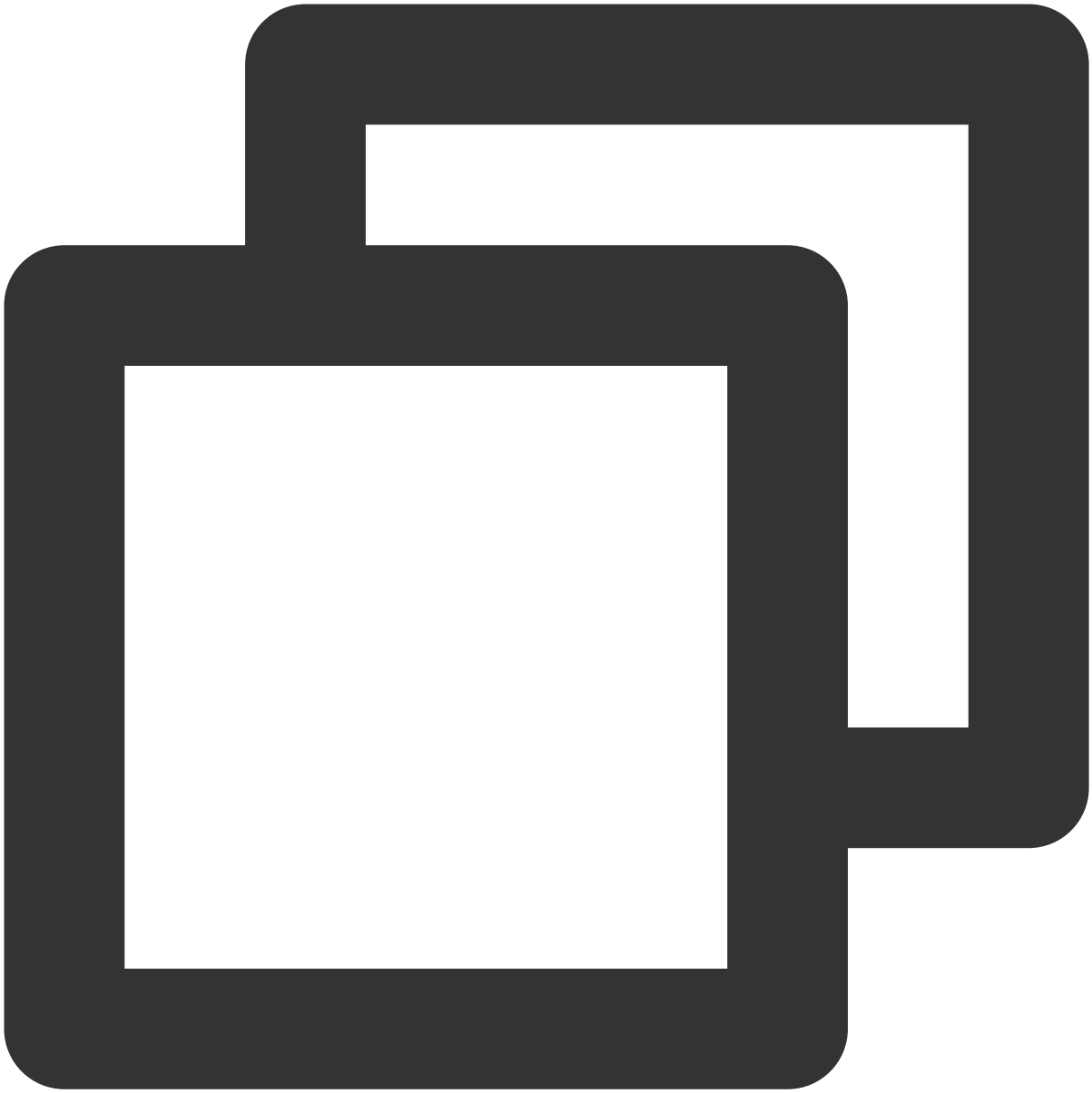
4. 在 gradle.properties 文件中加入下行，表示自动转换三方库以兼容 AndroidX：



```
android.enableJetifier=true
```

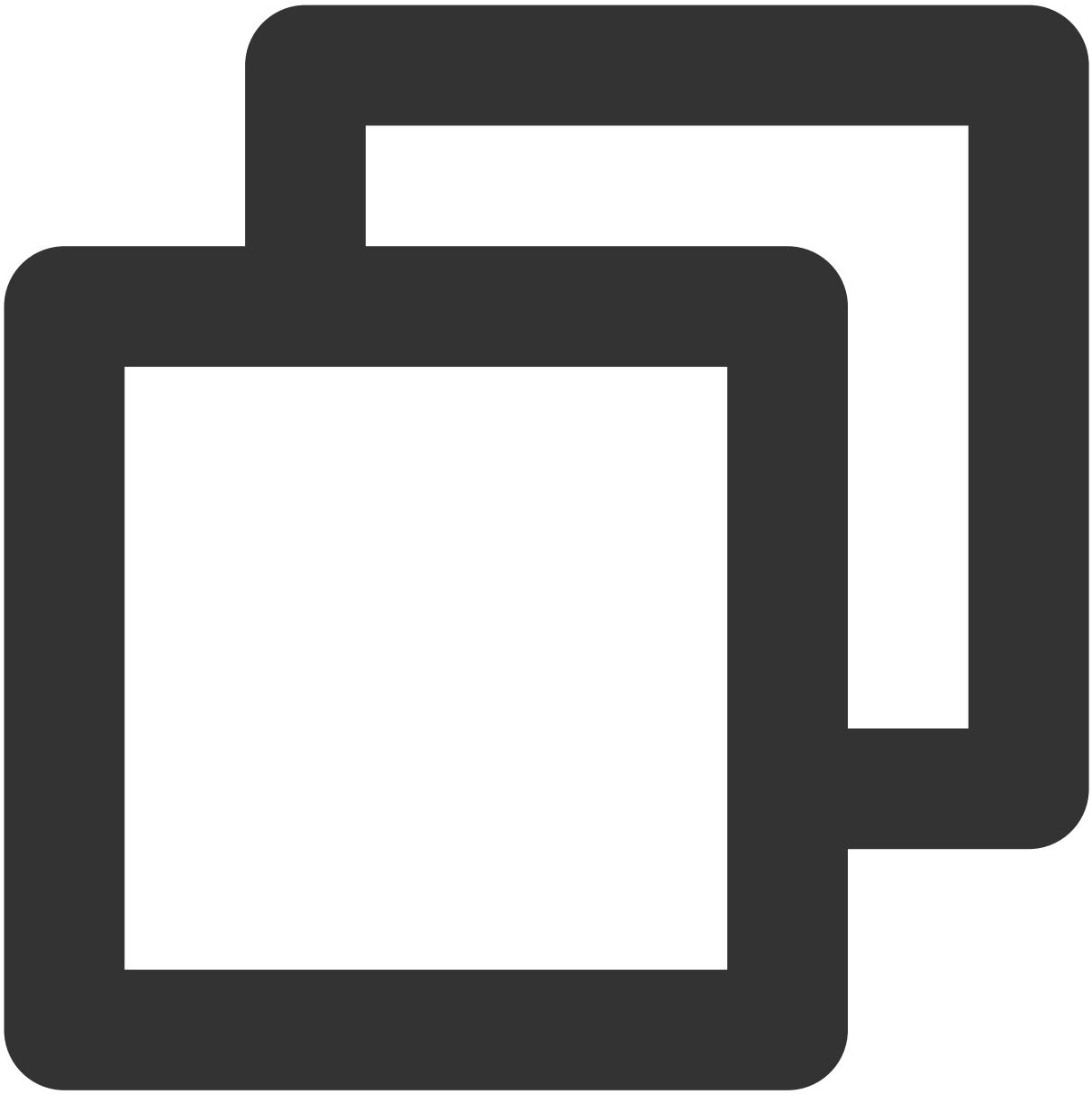
5.

添加 maven 仓库 和 Kotlin 支持，在 root 工程的 build.gradle 文件（与 settings.gradle 同级）中添加：



```
buildscript {
 repositories {
     mavenCentral()
     maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
 }
 dependencies {
     classpath 'com.android.tools.build:gradle:7.0.0'
     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.31"
 }
}
```

如果你使用 Gradle 8.x，则需要添加以下代码。



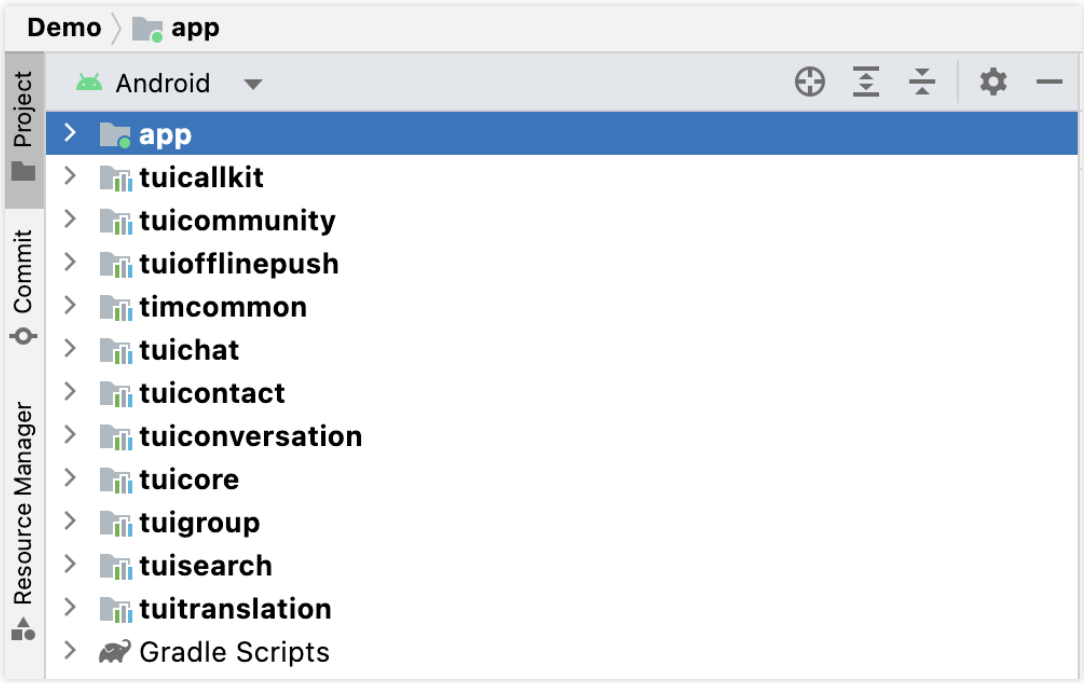
```
buildscript {
 repositories {
     mavenCentral()
     maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
 }
 dependencies {
     classpath 'com.android.tools.build:gradle:8.0.2'
     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.0"
 }
```

```
}
```

说明：

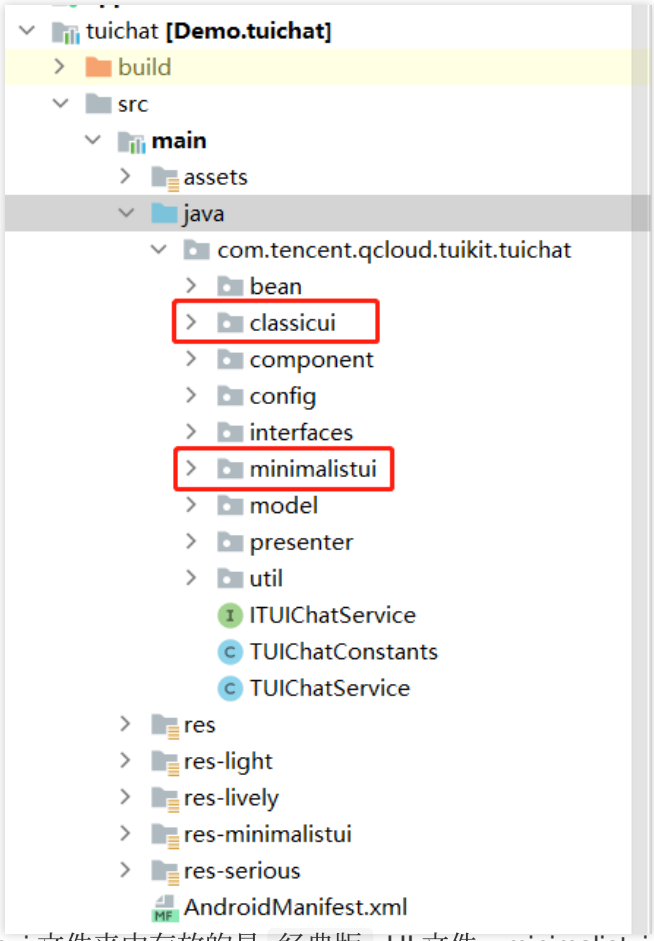
Kotlin 、Gradle 和 AGP 的版本对应关系可以在此查看。

6. 同步工程，编译运行。工程结构预期效果如图所示：



7. 裁剪不需要的 UI 文件（可选）

`经典版` 和 `简约版` UI 互不影响，可独立运行。`经典版` 和 `简约版` 的 UI 文件都在各 TUI 组件中，放在不同的文件夹里，以 `TUIChat` 组件为例：



classicui 文件夹中存放的是 `经典版` UI 文件，minimalistui 文件夹中存放的是 `简约版` UI 文件。

如果您要集成简约版 UI，可直接删除 classicui 文件夹，同时删除 AndroidManifest.xml 文件中经典版 UI 对应的 Activity 和 Service 。

注意：

经典版和简约版 UI 不能混用，集成多个组件时，您必须同时选择经典版 UI 或者 简约版 UI。例如，经典版 `TUIChat` 组件必须与经典版 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 组件搭配使用。同理，简约版 `TUIChat` 组件必须与简约版 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 组件搭配使用。

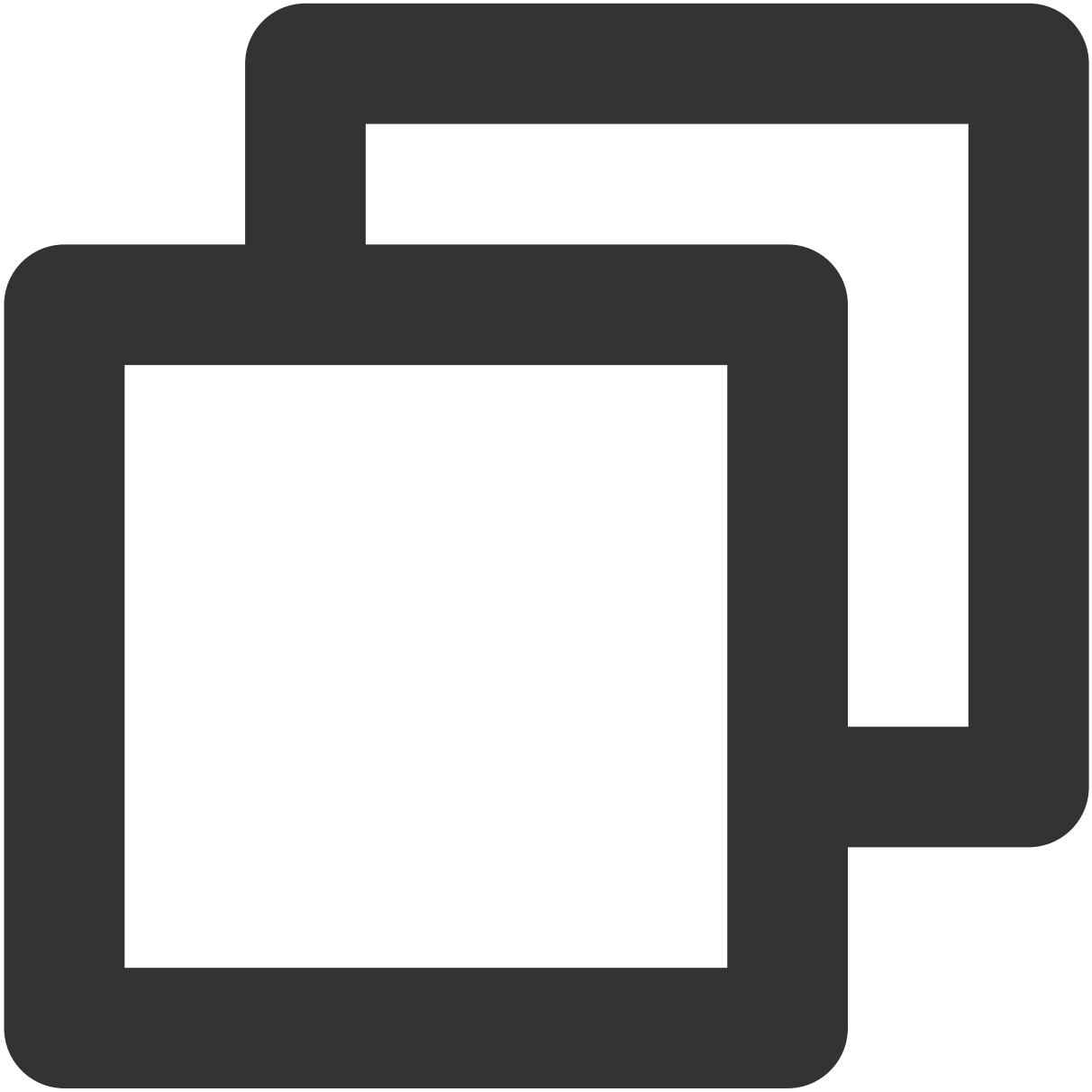
构建基础界面

集成 TUIKit 完成后，如果您想要继续构建聊天、会话等基础界面，请参考文档：构建聊天界面、构建会话列表等。

常见问题

提示 "Manifest merger failed : Attribute application@allowBackup value=(true) from AndroidManifest.xml" 如何处理？

IM SDK 中默认 `allowBackup` 的值为 `false` ，表示关闭应用的备份和恢复功能。
您可以在您的 `AndroidManifest.xml` 文件中删除 `allowBackup` 属性，表示关闭备份和恢复功能；也可以在 `AndroidManifest.xml` 文件的 application 节点中添加 `tools:replace="android:allowBackup"` 表示覆盖 IM SDK 的设置，使用您自己的设置。 例如：



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.tencent.qcloud.tuikit.myapplication">

    <application
```

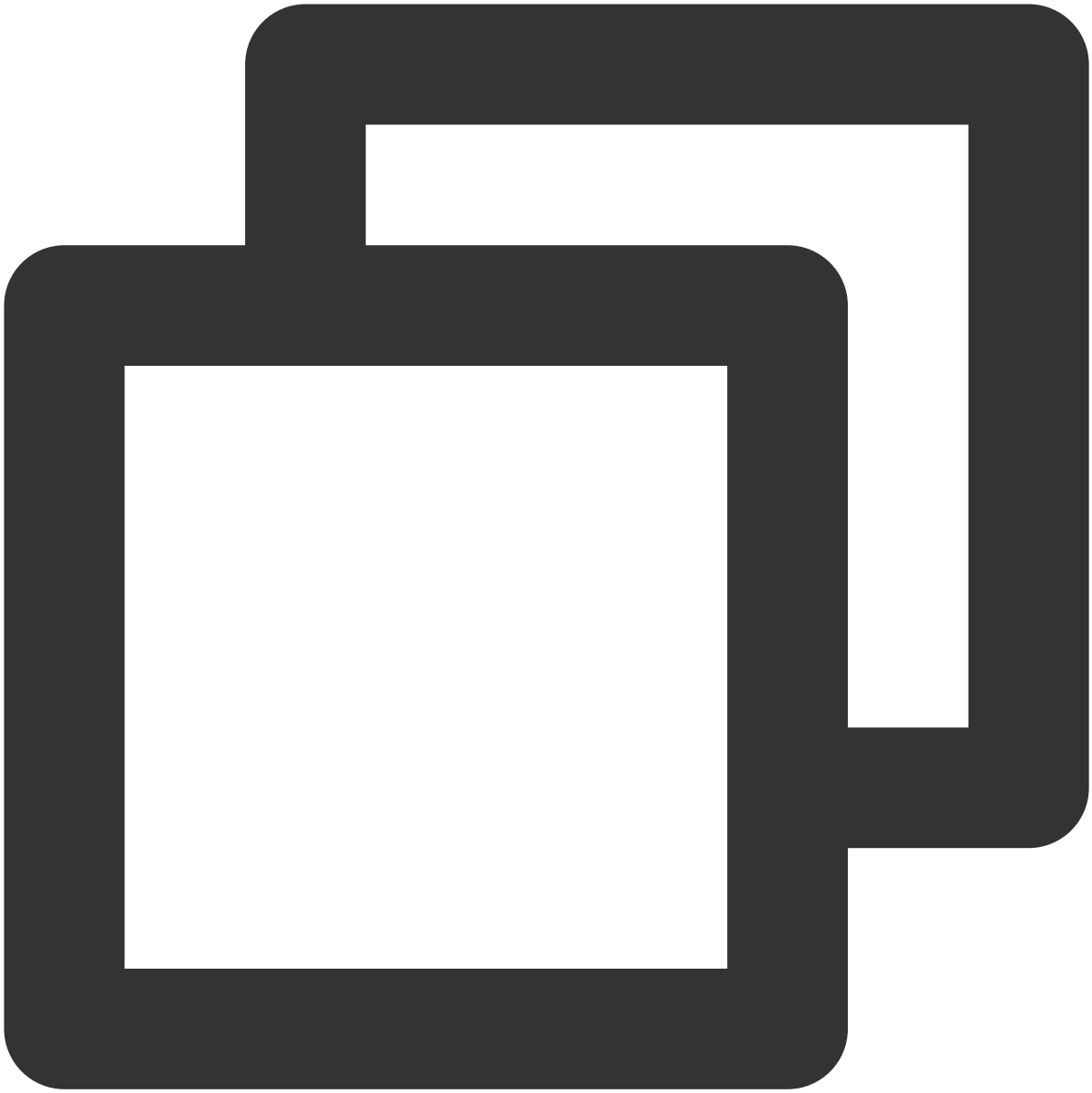
```
        android:allowBackup="true"
        android:name=".MApplication"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:replace="android:allowBackup">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

提示 "NDK at /Users/*/Library/Android/sdk/ndk-bundle did not have a source.properties file" 如何处理？**

只需要在 local.properties 文件中加入您的 NDK 路径，例如：`ndk.dir=/Users/***/Library/Android/sdk/ndk/16.1.4479499`

提示 "Cannot fit requested classes in a single dex file" 如何处理？

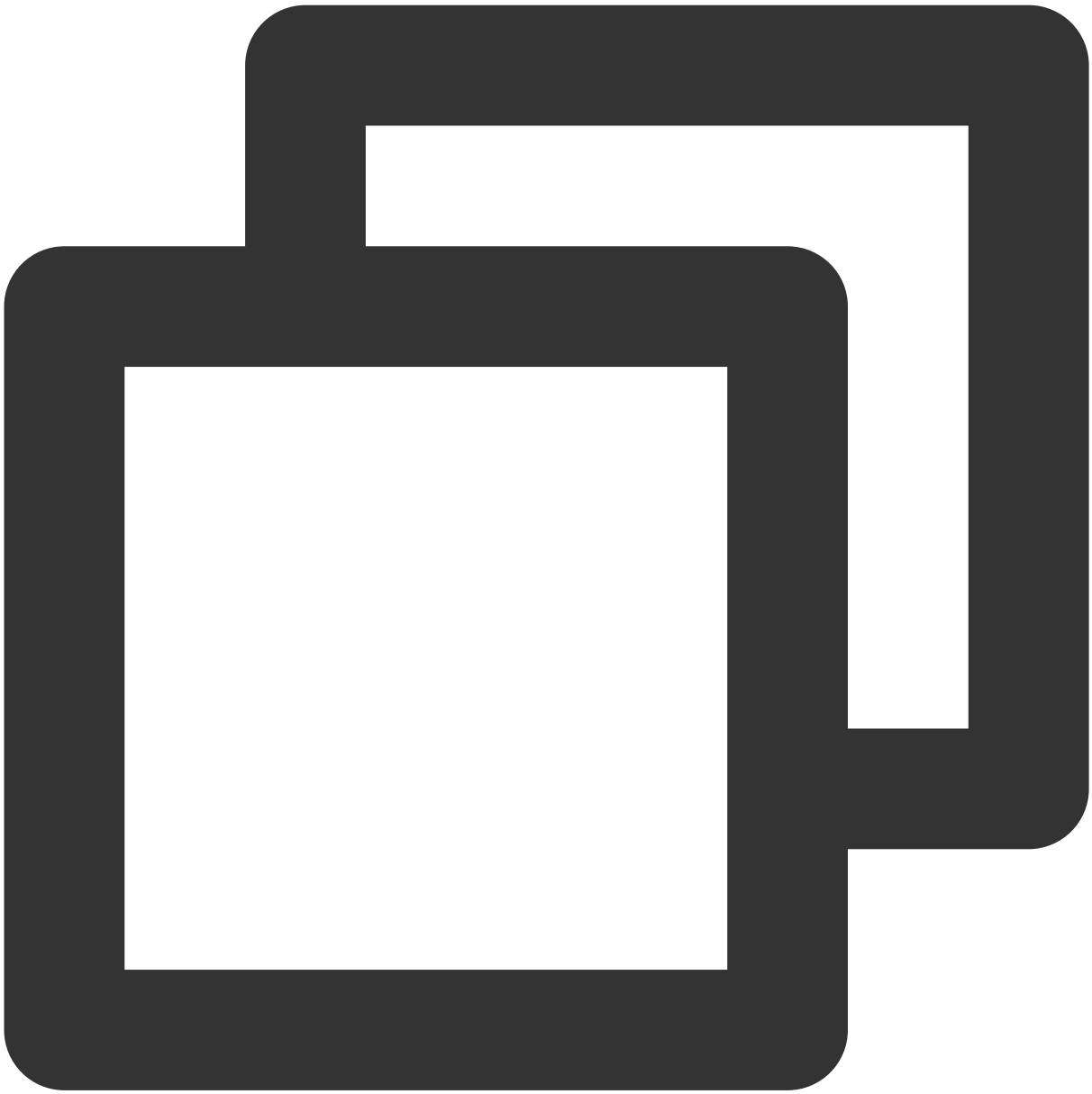
出现此问题可能是您的 API 级别设置比较低，需要在 App 的 build.gradle 文件中开启 `MultiDex` 支持，添加 `multiDexEnabled true` 和对应依赖：



```
android {
    defaultConfig {
        ...
        minSdkVersion 19
        targetSdkVersion 30
        multiDexEnabled true
    }
    ...
}
dependencies {
    implementation "androidx.multidex:multidex:2.0.1"
```

```
}
```

同时，在您的 Application 文件中添加以下代码：



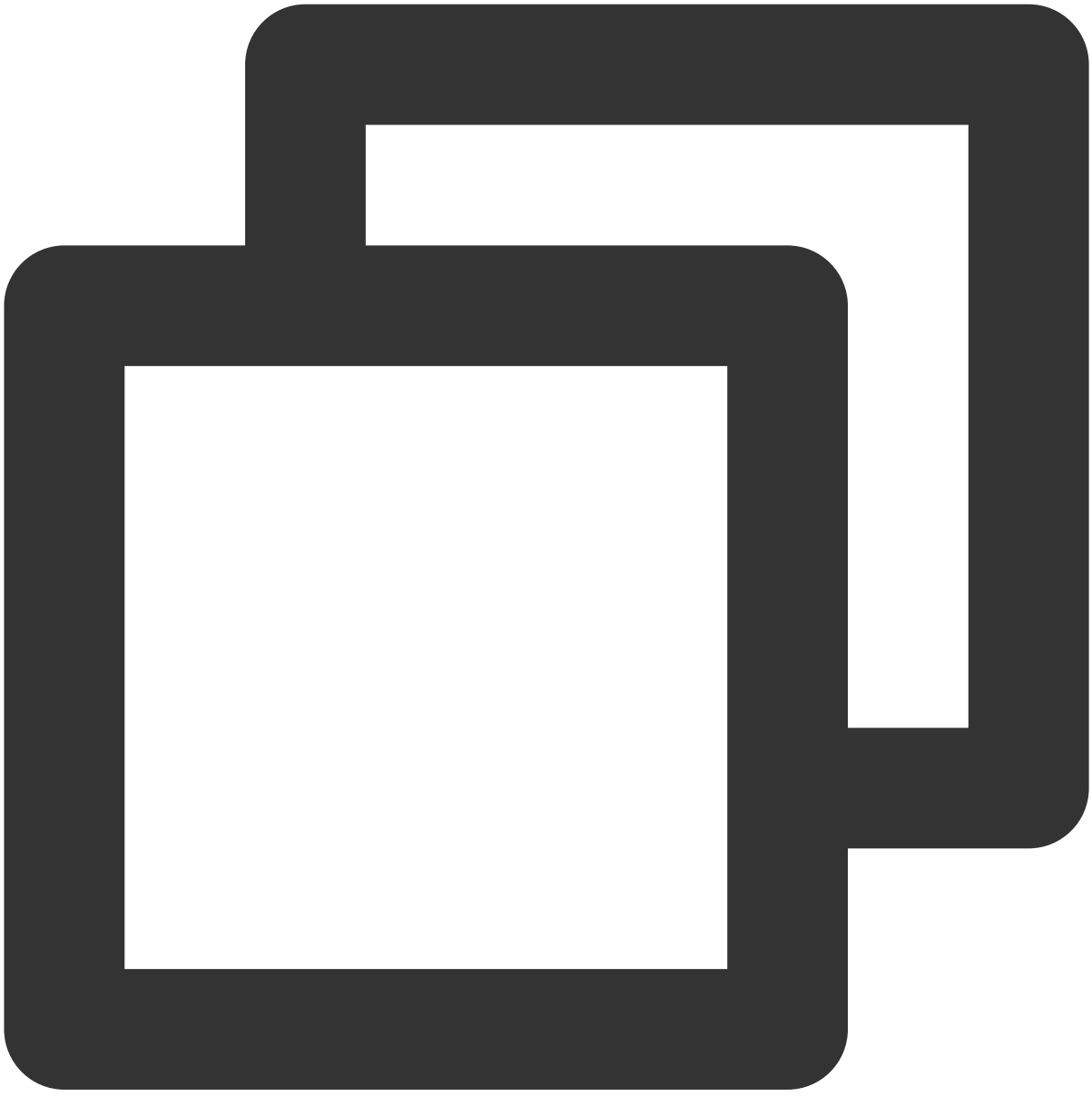
```
public class MyApplication extends SomeOtherApplication {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        MultiDex.install(this);
    }
}
```

提示 "Plugin with id 'kotlin-android' not found." 如何处理？

因为 `TUIChat` 组件使用了 Kotlin 代码，所以需要添加 Kotlin 构建插件。请参考 源码集成第 5 步。

Debug 版本的 App 功能正常，Release 版本的 App 功能出现异常 ？

出现此问题很大概率是混淆导致的，请尽量不要混淆 TUIKit 。可以添加如下混淆规则：



```
# Avoid deleting code logic
-dontshrink
-dontoptimize
# Avoid aliasing TUIKit
-keep class com.tencent.qcloud.** { *; }
```

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

iOS

最近更新时间：2024-06-25 10:43:44

本文将介绍如何集成 `TUIKit` 组件。

说明：

从 5.7.1435 版本开始，TUIKit 支持模块化集成，支持了经典版 UI，您可以根据自己的需求集成所需模块。

从 6.9.3557 版本开始，TUIKit 新增了全新的简约版 UI。

您可以根据需求自由选择经典版或简约版 UI 组件。如果您还不了解各个界面库的效果，可以查阅文档 TUIKit 界面库介绍。

开发环境要求

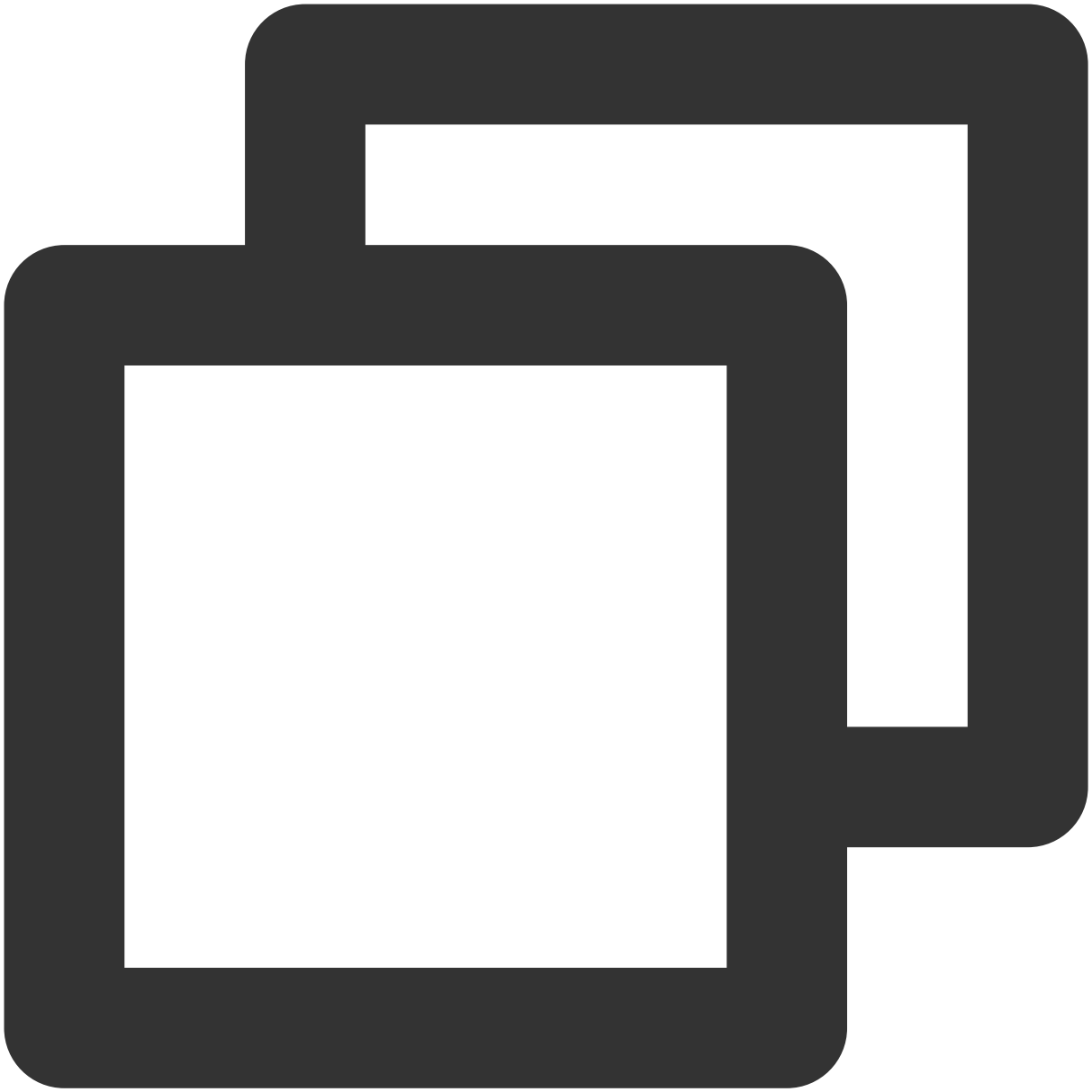
Xcode 10 及以上

iOS 9.0 及以上

CocoaPods 集成

1. 安装 CocoaPods。

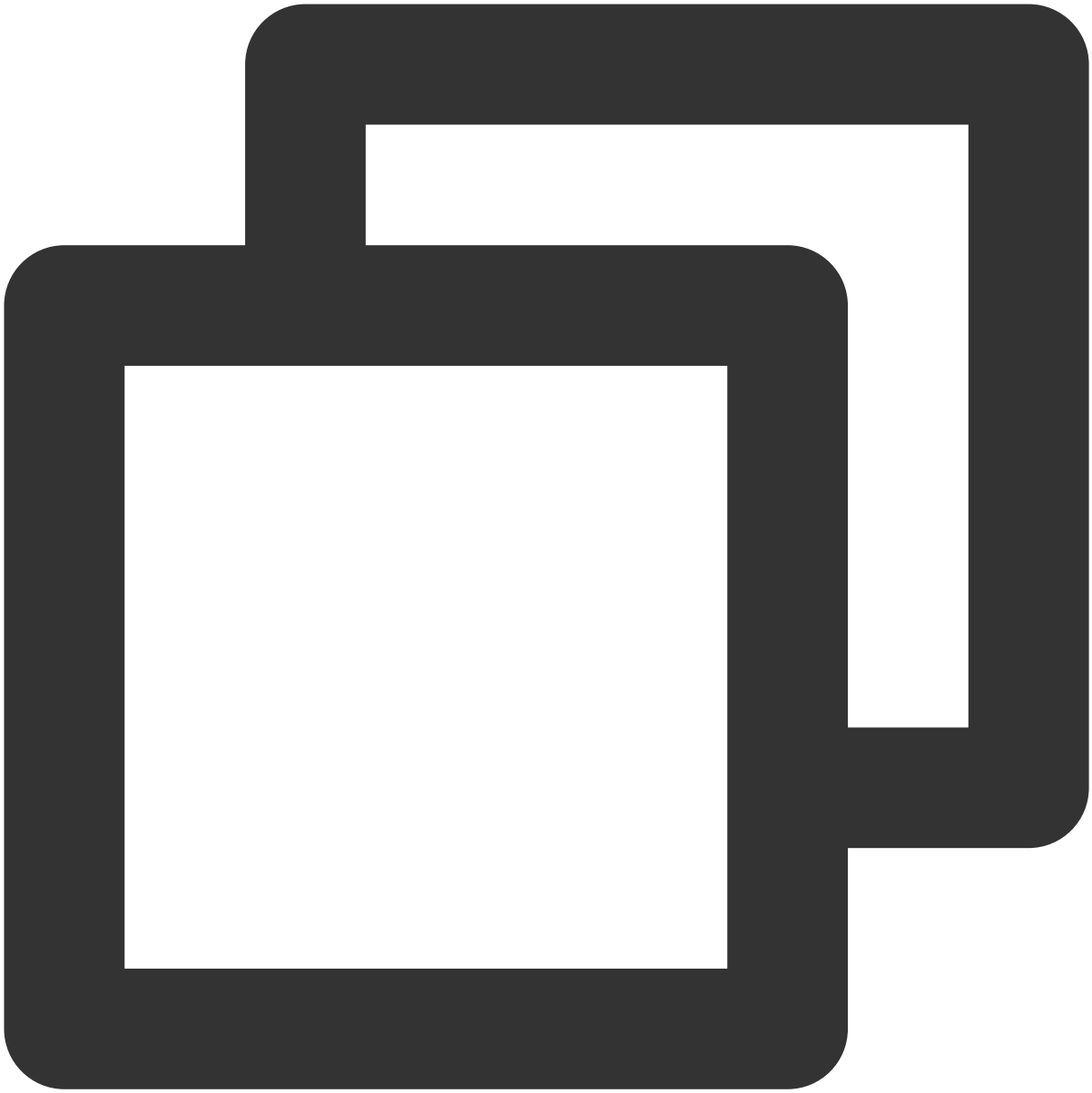
在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：



```
sudo gem install cocoapods
```

2. 创建 Podfile 文件。

进入项目所在路径输入以下命令行，之后项目路径下会出现一个 Podfile 文件。



```
pod init
```

3. 根据业务需求在 Podfile 中添加对应的 TUIKit 组件。组件之间相互独立，添加或删除均不影响工程编译。您可以按需选择不同的 Podfile 集成方式：

远程 CocoaPods 集成

DevelopmentPods 本地集成

以上两种集成方式的优缺点如下表所示：

集成方式	适合场景	优点	缺点

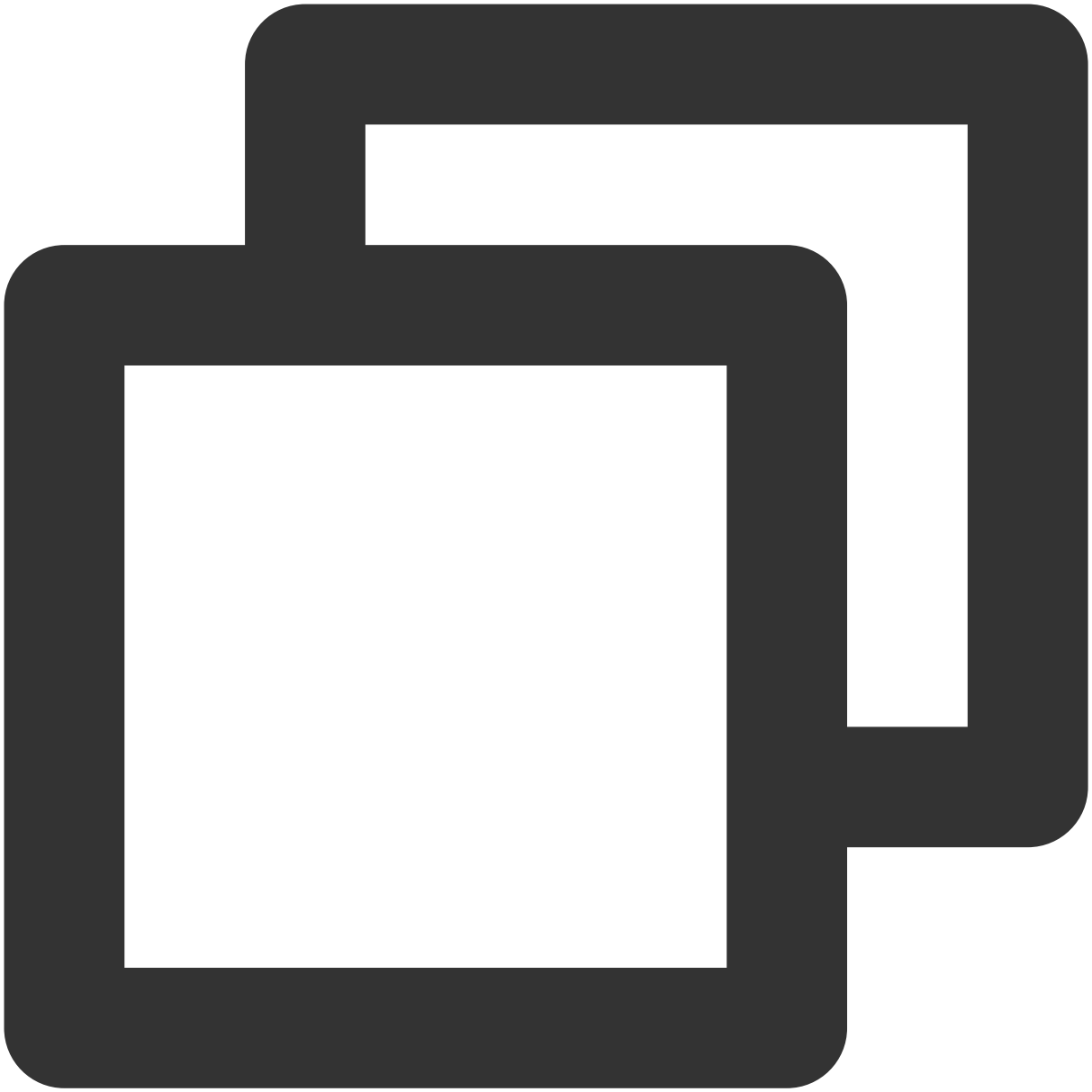
远程 CocoaPods 集成	适合无源码修改时的集成。	当 TUIKit 有版本更新时，您只需再次 `Pod update` 即可完成更新。	当您有源码修改，使用 `Pod update` 更新时，新版本的 TUIKit 会覆盖您的修改。
本地 DevelopmentPods 集成	适合有涉及源码自定义修改的客户。	当您有自己的 git 仓库时，可以跟踪修改。修改源码后，使用 `Pod update` 更新其他远程 Pod 库时，不会覆盖您的修改。	您需要手动将 TUIKit 源码覆盖您本地 TUIKit 文件夹进行更新。

远程 CocoaPods 集成

您可以在 Podfile 中按需添加组件库：

简约版

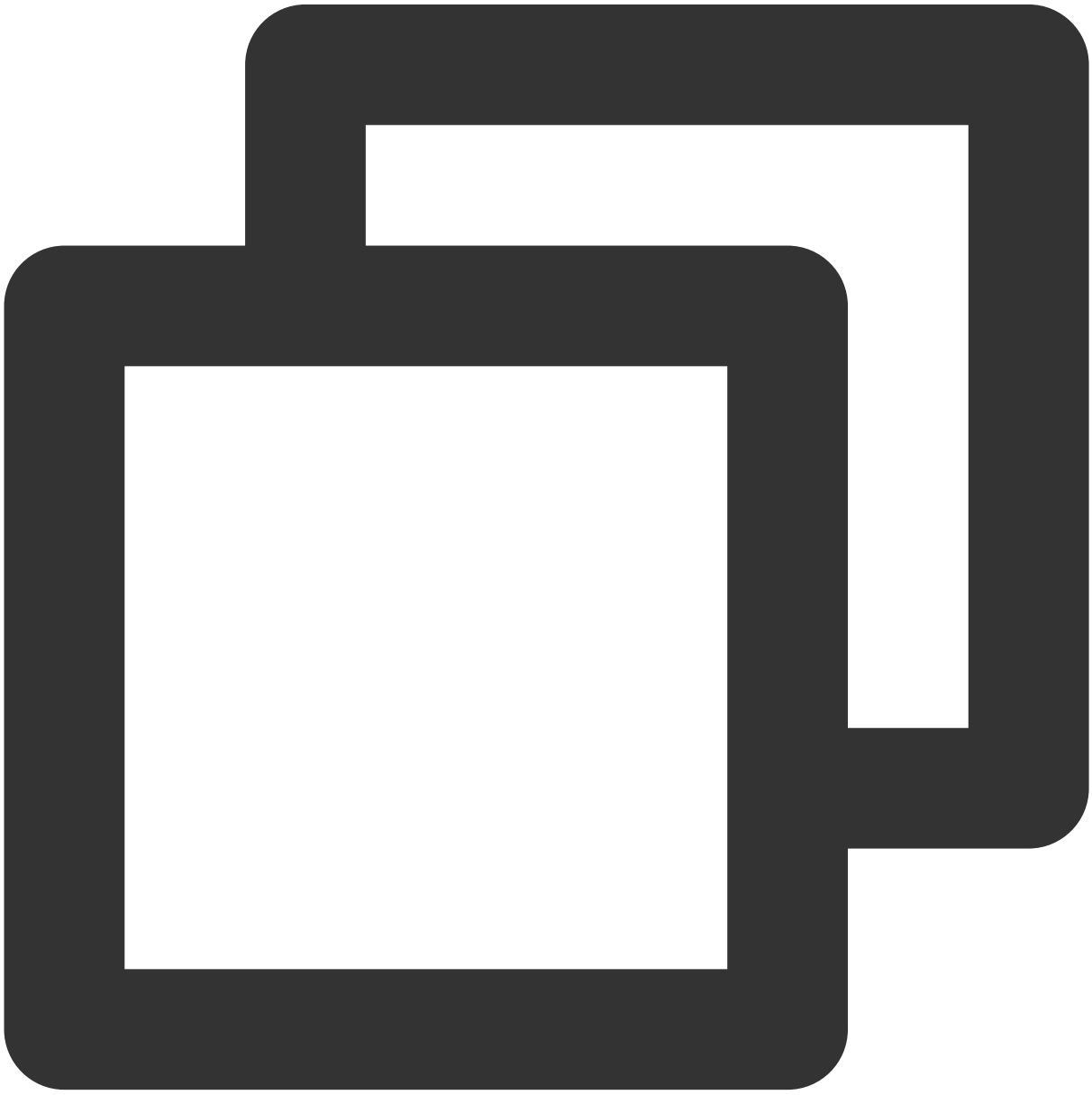
经典版



```
# Uncomment the next line to define a global platform for your project
# ...
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
# Prevent `*.xcassets` in TUIKit from conflicting with your project
install! 'cocoapods', :disable_input_output_paths => true

# Replace `your_project_name` with your actual project name
target 'your_project_name' do
  # Comment the next line if you don't want to use dynamic frameworks
  # TUIKit components are dependent on static libraries. Therefore, you need to mas
```

```
  # use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers, the Po
  # use_modular_headers!

  # Integrate the chat feature
  pod 'TUIChat/UI_Minimalist'
  # Integrate the conversation list feature
  pod 'TUIConversation/UI_Minimalist'
  # Integrate the relationship chain feature
  pod 'TUIContact/UI_Minimalist'
  # Integrate the group feature
  pod 'TUIGroup/UI_Minimalist'
  # Integrate the search feature (To use this feature, you need to purchase the Pre
  pod 'TUISearch/UI_Minimalist'
  # Integrate the audio/video call feature
  pod 'TUICallKit'
  # Integrate Translation Plugin, supported starting from version 7.2 (Value-added
  pod 'TUITranslationPlugin'
  # Integrate Session Tagging Plugin, supported starting from version 7.3
  pod 'TUIConversationMarkPlugin'
  # Integrate Speech-to-Text Plugin, supported starting from version 7.5
  pod 'TUIVoiceToTextPlugin'
  # Integrate message push plugin, supported starting from version 7.6
  pod 'TIMPush'
end

#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
```

```
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
          end
      end
  end
```



```
# Uncomment the next line to define a global platform for your project
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
# Prevent `*.xcassets` in TUIKit from conflicting with your project
install! 'cocoapods', :disable_input_output_paths => true

# Replace `your_project_name` with your actual project name
target 'your_project_name' do
  # Comment the next line if you don't want to use dynamic frameworks
  # TUIKit components are dependent on static libraries. Therefore, you need to mas
  # use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers, the Po
  # use_modular_headers!

  # Integrate the chat feature
  pod 'TUIChat/UI_Classic'
  # Integrate the conversation list feature
  pod 'TUIConversation/UI_Classic'
  # Integrate the relationship chain feature
  pod 'TUIContact/UI_Classic'
  # Integrate the group feature
  pod 'TUIGroup/UI_Classic'
  # Integrate the search feature (To use this feature, you need to purchase the Pre
  pod 'TUISearch/UI_Classic'
  # Integrate the audio/video call feature
  pod 'TUICallKit'
  # Integrate Voting Plugin, supported starting from version 7.1
  pod 'TUIPollPlugin'
  # Integrate Group Chain Plugin, supported starting from version 7.1
  pod 'TUIGroupNotePlugin'
  # Integrate Translation Plugin, supported starting from version 7.2 (Value-added
  pod 'TUITranslationPlugin'
  # Integrate Session Grouping Plugin, supported starting from version 7.3
  pod 'TUIConversationGroupPlugin'
  # Integrate Session Tagging Plugin, supported starting from version 7.3
  pod 'TUIConversationMarkPlugin'
  # Integrate Speech-to-Text Plugin, supported starting from version 7.5
  pod 'TUIVoiceToTextPlugin'
  # Integrate Customer Service Plugin, supported starting from version 7.6
  pod 'TUICustomerServicePlugin'
  # Integrate Bot Plugin, supported starting from version 7.7
  pod 'TUIChatBotPlugin'
  # Integrate message push plugin, supported starting from version 7.6
  pod 'TIMPush'
```

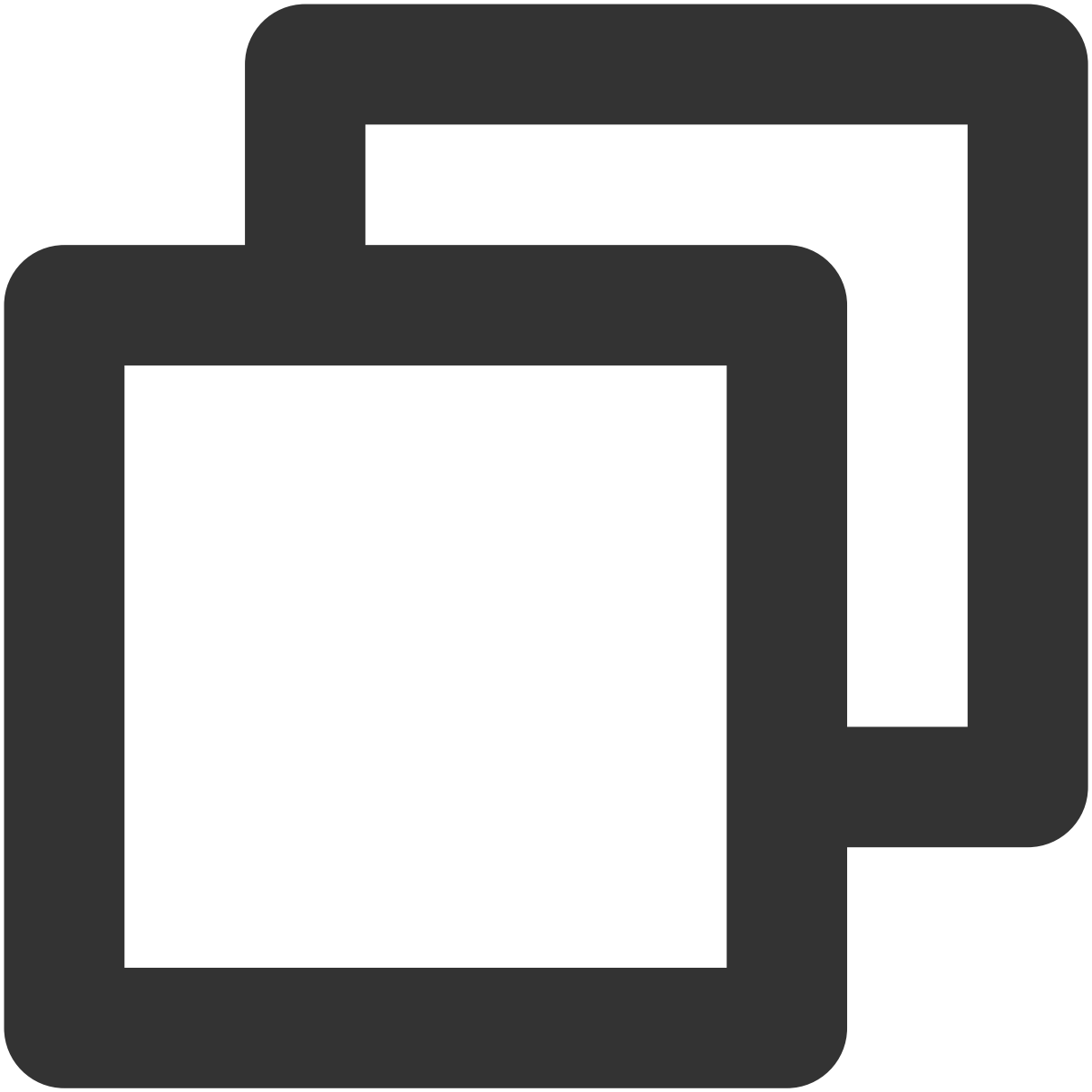
版权所有：腾讯云计算（北京）有限责任公司

```
end

#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```

说明

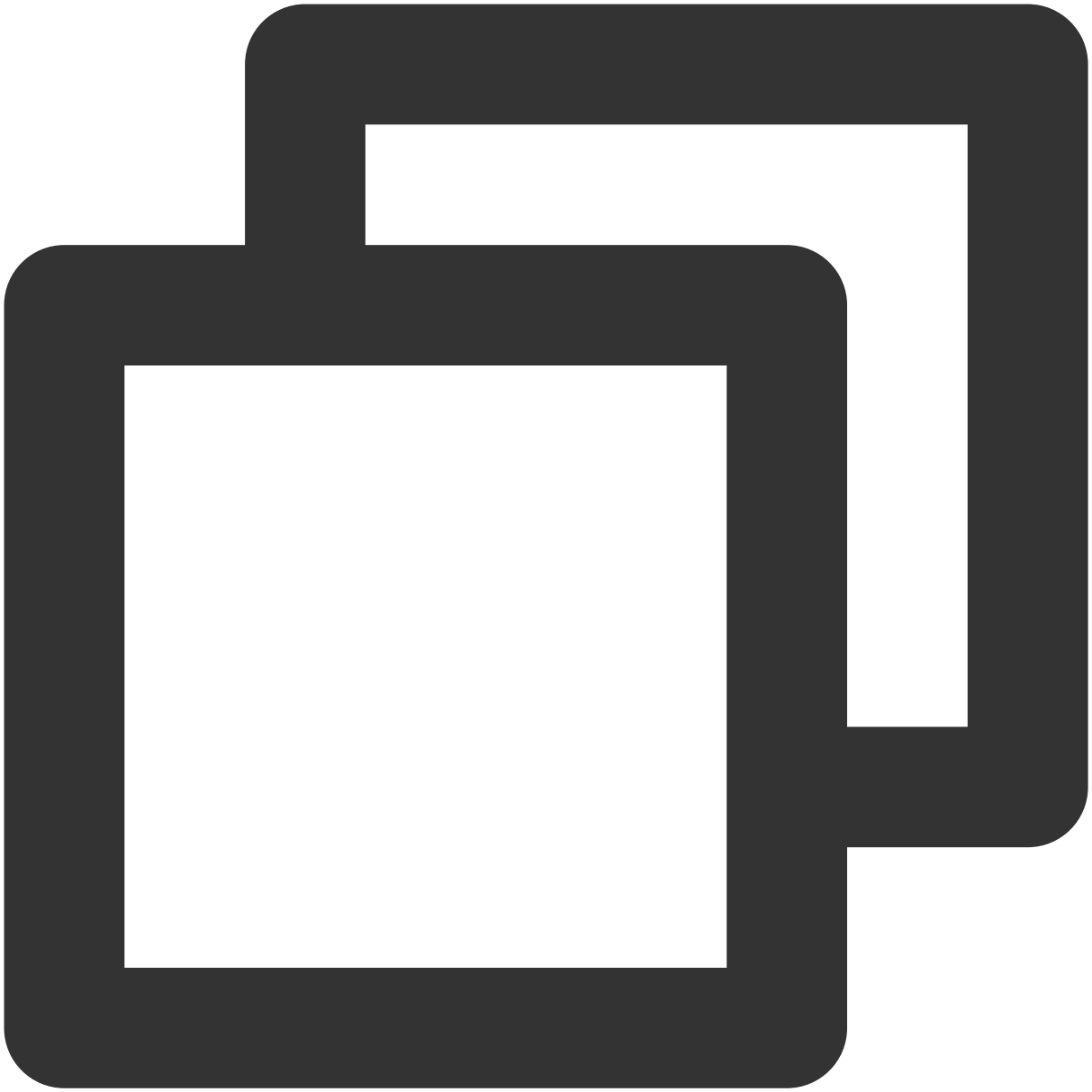
1. 如果您直接 `pod 'TUIChat'` ，不指定经典版或简约版，默认会集成两套版本 UI 组件。
2. 经典版和简约版 UI 不能混用，集成多个组件时，您必须同时全部选择经典版 UI 或简约版 UI。例如，经典版 `TUIChat` 组件必须与经典版 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 组件搭配使用。同理，简约版 `TUIChat` 组件必须与简约版 `TUIConversation` 、 `TUIContact` 、 `TUIGroup` 组件搭配使用。
3. 如果您使用的是 Swift，请开启 `use_modular_headers!` ，并将头文件引用改成 @import 模块名形式引用。

Podfile 修改完毕后，执行以下命令，安装 TUIKit 组件。



```
pod install
```

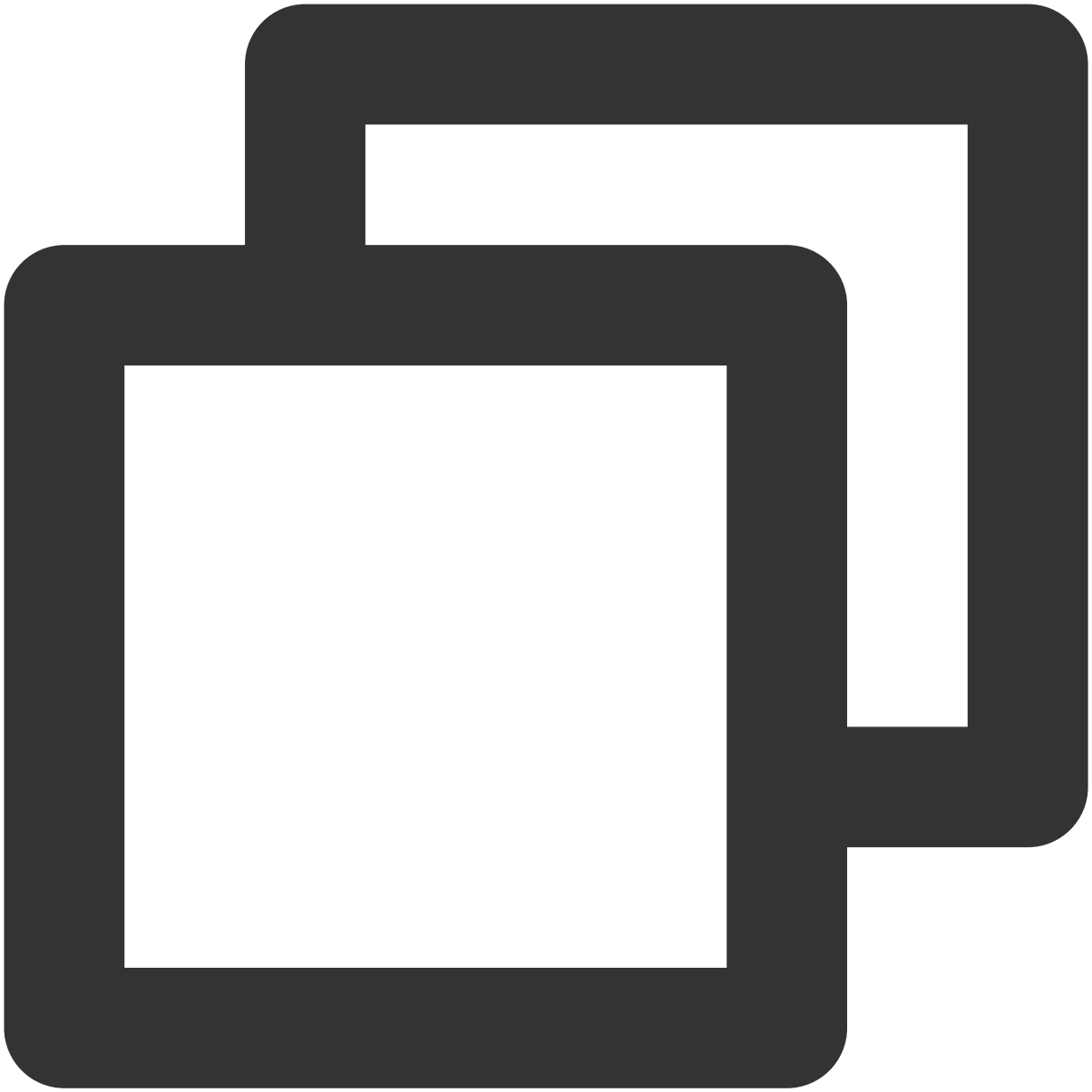
如果无法安装 TUIKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。



```
pod repo update
```

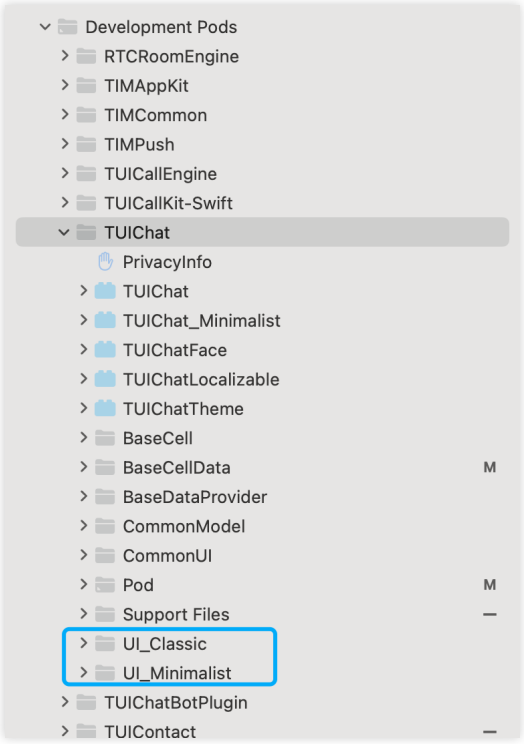
之后执行以下命令，更新组件库的 Pod 版本。

版权所有：腾讯云计算（北京）有限责任公司



```
pod update
```

集成全部的 TUIKit 组件后的项目结构：

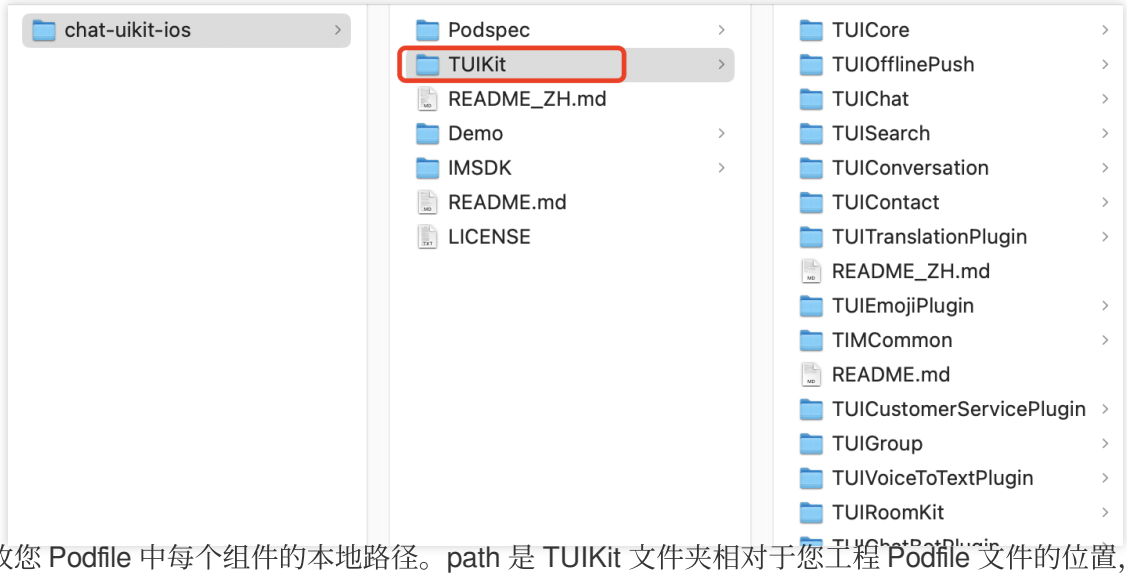


注意：

若您操作遇到错误，可查阅文末的常见问题。

本地 DevelopmentPods 源码集成

1. 从 GitHub 下载 TUIKit 源码。直接拖入您的工程目录下:



2. 修改您 Podfile 中每个组件的本地路径。path 是 TUIKit 文件夹相对于您工程 Podfile 文件的位置，常见的有：

TUIKit 文件夹位于您工程 Podfile 文件**父目录**： `pod 'TUICore', :path => "../TUIKit/TUICore"`

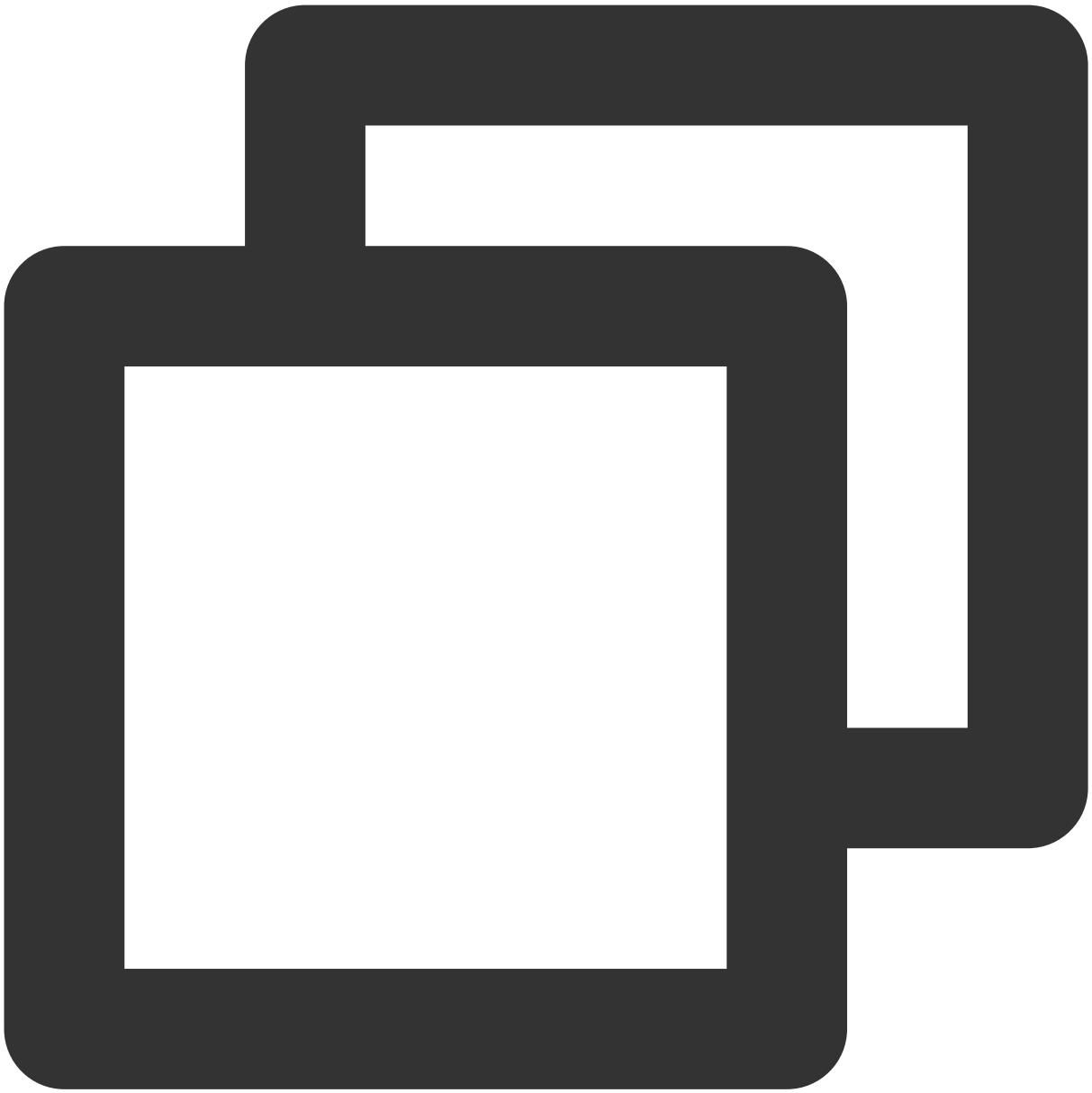
TUIKit 文件夹位于您工程 Podfile 文件**当前目录**： `pod 'TUICore', :path => "/TUIKit/TUICore"`

TUIKit 文件夹位于您工程 Podfile 文件**子目录**： `pod 'TUICore', :path => "./TUIKit/TUICore"`

以 TUIKit 文件夹位于您工程 Podfile 文件父目录为例：

版权所有：腾讯云计算（北京）有限责任公司

DevelopmentPodfile

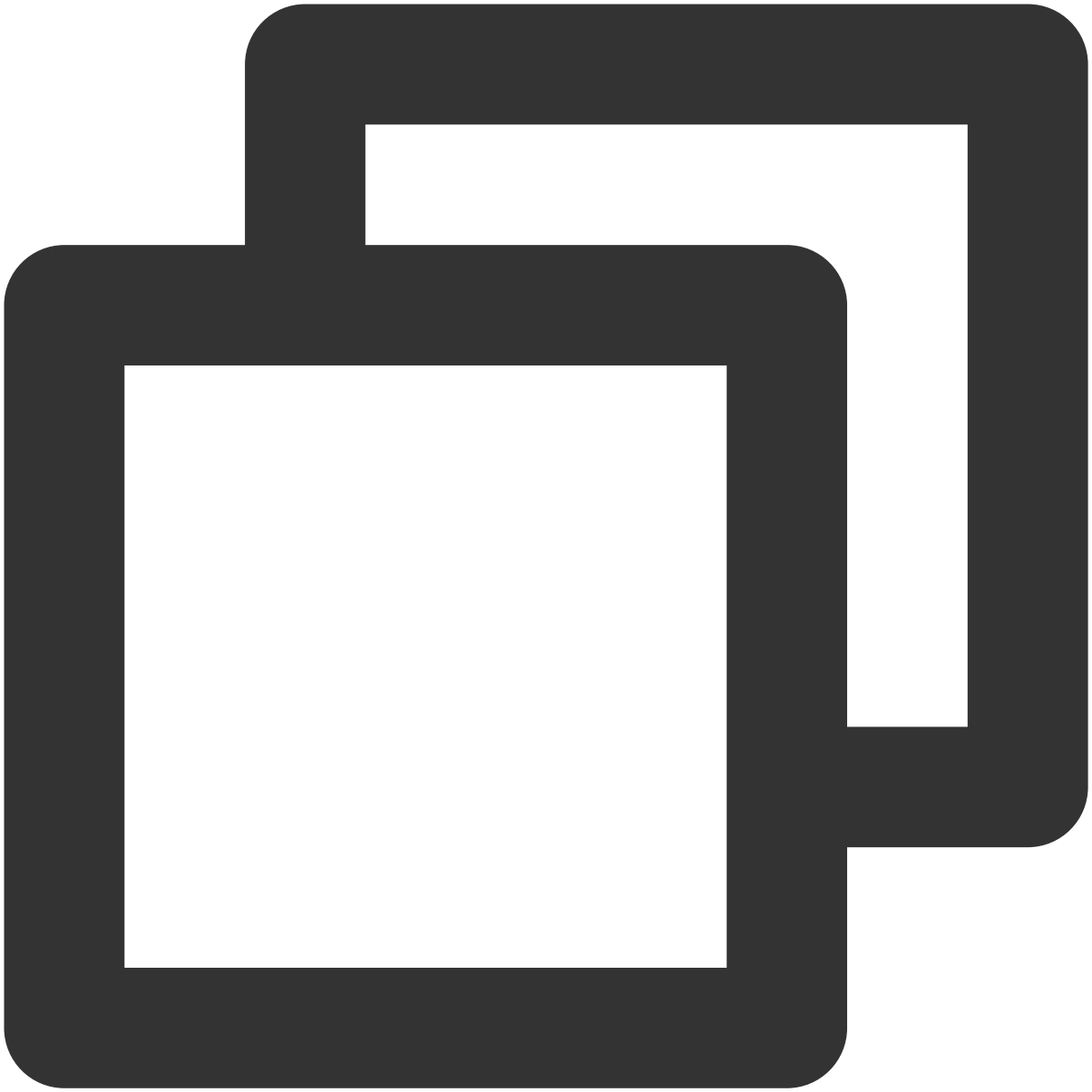


```
ment# Uncomment the next line to define a global platform for your project
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
install! 'cocoapods', :disable_input_output_paths => true

# Replace `your_project_name` with your actual project name
target 'your_project_name' do
  # Uncomment the next line if you're using Swift or would like to use dynamic fram
  use_frameworks!
```

```
  use_modular_headers!

  # Note: To perform upgrade when using a local integration solution, you need to a
  # https://github.com/TencentCloud/TIMSDK/tree/master/iOS/TUIKit
  # to obtain the latest component code, and place it in the local specified direct
  # Note: When custom modifications conflict with remote changes, you need to manua

  # Integrate the basic library (required)
  pod 'TUICore', :path => "../TUIKit/TUICore"
  pod 'TIMCommon', :path => "../TUIKit/TIMCommon"
  # Integrate TUIKit components (optional)
  # Integrate the chat feature
  pod 'TUIChat', :path => "../TUIKit/TUIChat"
  # Integrate the conversation list feature
  pod 'TUIConversation', :path => "../TUIKit/TUIConversation"
  # Integrate the relationship chain feature
  pod 'TUIContact', :path => "../TUIKit/TUIContact"
  # Integrate the group feature
  pod 'TUIGroup', :path => "../TUIKit/TUIGroup"
  # Integrate the search feature (To use this feature, you need to purchase the Ult
  pod 'TUISearch', :path => "../TUIKit/TUISearch"
  # Integrate the audio/video call feature
  pod 'TUICallKit'
  # Integrate the video conference feature
  pod 'TUIRoomKit'

  # Integrate the TUIKitPlugin plugin (optional)
  # Note: The TUIKitPlugin plugin version must be the same as the TUICore version.
  # Ensure that the plugin version matches spec.version in "../TUIKit/TUICore/TUICo

  # Integrate the voting plugin, supported from version 7.1
  pod 'TUIPollPlugin', '7.6.5011'
  # Integrate the group chain plugin, supported from version 7.1
  pod 'TUIGroupNotePlugin', '7.6.5011'
  # Integrate translation plugin, supported from version 7.2 (Value-added feature a
  pod 'TUITranslationPlugin', '7.6.5011'
  # Integrate the session grouping plugin, supported from version 7.3
  pod 'TUIConversationGroupPlugin', '7.6.5011'
  # Integrate the session tagging plugin, supported from version 7.3
  pod 'TUIConversationMarkPlugin', '7.6.5011'
  # Integrate the offline push feature
  pod 'TIMPush'

  # Other Pods
  pod 'MJRefresh'
  pod 'Masonry'
end
```

```
#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```

3. Podfile 修改完毕后，执行以下命令，安装本地 TUIKit 组件。示例：



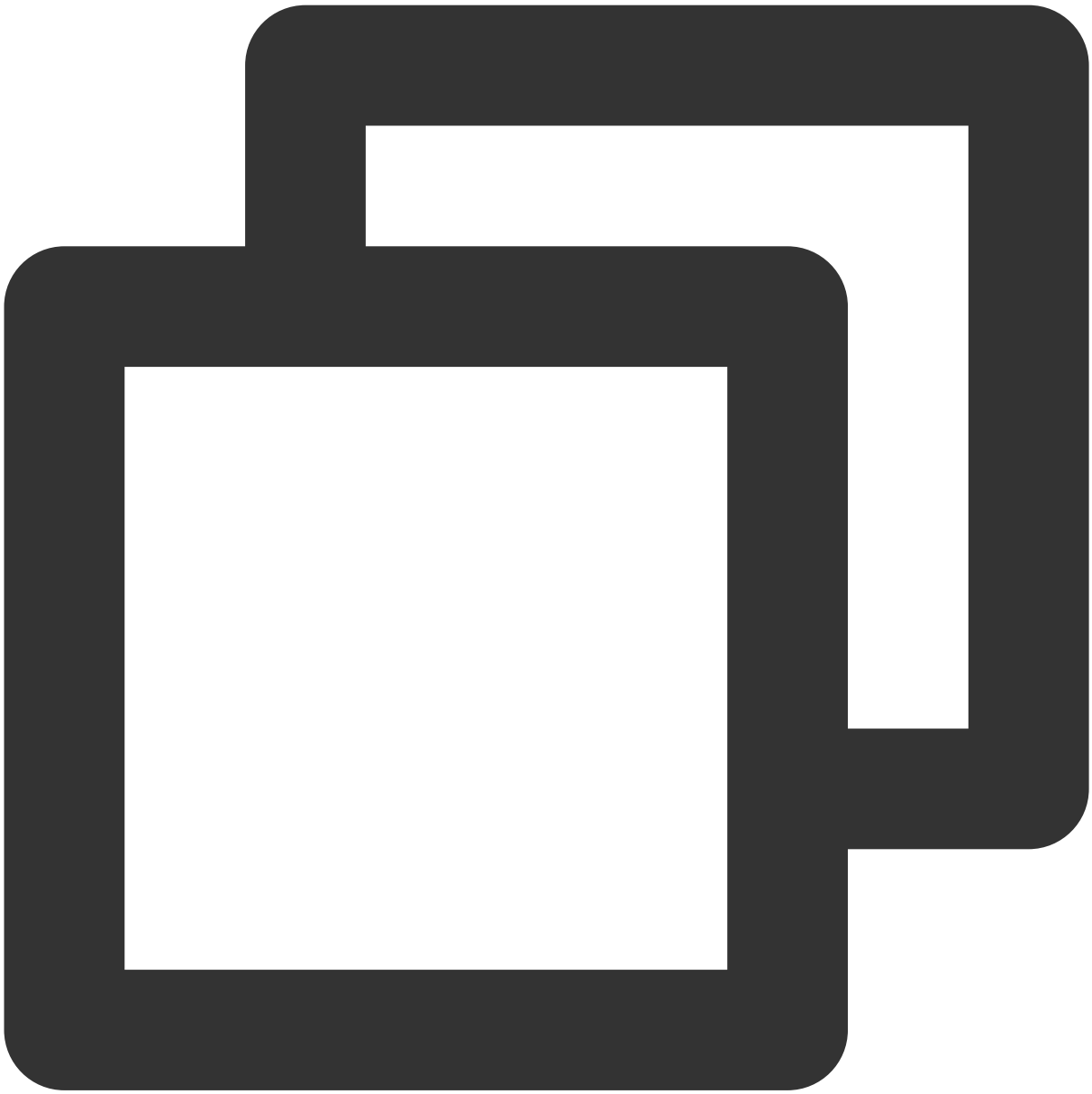
```
pod install
```

注意：

1. 使用本地集成方案时，如需升级时需要从 Github 获取最新的组件代码，覆盖您本地项目的 TUIKit 目录。
2. 当私有化修改和远端有冲突时，需要手动合并，处理冲突。
3. TUIKit 插件需要依赖 TUICore 的版本，请确保插件版本和 "../TUIKit/TUICore/TUICore.spec" 中的 spec.version 一致。

第三方库依赖

TUIKit 依赖的第三方库的最低版本如下，如果您的版本较低，请升级到最新版本。



```
- Masonry (1.1.0)
- MJExtension (3.4.1)
- MJRefresh (3.7.5)
- ReactiveObjC (3.1.1)
- SDWebImage (5.18.11):
  - SDWebImage/Core (= 5.18.11)
- SDWebImage/Core (5.18.11)
```

```
- SnapKit (5.6.0)
- SSZipArchive (2.4.3)
```

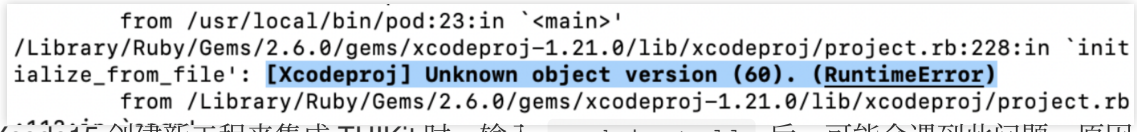
构建基础界面

集成 TUIKit 完成后，如果您想要继续构建聊天、会话等基础界面，请参考文档：构建聊天界面、构建会话列表等。

常见问题

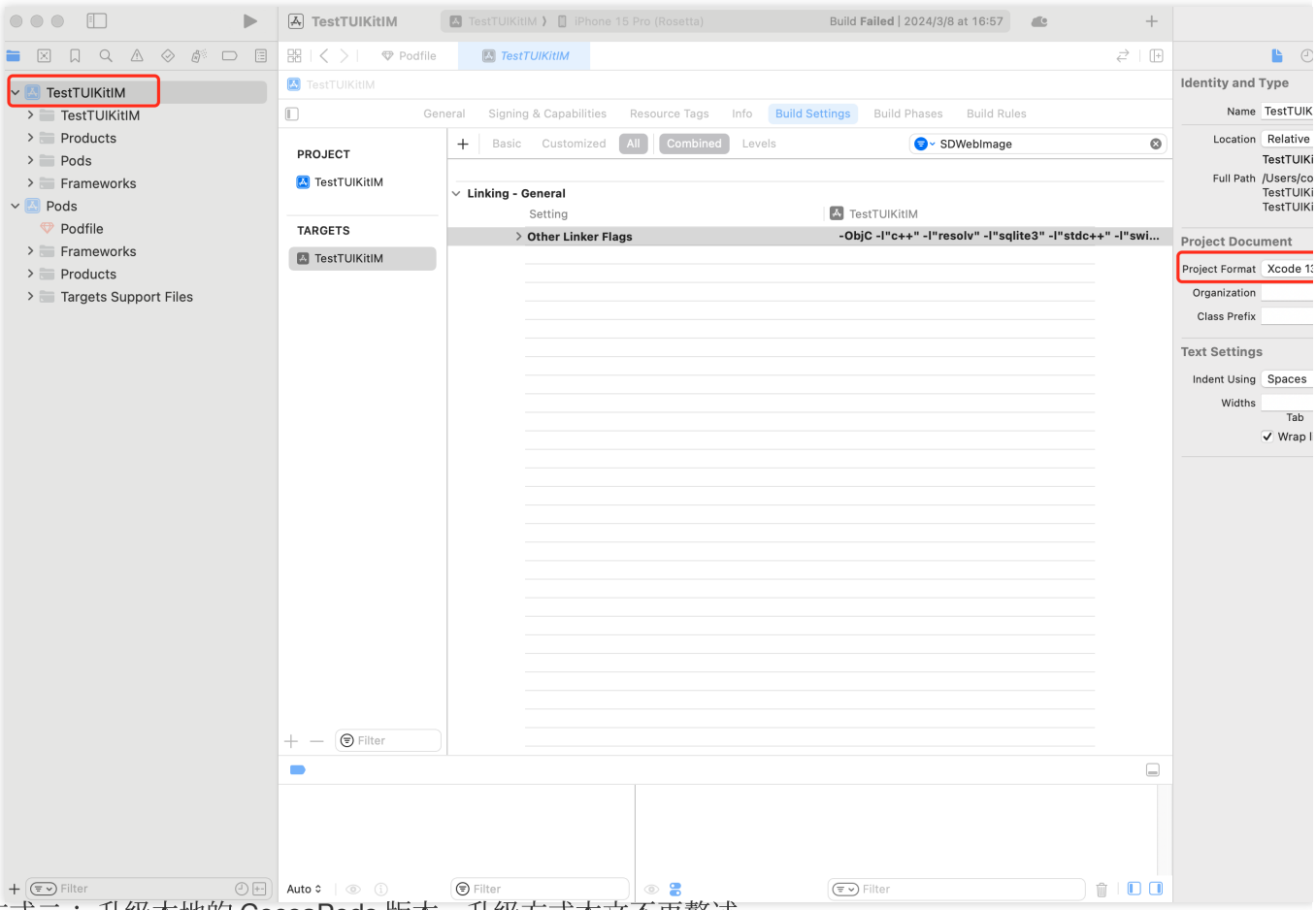
Xcode15 常见问题

集成时报错 [Xcodeproj] Unknown object version (60). (RuntimeError)



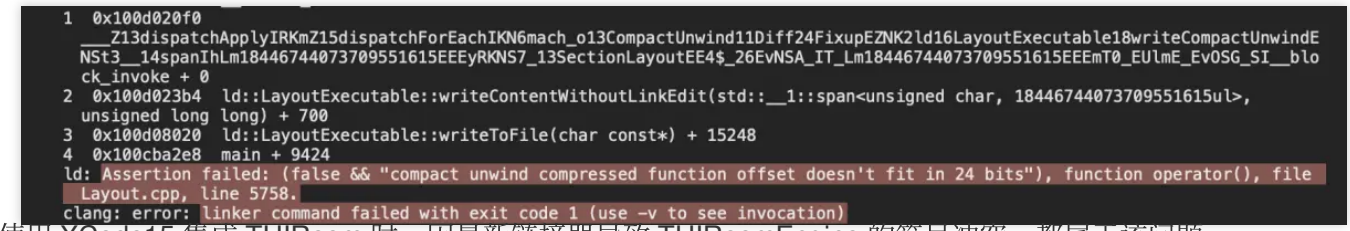
使用 Xcode15 创建新工程来集成 TUIKit 时，输入 `pod install` 后，可能会遇到此问题，原因是使用了较旧版本的 CocoaPods ，此时有两种解决办法：

解决方式一： 修改 Xcode 工程的 `Project Format` 版本为 Xcode13.0。

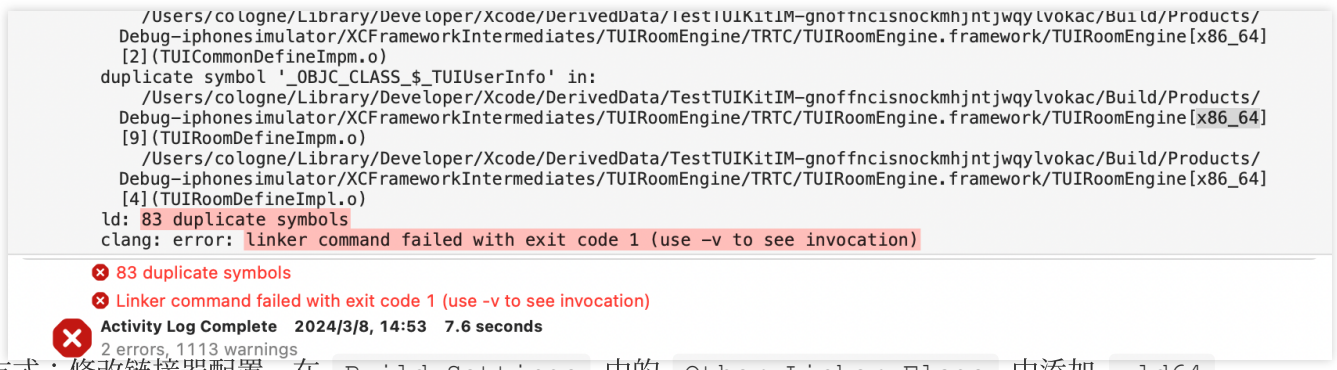


解决方式二：升级本地的 CocoaPods 版本，升级方式本文不再赘述。

Assertion failed: (false && "compact unwind compressed function offset doesn't fit in 24 bits"), function operator(), file Layout.cpp.

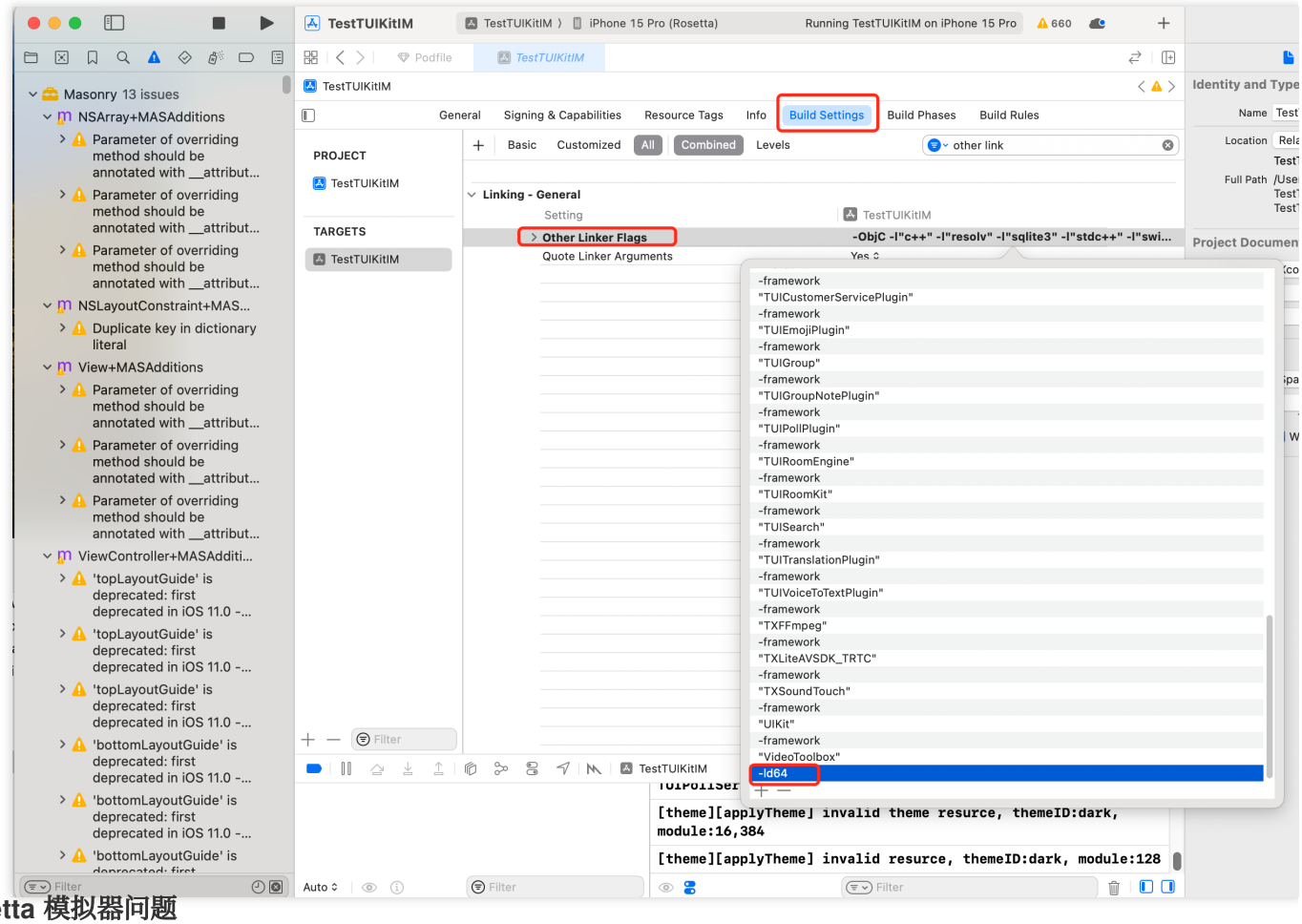


或是使用 XCode15 集成 TUIRoom 时，因最新链接器导致 TUIRoomEngine 的符号冲突，都属于该问题。



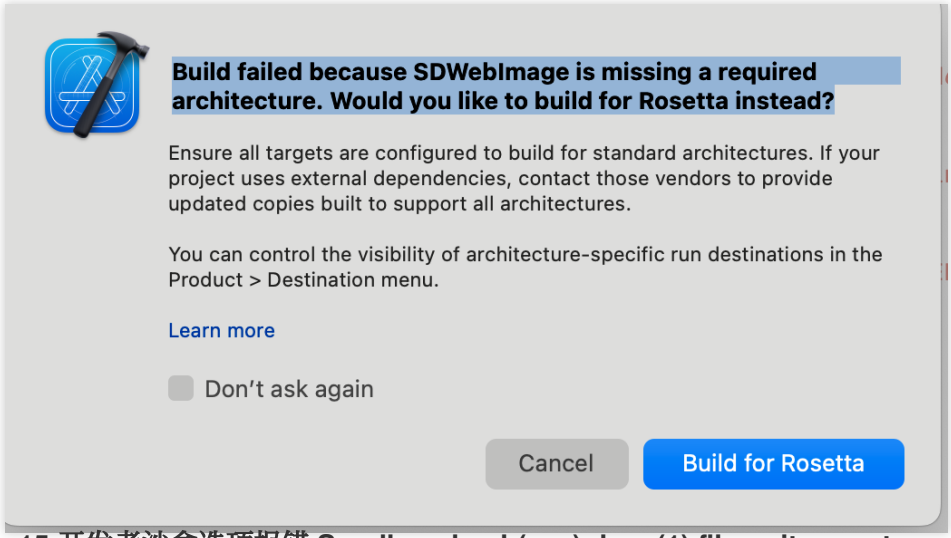
解决方式：修改链接器配置。在 `Build Settings` 中的 `Other Linker Flags` 中添加 `-ld64` 。

官方资料： https://developer.apple.com/forums/thread/735426

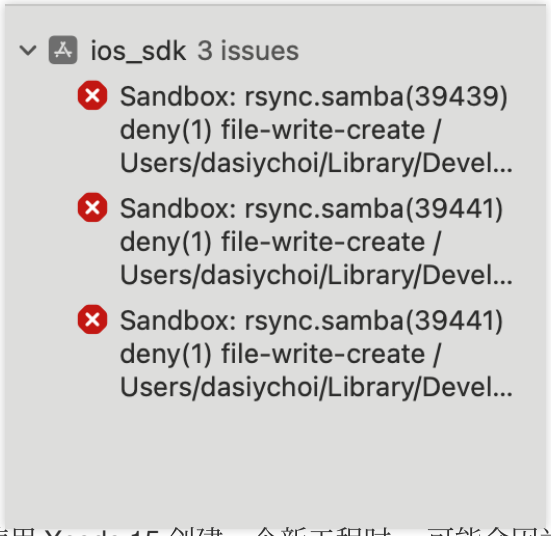


Rosetta 模拟器问题

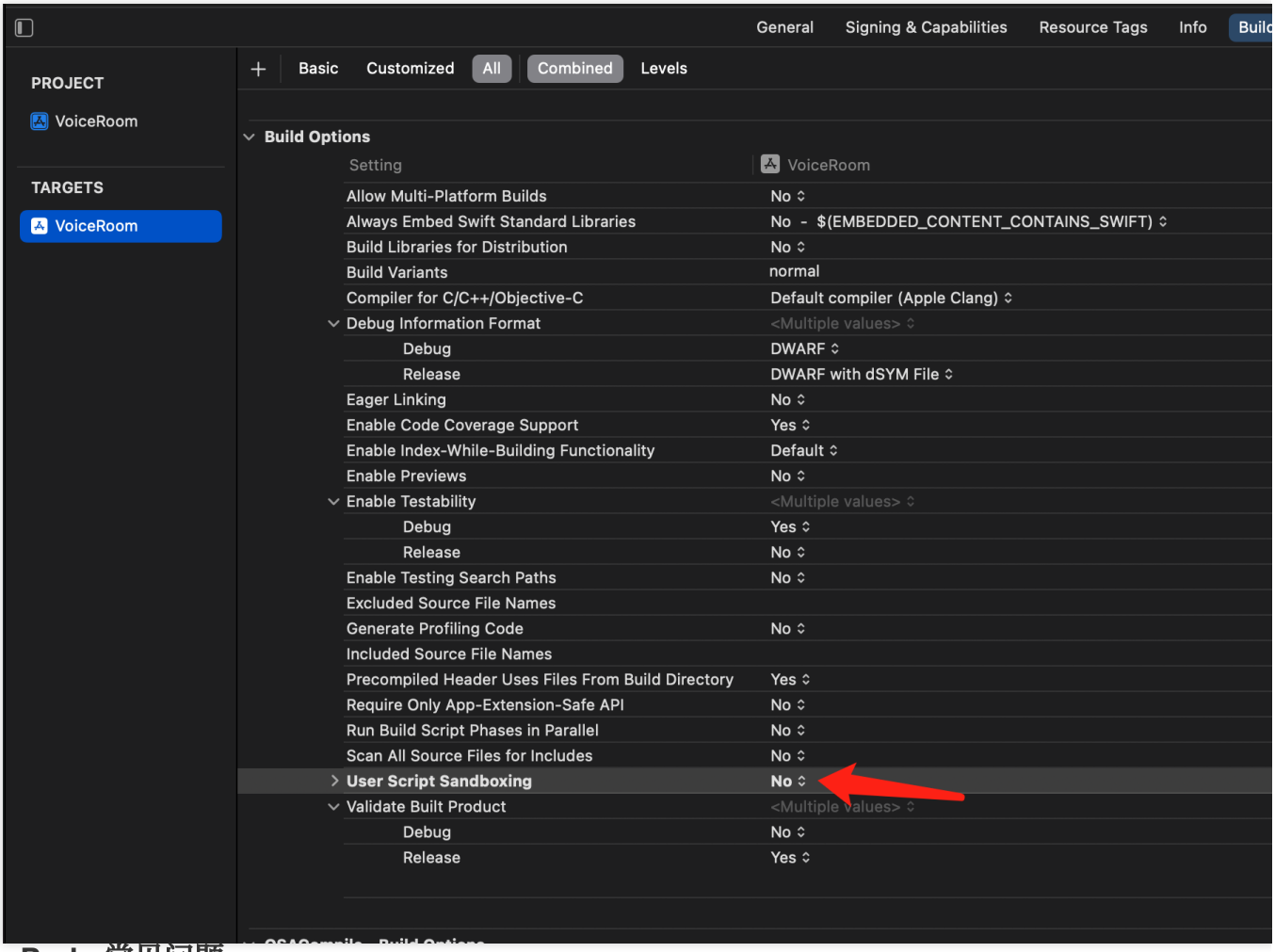
使用苹果芯片（m1\\m2等系列芯片）时会遇到这种弹框。原因是包括 SDWebImage 在内的三方库，并未支持 xcframework。不过苹果依旧给出了适配办法，就是在模拟器上开启 Rosetta 设置， 一般情况下编译时会自动弹出 Rosetta 选项。



Xcode 15 开发者沙盒选项报错 Sandbox: bash(xxx) deny(1) file-write-create



当您使用 Xcode 15 创建一个新工程时， 可能会因为此选项导致编译运行失败，建议您关闭此选项。

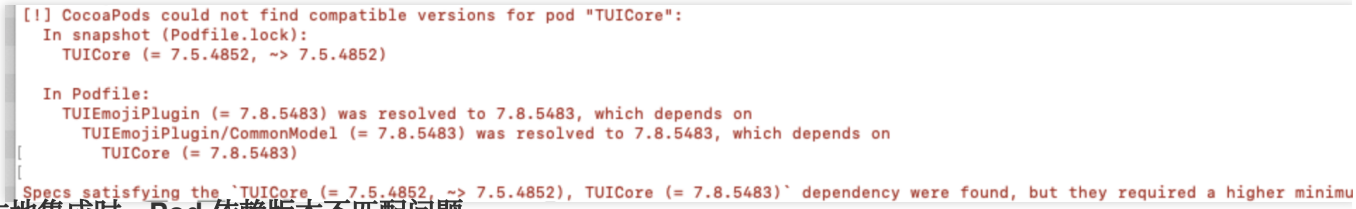


CocoaPods 常见问题

使用远端集成时，Pod 依赖版本不匹配问题

若您使用远端 CocoaPods 集成时，出现 Podfile.lock 和 插件依赖的 TUICore 版本不一致时，

此时请删除 Podfile.lock 文件， 并使用 `pod repo update` 更新本地代码仓库， 之后使用 `pod update` 重新更新即可。



使用本地集成时，Pod 依赖版本不匹配问题

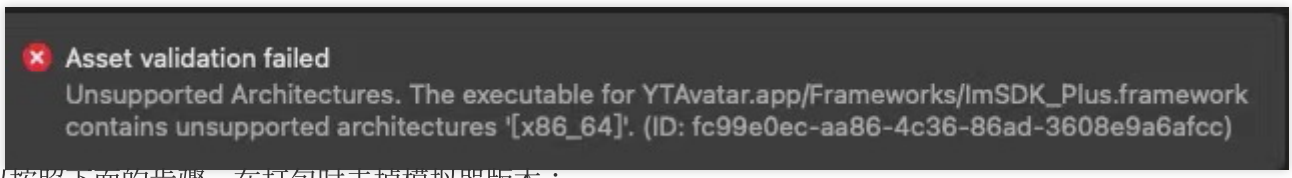
若您使用本地 DevelopmentPods 集成时出现插件依赖的 TUICore 版本较新，但本地 Pod 依赖的版本号是 1.0.0 ，此时请您参考 Podfile_local 和 TUICore.spec 修改。插件需要跟随版本，需要和 TUICore.spec 中一致。

第一次使用本地集成时，建议您下载我们的示例 Demo 工程，将 Podfile 文件内容替换为 Podfile_local 的内容，执行 `Pod update` 后相互参照。

上架常见问题

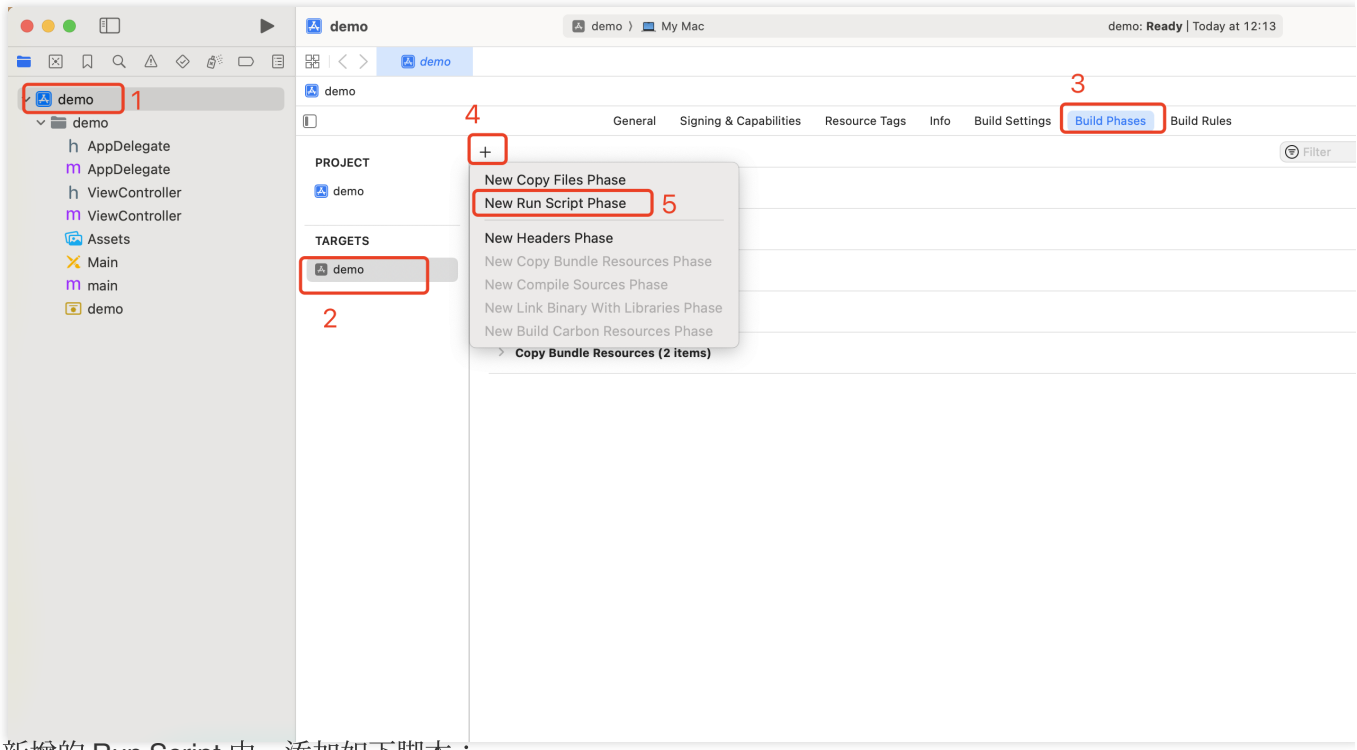
上架 Appstore 时打包失败，提示 Unsupported Architectures。

问题现象如下图，打包时提示 ImSDK_Plus.framework 中包含了 Appstore 不支持的 x86_64 模拟器版本。该问题是由于 SDK 为了方便开发者调试，发布时会默认带上模拟器版本。

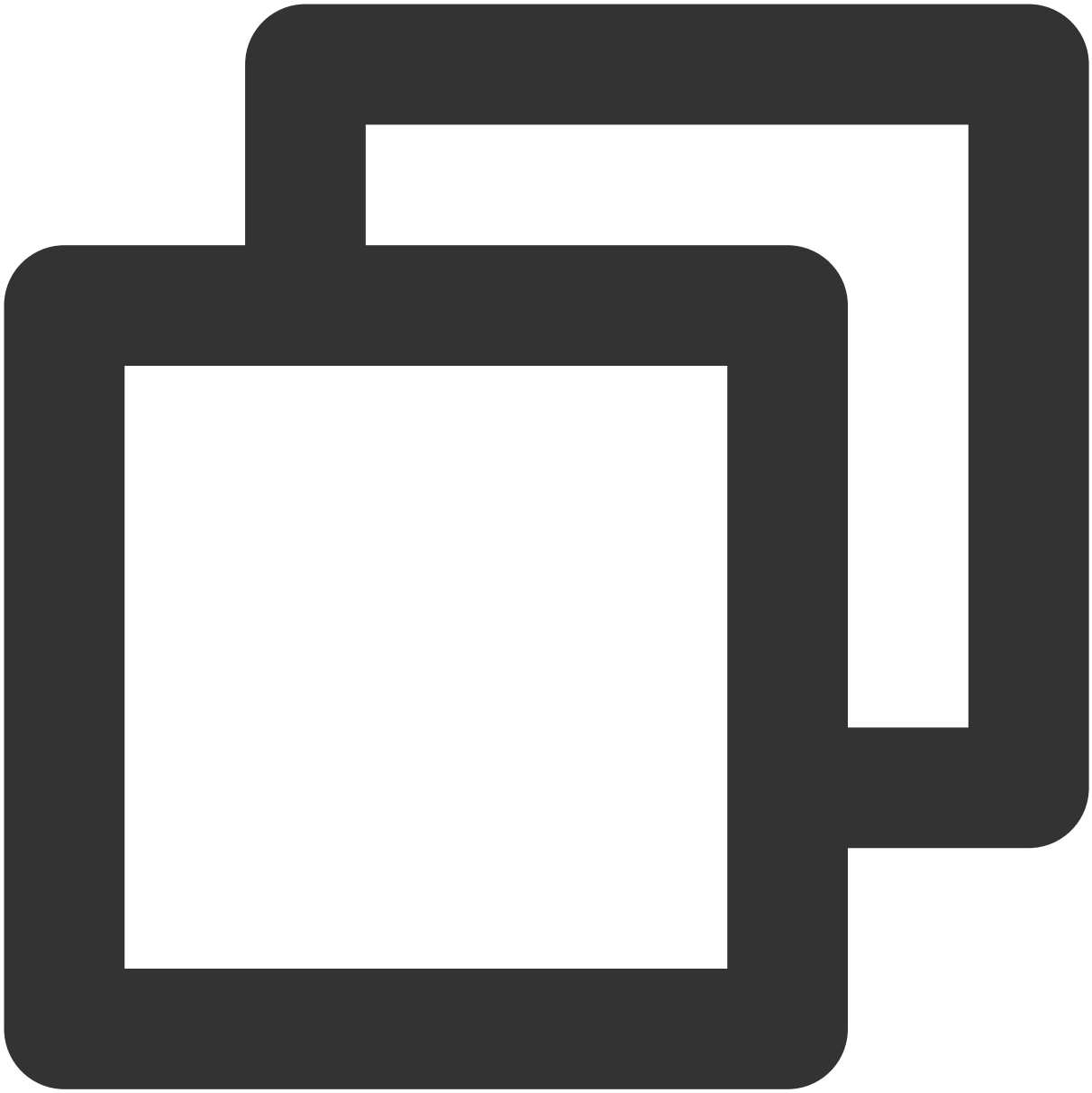


您可以按照下面的步骤，在打包时去掉模拟器版本：

1.1 选中您工程的 Target，并点击 `Build Phases` 选项，在当前面板中添加 `Run Script` ；

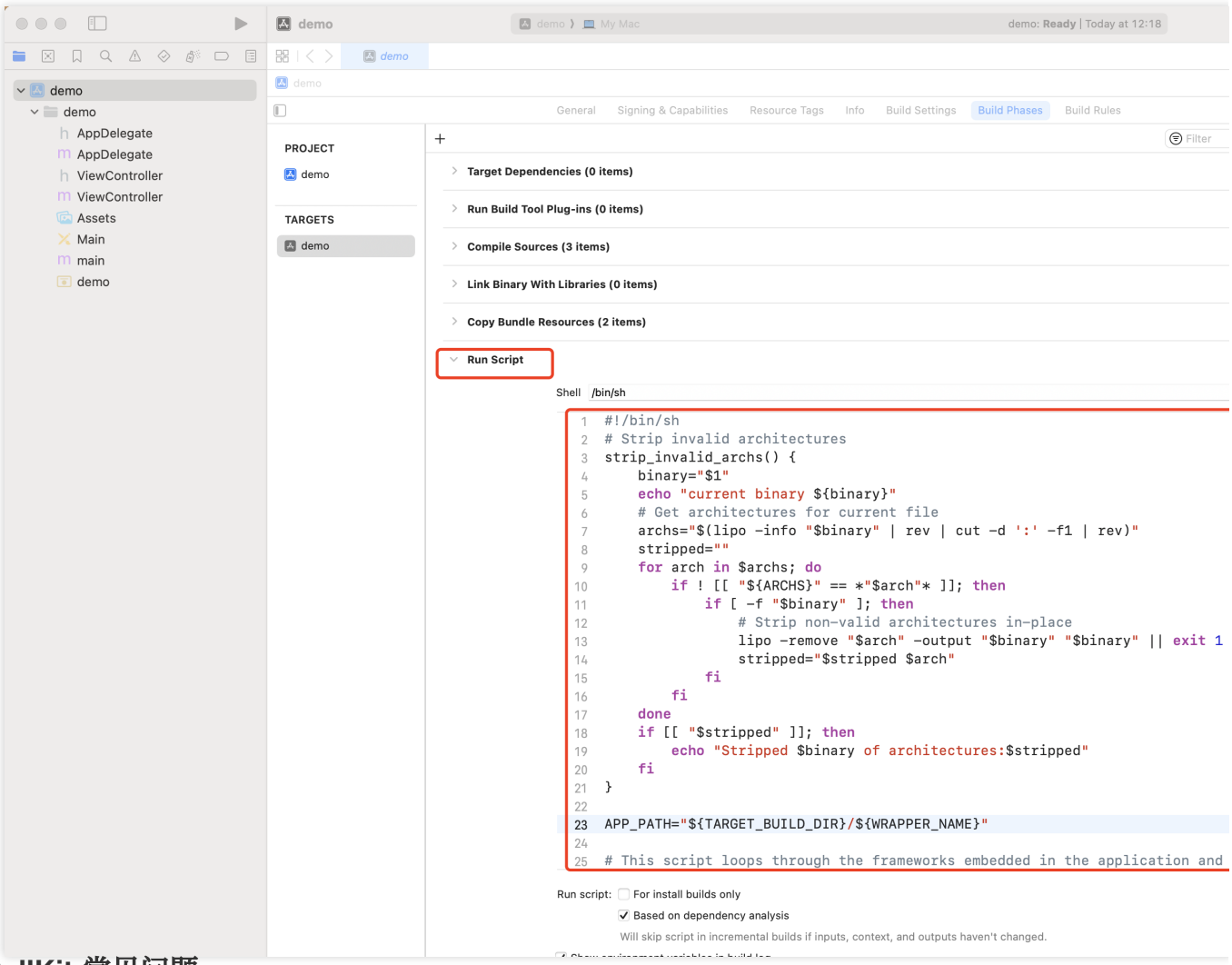


1.2 在新增的 Run Script 中，添加如下脚本：



```
#!/bin/sh

# Strip invalid architectures
strip_invalid_archs() {
    binary="$1"
    echo "current binary ${binary}"
    # Get architectures for current file
    archs="$(lipo -info "$binary" | rev | cut -d ':' -f1 | rev)"
    stripped=""
    for arch in $archs; do
        if ! [[ "${ARCHS}" == *"$arch"* ]]; then
```

```
            if [ -f "$binary" ]; then
                # Strip non-valid architectures in-place
                lipo -remove "$arch" -output "$binary" "$binary" || exit 1
                stripped="$stripped $arch"
            fi
        fi
    done
    if [[ "$stripped" ]]; then
        echo "Stripped $binary of architectures:$stripped"
    fi
}

APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
    FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecu
    FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
    echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"
    strip_invalid_archs "$FRAMEWORK_EXECUTABLE_PATH"
done
```

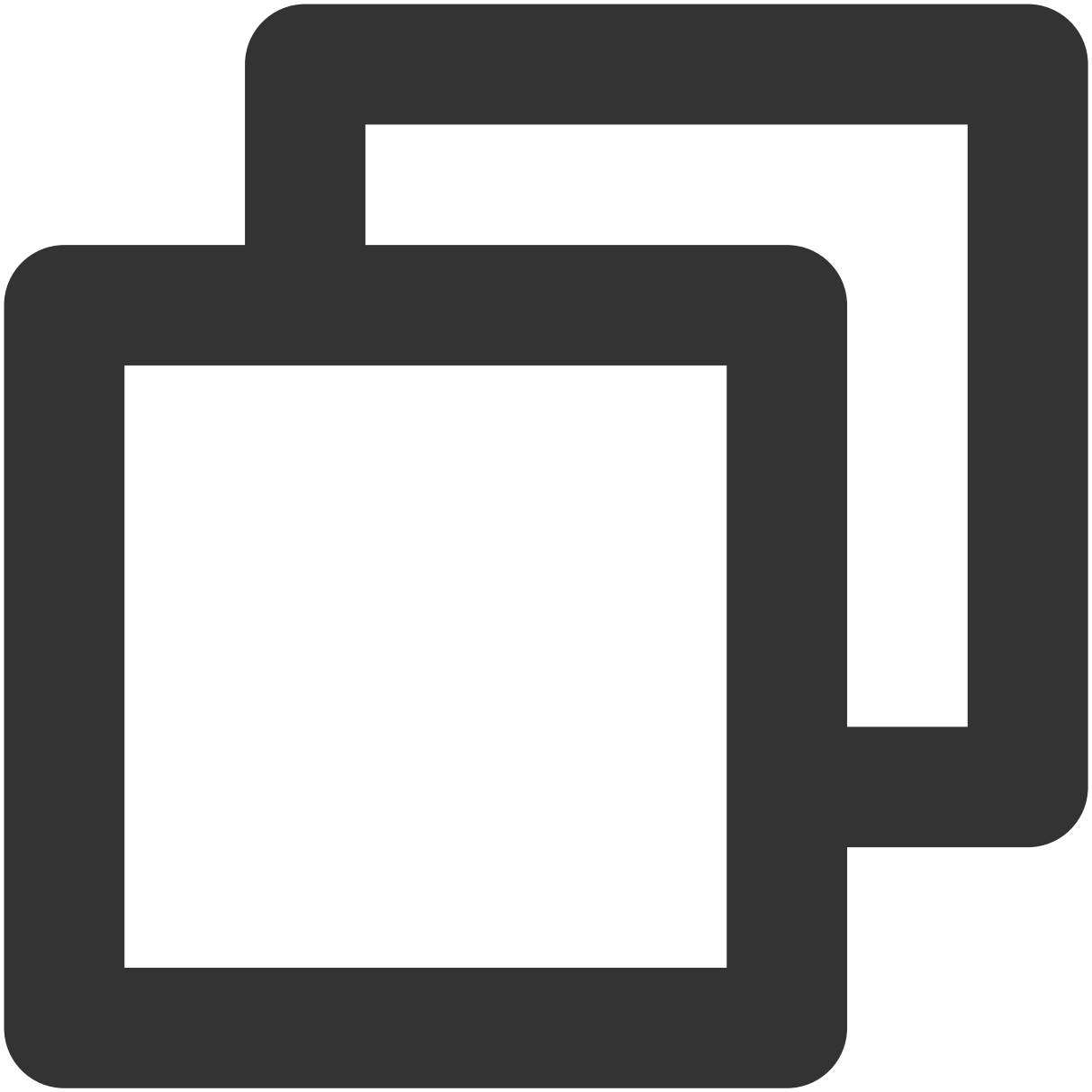


TUICallKit 常见问题

TUICallKit 和自己集成的音视频库冲突了？

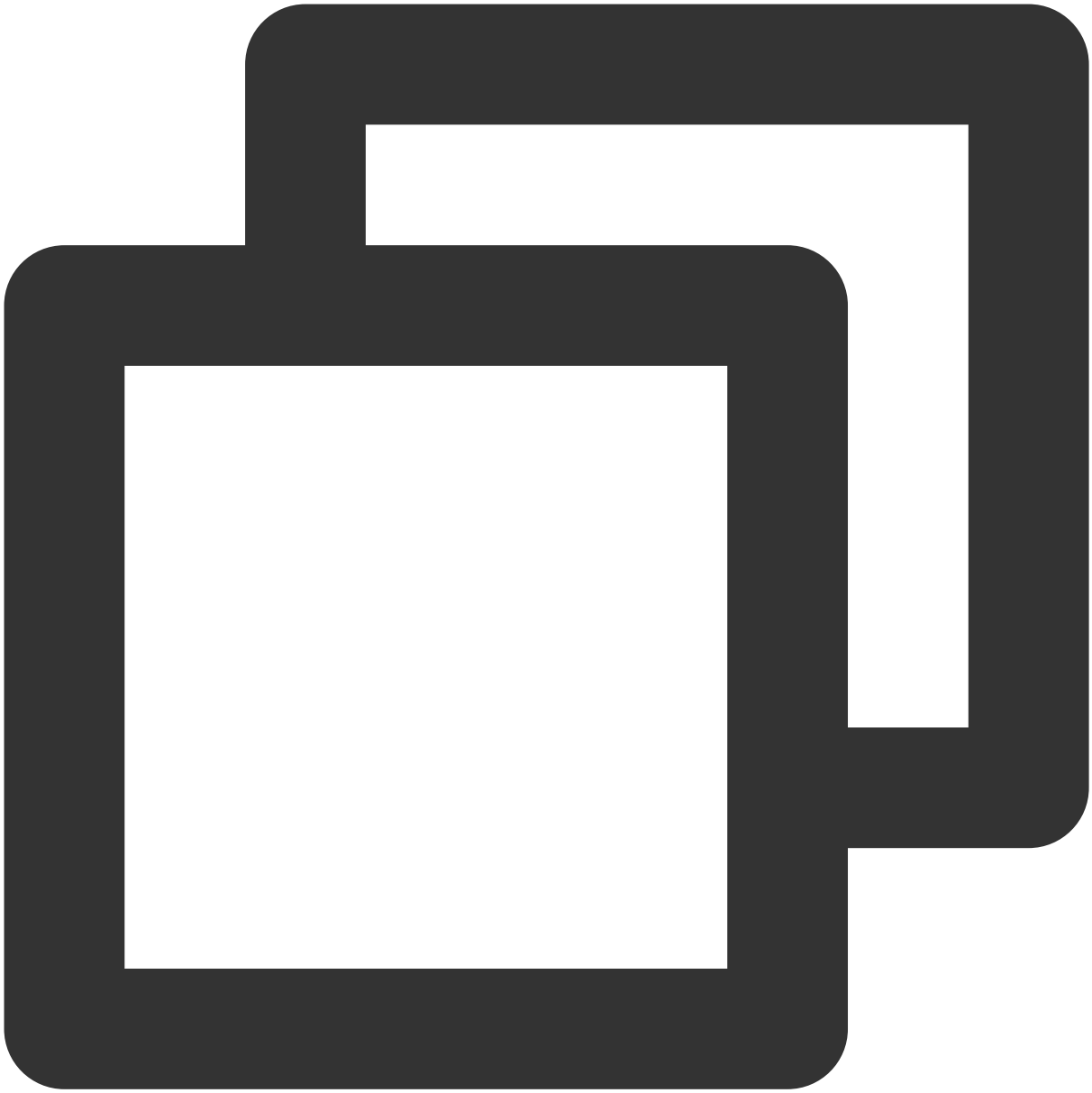
腾讯云的 音视频库 不能同时集成，可能存在符号冲突，可以按照下面的场景处理。

1.1 如果您使用了 `TXLiteAVSDK_TRTC` 库，不会发生符号冲突。可直接在 Podfile 文件中添加依赖：



```
pod 'TUICallKit'
```

1.2 如果您使用了 `TXLiteAVSDK_Professional` 库，会产生符号冲突。您可在 Podfile 文件中添加依赖：



```
pod 'TUICallKit/Professional'
```

1.3 如果您使用了 `TXLiteAVSDK_Enterprise` 库，会产生符号冲突。建议升级到 `TXLiteAVSDK_Professional` 后使用 `TUICallKit/Professional` 。

通话邀请的超时时间默认是多久？

通话邀请的默认超时时间是 30 秒。

在邀请超时时间内，被邀请者如果离线再上线，能否立即收到邀请？

1.1 如果是单聊通话邀请，被邀请者离线再上线可以收到通话邀请，TUIKit 内部会自动唤起通话邀请界面。

1.2 如果是群聊通话邀请，被邀请者离线再上线后会自动拉取最近 30 秒内的邀请，TUIKit 会自动唤起群通话界面。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Web & H5（react)

最近更新时间：2024-08-14 16:18:22

chat-uikit-react 介绍

chat-uikit-react 是基于腾讯云 Chat SDK 的一款 react UI 组件库，它提供了一些通用的 UI 组件，包含会话、聊天、群组等功能。基于这些精心设计的 UI 组件，您可以快速构建优雅的、可靠的、可扩展的 Chat 应用。
您可以直接体验下面的聊天。同时，您可以通过 **体验沙箱** 快速体验在线代码实现。

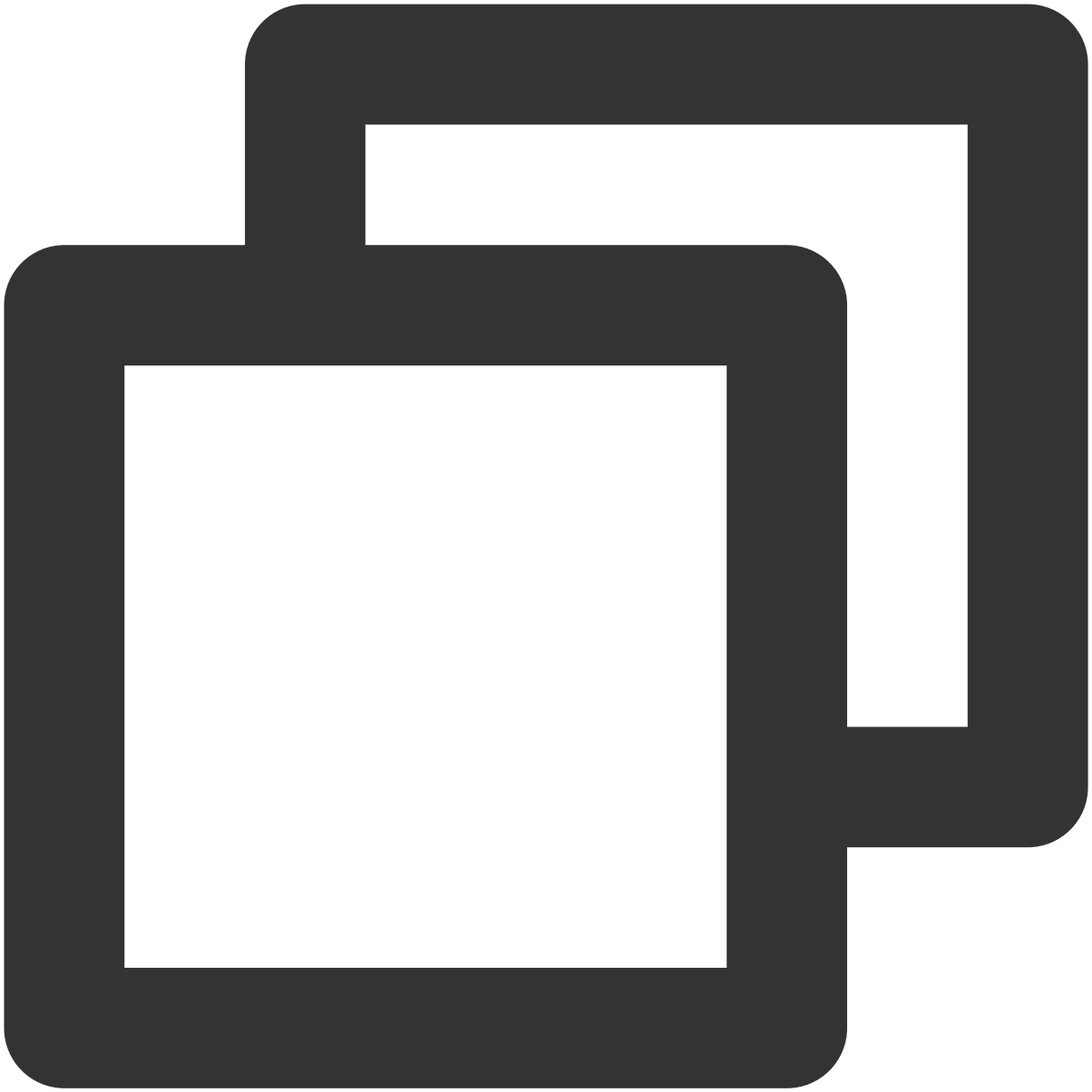
开发环境要求

React version 18+（不支持 17.x 版本）
TypeScript
Node.js version 16+
npm（版本请与 node 版本匹配）

chat-uikit-react 集成

步骤1：创建项目

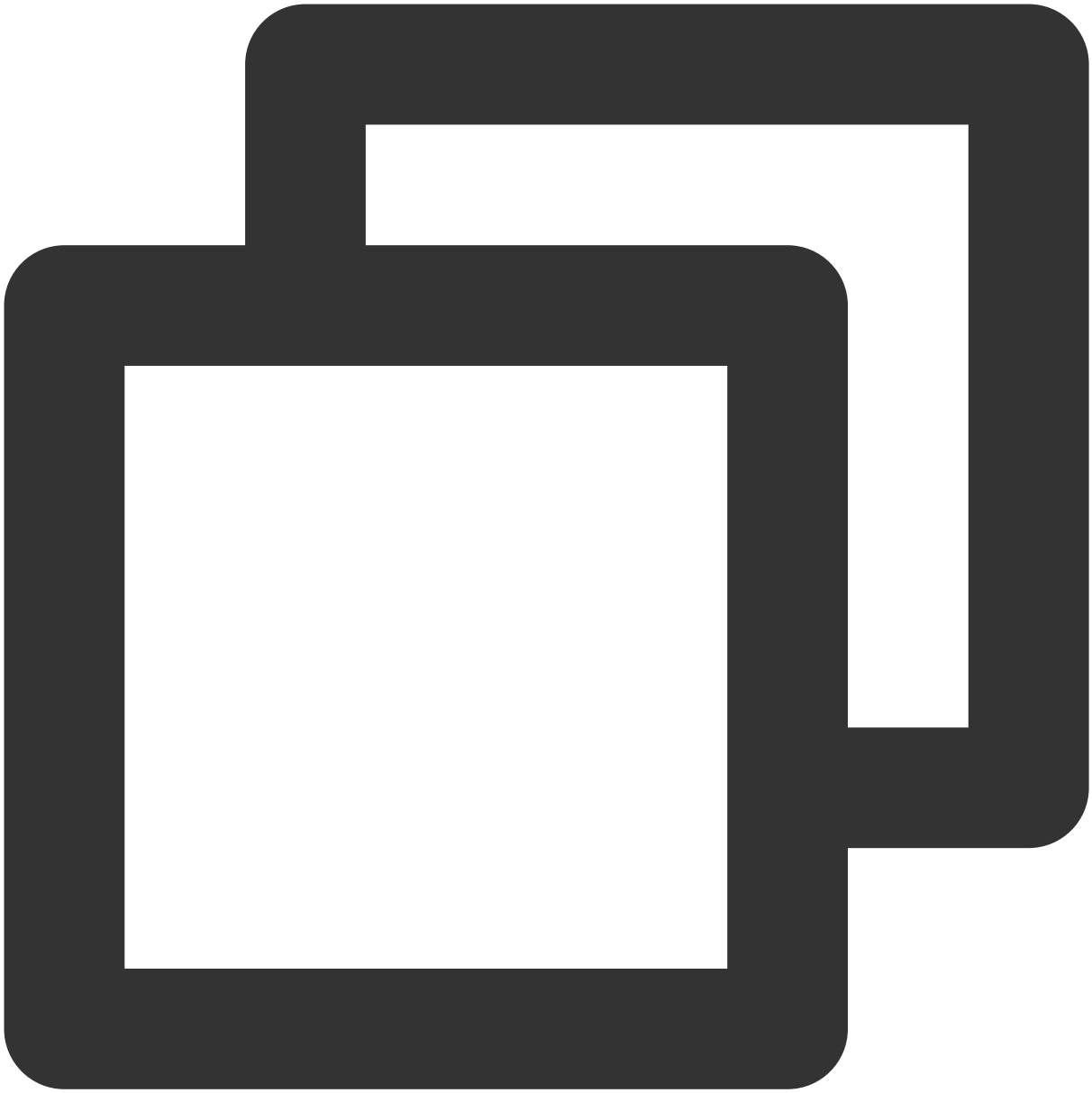
创建一个新的 React 项目，您可自行选择是否需要使用 ts 模版。



```
npx create-react-app sample-chat --template typescript
```

创建项目完成后，切换到项目所在目录。

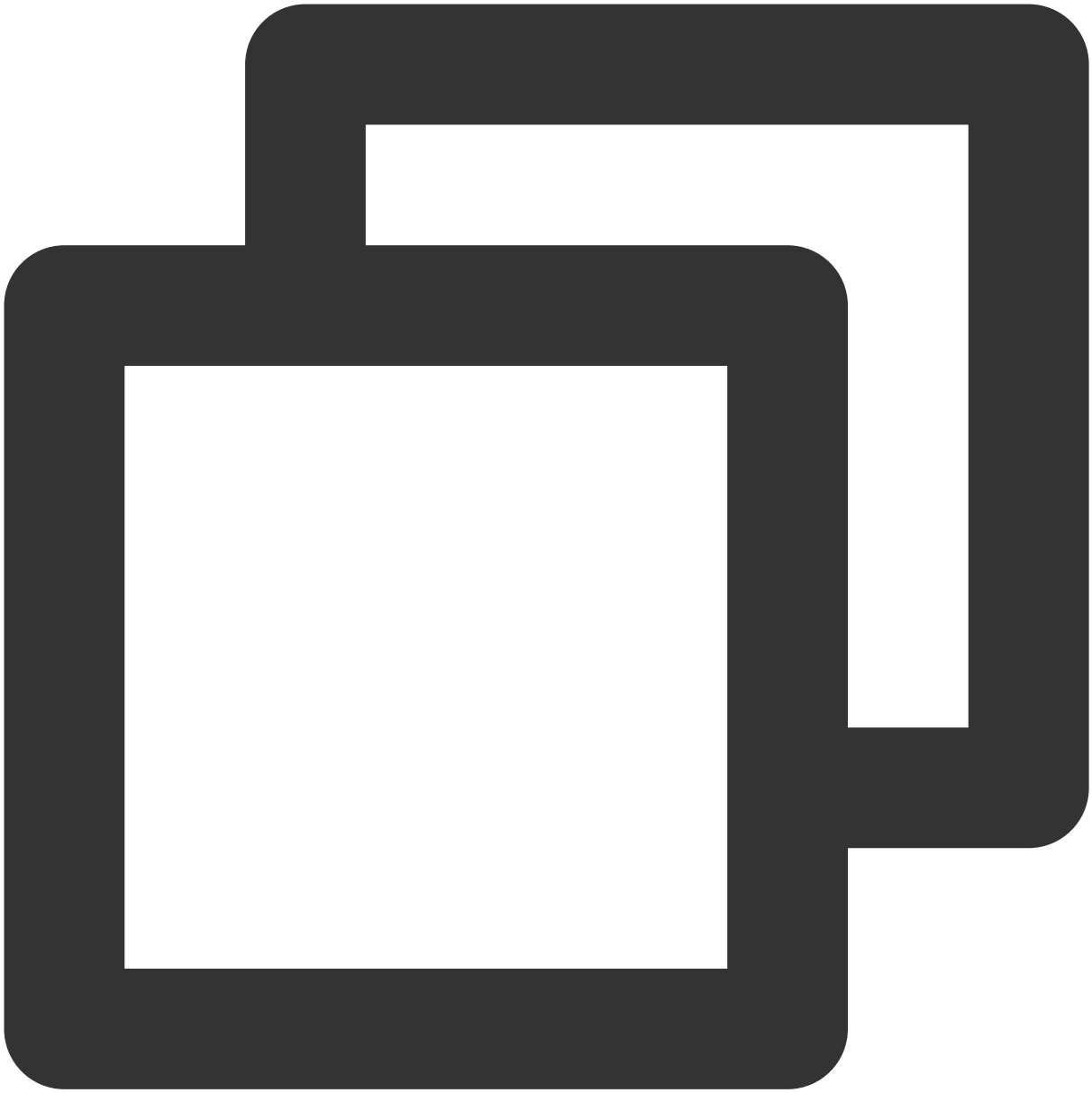
版权所有：腾讯云计算（北京）有限责任公司



```
cd sample-chat
```

步骤2：下载 chat-uikit-react 组件

通过 npm 方式下载 chat-uikit-react 并在项目中使用，另外在 GitHub 中也提供相关的 开源代码，您也可在此基础上进行开发您自己的组件库。



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

步骤3：引入 chat-uikit-react 组件

注意：

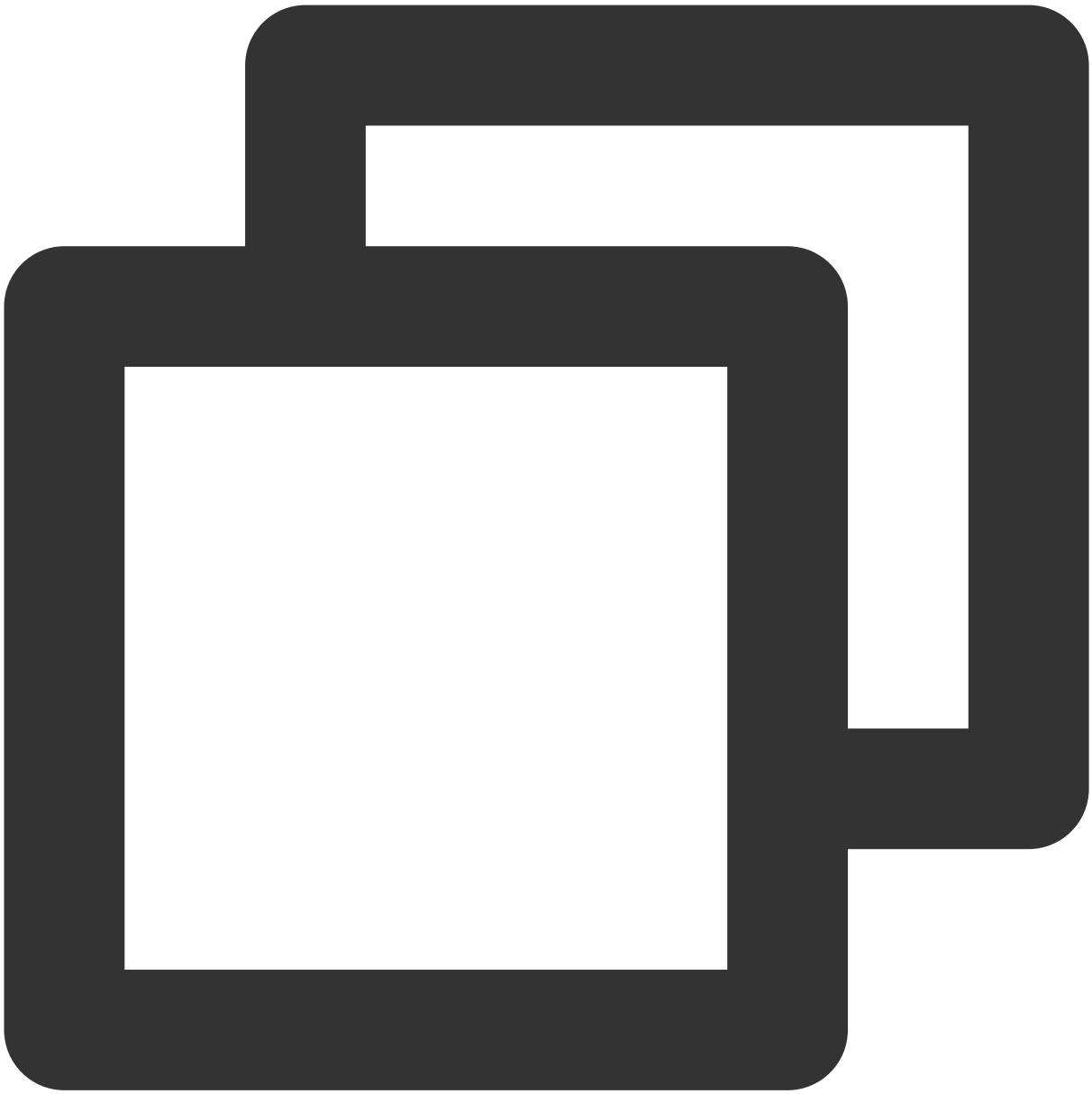
以下代码中未填入 `SDKAppID` 、 `userID` 和 `userSig` ，需在 步骤4 中获取相关信息后进行替换。

替换 App.tsx 中的内容，或者您可以新建一个组件引入。

示例 1: 集成 ConversationList & Chat

示例 2: 集成 Chat

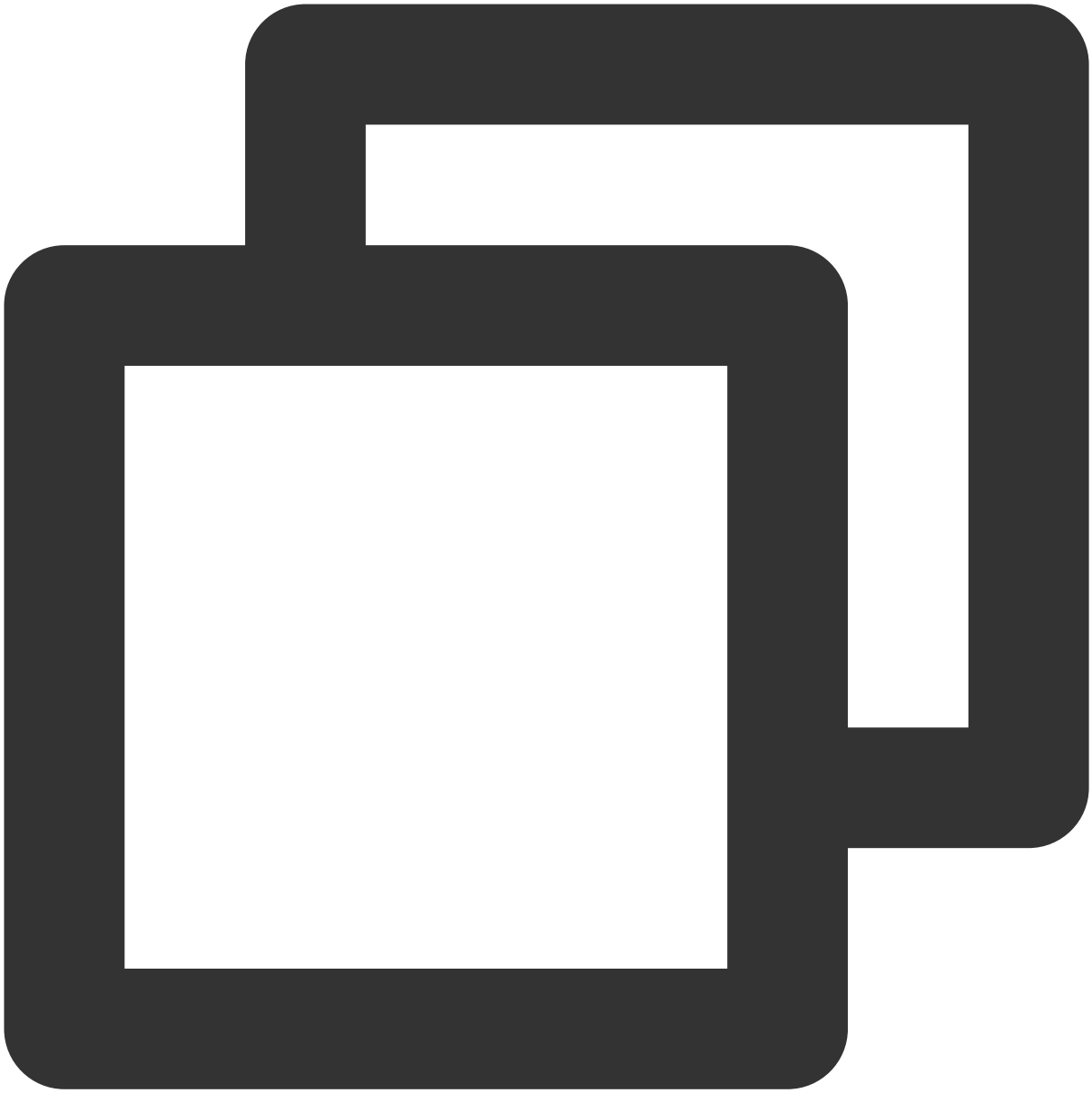
→ 在沙箱中体验



```
import { useEffect } from 'react';
import { TUIChat, TUIChatHeader, TUIConversation, TUIKit, TUIManage, TUIMessageInpu
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { TUILogin } from '@tencentcloud/tui-core';
import '@tencentcloud/chat-uikit-react/dist/cjs/index.css';

const config = {
  SDKAppID: 0, // Your SDKAppID, Get it from Step 4
  userID: 'YOUR_USER_ID', // Login UserID, Get it from Step 5
```

```
  userSig: 'YOUR_USER_SIG', // Your userSig, Get it from Step 5
}

export default function SampleChat() {
  const init = () => {
    TUILogin.login({
      ...config,
      useUploadPlugin: true
    }).then(() => {
      openChat();
    }).catch(() => { });
  }
  useEffect(() => {
    init();
  }, [])

  const openChat = () => {
    // 1v1 chat: conversationID = `C2C${userID}`
    // group chat: conversationID = `GROUP${groupID}`
    const userID = 'administrator'; // userID: Recipient of the Message userID, Get
    const conversationID = `C2C${userID}`;
    TUIConversationService.switchConversation(conversationID);
  };

  // language support en-US / zh-CN / ja-JP / ko-KR
  return (
    <TUIKit language={"en-US"}>
      <div className="sample-chat-left-container">
        <TUIProfile className="sample-chat-profile" />
        <TUIConversation />
      </div>
      <TUIChat>
        <TUIChatHeader />
        <TUIMessageList />
        <TUIMessageInput />
      </TUIChat>
      <TUIManage></TUIManage>
    </TUIKit>
  )
}
```

→ 在沙箱中体验



```
import { useEffect } from 'react';
import { TUIChat, TUIChatHeader, TUIKit, TUIMessageInput, TUIMessageList } from '@t
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { TUILogin } from '@tencentcloud/tui-core';
import '@tencentcloud/chat-uikit-react/dist/cjs/index.css';

const config = {
  SDKAppID: 0, // Your SDKAppID, Get it from Step 4
  userID: 'YOUR_USER_ID', // Login UserID, Get it from Step 5
  userSig: 'YOUR_USER_SIG', // Your userSig, Get it from Step 5
}
```

```
export default function SampleChat() {
  const init = () => {
    TUILogin.login({
      ...config,
      useUploadPlugin: true
    }).then(() => {
      openChat();
    }).catch(() => { });
  }
  useEffect(() => {
    init();
  }, [])

  const openChat = () => {
    // 1v1 chat: conversationID = `C2C${userID}`
    // group chat: conversationID = `GROUP${groupID}`
    const userID = 'administrator'; // userID: Recipient of the Message userID, Get
    const conversationID = `C2C${userID}`;
    TUIConversationService.switchConversation(conversationID);
  };

  // language support en-US / zh-CN / ja-JP / ko-KR
  return (
    <TUIKit language={"en-US"}>
      <TUIChat>
        <TUIChatHeader />
        <TUIMessageList />
        <TUIMessageInput />
      </TUIChat>
    </TUIKit>
  )
}
```

说明：

conversationID：会话 ID。会话ID组成方式：

C2C${userID}（单聊）

GROUP${groupID}（群聊）

关于群聊：

通过调用 API createGroup 可获取 groupID

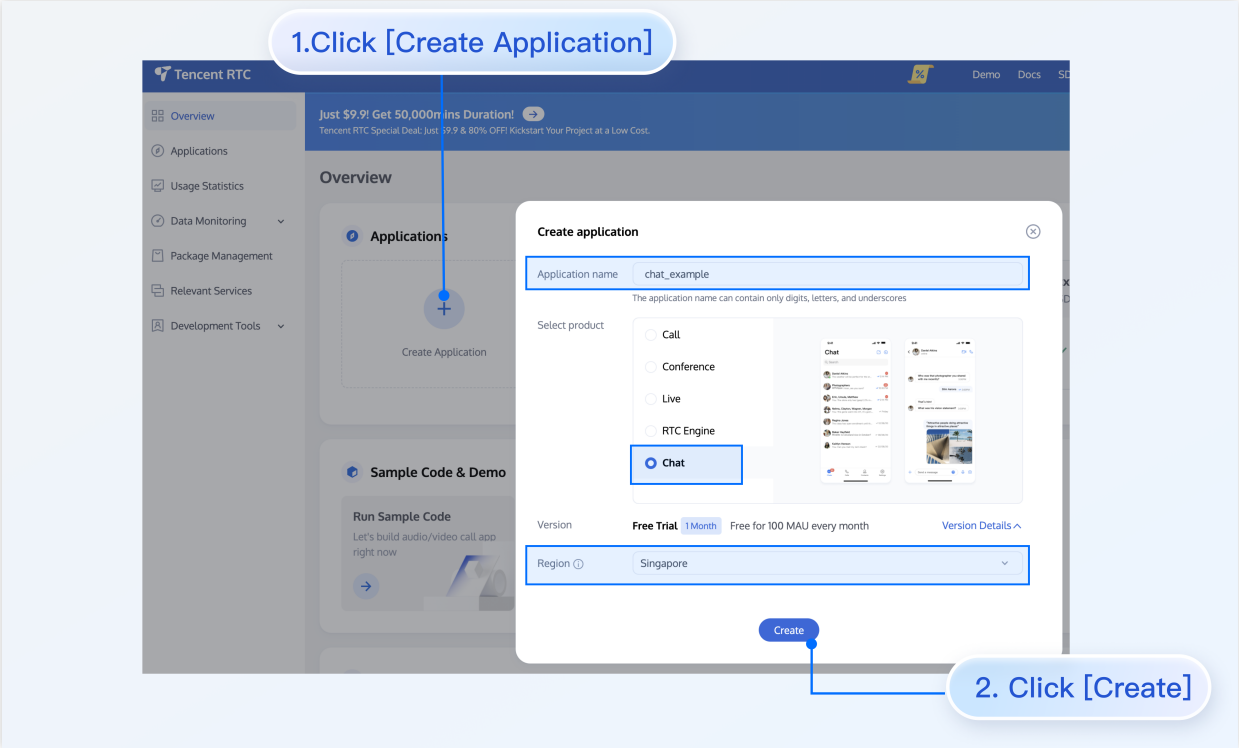
如果是直播群， 需要通过调用 API joinGroup 加入群，才可以进行聊天。

进入聊天

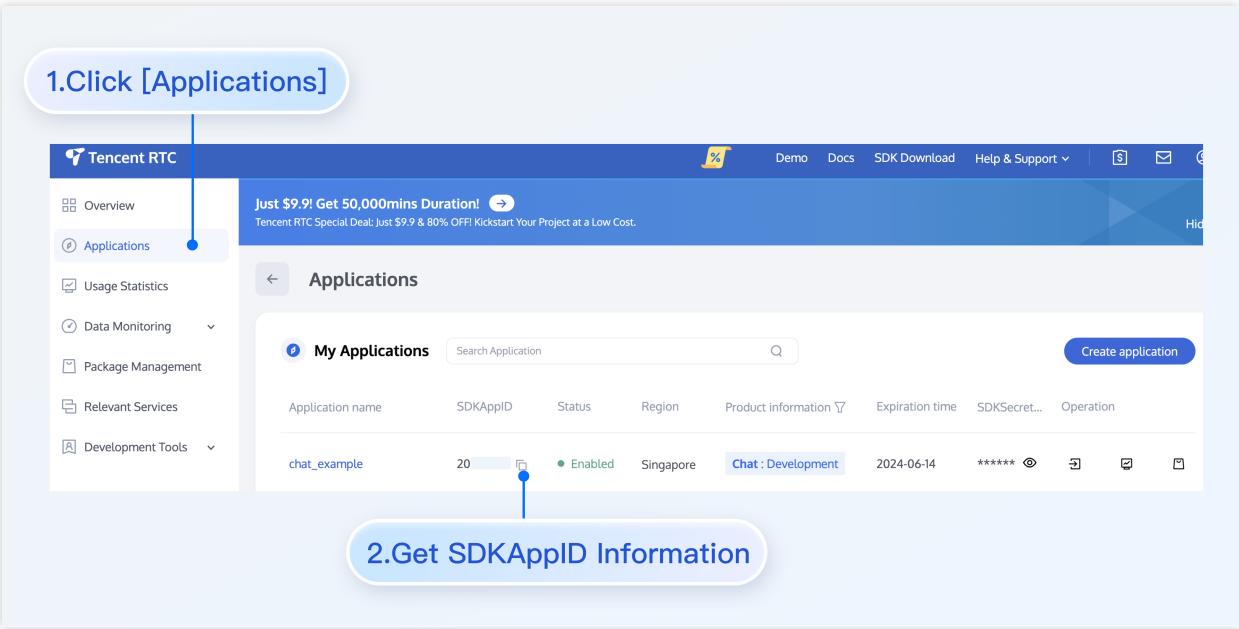
通过调用 API switchConversation，传入 conversationID 进入聊天页面。

步骤4：创建一个应用

1. 登录 Chat Console 。
2. 单击 **Create Application**，输入您的应用程序名称，然后单击 **Create**。



3. 创建完成后，您可以在控制台概览页面看到新应用的状态、服务版本、SDKAppID、创建时间、标签、过期时间。



步骤5：获取 userID 和 userSig

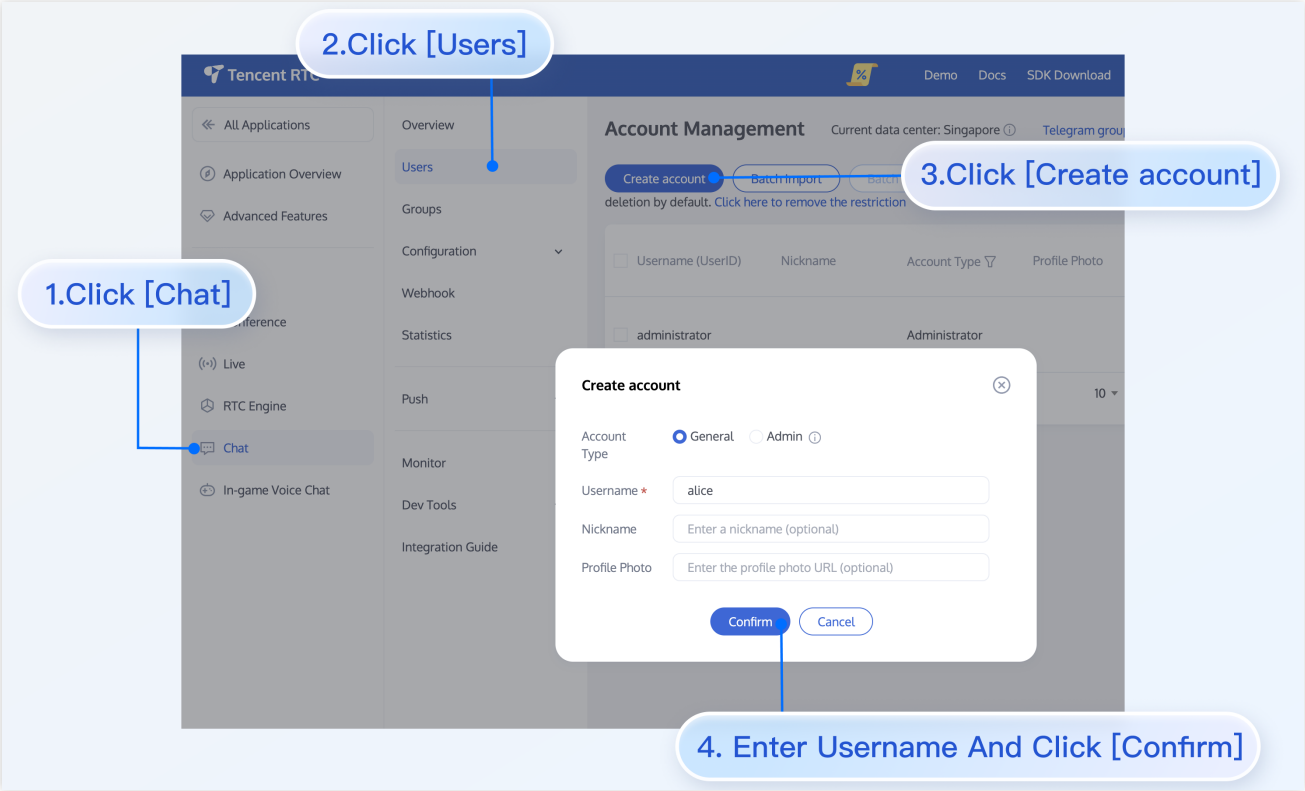
`userID`

单击进入您上面创建的 Application，会在左侧边栏看到 `Chat` 产品入口，单击进入。

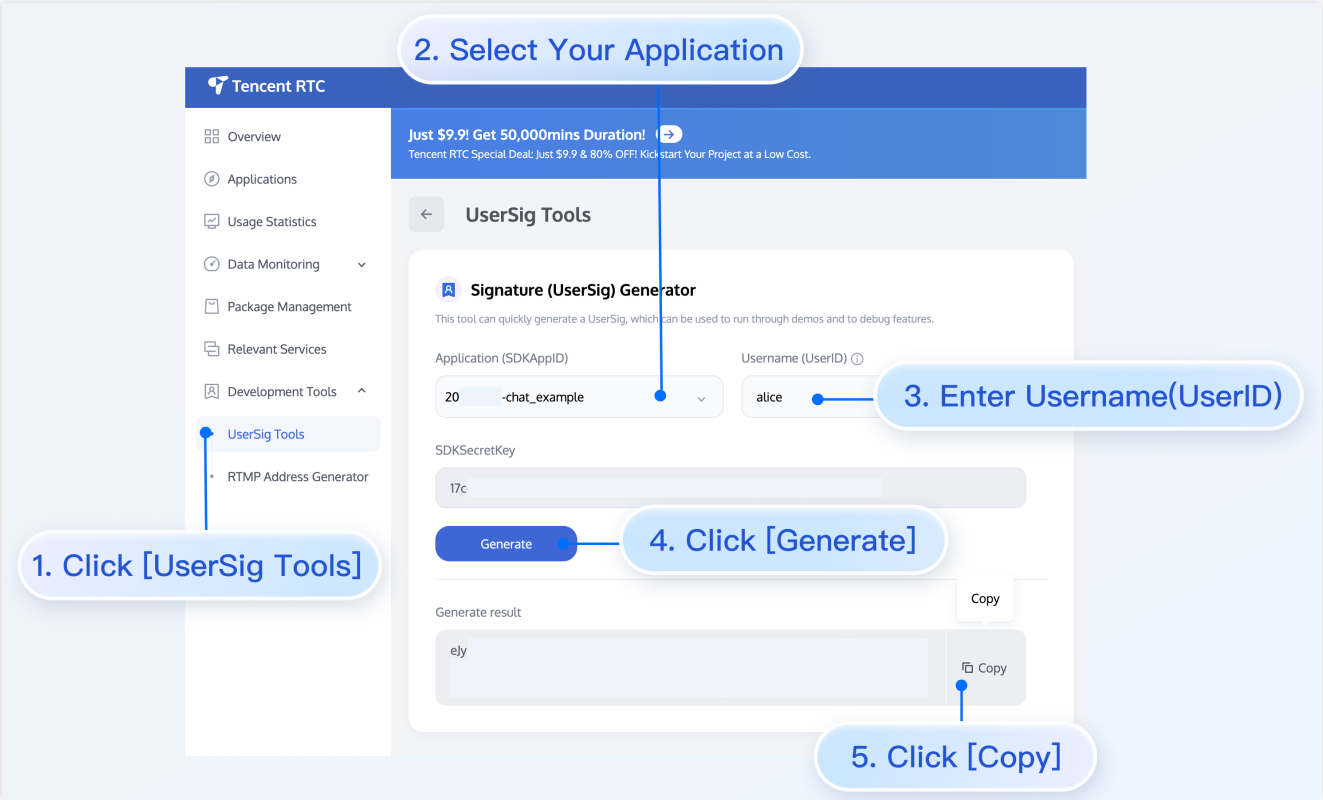
进入 Chat 产品子页面后，点击 `Users`，进入用户管理页面。

单击 `Create account`，弹出创建账号信息填写框。如果只是普通成员，我们建议您选择 `General` 类型。

为了您更好的体验消息收发等功能，建议您创建两个 userID。

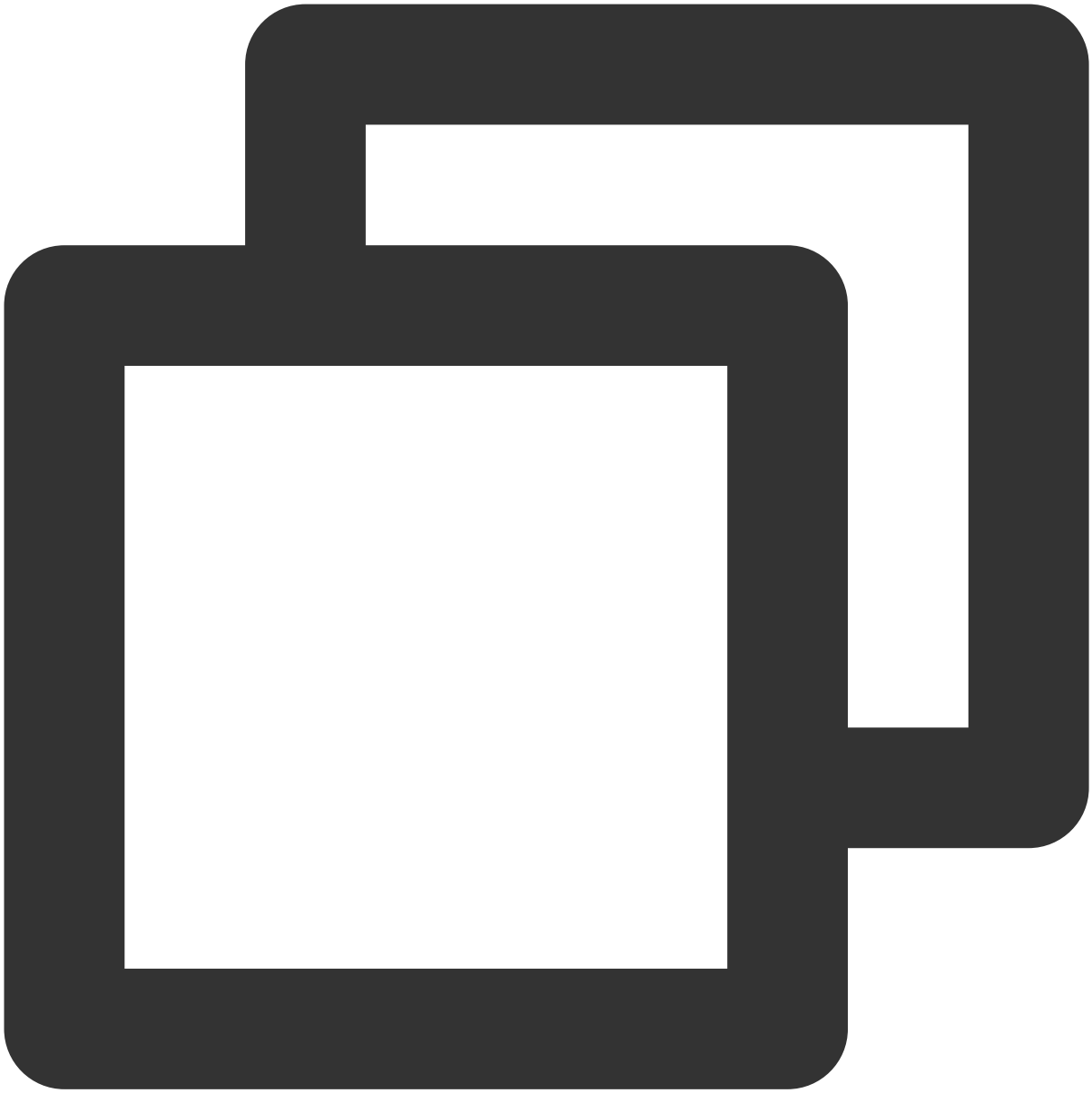


`userSig`，可使用控制台提供的开发工具实时生成，开发工具请点击 Chat Console > Development Tools > UserSig Tools > Signature (UserSig) Generator 。



步骤6：启动项目

替换 App.tsx 中的 SDKAppID、userID、userSig，然后运行命令如下：



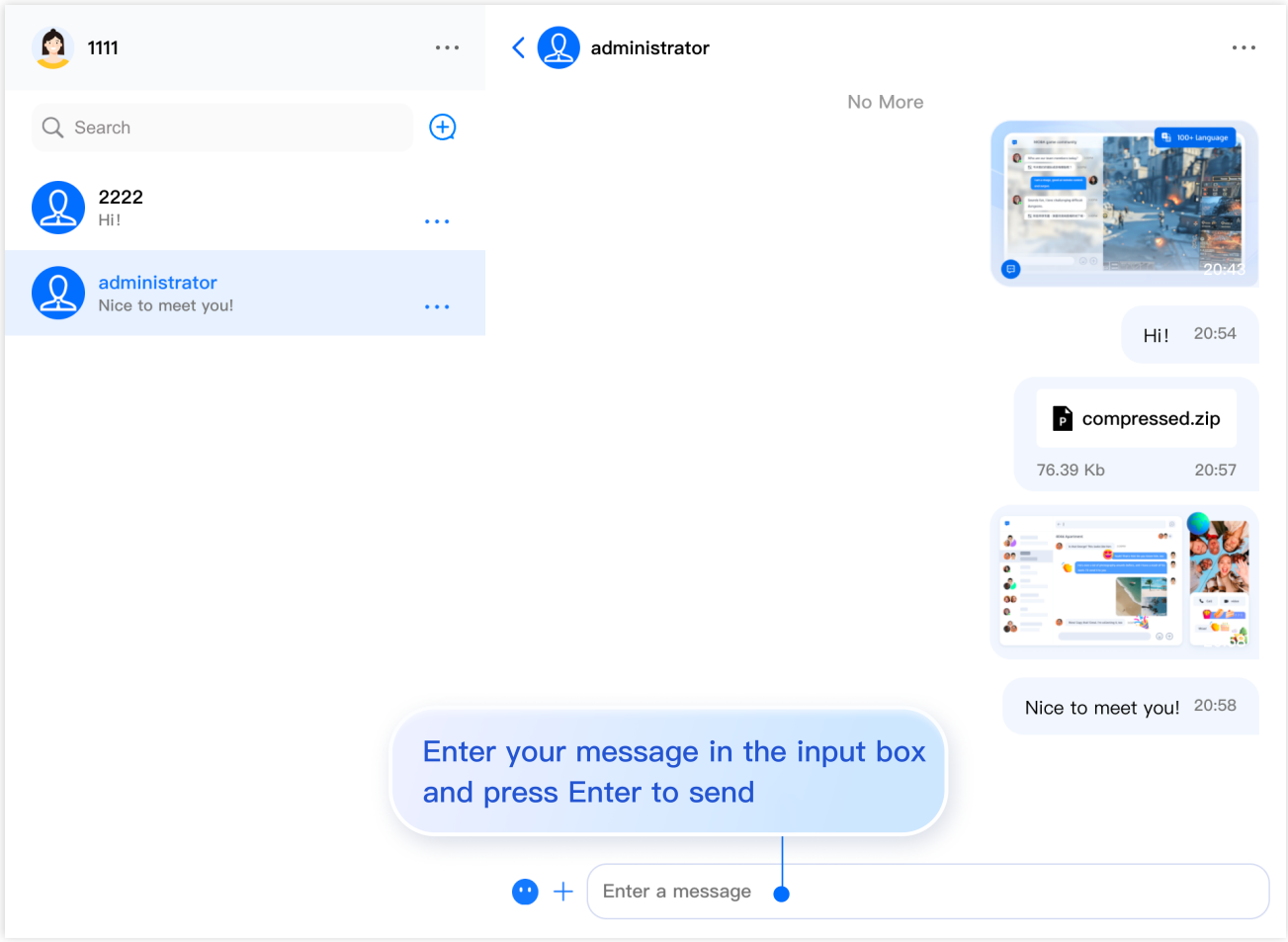
```
npm run start
```

注意：

1. 请确保 步骤3 代码中 `SDKAppID` 、 `userID` 和 `userSig` 均已成功替换，如未替换将会导致项目表现异常。
2. `userID` 和 `userSig` 为一一对应关系，具体参见 生成 UserSig。
3. 如遇到项目启动失败，请检查 开发环境要求 是否满足。

步骤7：发送您的第一条消息

在输入框中输入您的消息，然后按 Enter 发送。



常见问题

什么是 UserSig？

UserSig 是用户登录 Chat 的密码，其本质是对 UserID 等信息加密后得到的密文。

如何生成 UserSig？

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

注意：

本文示例代码采用的获取 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**。正确的 UserSig 签发方式请参见上文。

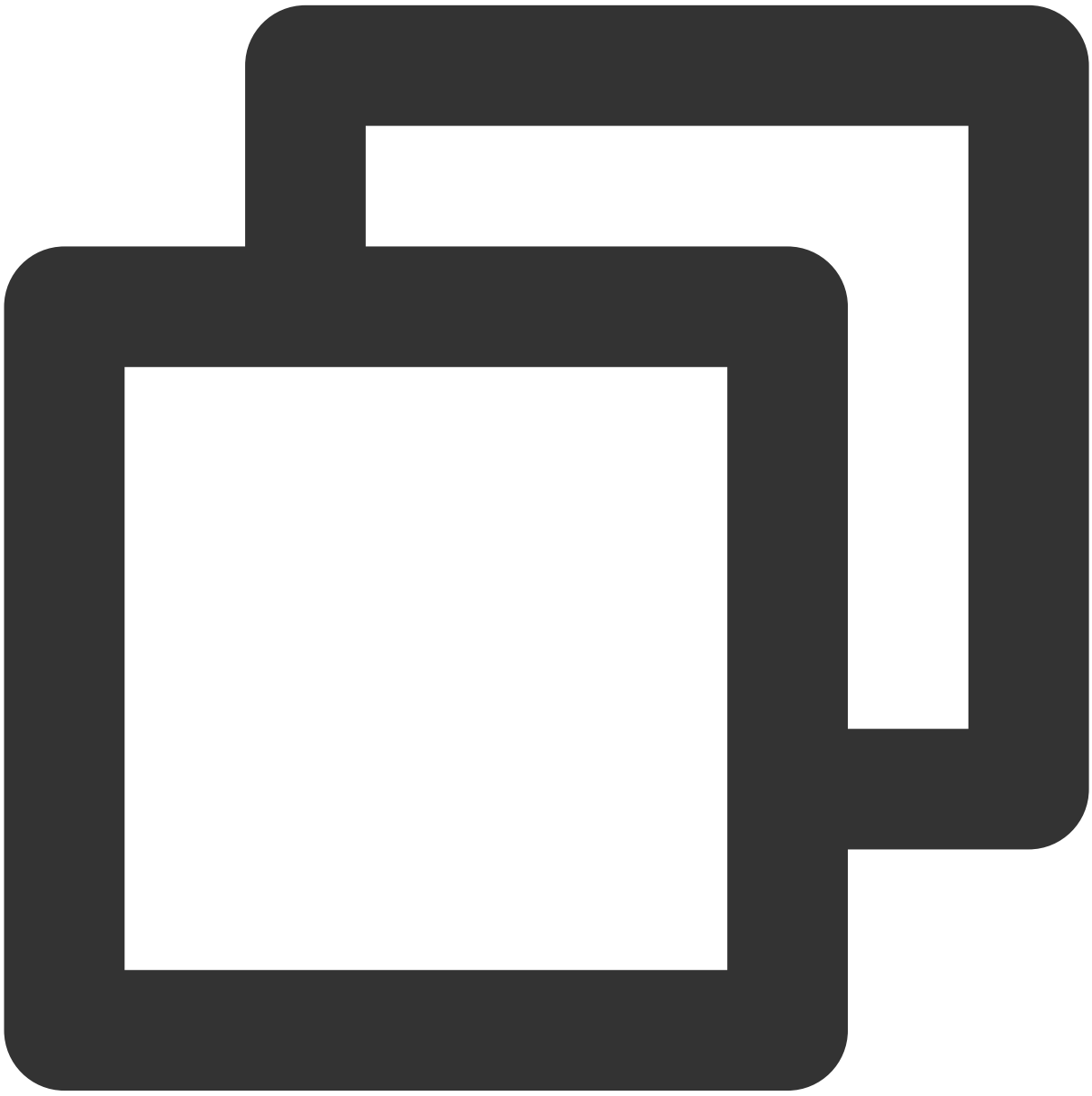
如何集成 UIKit 源码？

我么建议您优先采用 npm 集成方式，如果 npm 集成方式不能满足您更深层次的定制需求，您可以采用源码集成，源码集成方式如下：

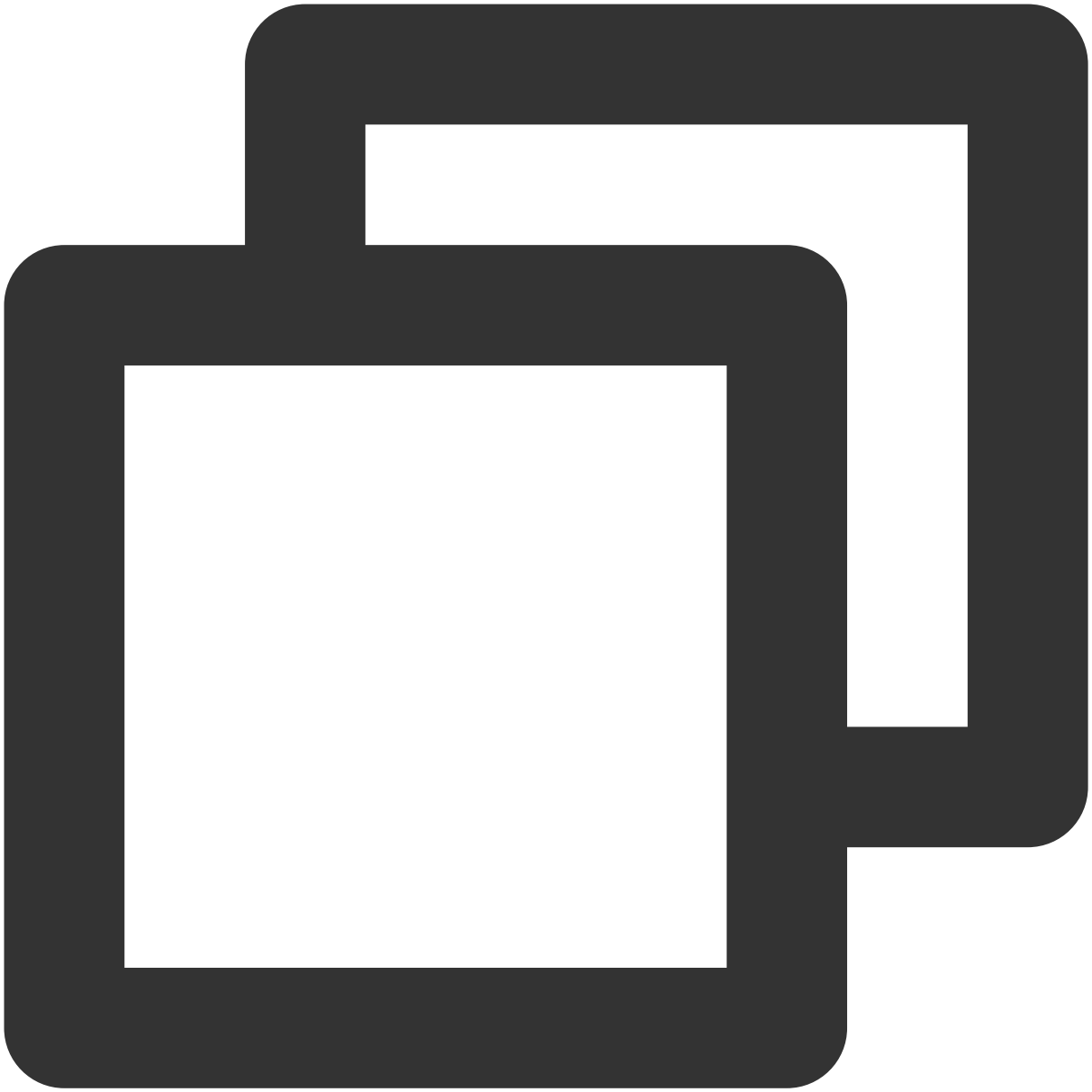
1. 将 TUIKit 拷贝到自己项目的 src 文件目录下：

macOS 端

Windows 端

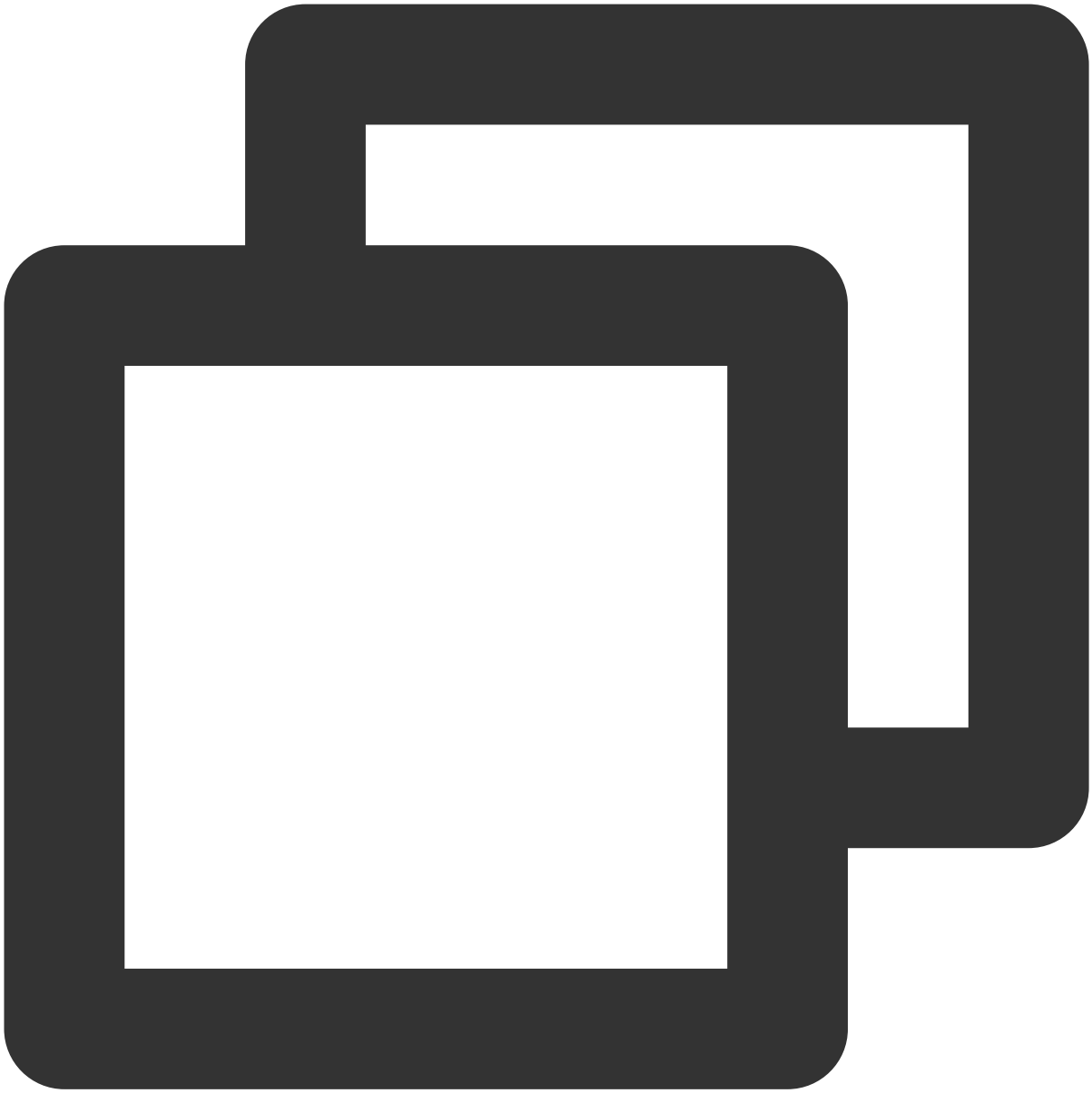


```
mkdir -p ./src/TUIKit && rsync -av  ./node_modules/@tencentcloud/chat-uikit-react/
```



```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-react .\\src\\TUIKit /i /e
```

2. 替换项目中所有 `@tencentcloud/chat-uikit-react` 为 `./TUIKit`



```
// 比如，替换 TUIKit、TUIChat 等组件引入源为 './TUIKit'
// before
import { TUIChat, TUIKit } from 'tencentcloud/chat-uikit-react';
// after
import { TUIChat, TUIKit } from './TUIKit';
```

交流与反馈

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

相关文档

UIKit 相关：

chat-uikit-react npm
Demo源码及跑通示例

实现更多功能，请参考 ChatEngine API 文档：

chat-uikit-engine API 手册
chat-uikit-engine npm

Web & H5 （Vue）

最近更新时间：2024-08-07 15:06:09

TUIKit 主要功能介绍

TUIKit 主要分为 TUIChat、TUIConversation、TUIGroup、TUIContact、TUISearch 几个 UI 子组件，每个 UI 组件负责展示不同的内容。

TUIKit for H5
2.聊天 (TUIChat)
腾讯云Web IM – Demo
发起聊天
操作指引
示例客服群
感觉灯光还不错
示例好友
[134条] 晚上去鬼屋探险啊?
Pinko.zhang
下班准备去哪里吃饭，约起?
潘艺竹
[草稿] 在项目伊始时做一个用户研究的作...
庄思纨
在项目伊始时做一个用户研究的作...
蔚琬
[语音]
消息
通讯录
我的
Linda
你好呀
你好哦，腾讯云即时通讯IM的开发者，终于等到你了！你可以在聊天框进行消息发送测试哦~
4次回复
李洋
邮件已经发出去，请注意查收
好的
等待接听
请输入消息
发送
添加好友/群聊
新的联系人
群聊通知
我的群聊
黑名单
联系人
Candy
皓辰
蒋茵
你好
全部
聊天记录
选择时间：全部
1.会话 (TUIConversation)
3.音视频通话(TUICallKit)
4.关系链(TUIContact)

TUIKit 消息云端搜索
全局搜索(TUISearch)
你好
示例群聊
全部
文本
文件
其他
选择时间：全部
今天
近三天
近7天
文本
银河造梦机
3009条相关文本
15:26
肥水不牛油
10条相关文本
15:26
小熊出击
3009条相关文本
15:26
月亮不会告白
3009条相关文本
2023/7/21
宇宙航行日记
3009条相关文本
2023/1/11
前任三秒
3009条相关文本
2022/12/12
10条与“你好”相关的文本
进入聊天 >
提莫
15:26
你好哇
肥水不牛油
15:26
你好你好
定位到聊天位置
肥水不牛油
2023/07/26
你好搞笑
肥水不牛油
2023/07/25
你好到哪了?
请输入消息
发送
搜索
全部 99
未读 12
示例客服
10:25
明天11:30的部门会议，你来...
示例群聊
10:32
[草稿] 在项目伊始时做一个用户...
Linda
11:03
好的，多谢
Pika
昨天
图片已发送，注意查收！
小林
昨天
麻烦告知一下，在线等您回复啊

开发环境要求

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

TUIKit 源码集成（ Web & H5 ）

步骤1：创建项目

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

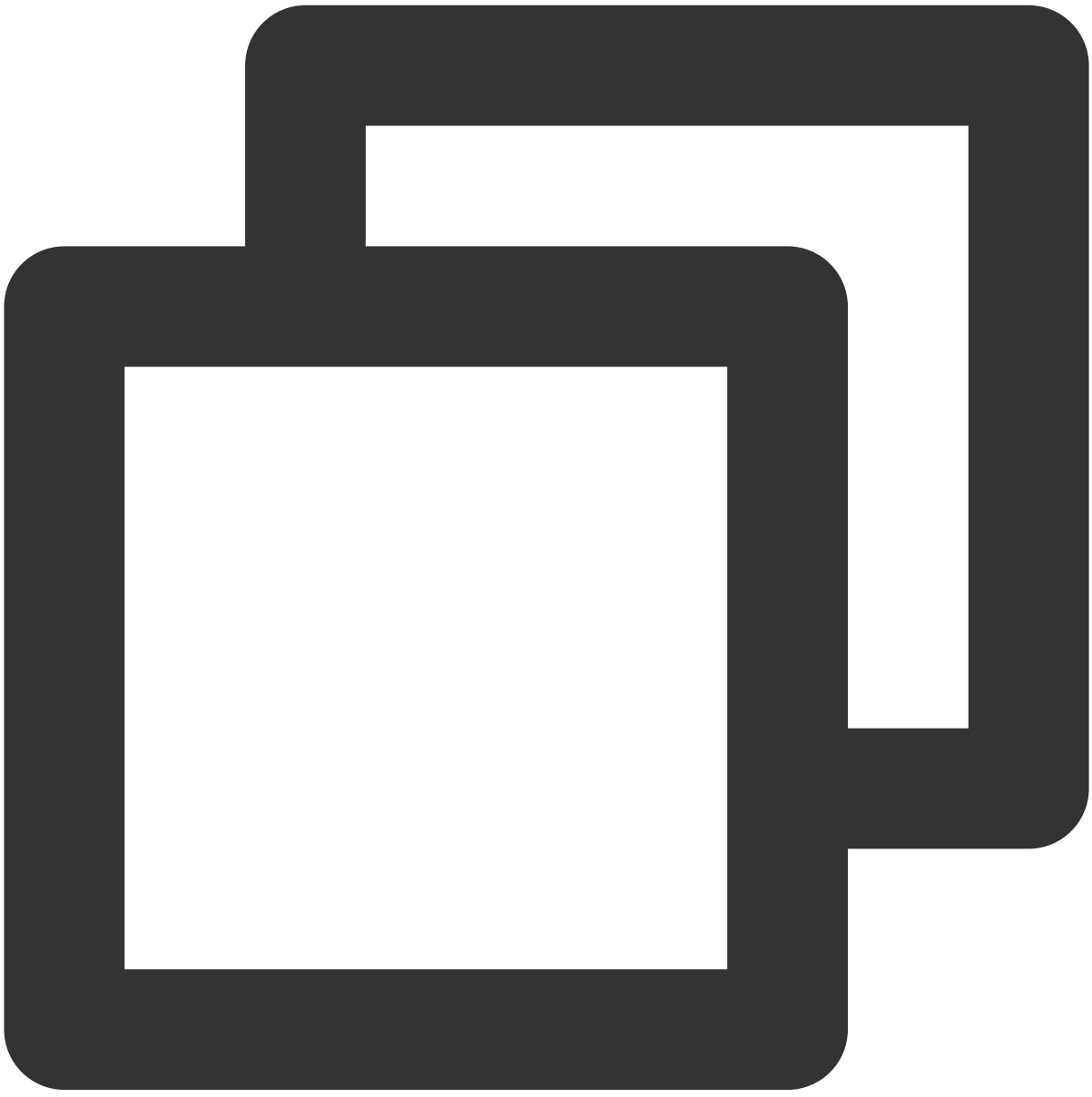
vite

注意：

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

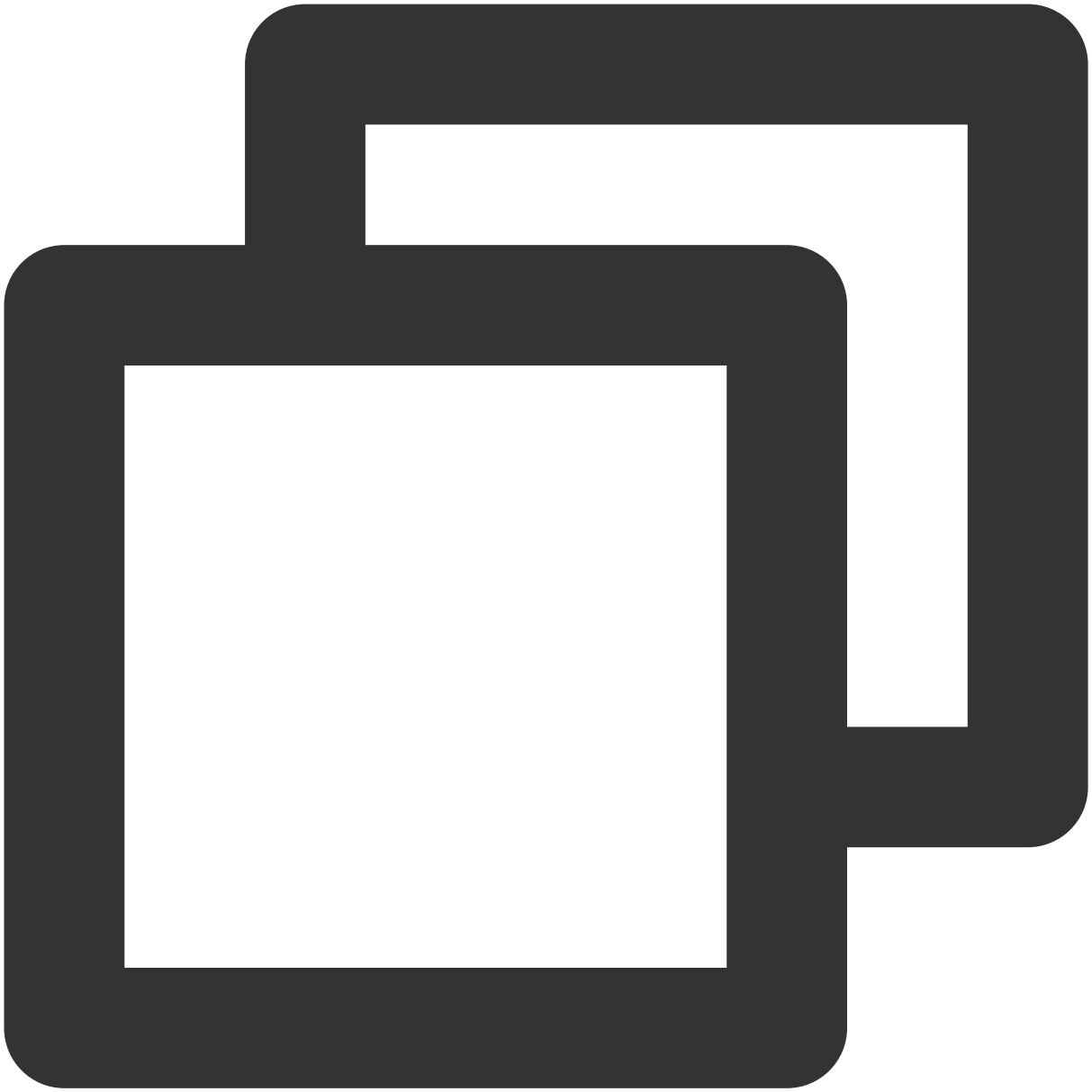
使用 vue-cli 方式创建项目， 配置 Vue2 / Vue3 + TypeScript + sass。

如果您尚未安装 vue-cli 或者 vue-cli 版本低于 5.0.0 ，可以在 terminal 或 cmd 中采用如下方式进行安装：



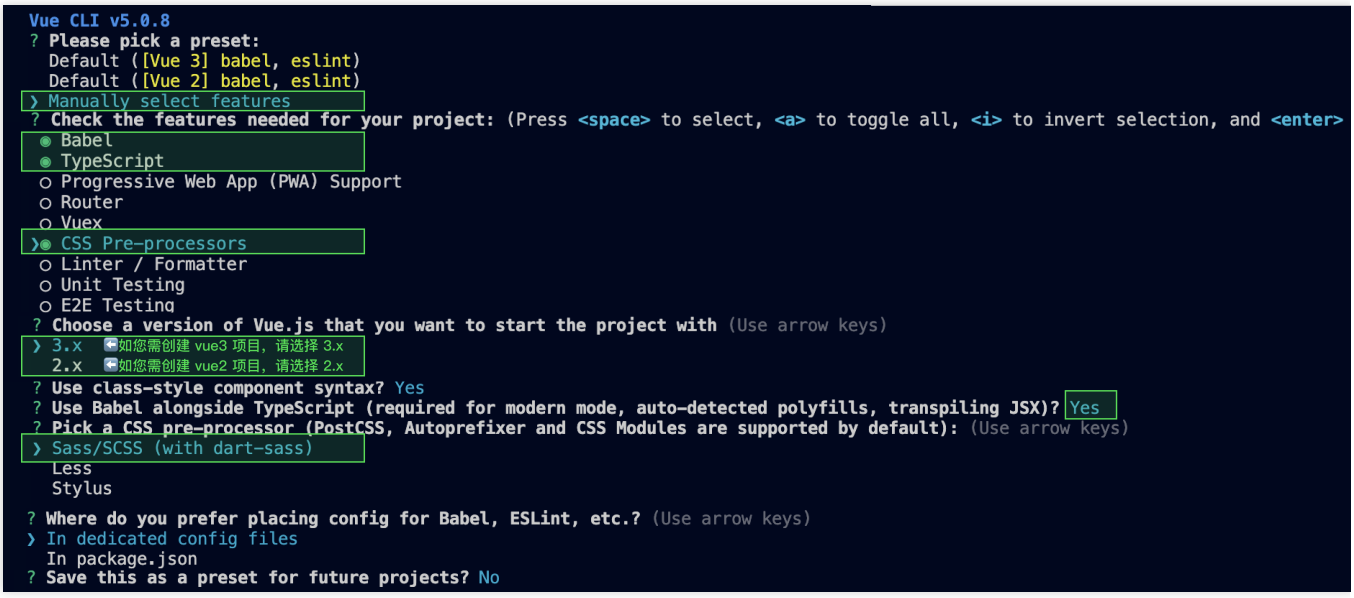
```
npm install -g @vue/cli@5.0.8 sass sass-loader@10.1.1
```

通过 vue-cli 创建项目，并选择下图中所选配置项。

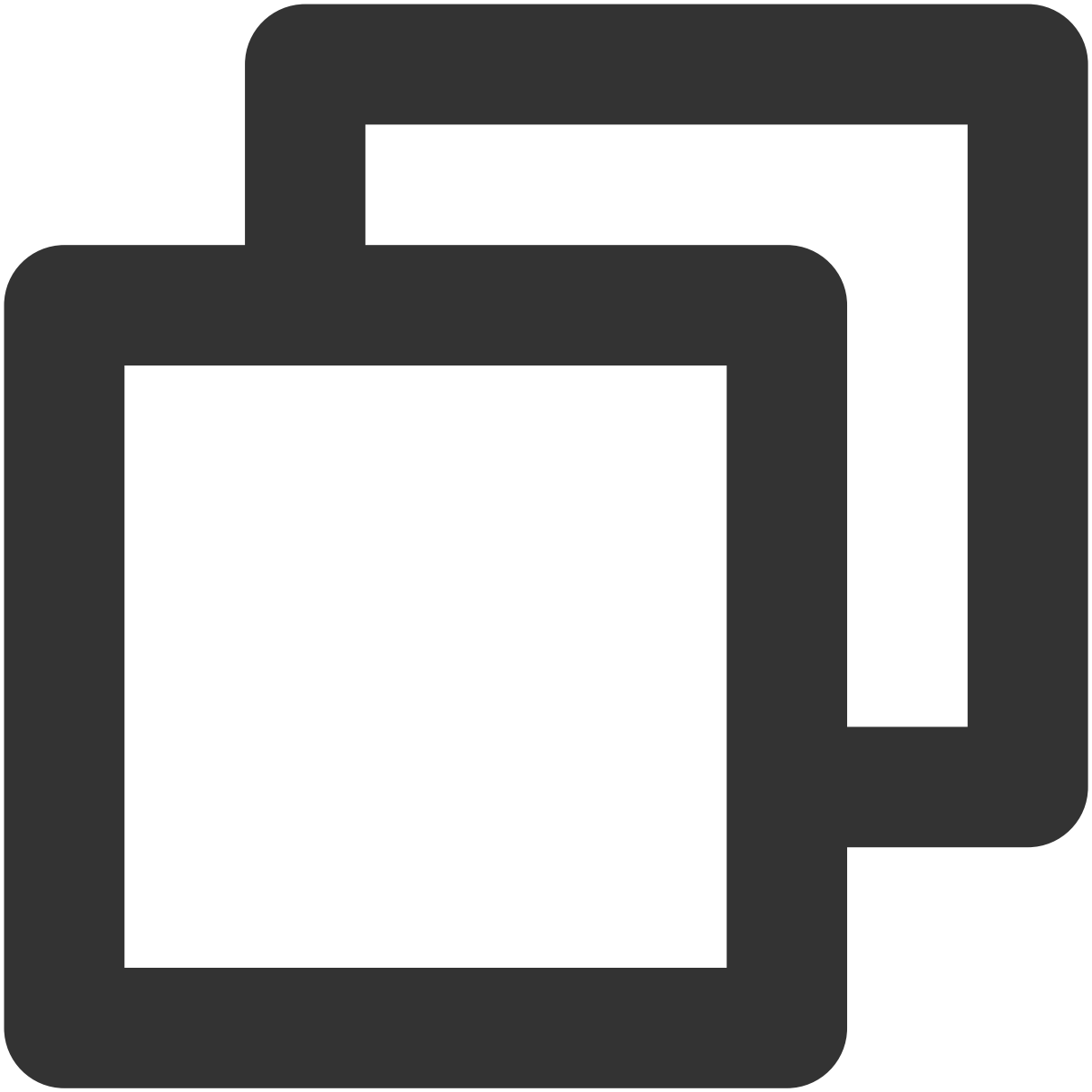


```
vue create chat-example
```

请务必保证按照如下配置选择：



创建完成后，切换到项目所在目录：

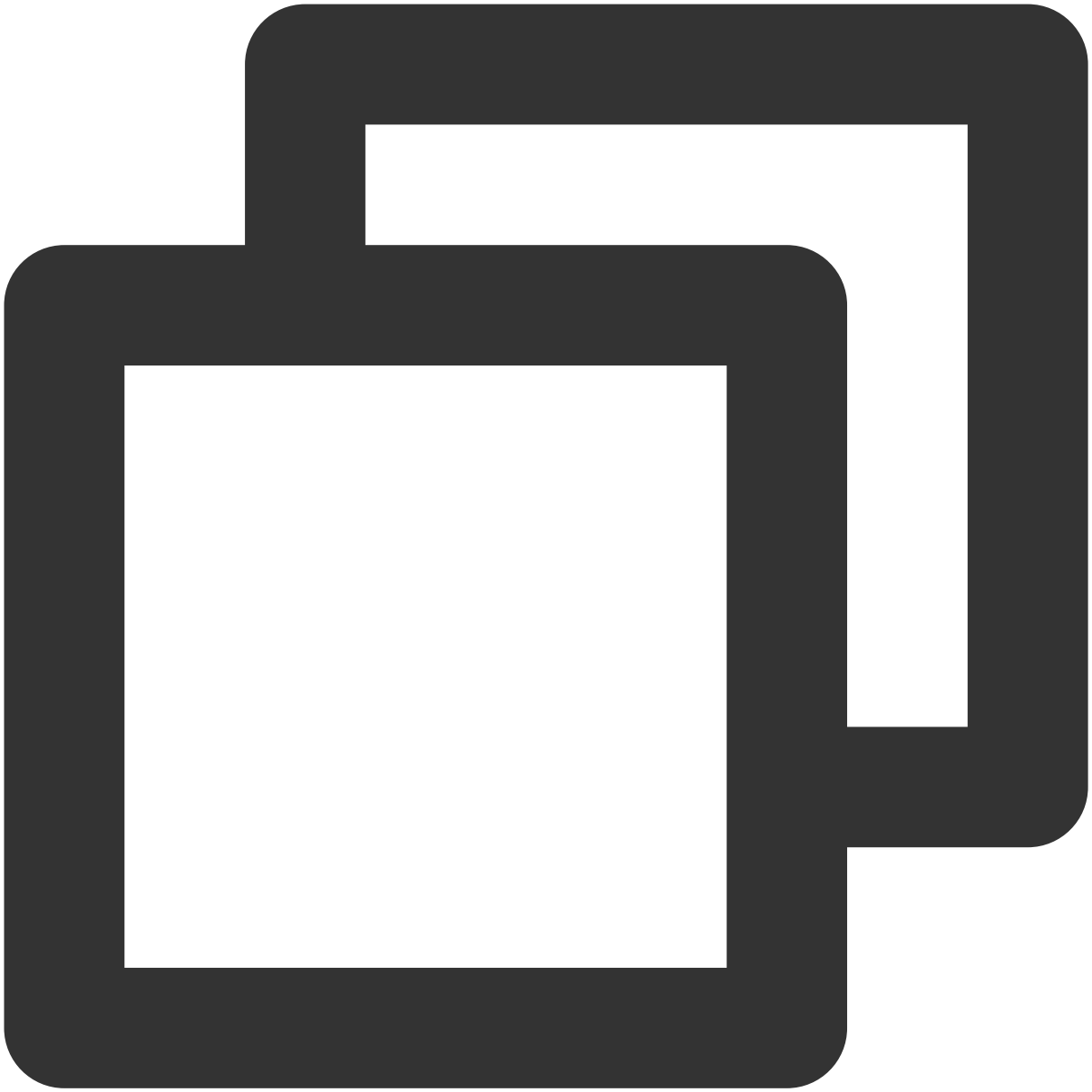


```
cd chat-example
```

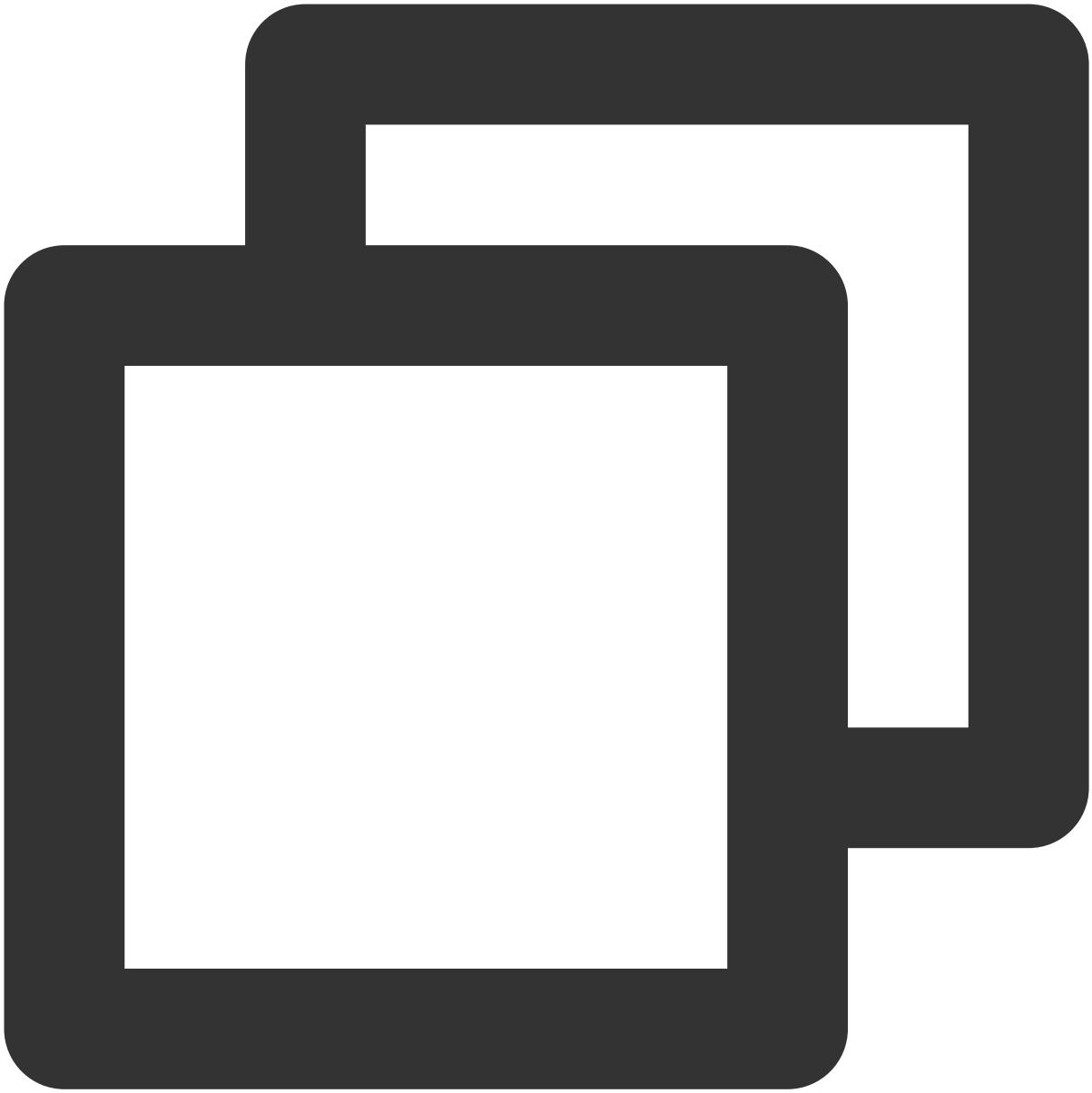
如果您是 vue2 项目，请根据您所使用的 Vue 版本进行以下相应的环境配置， vue3 项目请忽略。

vue2.7

vue2.6及以下



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

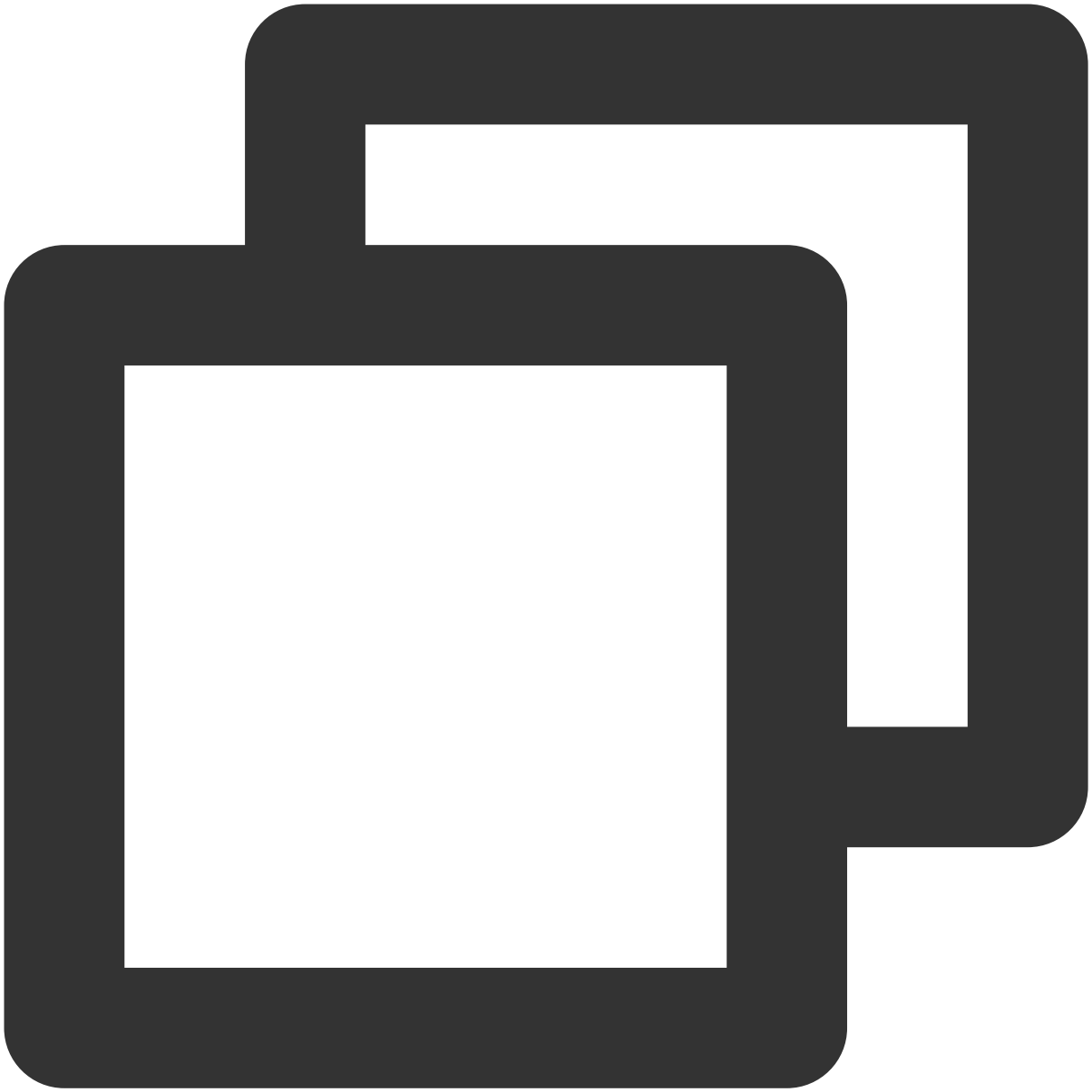


```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

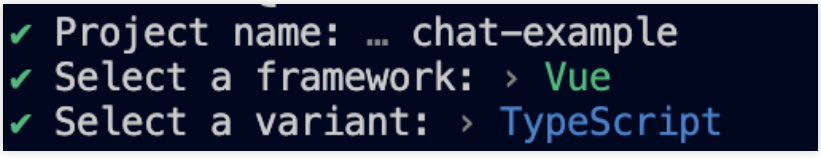
注意：

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

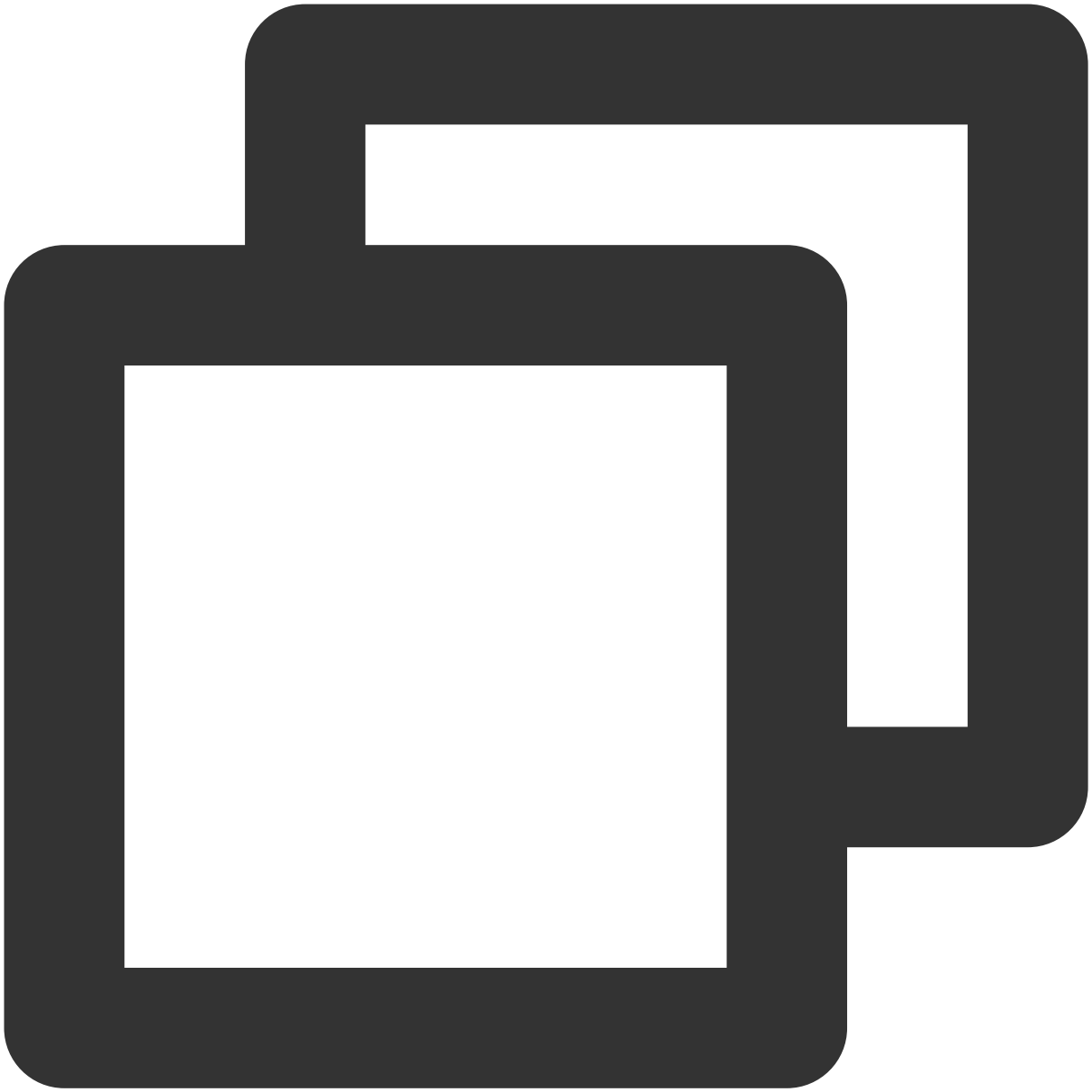
使用 vite 方式创建项目，按照下图选项配置 Vue + TypeScript 。



```
npm create vite@latest
```

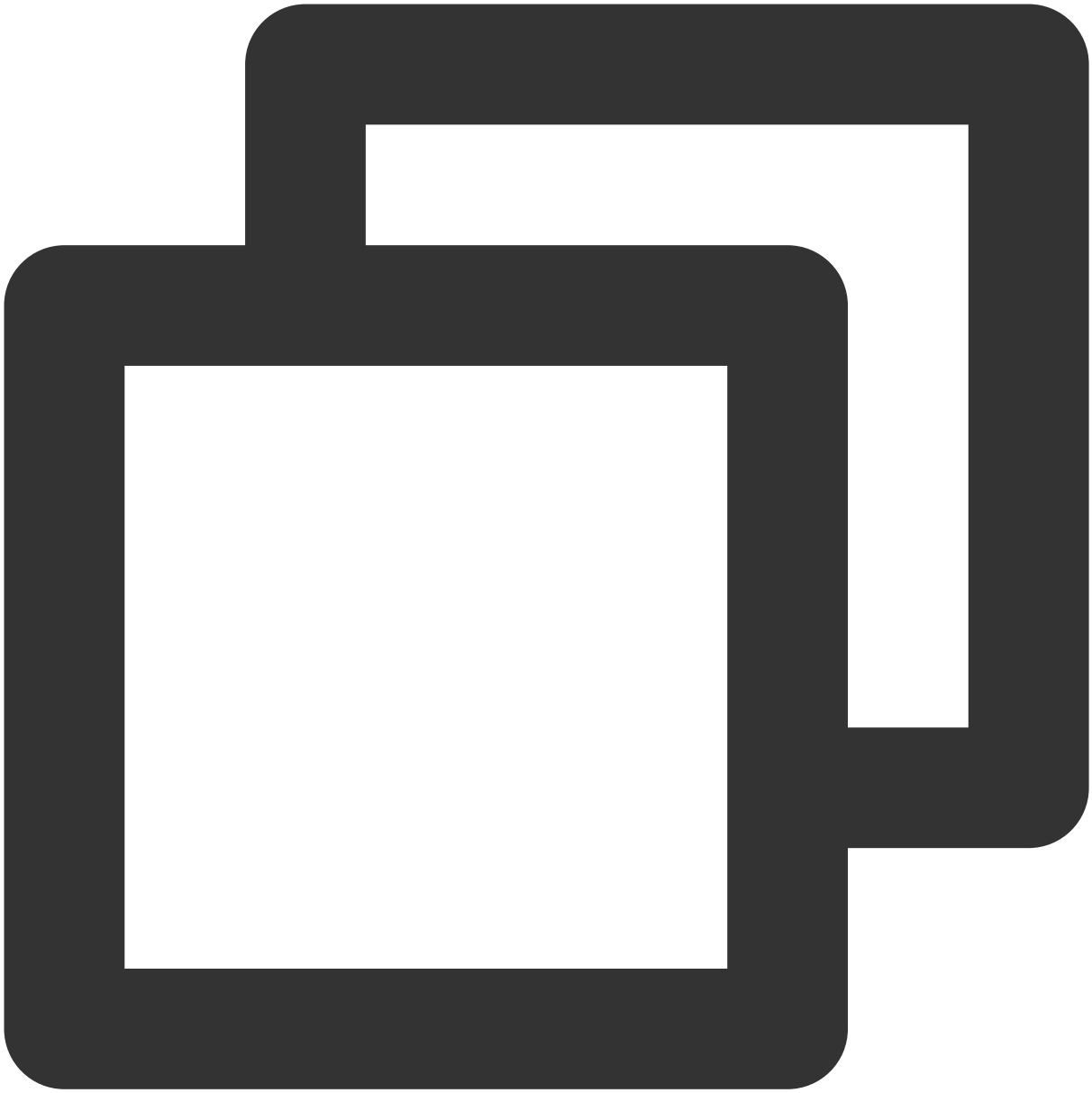


之后切换到项目目录，安装项目依赖：



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

安装 TUIKit 所需 sass 环境依赖：



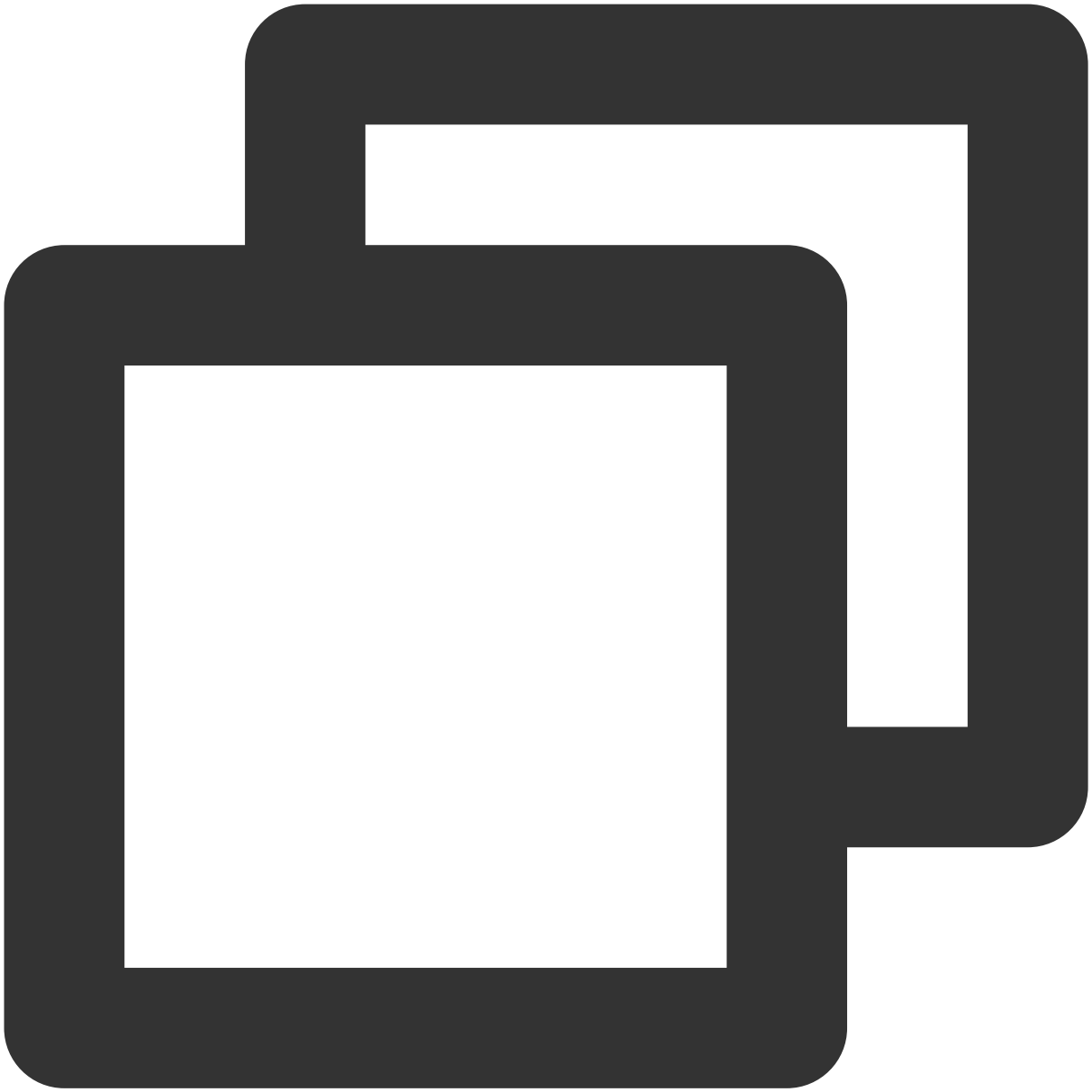
```
npm i -D sass sass-loader
```

步骤2：下载 TUIKit 组件

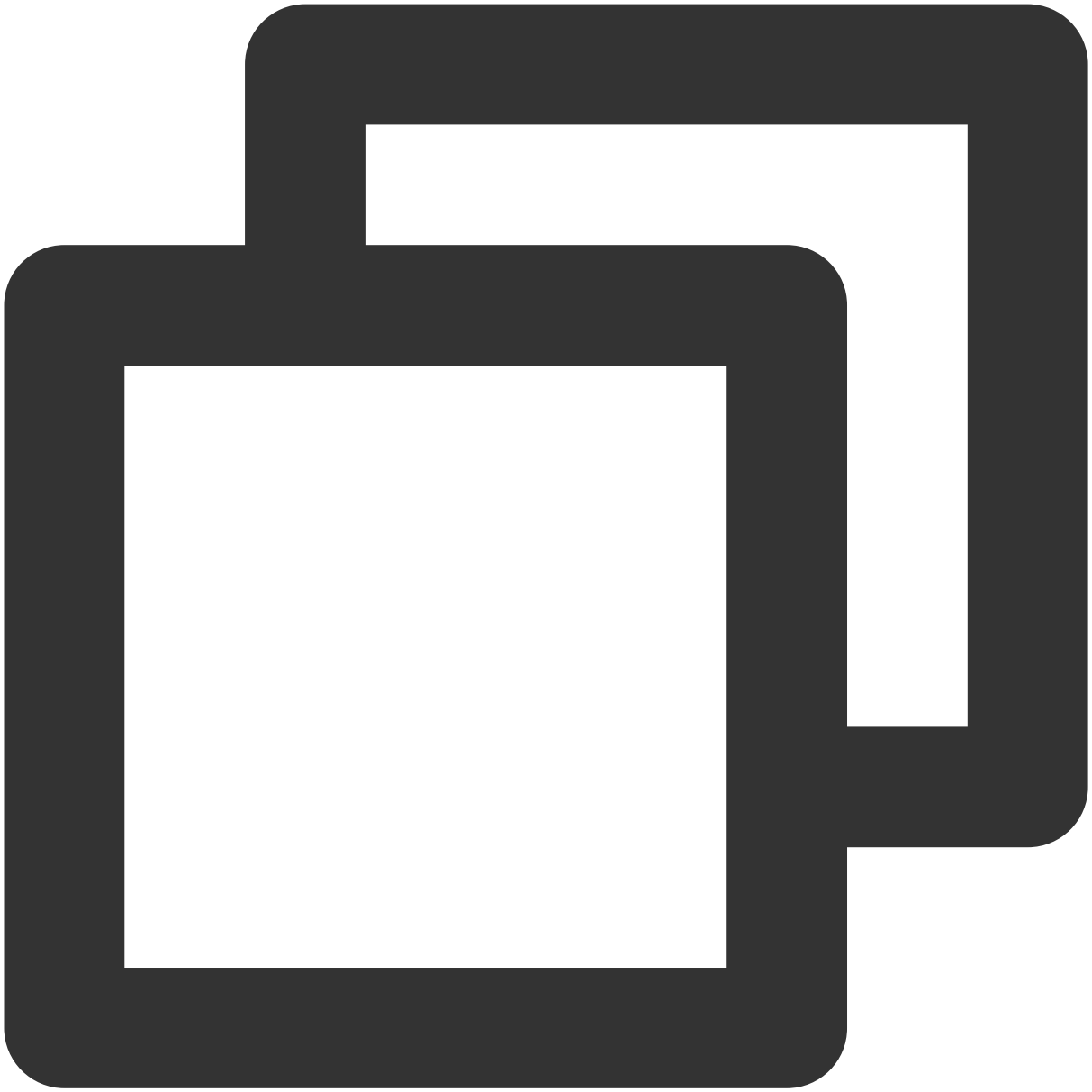
通过 npm 方式下载 TUIKit 组件，为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 src 目录下：

macOS 端

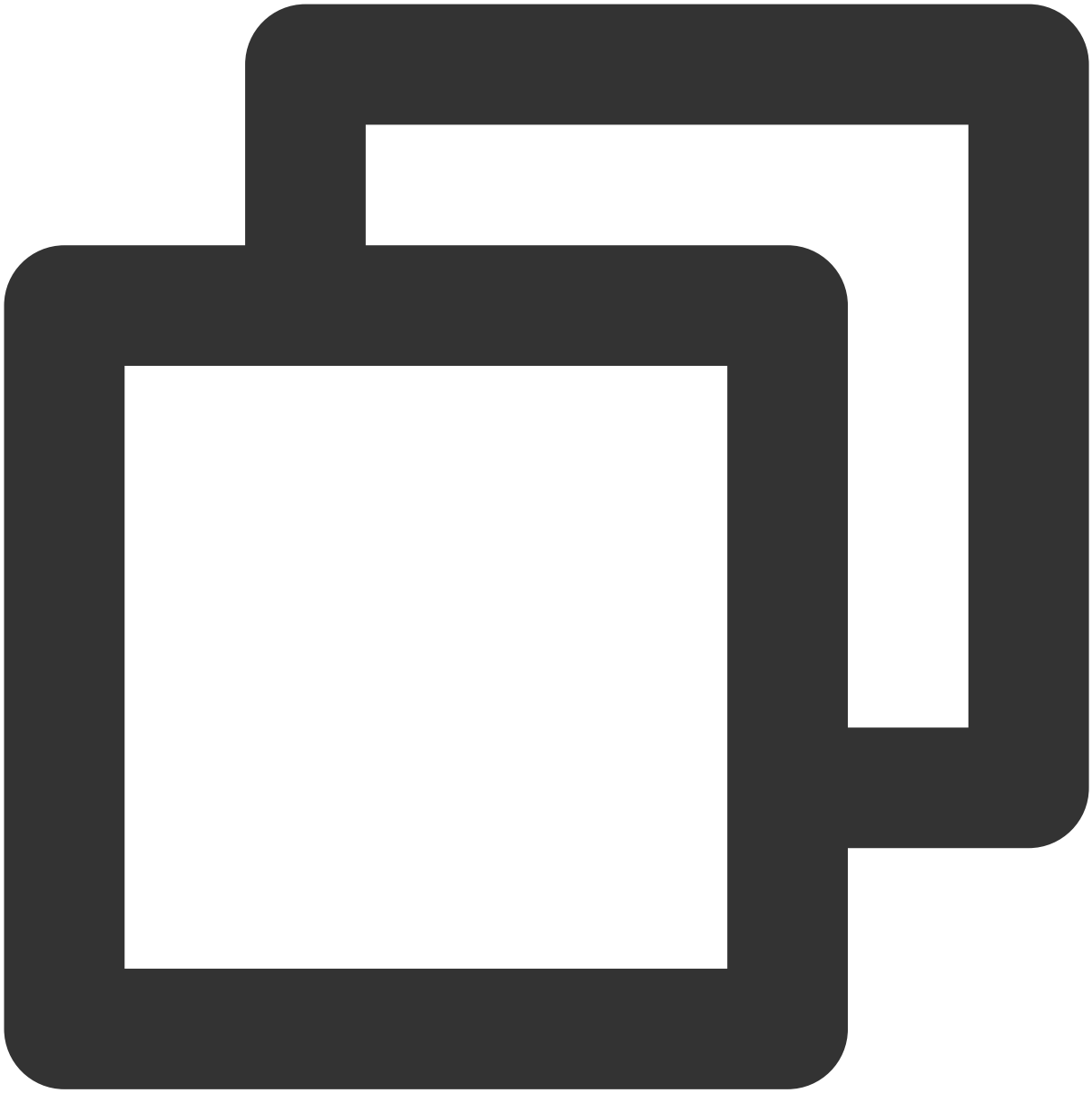
Windows 端



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



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



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

步骤3：引入 TUIKit 组件

在需要展示的页面，引入 TUIKit 的组件即可使用。

例如：在 App.vue 页面中实现以下代码，即可快速搭建聊天界面（以下示例代码同时支持 Web 端与 H5 端）：

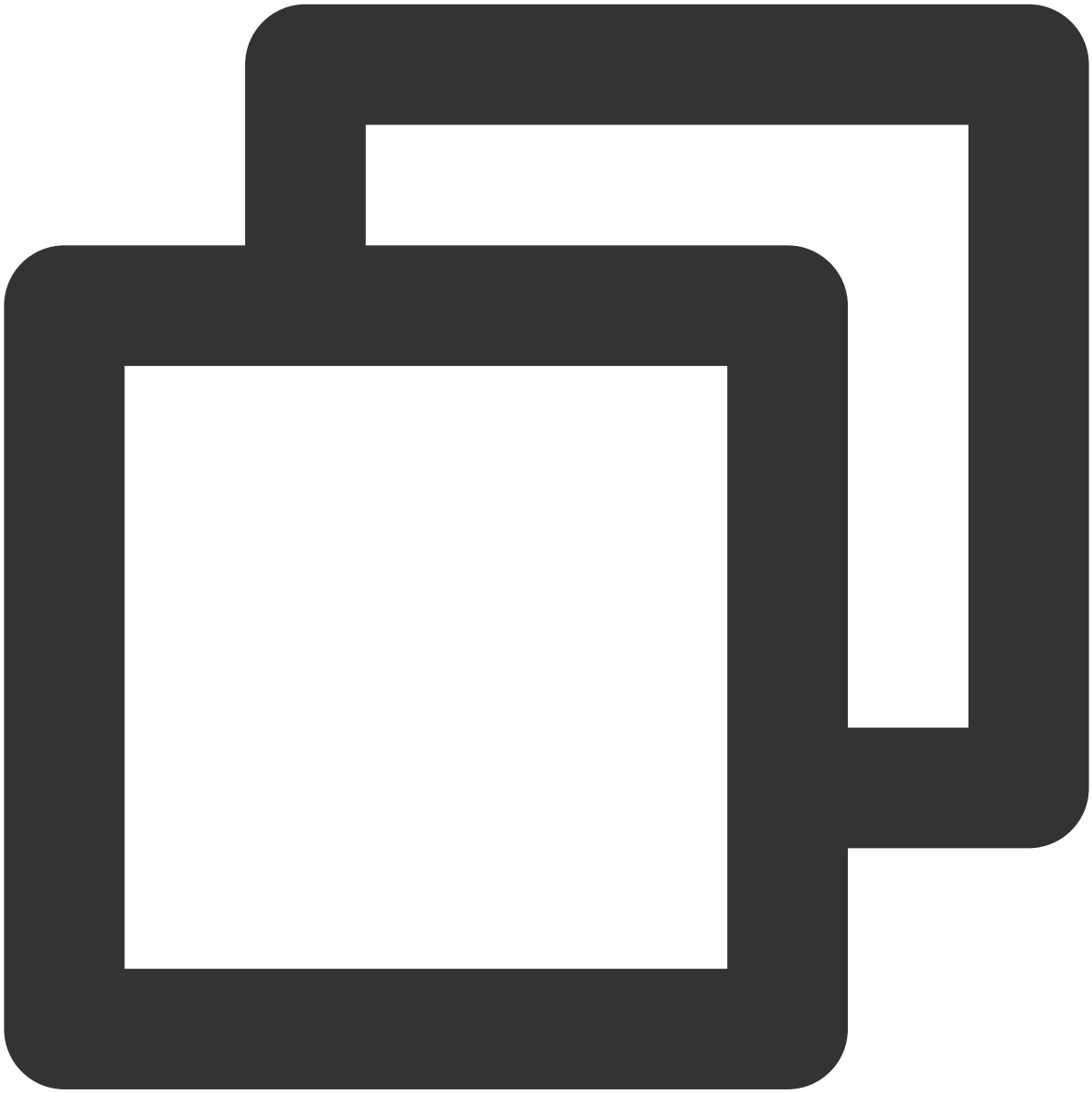
注意：

以下示例代码使用了 setup 语法，如果您的项目没有使用 setup 语法，请按照 Vue3/Vue2 的标准方式注册组件。

vue3

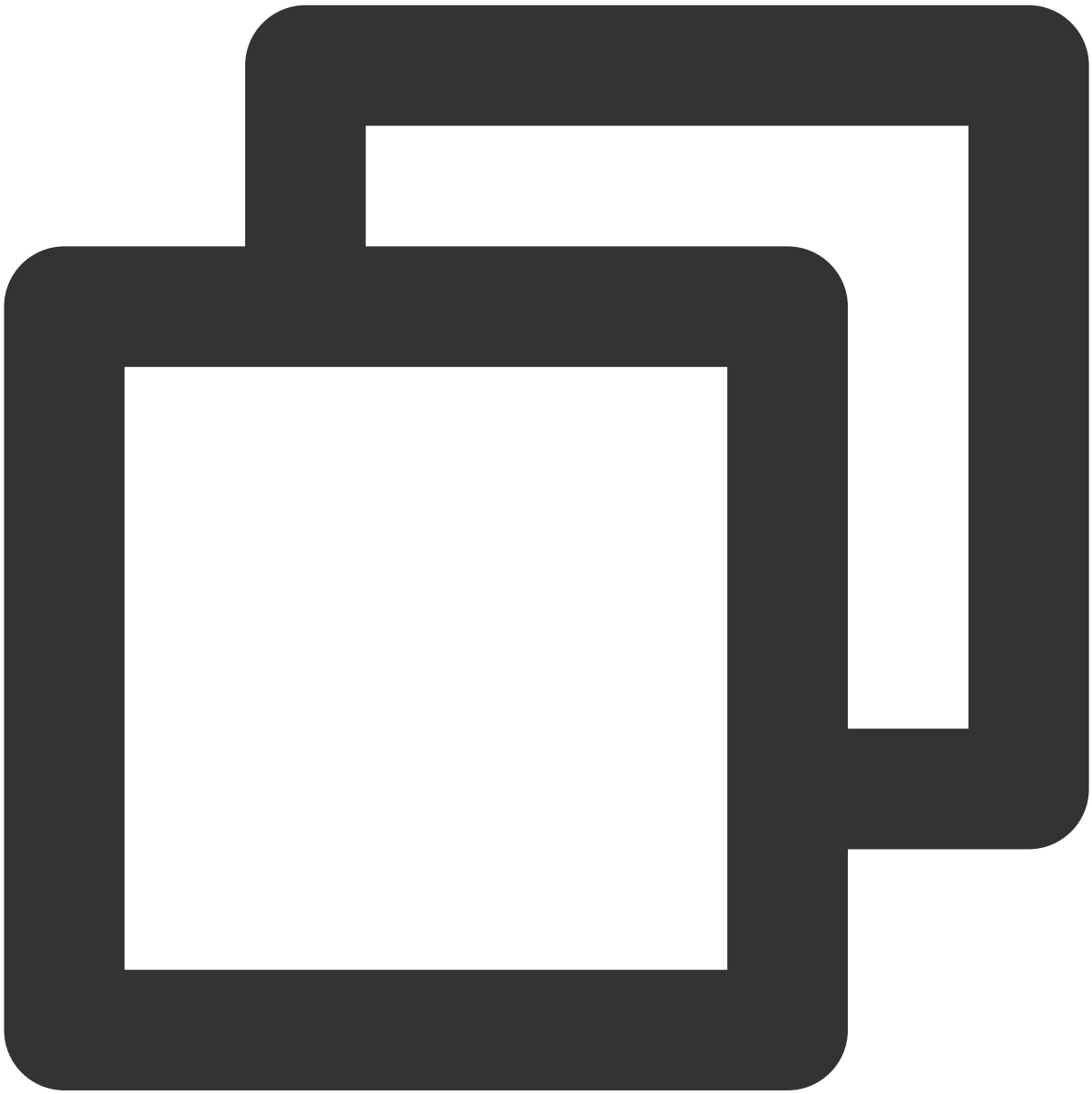
vue2.7

vue2.6及以下



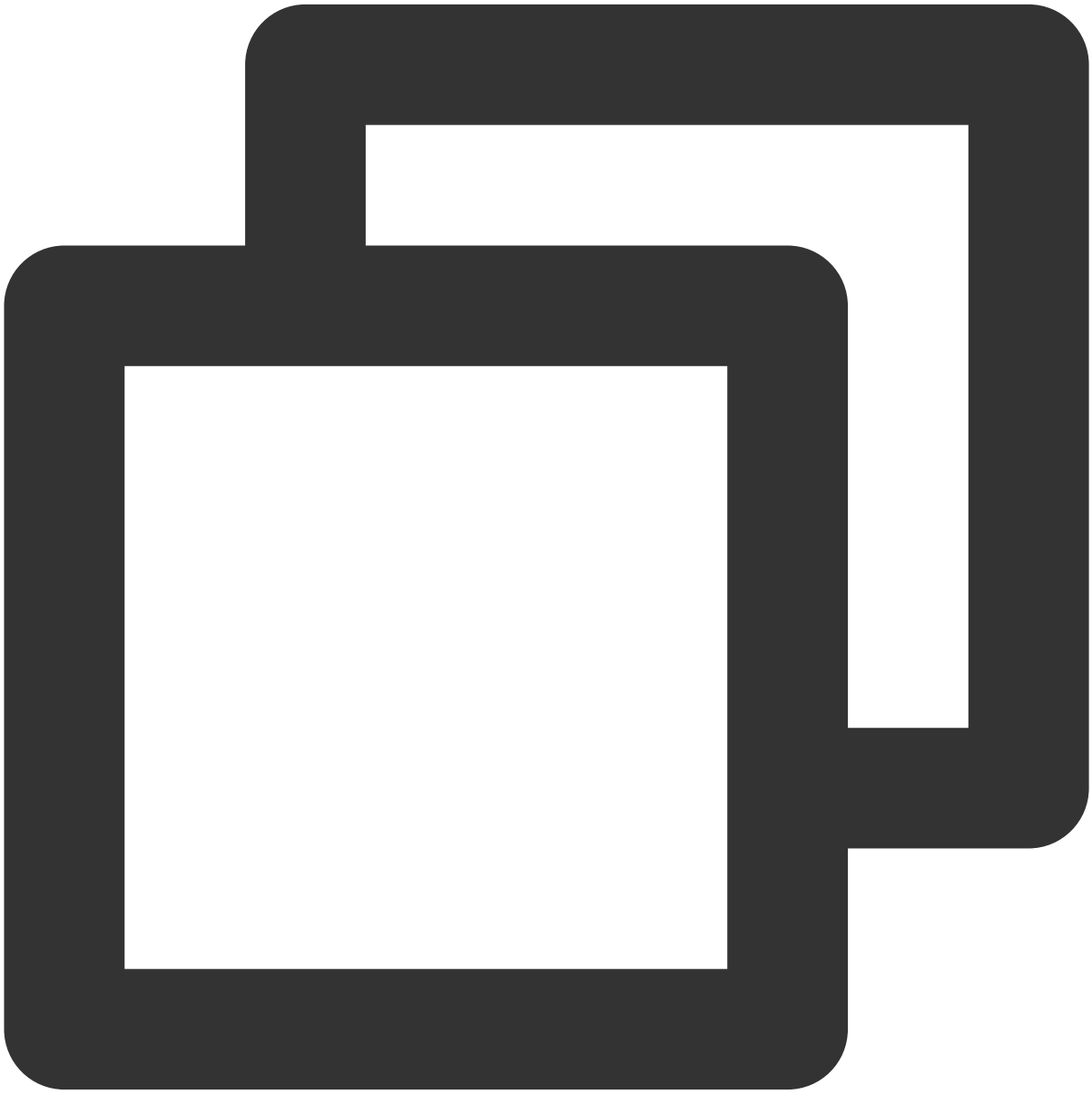
```
<template>
  <div id="app">
    <TUIKit :SDKAppID="YOUR_SDKAPPID" userID="YOUR_USERID" userSig="YOUR_USERSIG" /
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
```

```
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue';
</script>
<style lang="scss">
</style>
```



```
<template>
  <div id="app">
    <TUIKit :SDKAppID="YOUR_SDKAPPID" userID="YOUR_USERID" userSig="YOUR_USERSIG" /
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
```

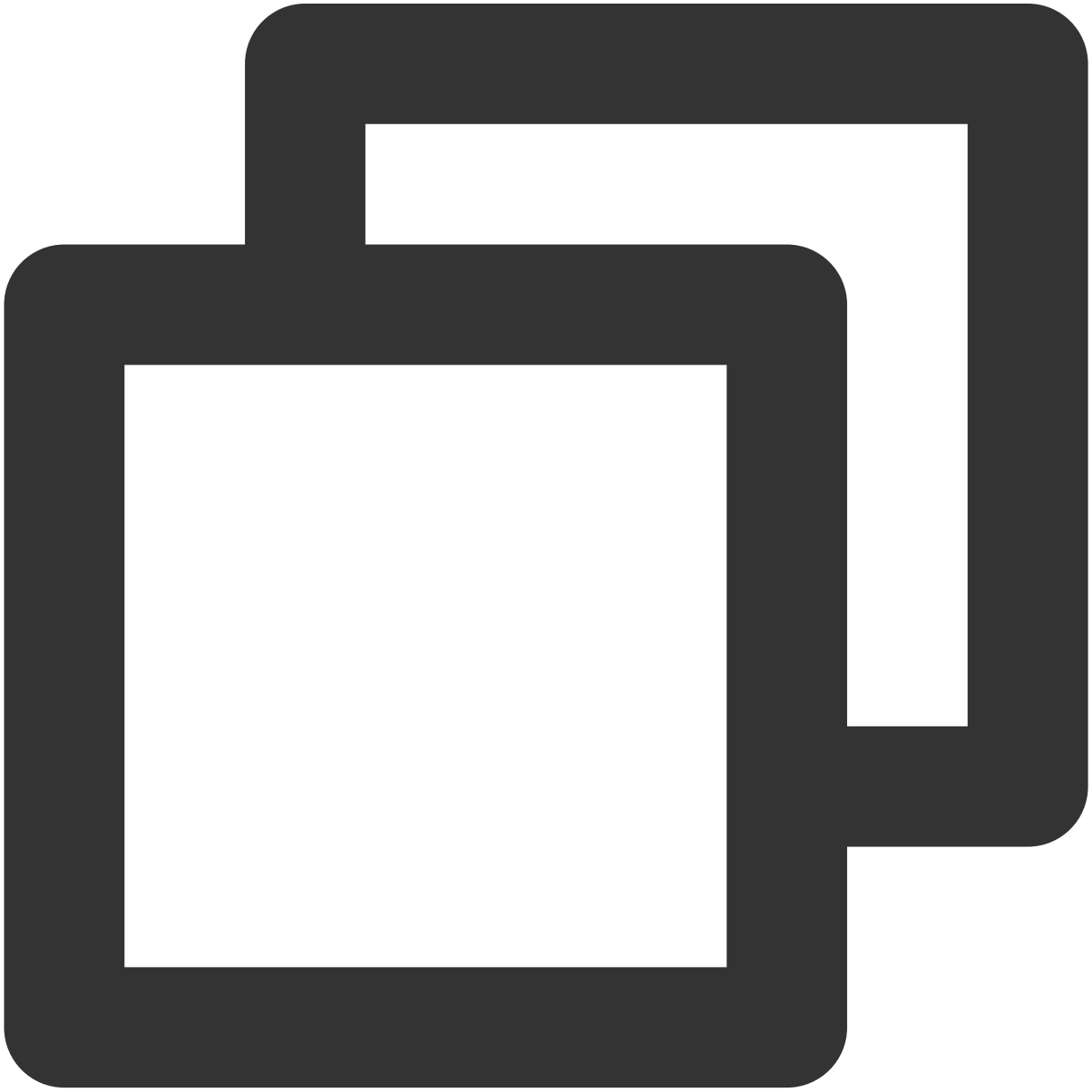
```
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue2';
</script>
<style lang="scss">
</style>
```



```
<template>
  <div id="app">
    <TUIKit :SDKAppID="YOUR_SDKAPPID" userID="YOUR_USERID" userSig="YOUR_USERSIG" /
```

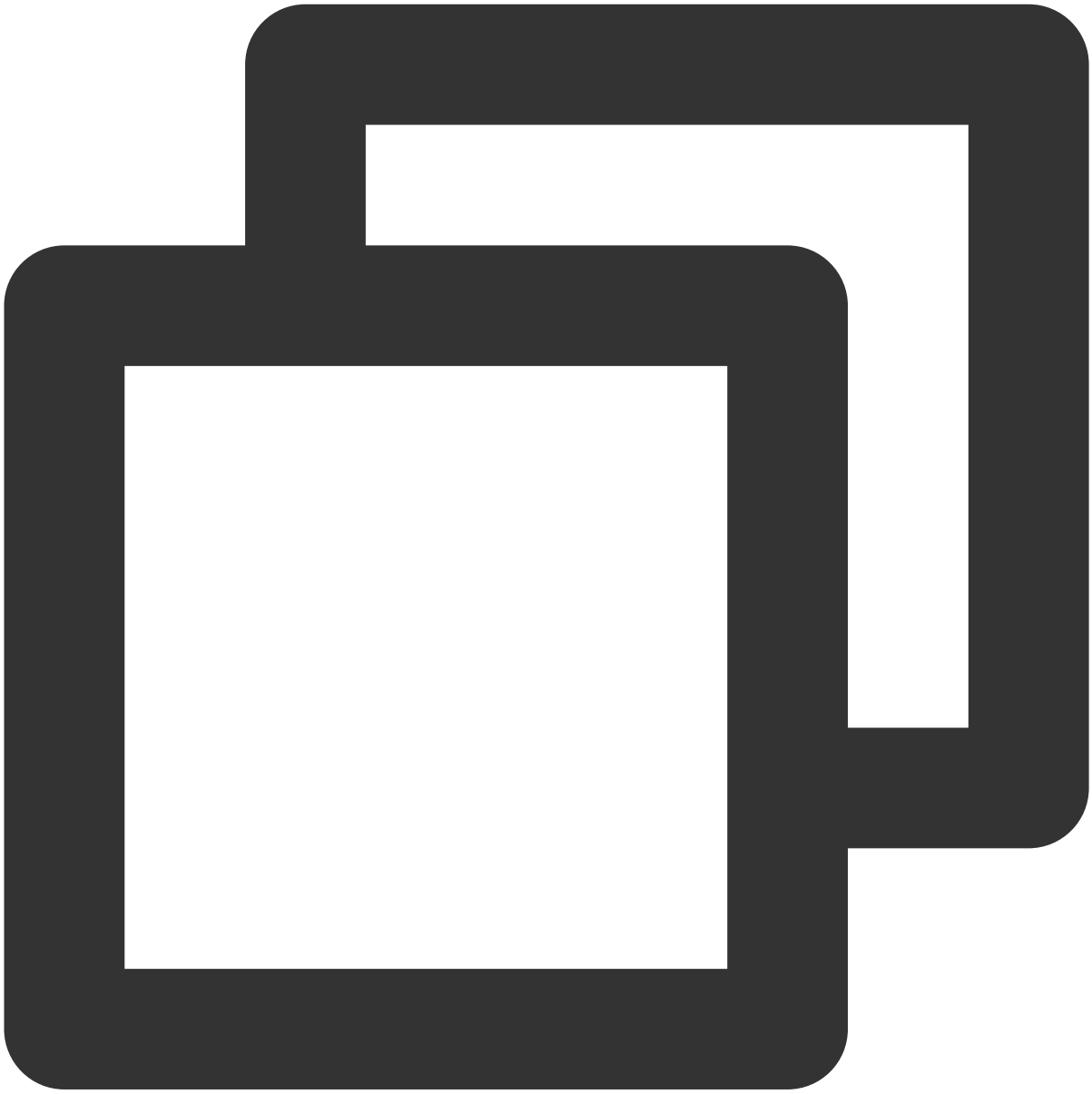
```
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue2.6';
</script>
<style lang="scss">
</style>
```

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



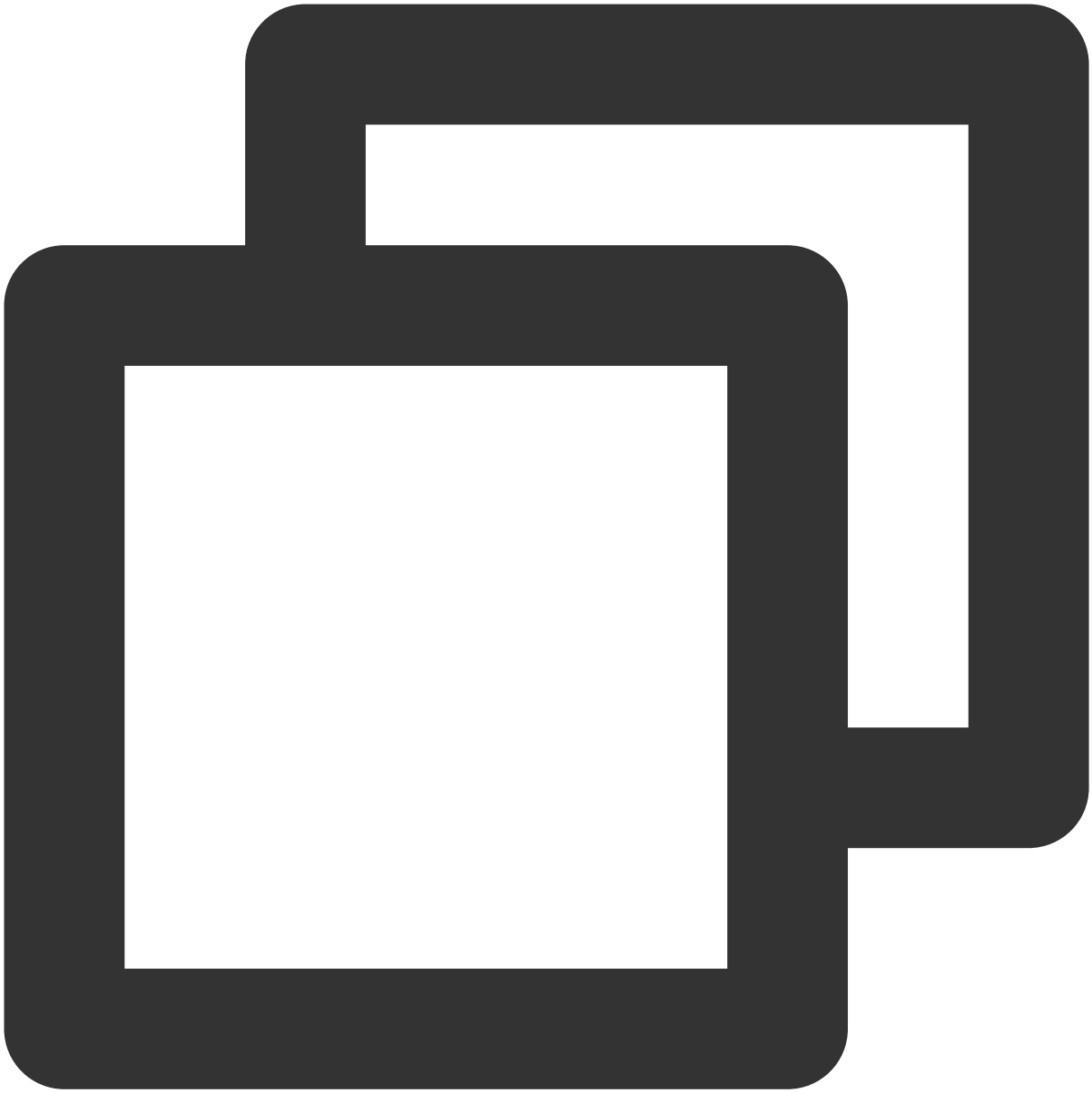
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

2. 在 `main.ts/mian.js` 中引入 VueCompositionAPI。



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

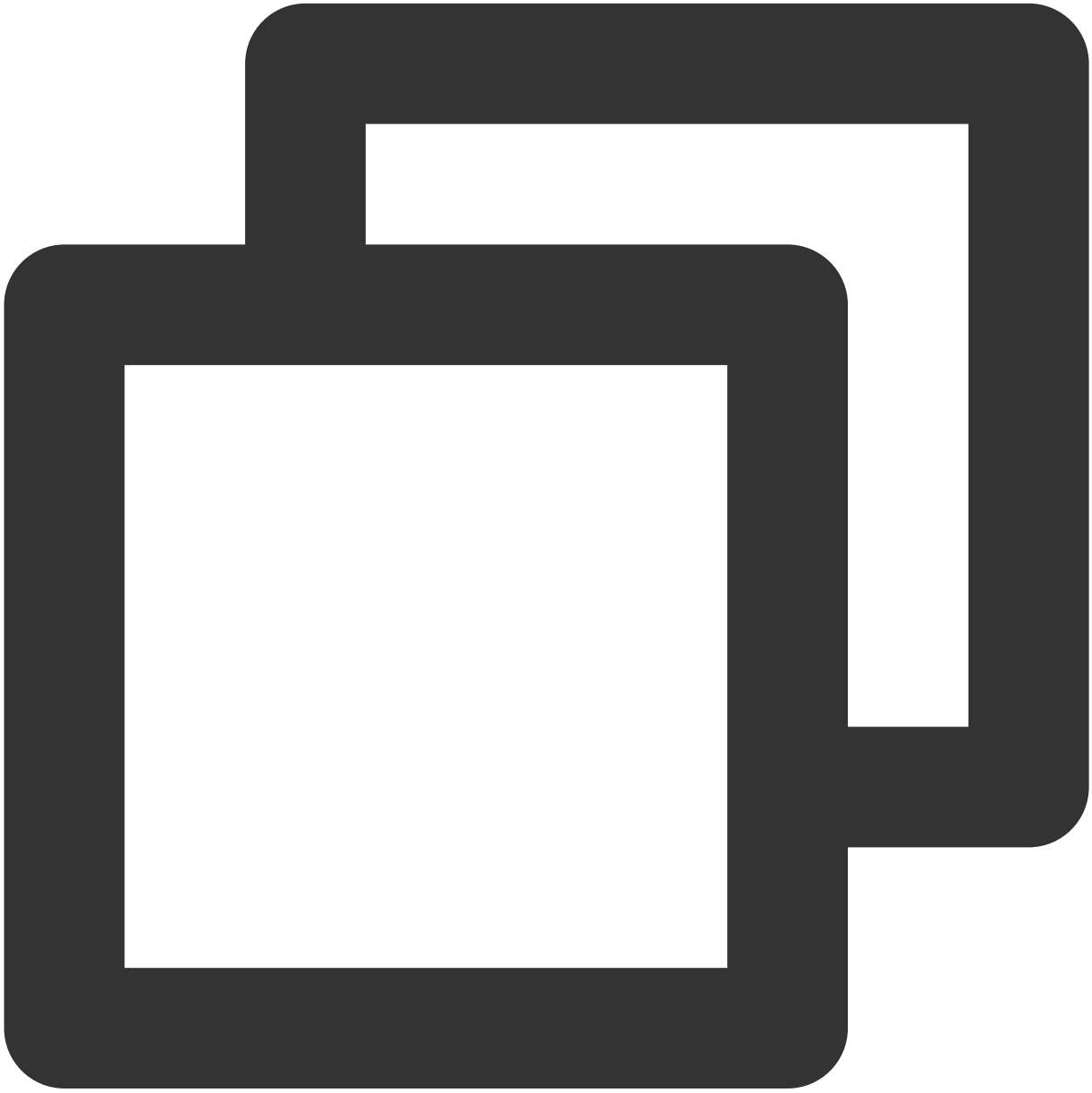
3. 在 `vue.config.js` 中增加，若没有该文件请新建。



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

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

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

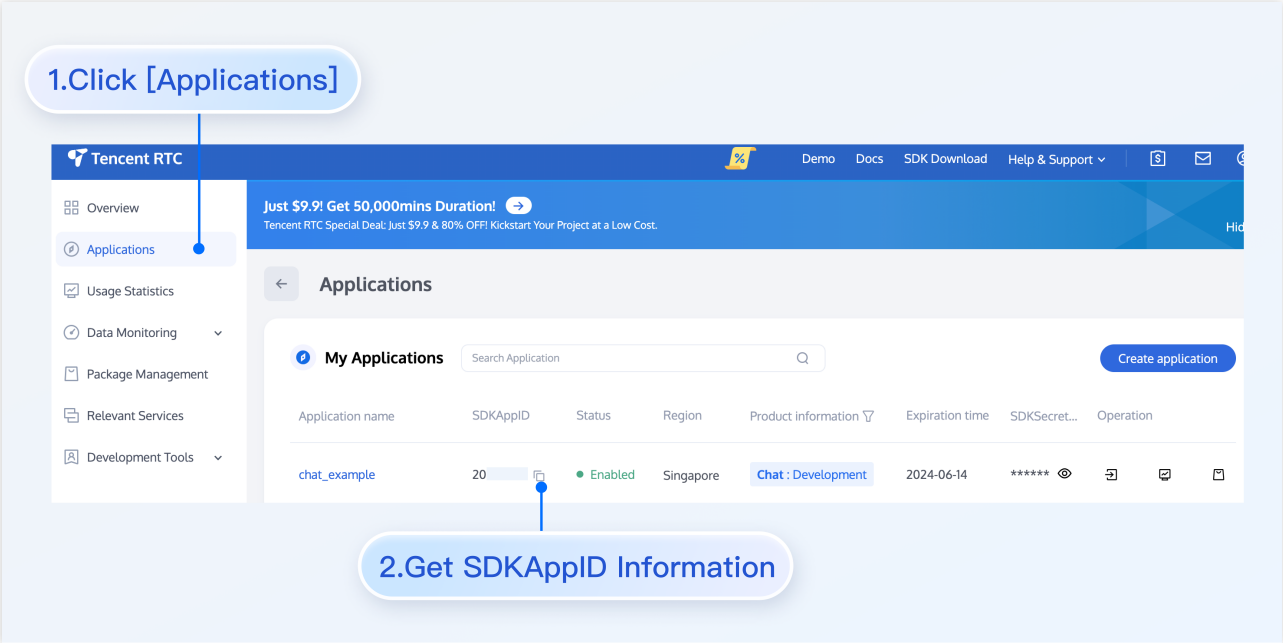


```
// 初始写法
export * from "vue";
// 替换为
export * from "@vue/composition-api";
```

步骤4： 获取 SDKAppID 、userID 与 userSig

设置 `<TUIKit>` 中的相关参数 SDKAppID 、userID 以及对应的 userSig :

`SDKAppID` ， 可通过 Chat Console 在 `Applications` 中获取:



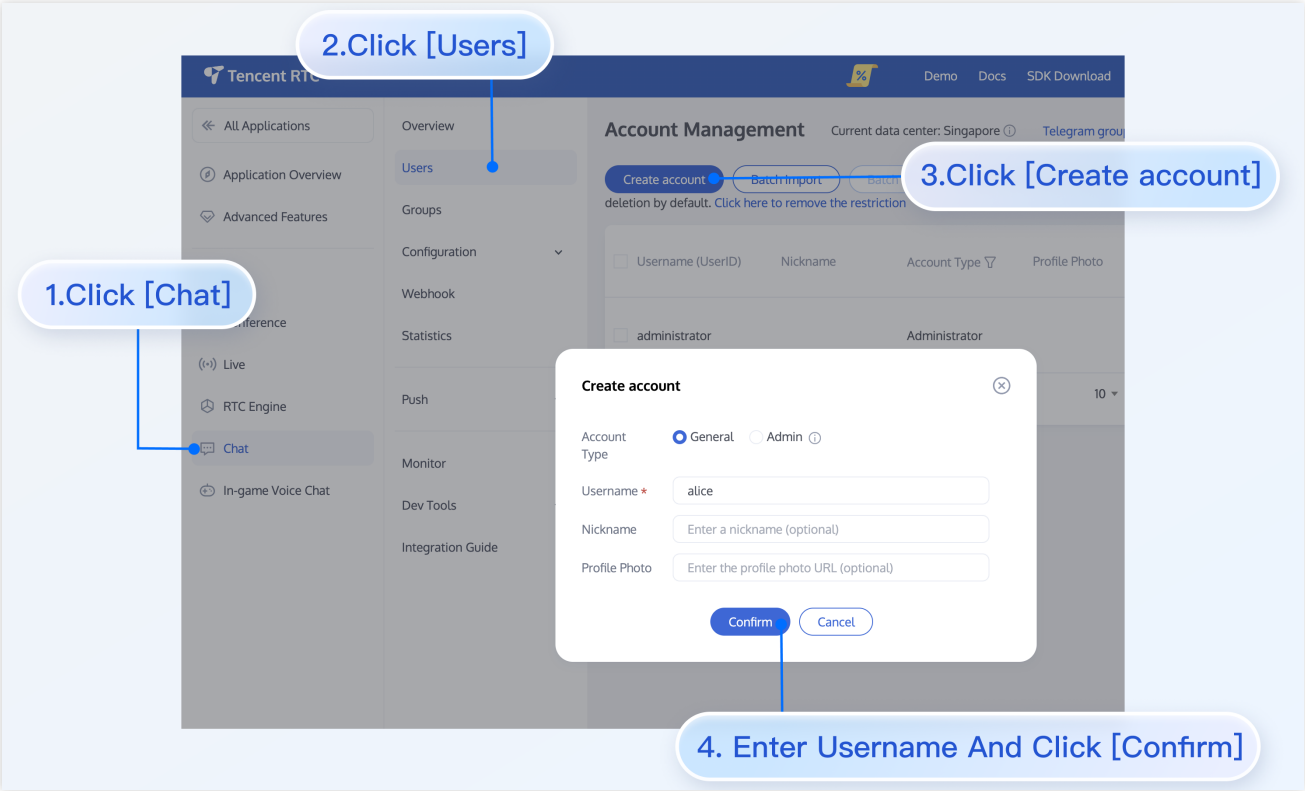
`userID`

单击进入您上面创建的 Application，会在左侧边栏看到 `Chat` 产品入口，单击进入。

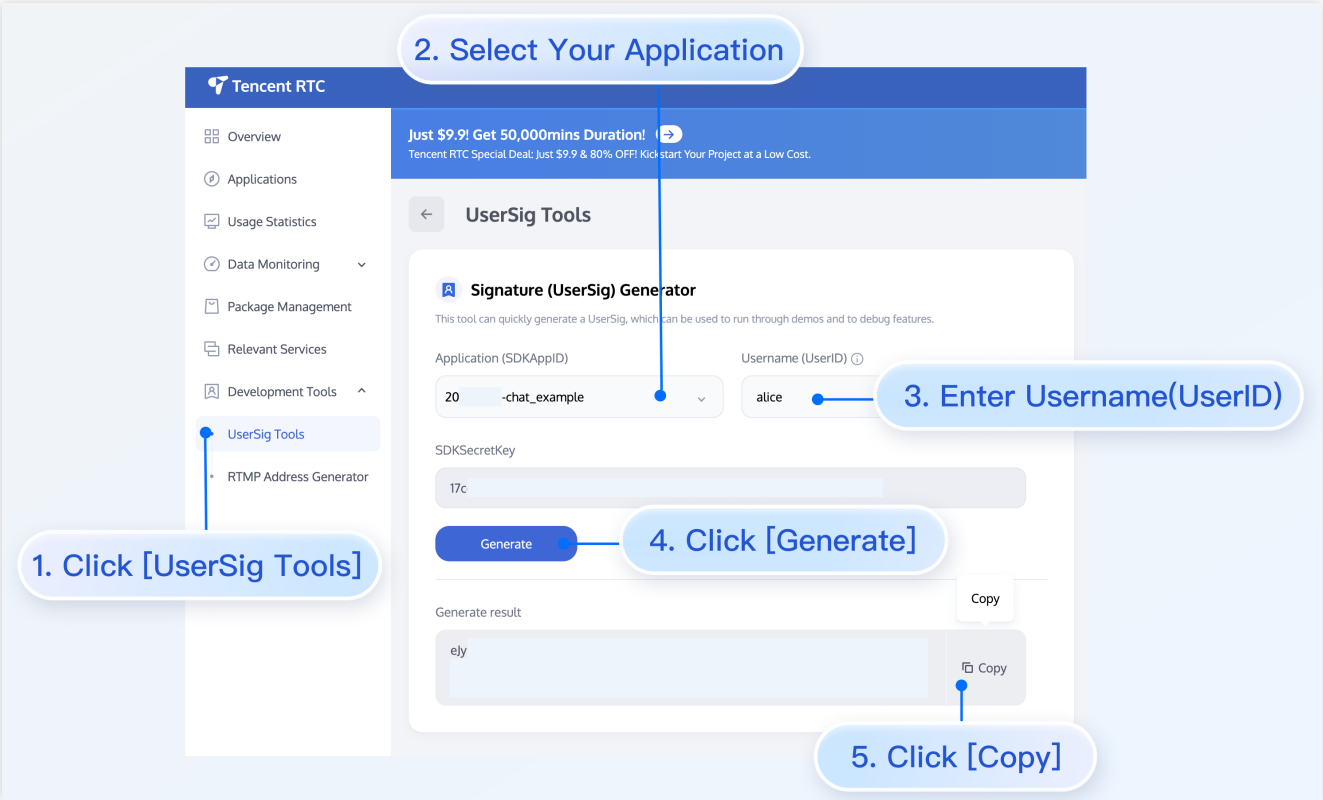
进入 Chat 产品子页面后，点击 `Users` ，进入用户管理页面。

单击 `Create account` ，弹出创建账号信息填写框。如果只是普通成员，我们建议您选择 `General` 类型。

为了您更好的体验消息收发等功能，建议您创建两个 userID。



`userSig` ，可使用控制台提供的开发工具实时生成，开发工具请点击 Chat Console > Development Tools > UserSig Tools > Signature (UserSig) Generator 。



步骤5：启动项目

执行以下命令启动项目：

vue-cli

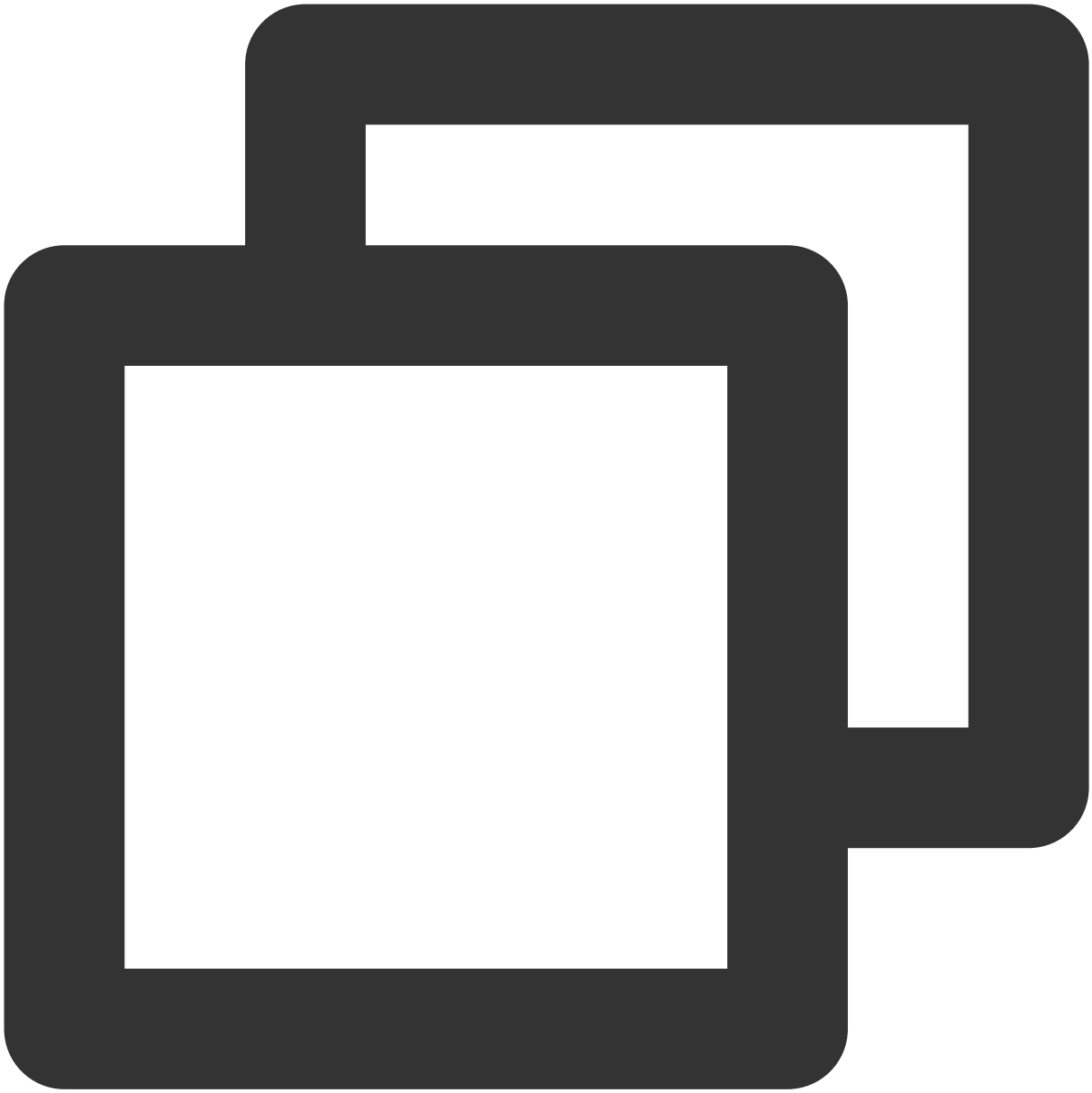
vite

说明：

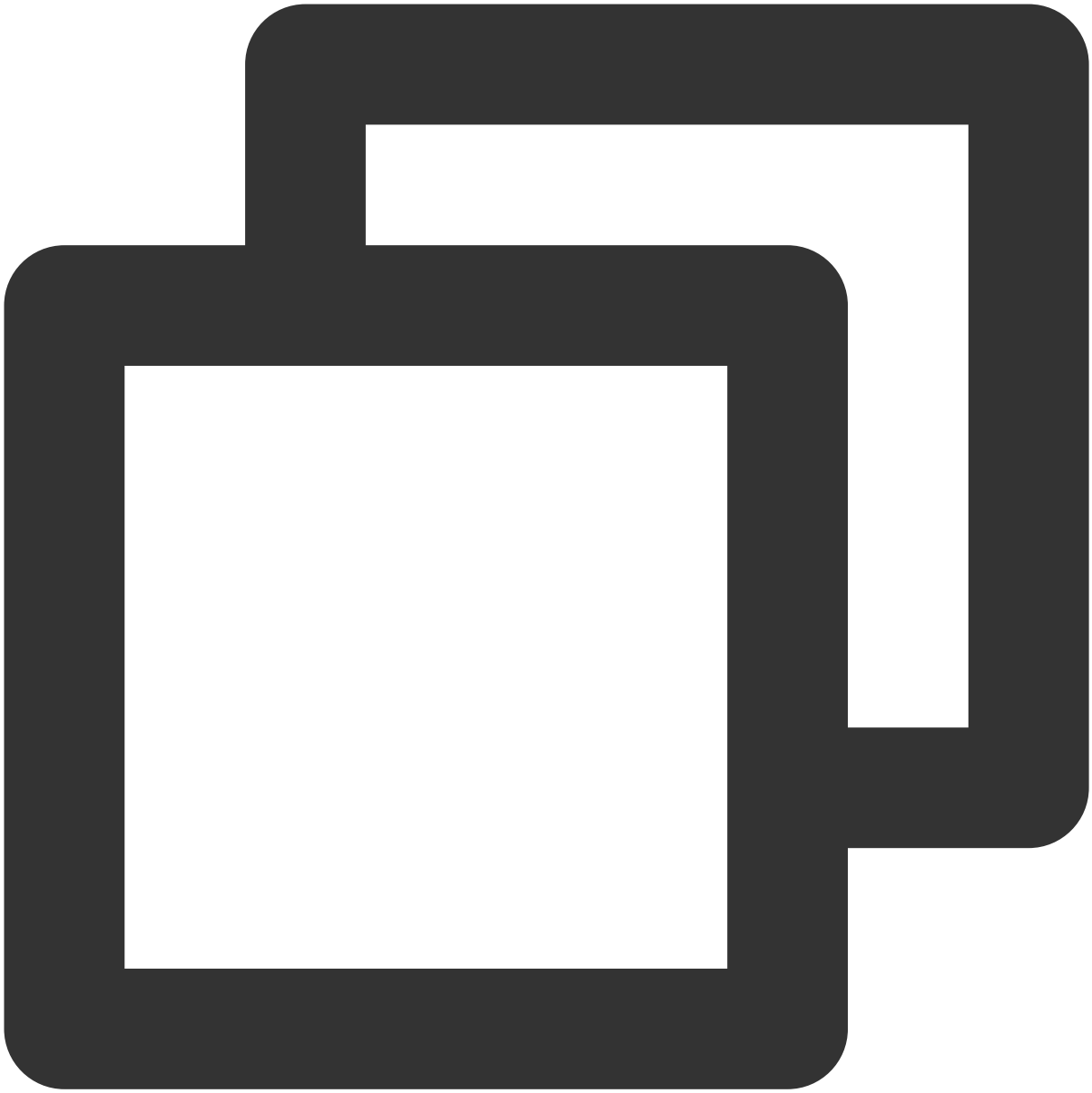
由于 vue-cli 默认开启 webpack 全局 overlay 报错信息提示，为了您有更好的体验，**建议您关闭全局 overlay 报错提示。**

webpack4

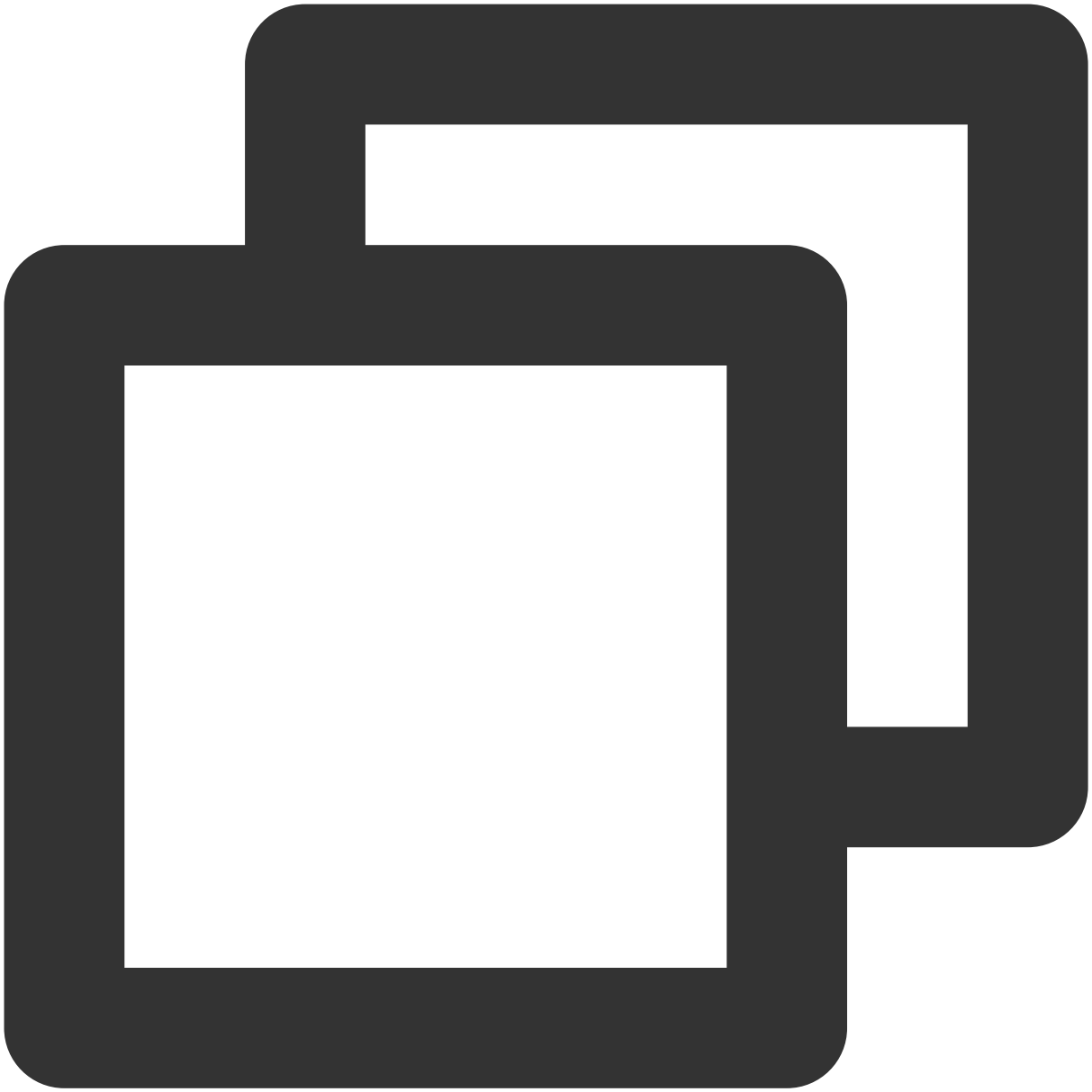
webpack3



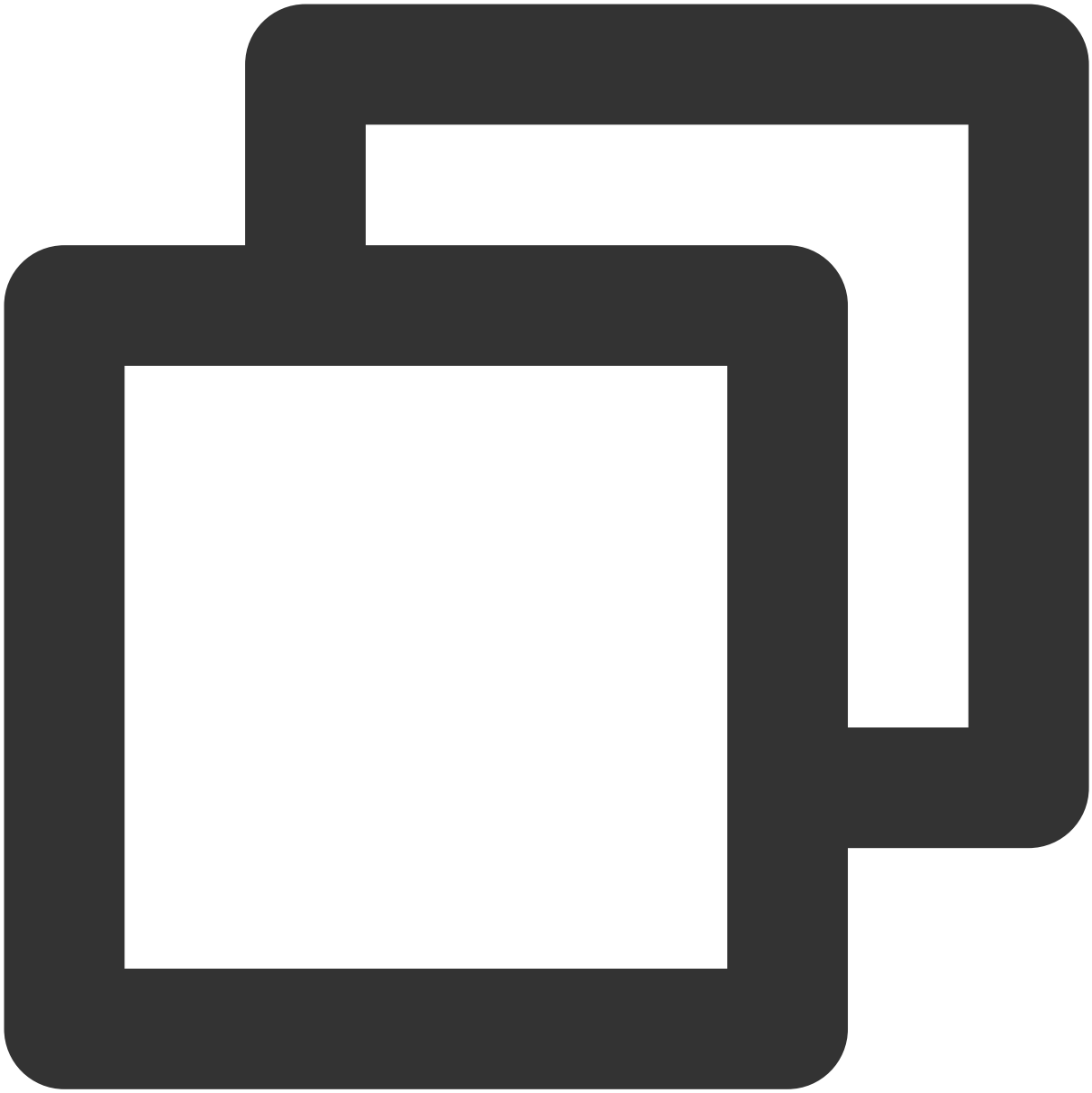
```
module.exports = defineConfig({
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```



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



```
npm run serve
```

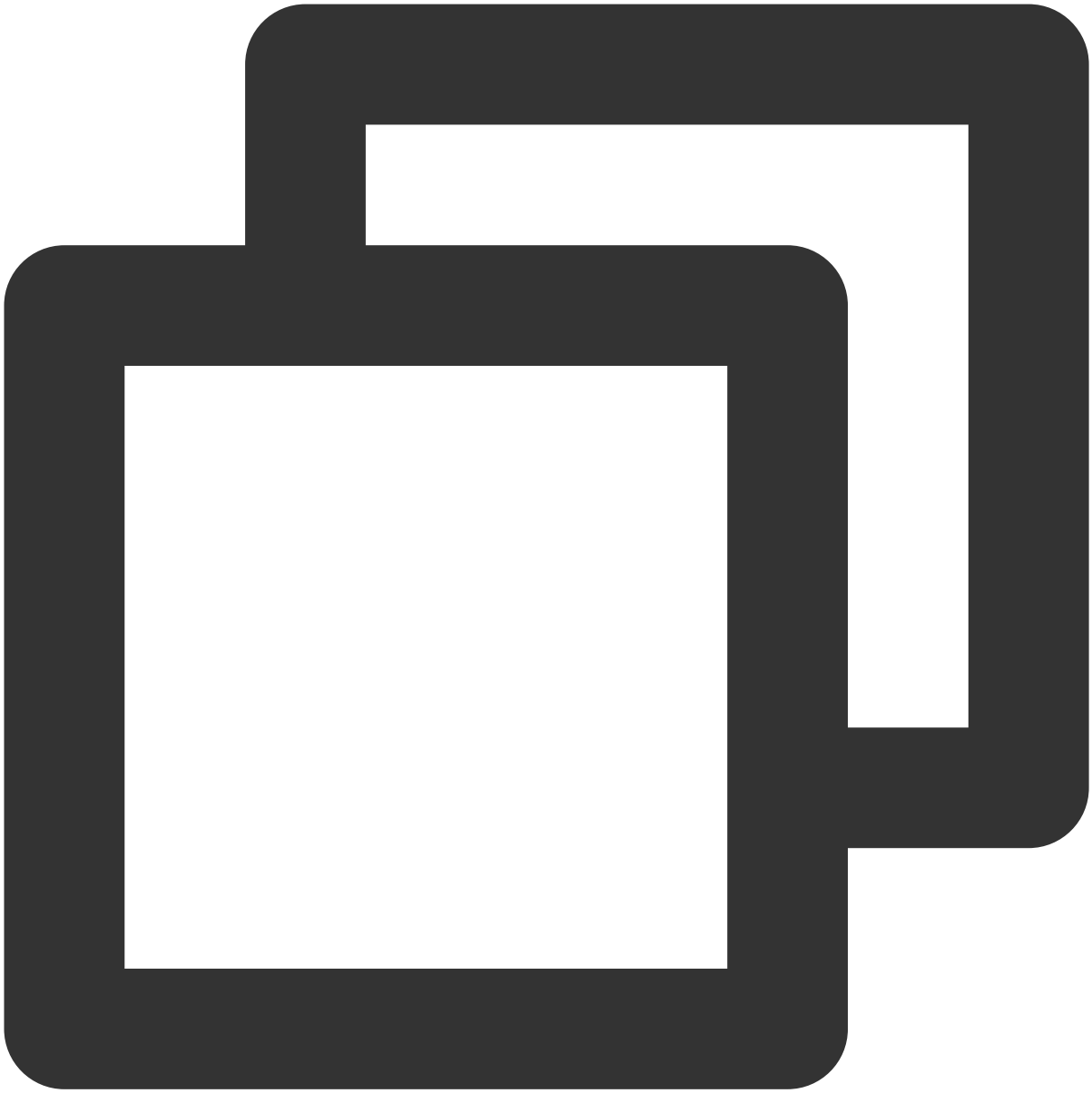


```
npm run dev
```

附加项：切换语言

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

您可以通过以下方式切换语言，更多切换方式请查看国际化界面语言。



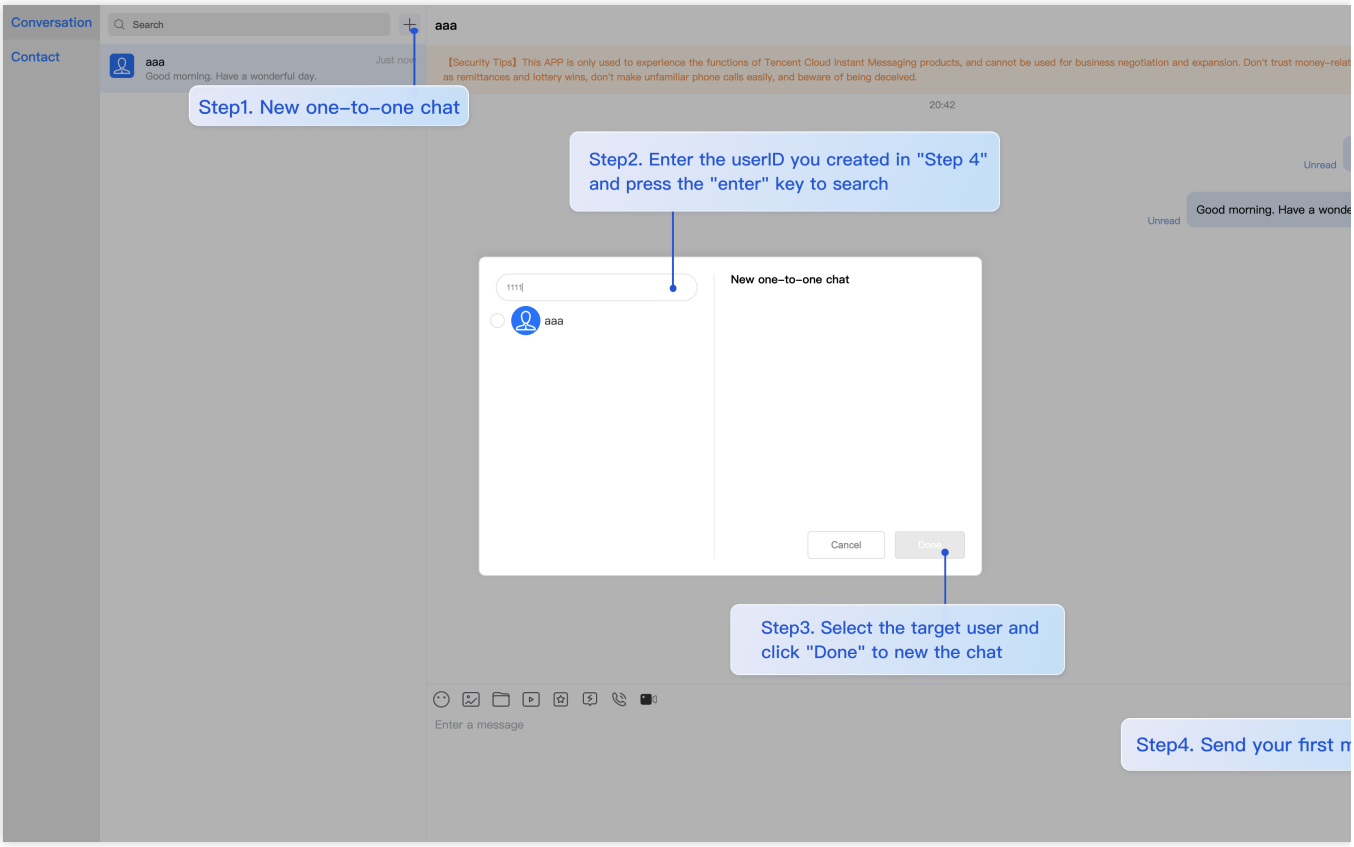
```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

步骤6：发送您的第一条消息

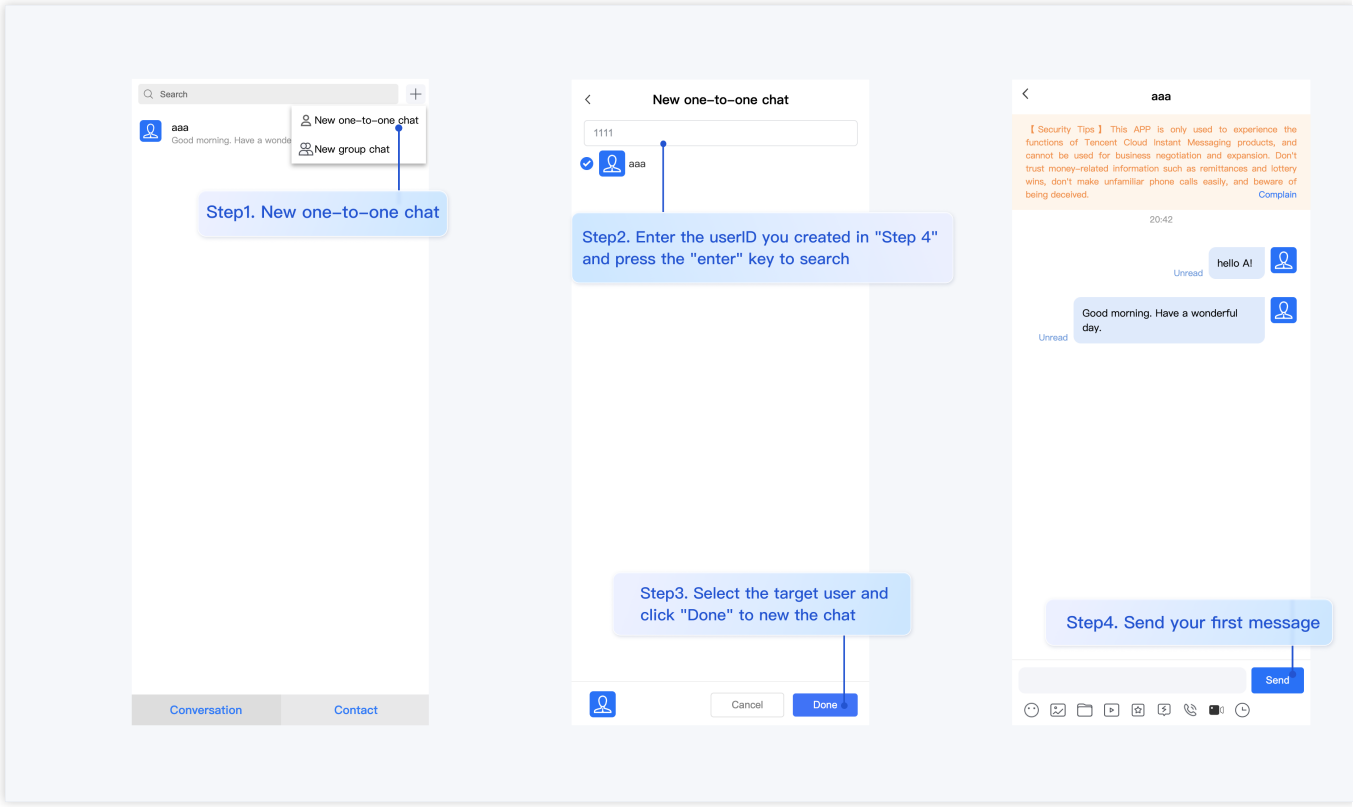
1. 项目启动之后单击左上角**发起单聊**。
2. 进入**发起单聊**弹窗。在搜索栏输入 步骤4 中创建的 userID，选中后单击**完成**。

3. 在输入框中输入消息并单击**发送**。

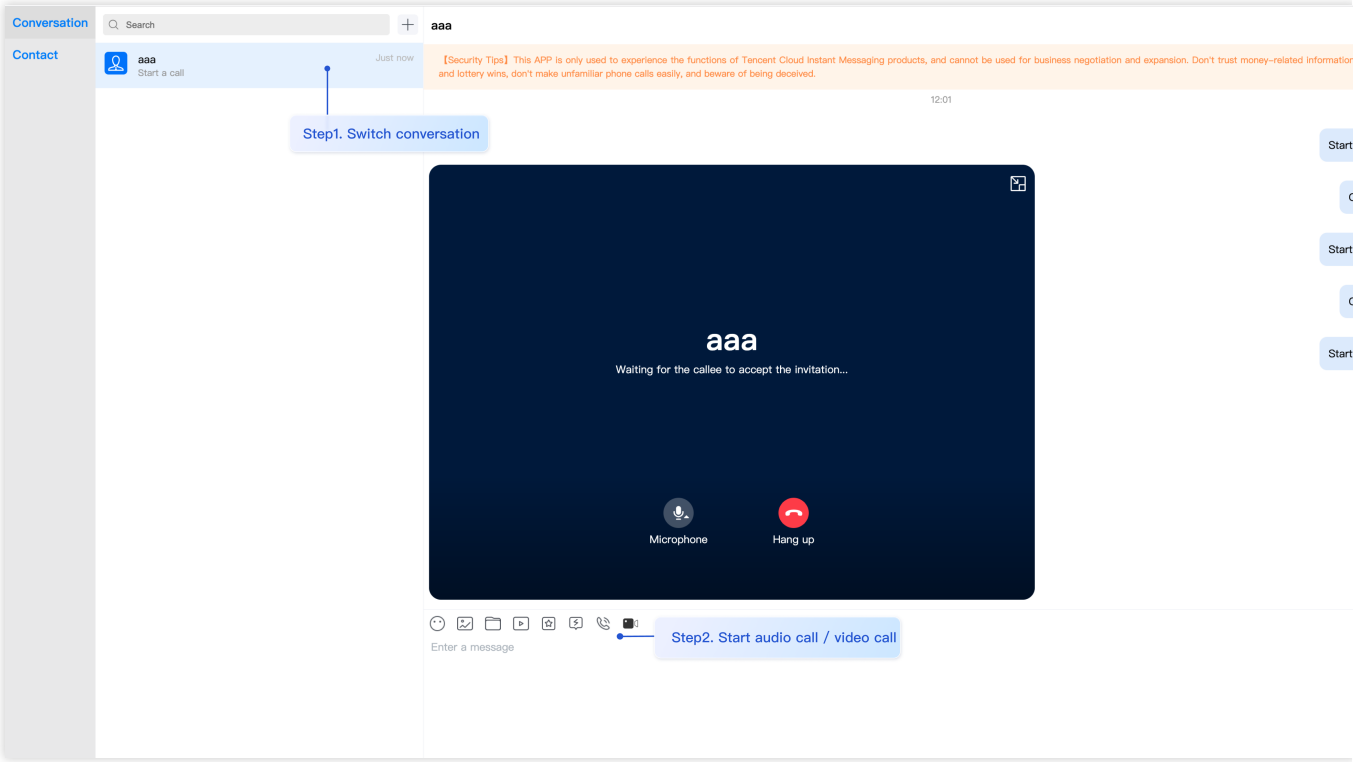
Web 端 “发送您的第一条消息” 具体步骤示例：



H5 端 “发送您的第一条消息” 具体步骤示例：



步骤7：拨打您的第一通电话



常见问题

产品服务类问题

1. 音视频通话能力包未开通？音视频通话发起失败？

请单击 音视频通话 > 常见问题 查看解决方案。

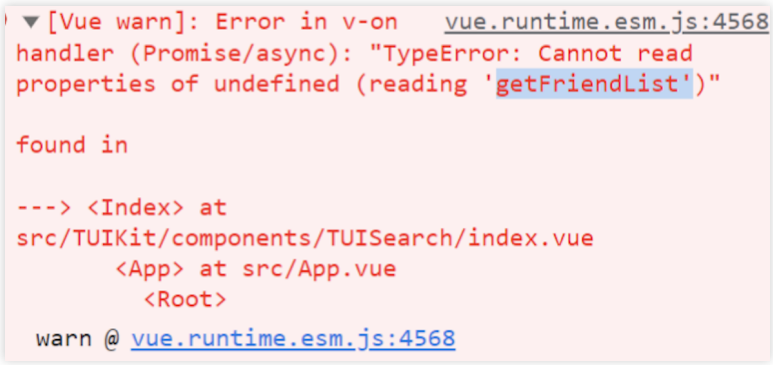
2. 什么是 UserSig？如何生成 UserSig？

UserSig 是用户登录 Chat 的密码，其本质是对 UserID 等信息加密后得到的密文。
UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。
注意：
本文示例代码采用的获取 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**。 正确的 UserSig 签发方式请参见上文。

接入报错类问题

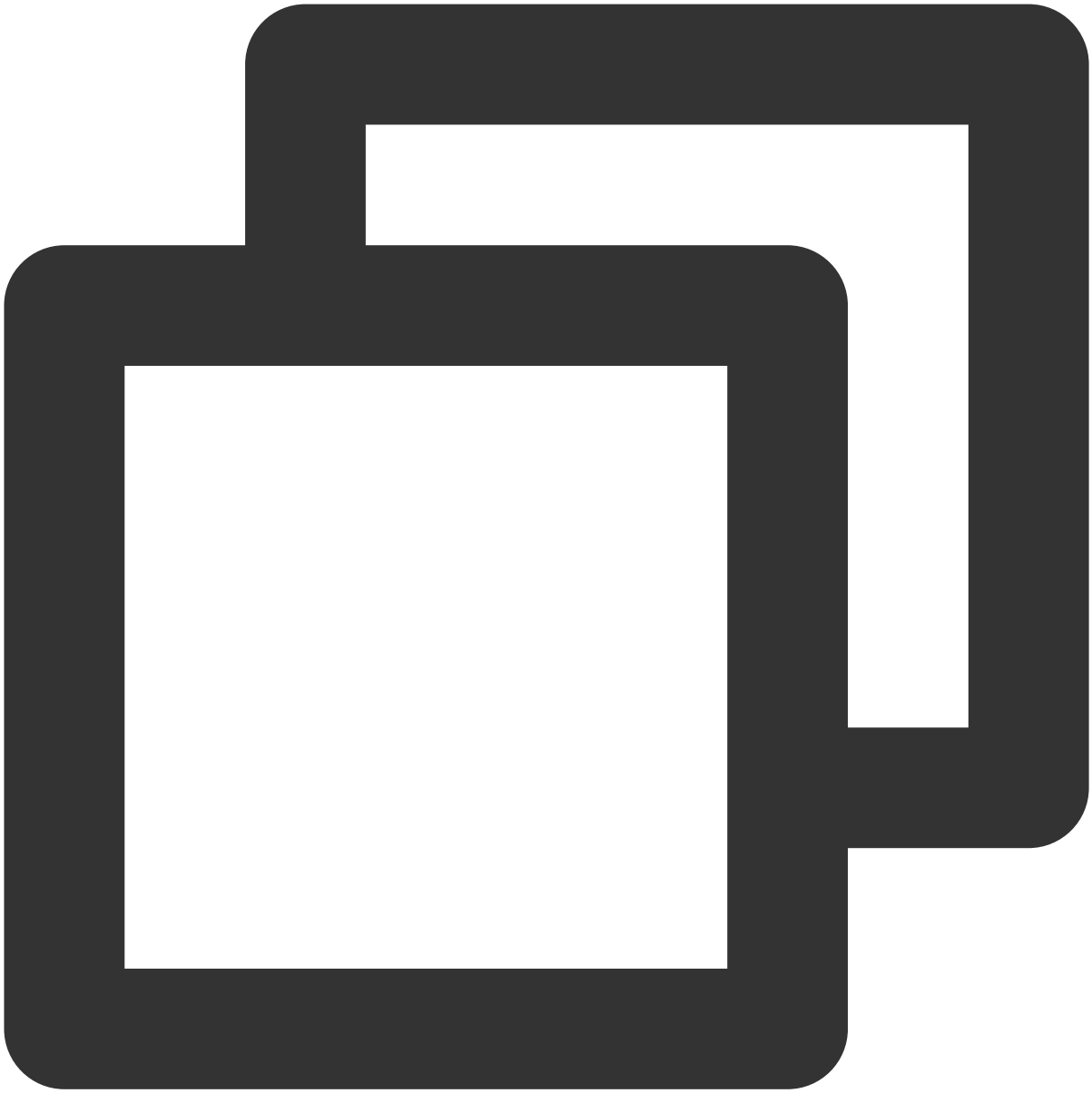
1. 运行时报错："TypeError: Cannot read properties of undefined (reading "getFriendList")"

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



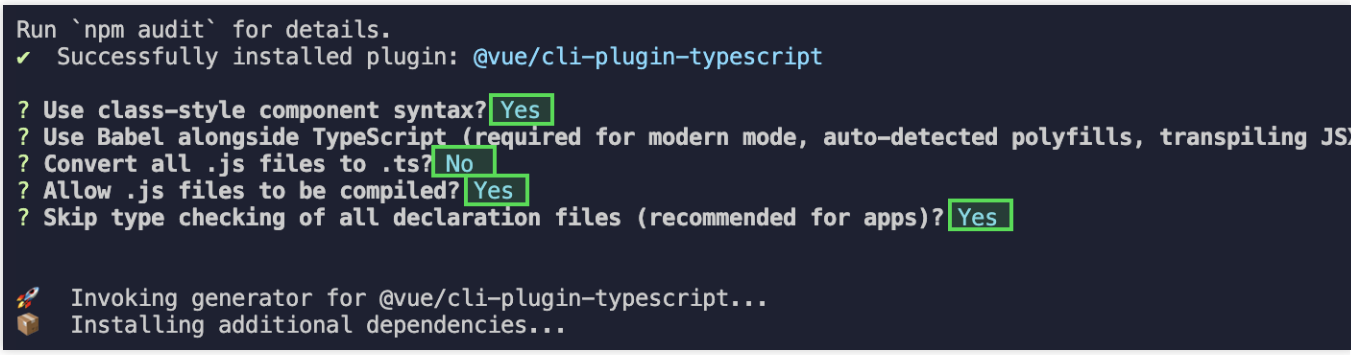
2. js 工程如何接入 TUIKit 组件?

TUIKit 仅支持 ts 环境运行，您可以通过渐进式配置 typescript 来使您项目中已有的 js 代码 与 TUIKit 中 ts 代码共存。
vue-cli
vite
请在您 vue-cli 脚手架创建的工程根目录执行：



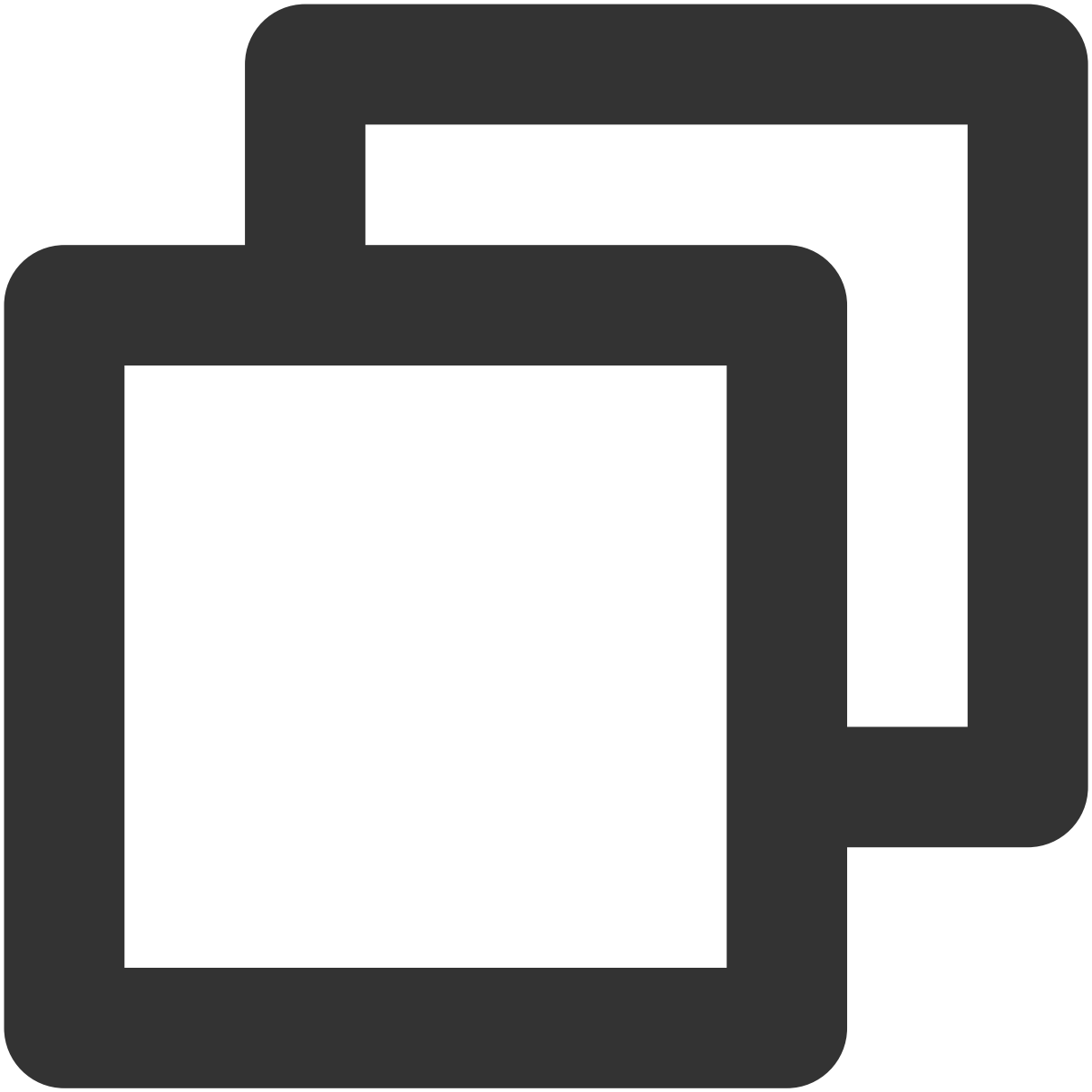
```
vue add typescript
```

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



完成以上步骤后，请重新运行项目！

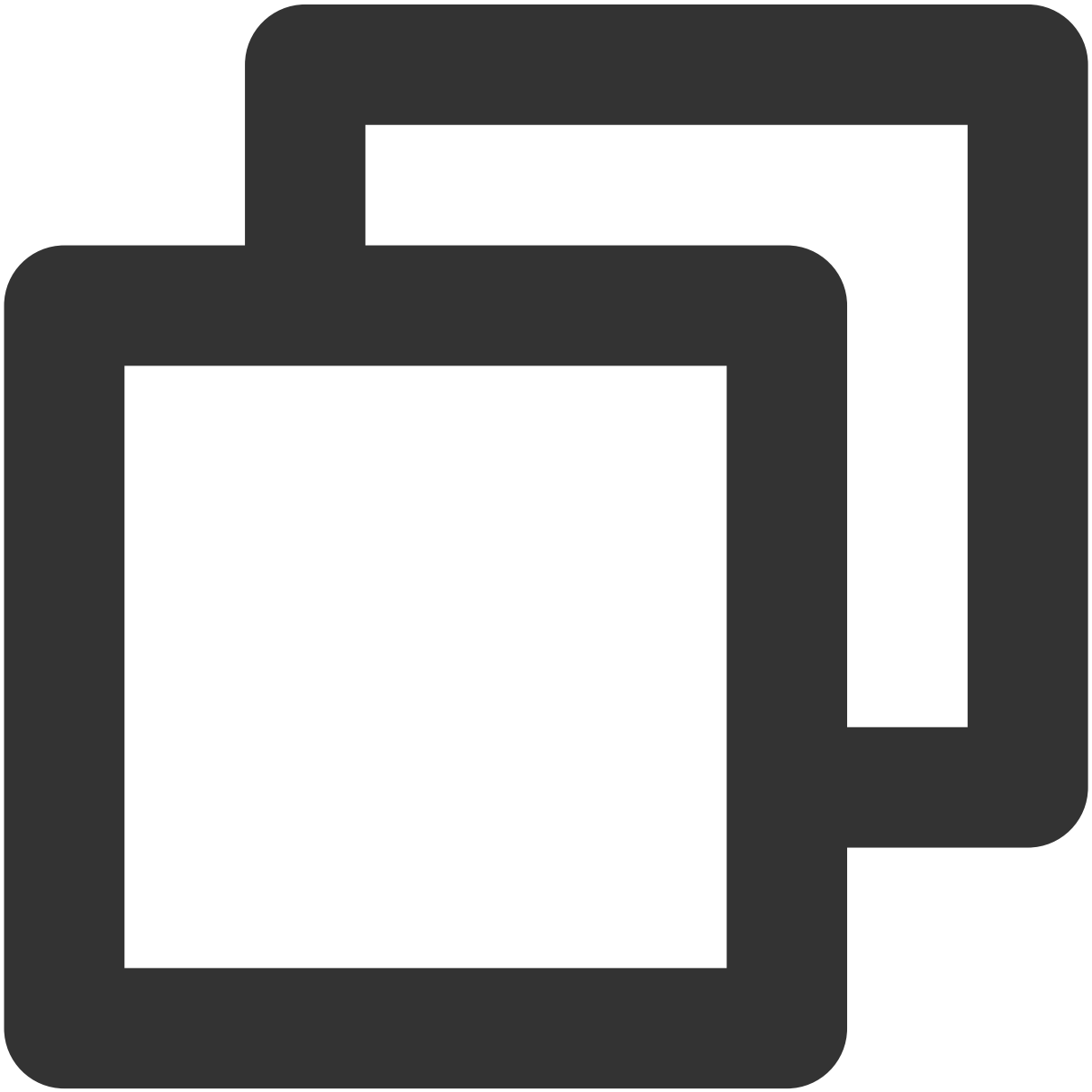
请在您 vite 创建的工程根目录执行：



```
npm install -D typescript
```

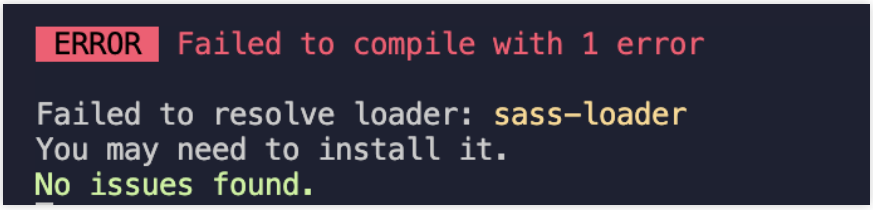
3. 运行时报错：/chat-example/src/TUIKit/components/TUIChat /message-input/message-input-editor.vue.ts（8，23）TS1005: expected.

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

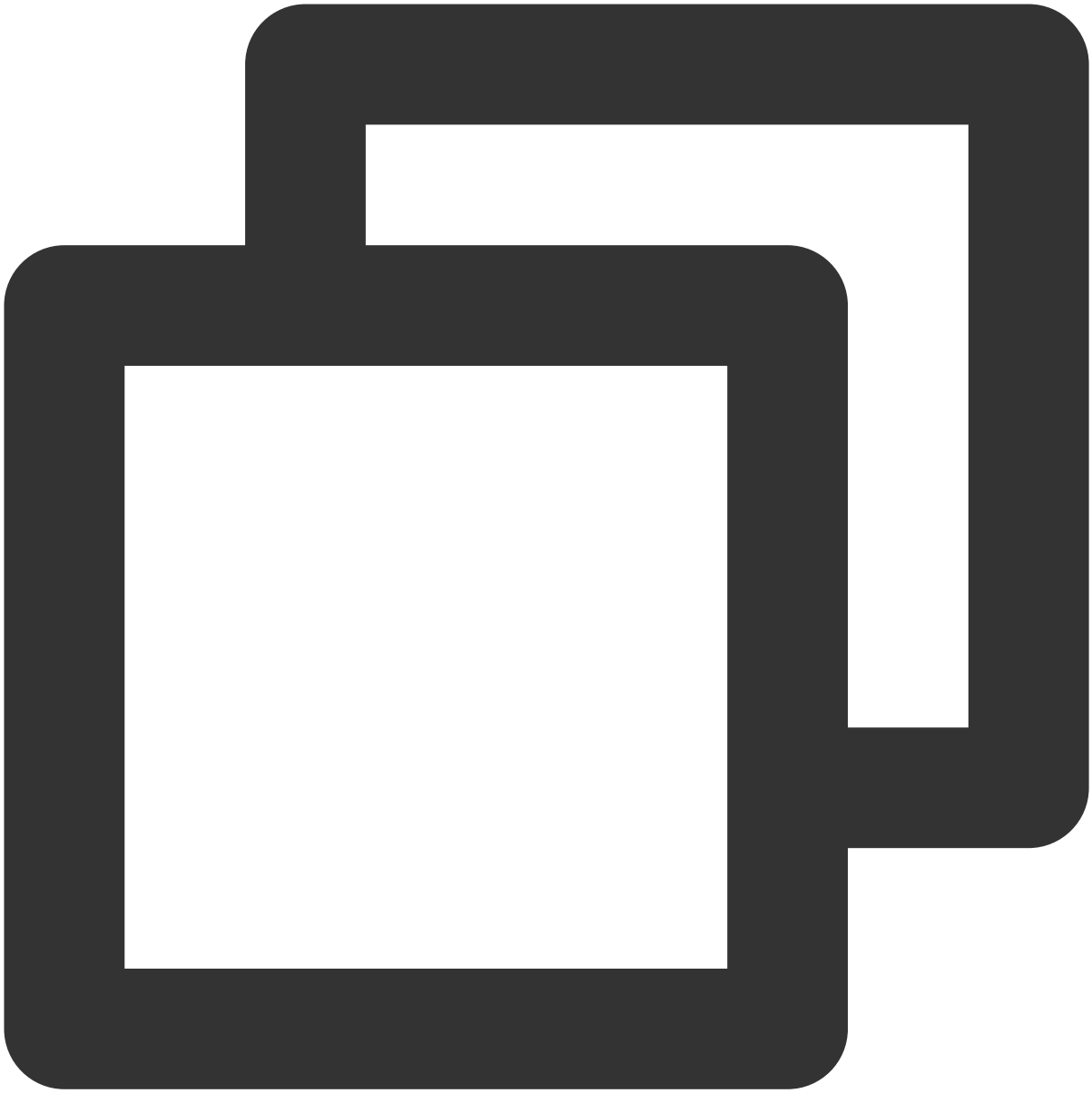


```
npm install -g @vue/cli@5.0.8
```

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



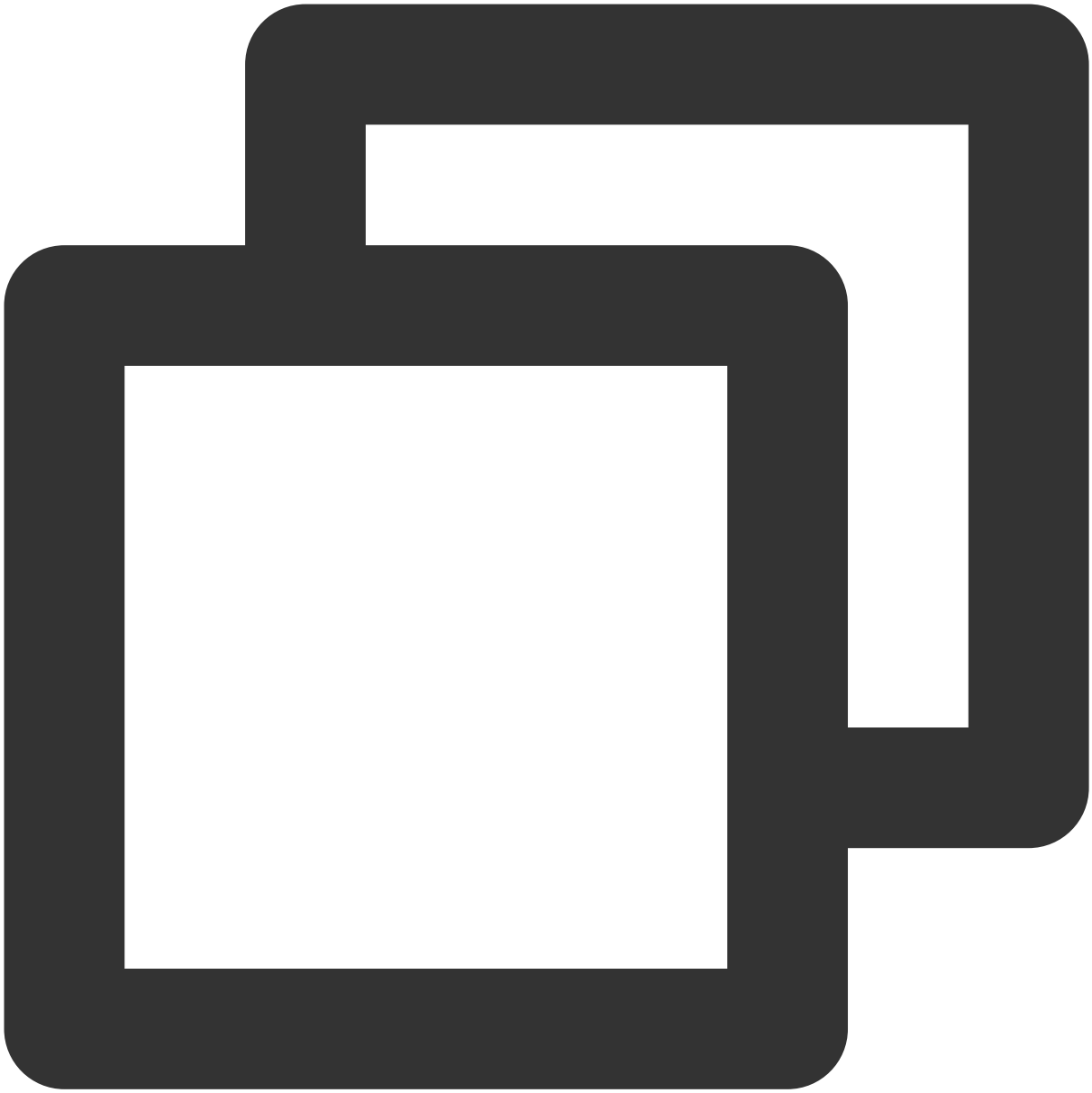
出现以上报错信息，是因为您未安装 `sass` 环境导致，请执行以下命令进行 `sass` 环境安装:



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

5. ESLint 其他报错？

若 chat-uikit-vue 拷贝到 src 目录汇总与您本地项目代码风格不一致导致报错，可将本组件目录屏蔽，如在项目根目录增加 .eslintignore 文件：



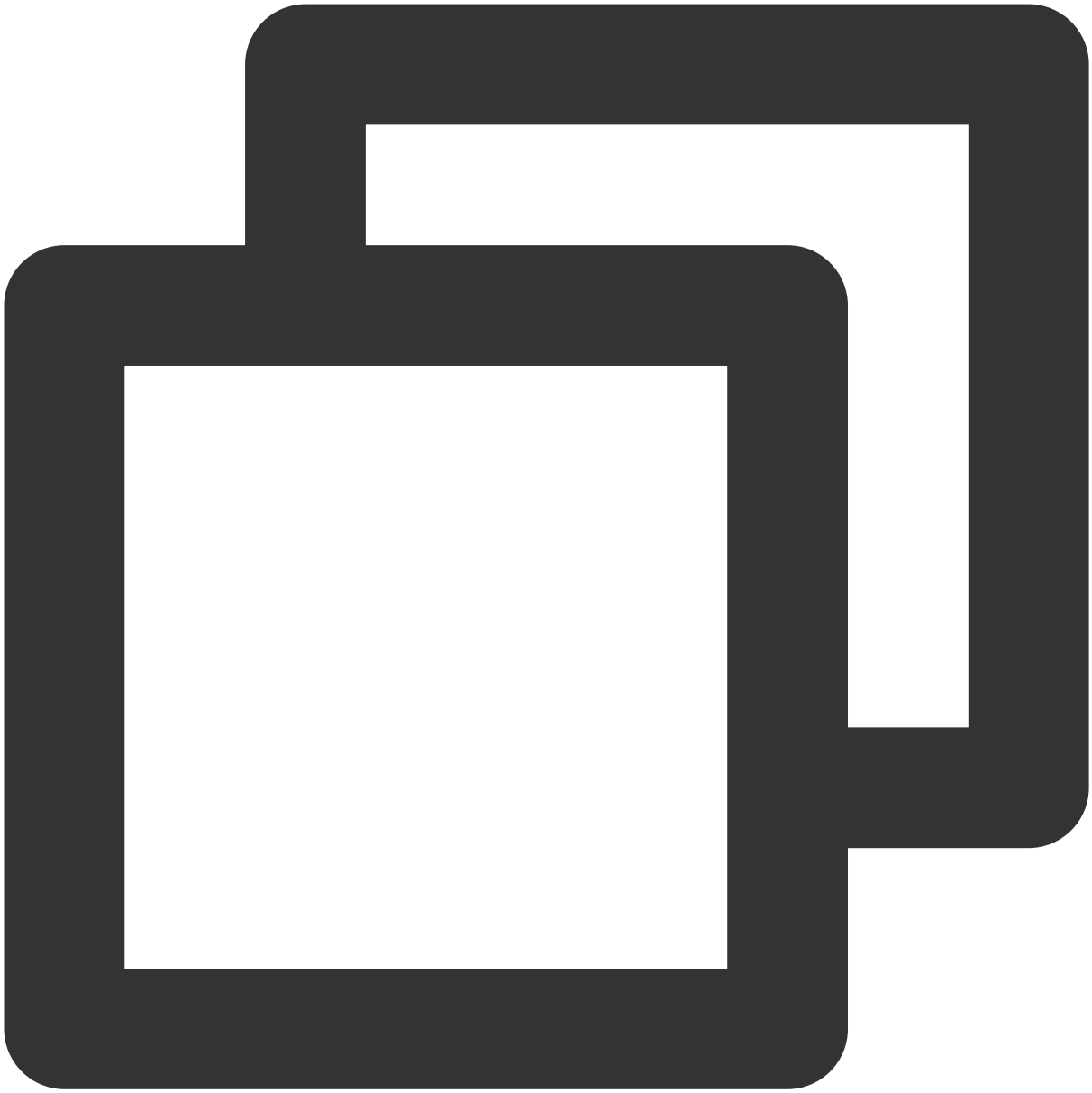
```
# .eslintignore
src/TUIKit
```

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

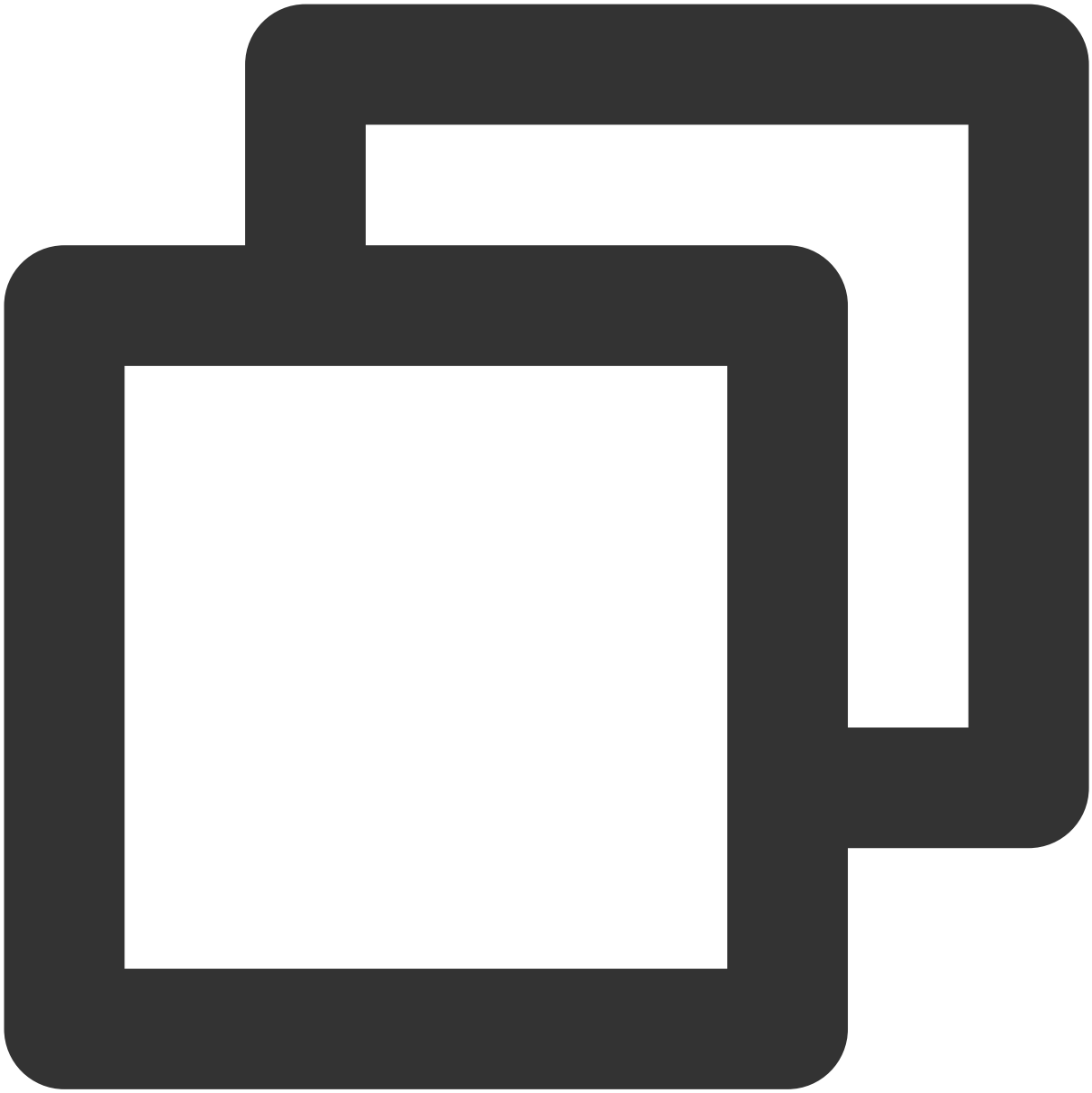
可以在您项目根目录的 vue.config.js 中进行关闭：

webpack4

webpack3



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



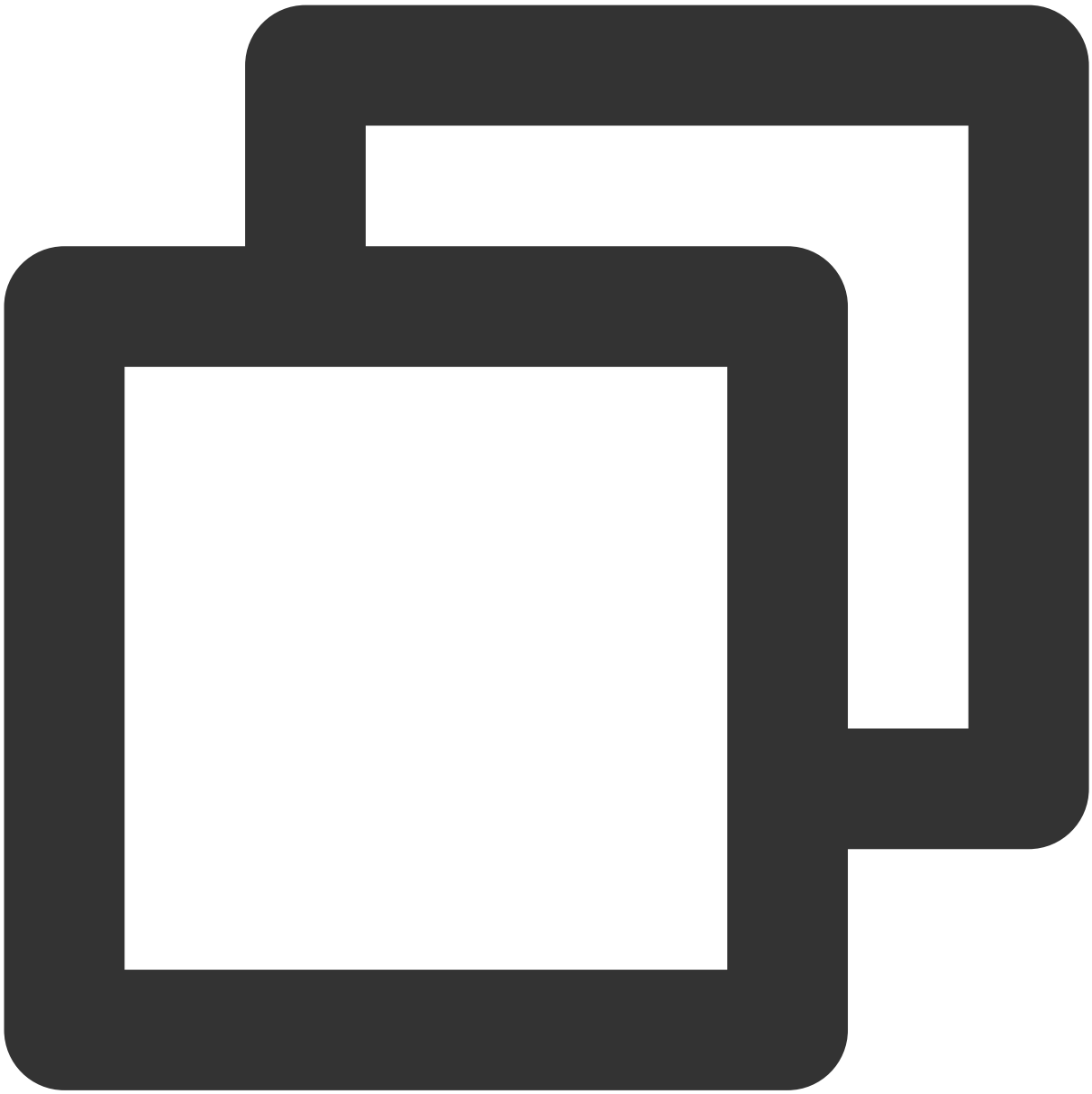
```
module.exports = {
  ...
  devServer: {
    overlay: false,
  },
};
```

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

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

版权所有：腾讯云计算（北京）有限责任公司

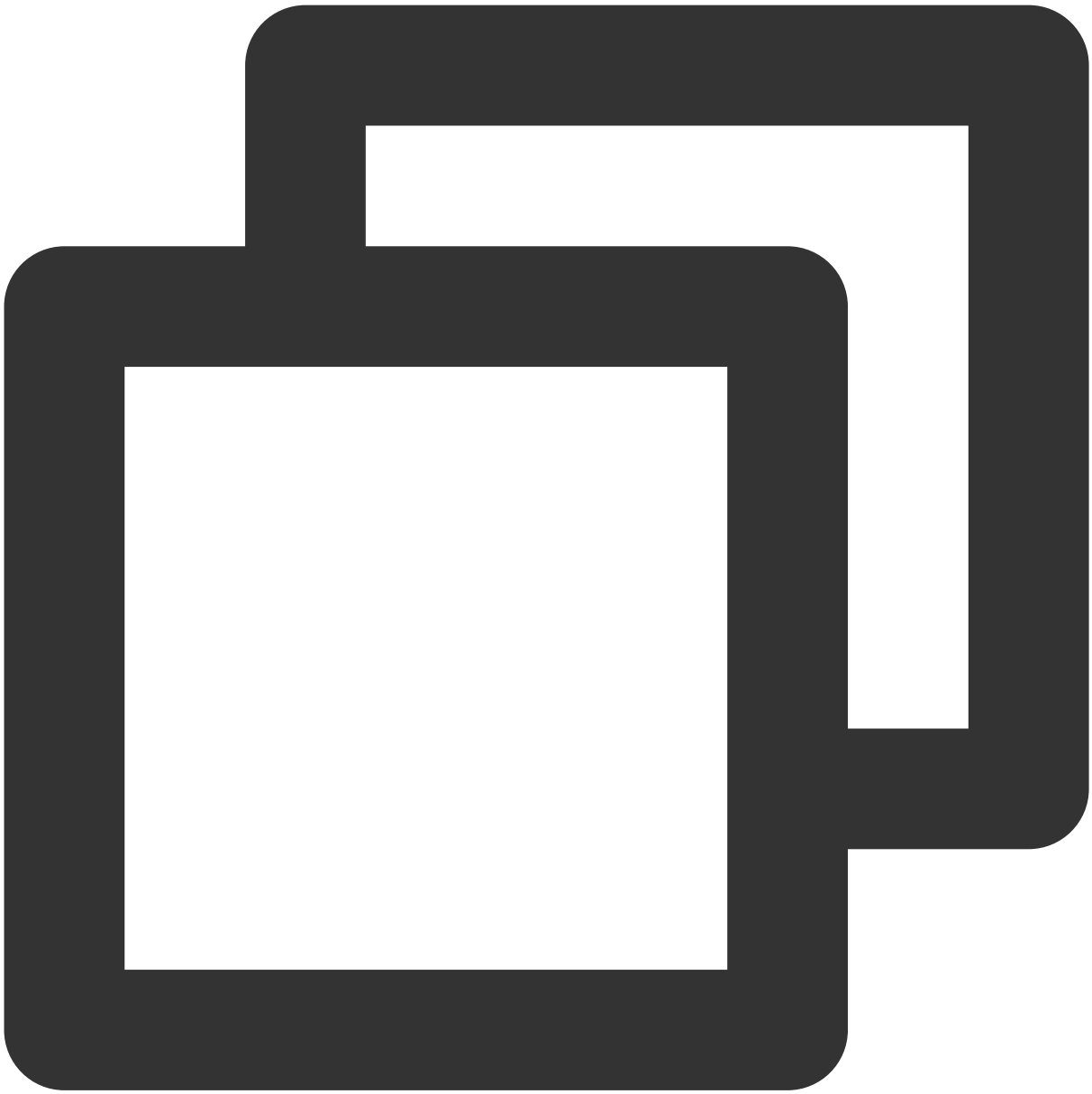
如果您出现此问题，您可以在 .eslintrc.js 文件中进行如下配置：



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

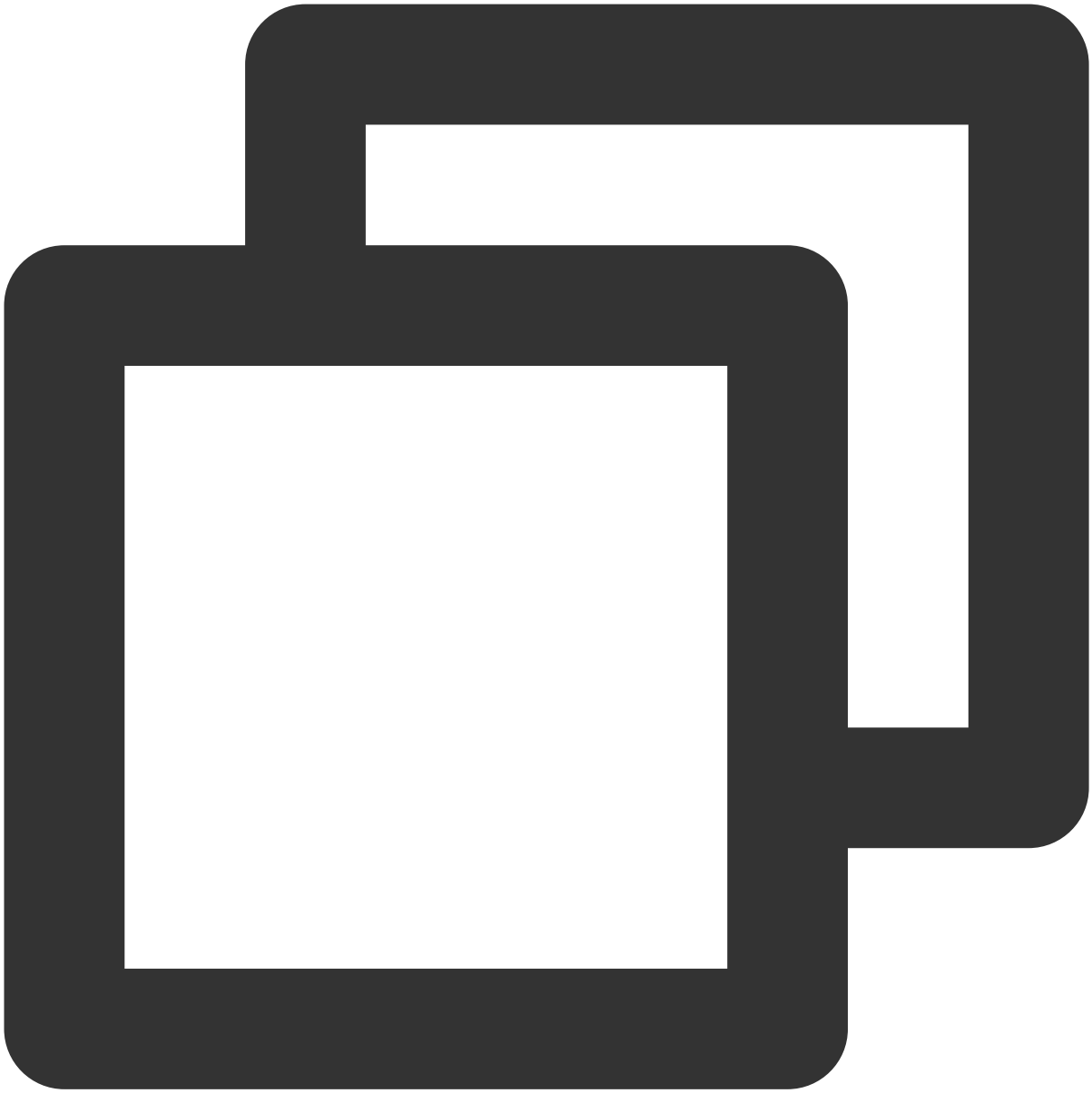
8. 出现 ERESOLVE unable to resolve dependency tree 怎么办？

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



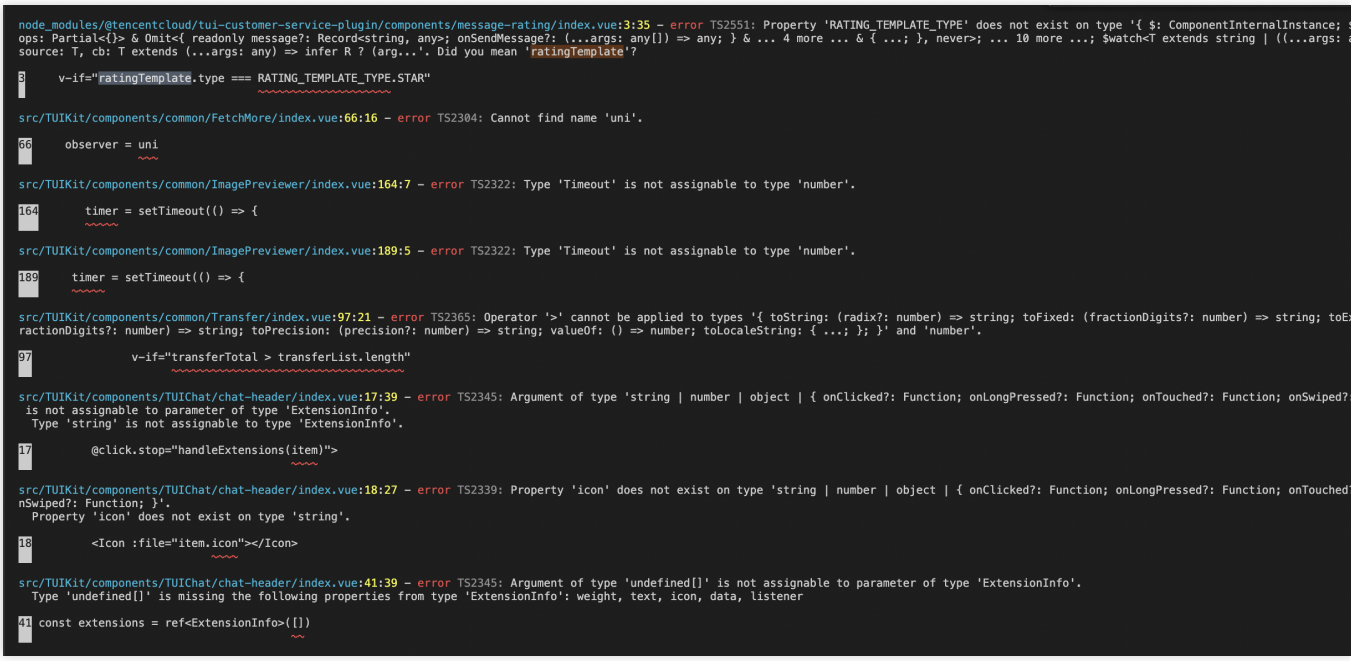
```
npm install --legacy-peer-deps
```

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



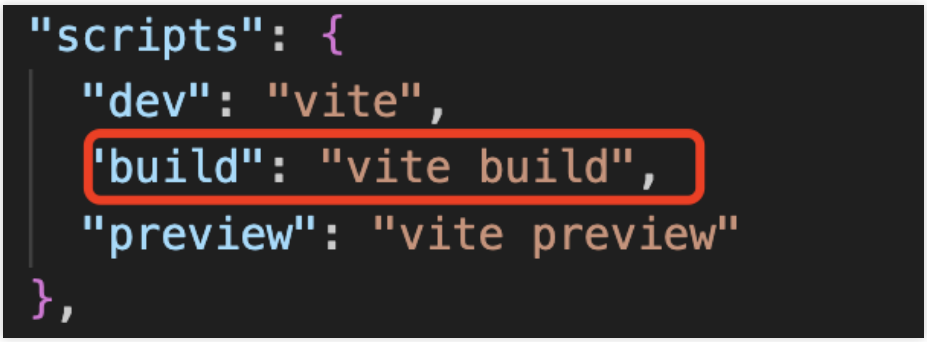
```
// 如果您是 vue2.7 项目，请在您项目根目录执行
npm i vue@2.7.9 vue-template-compiler@2.7.9
// 如果您是 vue2.6 项目，请在您项目根目录执行
npm i vue@2.6.14 vue-template-compiler@2.6.14
```

10. vite 项目 npm run build 之后 ts 报错?



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

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



交流与反馈

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

相关文档

Vue2 & Vue3 UIKit 相关：

chat-uikit-vue npm
Vue2 Demo源码及跑通示例
Vue3 Demo源码及跑通示例

Vue2 & Vue3 UIKit 逻辑层: engine 相关

chat-uikit-engine npm 仓库
chat-uikit-engine 接口文档

Unity

最近更新时间：2024-03-20 17:30:45

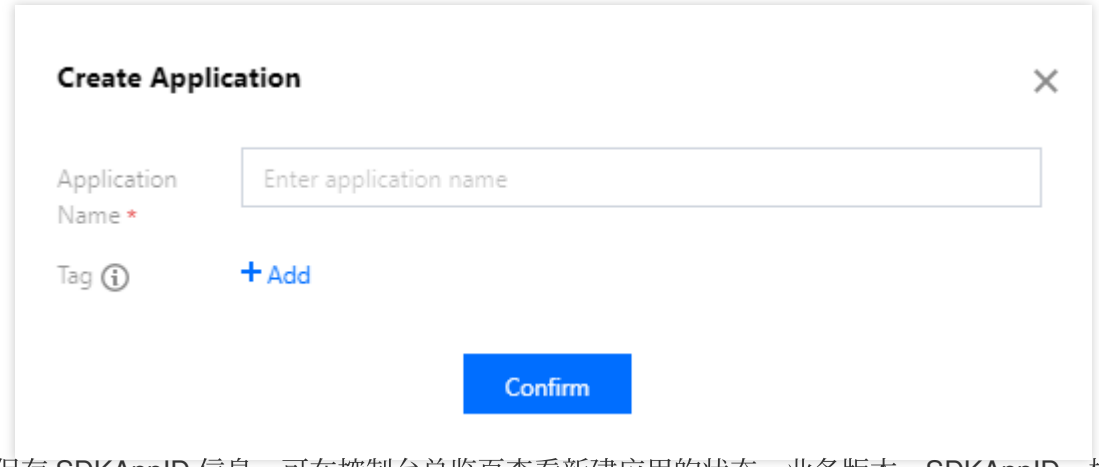
IM for Unity on iOS or Android.
此 IM(Instant Messaging) Unity UIKit & UIKit Demo 是基于Tencent Cloud IM Chat SDK实现的游戏场景业务 UI 组件库，目前包含了会话 (Conversation)和聊天 (Chat)组件，收发文字消息、收发表情包消息、自定义表情包等功能。在您的 Unity 项目下引用此 UIKit 可助您快速搭建您的聊天系统。
IM Unity UIKit & UIKit Demo 链接
即时通信 IM Demo

环境要求

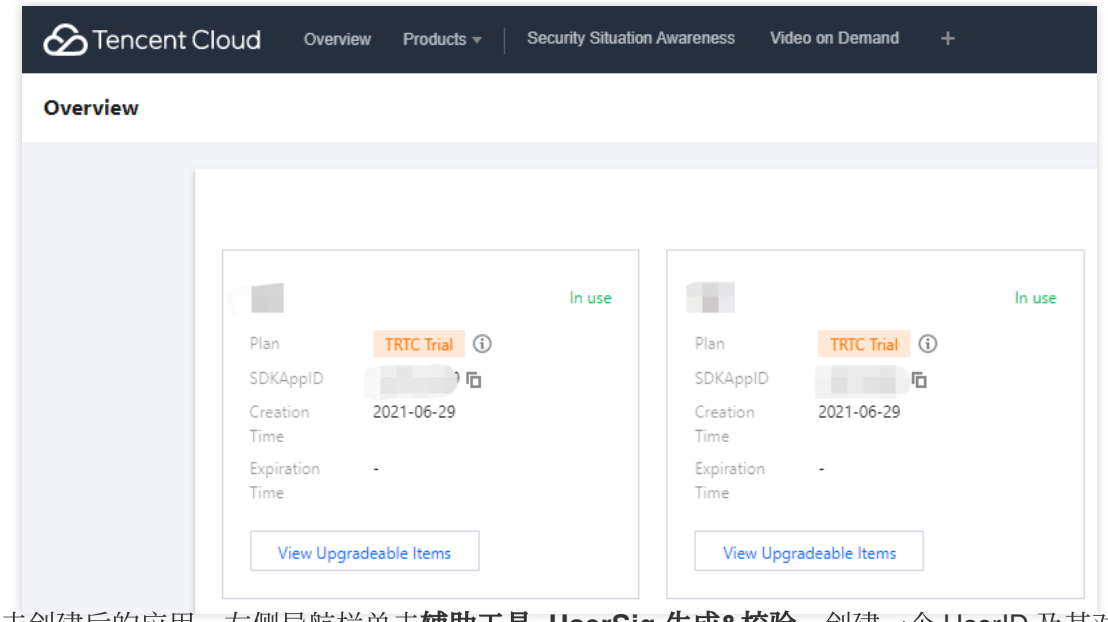
平台	版本
Unity	2019.4.15f1 及以上版本。
Android	Android Studio 3.5及以上版本，App 要求 Android 4.1及以上版本设备。
iOS	Xcode 11.0及以上版本，请确保您的项目已设置有效的开发者签名。

前提条件

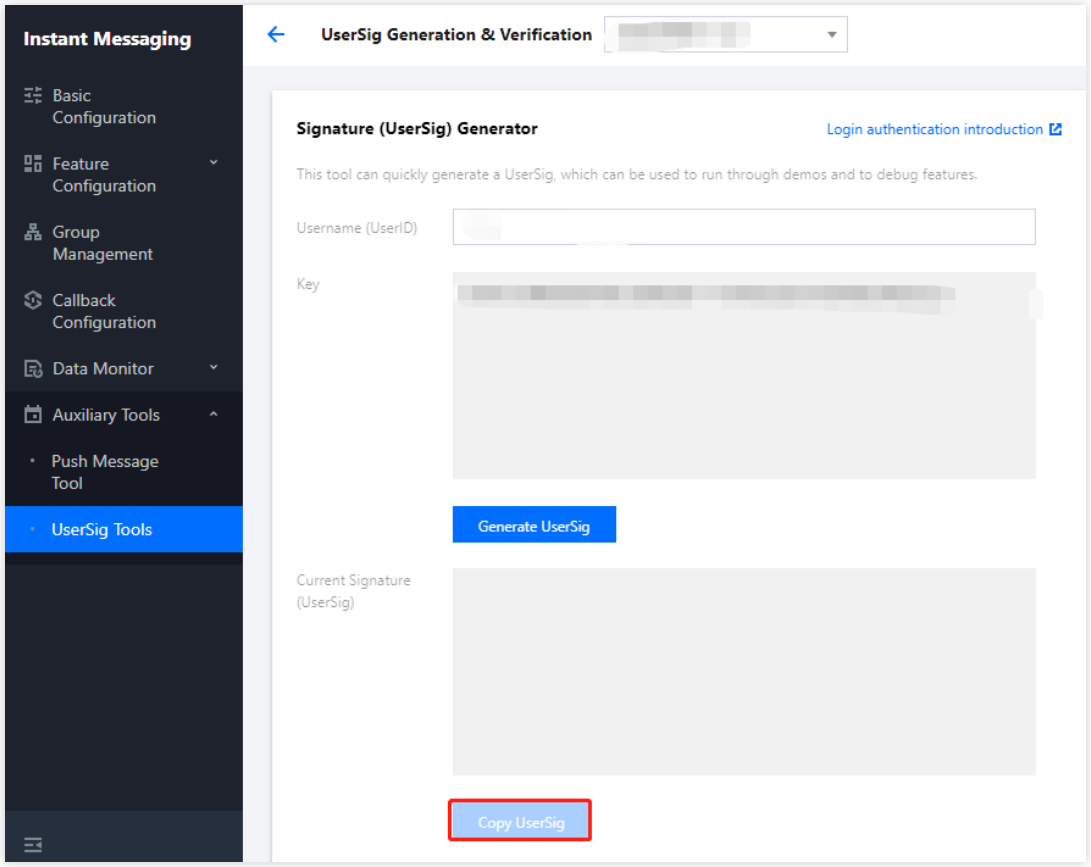
您已 注册腾讯云 账号，并完成 实名认证。
1. 登录 即时通信 IM 控制台。
说明：
如果您已有应用，请记录其 SDKAppID 并 获取密钥信息。
同一个腾讯云账号，最多可创建300个即时通信 IM 应用。若已有300个应用，您可以先 停用并删除 无需使用的应用后再创建新的应用。**应用删除后，该 SDKAppID 对应的所有数据和服务不可恢复，请谨慎操作。**
2. 单击**创建新应用**，在**创建应用**对话框中输入您的应用名称，单击**确定**。



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



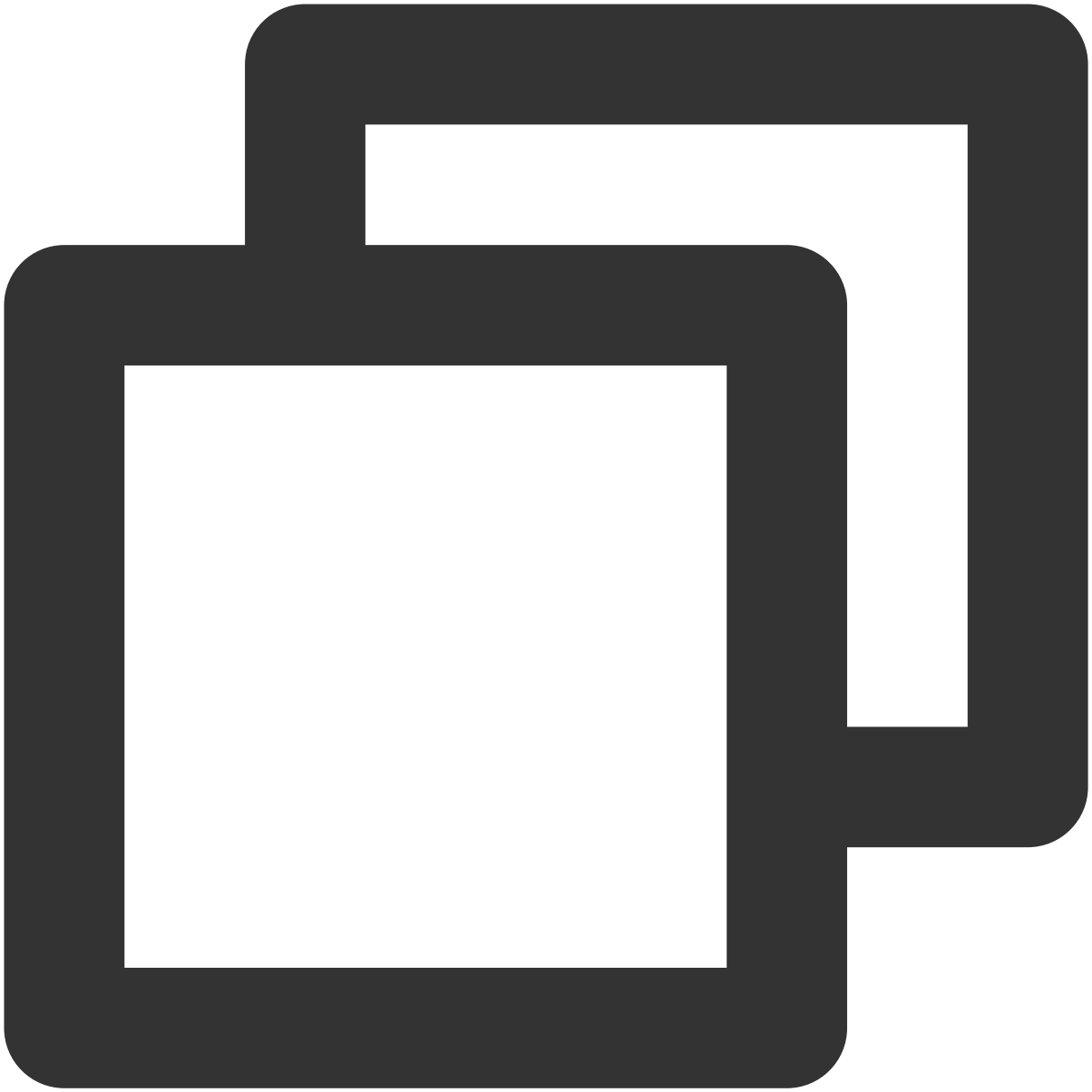
4. 单击创建后的应用，左侧导航栏单击**辅助工具>UserSig 生成&校验**，创建一个 UserID 及其对应的 UserSig，复制签名信息，后续登录使用。



如何将UIKit导入到项目中

导入AssetPackage

1. 创建/启动已存在的 Unity 项目。
2. 在 Packages/manifest.json 文件中的 dependencies 下添加：



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

3. 下载 UIKit github目录下的 chat-uikit-unity.unitypackage，并导入资源包。

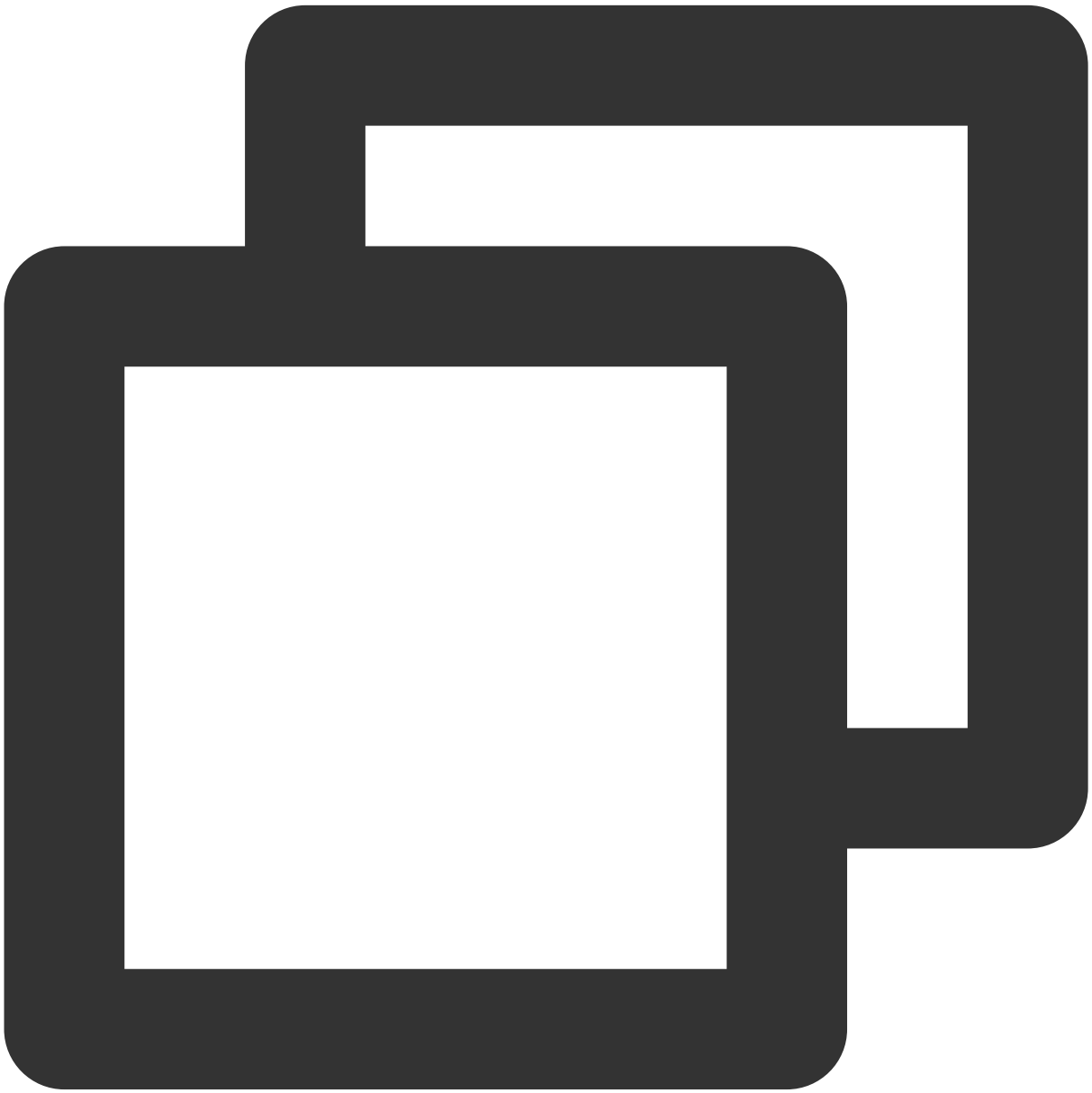
初始化并登录

初始化并登录 IM 有两种方式:

组件外部: 整个应用初始化并登录一次即可。需要自行添加相应的事件回调。
组件内部: 通过配置的方式将参数传入组件内部。建议您使用内部登录，UIKit 已帮您绑定了相应的事件回调，包括接收新消息的事件以及会话列表更新的事件。

方法1:组间外部

在您创建的 Unity 项目中初始化 IM, 注意 IM 应用只需初始化一次即可。如若在现有 IM 项目中集成可跳过该步骤。

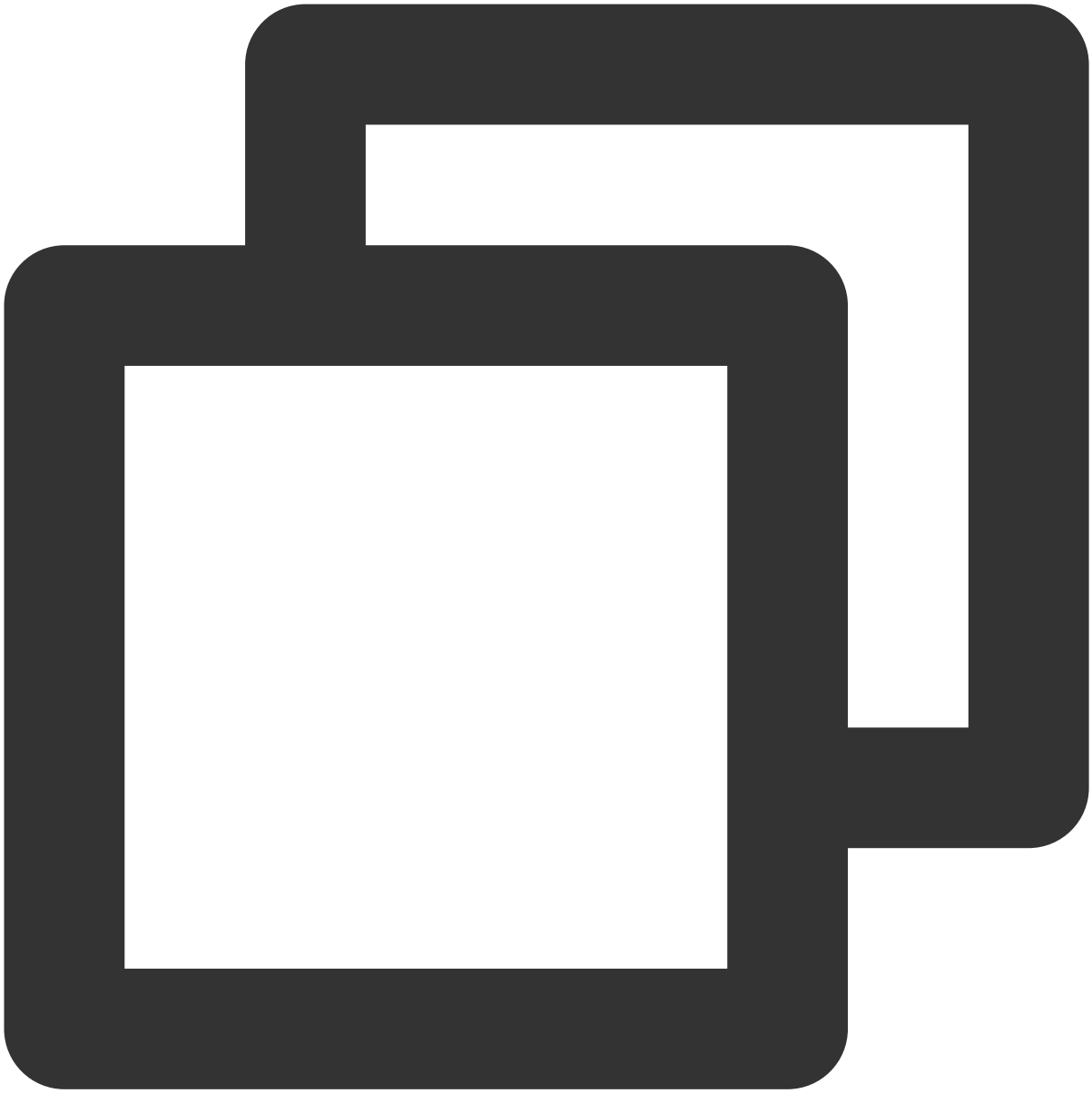


```
public static void Init() {
        int sdkappid = 0; // 从即时通信 IM 控制台获取应用 SDKAppID。
        SdkConfig sdkConfig = new SdkConfig();
```

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

        sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM

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

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

方法2:组件内部

您也可将SDKAppID、UserSig、UserID通过配置的方式传入组件内部进行 IM 的初始化和登录。（与demo运行方式相同）

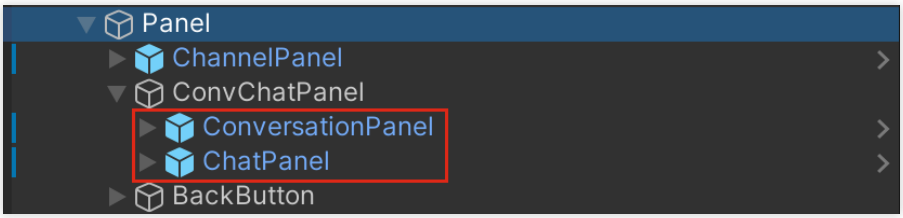


```
using com.tencent.imsdk.unity.uikit;
// demo 登录逻辑如下
public static void Init() {
  Core.SetConfig(sdkappid, userId, sdkUserSig);
  Core.Init();
  Core.Login();
}

// 若直接跑unity demo，推荐通过 Main 页面的手机号/验证码输入框登录。可以保证监听回调等正常运行。
// 由于电脑端unity不支持手机号登录，请将 userid输入到手机号输入框，userSig输入到验证码输入框。
// Assets/Example/Scripts/Main.cs
```

```
private void Login()
{
  // 将之前的Login函数内容注释掉，添加以下内容
  loginButton.interactable = false;
  Core.SetConfig(Config.sdkappid, phoneNumber.text, captcha.text);
  Core.Init();
  Core.Login(HandleAfterLogin);
  loginButton.interactable = true;
 }
```

使用 Conversation和Chat预制件

您可将下列预制件放入您的场景中，修改相应样式和layout。



项目结构

Assets/Example

该目录对应实际项目运行时显示的内容，包含Scenes的两个页面，分别对应的代码为 `Main.cs（登录界面）` 和 `Chat.cs（聊天界面）` 。

Chat 里包含单聊、群聊的内容，可以获取到会话（好友）列表并发送文字、表情包消息。Chat里的内容由 `Prefabs` 里的组件构成，可以通过修改 `Prefabs` 修改显示内容和样式。

Assets/Prefabs

下列组件可以联合使用（参考Scenes的Chat页面），也可根据需求将组件单独修改并使用。

ChatPanel

消息历史列表

消息展示区 `ConvMessagePanel`

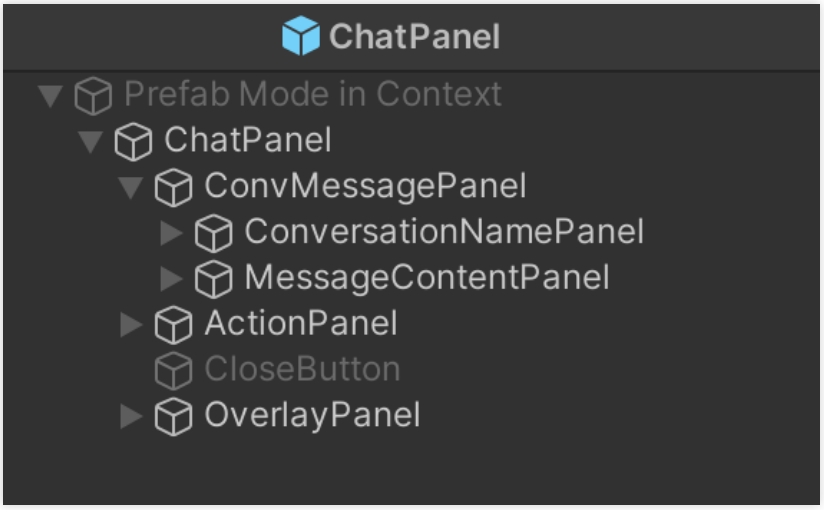
会话名展示区 `ConversationNamePanel`

历史消息展示区 `MessageContentPanel`

消息输入区 `ActionPanel`

表情包区 `OverlayPanel`

关闭聊天窗口按钮 `CloseButton`



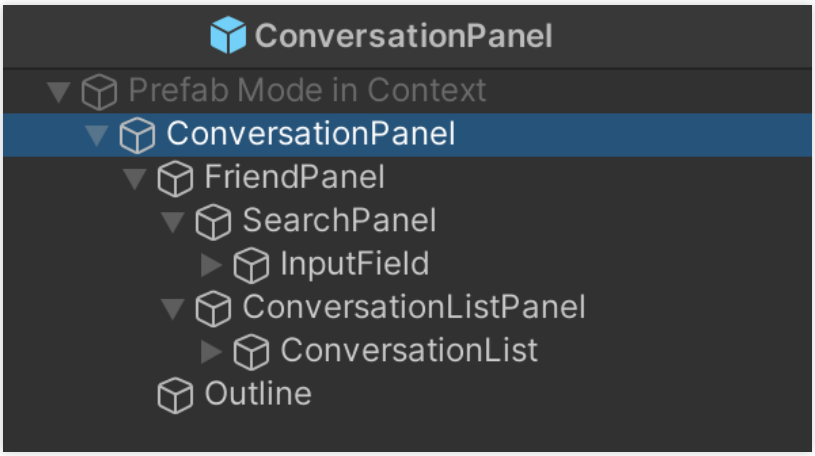
ConversationPanel

会话列表。现主要显示好友的单聊会话。相应代码在 `Script/Components/Conversation.cs` 里。每个会话的样式在 `ConversationItem.prefabs` 里。

会话列表区 `FriendPanel`

搜索区 `SearchPanel`

会话列表 `ConversationListPanel`



ChannelPanel

频道列表，由4个频道按钮组成，分别为 `世界` , `频道` , `组队` , `好友` 。其中前三个频道为群聊频道，好友频道为单聊频道并会显示单聊会话列表。频道按钮的点击事件和样式在 `Script/Components/Chat.cs` 里。

AvatarPanel

会话（ConversationItem）、单条聊天记录（messageItem等）里的头像样式。包含头像和段位头像。

ConversationItem

会话列表的会话样式，包含头像（AvatarPanel），会话名称以及段位。

MessageItem、MessageItemSelf

文字消息内容。分别为他人发送文字消息和自己发送文字消息。

头像区 `MessageSenderPanel`

消息区 `MessageContentPanel`

发送者信息区 `SenderNamePanel`

发送者名字 `MessageSender`

发送者段位Icon和名称 `Icon` 和 `Text`

消息体 `Panel`

StickerMessageItem,StickerMessageSelf。

表情包消息内容。内容与 MessageItem 相同。

GroupTipItem

群提醒消息内容，为用户进群、退群、admin消息等。包含群名和消息体。

TimeStamp

历史消息中的时间节点。

StickerItem,MenuItem

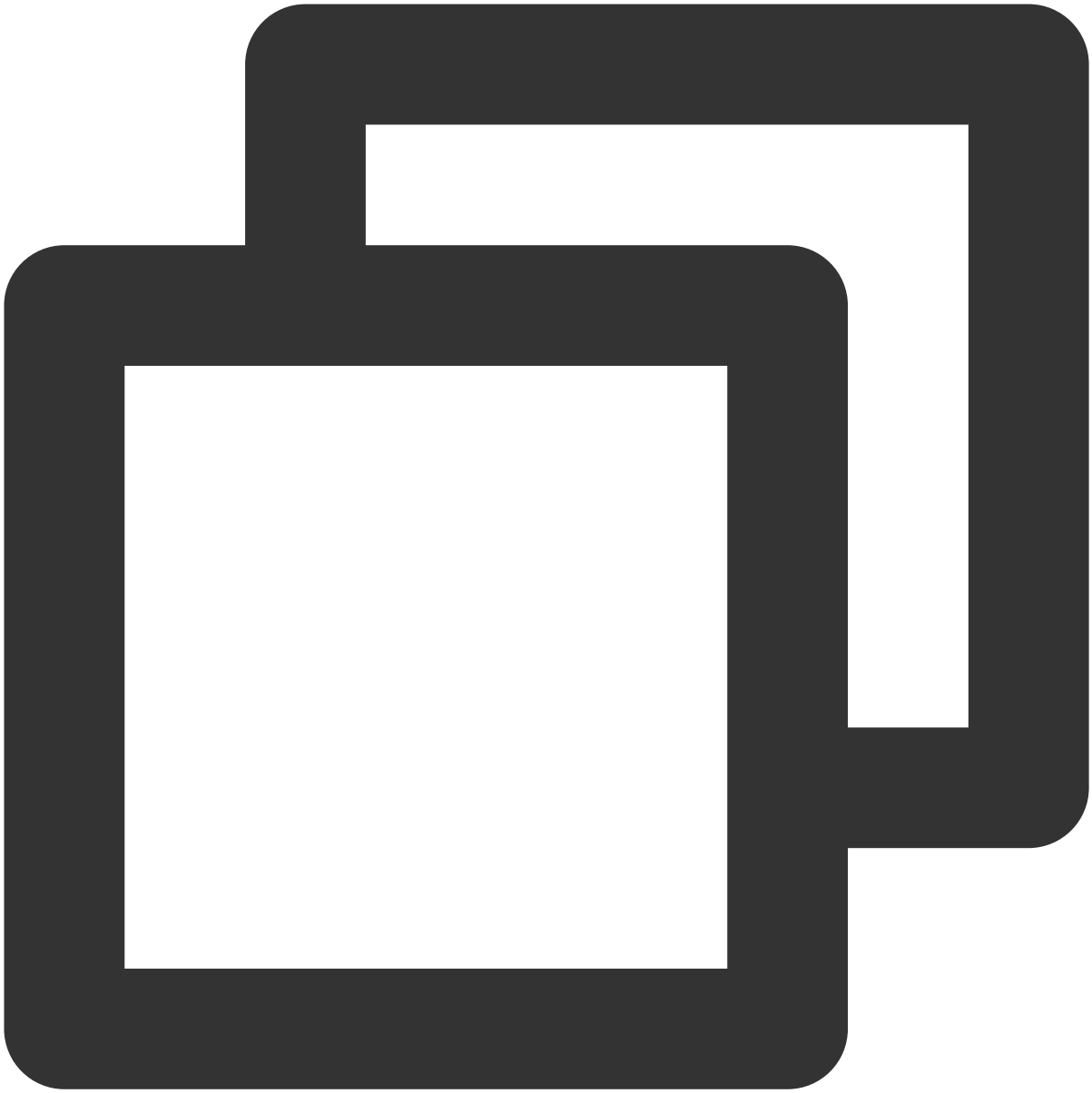
分别为表情包和快捷menu里的表情包。

如何启动demo项目

初始化登录

将SDKAppID、UserSig、UserID通过配置的方式传入组件内部进行 IM 的初始化和登录。

注意：整个项目只需要初始化一次



```
using com.tencent.imsdk.unity.uikit;
// demo 登录逻辑如下
public static void Init() {
  Core.SetConfig(sdkappid, userId, sdkUserSig);
  Core.Init();
  Core.Login();
  // 可传递函数
  // Core.Login(HandleAfterLogin);
}

// 若直接跑unity demo，推荐通过 Main页面的手机号/验证码输入框登录。可以保证监听回调等正常运行。
```

```
// 由于电脑端unity不支持手机号登录，请将 userid输入到手机号输入框，userSig输入到验证码输入框。
// Assets/Example/Scripts/Main.cs
private void Login()
{
  // 将之前的Login函数内容注释掉，添加以下内容
  loginButton.interactable = false;
  Core.SetConfig(Config.sdkappid, phoneNumber.text, captcha.text);
  Core.Init();
  Core.Login(HandleAfterLogin);
  loginButton.interactable = true;
 }
```

初始化登录后直接打开 Chat 页面即可。

频道

demo中分 `世界` 、 `频道` 、 `组队` 、 `好友` 四个频道。其中 `好友` 频道显示 C2C 会话和已填加的好友的列表，点击某个会话可开始聊天。
其他三个频道为群组会话，若需要在该频道发消息则需要先创建群组并将其 ID 添加到项目中。

创建群组

通过RestAPI添加

您可以通过后台 RestAPI中 `create_group` 创建群组。具体可见 链接。

在控制台添加

您也可以通过控制台创建群组。进入控制台中您的 IM 应用 -> 群组管理 -> 添加群组。

将群组添加到频道

进入 `Assets/Example/Scripts/Config/Config.cs` , 将创建的群组的群组ID填入 `communityID(社群)` , `channelID(频道)` , `groupID(组队)` 。
并在登录之后调用 `joinGroup` 即可实现登录后进入相应群组，并可以在群内发送消息。

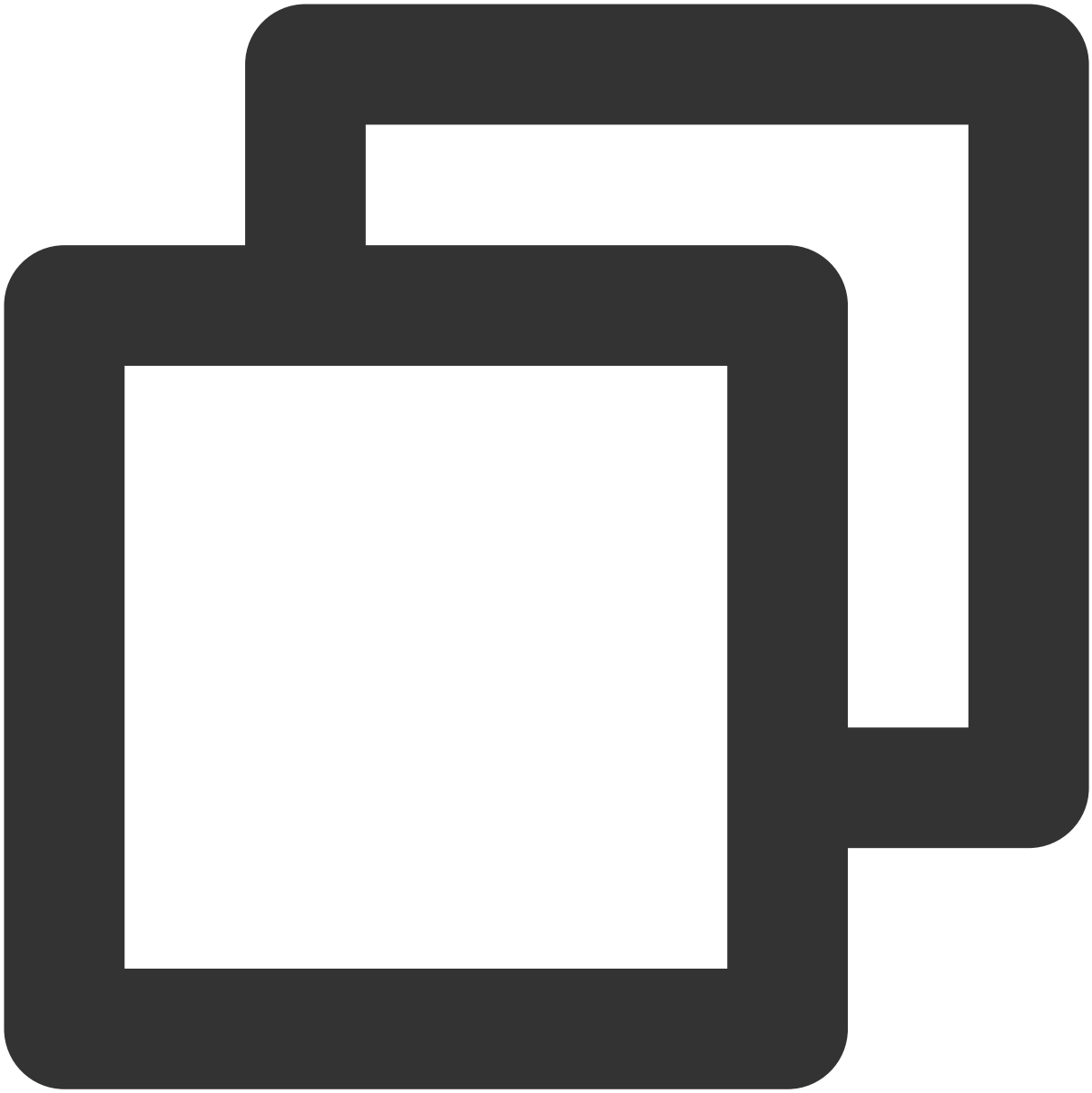
发送消息

若您有添加群组到频道中，您可以通过世界、频道、组队频道发送群聊消息。
您也可以在好友频道点击某个单聊会话发送单聊消息。

修改表情包和段位信息

段位

现各个用户段位为随机生成，若您需要使用段位信息，您可以在用户的自定义字段设置。



```
UserProfileCustemStringInfo teer = new UserProfileCustemStringInfo{
    user_profile_custom_string_info_key:"段位",
    user_profile_custom_string_info_value:"teer"
}
List<UserProfileCustemStringInfo> customArray = new List<UserProfileCustemStringInf
customArray.Add(teer);
TencentIMSDK.ProfileModifySelfUserProfile(new UserProfileItem{
    user_profile_item_custom_string_array:customArray;
});
```

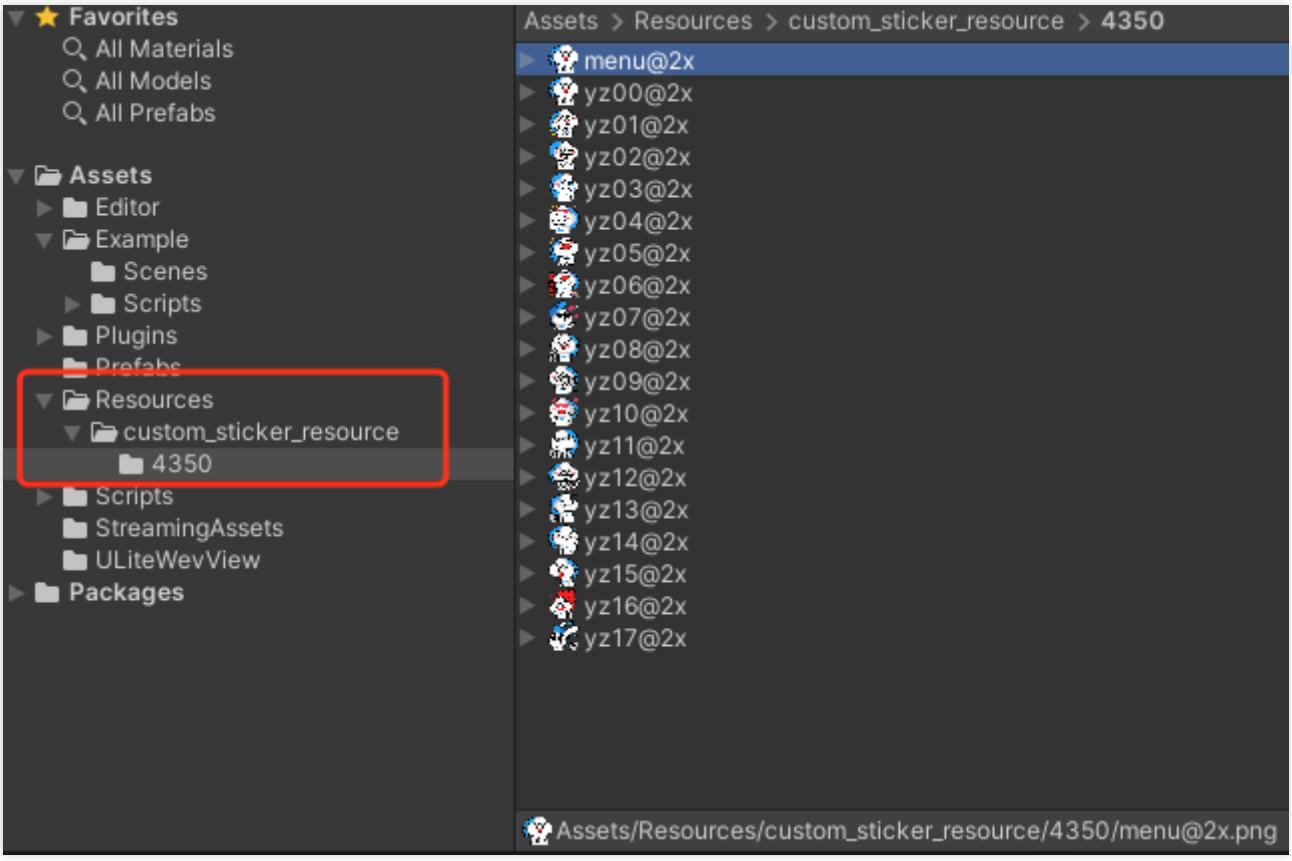
并按照段位的名称显示相应的段位图标。

1. 将段位对应的图标或者头像框加载到Resources里。（若使用Url获取时可忽略这一步）
2. 修改代码中头像框和图标的显示。需要修改的部分为会话列表和消息列表。
3. 会话列表。
4. 在获取会话的函数 `completeConvList` 中补充获取到的段位信息。最终显示的好友会话信息在 `friendProfiles` 列表中。
5. 在 `Conversation.cs` 中的 `GenerateList(会话列表渲染)` 中修改渲染的图标和头像。
6. 消息列表。
7. 在 `Chat.cs` 的 `RenderMessageForScroll` 中获取消息发送者的信息中的段位信息（若需要修改其他显示内容，也可以从这里获取）
8. 在 `MsgItem.cs` 中修改显示的样式等细节内容。

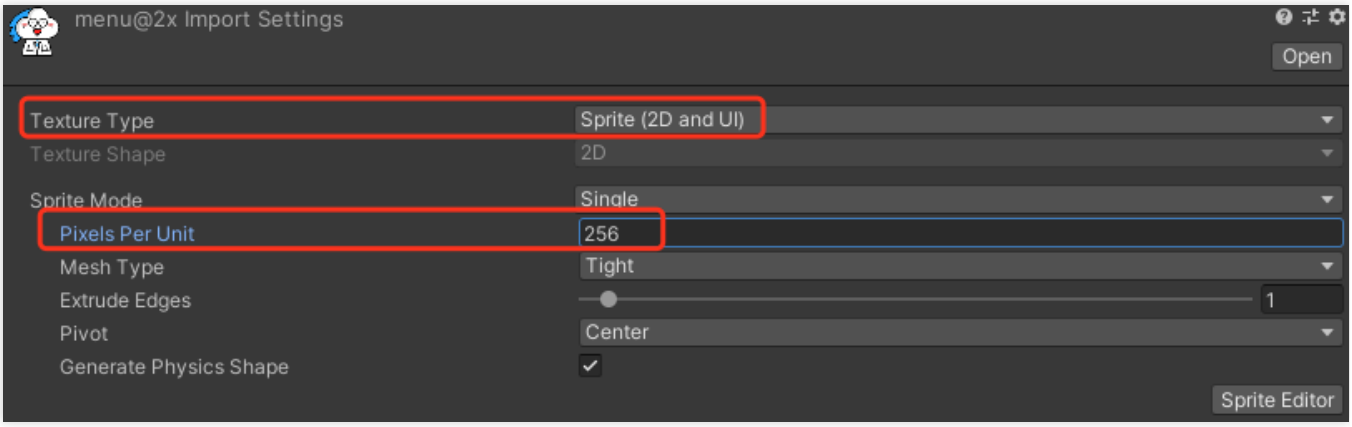
表情包

表情包使用 `StickerPanel` 显示在 `Chat.cs` 里的 `OverlayPanel` 中。您可以导入自己的表情包使用。（需要您提前导入自己的表情包）

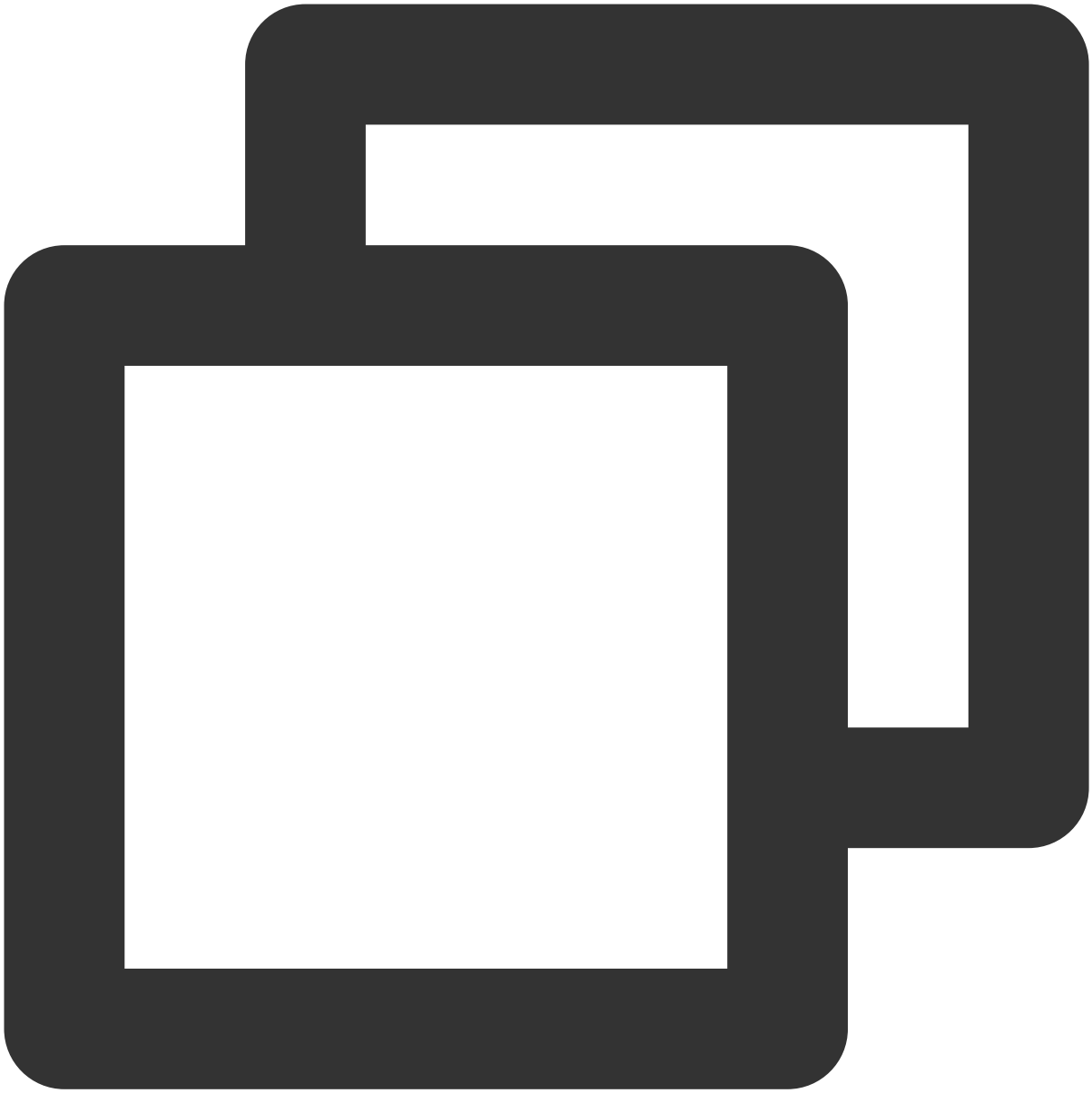
1. 在 `Assets/Resources` 文件夹内导入所用的表情包图片：



2. 更改图片的 `Texture Type` 为 `Sprite (2D and UI)`，并根据图片尺寸修改 `Pixels Per Unit` ：



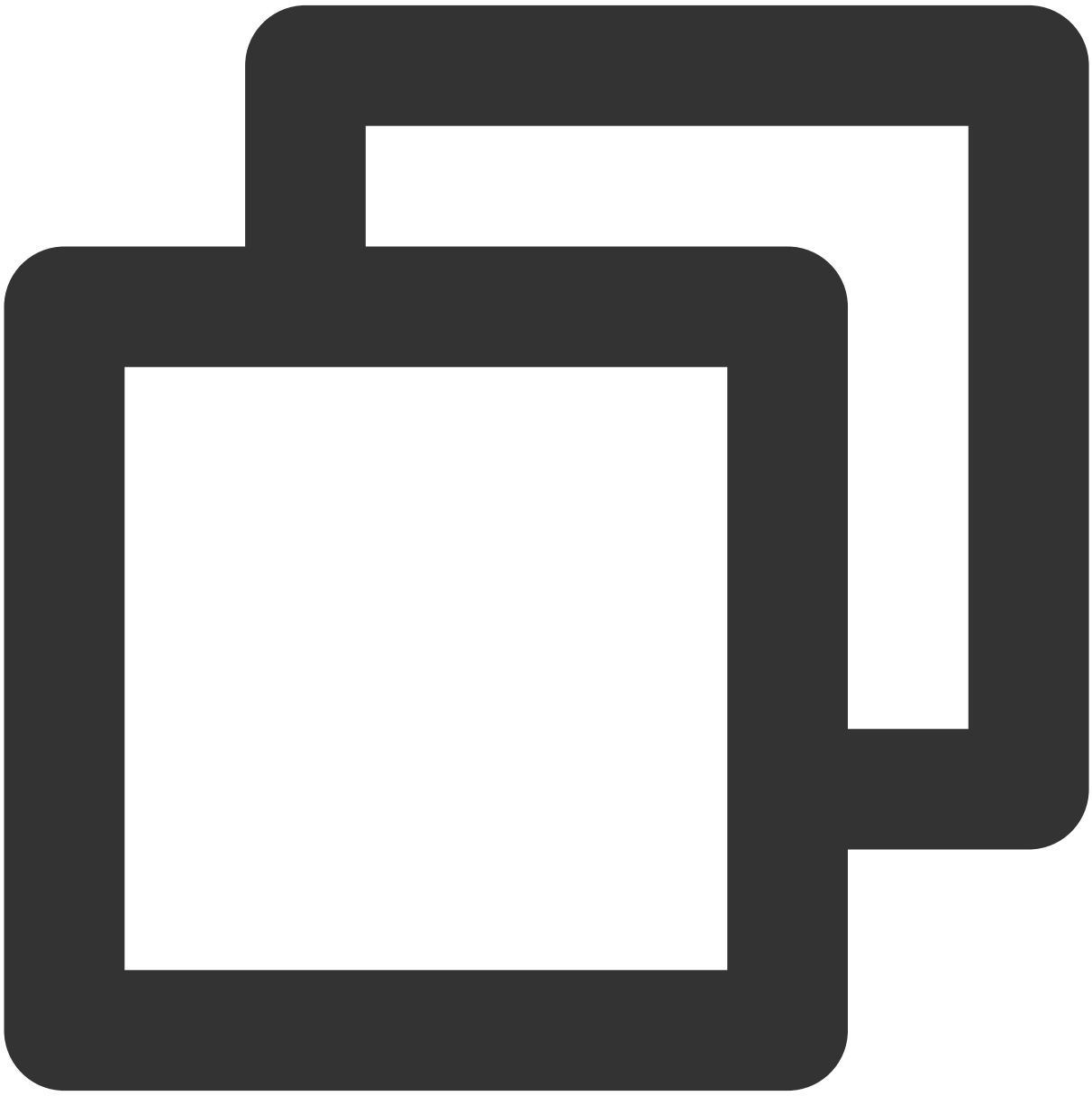
3. 定义相应的表情包数据：



```
// 生成表情包列表，StickerPackage 为一组表情包
List<StickerPackage> stickers = new List<StickerPackage> {
new StickerPackage {
  name = "4350",
  baseUrl = "custom_sticker_resource/4350", //Resource 文件夹内相对路径
  menuItem = new StickerItem { // 表情栏表情项目
    name = "menu@2x",
    index = 0,
  },
  stickerList = new List<StickerItem> { // 表情包项目组
    new StickerItem { // 具体表情包数据
```

```
        name = "menu@2x",
        index = 0 // 表情包顺序
      },
      }
    }
  };
```

4. 注册表情包给 UIKit：



```
using com.tencent.imsdk.unity.uikit;

    Core.SetStickerPackageList(Config.stickers);
```

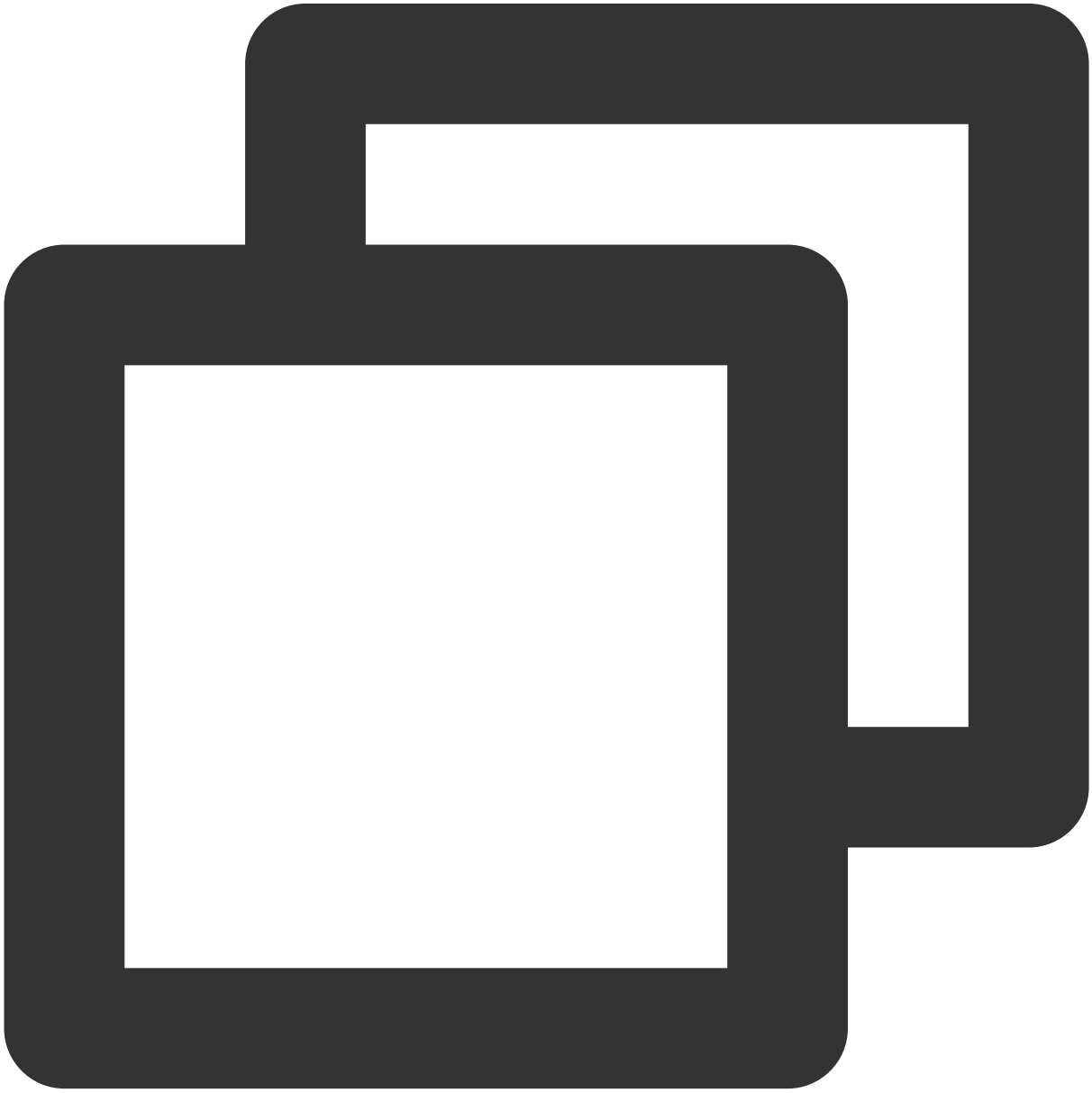
语言包

IM Unity UIKit Demo提供根据系统语言切换语言系统，现支持简体中文和英语。您可以按照需求增加语言或者修改里面的配置。

1. 语言资料

语言资料放在 `Resources/LanguageTxt` 里。现在包含 `Chinese.txt(简体中文)` 和 `English.txt(英文)` 。若需要其他语言，可以添加对应的txt文件。

该文件的结构如下：



```
//English.txt
Key:Value

//Chinese.txt
Key:值
```

注意：

Key应与其他语言的Key一致，并与后续的enum一致

Value为Key对应的该语言的值

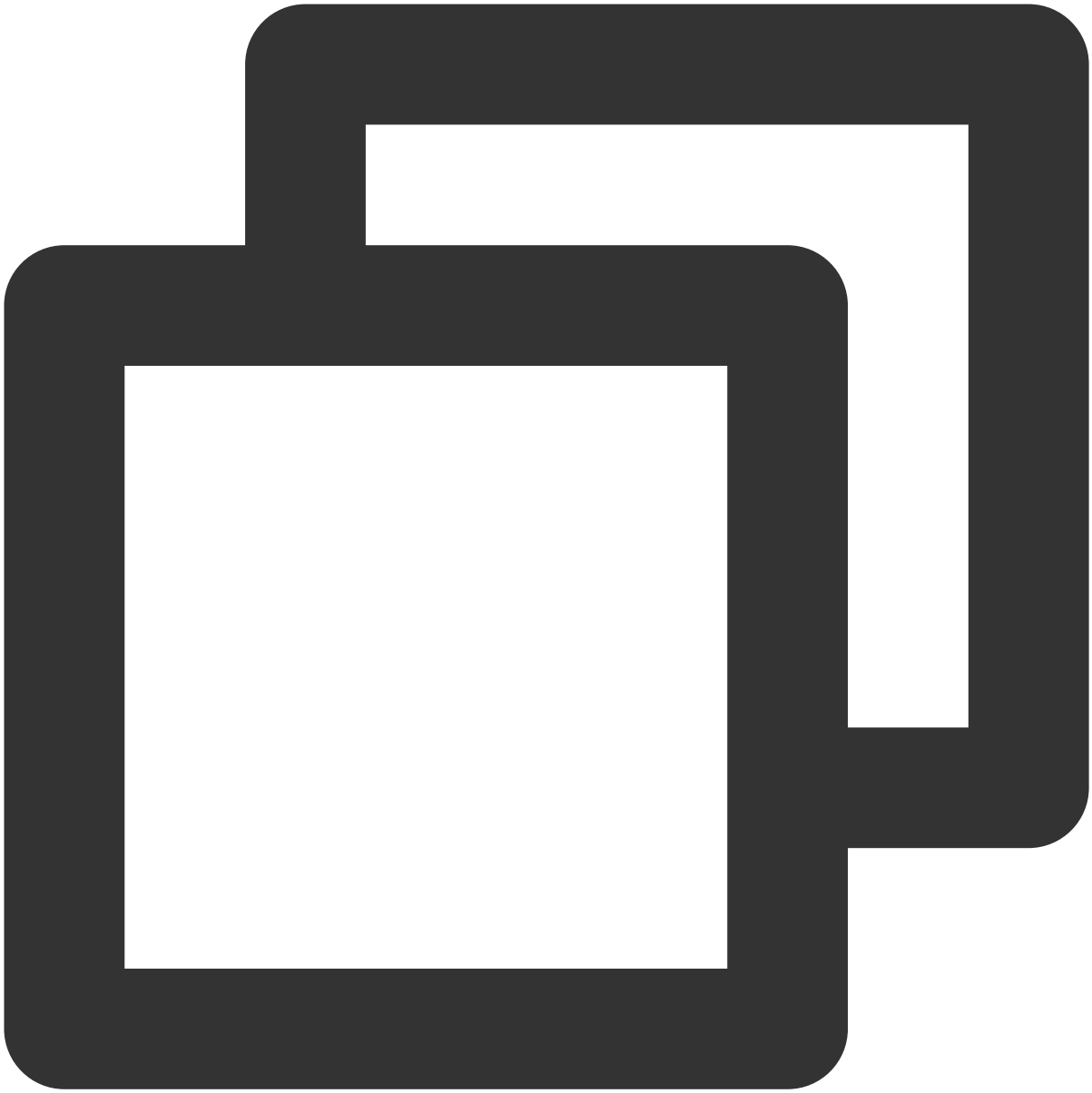
Key 和 Value之间使用冒号分隔开

2. 设置语言

3. 设置语言和词条

若你添加了语言，添加相应的语言词汇txt文件后在 `LanguageDataManager.cs` 中的 `Language` 中添加新的语言，并在 `LanguageTextName` 中增加对应的词条的Key。

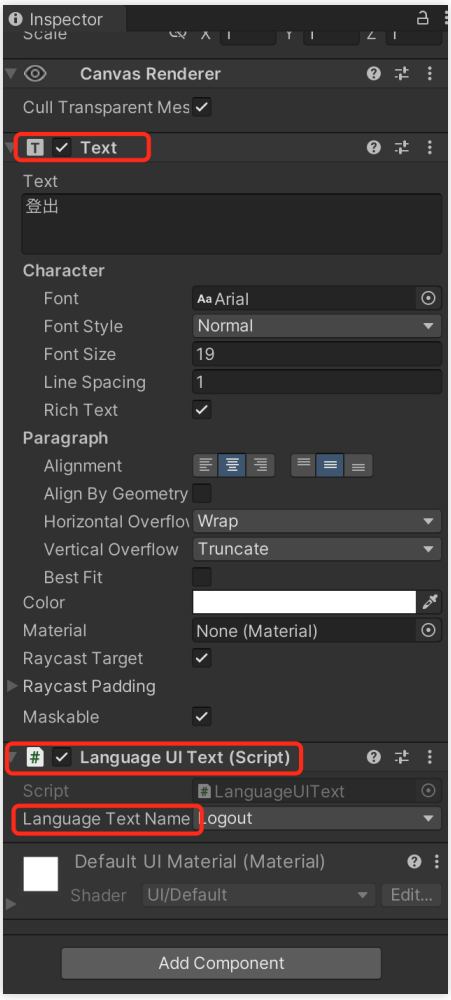
4. 加载语言词条文件



```
private Dictionary<string,string> EnglishDictionary = new Dictionary<string,string>
LoadLanguageTxt(Language.English);
```

5. 组件设置（静态修改）

在需要设置的text组件中添加 `LanguageUIText(Script)` component，将需要显示的词的Key选中。改显示的Key对应LanguageTextName中的enum和词汇文件里的Key。



6. 设置语言

若要设置语言，在软件开启时调用 `SetCurrentLanguageValue` 。若要固定语言，可直接在 `LanguageDataManager.cs` 对 `currentLanguage` 赋值（可当成默认语言）。该Demo根据系统语言判断并赋值。

若需要修改的组件不仅为静态组件，则简单的方法为将现在使用的语言保存到config中（在Demo中保存到了Core）在代码中判断显示。

API 文档

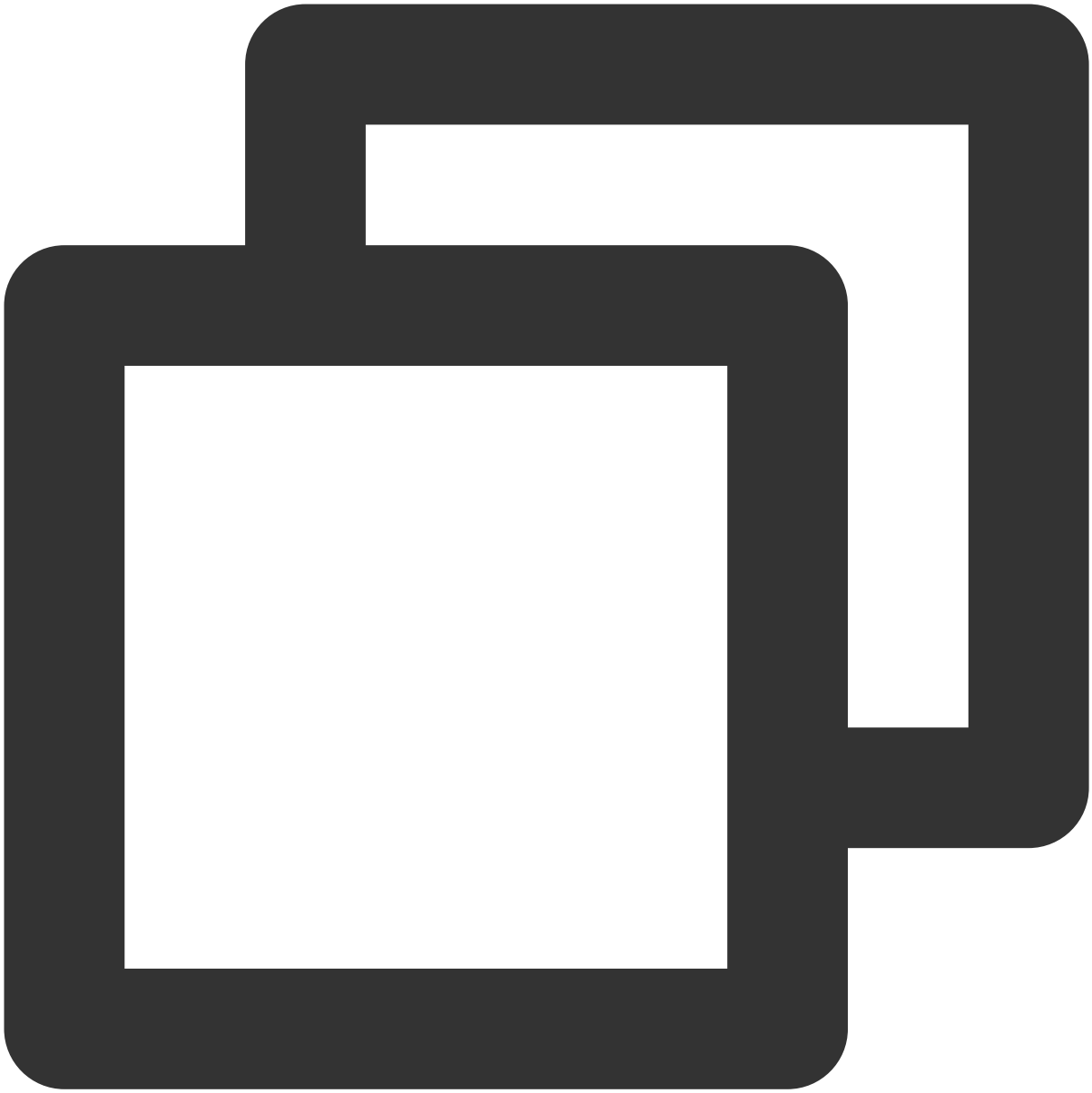
Tencent Cloud IM Chat SDK 文档链接

Tencent Cloud IM Chat SDK 官网链接

Tencent Cloud IM Chat SDK 快速入门

SetConfig

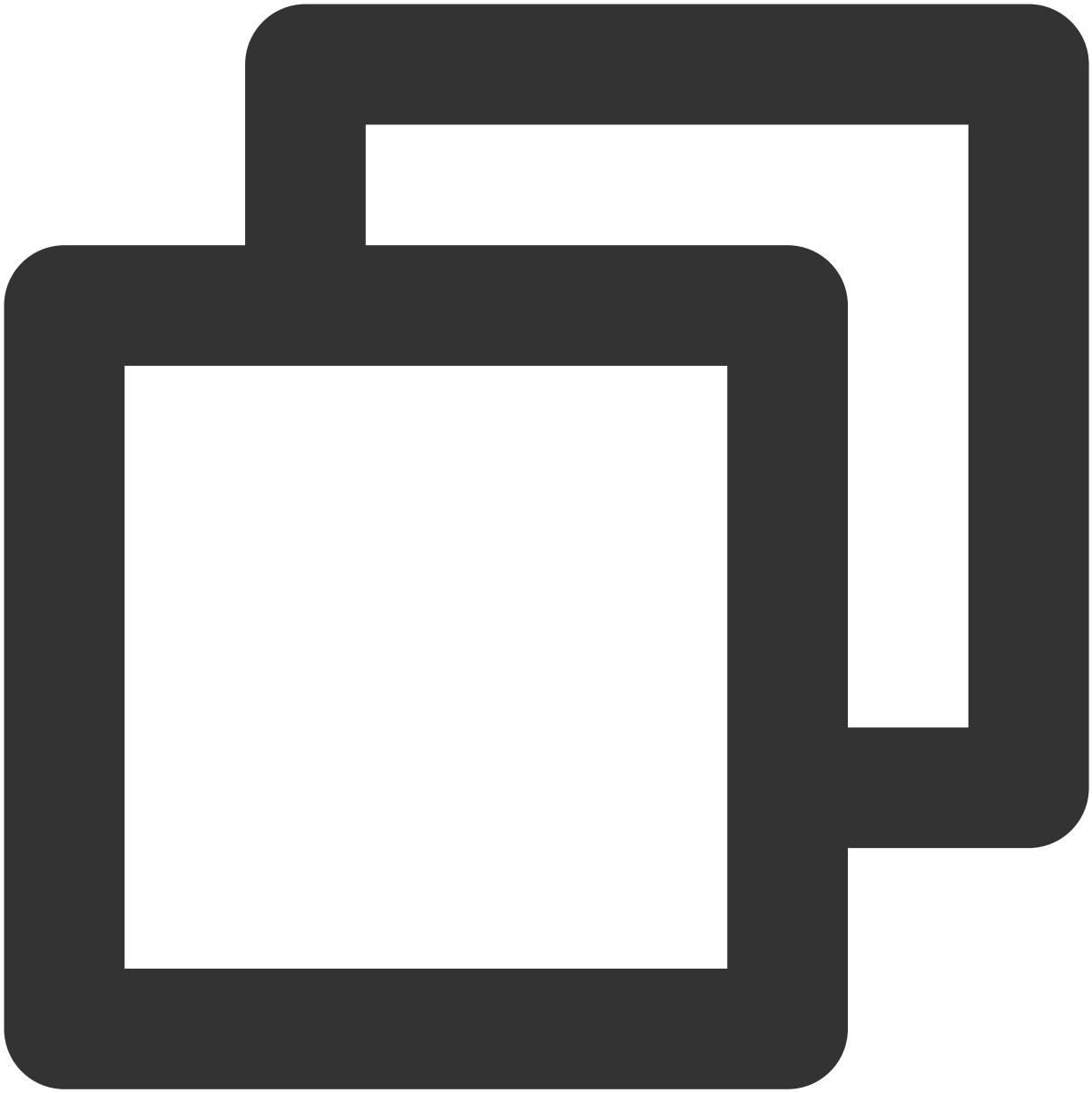
在 Init 前传入 Config 信息，包括 `sdkappid` ， `userid` 以及 `usersig` 。



```
using com.tencent.imsdk.unity.uikit;

   Core.SetConfig(sdkappid, userid, usersig);
```

Init

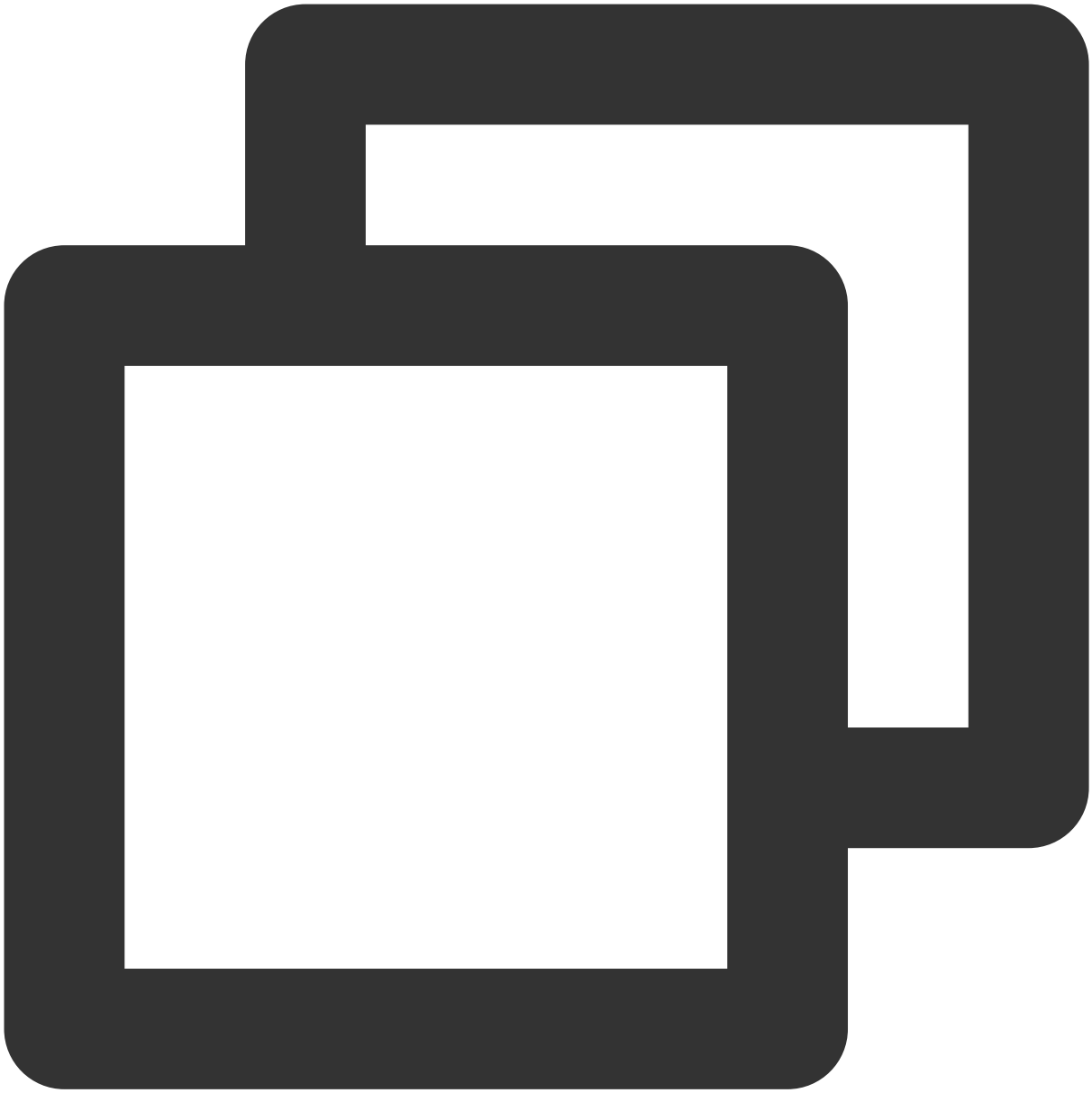
采用 UIKit 提供的 Init 方法来初始化 SDK，会自动绑定 `AddRecvNewMsgCallback` 和 `SetConvEventCallback` 回调。



```
using com.tencent.imsdk.unity.uikit;

    Core.Init();
```

SetStickerPackageList

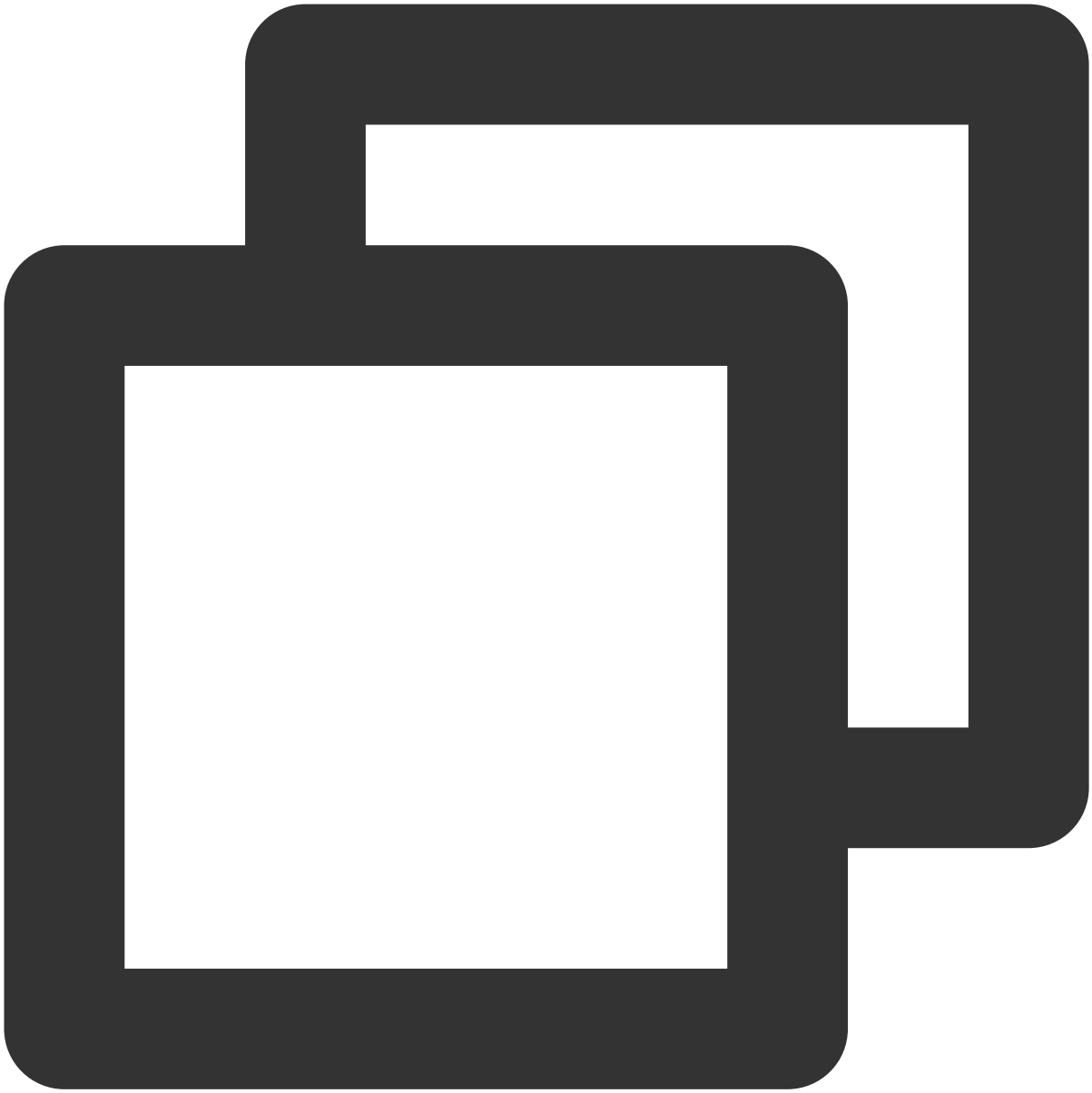
通过 `SetStickerPackageList` 设定表情包列表。



```
using com.tencent.imsdk.unity.uikit;

   Core.SetStickerPackageList(Config.stickers);
```

Login

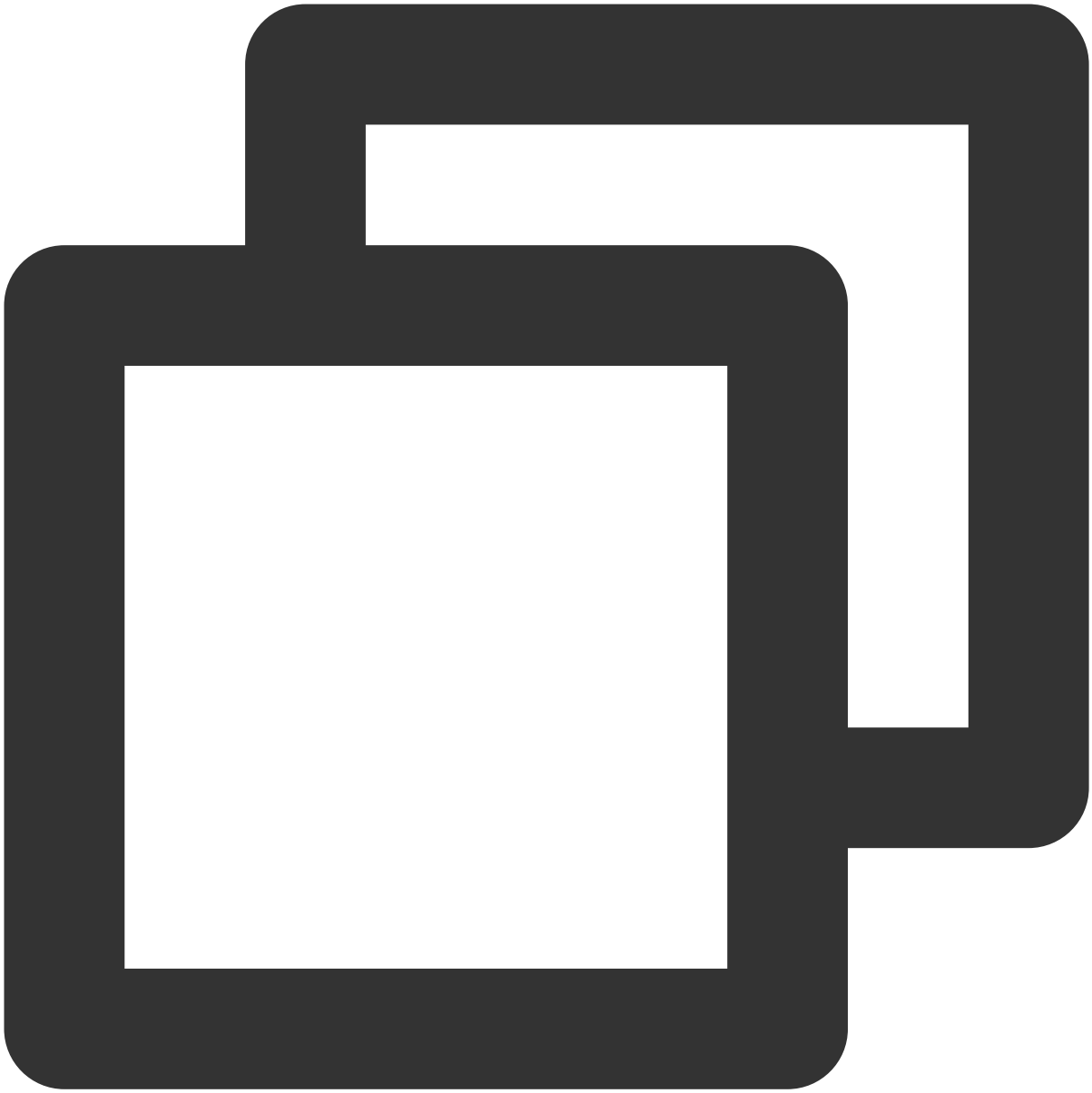
通过 `Login` 登录账号，登录完成后执行绑定的回调函数。



```
using com.tencent.imsdk.unity.uikit;

    Core.Login((params string[] args) => {
      // 处理Login回调
    });
```

SetMessageList

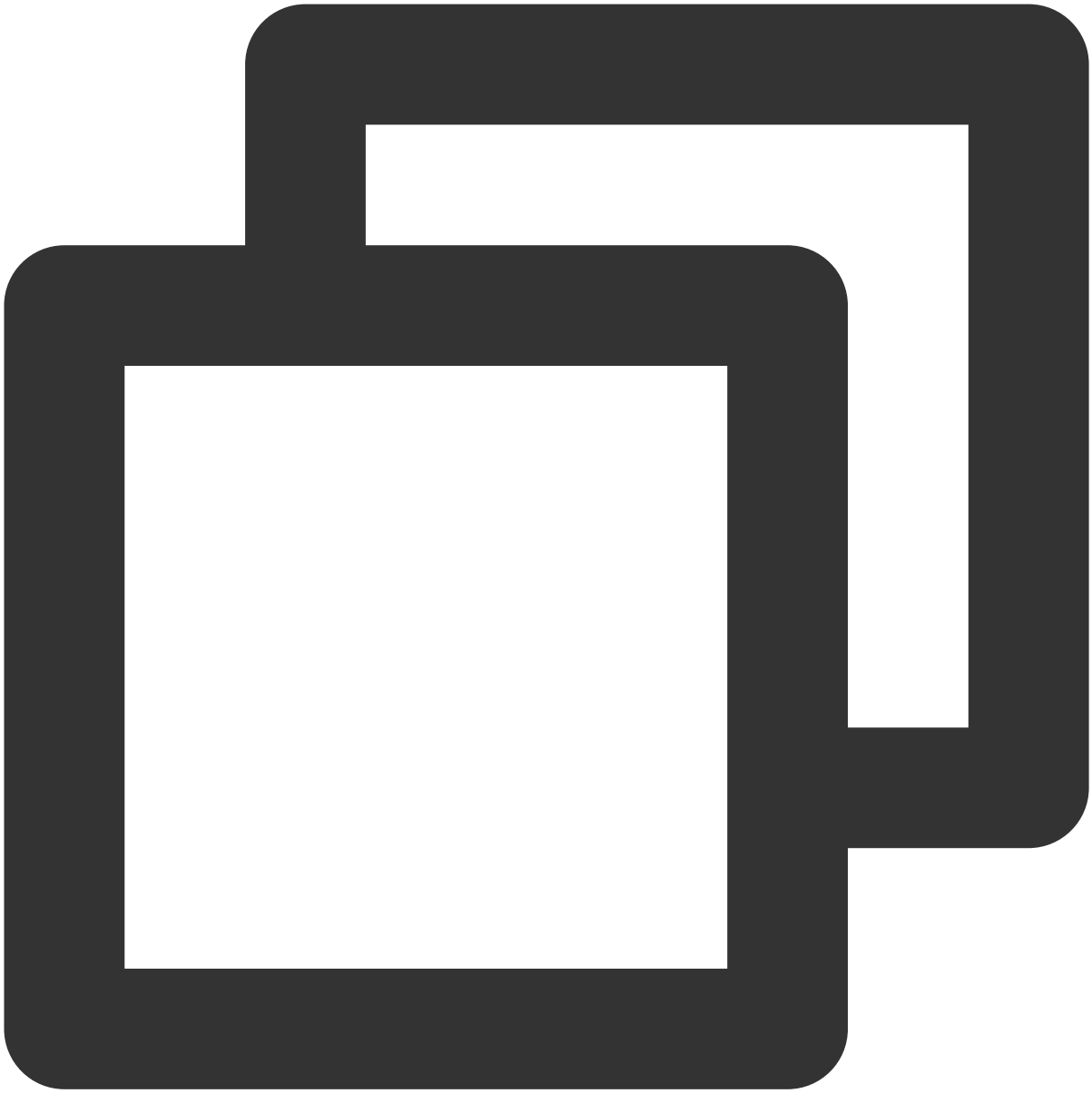
添加某个会话的消息列表，处理后合并到当前会话消息字典里，并触发 `OnMsgListChanged` 事件。



```
using com.tencent.imsdk.unity.uikit;

   Core.SetMessageList(currentConvID, newMsgList, isFinished);
```

SetCurrentConv

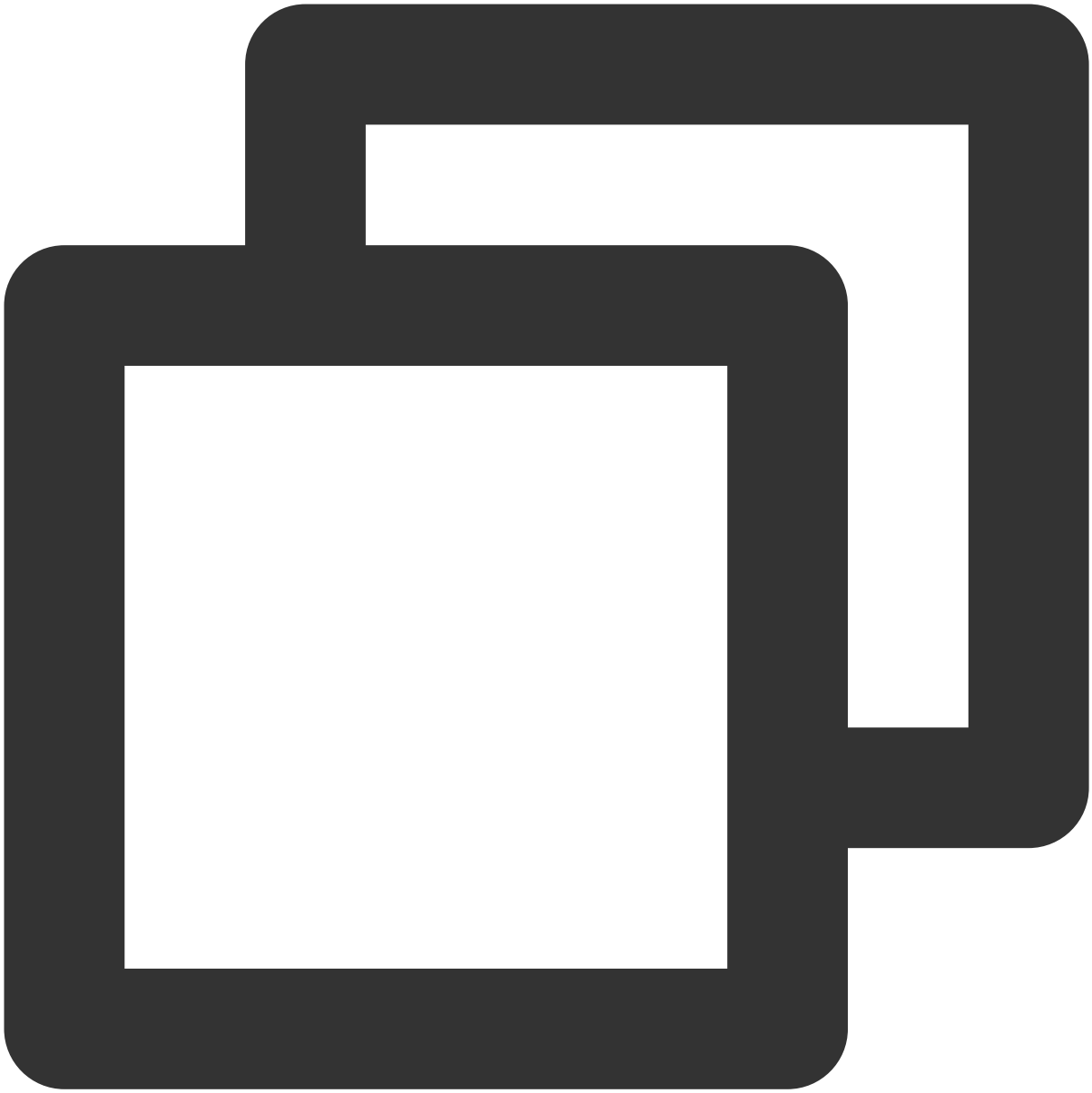
设置当前选中的会话，并触发 `OnCurrentConvChanged` 事件。



```
using com.tencent.imsdk.unity.uikit;

   Core.SetMessageList(convID, convType);
```

SetCurrentStickerIndex

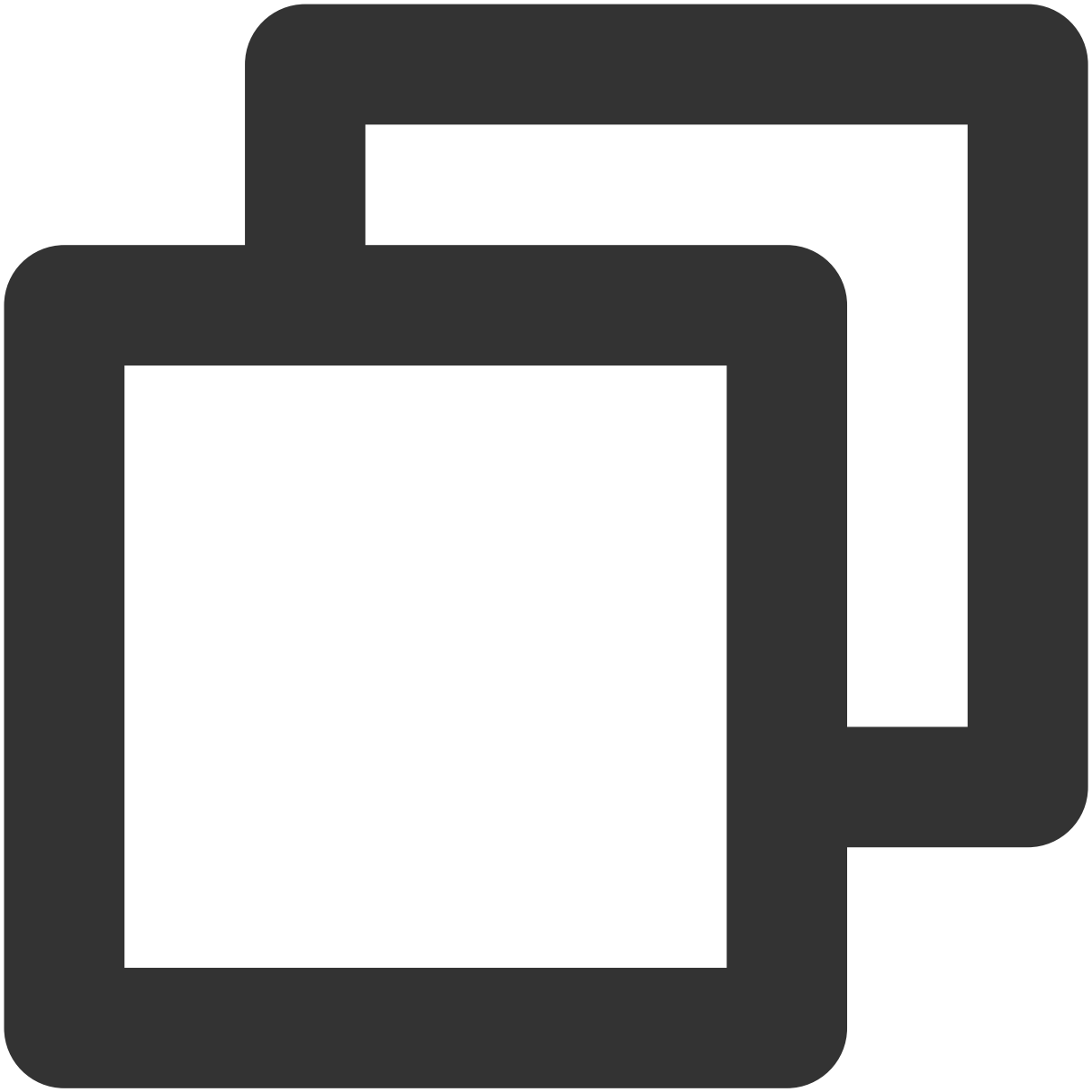
设置当前选中的表情包组，并触发 `OnCurrentStickerIndexChanged` 事件。



```
using com.tencent.imsdk.unity.uikit;

   Core.SetMessageList(stickerIndex);
```

Logout

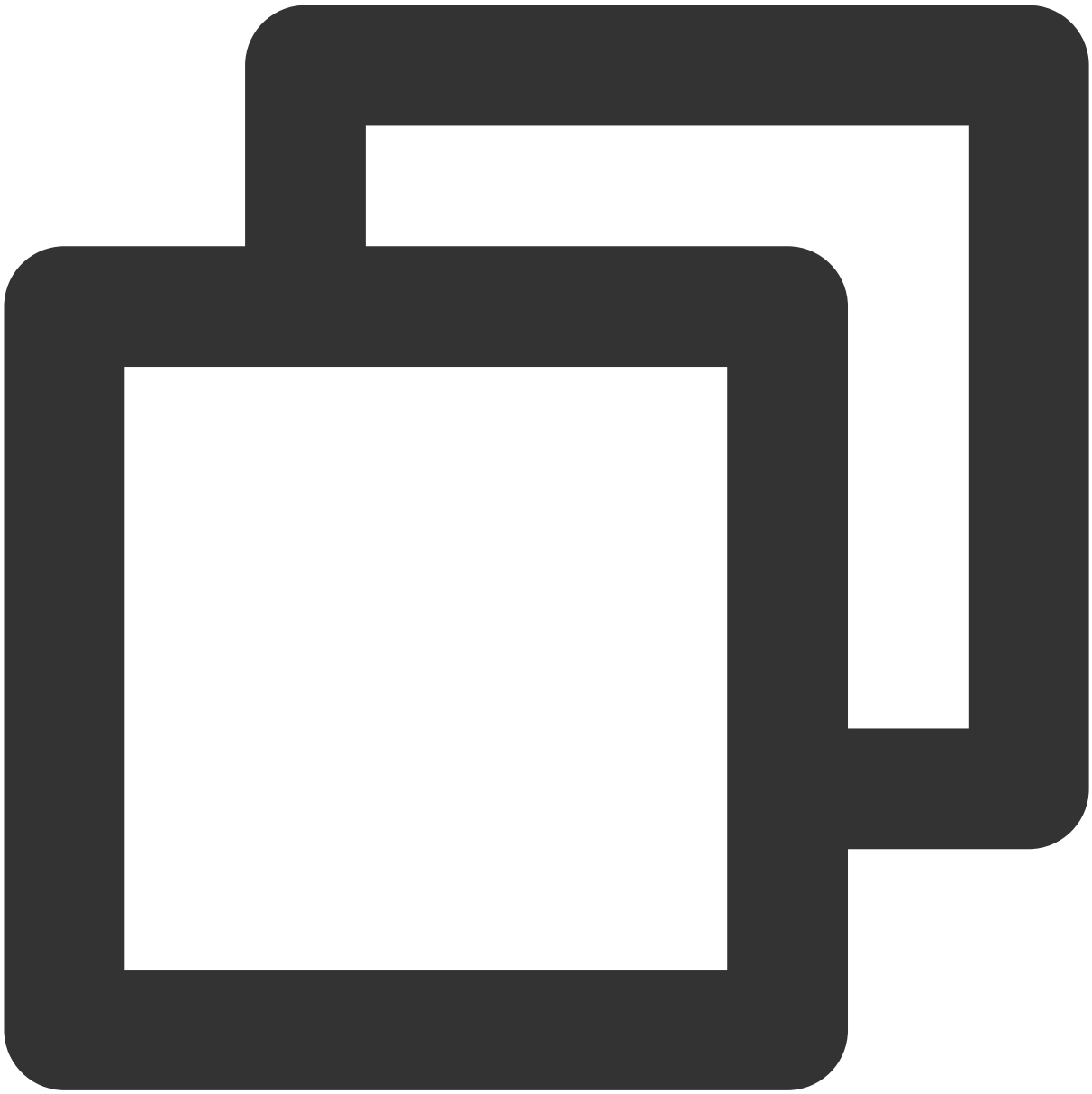
登出，并清空数据。



```
using com.tencent.imsdk.unity.uikit;

    Core.Logout((string[] parameters) => {
      // 处理Logout回调
    });
```

TencentIMSDK

Unity TencentIMSDK 提供了基于 Unity 平台的全面的即时通信能力。您可以使用 `TencentIMSDK` 来获取其他即时通信的相关功能。例如通过 `TencentIMSDK` 来获取用户资料。



```
using com.tencent.imsdk.unity;

    // 获取个人资料
    FriendShipGetProfileListParam param = new FriendShipGetProfileListParam
    {
      friendship_getprofilelist_param_identifier_array = new List<string>
      {
        "self_userid"
```

```
        }
    };

    TIMResult res = TencentIMSDK.ProfileGetUserProfileList(param, (int code, string
        // 处理异步逻辑
    });
```

交流与反馈

点此进入IM社群，享有专业工程师的支持，解决您的难题。

ReactNative

最近更新时间：2023-08-22 10:49:21

开发环境要求

TypeScript
Node（ 推荐使用 Node.js 官方 LTS 版本 16.17.0）
npm（版本请与 node 版本匹配）

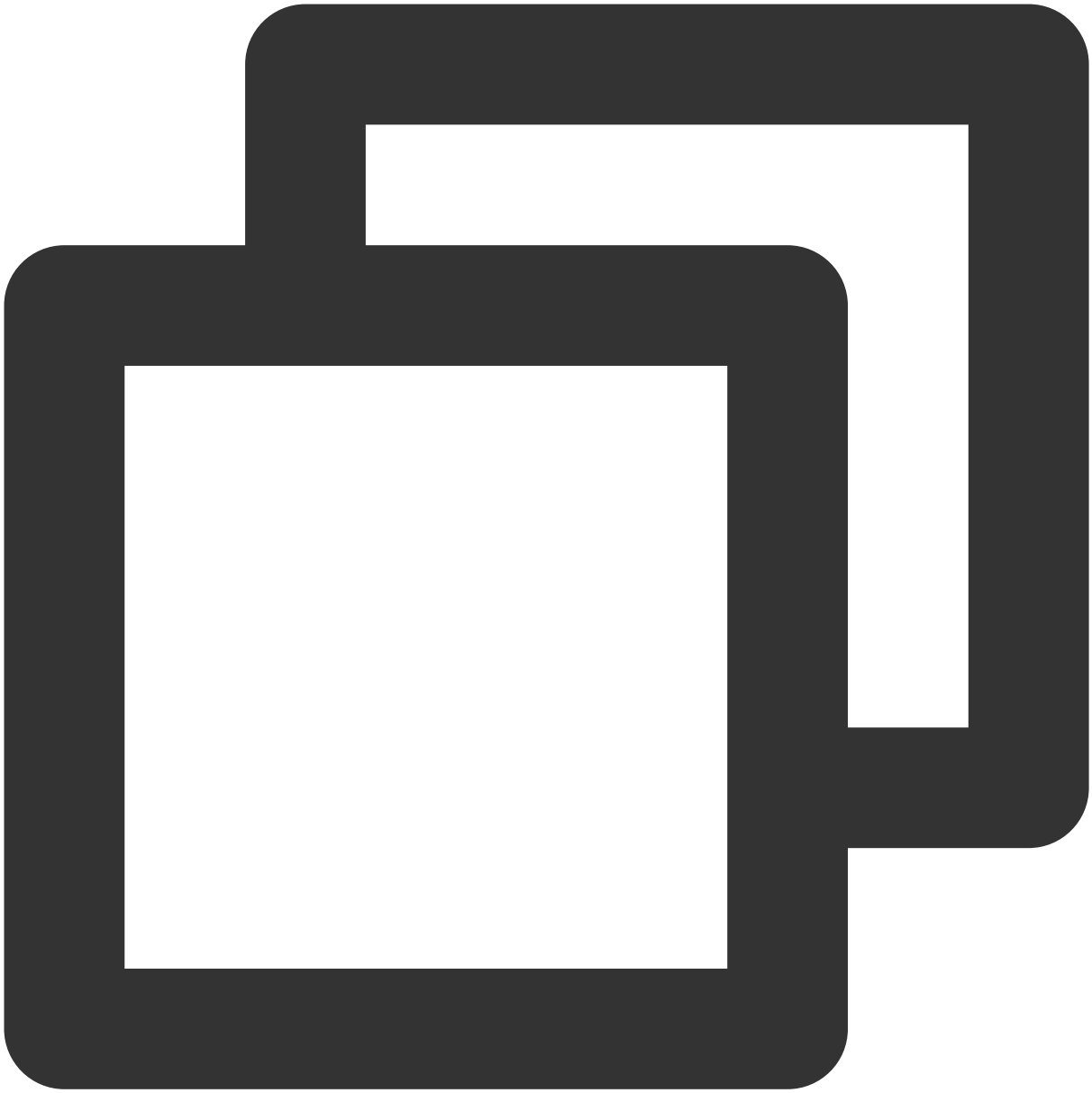
步骤1：环境搭建

请参见 React Naitve 官方文档搭建本地开发环境。

步骤2：下载并引入 TUIKit

通过 git 克隆 方式下载 TUIKit 组件。
本项目为 React Native Chat Demo，它是由 `react-native-tim-js` 及社区相关的开源包开发的 Demo 项目, 可帮助您快速开发一个即时通信聊天场景应用。

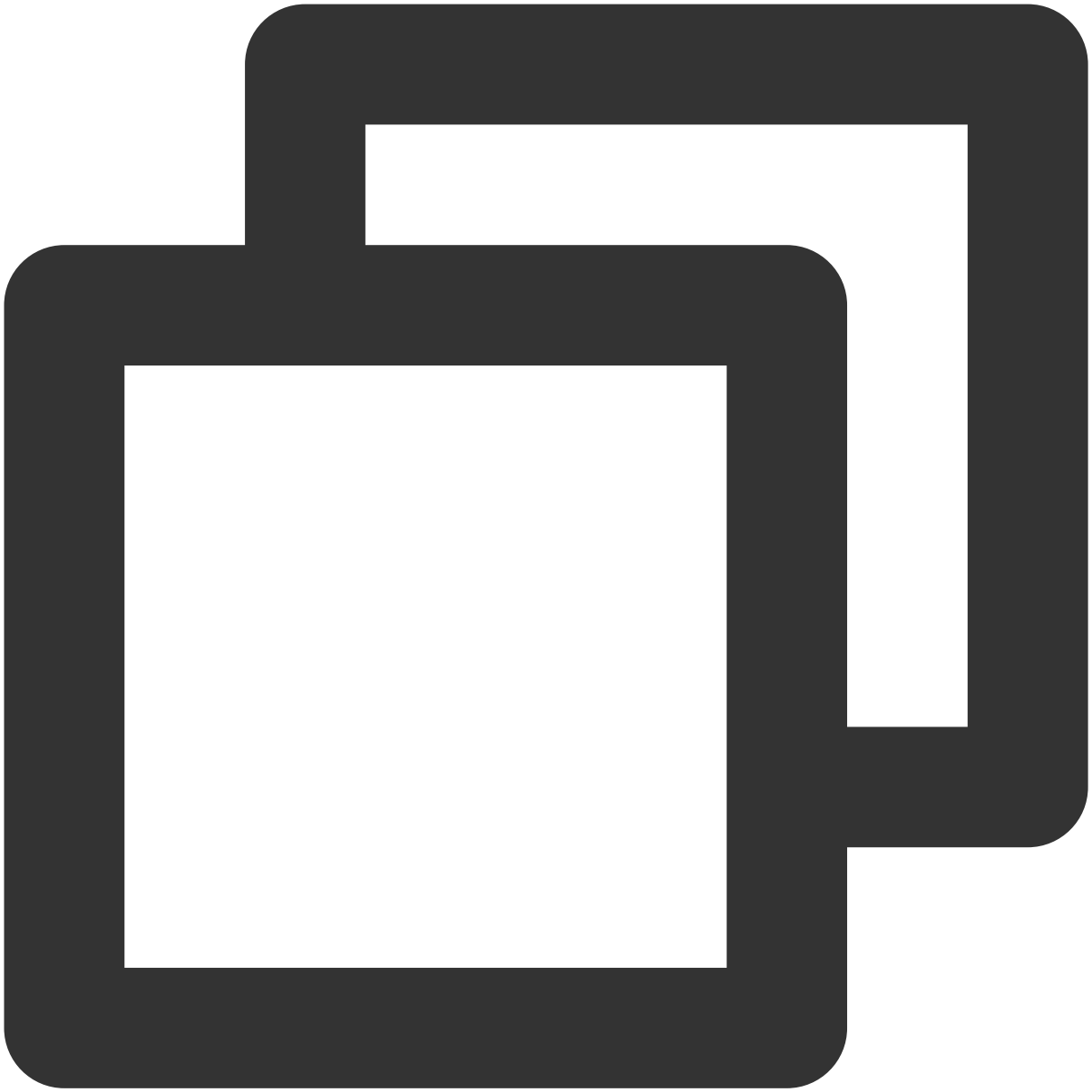
版权所有：腾讯云计算（北京）有限责任公司



```
git clone https://github.com/TencentCloud/chat-demo-react-native.git
```

步骤3：依赖安装

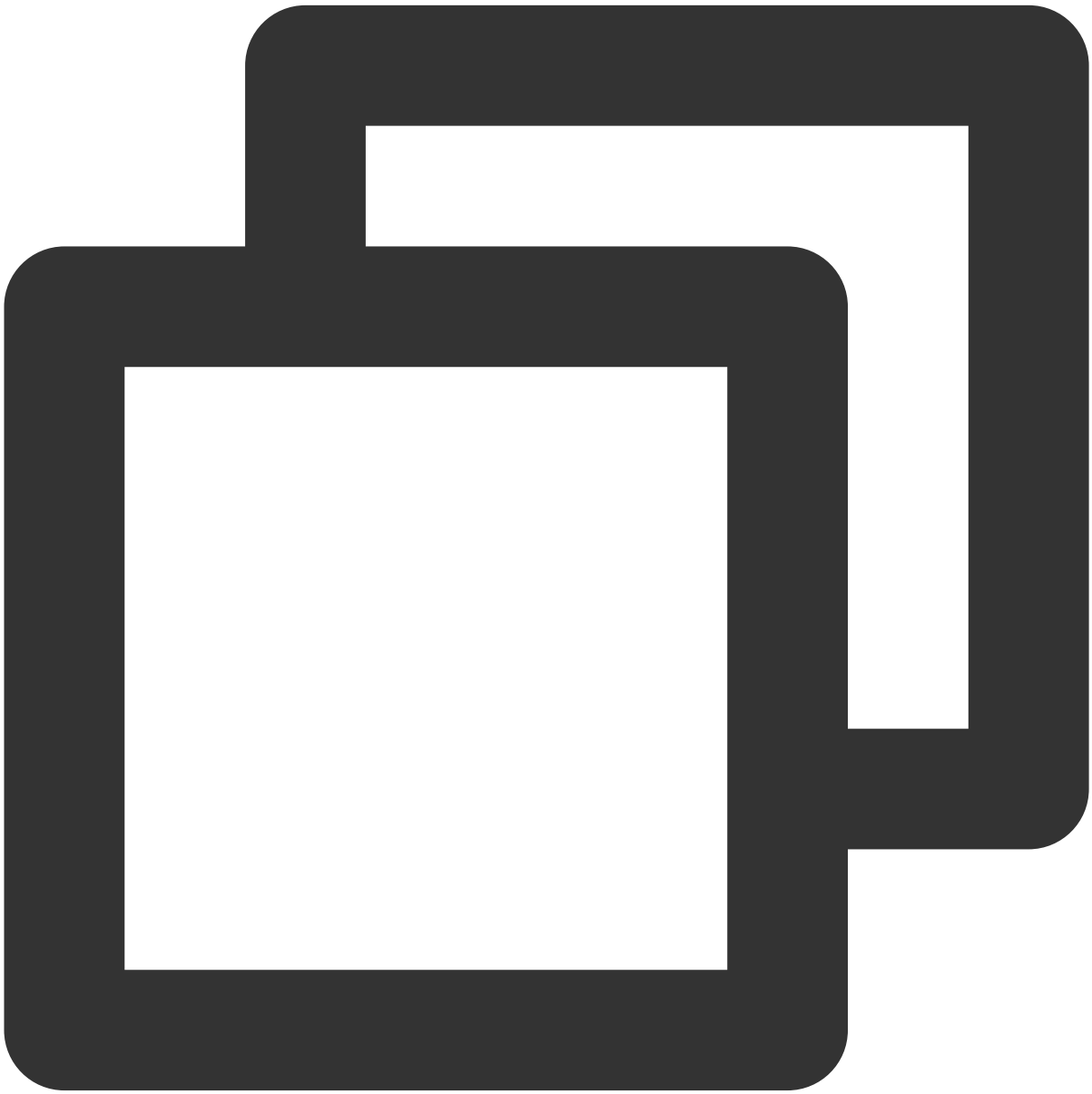
在项目的根目录下执行如下代码，安装项目所需依赖项。



```
// yarn
yarn
// npm
npm install
// ios
cd ios
pod install
```

步骤4：demo 运行

1. 请参见 官方文档 准备 `SDKAppID` 和 `UserSig`，您必须拥有正确的 SDKAppID，才能进行初始化。
2. 将准备好的 `SDKAppID` 和 `UserSig` 填入到 `src/pages/config.ts` 文件中。
3. 执行如下命令运行：



```
// yarn
yarn android
yarn ios
// npm
```

```
npm run android
npm run ios
```

常见问题

如何移植到现有项目中?

`src/TUIKit` 中为 TUIChat 等相关组件，可以直接复制到您的项目中，同时您还需要安装对应的依赖即可，对应的依赖查看 `package.json` 文件。

在 expo 项目中如何使用?

在 `expo` 中，如果您使用到的 package 包含 Native 代码，需要您使用 `development build`，具体信息可参见 官方文档。

如何在 demo 中安装Expo modules

https://docs.expo.dev/bare/installing-expo-modules/#usage

Android 项目报错 Task :react-native-create-thumbnail:compileDebugJavaWithJavac FAILED 怎么解决?

请参见 GitHub 相关问题。

项目报错 No toolchains found in the NDK toolchains folder for ABI with prefix: arm-linux-androideabi 怎么解决?

https://blog.csdn.net/python_yjys/article/details/127145470

项目报错 Xxx is not Fabric compatible yet 怎么解决?

找到提示的组件并换为原生组件即可。

项目报错 Undefined symbols for architecture x86_64 怎么解决?

https://stackoverflow.com/questions/71933392/react-native-ios-undefined-symbols-for-architecture-x86-64

项目报错 Execution failed for task ':react-native-gesture-handler:buildCMakeDebug[arm64-v8a]' 怎么解决?

https://github.com/software-mansion/react-native-gesture-handler/issues/2427

项目报错 This declaration is experimental and its usage must be marked with '@kotlin.ExperimentalStdlibApi' or '@OptIn(kotlin.ExperimentalStdlibApi::class)' 怎么解决?

https://stackoverflow.com/questions/74401011/kotlin-experimentalstdlibapi-or-optinkotlin-experimentalstdlibapiclass

项目报错 could not find module 'ExpoModulesCore' for target 'x86_64-apple-ios-simulator'; 怎么解决?

https://blog.csdn.net/boildoctor/article/details/112575041

出现应用闪退, 什么问题?

请检查权限是否申请。

单击拍照无反应？

拍照功能，请用真机调试。

交流与反馈

欢迎加入 知聊社群 进行技术交流和反馈问题。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-05-15 11:27:18

说明：

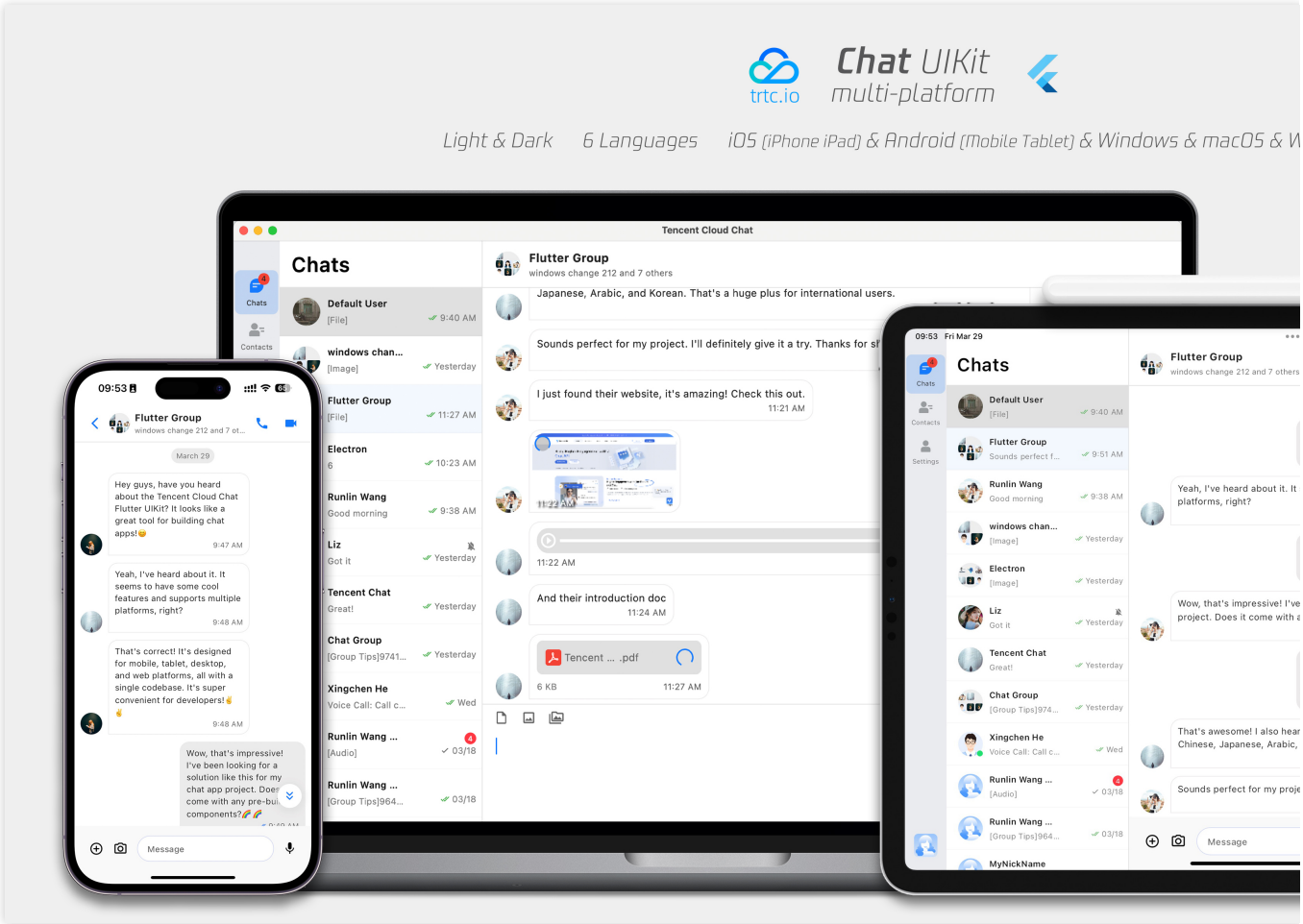
这份文档主要介绍我们新版的 Flutter UIKit（tencent_cloud_chat）。如果您需要浏览旧版 UIKit 的文档，请点击这里。

欢迎使用腾讯云IM开发的全新 Flutter Chat UIKit。在我们之前发布的 `tencent_cloud_chat_uikit` 版本两年后，我们很高兴推出这个完全重新设计和开发的工具包，它从头开始构建。

我们的新 Flutter Chat UIKit 旨在为开发者提供一套全面的工具，以便轻松创建功能丰富的聊天应用程序。

它采用模块化方法构建，让您可以选择所需的组件，同时保持应用程序轻量级和高效。

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



特点

1. **个性化外观**: UIKit内置深色和浅色模式，提供多种主题和外观定制选项，以满足您的业务需求。
2. **多平台兼容性**: 适应性强的单一代码库可确保跨各种平台的兼容性，包括**移动设备**（iOS/Android）、**平板电脑**（iPad和Android平板电脑）、**Web** 浏览器和**桌面软件**（Windows/macOS）。
3. **本地化支持:** 使用本地英语和其他语言选项开发，包括阿拉伯语、日语、韩语、简体中文和繁体中文。国际化功能确保了本地化的界面语言，并支持自定义和补充语言，其中阿拉伯语支持 **RTL** 用户界面。
4. **增强的性能:** UIKit提供了改进的消息列表性能、内存使用和精确的消息定位功能，以满足具有大量消息和导航到旧消息的情况。
5. **高级功能:** UIKit拥有众多高级功能，包括连续语音消息播放、增强的多媒体和文件消息体验以及直观的左右滑动以预览多媒体消息。
6. **精致的用户体验:** 丰富的动画、触觉反馈和精美的界面等细节优化有助于改善用户体验。网格风格的头像、重新设计的转发面板、群组成员选择器和改进的长按消息菜单等新功能进一步丰富了体验。
7. **模块化设计:** 组件被组织成模块化包，允许选择性导入并减少不必要的膨胀。每个包都支持内置导航转换，通过自动处理转换（例如对话和消息之间的转换）来简化开发和集成。
8. **对开发人员友好的方法:** 更统一、标准化的组件参数设计，更清晰的代码命名约定和详细的注释，以及选择全局或实例级配置管理的灵活性，使开发更轻松、更高效。

兼容性

我们的 UIKit 支持**手机端**, **平板端** 和**桌面端** UI 样式，并兼容 Android、iOS、macOS、Windows 和 Web（将在未来版本中支持）。
它内置支持英语、简体中文、繁体中文、日语、韩语和阿拉伯语（支持阿拉伯RTL界面）以及亮色和暗色外观样式。

要求

Flutter版本：3.16或更高
Dart版本：3.0或更高

开始使用

Demo

你可以参考 Demo 源码 配合此文档为了保证顺滑成功的接入流程.

引入包

基础包

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

模块化 UI 组件包

接下来，从以下列表中导入适合您需求的所需 UI 组件包：

tencent_cloud_chat_message

tencent_cloud_chat_conversation

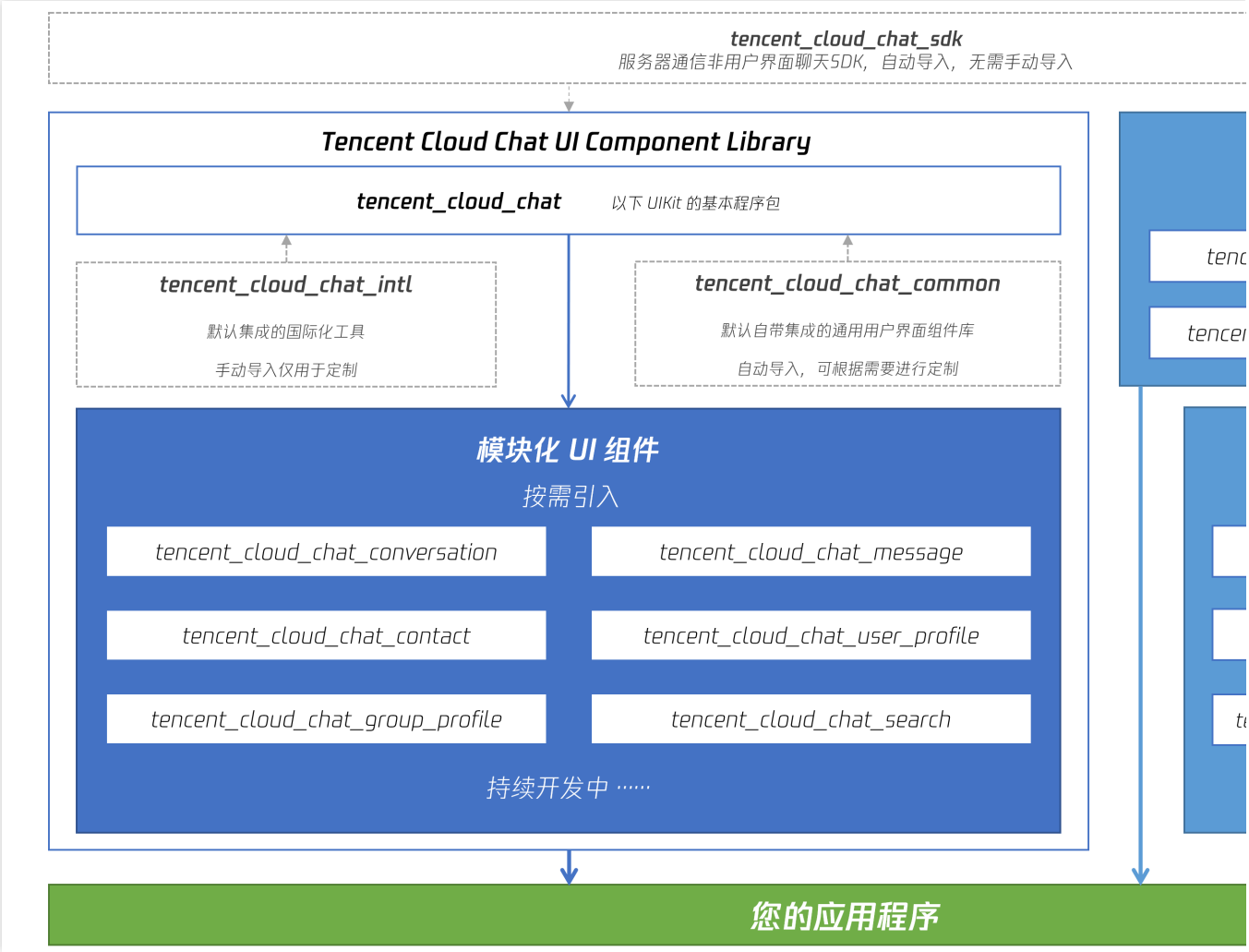
tencent_cloud_chat_contact

tencent_cloud_chat_user_profile

tencent_cloud_chat_group_profile

tencent_cloud_chat_search *（内测中）*

下面展示了我们的UIKit的架构：



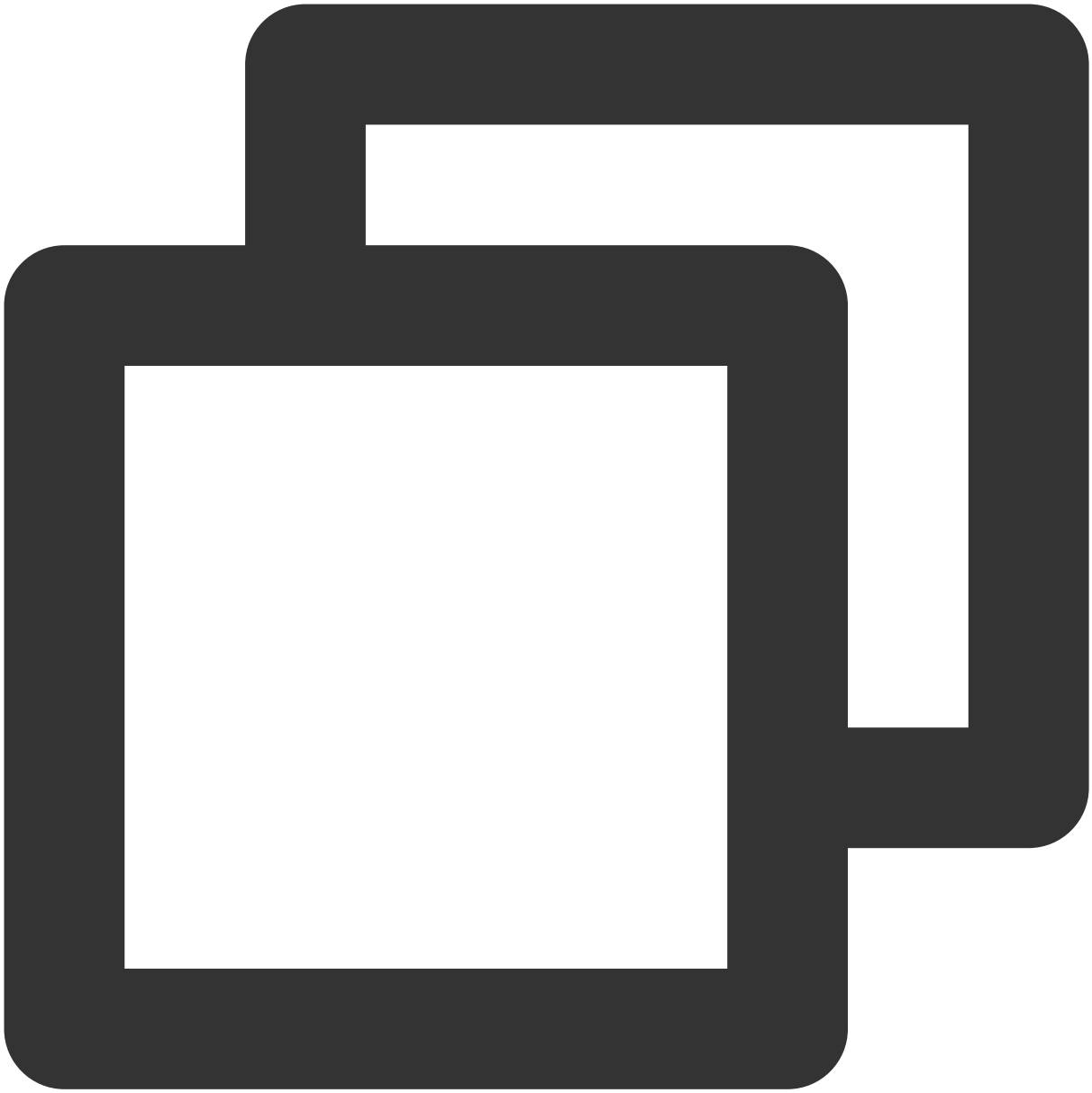
说明：

平台集成

在继续“基本用法”部分之前，请确保完成此处列出的其他平台集成步骤，特别是当您针对这些特定平台进行部署时。

Web / macOS: 如果您计划在 ***Web*** 或 ***macOS*** 平台上部署项目，请 参考此文档说明。

iOS: 打开 `ios/Podfile` ，并将最后一节替换为以下内容。



```
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['EXCLUDED_ARCHS[sdk=iphonesimulator*]'] = 'arm64'
          config.build_settings['ENABLE_BITCODE'] = 'NO'
          config.build_settings["ONLY_ACTIVE_ARCH"] = "NO"
        end
    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
```

Android / Windows: 不需要执行其他操作。

基本用法

在开始使用每个模块化包UI组件之前，您需要在项目中遵循一些初始设置步骤。

1. 准备必要的腾讯云IM 配置信息，如sdkappid、测试userID、userSig等。您可以参考此文档：

https://www.tencentcloud.com/document/product/1047/45907#.E5.89.8D.E5.BA.8F.E5.B7.A5.E4.BD.9C

2. **安装 Package:**

在您的Flutter项目中，安装主包和上面的“开始使用”部分中提到的可选模块化包。

3. **全局配置:**

导入 `TencentCloudChatMaterialApp` ：将项目的 `MaterialApp` 替换为 `TencentCloudChatMaterialApp` 。这将自动管理和配置语言、主题（*带有material3*）、主题模式和其他设置，确保UIKit的界面参数与您的项目保持一致。

这一步将接管项目的语言、主题和主题模式配置。如果您不希望我们为您的项目自动管理所有这些配置，您可以按照**以下指南**在您的项目中手动导入必要的功能。

手动实现UIKit的全局配置

我们建议将您的项目的 `MaterialApp` 替换为 `TencentCloudChatMaterialApp` 。此推荐方法自动管理全局配置，包括本地化、主题和主题模式。

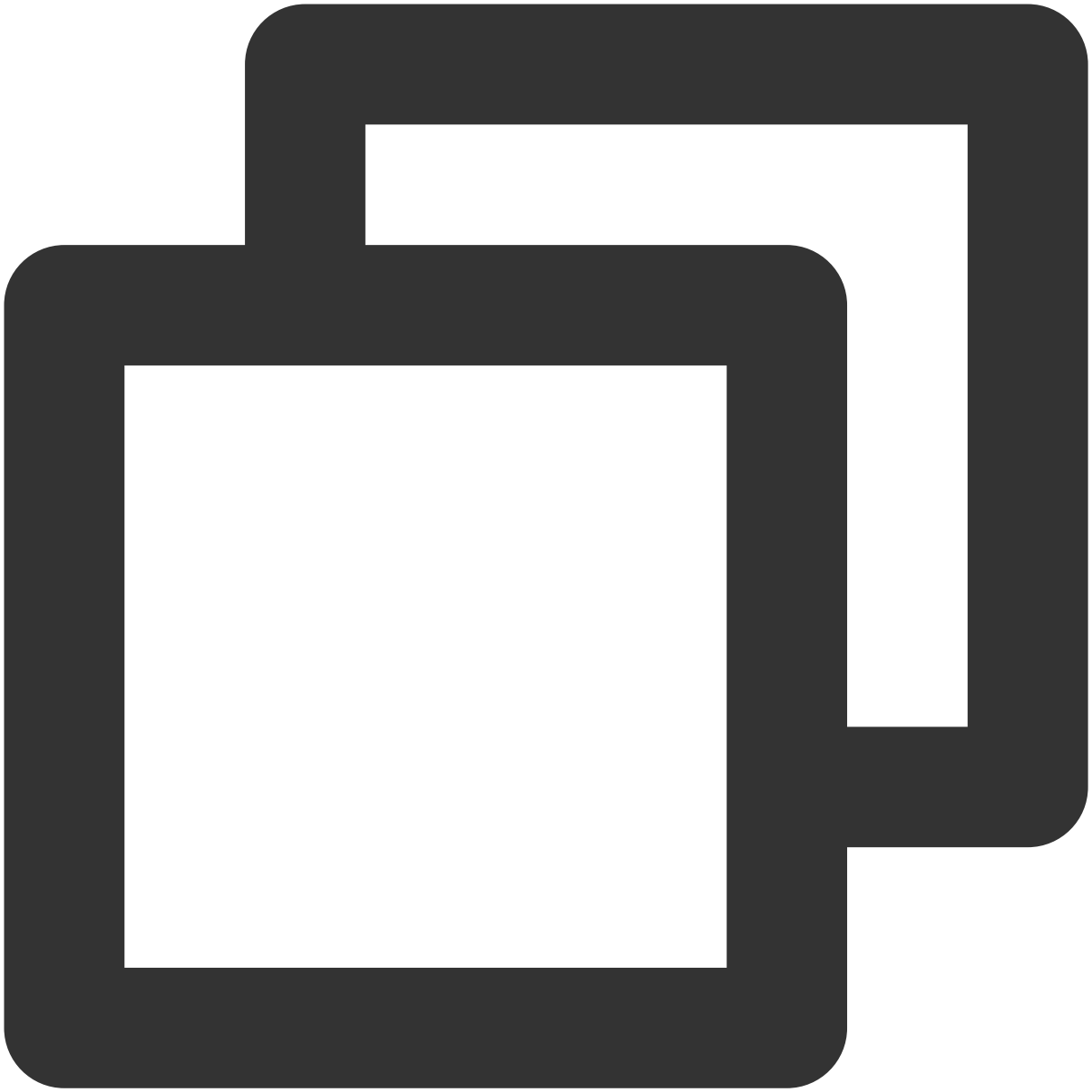
但是，如果由于大量自定义或使用其他包(如 `GET`)而希望保留项目的 `MaterialApp` ，则可以手动初始化UIKit。本指南将指导您完成该过程。

在全局配置中，本地化是必选的，而主题和主题模式设置是可选的。我们开始吧。

必要操作

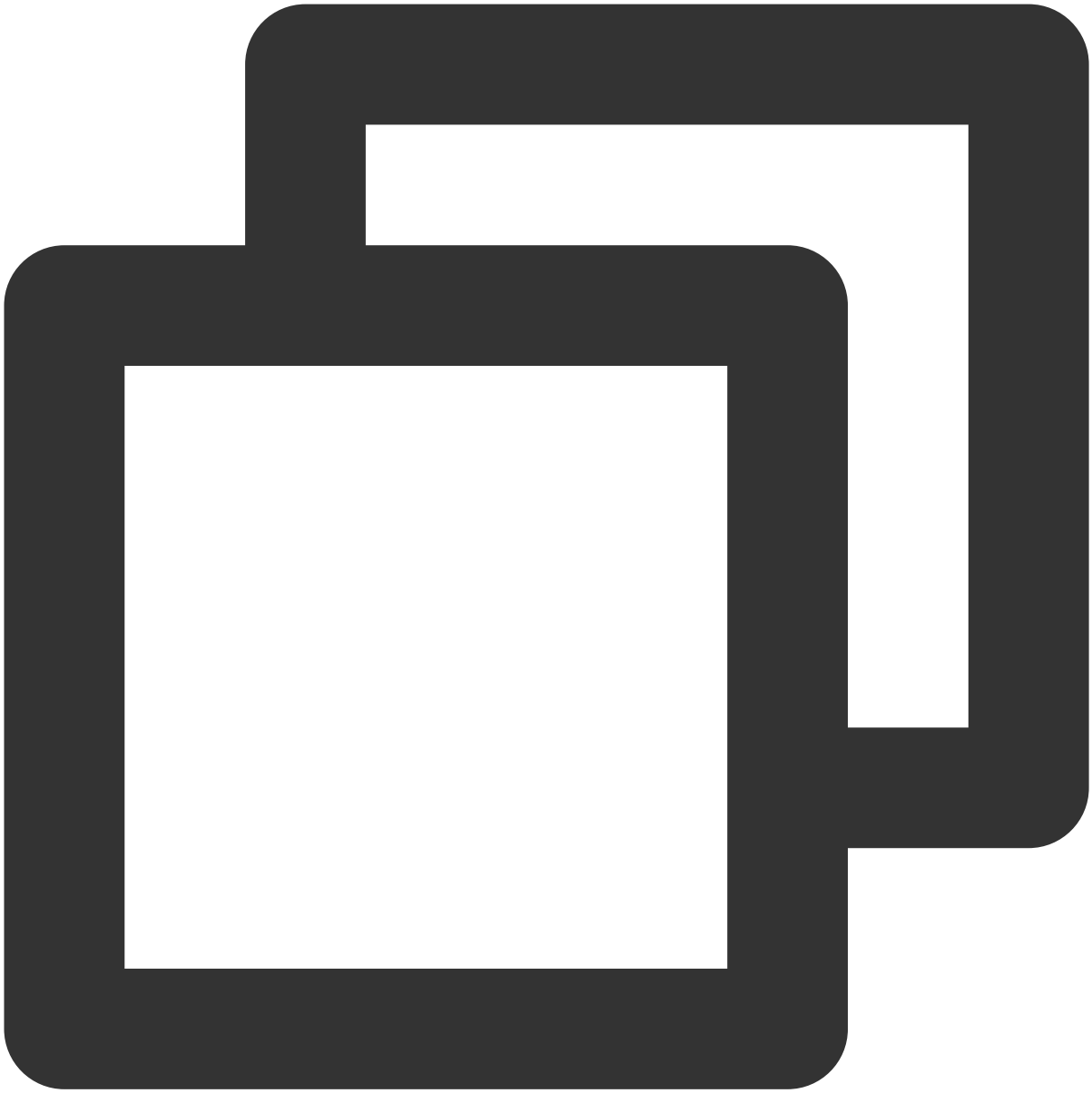
国际化语言

首先，将本地化工具导入到应用程序的入口文件中。



```
import 'package:tencent_cloud_chat_intl/localizations/tencent_cloud_chat_localizati
```

接下来，将本地化配置添加到 `MaterialApp` 或 `GetMaterialApp` 等第三方包提供的其他条目中。



```
MaterialApp(
  localizationsDelegates: const [
    /// Your configuration
    GlobalMaterialLocalizations.delegate,

    /// Add this line
    ...TencentCloudChatLocalizations.localizationsDelegates,  /// Add this line
  ],
  supportedLocales: [
    /// Your configuration
    ...S.delegate.supportedLocales,
```

```
    /// Add this line
    ...TencentCloudChatLocalizations.supportedLocales,
  ],
  /// ... Other configurations
)
```

此外，您可以根据您的业务逻辑设置语言区域设置 `locale` ，例如在应用启动时记录用户指定的语言，而不是遵循系统设置。此配置将同时应用于您的项目和聊天UIKit。

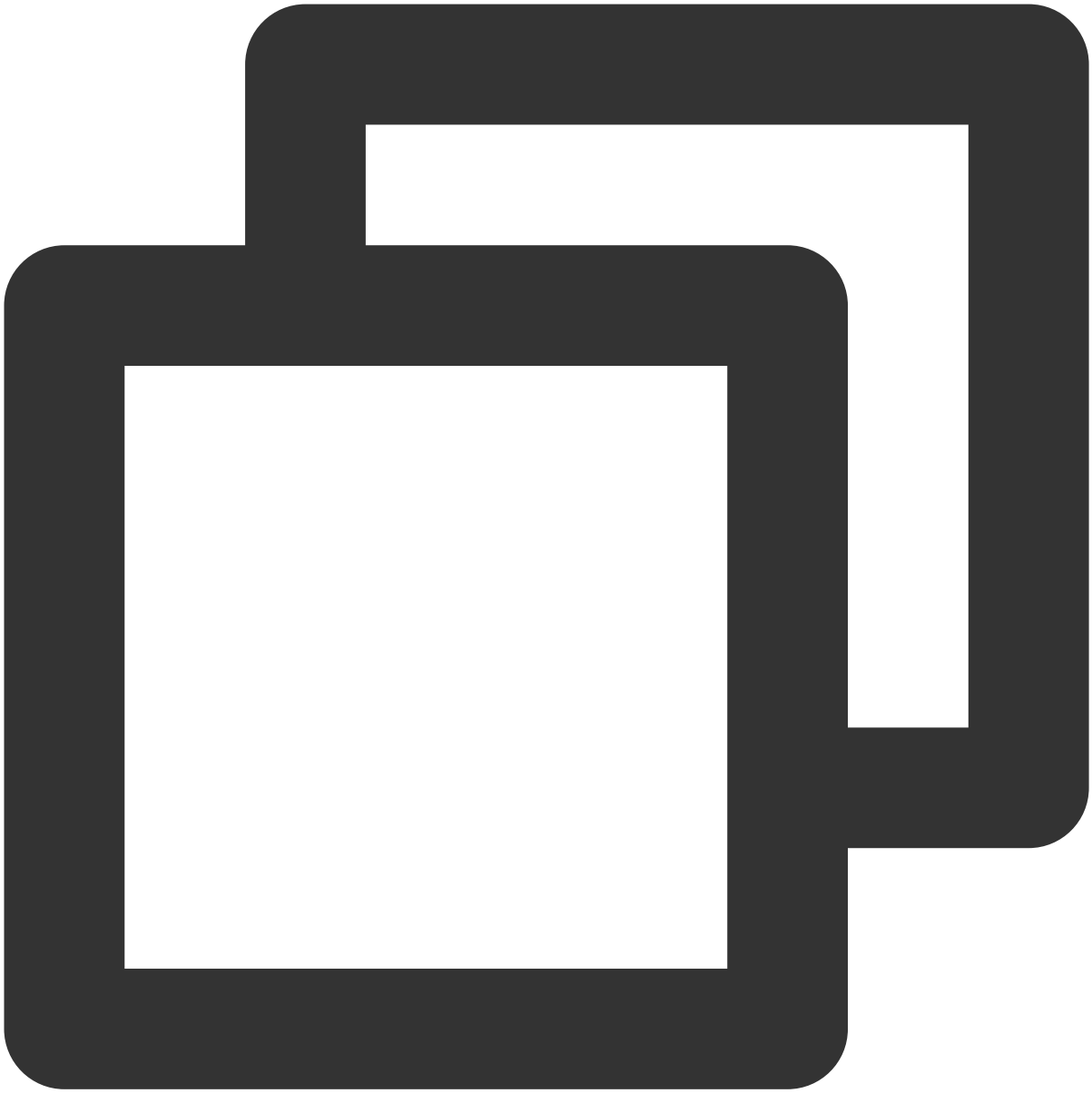
有关本地化定制的更多信息，包括添加或删除语言、添加本地化条目和修改翻译单词，请参考此指南。

可选操作

Theme / Theme Mode

UIKit 的主题数据由 `TencentCloudChatTheme` 类定义，通过 `TencentCloudChat.dataInstance.Theme` 全局维护和管理。

这允许您从任何位置访问主题：

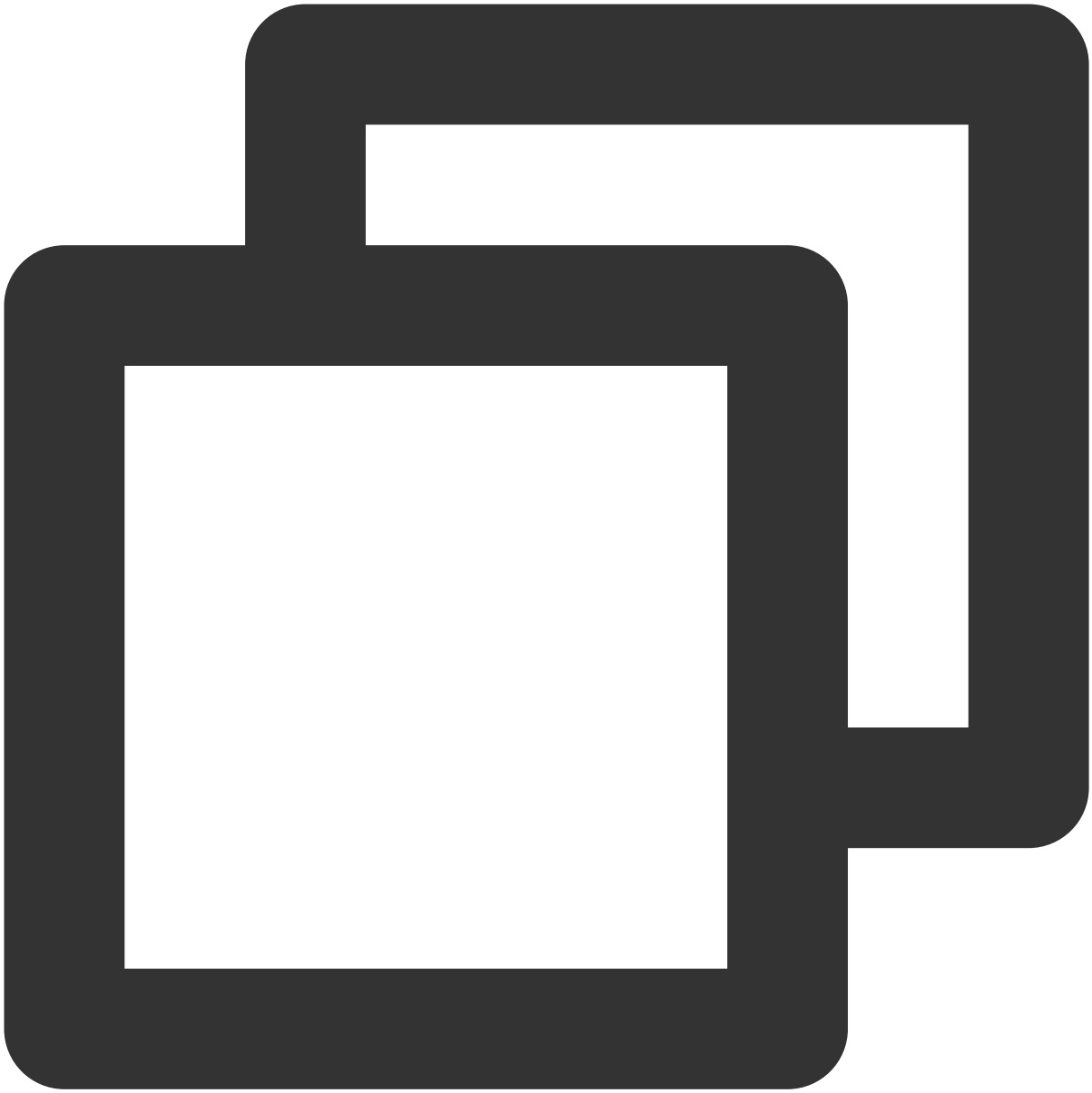


```
TencentCloudChatTheme theme = TencentCloudChat.dataInstance.theme;
```

此主题实例包括一个主题模型（包括亮色和暗色模式的主题数据）和亮度（亮色和暗色模式状态）。

此外，您可以通过我们为亮色和暗色模式提供的 Material 3 样式主题数据，从 `MaterialApp` 指定 `theme` 和 `darkTheme` 。您还可以根据我们维护的亮度状态设置 `themeMode` 状态。这确保了您的应用程序和我们的 Chat UIKit 在外观上保持一致，提高了用户体验。（您可以按照下面的描述自定义此主题样式。）

为实现此目的，我们建议将您的入口小部件（托管 `MaterialApp` 的小部件）转换为 `StatefulWidget` 。将 `TencentCloudChatTheme` 主题作为状态添加，并监听 `Stream<TencentCloudChatTheme>?` `themeDataListener` 以更新其值并根据动态、可自定义的主题数据构建应用程序。以下是一个示例代码：



```
// Theme instance for the Chat UIKit
TencentCloudChatTheme theme = TencentCloudChat.dataInstance.theme;

// Listener for theme data changes
Stream<TencentCloudChatTheme>? themeDataListener = TencentCloudChat.eventBusInstanc

// Callback for handling theme data changes
void _themeDataChangeCallback(TencentCloudChatTheme themeData) {
  setState(() {
    theme = themeData;
  });
```

```
}

// Adds a listener for theme data changes
void _addThemeDataChangeListener() {
  themeDataListener?.listen(
    _themeDataChangeCallback,
  );
}

@override
void initState() {
  super.initState();
  _addThemeDataChangeListener();
}

// .......

return MaterialApp(
  themeMode: theme.brightness != null ? (theme.brightness == Brightness.light ? The
  theme: theme.getThemeData(brightness: Brightness.light),
  darkTheme: theme.getThemeData(brightness: Brightness.dark),
   /// ... Other configurations
);
```

要自定义 Chat UIKit 的外观主题和全局主题（如果如上所示在 `MaterialApp` 中指定），请使用 `TencentCloudChatCoreController.setThemeColors` 方法为亮色和暗色模式指定外观颜色。有关具体使用说明，请参阅代码中的注释。

要切换主题模式（亮度），请使用 `TencentCloudChatCoreController.setBrightnessMode` 或 `TencentCloudChatCoreController.toggleBrightnessMode`。有关具体使用说明，请参阅代码中的注释。

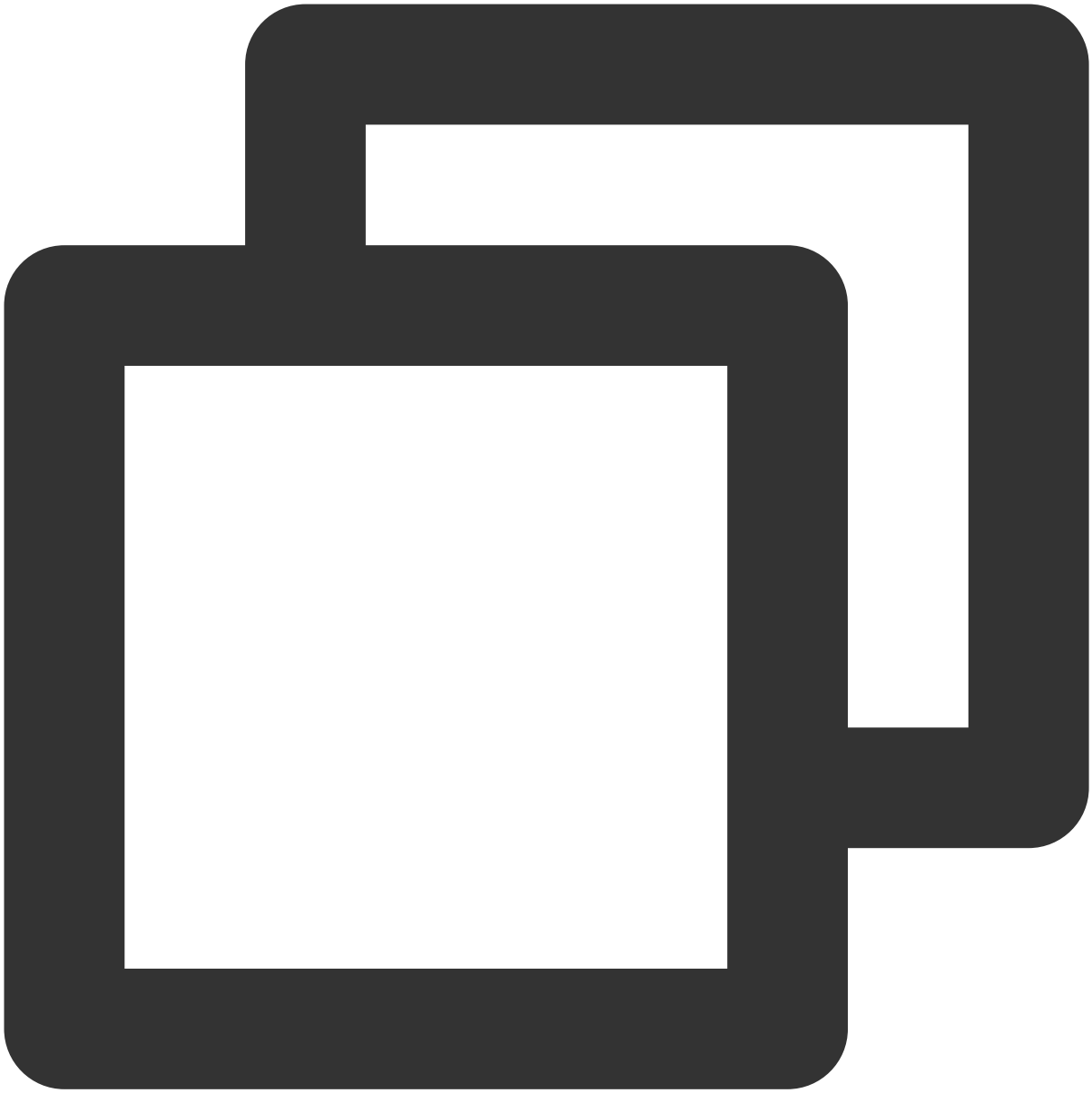
4. 初始化和登录：

调用 `TencentCloudChat.controller.initUIKit` 方法进行初始化和登录。调用说明和参考代码如下：

说明：

我们高度建议配置 `callbacks` 以可定制的方式通过 `Dialog` 或 `ToolTip` 高效地处理 SDK API 错误和需要用户关注的特定 UIKit 事件。

有关详细说明、用法和事件代码列表，请参阅本指南。



```
await TencentCloudChat.controller.initUIKit(
  config: TencentCloudChatConfig(), /// [可选]: 影响整个聊天界面的全局配置，包括用户相关配置
  options: TencentCloudChatInitOptions(
    sdkAppID: , /// [必需]: 腾讯云聊天应用的SDKAppID
    userID: , /// [必需]: 已登录用户的userID
    userSig: , /// [必需]: 已登录用户的userSig
  ),

  components: TencentCloudChatInitComponentsRelated( /// [必需]: 模块化UI组件相关设置，
    usedComponentsRegister: [
      /// [必需]: 聊天界面中使用的组件的注册函数列表。
```

```
        TencentCloudChatConversationManager.register,
        TencentCloudChatMessageManager.register,
        /// ......
        /// 上面的注册是示例。在此字段中，传入每个子模块UI包的注册。
        /// 安装每个子模块UI包后，需要在此声明才能使用。
      ],
      componentConfigs: TencentCloudChatComponentConfigs(
        /// [可选]：在此处为每个UI模块组件提供自定义配置。这些构建器将全局应用。
      ),
      componentBuilders: TencentCloudChatComponentBuilders(
        /// [可选]：在此处为每个UI模块组件提供自定义UI构建器。这些构建器将全局应用。
      ),
      componentEventHandlers: TencentCloudChatComponentEventHandlers(
        /// [可选]：在此处为UI组件相关事件提供自定义事件处理程序。这些构建器将全局应用。
      ),
    ),

    /// [关键]：强烈建议将以下回调侦听器集成到SDK事件、SDK API错误和需要用户关注的特定UIKit事件的
    /// 有关详细用法，请参阅本自述文件末尾的“引入UIKit回调”部分。
    callbacks: TencentCloudChatCallbacks(
      onTencentCloudChatSDKEvent: V2TimSDKListener(),  /// [可选]：处理SDK事件，如onKick
      onTencentCloudChatSDKFailCallback: (apiName, code, desc) {}, /// [可选]：处理SDK
      onTencentCloudChatUIKitUserNotificationEvent: (TencentCloudChatComponentsEnum c
    ),

    plugins: [],  /// [可选]：使用的插件，如tencent_cloud_chat_robot等。具体用法请参阅每个插
  );
```

完成 UIKit 的基本集成过程后，您可以继续探索每个模块化包的文档，完成各个 UI 组件的集成。

这将帮助您了解每个组件的具体用法和定制选项，让您创建满足需求的定制聊天应用。

说明：

本文档中列出了每个模块化包的文档。

模块化 UI 包的常见用法

对于大多数用例，只有 `TencentCloudChatConversation` 和 `TencentCloudChatContact` 组件（如果需要）需要手动实例化并添加到小部件中。只要在 `initUIKit` 调用期间的 `components` 参数中的 `usedComponentsRegister` 中声明了其他组件，就会根据用户操作自动导航。

要集成这两个基本组件，只需实例化它们并在 `build` 方法中返回它们，无需任何其他配置参数。

高级用法

模块化 UI 包的高级用法

组件输入参数

每个模块化UI组件包提供四个统一的输入参数：

options：确保正确功能的组件特定参数。一些通用组件可能不需要此参数。

config：一组组件特定配置，用于进行细粒度定制，例如调整消息组件的附件区域配置。

builders：构建组件内部小部件的方法集合，实现外部UI定制。每个构建器都包括必要的参数和方法，使数据和逻辑层方法随时可用。

eventHandlers：处理组件特定事件的回调，包括 `uiEventHandlers`（例如各种 `onTap` 类似事件）和 `lifeCycleEventHandlers`（例如在发送消息后触发的事件）。这些处理程序允许在响应用户交互和组件生命周期更改时进行自定义行为。

说明：

`options` 参数应在组件构造函数中指定。目前，只需要 `TencentCloudChatMessage`、`TencentCloudChatUserProfile` 和 `TencentCloudChatGroupProfile` 组件，用于指定目标用户或组。

其他三个参数可以在组件构造函数中为特定组件实例指定，也可以在 `initUIKit` 调用期间的 `components` 参数中全局指定，或者从每个组件的管理器中管理，影响相应组件的所有实例。

对于集成过程，我们建议使用**下面各节描述的全局配置方法**。

全局：配置组件

每个组件提供一组组件特定配置，用于进行细粒度定制，例如调整消息组件的附件区域配置。

有两种方法可以在全局范围内定制配置：在 `initUIKit` 期间和使用管理器。

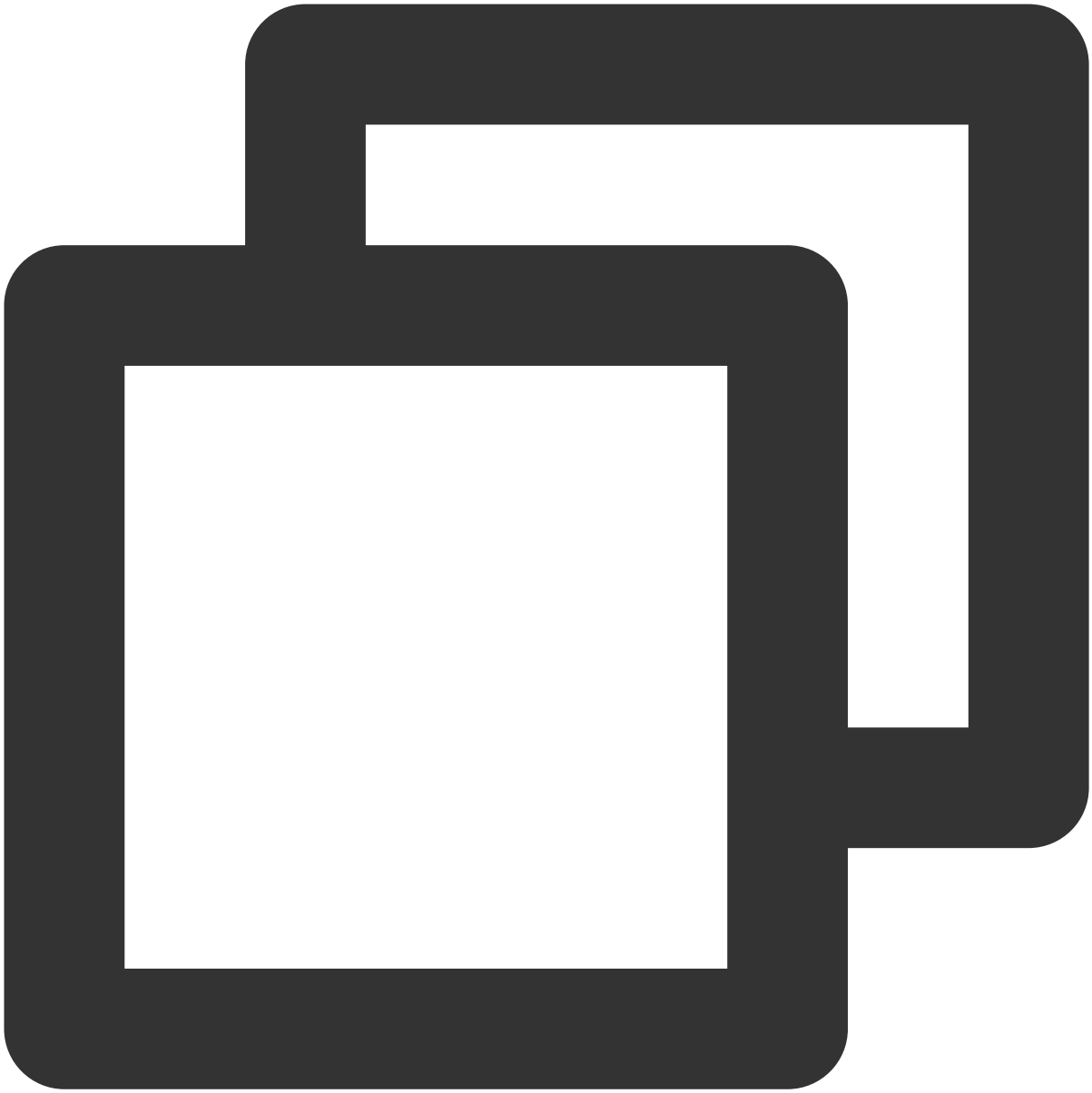
在初始化期间：在 `initUIKit` 调用期间使用 `components` 参数为每个模块化UI组件指定 `componentConfigs`。

通过管理器：利用每个组件的管理器从代码库的任何位置动态修改配置。

要动态修改影响相应组件所有实例的配置，请按照以下步骤操作：

1. 通过将 `Manager` 附加到组件名称（例如 `TencentCloudChatMessageManager`）从组件的管理器中访问全局 `config` 实例。
2. 调用 `setConfigs` 方法并传递要修改的配置。这将替换以前的配置并立即应用更改。

例如，您可以通过 `TencentCloudChatConversationManager` 对象（例如 `TencentCloudChatConversationManager.controller`）使用来自 `TencentCloudChatConversation` 组件的 `config` 修改一些配置：



```
TencentCloudChatConversationManager.config.setConfigs(
    useDesktopMode: true,
);
```

全局：自定义 UI 小部件

UI构建器允许外部 UI 定制。如果未定义构建器，则使用内置UI小部件。每个构建器都带有必需的参数和方法，可以轻松访问数据和逻辑层方法。这意味着您可以使用提供的上下文数据，例如特定的对话，返回适用于该上下文的构建器。

有两种模式可以在全局范围内定义自定义构建器：在 `initUIKit` 期间和通过管理器。

在初始化期间：在 `initUIKit` 调用期间使用 `components` 参数为每个模块化UI组件指定 `componentBuilders` 。

通过管理器：使用说明显示在下一节 **动态更新UI构建器**。

我们建议使用以下动态定义方法，允许从代码库的任何位置进行修改。

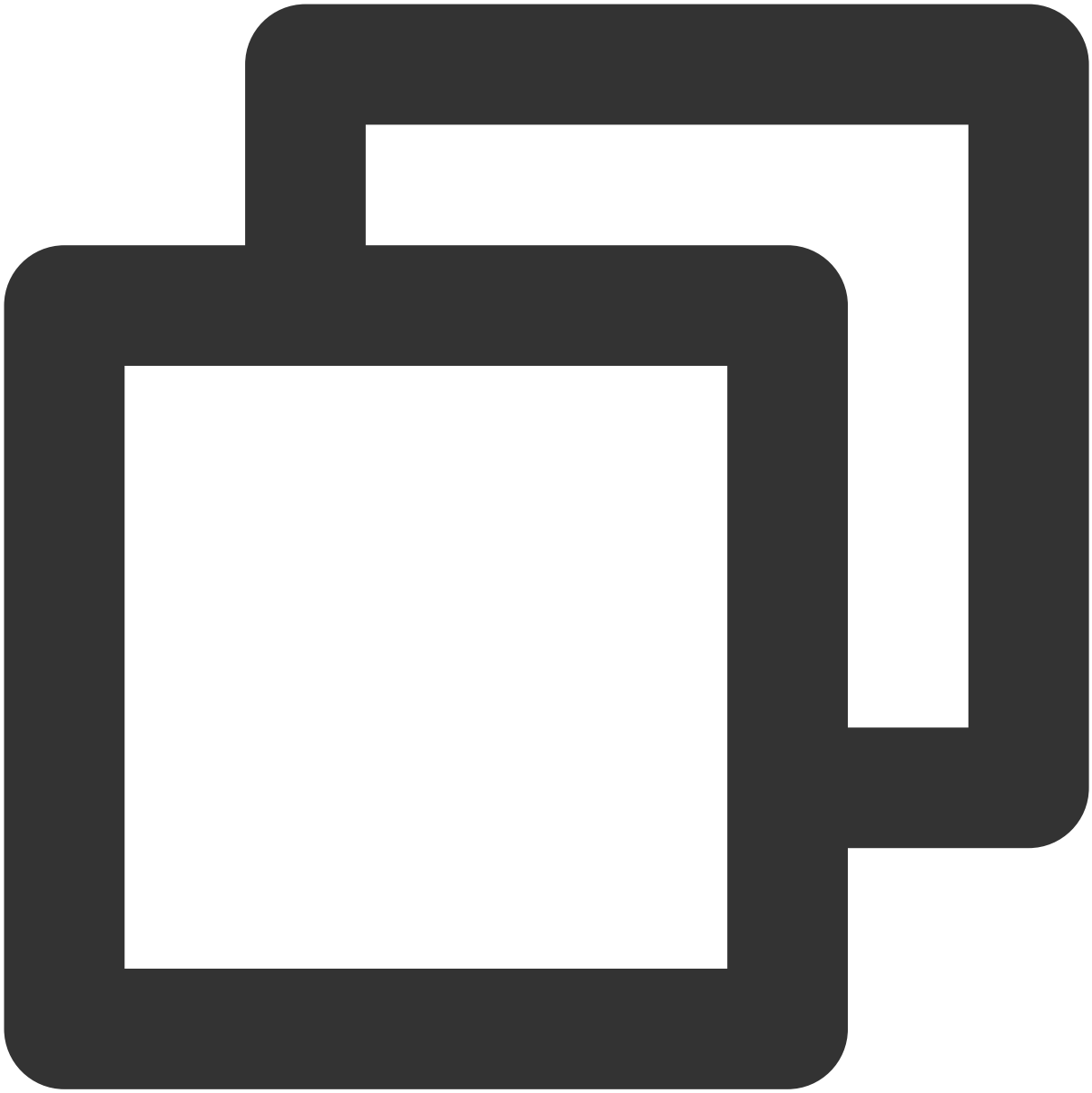
动态更新UI构建器

请注意，此方法仅适用于在 `initUIKit` 调用期间使用 `components` 参数定义的全局构建器或未指定自定义构建器时的默认构建器。此方法不能用于修改组件实例级别的构建器，即实例化组件时传递的 `builders` 参数。

要动态更新影响特定组件所有实例的UI构建器，请按照以下步骤操作：

1. 通过将 `Manager` 附加到组件名称（例如 `TencentCloudChatMessageManager` ）从组件的管理器中检索全局 `builder` 实例。
2. 在检索到的实例上调用 `setBuilders` 方法并提供您的自定义构建器。

例如，要自定义 `TencentCloudChatConversation` 组件的UI小部件，可以使用以下代码：



```
TencentCloudChatConversationManager.builder.setBuilders(
  conversationItemContentBuilder: (V2TimConversation conversation) => Container(),
  conversationHeaderBuilder: () => Container(),
);
```

在此示例中，您只需要指定要自定义的构建器，而其他构建器保持不变。

使用此方法，您可以在应用程序的任何位置动态更新全局构建器。

每个模块化UI组件都有一个与其特定UI小部件相关联的 `builder` ，使用方式在所有组件中保持一致。

全局：处理组件级事件

每个组件都配备了两种类型的事件：`uiEventHandlers`（例如，onTap类似事件）和`lifeCycleEventHandlers`（与业务相关的事件）。

通常，事件提供了一套全面的信息参数，以帮助您实现自定义业务逻辑。对于返回布尔值的事件（占大多数），返回`true`可防止执行默认业务逻辑，而返回`false`则允许继续。

自定义事件处理允许您的业务逻辑与默认UIKit操作无缝集成。例如，您可以自定义组件导航，如下面的 **案例：组件之间的手动导航** 部分所示。

有两种方法可以全局附加您的事件处理程序：

1. 在`initUIKit`调用期间，使用`components`参数并为每个模块化UI组件指定`componentEventHandlers`。
2. 使用每个组件的管理器从代码库的任何位置动态附加和更新事件处理程序。

要动态附加和更新监听来自所有实例的事件的事件处理程序，请按照以下步骤操作：

1. 通过将 Manager 附加到组件名称（例如 TencentCloudChatMessageManager）从组件的管理器中访问全局 eventHandlers 实例。
2. 对于`uiEventHandlers`或`lifeCycleEventHandlers`，调用`setEventHandlers`以更新特定事件处理程序。

注意：这将导致相应事件的先前附加的处理程序失效，即被覆盖。

有关示例用法，请参阅 **案例：组件之间的手动导航** 部分。

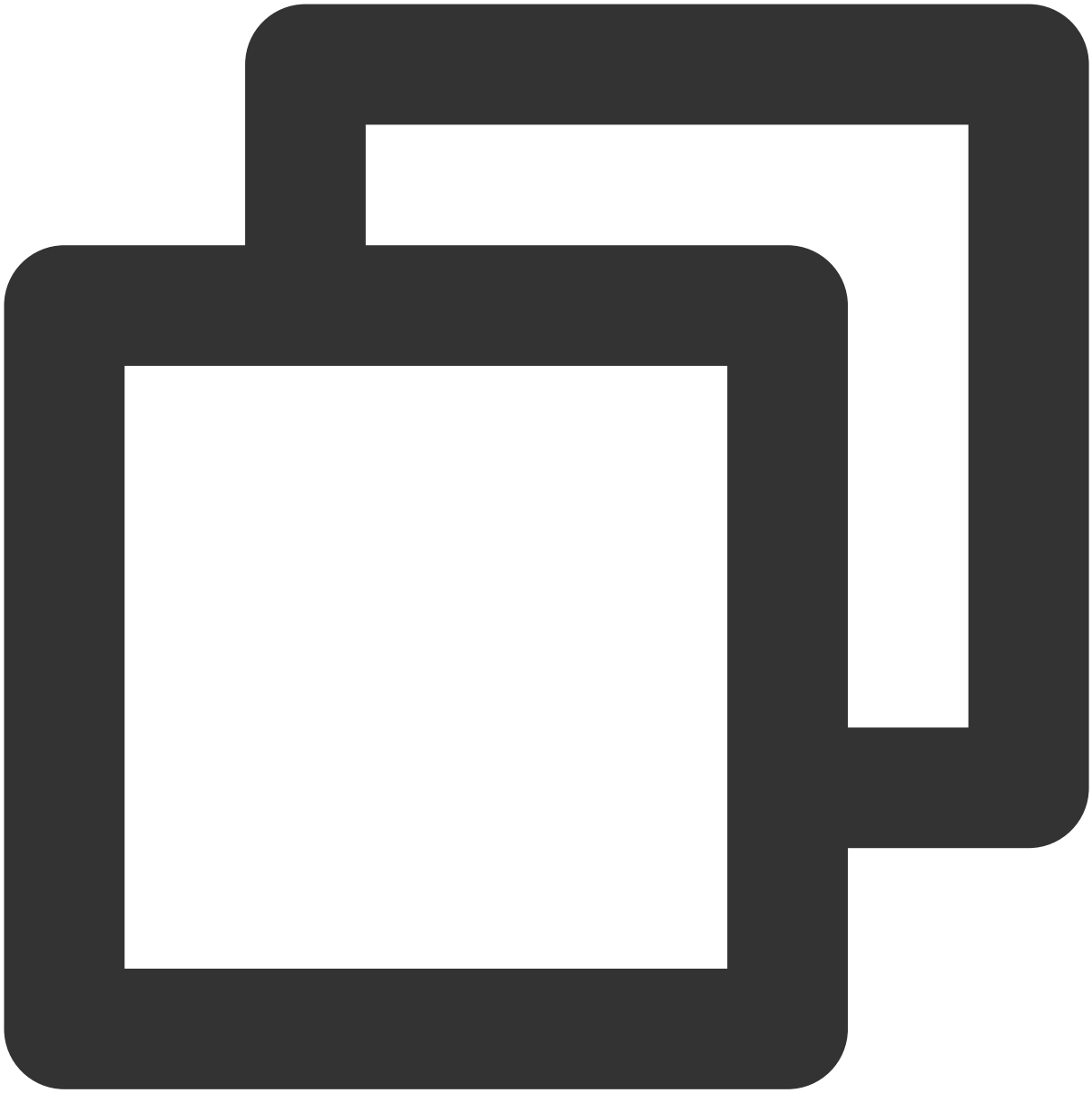
无论您选择哪种方法，您只需要附加所需的事件处理程序，而其他处理程序保持未指定。

案例：组件之间的手动导航

如前所述，只要声明了我们的组件，就支持在它们之间进行自动导航。但是，如果您的业务逻辑与自动导航不符（例如，您需要导航到其他组件或实现其他业务逻辑），您可以通过监听点击事件并阻止默认导航来手动处理事件，以满足您的需求。

对于提供的组件之间的手动导航，建议附加相应的`onTap`类似事件处理程序并返回`true`或`false`来决定是否继续执行内置的自动导航。

例如，当从`TencentCloudChatContact`组件点击联系人项时，您可以执行以下示例中显示的自定义导航：

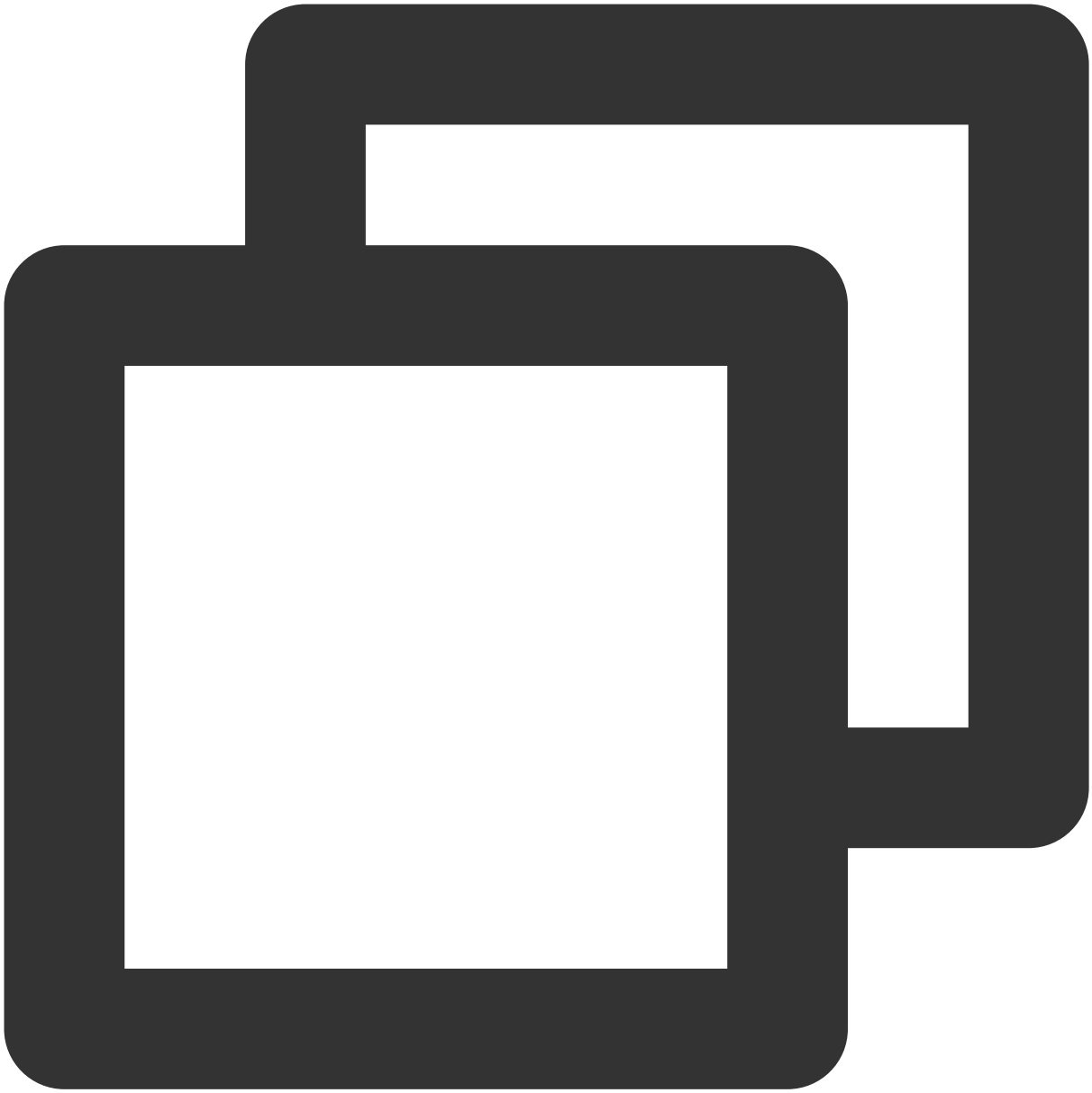


```
TencentCloudChatContactManager.eventHandlers.uiEventHandlers.setEventHandlers(
    onTapContactItem: ({
    String? userID,
    String? groupID,
    }) async {
        // 根据提供的userID、groupID和您的业务逻辑判断是否需要手动导航。
        if (needed) {
            // 您的自定义业务逻辑
            return true;
        } else {
            // 继续执行内置逻辑
```

```
                return false;
            }
        },
    );
```

全局：控制每个组件

每个组件都与一组控制方法相关联。这些方法提供了对组件行为的增强功能和控制。
要使用这些控制方法，首先从相应组件的管理器中检索 `controller` 实例，该实例是将 `Manager` 附加到组件名称（例如 `TencentCloudChatMessageManager` ）。然后，您可以调用 `controller` 实例提供的方法。例如，您可以通过 `TencentCloudChatMessageManager` 对象（例如 `TencentCloudChatMessageManager.controller` ）使用来自 `TencentCloudChatMessage` 组件的控制器。要发送消息并将其添加到消息列表UI中，请使用以下代码：



```
// 使用聊天SDK创建消息。
final res = await TencentCloudChat.instance.chatSDKInstance.messageSDK.createTextMe
if(res != null ){
  // 然后使用从TencentCloudChatMessageManager获取的控制器发送创建的消息。
  TencentCloudChatMessageManager.controller.sendMessage(createdMessage: res, userID
}
```

每个模块化UI组件都有一个与其特定功能相关联的控制器。使用方法与上面显示的示例控制器一致。

有关每个控制器方法的详细说明，请参阅每个方法提供的注释。

TencentCloudChat.controller中的其他方法

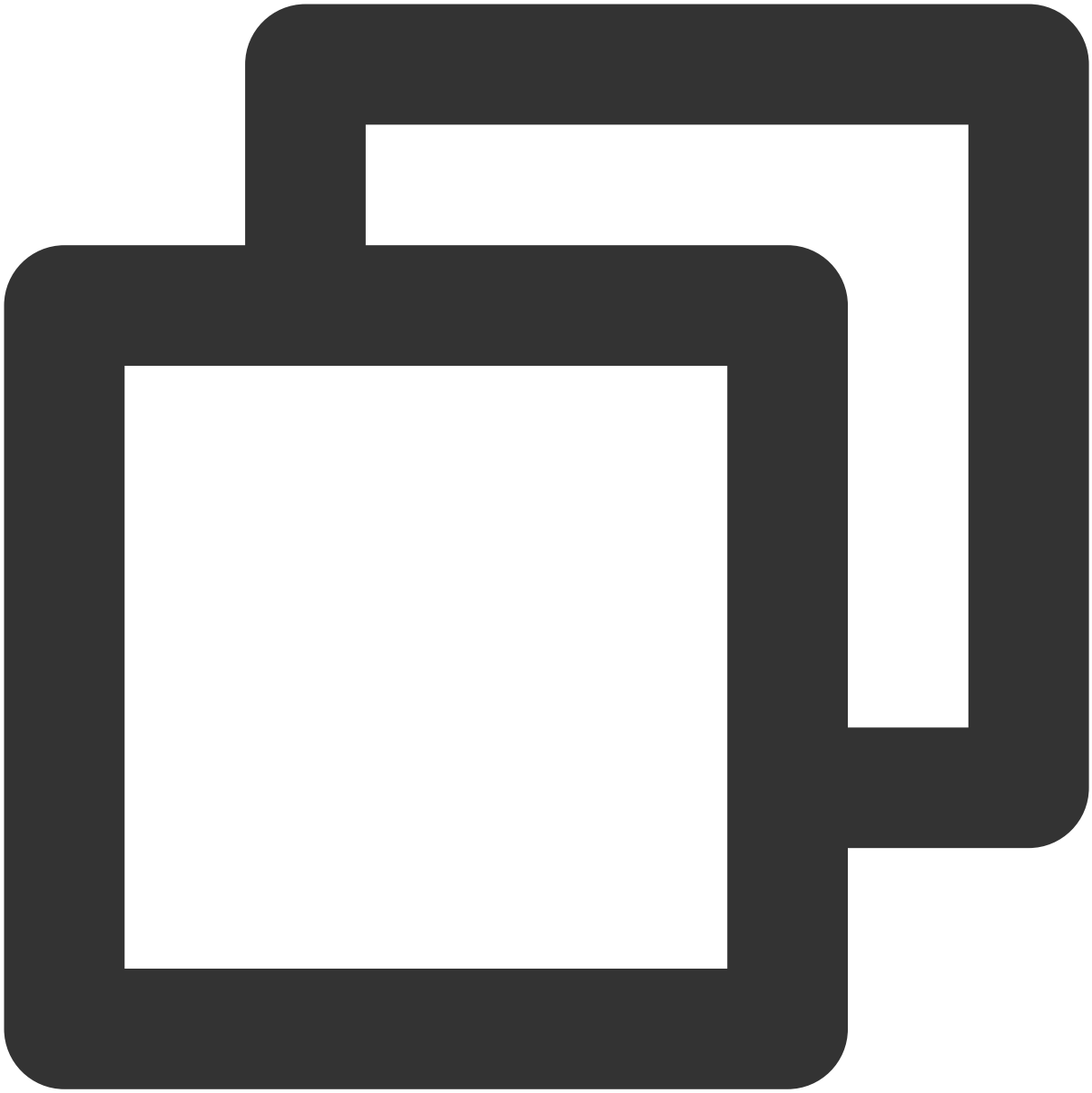
在上面的基本用法部分，我们解释了如何使用 `TencentCloudChat.controller` 初始化UIKit并登录。
此控制器还包含其他几个方法，可用于控制UIKit的一些全局方面。例如：
toggleBrightnessMode：此方法允许您在暗色和亮色模式之间切换。
getThemeData：此方法以material3 ThemeData类的形式返回内置主题配置。这可用于为您的 `MaterialApp` 配置 `theme` 参数，确保我们的UIKit和项目的其他组件具有一致的外观。
setThemeColors：此方法允许您自定义UIKit中暗色和亮色模式的颜色配置。这确保我们的UIKit和项目的其他组件具有一致的外观。此方法设置的配置将在所有UI组件中生效。
setBrightnessMode：此方法允许您设置当前亮度模式。
有关更多方法及其描述，请参阅每个方法的注释。这使您可以更好地控制UIKit的行为和外观，使其适应项目的需求。

模块化 UI 组件包

说明：
如果您在使用我们的模块化UI包时遇到下面列出的这些空安全错误，请参阅本指南以获得帮助。

```
I/flutter (10740): message: Null check operator used on a null value
I/flutter (10740): stack: #0      tL10n (package:tencent_cloud_chat_intl/tencent_cloud
I/flutter (10740): #1      tL10n (package:tencent_cloud_chat_intl/tencent_cloud_chat_i
```

这些问题可能源于在项目完全初始化之前过早使用Chat UIKit组件。
一种可行的解决办法是在使用这些组件之前手动运行后续代码。



```
TencentCloudChatIntl().init(context);
```

如果上述解决方案无效，请确保您已经用提供的 `TencentCloudChatMaterialApp` 替换了入口 `MaterialApp` ，或者如上面“基本用法”部分所述，在最早阶段手动实现了 UIKit 的全局配置。

Conversation

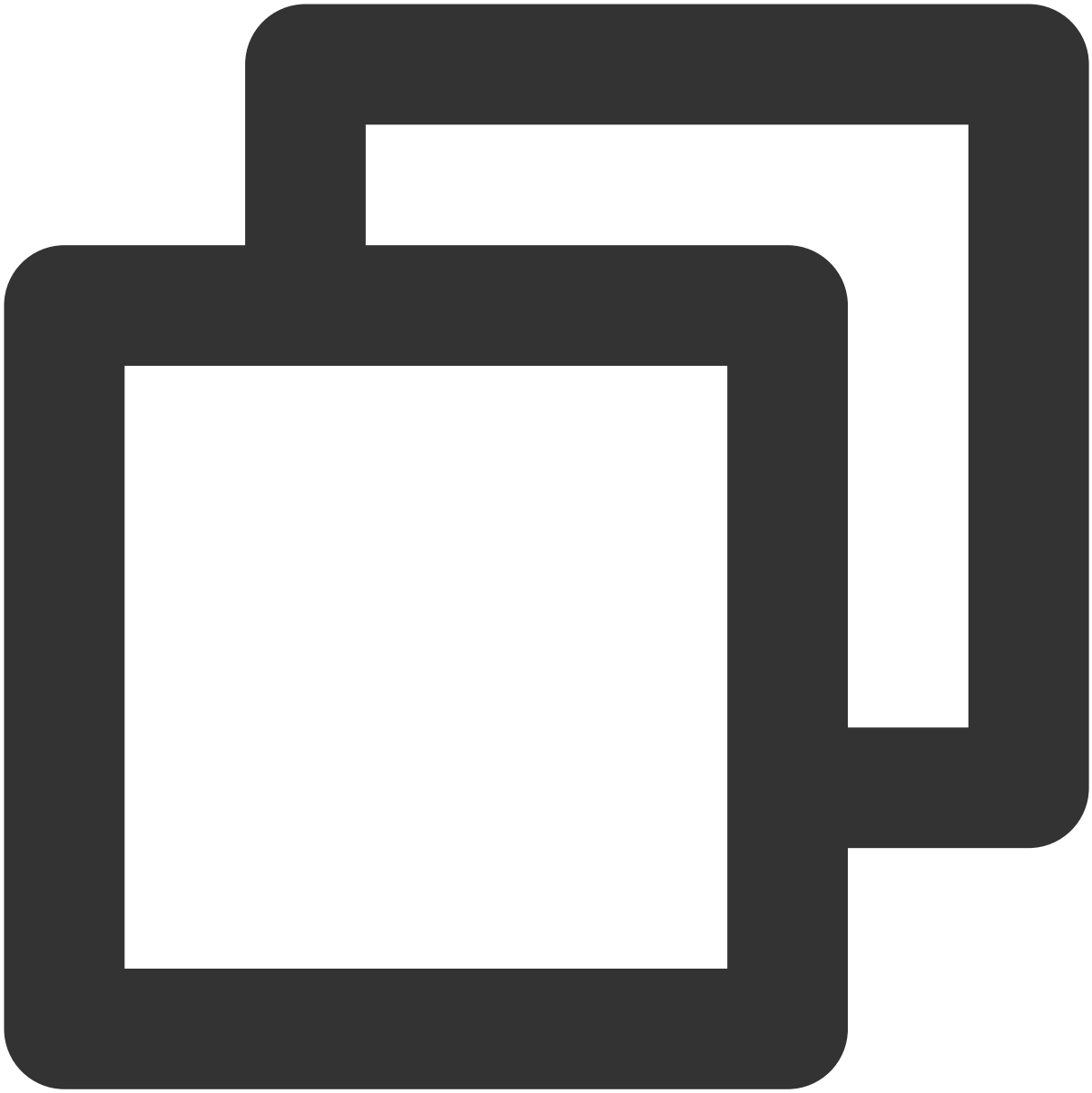
介绍腾讯云IM UIKit的会话组件，该组件旨在为您的聊天应用程序提供一个多功能的会话列表，无缝适应桌面和移动环境。

会话组件提供一个会话列表，显示所有参与的会话，按最后活动时间排序。它还支持管理会话信息，确保聊天体验顺畅有序。
当与tencent_cloud_chat_message组件一起使用时，会话组件可以在移动设备上点击会话时自动导航到相应的消息聊天页面。在桌面环境中，消息聊天页面出现在右侧区域，允许动态切换会话。

开始使用

导入和声明

首先，将tencent_cloud_chat_conversation UI模块添加到您的项目中。
安装完成后，您需要在 `TencentCloudChat.controller.initUIKit` 方法的 `components` 的 `usedComponentsRegister` 参数中注册此UI组件。以下是一个例子：



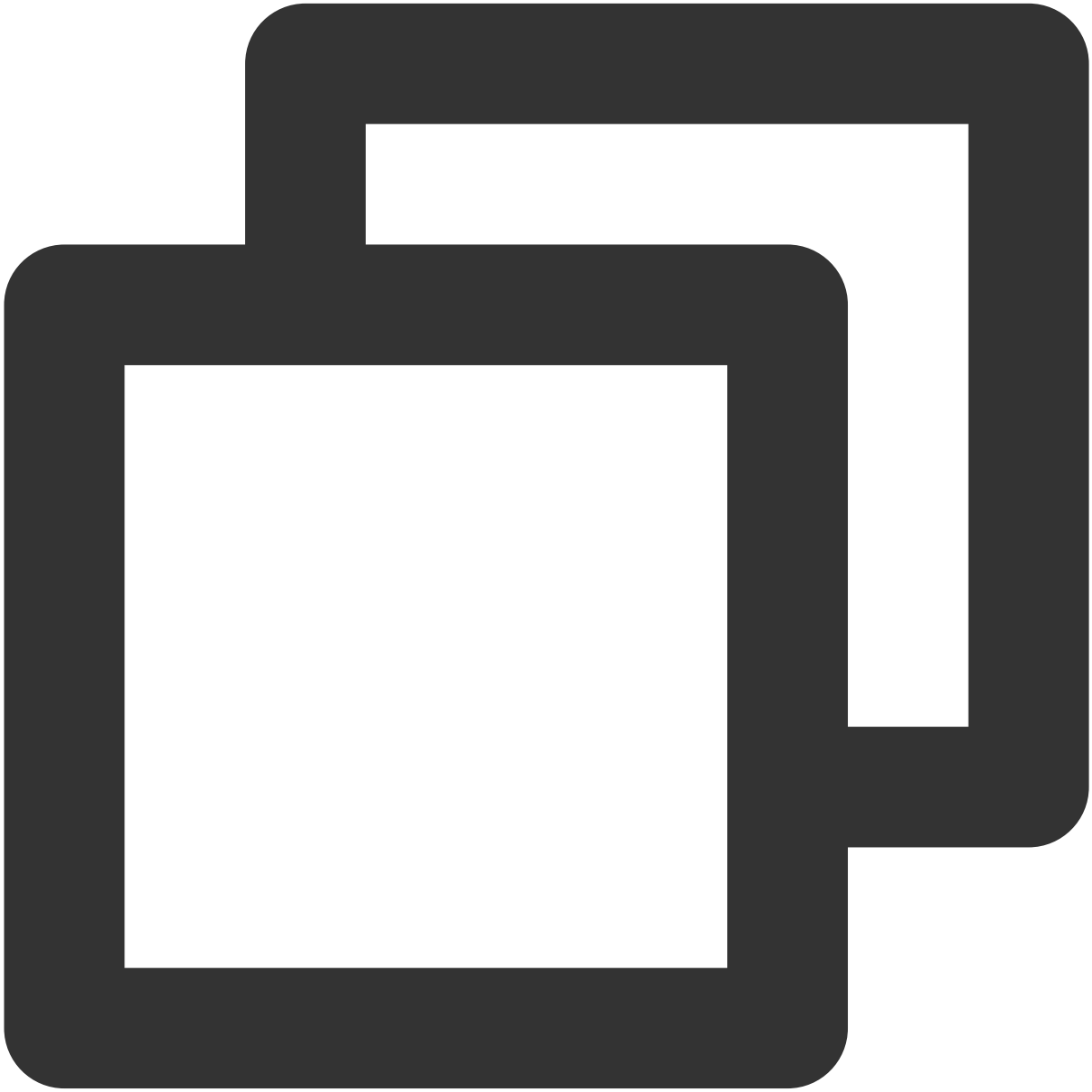
```
await TencentCloudChat.controller.initUIKit(
  components: TencentCloudChatInitComponentsRelated(
    usedComponentsRegister: [
      TencentCloudChatConversationManager.register, /// 添加这一行
      /// ...
    ],
  /// ...
  ),
  /// ...
);
```

实例化和使用组件

使用会话组件非常简单。只需实例化一个 `TencentCloudChatConversation` 实例，并在所需的页面上渲染它。

默认情况下，组件会自动获取并显示所有会话信息，无需任何额外参数。

您可以在想要显示会话列表的页面的 `build` 方法中使用此实例。



```
@override
Widget build(BuildContext context) {
  return const TencentCloudChatConversation();
}
```

只需几行代码，您就可以轻松地将会话组件集成到您的聊天应用程序中，并显示一个会话列表供用户交互。

定制细节

使用config

对于简单和基本的配置，您可以使用 `config` 参数。会话组件的 `config` 由 `TencentCloudChatConversationConfig` 类提供。
它包括控制各种数据类型的选项，如布尔值、整数和自定义参数。
例如， `useDesktopMode` 配置决定是否在桌面环境中，当与tencent_cloud_chat_message组件一起使用时，组件应跨越全屏宽度，将会话列表显示在左侧，将当前选定会话的 `Message` 组件显示在右侧，并支持动态切换。

使用builders

对于更深入的UI定制，您可以使用自定义构建器。会话组件的构建器由 `TencentCloudChatConversationBuilders` 类提供。
会话组件提供了几个构建器，如 `ConversationItemAvatarBuilder` 用于在会话项上显示头像， `ConversationItemContentBuilder` 用于在会话项中显示内容， `ConversationItemInfoBuilder` 用于在会话项中显示信息。

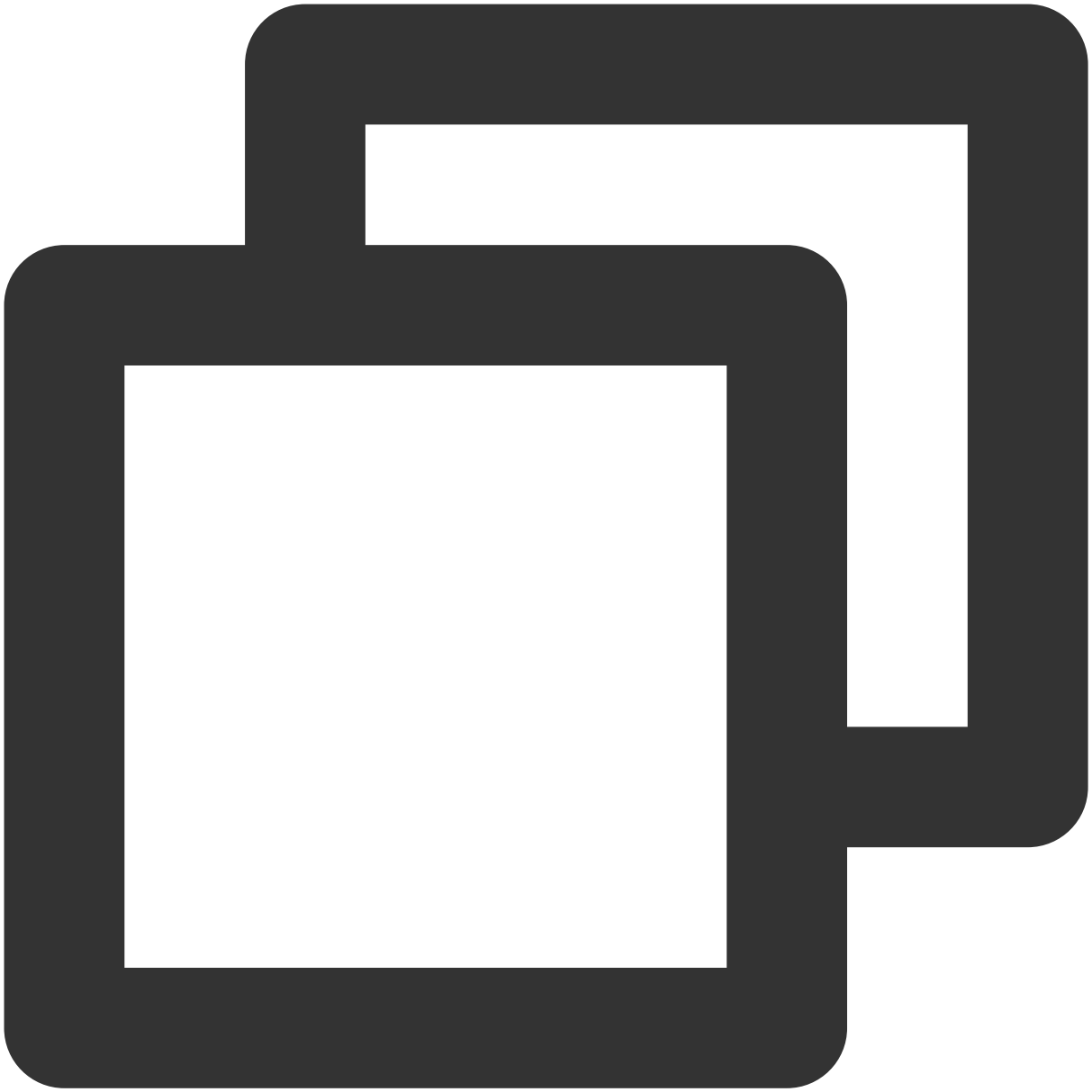
Message

此组件旨在通过提供基本和高级聊天功能，为您的聊天应用程序提供全面的消息体验。
消息组件由几个关键元素组成，包括用于显示会话信息的Header、用于展示消息历史的Message Listview和用于方便发送消息的Message Input。为了提升用户体验，它还包含丰富的动画和交互细节。
在其基础上，该组件提供了诸如发送、接收、复制、转发、预览和删除消息等基本聊天功能，确保流畅的聊天体验。
为了满足不同用户需求，它还包括高级功能。例如消息菜单、将消息标记为已读、显示群组阅读回执详情、支持表情反应、精确导航到特定消息、消息多选和提供广泛的定制功能。
当与tencent_cloud_chat_conversation和tencent_cloud_chat_contact组件一起使用时，消息组件可以实现无缝导航，无需手动实现导航。此外，当与tencent_calls_uikit集成时，它提供了发起语音/视频通话的能力，从而提高整体通信体验。
本质上，消息组件使您能够创建吸引人的、功能丰富的聊天应用程序，满足各种用户需求，提供愉悦的用户体验。

开始使用

导入和声明

首先，将tencent_cloud_chat_message UI模块添加到您的项目中。
安装完成后，您需要在 `TencentCloudChat.controller.initUIKit` 方法的 `components` 的 `usedComponentsRegister` 参数中注册此UI组件。以下是一个例子：

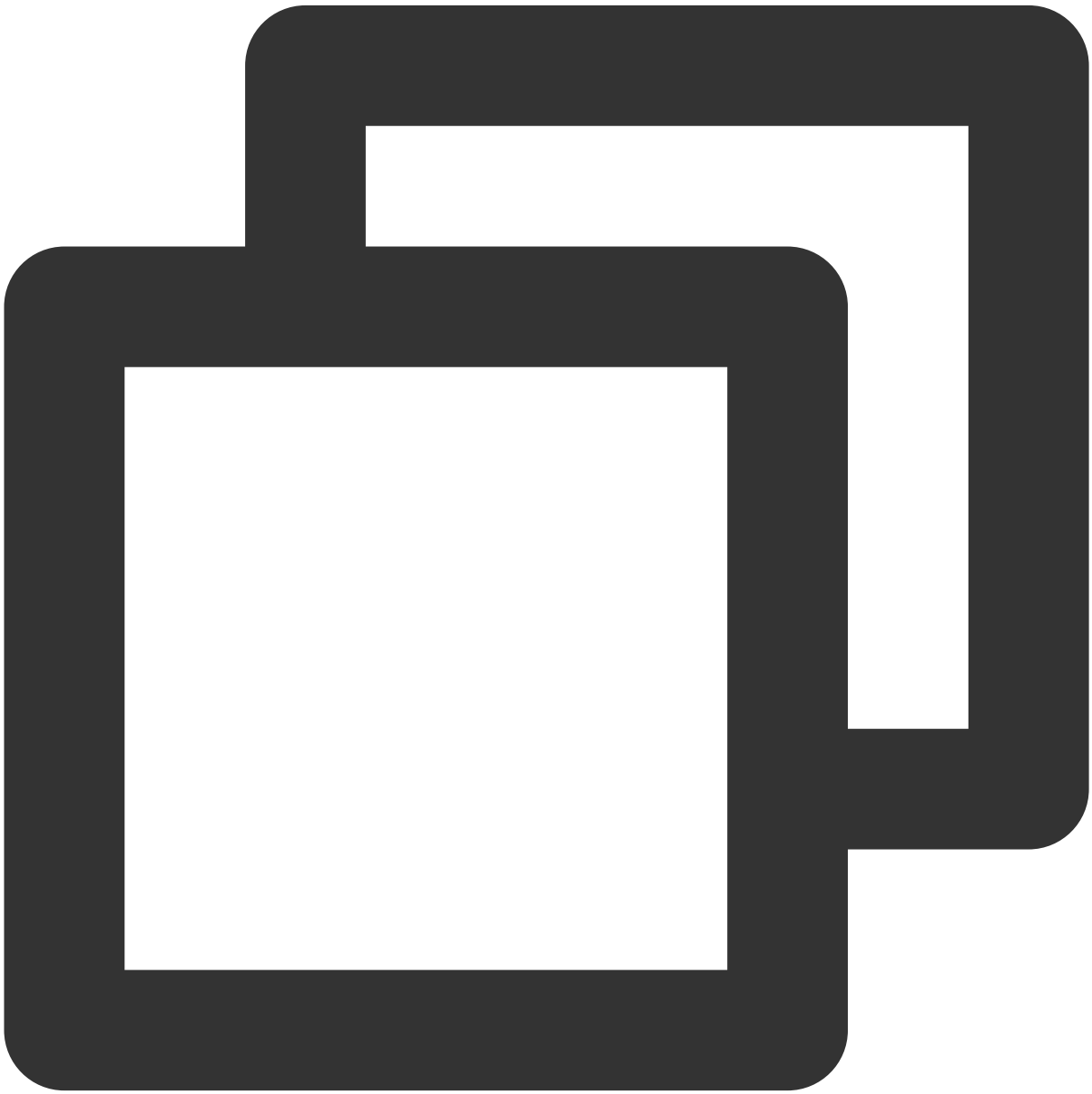


```
await TencentCloudChat.controller.initUIKit(
  components: TencentCloudChatInitComponentsRelated(
    usedComponentsRegister: [
      TencentCloudChatMessageManager.register, /// 添加这一行
      /// ...
    ],
  /// ...
  ),
  /// ...
);
```

如果您的项目包含模块化组件，如tencent_cloud_chat_conversation或tencent_cloud_chat_contact用于显示会话、联系人或群组列表，它们将自动从这些列表导航到消息组件。
如果仅需要从这些内置组件而不是从您的自定义页面进行导航，则 `Message` 组件集成仅需此单步完成。UIKit在内部处理导航转换，无需手动编码。
对于需要从自定义页面进行导航的项目，请参阅以下步骤。

导航到消息组件

在导航之前，准备一个 `TencentCloudChatMessageOptions` 实例，以指定聊天的会话：

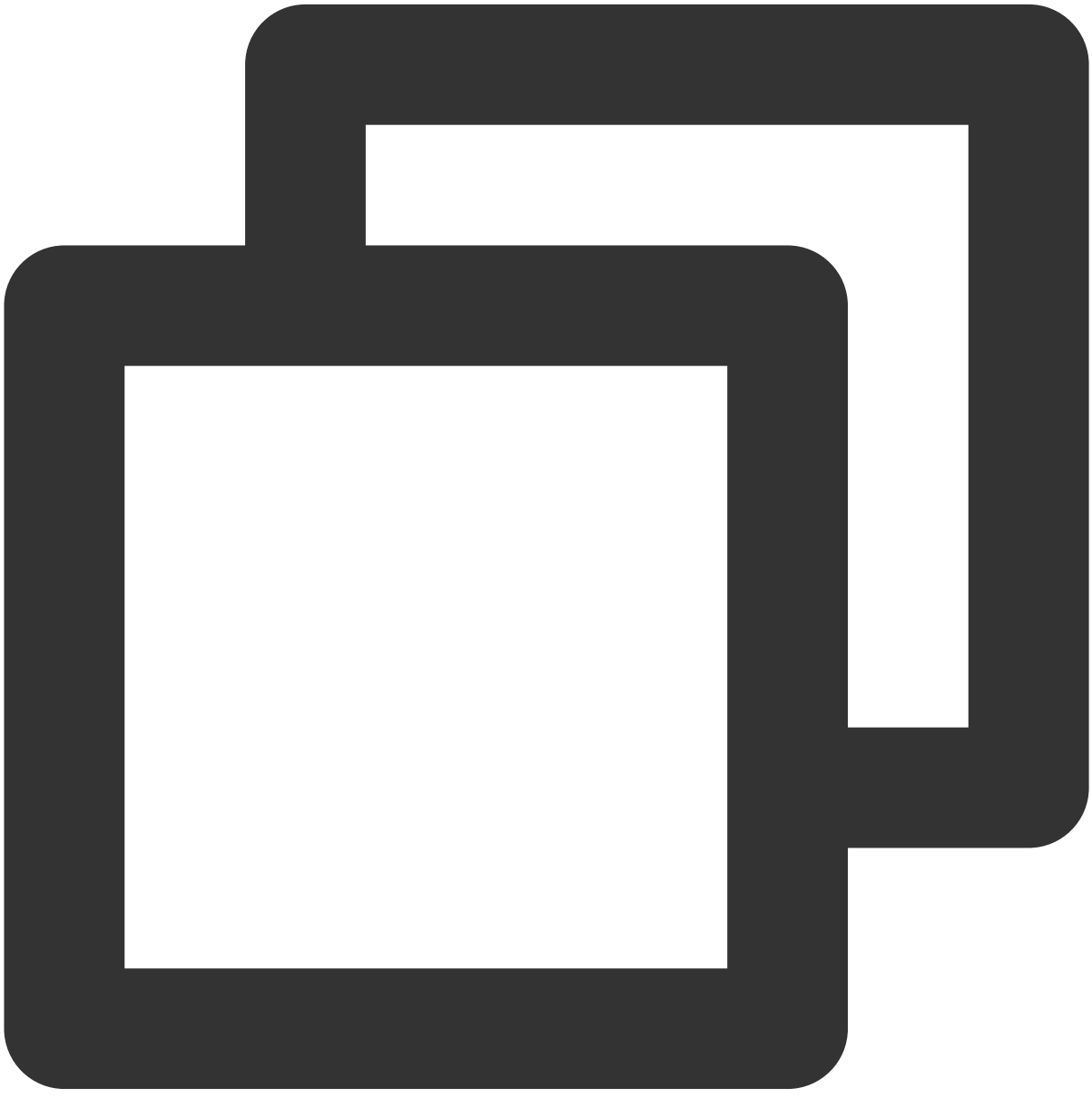


```
final messageOptions = TencentCloudChatMessageOptions(
```

```
      // 提供userID或groupID，表示聊天的会话。
      userID: "", // 对于一对一聊天，提供另一个用户的userID
      groupID: "", // 对于群聊，提供groupID
    );
```

使用一行代码轻松导航

只需调用 `navigateToMessage` 方法即可轻松导航到消息组件：

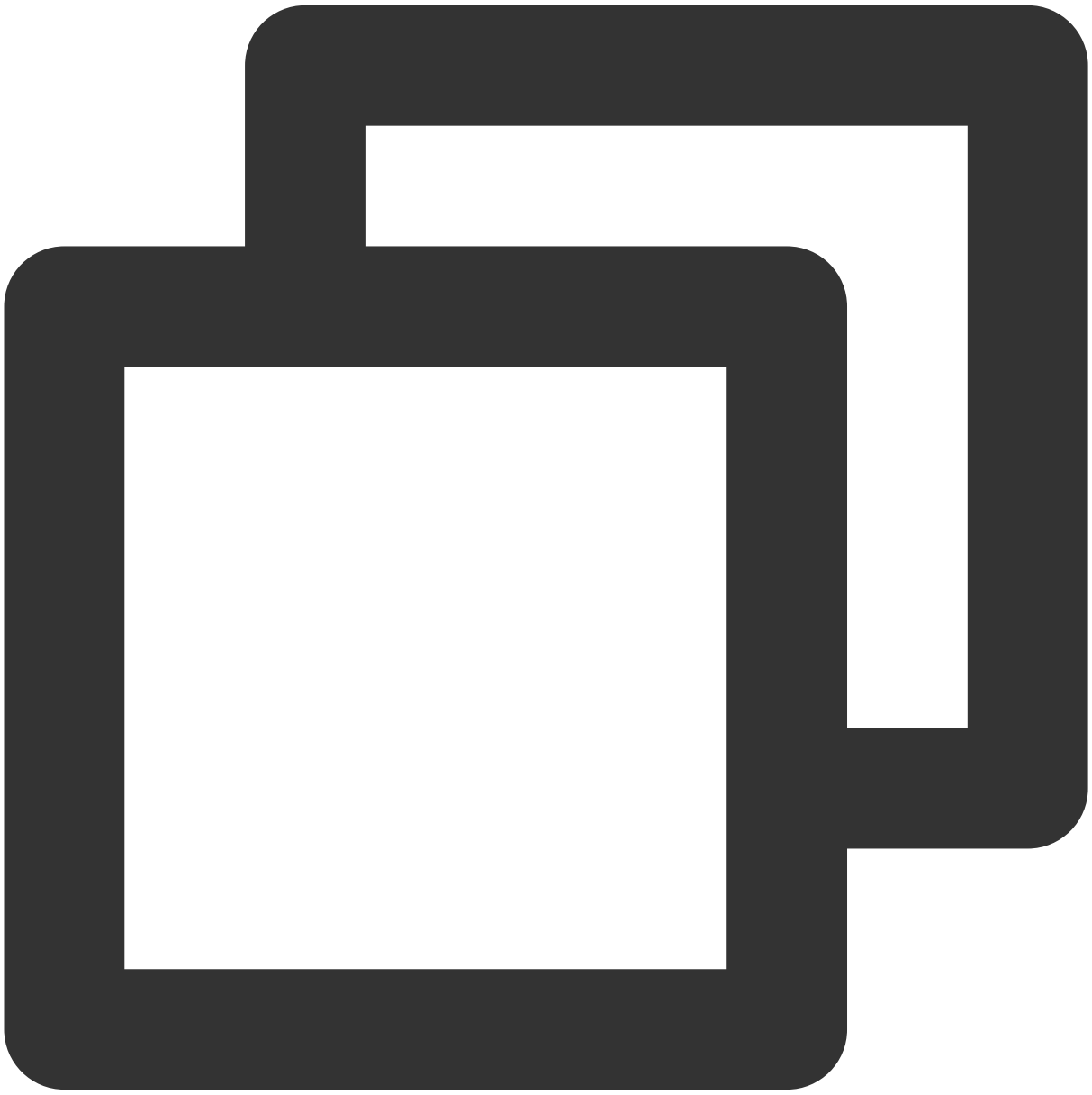


```
/// 使用上面构造的messageOptions
navigateToMessage(context: context, options: messageOptions);
```

手动导航

如果您需要手动处理导航，将组件包装在自定义页面中，或使用诸如 `TencentCloudChatMessageController` 之类的自定义功能，请先实例化一个 `TencentCloudChatMessage` 组件。

这将在将消息组件集成到应用程序时为您提供更大的控制和灵活性：



```
// 如果您需要使用控制器，请维护一个TencentCloudChatMessageController实例。
final TencentCloudChatMessageController messageController = TencentCloudChatMessage
```

```
final message = TencentCloudChatMessage(
      // 一定要提供选项。使用上面构造的messageOptions。
      options: messageOptions,

      // 如果您需要使用控制器，请提供一个控制器实例。
      controller: messageController,

      // 其他参数，如构建器，可以根据您的需求在此处全局指定或静态传入。有关详细用法，请参阅参数和
    );
```

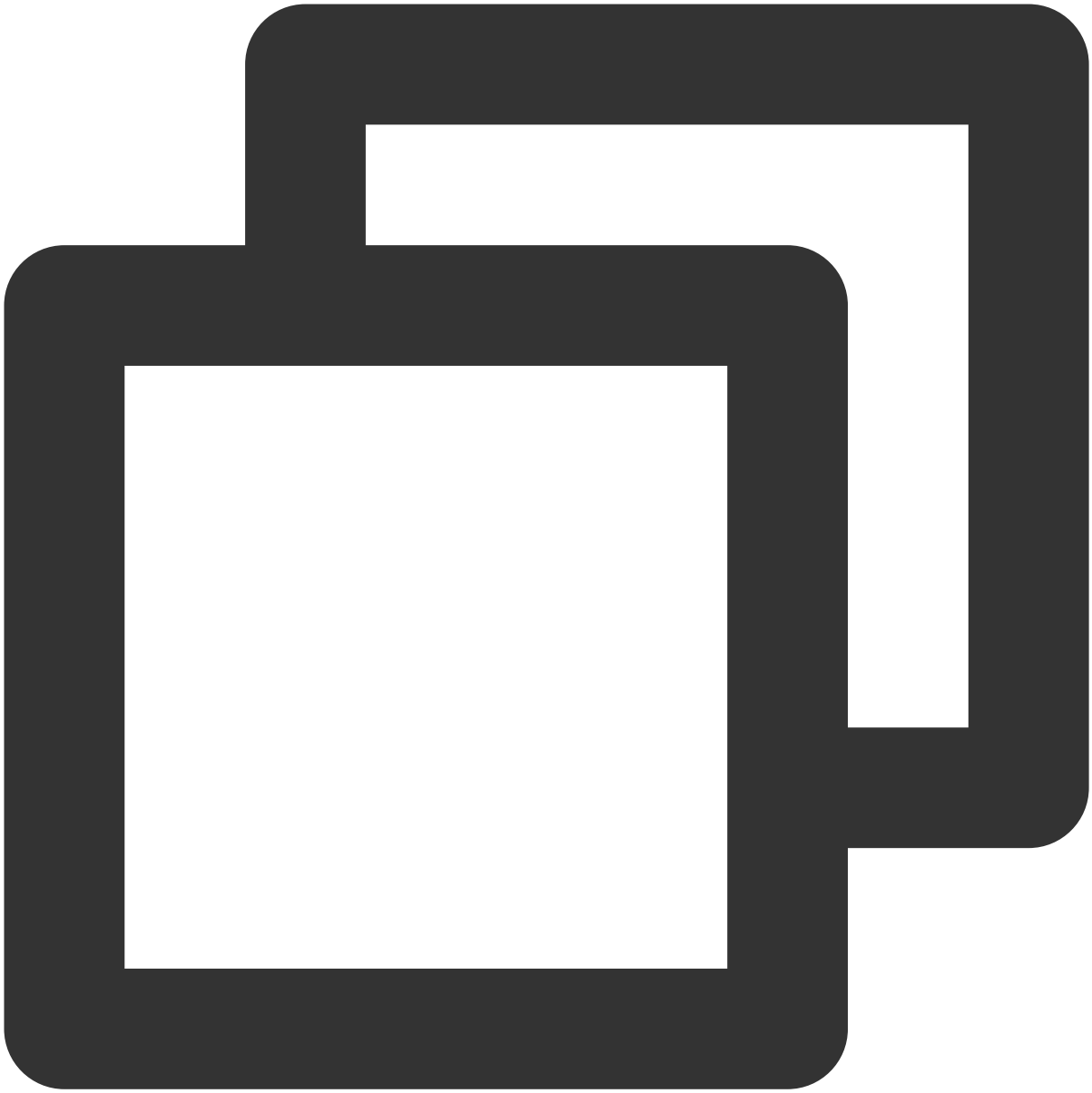
您可以将此实例化的组件放置在单独页面的 `build` 方法中，或像使用 `Navigator.push` 一样直接用于导航。如果您使用 `TencentCloudChatMessageController` ，建议将其维护在 `StatefulWidget` 的 `State` 中，使用单个实例来控制组件。有关具体用法，请参阅内部注释。

定制细节

您可以使用 `builders` 和 `config` 来定制消息组件的各个方面。这两个选项提供了不同程度的定制，使您能够根据自己的需求定制组件。

使用config

对于简单和基本的配置，您可以使用 `config` 参数。消息组件的 `config` 由 `TencentCloudChatMessageConfig` 类提供。
它包括控制各种数据类型的选项，如布尔值、整数和自定义参数。每个控制选项都是一个方法 `T Function({String? userID, String? groupID})` ，提供当前会话的 `userID` 或 `groupID` 信息。您可以使用这些字段返回适当的配置值。
这种方法允许您定义一个全局的 `TencentCloudChatMessageConfig` 类，在自动导航过程中生效，无需手动实例化一个 `TencentCloudChatMessage` 实例并传入。这是因为，在大多数情况下，不同类型的会话需要不同的配置参数。
下面是一个例子：



```
final messageConfig = TencentCloudChatMessageConfig(
    // 演示一个配置选项。
    // 是否在消息列表中显示其他用户的头像。
    showOthersAvatar: ({userID, groupID}){
      if(userID!=null&&userID.isNotEmpty){
        // 如果是一对一聊天，由于头像已经在标题中，所以不显示另一个用户的头像。
        return false;
      }
      // 如果是群聊，显示其他用户的头像。
      return true;
    }
```

```
    );
```

使用 builders

对于更深入的UI定制，您可以使用自定义构建器。消息组件的构建器由 `TencentCloudChatMessageBuilders` 类提供。
消息组件提供了一个总体的 `MessageLayoutBuilder` ，它进一步分为三个主要构建器： `MessageListViewBuilder` 用于显示消息列表， `MessageInputBuilder` 用于显示消息输入区域， `MessageHeaderBuilder` 用于显示顶部区域。它们基本上都暴露了 `String? userID` 和 `String? groupID` 参数，帮助您在自动导航过程中根据会话类型确定不同的UI样式，与 `config` 相同。
除此之外，还有更细粒度的构建器来帮助您定制更细节的内容，如消息渲染和消息布局。
此外，每个构建器都带有所需的参数和方法，使数据和逻辑层方法可以随时使用。例如，`messageInputBuilder`暴露了各种参数，如发送不同类型消息的方法、当前会话详细信息、群组成员列表等。这使您可以专注于输入区域的UI开发，并直接调用我们提供的发送消息的方法，加快您的开发过程。

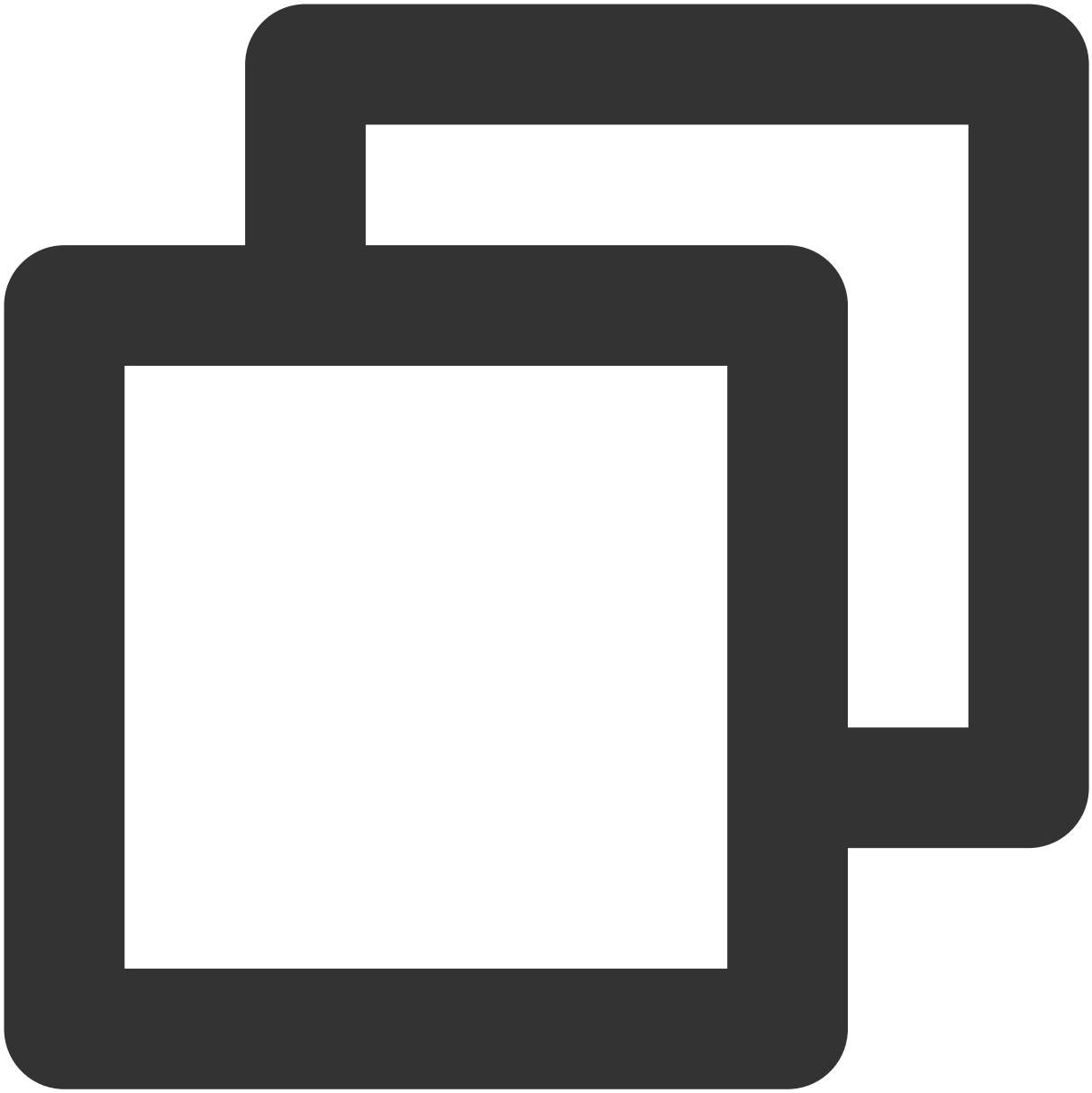
Contact

联系人组件，旨在为您的聊天应用程序提供多功能的联系人列表。
联系人组件提供一个联系人列表，显示所有已添加联系人，按其姓名首字母排序。它还支持显示其他信息，如已加入的群组列表、被屏蔽用户列表、请求将您添加为联系人的用户和群组消息通知。
当与tencent_cloud_chat_message组件一起使用时，联系人组件可以在移动和桌面环境下点击联系人或群组时自动导航到相应的消息聊天页面。这种无缝集成确保了您的用户获得顺畅有序的聊天体验。

开始使用

导入和声明

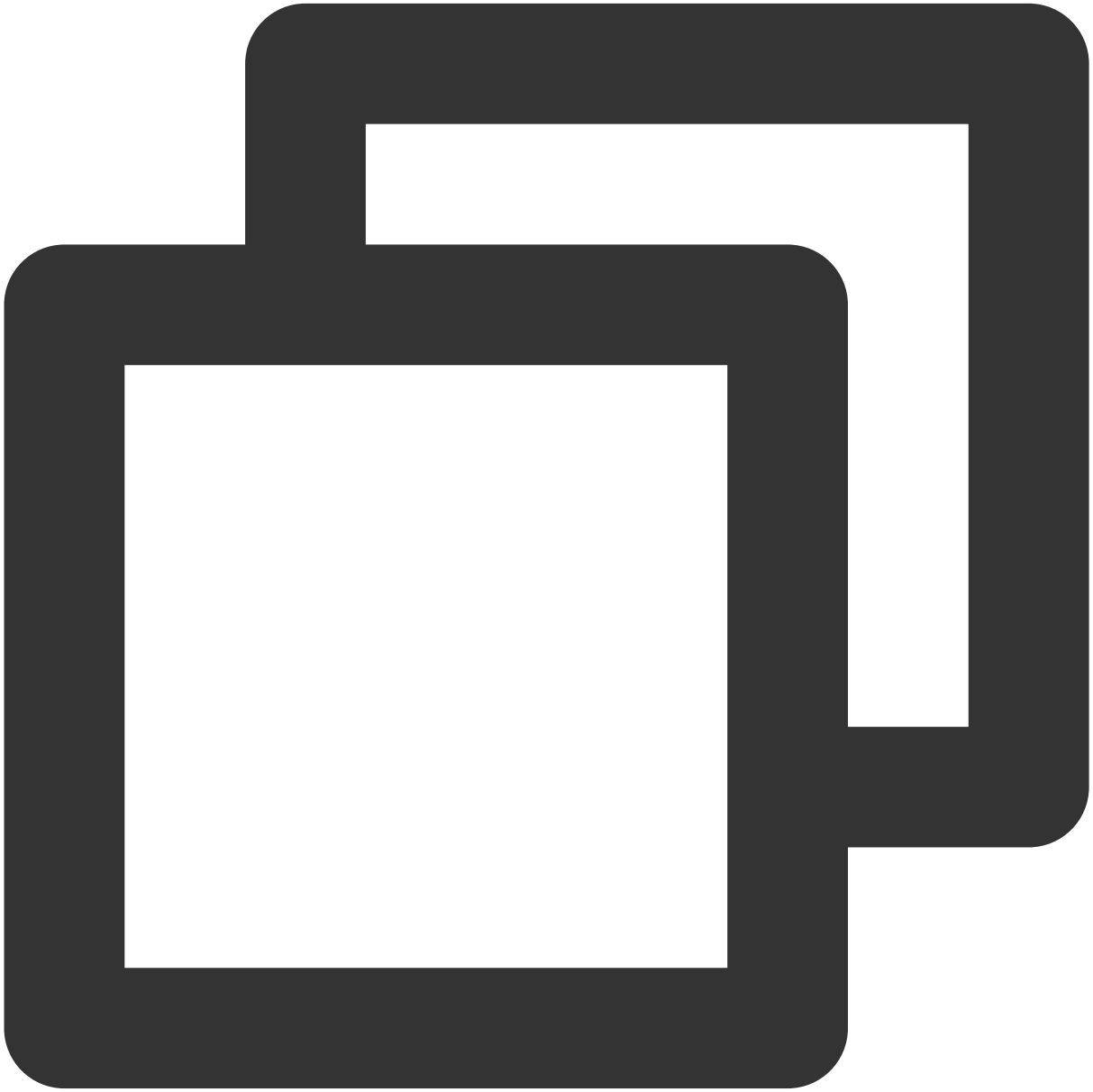
首先，将tencent_cloud_chat_conversation UI模块添加到您的项目中。
安装完成后，您需要在 `TencentCloudChat.controller.initUIKit` 方法的 `components` 的 `usedComponentsRegister` 参数中注册此UI组件。以下是一个例子：



```
await TencentCloudChat.controller.initUIKit(
  components: TencentCloudChatInitComponentsRelated(
    usedComponentsRegister: [
      TencentCloudChatContactManager.register, /// 添加这一行
      /// ...
    ],
  /// ...
  ),
  /// ...
);
```

实例化和使用组件

使用联系人组件非常简单。只需实例化一个 `TencentCloudChatContact` 实例，并在所需页面上渲染它。
默认情况下，组件将自动获取并显示所有联系人信息，无需任何其他参数。
您可以在要显示联系人列表的页面的 `build` 方法中使用此实例，以及加入群组列表、被屏蔽用户列表、请求将您添加为联系人的用户和群组消息通知的入口。



```
@override
Widget build(BuildContext context) {
  return const TencentCloudChatContact();
}
```

只需几行代码，您就可以轻松地将联系人组件集成到您的聊天应用程序中，供用户与之互动。

定制细节

使用 config

对于简单和基本的配置，您可以使用 `config` 参数。联系人组件的 `config` 由 `TencentCloudChatContactConfig` 类提供。它包括控制各种数据类型的选项，如布尔值、整数和自定义参数。

使用 builders

对于更深入的 UI 定制，您可以使用自定义构建器。联系人组件的构建器由 `TencentCloudChatContactBuilders` 类提供。

User Profile

说明：

此模块化 UI 包尚在内测中，如需使用，请联系我们获取源码。谢谢！

用户资料组件旨在为您的聊天应用程序提供一个详细的用户资料页面。

该组件不仅显示用户信息，如头像、昵称和其他基本详细信息，还提供了广泛的关系管理和更多功能。

用户可以为他们的联系人设置备注、添加或删除联系人、屏蔽用户，并执行各种其他操作。该组件还有助于配置会话设置，包括固定会话和管理消息通知。

与 tencent_cloud_chat_message 组件集成时，用户资料组件确保在消息和用户资料页面之间无缝导航，无需手动实现导航。

此外，与 tencent_calls_uikit 搭配使用时，它允许用户直接从用户资料页面发起语音和视频通话，进一步增强整体通信体验。

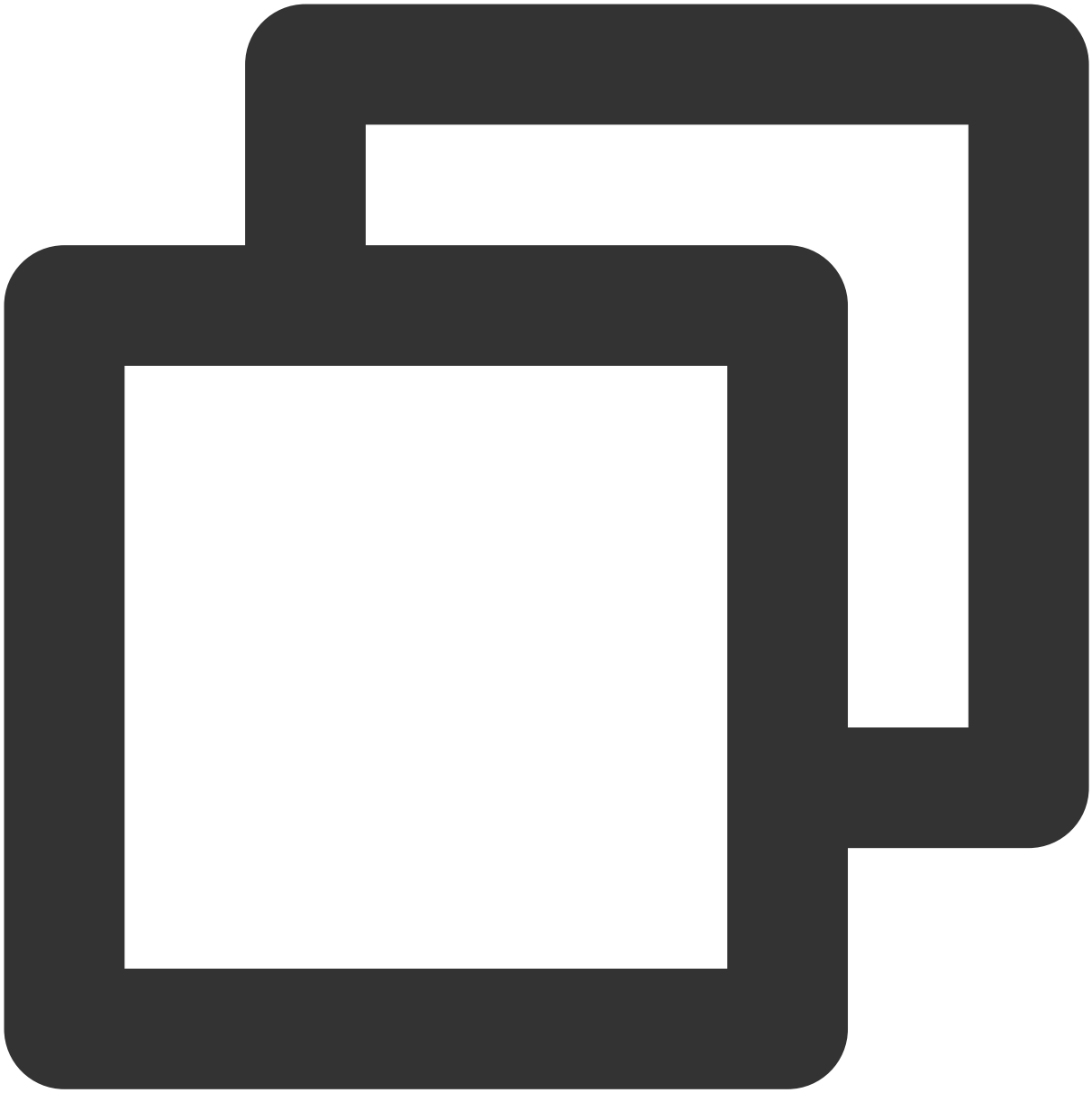
凭借其直观的设计和强大的功能，用户资料组件满足了广泛的用户需求和偏好，确保了顺畅而引人入胜的用户体验。

开始使用

导入和声明

首先，将 tencent_cloud_chat_user_profile UI 模块添加到您的项目中。

安装完成后，您需要在 `TencentCloudChat.controller.initUIKit` 方法的 `components` 的 `usedComponentsRegister` 参数中注册此UI组件。以下是一个例子：

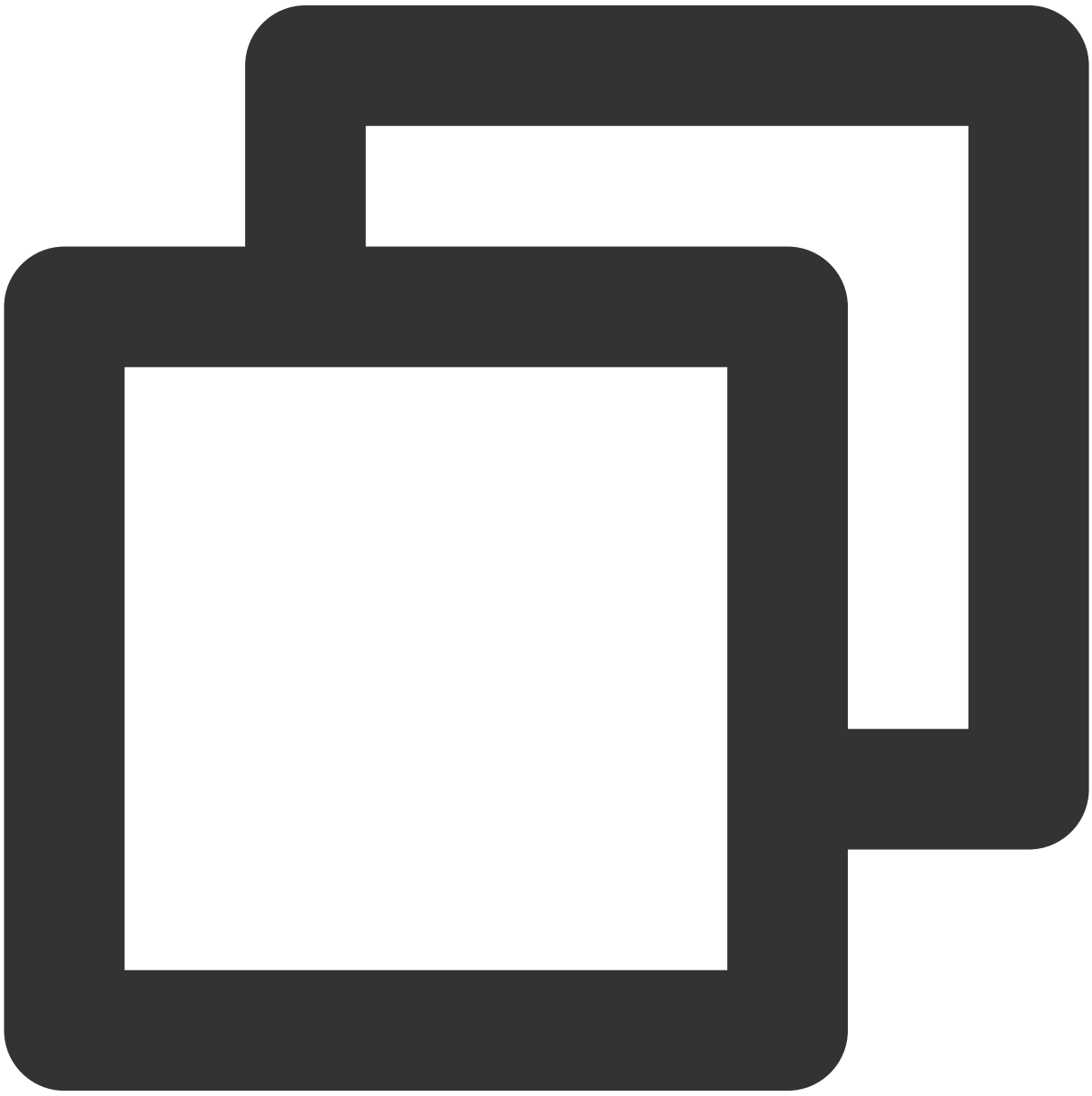


```
await TencentCloudChat.controller.initUIKit(
  components: TencentCloudChatInitComponentsRelated(
    usedComponentsRegister: [
      TencentCloudChatUserProfileManager.register, /// 添加这一行
      /// ...
    ],
  /// ...
  ),
  /// ...
);
```

如果您的项目包含模块化组件，例如 tencent_cloud_chat_message 用于显示会话，它们将自动导航到此用户资料组件。
如果仅需要从这些内置组件而不是从您的自定义页面进行导航，则 `用户资料` 组件集成仅需此单步完成。UIKit 在内部处理导航转换，无需手动编码。
对于需要从自定义页面进行导航的项目，请参阅以下步骤。

导航到用户资料组件

在导航之前，准备一个 `TencentCloudChatUserProfileOptions` 实例，以指定目标用户：

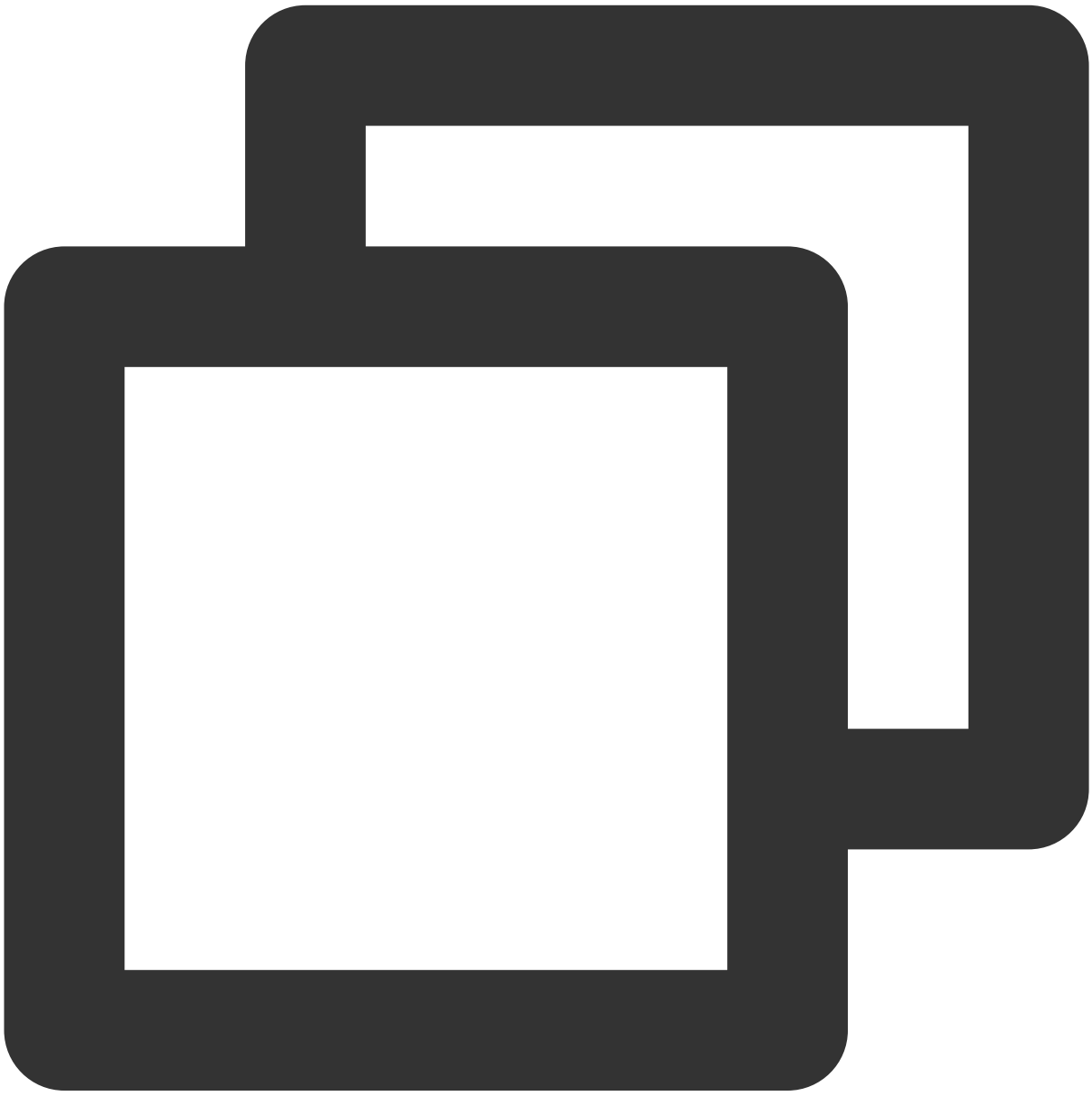


```
final userProfileOptions = TencentCloudChatUserProfileOptions(
```

```
        userID: "", // 提供一个用户的 userID
    );
```

使用一行代码轻松导航

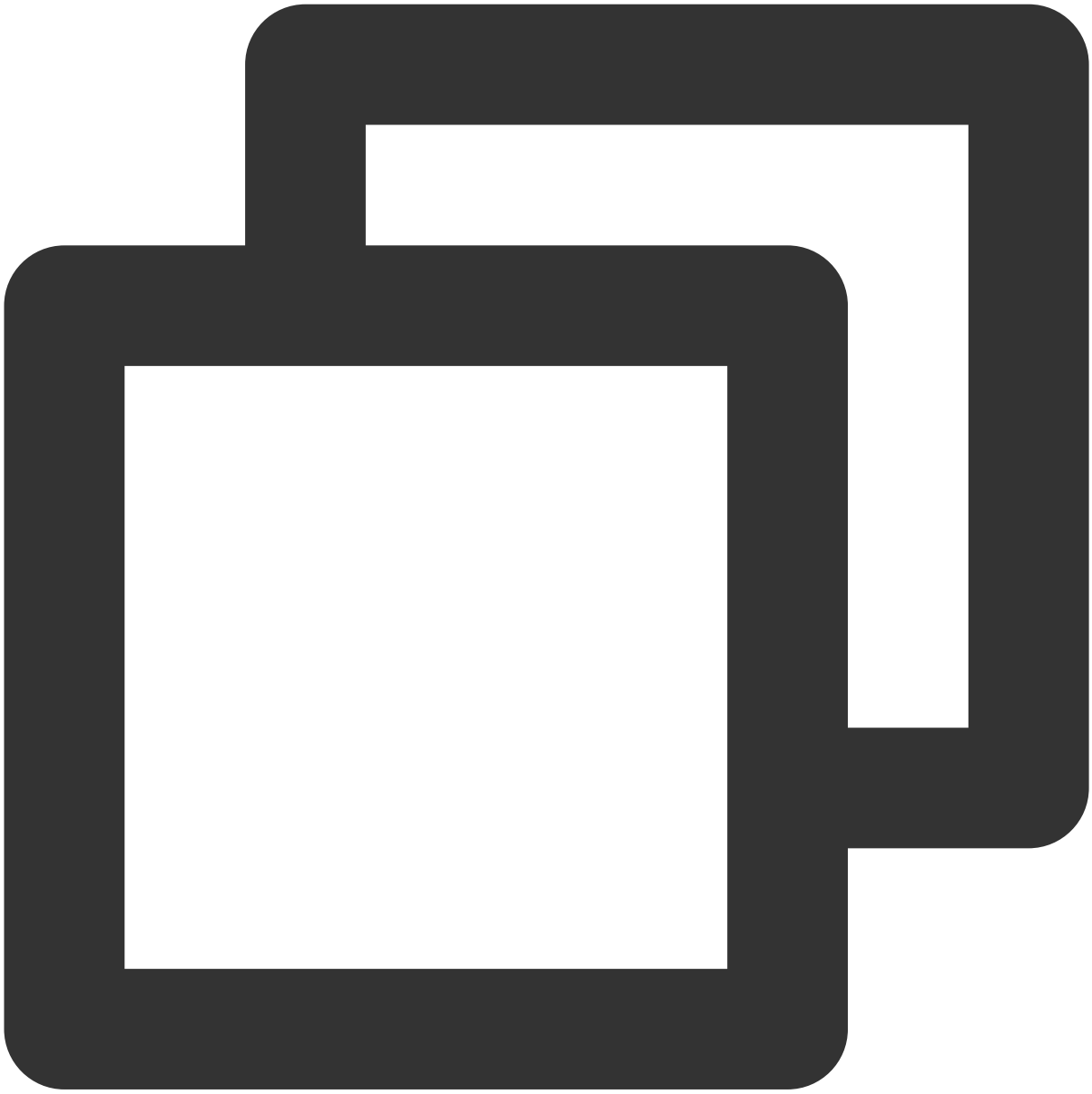
只需调用 `navigateToUserProfile` 方法即可轻松导航到用户资料组件：



```
/// 使用上面构造的userProfileOptions
navigateToUserProfile(context: context, options: userProfileOptions);
```

手动导航

如果您需要手动处理导航，或将组件包装在自定义页面中，请先实例化一个 `TencentCloudChatUserProfile` 组件。
这将在将用户资料组件集成到应用程序时为您提供更大的控制和灵活性：



```
final userProfile = TencentCloudChatUserProfile(
  // 一定要提供选项。使用上面构造的userProfileOptions。
  options: userProfileOptions,

  // 其他参数，如构建器，可以根据您的需求在此处全局指定或静态传入。有关详细用法，请参阅参数和
);
```

您可以将此实例化的组件放置在单独页面的 `build` 方法中，或像使用 `Navigator.push` 一样直接用于导航。

定制细节

使用 config

对于简单和基本的配置，您可以使用 `config` 参数。联系人组件的 `config` 由 `TencentCloudChatUserProfileConfig` 类提供。它包括控制各种数据类型的选项，如布尔值、整数和自定义参数。

使用 builders

对于更深入的UI定制，您可以使用自定义构建器。联系人组件的构建器由 `TencentCloudChatUserProfileBuilders` 类提供。

Group Profile

说明：

此模块化 UI 包尚在内测中，如需使用，请联系我们获取源码。谢谢！

这个组件旨在通过详细且互动的群组资料页面，增强您的聊天应用程序。

群组资料组件提供了群组信息的全面视图，如群组头像、群组 ID、成员列表、群组类型和群组公告等功能。

除了显示群组信息外，此组件还有助于执行各种群组管理任务。对于群组所有者或管理员，它提供了管理成员列表（包括邀请或删除成员）、编辑群组公告和执行其他群组管理操作（如群组静音）等功能。

此外，它还允许用户管理与群组相关的会话设置，包括但不限于固定会话、管理消息通知和退出群组聊天。

当与tencent_cloud_chat_message组件一起使用时，群组资料组件确保在消息和群组资料页面之间无缝导航，无需手动实现导航。

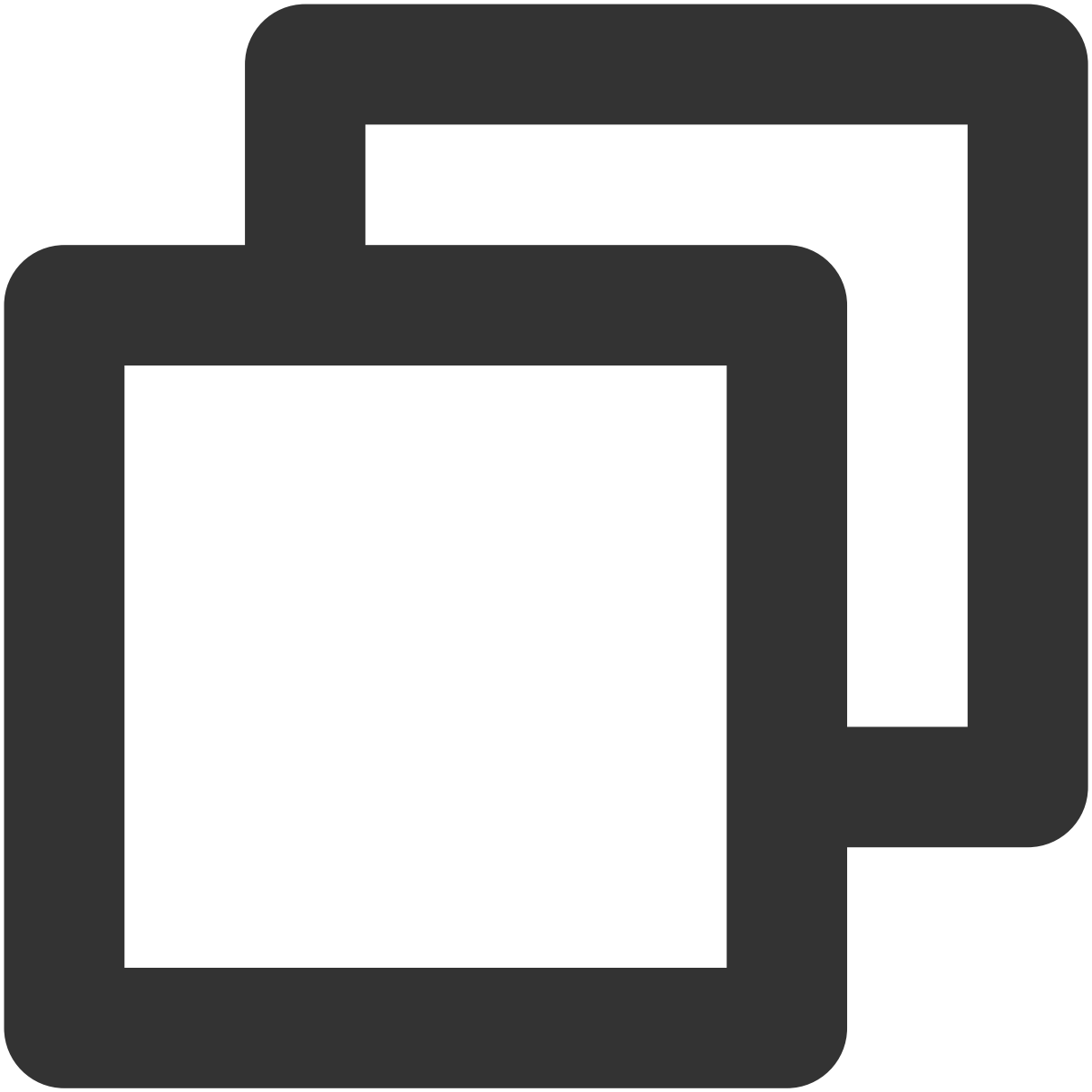
凭借其直观的设计和强大的功能，群组资料组件满足了广泛的用户需求和偏好，确保了顺畅而引人入胜的群组聊天体验。

开始使用

导入和声明

首先，将 tencent_cloud_chat_group_profile UI 模块添加到您的项目中。

安装完成后，您需要在 `TencentCloudChat.controller.initUIKit` 方法的 `components` 的 `usedComponentsRegister` 参数中注册此 UI 组件。以下是一个例子：

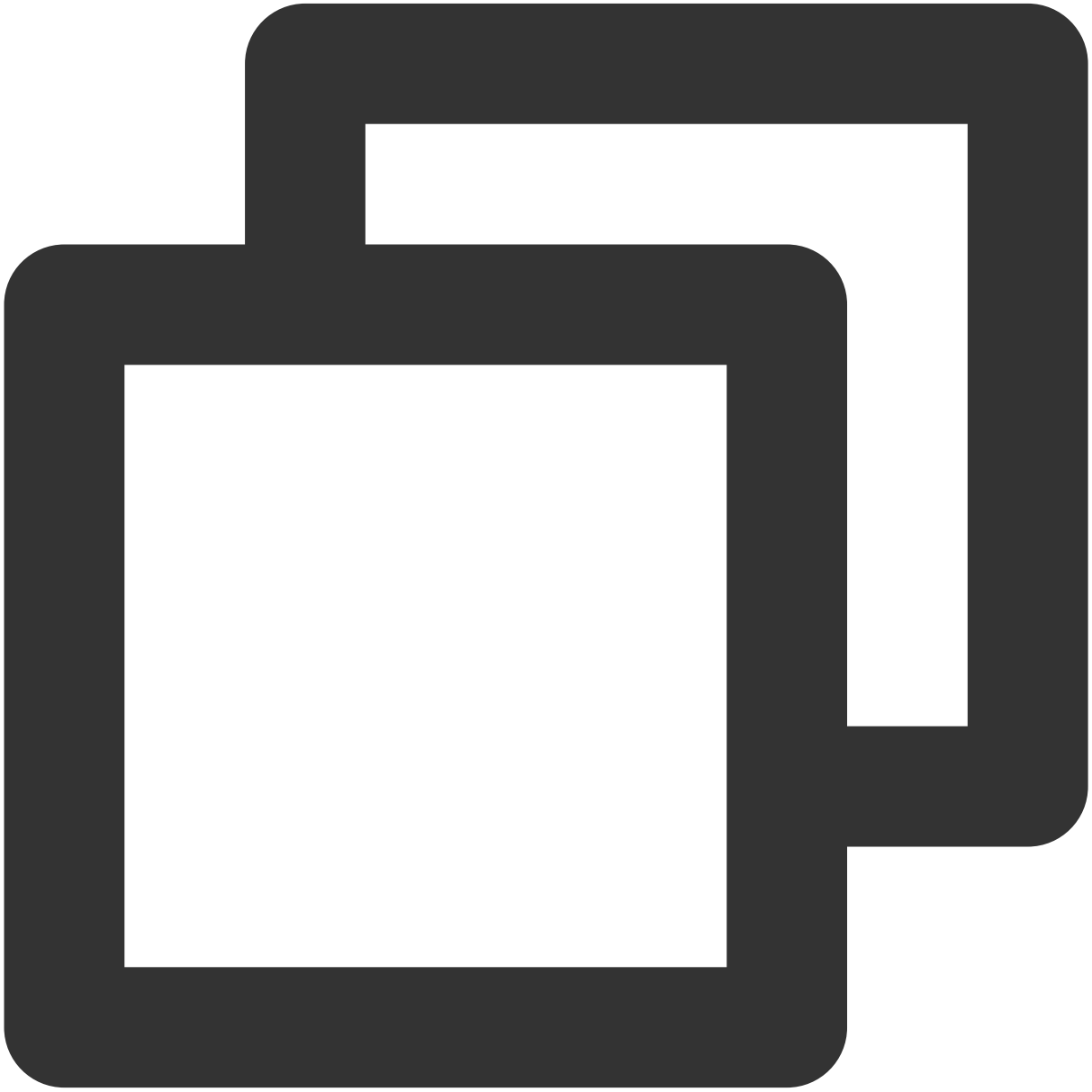


```
await TencentCloudChat.controller.initUIKit(
  components: TencentCloudChatInitComponentsRelated(
    usedComponentsRegister: [
      TencentCloudChatGroupProfileManager.register, /// 添加这一行
      /// ...
    ],
  /// ...
  ),
  /// ...
);
```

如果您的项目包含模块化组件，如tencent_cloud_chat_message用于显示会话，它们将自动导航到此群组资料组件。
如果仅需要从这些内置组件而不是从您的自定义页面进行导航，则 `群组资料` 组件集成仅需此单步完成。UIKit 在内部处理导航转换，无需手动编码。
对于需要从自定义页面进行导航的项目，请参阅以下步骤。

导航到群组资料组件

在导航之前，准备一个 `TencentCloudChatGroupProfileOptions` 实例，以指定目标用户：

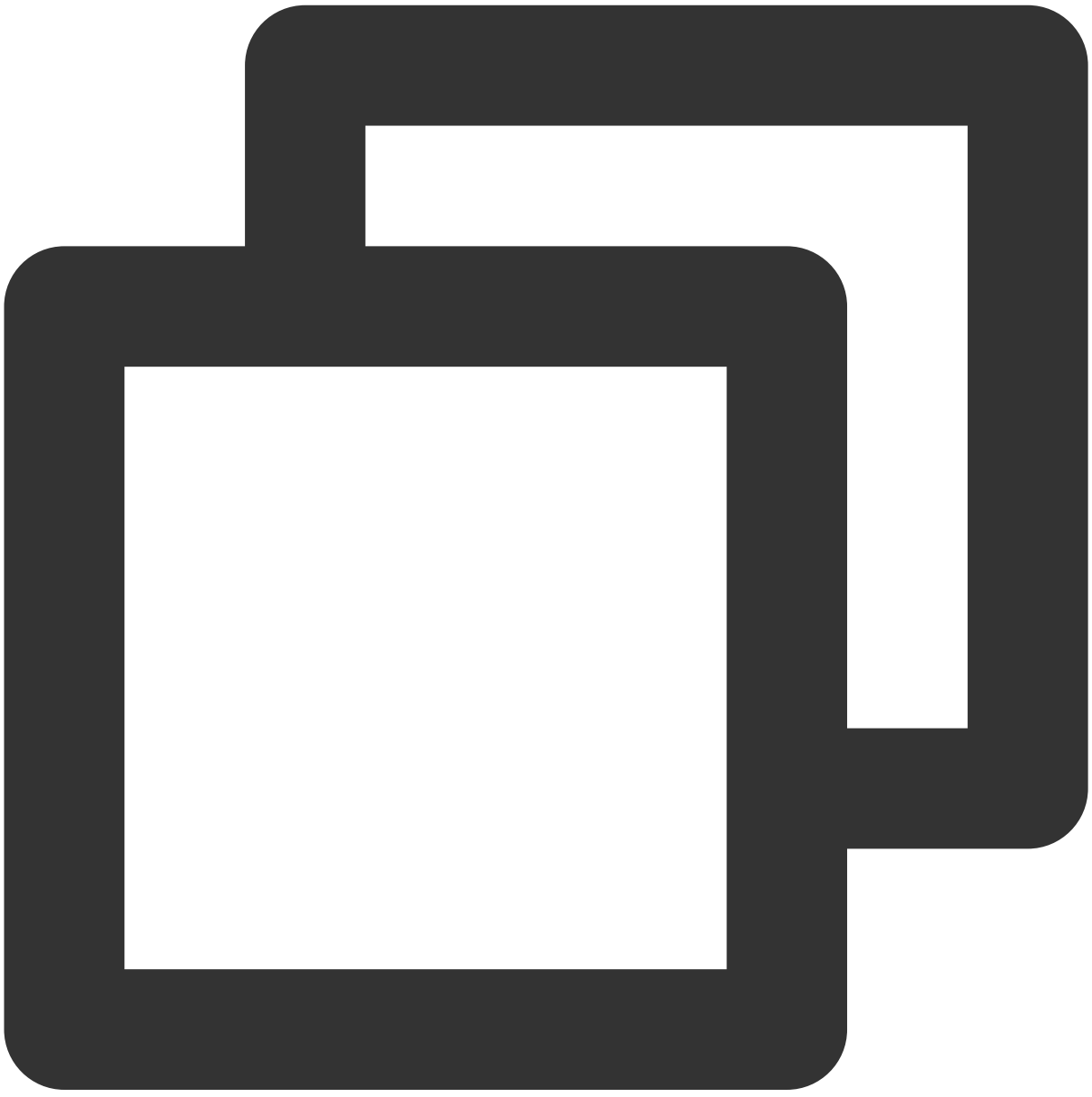


```
final groupProfileOptions = TencentCloudChatGroupProfileOptions(
```

```
  groupID: "", // 提供群组ID
    );
```

使用一行代码轻松导航

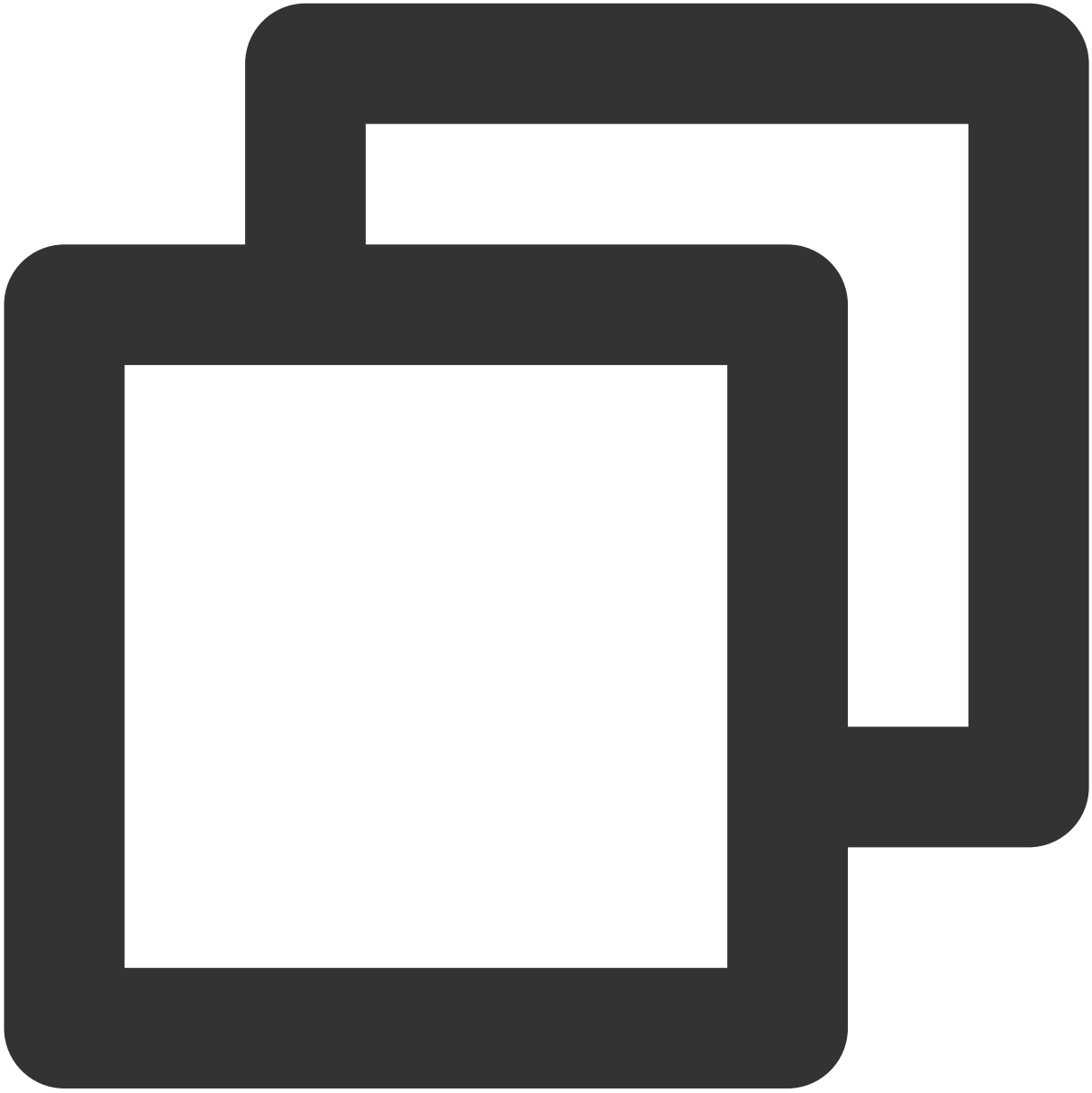
只需调用 `navigateToUserProfile` 方法即可轻松导航到群组资料组件：



```
/// 使用上面构造的userProfileOptions
navigateToGroupProfile(context: context, options: groupProfileOptions);
```

手动导航

如果您需要手动处理导航，或将组件包装在自定义页面中，请先实例化一个 `TencentCloudChatGroupProfile` 组件。
这将在将群组资料组件集成到应用程序时为您提供更大的控制和灵活性：



```
final groupProfile = TencentCloudChatGroupProfile(
  // 一定要提供选项。使用上面构造的groupProfileOptions。
  options: groupProfileOptions,

  // 其他参数，如构建器，可以根据您的需求在此处全局指定或静态传入。有关详细用法，请参阅参数和
);
```

您可以将此实例化的组件放置在单独页面的 `build` 方法中，或像使用 `Navigator.push` 一样直接用于导航。

定制细节

使用 config

对于简单和基本的配置，您可以使用 `config` 参数。群组资料组件的 `config` 由 `TencentCloudChatGroupProfileConfig` 类提供。它包括控制各种数据类型的选项，如布尔值、整数和自定义参数。

使用 builders

对于更深入的UI定制，您可以使用自定义构建器。群组资料组件的构建器由 `TencentCloudChatGroupProfileBuilders` 类提供。

结语

我们希望这份文档能帮助您理解我们新的 Flutter IM UIKit 的强大和灵活性。凭借其模块化设计和广泛的可定制选项，它为构建聊天应用程序提供了全面的解决方案。其高级功能，如会话管理、消息处理和内置导航转换，使其成为开发人员的强大工具。我们期待看到您将用我们的 UIKit 创建的惊人应用程序。如果您有任何问题或需要进一步的信息，请随时联系我们。

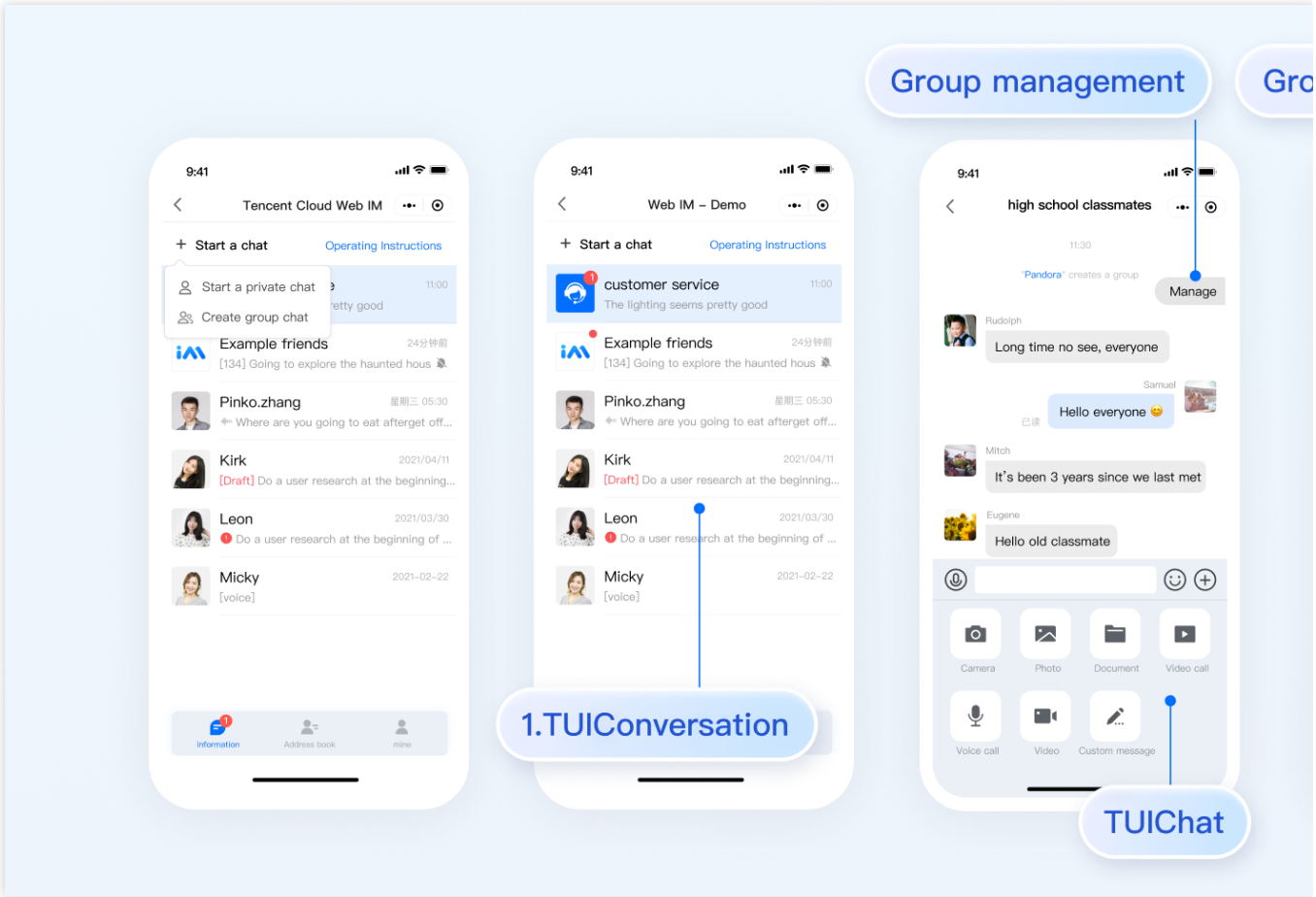
版权所有：腾讯云计算（北京）有限责任公司

uniapp

最近更新时间：2024-02-01 11:12:29

chat-uikit-uniapp 介绍

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



支持平台

Android
iOS
微信小程序

H5

开发环境要求

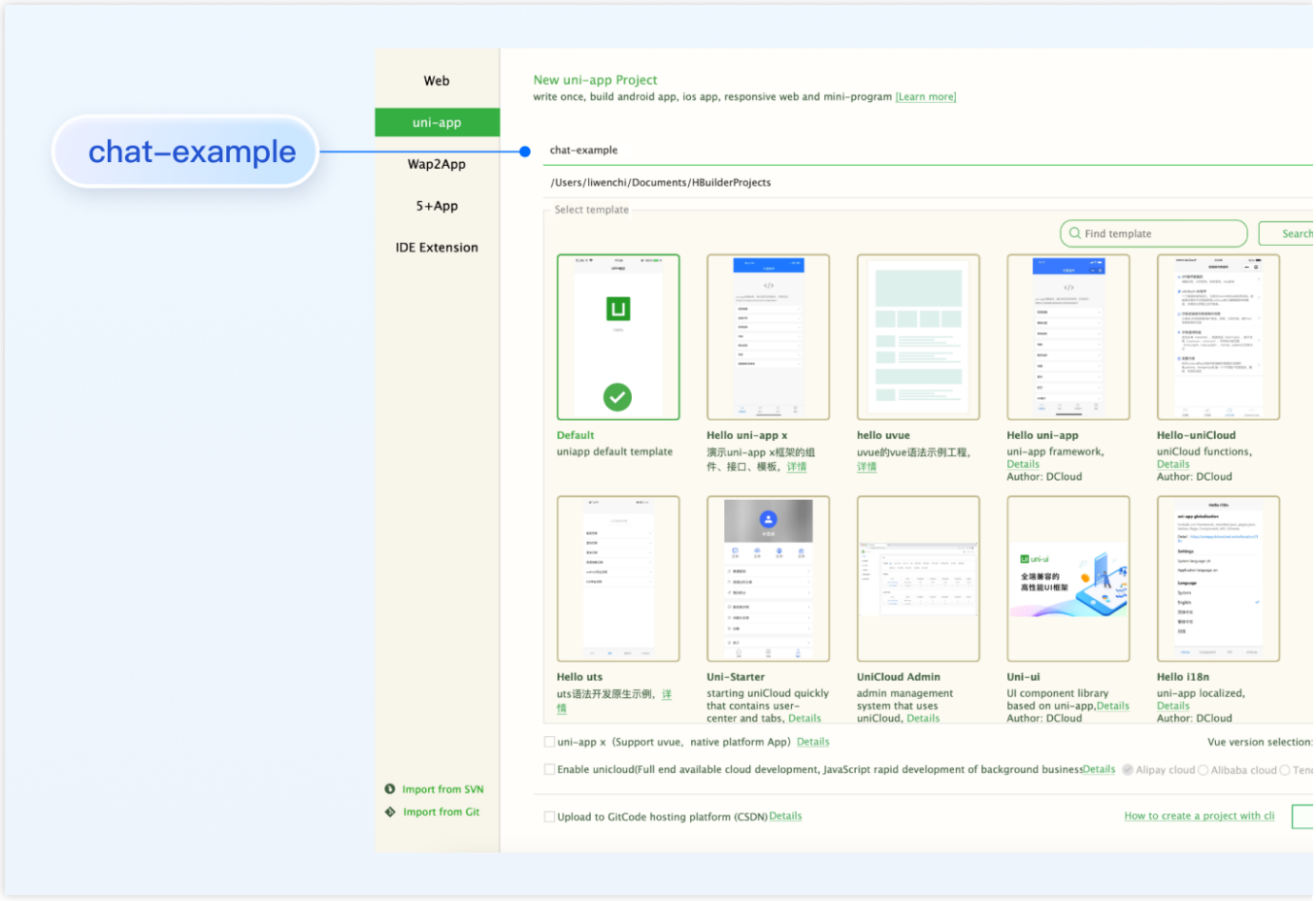
HBuilderX （HBuilderX 版本 >= 3.8.4.20230531）或者升级到新版本
Vue2 / Vue3
sass（sass-loader 版本 ≤ 10.1.1）
node（12.13.0 ≤ node 版本 ≤ 17.0.0, 推荐使用 Node.js 官方 LTS 版本 16.17.0）
npm（版本请与 node 版本匹配）

TUIKit 源码集成

完成以下步骤即可发送您的第一条消息。

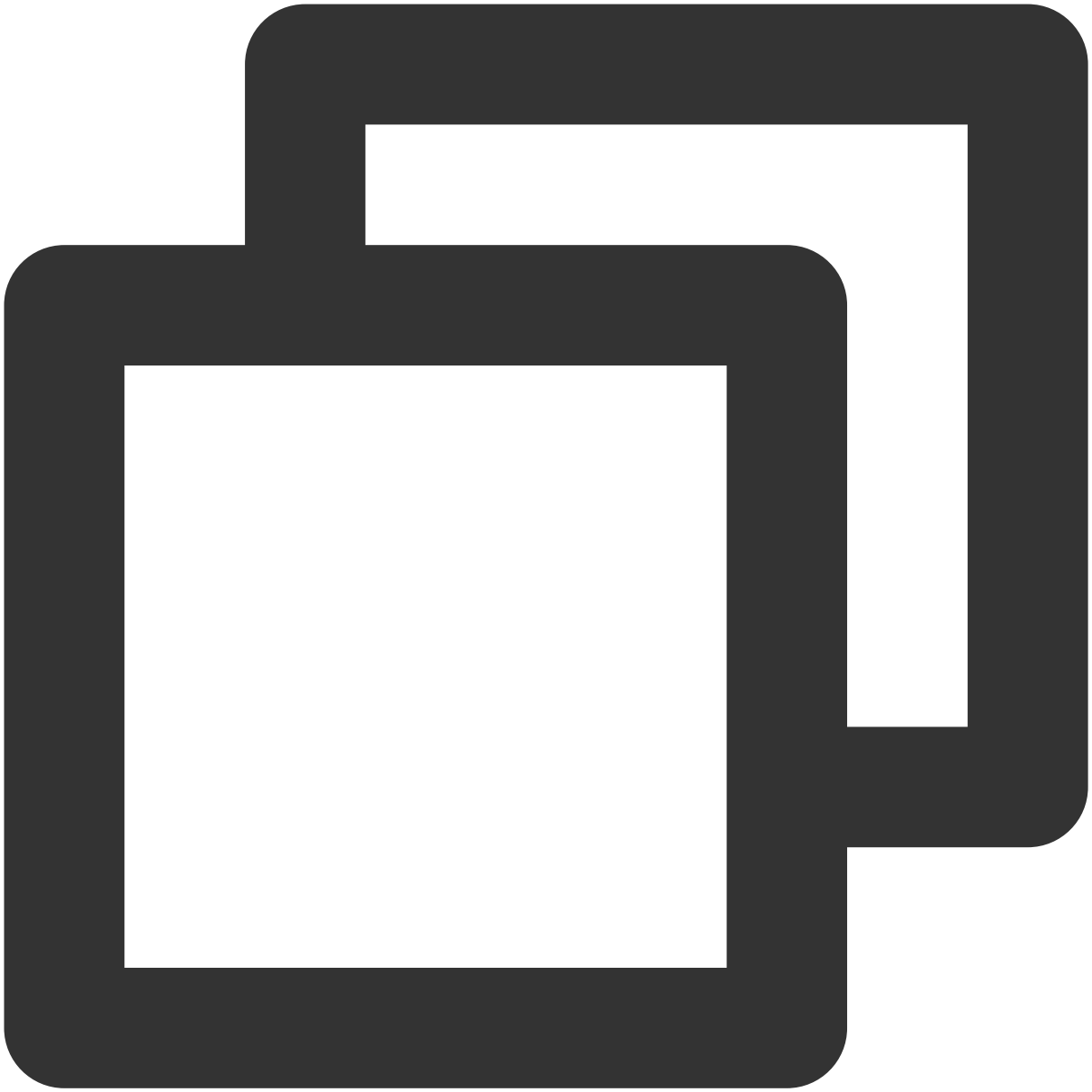
步骤1：创建项目 （已有项目可忽略）

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



步骤2：下载 TUIKit 组件

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



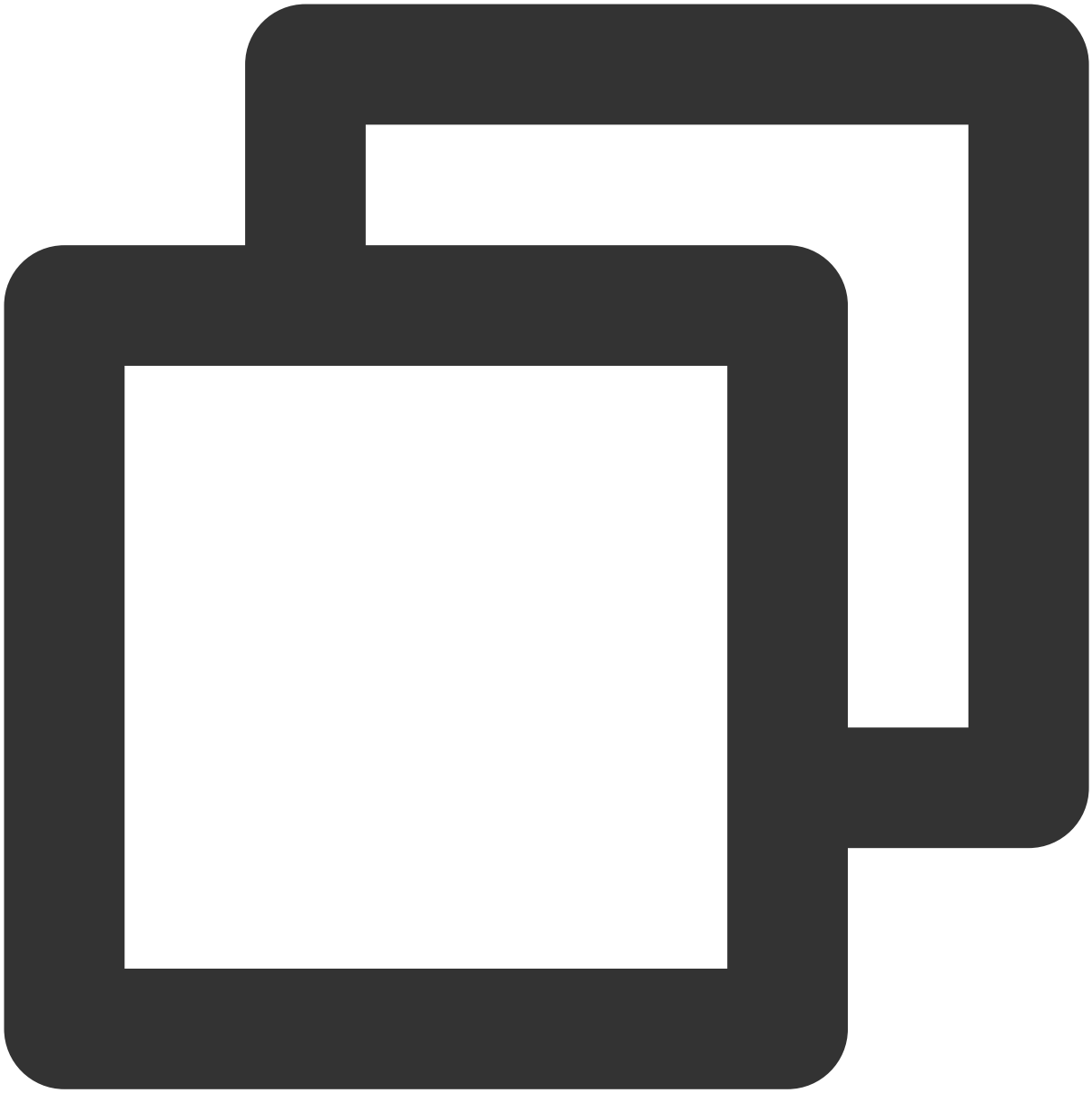
```
npm init -y
```

下载 TUIKit 并拷贝至源码中：

macOS 端

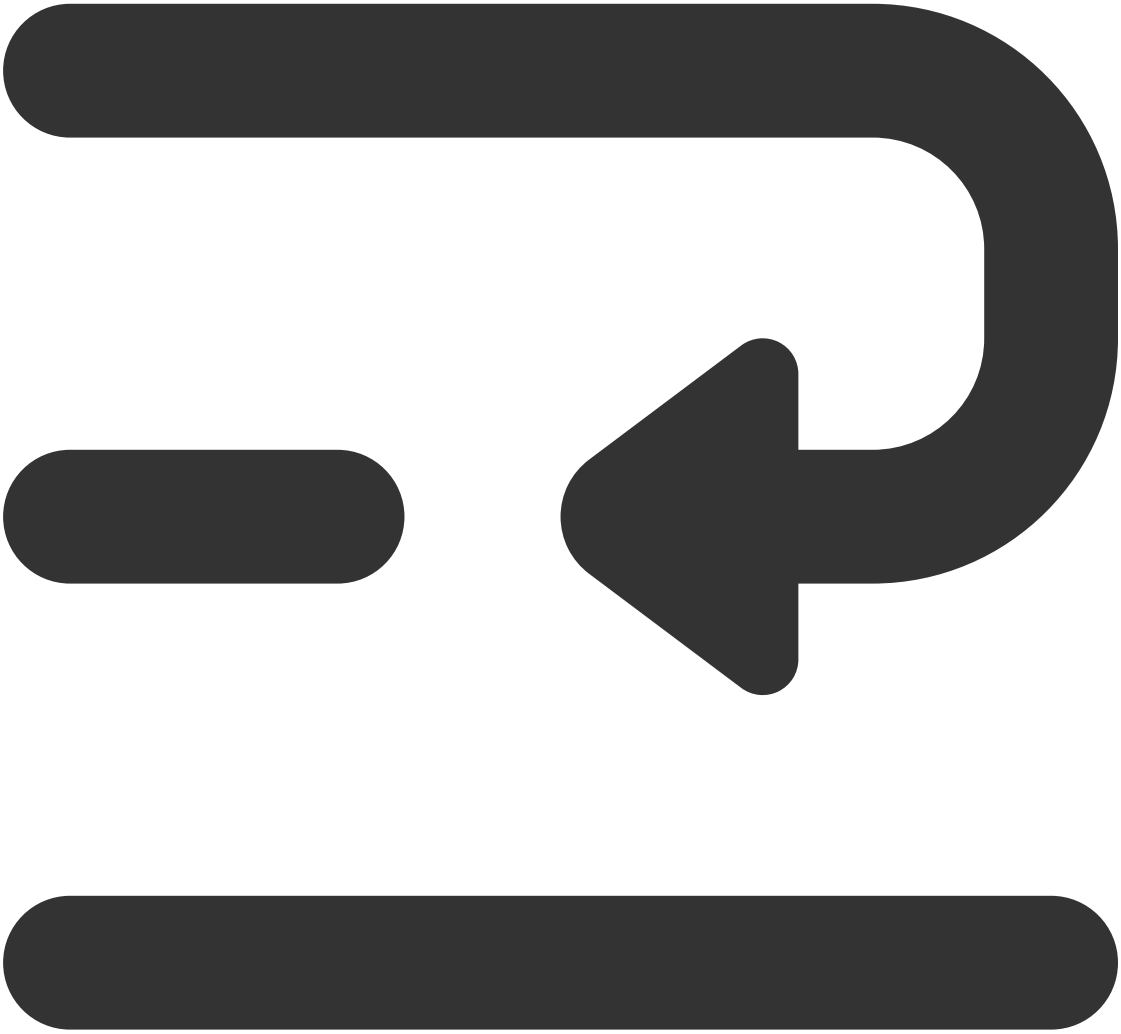
Windows 端

通过 npm 方式下载 TUIKit 组件：

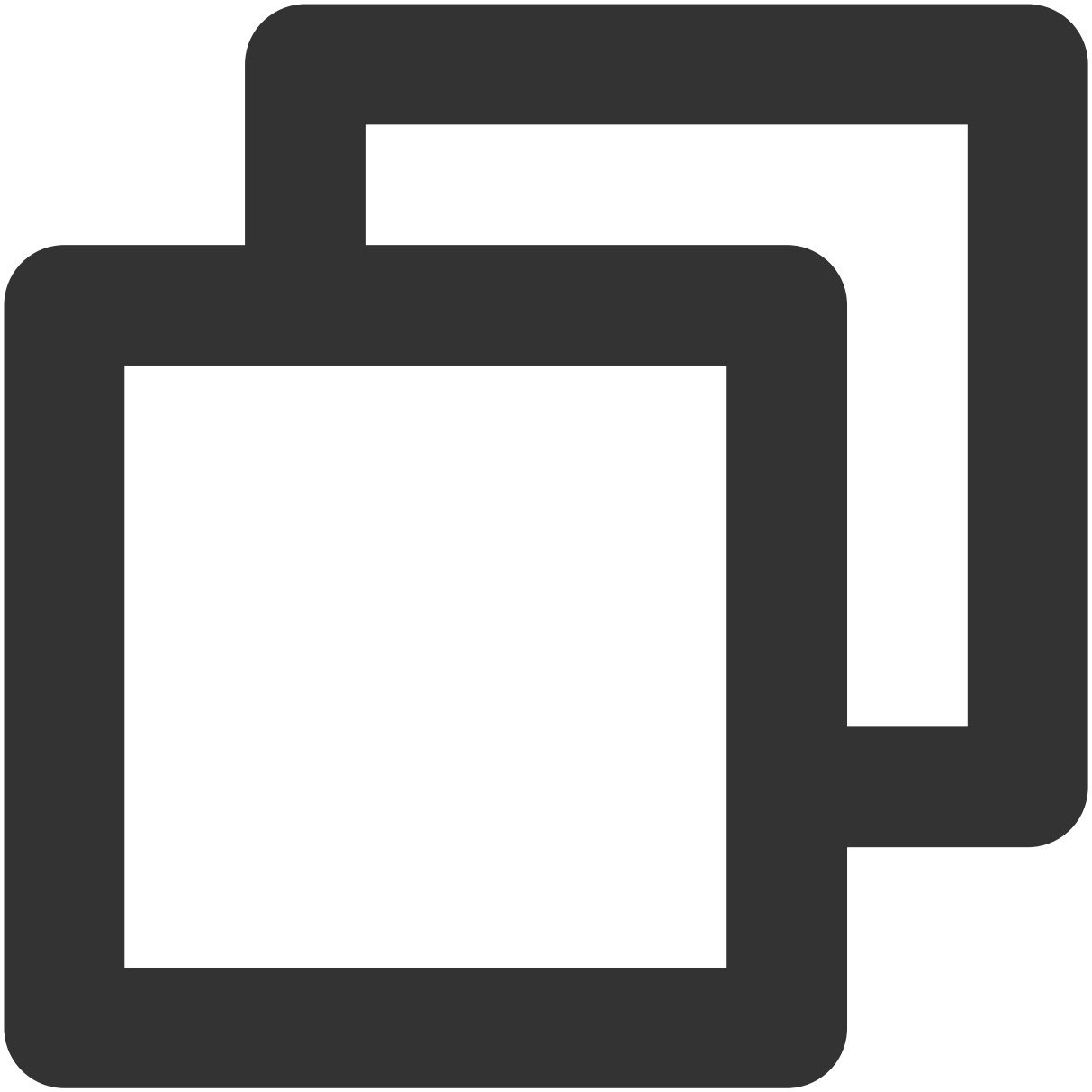


```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

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

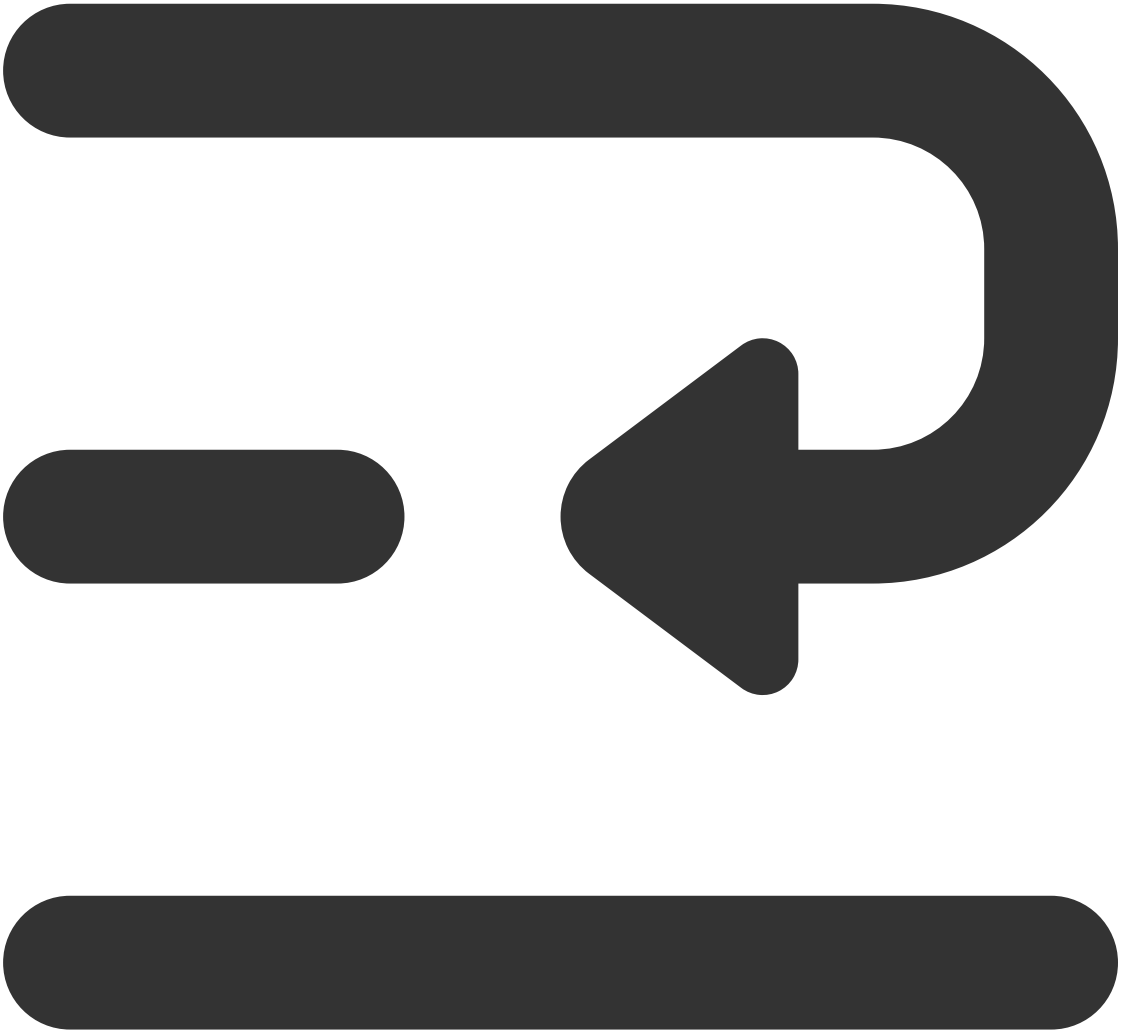


版权所有：腾讯云计算（北京）有限责任公司

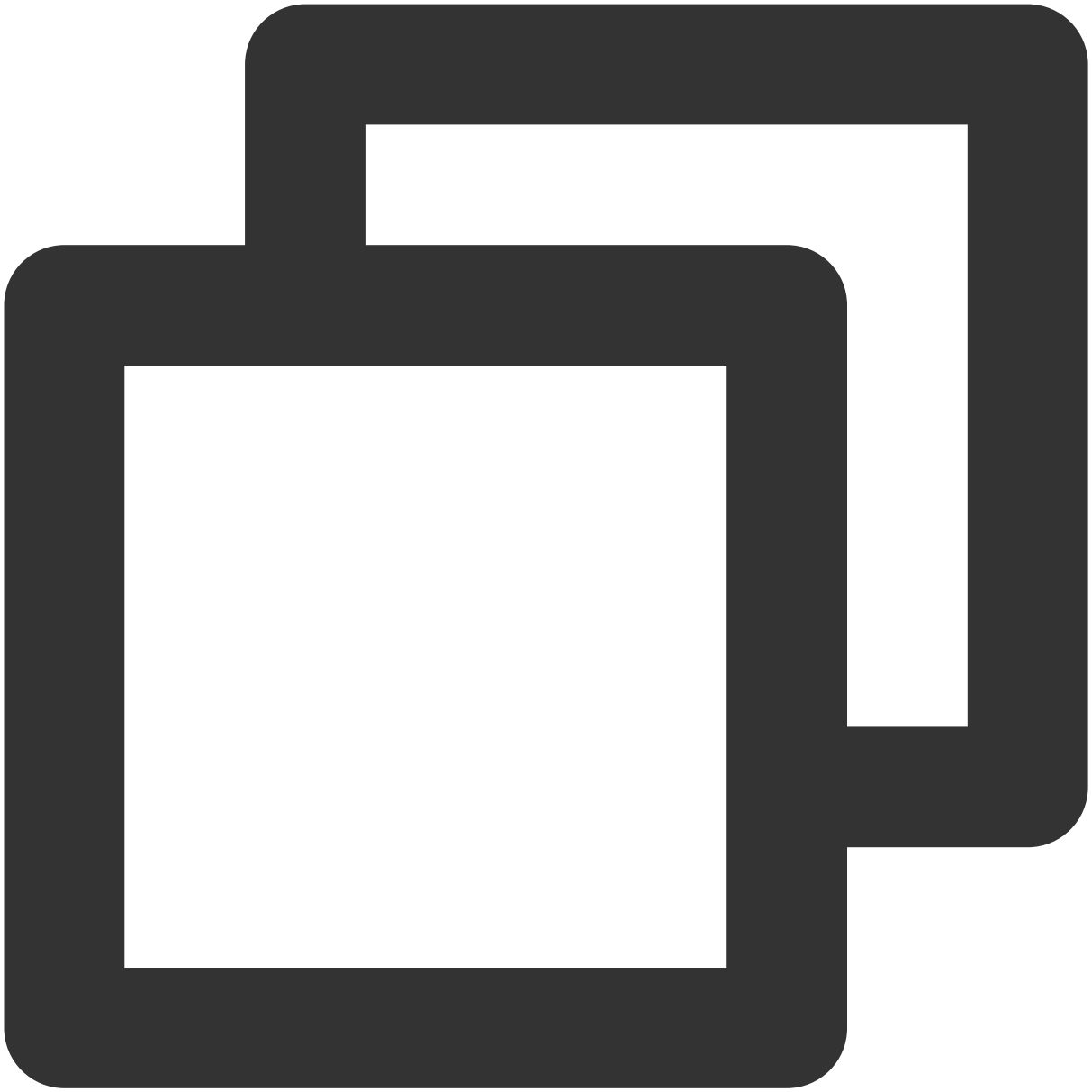


```
mkdir -p ./TUIKit && rsync -av --exclude={'node_modules','package.json','excluded-l
```

版权所有：腾讯云计算（北京）有限责任公司

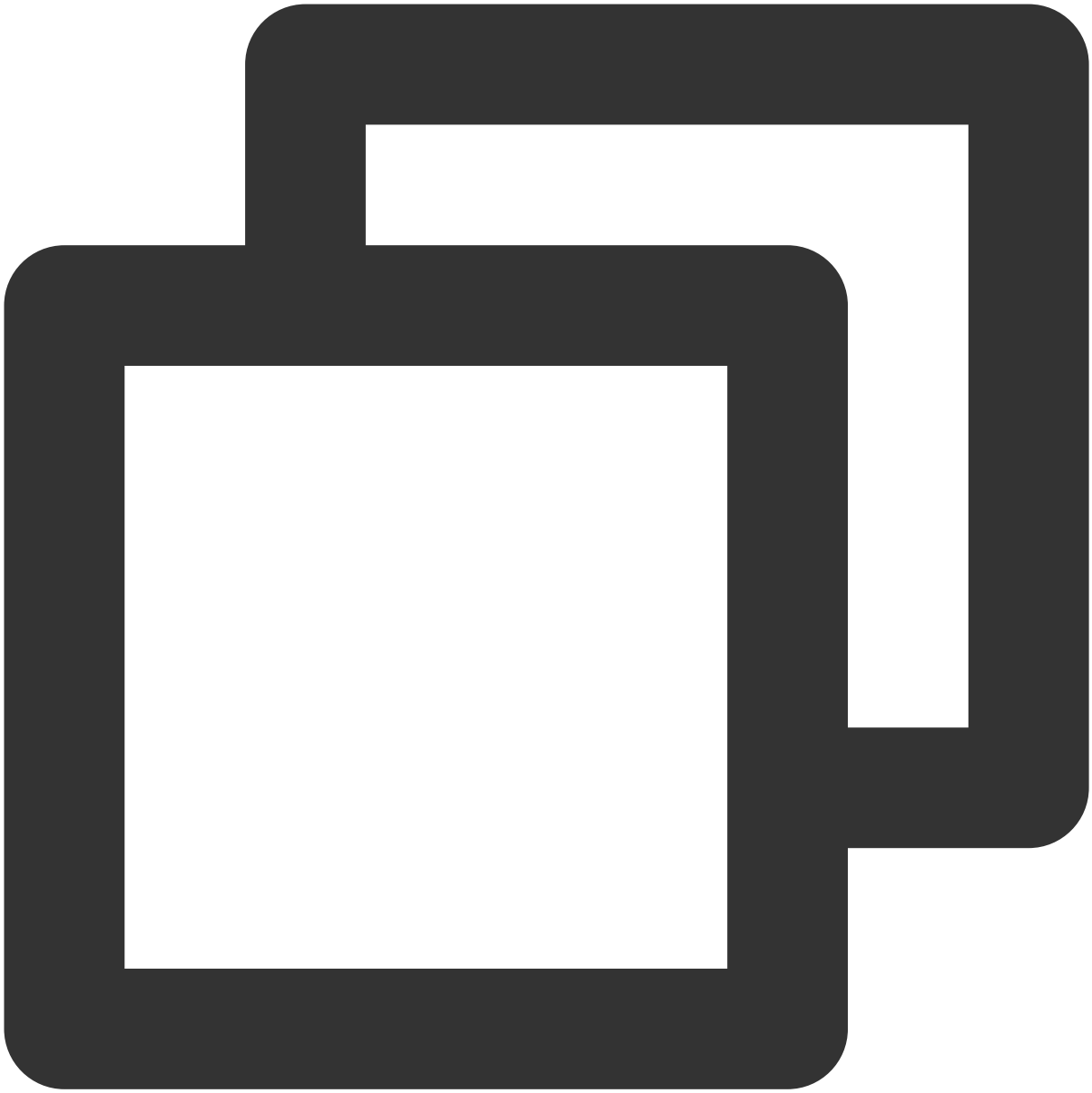


版权所有：腾讯云计算（北京）有限责任公司



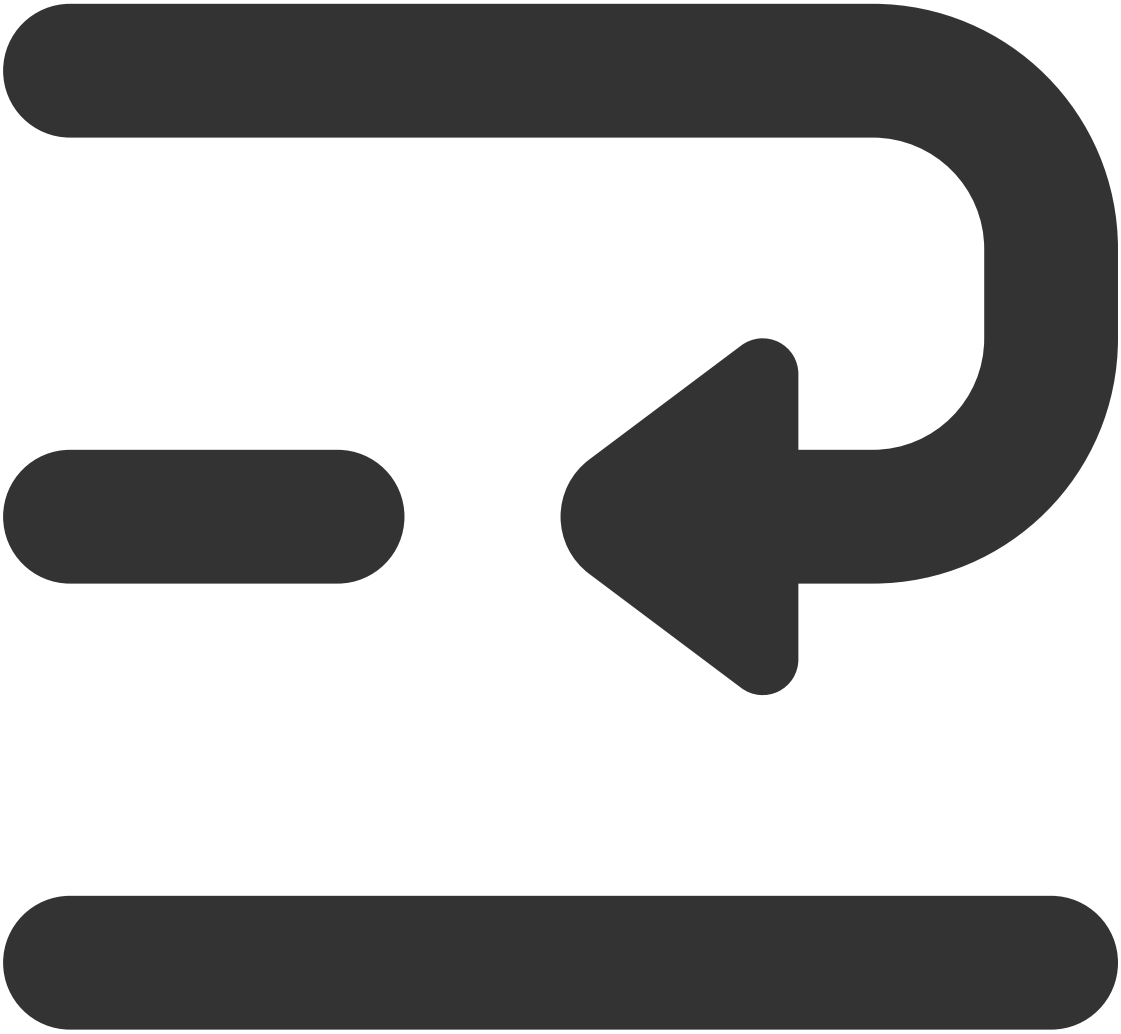
```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av ./node_modules/@tencentc
```

通过 npm 方式下载 TUIKit 组件：

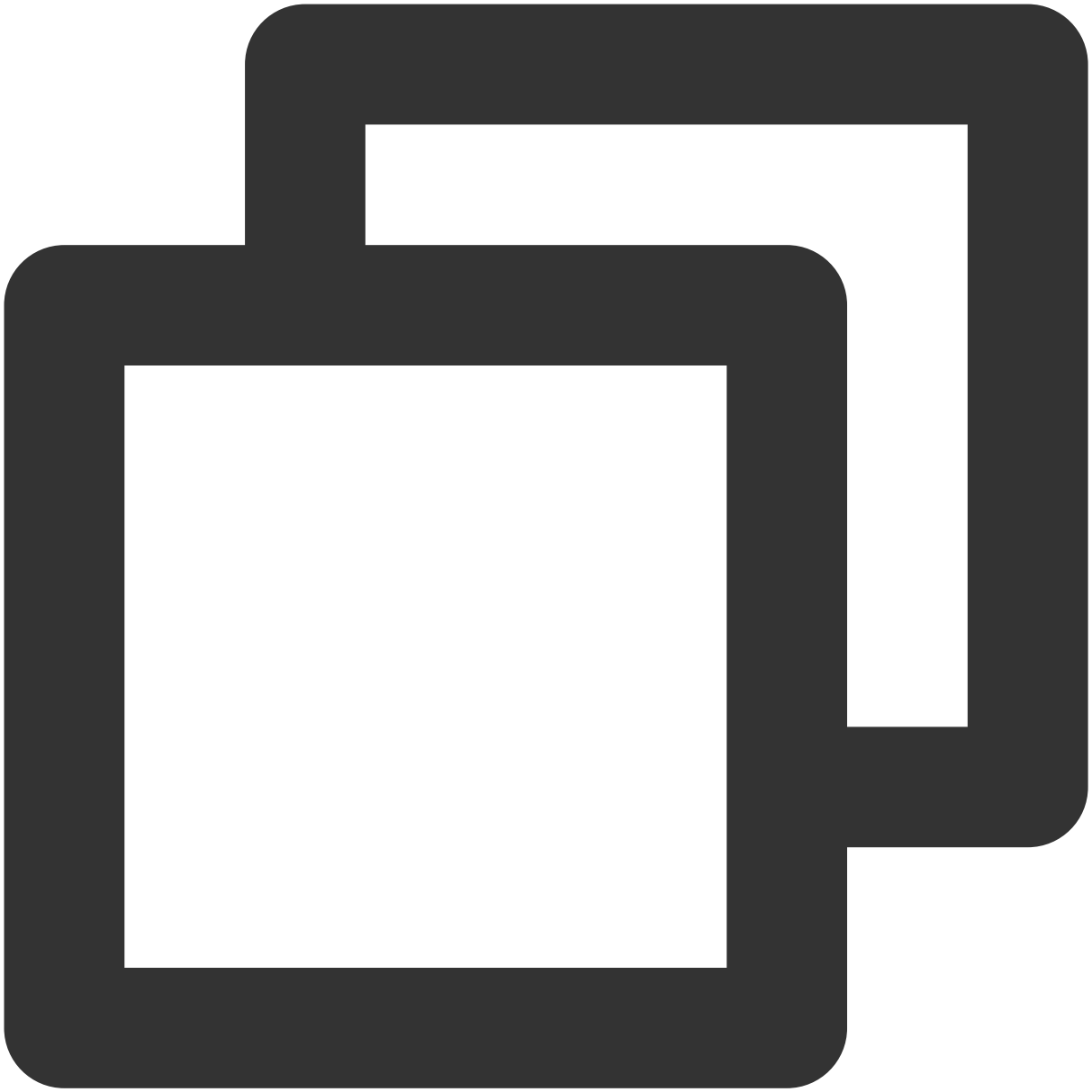


```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

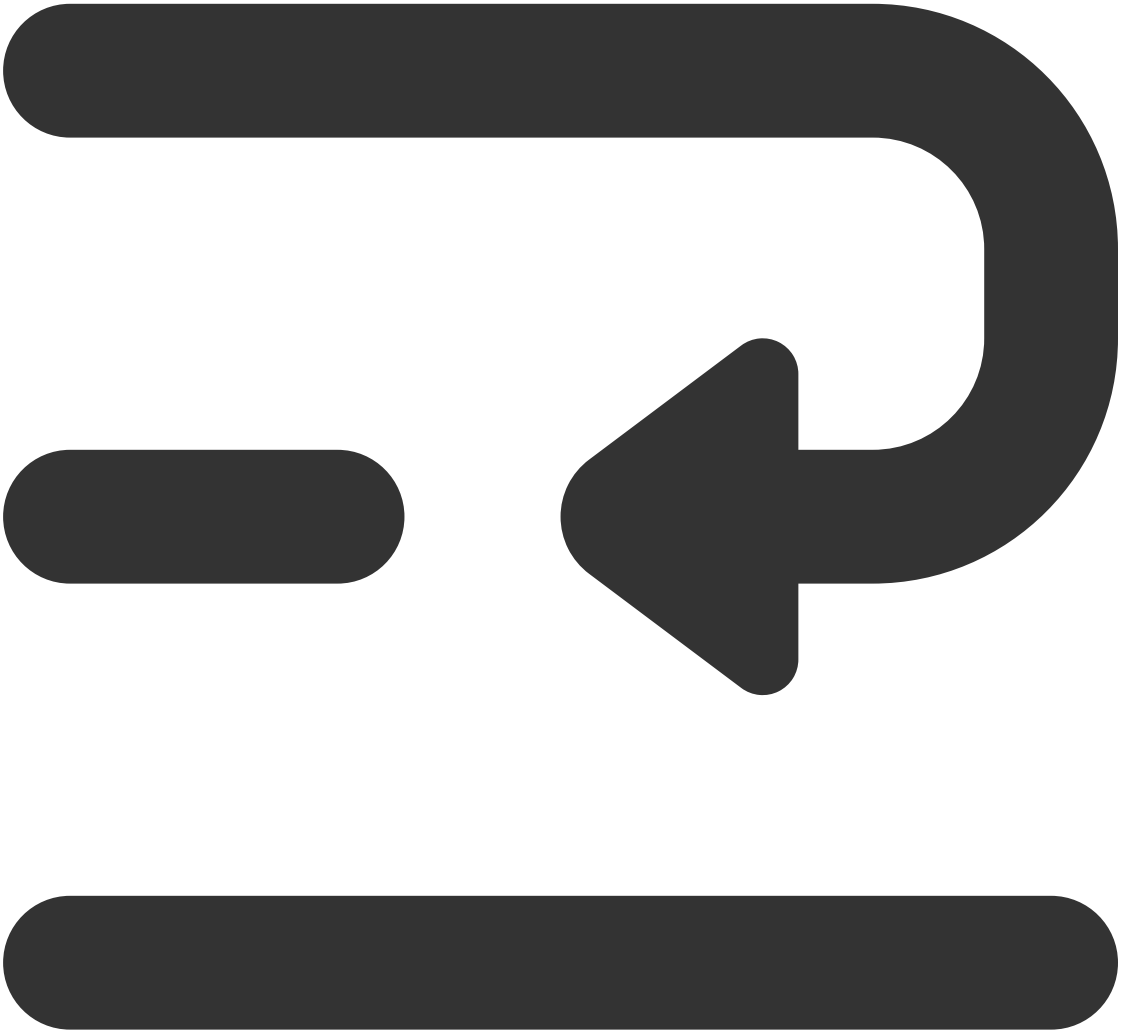
为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 pages 目录下，请在自己的项目根目录下执行以下命令：



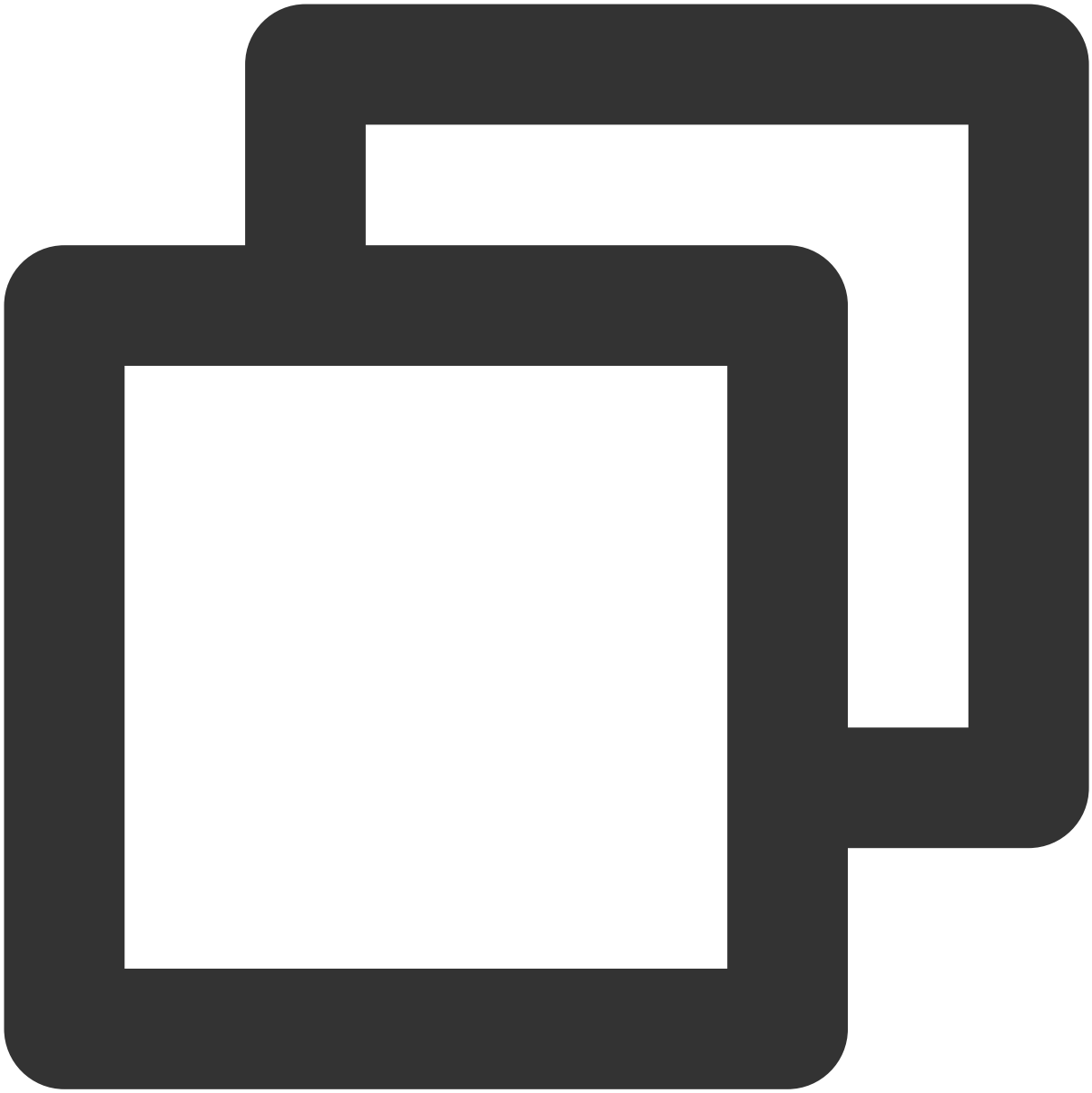
版权所有：腾讯云计算（北京）有限责任公司



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



版权所有：腾讯云计算（北京）有限责任公司

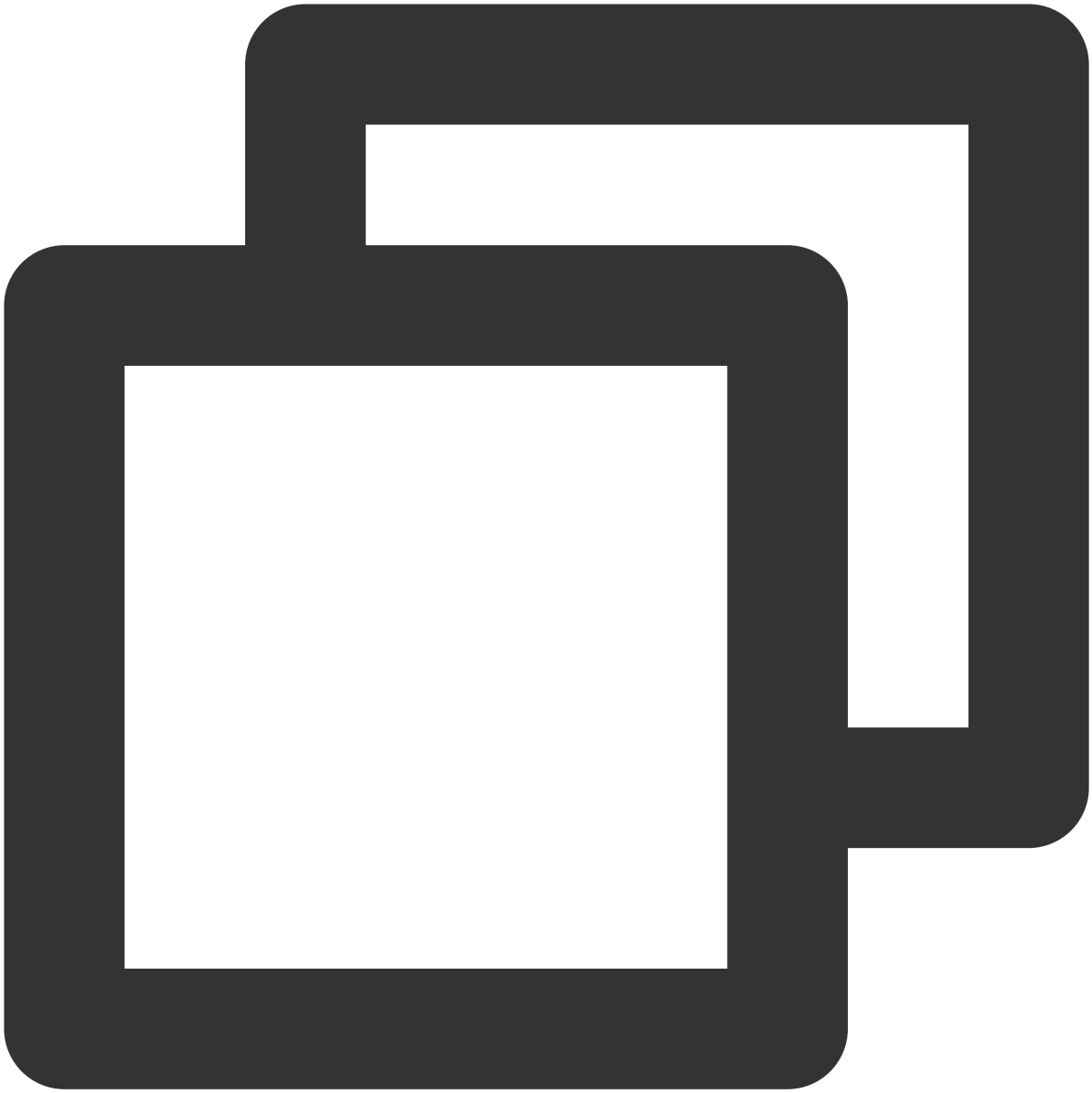


```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin .\\TUIKit\\tui-cu
```

步骤3：引入 TUIKit 组件

1. 工程配置

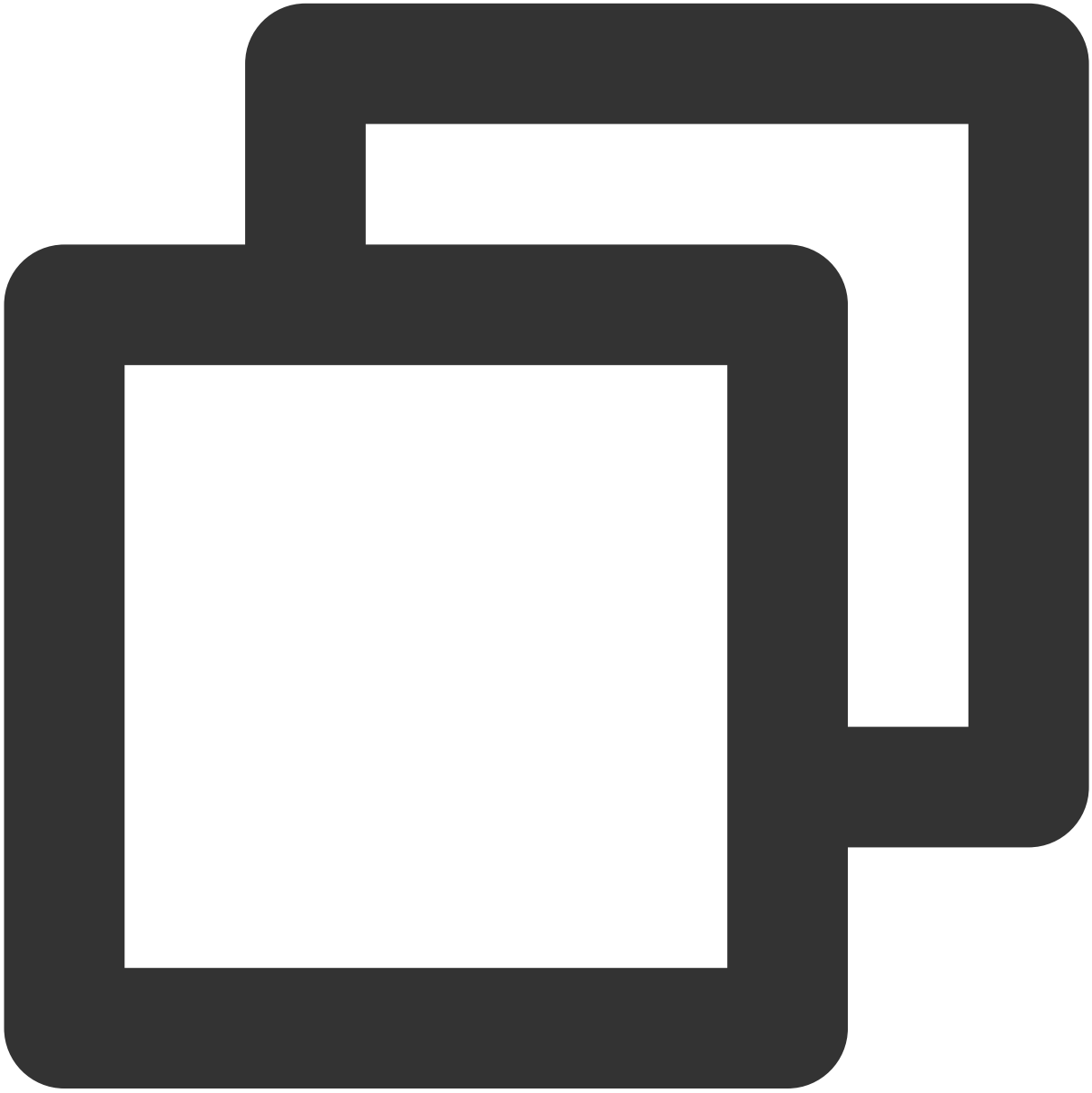
在根目录下创建 vue.config.js （vue3 项目请忽略此步骤）



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

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

在 `manifest.json` 文件的源码视图中开启分包配置



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

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

2. 集成 TUIKIt

注意：

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

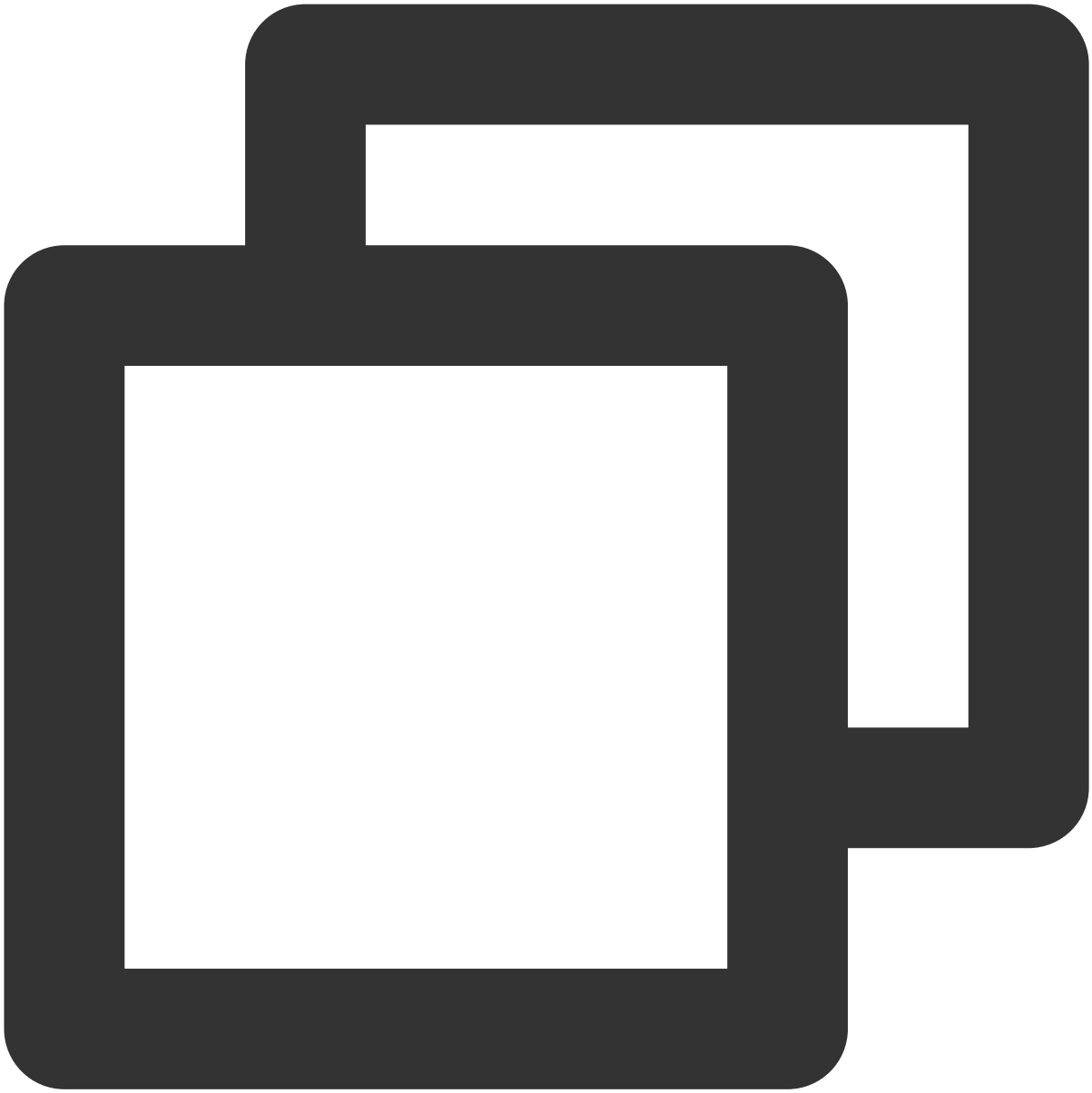
main.js 文件

pages.json 文件

App.vue 文件

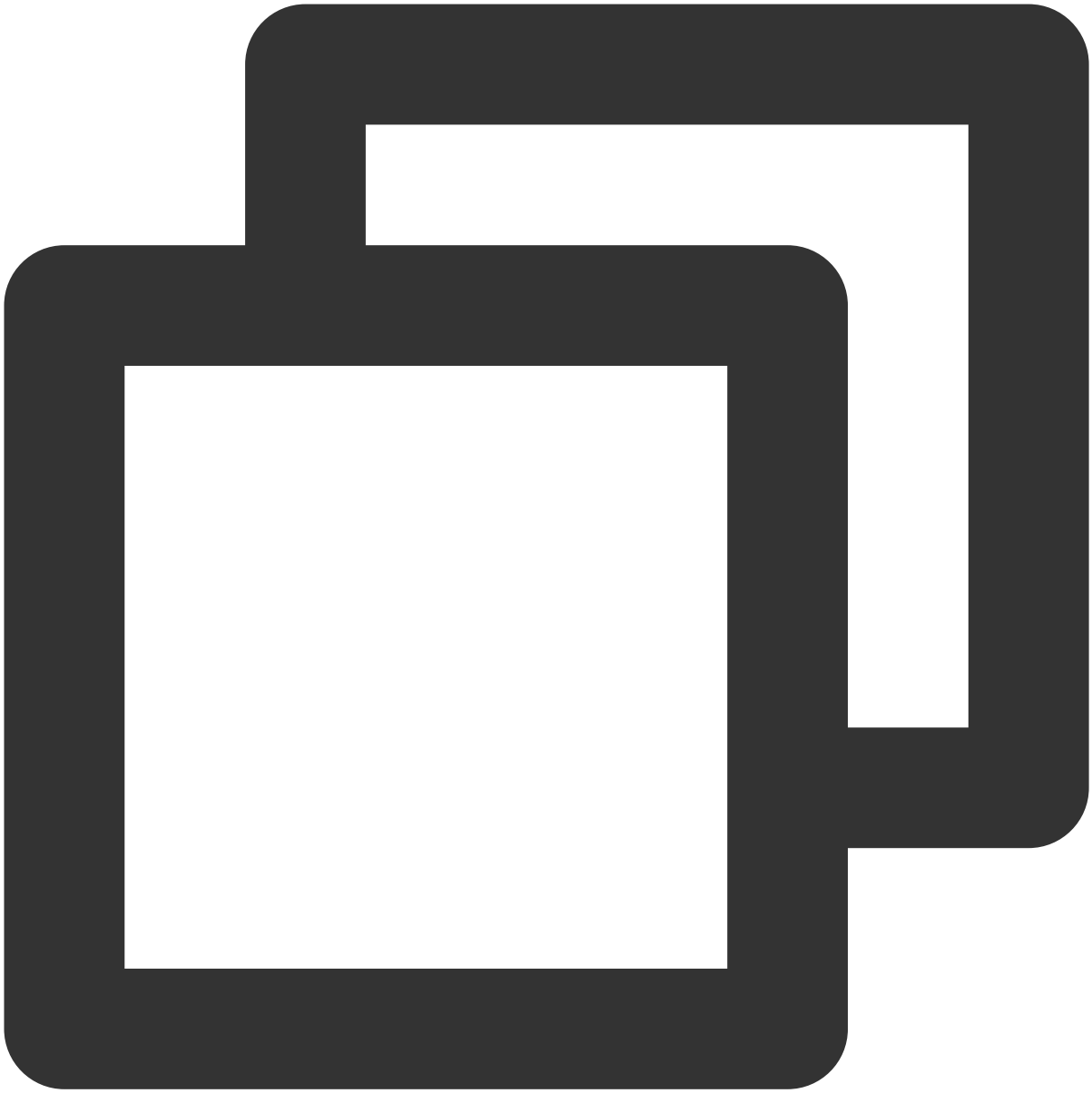
小程序分包首页

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



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

import App from './App';

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

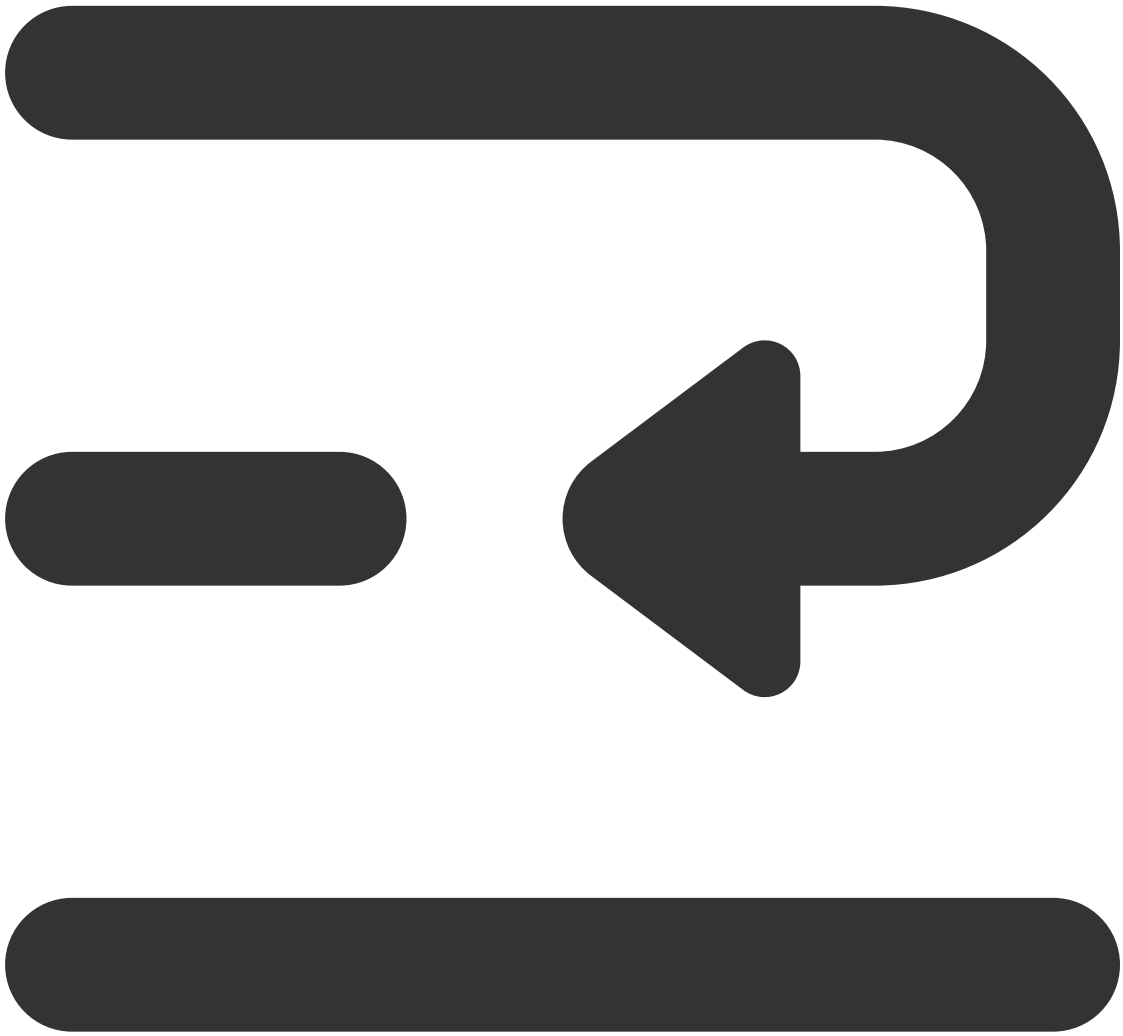
```
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

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



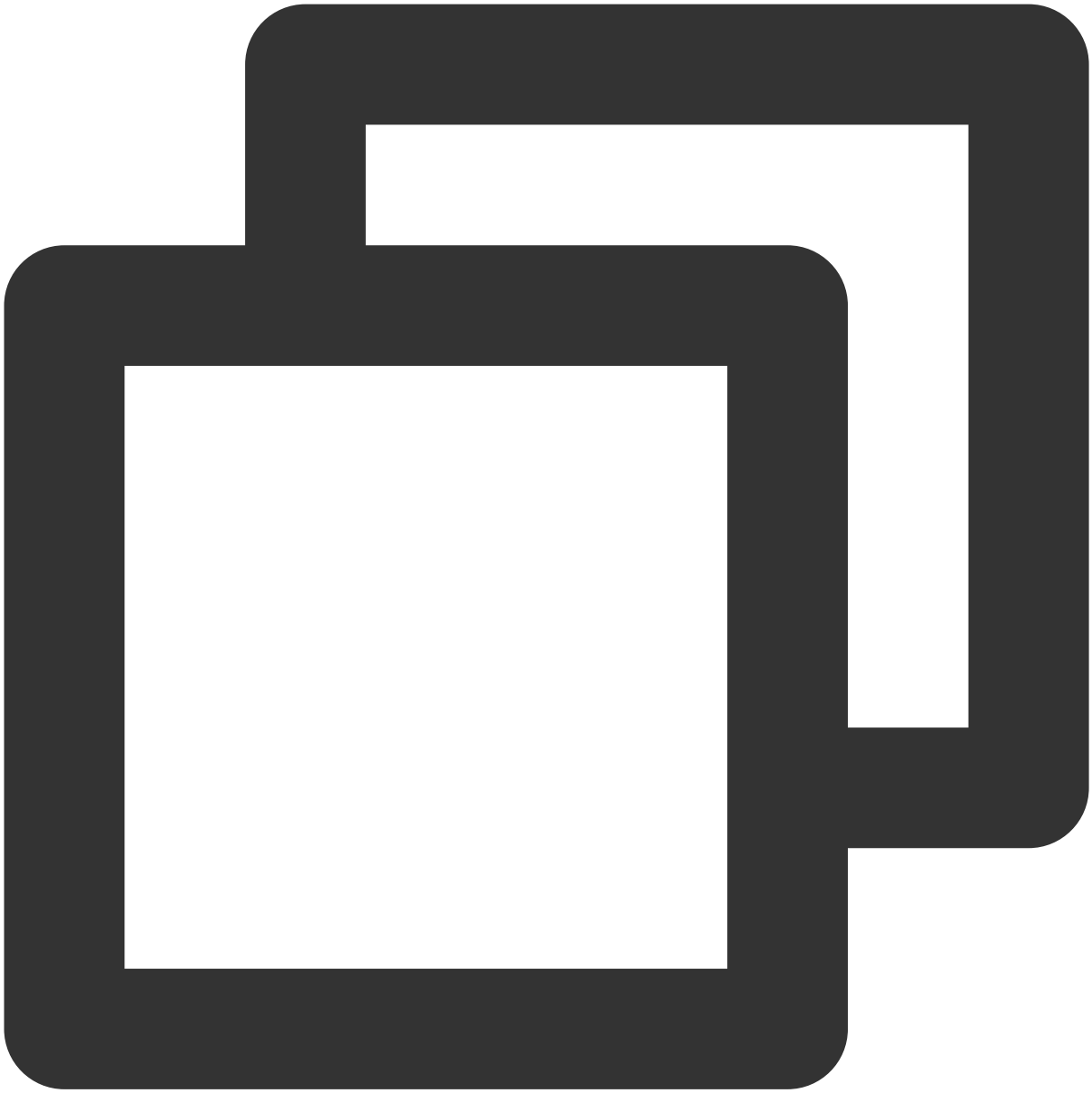
```
{
 "pages": [{
  "path": "pages/index/index" // 您的项目首页
 }],
 "subPackages": [{
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIConversation/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
```

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

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

```
  }
}
```



版权所有：腾讯云计算（北京）有限责任公司



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

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

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

```
}
</style>
```

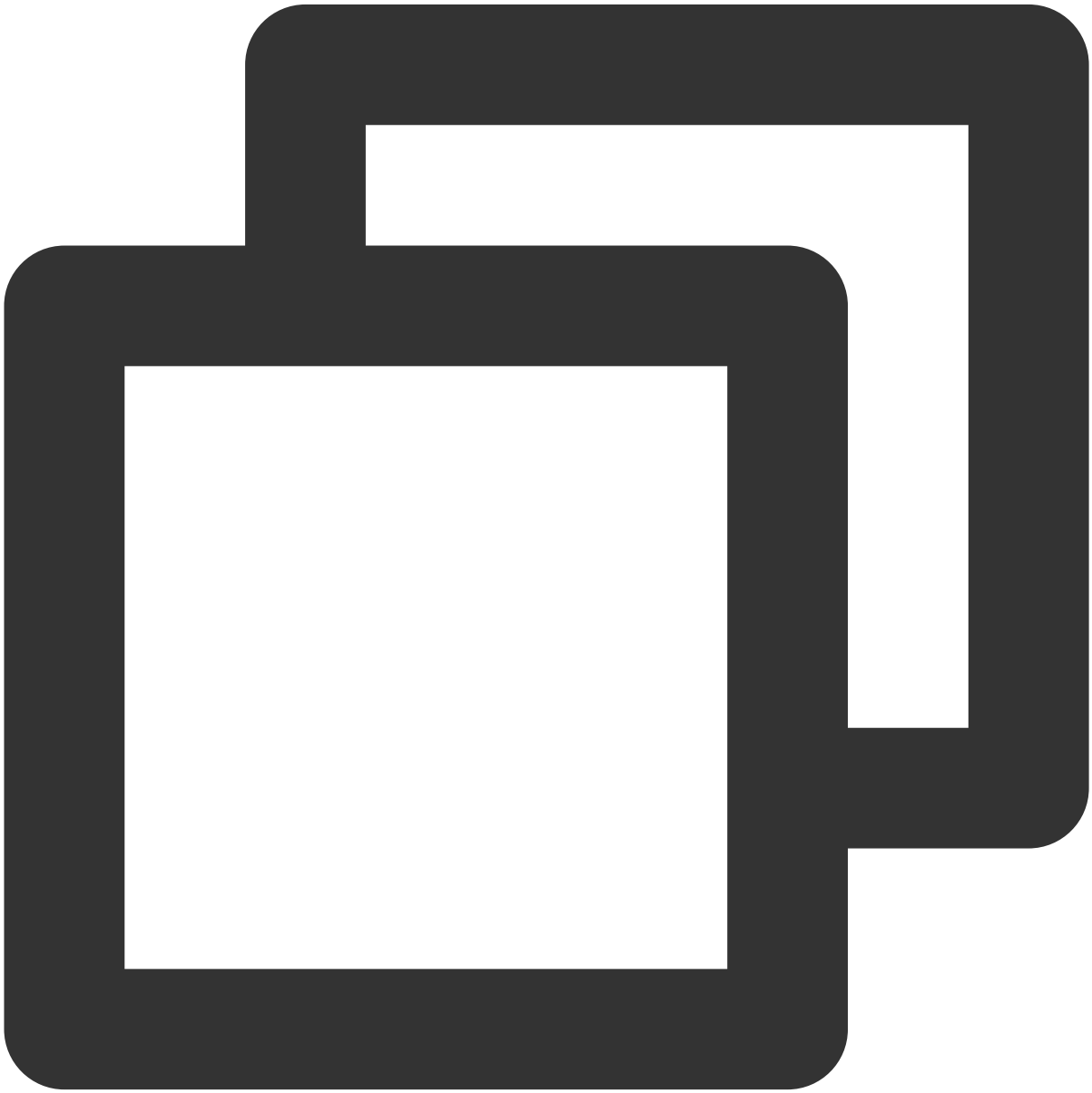
注意：

小程序默认分包集成，需要在 TUIKit 首启动页面完成 login。

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

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

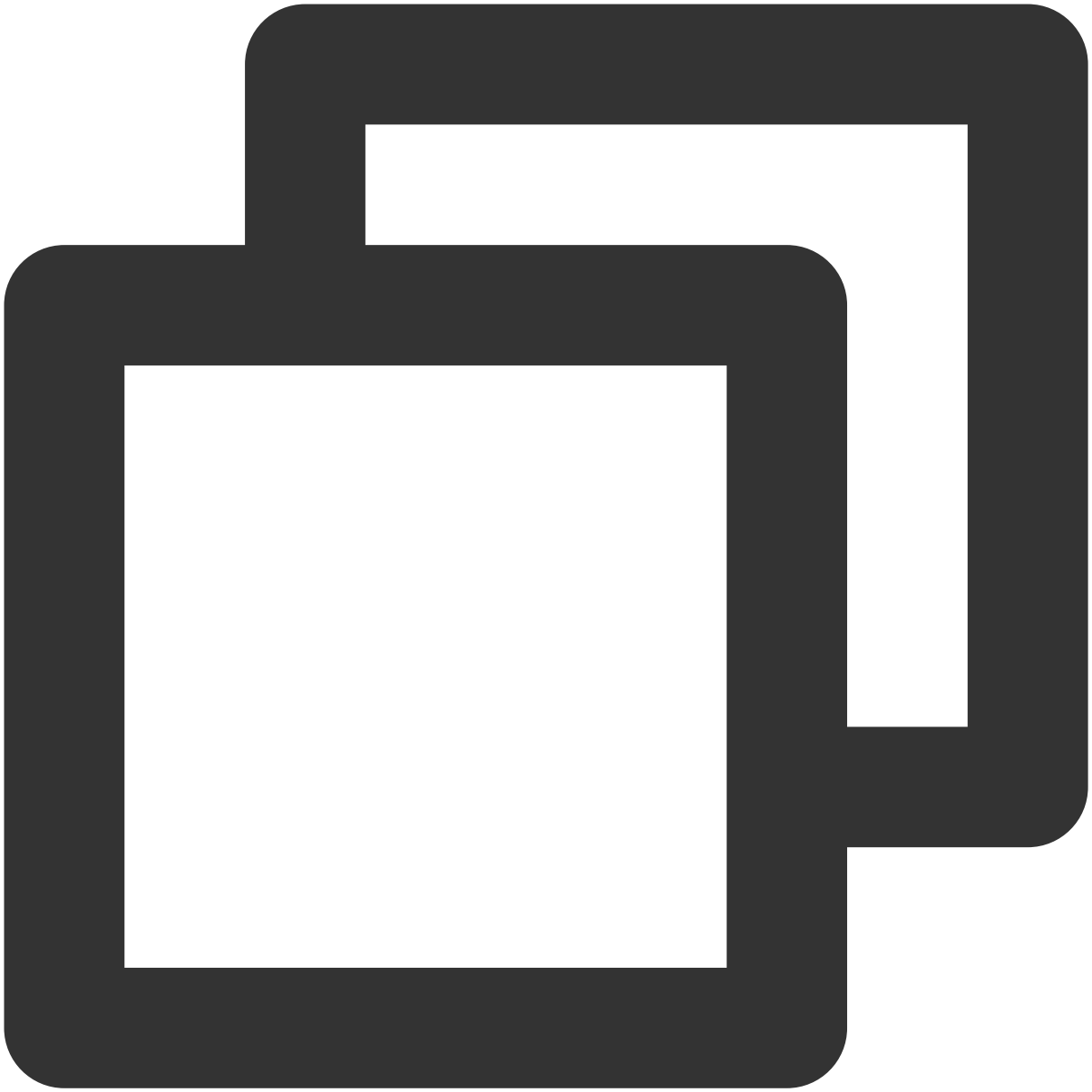
步骤1: 在 TUIKit/components/TUIConversation 文件夹下创建 subPackage-init.ts 文件



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

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

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

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

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



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

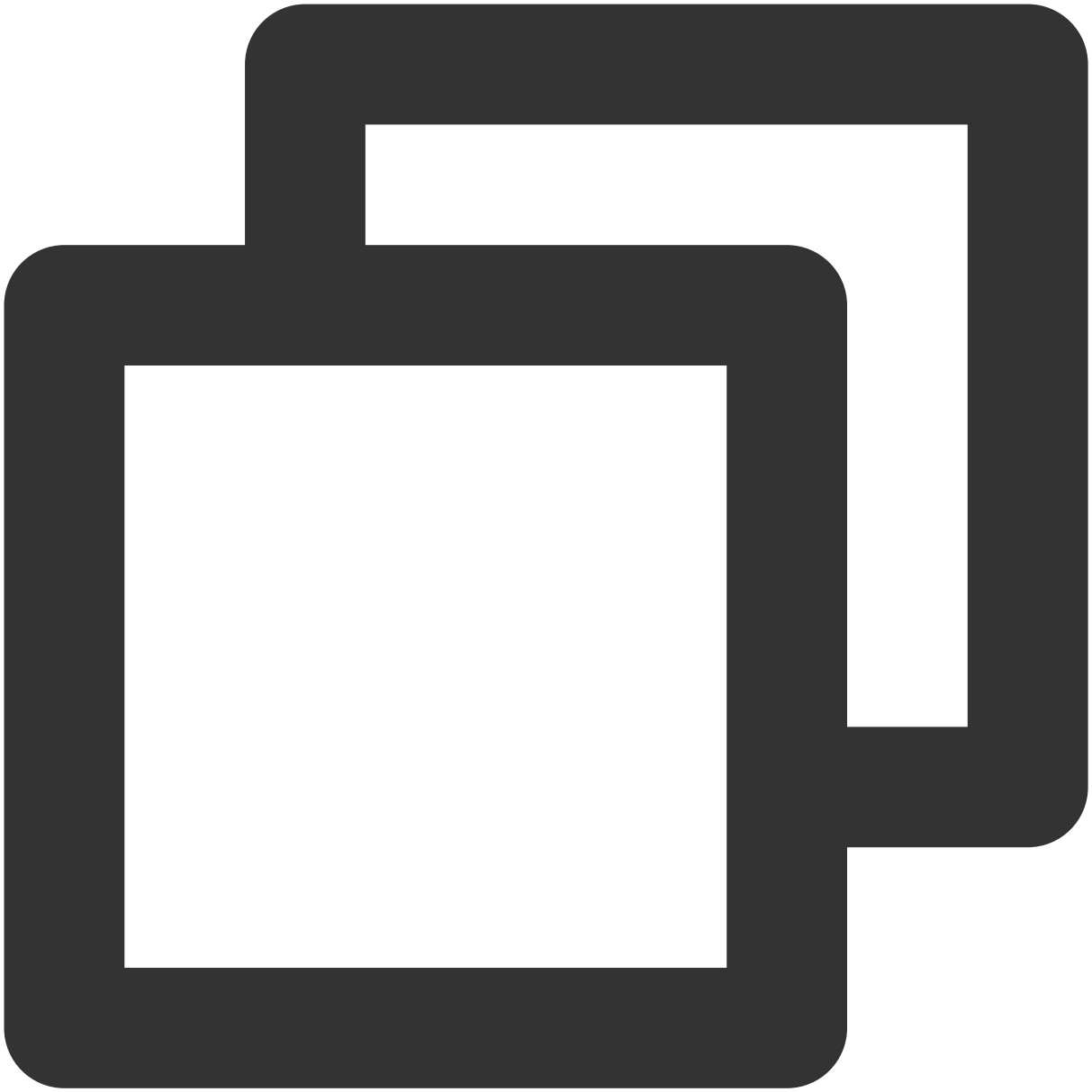
如图所示：

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

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

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

在 pages/index 文件夹下创建 index.vue 文件



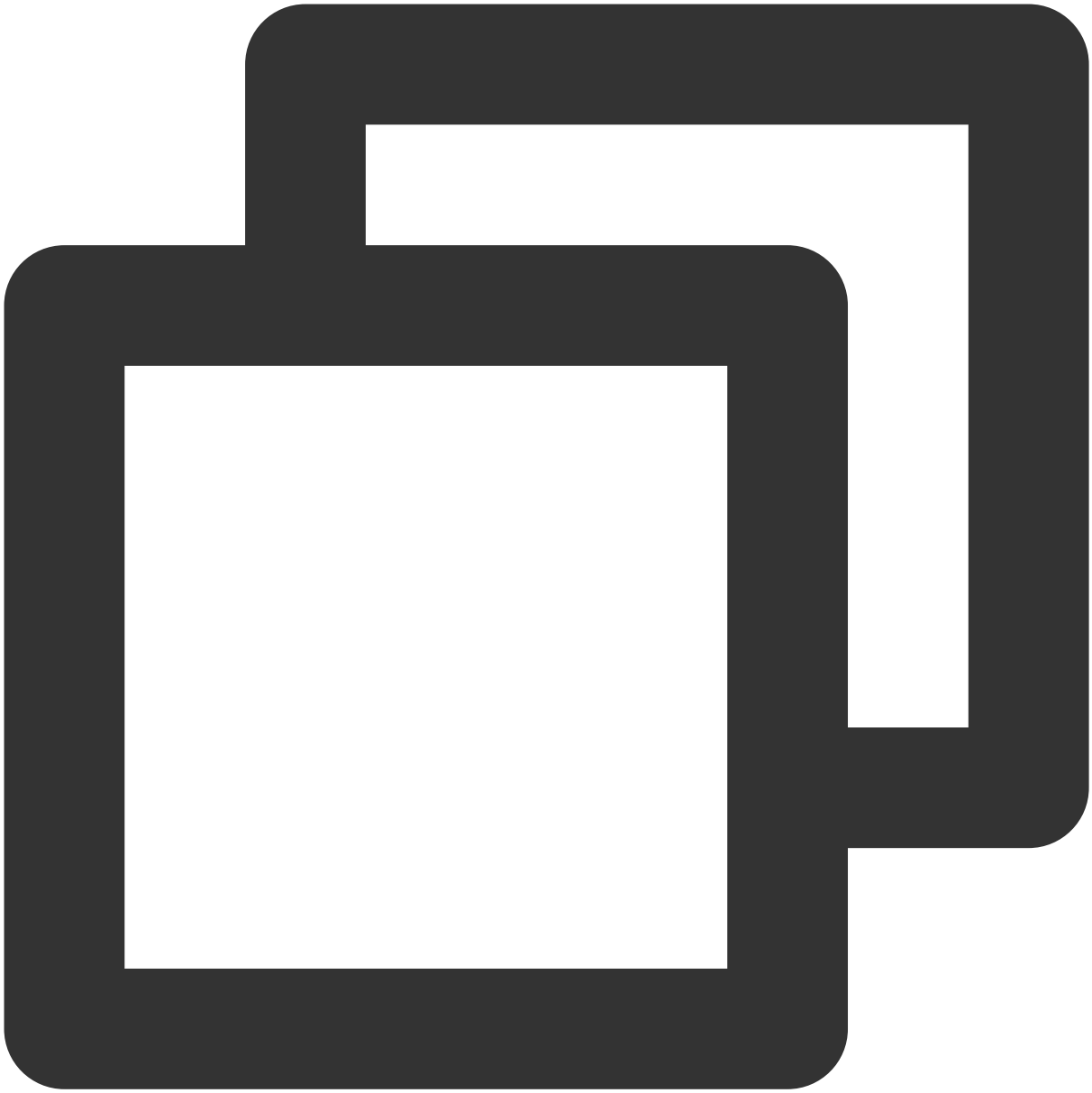
```
<template>
  <div class="index">
        <p class="index-button" @click="openConversation">打开 TUIKit 会话</p>
        <p class="index-button" @click="openContact">打开 TUIKit 联系人</p>
  </div>
</template>
<script>
export default {
methods: {
  // 打开 TUIKit 会话列表
  openConversation() {
```

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

步骤4：获取 SDKAppID、secretKey、userID

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

版权所有：腾讯云计算（北京）有限责任公司



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

获取 SDKAppID、secretKey

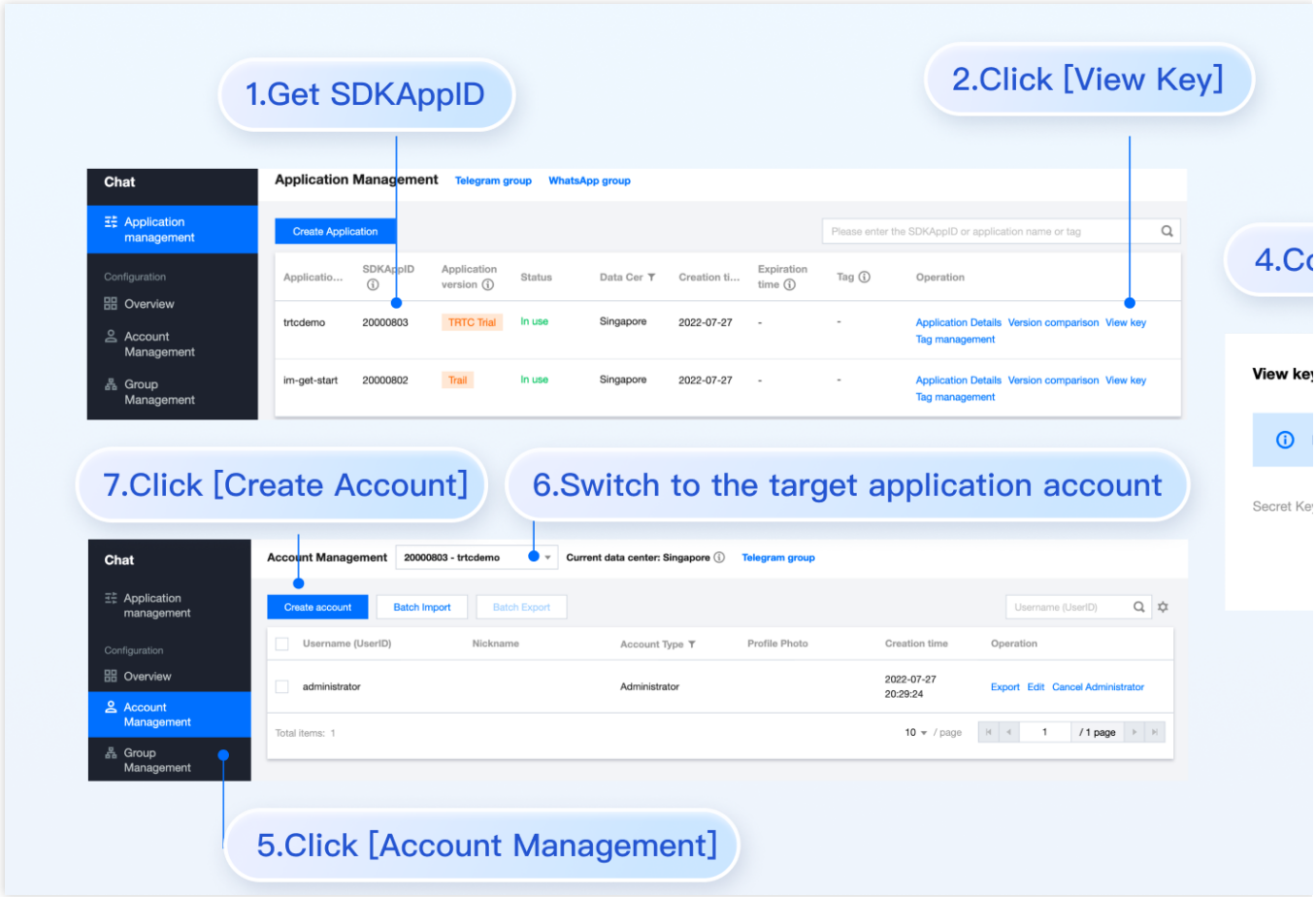
在即时通信 IM 控制台中的**应用管理**页面下，可以看到您创建的应用，第二列即是 SDKAppID，然后单击操作中的**查看密钥**。网站会弹出查看密钥的对话框，然后单击**显示密钥**，即可查看密钥。

创建 userID 为 test-user1 的账户

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

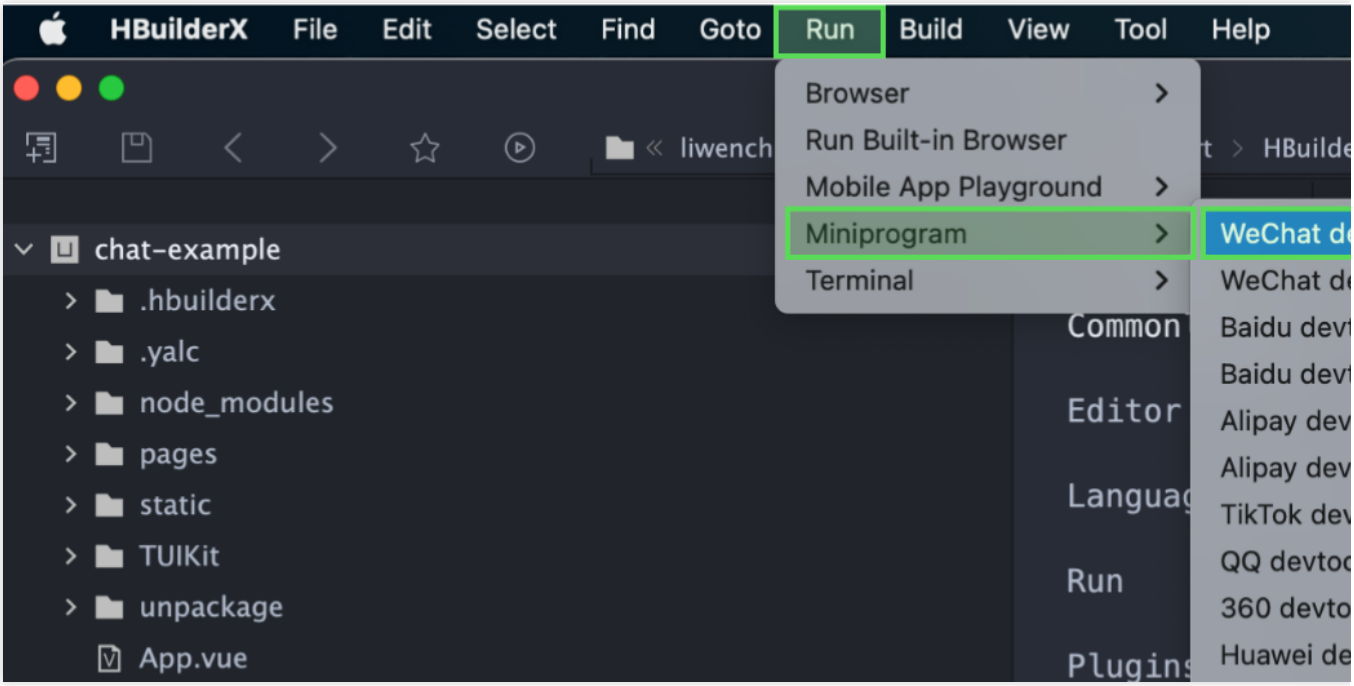
说明：

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



步骤5：启动项目

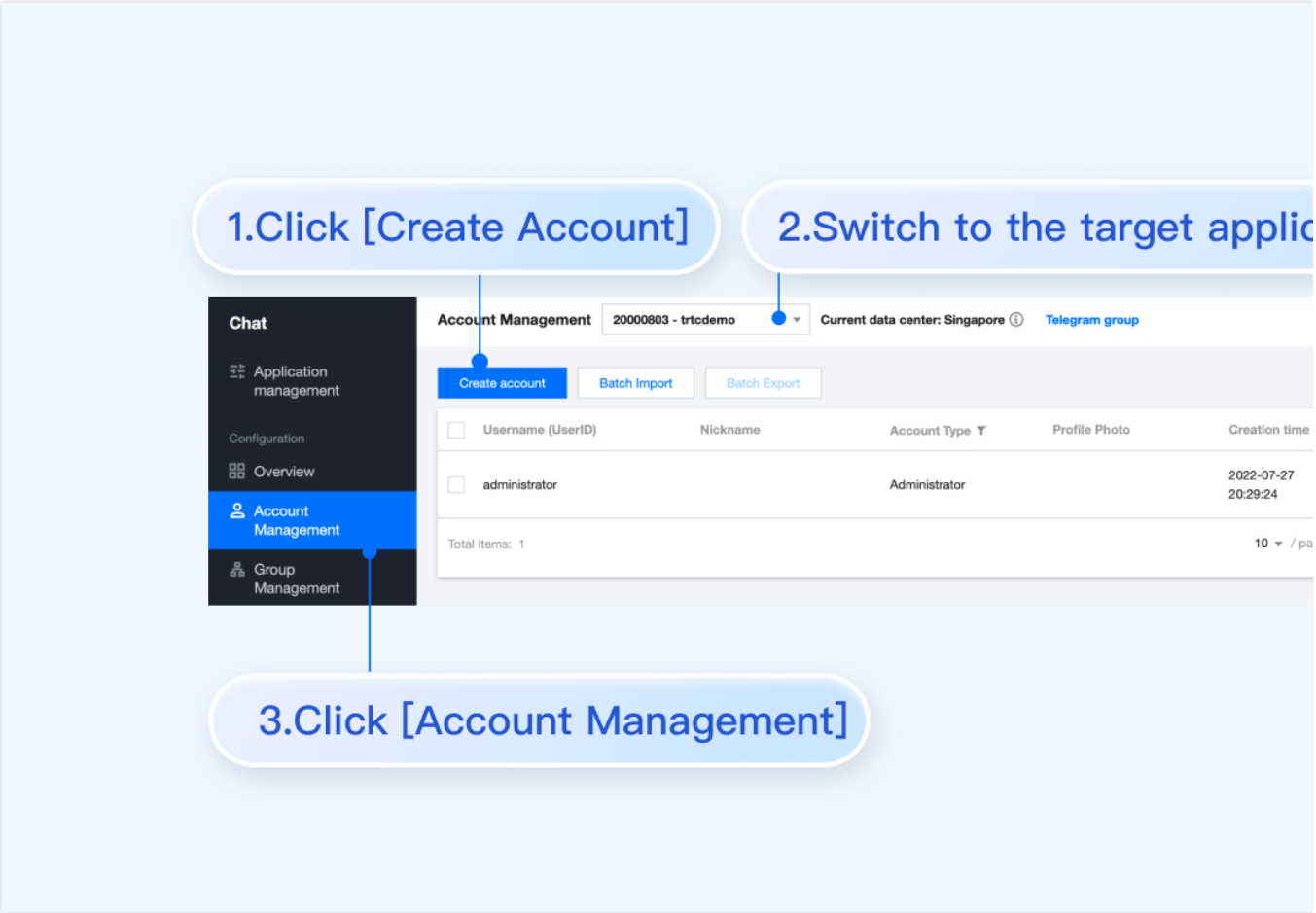
1. 使用 HBuilderX 启动该项目，点击“运行-运行到小程序模拟器-微信开发者工具”。



2. 如果 HBuilderX 没有自动拉起微信开发者工具，请使用微信开发者工具手动打开编译后的项目。
使用微信开发者工具打开项目根目录下的 `unpackage/dist/dev/mp-weixin` 即可。
3. 打开项目后，在微信开发者工具 “详情-本地设置” 中勾选 “不校验合法域名、web-view（业务域名）、TLS版本以及 HTTPS 证书”。

步骤6：发送您的第一条消息

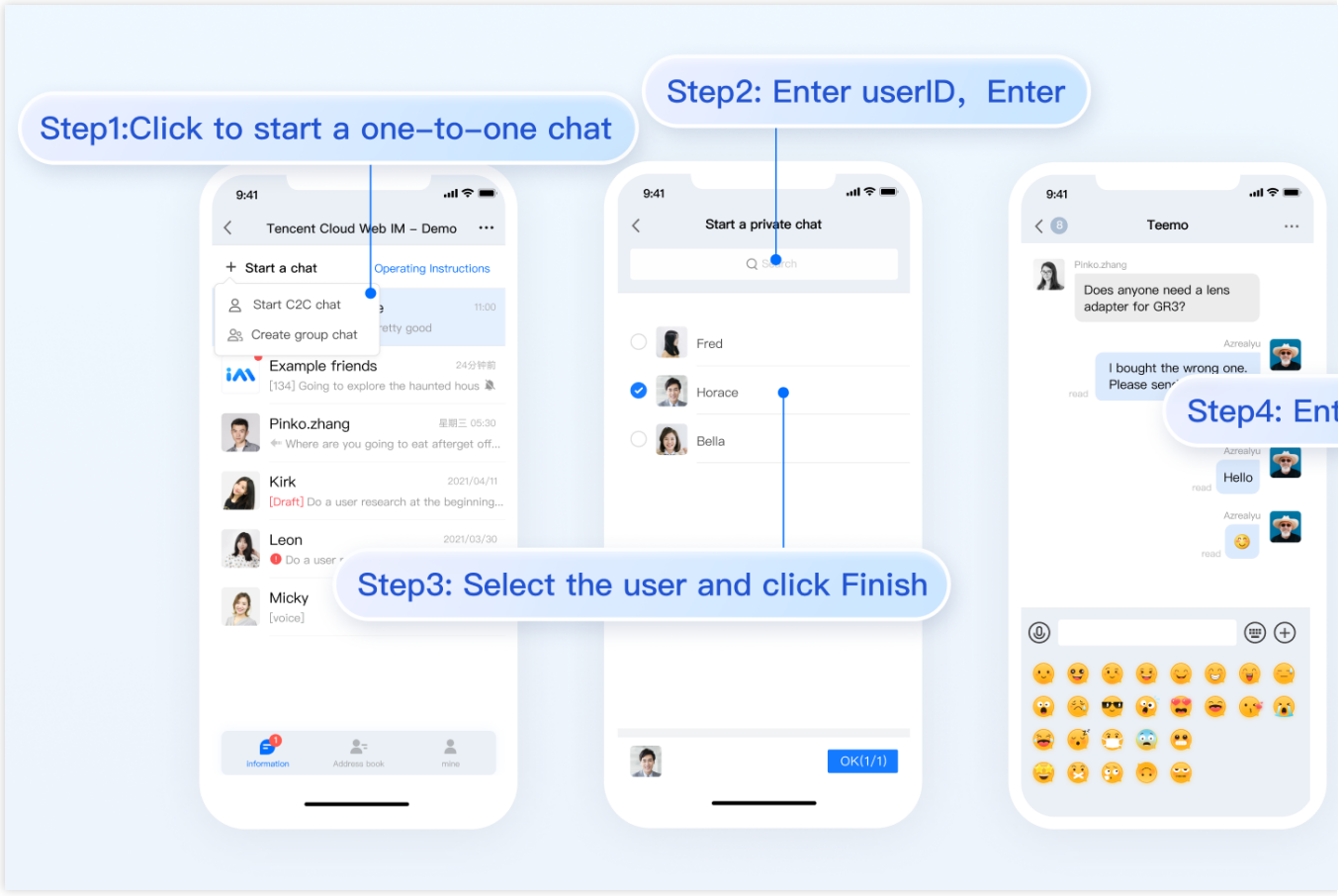
1. 通过即时通信 IM 控制台创建一个 User 账号
从左侧边栏进入**账号管理**页面，单击**新建账号**并创建一个普通账号 userID：test-user2。



2. 运行项目并发起会话

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

版权所有：腾讯云计算（北京）有限责任公司



更多高级特性

音视频通话 TUICallKit 插件

说明：

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

如果您需要集成通话功能，可参考以下文档实现。

打包到 APP 请参考： 音视频通话（客户端）

打包到小程序请参考：音视频通话（小程序）

打包到 H5 请参考官：音视频通话（H5）

敬请期待。

TIMPush 离线推送插件

说明

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

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

版权所有：腾讯云计算（北京）有限责任公司

敬请期待。

独立集成 TUIChat 组件

可参考 独立集成 TUIChat 组件 方案。

常见问题

更多问题请参见 Uniapp 常见问题。

交流与反馈

点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

参考文档

UIKit （vue2 / vue3）相关：

chat-uikit-uniapp (vue2/vue3) github 源码

chat-uikit-uniapp npm 快速接入

ChatEngine 相关：

ChatEngine API 手册

ChatEngine npm

版权所有：腾讯云计算（北京）有限责任公司

仅集成聊天
Android

最近更新时间：2024-06-24 17:23:57

本文将介绍如何集成 `TUIChat` 聊天组件。

说明：

从 5.7.1435 版本开始，TUIChat 支持了经典版 UI 组件。

从 6.9.3557 版本开始，TUIChat 新增了全新的简约版 UI 组件。

您可以根据需求自由选择经典版或简约版 UI 组件。

效果展示

TUIChat 提供了私信聊天（1V1）和群聊（Group）功能，支持对消息的多种操作，例如发送不同类型的消息、对消息长按点赞/回复/引用、查询消息已读回执详情等。您可以仅集成 TUIChat 到您的 App 中。聊天界面使用场景非常广泛，例如房产中介咨询、在线医疗问诊、电商在线客服、保险远程定损等。

界面效果如下图所示：

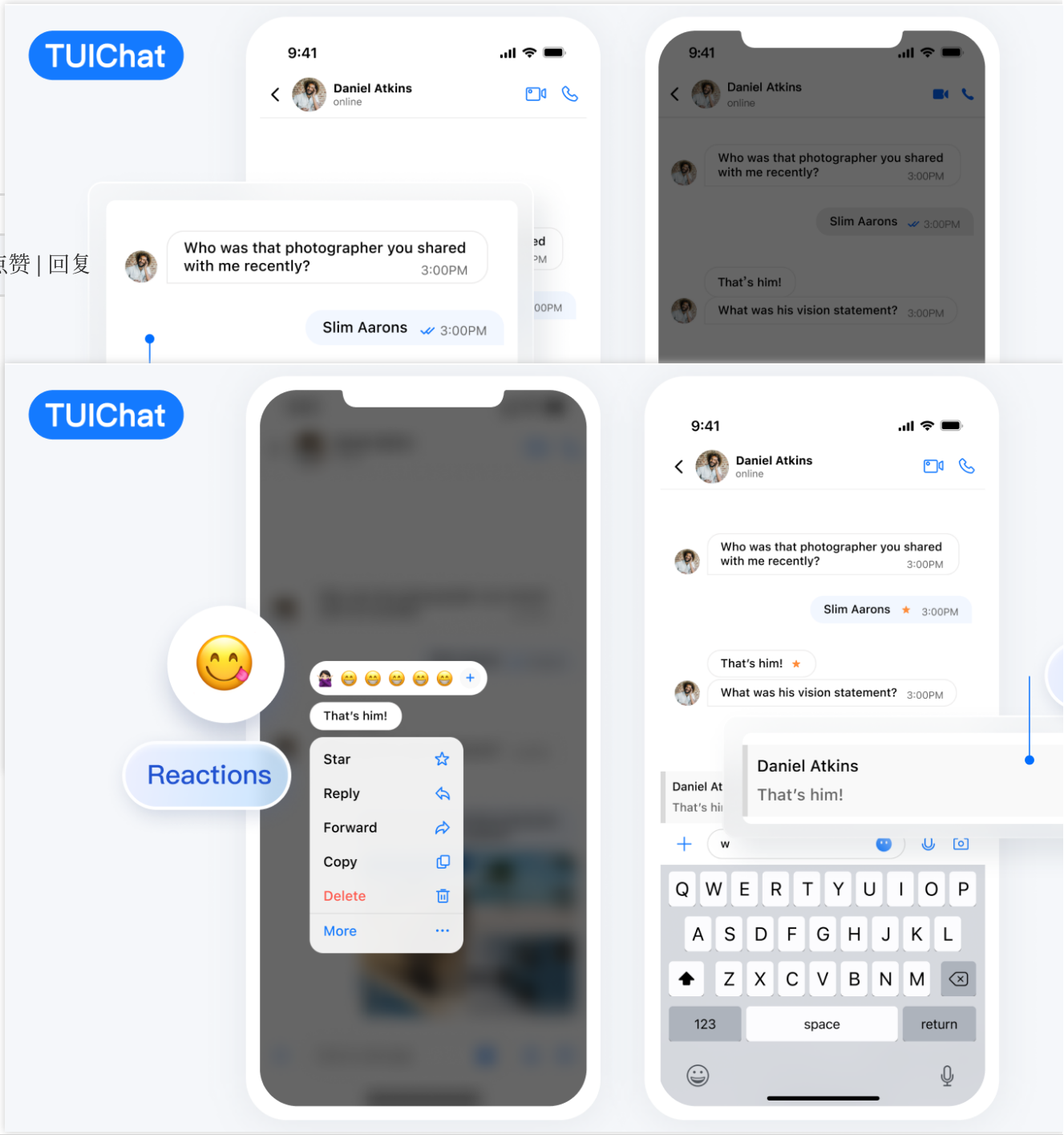
简约版

RTL语言

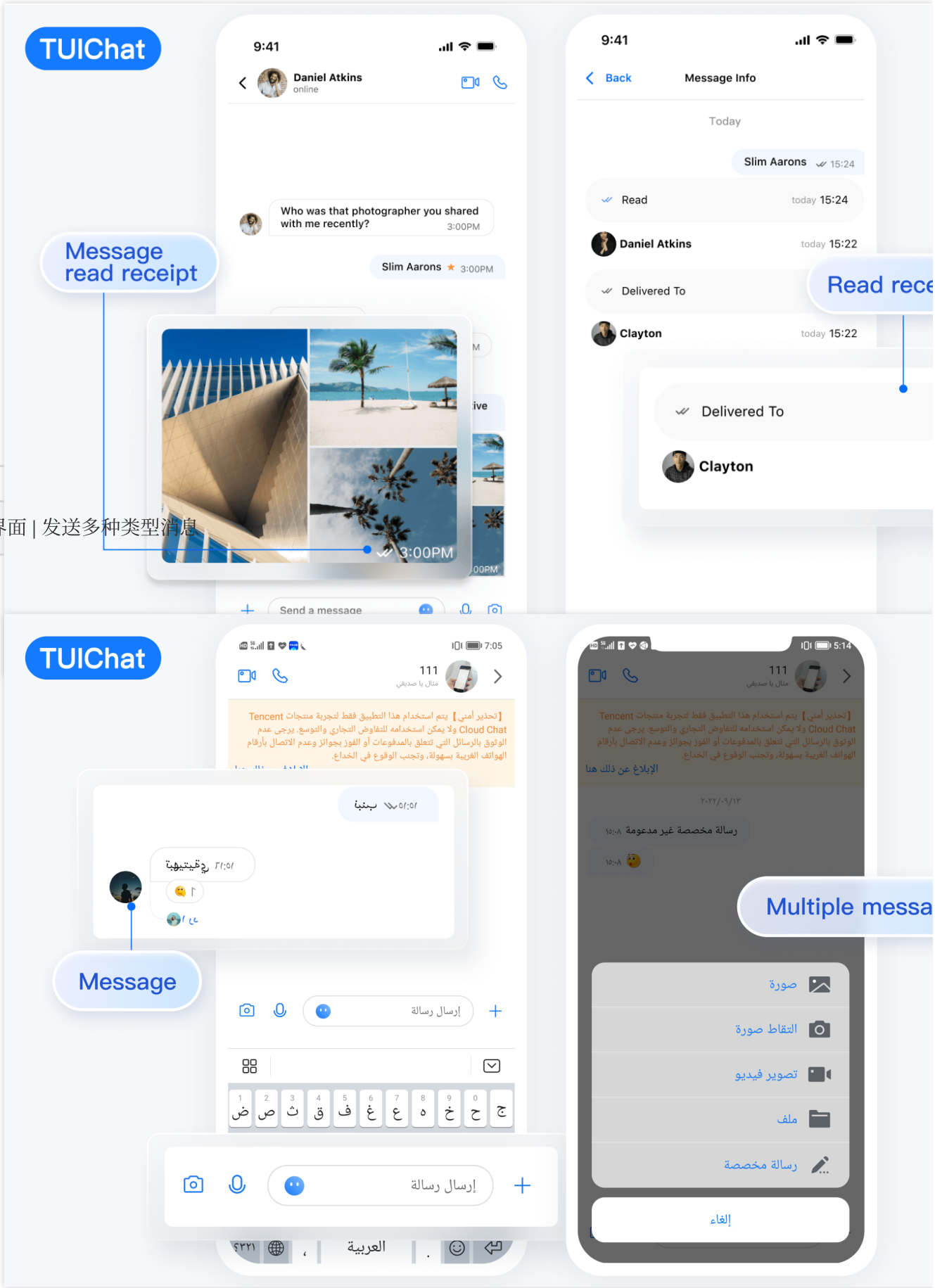
经典版

消息界面 | 发送多种类型消息

消息点赞 | 回复

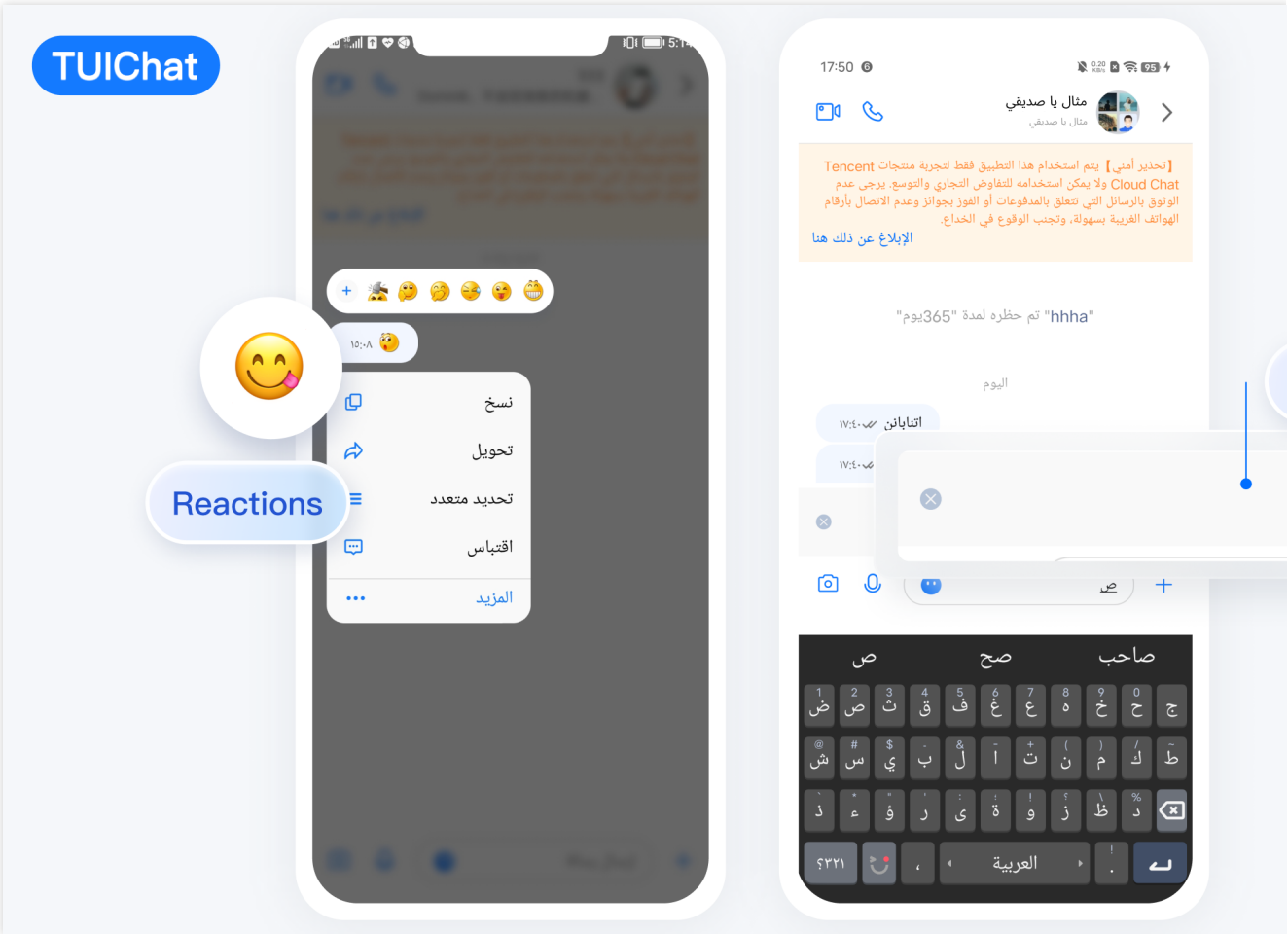


消息已读回执 | 已读回执详情

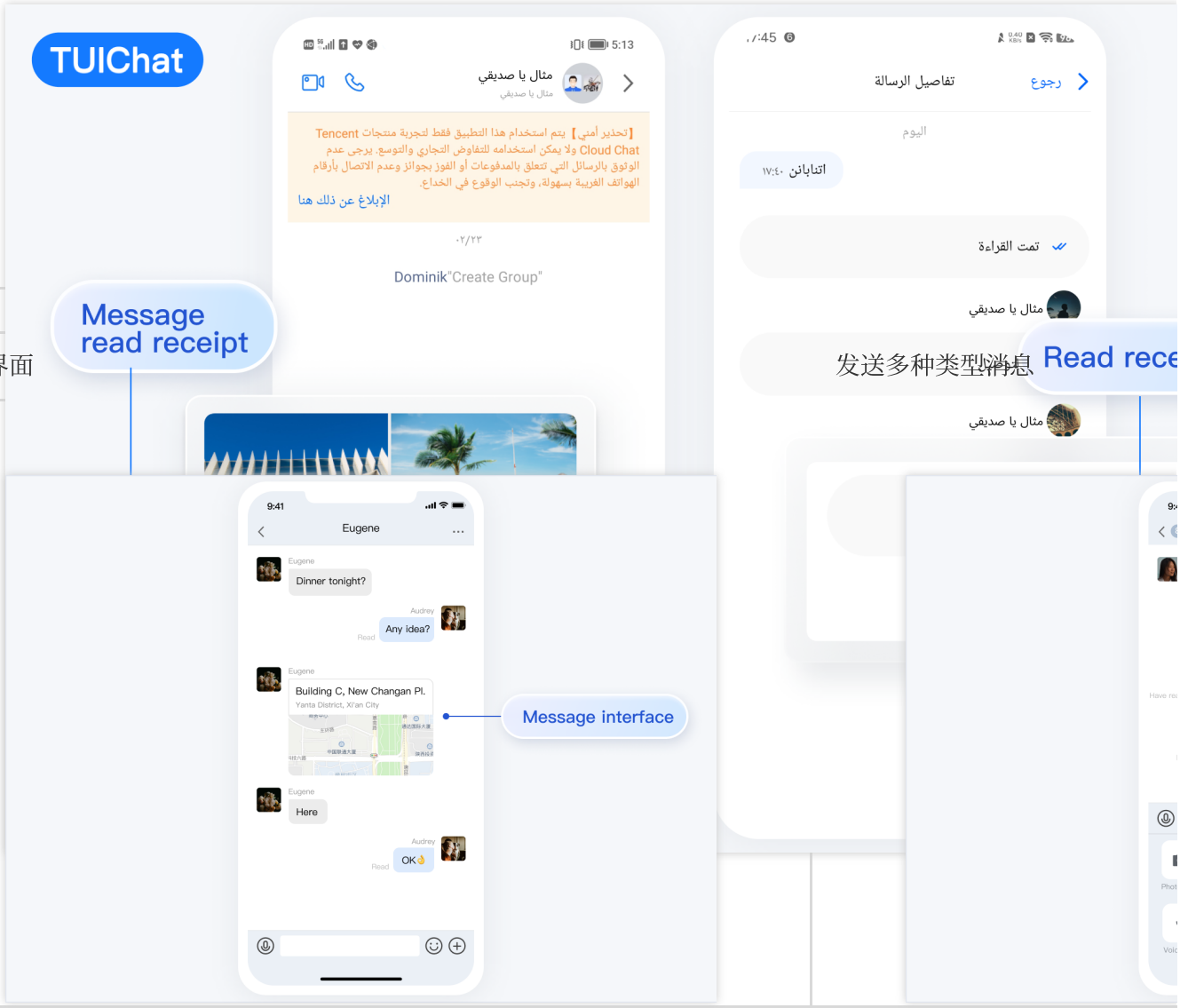


消息界面 | 发送多种类型消息

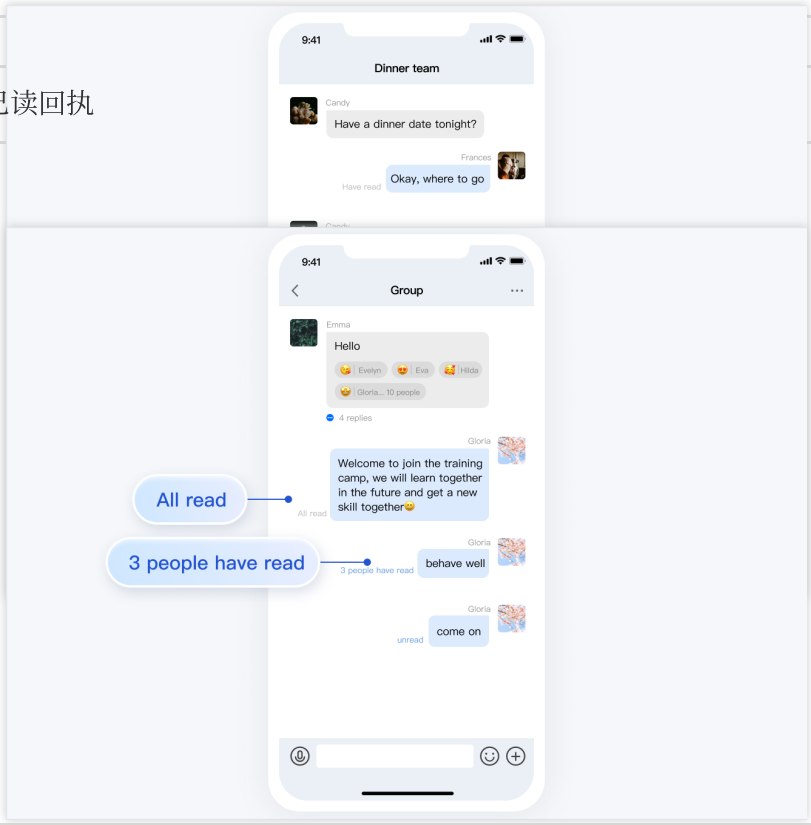
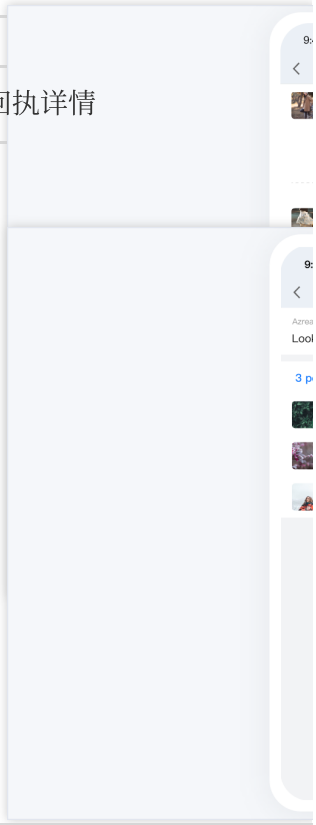
消息点赞 | 回复



消息已读回执 | 已读回执详情

消息界面	发送多种类型消息
	

消息点赞/回复/引用	消息回复详情

消息已读回执	已读回执详情
	

开发环境要求

Android Studio-Giraffe

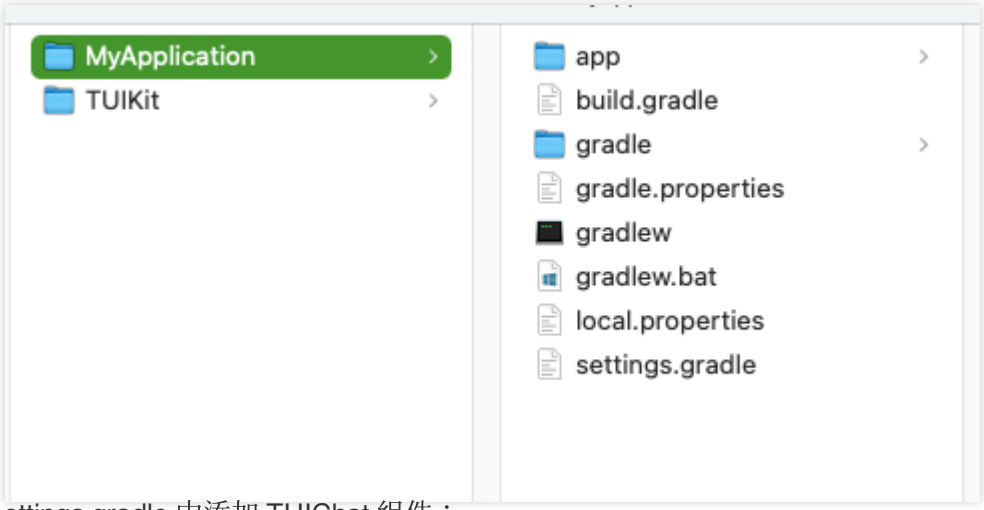
Gradle-7.2

Android Gradle Plugin Version-7.0.0

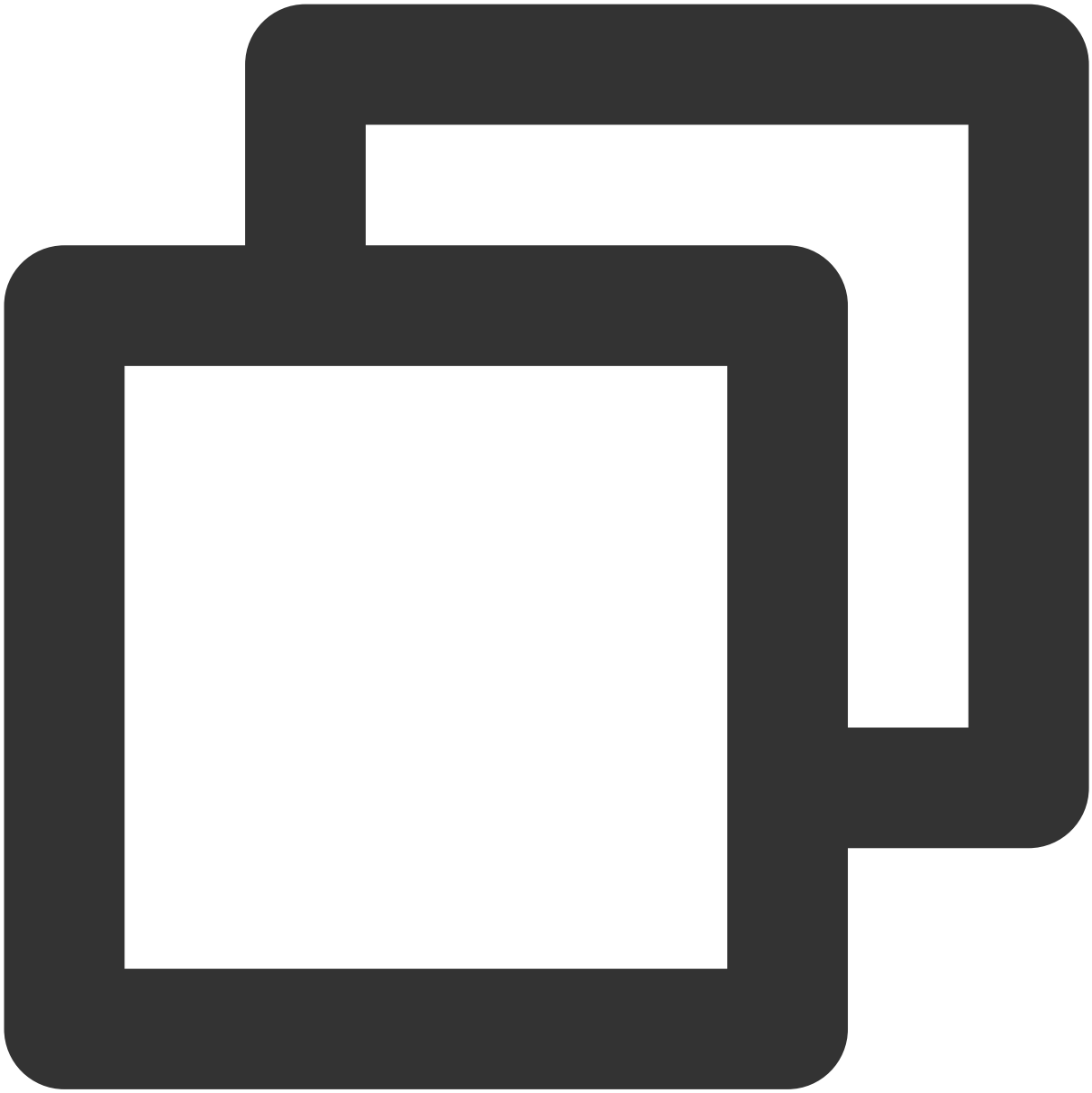
kotlin-gradle-plugin-1.5.31

集成 TUIChat 源码

1. 从 GitHub 下载 TUIKit 源码。使 TUIKit 文件夹跟自己的工程文件夹同级，例如：



2. 在 settings.gradle 中添加 TUIChat 组件：

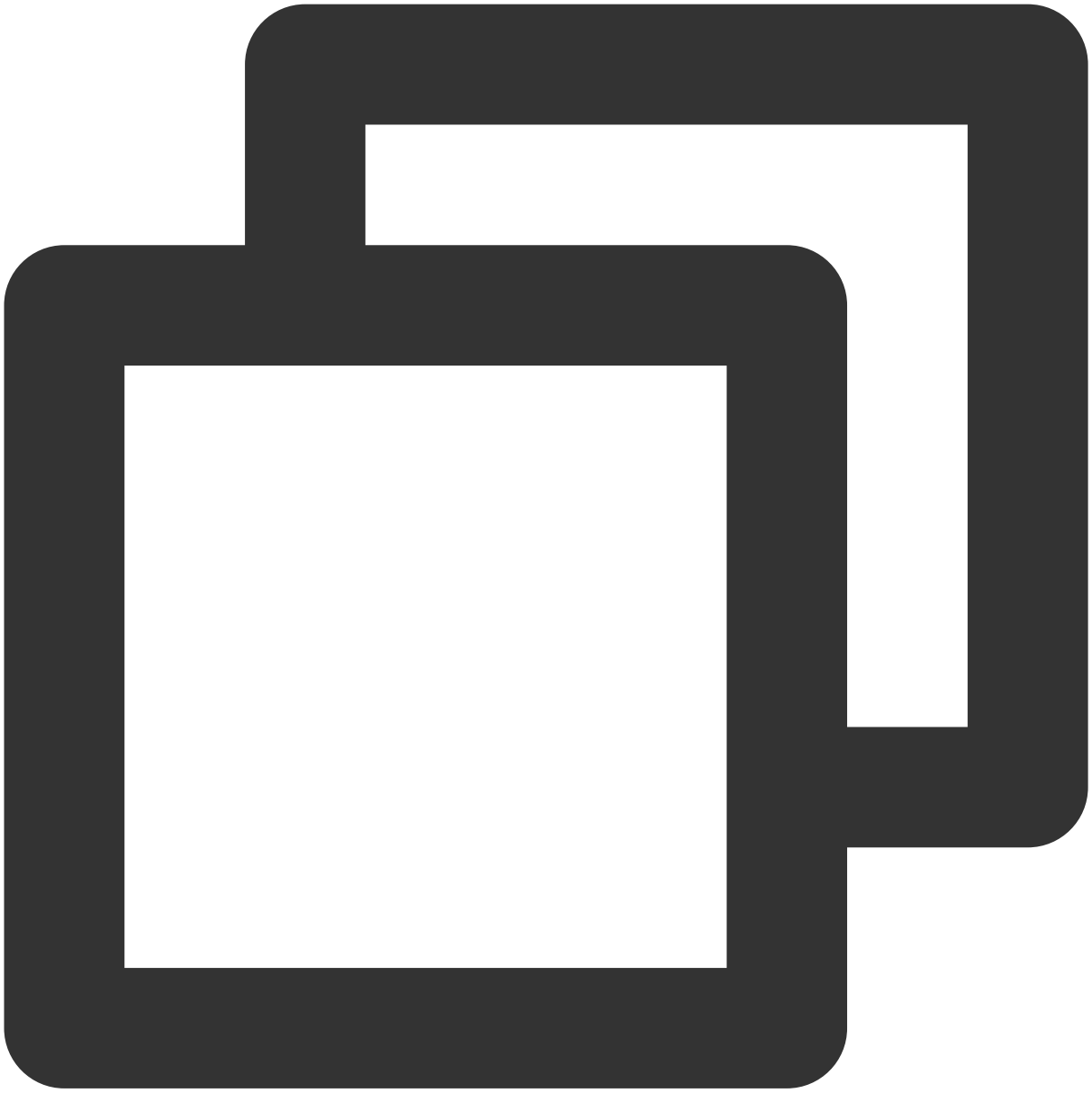


```
// Include the internal communication module (required module)
include ':tuicore'
project(':tuicore').projectDir = new File(settingsDir, '../TUIKit/TUICore/tuicore')

// Include the Chat component common module (required module)
include ':timcommon'
project(':timcommon').projectDir = new File(settingsDir, '../TUIKit/TIMCommon/timco

// Include the chat feature module (basic feature module)
include ':tuichat'
project(':tuichat').projectDir = new File(settingsDir, '../TUIKit/TUIChat/tuichat')
```

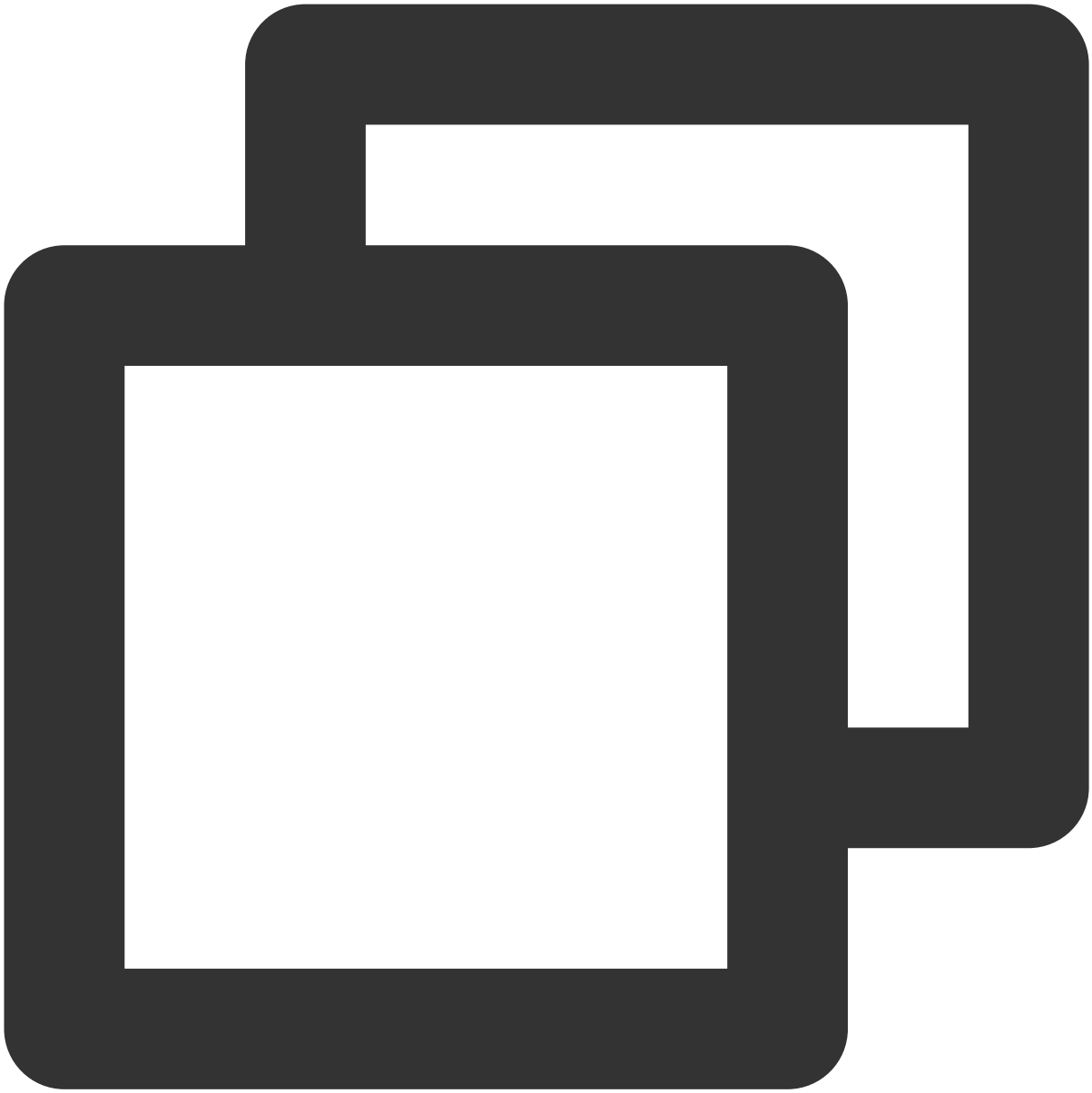
3. 在 App 模块中添加 TUIChat 依赖：



```
api project(':tuichat')
```

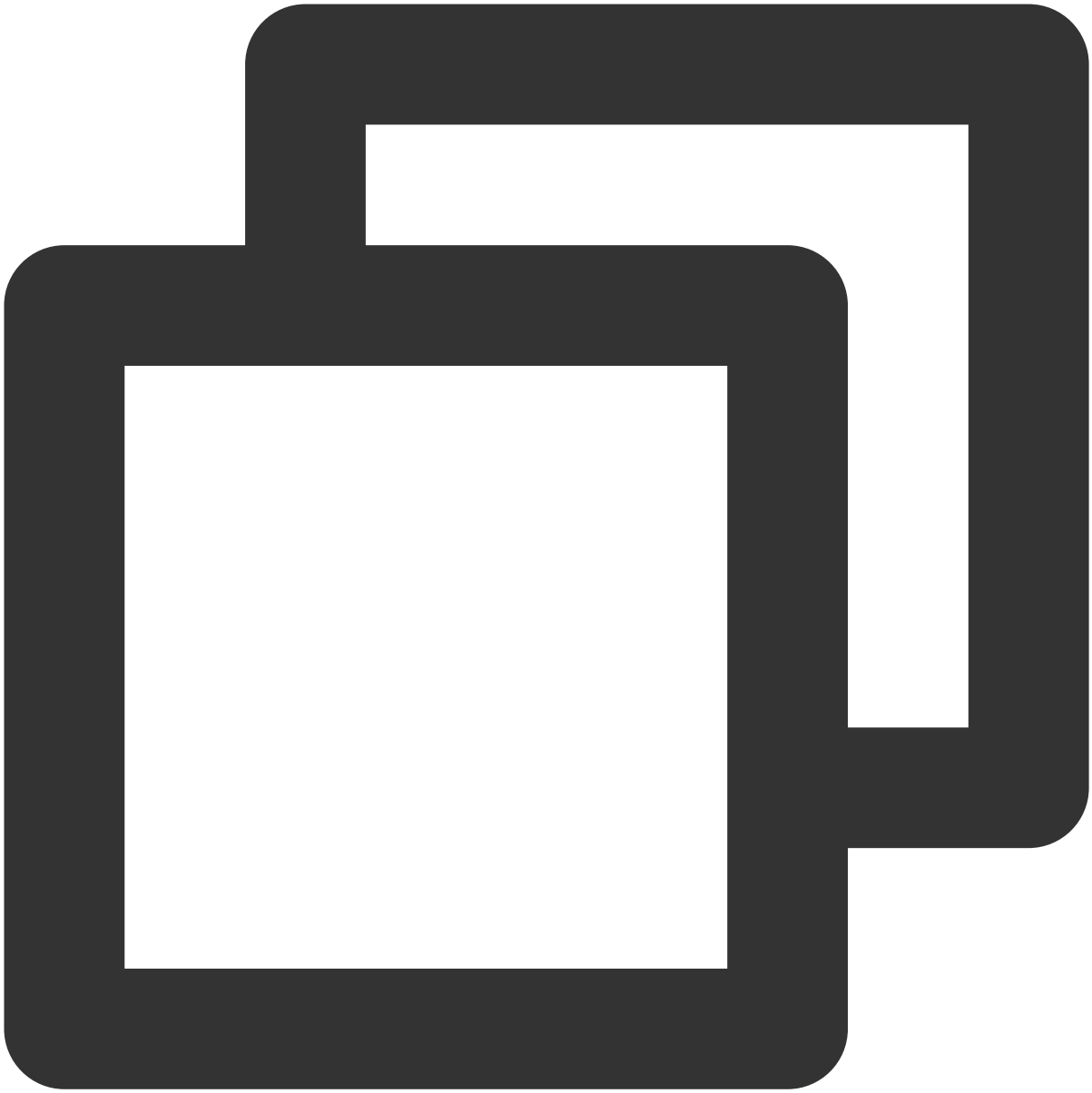
4.

添加 maven 仓库 和 Kotlin 支持，在 root 工程的 build.gradle 文件（与 settings.gradle 同级）中添加：



```
buildscript {
 repositories {
     mavenCentral()
 }
 dependencies {
     classpath 'com.android.tools.build:gradle:7.0.0'
     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.31"
 }
}
```

如果您使用 Gradle 8.x，则需要添加以下代码。



```
buildscript {
 repositories {
     mavenCentral()
     maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
 }
 dependencies {
     classpath 'com.android.tools.build:gradle:8.0.2'
     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.0"
 }
}
```

构建聊天界面

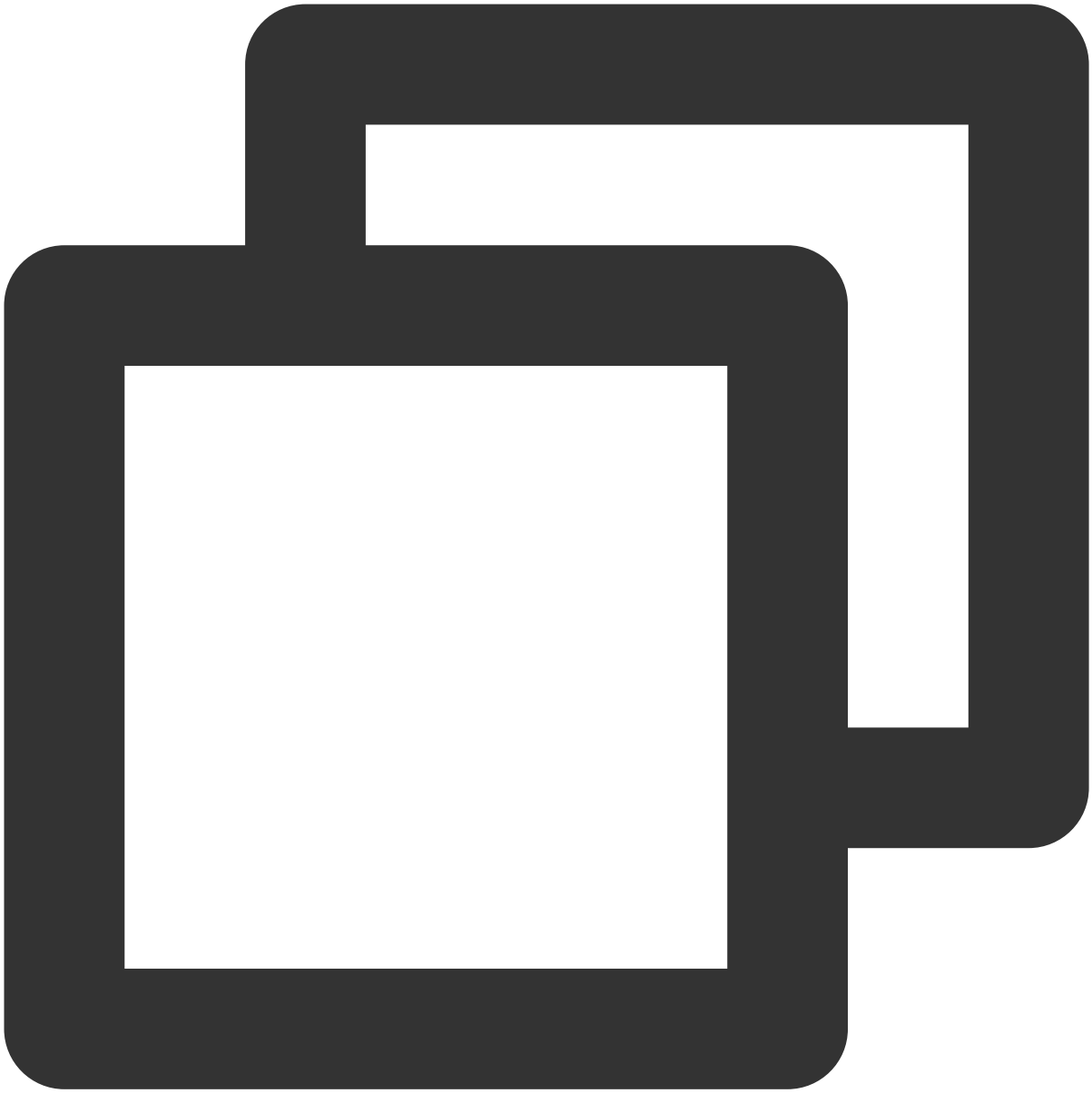
集成 TUIChat 完成后，如果您想要继续构建聊天界面，请参考文档：构建聊天界面。

常见问题

提示 "Manifest merger failed : Attribute application@allowBackup value=(true) from AndroidManifest.xml" 如何处理？

IM SDK 中默认 `allowBackup` 的值为 `false` ，表示关闭应用的备份和恢复功能。

您可以在您的 `AndroidManifest.xml` 文件中删除 `allowBackup` 属性，表示关闭备份和恢复功能；也可以在 `AndroidManifest.xml` 文件的 application 节点中添加 `tools:replace="android:allowBackup"` 表示覆盖 IM SDK 的设置，使用您自己的设置。 例如：



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.tencent.qcloud.tuikit.myapplication">

    <application
        android:allowBackup="true"
        android:name=".MApplication"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

```
        android:theme="@style/Theme.MyApplication"
        tools:replace="android:allowBackup">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

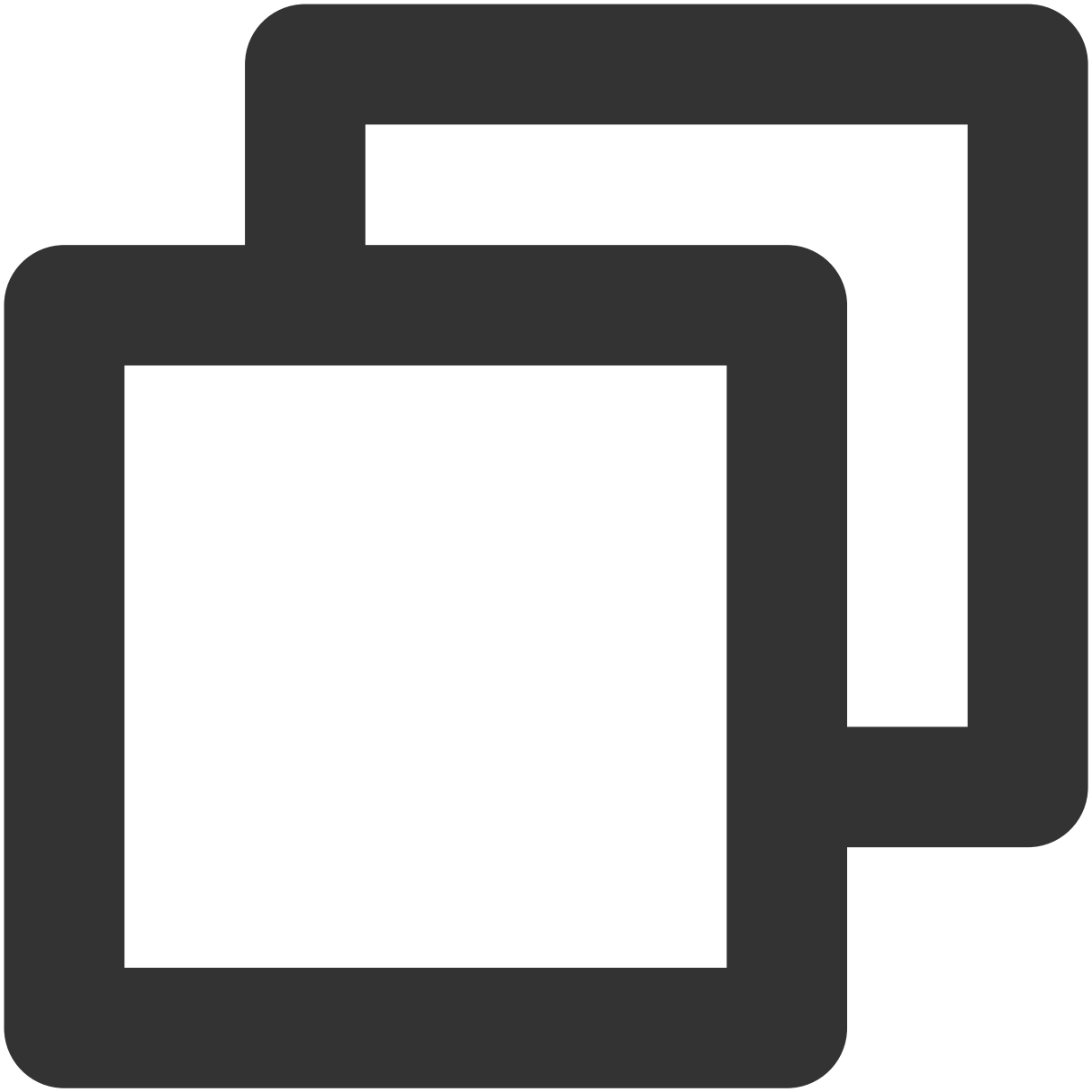
</manifest>
```

提示 "NDK at /Users/*/Library/Android/sdk/ndk-bundle did not have a source.properties file" 如何处理？**

只需要在 local.properties 文件中加入您的 NDK 路径，例如：`ndk.dir=/Users/***/Library/Android/sdk/ndk/16.1.4479499`

提示 "Cannot fit requested classes in a single dex file" 如何处理？

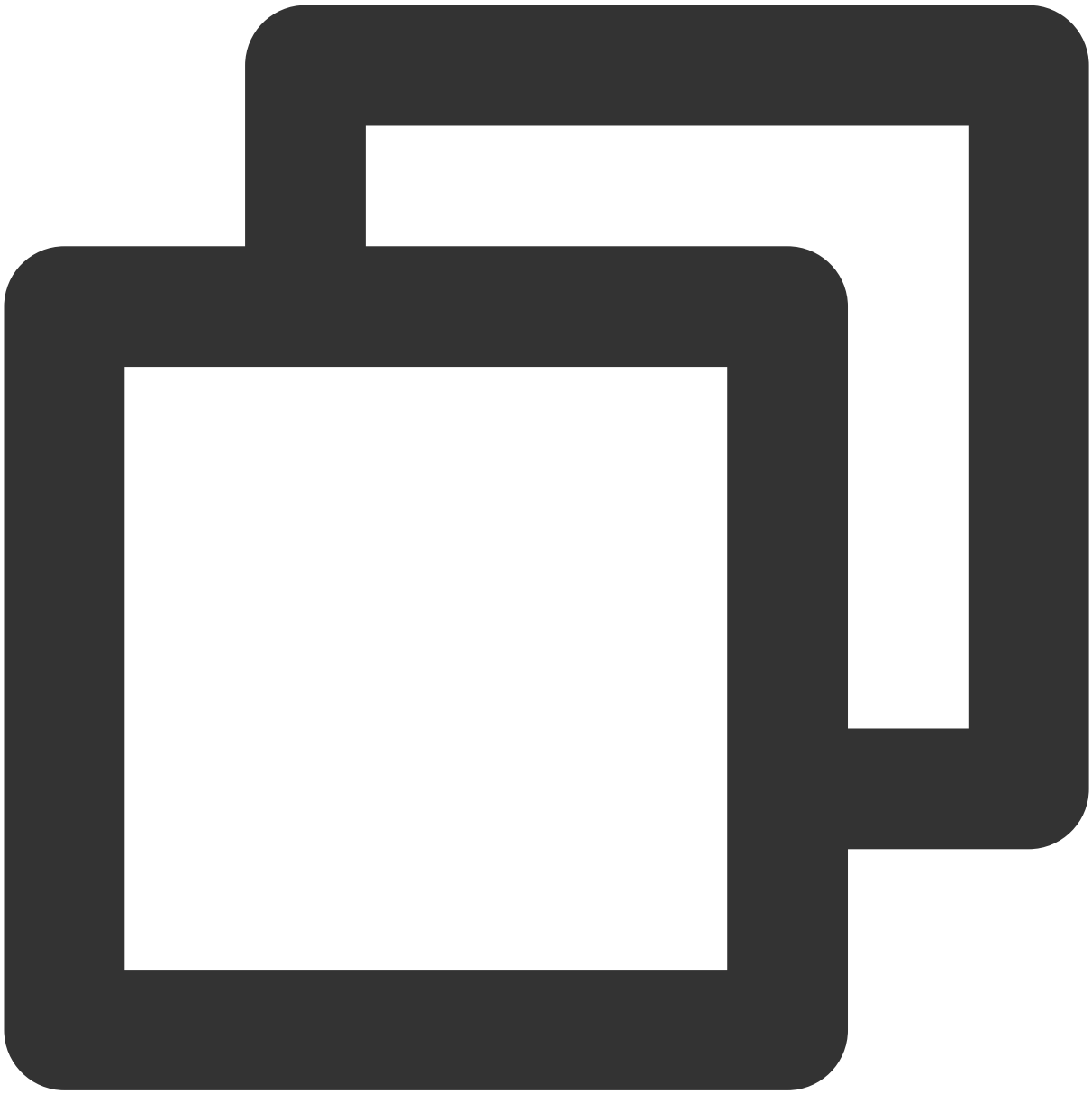
出现此问题可能是您的 API 级别设置比较低，需要在 App 的 build.gradle 文件中开启 `MultiDex` 支持，添加 `multiDexEnabled true` 和对应依赖：



```
android {
    defaultConfig {
        ...
        minSdkVersion 19
        targetSdkVersion 30
        multiDexEnabled true
    }
    ...
}
dependencies {
    implementation "androidx.multidex:multidex:2.0.1"
```

```
}
```

同时，在您的 Application 文件中添加以下代码：



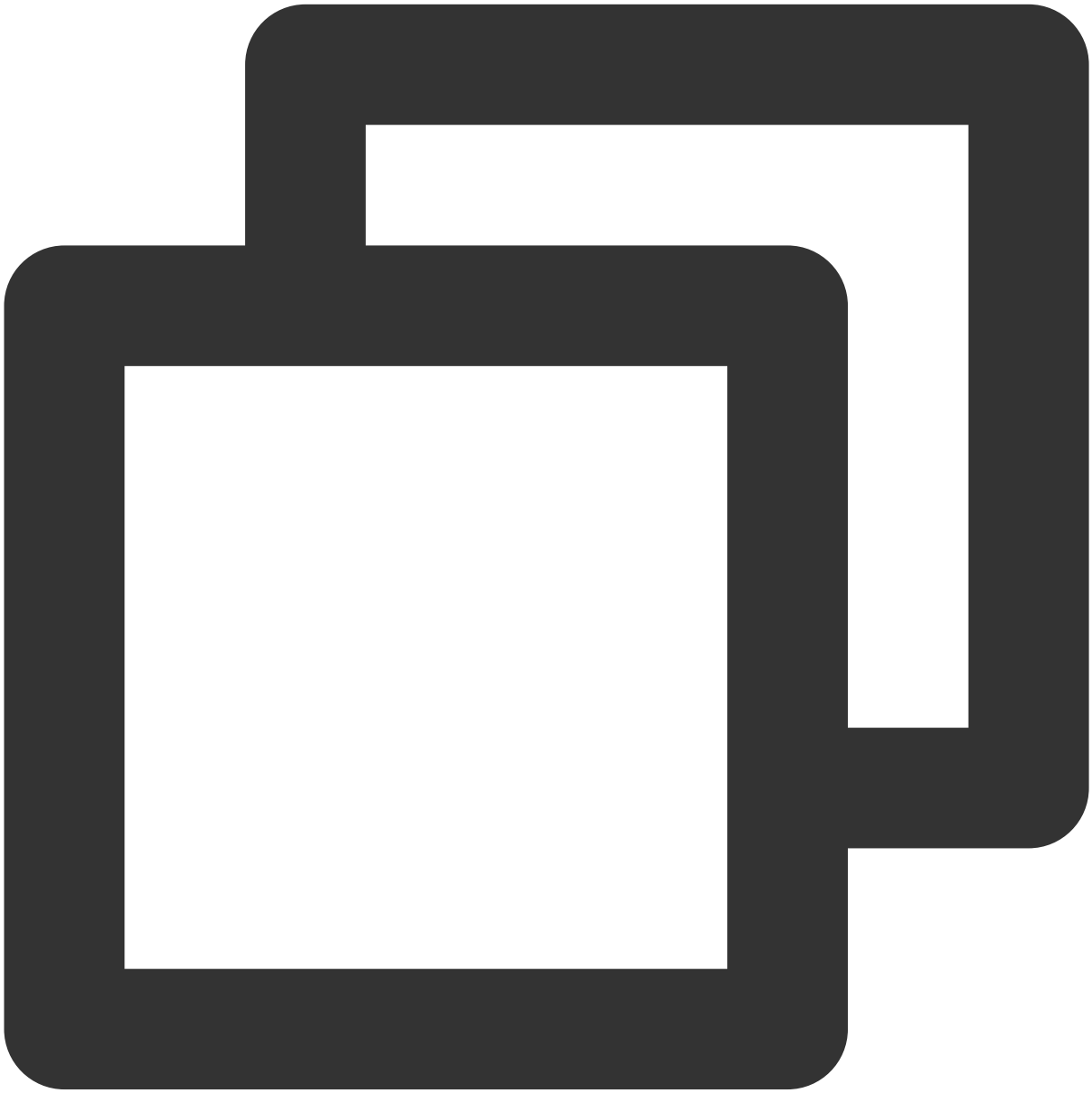
```
public class MyApplication extends SomeOtherApplication {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        MultiDex.install(this);
    }
}
```

提示 "Plugin with id 'kotlin-android' not found." 如何处理？

因为 `TUIChat` 组件使用了 Kotlin 代码，所以需要添加 Kotlin 构建插件。请参考上文源码集成第 4 步。

Debug 版本的 App 功能正常，Release 版本的 App 功能出现异常 ？

出现此问题很大概率是混淆导致的，请尽量不要混淆 TUIKit 。可以添加如下混淆规则：



```
# Avoid deleting code logic
-dontshrink
-dontoptimize
# Avoid aliasing TUIKit
```

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

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

iOS

最近更新时间：2024-06-24 17:24:35

本文将介绍如何集成 `TUIChat` 聊天组件。

说明：

从 5.7.1435 版本开始，TUIChat 支持了经典版 UI 组件。

从 6.9.3557 版本开始，TUIChat 新增了全新的简约版 UI 组件。

您可以根据需求自由选择经典版或简约版 UI 组件。

效果展示

TUIChat 提供了私信聊天（1V1）和群聊（Group）功能，支持对消息的多种操作，例如发送不同类型的消息、对消息长按点赞/回复/引用、查询消息已读回执详情等。您可以仅集成 TUIChat 到您的 App 中。聊天界面使用场景非常广泛，例如房产中介咨询、在线医疗问诊、电商在线客服、保险远程定损等。

界面效果如下图所示：

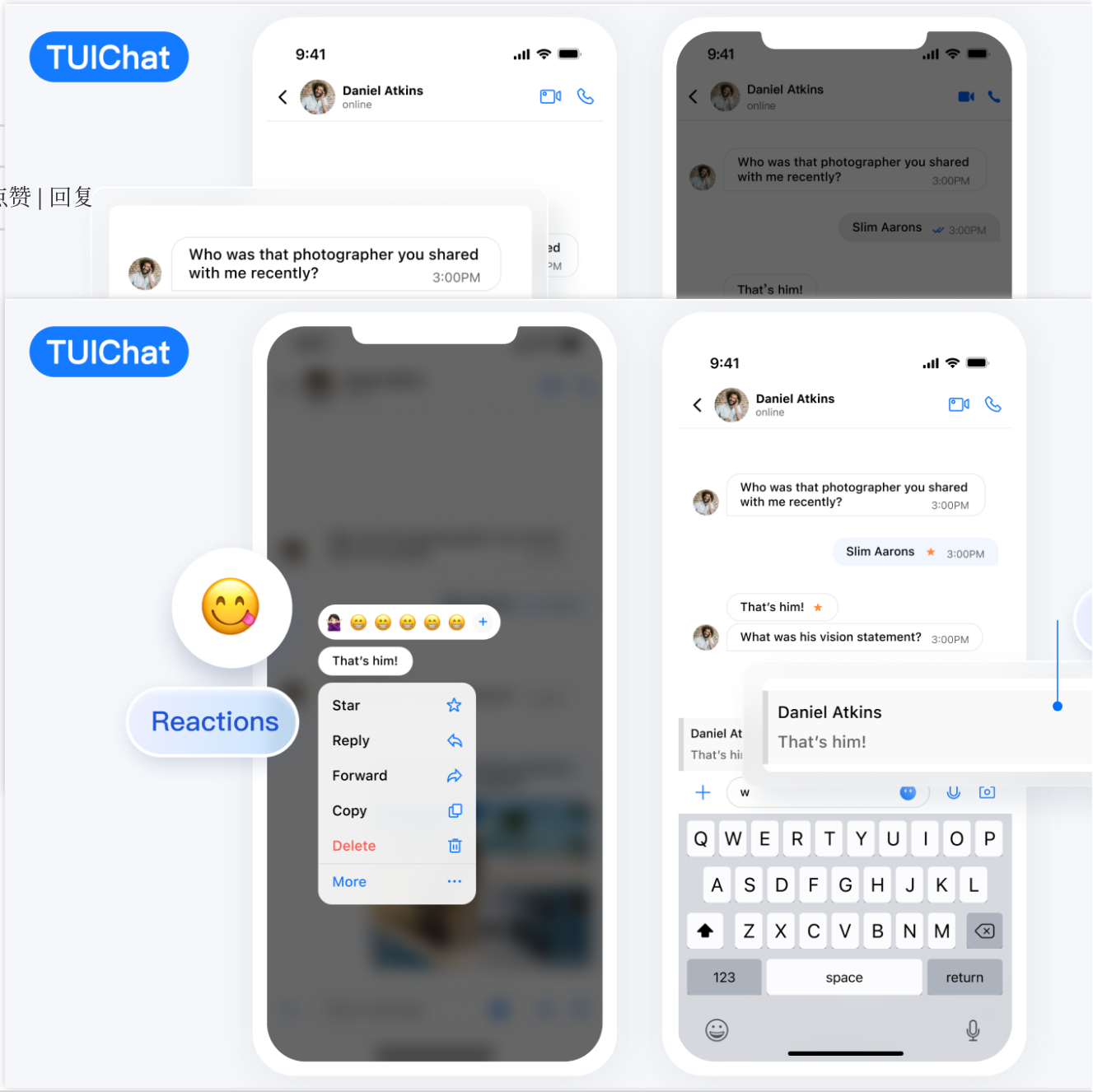
简约版

RTL 语言

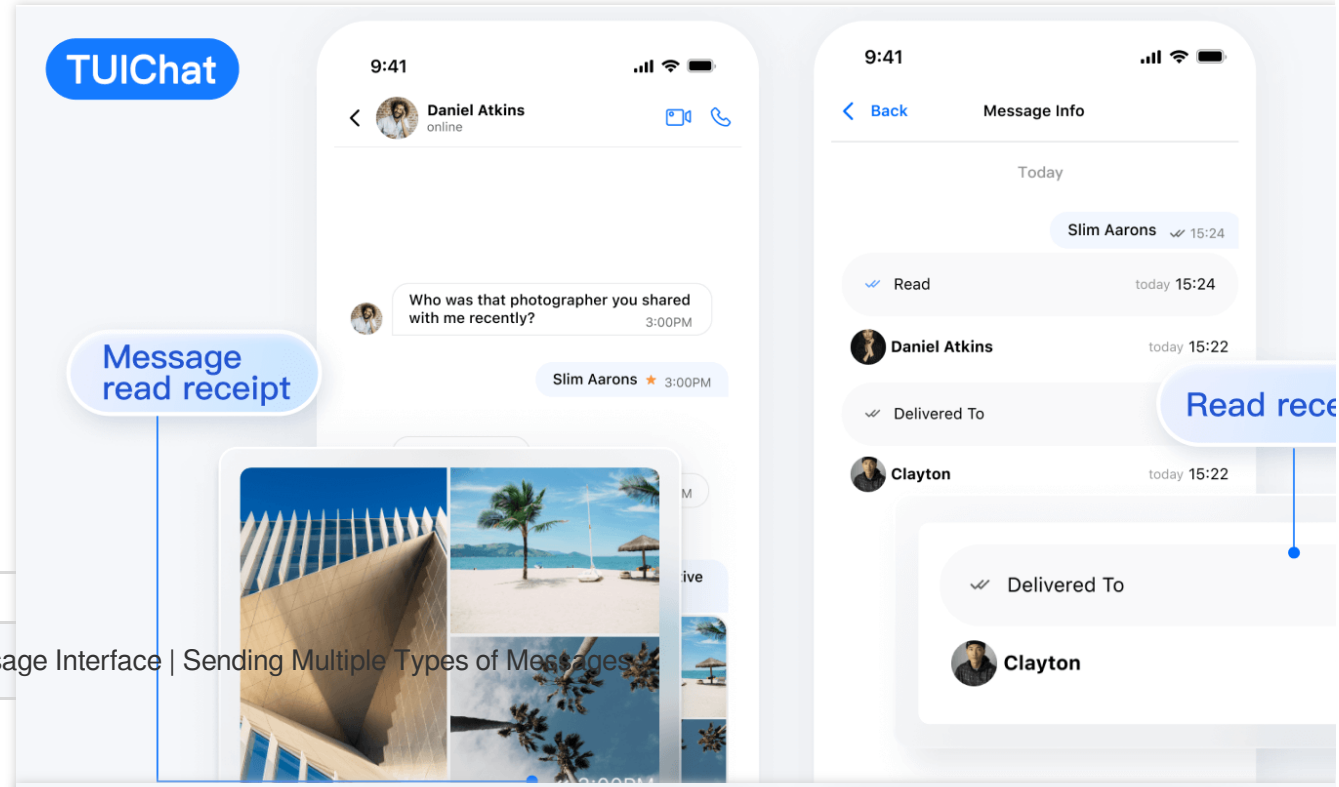
经典版

消息界面 | 发送多种类型消息

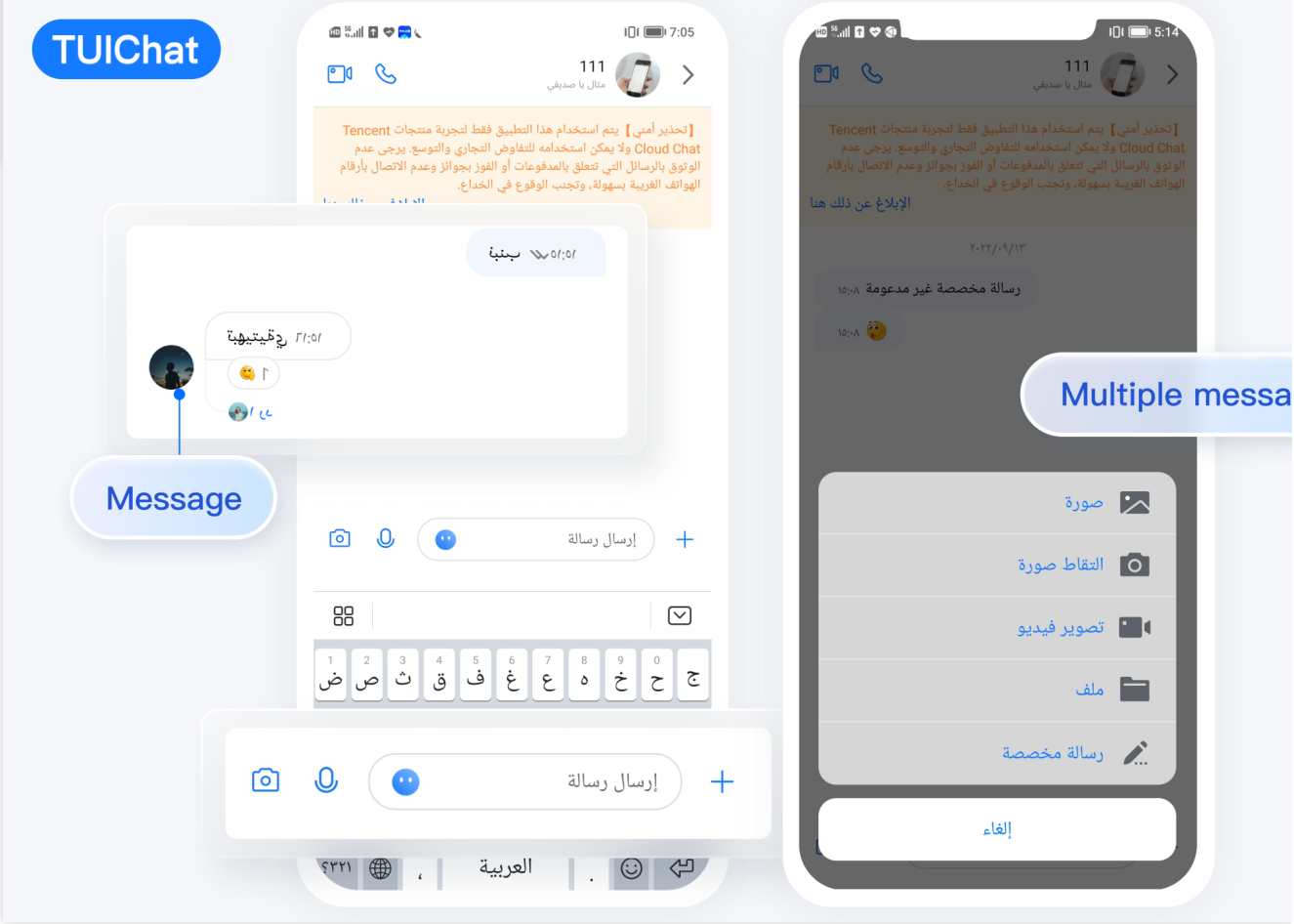
消息点赞 | 回复



消息已读回执 | 已读回执详情



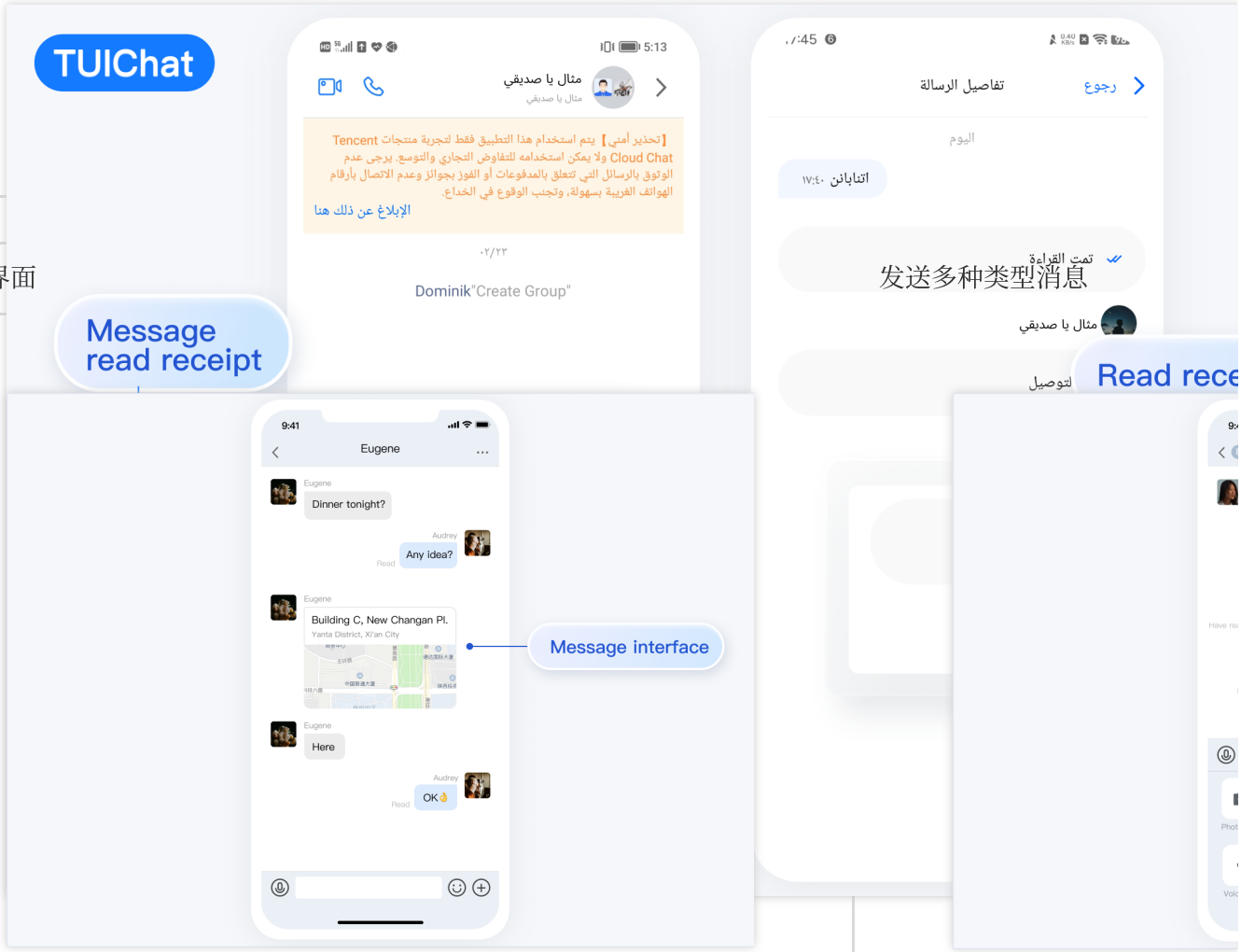
Message Interface | Sending Multiple Types of Messages



Message Like | Reply

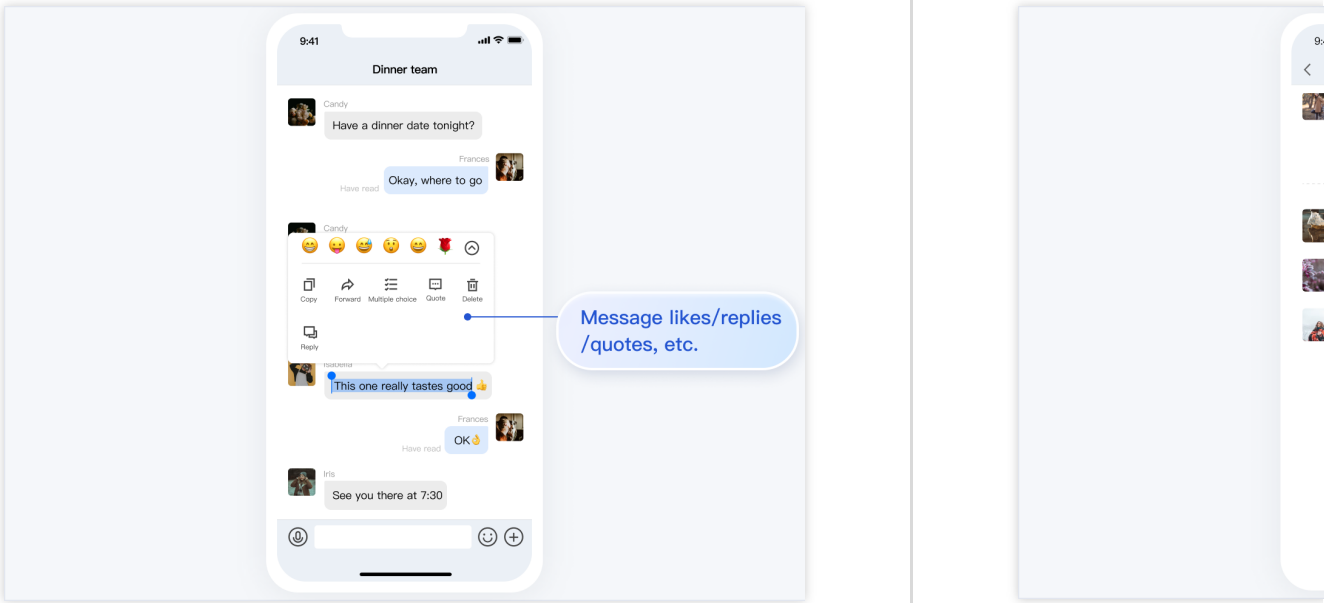
Message Read Receipt | Read Receipt Details

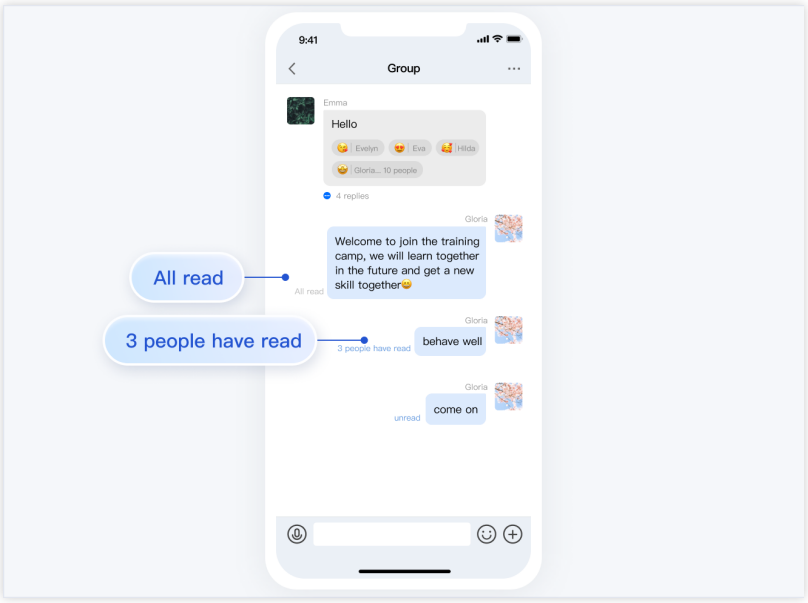
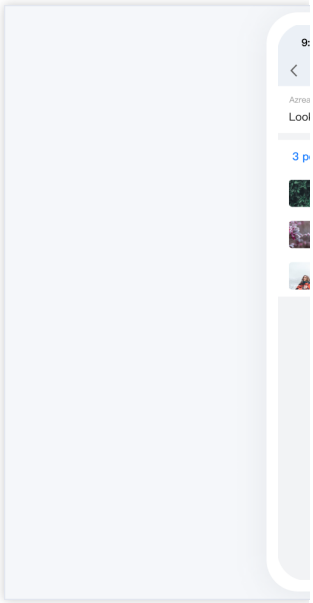
消息界面



消息点赞/回复/引用

消息回复详情



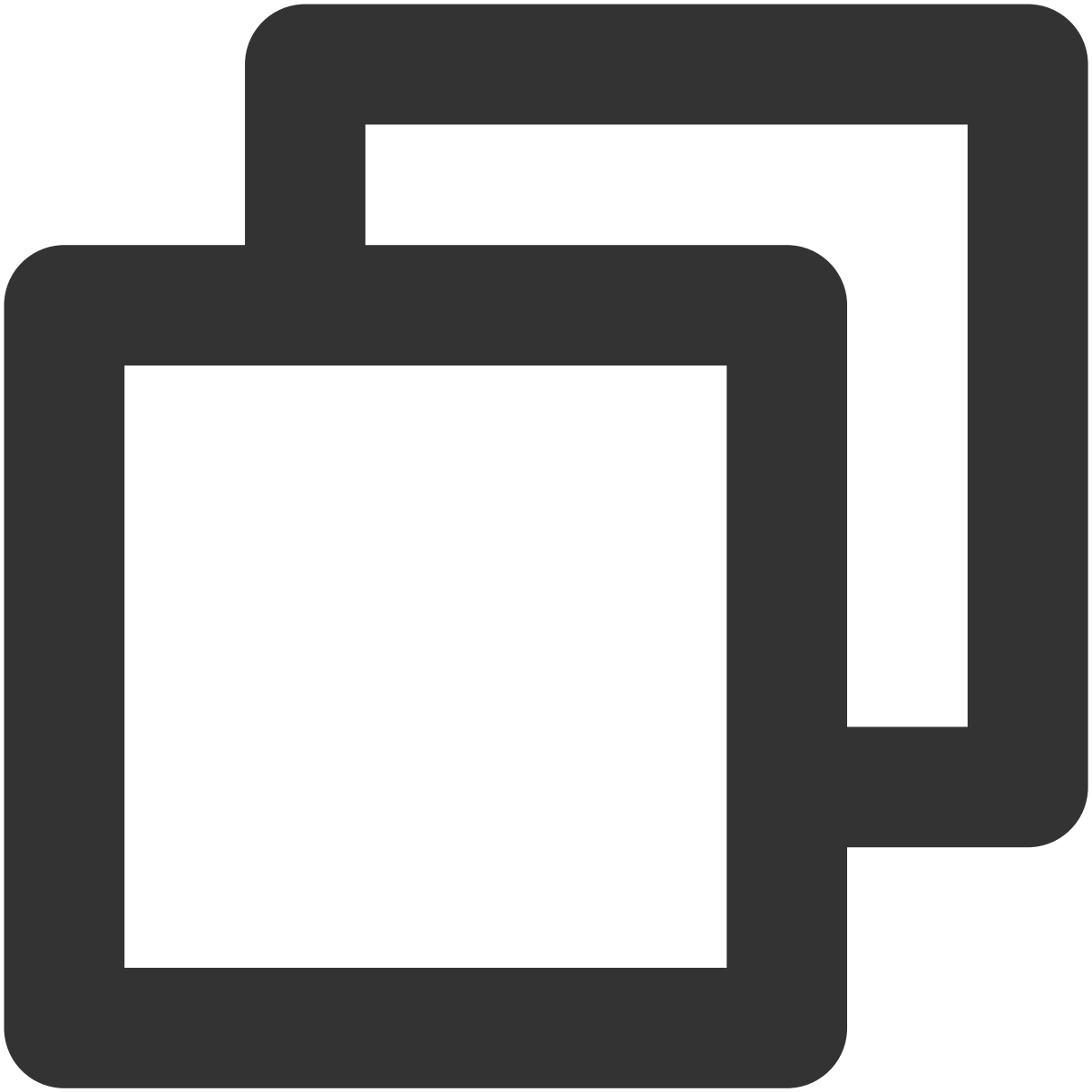
消息已读回执	已读回执详情
	

开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

CocoaPods 集成

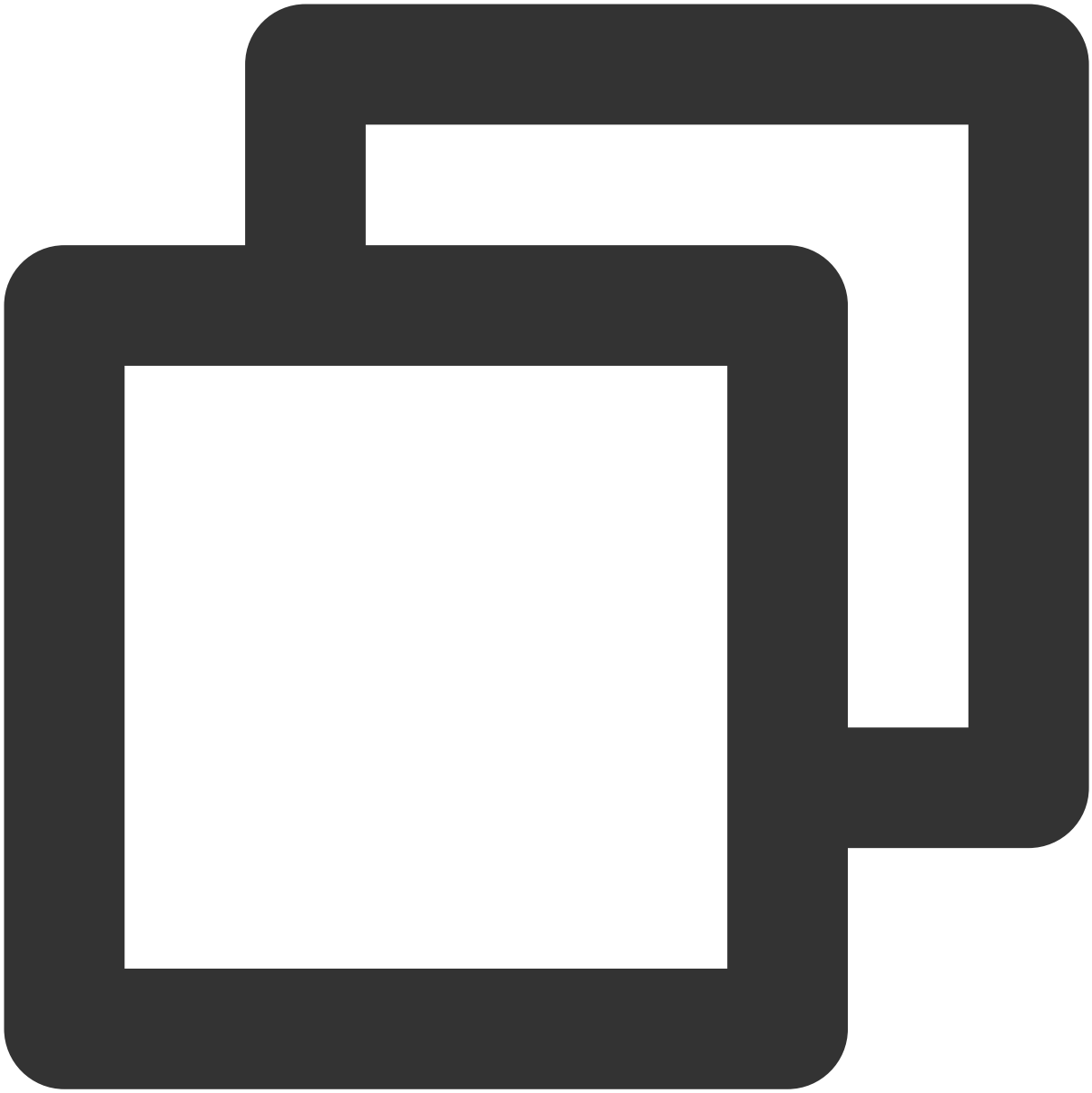
1. 安装 CocoaPods。
在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：



```
sudo gem install cocoapods
```

2. 创建 Podfile 文件。

进入项目所在路径输入以下命令行，之后项目路径下会出现一个 Podfile 文件。



```
pod init
```

3. 根据业务需求在 Podfile 中添加对应的 TUIChat 组件。您可以按需选择不同的 Podfile 集成方式：

远程 CocoaPods 集成

DevelopmentPods 本地集成

以上两种集成方式的优缺点如下表所示：

集成方式	适合场景	优点	缺点
远程 CocoaPods 集成	适合无源码修	当 TUIChat 有版本更新时，您	当您有源码修改，使用 `Pod`

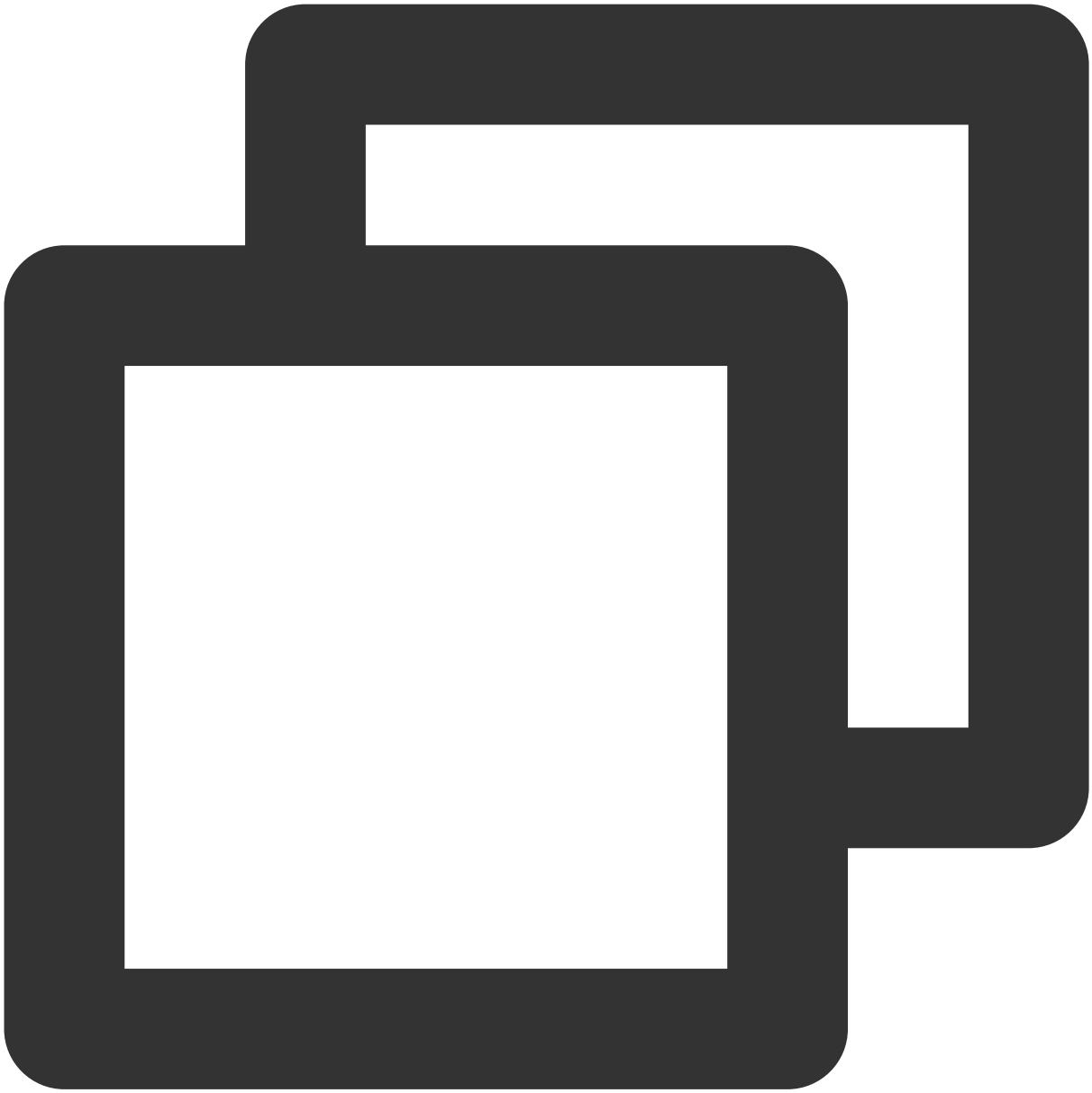
	改时的集成。	只需再次 `Pod update` 即可完成更新。	`update` 更新时，新版本的 TUIChat 会覆盖您的修改。
本地 DevelopmentPods 集成	适合有涉及源码自定义修改的客户。	当您有自己的 git 仓库时，可以跟踪修改。修改源码后，使用 `Pod update` 更新其他远程 Pod 库时，不会覆盖您的修改。	您需要手动将 TUIChat 源码覆盖您本地 TUIChat 文件夹进行更新。

远程 CocoaPods 集成

您可以在 Podfile 中添加 TUIChat 组件库：

简约版

经典版



```
# Uncomment the next line to define a global platform for your project.
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
# Prevent `*.xcassets` in TUIChat components from conflicting with your project.
install! 'cocoapods', :disable_input_output_paths => true

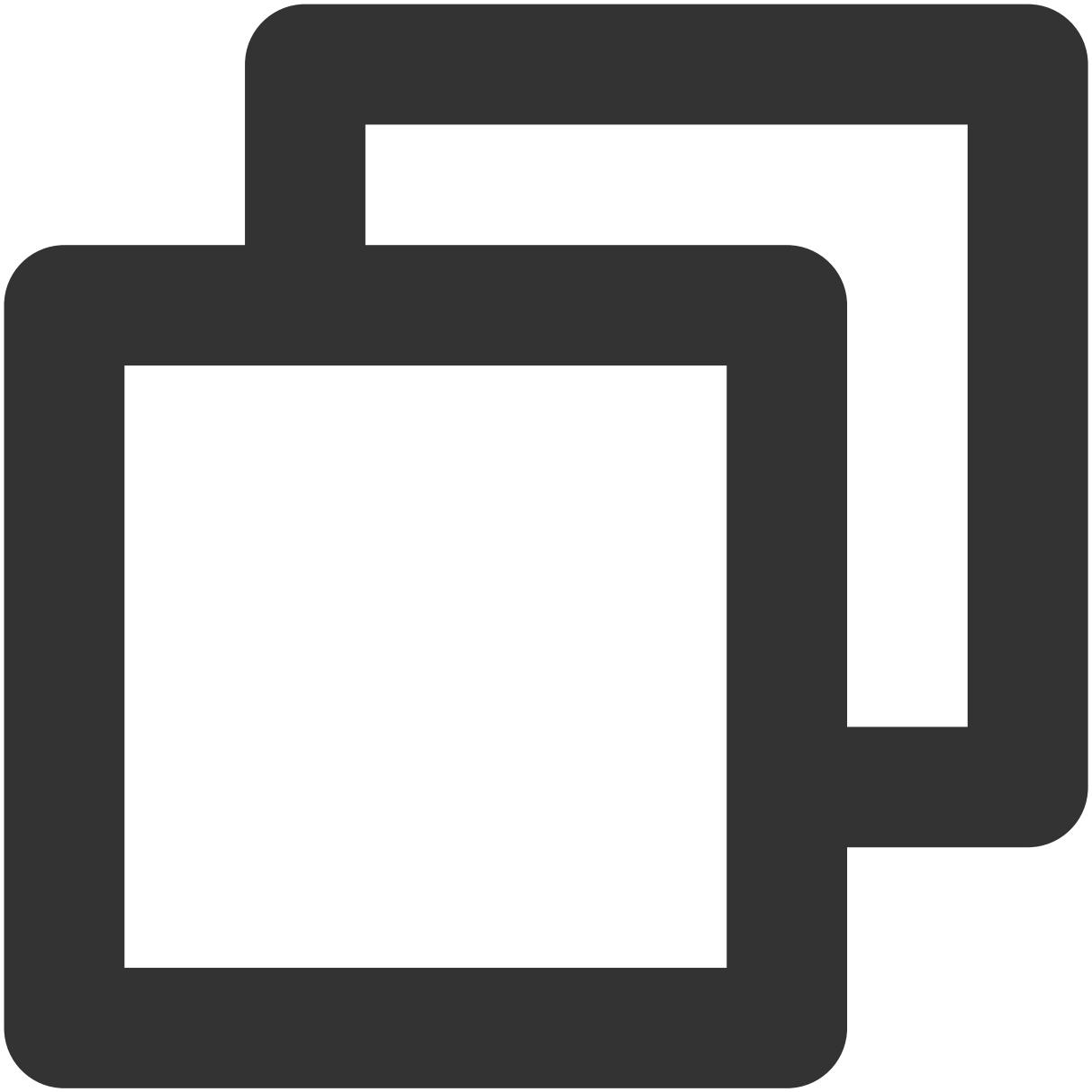
# Replace `your_project_name` with your actual project name.
target 'your_project_name' do
  use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers, the Po
```

```
  # use_modular_headers!

  # Integrate the chat feature.
  pod 'TUIChat/UI_Minimalist'
end

#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```



```
# Uncomment the next line to define a global platform for your project.
source 'https://github.com/CocoaPods/Specs.git'
# Prevent `*.xcassets` in TUIChat components from conflicting with your project.
install! 'cocoapods', :disable_input_output_paths => true

# Replace your_project_name with your actual project name.
target 'your_project_name' do
  use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers, the Po
  # use_modular_headers!
```

```
  # Integrate the chat feature.
  pod 'TUIChat/UI_Classic'
end

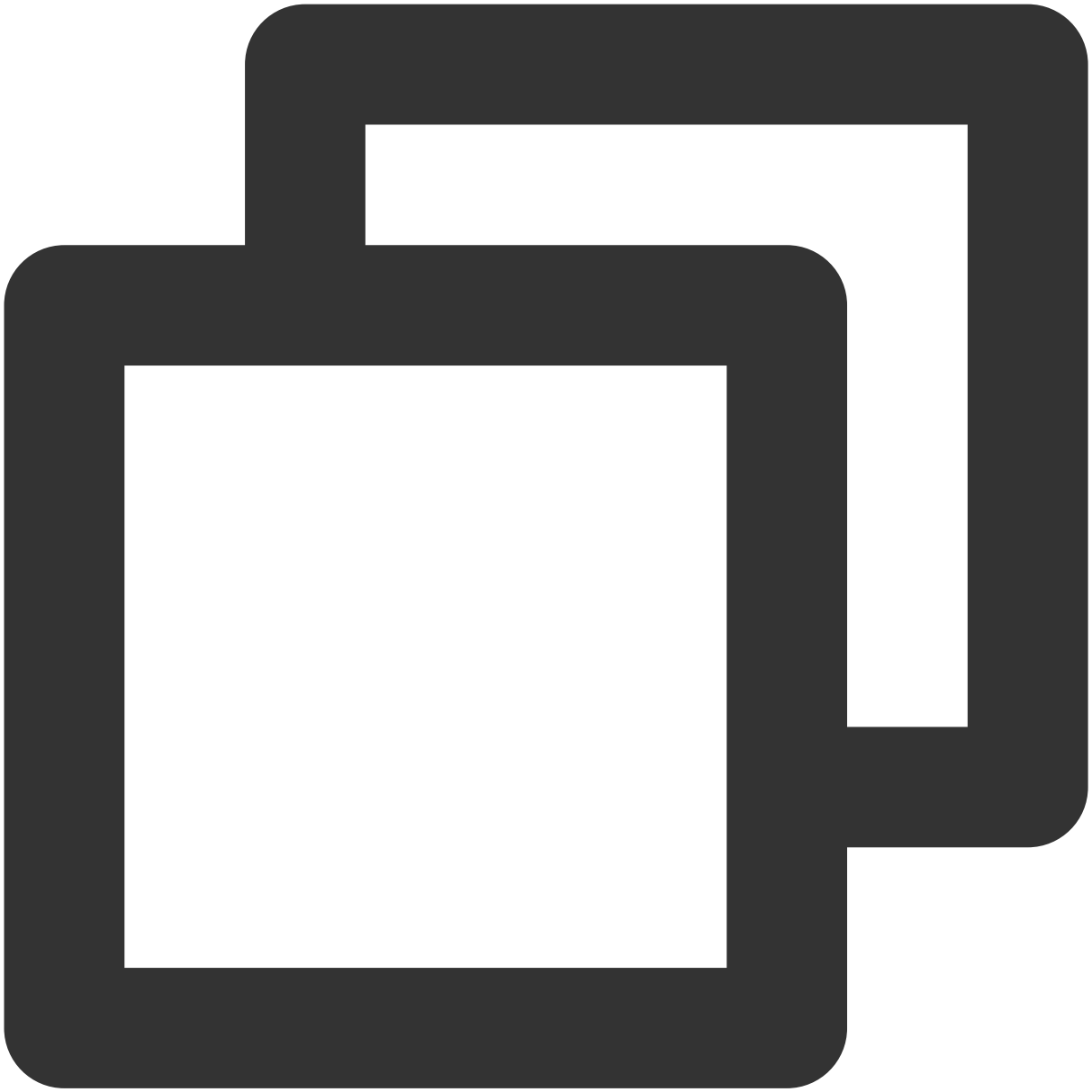
#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```

说明：

如果您直接 `pod 'TUIChat'`，不指定经典版或简约版，默认会集成两套版本 UI 组件。

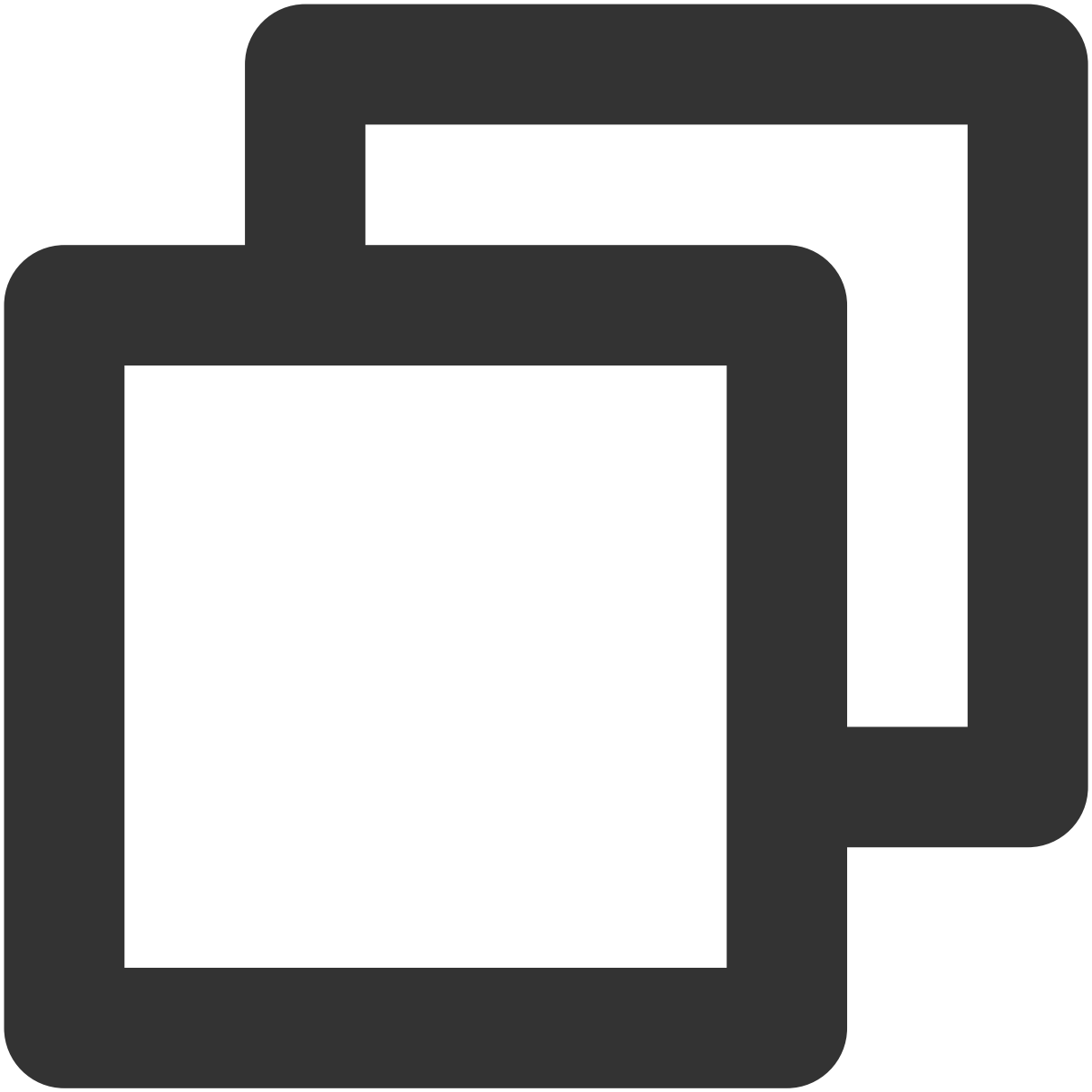
如果您使用的是 Swift，请开启 `use_modular_headers!` ，并将头文件引用改成 @import 模块名形式引用。

Podfile 修改完毕后，执行以下命令，安装 TUIChat 组件。



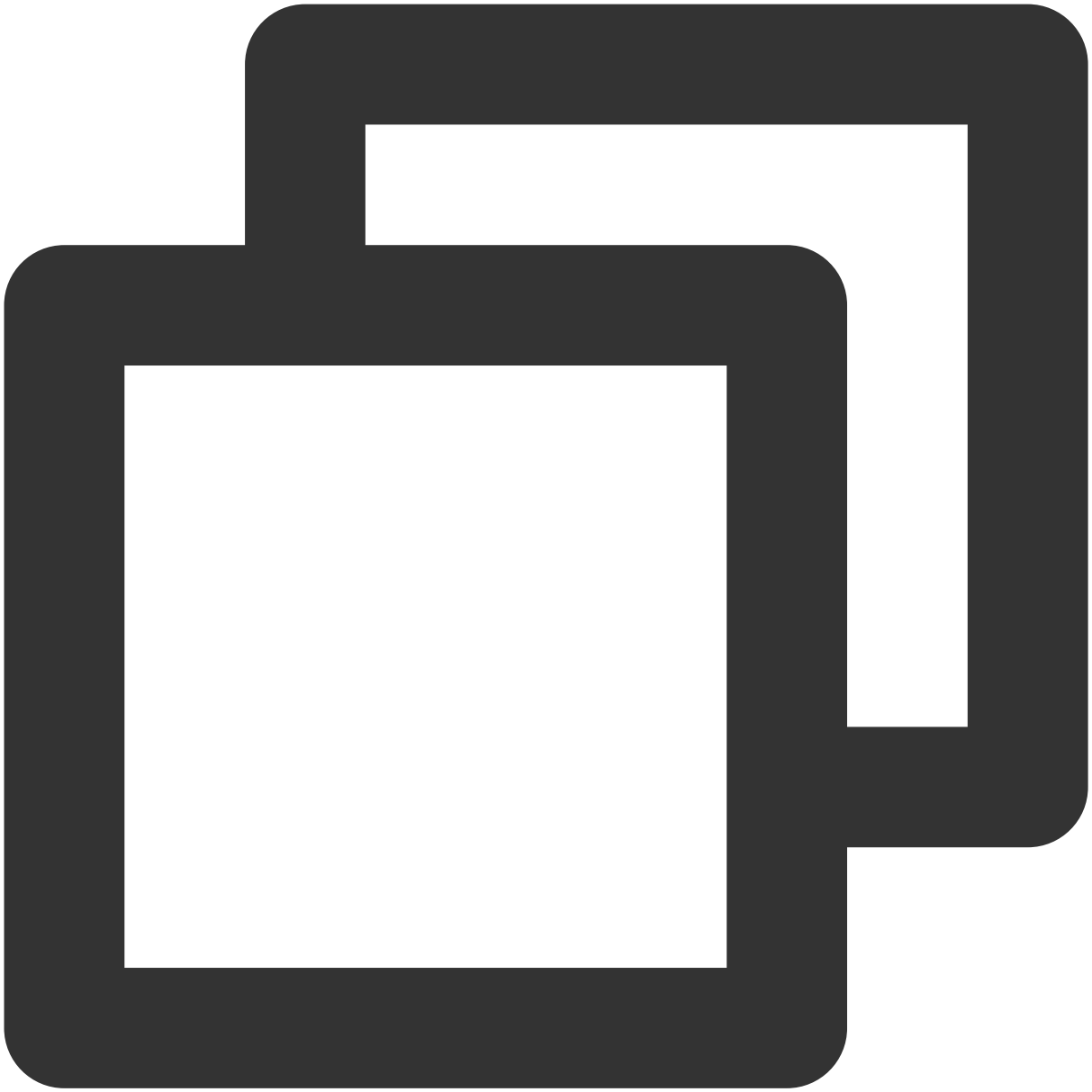
```
pod install
```

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



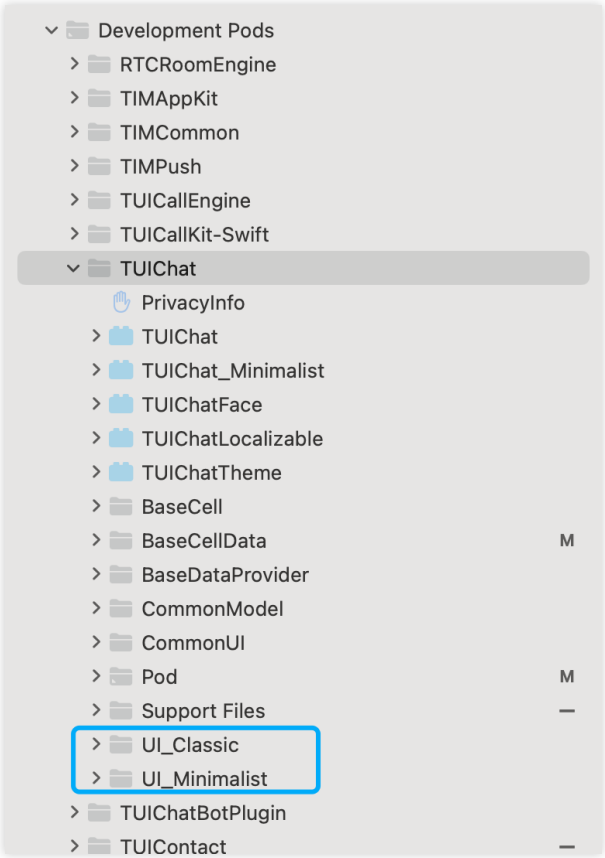
```
pod repo update
```

之后执行以下命令，更新组件库的 Pod 版本。



```
pod update
```

集成 TUIChat 组件后的项目结构：

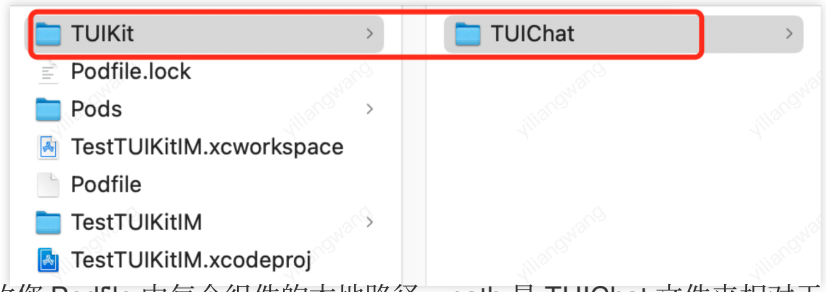


注意：

若您操作遇到错误，可查阅文末的常见问题。

本地 DevelopmentPods 源码集成

1. 从 GitHub 下载 TUIChat 源码。直接拖入您的工程目录下，如：`TestTUIKitIM/TUIKit/TUIChat`。



2. 修改您 Podfile 中每个组件的本地路径。path 是 TUIChat 文件夹相对于您工程 Podfile 文件的位置，常见的有：

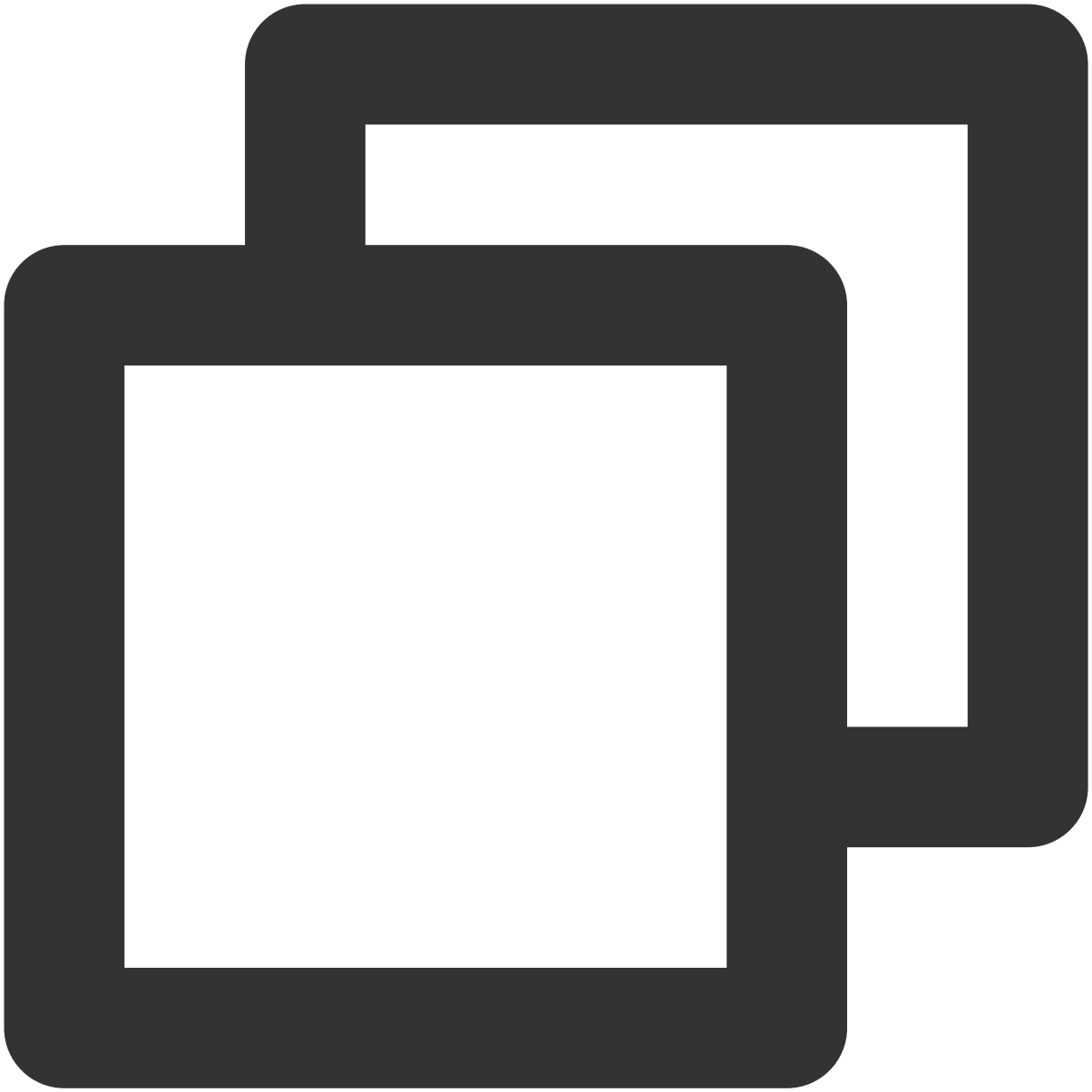
TUIChat 文件夹位于您工程 Podfile 文件**父目录**：`pod 'TUIChat', :path => "../TUIKit/TUIChat"`

TUIChat 文件夹位于您工程 Podfile 文件**当前目录**：`pod 'TUIChat', :path => "/TUIKit/TUIChat"`

TUIChat 文件夹位于您工程 Podfile 文件**子目录**：`pod 'TUIChat', :path => "./TUIKit/TUIChat"`

以 TUIChat 文件夹位于您工程 Podfile 文件父目录为例：

Development Podfile



```
# Uncomment the next line to define a global platform for your project.
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
install! 'cocoapods', :disable_input_output_paths => true

# Replace `your_project_name` with your actual project name.
target 'your_project_name' do
  # Uncomment the next line if you're using Swift or would like to use dynamic fram
  use_frameworks!
  use_modular_headers!
```

```
  # Note: When using the local integration solution, upgrade by downloading the lat
  # and placing it in the designated local directory, such as /TIMSDK/ios/TUIKit/TU
  # Note: When private modifications conflict with remote changes, manual merging i

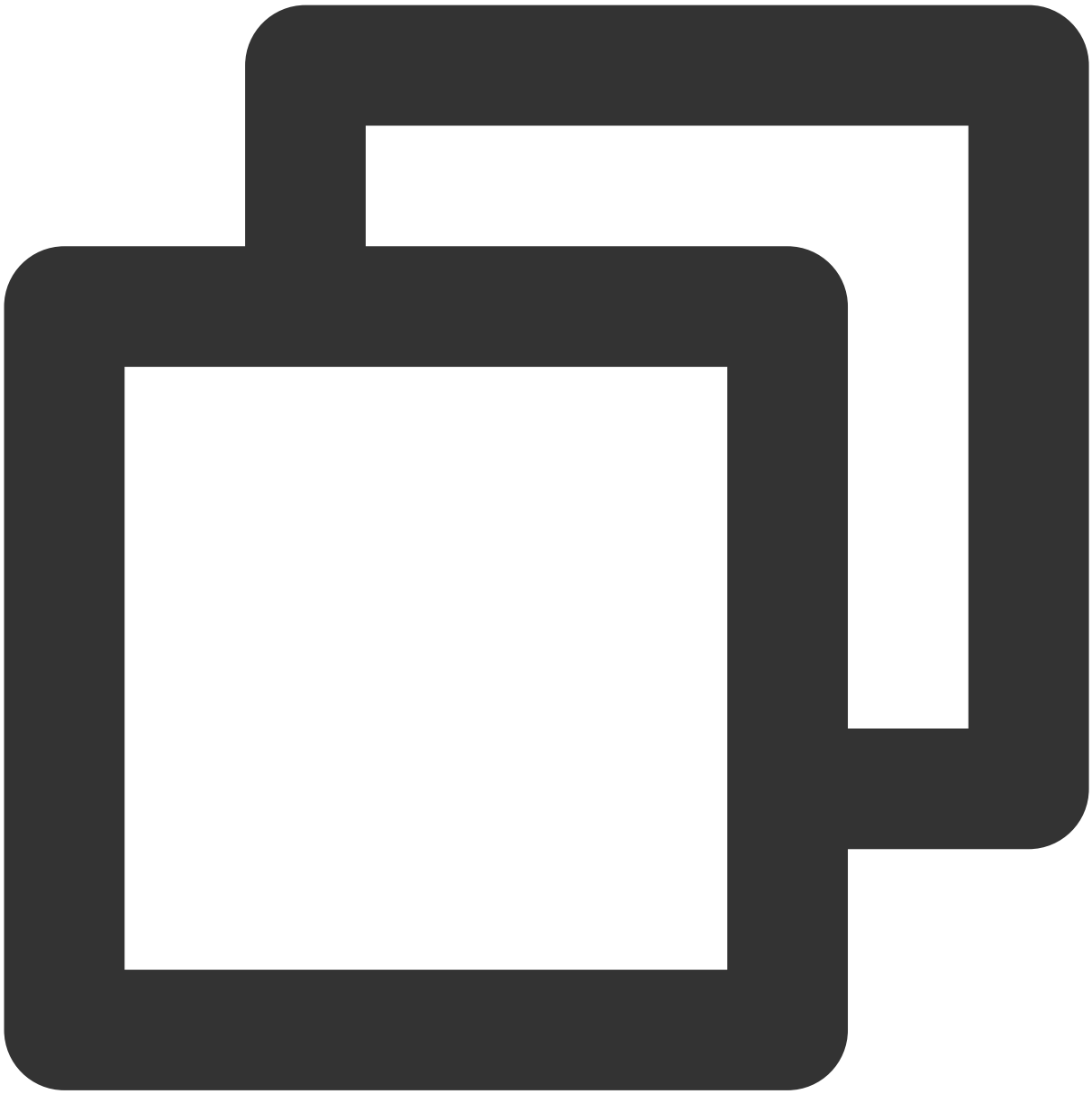
  # Integrate the basic library (required).
  pod 'TUICore', :path => "../TUIKit/TUICore"
  pod 'TIMCommon', :path => "../TUIKit/TIMCommon"

  # Integrate the chat feature.
  pod 'TUIChat', :path => "../TUIKit/TUIChat"

  # Other Pod
  pod 'MJRefresh'
  pod 'Masonry'
end

#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            #Fix Xcode14 Bundle target error
            config.build_settings['EXPANDED_CODE_SIGN_IDENTITY'] = ""
            config.build_settings['CODE_SIGNING_REQUIRED'] = "NO"
            config.build_settings['CODE_SIGNING_ALLOWED'] = "NO"
            config.build_settings['ENABLE_BITCODE'] = "NO"
            config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = "13.0"
            #Fix Xcode15 other links  flag  -ld64
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' ' -f2
            if xcode_version >= 15
              xcconfig_path = config.base_configuration_reference.real_path
              xcconfig = File.read(xcconfig_path)
              if xcconfig.include?("OTHER_LDFLAGS") == false
                xcconfig = xcconfig + "\\n" + 'OTHER_LDFLAGS = $(inherited) "-ld64"
              else
                if xcconfig.include?("OTHER_LDFLAGS = $(inherited)") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS", "OTHER_LDFLAGS = $(inher
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)", 'OTHER_LD
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```

3. Podfile 修改完毕后，执行以下命令，安装本地 TUIChat 组件。示例：



```
pod install
```

注意：

使用本地集成方案时，如需升级时可进入 Github-TUIChat 升级。

获取最新的组件代码，覆盖本地目录如：TIMSDK/iOS/TUIKit/TUIChat。

当私有化修改和远端有冲突时，需要手动合并，处理冲突。

TUIChat 插件需要依赖 TUICore 的版本，务必确保插件版本和 "../TUIKit/TUICore/TUICore.spec"中的 spec.version 一致。

若您操作遇到错误，可查阅文末的常见问题。

构建聊天界面

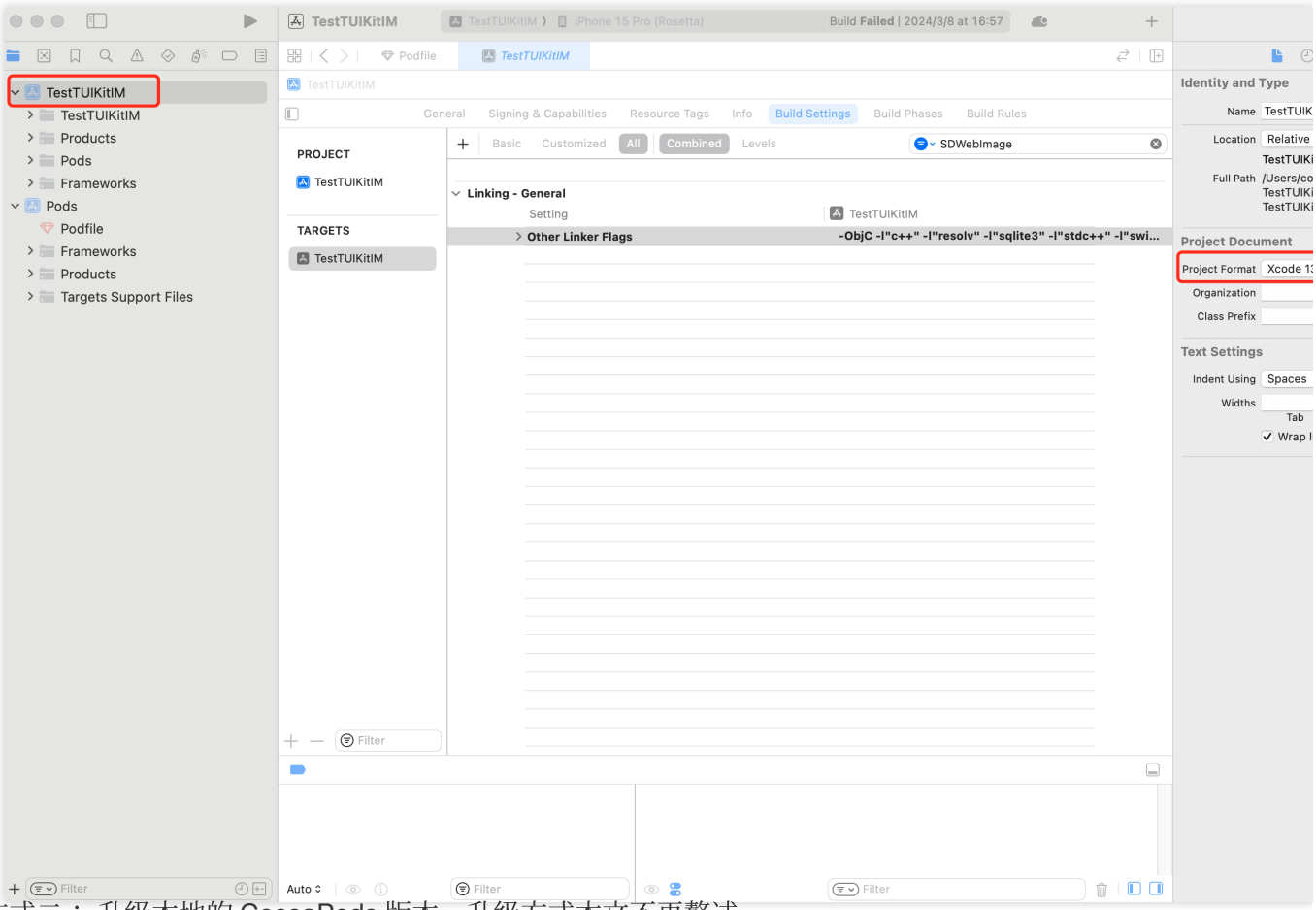
集成 TUIChat 完成后，如果您想要继续构建聊天界面，请参考文档：构建聊天界面。

常见问题

Xcode15 常见问题

集成时报错 [Xcodeproj] Unknown object version (60). (RuntimeError)

使用 Xcode15 创建新工程来集成 TUIChat 时，输入 `pod install` 后，可能会遇到此问题，原因是使用了较旧版本的 CocoaPods ，此时有两种解决办法：
解决方式一： 修改 Xcode 工程的 `Project Format` 版本为 Xcode13.0。



解决方式二：升级本地的 CocoaPods 版本，升级方式本文不再赘述。

Assertion failed: (false && "compact unwind compressed function offset doesn't fit in 24 bits"), function operator(), file Layout.cpp.

```
1  0x100d020f0
   ___Z13dispatchApplyIRKmZ15dispatchForEachIKN6mach_o13CompactUnwind11Diff24FixupEZNK2ld16LayoutExecutable18writeCompactUnwindE
  NSt3__14spanIhLm18446744073709551615EEEyRKNS7_13SectionLayoutEE4$_26EvNSA_IT_Lm18446744073709551615EEEmT0_EUlmE_EvOSG_SI__blo
  ck_invoke + 0
2  0x100d023b4  ld::LayoutExecutable::writeContentWithoutLinkEdit(std::__1::span<unsigned char, 18446744073709551615ul>,
  unsigned long long) + 700
3  0x100d08020  ld::LayoutExecutable::writeToFile(char const*) + 15248
4  0x100cba2e8  main + 9424
ld: Assertion failed: (false && "compact unwind compressed function offset doesn't fit in 24 bits"), function operator(), file
  Layout.cpp, line 5758.
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

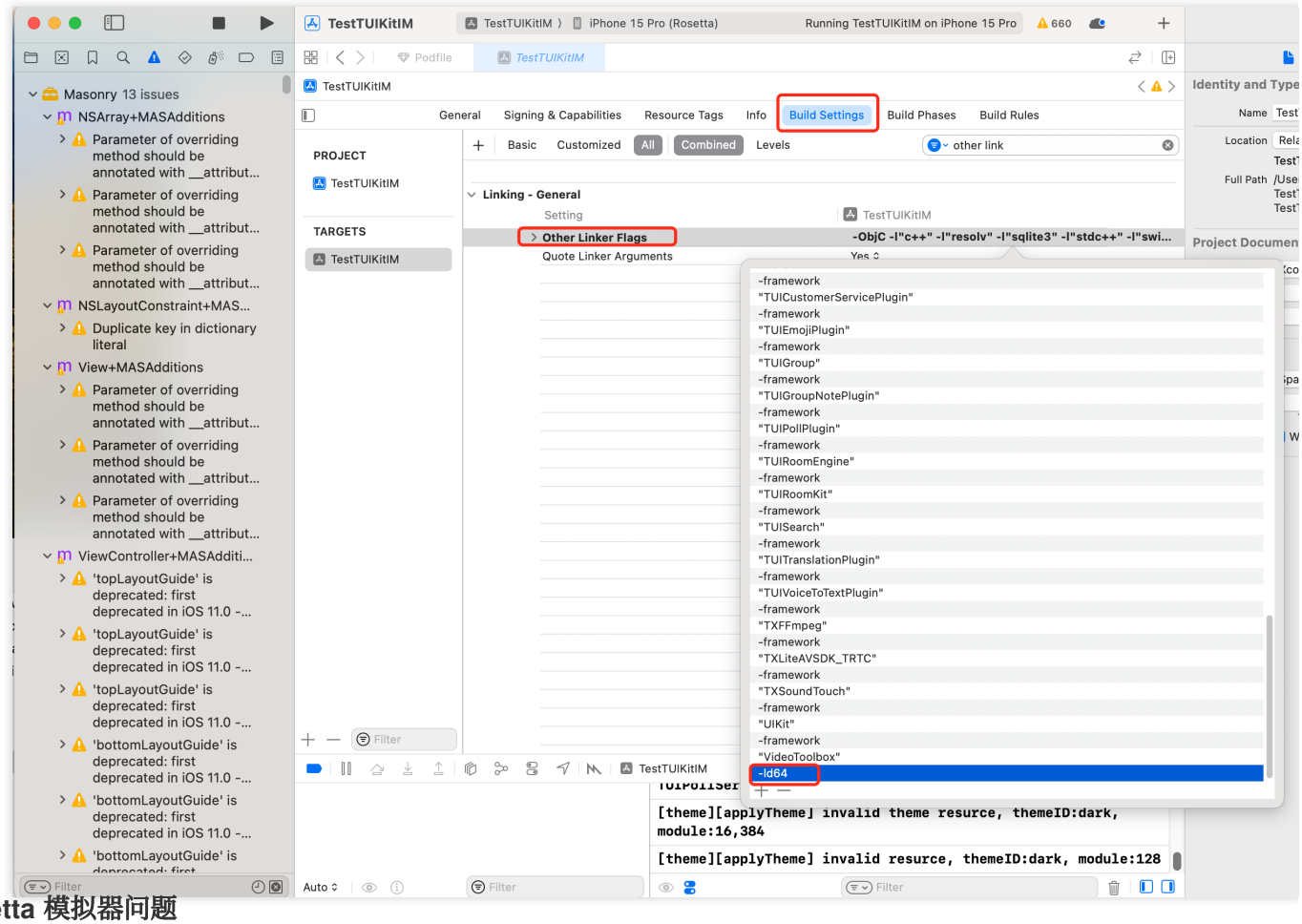
或是使用 XCode15 集成 TUIRoom 时，因最新链接器导致 TUIRoomEngine 的符号冲突，都属于该问题。

```
      /Users/cologne/Library/Developer/Xcode/DerivedData/TestTUIKitIM-gnoffncisnockmhjntjwqylvokac/Build/Products/
    Debug-iphonesimulator/XCFrameworkIntermediates/TUIRoomEngine/TRTC/TUIRoomEngine.framework/TUIRoomEngine[x86_64]
    [2](TUICommonDefineImpm.o)
  duplicate symbol '_OBJC_CLASS_$_TUIUserInfo' in:
      /Users/cologne/Library/Developer/Xcode/DerivedData/TestTUIKitIM-gnoffncisnockmhjntjwqylvokac/Build/Products/
    Debug-iphonesimulator/XCFrameworkIntermediates/TUIRoomEngine/TRTC/TUIRoomEngine.framework/TUIRoomEngine[x86_64]
    [9](TUIRoomDefineImpm.o)
      /Users/cologne/Library/Developer/Xcode/DerivedData/TestTUIKitIM-gnoffncisnockmhjntjwqylvokac/Build/Products/
    Debug-iphonesimulator/XCFrameworkIntermediates/TUIRoomEngine/TRTC/TUIRoomEngine.framework/TUIRoomEngine[x86_64]
    [4](TUIRoomDefineImpl.o)
  ld: 83 duplicate symbols
  clang: error: linker command failed with exit code 1 (use -v to see invocation)

  83 duplicate symbols
  Linker command failed with exit code 1 (use -v to see invocation)
  Activity Log Complete  2024/3/8, 14:53  7.6 seconds
  2 errors, 1113 warnings
```

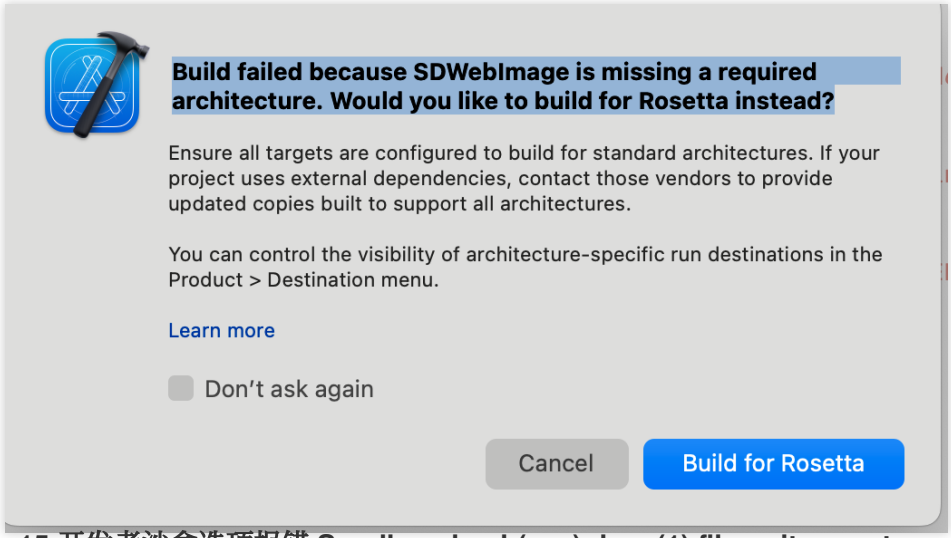
解决方式：修改链接器配置。在 `Build Settings` 中的 `Other Linker Flags` 中添加 `-ld64` 。

官方资料： https://developer.apple.com/forums/thread/735426

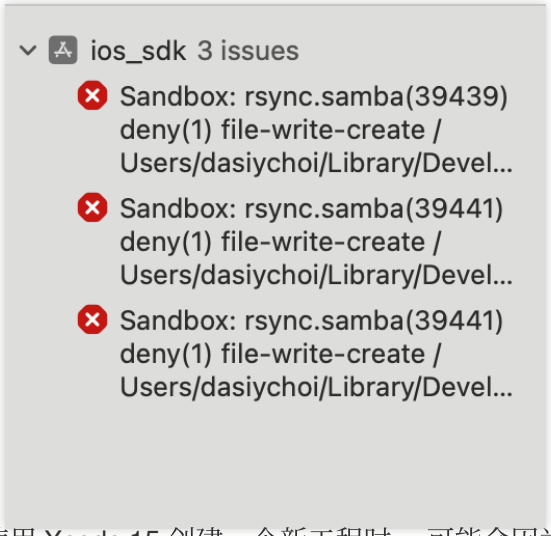


Rosetta 模拟器问题

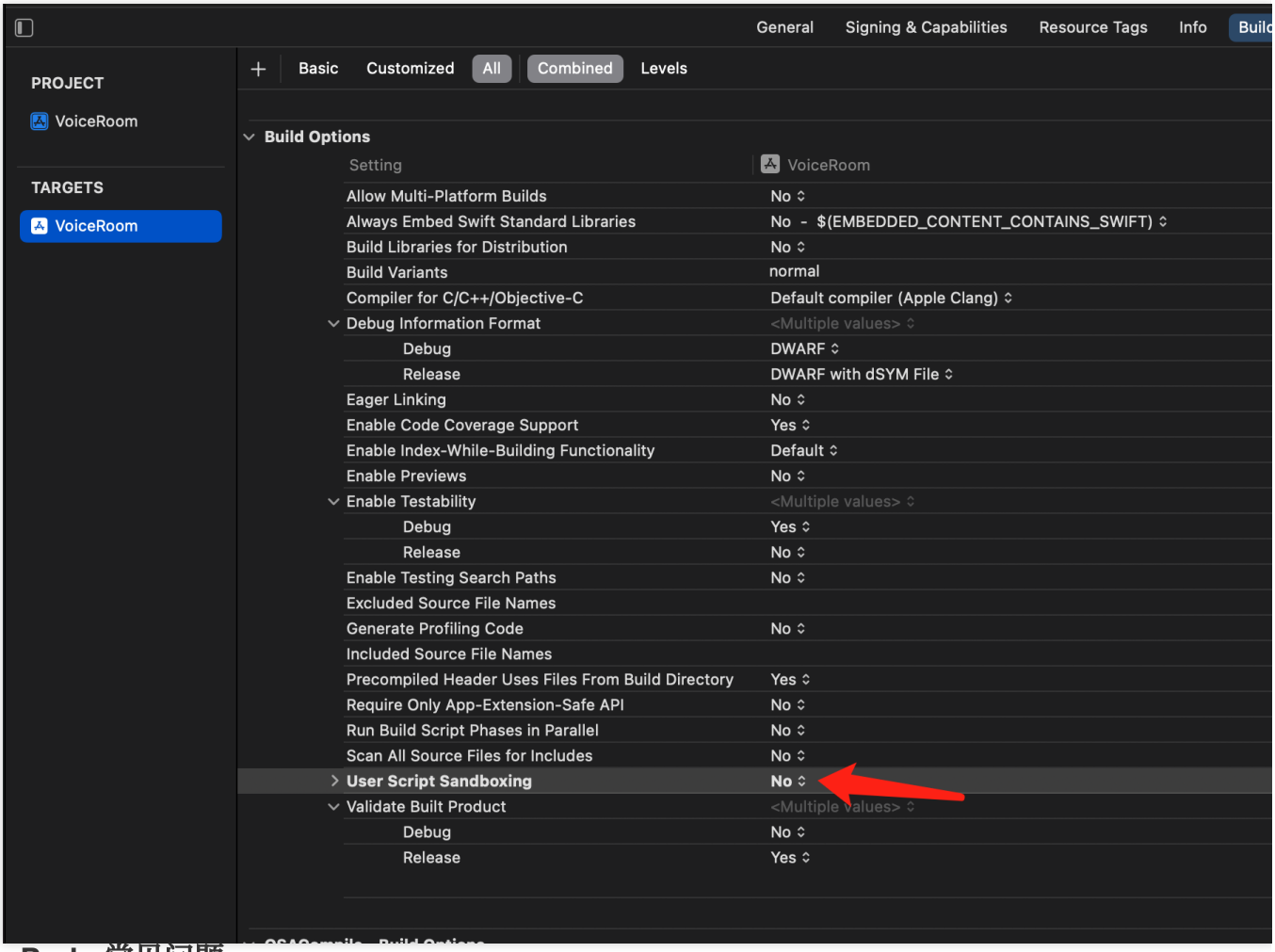
使用苹果芯片（m1\\m2等系列芯片）时会遇到这种弹框。原因是包括 SDWebImage 在内的三方库，并未支持 xcframework。不过苹果依旧给出了适配办法，就是在模拟器上开启 Rosetta 设置， 一般情况下编译时会自动弹出 Rosetta 选项。



Xcode 15 开发者沙盒选项报错 Sandbox: bash(xxx) deny(1) file-write-create



当您使用 Xcode 15 创建一个新工程时， 可能会因为此选项导致编译运行失败，建议您关闭此选项。

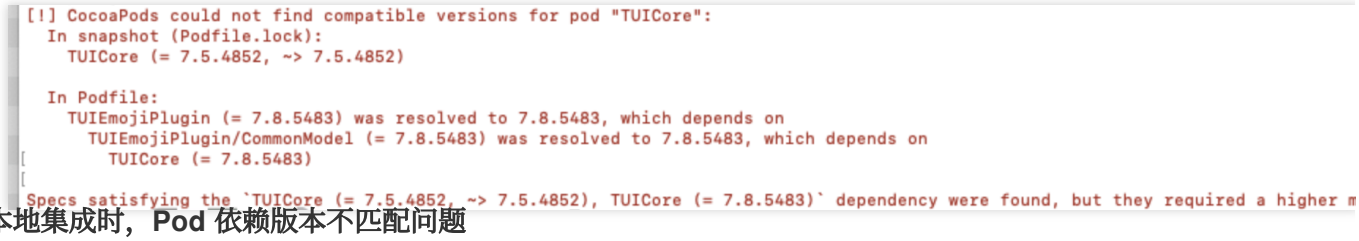


CocoaPods 常见问题

使用远端集成时，Pod 依赖版本不匹配问题

若您使用远端 CocoaPods 集成时，出现 Podfile.lock 和 插件依赖的 TUICore 版本不一致时，

此时请删除 Podfile.lock 文件， 并使用 `pod repo update` 更新本地代码仓库， 之后使用 `pod update` 重新更新即可。



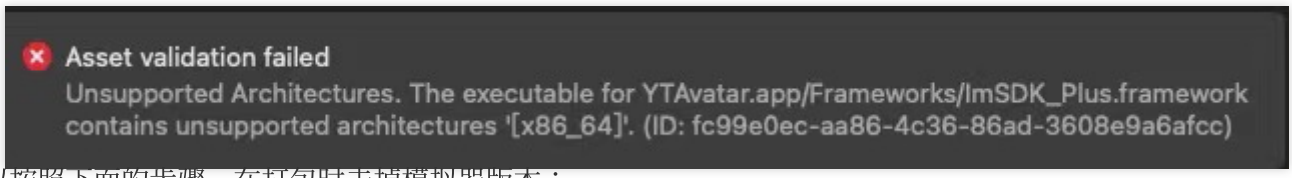
使用本地集成时，Pod 依赖版本不匹配问题

若您使用本地 DevelopmentPods 集成时出现插件依赖的 TUICore 版本较新，但本地 Pod 依赖的版本号是 1.0.0 ，此时请您参考 Podfile_local 和 TUICore.spec 修改。插件需要跟随版本，需要和 TUICore.spec 中一致。
第一次使用本地集成时，建议您下载我们的示例 Demo 工程，将 Podfile 文件内容替换为 Podfile_local 的内容，执行 `Pod update` 后相互参照。

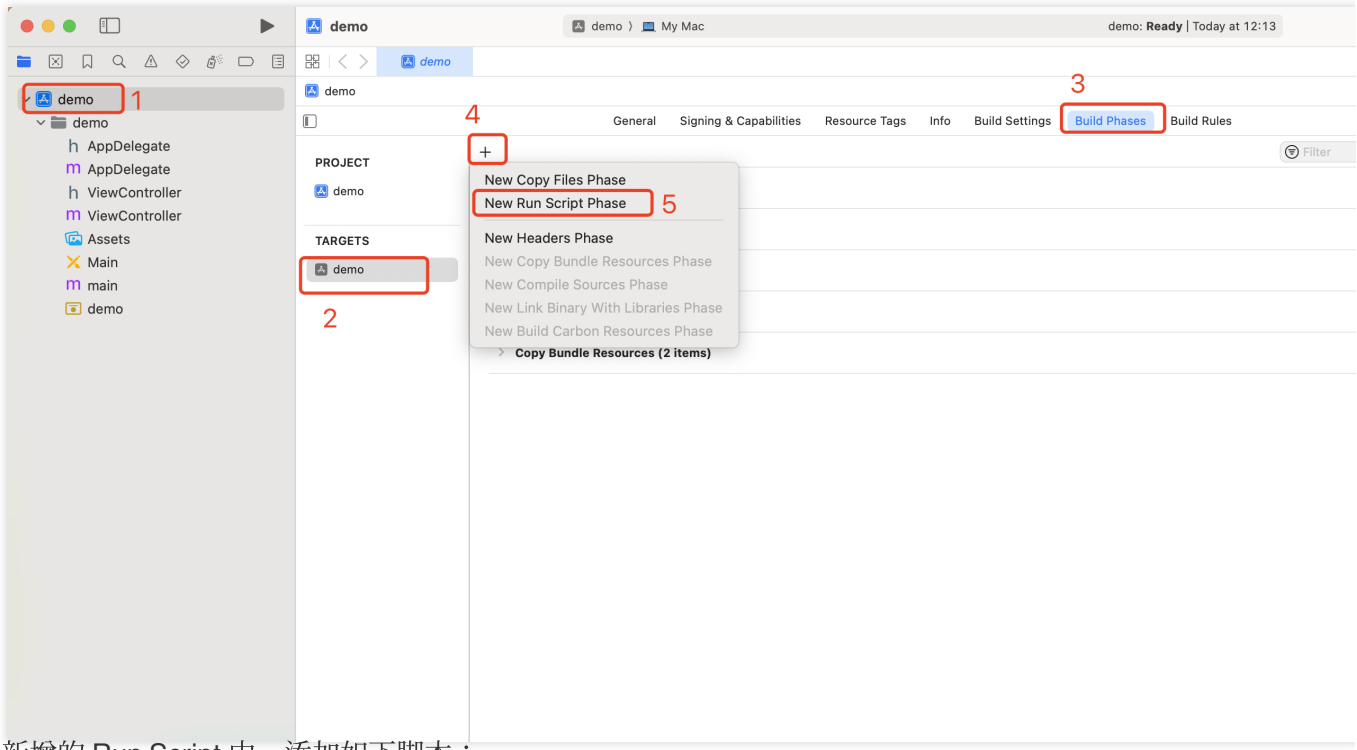
上架常见问题

上架 Appstore 时打包失败，提示 Unsupported Architectures。

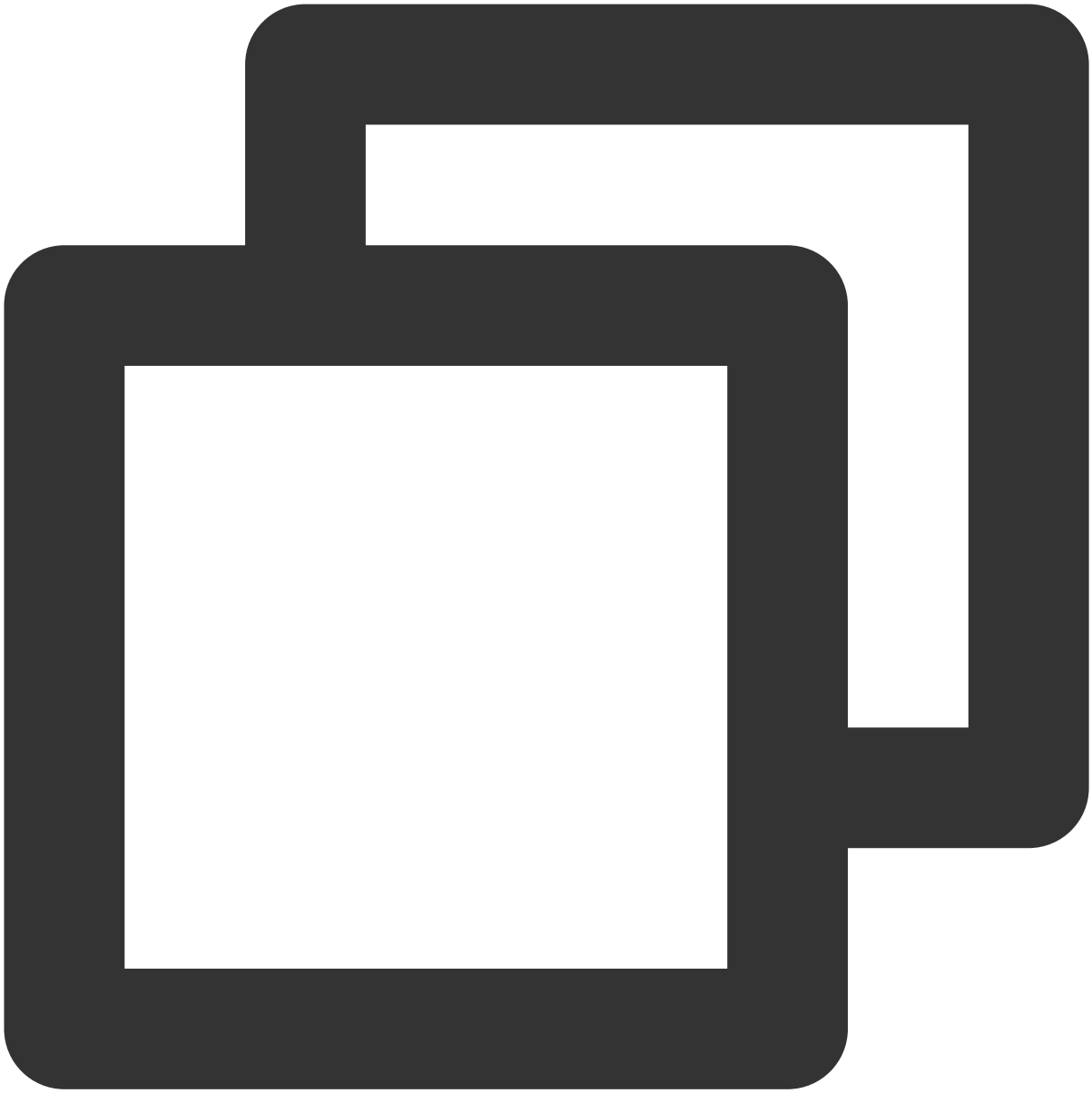
问题现象如下图，打包时提示 ImSDK_Plus.framework 中包含了 Appstore 不支持的 x86_64 模拟器版本。该问题是由于 SDK 为了方便开发者调试，发布时会默认带上模拟器版本。



您可以按照下面的步骤，在打包时去掉模拟器版本：
1.1 选中您工程的 Target，并点击 `Build Phases` 选项，在当前面板中添加 `Run Script` ；



1.2 在新增的 Run Script 中，添加如下脚本：



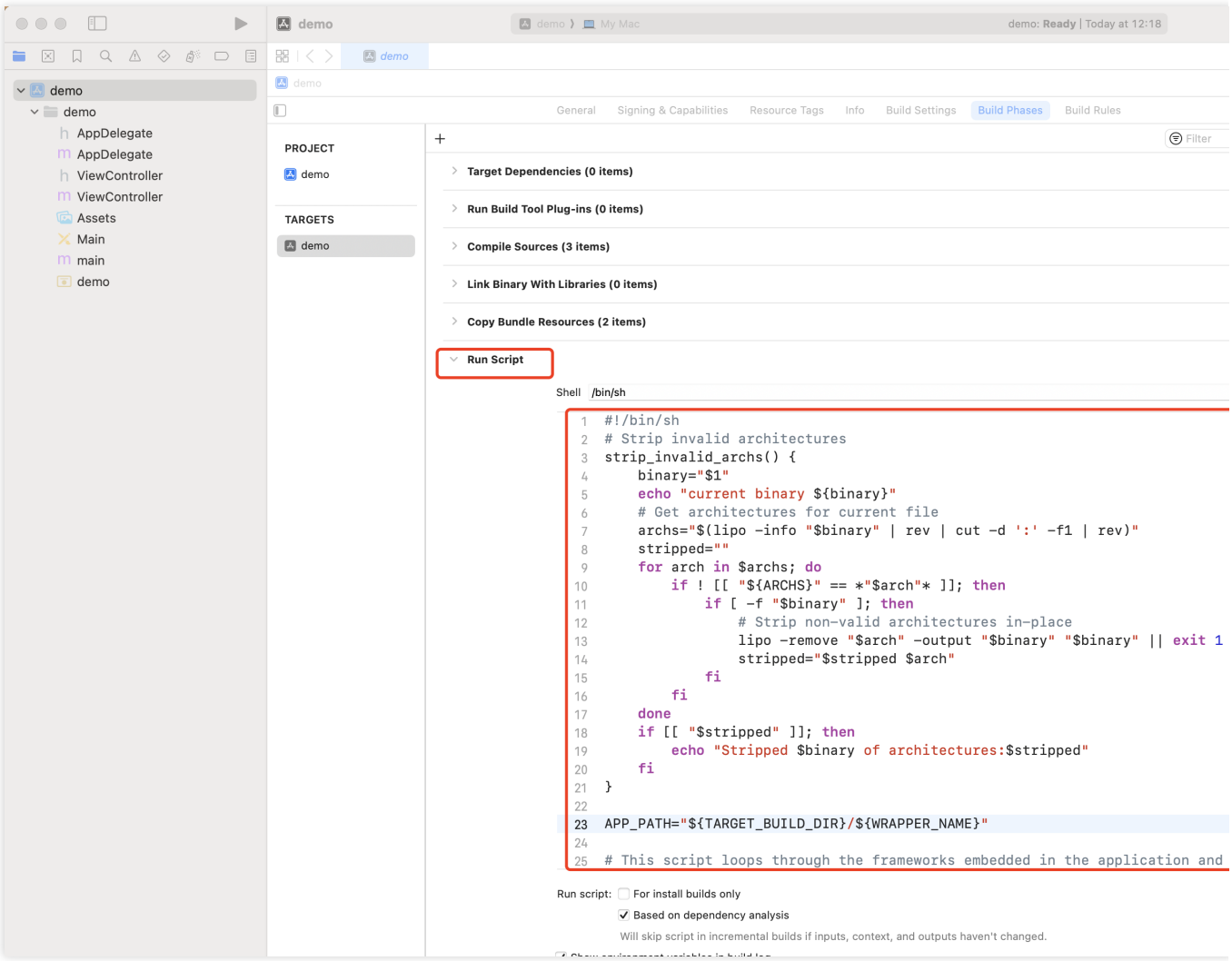
```
#!/bin/sh

# Strip invalid architectures
strip_invalid_archs() {
    binary="$1"
    echo "current binary ${binary}"
    # Get architectures for current file
    archs="$(lipo -info "$binary" | rev | cut -d ':' -f1 | rev)"
    stripped=""
    for arch in $archs; do
        if ! [[ "${ARCHS}" == *"$arch"* ]]; then
```

```
            if [ -f "$binary" ]; then
                # Strip non-valid architectures in-place
                lipo -remove "$arch" -output "$binary" "$binary" || exit 1
                stripped="$stripped $arch"
            fi
        fi
    done
    if [[ "$stripped" ]]; then
        echo "Stripped $binary of architectures:$stripped"
    fi
}

APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
    FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecu
    FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
    echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"
    strip_invalid_archs "$FRAMEWORK_EXECUTABLE_PATH"
done
```



联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Web（Vue）

最近更新时间：2024-08-07 15:09:12

适用场景

Web & H5 平台，独立集成私信聊天（1V1）或者群聊（Group），如房产中介咨询、电商在线客服、保险远程定损等。

开发环境要求

Vue (全面支持 Vue2 & Vue3 ， 请您在下方接入时选择您所匹配的 Vue 版本接入指引进行接入)

TypeScript （ 如您是 js 项目， 请跳转至 js 工程如何接入 TUIKit 组件进行配置 ts 渐进式支持）
sass（sass-loader 版本 ≤ 10.1.1）
node（node.js ≥ 16.0.0）
npm（版本请与 node 版本匹配）

集成指引

请按照 集成 TUIKit 的步骤进行操作，操作完成后，您需要按找以下步骤配置 TUIChat。

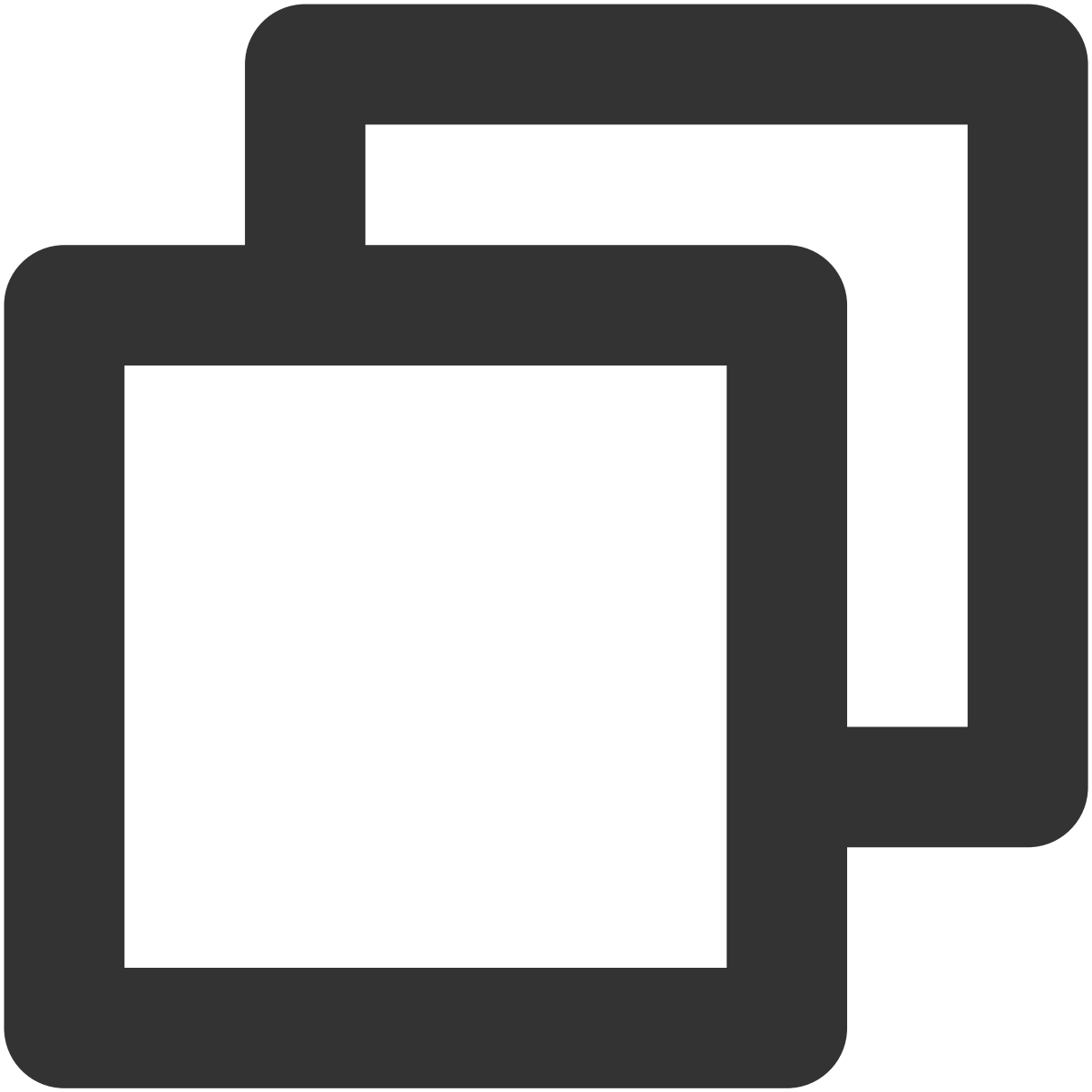
集成 <TUIChat>

在需要展示的页面，引入 TUIChat 的组件即可使用。

例如：在 App.vue 页面中实现以下代码，即可快速搭建聊天界面并开启指定会话：

说明：

conversationID：会话 ID。会话ID组成方式如下：
C2C${userID}（单聊），比如 `C2C123456`
GROUP${groupID}（群聊），比如 `GROUP123456`



```
<template>
  <div id="app">
    <TUIKit
      :SDKAppID="0"
      userID="YOUR_USERID"
      userSig="YOUR_USERSIG"
      conversationID="YOUR_CONVERSATIONID"
      :style="{ width: '500px', height: '800px', margin: '0 auto', boxShadow: '0 11
    >
      <TUIChat><h1>Welcome to Tencent Cloud Chat</h1></TUIChat>
    </TUIKit>
```

```
  </div>
</template>
<script lang="ts" setup>
import { TUIKit, TUIChat } from "./TUIKit";
</script>
<style lang="scss"></style>
```

启动项目

执行以下命令启动项目：

vue-cli

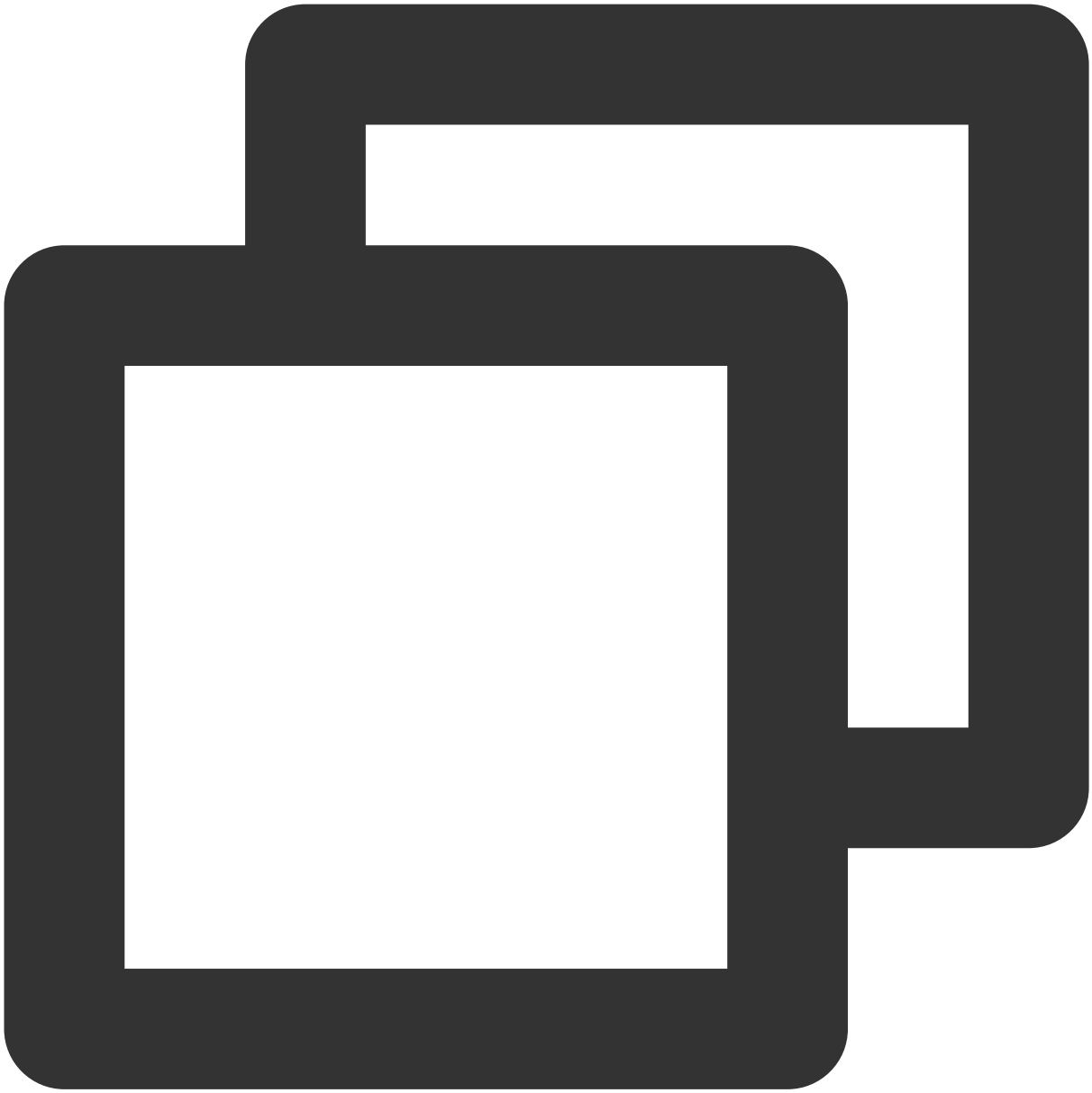
vite

说明：

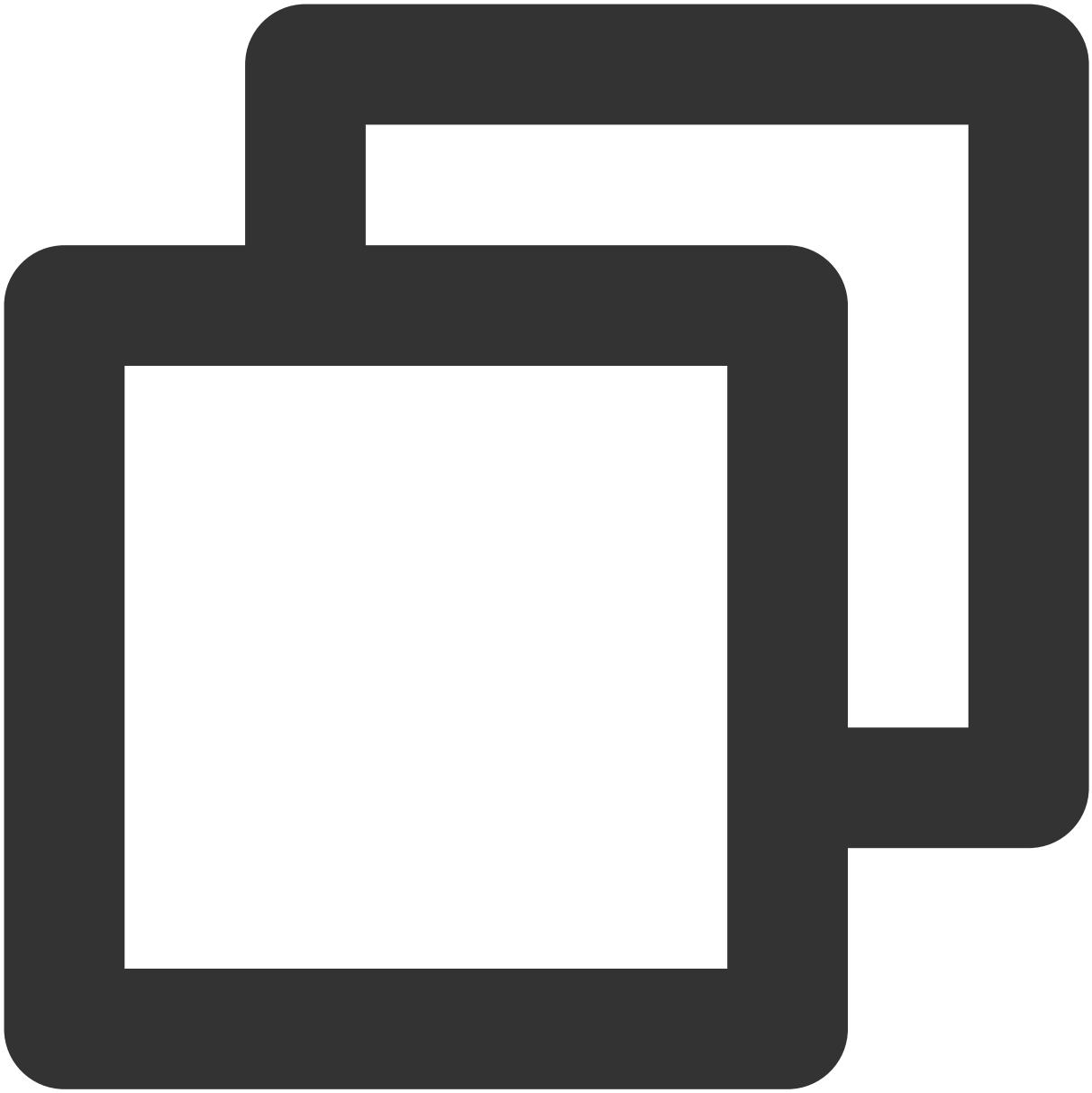
由于 vue-cli 默认开启 webpack 全局 overlay 报错信息提示，为了您有更好的体验，**建议您关闭全局 overlay 报错提示**。

webpack4

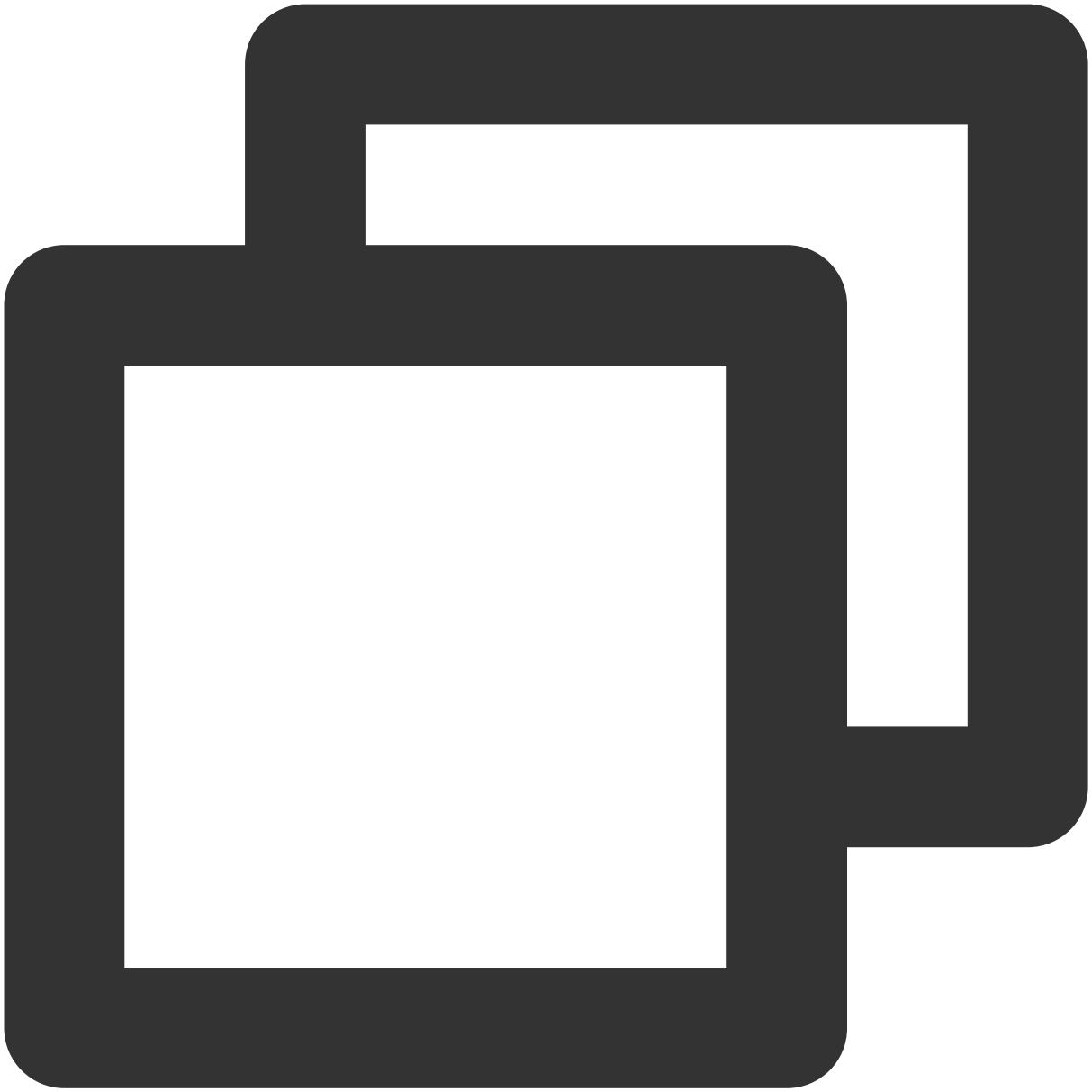
webpack3



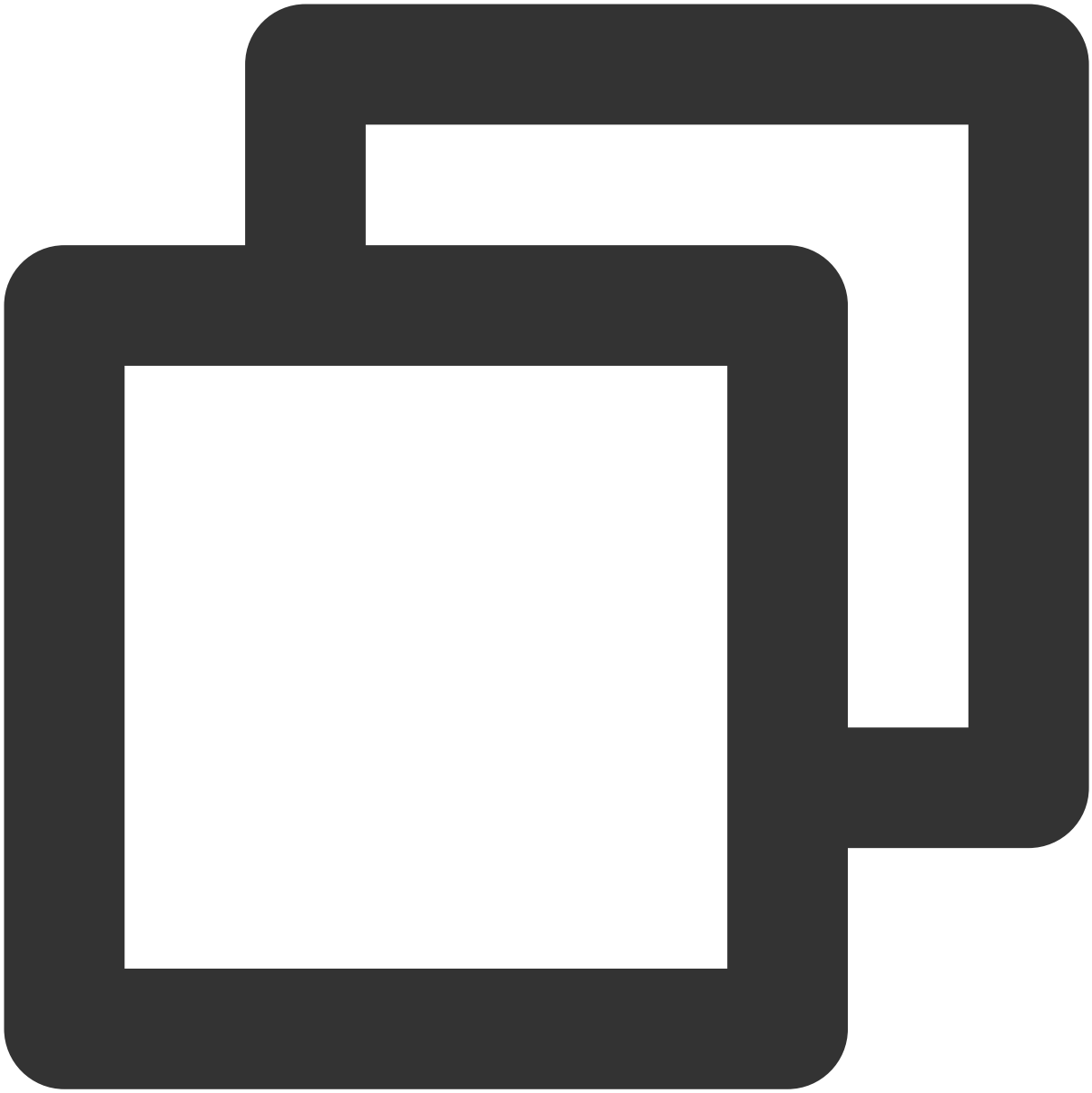
```
module.exports = defineConfig({
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```



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



```
npm run serve
```



```
npm run dev
```

常见问题

请单击 常见问题 查看解决方案。

相关文档

Vue2 & Vue3 UIKit 相关：

chat-uikit-vue npm
Vue2 Demo源码及跑通示例
Vue3 Demo源码及跑通示例

Vue2 & Vue3 UIKit 逻辑层: engine 相关

chat-uikit-engine npm 仓库
chat-uikit-engine 接口文档

交流与反馈

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

React

最近更新时间：2024-08-05 14:47:40

适用场景

Web & H5 平台，独立集成私信聊天（1V1）或者群聊（Group），例如房产中介咨询、电商在线客服、保险远程定损等。
您可以直接体验下面的聊天。同时，您可以通过 **体验沙箱** 快速体验在线代码实现。

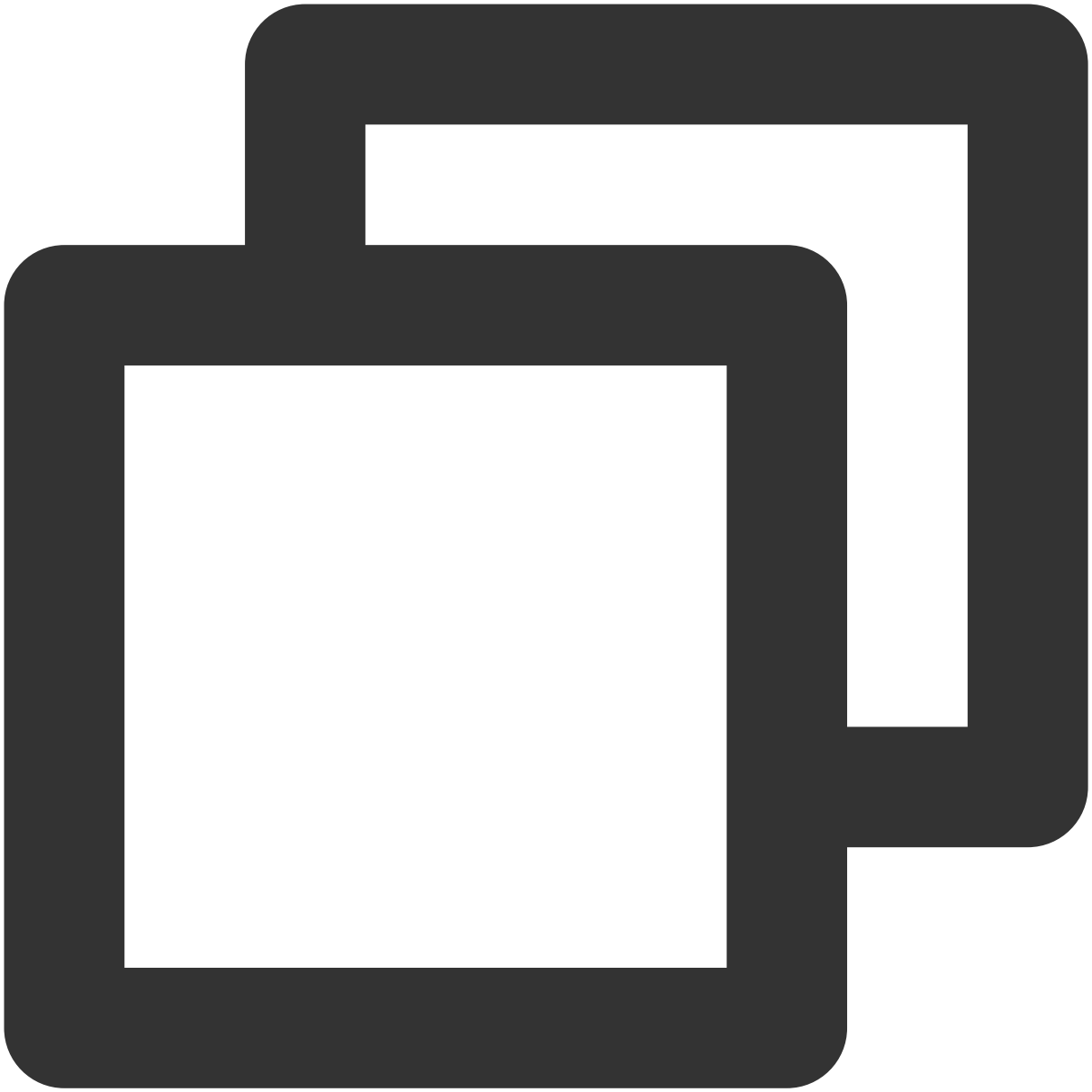
开发环境要求

React version 18+ (Version 17.x is not supported.)
TypeScript
Node.js version 16+
npm（版本请与 node 版本匹配）

chat-uikit-react 集成

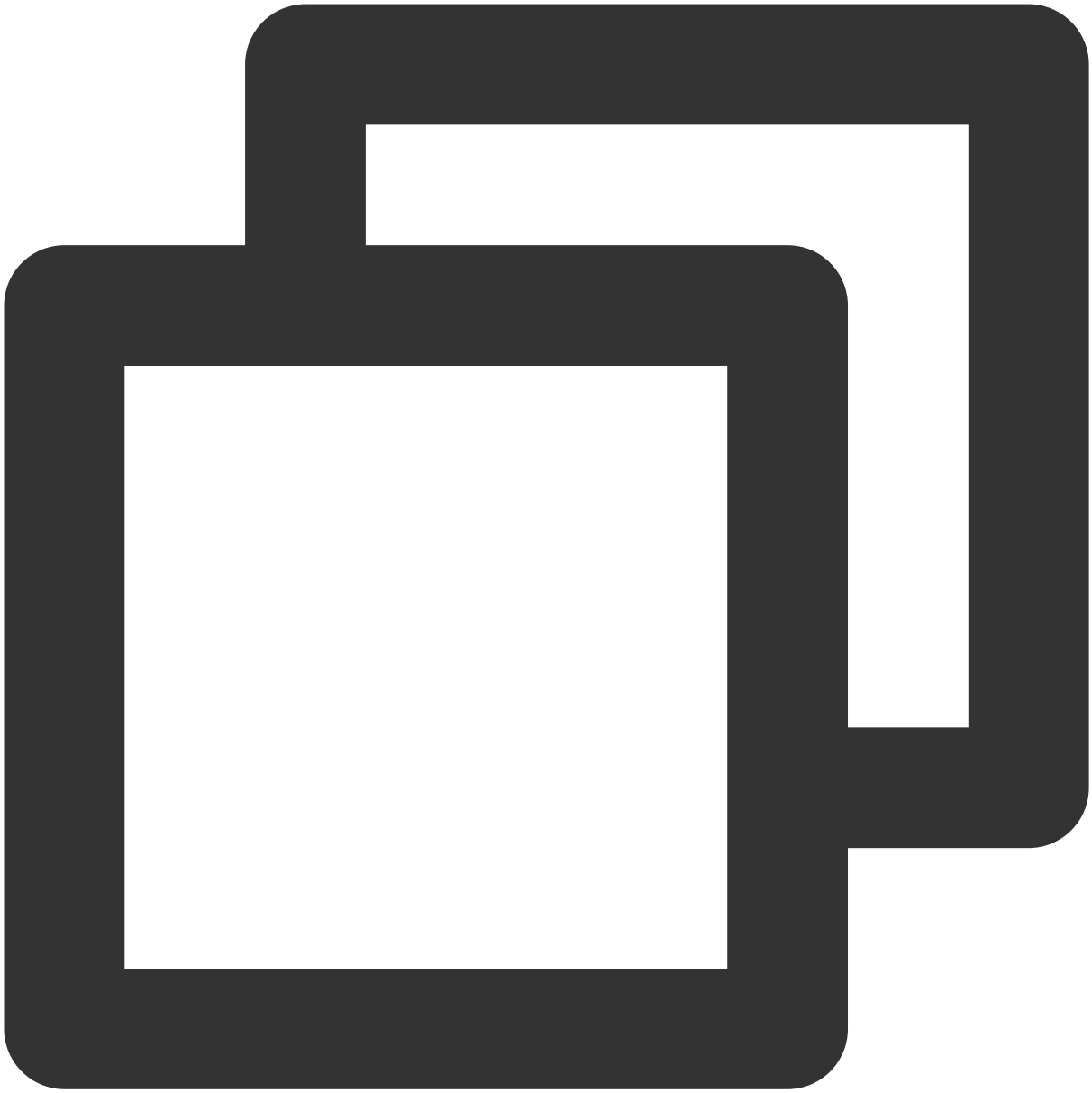
步骤1：创建项目

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



```
npx create-react-app sample-chat --template typescript
```

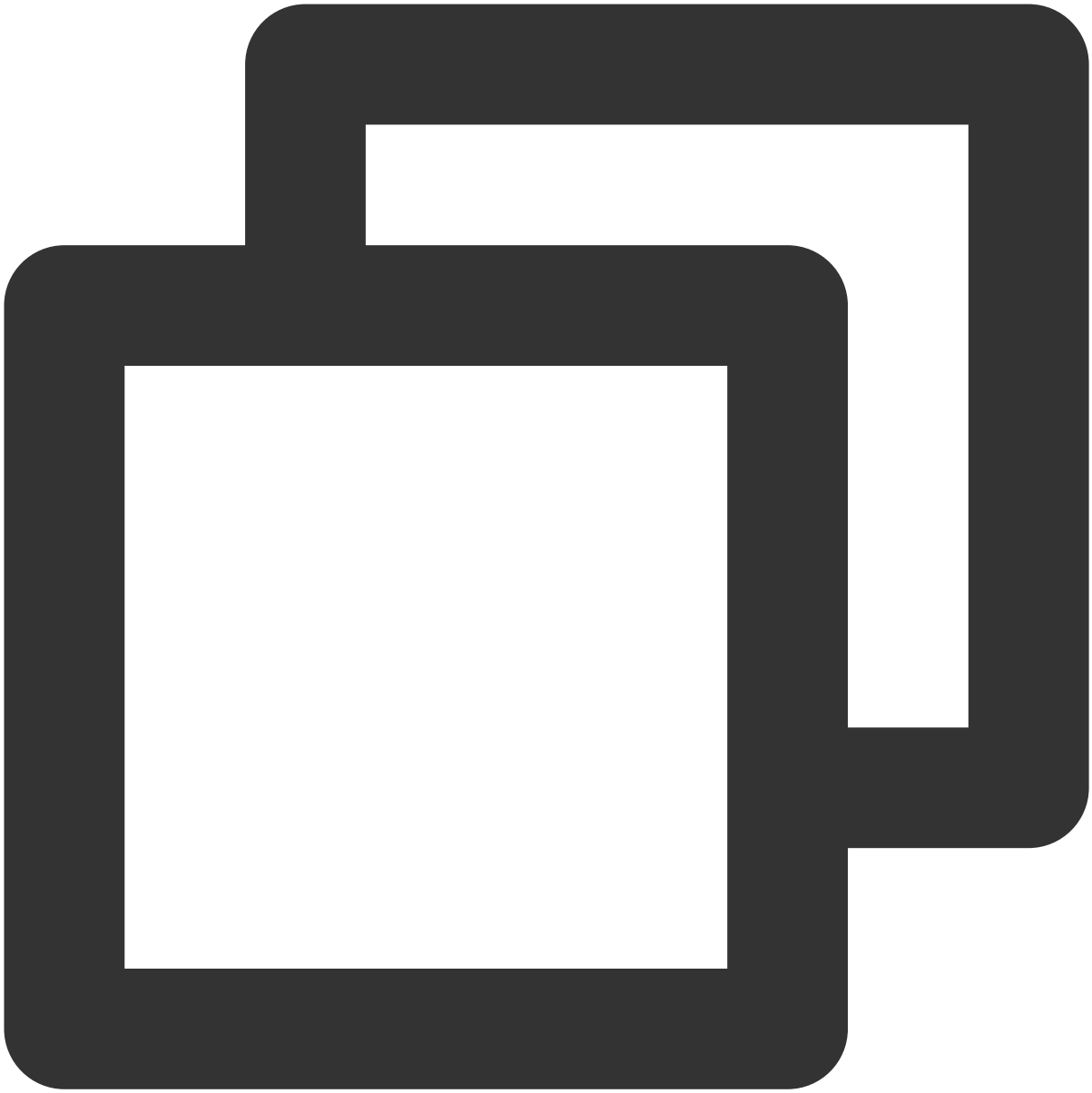
2. 创建项目完成后，切换到项目所在目录。



```
cd sample-chat
```

步骤2：下载 chat-uikit-react 组件

通过 npm 方式下载 chat-uikit-react 并在项目中使用，另外在 GitHub 中也提供相关的 开源代码，您也可在此基础上进行开发您自己的组件库。



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

步骤3：引入 chat-uikit-react 组件，填写 userID / groupID，打开会话进行聊天

注意：

以下代码中未填入 `SDKAppID` 、 `userID` 和 `userSig` ，需在 步骤4 中获取相关信息后进行替换。

说明：

conversationID：会话 ID。会话ID组成方式：

C2C${userID}（单聊）

GROUP${groupID}（群聊）

关于群聊：

通过调用 API createGroup 可获取 groupID

如果是直播群， 需要通过调用 API joinGroup 加入群，才可以进行聊天。

进入聊天

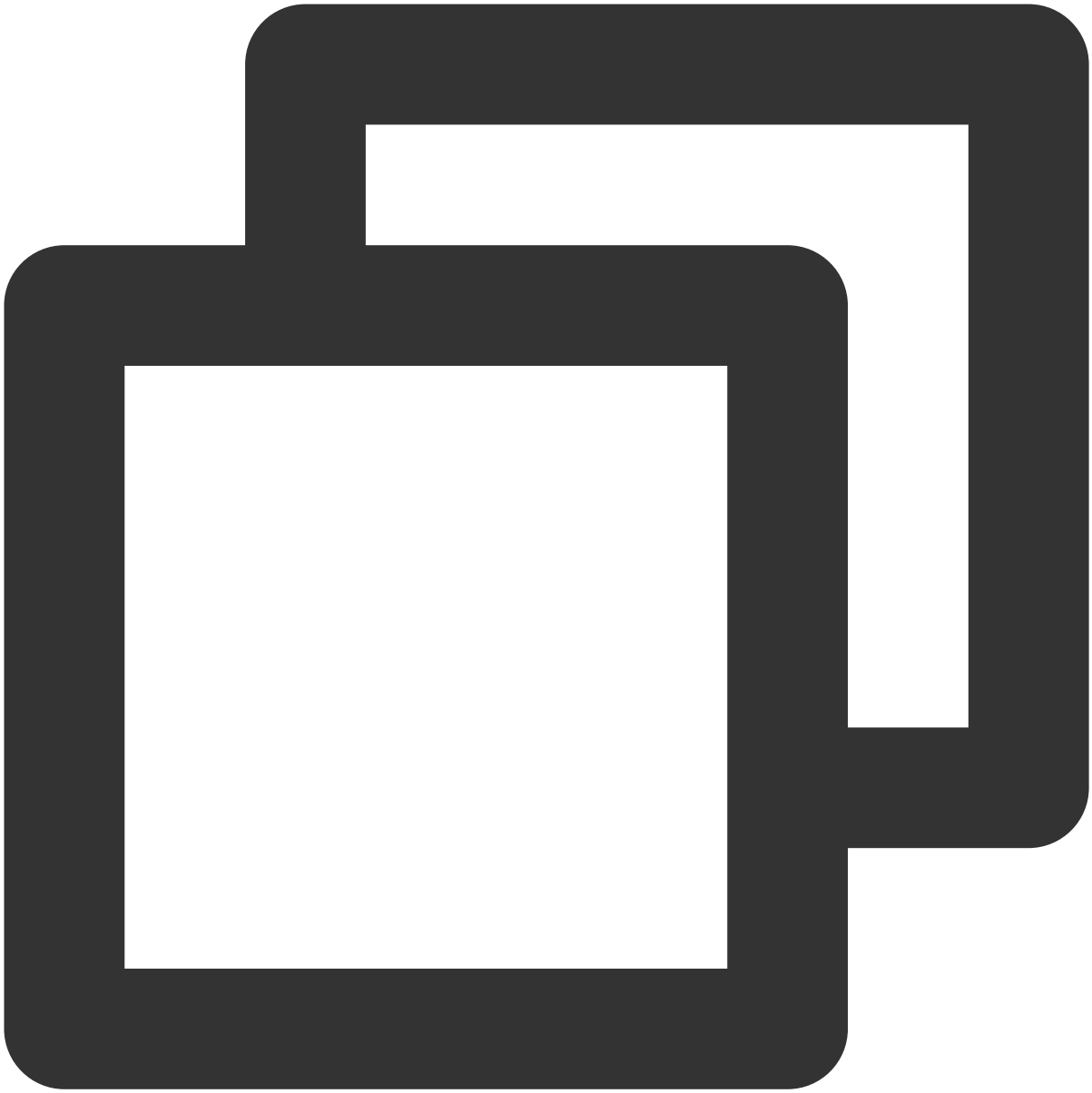
通过调用 API switchConversation，传入 conversationID 进入聊天页面。

npm 集成方式

源码集成方式

替换 App.tsx 中的内容，或者您可以新建一个组件引入。

体验沙箱



```
import React, { useEffect } from 'react';
import { TUIKit, TUIChat } from '@tencentcloud/chat-uikit-react';
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from "@tencentcloud/chat-uikit-engine";
import '@tencentcloud/chat-uikit-react/dist/cjs/index.css';

const config = {
  SDKAppID: 0, // Your SDKAppID, Get it from Step 4
  userID: 'test-user1', // Login UserID
  userSig: '', // Your userSig
}
```

```
export default function SampleChat() {
  const init = () => {
    TUILogin.login({
    ...config,
    useUploadPlugin: true
    }).then(() => {
      openChat();
    }).catch(() => {});
  }

  const openChat = () => {
    // 1v1 chat: conversationID = `C2C${userID}`
    // group chat: conversationID = `GROUP${groupID}`
    const userID = 'test-user2'; // userID: Recipient of the Message userID, Get it
    const conversationID = `C2C${userID}`;
    TUIConversationService.switchConversation(conversationID);
  };

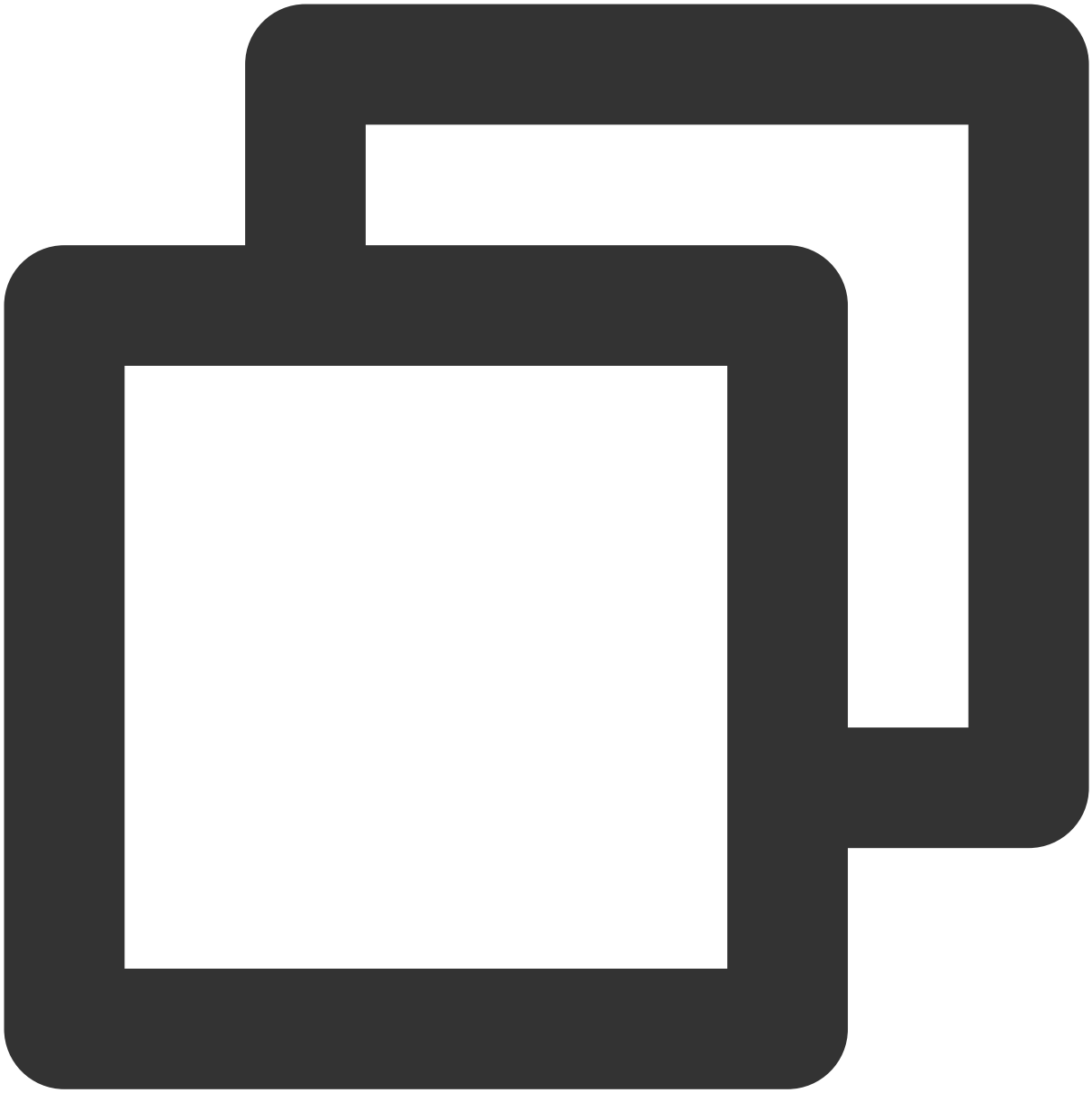
  useEffect(() => {
    init();
  }, []);

  // language support en or zh
  return (
    <TUIKit language={'en-US'}>
      <TUIChat/>
    </TUIKit>
  )
}
```

1. 将 TUIKit 拷贝到自己项目的 src 文件目录下：

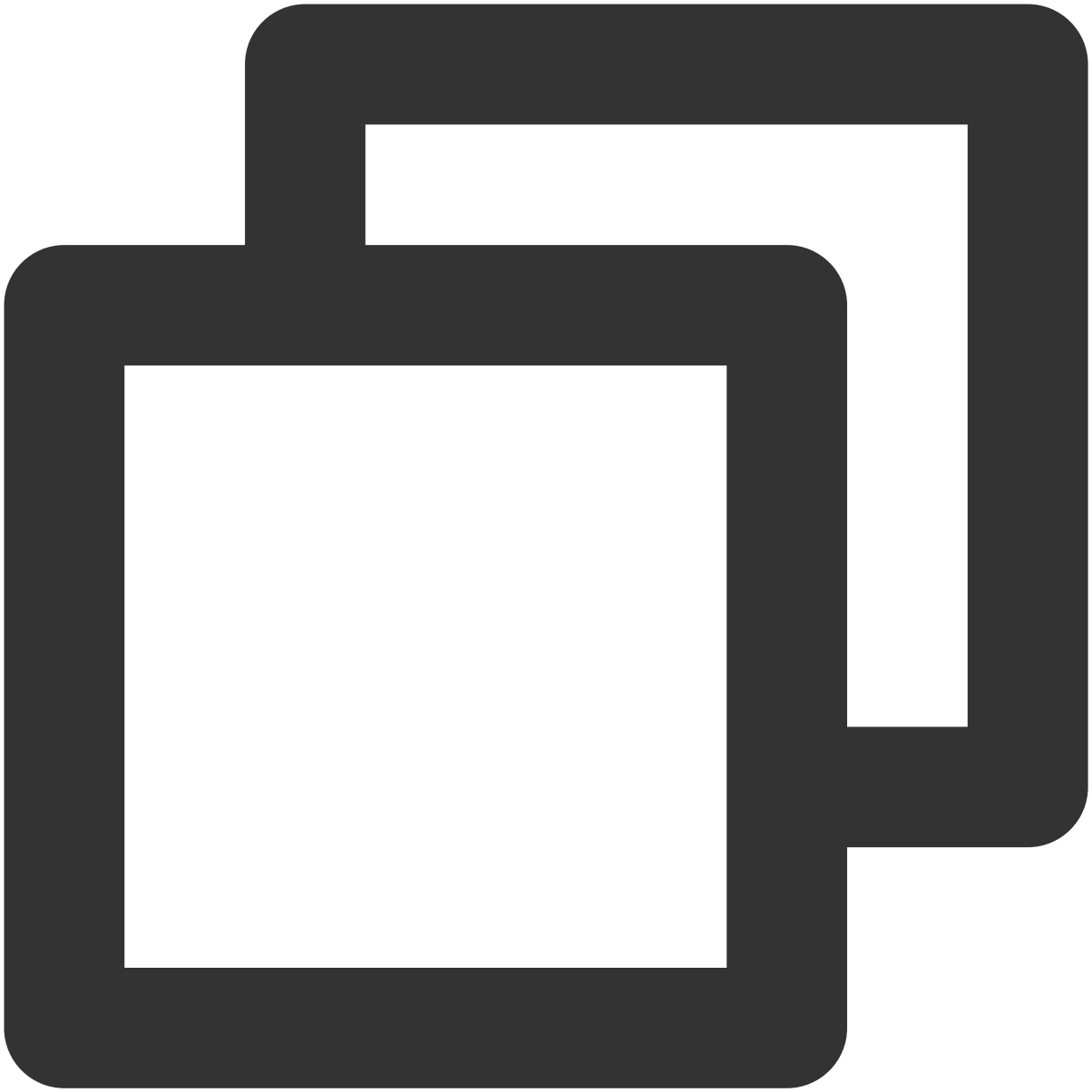
macOS 端

Windows 端



```
mkdir -p ./src/TUIKit && rsync -av  ./node_modules/@tencentcloud/chat-uikit-react/
```

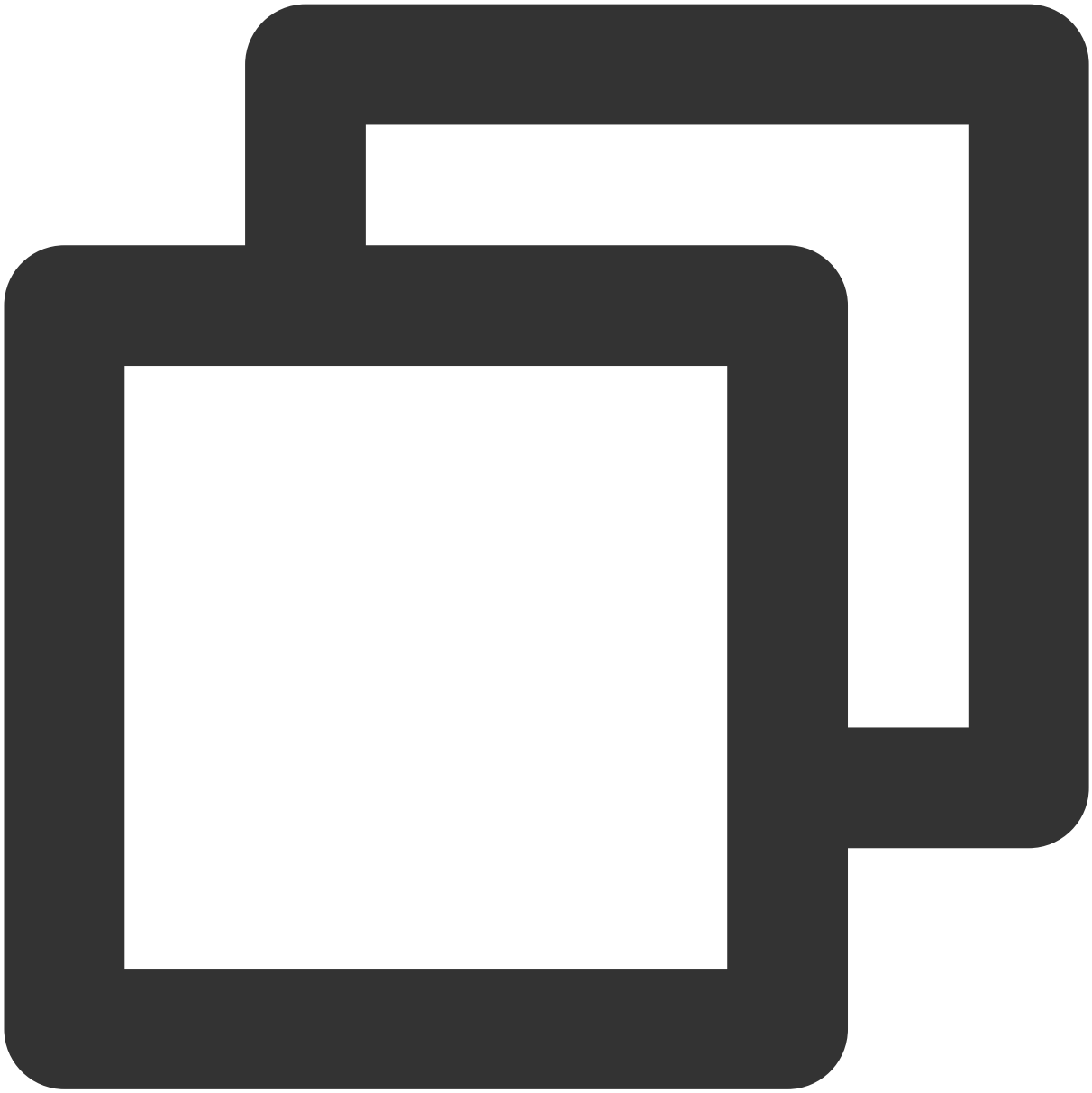
版权所有：腾讯云计算（北京）有限责任公司



```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-react .\\src\\TUIKit /i /e
```

2. 替换 App.tsx 中的内容，或者您可以新建一个组件引入。

体验沙箱



```
import React, { useEffect, useState } from 'react';
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIKit, TUIChat } from './TUIKit/index';
import { TUIConversationService } from "@tencentcloud/chat-uikit-engine";

const config = {
  SDKAppID: 0, // Your SDKAppID
  userID: 'test-user1', // Login UserID
  userSig: '', // Your userSig
}
export default function SampleChat() {
```

```
  const init = () => {
    TUILogin.login({
    ...config,
    useUploadPlugin: true
    }).then(() => {
      openChat();
    }).catch(() => {});
  }

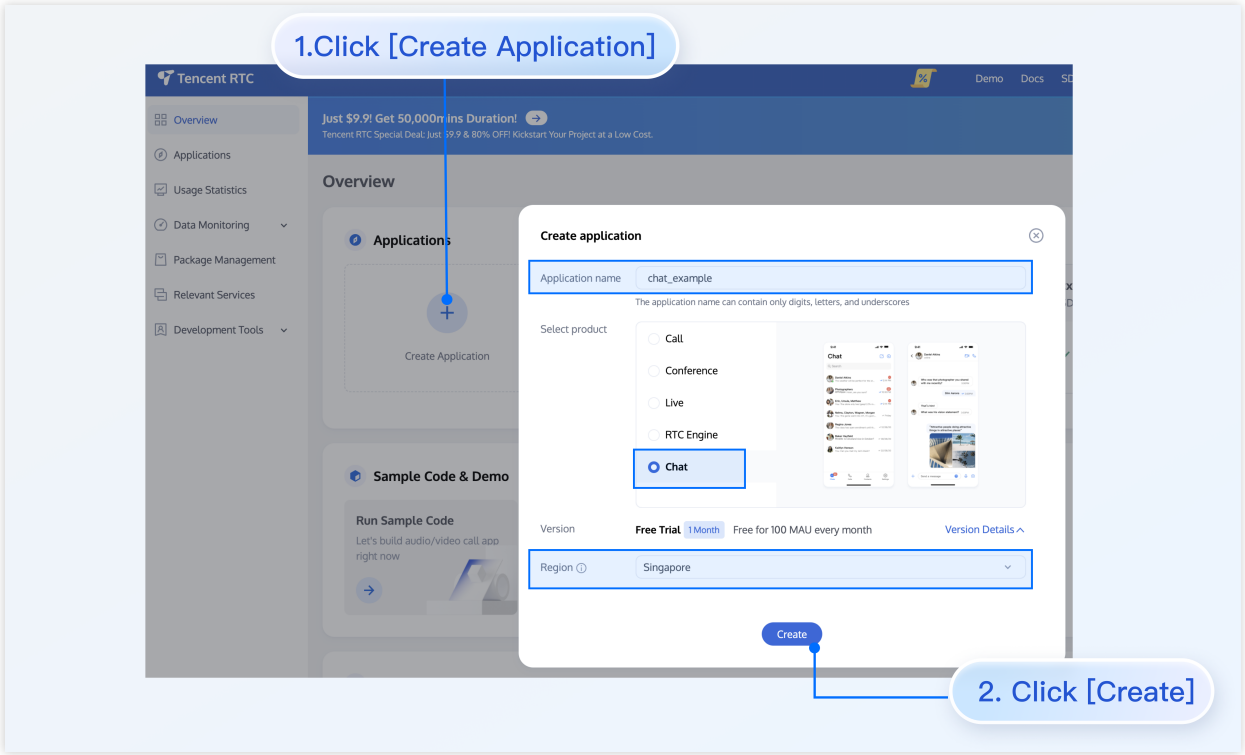
  const openChat = () => {
    // 1v1 chat: conversationID = `C2C${userID}`
    // group chat: conversationID = `GROUP${groupID}`
    const userID = 'test-user2'; // userID: Recipient of the Message userID, Get it
    const conversationID = `C2C${userID}`;
    TUIConversationService.switchConversation(conversationID);
  };

  useEffect(() => {
    init();
  }, []);

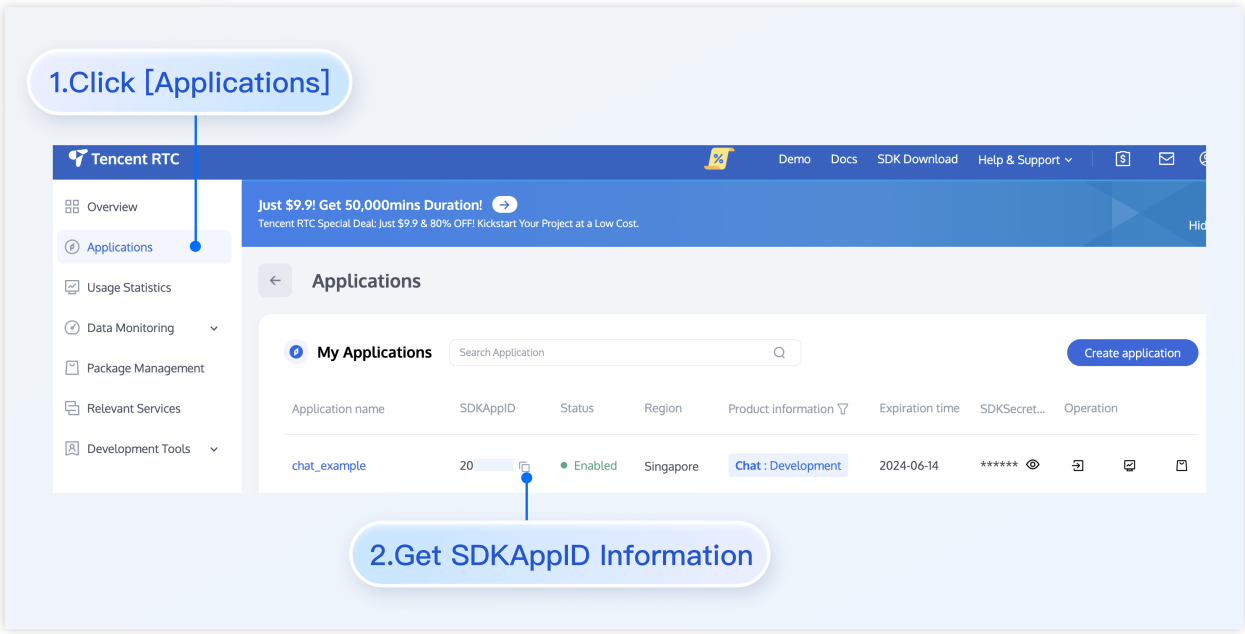
 // language support en or zh
  return (
    <TUIKit language={'en-US'}>
      <TUIChat/>
    </TUIKit>
  )
}
```

步骤4：创建一个应用

1. 登录 Chat Console 。
2. 单击“Create Application”，输入您的应用程序名称，然后单击“Create”。



3. 创建完成后，您可以在控制台概览页面看到新应用的状态、服务版本、SDKAppID、创建时间、标签、过期时间。



步骤5：获取 userID 和 userSig

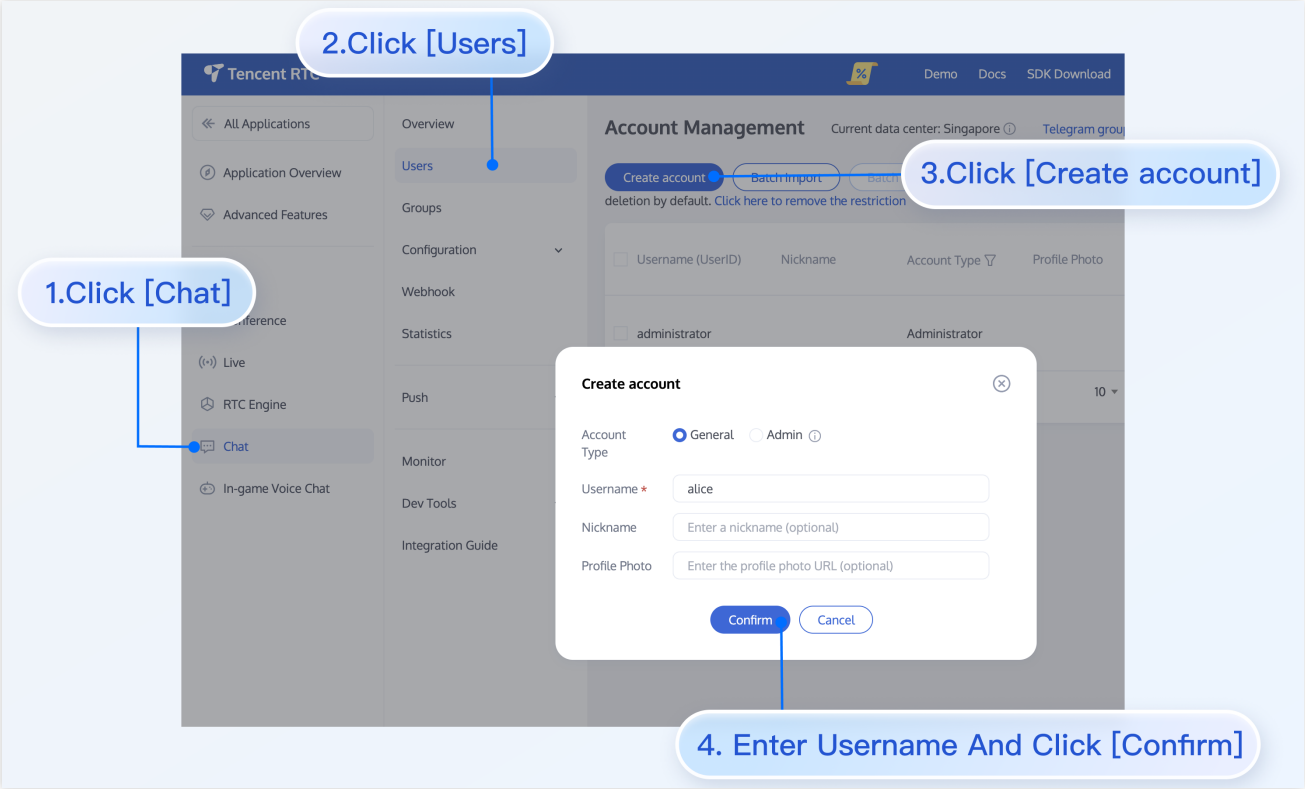
`userID`

单击进入您上面创建的 Application，会在左侧边栏看到 `Chat` 产品入口，单击进入。

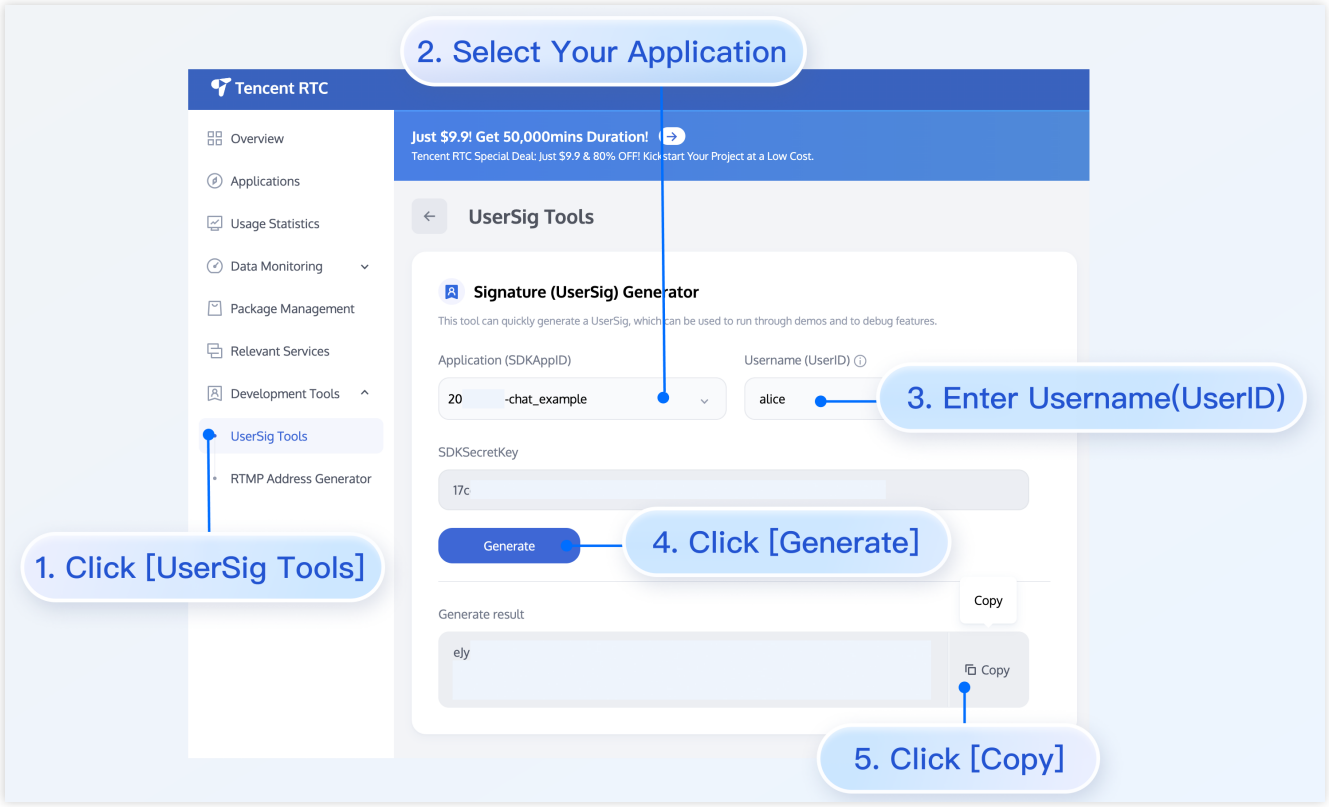
进入 Chat 产品子页面后，点击 `Users`，进入用户管理页面。

版权所有：腾讯云计算（北京）有限责任公司

单击 `Create account`，弹出创建账号信息填写框。如果只是普通成员，我们建议您选择 `General` 类型。为了您更好的体验消息收发等功能，建议您创建两个 userID。

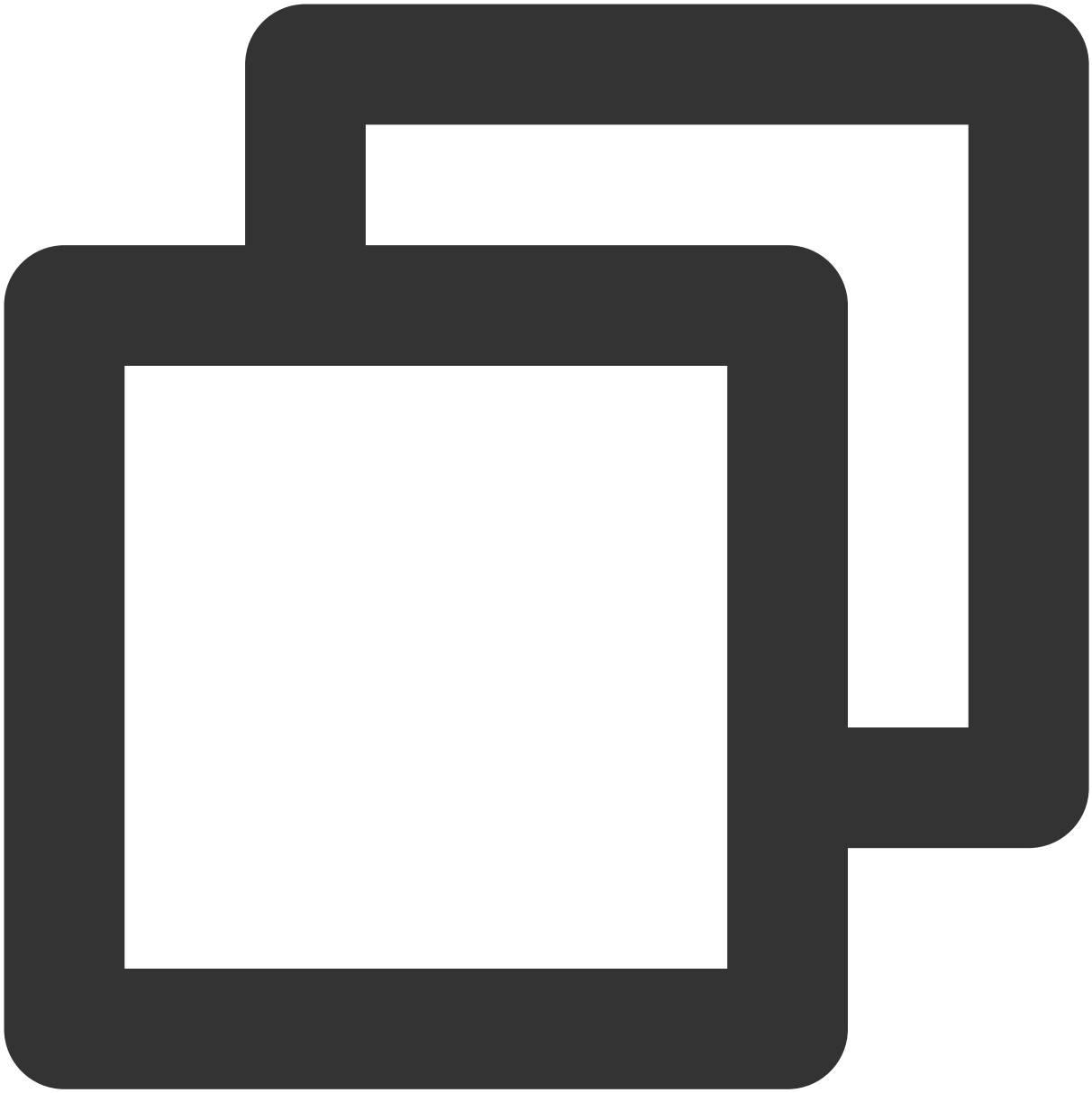


`userSig`，可使用控制台提供的开发工具实时生成，开发工具请点击 Chat Console > Development Tools > UserSig Tools > Signature (UserSig) Generator 。



步骤6：启动项目

替换 App.tsx 中的 SDKAppID、userID、userSig，然后运行命令如下：



```
npm run start
```

注意：

1. 请确保 步骤3 代码中 `SDKAppID` 、 `userID` 和 `userSig` 均已成功替换，如未替换将会导致项目表现异常。
2. `userID` 和 `userSig` 为一一对应关系，具体参见 生成 UserSig。
3. 如遇到项目启动失败，请检查 开发环境要求 是否满足。

步骤7：发送您的第一条消息

在输入框输入消息，按下 **enter** 键发送。

常见问题

什么是 UserSig？

UserSig 是用户登录 Chat 的密码，其本质是对 UserID 等信息加密后得到的密文。

如何生成 UserSig？

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

注意：

本文示例代码采用的获取 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**。 正确的 UserSig 签发方式请参见上文。

是否支持 react 17.x 版本？

目前不支持 17.x 版本，仅支持 React ≥ v18.0。

参考文档

UIKit 相关：

chat-uikit-react npm
Demo源码及跑通示例

实现更多功能，请参考 ChatEngine API 文档：

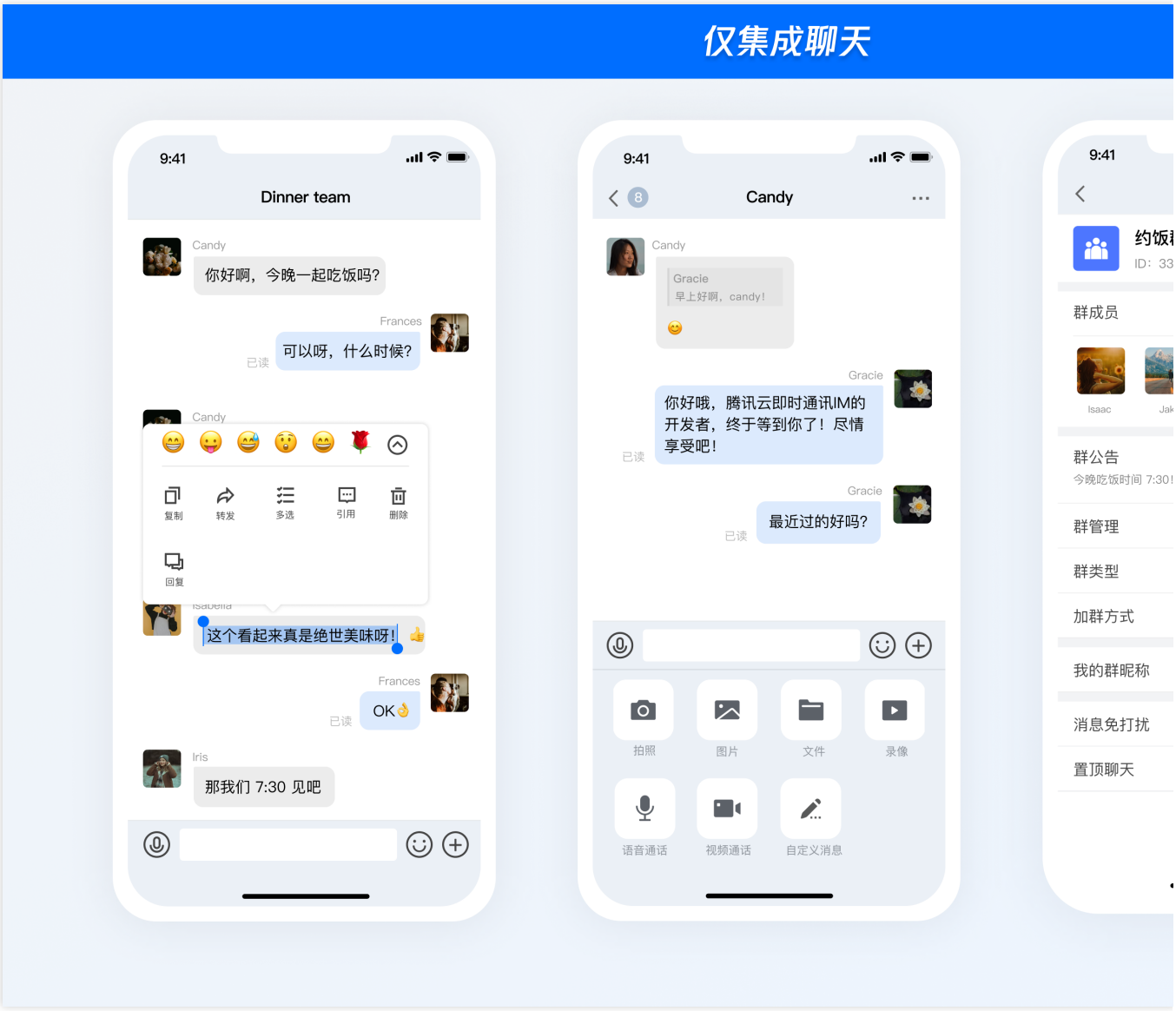
chat-uikit-engine API 手册
chat-uikit-engine npm

uni-app

最近更新时间：2024-06-03 16:50:17

适用场景

uniapp 平台，独立集成私信聊天（1V1）或者群聊（Group），如房产中介咨询、电商在线客服、保险远程定损等。



开发环境要求

HBuilderX （HBuilderX 版本 >= 3.8.4.20230531）或者升级到最新版本

Vue2 / Vue3

sass（sass-loader 版本 ≤ 10.1.1）

版权所有：腾讯云计算（北京）有限责任公司

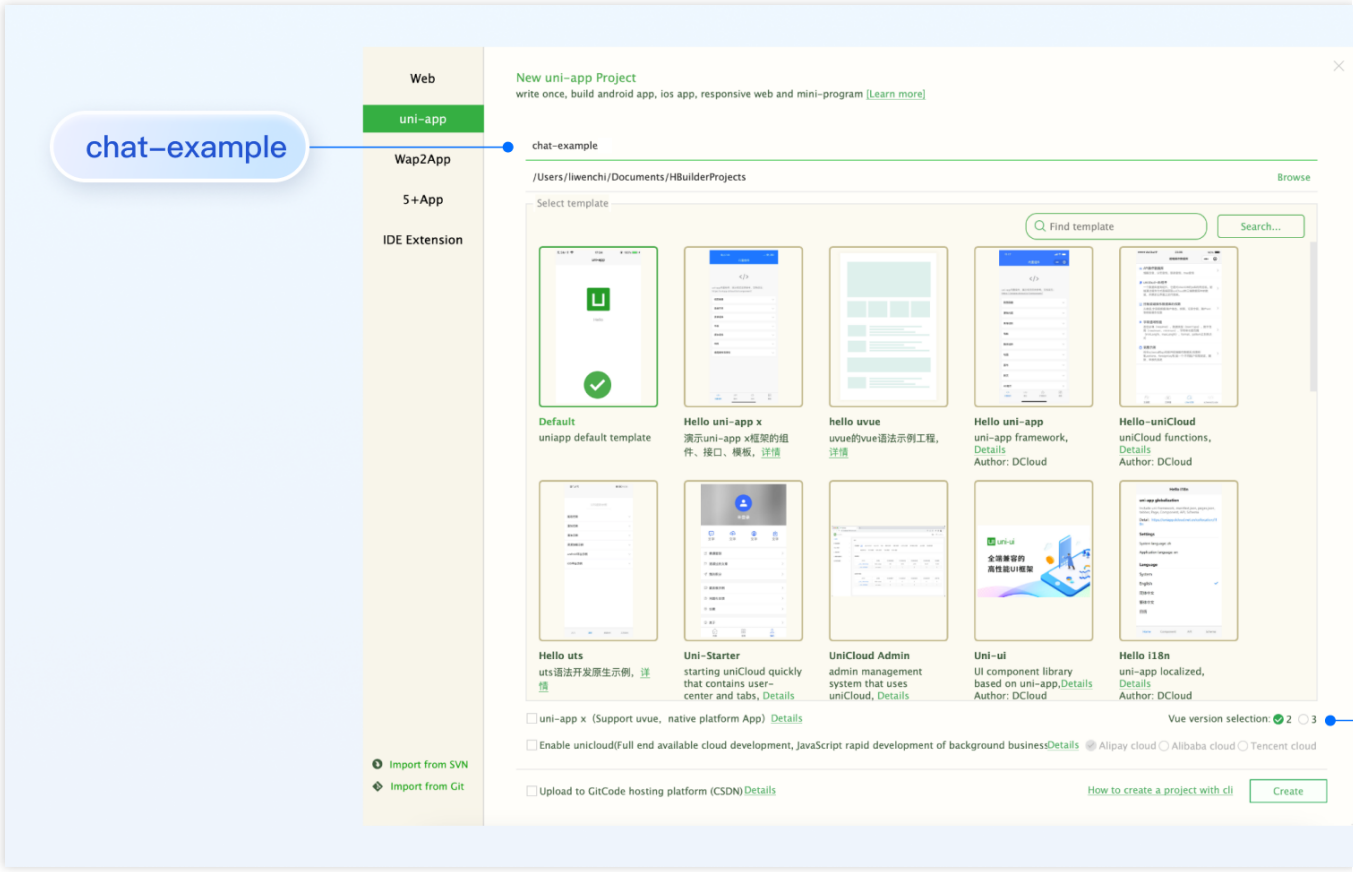
node（12.13.0 ≤ node 版本 ≤ 17.0.0, 推荐使用 Node.js 官方 LTS 版本 16.17.0）
npm（版本请与 node 版本匹配）

集成 TUIChat

通过以下步骤发送您的第一条消息。

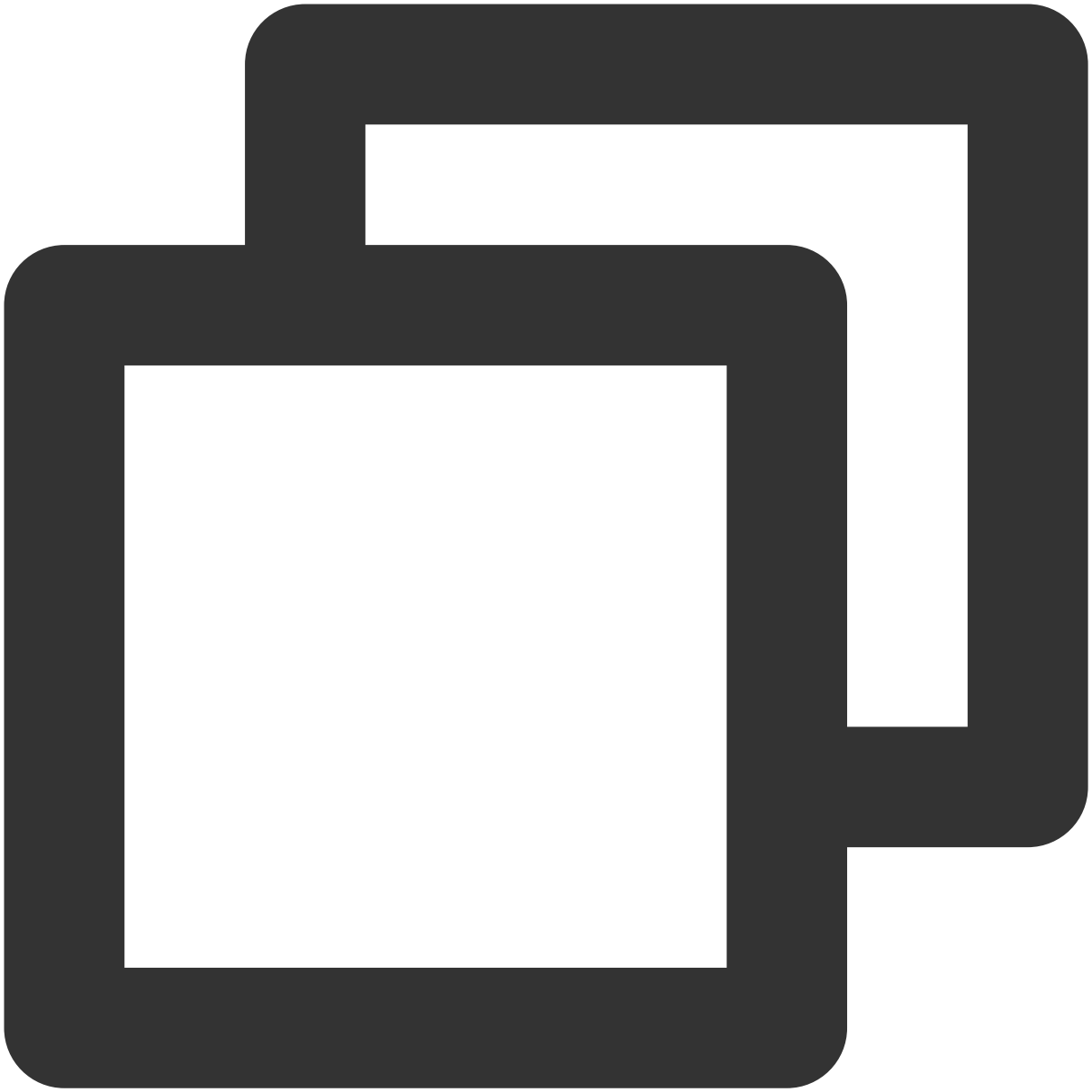
步骤1：创建项目 （已有项目可忽略）

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



步骤2：下载 TUIKit

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



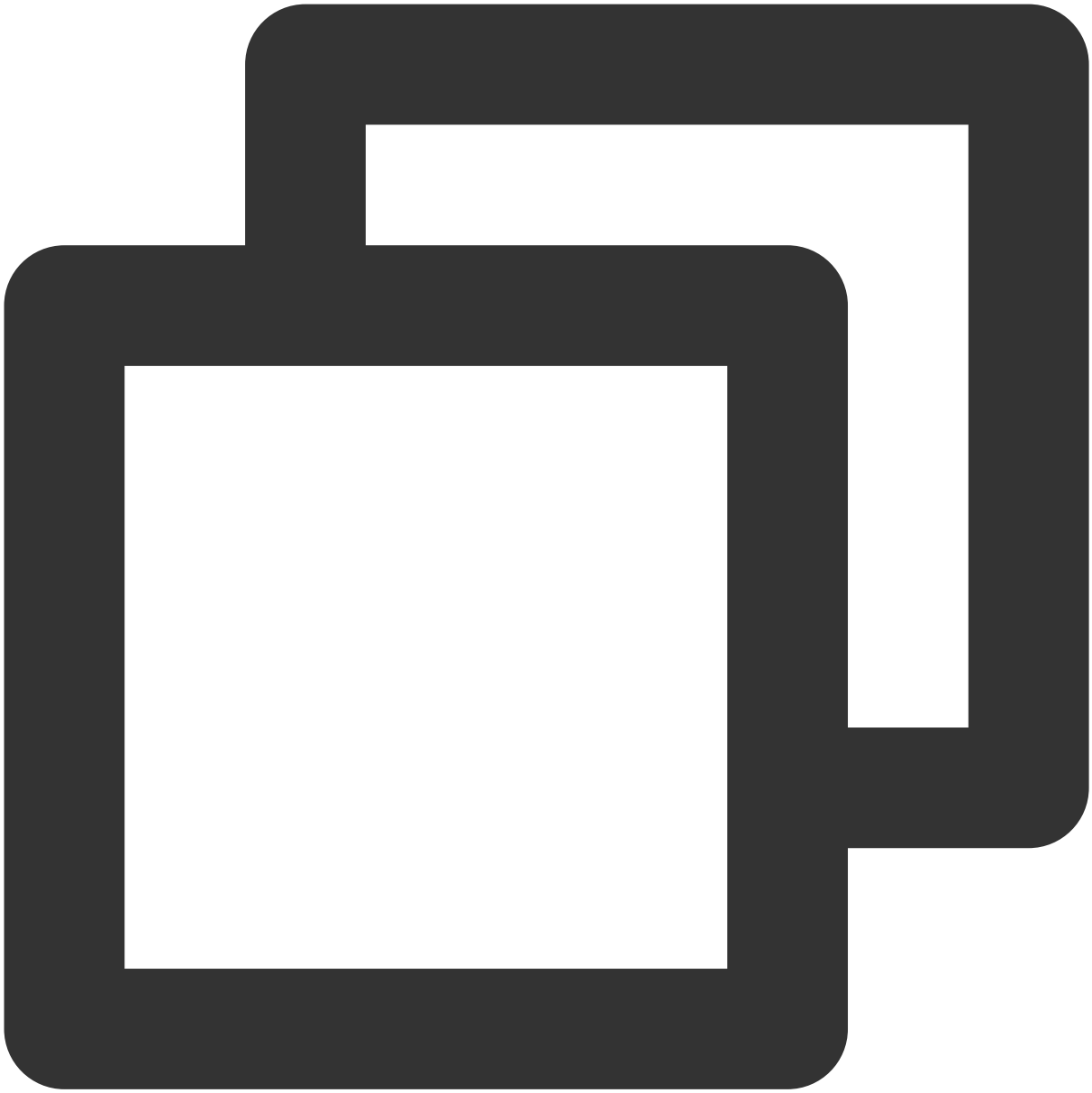
```
npm init -y
```

下载 TUIKit 并拷贝至源码中：

macOS 端

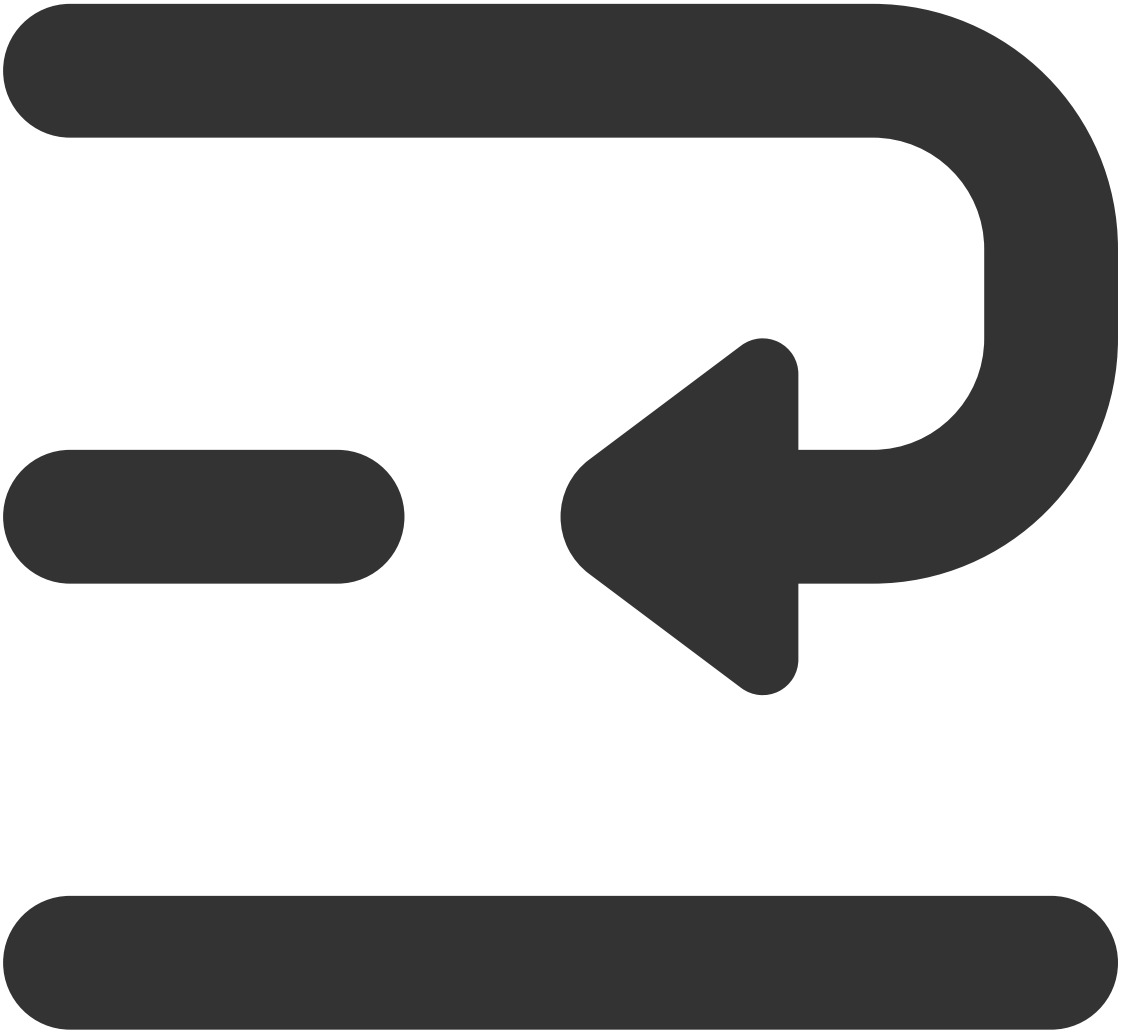
Windows 端

通过 npm 方式下载 TUIKit 组件：

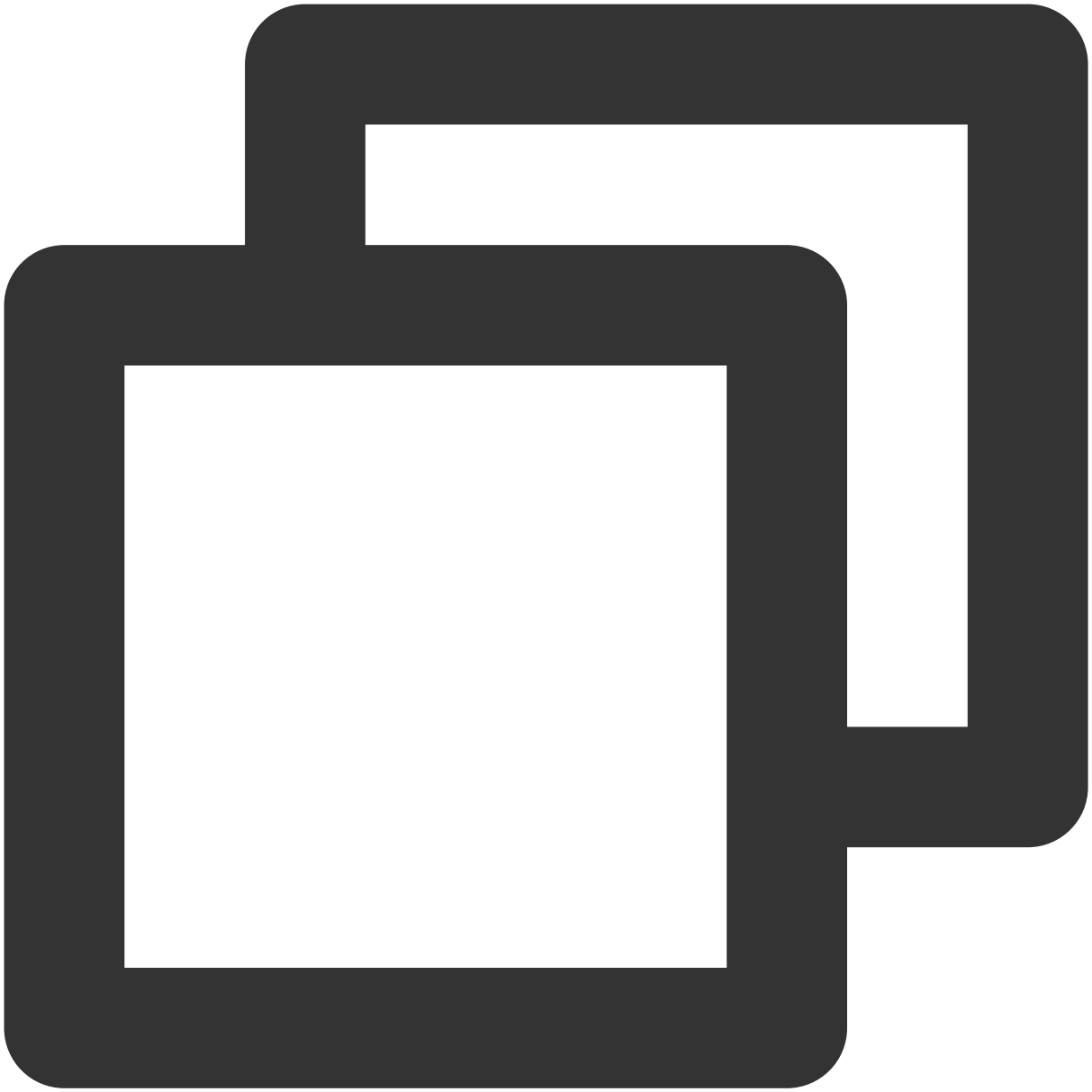


```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

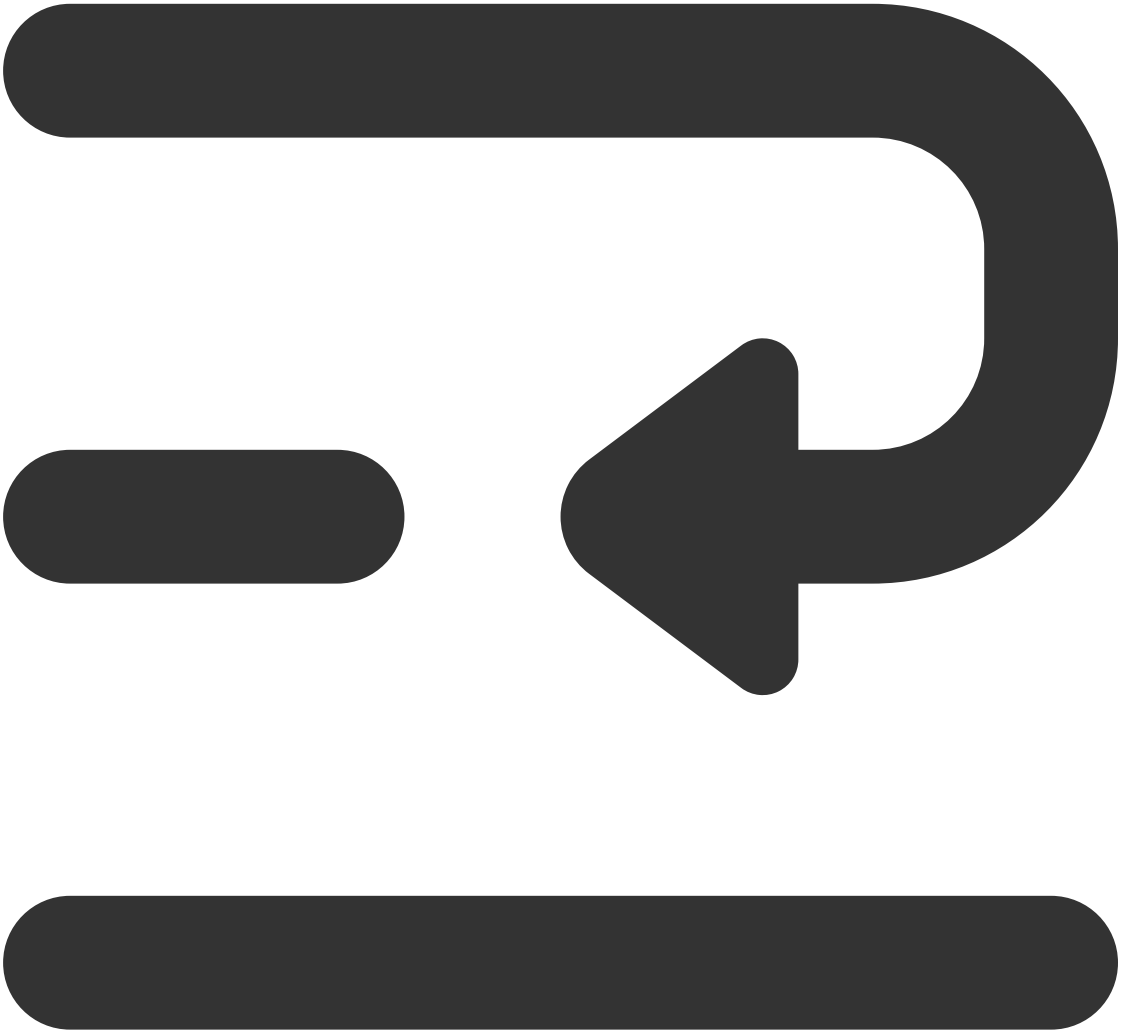
为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 pages 目录下，请在自己的项目根目录下执行以下命令：



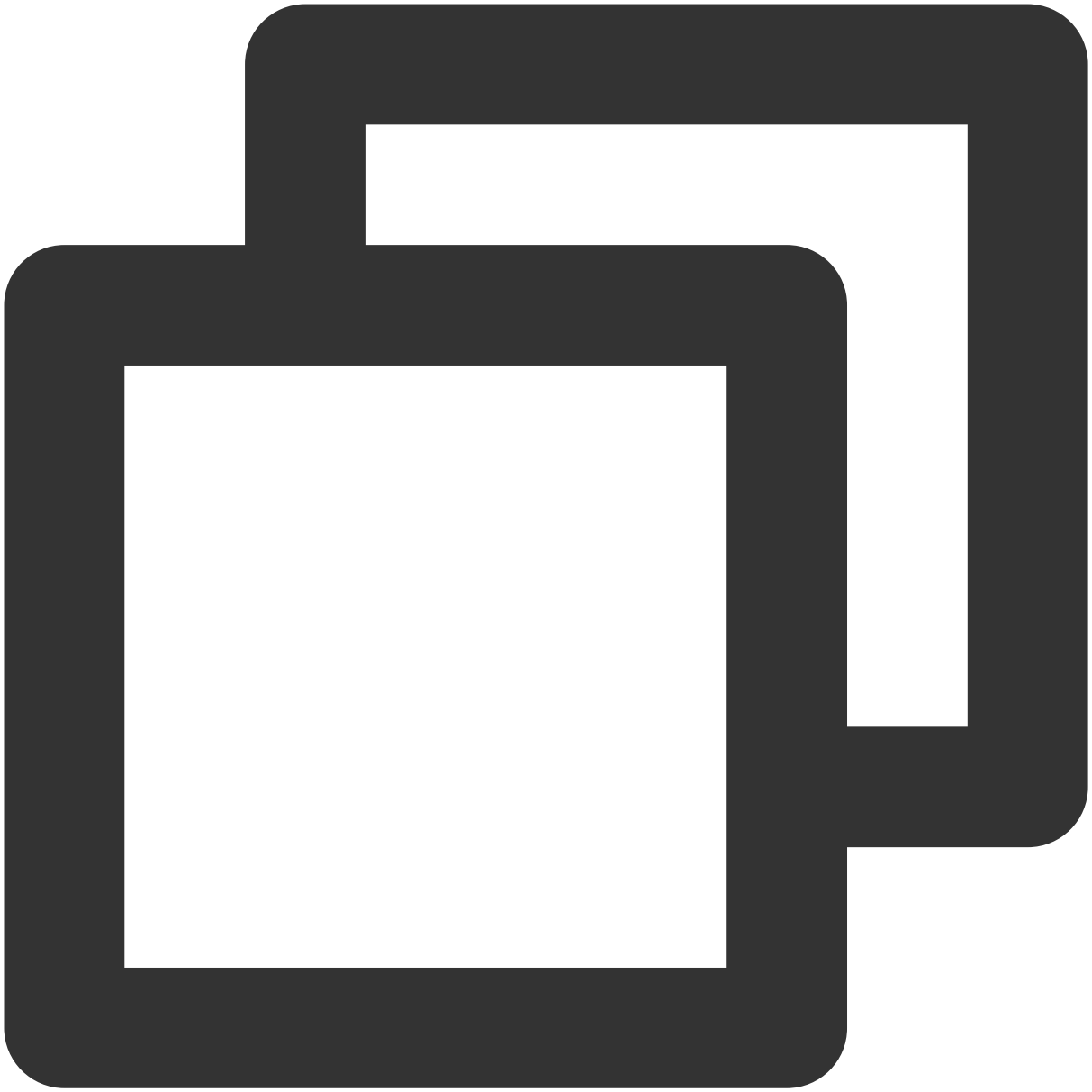
版权所有：腾讯云计算（北京）有限责任公司



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

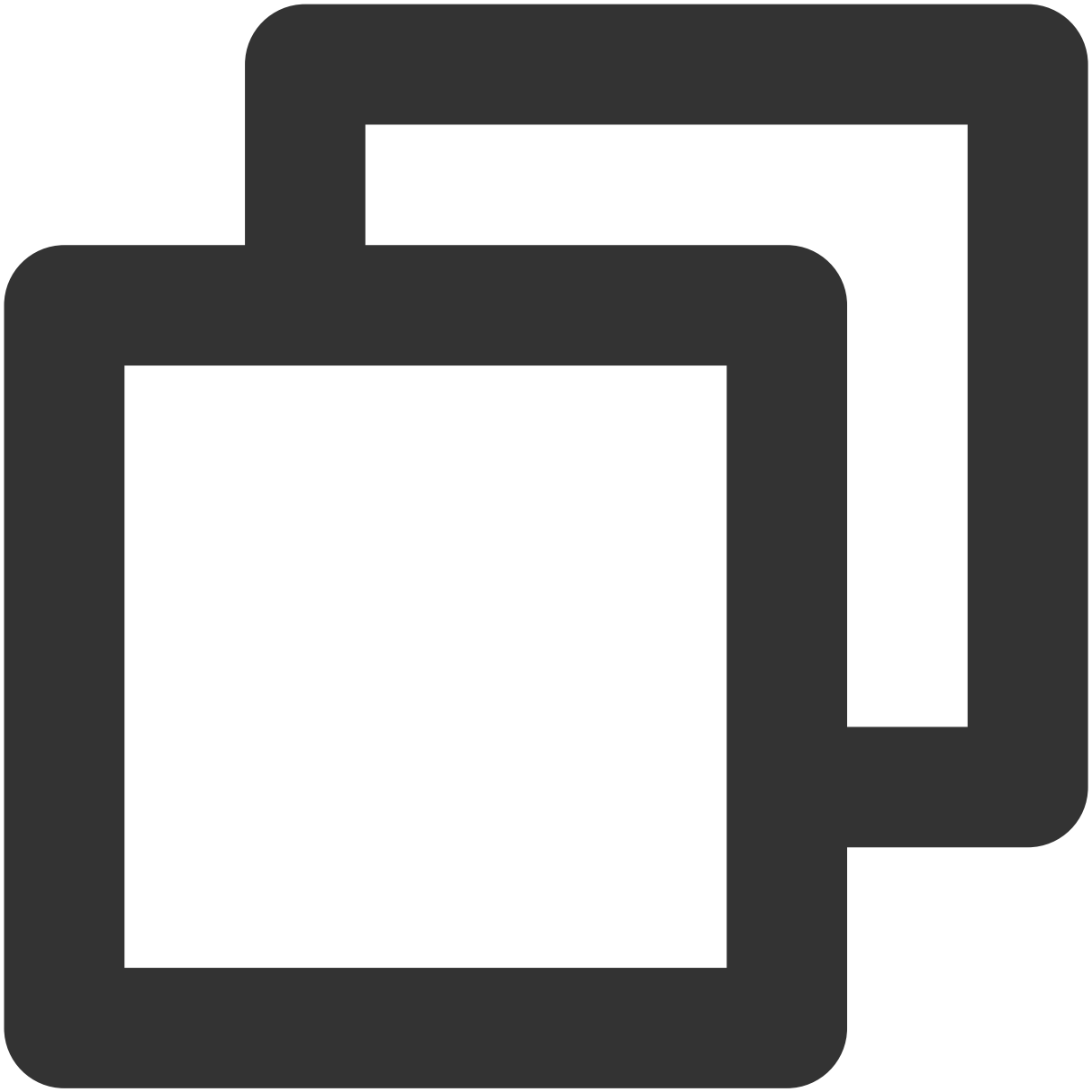


版权所有：腾讯云计算（北京）有限责任公司



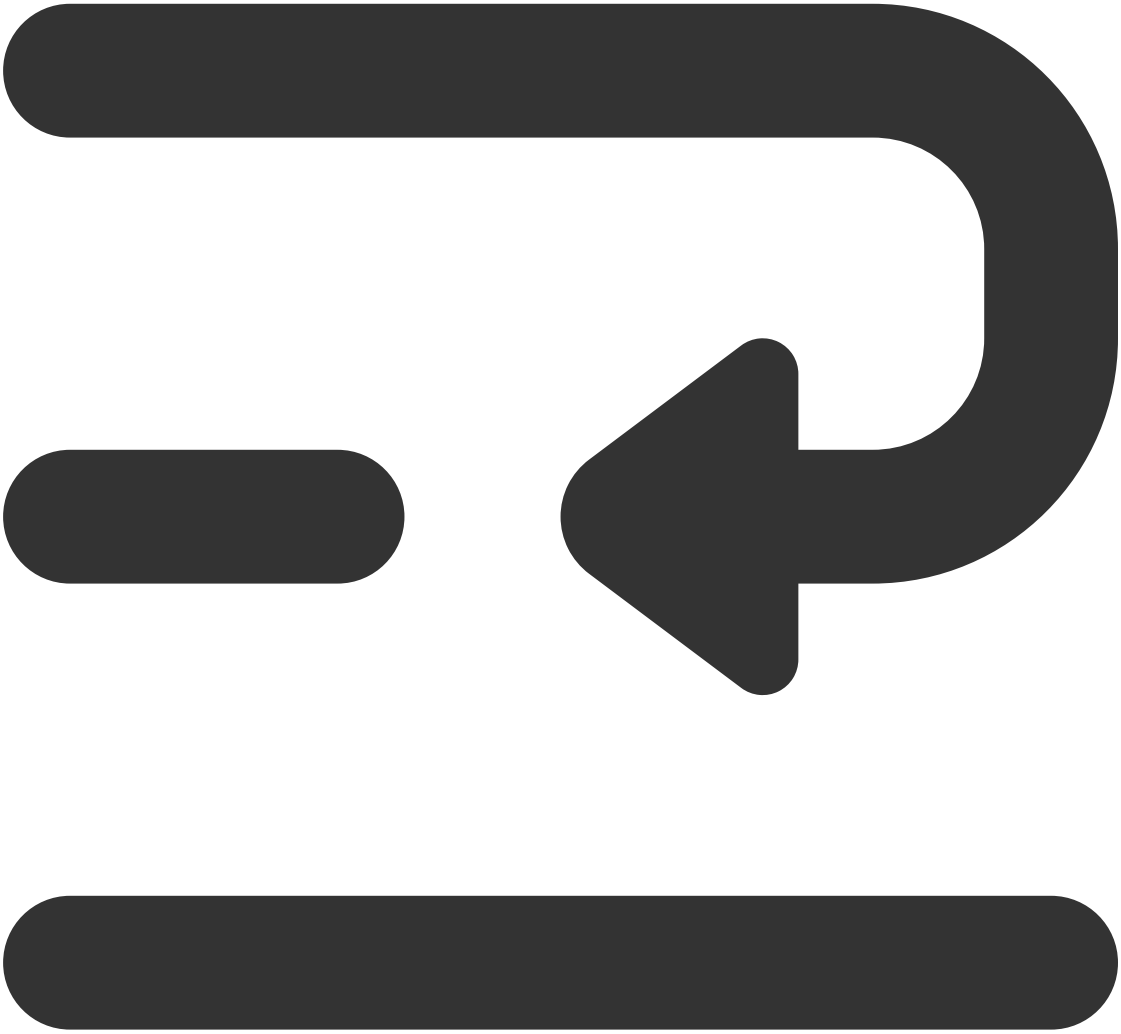
```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av ./node_modules/@tencentc
```

通过 npm 方式下载 TUIKit 组件：

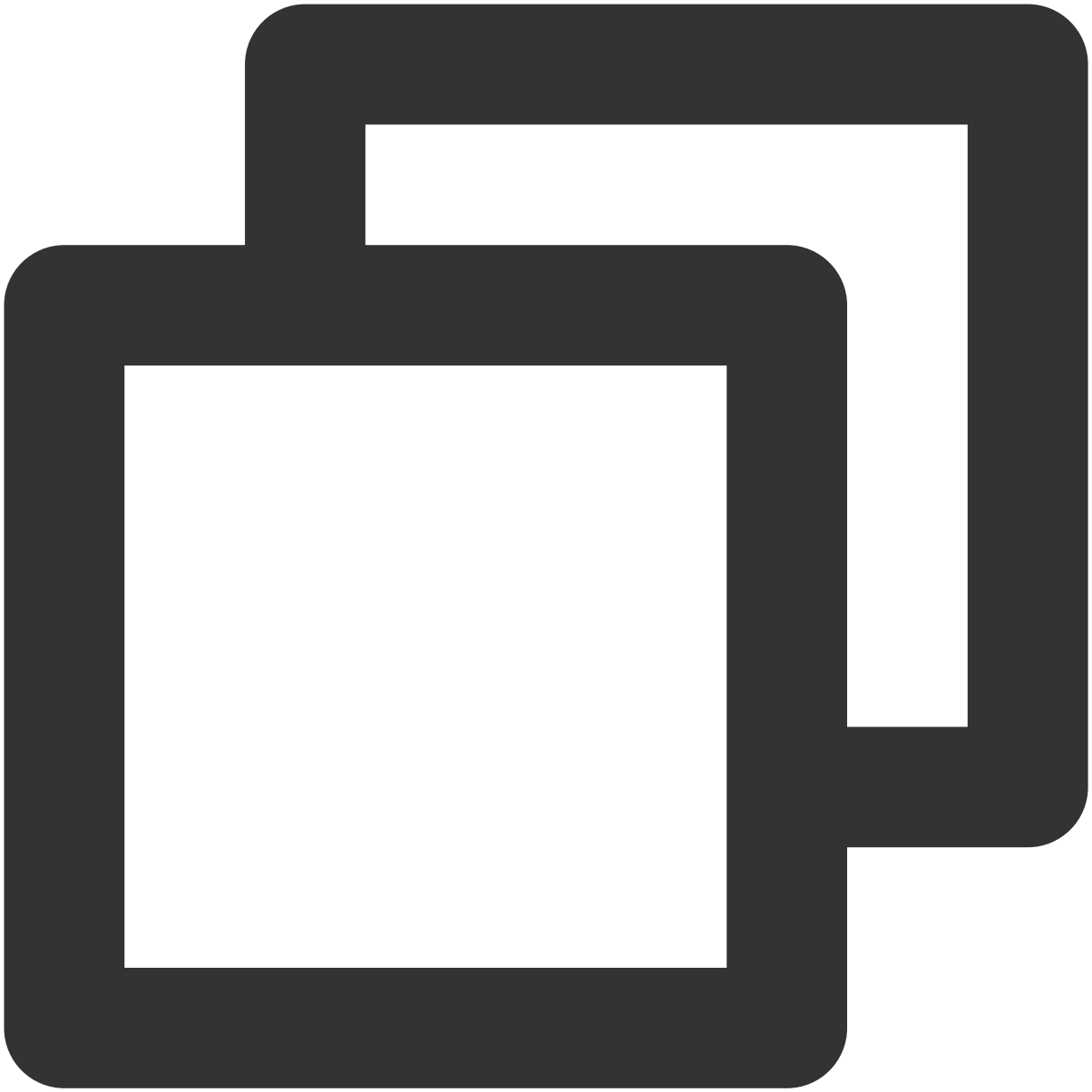


```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

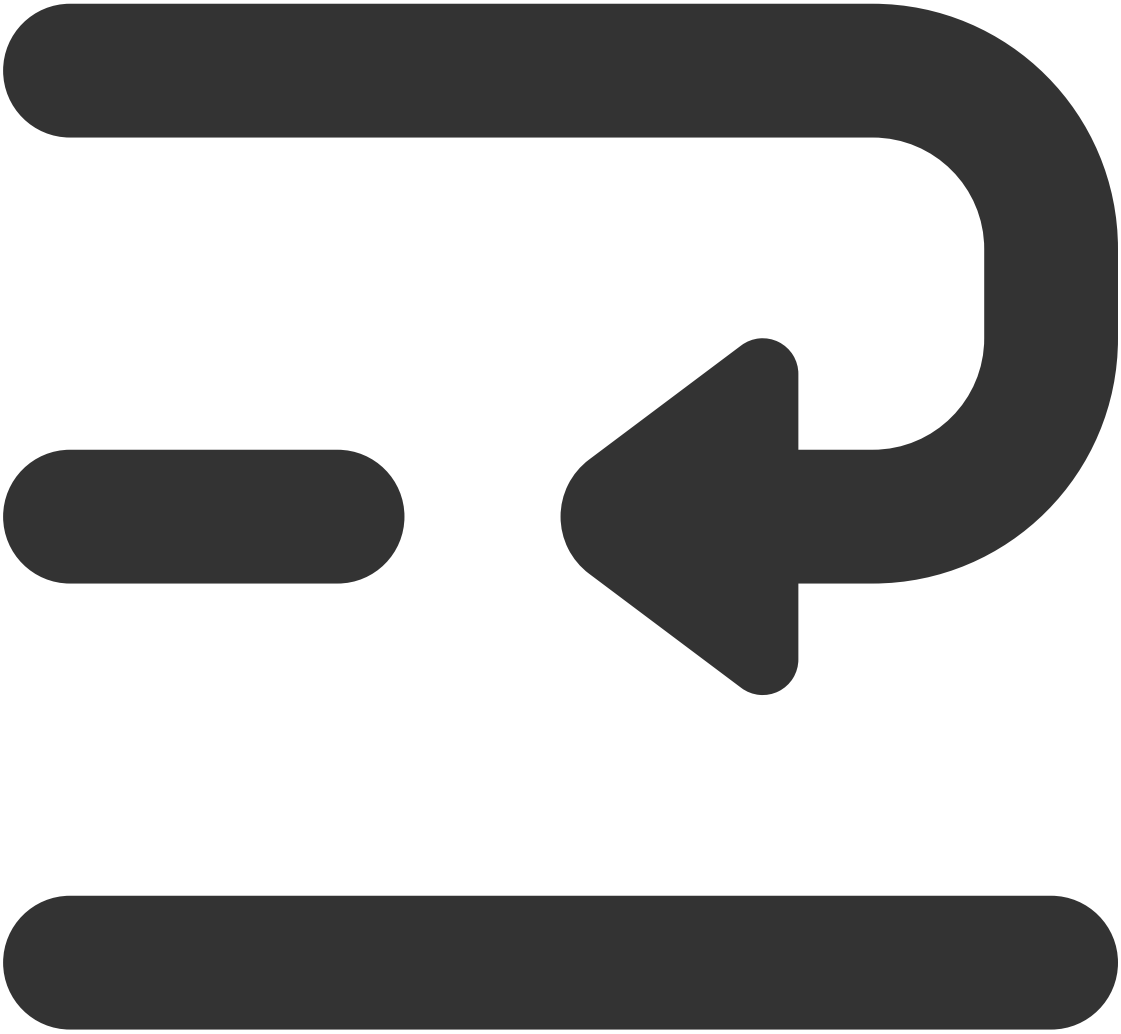
为了方便您后续的拓展，建议您将 TUIKit 组件复制到自己工程的 pages 目录下，请在自己的项目根目录下执行以下命令：



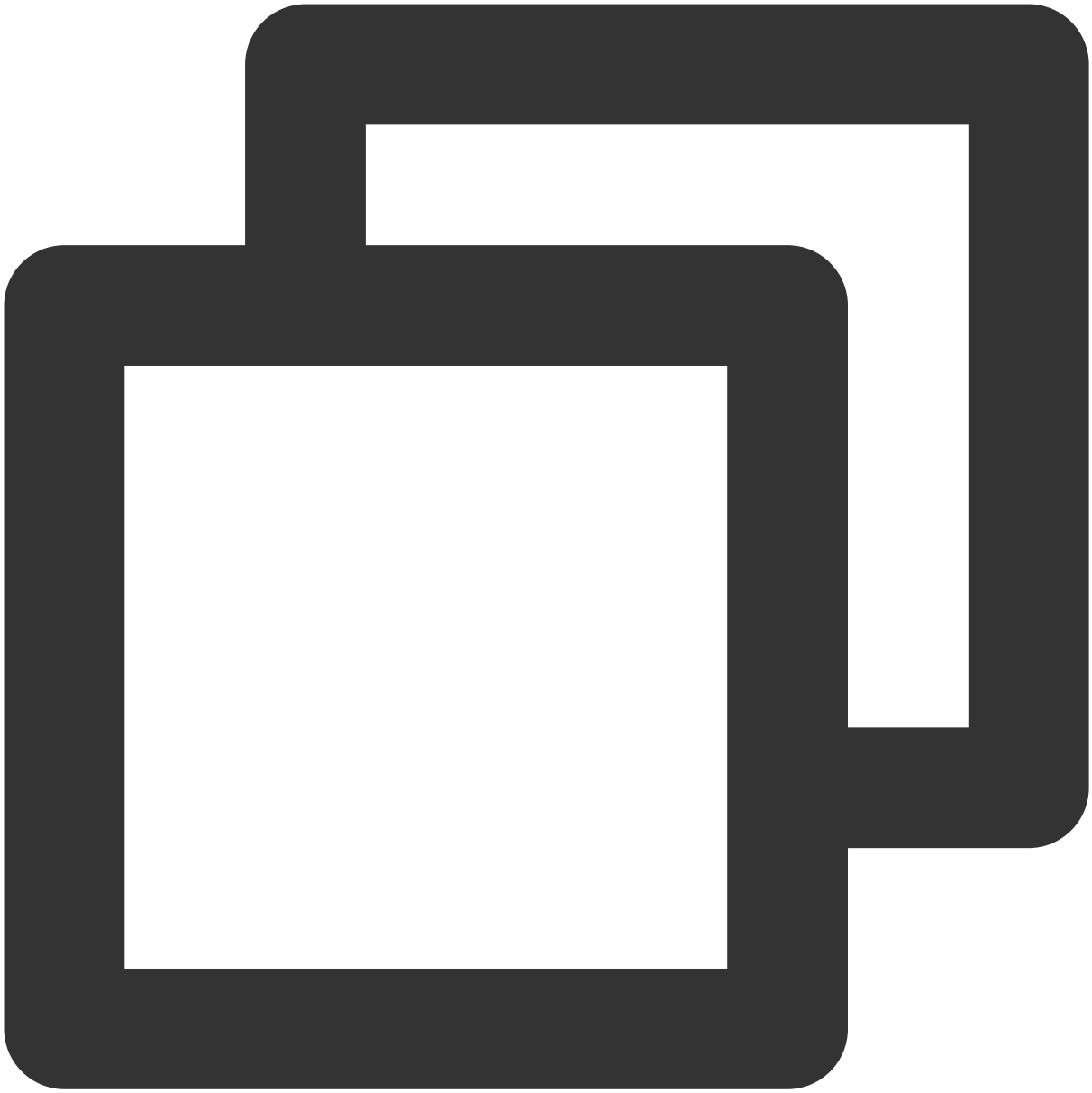
版权所有：腾讯云计算（北京）有限责任公司



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



版权所有：腾讯云计算（北京）有限责任公司

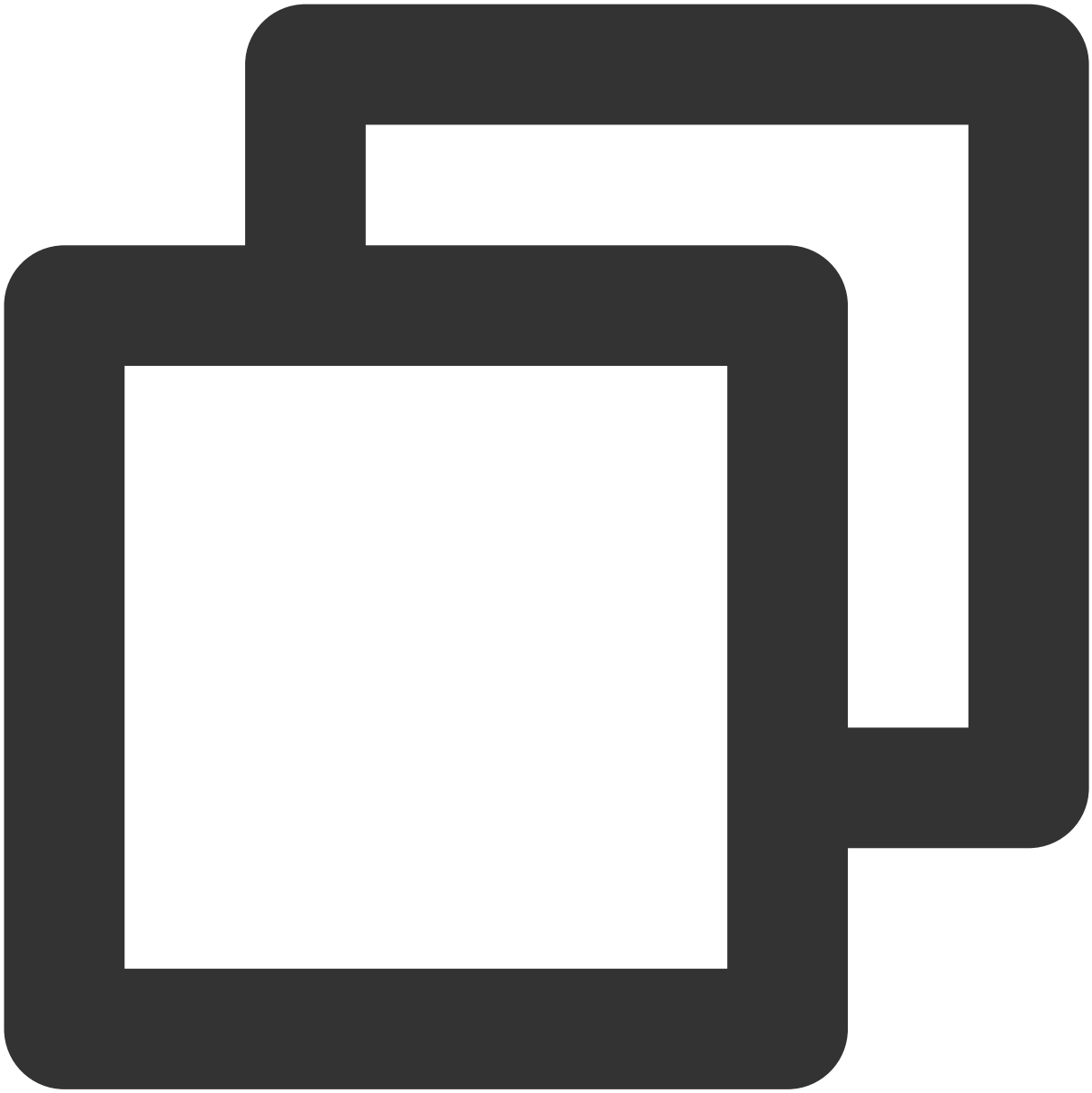


```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin .\\TUIKit\\tui-cu
```

步骤3：引入 TUIKit

1. 工程配置

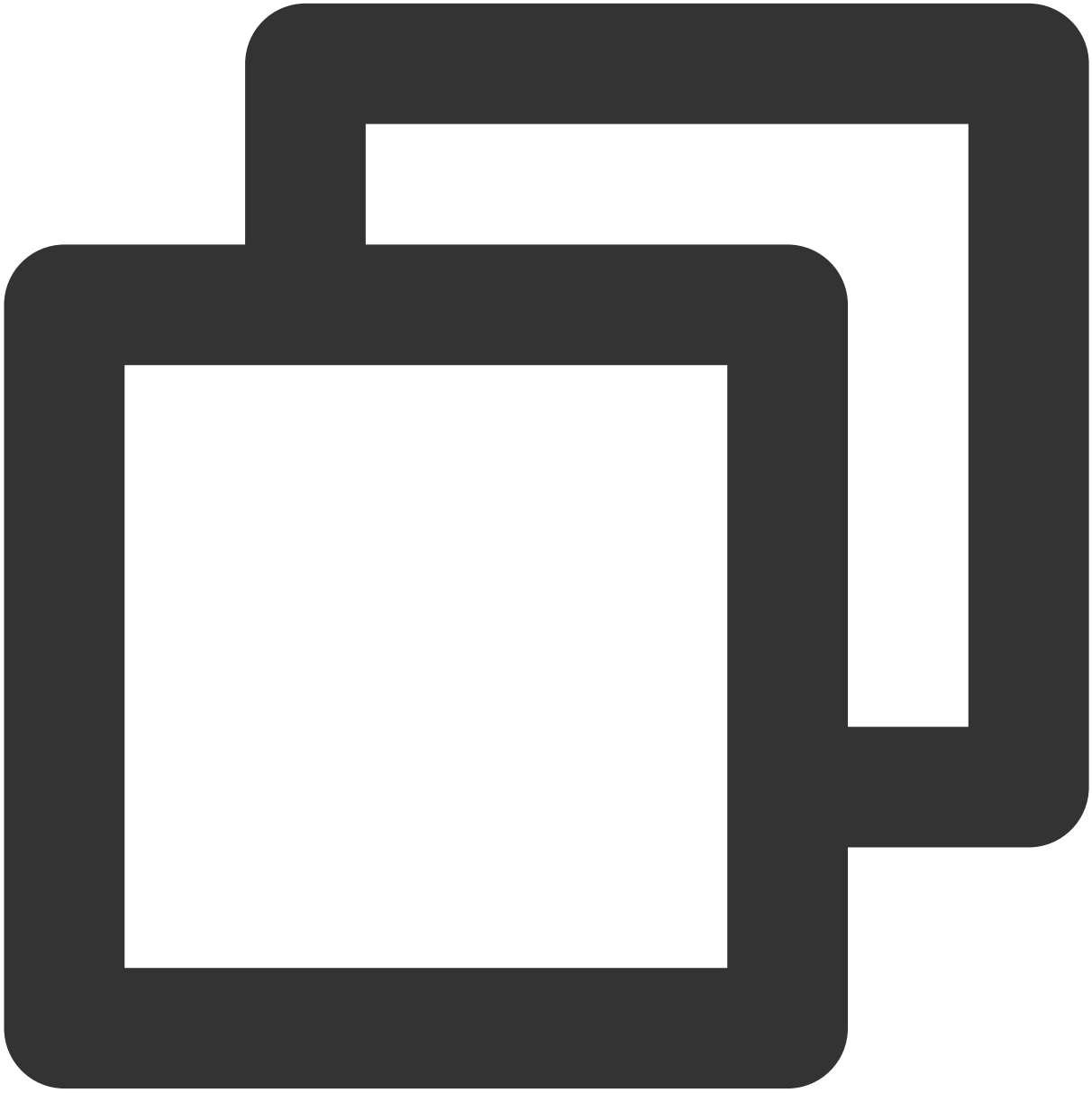
在根目录下创建 vue.config.js （vue3 项目请忽略此步骤）



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

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

在 `manifest.json` 文件的源码视图中开启分包配置



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

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

2. 集成 TUIKit

注意：

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

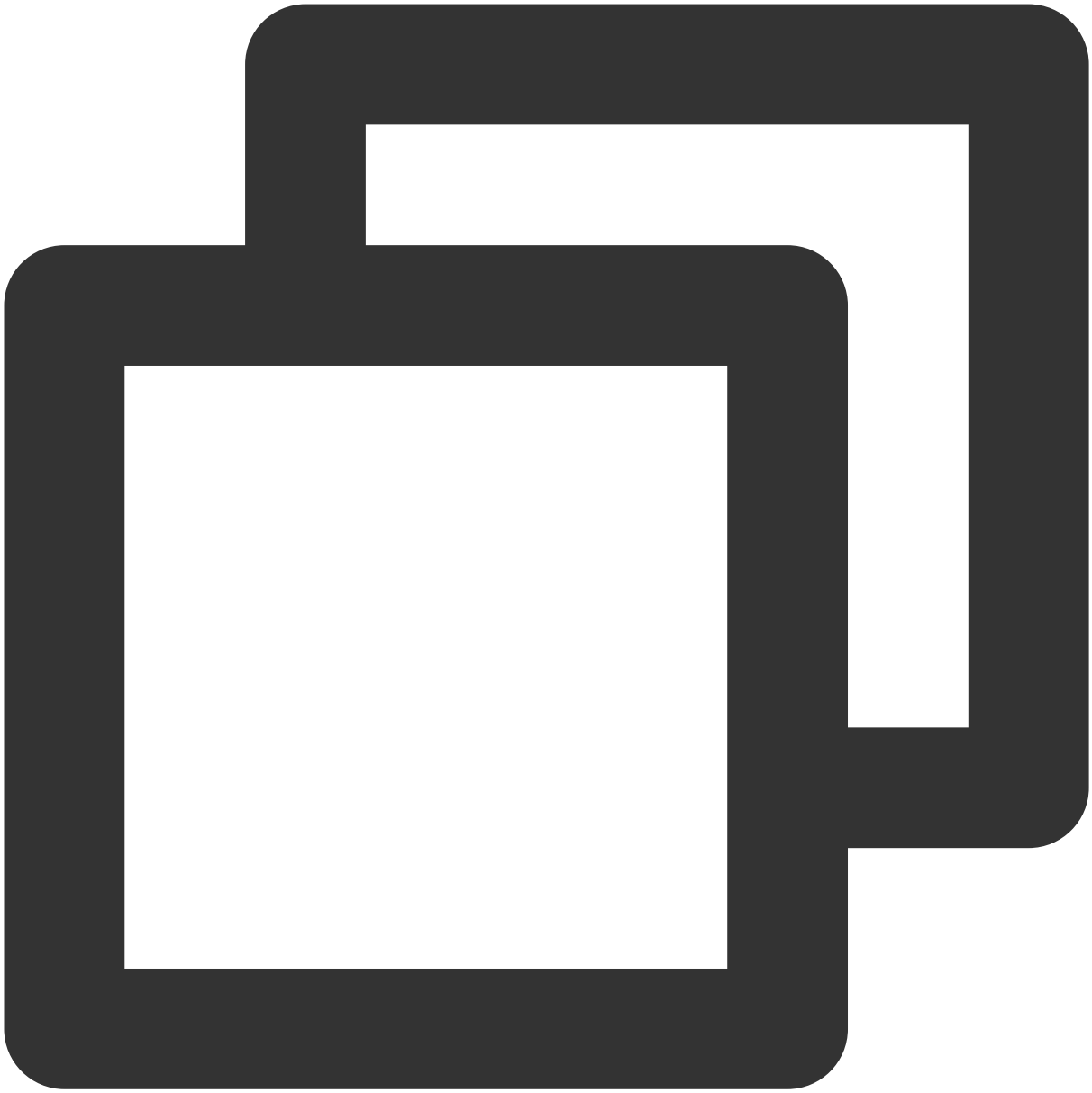
main.js 文件

pages.json 文件

App.vue 文件

小程序分包首页

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



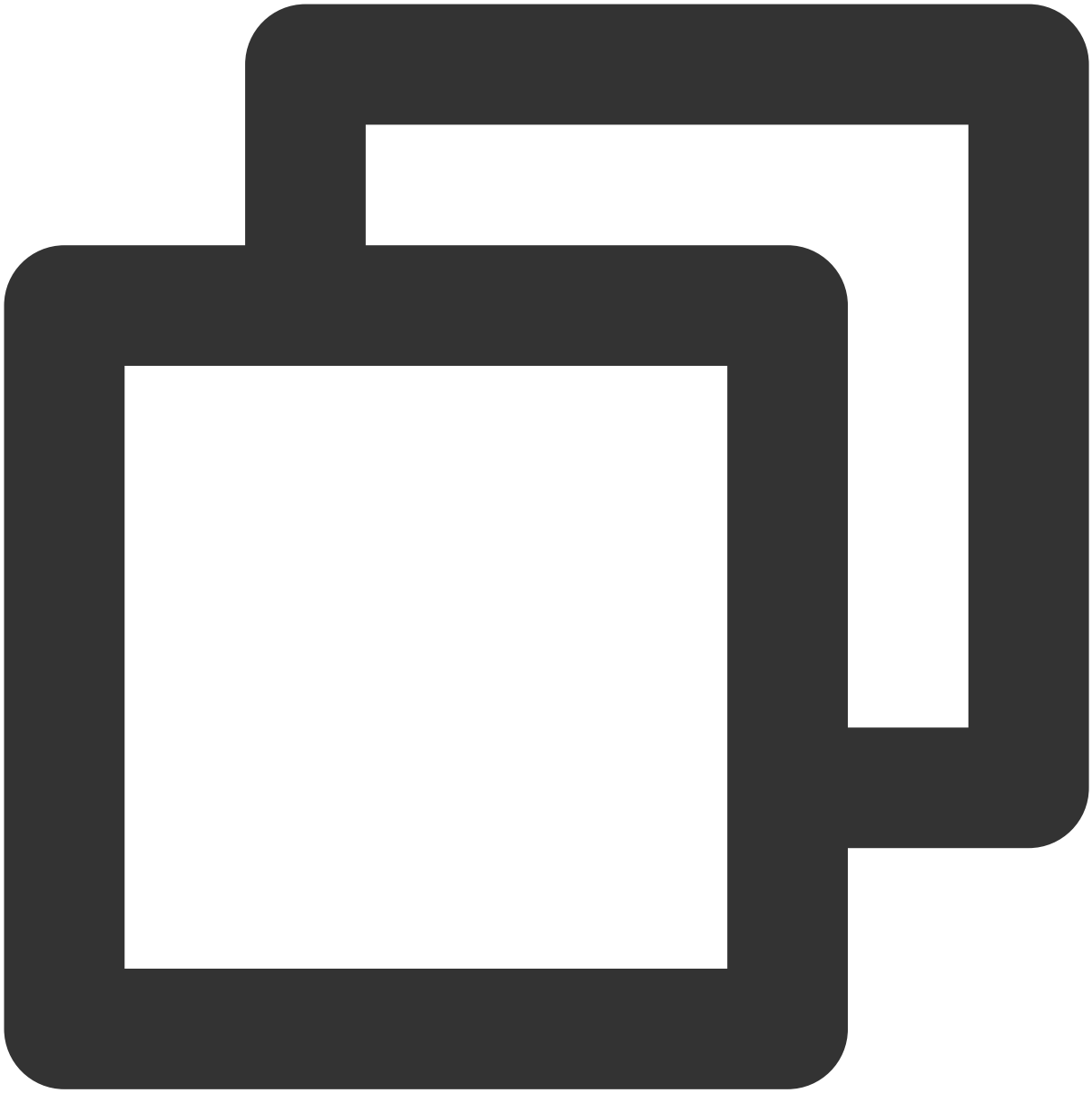
```
// 引入主包依赖
import TencentCloudChat  from "@tencentcloud/chat";
import TUICore from "@tencentcloud/tui-core";

import App from './App';

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

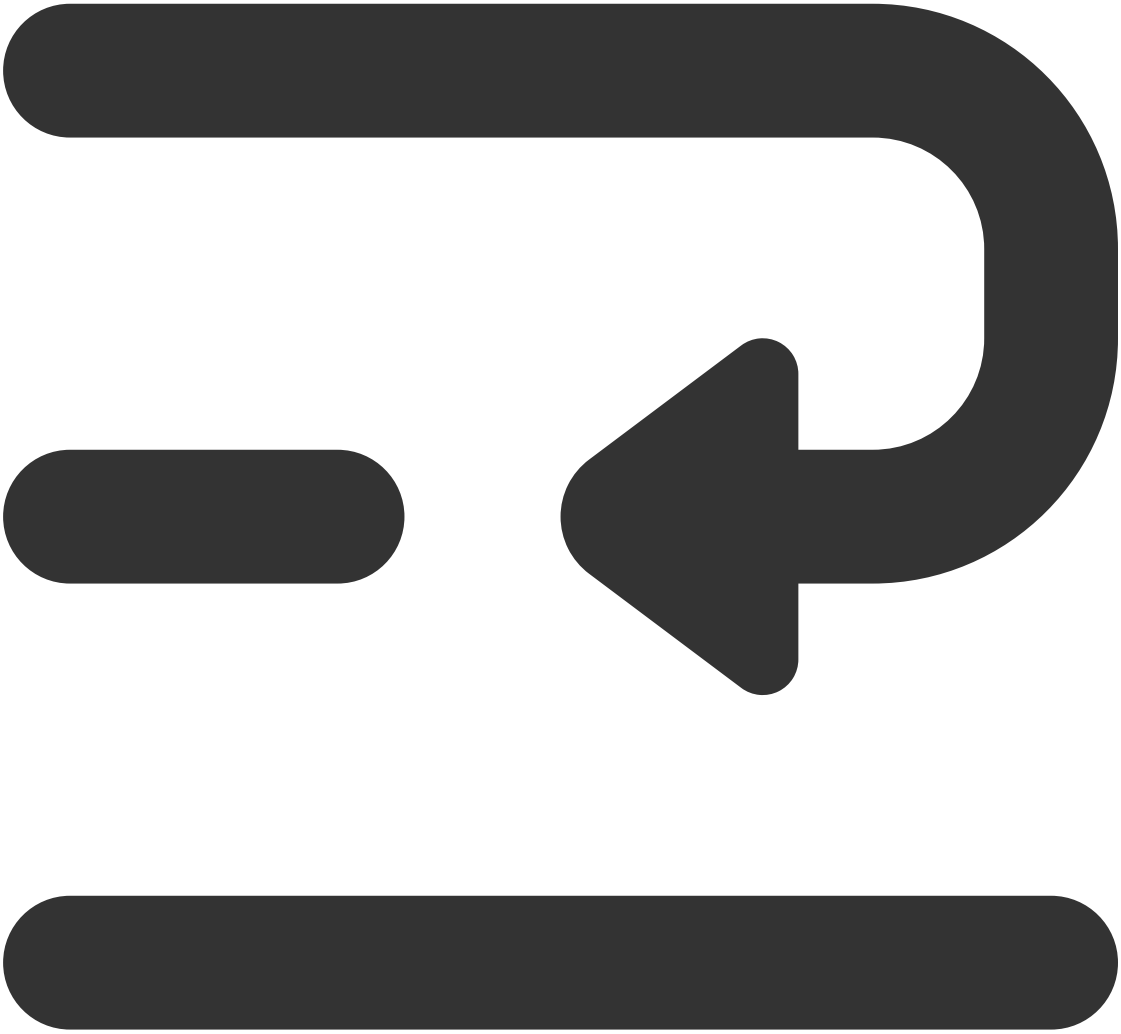
```
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

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

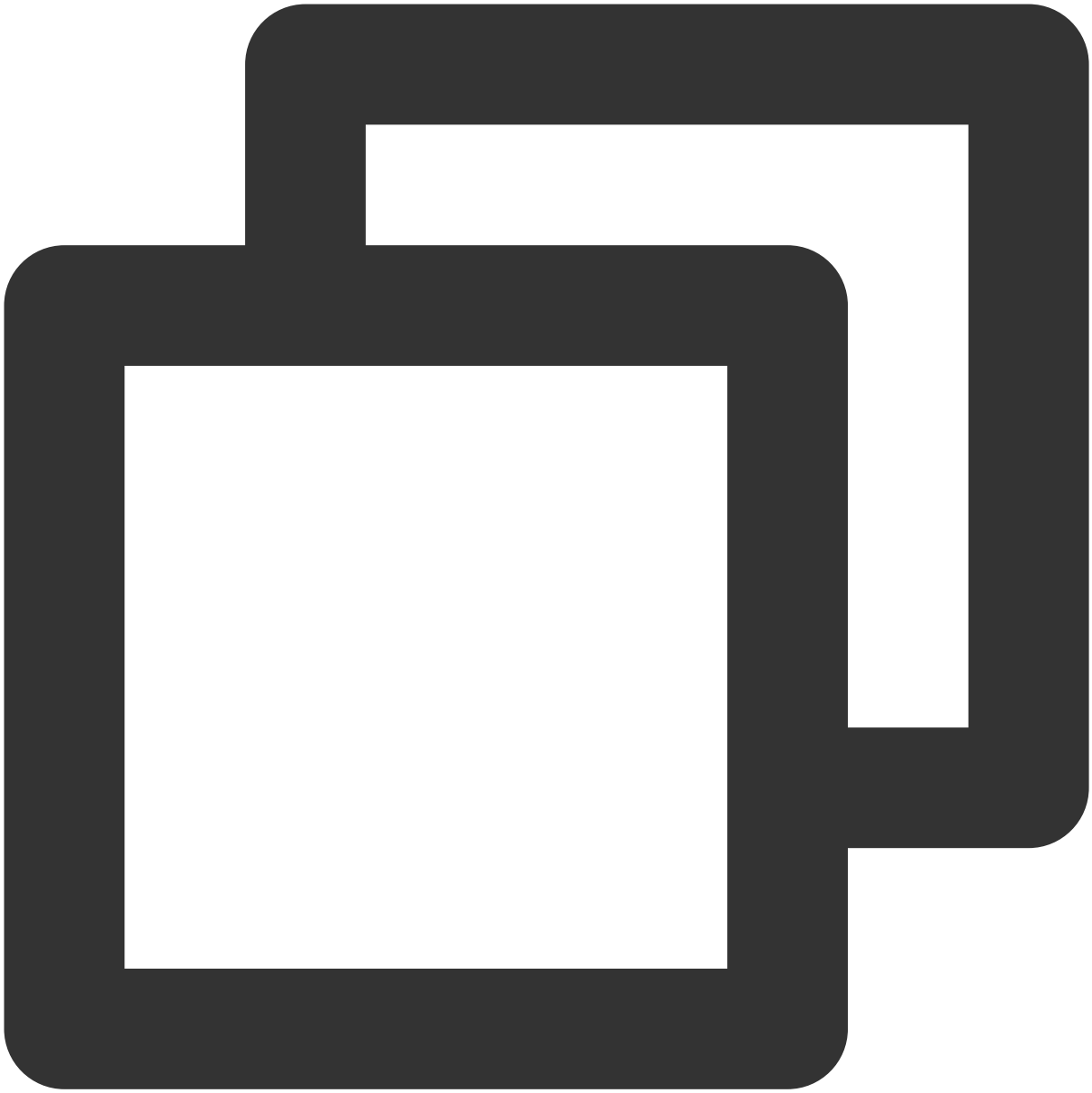


```
{
 "pages": [{
  "path": "pages/index/index" // 您的项目首页
 }],
 "subPackages": [{
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIChat/index",
    "style": {
     "navigationBarTitleText": "腾讯云 IM"
```

```
      }
     },
      // 集成 chat 组件，必须配置该路径：视频播放
     {
      "path": "components/TUIChat/video-play",
      "style": {
       "navigationBarTitleText": "腾讯云 IM"
      }
     },
     {
      "path": "components/TUIChat/web-view",
      "style": {
       "navigationBarTitleText": "腾讯云 IM"
      }
     },
     {
      "path": "components/TUIContact/index",
      "style": {
       "navigationBarTitleText": "腾讯云 IM"
      }
     },
     {
      "path": "components/TUIGroup/index",
      "style": {
       "navigationBarTitleText": "腾讯云 IM"
      }
     }
    ]
  }],
  "preloadRule": {
   "TUIKit/components/TUIChat/index": {
    "network": "all",
    "packages": ["TUIKit"]
   }
  },
  "globalStyle": {
   "navigationBarTextStyle": "black",
   "navigationBarTitleText": "uni-app",
   "navigationBarBackgroundColor": "#F8F8F8",
   "backgroundColor": "#F8F8F8"
  }
}
```



版权所有：腾讯云计算（北京）有限责任公司



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

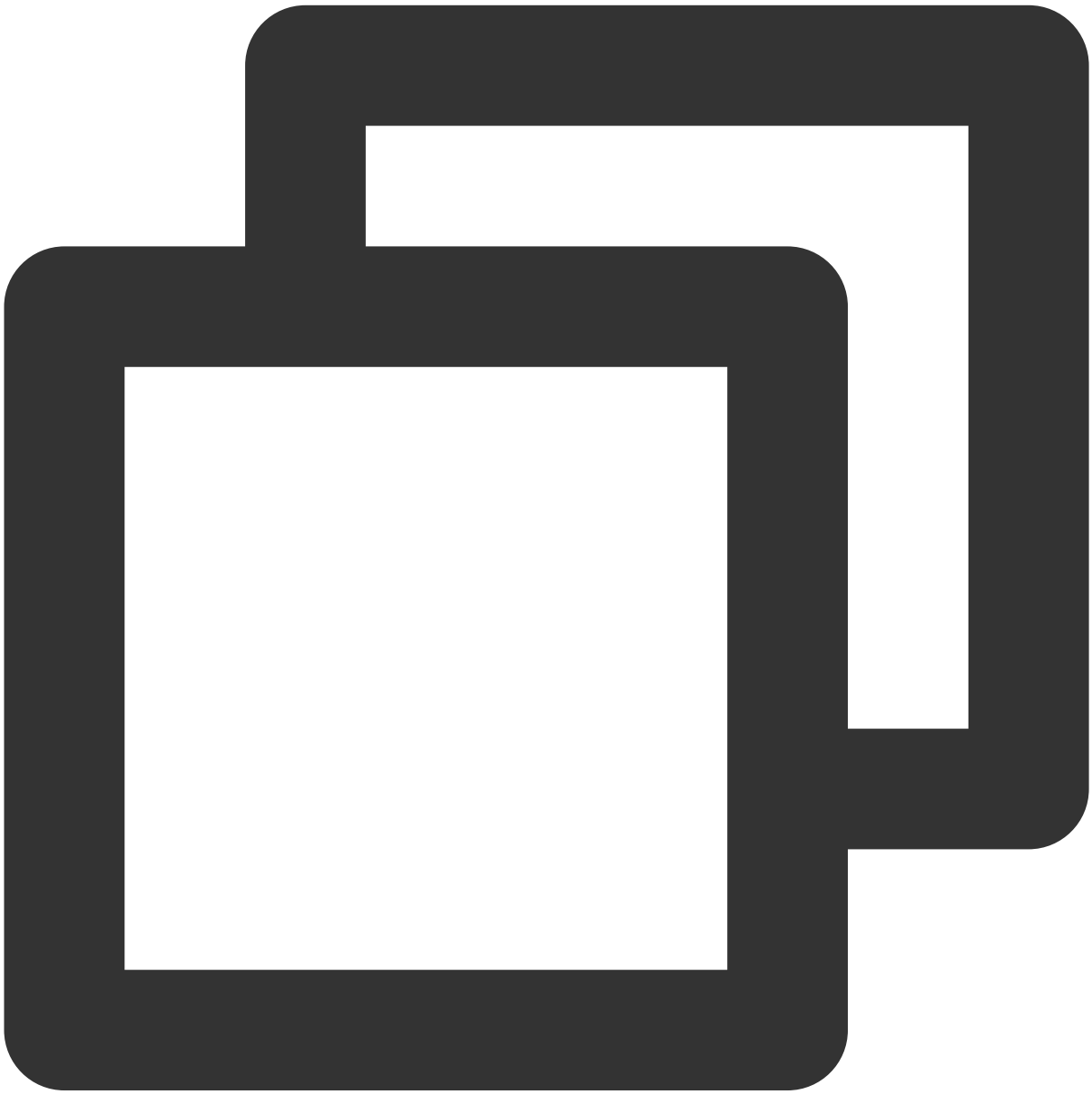
```
};
uni.$chat_userID = config.userID;
uni.$chat_SDKAppID = config.SDKAppID;
uni.$chat_secretKey = config.secretKey;

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

```
}
</style>
```

示例：小程序分包 TUIKit 首启动页面为 **TUIChat** 页面 （如果您不需要打包小程序，可忽略此配置页面）。

请在文件路径： `TUIKit/components/TUIChat/index.vue` 添加以下内容：

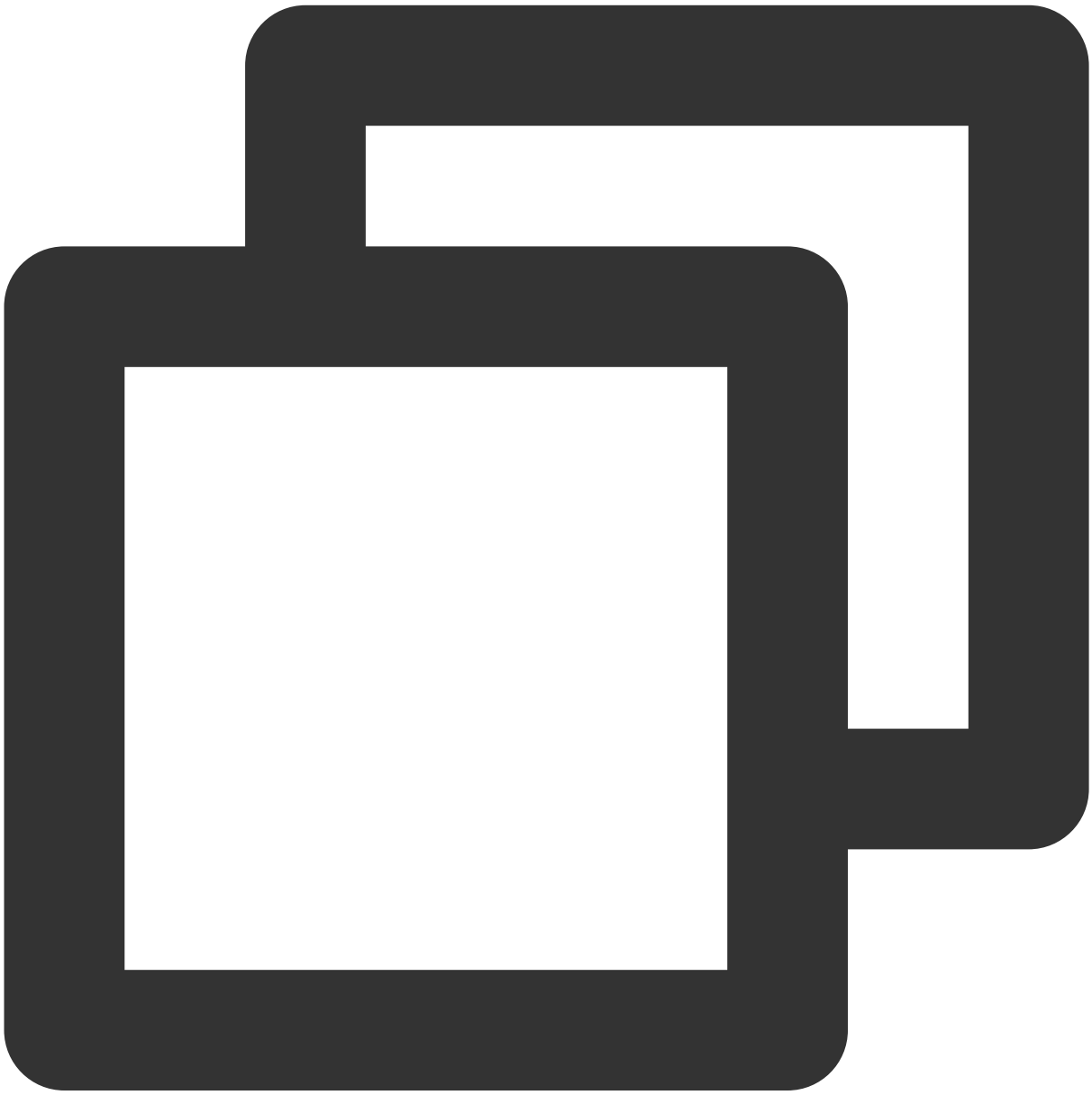


```
// #ifdef MP-WEIXIN
import { TUIChatKit, genTestUserSig } from "../../index.ts";
import { vueVersion, onMounted } from "../../adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";
import { onLoad } from '@dcloudio/uni-app';
```

```
// #endif
```

注意：

由于条件编译的兼容问题，以下条件编译代码必须写在 const 变量下边。



```
// TUIChatKit 初始化
// 注意：由于条件编译的兼容问题，以下条件编译代码必须写在 const 变量下边
// #ifdef MP-WEIXIN
TUIChatKit.init();
uni.$chat_userSig = genTestUserSig({
        userID: uni.$chat_userID,
```

```
        SDKAppID: uni.$chat_SDKAppID,
        secretKey: uni.$chat_secretKey
}).userSig;

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

如图所示：

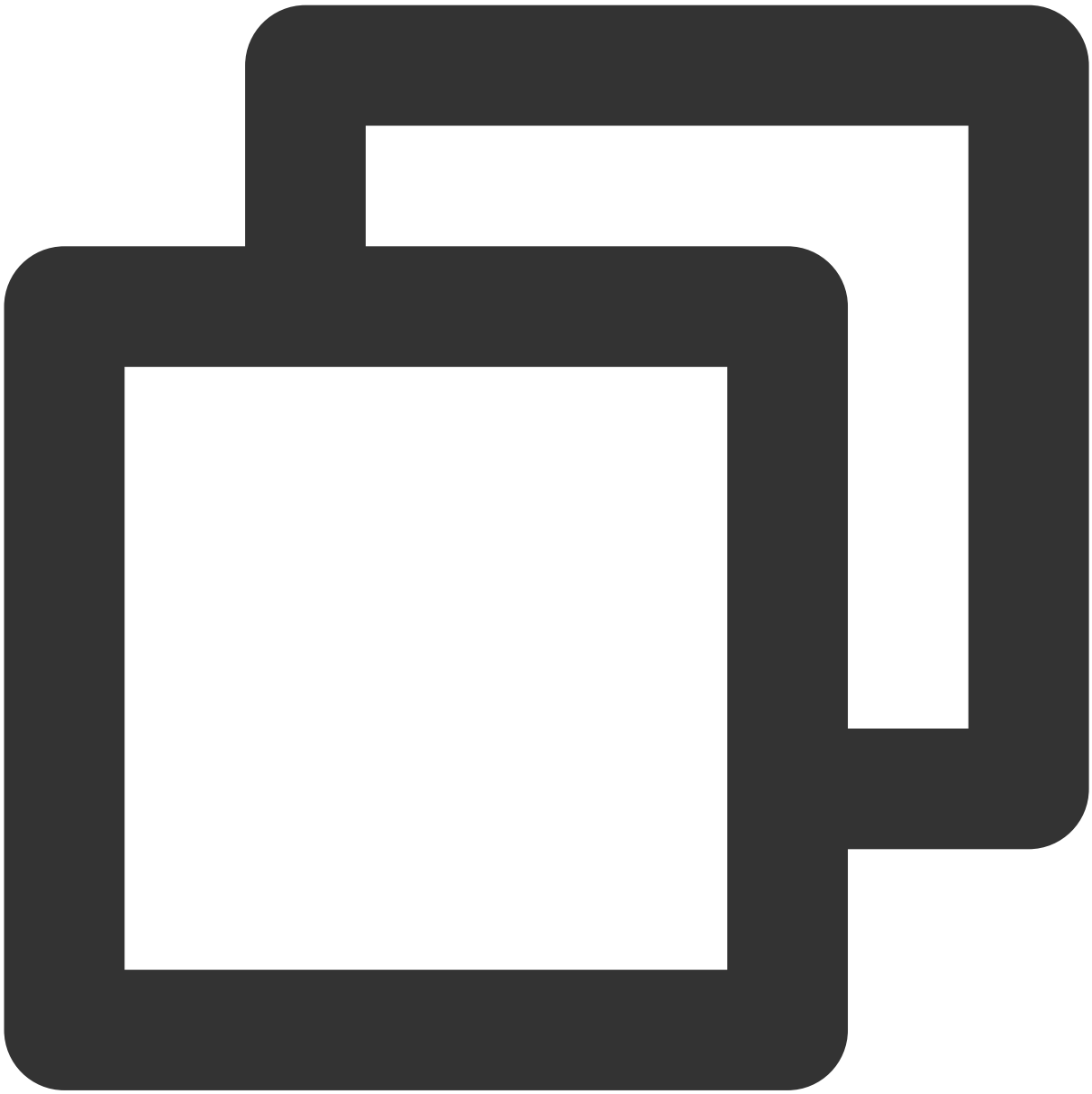
```
const isToolbarShow = ref<boolean>(!isUniFrameWork);
const messageInputRef = ref();
const currentConversationID = ref();
// 是否显示群组管理
const isGroup = ref(false);
const groupID = ref("");
const groupManageExt = ref<ExtensionInfo>(undefined);

// TUIChatKit 初始化
// 注意：由于条件编译的兼容问题，以下条件编译代码必须写在 const 变量后边
// #ifdef MP-WEIXIN
TUIChatKit.init();
uni.$chat_userSig = genTestUserSig({
  userID: uni.$chat_userID,
  SDKAppID: uni.$chat_SDKAppID,
  secretKey: uni.$chat_secretKey
}).userSig;

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

步骤4：获取 SDKAppID、secretKey、userID

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



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

获取 SDKAppID、secretKey

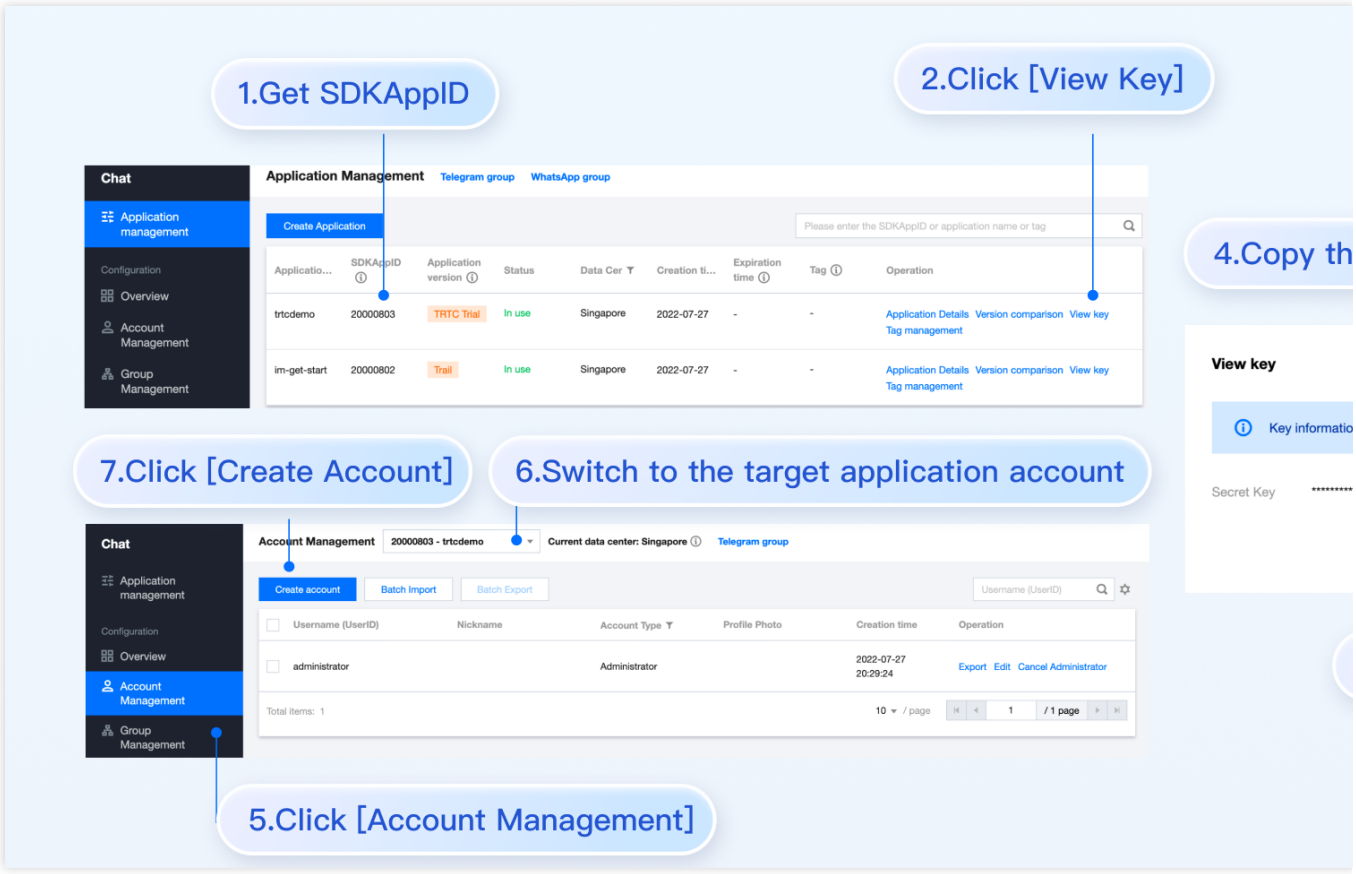
在 即时通信 IM 控制台中的应用管理页面下，可以看到您创建的应用，第二列即是 SDKAppID，然后点击操作中的查看密钥。网站会弹出查看密钥的对话框，然后点击显示密钥，即可查看密钥。

创建 userID 为 test-user1 的账户

说明：

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

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



步骤5：在项目主包首页中配置单聊和群聊的入口

在 pages/index 文件夹下创建 index.vue 文件，填写 userID/groupID：

说明：

conversationID：会话 ID。会话ID组成方式：

C2C${userID}（单聊）

GROUP${groupID}（群聊）

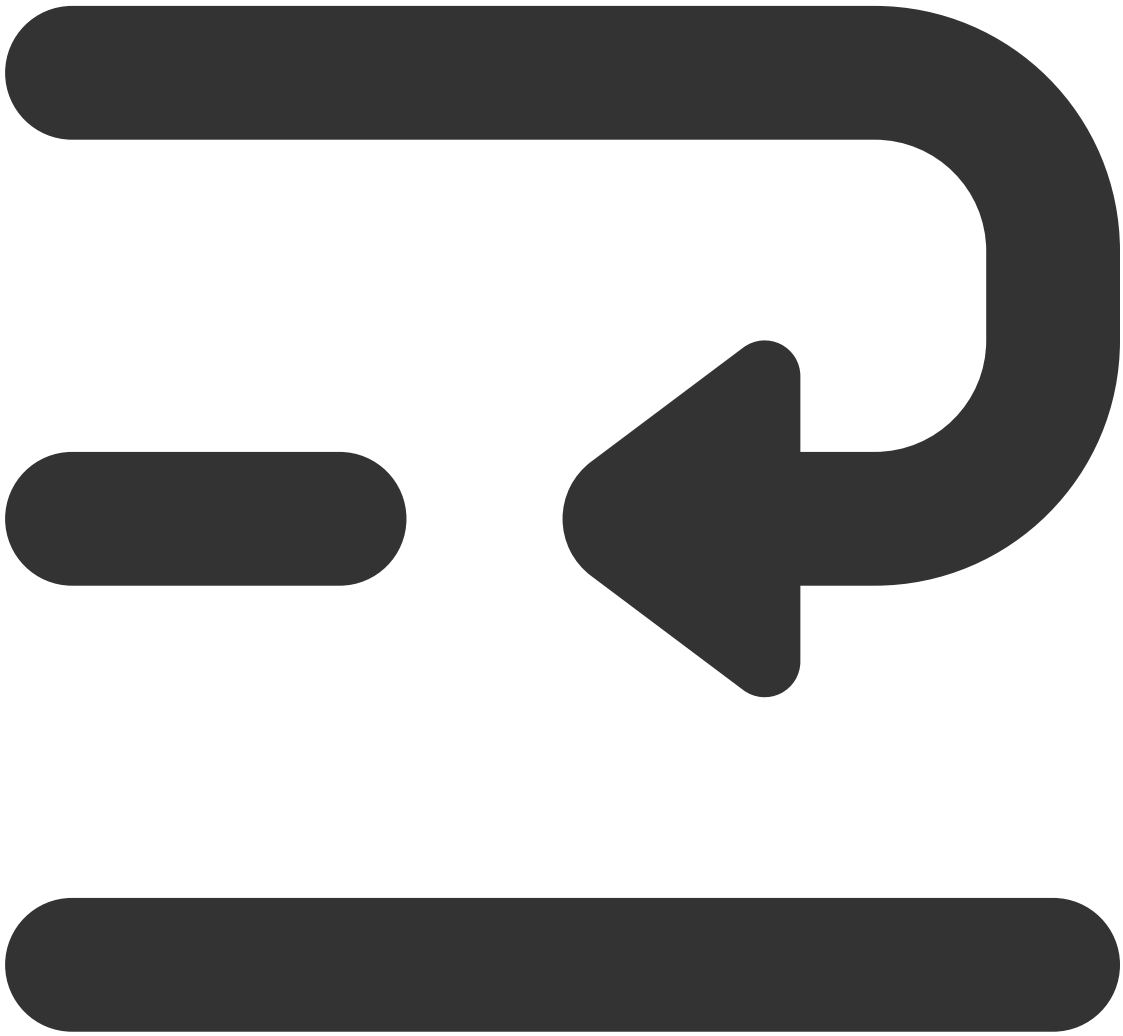
关于群聊：

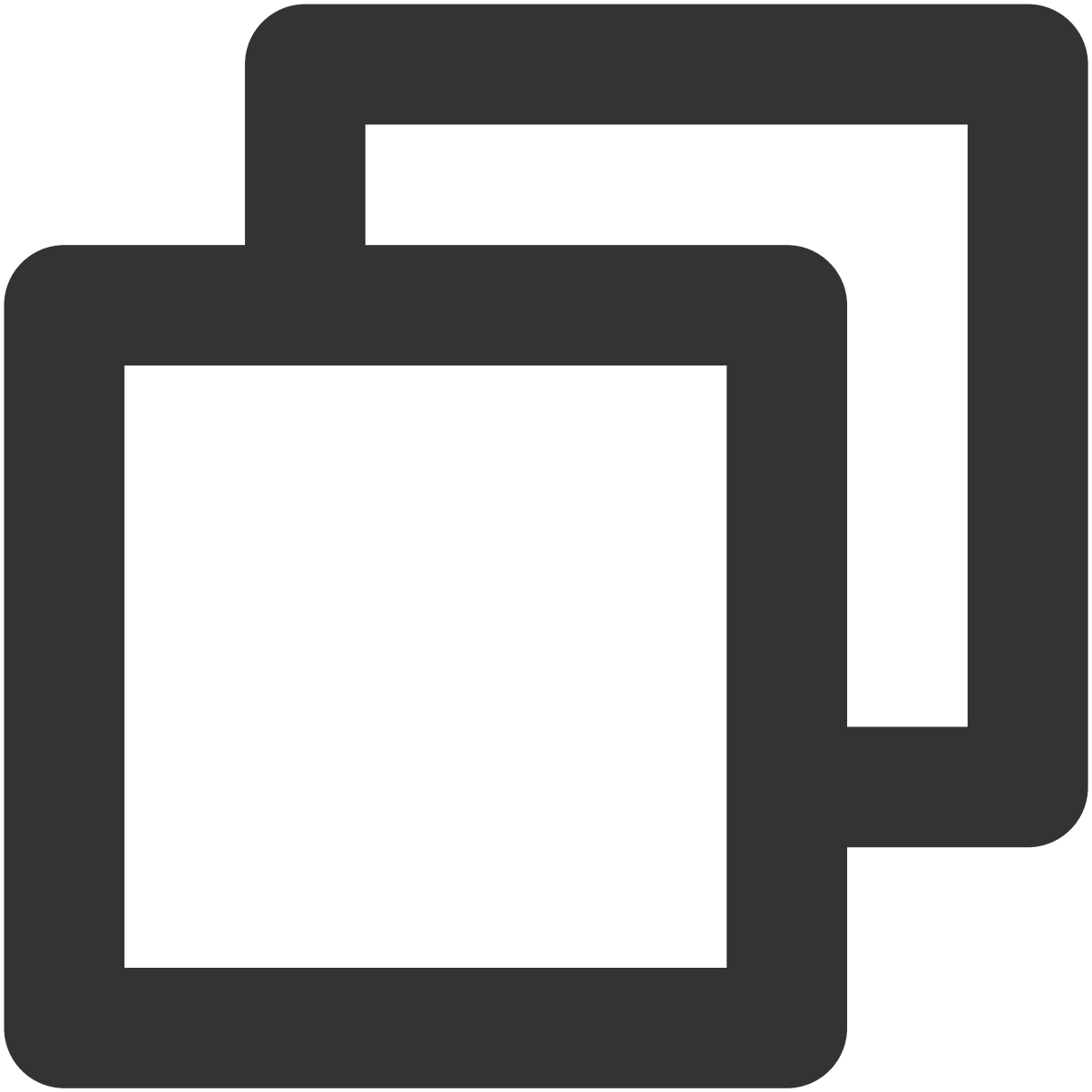
通过调用 createGroup 创建群组后获取对应的 groupID

如果是直播群， 需要通过调用 API joinGroup加入群，才可以进行聊天。

进入聊天

通过调用 switchConversation，进入聊天页面。





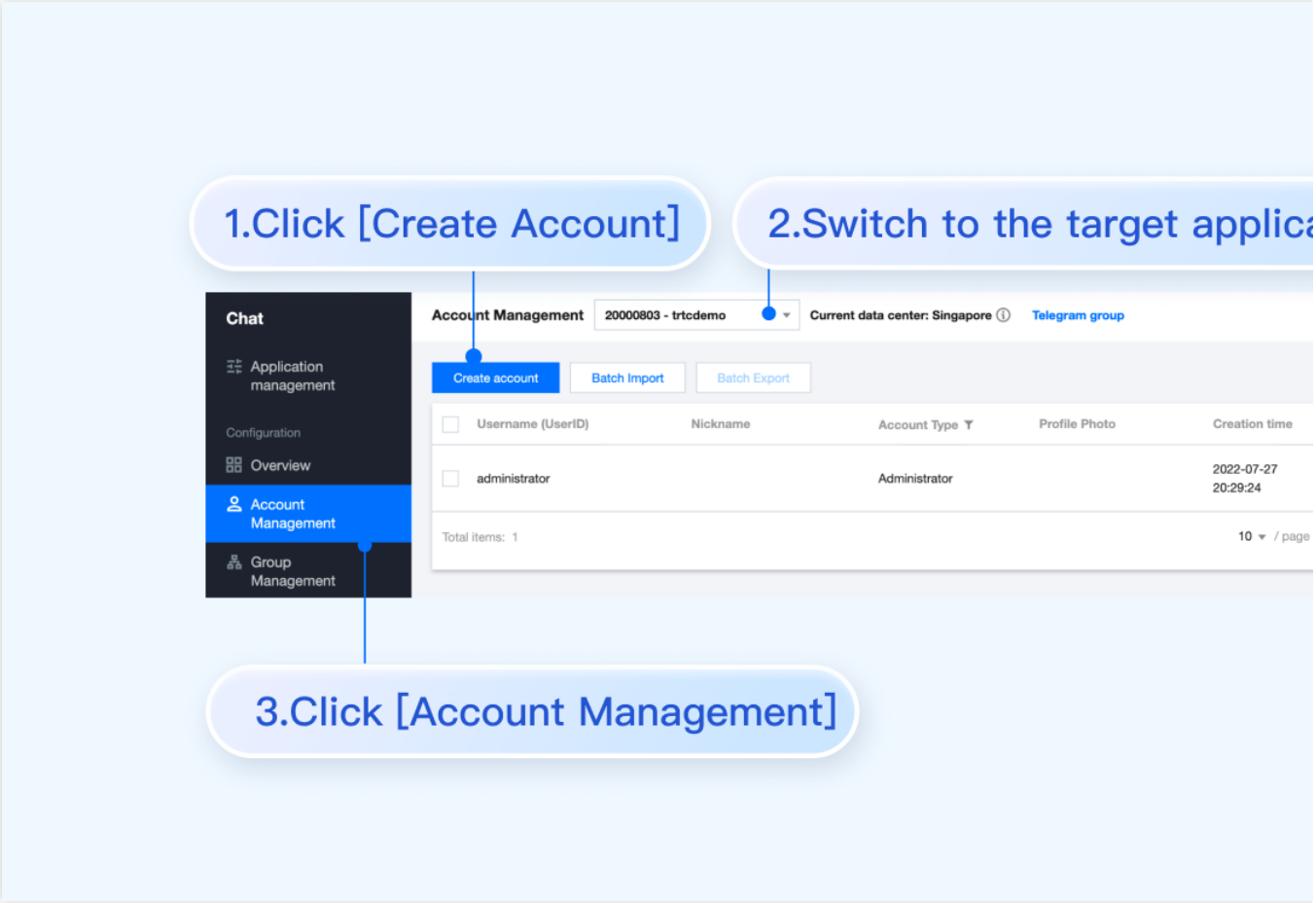
```
<template>
  <div class="TUI-chat">
    <p class="TUI-chat-button" @click="openChat">打开 1v1 聊天</p>
    <p class="TUI-chat-button" @click="openGroupChat">打开群聊</p>
  </div>
</template>
<script>
import { TUIConversationService } from "@tencentcloud/chat-uikit-engine";
export default {
  components: {},
  data() {
```

```
    return {
      userID: "test-user2", // 请填写对端 userID 参考步骤6
      groupID: "", // 通过调用 API createGroup 可获取 groupID，具体可参考：https://web.s
    };
  },
  methods: {
    // 打开 1v1 聊天
    openChat() {
      // 切换会话进入聊天
      const conversationID = `C2C${this.userID}`;
      // #ifdef APP-PLUS || H5
      TUIConversationService.switchConversation(conversationID);
      // #endif
      uni.navigateTo({
        url: `/TUIKit/components/TUIChat/index?conversationID=${conversationID}`,
      });
    },
    // 打开群聊
    openGroupChat() {
      const conversationID = `GROUP${this.groupID}`;
      // #ifdef APP-PLUS || H5
      TUIConversationService.switchConversation(conversationID);
      // #endif
      uni.navigateTo({
        url: `/TUIKit/components/TUIChat/index?conversationID=${conversationID}`,
      });
    },
  },
};
</script>
<style lang="scss" scoped>
.TUI-chat {
  display: flex;
  flex-direction: column;
  align-items: center;
  height: 100%;
  &-button {
    width: 180px;
    padding: 10px 40px;
    background-color: #006eff;
    color: #fff;
    font-size: 16px;
    margin-top: 65px;
    border-radius: 30px;
    text-align: center;
  }
}
```

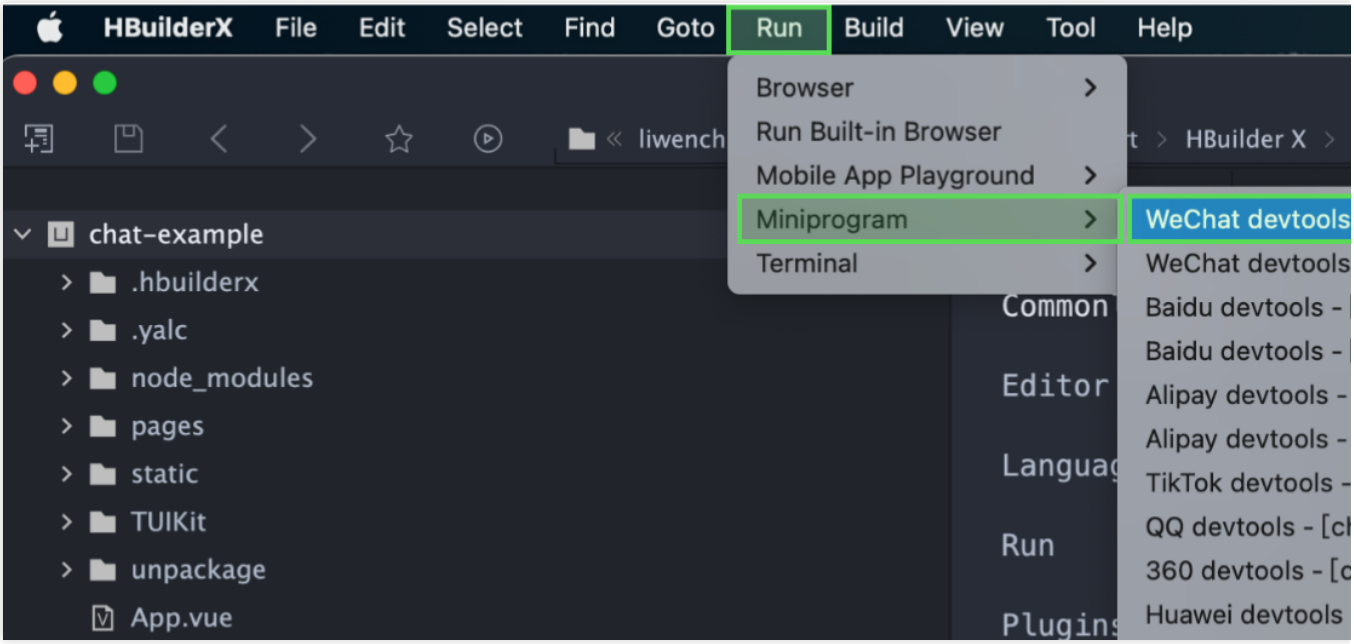
```
</style>
```

步骤6：发送您的第一条消息

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



2. 使用 HBuilderX 启动该项目，单击**运行 > 运行到小程序模拟器 > 微信开发者工具**。



3. 如果 HBuilderX 没有自动拉起微信开发者工具，请使用微信开发者工具手动打开编译后的项目。
使用微信开发者工具打开项目根目录下的 `unpackage/dist/dev/mp-weixin` 即可。
4. 打开项目后，在微信开发者工具的**详情 > 本地设置**中勾选**不校验合法域名、web-view（业务域名）、TLS版本以及 HTTPS 证书**。
5. 发起会话
单击**打开 1v1 聊天**，发送您的第一条消息。

更多高级特性

音视频通话 TUICallKit 插件

说明：

TUIKit 中默认没有集成 TUICallKit 音视频组件，TUICallKit 主要负责语音、视频通话。
如果您需要集成通话功能，可参考以下文档实现。
打包到 APP 请参考： 音视频通话（客户端）
打包到小程序请参考：音视频通话（小程序）
打包到 H5 请参考官：音视频通话（H5）
敬请期待。

TIMPush 离线推送插件

说明

TUIKit 中默认没有集成 TIMPush 离线推送插件。TIMPush 是腾讯云即时通信 IM Push 插件。目前离线推送支持 Android 和 iOS 平台，设备有：华为、小米、OPPO、vivo、魅族 和 苹果手机。
如果您需要在 App 中集成离线推送能力，请参见 uni-app 离线推送 实现。

敬请期待。

常见问题

1. 独立集成场景下，如何清除会话未读数？

答：在执行“步骤2 -> 集成 TUIKit 组件 -> 小程序分包首页”这一步时，TUIChat 在 onLoad 事件里调用了 `TUIConversationService.switchConversation()` 方法，该方法会主动清除当前会话的未读数，因此不需要手动清除会话未读数。

更多问题请参见 Uniapp FAQ。

技术咨询

点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

参考文档

UIKit （vue2 / vue3）相关：

chat-uikit-uniapp (vue2/vue3) github 源码

chat-uikit-uniapp npm 快速接入

ChatEngine 相关：

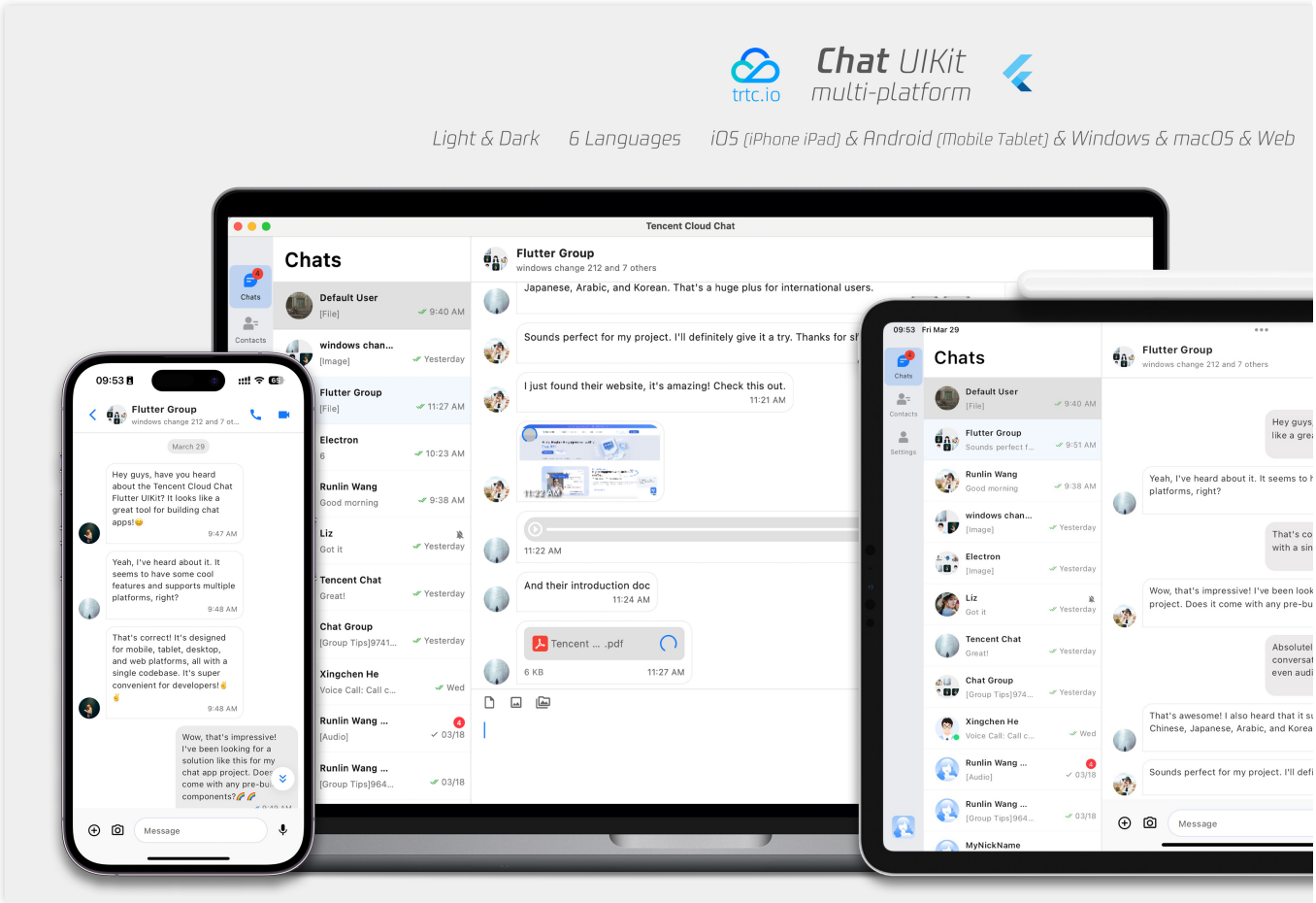
ChatEngine API 手册

ChatEngine npm

Flutter

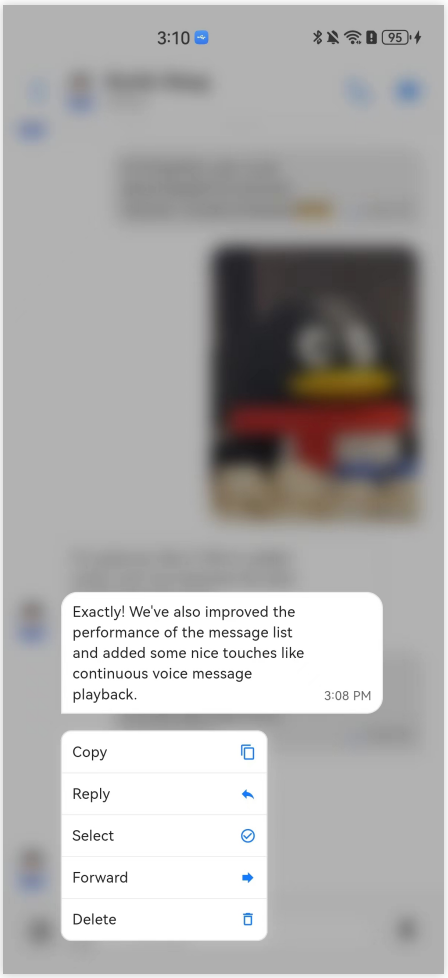
最近更新时间：2024-05-27 16:43:57

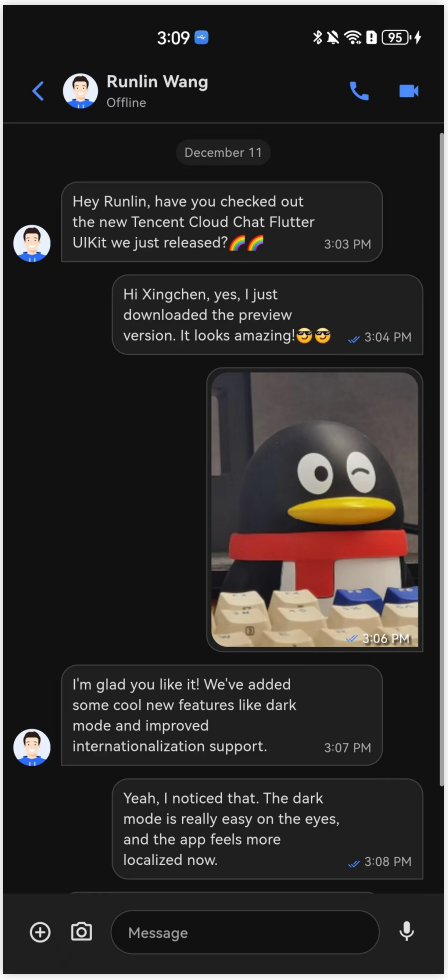
Flutter Chat UIKit 旨在为开发者提供一套全面的工具，以便轻松创建功能丰富的聊天应用程序。
它采用模块化方法构建，让您可以选择所需的组件，同时保持应用程序轻量级和高效。
其中的 `TencentCloudChatMessage` 组件提供了私信聊天（1V1）和群聊（Group）功能，支持对消息的多种操作，例如发送不同类型的消息、对消息长按回复/引用、查询消息已读回执详情等。
您可以仅集成 `TencentCloudChatMessage` 到您的 App 中。聊天界面使用场景非常广泛，例如房产中介咨询、在线医疗问诊、电商在线客服、保险远程定损等。
界面效果如下图所示：

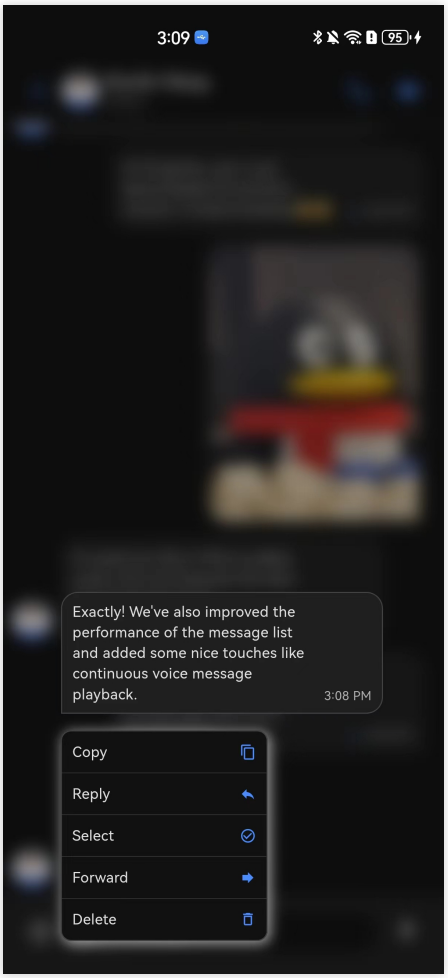


手机端
电脑端 & 平板端

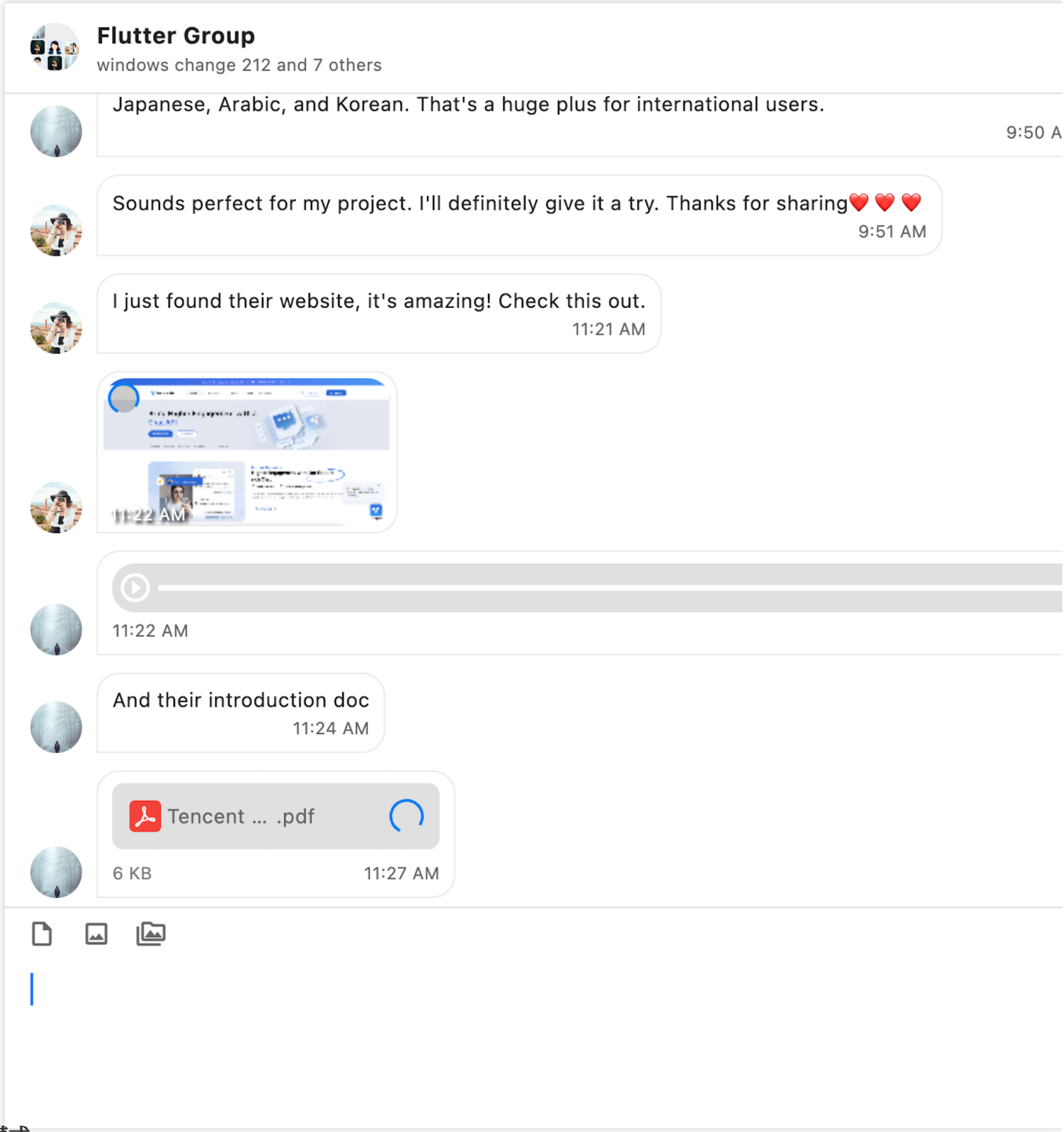
3:11
Runlin Wang
Offline
December 11
Hey Runlin, have you checked out the new Tencent Cloud Chat Flutter UIKit we just released?🌈🌈
3:03 PM
Hi Xingchen, yes, I just downloaded the preview version. It looks amazing!😎😎
3:04 PM
3:06 PM
I'm glad you like it! We've added some cool new features like dark mode and improved internationalization support.
3:07 PM
Yeah, I noticed that. The dark mode is really easy on the eyes, and the app feels more
That's awesome! I can't wait to see what else you guys have in store for the future.


3:10
Exactly! We've also improved the performance of the message list and added some nice touches like continuous voice message playback.
3:08 PM
Copy
Reply
Select
Forward
Delete

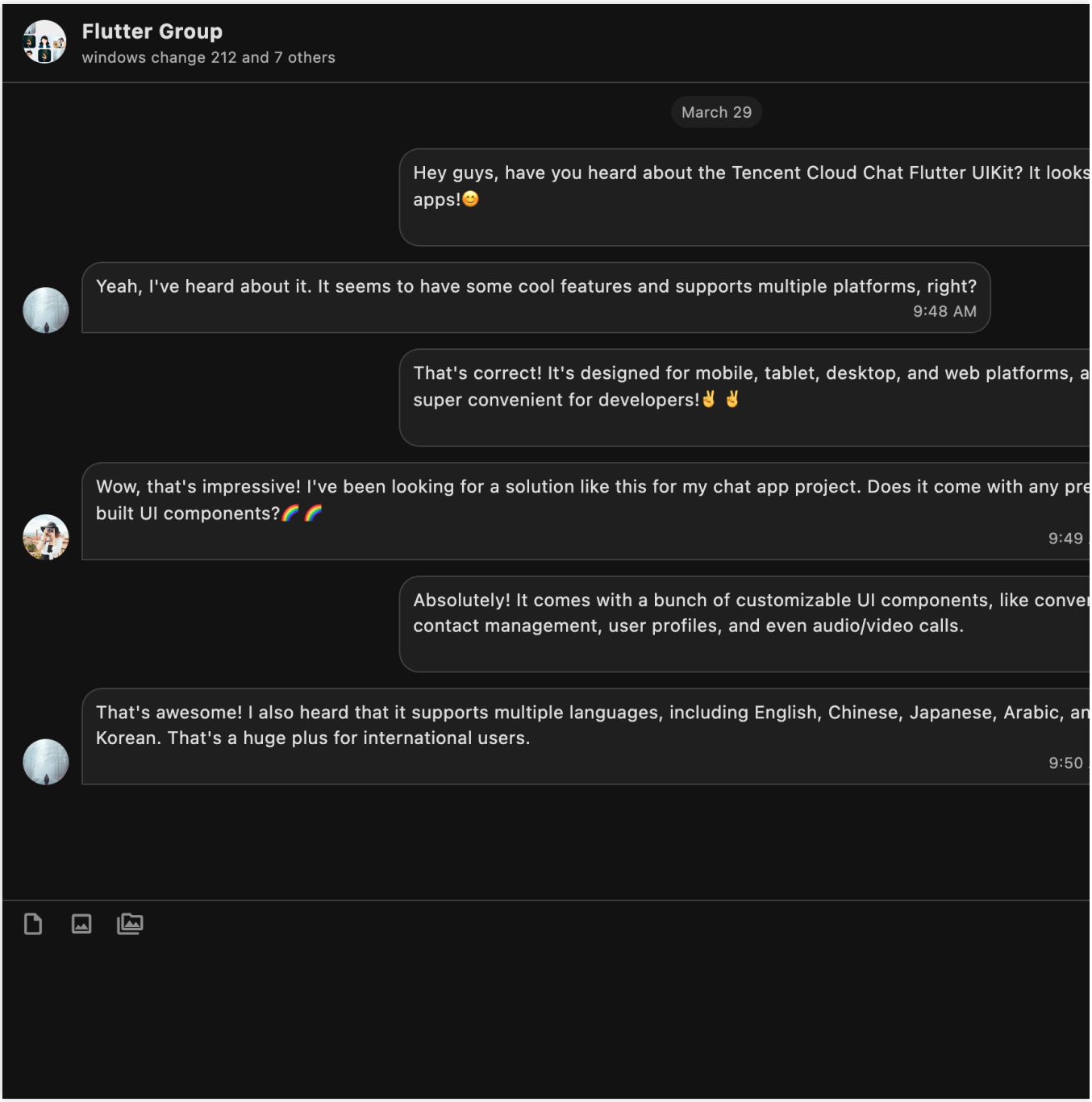

3:09
Runlin Wang
Offline
December 11
Hey Runlin, have you checked out the new Tencent Cloud Chat Flutter UIKit we just released?🌈🌈
3:03 PM
Hi Xingchen, yes, I just downloaded the preview version. It looks amazing!😎😎
3:04 PM
3:06 PM
I'm glad you like it! We've added some cool new features like dark mode and improved internationalization support.
3:07 PM
Yeah, I noticed that. The dark mode is really easy on the eyes, and the app feels more localized now.
3:08 PM
Message


3:09
95
Exactly! We've also improved the performance of the message list and added some nice touches like continuous voice message playback.
3:08 PM
Copy
Reply
Select
Forward
Delete

日间模式


Flutter Group
windows change 212 and 7 others
Japanese, Arabic, and Korean. That's a huge plus for international users.
9:50 A
Sounds perfect for my project. I'll definitely give it a try. Thanks for sharing❤️ ❤️ ❤️
9:51 AM
I just found their website, it's amazing! Check this out.
11:21 AM
11:22 AM
11:22 AM
And their introduction doc
11:24 AM
Tencent ... .pdf
6 KB
11:27 AM

黑暗模式



特点

1. **个性化外观**: UIKit内置深色和浅色模式，提供多种主题和外观定制选项，以满足您的业务需求。
2. **多平台兼容性**: 适应性强的单一代码库可确保跨各种平台的兼容性，包括**移动设备**（iOS/Android）、**平板电脑**（iPad和Android平板电脑）、**Web** 浏览器和**桌面软件**（Windows/macOS）。
3. **本地化支持:** 使用本地英语和其他语言选项开发，包括阿拉伯语、日语、韩语、简体中文和繁体中文。国际化功能确保了本地化的界面语言，并支持自定义和补充语言，其中阿拉伯语支持 **RTL** 用户界面。
4. **增强的性能:** UIKit提供了改进的消息列表性能、内存使用和精确的消息定位功能，以满足具有大量消息和导航到旧消息的情况。

版权所有：腾讯云计算（北京）有限责任公司

5. **高级功能:** UIKit拥有众多高级功能，包括连续语音消息播放、增强的多媒体和文件消息体验以及直观的左右滑动以预览多媒体消息。
6. **精致的用户体验:** 丰富的动画、触觉反馈和精美的界面等细节优化有助于改善用户体验。网格风格的头像、重新设计的转发面板、群组成员选择器和改进的长按消息菜单等新功能进一步丰富了体验。
7. **模块化设计:** 组件被组织成模块化包，允许选择性导入并减少不必要的膨胀。每个包都支持内置导航转换，通过自动处理转换（例如对话和消息之间的转换）来简化开发和集成。
8. **对开发人员友好的方法:** 更统一、标准化的组件参数设计，更清晰的代码命名约定和详细的注释，以及选择全局或实例级配置管理的灵活性，使开发更轻松、更高效。

兼容性

我们的 UIKit 支持**手机端**, **平板端** 和**桌面端** UI 样式，并兼容 Android、iOS、macOS、Windows 和 Web（将在未来版本中支持）。
它内置支持英语、简体中文、繁体中文、日语、韩语和阿拉伯语（支持阿拉伯RTL界面）以及亮色和暗色外观样式。

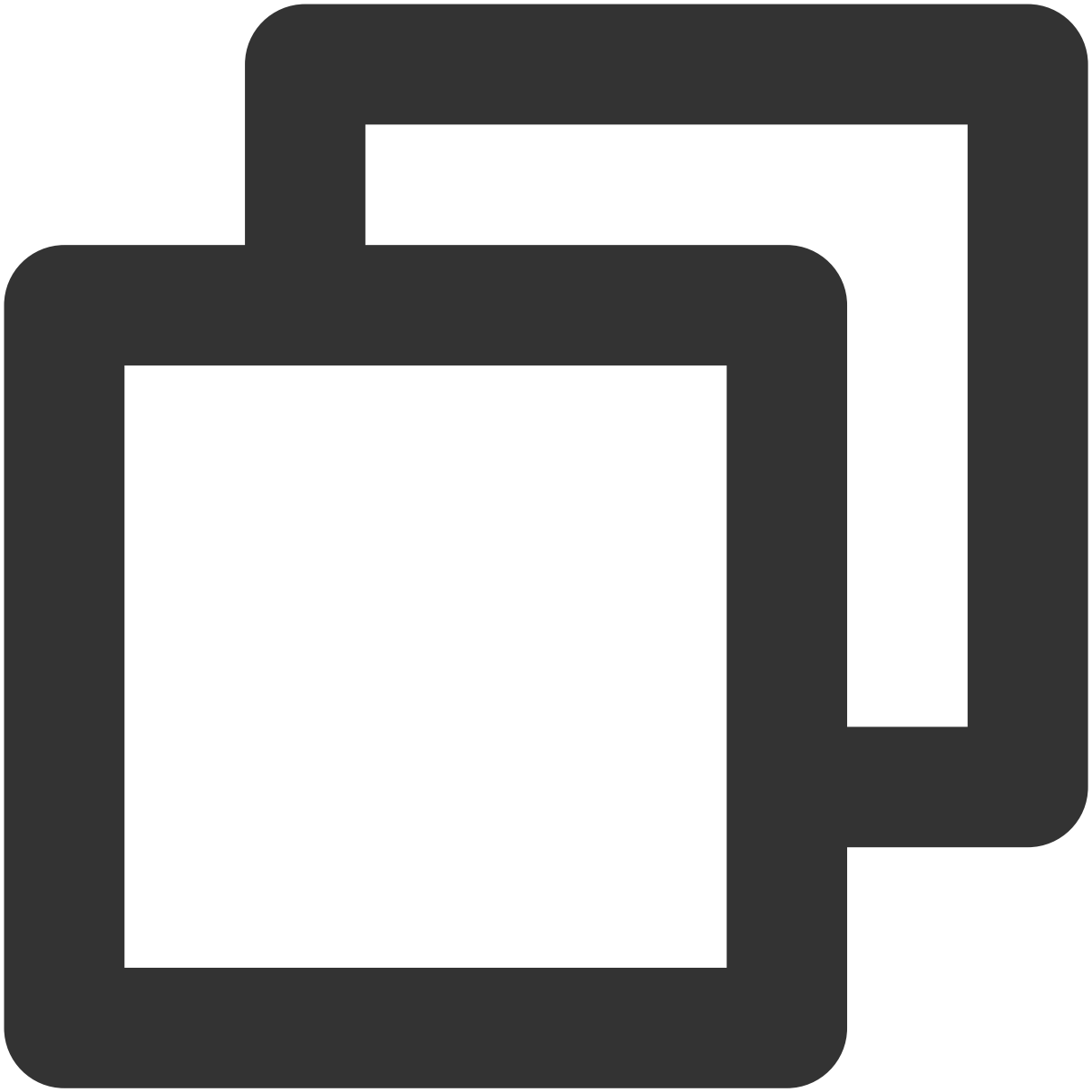
要求

Flutter 版本：3.16或更高。
Dart 版本：3.0或更高。

开始使用

引入包

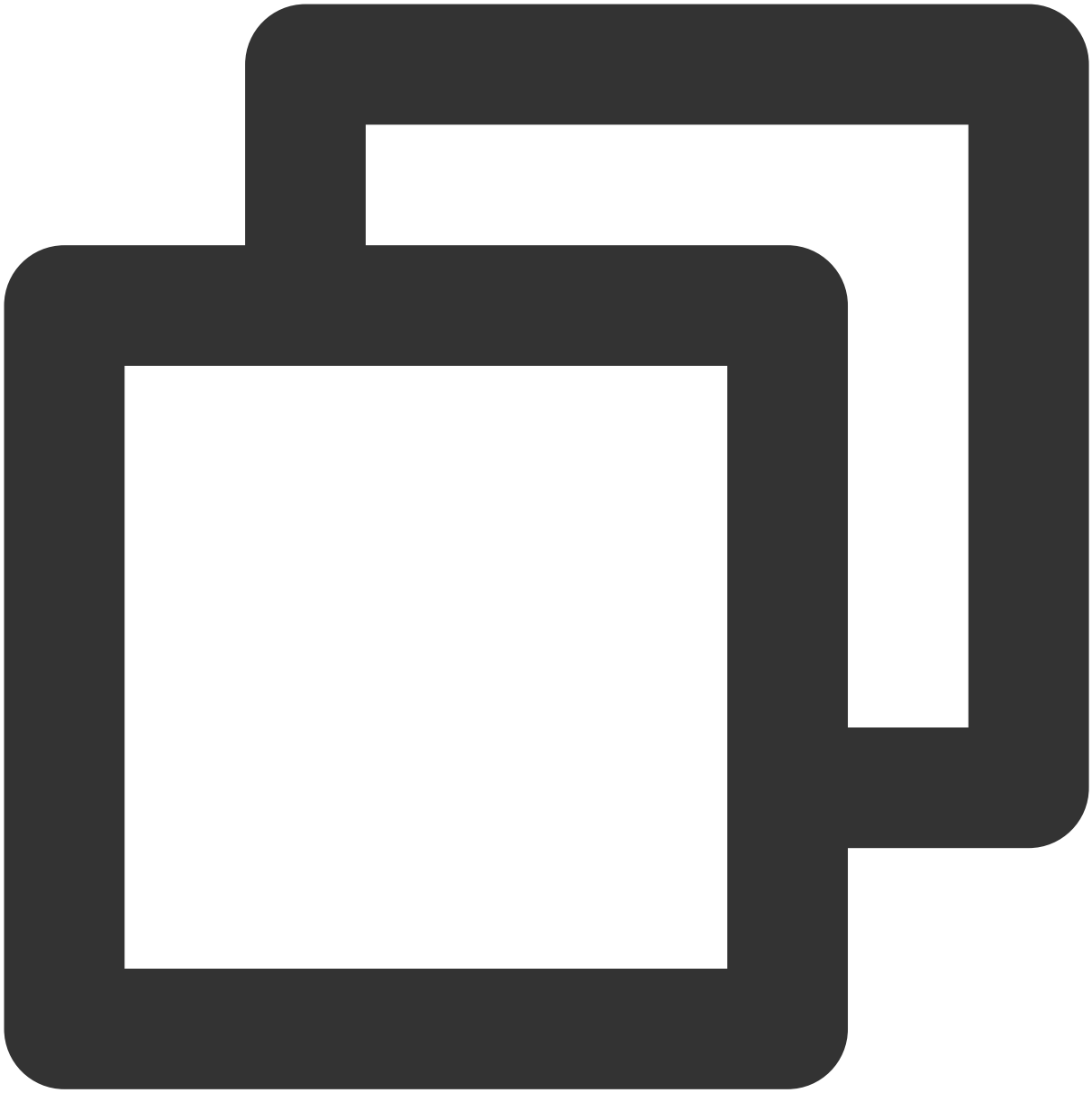
要开始使用我们的UIKit，首先导入基础包，tencent_cloud_chat。



```
flutter pub add tencent_cloud_chat
```

模块化组件包

然后，导入模块化的UI组件包为了 Message Chat, tencent_cloud_chat_message:



```
flutter pub add tencent_cloud_chat_message
```

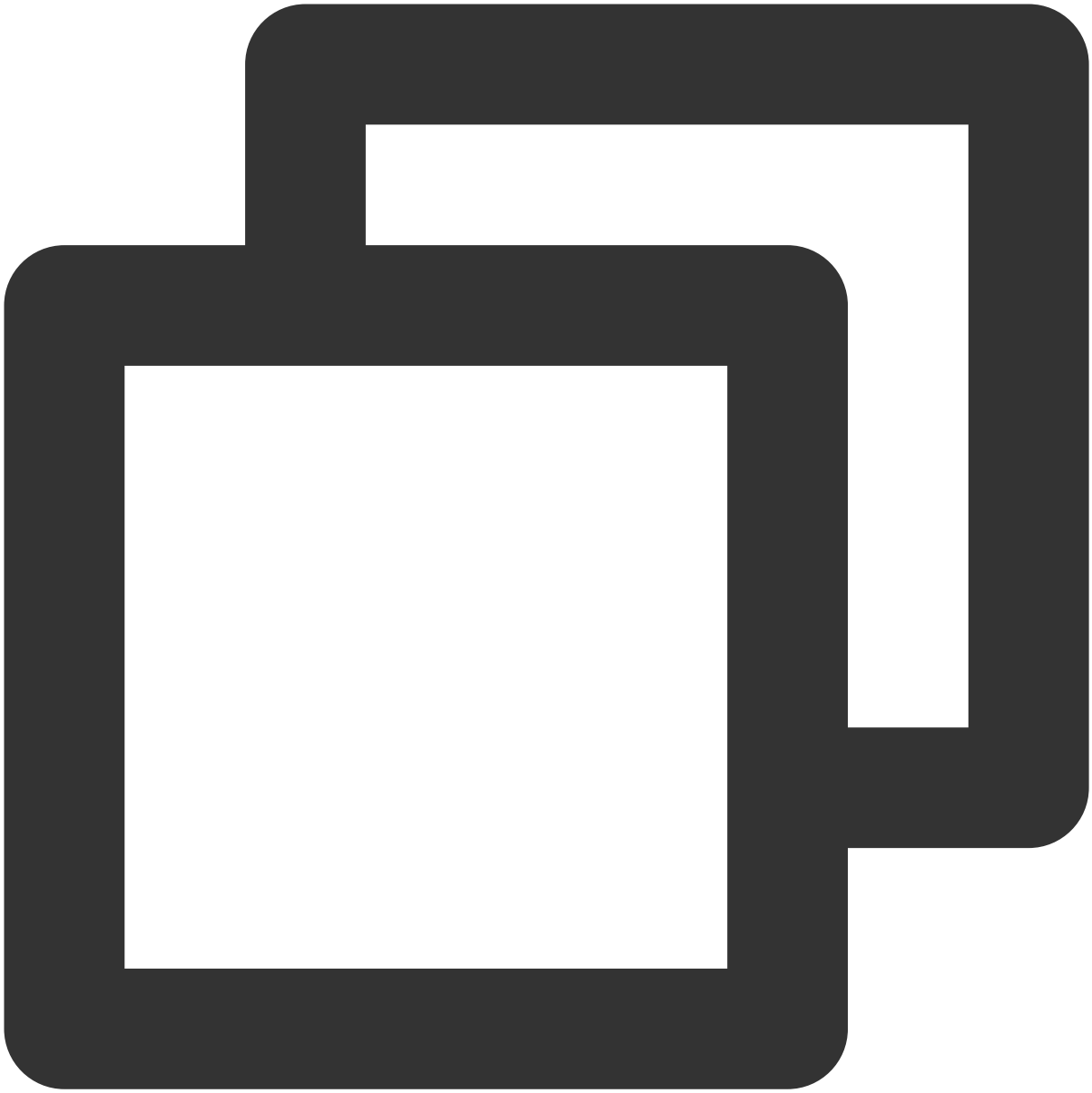
说明：

平台集成

在继续“基本用法”部分之前，请确保完成此处列出的其他平台集成步骤，特别是当您针对这些特定平台进行部署时。

Web / macOS: 如果您计划在 ***Web*** 或 ***macOS*** 平台上部署项目，请 参考此文档说明。

iOS: 打开 `ios/Podfile` ，并将最后一节替换为以下内容。



```
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['EXCLUDED_ARCHS[sdk=iphonesimulator*]'] = 'arm64'
          config.build_settings['ENABLE_BITCODE'] = 'NO'
          config.build_settings["ONLY_ACTIVE_ARCH"] = "NO"
        end
    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
```

Android / Windows: 不需要执行其他操作。

初始化 UIKit

在开始使用每个模块化包UI组件之前，您需要在项目中遵循一些初始设置步骤。

1. 准备必要的腾讯云IM 配置信息，如sdkappid、测试userID、userSig等。详情可参见：Demo 专区 > 快速跑通 > Flutter。

2. **安装 Package:**

在您的Flutter项目中，安装主包和上面的“开始使用”部分中提到的可选模块化包。

3. **全局配置:**

导入 `TencentCloudChatMaterialApp` ：将项目的 `MaterialApp` 替换为 `TencentCloudChatMaterialApp` 。这将自动管理和配置语言、主题（*带有material3*）、主题模式和其他设置，确保UIKit的界面参数与您的项目保持一致。

这一步将接管项目的语言、主题和主题模式配置。如果您不希望我们为您的项目自动管理所有这些配置，您可以按照**以下指南**在您的项目中手动导入必要的功能。

手动实现UIKit的全局配置

我们建议将您的项目的 `MaterialApp` 替换为 `TencentCloudChatMaterialApp` 。此推荐方法自动管理全局配置，包括本地化、主题和主题模式。

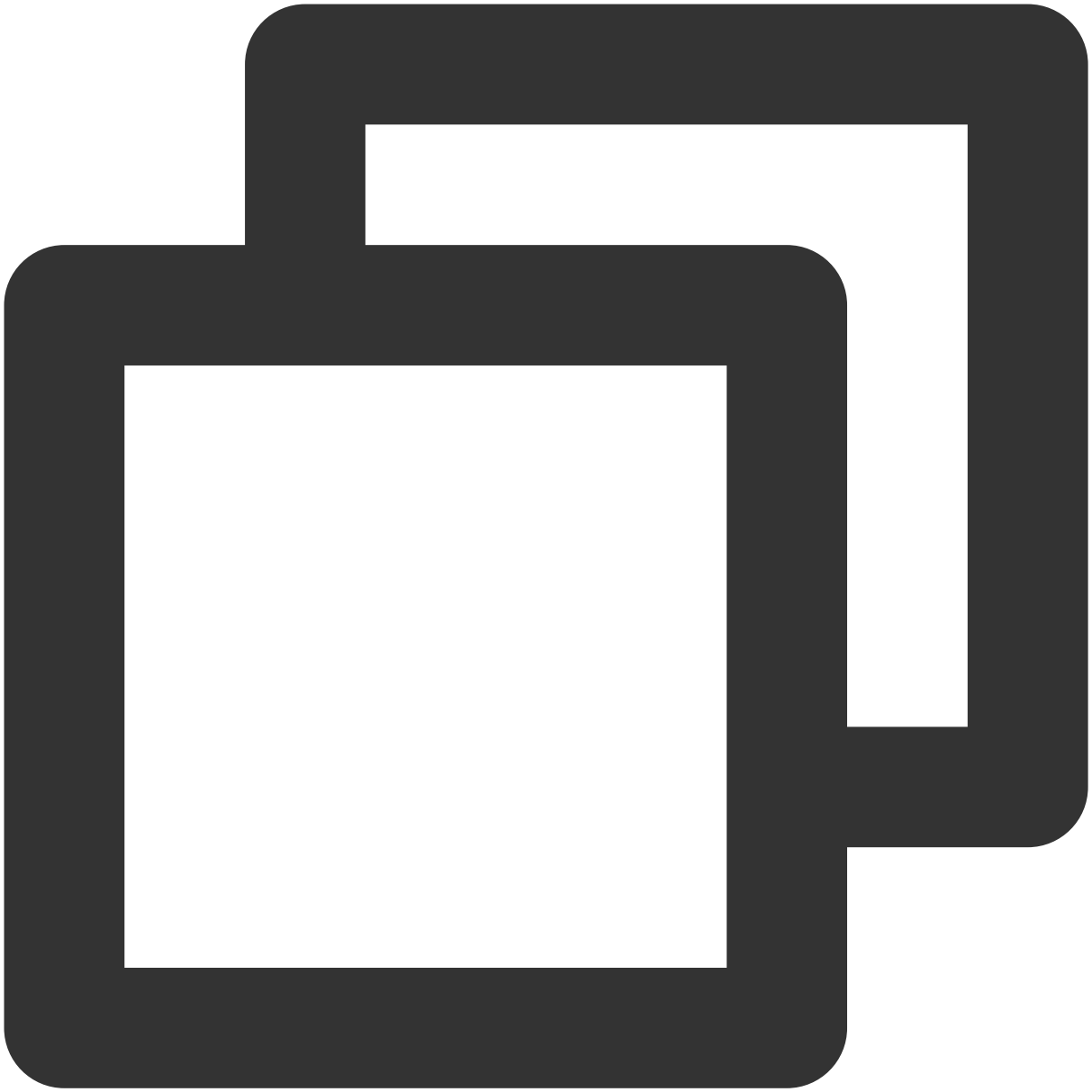
但是，如果由于大量自定义或使用其他包(如 `GET`)而希望保留项目的 `MaterialApp` ，则可以手动初始化UIKit。本指南将指导您完成该过程。

在全局配置中，本地化是必选的，而主题和主题模式设置是可选的。我们开始吧。

必要操作

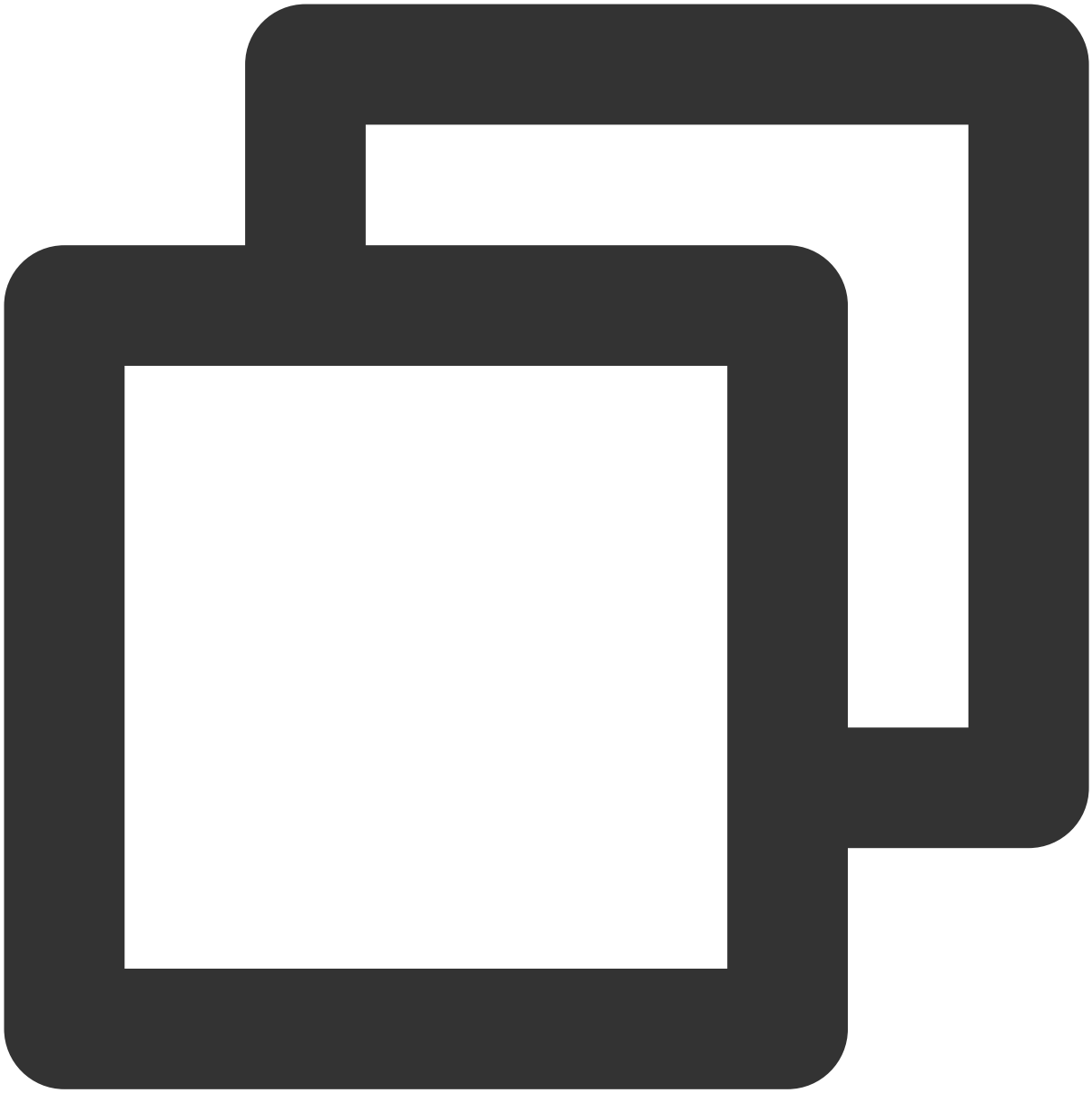
国际化语言

首先，将本地化工具导入到应用程序的入口文件中。



```
import 'package:tencent_cloud_chat_intl/localizations/tencent_cloud_chat_localizati
```

接下来，将本地化配置添加到 `MaterialApp` 或 `GetMaterialApp` 等第三方包提供的其他条目中。



```
MaterialApp(
  localizationsDelegates: const [
    /// Your configuration
    GlobalMaterialLocalizations.delegate,

    /// Add this line
    ...TencentCloudChatLocalizations.localizationsDelegates,  /// Add this line
  ],
  supportedLocales: [
    /// Your configuration
    ...S.delegate.supportedLocales,
```

```
    /// Add this line
    ...TencentCloudChatLocalizations.supportedLocales,
  ],
  /// ... Other configurations
)
```

此外，您可以根据您的业务逻辑设置语言区域设置 `locale` ，例如在应用启动时记录用户指定的语言，而不是遵循系统设置。此配置将同时应用于您的项目和聊天UIKit。

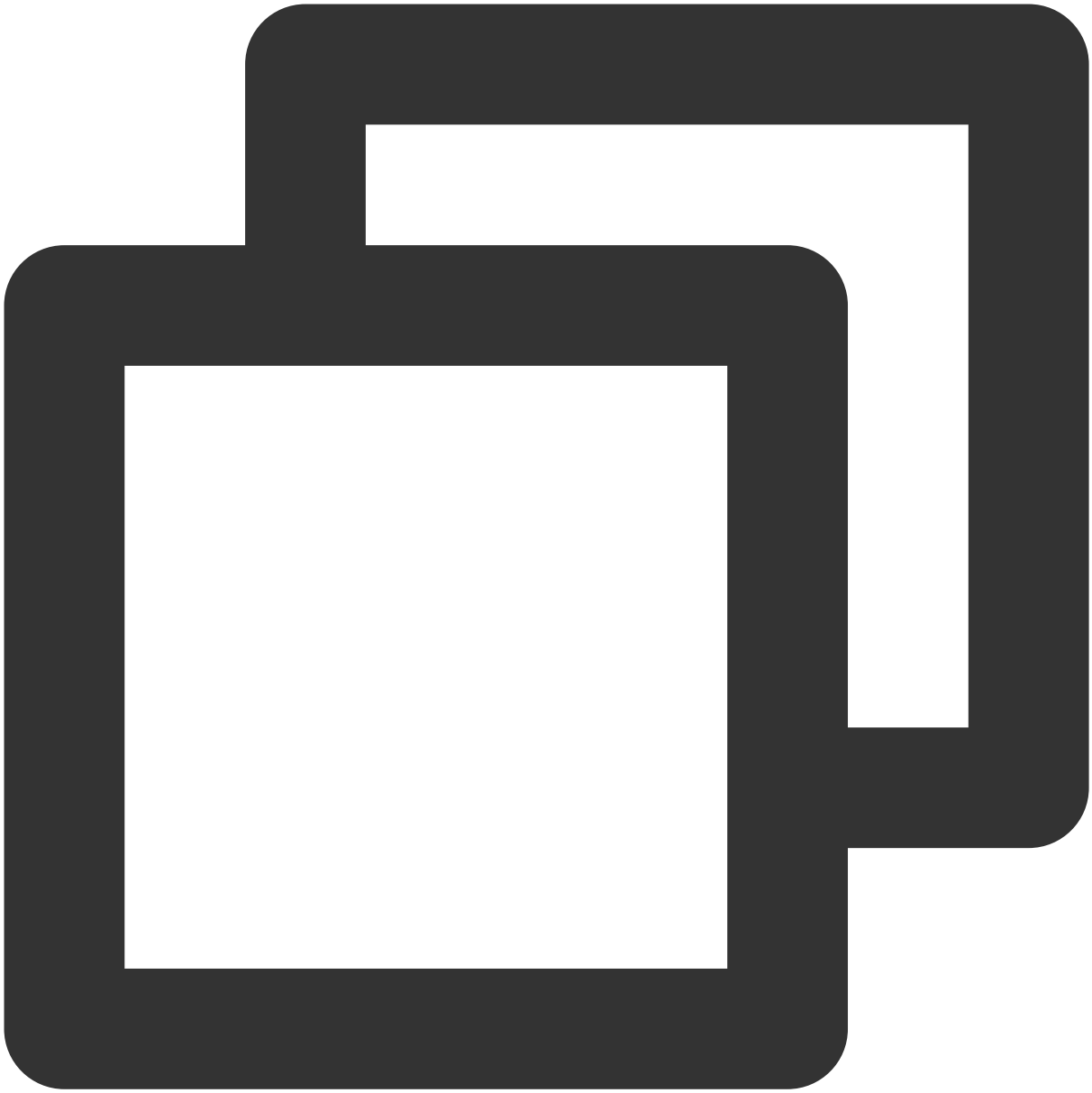
有关本地化定制的更多信息，包括添加或删除语言、添加本地化条目和修改翻译单词，请参考此指南。

可选操作

Theme / Theme Mode

UIKit 的主题数据由 `TencentCloudChatTheme` 类定义，通过 `TencentCloudChat.dataInstance.Theme` 全局维护和管理。

这允许您从任何位置访问主题：

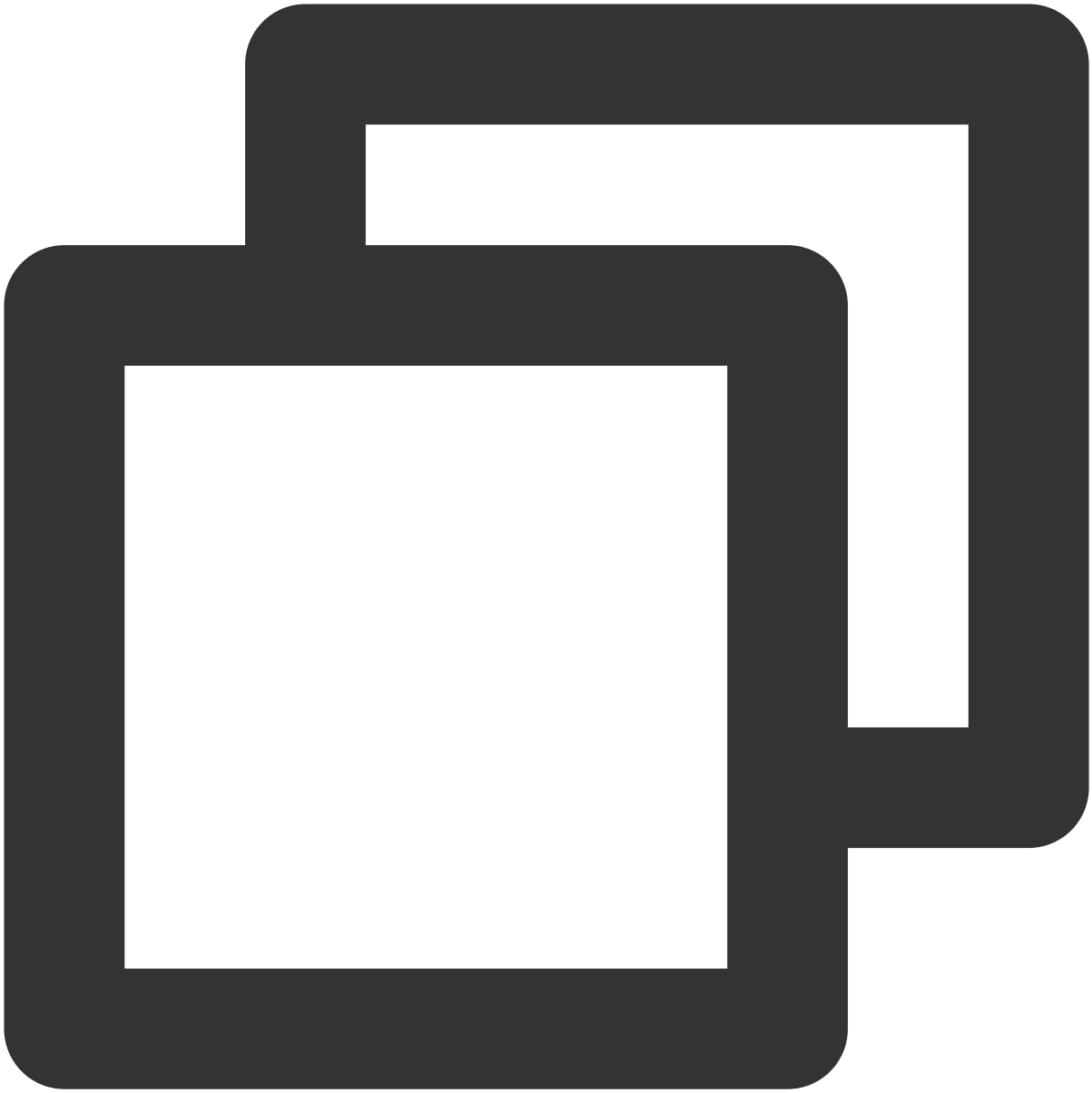


```
TencentCloudChatTheme theme = TencentCloudChat.dataInstance.theme;
```

此主题实例包括一个主题模型（包括亮色和暗色模式的主题数据）和亮度（亮色和暗色模式状态）。

此外，您可以通过我们为亮色和暗色模式提供的 Material 3 样式主题数据，从 `MaterialApp` 指定 `theme` 和 `darkTheme` 。您还可以根据我们维护的亮度状态设置 `themeMode` 状态。这确保了您的应用程序和我们的 Chat UIKit 在外观上保持一致，提高了用户体验。（您可以按照下面的描述自定义此主题样式。）

为实现此目的，我们建议将您的入口小部件（托管 `MaterialApp` 的小部件）转换为 `StatefulWidget` 。将 `TencentCloudChatTheme` 主题作为状态添加，并监听 `Stream<TencentCloudChatTheme>?` `themeDataListener` 以更新其值并根据动态、可自定义的主题数据构建应用程序。以下是一个示例代码：



```
// Theme instance for the Chat UIKit
TencentCloudChatTheme theme = TencentCloudChat.dataInstance.theme;

// Listener for theme data changes
Stream<TencentCloudChatTheme>? themeDataListener = TencentCloudChat.eventBusInstanc

// Callback for handling theme data changes
void _themeDataChangeCallback(TencentCloudChatTheme themeData) {
  setState(() {
    theme = themeData;
  });
```

版权所有：腾讯云计算（北京）有限责任公司

```
}

// Adds a listener for theme data changes
void _addThemeDataChangeListener() {
  themeDataListener?.listen(
    _themeDataChangeCallback,
  );
}

@override
void initState() {
  super.initState();
  _addThemeDataChangeListener();
}

// .......

return MaterialApp(
  themeMode: theme.brightness != null ? (theme.brightness == Brightness.light ? The
  theme: theme.getThemeData(brightness: Brightness.light),
  darkTheme: theme.getThemeData(brightness: Brightness.dark),
   /// ... Other configurations
);
```

要自定义 Chat UIKit 的外观主题和全局主题（如果如上所示在 `MaterialApp` 中指定），请使用 `TencentCloudChatCoreController.setThemeColors` 方法为亮色和暗色模式指定外观颜色。有关具体使用说明，请参阅代码中的注释。

要切换主题模式（亮度），请使用 `TencentCloudChatCoreController.setBrightnessMode` 或 `TencentCloudChatCoreController.toggleBrightnessMode`。有关具体使用说明，请参阅代码中的注释。

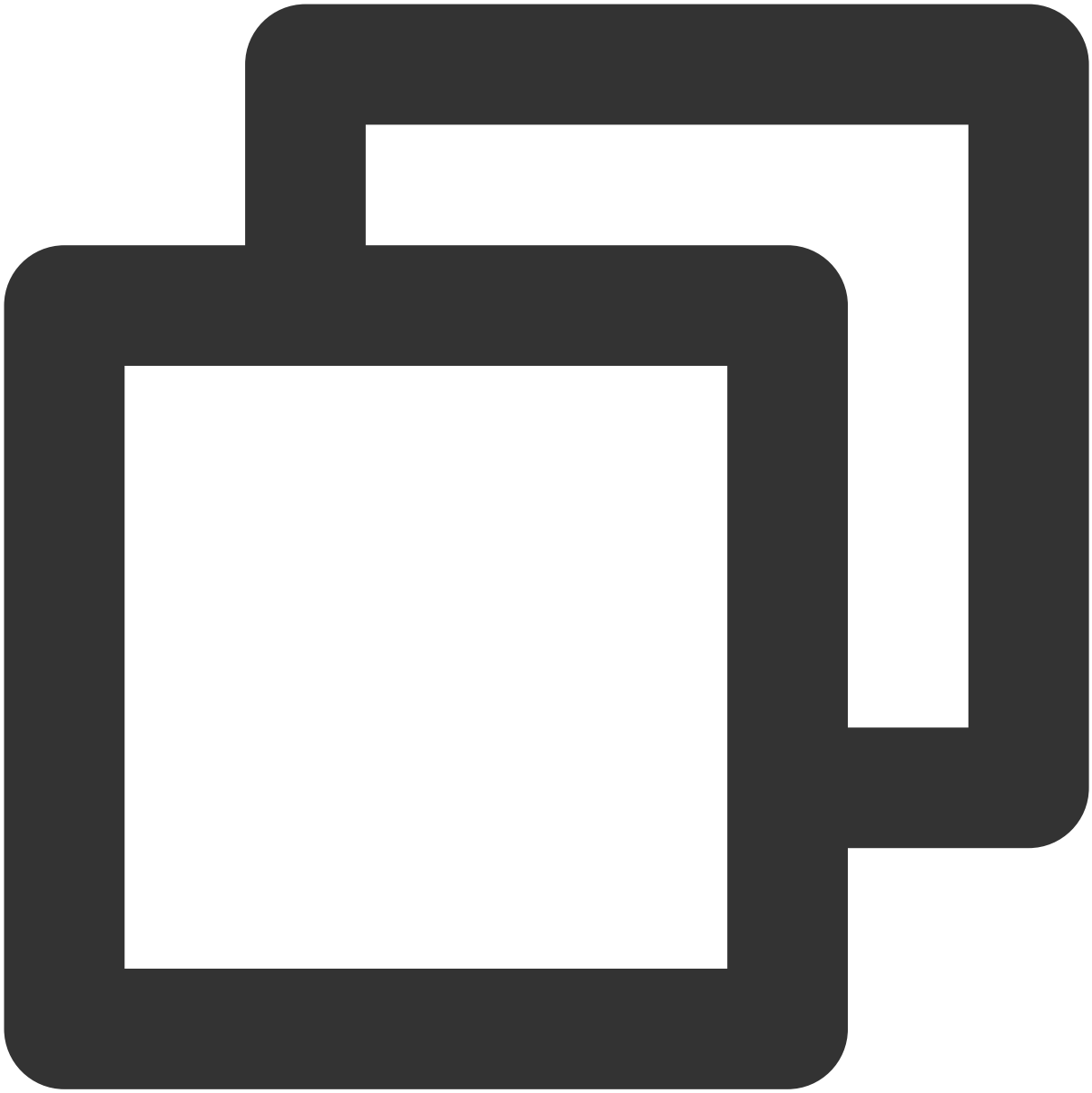
4. 初始化和登录：

调用 `TencentCloudChat.controller.initUIKit` 方法进行初始化和登录。调用说明和参考代码如下：

说明：

我们高度建议配置 `callbacks` 以可定制的方式通过 `Dialog` 或 `ToolTip` 高效地处理 SDK API 错误和需要用户关注的特定 UIKit 事件。

有关详细说明、用法和事件代码列表，请参阅本指南。



```
await TencentCloudChat.controller.initUIKit(
  config: TencentCloudChatConfig(), /// [可选]: 影响整个聊天界面的全局配置，包括用户相关配置
  options: TencentCloudChatInitOptions(
    sdkAppID: , /// [必需]: 腾讯云聊天应用的SDKAppID
    userID: , /// [必需]: 已登录用户的userID
    userSig: , /// [必需]: 已登录用户的userSig
  ),

  components: TencentCloudChatInitComponentsRelated( /// [必需]: 模块化UI组件相关设置，
    usedComponentsRegister: [
      /// [必需]: 聊天界面中使用的组件的注册函数列表。
```

```
      /// 只需要使用 TencentCloudChatMessage 中的 `register` 即可。
      TencentCloudChatMessageManager.register,
    ],
    componentConfigs: TencentCloudChatComponentConfigs(
      /// [可选]: 在此处为每个UI模块组件提供自定义配置。这些构建器将全局应用。
    ),
    componentBuilders: TencentCloudChatComponentBuilders(
      /// [可选]: 在此处为每个UI模块组件提供自定义UI构建器。这些构建器将全局应用。
    ),
    componentEventHandlers: TencentCloudChatComponentEventHandlers(
      /// [可选]: 在此处为UI组件相关事件提供自定义事件处理程序。这些构建器将全局应用。
    ),
  ),

  /// [关键]: 强烈建议将以下回调侦听器集成到SDK事件、SDK API错误和需要用户关注的特定UIKit事件的
  /// 有关详细用法，请参阅本自述文件末尾的“引入UIKit回调”部分。
  callbacks: TencentCloudChatCallbacks(
    onTencentCloudChatSDKEvent: V2TimSDKListener(),  /// [可选]: 处理SDK事件，如onKick
    onTencentCloudChatSDKFailCallback: (apiName, code, desc) {}, /// [可选]: 处理SDK
    onTencentCloudChatUIKitUserNotificationEvent: (TencentCloudChatComponentsEnum c
  ),

  plugins: [],  /// [可选]: 使用的插件，如tencent_cloud_chat_robot等。具体用法请参阅每个插
);
```

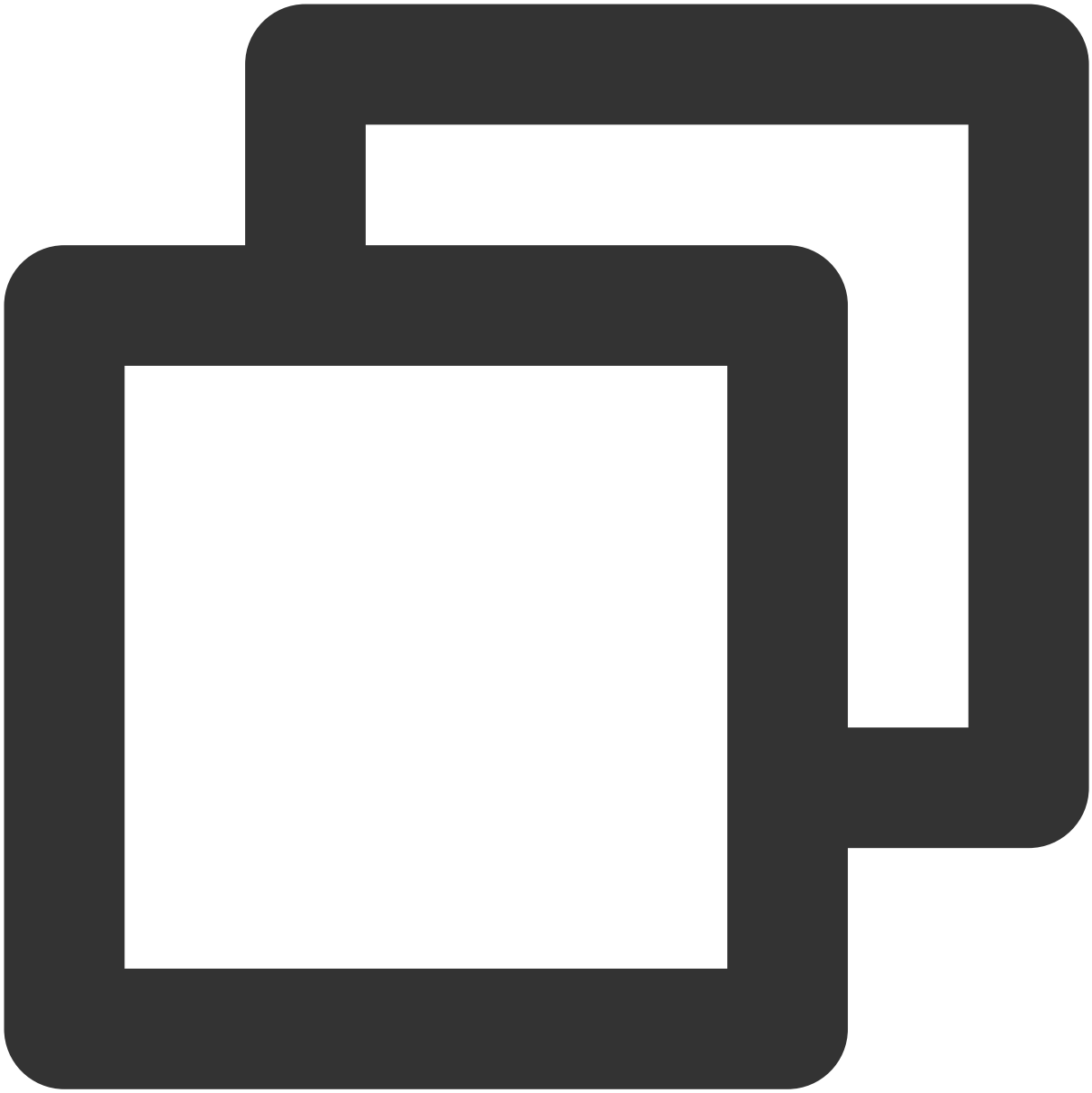
发起聊天

Chat UIKit 为用户创建聊天模块提供了全面的解决方案。

通过指定聊天用户选项定向到 `TencentCloudChatMessage` ，即可为其无缝搭建聊天模块，同时满足一对一和群聊的需求。

自动导航到消息组件

利用 Chat UIKit 自带的自动导航功能，通过 `TencentCloudChatMessageOptions` 调用 `navigateToMessage` 方法即可轻松发起聊天，如下所示：



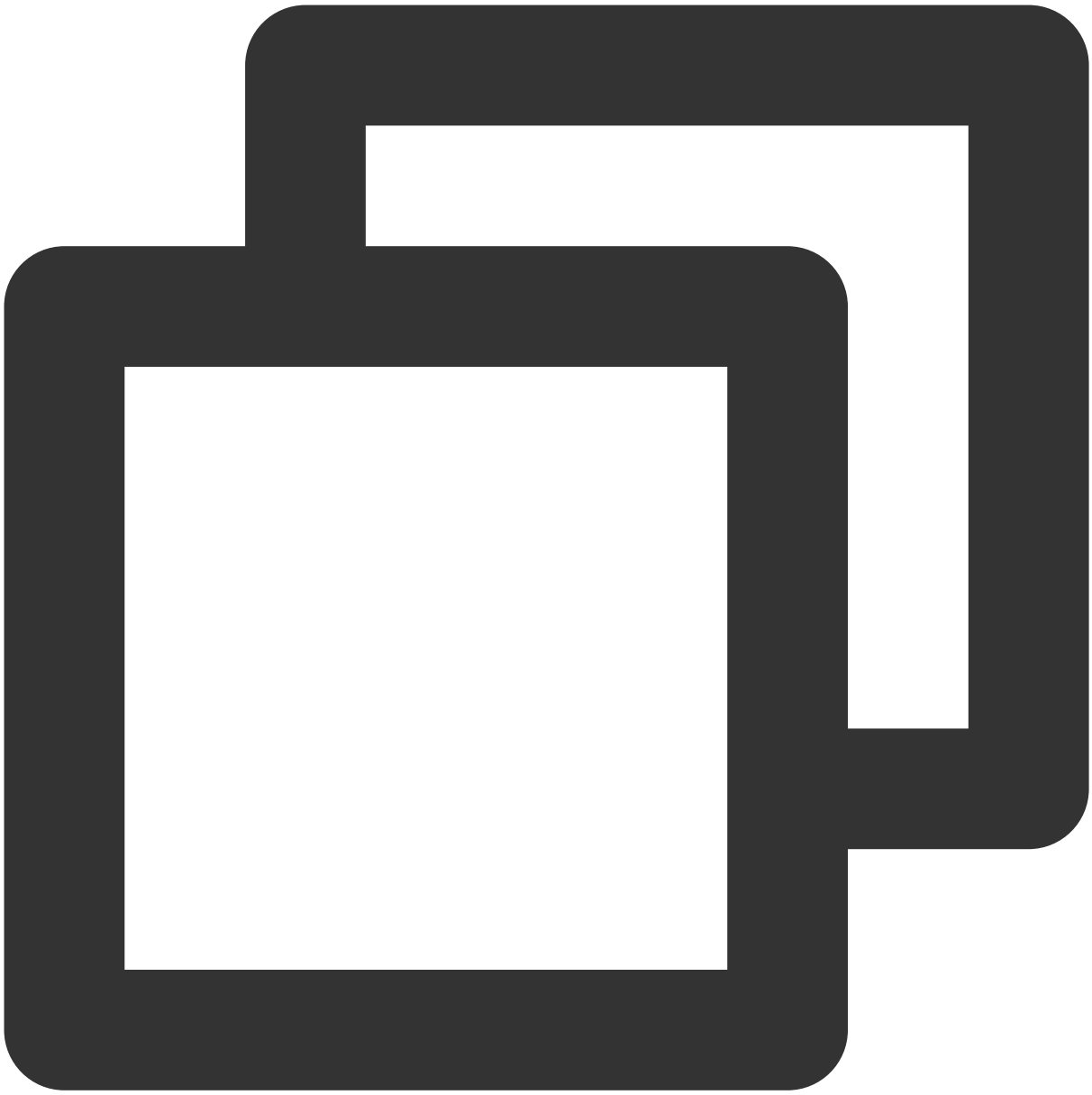
```
final messageOptions = TencentCloudChatMessageOptions(
      // 提供UserID或GroupID，指示聊天对话。
      userID: "", // 对于一对一聊天，提供另一个用户的用户 ID
      groupID: "", // 对于群聊，请提供 groupID
    );
/// 使用上面构造的 messageOptions
navigateToMessage(context: context, options: messageOptions);
```

通过提供用户 ID 或 groupID，可以轻松地发起一对一或群聊的聊天对话。

手动导航到消息组件

版权所有：腾讯云计算（北京）有限责任公司

如果你需要手动处理导航，或将组件包装在你的自定义页面中，那么要实例化一个 `TencentCloudChatMessage` 组件。



```
final messageOptions = TencentCloudChatMessageOptions(
      // 提供UserID或GroupID，指示聊天对话。
      userID: "", // 对于一对一聊天，提供另一个用户的用户 ID
      groupID: "", // 对于群聊，请提供 groupID
    );

final Widget message = TencentCloudChatMessage(
      options: messageOptions,
```

```
      // ... 其他参数，如 builders，可以根据您的需求在全局范围内指定或在此处静态传入。有关详细用
    );
```

你可以将这个实例化的组件放在单独页面的 `build` 方法中，或像使用 `Navigator.push` 一样直接用于导航。

高级用法

一旦你实现了基本的使用步骤，你的项目中就会有一个带有默认用户界面和业务逻辑的聊天消息模块。然而，如果这些默认设置不能完全满足你的业务需求，有几种方式可以自定义模块：

Controller：使用控制器管理消息小部件。这可能涉及到根据需要发送额外的消息，滚动消息列表等操作。

Config：使用 config 调整基本设置。

Builders：使用 builder 进一步自定义 UI 部件。每个构建器都配备了数据（构建自定义部件所需的基本参数）、方法（业务逻辑相关方法）和部件（每个构建器的默认原子化部件）。

EventHandlers：附加监听器到 `eventHandlers` 来管理特定组件的事件。这包括 `uiEventHandlers`（如各种 onTap 类似的事件）和 `lifeCycleEventHandlers`（如发送消息后触发的事件）。

这些高级实现方法在所有 Chat UIKit 组件中都保持一致。要深入了解这些高级特性，你可以参考**模块化 UI 包的高级用法**。

总结

以上步骤提供了一个快速指南，介绍了如何单独集成消息聊天组件。如果你希望了解聊天 UIKit 的完整使用方法，或者有任何未解决的问题，请参阅**完整版 UIKit 快速接入文档**。

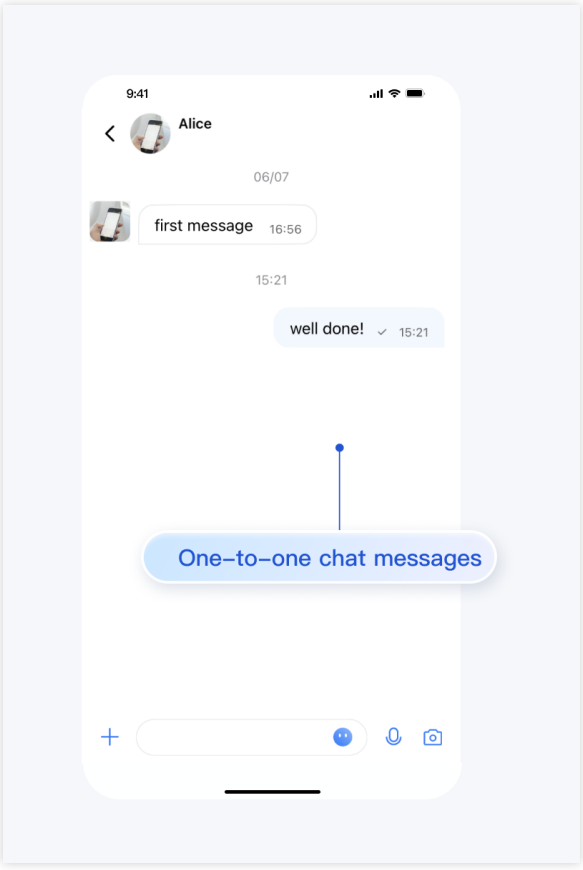
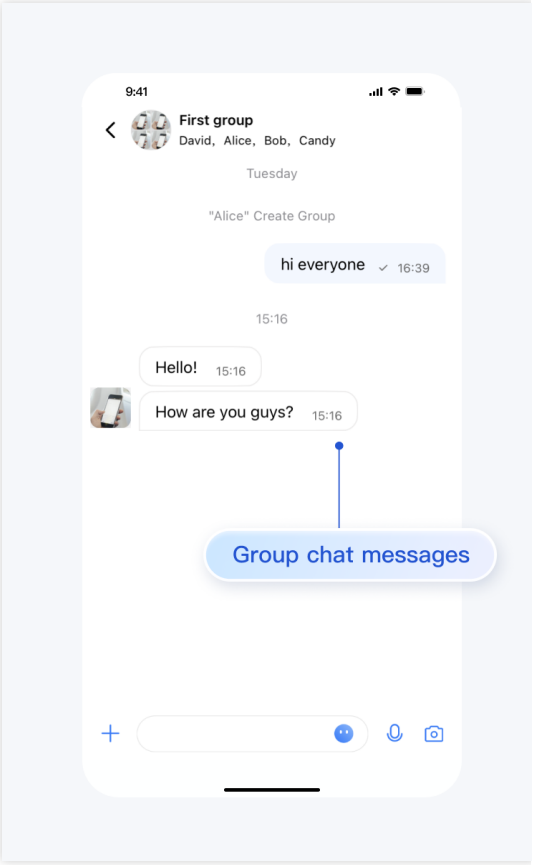
构建基础界面
聊天界面
Android

最近更新时间：2024-08-14 10:34:03

本文会引导您构建聊天界面。

效果展示

聊天界面发送消息效果如下所示：

单聊界面	群聊界面
	

开发环境要求

Android Studio-Giraffe
Gradle-7.2
Android Gradle Plugin Version-7.0.0
kotlin-gradle-plugin-1.5.31

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIChat` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

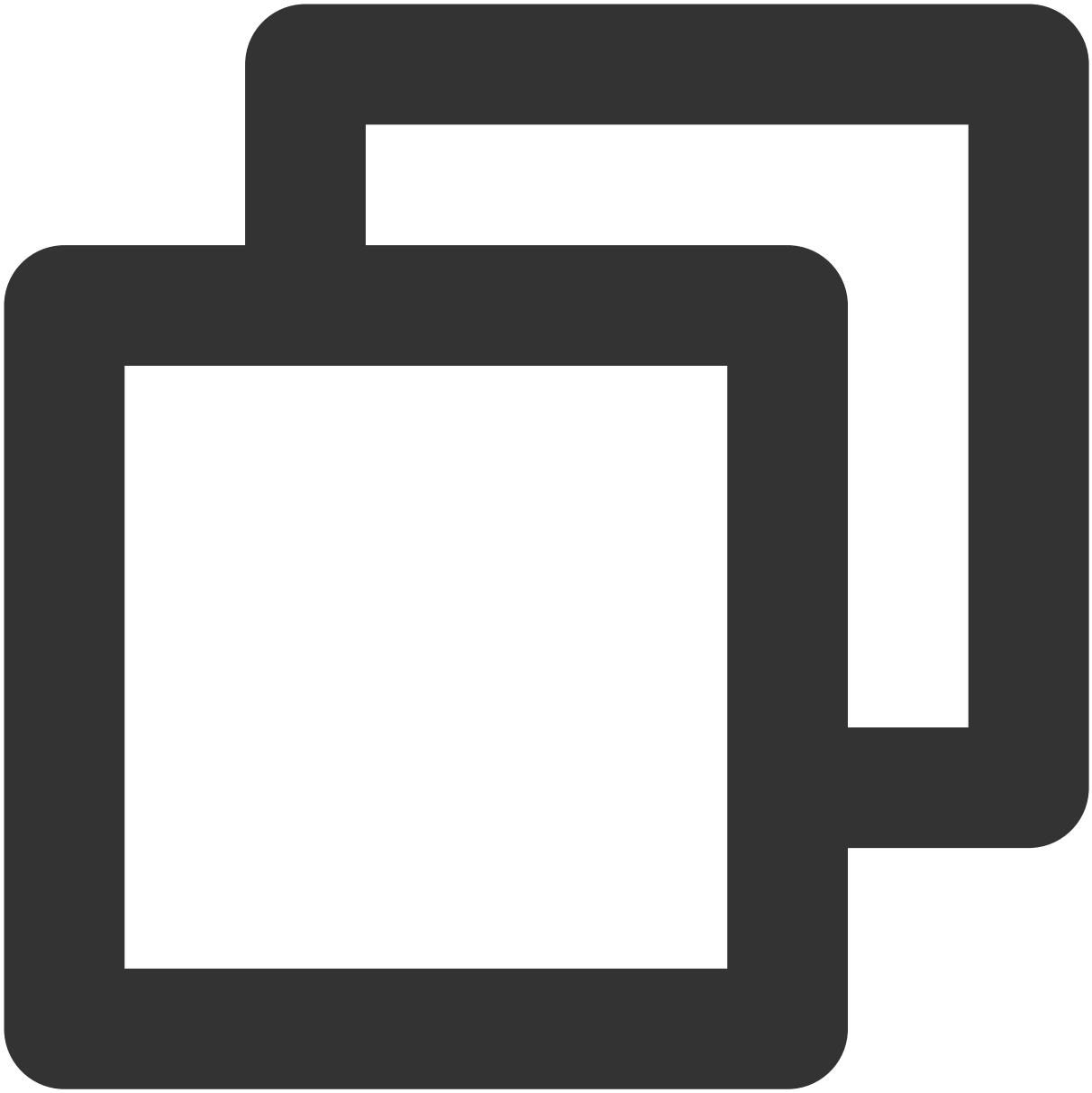
注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上4步，请先参见 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

步骤说明

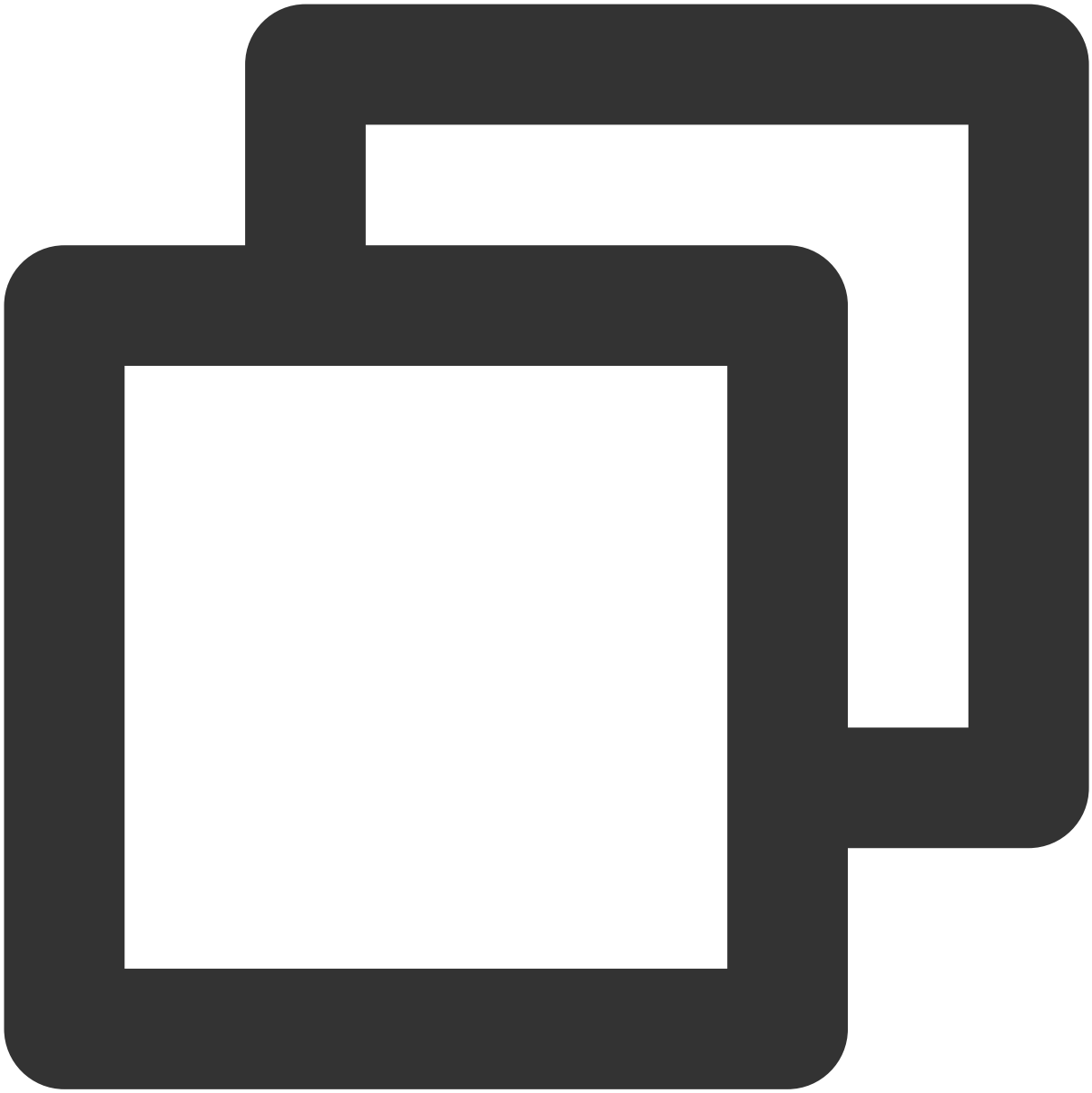
如果您想跳转到单聊消息界面，可以直接参考 快速开始，本文不再赘述。
如果跳转群聊界面，需要传入有效 groupID。这里的前提是您有一个已经存在群组的 groupID。有两种简便方式可获取：
1. 去控制台创建一个 group，操作路径：**Applications > Your App > Chat > Groups > Group Management > Add Group**。创建成功后，您可以直接在当前页看到 groupID。
2. 按照文档 创建群组 的指引，手动在 TUIKit 里创建一个群组，群组详情页中会展示 groupID。

跳转群聊界面示例代码如下所示：
简约版
经典版



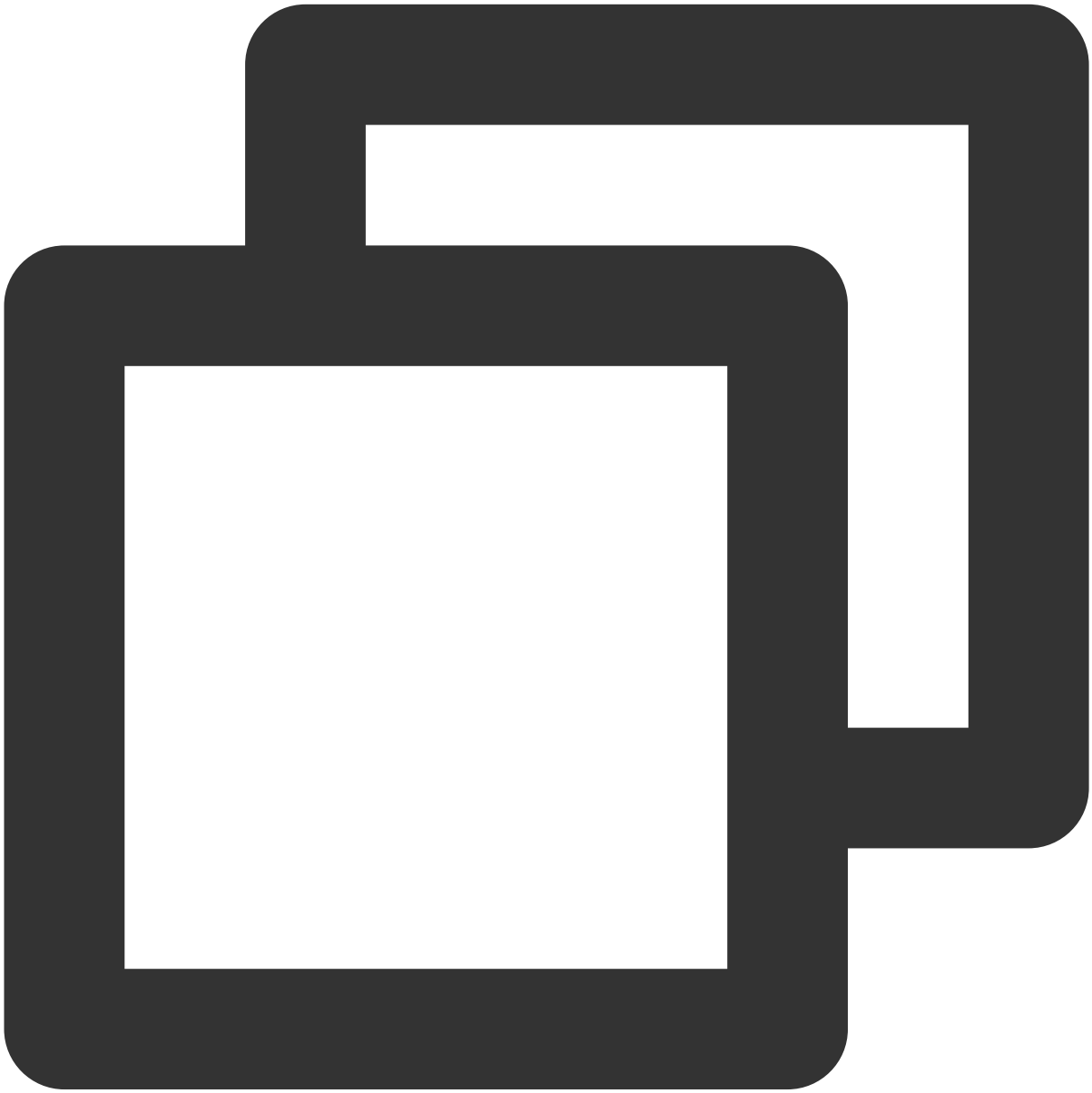
```
Intent intent;
if (isGroup) {
    intent = new Intent(this, TUIGroupChatMinimalistActivity.class);
} else {
    intent = new Intent(this, TUIC2CChatMinimalistActivity.class);
}
// If it's a C2C chat, chatID is the other person's UserID; if it's a Group chat, c
intent.putExtra(TUIConstants.TUIChat.CHAT_ID, "chatID");
intent.putExtra(TUIConstants.TUIChat.CHAT_TYPE, isGroup ? V2TIMConversation.V2TIM_G
startActivity(intent);
```

版权所有：腾讯云计算（北京）有限责任公司



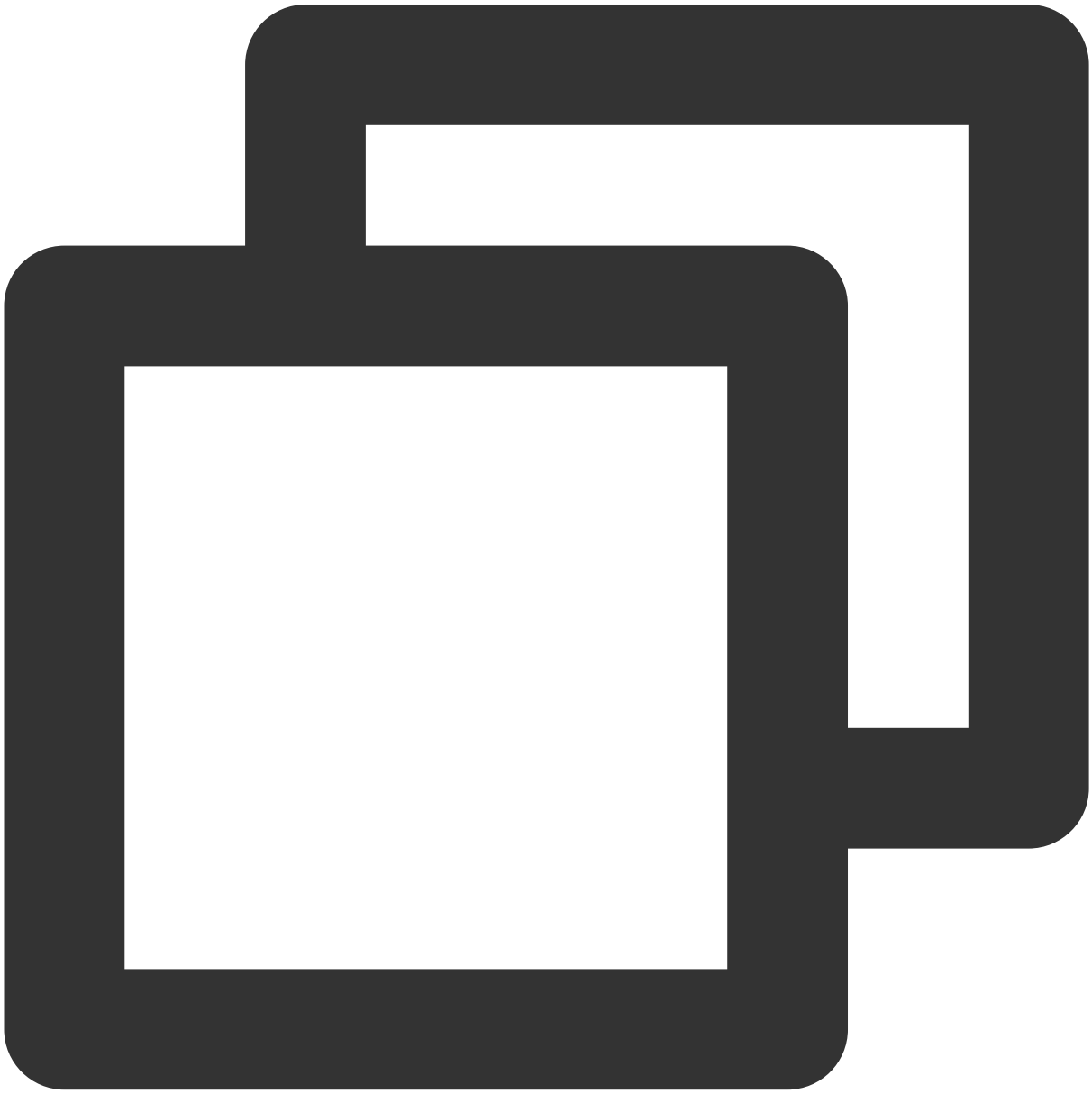
```
Intent intent;
if (isGroup) {
    intent = new Intent(this, TUIGroupChatActivity.class);
} else {
    intent = new Intent(this, TUIC2CChatActivity.class);
}
// If it's a C2C chat, chatID is the other person's UserID; if it's a Group chat, c
intent.putExtra(TUIConstants.TUIChat.CHAT_ID, "chatID");
intent.putExtra(TUIConstants.TUIChat.CHAT_TYPE, isGroup ? V2TIMConversation.V2TIM_G
startActivity(intent);
```

您也可以将 TUIChat 聊天界面，嵌入到自己的 Activity 中。
示例代码如下所示：
简约版
经典版



```
Fragment fragment;
// If it's a C2C chat, chatID is the other person's UserID; if it's a Group chat, c
if (isGroup) {
    GroupChatInfo groupChatInfo = new GroupChatInfo();
    groupChatInfo.setId(chatID);
    TUIGroupChatMinimalistFragment tuiGroupChatFragment = new TUIGroupChatMinimalis
```

```
    tuiGroupChatFragment.setChatInfo(groupChatInfo);
    fragment = tuiGroupChatFragment;
} else {
    C2CChatInfo c2cChatInfo = new C2CChatInfo();
    c2cChatInfo.setId(chatID);
    TUIC2CChatMinimalistFragment tuic2CChatFragment = new TUIC2CChatMinimalistFragm
    tuic2CChatFragment.setChatInfo(c2cChatInfo);
    fragment = tuic2CChatFragment;
}
getSupportFragmentManager().beginTransaction()
        .add(R.id.chat_fragment_container, fragment).commitAllowingStateLoss()
```



```
Fragment fragment;
// If it's a C2C chat, chatID is the other person's UserID; if it's a Group chat, c
if (isGroup) {
    GroupChatInfo groupChatInfo = new GroupChatInfo();
    groupChatInfo.setId(chatID);
    TUIGroupChatFragment tuiGroupChatFragment = new TUIGroupChatFragment();
    tuiGroupChatFragment.setChatInfo(groupChatInfo);
    fragment = tuiGroupChatFragment;
} else {
    C2CChatInfo c2cChatInfo = new C2CChatInfo();
    c2cChatInfo.setId(chatID);
    TUIC2CChatFragment tuic2CChatFragment = new TUIC2CChatFragment();
    tuic2CChatFragment.setChatInfo(c2cChatInfo);
    fragment = tuic2CChatFragment;
}
getSupportFragmentManager().beginTransaction()
        .add(R.id.chat_fragment_container, fragment).commitAllowingStateLoss();
```

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

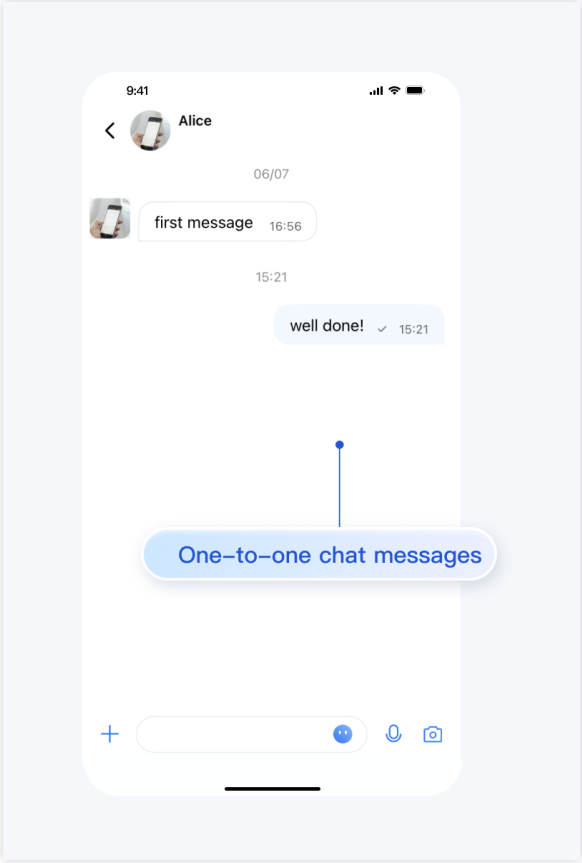
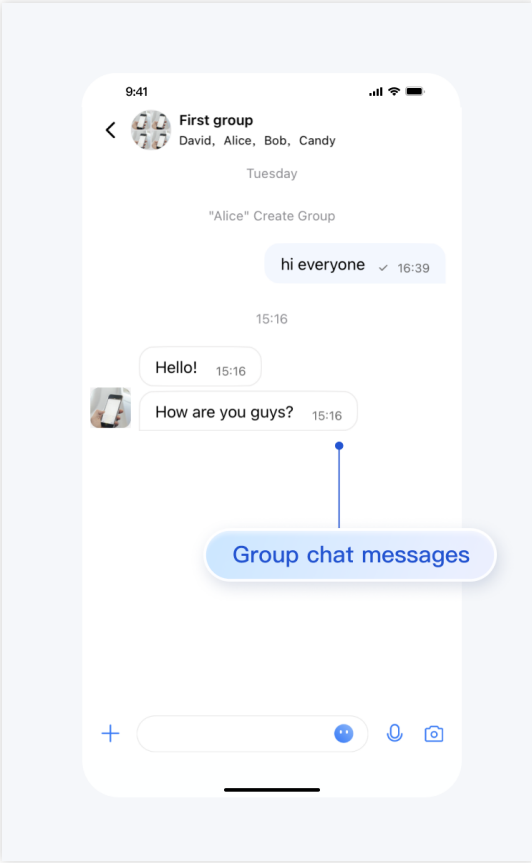
iOS

最近更新时间：2024-06-24 16:53:17

本文会引导您构建聊天界面。

效果展示

聊天界面发送消息效果如下所示：

单聊界面	群聊界面
	

开发环境要求

Xcode 10 及以上

iOS 9.0 及以上

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIChat` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

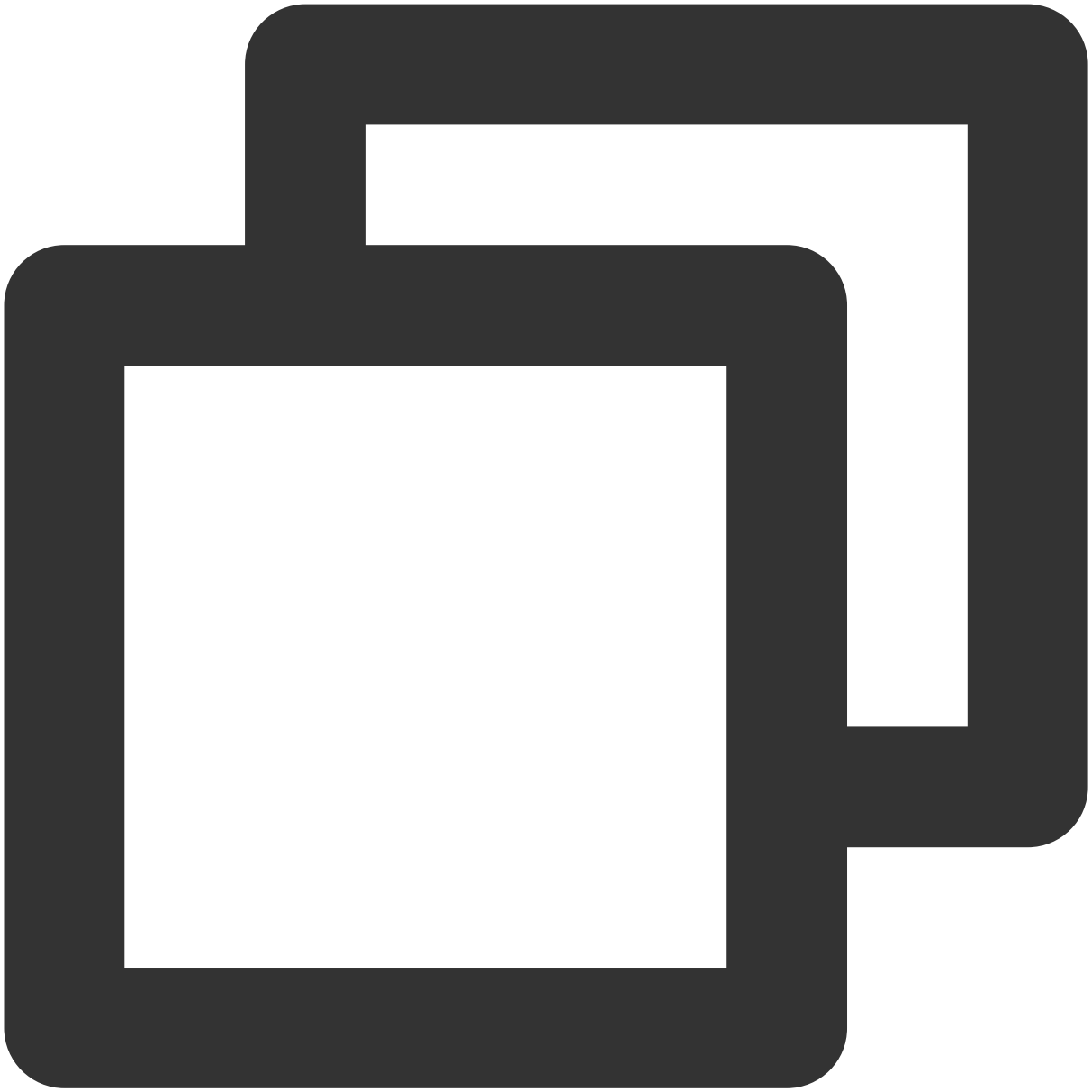
注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

步骤说明

如果您想跳转到单聊消息界面，可以直接参考 快速开始，本文不再赘述。
如果跳转群聊界面，需要传入有效 groupID。这里的前提是您有一个已经存在群组的 groupID。有两种简便方式可获取：
1. 去控制台创建一个 group，操作路径：Applications > Your App > Chat > Groups > Group Management > Add Group。创建成功后，您可以直接在当前页看到 groupID。
2. 按照文档 创建群聊 的指引，手动在 TUIKit 里创建一个群组，群组详情页中会展示 groupID。

跳转群聊界面示例代码如下所示：
简约版
经典版



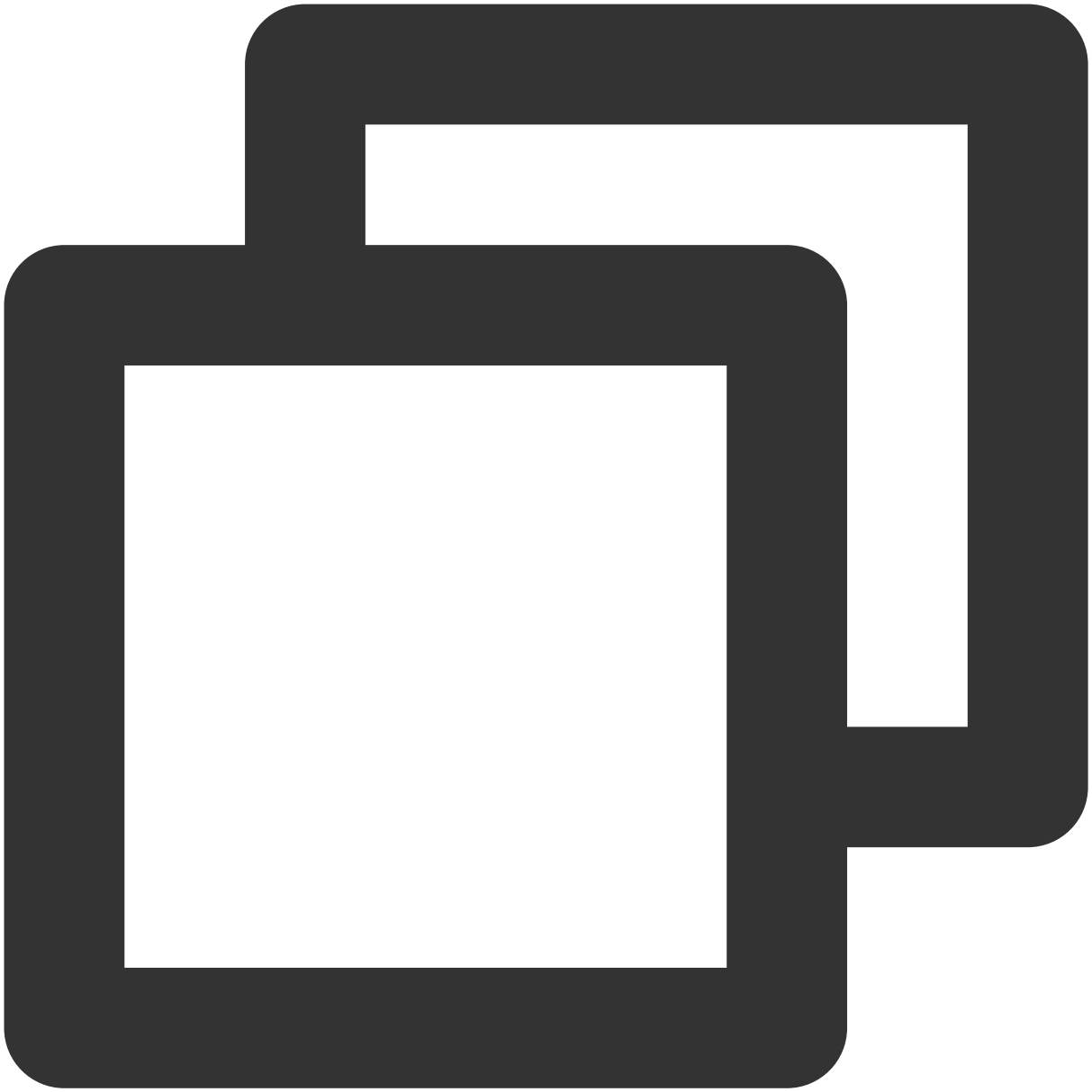
```
#import "TUIBaseChatViewController_Minimalist.h"
#import "TUIC2CChatViewController_Minimalist.h"
#import "TUIGroupChatViewController_Minimalist.h"

// ChatViewController is your own ViewController
@implementation ChatViewController
- (void)viewDidLoad {
  // Create conversation data.
  TUIChatConversationModel *conversationData = [[TUIChatConversationModel alloc] in
  // Pass userID for 1v1 chat, while groupID for group chat.
  conversationData.userID = @"userID";
```

```
    conversationData.groupID = @"groupID";

    // Create chatVC by groupID or userID.
    TUIBaseChatViewController_Minimalist *chatVC = nil;
    if (conversationData.groupID.length > 0) {
        chatVC = [[TUIGroupChatViewController_Minimalist alloc] init];
    } else if (conversationData.userID.length > 0) {
        chatVC = [[TUIC2CChatViewController_Minimalist alloc] init];
    }
    [chatVC setConversationData:conversationData];

    // Option 1: push chatVC.
    [self.navigationController pushViewController:chatVC animated:YES];
    // Option 2: add chatVC to your own ViewController.
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}
@end
```



```
#import "TUIBaseChatViewController.h"
#import "TUIC2CChatViewController.h"
#import "TUIGroupChatViewController.h"

// ChatViewController is your own ViewController
@implementation ChatViewController
- (void)viewDidLoad {
  // Create conversation data.
  TUIChatConversationModel *conversationData = [[TUIChatConversationModel alloc] in
  // Pass userID for 1v1 chat, while groupID for group chat.
  conversationData.userID = @"userID";
```

```
    conversationData.groupID = @"groupID";

    // Create chatVC by groupID or userID.
    TUIBaseChatViewController *chatVC = nil;
    if (conversationData.groupID.length > 0) {
        chatVC = [[TUIGroupChatViewController alloc] init];
    } else if (conversationData.userID.length > 0) {
        chatVC = [[TUIC2CChatViewController alloc] init];
    }
    [chatVC setConversationData:conversationData];

    // Option 1: push chatVC.
    [self.navigationController pushViewController:chatVC animated:YES];
    // Option 2: add chatVC to your own ViewController.
    // [self addChildViewController:chatVC];
    // [self.view addSubview:chatVC.view];
}
@end
```

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

会话列表

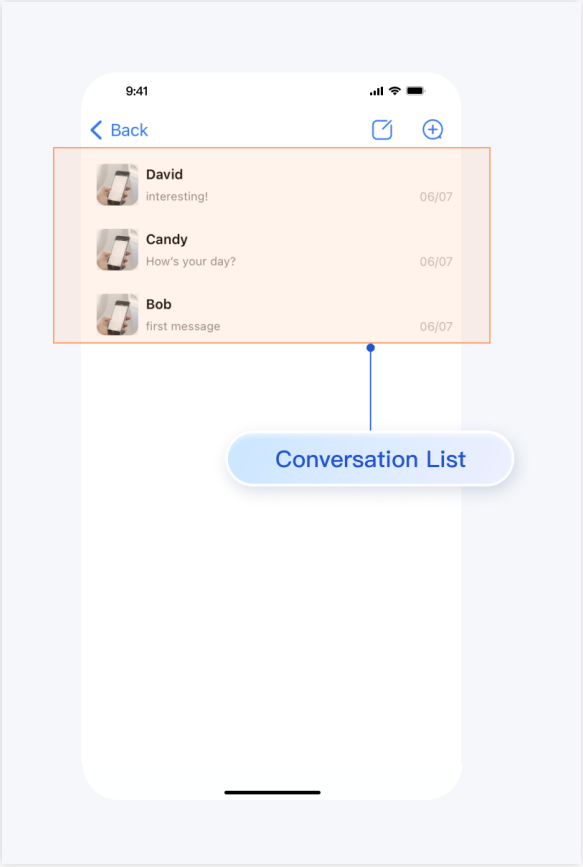
Android

最近更新时间：2024-06-24 16:36:21

本文会引导您构建会话列表界面。

效果展示

加载会话列表效果如下所示：



开发环境要求

Android Studio-Giraffe

Gradle-7.2

Android Gradle Plugin Version-7.0.0

kotlin-gradle-plugin-1.5.31

前置条件

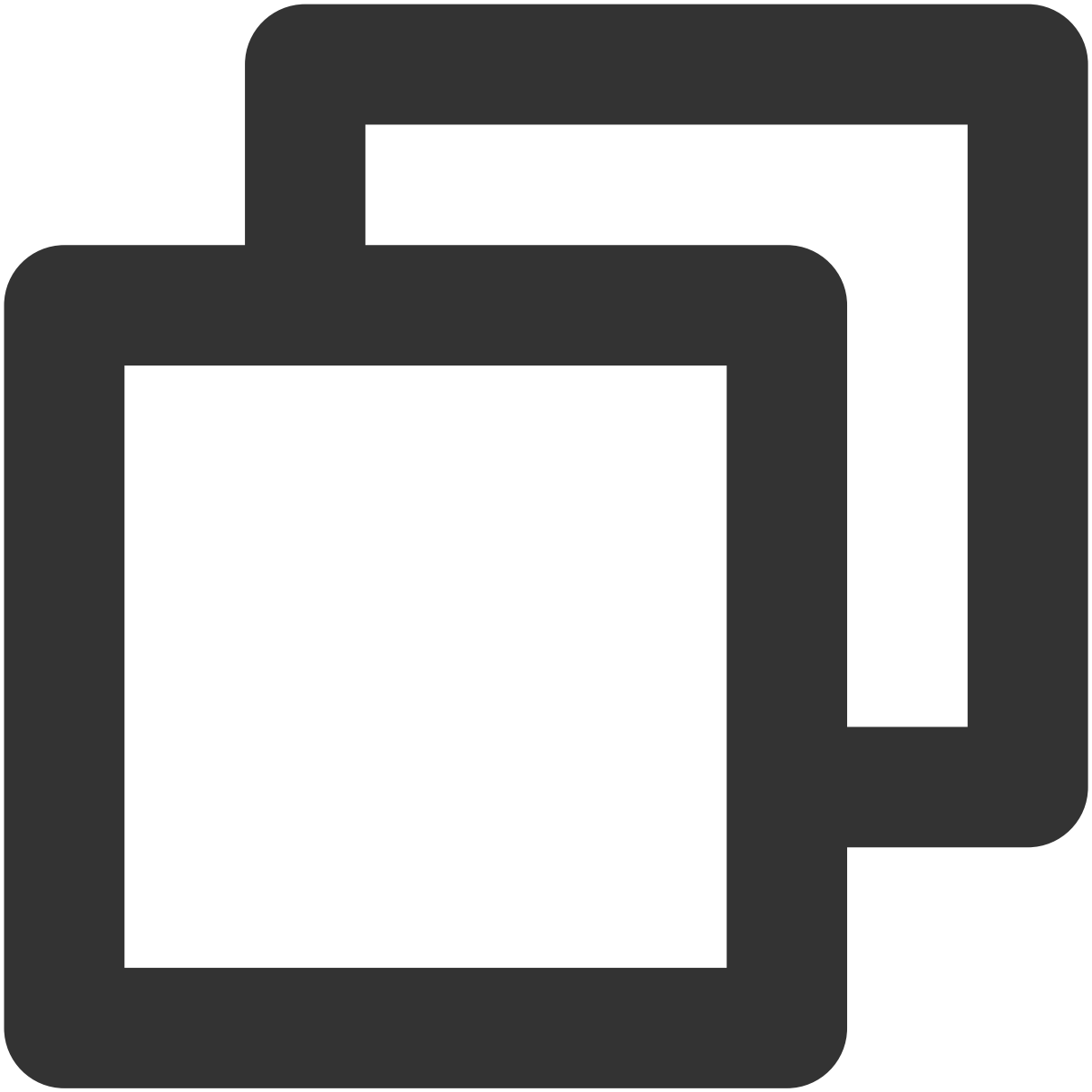
在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIConversation` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

步骤说明

构建会话列表只需要把会话列表对应的 Fragment 添加到您的 Activity 中即可。添加后，Fragment 会自动读取最近的会话。如果同时集成了聊天界面，用户点击会话列表中的某一行，会自动跳转到相应的聊天界面。
MainActivity 布局文件：



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

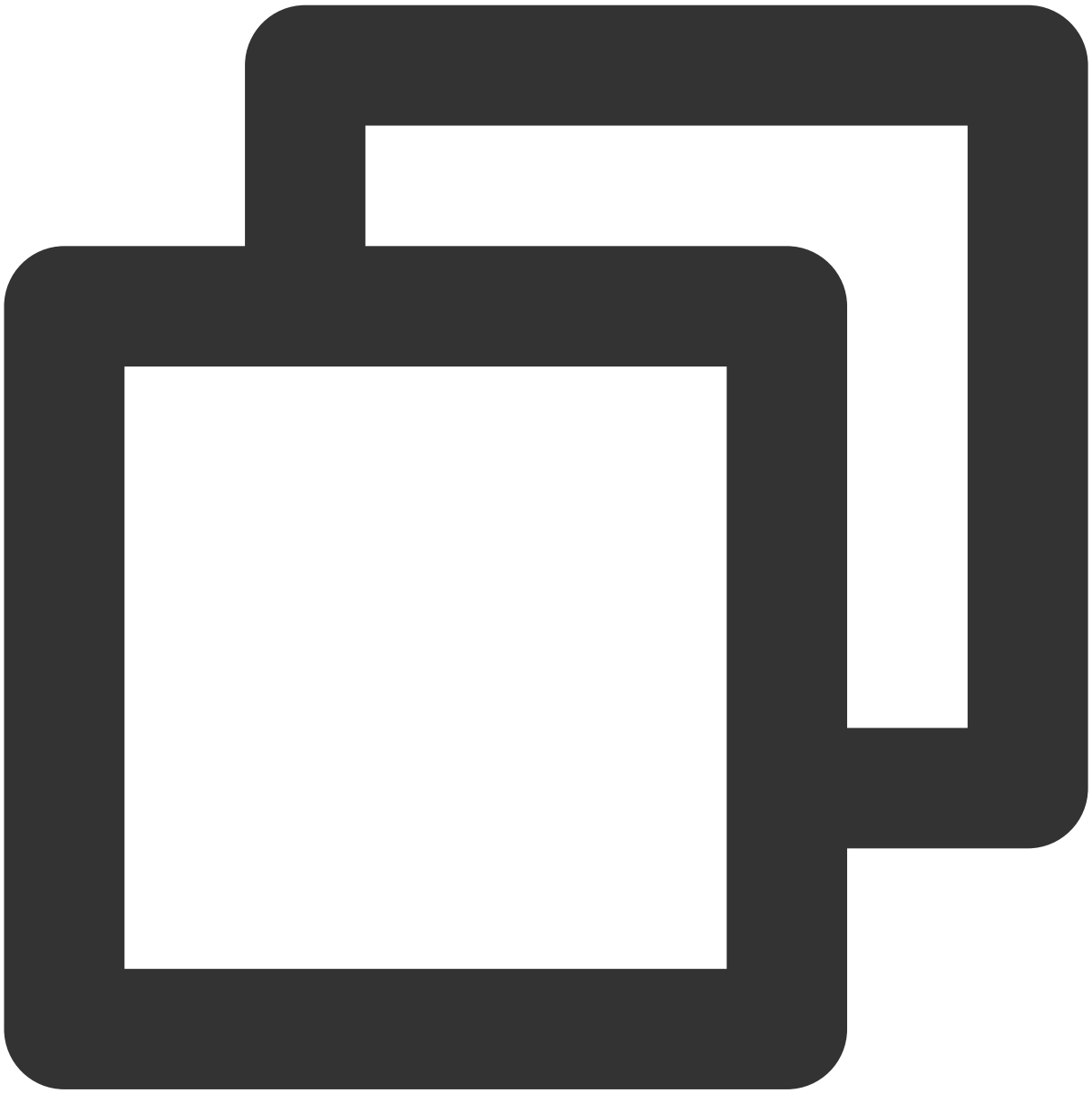
    <FrameLayout
        android:id="@+id/conversation_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>
```

MainActivity Java 文件：

简约版

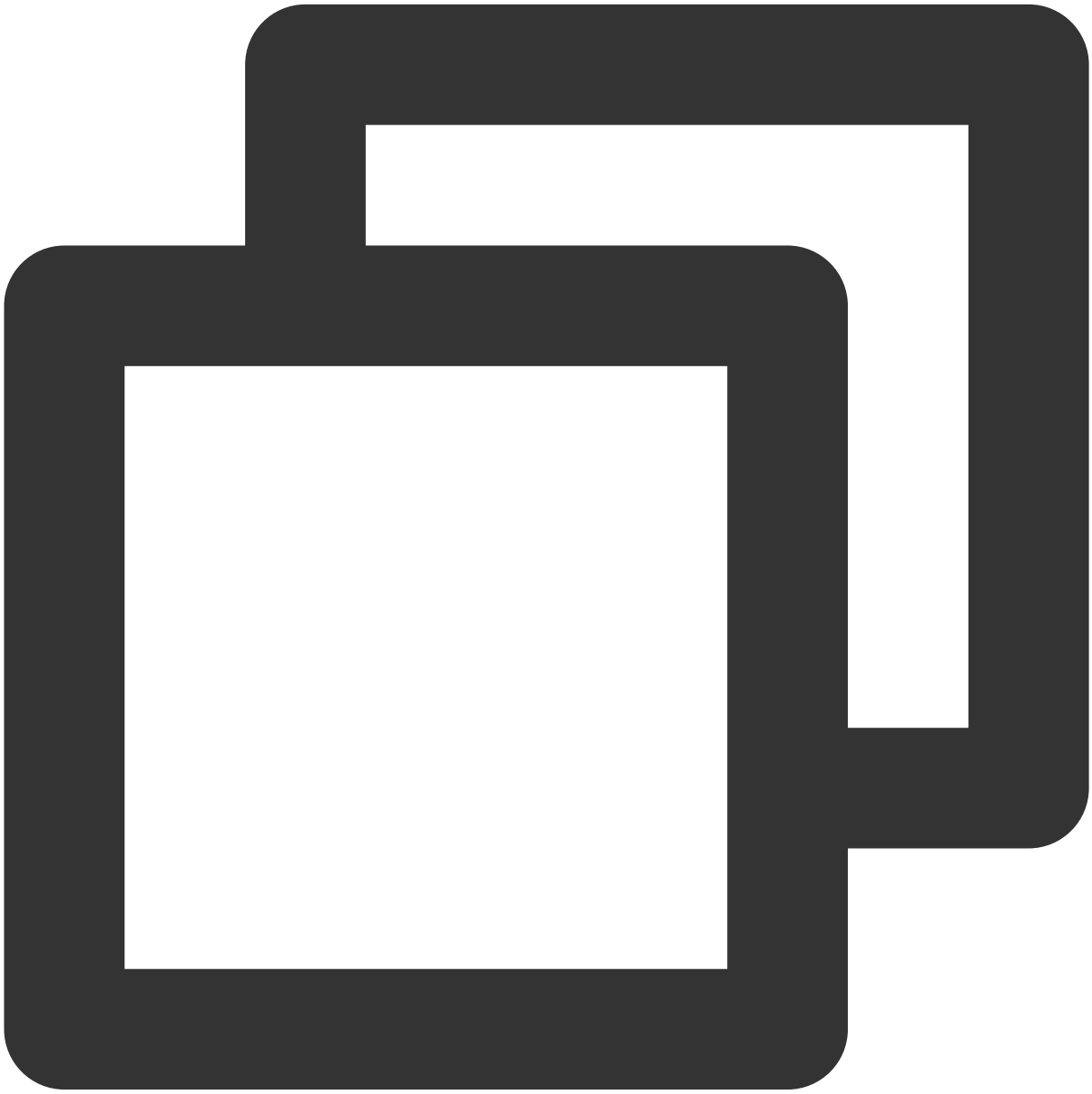
经典版



```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
```

```
        TUIConversationMinimalistFragment fragment = new TUIConversationMinimalistF
        getSupportFragmentManager()
                .beginTransaction()
                .add(R.id.conversation_view, fragment)
                .commitAllowingStateLoss();
    }
}
```



```
public class MainActivity extends AppCompatActivity {

    @Override
```

```
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        TUIConversationFragment fragment = new TUIConversationFragment();
        getSupportFragmentManager()
                .beginTransaction()
                .add(R.id.conversation_view, fragment)
                .commitAllowingStateLoss();
    }
}
```

注意：

如果您事先没有跟任何人、任何群组发送过消息，是不会产生会话的，此时添加 TUIConversationMinimalistFragment，列表为空。为了体验效果，建议您先给一些账号发送消息，触发会话的产生。如果您想了解如何在聊天界面发送消息，请参考文档：构建聊天界面。

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

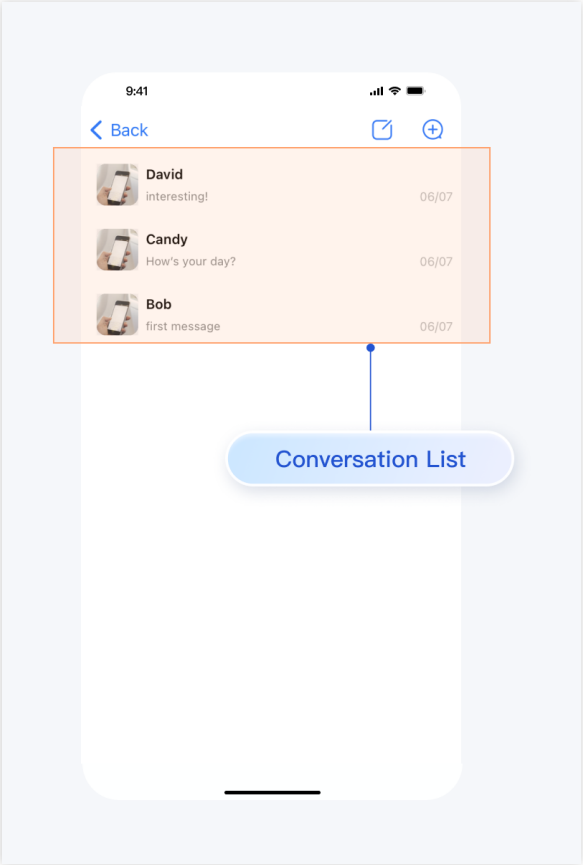
iOS

最近更新时间：2024-06-24 16:37:07

本文会引导您构建会话列表界面。

效果展示

加载会话列表效果如下所示：



开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：

1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIConversation` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：

1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。

如果您已经完成，请继续阅读下文。

步骤说明

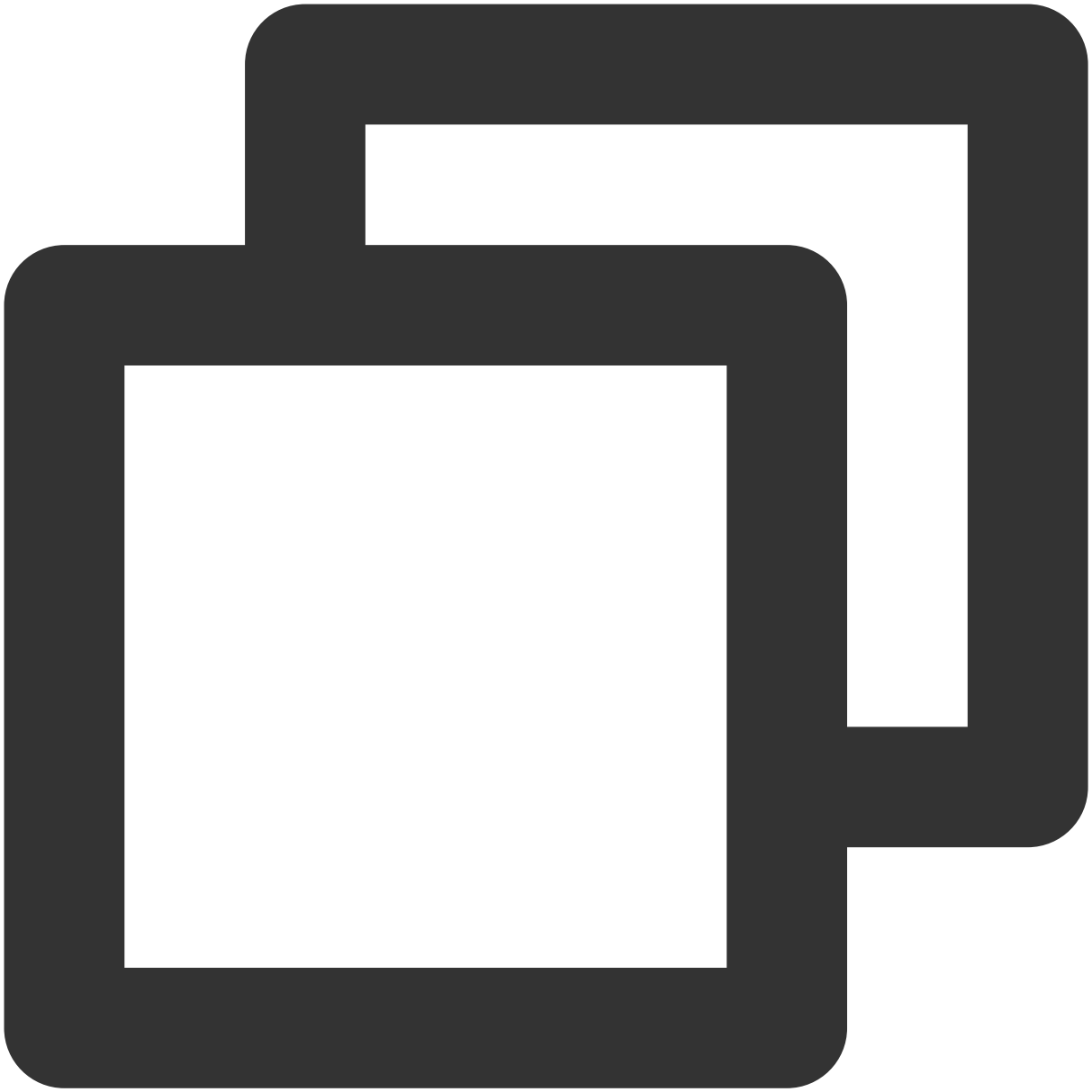
构建会话列表通常分为以下 3 步：

1. 加载会话列表。列表对应着 `TUIConversationListController` 对象。加载后， `TUIConversationListController` 会自动读取最近会话。
2. 用户点击会话列表中的某一行， `TUIConversationListController` 会抛出 `didSelectConversation` 事件。
3. 在 `didSelectConversation` 中响应点击，通常是进入该会话的聊天界面。

示例代码如下所示：

简约版

经典版



```
#import "TUIConversationListController_Minimalist.h"
#import "TUIBaseChatViewController_Minimalist.h"
#import "TUIGroupChatViewController_Minimalist.h"
#import "TUIC2CChatViewController_Minimalist.h"

// ConversationController is your own ViewController
@implementation ConversationController
- (void)viewDidLoad {
    [super viewDidLoad];
    // TUIConversationListController_Minimalist
    TUIConversationListController_Minimalist *vc = [[TUIConversationListController_
```

```
    vc.delegate = self;

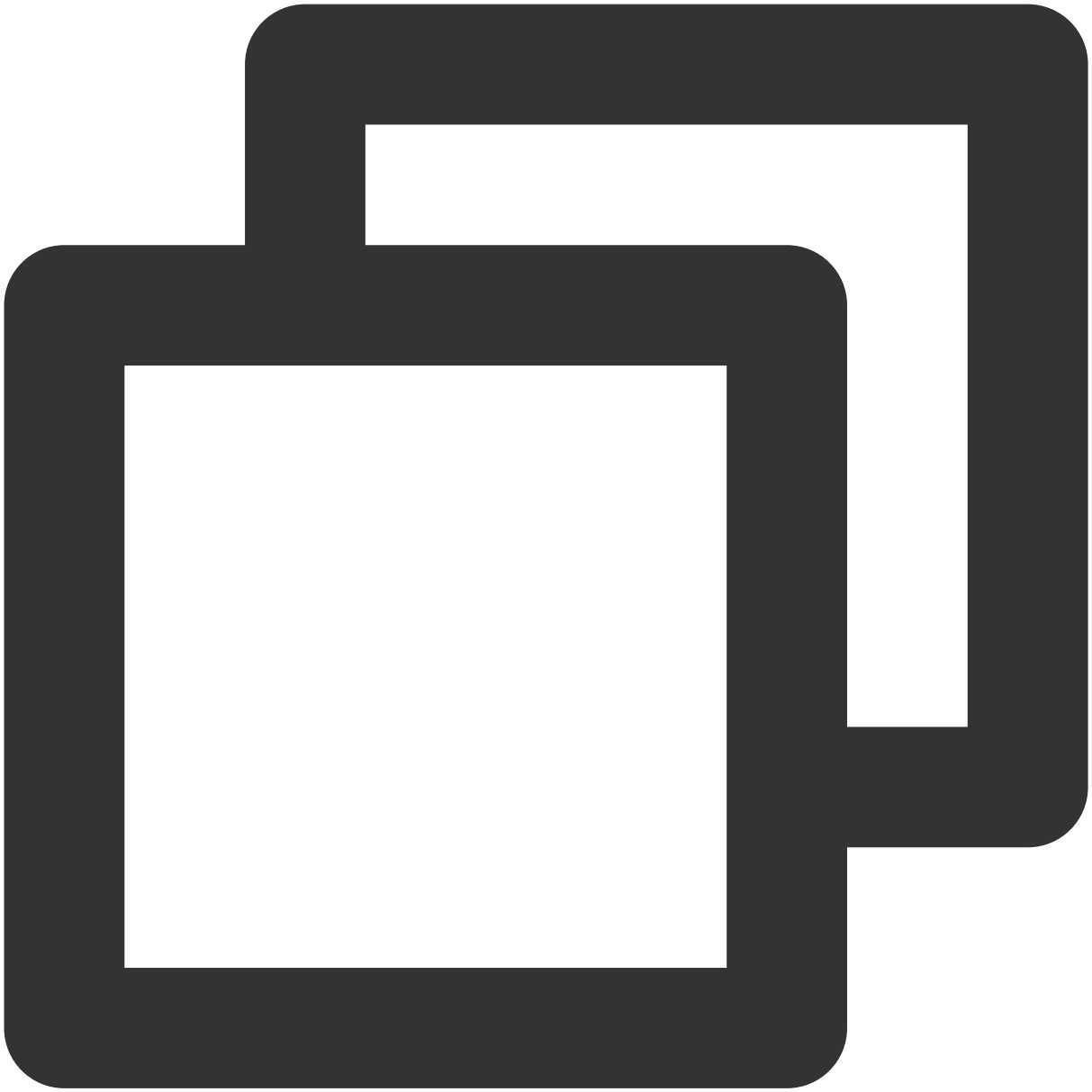
    // Option 1: push vc.
    [self.navigationController pushViewController:vc animated:YES];
    // Option 2: Add TUIConversationListController_Minimalist to your own ViewContr
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}

- (void)conversationListController:(TUIConversationListController_Minimalist *)conv
            didSelectConversation:(TUIConversationCellData *)conversation {
    // Conversation list click event, typically, opening the chat UI
    TUIChatConversationModel *conversationData = [TUIChatConversationModel new];
    conversationData.userID = conversation.userID;
    conversationData.title = conversation.title;
    conversationData.faceUrl = conversation.faceUrl;

    // Create chatVC by groupID or userID.
    TUIBaseChatViewController_Minimalist *chatVC = nil;
    if (conversationData.groupID.length > 0) {
        chatVC = [[TUIGroupChatViewController_Minimalist alloc] init];
    } else if (conversationData.userID.length > 0) {
        chatVC = [[TUIC2CChatViewController_Minimalist alloc] init];
    }
    chatVC.conversationData = conversationData;

    // Option 1: push chatVC.
    [self.navigationController pushViewController:chatVC animated:YES];
    // Option 2: add chatVC to your own ViewController.
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}

@end
```



```
#import "TUIConversationListController.h"
#import "TUIBaseChatViewController_Minimalist.h"
#import "TUIGroupChatViewController.h"
#import "TUIC2CChatViewController.h"

// ConversationController is your own ViewController
@implementation ConversationController
- (void)viewDidLoad {
    [super viewDidLoad];
    // TUIConversationListController
    TUIConversationListController *vc = [[TUIConversationListController alloc] init
```

```
    vc.delegate = self;

    // Option 1: push vc.
    [self.navigationController pushViewController:vc animated:YES];
    // Option 2: Add TUIConversationListController to your own ViewController
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}

- (void)conversationListController:(TUIConversationListController *)conversationCon
            didSelectConversation:(TUIConversationCellData *)conversation {
    // Conversation list click event, typically, opening the chat UI
    TUIChatConversationModel *conversationData = [TUIChatConversationModel new];
    conversationData.userID = conversation.userID;
    conversationData.title = conversation.title;
    conversationData.faceUrl = conversation.faceUrl;

    // Create chatVC by groupID or userID.
    TUIBaseChatViewController *chatVC = nil;
    if (conversationData.groupID.length > 0) {
        chatVC = [[TUIGroupChatViewController alloc] init];
    } else if (conversationData.userID.length > 0) {
        chatVC = [[TUIC2CChatViewControlleralloc] init];
    }
    chatVC.conversationData = conversationData;

    // Option 1: push chatVC.
    [self.navigationController pushViewController:chatVC animated:YES];
    // Option 2: add chatVC to your own ViewController.
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}

@end
```

注意：

如果您事先没有跟任何人、任何群组发送过消息，是不会产生会话的，此时加载 `TUIConversationListController` ，列表为空。为了体验效果，建议您先给一些账号发送消息，触发会话的产生。如果您想了解如何在聊天界面发送消息，请参考文档：构建聊天界面。

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

联系人界面

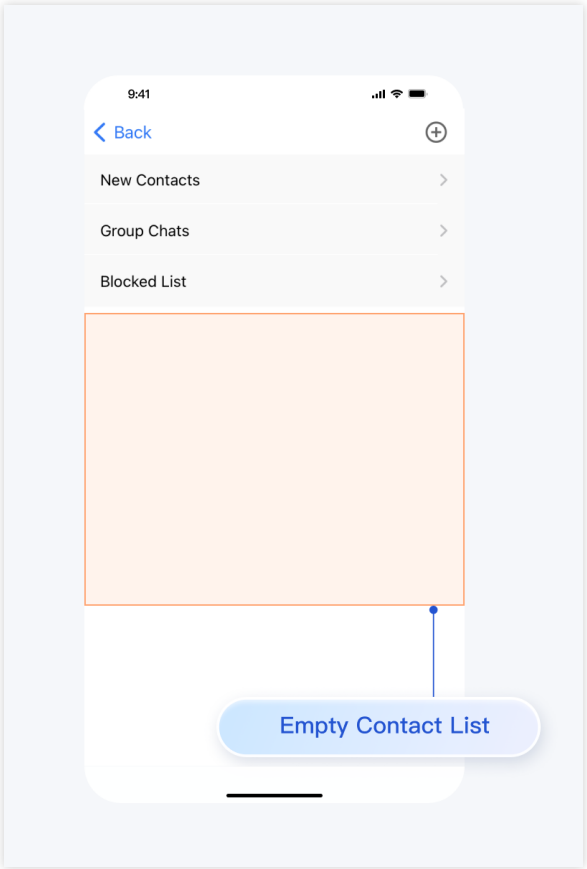
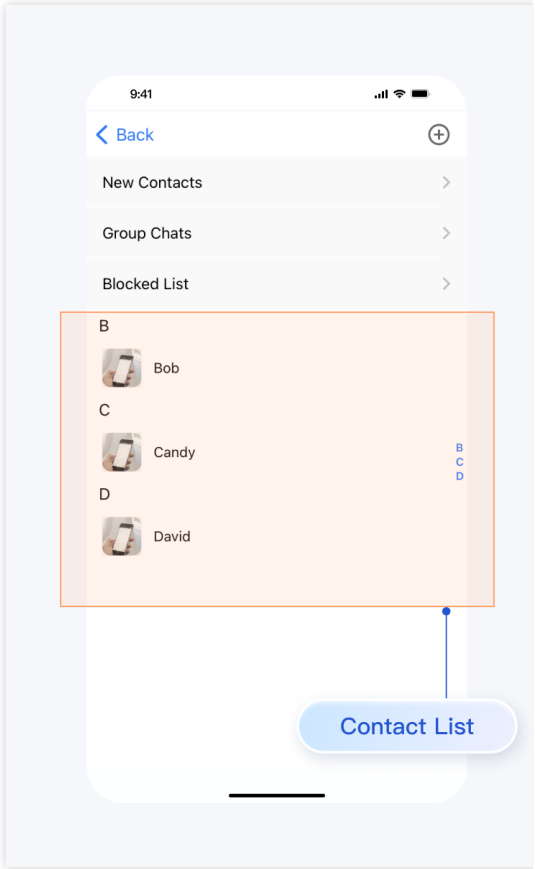
Android

最近更新时间：2024-06-24 16:38:20

本文会引导您构建联系人界面。

效果展示

如果您事先没有添加过联系人，加载出来的联系人界面是空的。添加联系人后，联系人会显示在界面列表中，如下图所示：

联系人列表为空	联系人列表非空
	

开发环境要求

Android Studio-Giraffe

版权所有：腾讯云计算（北京）有限责任公司

Gradle-7.2
Android Gradle Plugin Version-7.0.0
kotlin-gradle-plugin-1.5.31

前置条件

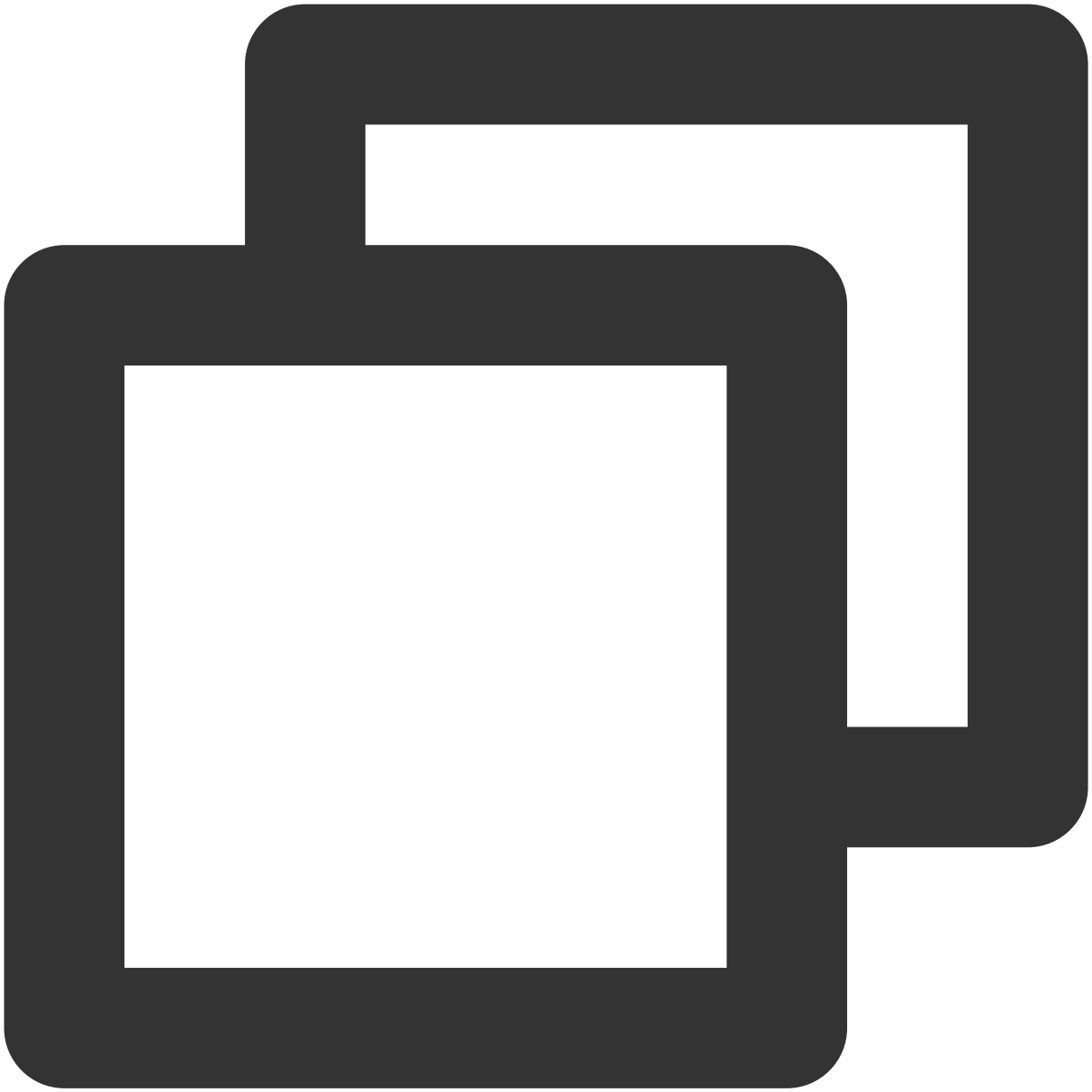
在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIContact` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

步骤说明

联系人界面只需把联系人列表对应的 Fragment 添加到您的 Activity 中即可。
MainActivity 布局文件：



```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

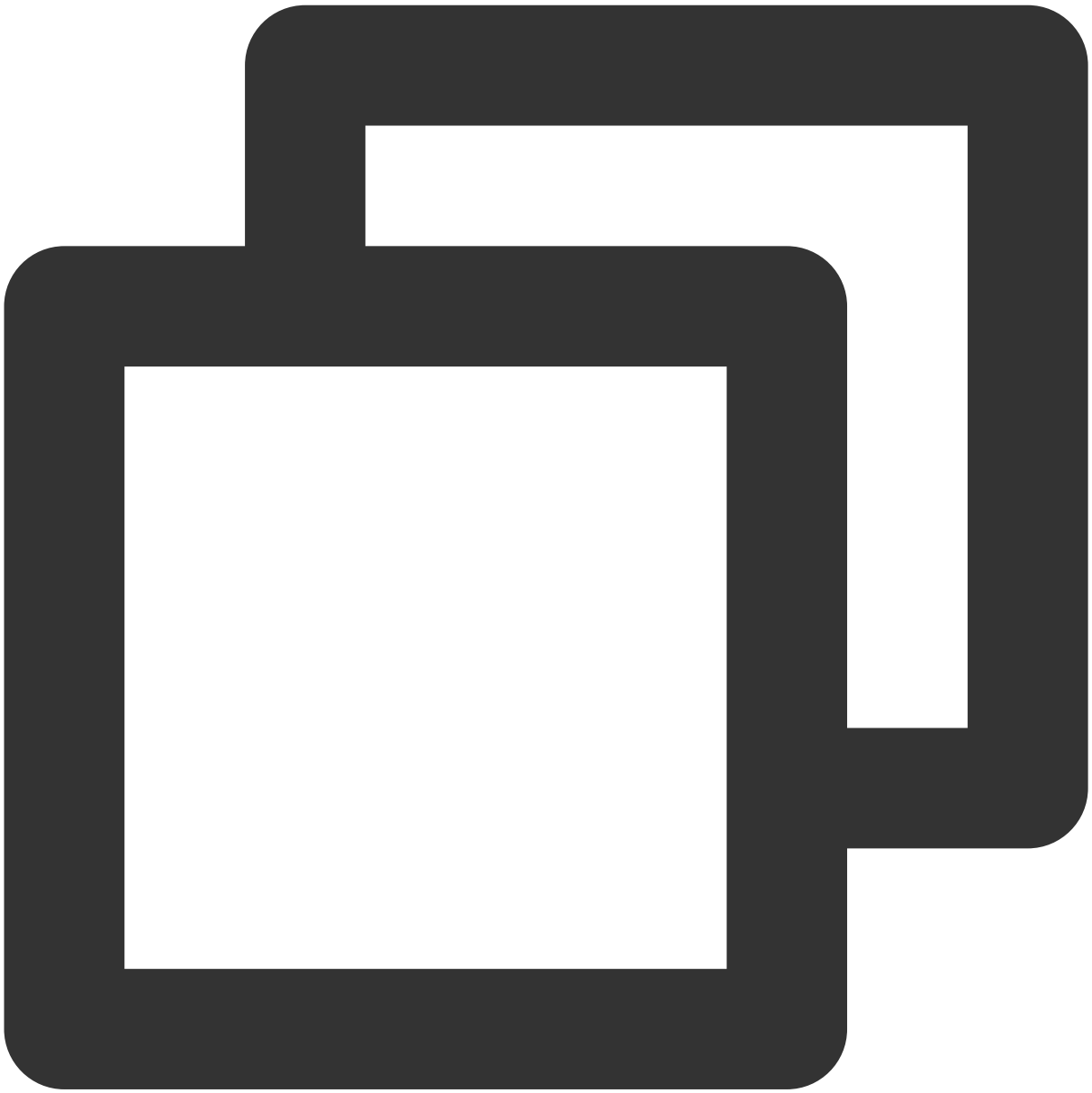
    <FrameLayout
        android:id="@+id/contact_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>
```

MainActivity Java 文件：

简约版

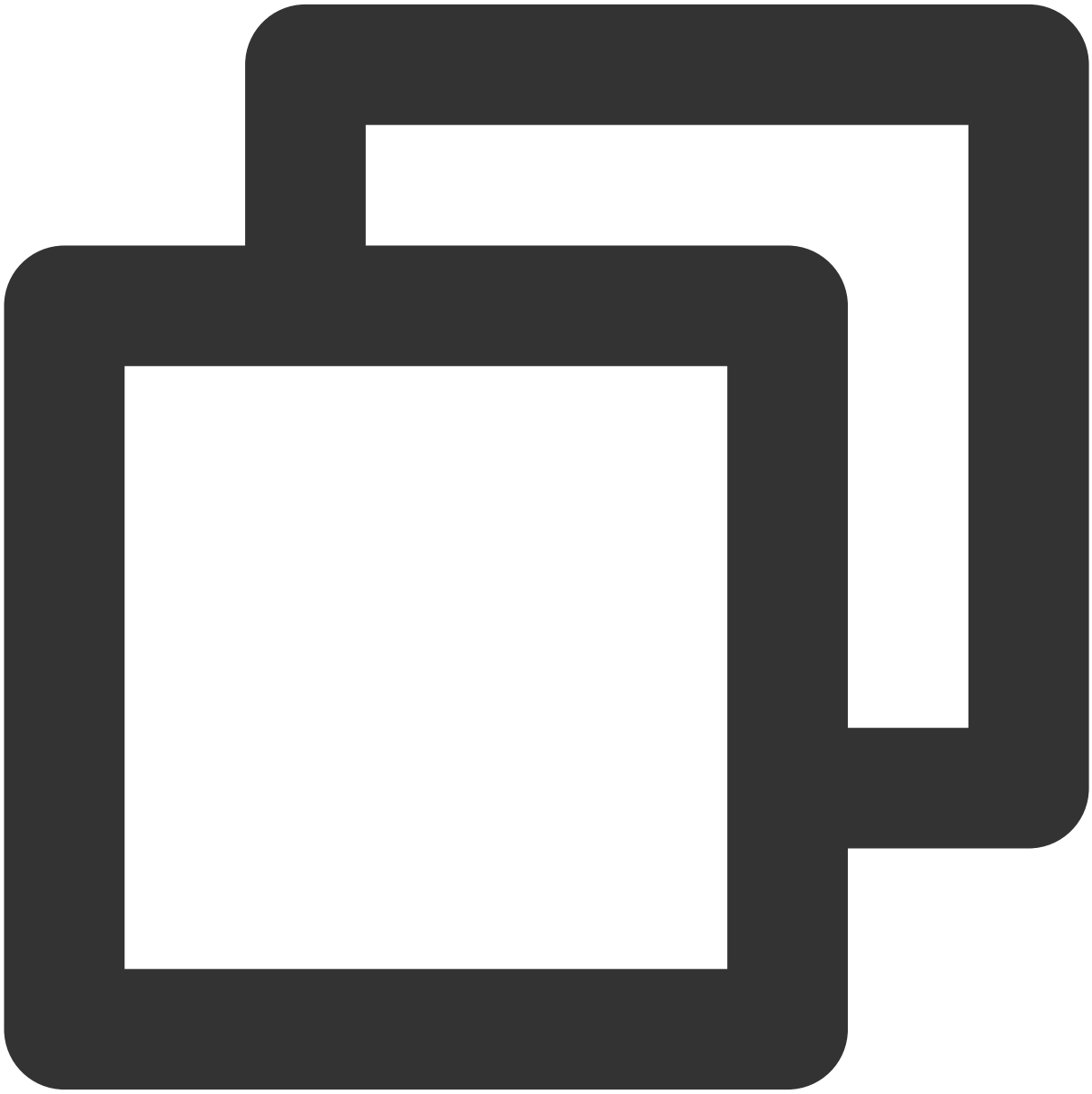
经典版



```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
```

```
        TUIContactMinimalistFragment fragment = new TUIContactMinimalistFragment();
        getSupportFragmentManager()
                .beginTransaction()
                .add(R.id.contact_view, fragment)
                .commitAllowingStateLoss();
    }
}
```

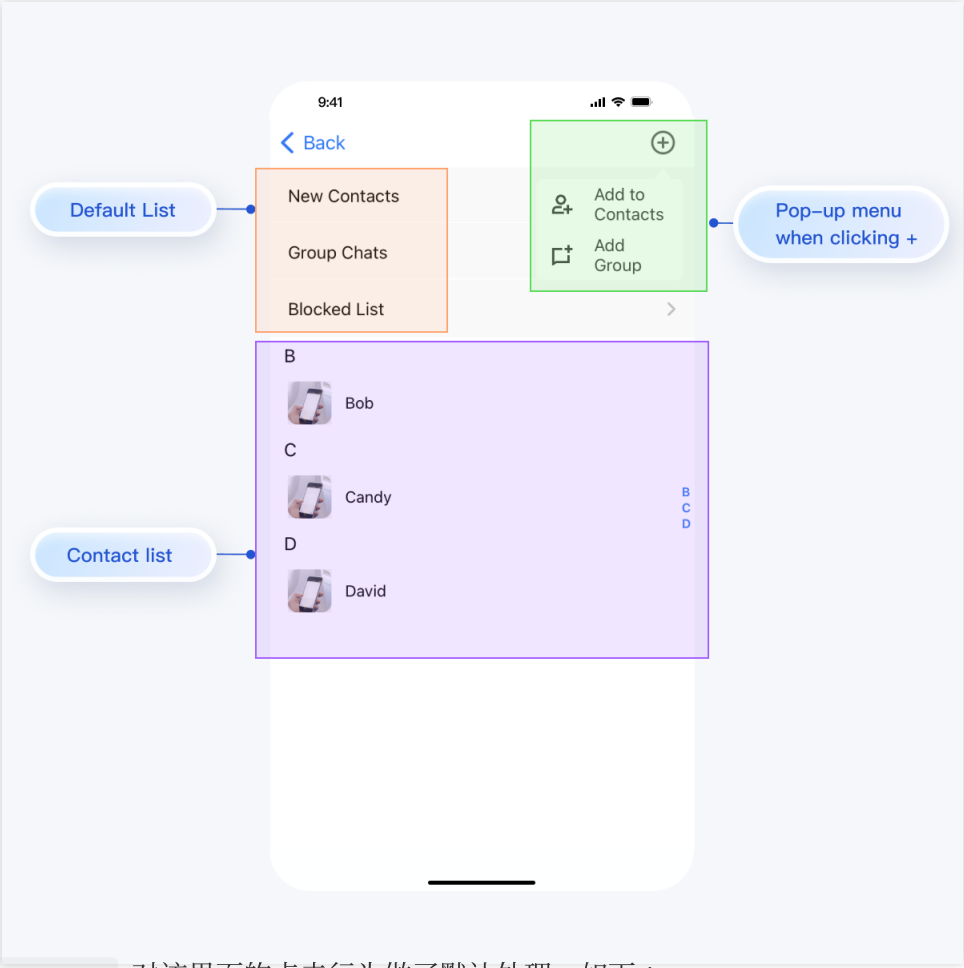


```
public class MainActivity extends AppCompatActivity {

    @Override
```

```
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        TUIContactFragment fragment = new TUIContactFragment();
        getSupportFragmentManager()
                .beginTransaction()
                .add(R.id.contact_view, fragment)
                .commitAllowingStateLoss();
    }
}
```

联系人界面功能分区如下图所示：



`TUIContact` 对该界面的点击行为做了默认处理，如下：

动作	效果
单击 `New Contacts`	展示未处理的加好友请求
单击 `Group Chats`	展示当前登录账号的所有群聊
单击 `Blocked List`	展示当前登录账号的黑名单

单击联系人头像	进入联系人管理界面
单击右上角界面 `+` 号	弹出 `Add to Contacts`，`Add Group` 菜单

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

版权所有：腾讯云计算（北京）有限责任公司

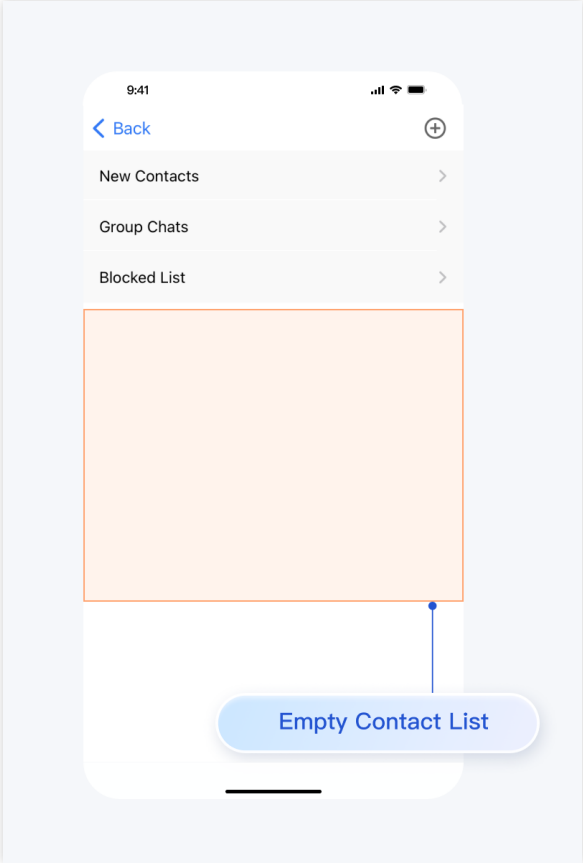
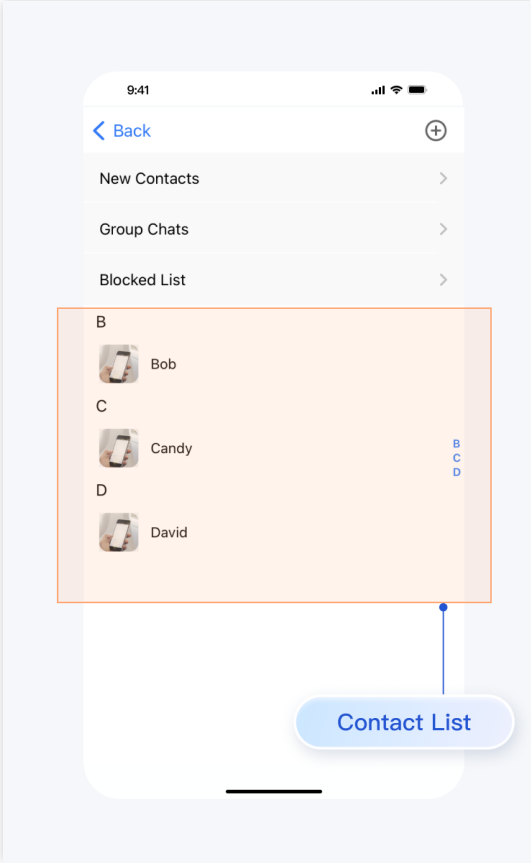
iOS

最近更新时间：2024-06-24 16:39:05

本文会引导您构建联系人界面。

效果展示

如果您事先没有添加过联系人，加载出来的联系人界面是空的。添加联系人后，联系人会显示在界面列表中，如下图所示：

联系人列表为空	联系人列表非空
	

开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

前置条件

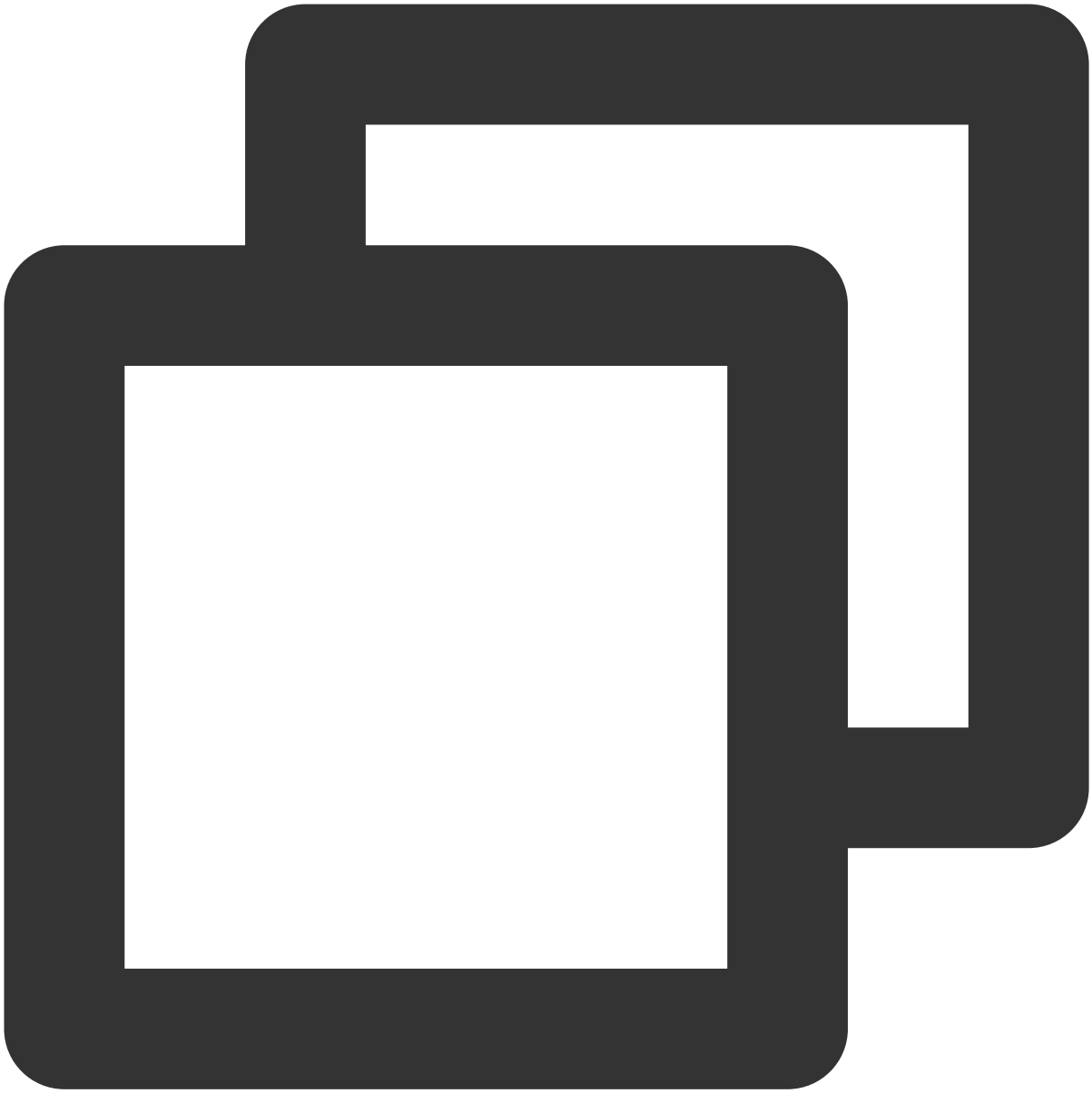
在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIContact` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

步骤说明

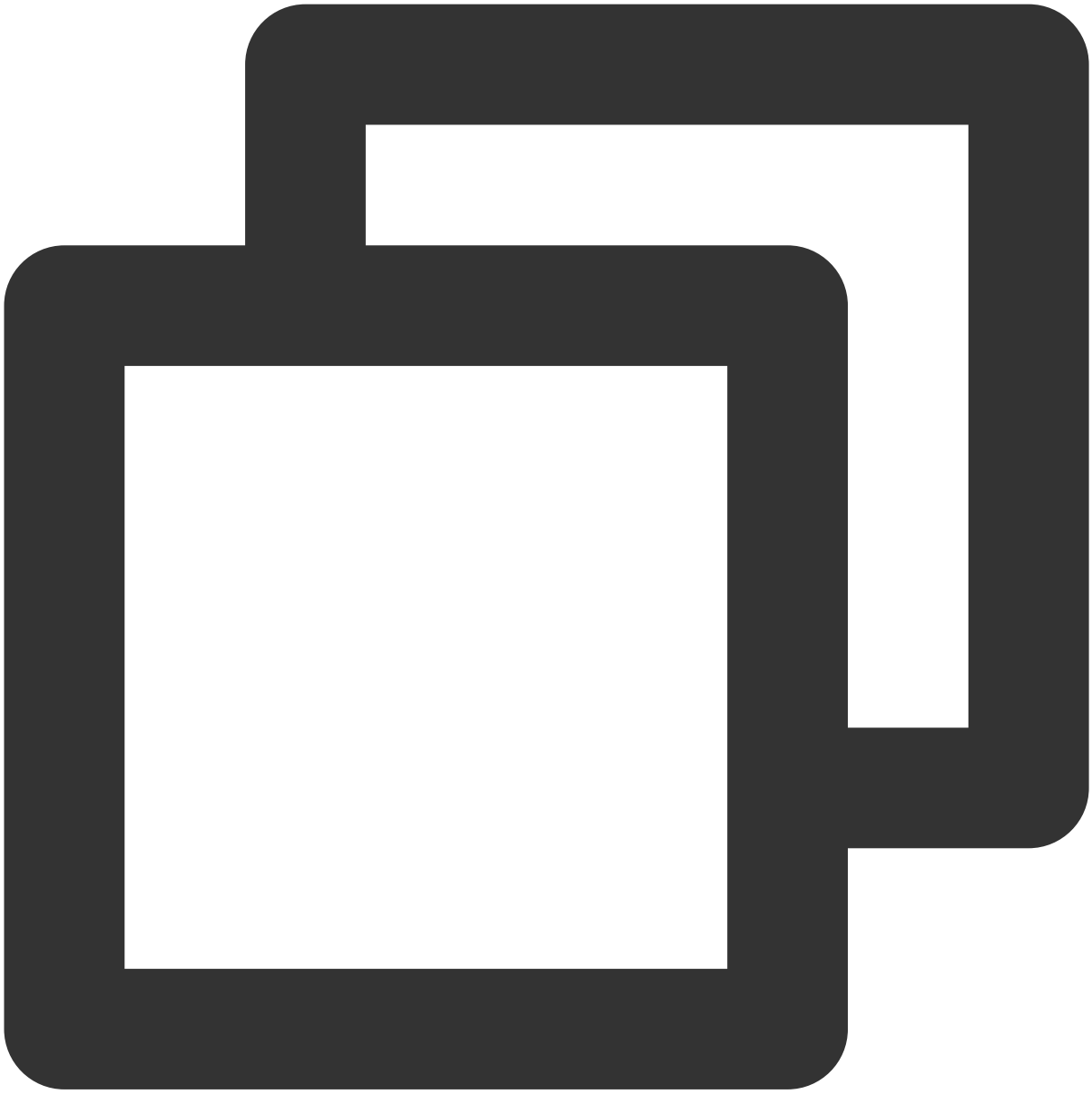
联系人界面只需创建 `TUIContactController` 对象并显示出来即可。
示例代码如下：
简约版
经典版



```
#import "TUIContactController_Minimalist.h"

// ContactController is your own ViewController
@implementation ContactController
- (void)viewDidLoad {
  // Create TUIContactController_Minimalist
  TUIContactController_Minimalist *vc = [[TUIContactController_Minimalist alloc] in
  // Option 1: push vc.
  [self.navigationController pushViewController:vc animated:YES];
  // Option 2: add vc to your own ViewController.
  // [self addChildViewController:vc];
```

```
  // [self.view addSubview:vc.view];
}
@end
```

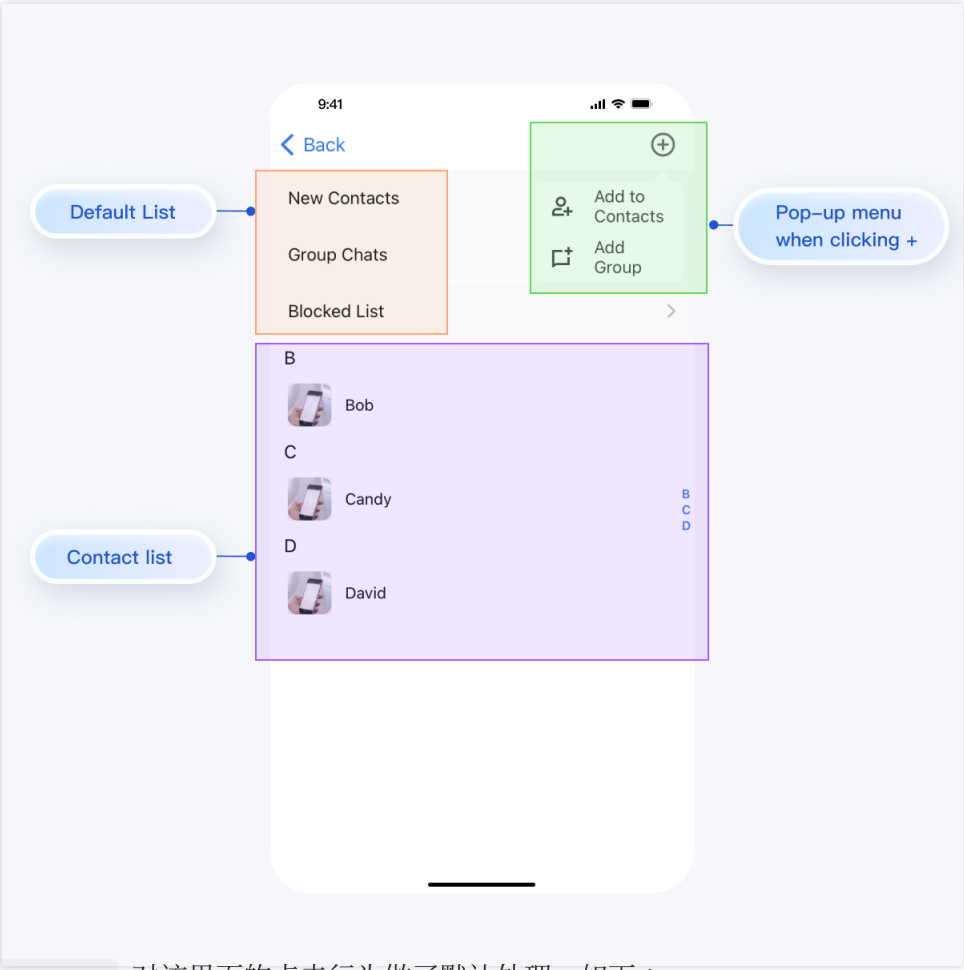


```
#import "TUIContactController.h"

// ContactController is your own ViewController
@implementation ContactController
- (void)viewDidLoad {
  // Create TUIContactController
  TUIContactController *vc = [[TUIContactController alloc] init];
```

```
    // Option 1: push vc.
    [self.navigationController pushViewController:vc animated:YES];
    // Option 2: add vc to your own ViewController.
    // [self addChildViewController:vc];
    // [self.view addSubview:vc.view];
}
@end
```

联系人界面功能分区如下图所示：



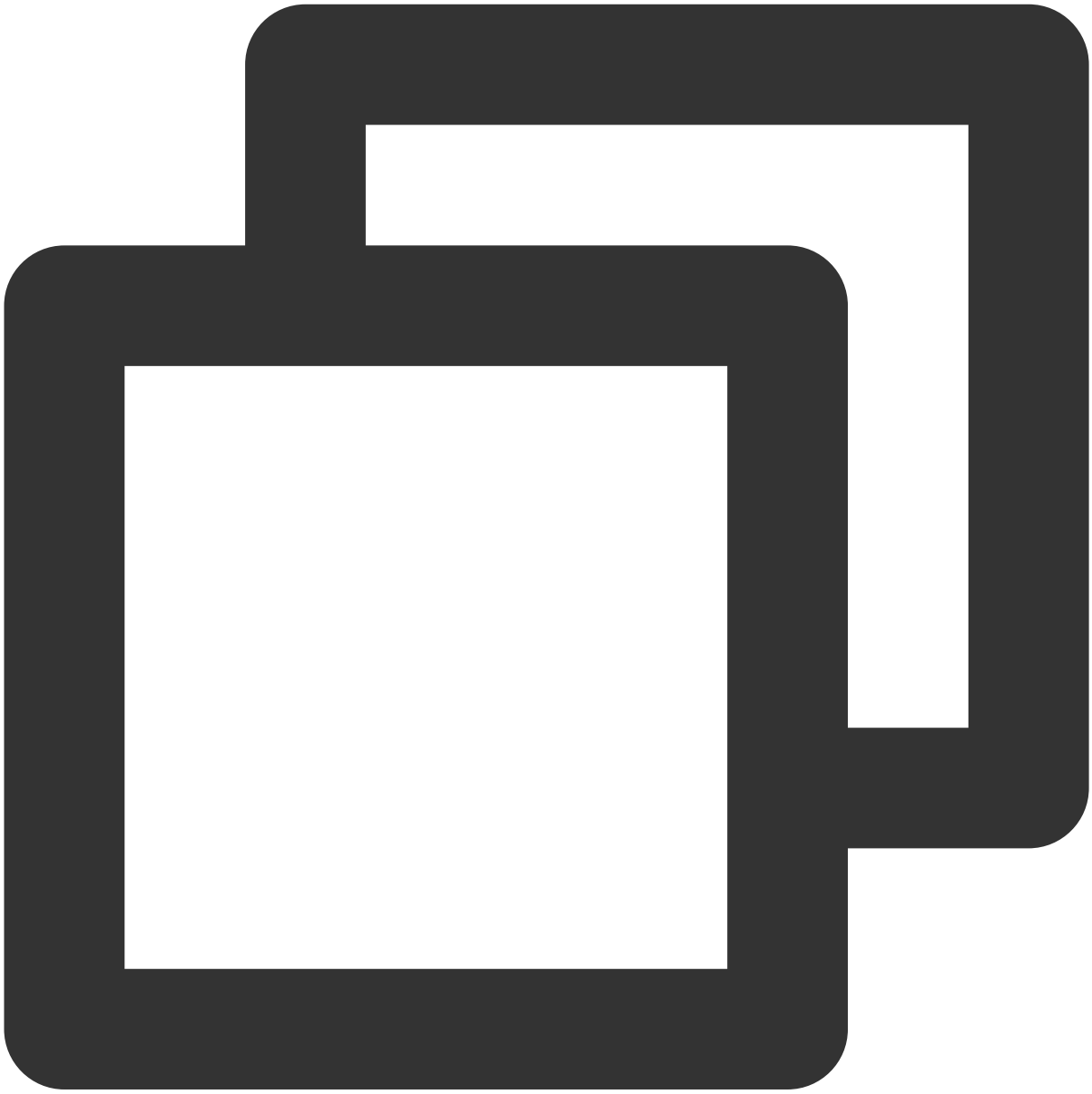
`TUIContact` 对该界面的点击行为做了默认处理，如下：

动作	效果
点击 `New Contacts`	展示未处理的加好友请求
点击 `Group Chats`	展示当前登录账号的所有群聊
点击 `Blocked List`	展示当前登录账号的黑名单
点击联系人头像	进入联系人管理界面
点击右上角界面 `+` 号	弹出 `Add to Contacts`，`Add Group` 菜单

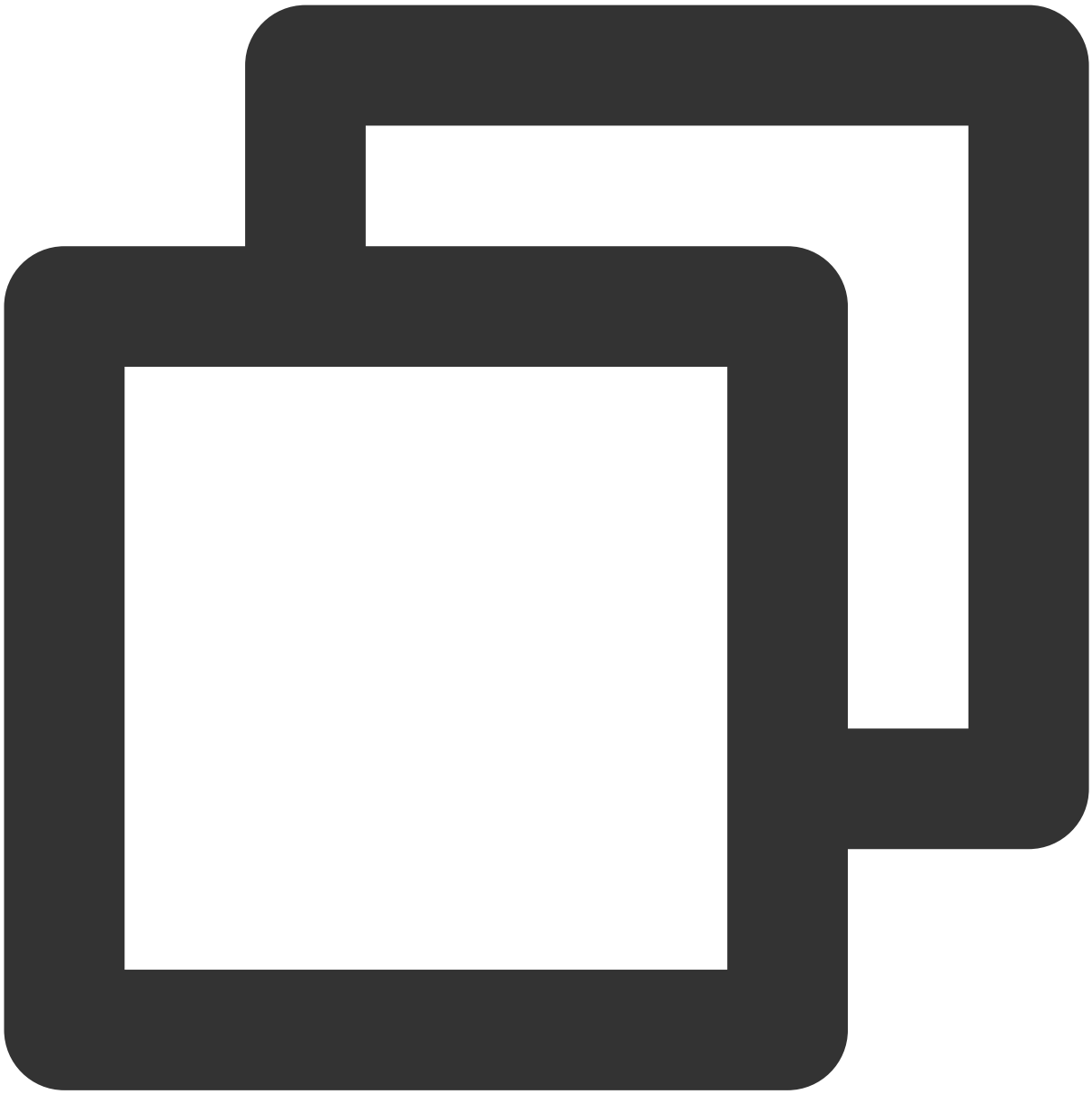
其中，点击联系人头像、点击 `Add to Contacts` 、点击 `Group Chats` 的行为，您可以通过实现 `TUIContactControllerListener` 中的方法自定义：

简约版

经典版



```
@protocol TUIContactControllerListener_Minimalist <NSObject>
@optional
- (void)onSelectFriend:(TUICommonContactCell *)cell;
- (void)onAddNewFriend:(TUICommonTableViewCell *)cell;
- (void)onGroupConversation:(TUICommonTableViewCell *)cell;
@end
```



```
@protocol TUIContactControllerListener <NSObject>
@optional
- (void)onSelectFriend:(TUICommonContactCell *)cell;
- (void)onAddNewFriend:(TUICommonTableViewCell *)cell;
- (void)onGroupConversation:(TUICommonTableViewCell *)cell;
@end
```

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群 ，您将获得可靠的技术支持。

添加联系人

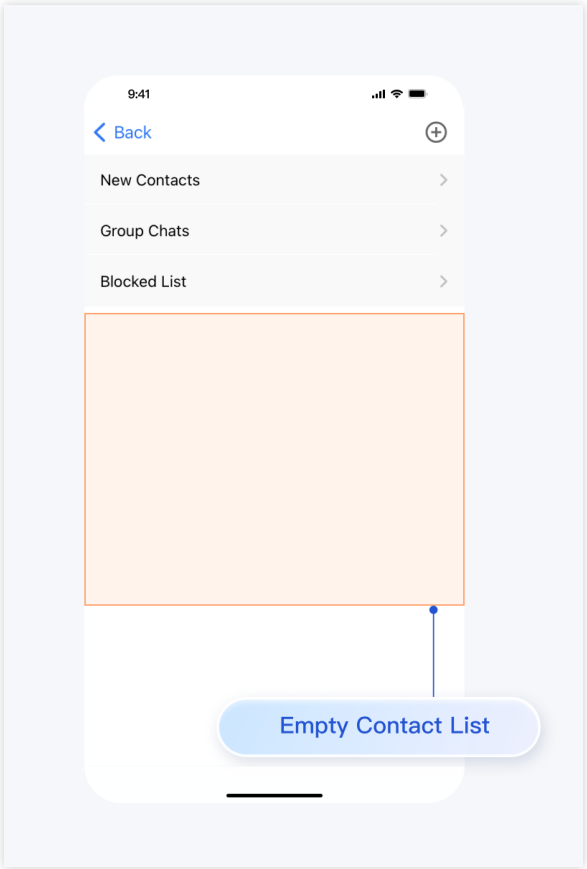
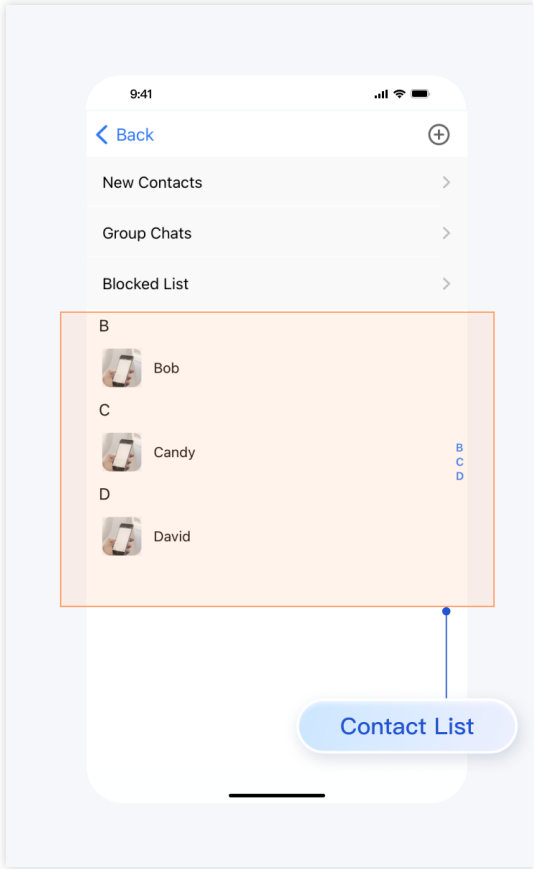
Android

最近更新时间：2024-06-24 16:40:18

本文会引导您在 TUIKit 上添加联系人。

效果展示

如果您事先没有添加过联系人，加载出来的联系人界面是空的。添加联系人后，联系人会显示在界面列表中，如下图所示：

联系人列表为空	联系人列表非空
	

开发环境要求

Android Studio-Giraffe

版权所有：腾讯云计算（北京）有限责任公司

Gradle-7.2
Android Gradle Plugin Version-7.0.0
kotlin-gradle-plugin-1.5.31

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIContact` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

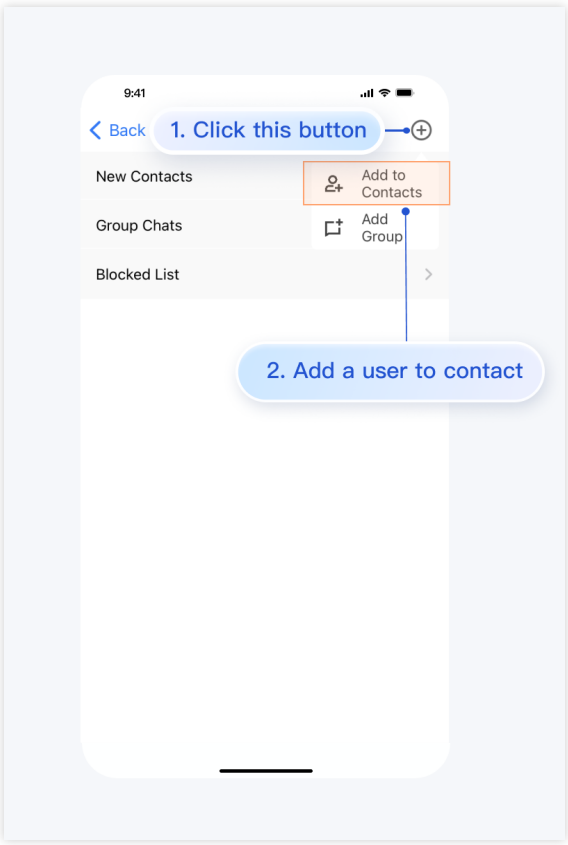
如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

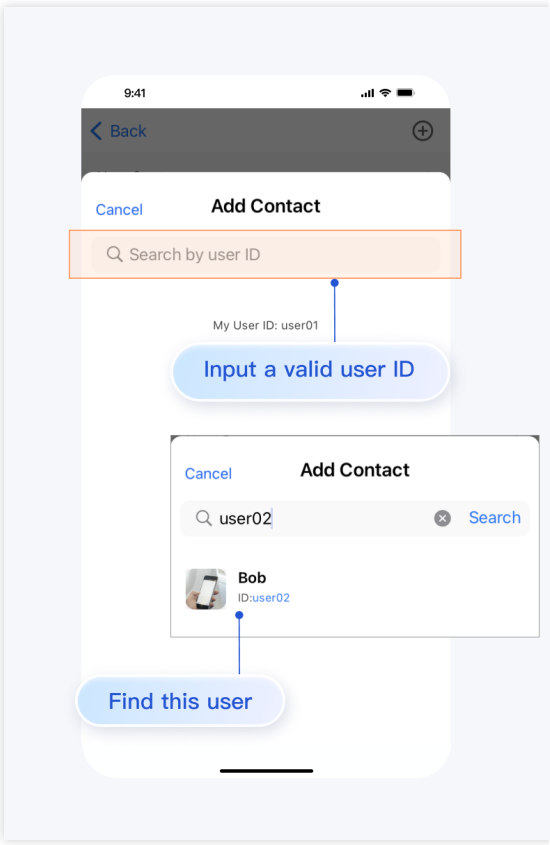
步骤说明

我们推荐您直接在 `TUKit` 中操作，手动添加联系人，步骤如下：
1. 按照 构建联系人界面 中的介绍，将联系人界面展示出来。
2. 点击该界面右上角的 `+` 按钮，在子菜单中选择 `Add to Contacts` 。
3. 输入有效的 userID，搜索出用户。您可以去控制台 `Account Managerment` 页面获取有效的 userID。页面路径：Applications > Your App > Chat > Users > Account Management。
4. 添加用户为联系人。

效果如下图所示：

点击 `Add to Contacts`	搜索用户





一般情况下，发送添加请求后，对方会默认同意。但如果对方账号设置过 `Friend Request Verification` ，可能会有不同表现。该选项路径为：

Applications > `Your App` > Chat > Users > Account Management > `Choose an account` > Edit > Friend Request Verification。

`Friend Request Verification` 取值如下：

Friend Request Verification 取值	含义
Accept all friend requests	默认值，接受所有的好友请求
Manually accept or reject friend requests	需被请求方手动操作，同意或拒绝好友请求
Reject all friend requests	拒绝所有好友请求

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

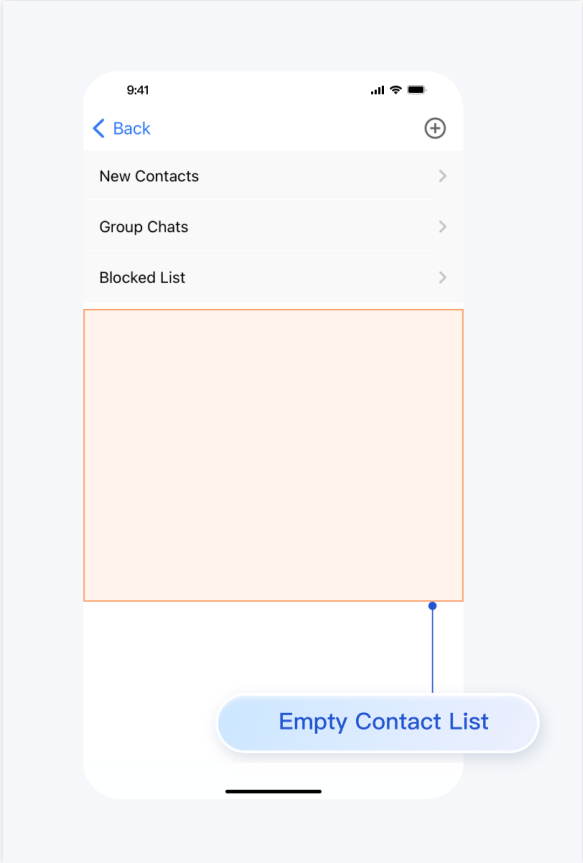
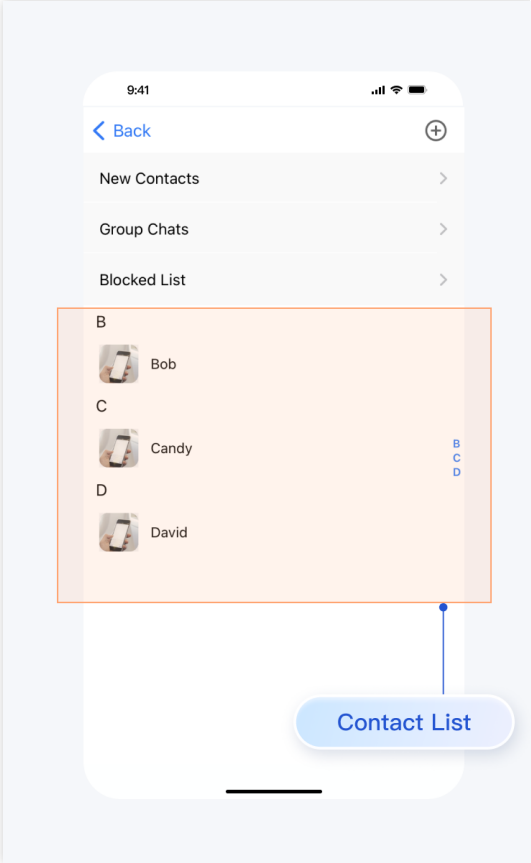
iOS

最近更新时间：2024-06-24 16:41:18

本文会引导您在 TUIKit 上添加联系人。

效果展示

如果您事先没有添加过联系人，加载出来的联系人界面是空的。添加联系人后，联系人会显示在界面列表中，如下图所示：

联系人列表为空	联系人列表非空
	

开发环境要

Xcode 10 及以上
iOS 9.0 及以上

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 或 `TUIContact` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

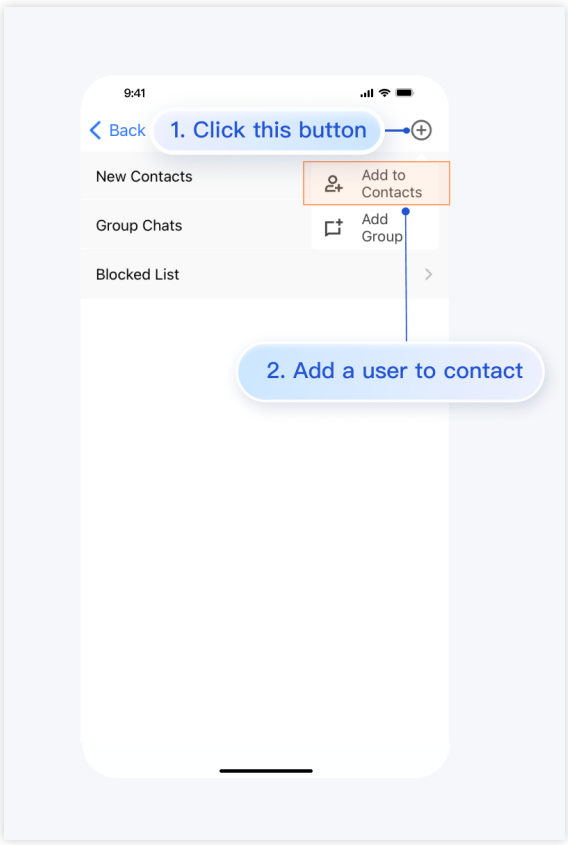
如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

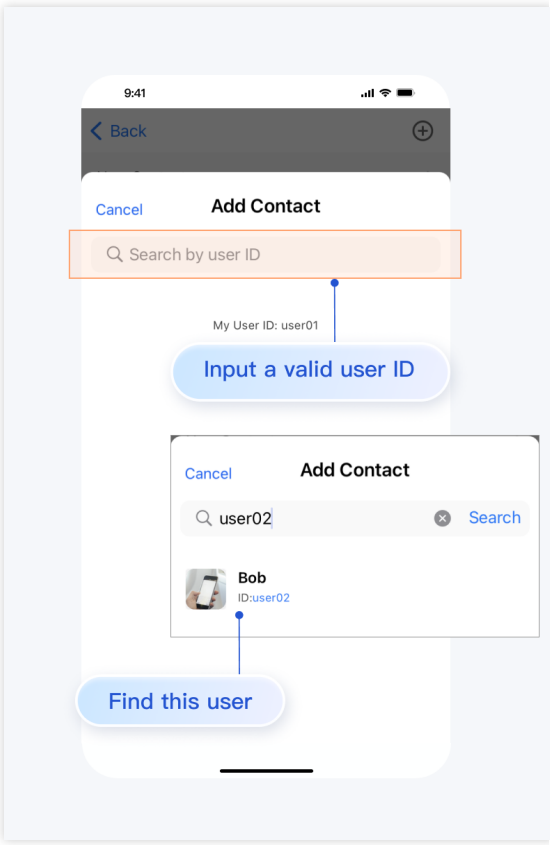
步骤说明

我们推荐您直接在 `TUKit` 中操作，手动添加联系人，步骤如下：
1. 按照 构建联系人界面 中的介绍，将联系人界面展示出来。
2. 点击该界面右上角的 `+` 按钮，在子菜单中选择 `Add to Contacts` 。
3. 输入有效的 userID，搜索出用户。您可以去控制台 `Account Managerment` 页面获取有效的 userID。页面路径：Applications > Your App > Chat > Users > Account Management。
4. 添加用户为联系人。

效果如下图所示：

点击 `Add to Contacts`	搜索用户





一般情况下，发送添加请求后，对方会默认同意。但如果对方账号设置过 `Friend Request Verification`，可能会有不同表现。该选项路径为：

Applications > `Your App` > Chat > Users > Account Management > `Choose an account` > Edit > Friend Request Verification。

`Friend Request Verification` 取值如下：

Friend Request Verification 取值	含义
Accept all friend requests	默认值，接受所有的好友请求
Manually accept or reject friend requests	需被请求方手动操作，同意或拒绝好友请求
Reject all friend requests	拒绝所有好友请求

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

创建群组

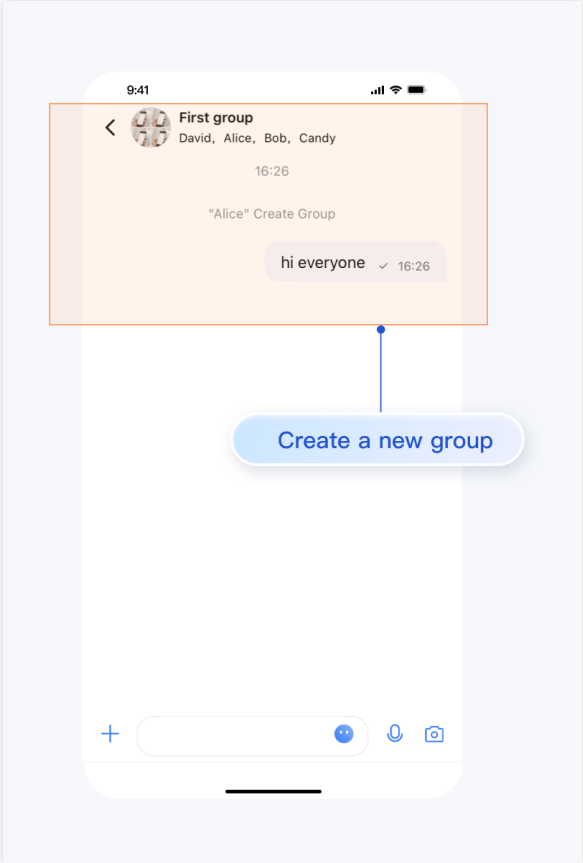
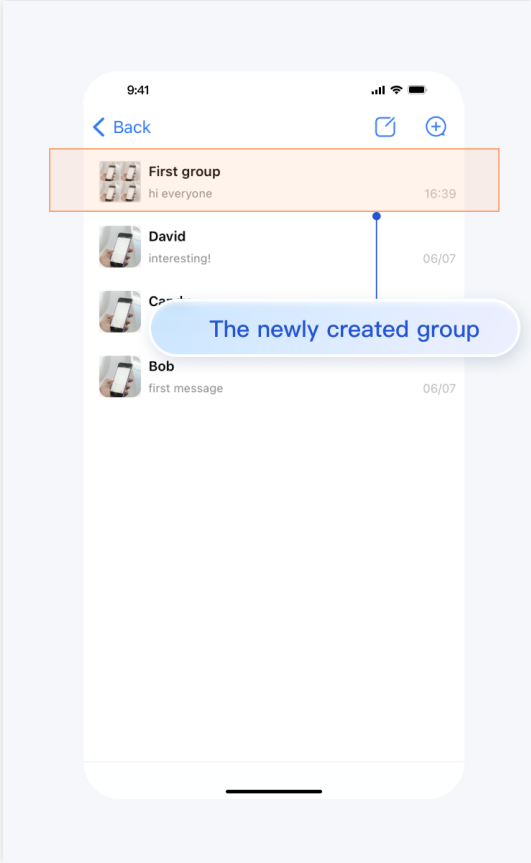
Android

最近更新时间：2024-06-24 16:42:20

本文会引导您在 TUIKit 上创建群组。

效果展示

创建群聊完成后，您可以开始在群组中发送消息，进行互动了。如果此时您退回到会话列表，会在列表中发现刚才创建的群聊：

创建群聊成功	会话列表更新
	

开发环境要求

Android Studio-Giraffe

Gradle-7.2

Android Gradle Plugin Version-7.0.0

kotlin-gradle-plugin-1.5.31

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：

1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：

1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。

如果您已经完成，请继续阅读下文。

创建群组

在 `TUIKit` 上手动创建群组有两个前提：

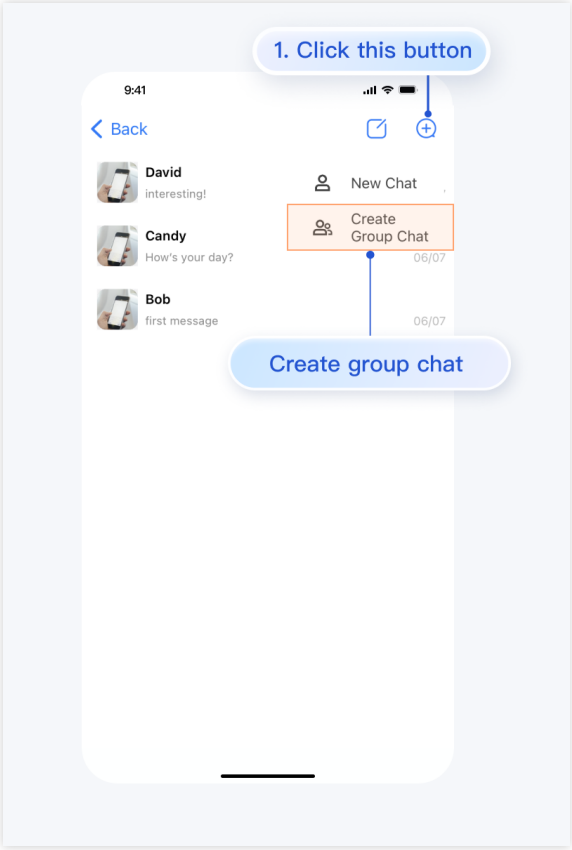
1. 加载出会话列表。请参考文档：构建会话列表。
2. 添加过一些联系人。请参考文档：添加联系人。

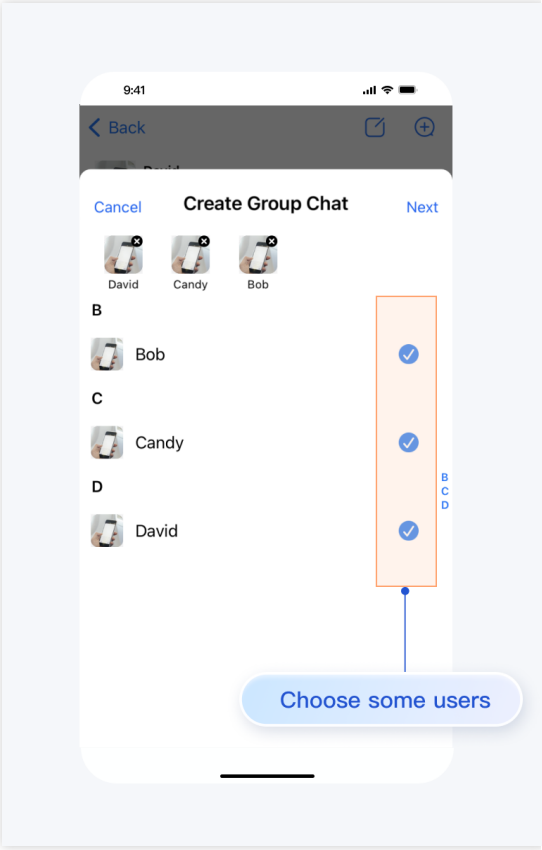
接下来还需要 3 步：

1. 在加载出的会话列表界面，点击右上方的 `+` 号，弹出子菜单，选择 `Create Group Chat` 。
2. 选择若干群组成员。这些成员是您添加的联系人。如果您之前没有添加过任何联系人，此界面将无人可选。
3. 设置群聊的名称、类型、头像等。

效果如下图所示：

单击 Create Group Chat	选择群组成员





更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

版权所有：腾讯云计算（北京）有限责任公司

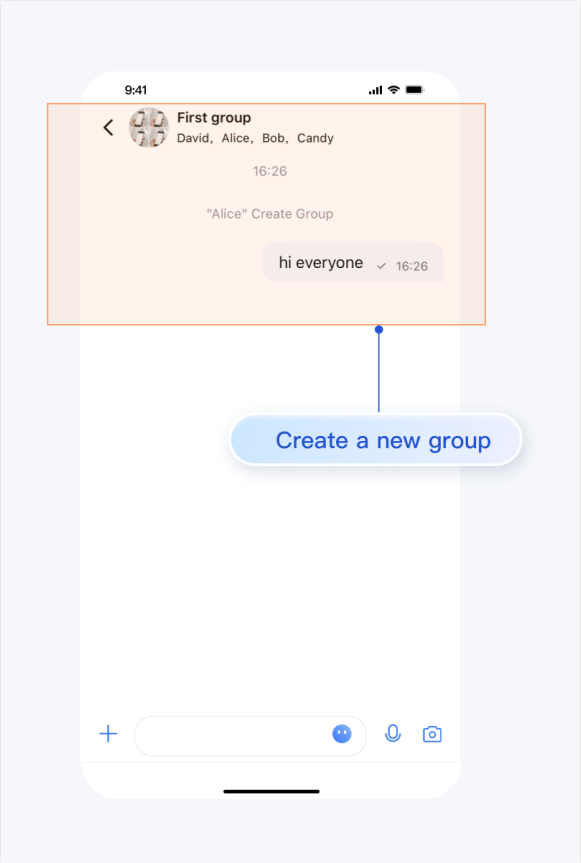
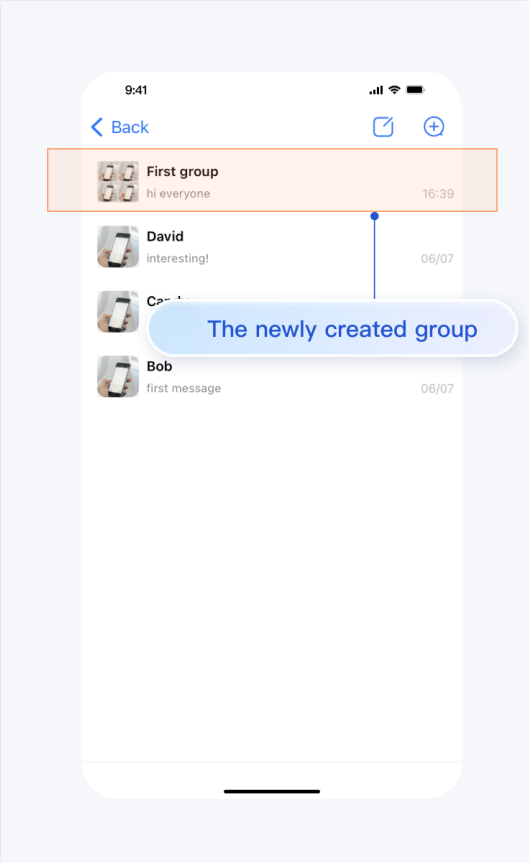
iOS

最近更新时间：2024-06-24 16:43:40

本文会引导您在 TUIKit 上创建群组。

效果展示

创建群聊完成后，您可以开始在群组中发送消息，进行互动了。如果此时您退回到会话列表，会在列表中发现刚才创建的群聊：

创建群聊成功	会话列表更新
	

开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUIKit` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

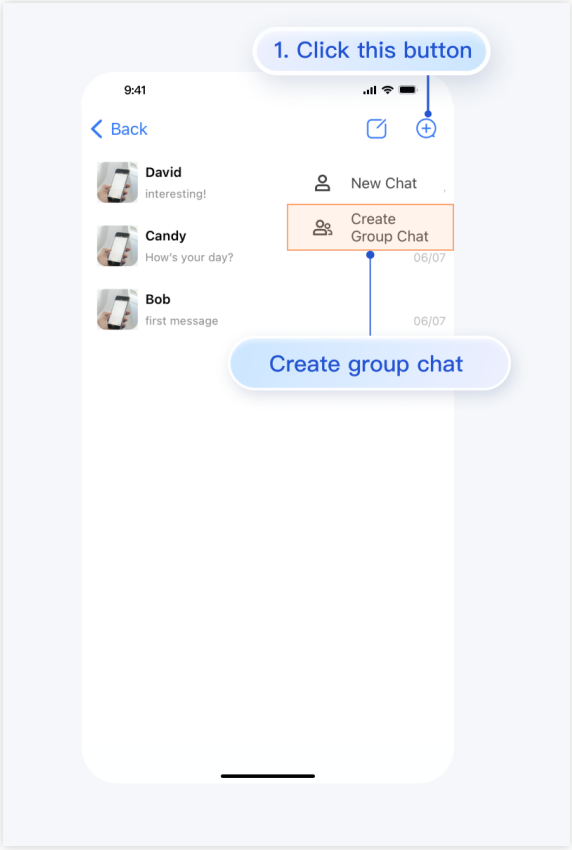
创建群组

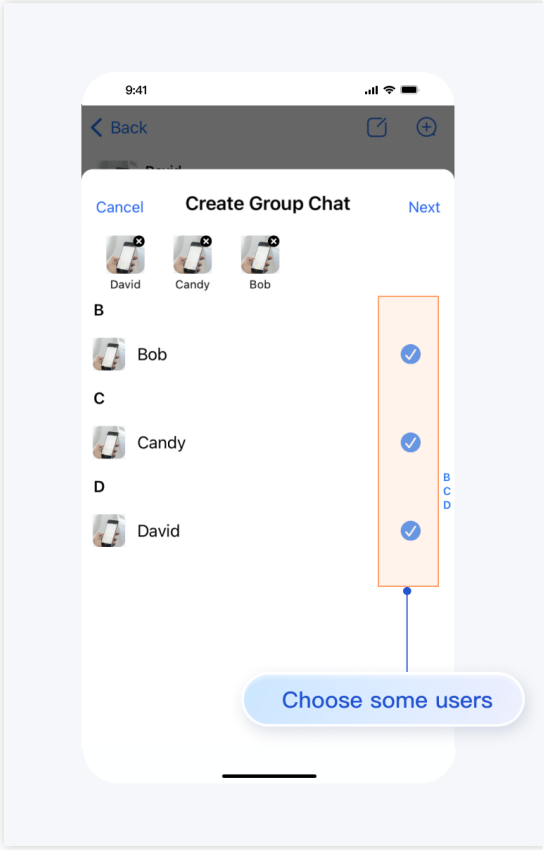
在 `TUIKit` 上手动创建群组有两个前提：
1. 加载出会话列表。请参考文档：构建会话列表。
2. 添加过一些联系人。请参考文档：添加联系人。

接下来还需要 3 步：
1. 在加载出的会话列表界面，点击右上方的 `+` 号，弹出子菜单，选择 `Create Group Chat` 。
2. 选择若干群组成员。这些成员是您添加的联系人。如果您之前没有添加过任何联系人，此界面将无人可选。
3. 设置群聊的名称、类型、头像等。

效果如下图所示：

点击 `Create Group Chat`	选择群组成员





更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

版权所有：腾讯云计算（北京）有限责任公司

音视频通话功能

Android

最近更新时间：2024-06-24 16:44:45

本文会引导您构建音视频通话功能。

效果展示

视频通话效果如下图所示：

语音通话效果如下图所示：

开发环境要求

Android Studio-Giraffe

Gradle-7.2

Android Gradle Plugin Version-7.0.0

kotlin-gradle-plugin-1.5.31

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：

1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUICallKit` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

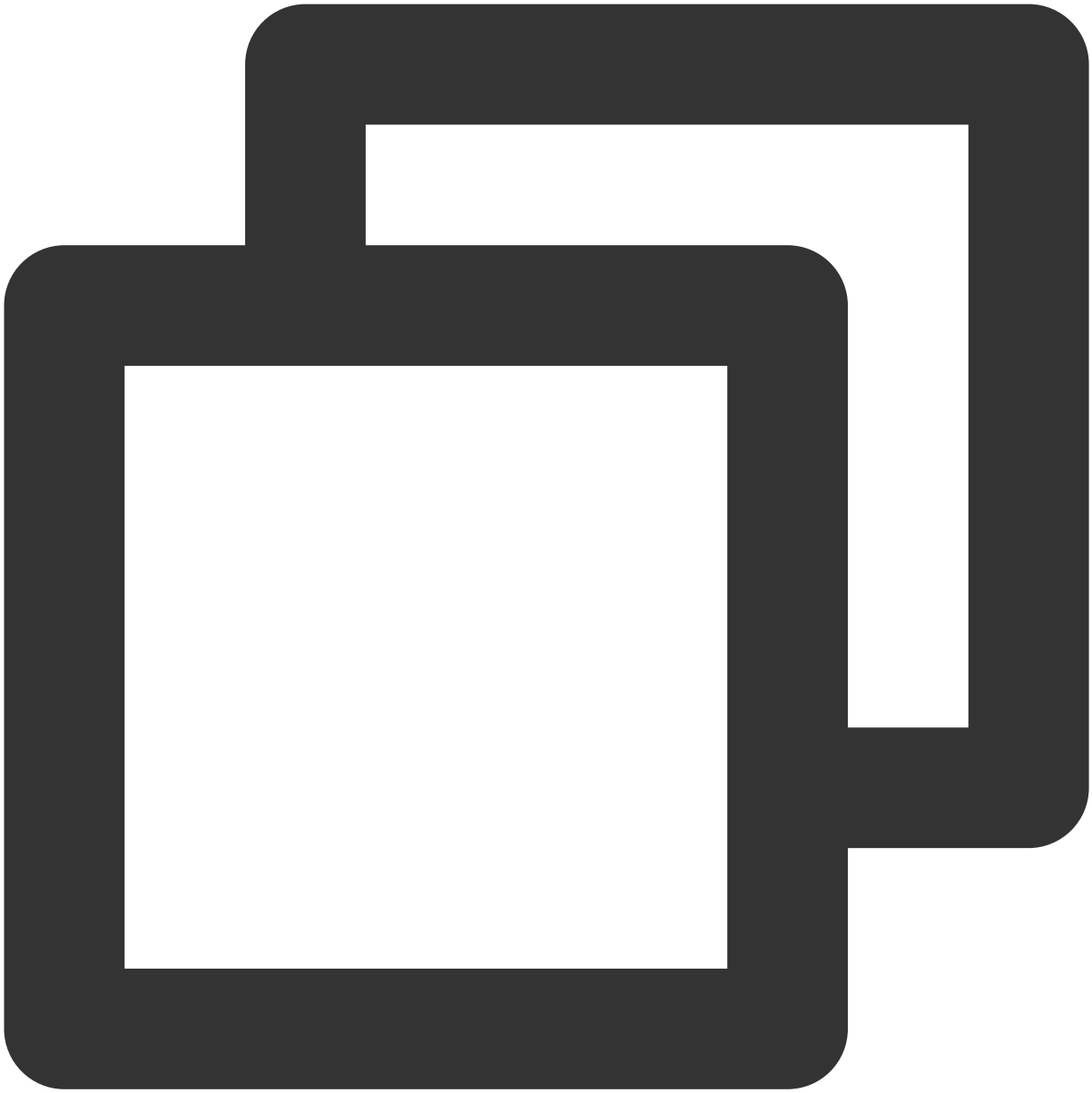
注意：

1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

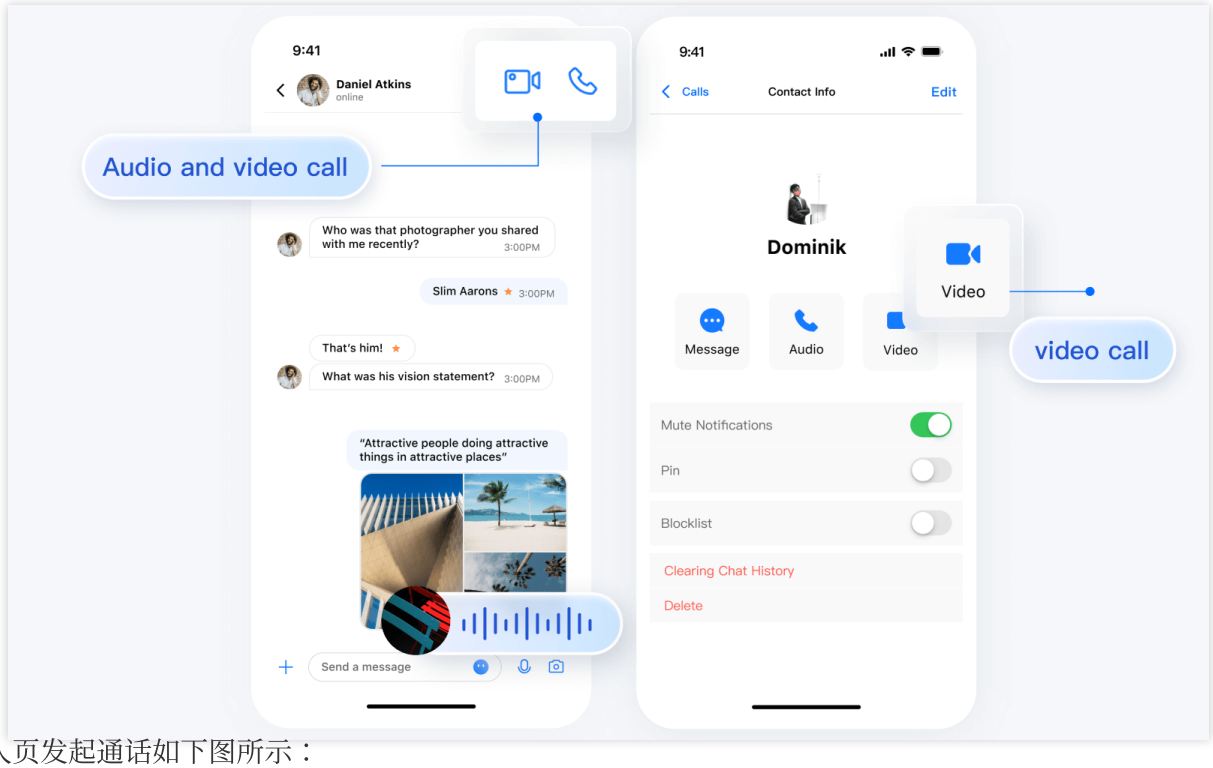
集成步骤

1. 登录 Chat 控制台 开通音视频服务。具体步骤可参考文档：开通服务。
2. 在弹出的开通实时音视频 `TRTC` 服务对话框中，单击“确认”，系统将为您在 实时音视频控制台 创建一个与当前 Chat 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。
3. 集成 `TUICallKit` 组件。在 App 的 build.gradle 文件中添加对 `TUICallKit` 的依赖：

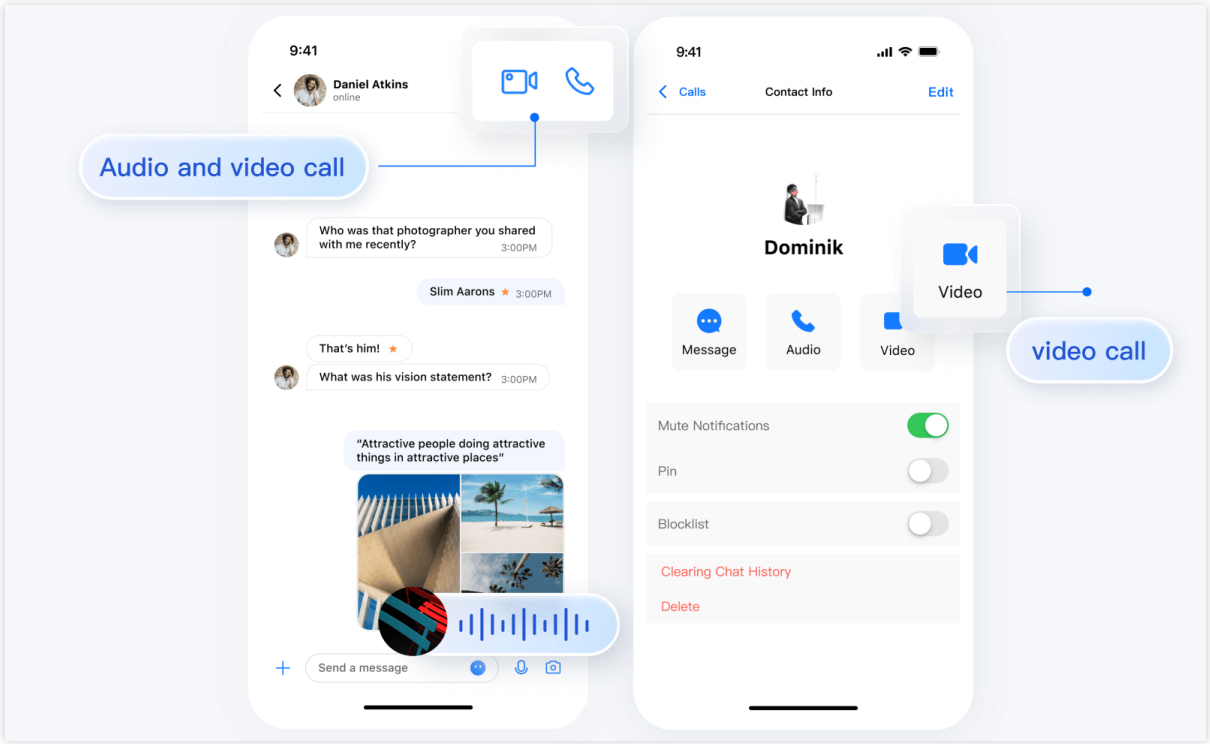


```
api project(':tuicallkit')
```

集成 `TUICallKit` 组件后，聊天界面和联系人资料界面默认会出现 “视频通话” 和 “语音通话” 两个按钮，当用户点击按钮时，`TUICallKit` 会自动展示通话邀请 UI，并给对方发起通话邀请请求。
当用户在线收到通话邀请时，`TUICallKit` 会自动展示通话接收 UI，用户可以选择同意或者拒绝通话。
当用户离线收到通话邀请时，如需唤起 App 通话，就要使用到离线推送能力。
消息页发起通话如下图所示：



联系人页发起通话如下图所示：



添加离线推送

详情可以参见文档：离线推送。配置完成后，当单击接收到的音视频通话离线推送通知时，`TUICallKit` 会自动拉起音视频通话邀请界面。

添加 AI 降噪

集成 `TUIChat` 和 `TUICallkit` 的组件后，在聊天界面发送语音消息时，即可录制带 AI 降噪和自动增益的语音消息。下面是使用两台手机同时录制的语音消息对比：

注意：

1. 该功能需要购买音视频通话能力进阶版及以上套餐。套餐过期后，录制语音消息会切换到系统 API 进行录音。
2. 仅 SDK 7.0 及以上版本支持。

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

iOS

最近更新时间：2024-06-24 16:46:11

本文会引导您构建音视频通话功能。

效果展示

视频通话效果如下图所示：

语音通话效果如下图所示：

开发环境要求

Xcode 10 及以上
iOS 9.0 及以上

前置条件

在构建界面之前，请确保您已经完成了以下 4 件事：
1. 在控制台创建了一个应用。
2. 在控制台创建了一些用户账号。
3. 集成了 `TUICallKit` 。
4. 调用 `TUILogin` 的 `login` 接口登录组件。

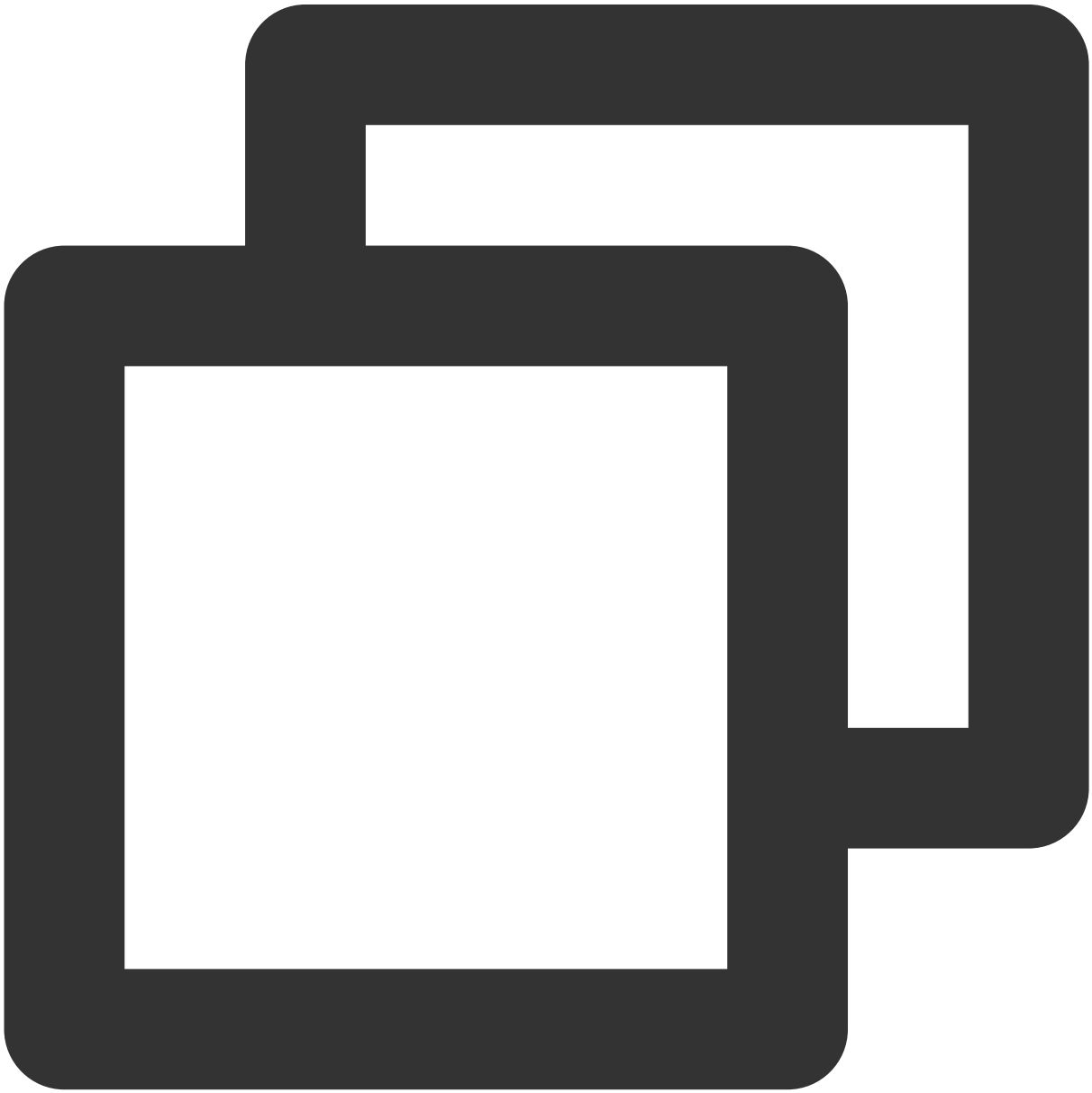
注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。

如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。

如果您已经完成，请继续阅读下文。

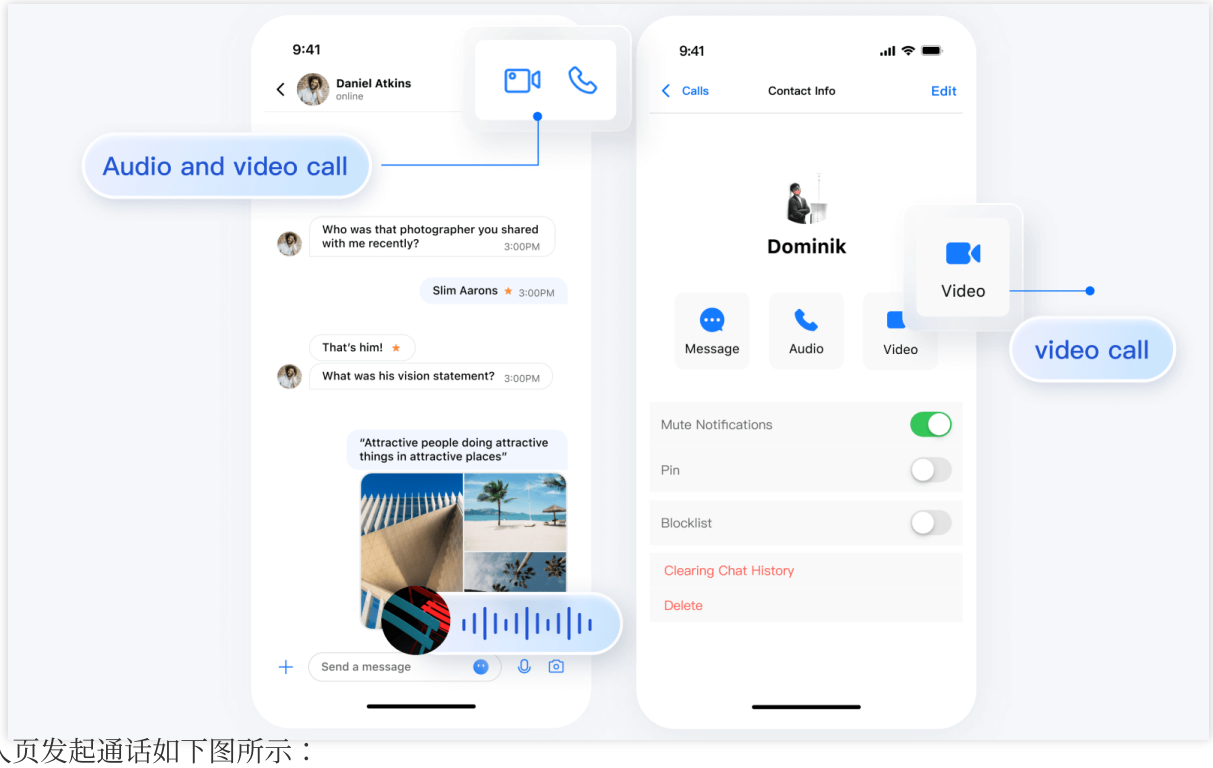
集成步骤

1. 登录 Chat 控制台 开通音视频服务。具体步骤可参考文档：开通服务。
2. 在弹出的开通实时音视频 `TRTC` 服务对话框中，单击“确认”，系统将为您在 实时音视频控制台 创建一个与当前 Chat 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。
3. 集成 `TUICallKit` 组件。在 podfile 文件中添加以下内容：

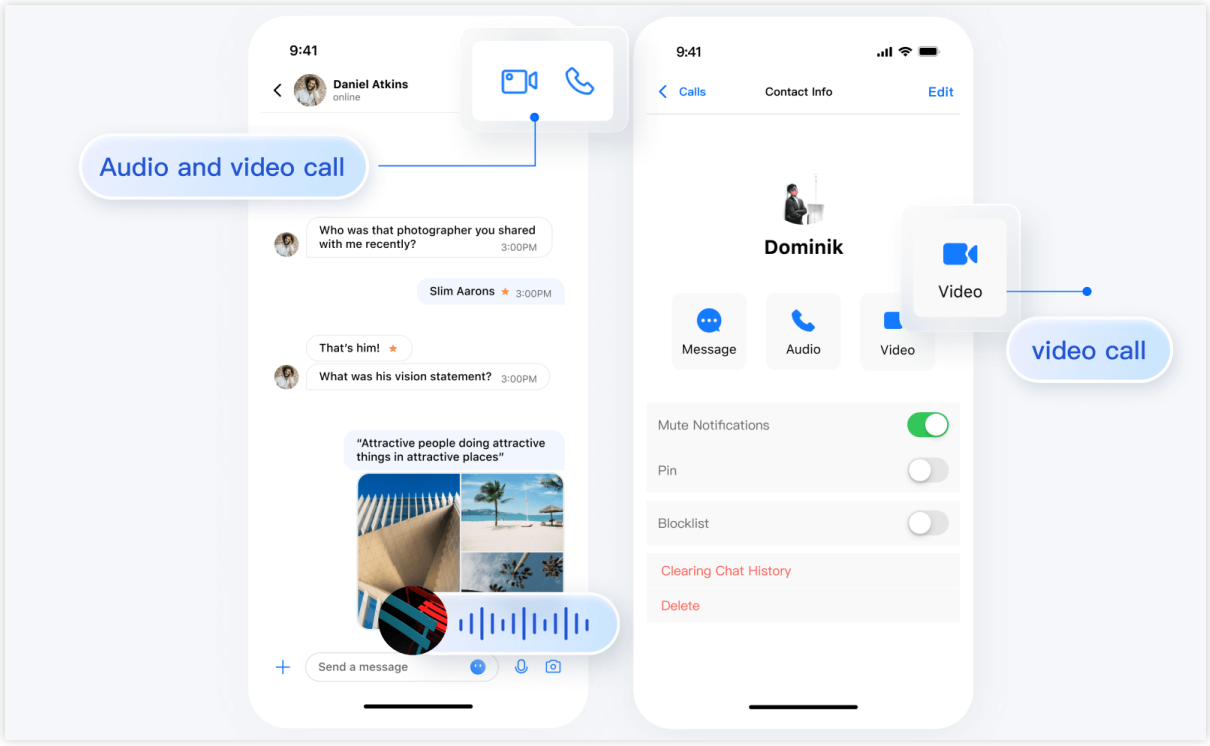


```
// Integrate the TUICallKit component
pod 'TUICallKit'
```

集成 `TUICallKit` 组件后，聊天界面和联系人资料界面默认会出现“视频通话”和“语音通话”两个按钮，当用户点击按钮时，`TUICallKit` 会自动展示通话邀请 UI，并给对方发起通话邀请请求。
当用户在线收到通话邀请时，`TUICallKit` 会自动展示通话接收 UI，用户可以选择同意或者拒绝通话。
当用户离线收到通话邀请时，如需唤起 App 通话，就要使用到离线推送能力。
消息页发起通话如下图所示：



联系人页发起通话如下图所示：



添加离线推送

详情可以参见文档：离线推送。配置完成后，当单击接收到的音视频通话离线推送通知时，`TUICallKit` 会自动拉起音视频通话邀请界面。

添加 AI 降噪

集成 `TUIChat` 和 `TUICallkit` 的组件后，在聊天界面发送语音消息时，即可录制带 AI 降噪和自动增益的语音消息。下面是使用两台手机同时录制的语音消息对比：

注意：

1. 该功能需要购买音视频通话能力进阶版及以上套餐。套餐过期后，录制语音消息会切换到系统 API 进行录音。
2. 仅 SDK 7.0 及以上版本支持。

更多实践

您可以本地 运行 TUIKitDemo 源码，探索更多的界面实现。

联系我们

如果您对本文有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

修改界面主题

Android

最近更新时间：2024-03-15 18:08:31

概述

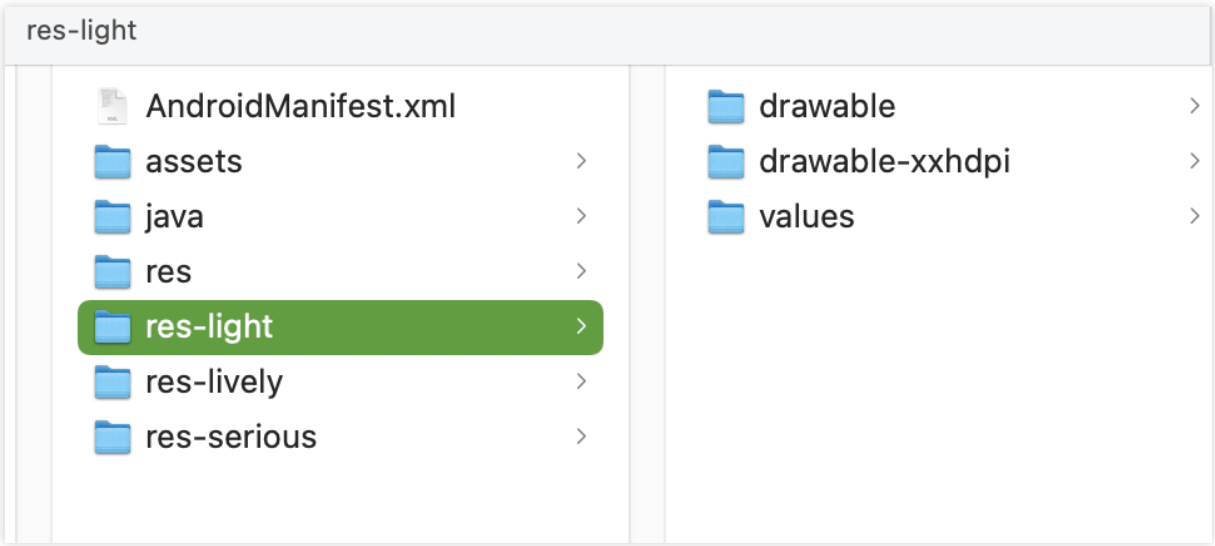
TUI 组件默认内置了：轻量、活泼、深沉共三套主题。您可以任意切换或者修改内置主题，也可以按需新增主题。

注意：

仅经典版 UI 支持切换、修改和新增主题，简约版 UI 不支持。

主题资源

您可以在任一 TUI 组件内部的 `res` 文件夹下看到该组件所支持的主题资源。以 TUIChat 组件为例，您可以在 `TUIChat/tuichat/src/main/` 下看到资源文件夹；`res-light` 、`res-serious` 和 `res-lively` 文件夹中分别为 TUIChat 组件内置的轻量版、深沉版和活泼版主题资源，`res` 文件夹中为通用资源。

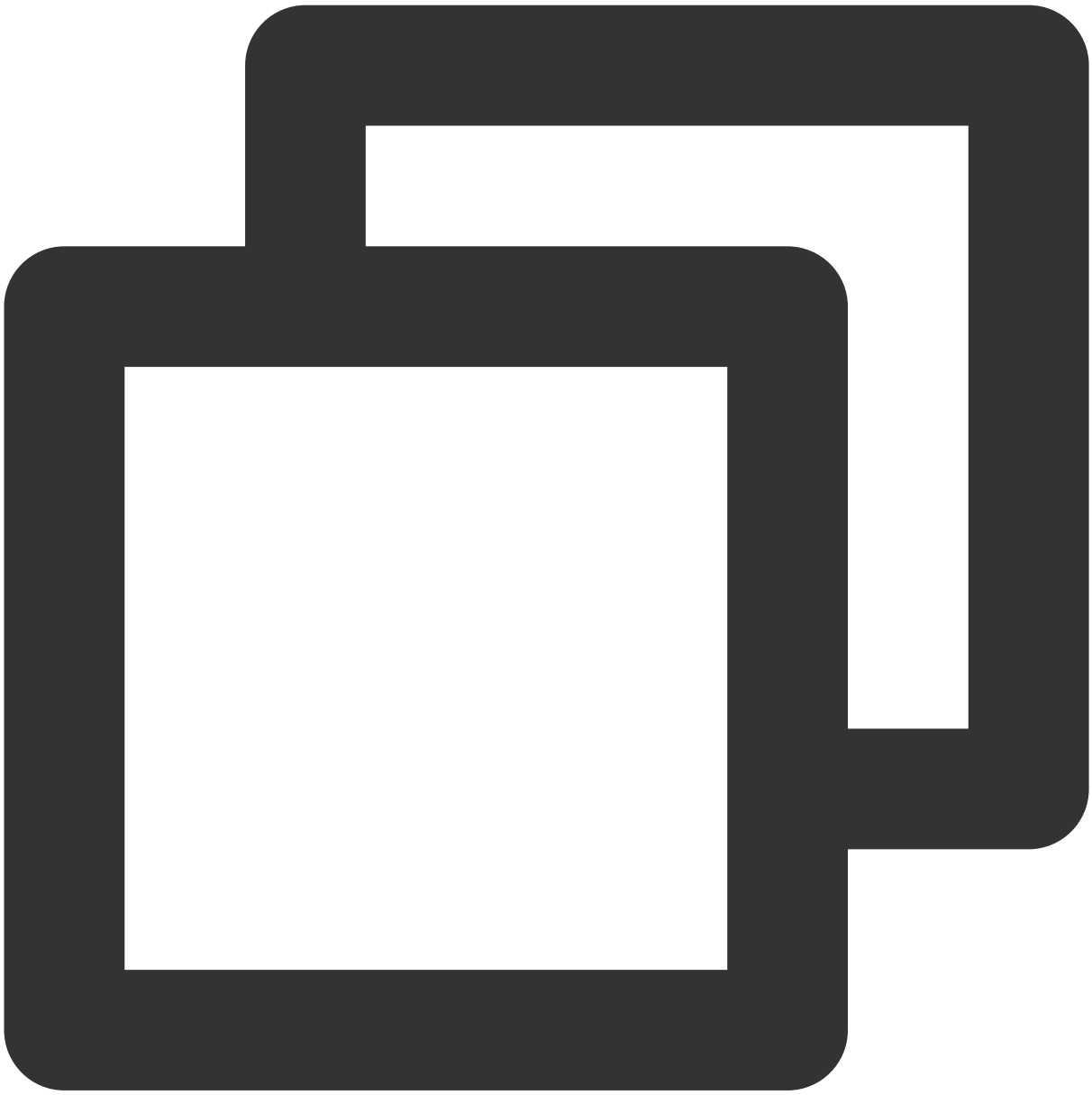


主题资源文件夹的目录结构与通用资源的目录结构一致。

应用主题

`TUIKit` 默认使用轻量版主题。当您需要对 TUI 组件以及您的 App 主工程设置主题时，可以调用 `TUIThemeManager` 的 `changeTheme` 方法来设置当前主题。

您可以参考 `TUIKitDemo` 的 ThemeSelectActivity.java 文件中的代码。
也可以使用如下方法切换主题：



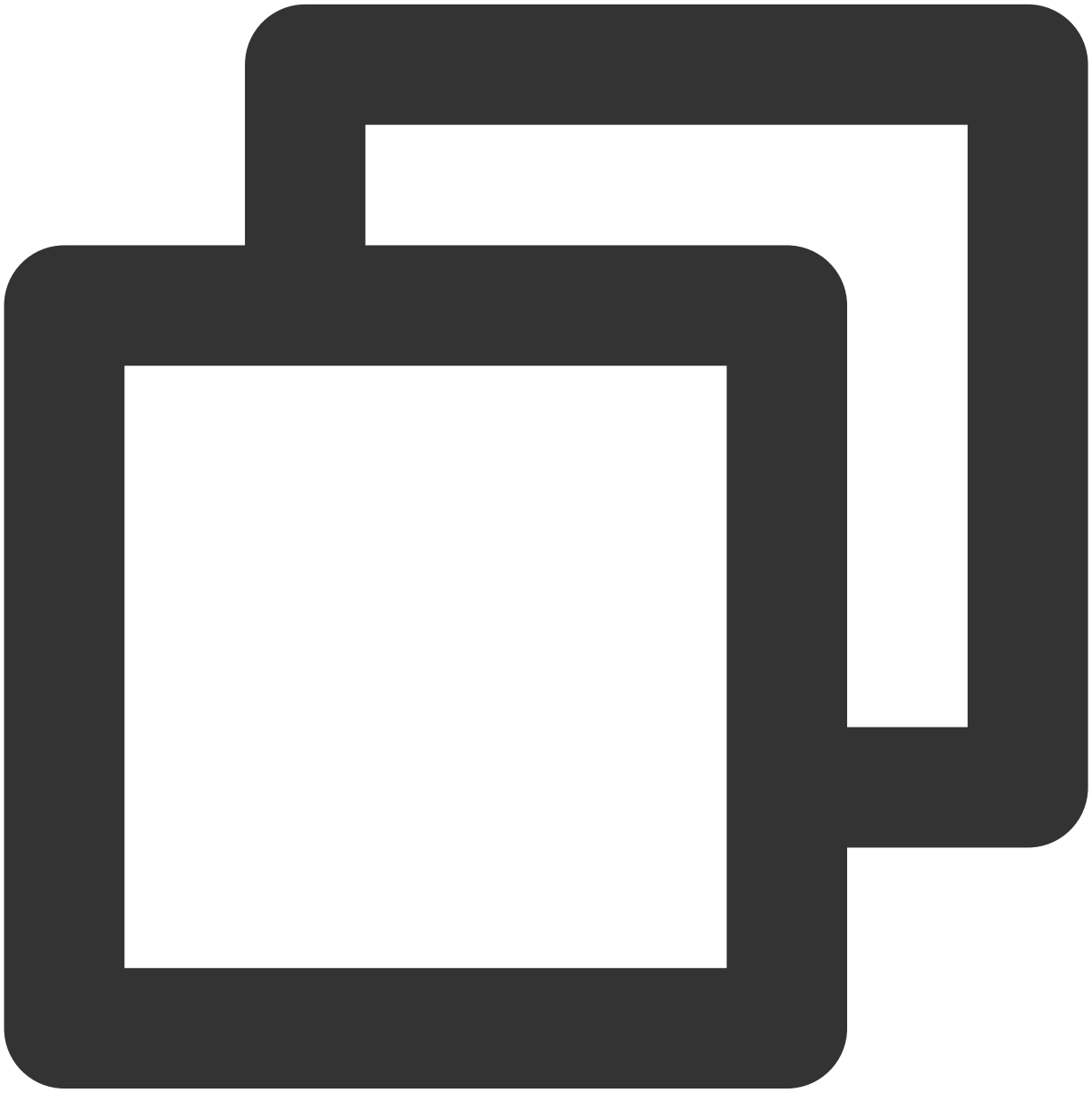
```
// 轻量版 themeID 为 0，  活泼版 themeID 为 1，  深沉版 themeID 为 2
TUIThemeManager.getInstance().changeTheme(context, themeID);
System.exit(0);
Intent intent = context.getPackageManager().getLaunchIntentForPackage(context.getPa
context.startActivity(intent);
```

获取主题内资源

说明

主题属性皆定义在各组件的 `src/main/res/values/tui_theme_attrs.xml` 文件中，属性名不可重复。

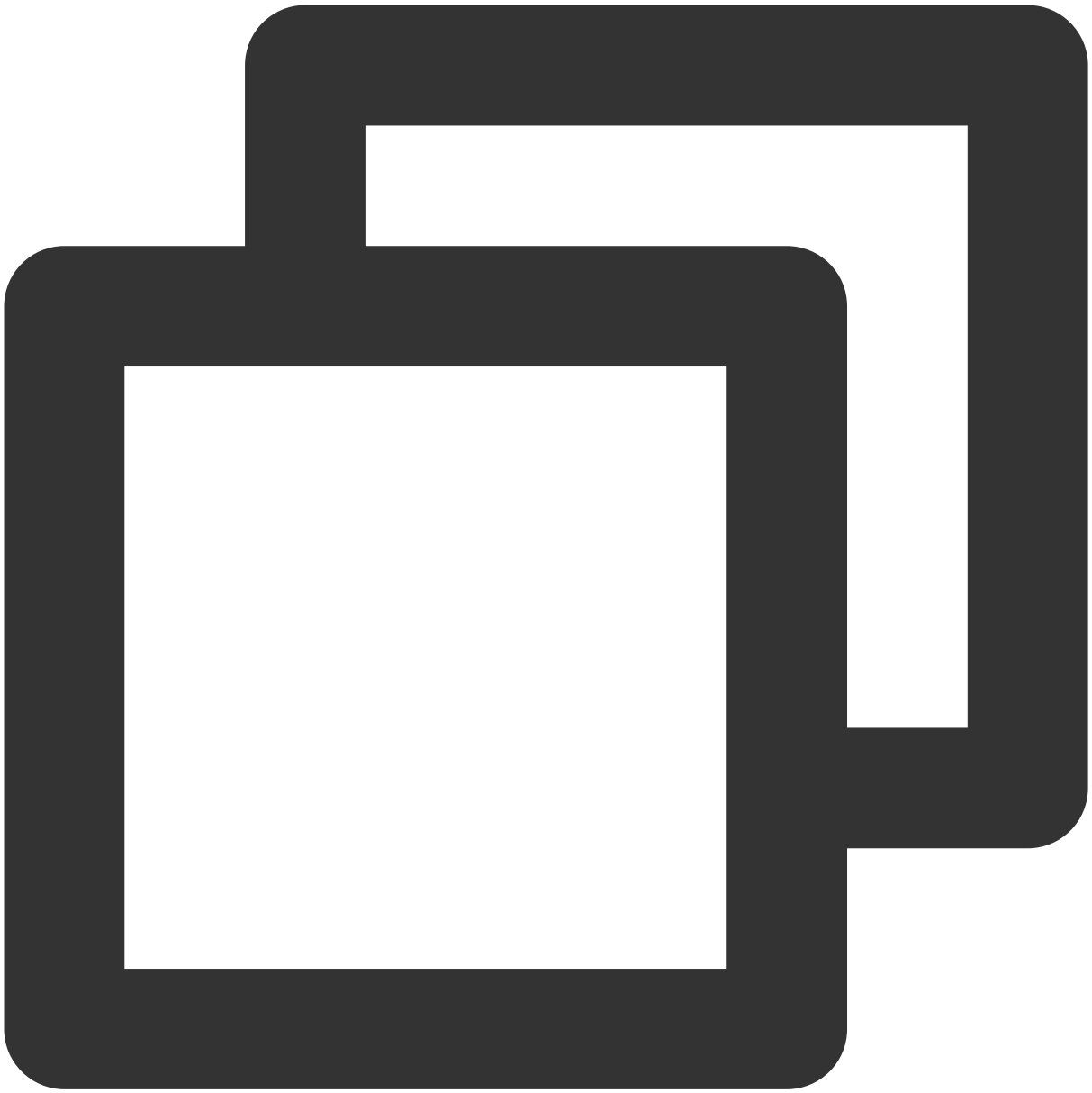
应用主题成功之后, 在 Java 代码中，可以调用 TUIThemeManager.getAttrResId(context, attrID) 方法根据主题属性来获取资源 ID，然后再根据获取到的资源 ID 获取真正的资源，例如：



```
mArrowImageView.setBackgroundResource(TUIThemeManager.getAttrResId(getContext(), R.

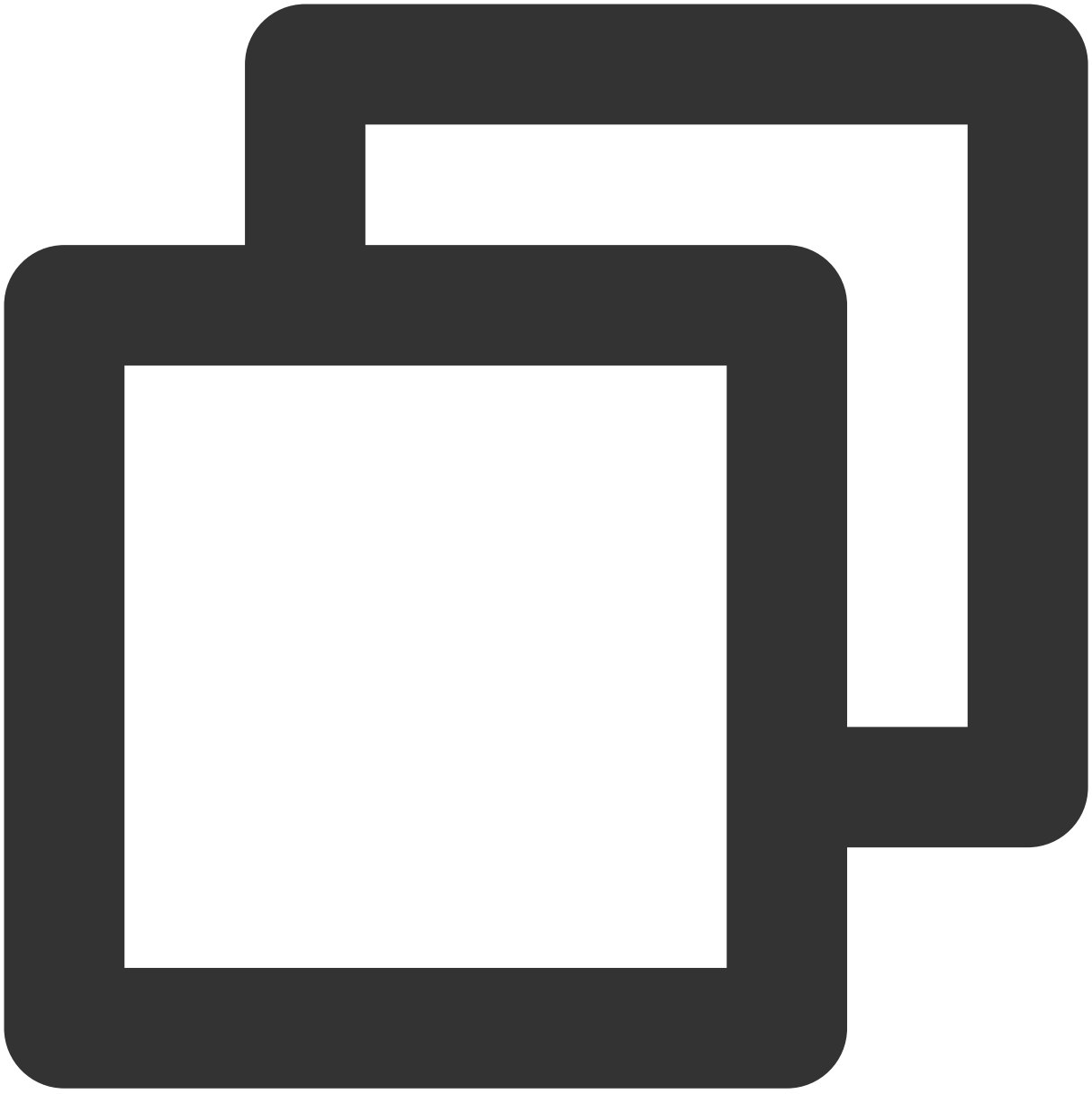
replyContentTv.setTextColor(resources.getColor(TUIThemeManager.getAttrResId(context
```

在 XML 资源文件中，可以使用 ?attr/** 的方式，根据主题属性来使用当前主题下的资源，例如：



```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="ring"
    android:innerRadius="22.5dp"
    android:thickness="1.5dp"
    android:useLevel="false">

    <solid android:color="?attr/core_primary_color" />
</shape>
```



```
<ImageView
    android:id="@+id/demo_login_theme_arrow"
    android:layout_width="9.6dp"
    android:layout_height="9.6dp"
    android:layout_gravity="center"
    android:background="?attr/demo_login_language_arrow" />
```

注意

上面两种方法只能获取当前已经应用成功了的主题的资源 ID，无法获取未应用的主题下的资源 ID。

修改内置主题

TUI 组件目前可以修改内置的主题，按照下列步骤，可对内置主题的颜色、字体、图片等资源做自定义变更。

1. 找到要修改的主题下具体的资源；
2. 替换或者修改资源；
3. 切换到对应主题下，查看效果。

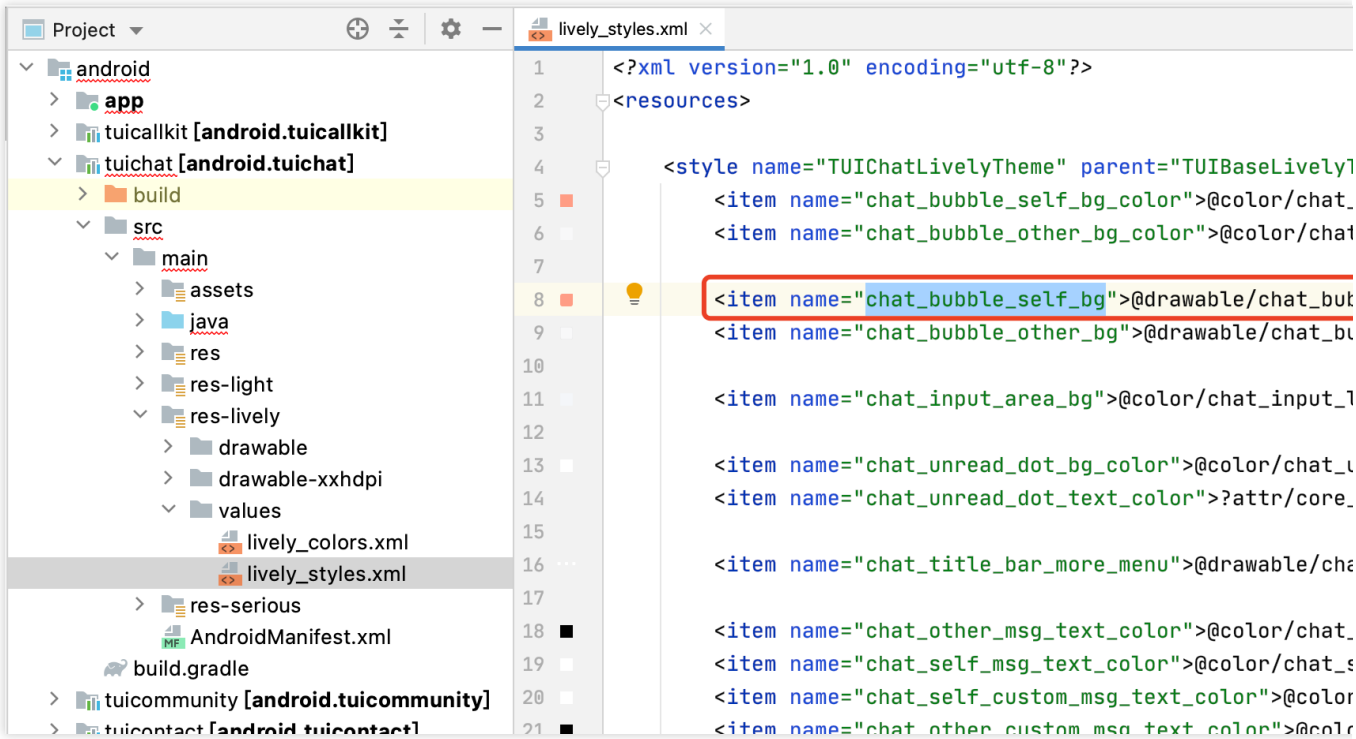
例如，`TUIChat` 组件中自己发出的文本消息的气泡背景色，在不同主题下有不一样的颜色。

该背景色在内置的 “活泼” 主题下的色值为 #FF9D85，现在想要修改成 #EA286C，您只需要按照如下步骤操作即可：

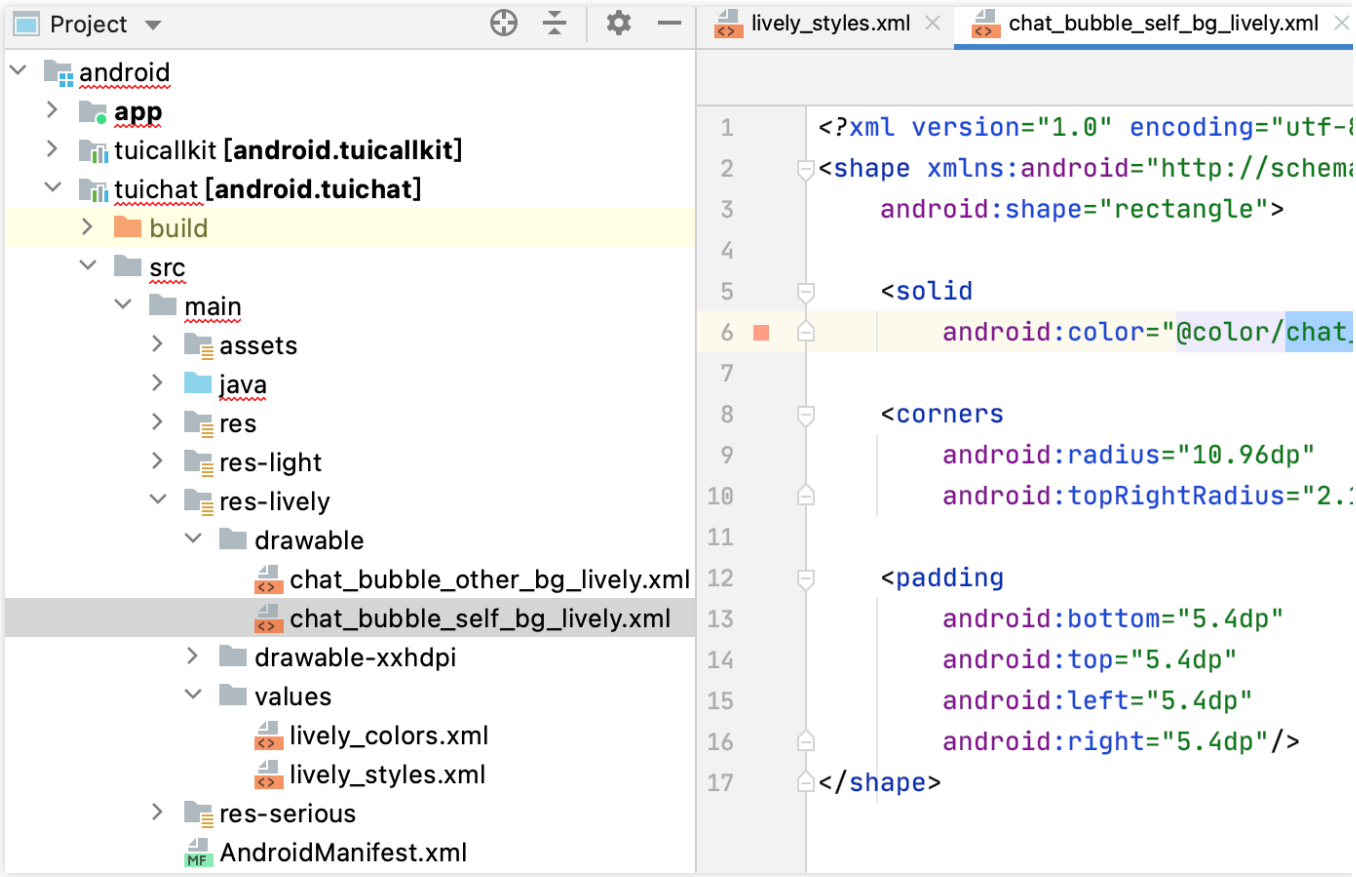
1. 从 `TUIChat` 源码中找到自己发送的文本消息的气泡使用的背景是 `R.attr.chat_bubble_self_bg` 属性：

```
if (msg.isSelf()) {
    if (properties.getRightBubble() != null && properties.getRightBubble().getConstantState() != nu
        msgArea.setBackground(properties.getRightBubble().getConstantState().newDrawable());
    } else {
        msgArea.setBackgroundResource(TUIThemeManager.getAttrResId(itemView.getContext(), R.attr.ch
    }
} else {
    if (properties.getLeftBubble() != null && properties.getLeftBubble().getConstantState() != null
        msgArea.setBackground(properties.getLeftBubble().getConstantState().newDrawable());
    } else {
        msgArea.setBackgroundResource(TUIThemeManager.getAttrResId(itemView.getContext(), R.attr.ch
    }
}
```

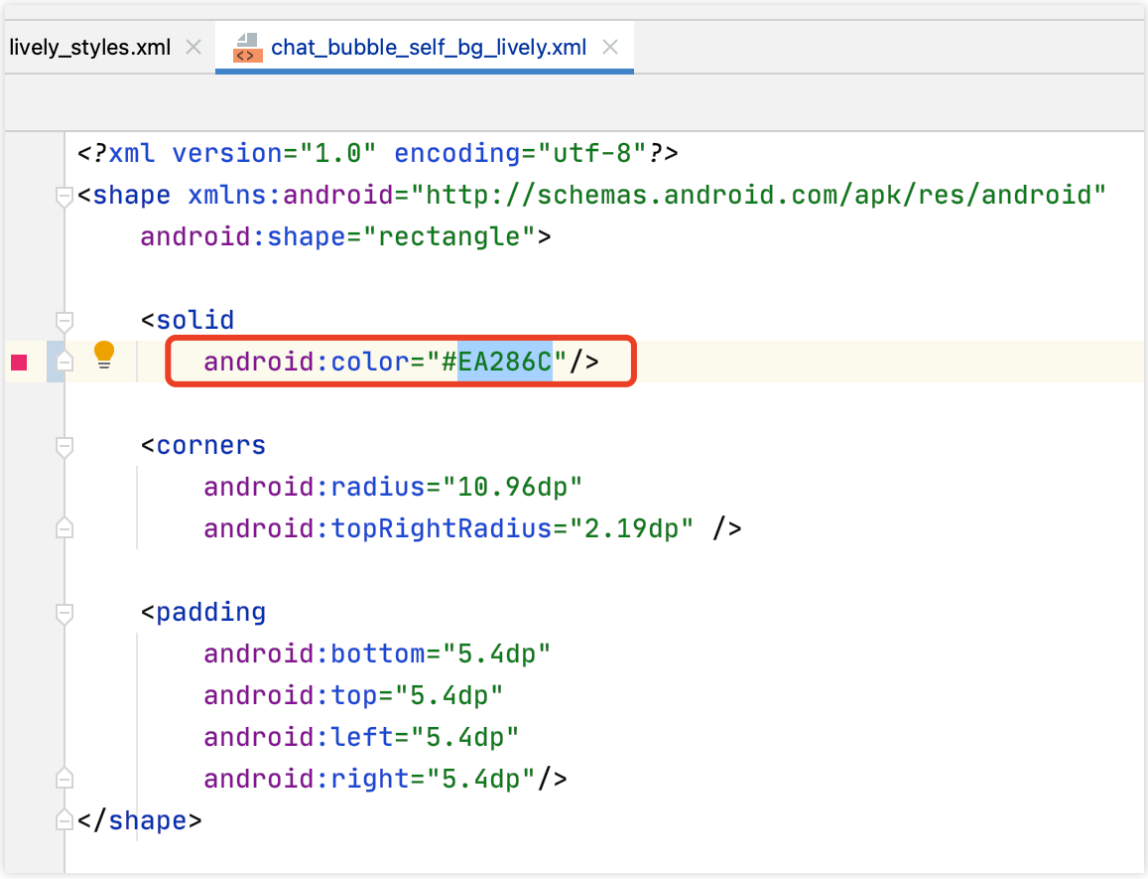
在 `tuichat/src/main/res-lively/values/lively_styles.xml` 文件中找到，`chat_bubble_self_bg` 属性对应的资源是 `@drawable/chat_bubble_self_bg_lively` ：



打开对应的资源文件，发现背景色是 `@color/chat_bubble_self_color_lively` ：



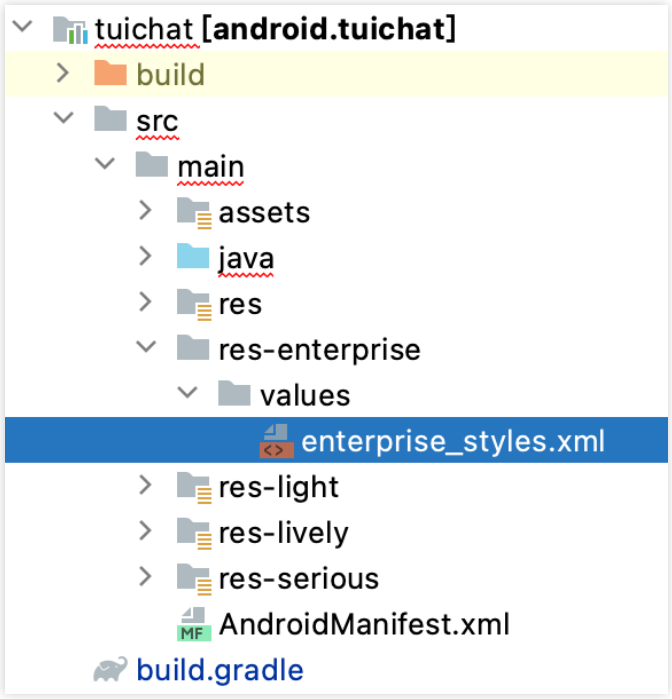
2. 上一步已经找到要替换的资源，将 `@drawable/chat_bubble_self_bg_lively` 资源中的 `@color/chat_bubble_self_color_lively` 的色值替换为 `#EA286C` ：



3. 保存文件，重新编译安装应用，切换主题为活泼版主题，即可看到效果。

新增主题

如果内置的 3 套主题无法满足您的需求，您可以自行按照如下步骤为组件新增一套全新的主题。
以添加一套 `商务版（Enterprise）` 主题为例：
1. 在每个组件中，与其他主题目录同级，在资源目录下新建主题目录 `res-enterprise` ：
`res-enterprise/values/` 目录下新建 `enterprise_styles.xml` 文件， `enterprise_styles.xml` 文件中存储主题属性与真正资源的映射。



注意

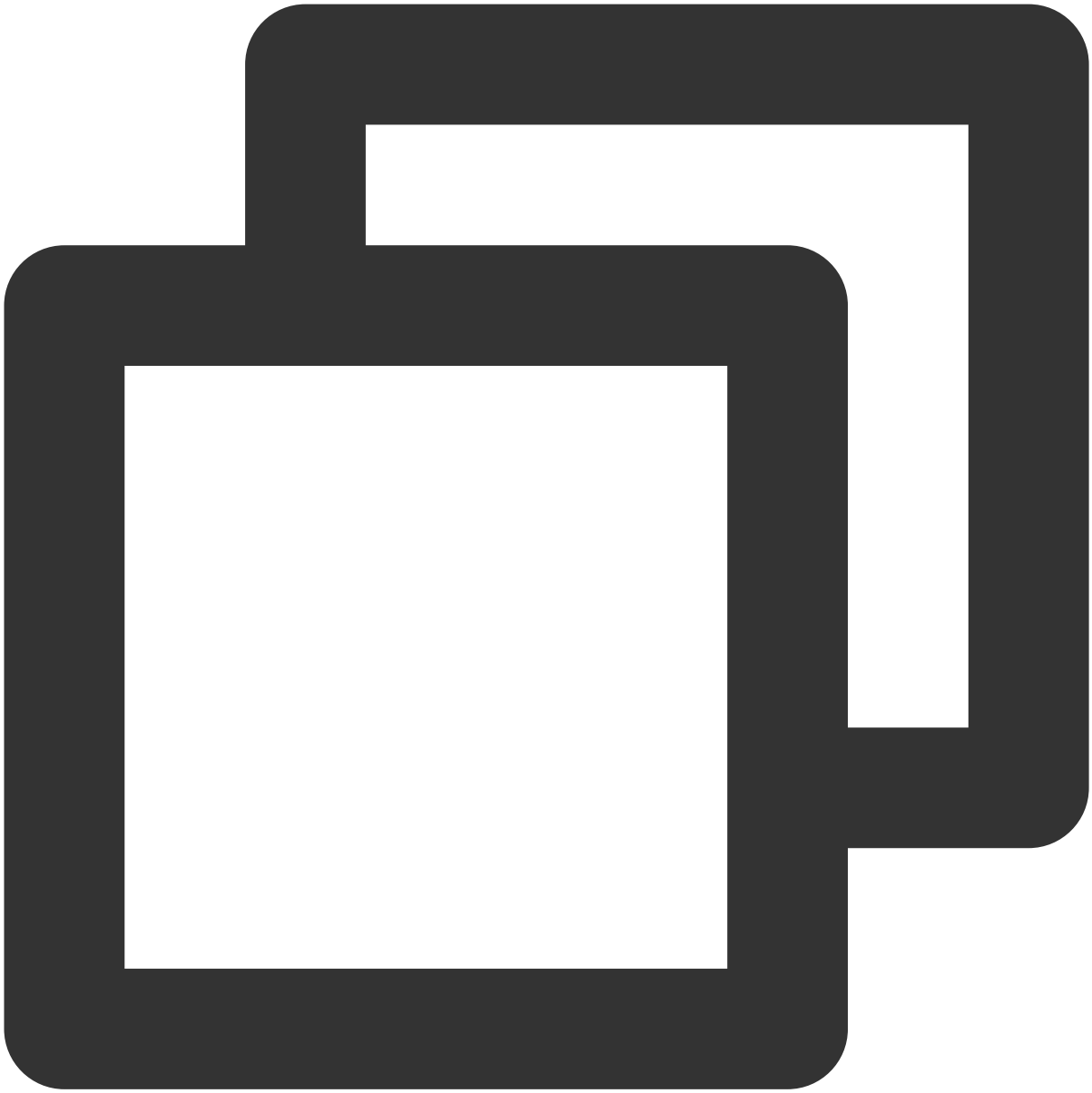
1、 `res-enterprise` 目录下必须包含所有要参与切换主题的资源，否则切换到 `商务版` 主题后应用会因为找不到资源而崩溃；

2、主题资源名不能跟系统资源名重复，也不可与已有资源名重复，否则会在编译期和运行期出现错误，因此要保证资源命名全局唯一。

2. 在 `enterprise_styles.xml` 文件中建立主题资源映射：

组件的 `src/main/res/values/tui_theme_attrs.xml` 文件中声明了需要参与切换主题的属性，这些属性需要在每个主题下都有对应的实现。

`src/main/res/values/enterprise_styles.xml` 文件中保存属性和资源的映射，例如：



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="TUIChatEnterpriseTheme" parent="TUIBaseEnterpriseTheme">
        <item name="chat_bubble_self_bg">@drawable/chat_bubble_self_bg_enterprise</
        <item name="chat_bubble_other_bg">@drawable/chat_bubble_other_bg_enterprise
        <item name="chat_input_area_bg">@color/chat_input_layout_bg_enterprise</ite
        <item name="chat_unread_dot_bg_color">@color/chat_unread_dot_bg_color_enter
        <item name="chat_unread_dot_text_color">?attr/core_primary_color</item>
        <item name="chat_title_bar_more_menu">@drawable/chat_title_bar_more_menu_en
        <item name="chat_other_msg_text_color">@color/chat_other_msg_text_color_ent
```

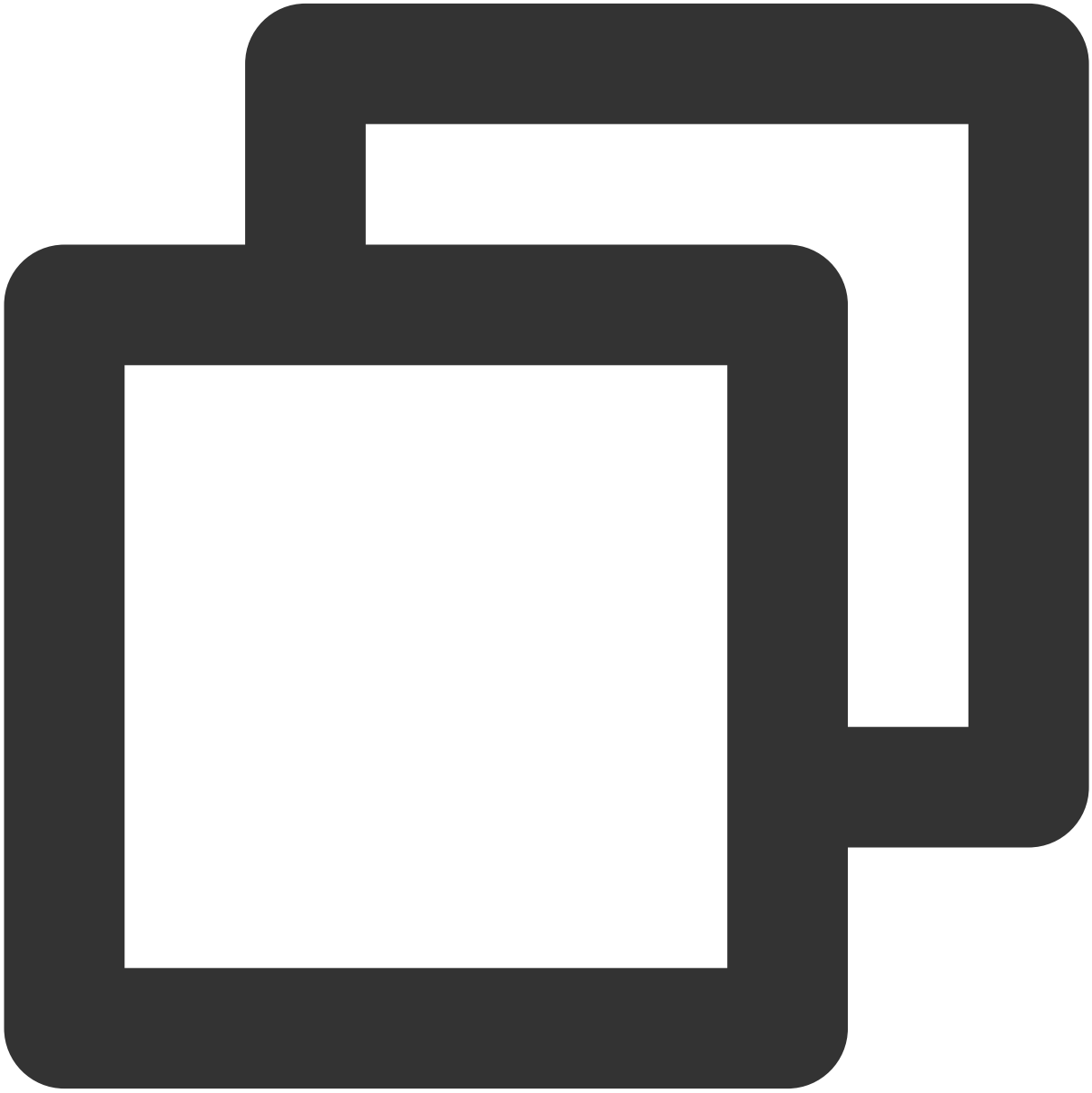
版权所有：腾讯云计算（北京）有限责任公司

```
        ...

    </style>
</resources>
```

3. 在组件的 `build.gradle` 文件中添加配置，指定资源目录：

指定资源目录参与 `App` 打包，每个组件的 `build.gradle` 文件中都需要添加编译资源目录：



```
android {
    ...
    // 主题资源文件夹
```

```
    sourceSets {
        main {
            res.srcDirs += "src/main/res-light"
            res.srcDirs += "src/main/res-lively"
            res.srcDirs += "src/main/res-serious"
            res.srcDirs += "src/main/res-enterprise"
        }
    }
}
```

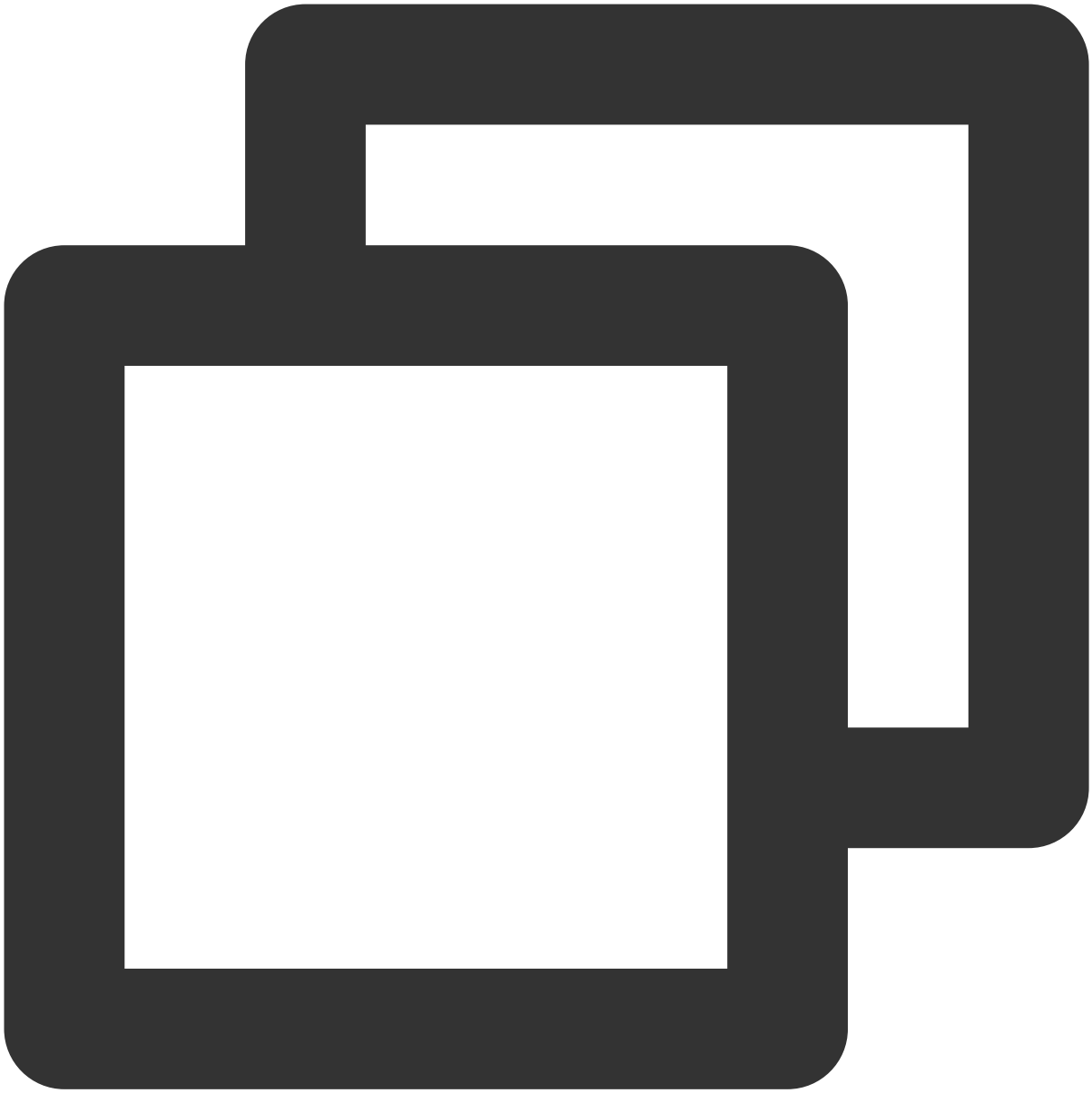
4. 应用启动时注册主题：

注册了 `Enterprise` 主题之后，才可以应用 `Enterprise` 主题。每个组件以及 `App` 的主题都要进行注册。

主题注册得越早越好，一般在 `Application` 启动时注册，这样 `Activity` 创建时就可以使用当前主题。

注意

`0-2` 分别为内置主题 ID，因此新增的主题 ID 必须大于等于 `3` 。

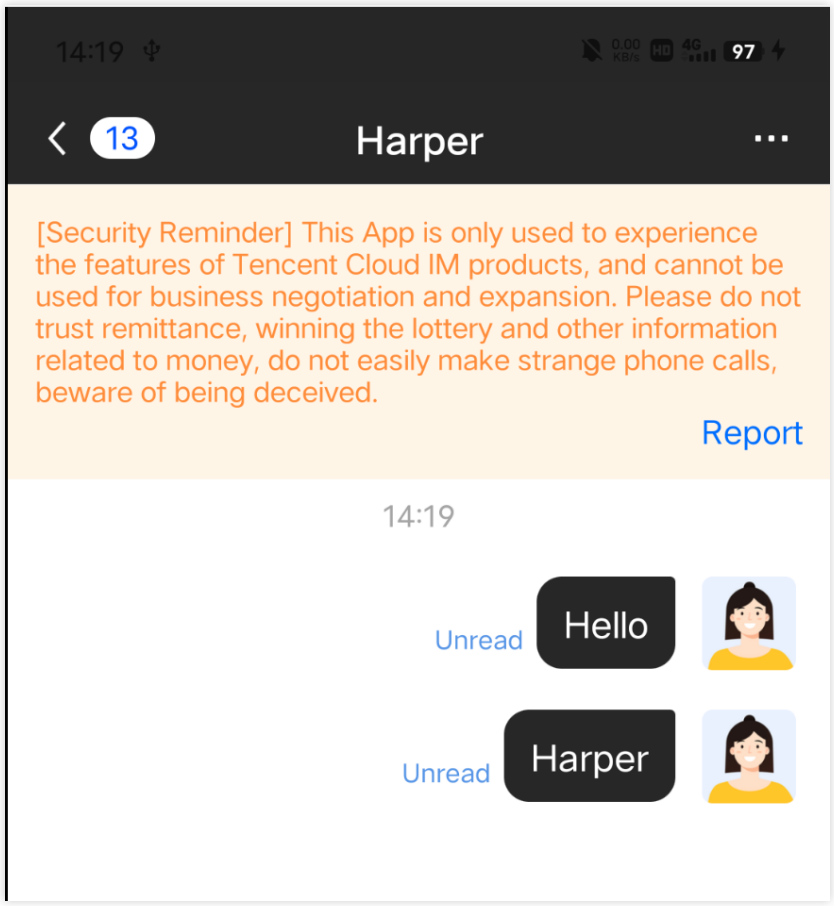


```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        int enterpriseThemeID = 3;
        TUIThemeManager.addTheme(enterpriseThemeID, R.style.DemoEnterpriseTheme);
        TUIThemeManager.addTheme(enterpriseThemeID, R.style.TUIChatEnterpriseTheme)
        TUIThemeManager.addTheme(enterpriseThemeID, R.style.TUIContactEnterpriseThe
        TUIThemeManager.addTheme(enterpriseThemeID, R.style.TUIGroupEnterpriseTheme
        // 切换主题
        TUIThemeManager.getInstance().changeTheme(this, enterpriseThemeID);
    }
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
```

5. 切换到新增的 `Enterprise` 主题，即可看到新增加的主题风格：



主题样式表

基础样式

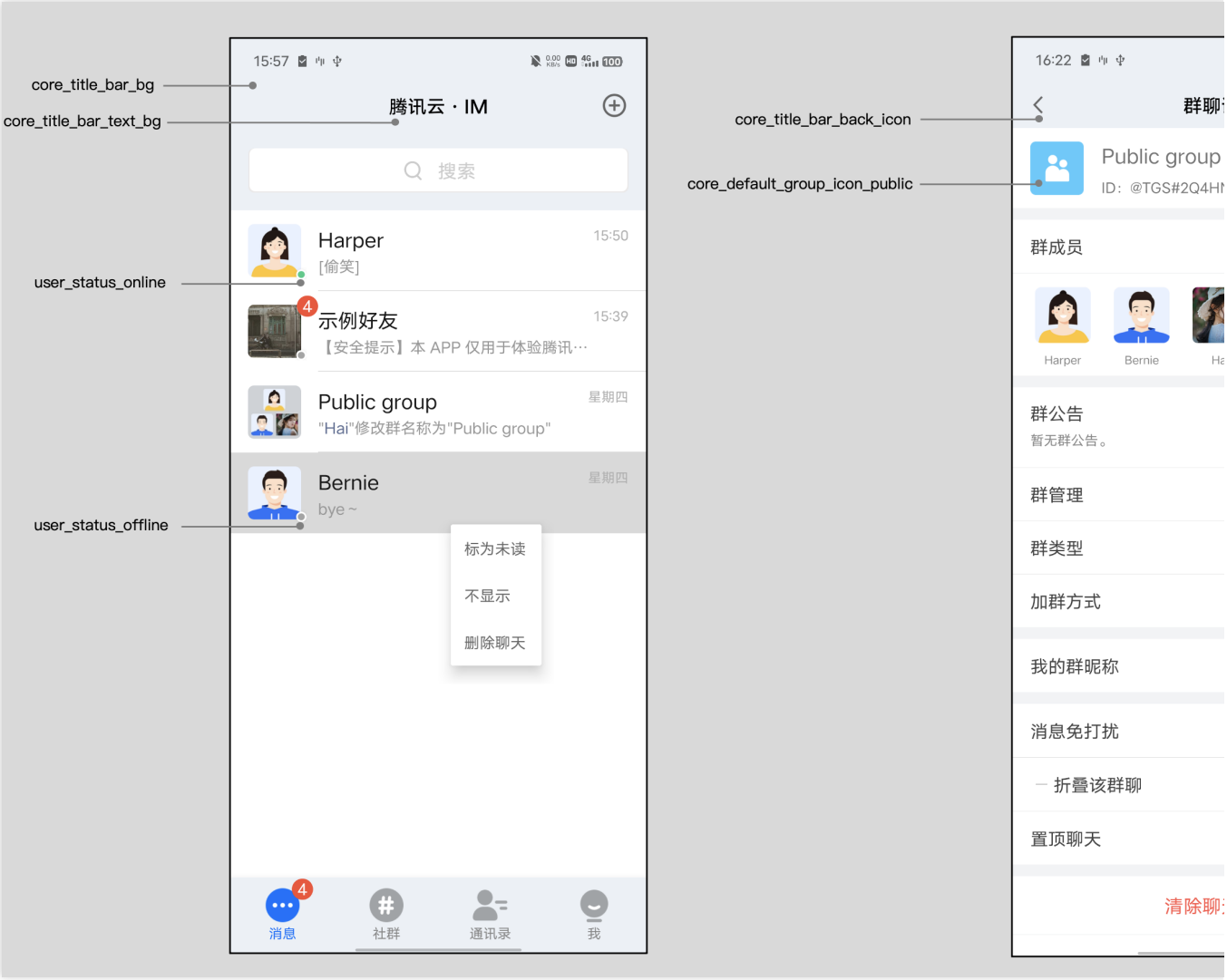
存储位置

基础样式均存在于 `TUICore` 组件中，由各个组件引用。

基础样式提供了公共的 UI 规范，例如：首选背景色、分割线颜色等。可以通过修改基础样式来同时影响其他各个组件。

您可以在 `TUICore/tuicore/src/main/res/values/tui_theme_attrs.xml` 文件中看到 `TUICore` 的所有主题属性。主题属性对应的资源放在 `tuicore/src/main/res-***` 文件夹中。

UI 样式表



图标

属性名称	属性说明
core_title_bar_back_icon	标题栏返回按钮图标
core_default_group_icon_public	默认 Public 群头像图标
core_default_group_icon_work	默认 Work 群头像图标
core_default_group_icon_meeting	默认 Meeting 群头像图标
core_default_group_icon_community	默认 Community 群头像图标
core_default_user_icon	默认用户头像图标
user_status_online	用户在线状态图标

版权所有：腾讯云计算（北京）有限责任公司

user_status_offline	用户离线状态图标
core_selected_icon	选中图标

背景色

属性名称	属性说明
core_light_bg_title_text_color	浅色背景下标题文字颜色
core_light_bg_primary_text_color	浅色背景下主要文字颜色
core_light_bg_secondary_text_color	浅色背景下次要文字颜色
core_light_bg_secondary2_text_color	浅色背景下再次文字颜色
core_light_bg_disable_text_color	浅色背景下不可用文字颜色
core_dark_bg_primary_text_color	深色背景下主要文字颜色
core_bg_color	主要背景色
core_primary_color	主题色
core_error_tip_color	错误提示颜色
core_success_tip_color	成功提示颜色
core_bubble_bg_color	气泡背景色
core_divide_color	分割线颜色
core_border_color	边框颜色
core_header_start_color	标题栏起始色
core_header_end_color	标题栏终点色
core_btn_normal_color	按钮常态颜色
core_btn_pressed_color	按钮按下颜色
core_btn_disable_color	按钮不可用颜色
core_title_bar_bg	标题栏背景
core_title_bar_text_bg	标题栏文字背景色

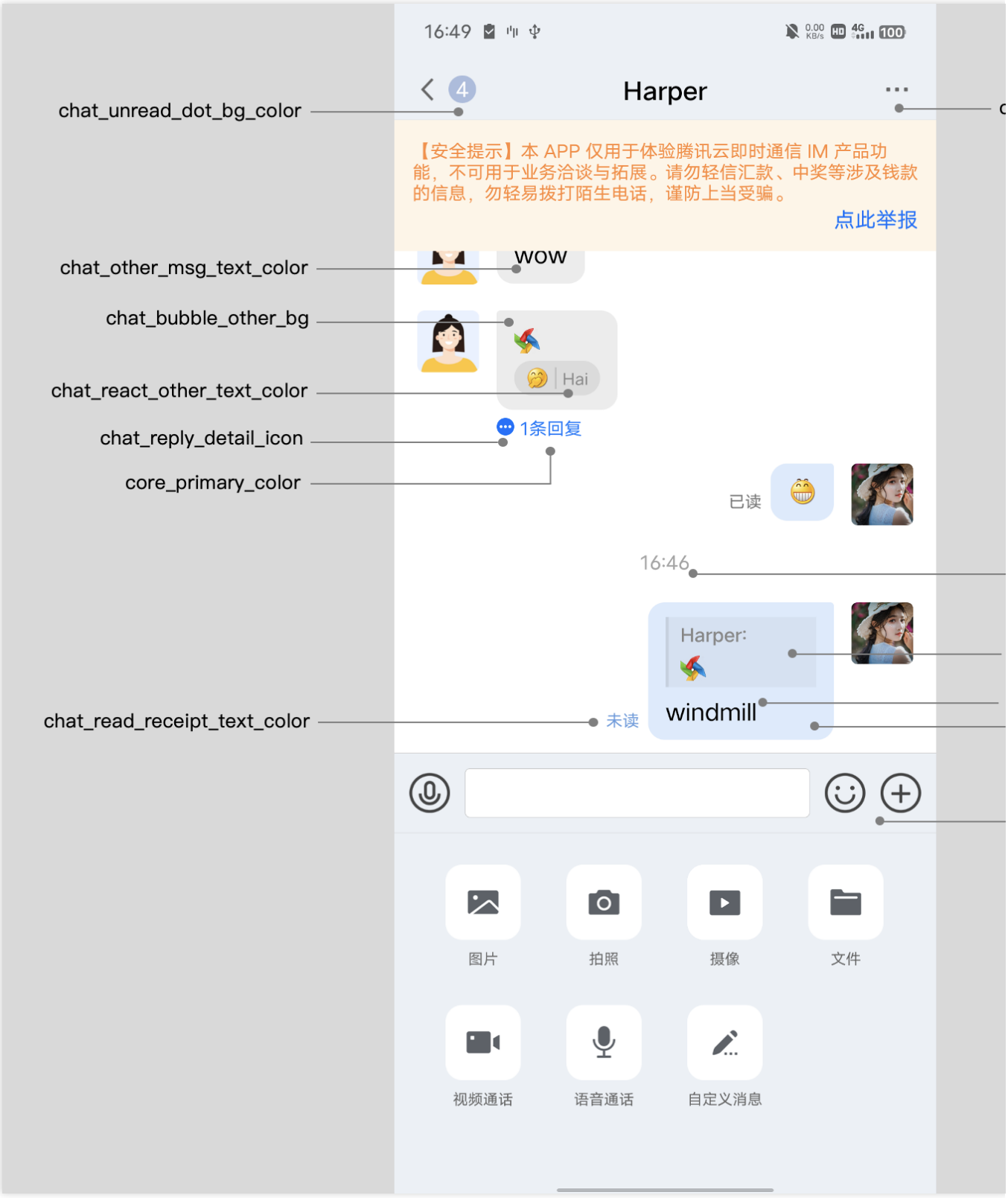
版权所有：腾讯云计算（北京）有限责任公司

Chat 页面样式

存储位置

您可以在 `TUIChat/tuichat/src/main/res/values/tui_theme_attrs.xml` 文件中看到 `TUIChat` 的所有主题属性。主题属性对应的资源放在 `tuichat/src/main/res-***` 文件夹中。

UI 样式表



图标

属性名称	属性说明
chat_title_bar_more_menu	标题栏菜单图标

chat_reply_detail_icon	回复详情图标
chat_jump_recent_down_icon	消息列表向下跳转图标
chat_jump_recent_up_icon	消息列表向上跳转图标

背景色

属性名称	属性说明
chat_bubble_self_bg	己方消息的气泡背景
chat_bubble_other_bg	对方消息的气泡背景
chat_bubble_self_bg_color	己方消息的气泡背景颜色
chat_bubble_other_bg_color	对方消息的气泡背景颜色
chat_input_area_bg	输入界面背景色
chat_unread_dot_bg_color	未读图标背景色
chat_unread_dot_text_color	未读图标中文字颜色
chat_other_msg_text_color	对方消息文字颜色
chat_self_msg_text_color	己方消息文字颜色
chat_self_custom_msg_text_color	己方自定义消息文字颜色
chat_other_custom_msg_text_color	对方自定义消息文字颜色
chat_self_custom_msg_link_color	己方自定义消息中链接文字颜色
chat_other_custom_msg_link_color	对方自定义消息中链接文字颜色
chat_tip_text_color	提示消息文字颜色
chat_self_reply_quote_bg_color	己方回复和引用消息背景色
chat_other_reply_quote_bg_color	对方回复和引用消息背景色
chat_self_reply_line_bg_color	己方回复消息竖线背景色
chat_other_reply_line_bg_color	对方回复消息竖线背景色
chat_self_reply_quote_text_color	己方回复消息中原始消息文字颜色
chat_other_reply_quote_text_color	对方回复消息中原始消息文字颜色

版权所有：腾讯云计算（北京）有限责任公司

chat_self_reply_text_color	己方回复消息文字颜色
chat_other_reply_text_color	对方回复消息文字颜色
chat_read_receipt_text_color	已读回执文字颜色
chat_react_text_color	表情回应己方文字颜色
chat_react_other_text_color	表情回应对方文字颜色
chat_pressed_bg_color	长按弹窗中按钮按下背景色

Group 页面样式

存储位置

您可以在 `TUIGroup/tuigroup/src/main/res/values/tui_theme_attrs.xml` 文件中看到 `TUIGroup` 的所有主题属性。主题属性对应的资源放在 `tuigroup/src/main/res-***` 文件夹中。

UI 样式表

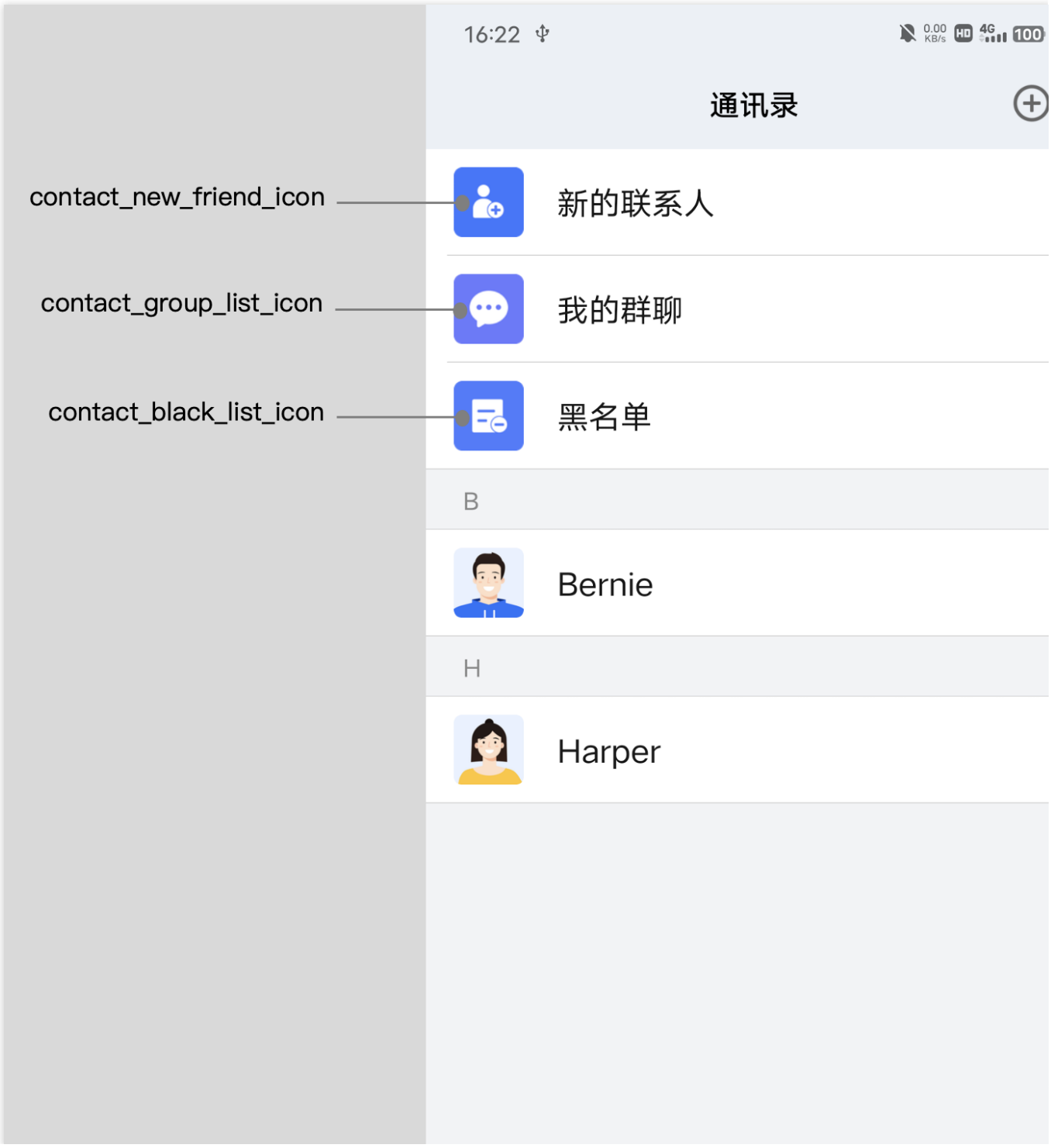
属性名称	属性说明
group_add_icon	添加按钮图标

Contact 页面样式

存储位置

您可以在 `TUIContact/tuicontact/src/main/res/values/tui_theme_attrs.xml` 文件中看到 `TUIContact` 的所有主题属性。主题属性对应的资源放在 `tuicontact/src/main/res-***` 文件夹中。

UI 样式表



属性名称	属性说明
contact_new_friend_icon	新的联系人菜单图标
contact_group_list_icon	我的群聊菜单图标
contact_black_list_icon	黑名单菜单图标

iOS

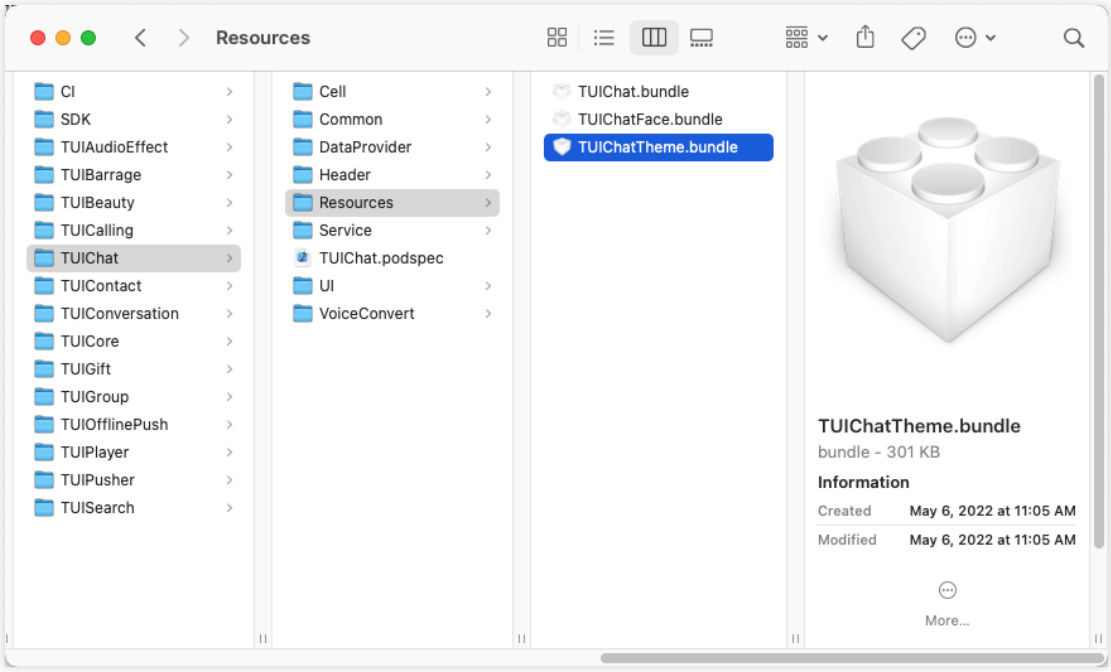
最近更新时间：2023-09-28 10:11:02

概述

TUI 组件默认内置了：轻量、深沉、活泼、深色共四套主题，以及支持跟随 iOS 系统自动切换深色主题和默认设置。您可以任意切换或者修改内置主题，也可以按需新增主题。

主题资源

您可以在任一 TUI 组件内部的 `Resources` 文件夹下看到该组件所支持的主题资源。以 TUIChat 组件为例，您可以在 `TUIChat/Resources/` 下看到 `TUIChatTheme.bundle` 文件。该文件即为 TUIChat 组件内置的主题资源。



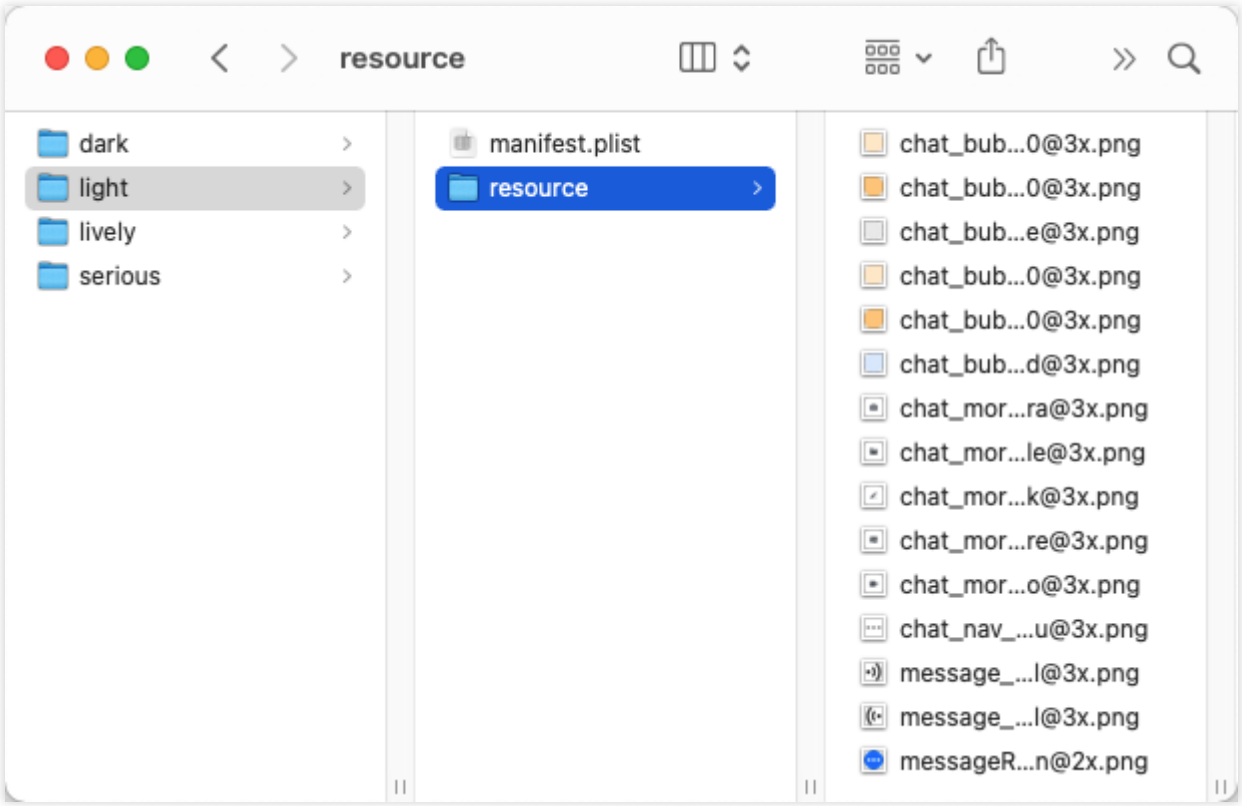
选中 `TUIChatTheme.bundle` 文件，并右键选择 `Show Package Contents` 即可看到内置的四套主题资源，文件夹名称即为主题 ID。

版权所有：腾讯云计算（北京）有限责任公司

我们以 “轻量主题” 为例，主题 ID 为 `light` ，每套主题内部均包含两个元素：`manifest.plist` 文件和 `resource` 资源文件夹。

manifest.plist 文件存储了当前主题所使用的图片、字体、颜色等要素的值，同一组件中不同主题下的 `manifest.plist` 文件中的 key 均相同；

resource 资源文件夹存储了当前主题所使用的资源，例如图片。

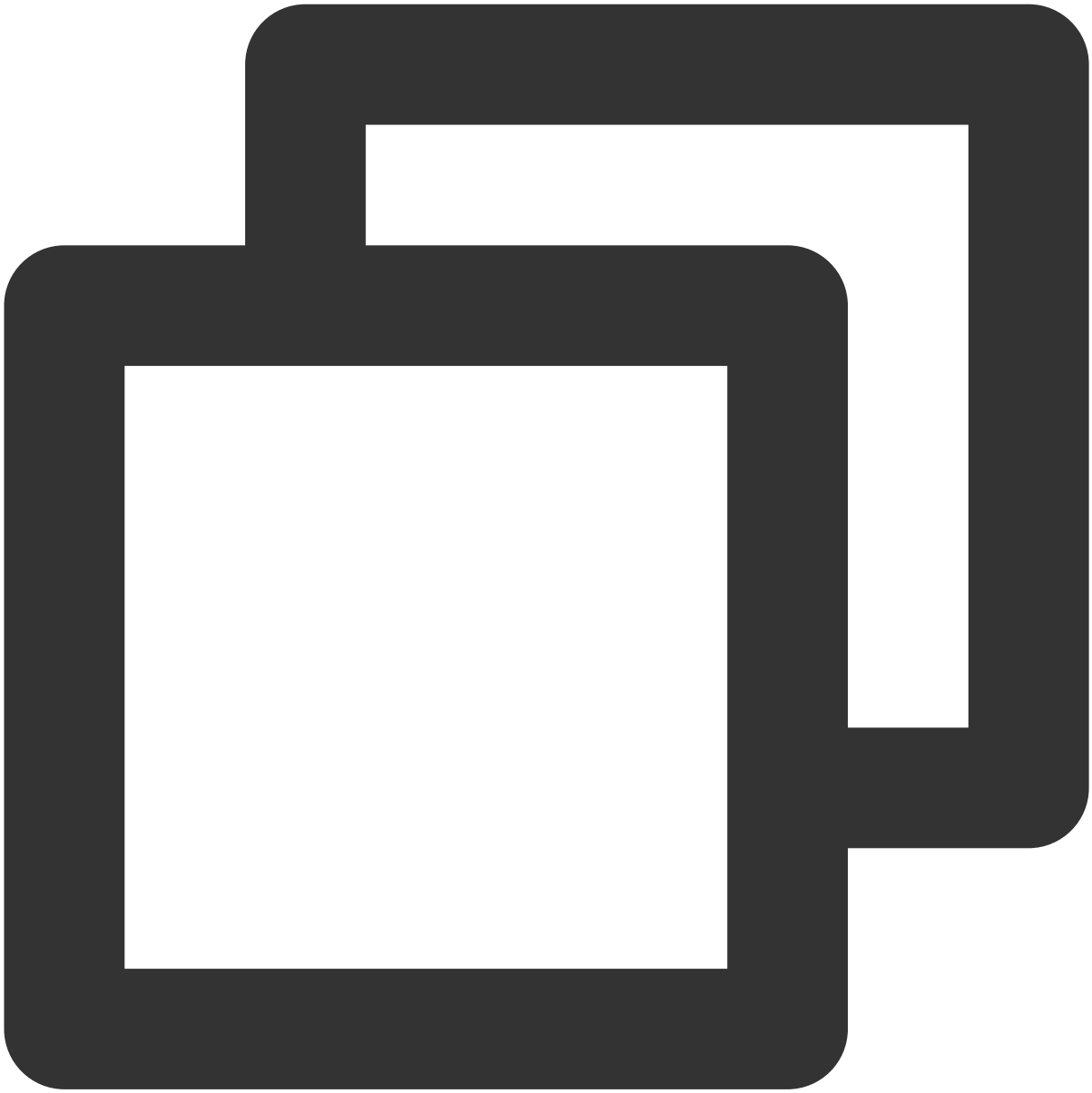


不管是修改内置主题，还是新增一套主题，都需要修改各组件下的 `manifest.plist` 文件。

应用主题

当您选中某个主题后，需要对 TUI 组件以及您的 App 主工程设置对应的主题。您可以调用 `TUIThemeManager` 的 `-applyTheme:forModule:` 方法，为指定组件应用主题。

您可以参考 TUIKitDemo 的 ThemeSelectController 的 `+applyTheme:` 方法。



```
+ (void)applyTheme:(NSString * __nullable)themeID {
    // 获取 App 上次启动所使用的主题 ID
    NSString *lastThemeID = [self getCacheThemeID];
    if (themeID.length) {
        lastThemeID = themeID;
    }

    // 组件：应用/卸载主题
    if (lastThemeID == nil || lastThemeID.length == 0 || [lastThemeID isEqualToStri
        // 主题 ID 为空，或者明确跟随系统，卸载所有组件的主题
        [TUIShareThemeManager unApplyThemeForModule:TUIThemeModuleAll];
```

版权所有：腾讯云计算（北京）有限责任公司

```
    } else {
          // 为所有组件应用主题
        [TUIShareThemeManager applyTheme:lastThemeID forModule:TUIThemeModuleAll];
    }

    // 系统暗黑样式：与主题互斥
    dispatch_async(dispatch_get_main_queue(), ^{
        if (@available(iOS 13.0, *)) {
            if (lastThemeID == nil || lastThemeID.length == 0 || [lastThemeID isEqu
                // 跟随系统变化
                UIApplication.sharedApplication.keyWindow.overrideUserInterfaceStyl
            } else if ([lastThemeID isEqual:@"dark"]) {
                // 强制切换成黑夜模式
                UIApplication.sharedApplication.keyWindow.overrideUserInterfaceStyl
            } else {
                // 忽略系统变化，强制切换成白天模式，并使用主题
                UIApplication.sharedApplication.keyWindow.overrideUserInterfaceStyl
            }
        }
    });
}
```

修改内置主题

TUI 组件目前支持修改内置的主题，只需要按照下列步骤，即可对内置主题的颜色、字体、图片等要素做自定义变更。

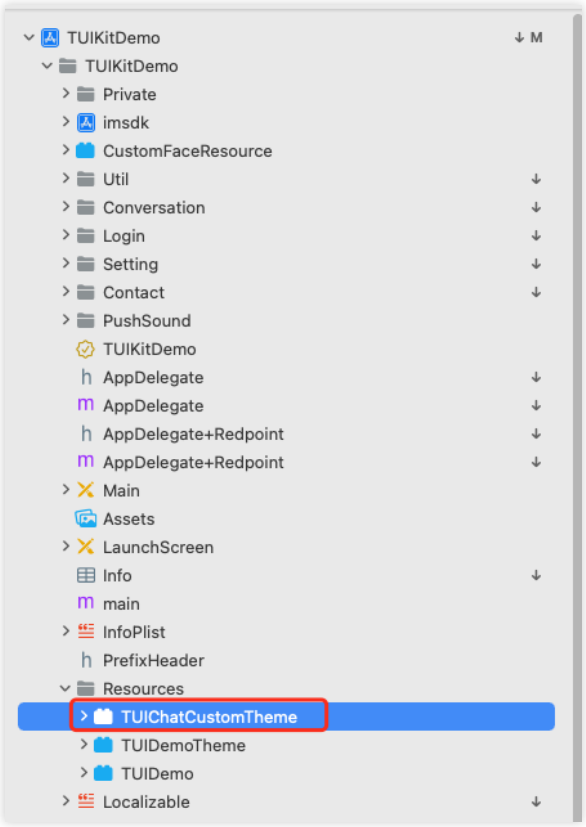
拷贝 TUI 组件内置的主题资源包到您的工程中，并修改各主题下对应的主题要素；

在 App 启动时，将您已经修改好的主题资源包路径注册到 TUI 组件中；

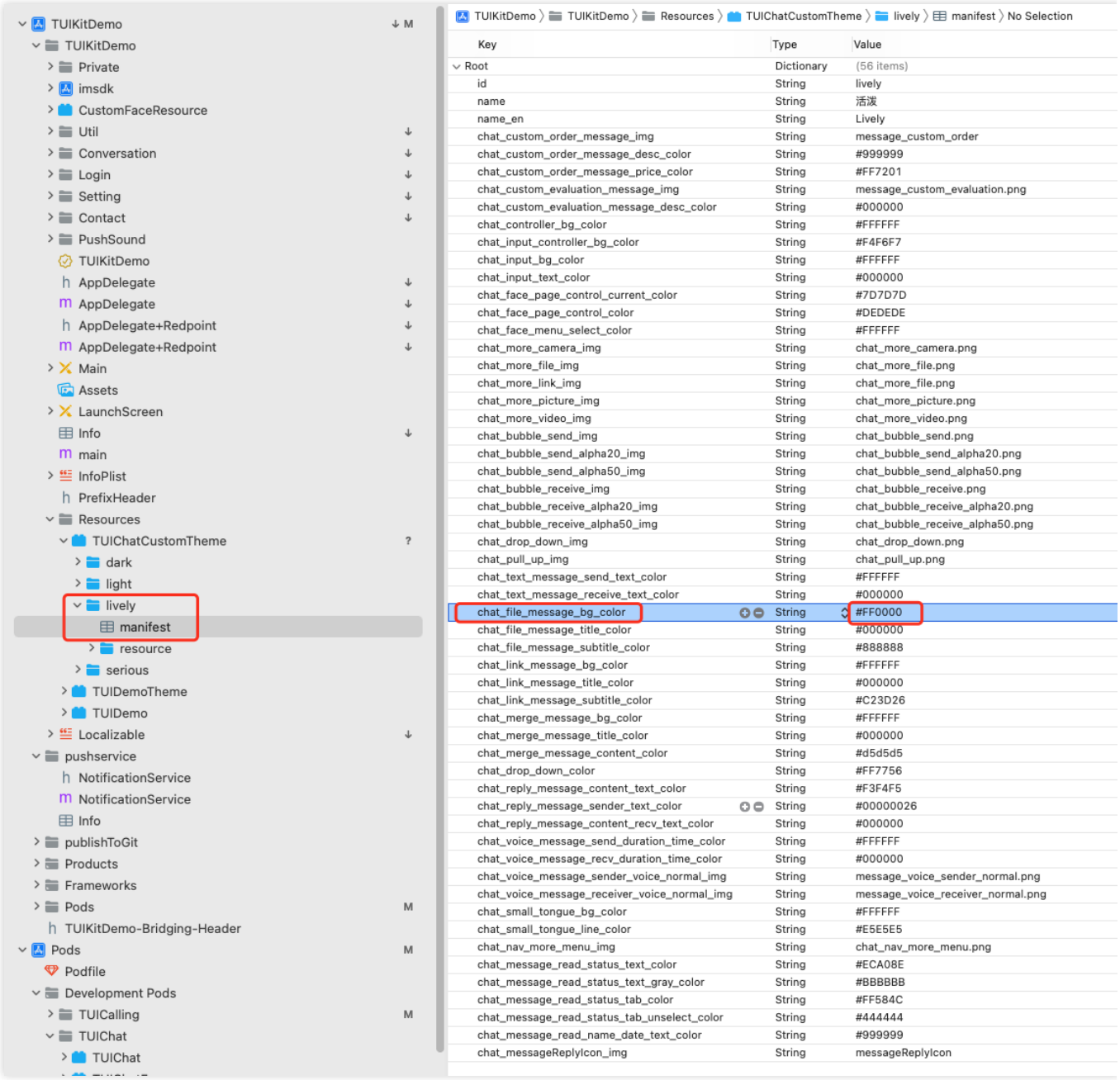
当切换到对应主题后，TUI 组件会自动应用您修改后的主题包。

例如，TUIChat 组件中文件消息的背景色，在不同主题下有不一样的颜色。该背景色在内置的 “活泼” 主题下的色值为 #FFFFFF，现在想要修改成 #FF0000，您只需要按照如下步骤操作即可：

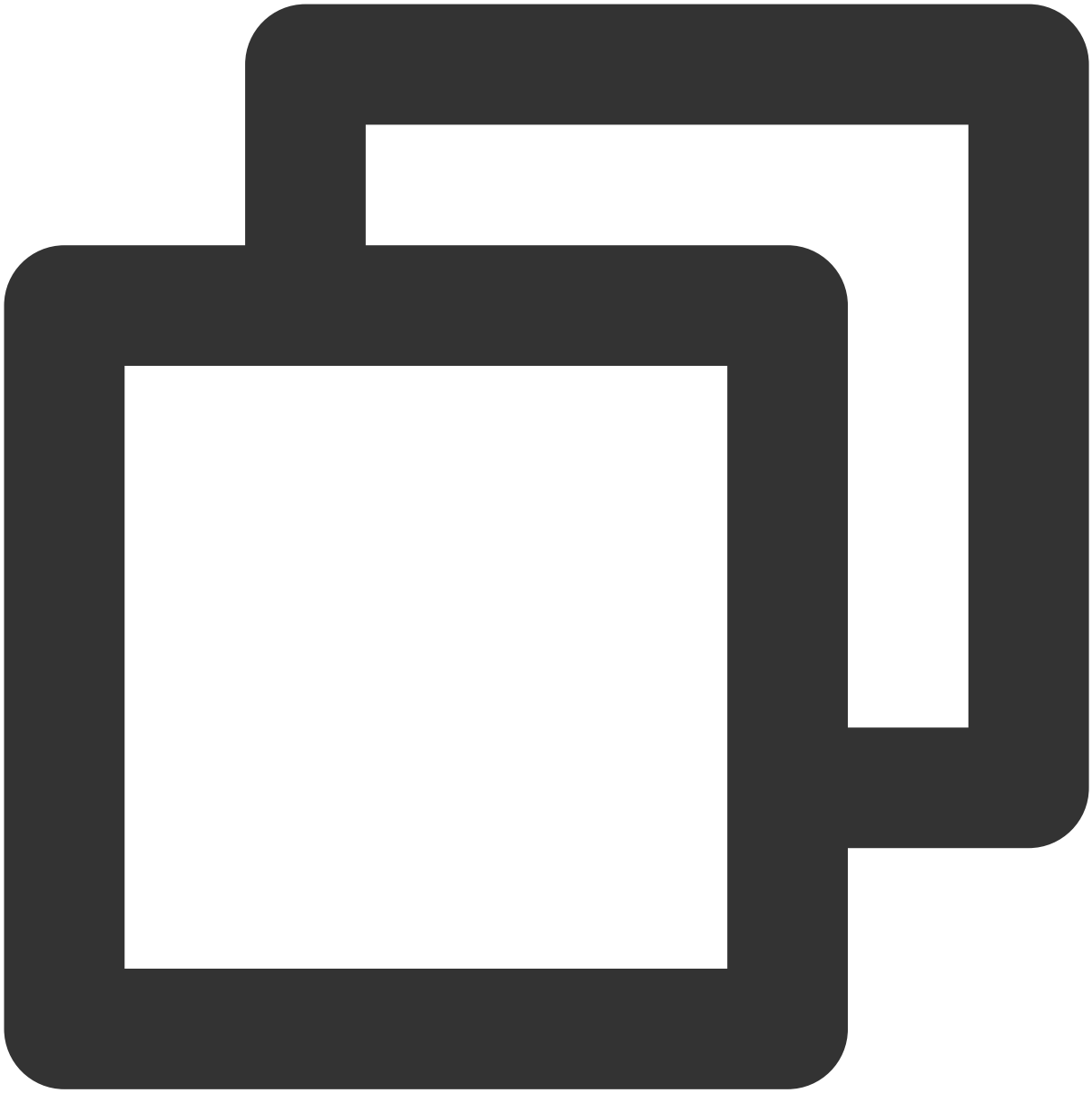
1. 将 TUIChat 组件 `TUIChat/Resources/TUIChatTheme.bundle` 文件拷贝一份到您的主工程中，并重命名为 `TUIChatCustomTheme.bundle` ；



2. 修改 manifest.plist 文件中标识文件消息背景色的 value 值，其中各 key 值的释义详见 Chat 页面样式；



3. 在 `- application:didFinishLaunchingWithOptions:` 方法中，调用 `TUIRegisterThemeResourcePath` 注册修改后的主题包资源路径，用于覆盖 TUIChat 内置的主题包，您可以参考 TUIKitDemo 的 AppDelegate 文件；



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

    // 自定义修改 TUIChat 组件的主题 - 修改主题资源包中的内置主题
    // -- 1. 获取自定义后的资源包路径
    NSString *customChatThemePath = [NSBundle.mainBundle pathForResource:@"TUIChatC
    // -- 2. 给 TUIChat 组件注册自定义的主题资源包路径，用于覆盖内置的主题，note: 此时只能覆盖
    TUIRegisterThemeResourcePath(customChatThemePath, TUIThemeModuleChat);

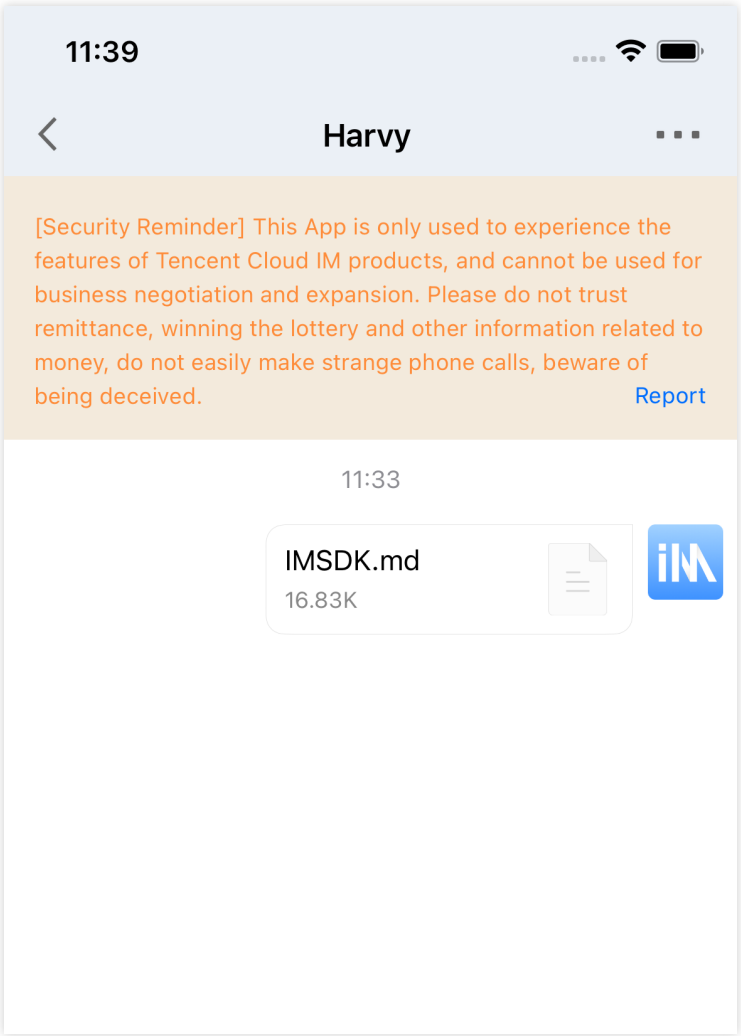
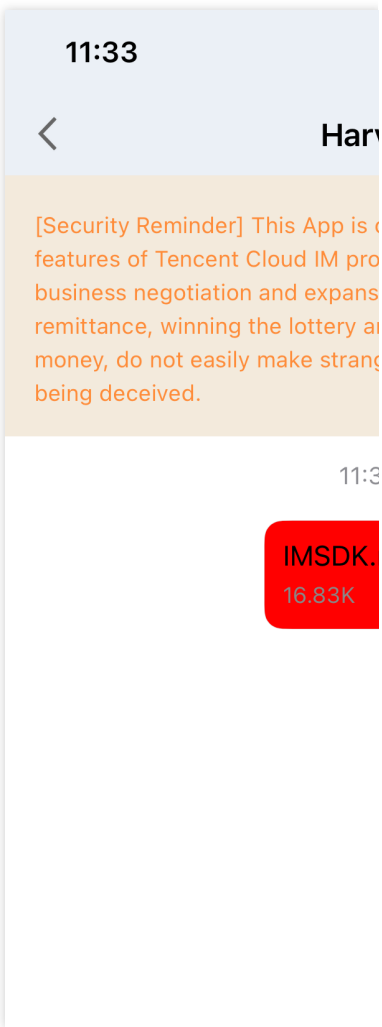
    // TUIKitDemo 注册主题
    NSString *demoThemePath = [NSBundle.mainBundle pathForResource:@"TUIDemoTheme.b
    TUIRegisterThemeResourcePath(demoThemePath, TUIThemeModuleDemo);
```

```
    [ThemeSelectController applyLastTheme];

    [self setupListener];
    [self setupGlobalUI];
    [self setupConfig];
    [self tryAutoLogin];

    return YES;
}
```

4. 再次启动 App，您可以看到对应的背景色已经被成功修改。

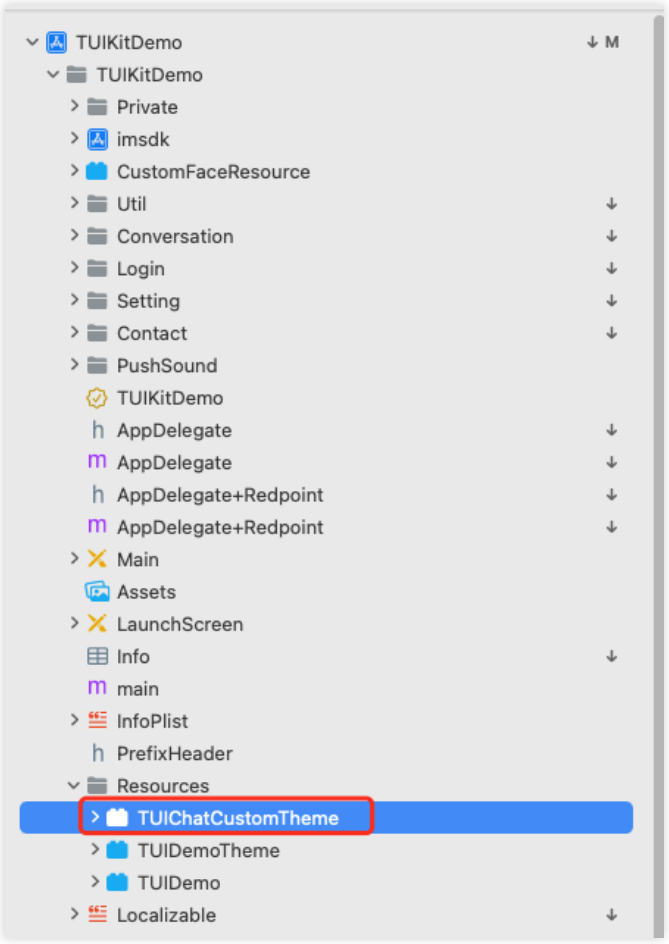
修改前	修改后
	

说明

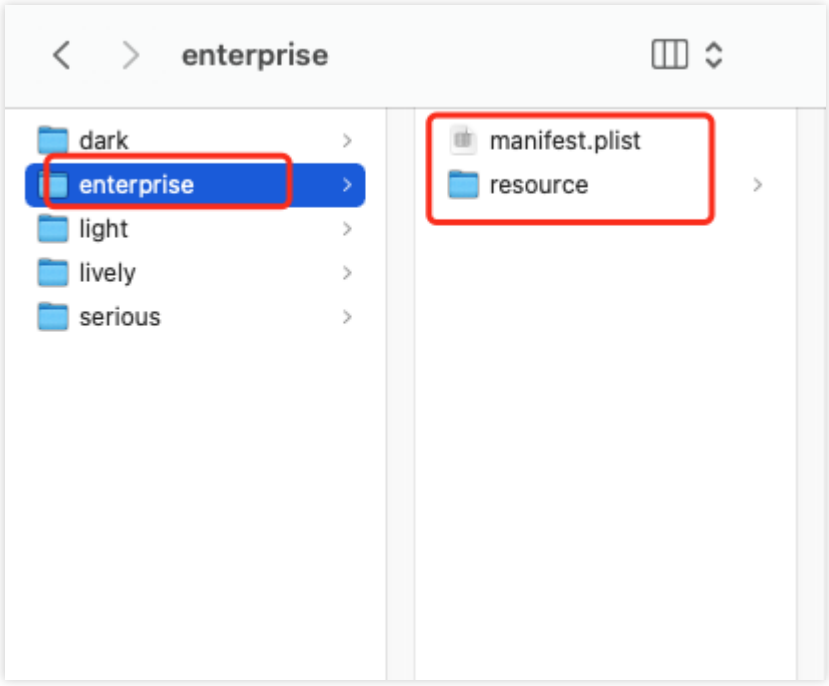
同样地，如果您想修改某个内置的图标，依然可以将图标资源存放到主题文件夹下的 Resource 文件夹中，然后修改 manifest 中对应 key 的 value 值。

新增主题

如果内置的 4 套主题无法满足您的需求，您可以自行按照如下步骤为组件新增一套全新的主题：
拷贝 TUI 组件内置的主题资源包到您的工程中，在主题资源包中新建一个主题资源文件夹，文件夹名称即为新增主题的 `themeID` ；
将 TUI 组件内置的主题资源包中的 light 文件夹下的 `manifest.plist` 文件拷贝至新创建的主题文件夹中，并修改 `id` 、 `name` 以及 `name_en` 的值；
在新创建的主题文件夹中新建一个 `resource` 文件夹，用于存放新增主题的资源文件；
按需修改新增主题的 manifest.plist 文件；
在 App 启动时，将您已经修改好的主题资源包路径注册到 TUI 组件中，并应用当前的新增主题。
例如，给 TUIChat 组件新创建一套新的主题 “商务”， `themeID` 为 `enterprise` ，您可以按照如下步骤操作：
1. 将 TUIChat 组件 `TUIChat/Resources/TUIChatTheme.bundle` 文件拷贝一份到您的主工程中，并重命名为 `TUIChatCustomTheme.bundle` ；

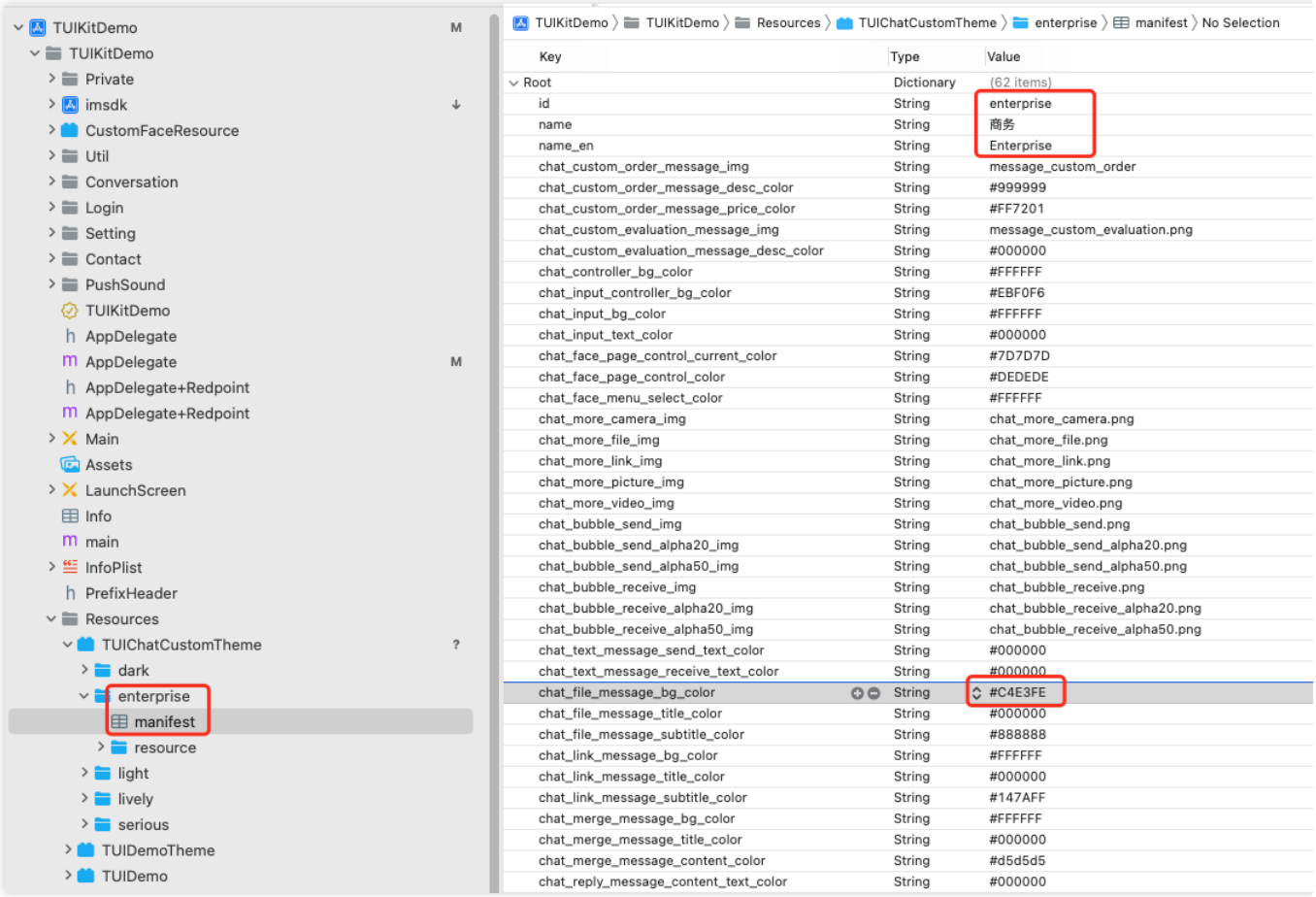


2. 复制 `TUIChatCustomTheme.bundle` 下的 `light` 文件夹，并重命名为 `enterprise` ；



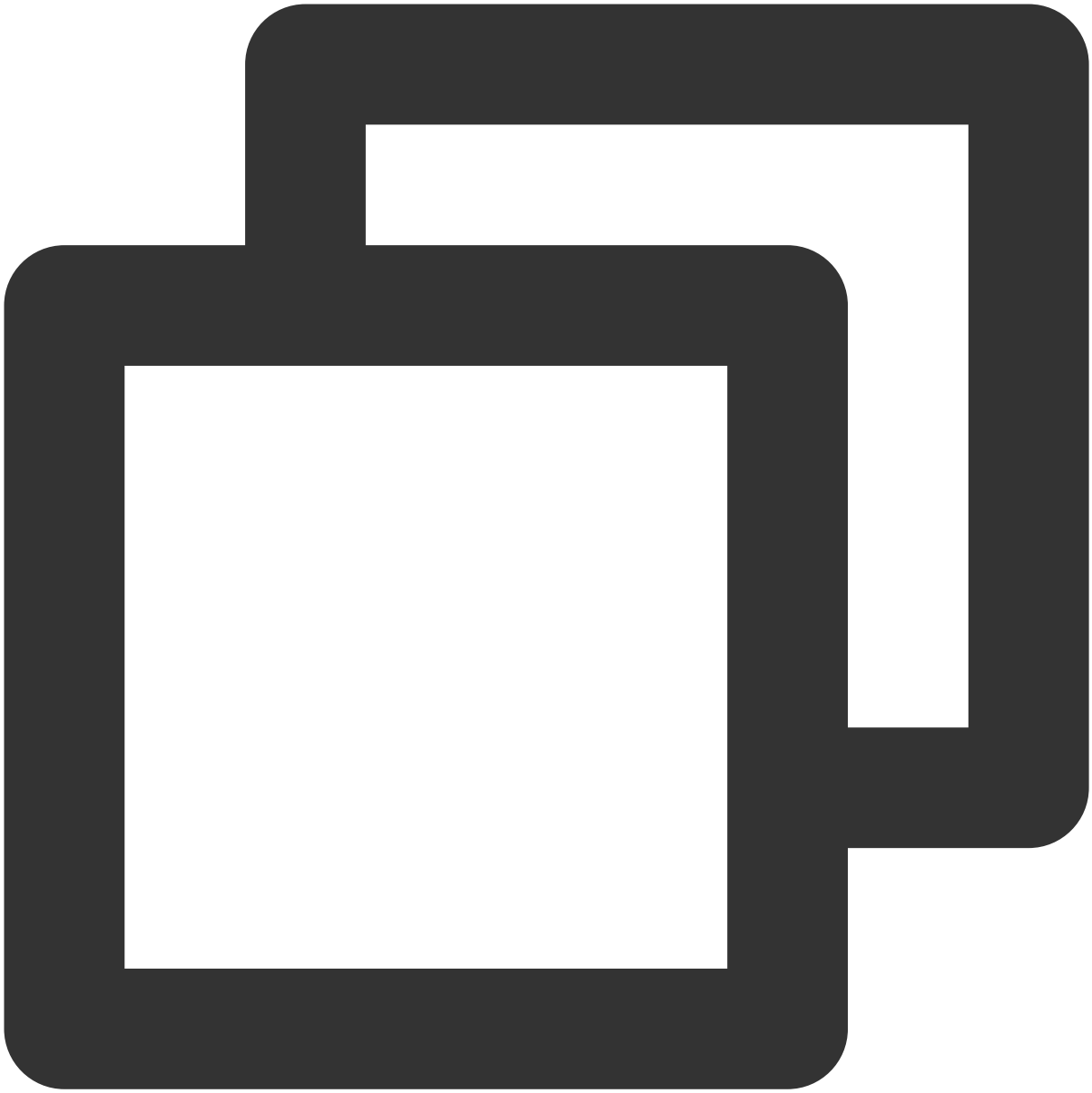
3. 按需修改 enterprise 文件夹下的 `manifest.plist` 文件中的 value 值，其中各 key 值的释义详见 Chat 页面样式；

例如将文件消息的背景色修改为 #C4E3FE。



4. 在 App 启动时，将您已经修改好的主题资源包路径注册到 TUI 组件中，并应用当前的新增主题。

版权所有：腾讯云计算（北京）有限责任公司



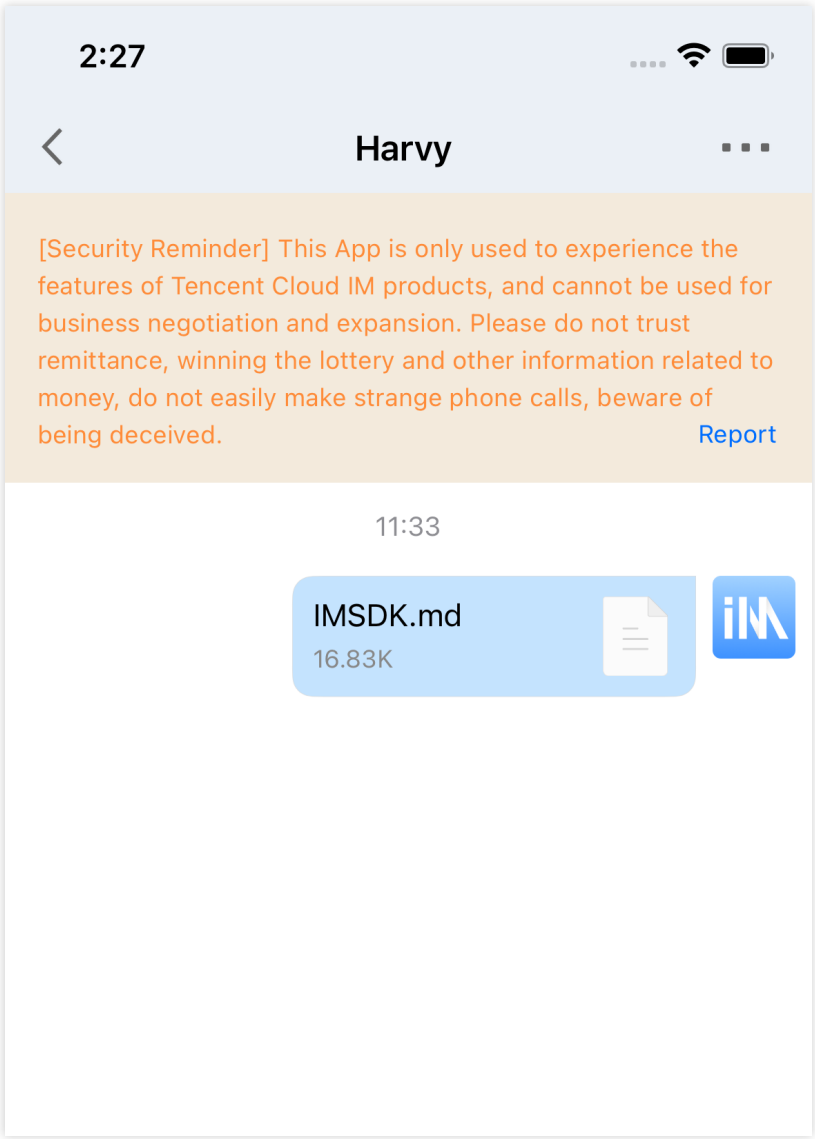
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD

    // 自定义修改 TUIChat 组件的主题 - 在主题资源包中新增了一套主题
    NSString *customChatThemePath = [NSBundle.mainBundle pathForResource:@"TUIChatC
    TUIRegisterThemeResourcePath(customChatThemePath, TUIThemeModuleChat);

    // 应用主题，根据 themeID 对 TUIChat 设置主题
    [TUIShareThemeManager applyTheme:@"enterprise" forModule:TUIThemeModuleChat];

    return YES;
}
```

5. 再次启动 App，您可以看到新增到的商务主题被成功应用到了 App。



主题样式表

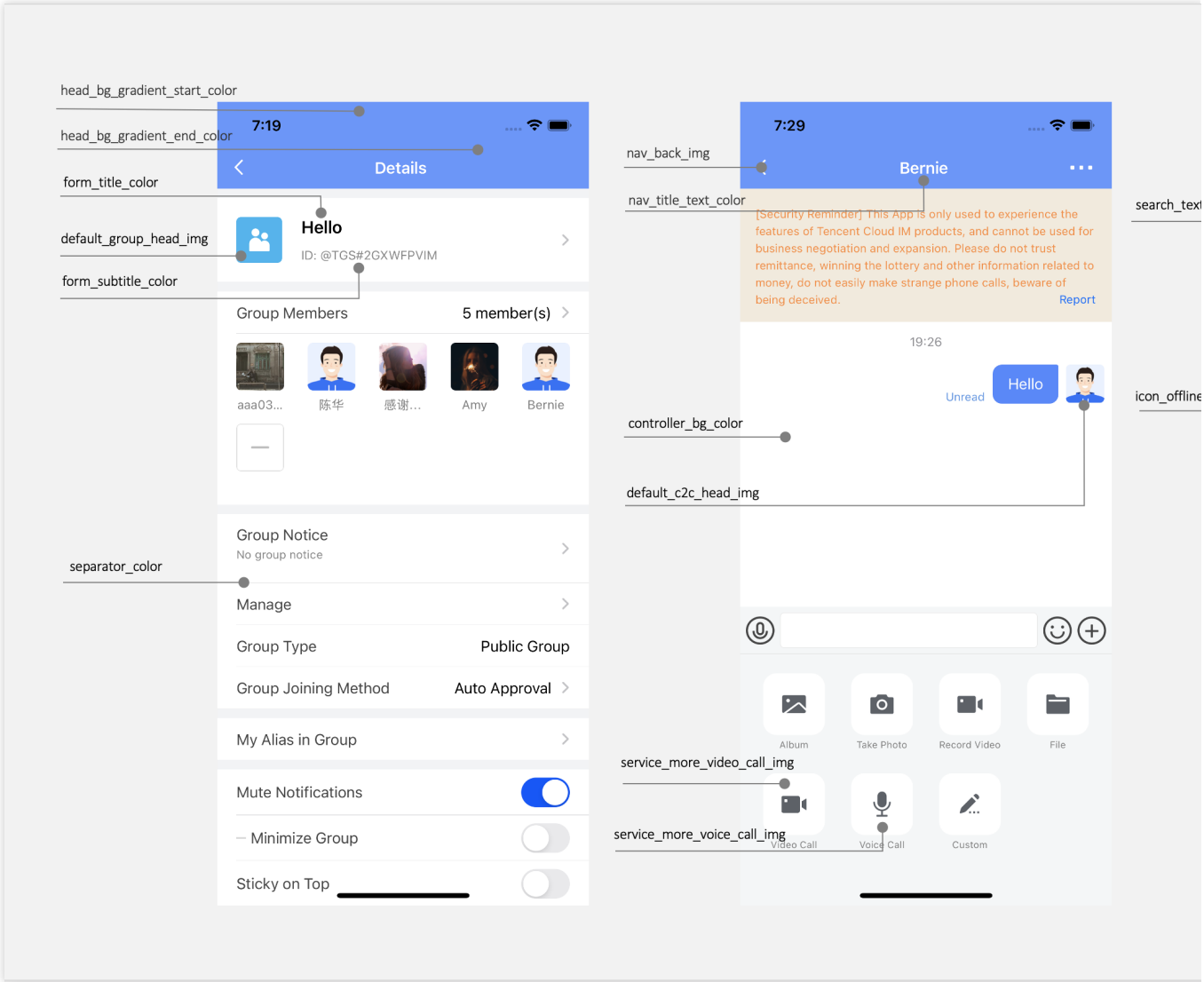
基础样式

存储位置

基础样式均存在于 `TUICore` 组件中，由各个组件引用。

您可以在 `TUICore` 组件的 `TUICore/Resources/TUICoreTheme.bundle` 下的任一主题文件夹中的 `manifest.plist` 文件中看到对应样式的 `key` 。
在基础样式中， `TUICore` 提供了公共的 UI 规范，例如：首选背景色、分割线颜色等。您可以通过修改基础样式来同时影响其他各个组件。

UI 样式表



图标

样式 key	样式释义
nav_back_img	导航栏返回按钮的图片名
default_group_head_public_img	Public 类型的群聊，默认头像
default_group_head_meeting_img	Meeting 类型的群聊，默认头像

default_group_head_avchatroom_img	AVChatRoom 类型的群聊，默认头像
default_group_head_img	默认的群头像
default_c2c_head_img	缺省的用户头像
service_more_video_call_img	聊天界面更多选项卡中的视频通话图标
service_more_voice_call_img	聊天界面更多选项卡中的语音通话图标
icon_online_status_img	用户在线状态的图标
icon_offline_status_img	用户离线状态的图标

颜色

样式 key	样式释义
primary_theme_color	主题色，描述了当前主题下的主色调
common_switch_on_color	通用 UISwitch 组件，开关打开时的颜色
head_bg_gradient_start_color	导航栏顶部背景颜色，渐变色
head_bg_gradient_end_color	导航栏顶部背景颜色，渐变色
separator_color	分割线颜色
controller_bg_color	控制器背景颜色
form_title_color	表单，UITableViewCell 的主标题文本颜色
form_subtitle_color	表单，UITableViewCell 的副标题文本颜色
form_desc_color	表单，UITableViewCell 的描述信息文本颜色
form_bg_color	表单，UITableViewCell 的背景颜色
form_green_button_text_color	表单，UITableViewCell 中绿色主题按钮的文本颜色
form_green_button_bg_color	表单，UITableViewCell 中绿色主题按钮的背景颜色
form_green_button_highlight_bg_color	表单，UITableViewCell 中绿色主题按钮高亮时的文本颜色
form_white_button_text_color	表单，UITableViewCell 中白色主题按钮的文本颜色
form_white_button_bg_color	表单，UITableViewCell 中白色主题按钮的背景颜色
form_key_text_color	表单，UITableViewCell 中描述文本颜色

版权所有：腾讯云计算（北京）有限责任公司

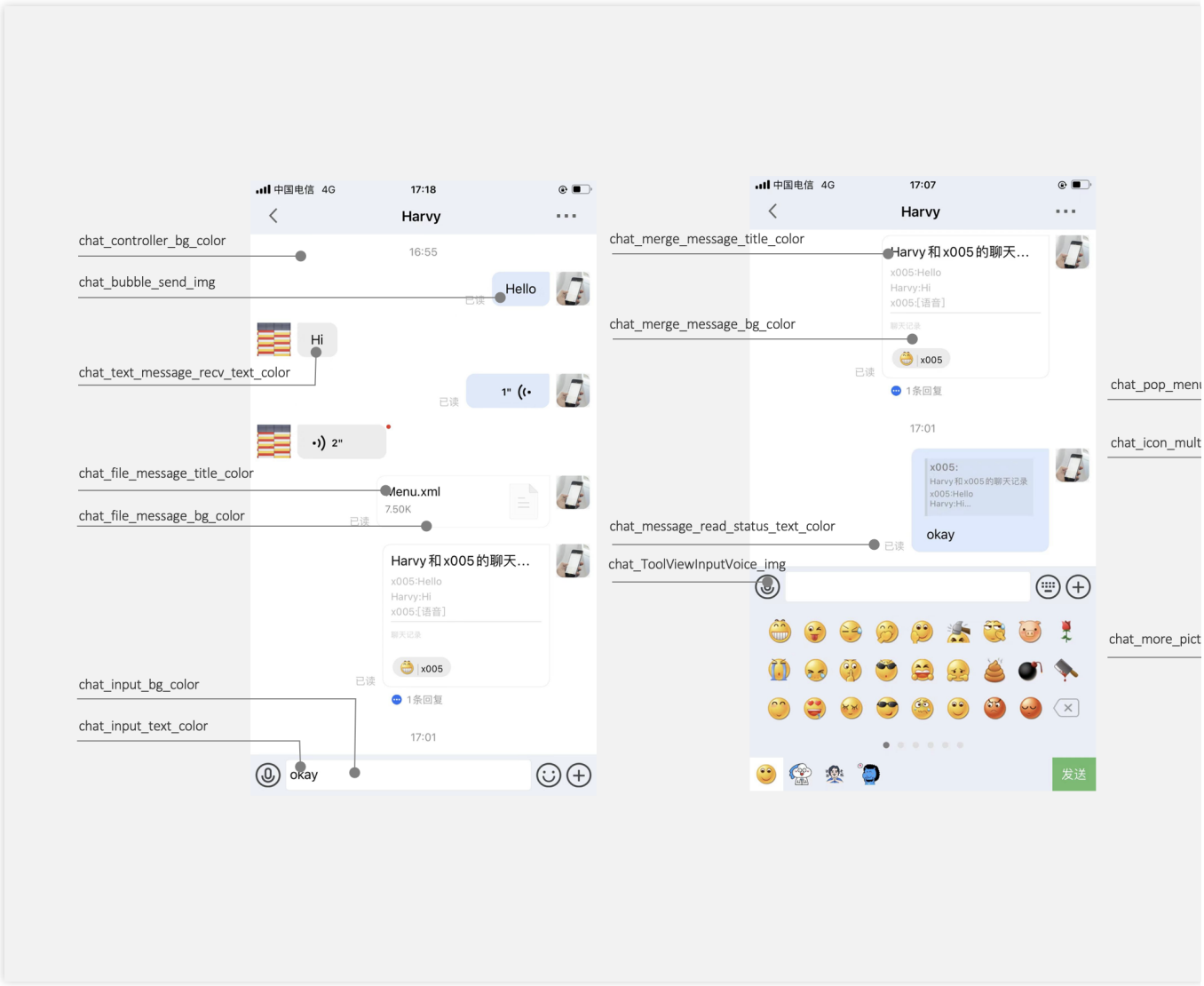
form_value_text_color	表单，UITableViewCell 中值的文本颜色
nav_title_text_color	导航栏文本颜色
nav_back_img	导航栏返回按钮的图片名
search_textfield_bg_color	搜索栏输入框的背景颜色

Chat 页面样式

存储位置

Chat 页面样式均存在于 `TUIChat` 组件中，供 Chat 页面使用。
您可以在 `TUIChat` 组件的 `TUIChat/Resources/TUIChatTheme.bundle` 下的任一主题文件夹中的 `manifest.plist` 文件中看到对应样式的 `key` 。

UI 样式表



图标

样式 key	样式释义
chat_more_camera_img	更多选项卡，相机图标
chat_more_file_img	更多选项卡，文件图标
chat_more_link_img	更多选项卡，自定义图标
chat_more_picture_img	更多选项卡，图片图标
chat_more_video_img	更多选项卡，录像图标
chat_bubble_send_img	消息气泡，发送时的背景图片
chat_bubble_receive_img	消息气泡，接收时的背景图片
chat_voice_message_sender_voice_normal_img	语音消息，发送消息时正常状态下的背景图片

版权所有：腾讯云计算（北京）有限责任公司

chat_voice_message_receiver_voice_normal_img	语音消息，接收消息时正常状态下的背景图片
chat_icon_copy_img	聊天页面，长按消息后弹出的菜单页面中的 “复制” 图标
chat_icon_delete_img	聊天页面，长按消息后弹出的菜单页面中的 “删除” 图标
chat_icon_recall_img	聊天页面，长按消息后弹出的菜单页面中的 “撤回” 图标
chat_icon_multi_img	聊天页面，长按消息后弹出的菜单页面中的 “多选” 图标
chat_icon_forward_img	聊天页面，长按消息后弹出的菜单页面中的 “转发” 图标
chat_icon_reply_img	聊天页面，长按消息后弹出的菜单页面中的 “回复” 图标
chat_icon_reference_img	聊天页面，长按消息后弹出的菜单页面中的 “引用” 图标
chat_ToolViewInputVoice_img	聊天页面，输入栏 “语音/键盘” 切换按钮的图标
chat_ToolViewEmotion_img	聊天页面，输入栏 “表情/键盘” 切换按钮的图标
chat_ToolViewKeyboard_img	聊天页面，输入栏 “键盘” 按钮的图标

颜色

样式 key	样式释义
chat_controller_bg_color	聊天页面，背景颜色
chat_input_controller_bg_color	聊天页面，输入控制页面背景色
chat_input_bg_color	聊天页面，输入框背景颜色
chat_input_text_color	聊天页面，输入框文本颜色
chat_face_page_control_current_color	表情选项卡，分页控件，当前页的颜色
chat_face_page_control_color	表情选项卡，分页控件，默认颜色
chat_text_message_send_text_color	文本消息，发送消息时显示的文本颜色
chat_text_message_receive_text_color	文本消息，接收消息时显示的文本颜色
chat_file_message_bg_color	文件消息，背景颜色
chat_file_message_title_color	文件消息，标题文本颜色
chat_file_message_subtitle_color	文件消息，副标题文本颜色
chat_merge_message_bg_color	合并消息，背景颜色

版权所有：腾讯云计算（北京）有限责任公司

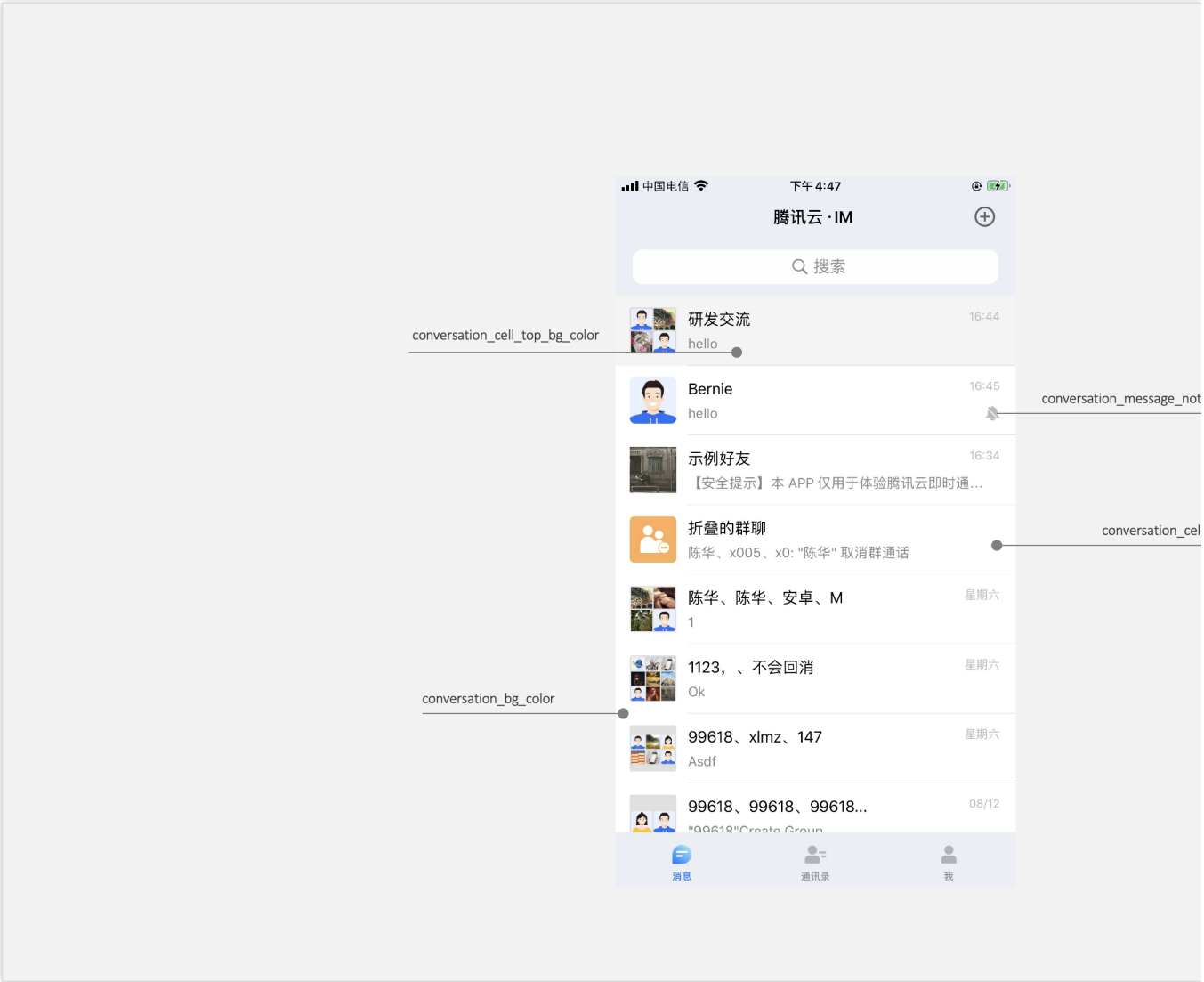
chat_merge_message_title_color	合并消息，标题文本颜色
chat_merge_message_content_color	合并消息，内容文本颜色
chat_drop_down_color	聊天页面，向下箭头的颜色
chat_voice_message_send_duration_time_color	语音消息，发送消息时显示的时长文本颜色
chat_voice_message_recv_duration_time_color	语音消息，接收消息时显示的时长文本颜色
chat_small_tongue_bg_color	聊天页面，“回到最新位置” 组件的背景色
chat_small_tongue_line_color	聊天页面，“回到最新位置” 组件的分割线颜色
chat_pop_menu_bg_color	聊天页面，长按消息后弹出的菜单页面的背景色
chat_pop_menu_text_color	聊天页面，长按消息后弹出的菜单页面的文本颜色
chat_message_read_status_text_color	聊天页面，消息已读状态文本提示语颜色

Conversation 页面样式

存储位置

Conversation 页面样式均存在于 `TUIConversation` 组件中，供 Conversation 页面使用。
您可以在 `TUIConversation` 组件的 TUIConversation/Resources/TUIConversationTheme.bundle 下的任一主题文件夹中的 manifest.plist 文件中看到对应样式的 key。

UI 样式表



样式 key	样式释义
conversation_cell_bg_color	会话列表页面，普通会话的 UITableViewCell 的背景颜色
conversation_cell_top_bg_color	会话列表页面，置顶会话的 UITableViewCell 的背景颜色
conversation_bg_color	会话列表页面的背景颜色
conversation_message_not_disturb_img	会话列表页面，消息免打扰图标

Group 页面样式

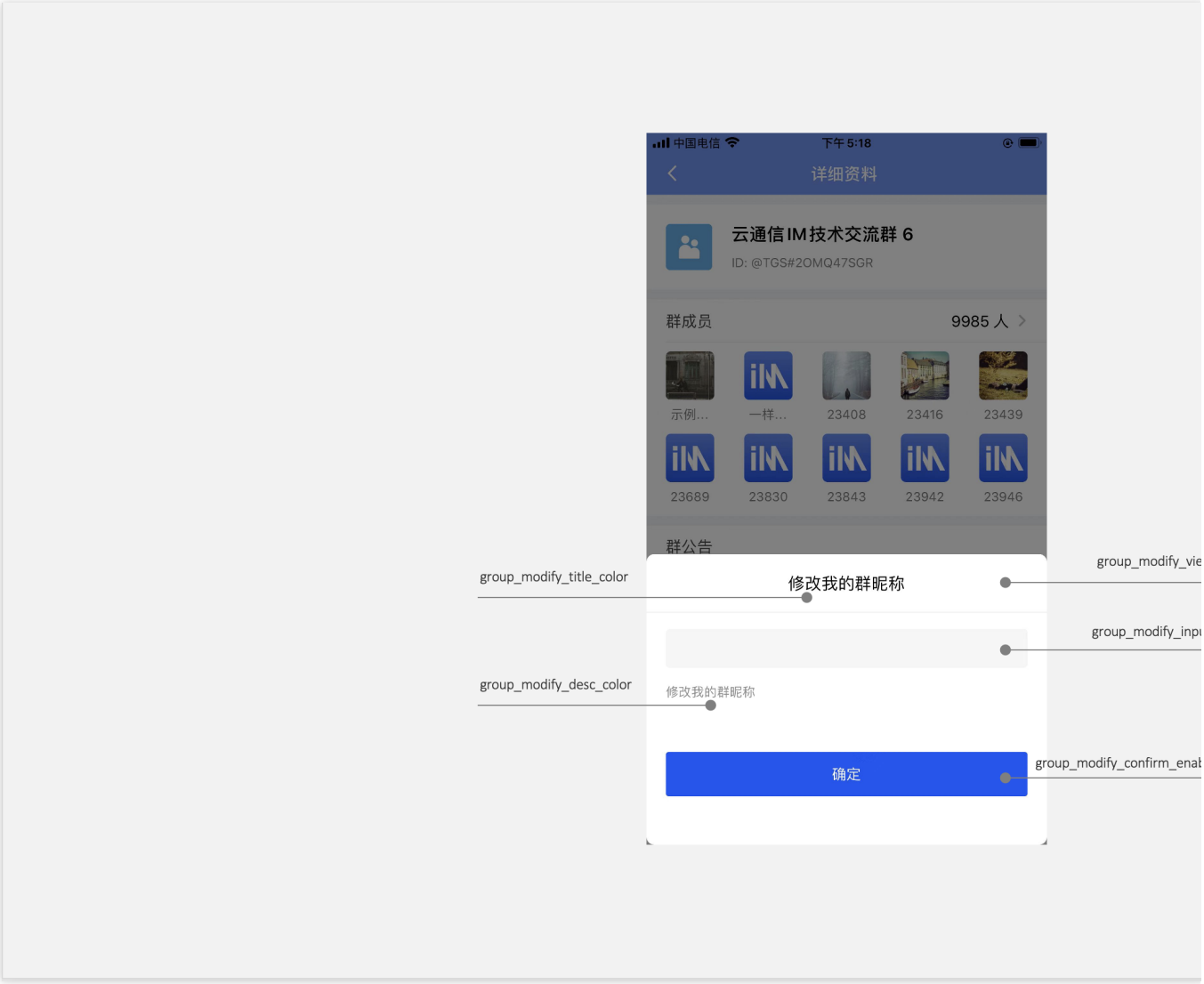
存储位置

Group 页面样式均存在于 `TUIGroup` 组件中，供 Group 页面使用。

版权所有：腾讯云计算（北京）有限责任公司

您可以在 `TUIGroup` 组件的 TUIGroup/Resources/TUIGroupTheme.bundle 下的任一主题文件夹中的 manifest.plist 文件中看到对应样式的 key。

UI 样式表



样式 key	样式释义
group_modify_view_bg_color	群/个人信息修改界面，背景色
group_modify_container_view_bg_color	群/个人信息修改界面，容器背景色
group_modify_title_color	群/个人信息修改界面，标题文本颜色
group_modify_desc_color	群/个人信息修改界面，描述信息文本颜色
group_modify_input_bg_color	群/个人信息修改界面，输入框背景颜色

版权所有：腾讯云计算（北京）有限责任公司

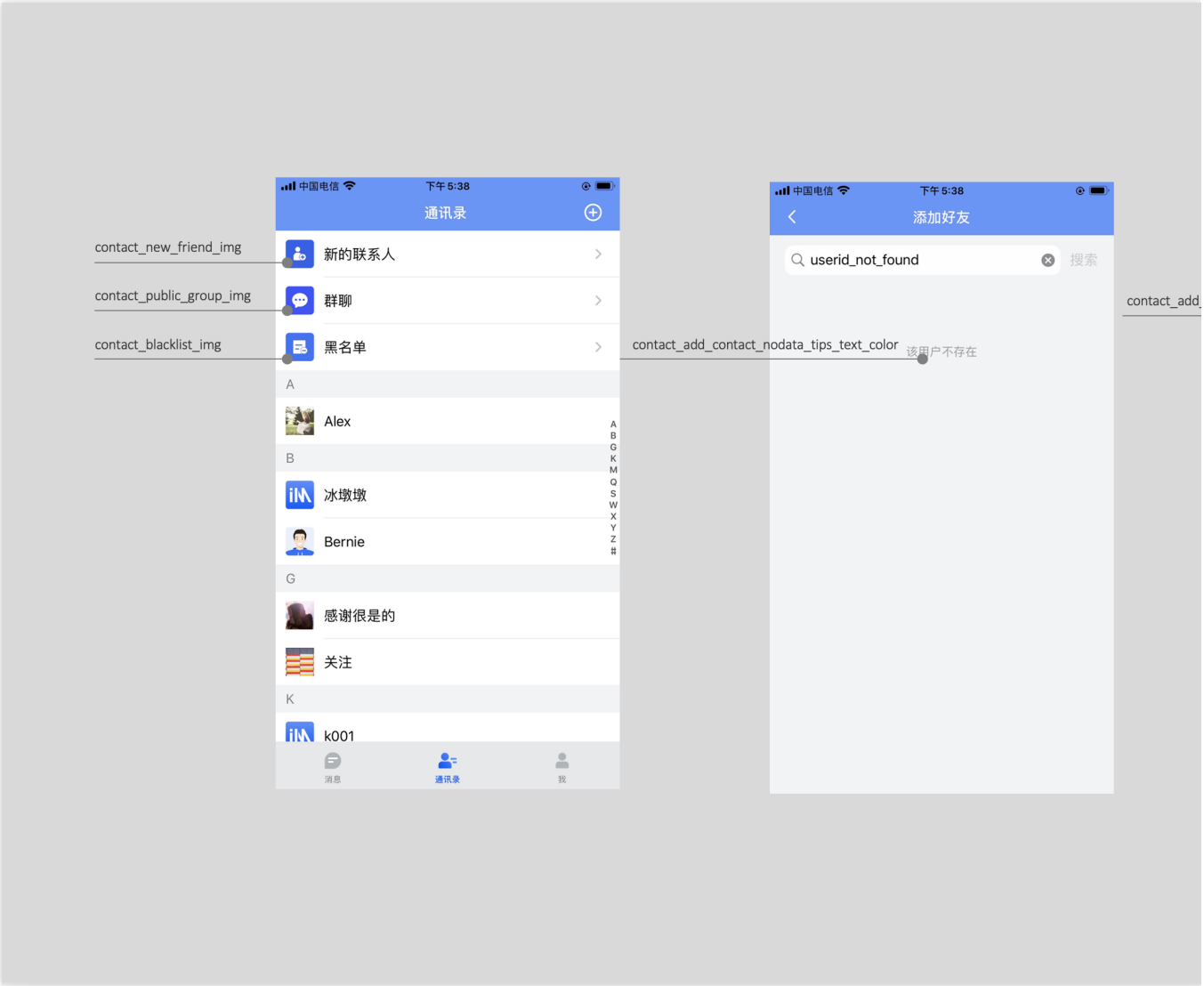
group_modify_input_text_color	群/个人信息修改界面，输入框文本颜色
group_modify_confirm_enable_bg_color	群/个人信息修改界面，确认按钮可点击状态的背景颜色
group_modify_confirm_disable_bg_color	群/个人信息修改界面，确认按钮不可点击状态的背景颜色

Contact 页面样式

存储位置

Contact 页面样式均存在于 `TUIContact` 组件中，供 Contact 页面使用。
您可以在 `TUIContact` 组件的 TUIContact/Resources/TUIContactTheme.bundle 下的任一主题文件夹中的 manifest.plist 文件中看到对应样式的 key。

UI 样式表



样式 key	样式释义
contact_new_friend_img	通讯录页面，新的联系人图标
contact_blacklist_img	通讯录页面，黑名单图标
contact_public_group_img	通讯录页面，群聊图标
contact_add_contact_tips_text_color	添加好友页面，我的用户 ID 提示文本的颜色
contact_add_contact_nodata_tips_text_color	添加好友页面，查找的用户不存在时的提示文本颜色

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2023-05-29 15:10:45

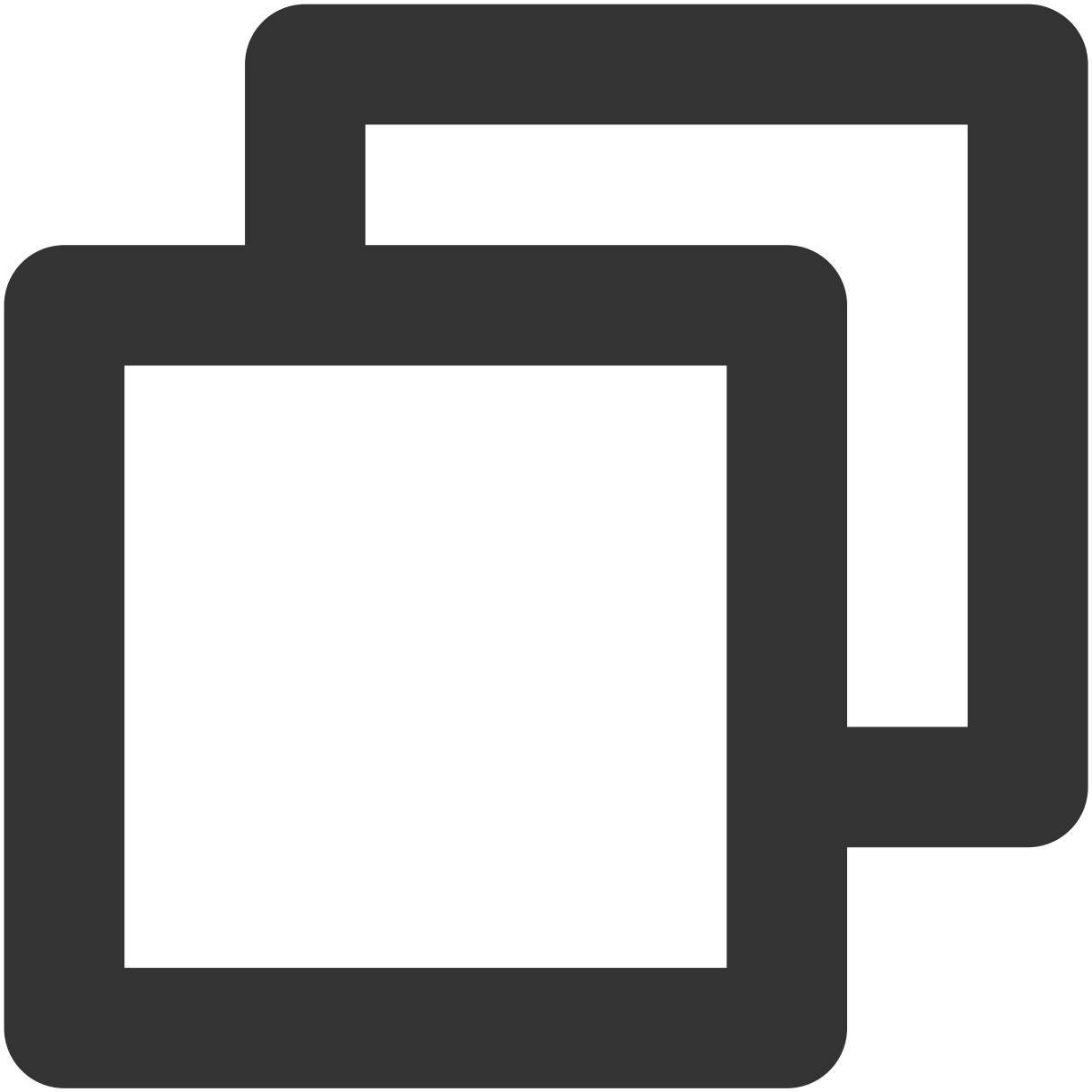
功能描述

TUIKit 从 v1.1.0 版本开始，主题风格颜色能力得到大幅度完善。
TUIKit默认提供颜色配置，您可以直接使用，无需任何配置。
但同时，您也可以很方便的自定义，TUIKit界面中，各处众多颜色配置。

自定义方式

步骤一：定义 TUIKit 颜色对象

在此对象中，您可以定义TUIKit界面中，各处的颜色配置。
请直接实例化一个 `TUITheme()` 对象。并修改里面的各个参数，以覆盖默认值，使用自定义颜色。
该对象构造函数，可配置的颜色有如下：



```
// 应用主色
// Primary Color For The App
final Color? primaryColor;

// 应用次色
// Secondary Color For The App
final Color? secondaryColor;

// 提示颜色，用于次级操作或提示
// Info Color, Used For Secondary Action Or Info
final Color? infoColor;
```

```
// 浅背景颜色，比主背景颜色浅，用于填充缝隙或阴影
// Weak Background Color, Lighter Than Main Background, Used For Marginal Space O
final Color? weakBackgroundColor;

// 宽屏幕：浅白背景颜色，比浅背景颜色浅
// Weak Background Color, Lighter Than Main Background, Used For Marginal Space O
final Color? wideBackgroundColor;

// 浅分割线颜色，用于分割线或边框
// Weak Divider Color, Used For Divider Or Border
final Color? weakDividerColor;

// 浅字色
// Weak Text Color
final Color? weakTextColor;

// 深字色
// Dark Text Color
final Color? darkTextColor;

// 浅主色，用于AppBar或Panels
// Light Primary Color, Used For AppBar Or Several Panels
final Color? lightPrimaryColor;

// 字色
// TextColor
final Color? textColor;

// 警示色，用于危险操作
// Caution Color, Used For Warning Actions
final Color? cautionColor;

// 群主标识色
// Group Owner Identification Color
final Color? ownerColor;

// 群管理员标识色
// Group Admin Identification Color
final Color? adminColor;

// 白色
// white
final Color? white;

// 黑色
// black
```

```
  final Color? black;

  // 输入框填充色
  // input fill color
  final Color? inputFillColor;

  // 灰色文本
  // grey text color
  final Color? textgrey;

  /// 新版本颜色从这里开始
  ///
  /// Appbar 背景颜色
  final Color? appbarBgColor;

  /// Appbar 文字颜色
  final Color? appbarTextColor;

  /// 消息列表多选面板背景颜色
  final Color? selectPanelBgColor;

  /// 消息列表多选面板文字及icon颜色
  final Color? selectPanelTextIconColor;
  /// 会话列表背景颜色
  final Color? conversationItemBgColor;

  /// 会话列表边框颜色
  final Color? conversationItemBorderColor;

  /// 会话列表选中背景颜色
  final Color? conversationItemActiveBgColor;

  /// 会话列表置顶背景颜色
  final Color? conversationItemPinedBgColor;

  /// 会话列表Title字体颜色
  final Color? conversationItemTitleTextColor;

  /// 会话列表LastMessage字体颜色
  final Color? conversationItemLastMessageTextColor;

  /// 会话列表Time字体颜色
  final Color? conversationItemTitmeTextColor;

  /// 会话列表用户在线状态背景色
  final Color? conversationItemOnlineStatusBgColor;
```

```
  /// 会话列表用户离线状态背景色
  final Color? conversationItemOfflineStatusBgColor;

  /// 会话列表未读数背景颜色
  final Color? conversationItemUnreadCountBgColor;

  /// 会话列表未读数字体颜色
  final Color? conversationItemUnreadCountTextColor;

  /// 会话列表草稿字体颜色
  final Color? conversationItemDraftTextColor;

  /// 会话列表收到消息不提醒Icon颜色
  final Color? conversationItemNoNotificationIconColor;

  /// 会话列表侧滑按钮字体颜色
  final Color? conversationItemSliderTextColor;

  /// 会话列表侧滑按钮Clear背景颜色
  final Color? conversationItemSliderClearBgColor;

  /// 会话列表侧滑按钮Pin背景颜色
  final Color? conversationItemSliderPinBgColor;

  /// 会话列表侧滑按钮Delete背景颜色
  final Color? conversationItemSliderDeleteBgColor;

  /// 会话列表宽屏模式选中时背景颜色
  final Color? conversationItemChooseBgColor;

  /// 聊天页背景颜色
  final Color? chatBgColor;

  /// 聊天页背景颜色
  final Color? chatTimeDividerTextColor;

  /// 聊天页导航栏背景颜色
  final Color? chatHeaderBgColor;

  /// 聊天页导航栏Title字体颜色
  final Color? chatHeaderTitleTextColor;

  /// 聊天页导航栏Back字体颜色
  final Color? chatHeaderBackTextColor;

  /// 聊天页导航栏Action字体颜色
  final Color? chatHeaderActionTextColor;
```

```
/// 聊天页历史消息列表字体颜色
final Color? chatMessageItemTextColor;

/// 聊天页历史消息列表来自自己时背景颜色
final Color? chatMessageItemFromSelfBgColor;

/// 聊天页历史消息列表来自非自己时背景颜色
final Color? chatMessageItemFromOthersBgColor;

/// 聊天页历史消息列表已读状态字体颜色
final Color? chatMessageItemUnreadStatusTextColor;

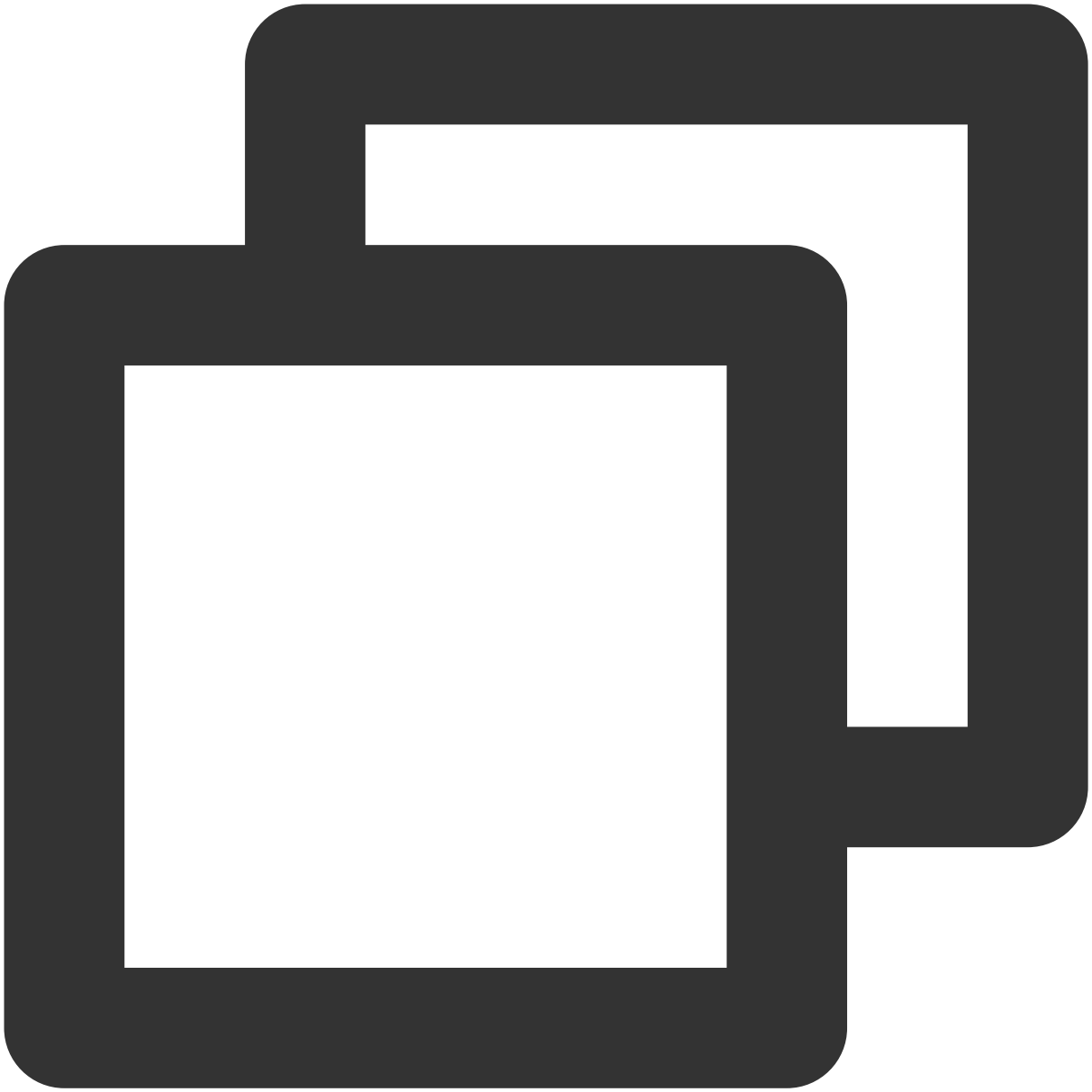
/// 聊天页历史消息列表小舌头背景颜色
final Color? chatMessageTongueBgColor;

/// 聊天页历史消息列表小舌头字体颜色
final Color? chatMessageTongueTextColor;
```

步骤二：启用配置

调用 TUIKit 提供的 `setTheme` 方法，传入上一步定义的颜色对象 `TUITheme()` 即可。

该方法可随时调用，动态修改。



```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.setTheme(theme: TUITheme());
```

联系我们

如果您在接入使用过程中有任何疑问，请通过如下方式联系我们。

Telegram Group

WhatsApp Group

版权所有：腾讯云计算（北京）有限责任公司

设置界面风格

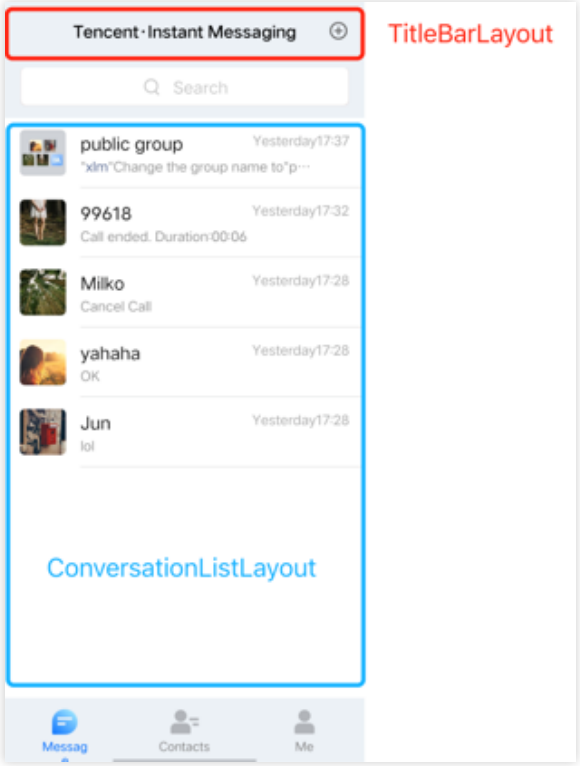
Android

最近更新时间：2023-10-07 09:22:53

本文介绍如何设置 Android 界面风格。

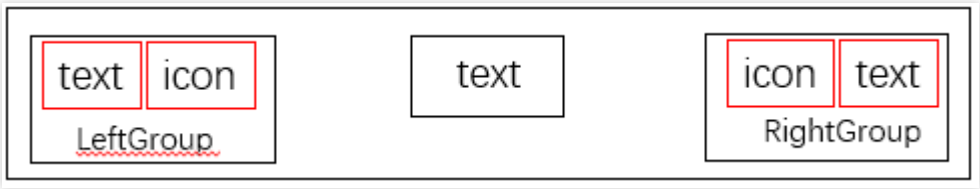
设置会话列表

会话列表由标题区 TitleBarLayout 与列表区 ConversationListLayout 组成，每部分都提供了 UI 样式以及事件注册的接口可供修改。



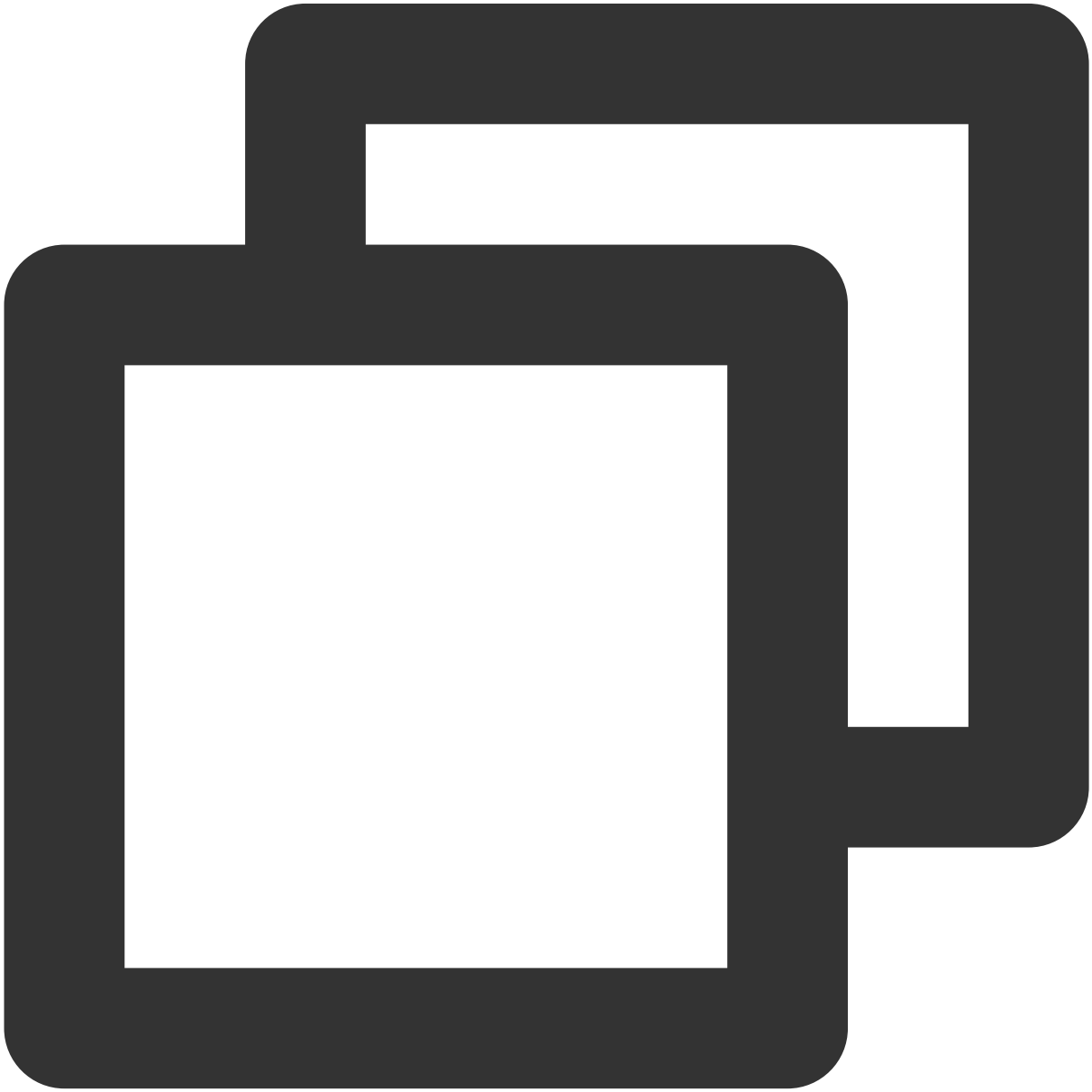
设置标题样式

标题区除了本身作为 View 所具有的属性功能之外，还包含左、中、右三块区域，如下图所示：



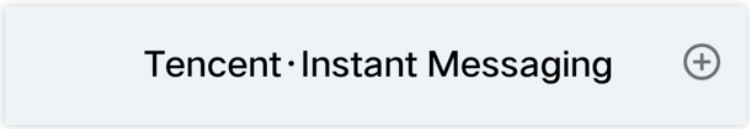
您可参见 ITitleBarLayout 进行自定义修改。

例如，在会话列表中，隐藏左边的 LeftGroup，设置中间的标题，隐藏右边的文本和图片按钮，代码如下：（参见 MainActivity 中的实现）

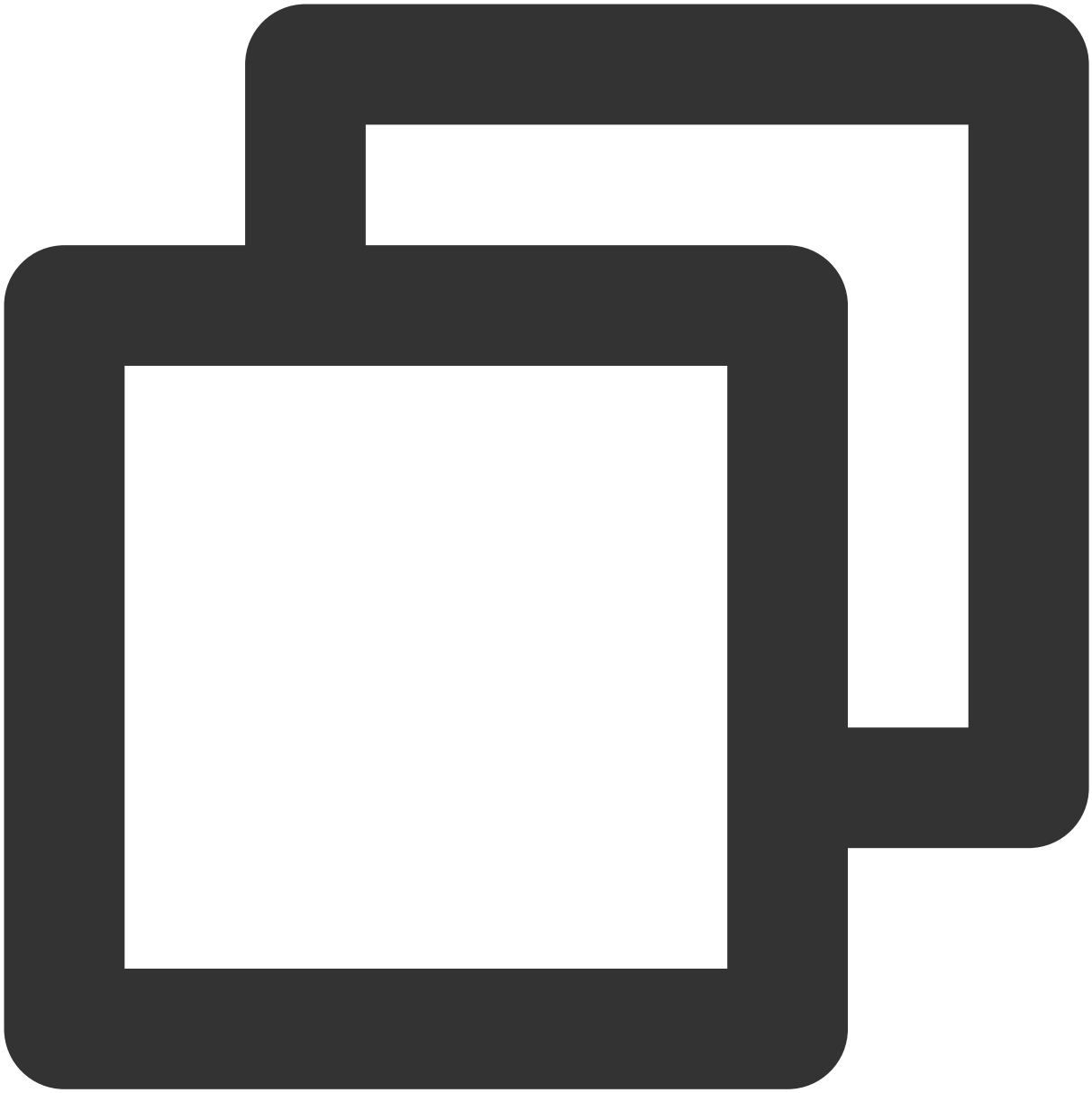


```
mainTitleBar.setTitle(getResources().getString(R.string.conversation_title), ITitle
mainTitleBar.getLeftGroup().setVisibility(View.GONE);
mainTitleBar.getRightGroup().setVisibility(View.VISIBLE);
mainTitleBar.setRightIcon(R.drawable.more_btn);
```

效果如下图所示：



另外，您也可以定制点击事件：

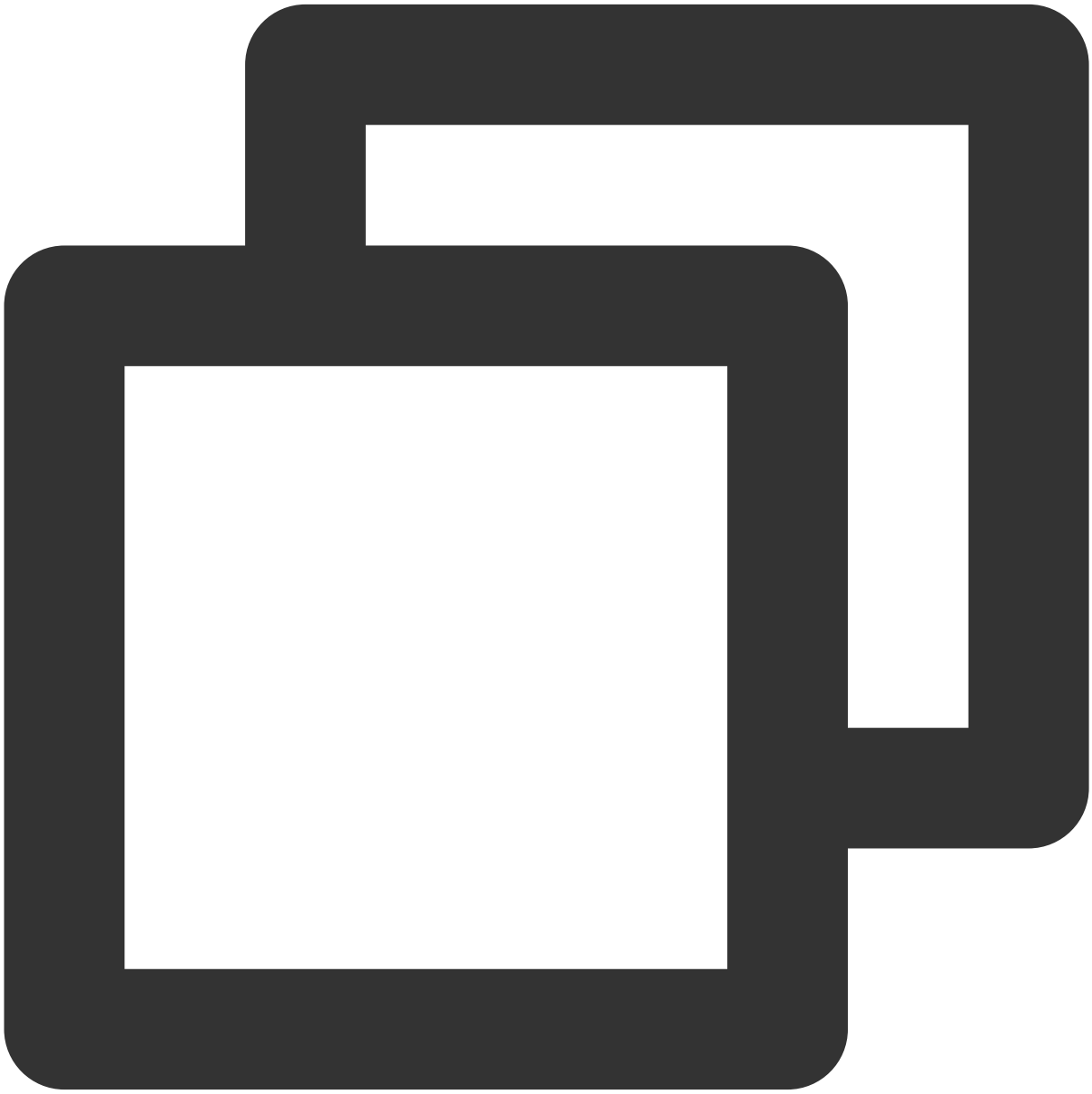


```
Menu menu = new Menu(this, mainTitleBar);
mainTitleBar.setOnRightClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (menu == null) {
            return;
        }
        if (menu.isShowing()) {
            menu.hide();
        } else {
            menu.show();
```

```
            }
        }
    });
```

设置会话列表样式

列表区的自定义 layout 继承自 RecyclerView，登录后 TUIKit 会根据用户名从 SDK 读取该用户的会话列表。
会话列表提供一些常用功能定制，例如，头像是否圆角、背景、字体大小、点击与长按事件等。示例代码如下：



```
public static void customizeConversation(final ConversationLayout layout) {
    // 从 ConversationLayout 获取会话列表
```

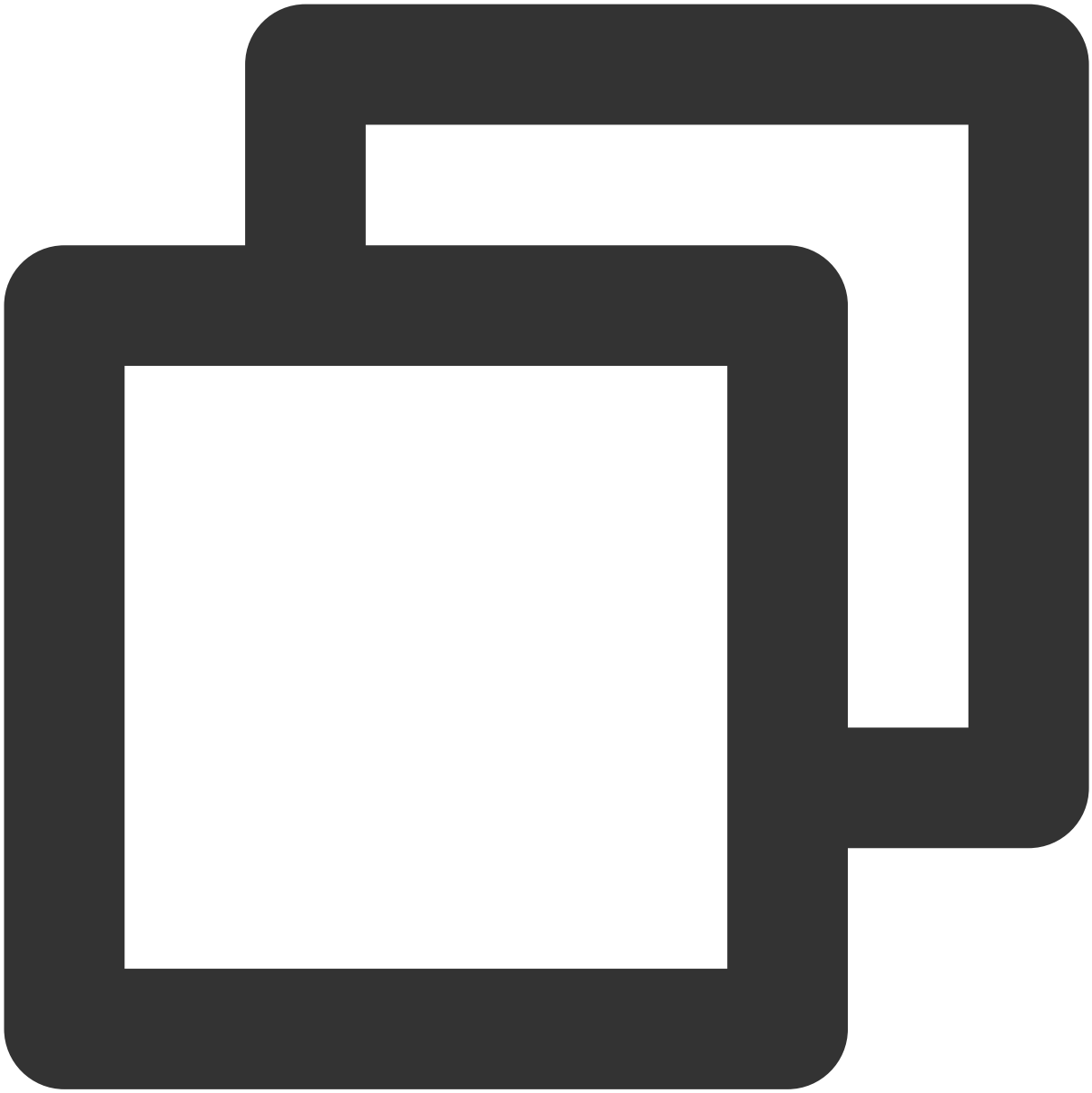
```
    ConversationListLayout listLayout = layout.getConversationList();
    listLayout.setItemTopTextSize(16); // 设置 item 中 top 文字大小
    listLayout.setItemBottomTextSize(12);// 设置 item 中 bottom 文字大小
    listLayout.setItemDateTextSize(10);// 设置 item 中 timeline 文字大小
    listLayout.setItemAvatarRadius(5); // 设置 adapter item 头像圆角大小
    listLayout.disableItemUnreadDot(false);// 设置 item 是否不显示未读红点，默认显示
    // 长按弹出菜单
    listLayout.setOnItemLongClickListener(new ConversationListLayout.OnItemLongClic
        @Override
        public void OnItemLongClick(View view, int position, ConversationInfo conve
            startPopShow(view, position, conversationInfo);
        }
    });
}
```

更多详细信息请参见 ConversationLayoutSetting.java。

设置头像的风格样式

IM SDK 不做头像存储，需要集成者有头像存储接口获取头像 URL，这里 TUIKit 通过随机头像接口进行演示，如何设置头像。

首先您需要在个人资料页面中，上传头像图片，调用修改资料接口。

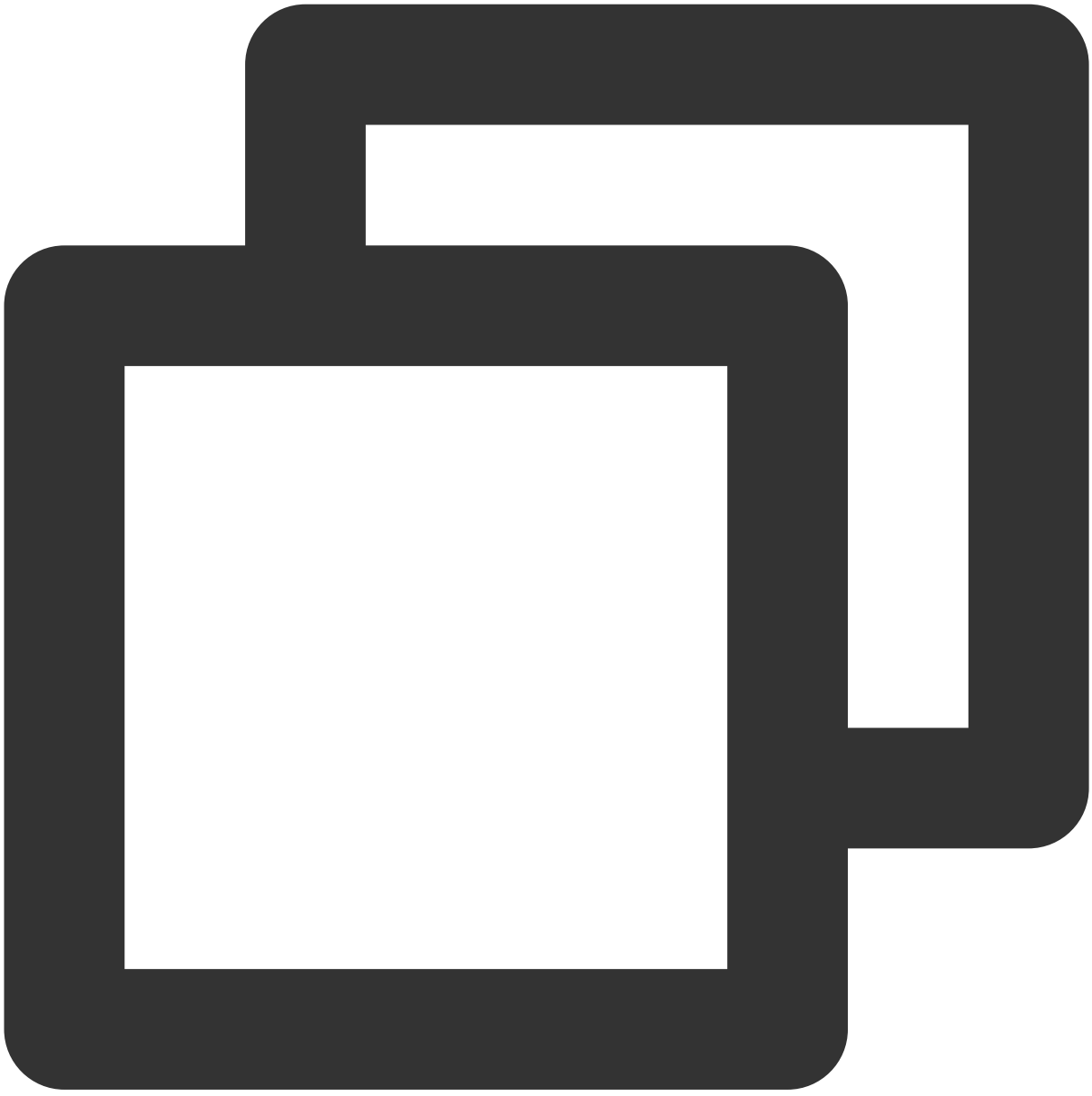


```
V2TIMUserFullInfo v2TIMUserFullInfo = new V2TIMUserFullInfo();
// 头像
if (!TextUtils.isEmpty(mIconUrl)) {
    v2TIMUserFullInfo.setFaceUrl(mIconUrl);
}
V2TIMManager.getInstance().setSelfInfo(v2TIMUserFullInfo, new V2TIMCallback() {
    @Override
    public void onError(int code, String desc) {
    }

    @Override
```

```
    public void onSuccess() {
    }
});
```

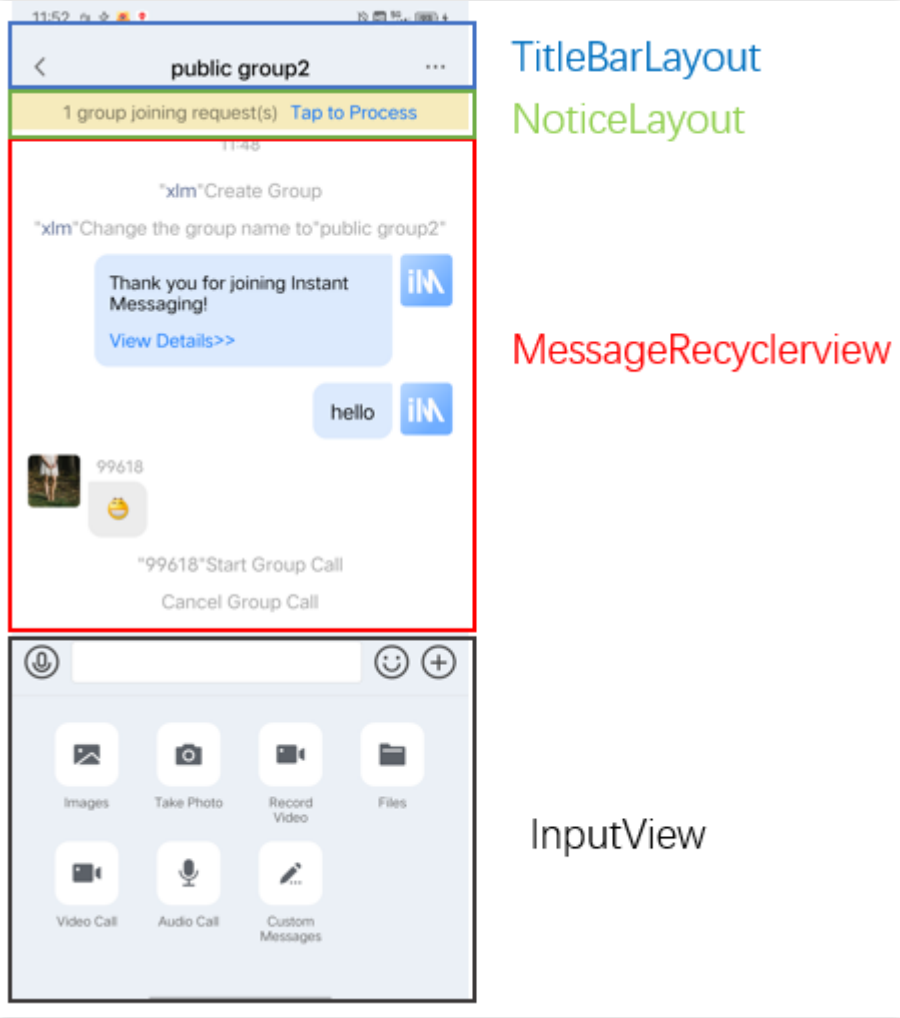
会话列表设置头像ConversationCommonHolder.java中进行获取展示：

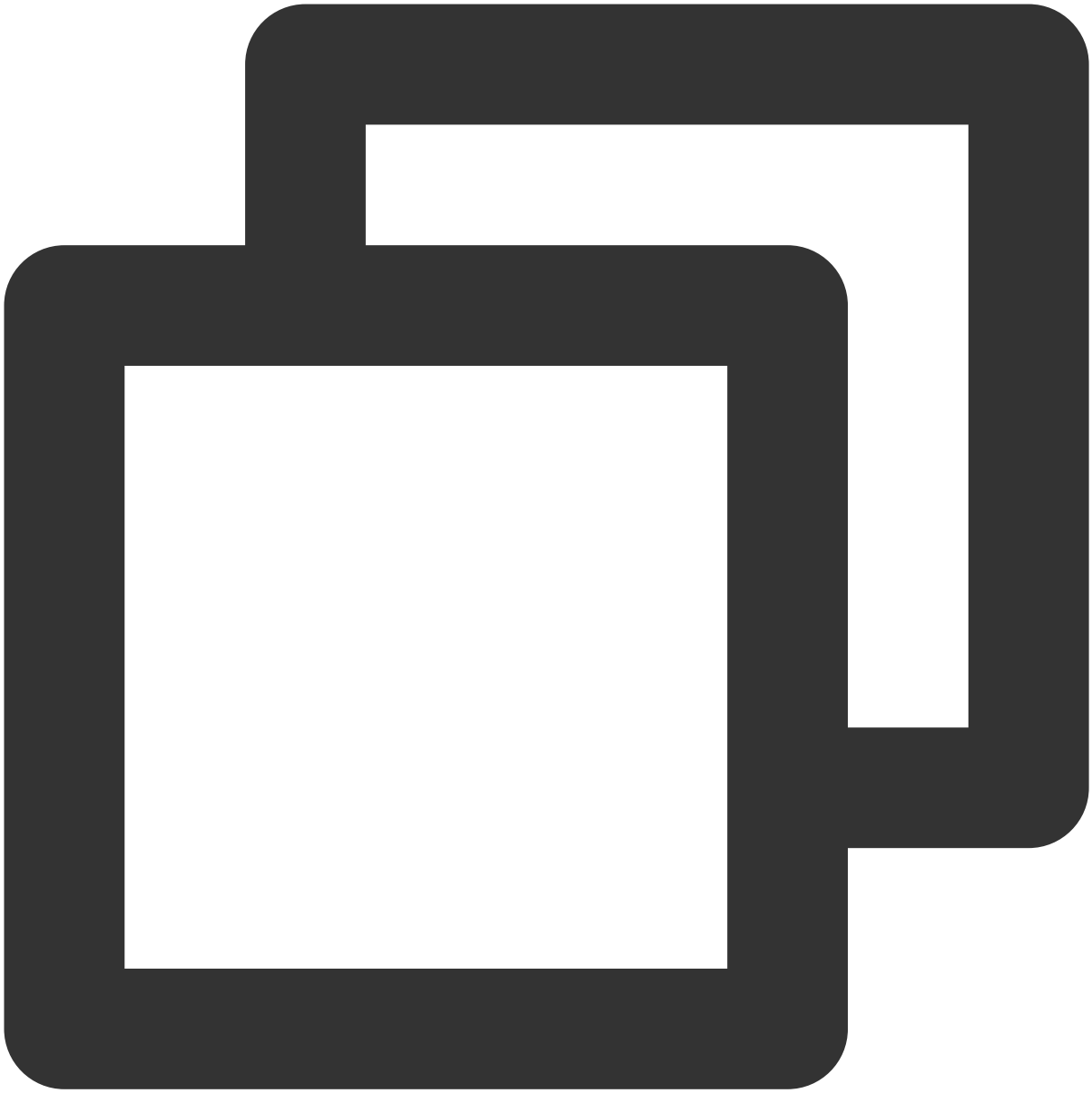


```
conversationIconView.setConversation(conversation);
```

设置聊天窗口的样式

聊天窗口包含标题区 TitleBarLayout，用法与会话列表相同。除此之外，聊天窗口还包含三个区域，从上到下为通知区 NoticeLayout、消息区 MessageRecyclerView 和输入区 InputView，如下图所示：





```
/**
 * 获取聊天窗口 Input 区域 Layout
 *
 * @return
 */
InputView getInputLayout();
/**
 * 获取聊天窗口 Message 区域 Layout
 *
 * @return
 */
```

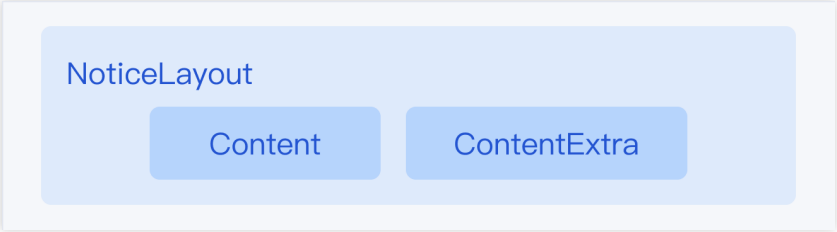
```
MessageRecyclerView getMessageLayout();

/**
 * 获取聊天窗口 Notice 区域 Layout
 *
 * @return
 */
NoticeLayout getNoticeLayout();
```

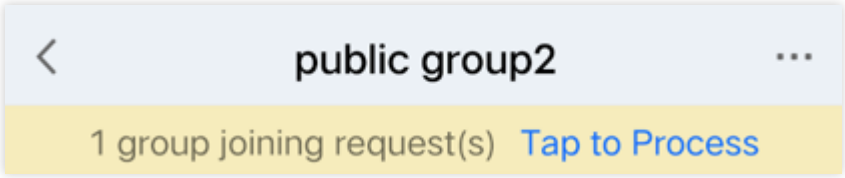
更多详细信息请参见 ChatLayoutSetting.java。

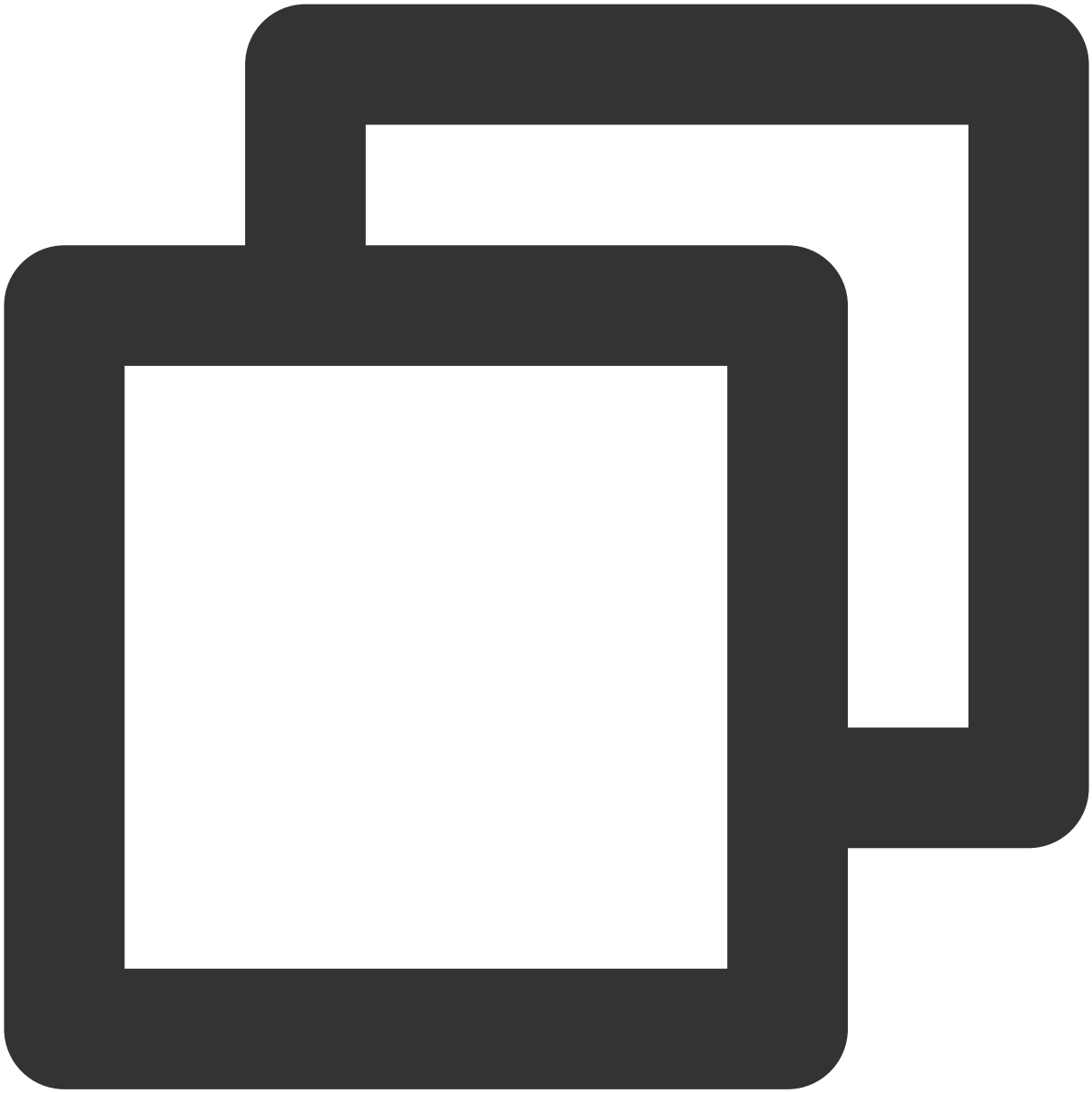
设置通知区域 NoticeLayout 样式

通知区域由两个 TextView 组成，如下图所示：



效果如下图所示：



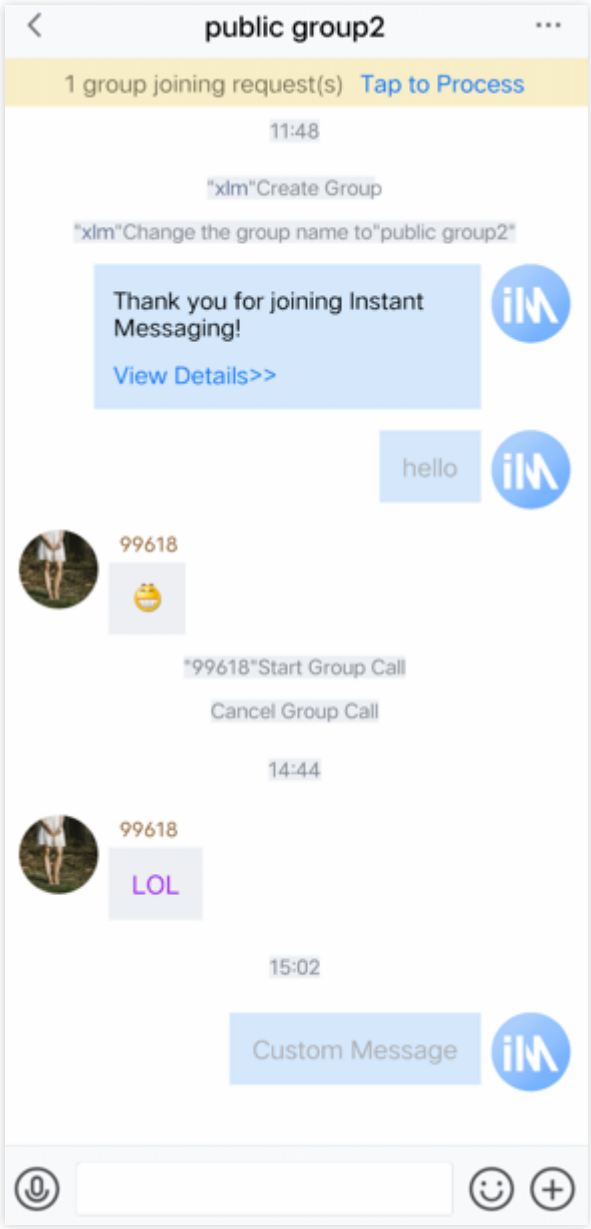


```
// 从 ChatView 获取 NoticeLayout
NoticeLayout noticeLayout = layout.getNoticeLayout();
// 可以使通知区域一致展示
noticeLayout.alwaysShow(true);
// 设置通知主题
noticeLayout.getContent().setText("现在插播一条广告");
// 设置通知提醒文字
noticeLayout.getContentExtra().setText("参看有奖");
// 设置通知的点击事件
noticeLayout.setOnNoticeClickListener(new View.OnClickListener() {
    @Override
```

```
    public void onClick(View v) {
        ToastUtil.toastShortMessage("赏白银五千两");
    }
});
```

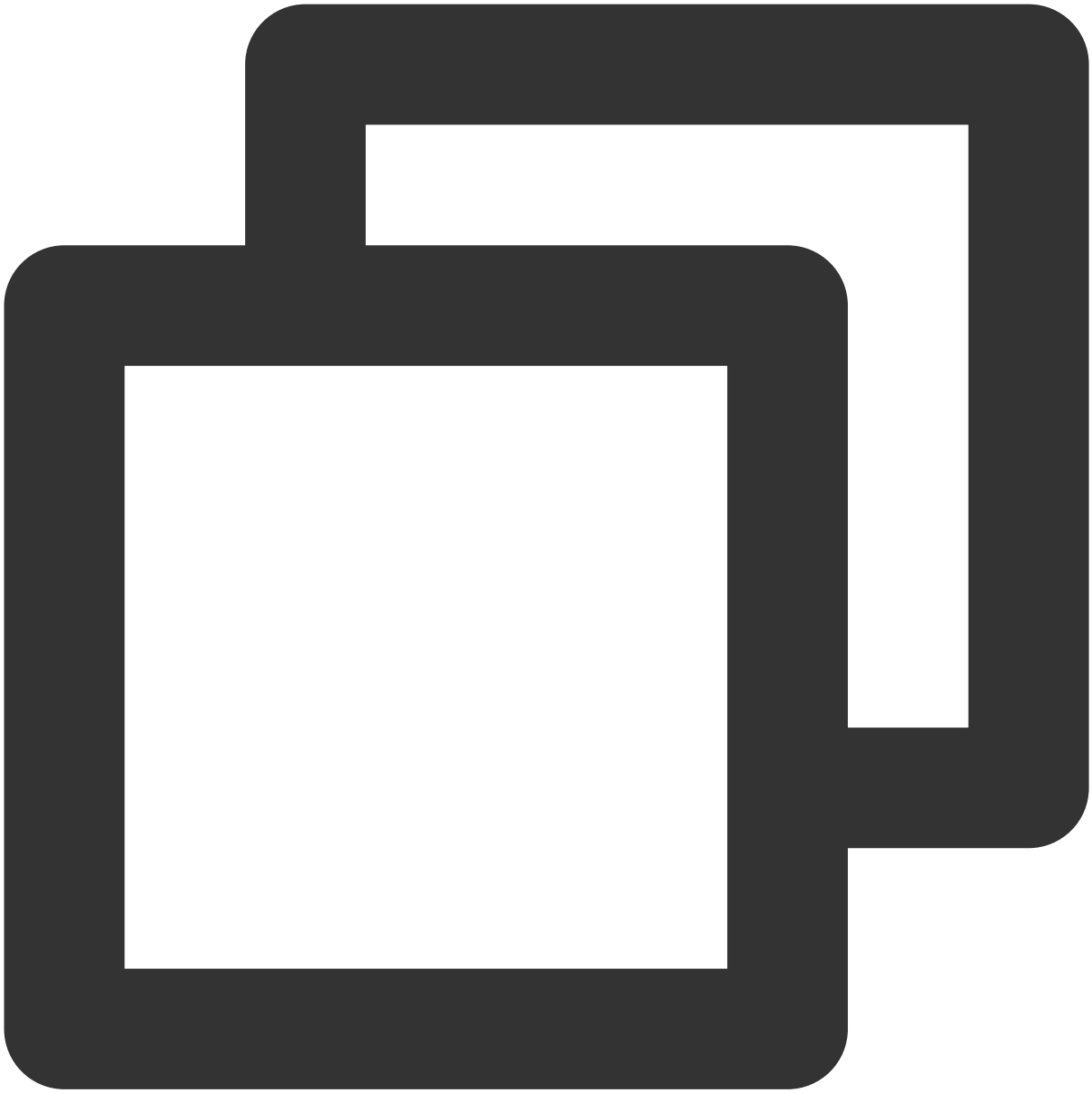
设置消息区域 MessageRecyclerView 样式

MessageRecyclerView 继承自 RecyclerView，本文提供自定义修改聊天背景、气泡、文字、是否显示昵称等常见的用法，更多详情请参见 IMessageProperties.java。



设置聊天窗口的背景色

您可以自定义设置聊天背景色，示例代码如下：

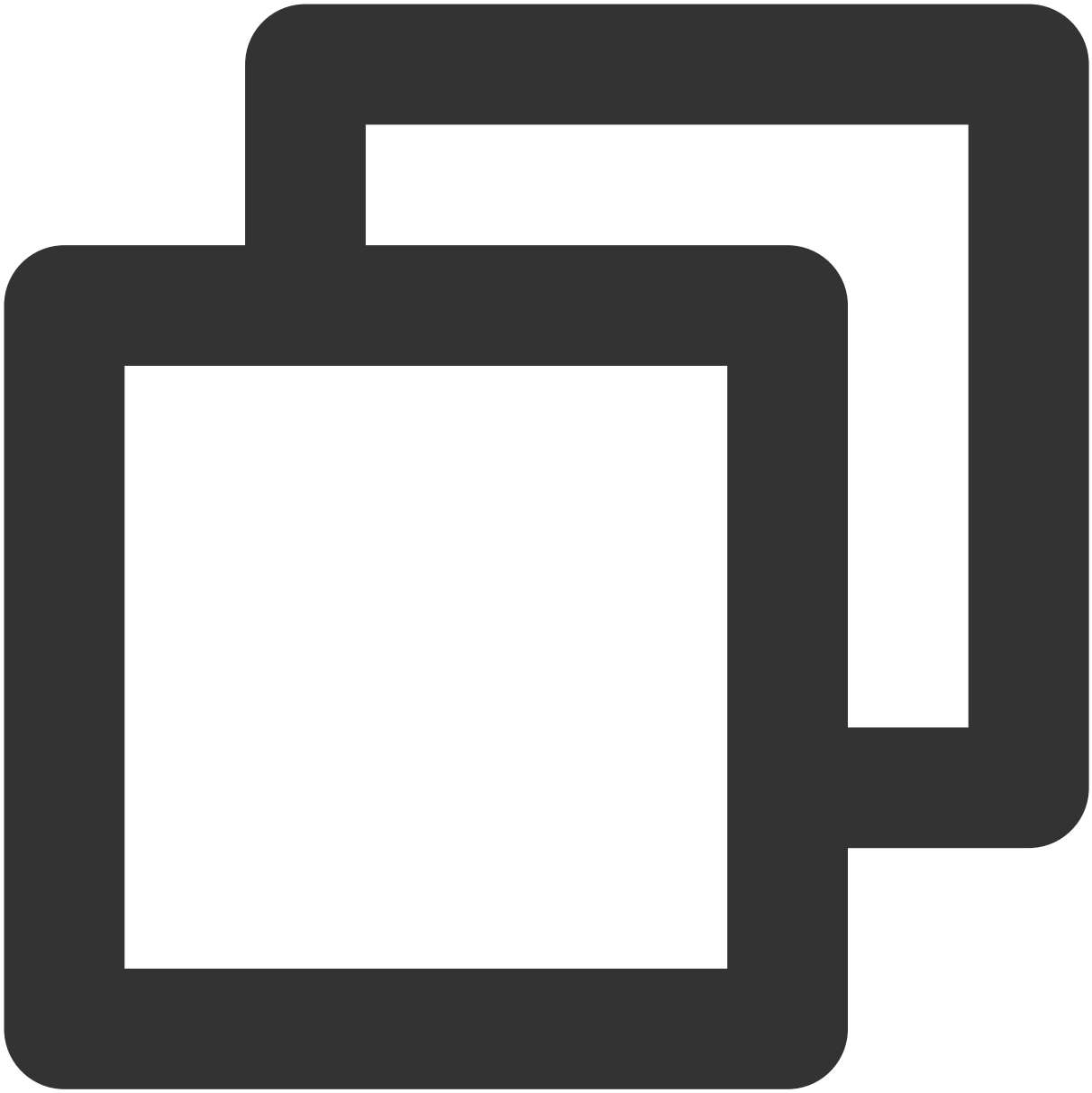


```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
////// 设置聊天背景 //////
messageRecyclerView.setBackground(new ColorDrawable(0xB0E2FF00));
```

设置发送者的头像样式

TUIKit 的界面在显示用户时，会从用户资料中读取头像地址并显示。

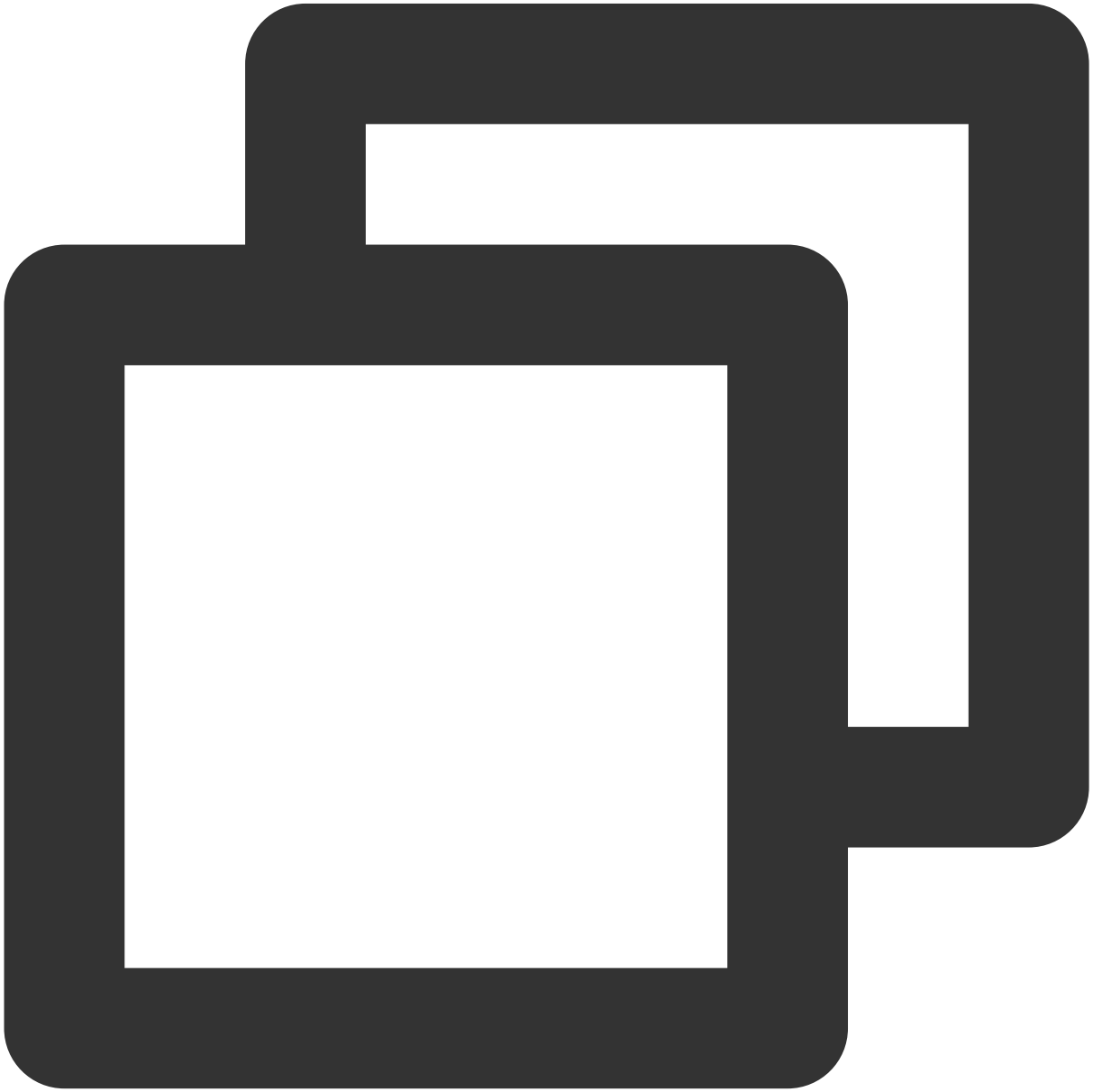
示例代码如下：



```
// 聊天界面设置头像
if (!TextUtils.isEmpty(msg.getFaceUrl())) {
    List<Object> urllist = new ArrayList<>();
    urllist.add(msg.getFaceUrl());
    if (isForwardMode) {
        leftUserIcon.setIconUrls(urllist);
    } else {
        if (msg.isSelf()) {
            rightUserIcon.setIconUrls(urllist);
        } else {
            leftUserIcon.setIconUrls(urllist);
```

```
        }
    }
} else {
    rightUserIcon.setIconUrls(null);
    leftUserIcon.setIconUrls(null);
}
```

如果用户没有设置头像会显示默认头像，您可以自定义设置默认头像、头像是否圆角以及头像大小等。
示例代码如下：

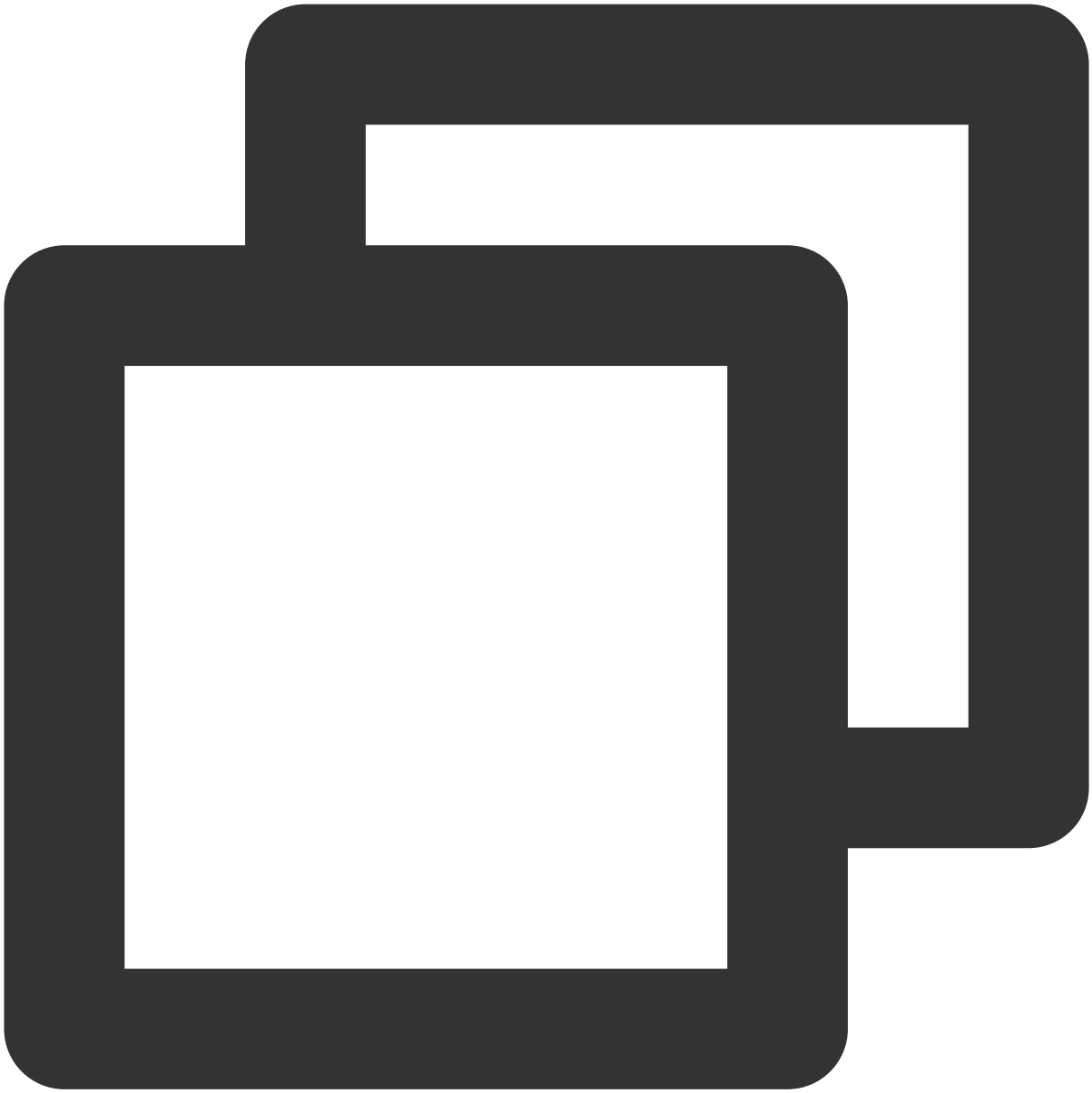


```
// 从 ChatView 获取 MessageRecyclerView
```

```
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
////// 设置聊天背景 //////
messageRecyclerView.setBackground(new ColorDrawable(0xFFEFE5D4));
////// 设置头像 //////
// 设置默认头像，默认与朋友与自己的头像相同
messageRecyclerView.setAvatar(R.drawable.core_default_user_icon_light);
// 设置头像圆角
messageRecyclerView.setAvatarRadius(50);
// 设置头像大小
messageRecyclerView.setAvatarSize(new int[]{68, 68});
```

设置气泡的背景色

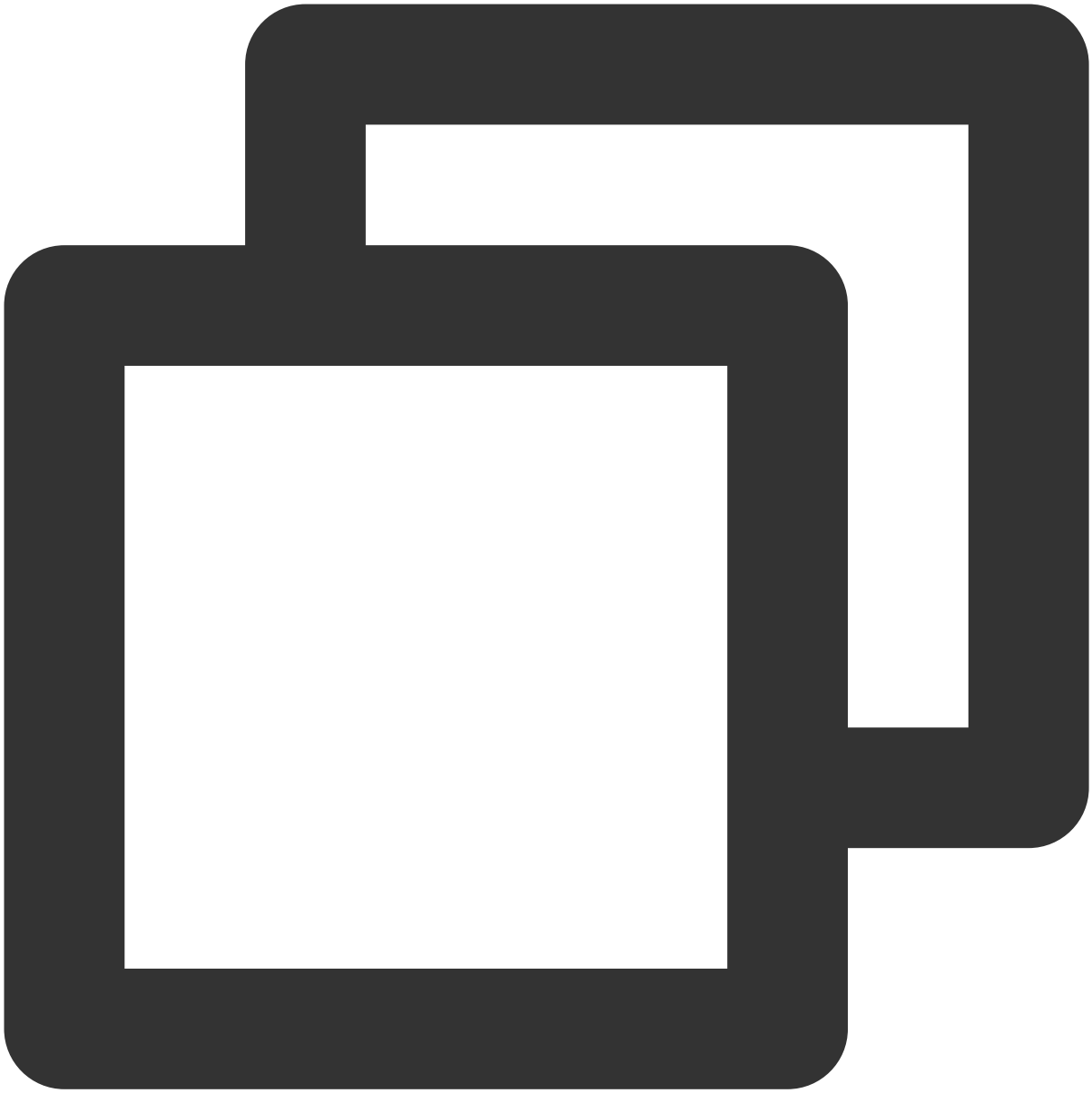
左边为对方的气泡，右边为自己的气泡，您可以自定义设置双方的气泡背景。
示例代码如下：



```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
// 设置自己聊天气泡的背景
messageRecyclerView.setRightBubble(context.getResources().getDrawable(R.drawable.ch
// 设置朋友聊天气泡的背景
messageRecyclerView.setLeftBubble(context.getResources().getDrawable(R.drawable.cha
```

您可以自定义设置昵称的字体大小与颜色等，但双方昵称样式必须保持一致。

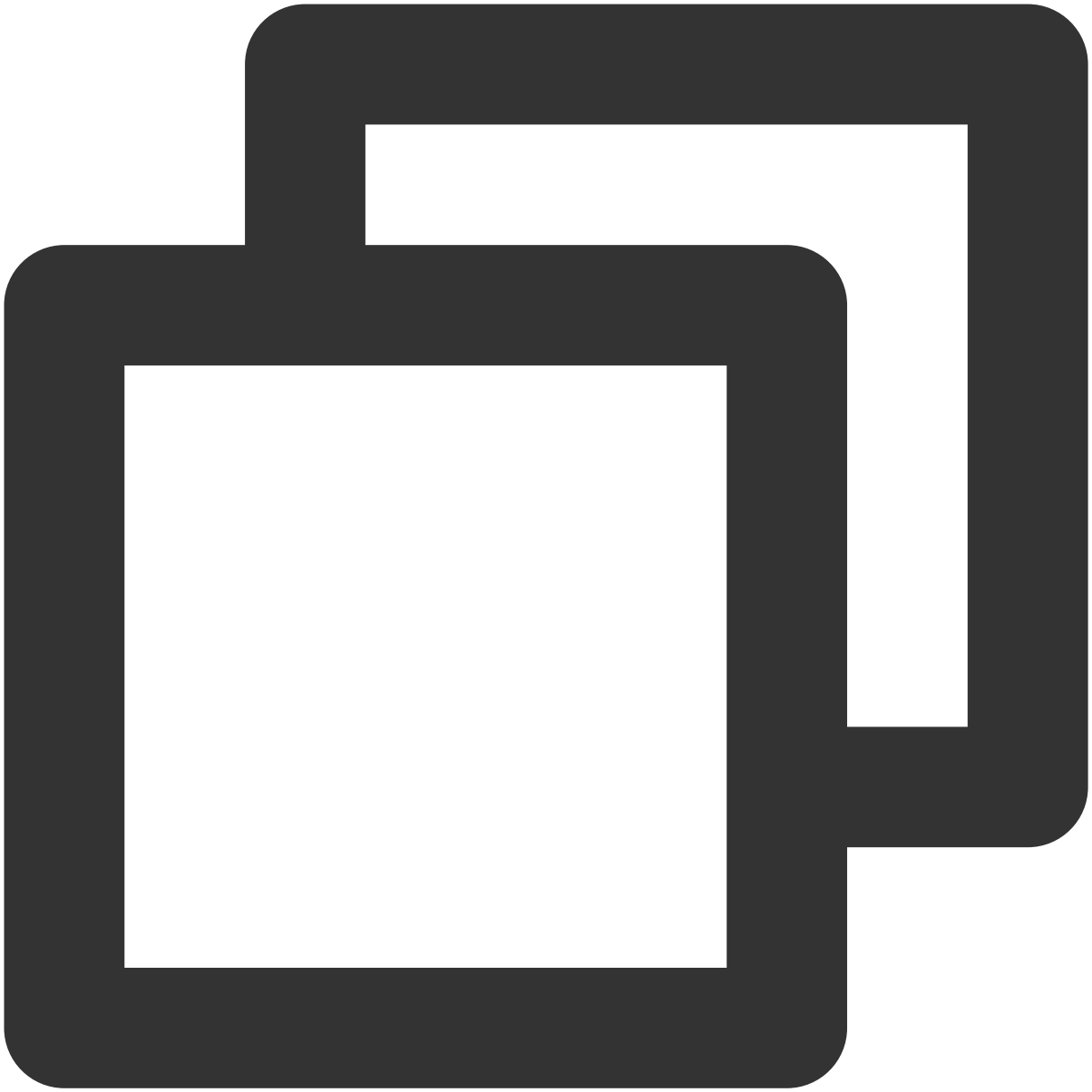
示例代码如下：



```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
////// 设置昵称样式（对方与自己的样式保持一致）//////
messageRecyclerView.setNameFontSize(12);
messageRecyclerView.setNameFontColor(0x8B5A2B00);
```

设置聊天内容样式

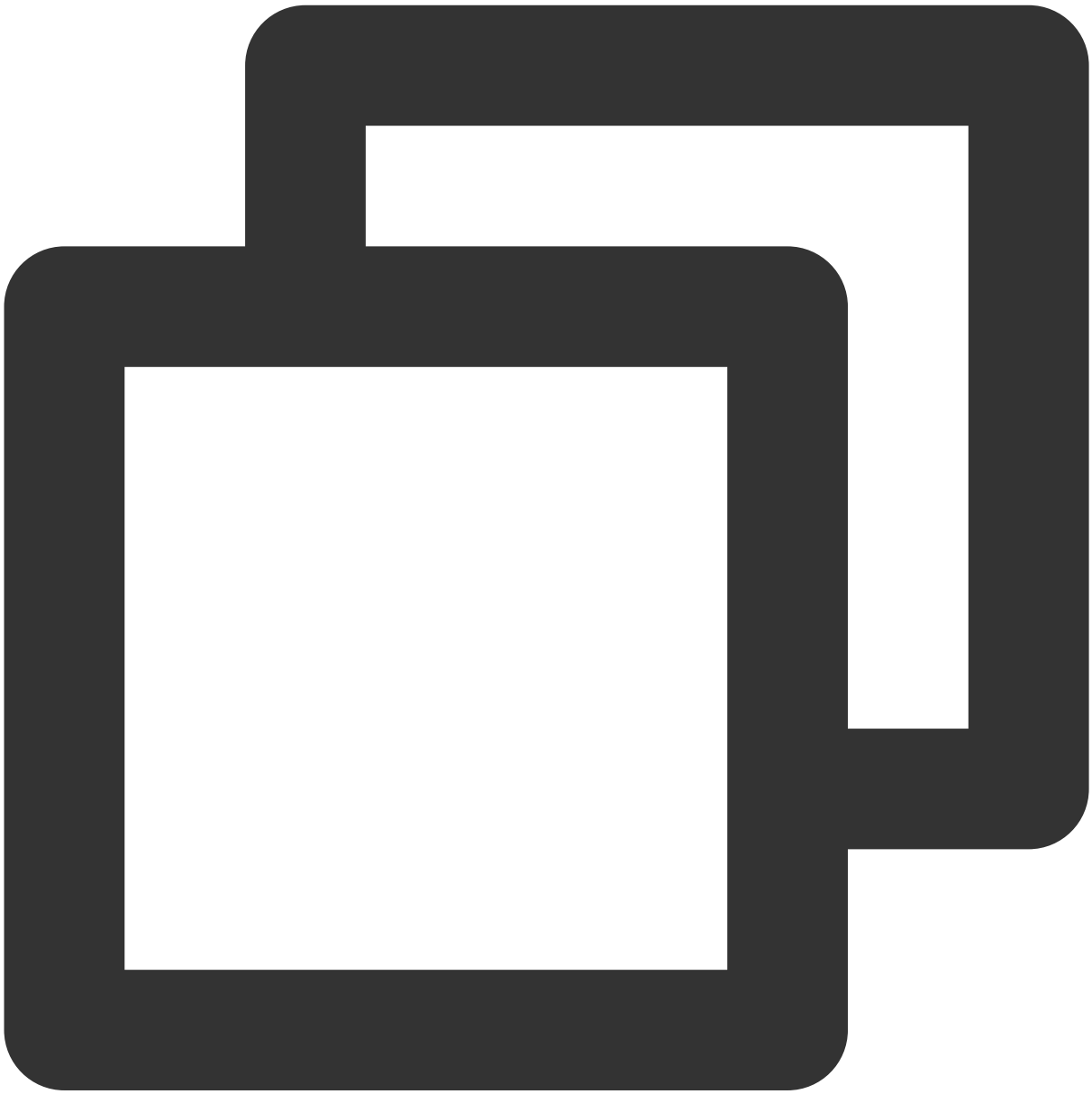
您可以自定义设置聊天内容的字体大小、双方字体颜色等，但双方字体大小必须保持一致。
示例代码如下：



```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
// 设置聊天内容字体大小，朋友和自己用一种字体大小
messageRecyclerView.setChatContextFontSize(15);
// 设置自己聊天内容字体颜色
messageRecyclerView.setRightChatContentFontColor(0xA9A9A900);
// 设置朋友聊天内容字体颜色
messageRecyclerView.setLeftChatContentFontColor(0xA020F000);
```

设置聊天时间线样式

您可以自定义设置聊天时间线的背景、字体大小以及字体颜色等。
示例代码如下：

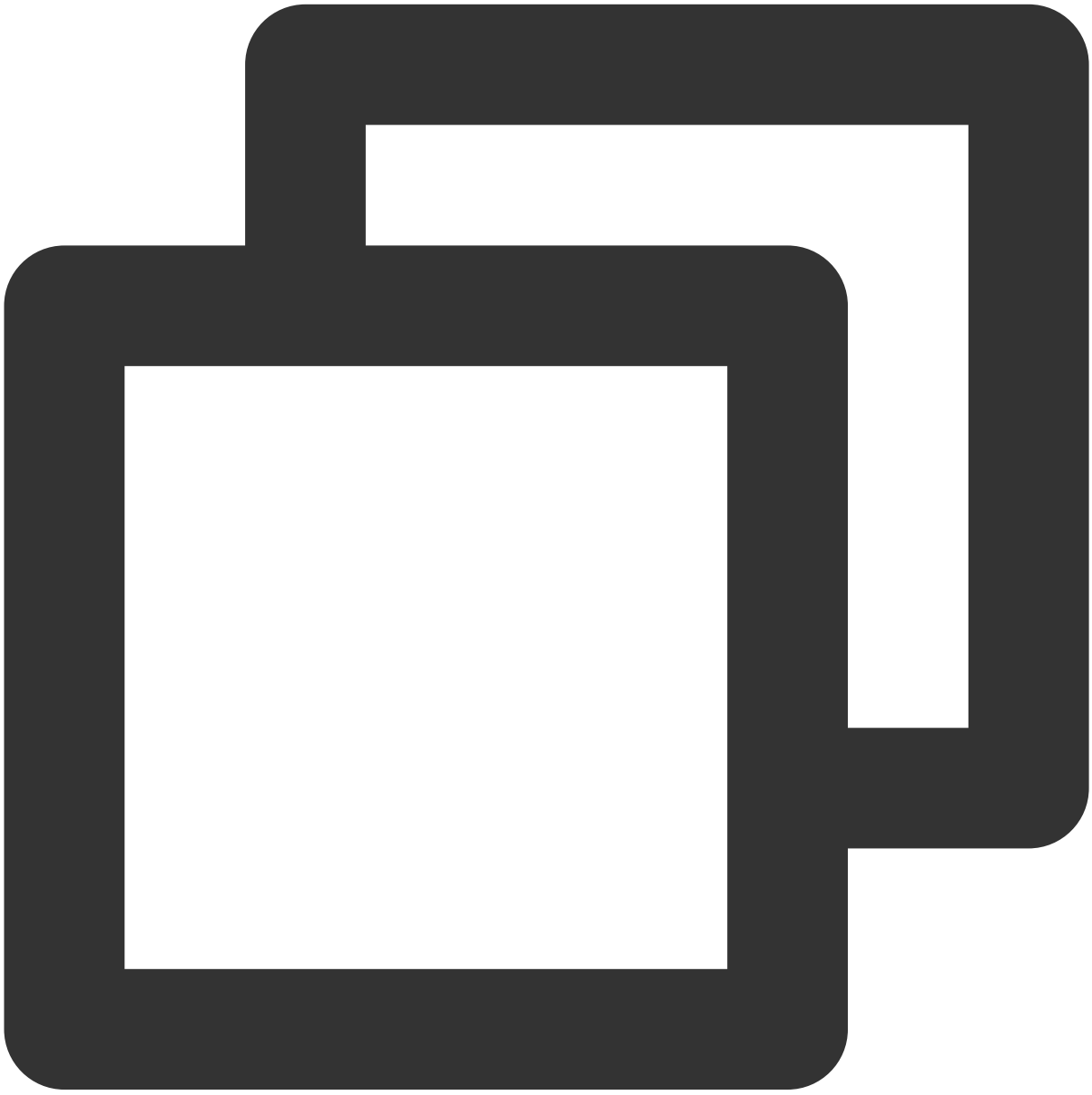


```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
// 设置聊天时间线的背景
messageRecyclerView.setChatTimeBubble(new ColorDrawable(0x8B691400));
// 设置聊天时间的字体大小
messageRecyclerView.setChatTimeFontSize(20);
// 设置聊天时间的字体颜色
messageRecyclerView.setChatTimeFontColor(0xEE00EE00);
```

版权所有：腾讯云计算（北京）有限责任公司

设置聊天的提示信息样式

您可以自定义设置提示信息的背景、字体大小以及字体颜色等。
示例代码如下：

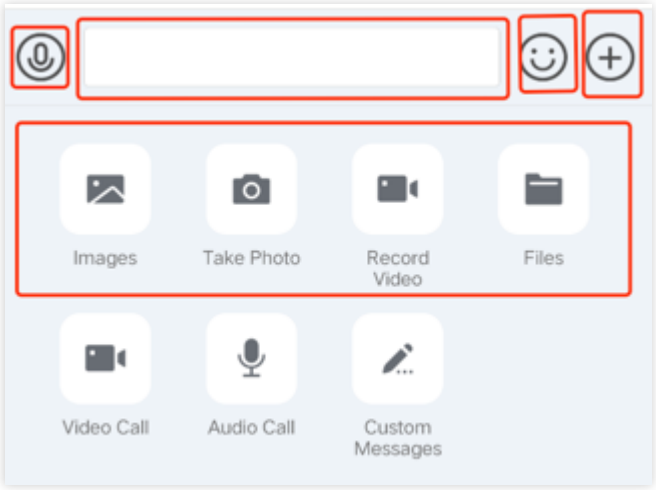


```
// 从 ChatView 获取 MessageRecyclerView
MessageRecyclerView messageRecyclerView = layout.getMessageLayout();
// 设置提示的背景
messageRecyclerView.setTipsMessageBubble(new ColorDrawable(0xA020F000));
// 设置提示的字体大小
messageRecyclerView.setTipsMessageFontSize(20);
```

```
// 设置提示的字体颜色
messageRecyclerView.setTipsMessageFontColor(0x7CFC0000);
```

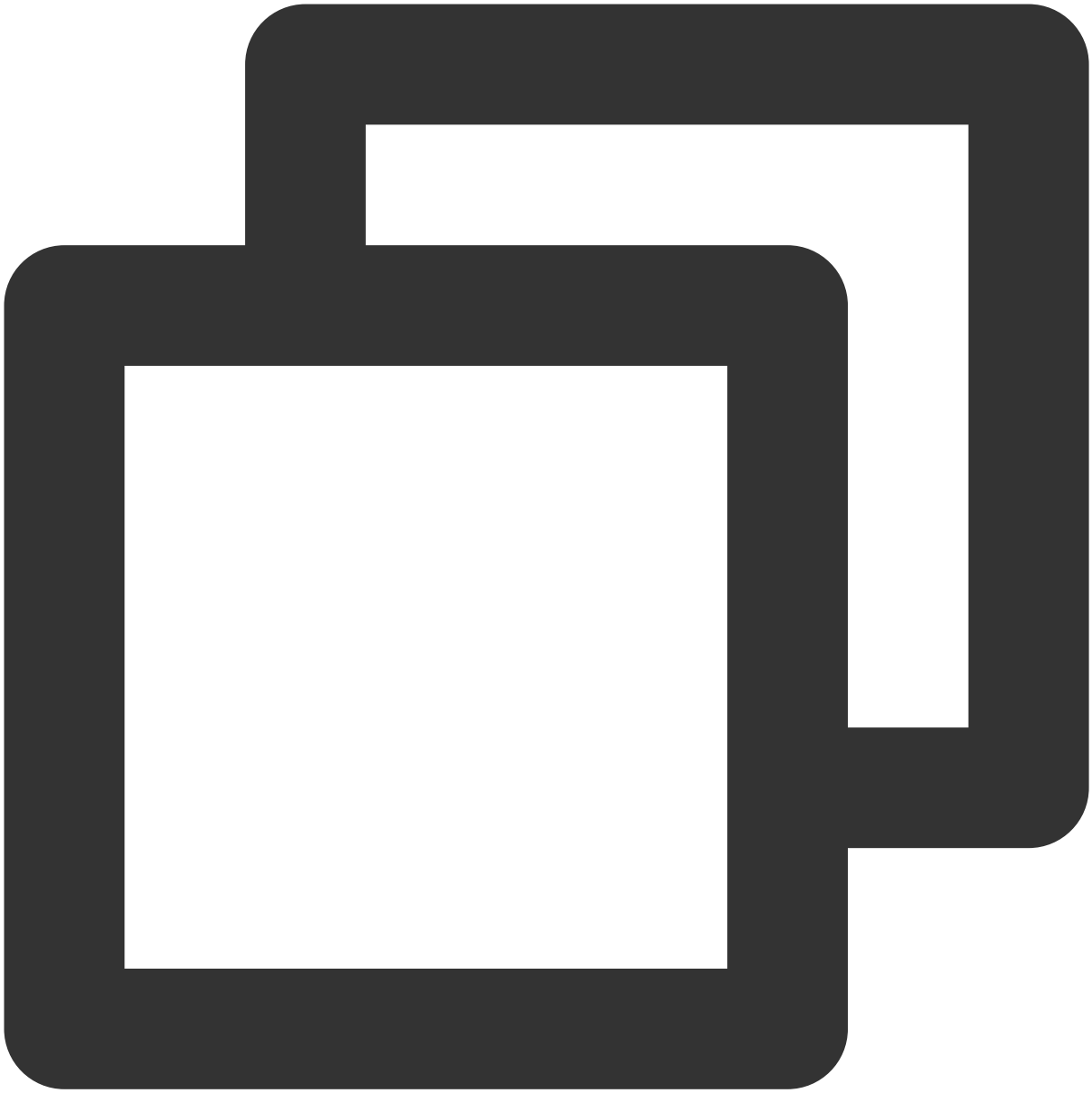
设置输入区域 InputView

输入区域 InputView，包含语音输入、文字输入、表情输入以及更多的“+”输入。



隐藏不要的功能

您可以自定义隐藏或展示更多“+”面板的图片、拍照、摄像以及发送文件的功能。

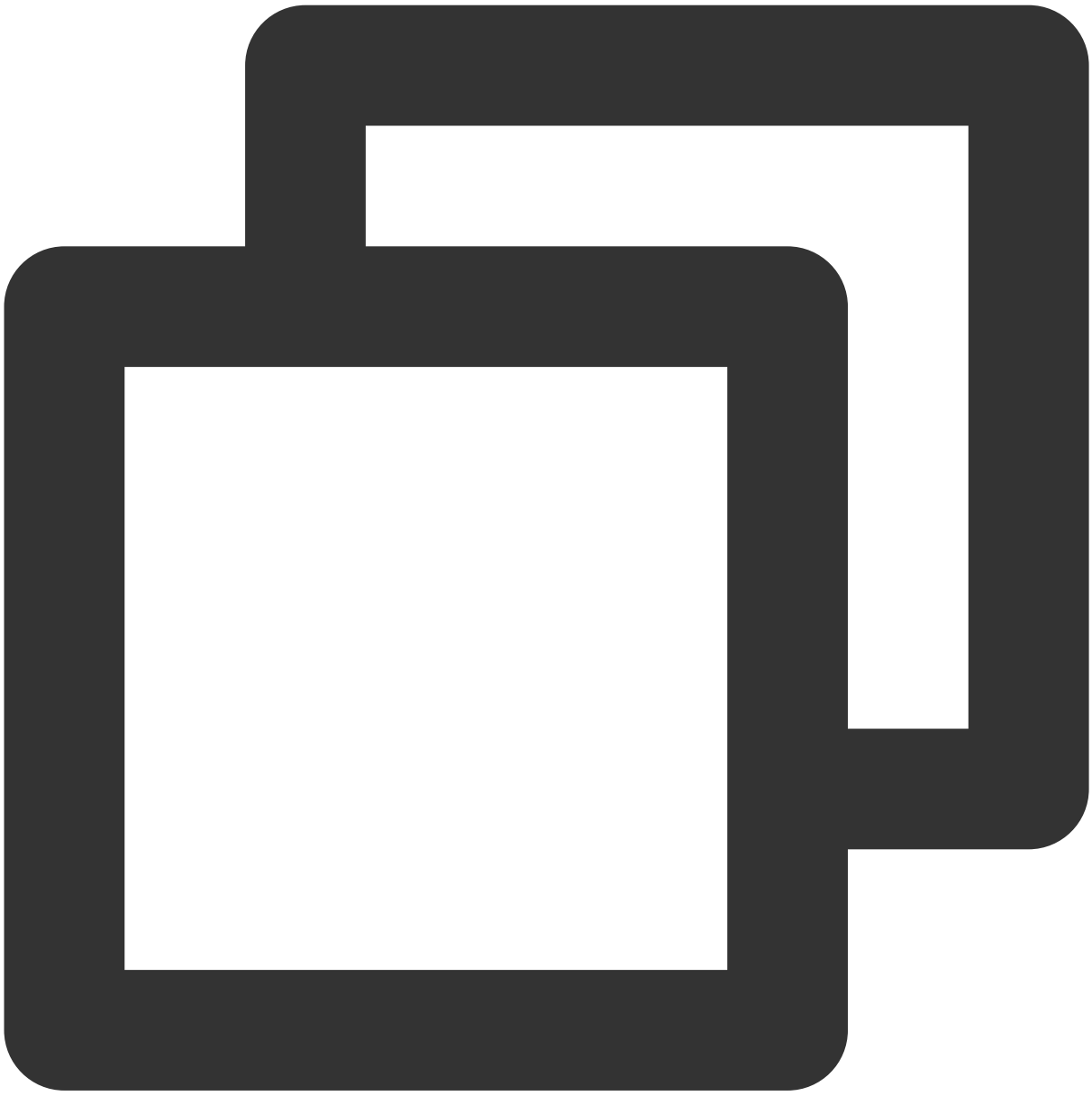


```
// 从 ChatView 获取 InputLayout
InputView inputView = layout.getInputLayout();
// 隐藏拍照并发送
inputView.disableCaptureAction(true);
// 隐藏发送文件
inputView.disableSendFileAction(true);
// 隐藏发送图片
inputView.disableSendPhotoAction(true);
// 隐藏摄像并发送
inputView.disableVideoRecordAction(true);
```

增加自定义的功能

您可以自定义新增更多“+”面板的动作单元实现相应的功能。

本文以隐藏发送文件，增加一个动作单元且该动作单元会发送一条消息为例，示例代码如下：

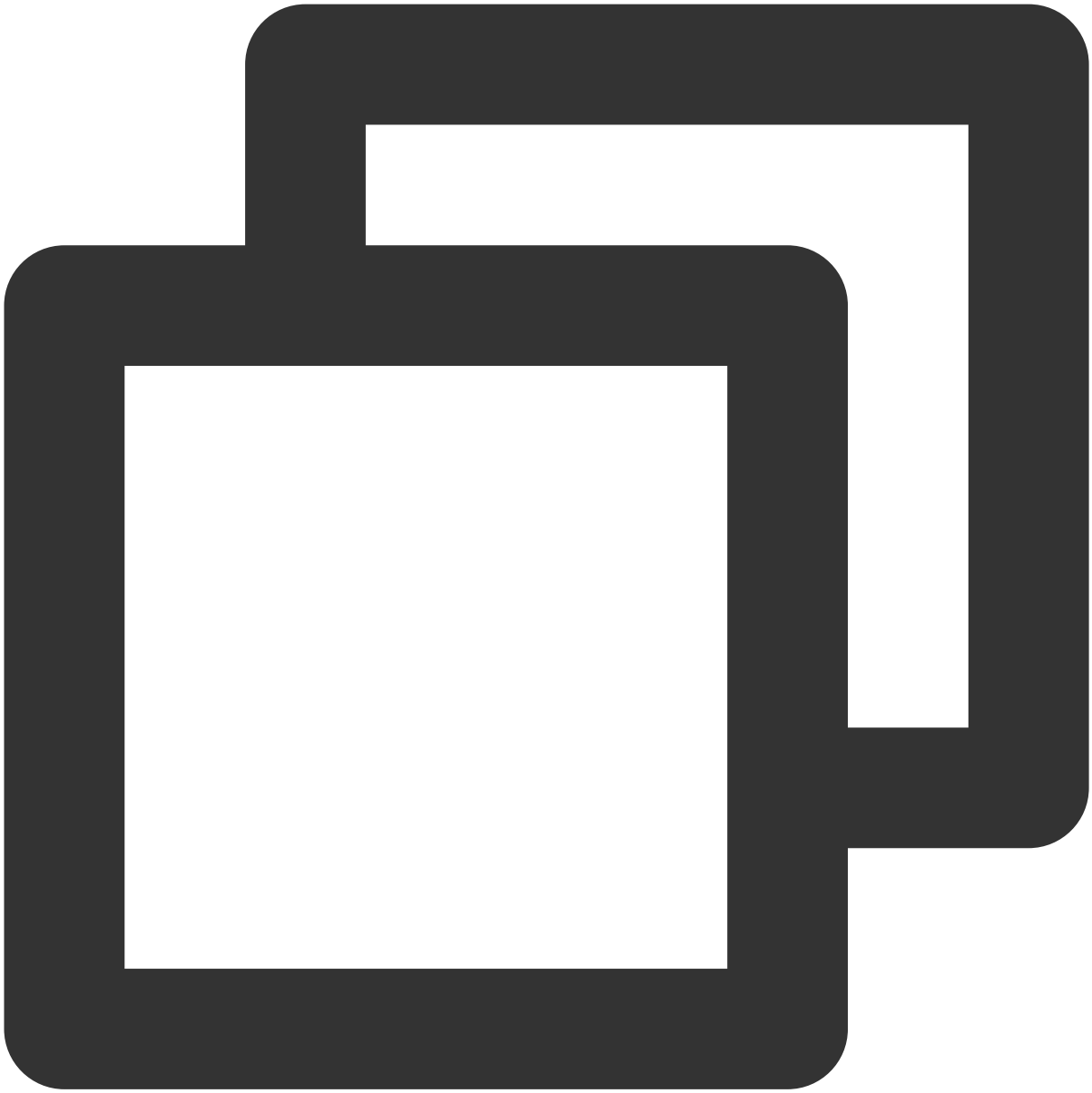


```
// 从 ChatView 获取 InputView
InputView inputView = layout.getInputLayout();
// 隐藏发送文件
inputView.disableSendFileAction(true);
// 定义一个动作单元
InputMoreActionUnit unit = new InputMoreActionUnit();
unit.setIconResId(R.drawable.default_user_icon); // 设置单元的图标
```

```
unit.setTitleId(R.string.profile); // 设置单元的文字标题
unit.setOnClickListener(unit.new OnActionClickListener() { // 定义点击事件
    @Override
    public void onClick() {
        ToastUtil.toastShortMessage("自定义的更多功能");
        MessageInfo info = MessageInfoUtil.buildTextMessage("我是谁");
        layout.sendMessage(info, false);
    }
});
// 把定义好的单元增加到更多面板
inputView.addAction(unit);
```

替换点击“+”的事件

您可以自定义替换更多“+”面板的各个动作单元的功能。

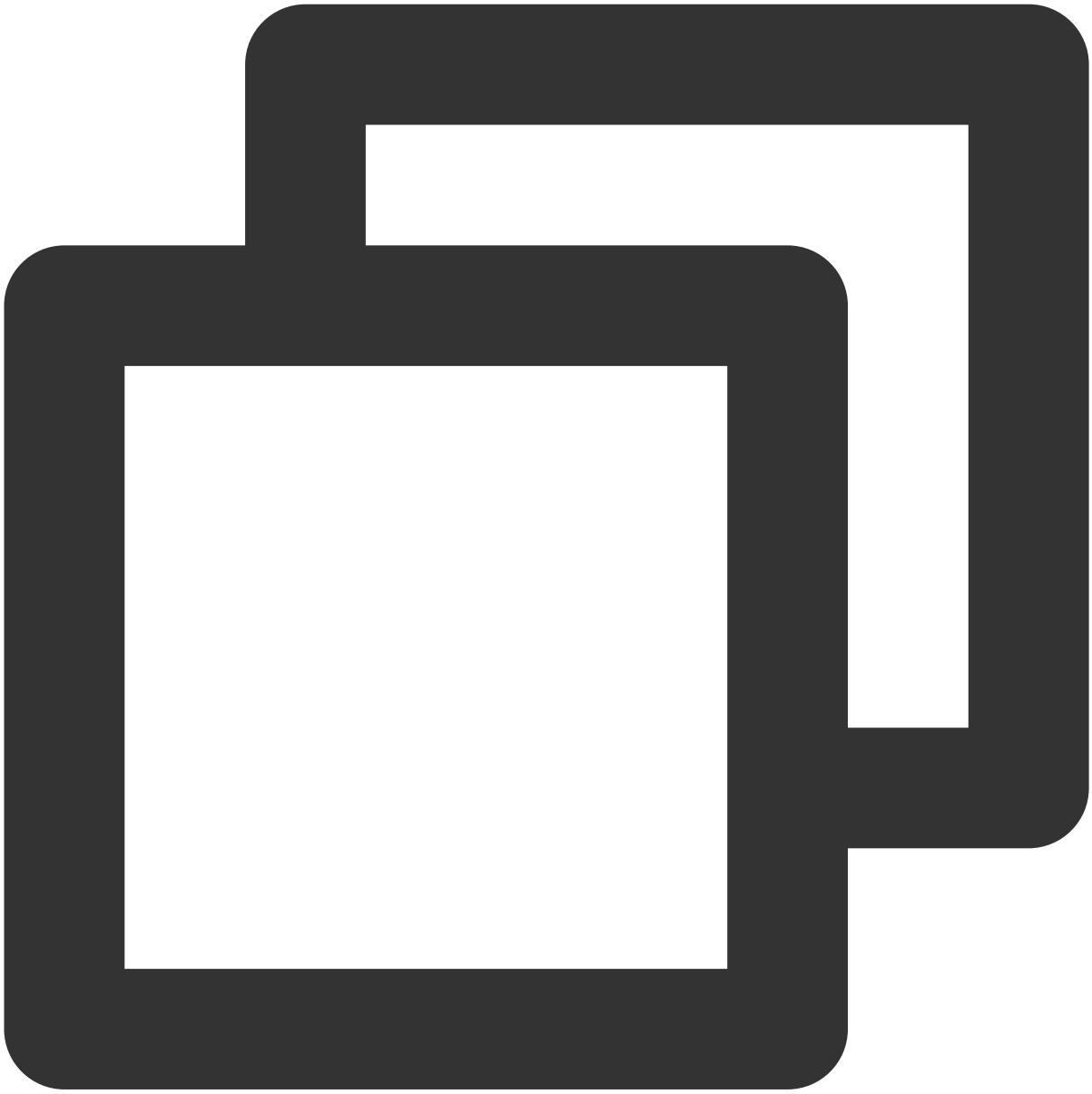


```
// 从 ChatView 获取 InputView
InputView inputView = layout.getInputLayout();
// 可以用自定义的事件来替换更多功能的入口
inputView.replaceMoreInput(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ToastUtil.toastShortMessage("自定义的更多功能按钮事件");
        MessageInfo info = MessageInfoUtil.buildTextMessage("自定义的消息");
        layout.sendMessage(info, false);
    }
});
```

版权所有：腾讯云计算（北京）有限责任公司

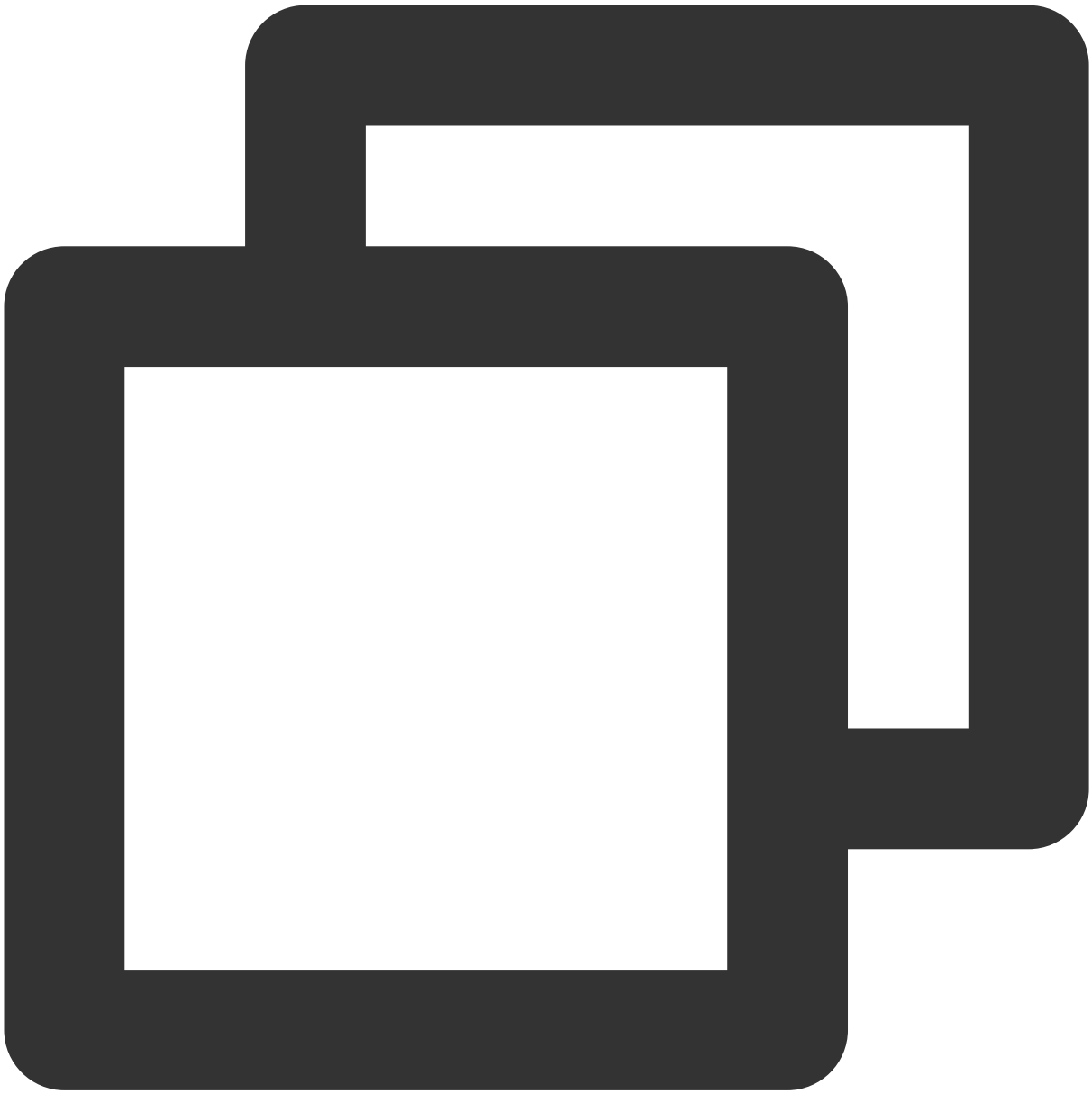
替换点击“+”弹出的面板

您可以自定义更多“+”面板的样式、各个动作单元以及其对应的功能。



```
// 从 ChatView 获取 InputView
InputView inputView = layout.getInputLayout();
// 可以用自定义的 fragment 来替换更多功能
inputView.replaceMoreInput(new CustomInputFragment());
```

新面板 CustomInputFragment 的实现和普通的 Fragment 没有区别，在 onCreateView 时 inflate 自己的 View，设置事件即可。本文以添加两个按钮，点击时弹出 toast 为例，示例代码如下：



```
public static class CustomInputFragment extends BaseInputFragment {
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container
        View baseView = inflater.inflate(R.layout.test_chat_input_custom_fragment,
        Button btn1 = baseView.findViewById(R.id.test_send_message_btn1);
        btn1.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                ToastUtil.toastShortMessage("发送一条超链接消息");
            }
        });
        Button btn2 = baseView.findViewById(R.id.test_send_message_btn2);
        btn2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ToastUtil.toastShortMessage("发送一条视频文字混合消息");
            }
        });
        return baseView;
    }
}
```

iOS

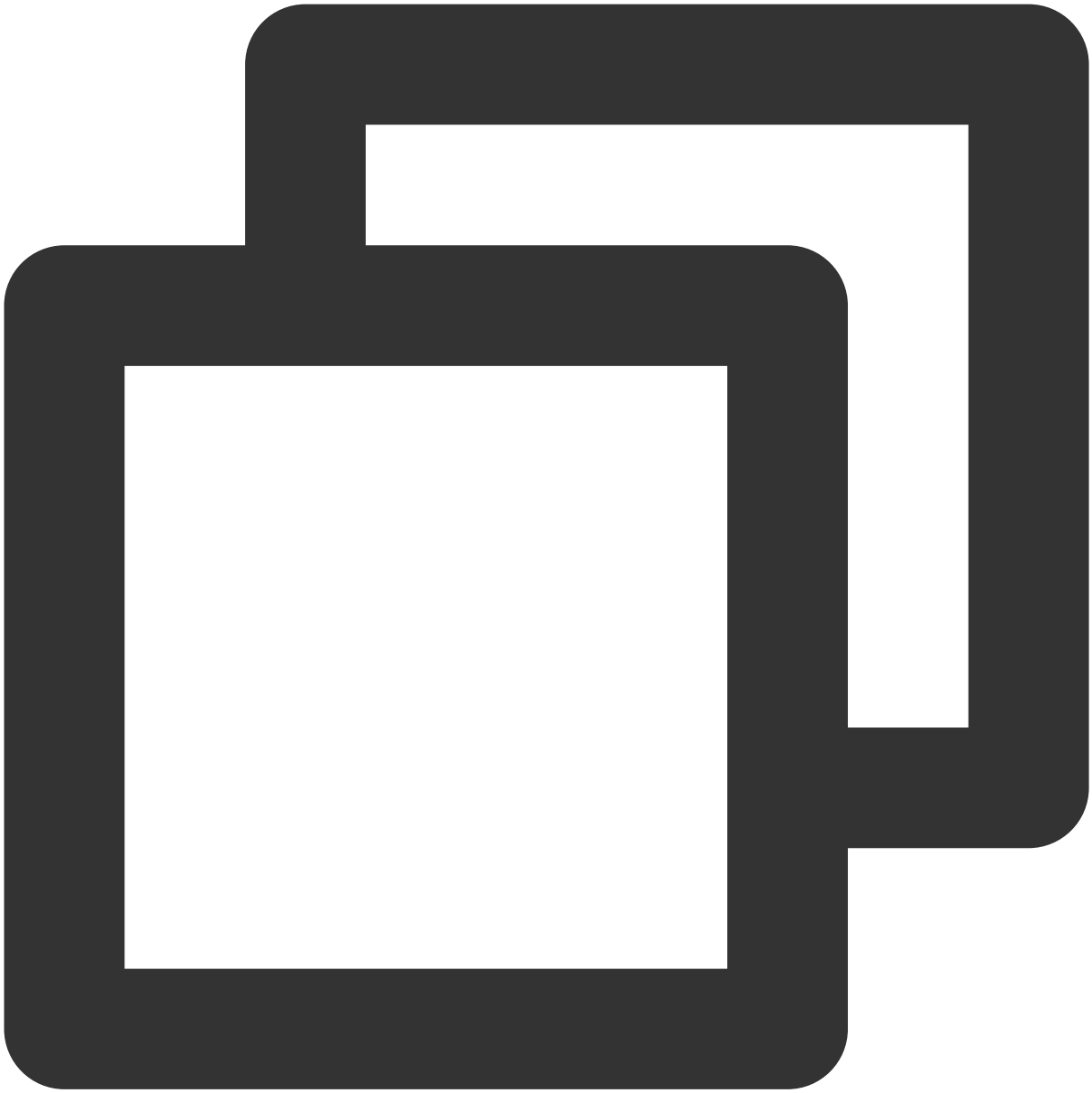
最近更新时间：2024-01-31 12:27:26

本文介绍如何设置 iOS 界面风格。

设置头像的样式

设置缺省的头像图片

TUIKit 的界面在显示用户时，会从用户资料中读取头像地址并显示。如果用户没有设置头像，则显示默认头像。您可以在 TUIKit 初始化前，自定义默认头像的图片，示例代码如下：



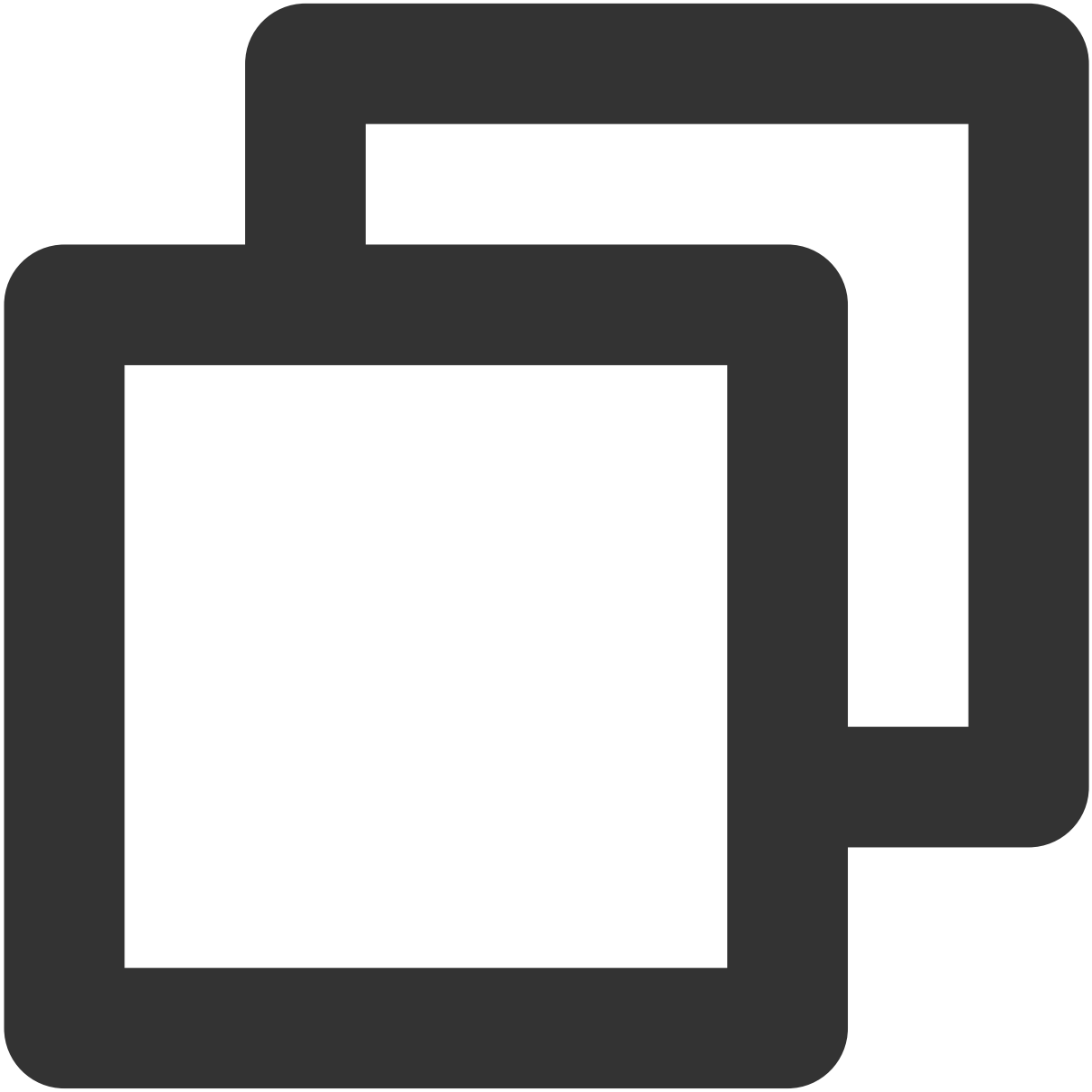
```
TUIConfig *config = [TUIConfig defaultConfig];
// 修改默认头像
config.defaultAvatarImage = [UIImage imageNamed:@"your image"];
// 修改默认群组头像
config.defaultGroupAvatarImage = [UIImage imageNamed:@"your group image"];
// 禁用群组九宫格头像
config.enableGroupGridAvatar = NO;
```

说明：

群组头像默认展示九宫格头像。九宫格头像是 TUIKit 使用群成员的头像合成的，如果九宫格合成失败或群组里只有一个成员，TUIKit 会展示 `defaultGroupAvatarImage` 。
如果您不想使用九宫格头像，可以禁用九宫格头像。

设置头像的形状

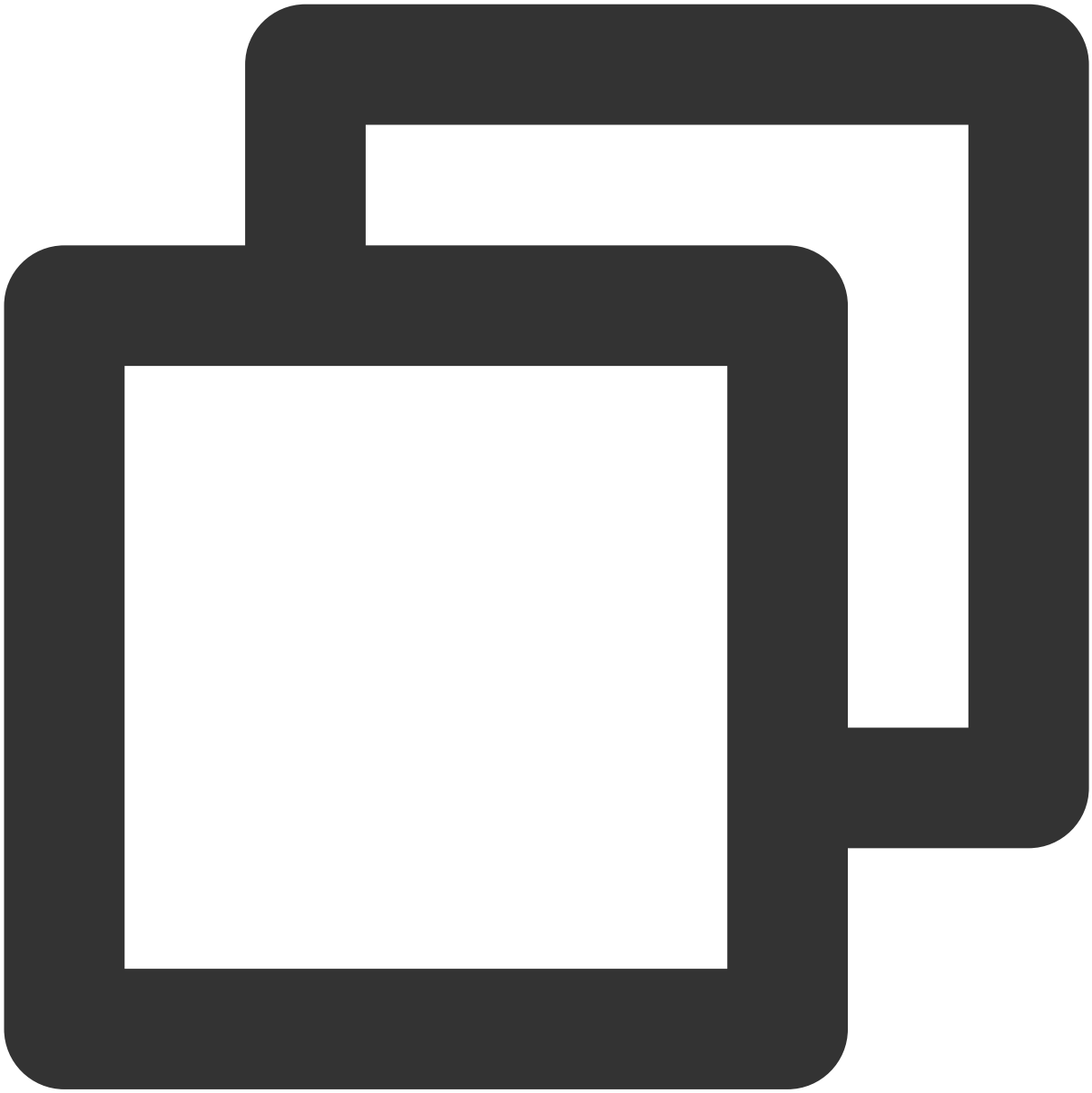
TUIKit 提供 3 种头像类型：矩形直角头像、圆形头像和矩形圆角头像。



```
typedef NS_ENUM(NSInteger, TUIKitAvatarType) {
    TAvatarTypeNone,                /*矩形直角头像*/
    TAvatarTypeRounded,             /*圆形头像*/
```

```
    TAvatarTypeRadiusCorner,      /*圆角头像*/
};
```

您可以在 TUIKit 初始化前，选择头像类型，示例代码如下：

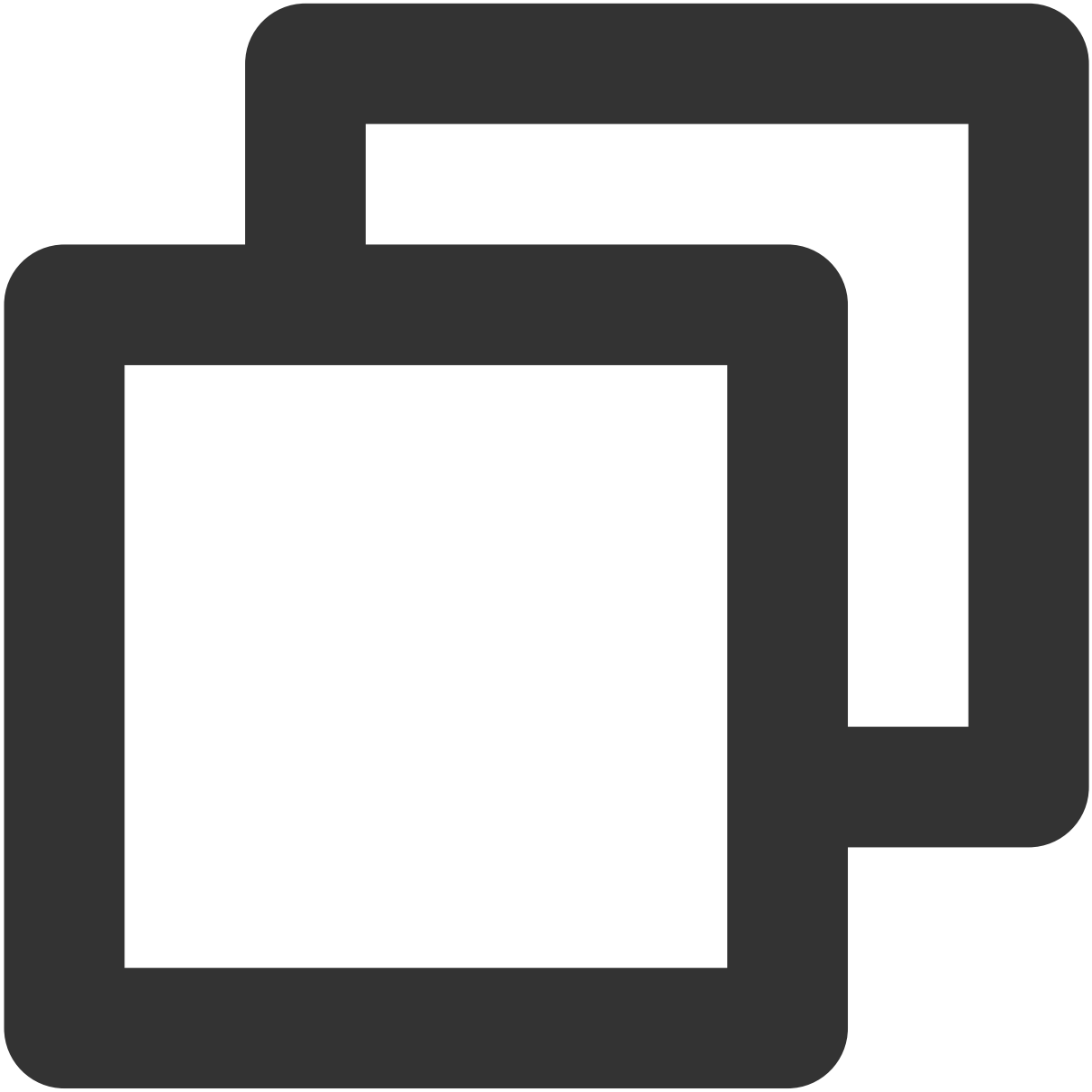


```
TUIConfig *config = [TUIConfig defaultConfig];
// 修改头像类型为圆角矩形，圆角大小为 5
config.avatarType = TAvatarTypeRadiusCorner;
config.avatarCornerRadius = 5.f;
```

设置聊天窗口的样式

设置聊天窗口的背景色

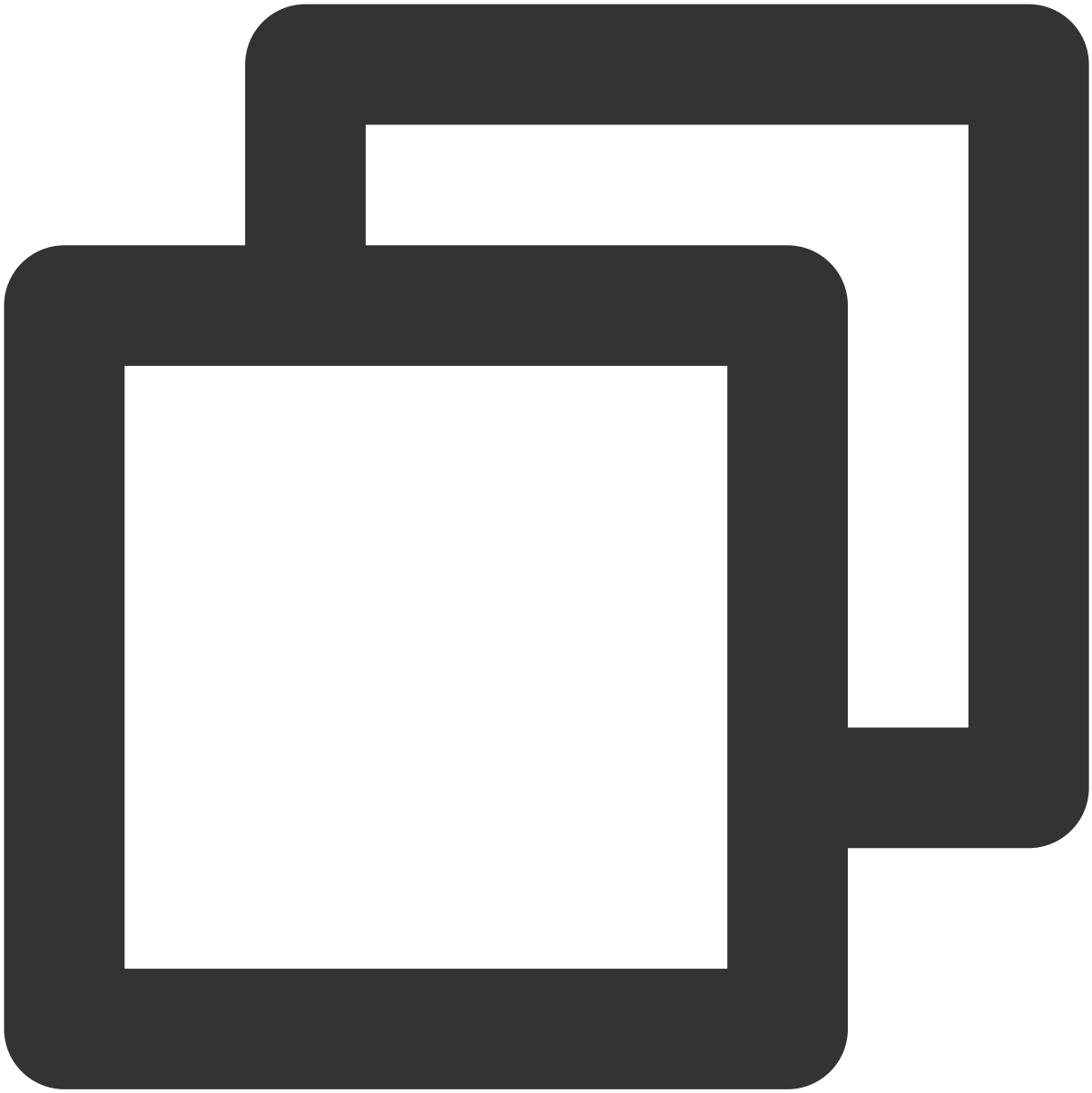
您可以在初始化聊天窗口前，自定义窗口背景色，示例代码如下：



```
[TUIChatConfig defaultConfig].backgroudColor = [UIColor greenColor];
```

设置聊天窗口的背景图片

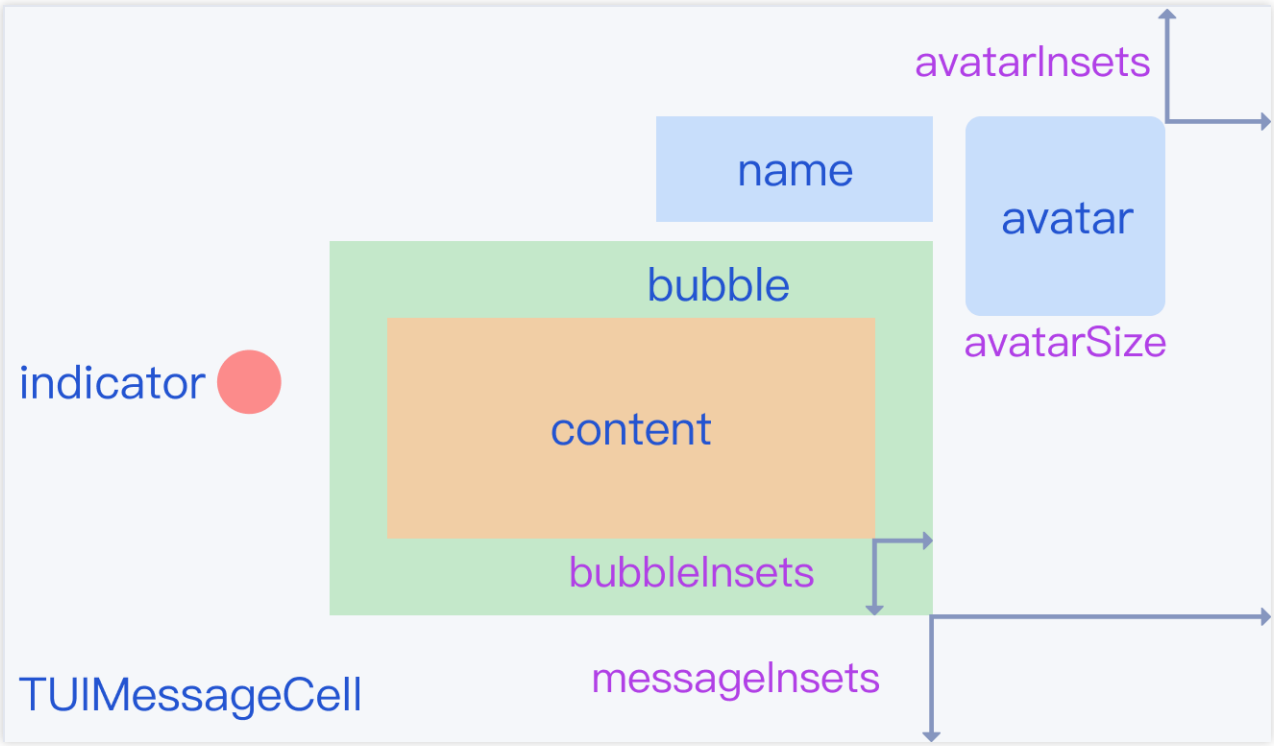
您可以在初始化聊天窗口前，自定义窗口背景图片，示例代码如下：



```
[TUIChatConfig defaultConfig].backgroudImage = [UIImage imageNamed:@"your chat back
```

设置消息气泡的样式

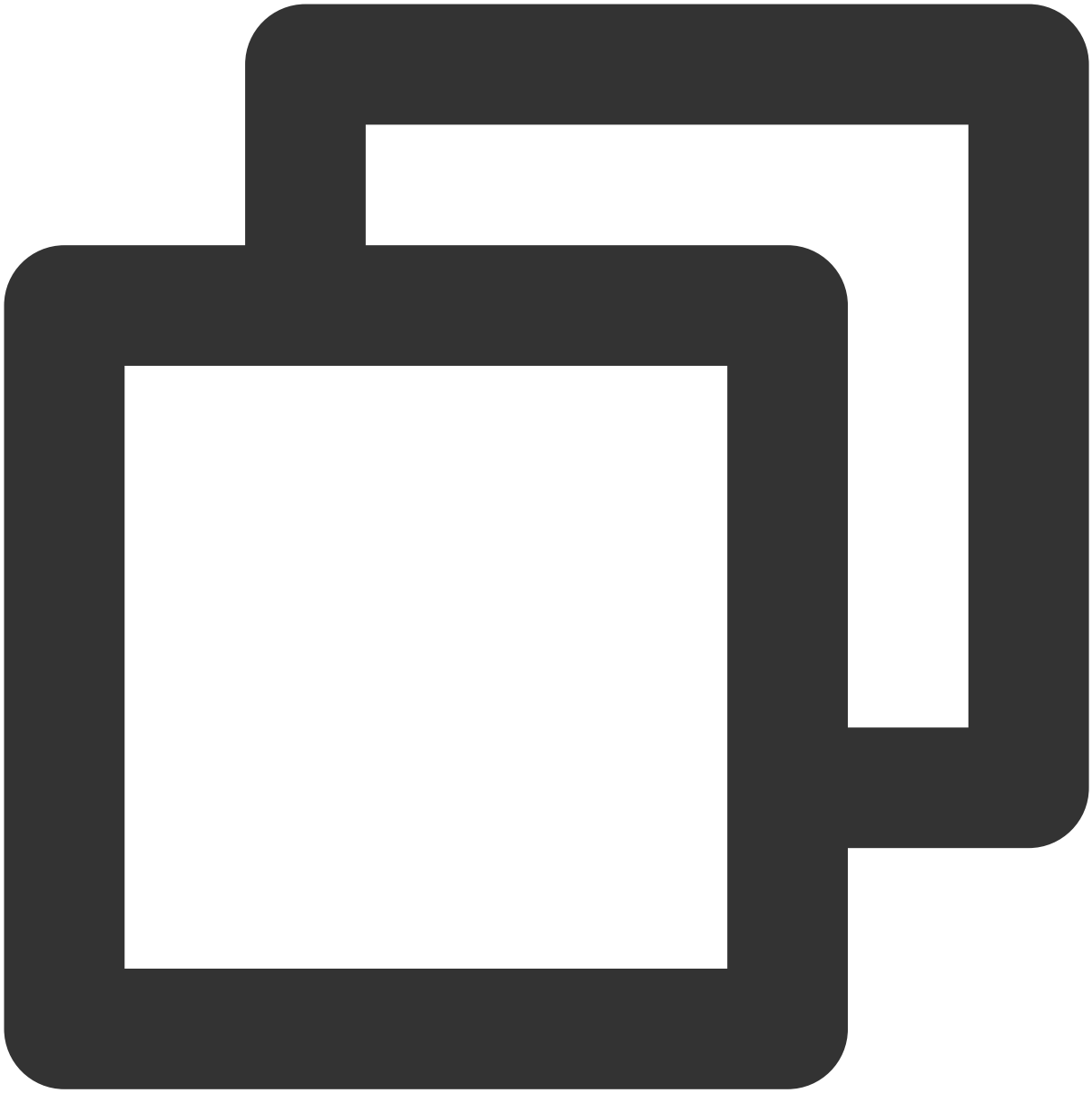
聊天界面消息 View 的组合方式如下图所示：



设置消息的字体和颜色

文字消息的数据来自于 TUITextMessageCellData 类，通过它的接口可以修改文字消息的字体和颜色。

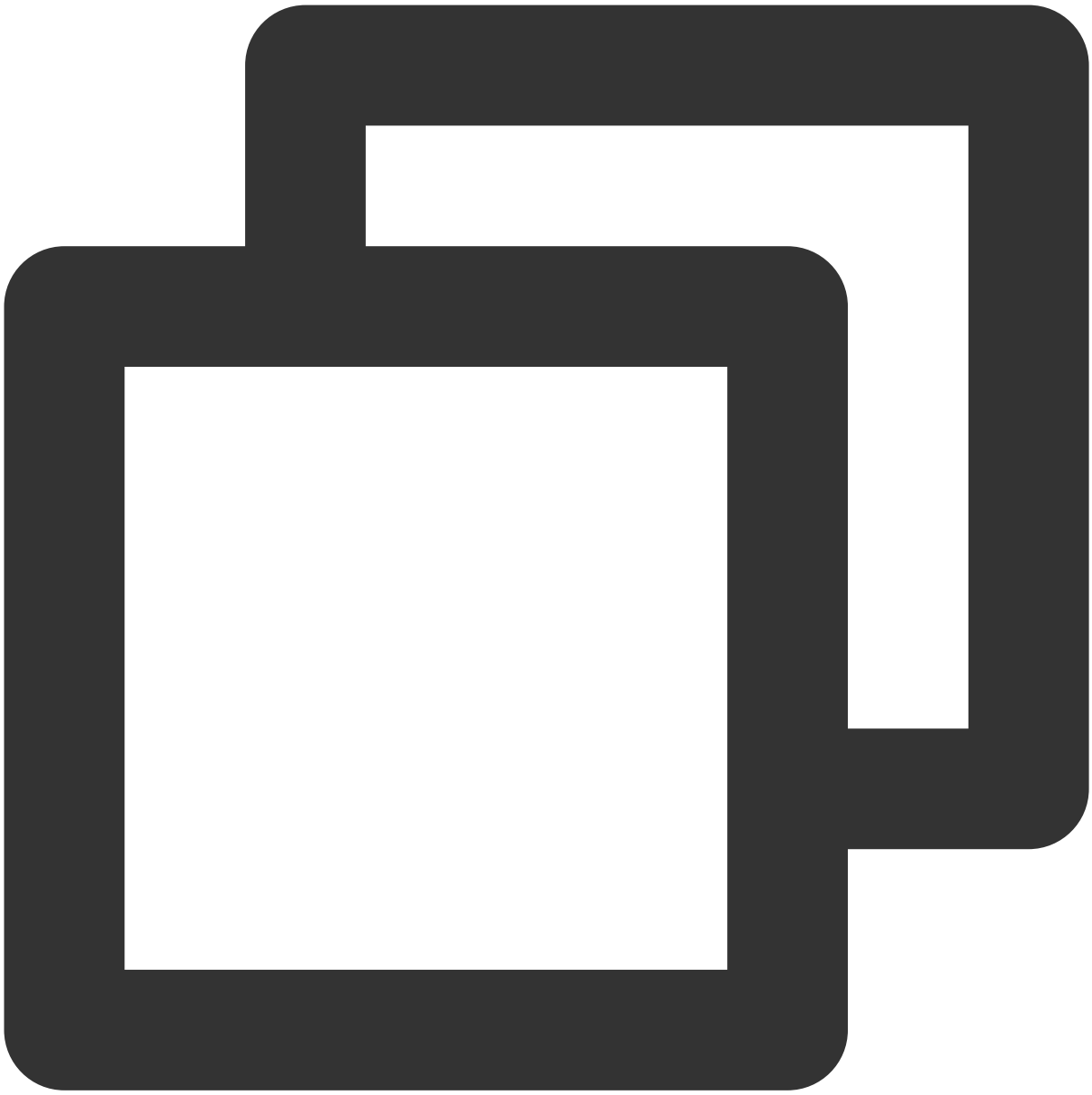
您可以在初始化聊天窗口前，自定义消息的字体和颜色，示例代码如下：



```
// 设置发送文字消息的字体和颜色
[TUITextMessageCellData setOutgoingTextFont:[UIFont systemFontOfSize:20]];
[TUITextMessageCellData setOutgoingTextColor:[UIColor blueColor]];
// 设置接收文字消息的字体和颜色
[TUITextMessageCellData setIncommingTextFont:[UIFont systemFontOfSize:20]];
[TUITextMessageCellData setIncommingTextColor:[UIColor purpleColor]];
```

设置气泡的背景图片

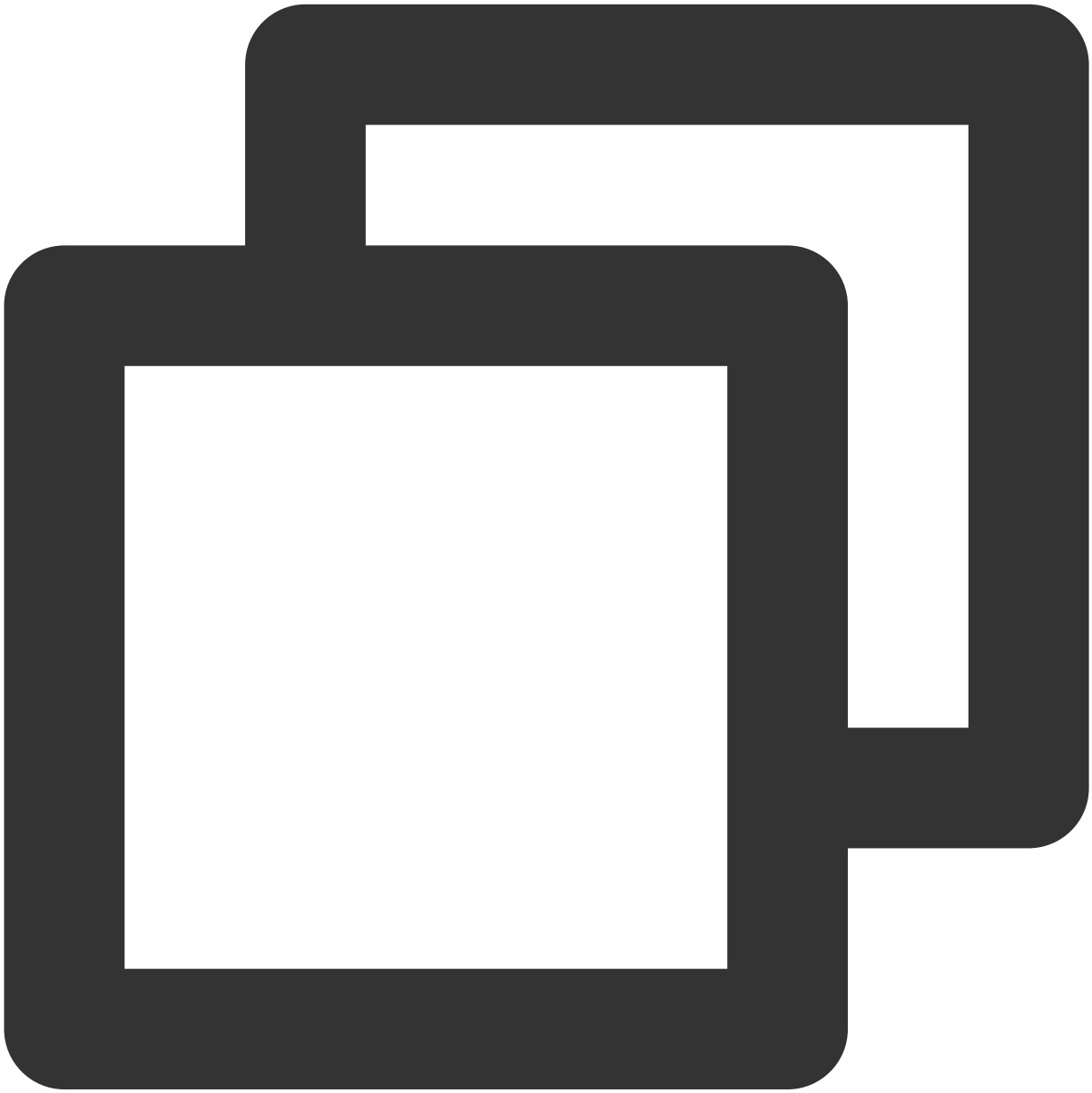
气泡 Cell 显示的图片从 TUIBubbleMessageCellData 获取，该对象提供了类方法可以设置图片。
您可以在初始化聊天窗口前，自定义气泡背景图片，示例代码如下：



```
// 设置发送气泡，包括普通状态和选中状态
[TUIBubbleMessageCellData setOutgoingBubble:[UIImage imageNamed:@"outgoing_bubble"]
[TUIBubbleMessageCellData setOutgoingHighlightedBubble:[UIImage imageNamed:@"outgoi
// 设置接收气泡，包括普通状态和选中状态
[TUIBubbleMessageCellData setIncommingBubble:[UIImage imageNamed:@"incoming_bubble"
[TUIBubbleMessageCellData setIncommingHighlightedBubble:[UIImage imageNamed:@"incom
```

设置气泡的边距

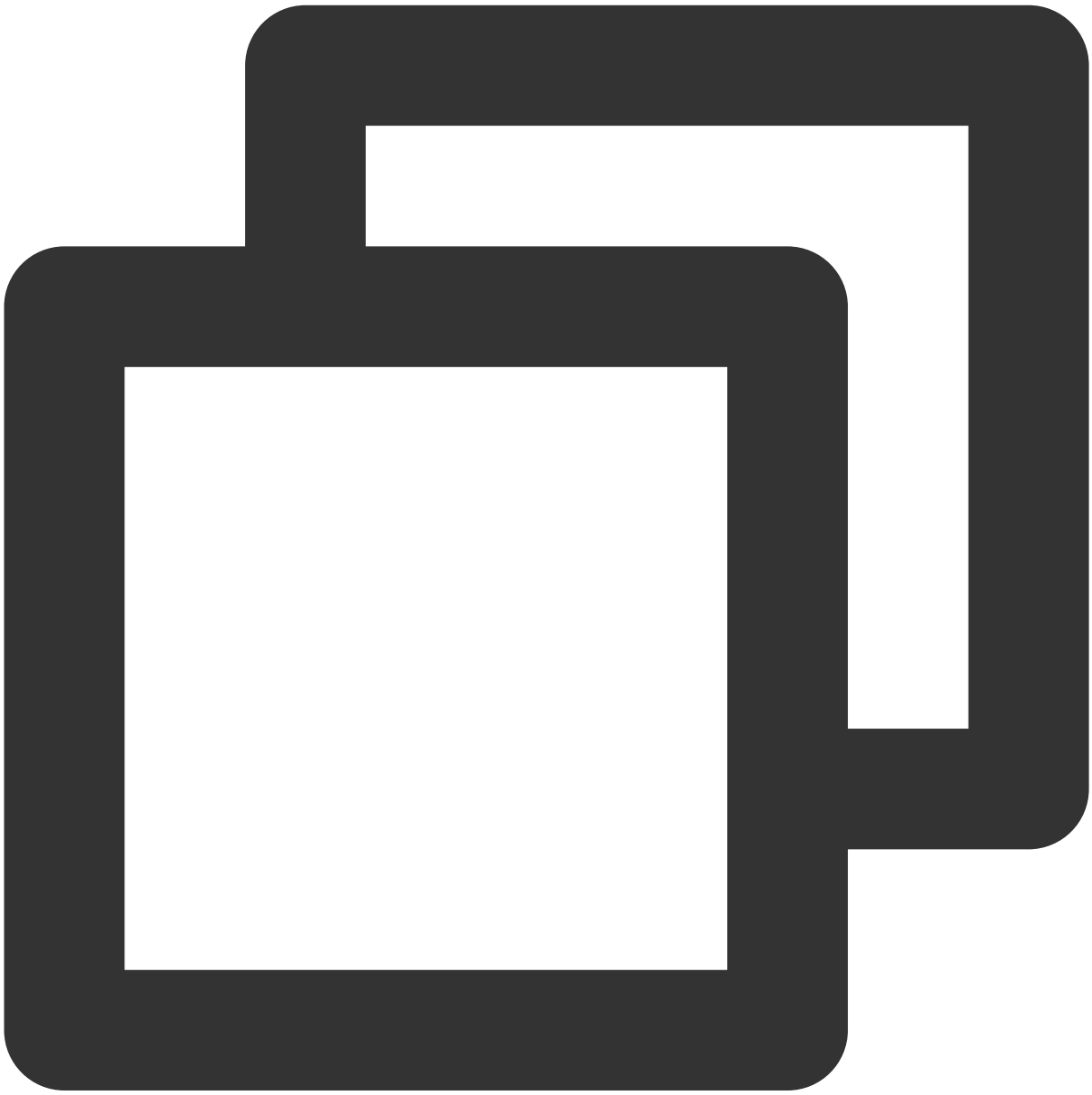
在 TUIKit 中，文字和声音都会用气泡显示，TUIMessageCellLayout 提供了类方法设置 bubbleInsets。
您可以在初始化聊天窗口前，自定义气泡边距，示例代码如下：



```
// 设置发送消息的气泡边距
[TUIMessageCellLayout outgoingTextMessageLayout].bubbleInsets = UIEdgeInsetsMake(20
// 设置接收消息的气泡边距
[TUIMessageCellLayout incommingTextMessageLayout].bubbleInsets = UIEdgeInsetsMake(2
```

设置发送者的头像样式

修改 TUIMessageCellLayout 的相关属性，可设置发送者的头像样式。
您可以在初始化聊天窗口前，自定义发送者的头像样式，示例代码如下：



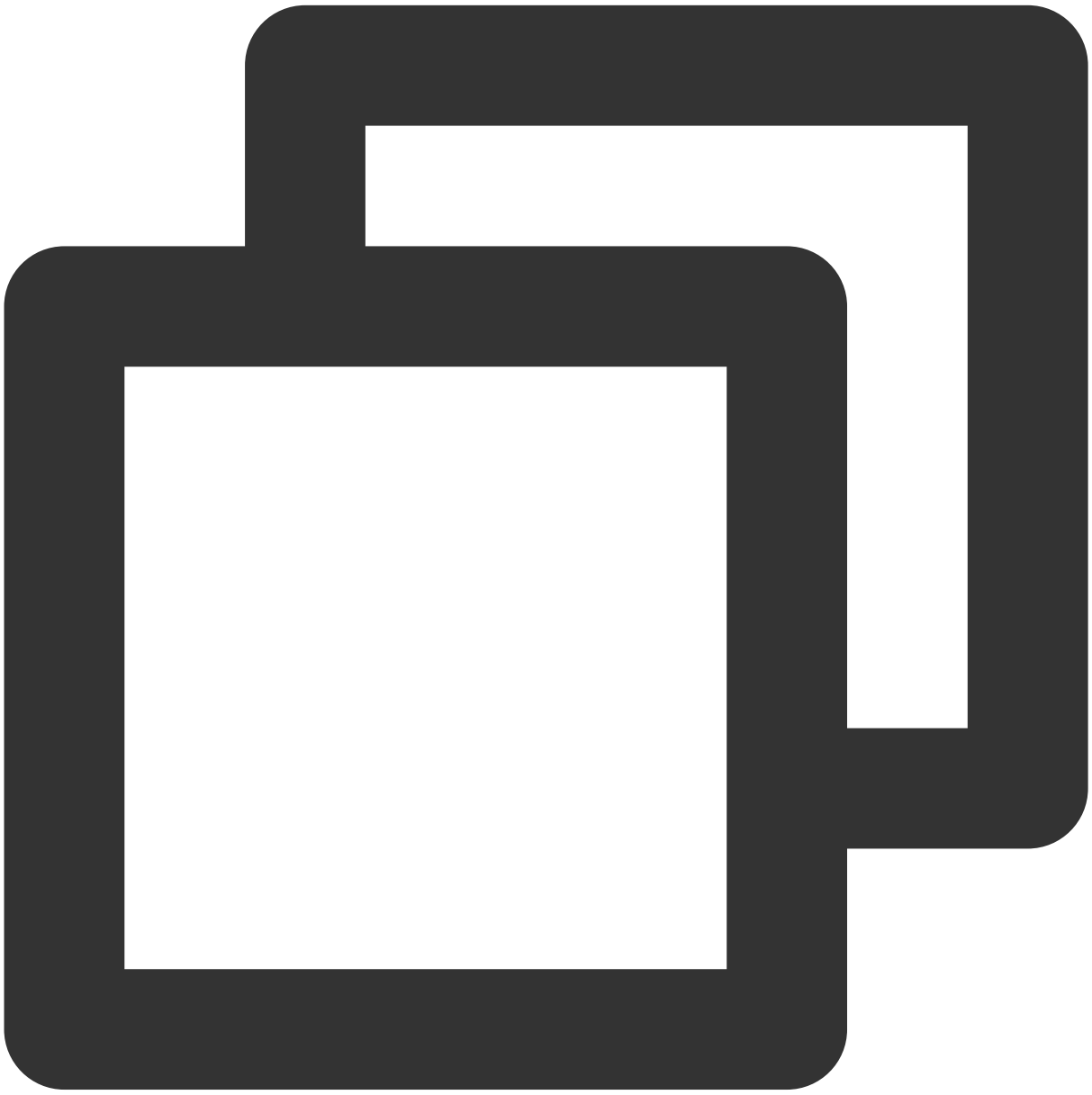
```
// 设置发送者头像大小和位置
[TUIMessageCellLayout outgoingTextMessageLayout].avatarSize = CGSizeMake(80, 80);
[TUIMessageCellLayout outgoingTextMessageLayout].avatarInsets = UIEdgeInsetsMake(10
// 设置接收者头像大小和位置
[TUIMessageCellLayout incommingTextMessageLayout].avatarSize = CGSizeMake(80, 80);
[TUIMessageCellLayout incommingTextMessageLayout].avatarInsets = UIEdgeInsetsMake(1
```

注意：

其他消息类型请获取对应的 layout 实例设置头像的大小和位置。

设置消息的昵称样式

修改 TUIMessageCellLayout 的相关属性，可设置消息的发送方昵称字体和颜色。
您可以在初始化聊天窗口前，自定义消息的昵称样式，示例代码如下：



```
// 设置收到的消息的昵称字体及颜色
[TUIMessageCellData setIncommingNameFont:[UIFont systemFontOfSize:20]];
[TUIMessageCellData setIncommingNameColor:[UIColor blueColor]];
// 设置自己的昵称字体及颜色，默认情况下不显示自己的昵称
```

```
[TUIMessageCellData setOutgoingNameFont:[UIFont systemFontOfSize:20]];
[TUIMessageCellData setOutgoingNameColor:[UIColor purpleColor]];
```

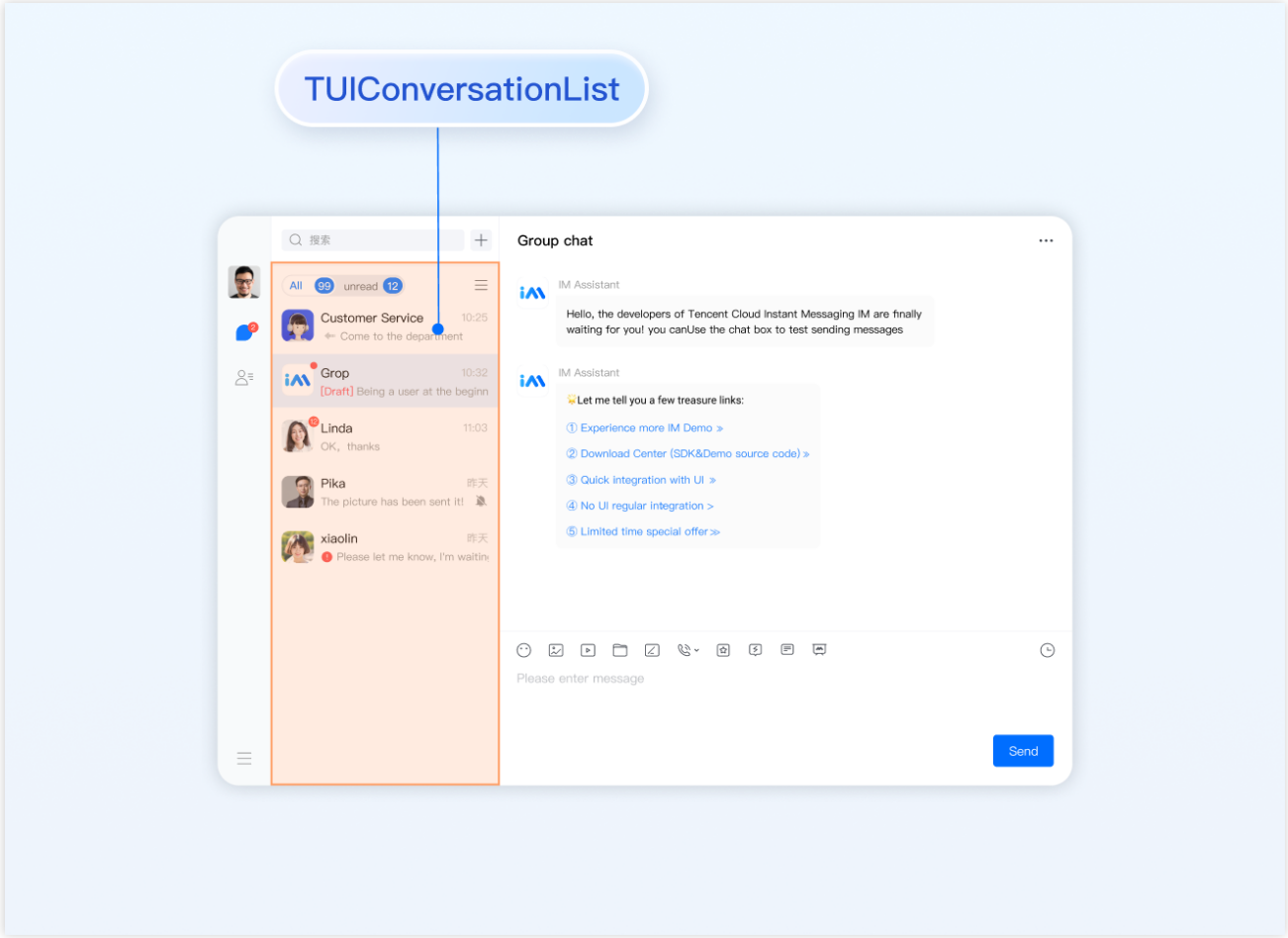
Web(Vue)

最近更新时间：2024-01-31 12:32:35

本文介绍如何设置 Web 界面风格。

设置会话列表

TUIConversation 提供会话列表功能。会话列表主要由会话列表区组成，会话列表区提供了 UI 样式可供修改。



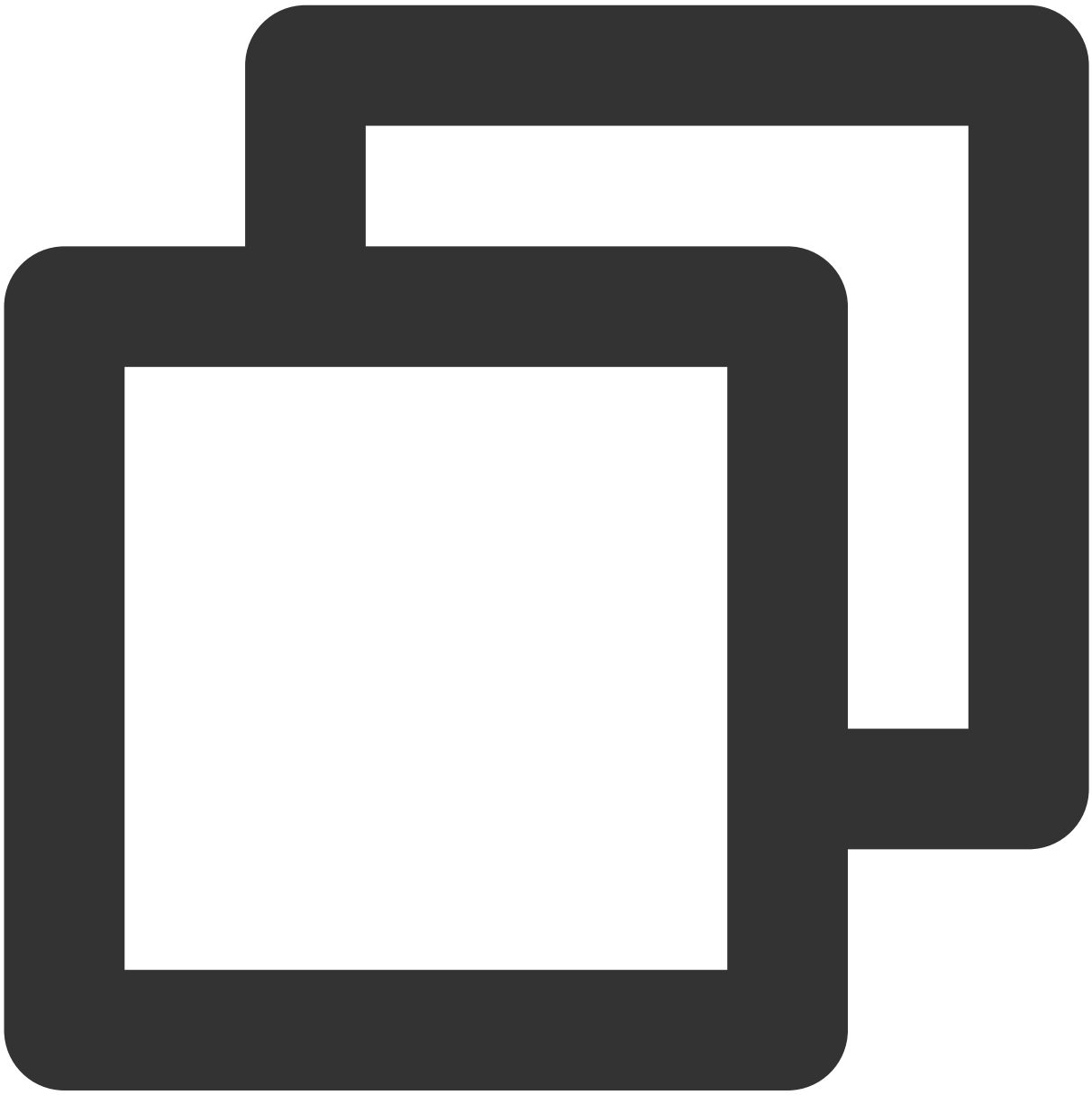
设置会话列表样式

登录后 TUIKit 会根据用户名从 SDK 读取该用户的会话列表。会话列表提供一些常用功能定制，例如头像样式、背景、字体大小、点击与长按事件等。

会话列表中单个列表项展示主要在路径 `TUIKit/components/TUIConversation/conversation-`

`list/index.vue` 文件中。

示例代码如下：

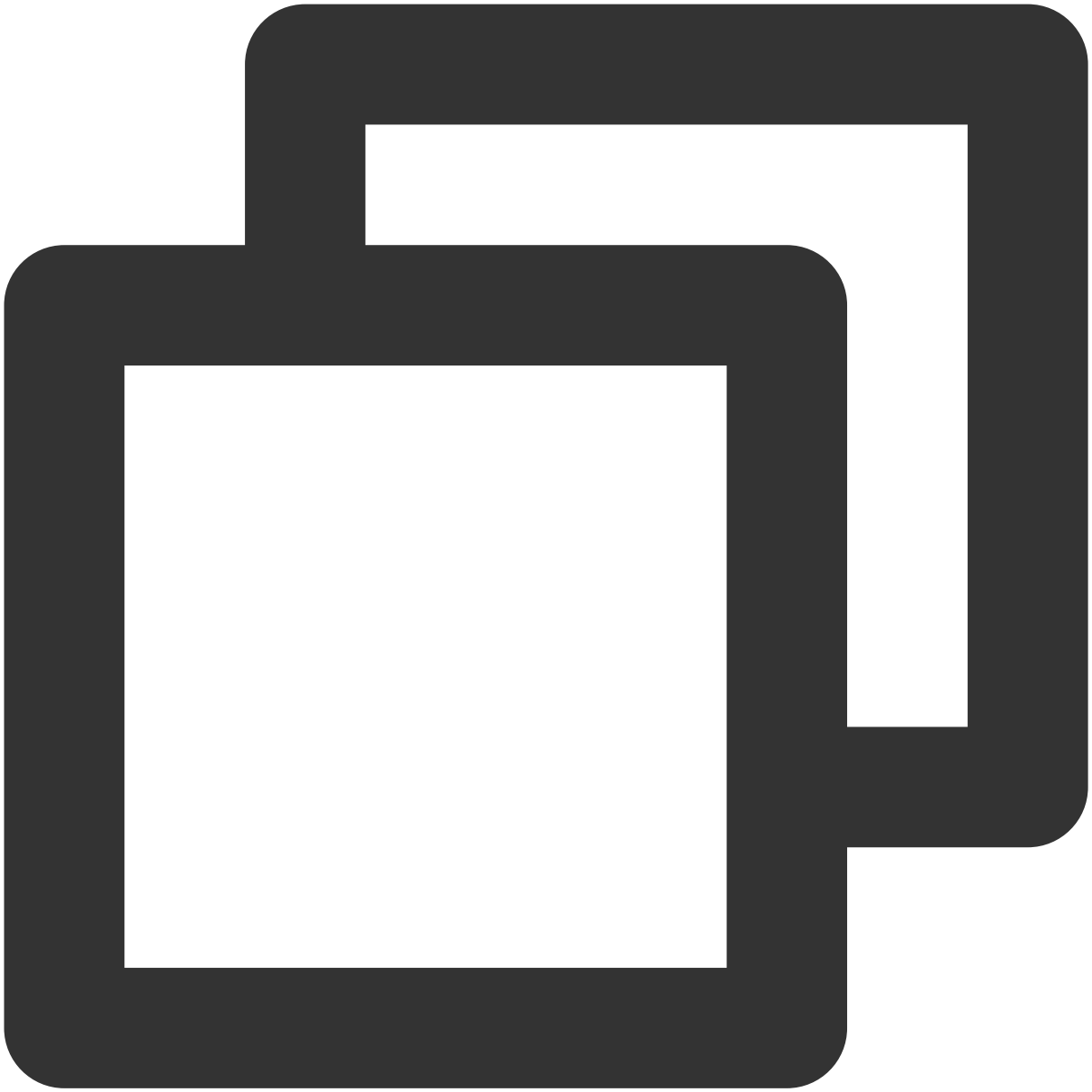


```
<template>
  <div class="tui-conversation-list">
    <!-- Conversation List operation panel -->
    <ActionsMenu .../>
    <!-- Conversation List Main -->
    <div v-for="(conversation, index) in conversationList" ...>
      <!-- Conversation List Item -->
      <div :class="['TUI-conversation-item']">
```

```
        <aside class="left">
          <!-- Avatar -->
          <img class="avatar" :src="conversation.getAvatar()" />
          <!-- User Online Status -->
          <div ... :class="['online-status']"></div>
          <!-- Conversation Unread Count -->
          <span class="num" ...>...</span>
          <!-- Conversation Unread Red Dot (displayed in Do Not Disturb mode) -->
          <span class="num-notify" ...>...</span>
        </aside>
        <div class="content">
          <div class="content-header">
            <!-- Conversation Name -->
            <label class="content-header-label">
              <p class="name">{{ conversation.getShowName() }}</p>
            </label>
            <!-- Conversation Last Message -->
            <div class="middle-box">
              <!-- Conversation Last Message When Mentiond -->
              <span class="middle-box-at" ...>{{ conversation.getGroupAtInfo() }}</
              <!-- Conversation Last Message Content -->
              <p class="middle-box-content">{{ conversation.getLastMessage("text")
            </div>
          </div>
          <div class="content-footer">
            <!-- Conversation Lastest Message Time -->
            <span class="time">{{ conversation.getLastMessage("time") }}</span>
            <!-- Conversation Muted Flag -->
            <Icon v-if="conversation.isMuted" :file="muteIcon"></Icon>
          </div>
      ...
</template>
```

您可以在路径 `TUIKit/components/TUIConversation/conversation-list/style/web.scss` 下设置会话列表中列表项的样式。

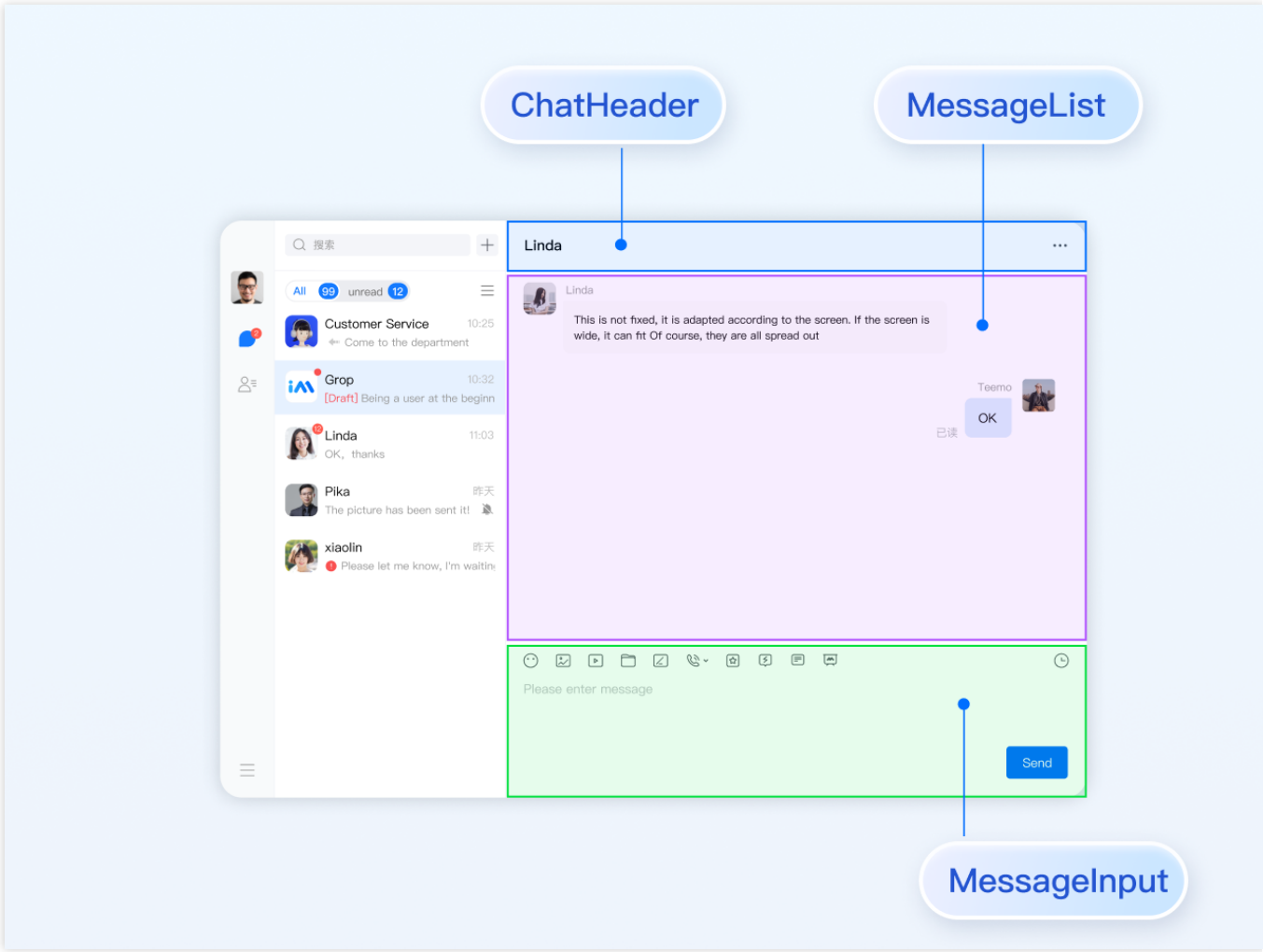
设置会话列表中头像样式示例代码如下：



```
.TUI-conversation {
    &-item {
        .left {
            .avatar {
                width: 30px; // avatar width
                height: 30px; // avatar height
                border-radius: 5px; // avatar border radius
            }
        }
    }
}
```

设置聊天窗口的样式

TUIChat 提供聊天窗口。聊天窗口包含三个区域，从上到下为标题栏区、消息区和输入区，如下图所示：



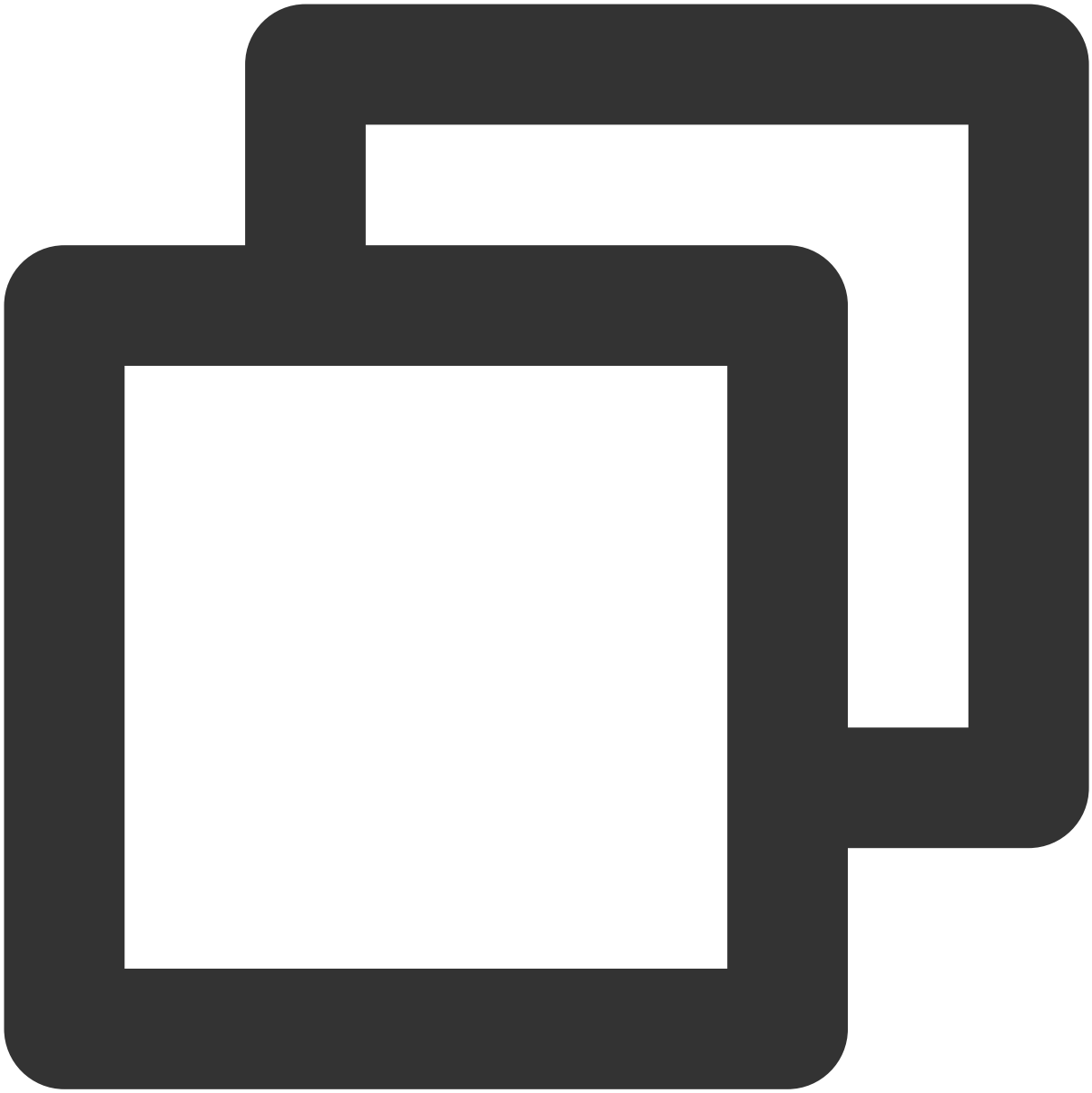
聊天窗口相关配置主要在路径 `src/TUIKit/components/TUIChat` 文件目录中。

设置标题栏区样式

标题栏由左右三个区域组成，如下图所示：

聊天窗口标题栏相关代码主要在路径 `src/TUIKit/components/TUIChat/chat-header/index.vue` 文件中。聊天窗口标题栏区提供一些常用功能定制，例如背景、字体大小、按钮图标、点击事件、功能开关等。

示例代码如下：



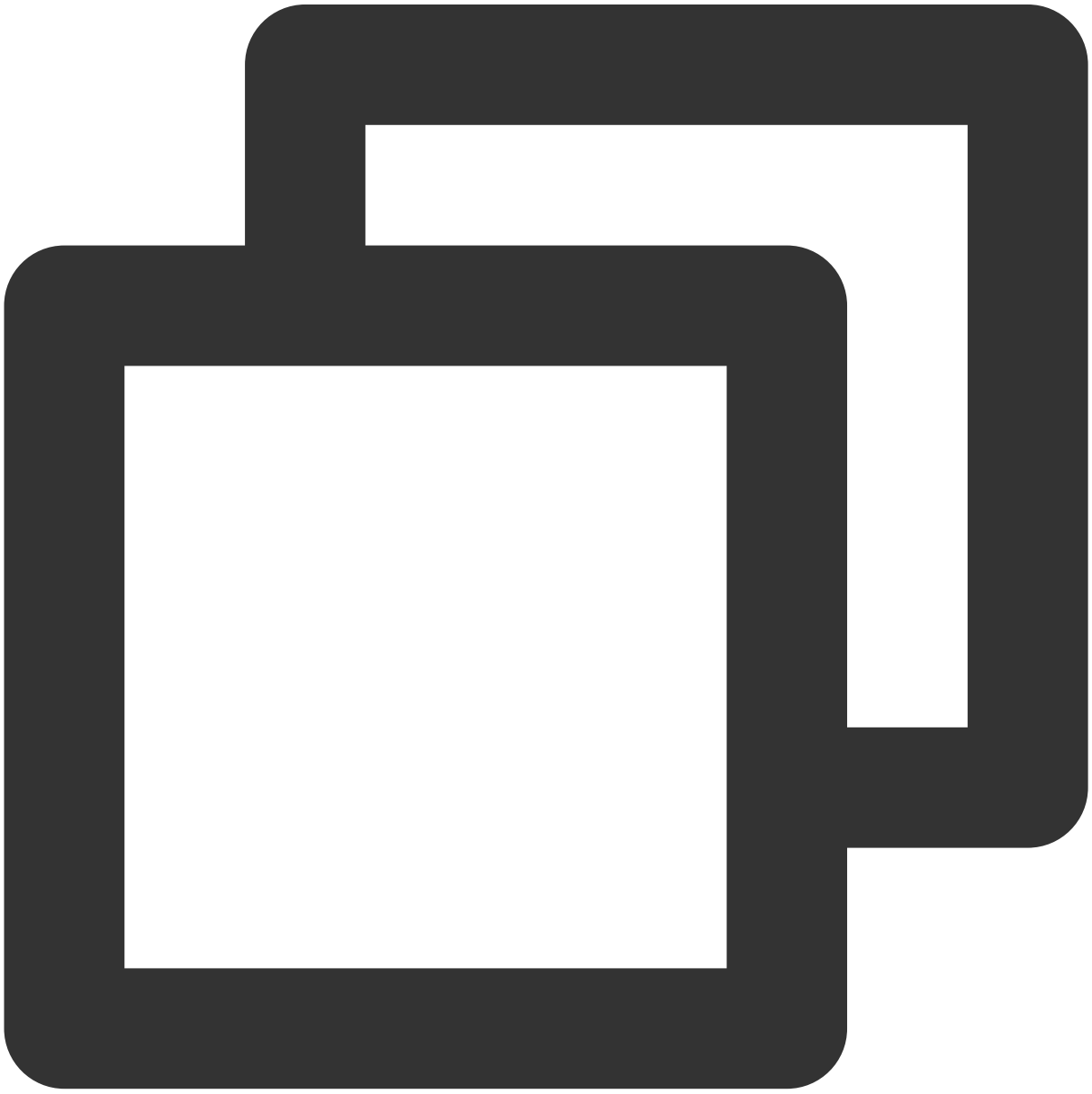
```
<template>
  <div :class="['chat-header', !isPC && 'chat-header-h5']">
    ...
    <!-- Chat name / [Typing...] status prompt-->
    <div :class="['chat-header-content', ...]">
      {{ currentConversationName }}
    </div>
    <!-- Group chat settings extension -->
    <div :class="['chat-header-setting', ...]">
      <div v-for="(item, index) in extensions" :key="index" @click.stop="handleExte
        <Icon :file="item.icon"></Icon>
```

版权所有：腾讯云计算（北京）有限责任公司

```
        </div>
      </div>
    </div>
  </template>
```

您可以在 `src/TUIKit/components/TUIChat/chat-header/index.vue` 文件中设置聊天窗口标题栏区样式。

设置聊天窗口标题栏区的字体大小和背景色示例代码如下：



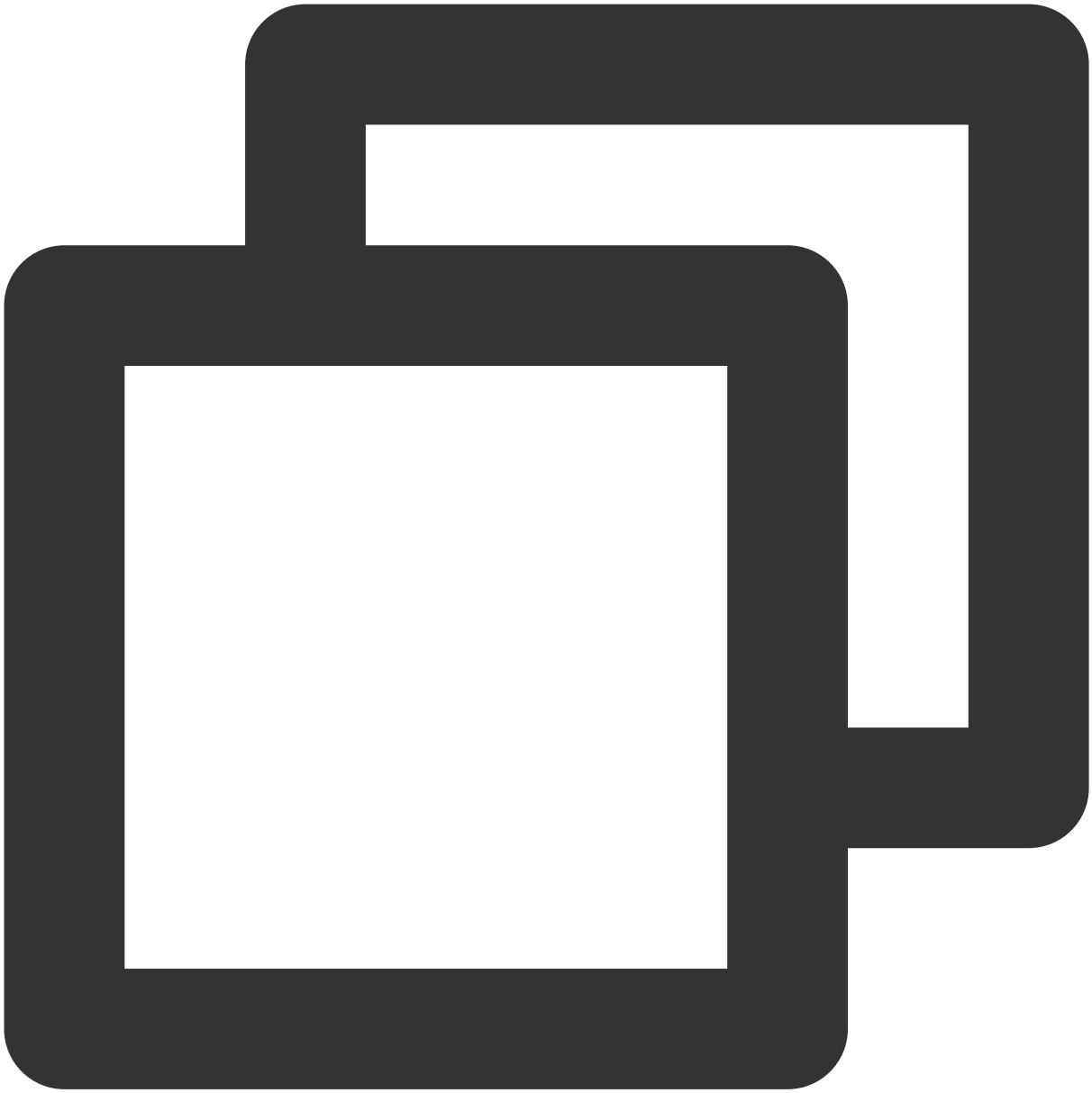
```
.chat-header {
   background-color: #147AFF;// chat background color
```

```
    &-content{
        font-size: 16px;// chat name font size
    }
}
```

设置消息区样式

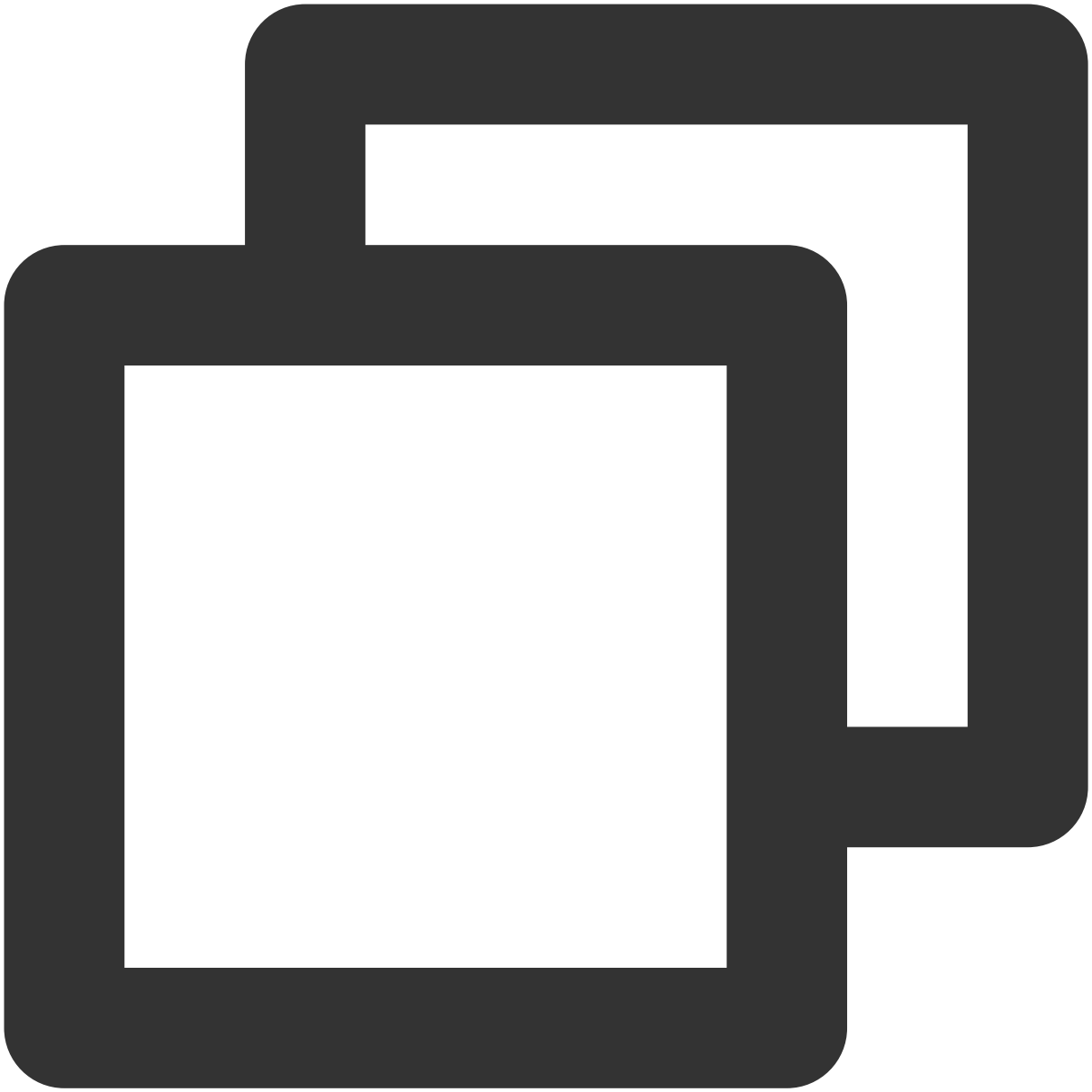
设置聊天窗口的背景

您可以在路径 `src/TUIKit/components/TUIChat/message-list/style/web.scss` 下自定义设置聊天背景色或背景图片。
设置聊天窗口消息区的背景色的示例代码如下：



```
.TUI-chat {
  ...
  &-message-list {
     background-color: #006eff;
  }
}
```

设置聊天窗口消息区的背景图片的示例代码如下：



```
.TUI-chat {
  ...
  &-message-list {
      background-image: url(https://qcloudimg.tencent-cloud.cn/raw/176cddbfb778a4bb
  }
}
```

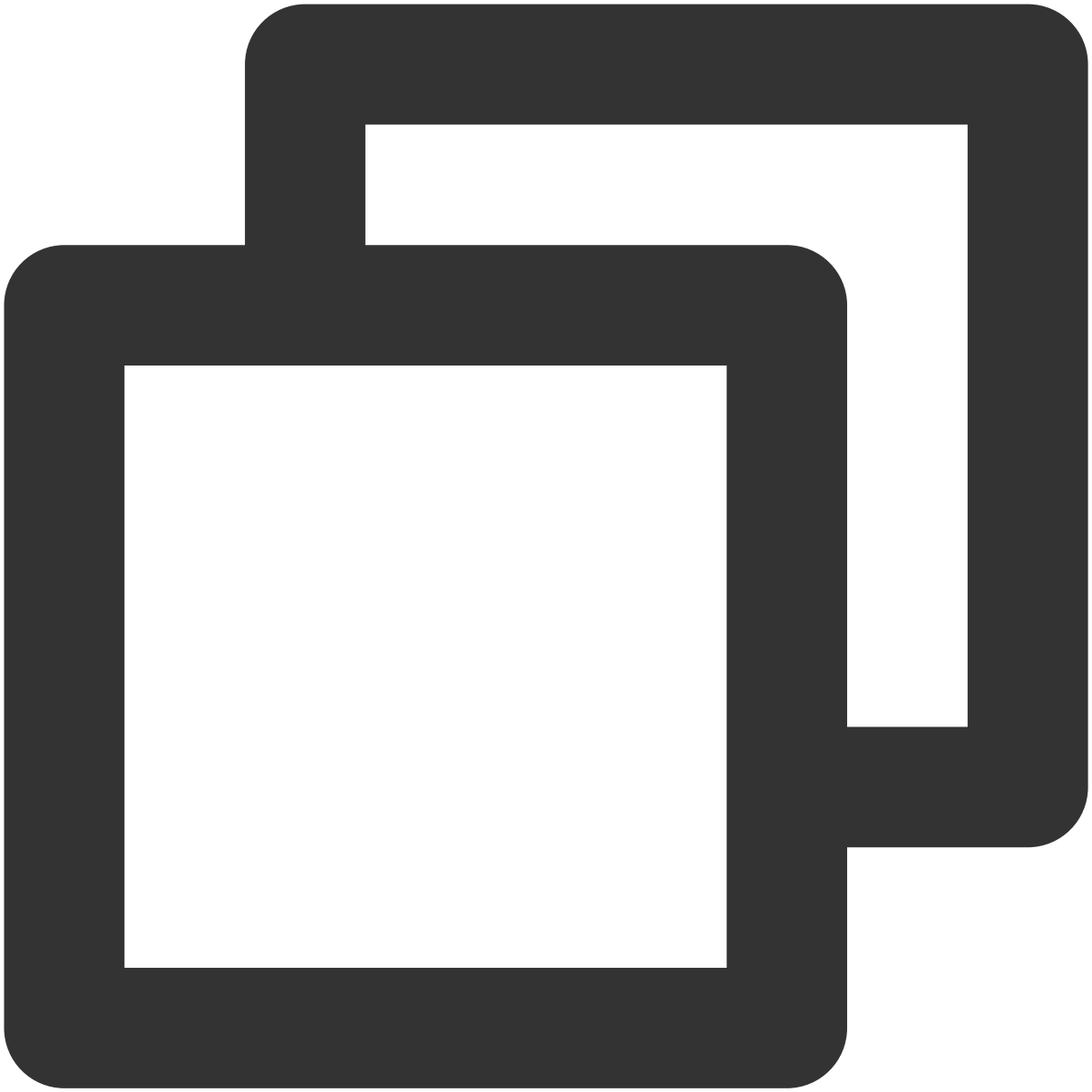
设置发送者的头像样式

消息区中的头像相关代码主要在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中, 采用公共组件 Avatar 实现。如果用户没有设置头像会显示默认头像，您可以自定义设置默认头像、头像是否圆角以及头像大小等。

<Avatar> 组件：

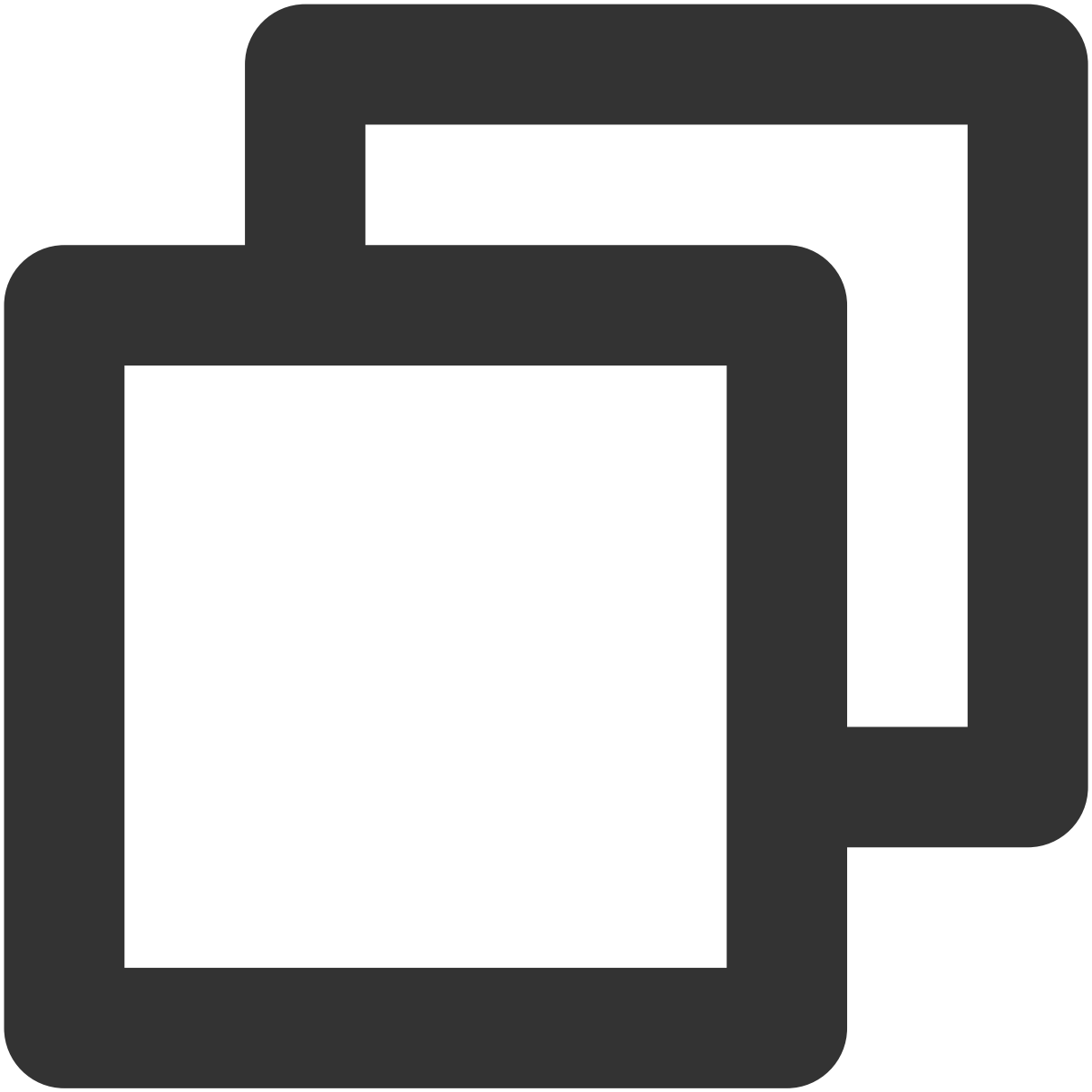
参数名	参数类型	是否必须	默认值
url	string	是	"https://web.sdk.qcloud.com/component/TUIKit/assets/avatar_21.png
size	string	否	"36px"
borderRadius	string	否	"5px"
useSkeletonAnimation	boolean	否	false

设置默认头像配合骨架屏示例代码如下：



```
<Avatar
   useSkeletonAnimation
   :url="message.avatar || ''"
/>
```

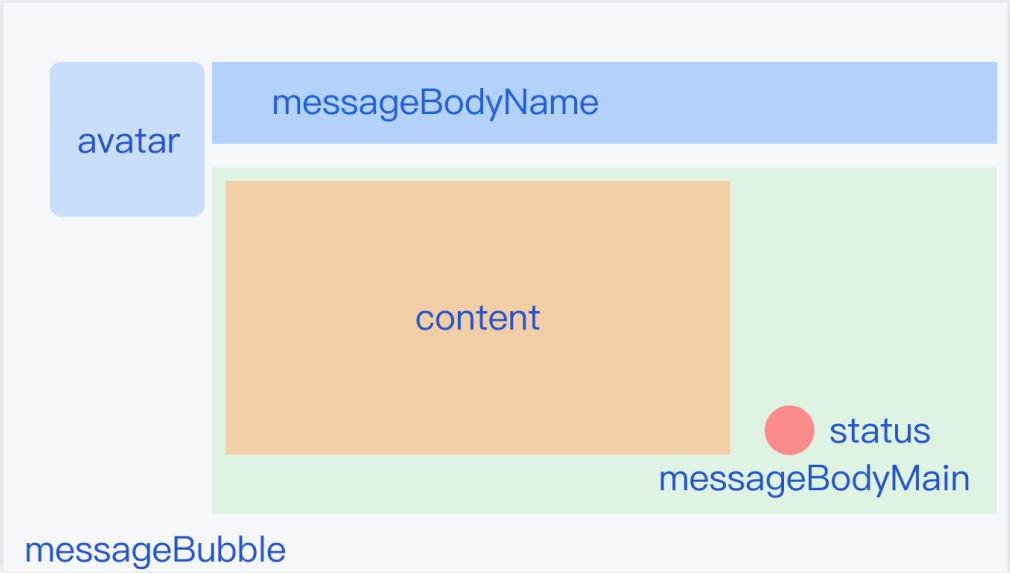
设置头像形状、大小示例代码如下：



```
<Avatar
   useSkeletonAnimation
   :url="message.avatar || ''"
   size="40px"
   borderRadius="0px"
/>
```

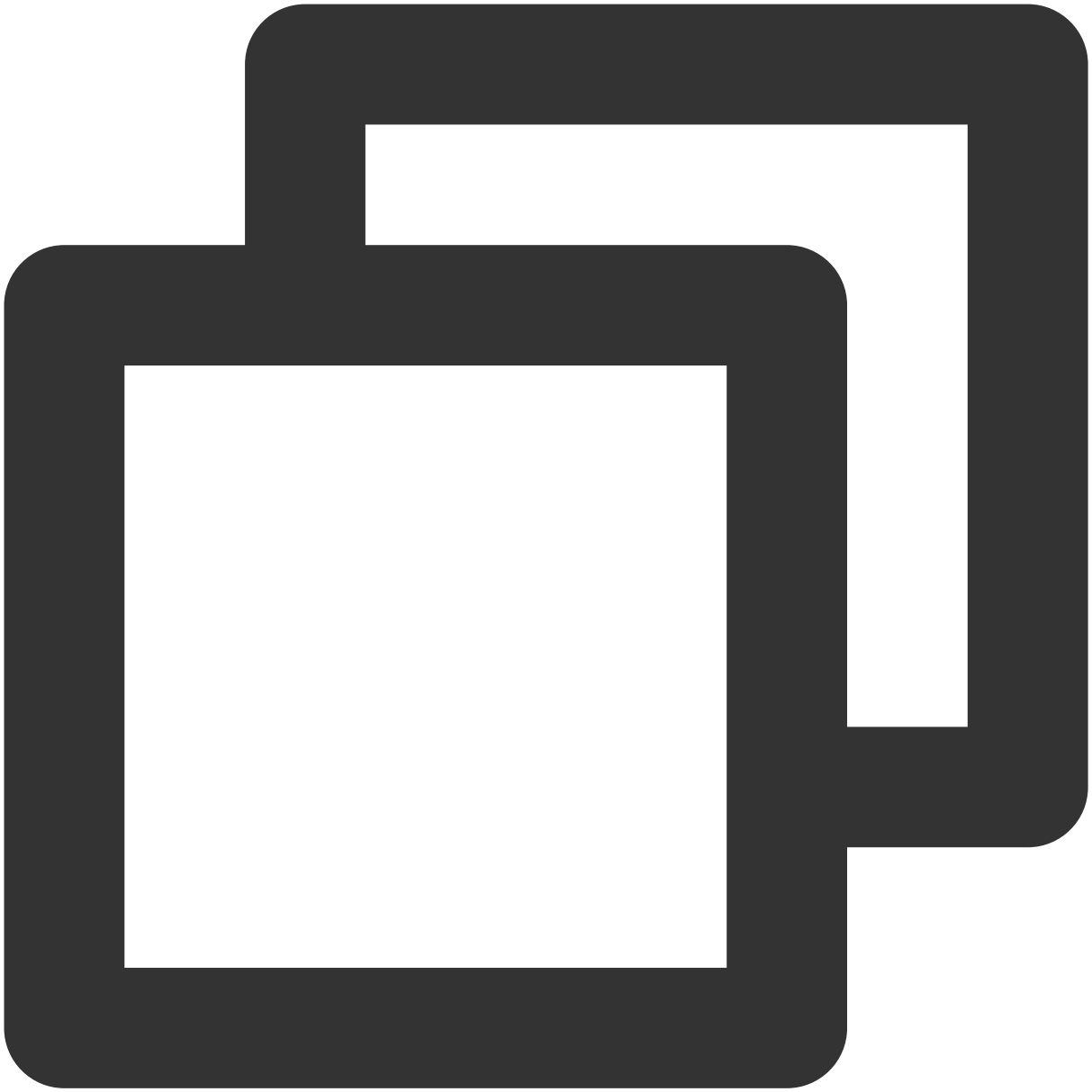
设置气泡的背景色

消息区中单条消息包括 avatar 头像、messageBodyName 昵称区域、content 内容区以及 status 状态区组成。content 区域可以解析展示包括文字、语音、图片、视频、文件、自定义消息等多种类型消息，结构如图所示：



聊天窗口消息区中，左边为对方的气泡，右边为自己的气泡，您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中自定义设置双方的气泡背景。

设置消息气泡颜色示例代码如下：



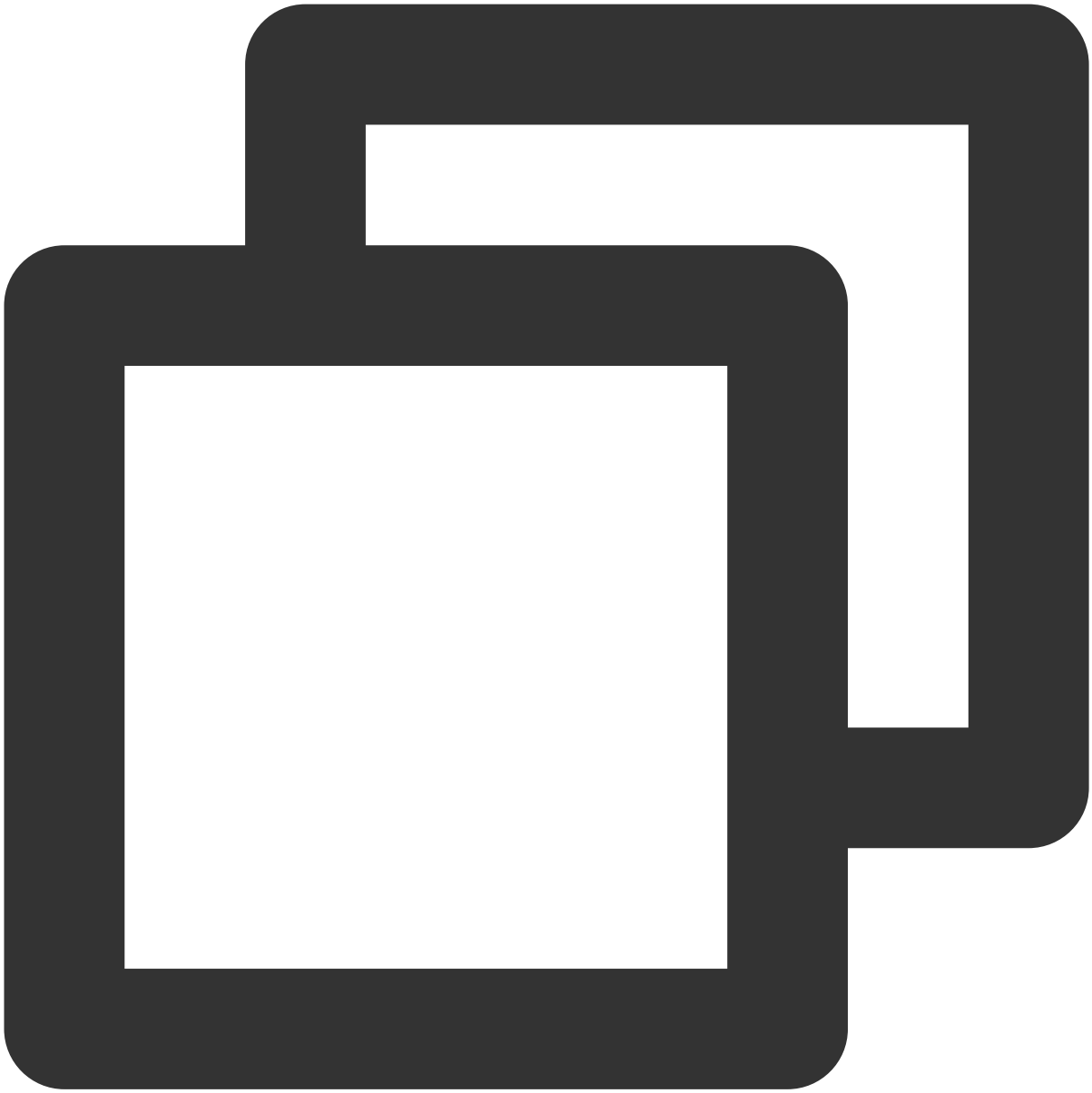
```
.message-bubble {
  .message-bubble-main-content {
    .message-body {
      .message-body-main {
        .content-in {
          background: #fbfbfb; // Set the color of the receiving message bubble
          border-radius: 0px 10px 10px 10px;
        }
        .content-out {
          background: #dceafd; // Set the color of the sender message bubble
          border-radius: 10px 0px 10px 10px;
```

```
            }
          }
        }
      }
    }
```

设置发送者的昵称样式

您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中自定义设置昵称的字体大小与颜色等。

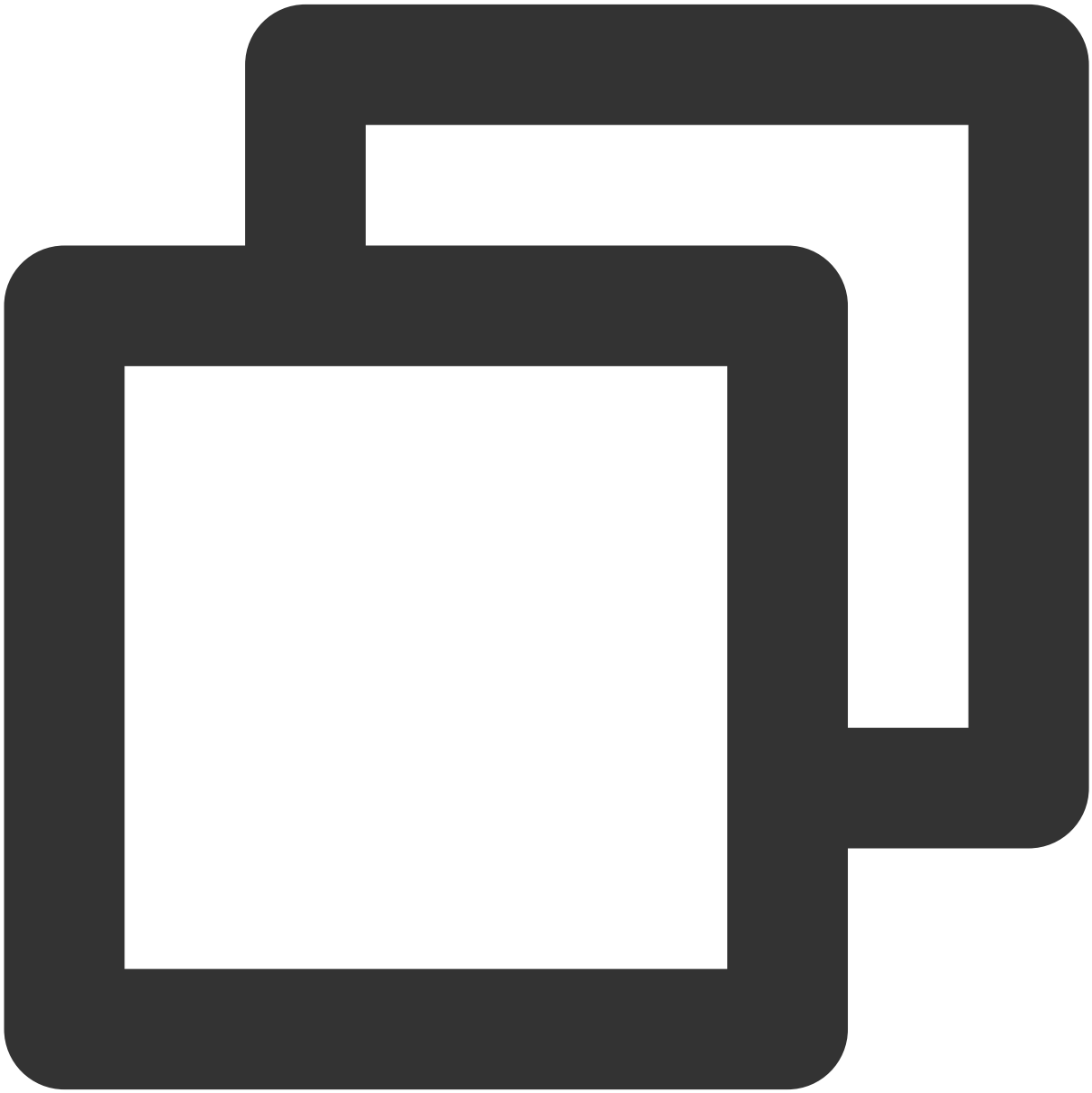
设置发送者昵称样式示例代码如下：



```
.message-bubble {
  .message-bubble-main-content {
    .message-body {
      .message-body-nickName {
        font-weight: 500; // Set the font weight of the sender's nickname
        font-size: 14px; // Set sender nickname font size
        color: #999999; // Set the font color of the sender's nickname
      }
    }
  }
}
```

设置聊天内容样式

您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-text.vue` 文件中自定义设置聊天内容的字体大小、双方字体颜色、emoji 表情大小等。
设置聊天内容样式示例代码如下：

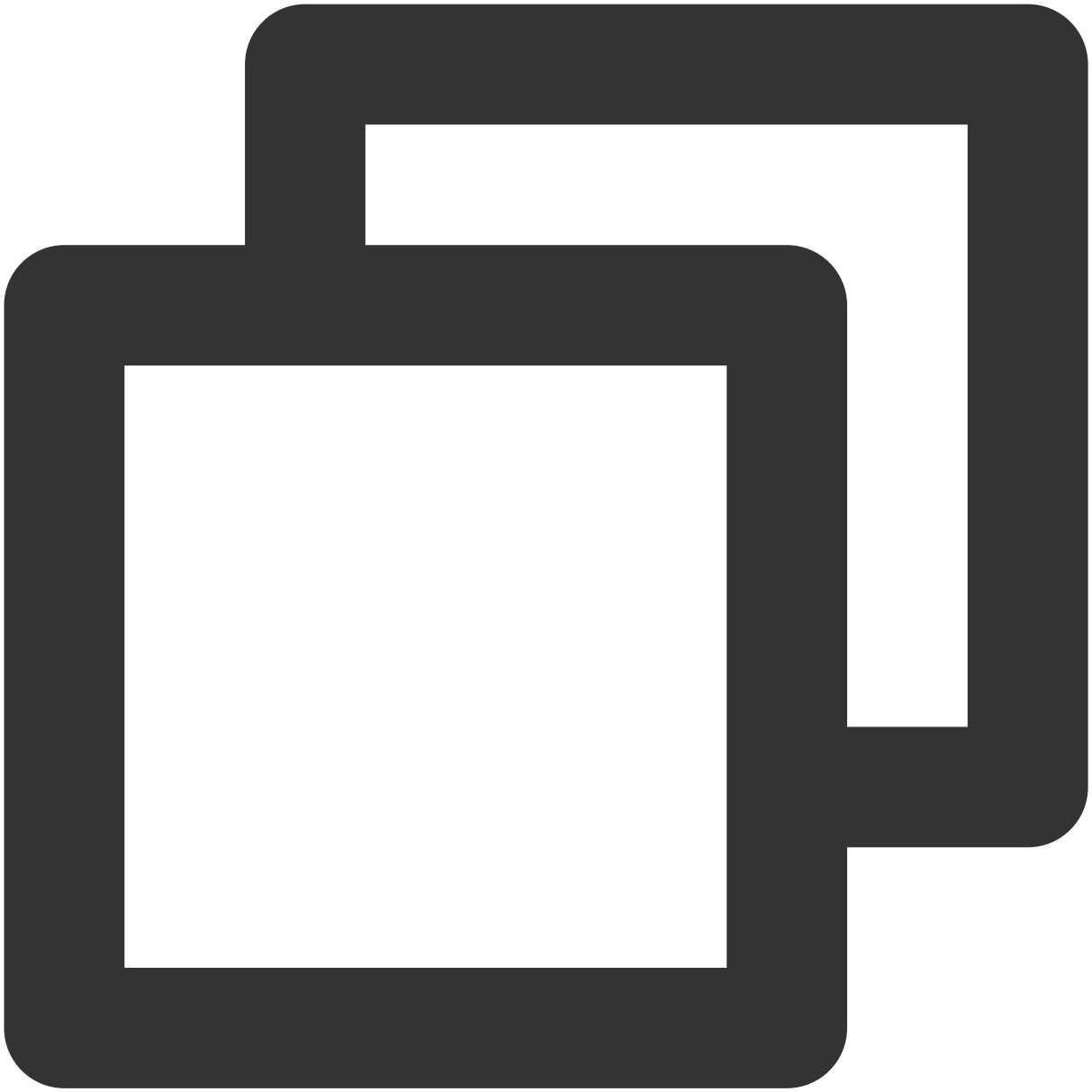


```
.emoji {
    width: 20px;// emoji width
    height: 20px;// emoji height
}
```

```
.text {
    white-space: pre-wrap;
    font-size: 14px;// text message font size
    color: #999999;// text message font color
}
```

设置聊天的提示信息样式

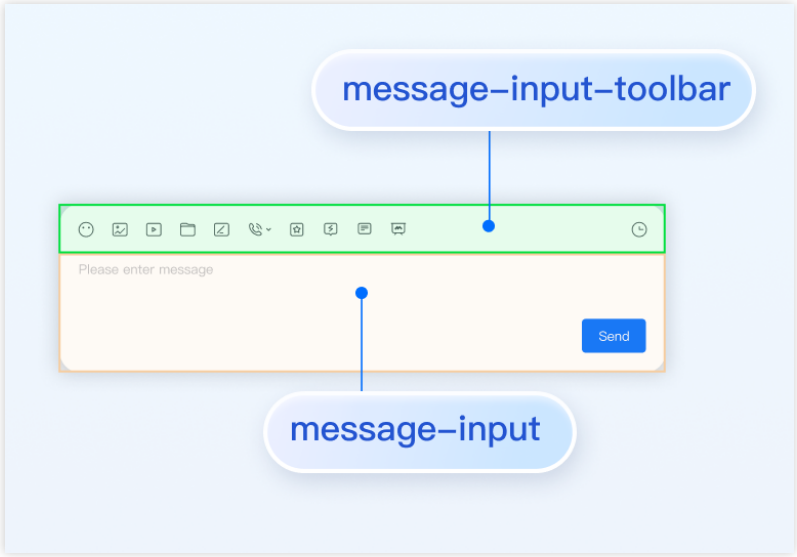
您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-tip.vue` 文件中自定义设置提示信息的背景、字体大小以及字体颜色等。
示例代码如下：



```
.message-tip {
    margin: 0 auto;
    color: #999999;// message tip font color
    font-size: 14px;// message tip font size
    background: red;// message tip background color
}
```

设置输入区域 InputView

输入区域包含文字输入、表情输入、图片发送、视频发送、文件发送、评价发送、常用语发送等功能。

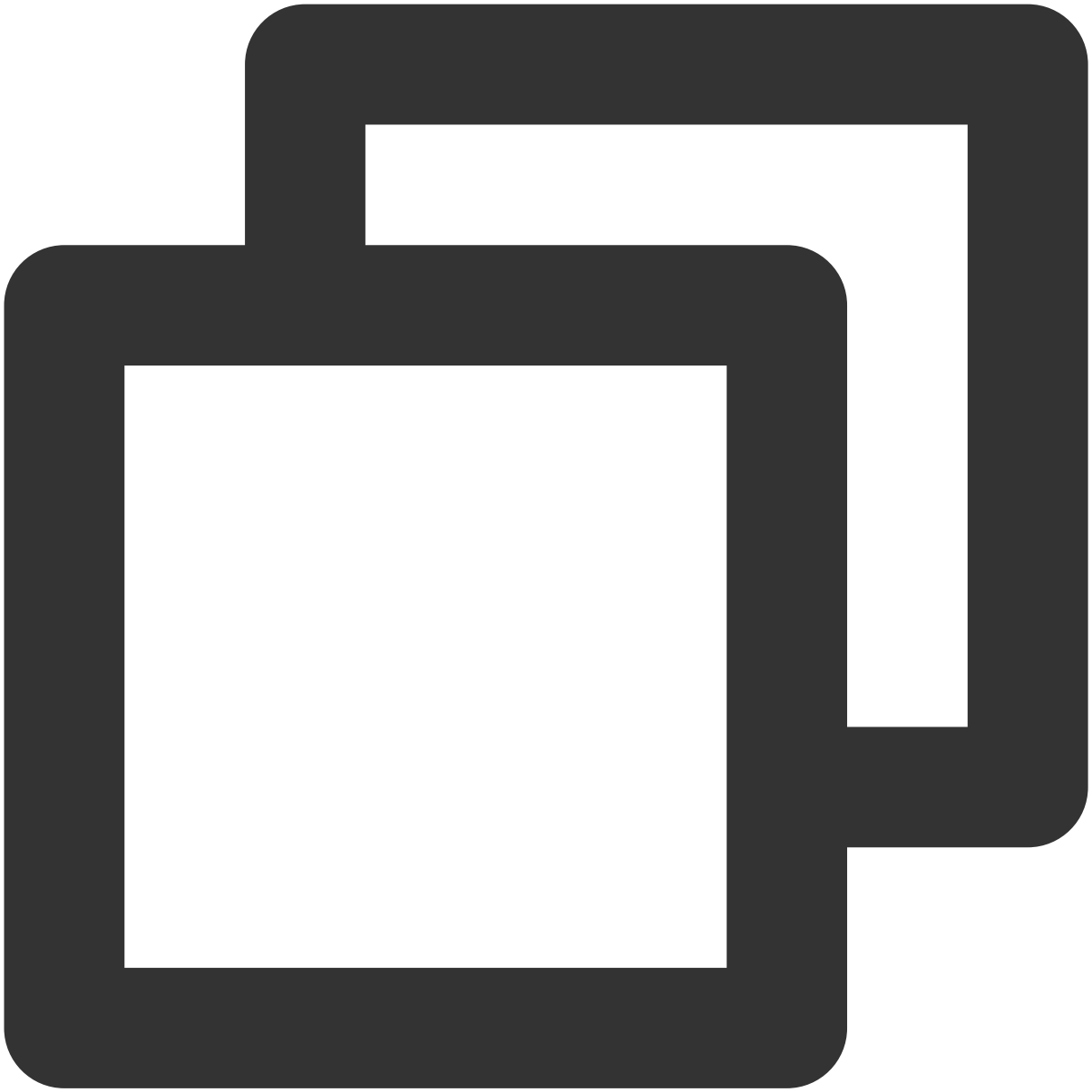


隐藏不需要的功能

您可以自定义隐藏功能模块的发送图片、发送文件以及发送评价等功能。

输入区功能模块在 `src/TUIKit/components/TUIChat/message-input-toolbar/index.vue` 文件中进行注册，您可以在该文件中注释或删除您不需要的功能。

例如：不需要使用常用语功能，注释常用语 `<Words>` 组件，示例代码如下：



```
<div>
  <div>
    <!-- Emoji Picker -->
    <EmojiPicker v-if="!isUniFrameWork"></EmojiPicker>
    <!-- Taking photos, only available on uniapp -->
    <ImageUpload v-if="isUniFrameWork" imageSourceType="camera"></ImageUpload>
    <!-- Image Upload -->
    <ImageUpload imageSourceType="album"></ImageUpload>
    <!-- File Upload -->
    <FileUpload v-if="!isUniFrameWork"></FileUpload>
    <!-- Video Upload -->
```

```
      <VideoUpload videoSourceType="album"></VideoUpload>
      <!-- Taking videos, only available on uniapp -->
      <VideoUpload v-if="isUniFrameWork" videoSourceType="camera"></VideoUpload>
      <!-- Evaluate -->
      <Evaluate></Evaluate>
      <!-- Commonly Used Phrases -->
      <!-- <Words></Words> -->
    </div>
  </div>
```

交流与反馈

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

H5(Vue)

最近更新时间：2024-01-31 12:36:32

本文介绍如何设置 H5 界面风格

设置会话列表

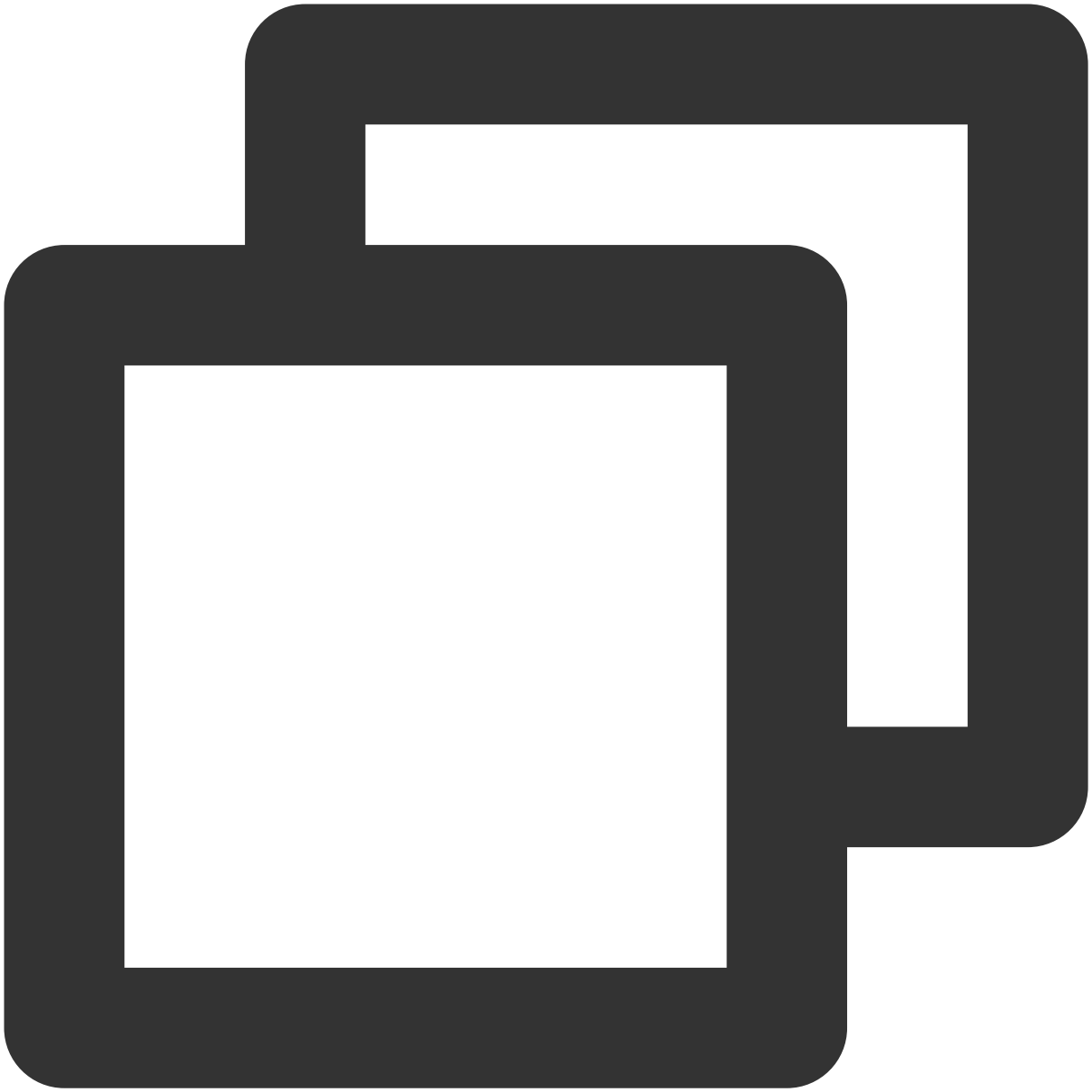
TUIConversation 提供会话列表功能。会话列表主要由会话列表区组成，会话列表区提供了 UI 样式可供修改。

设置会话列表样式

登录后 TUIKit 会根据用户名从 SDK 读取该用户的会话列表。会话列表提供一些常用功能定制，例如，头像样式、背景、字体大小、点击与长按事件等。

会话列表中单个列表项展示主要在路径 `TUIKit/components/TUIConversation/conversation-list/index.vue` 文件中。

示例代码如下：

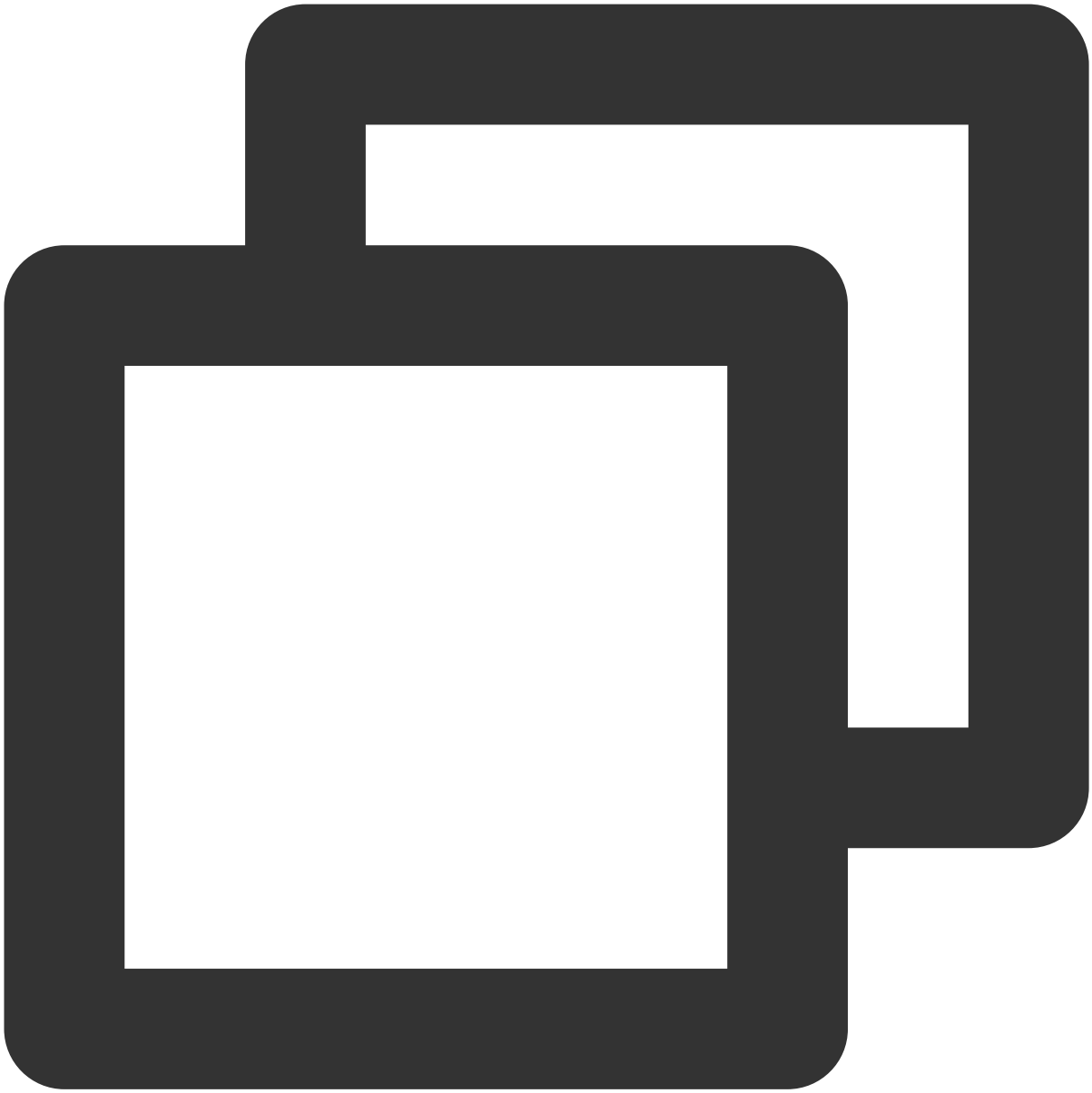


```
<template>
  <div class="tui-conversation-list">
    <!-- Conversation List operation panel -->
    <ActionsMenu .../>
    <!-- Conversation List Main -->
    <div v-for="(conversation, index) in conversationList" ...>
      <!-- Conversation List Item -->
      <div :class="['TUI-conversation-item']">
        <aside class="left">
          <!-- Avatar -->
          <img class="avatar" :src="conversation.getAvatar()" />
```

```
            <!-- User Online Status -->
            <div ... :class="['online-status']"></div>
            <!-- Conversation Unread Count -->
            <span class="num" ...>...</span>
            <!-- Conversation Unread Red Dot (displayed in Do Not Disturb mode) -->
            <span class="num-notify" ...>...</span>
          </aside>
          <div class="content">
            <div class="content-header">
              <!-- Conversation Name -->
              <label class="content-header-label">
                <p class="name">{{ conversation.getShowName() }}</p>
              </label>
              <!-- Conversation Last Message -->
              <div class="middle-box">
                <!-- Conversation Last Message When Mentiond -->
                <span class="middle-box-at" ...>{{ conversation.getGroupAtInfo() }}</
                <!-- Conversation Last Message Content -->
                <p class="middle-box-content">{{ conversation.getLastMessage("text")
              </div>
            </div>
            <div class="content-footer">
              <!-- Conversation Lastest Message Time -->
              <span class="time">{{ conversation.getLastMessage("time") }}</span>
              <!-- Conversation Muted Flag -->
              <Icon v-if="conversation.isMuted" :file="muteIcon"></Icon>
            </div>
        ...
  </template>
```

您可以在路径 `TUIKit/components/TUIConversation/conversation-list/style/h5.scss` 下设置会话列表中列表项的样式。

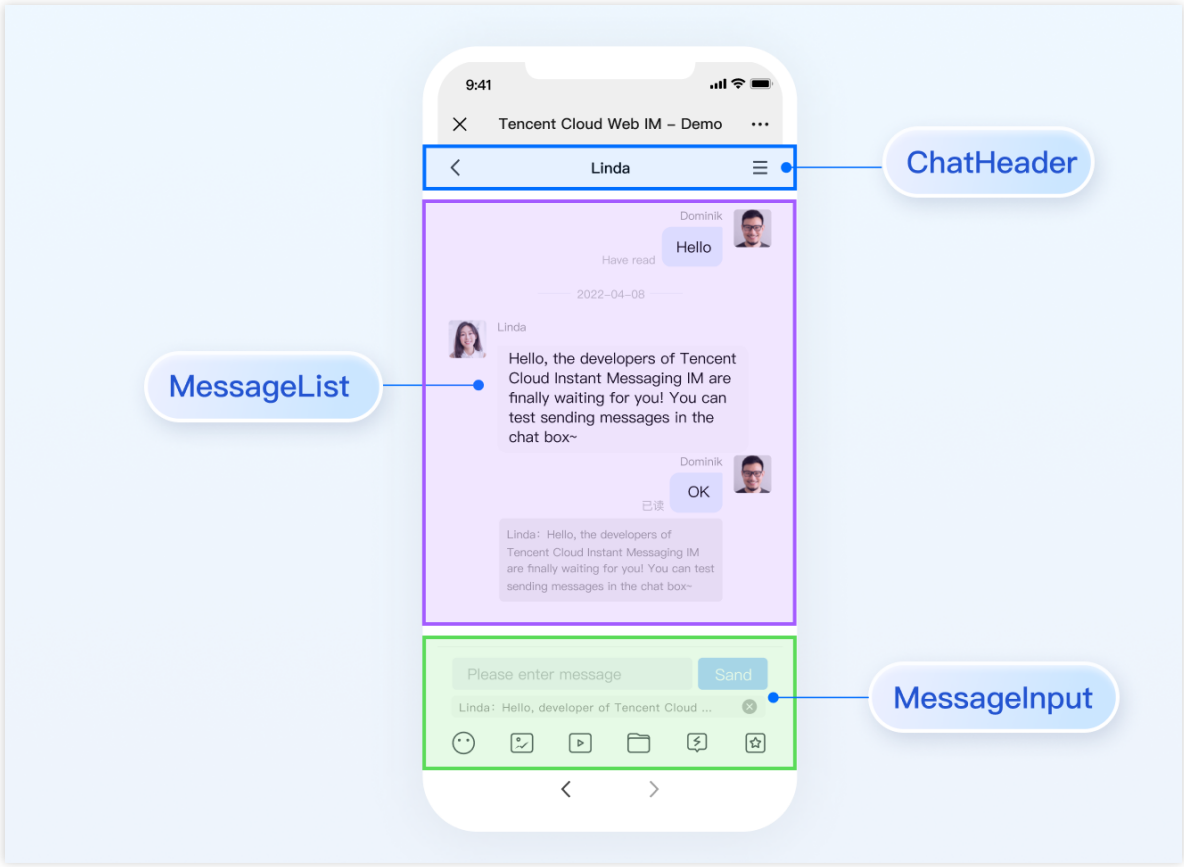
设置会话列表中头像样式示例代码如下：



```
.TUI-conversation-content {
  .TUI-conversation-item {
    .left {
      .avatar {
        width: 40px; // avatar width
        height: 40px; // avatar height
        border-radius: 0px; // avatar border radiu
      }
    }
  }
}
```

设置聊天窗口的样式

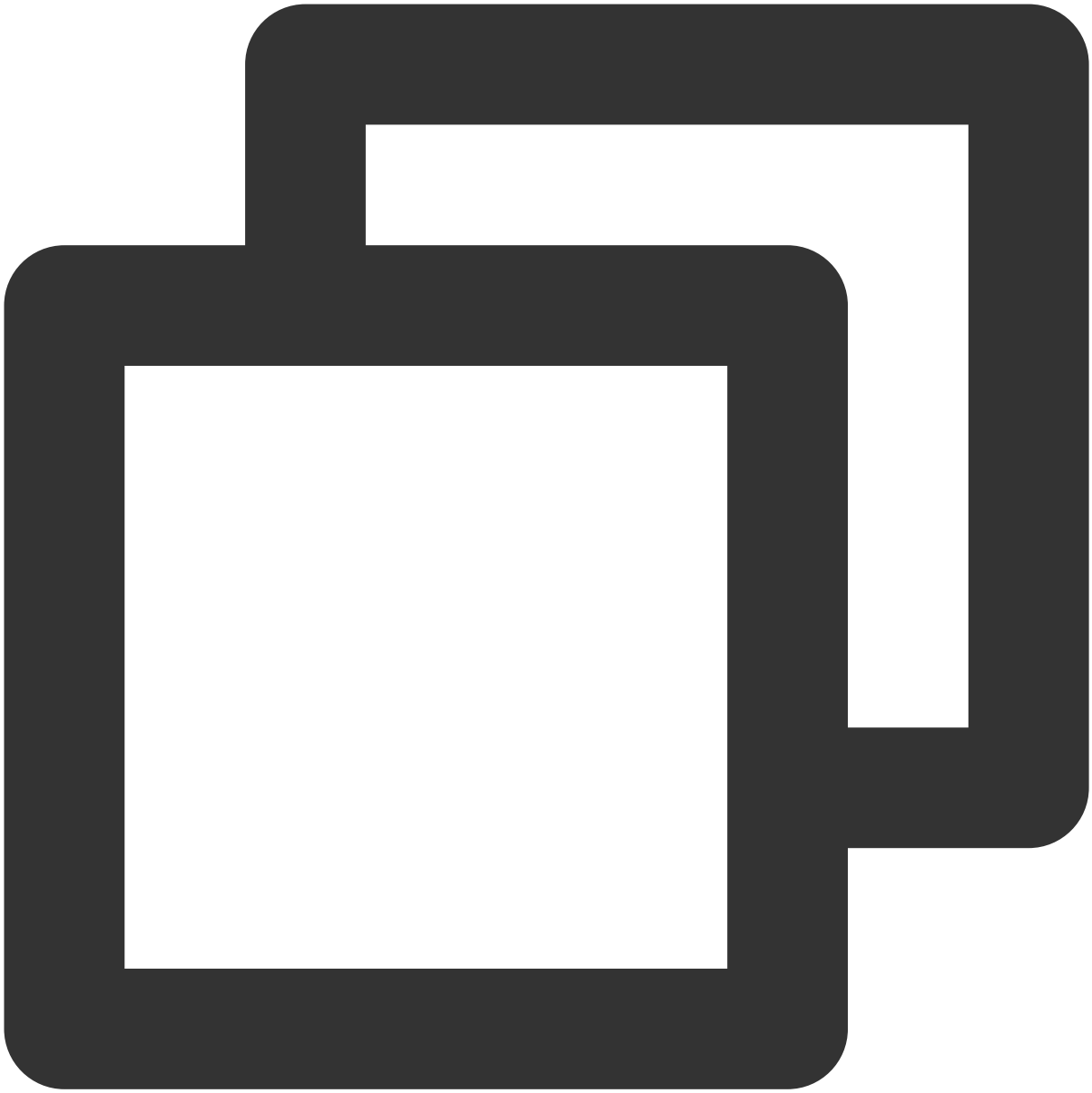
TUIChat 提供聊天窗口。聊天窗口包含三个区域，从上到下为标题栏区、消息区和输入区，如下图所示：



聊天窗口相关配置主要在路径 `src/TUIKit/components/TUIChat` 文件目录中。

设置标题栏区样式

标题栏由左中右三个区域组成，如下图所示：
聊天窗口标题栏相关代码主要在路径 `src/TUIKit/components/TUIChat/chat-header/index.vue` 文件中。聊天窗口标题栏区提供一些常用功能定制，例如，背景、字体大小、按钮图标、点击事件、功能开关等。
示例代码如下：

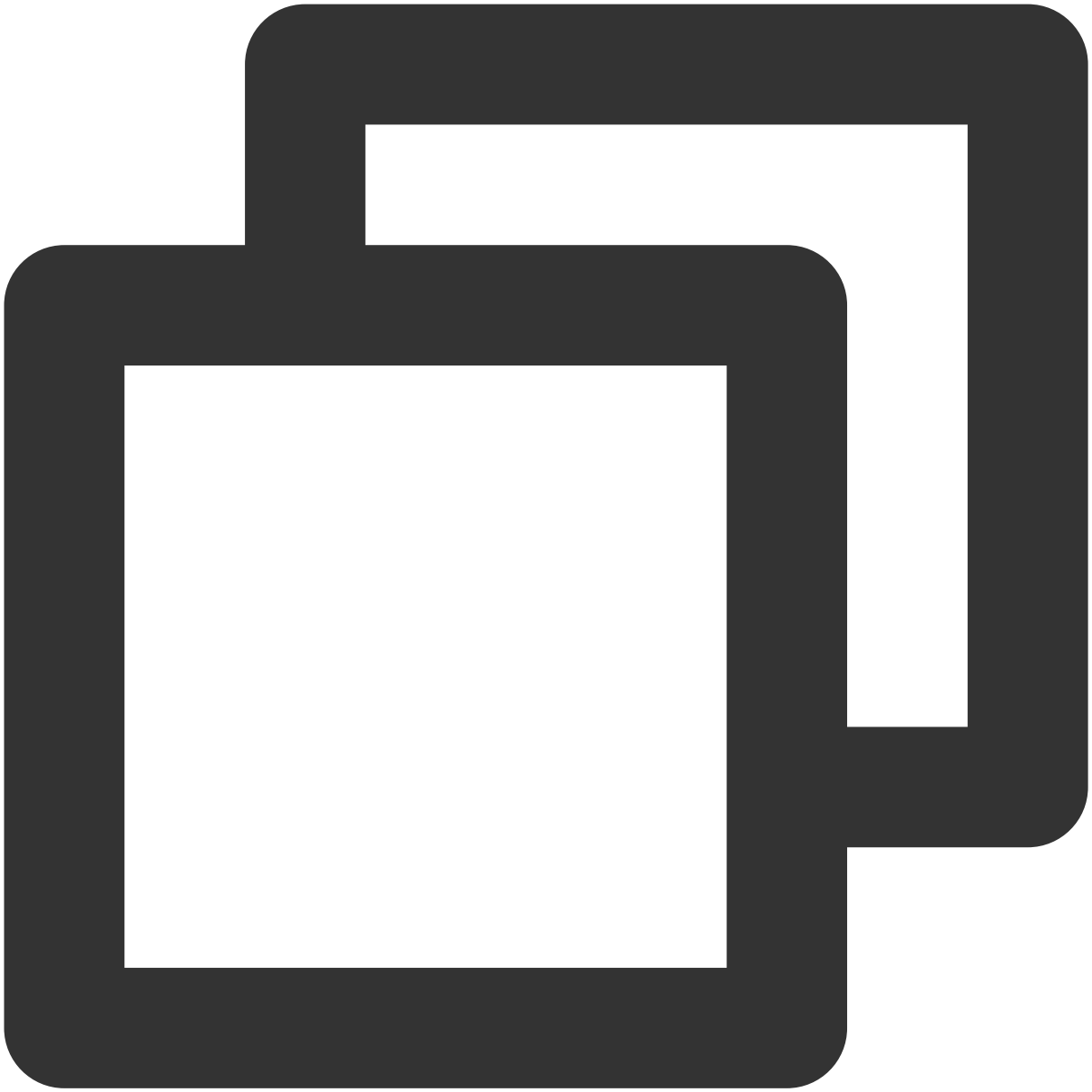


```
<template>
  <div :class="['chat-header', !isPC && 'chat-header-h5']">
    <!-- H5 Back Button-->
    <div
      v-show="!isPC"
      :class="['chat-header-back', !isPC && 'chat-header-h5-back']"
    >
      <Icon :file="backSVG"></Icon>
    </div>
    <!-- Chat name / [Typing...] status prompt-->
    <div :class="['chat-header-content', ...]">
```

```
        {{ currentConversationName }}
      </div>
      <!-- Group chat settings extension -->
      <div :class="['chat-header-setting', ...]">
        <div v-for="(item, index) in extensions" :key="index" @click.stop="handleExte
          <Icon :file="item.icon"></Icon>
        </div>
      </div>
    </div>
  </template>
```

您可以在 `src/TUIKit/components/TUIChat/chat-header/index.vue` 文件中设置聊天窗口标题栏区样式。

设置聊天窗口标题栏区的字体大小和背景色示例代码如下：

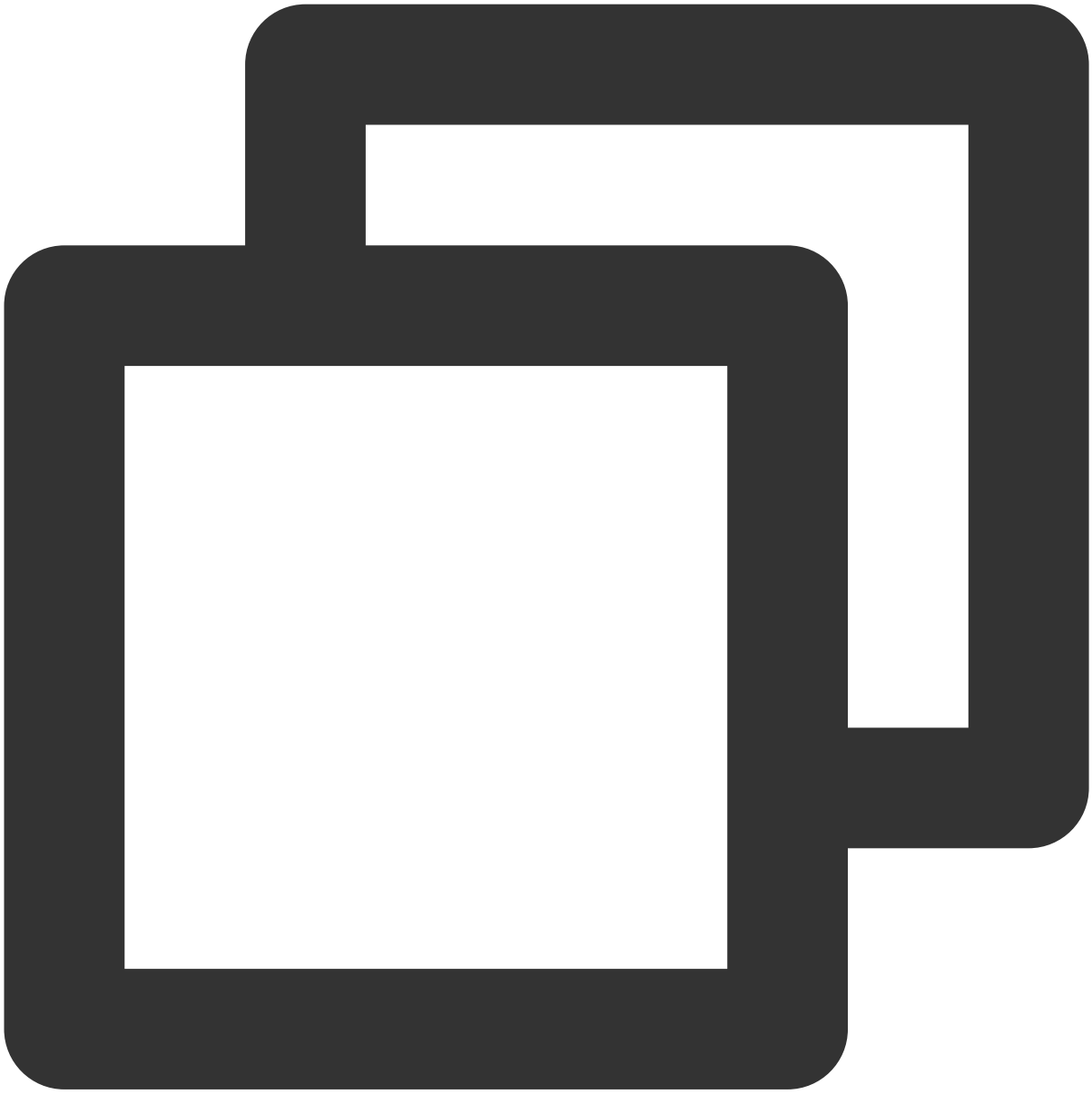


```
.chat-header-h5 {
    background-color: #147AFF;// chat background color
    &-content{
      font-size: 16px;// chat name font size
    }
}
```

设置消息区样式

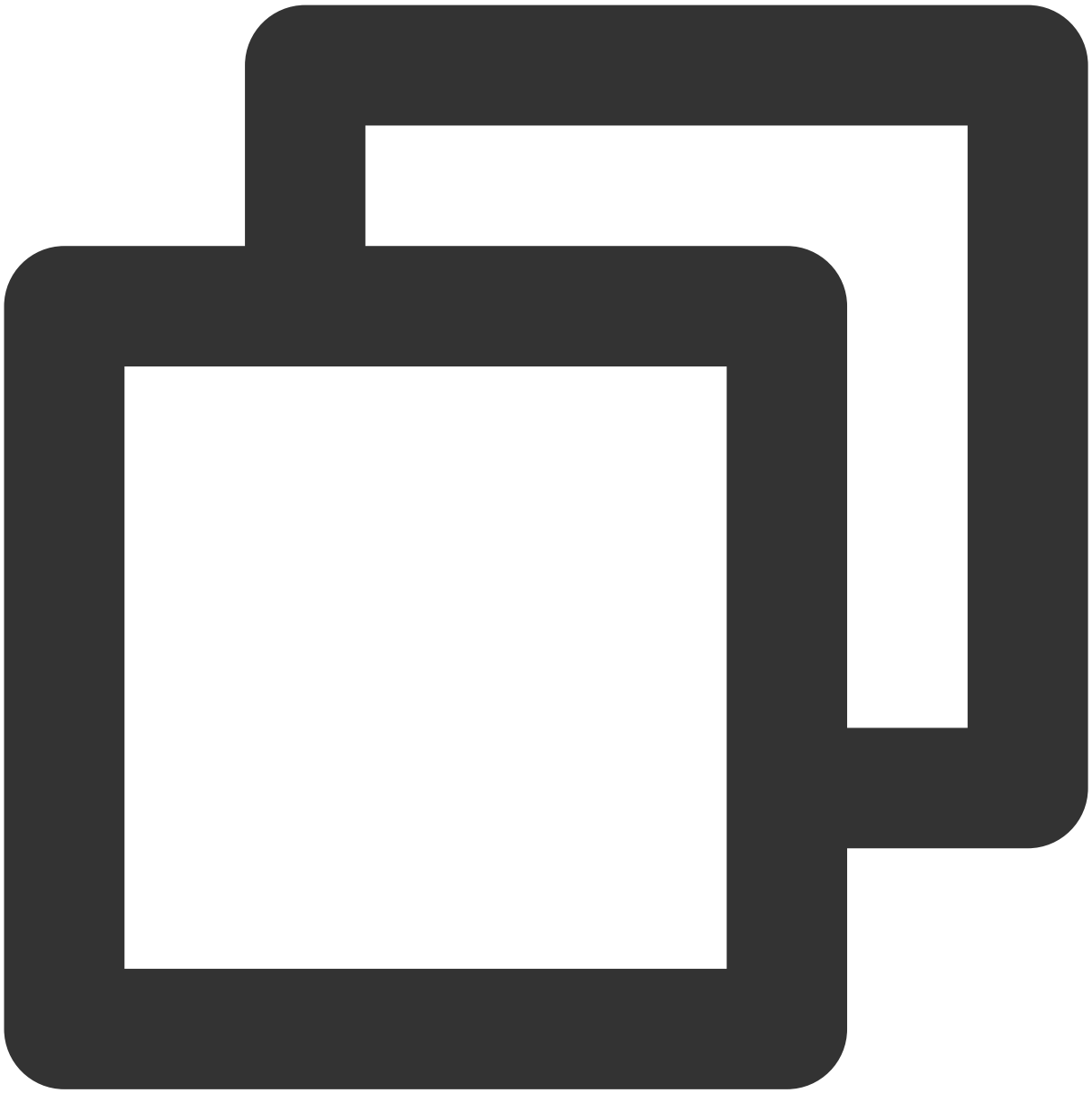
设置聊天窗口的背景

您可以在路径 `src/TUIKit/components/TUIChat/message-list/style/h5.scss` 下自定义设置聊天背景色或背景图片。
设置聊天窗口消息区的背景色的示例代码如下：



```
.TUI-chat-h5 {
  ...
  &-message-list {
     background-color: #006eff;
  }
}
```

设置聊天窗口消息区的背景图片的示例代码如下：



```
.TUI-chat-h5 {
  ...
  &-message-list {
      background-image: url(https://qcloudimg.tencent-cloud.cn/raw/176cddbfb778a4bb
  }
}
```

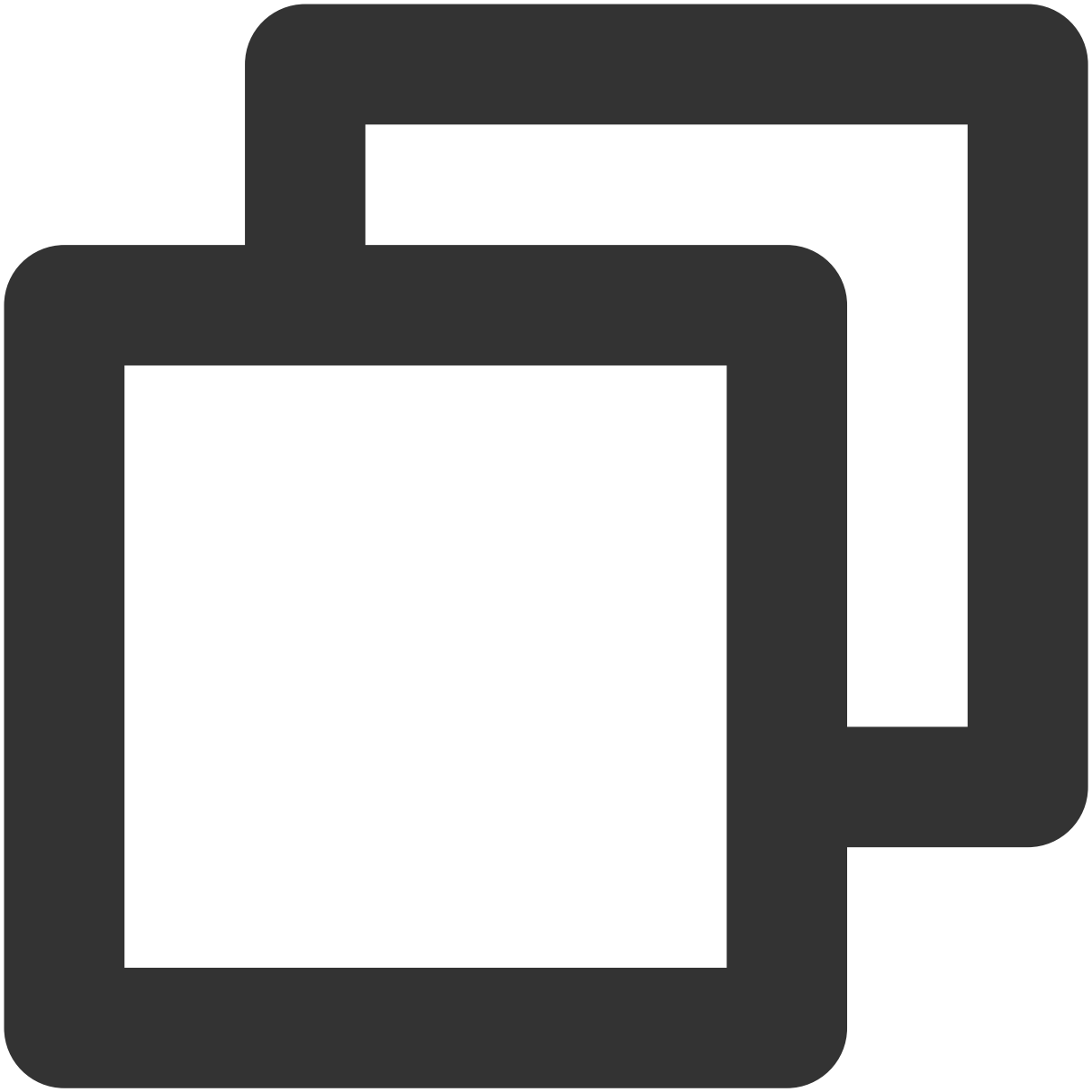
设置发送者的头像样式

消息区中的头像相关代码主要在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中, 采用公共组件 Avatar 实现。如果用户没有设置头像会显示默认头像，您可以自定义设置默认头像、头像是否圆角以及头像大小等。

<Avatar> 组件：

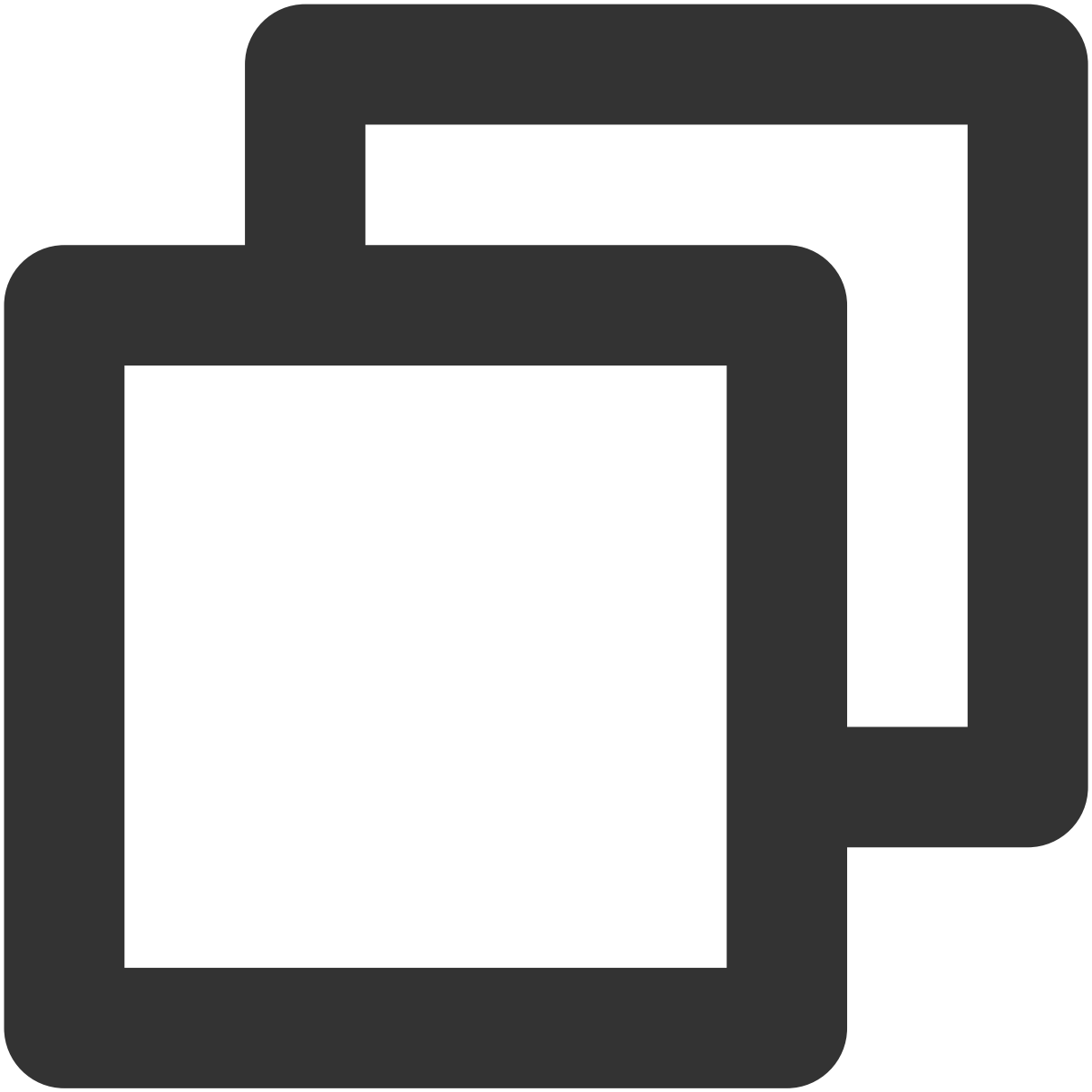
参数名	参数类型	是否必须	默认值
url	string	是	"https://web.sdk.qcloud.com/component/TUIKit/assets/avatar_21.png
size	string	否	"36px"
borderRadius	string	否	"5px"
useSkeletonAnimation	boolean	否	false

设置默认头像配合骨架屏示例代码如下：



```
<Avatar
   useSkeletonAnimation
   :url="message.avatar || ''"
/>
```

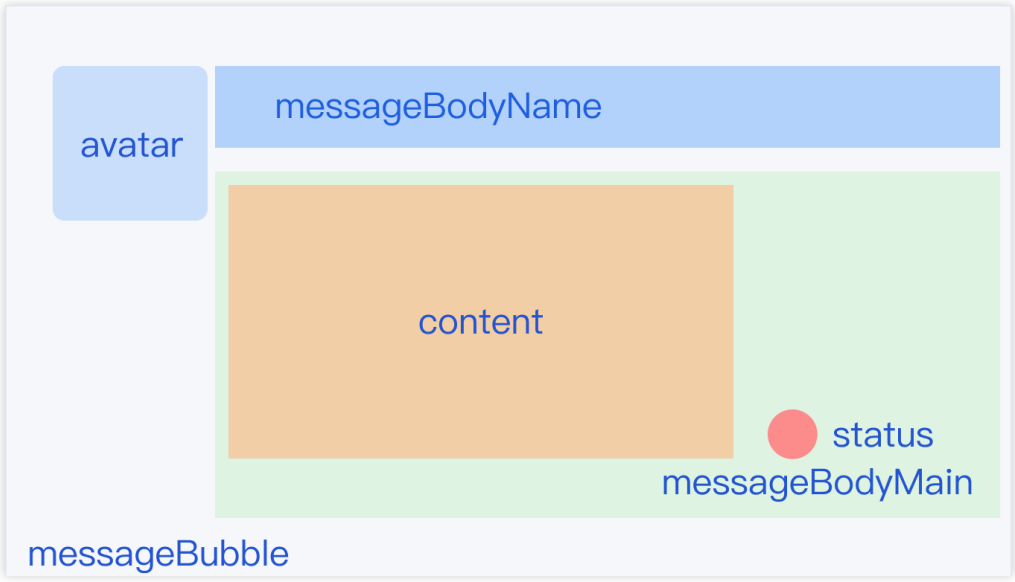
设置头像形状、大小示例代码如下：



```
<Avatar
   useSkeletonAnimation
   :url="message.avatar || ''"
   size="40px"
   borderRadius="0px"
/>
```

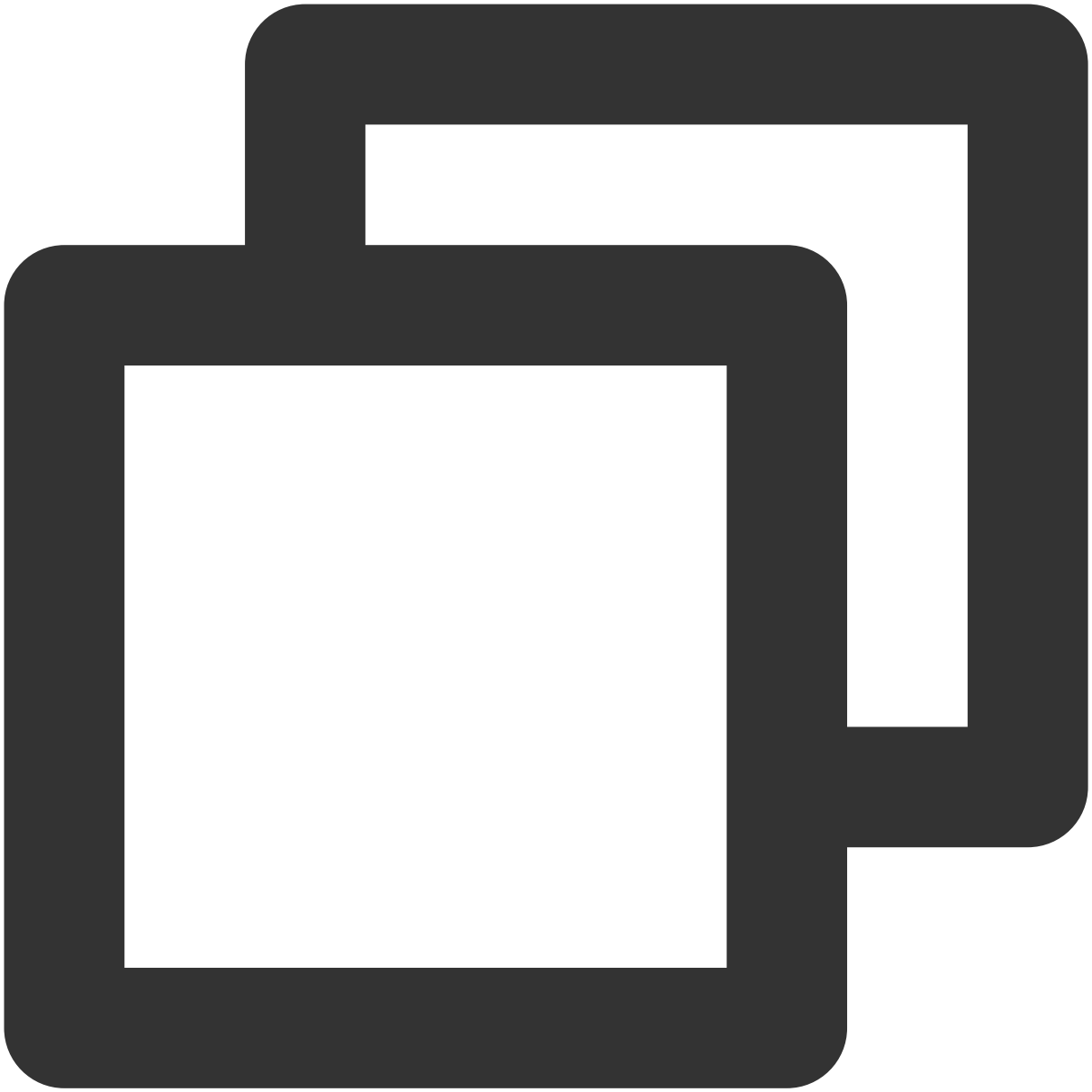
设置气泡的背景色

消息区中单条消息包括 avatar 头像、messageBodyName 昵称区域、content 内容区以及 status 状态区组成。content 区域可以解析展示包括文字、语音、图片、视频、文件、自定义消息等多种类型消息，结构如图所示：



聊天窗口消息区中，左边为对方的气泡，右边为自己的气泡，您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中自定义设置双方的气泡背景。

设置消息气泡颜色示例代码如下：



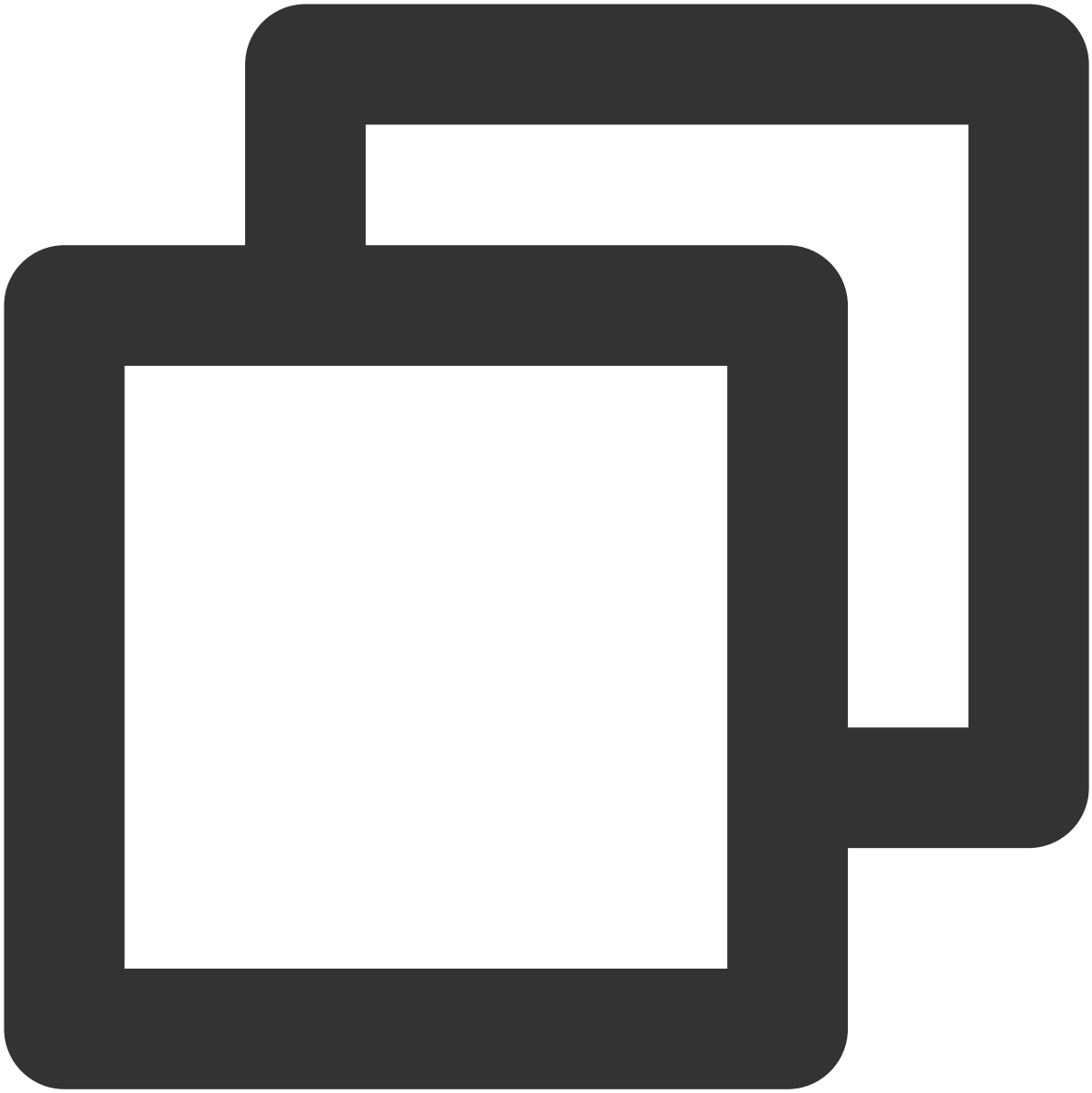
```
.message-bubble {
  .message-bubble-main-content {
    .message-body {
      .message-body-main {
        .content-in {
          background: #fbfbfb; // Set the color of the receiving message bubble
          border-radius: 0px 10px 10px 10px;
        }
        .content-out {
          background: #dceafd; // Set the color of the sender message bubble
          border-radius: 10px 0px 10px 10px;
```

```
            }
          }
        }
      }
    }
```

设置发送者的昵称样式

您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-bubble.vue` 文件中自定义设置昵称的字体大小与颜色等。

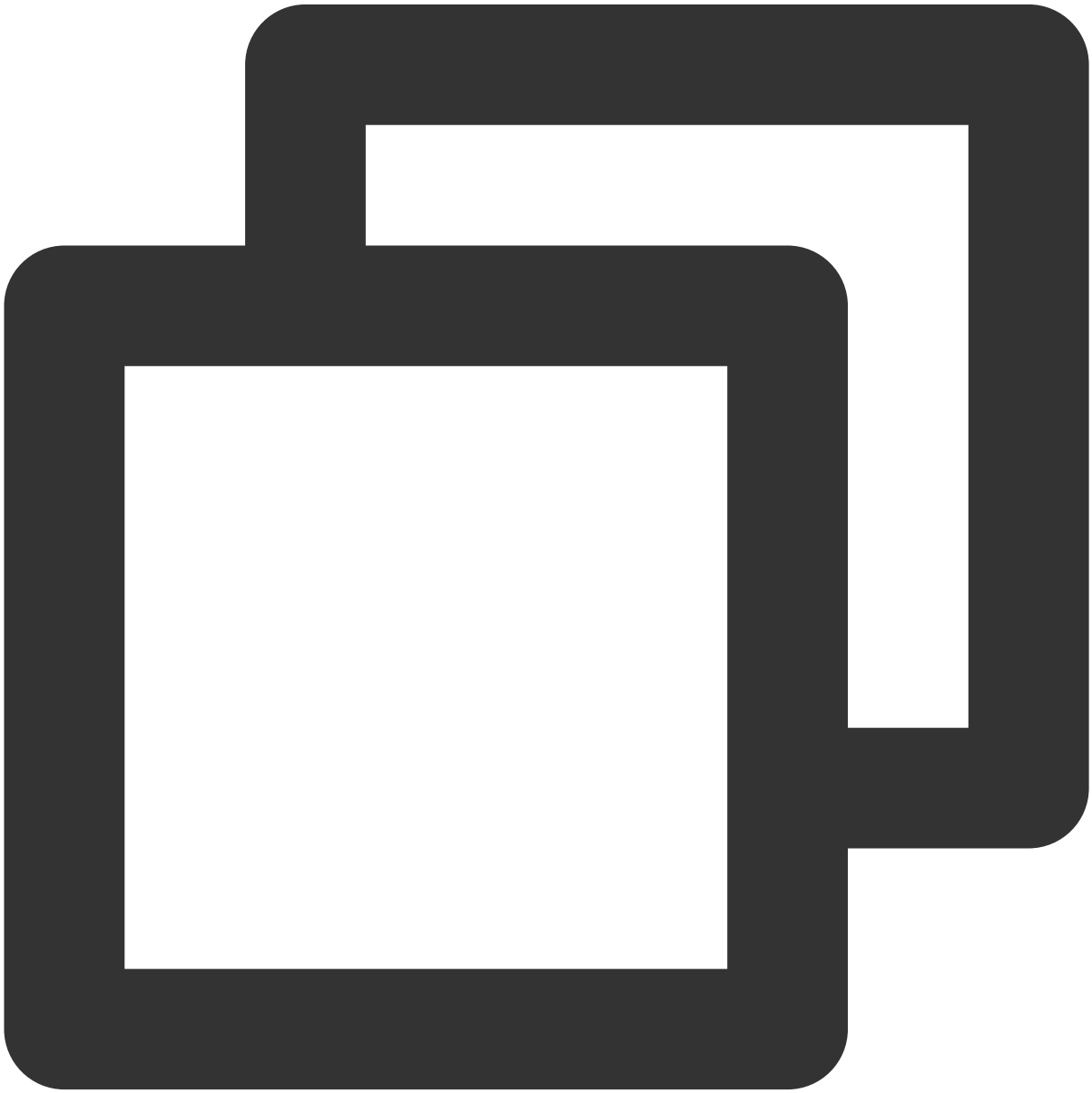
设置发送者昵称样式示例代码如下：



```
.message-bubble {
  .message-bubble-main-content {
    .message-body {
      .message-body-nickName {
        font-weight: 500; // Set the font weight of the sender's nickname
        font-size: 14px; // Set sender nickname font size
        color: #999999; // Set the font color of the sender's nickname
      }
    }
  }
}
```

设置聊天内容样式

您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-text.vue` 文件中自定义设置聊天内容的字体大小、双方字体颜色、emoji 表情大小等。
设置聊天内容样式示例代码如下：



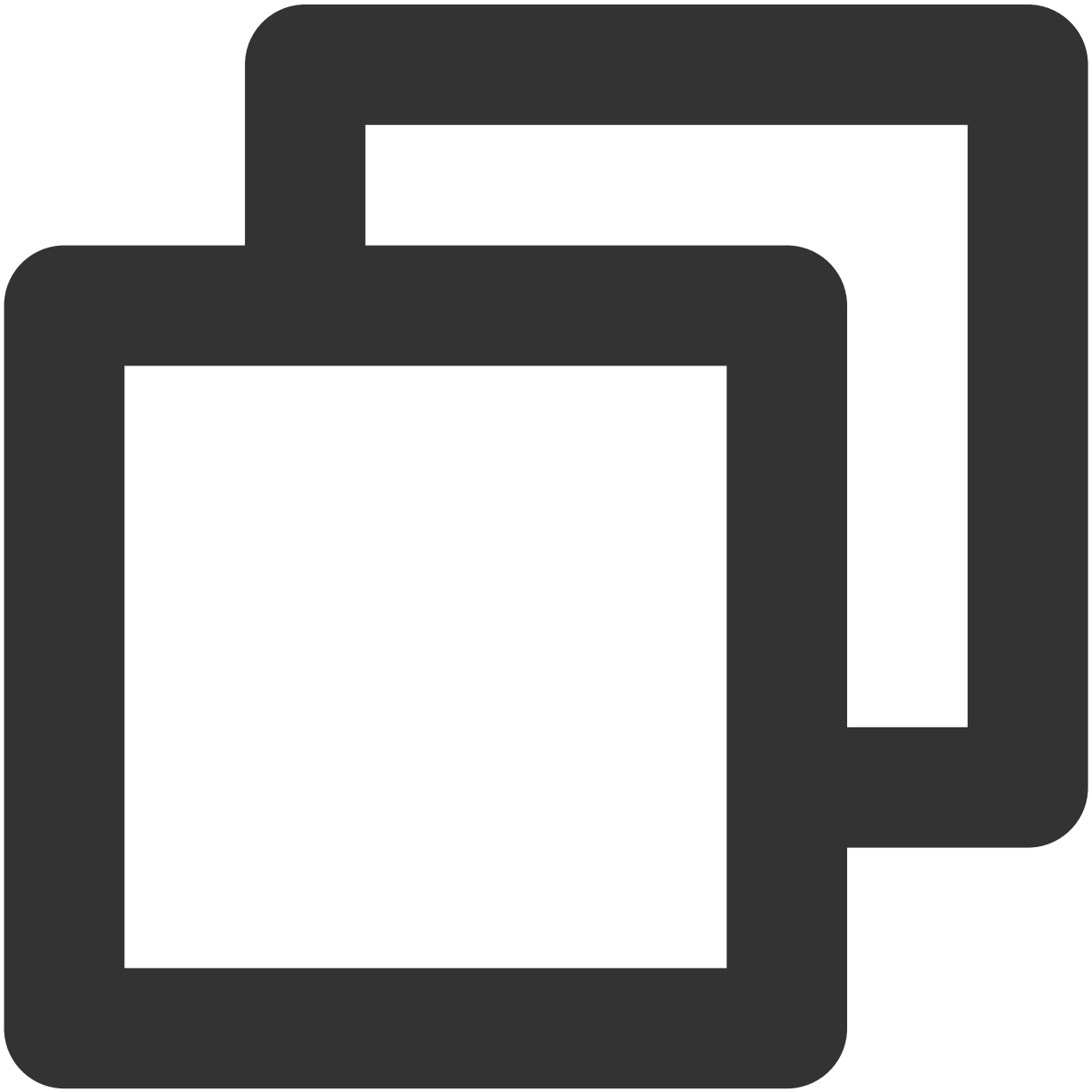
```
.text-img {
    width: 20px;// emoji width
    height: 20px;// emoji height
.text-box {
```

```
    white-space: pre-wrap;
    font-size: 14px;// text message font size
    color: #999999;// text message font color
}
```

设置聊天的提示信息样式

您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-tip.vue` 文件中自定义设置提示信息的背景、字体大小以及字体颜色等。

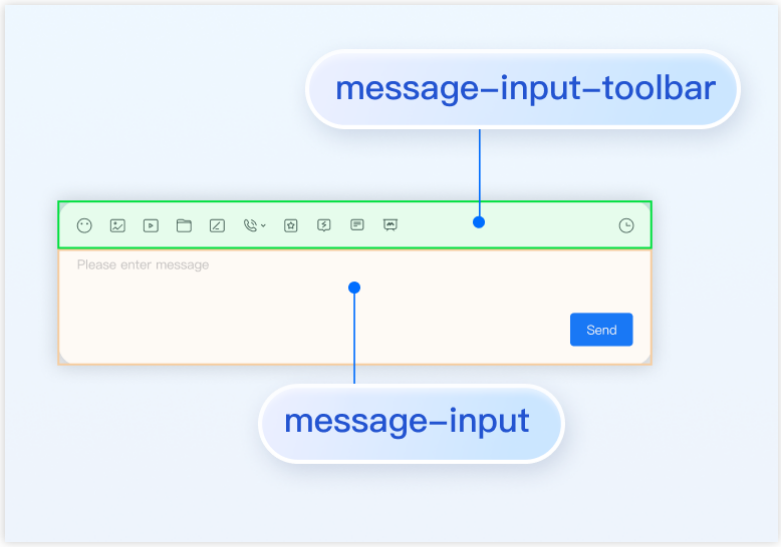
示例代码如下：



```
.message-tip {
    margin: 0 auto;
    color: #999999;// message tip font color
    font-size: 14px;// message tip font size
    background: red;// message tip background color
}
```

设置输入区域 InputView

输入区域包含文字输入、表情输入、图片发送、视频发送、文件发送、评价发送、常用语发送等功能。

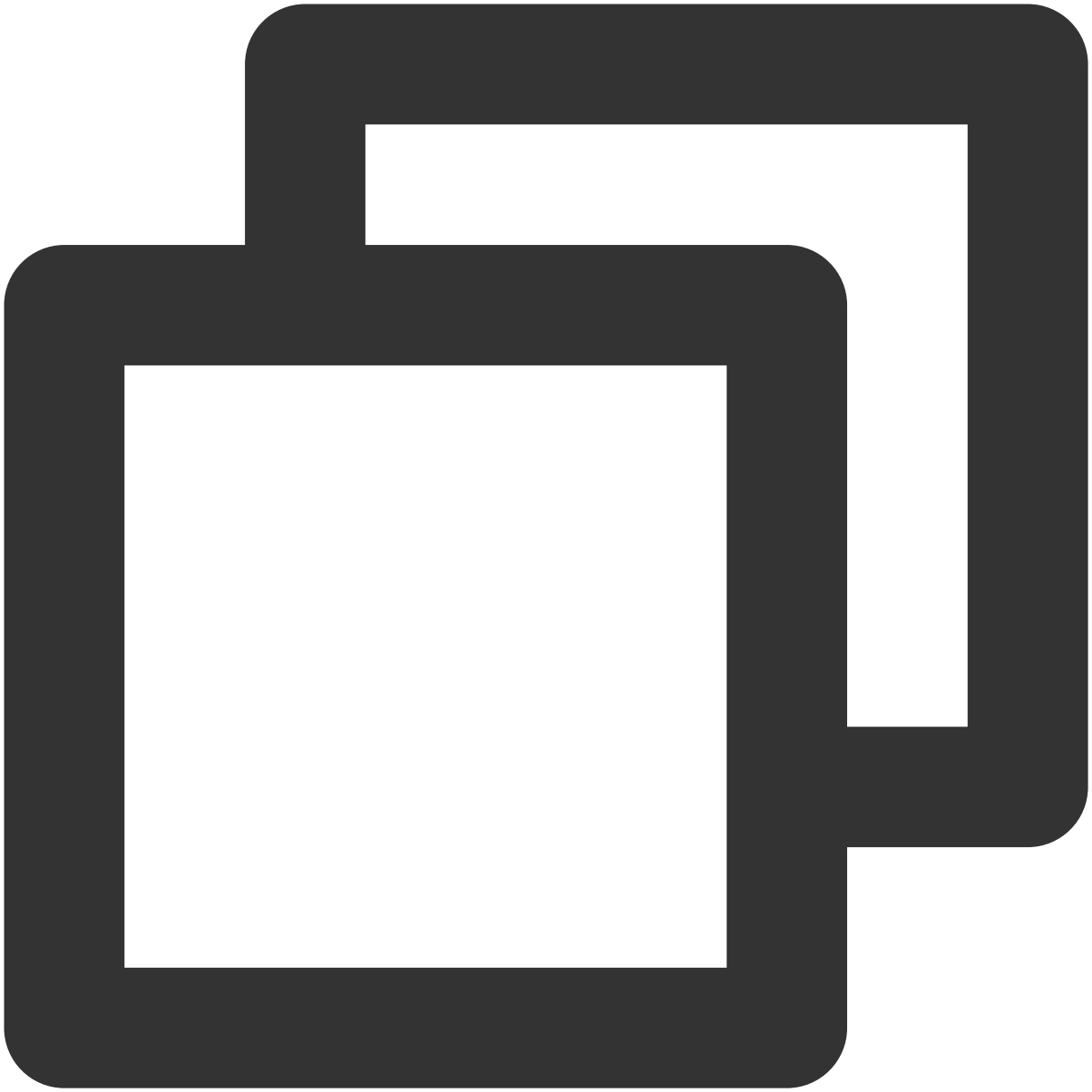


隐藏不需要的功能

您可以自定义隐藏功能模块的发送图片、发送文件以及发送评价等功能。

输入区功能模块在 `src/TUIKit/components/TUIChat/message-input-toolbar/index.vue` 文件中进行注册，您可以在该文件中注释或删除您不需要的功能。

例如：不需要使用常用语功能，注释常用语 `<Words>` 组件，示例代码如下：



```
<div>
 <div>
   <!-- Emoji Picker -->
   <EmojiPicker v-if="!isUniFrameWork"></EmojiPicker>
   <!-- Taking photos, only available on uniapp -->
   <ImageUpload v-if="isUniFrameWork" imageSourceType="camera"></ImageUpload>
   <!-- Image Upload -->
   <ImageUpload imageSourceType="album"></ImageUpload>
   <!-- File Upload -->
   <FileUpload v-if="!isUniFrameWork"></FileUpload>
   <!-- Video Upload -->
```

```
      <VideoUpload videoSourceType="album"></VideoUpload>
      <!-- Taking videos, only available on uniapp -->
      <VideoUpload v-if="isUniFrameWork" videoSourceType="camera"></VideoUpload>
      <!-- Evaluate -->
      <Evaluate></Evaluate>
      <!-- Commonly Used Phrases -->
      <!-- <Words></Words> -->
    </div>
  </div>
```

交流与反馈

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

Flutter

最近更新时间：2023-05-29 15:12:23

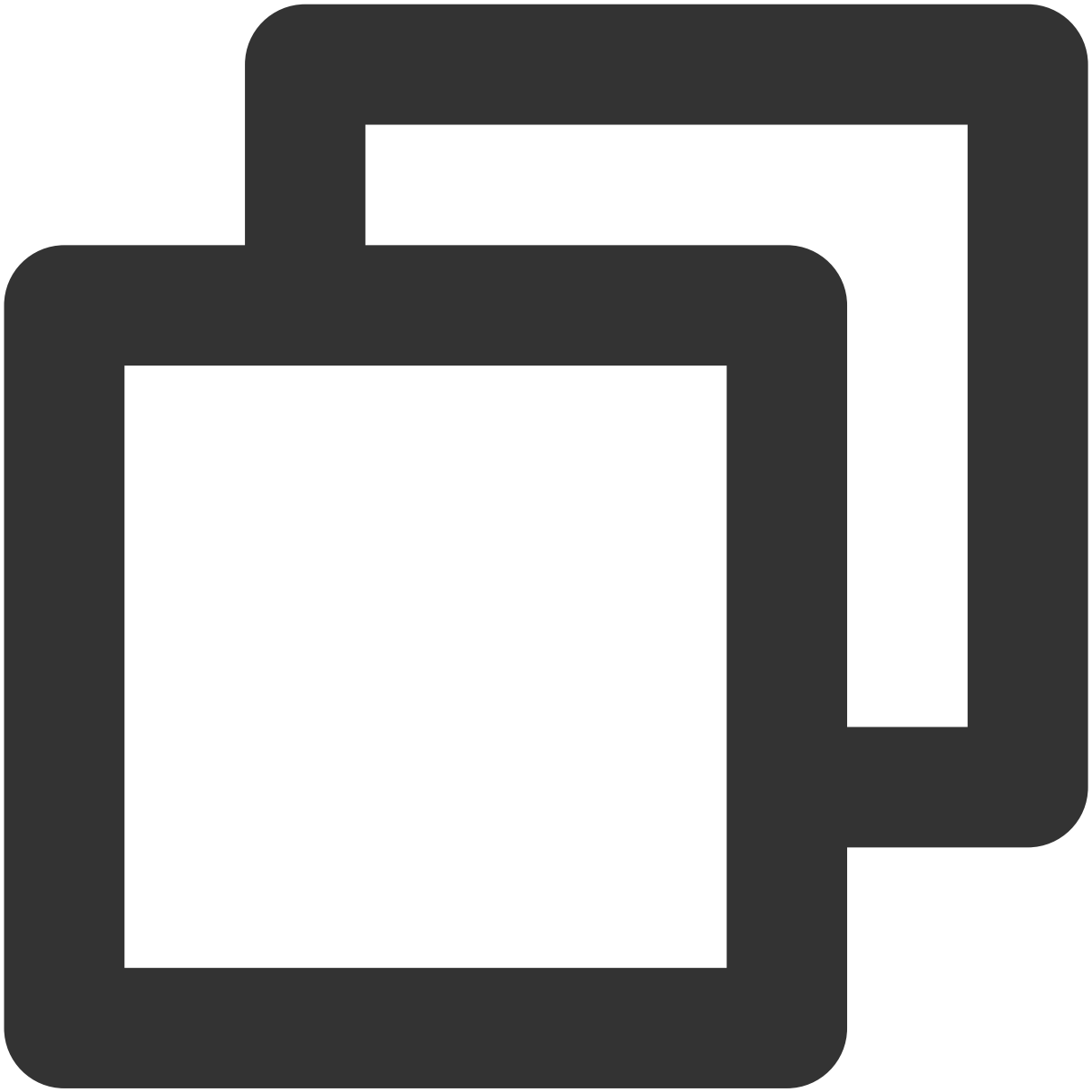
功能描述

TUIKit 从 v1.1.0 版本开始，主题风格颜色能力得到大幅度完善。
TUIKit默认提供颜色配置，您可以直接使用，无需任何配置。
但同时，您也可以很方便的自定义，TUIKit界面中，各处众多颜色配置。

自定义方式

步骤一：定义 TUIKit 颜色对象

在此对象中，您可以定义TUIKit界面中，各处的颜色配置。
请直接实例化一个 `TUITheme()` 对象。并修改里面的各个参数，以覆盖默认值，使用自定义颜色。
该对象构造函数，可配置的颜色有如下：



```
// 应用主色
// Primary Color For The App
final Color? primaryColor;

// 应用次色
// Secondary Color For The App
final Color? secondaryColor;

// 提示颜色，用于次级操作或提示
// Info Color, Used For Secondary Action Or Info
final Color? infoColor;
```

```
// 浅背景颜色，比主背景颜色浅，用于填充缝隙或阴影
// Weak Background Color, Lighter Than Main Background, Used For Marginal Space O
final Color? weakBackgroundColor;

// 宽屏幕：浅白背景颜色，比浅背景颜色浅
// Weak Background Color, Lighter Than Main Background, Used For Marginal Space O
final Color? wideBackgroundColor;

// 浅分割线颜色，用于分割线或边框
// Weak Divider Color, Used For Divider Or Border
final Color? weakDividerColor;

// 浅字色
// Weak Text Color
final Color? weakTextColor;

// 深字色
// Dark Text Color
final Color? darkTextColor;

// 浅主色，用于AppBar或Panels
// Light Primary Color, Used For AppBar Or Several Panels
final Color? lightPrimaryColor;

// 字色
// TextColor
final Color? textColor;

// 警示色，用于危险操作
// Caution Color, Used For Warning Actions
final Color? cautionColor;

// 群主标识色
// Group Owner Identification Color
final Color? ownerColor;

// 群管理员标识色
// Group Admin Identification Color
final Color? adminColor;

// 白色
// white
final Color? white;

// 黑色
// black
```

```
  final Color? black;

  // 输入框填充色
  // input fill color
  final Color? inputFillColor;

  // 灰色文本
  // grey text color
  final Color? textgrey;

  /// 新版本颜色从这里开始
  ///
  /// Appbar 背景颜色
  final Color? appbarBgColor;

  /// Appbar 文字颜色
  final Color? appbarTextColor;

  /// 消息列表多选面板背景颜色
  final Color? selectPanelBgColor;

  /// 消息列表多选面板文字及icon颜色
  final Color? selectPanelTextIconColor;

  /// 会话列表背景颜色
  final Color? conversationItemBgColor;

  /// 会话列表边框颜色
  final Color? conversationItemBorderColor;

  /// 会话列表选中背景颜色
  final Color? conversationItemActiveBgColor;

  /// 会话列表置顶背景颜色
  final Color? conversationItemPinedBgColor;

  /// 会话列表Title字体颜色
  final Color? conversationItemTitleTextColor;

  /// 会话列表LastMessage字体颜色
  final Color? conversationItemLastMessageTextColor;

  /// 会话列表Time字体颜色
  final Color? conversationItemTitmeTextColor;

  /// 会话列表用户在线状态背景色
  final Color? conversationItemOnlineStatusBgColor;
```

```
/// 会话列表用户离线状态背景色
final Color? conversationItemOfflineStatusBgColor;

/// 会话列表未读数背景颜色
final Color? conversationItemUnreadCountBgColor;

/// 会话列表未读数字体颜色
final Color? conversationItemUnreadCountTextColor;

/// 会话列表草稿字体颜色
final Color? conversationItemDraftTextColor;

/// 会话列表收到消息不提醒Icon颜色
final Color? conversationItemNoNotificationIconColor;

/// 会话列表侧滑按钮字体颜色
final Color? conversationItemSliderTextColor;

/// 会话列表侧滑按钮Clear背景颜色
final Color? conversationItemSliderClearBgColor;

/// 会话列表侧滑按钮Pin背景颜色
final Color? conversationItemSliderPinBgColor;

/// 会话列表侧滑按钮Delete背景颜色
final Color? conversationItemSliderDeleteBgColor;

/// 会话列表宽屏模式选中时背景颜色
final Color? conversationItemChooseBgColor;

/// 聊天页背景颜色
final Color? chatBgColor;

/// 聊天页背景颜色
final Color? chatTimeDividerTextColor;

/// 聊天页导航栏背景颜色
final Color? chatHeaderBgColor;

/// 聊天页导航栏Title字体颜色
final Color? chatHeaderTitleTextColor;

/// 聊天页导航栏Back字体颜色
final Color? chatHeaderBackTextColor;

/// 聊天页导航栏Action字体颜色
```

```
  final Color? chatHeaderActionTextColor;

  /// 聊天页历史消息列表字体颜色
  final Color? chatMessageItemTextColor;

  /// 聊天页历史消息列表来自自己时背景颜色
  final Color? chatMessageItemFromSelfBgColor;

  /// 聊天页历史消息列表来自非自己时背景颜色
  final Color? chatMessageItemFromOthersBgColor;

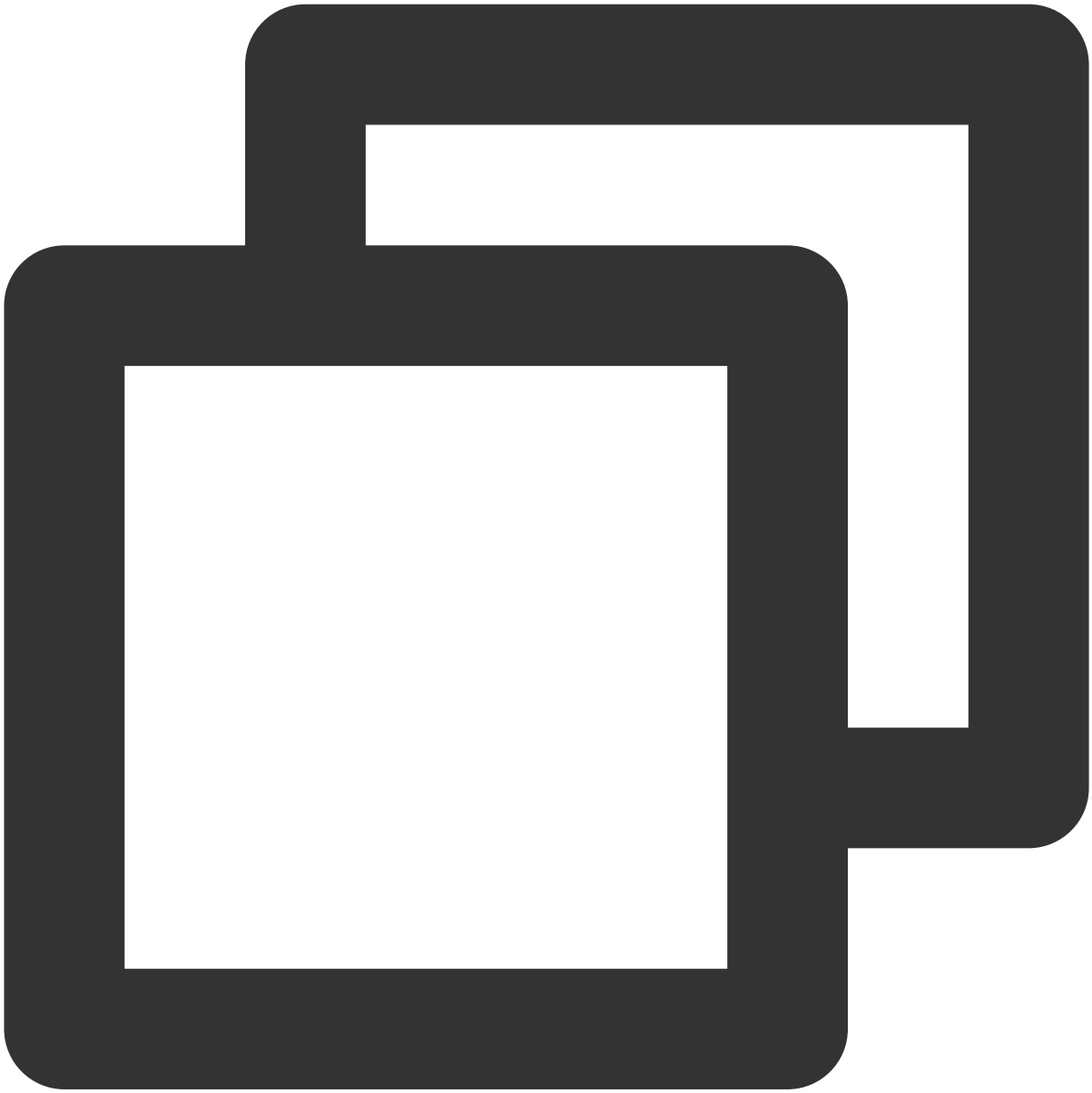
  /// 聊天页历史消息列表已读状态字体颜色
  final Color? chatMessageItemUnreadStatusTextColor;

  /// 聊天页历史消息列表小舌头背景颜色
  final Color? chatMessageTongueBgColor;

  /// 聊天页历史消息列表小舌头字体颜色
  final Color? chatMessageTongueTextColor;
```

步骤二：启用配置

调用 TUIKit 提供的 `setTheme` 方法，传入上一步定义的颜色对象 `TUITheme()` 即可。
该方法可随时调用，动态修改。



```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.setTheme(theme: TUITheme());
```

联系我们

如果您在接入使用过程中有任何疑问，请通过如下方式联系我们。

Telegram Group

WhatsApp Group

版权所有：腾讯云计算（北京）有限责任公司

实现本地搜索

Android

最近更新时间：2024-05-20 15:09:57

TUIKit 中的 TUISearch 实现了本地搜索，支持搜索本地存储的聊天记录、联系人、群聊等。搜索可以帮助用户从纷繁的信息中快速找到目标，也可作为运营工具，增加相关内容的引导，简洁高效。

注意：

“本地搜索”为 IM 旗舰版功能，购买旗舰版 后可使用，详见 价格说明。

功能展示

搜索接口的界面分为多个部分，第一部分是搜索好友，第二部分是搜索群组、群成员，第三部分是搜索消息且按照会话分组。

您可通过 下载安装应用 即刻体验。

接入指引

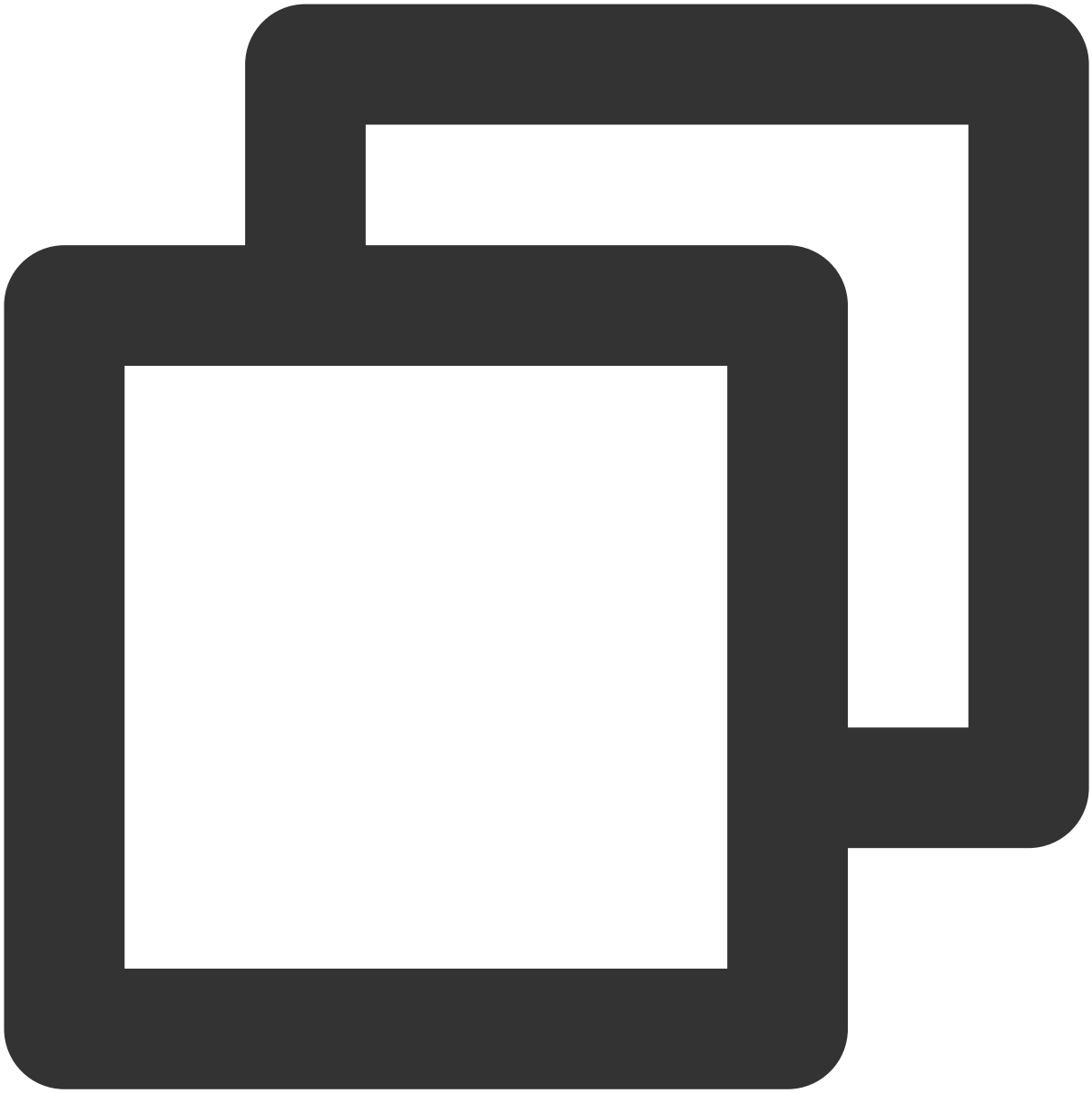
以下步骤将向您演示如何接入 TUISearch 组件。

购买套餐包

请单击前往 购买旗舰版。

集成 TUISearch

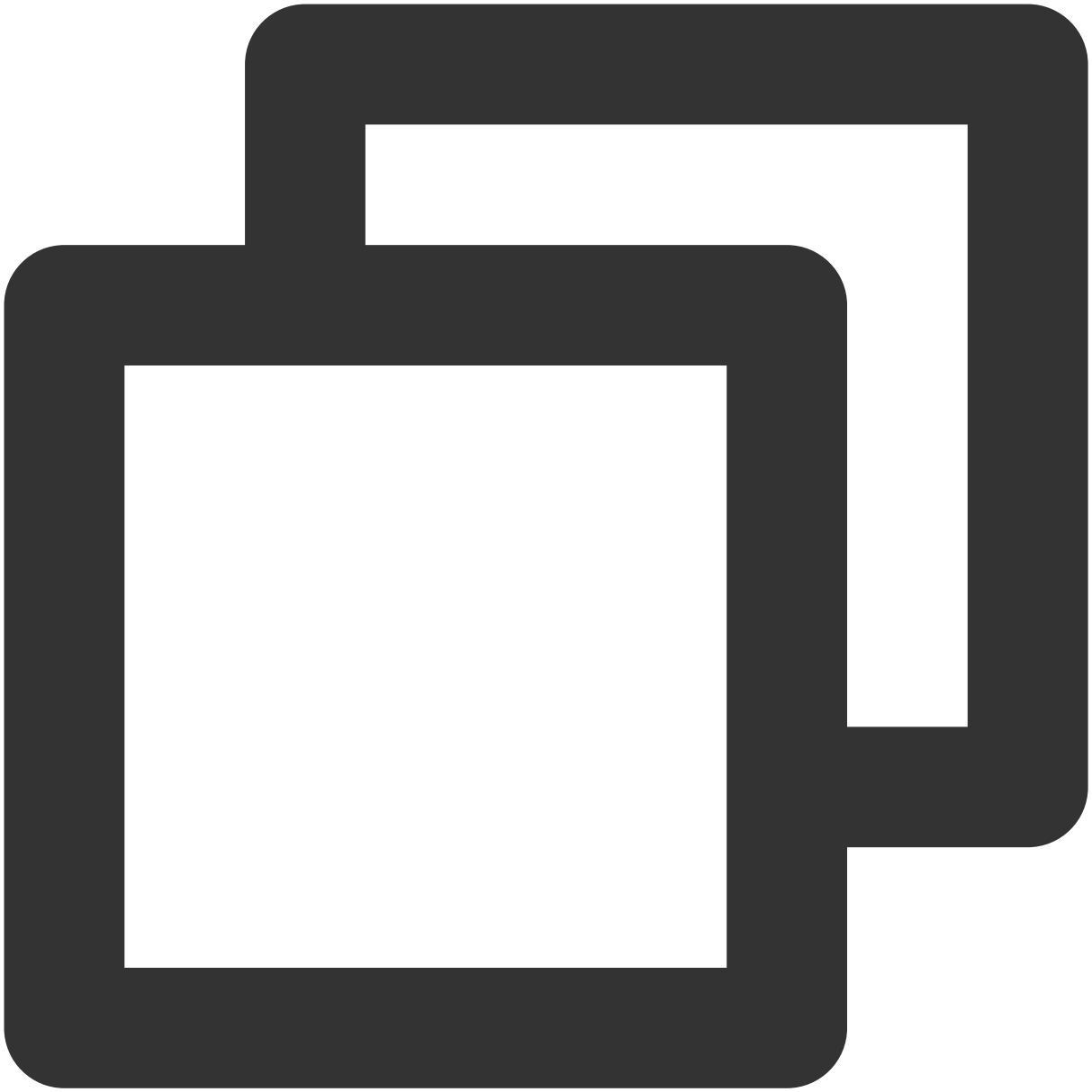
在 `APP` 的 `build.gradle` 文件中添加对 `tuisearch` 的依赖：



```
api project(':tuisearch')
```

登录 TUIKit

您需要调用 `TUICore` 的 `TUILogin` 登录 TUIKit。登录接口内部会默认初始化，不需要额外调用初始化。

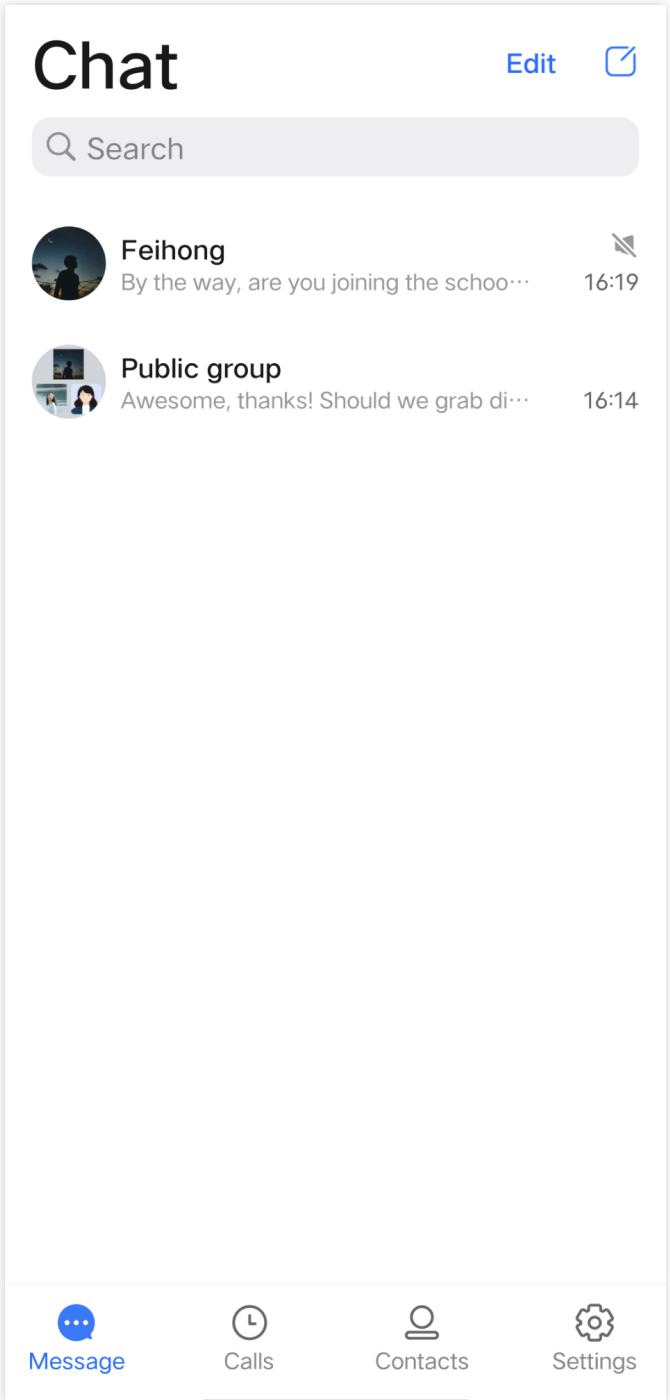


```
TUILogin.login(this, SDKAPPID, userID, userSig, new TUICallback() {
    @Override
    public void onError(final int code, final String desc) {
        // Login fails.
    }

    @Override
    public void onSuccess() {
        // Login succeeded
    }
});
```

启动搜索界面

1. 如果您集成了 TUIConversation 和 TUISearch 组件，此时不需要额外处理，searchBar 默认展示在会话列表的上方。如图所示：



2. 如果您仅集成 TUISearch，此时需要添加自己的搜索视图，然后点击启动 SearchMainMinimalistActivity （经典版 UI 为 SearchMainActivity）即可。

常见问题

1、如何搜索自定义消息

您需要使用接口 createCustomMessage (byte[] data, String description, byte[] extension) 来创建并发送自定义消息，把需要搜索的文本放到 `description` 参数中。

如果您使用接口 createCustomMessage (byte[] data) 创建自定义消息，本地保存的是二进制数据流，无法被搜索到。

如果您配置了离线推送功能，参数 `description` 设置后，自定义消息也会有离线推送且通知栏展示该参数内容。

如果不需要离线推送可以用发消息接口 sendMessage 的参数 V2TIMOfflinePushInfo 中的 disablePush 来控制。

如果推送的通知栏内容不想展示为被搜索的文本，可以用参数 V2TIMOfflinePushInfo 中的 setDesc 来另外设置推送内容。

2、如何搜索富媒体消息

富媒体消息包含文件、图片、语音、视频消息。

对于文件消息，界面通常显示文件名，因此创建时可以设置 `fileName` 参数，作为被搜索的内容，如果 `fileName` 不设置则会从 `filePath` 提取文件名，并且都会保存到本地和服务器。

而对于图片、语音、视频消息，界面通常显示缩略图或时长，可以指定消息类型做分类搜索，但不能通过关键字搜索。

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2024-05-20 17:06:39

TUIKit 中的 TUISearch 实现了本地搜索，支持搜索本地存储的聊天记录、联系人、群聊等。搜索可以帮助用户从纷繁的信息中快速找到目标，也可作为运营工具，增加相关内容的引导，简洁高效。

注意：

“本地搜索”为 IM 旗舰版功能，购买旗舰版 后可使用，详情请参见 价格说明。

功能展示

搜索接口的界面分为多个部分，第一部分是搜索好友，第二部分是搜索群组、群成员，第三部分是搜索消息且按照会话分组。

您可通过 下载安装应用 即刻体验。

接入指引

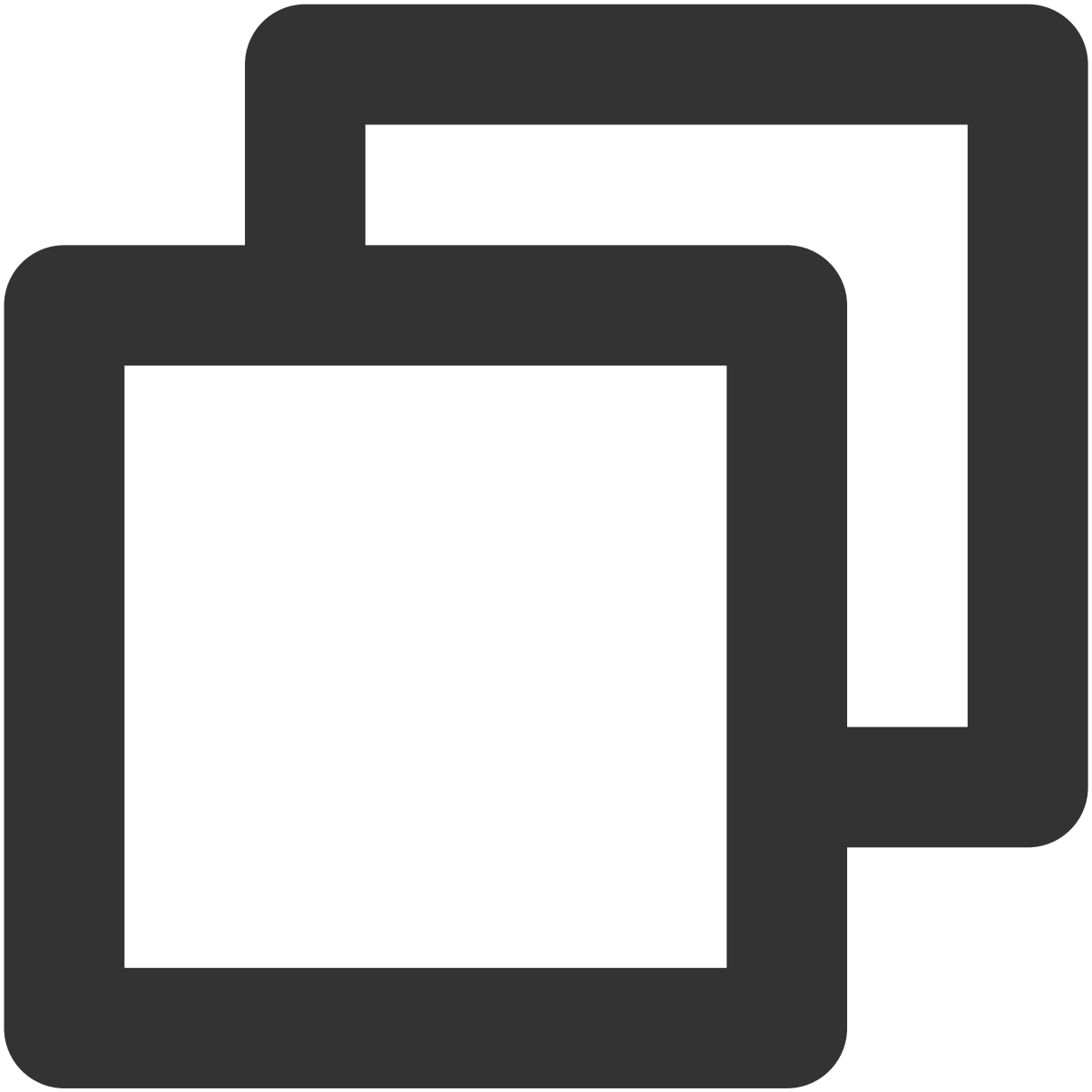
以下步骤将向您演示如何接入 TUISearch 组件。

购买套餐包

请单击前往 购买旗舰版。

集成 TUISearch

在 Podfile 文件中添加以下内容：

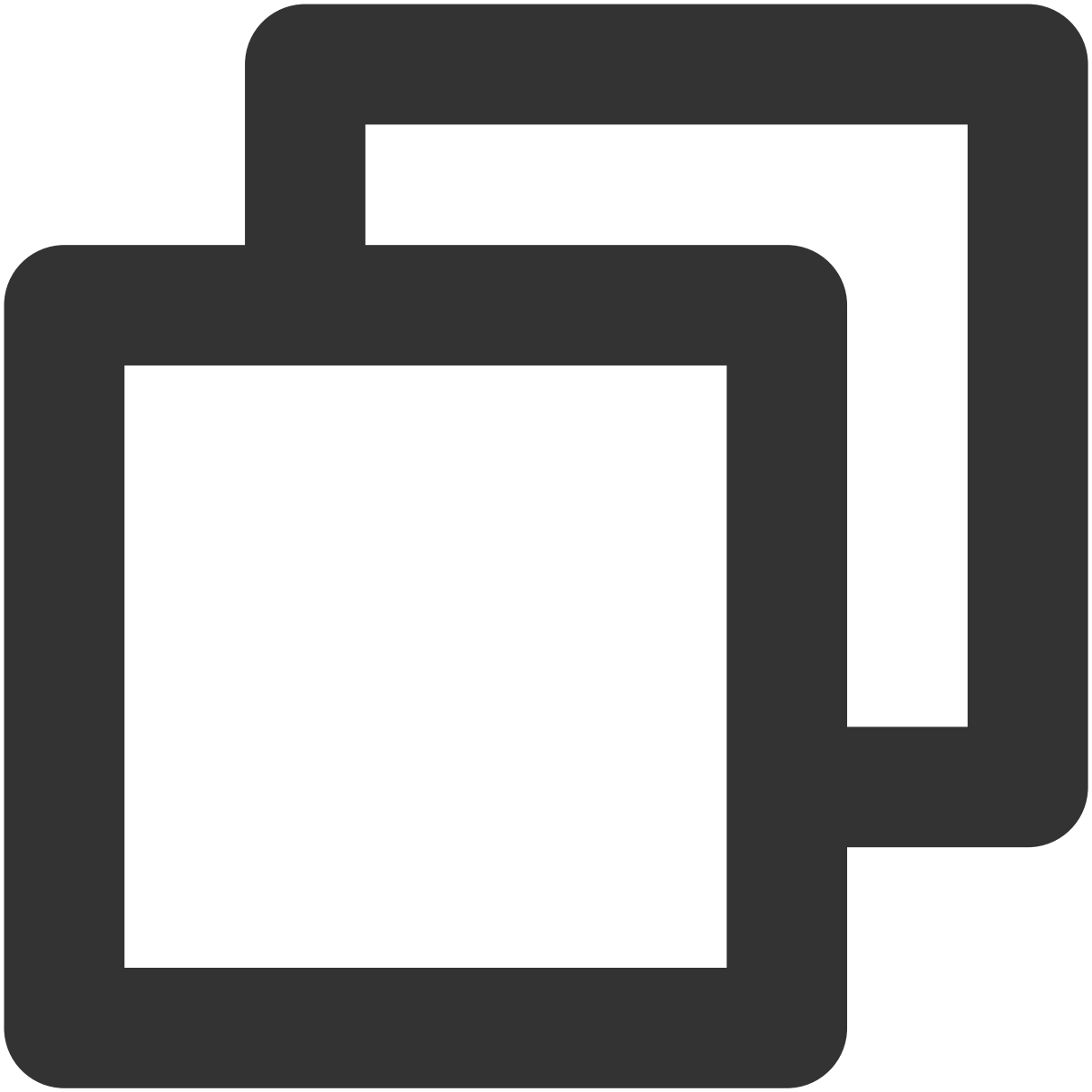


```
pod 'TUISearch'
```

添加后执行 `pod instal` 。

登录 TUIKit

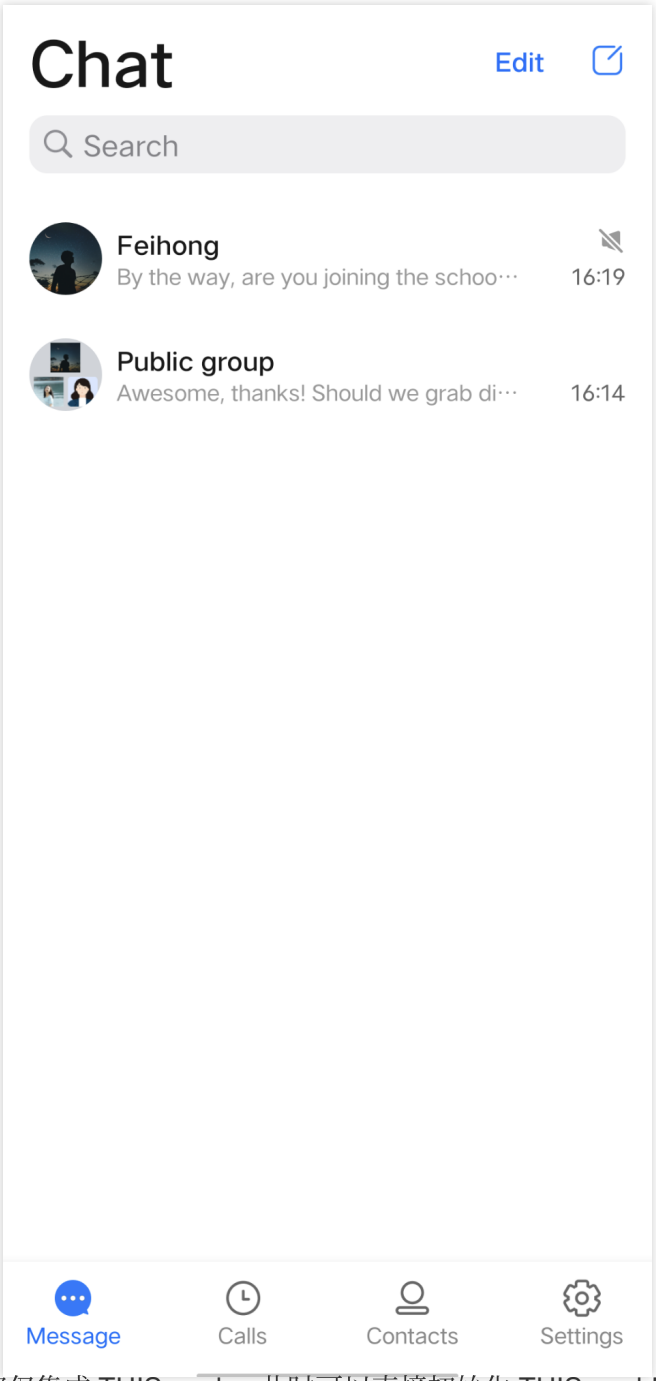
您需要调用 `TUICore` 的 `TUILogin` 登录 TUIKit。登录接口内部会默认初始化，不需要额外调用初始化。



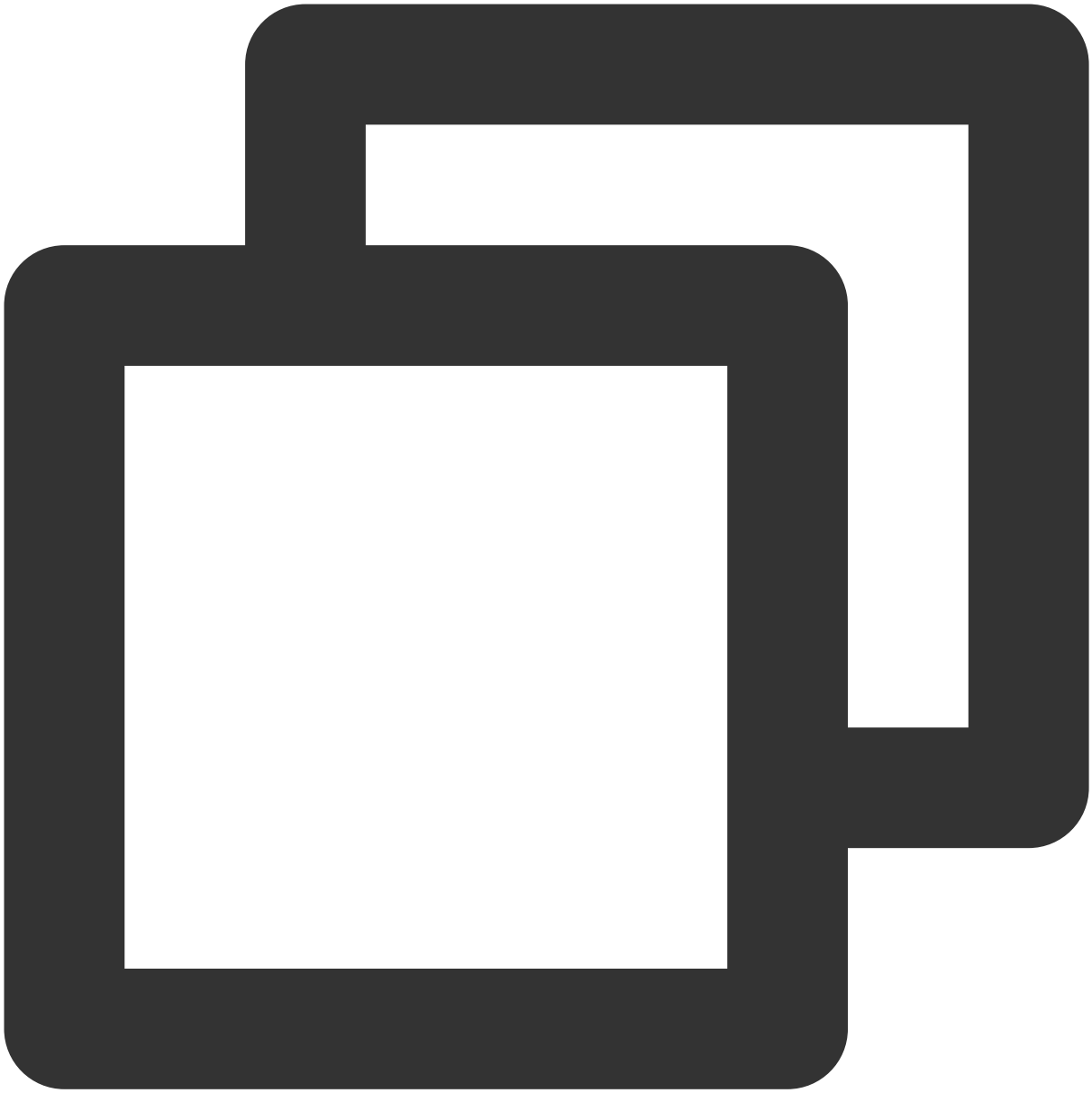
```
[TUILogin login:SDKAPPID
         userID:userID
        userSig:userSig
           succ:^{
        // Login succeeded
} fail:^(int code, NSString *msg) {
        // Login fails.
}];
```

启动搜索界面

如果您集成了 TUIConversation 和 TUISearch 组件，此时不需要额外处理，searchBar 默认展示在会话列表的上方。如图所示：



如果您仅集成 TUISearch，此时可以直接初始化 TUISearchBar 并将其添加到自己的视图上即可。TUISearchBar 内部封装了搜索的 UI 逻辑和界面，添加 TUISearchBar 后，点击即可触发搜索。示例代码如下：



```
// init
TUISearchBar *searchBar = [[TUISearchBar alloc] init];
// self.containerView indicates your own view
[self.containerView addSubview:searchBar];
```

常见问题

如何搜索自定义消息？

您需要使用接口 createCustomMessage:desc:extension 来创建并发送自定义消息，把需要搜索的文本放到 `desc` 参数中。
如果您使用接口 createCustomMessage 创建自定义消息，本地保存的是二进制数据流，无法被搜索到。
如果您配置了离线推送功能，设置参数 `desc` 后，自定义消息也会有离线推送且通知栏展示该参数内容。
如果不需要离线推送可以用发消息接口 sendMessage 的参数 V2TIMOfflinePushInfo 中的 disablePush 来控制。
如果不希望推送的内容展示为被搜索的文本，可以用参数 V2TIMOfflinePushInfo 中的 desc 来另外设置推送内容。

如何搜索富媒体消息？

富媒体消息包含文件、图片、语音、视频消息。
对于文件消息，界面通常显示文件名，因此创建时可以设置 `fileName` 参数，作为被搜索的内容，如果 `fileName` 不设置则会从 `filePath` 提取文件名，并且都会保存到本地和服务器。
而对于图片、语音、视频消息，界面通常显示缩略图或时长，可以指定消息类型做分类搜索，但不能通过关键字搜索。

版权所有：腾讯云计算（北京）有限责任公司

Web & H5 & Uniapp（Vue）

最近更新时间：2024-07-10 16:26:41

功能体验

含 UI 集成

快速集成消息云端搜索

Web&H5 Vue2&Vue3

Uniapp Vue2&Vue3

步骤1: 集成TUIKit

@tencentcloud/chat-uikit-vue ≥ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

注意：

每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。

步骤3：搜索您的第一条消息

在完成 Vue2 & Vue3 TUIKit 快速集成指引 - 步骤6: 发送您的第一条消息后，搜索您刚才发送的消息。

步骤1: 集成TUIKit

@tencentcloud/chat-uikit-uniapp ≥ 2.0.6，如未集成，请请务必先根据 uniapp TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

注意：

每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。

步骤3：搜索你的第一条消息

在完成 Uniapp TUIKit 快速集成指引 - 步骤6: 发送您的第一条消息后，搜索您刚才发送的消息。

独立引入消息云端搜索

说明：

以上 快速集成消息云端搜索 中已默认包含消息云端搜索全部功能，无需重复引入。

如果您想独立引入 <TUISearch> 消息云端搜索，请参考以下教程。

Web&H5 Vue2&Vue3

Uniapp Vue2&Vue3

前提条件

@tencentcloud/chat-uikit-vue ≥ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引进行集成。

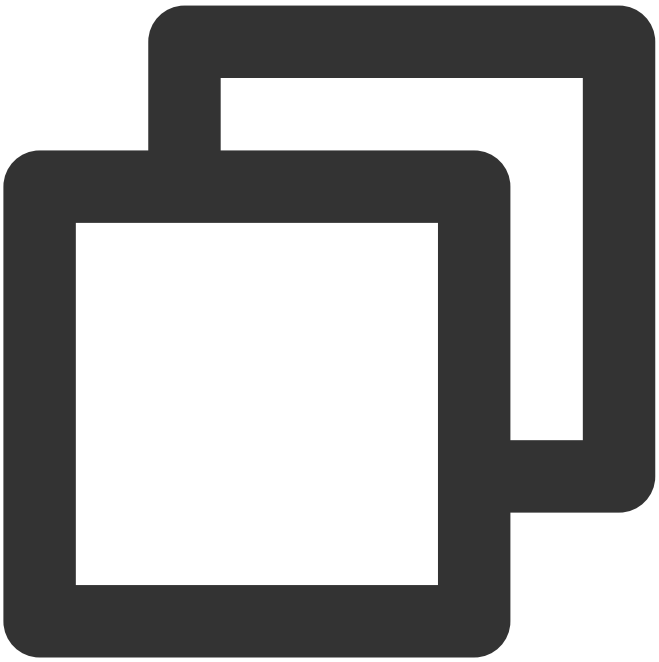
引入 <TUISearch>

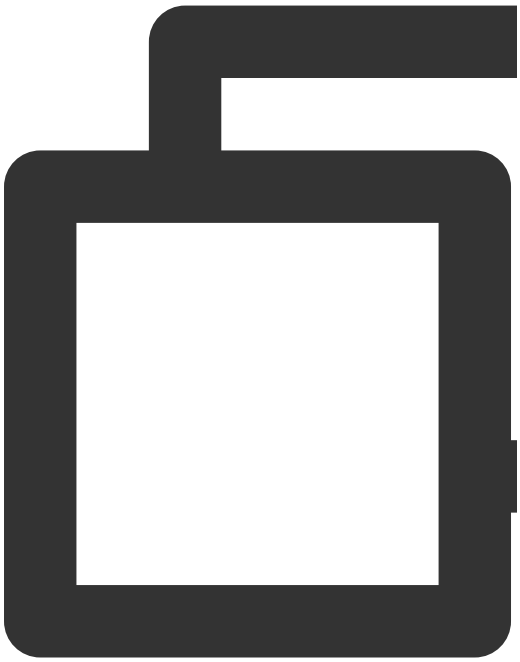
在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
searchType	String	global：全局搜索（default）
		conversation：会话内搜索

<TUISearch> 效果展示

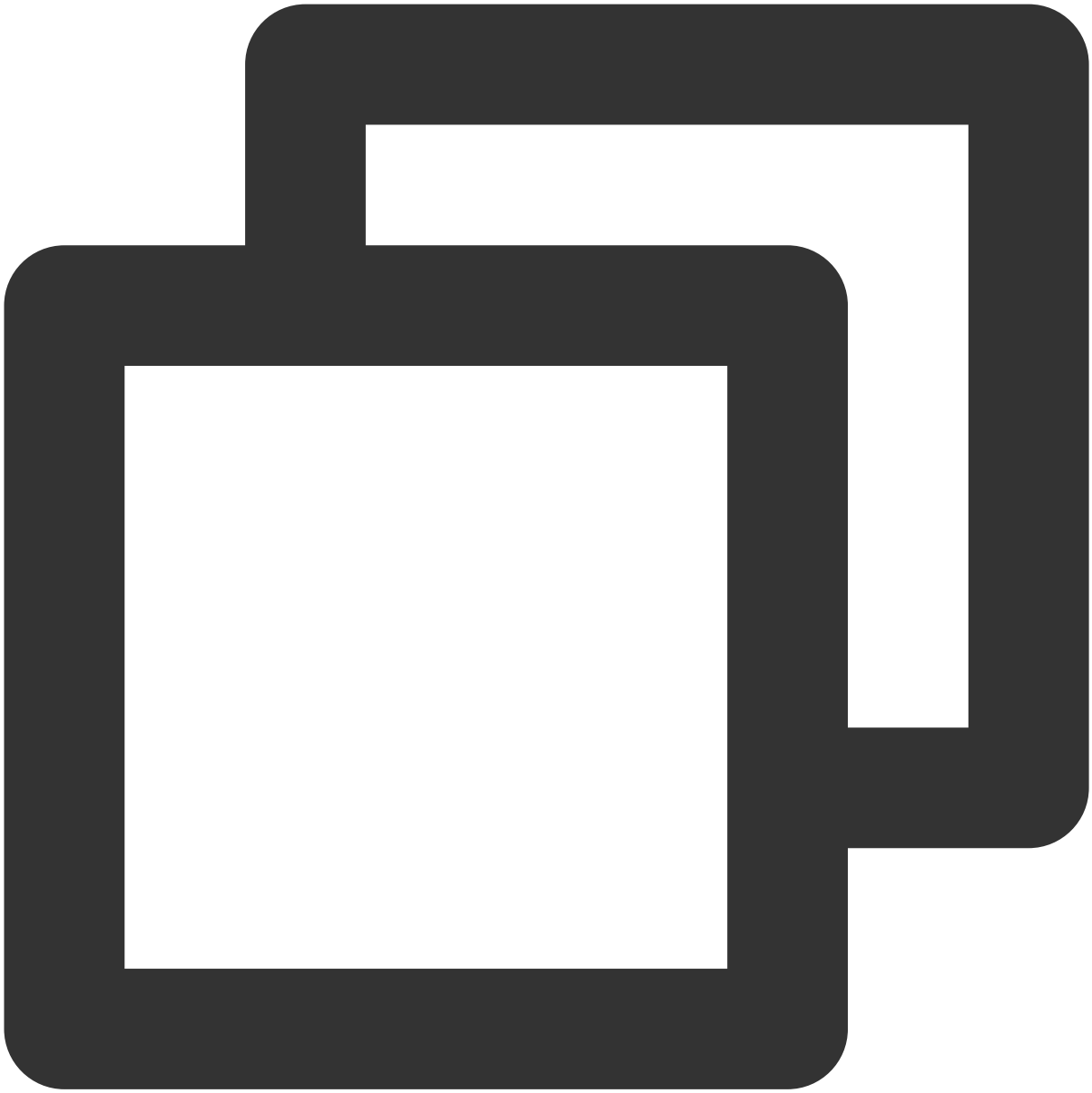




```
<TUISearch searchType="global" />
```

```
<TUISearch searchType="conversat
```

使用 TUISearch



```
import { TUISearch } from "@tencentcloud/chat-uikit-vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>`，如您不按照默认集成方式使用，可在 `TUIKit/index.vue` 中，注释掉 `<TUISearch>` 即可。

Uniapp TUISearch 支持两种方式引入：组件方式引入与界面方式引入。

前提条件

@tencentcloud/chat-uikit-uniapp ≥ 2.0.6，如未集成，请请务必先根据 uniapp TUIKit 快速集成指引 进行集成。

组件方式引入

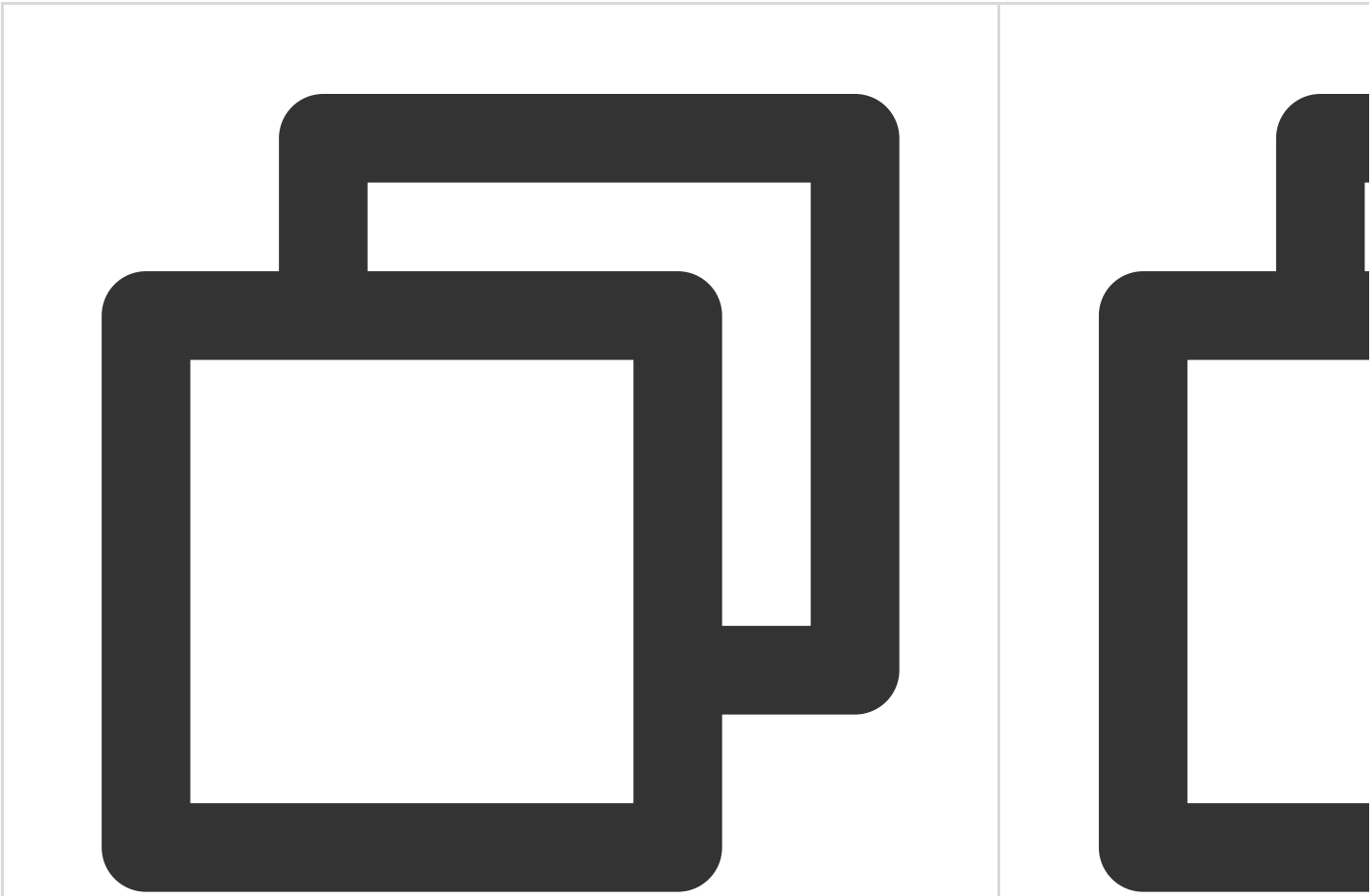
界面方式引入

在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
searchType	String	global：全局搜索
		conversation：会话内搜索（default）

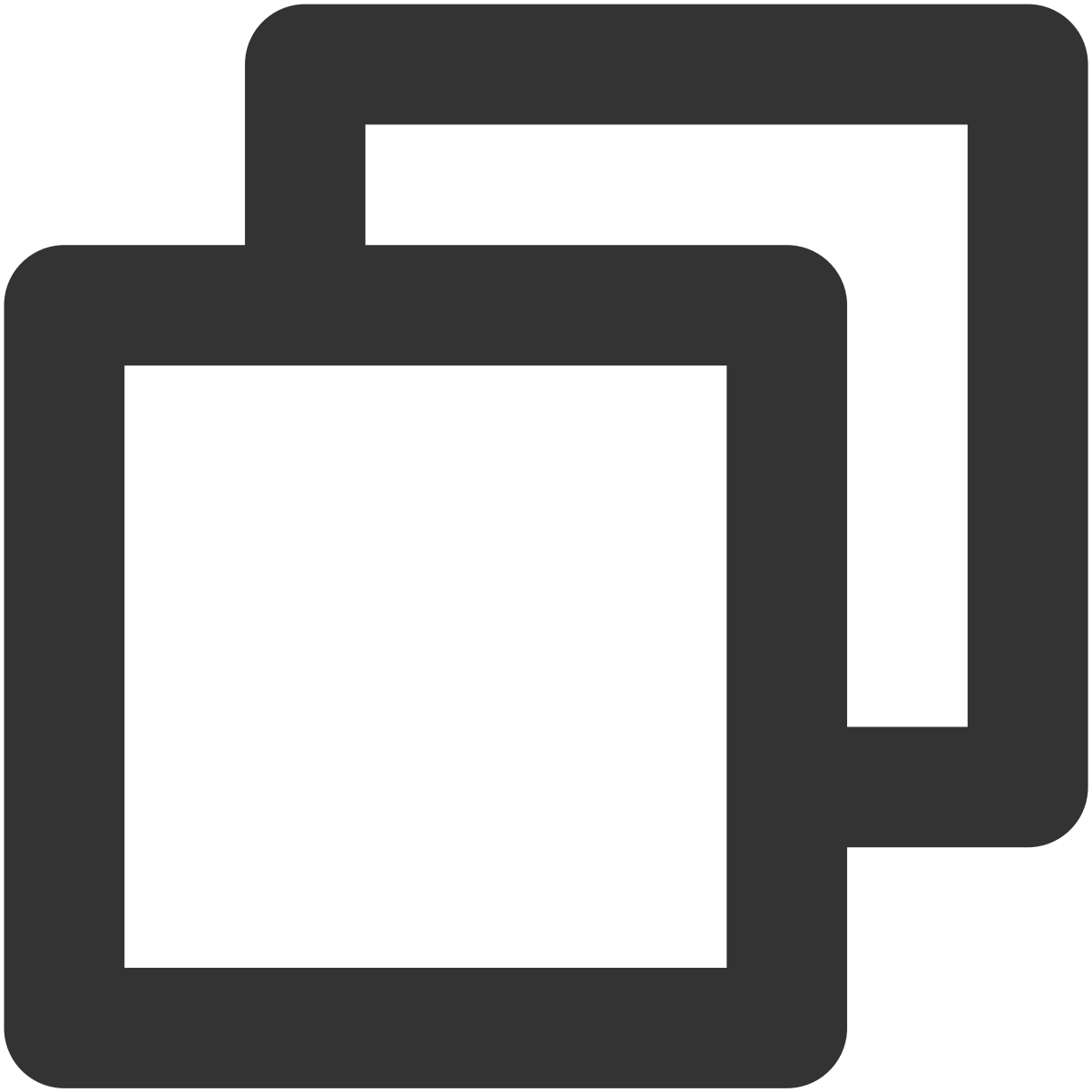
<TUISearch> 效果展示



```
<TUISearch searchType="global" />
```

```
<TUISearch searchTyp
```

使用 TUISearch

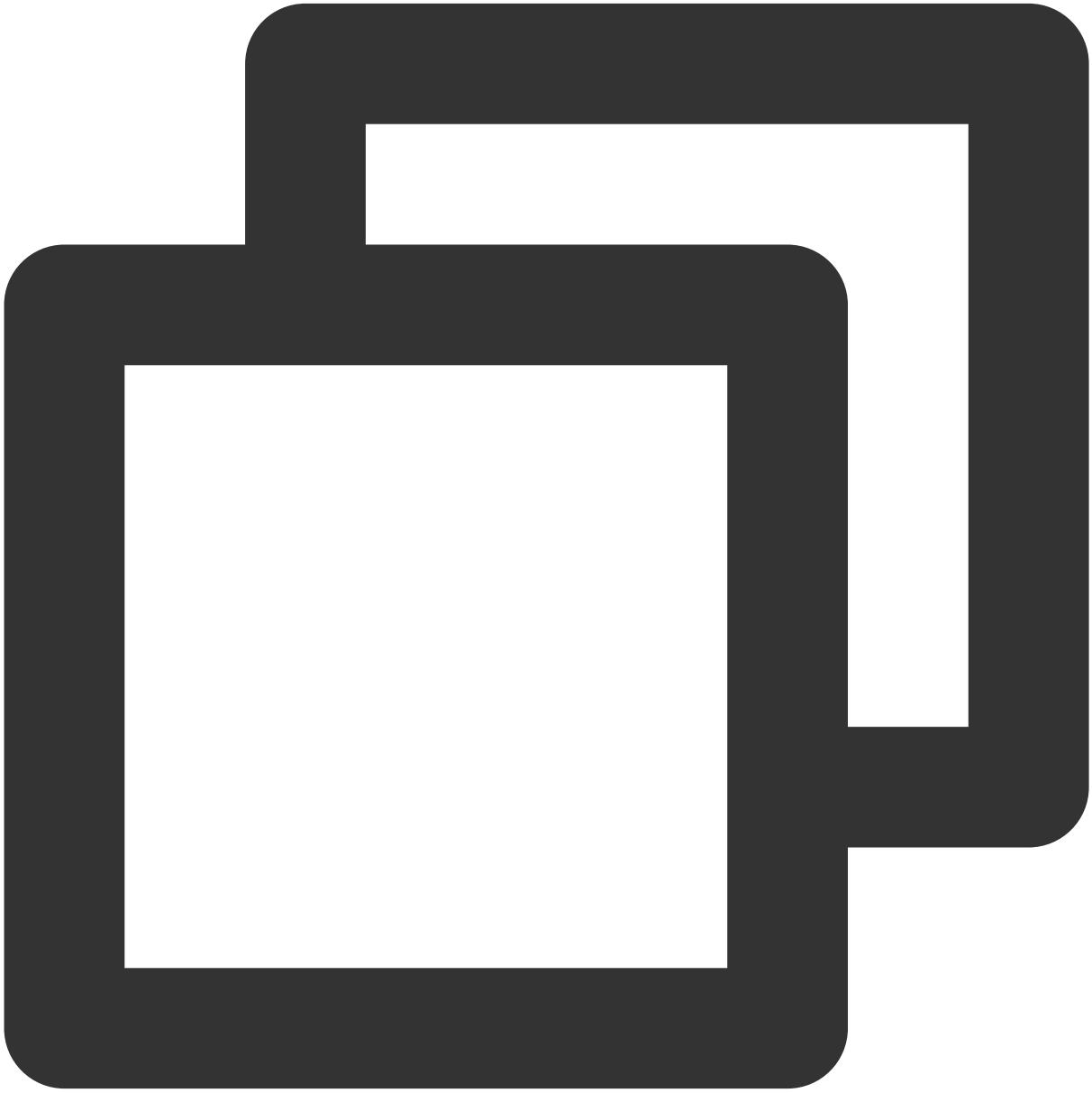


```
// 以下路径仅为示例路径，请根据您项目自身路径进行调整
import { TUISearch } from "/TUIKit/components/TUISearch/index.vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>` ， 如您不按照默认集成方式使用，可在 `TUIKit/components/TUIConversation/index.vue` 中，注释掉 `<TUISearch>` 即可。

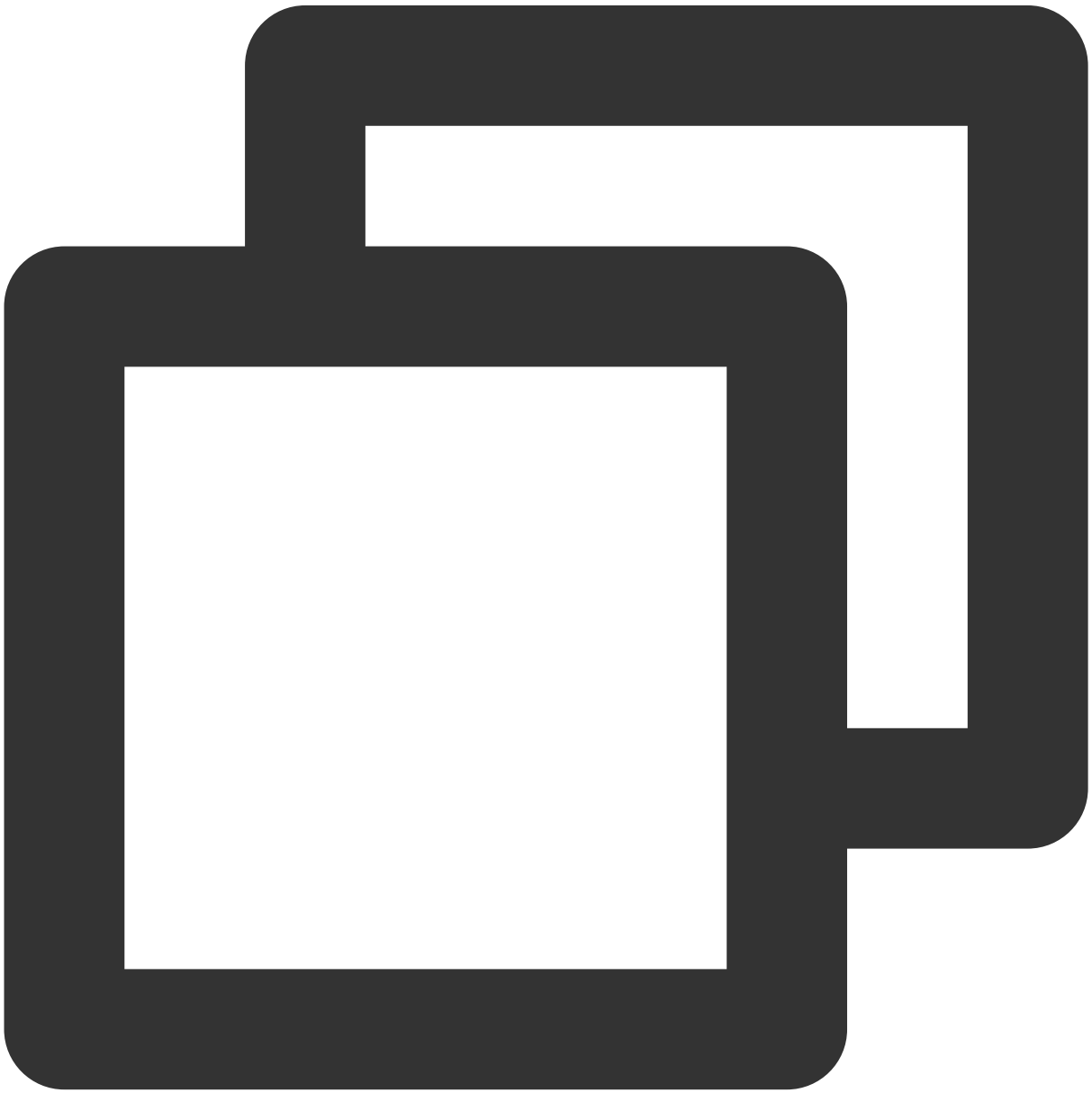
在 pages.json 新增 TUISearch 页面



```
{
  "pages": [
    ...,
    {
      "path": "TUIKit/components/TUISearch/index",
      "style": {
```

```
          "navigationBarTitleText": "聊天记录"
        }
      }
    ],
    ...
}
```

跳转到 TUISearch 界面



```
uni.navigateTo({
    url: "/TUIKit/components/TUISearch/index",
```

```
});
```

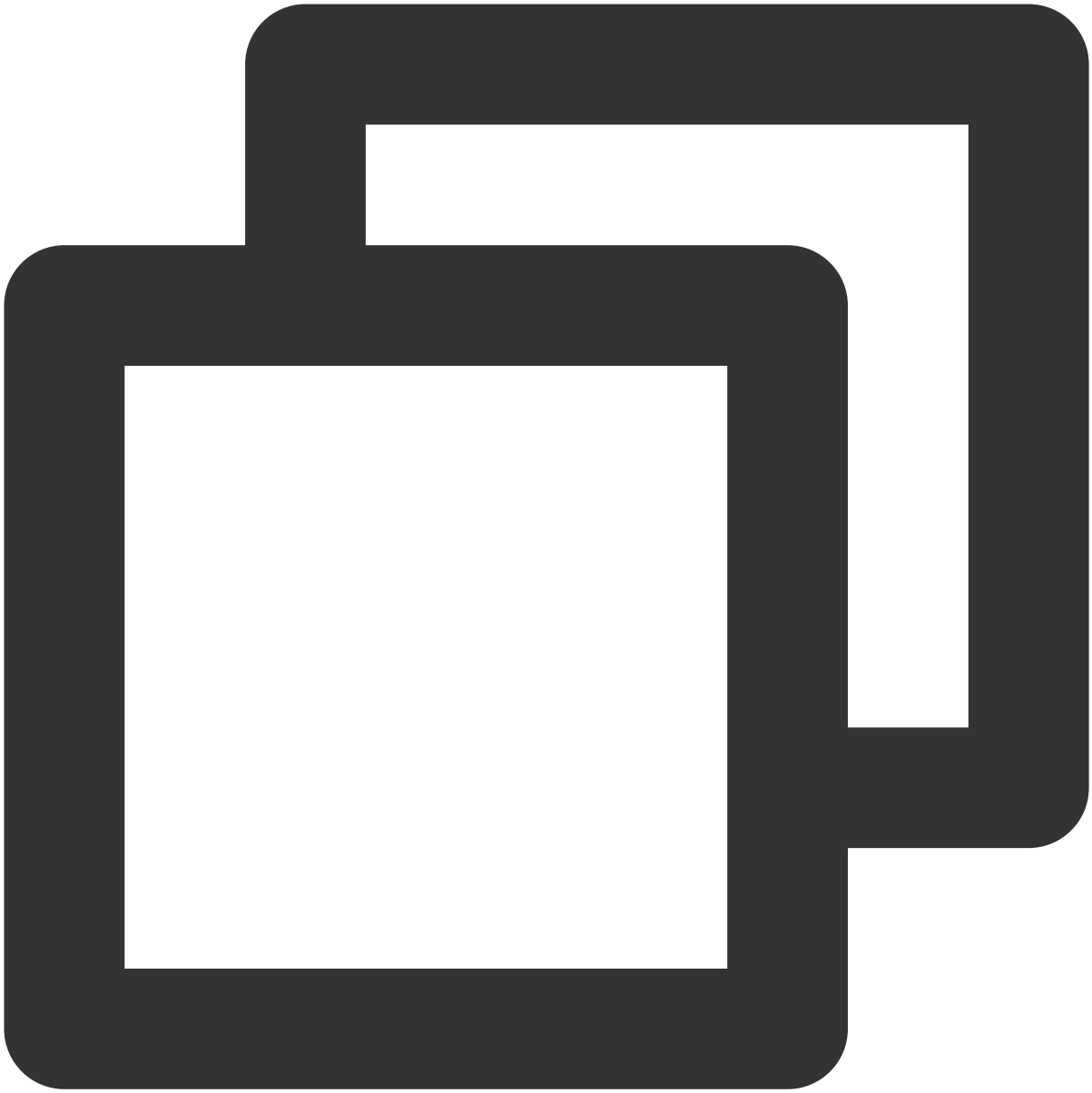
高级指引

新增搜索消息类型

原“全局搜索”消息类型列表	新增后“全局搜索”消息类型列表

目录位置： `src/TUIKit/components/TUISearch/search-type-list.ts`

`searchMessageTypeList` 中包含了所有“搜索消息类型” Tab 定义，如需新增 `searchMessageTypeList` 未定义的搜索消息类型，请按照以下结构在 `searchMessageTypeList` 中进行新增：



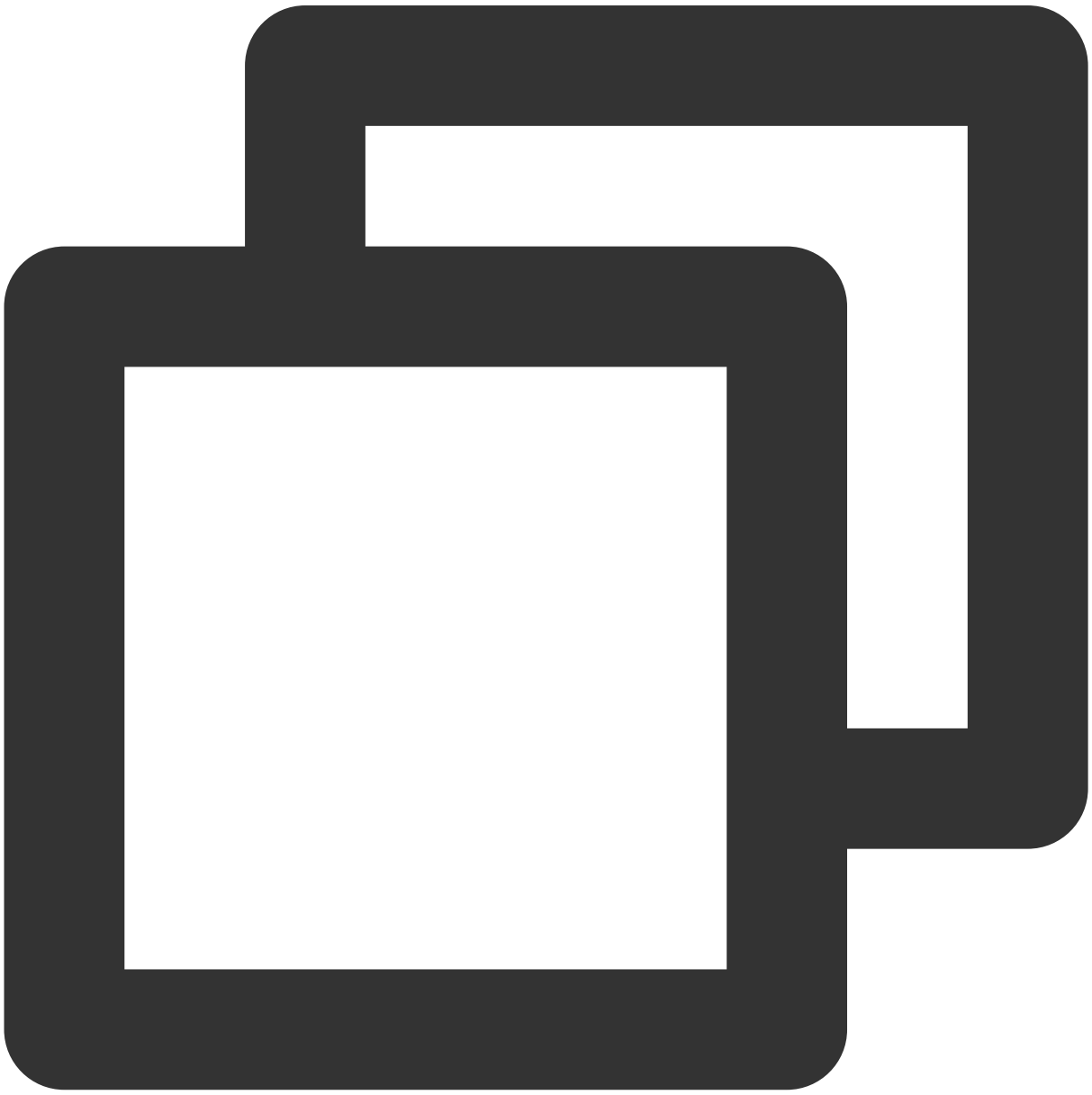
```
[keyName: string]: {
    key: string;// 消息搜索类型 key，请保持唯一性
    label: string;// 消息搜索类型渲染 label
    value: Array<string>;// 消息搜索类型实际搜索列表
 };

 // 例如，定义搜索自定义类型消息
export const searchMessageTypeList = {
    ...
    customMessage: {
      key: "customMessage",// 消息搜索类型 key，请保持唯一性
```

```
        label: "自定义",// 消息搜索类型渲染 label
        value: [TUIChatEngine.TYPES.MSG_CUSTOM],// 消息搜索类型实际搜索列表
      }
  };
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` 以及 `src/TUIKit/locales/en/TUISearch.ts` 增加相应的国际化词条进行翻译。

如需将已定义的 `searchMessageTypeList` 中某类型增加到**全局搜索类型列表**或者**会话内搜索类型列表**，仅需将其 `key` 填入 `globalSearchTypeKeys` 或 `conversationSearchTypeKeys` 即可。



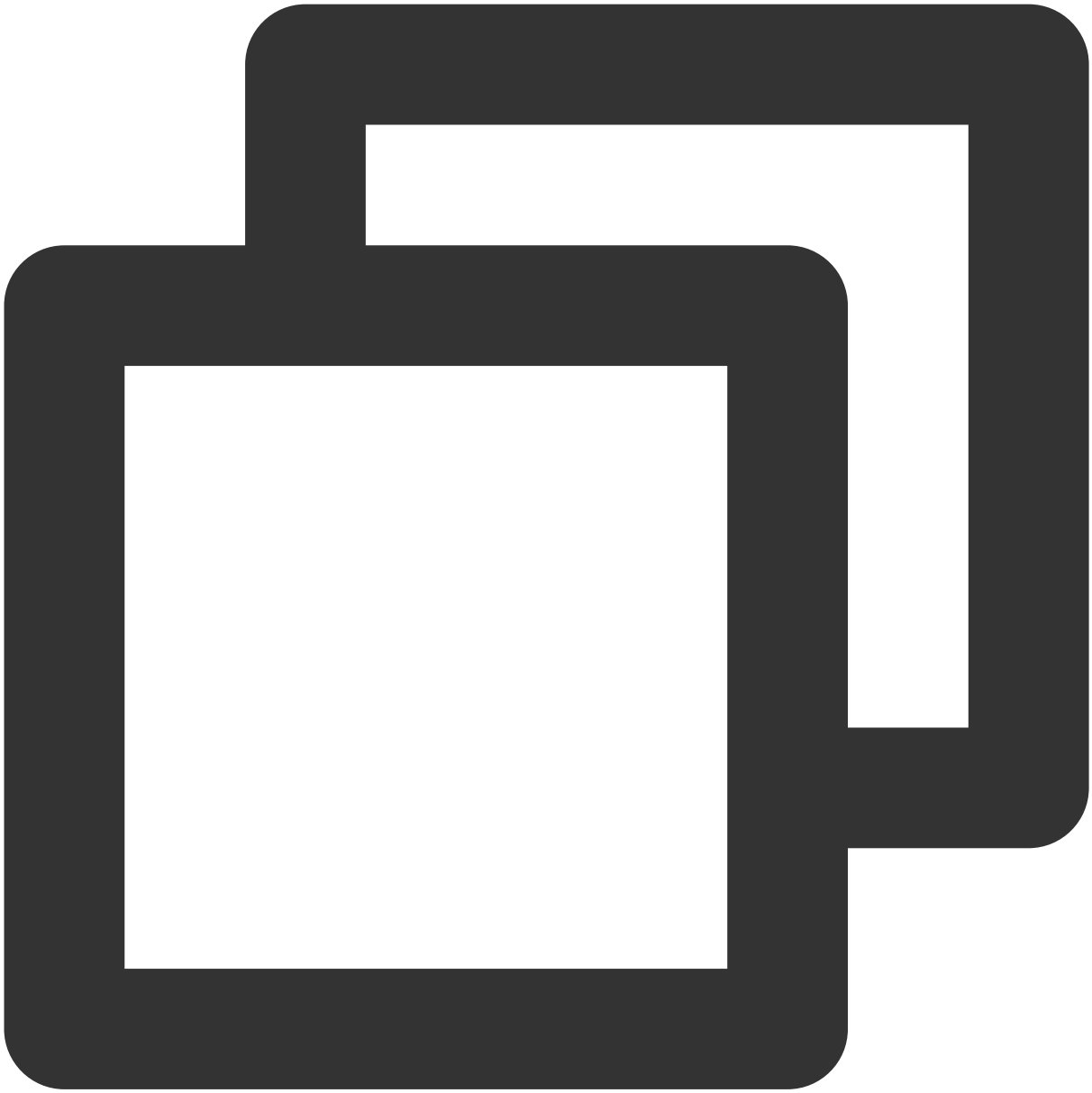
```
// 例如，将以上新增的 自定义消息 customMessage 应用到“全局搜索”消息类型列表
export const globalSearchTypeKeys = [..., "customMessage"];
// 例如，将以上新增的 自定义消息 customMessage 应用到“会话内搜索”消息类型列表
export const conversationSearchTypeKeys = [..., "customMessage"];
```

新增消息云端搜索时间范围

原“全局搜索”时间范围列表	新增后“全局搜索”时间范围列表

目录位置： `src/TUIKit/components/TUISearch/search-time-list.ts`

`searchMessageTimeList` 中包含了所有“搜索时间范围” Tab 定义，如需新增 `searchMessageTimeList` 未定义的搜索时间范围类型，请按照以下结构在 `searchMessageTimeList` 中进行新增：



```
[keyName: string]: {
    key: string;// 消息搜索时间范围 key，请保持唯一性
    label: string;// 消息搜索时间范围渲染 label
    value: {
      timePosition: number; // 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
      timePeriod: number; // 从 timePosition 向前搜索的时间范围
    };
};

 // 例如，定义搜索“近两天“时间范围
export const searchMessageTimeList = {
```

```
    ...
    twoDay: {
       key: "twoDay",// 消息搜索时间范围 key，请保持唯一性
       label: "近两天",// 消息搜索时间范围渲染 label
       value: {
         timePosition: 0,// 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
         timePeriod: 2 * oneDay,// 从 timePosition 向前搜索的时间范围
       },
   },
};
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` 以及 `src/TUIKit/locales/en/TUISearch.ts` 增加相应的国际化词条进行翻译。

Flutter

最近更新时间：2024-01-31 14:22:31

TUIKit 中实现了本地搜索，可支持搜索本地存储的聊天记录、联系人、群聊等。搜索可以帮助用户从纷繁的信息中快速找到目标，也可作为运营工具，增加相关内容的引导，简洁高效。

注意：

“本地搜索”为 IM 旗舰版功能，购买旗舰版 后可使用，详见 价格说明。

功能展示

组件名	组件功能
TIMUIKitSearch	全局搜索
TIMUIKitSearchMsgDetail	会话内搜索，单聊&群聊

全局搜索的界面分为三部分，第一部分是搜索好友，第二部分是搜索群组、群成员，第三部分是搜索消息且按照会话分组。

您可 下载 Demo 应用 即刻体验。

接入指引

以下步骤将向您演示如何接入 TUIKit 本地搜索组件。

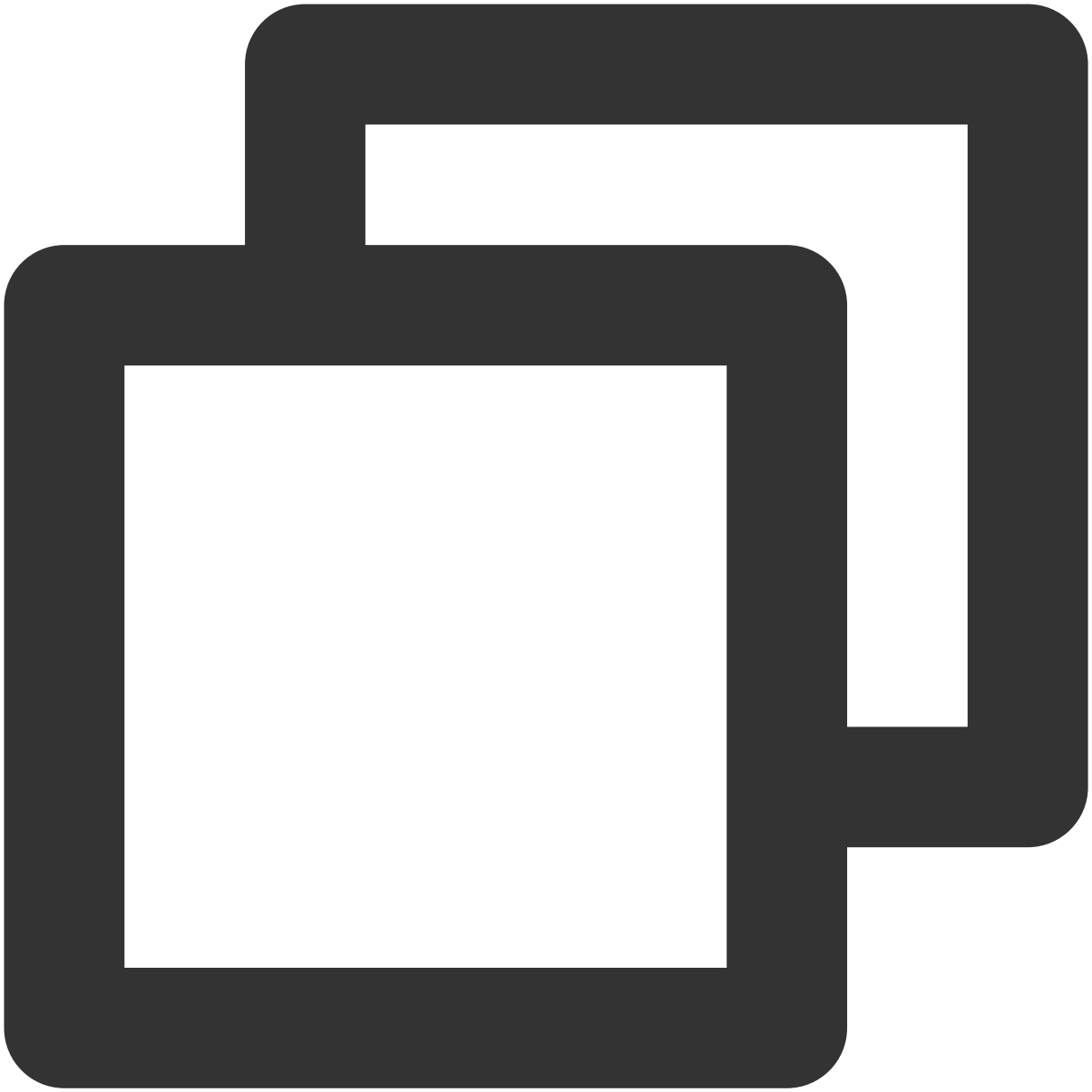
购买套餐包

请单击前往 购买旗舰版。

引入本地搜索

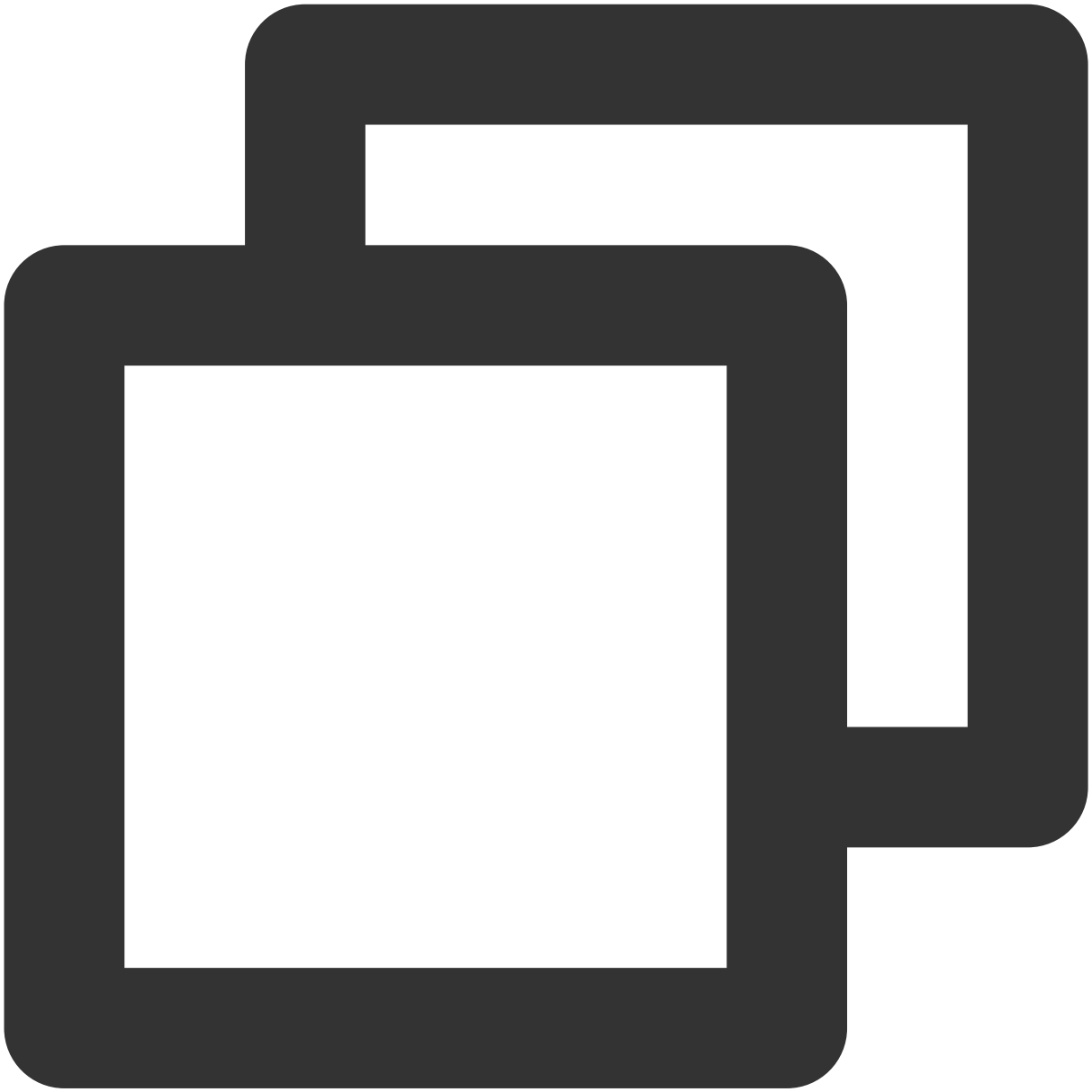
在 全局搜索 页面的文件中引入以下内容：

版权所有：腾讯云计算（北京）有限责任公司



```
/// 集成 TIMUIKitSearch 组件
import 'package:tencent_cloud_chat_uikit/ui/views/TIMUIKitSearch/tim_uikit_search.d
```

在 会话内搜索 页面的文件中引入以下内容：



```
/// 集成 TIMUIKitSearchMsgDetail 组件
import 'package:tencent_cloud_chat_uikit/ui/views/TIMUIKitSearch/tim_uikit_search_m
```

全局搜索 界面

TIMUIKitSearch 为全局搜索组件，搜索的结果为匹配搜索关键字的**联系人**、**群组**、**聊天记录**。
TIMUIKitSearch 的一般使用场景为在消息列表上方放置，点击进入全局搜索组件。
代码示例可见 此文档。

会话内搜索 界面

TIMUIKitSearchMsgDetail 为搜索聊天信息的组件，搜索的结果为匹配搜索关键字的**聊天记录**。
TIMUIKitSearchMsgDetail 的使用场景多样，例如：
在全局搜索点击**聊天记录**时进入；
在用户资料页面查询**聊天记录**时进入；
在群组资料页面查询**聊天记录**时进入。
代码示例可见 此文档。

常见问题

1、如何搜索富媒体消息

富媒体消息包含文件、图片、语音、视频消息。
对于文件消息，界面通常显示文件名，因此创建时可以设置 `fileName` 参数，作为被搜索的内容，如果 `fileName` 不设置则会从 `filePath` 提取文件名，并且都会保存到本地和服务器。
而对于图片、语音、视频消息，界面通常显示缩略图或时长，可以指定消息类型做分类搜索，但不能通过关键字搜索。

接入离线推送

Android

最近更新时间：2024-01-31 14:23:28

概述

即时通信 IM 的终端用户需要随时都能够得知最新的消息，而由于移动端设备的性能与电量有限，当 App 处于后台时，为了避免维持长连接而导致的过多资源消耗，即时通信 IM 推荐您使用各厂商提供的系统级推送通道来进行消息通知，系统级的推送通道相比第三方推送拥有更稳定的系统级长连接，可以做到随时接受推送消息，且资源消耗大幅降低。

注意：

在没有主动退出登录的情况下，应用退后台、手机锁屏、或者应用进程被用户主动杀掉三种场景下，如果想继续接收到 IM 消息提醒，可以接入即时通信 IM 离线推送。

如果应用主动调用 logout 退出登录，或者多端登录被踢下线，即使接入了 IM 离线推送，也收不到离线推送消息。

集成 TUIOfflinePush 跑通离线推送功能

集成 TUIOfflinePush 组件之前，请您先 注册应用到厂商推送平台，登录腾讯云账号进行 IM 控制台配置 和 配置离线推送跳转界面。之后按照如下步骤操作即可快速接入 IM 离线推送。

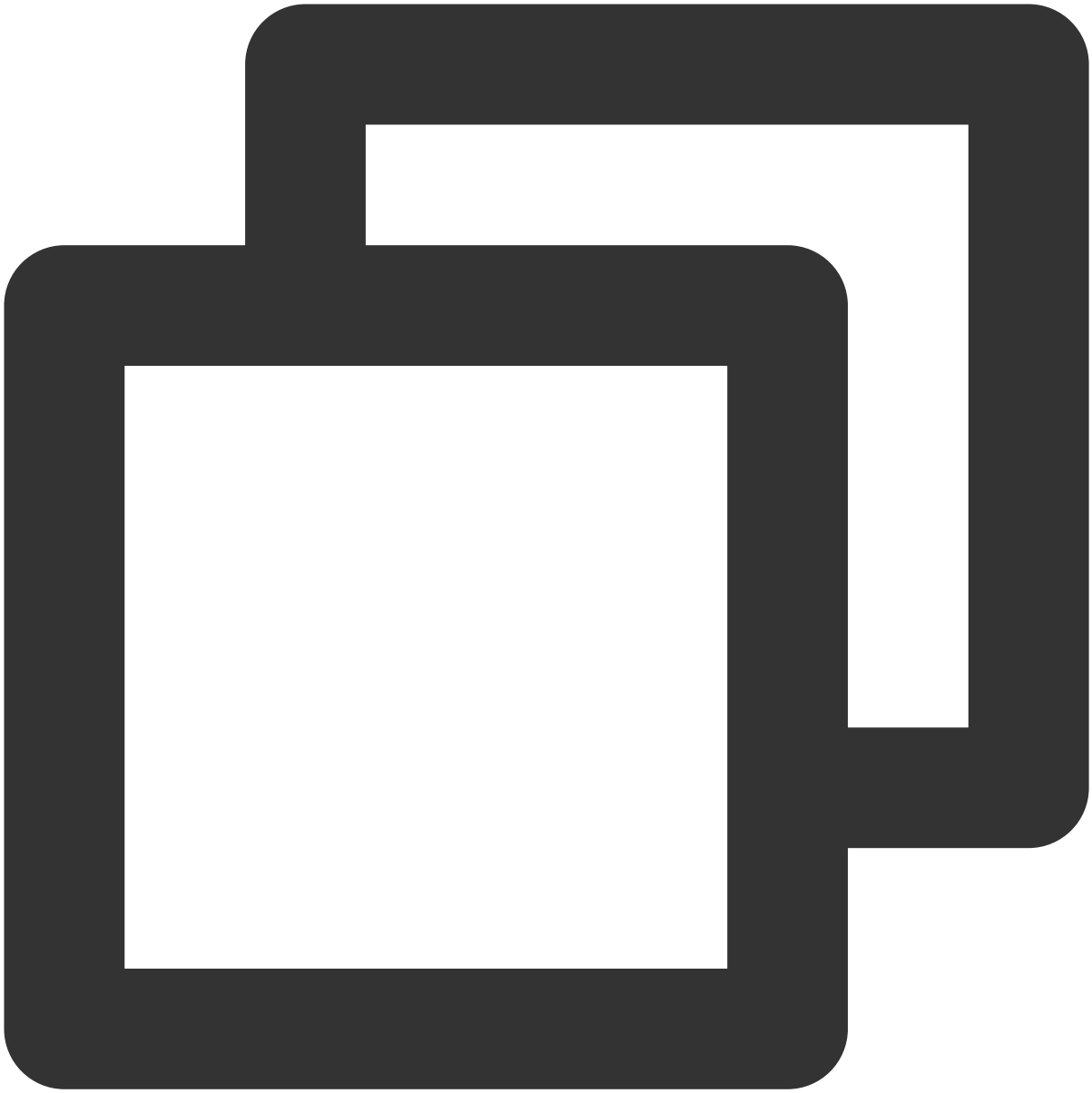
注意：

如果您想尽可能简单地接入 TUIOfflinePush 组件，您需要使用 TUICore 组件中的 TUILogin 提供的 login/logout 接口登录/登出，此时 TUIOfflinePush 组件会自动感知登录/登出事件。如果您不想使用 TUILogin 提供的接口，您需要在完成登录/登出操作后，手动调用 TUIOfflinePushManager 的接口 registerPush/unRegisterPush。

组件支持的厂商有：小米、华为、荣耀、OPPO、vivo、魅族和 Google FCM。

步骤1：集成 TUIOfflinePush 组件

版权所有：腾讯云计算（北京）有限责任公司



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

vivo 和荣耀适配

根据 vivo 和荣耀厂商接入指引，需要将 APPID 和 APPKEY 添加到清单文件中，否则会出现编译问题：

方法1

方法2

```android{...defaultConfig{...manifestPlaceholders=["VIVO_APPKEY":"您应用分配的证书APPKEY","VIVO_APPID":"您应用分配的证书APPID""HONOR_APPID":"您应用分配的证书APPID"]}}```
```

// vivo end
// honor begin

// honor end```

```

//

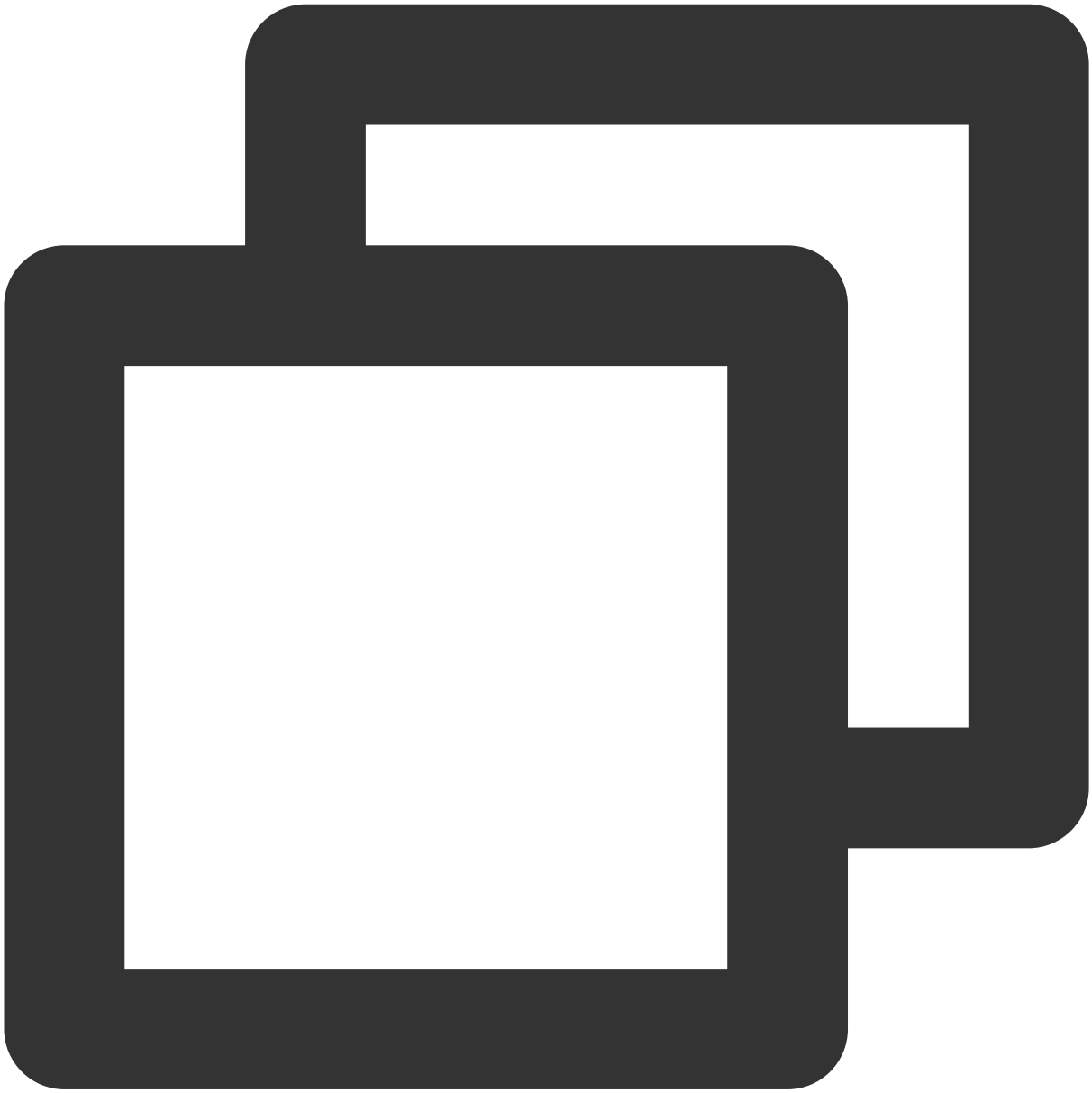
vivo

begin

**华为和 Google FCM 适配**

华为和 Google FCM 需要按照厂商方法，集成对应的 plugin 和 json 配置文件。

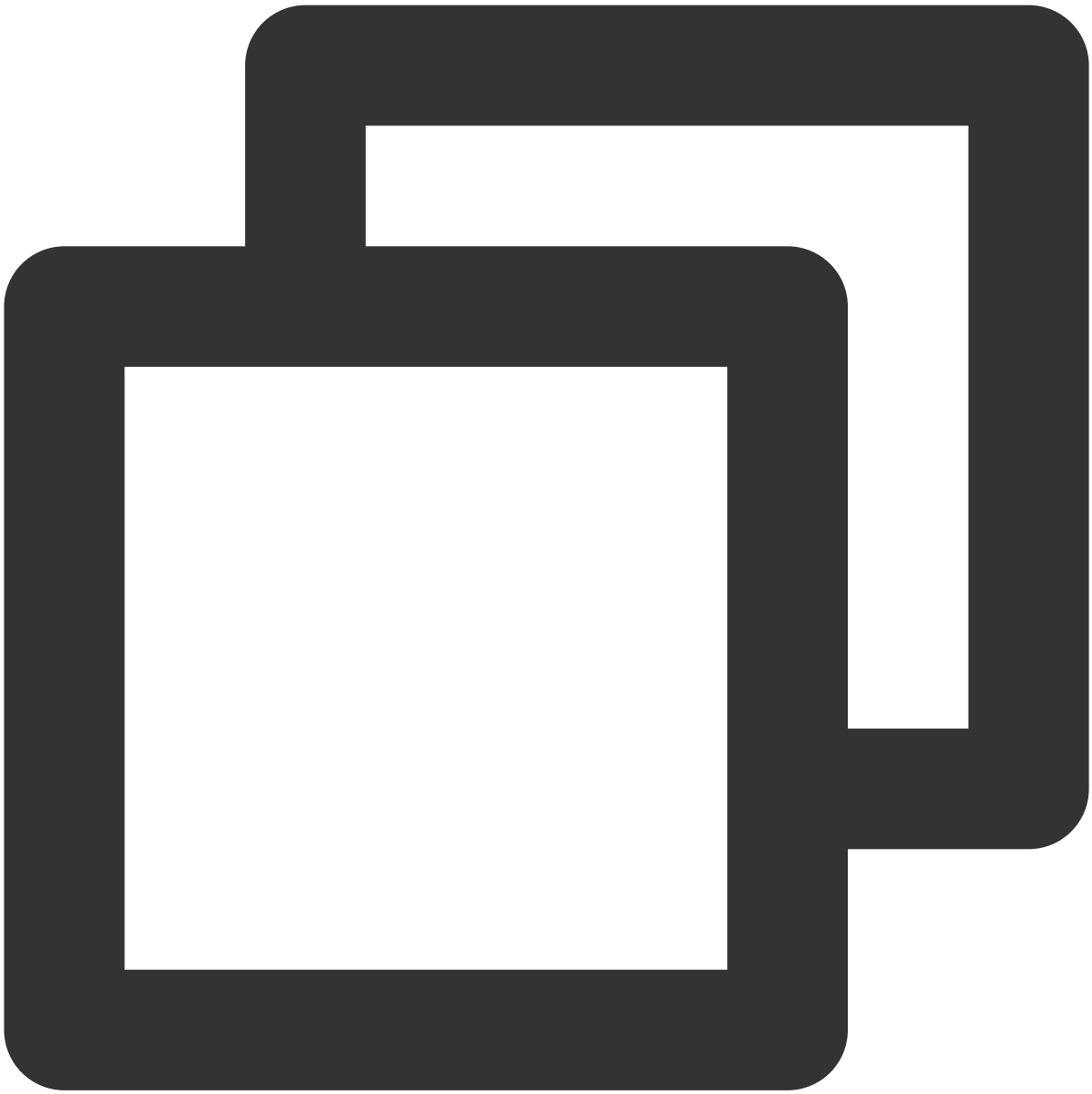
1. 下载配置文件添加到工程根目录。
2. 在项目级 build.gradle 文件中 buildscript -> dependencies 下添加以下配置：
```



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

dependencies {
...
classpath 'com.google.gms:google-services:4.2.0'
classpath 'com.huawei.agconnect:agcp:1.4.1.300'
}
```

3. 在应用级 build.gradle 文件中添加下方配置。

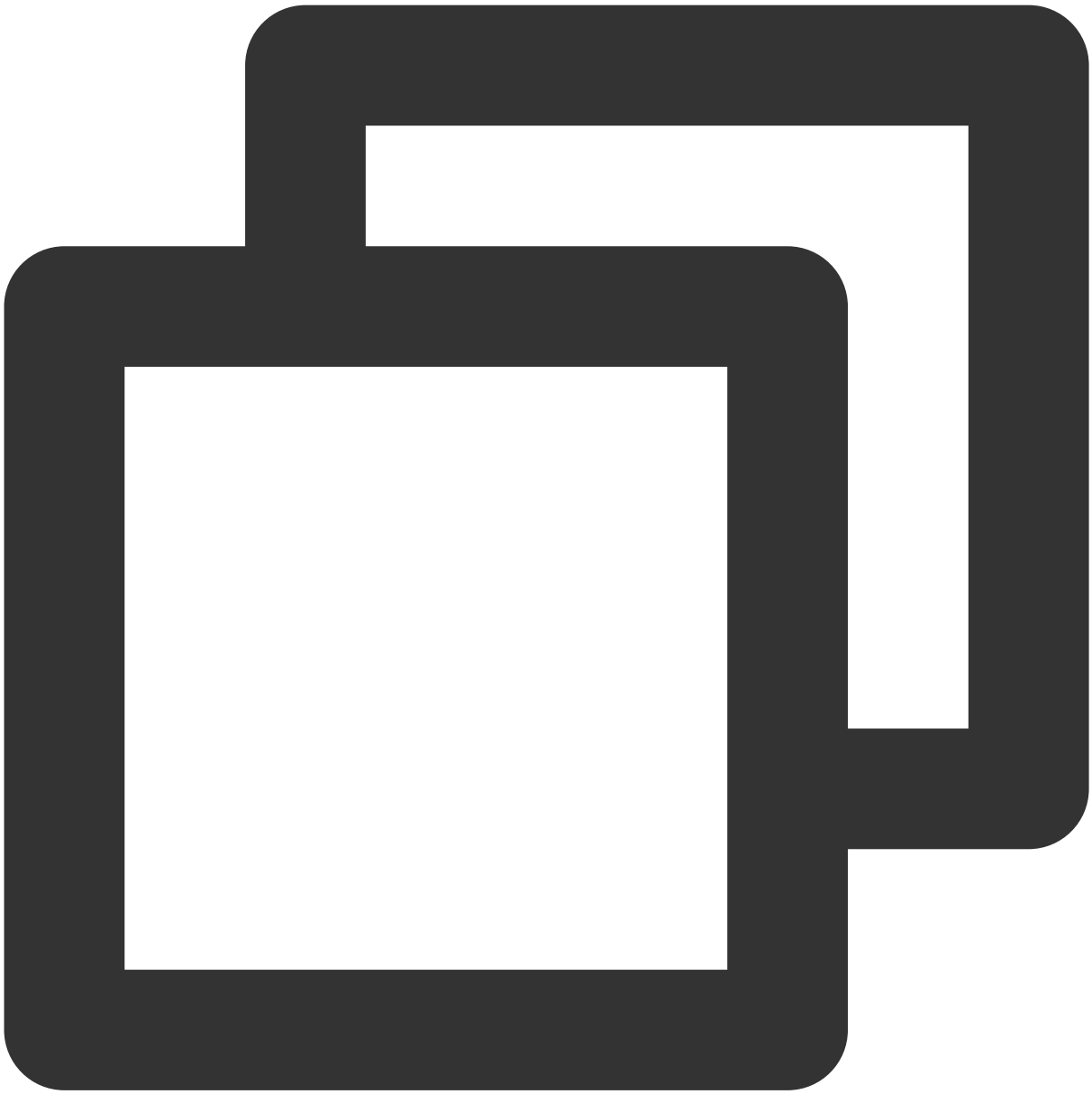


```
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
```

步骤2：推送参数配置

推送证书添加成功之后，IM 控制台会为您分配一个证书 ID，请您填充到 PrivateConstants 的配置参数里，该证书 ID 会在注册推送服务和上报 token 时使用, 以小米为例：

填充的参数如下：



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

以上步骤完成后，就可以收到离线推送通知了。

步骤3：发消息时设置离线推送参数

发送消息时，请您设置离线推送参数，具体请参见 发消息时设置离线推送参数 。

步骤4：解析离线推送消息

收到离线推送的通知栏消息，点击会自动跳转到您配置的跳转界面，该界面可以获取透传的离线消息，请您参见 解析离线推送消息 。

常见问题

离线推送怎么自定义推送的声音？

从 SDK 6.1.2155 版本开始，支持设置自定义铃音，支持机型有华为、小米、FCM 和 APNS。方法参见：V2TIMOfflinePushInfo 接口 setAndroidSound() 和 setIOSSound()。

收不到离线推送怎么排查？

1. OPPO 手机

OPPO 手机收不到推送一般有以下几种情况：
按照 OPPO 推送官网要求，在 Android 8.0 及以上系统版本的 OPPO 手机上必须配置 ChannelID，否则推送消息无法展示。配置方法可以参见 setAndroidOPPOChannelID。
OPPO 安装应用通知栏显示默认关闭，需要确认下开关状态。

2. Google FCM

收不到推送需要确认下 IM 控制台是否正确上传证书。排查路径参照文档 “IM 控制台配置 - Google FCM”，对照示意图看下是否添加正确。

3. 发送消息为自定义消息

自定义消息的离线推送和普通消息不太一样，自定义消息的内容我们无法解析，不能确定推送的内容，所以默认不推送，如果您有推送需求，需要您在 sendMessage 的时候设置 offlinePushInfo 的 desc 字段，推送的时候会默认展示 desc 信息。

4. 设备通知栏设置影响

离线推送的直观表现就是通知栏提示，所以同其他通知一样受设备通知相关设置的影响，以华为为例：
“手机设置-通知-锁屏通知-隐藏或者不显示通知”，会影响锁屏状态下离线推送通知显示。
“手机设置-通知-更多通知设置-状态栏显示通知图标”，会影响状态栏下离线推送通知的图标显示。

“手机设置-通知-应用的通知管理-允许通知”，打开关闭会直接影响离线推送通知显示。
“手机设置-通知-应用的通知管理-通知铃声” 和 “手机设置-通知-应用的通知管理-静默通知”，会影响离线推送通知铃音的效果。

5. 按照流程接入完成，还是收不到离线推送

首先在 IM 控制台通过 离线测试工具 自测下是否可以正常推送。
推送异常情况，设备状态异常，需要检查下 IM 控制台配置各项参数是否正确，再者需要检查下代码初始化注册逻辑，包括厂商推送服务注册和 IM 设置离线推送配置相关逻辑是否正确设置。
推送异常情况，设备状态正常，需要看下是否需要正确填写 channel ID 或者后台服务是否正常。
离线推送依赖厂商能力，一些简单的字符可能会被厂商过滤不能透传推送。
如果离线推送消息出现推送不及时或者偶尔收不到情况，需要看下厂商的推送限制。

跳转界面不成功怎么排查？

单击离线推送消息的通知栏，跳转到指定界面，原理是后台根据您在控制台配置的各个厂商的跳转方式和界面参数，根据厂商接口规则，传递给厂商服务器，单击时候进行对应界面启动跳转。对应界面启动还依赖清单文件的配置，必须和控制台配置的相对应，才能正确启动和跳转。
1. 首先需要重点排查下控制台和清单文件相关配置是否对应且正确，可参见 TUIKitDemo 的配置，注意部分厂商提供接口方式存在差异。
2. 如果跳转到了配置的界面，需要再看下配置界面内离线消息的解析和界面重定向是否正常。

厂商推送限制

1. 国内厂商都有消息分类机制，不同类型也会有不同的推送策略。如果想要推送及时可靠，需要按照厂商规则设置自己应用的推送类型为高优先级的系统消息类型或者重要消息类型。反之，离线推送消息会受厂商推送消息分类影响，与预期会有差异。
2. 另外，一些厂商对于应用每天的推送数量也是有限制的，可以在厂商控制台查看应用每日限制的推送数量。
如果离线推送消息出现推送不及时或者偶尔收不到情况，需要考虑下这里：
华为：将推送消息分为服务与通讯类和资讯营销类，推送效果和策略不同。另外，消息分类还和自分类权益有关：
无自分类权益，推送消息厂商还会进行二次智能分类 。
有申请自分类权益，消息分类会按照自定义的分类进行推送。
具体请参见 厂商描述。
vivo：将推送消息分为系统消息类和运营消息类，推送效果和策略不同。系统消息类型还会进行厂商的智能分类二次修正，若智能分类识别出不是系统消息，会自动修正为运营消息，如果误判可邮件申请反馈。另外，消息推送也受日推总数量限制，日推送量由应用在厂商订阅数统计决定。
具体请参见 厂商描述1 或 厂商描述2。
OPPO：将推送消息分为私信消息类和公信消息类，推送效果和策略不同。其中私信消息是针对用户有一定关注度，且希望能及时接收的信息，私信通道权益需要邮件申请。公信通道推送数量有限制。
具体请参见 厂商描述1 或 厂商描述2。

小米：将推送消息分为重要消息类和普通消息类，推送效果和策略不同。其中重要消息类型仅允许即时通讯消息、个人关注动态提醒、个人事项提醒、个人订单状态变化、个人财务提醒、个人状态变化、个人资源变化、个人设备提醒这8类消息推送，可以在厂商控制台申请开通。普通消息类型推送数量有限制。

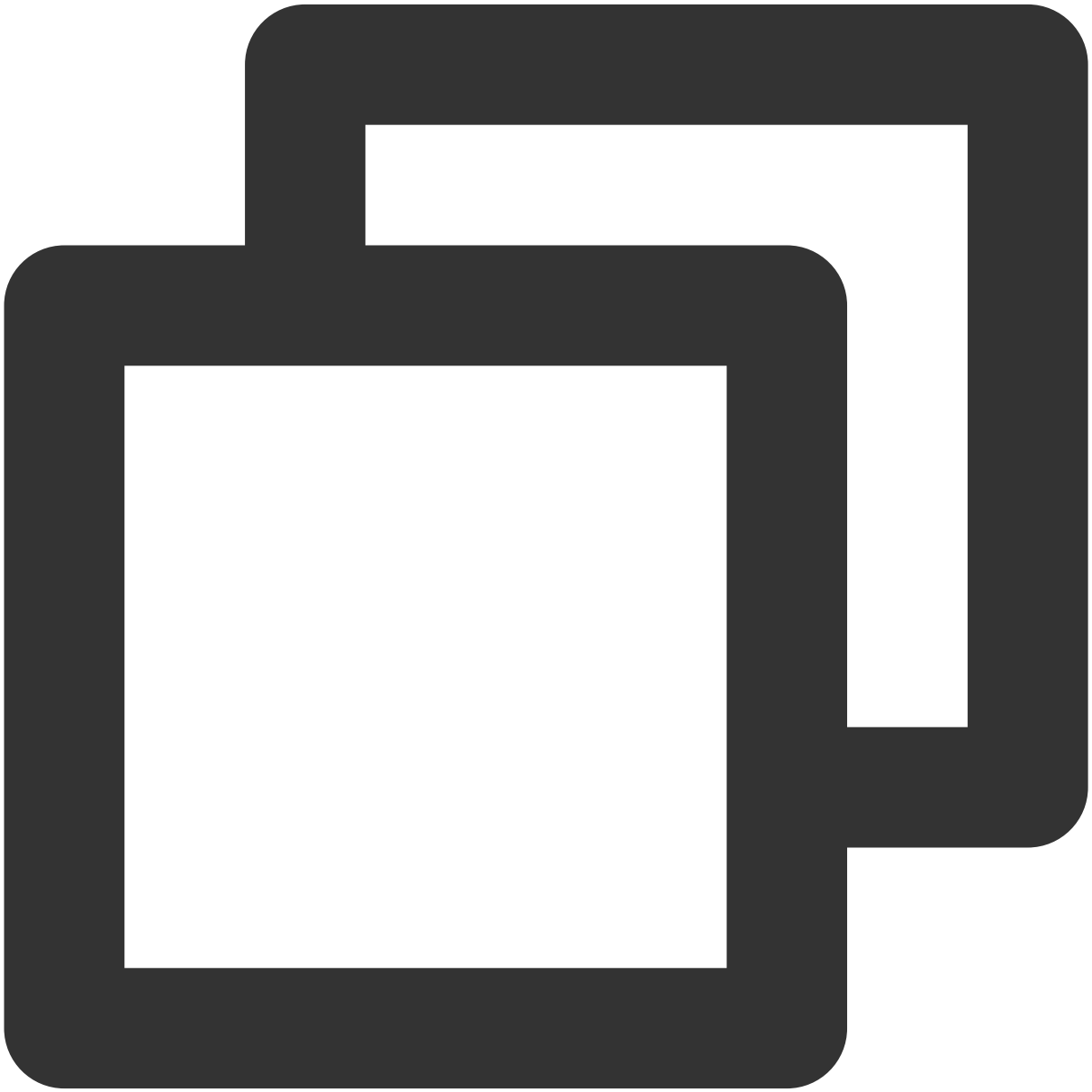
具体请参见 厂商描述1 或 厂商描述2。

魅族：推送消息数量有限制。

具体请参见 厂商描述。

FCM：推送上行消息频率有限制。

具体请参见 厂商描述。



iOS

最近更新时间：2024-03-14 14:24:34

概述

即时通信 IM 的终端用户需要随时都能够得知最新的消息，而由于移动端设备的性能与电量有限，当 App 处于后台时，为了避免维持长连接而导致的过多资源消耗，即时通信 IM 推荐您使用 Apple 提供的系统级推送通道（APNs）来进行消息通知，APNs 相比第三方推送拥有更稳定的系统级长连接，可以做到随时接受推送消息，且资源消耗大幅降低。

注意：

在没有主动退出登录的情况下，应用退后台、手机锁屏、或者应用进程被用户主动杀掉三种场景下，如果想继续接收到 IM 消息提醒，可以接入即时通信 IM 离线推送。

如果应用主动调用 logout 退出登录，或者多端登录被踢下线，即使接入了 IM 离线推送，也收不到离线推送消息。

集成 TUIOfflinePush 跑通离线推送功能

集成 TUIOfflinePush 组件之前，需要先向 Apple 申请 APNs 推送证书，然后 上传推送证书到 IM 控制台 。之后按照如下步骤操作即可快速接入 IM 离线推送。

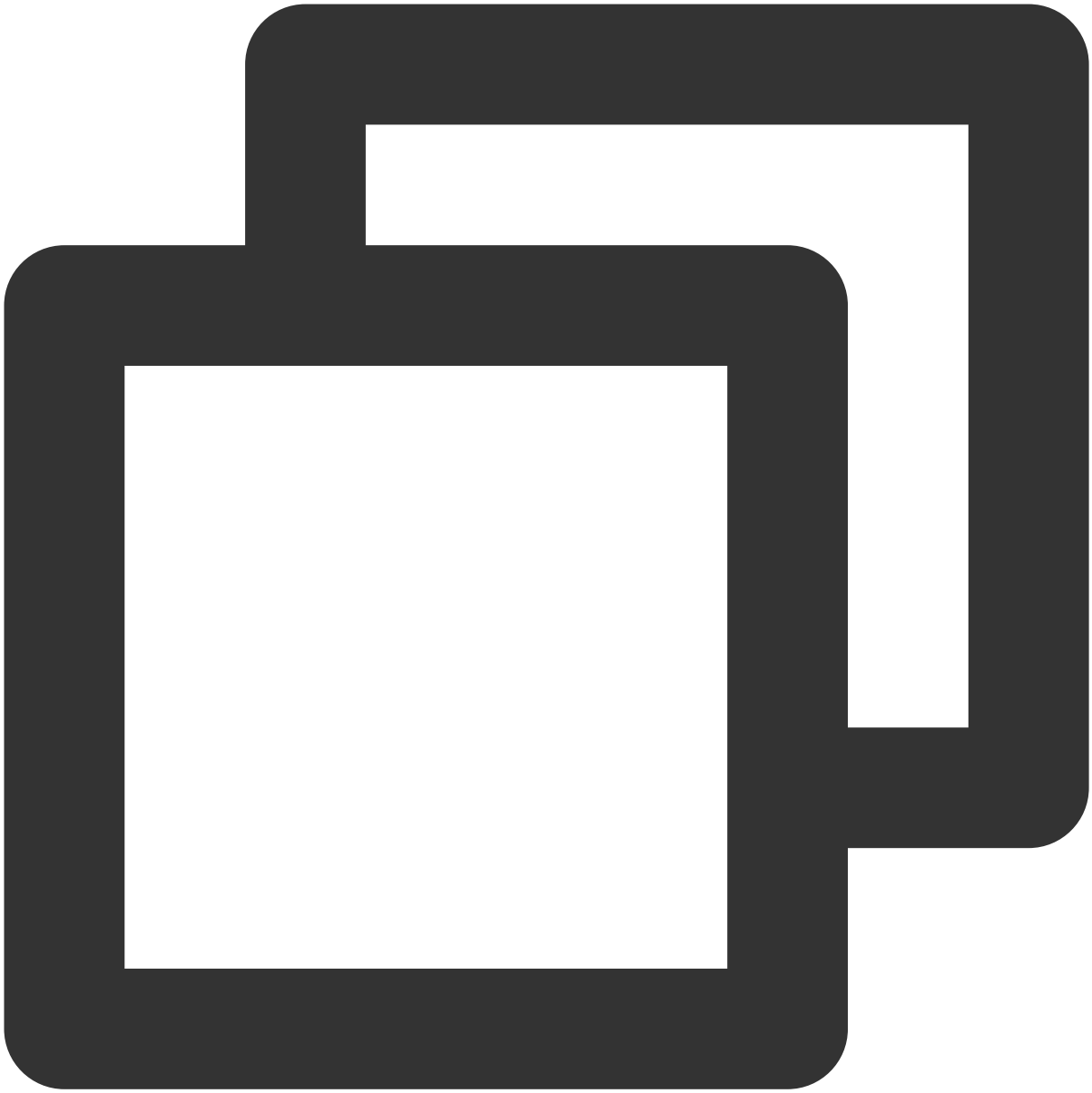
1. 集成 TUIOfflinePush 组件
2. 配置推送参数
3. 点击离线推送后自定义跳转

说明：

如果您想尽可能简单地接入 TUIOfflinePush 组件，您需要使用 TUICore 组件中的 TUILogin 提供的 login/logout 接口登录/登出，此时 TUIOfflinePush 组件会自动感知登录/登出事件。如果您不想使用 TUILogin 提供的接口，可参见 TUIOfflinePush 的 高级用法-自定义登录/登出。

步骤1：集成 TUIOfflinePush 组件

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

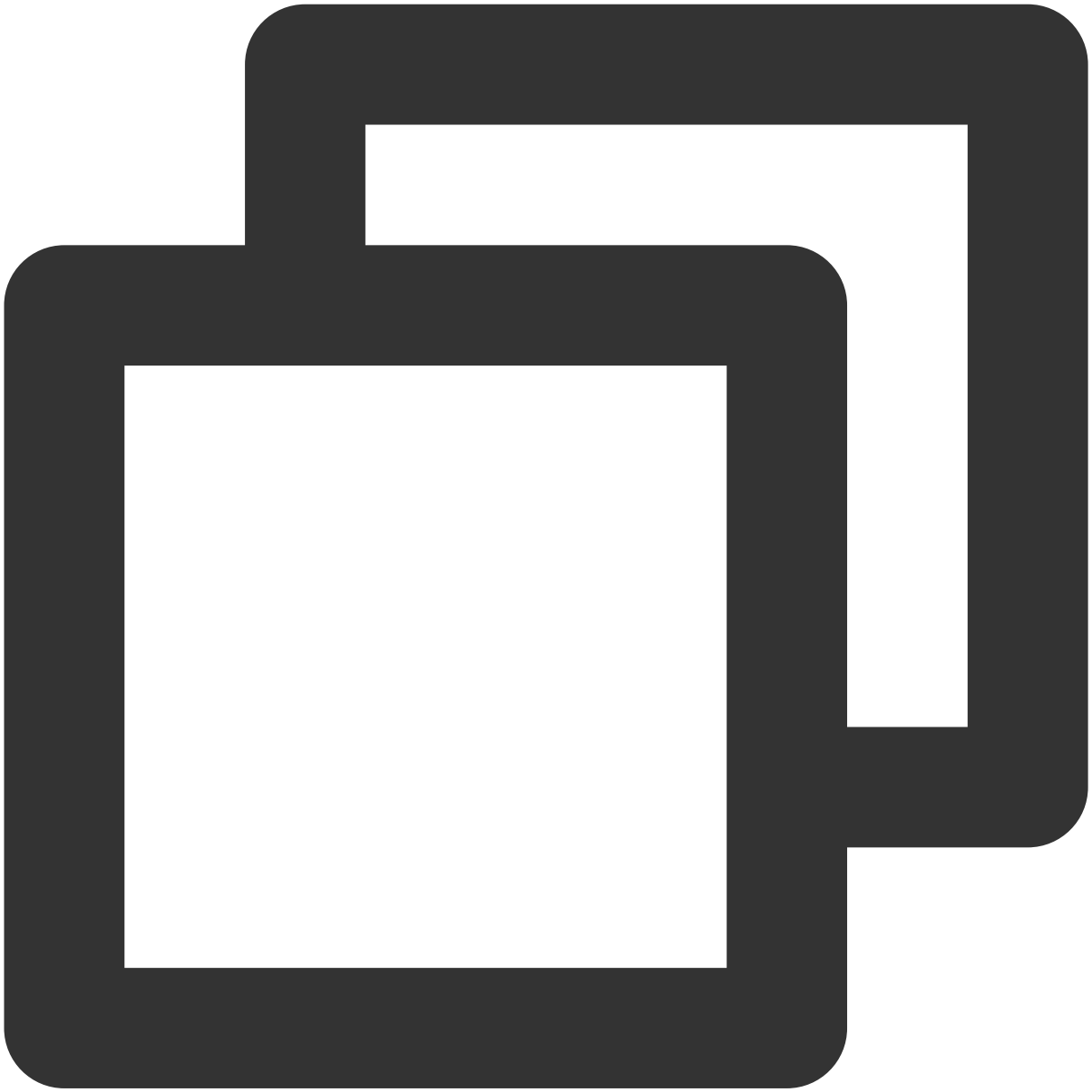


```
# 防止 TUI 组件里的 *.xcassets 与您项目里面冲突。
install! 'cocoapods', :disable_input_output_paths => true

# TUI 组件依赖了静态库，需要屏蔽如下设置，如果报错，请参见常见问题说明。
# use_frameworks!

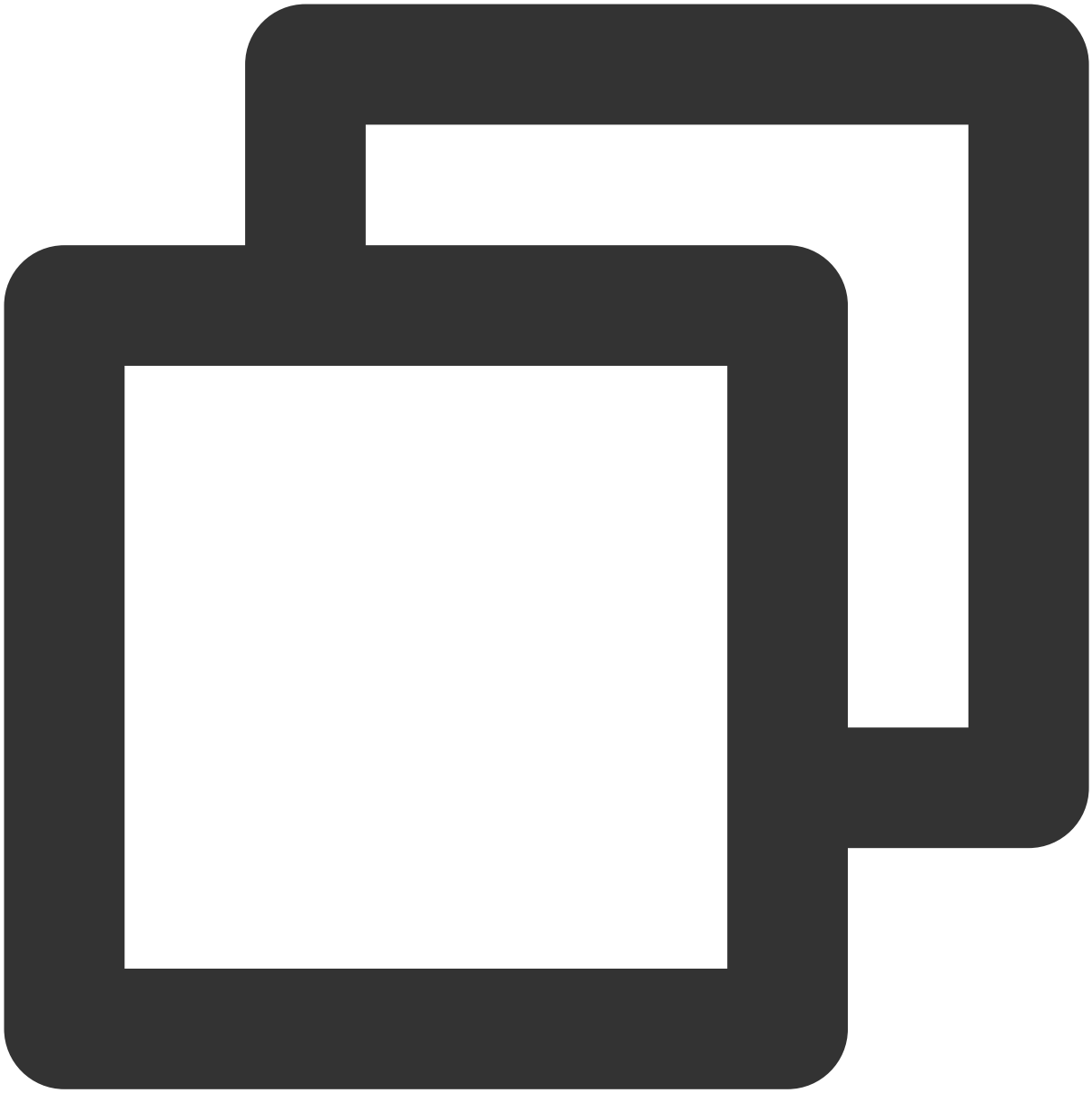
# 集成离线推送组件
pod 'TUIOfflinePush'
```

2. 执行以下命令，安装 TUIOfflinePush 组件。



```
pod install
```

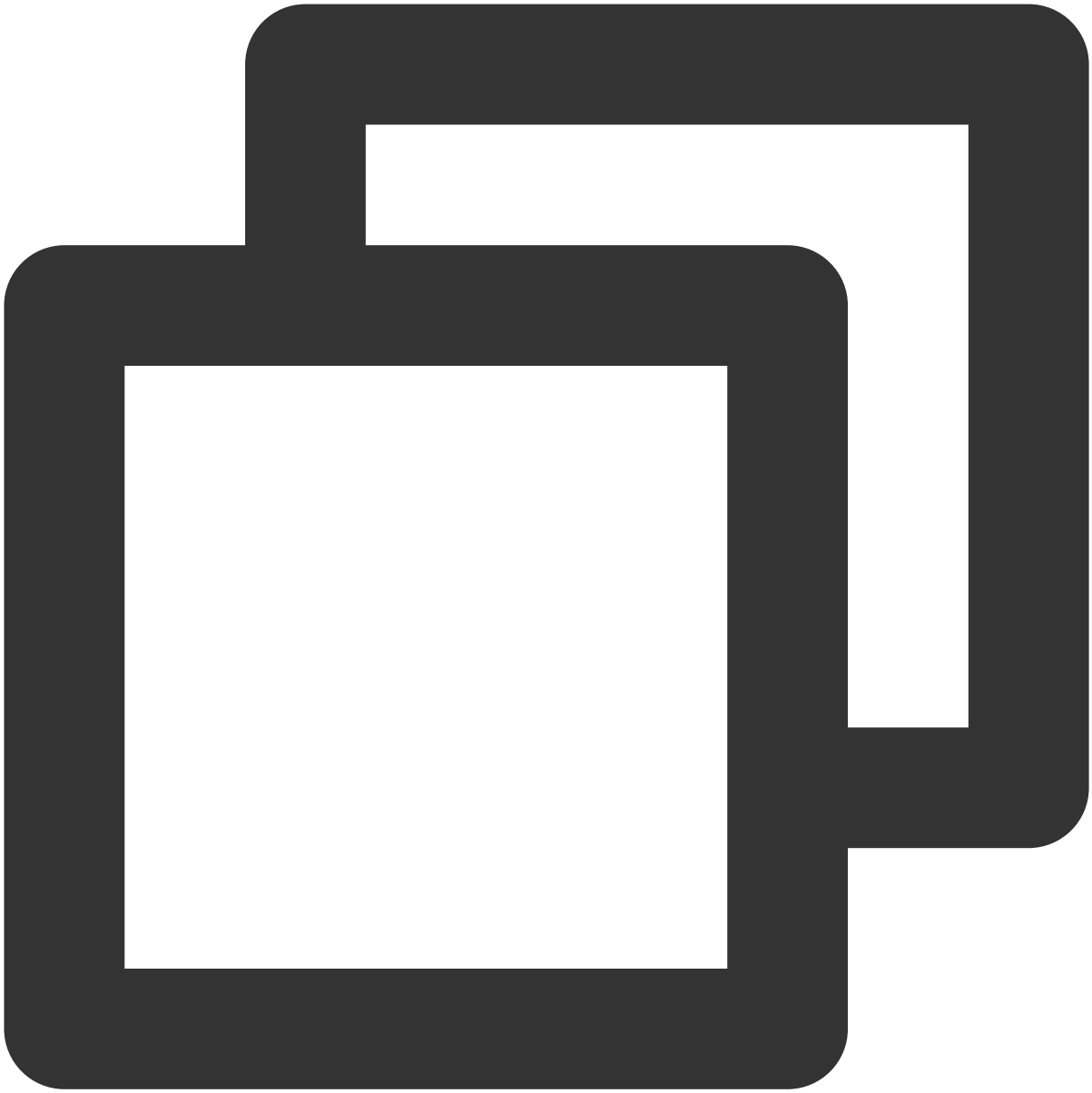
如果无法安装 TUIKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。



```
pod repo update
```

步骤2：配置推送参数

1. 当您 上传证书到 IM 控制台 后，IM 控制台会为您分配一个证书 ID 。
2. 您需要在 AppDelegate 中，调用宏 `TUIOfflinePushCertificateIDForAPNS` 设置下证书 ID 即可。



```
@implementation AppDelegate

#ifdef DEBUG
// 配置开发环境证书
TUIOfflinePushCertificateIDForAPNS(31287)
#else
// 配置生产环境证书
TUIOfflinePushCertificateIDForAPNS(31288)
#endif

@end
```

说明：

`TUIOfflinePushCertificateIDForAPNS` 是组件内置的宏定义，您只需要在 `AppDelegate` 的 `@implementation` 中的任意位置中调用即可。

步骤3：点击离线推送后自定义跳转

1. 点击通知栏的离线推送后，TUIOfflinePush 组件已支持推送内容的解析。
2. 如果要实现跳转到聊天列表，您只需要在 AppDelegate 中实现 `- navigateToTUIChatViewController:groupID:` 跳转方法即可。

说明：

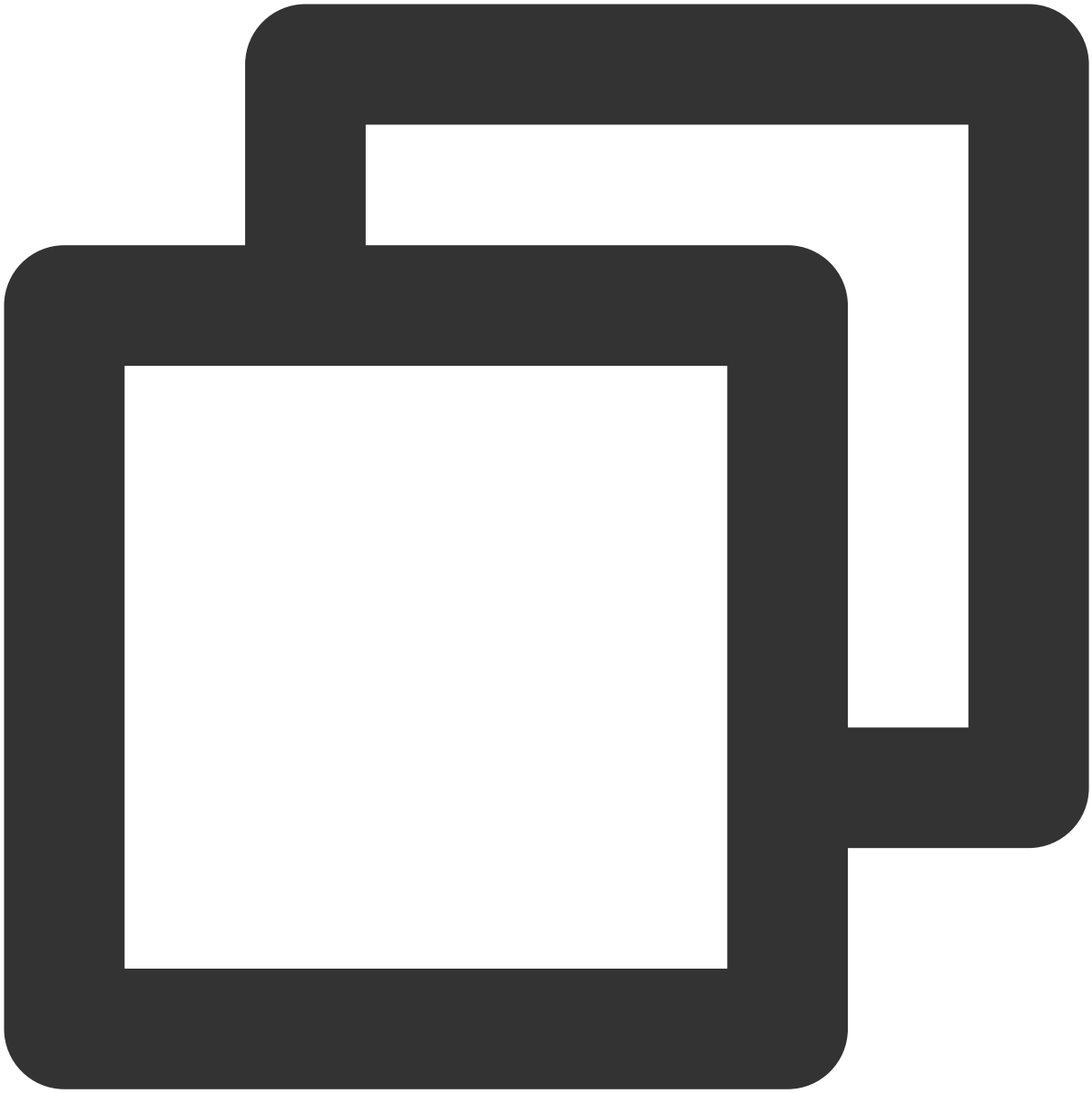
TUIOfflinePush 组件默认已经从离线推送中解析出当前推送的 userID 和 groupID。

如果 groupID 不为空，说明当前点击的是群聊离线消息。

如果 groupID 为空且 userID 不为空，说明当前点击的是单聊离线消息。

您需要在 AppDelegate 的 @implementation 中实现 `- navigateToTUIChatViewController:groupID:` 方法。

以下是示例代码，当点击离线推送后先获取当前的会话页面，然后通过会话页面 push 到聊天页面。您可以按需实现自己的跳转逻辑。



```
// 统一点击跳转
// 您可以直接拷贝当前的方法名到您的 AppDelegate 中
- (void)navigateToTUIChatViewController:(NSString *)userID groupID:(NSString *)grou
{
  // 示例：点击推送通知后，首先跳转到会话列表页面，然后再会话列表页跳转到聊天页面
  // 1. 获取当前 app 的 tabBarController
  // 2. 获取 tabBarController 的 firstObject，也即 ConversationController
  // 3. 执行 pushToViewController: 跳转到 ChatViewController
  // 跳转到聊天页面后，支持点击左上角的返回按钮回退到主页面

    UITabBarController *tab = [self getMainController];
```

```
    if (![tab isKindOfClass: UITabBarController.class]) {
        // 正在登录中
        return;
    }
    if (tab.selectedIndex != 0) {
        [tab setSelectedIndex:0];
    }
    self.window.rootViewController = tab;
    UINavigationController *nav = (UINavigationController *)tab.selectedViewControl
    if (![nav isKindOfClass:UINavigationController.class]) {
        return;
    }

    UIViewController *vc = nav.viewControllers.firstObject;
    if (![vc isKindOfClass:NSClassFromString(@"ConversationController")]) {
        return;
    }
    if ([vc respondsToSelector:NSSelectorFromString(@"pushToChatViewController:user
        [vc performSelector:NSSelectorFromString(@"pushToChatViewController:userID:
    }
}
```

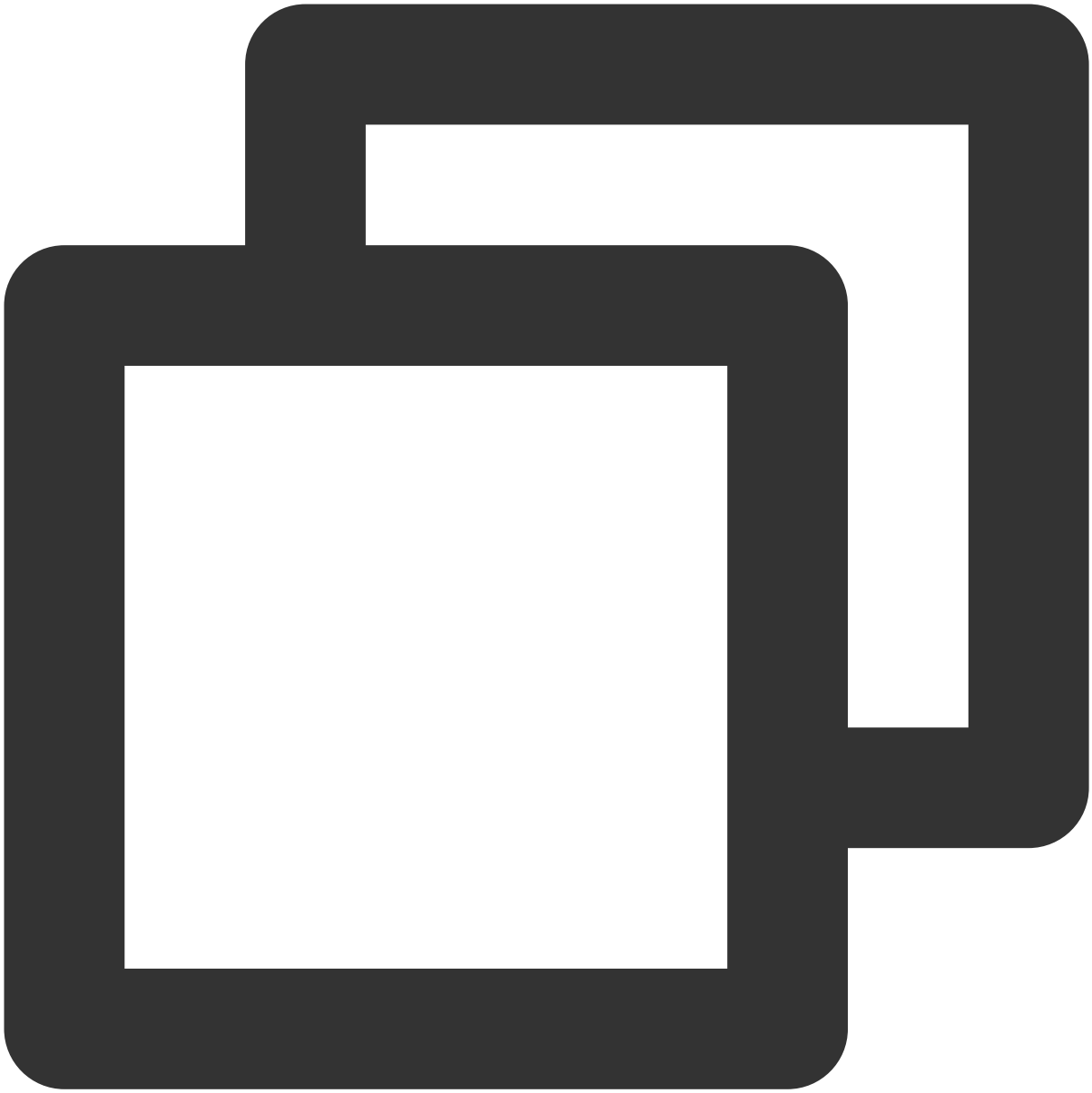
高级用法

1. 自定义登录/登出

TUIOfflinePush 默认使用了 TUICore 组件中的 TUILogin 提供的 login/logout 接口。如果您想自己实现 App/IM 的登录，不依赖 TUILogin，您需要在完成登录/登出操作后，手动调用 `registerService` 和 `unregisterService` 接口。

说明：

如果您使用了 TUILogin 的登录/登出，无需再调用上述两个接口。



```
// 您登录完成后的回调
- (void)onLoginSuccess
{
    // 调用 TUIOfflinePush 组件的登录
    [TUIOfflinePushManager.shareManager registerService];
}

// 您登出成功后的回调
- (void)onLogoutSuccess
{
    // 调用 TUIOfflinePush 的登出
```

```
    [TUIOfflinePushManager.shareManager unregisterService];
}
```

2. 自定义离线内容解析

TUIOfflinePush 默认参与解析了离线推送的内容，并通过 `-navigateToTUIChatViewController:groupID:` 接口回调给业务层自定义跳转。

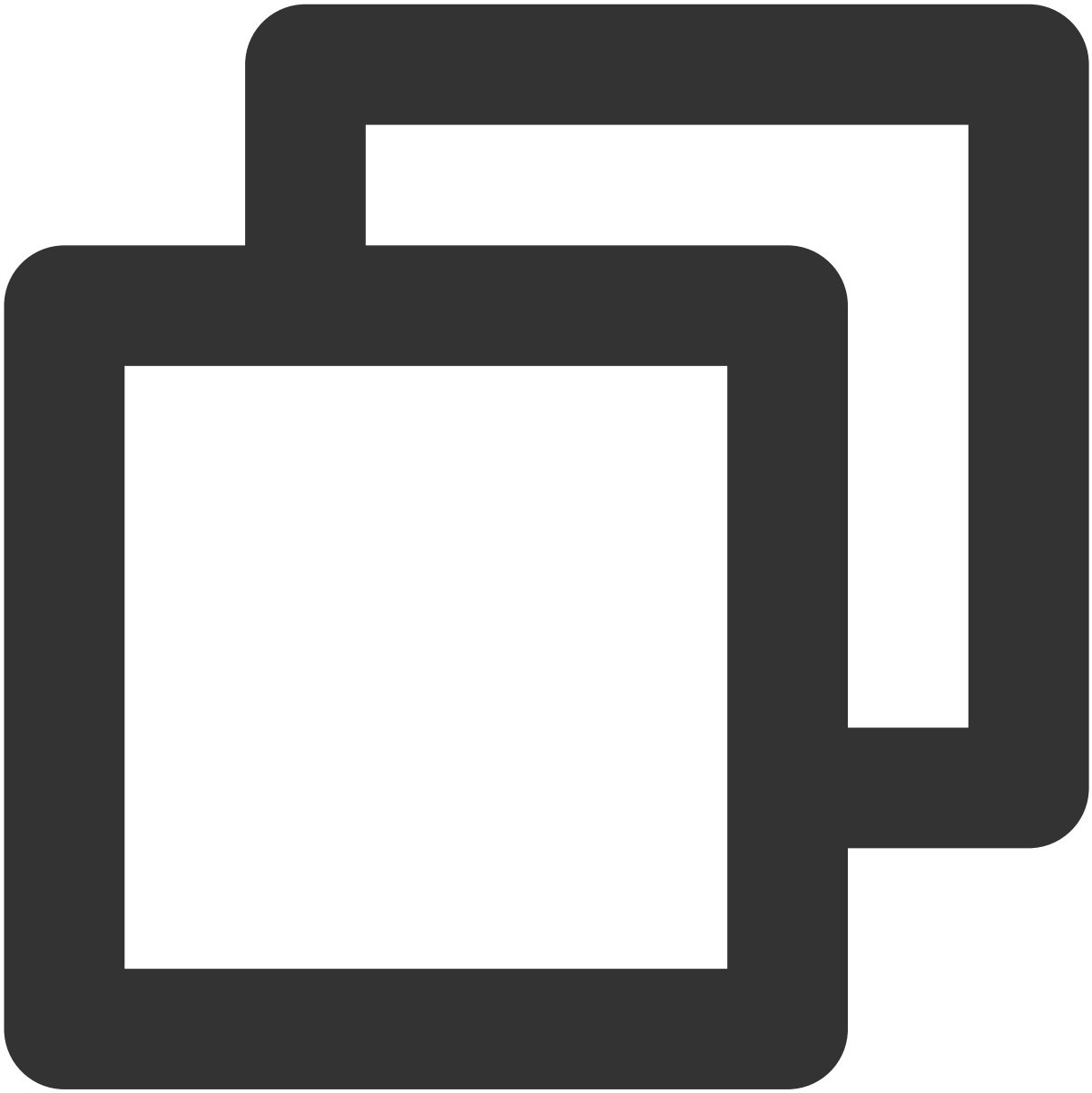
如果您想自定义解析离线推送的内容，或者查看收到的离线推送，可以在您的 AppDelegate 中实现 `-processTUIOfflinePushNotification:` 方法。

说明：

关于方法的返回值

如果返回 YES，那么组件将不再执行默认解析逻辑，完全交由业务层自行处理。

如果返回 NO，组件会继续执行默认解析逻辑，继续回调 - navigateToTUIChatViewController:groupID: 方法。



```
// 统一收到离线推送
- (BOOL)processTUIOfflinePushNotification:(NSDictionary *)userInfo
{
    // 自定义解析收到的 userInfo
    NSLog(@">>> 您可以在此处自定义解析, %@", userInfo);

    // 如果您不想执行 TUIOfflinePush 默认的解析逻辑，直接返回 YES
    // 如果您只是想查看推送的内容，依然依赖 TUIOfflinePush 的默认解析及统一跳转逻辑，直接返回 N
    return NO;
}
```

常见问题

普通消息为什么收不到离线推送？

首先，请检查下 App 的运行环境和证书的环境是否一致，如果不一致，收不到离线推送。
其次，检查下 App 和证书的环境是否为生产环境。如果是开发环境，向苹果申请 `deviceToken` 可能会失败，生产环境暂时没有发现这个问题，请切换到生产环境测试。

自定义消息为什么收不到离线推送？

自定义消息的离线推送和普通消息不太一样，自定义消息的内容我们无法解析，不能确定推送的内容，所以默认不推送，如果您有推送需求，需要您在 sendMessage 的时候设置 offlinePushInfo 的 `desc` 字段，推送的时候会默认展示 `desc` 信息。

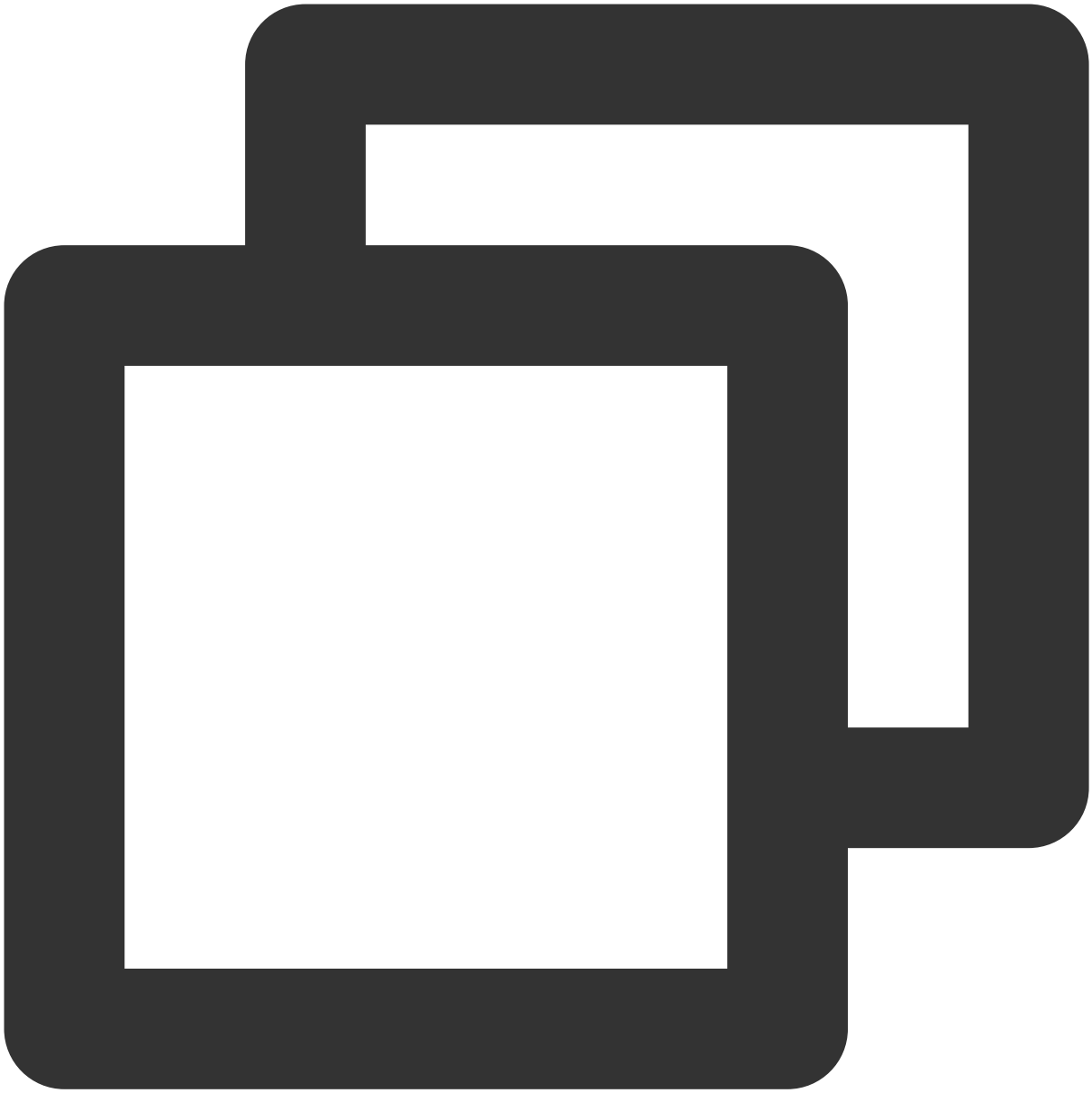
如何关闭离线推送消息的接收？

如果您想关闭离线推送消息的接收，可以通过设置 setAPNS 接口的 `config` 参数为 `nil` 来实现。该功能从 5.6.1200 版本开始支持。

收不到推送，且后台报错 bad devicetoken。

Apple 的 deviceToken 与当前编译环境有关。如果 登录 IMSDK 后上传 deviceToken 到腾讯云 所使用的证书 ID 和 token 不一致，就会报错。
如果使用的是 Release 环境编译，则 `-application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是发布环境的 token，此时 businessID 需要设置生产环境的 证书 ID。
如果使用的是 Debug 环境编译，则 `-application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是开发环境的 token，此时 businessID 需要设置开发环境的 证书 ID。

版权所有：腾讯云计算（北京）有限责任公司



```
V2TIMAPNSConfig *confg = [[V2TIMAPNSConfig alloc] init];
/* 用户自己到苹果注册开发者证书，在开发者帐号中下载并生成证书(p12 文件)，将生成的 p12 文件传到腾讯
//推送证书 ID
confg.businessID = sdkBusiId;
confg.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:confg succ:^{
   NSLog(@"%s, succ, %@", __func__, supportTPNS ? @"TPNS": @"APNS");
} fail:^(int code, NSString *msg) {
   NSLog(@"%s, fail, %d, %@", __func__, code, msg);
}];
```

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

此问题现象是由于 APNs 服务不稳定导致的，可尝试通过以下方式解决：

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

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-01-31 14:24:45

腾讯云即时通信 IM 的终端用户需要随时都能够得知最新的消息，而由于移动端设备的性能与电量有限，当 App 处于后台时，为了避免维持长连接而导致的过多资源消耗，腾讯云即时通信 IM 推荐您使用各厂商提供的系统级推送通道来进行消息通知，系统级的推送通道相比第三方推送拥有更稳定的系统级长连接，可以做到随时接受推送消息，且资源消耗大幅降低。

说明：

在没有主动退出登录的情况下，应用退后台、手机锁屏、或者应用进程被用户主动杀掉三种场景下，如果想继续接收到 IM 消息提醒，可以接入即时通信 IM 离线推送。

如果应用主动调用 logout 退出登录，或者多端登录被踢下线，即使接入了 IM 离线推送，也收不到离线推送消息。

使用腾讯云 IM 厂商推送 Flutter 集成插件的离线推送能力，可快速接入主流厂商（苹果iOS/Google FCM/OPPO/VIVO/华为/小米/魅族）的离线推送。

本教程含接入腾讯云即时通信 IM 离线推送全流程。插件已封装上述厂商的 SDK，使用时仅需简单改造调用即可。

如果您的应用不需要离线推送，或场景不满足离线推送的需求，请直接看本文最后一节 “在线推送-在本地创建新消息通知” 在线推送部分。

如果您的应用已经自行完成厂商离线推送，仅需查看本教程 第一步 和 第五步，在控制台内录入厂商信息，并在应用登录后，上报证书 ID 即可。

插件 API 概览

说明：

以下 API 若无特殊说明，均可自动兼容 Android/iOS 平台及支持厂商，插件内部进行平台及厂商判断，您直接调用即可。

API	说明
构造函数（TimUiKitPushPlugin）	实例化一个 Push 插件对象，并确定是否使用 Google Service FCM
init	初始化插件，绑定点击通知回调事件及传入厂商渠道信息
uploadToken	自动获取设备 Token 及证书 ID，自动上传至腾讯云 IM 服务端
requireNotificationPermission	申请推送权限
setBadgeNum	设置未读数角标 （仅支持部分 Android 设备，可参见 API 代码参数说明）
clearAllNotification	清除通知栏内，当前应用，所有的通知
getDevicePushConfig	获取当前厂商的推送相关信息，含机型/证书 ID/Token

getDevicePushToken	获取当前厂商的推送 Token
getOtherPushType	获取厂商信息
getBuzId	获取当前厂商对应的腾讯云控制台上注册的证书 ID
createNotificationChannel	为Android机型创建通知Channel渠道，详见Google官方文档
clearAllNotification	清除通知栏内，当前应用，所有的通知
displayNotification	在客户端本地，手动创建一条消息通知
displayDefaultNotificationForMessage	在客户端本地，按照默认的规则，自动为一个 `V2TimMessage` 创建一个消息通知

接入准备（注册厂商）

需要完成厂商开发者账号申请（一般需要企业认证），创建应用，申请 PUSH 权限，拿到 key 信息。

Apple

iOS

1. 请根据该教程，完成 Apple 推送证书申请。
2. 分别将申请到的生产环境和开发环境证书托管至即时通信 IM 控制台。
3. 打开 IM 控制台-基础配置 右侧，添加 iOS 证书。

Android

Google FCM

1. 前往 Google Firebase控制台 创建一个项目，无需启用 Google Analysis。
2. 单击您的应用卡片，进入**应用配置**页面。
3. 单击 Project Overview 右侧的

，选择**项目设置>服务帐号**，单击**生成新的私钥**下载私钥文件。
4. 将该私钥文件托管至即时通信 IM 控制台。在 IM 控制台-基础配置 右侧，添加 Android 证书。选择 Google 后，请选择上传证书。

OPPO

开通服务

请参见 OPPO PUSH 服务开启指南 注册开发者账号，创建应用，并开通 PUSH 服务。
在 OPPO 推送平台 >**配置管理>应用配置**页面，您可以查看详细的应用信息。记录 AppId、AppKey、AppSecret 和 MasterSecret 信息。

创建消息通道

按照 OPPO 官网要求，在 OPPO Android 8.0 及以上系统版本必须配置 ChannelID，否则推送消息无法展示。您需要先在 App 中创建对应的 ChannelID（例如 `tuikit`）。
请在**配置管理-新建通道**内，创建一个新通道。**通道ID**即为**Channel ID**。
说明：
OPPO 对于公信通道有每日上限，对于通讯类型消息，建议参见 OPPO 官方文档 申请私信通道。

上传证书至控制台

1. 在 IM 控制台-基础配置 右侧，添加 Android 证书。选择 OPPO 后，请填写相关信息。
2. ChannelID 请填写此前在 OPPO 控制台为通讯能力申请的专用通道，最好是私信，以免到达每日推送上限。
3. 打开方式请选择**打开应用内指定页面** > **activity**，填写 `com.tencent.flutter.tim_ui_kit_push_plugin.pushActivity.OPPOMessageActivity` 。

小米

开通服务

打开 小米开放平台官网 进行注册并通过开发者认证。
说明：
认证过程大约需要2天左右，请务必提前阅读 小米推送服务启用指南，以免影响您的接入进度。
在小米开放平台创建应用，并选择**应用服务>PUSH服务**，创建推送服务应用。
小米推送服务应用创建完成后，在应用详情中，您可以查看详细的应用信息。
记录**主包名**、**AppID**、**AppSecret** 信息。

上传证书至控制台

在 IM 控制台-基础配置 右侧，添加 Android 证书。选择小米后，请填写相关信息，行为请选择打开应用。

vivo

开通服务

打开 vivo 开放平台官网 进行注册并通过开发者认证。
说明：
认证过程大约需要3天左右，请务必提前阅读 vivo 推送服务说明，以免影响您的接入进度。
1. 登录 vivo 开放平台的管理中心，选择**消息推送>创建>测试推送**，创建 vivo 推送服务应用。
2. vivo 推送服务应用创建完成后，在应用详情中，您可以查看详细的应用信息。记录 APP ID、APP key 和 App secret信息。

说明：
vivo 要求应用在上架后，才能使用正式推送服务。如果您需要在开发中调试 vivo 机器，请参见本文最后一节 vivo 调试 内容，开启测试模式。

上传证书至控制台

在 IM 控制台-基础配置 右侧，添加 Android 证书。选择 vivo 后，请填写相关信息。
单击后续动作请选择为：打开应用内指定页面。
应用内页面 配置为： `tencent_im_push://${替换成您的包名}/message?#Intent;scheme=tencent_im_push;launchFlags=0x4000000;end`

华为

获取密钥

1. 进入 华为开放平台，注册和登录开发者账号，详情参见 账号注册认证（如果您是新注册账号，需进行实名认证）。
2. 在华为推送平台中新建应用，详情参见 创建应用。记录**AppID**、**AppSecret** 信息。

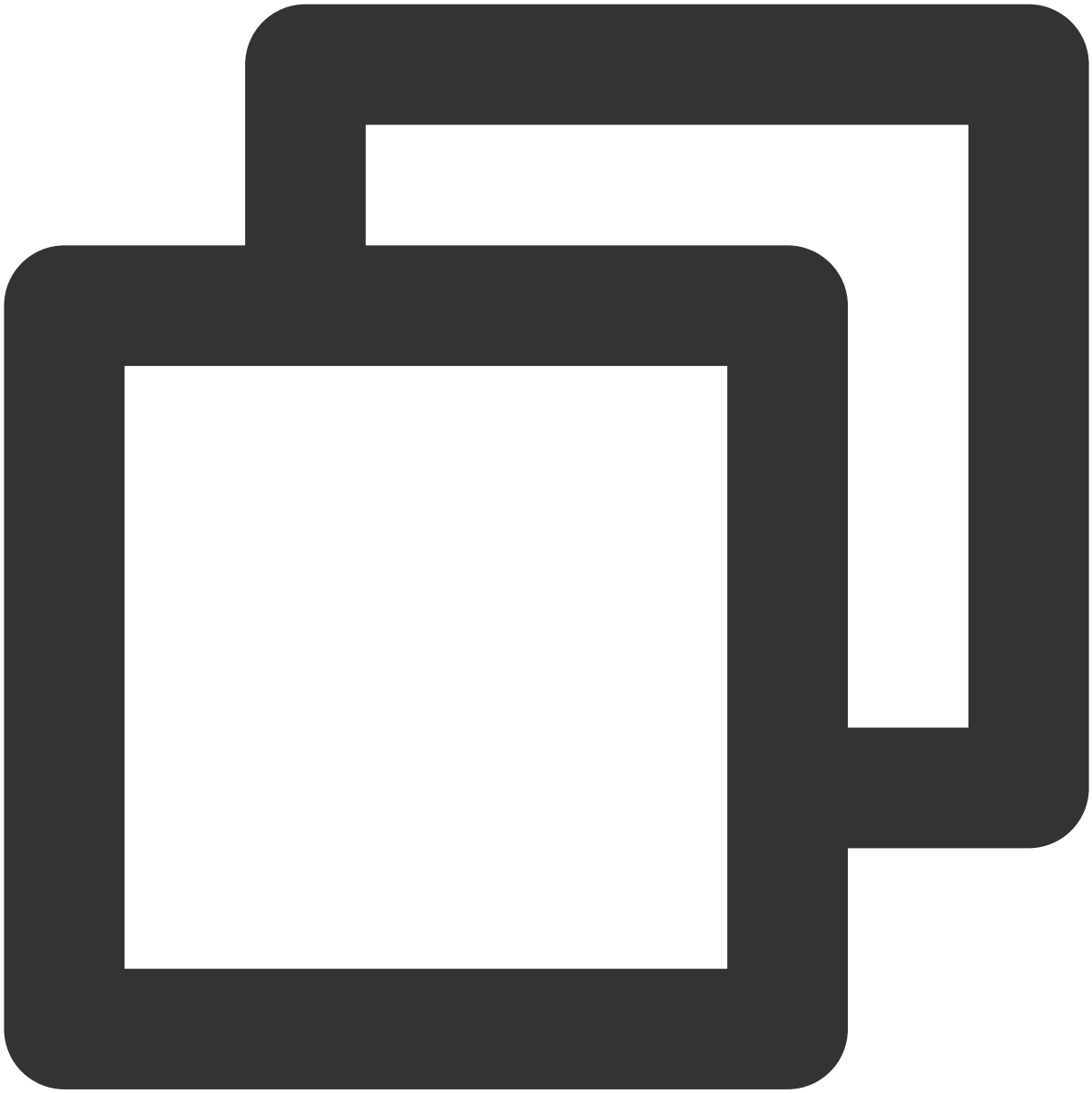
说明：
若在**应用信息>我的应用**中无法找到 SecretKey，可前往项目**设置>常规**中查看 Client Secret。

配置 SHA256 证书指纹

获取 SHA256 证书指纹，并在华为推送平台中配置证书指纹，**单击**

保存。证书指纹获取可参见 生成签名证书指纹。

说明：
如果您的应用需要经过流水线编译发布，每次编译在不同的构建机上进行，可在本地创建 `keystore.jks` 密钥文件，得到该 keystore 的 SHA256 值，填入华为推送平台中。
在流水线的构建脚本中，对完成构建后的产物进行归档对齐，及使用刚才的 keystore 签名。此时该最终产物签名 SHA256 值即可保持一致。代码如下：



```
zipalign -v -p 4 构建生成的apk.apk 打包生成的apk_aligned.apk
apksigner sign --ks keystore.jks --ks-pass pass:您创建的keystore密码 --out 最终签名  完
```

获取华为推送配置文件

登录华为开放平台，进入**我的项目**> 选择项目 > **项目设置**，下载华为应用最新配置文件 agconnect-services.json。放置于 `android/app` 目录下。

打开推送服务开关

在华为推送平台，单击**全部服务**>**推送服务**，进入推送服务页面。

在**推送服务**页面，单击**立即开通**，详情请参见 打开推送服务开关。

上传证书至控制台

1. 在 IM 控制台-基础配置 右侧，添加 Android 证书。
2. 选择华为后，请填写相关信息。

角标参数 请填写 Android 应用入口 Activity 类，如我们DEMO的 `com.tencent.flutter.tuikit` ，否则华为通道下发通知的角标设置将不生效。

点击后续动作 请选择打开应用。

魅族

开通服务

1. 打开 魅族开放平台官网 进行注册并通过开发者认证。

说明：

认证过程大约需要3天左右，请务必提前阅读 魅族 Flyme 推送接入文档，以免影响您的接入进度。

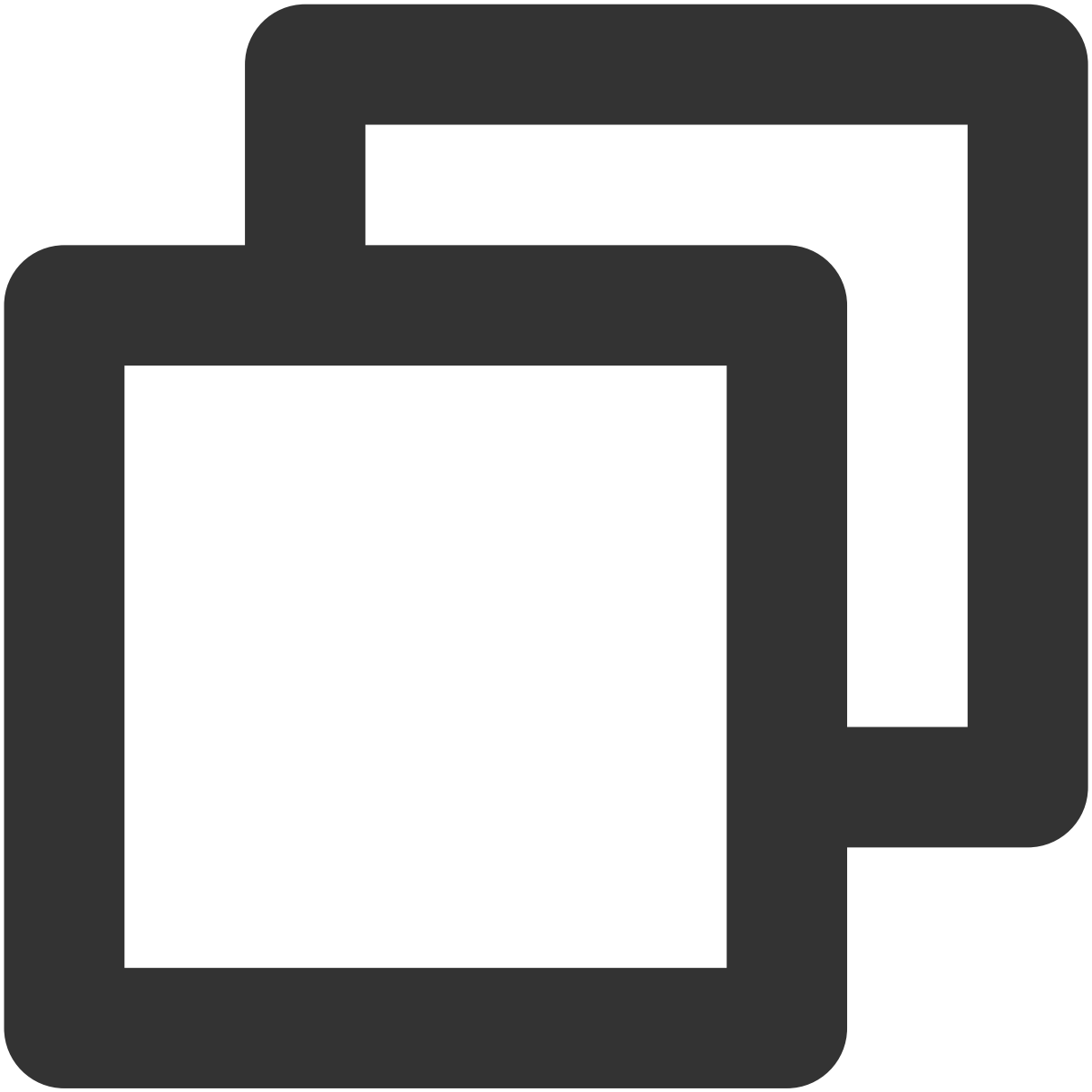
2. 登录魅族开放平台的管理控制台，选择**服务>集成推送服务>推送后台**，创建魅族推送服务应用。
3. 魅族推送服务应用创建完成后，在应用详情中，您可以查看详细的应用信息。记录**应用包名**、**App ID**、**App Secret**信息。

上传证书至控制台

1. 在 IM 控制台-基础配置 右侧，添加 Android 证书。
2. 选择魅族后，请填写相关信息。**单击后续动作**请选择：**打开应用**。

使用插件跑通离线推送（全览 + Android）

在您的项目中安装 IM Flutter 离线推送插件：



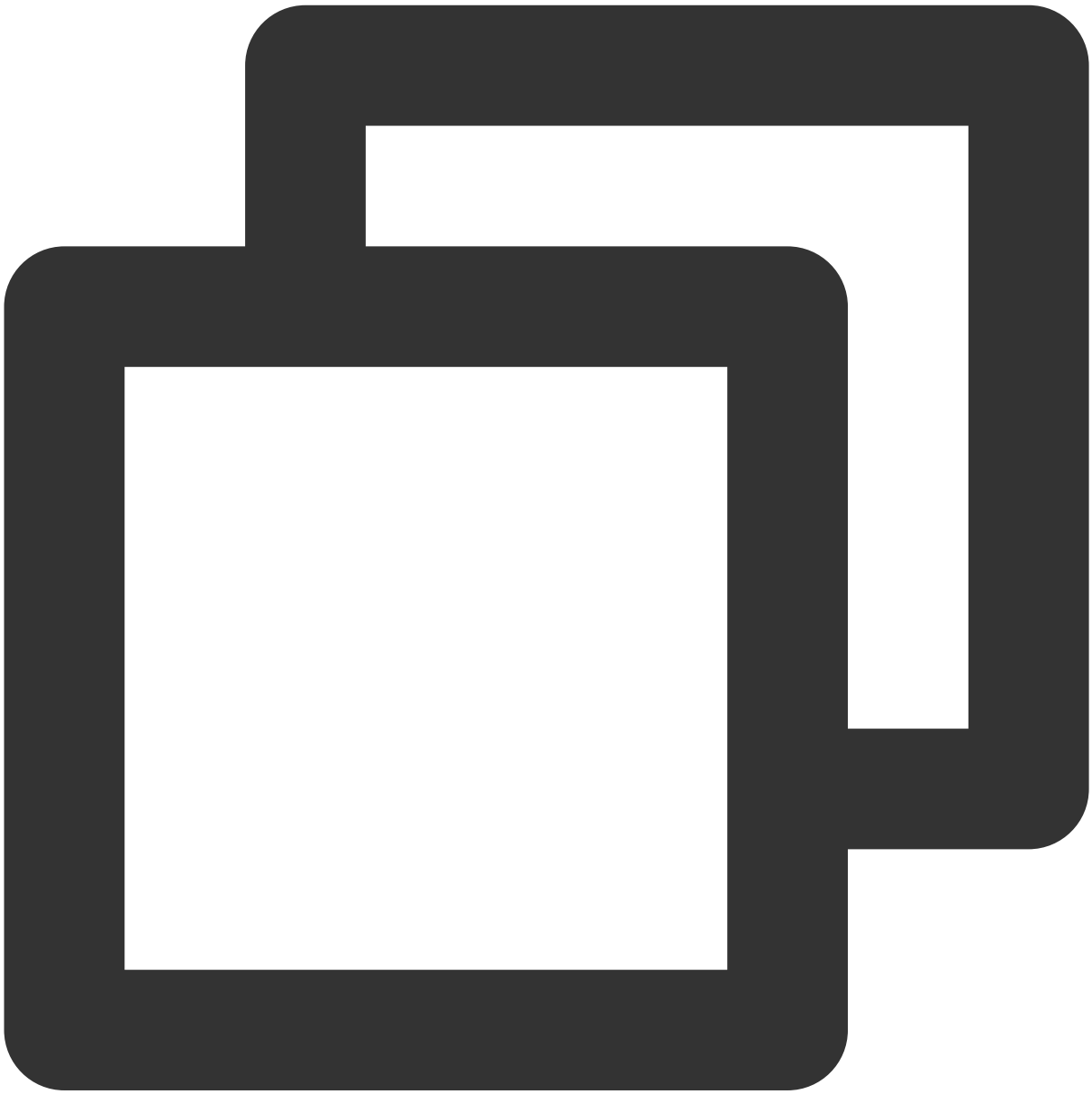
```
flutter pub add tim_ui_kit_push_plugin
```

在插件市场，启用推送插件。

步骤1: 汇总常量类

1. 完成 接入准备（注册厂商）的配置后，可在即时通信 IM 的控制台首页右侧，查看我们后台为您的厂商渠道 App 信息分配的证书 ID。
2. 请将这些信息，配上厂商渠道的账号信息，实例化一个静态的 `PushAppInfo` 类，汇总起来。后续步骤需要传入此对象。

3. 该类支持配置所有您需要接入厂商推送机型的信息。无需完整填写构造函数字段。若需要使用某个厂商平台，请完整填写该平台相关字段。



```
import 'package:tim_ui_kit_push_plugin/model/appInfo.dart';

static final PushAppInfo appInfo = PushAppInfo(
 hw_buz_id: , // 华为证书ID
 mi_app_id: , // 小米APPID
 mi_app_key: , // 小米APPKey
 mi_buz_id: , // 小米证书ID
 mz_app_id: , // 魅族APPID
```

```
  mz_app_key: , // 魅族APPKey
  mz_buz_id: , // 魅族证书ID
  vivo_buz_id: , // vivo证书ID
  oppo_app_key: , // OPPO APPKey
  oppo_app_secret: , // OPPO APP Secret
  oppo_buz_id: , // OPPO证书ID
  oppo_app_id: , // OPPO APPID
  google_buz_id: , // Google FCM证书ID
  apple_buz_id: , // Apple证书ID
);
```

说明：

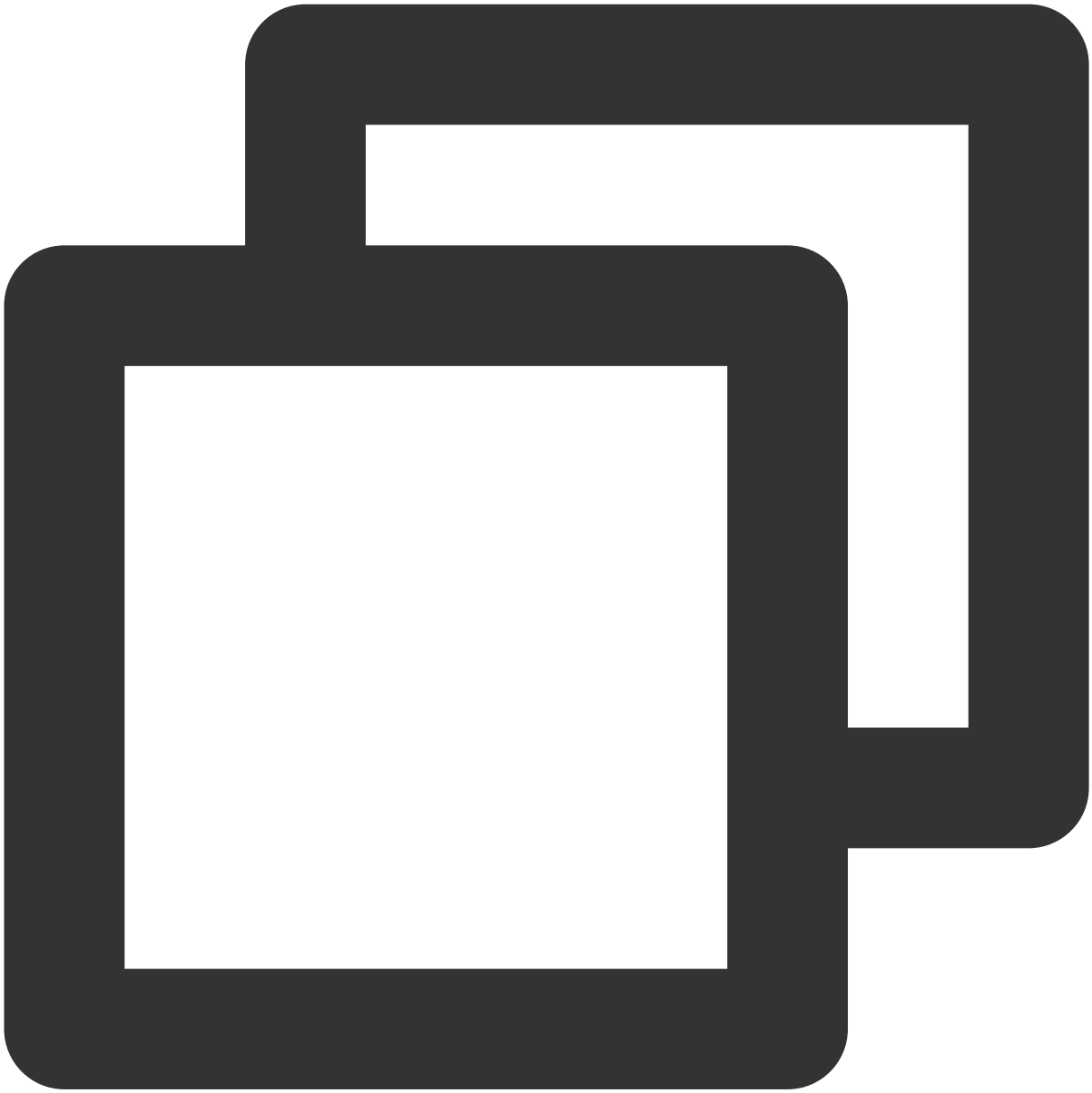
可参见我们DEMO lib/utils/push/push_constant.dart文件 中的做法。

步骤2: 代码中添加厂商工程配置

Google FCM

兼容 Android 模拟器调试

如果需要使用 Firebase Emulator Suite，请打开 `android/app/src/main/AndroidManifest.xml` 文件，在 `application` 中新增 `usesCleartextTraffic` 字段。



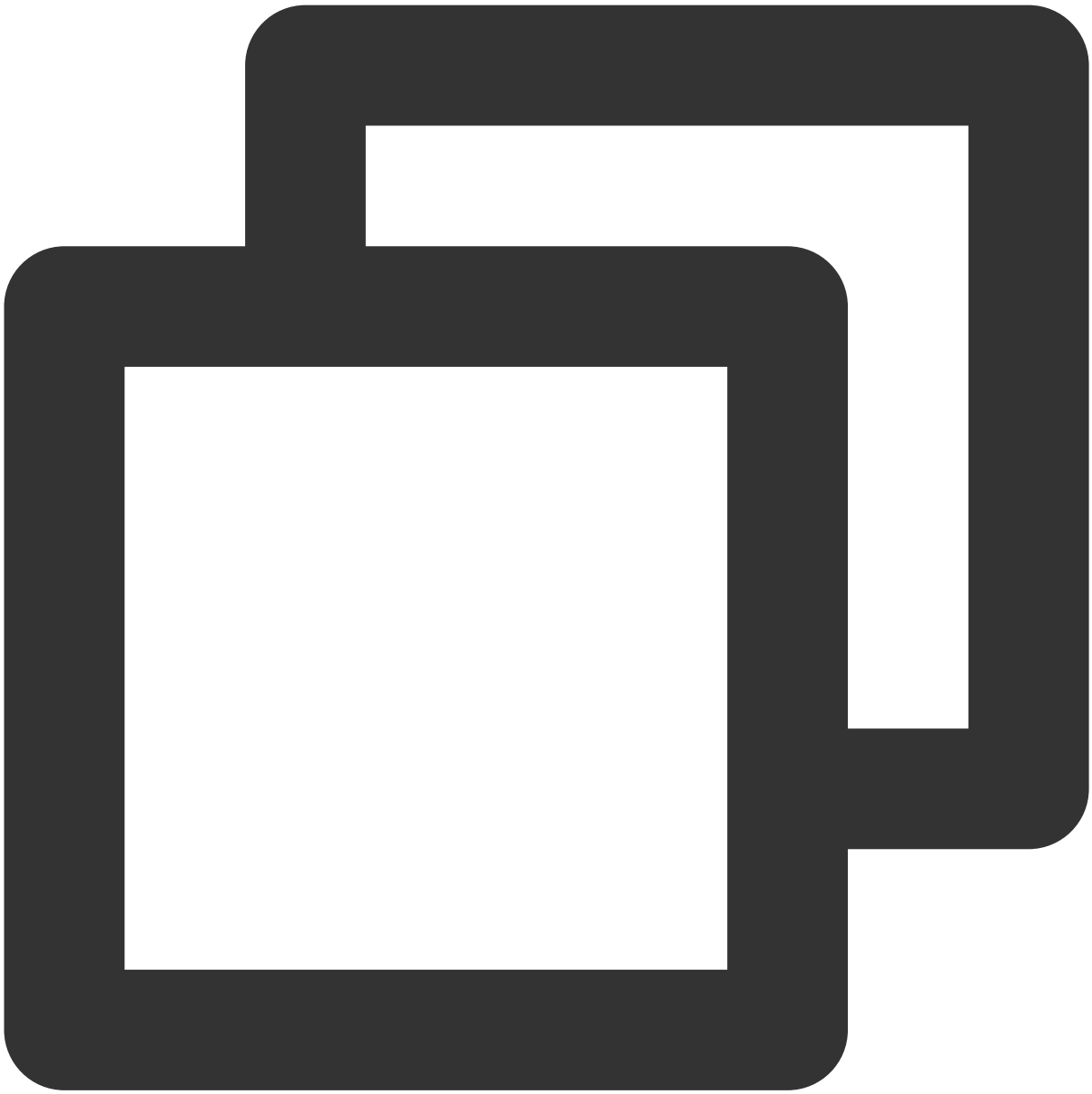
```
<application
    android:usesCleartextTraffic="true" // 增加本行
>
  <!-- possibly other elements -->
</application>
```

集成 Google Firebase Flutter 能力

1. 请打开 pubspec.yaml 文件，添加对 `firebase_core` 的依赖，使用1.12.0版本。

说明：

由于最新版 Google Firebase Flutter 插件最低支持的Dart版本为2.16.0，此处限制为2022年3月发布的1.12.0版本。

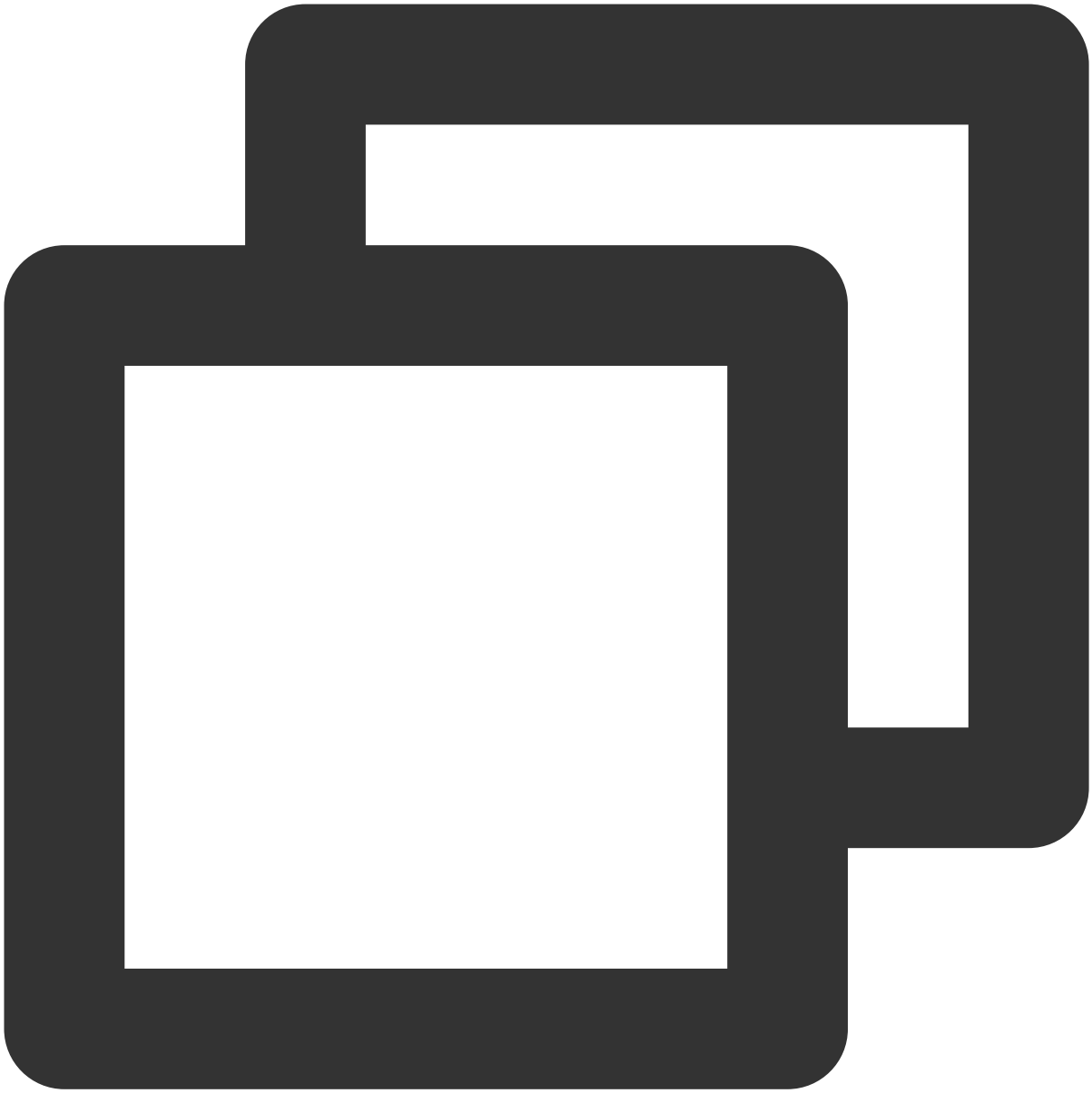


```
dependencies:
  firebase_core: 1.12.0
```

2. 执行 `flutter pub get` 完成安装。

3. 在控制台内，执行以下命令，结合操作提示，完成配置 Google Firebase Flutter 项目。

详见 Google FlutterFire 官方文档。

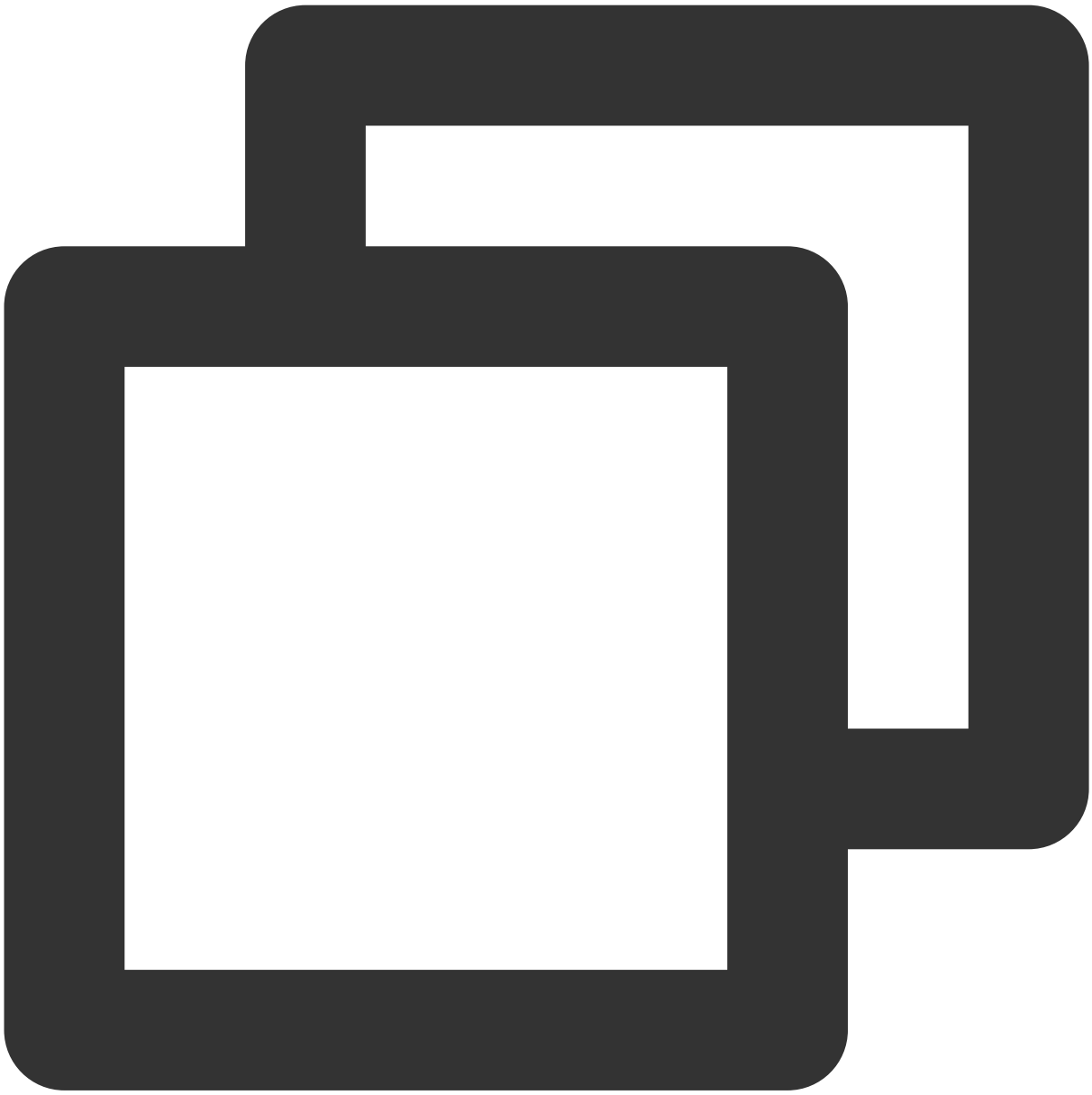


```
// 安装Firebase CLI
npm install -g firebase-tools
curl -sL https://firebase.tools | bash

dart pub global activate flutterfire_cli

// 生成配置文件
flutterfire configure
```

4. 执行该步骤后，会将此项目与您在 Google Firebase 创建的项目关联起来。
`main()` 方法中初始化 FirebaseAPP。



```
WidgetsFlutterBinding.ensureInitialized();

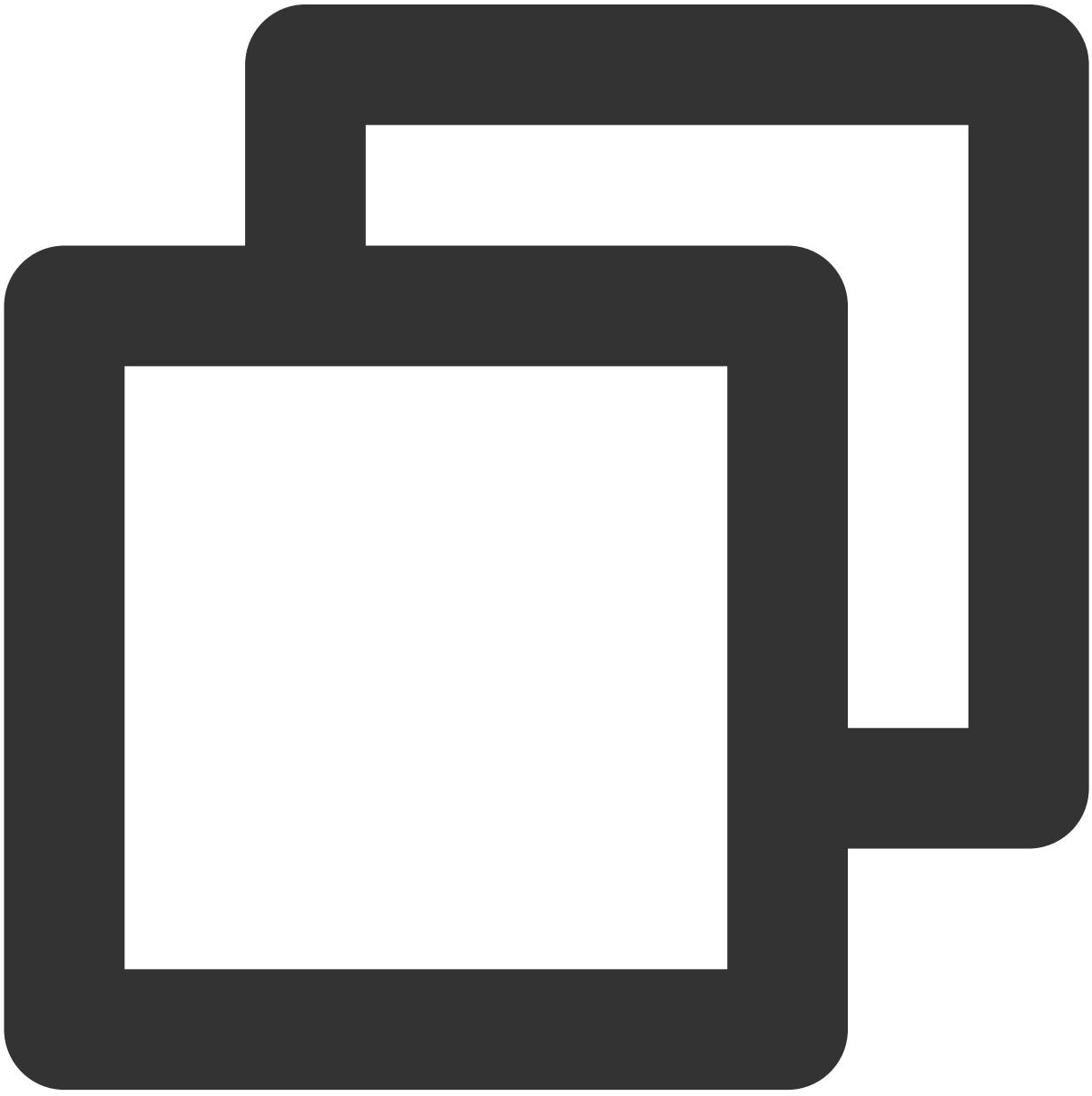
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

不选装 Google FCM 推送

1. 由于国内大部分机型不支持 Google Service，开发者可无需执行此配置。
2. 后续引入插件时，将 `isUseGoogleFCM` 字段设为 false 即可。

华为

1. 打开文件 `android/build.gradle` 。
2. **buildscript>repositories & dependencies**下分别添加华为仓库地址和 HMS gradle 插件依赖：

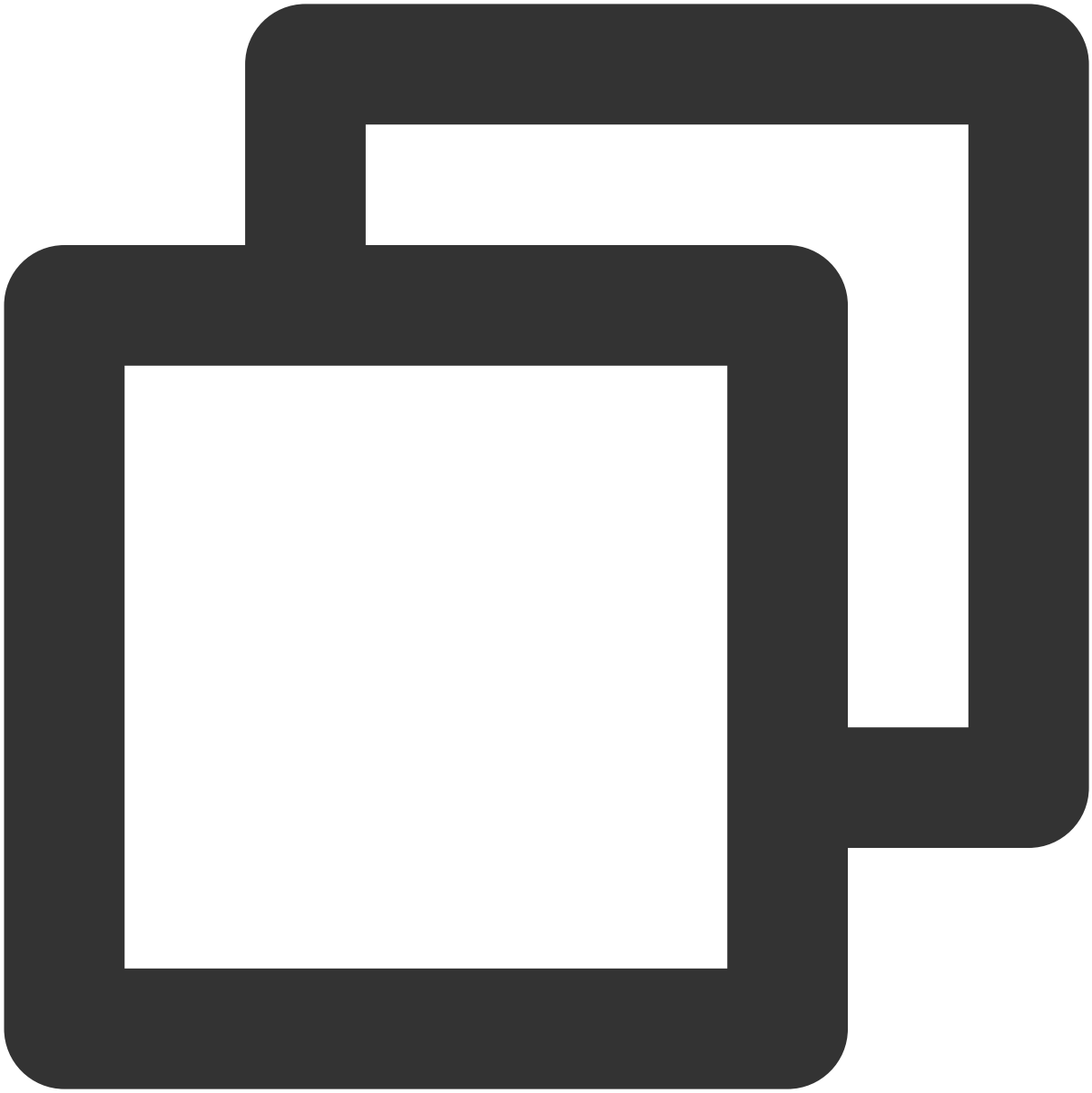


```
buildscript {
 repositories {
     google()
```

```
        jcenter()
        maven {url 'https://developer.huawei.com/repo/'}      // 添加华为 maven 仓库地址
    }
    dependencies {
        // 其他classpath配置
        classpath 'com.huawei.agconnect:agcp:1.3.1.300'      // 添加华为推送 gradle 插件依
    }

    // Set release signing and passwords in the same build configuration file
    signingConfigs {
       release {
           storeFile file('<keystore_file>')
           storePassword '<keystore_password>'
           keyAlias '<key_alias>'
           keyPassword '<key_password>'
       }
   }
   buildTypes {
       // debug模式也要使用证书编译，否则华为指纹验证不通过
       debug {
           signingConfig signingConfigs.release
       }
       release {
           signingConfig signingConfigs.release
       }
    }
   }
```

3. 打开 `android/build.gradle` 文件，在**allprojects>repositories**下添加华为依赖仓库地址：

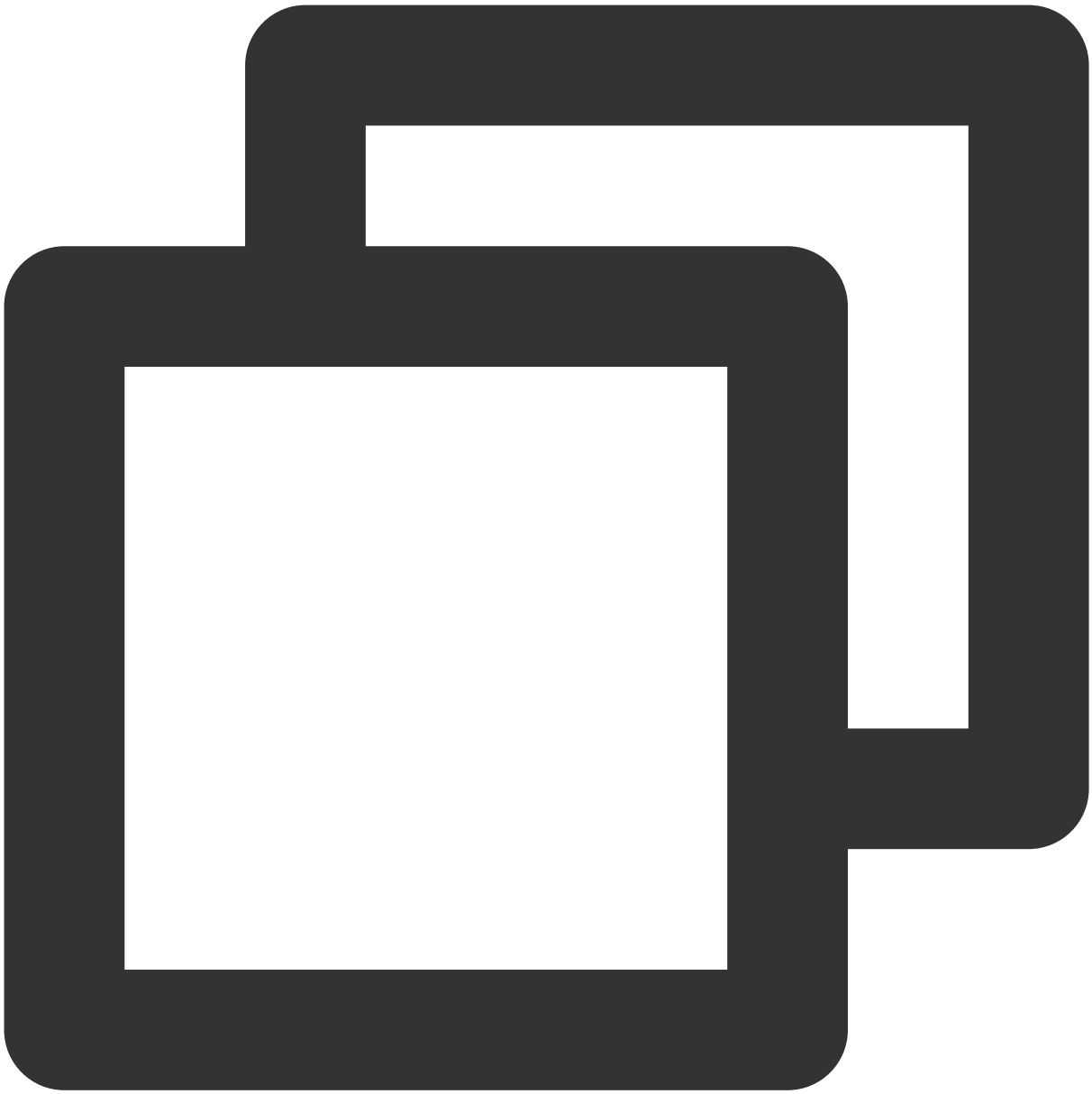


```
allprojects {
 repositories {
     google()
     jcenter()
     maven {url 'https://developer.huawei.com/repo/'}      // 添加华为 maven 仓库地址
 }
}
```

4. 登录华为开放平台，进入**我的项目**> 选择项目 > **项目设置**，下载华为应用最新配置文件 agconnect-services.json。放置于 `android/app` 目录下。

应用层引入 HMS SDK gradle 插件

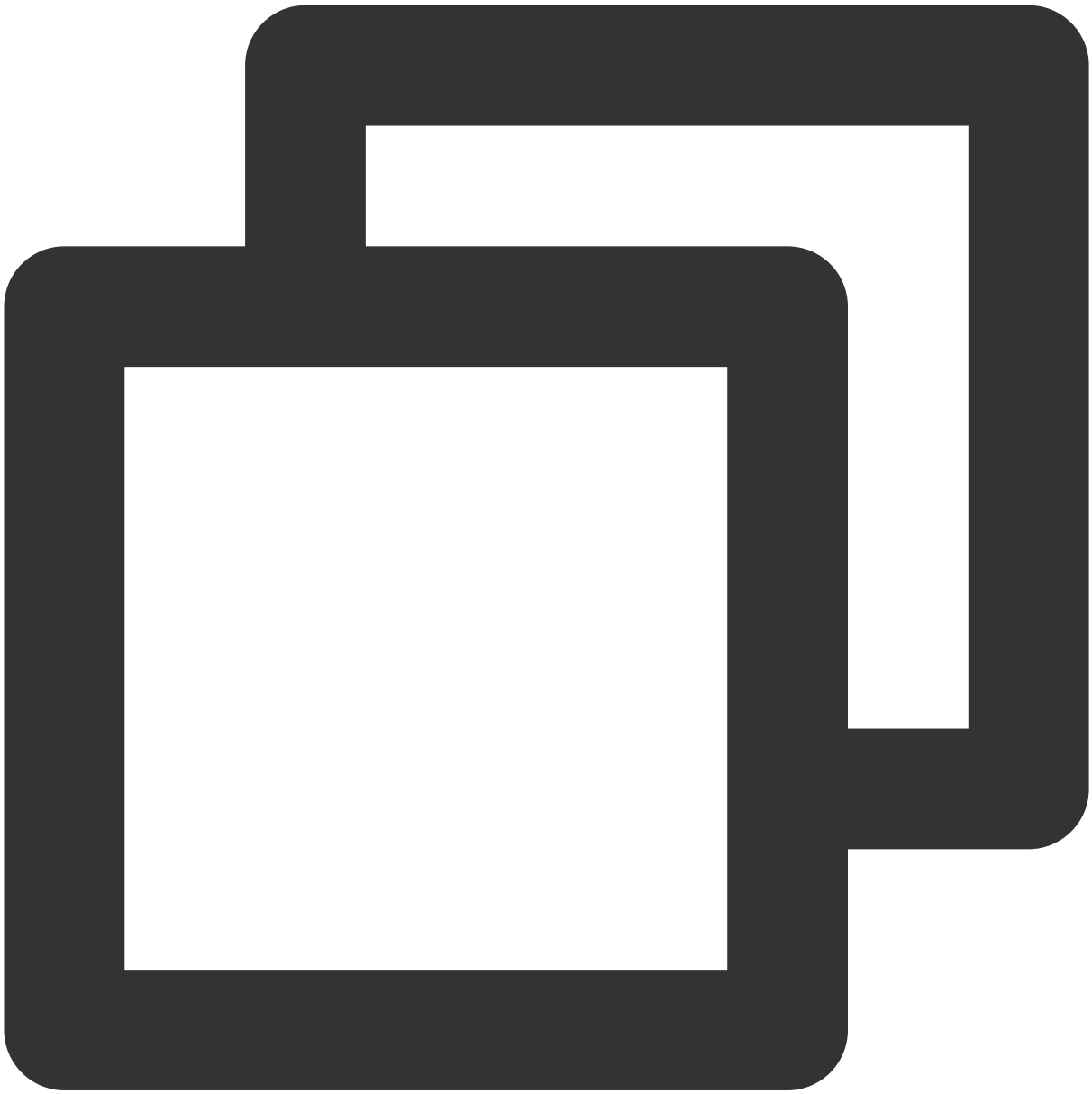
打开 `android/app/build.gradle` 文件，添加以下配置：



```
// app 其他 gradle 插件
apply plugin: 'com.huawei.agconnect'      // HMS SDK gradle 插件
android {
    // app 配置内容
}
```

华为/新荣耀推送角标权限

打开 `android/app/src/main/AndroidManifest.xml` 文件，如下添加 uses-permission 。

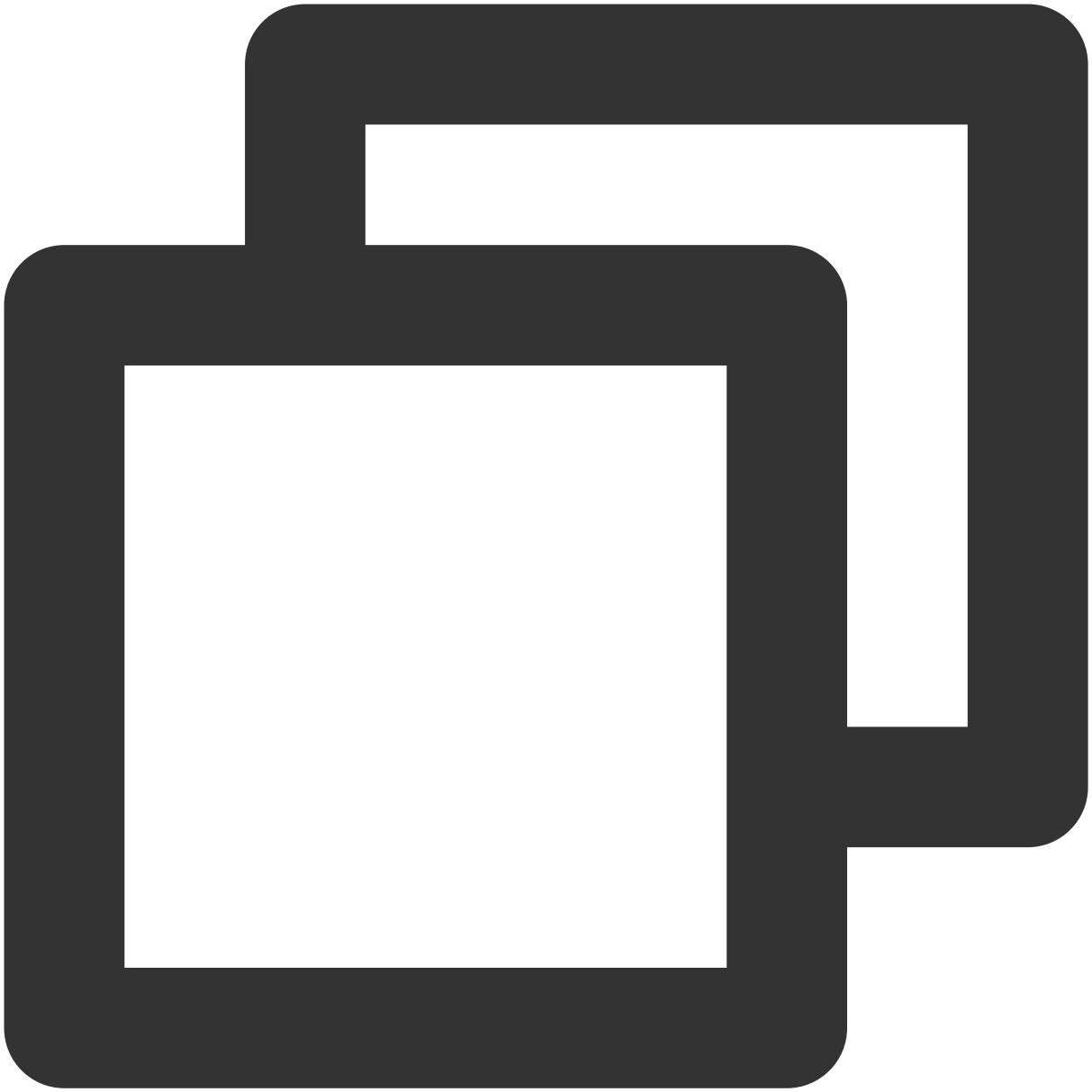


```
<uses-permission android:name = "com.huawei.android.launcher.permission.CHANGE_BADG
<uses-permission android:name = "com.hihonor.android.launcher.permission.CHANGE_BAD
```

vivo

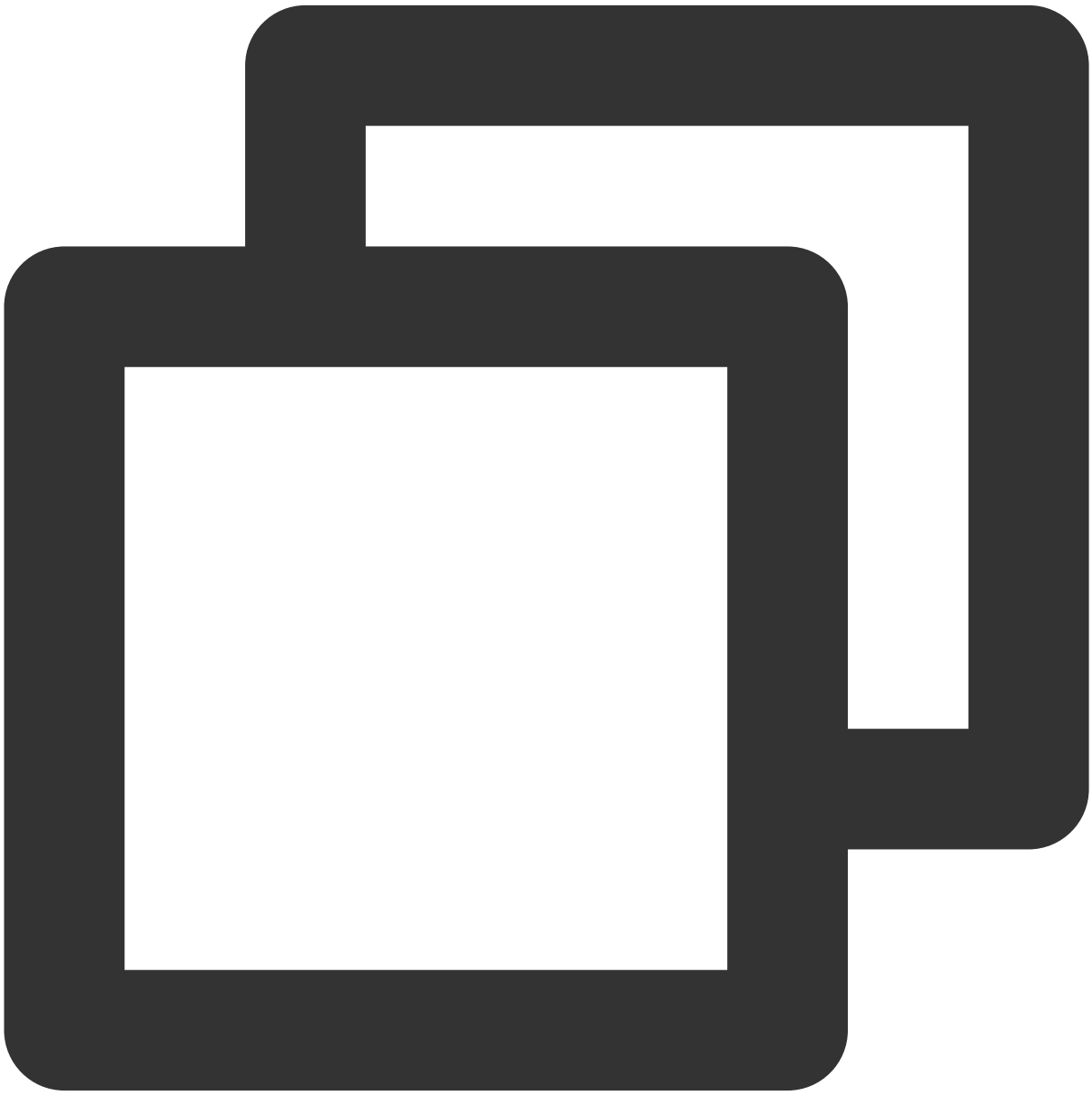
配置 APPID 及 APPKey

打开 `android/app/build.gradle` 文件，如下配置 vivo 的 **APPID** 和 **App_Key**。



```
android: {
  defaultConfig {
    manifestPlaceholders = [
      ....
      vivo_APPID: "vivo的APPID"
      vivo_APPKEY:"vivo的APP_Key",
      .....
    ]
  }
}
```

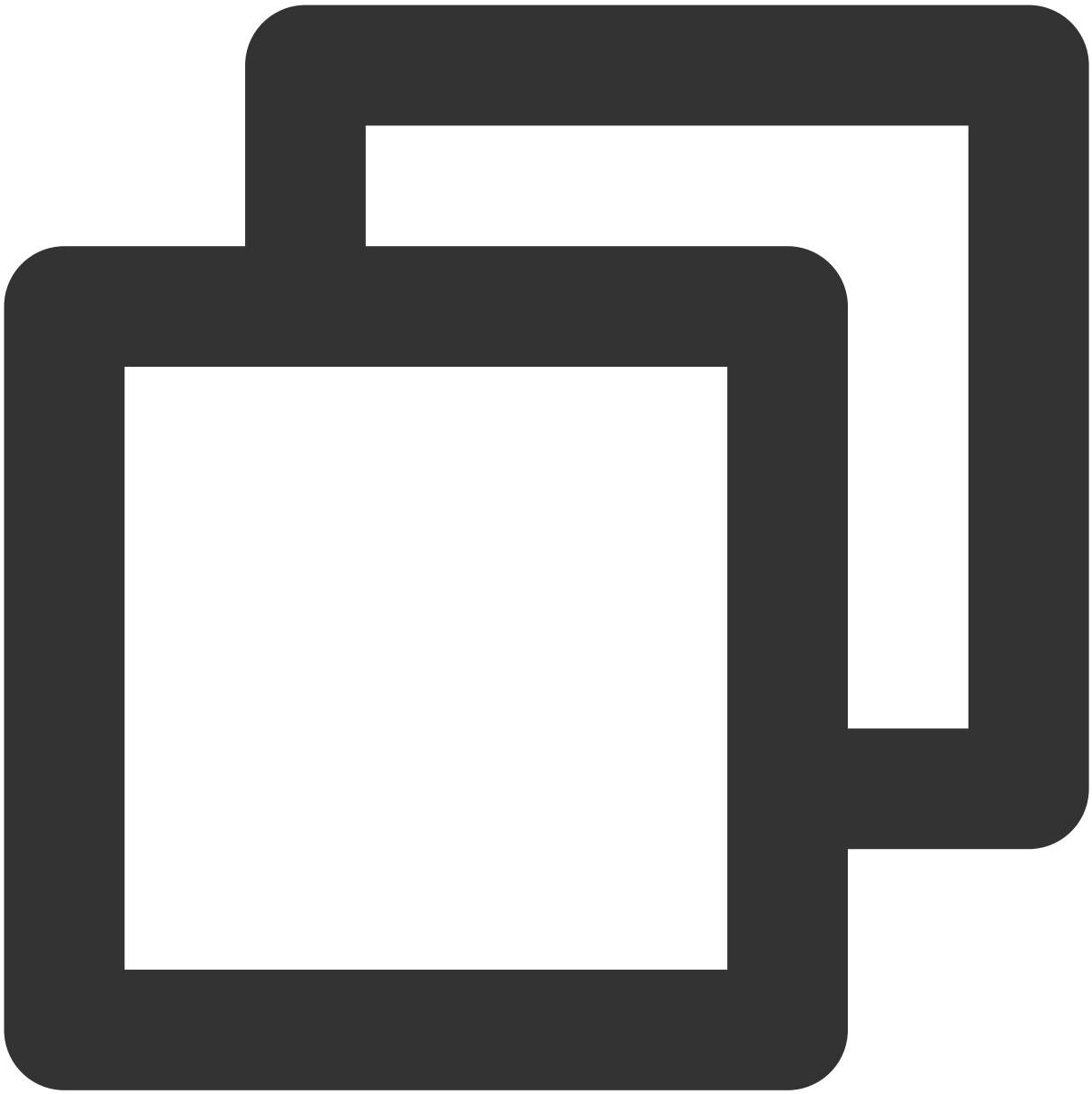
打开 `android/app/src/main/AndroidManifest.xml` 文件，在 `<application>` 中，如下添加meta-data。



```
    <meta-data
        android:name="com.vivo.push.api_key"
        android:value="填入您申请的vivo API_KEY" />
    <meta-data
        android:name="com.vivo.push.app_id"
        android:value="填入您申请的vivo API_ID" />
</application>
```

VIVO角标权限

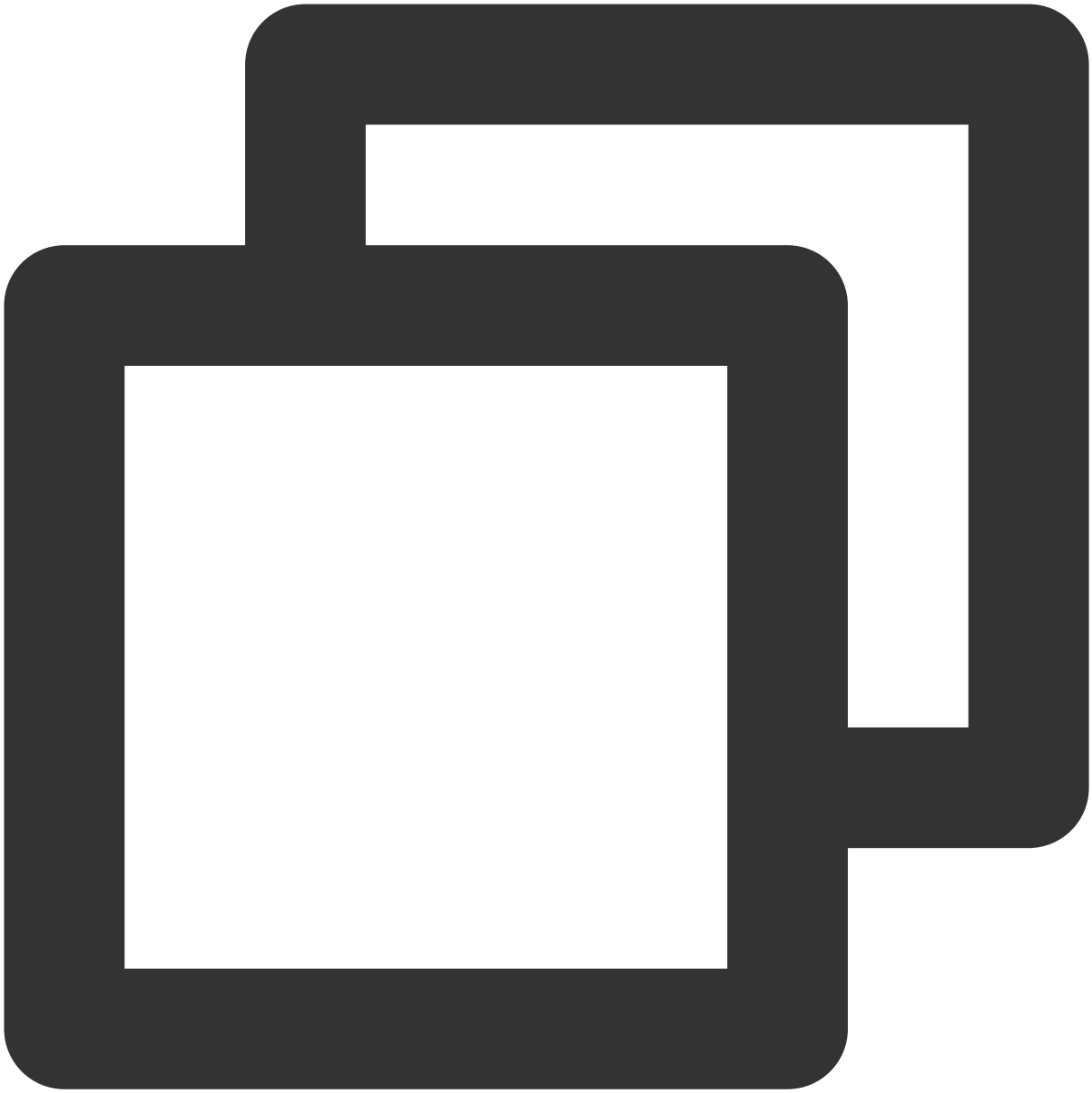
打开 `android/app/src/main/AndroidManifest.xml` 文件，如下添加 uses-permission 。



```
<uses-permission android:name="com.vivo.notification.permission.BADGE_ICON" />
```

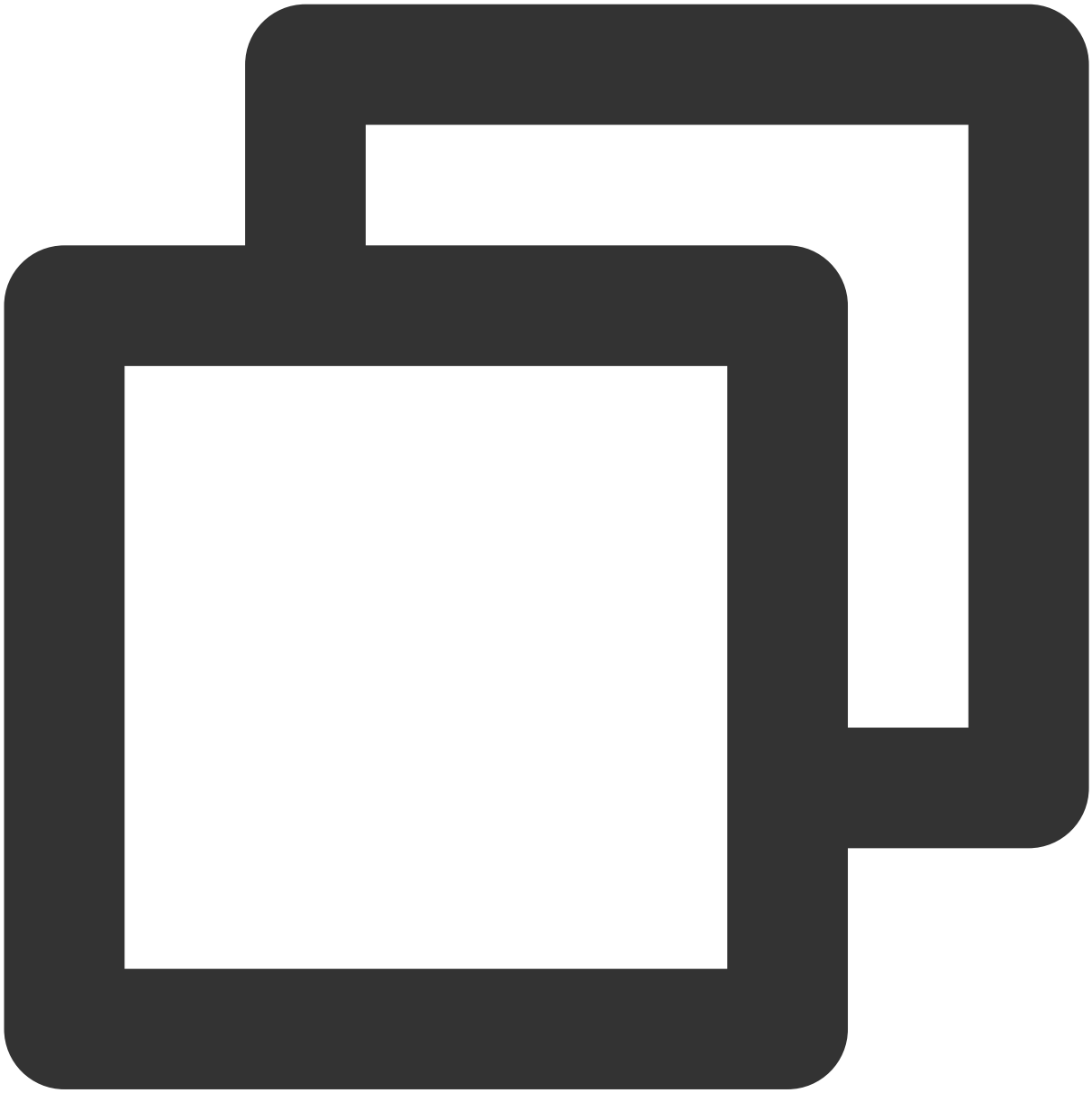
小米/OPPO/魅族

1. 打开 `android/app/build.gradle` 文件，在 `defaultConfig` 中加入包名。



```
defaultConfig {
 applicationId "${替换成您的包名}"
 ...
}
```

2. 打开 `android/app/src/main/AndroidManifest.xml` 文件，配置各厂商权限列表。



```
<!--小米 开始-->
<permission
    android:name="${替换成您的包名}.permission.MIPUSH_RECEIVE"
    android:protectionLevel="signature" />
<uses-permission android:name="${替换成您的包名}.permission.MIPUSH_RECEIVE" />
<!--小米 结束-->

<!--OPPO 开始-->
<uses-permission android:name="com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE" />
<uses-permission android:name="com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE" />
<!--OPPO 结束-->
```

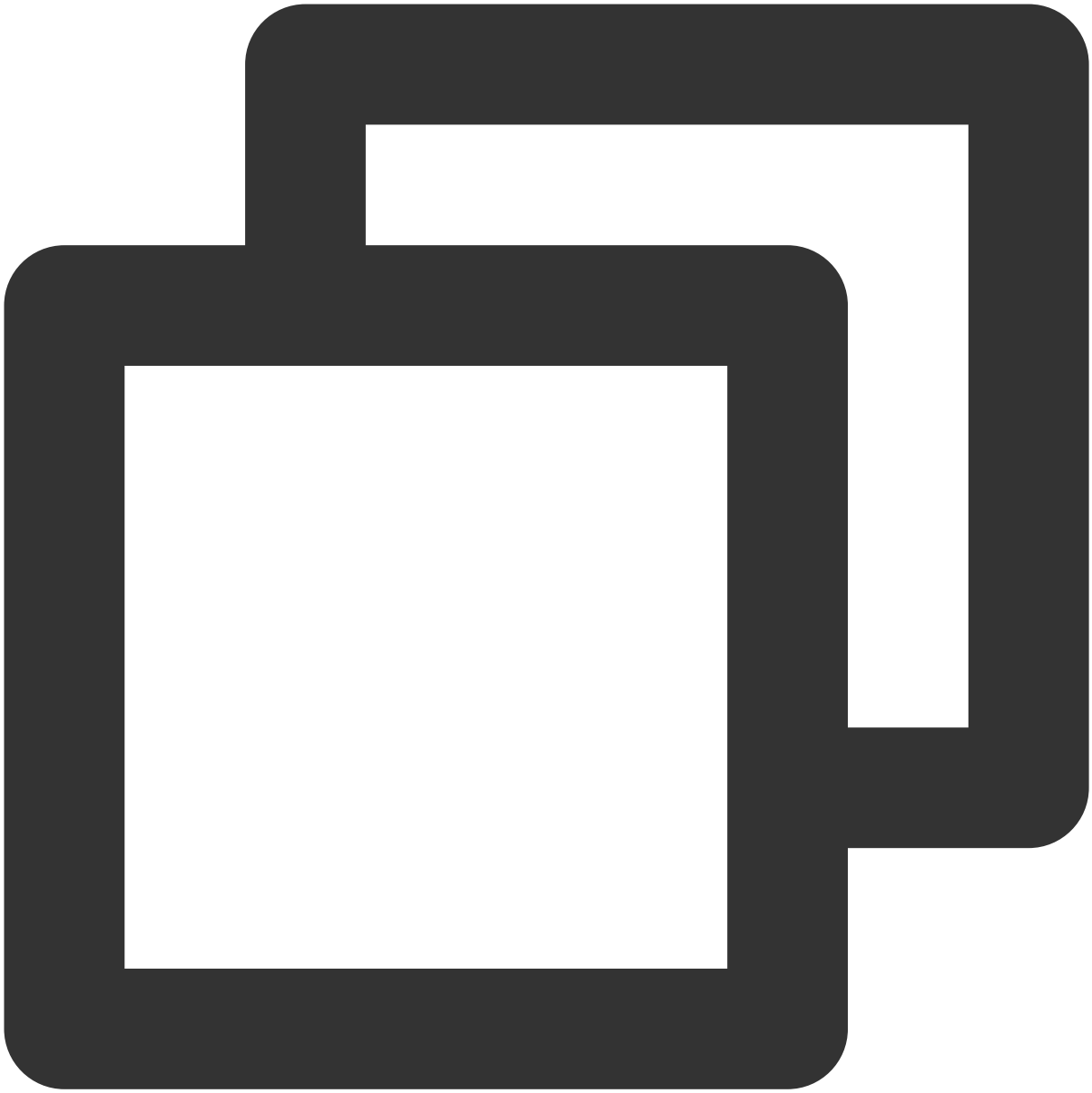
```
<!--魅族 开始-->
<!-- 可选，用于兼容 Flyme5 且推送服务是旧版本的情况-->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!-- 兼容 Flyme5 的权限配置-->
<uses-permission android:name="com.meizu.flyme.push.permission.RECEIVE" />
<permission android:name="${替换成您的包名}.push.permission.MESSAGE"
    android:protectionLevel="signature"/>
<uses-permission android:name="${替换成您的包名}.push.permission.MESSAGE" />
<!-- 兼容 Flyme3 的权限配置-->
<uses-permission android:name="com.meizu.c2dm.permission.RECEIVE" />
<permission android:name="${替换成您的包名}.permission.C2D_MESSAGE" android:protecti
    />
<uses-permission android:name="${替换成您的包名}.permission.C2D_MESSAGE"/>
<!--魅族 结束-->
```

步骤3: 应用启动时初始化

1. 调用插件 `init` 方法。该步骤会完成初始化各厂商通道。
2. 该步骤建议在应用启动后就执行调用。

说明：

由于国内大部分 Android 设备不支持 Google Service, 因此提供一个开关 `isUseGoogleFCM` 供开发者根据主要用户群体判断，是否启用 Google Firebase Cloud Messaging 推送服务。



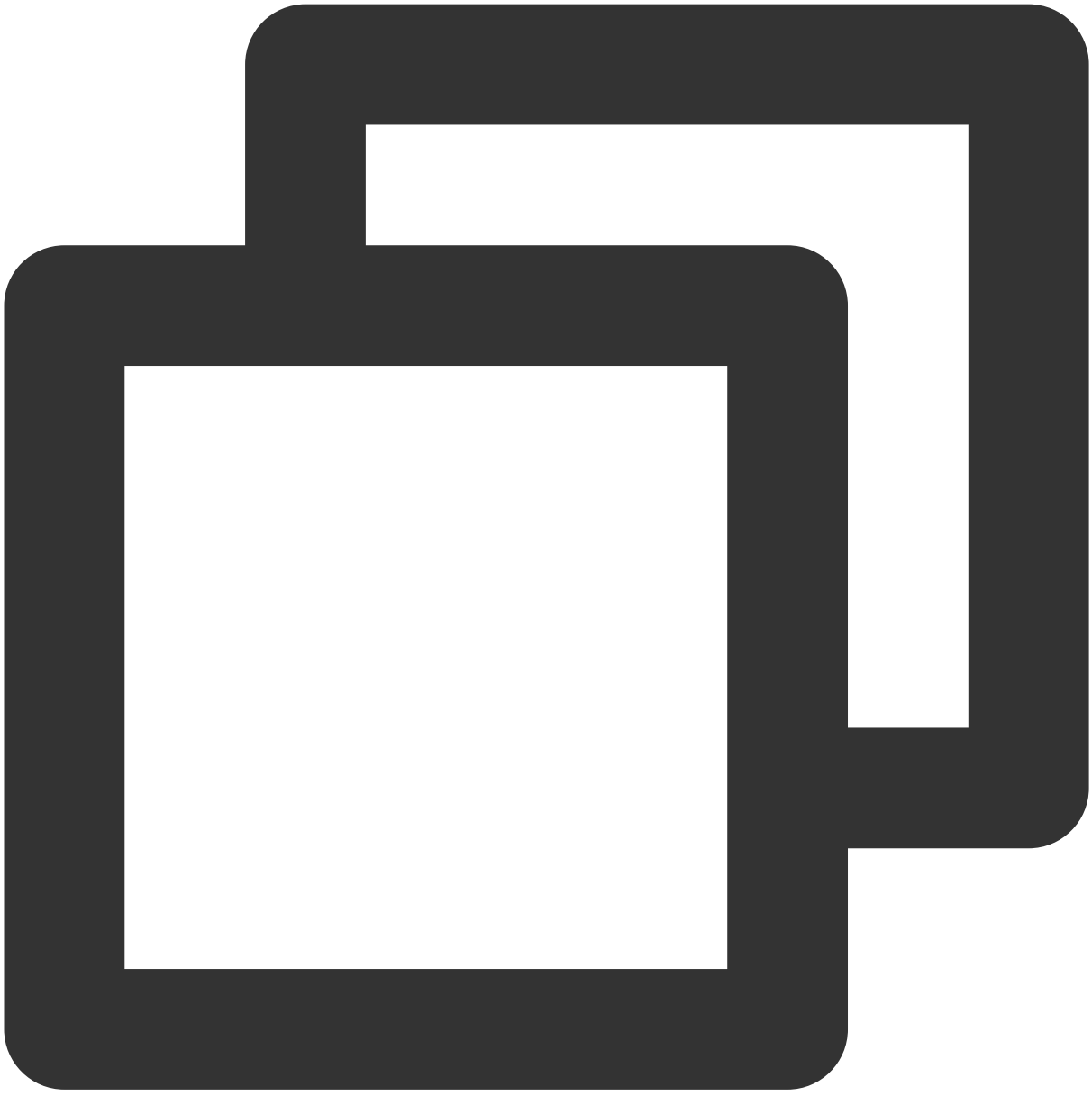
```
import 'package:tim_ui_kit_push_plugin/tim_ui_kit_push_plugin.dart';

final TimUiKitPushPlugin cPush = TimUiKitPushPlugin(
  isUseGoogleFCM: bool, // 是否启用Google Firebase Cloud Messaging，默认true启用
);
cPush.init(
    pushClickAction: pushClickAction, // 单击通知后的事件回调，会在STEP6讲解
    appInfo: PushConfig.appInfo, // 传入STEP1做的appInfo
);
```

3. 初始化结束后，需要为部分厂商创建消息通道，如OPPO和小米均需此配置。调用 `createNotificationChannel` 方法即可。

说明：

如果向厂商申请的 channel ID 一致，同一个 channel ID 调用一次即可。

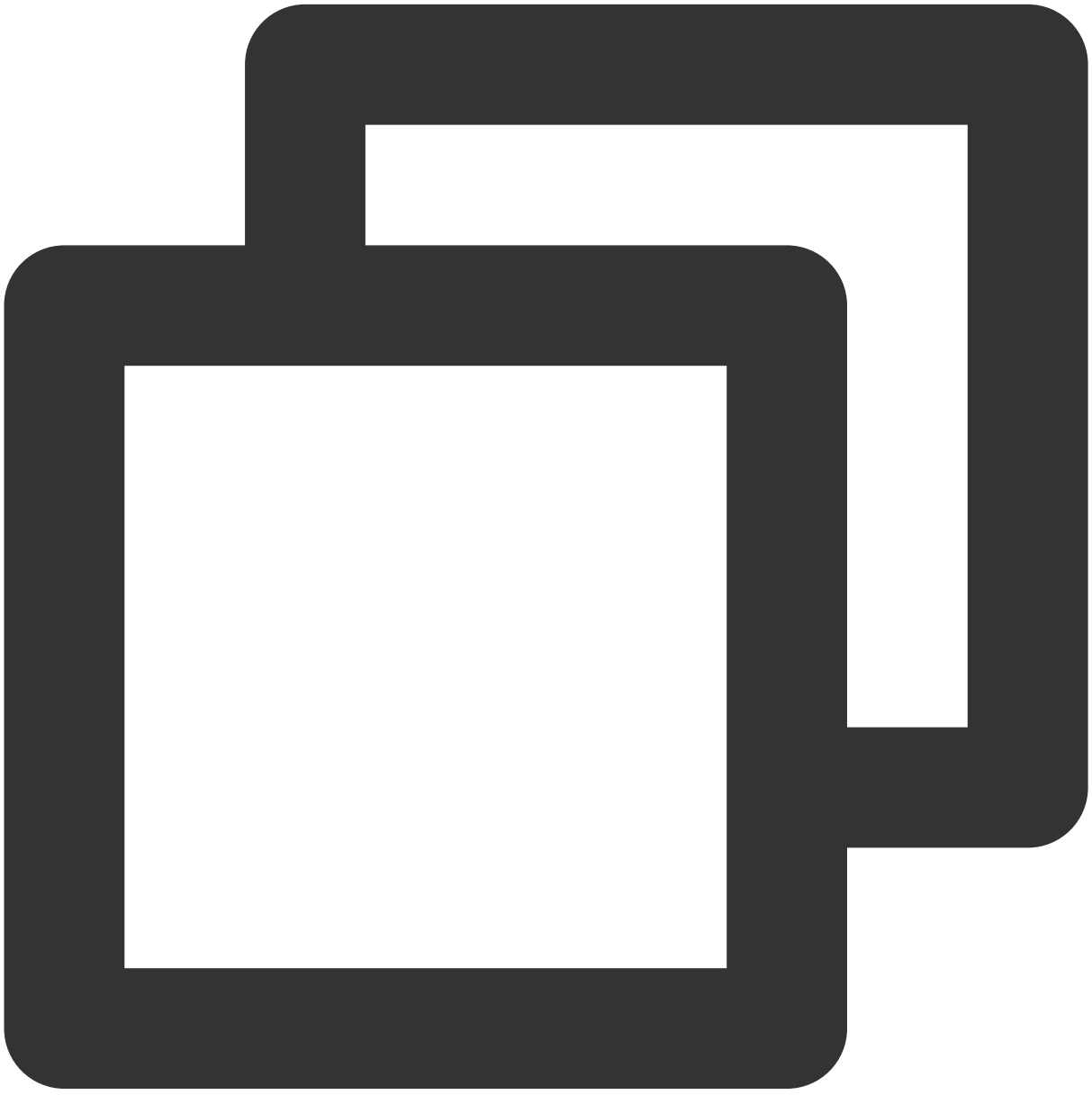


```
cPush.createNotificationChannel(
  channelId: "new_message",
  channelName: "消息推送",
  channelDescription: "推送新聊天消息");
```

4. 部分厂商（如 OPPO）默认不提供推送权限，需要开发者手动申请。调用 `requireNotificationPermission` 方法即可。

说明：

申请权限的时机可由您自行决定，您可以在用户登录成功后再调用。



```
cPush.requireNotificationPermission();
```

步骤4: 上报 Token 及证书 ID

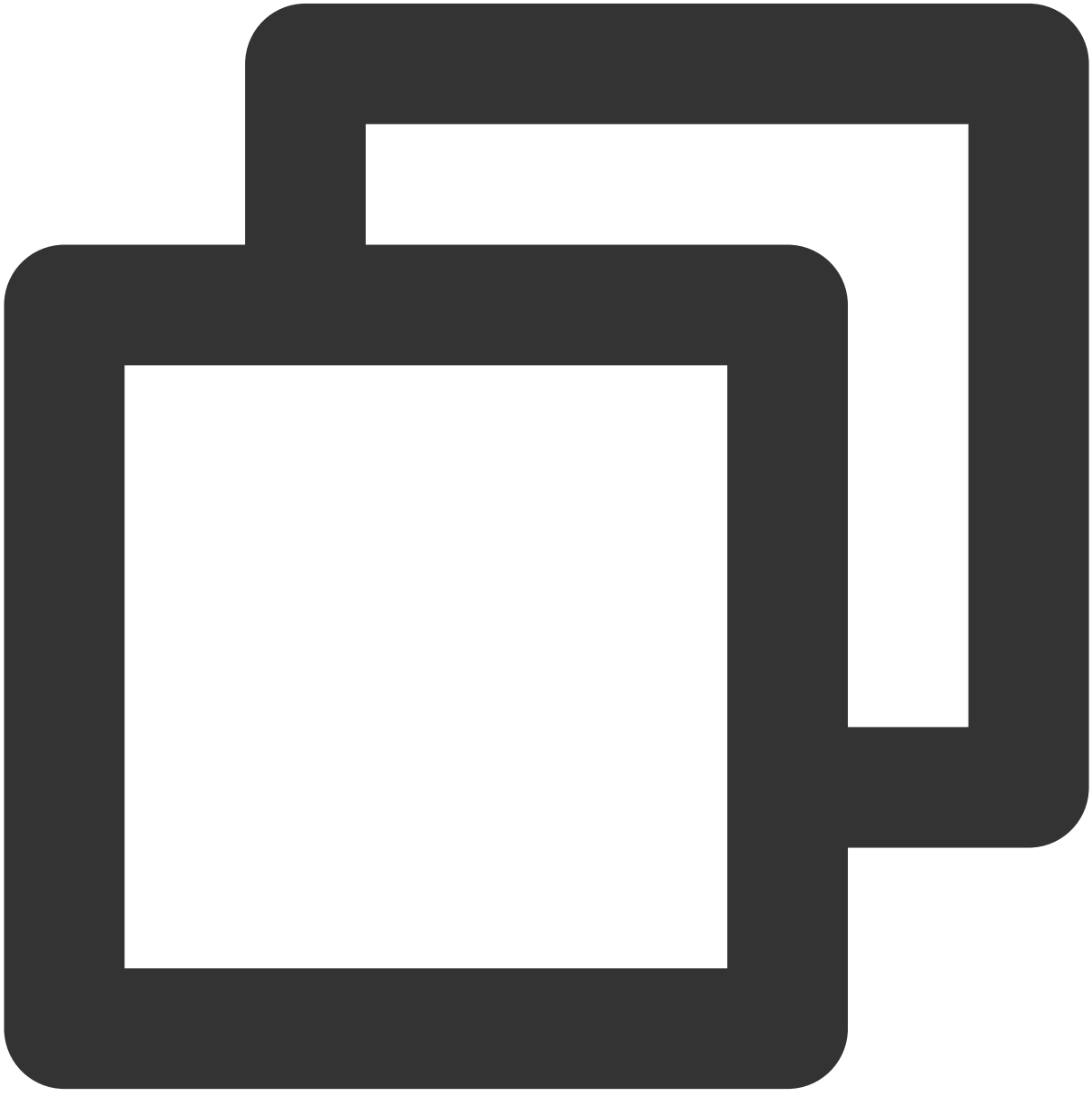
需要将当前设备对应厂商的证书 ID 及 Device Token 上报至腾讯云即时通信后台，服务端才可正常使用厂商通道下行通知。

插件支持自动在appInfo内找到当前厂商的证书ID，并自动完成Token上报。

说明：

根据个保法内隐私相关规定，请在用户Login后再调用该方法上报。

Device Token 在同一设备保持一致，仅需在登录时上报一次即可，无需每次启动都上报。



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

final TimUiKitPushPlugin cPush = TimUiKitPushPlugin(
```

```
    isUseGoogleFCM: false,
  );

final bool isUploadSuccess = await cPush.uploadToken(PushConfig.appInfo);
```

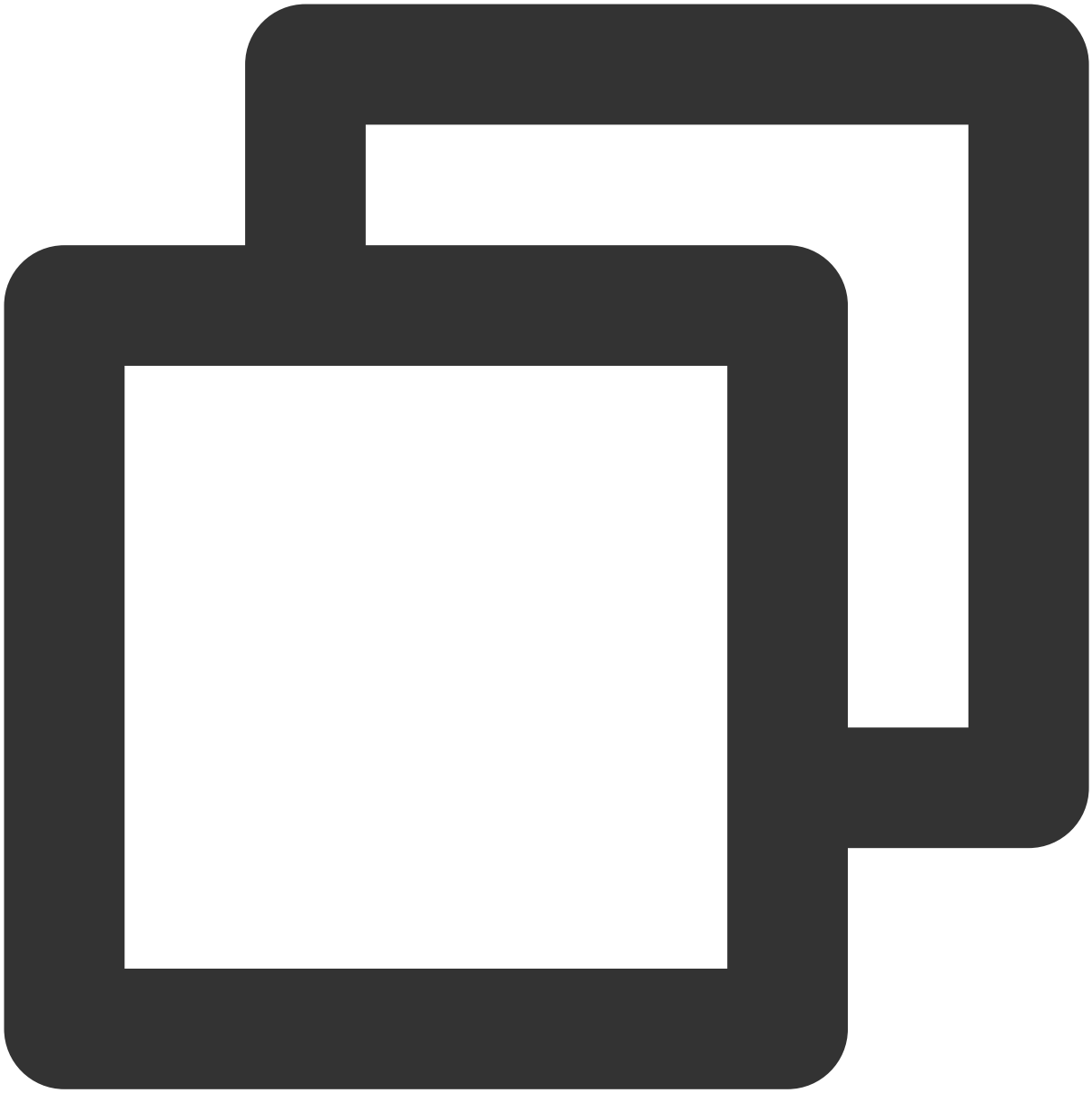
步骤5: 前后台切换监听

1. 需要在每次切换前后台时，通过 IM SDK 上报 IM 后端当前状态。
2. 若为前台在线状态，则收到新消息不触发 notification 推送；反之则会进行推送。
3. 具体请参见 Flutter 官方监听前后台切换方案。

建议：在应用切换到 inactive/paused 状态前，使用插件中 `setBadgeNum( int badgeNum )` 方法，将最新未读数同步至桌面角标。iOS角标由IM SDK自动管理，此处本插件支持配置 XIAOMI(MIUI6 - MIUI 11机型), HUAWEI, HONOR, vivo 及 OPPO 设备角标。

说明：

OPPO角标属于OPPO侧高级权益，不默认开放。如需使用，请自行联系OPPO应用推送权益对接人。



```
/// coreInstance
@override
Future<V2TimCallback> setOfflinePushStatus({required AppStatus status, int? totalCo
  if(Platfrom.isIOS){
    return;
  }
  if(status == AppStatus.foreground){
    // 当应用status为前台时，上报doForeground()
    return TencentImSDKPlugin.v2TIMManager
        .getOfflinePushManager()
        .doForeground();
```

```
  }else{
    // 当应用status为后台时，上报doBackground()，并带上未读数
    return TencentImSDKPlugin.v2TIMManager
        .getOfflinePushManager()
        .doBackground(unreadCount: totalCount ?? 0);
  }
}

/// App
final TimUiKitPushPlugin cPush = TimUiKitPushPlugin(
    isUseGoogleFCM: false,
  );

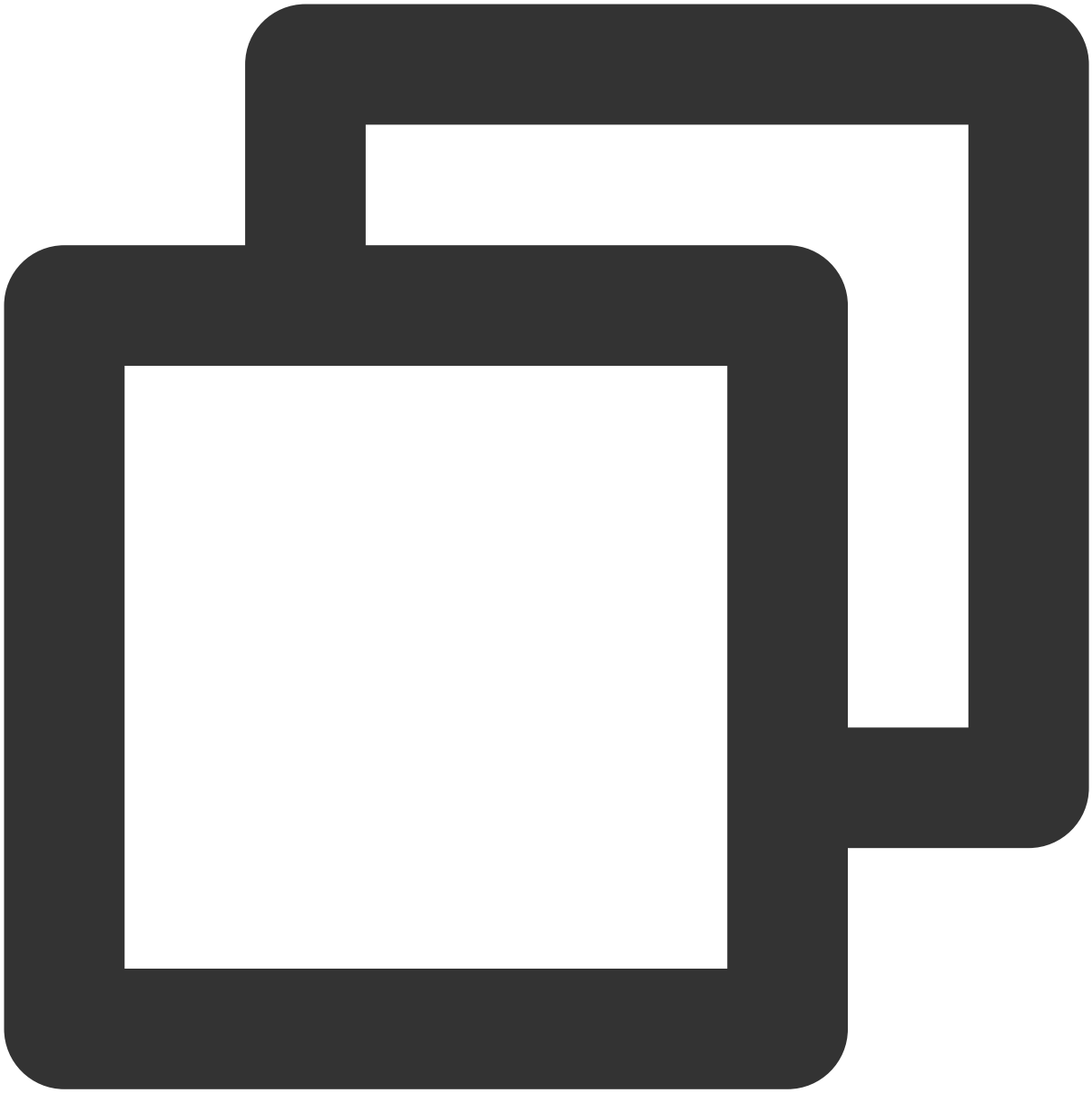
@override
void didChangeAppLifecycleState(AppLifecycleState state) async {
  print("--" + state.toString());
  int? unreadCount = await _getTotalUnreadCount();
  switch (state) {
    case AppLifecycleState.inactive:
      _coreInstance.setOfflinePushStatus(status: AppStatus.background, totalCount:
      if(unreadCount != null){
        cPush.setBadgeNum(unreadCount);
      }
      break;
    case AppLifecycleState.resumed:
      _coreInstance.setOfflinePushStatus(status: AppStatus.foreground);
      break;
    case AppLifecycleState.paused:
      _coreInstance.setOfflinePushStatus(status: AppStatus.background, totalCount:
      if(unreadCount != null){
        cPush.setBadgeNum(unreadCount);
      }
      break;
  }
}
```

步骤6: 发消息配置及单击通知跳转

发送消息

直接通过 SDK 发送

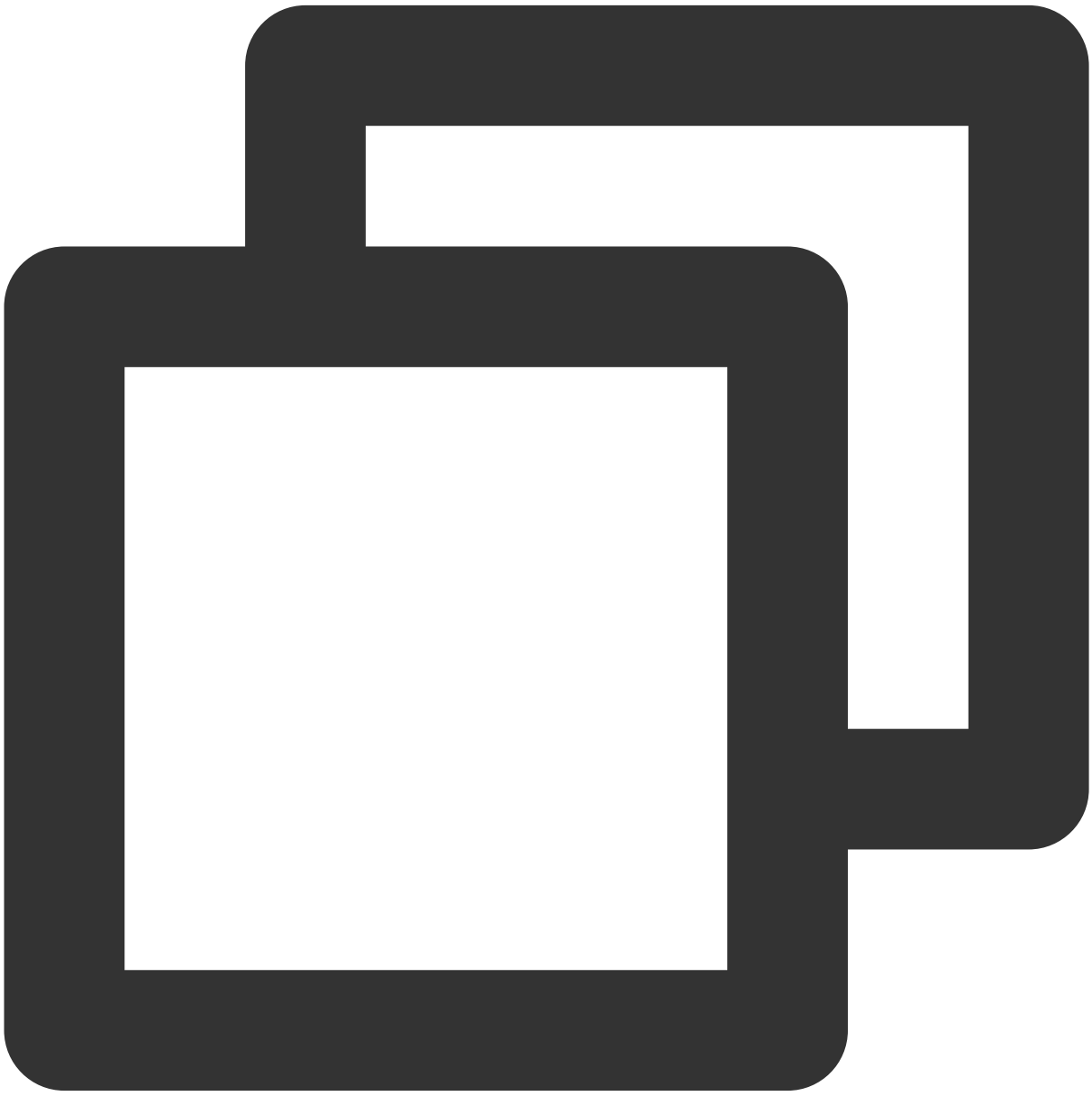
如您自行接入腾讯云 IM SDK，请在发消息时配置 `OfflinePushInfo offlinePushInfo` 字段。



```
OfflinePushInfo({
    this.title = '', // 推送通知标题。留空字符串时，按照优先级，IM后台自动替换成 sender的昵称
    this.desc = '', // 推送第二行小字部分
    this.disablePush = false,
    this.ext = '', // 推送内额外信息，对方可于单击通知跳转时拿到。建议传含Conversation信息的J
    this.androidOPPOChannelID = '', // OPPO的channel ID
  });
```

接入TUIKit

如果您使用我们的 Flutter TUIKit 组件库，可直接在 `TIMUIKitChat` 组件 `TIMUIKitChatConfig` 中，使用 `notificationTitle` / `notificationOPPOChannelID` / `notificationBody` / `notificationExt` / `notificationIOSSound` 定义自定义推送。详情如下：



```
TIMUIKitChat(
    config: TIMUIKitChatConfig(
            notificationTitle: "",// 推送通知标题。留空字符串时，按照优先级，IM后台自动替换成
            notificationOPPOChannelID: "", // 用于推送消息的OPPO配置Channel ID
            notificationBody: (V2TimMessage message, String convID, ConvType convTy
              return "您根据给出的参数自定义的第二行通知";
            },
```

```
            notificationExt: (V2TimMessage message, String convID, ConvType convTyp
              // 您根据给出的参数自定义的EXT字段：此处建议传conversation id，JSON格式，即如
              String createJSON(String convID){
                return "{\\"conversationID\\": \\"$convID\\"}";
              }
              String ext =  (convType == ConvType.c2c
                  ? createJSON("c2c_${message.sender}")
                  : createJSON("group_$convID"));
              return ext;
            }
        )
)
```

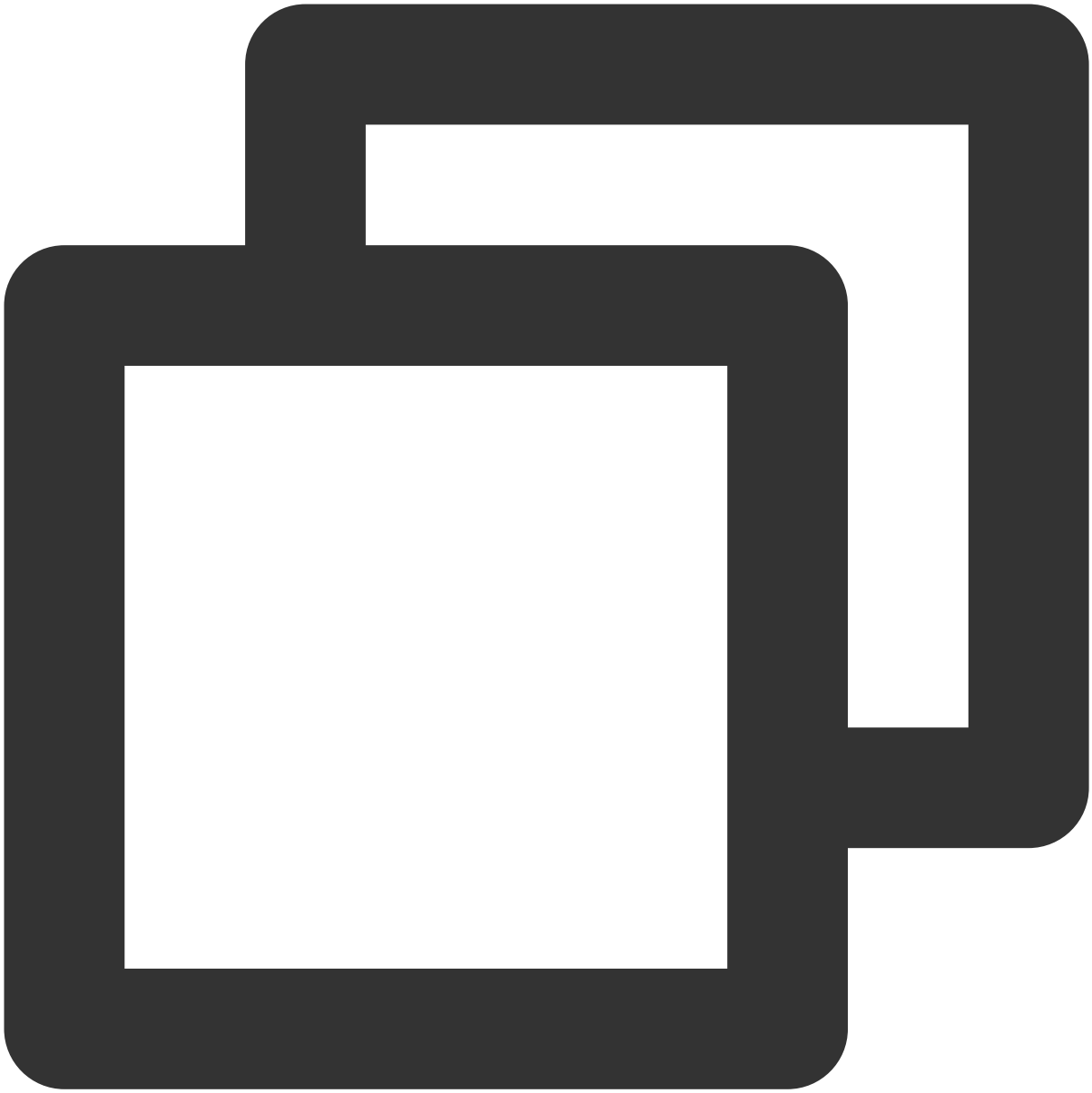
处理单击回调

1. 此时填上 步骤3 初始化时，为 pushClickAction 埋的坑。
2. 初始化时，注册该回调方法，可拿到含推送本体及 ext 信息在内的 Map。
3. 如果上一步创建 OfflinePushInfo 时，在 ext 内传入了含 conversationID 的 JSON，此时即可直接跳转到对应 Chat。

说明：

在后台跳转情况下，此时 Flutter 首页可能已经 unmounted，无法为跳转提供 context，因此建议启动时缓存一个 context，保证跳转成功。

建议跳转成功后，清除通知栏中其他通知消息，避免太多IM消息堆积在通知栏中。调用插件中 `clearAllNotification()` 方法即可。



```
BuildContext? _cachedContext;
final TimUiKitPushPlugin cPush = TimUiKitPushPlugin(
      isUseGoogleFCM: false,
    );
// 仅限TUIKit
final TIMUIKitChatController _timuiKitChatController =
  TIMUIKitChatController();

@override
void initState() {
  super.initState();
```

```
  _cachedContext = context;
}

void handleClickNotification(Map<String, dynamic> msg) async {
    String ext = msg['ext'] ?? "";
    Map<String, dynamic> extMsp = jsonDecode(ext);
    String convId = extMsp["conversationID"] ?? "";

    // 【TUIKit】若当前的会话与要跳转至的会话一致，则不跳转。
    final currentConvID = _timuiKitChatController.getCurrentConversation();
    if(currentConvID == convId.split("_")[1]){
      return;
    }

    final targetConversationRes = await TencentImSDKPlugin.v2TIMManager
        .getConversationManager()
        .getConversation(conversationID: convId);

    V2TimConversation? targetConversation = targetConversationRes.data;

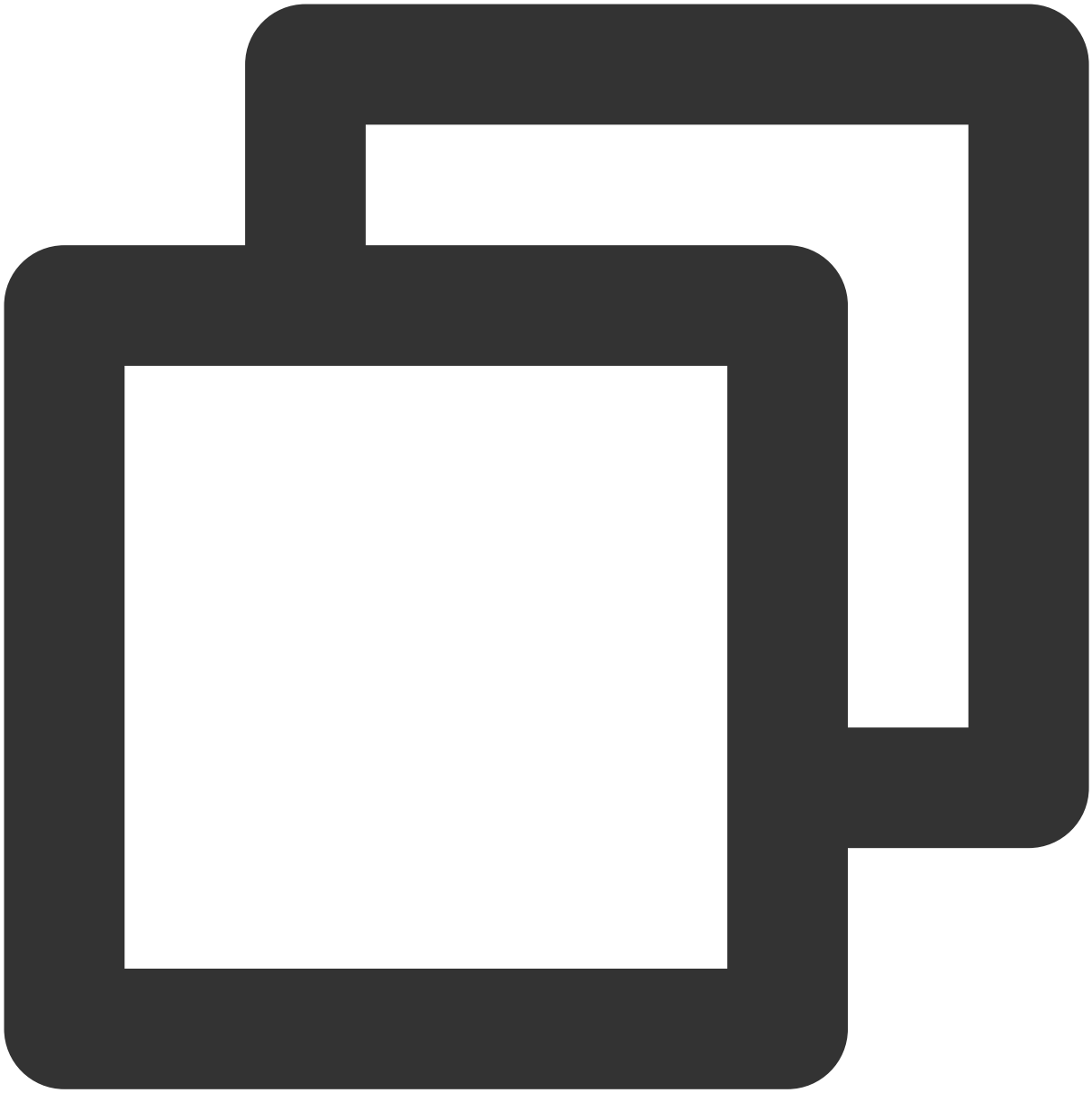
    if(targetConversation != null){
      cPush.clearAllNotification();
      Navigator.push(
          _cachedContext ?? context,
          MaterialPageRoute(
            builder: (context) => Chat(
              selectedConversation: targetConversation,
            ),
          ));
    }
  }
```

步骤7: 使用 TRTC 打单聊语音/视频通话，发送离线推送

一般情况下，发起 TRTC 通话使用信令消息通知对方。您可在信令消息中，按照 步骤6，加入 `offlinePushInfo` 字段。

Flutter 通话插件接入

1. 如果您使用到我们的 tim_ui_kit_calling_plugin 插件，请将其升级至0.2.0版本以上，即可使用离线推送能力。
2. 参见如下示例，直接在 `call` 方法第三个参数中，传入 `offlinePush` 对象即可。



```
final user = await sdkInstance.getLoginUser();
final myId = user.data;
OfflinePushInfo offlinePush = OfflinePushInfo(
  title: "",
  desc: "邀请您语音通话",
  ext: "{\\"conversationID\\": \\"c2c_$myId\\"}",
  disablePush: false,
  ignoreIOSBadge: false,
  androidOPPOChannelID: PushConfig.OPPOChannelID
);
```

```
_calling?.call(widget.selectedConversation.userID!, CallingScenes.Audio, offlinePus
```

说明：

通话群邀请暂不支持离线推送。

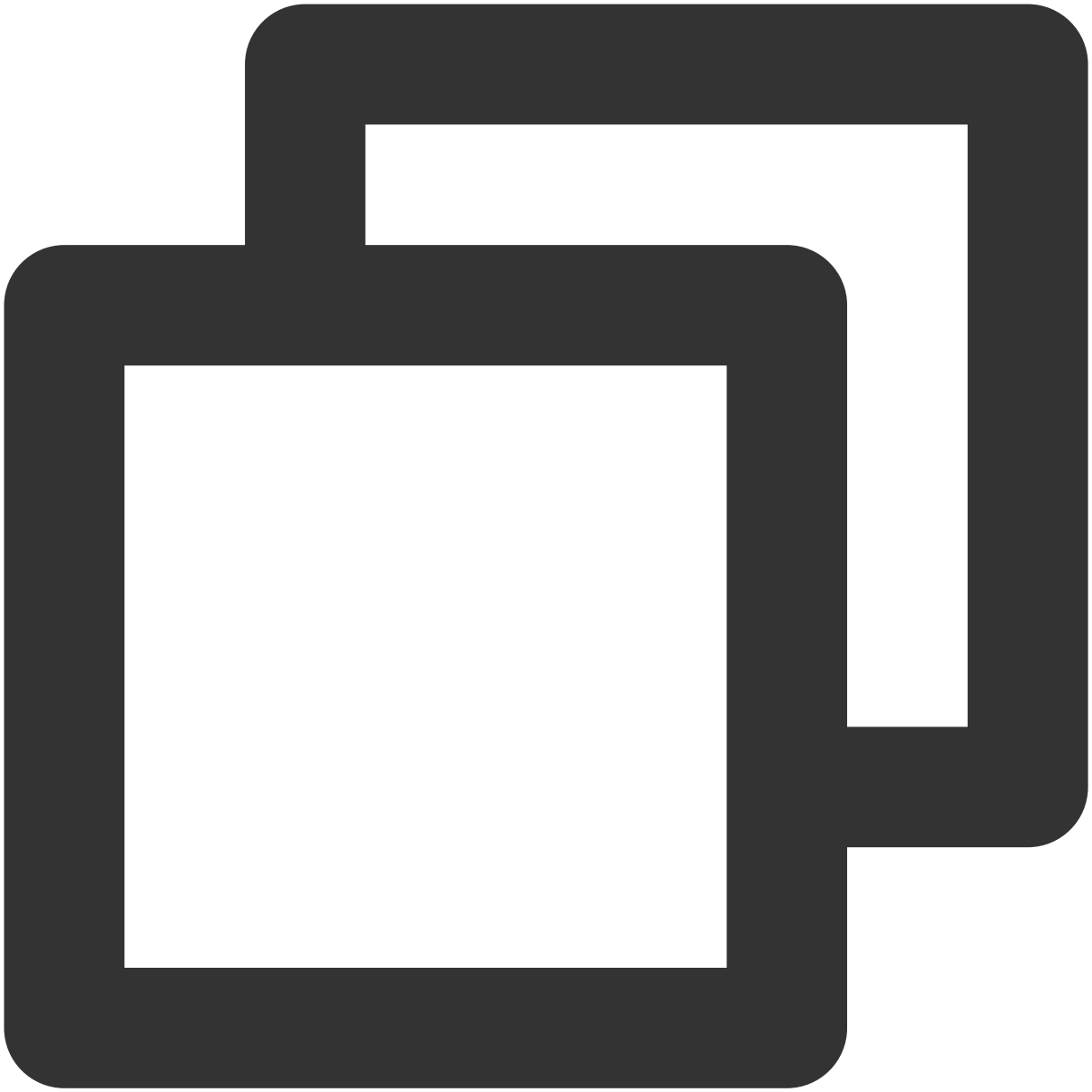
使用插件跑通离线推送（iOS 增补）

本部分在使用插件跑通离线推送（Android）完成的基础上，补充对应步骤 iOS 端需要做的事情。
该页面没有提到过的步骤，和 Android 端一致。

步骤2: 代码中添加 iOS 工程配置

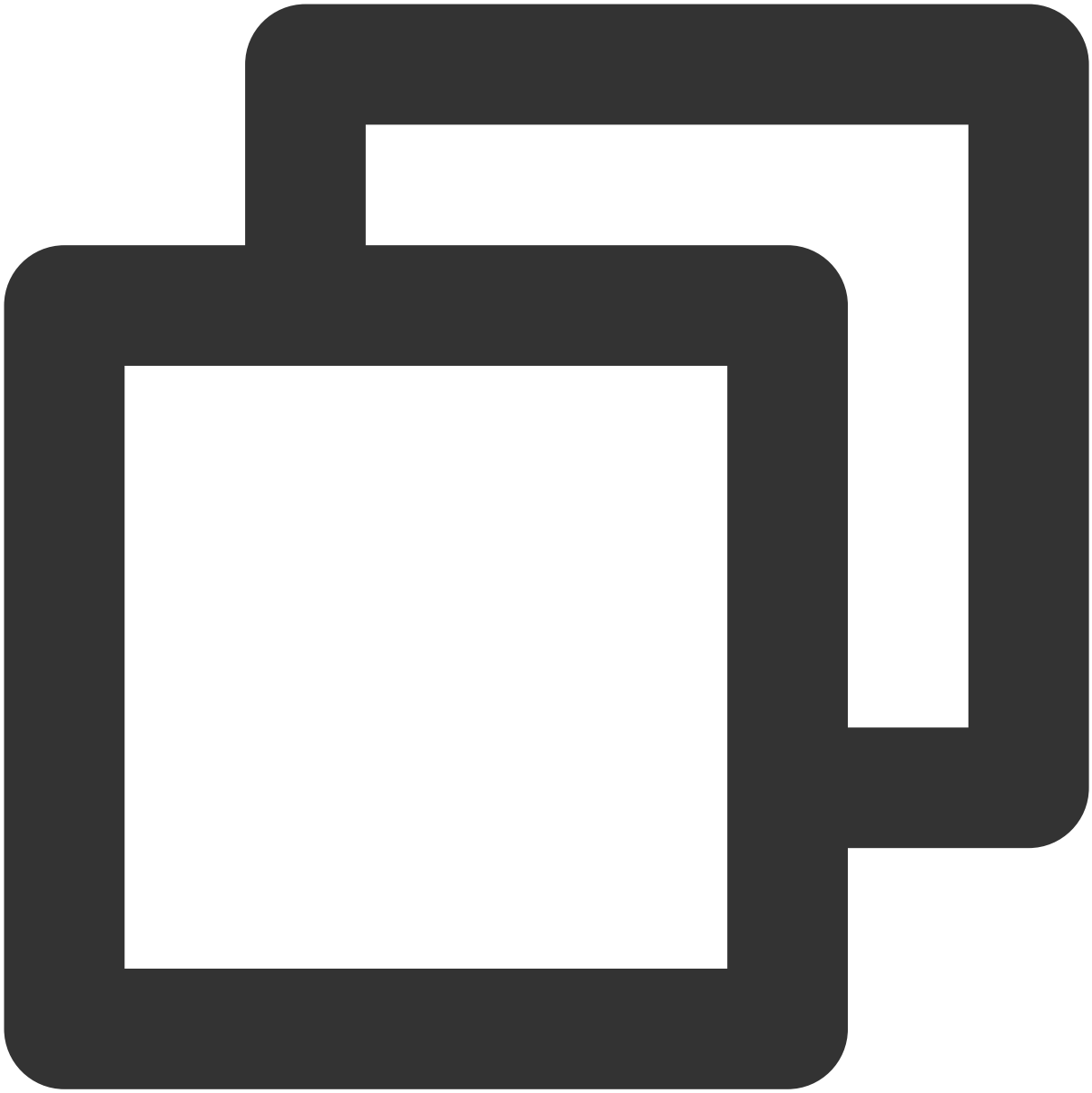
1. 使用 Xcode 打开您的项目，在 **Runner>Target** 中，配置支持 **Push** 的 **Signing Profile**。
2. 并在左上角新增 `Push Notification` 的 Capability。
3. 执行 `flutter pub get` 安装好插件后进入 iOS 目录，执行： `pod install` 安装依赖库。
4. 将以下代码添加到 iOS 工程下 `ios/Runner/AppDelegate.swift` 文件 `didFinishLaunchingWithOptions` 方法中。

Objective-C：



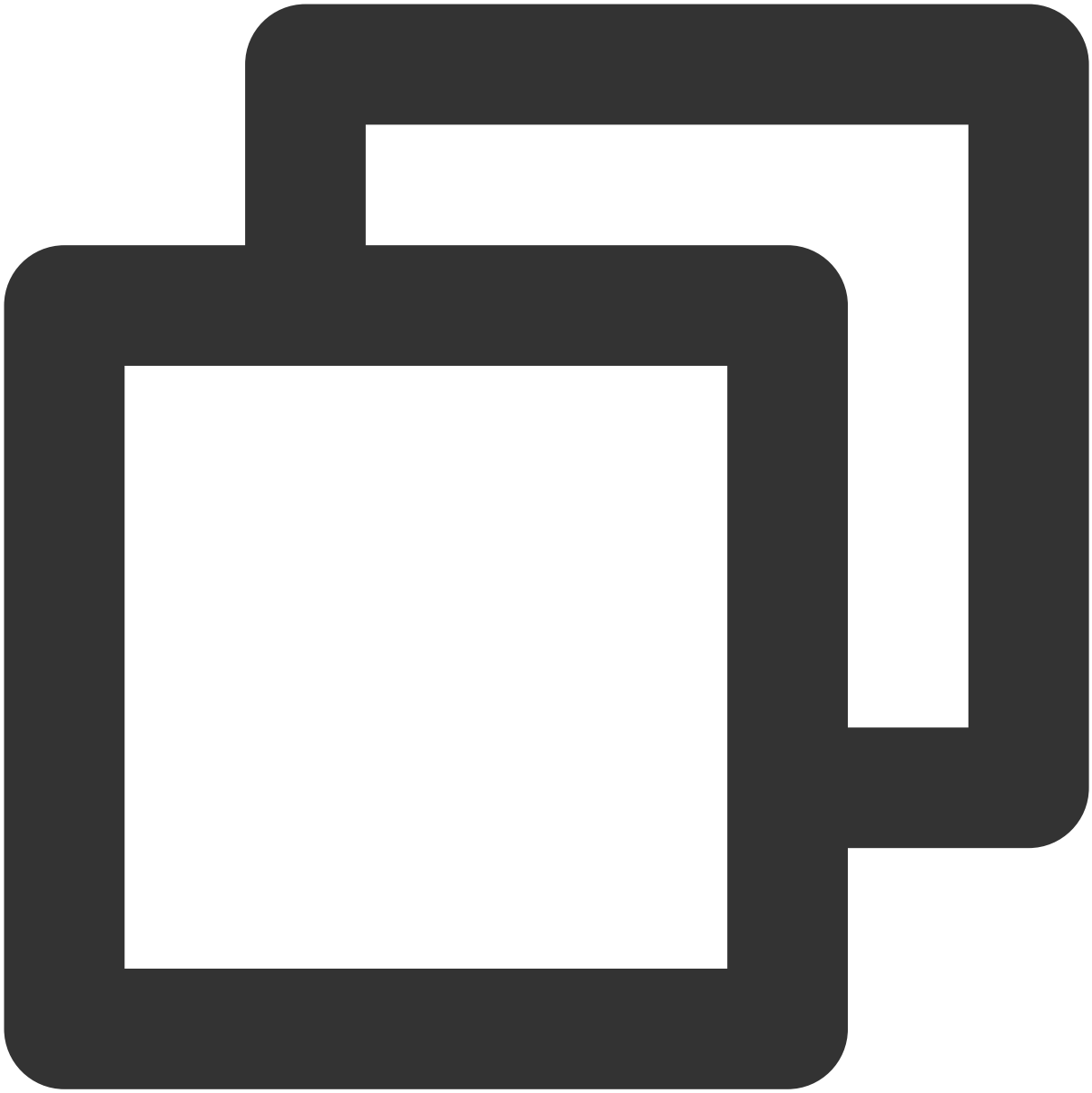
```
if (@available(iOS 10.0, *)) {
  [UNUserNotificationCenter currentNotificationCenter].delegate = (id<UNUserNotific
}
```

Swift：



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

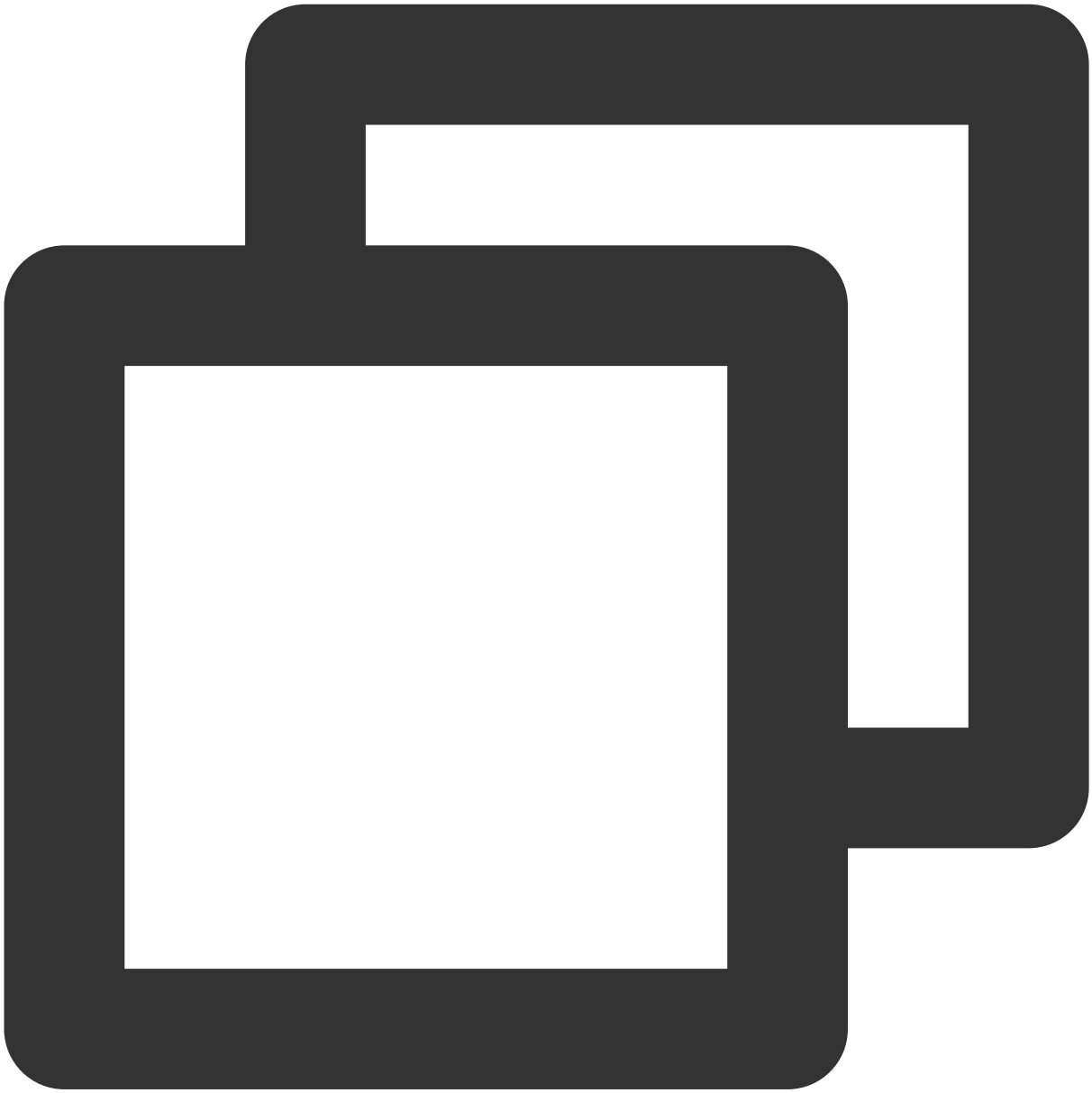
5. 如果不使用 Google Firebase 套件，需要在 `info.plist` 加入如下字段。



```
<key>flutter_apns.disable_firebase_core</key>
<false/>
```

步骤3: 应用启动时初始化

调用插件 `init` 方法。该步骤会完成初始化各厂商通道，并申请厂商通知权限。该步骤建议在应用启动后就执行调用。



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

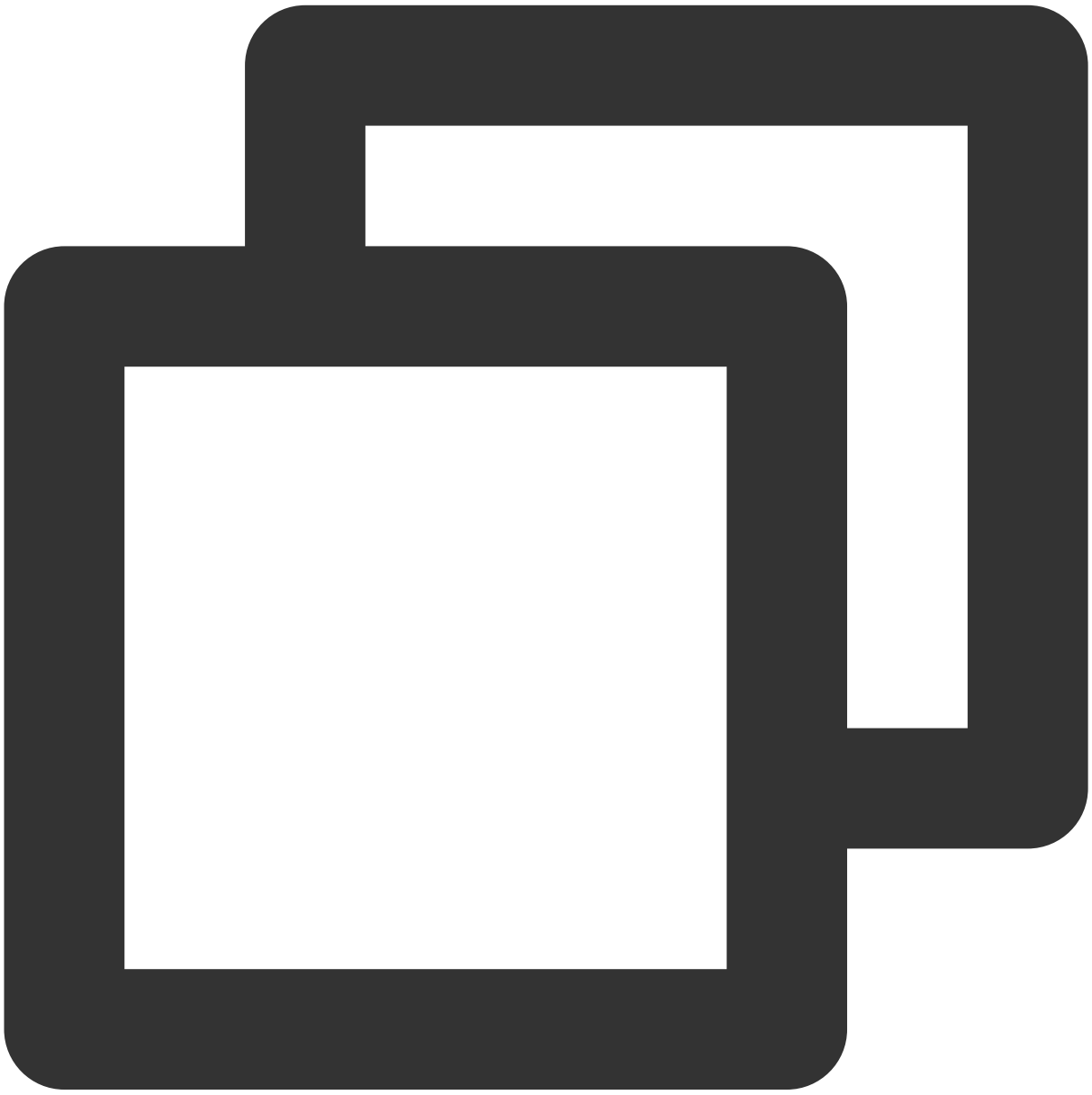
final TimUiKitPushPlugin cPush = TimUiKitPushPlugin();
cPush.init(
    pushClickAction: pushClickAction, // 单击通知后的事件回调，会在STEP6讲解
    appInfo: PushConfig.appInfo, // 传入STEP1做的appInfo
);
```

步骤6: 发消息配置及单击通知跳转

发送消息

直接通过 SDK 发送

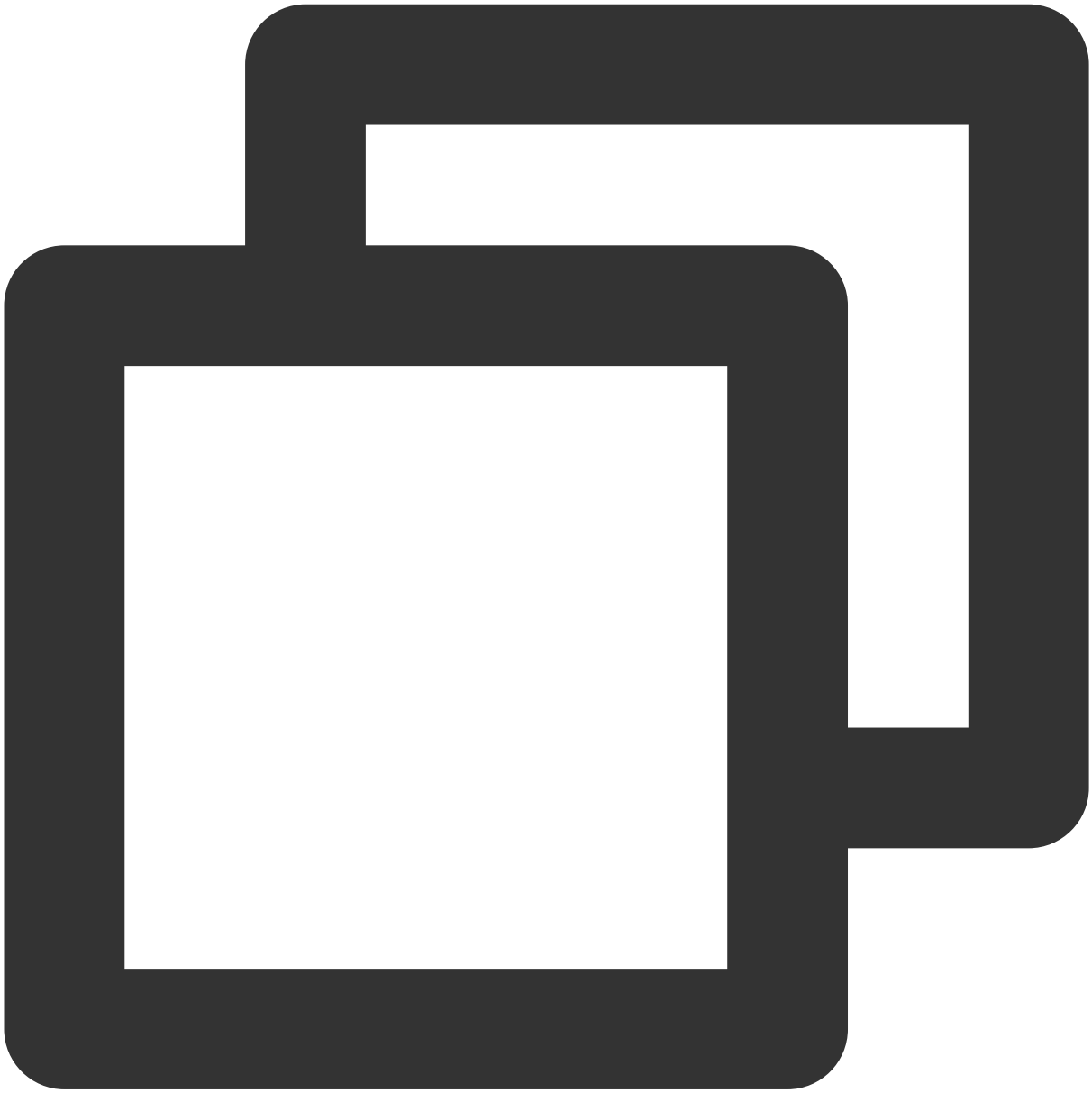
如您自行接入腾讯云 IM SDK，请在发消息时配置 `OfflinePushInfo offlinePushInfo` 字段。



```
OfflinePushInfo({
    // ..其他配置
    this.iOSSound = "", // iOS离线推送声音设置，  当 iOSSound = kIOSOfflinePushNoSound,
    this.ignoreIOSBadge = false,
  });
```

接入TUIKit

如果您使用我们的 Flutter TUIKit 组件库，可直接在 `TIMUIKitChat` 组件 `TIMUIKitChatConfig` 中，使用 `notificationTitle` / `notificationOPPOChannelID` / `notificationBody` / `notificationExt` / `notificationIOSSound` 定义自定义推送。详情如下：



```
TIMUIKitChat(
    config: TIMUIKitChatConfig(
            // ..其他配置
            notificationIOSSound: "", // iOS离线推送声音设置，  当 iOSSound = kIOSOffli
        )
```

```
)
```

调试

离线推送自查

您可使用 离线推送自查 工具，检测终端状态/证书上报及发送测试消息。

vivo 调试

由于 vivo 官方限制，应用在 vivo 应用市场上架前，不允许使用正式 PUSH 能力，详见此文档。
开发过程中，需要调试，请参见本步骤：
1. 获取测试设备（vivo 真机）的 regId（我们称做 Device Token）。
2. 在 vivo 控制台内，添加该设备为测试设备。
3. 此时可推送测试消息至测试设备。可参见 vivo 单播推送文档。
4. 由于腾讯云 IM 控制台的测试推送，和直接使用 IM SDK 发送聊天消息的推送，均不能修改推送模式为测试。因此请使用我们提供的，可触发测试消息的JS脚本，单击此处下载
5. 下载后，请根据顶部五行注释，填入vivo相关参数。默认ext为 `conversationID` ，如果在处理单击回调跳转（可参见 步骤6）时需要其他字段，请自行修改JS代码。
6. 执行脚本。 `npm install axios` `npm install js-md5` 后 `node testvivo` 。推送结果会显示在 log 最后一行。
7. 此时测试终端可收到测试消息推送，单击消息后，可触发 Dart 层回调。

厂商推送限制

1、国内厂商都有消息分类机制，不同类型也会有不同的推送策略。如果想要推送及时可靠，需要按照厂商规则设置自己应用的推送类型为高优先级的系统消息类型或者重要消息类型。反之离线推送消息会受厂商推送消息分类影响，与预期会有差异。
2、另外，一些厂商对于应用每天的推送数量也是有限制的，可以在厂商控制台查看应用每日限制的推送数量。
如果离线推送消息出现推送不及时或者偶尔收不到情况，需要考虑下这里：
华为：将推送消息分为服务与通讯类和资讯营销类，推送效果和策略不同。另外，消息分类还和自分类权益有关：
无自分类权益，推送消息厂商还会进行二次智能分类 。
有申请自分类权益，消息分类会按照自定义的分类进行推送。
具体请参见 厂商描述。
vivo：将推送消息分为系统消息类和运营消息类，推送效果和策略不同。系统消息类型还会进行厂商的智能分类二次修正，若智能分类识别出不是系统消息，会自动修正为运营消息，如果误判可邮件申请反馈。另外，消息推送也受

日推总数量限制，日推送量由应用在厂商订阅数统计决定。
具体请参见 厂商描述1 或 厂商描述2。
OPPO：将推送消息分为私信消息类和公信消息类，推送效果和策略不同。其中私信消息是针对用户有一定关注度，且希望能及时接收的信息，私信通道权益需要邮件申请。公信通道推送数量有限制。
具体请参见 厂商描述1 或 厂商描述2。
小米：将推送消息分为重要消息类和普通消息类，推送效果和策略不同。其中重要消息类型仅允许即时通讯消息、个人关注动态提醒、个人事项提醒、个人订单状态变化、个人财务提醒、个人状态变化、个人资源变化、个人设备提醒这8类消息推送，可以在厂商控制台申请开通。普通消息类型推送数量有限制。
具体请参见 厂商描述1 或 厂商描述2。
魅族：推送消息数量有限制，具体可参见 魅族平台合约。
FCM：推送上行消息频率有限制。
具体请参见 厂商描述。

收不到离线推送怎么排查？

1、OPPO 手机

OPPO 手机收不到推送一般有以下几种情况：
按照 OPPO 推送官网要求，在 Android 8.0 及以上系统版本的 OPPO 手机上必须配置 ChannelID，否则推送消息无法展示。配置方法可以参见 OPPO 推送配置。
在消息中 透传的离线推送的自定义内容 不是 JSON 格式，会导致 OPPO 手机收不到推送。
OPPO 安装应用通知栏显示默认关闭，需要确认下开关状态。

2、发送消息为自定义消息

自定义消息的离线推送和普通消息不太一样，自定义消息的内容我们无法解析，不能确定推送的内容，所以默认不推送，如果您有推送需求，需要您在 `sendMessage` 的时候设置 `offlinePushInfo` 的 `desc` 字段，推送的时候会默认展示 desc 信息。

3、设备通知栏设置影响

离线推送的直观表现就是通知栏提示，所以同其他通知一样受设备通知相关设置的影响，以华为为例：
“手机设置-通知-锁屏通知-隐藏或者不显示通知”，会影响锁屏状态下离线推送通知显示。
“手机设置-通知-更多通知设置-状态栏显示通知图标”，会影响状态栏下离线推送通知的图标显示。
“手机设置-通知-应用的通知管理-允许通知”，打开关闭会直接影响离线推送通知显示。
“手机设置-通知-应用的通知管理-通知铃声” 和 “手机设置-通知-应用的通知管理-静默通知”，会影响离线推送通知铃音的效果。

4、按照流程接入完成，还是收不到离线推送

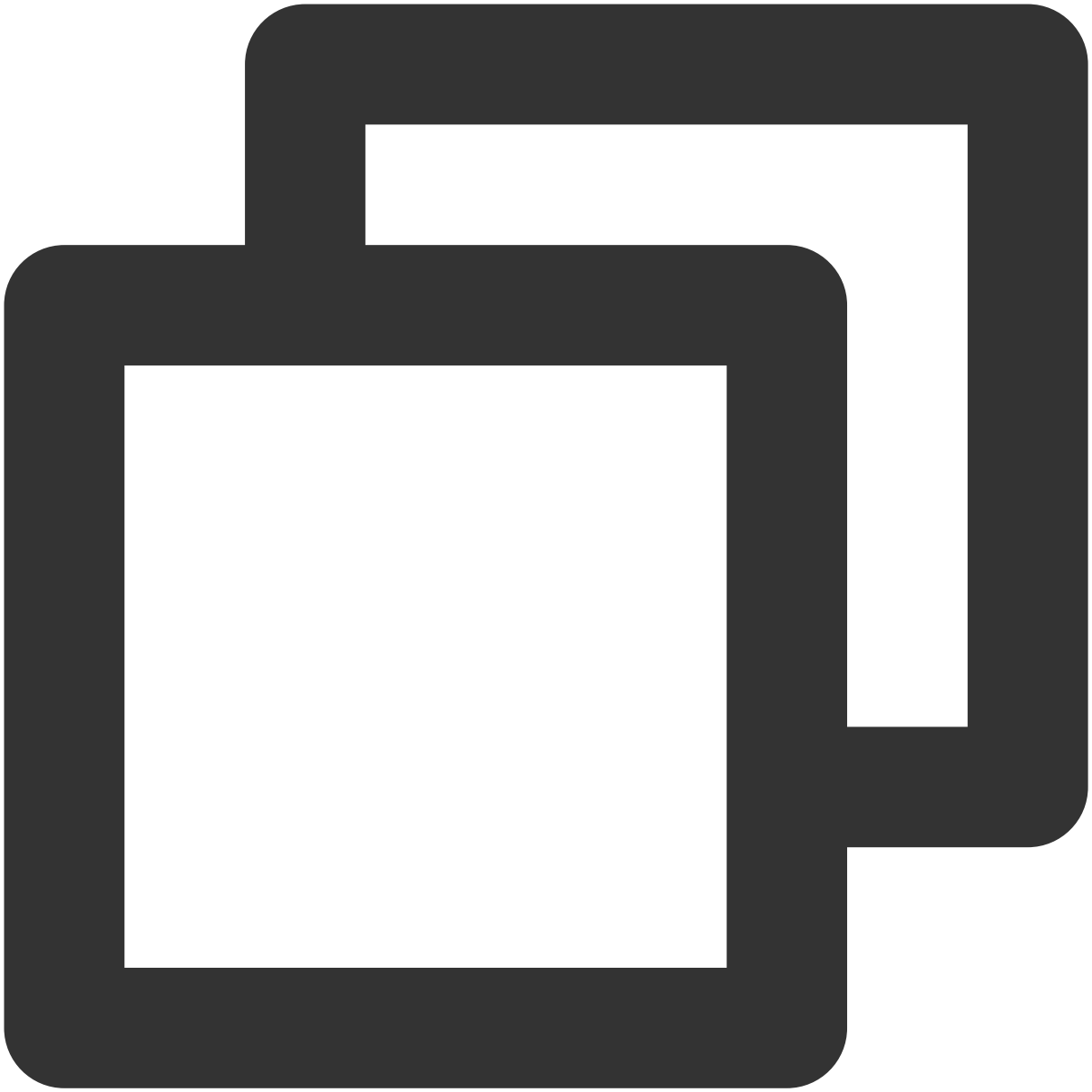
首先在 IM 控制台通过 离线测试工具 自测下是否可以正常推送。
推送异常情况，设备状态异常，需要检查下 IM 控制台配置各项参数是否正确，再者需要检查下代码初始化注册逻辑，包括厂商推送服务注册和 IM 设置离线推送配置相关逻辑是否正确设置。
推送异常情况，设备状态正常，需要看下是否需要正确填写 channel ID 或者后台服务是否正常。
离线推送依赖厂商能力，一些简单的字符可能会被厂商过滤不能透传推送。如 OPPO 则对 ext 字段限制为 JSON 格式。
如果离线推送消息出现推送不及时或者偶尔收不到情况，需要看下厂商的推送限制。

在线推送-在本地创建新消息通知

本文以上部分介绍了，如何使用本插件，结合腾讯云 IM 后端的推送服务，实现通过厂商通道的离线推送。
但是，在某些情况下，厂商离线推送并不适用。如，您的目标客户端机型非我们兼容的厂商，使用华强北定制的 Android 设备等。
此时，您只得通过在线监听收到新消息回调，在客户端上，手动触发创建通知。这仅适用于，应用未被kill掉，还处于前后台状态，能正常与 IM 服务端通信。
为此种情况，本插件在0.3版本中，新增两个本地创建消息的方法，`displayNotification` 自定义通知，及 `displayDefaultNotificationForMessage` 根据消息生成默认通知，您可按需使用。

接入前准备

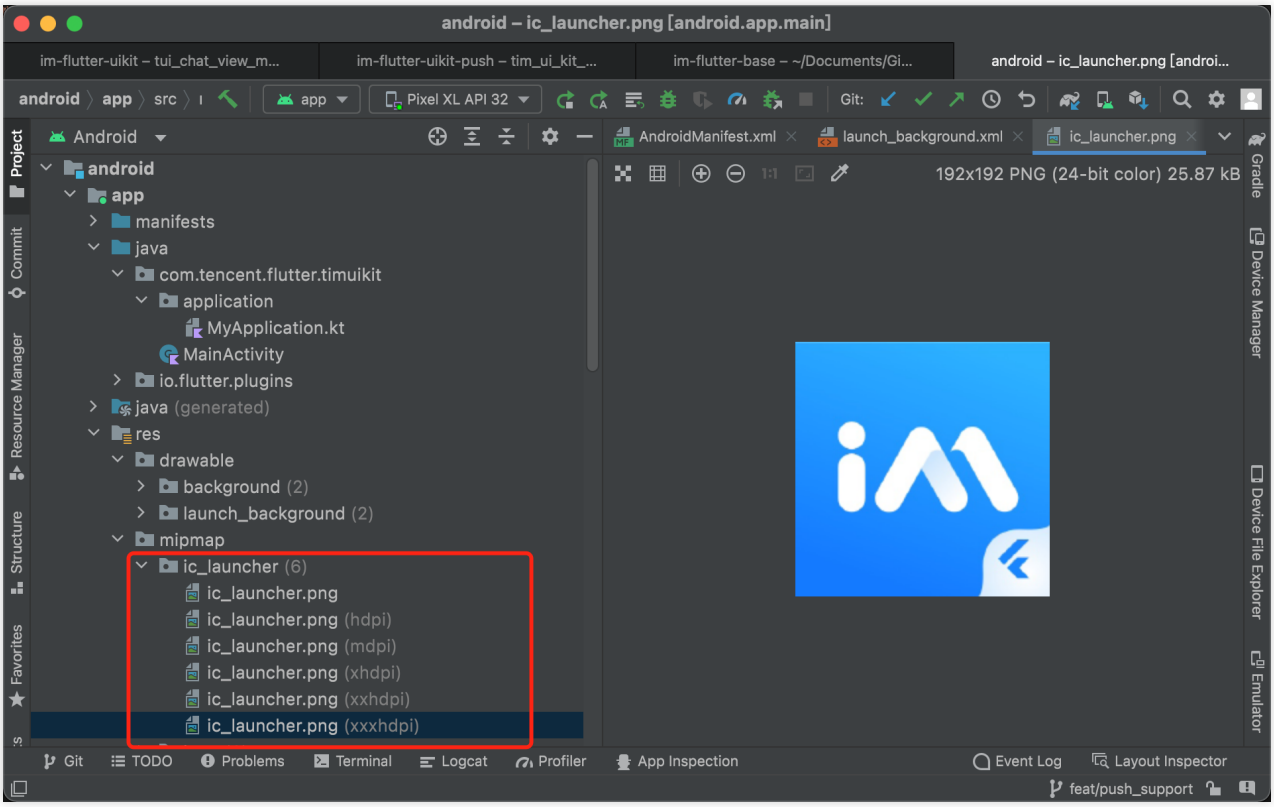
在您的项目中安装 IM Flutter 推送插件：



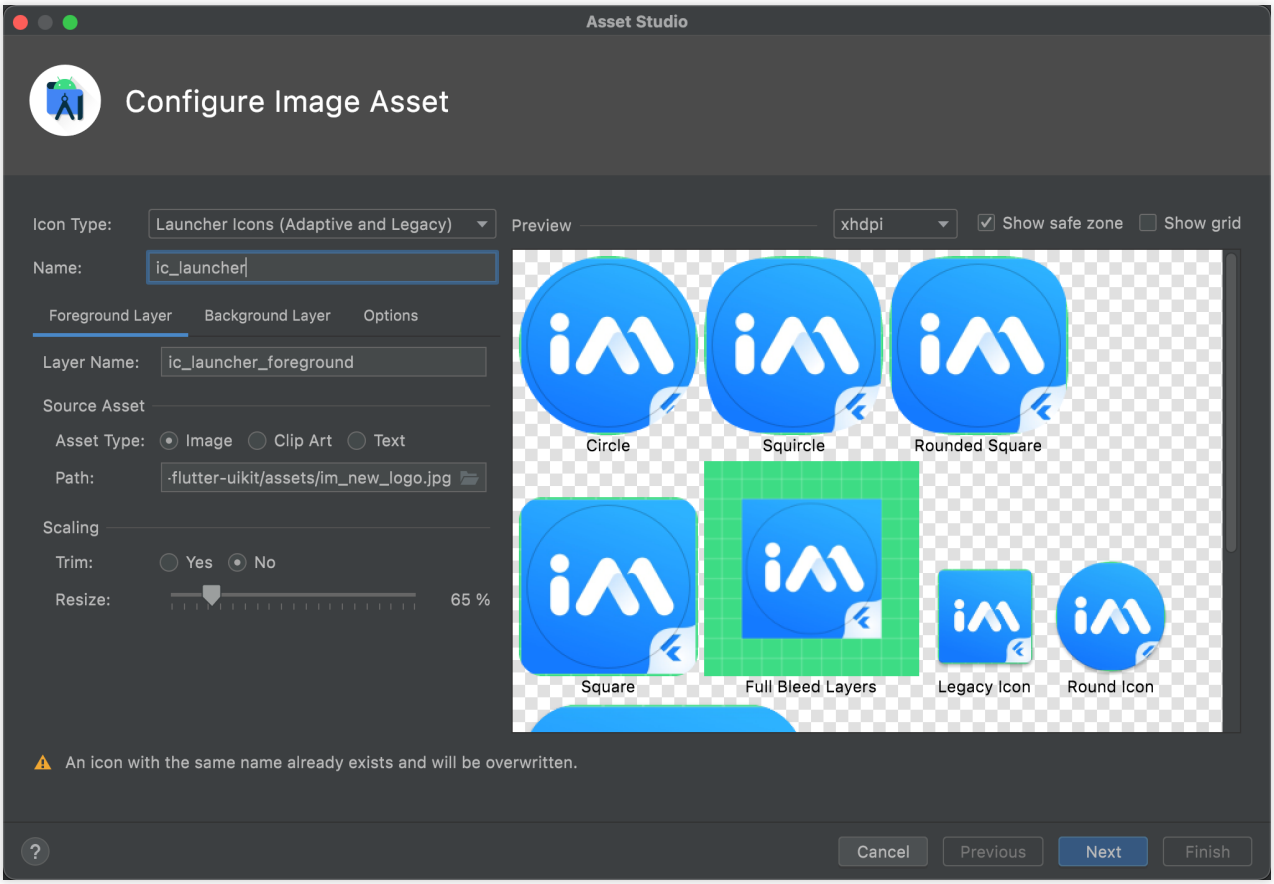
```
flutter pub add tim_ui_kit_push_plugin
```

Android

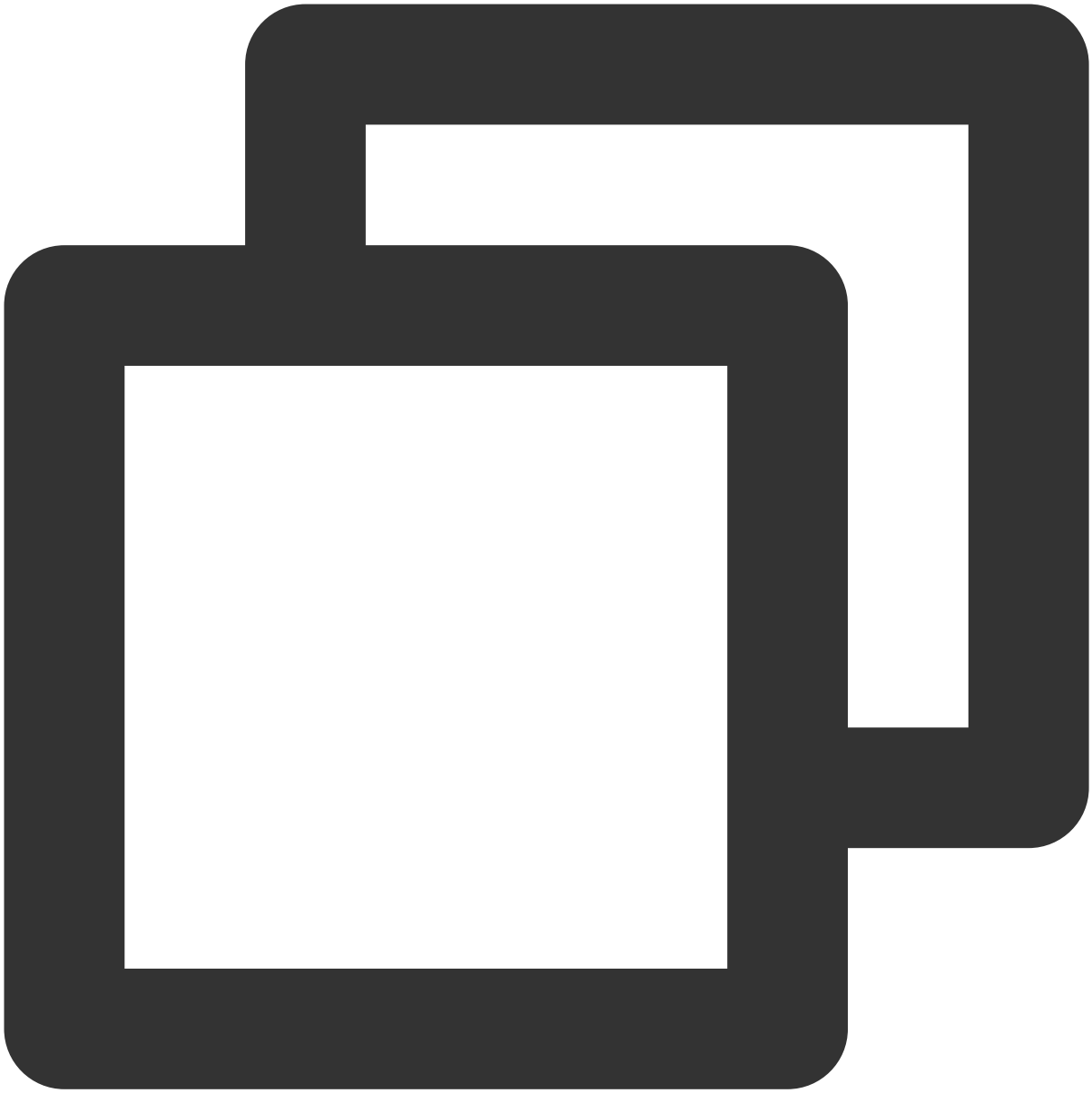
1. 确保 `@mipmap/ic_launcher` 存在且为您的应用 iCon。完整路径： `android/app/src/main/res/mipmap/ic_launcher.png`



如果不存在，可手动将您的应用 iCon 复制进去，或通过 Android Studio 自动创建不同分辨率版本（ `mipmap` 目录右键， `New` => `Image Asset` ）。



2. 打开 `android/app/src/main/AndroidManifest.xml` 文件，在您应用的主 activity 中，添加如下代码。

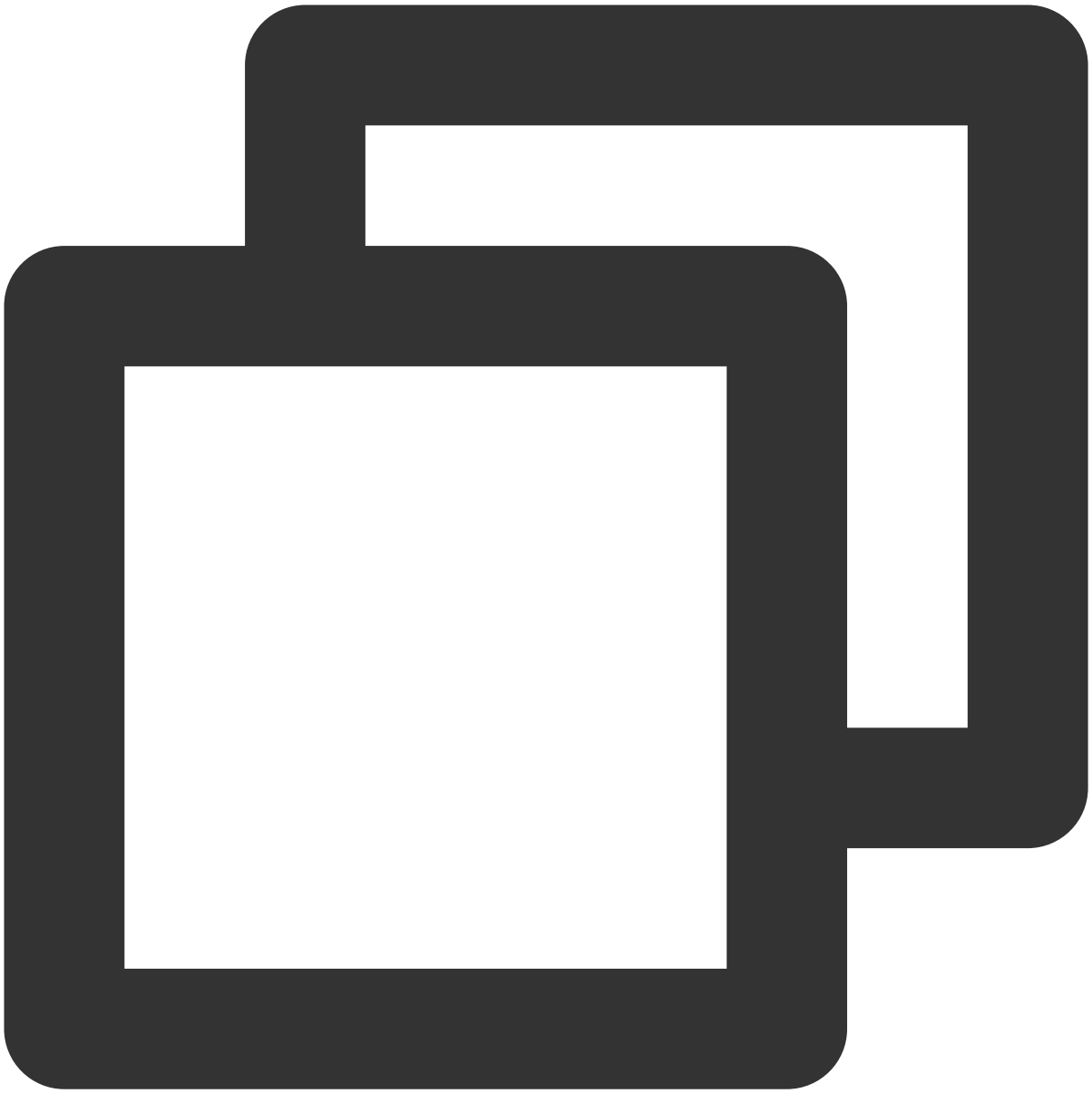


```
<activity
 android:showWhenLocked="true"
 android:turnScreenOn="true">
```

iOS

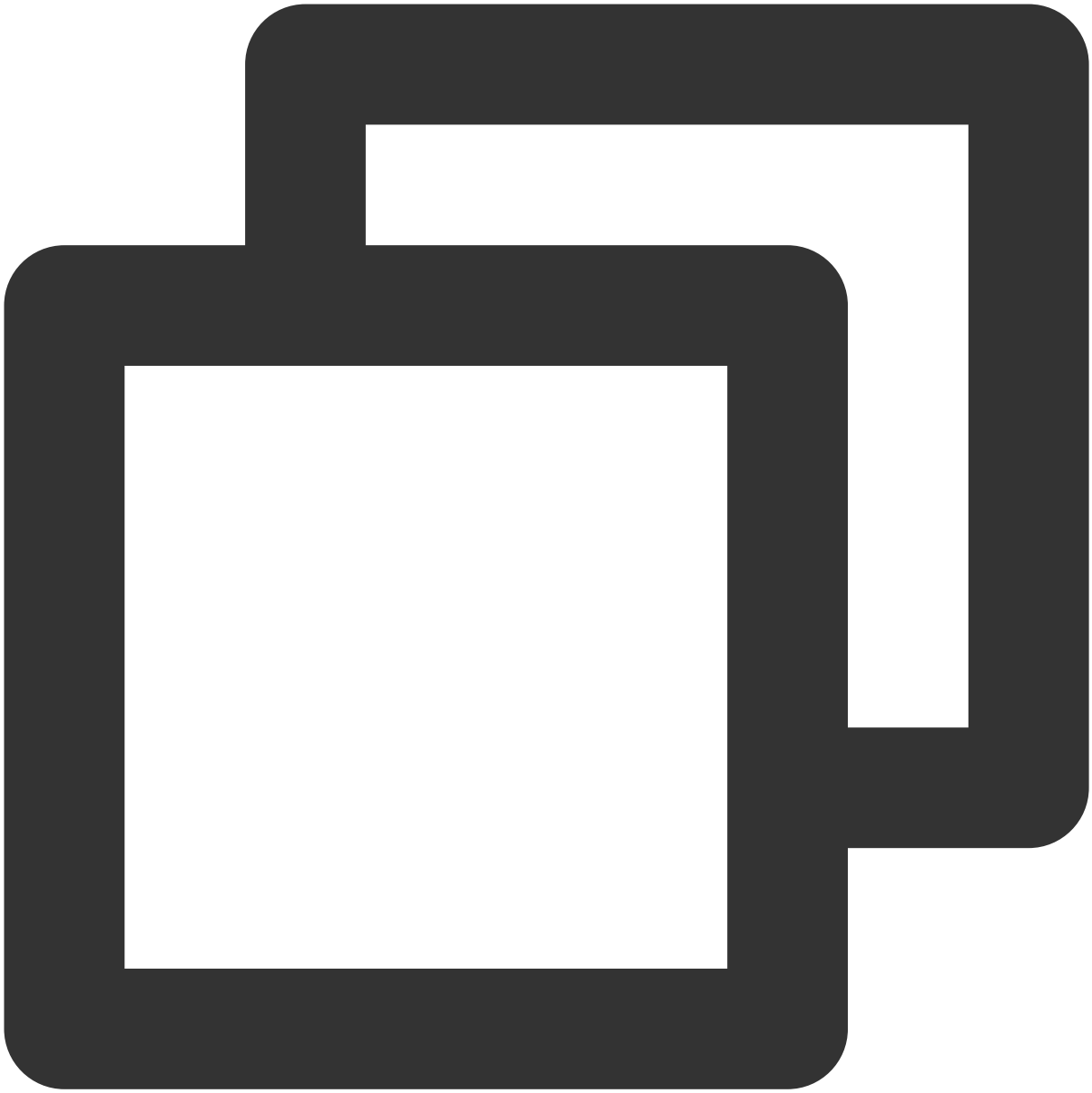
如果您已经配置 iOS 端离线推送，可忽略本部分。若无，请在 `ios/Runner/AppDelegate.swift` 或 `ios/Runner/AppDelegate.m` 文件中， `didFinishLaunchingWithOptions` 函数内，添加如下代码。可参考我们的DEMO。

Objective-C:



```
if (@available(iOS 10.0, *)) {
  [UNUserNotificationCenter currentNotificationCenter].delegate = (id<UNUserNotific
}
```

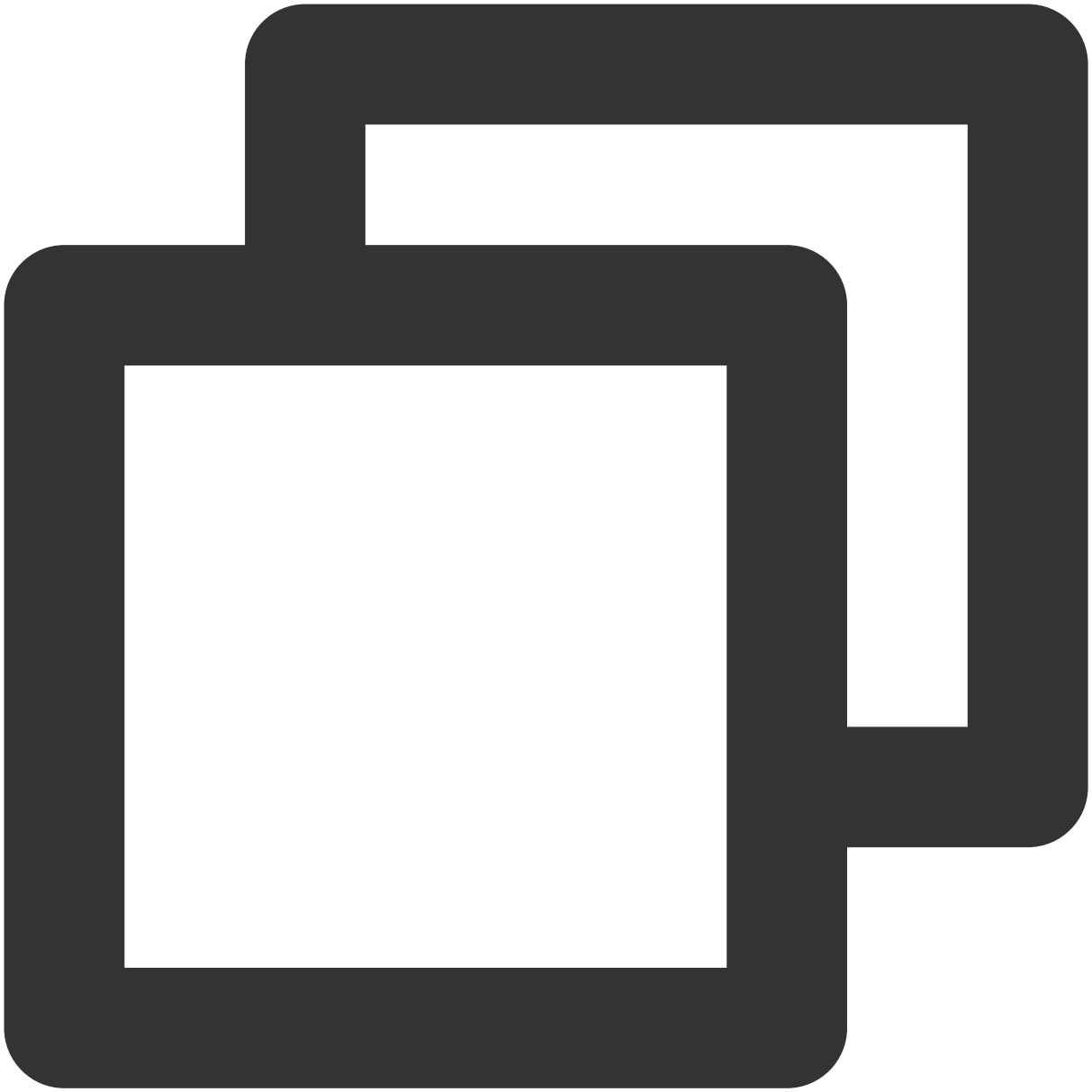
Swift:



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

初始化插件

请在 IM SDK 初始化完成后，初始化本 Push 插件。实例化一个 `cPush` 插件类，供后续调用。



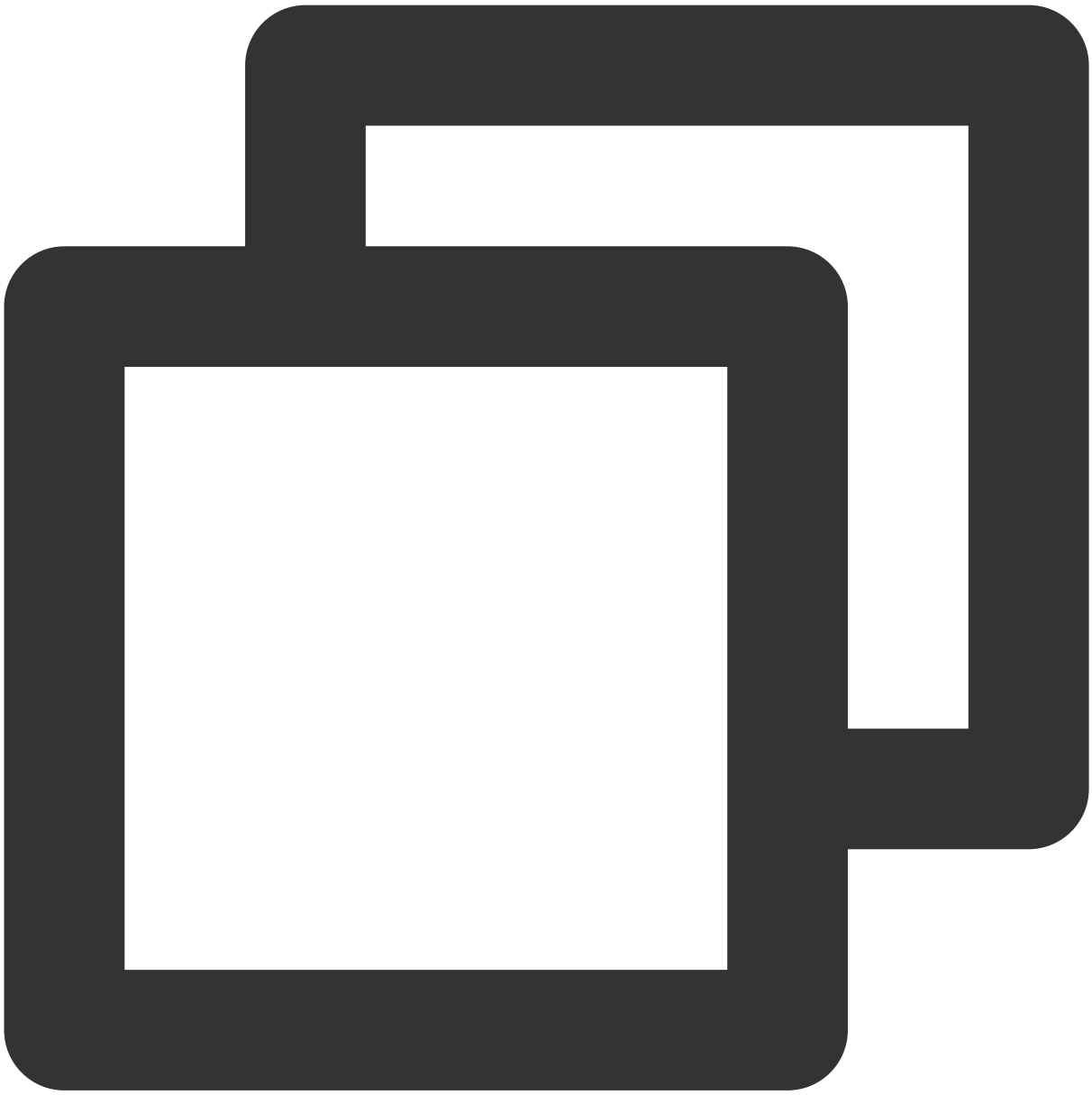
```
final TimUiKitPushPlugin cPush = TimUiKitPushPlugin();

cPush.init(
  // 此处绑定点击通知的跳转函数，下文会介绍
  pushClickAction: onClickNotification,
);
```

监听新消息回调触发通知

监听 `V2TimAdvancedMsgListener`

如果您已经挂载监听 `V2TimAdvancedMsgListener` ，可忽略本部分；若无，请在 IM login 后，挂载监听。
代码如下：



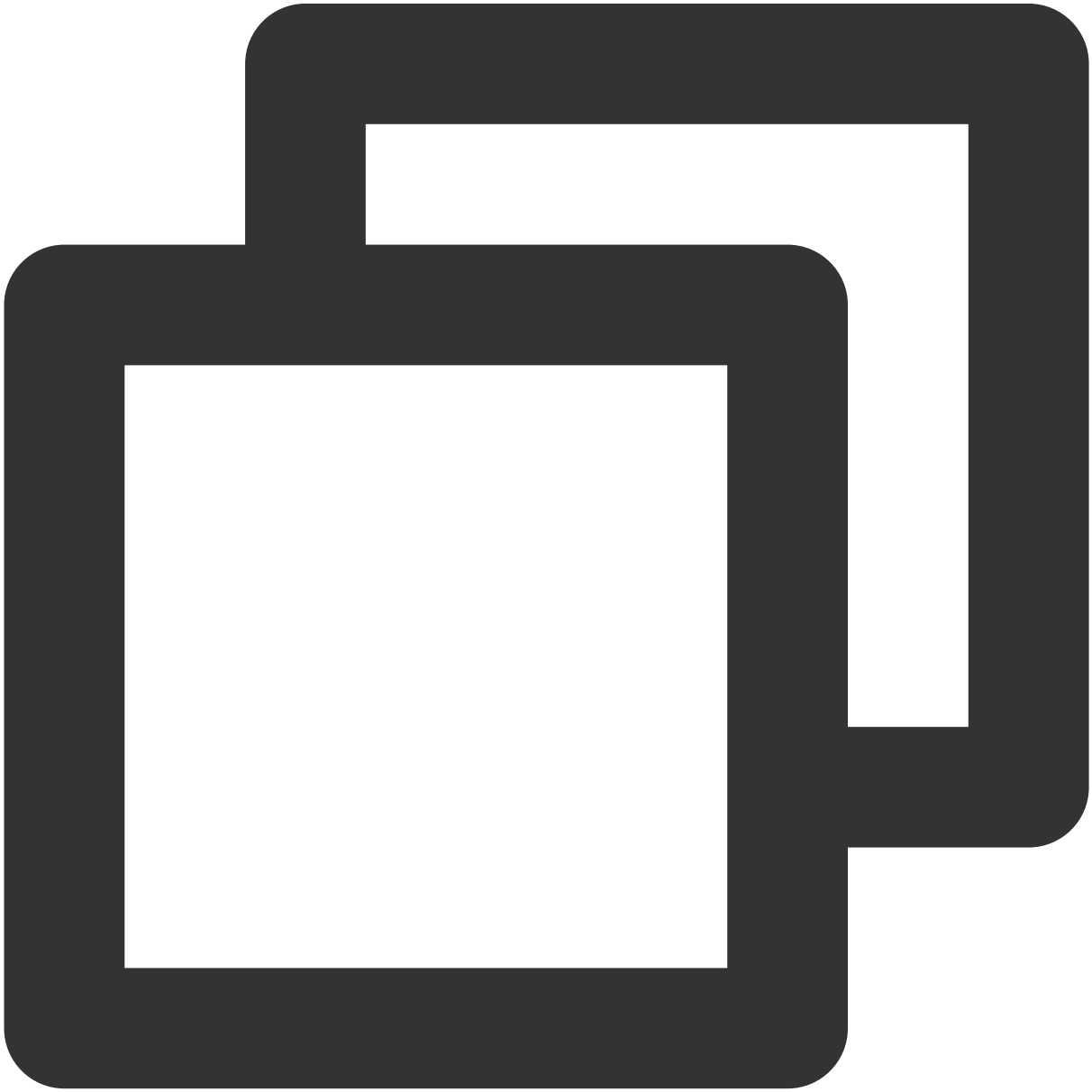
```
final advancedMsgListener = V2TimAdvancedMsgListener(
  onRecvNewMessage: (V2TimMessage newMsg) {
    // 这里完成监听回调触发事件
    // 下一步创建的方法，请在这里调用
  },
});

TencentImSDKPlugin.v2TIMManager
```

```
        .getMessageManager()
        .addAdvancedMsgListener(listener: advancedMsgListener);
```

触发本地消息通知

请从我们提供的两个 API 中，`displayNotification` 自定义通知，及 `displayDefaultNotificationForMessage` 根据消息生成默认通知，选一个合适的 API。
对于Android端，这两个 API 均需传入 `channelID` 及 `channelName` 。若还未创建 Android Push Channel ，请使用插件 `createNotificationChannel` API 创建。

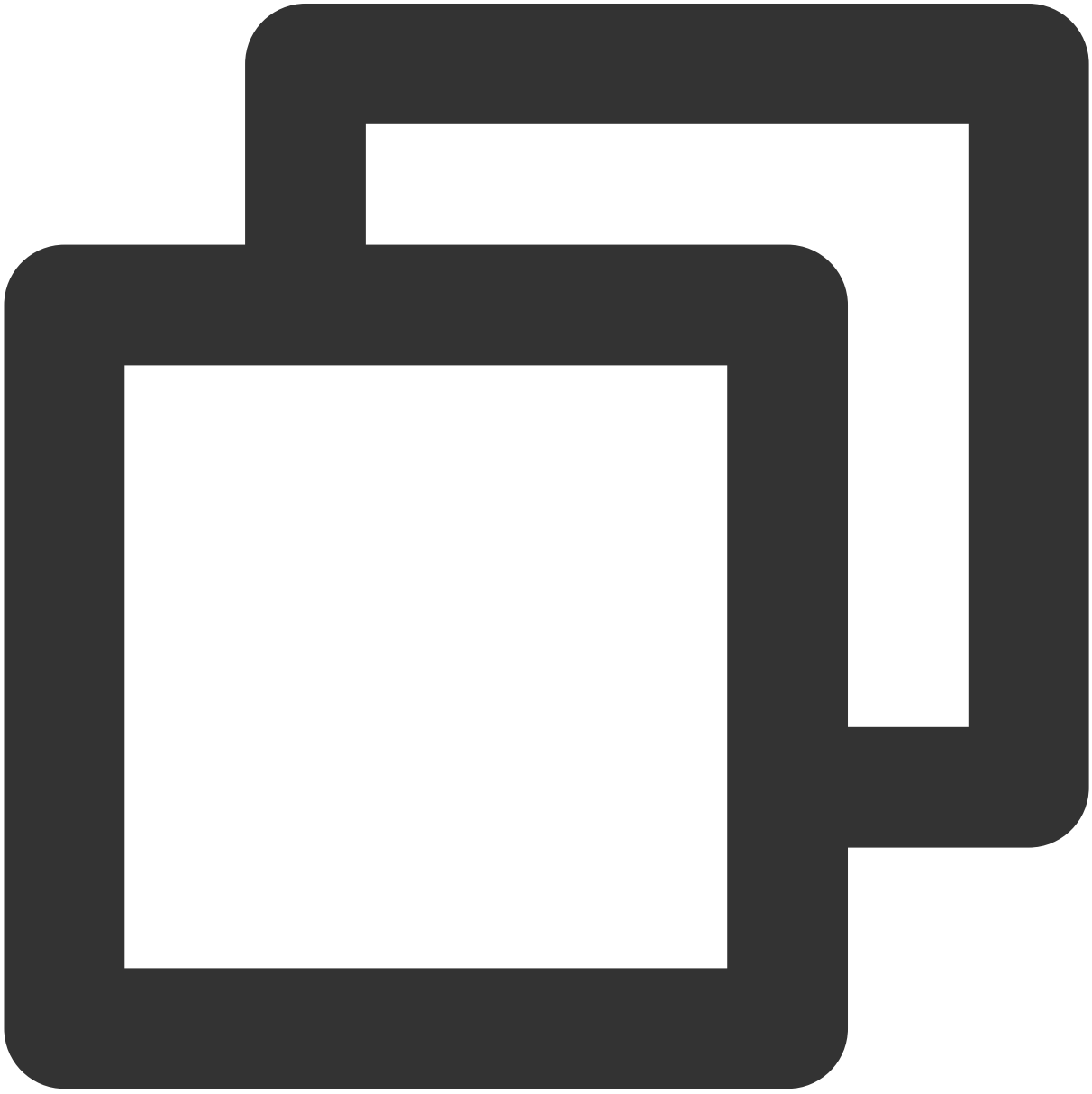


```
cPush.createNotificationChannel(
        channelId: "new_message",
        channelName: "消息推送",
        channelDescription: "推送新聊天消息");
```

displayNotification

本 API 需要您提供 `title`，`body`，及 `ext` 用于点击跳转信息，三个参数。您可以根据需要自行解析收到的 `V2TimMessage`，生成这三个字段。

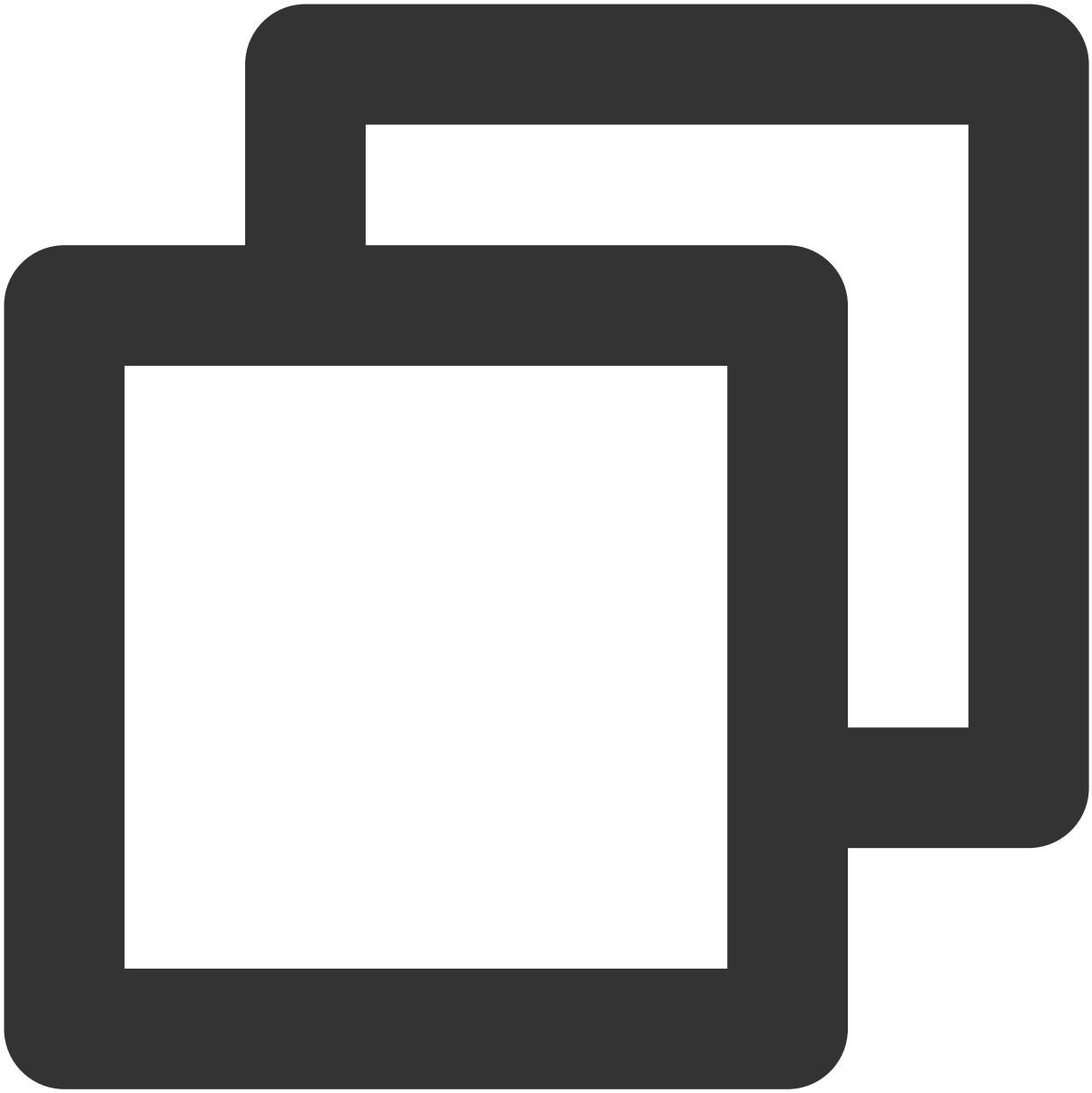
为便于跳转，此处 ext 的生成规则可查看 `displayDefaultNotificationForMessage` 的代码。



```
cPush.displayNotification(
  channelID: "new_message",
  channelName: "消息推送",
  title: "",
  body: "",
  ext: ""
);
```

displayDefaultNotificationForMessage

为了方便，推荐您使用此 API，自动根据 `V2TimMessage`，生成通知。
您只需传入一个 `V2TimMessage` 即可。



```
cPush.displayDefaultNotificationForMessage(
        message: message, channelID: "new_message", channelName: "消息推送");
```

点击通知跳转

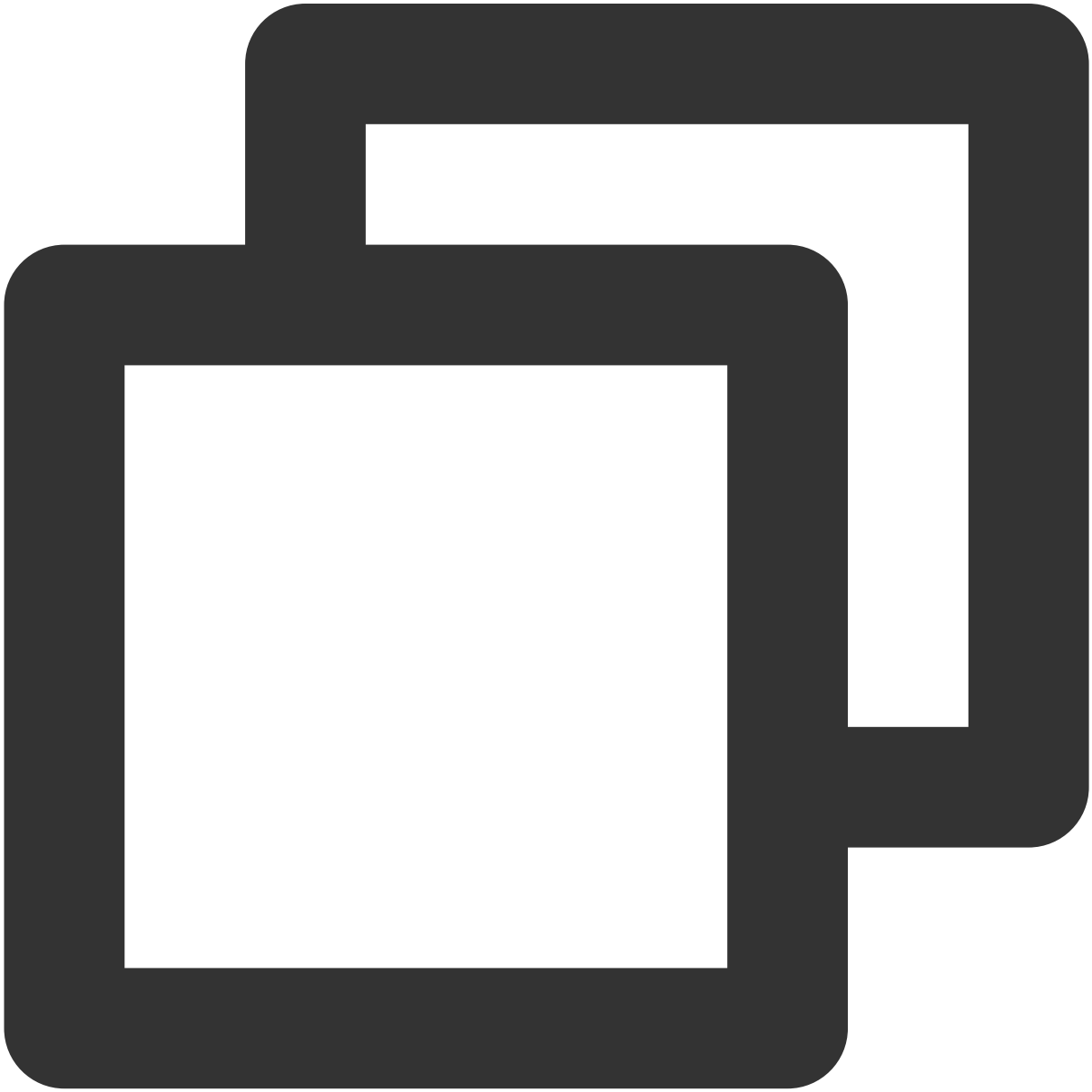
本步骤与 上文离线推送的步骤6 单击回调一致，均为在 ext 中，读取需要跳转的 conversation，并导航过去。
如果您在上一步使用 `displayDefaultNotificationForMessage` ，或在 `displayNotification` 中使用与default相同的 ext 生成函数，此时的 ext 结构为： `"conversationID": "对应的conversation"` 。
此时，填上初始化时，为 pushClickAction 埋的坑。

初始化时，注册该回调方法，可拿到含推送本体及 ext 信息在内的 Map。

说明：

在后台跳转情况下，此时 Flutter 首页可能已经 unmounted，无法为跳转提供 context，因此建议启动时缓存一个 context，保证跳转成功。

建议跳转成功后，清除通知栏中其他通知消息，避免太多 IM 消息堆积在通知栏中。调用插件中 `clearAllNotification()` 方法即可。



```
BuildContext? _cachedContext;
final TimUiKitPushPlugin cPush = TimUiKitPushPlugin(
      isUseGoogleFCM: false,
```

```
    );
// 仅限TUIKit
final TIMUIKitChatController _timuiKitChatController =
  TIMUIKitChatController();

@override
void initState() {
  super.initState();
  _cachedContext = context;
}

void onClickNotification(Map<String, dynamic> msg) async {
    String ext = msg['ext'] ?? "";
    Map<String, dynamic> extMsp = jsonDecode(ext);
    String convId = extMsp["conversationID"] ?? "";

    // 【TUIKit】若当前的会话与要跳转至的会话一致，则不跳转
    final currentConvID = _timuiKitChatController.getCurrentConversation();
    if(currentConvID == convId.split("_")[1]){
      return;
    }

    final targetConversationRes = await TencentImSDKPlugin.v2TIMManager
        .getConversationManager()
        .getConversation(conversationID: convId);

    V2TimConversation? targetConversation = targetConversationRes.data;

    if(targetConversation != null){
      cPush.clearAllNotification();
      Navigator.push(
          _cachedContext ?? context,
          MaterialPageRoute(
            builder: (context) => Chat(
              selectedConversation: targetConversation,
            ),
          ));
    }
  }
```

如果您自定义了 `ext` 结构，则需自实现点击跳转函数。

此时，您已完成在线推送的接入。测试通过后，你可以在 `onRecvNewMessage` 内定义，触发推送通知的时机及场景。

用户在线状态

Android

最近更新时间：2024-05-20 15:12:45

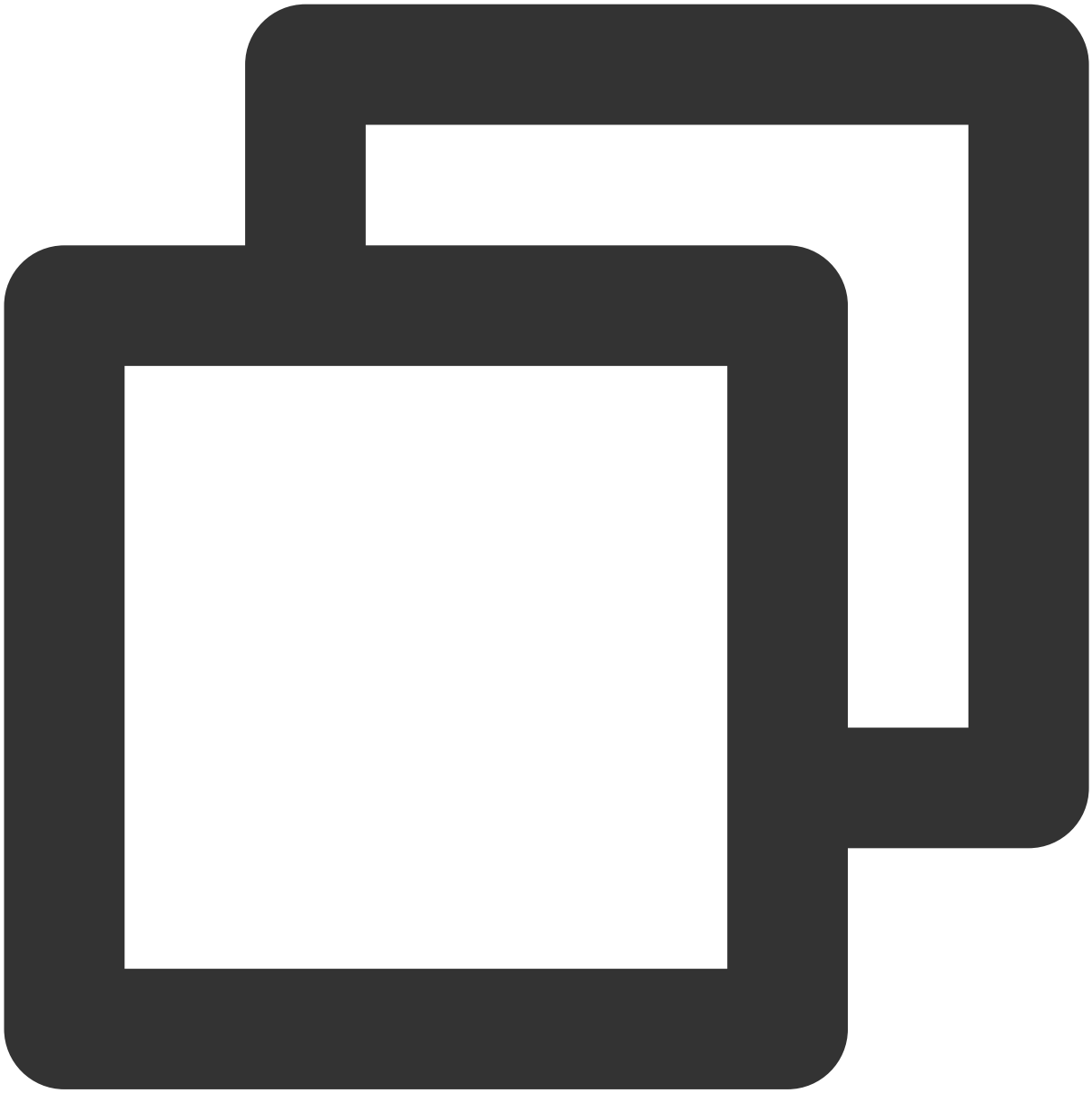
功能描述

TUIKit 从 6.5.2803 版本开始支持用户在线状态展示。
开启“显示用户在线状态” 后，会在会话列表和联系人列表的用户头像上显示用户的在线状态。当绿圈出现时表示对方在线，没有绿圈则表示对方当前离线。
关闭“显示用户在线状态” 时，不再显示好友的用户在线状态。
注意：
“用户在线状态”功能仅进阶版套餐支持，使用前请确认已开通进阶版套餐。
“用户在线状态”功能需要在 Chat 控制台 打开用户状态开关，使用前请确认开关已经打开。

开启会话列表用户在线状态

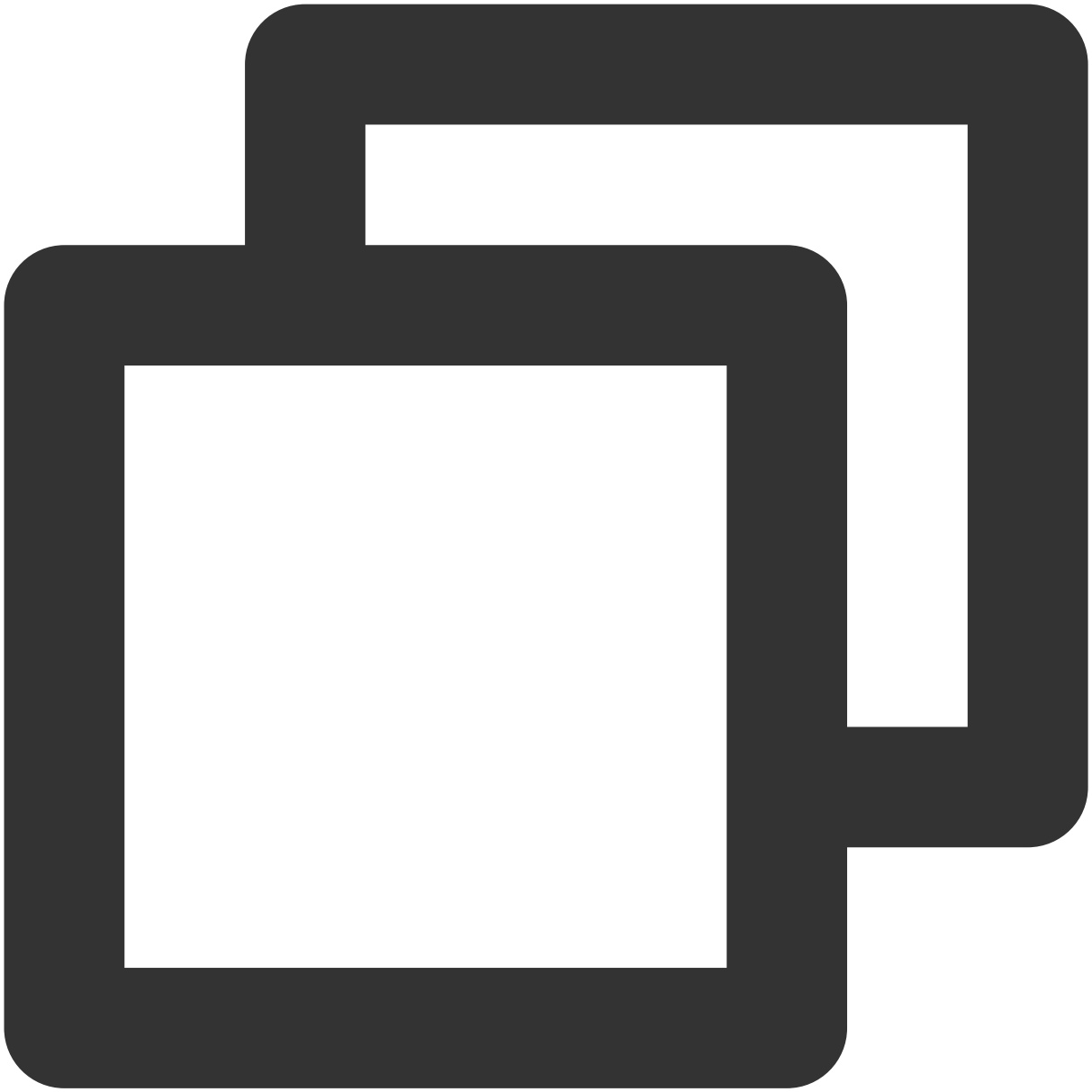
在 `TUIConversation` 组件的 TUIConversationConfig.java 文件中提供了“用户在线状态”功能开关 **isShowUserStatus**, 其类型为 boolean，默认为 false 。

版权所有：腾讯云计算（北京）有限责任公司



```
public class TUIConversationConfig {
    private boolean isShowUserStatus;
}
```

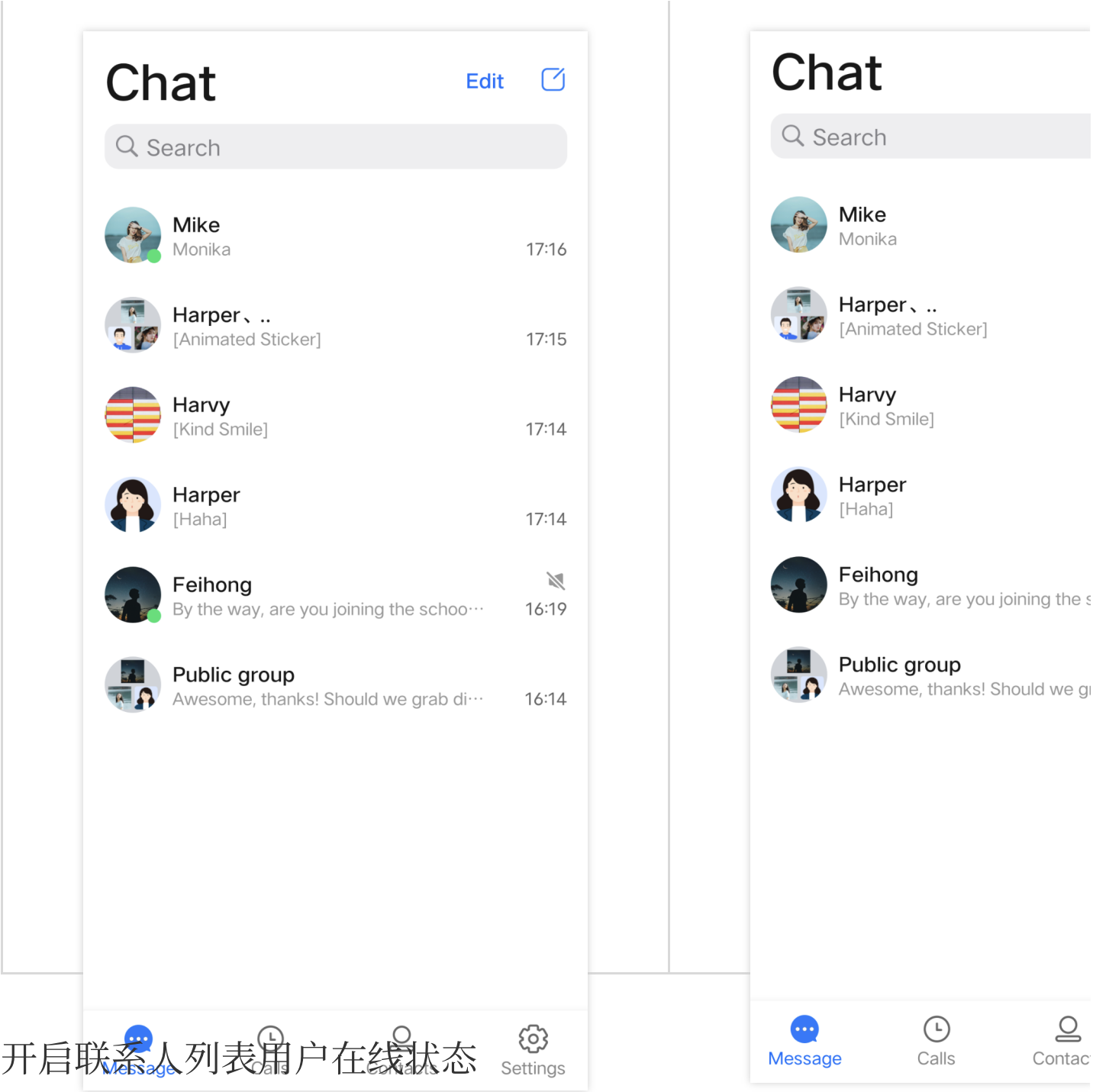
如果想开启会话列表展示用户在线状态功能，首先请开通进阶套餐包，然后在 Chat 控制台 打开用户状态功能的开关，再将 **isShowUserStatus** 的默认值改为 `true` ，或者在会话页面初始化之前调用以下方法来开启。



```
TUIConversationConfig.getInstance().setShowUserStatus(true);
```

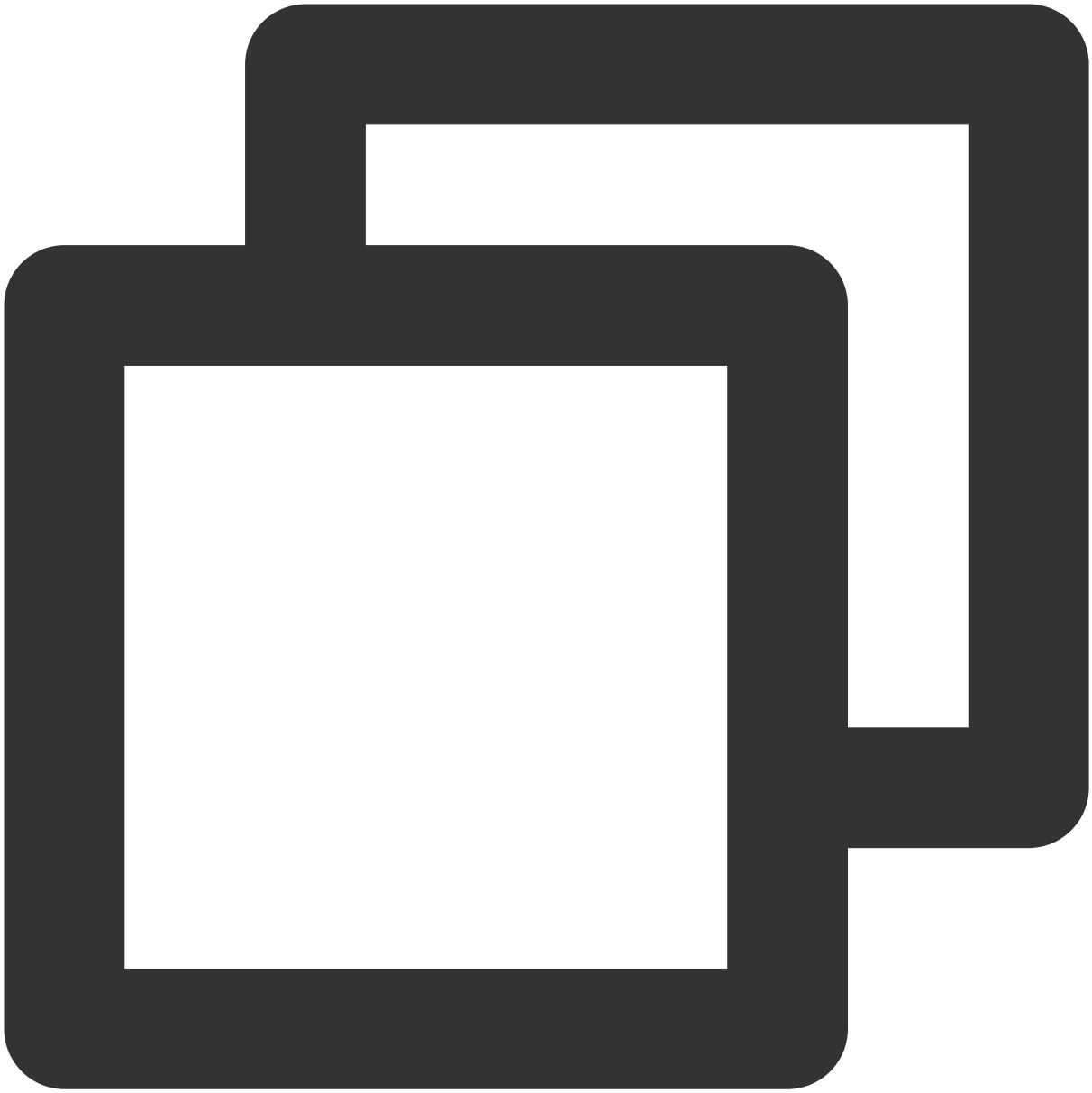
会话列表效果

开启“显示用户在线状态”	关闭“显示用户在线状态”



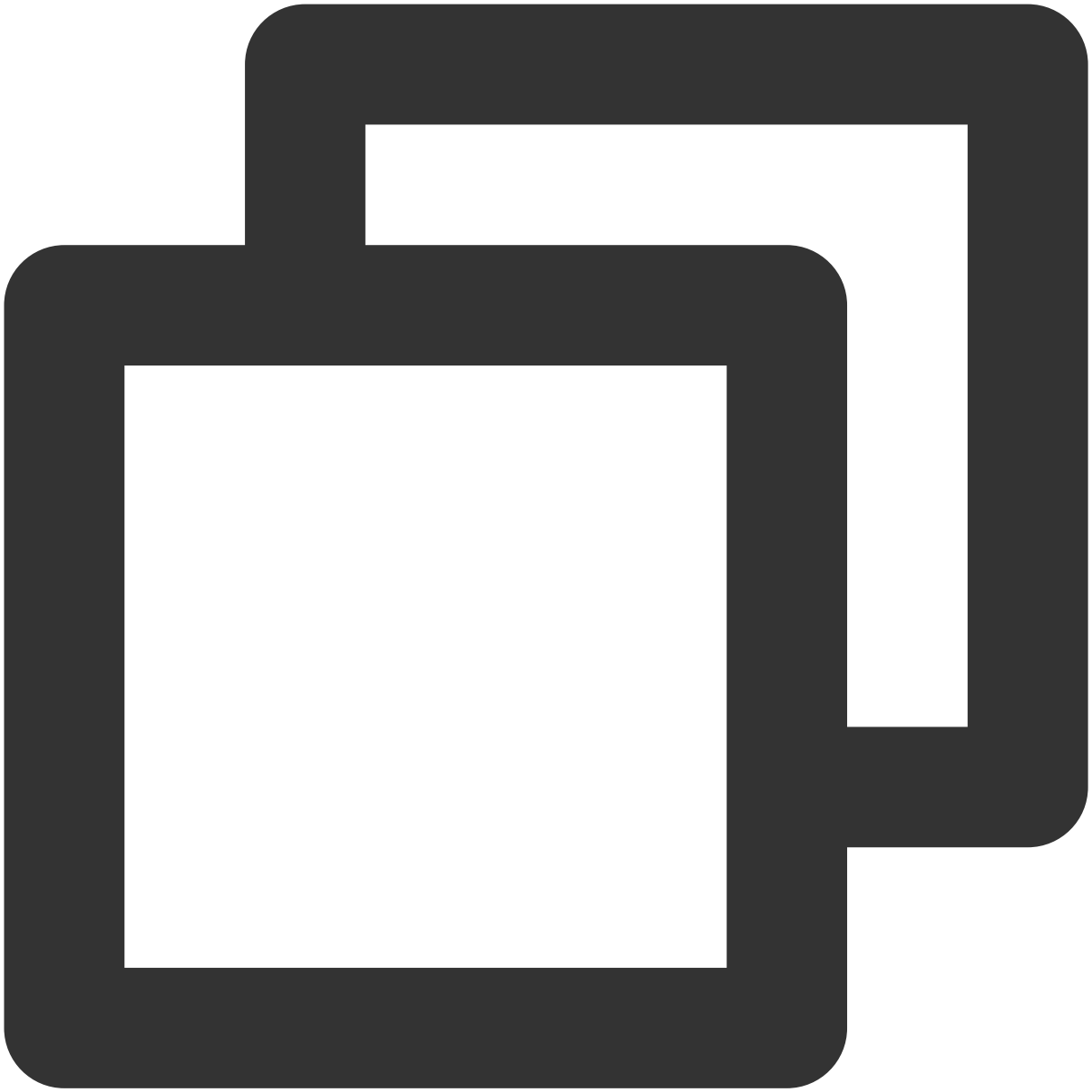
开启联系人列表用户在线状态

在 `TUIContact` 组件的 TUIContactConfig.java 文件中提供了“用户在线状态”功能开关 **isShowUserStatus** , 其类型为 boolean，默认为 false 。



```
public class TUIContactConfig {
    private boolean isShowUserStatus;
}
```

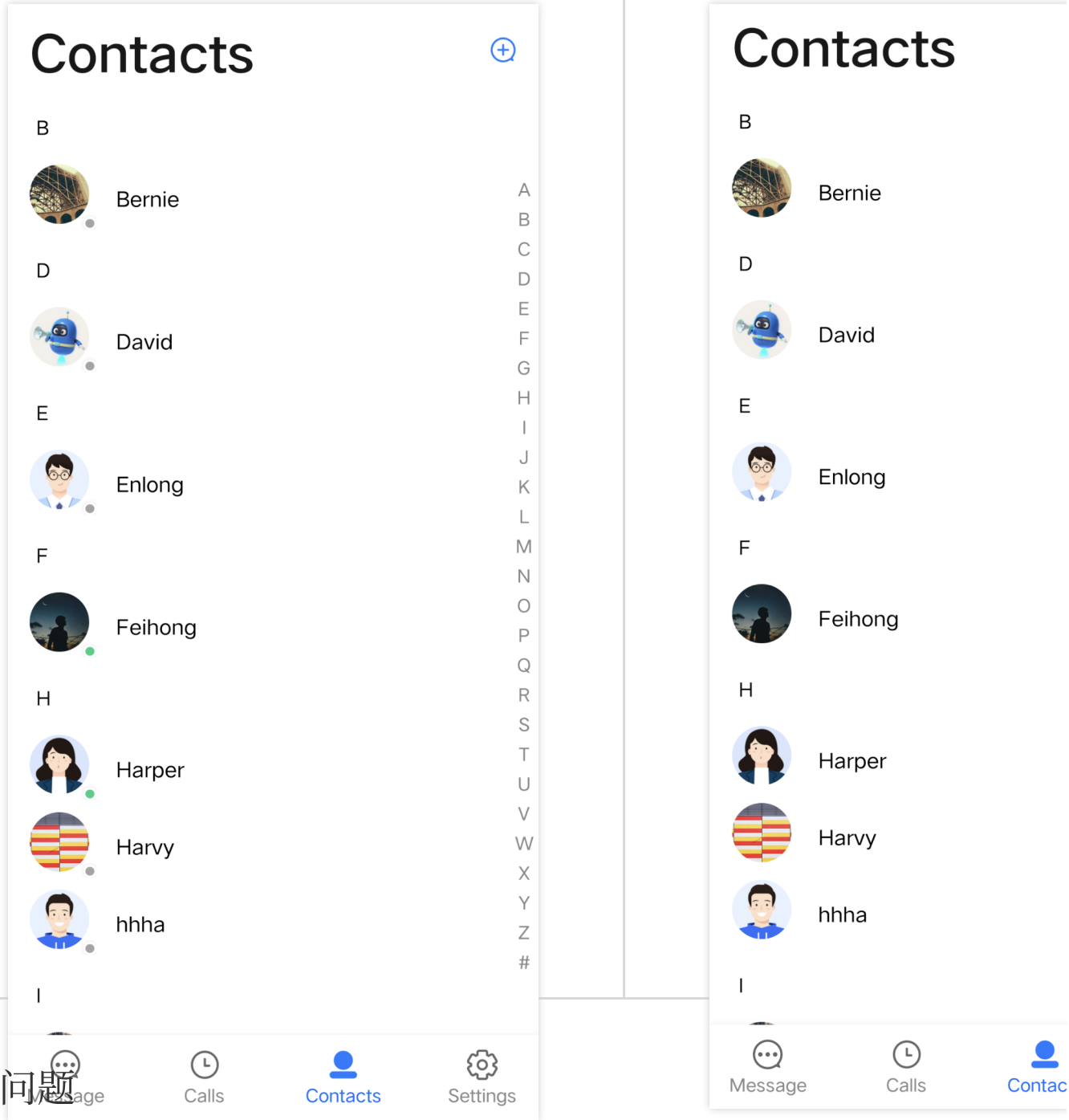
如果想开启联系人列表展示用户在线状态功能，首先请开通进阶套餐包，然后在 Chat 控制台 打开用户状态功能的开关，再将 **isShowUserStatus** 的默认值改为 `true` ，或者在联系人列表页面初始化之前调用以下方法来开启。



```
TUIContactConfig.getInstance().setShowUserStatus(true);
```

联系人列表效果

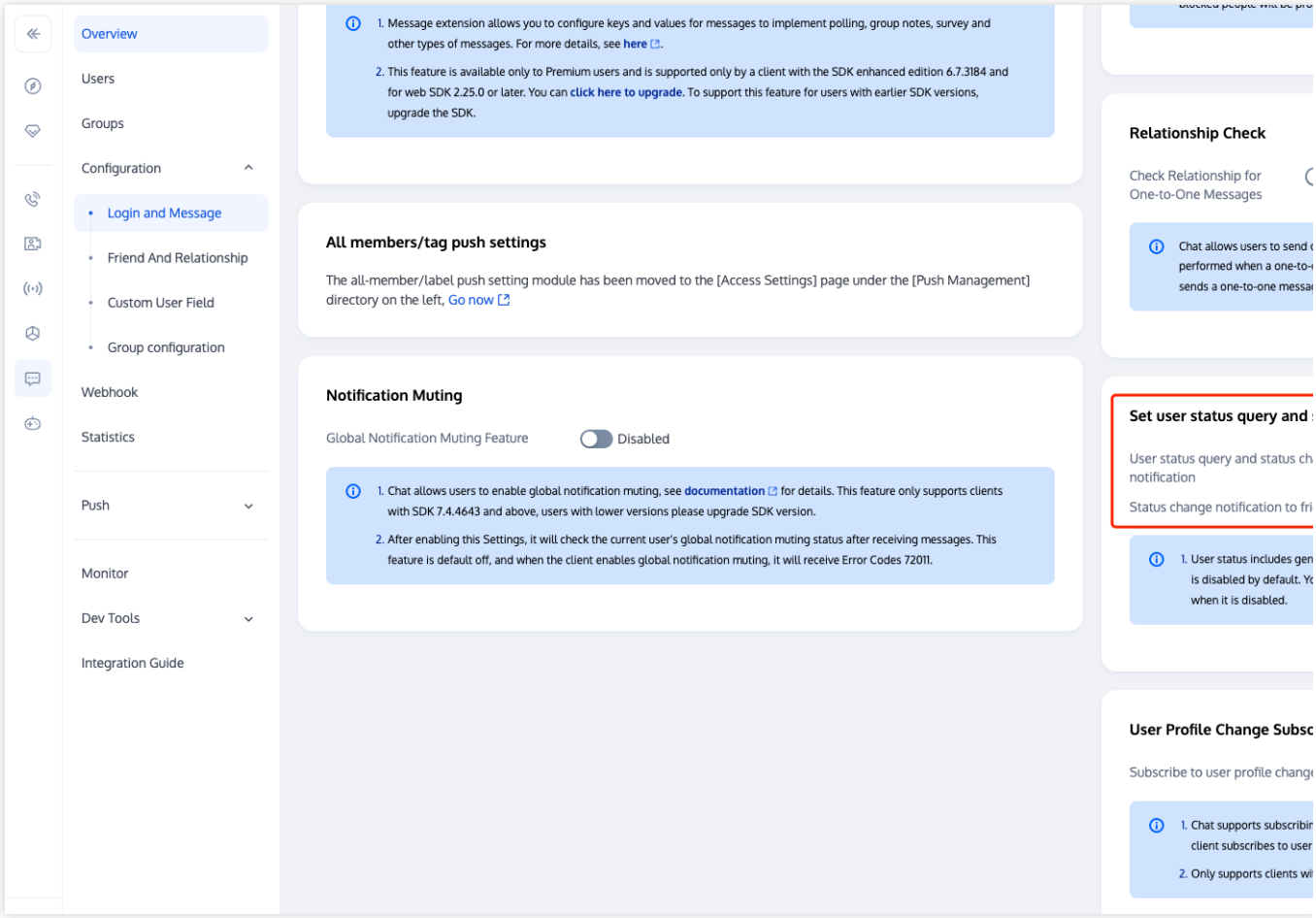
开启“显示用户在线状态”	关闭“显示用户在线状态”



常见问题

调用订阅/取消订阅接口时，接口提示 “72001” 的错误码。

72001 错误码表示在控制台上并没有开启对应的能力，请登录 Chat 控制台 打开对应的功能开关。



Error: 套餐包不支持该接口的使用，请升级到进阶版套餐。

“用户在线状态”功能仅进阶版套餐包支持，该报错信息表示您当前的套餐包不支持此能力，请登录 Chat 购买页 开通进阶版套餐包进行体验。

交流与反馈

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

版权所有：腾讯云计算（北京）有限责任公司

iOS

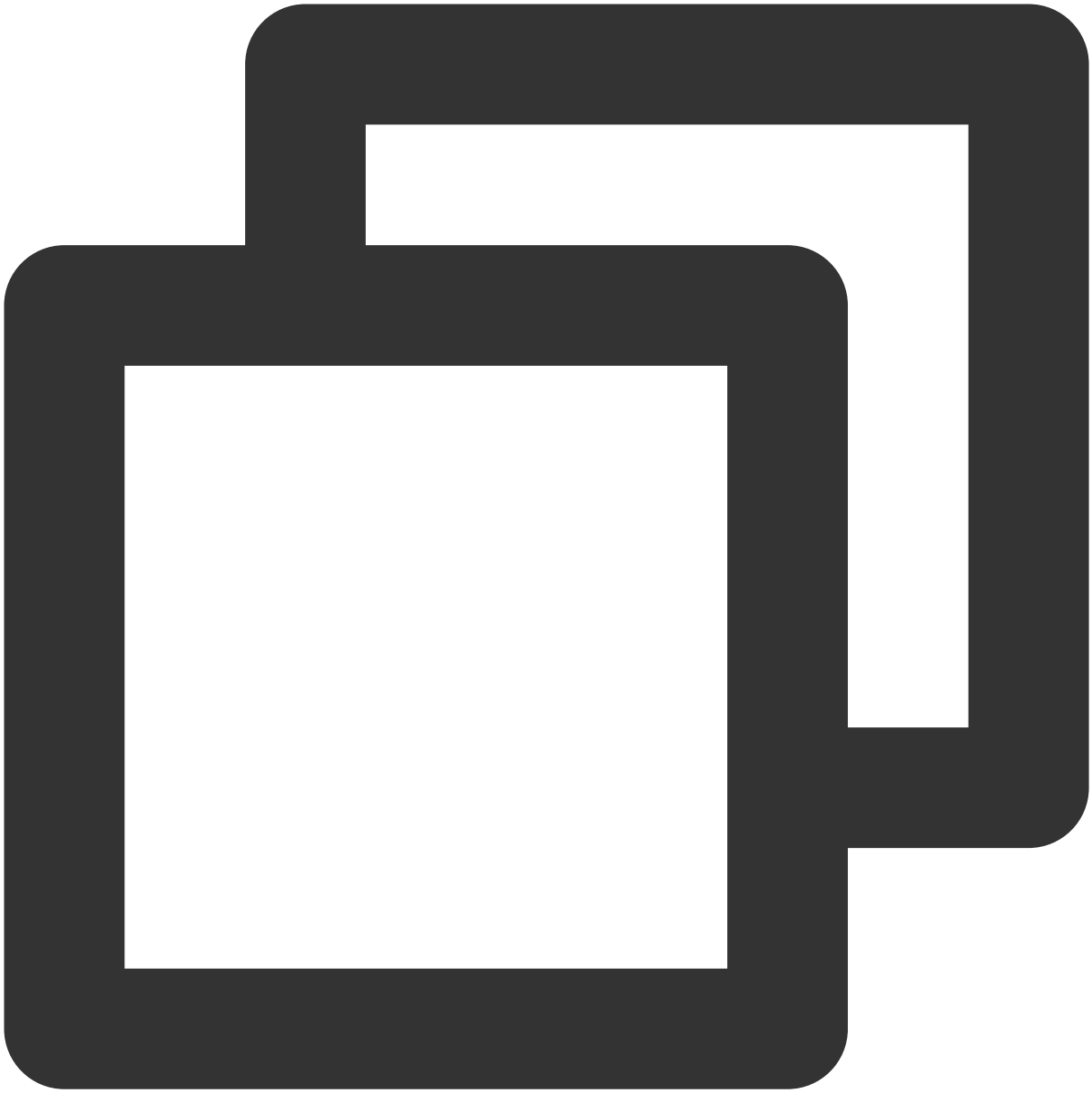
最近更新时间：2024-05-20 17:09:47

功能描述

TUIKit 从 6.5.2803 版本开始支持用户在线状态展示。
开启“显示用户在线状态” 后，会在会话列表和联系人列表的用户头像上显示用户的在线状态。当绿圈出现时表示对方在线，没有绿圈则表示对方当前离线。
关闭“显示用户在线状态” 时，不再显示好友的用户在线状态。
注意：
“用户在线状态”功能仅进阶版套餐支持，使用前请确认已开通进阶版套餐。
“用户在线状态”功能需要在 Chat 控制台 打开用户状态开关，使用前请确认开关已经打开。

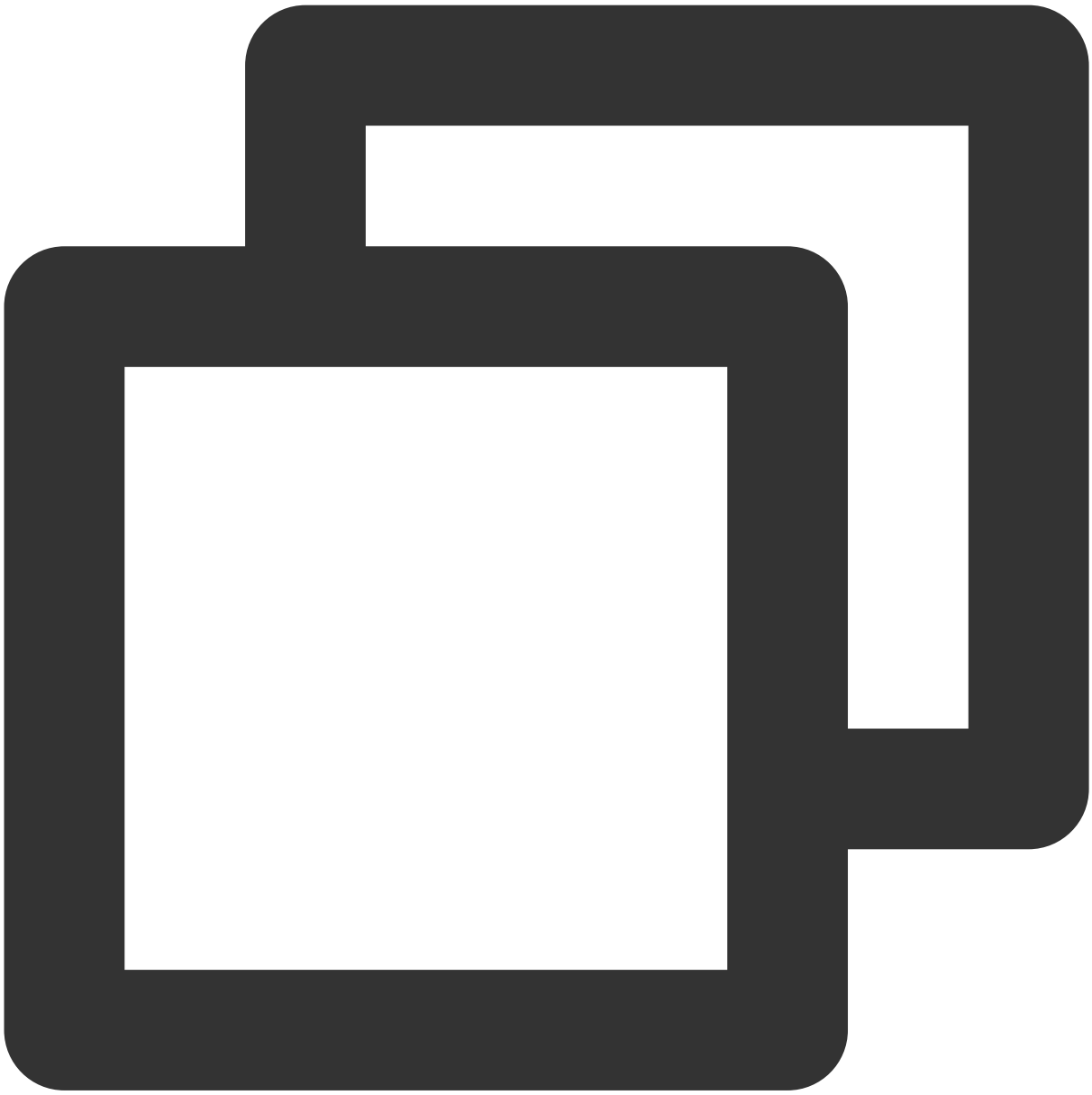
开启用户在线状态

在 `TUICore` 组件的 TUIConfig 文件中提供了“用户在线状态”功能开关 **displayOnlineStatusIcon**, 其类型为 BOOL，默认为 NO 。



```
- (id)init
{
self = [super init];
if(self){
//...其他配置
self.displayOnlineStatusIcon = NO;
}
return self;
}
```

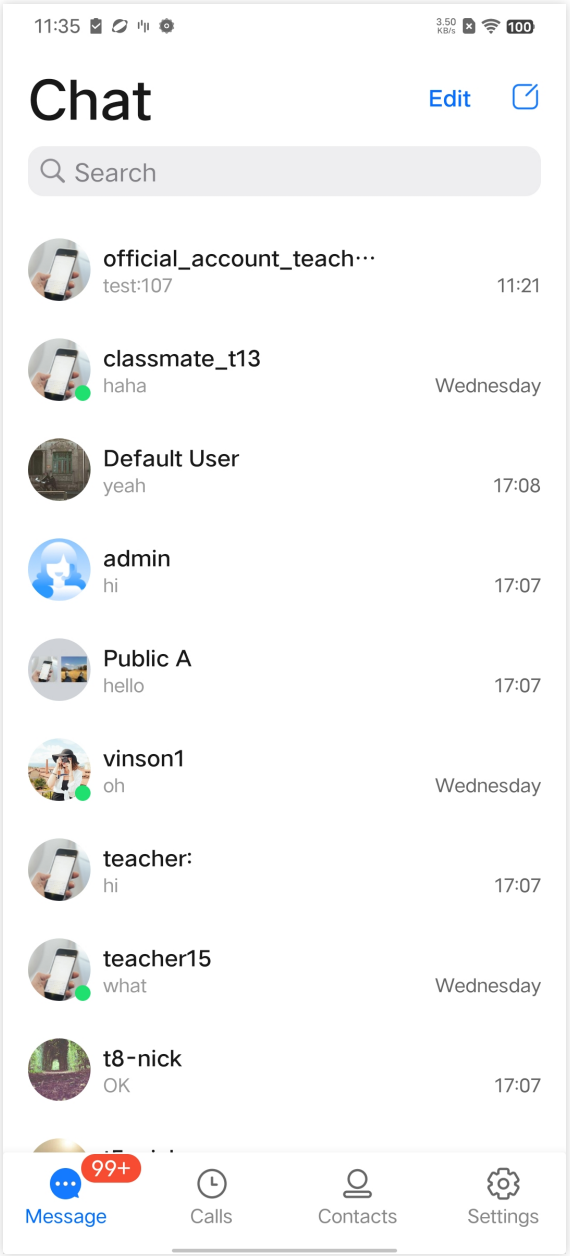
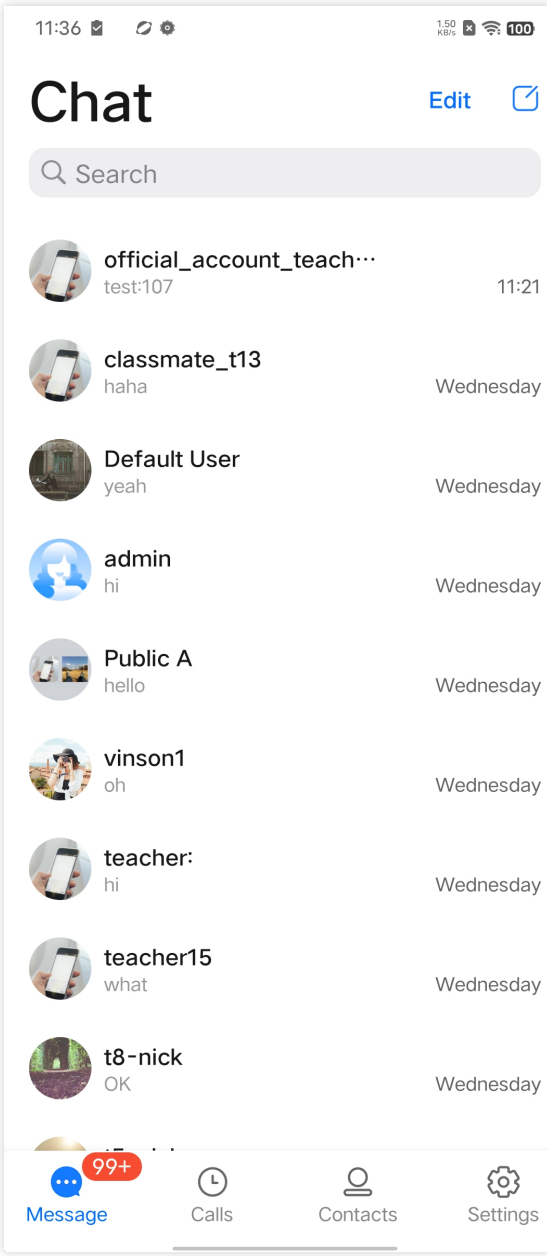
如果想开启会话列表展示用户在线状态功能，首先请开通进阶套餐包，然后在 Chat 控制台 打开用户状态功能的开关，再将 **displayOnlineStatusIcon** 的默认值改为 `YES` ，或者在会话页面初始化之前调用以下方法来开启。



```
[TUIConfig defaultConfig].displayOnlineStatusIcon = YES;
```

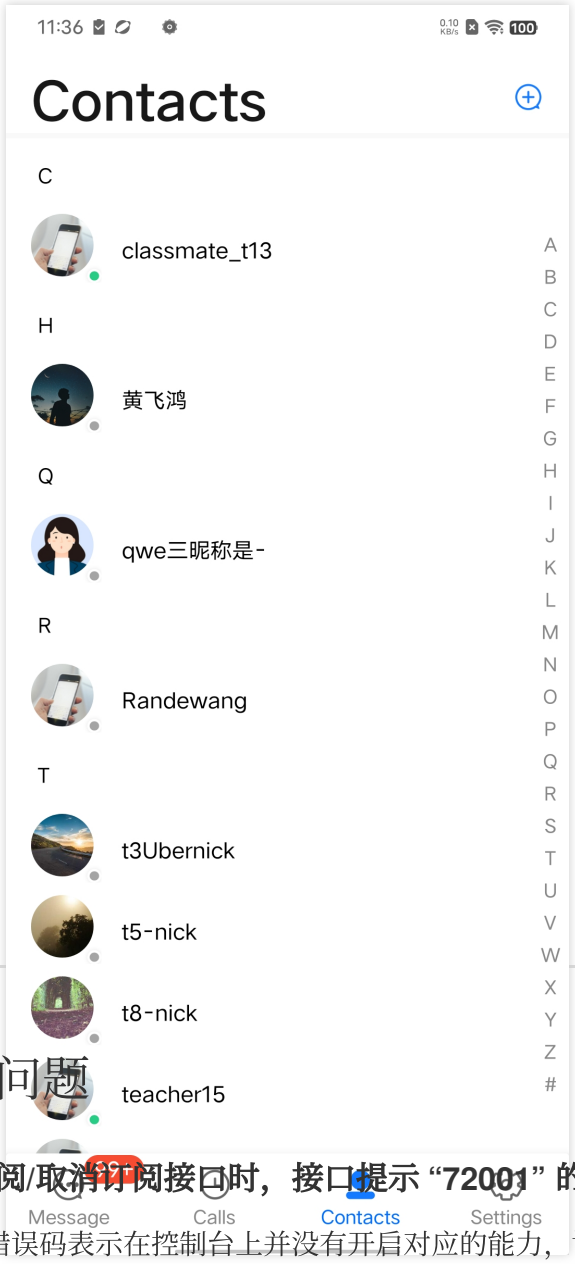
效果展示

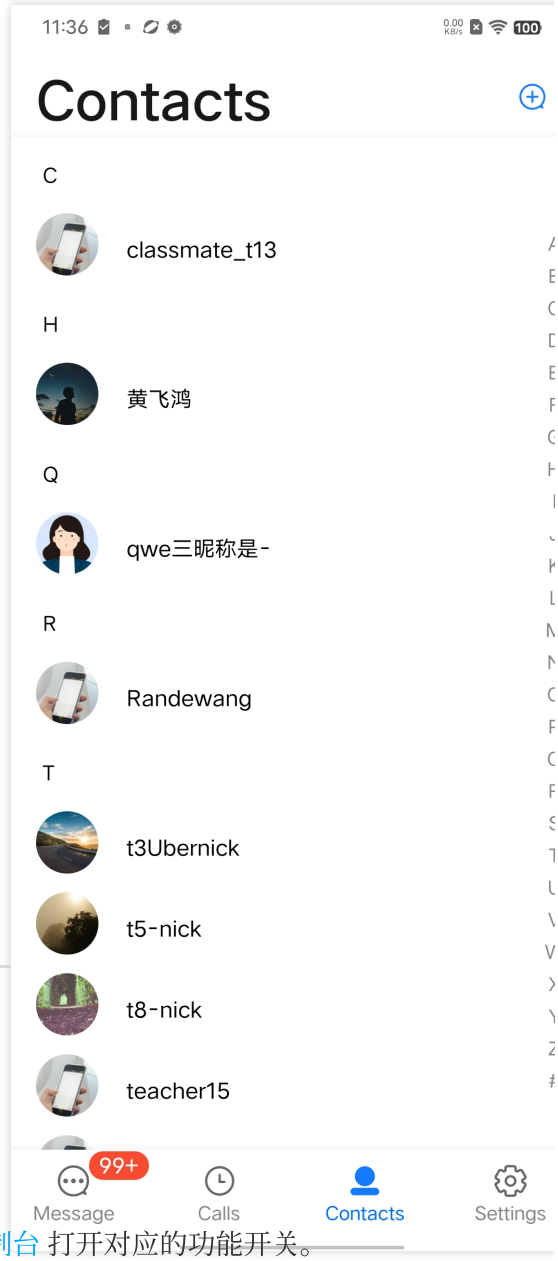
会话列表

Enabling "Show User Online Status"	Disabling "Show User Online Status"
	

联系人列表

开启“显示用户在线状态”	关闭“显示用户在线状态”

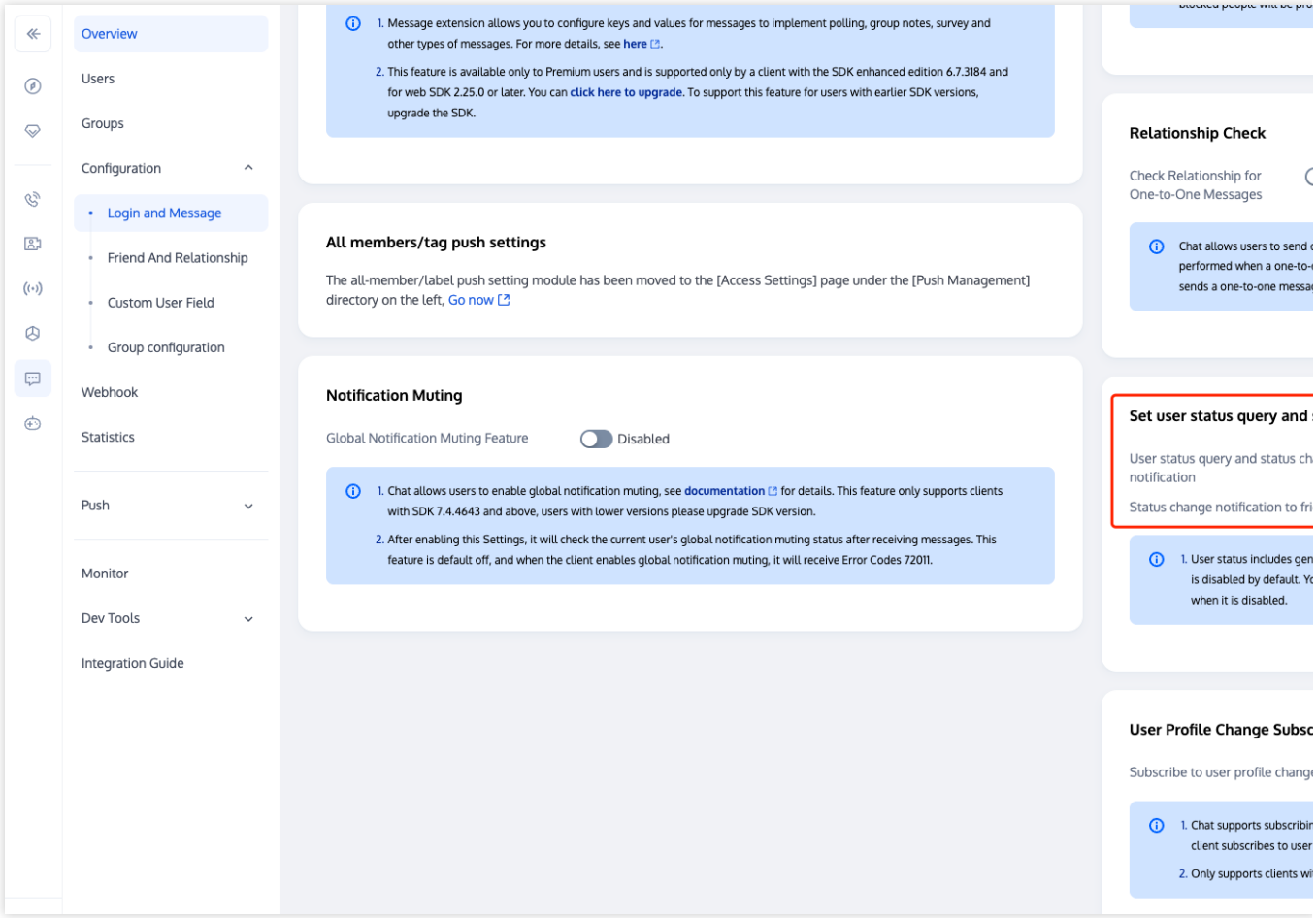




常见问题

调用订阅/取消订阅接口时，接口提示 “72001” 的错误码。

72001 错误码表示在控制台上并没有开启对应的能力，请登录 Chat 控制台 打开对应的功能开关。



Error: 套餐包不支持该接口的使用，请升级到进阶版套餐。

“用户在线状态”功能仅进阶版套餐包支持，该报错信息表示您当前的套餐包不支持此能力，请登录 Chat 购买页 开通进阶版套餐包进行体验。

交流与反馈

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

版权所有：腾讯云计算（北京）有限责任公司

Web & H5 & Uniapp (Vue)

最近更新时间：2024-07-10 16:26:41

功能描述

@tencentcloud/chat-uikit-vue 从 v2.0.0 版本开始，已支持“用户在线状态”功能， 效果如下：

注意：

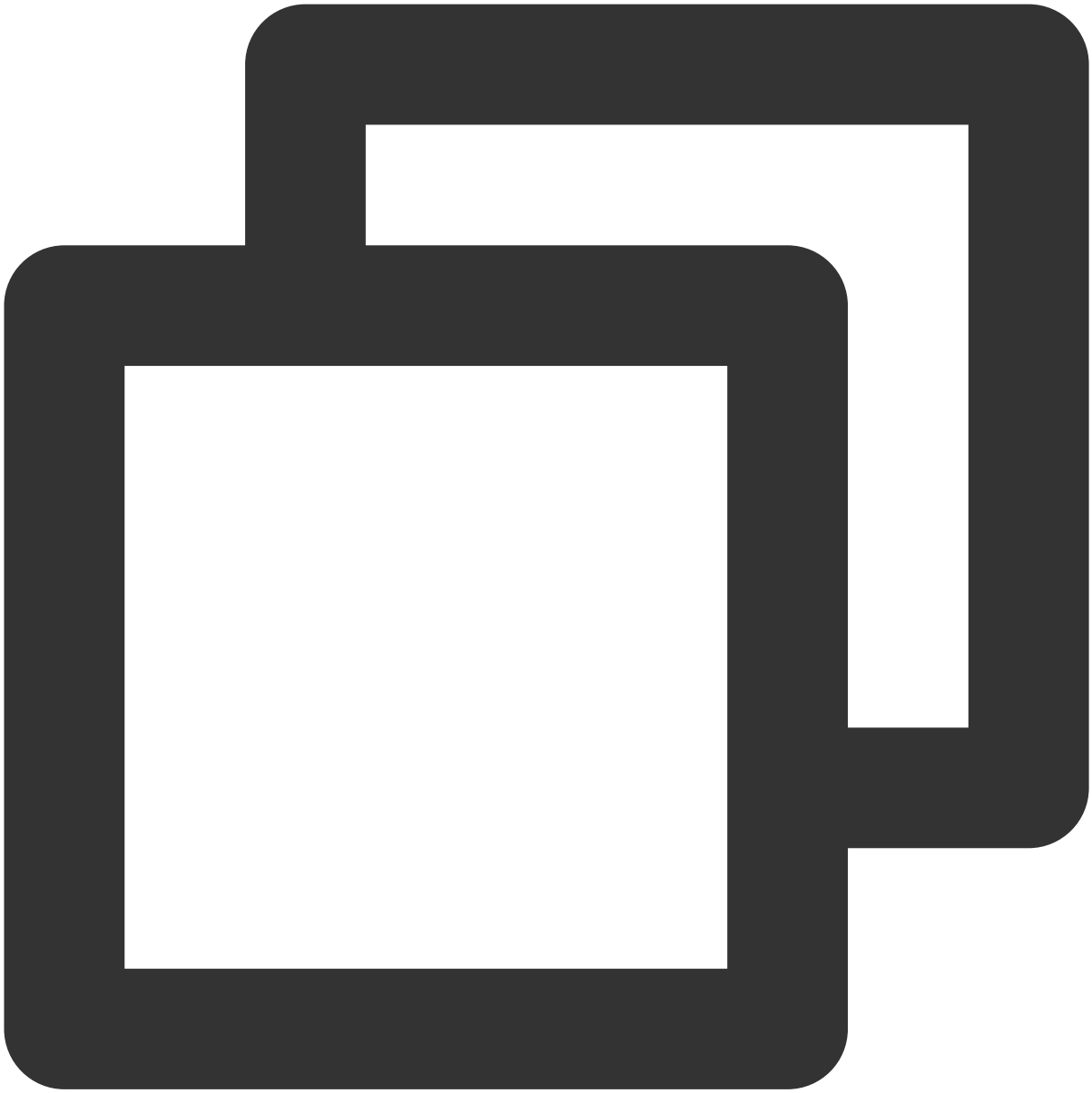
“用户在线状态”功能仅进阶版支持，使用前请确认已开通进阶版套餐。

“用户在线状态”功能需要在 Chat Console 打开用户状态开关，使用前请确认开关已经打开。

开启/关闭用户在线状态

“用户在线状态”为默认关闭，您需要按照以下步骤开启：

版权所有：腾讯云计算（北京）有限责任公司



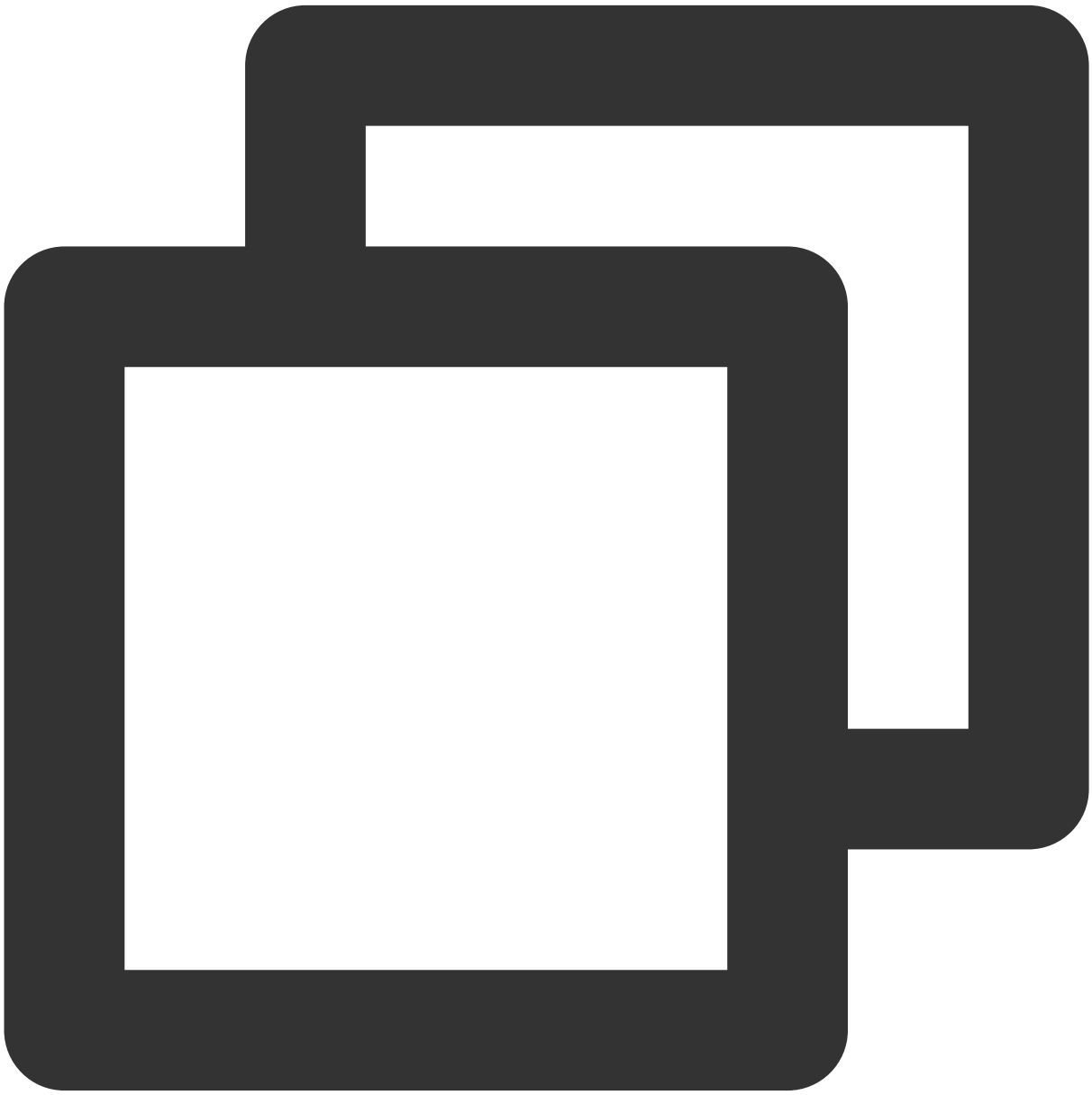
```
import { TUIUserService } from "@tencentcloud/chat-uikit-engine";

// open user online status
// This interface is only valid when called after successful login
TUIUserService.switchUserStatus({ displayOnlineStatus: true });

// close user online status
// This interface is only valid when called after successful login
TUIUserService.switchUserStatus({ displayOnlineStatus: false });
```

注意：

以上接口 `TUIUserService.switchUserStatus` **仅在登录成功后有效**，请务必在登录后再调用该接口。
以下是登录后调用该接口开启用户在线状态示例代码：



```
import { TUILogin } from "@tencentcloud/tui-core";
import { TUIUserService } from "@tencentcloud/chat-uikit-engine";

TUILogin.login(loginInfo).then((res: any) => {
  TUIUserService.switchUserStatus({ displayOnlineStatus: true });
});
```

扩展资料：TUIKit 内部是如何实现“用户在线状态”功能的？

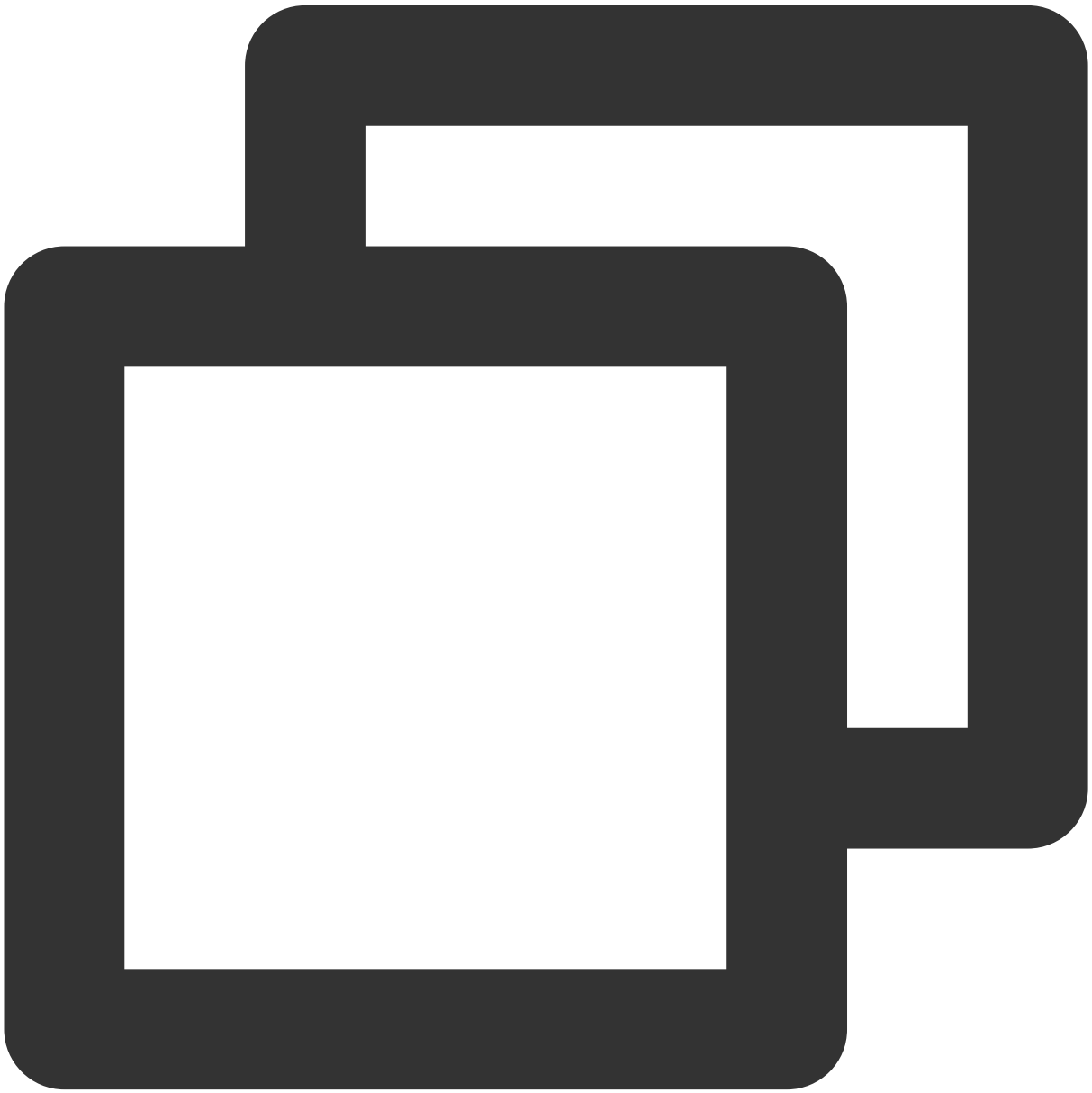
说明：

以下内容仅为辅助阅读资料，用户在线状态功能已在 TUIKit 中默认包含，不需要用户手动实现。

`TUIConversion` 与 `TUIContact` 组件中均支持“用户在线状态”功能，以下以 `TUIContact` 为例进行讲解：

1. 监听用户在线状态列表变化

在 `TUIKit/components/TUIContact/contact-list/index.vue` 中， 监听用户在线状态列表变化：

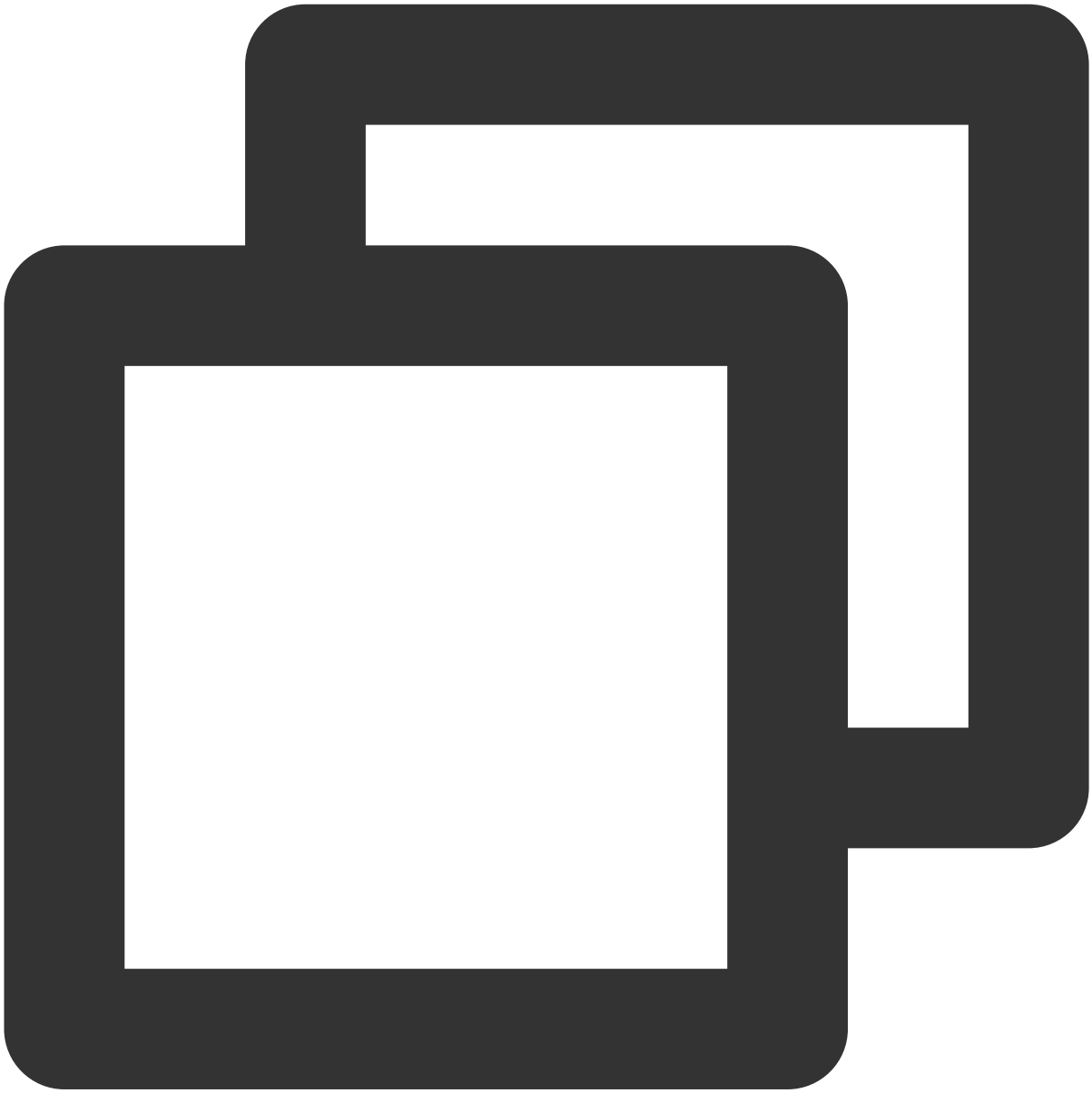


```
TUIStore.watch(StoreName.USER, {
  ...
  displayOnlineStatus: (status: boolean) => {
    displayOnlineStatus.value = status;
  },
  userStatusList: (list: Map<string, IUserStatus>) => {
    list?.size && (userOnlineStatusMap.value = Object.fromEntries(list?.entries()))
  },
});
```

2. 用户在线状态展示

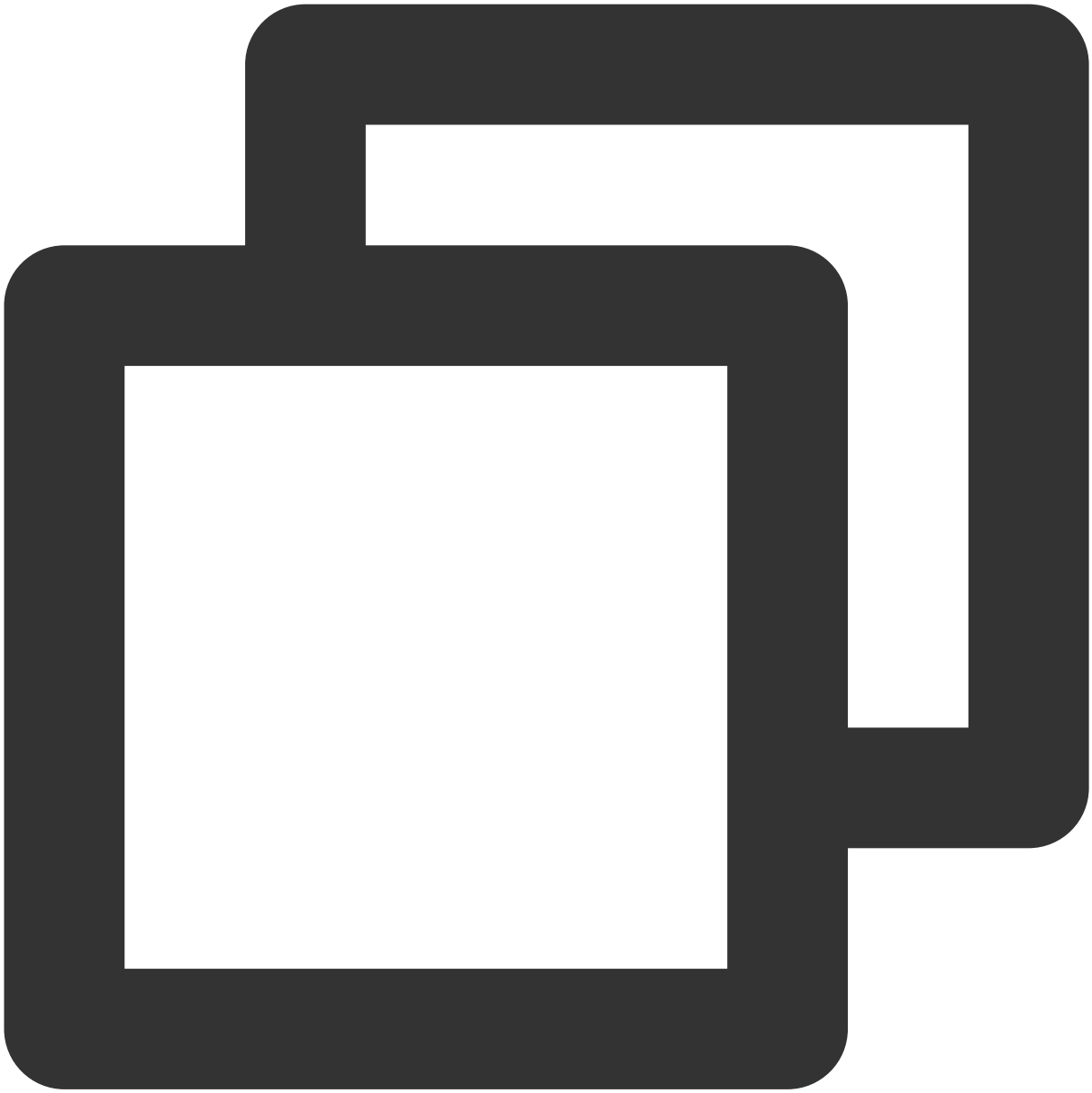
在 `TUIKit/components/TUIContact/contact-list/contact-list-item/index.vue` 中：

2.1 解析该用户在线状态：



```
function getOnlineStatus(): boolean {
  return (
    props.displayOnlineStatus &&
    props.userOnlineStatusMap &&
    props.item?.userID &&
    props.userOnlineStatusMap?.[props.item.userID]?.statusType === TUIChatEngine.TY
  );
};
```

2.2 展示该用户在线状态：

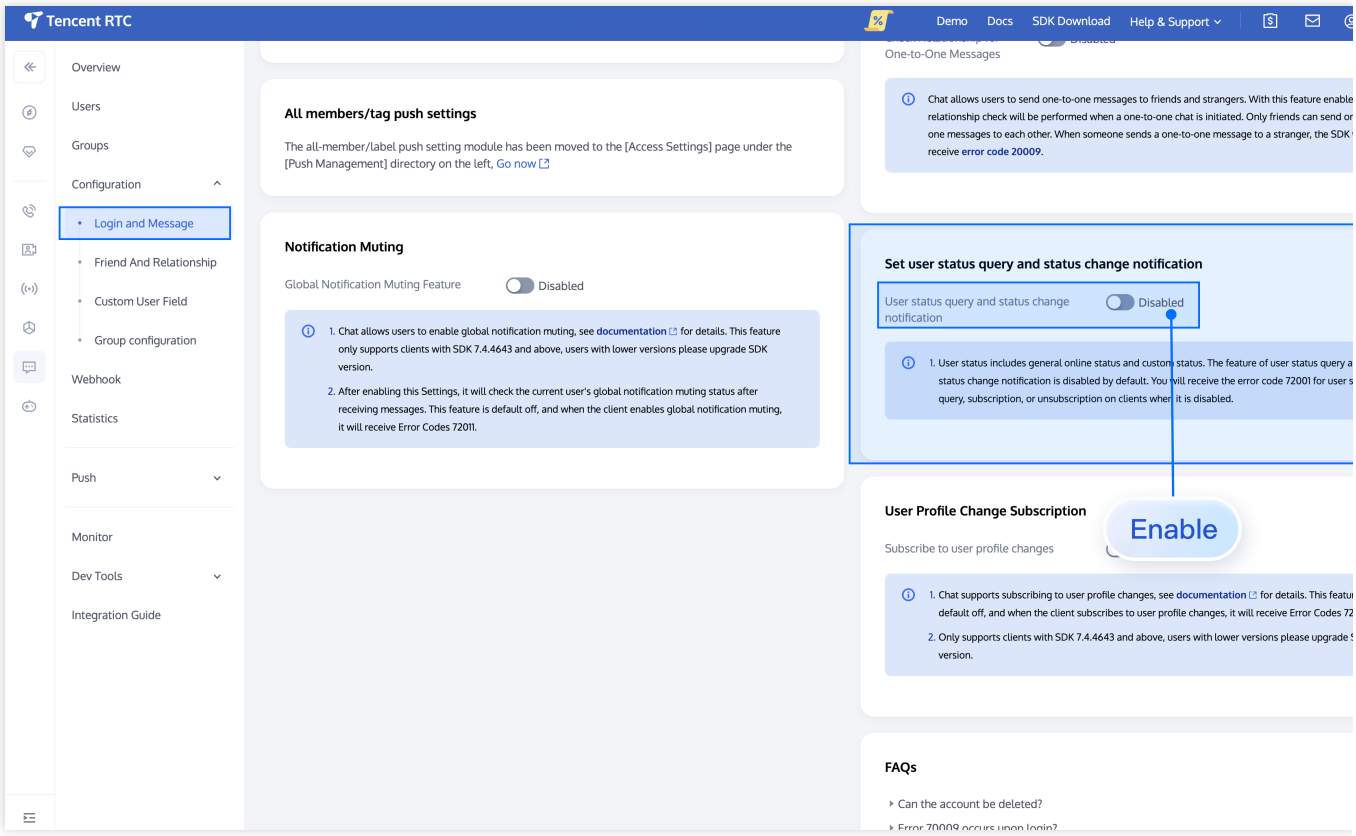


```
<div
  v-if="props.displayOnlineStatus"
  :class="{
    'online-status': true,
    'online-status-online': isOnline,
    'online-status-offline': !isOnline
  }"
></div>
```

常见问题

调用订阅/取消订阅接口时，接口提示 “72001” 的错误码

72001 错误码表示在控制台上并没有开启对应的能力，请登录 Chat Console 打开对应的功能开关。



Error: 套餐包不支持该接口的使用，请升级到旗舰版套餐

“用户在线状态”功能仅旗舰版套餐支持，该报错信息表示您当前的套餐包不支持此能力。

交流与反馈

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

Flutter

最近更新时间：2024-01-31 15:02:18

功能描述

TUIKit 从 v0.1.3 版本开始，支持在会话列表及联系人列表，展示用户在线状态。

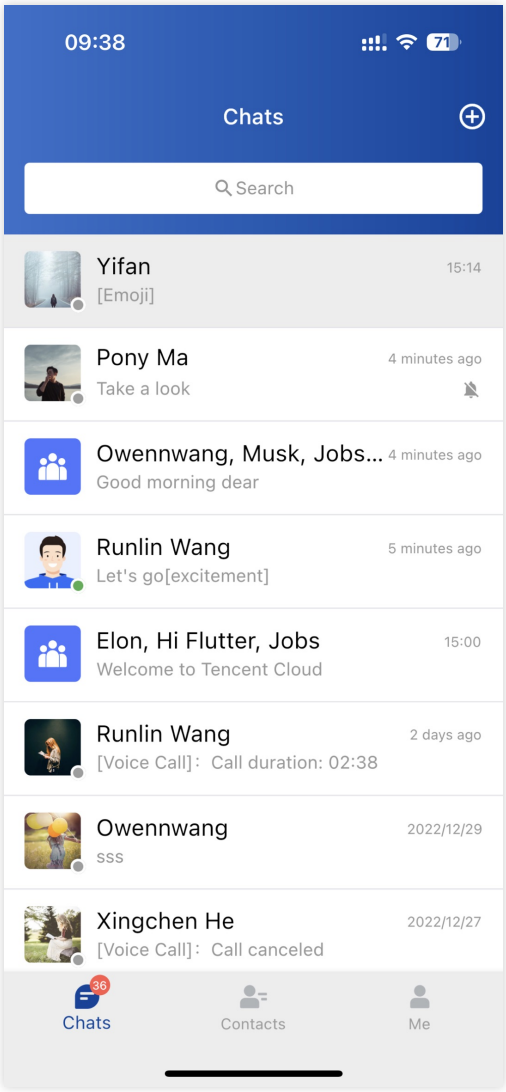
注意：

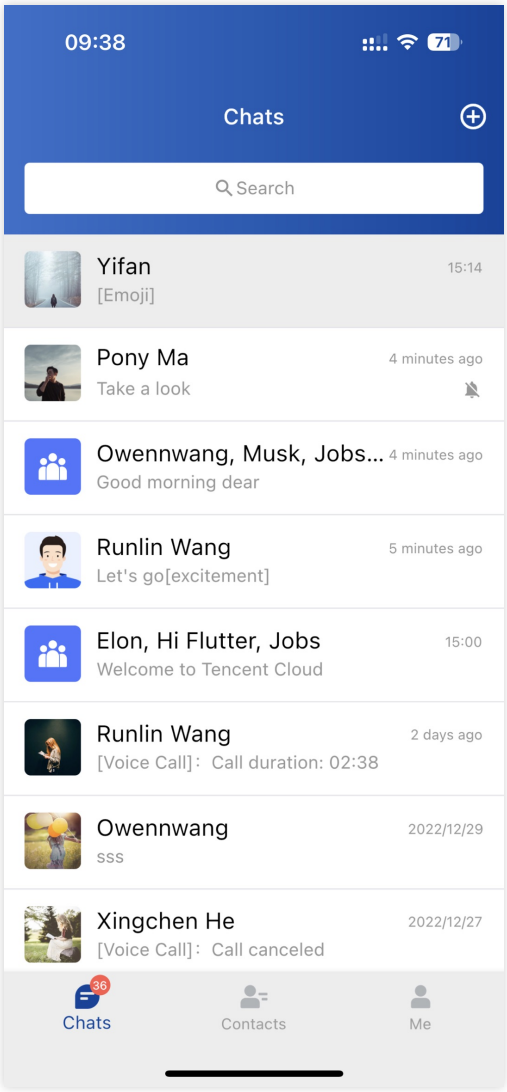
“用户在线状态”功能仅旗舰版套餐支持，使用前请确认。

效果展示

会话列表

开启“显示用户在线状态”	关闭“显示用户在线状态”

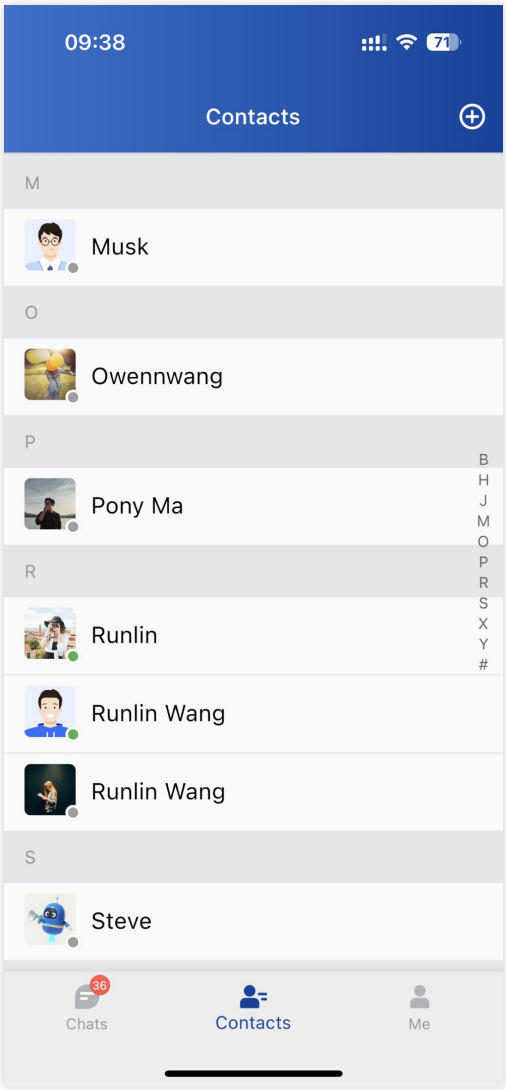


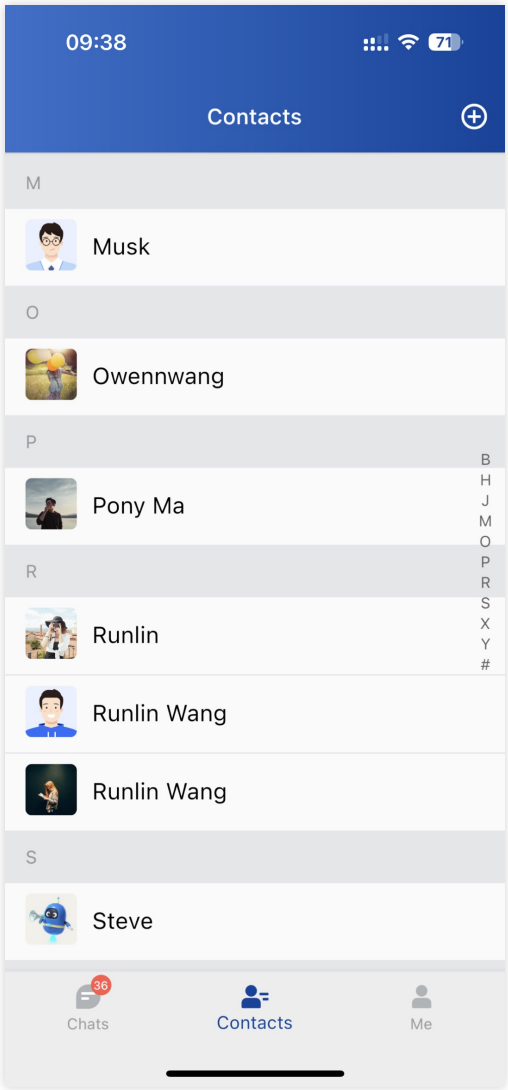


联系人列表

开启“显示用户在线状态”	关闭“显示用户在线状态”

版权所有：腾讯云计算（北京）有限责任公司

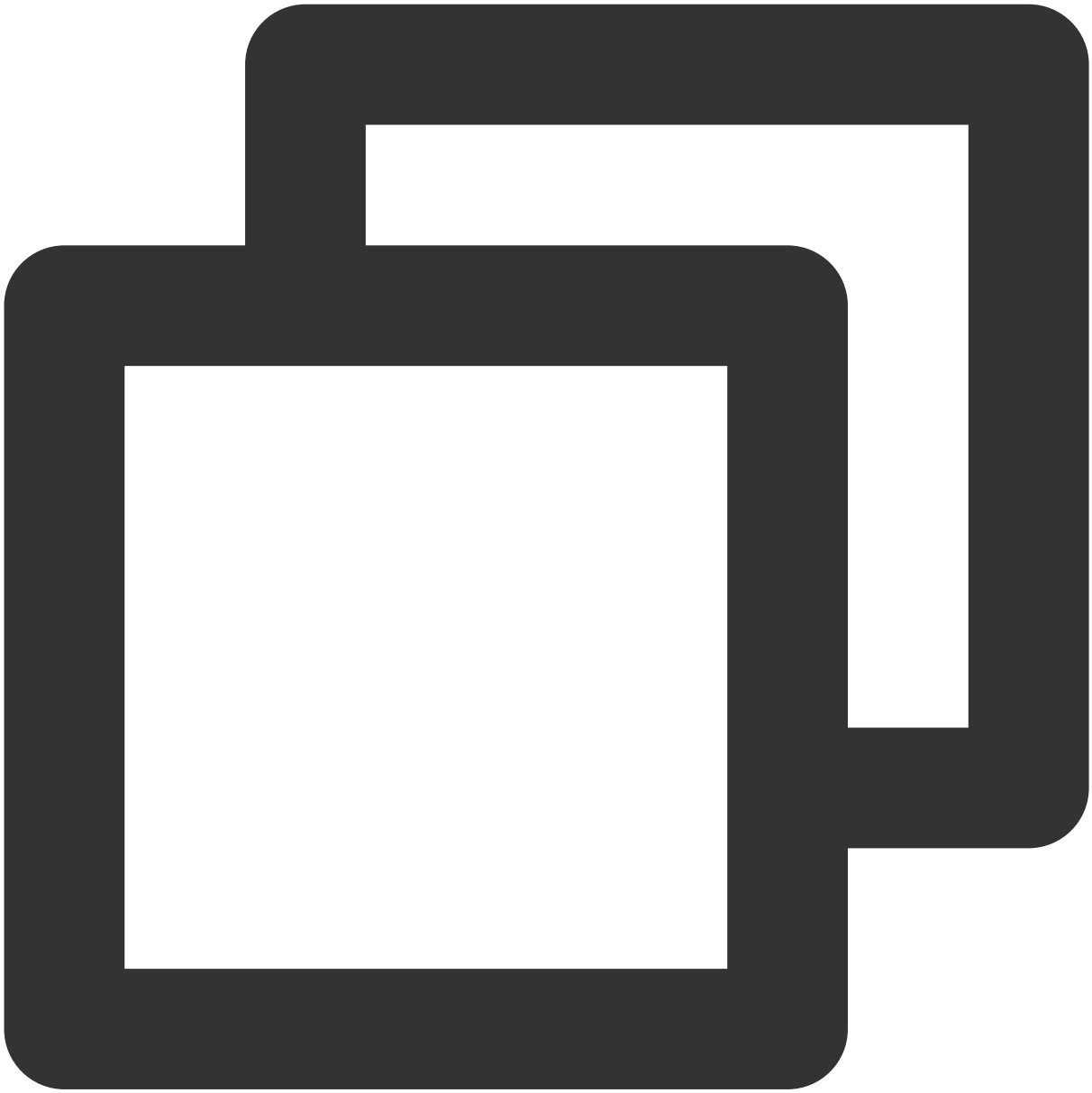




使用此功能

请在初始化TUIKit时，通过配置TUIKit全局 `TIMUIKitConfig` 中在线状态功能字段 `isShowOnlineStatus` ，来控制此功能开启或关闭。

版权所有：腾讯云计算（北京）有限责任公司



```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();

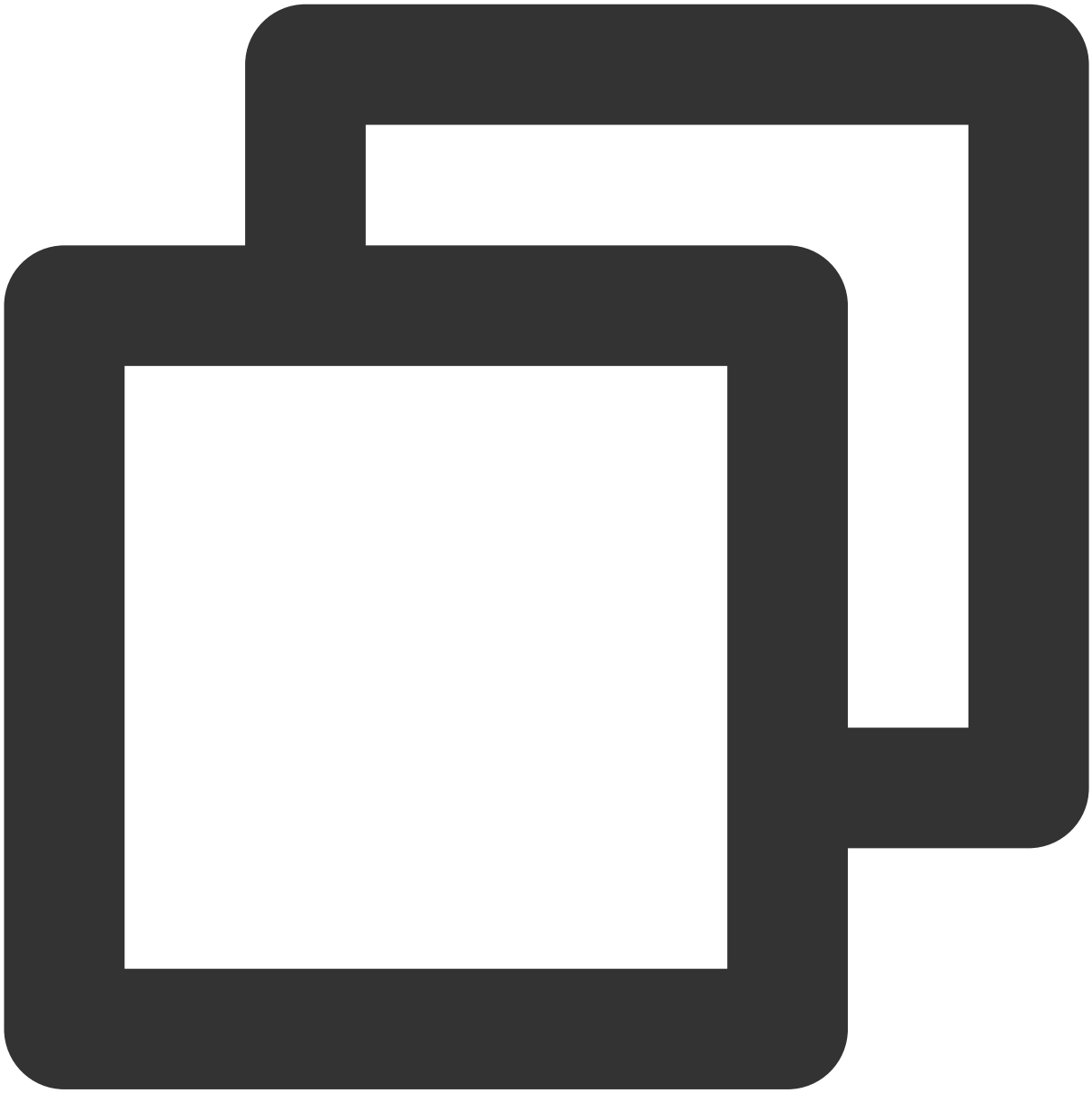
_coreInstance.init(
  config: const TIMUIKitConfig(
    isShowOnlineStatus: true 或 false,  // 添加此行
    // ... 其他TUIKit全局配置
  ),
  // ... 其他启动配置
);
```

此配置是后续配置的总开关，请在此状态打开后，再继续开启后续页面在线状态。

会话列表用户在线状态

TIMUIKitConversation 提供会话列表功能。

在 TIMUIKitConversation 顶层提供了“用户在线状态”功能开关 **isShowOnlineStatus**， 其类型为 boolean，默认为 `true` 。

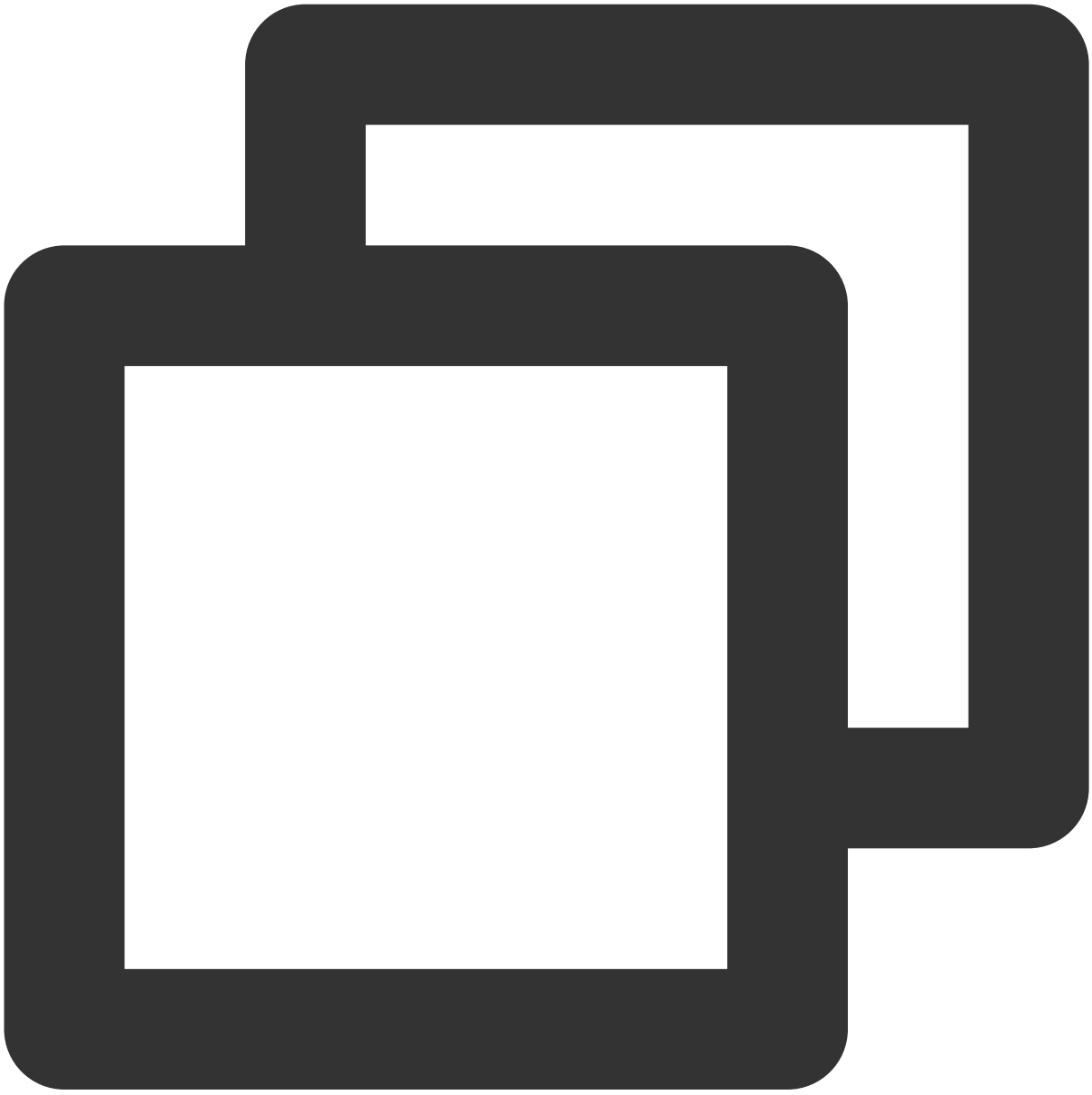


```
TIMUIKitConversation(
  isShowOnlineStatus: true 或 false,
  // ... 其他 TIMUIKitConversation 配置
)
```

联系人列表用户在线状态

TIMUIKitContact 提供联系人列表功能。

在 TIMUIKitContact 顶层提供了“用户在线状态”功能开关 **isShowOnlineStatus**， 其类型为 boolean，默认为 `true` 。



```
TIMUIKitContact(
  isShowOnlineStatus: true 或 false,
  // ... 其他 TIMUIKitContact 配置
)
```

联系我们

如果您在接入使用过程中有任何疑问，请通过如下方式联系我们。

Telegram Group

WhatsApp Group

版权所有：腾讯云计算（北京）有限责任公司

对方正在输入

Android

最近更新时间：2024-03-21 14:37:57

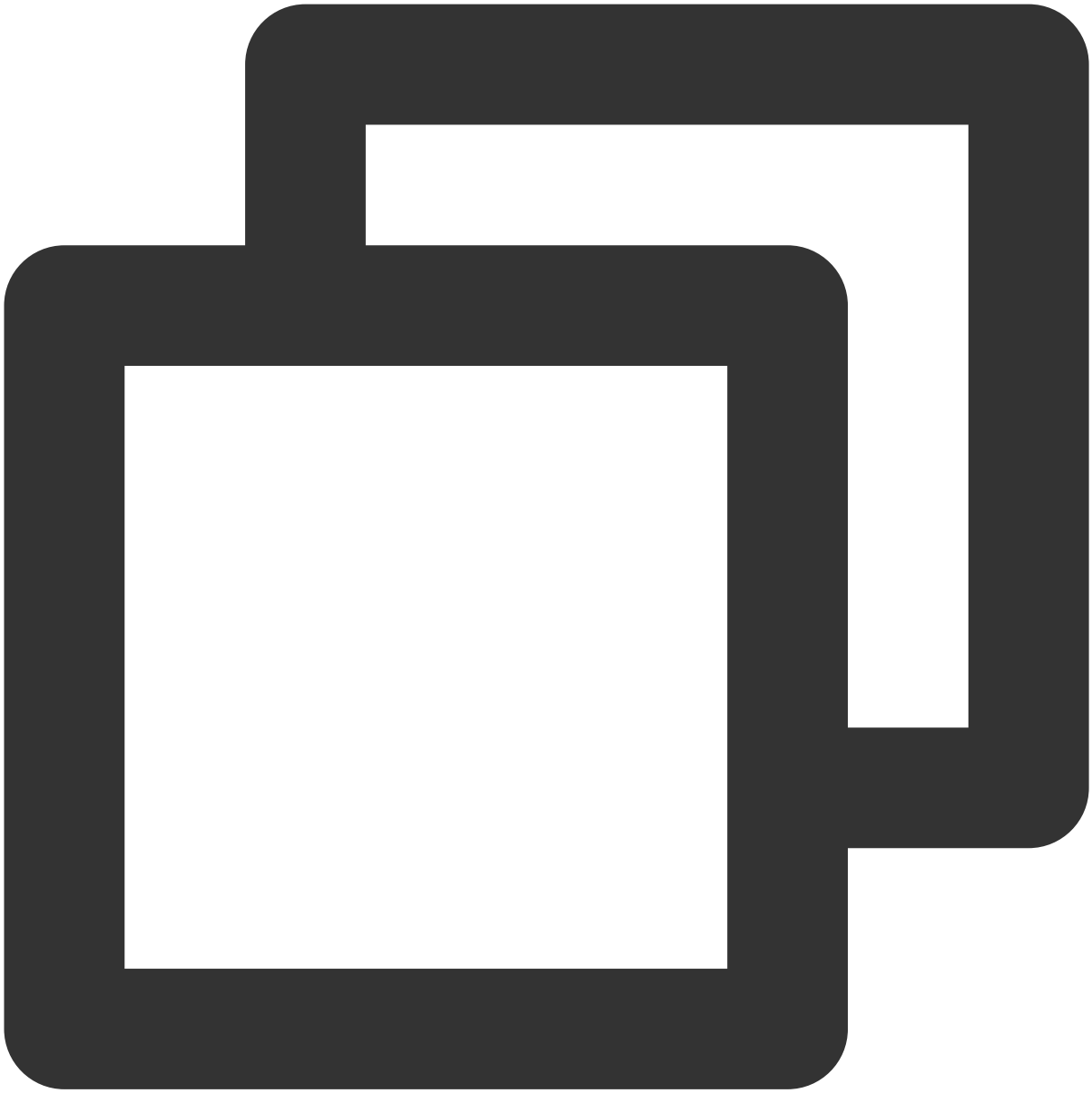
功能描述

TUIKit 从 6.5.2803 版本开始支持在经典版 UI 中单聊会话“对方正在输入”功能。
本功能使用 IMSDK 在线消息 能力实现。

开启“对方正在输入”	关闭“对方正在输入”

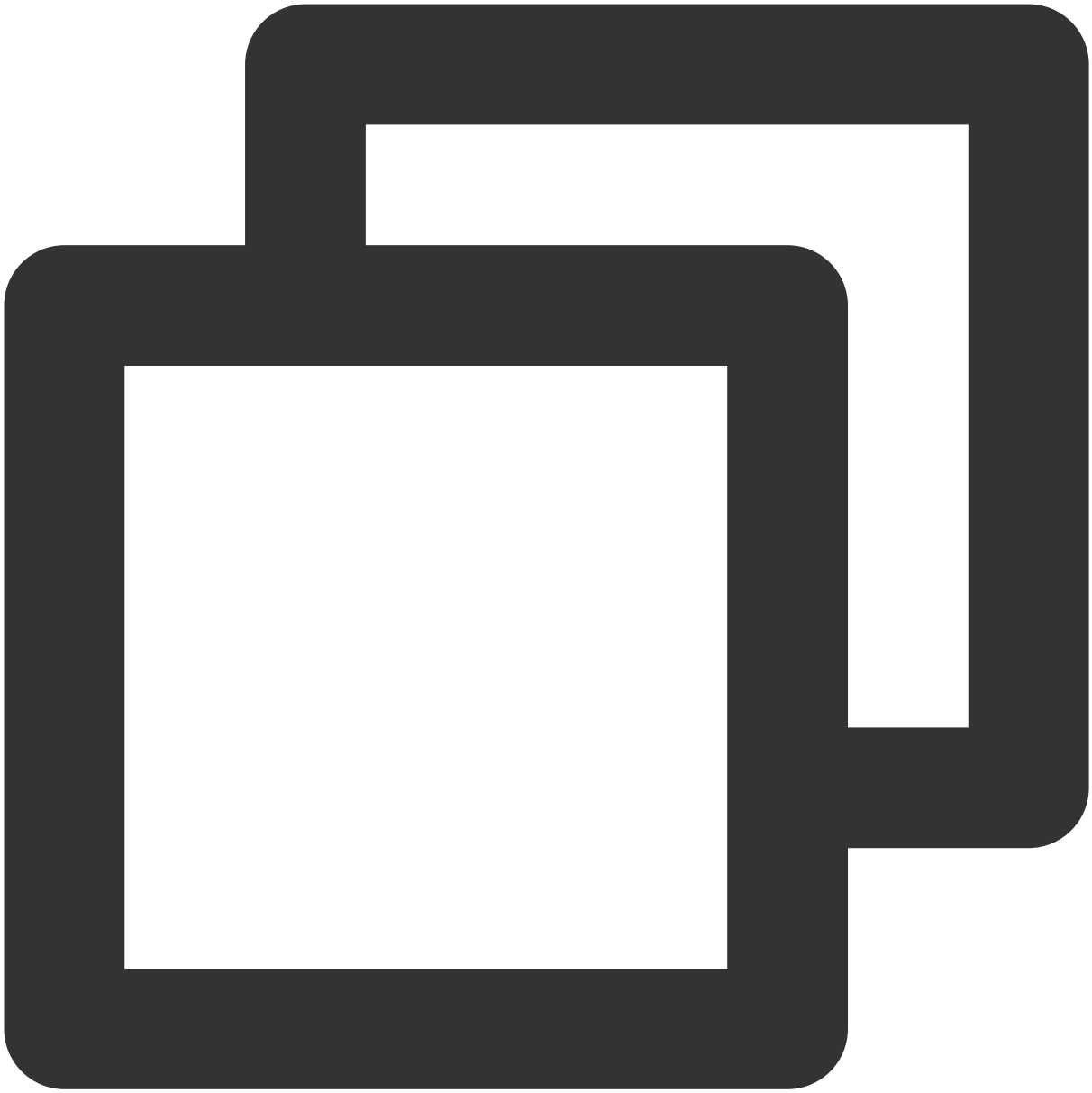
关闭对方正在输入

在 `TUIChat` 组件中的 GeneralConfig.java 文件里，提供了“用户正在输入”功能开关 **enableTypingStatus** , 其类型为 boolean，默认为 `true` 。



```
public class GeneralConfig {
    private boolean enableTypingStatus = true;
}
```

如果想关闭对方正在输入功能，只需把 **enableTypingStatus** 的默认值改为 `false` ，或者在聊天页面初始化之前调用以下方法来关闭。



```
TUIChatConfigs.getConfigs().getGeneralConfig().setEnableTypingStatus(false);
```

常见问题

为什么开启开关后没有对方正在输入提示？

在单聊会话中显示“对方正在输入...”的规则是：对方在30秒内向您发送过消息且当前正在输入文字。

交流与反馈

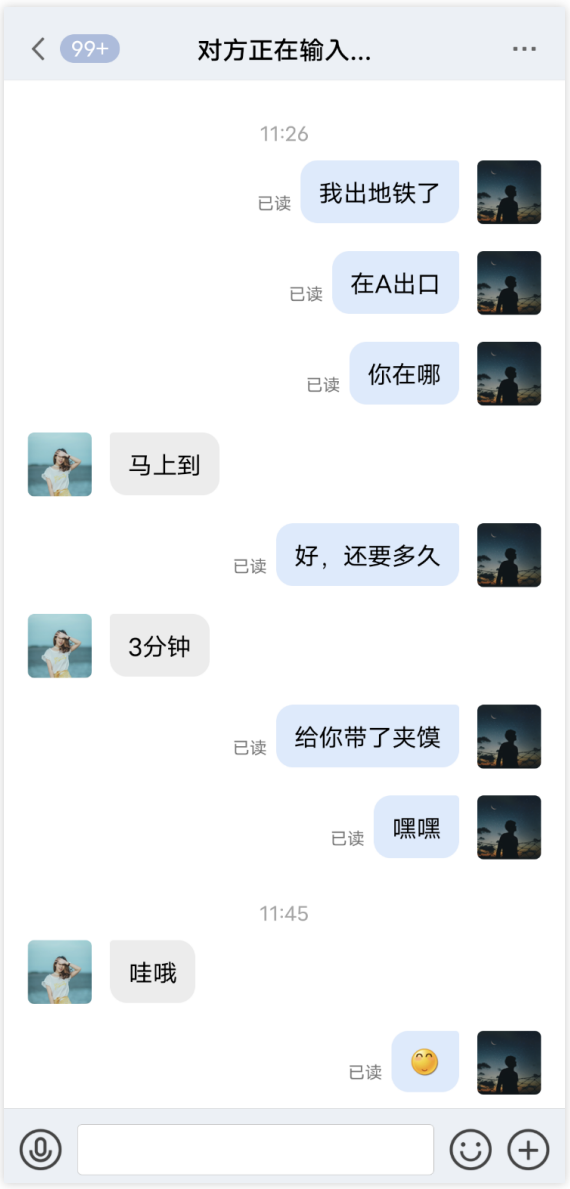
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

iOS

最近更新时间：2024-03-21 14:38:30

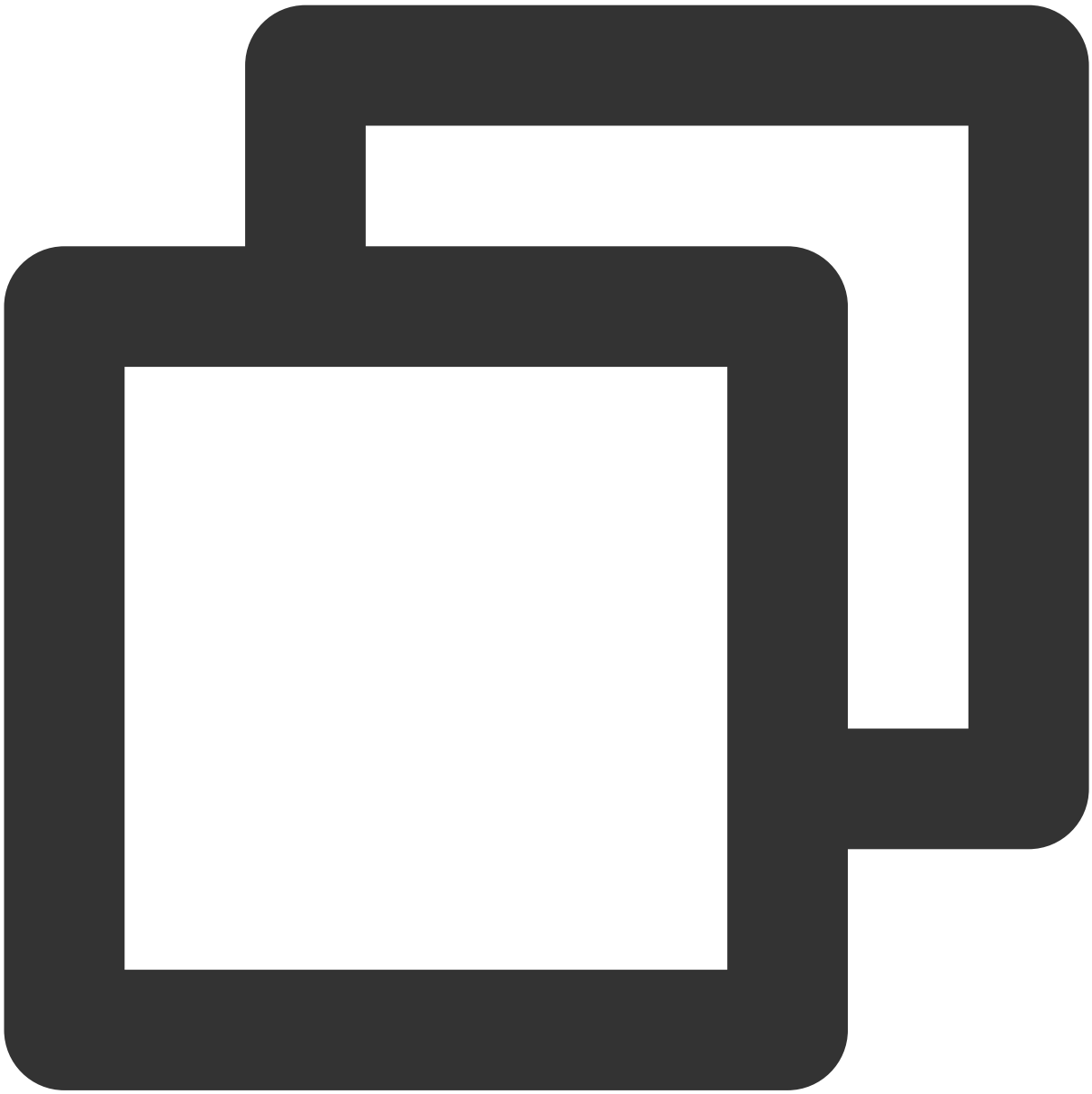
功能描述

TUIKit 从 6.5.2803 版本开始支持在经典版 UI 中单聊会话“对方正在输入”功能。
本功能使用 IMSDK 在线消息 能力实现。

开启“对方正在输入”	关闭“对方正在输入”
	

关闭对方正在输入

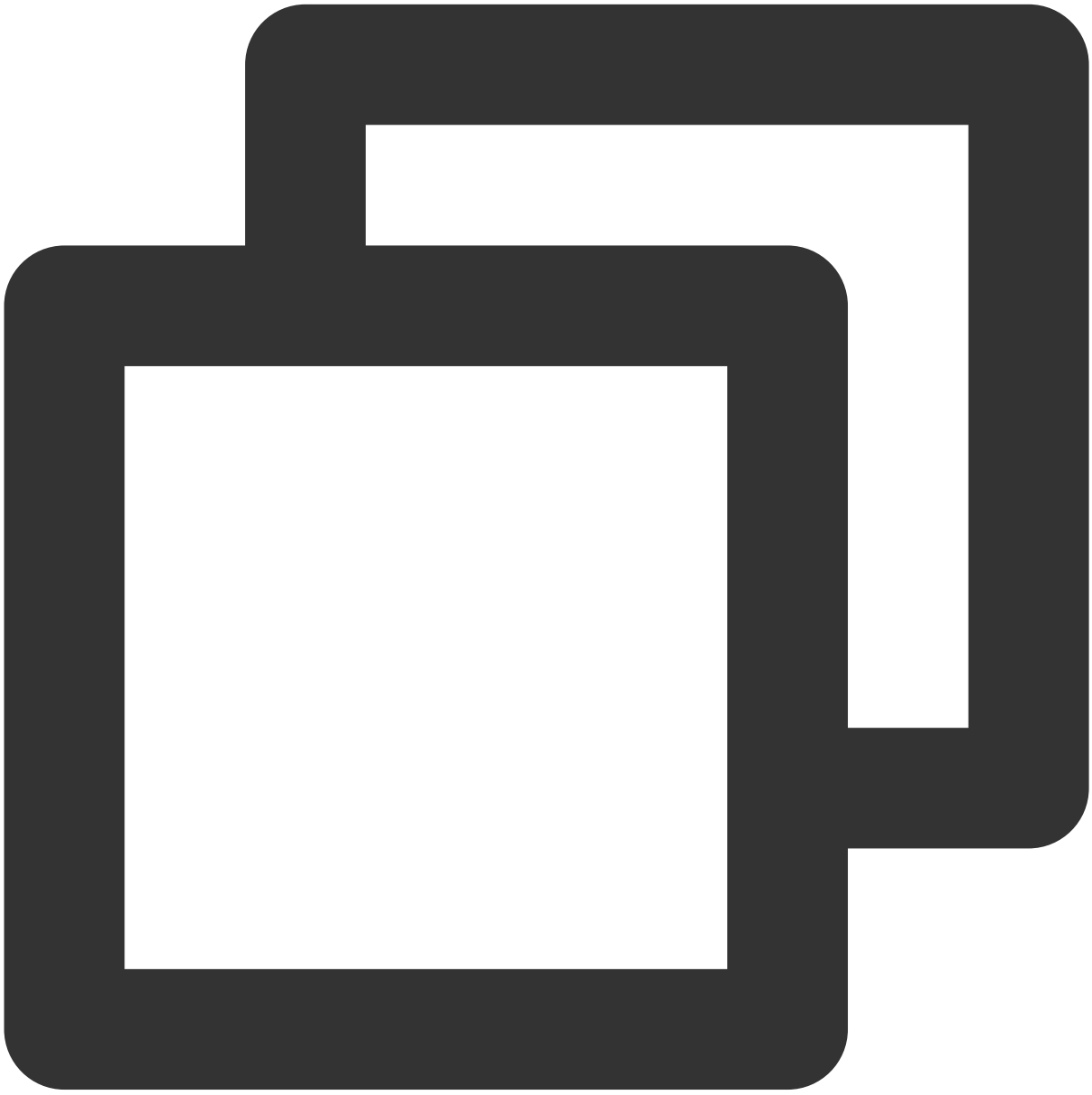
在 `TUIChat` 组件中的 TUIChatConfig 文件里，提供了“用户正在输入”功能开关 **enableTypingStatus** , 其类型为 BOOL，默认为 `YES` 。



```
- (id)init
{
    self = [super init];
    if(self){
      self.enableTypingStatus = YES;
```

```
    }
    return self;
}
```

如果想关闭对方正在输入功能，只需把 **enableTypingStatus** 的默认值改为 `NO` ，或者在聊天页面初始化之前调用以下方法来关闭。



```
TUIChatConfig.defaultConfig.enableTypingStatus = NO;
```

常见问题

为什么开启开关后没有对方正在输入提示？

在单聊会话中显示“对方正在输入...”的规则是：对方在30秒内向您发送过消息且当前正在输入文字。

交流与反馈

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

Web & H5 & Uniapp (Vue)

最近更新时间：2024-06-12 17:44:13

功能描述

@tencentcloud/chat-uikit-vue 从 v2.0.0 版本开始，已支持 C2C 会话“对方正在输入”功能， 效果如下：

显示“对方正在输入...”的规则：

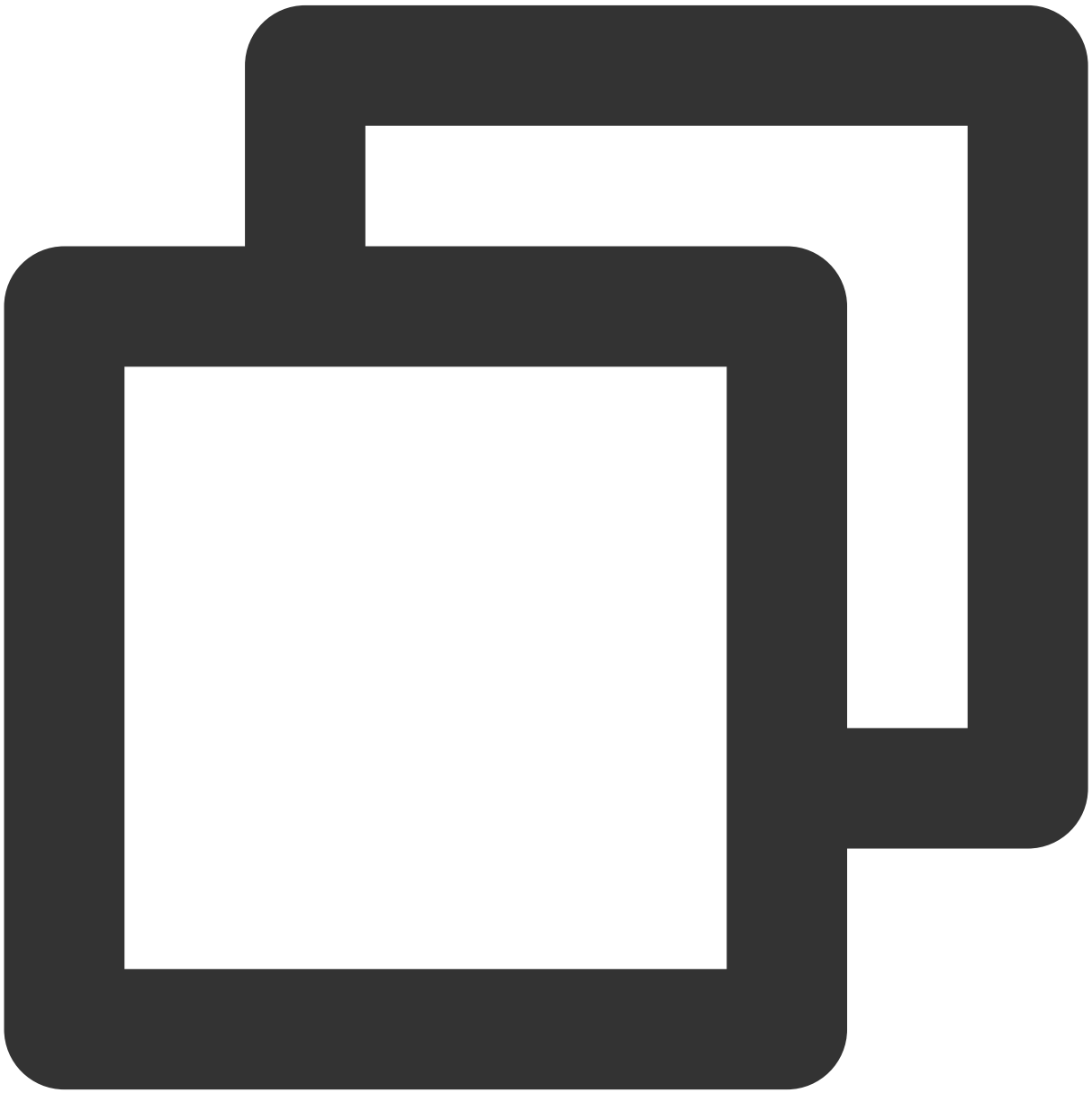
1. 开启“对方正在输入”开关（默认已开启）。

版权所有：腾讯云计算（北京）有限责任公司

2. 在当前 C2C 会话中，对方在30秒内向您发送过消息且当前正在输入文字。

开启/关闭对方正在输入

“对方正在输入”为默认开启，您无需重复按照以下步骤进行开启。



```
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";

// 开启对方正在输入
```

```
// enable typing
TUIStore.update(StoreName.APP, "enableTyping", true);

// 关闭对方正在输入
// disable typing
TUIStore.update(StoreName.APP, "enableTyping", false);
```

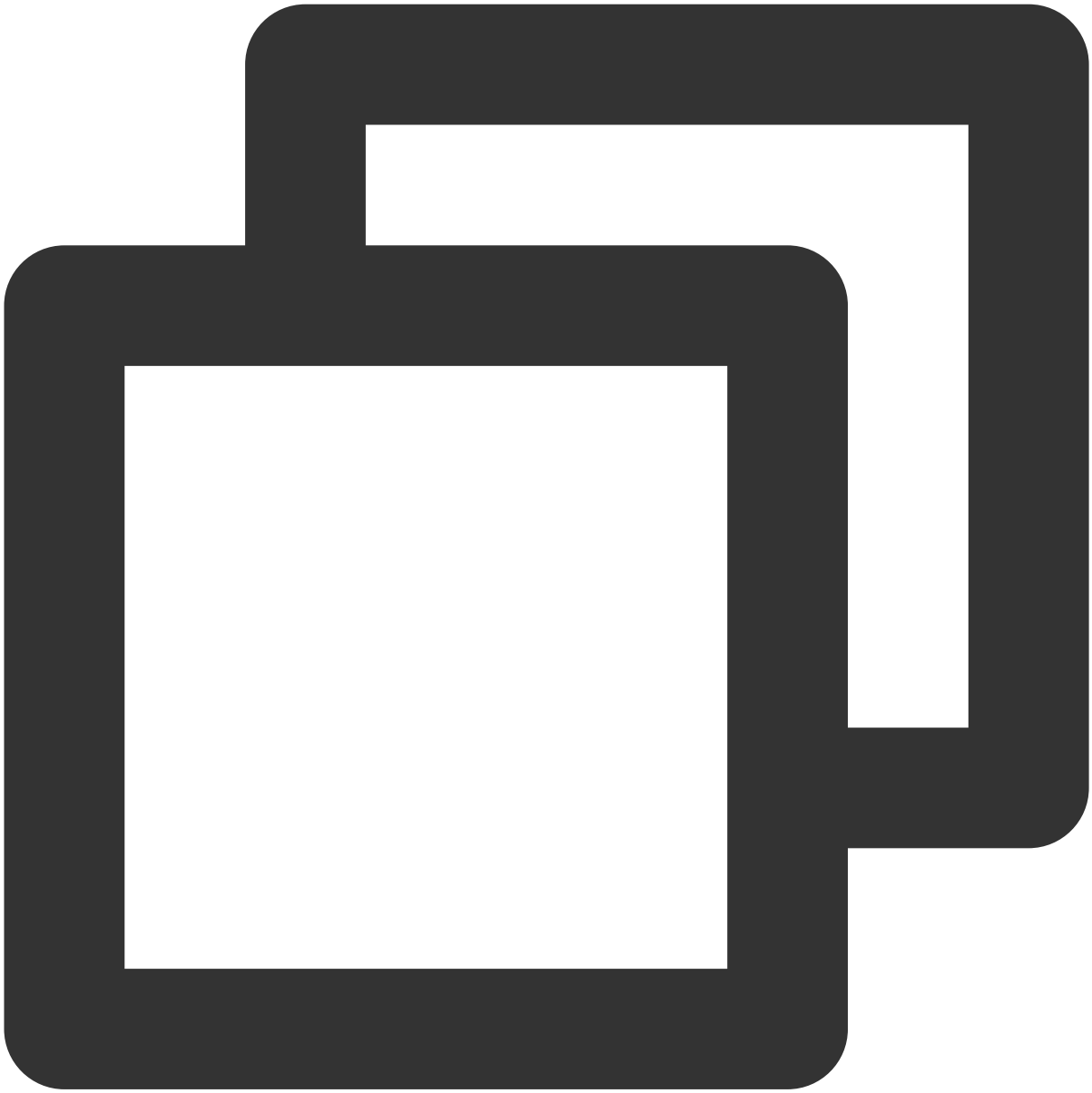
扩展资料: TUIKit 内部是如何实现“对方正在输入...”的？

说明：

以下内容仅为辅助阅读资料，对方正在输入功能已在 TUIKit 中默认包含，不需要用户手动实现。

1. 发送端：监听输入的开始与结束，向对端发送在线消息

在 `TUIKit/components/TUIChat/message-input/index.vue` ， 可以通过 TUIChatService.enterTypingState() 发送开启输入状态，通过 TUIChatService.leaveTypingState() 发送结束输入状态。



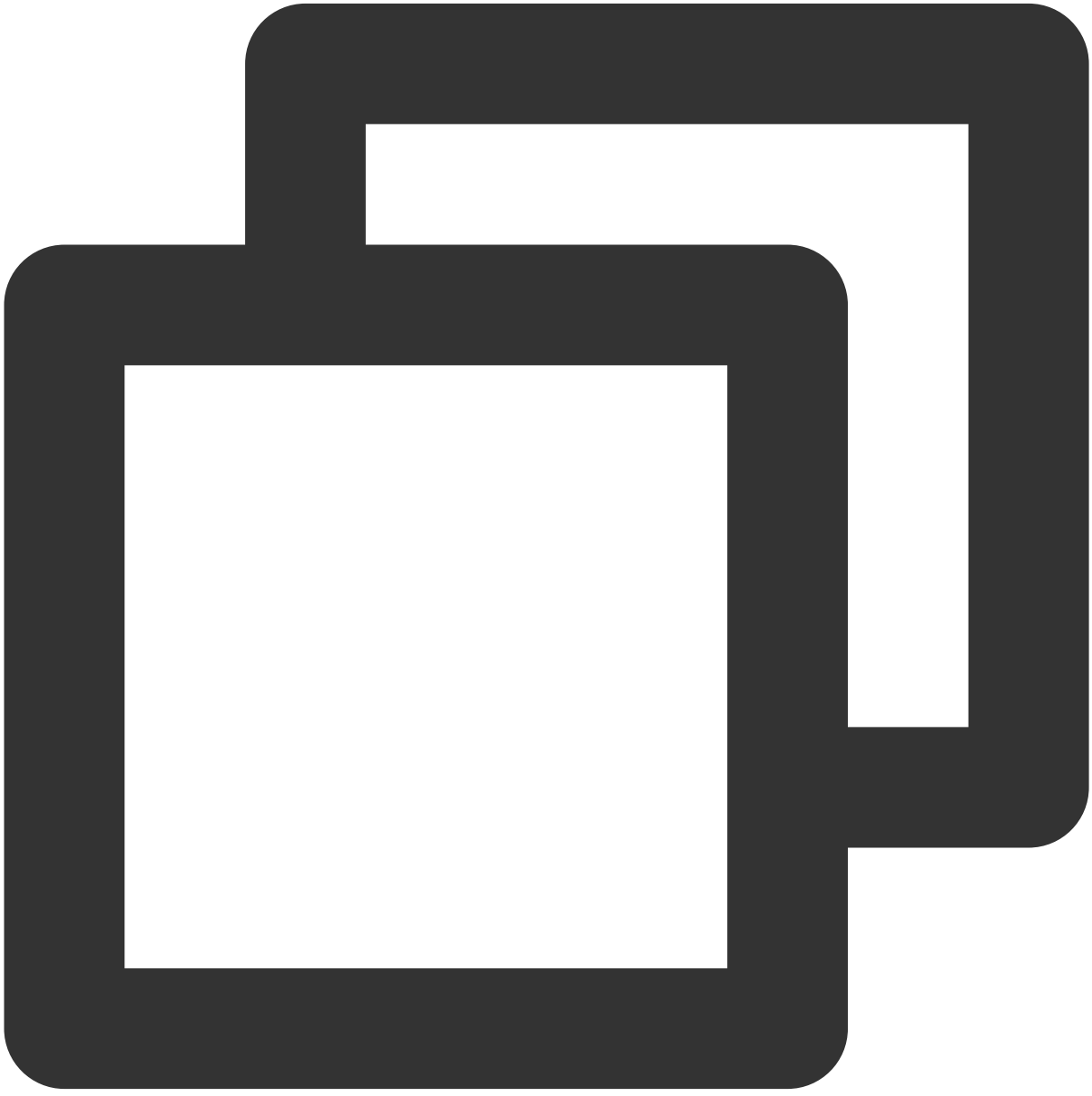
```
// TUIKit/components/TUIChat/message-input/index.vue
const onTyping = (inputContentEmpty: boolean, inputBlur: boolean) => {
  sendTyping(inputContentEmpty, inputBlur);
};

// TUIKit/components/TUIChat/utils/sendMessage.ts
export const sendTyping = (inputContentEmpty: boolean, inputBlur: boolean) => {
  if (!inputContentEmpty && !inputBlur) {
    TUIChatService.enterTypingState();
  } else {
    TUIChatService.leaveTypingState();
```

```
  }
};
```

2. 接收端: 监听发送端输入状态并展示

在 `TUIKit/components/TUIChat/chat-header/index.vue` ，通过监听 typingStatus 监听 C2C 会话中对端输入状态。



```
TUIStore.watch(StoreName.CHAT, {
  typingStatus: (status: boolean) => {
    typingStatus.value = status;
```

```
    switch (typingStatus.value) {
      case true:
        currentConversationName.value =
          TUITranslateService.t("TUIChat.对方正在输入");
        break;
      case false:
        currentConversationName.value =
          currentConversation?.value?.getShowName();
        break;
    }
  },
});
```

常见问题

为什么开启开关后没有对方正在输入提示？显示“对方正在输入...”的规则是什么？

1. 开启“对方正在输入”开关（默认已开启）
2. 在当前 C2C 会话中，对方在30秒内向您发送过消息且当前正在输入文字。

交流与反馈

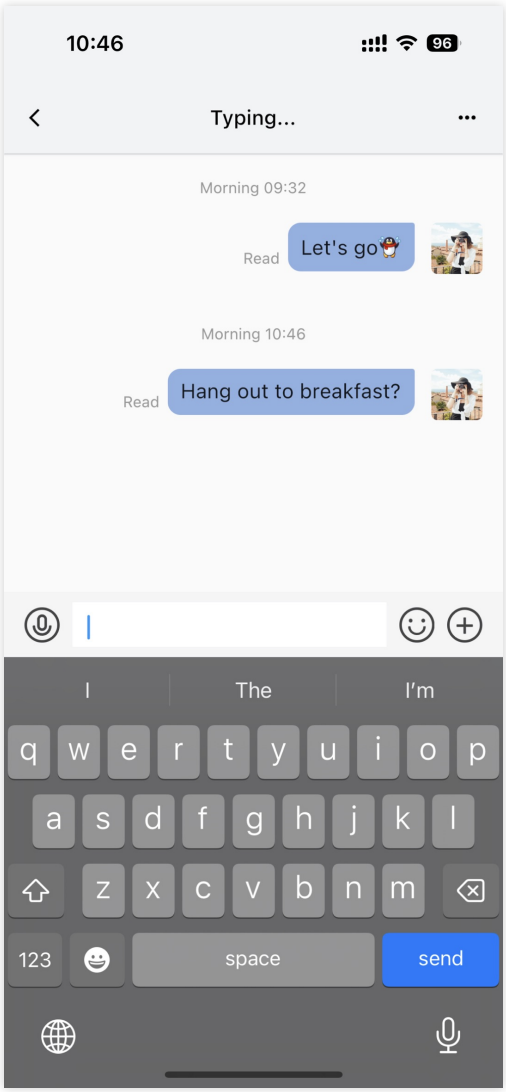
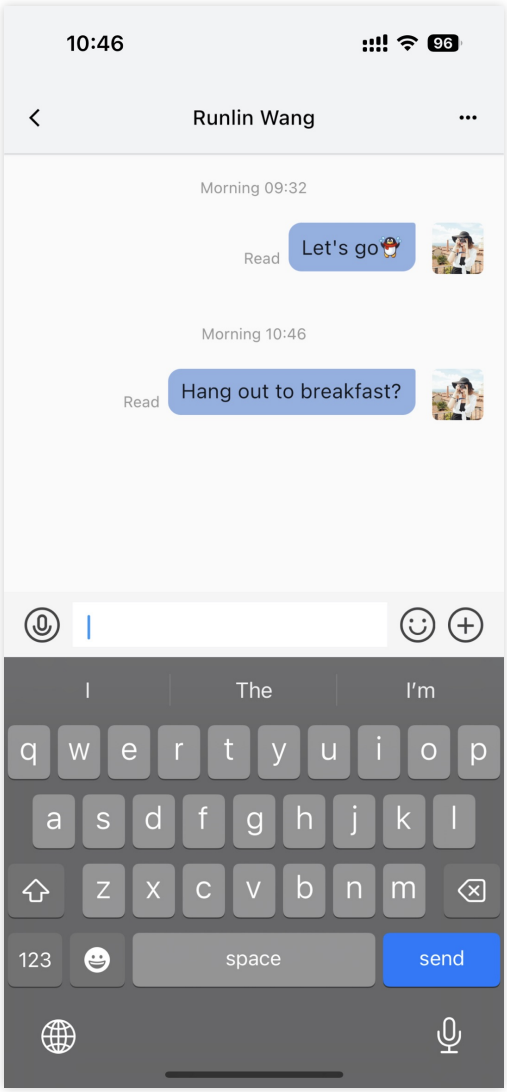
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

Flutter

最近更新时间：2024-01-31 15:02:47

功能描述

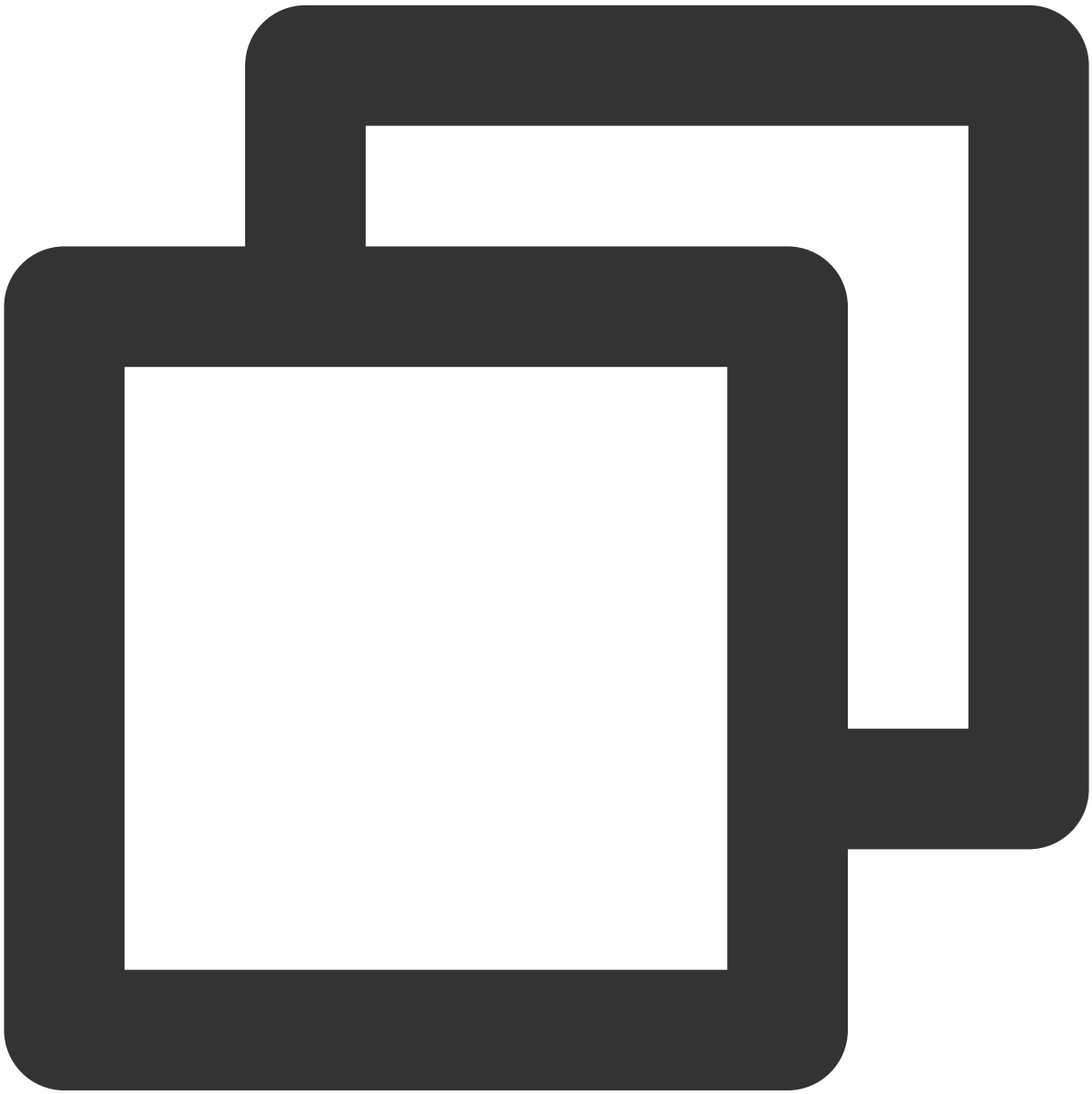
TUIKit 从 v0.1.3 版本开始，支持单聊会话“对方正在输入”功能。
本功能使用更多腾讯云IM SDK在线消息能力实现。

开启“对方正在输入”	关闭“对方正在输入”
	

版权所有：腾讯云计算（北京）有限责任公司

控制对方正在输入能力状态

在 TIMUIKitChat 的配置参数 `config` 中，提供了“对方正在输入”功能开关 **showC2cMessageEditStatus** , 其类型为 boolean，默认为 `true` 。



```
TIMUIKitChat(
  config: TIMUIKitChatConfig(
    showC2cMessageEditStatus: true 或 false,
    // ... 其他 config 配置
  ),
```

```
    // ... 其他 TIMUIKitChat 参数
)
```

联系我们

如果您在接入使用过程中有任何疑问，请通过如下方式联系我们。

Telegram Group

WhatsApp Group

消息已读回执
Android

最近更新时间：2024-07-05 15:24:08

功能描述

已读回执（Read Receipt）用于通知发送人“接收人已经阅读了发送的消息”。当接收人阅读消息后，上报消息已读，后台系统会生成一条通知，并将其发送给发送人，以告知消息已被查看。
在即时通讯工具（WhatsApp、微信等）中，当接收人查看消息时，发送人会看到消息旁边的已读标记，例如蓝色的对勾或“已读”字样。
说明：
“回执”的含义是“回复的收据”，它代表了一种确认接收的凭证。当您发送一条消息，并请求一个回执，您实际上是在请求对方“我想确认你们是否接收并阅读了我的消息”。这个确认就像是一张“收据”，证明您的消息已经被接收。
已读回执有助于确保重要信息已被查看，但也可能引发心理压力和隐私问题，因此我们支持用户关闭已读回执功能。
注意：
1. 该功能仅进阶版支持，请购买进阶版后使用。
2. 群聊消息已读回执从 TUIKit 6.2.2363 及以上版本支持。
3. 单聊消息已读回执从 TUIKit 6.3.2609及以上版本支持。

效果展示

单聊消息已读回执

通过消息上的 ✔ 或者高亮的 ✔✔展示。

群聊消息已读回执

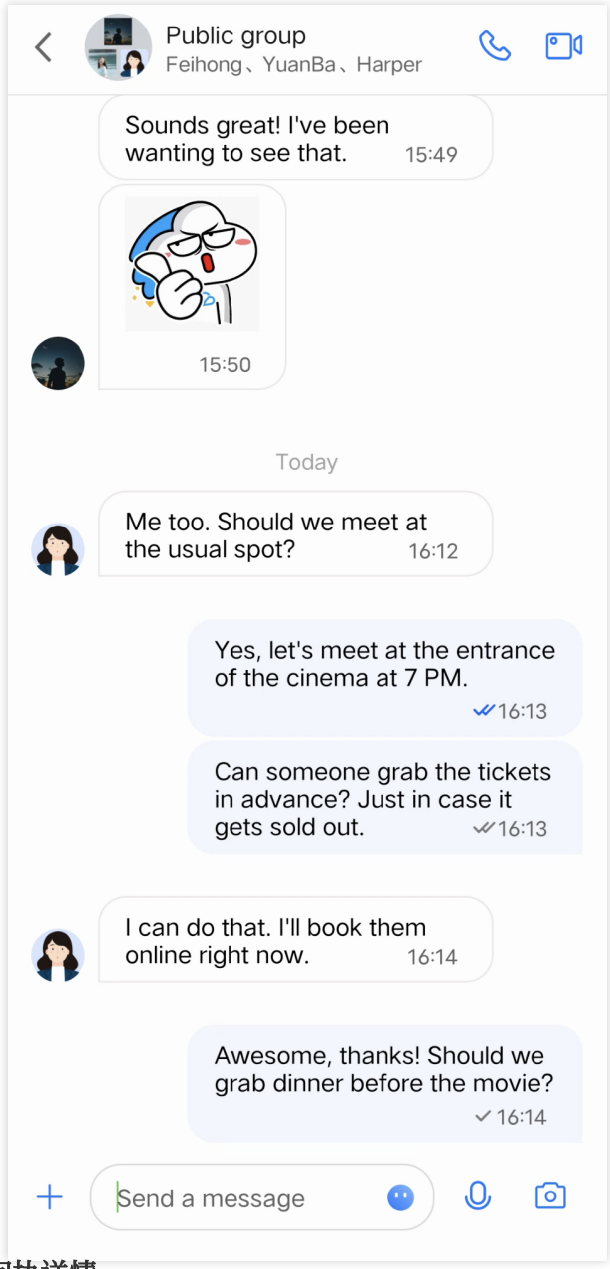
消息上显示消息阅读情况：

无人阅读时，显示 ✔ ；

部分人阅读时，显示灰色的 ✔✔ ；

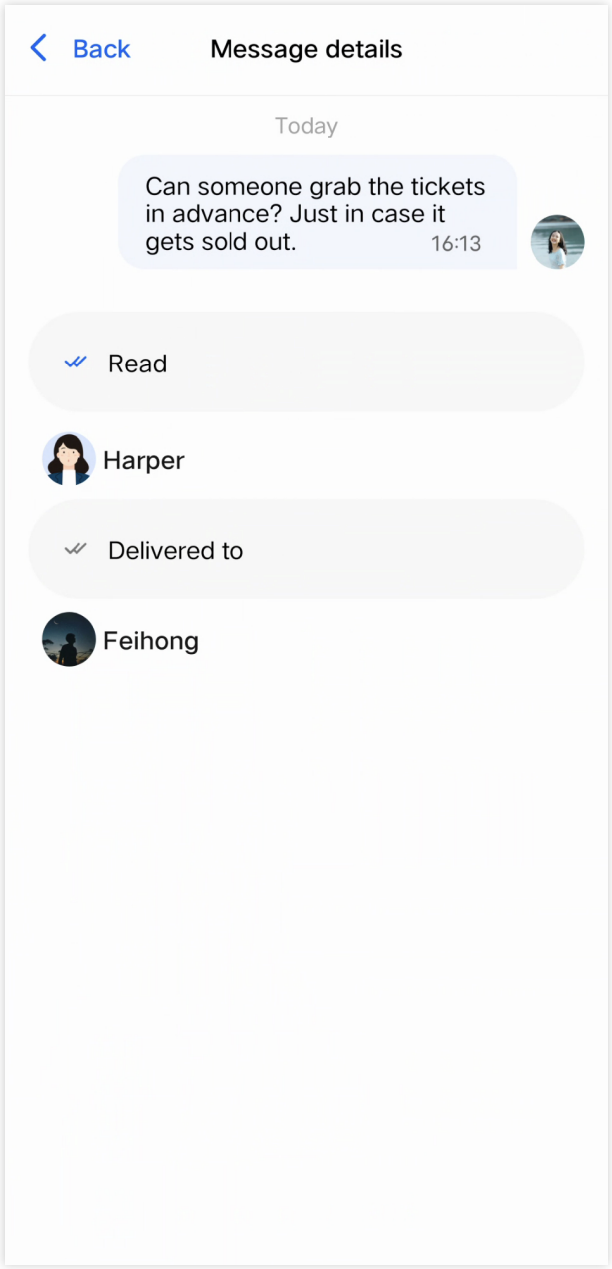
所有人已读时，显示高亮的 ✔✔ 。

消息列表



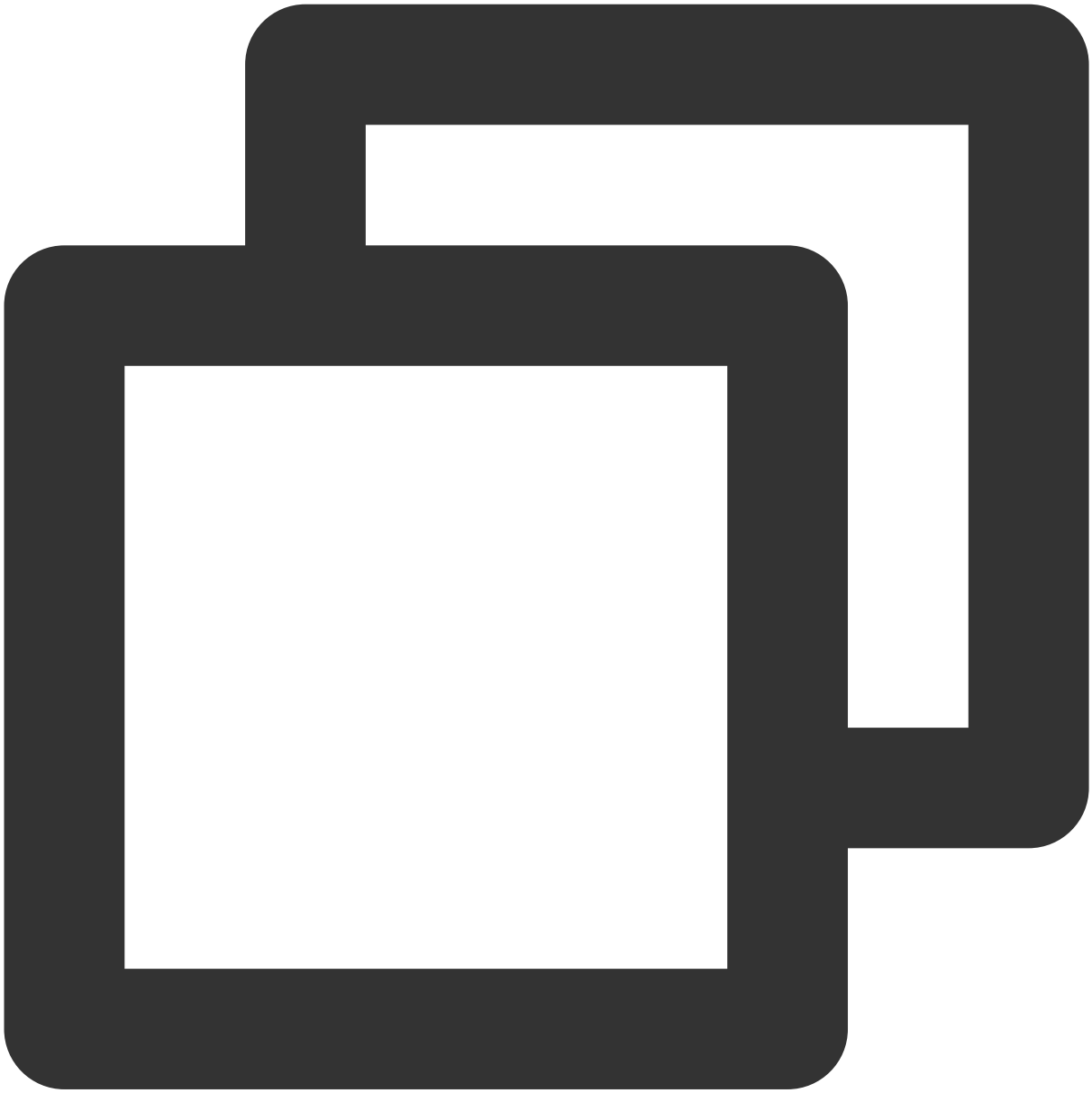
已读回执详情

点击已读状态即可进入已读回执详情页面。



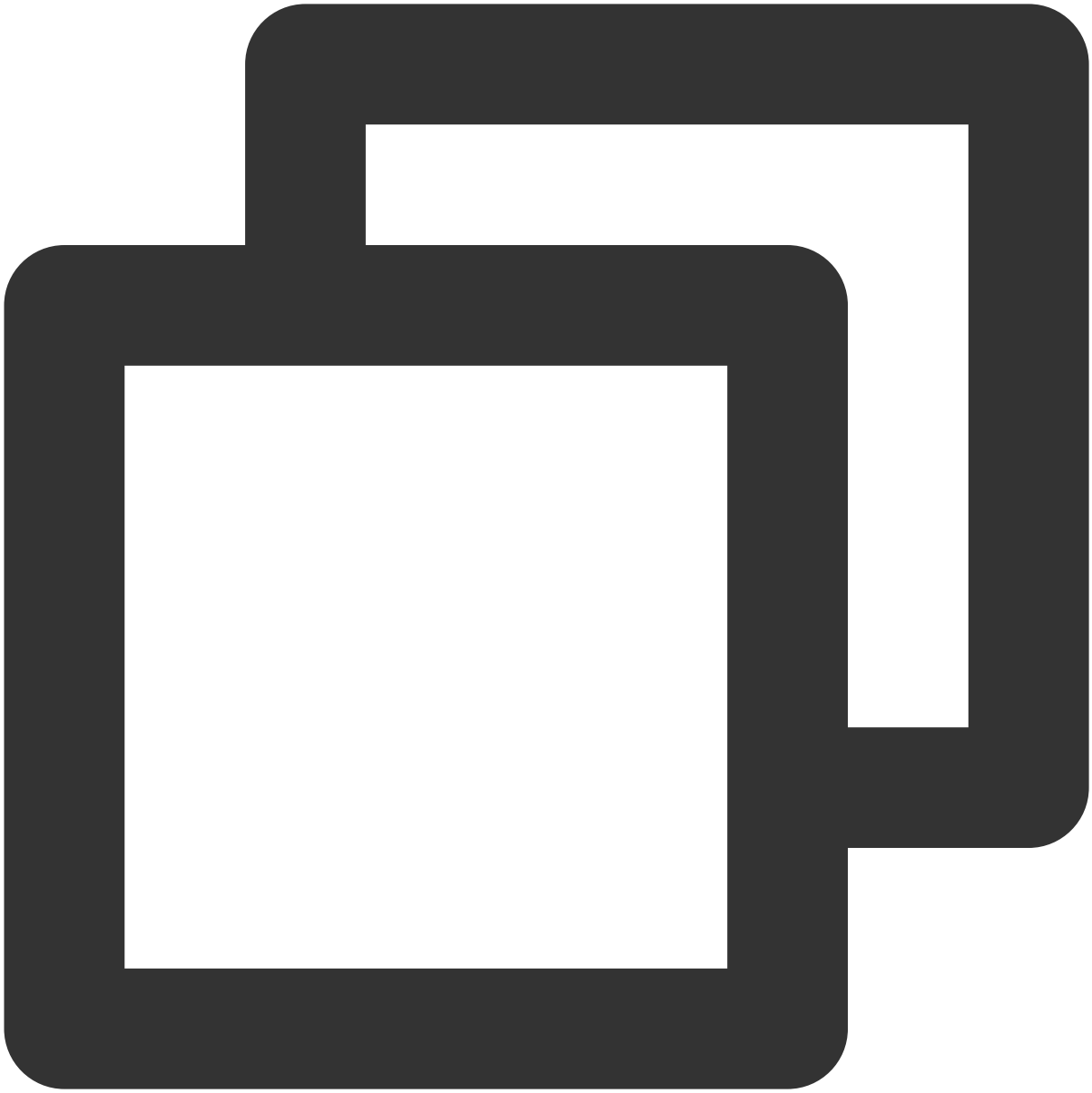
开启消息已读回执

在 `TUIChat` 组件中的 GeneralConfig.java 文件里，提供了“消息已读回执”功能开关 `msgNeedReadReceipt` ，其类型为 boolean，默认为 `false` 。



```
public class GeneralConfig {
    private boolean msgNeedReadReceipt = false;
}
```

如果想开启消息已读回执功能，首先请开通旗舰版套餐包，然后把 `msgNeedReadReceipt` 的默认值改为 `true` ，或者在聊天页面初始化之前调用以下方法来开启。



```
TUIChatConfigs.getConfigs().getGeneralConfig().setMsgNeedReadReceipt(true);
```

常见问题

Error: The usage of this API is not supported by the package. Please upgrade to the premium version.
“消息已读回执” 功能仅进阶版支持，该报错信息表示您当前的套餐包不支持此能力，请 购买进阶版 后使用。

iOS

最近更新时间：2024-07-05 15:24:08

功能描述

已读回执（Read Receipt）用于通知发送人“接收人已经阅读了发送的消息”。当接收人阅读消息后，上报消息已读，后台系统会生成一条通知，并将其发送给发送人，以告知消息已被查看。
在即时通讯工具（WhatsApp、微信等）中，当接收人查看消息时，发送人会看到消息旁边的已读标记，例如蓝色的对勾或“已读”字样。
说明：
“回执”的含义是“回复的收据”，它代表了一种确认接收的凭证。当您发送一条消息，并请求一个回执，您实际上是在请求对方“我想确认你们是否接收并阅读了我的消息”。这个确认就像是一张“收据”，证明您的消息已经被接收。
已读回执有助于确保重要信息已被查看，但也可能引发心理压力和隐私问题，因此我们支持用户关闭已读回执功能。
注意：
1. 该功能仅进阶版支持，请购买进阶版后使用。
2. 群聊消息已读回执从 TUIKit 6.2.2363 及以上版本支持。
3. 单聊消息已读回执从 TUIKit 6.3.2609及以上版本支持。

效果展示

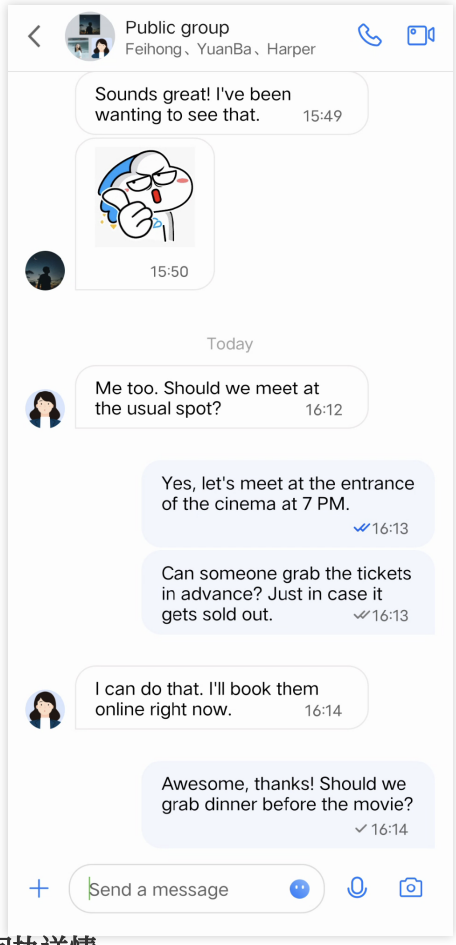
单聊消息已读回执

通过消息上的 ✔ 或者高亮的 ✔✔展示。

群聊消息已读回执

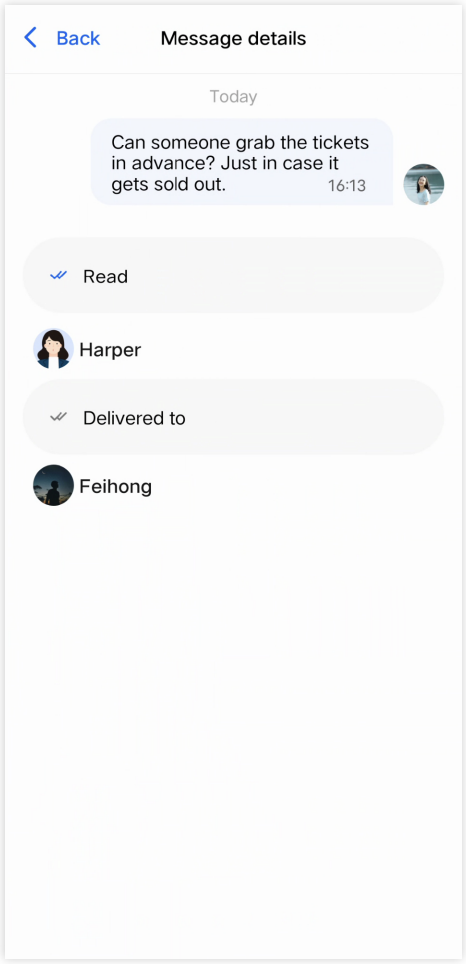
消息上显示消息阅读情况：无人阅读时，显示 ✔ ；部分人阅读时，显示灰色的 ✔✔ ；所有人已读时，显示高亮的 ✔✔ 。

消息列表



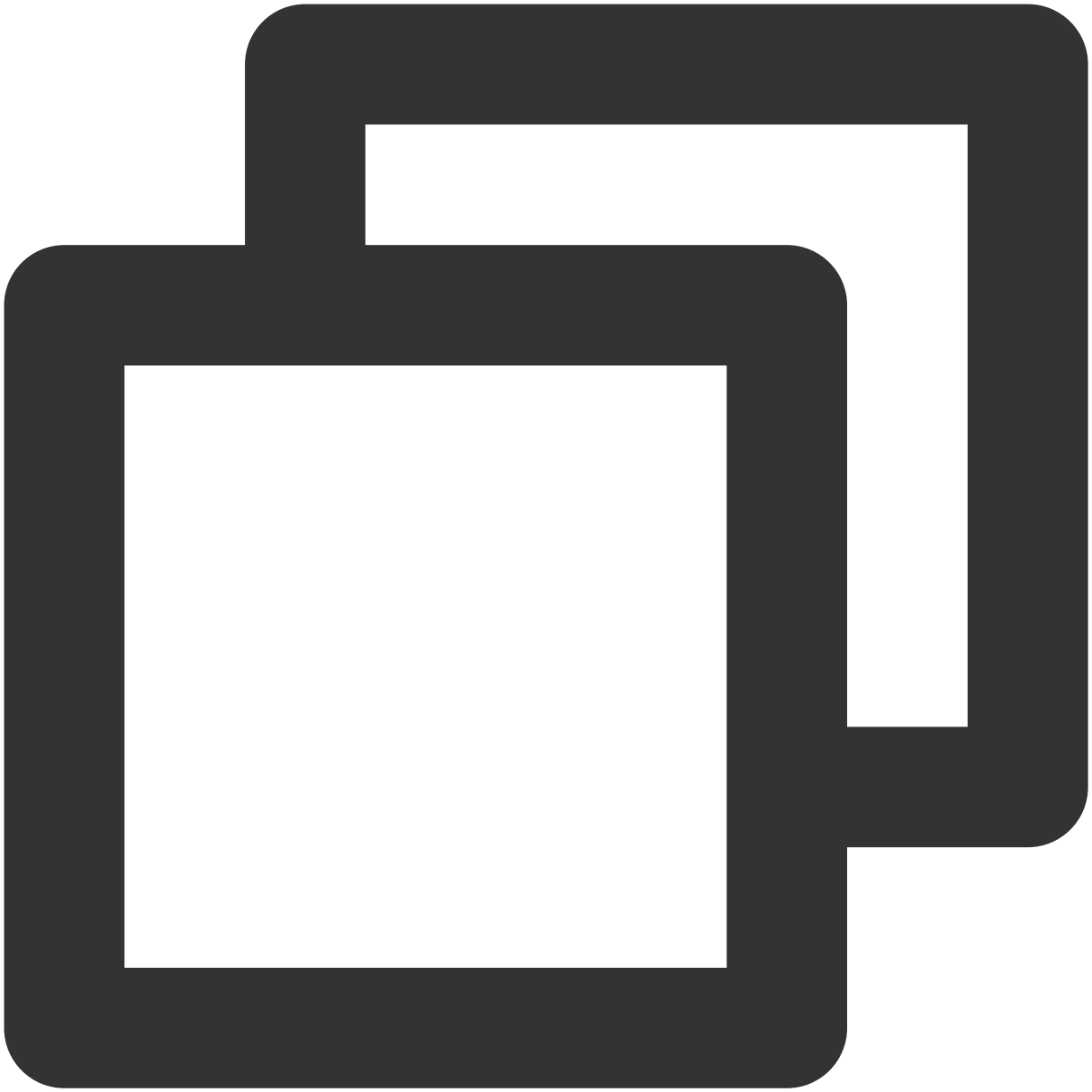
已读回执详情

单击已读状态即可进入已读回执详情页面。



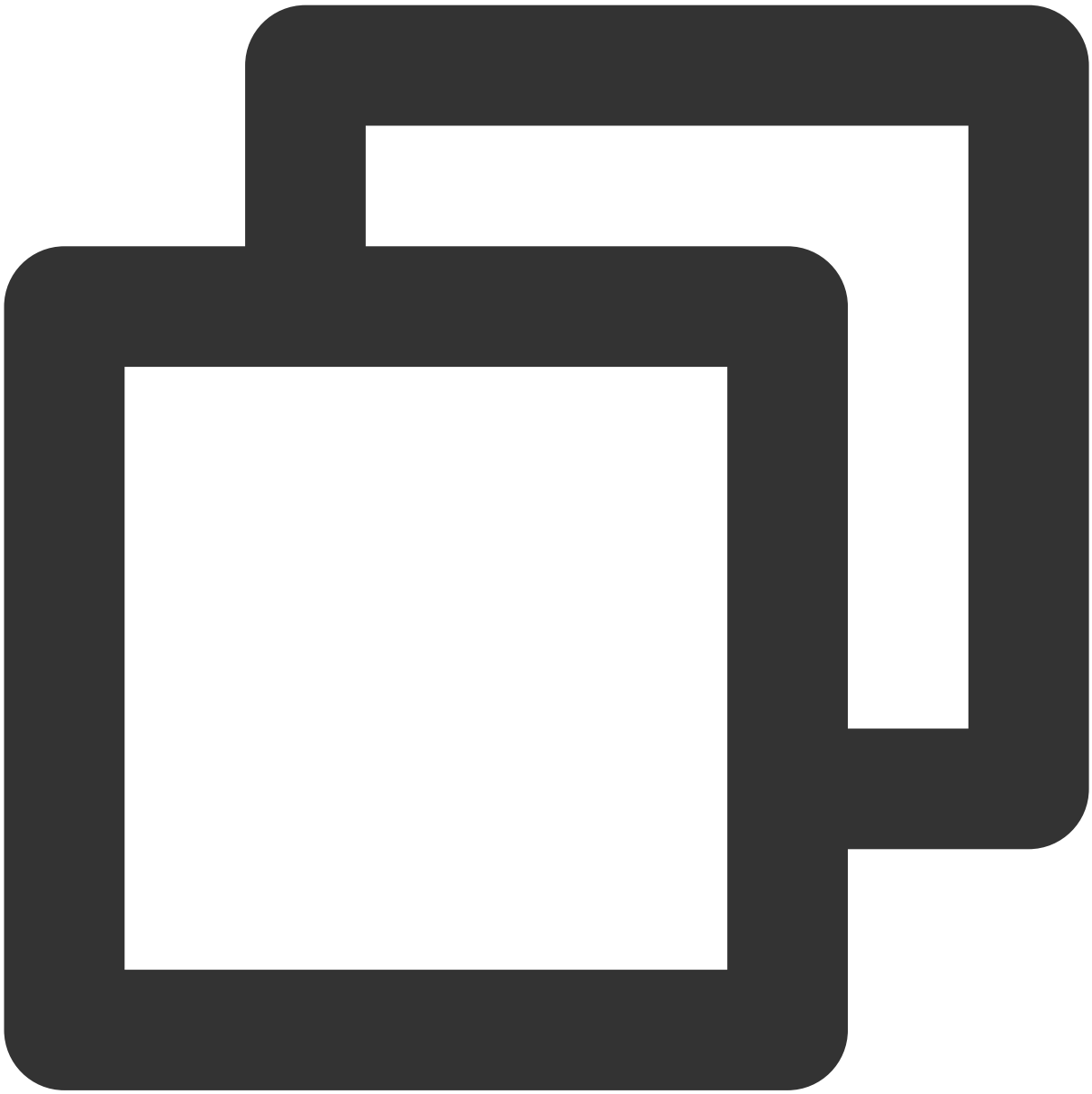
开启消息已读回执

在 `TUIChat` 组件中的 TUIChatConfig 文件里，提供了“消息已读回执”功能开关 `msgNeedReadReceipt` ，其类型为 BOOL，默认为 `NO` 。



```
- (id)init {
    self = [super init];
    if (self) {
      //...other config
      self.msgNeedReadReceipt = NO;
    }
    return self;
}
```

如果想开启消息已读回执功能，首先请开通旗舰版套餐包，然后把 `msgNeedReadReceipt` 的默认值改为 `YES` ，或者在聊天页面初始化之前调用以下方法来开启。



```
TUIChatConfig.defaultConfig.msgNeedReadReceipt = YES;
```

常见问题

Error: The usage of this API is not supported by the package. Please upgrade to the premium version.

“消息已读回执” 功能仅进阶版支持，该报错信息表示您当前的套餐包不支持此能力，请购买进阶版后使用。

Web & H5 & Uniapp（Vue）

最近更新时间：2024-07-10 16:26:41

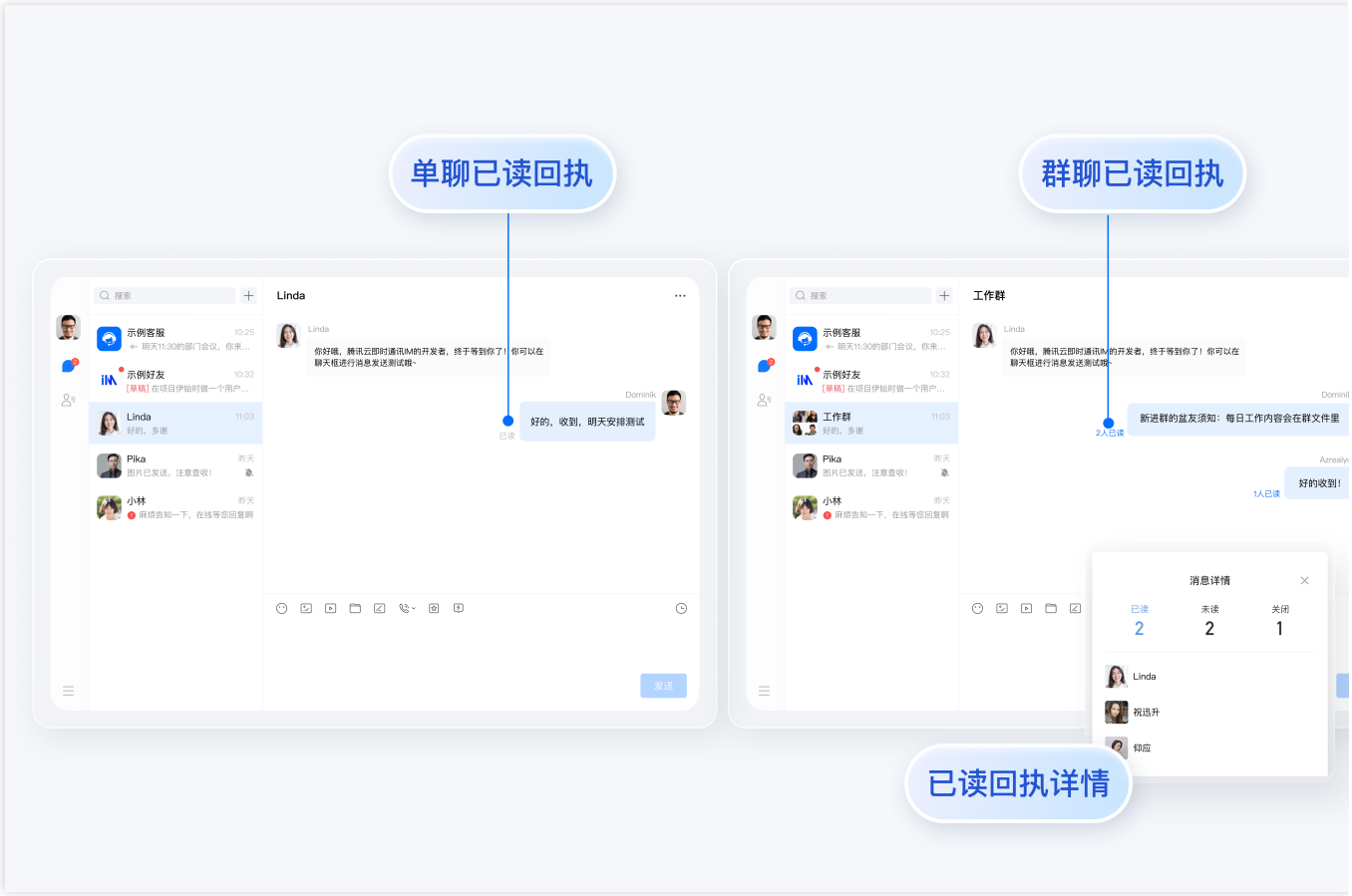
功能描述

开启“消息已读回执”功能后，TUIChat 组件会监听消息的滚动，当未读消息出现在接收方聊天窗口的可视区域内时，将自动触发向发送方发送已读回执，精确监控每一条消息的阅读状态。

注意

TUIKit 从 v2.0.0 版本开始支持群聊和单聊的消息已读回执功能，表情回应功能仅进阶版套餐支持，请购买进阶版后使用。

效果展示



如何开启消息已读回执

设置支持已读回执的群类型

如果是群消息已读回执，需要先在 Chat Console > Chat > Configuration > Group Configuration > Read receipts for group messages 中设置支持已读回执消息的群类型。

注意：

直播群（AVChatRoom）或社群（Community）**不支持**已读回执功能。

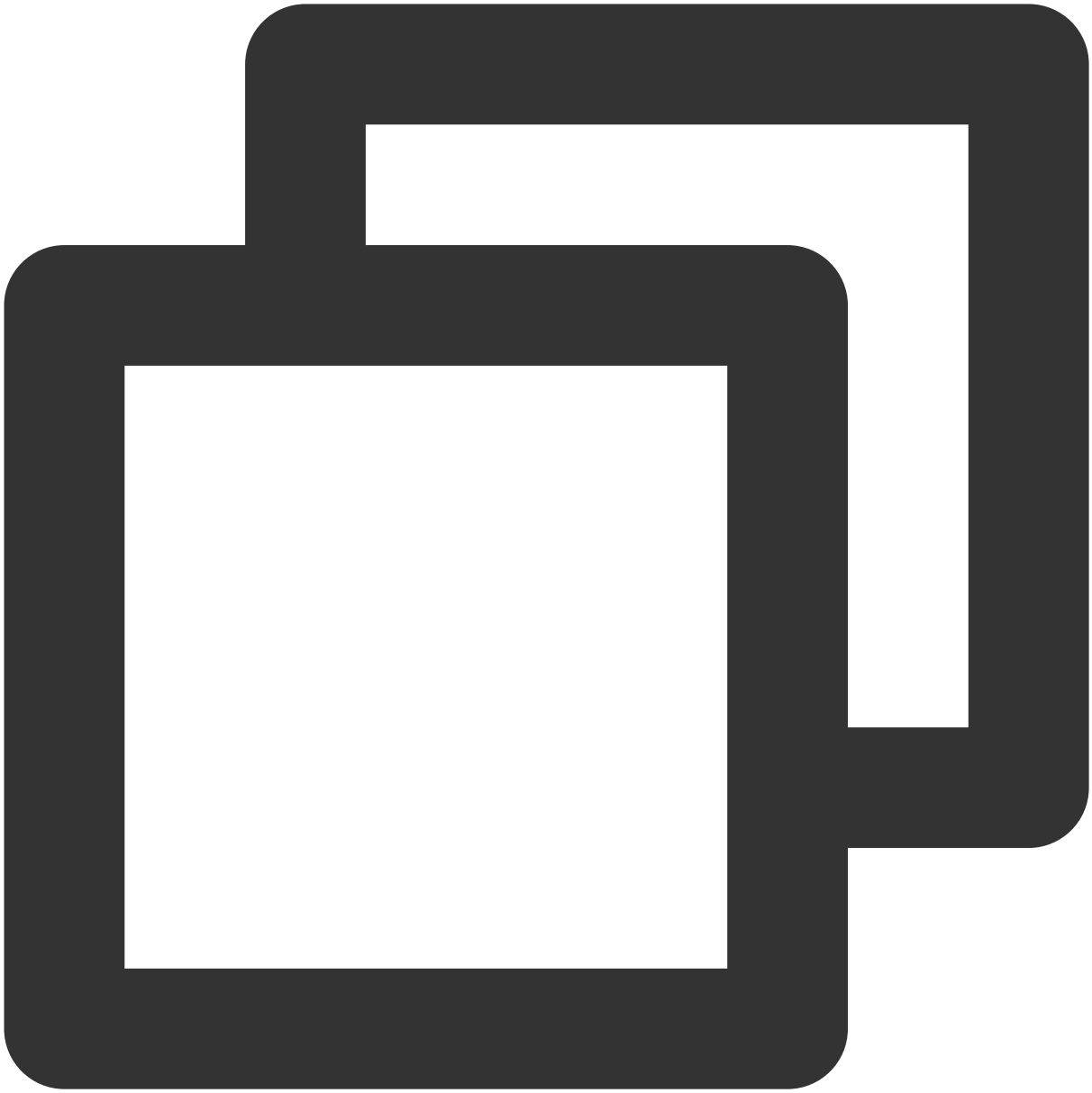
用户侧控制开启\\关闭已读回执

说明：

在控制台进行群聊的已读回执配置后，已经默认开启已读回执能力，如无特殊需求，无需在用户侧操作开关。

在用户侧也支持手动开启\\关闭已读回执能力（该项默认是开启的），但如果关闭，他人无法看到自己是否已读（即不会发送已读回执），同时自己也看不到他人是否阅读自己的消息（不展示自己发送消息的已读状态）。

在登录成功后，利用 `TUIUserService.switchMessageReadStatus(isDisplay: boolean)` Api 来控制该开关。



```
import { TUIUserService } from "@tencentcloud/chat-uikit-engine";

TUIUserService.switchMessageReadStatus(true);  // 开启
TUIUserService.switchMessageReadStatus(false); // 关闭
```

扩展资料

说明：

以下内容仅为辅助阅读资料，已读回执功能已在旗舰版 TUIKit 中包含，不需要用户手动实现。

群聊和单聊的已读回执规则

开启已读回执功能后，发送的 `Message` 的 `needReadReceipt` 字段默认为 `true` ，当消息处于对方消息列表的可视位置时，对方会发送已读回执。但要注意的是：单聊和群聊在开启已读回执功能前后的规则有所不同。

群聊的已读回执规则

1. 群聊开启已读回执之前：
不展示已读状态。
2. 群聊开启已读回执之后：
根据 `Message` 的 `readReceiptInfo` 获得已读数和未读数。
已读数为0：展示“未读”。
未读数为0：展示“全部已读”。
否则展示“x 人已读”，其中 x 为已读数。

单聊的已读回执规则

1. 单聊开启已读回执之前：
展示已读状态，但属于全量已读，当用户点击进入会话时，无论是否看到消息，所有未读消息会全部置为已读。根据 `Message` 的 `isPeerRead` 判断消息是已读或未读。
2. 单聊开启已读回执之后：
根据单条消息 `Message` 的 `readReceiptInfo.isReceiptPeerRead` 字段（布尔值）获取是否已读，从而判断处于“已读”、“未读”中的一种状态。

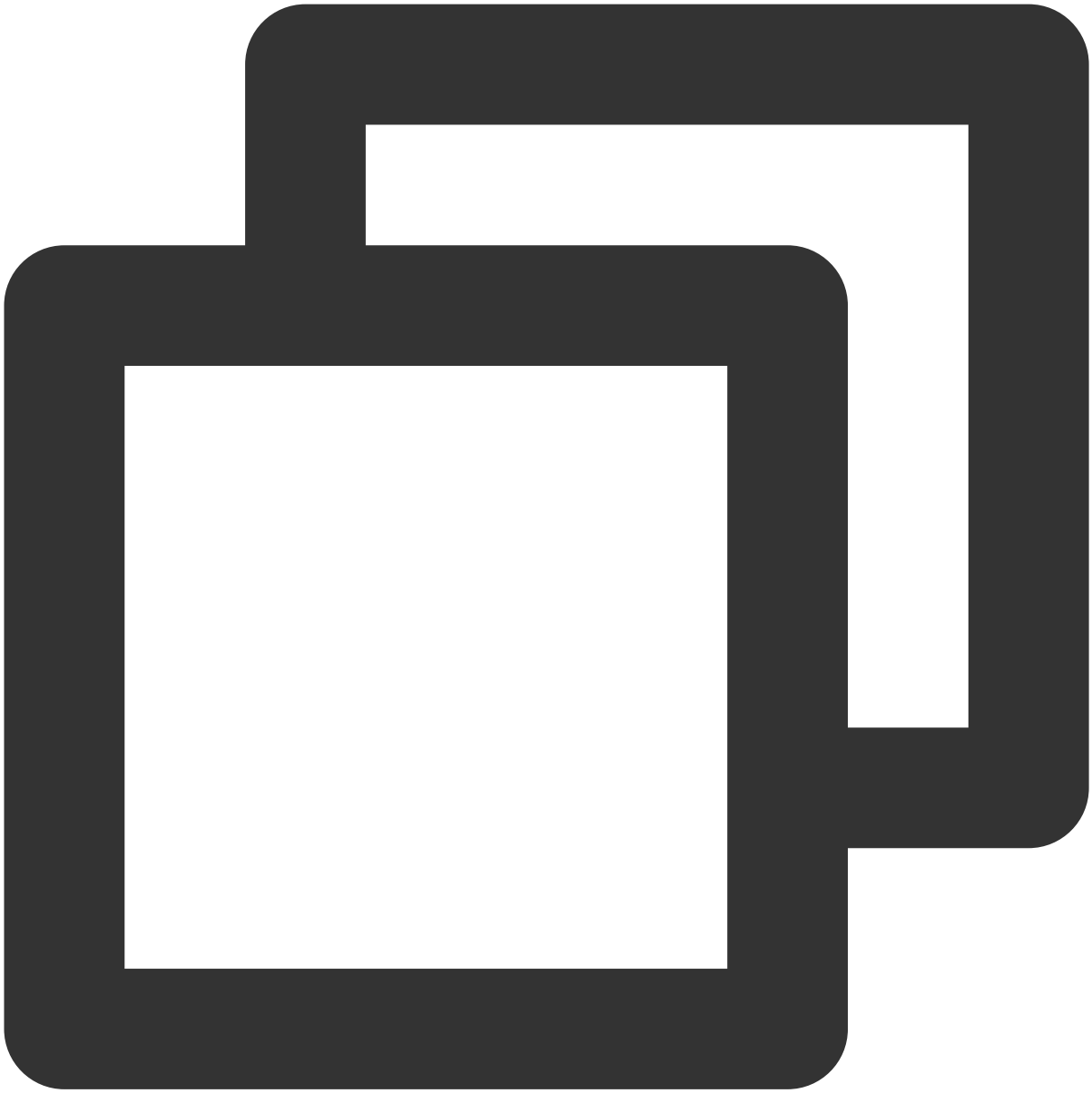
常见问题

1. Error: 套餐包不支持该接口的使用，请升级到旗舰版套餐

“群消息已读回执”功能仅旗舰版套餐支持，该报错信息表示您当前的套餐包不支持此能力，请开通旗舰版进行体验。

2. 如何关闭已读功能？

请参考本文档 2.2 节的内容，通过 `TUIUserService.switchMessageReadStatus(isDisplay: boolean)` 关闭已读功能。



```
import { TUIUserService } from "@tencentcloud/chat-uikit-engine";

TUIUserService.switchMessageReadStatus(false);
```

交流与反馈

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

Flutter

最近更新时间：2024-01-31 15:03:43

功能描述

TUIKit 从 v0.0.8 版本开始，支持“单聊消息已读回执”与“群消息已读回执”功能。

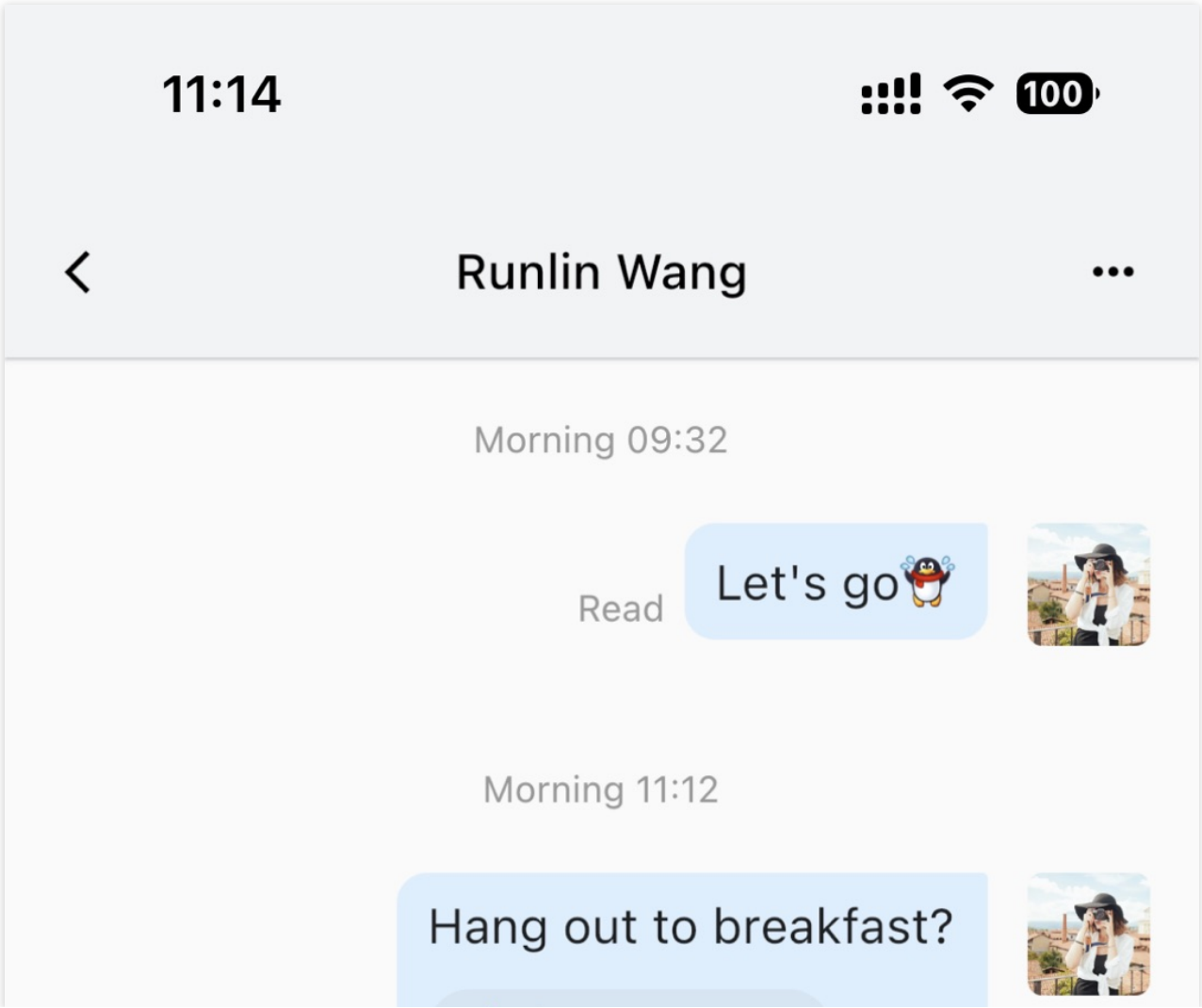
注意：

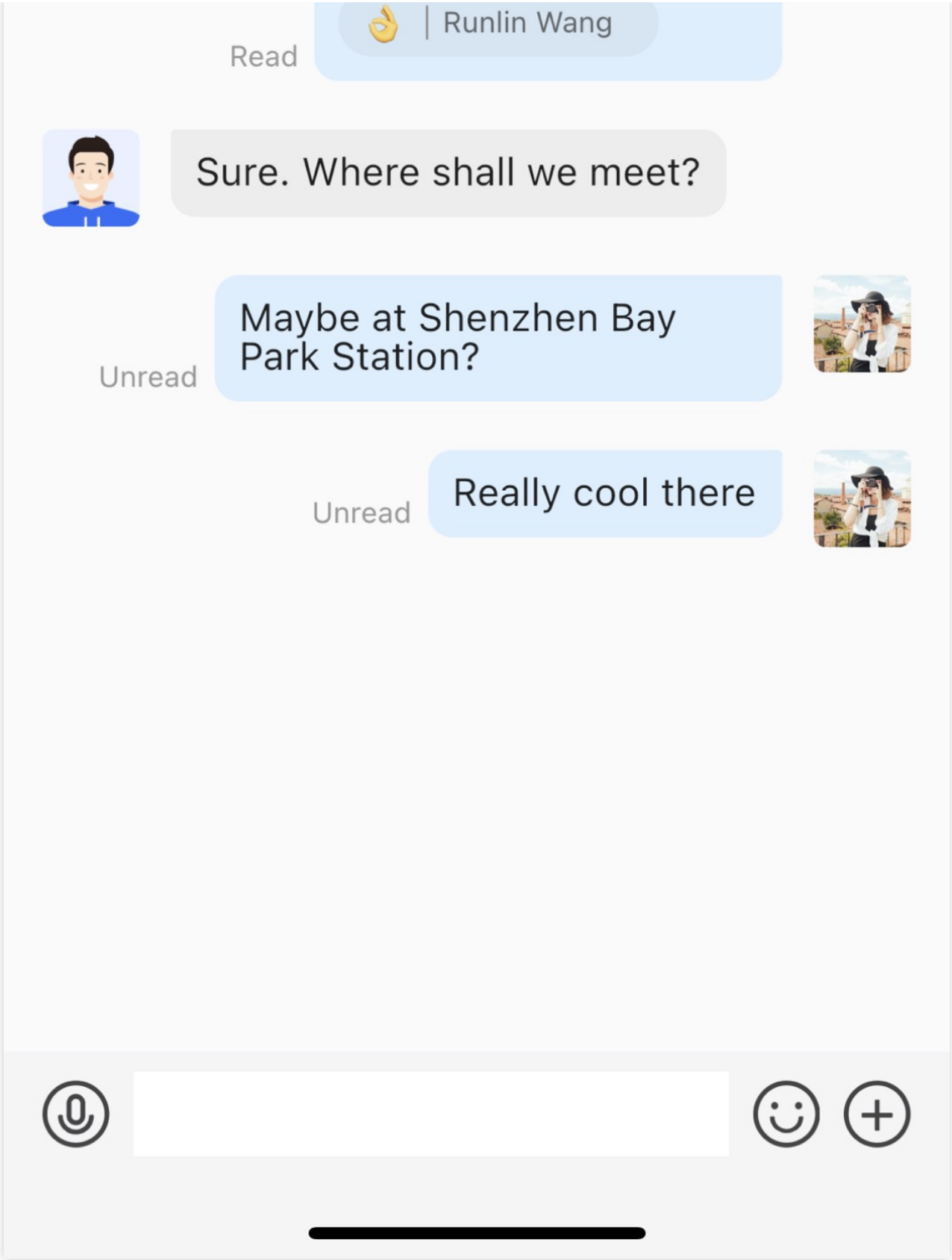
“群消息已读回执”功能仅旗舰版套餐支持，使用前请确认。

效果展示

单聊消息已读回执

通过消息左侧 “已读” / “未读” 展示。



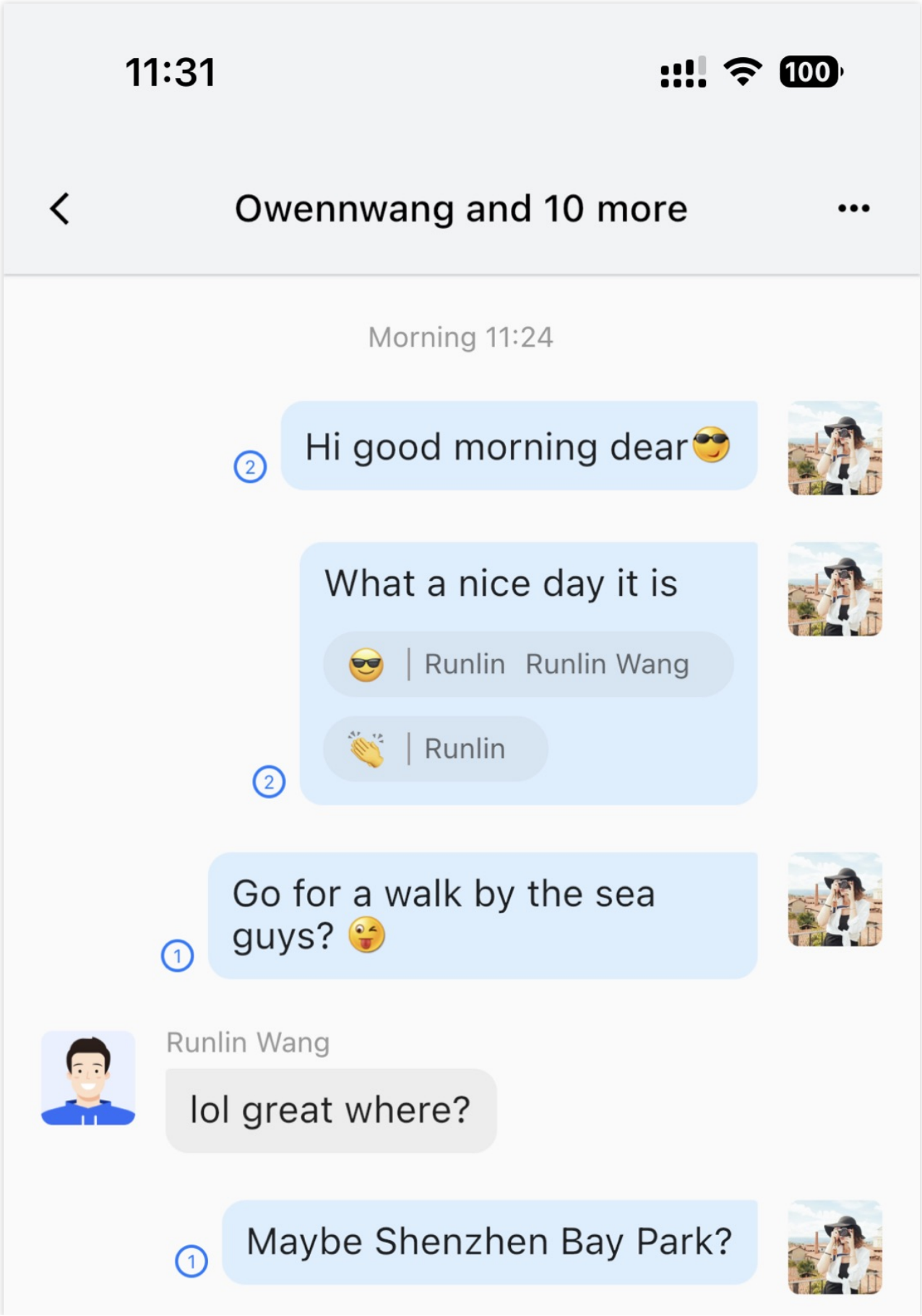


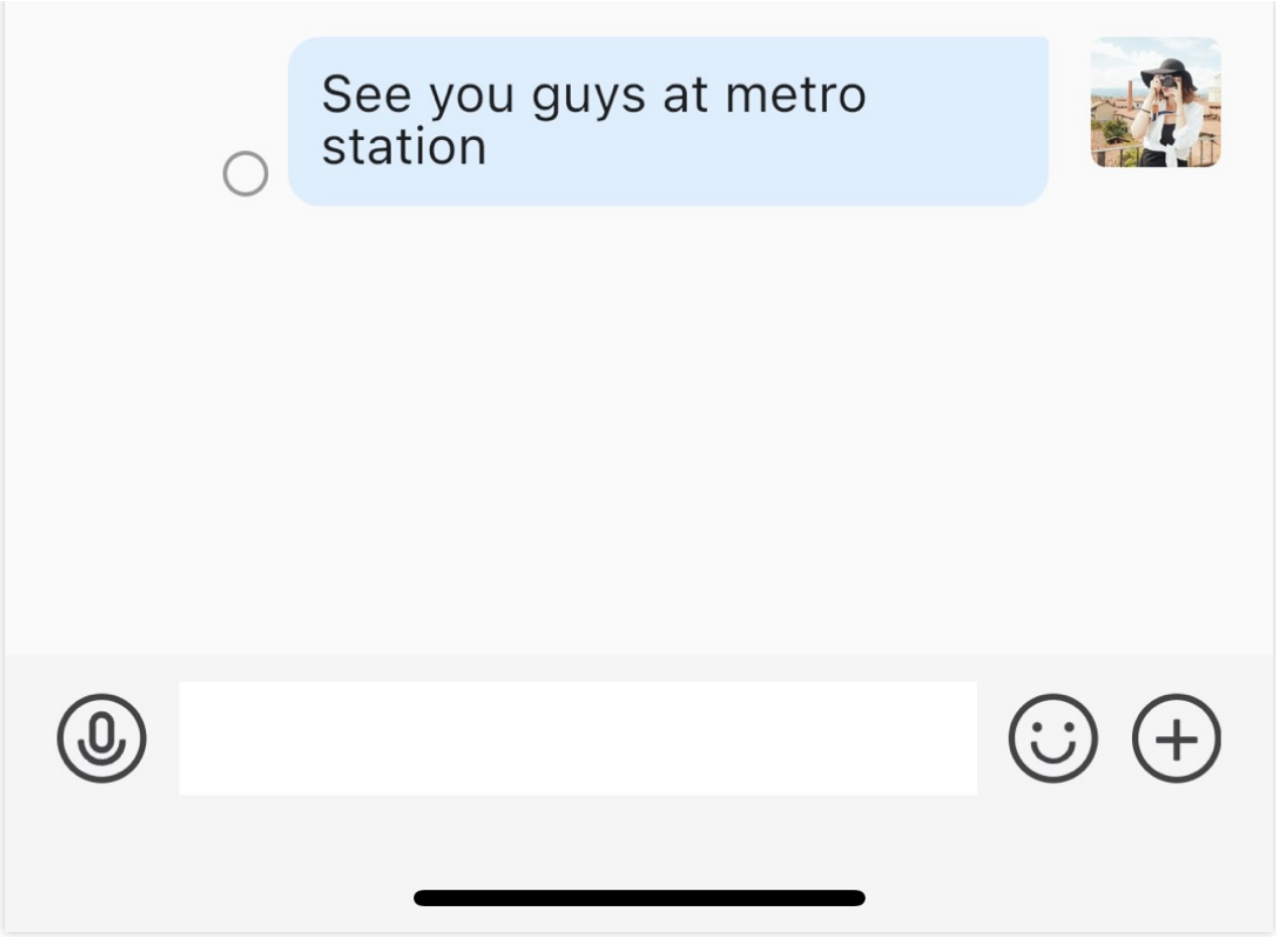
群聊消息已读回执

通过消息旁圆圈，体现群成员已读数量。点击后，进入展示详情。

版权所有：腾讯云计算（北京）有限责任公司

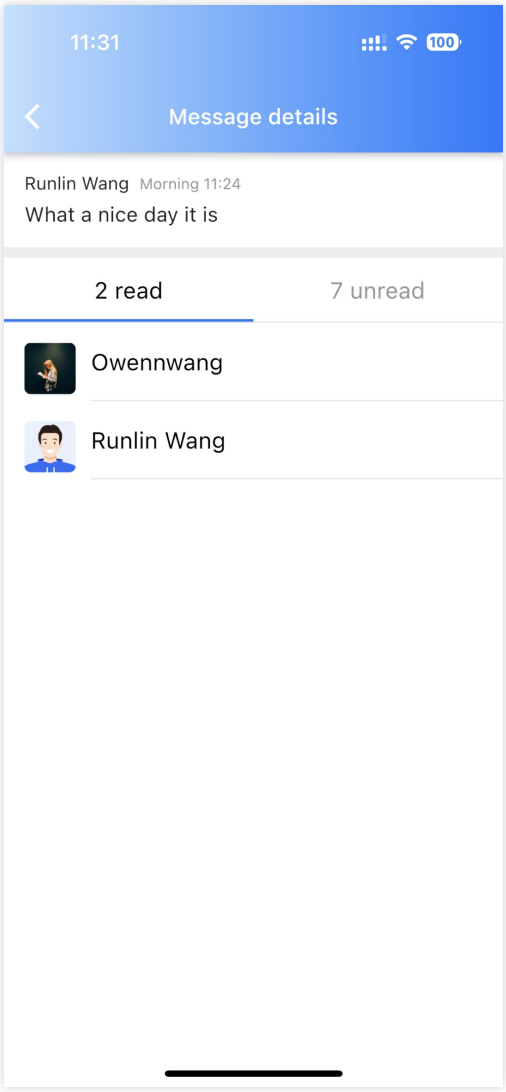
消息列表

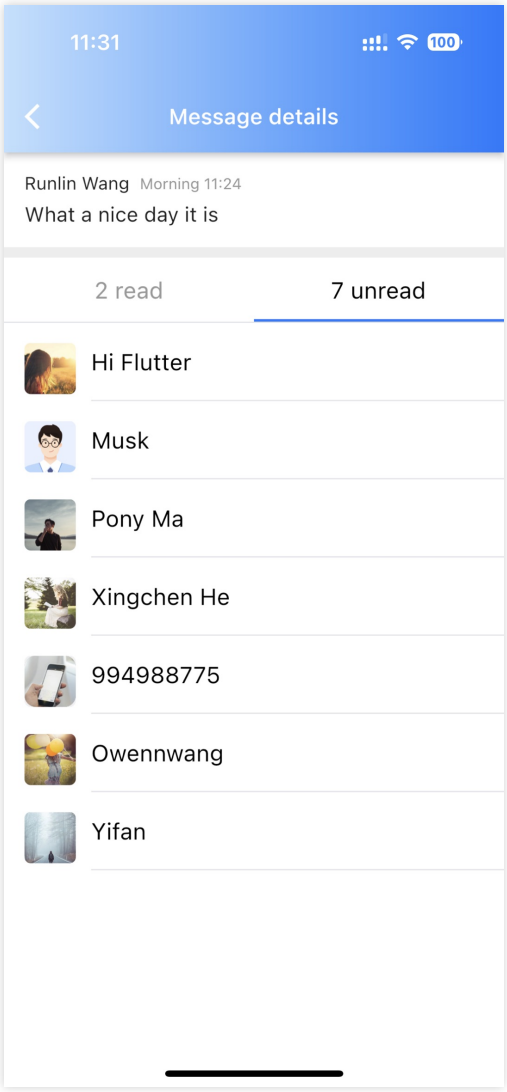

11:31
100
Owennwang and 10 more
Morning 11:24
Hi good morning dear
What a nice day it is
Runlin Runlin Wang
Runlin
Go for a walk by the sea guys?
Runlin Wang
lol great where?
Maybe Shenzhen Bay Park?



已读回执详情

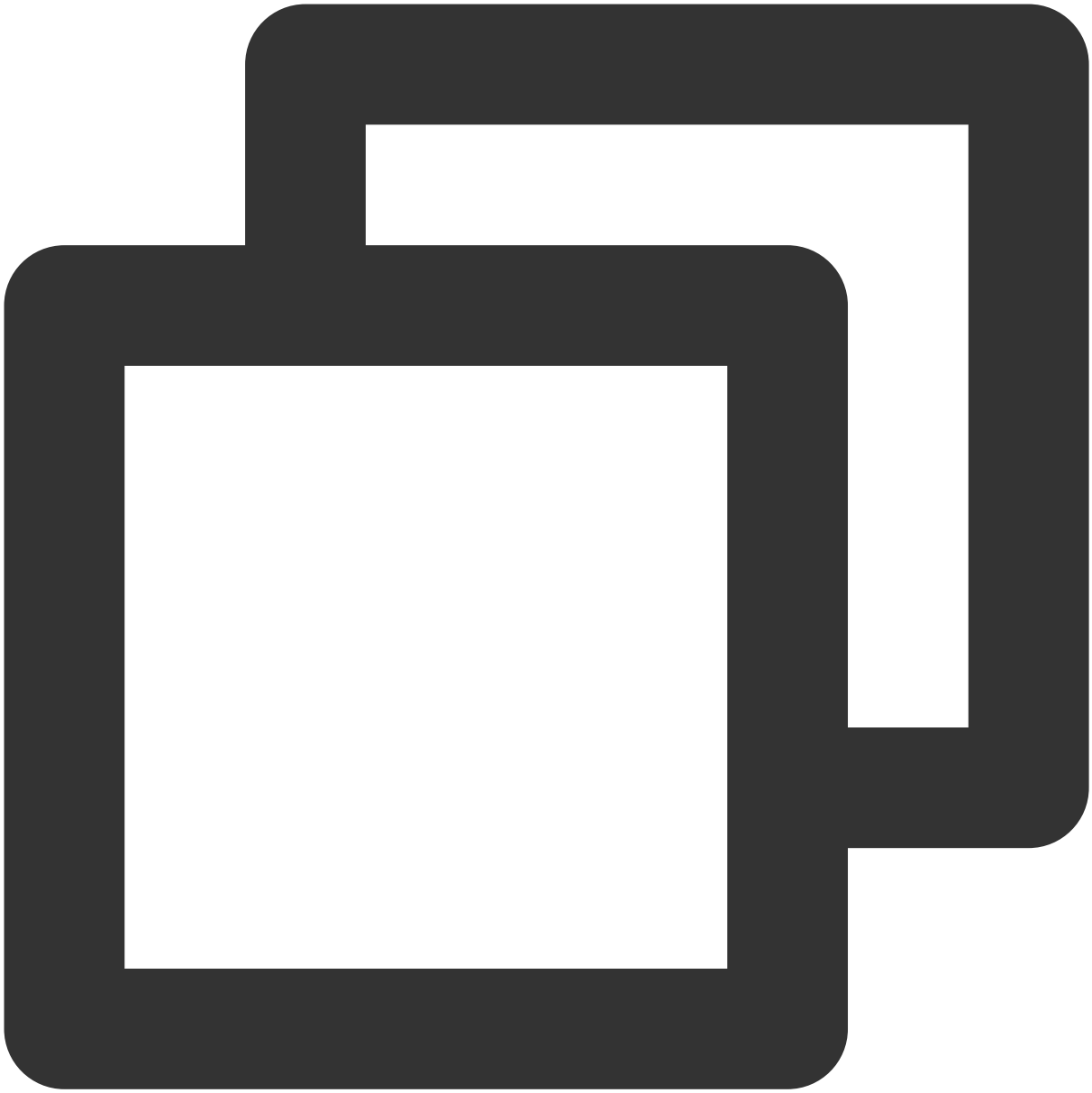
已读 群成员	未读 群成员





控制消息已读回执

在 TIMUIKitChat 的配置参数 `config` 中，提供了一系列“消息已读回执”功能开关，具体如下代码说明。对于 Boolean 类型的配置开关，默认值均为 `true` 。



```
TIMUIKitChat(
  config: TIMUIKitChatConfig(
    isShowReadingStatus: true 或 false, //  【单聊】是否展示单聊消息已读回执
    isShowGroupReadingStatus: true 或 false, //  【群聊】是否展示群聊消息已读回执
    isReportGroupReadingStatus: true 或 false, //  【群聊】是否上报群聊消息已读回执
    groupReadReceiptPermissionList: [
      GroupReceiptAllowType.work,
      GroupReceiptAllowType.meeting,
      GroupReceiptAllowType.public
    ], //  【群聊】哪些类型的群，需要展示群聊消息已读回执
    // ... 其他 config 配置
```

```
    ),
    // ... 其他 TIMUIKitChat 参数
)
```

联系我们

如果您在接入使用过程中有任何疑问，请通过如下方式联系我们。

Telegram Group

WhatsApp Group

版权所有：腾讯云计算（北京）有限责任公司

消息表情回应

Android

最近更新时间：2024-07-05 15:24:08

功能描述

从 7.8 版本开始，表情回应功能由 `TUIEmojiPlugin` 插件提供。
如果您不需要表情回应功能，不集成该插件即可。长按文本消息时，不会显示表情回应模块。
如果您需要该功能，需集成 `TUIChat` 和 `TUIEmojiPlugin` 。集成方法请参见《集成基础功能》。集成后不需要进行任何设置，长按文本消息时，自动显示表情回应按钮。

注意：

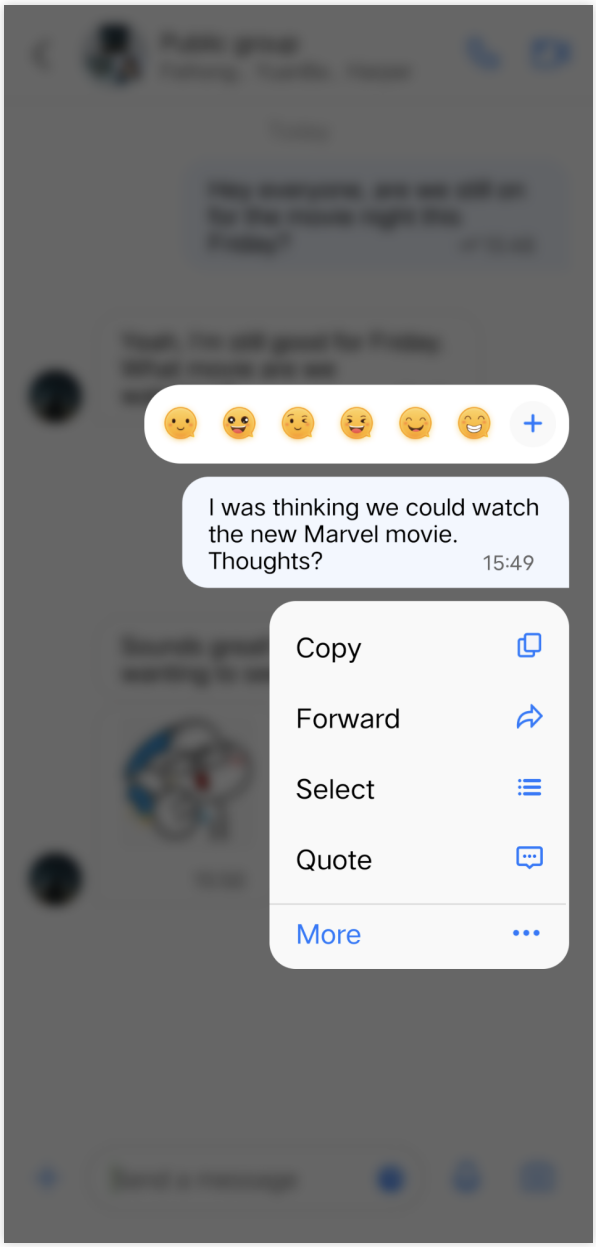
表情回应功能仅进阶版套餐支持，请购买进阶版后使用。

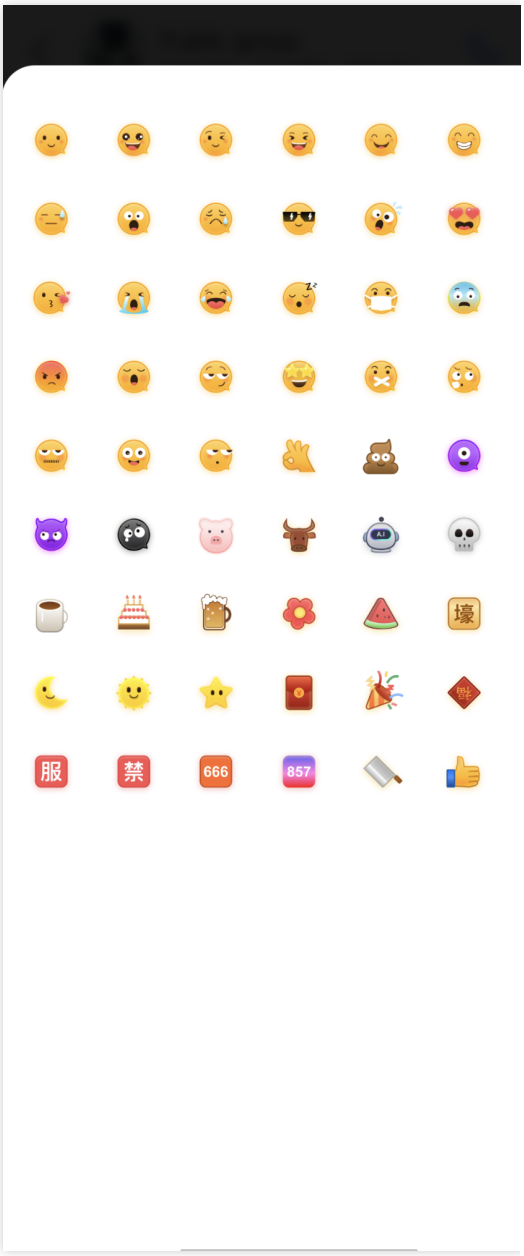
效果展示

发送表情回应

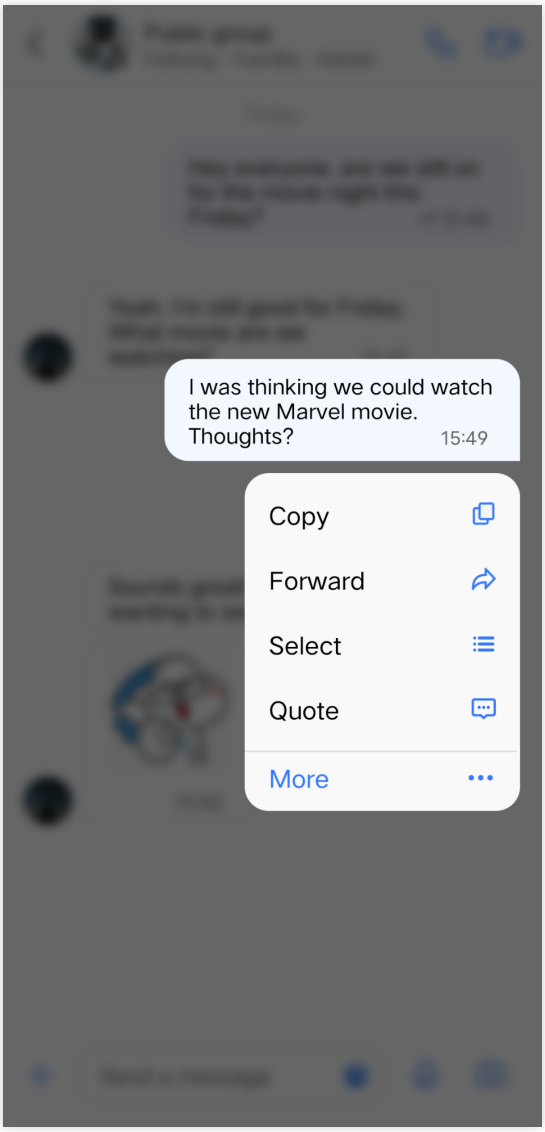
集成表情回应能力后，长按消息菜单上方会多一条表情选择区。该区域支持单击右侧按钮扩大，展示更多表情。单击表情即可对该消息进行表情回应。如果已经使用该表情对该消息进行回应过，单击表情后会取消回应。

长按消息菜单	更多表情





如果不集成表情回应，消息长按弹窗上方不会展示表情回应入口，如图所示：



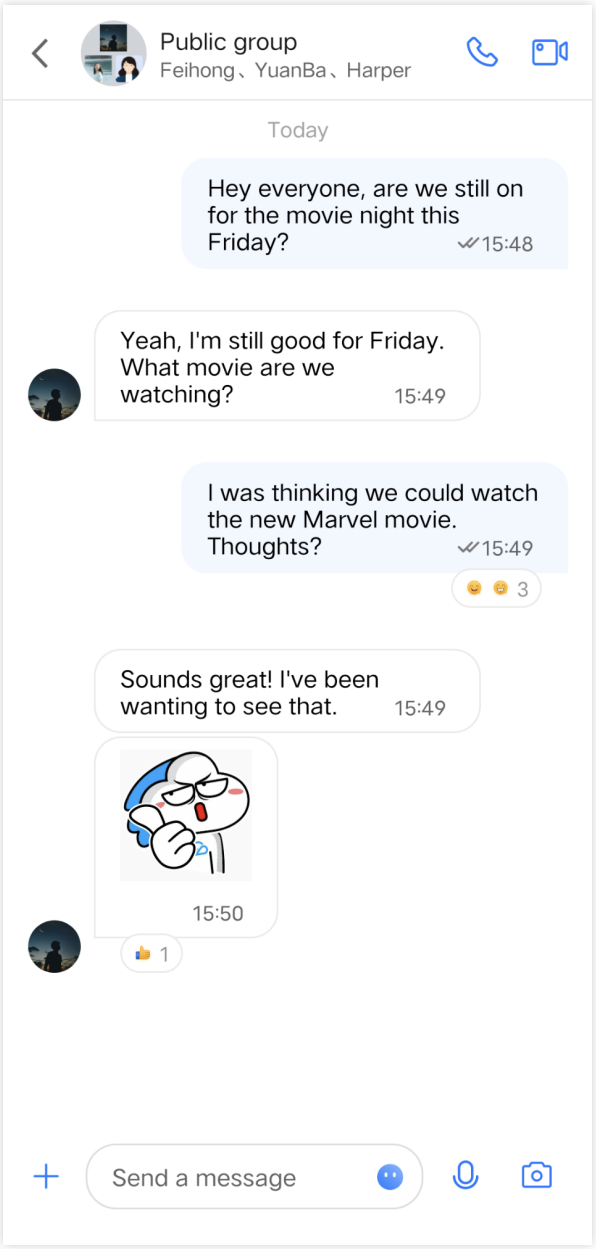
展示表情回应

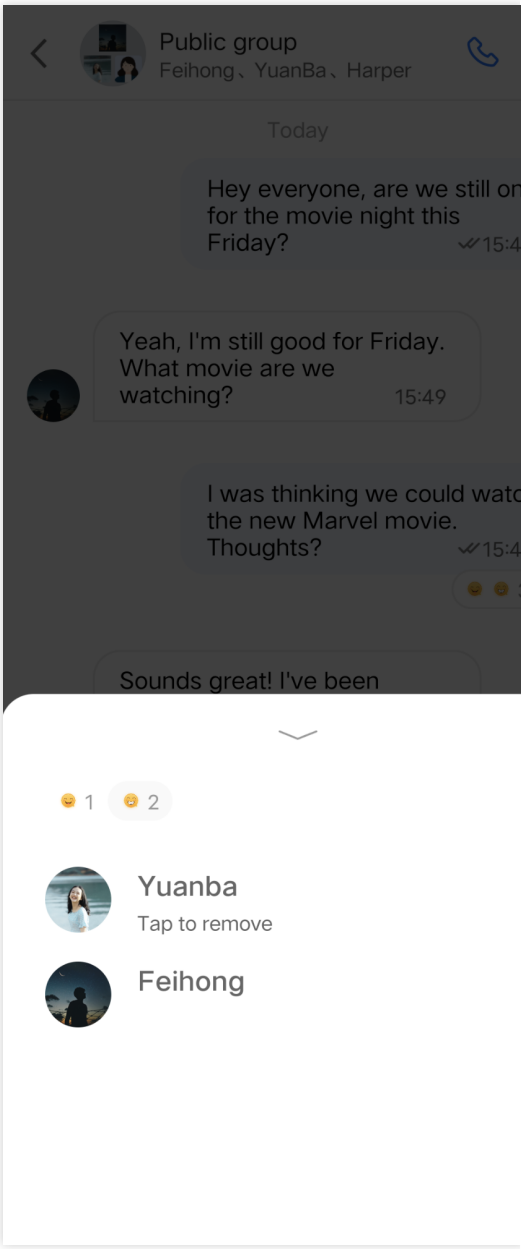
一条消息收到的回应表情，都会展示在这条消息的下方，会话中所有成员均可看到。

在消息下方，会显示回应的表情和回应了该表情的聊天成员昵称，单击表情或昵称可以查看该消息的表情回应详情。

在表情回应详情界面，单击自己发送的表情回应，可快速撤回该表情回应。

表情回应预览	表情回应详情


Public group
Feihong、YuanBa、Harper
Today
Hey everyone, are we still on for the movie night this Friday?
15:48
Yeah, I'm still good for Friday. What movie are we watching?
15:49
I was thinking we could watch the new Marvel movie. Thoughts?
15:49
3
Sounds great! I've been wanting to see that.
15:49
15:50
1
Send a message


Public group
Feihong、YuanBa、Harper
Today
Hey everyone, are we still on for the movie night this Friday?
Yeah, I'm still good for Friday. What movie are we watching?
15:49
I was thinking we could watch the new Marvel movie. Thoughts?
Sounds great! I've been
1
2
Yuanba
Tap to remove
Feihong

iOS

最近更新时间：2024-07-05 15:24:08

功能描述

从 7.8 版本开始，表情回应功能由 `TUIEmojiPlugin` 插件提供。
如果您不需要表情回应功能，不集成该插件即可。长按文本消息时，不会显示表情回应模块。
如果您需要该功能，需集成 `TUIChat` 和 `TUIEmojiPlugin` 。集成方法请参见《集成基础功能》。集成后不需要进行任何设置，长按文本消息时，自动显示表情回应按钮。

注意：

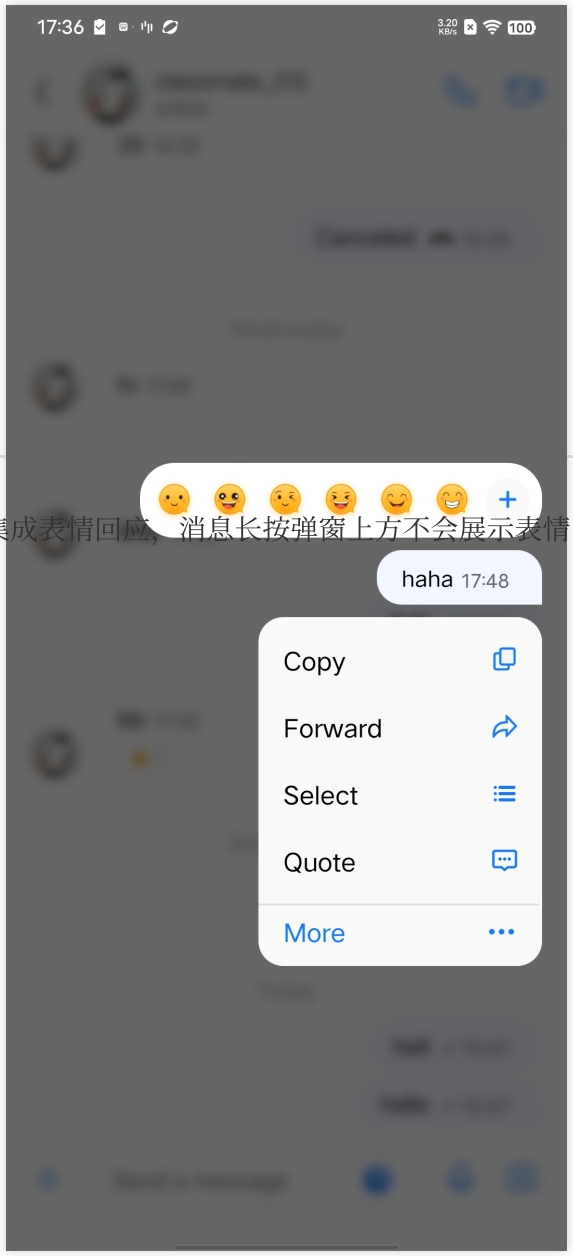
表情回应功能仅进阶版套餐支持，请购买进阶版后使用。

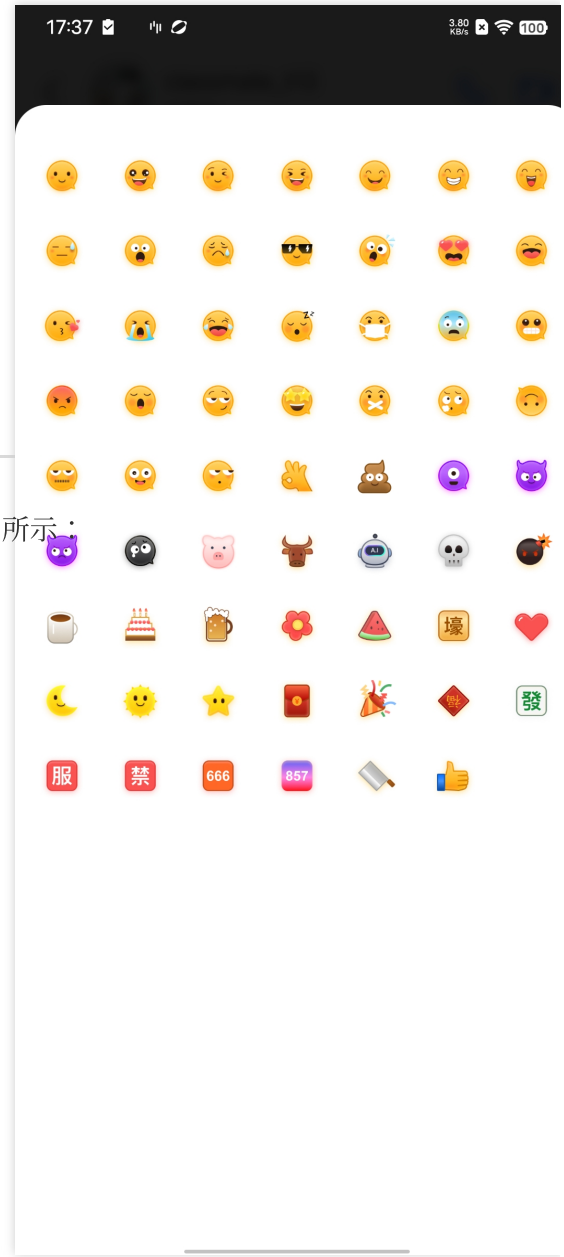
效果展示

发送表情回应

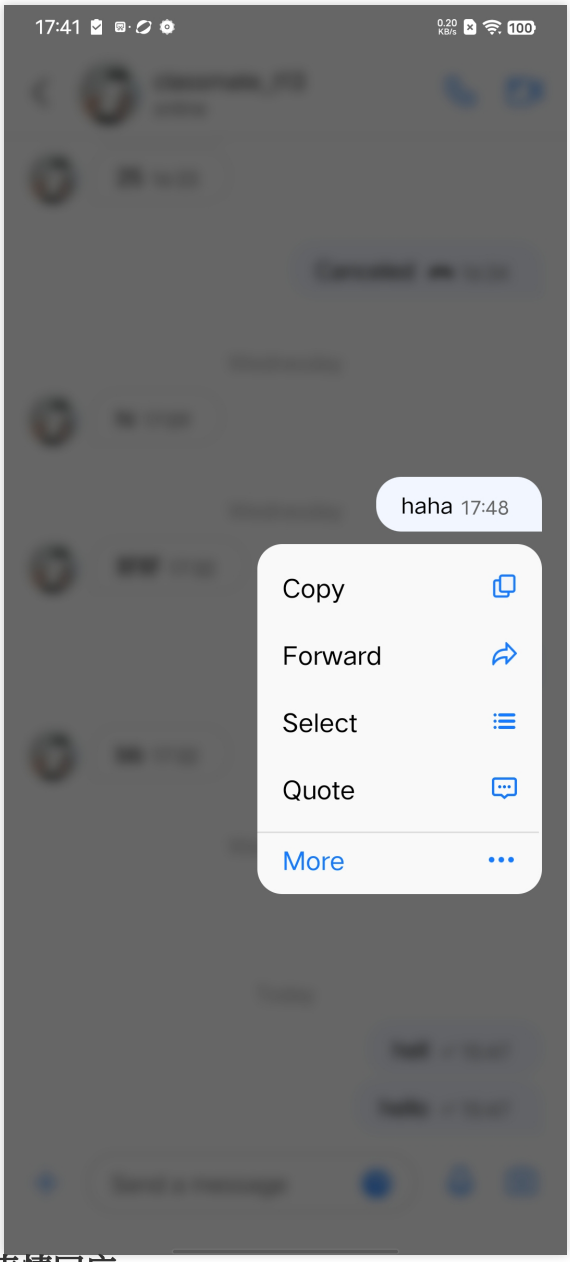
集成表情回应能力后，长按消息菜单上方会多一条表情选择区。该区域支持单击右侧按钮扩大，展示更多表情。单击表情即可对该消息进行表情回应。如果已经使用该表情对该消息进行回应过，单击表情后会取消回应。

长按消息菜单	更多表情





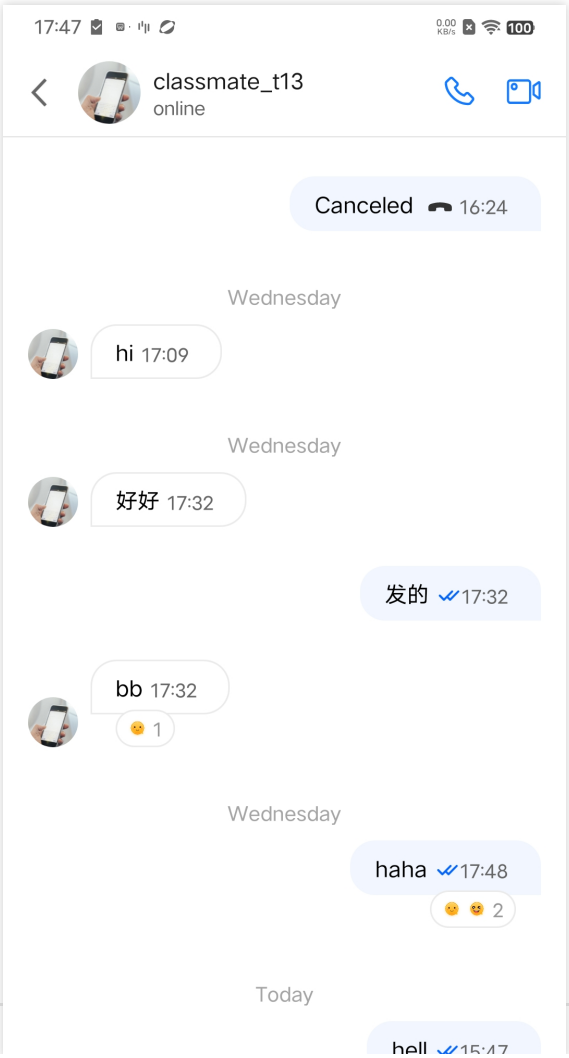
如果不集成表情回应，消息长按弹窗上方不会展示表情回应入口，如图所示：



展示表情回应

一条消息收到的回应表情，都会展示在这条消息的下方，会话中所有成员均可看到。
在消息下方，会显示回应的表情和回应了该表情的聊天成员昵称，单击表情或昵称可以查看该消息的表情回应详情。
在表情回应详情界面，单击自己发送的表情回应，可快速撤回该表情回应。

表情回应预览	表情回应详情


17:47
classmate_t13
online
Canceled 16:24
Wednesday
hi 17:09
Wednesday
好好 17:32
发的 17:32
bb 17:32
1
Wednesday
haha 17:48
2
Today

17:47
classmate_t13
online
Canceled 16:24
Wednesday
hi 17:09
Wednesday
好好 17:32
发的 17:32
bb 17:32
1
1
teacher10
Tap to remove

Web & H5 & Uniapp（Vue）

最近更新时间：2024-07-10 16:26:41

功能描述

表情回应功能**默认集成**。集成方法请参见集成 TUIKit。集成后不需要进行任何设置，长按文本消息时，自动显示**表情回应按钮**。

集成表情回应能力后，长按消息菜单上方会多一条表情选择区。该区域支持单击右侧按钮扩大，展示更多表情。单击表情即可对该消息进行表情回应。如果已经使用该表情对该消息进行回应过，单击表情后会取消回应。

注意：

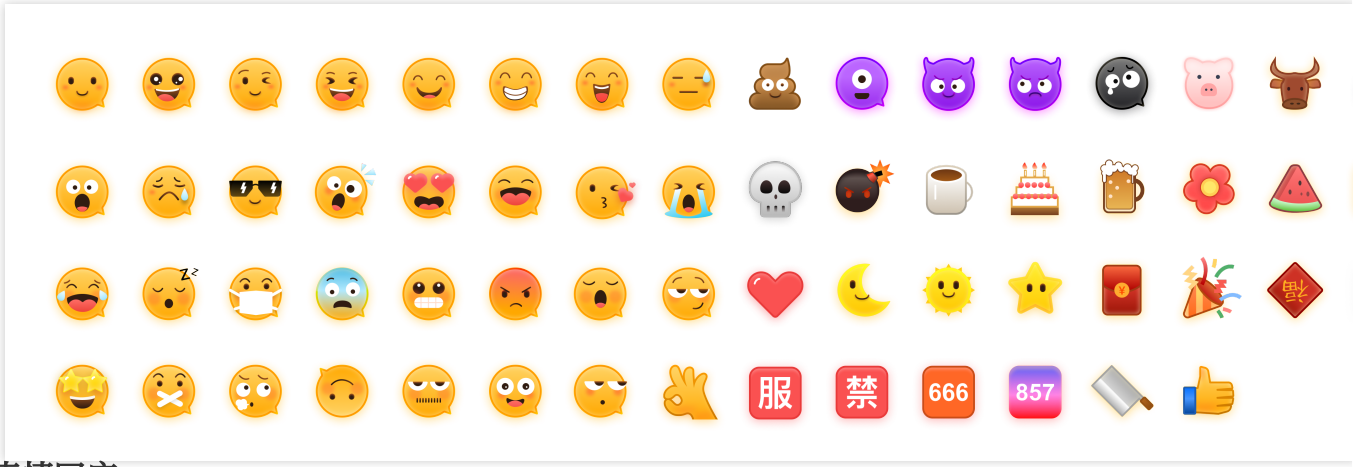
TUIKit 从 v2.0.5 版本开始支持消息表情回应功能，表情回应功能仅进阶版套餐支持，请购买进阶版后使用。

效果展示

发送表情回应

集成表情回应能力后，右键单击消息气泡，靠近消息本身的方向，会多一条表情选择区。该区域支持单击“更多”扩大，展示更多表情。

可选表情回应列表



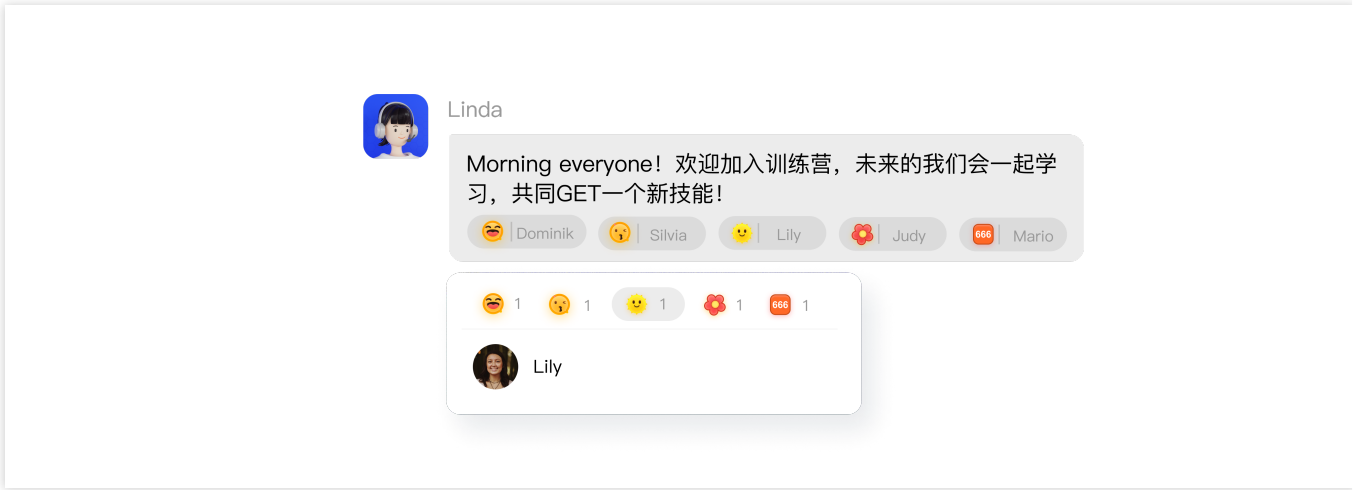
展示表情回应

一条消息收到的所有回应表情，都会展示在这条消息的下方，会话中所有成员均可看到。

在消息下方，回应表情后面会显示该表情的发送人姓名。

单击展示中表情，可以方便快捷回应同样的表情，或取消该表情。

当发送同一个回应表情人数过多被省略时，单击最后的“...共xx人”，可查看完整的回应成员名单。



交流与反馈

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

Flutter

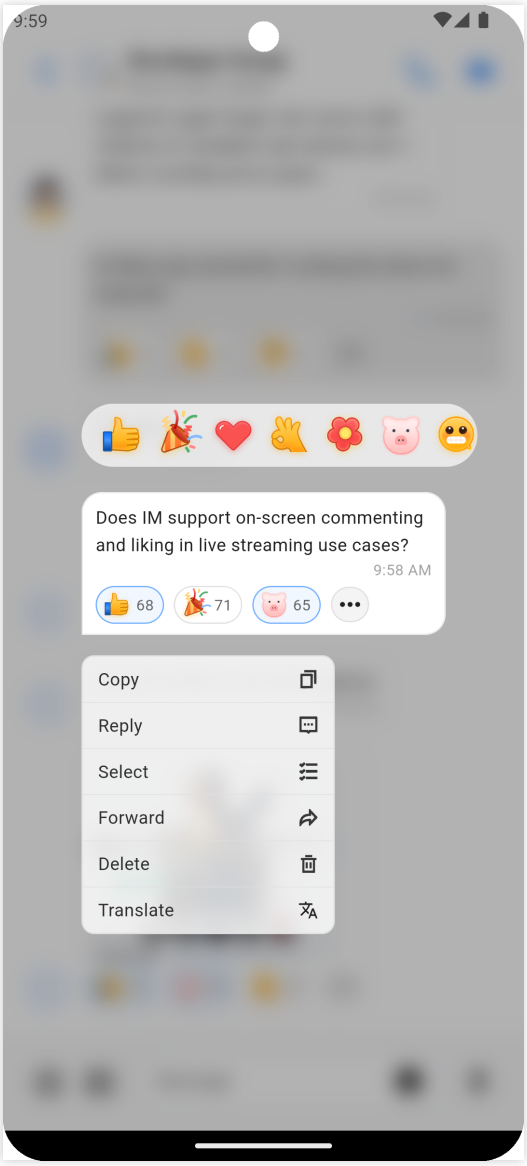
最近更新时间：2024-07-08 16:10:45

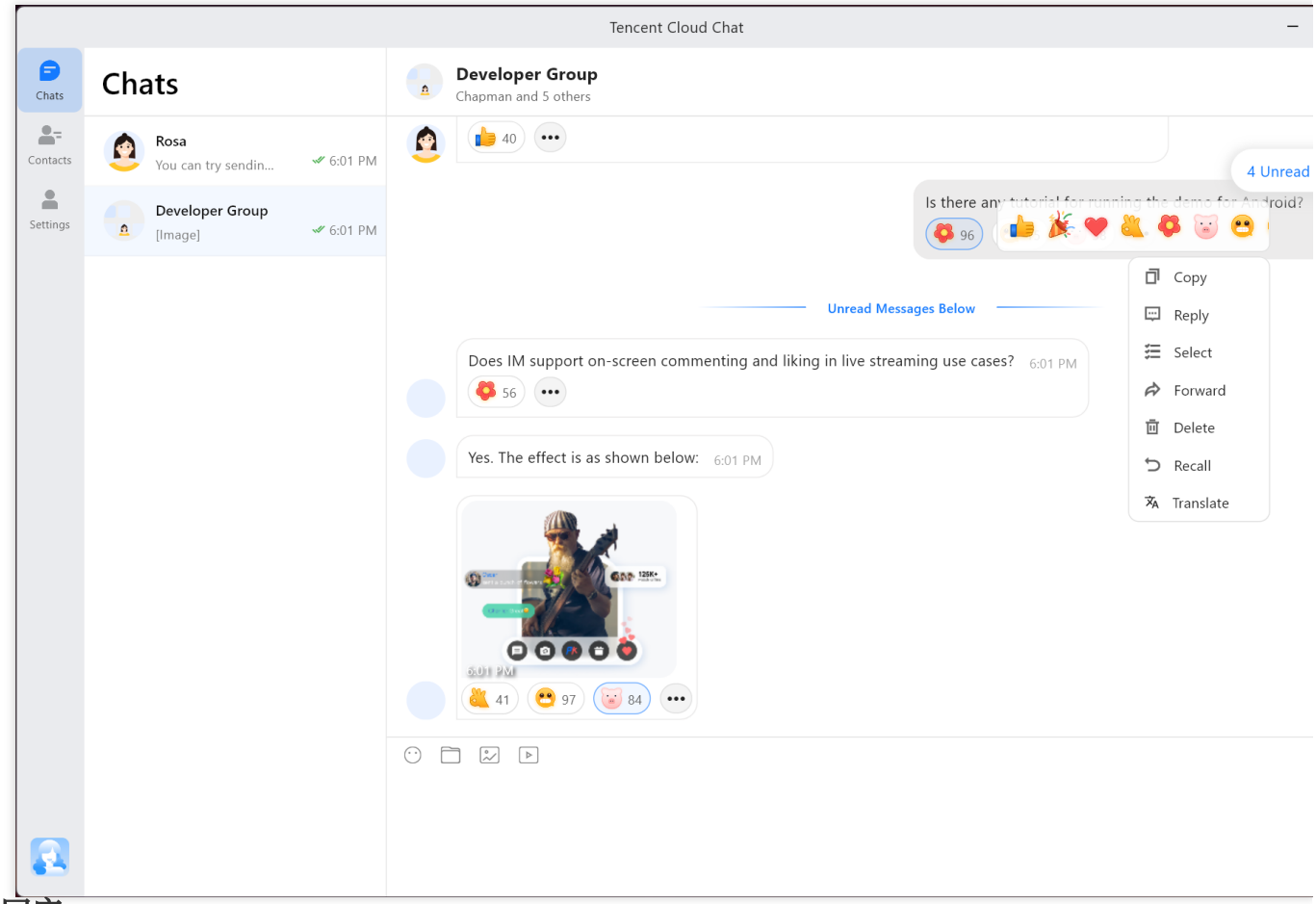
描述

消息回应功能可通过 tencent_cloud_chat_message_reaction 插件使用。它包括两个主要部分：
回应选择器：允许用户选择和发送回应。
回应列表：显示对消息的回应。

发送回应

手机：按住您想要做出反应的消息。
桌面：右键单击您想要做出反应的消息。
这将打开消息上下文菜单。
启用模块后，菜单将包括一个额外的回应选择器区域。
移动
桌面


9:59
Does IM support on-screen commenting and liking in live streaming use cases?
9:58 AM
68
71
65
Copy
Reply
Select
Forward
Delete
Translate



查看回应

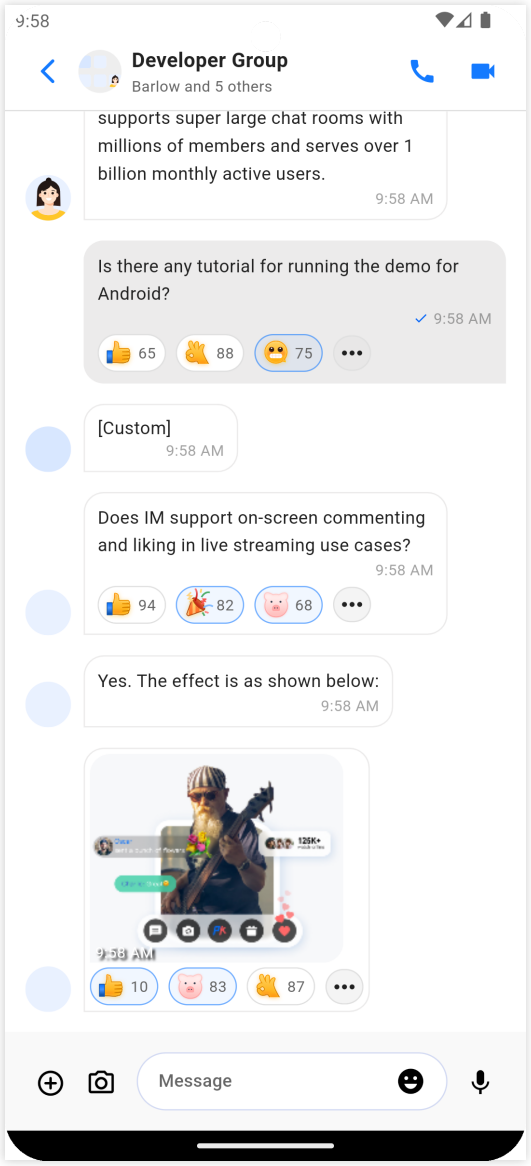
所有消息回应都显示在消息下方，并且聊天中的所有用户都可以看到。

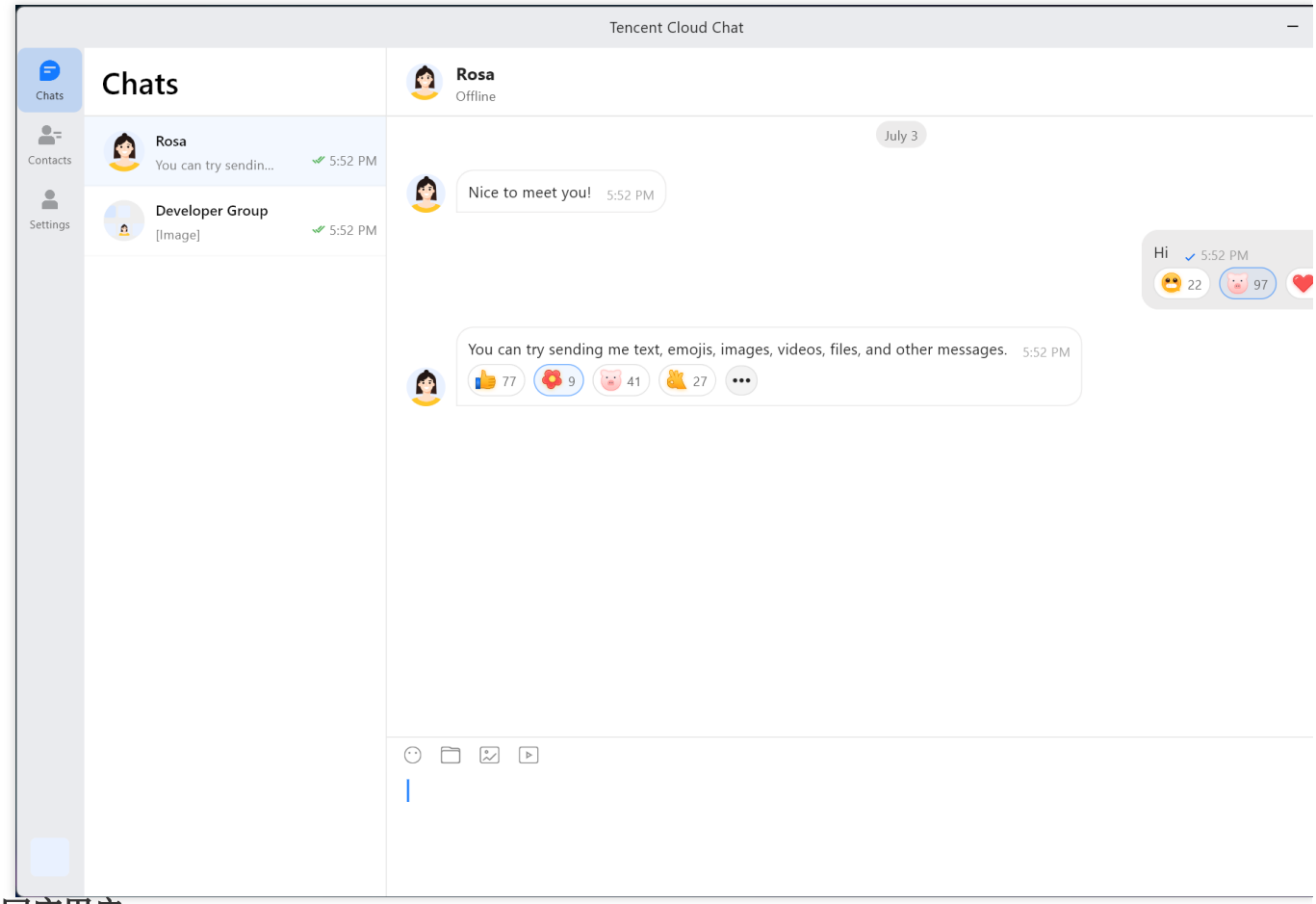
对特定表情符号做出反应的用户总数显示在贴纸旁边。

通过点击任何反应，用户可以快速发送相同的反应或删除他们的回应。

移动

桌面


9:58
Developer Group
Barlow and 5 others
supports super large chat rooms with millions of members and serves over 1 billion monthly active users.
9:58 AM
Is there any tutorial for running the demo for Android?
9:58 AM
65
88
75
[Custom]
9:58 AM
Does IM support on-screen commenting and liking in live streaming use cases?
9:58 AM
94
82
68
Yes. The effect is as shown below:
9:58 AM
9:58 AM
10
83
87
Message



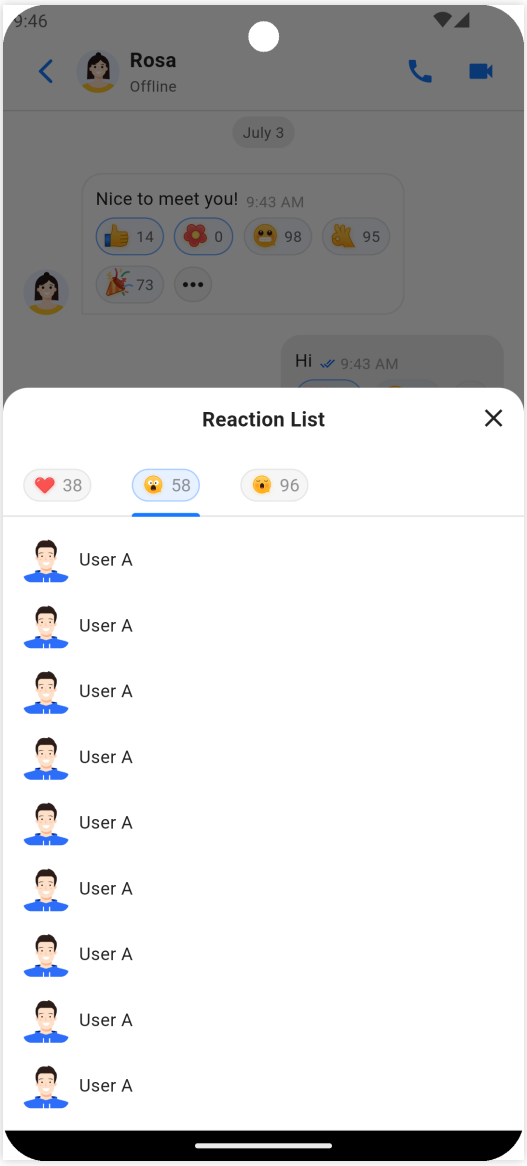
查看回应用户

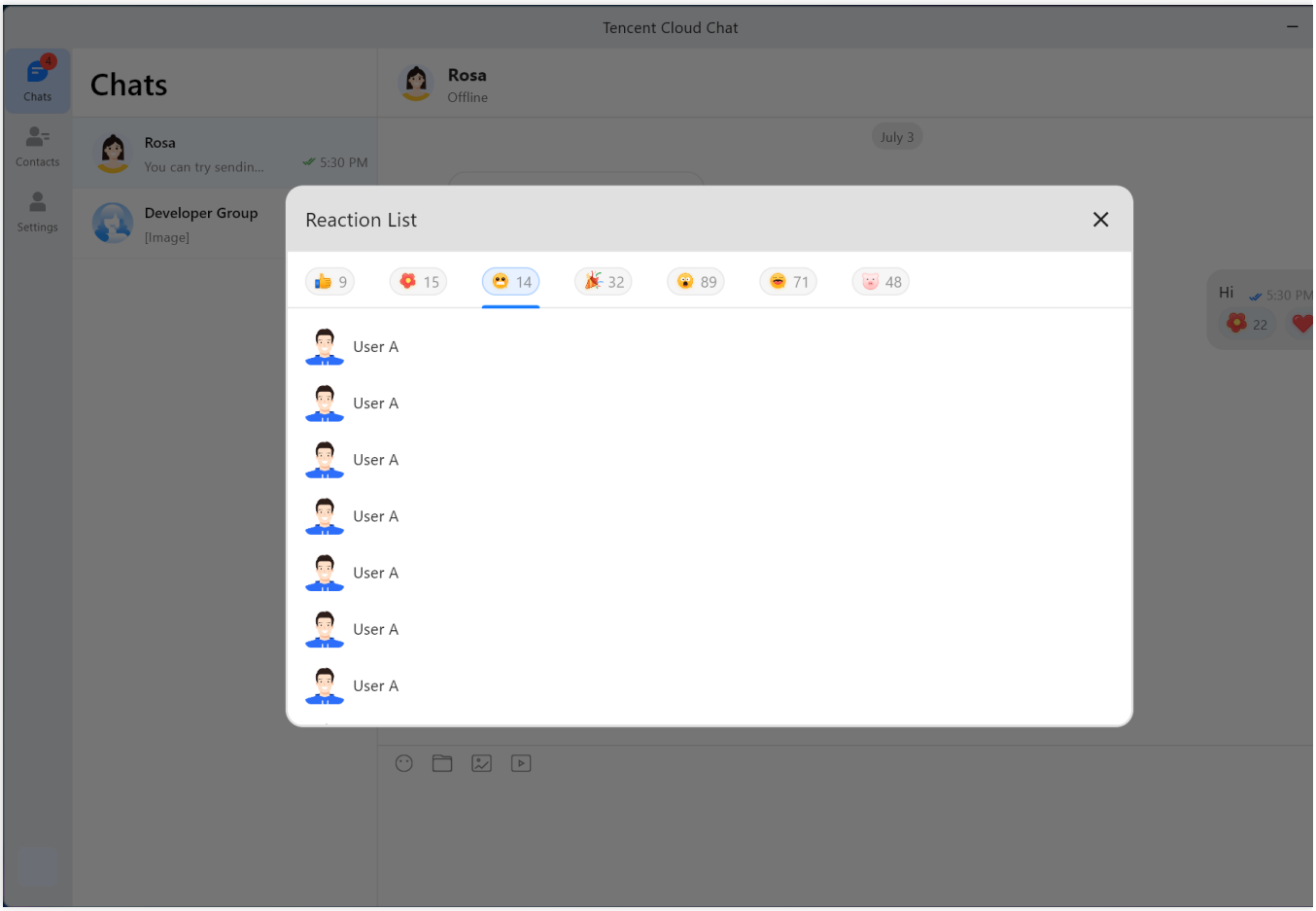
要查看对消息做出回应的用户列表，请单击 “...“ 最后一个回应旁边的按钮。

这将显示对每个特定贴纸做出回应的所有用户的列表。

移动

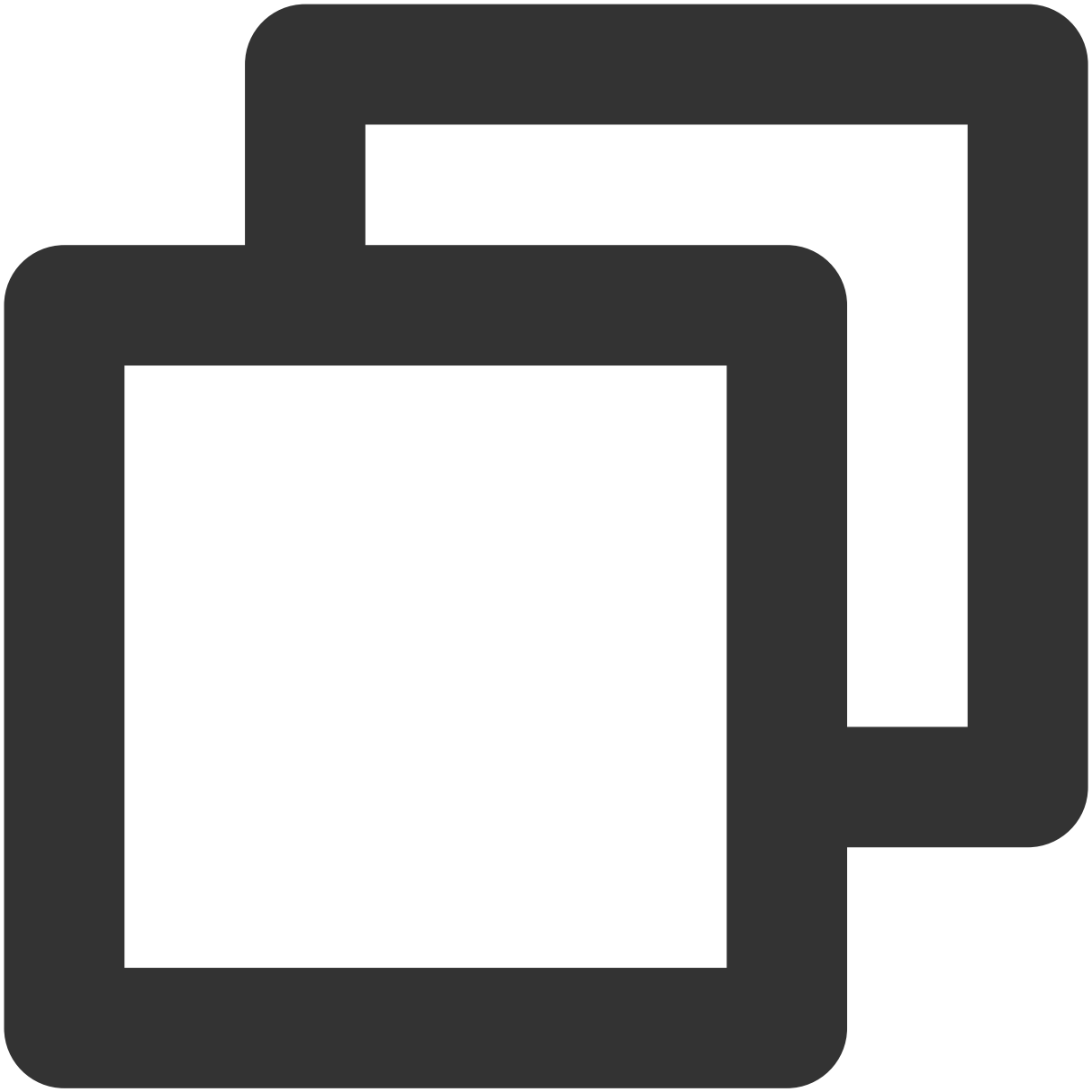
桌面


9:46
Rosa
Offline
July 3
Nice to meet you! 9:43 AM
14
0
98
95
73
Hi 9:43 AM
Reaction List
38
58
96
User A
User A
User A
User A
User A
User A
User A
User A
User A



使用

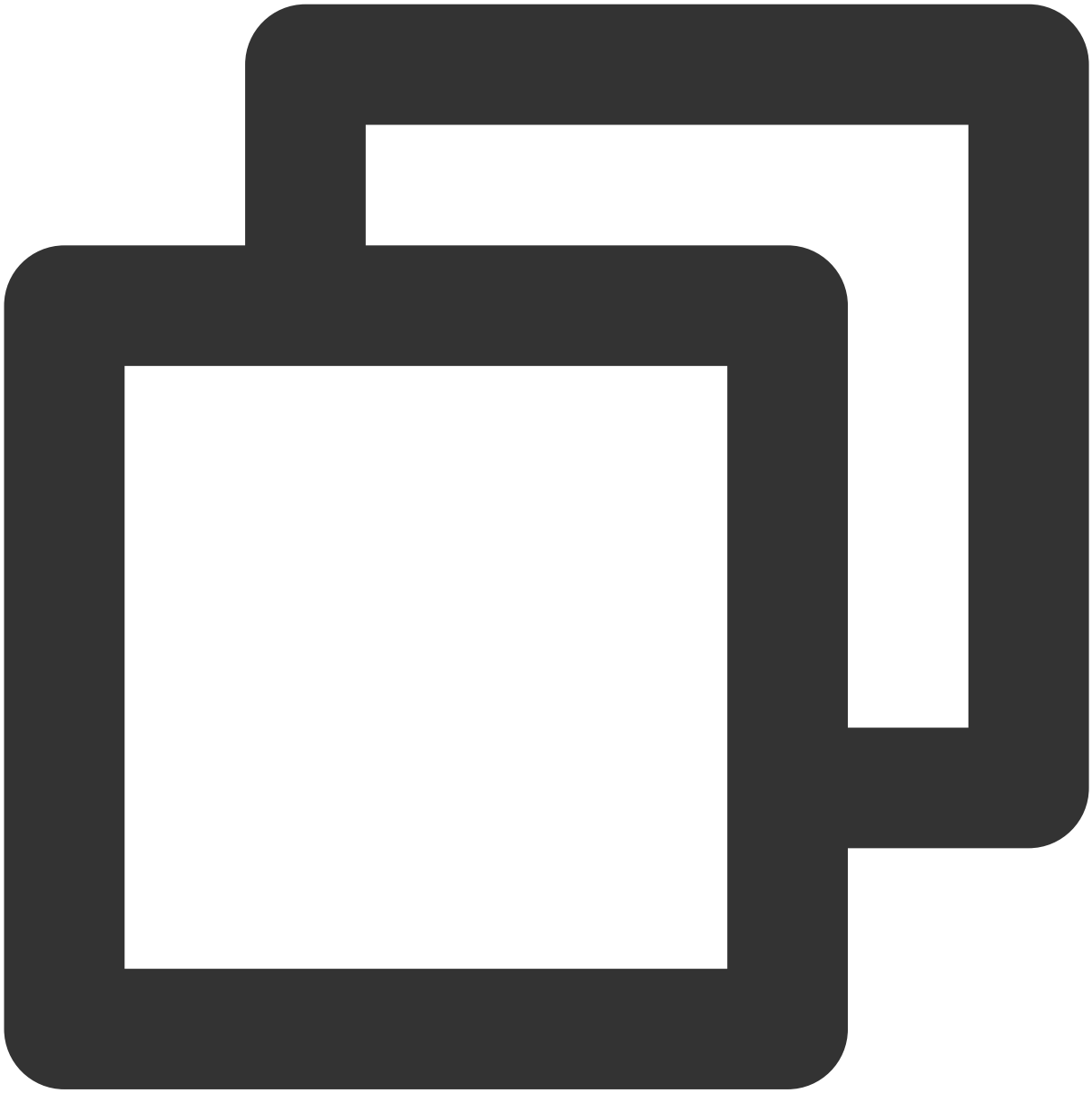
安装消息回应插件首先



```
flutter pub add tencent_cloud_chat_message_reaction
```

Add the following code to the `plugins` array in `initUIKit`:

加入以下代码至 `plugins` 数组在 `initUIKit` 中：



```
TencentCloudChatPluginItem(
  name: "messageReaction",
  pluginInstance: TencentCloudChatMessageReaction(
    context: context,
  ),
)
```

这将允许在您的应用程序中使用消息回应插件。

消息引用
Android & iOS

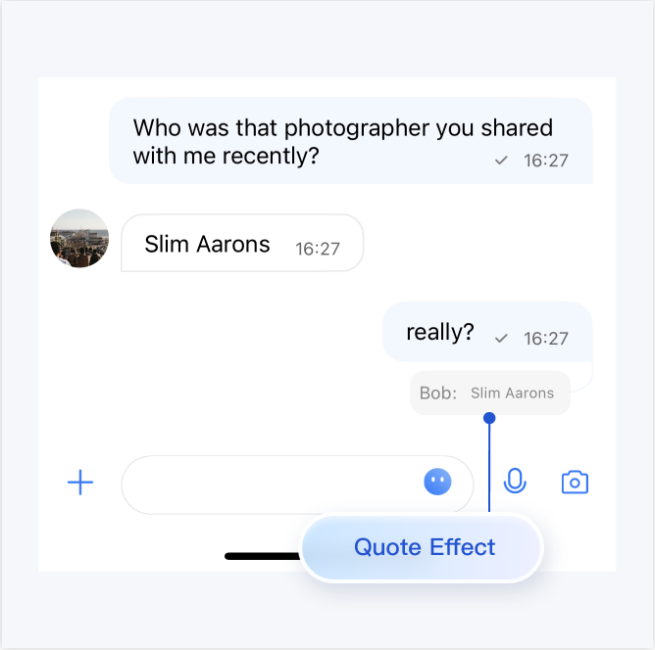
最近更新时间：2024-06-14 10:29:49

功能描述

在 TUIChat 的消息列表中，您可以引用之前的消息以表示对特定的消息进行回复。回复后还可以通过单击引用的消息跳转至原始消息，原始消息将会高亮闪烁。

效果展示

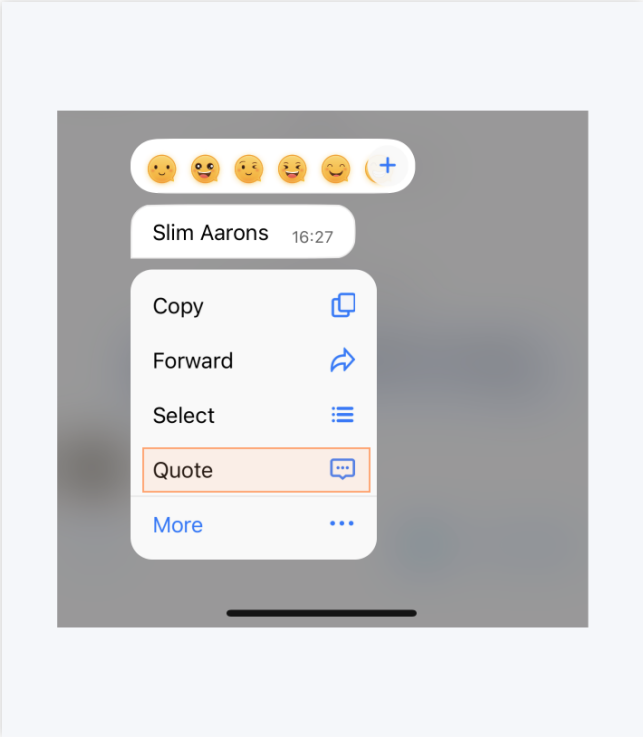
您可以在 TUIChat 消息列表中，长按消息引用体验效果如下：



功能说明

引用一条消息

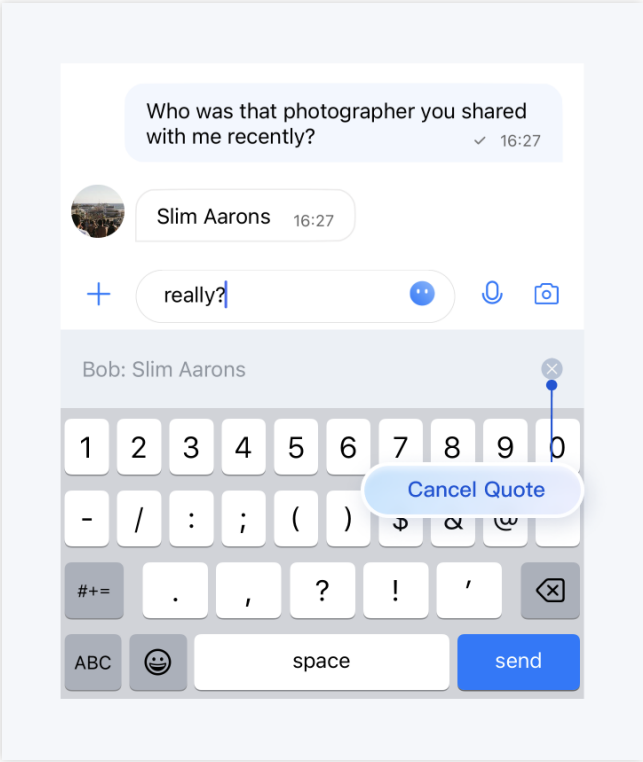
长按消息，消息上会弹出消息工具栏。单击工具栏中的引用按钮，对该消息进行引用。



取消消息引用

在消息被引用但还未发出时，通过单击引用之后的关闭按钮，可以取消消息引用。

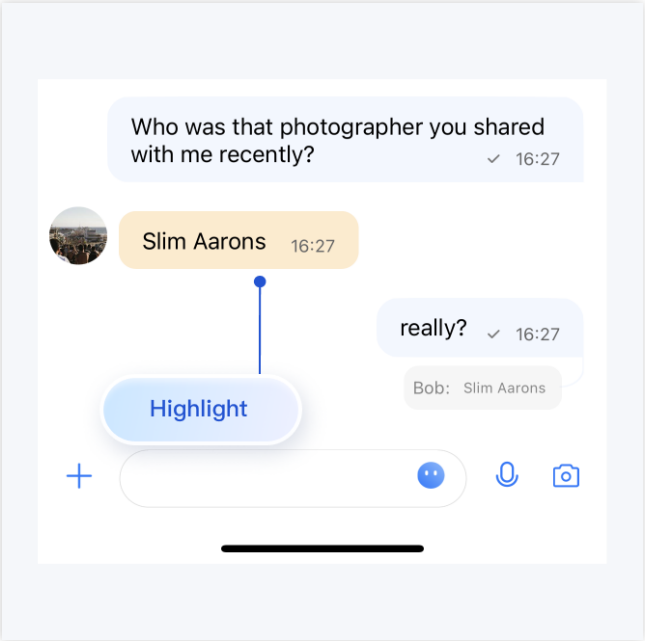
版权所有：腾讯云计算（北京）有限责任公司



查看被引用消息

单击引用消息的引用内容，可以定位到原始消息，原始消息将进行高亮闪烁：
当被引用的消息处于屏幕内，单击引用消息的引用内容，只进行高亮闪烁。
当被引用的消息不处于屏幕内，但处于消息列表中时，单击引用内容，消息列表会自动滚动到原始消息处，并进行高亮闪烁。
当被引用的消息不处于屏幕内，也不处于消息列表时，单击引用内容，不会跳转到原始消息，也不会高亮闪烁。

版权所有：腾讯云计算（北京）有限责任公司



联系我们

如果您对本功能有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Web & H5 & Uniapp（Vue）

最近更新时间：2024-06-14 10:28:00

功能描述

在 TUIChat 的会话中，您可以引用之前的消息以表示对特定的消息进行回复，回复后还可以通过点击引用的消息跳转至原始消息，原始消息将会高亮闪烁。

版权所有：腾讯云计算（北京）有限责任公司

消息引用

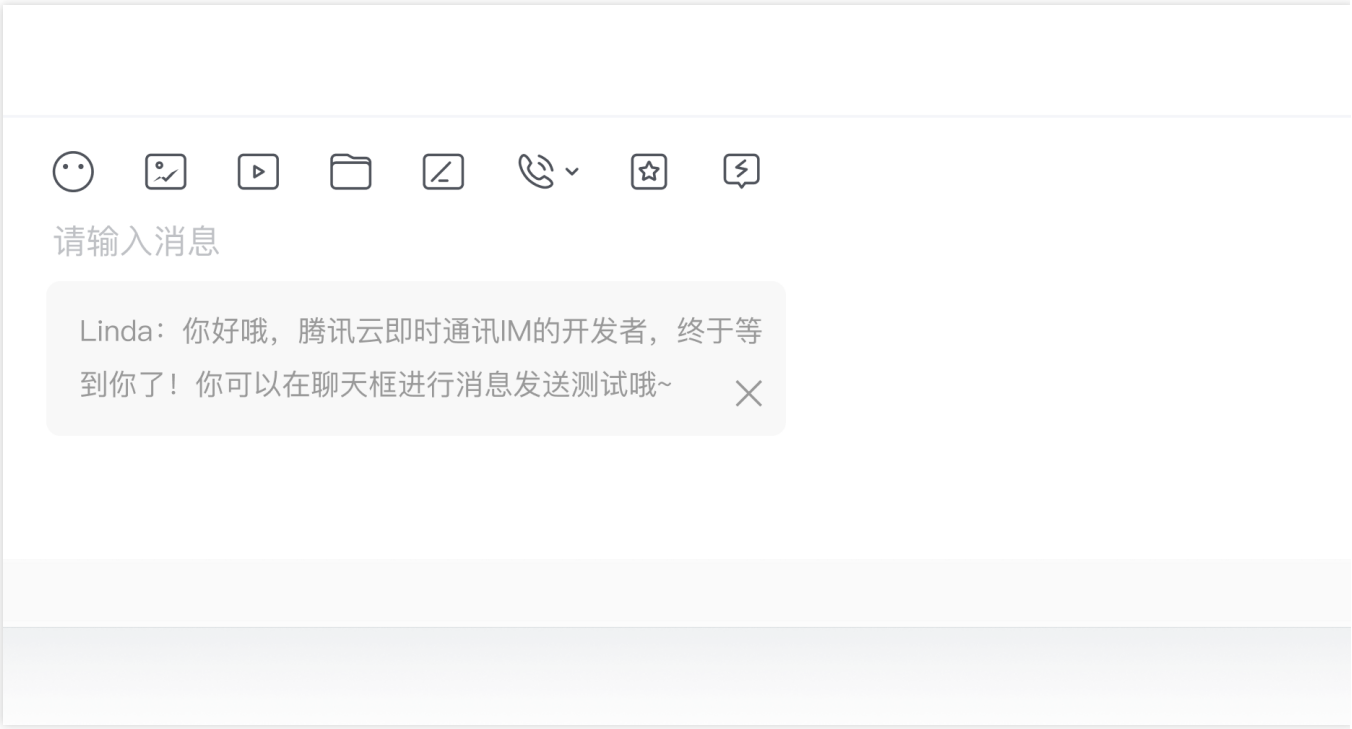
引用一条消息

Web 端右键消息，移动端长按消息，在消息上会弹出消息工具栏，点击工具栏中的引用按钮，即可对该消息进行引用。

版权所有：腾讯云计算（北京）有限责任公司

取消消息引用

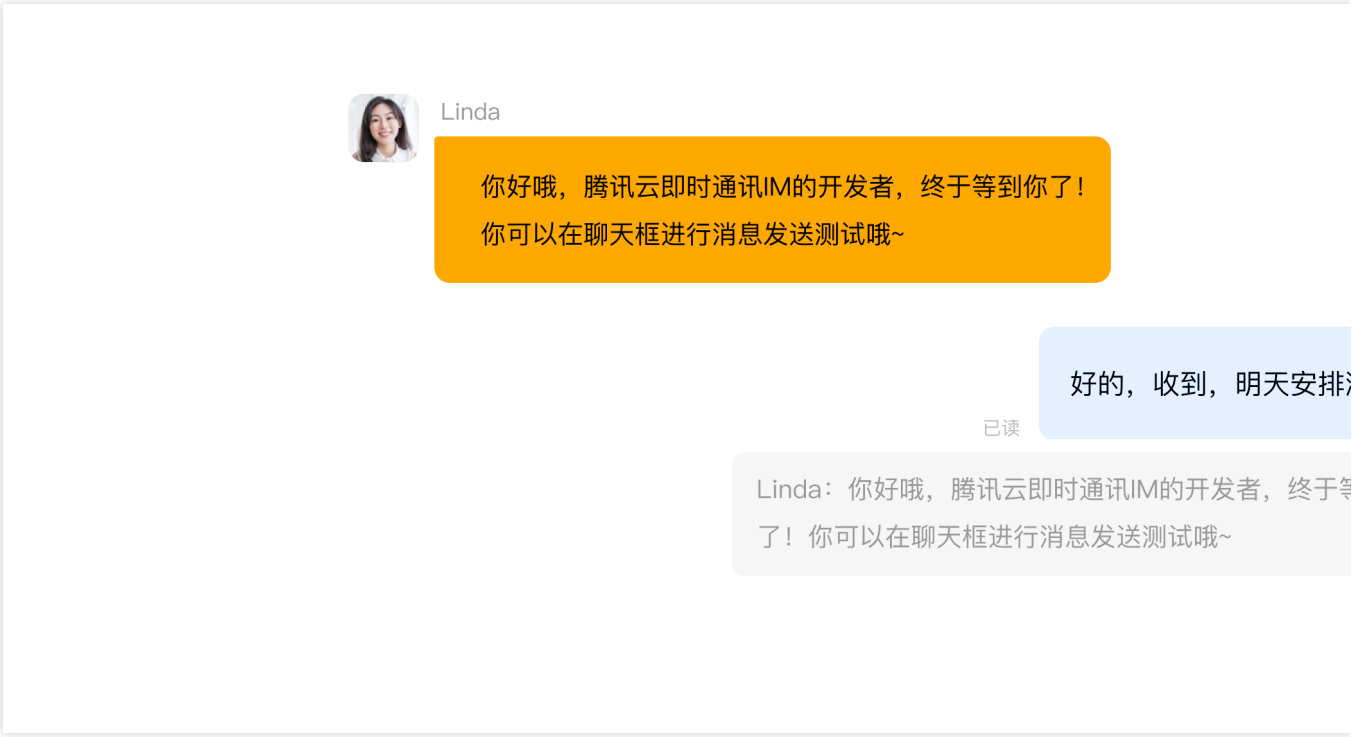
在消息被引用但还未发出时，通过点击引用之后的关闭按钮，可以取消消息引用。



查看被引用消息

点击引用消息的引用内容，可以定位到原始消息，原始消息将进行高亮闪烁：

当被引用的消息处于屏幕内，点击引用消息的引用内容，只进行高亮闪烁。
当被引用的消息不处于屏幕内，但处于消息列表中时，点击引用内容，消息列表会自动滚动到原始消息处，并进行高亮闪烁。
当被引用的消息不处于屏幕内，也不处于消息列表时，点击引用内容，不会跳转到原始消息，也不会高亮闪烁。



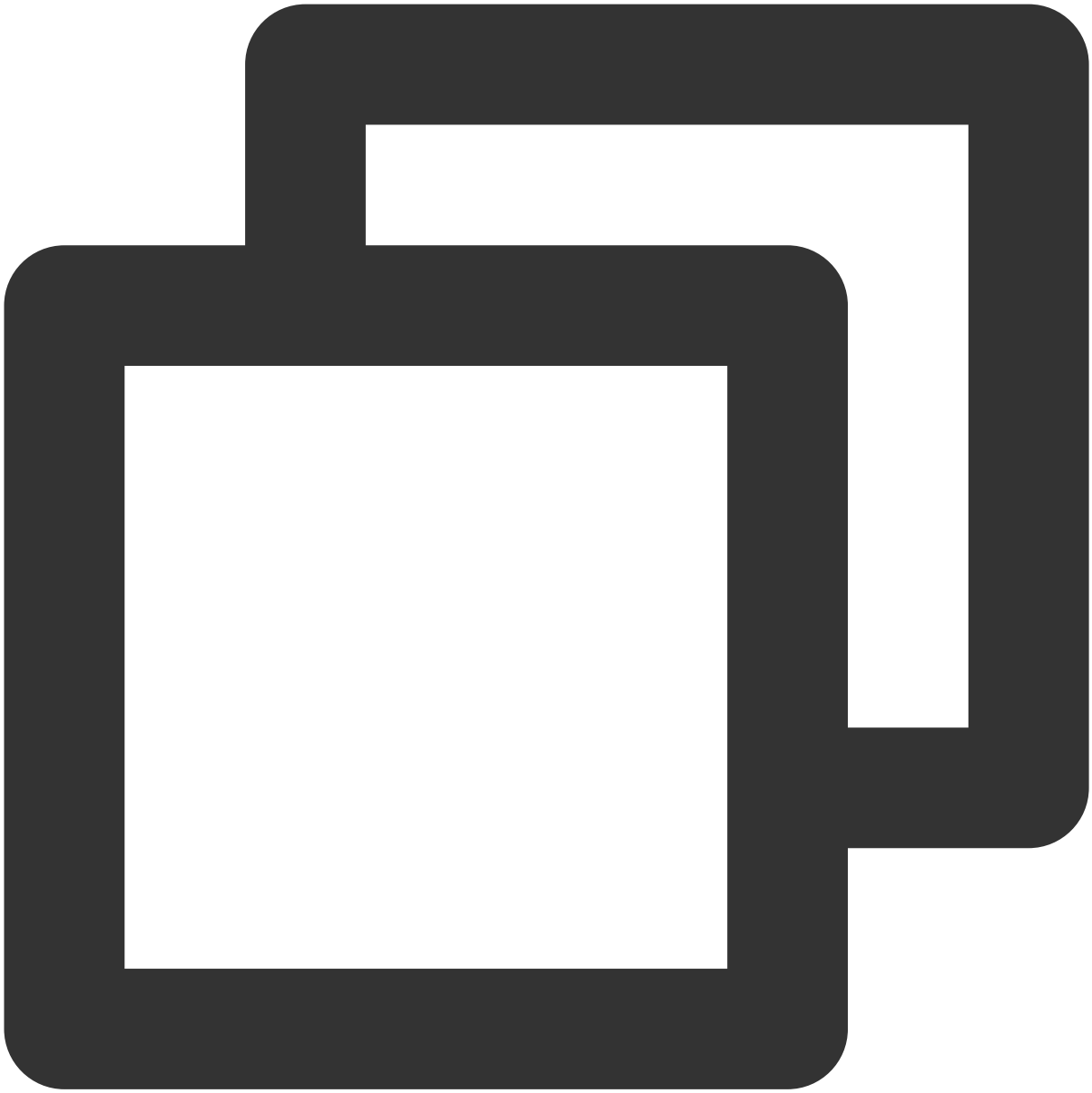
扩展资料

说明：
以下内容仅为辅助阅读资料，消息引用功能已在 TUIKit 中默认包含，不需要用户手动实现。

如何实现消息引用

引用一条消息

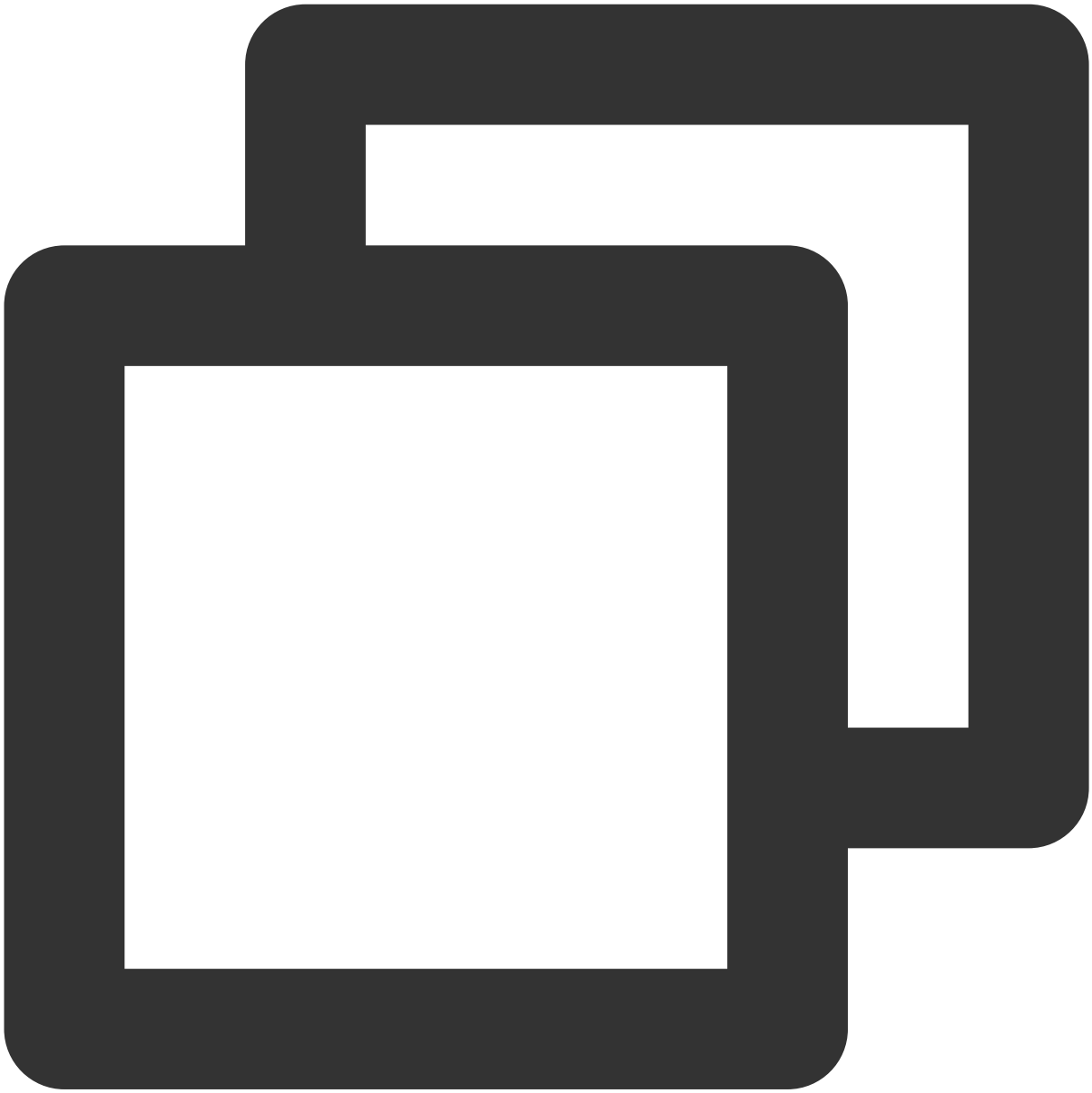
在点击引用按钮时，会调用消息对应 Message 上的 `quoteMessage()` 方法，并将消息的引用记录到 Store 里。



```
function quoteMessage(message) {
  message?.quoteMessage();
}
```

展示引用消息和取消引用

在输入框组件中监听 TUIStore 引用记录的变化，通过回调函数可以拿到被引用的消息。

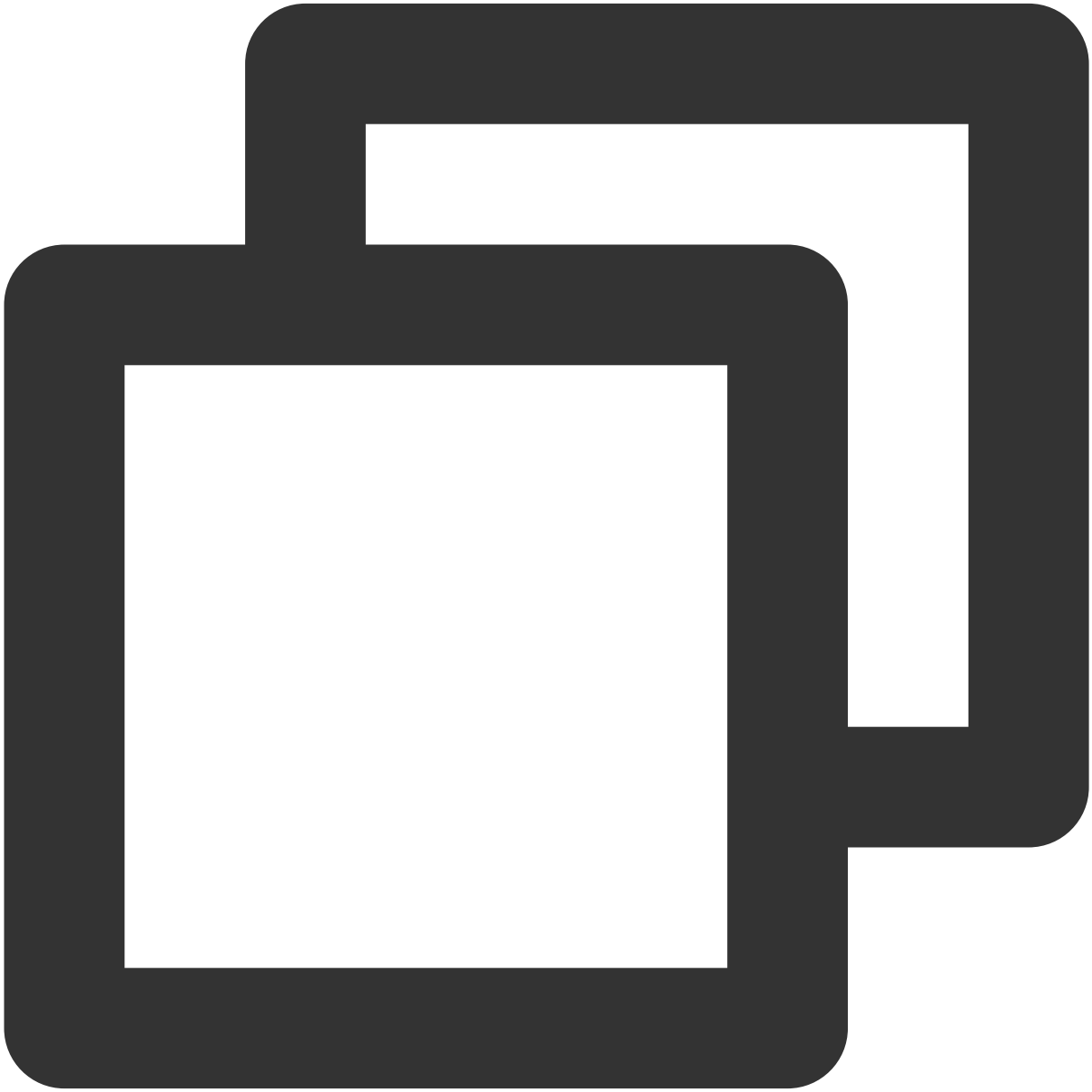


```
const quoteMessage = ref<IMessageModel>();

onMounted(() => {
  TUIStore.watch(StoreName.CHAT, {
    quoteMessage: (options: { message: IMessageModel, type: string }) => {
      if (options.message && options.type === "quote") {
        // 监听到新的消息引用
        quoteMessage.value = options.message;
      } else {
        // 监听到取消消息引用
        quoteMessage.value = undefined;
```

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

取消引用时，将引用记录从 TUIStore 中删除即可。

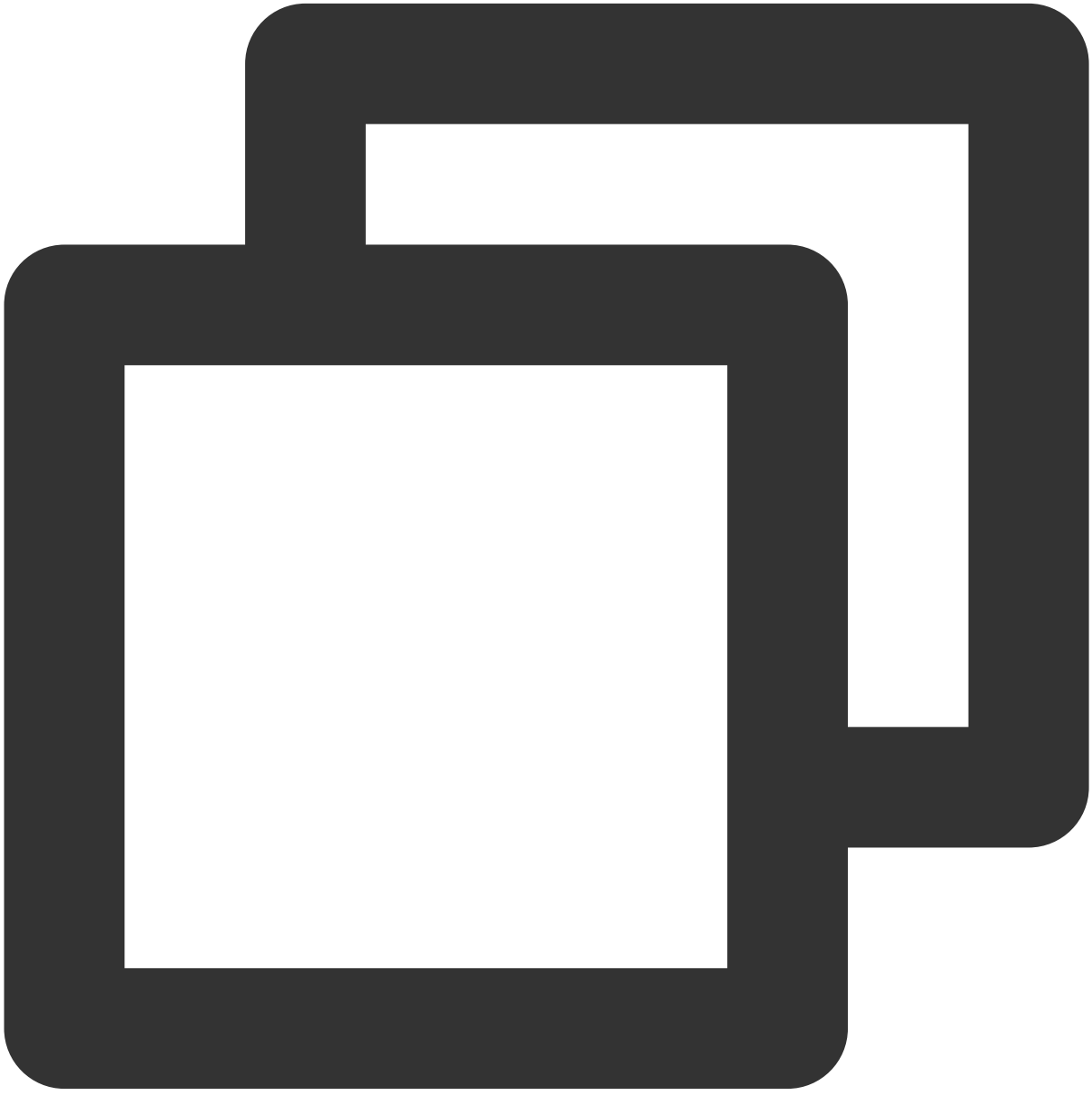


```
function cancelQuote() {
  TUIStore.update(StoreName.CHAT, "quoteMessage", { message: undefined, type: "quot
}
```

常见问题

我如何屏蔽引用功能？

在文件 `TUIKit/components/TUIChat/message-list/message-tool/index.vue` 中注释掉对象数组 `actionItems` 中的引用部分，如下所示：



```
actionItems = [
  {...},
  {...},
  // {
```

```
    //   text: TUITranslateService.t("TUIChat.引用"),
    //   iconUrl: quoteIcon,
    //   renderCondition() {
    //     if (!message.value) return false;
    //     const _message = TUIStore.getMessageModel(message.value.ID);
    //     return message.value?.status === "success" && !_message.getSignalingInfo()
    //   },
    //   clickEvent: quoteMessage,
    // }
]
```

交流与反馈

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

Flutter

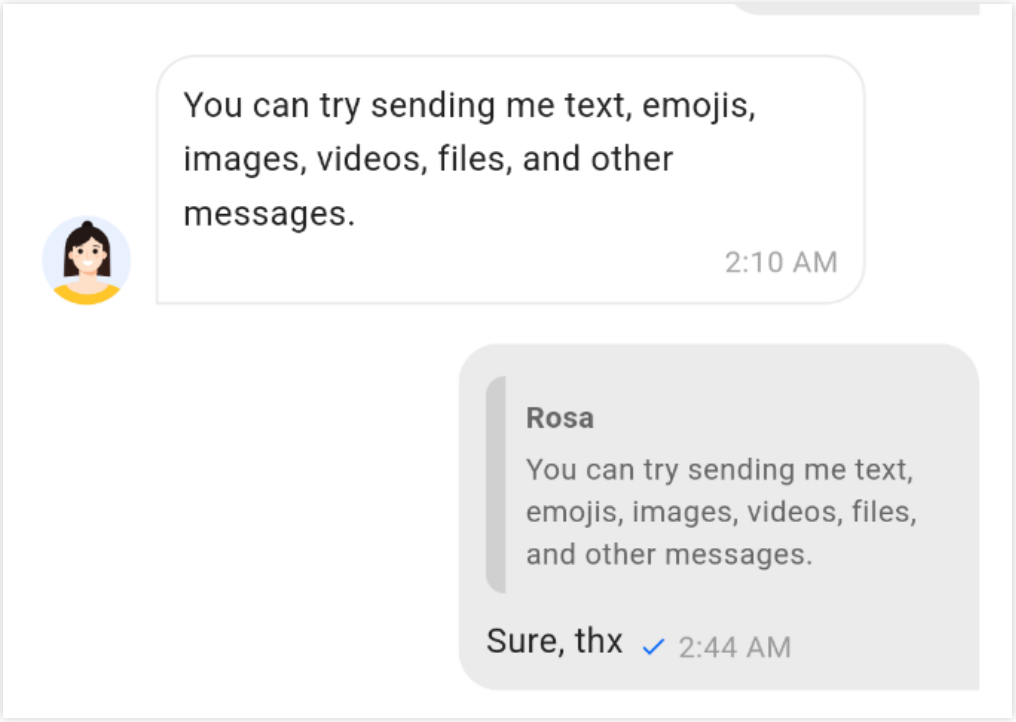
最近更新时间：2024-07-08 16:09:44

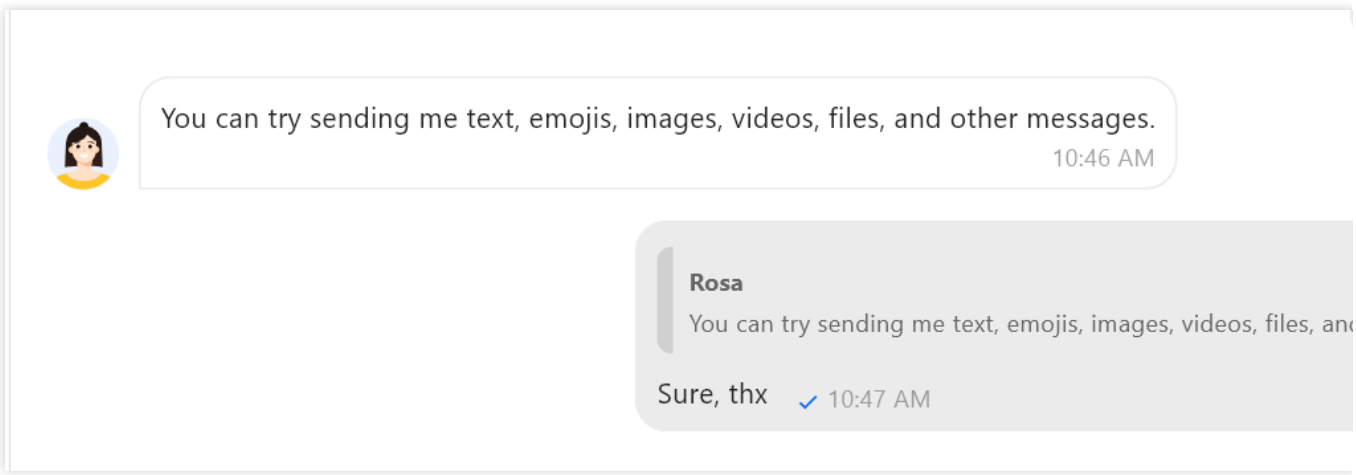
描述

在 TencentCloudChatMessage 的消息列表中，您可以通过引用特定的先前消息进行回复。回复后，点击被引用的消息将跳转到原始消息，并将其高亮显示。
引用消息有两种模式："消息引用"和"消息回复"。
消息引用：仅引用消息。消息上下文菜单中显示的文本为"引用"。
消息回复：引用并回复消息，在群聊中提及消息发送者。消息上下文菜单中显示的文本为"回复"。

效果展示

您可以在 TencentCloudChatMessage 消息列表中，长按消息引用体验效果如下：
移动端
桌面端





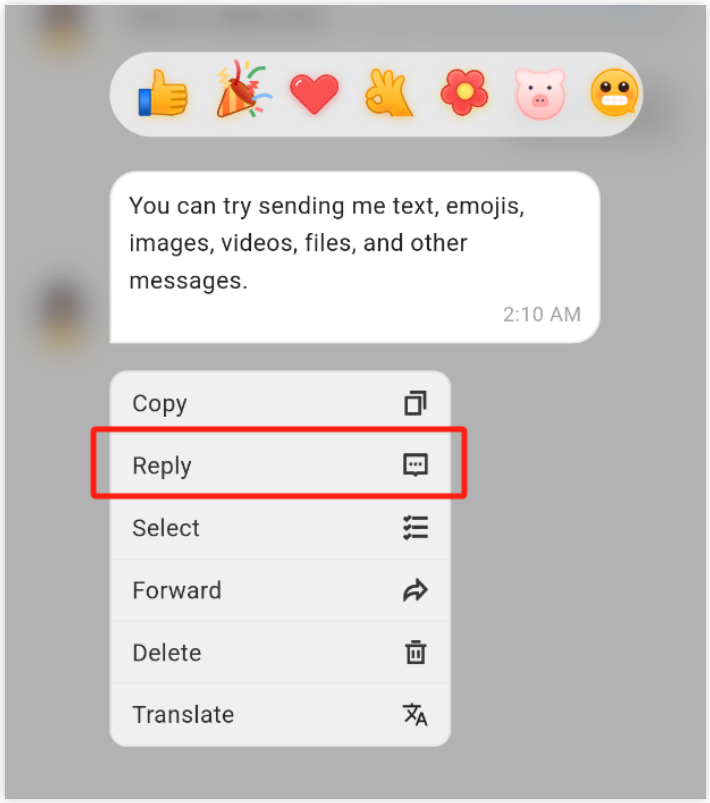
功能说明

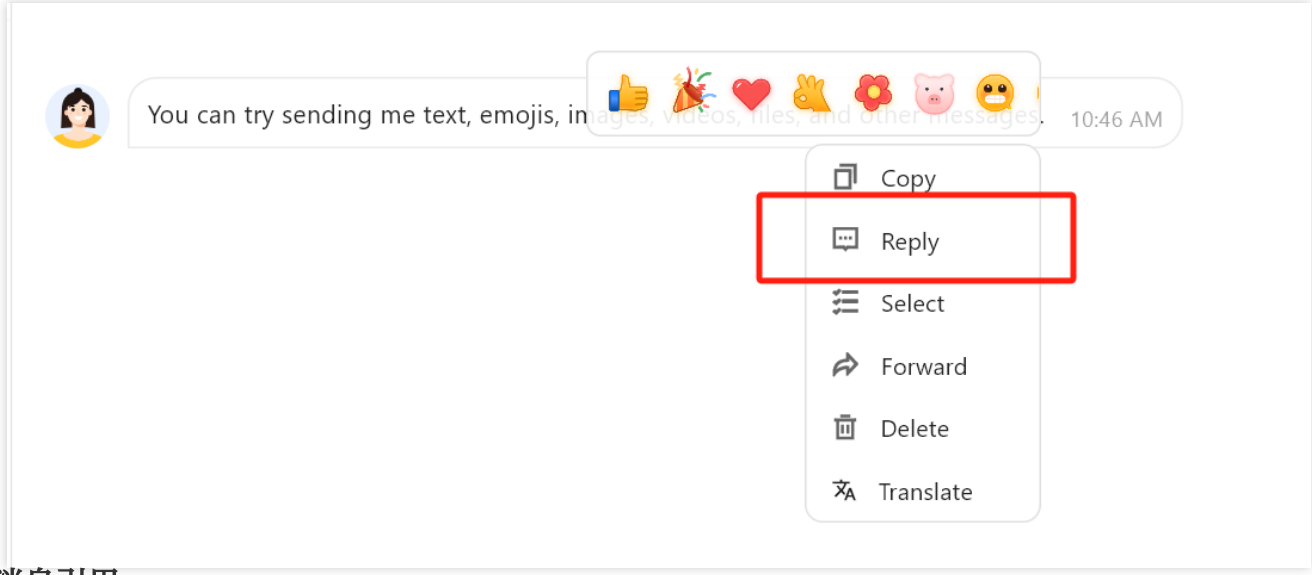
引用一条消息

长按消息，消息上会弹出消息工具栏。单击工具栏中的引用按钮，对该消息进行引用。

移动端

桌面端



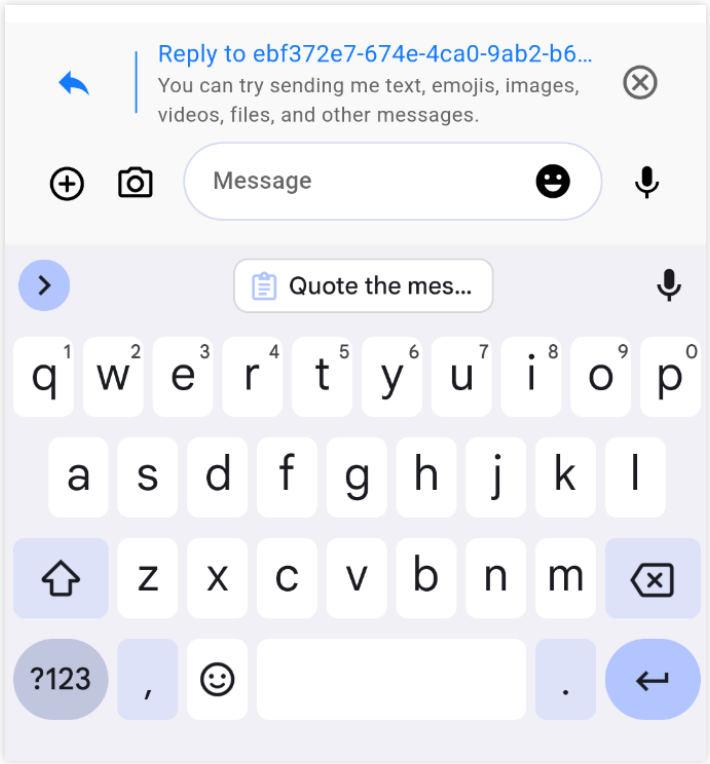


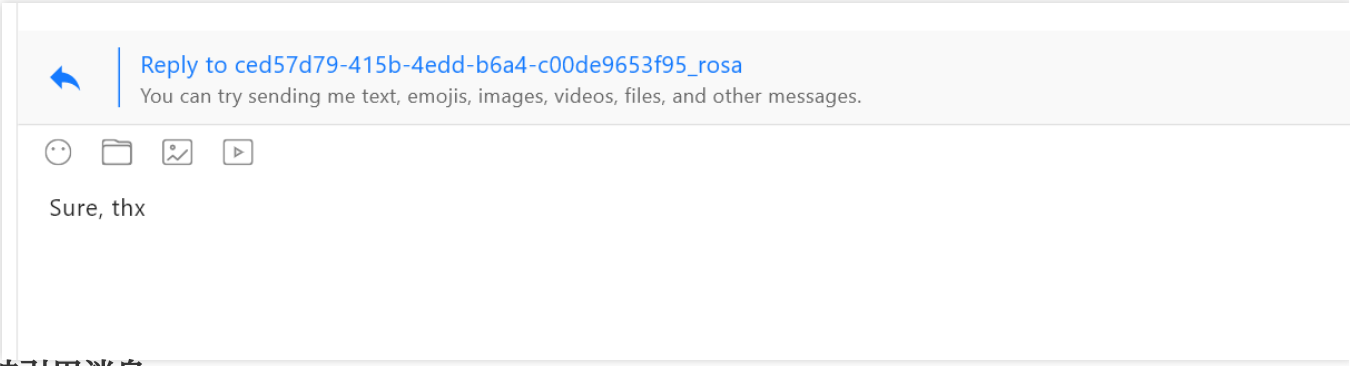
取消消息引用

在消息被引用但还未发出时，通过单击引用之后的关闭按钮，可以取消消息引用。

移动端

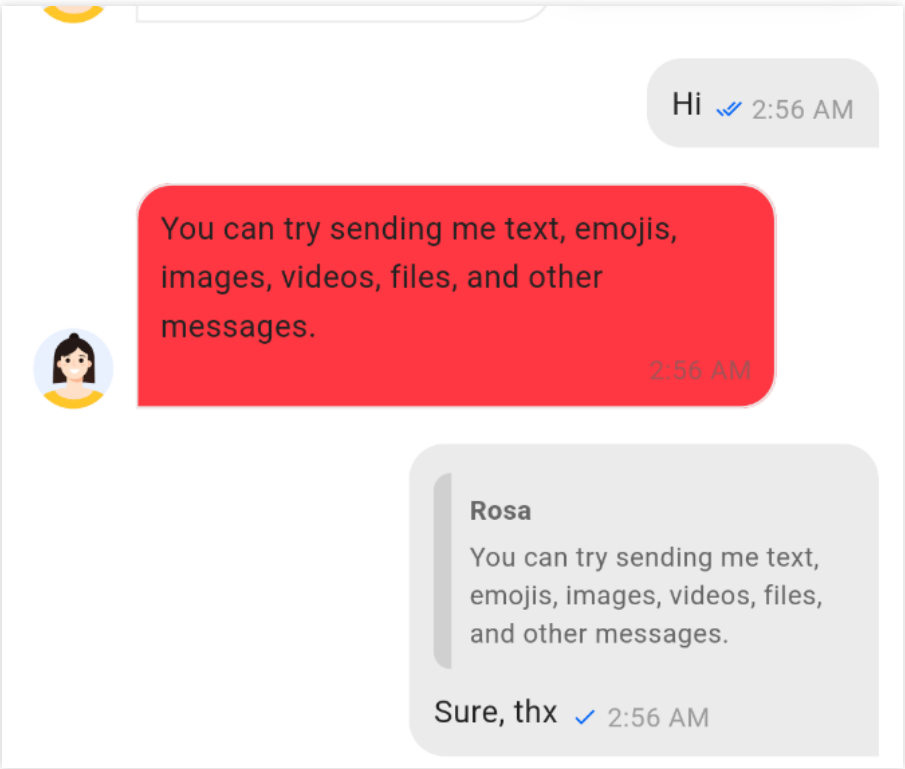
桌面端

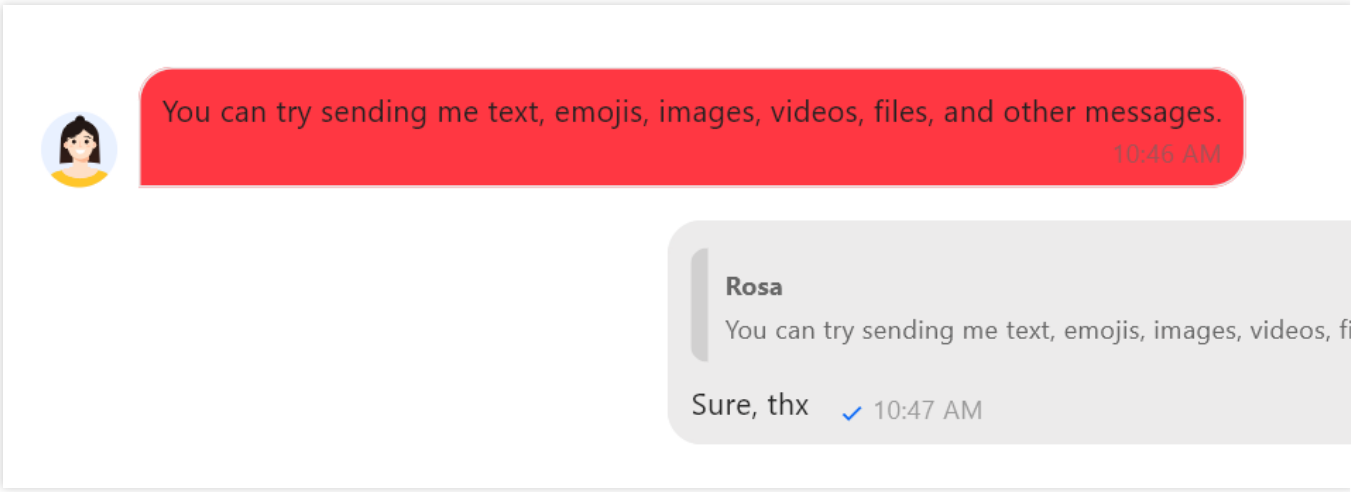




查看被引用消息

单击引用消息的引用内容，可以定位到原始消息，原始消息将进行高亮闪烁：
当被引用的消息处于屏幕内，单击引用消息的引用内容，只进行高亮闪烁。
当被引用的消息不处于屏幕内，但处于消息列表中时，单击引用内容，消息列表会自动滚动到原始消息处，并进行高亮闪烁。
当被引用的消息不处于屏幕内，也不处于消息列表时，单击引用内容，不会跳转到原始消息，也不会高亮闪烁。
移动端
桌面端



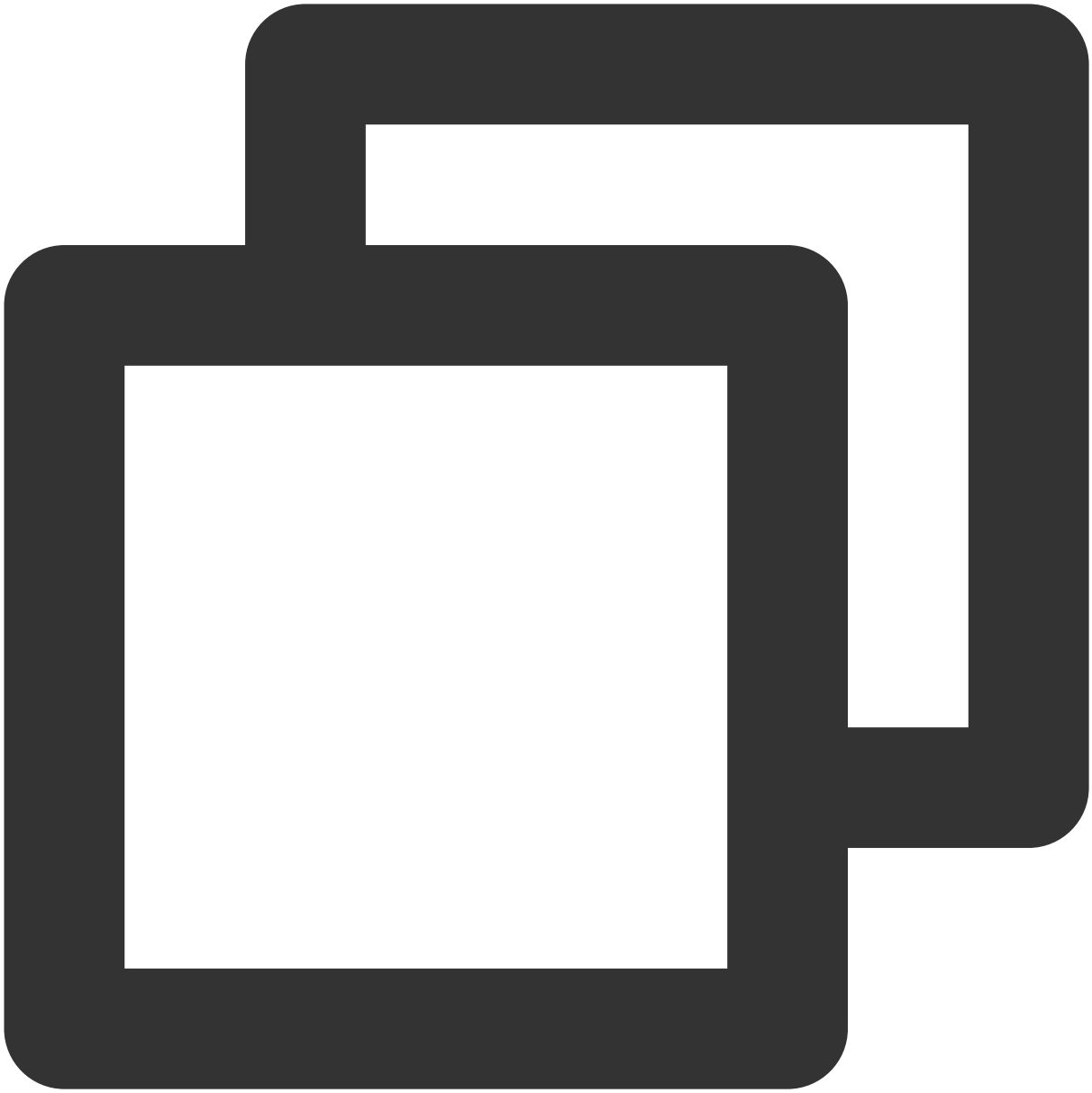


使用方法

此模块默认自动启用。

您可以在 `TencentCloudChatMessageConfig` 中指定 `enableReplyWithMention` 以选择使用哪种引用消息模式。

示例：



```
TencentCloudChatMessageConfig(
  enableReplyWithMention: ({String? groupID, String? userID, String? topicID}) => t
)
```

联系我们

如果您对本功能有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

版权所有：腾讯云计算（北京）有限责任公司

文本消息翻译
Android & iOS

最近更新时间：2024-06-14 15:23:43

功能描述

文本消息翻译功能：当您进入了聊天界面后，可以手动长按消息列表中的文本消息气泡，在出现的菜单中，单击翻译按钮，翻译文本。

注意

文本消息翻译功能由插件提供，使用需集成 `TUITranslationPlugin` ，7.2 及以上版本支持。

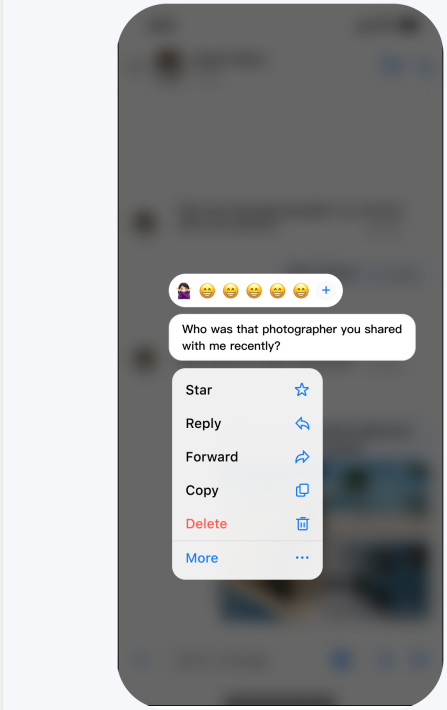
文本翻译是增值付费功能。当前此功能处于内测阶段，您可通过 Telegram 技术交流群 联系我们们为您开通体验完整功能。

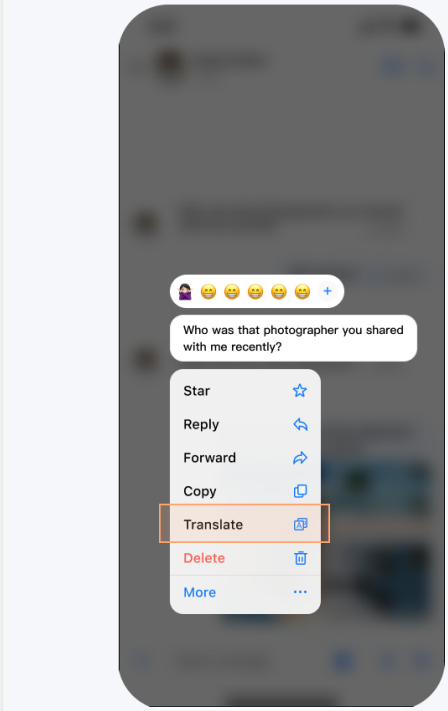
效果展示

集成翻译服务前后效果图如下所示：

简约版

未集成翻译插件，不显示翻译按钮	集成了翻译插件，显示翻译按钮





功能说明

集成插件

从 7.2 版本开始，翻译功能由插件 `TUITranslationPlugin` 提供。
如果您不需要翻译功能，不集成该插件即可。长按文本消息时，不会显示翻译按钮。
如果您需要翻译功能，需集成 `TUIChat` 和 `TUITranslationPlugin` 。集成方法请参见《集成 TUIKit》（Android/iOS）。集成后不需要进行任何设置，长按文本消息时，自动显示翻译按钮。
集成 `TUITranslationPlugin` 后，您还可以设置翻译的目标语言。默认目标语言是当前 `TUIChat` 所使用的语言。

注意：

`TUITranslationPlugin` 依赖于 `TUIChat` ，不能单独集成。
仅支持文本消息、文本类的引用或回复消息，图片、语音、视频、文件、表情、自定义消息等不支持翻译。
并不是任意源语言都能翻译成设定的目标语言。例如，英语可以翻译成印地语，但是中文不可以翻译成印地语。目前已经支持的翻译语言请参见 文本翻译语言支持。如果翻译失败，请查询该文档切换源语言或目标语言。

联系我们

如果您对本功能有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Web & H5 & Uniapp（Vue）

最近更新时间：2024-06-14 10:28:36

功能描述

TUIKit 从 v2.1.0 版本之后开始提供文本消息翻译功能，TUIChat 组件中的消息工具支持对文本消息进行翻译，目前可以将主流语言翻译为英语、中文、俄语、日语、韩语等多种语言，具体支持情况参见 文本翻译支持列表。
文本翻译后，翻译的结果会在本次登录中持续保留，切换会话后翻译结果依然保存。如果切换会话且更换了翻译目标语言，之前的翻译结果会自动翻译为新的目标语言。

注意：

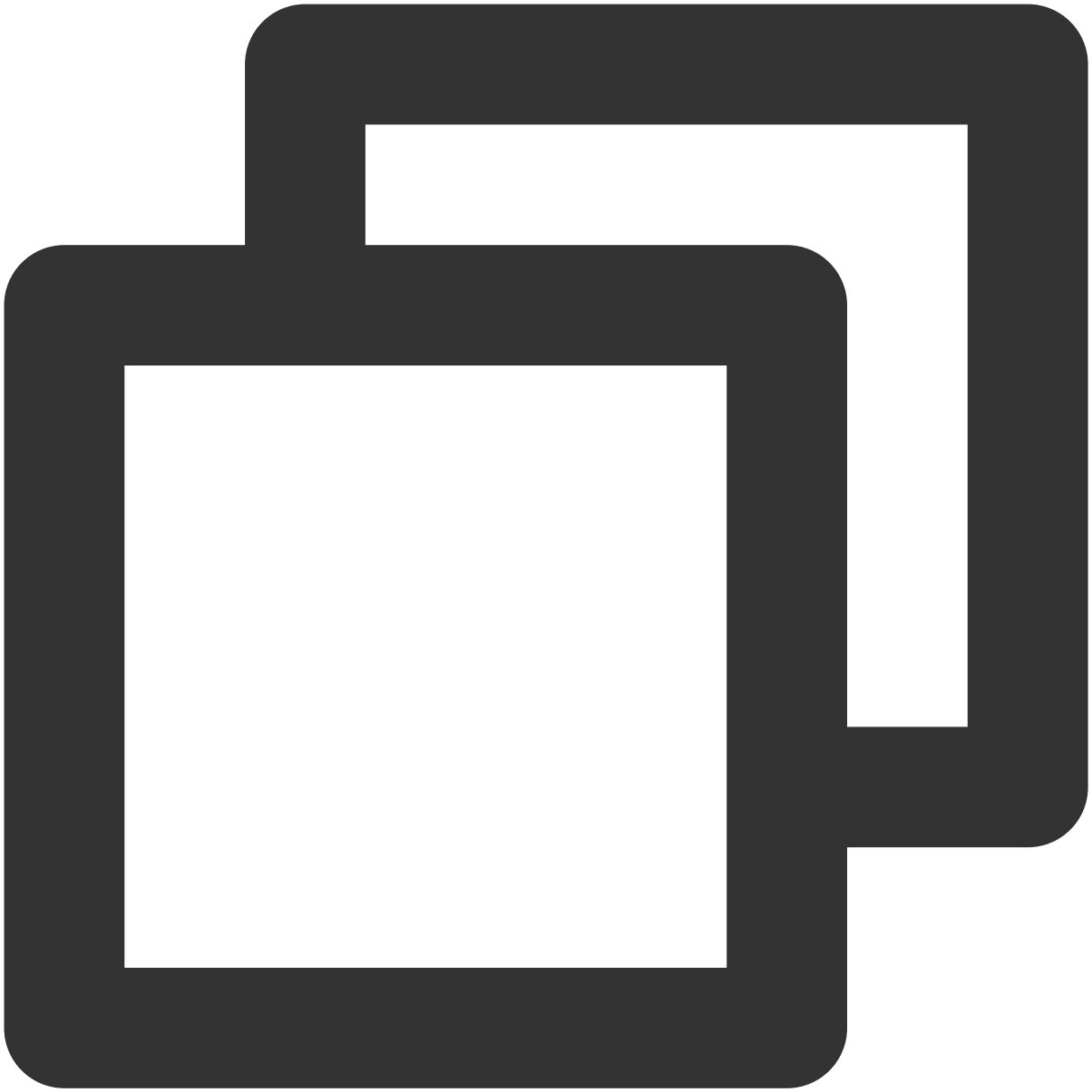
“文本消息翻译”功能处于限免阶段，您可通过 Telegram 技术交流群组 联系我们为您开通体验完整功能。

版权所有：腾讯云计算（北京）有限责任公司

功能介绍

设置翻译的目标语言

翻译的目标语言默认为中文，也可以手动切换翻译目标语言，从 `@tencentcloud/chat-uikit-engine` 导出 `TUIChatService` 并调用 `setTranslationLanguage` 方法，如下文代码所示：



```
import { TUIChatService } from "@tencentcloud/chat-uikit-engine";

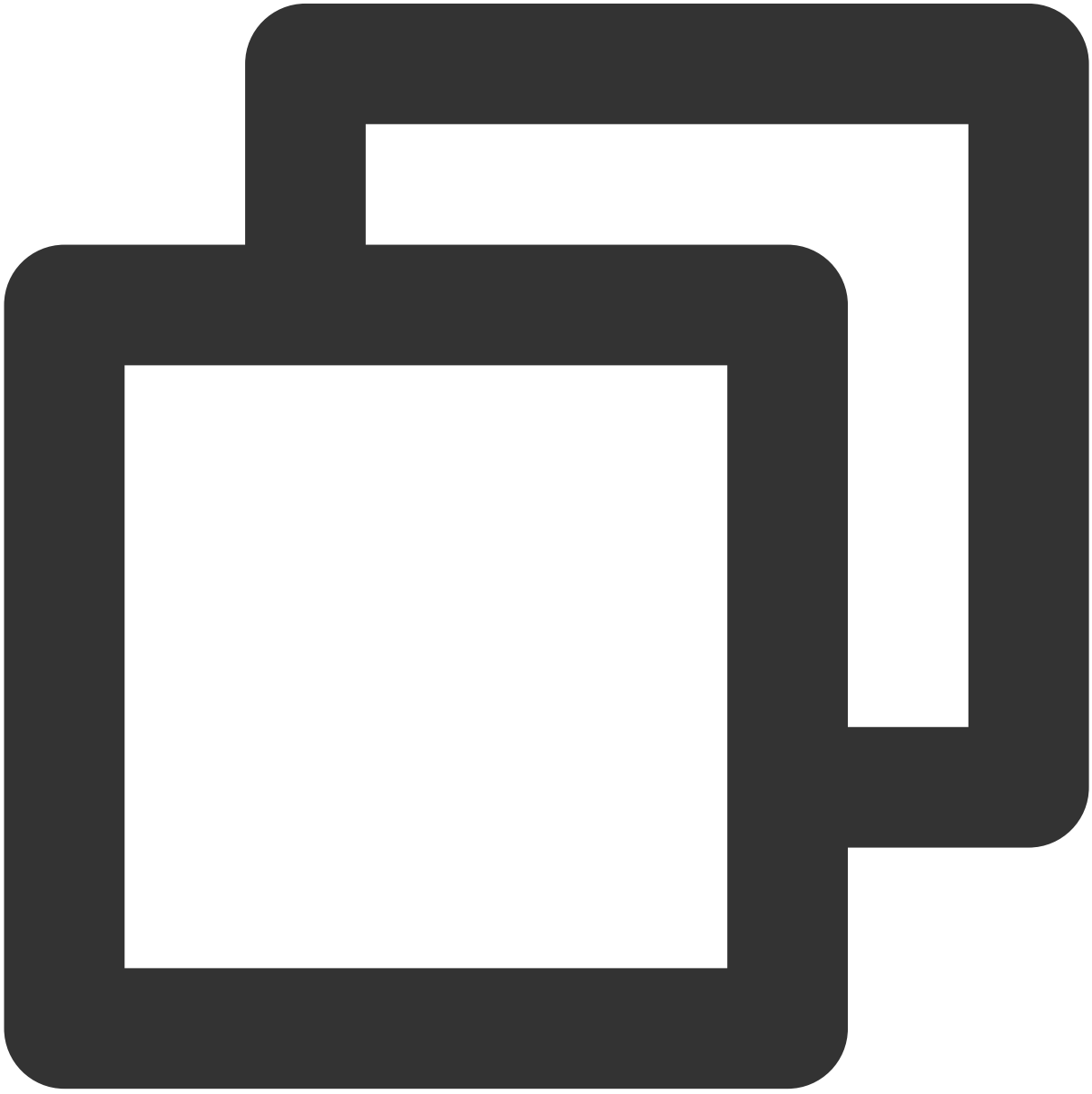
// 切换为英文
TUIChatService.setTranslationLanguage('en');
// 切换为中文
TUIChatService.setTranslationLanguage('zh');
// 切换为日语 也支持其他语言
TUIChatService.setTranslationLanguage('jp');
```

说明：

重新登录后会恢复为默认翻译为中文。

设置翻译的源语言

目前在 TUIKit 中**默认且建议**您使用自动检测源语言，但是也可以手动设置源语言以保证翻译的准确性。
在 `TUIChat/utils/translation.ts` 中实现了一个翻译的工具类 `Translator`，其中可以按照 文本翻译语言支持列表 更改 `sourceLanguage` 这一属性。



```
TUIChatService.translateText({
    sourceTextList: [text],
    sourceLanguage: 'auto', // zh/en/jp/kr
```

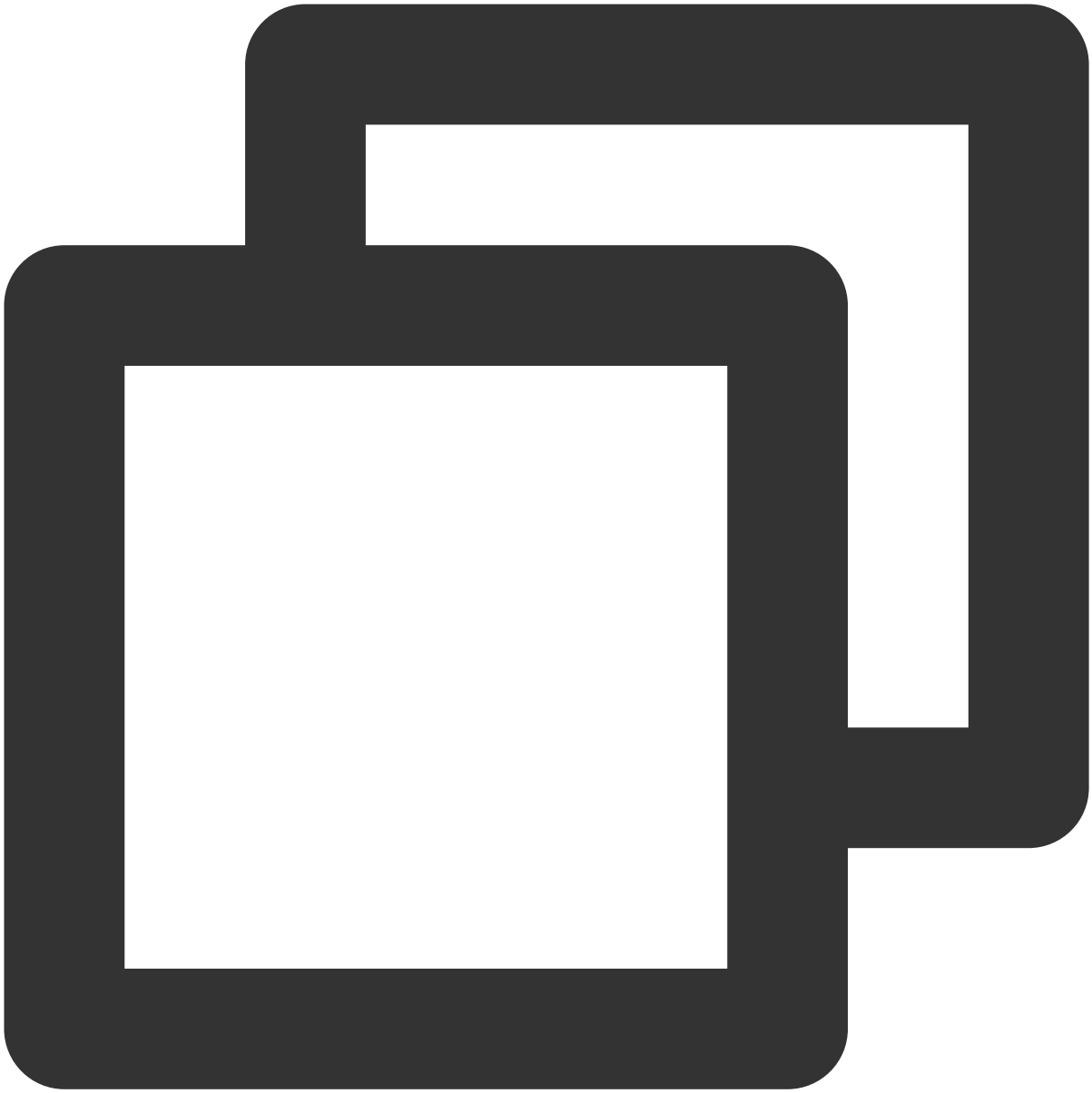
```
}).then((response) => {
    // response data
});
```

扩展资料

以下内容仅为辅助阅读资料，文本消息翻译以及相关功能已在 TUIKit 2.1.0 之后实现，不需要用户手动实现。

查询当前套餐是否支持翻译功能

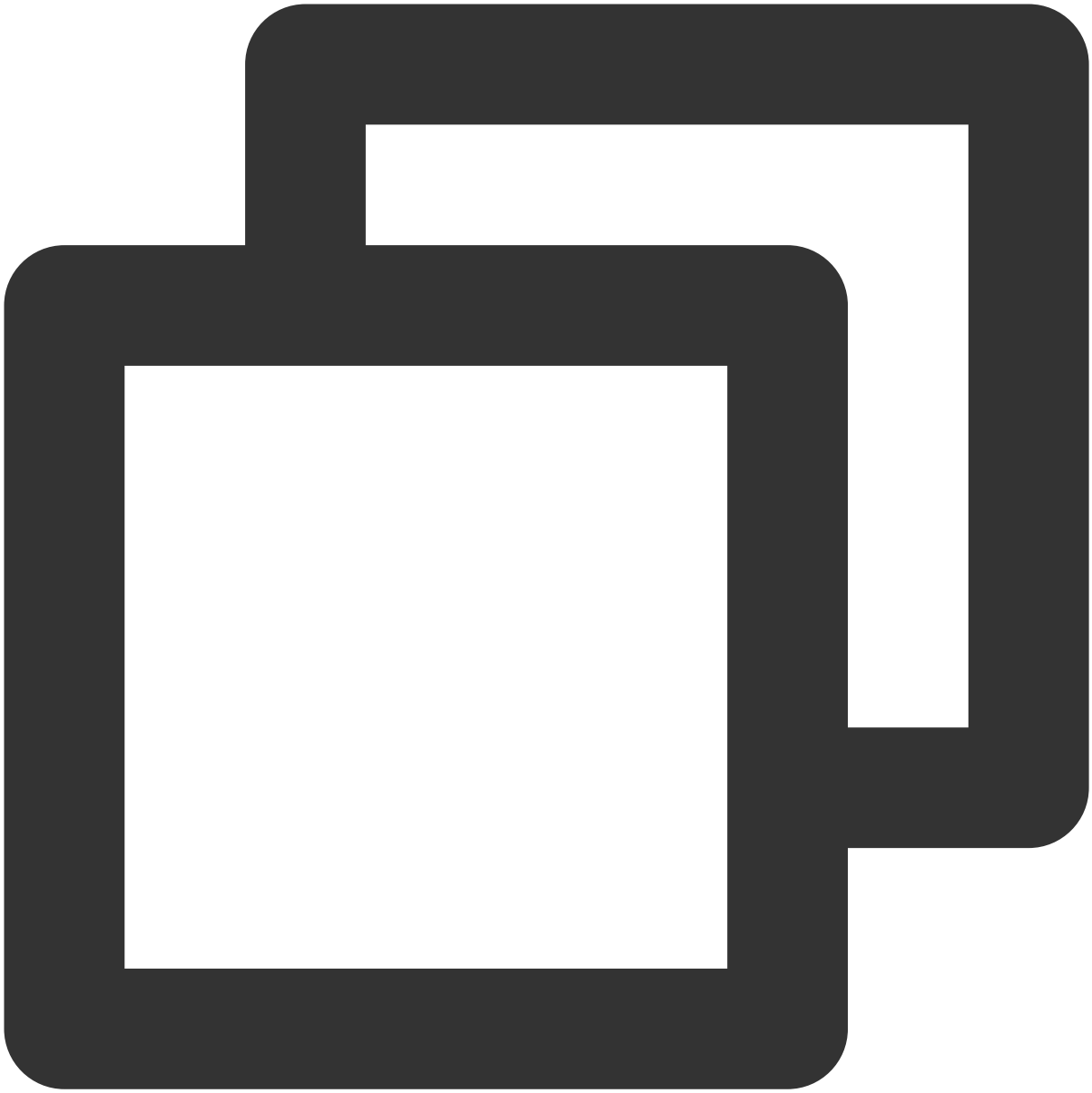
通过 `TUIStore` 查询 `StoreName.APP` 中 `enabledTranslationPlugin` 这个 key，得到的布尔值即是开通结果。



```
const enable = TUIStore.getData(StoreName.APP, 'enabledTranslationPlugin');
```

移除翻译功能

在文件 `TUIKit/components/TUIChat/message-list/message-tool/index.vue` 中，删除 `actionItems` 中 key 为 `translate` 的对象即可。



```
{
    key: 'translate',
    text: TUITranslateService.t('TUIChat.翻译'),
    visible: false,
    iconUrl: translateIcon,
    renderCondition() {},
    clickEvent: translateMessage,
}
```

文本翻译语言支持列表

源语言	支持的目标语言
zh（简体中文）	en（英语）、ja（日语）、ko（韩语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）、vi（越南语）、id（印尼语）、th（泰语）、ms（马来语）
zh-TW（繁体中文）	en（英语）、ja（日语）、ko（韩语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）、vi（越南语）、id（印尼语）、th（泰语）、ms（马来语）
en（英语）	zh（中文）、ja（日语）、ko（韩语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）、vi（越南语）、id（印尼语）、th（泰语）、ms（马来语）、ar（阿拉伯语）、hi（印地语）
ja（日语）	zh（中文）、en（英语）、ko（韩语）
ko（韩语）	zh（中文）、en（英语）、ja（日语）
fr（法语）	zh（中文）、en（英语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）
es（西班牙语）	zh（中文）、en（英语）、fr（法语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）
it（意大利语）	zh（中文）、en（英语）、fr（法语）、es（西班牙语）、de（德语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）
de（德语）	zh（中文）、en（英语）、fr（法语）、es（西班牙语）、it（意大利语）、tr（土耳其语）、ru（俄语）、pt（葡萄牙语）
tr（土耳其语）	zh（中文）、en（英语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、ru（俄语）、pt（葡萄牙语）
ru（俄语）	zh（中文）、en（英语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、pt（葡萄牙语）
pt（葡萄牙语）	zh（中文）、en（英语）、fr（法语）、es（西班牙语）、it（意大利语）、de（德语）、tr（土耳其语）、ru（俄语）
vi（越南语）	zh（中文）、en（英语）
id（印尼语）	zh（中文）、en（英语）
th（泰语）	zh（中文）、en（英语）
ms（马来语）	zh（中文）、en（英语）
ar（阿拉伯语）	en（英语）

版权所有：腾讯云计算（北京）有限责任公司

hi（印地语）	en（英语）

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-07-08 16:08:43

功能描述

文本消息翻译功能：当您进入了聊天界面后，可以手动长按消息列表中的文本消息气泡，在出现的菜单中，单击翻译按钮，翻译文本。

注意：

文本消息翻译功能由插件提供，使用需集成 `tencent_cloud_chat_text_translate` ，1.4.1 及以上版本支持。

文本翻译是增值付费功能。当前此功能处于内测阶段，您可通过 Telegram 技术交流群 联系我们们为您开通体验完整功能。

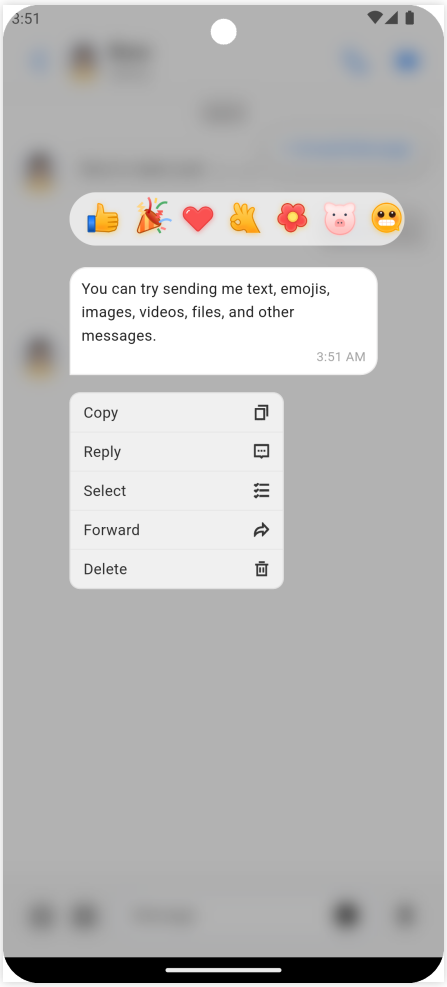
效果展示

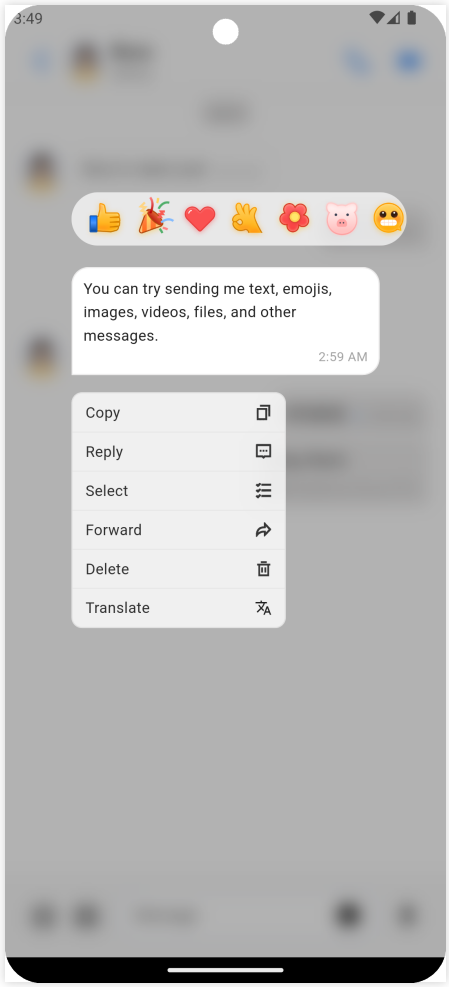
集成翻译服务前后效果图如下所示：

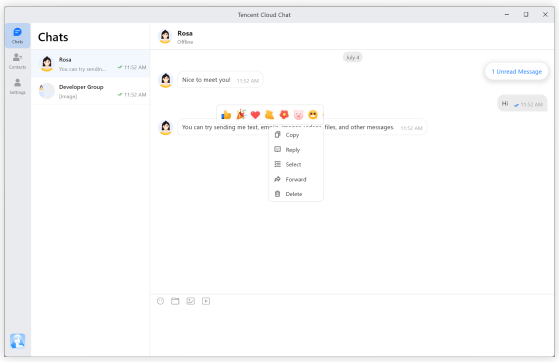
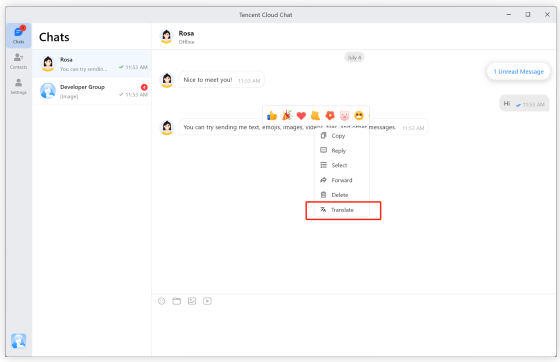
移动端

桌面端

未集成翻译插件，不显示翻译按钮	集成了翻译插件，显示翻译按钮	文本消





未集成翻译插件，不显示翻译按钮	集成了翻译插件，显示翻译按钮
	

功能概览

集成插件

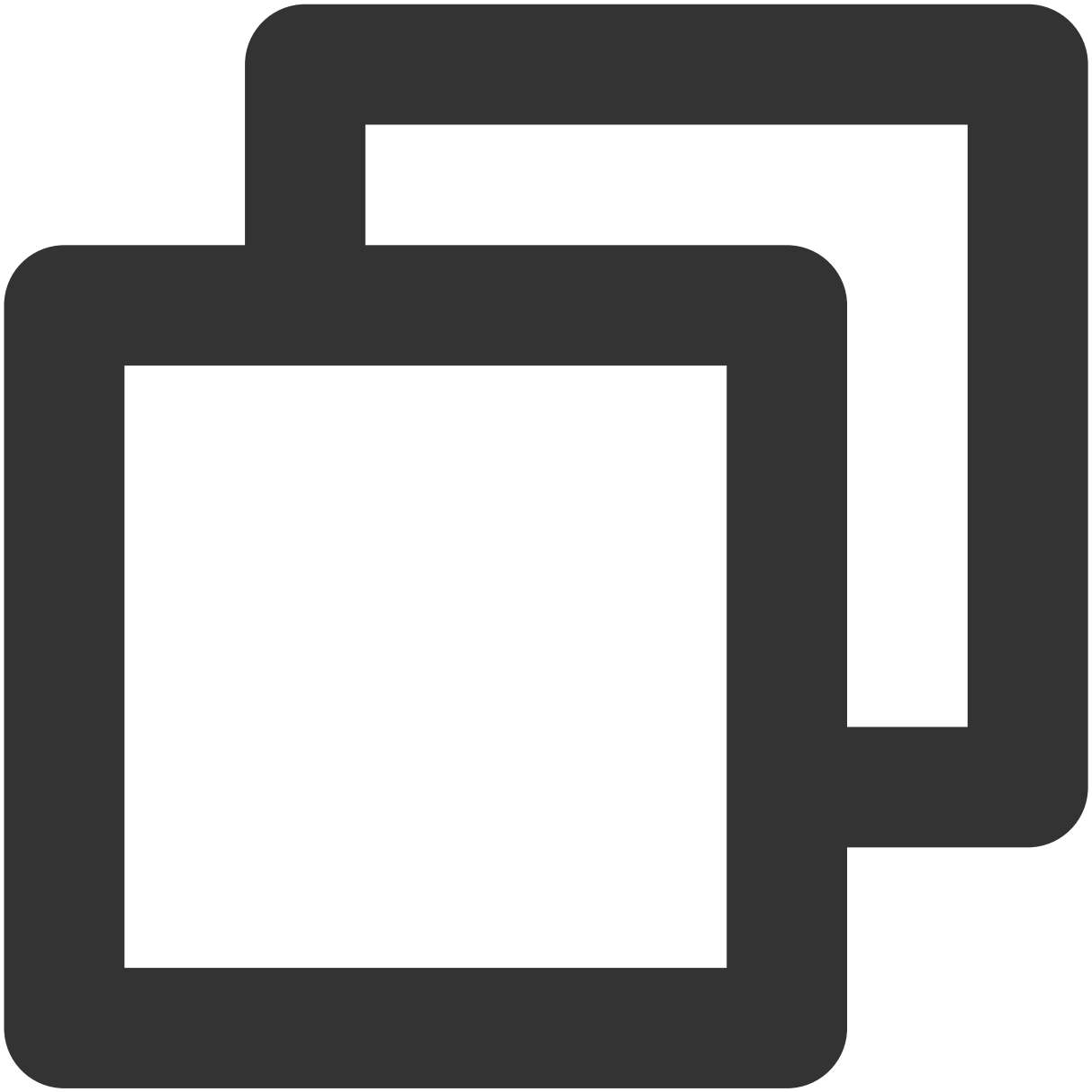
从1.4.1版本开始，翻译功能由插件 `tencent_cloud_chat_text_translate` 提供。
如果您不需要翻译功能，则无需集成该插件。长按文本消息时不会显示翻译按钮。
如果您需要翻译功能，必须集成 `tencent_cloud_chat_message` 和 `tencent_cloud_chat_text_translate` 。长按文本消息时会自动显示翻译按钮。
集成 `tencent_cloud_chat_text_translate` 后，您还可以设置翻译的目标语言。默认目标语言为UIKit当前使用的语言。

注意：

tencent_cloud_chat_text_translate依赖于tencent_cloud_chat_message，不能单独集成。
仅支持文本消息和文本类型的引用或回复。不支持图片、语音、视频、文件、表情和自定义消息的翻译。
不是所有源语言都可以翻译成设置的目标语言。例如，英语可以翻译成印地语，但中文不能翻译成印地语。目前支持的翻译语言请参考支持的文本翻译语言。如果翻译失败，请参考该文档更改源语言或目标语言。

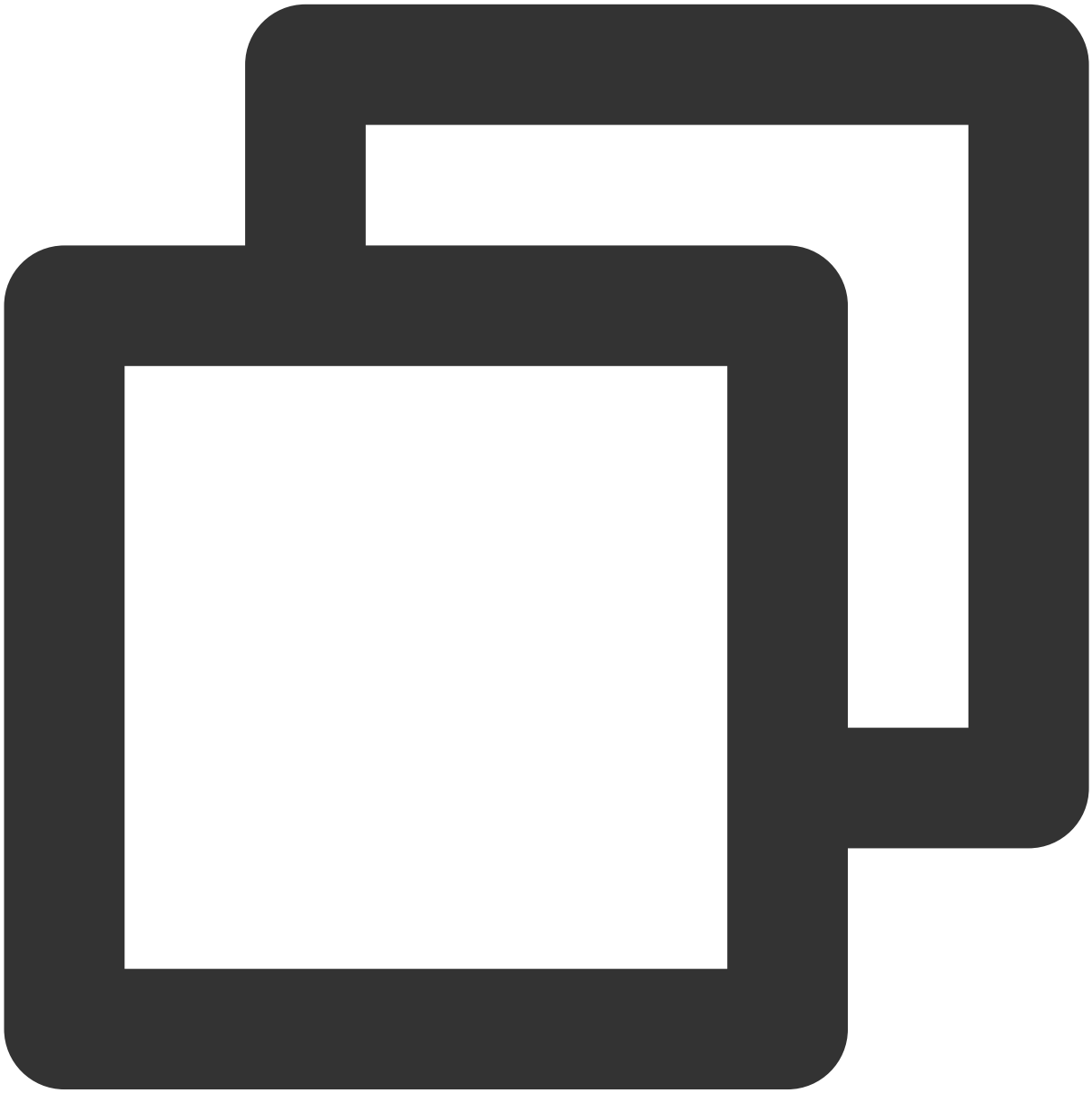
使用方法

首先，安装tencent_cloud_chat_text_translate插件：



```
flutter pub add tencent_cloud_chat_text_translate
```

要启用插件，在 `initUIKit` 中的 `plugins` 列表添加以下代码：



```
TencentCloudChatPluginItem(
  name: "textTranslate",
  pluginInstance: TencentCloudChatTextTranslate(),
),
```

联系我们

如果您对本功能有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

语音转文字
Android & iOS

最近更新时间：2024-06-14 15:23:10

功能描述

语音转文字功能：当您进入了聊天界面后，可以手动长按消息列表中的语音消息气泡，在出现的菜单中，单击转文字。

注意

语音转文字功能由插件提供，使用需集成 `TUIVoiceToTextPlugin` ，7.5 及以上版本支持。

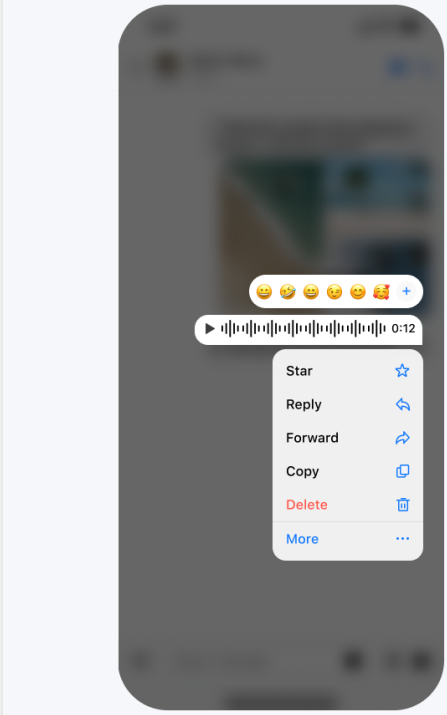
语音转文字是增值付费功能。当前此功能处于内测阶段，您可通过Telegram 技术交流群联系我们为您开通体验完整功能。

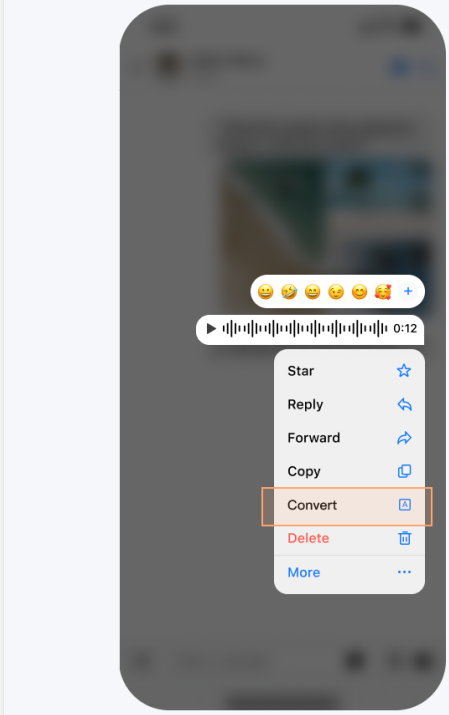
效果展示

集成语音转文字插件前后效果图如下所示：

简约版

未集成转文字插件，不显示转文字按钮	集成了转文字插件，显示转文字按钮





功能说明

集成插件

从 7.5 版本开始，语音转文字功能由插件 `TUIVoiceToTextPlugin` 提供。
如果您不需要语音转文字功能，不集成该插件即可。长按语音消息时，不会显示转文字按钮。
如果您需要语音转文字功能，需集成 `TUIChat` 和 `TUIVoiceToTextPlugin` 。集成方法请参见《集成 TUIKit》（Android/iOS）。集成后不需要进行任何设置，长按语音消息时，自动显示转文字按钮。

注意：

`TUIVoiceToTextPlugin` 依赖于 `TUIChat` ，不能单独集成。
仅支持语音消息，文本、图片、视频、文件、表情、自定义消息等不支持转文字。
目前已经支持的源语言有中文、粤语、英文和日文，具体请参见 《语言支持》。

联系我们

如果您对本功能有疑问，欢迎加入Telegram 技术交流群，您将获得可靠的技术支持。

Web & H5 & Uniapp（Vue）

最近更新时间：2024-06-13 11:53:36

功能描述

TUIKit 从 v2.1.2 版本之后开始提供**语音转文字功能**，TUIChat 组件中的消息工具支持对语音消息转换成文字，目前可以识别普通话、粤语、英语、日语并转换为对应的文字。
语音转文字后，转换的结果会在本次登录中持续保留，切换会话后转换结果依然保存。

注意：

“语音转文字”功能处于限免阶段，您可通过 Telegram 技术交流群组 联系我们为您开通体验完整功能。

版权所有：腾讯云计算（北京）有限责任公司

功能介绍

设置语音转文字的源语言

语音转文字的源语言默认情况下，能够自动识别普通话、粤语、英语，因此在简单场景下用户不需要手动设置源语言。当然，为了保证识别的精确性，也可以手动设置识别的源语言，目前支持以下四个选项：

zh (cmn-Hans-CN)，中文通用。

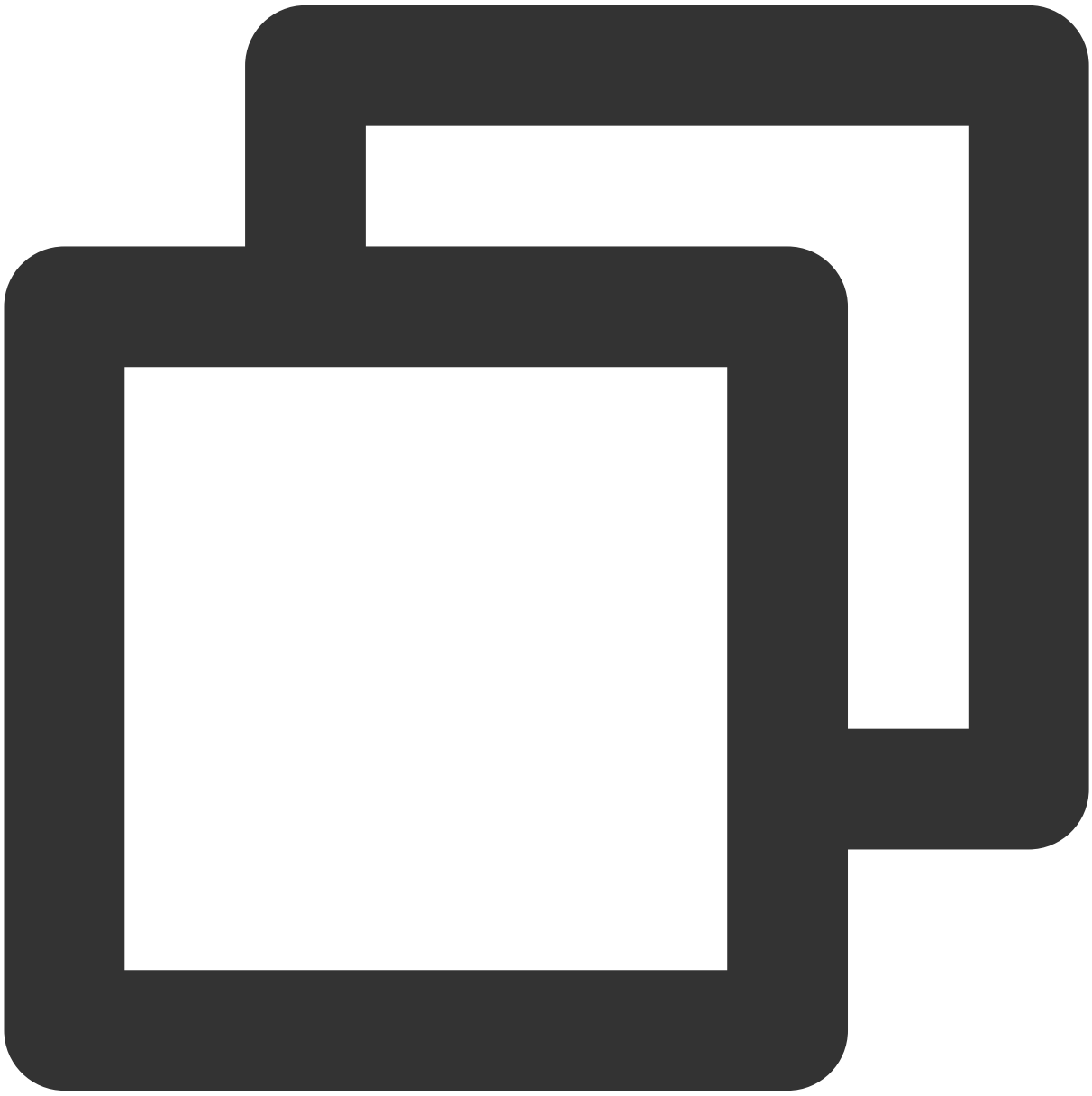
en-US，英语。

yue-Hant-HK，粤语。

ja-JP，日语。

版权所有：腾讯云计算（北京）有限责任公司

具体修改方法是在文件 `TUIKit/components/TUIChat/utils/convertVoiceToText.ts` 中，修改 `TUIChatService.convertVoiceToText` 方法调用的入参（以日语为例），添加 `language` 属性：



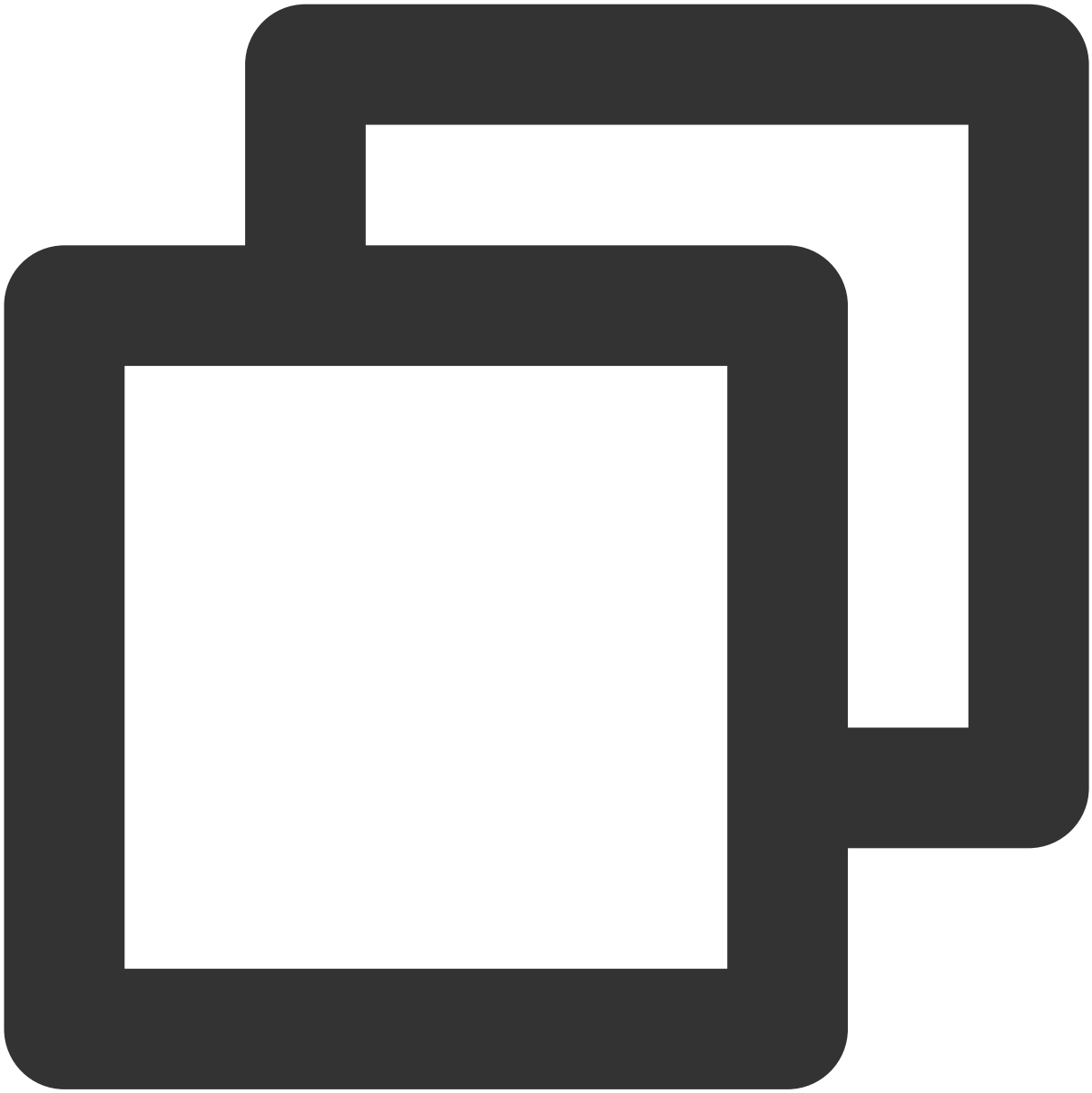
```
const response: IChatResponese<{ result: string }> = await TUIChatService.convertVo
  message: msg,
  language: 'ja-JP', // add this new line
});
```

扩展资料

以下内容仅为辅助阅读资料，语音转文字以及相关功能已在 TUIKit 2.1.1 之后实现，不需要用户手动实现。

查询当前套餐是否支持功能

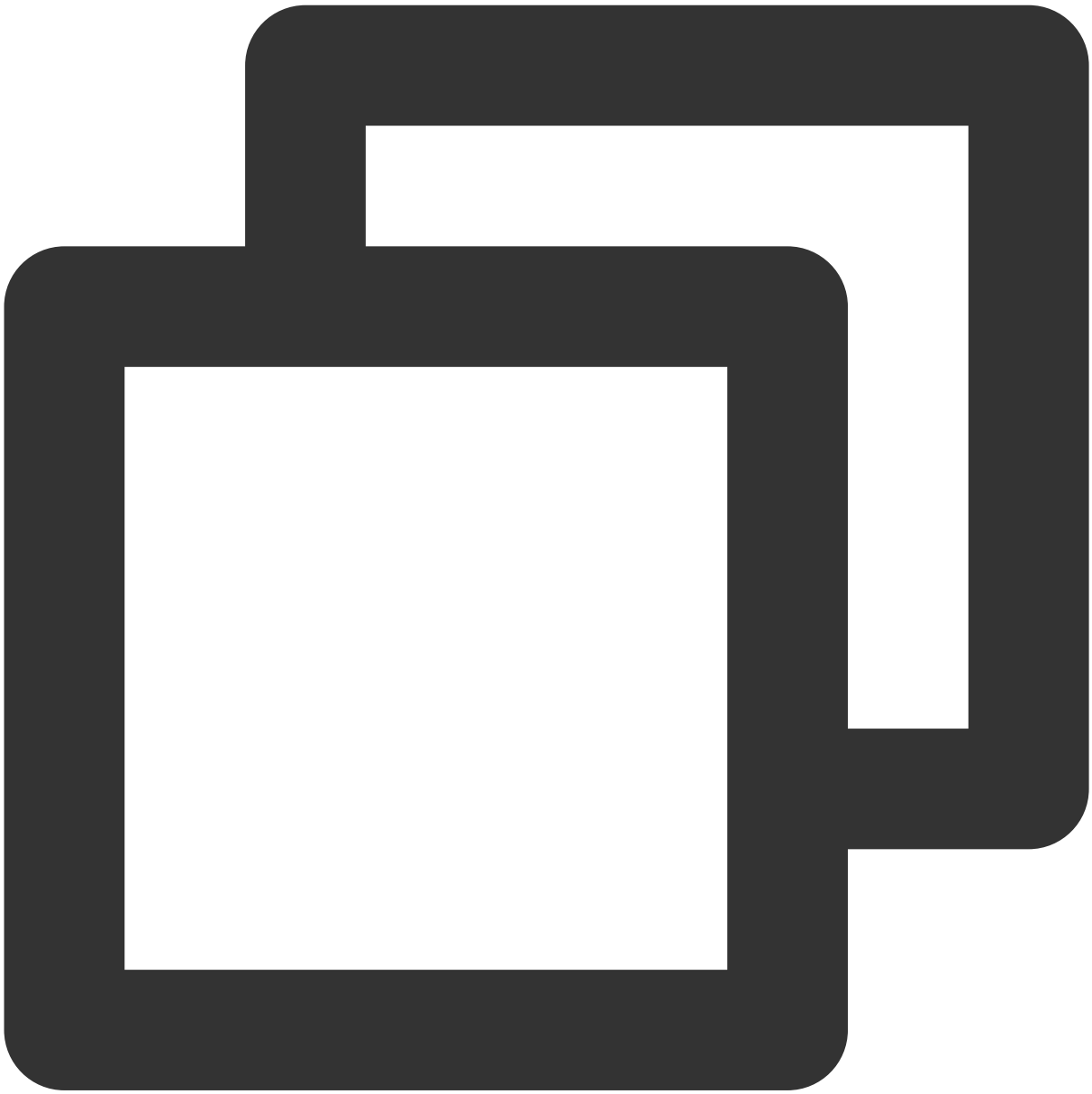
通过 `TUIStore` 查询 `StoreName.APP` 中 `enabledVoiceToText` 这个 key，得到的布尔值即是开通结果。



```
const enable = TUIStore.getData(StoreName.APP, 'enabledVoiceToText');
if (enable) {
  // ...
}
```

移除语音转文字功能

在文件 `TUIKit/components/TUIChat/message-list/message-tool/index.vue` 中，删除 `actionItems` 中 key 为 `convert` 的对象即可。



```
{
    key: 'convert',
    text: TUITranslateService.t('TUIChat.转文字'),
    visible: false,
    iconUrl: convertText,
    renderCondition() {},
    clickEvent: convertVoiceToText,
}
```

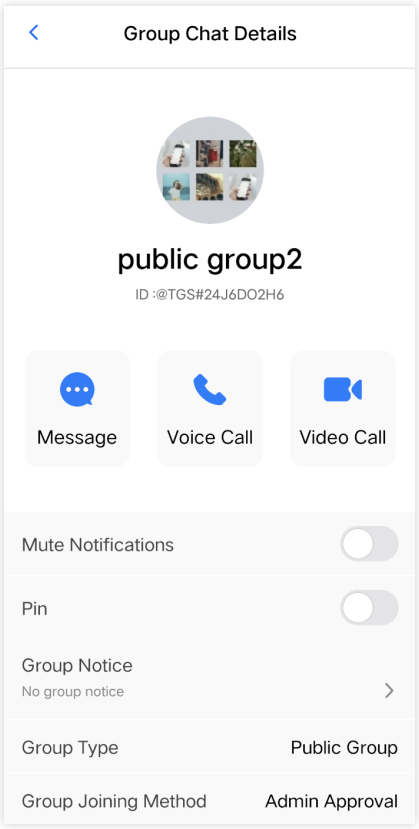
国际化界面语言

Android

最近更新时间：2024-05-20 15:02:11

功能描述

Android 端 `TUIKit` 默认自带 **英语、简体中文** 和 **阿拉伯语** 语言包，作为界面展示语言。
根据此文档指引，您可以使用默认语言包，也可自定义语言翻译表述和增加其他语言包。
说明：
TUIKit 从 7.5.4852 版本开始，新增 RTL 语言（文字方向从右到左的语言，比如阿拉伯语、希伯来语等）支持，当应用内语言为 RTL 语言时，TUIKit 会自动切换到 RTL 样式；同时内置语言新增了阿拉伯语。

英文	阿拉伯语	简体中文
		

使用自带语言

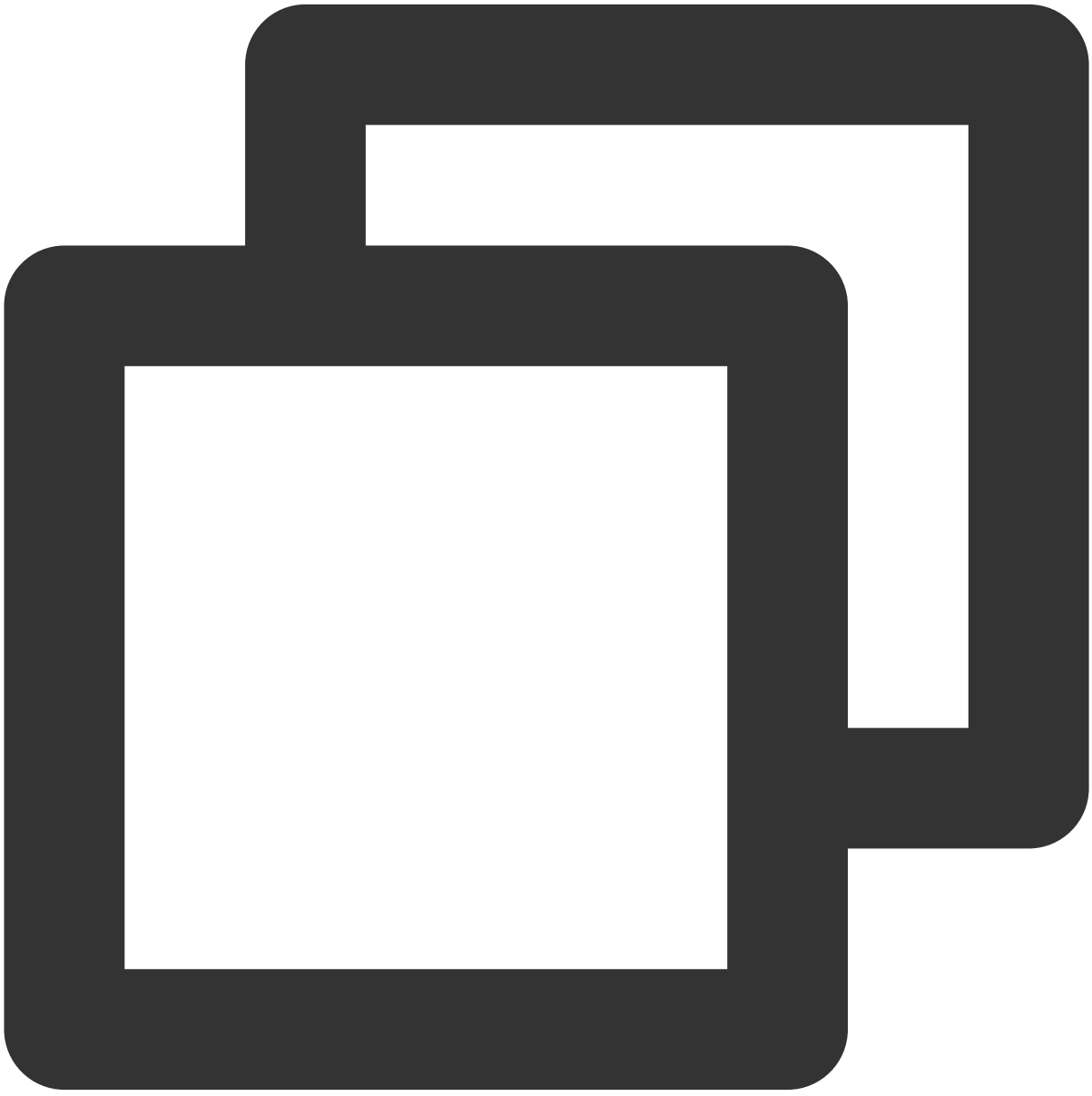
如果您的 App 需要的语言仅包括 **英语**、 **阿拉伯语** 和 **简体中文**，请参考本部分。

跟随系统语言

直接使用 `TUIKit` 即可，无需额外步骤。组件内部语言会跟随系统语言。

指定显示的语言

如果您需要指定 TUIKit 界面的语言，需要在 Appliction 初始化时调用以下代码进行设置，例如设置为中文：



```
public class MyApplication extends Application {
    @Override
```

```
    protected void onCreate() {
        super.onCreate();
        TUIThemeManager.getInstance().changeLanguage(this, TUIThemeManager.LANGUAGE
    }

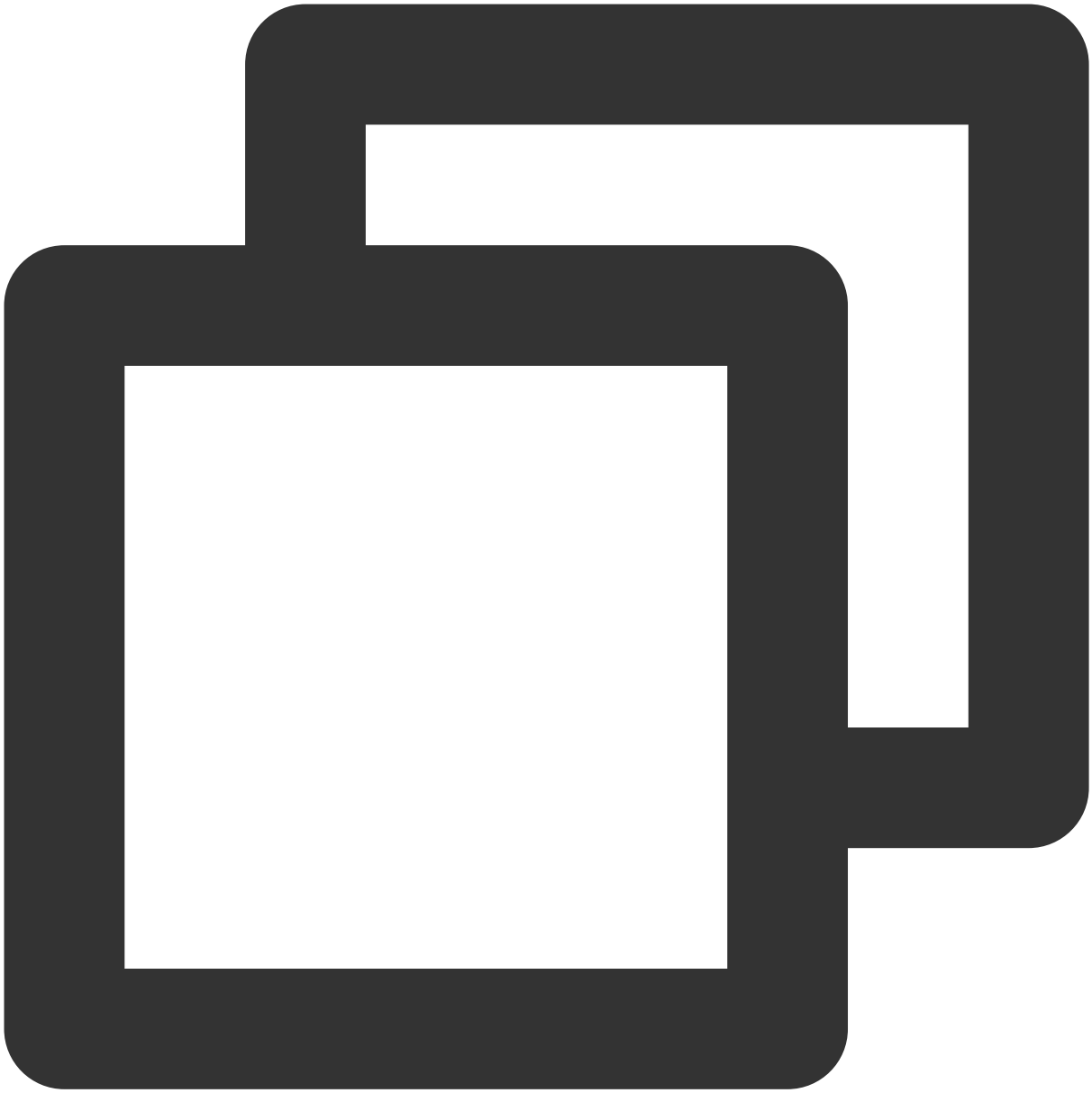
    /**
    * The available language options are enumerated as follows:
    * TUIThemeManager.LANGUAGE_EN         ---- English
    * TUIThemeManager.LANGUAGE_ZH_CN      ---- Simplified Chinese
    * TUIThemeManager.LANGUAGE_AR         ---- Arabic
    */
}
```

注意

调用 `changeLanguage` 方法并不会自动刷新 UI，需要获取字符串之后重新设置到控件上才能生效。

动态修改语言

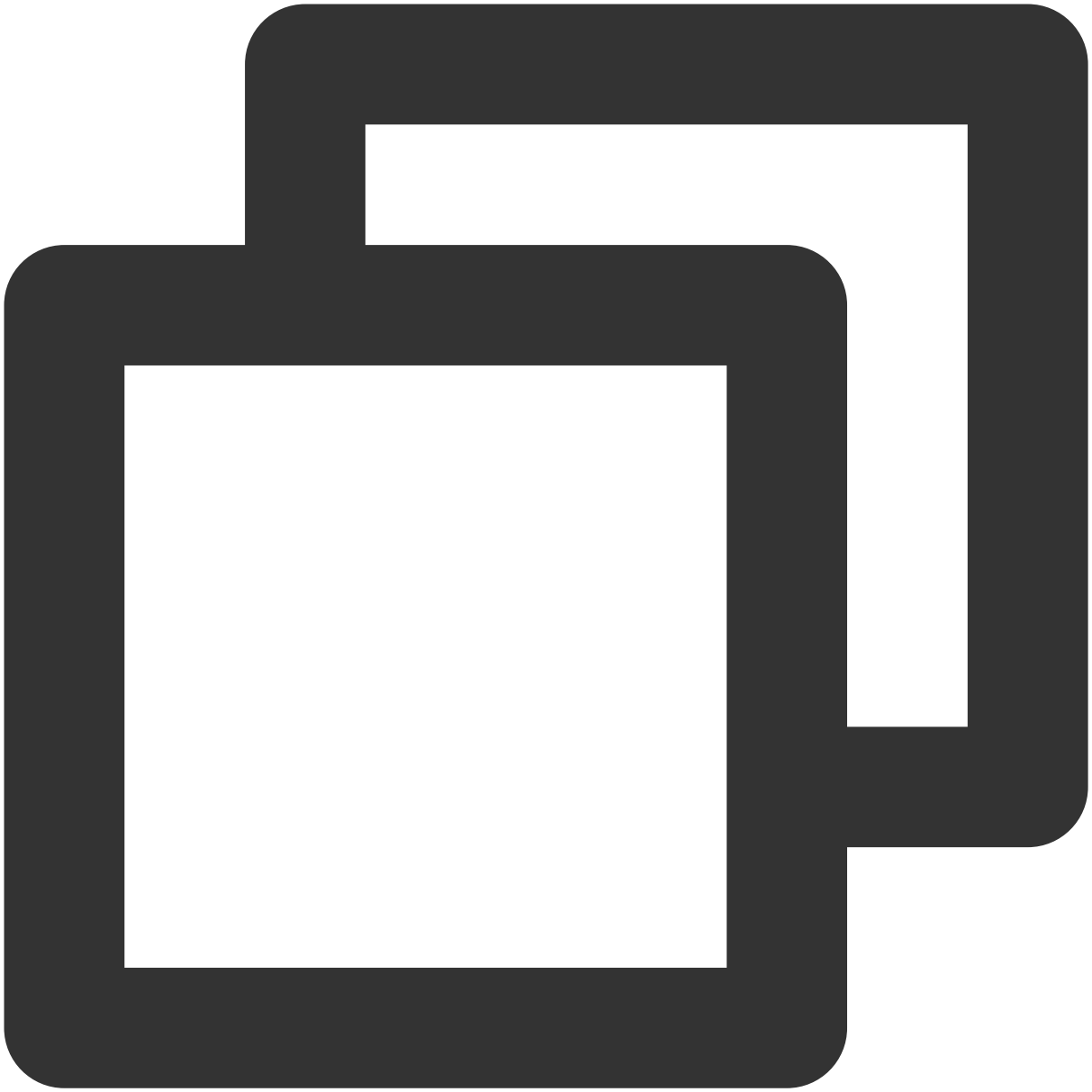
您可以参考 `TUIKitDemo` 的 LanguageSelectActivity.java 文件中的代码。也可以使用如下方法切换语言，例如切换为中文：



```
TUIThemeManager.getInstance().changeLanguage(context, TUIThemeManager.LANGUAGE_ZH_C
System.exit(0);
Intent intent = context.getPackageManager().getLaunchIntentForPackage(context.getPa
context.startActivity(intent);
```

使用 WebView 之后发现语言切换失败处理方法

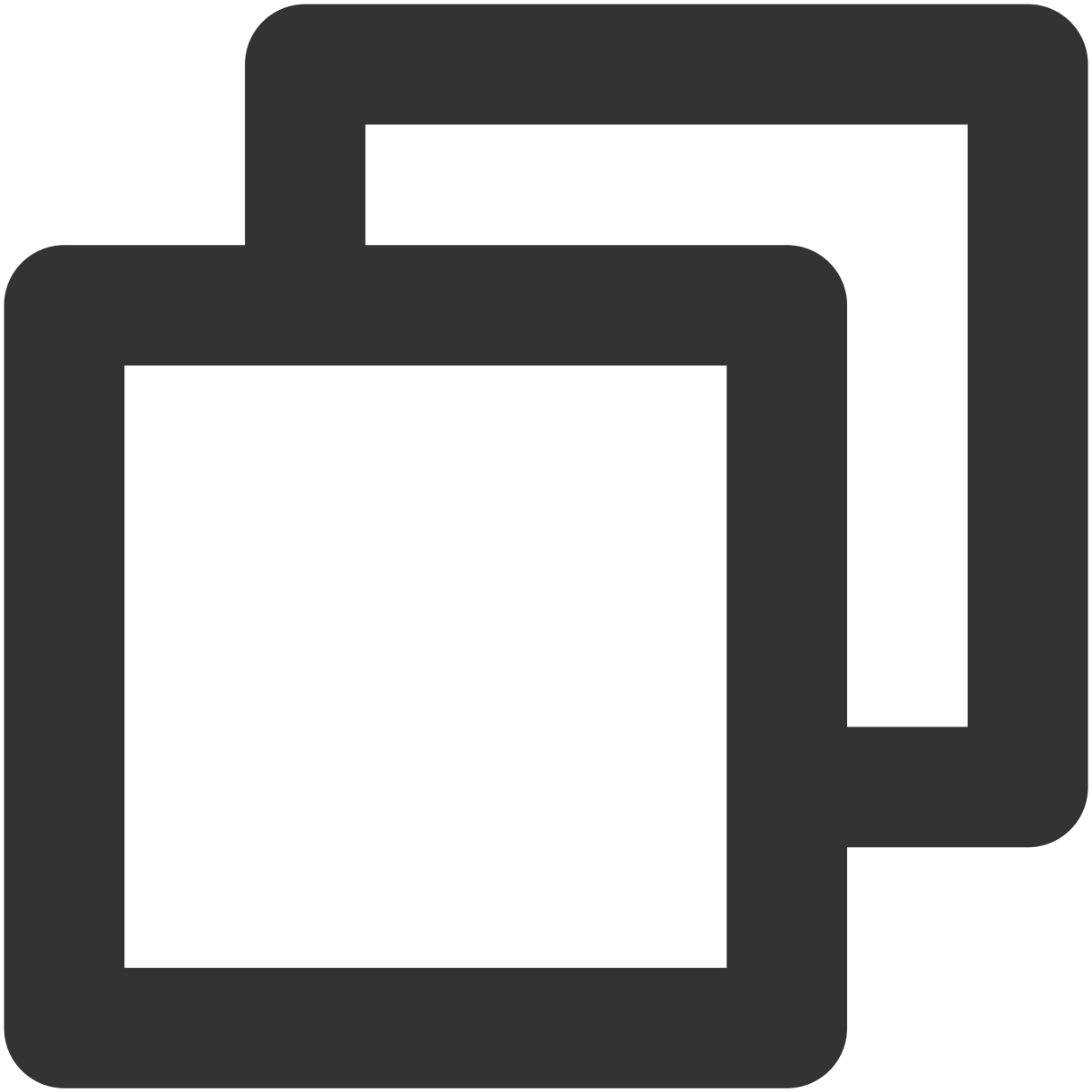
使用 WebView 之后导致语言切换失败是 Android 7 及以后版本的 bug，原因是 WebView 初始化时会修改 App 的语言为手机系统语言。需要在 TUIThemeManager.java 的 `setThemeInternal` 方法中调用以下代码解决此问题：



```
setWebViewLanguage(appContext);
```

添加之后：

版权所有：腾讯云计算（北京）有限责任公司



```
private void setThemeInternal(Context context) {
    if (context == null) {
        return;
    }

    Context appContext = context.getApplicationContext();
    if (!isInit) {
        isInit = true;
        if (appContext instanceof Application) {
            ((Application) appContext).registerActivityLifecycleCallbacks(new Theme
        }
```

```
        /**
        * add code here  begin
        */
        setWebViewLanguage(appContext);
        /**
        * add code here  end
        */
        Locale defaultLocale = getLocale(appContext);
        SPUtils spUtils = SPUtils.getInstance(SP_THEME_AND_LANGUAGE_NAME);
        currentLanguage = spUtils.getString(SP_KEY_LANGUAGE, defaultLocale.getLangu
        currentThemeID = spUtils.getInt(SP_KEY_THEME, THEME_LIGHT);

        // The language only needs to be initialized once
        applyLanguage(appContext);
    }
    // The theme needs to be updated multiple times
    applyTheme(appContext);
}
```

使用更多语言/自定义翻译表述

如果您的 App 需要支持更多语言，或更改部分词条的翻译，请参考本部分。
本章节以 `TUIGroup` 组件添加韩语语言包为例，讲解新增语言包和自定义翻译的流程。

新增语言资源文件

在 Android Studio 中的 `TUIGroup` 组件目录下，右键菜单中新增 Android Resource File：
输入文件名 `strings`，由 `Locale` 维度创建资源目录：
语言选择为韩语（"ko: Korean"），地区选为"KR: South Korea"，点击确定，这样就创建好了韩语资源文件 `values-ko-rKR/strings.xml`。

个性化自定义翻译

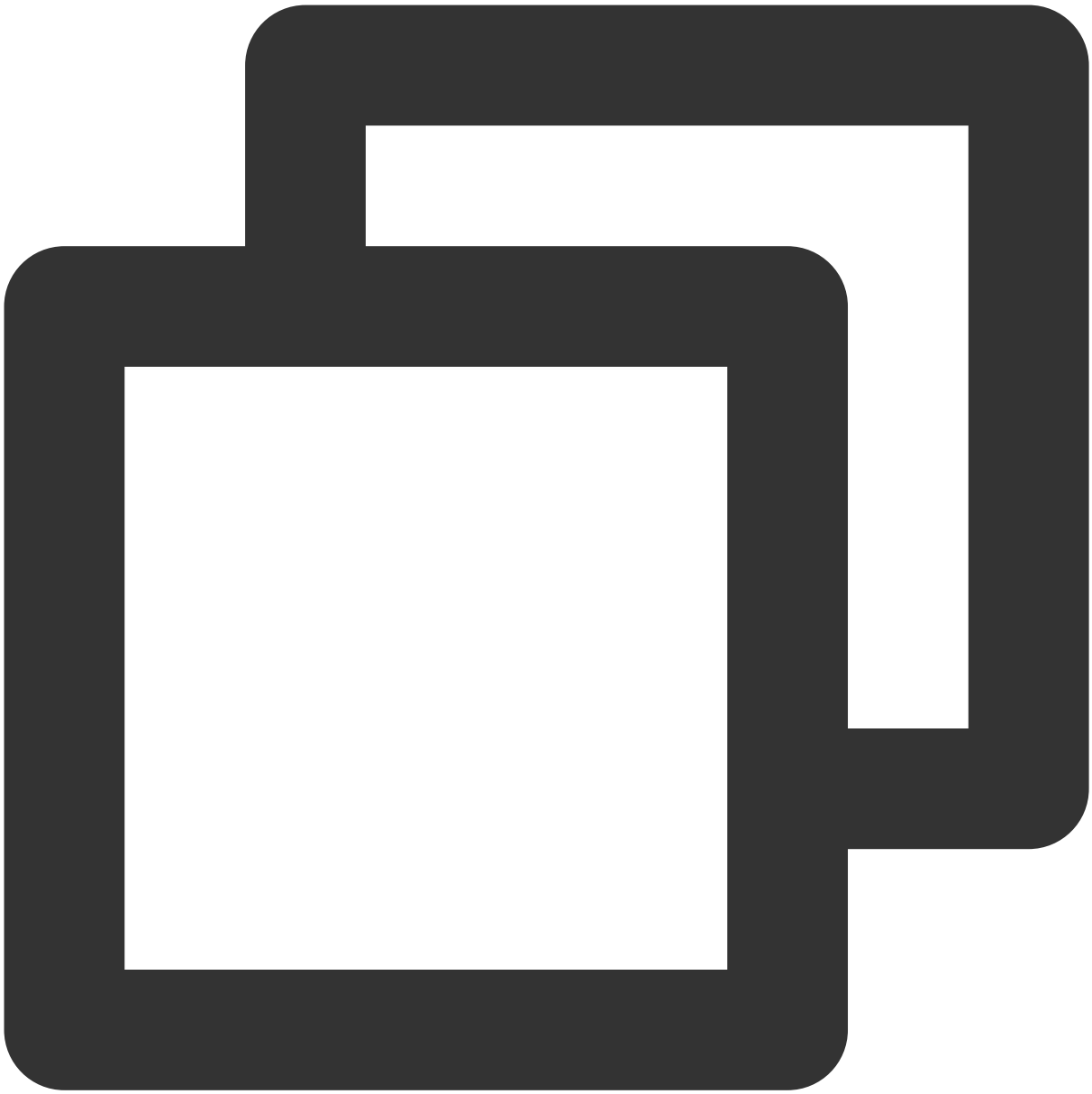
上一步已经创建好了韩语资源文件 `values-ko-rKR/strings.xml`，现在把 `values/strings.xml` 文件中的内容复制到 `values-ko-rKR/strings.xml`，用韩语替换对应的英文，如图所示：
不同语言资源文件中语言的 `name` 是相同的，具体内容可以自定义翻译。

跟随系统语言

直接使用 `TUIKit` 即可，将手机默认语言设置为韩语后启动 App，App 语言可以自动显示为韩语。

指定显示的语言

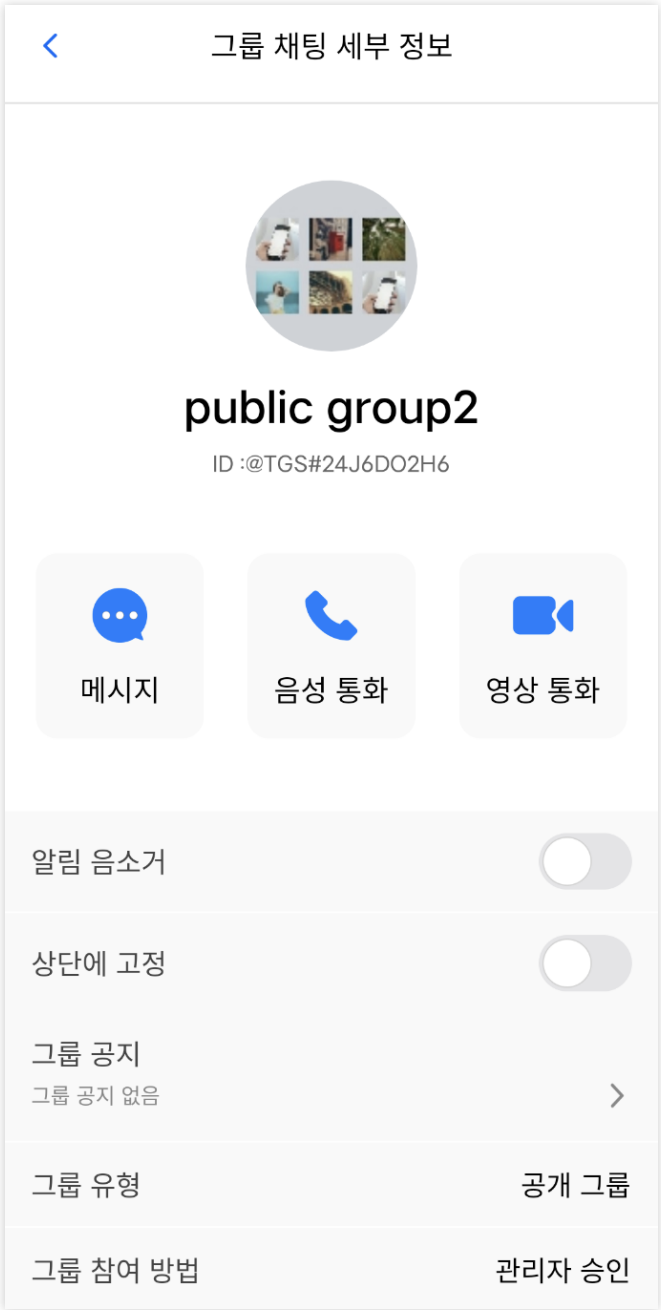
如果您需要指定 `TUIKit` 界面的语言为韩语，应该先在 `Appliction` 初始化时向语言管理器中添加韩语，然后再设置 `TUIKit` 界面的语言为韩语：



```
public class MyApplication extends Application {
    @Override
    protected void onCreate() {
        super.onCreate();
        // Add Korean
        TUIThemeManager.addLanguage("ko-rKR", Locale.KOREA);
        // Change the application language to Korean.
        TUIThemeManager.getInstance().changeLanguage(this, "ko-rKR");
```

```
        }
    }
```

效果如图所示：

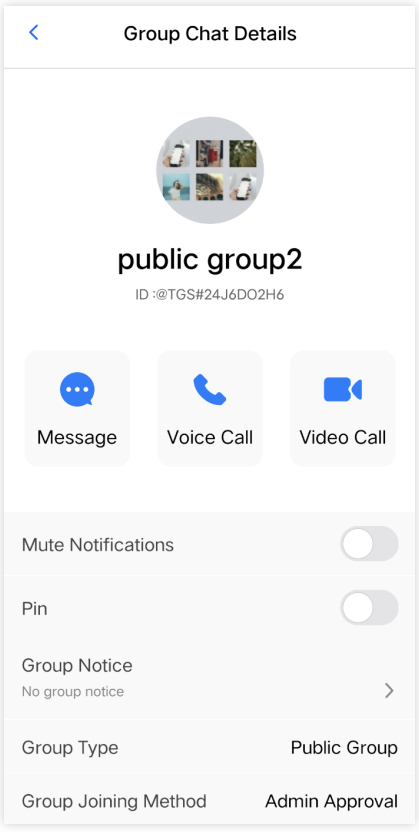

그룹 채팅 세부 정보
public group2
ID :@TGS#24J6DO2H6
메시지
음성 통화
영상 통화
알림 음소거
상단에 고정
그룹 공지
그룹 공지 없음
그룹 유형
공개 그룹
그룹 참여 방법
관리자 승인

iOS

最近更新时间：2024-05-20 17:08:56

功能描述

iOS 端 `TUIKit` 默认自带 **英文、简体中文、** 和 **阿拉伯语** 语言包，作为界面展示语言。
根据此文档指引，您可以使用默认语言包，也可自定义语言翻译表述和增加其他语言包。
说明：
TUIKit 从 7.5.4852 版本开始，新增 RTL 语言（文字方向从右到左的语言，比如阿拉伯语、希伯来语等）支持，同时内置语言新增了阿拉伯语。

英文	阿拉伯语	简体中文
		

使用自带语言

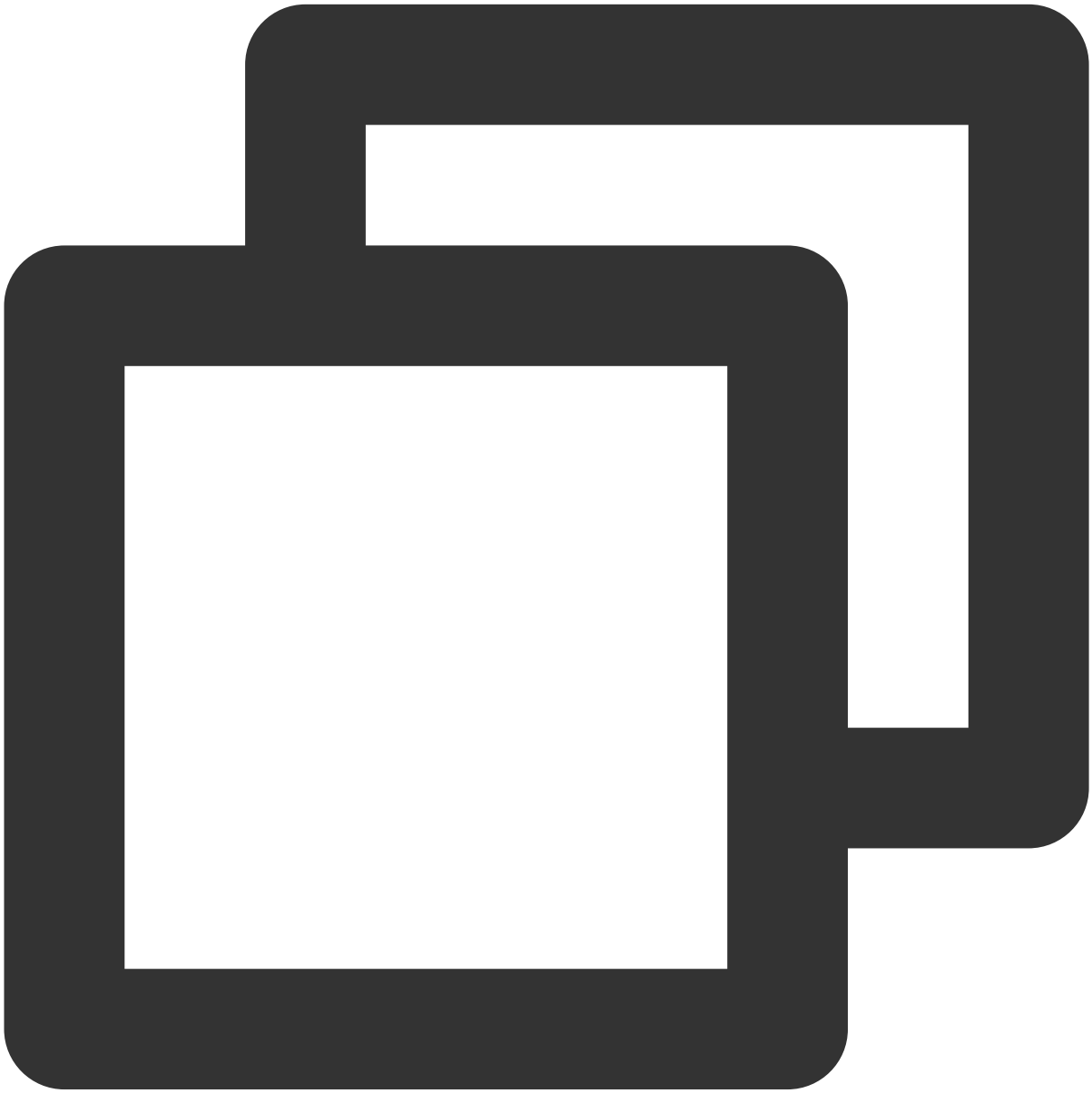
如果您的 App 需要的语言仅包括 **英语、简体中文、** 和 **阿拉伯语**，请参考本部分。

跟随系统语言

直接使用 `TUIKit` 即可，无需额外步骤。组件内部语言会跟随系统语言。

指定显示的语言

如果您需要指定 `TUIKit` 界面语言，请在 `[TUIGlobalization setCustomLanguage:@""]` 中传入需要的语言，指定语言后，组件内部不再跟随系统语言。
语言可选项，取值为：



```
@"zh-Hans" ,//simple Chinese
@"en", // English
```

```
  @"ar", // Arabic
```

说明：

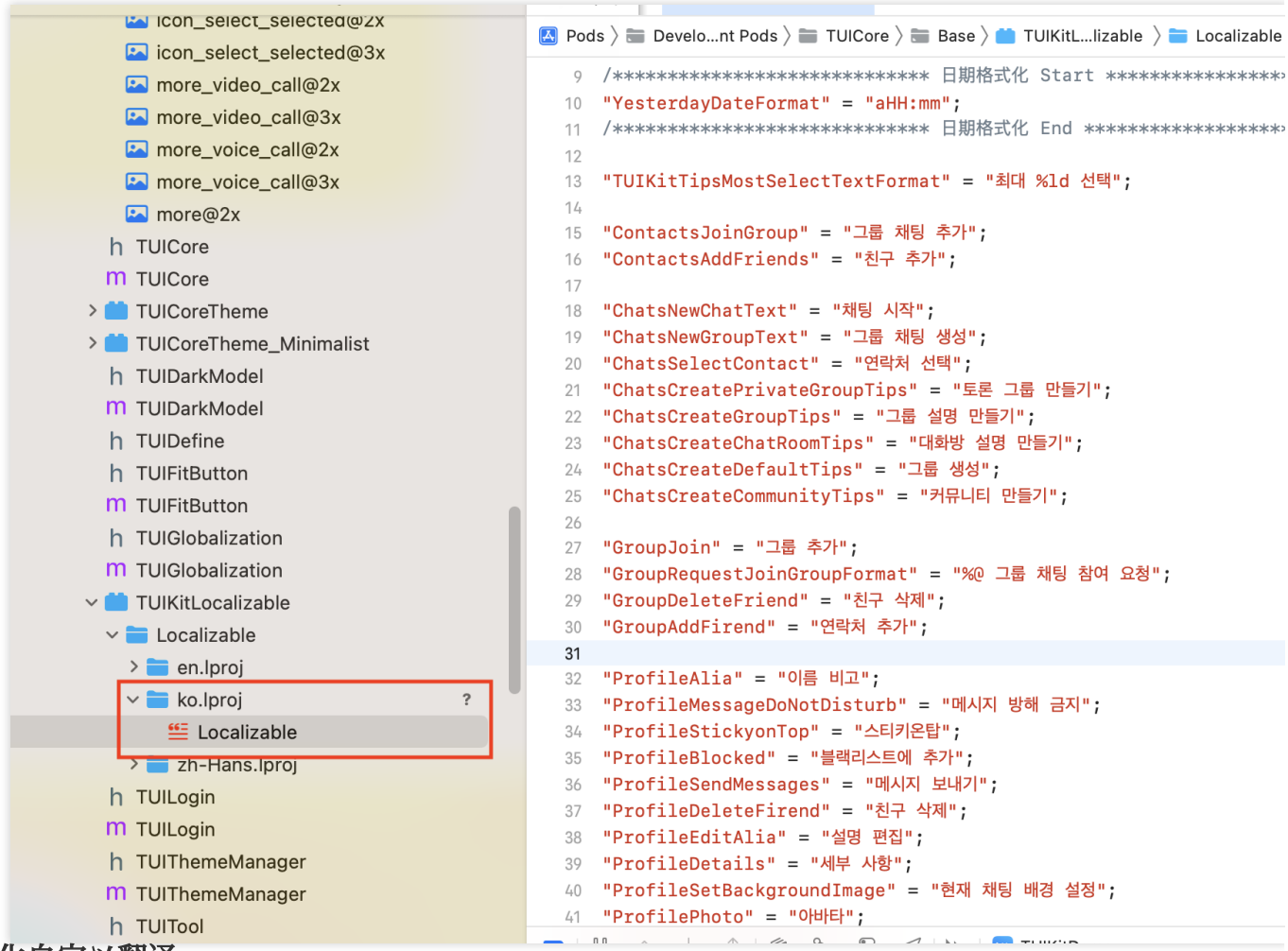
语言代码清单见 附录。

使用更多语言/自定义翻译表述

如果您的 App 需要支持更多语言，或更改部分词条的翻译，请参考本部分。
本章节以添加韩语语言包为例，讲解新增语言包和自定义翻译的流程。

新增语言资源文件

我们自带的所有语言包，以String文件模板的形式，存储于您项目里Pods中 `TUICore` 组件的 `TUIKitLocalizable/Localizable/` 路径。
请新建目录并命名为 `{语言编码}.lproj` ，在此目录下新增 `Localizable.strings` 文件，其中， `${语言编码}` 需要替换为 ISO 639-1 语言代码。（可以直接复制您熟悉的语言文件，如zh-Hans.lproj，并直接修改目录名称）
如果您需要兼容支持多个新语言，复制多份，并准确指定每一份的语言编码即可。
以韩语为例，新增 `ko.lproj/Localizable.strings` 的语言资源文件：



个性化自定义翻译

上一步已经创建好了韩语资源文件 `ko.lproj/Localizable.strings`，不同语言资源文件中语言的 `key` 是相同的，具体内容可以自定义翻译。

跟随系统语言

如果是简体中文、繁体中文、英文、韩语、俄语、乌克兰语，添加完lproj资源包后直接使用TUIKit即可，无需额外步骤。

如果是其他语种，则需要在 `TUIGlobalization.m` 中的 `+ (NSString *)tk_localizableLanguageKey` 中新增。

新增的语言词条包命名需要 符合标准，组件内部会跟随系统语言自适应。

说明

语言代码清单见 ISO 639-1 语言代码。

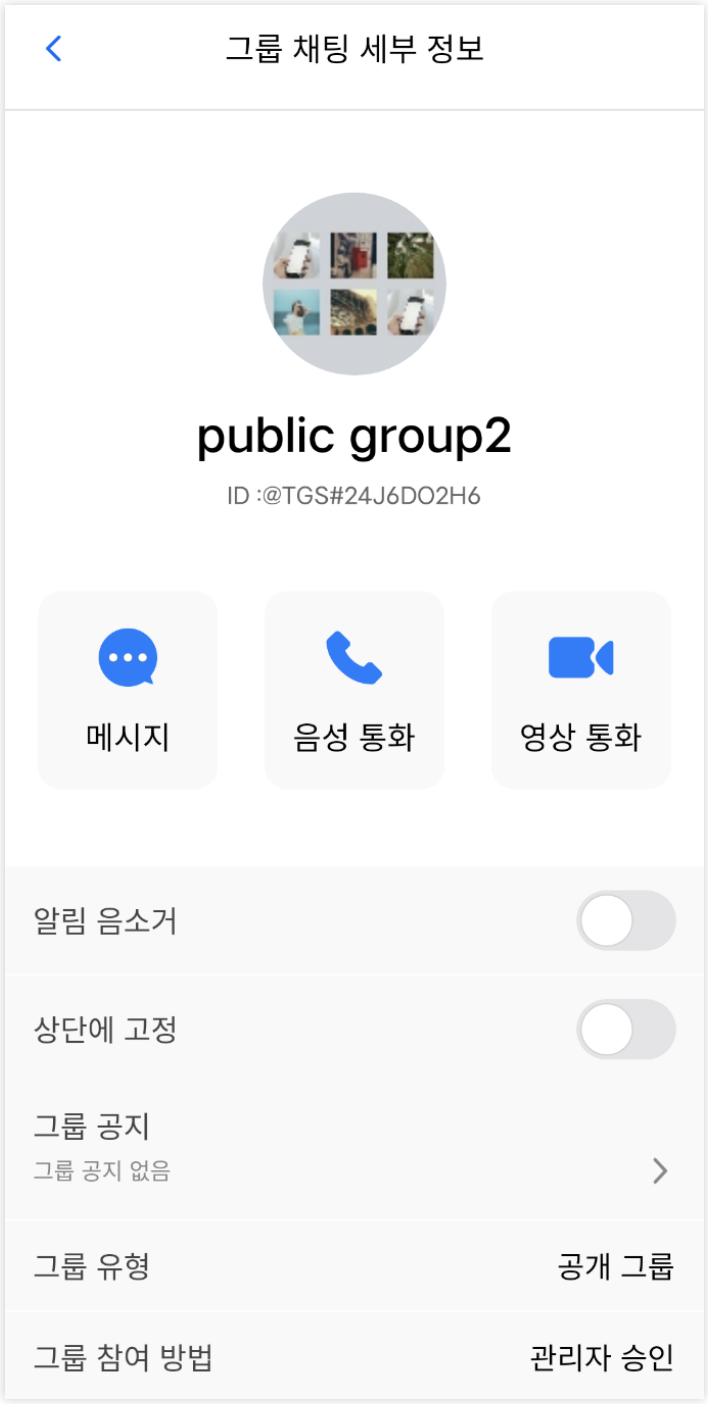
指定显示的语言

如果您需要指定 `TUIKit` 界面语言，请在 `[TUIGlobalization setCustomLanguage:@""]` 中传入需要的语言，指定语言后，组件内部不再跟随系统语言。

语言可选项，取值为 ISO 639-1 语言代码

以韩语为例如 `[TUIGlobalization setCustomLanguage:@"ko"];`

效果如图所示：



附录：语言代码表

语言	代码	语言	代码
阿拉伯语	ar	保加利亚语	bg

克罗地亚语	hr	捷克语	cs
丹麦语	da	德语	de
希腊语	el	英语	en
爱沙尼亚语	et	西班牙语	es
芬兰语	fi	法语	fr
爱尔兰语	ga	印地语	hi
匈牙利语	hu	希伯来语	he
意大利语	it	日语	ja
朝鲜语/韩语	ko	拉脱维亚语	lv
立陶宛语	lt	荷兰语	nl
挪威语	no	波兰语	pl
葡萄牙语	pt	瑞典语	sv
罗马尼亚语	ro	俄语	ru
塞尔维亚语	sr	斯洛伐克语	sk
斯洛文尼亚语	sl	泰语	th
土耳其语	tr	乌克兰语	uk
中文（简体）	zh-Hans	中文（繁体）	zh-Hant

完整版 请见此处。

版权所有：腾讯云计算（北京）有限责任公司

Web & H5 (Vue)

最近更新时间：2024-06-12 17:44:13

功能描述

说明：

高级国际化多语言能力在@tencentcloud/chat-uikit-vue v2.0.0 版本后有较大改进与变动。

本文档所示为新版本的用法。请**确保您项目依赖的 @tencentcloud/chat-uikit-vue ≥ v2.0.0**。

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

根据此文档指引，您可以使用默认语言包，也可使用自定义的高级国际化能力，包括新增语言、新增词条或修改现有词条翻译。

使用自带语言及词条库

如果您的 App，需要的语言仅包括**英语/简体中文**，且不需要新增词条或修改现有词条翻译，请参考本部分。

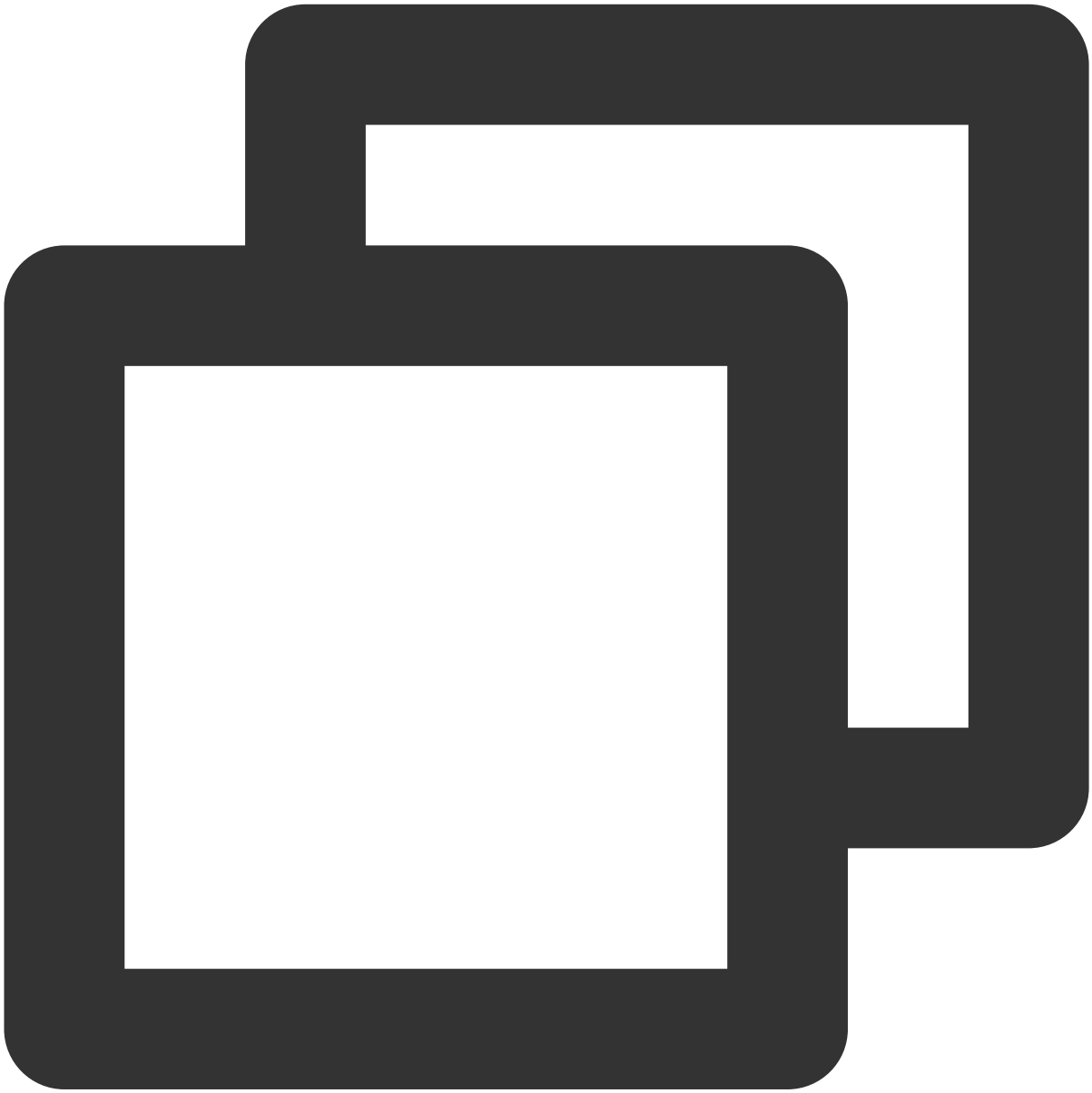
注意：

后续将开放更多语言支持，敬请期待！

版权所有：腾讯云计算（北京）有限责任公司

指定显示语言

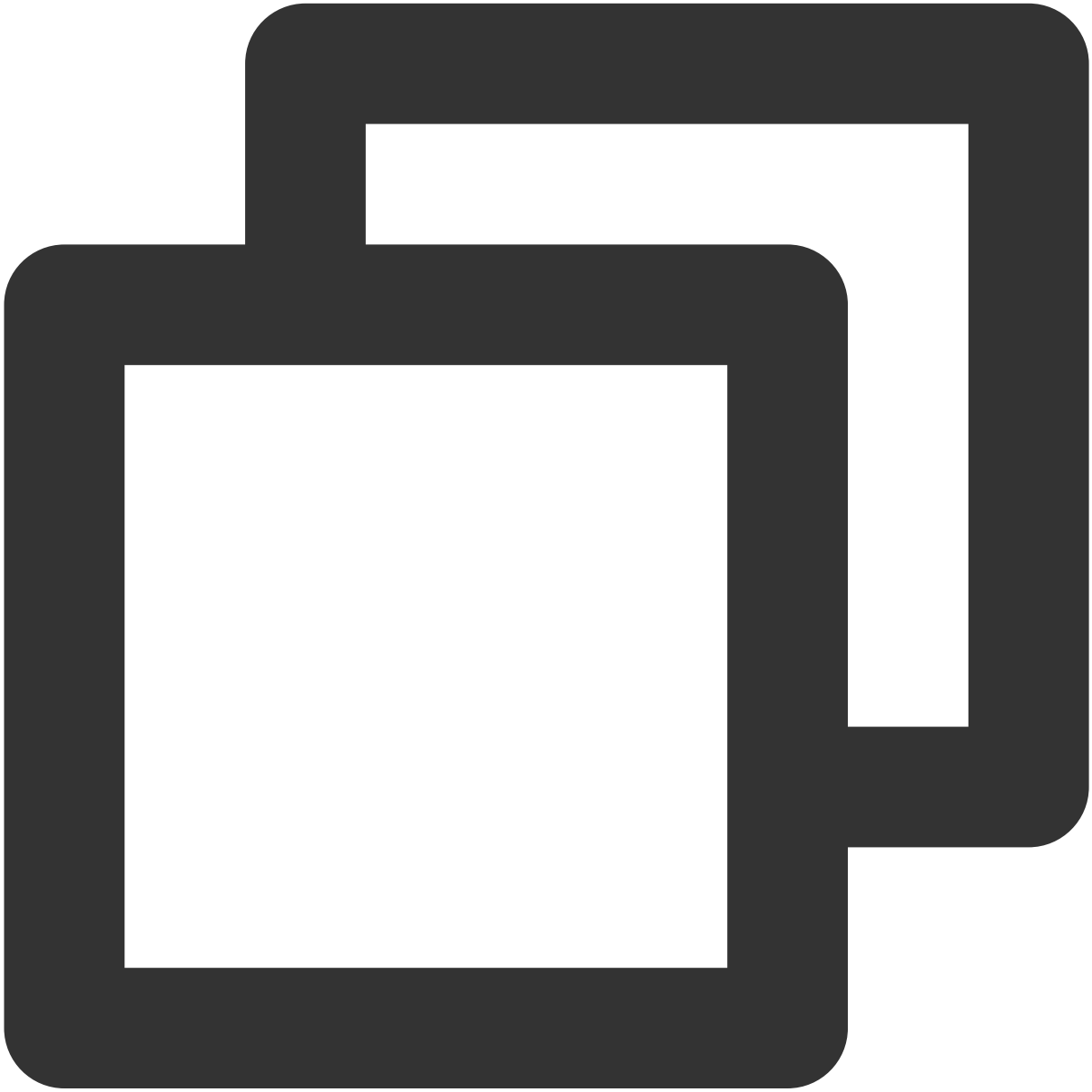
如果您需要指定 TUIKit 界面的语言，需要在 App 初始化时调用以下代码进行设置。



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

实时动态修改

当您采用 `TUITranslateService.changeLanguage` 进行语言切换时，您当前的语言并不会实时修改，需刷新后才能生效。
您可以通过切换 `页面/组件 key` 的方式，实现语言实时动态修改与展示。
比如，实时动态切换 `TUIConversation` 的语言：



```
<template>
  <div class="home">
    <div class="button-container">
      <div class="button" @click="changeLanguage('en')">English</div>
      <div class="button" @click="changeLanguage('zh')">简体中文</div>
```

```
    </div>
    <div class="conversation-container">
      <TUIConversation :key="locale" />
    </div>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
import { TUIConversation } from "./TUIKit";
const locale = ref("zh");
const changeLanguage = (language: string) => {
  TUITranslateService.changeLanguage(language).then(() => {
    locale.value = language;
  });
};
</script>
<style scoped lang="scss">
.home {
  width: 400px;
  .button-container {
    display: flex;
    flex-direction: column;
    .button {
      margin: 10px;
      background-color: #006eff;
      color: #ffffff;
      text-align: center;
      padding: 5px;
      border-radius: 30px;
      cursor: pointer;
    }
  }
}
</style>
```

自定义语言词条

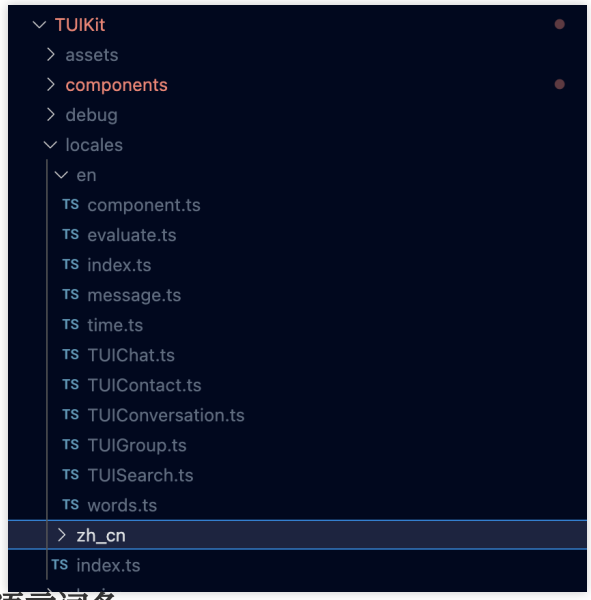
新增语言词条

如果您需要对现有的 **简体中文、英语** 语言包词条进行扩充或修改，可以在 `src/TUIKit/locales` 目录下进行新增或修改词条。

locales 词条包分为两部分，en 目录下为 英文词条， zh_cn 目录下为 简体中文词条。

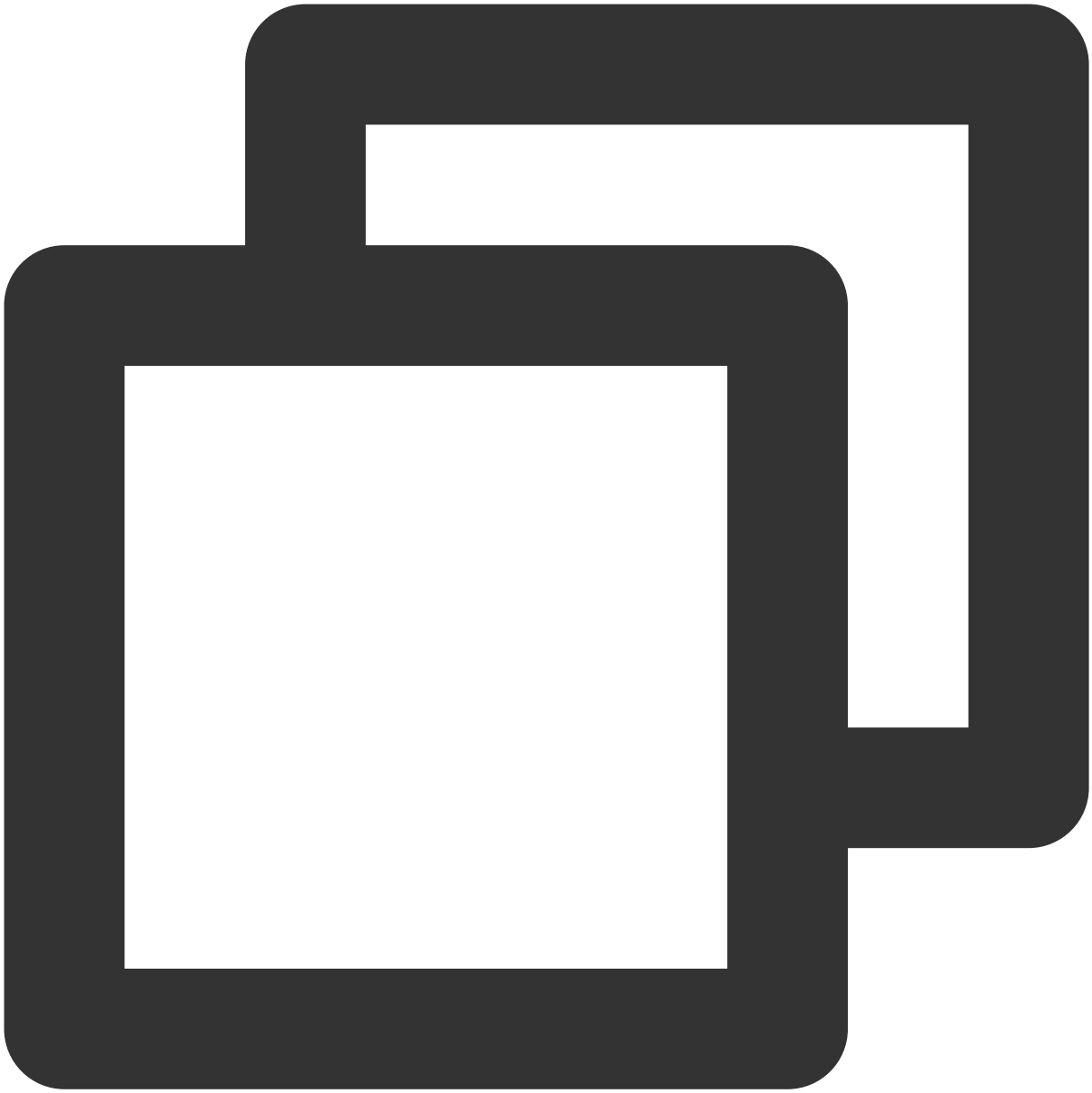
注意：

如果您同时需要使用简体中文、英语，新增或修改词条时**请务必在 en 和 zh_cn 两文件夹相同子目录下同步修改**。
locales 词条包目录结构如下图：



使用语言词条

以下使用 `TUITranslateService.t()` 进行词条翻译，更多接口详情请参考 TUITranslateService。



```
<template>
    <div>{{ TUITransalteService.t("TUIChat.${yourLocaleKey}")}}</div>
</template>
<script setup lang="ts">
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
</script>
```

交流与反馈

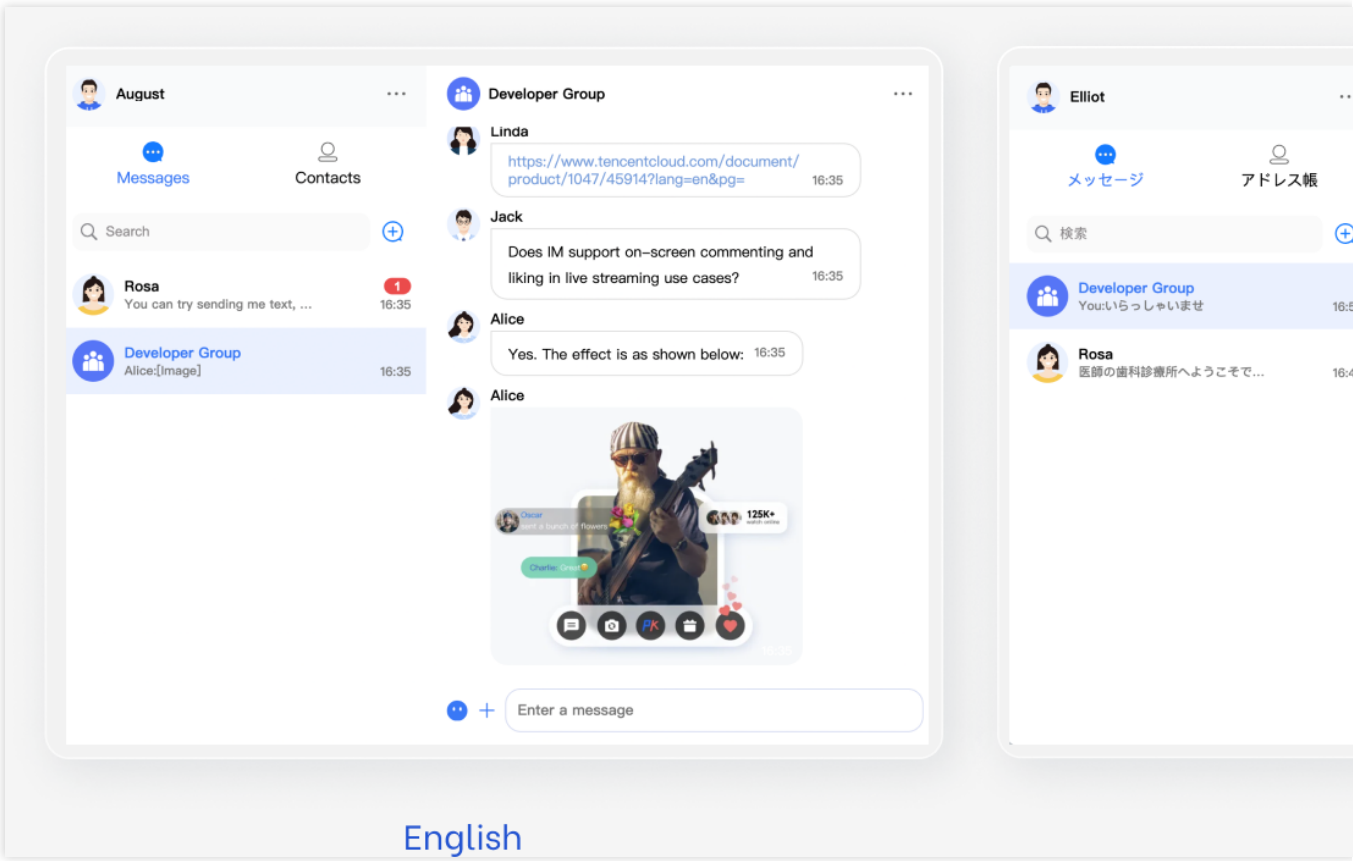
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

React

最近更新时间：2024-08-05 14:26:31

功能描述

`TUIKit` 默认自带 **英语、日语、韩语、中文** 语言包，作为界面展示语言。
根据此文档指引，您可以使用默认语言包，也可使用自定义的高级国际化能力，包括新增语言、新增词条或修改现有词条翻译。



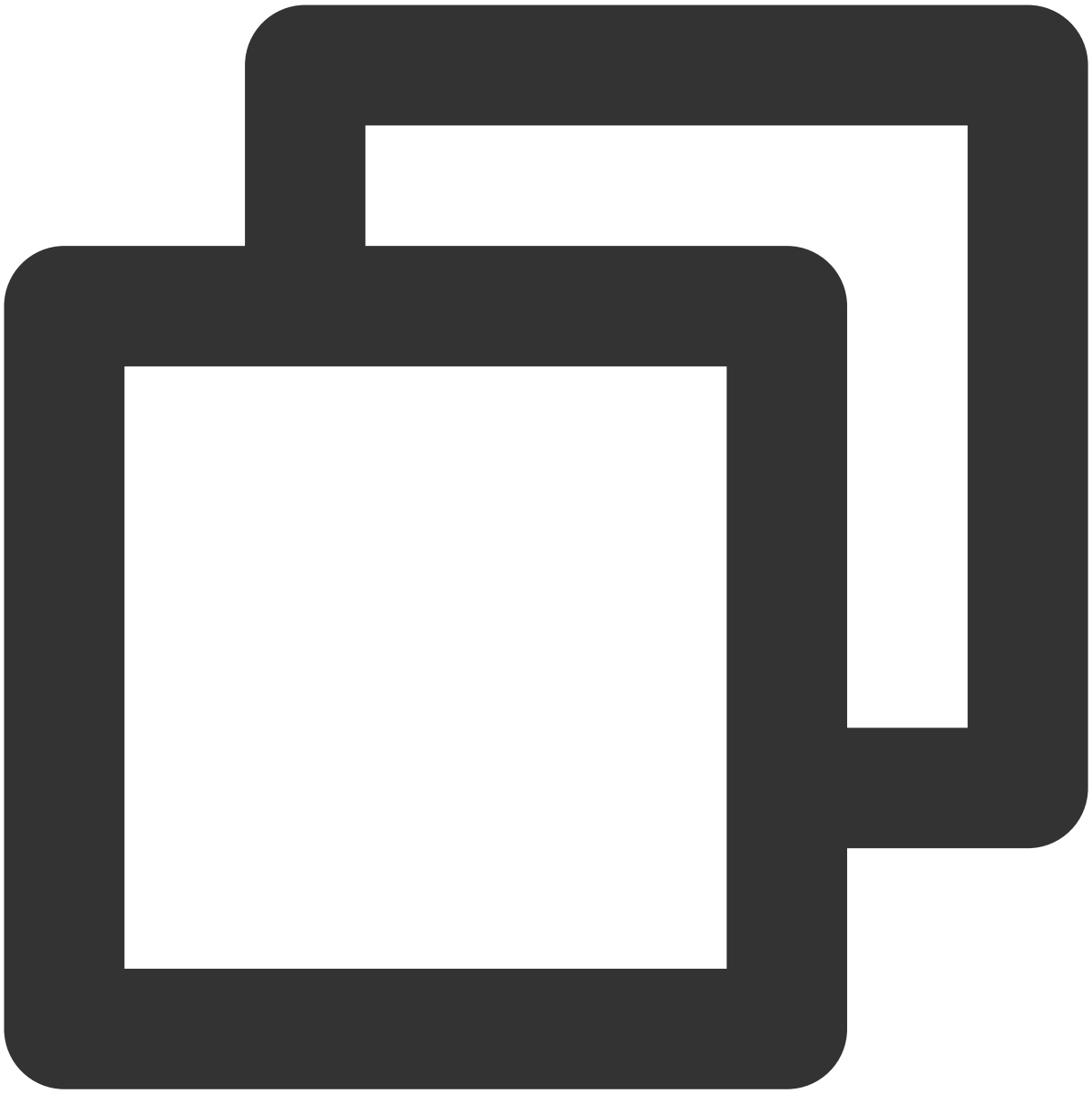
使用自带语言及词条库

如果您的 App，需要的语言仅包括**英语 / 日语 / 韩语 / 中文** ，且不需要新增词条或修改现有词条翻译，请参考本部分。

指定语言

版权所有：腾讯云计算（北京）有限责任公司

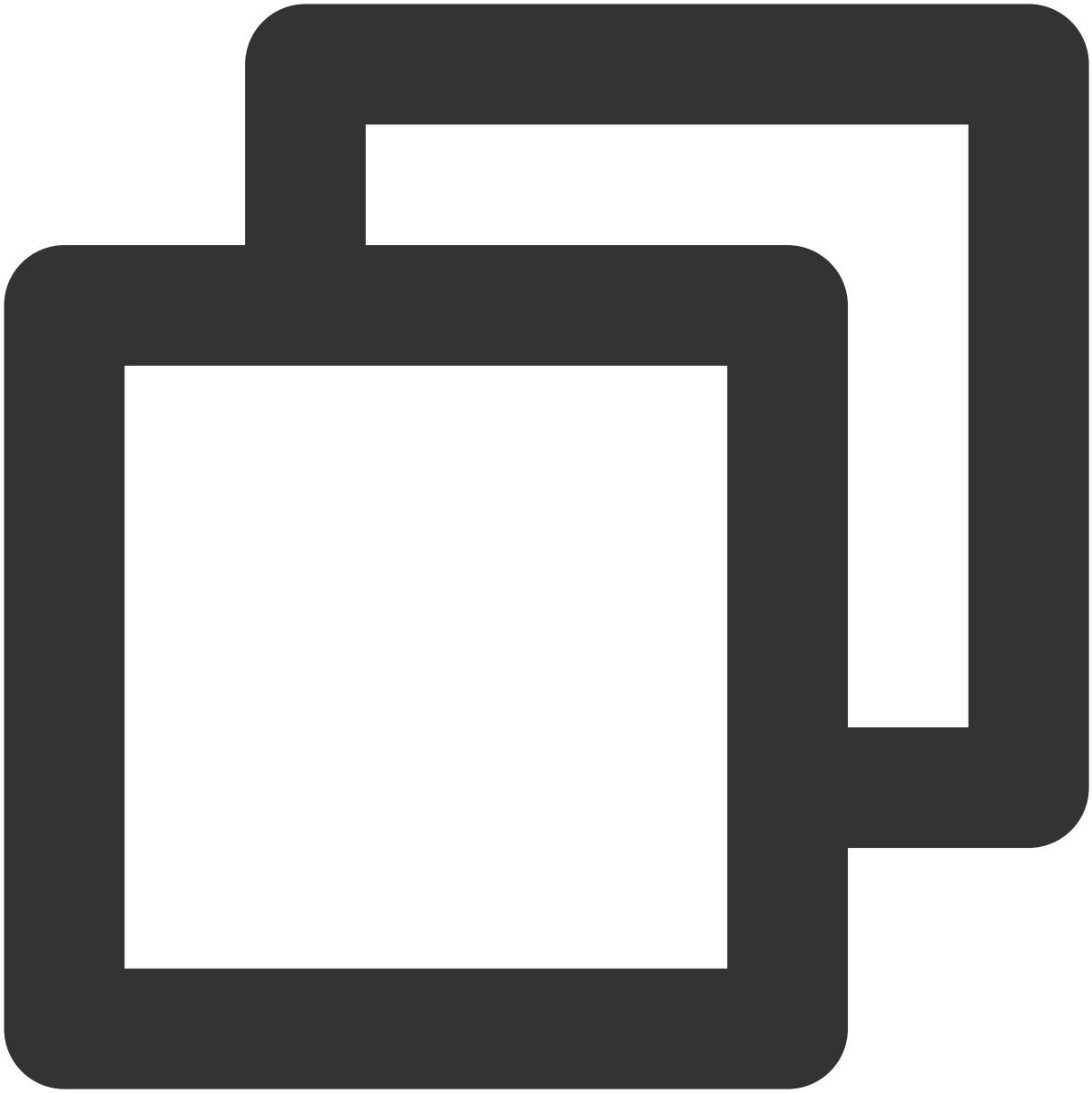
如果您需要指定 `TUIKit` 界面的语言，需要在引入 `TUIKit` 时设置 `language` 。



```
// language support en-US / zh-CN / ja-JP / ko-KR
<TUIKit language={'en-US'}></TUIKit>
```

动态切换语言

动态切换 `TUIKit` 的语言：



```
import React, { useState } from 'react';
import { TUIKit } from '@tencentcloud/chat-uikit-react';

// language support en-US / zh-CN / ja-JP / ko-KR
const languageList = ['en-US','zh-CN','ja-JP','ko-KR']

export default function SampleChat() {
  // language setting
  const [currentLanguage, setCurrentLanguage] = useState('en-US');
  const changeLanguage = (Language: string) => {
    setCurrentLanguage(Language);
```

```
  };

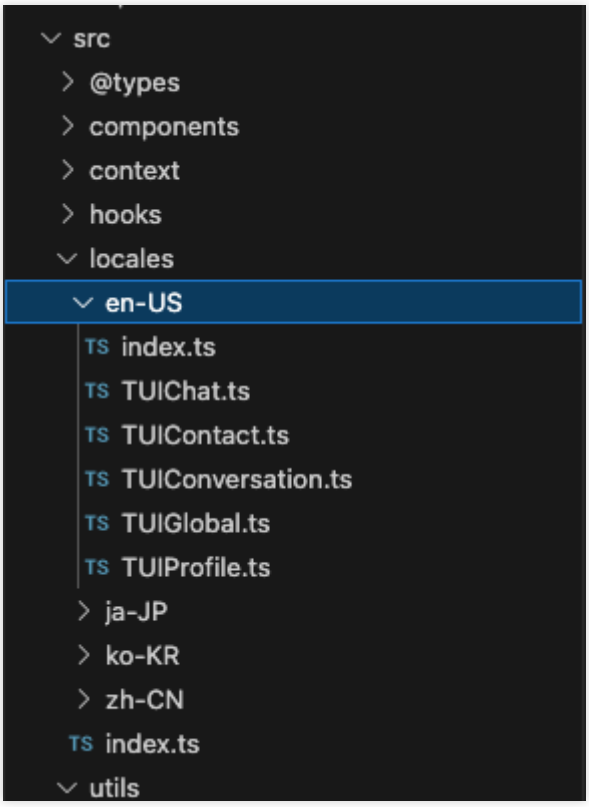
  return (
    // select language
    // <div @click="changeLanguage('en-US')">English</div>
    <TUIKit language={currentLanguage}></TUIKit>
  )
}
```

自定义语言词条

新增语言词条

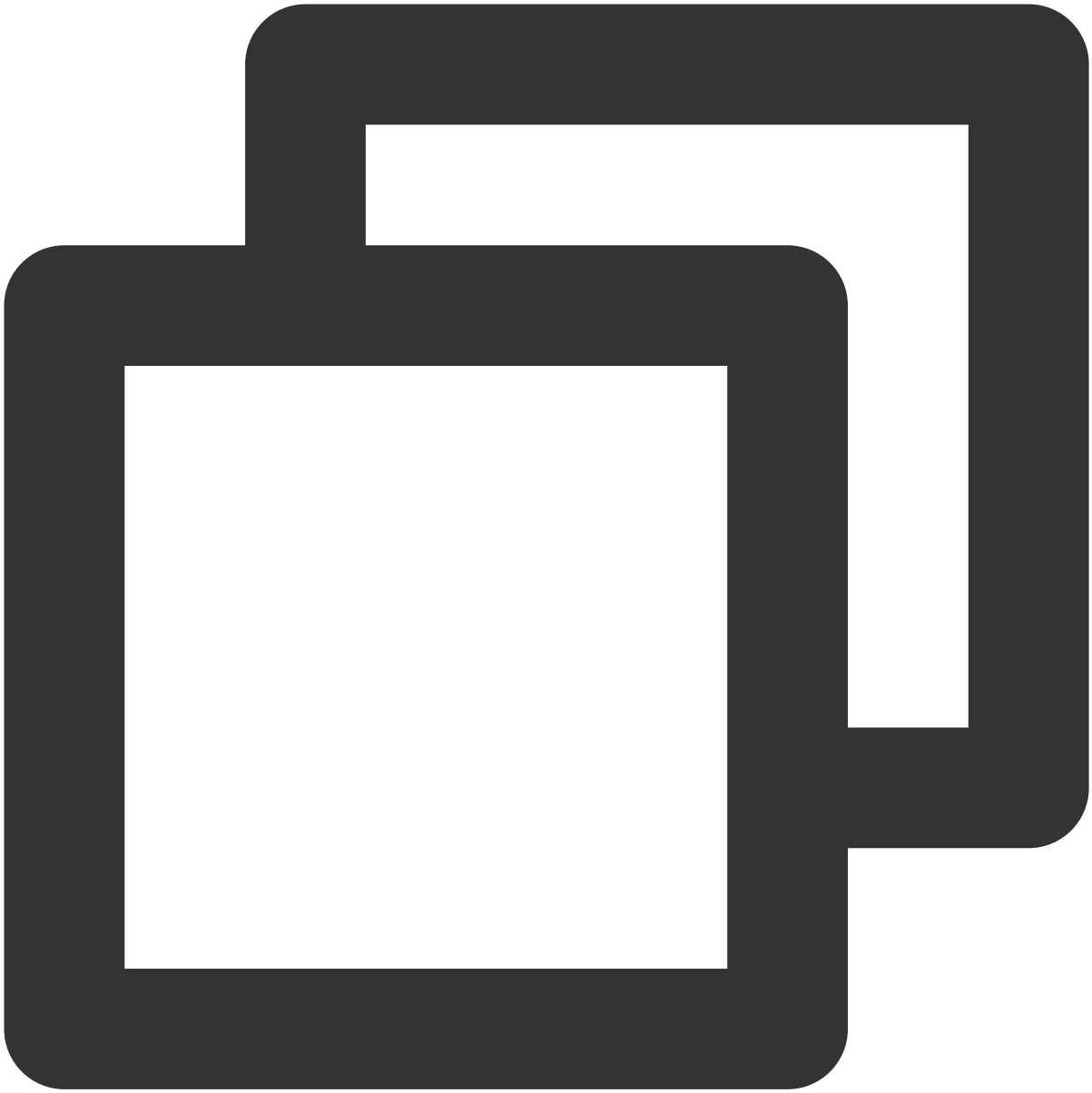
如果您需要对现有的 **英语 / 日语 / 韩语 / 中文** 语言包词条进行扩充或修改，可以在 `/TUIKit/src/locales` 目录下进行新增或修改词条。

locales 词条包目录结构如下图：



使用语言词条

以下使用 `react-i18next` 中的 `useTranslation` 进行词条翻译，更多接口详情请参考 react-i18next。



```
import { useTranslation } from 'react-i18next';
const { t } = useTranslation();

<p className="tui-contacts-list-title">{t('TUIContact.New Contacts')}</p>
```

交流与反馈

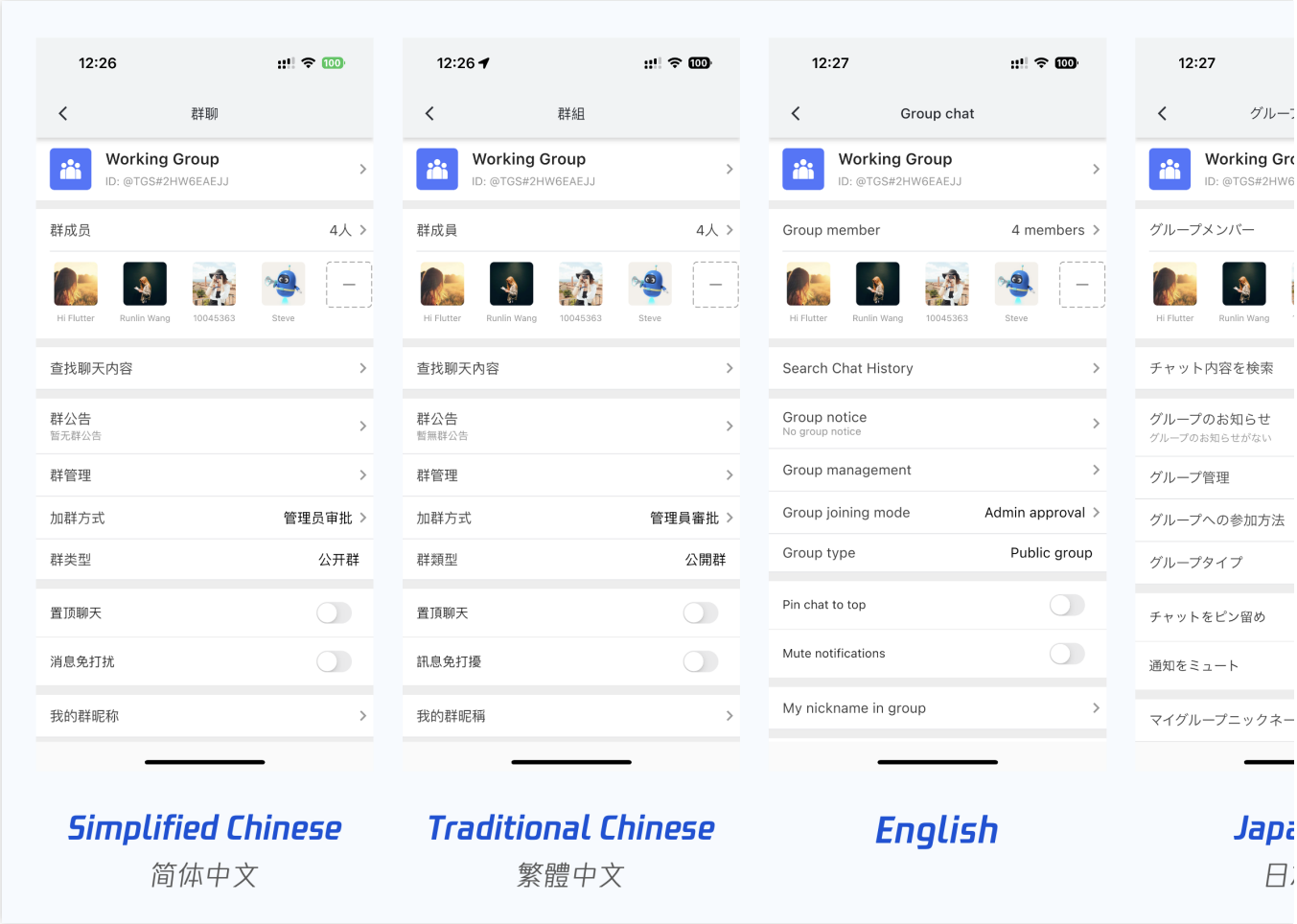
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-07-08 16:11:59

腾讯云 IM TUIKit 默认支持英语、简体中文、繁体中文、日语、韩语和阿拉伯语（支持 RTL）。



使用我们提供的语言条目键，您可以根据本教程的说明，为您的项目添加其他界面语言，甚至新增语言条目。

为了实现这一点，我们提供了一个国际化工具。

此工具包提供了一个轻量级、强大且对开发者友好的国际化语言工具，专为我们的包和客户的应用程序量身打造。

该工具基于 官方 Flutter `intl` 解决方案 进行了进一步开发和封装，以更好地满足我们的需求。

建议在使用此工具之前熟悉 官方国际化解决方案。对于 `.arb` 文件的语言模板编码和该工具未覆盖的其他因素，处理方式与 官方解决方案 相同。

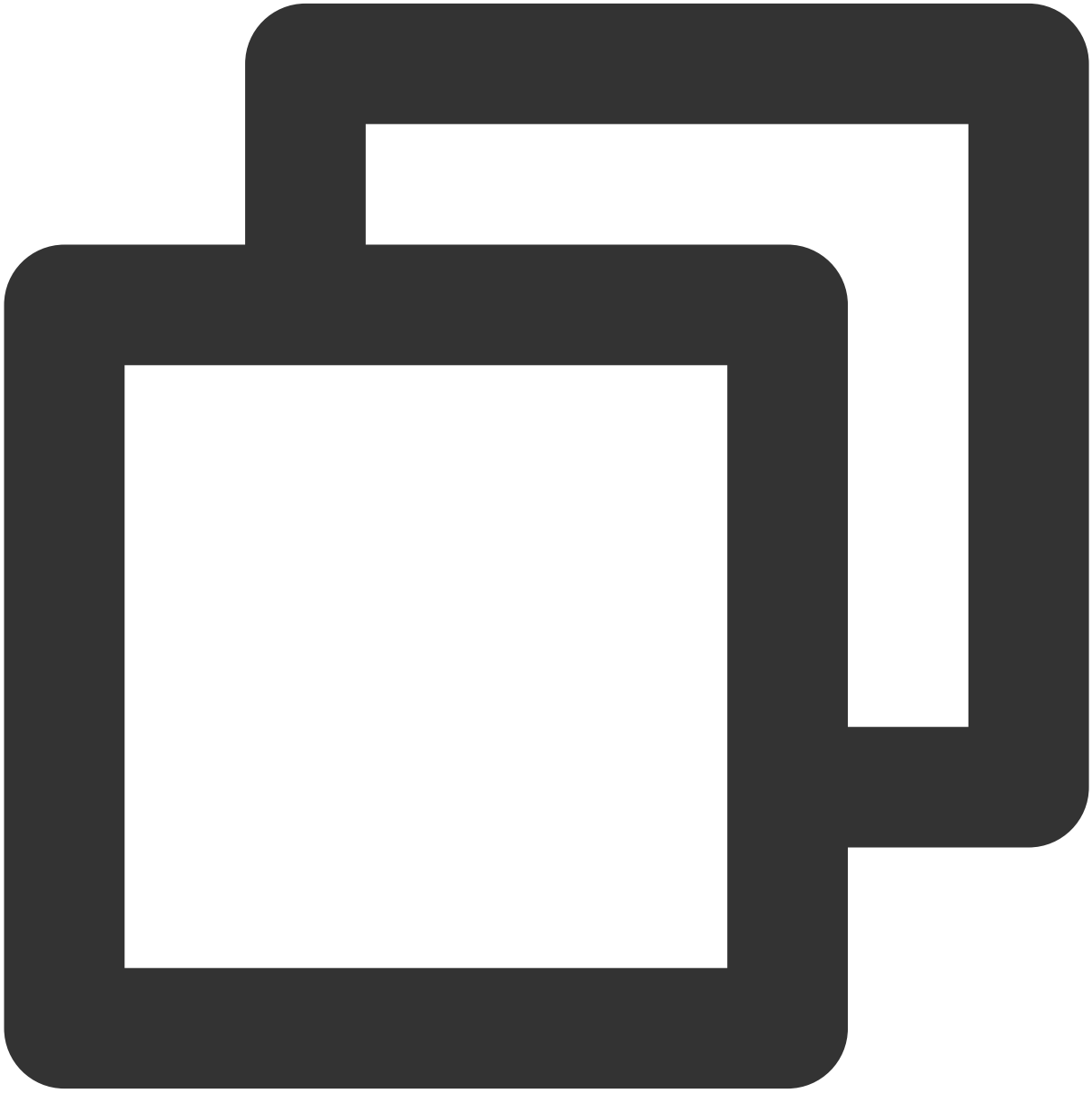
通过此工具包，您可以轻松管理多语言翻译条目，添加新条目，修改现有条目，甚至将新语言集成到您的项目中。

这大大简化了为聊天应用程序以及其他具有国际化需求的应用程序创建多语言用户体验的过程。

版权所有：腾讯云计算（北京）有限责任公司

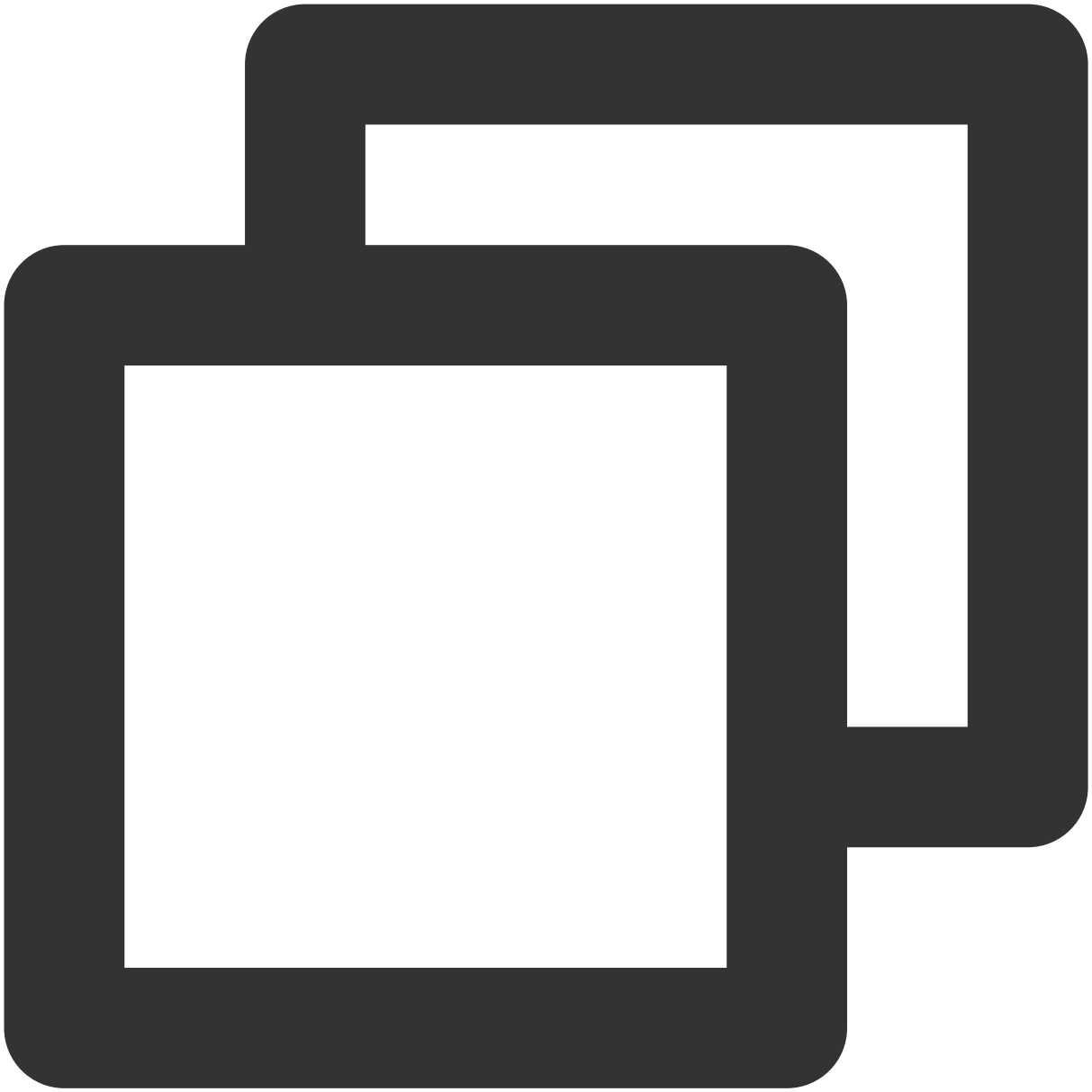
访问预定义的本地化字符串

由于我们的 UIKit 库已经将此工具包作为依赖项包含在内，因此您无需手动添加。
通过此设置，您可以轻松使用五种默认语言中的内置语言键条目，无需进一步实施。
1. 在您的项目中导入 `tencentcloud_chat_uikit_intl.dart` 文件：



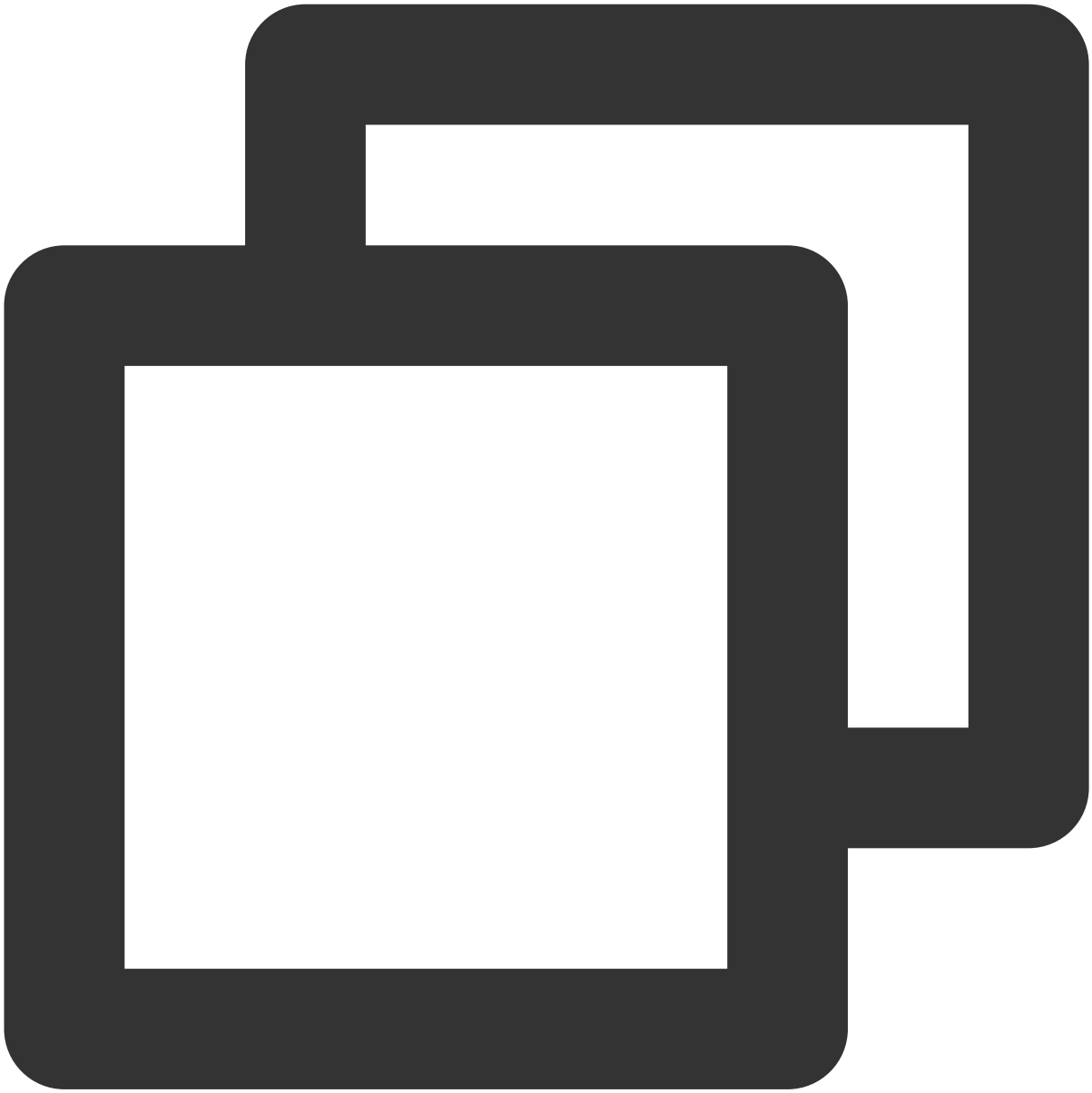
```
import 'package:tencentcloud_chat_uikit_intl/tencentcloud_chat_uikit_intl.dart';
```

2. 在导航到主页之前，使用 `BuildContext` 在主部件树中初始化 `TencentCloudChatUIKitIntl` ：



```
TencentCloudChatUIKitIntl.init(context);
```

3. 使用 `tL10n` 全局变量访问本地化字符串：



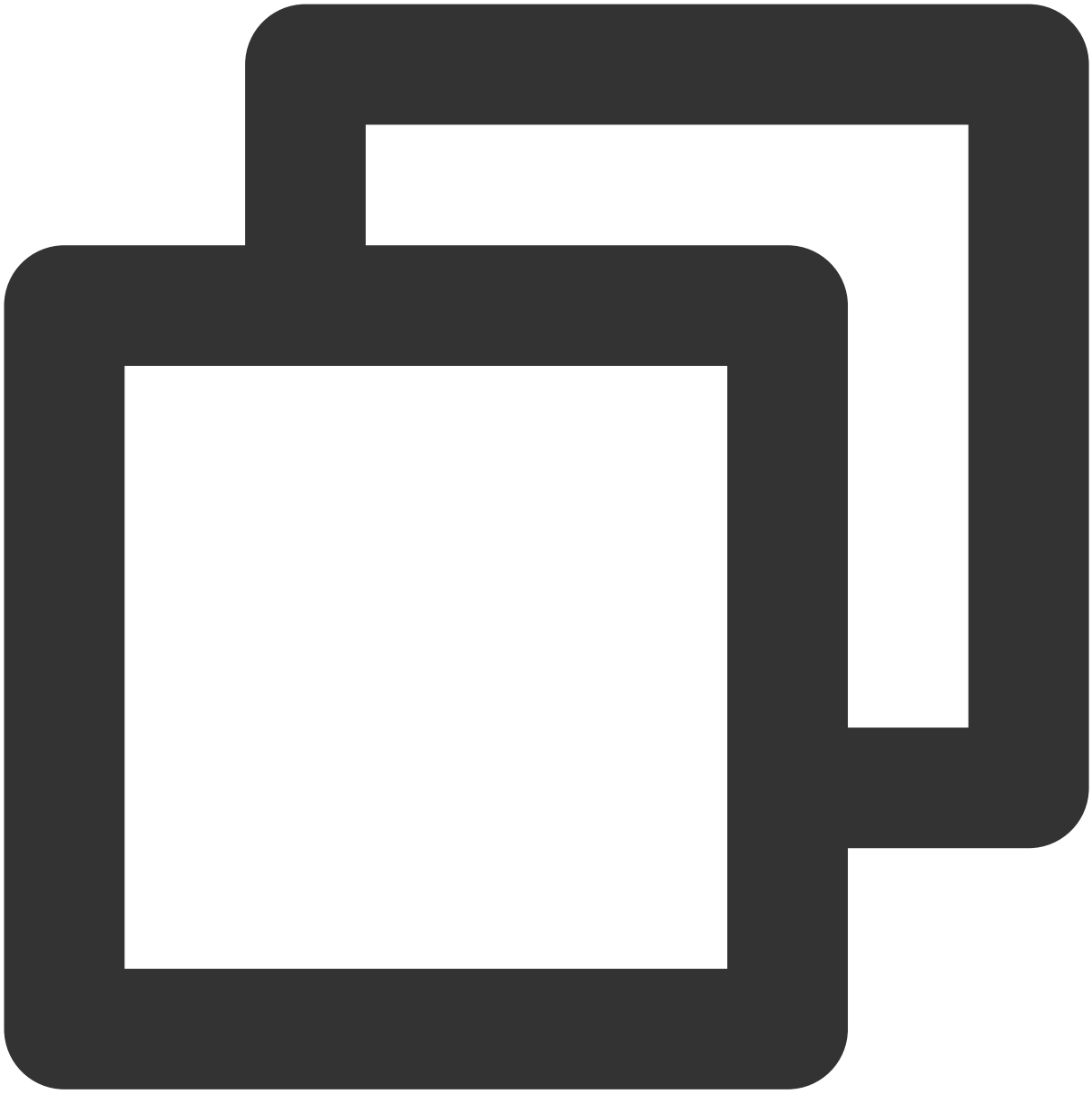
```
String album = tL10n.album;
```

通过遵循这些步骤，您可以轻松访问和使用 `tencentcloud_chat_uikit_intl` 包在您的项目中提供的现有本地化字符串。

定制国际化

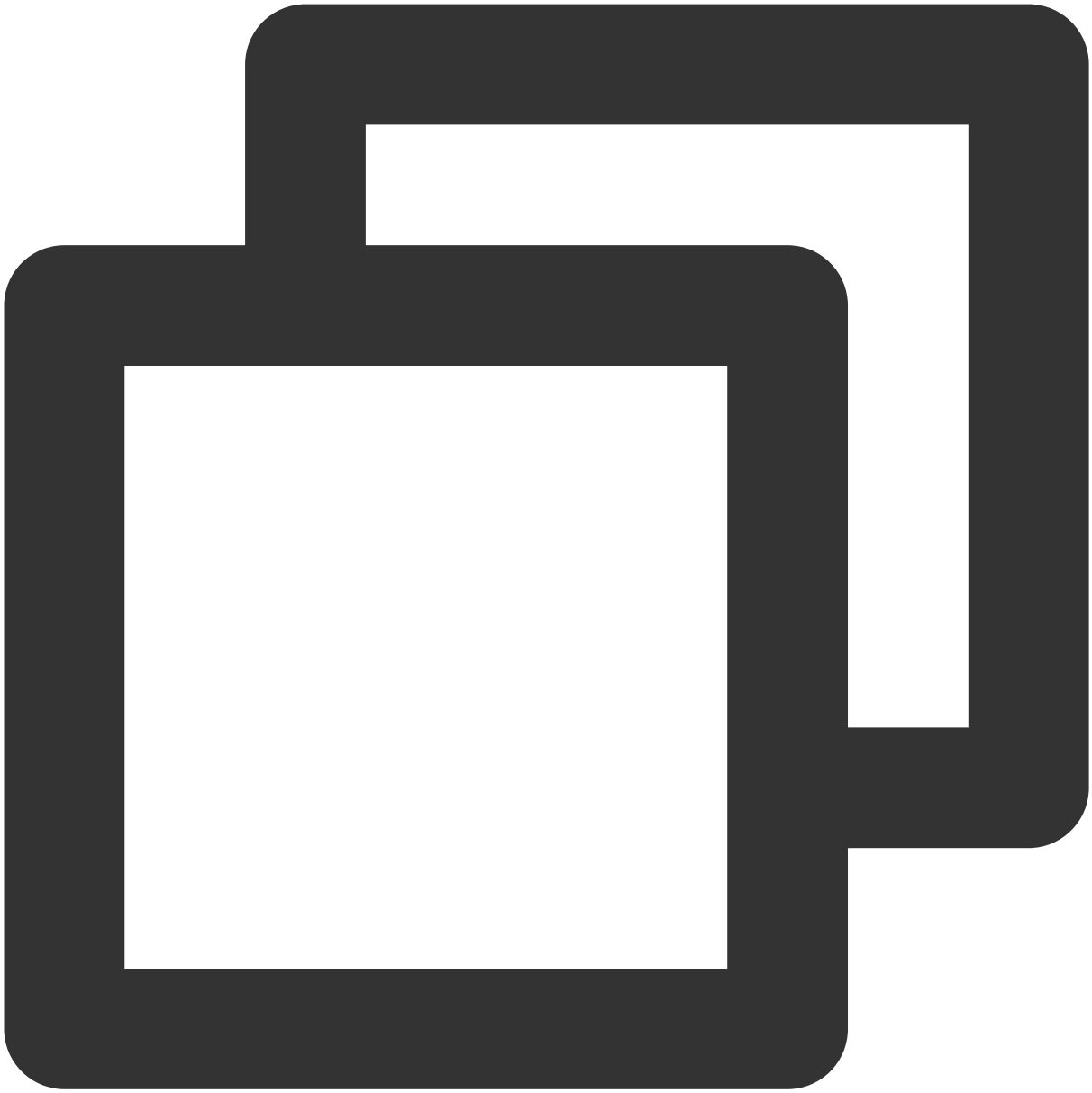
如果您想定制国际化功能，例如添加新支持的语言或修改现有翻译，请按照以下步骤操作：

1. 在 工具的 GitHub 仓库 上 Fork `tencentcloud_chat_uikit_intl` 仓库：
https://github.com/RoleWong/tencent_chat_intl_tool。这将创建该仓库的副本到您的 GitHub 账户下。
2. 将 Fork 的仓库克隆到您的本地机器上的一个目录。您可以使用以下 Git 命令执行此操作：



```
git clone https://github.com/<your-username>/tencentcloud_chat_uikit_intl.git
```

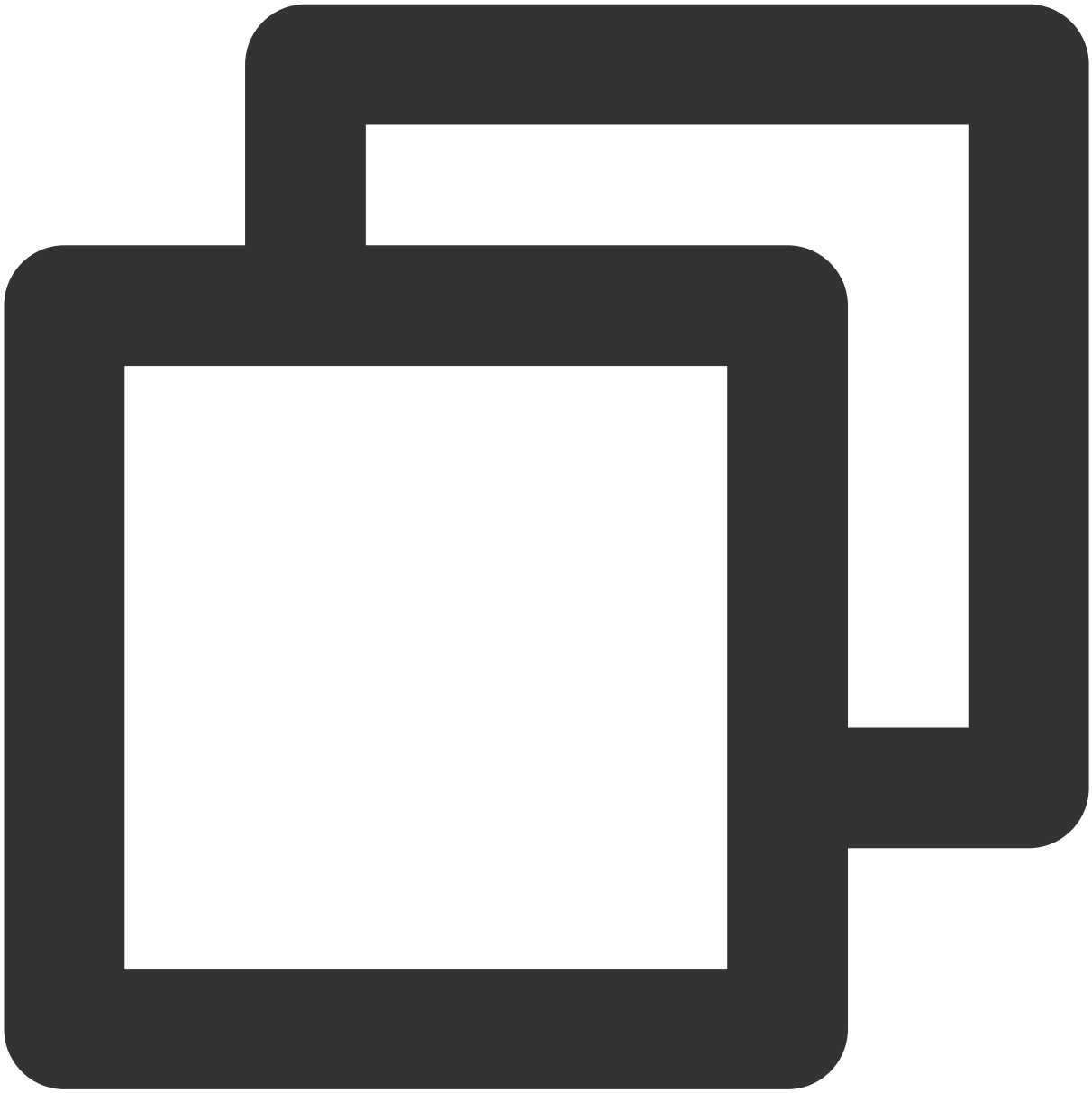
3. 使用 `dependency_overrides` 将 Fork 仓库的本地路径添加到您的项目的 `pubspec.yaml` 文件中：



```
dependency_overrides:
  tencentcloud_chat_uikit_intl:
    path: /path/to/your/forked/repository
```

将 `/path/to/your/forked/repository` 替换为本地克隆仓库的实际路径。

4. 在您的项目目录中运行以下命令：



```
dart run tencentcloud_chat_uikit_intl
```

该脚本将引导您完成定制国际化的过程，包括添加新语言条目和修改现有翻译。

添加新语言条目

1. 将新语言条目以 JSON 格式添加到项目根目录中的 `new_language_entries.txt` 文件中。确保遵循 Flutter intl 语法标准。您可以参考官方文档：https://docs.flutter.dev/ui/accessibility-and-localization/internationalization#adding-your-own-localized-messages。

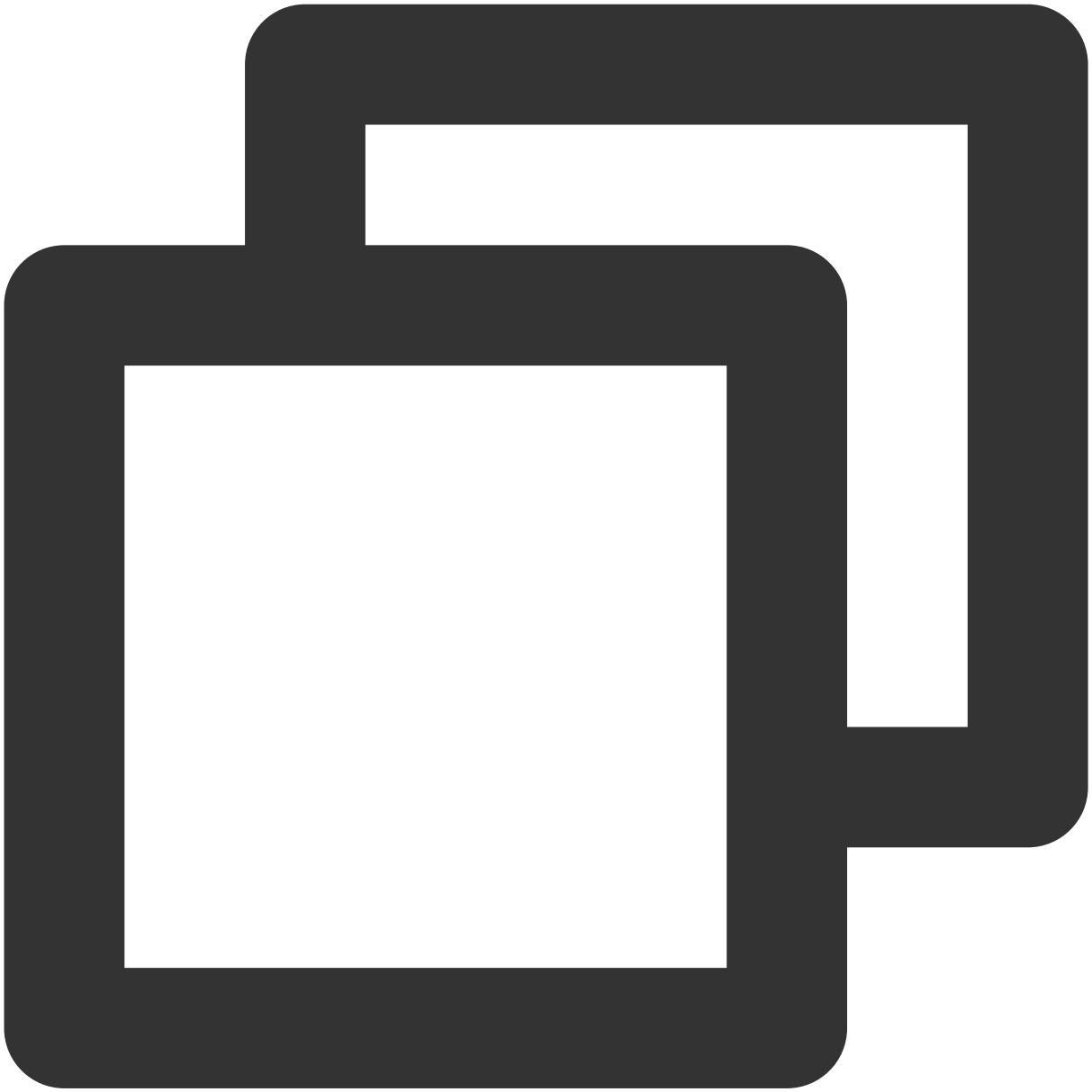
2. 运行 `dart run tencentcloud_chat_uikit_intl` 命令并选择选项 `A` 将新条目合并到工具的内置 ARB 文件中。
3. 添加新条目后，继续进行下一步以翻译它们。

修改现有翻译和添加新语言支持

1. 运行 `dart run tencentcloud_chat_uikit_intl` 命令并选择选项 `B` 将内置语言条目（ARB 文件）复制到您的项目目录。
2. 根据需要修改 `languages` 目录中的 ARB 文件。
3. 要添加对新语言的支持，请按照以下步骤操作：
导航到项目中的 `languages` 目录。
选择一个您熟悉的区域设置 `.arb` 文件并复制一份。
将复制的文件重命名为 `l10n_${language code}_${script code}_${country code}.arb`，其中 `language code` 是必需的，`script code` 和 `country code` 是可选的，例如 `l10n_fr.arb` 表示法语，`l10n_zh_Hant_HK.arb` 表示香港繁体中文。
如果添加一个新区域设置指定了脚本代码或国家代码，也要创建一个基础区域设置文件（不带脚本代码或国家代码）。
将新区域设置文件中的所有条目翻译成相应的语言。
4. 运行 `dart run tencentcloud_chat_uikit_intl` 命令并选择选项 `C` 应用您的更改。

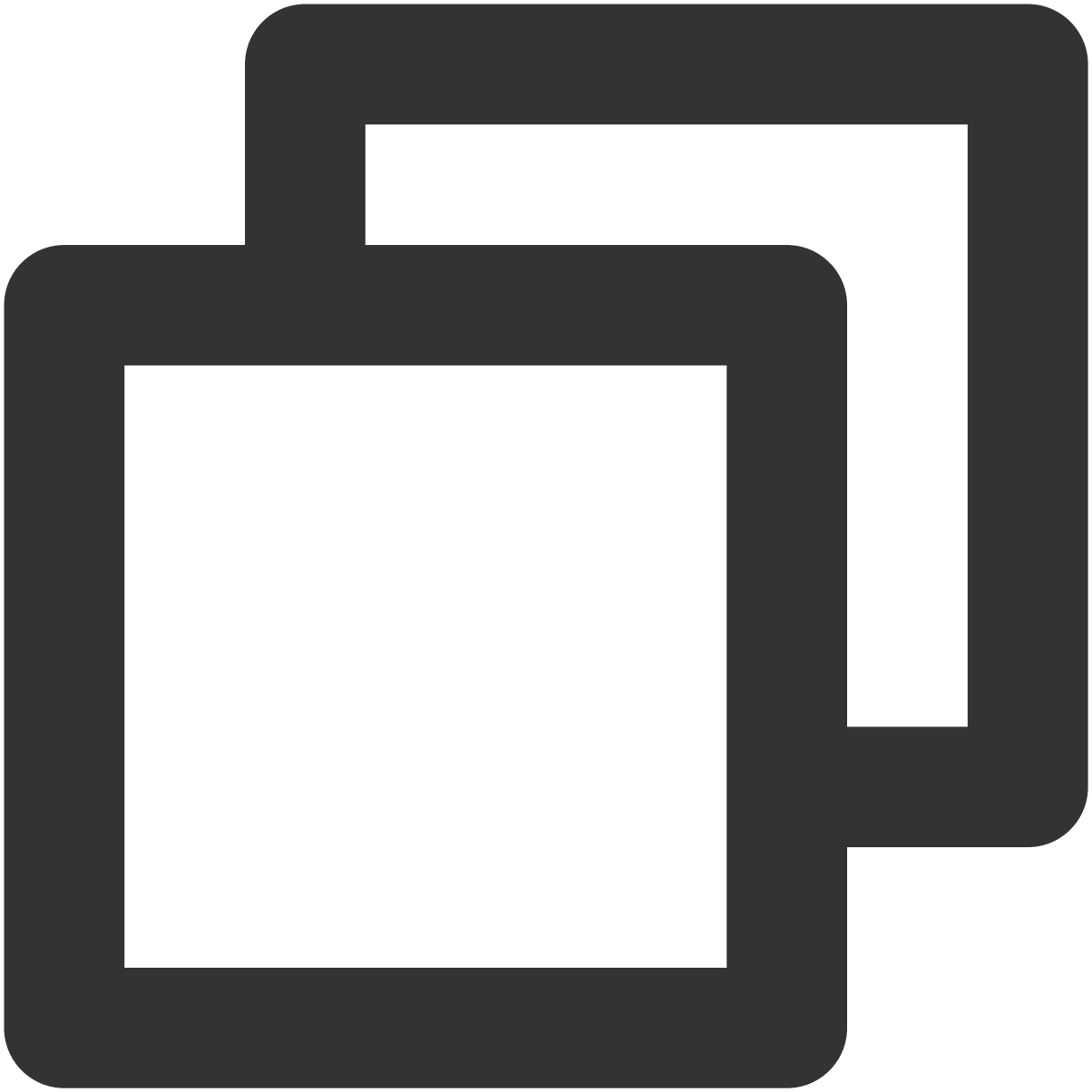
工具更新说明

为了保持版本一致性，腾讯云聊天 intl 包将与腾讯云聊天 UIKit 同步更新。每次更新时，我们会将 Chat UIKit 最新版本中的新条目添加到此包中。所有更新将同步发布在 pub.dev 和 GitHub 仓库。
如果您已将此包 Fork 到您的 GitHub 账户，请注意每当 Chat UIKit 更新时，您需要通过 `pull upstream` 操作同步此包的最新条目库到您的 Fork 版本。这确保了您的 Fork 版本包含您添加或修改的条目以及我们每个版本添加的新条目。在合并代码并解决冲突时，请确保每个 JSON 条目库完好无损。
如果合并后的 JSON 文件不能直接使用，您可以按照上面第 7 步中的说明重新运行程序并选择选项 C 以应用更新。请注意，在执行选项 C 之前，您需要确保每个语言条目 JSON 完整且无错误。
以下是 `pull upstream` Git 操作的分步示例：
1. 首先，将上游远程仓库添加到本地仓库：



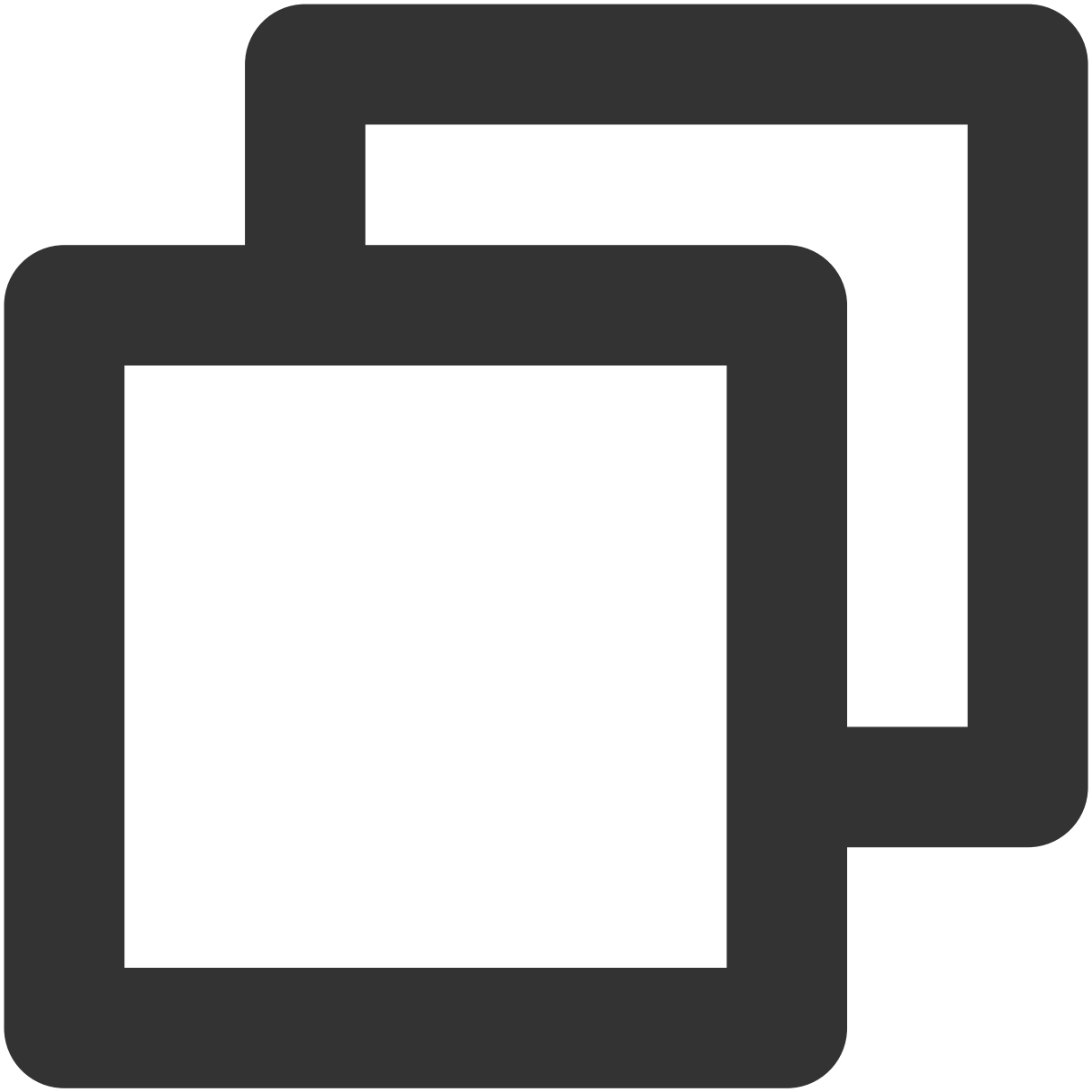
```
git remote add upstream https://github.com/RoleWong/tencent_chat_intl_tool
```

2. 从上游仓库获取最新更改：



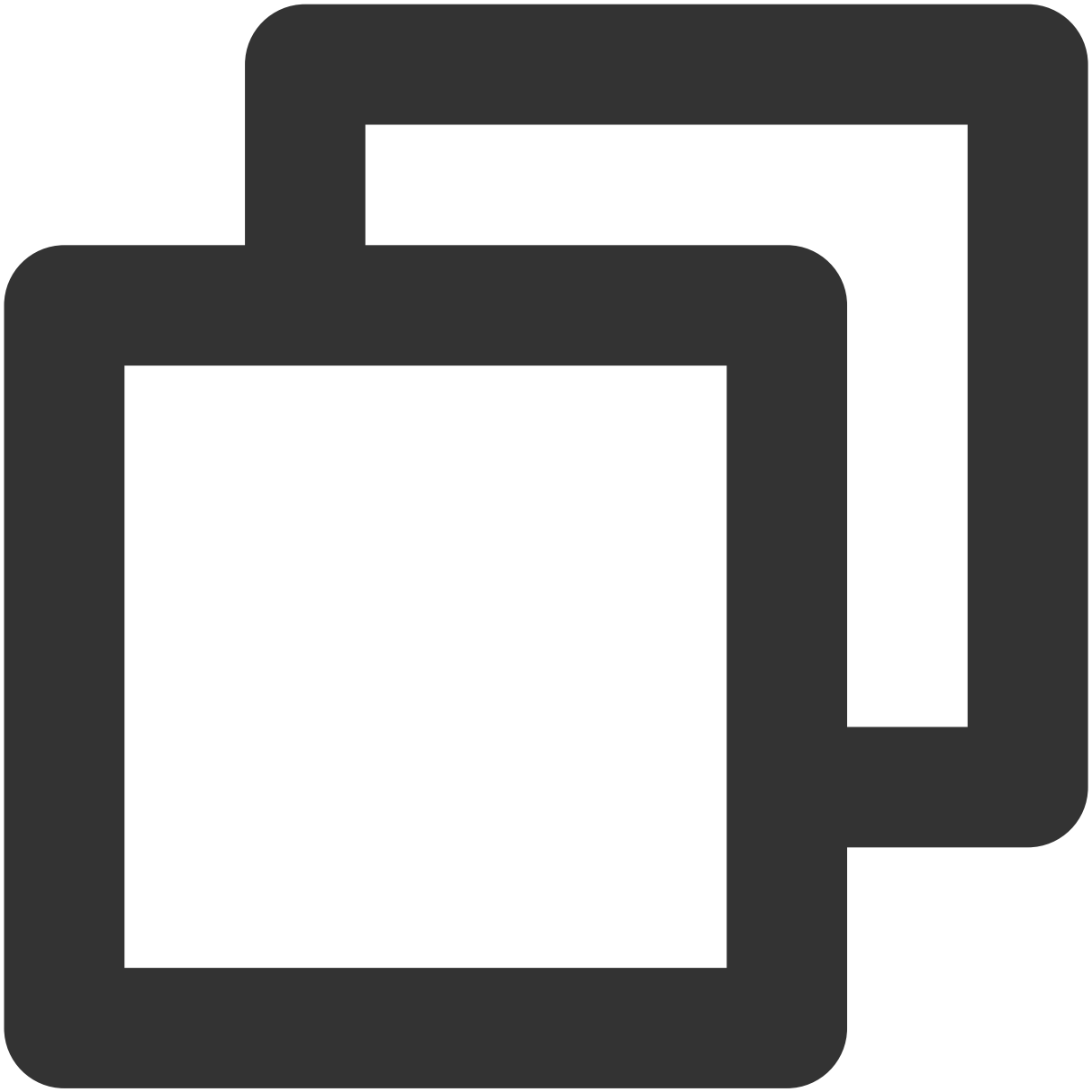
```
git fetch upstream
```

3. 将本地仓库切换到您要更新的分支（例如 `main` 或 `master`）：



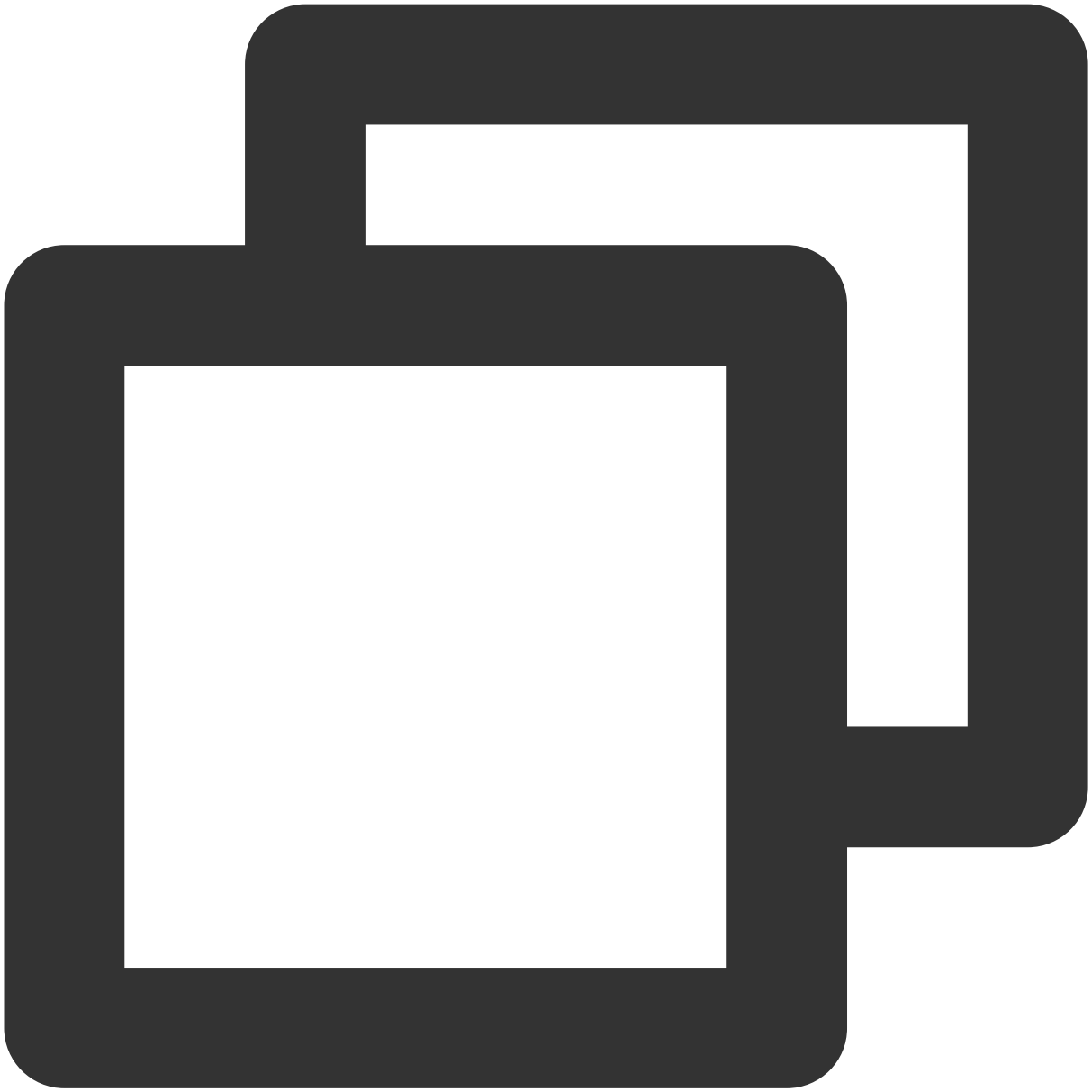
```
git checkout main
```

4. 将上游仓库的更改合并到本地仓库：



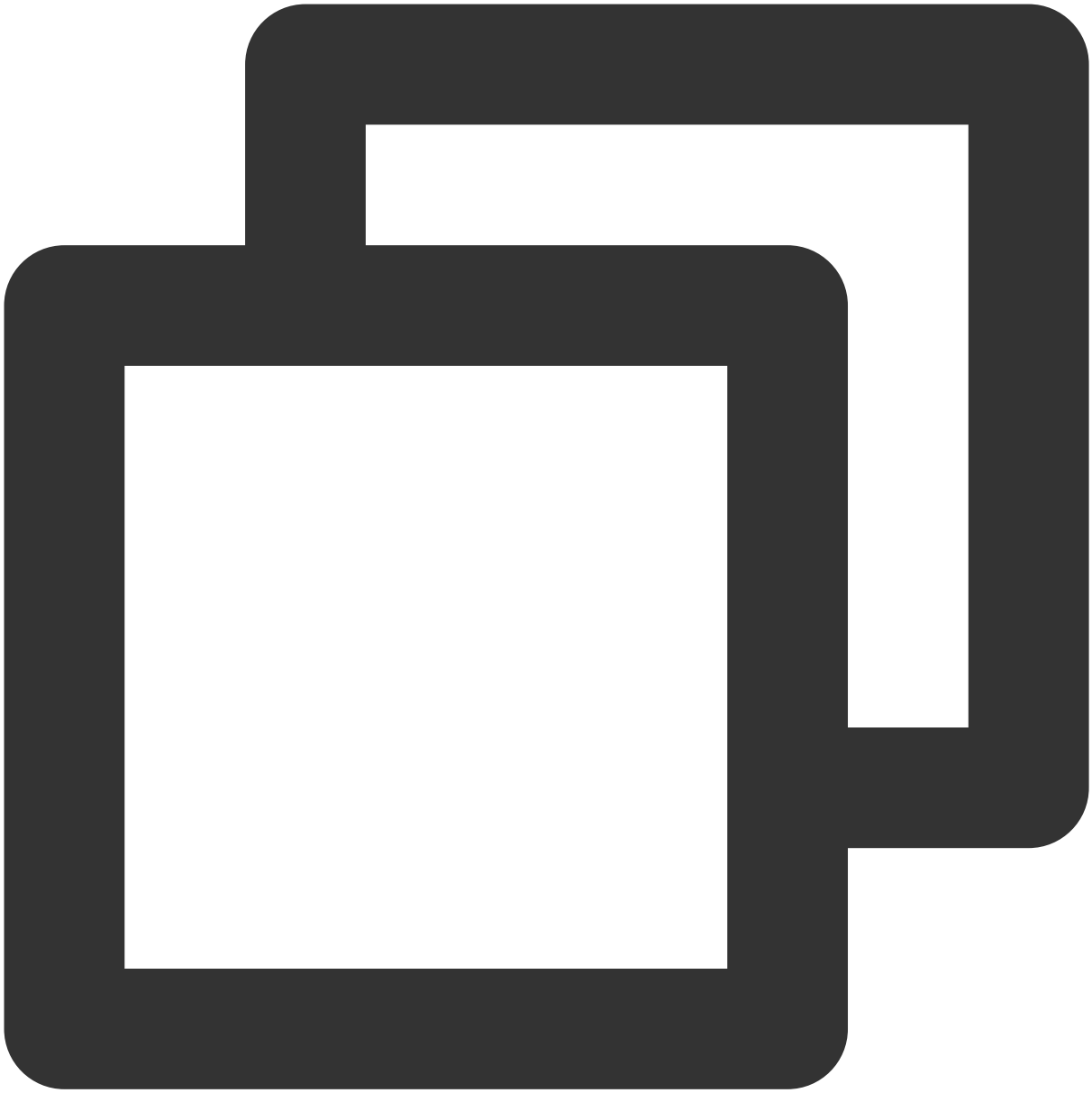
```
git merge upstream/main
```

5. 如果有任何冲突，请在编辑器中解决它们，确保每个 JSON 条目库完好无损。
6. 解决冲突后提交更改：



```
git add .
git commit -m "Merge upstream changes and resolve conflicts"
```

7. 将更改推送到您的远程仓库：



```
git push origin main
```

现在，您的 Fork 版本包含最新的条目库。如果需要应用更新，请按照上面第 7 步中的说明重新运行程序并选择选项 C。

附录：语言代码表

语言	代码	语言	代码

阿拉伯语	ar	保加利亚语	bg
克罗地亚语	hr	捷克语	cs
丹麦语	da	德语	de
希腊语	el	英语	en
爱沙尼亚语	et	西班牙语	es
芬兰语	fi	法语	fr
爱尔兰语	ga	印地语	hi
匈牙利语	hu	希伯来语	he
意大利语	it	日语	ja
朝鲜语/韩语	ko	拉脱维亚语	lv
立陶宛语	lt	荷兰语	nl
挪威语	no	波兰语	pl
葡萄牙语	pt	瑞典语	sv
罗马尼亚语	ro	俄语	ru
塞尔维亚语	sr	斯洛伐克语	sk
斯洛文尼亚语	sl	泰语	th
土耳其语	tr	乌克兰语	uk
中文（简体）	zh-Hans	中文（繁体）	zh-Hant

完整版请见此处。

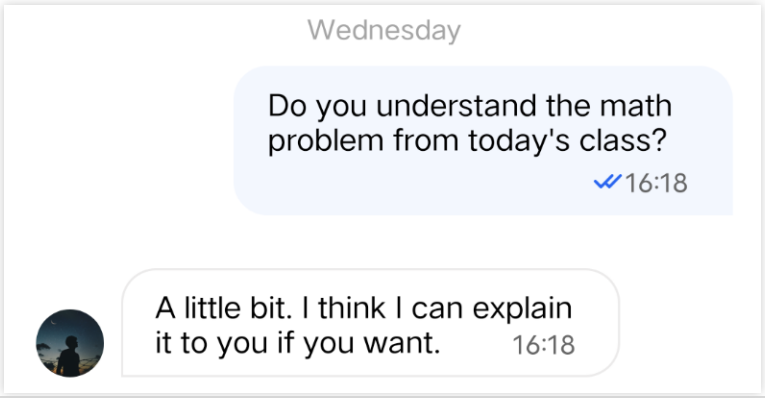
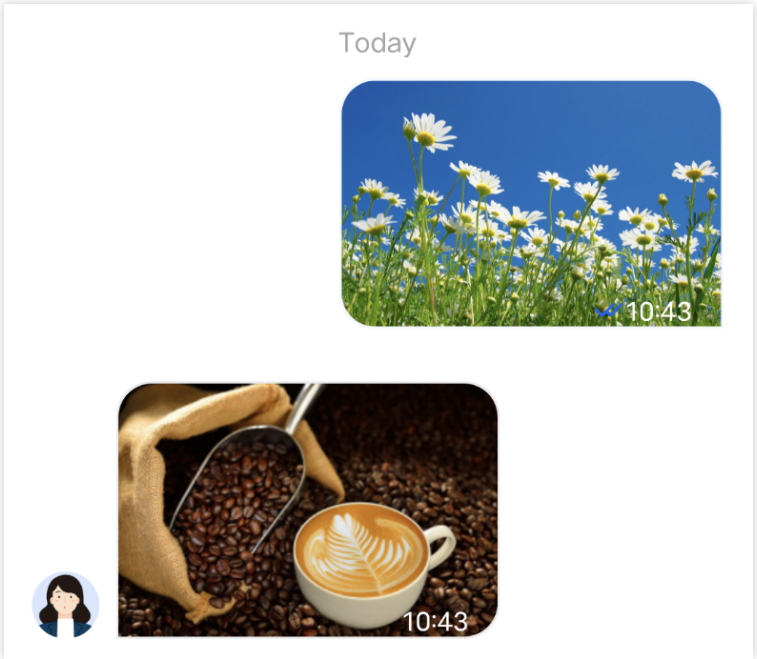
添加自定义消息

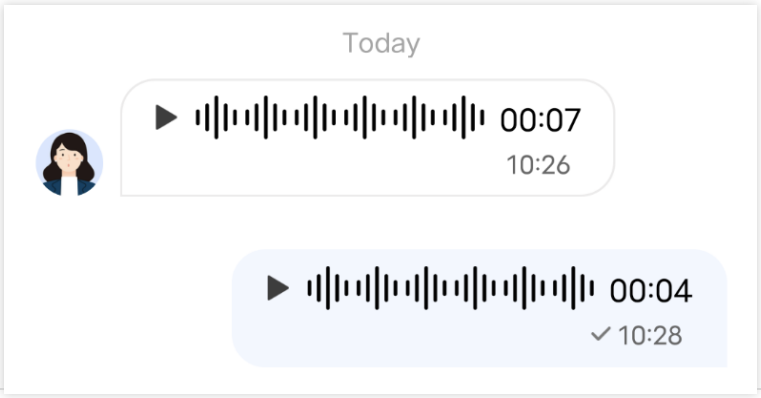
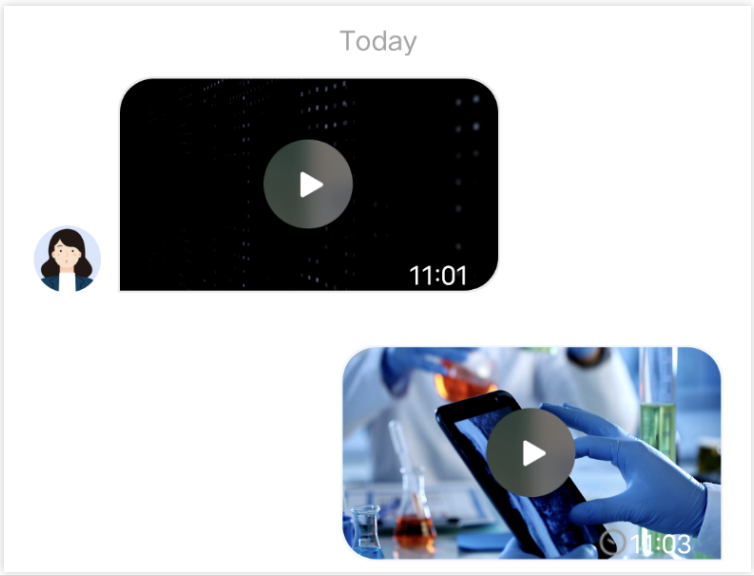
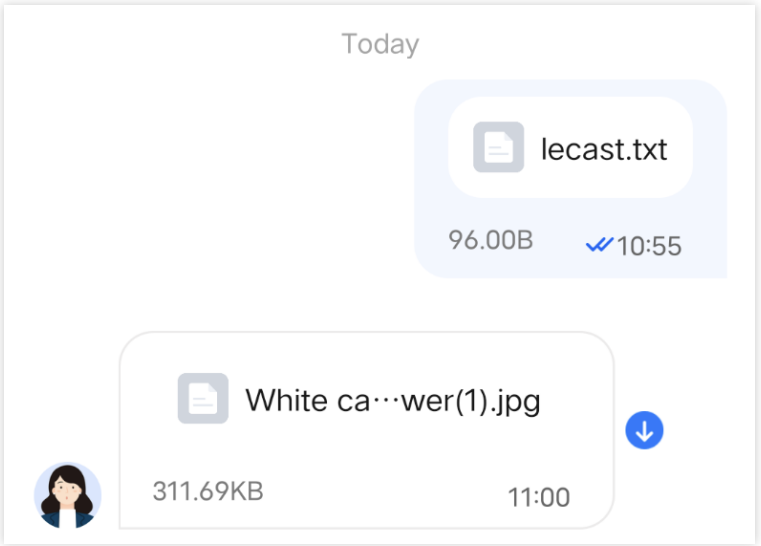
Android

最近更新时间：2024-05-20 15:17:21

TUIKit 默认实现了文本、图片、语音、视频、文件等基本消息类型的发送和展示，如果这些消息类型满足不了您的需求，您可以新增自定义消息类型。

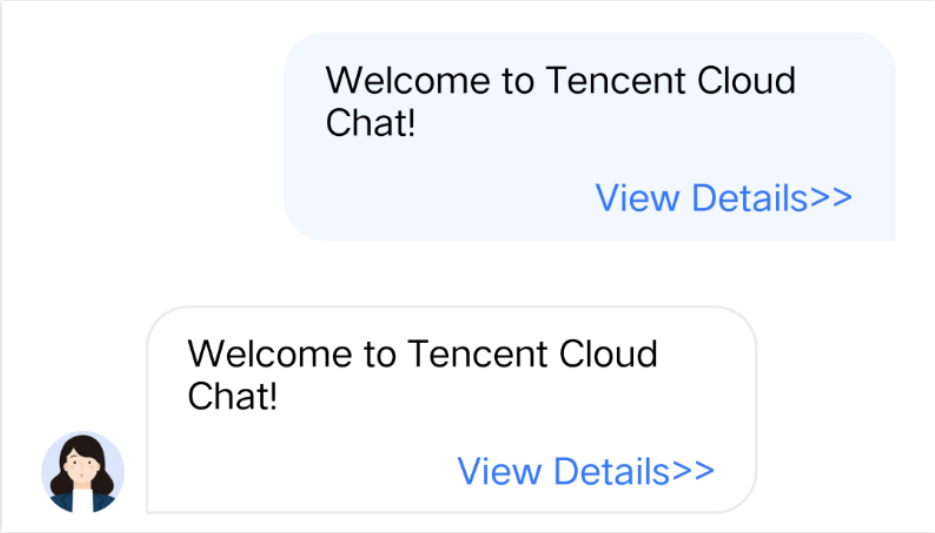
基本消息类型

消息类型	显示效果图
文本类消息	
图片类消息	

语音类消息	
视频类消息	
文件类消息	

自定义消息

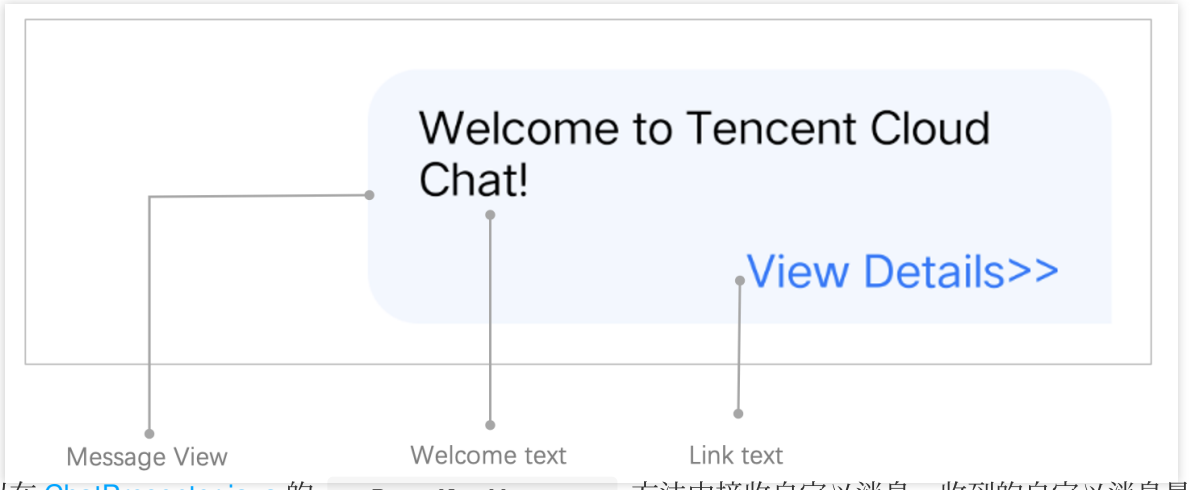
如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。下文以发送一条可跳转至浏览器的超文本作为自定义消息为例，帮助您快速了解实现流程。
TUIKit 内置的自定义消息样式如下图所示：



说明：
TUIKit 在 5.8.1668 版本重新设计了一套自定义消息方案，新方案较旧方案有很大的改动，实现起来更简单快捷。旧方案 API 继续保留，但不再维护。
我们**强烈建议**您升级到 5.8.1668 及以上版本，使用新方案实现自定义消息。

展示自定义消息

TUIKit 内置的自定义消息 View 元素如下图所示：

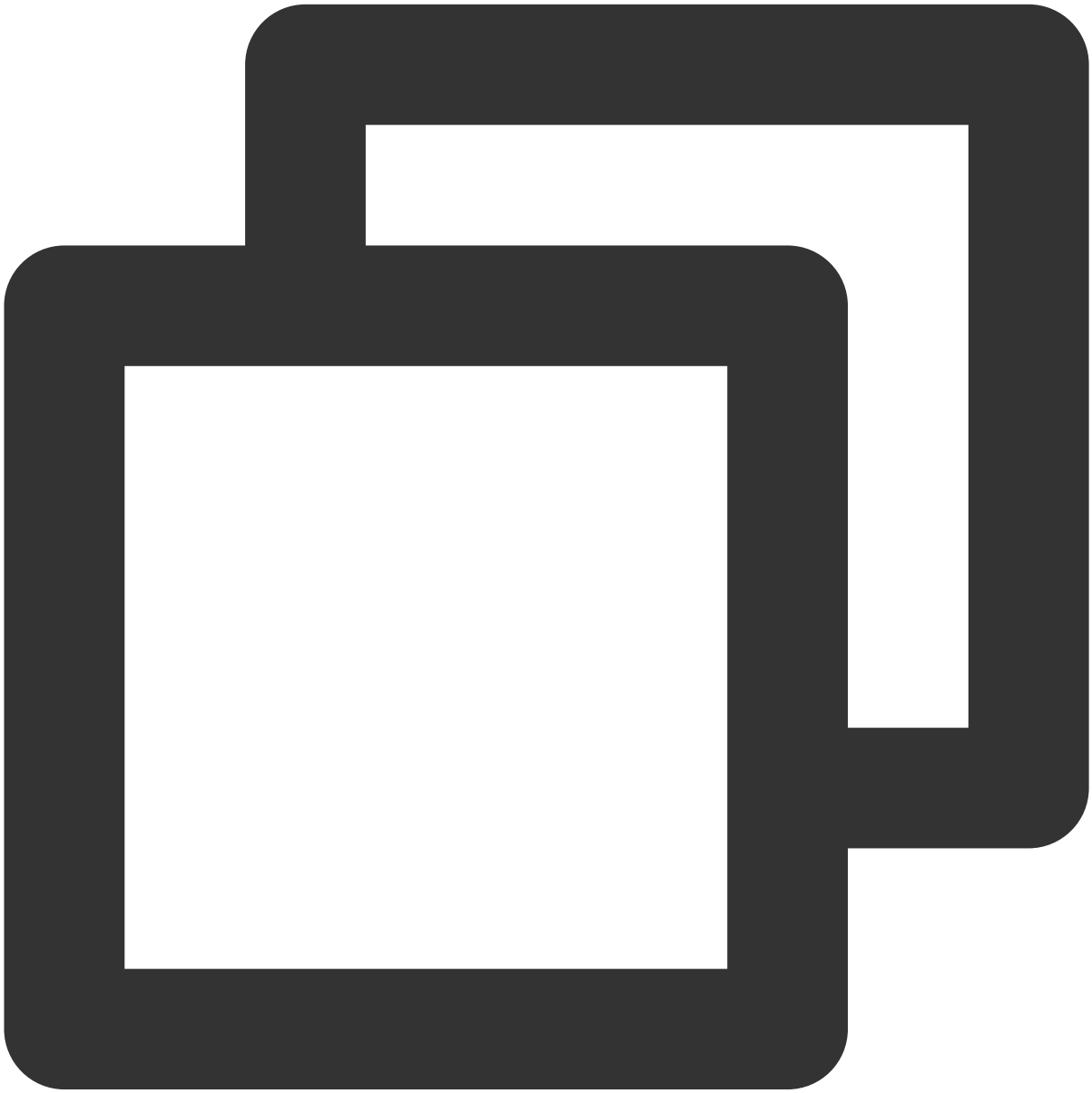


您可以在 ChatPresenter.java 的 `onRecvNewMessage` 方法中接收自定义消息。收到的自定义消息最终会以 `MessageViewHolder` 的形式展示在消息列表中，`MessageViewHolder` 绘制所需的数据我们称之为 `MessageBean` 。
下面我们分步骤讲解下如何展示自定义消息。

实现自定义消息 MessageBean 类

1. 在 `TUIChat/tuichat/src/main/java/com/tencent/qcloud/tuikit/tuichat/bean/message/` 文件夹下新建 `CustomLinkMessageBean.java` 文件，`CustomLinkMessageBean` 类继承自 `TUIMessageBean`，用于存储显示的文字和要跳转的链接。

示例代码如下：



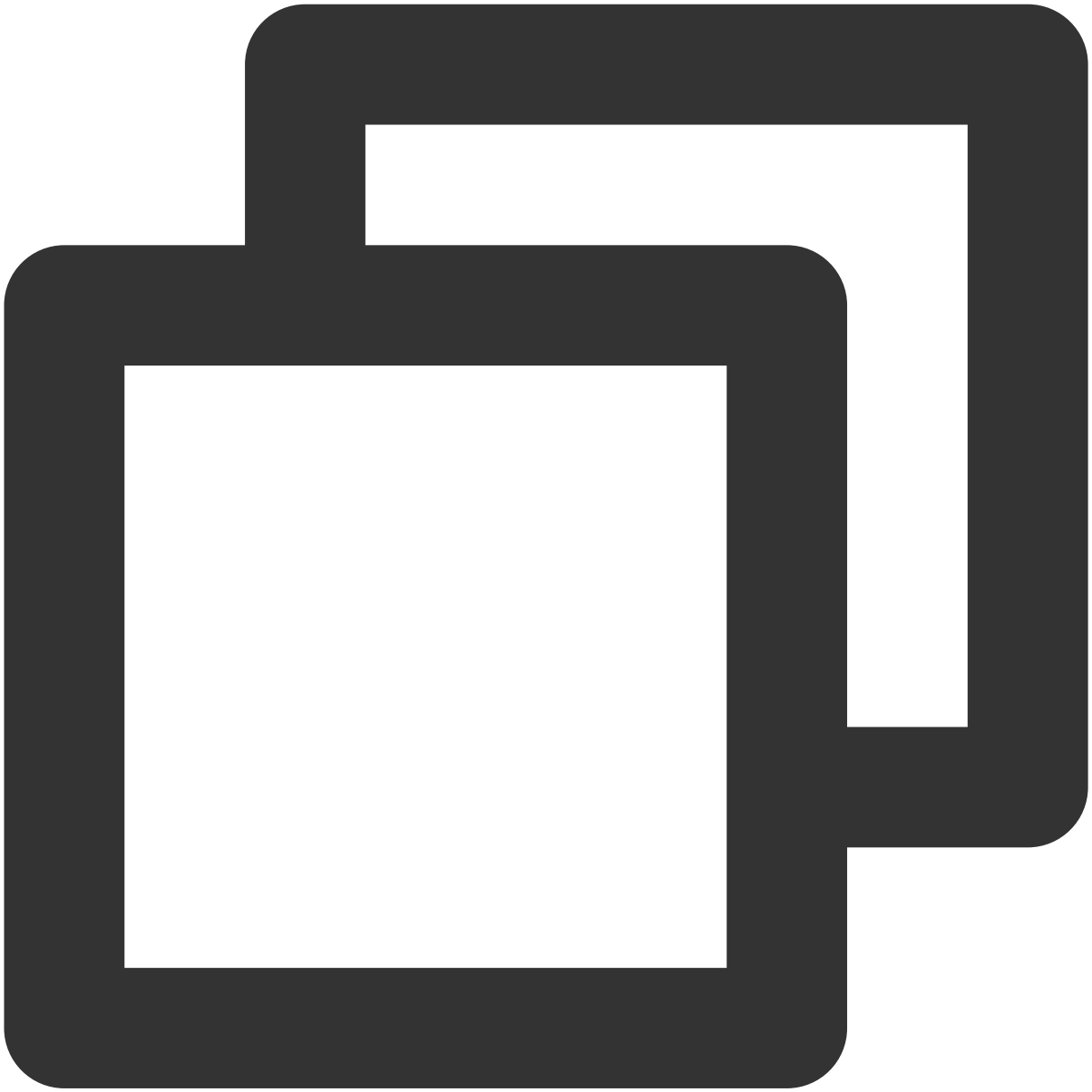
```
public class CustomLinkMessageBean extends TUIMessageBean {
    private String text;
    private String link;
```

```
    public String getText() {
        return text;
    }

    public String getLink() {
        return link;
    }
}
```

2. 重写 `CustomLinkMessageBean` 的 `onProcessMessage(message)` 方法，用于实现对自定义消息的解析。

示例代码如下：



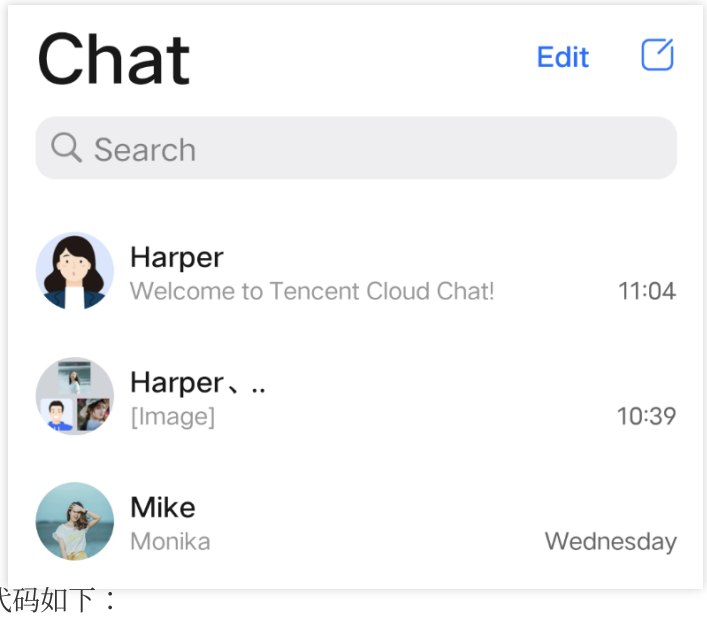
```
@Override
public void onProcessMessage(V2TIMMessage v2TIMMessage) {
    // Custom message view implementation. Here we configure to display only the te
    text = "";
    link = "";
    String data = new String(v2TIMMessage.getCustomElem().getData());
    try {
        HashMap map = new Gson().fromJson(data, HashMap.class);
        if (map != null) {
            text = (String) map.get("text");
            link = (String) map.get("link");
```

```
            }
        } catch (JsonSyntaxException e) {

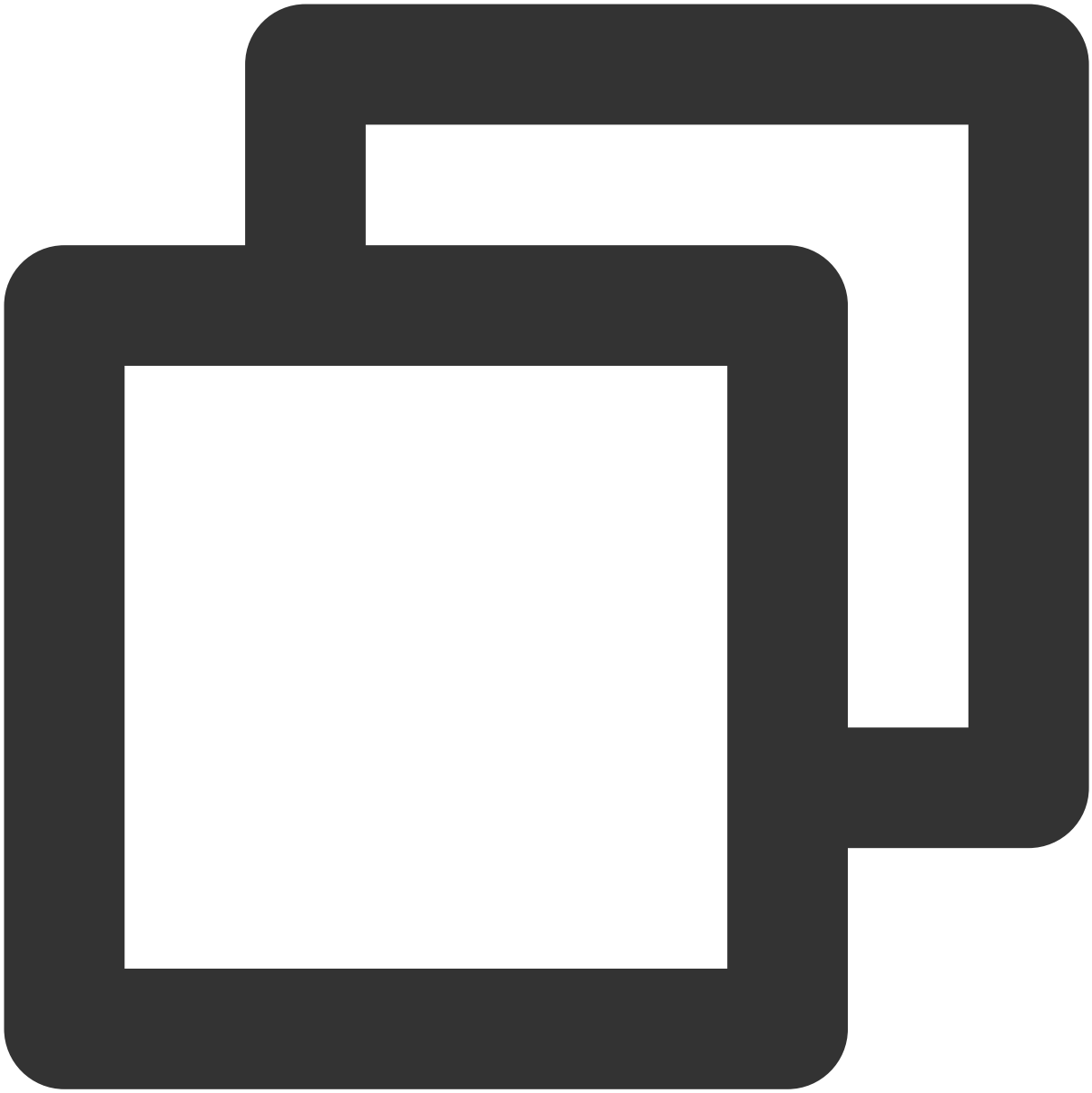
        }
        setExtra(text);
    }
```

3. 重写 `CustomLinkMessageBean` 的 `onGetDisplayString()` 方法，用于生成在会话列表中的文字摘要。

实现后的效果如下：



示例代码如下：



```
@Override
public String onGetDisplayString() {
    return text;
}
```

实现 MessageViewHolder 类

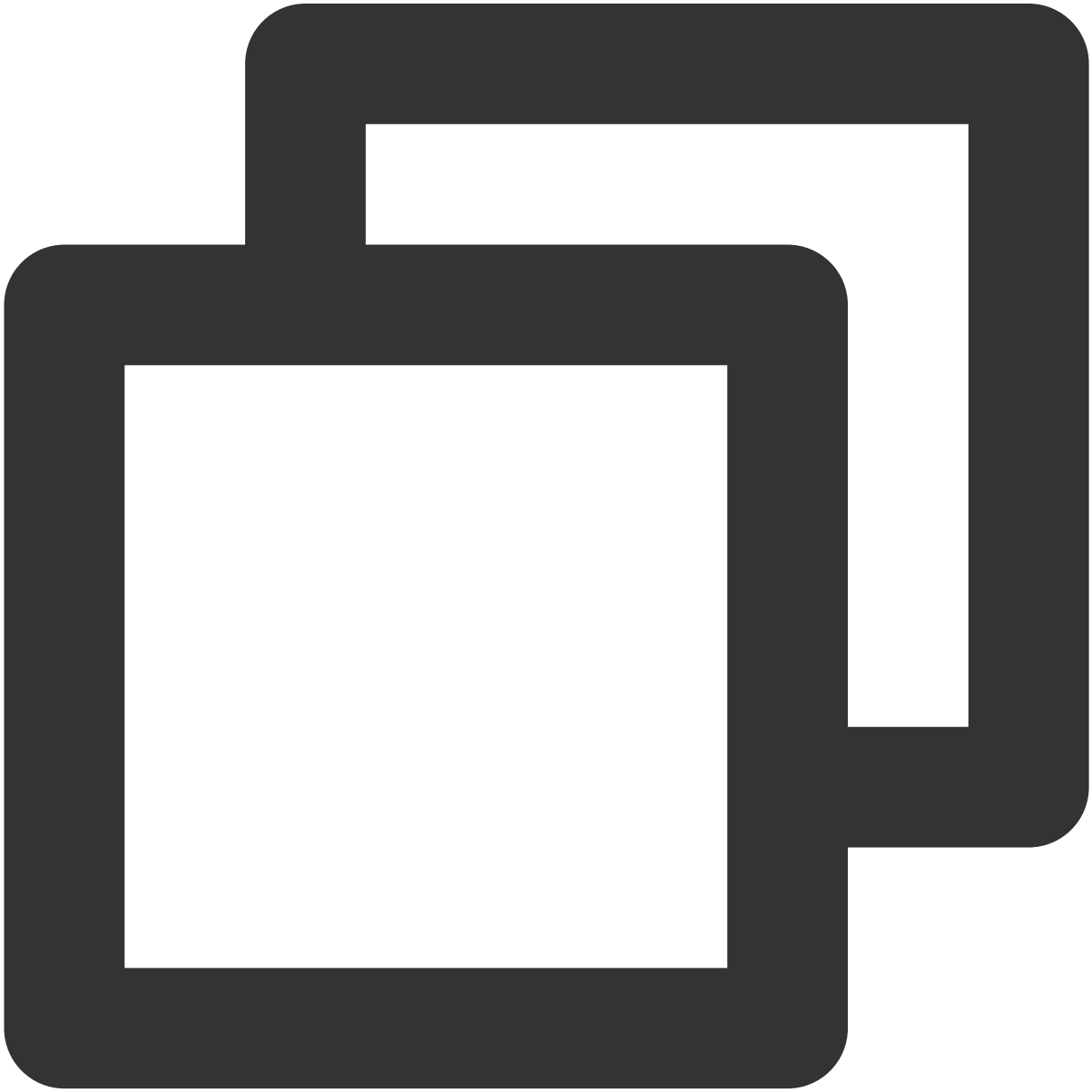
1. 在 `Android/TUIChat/tuichat/src/main/java/com/tencent/qcloud/tuikit/tuichat/minimalistui/widget/message/viewholder/` 文件夹下新建 `CustomLinkMessageHolder.java` 文

件， `CustomLinkMessageHolder` 继承自 `MessageContentHolder` ，用于实现自定义消息气泡的样式布局和点击事件。

注意：

如果是使用经典版 UI，CustomLinkMessageHolder 需要继承 com.tencent.qcloud.tuikit.timcommon.**classicui**.widget.message.MessageContentHolder，如果是使用 简约版 UI，则需要继承 com.tencent.qcloud.tuikit.timcommon.**minimalistui**.widget.message.MessageContentHolder

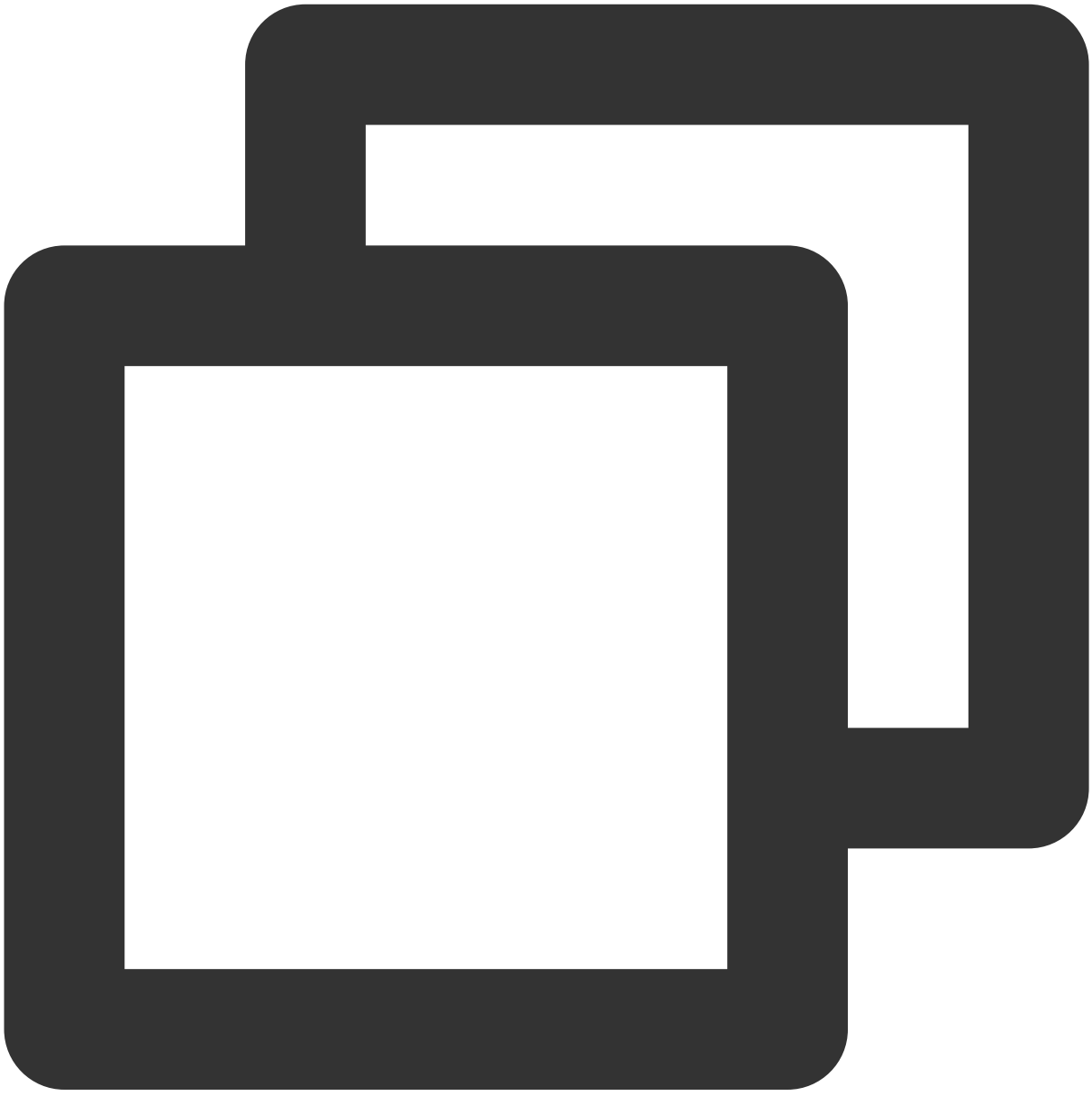
2. 示例代码如下：



```
public class CustomLinkMessageHolder extends MessageContentHolder {
```

```
    public CustomLinkMessageHolder(View itemView) {
        super(itemView);
    }
}
```

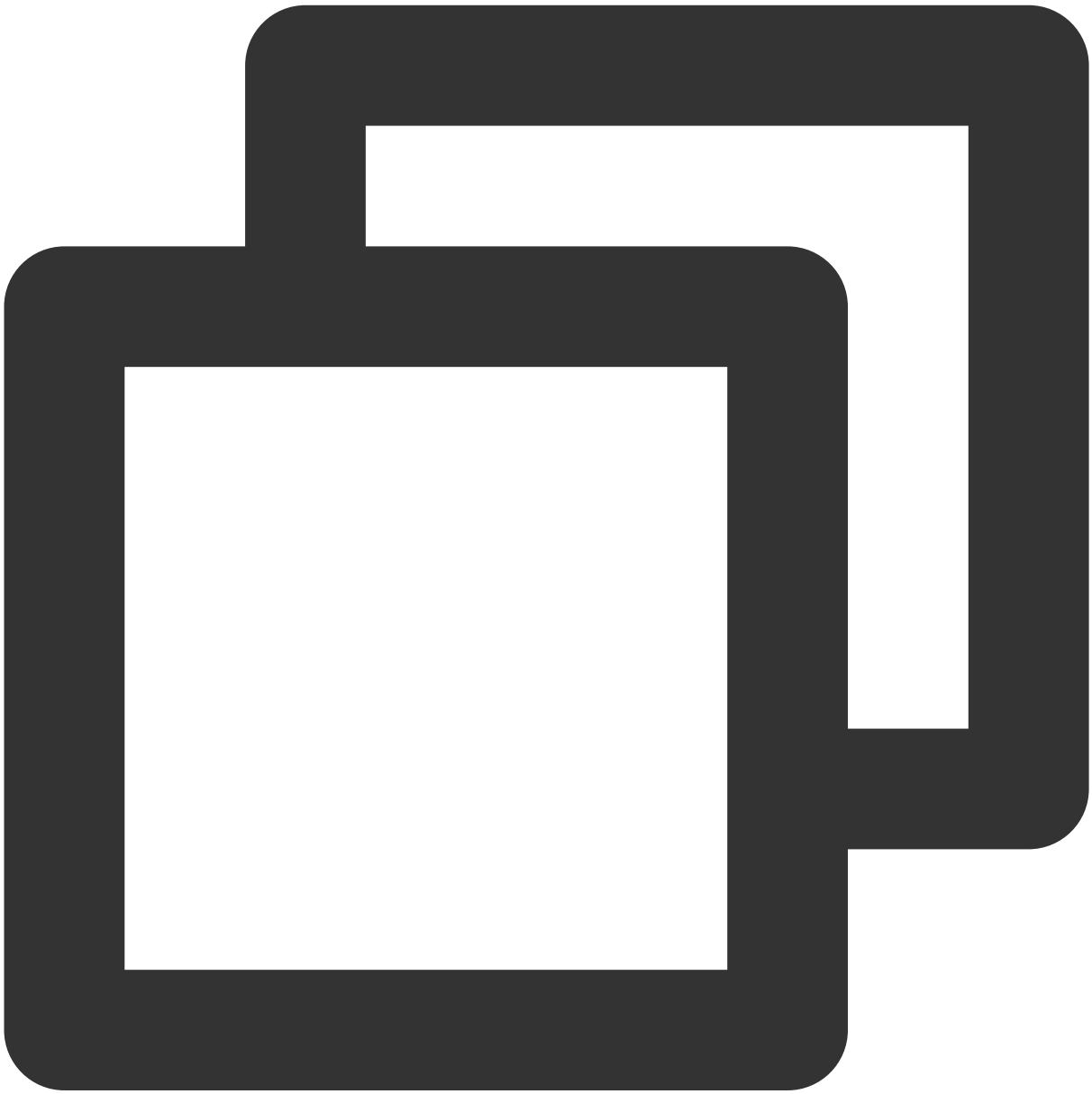
3. 重写 `CustomLinkMessageHolder` 的 `getVariableLayout` 方法，返回展示自定义消息的布局。
示例代码如下：



```
@Override
public int getVariableLayout() {
    return R.layout.test_custom_message_layout;
```

```
}
```

布局文件 `test_custom_message_layout` 如下：

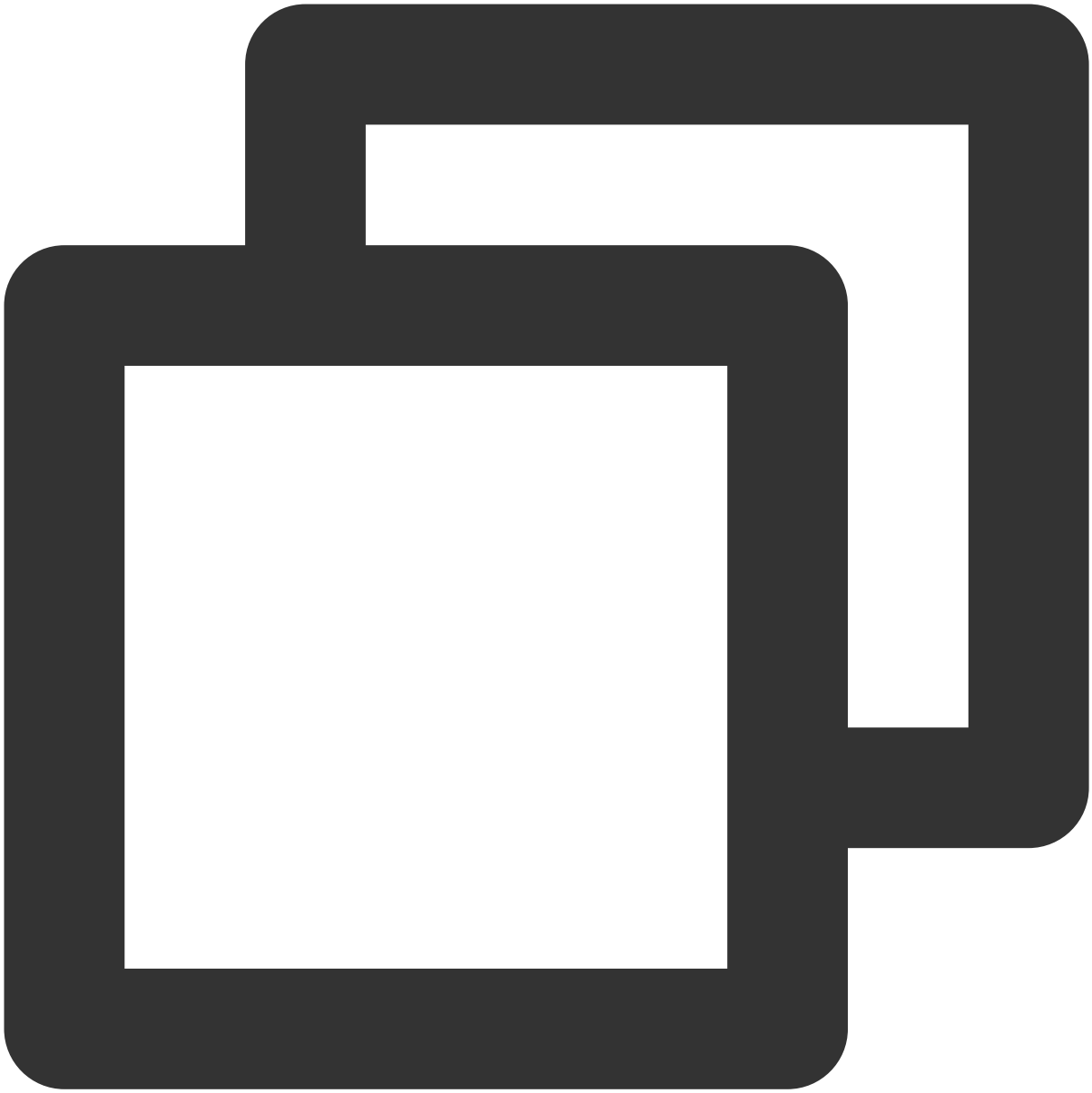


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/test_custom_message_tv"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="?attr/chat_self_custom_msg_text_color" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">
        <TextView
            android:id="@+id/link_tv"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textAlignment="viewEnd"
            android:text="@string/test_custom_message"
            android:textColor="?attr/chat_self_custom_msg_link_color" />
    </LinearLayout>
</LinearLayout>
```

4. 重写 `CustomLinkMessageHolder` 的 `layoutVariableViews` 方法，用于把自定义消息渲染到布局上，并添加自定义消息的点击事件。

示例代码如下：



```
@Override
public void layoutVariableViews(TUIMessageBean msg, int position) {
    // Custom message view implementation. Here we configure to display only the te

    TextView textView = itemView.findViewById(R.id.test_custom_message_tv);
    String text = "";
    String link = "";
    if (msg instanceof CustomLinkMessageBean) {
        text = ((CustomLinkMessageBean) msg).getText();
        link = ((CustomLinkMessageBean) msg).getLink();
    }
```

版权所有：腾讯云计算（北京）有限责任公司

```
        textView.setText(text);
        msgContentFrame.setClickable(true);
        String finalLink = link;
        msgContentFrame.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.setAction("android.intent.action.VIEW");
                Uri content_url = Uri.parse(finalLink);
                intent.setData(content_url);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                TUIChatService.getAppContext().startActivity(intent);
            }
        });
    }
```

注册自定义消息

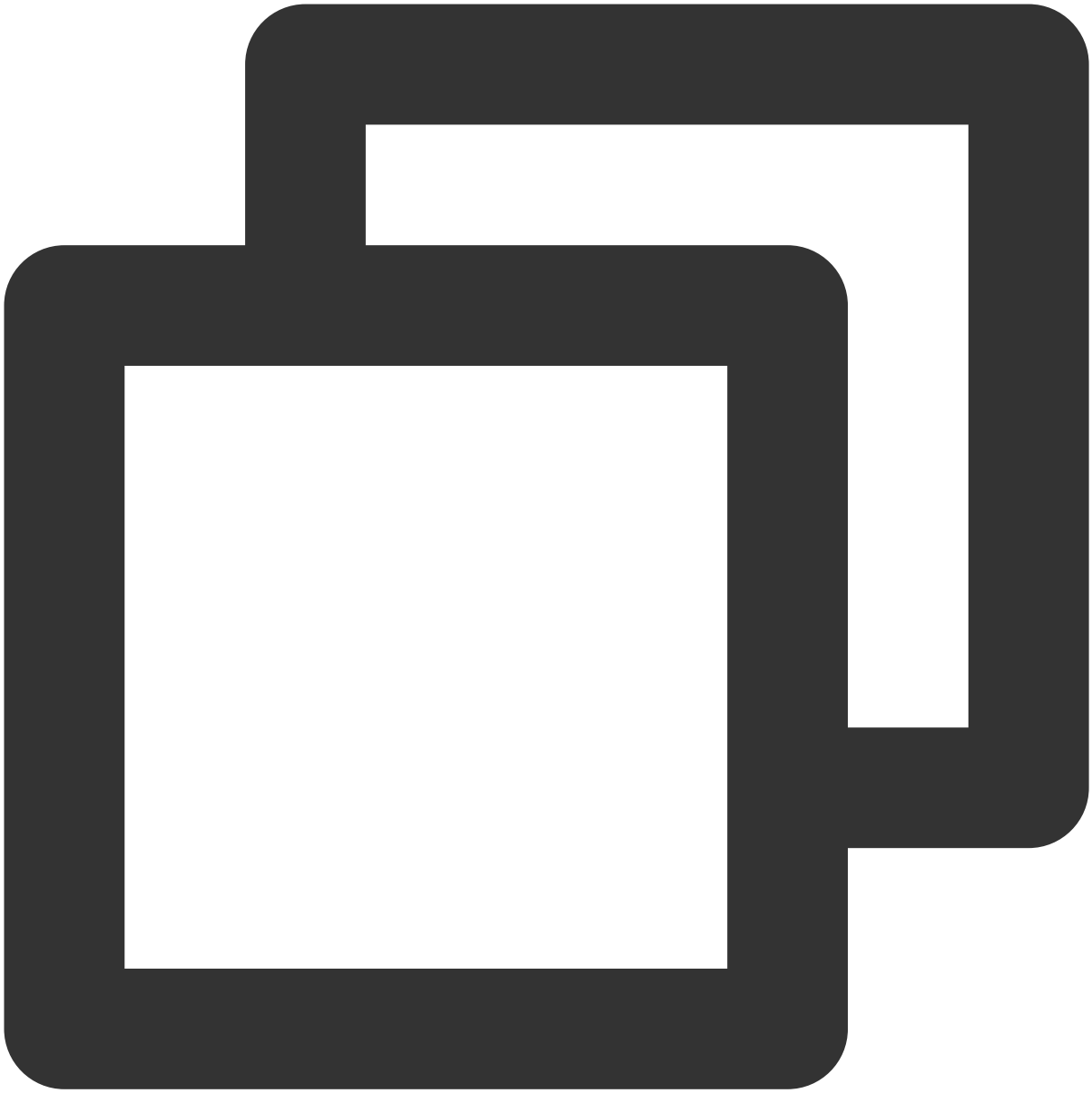
注意：

每一种自定义消息都必须有唯一的 businessID，区分大小写，不可跟其他自定义消息的 businessID 重复。TUIChat 需要根据此 businessID 找到对应的自定义消息。

新增自定义消息的 businessID 也不能和 TUIKit 内置自定义消息的 businessID 重复。

在 App 初始化时，调用 TUIChatConfigs.registerCustomMessage 接口，向 TUIChat 注册自定义消息。

示例代码如下：



```
// Self Definition message businessID (Note: Duplicates not allowed)
public static final String CUSTOM_LINK_MESSAGE_BUSINESS_ID = "text_link";
/** Register the self Definition message with TUIChat. The three parameters are
* @param businessID Self Definition message businessID
* @param messageBeanClass Self Definition message MessageBean type
* @param messageViewHolderClass Self Definition message MessageViewHolder type
*/
TUIChatConfigs.registerCustomMessage(CUSTOM_LINK_MESSAGE_BUSINESS_ID,
                                     CustomLinkMessageBean.class,
                                     CustomLinkMessageHolder.class);
```

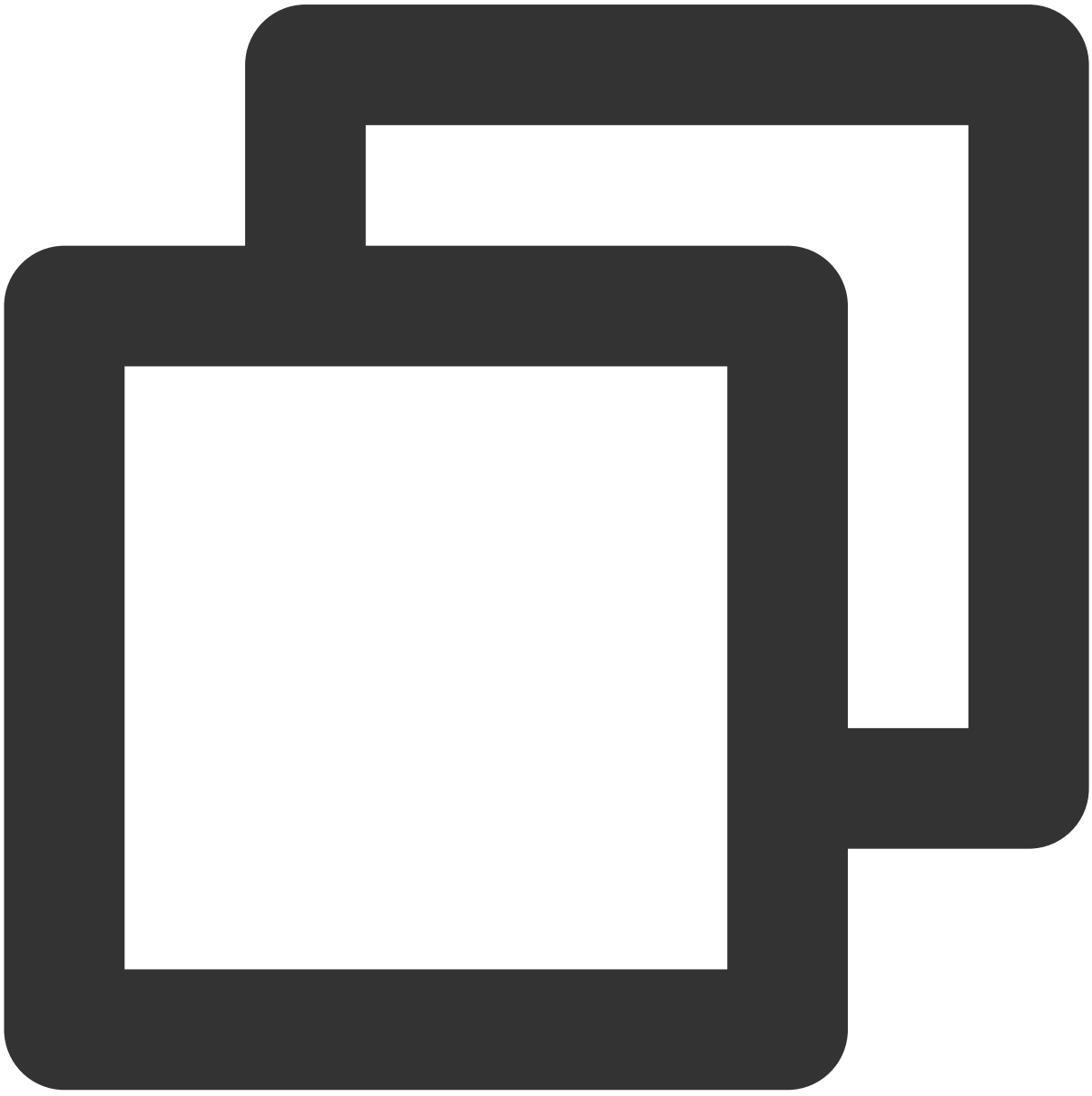
版权所有：腾讯云计算（北京）有限责任公司

除此之外，TUIChatConfigs 还提供了 registerCustomMessage 方法的另一个重载，支持注册简约版 UI 下的自定义消息，同时支持消息布局为空布局。详情可参见 TUIChatConfigs.java 文件。

发送自定义消息

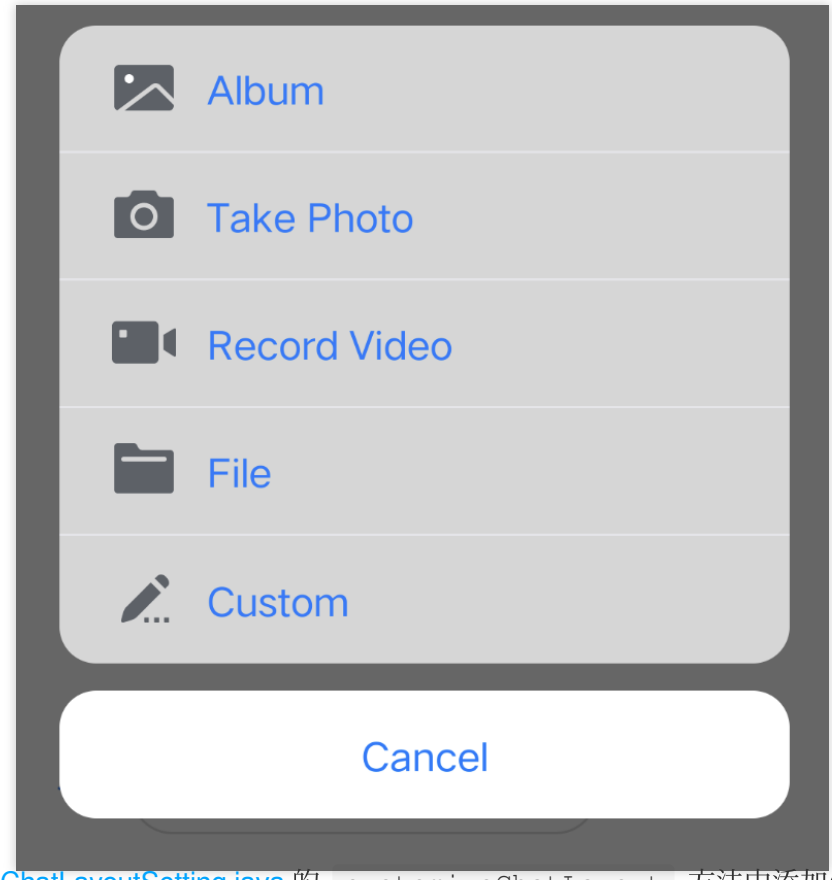
注意：

自定义消息内容必须为 JSON 格式。其中 "businessID" 字段为必填项。可根据业务需求添加其他字段，单条消息大小上限为 12KB。例如：



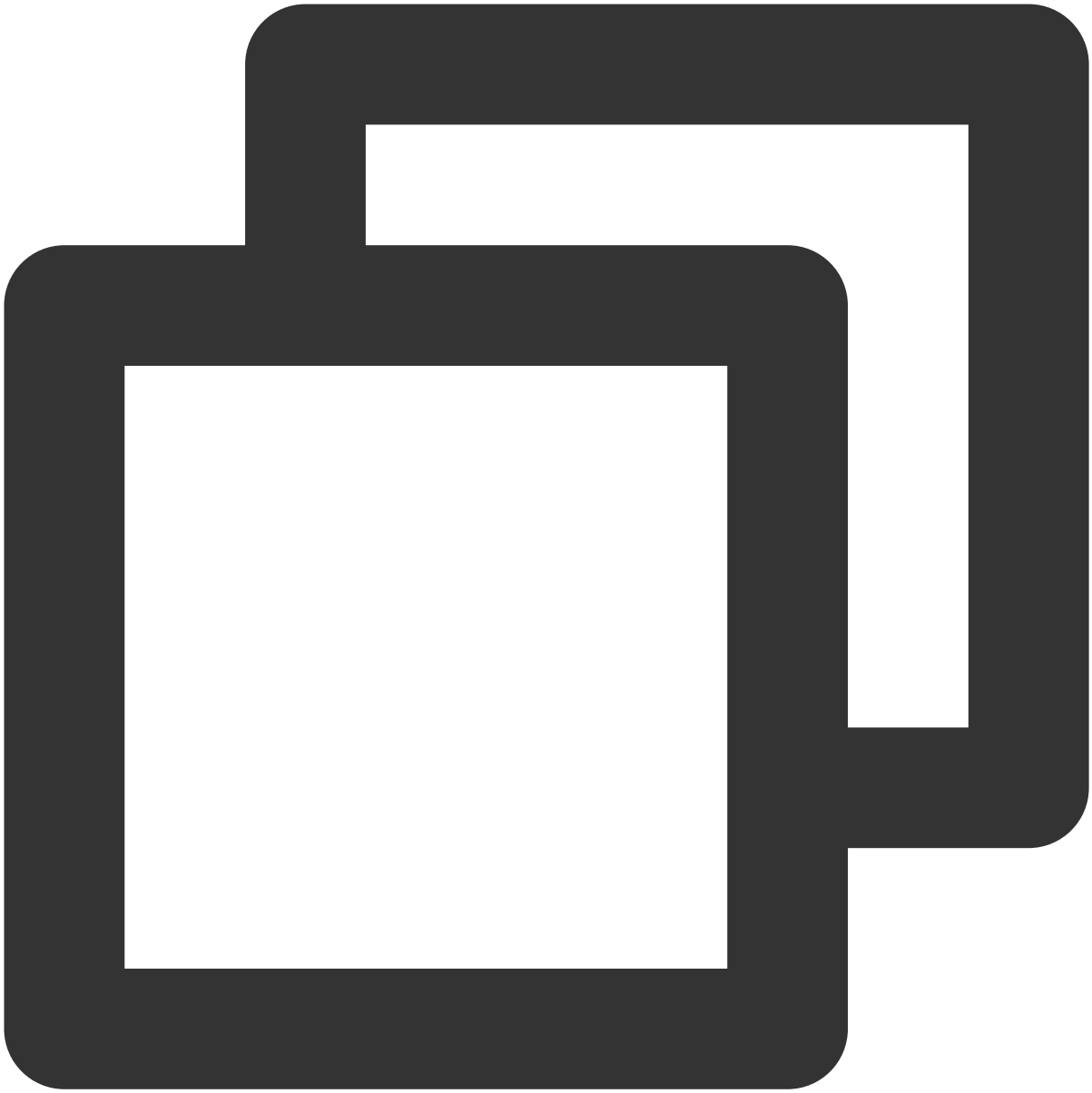
```
{
    "businessID":"text_link",
    "link":"https://trtc.io/products/chat",
    "text":"Welcome to Tencent Cloud Chat!"
}
```

如下图所示，自定义消息发送按钮主要由文本 `title` 和图片 `icon` 组成：



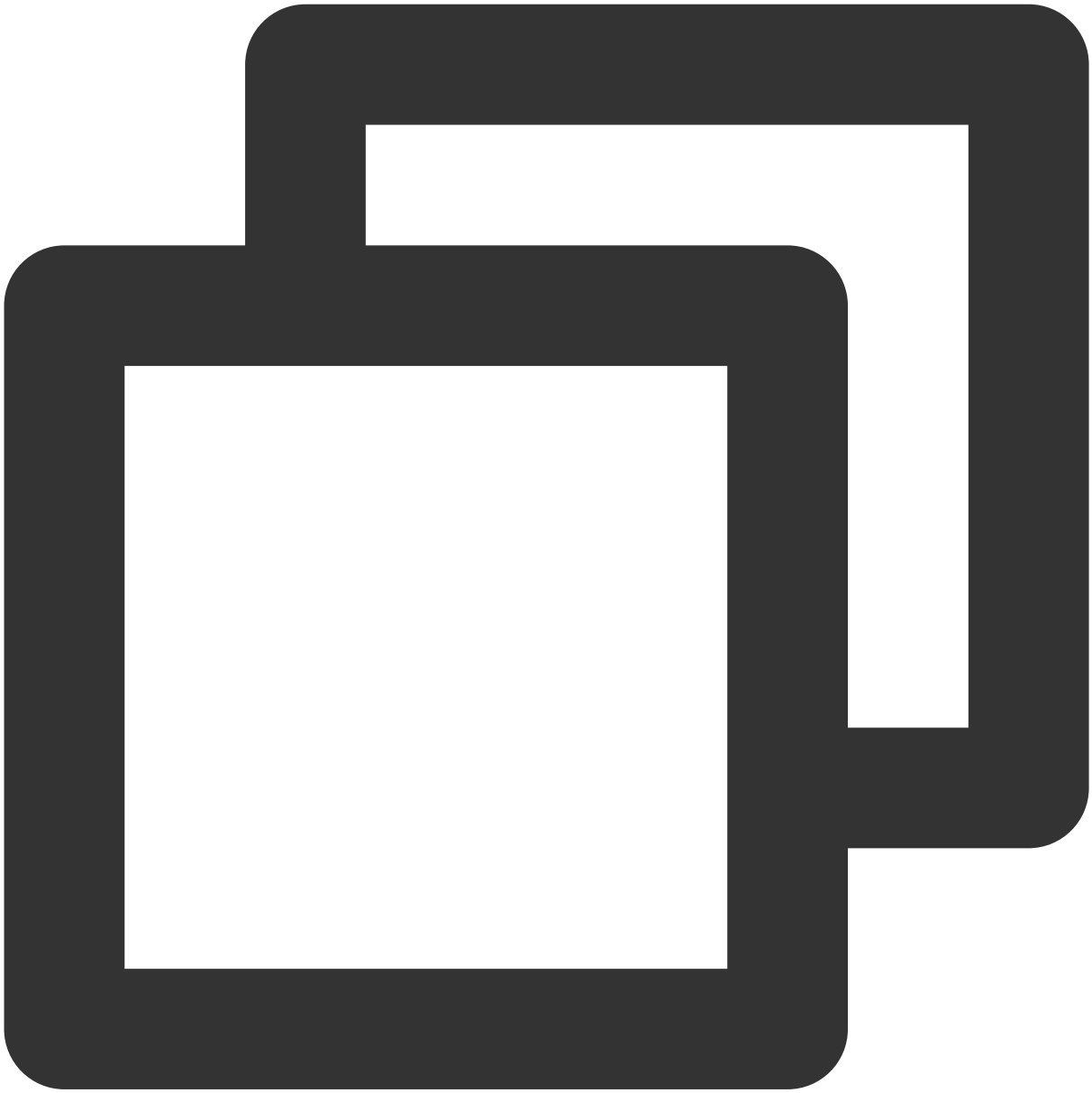
1. 在 ChatLayoutSetting.java 的 `customizeChatLayout` 方法中添加代码，添加自定义消息发送按钮。

示例代码如下：



```
InputMoreActionUnit unit = new InputMoreActionUnit() {};
unit.setIconResId(R.drawable.custom);
unit.setName("Custom");
unit.setActionId(CustomHelloMessage.CUSTOM_HELLO_ACTION_ID);
unit.setPriority(10);
inputView.addAction(unit);
```

2. 为上面步骤创建的自定义消息发送按钮设置点击监听，点击消息发送按钮后就可以创建一条自定义消息发送。
自定义消息是一段 JSON 数据，在 JSON 中定义 `businessID` 字段来唯一标识这条消息类型。
示例代码如下：



```
unit.setOnClickListener(unit.new OnActionClickListener() {
    @Override
    public void onClick() {
        Gson gson = new Gson();
        CustomHelloMessage customHelloMessage = new CustomHelloMessage();
        customHelloMessage.businessID = "text_link";
        customHelloMessage.text = "Welcome to Tencent Cloud Chat!";
        customHelloMessage.link = "https://trtc.io/products/chat";
        String data = gson.toJson(customHelloMessage);
        TUIMessageBean info = ChatMessageBuilder.buildCustomMessage(data, customHel
        layout.sendMessage(info, false);
```

```
    }
});
```

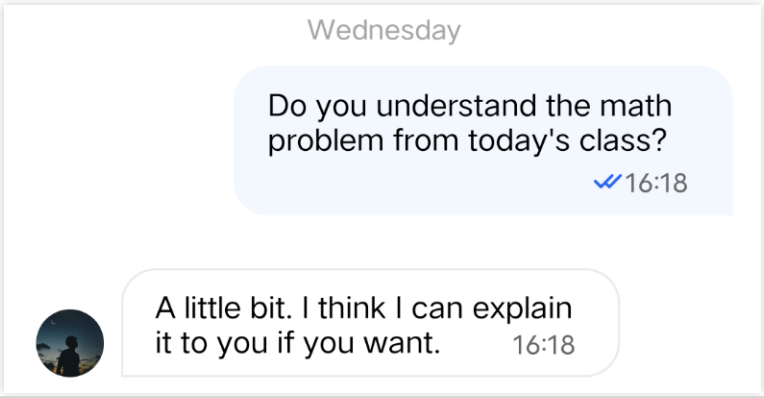
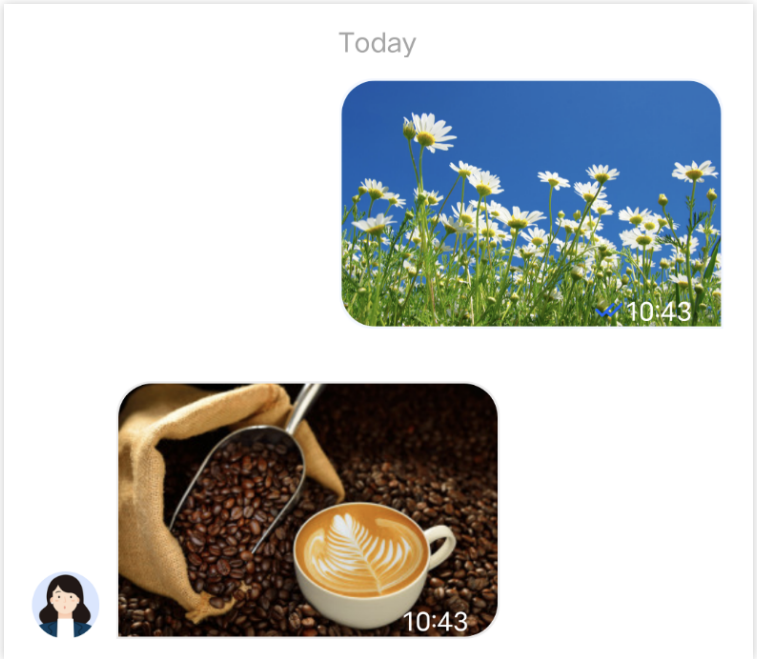
版权所有：腾讯云计算（北京）有限责任公司

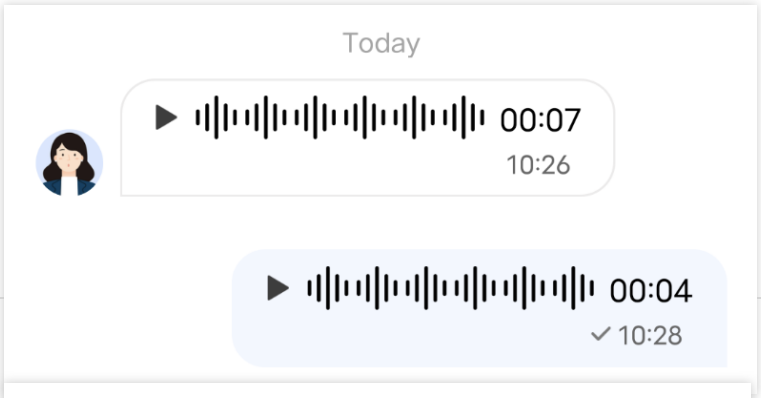
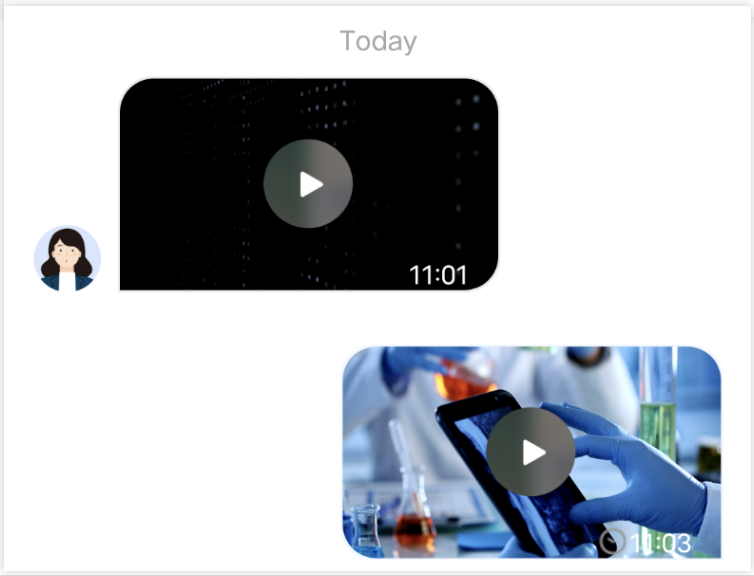
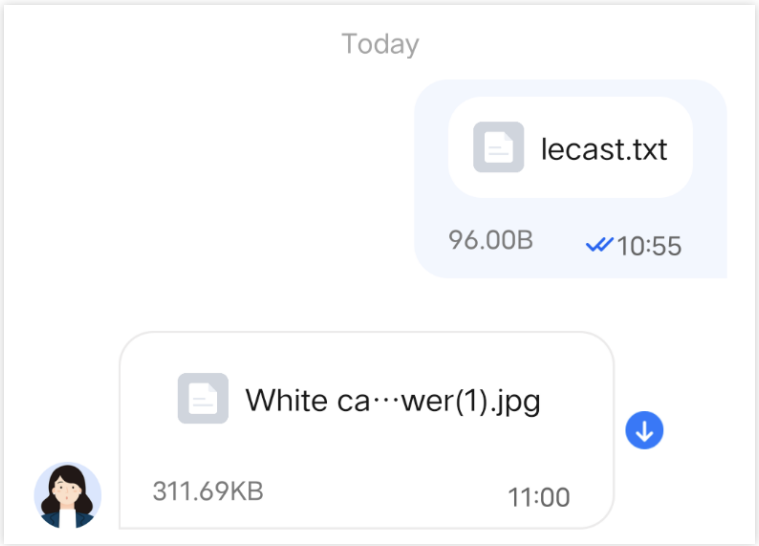
iOS

最近更新时间：2024-06-21 17:11:46

TUIKit 默认实现了文本、图片、语音、视频、文件等基本消息类型的发送和展示，如果这些消息类型满足不了您的需求，您可以新增自定义消息类型。

基本消息类型

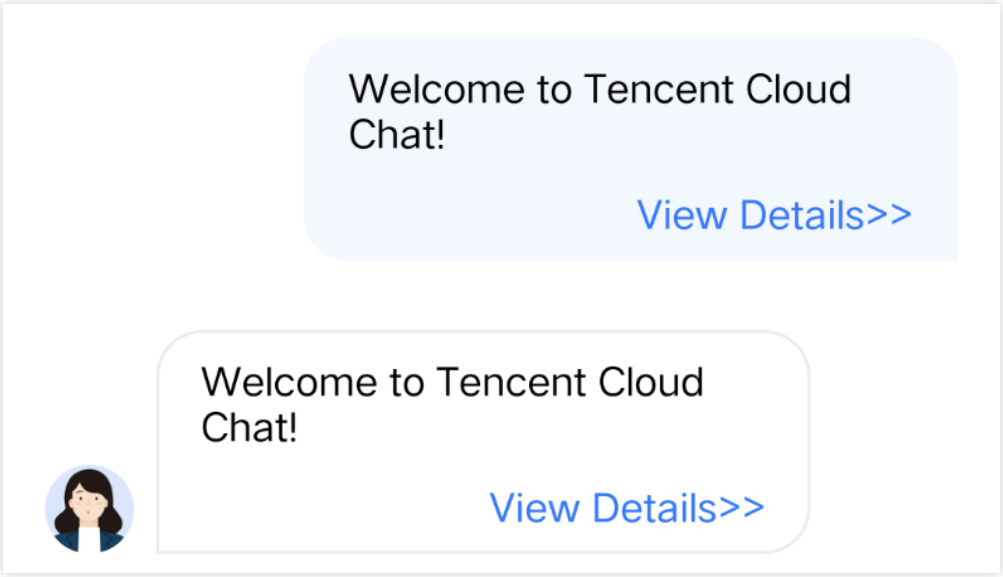
消息类型	显示效果图
文本类消息	
图片类消息	
语音类消息	

	
视频类消息	
文件类消息	

自定义消息

如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。下文以发送一条可跳转至浏览器的超文本作为自定义消息为例，帮助您快速了解实现流程。

TUIKit 内置的自定义消息样式如下图所示：



注意：

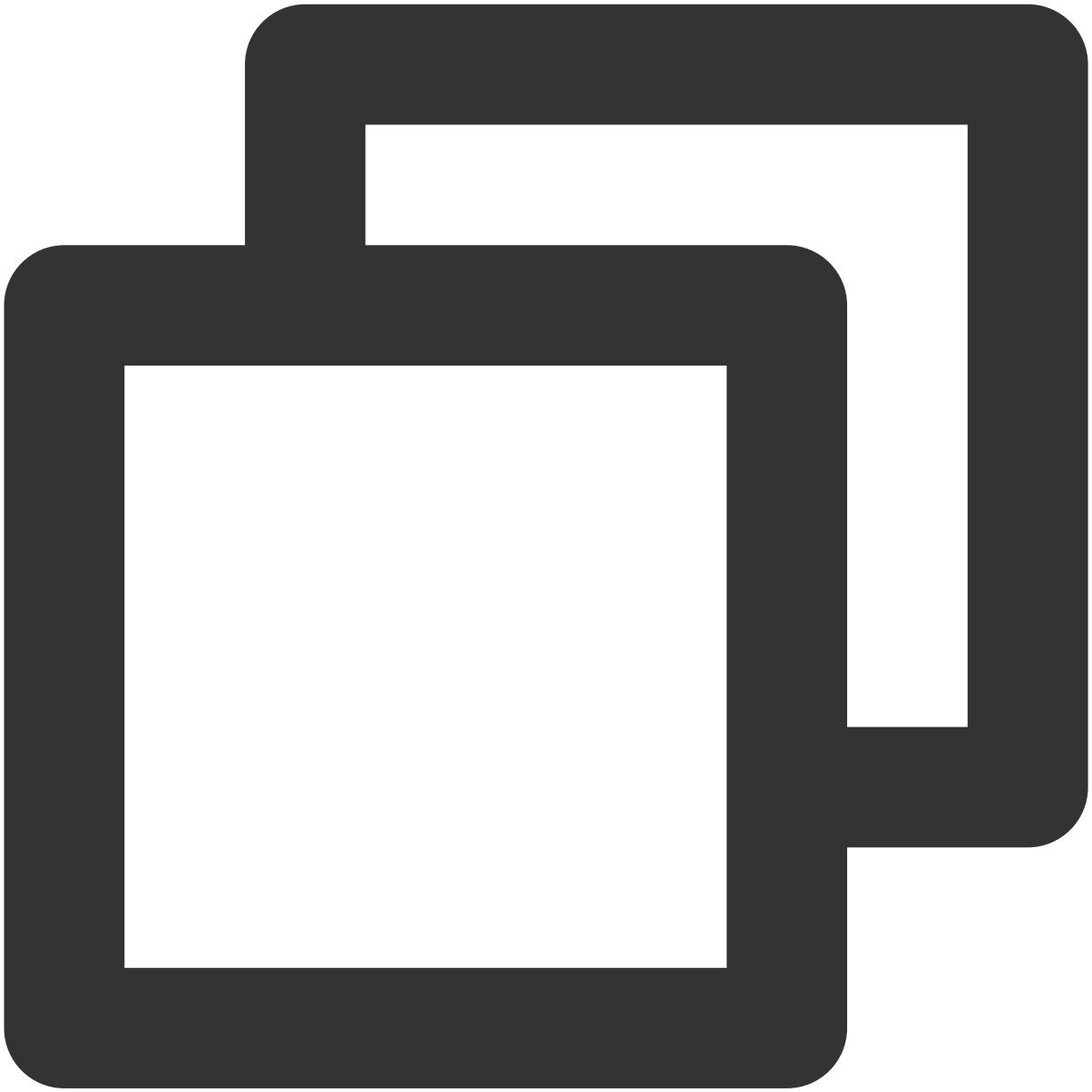
TUIKit 在 7.4.4643 版本重新设计了一套自定义消息注册机制，新旧方案变动较大，但可以支持不同的 UI 样式，建议您升级到 7.4.4643 版本。本文将以 7.4.4643 版本为例讲解。

展示自定义消息

您可以在 TUIMessageBaseDataProvider.m 的 `onRecvNewMessage` 函数内接收自定义消息。
收到的自定义消息最终会以 `Cell` 的形式展示在消息列表中， `Cell` 绘制所需的数据我们称之为 `CellData` 。
下面我们分步骤讲解下如何展示自定义消息。

创建自定义 CellData

1. 在 `TUIChat/BaseCellData/Custom` 文件夹下新建 TUILinkCellData.h 和 TUILinkCellData.m 文件，继承自 `TUIMessageCellData` ，用于存储显示的文字和跳转的链接。
示例代码如下：



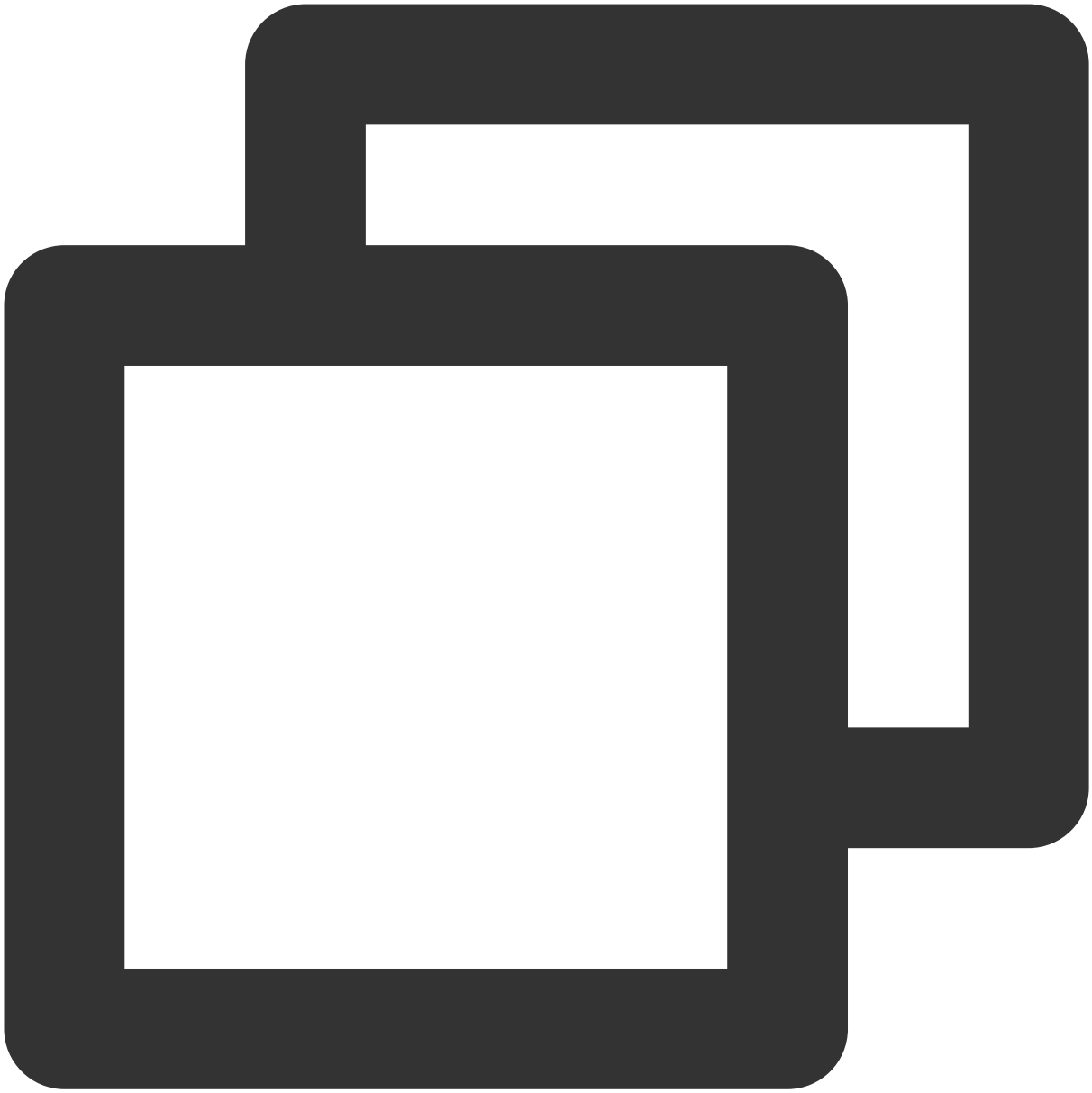
```
@interface TUILinkCellData : TUIMessageCellData

@property NSString *text;
@property NSString *link;

@end
```

2. 重写父类的 `getCellData:` 方法。用于把 `V2TIMMessage` 转换成消息列表 `Cell` 的绘制数据 `TUILinkCellData` 。

示例代码如下：

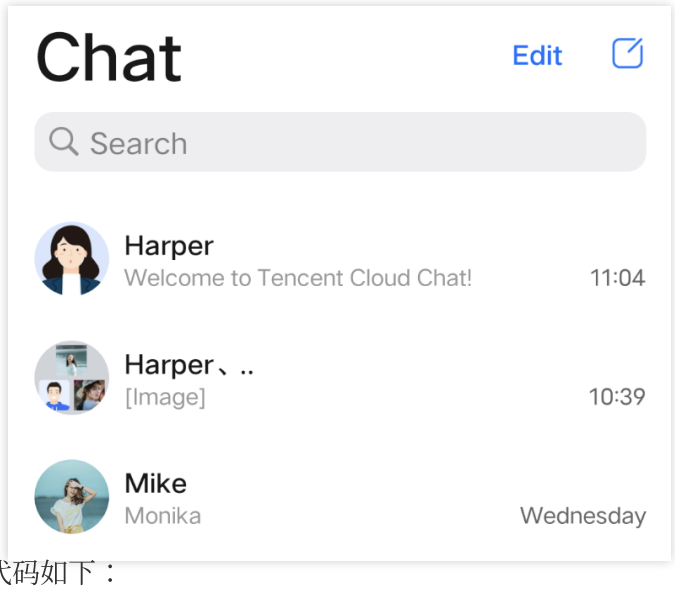


```
@implementation TUILinkCellData
+ (TUIMessageCellData *)getCellData:(V2TIMMessage *)message {
        NSDictionary *param = [NSJSONSerialization JSONObjectWithData:message.custo
        TUILinkCellData *cellData = [[TUILinkCellData alloc] initWithDirection:mess
        cellData.innerMessage = message;
        cellData.msgID = message.msgID;
        cellData.text = param[@"text"];
        cellData.link = param[@"link"];
        cellData.avatarUrl = [NSURL URLWithString:message.faceURL];
        return cellData;
 }
```

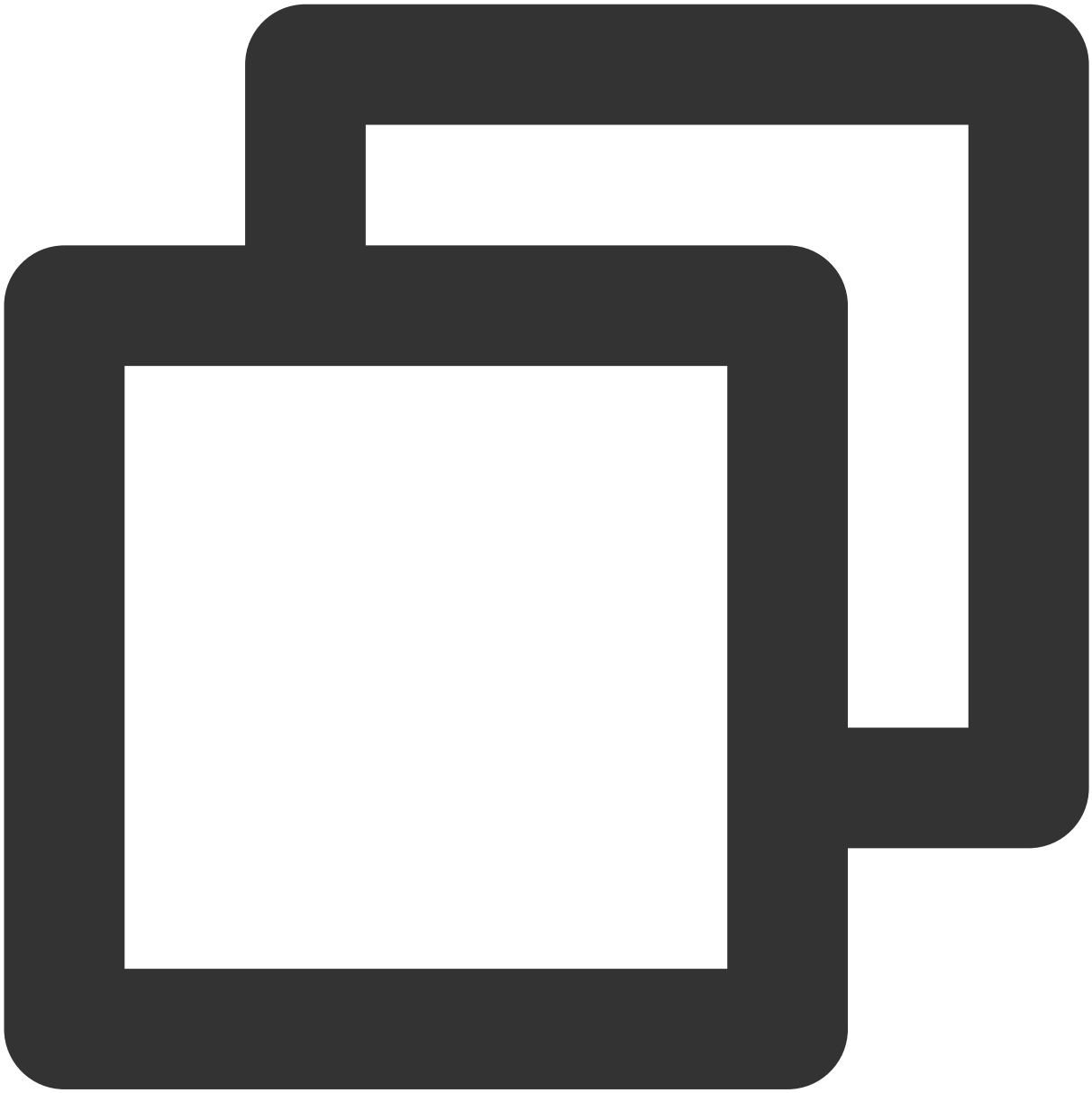
```
@end
```

3. 重写父类的 `getDisplayString:` 方法。用于把 `V2TIMMessage` 转换成会话列表 `lastMsg` 的展示文本信息。

会话列表 `lastMsg` 展示文本指的是当用户停留在会话列表，每个会话 cell 会显示当前会话最后一条消息。如下图所示：



示例代码如下：

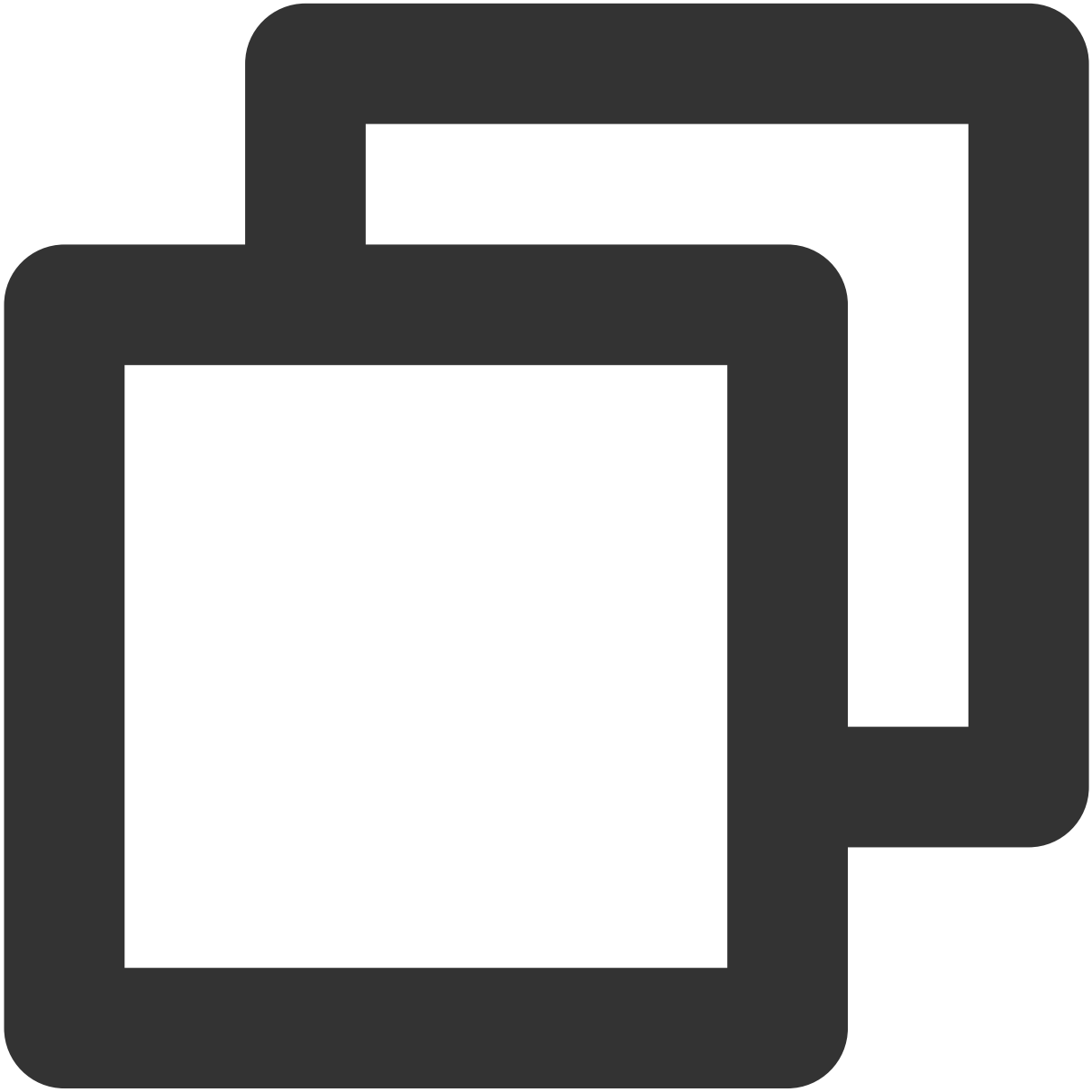


```
@implementation TUILinkCellData
+ (NSString *)getDisplayString:(V2TIMMessage *)message {
    NSDictionary *param = [NSJSONSerialization JSONObjectWithData:message.customEle
    return param[@"text"];
}
@end
```

创建自定义 Cell

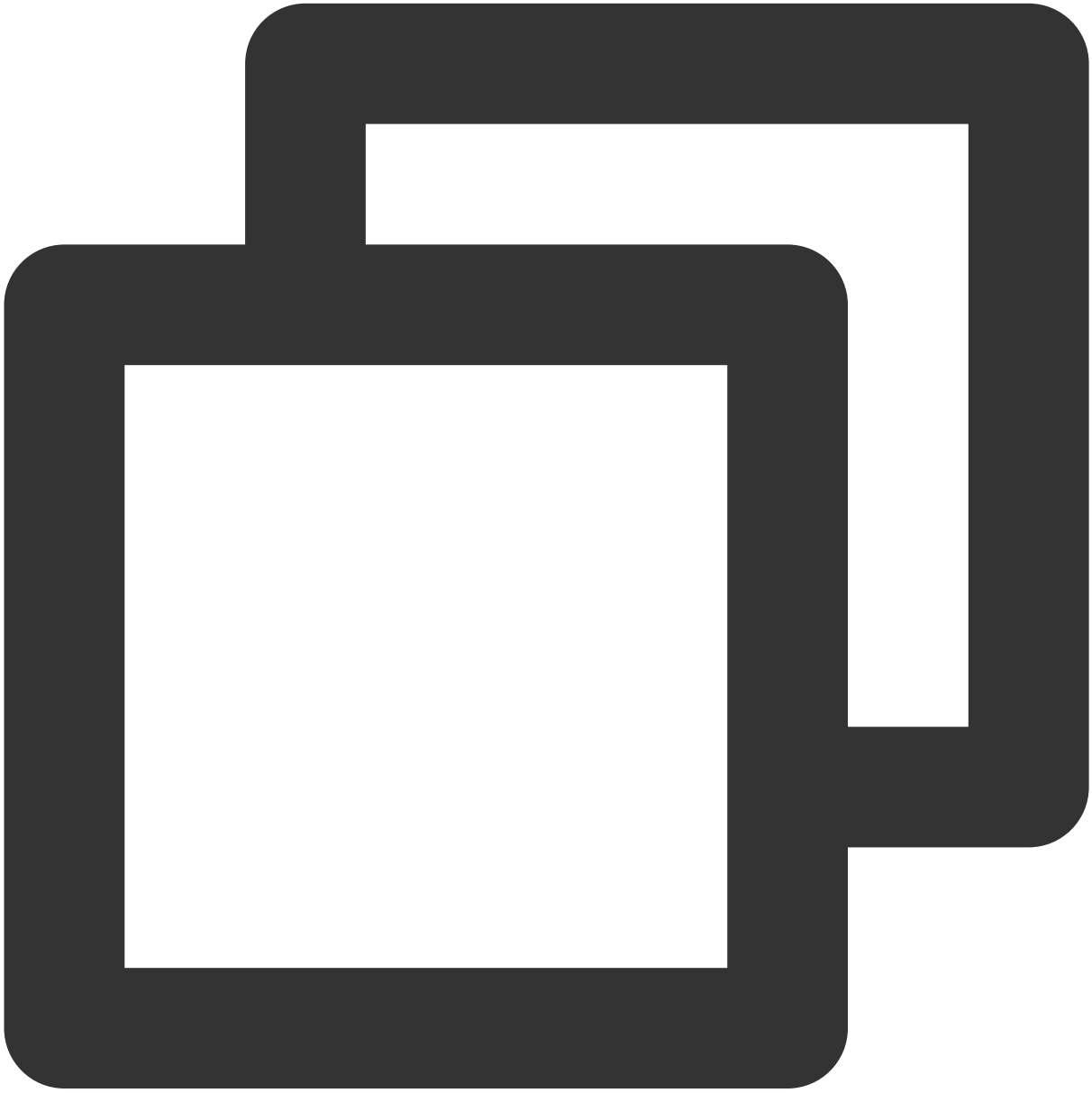
1. 在 `TUIChat/UI_Minimalist/Cell/Custom` 文件夹下新建 TUILinkCell_Minimalist.h 和 TUILinkCell_Minimalist.m 文件，继承自 `TUIBubbleMessageCell_Minimalist` ，用于绘制 `TUILinkCellData` 数据。

示例代码如下：



```
@interface TUILinkCell_Minimalist : TUIBubbleMessageCell_Minimalist
@property UILabel *myTextLabel;  // Display text
@property UILabel *myLinkLabel;  // Link redirection text
- (void)fillWithData:(TUILinkCellData *)data;// Draw UI
@end
```

2. 重写父类 `initWithStyle:reuseIdentifier:` 方法，创建 `myTextLabel` 和 `myLinkLabel` 文本展示对象，并添加至 `container` 容器。
示例代码如下：

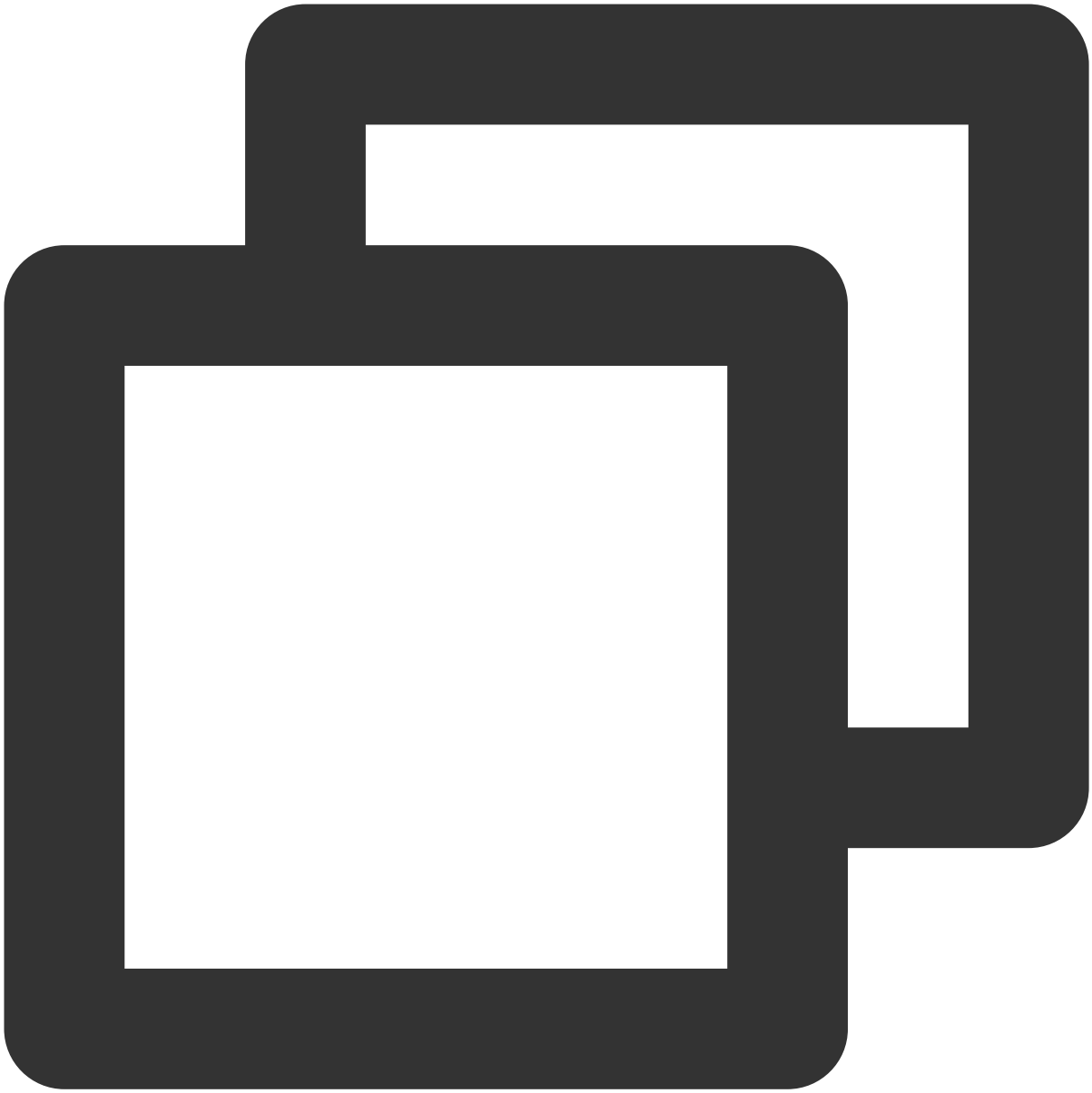


```
@implementation TUILinkCell_Minimalist
// Initialize the control
- (instancetype)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString
{
  self = [super initWithStyle:style reuseIdentifier:reuseIdentifier];
  if (self) {
  self.myTextLabel = [[UILabel alloc] init];
```

```
    [self.container addSubview:self.myTextLabel];
    self.myLinkLabel = [[UILabel alloc] init];
    self.myLinkLabel.text = @"View details>>";
    [self.container addSubview:_myLinkLabel];
    }
    return self;
}
@end
```

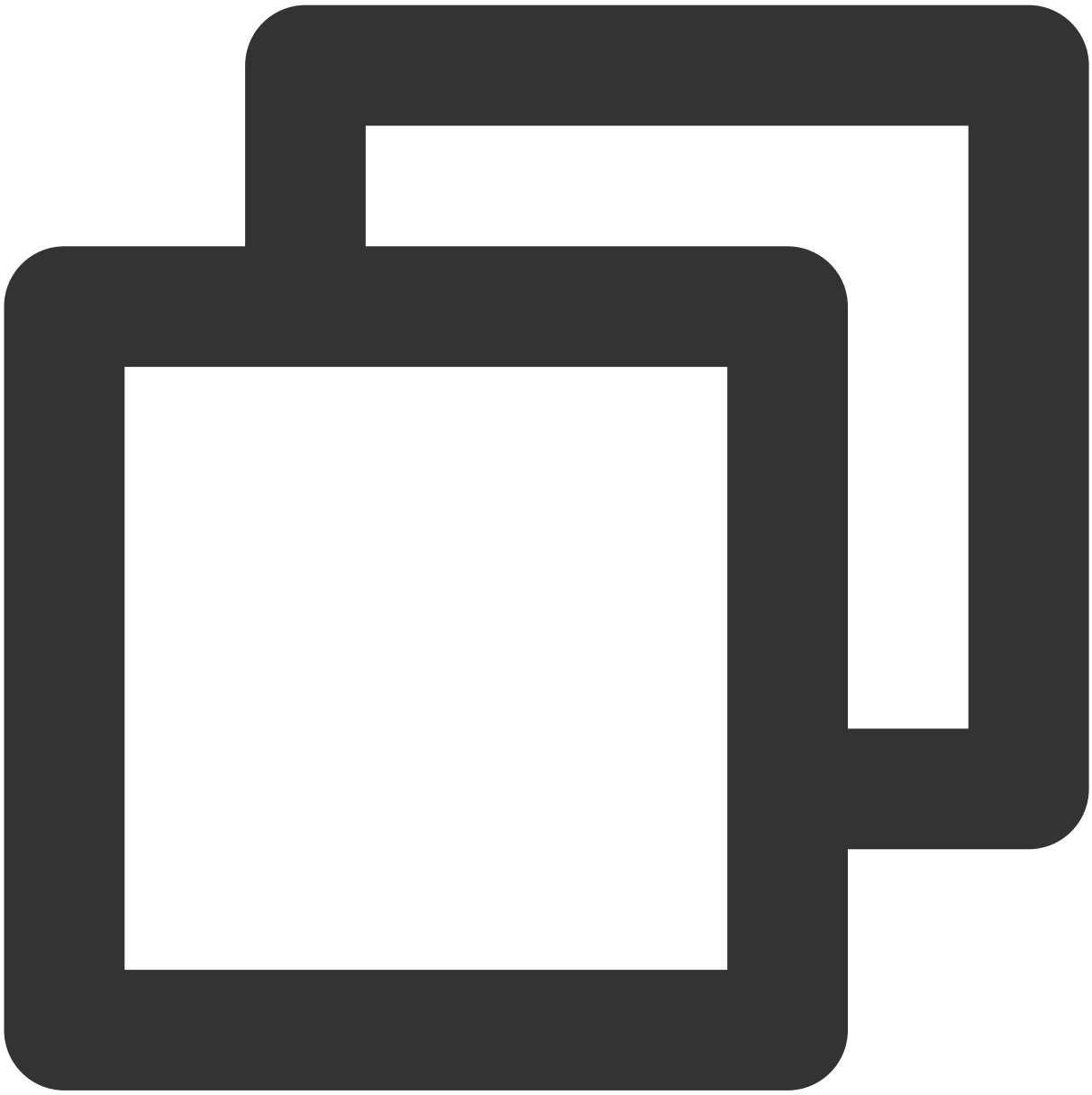
3. 重写父类 `fillWithData:` 方法，在 `TUILinkCell_Minimalist` 中自定义展示 `TUILinkCellData` 数据。

示例代码如下：



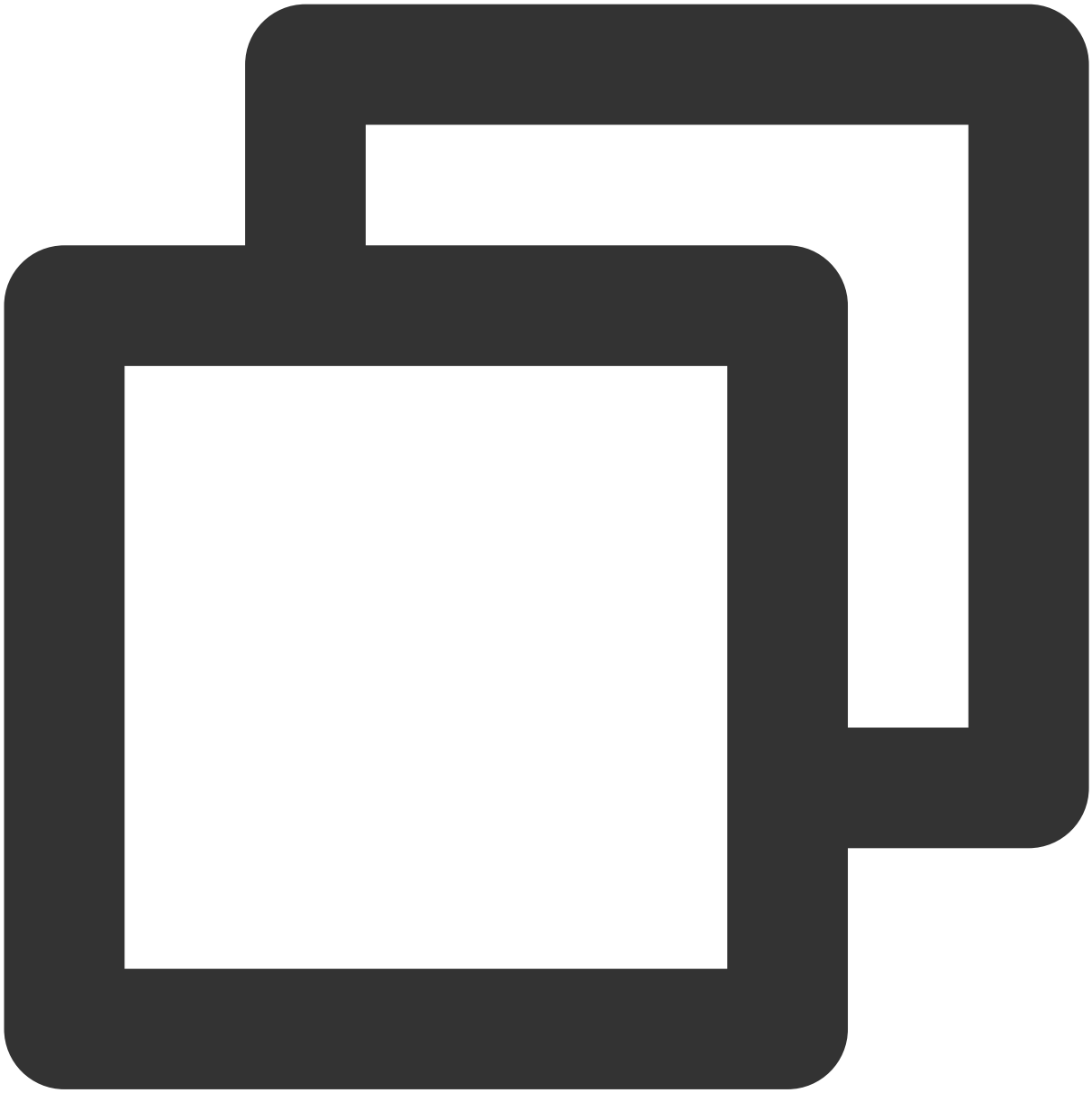
```
@implementation TUILinkCell_Minimalist
// Draw the cell based on cellData
- (void)fillWithData:(TUILinkCellData *)data;
{
  [super fillWithData:data];
  self.myTextLabel.text = data.text;
}
@end
```

4. 重写父类 `layoutSubviews` 方法，自定义控件的布局。
示例代码如下：



```
// Set the control coordinates
- (void)layoutSubviews
{
  [super layoutSubviews];
  self.myTextLabel.mm_top(10).mm_left(10).mm_flexToRight(10).mm_flexToBottom(50);
  self.myLinkLabel.mm_sizeToFit().mm_left(10).mm_bottom(10);
}
@end
```

5. 重写父类的 getContentSize： 方法，用于计算 cellData 内容所占绘制区域的大小。
示例代码如下：



```
+ (CGSize)getContentSize:(TUIMessageCellData *)data {
 NSAssert([data isKindOfClass:TUILinkCellData.class], @"data must be kind of TUILin
 TUILinkCellData *linkCellData = (TUILinkCellData *)data;

 CGFloat textMaxWidth = 245.f;
 CGRect rect = [linkCellData.text boundingRectWithSize:CGSizeMake(textMaxWidth, MAX
                                                 options:NSStringDrawingUsesLineFragm
                                              attributes:@{NSFontAttributeName : [UIF
```

```
                                                    context:nil];
    CGSize size = CGSizeMake(textMaxWidth + 15, rect.size.height + 56);
    return size;
}
```

将您的自定义 Cell 和 CellData 注册进 TUIChat

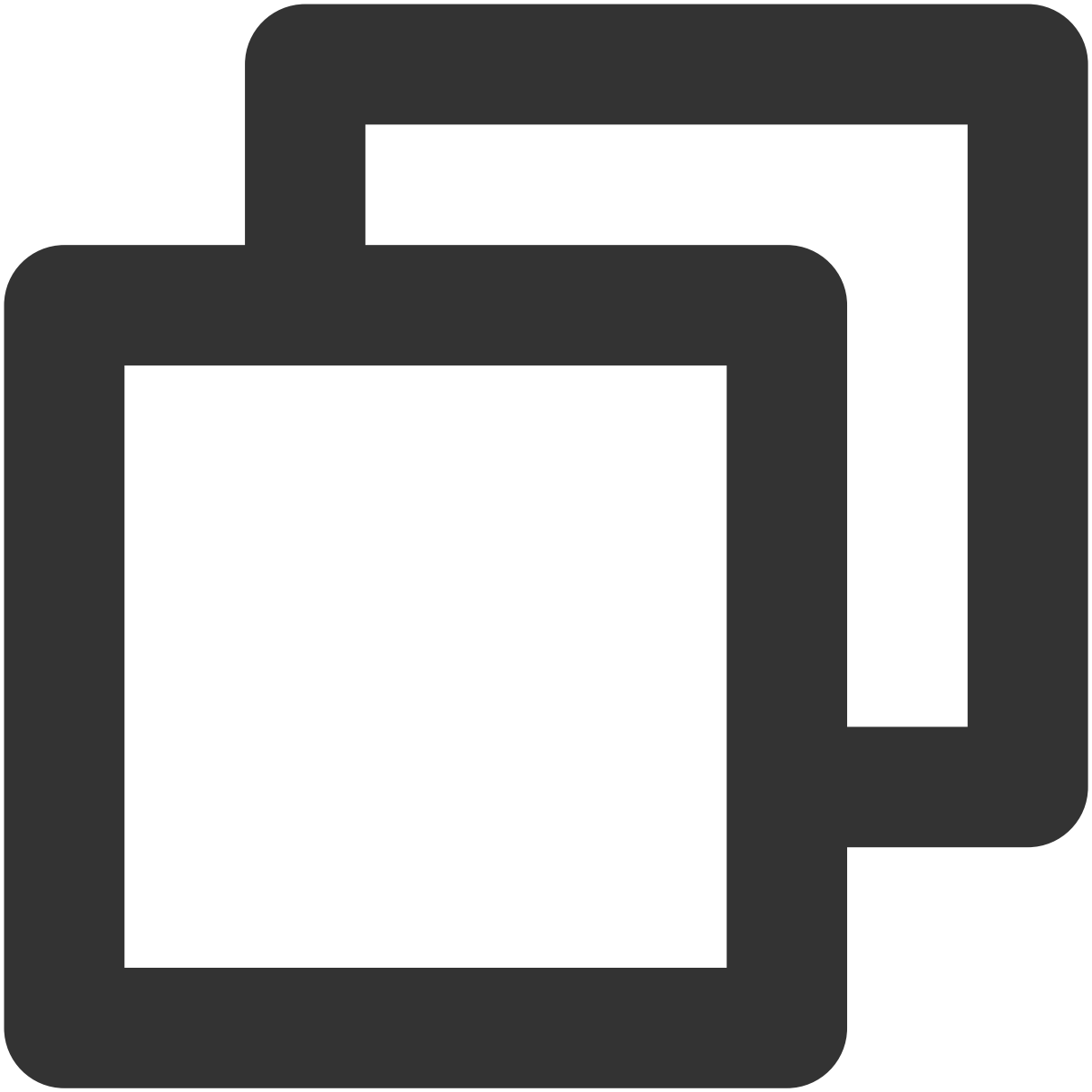
注意：

每一种自定义消息都必须有唯一的 businessID，区分大小写，不可跟其他自定义消息的 businessID 重复。TUIChat 需要根据此 businessID 找到对应的自定义消息。

新增自定义消息的 businessID 也不能和 TUIKit 内置自定义消息的 businessID 重复。

方式一：当您采用 DevelopPods 源码集成时，并想要在组件内部直接修改需求，您可以在 TUIChat 组件内部按照以下操作直接进行修改。

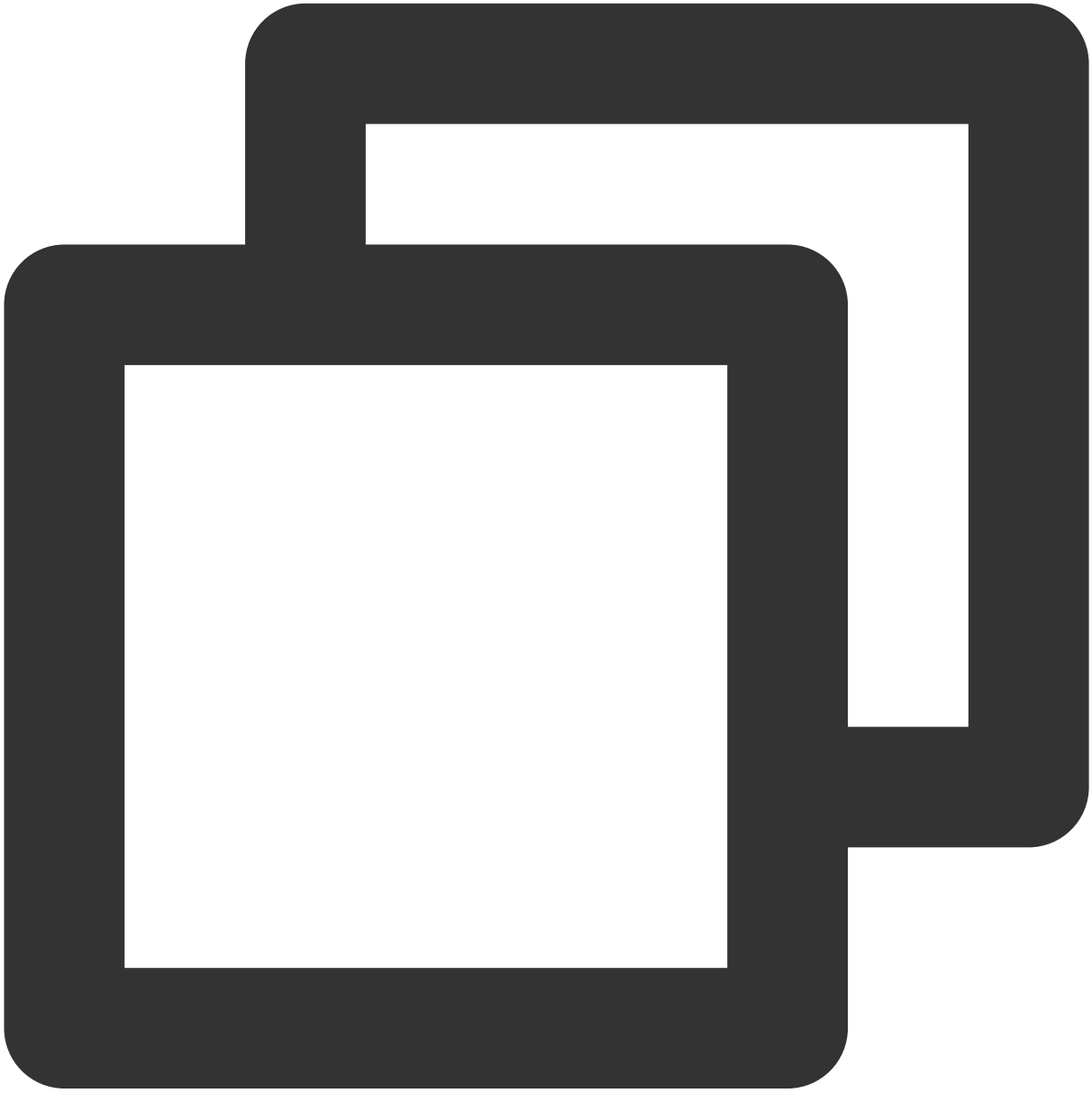
在 TUIMessageCellConfig_Minimalist.m 的 registerExternalCustomMessageInfo 中注册您自己的自定义 Cell：



```
@implementation TUIMessageCellConfig_Minimalist (CustomMessageRegister)
+ (void)registerExternalCustomMessageInfo {
    // Insert your own custom message UI here, your businessID can not be same with
    // Example:
    [self registerCustomMessageCell:@"TUILinkCell_Minimalist" messageCellData:@"TUI
}
```

方式二：通过 Pod 集成 TUIChat。

在 App 初始化时，您也可以通过 TUIChatConfig.h 的 `registerCustomMessage` 函数里主动注册 `cell` 和 `cellData` 信息。
示例代码如下：



```
// The custom message's businessID (note that there should not be a duplication)
#define BussinessID_TextLink @"text_link"

/** Register custom message's to TUIChat. The three parameters are
* @param businessID Custom message's businessID
* @param messagellClass Custom message's NSString type
* @param messageCellDataClassName Custom message's NSString type
```

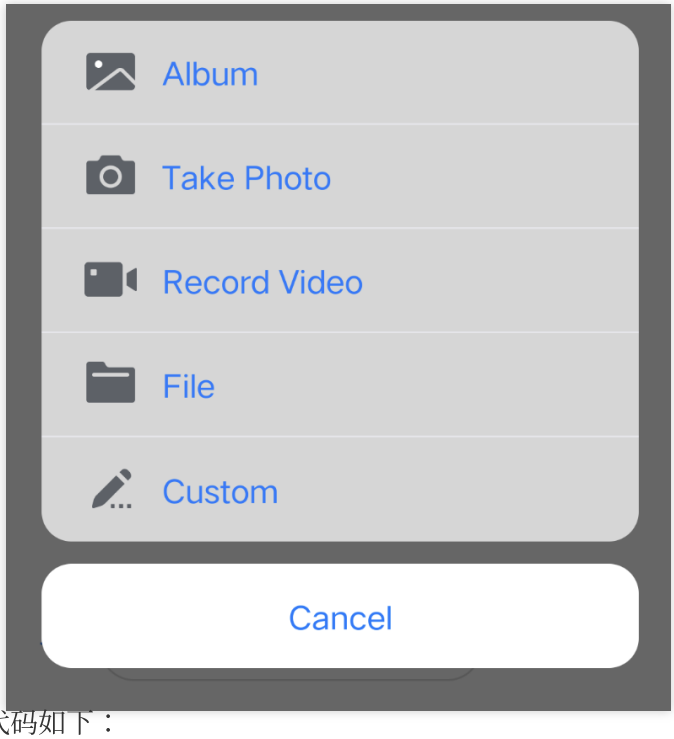
```
 * @param styleType UI style corresponding to this custom message, for example TUICh
 */
- (void)registerCustomMessageCell {

    [TUIChatConfig.defaultConfig registerCustomMessage:BussinessID_TextLink
                                    messageCellClassName:@"TUILinkCell_Minimalist"
                               messageCellDataClassName:@"TUILinkCellData"
                                              styleType:TUIChatRegisterCustomMessage
     ];
}
```

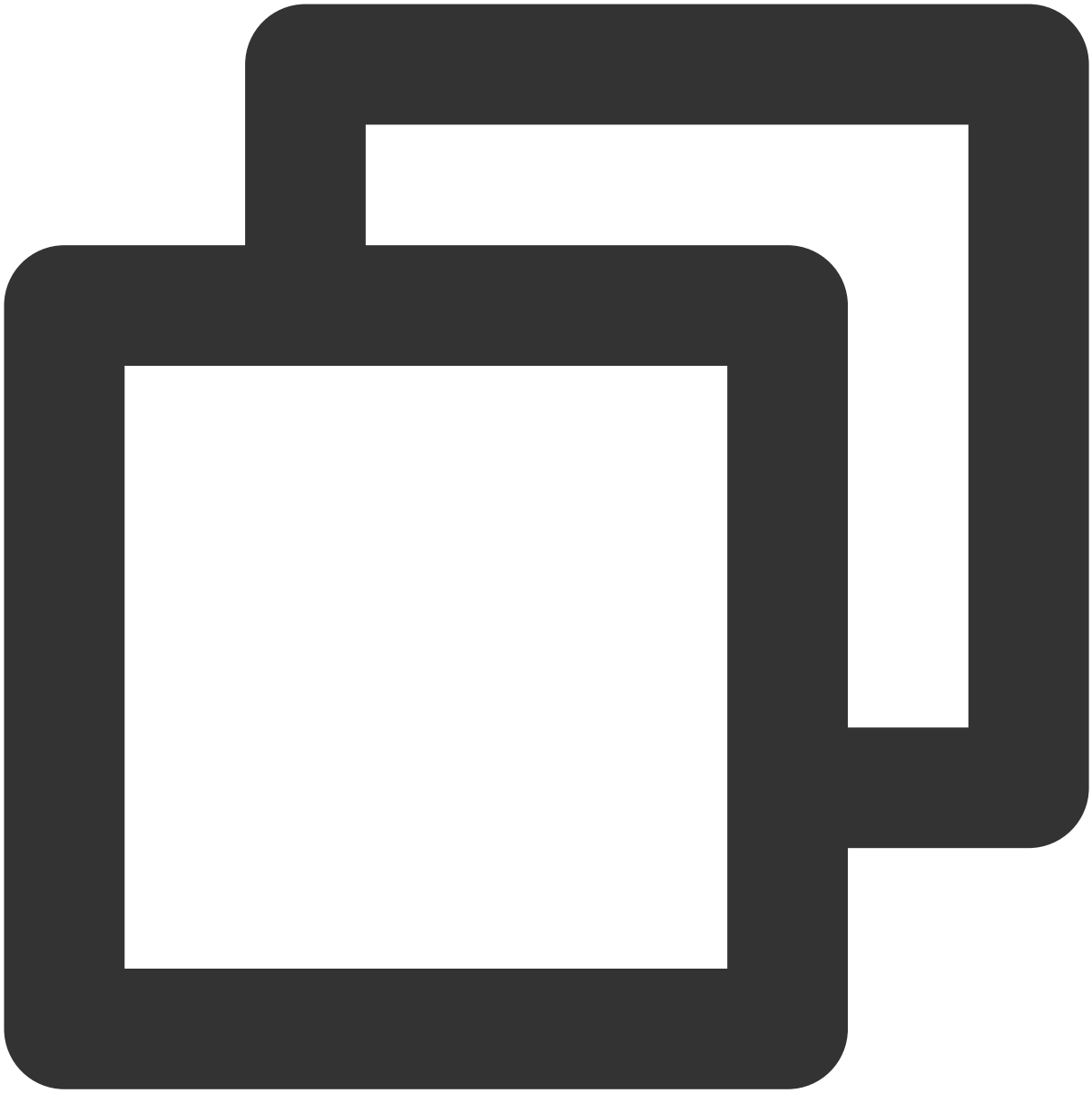
发送自定义消息

如下图所示，自定义消息发送按钮主要由文本 `title` 和图片 `leftMark` 组成。您可以通过在 TUIChatDataProvider 的 `customInputMoreActionItemList` 属性中新增 `TUICustomActionSheetItem` 对象来添加自定义按钮。

您可以通过设置 `TUICustomActionSheetItem` 的 `title` 和 `leftMark` 属性来自定义您想展示的文字和图片信息；如果您想调整按钮的展示顺序，可以设置 `priority` 属性，其中 `priority` 值越大按钮越靠前；您也可以设置 `actionHandler` 来响应该按钮的点击事件，实现自己的业务逻辑。



示例代码如下：



```
@implementation TUIChatDataProvider
- (NSArray<TUICustomActionSheetItem *> *)customInputMoreActionItemList {
    if (_customInputMoreActionItemList == nil) {
        NSMutableArray *arrayM = [NSMutableArray array];
        if (TUIChatConfig.defaultConfig.enableWelcomeCustomMessage) {
            __weak typeof(self) weakSelf = self;
            TUICustomActionSheetItem *link =
                [[TUICustomActionSheetItem alloc] initWithTitle:TIMCommonLocalizabl
                       leftMark:[UIImage imageNamed:TUIChatImagePath_Minimalist(@"i
                withActionHandler:^(UIAlertAction *_Nonnull action) {
                    link.priority = 100;
```

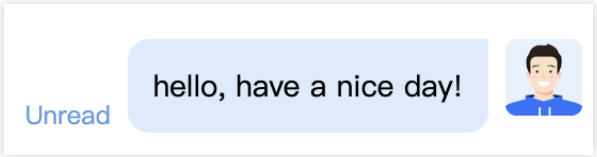
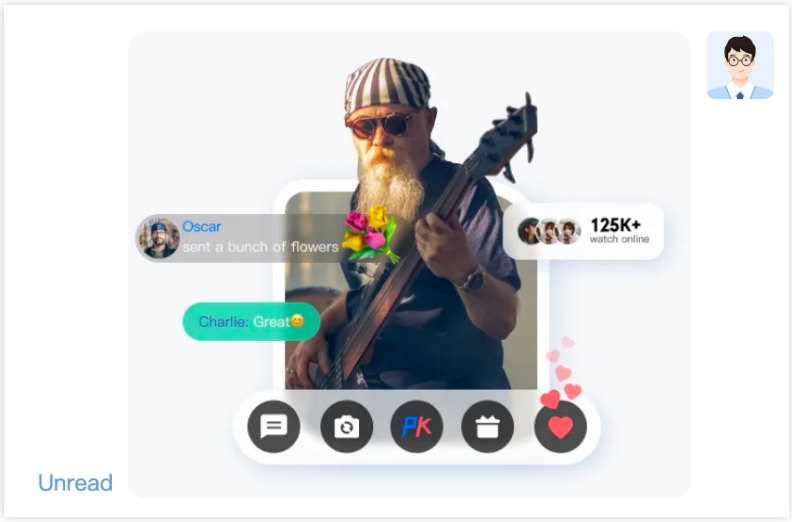
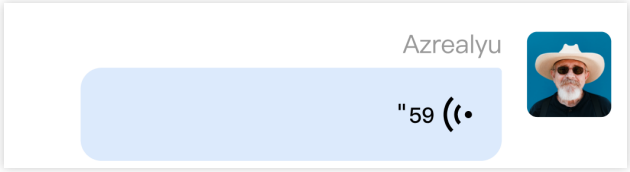
```
                    NSString *text = TIMCommonLocalizableString(TUIKitWelcome);
                    NSString *link = TUITencentCloudHomePageEN;
                    NSString *language = [TUIGlobalization tk_localizableLanguageKe
                    if ([language containsString:@"zh-"]) {
                        link = TUITencentCloudHomePageCN;
                    }
                    NSError *error = nil;
                    NSDictionary *param = @{BussinessID : BussinessID_TextLink, @"t
                    NSData *data = [NSJSONSerialization dataWithJSONObject:param op
                    if (error) {
                        NSLog(@"[%@] Post Json Error", [self class]);
                        return;
                    }
                       V2TIMMessage *message = [TUIMessageDataProvider getCustomMes
                    if ([weakSelf.delegate respondsToSelector:@selector(dataProvide
                        [weakSelf.delegate dataProvider:weakSelf sendMessage:messag
                    }
                }];
                    [arrayM addObject:link];
                }
                _customInputMoreActionItemList = [NSArray arrayWithArray:arrayM];
    }
    return _customInputMoreActionItemList;
}
@end
```

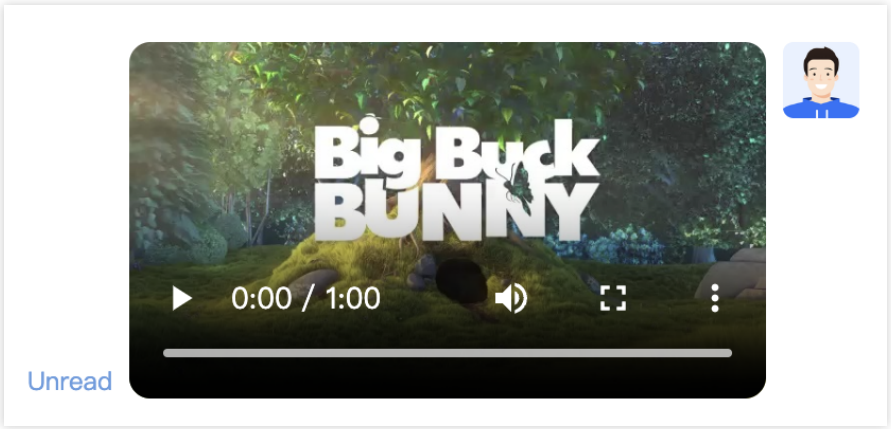
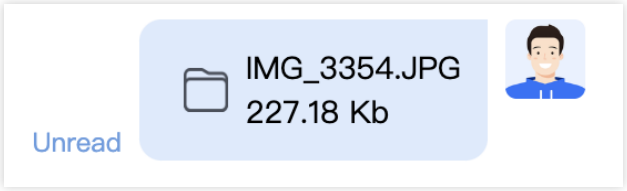
Web & H5 & Uniapp (Vue)

最近更新时间：2024-01-31 17:20:35

TUIKit 默认实现了文本、图片、语音、视频、文件等基本消息类型的发送和展示，如果这些消息类型满足不了您的需求，您可以新增自定义消息类型。

基本消息类型

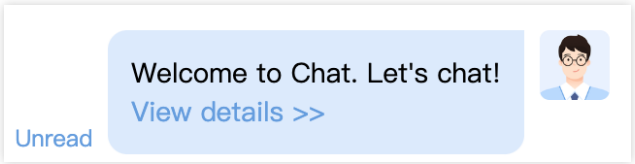
消息类型	显示效果图
文本类消息	
图片类消息	
语音类消息	

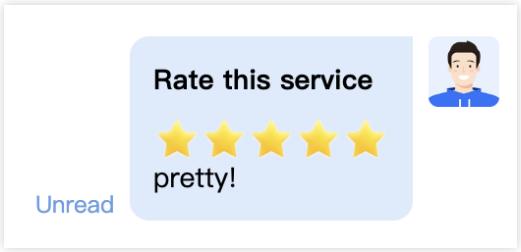
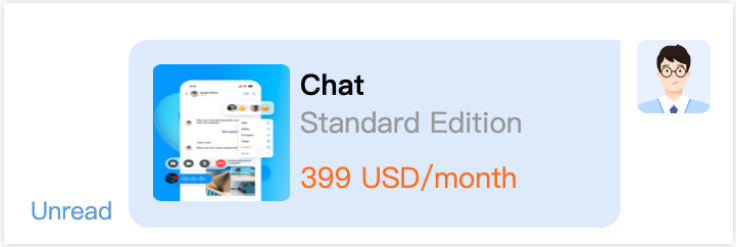
视频类消息	
文件类消息	

自定义消息

如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。

TUIKit 中内置了几种自定义消息样式，如下图所示：

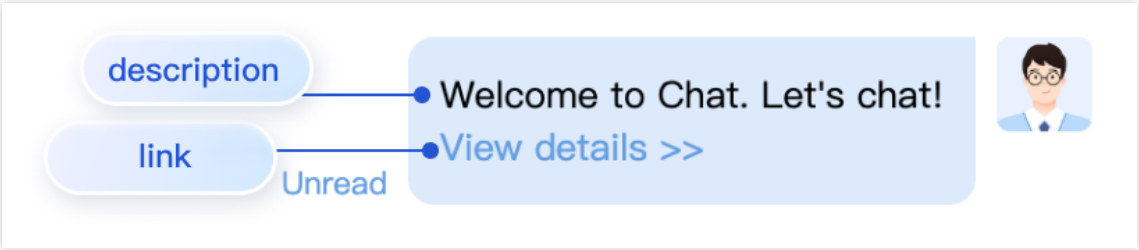
自定义消息预设样式	显示效果图
超文本类消息	
评价类消息	

	
订单类消息	

下文以发送一条可跳转至浏览器的超文本作为自定义消息为例，帮助您快速了解实现流程。

展示自定义消息

TUIKit 内置的超文本类自定义消息 cell 元素如下图所示：

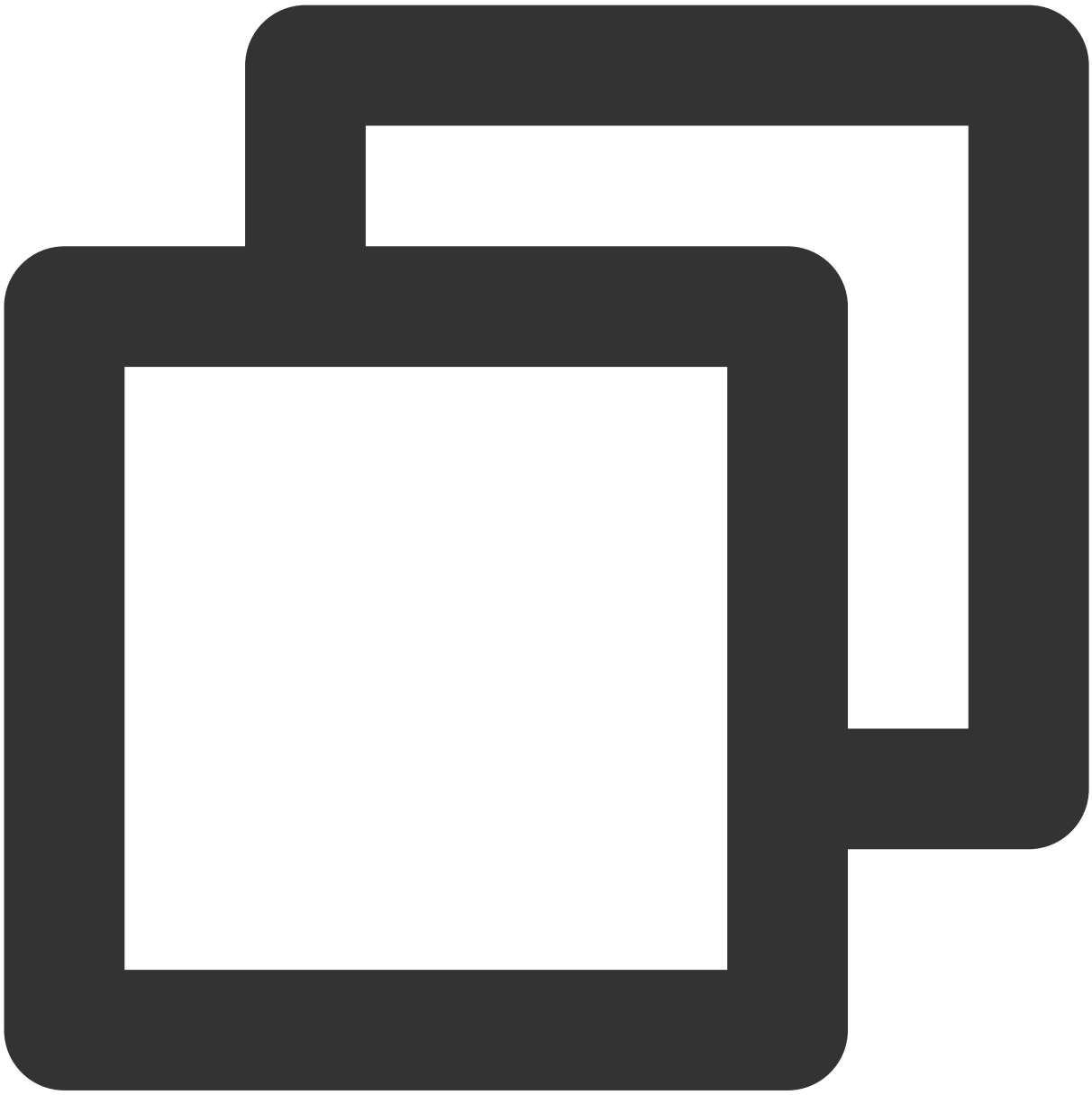


自定义类消息和其他普通类型消息接收方式一致，所有类型消息都通过 `TUIStore.watch(StoreName.CHAT, { messageList: onMessageListUpdated })` 来监听获取。
收到的自定义消息根据相应的具体类型字段以不同的形式展示在消息列表中 。
下面我们讲解下如何展示自定义消息。

创建自定义消息展示结构

自定义消息的展示主要通过在 `messageBubble` 的自定义消息类型内容区渲染 `messgaeCustom` 实现。
您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-`

`custom.vue` 文件下新增您需要的自定义消息展示结构样式。
以超文本类型消息展示结构为例，示例代码如下：



```
<template v-else-if="isCustom.businessID === 'text_link'">
  <div class="textLink">
    <p>{{ isCustom.text }}</p>
    <a :href="isCustom.link" target="view_window">
      {{
        TUITranslateService.t("message.custom.查看详情>>")
      }}
    </a>
```

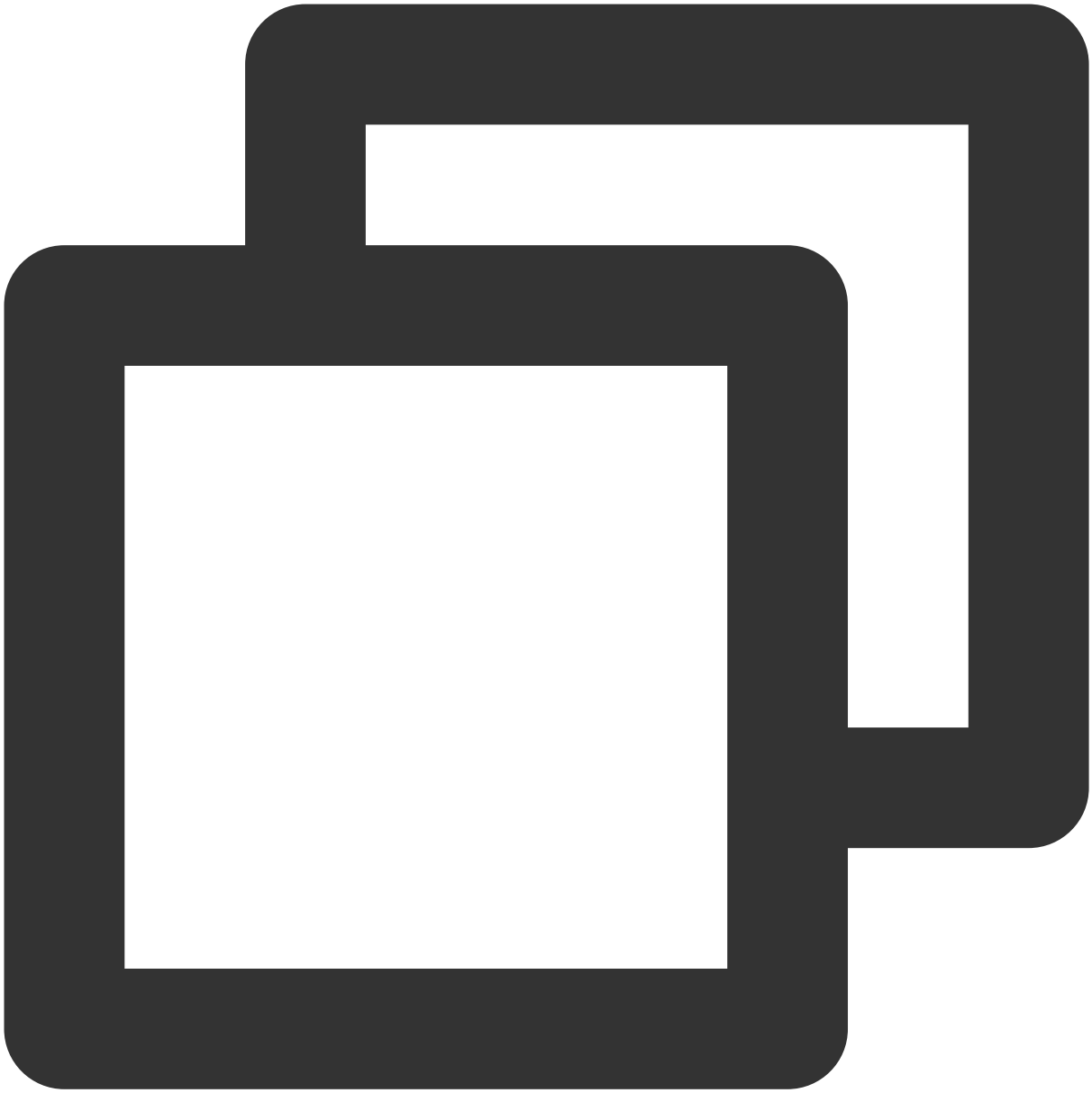
```
    </div>
  </template>
```

发送自定义消息

您可以通过调用路径 TUIKit 逻辑层 engine 的 `TUIChatService.sendCustomMessage` 方法来发送一条自定义消息，详情请参见：SendCustomMessage。
以下是几种 TUIKit 内置自定义样式消息发送示例：

sendCustomMessage(options, sendMessageOptions) ⇒ {Promise.<any>}

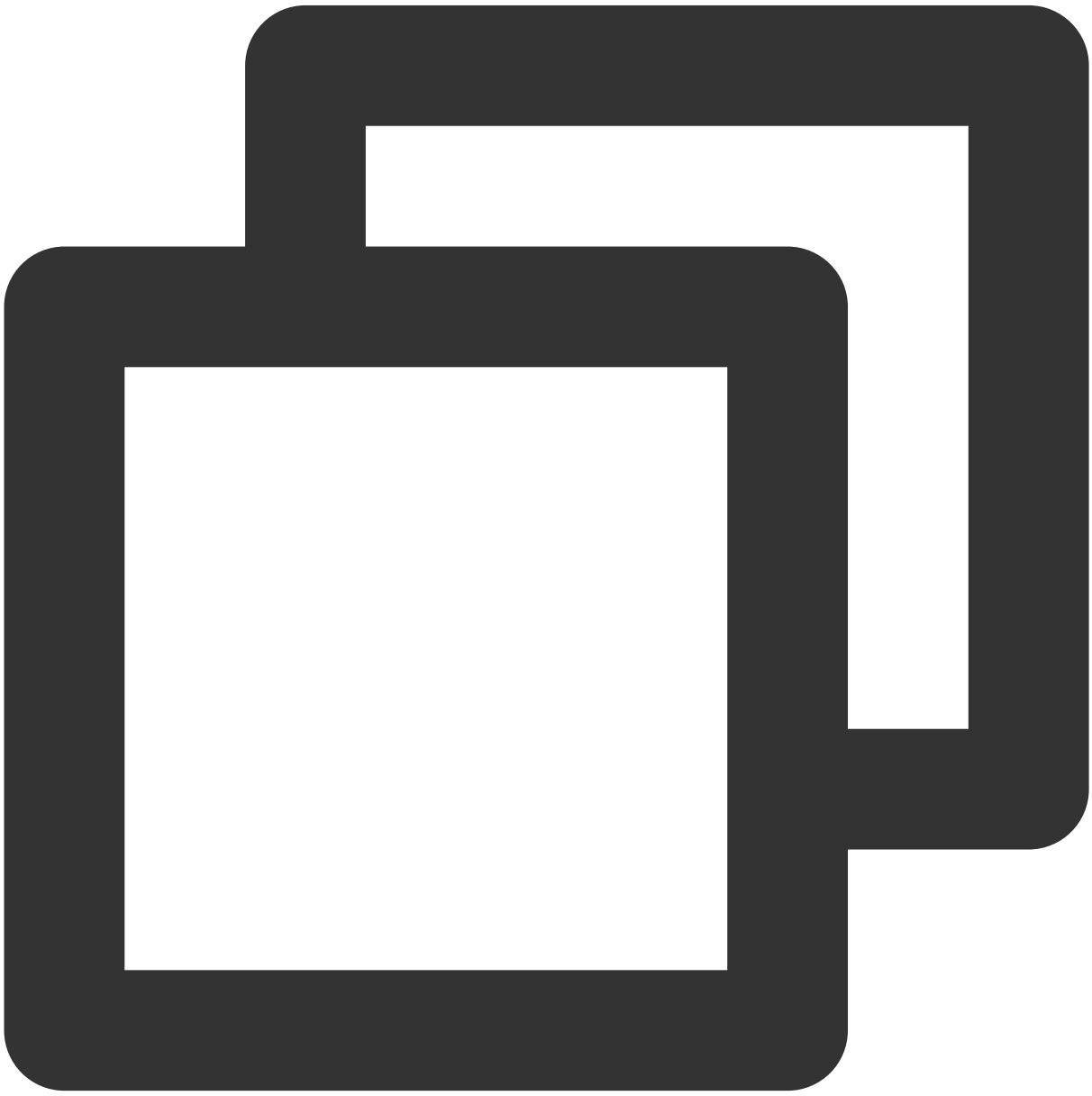
example1: 发送自定义评价消息



```
import { TUIChatService } from "@tencentcloud/chat-uikit-engine";
let promise = TUIChatService.sendCustomMessage({
  payload: {
    data: JSON.stringify({
      businessID: "evaluation",
      version: 1,
      score: 5,
      comment: "so pretty!!!"
    }),
    description: "Evaluation of this service",
    extension: "Evaluation of this service"
```

```
  }
});
promise.catch((error) => {
   ...
});
```

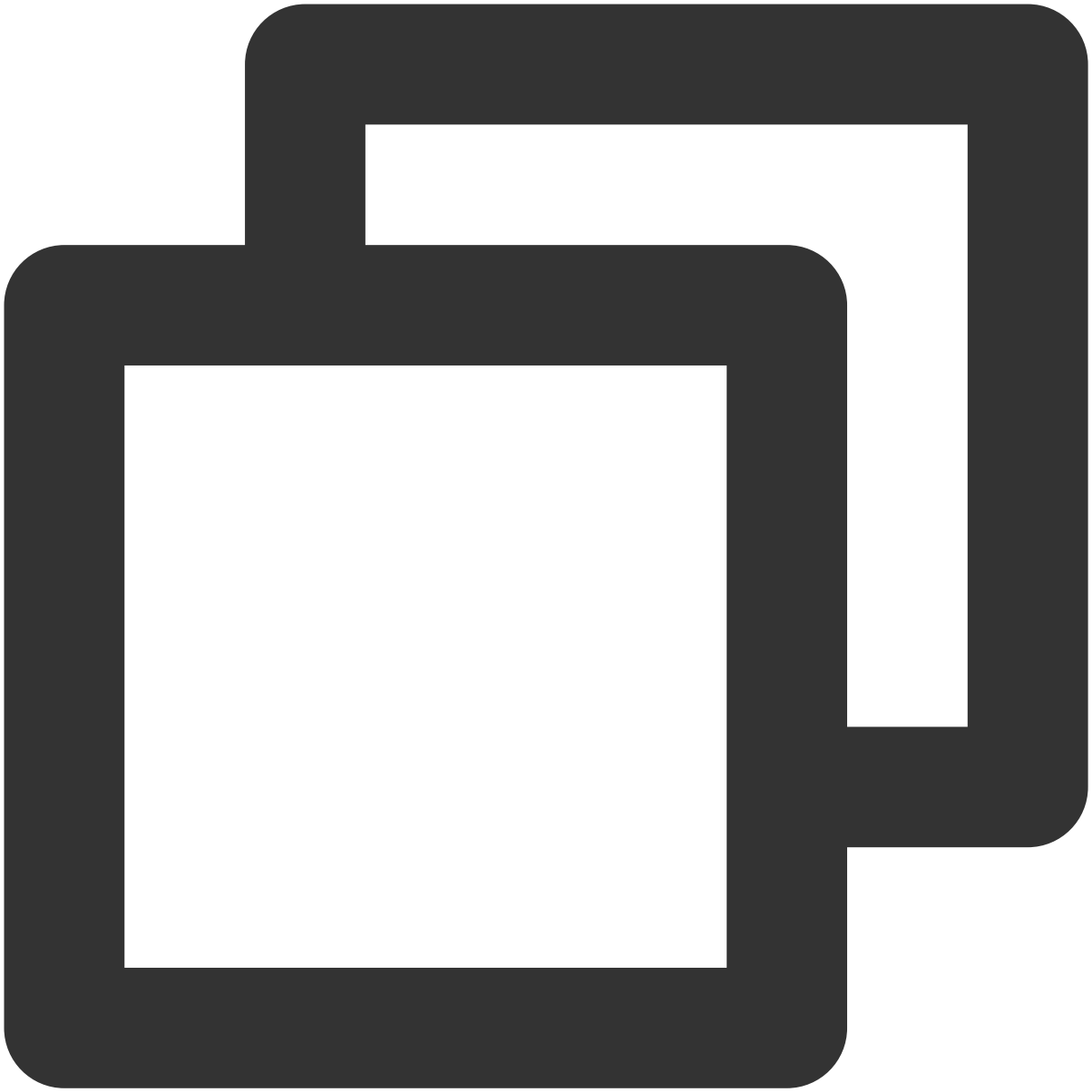
example2: 发送自定义超文本消息



```
import { TUIChatService } from "@tencentcloud/chat-uikit-engine";
let promise = TUIChatService.sendCustomMessage({
  payload: {
    data: JSON.stringify({
```

```
        businessID: "text_link",
        text: "Welcome to Chat. Let's chat!",
        link: "https://web.sdk.qcloud.com/im/demo/intl/index.html?scene=social"
      }),
      description: "",
      extension: ""
    }
});
promise.catch((error) => {
    ...
});
```

example3: 发送自定义订单消息



```
import { TUIChatService } from "@tencentcloud/chat-uikit-engine";
let promise = TUIChatService.sendCustomMessage({
  payload: {
    data: JSON.stringify({
      businessID: "order",
      title: "Chat",
      description: "Standard Edition",
      price: "399 USD/month",
      link: "https://buy.tencentcloud.com/avc",
      imageUrl: "https://1302445663.vod2.myqcloud.com/cea47bfavodsgp1302445663/fd67
    }),
```

```
    description: "",
    extension: ""
  }
});
promise.catch((error) => {
   ...
});
```

参数说明：

名称	类型	可选类型	描述
options	SendMessageParams	必选	自定义消息相关参数
sendMessageOptions	SendMessageOptions	可选	消息发送选项

返回值

`Promise.<any>`

交流与反馈

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

Flutter

最近更新时间：2024-01-31 14:16:47

TUIKit 默认实现了文本、图片、语音、视频、文件等基本消息类型的发送和展示，如果这些消息类型满足不了您的需求，您可以新增自定义消息类型。

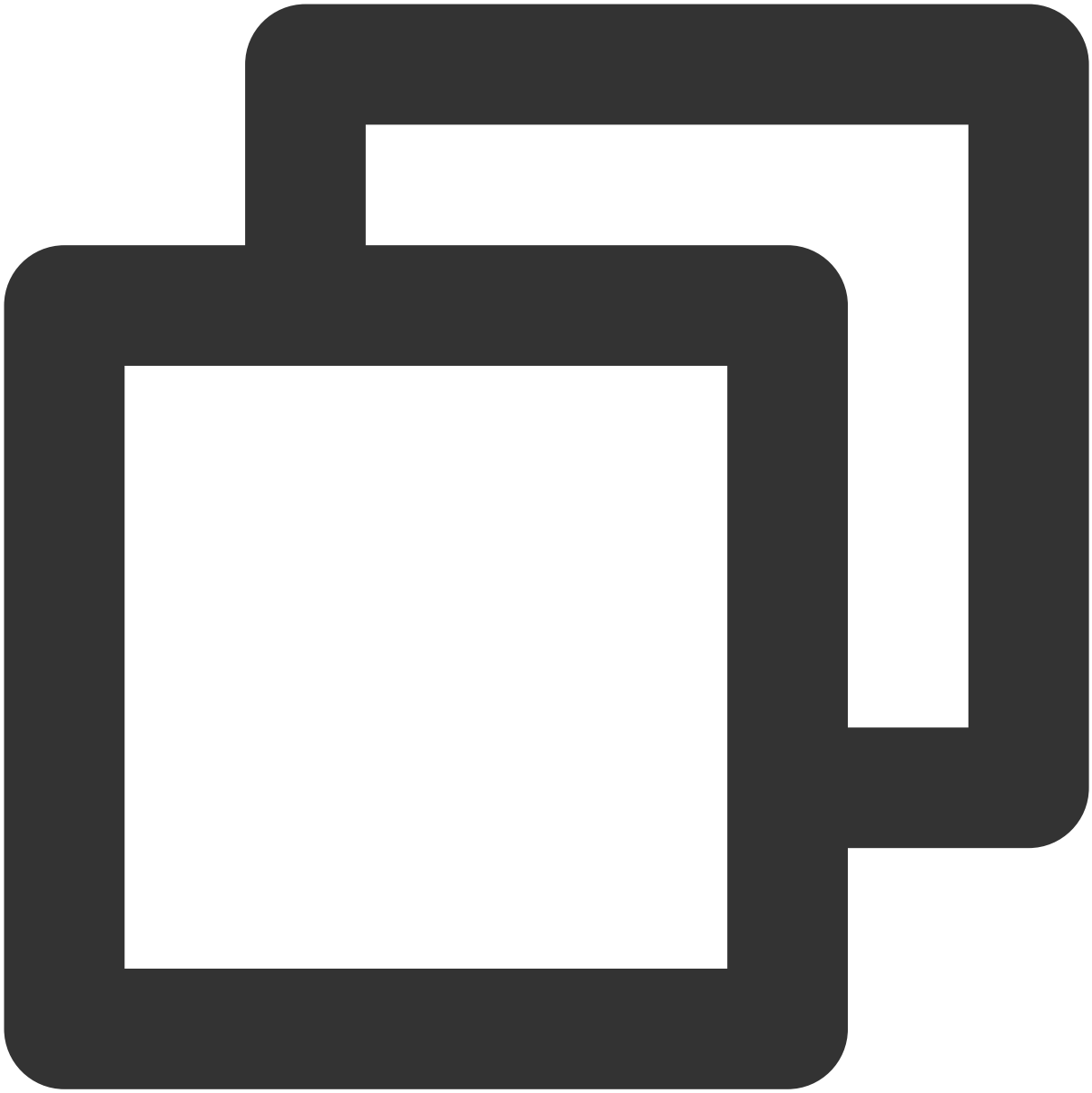
自定义消息

如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。
下文以发送一条可跳转至浏览器的超文本作为自定义消息为例，帮助您快速了解实现流程。
下面我们分步骤讲解使用自定义消息。

展示自定义消息

自定义消息携带的信息，以 String 格式，存储于 `V2TimMessage.V2TimCustomElem.data` 中。如果需要传递的信息量大，建议使用 JSON 格式。
展示自定义消息基本逻辑：解析时，将 JSON String 解析成 Map，用于实例化一个预定义的类，以该对象内的数据，渲染自定义消息体。
1. 定义一个类，用于自定义消息解析后的结构。并完成一个 fromJSON 的方法，通过 Map 对其实例化。
以上述包含链接及文本的自定义消息格式举例：

版权所有：腾讯云计算（北京）有限责任公司

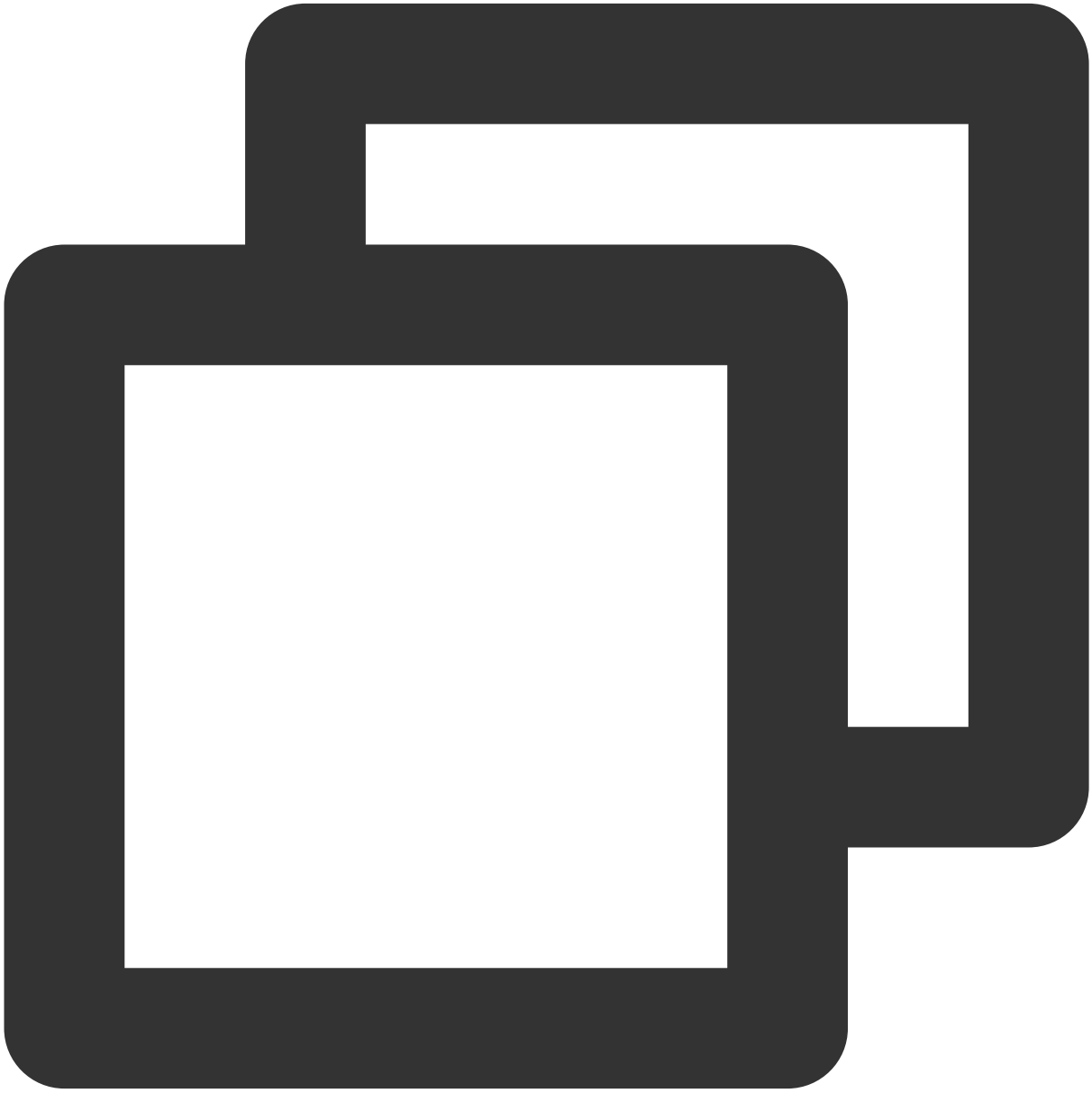


```
class CustomMessage {
  // 此处的内容请根据需要自行定义
  String? link;
  String? text;
  String? businessID;

  CustomMessage.fromJSON(Map json) {
 link = json["link"];
 text = json["text"];
 businessID = json["businessID"];
  }
```

```
}
```

2. 编写一个方法，用于实现对自定义消息的解析，得到一个数据对象。
示例代码如下：

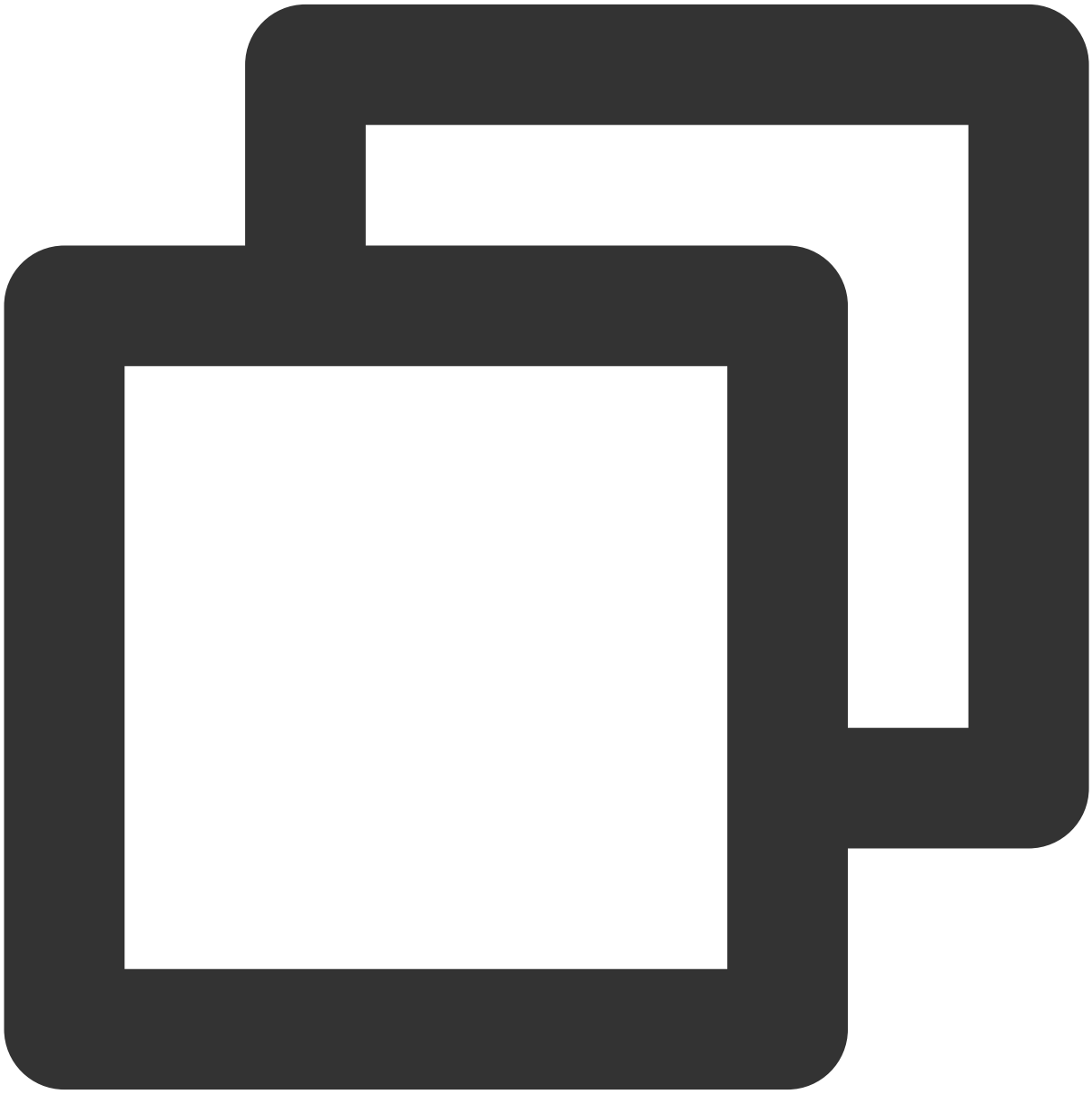


```
CustomMessage? getCustomMessageData(V2TimCustomElem? customElem) {
  try {
 if (customElem?.data != null) {
   final customMessage = jsonDecode(customElem!.data!);
   return CustomMessage.fromJSON(customMessage);
 }
```

```
  return null;
   } catch (err) {
  return null;
   }
}
```

3. 在 `TIMUIKitChat` 中，使用 `messageItemBuilder` 内 `customMessageItemBuilder`，渲染自定义消息。

示例代码如下：



```
messageItemBuilder: MessageItemBuilder(
```

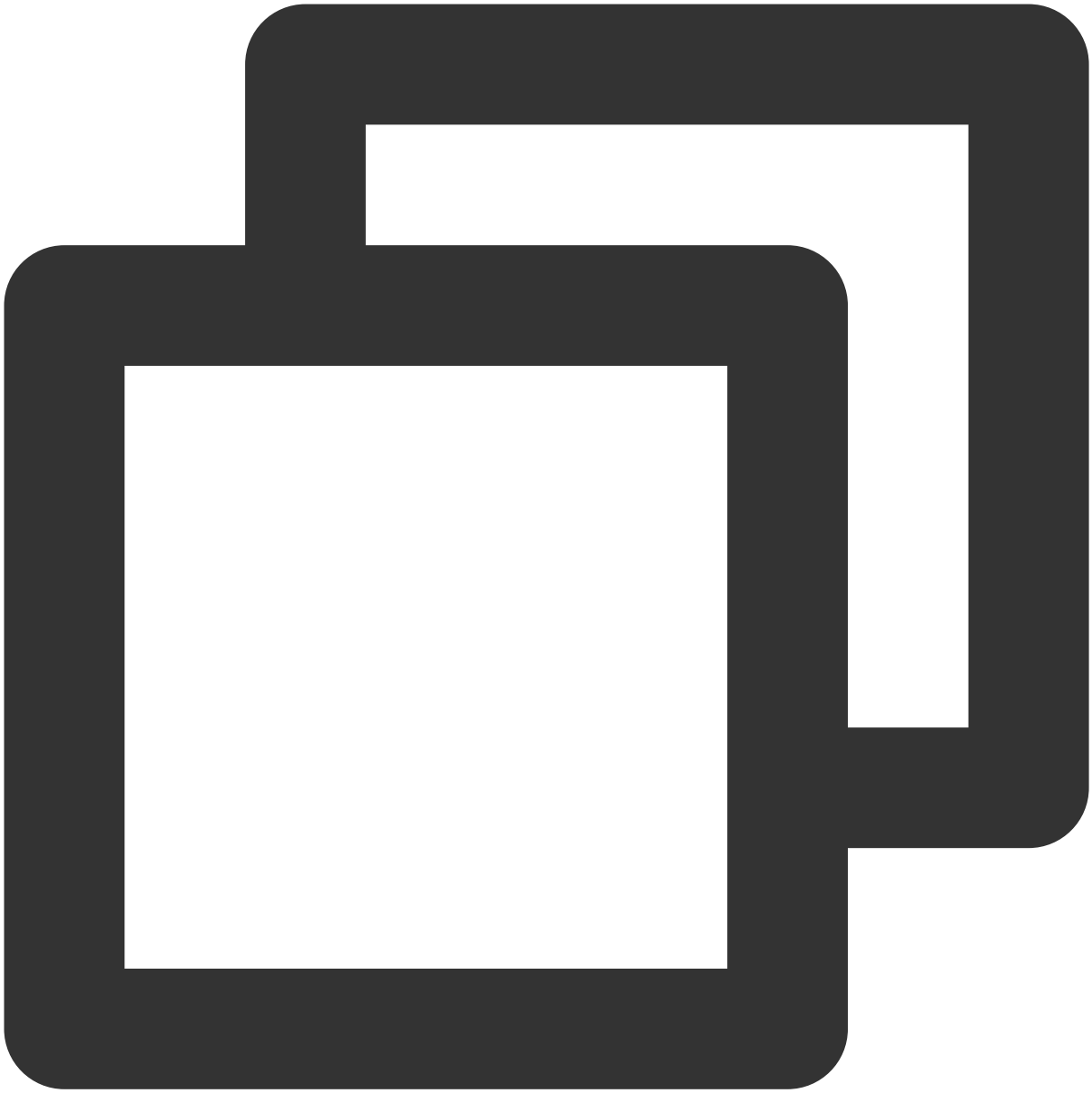
```
   customMessageItemBuilder: (message, isShowJump, clearJump) {
  final CustomMessage customMessage = getCustomMessageData(message.customElem);
  if (linkMessage != null) {
    final String option1 = linkMessage.link ?? "";
    return Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(linkMessage.text ?? ""),
        MarkdownBody(
          data: TIM_t_para(
              "[查看详情 >>]({{option1}})", "[查看详情 >>]($option1)")(
              option1: option1),
          styleSheet: MarkdownStyleSheet.fromTheme(ThemeData(
              textTheme: const TextTheme(
                  bodyText2: TextStyle(fontSize: 16.0))))
              .copyWith(
            a: TextStyle(color: LinkUtils.hexToColor("015fff")),
          ),
        )
      ],
    );
  }
   }
),
```

发送自定义消息

发送自定义消息基本步骤：使用 SDK 创建一条自定义消息，将要发送的内容转成 JSON String 格式放入 `data` 中，并调用 `TIMUIKitChatController` 的 `sendMessage` 接口发送。

此处演示示例以直接通过更多功能面板，创建并发送一条自定义消息为例。

1. 实例化一个消息控制器，并传入 `TIMUIKitChat` 中。

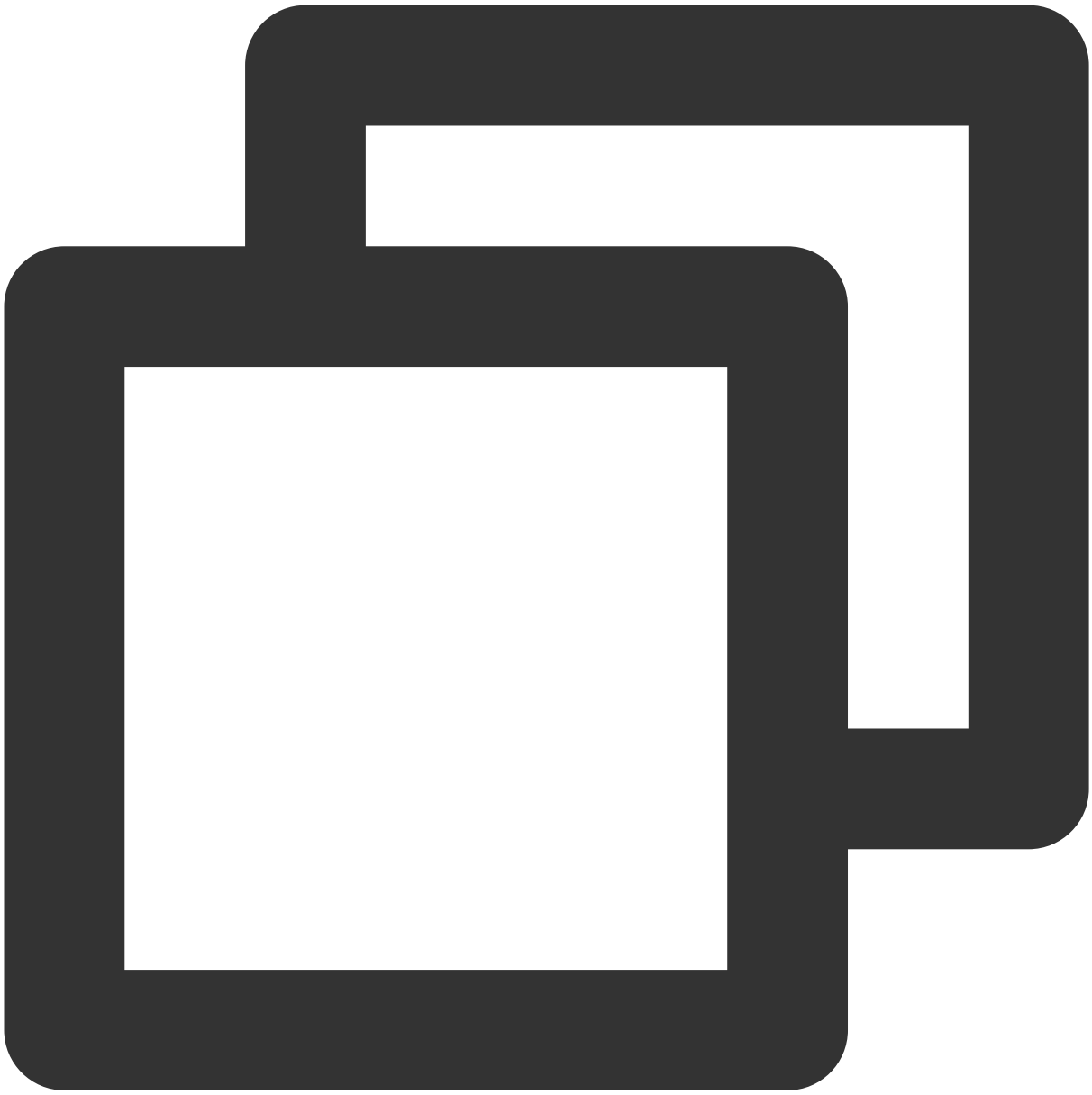


```
final TIMUIKitChatController _timuiKitChatController =
   TIMUIKitChatController();

return TIMUIKitChat(
   controller: _timuiKitChatController,
   // ...其他参数
   )
```

2. 为 `TIMUIKitChat` 的 `morePanelConfig` 属性中的 `extraAction` 数组新增一项，添加自定义消息发送按钮。

更多功能面板的按钮，主要由文本 `title` 和图片 `icon` 组成。
示例代码如下：

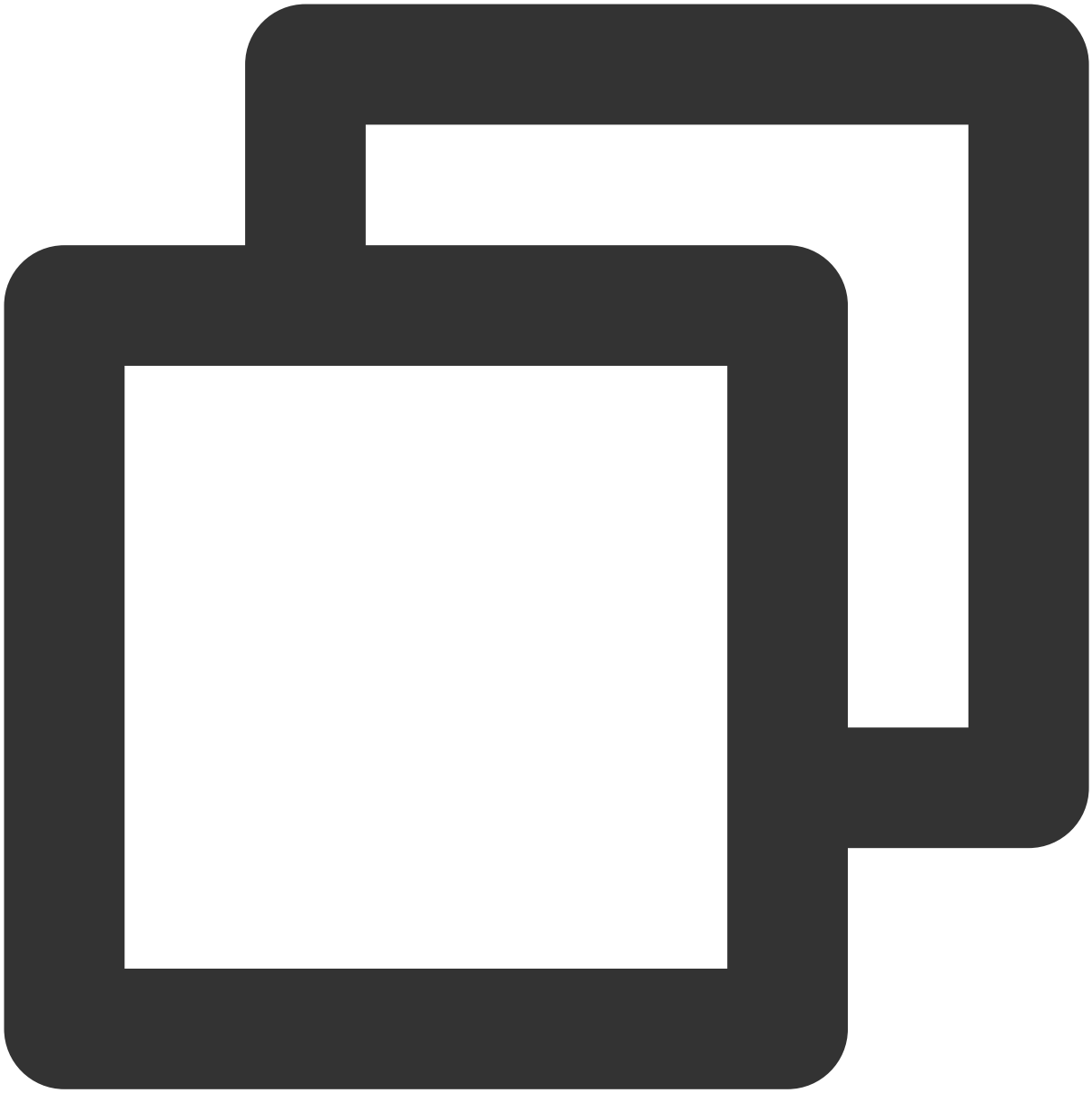


```
morePanelConfig: MorePanelConfig(
  extraAction: [
 MorePanelItem(
     id: "customMessage",
     title: imt("自定义消息"),
     onTap: (c) {
       _sendCustomMessage();
     },
```

```
      icon: Container(
        height: 64,
        width: 64,
        margin: const EdgeInsets.only(bottom: 4),
        decoration: const BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.all(Radius.circular(5))),
        child: SvgPicture.asset(
          "images/custom-msg.svg",
          package: 'tencent_cloud_chat_uikit',
          height: 64,
          width: 64,
        ),
      )
  ),
  // ... 其他更多功能面板按钮
   ],
   // ...其他参数
)
```

3. 实现上述步骤所需的自定义消息发送方法。

示例：点击消息发送按钮后就可以创建一条自定义消息发送，代码如下：



```
_sendCustomMessage() async {
  // 创建自定义消息，下方的data/desc/extension可由您自行定义内容。
  V2TimValueCallback<V2TimMsgCreateInfoResult> createCustomMessageRes =
   await TencentImSDKPlugin.v2TIMManager
       .getMessageManager()
       .createCustomMessage(
         data:
             '{"businessID":"text_link","link":"https://cloud.tencent.com/document/
         desc: '自定义desc',
         extension: '自定义extension',
       );
```

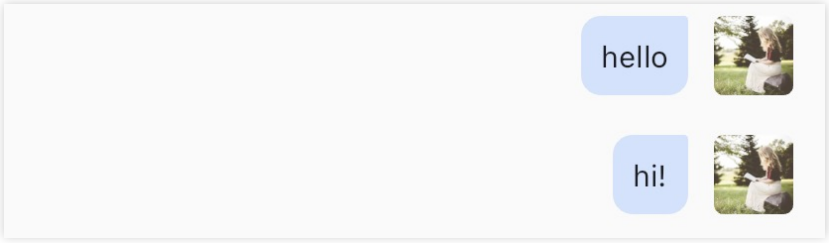
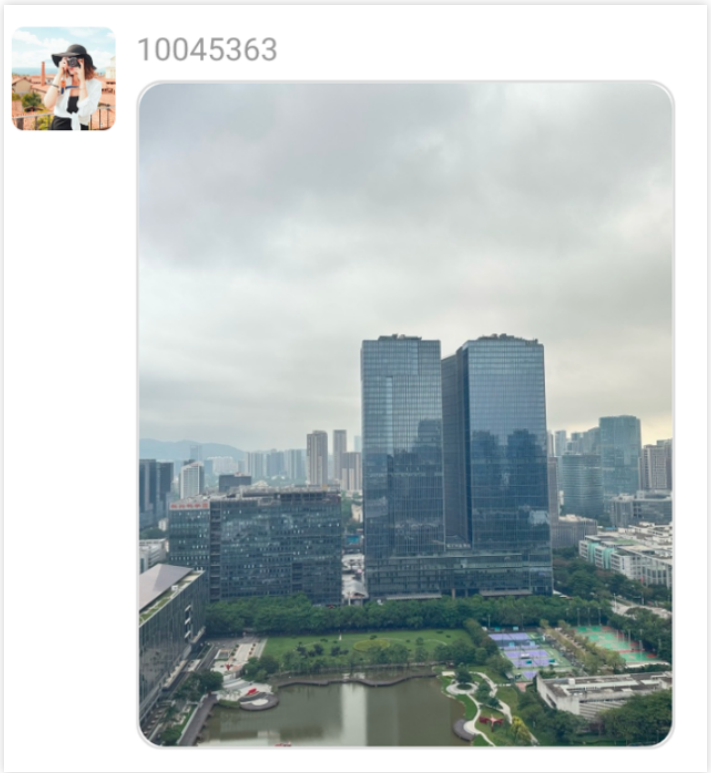
```
   if (createCustomMessageRes.code == 0) {
  String? id = createCustomMessageRes.data?.id;
  // 发送自定义消息
  V2TimValueCallback<V2TimMessage>? sendMessageRes =
      await _timuiKitChatController.sendMessage(
          messageInfo: createCustomMessageRes.data?.messageInfo);
  if (sendMessageRes!.code == 0) {
    // 发送成功
    sendMessageRes.data?.customElem?.data; //自定义data
    sendMessageRes.data?.customElem?.desc; //自定义desc
    sendMessageRes.data?.customElem?.extension; //自定义extension
  }
   }
}
```

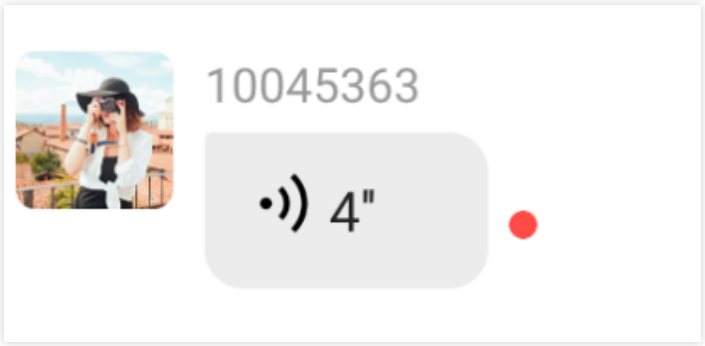
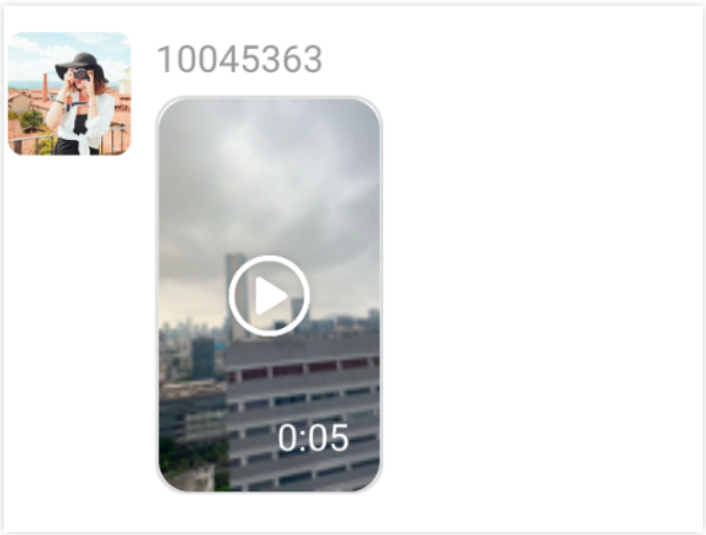
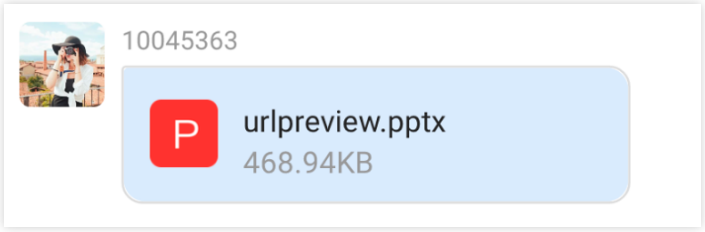
ReactNative

最近更新时间：2023-08-23 17:25:01

TUIKit 默认实现了文本、图片、语音、视频、文件等基本消息类型的发送和展示，如果这些消息类型满足不了您的需求，您可以新增自定义消息类型。

基本消息类型

消息类型	显示效果图
文本类消息	
图片类消息	

语音类消息	
视频类消息	
文件类消息	

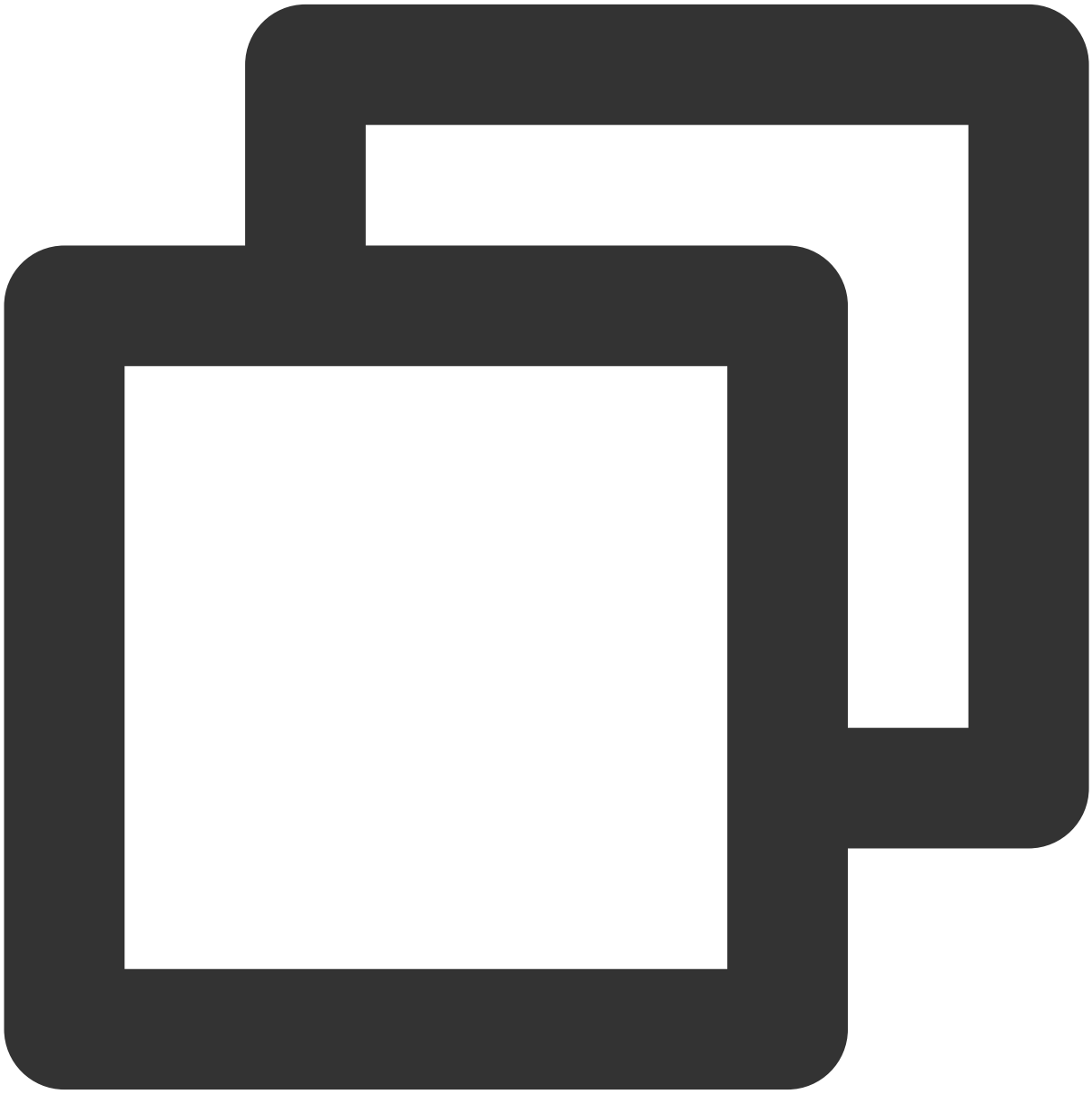
自定义消息

如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。

自定义类消息和其他普通类型消息接收方式一致，所有类型消息都通过监听 onRecvNewMessage 事件来获取。
收到的自定义消息根据相应的具体类型字段以不同的形式展示在消息列表中 。
下面我们讲解下如何展示自定义消息。

创建自定义消息展示结构

您可以在路径 `src/TUIKit/components/TUIMessage/element/custom_element.tsx` 文件下新增您需要的自定义消息展示结构样式。



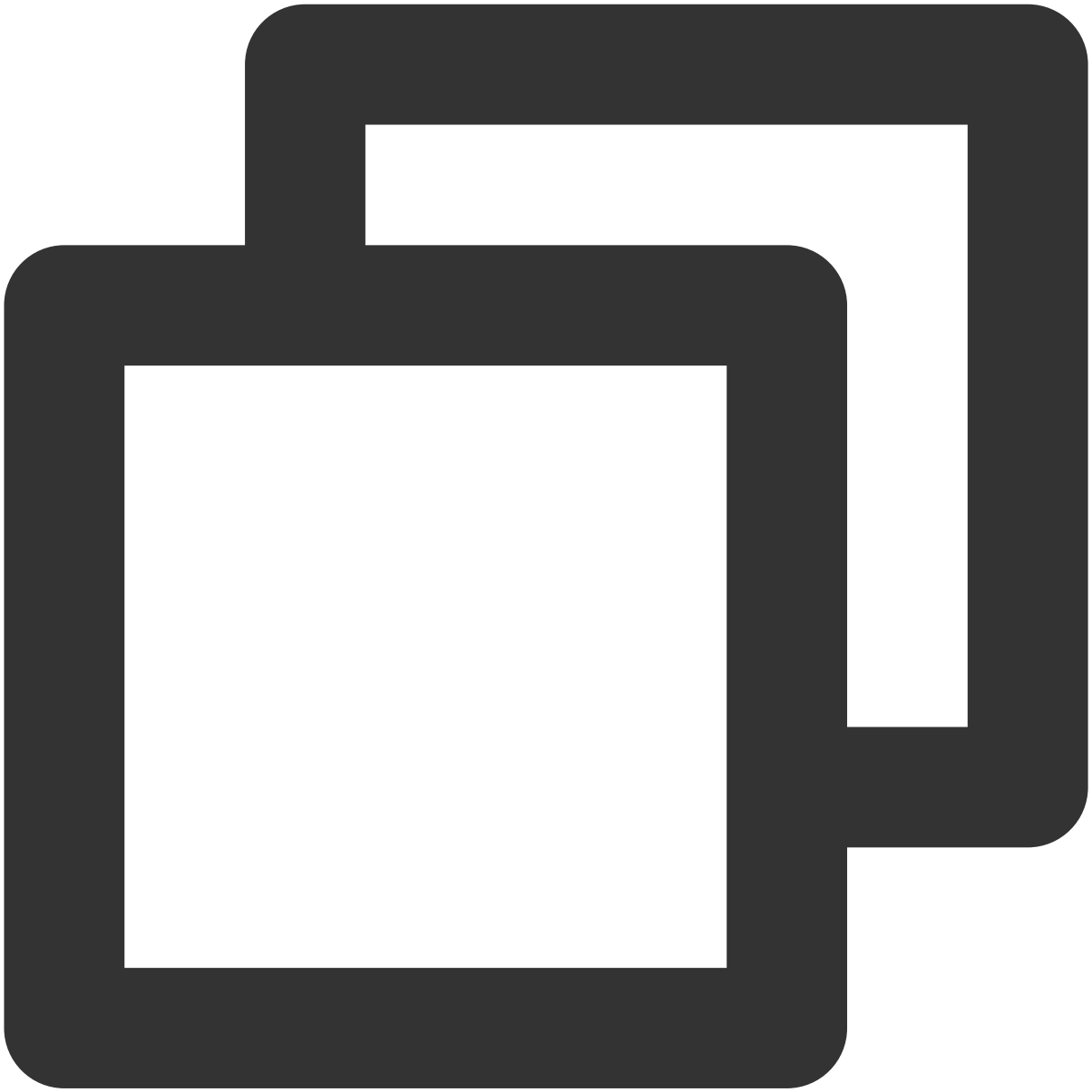
```
import React from 'react';
import {Text} from 'react-native';
```

```
import type {V2TimMessage} from 'react-native-tim-js/lib/typescript/src/interface';

export const CustomElement = (props: {message: V2TimMessage}) => {
  const {message} = props; //message包含了当前自定义消息的所有属性，您可以根据自定义需求来更
  console.log(message);
  return <Text>["自定义消息"]</Text>;
};
```

发送自定义消息

您可以通过调用 createCustomMessage 方法来发送一条自定义消息。 您可以通过构建不同的 data 数据项来实现发送不同类型的自定义消息。
发送自定义消息示例代码如下：



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

// 创建自定义消息
const createCustomMessageRes = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .createCustomMessage({
        data: '自定义data',
        desc: '自定义desc',
        extension: '自定义extension',
    });
if (createCustomMessageRes.code === 0) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
    const id = createCustomMessageRes.data?.id;
    // 发送自定义消息
    // 在sendMessage时，若只填写receiver则发个人用户单聊消息
    //                 若只填写groupID则发群组消息
    //                 若填写了receiver与groupID则发群内的个人用户，消息在群聊中显示，只有指
    const sendMessageRes = await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .sendMessage({ id: id!, receiver: 'userID', groupID: 'groupID' });
    if (sendMessageRes.code === 0) {
        // 发送成功
        sendMessageRes.data?.customElem?.data; //自定义data
        sendMessageRes.data?.customElem?.desc; //自定义desc
        sendMessageRes.data?.customElem?.extension; //自定义extension
    }
}
```

交流与反馈

点此进入IM社群，享有专业工程师的支持，解决您的难题

添加自定义表情

Android

最近更新时间：2024-05-20 15:04:25

TUIChat 支持添加自定义表情。

说明：

TUIChat 支持添加这些格式的图片作为自定义表情： JPEG、JPG、PNG、BMP ， 从 7.8 版本开始，支持 GIF 格式。

新增自定义表情

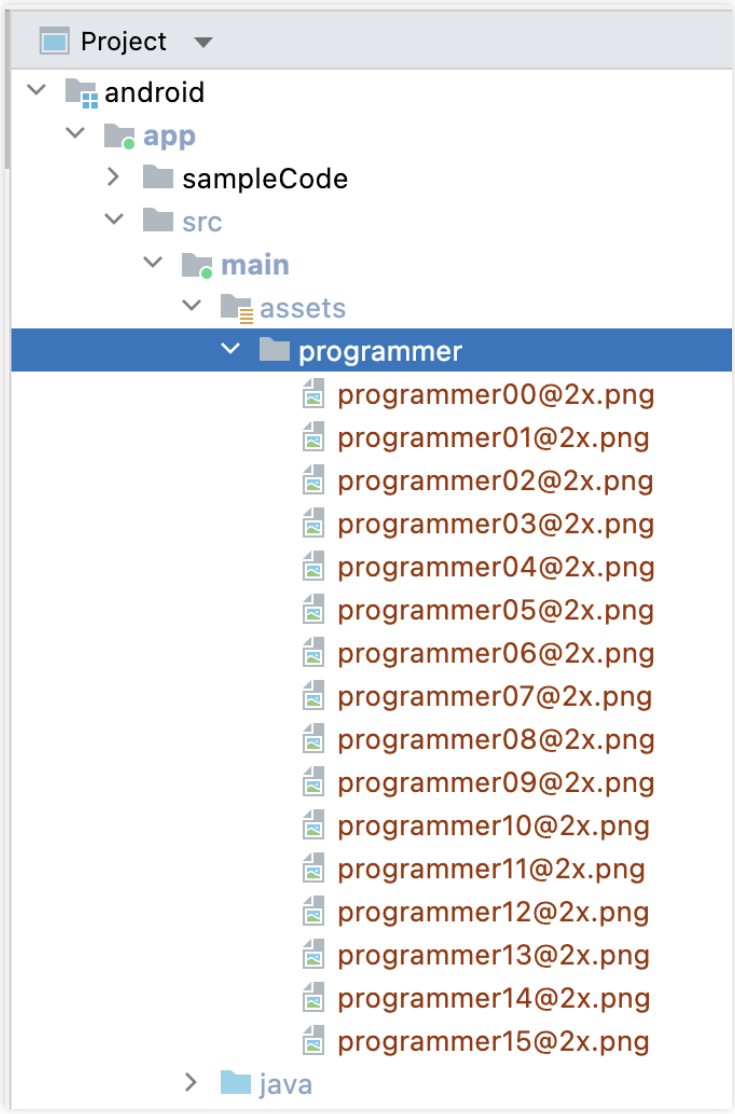
TUIChat 支持从沙盒、assets 目录以及网络路径加载自定义表情。

以添加 assets 目录下的 programmer 表情为例：

准备表情包资源

App 的 src/main 文件夹下新建 assets 文件夹，将表情文件夹放在 App 的 assets 目录下：

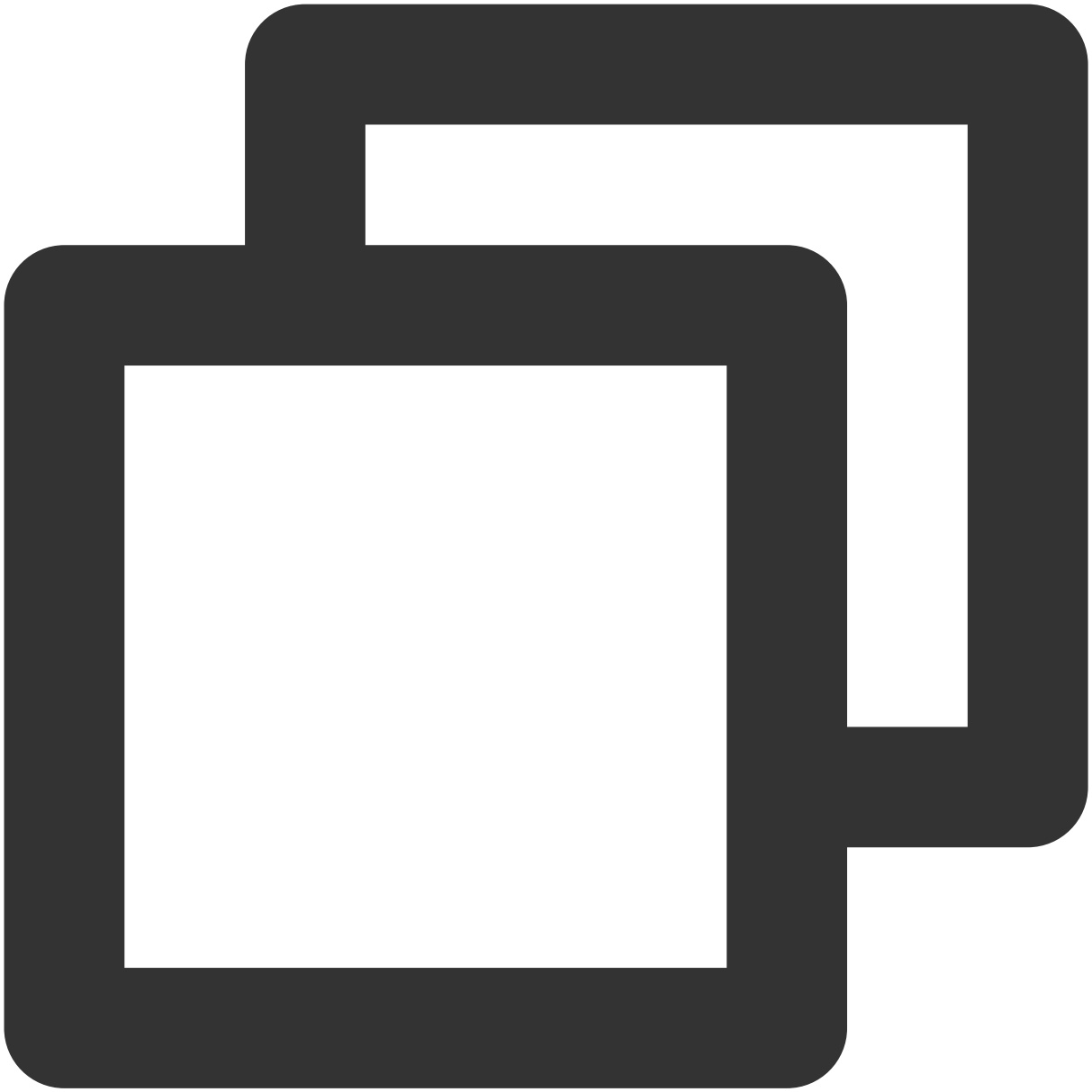
版权所有：腾讯云计算（北京）有限责任公司



添加表情包

在应用启动时调用接口将自定义表情添加到 FaceManager ：

每个表情包都有唯一的 faceGroupID，表情包中的每个表情对应一个 faceKey，表情包添加到 FaceManager 中之后“更多表情”输入界面会根据 faceGroupID 的大小进行排序显示。

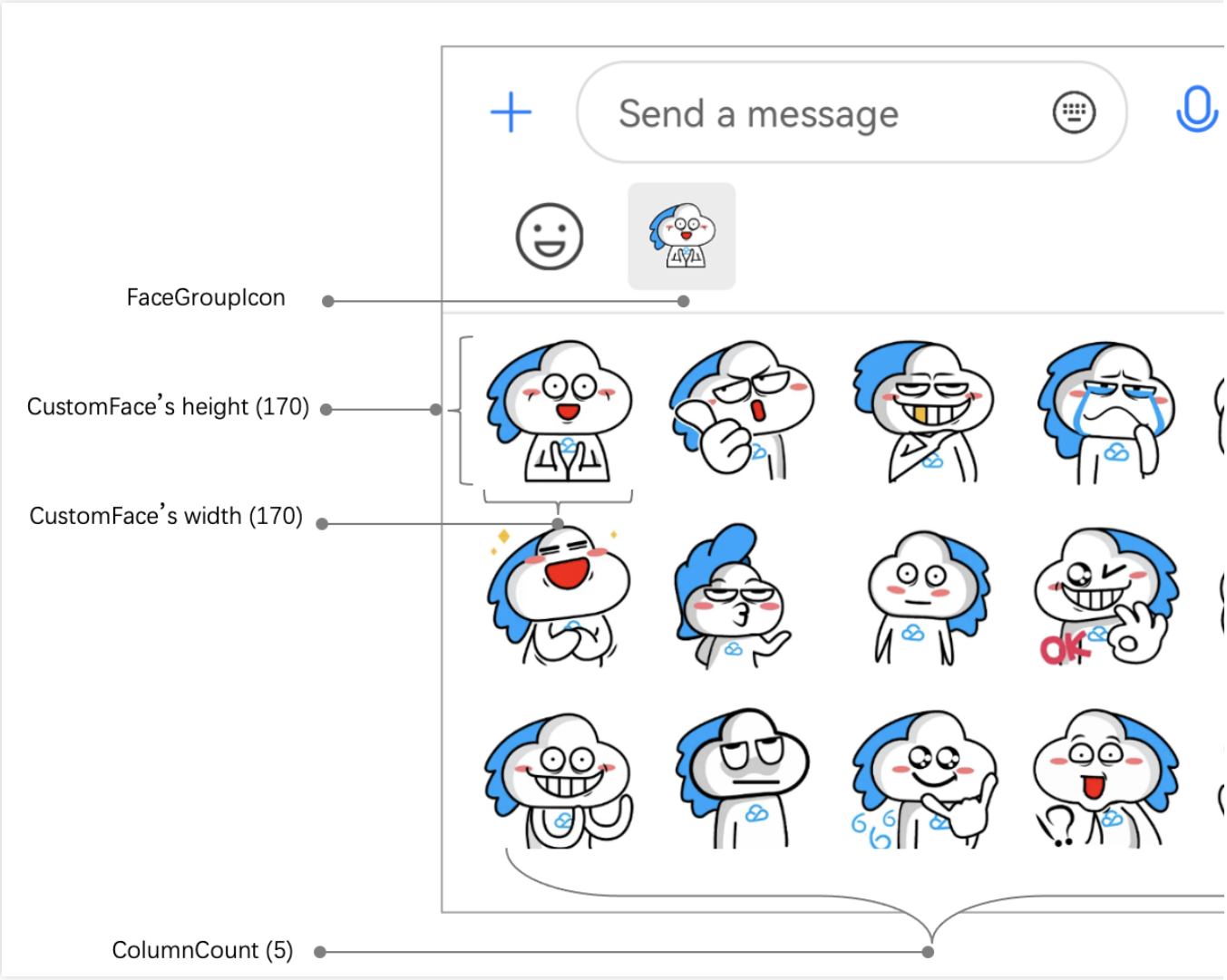


```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        FaceGroup programmerGroup = new FaceGroup();
        // The number of emojis displayed per row on the **More emojis** input UI
        programmerGroup.setPageColumnCount(5);
        // The thumbnail of the sticker
        programmerGroup.setFaceGroupIconUrl("file:///android_asset/programmer/progr
        // The name of the sticker
        programmerGroup.setGroupName("programmer");
        for (int i = 0; i < 16; i++) {
```

```
            CustomFace customFace = new CustomFace();
            String index = "" + i;
            if (i < 10) {
                index = "0" + i;
            }
            // Put emojis in the `assets` directory. If the sandbox path or network
            customFace.setAssetPath("programmer/programmer" + index + "@2x.png");
            // The `key` of the emoji
            String faceKey = "programmer" + index;
            customFace.setFaceKey(faceKey);
            // The width of the emoji on the **More emojis** input UI
            customFace.setWidth(170);
            // The height of the emoji on the **More emojis** input UI
            customFace.setHeight(170);
            programmerGroup.addFace(faceKey, customFace);
        }
        // Register the sticker in `FaceManager`. `FaceGroupID` is `1`.
        FaceManager.addFaceGroup(1, programmerGroup);
    }
}
```

添加成功的效果

添加成功之后，打开聊天界面“更多表情”输入界面即可看到新添加的表情包：

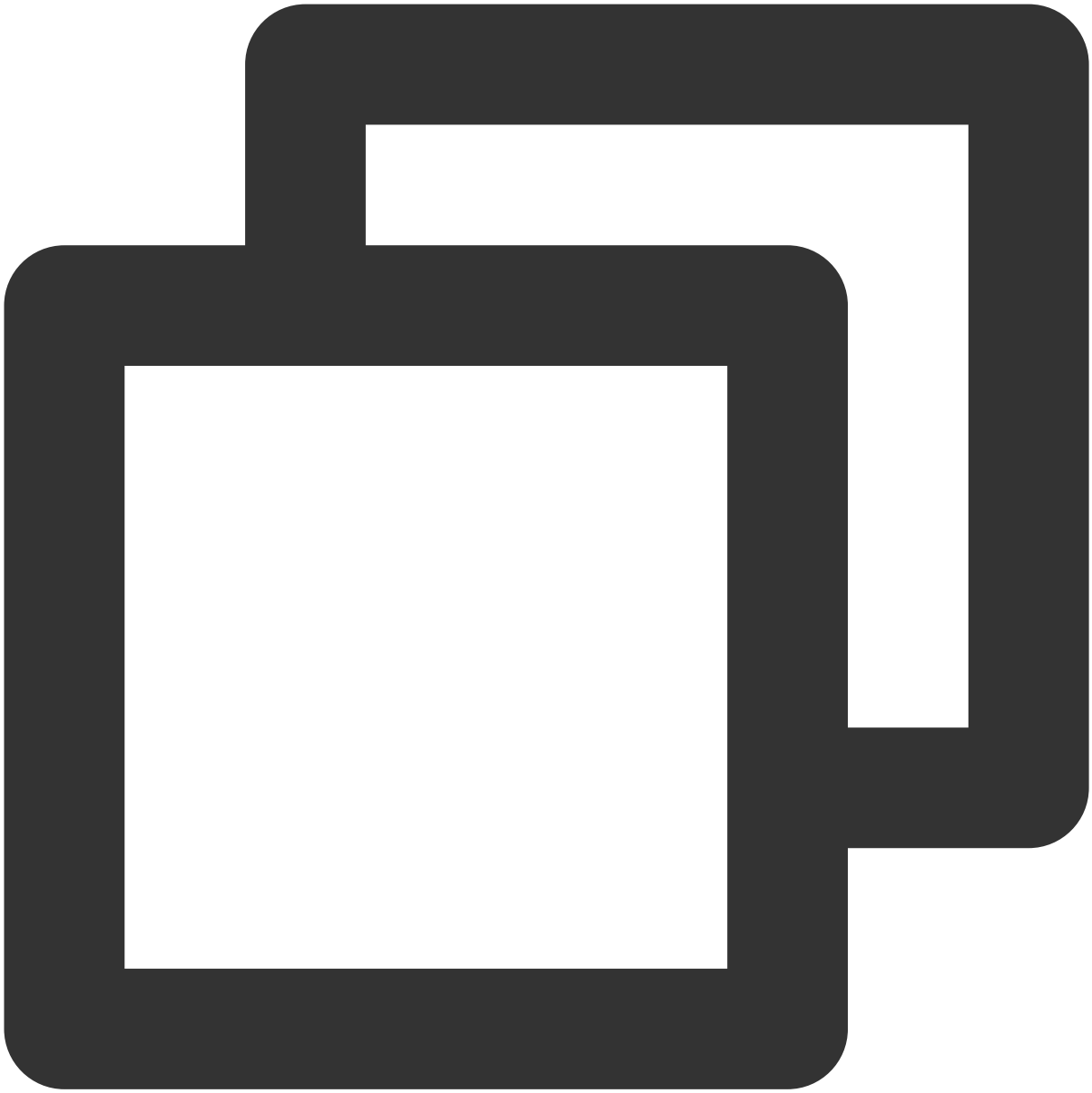


注意

faceGroupID 是大于 0 的整数而且不可重复。

发送自定义表情

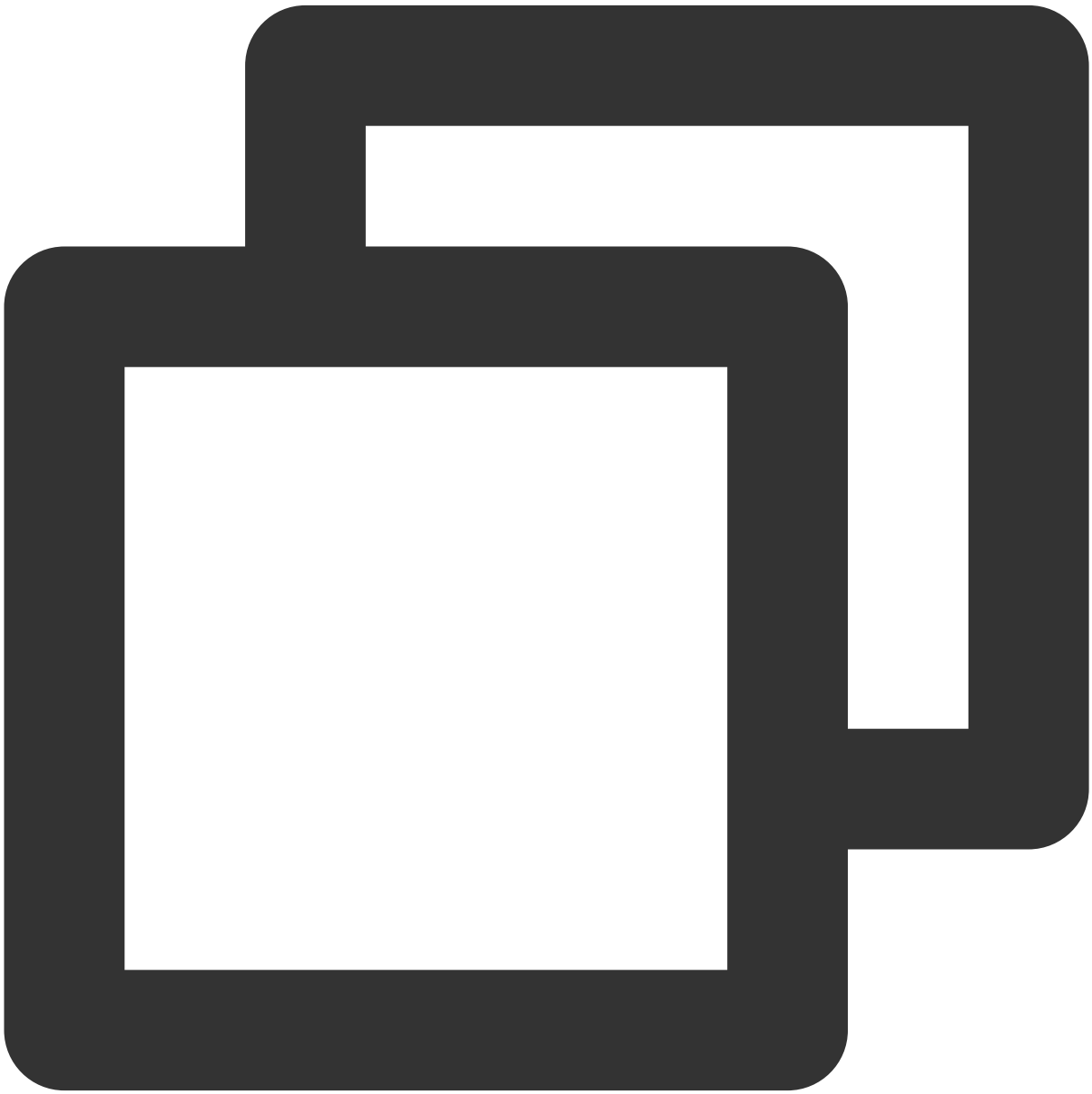
添加自定义表情之后，可以在聊天的“更多表情”输入界面看到已经添加的表情，点击表情即可发送。
也可以使用代码生成表情消息然后发送，例如：



```
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager()
        .createFaceMessage(faceGroupID, faceKey.getBytes());
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage,
        userID,
        null,
        V2TIMMessage.V2TIM_PRIORITY_DEFAULT,
        false,
        null,
        new V2TIMSendCallback<V2TIMMessage>() {...}
```

解析自定义表情

接收到自定义表情消息之后，TUIKit 会将 IMSDK 的 V2TIMMessage 解析为 FaceMessageBean 类型，可以由 FaceMessageBean 获得自定义表情的 faceGroupID 和 faceKey：



```
TUIMessageBean messageBean = ChatMessageParser.parseMessage(v2TIMMessage);
FaceMessageBean faceMessageBean = null;
if (messageBean instanceof FaceMessageBean) {
    faceMessageBean = (FaceMessageBean) messageBean;
}
```

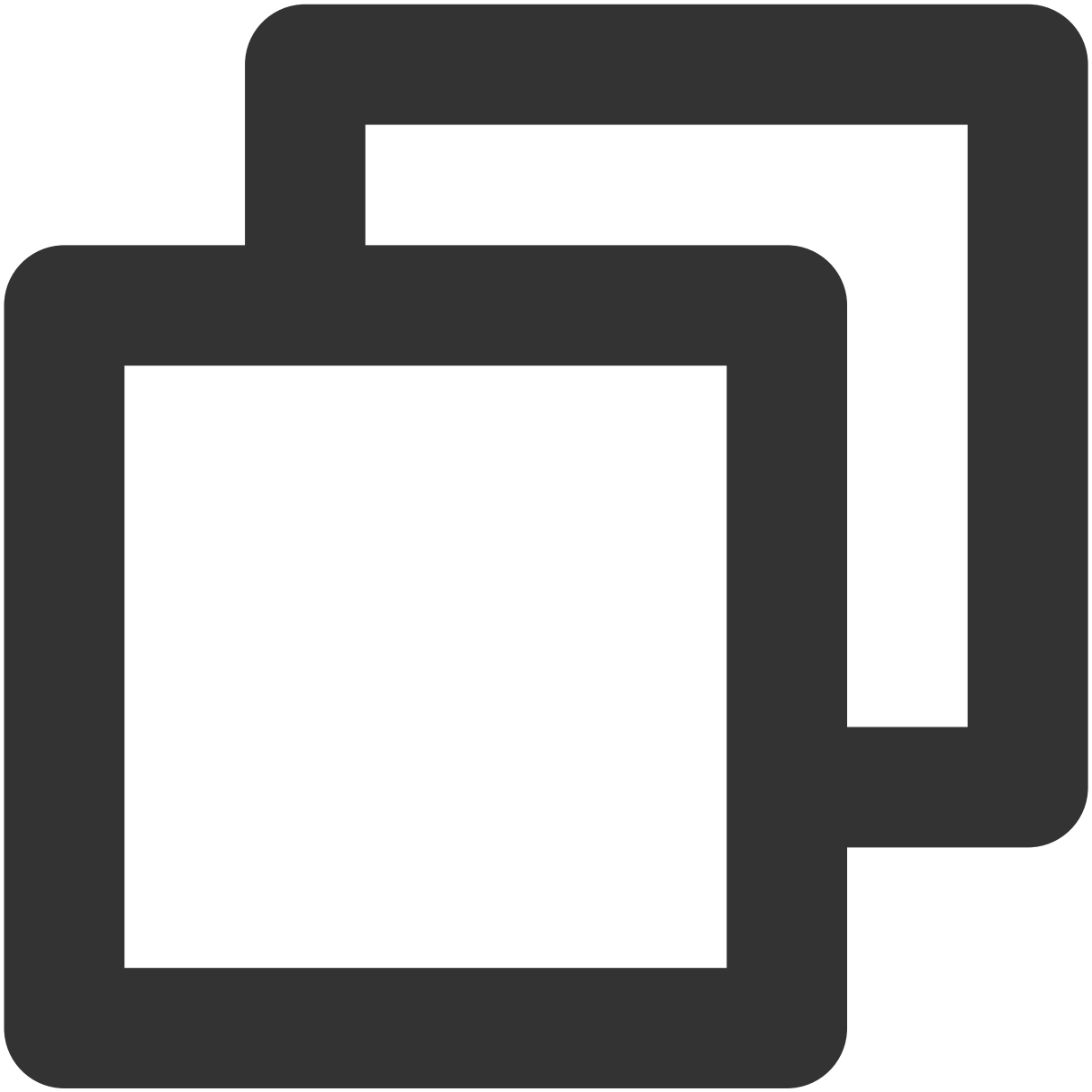
版权所有：腾讯云计算（北京）有限责任公司

```
if (faceMessageBean != null) {
    int faceGroupID = faceMessageBean.getIndex();
    String faceKey = null;
    if (faceMessageBean.getData() != null) {
        faceKey = new String(faceMessageBean.getData());
    }
}
```

渲染自定义表情

调用现有接口渲染

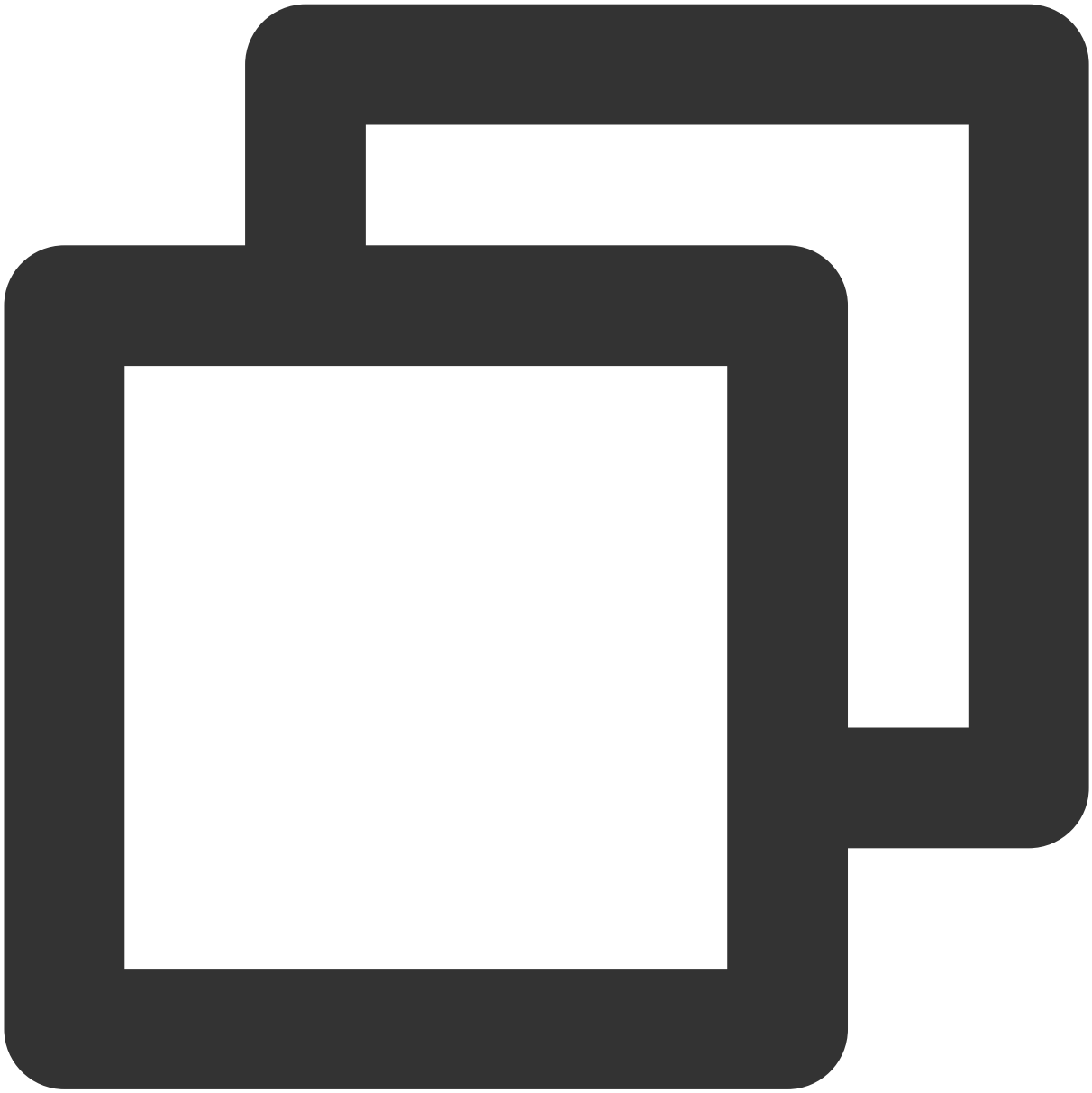
得到自定义表情的 faceGroupID 和 faceKey 之后，可以调用 FaceManager 的 loadFace 方法直接加载到传入的 imageView 上：



```
FaceManager.loadFace(faceGroupID, faceKey, imageView);
```

自定义渲染

也可以通过表情的 faceGroupID 和 faceKey，从 FaceManager 中获取到表情的真实 url，再通过得到的 url 自定义渲染，例如：



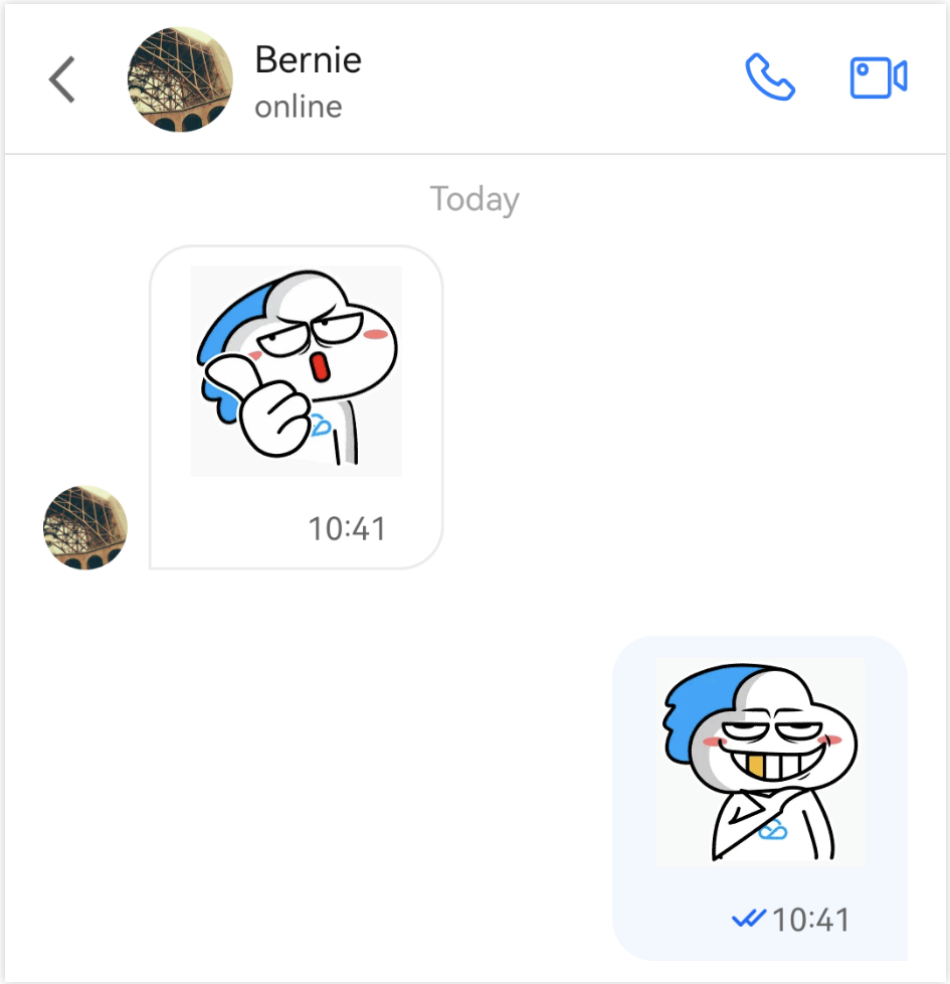
```
String faceUrl = "";
List<FaceGroup> faceGroupList = FaceManager.getFaceGroupList();
for(FaceGroup faceGroup : faceGroupList) {
    if (faceGroup.getGroupID() == faceGroupID) {
        ChatFace face = faceGroup.getFace(faceKey);
        if (face != null) {
            faceUrl = face.getFaceUrl();
        }
    }
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
// load faceUrl into view
```

渲染效果

渲染效果如图所示：



iOS

最近更新时间：2024-05-20 17:05:47

概述

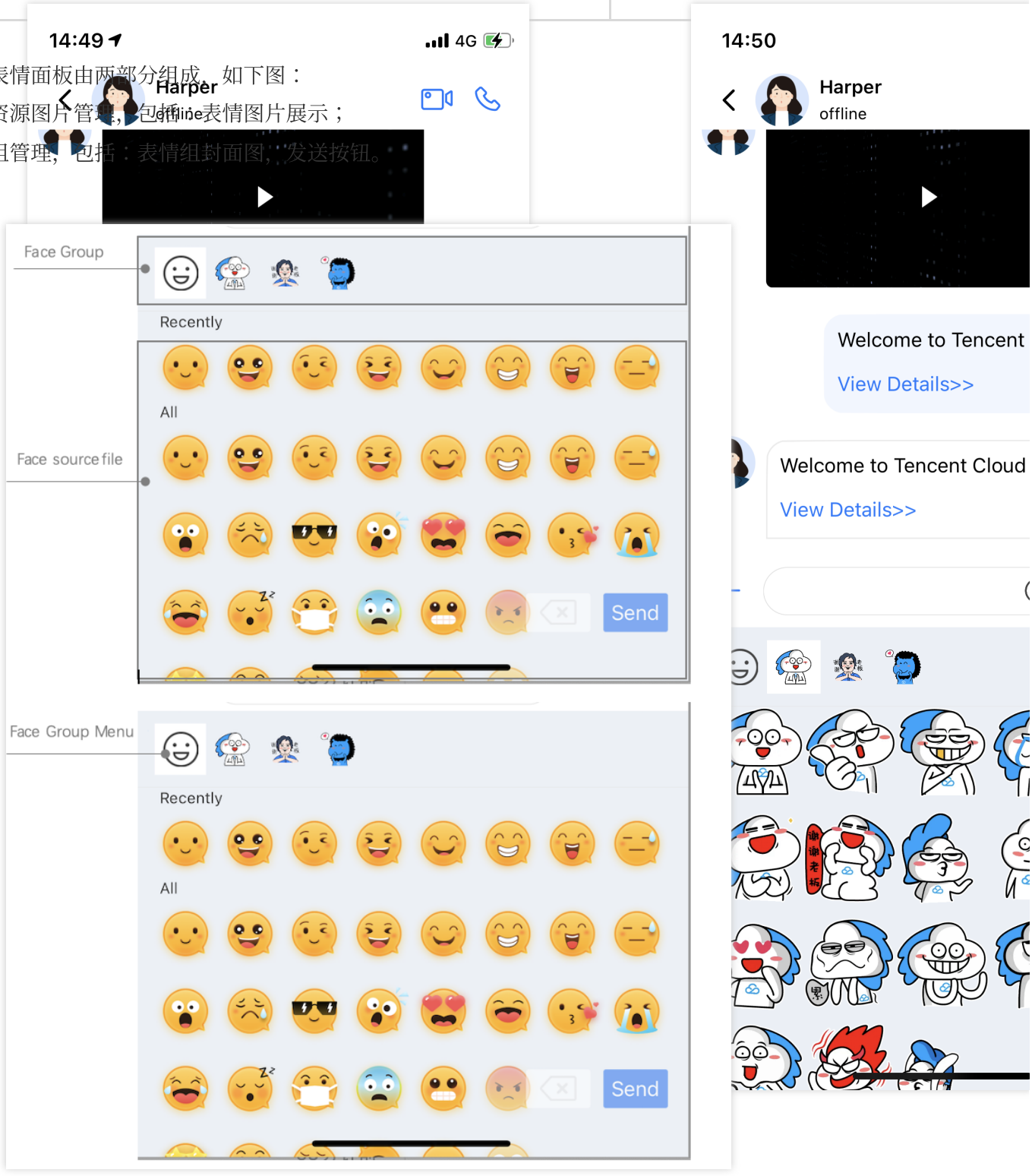
TUIChat 表情面板内置了部分 emoji 小表情，您也可以按需添加自定义表情。本文重点讲解添加自定义表情。

内置小表情面板	自定义表情面板

整个表情面板由两部分组成，如下图：

表情资源图片管理，包括：表情图片展示；

表情组管理，包括：表情组封面图，发送按钮。



新增自定义表情包

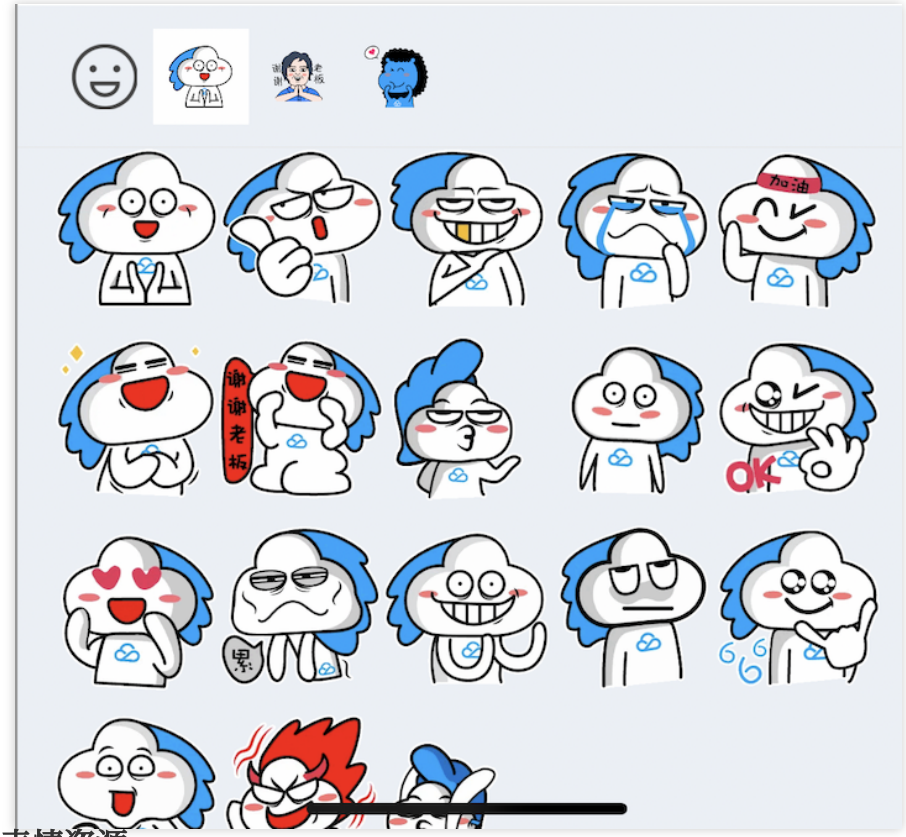
新增一套自定义表情包，您只需要按照如下两个步骤配置即可：

1. 准备表情资源

2. 启动 App 时加载表情包

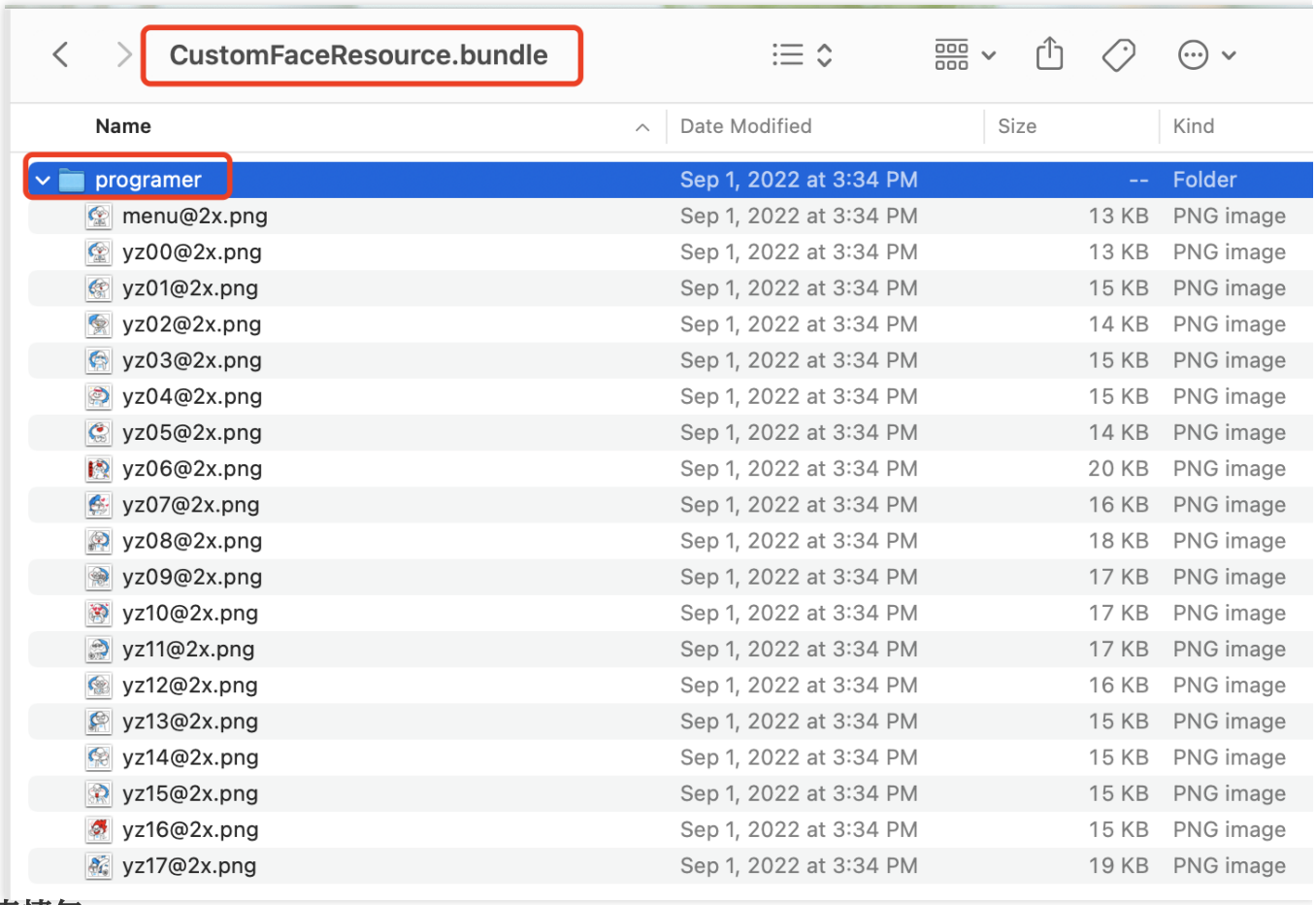
需要说明的是，TUIChat 已经内置了表情包的发送和解析逻辑，您可以很轻松地实现自定义表情包的多端互通。

接下来以“programer”这套自定义表情为例，演示如何添加自定义表情包，如下图。



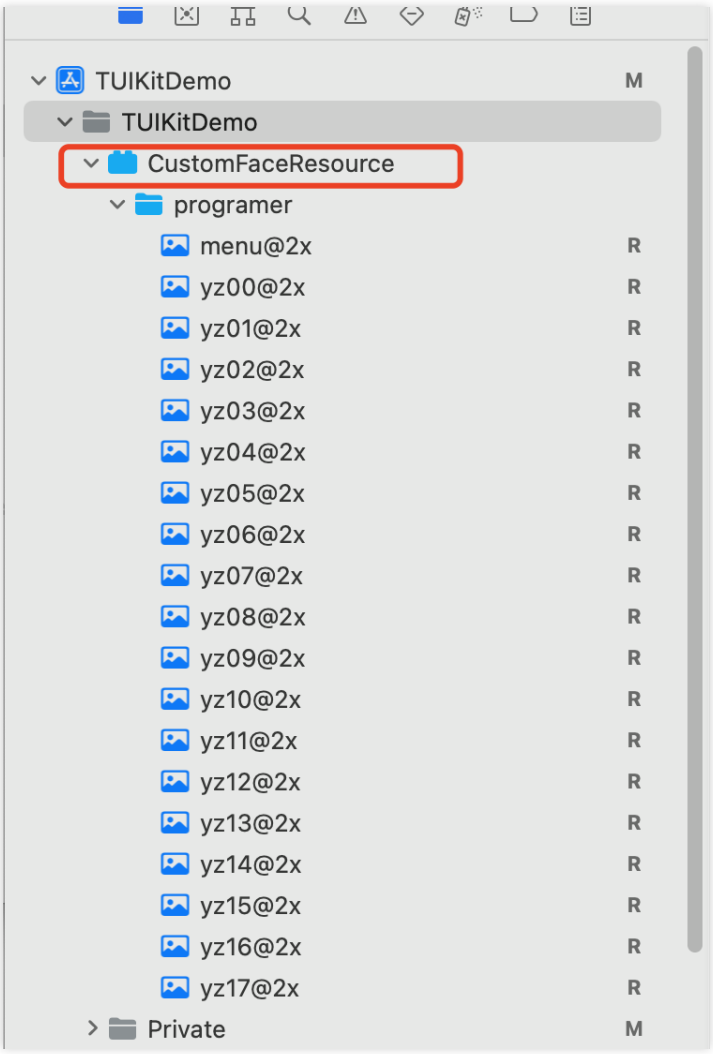
准备表情资源

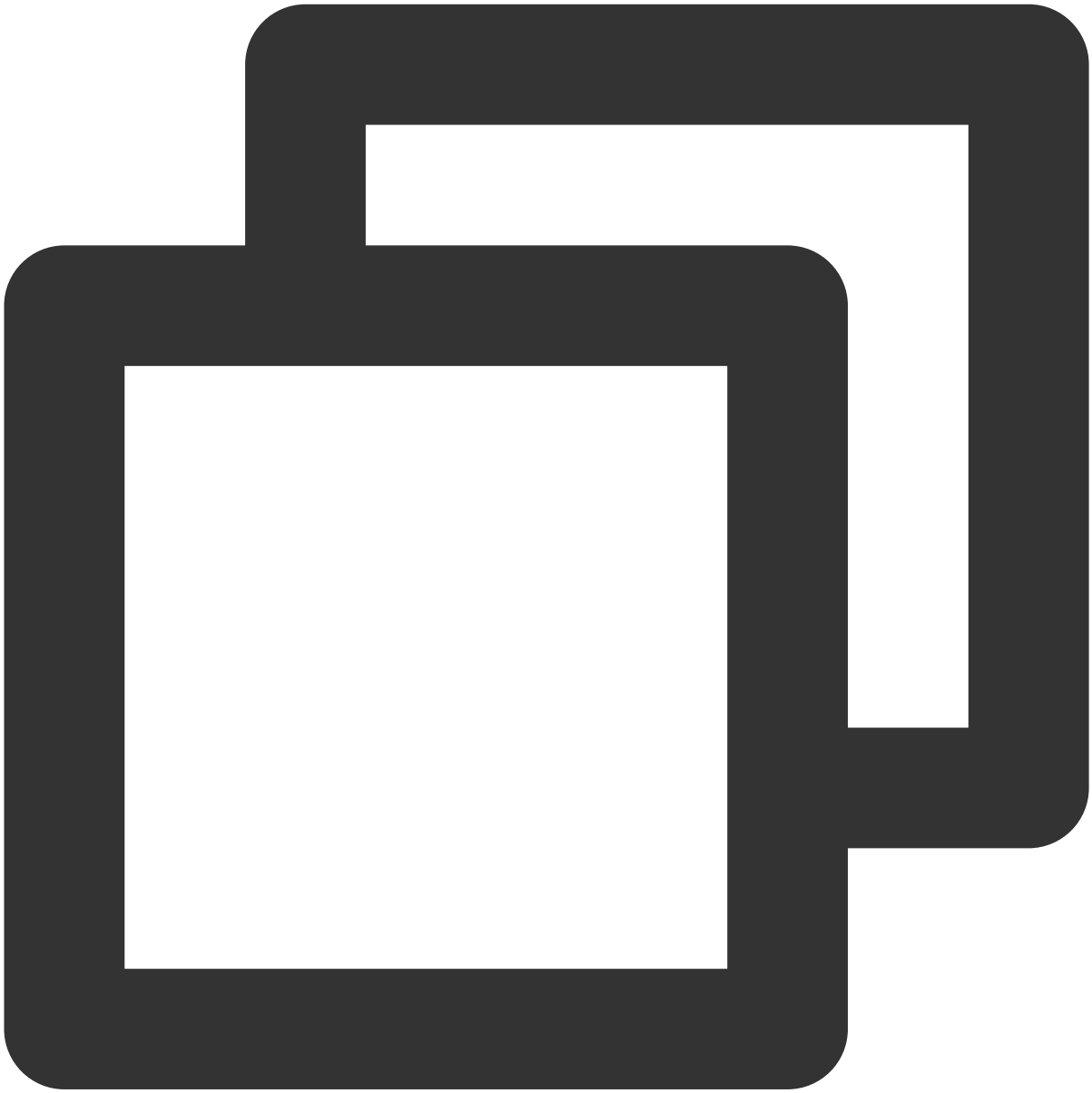
在添加表情包之前，您首先需要准备一套拥有版权的表情资源。如下图，只需要将您的表情图片打包成 bundle 文件即可。



加载表情包

如下图，将含有 “programer” 表情资源的自定义表情包 `CustomFaceResource.bundle` 拖到您的 xcode 工程中。然后在 App 启动时加载即可。


TUIKitDemo
M
TUIKitDemo
CustomFaceResource
programer
menu@2x
R
yz00@2x
R
yz01@2x
R
yz02@2x
R
yz03@2x
R
yz04@2x
R
yz05@2x
R
yz06@2x
R
yz07@2x
R
yz08@2x
R
yz09@2x
R
yz10@2x
R
yz11@2x
R
yz12@2x
R
yz13@2x
R
yz14@2x
R
yz15@2x
R
yz16@2x
R
yz17@2x
R
Private
M



```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    app = self;
    // Load the emoji resources when starting the app
    [self setupCustomSticker];
    return YES;
}

- (void)setupCustomSticker {
    // 1. Get the path of the bundle file of the custom sticker.
    NSString *customFaceBundlePath = [[NSBundle mainBundle] pathForResource:@"Custo
```

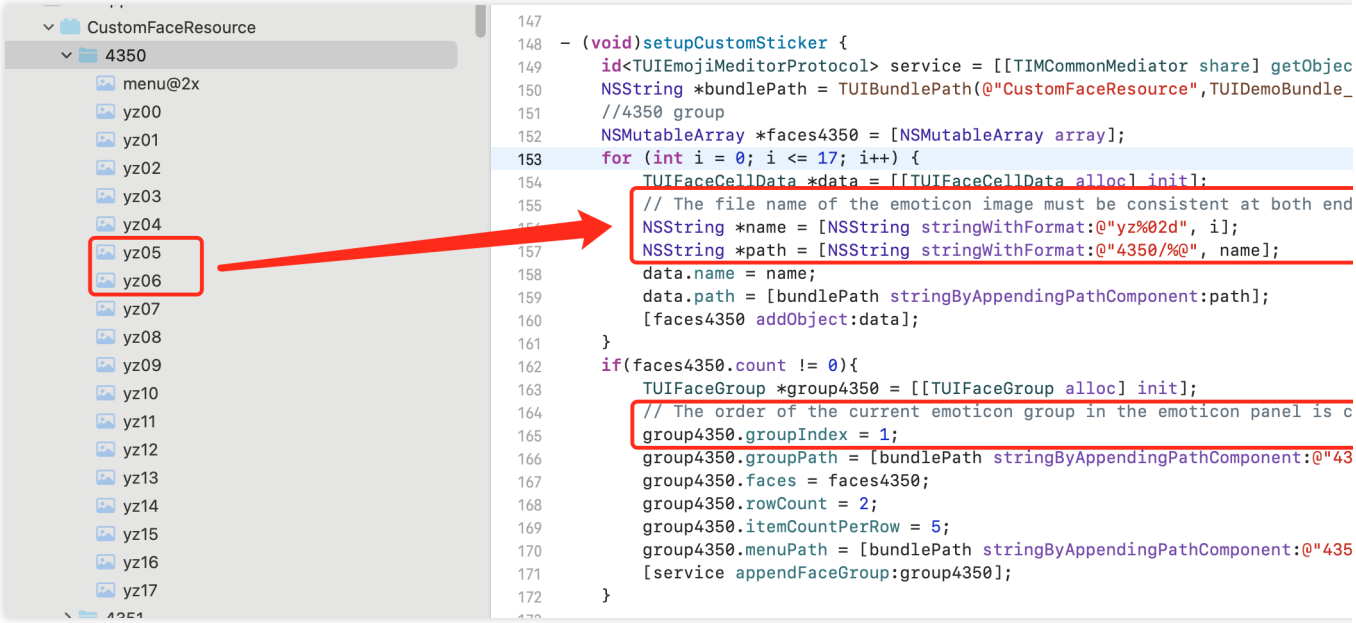
```
    // 2. Load the custom emoji group
    // 2.1 Load the `programer` emoji resource images and parse them into `TUIFaceC
    NSMutableArray<TUIFaceCellData *> *faceItems = [NSMutableArray array];
    for (int i = 0; i <= 17; i++) {
        TUIFaceCellData *data = [[TUIFaceCellData alloc] init];
        // The filename of the emoji resource images (the extension can be saved) f
        data.name = [NSString stringWithFormat:@"yz%02d", i];
        // The path of the emoji resource images for local display
        data.path = [customFaceBundlePath stringByAppendingPathComponent:[NSString
        [faceItems addObject:data];
    }
    // 2.2 Create the `programer` emoji group and parse it into `TUIFaceGroup`
    TUIFaceGroup *programGroup = [[TUIFaceGroup alloc] init];
    // Indicate the serial number of the current emoji group on the emoji panel for
    // Note that `groupIndex` starts from `0` and indicates the actual position of
    programGroup.groupIndex = 1;
    // The root path of the current sticker in the bundle file of the custom emojis
    programGroup.groupPath = [customFaceBundlePath stringByAppendingPathComponent:@
    // The emoji resources in the current sticker
    programGroup.faces = faceItems;
    // The layout of the current sticker
    programGroup.rowCount = 2;
    programGroup.itemCountPerRow = 5;
    // The path of the thumbnail of the current sticker (without the extension)
    programGroup.menuPath = [customFaceBundlePath stringByAppendingPathComponent:@"

    // 3. Add the `programer` emoji group to the emoji panel
    id<TUIEmojiMeditorProtocol> service = [[TIMCommonMediator share] getObject:@pro
    [service appendFaceGroup:programGroup];
}
```

多端互通

TUIChat 已经内置了表情包发送和解析逻辑，您只需要将如下两个属性在各个平台保持一致即可：
表情包中的图片文件名一致，也即 App 启动加载表情包时解析成 `TUIFaceCellData` 的 `name` 字段值需要多端一致；
表情包在表情面板中的顺序一致，也即 App 启动加载表情包时解析成 `TUIFaceGroup` 的 `groupIndex` 字段值需要多端一致。
当上述两个信息一致后，TUIChat 内置的表情包发送逻辑会将表情文件名和所属的表情包索引信息发给其他端，从而实现多端互通。
需要注意的是，groupIndex 是从 0 开始的，标识了当前表情包在表情面板中的实际位置（内置的 emoji 表情组默认是 0）。

版权所有：腾讯云计算（北京）有限责任公司



表情面板高级配置

调整表情面板顺序

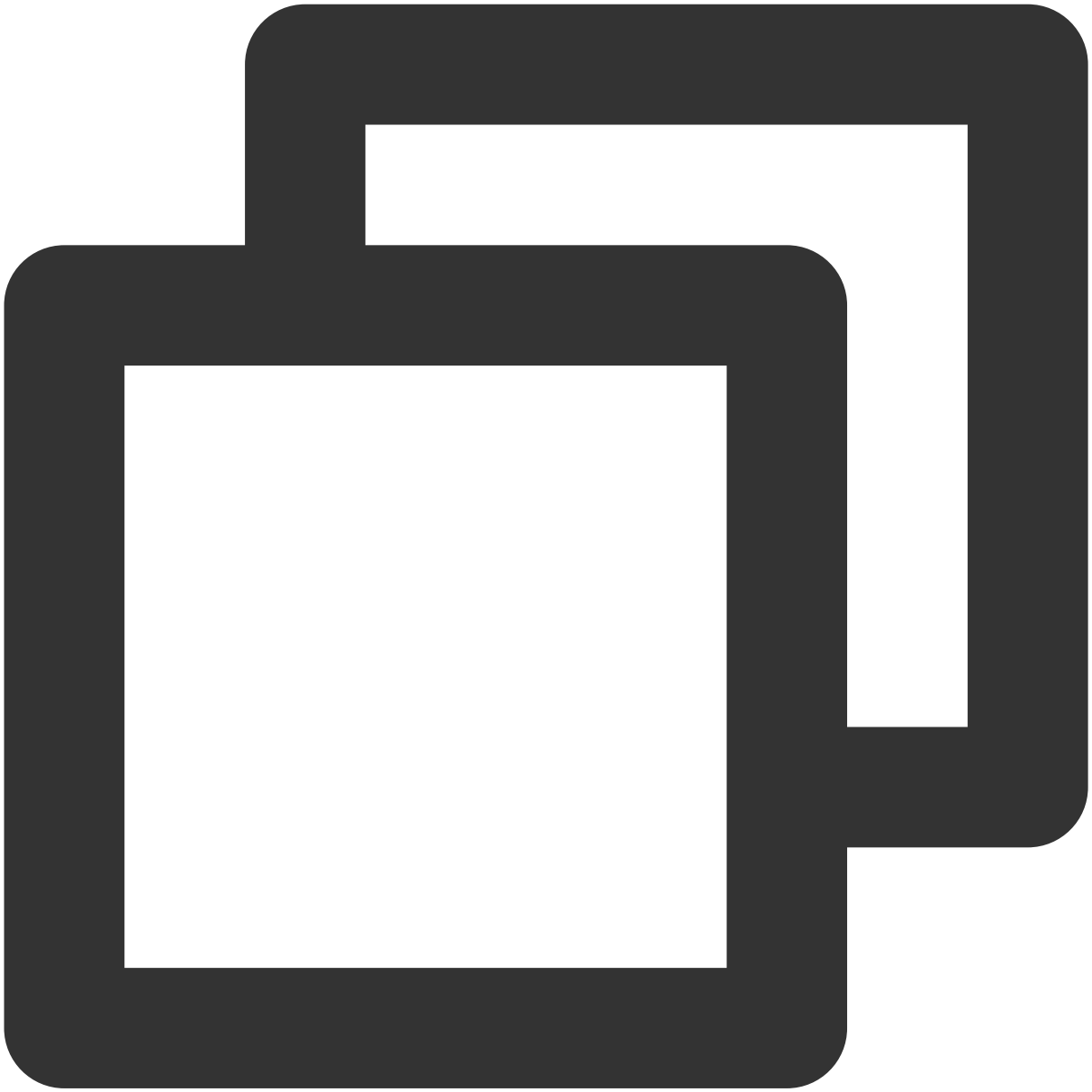
TUIChat 的表情面板支持调整表情组的顺序，您只需要按照实际顺序调用 `TUIConfig` 的 `-appendFaceGroup:` 方法即可。

如果您想将内置 emoji 表情组调整到自定义表情后面，需要按照如下方式操作：

获取当前表情面板内置的表情组 `TUIConfig.defaultConfig.faceGroups` ；

重新排序；

将已经排好序的表情组列表赋值给表情面板。



```
- (void)setupCustomSticker {
    // 1. Get the path of the bundle file of the custom sticker.
    NSString *customFaceBundlePath = [[NSBundle mainBundle] pathForResource:@"Custo
    // 2. Load the custom emoji group
    // 2.1 Load the `programer` emoji resource images and parse them into `TUIFaceC
    NSMutableArray<TUIFaceCellData *> *faceItems = [NSMutableArray array];
    for (int i = 0; i <= 17; i++) {
        TUIFaceCellData *data = [[TUIFaceCellData alloc] init];
        // The filename of the emoji resource images (the extension can be saved) f
        data.name = [NSString stringWithFormat:@"yz%02d", i];
```

```
        // The path of the emoji resource images for local display
        data.path = [customFaceBundlePath stringByAppendingPathComponent:[NSString
        [faceItems addObject:data];
    }
    // 2.2 Create the `programer` emoji group and parse it into `TUIFaceGroup`
    TUIFaceGroup *programGroup = [[TUIFaceGroup alloc] init];
    // Indicate the serial number of the current emoji group on the emoji panel for
    // Note that `groupIndex` starts from `0` and indicates the actual position of
    programGroup.groupIndex = 0;
    // The root path of the current sticker in the bundle file of the custom emojis
    programGroup.groupPath = [customFaceBundlePath stringByAppendingPathComponent:@
    // The emoji resources in the current sticker
    programGroup.faces = faceItems;
    // The layout of the current sticker
    programGroup.rowCount = 2;
    programGroup.itemCountPerRow = 5;
    // The path of the thumbnail of the current sticker (without the extension)
    programGroup.menuPath = [customFaceBundlePath stringByAppendingPathComponent:@"

    // 3. Add the `programer` emoji group to the front of the emoji panel
    id<TUIEmojiMeditorProtocol> service = [[TIMCommonMediator share] getObject:@pro
    [service appendFaceGroup:programGroup];
}
```

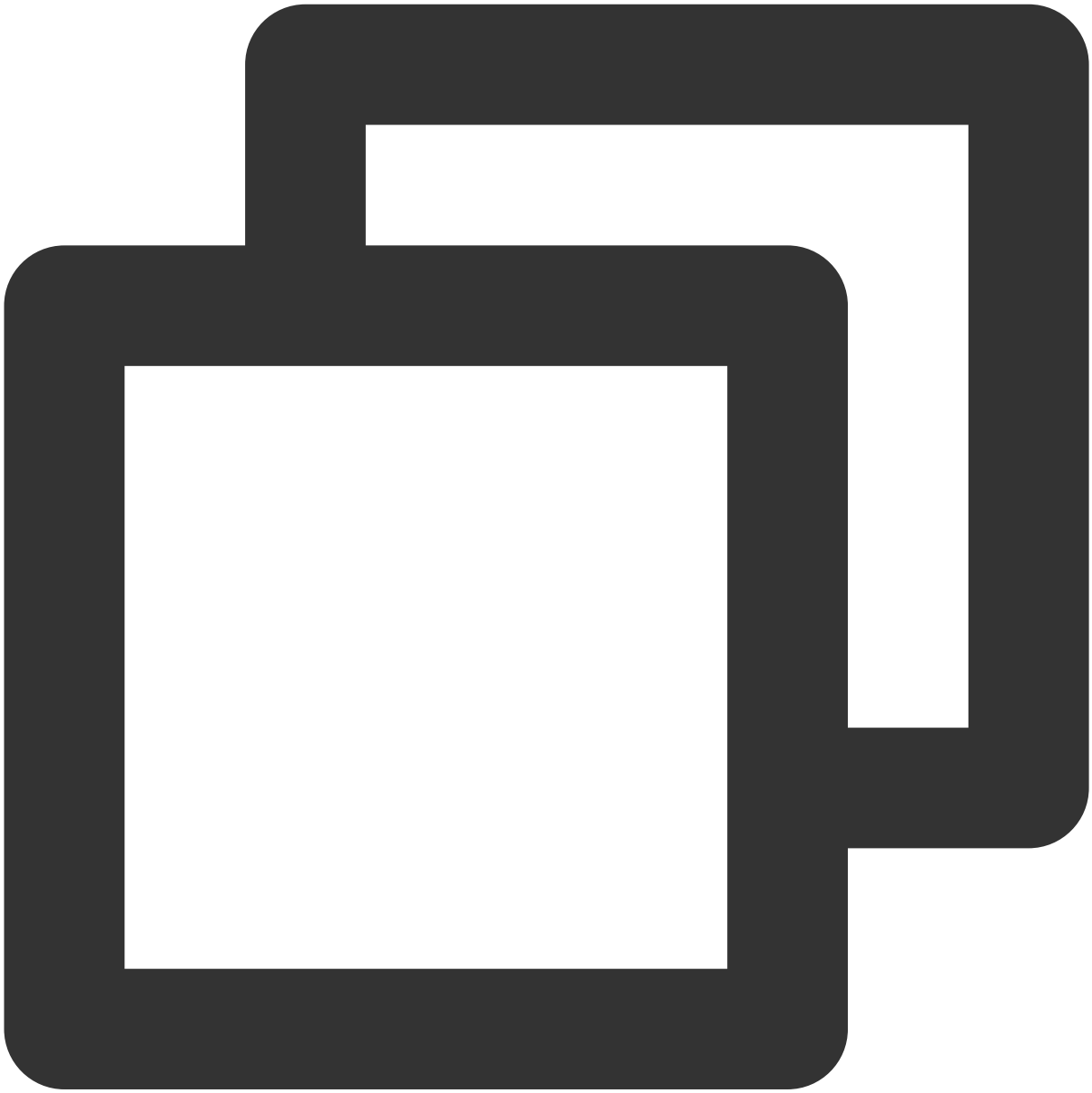
说明：

由于表情包多端互通依赖于表情图片的名称和表情组所在面板的顺序，当您调整本地顺序之后，需要保证 groupIndex 与您实际顺序一致，方便各端互通。

修改表情组封面

您可以在加载自定义表情组时，给 `TUIFaceGroup` 的 `menuPath` 属性设置封面图的路径（无需 @2x.png 的扩展名）来自定义表情组封面。

例如，将 "programer" 表情组中的 `menu@2x.png` 图片作为封面图片。



```
- (void)setupCustomSticker {
    ....

    // 2.2 Create the `programer` emoji group and parse it into `TUIFaceGroup`
    TUIFaceGroup *programGroup = [[TUIFaceGroup alloc] init];
    ....
    // The path of the thumbnail of the current sticker (without the extension)
    programGroup.menuPath = [customFaceBundlePath stringByAppendingPathComponent:@"
    ....

    ....
```

```
}
```

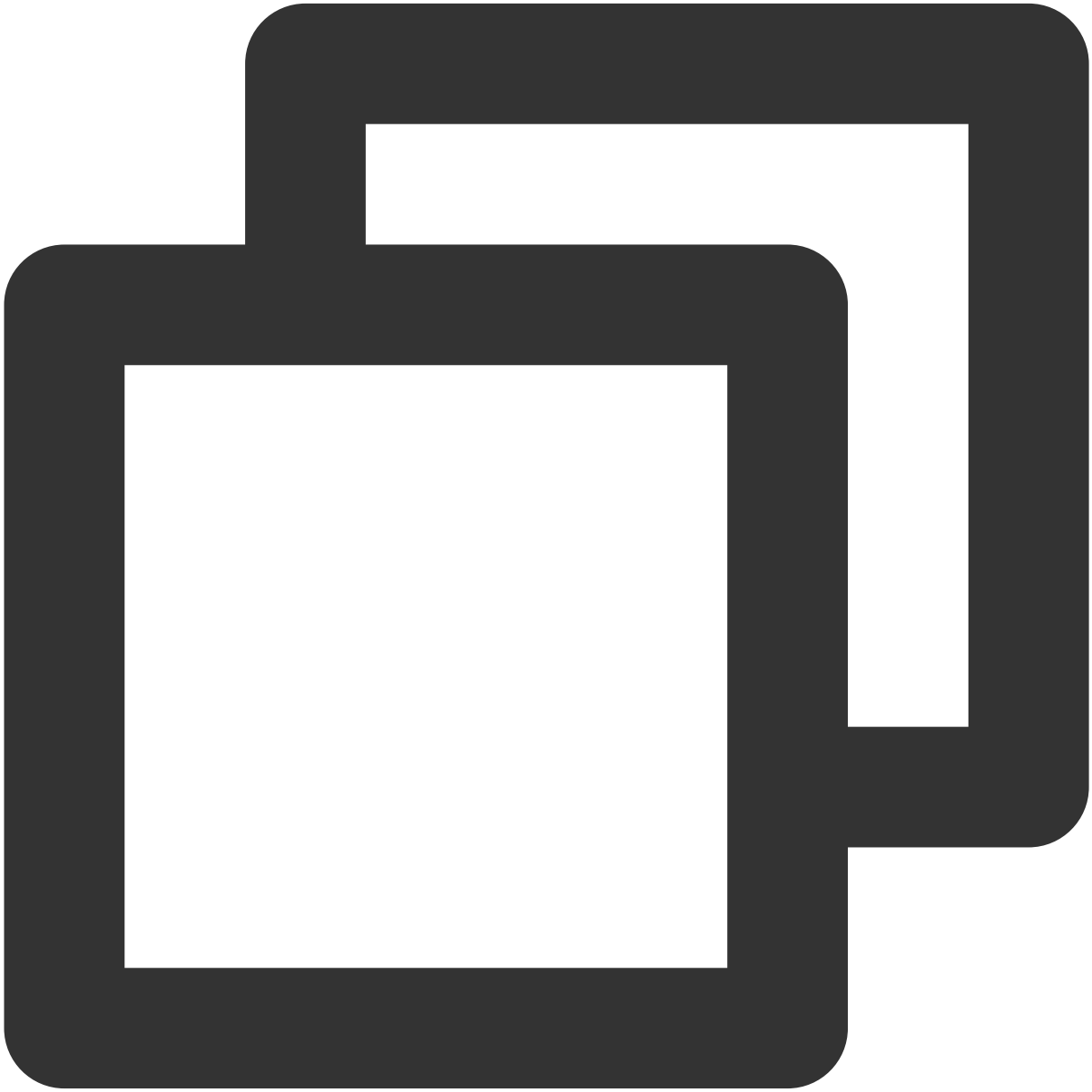
调整表情图片的布局

目前 TUIChat 表情面板针对表情图片的布局，支持以下两个样式：

rowCount，当前表情组内图片显示的行数；

itemCountPerRow，每行展示的表情图片的个数。

例如，调整 “programer” 表情组中的表情图片排列规则是每页 2 行，每行最多 5 张图片。



```
- (void)setupCustomSticker {
```

```
    ...

    // 2.2 Create the `programer` emoji group and parse it into `TUIFaceGroup`
    TUIFaceGroup *programGroup = [[TUIFaceGroup alloc] init];
    // The layout of the current sticker
    programGroup.rowCount = 2;
    programGroup.itemCountPerRow = 5;

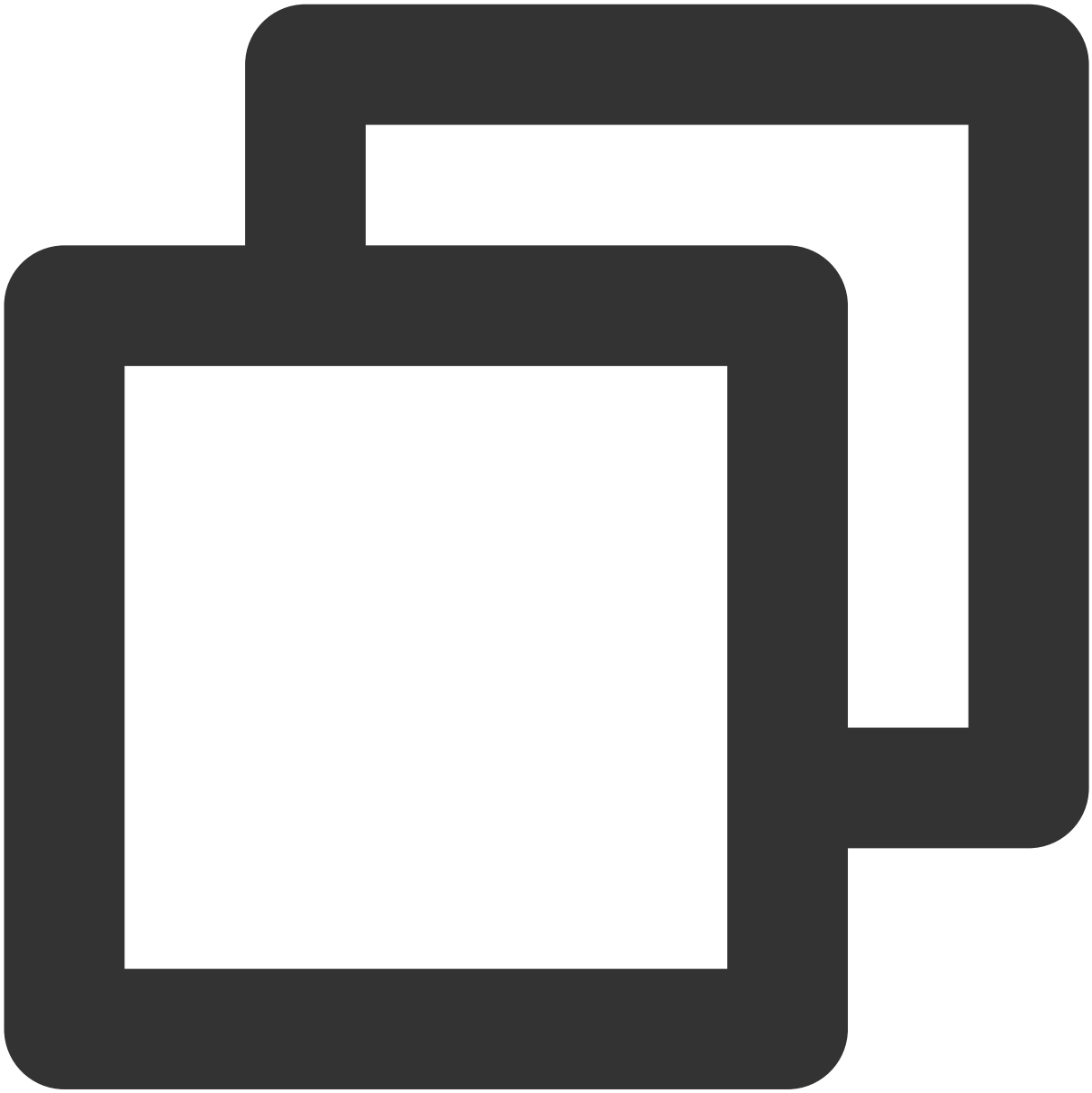
    ...
}
```

表情包渲染原理

TUIChat 内置了表情包的发送和渲染机制，您无需关注本部分内容。
如果您想修改源码，或者需要将自定义表情内容编码后直接透传，可以参考该部分。

发送表情

TUIChat 的表情面板由 UICollectionView 组成，当点击每个表情图片后会触发 `TUIInputController` 的 `-faceView:didSelectItemAtIndexPath:` 方法，并将您点选的表情名称和对应表情组在面板中的索引信息回调给您。
您可以在回调中通过两个步骤将表情发送出去：
使用表情名称和表情组索引创建表情消息；
调用 TUIChat 的方法将表情消息发送出去。

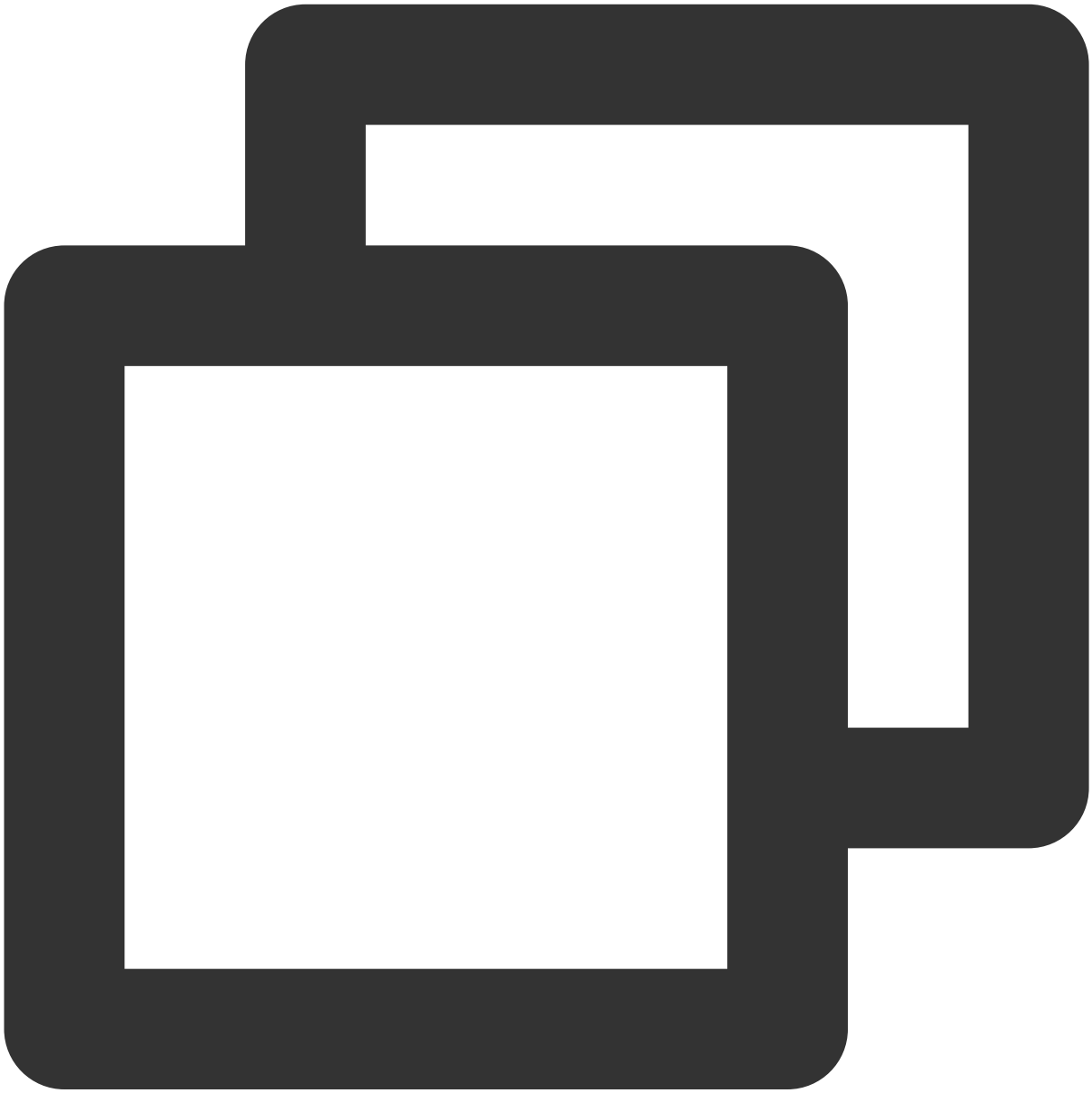


```
- (void)faceView:(TUIFaceView *)faceView didSelectItemAtIndexPath:(NSIndexPath *)in
{
    TUIFaceGroup *group = [TUIConfig defaultConfig].faceGroups[indexPath.section];
    TUIFaceCellData *face = group.faces[indexPath.row];
    if(indexPath.section == 0){
        // Built-in emojis need to be displayed in the input box.
        [_inputBar addEmoji:face];
    }
    else{
        // Custom emojis are directly sent to the receiver.
        if (face.name) {
```

```
            // Create an emoji message
            V2TIMMessage *message = [[V2TIMManager sharedInstance] createFaceMessag
            // Send the message to receiver
            if(_delegate && [_delegate respondsToSelector:@selector(inputController
                [_delegate inputController:self didSendMessage:message];
            }
        }
    }
}
```

解析表情并渲染

当收到对端的表情消息后，TUIChat 会触发 `TUIFaceMessageCellData` 的 `- getCellData:` 方法，并在其中将表情消息解析成用于展示表情的 `TUIFaceMessageCellData` 。
TUIChat 会将解析到的 `TUIMessageCellData` 赋值给 `TUIFaceMessageCell` 用于渲染。
关于整个 TUIChat 的消息解析流程可以参见 含 UI 集成方案 - 添加自定义消息 。



```
+ (TUIMessageCellData *)getCellData:(V2TIMMessage *)message{
    // Parse the emoji information after receiving the message
    V2TIMFaceElem *elem = message.faceElem;

    // Create the `TUIFaceMessageCellData` for emoji display
    TUIFaceMessageCellData *faceData = [[TUIFaceMessageCellData alloc] initWithDire
    // Get the order information of the current emoji group on the emoji panel
    faceData.groupIndex = elem.index;
    // Get the filename of the emoji image
    faceData.faceName = [[NSString alloc] initWithData:elem.data encoding:NSUTF8Str
    // Get the specific path of the local sticker of the emoji image based on the n
```

```
    for (TUIFaceGroup *group in [TUIConfig defaultConfig].faceGroups) {
        if(group.groupIndex == faceData.groupIndex){
            NSString *path = [group.groupPath stringByAppendingPathComponent:faceDa
            faceData.path = path;
            break;
        }
    }
    faceData.reuseId = TFaceMessageCell_ReuseId;
    return faceData;
}
```

Web & H5 & Uniapp

最近更新时间：2024-06-19 11:42:12

说明：

本文所述的自定义表情能力用法适用于 **@tencentcloud/chat-uikit-vue ≥ 2.2.0** 或 **@tencentcloud/chat-uikit-uniapp ≥ 2.2.0**。

我们的 `TUIChat` 组件中，支持发送及接收两种类型的表情：

表情类型	发送形式	是否文字混排	发送内容	TUIKit 默认自带
小表情	文本消息 MSG_TEXT	是	表情图片Key	默认包含
大表情	表情消息 MSG_FACE	否	index: 表情组emojiGroupID data：表情图片Key	默认包含

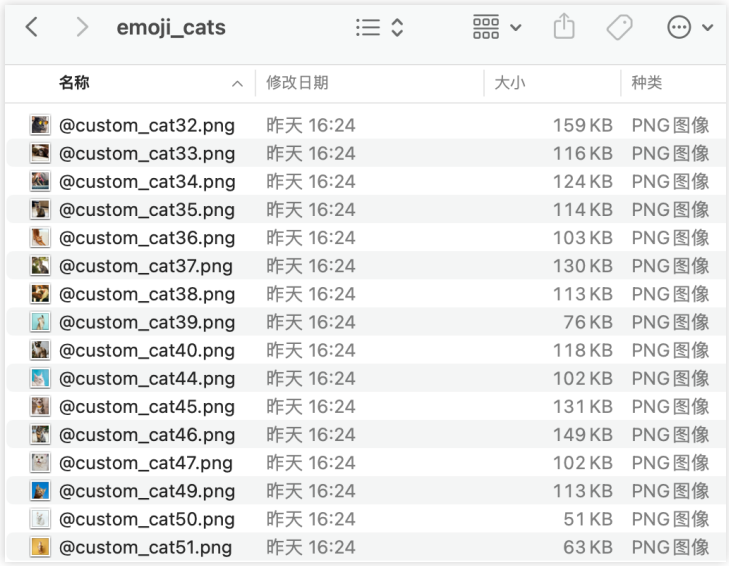
具体效果如下：

自定义大表情

TUIChat 支持从**网络路径**加载自定义大表情。下文中将以添加一组猫咪表情包为例讲解如何新增一组您的自定义大表情。

步骤1：准备大表情资源

1. 请您为每个表情包资源文件命名为包含 `@custom` 格式标识符的文件名：TUIKit 中需要通过匹配 `@custom` 来解析自定义表情资源。
2. 因为表情包资源的解析方式为 `URL + 表情图片Key` ，所以请务必将您要新增的表情包资源上传至统一的网络路径 `URL` 下。



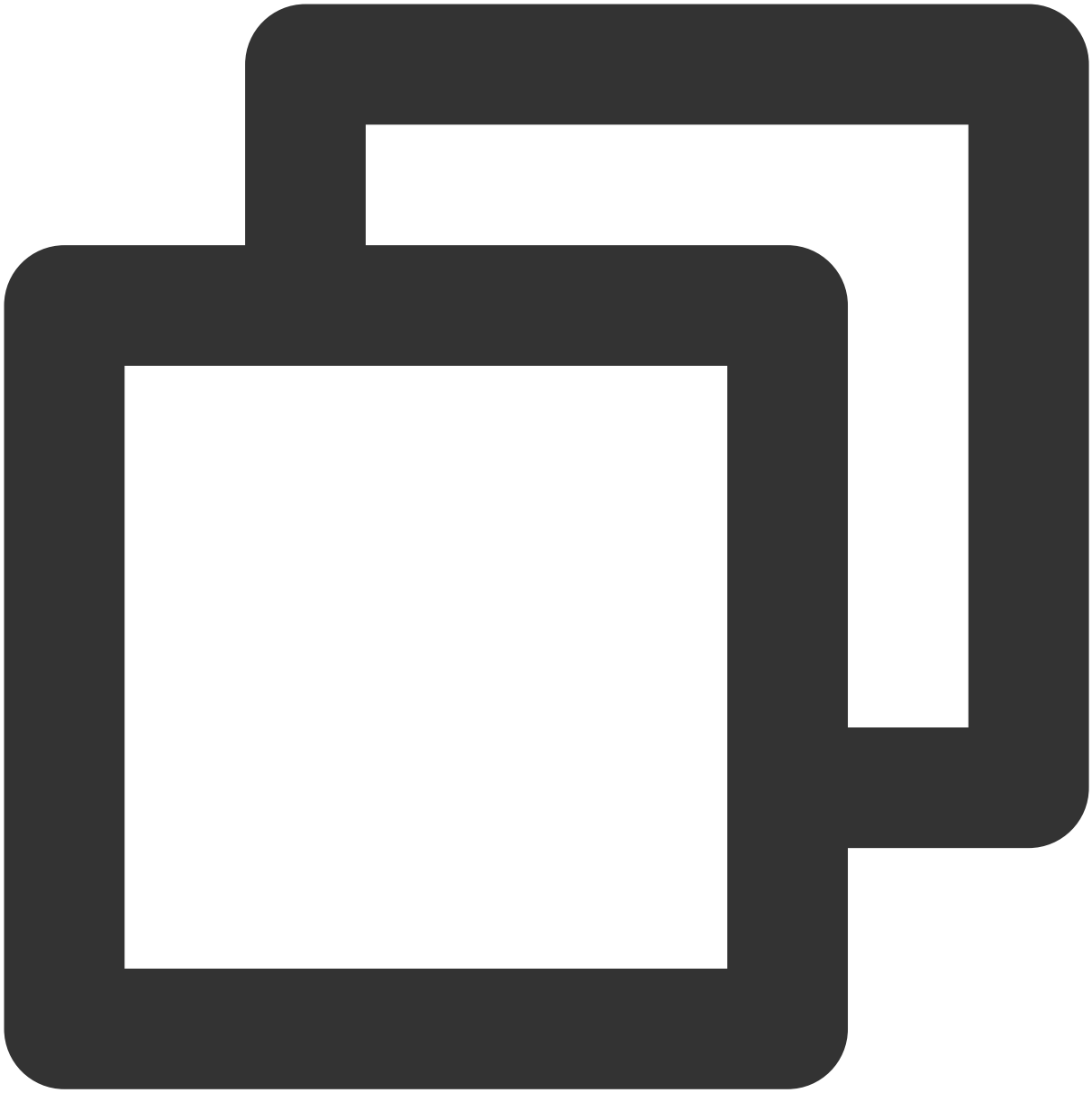
步骤2：添加大表情包

在您的表情包资源均已上传至网络后，需要在 `TUIKit 源码` 中对新增表情包资源增加相关声明。

`TUIKit/components/TUIChat/emoji-config/custom-emoji.ts` 文件是自定义表情的声明文件。

注意：

如您有多端互通需求，请您务必保证每组自定义表情的 emojiGroupID 与其中每个表情的 key 在所有端定义一致。



```
import { EMOJI_TYPE } from '../../../constant';

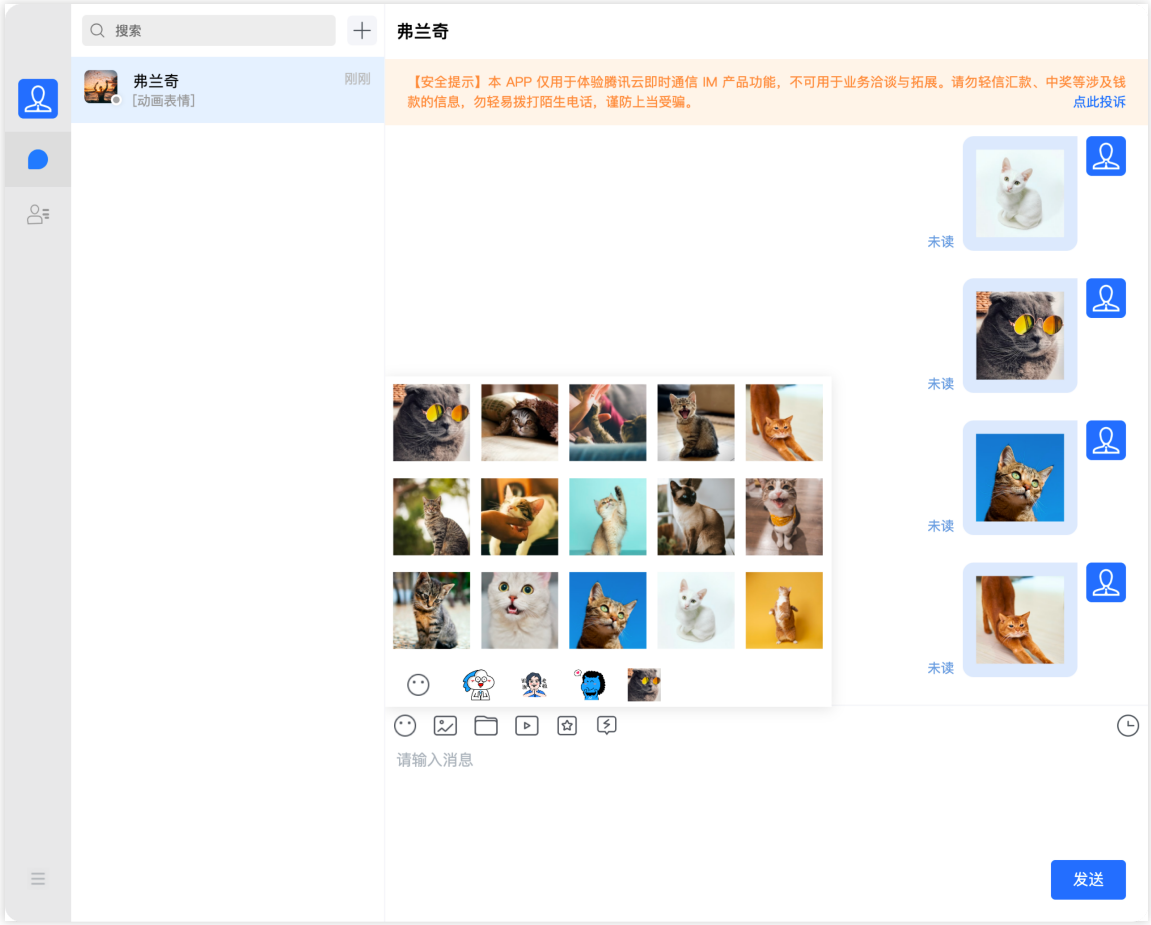
// 填写您的自定义大表情网络路径，示例如下
export const CUSTOM_BIG_EMOJI_URL = 'https://web.sdk.qcloud.com/im/test/emoji-test/

// 声明您的自定义大表情列表组，示例如下
export const CUSTOM_BIG_EMOJI_GROUP_LIST: IEmojiGroupList = [
  {
    emojiGroupID: 4, // 表情组 ID
    type: EMOJI_TYPE.CUSTOM, // 表情组类型：CUSTOM 自定义表情组
    url: CUSTOM_BIG_EMOJI_URL, // 表情组公共 Url 前缀
```

```
    // 表情列表声明，格式为 Key + 表情文件类型后缀，请务必保证列表中每个 Key 保持唯一性
    list: ['@custom_cat32.png', '@custom_cat33.png', '@custom_cat34.png', '@custom_
      '@custom_cat40.png', '@custom_cat45.png', '@custom_cat46.png', '@custom_cat47
  },
];
```

步骤3：渲染自定义大表情

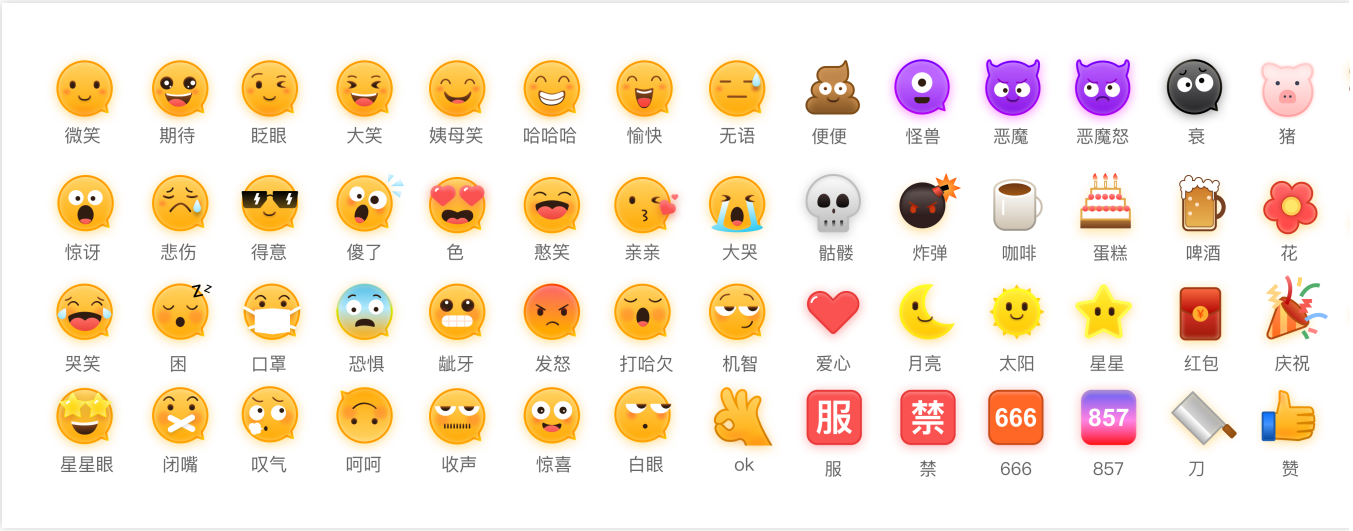
TUIKit 内部内置自定义大表情解析功能，完成上述步骤，您将得到如下效果：



自定义小表情

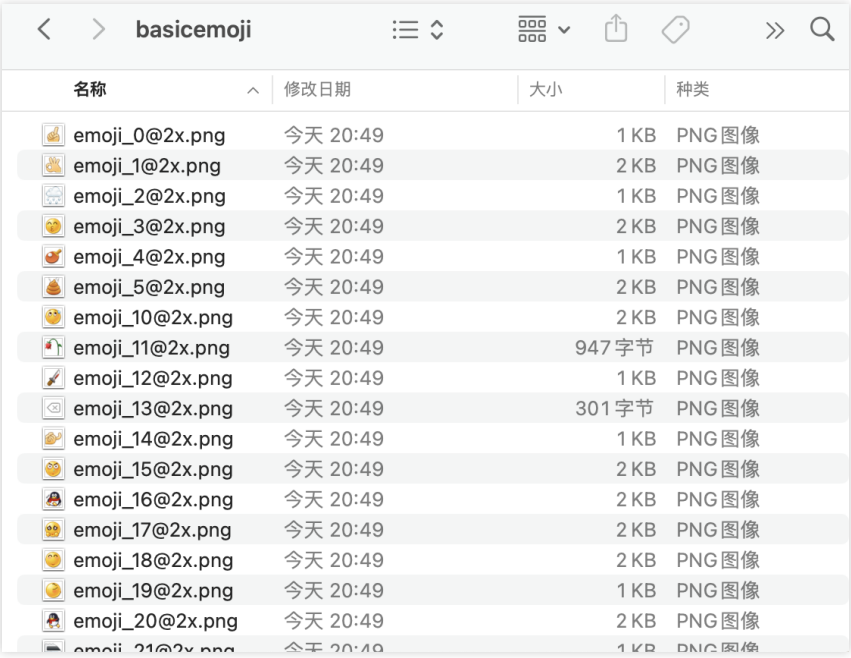
内置小表情资源

TUIKit 默认包含一套黄脸表情资源。黄脸表情版权归腾讯云所有，如需授权使用，请您联系我们。黄脸表情图片与含义对照如下：



步骤1：准备小表情资源

因为表情包资源的解析方式为 `baseURL + 表情图片Key` ，所以请您务必将要新增的表情包资源上传至统一的网络路径 `baseURL` 下。



步骤2：替换小表情资源

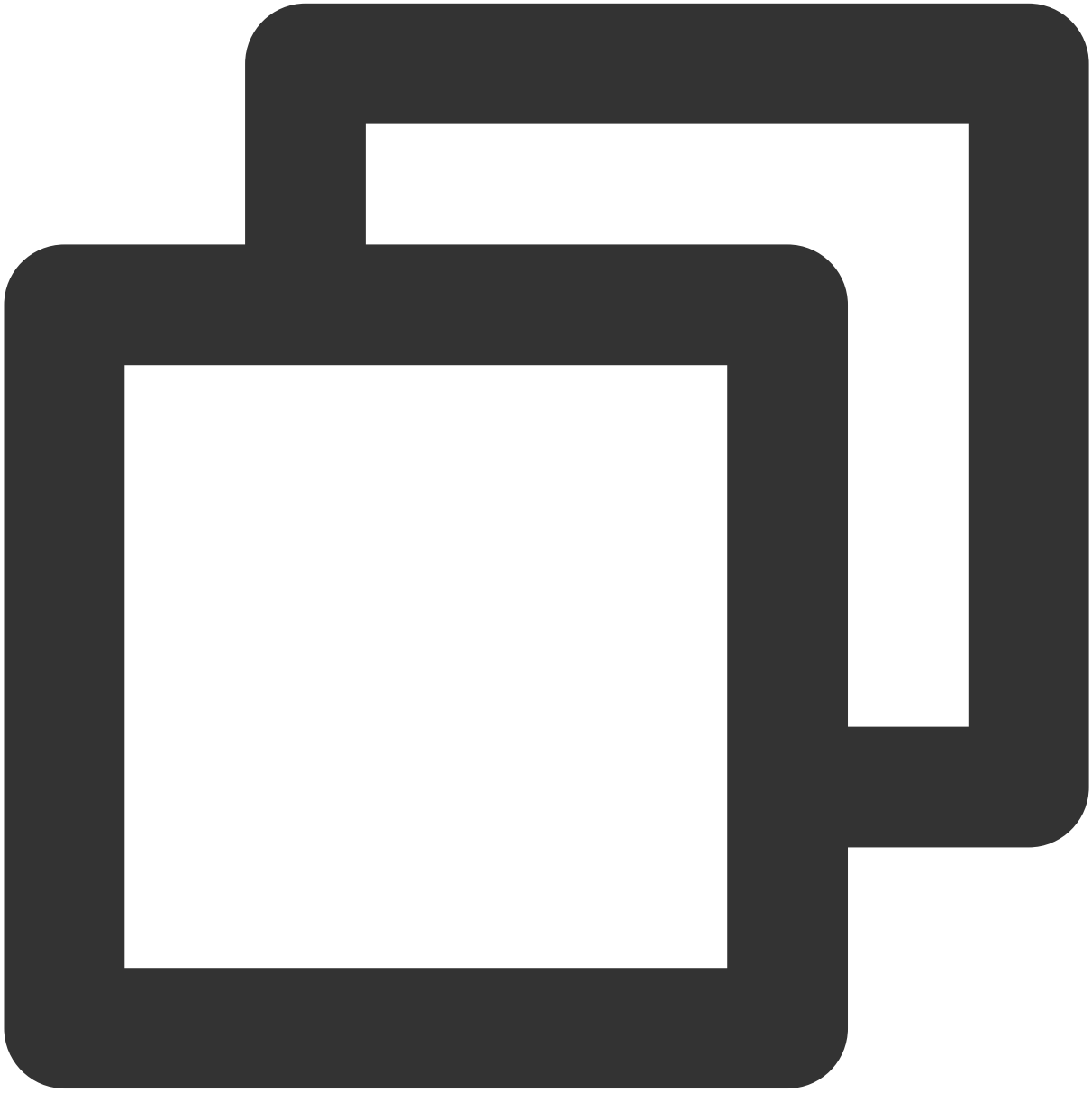
2.1 添加自定义小表情声明

因为 TUIKit 中需要通过匹配 `@custom` 来解析自定义表情资源，**每个自定义小表情资源 key 必须包含 `@custom` 格式标识符。**

目标文件目录：`TUIKit/components/TUIChat/emoji-config/custom-emoji.ts`

注意：

如您有多端互通需求，请务必保证每个自定义小表情的 key 在所有端定义一致。



```
// 填写您的自定义小表情网络路径，示例如下
// export const CUSTOM_BASIC_EMOJI_URL = 'https://web.sdk.qcloud.com/im/assets/emoj
export const CUSTOM_BASIC_EMOJI_URL = '';

// 配置 CUSTOM_BASIC_EMOJI_URL_MAPPING：
// CUSTOM_BASIC_EMOJI_URL_MAPPING 记录每个 emojiKey 与其对应表情图片 url 的映射关系，url
// 每个 emojiKey 必须包含 @custom 标识才能解析，示例如下
```

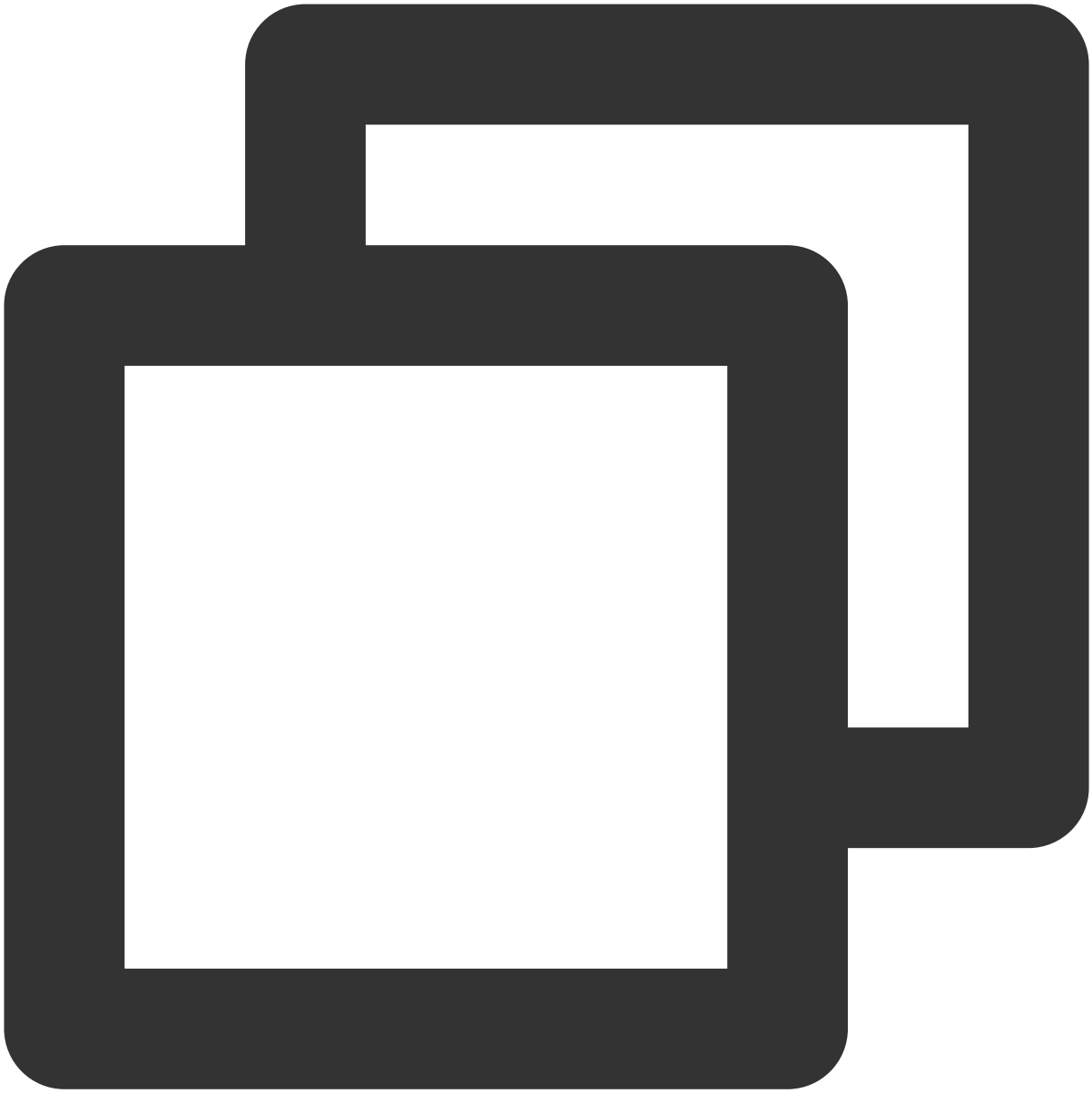
版权所有：腾讯云计算（北京）有限责任公司

```
export const CUSTOM_BASIC_EMOJI_URL_MAPPING = {
  '[@custom_Expect]': 'emoji_0@2x.png',
  '[@custom_Blink]': 'emoji_1@2x.png',
  '[@custom_Guffaw]': 'emoji_2@2x.png',
  '[@custom_KindSmile]': 'emoji_3@2x.png',
  ...
}
```

2.2 为自定义小表情添加国际化文本声明

添加中文含义 key-value 对照声明

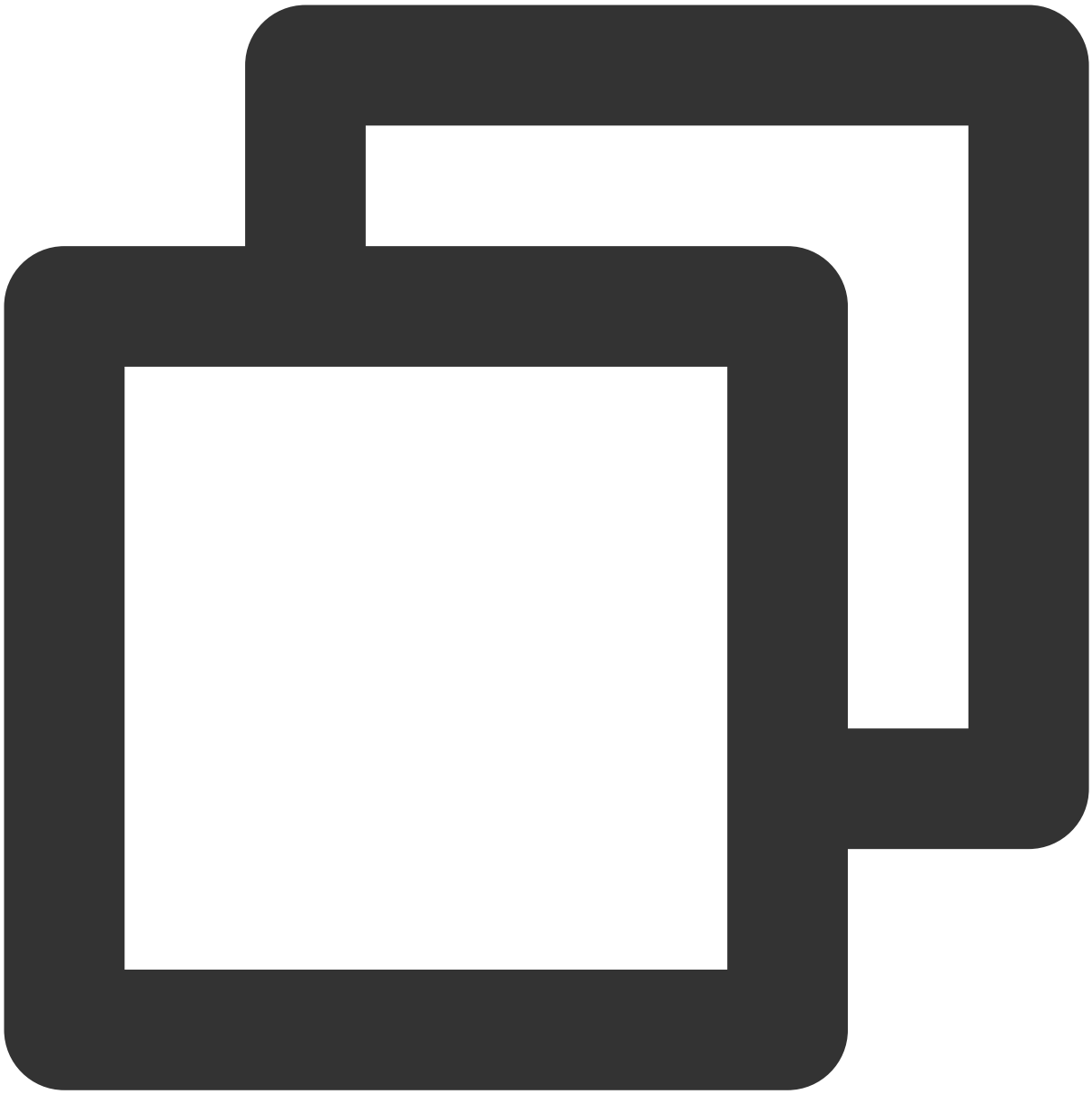
目标文件目录：`TUIKit/components/TUIChat/emoji-config/locales/zh_cn.ts`



```
// 重写中文词条 key-value 声明，示例如下
const Emoji = {
  '[@custom_Expect]': '[期待]',
  '[@custom_Blink]': '[眨眼]',
  '[@custom_Guffaw]': '[大笑]',
  ...
 }
```

添加英文含义 key-value 对照声明

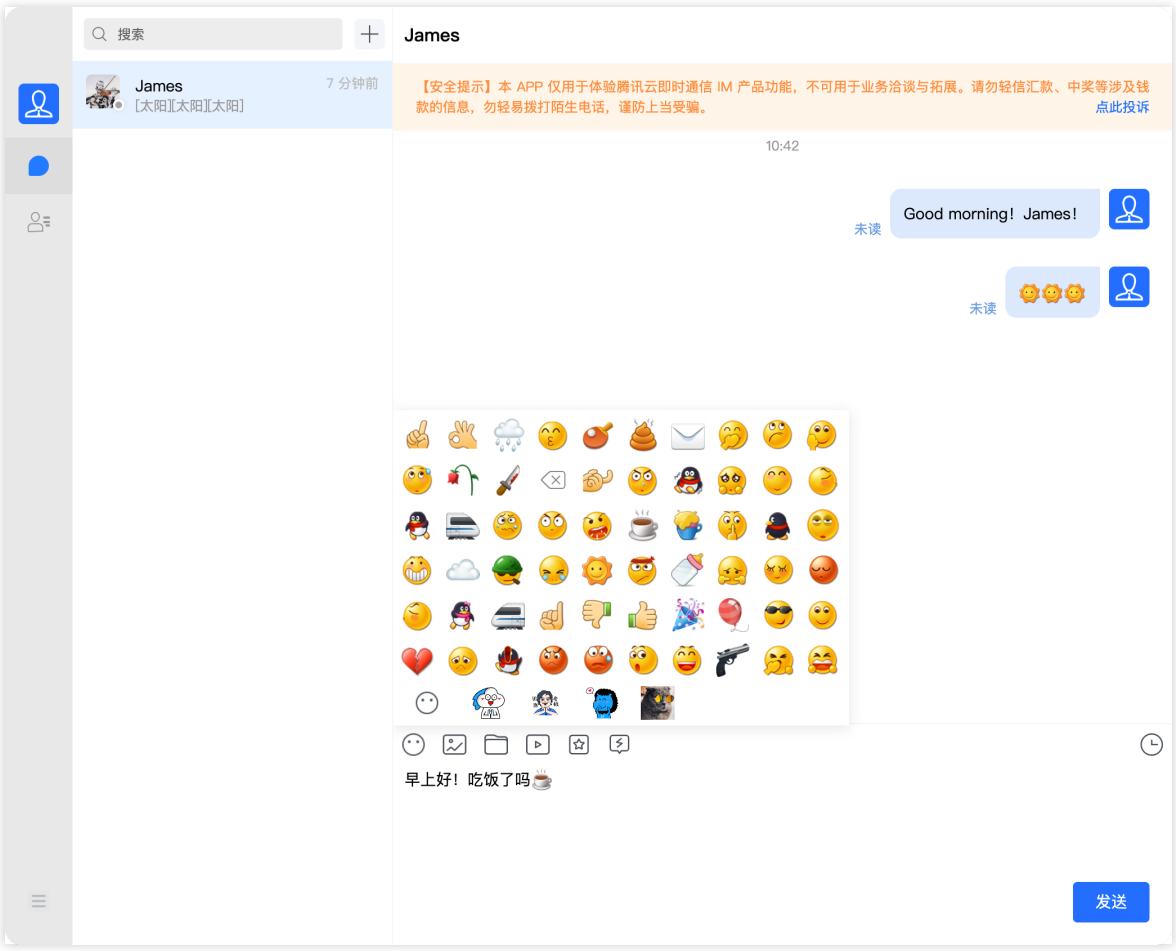
目标文件目录： `TUIKit/components/TUIChat/emoji-config/locales/en.ts`



```
// 重写英文词条 key-value 声明，示例如下
const Emoji = {
  '[@custom_Expect]': '[Expect]',
  '[@custom_Blink]': '[Blink]',
  '[@custom_Guffaw]': '[Guffaw]',
  ...
 }
```

步骤3：渲染自定义小表情

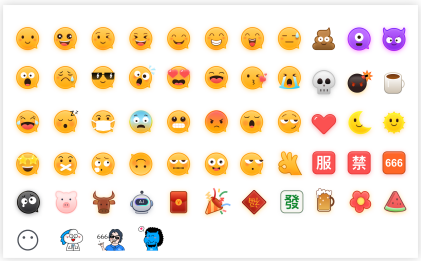
TUIKit 内部内置自定义小表情解析功能，当您完成以上步骤替换小表情后，小表情可以在 TUIChat 内部自动解析上屏，效果如下：

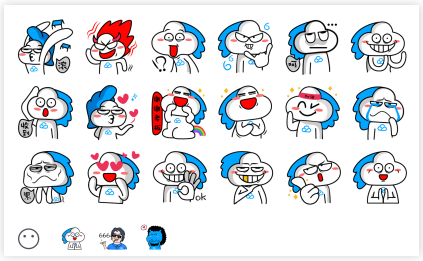


删除系统自带表情

系统自带表情 emojiGroupID 对应表情包如下：

emojiGroupID = 0	emojiGroupID = 1	emojiGroup



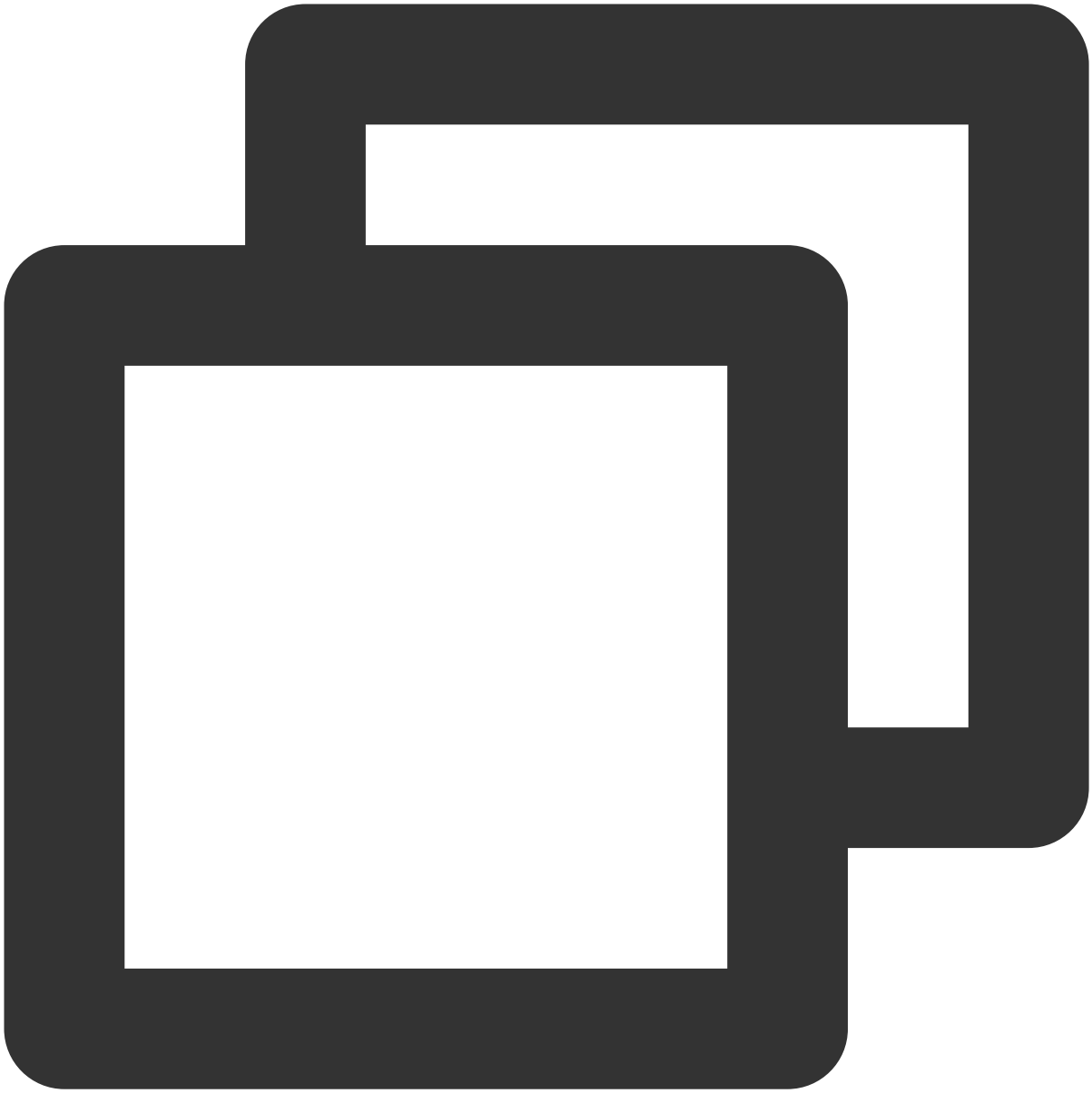


如不需要以上某组表情，可以在 `TUIKit/components/TUIChat/emoji-config/default-emoji.ts` 删除对于该组表情的定义即可：

注意：

如您有多端互通需求，请务必保证在各端删除同样的表情组，并且请保证删除后各端表情组配置相同。

版权所有：腾讯云计算（北京）有限责任公司



```
// 比如，删除 emojiGroupID = 1 的大表情组
export const BIG_EMOJI_GROUP_LIST: IEmojiGroupList = [
  // {
  //   emojiGroupID: 1,
  //   type: EMOJI_TYPE.BIG,
  //   url: DEFAULT_BIG_EMOJI_URL,
  //   list: ['yz00', 'yz01', 'yz02', 'yz03', 'yz04', 'yz05', 'yz06', 'yz07', 'yz08
  //     'yz09', 'yz10', 'yz11', 'yz12', 'yz13', 'yz14', 'yz15', 'yz16', 'yz17'],
  // },
  ...
];
```

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2024-07-08 16:11:21

以下为您介绍，如何为腾讯云 IM Flutter UIKit 引入表情能力。

介绍

我们的 `Message` 组件中，支持发送及接收两种类型的表情：

表情类型	发送形式	是否文字混排	发送内容	解析方式	引入方式	TUIKit 默认自带
小图片表情	文本消息	是	表情图片名称	根据名称，自动匹配本地 Asset 图片资源	默认的可直接用，自定义图片资源预存于 Asset，并定义 `List`	一套默认小表情图库（如下发截图），可直接使用
大图片表情	表情消息	否	`baseURL` 拼接图片文件名，表情图片 Asset 路径	通过路径，解析 Asset 资源	图片资源预存于Asset，并定义 `List`	-

案例名称	小图片表情 （我们自己设计）	腾讯云大图片表情
表情类型	小图片表情	大图片表情
案例说明	**默认自带启用**，位于第一栏	没有默认提供，可以按照文档所述用法，快速传入自定义表情包
截图展示		

图1

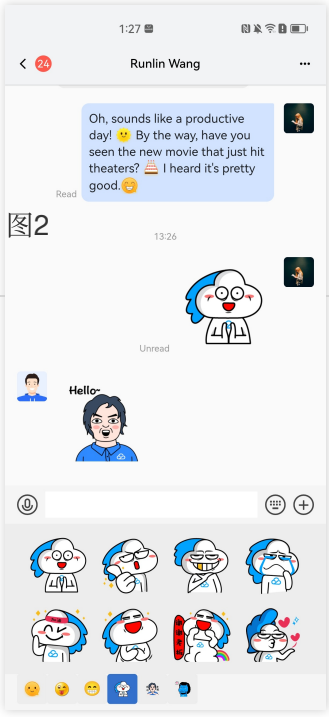
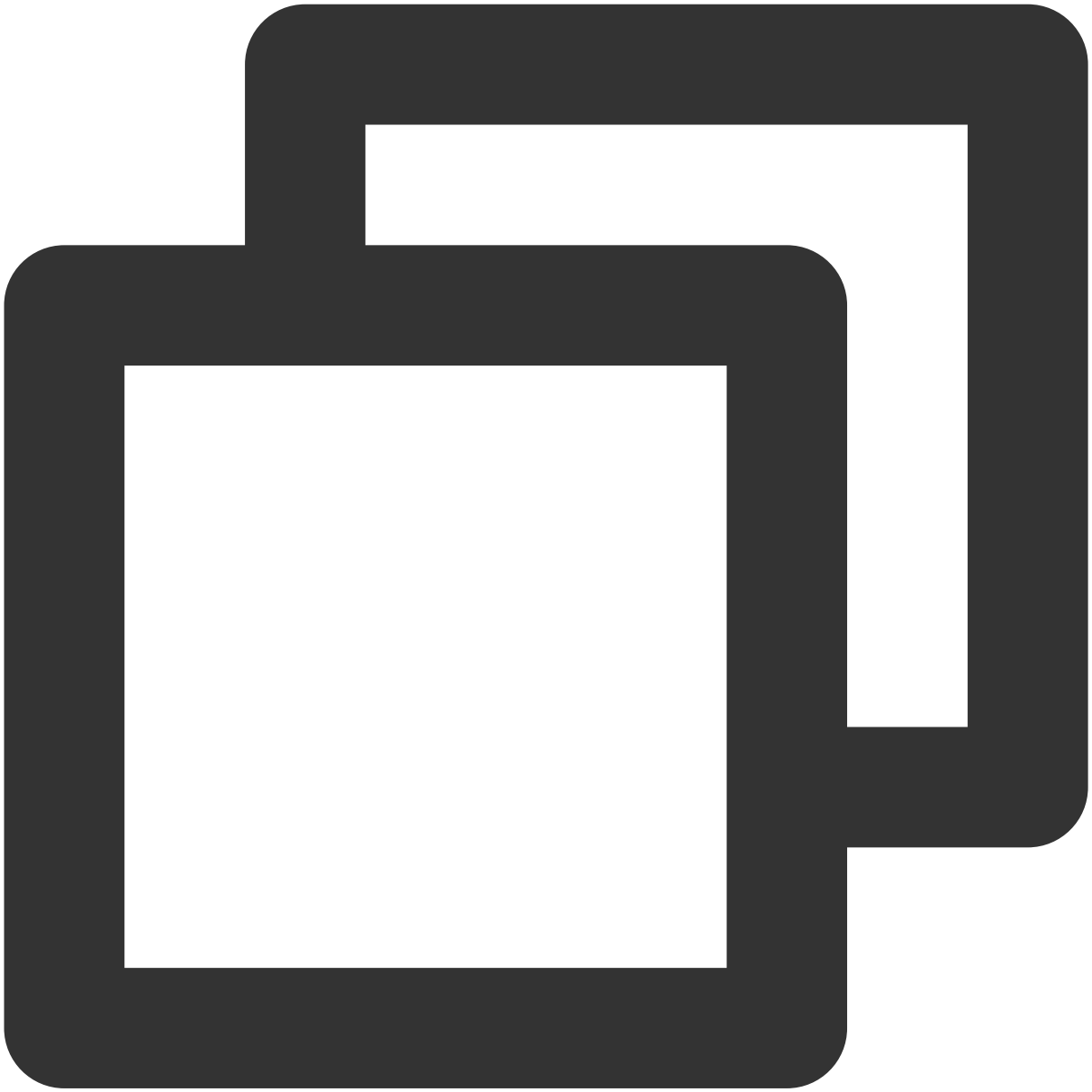


图2

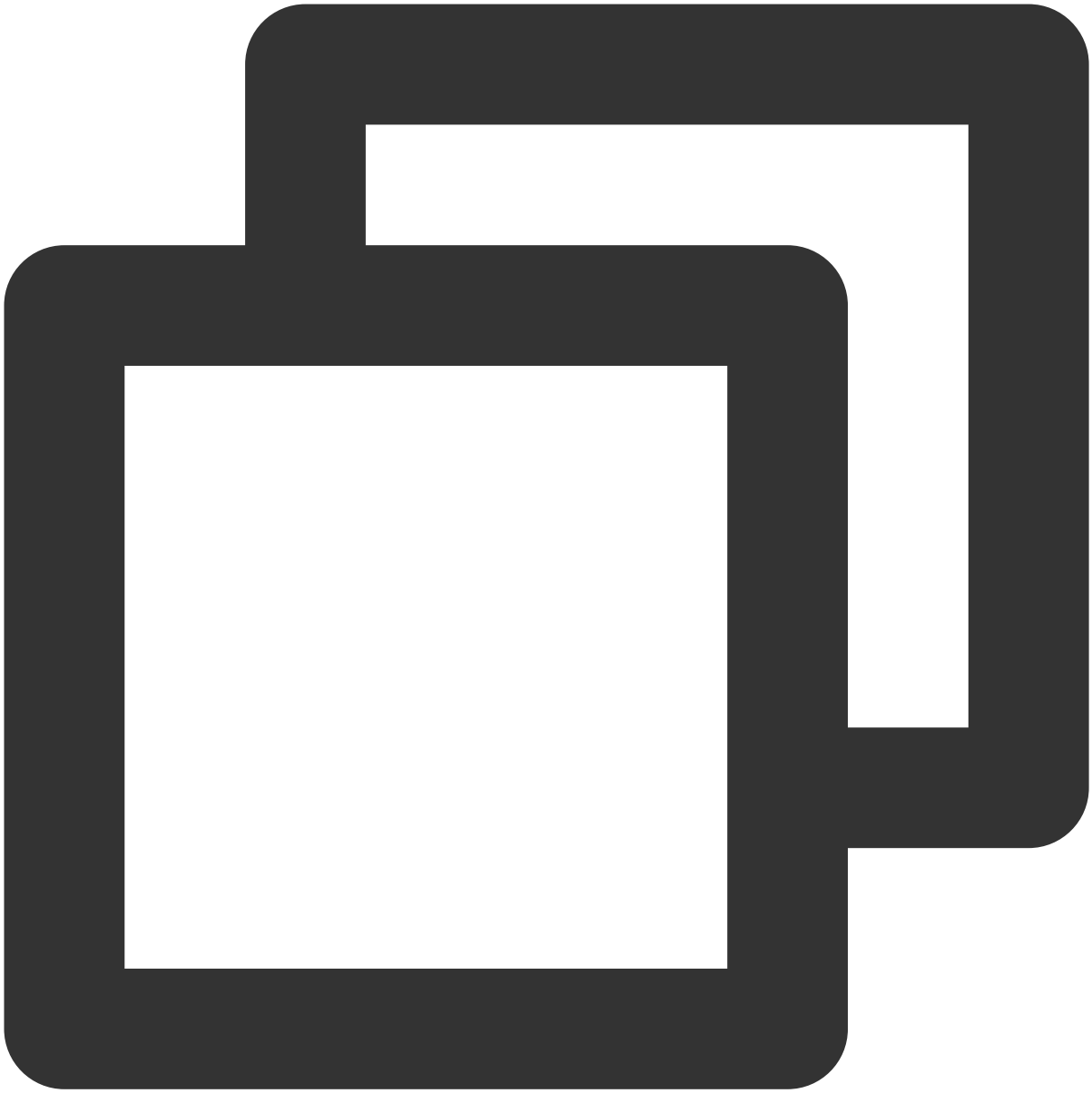
使用方法

首先，安装 tencent_cloud_chat_sticker 插件：



```
flutter pub add tencent_cloud_chat_sticker
```

在 `initUIKit` 中的 `plugins` 列表添加以下代码以启用此插件：



```
TencentCloudChatPluginItem(
  name: "sticker",
  initData: TencentCloudChatStickerInitData(
    userID: TencentCloudChatLoginData.userID,
  ).toJson(),
  pluginInstance: TencentCloudChatStickerPlugin(
    context: context,
  ),
)
```

在 `TencentCloudChatStickerInitData` 对象中，您可以选择启用哪些默认的表情包，并根据需要添加更多的表情包。

版权所有：腾讯云计算（北京）有限责任公司