即时通信 IM

聊天互动（含 UI）

产品文档



腾讯云

©2013-2025 Tencent Cloud International Pte. Ltd.

【版权声明】

©2013-2025 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】



及其他腾讯云服务相关的商标均为腾讯集团下的相关公司主体所有。另外，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
React Native
Flutter
快速开始
Android
iOS
Flutter
集成 TUIKit
Android
iOS
Web & H5（react)
Web & H5 （Vue）
Unity
React Native
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

©2013-2025 Tencent Cloud International Pte. Ltd.

Android
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
全局
聊天界面
会话列表
群组设置
联系人
iOS
全局
聊天界面
会话列表
群组设置
联系人
Web(Vue)
H5(Vue)
React
Flutter
实现本地搜索
Android
iOS
Web & H5 & Uniapp（Vue）
接入离线推送
Android

©2013-2025 Tencent Cloud International Pte. Ltd.

厂商配置
快速接入
客户端 API
iOS
厂商配置
快速接入
客户端 API
uni-app
厂商配置
快速接入
客户端 API
Flutter
厂商配置
快速接入
客户端 API
React Native
厂商配置
快速接入
客户端 API
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)
消息已读回执
Android
iOS
Web & H5 & Uniapp（Vue）
消息表情回应
Android
iOS
Web & H5 & Uniapp（Vue）
Flutter
消息引用

©2013-2025 Tencent Cloud International Pte. Ltd.

Android & iOS
Web & H5 & Uniapp（Vue）
React Native
Flutter
文本消息翻译
Flutter
国际化界面语言
Android
iOS
Web & H5 (Vue)
React
Flutter
React Native
添加自定义消息
Android
iOS
Web & H5 & Uniapp (Vue)
添加自定义表情
Android
iOS
Web & H5 & Uniapp
Flutter
自定义 UI 组件
Conversation List（React）
ConversationList
ConversationListContext
ConversationPreview
ConversationSearch
ConversationActions

©2013-2025 Tencent Cloud International Pte. Ltd.

聊天互动（含 UI）
TUIKit 界面库
Android & iOS

最近更新时间：2025-06-23 16:14:02

TUIKit 介绍

TUIKit 是基于腾讯云 Chat 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 Chat SDK 相应的接口实现 Chat 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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

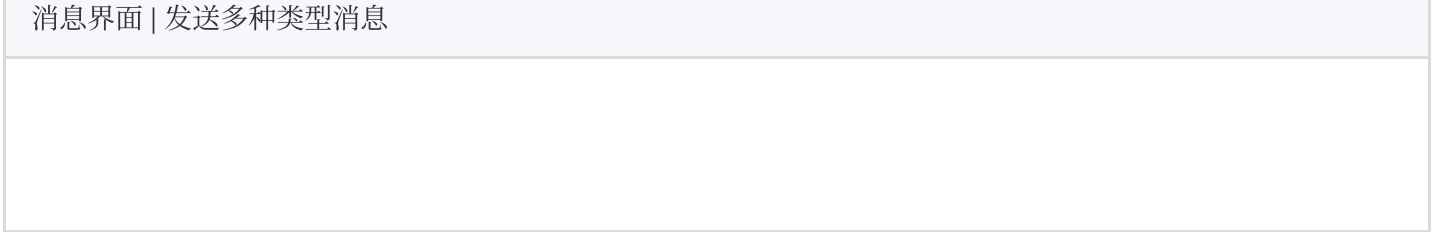
RTL语言

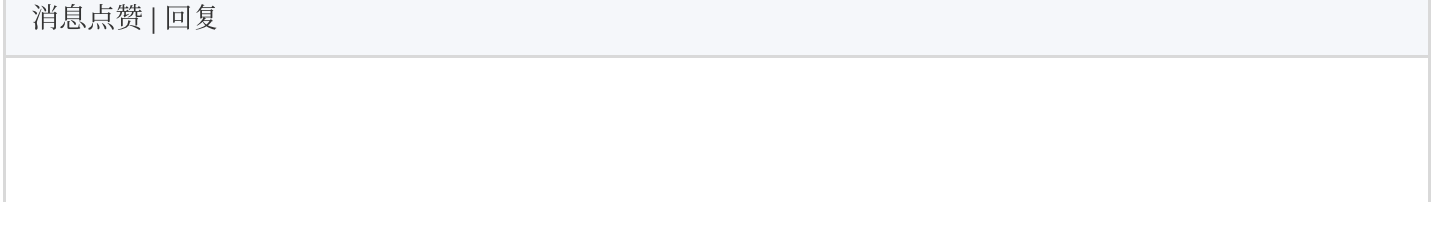
经典版

消息界面 | 发送多种类型消息

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


©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

说明：

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工

©2013-2025 Tencent Cloud International Pte. Ltd.

单 联系我们。

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

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

加群方式管理 | 权限管理

©2013-2025 Tencent Cloud International Pte. Ltd.

群资料及管理 | 群成员管理

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

Web

最近更新时间：2025-01-13 12:01:25

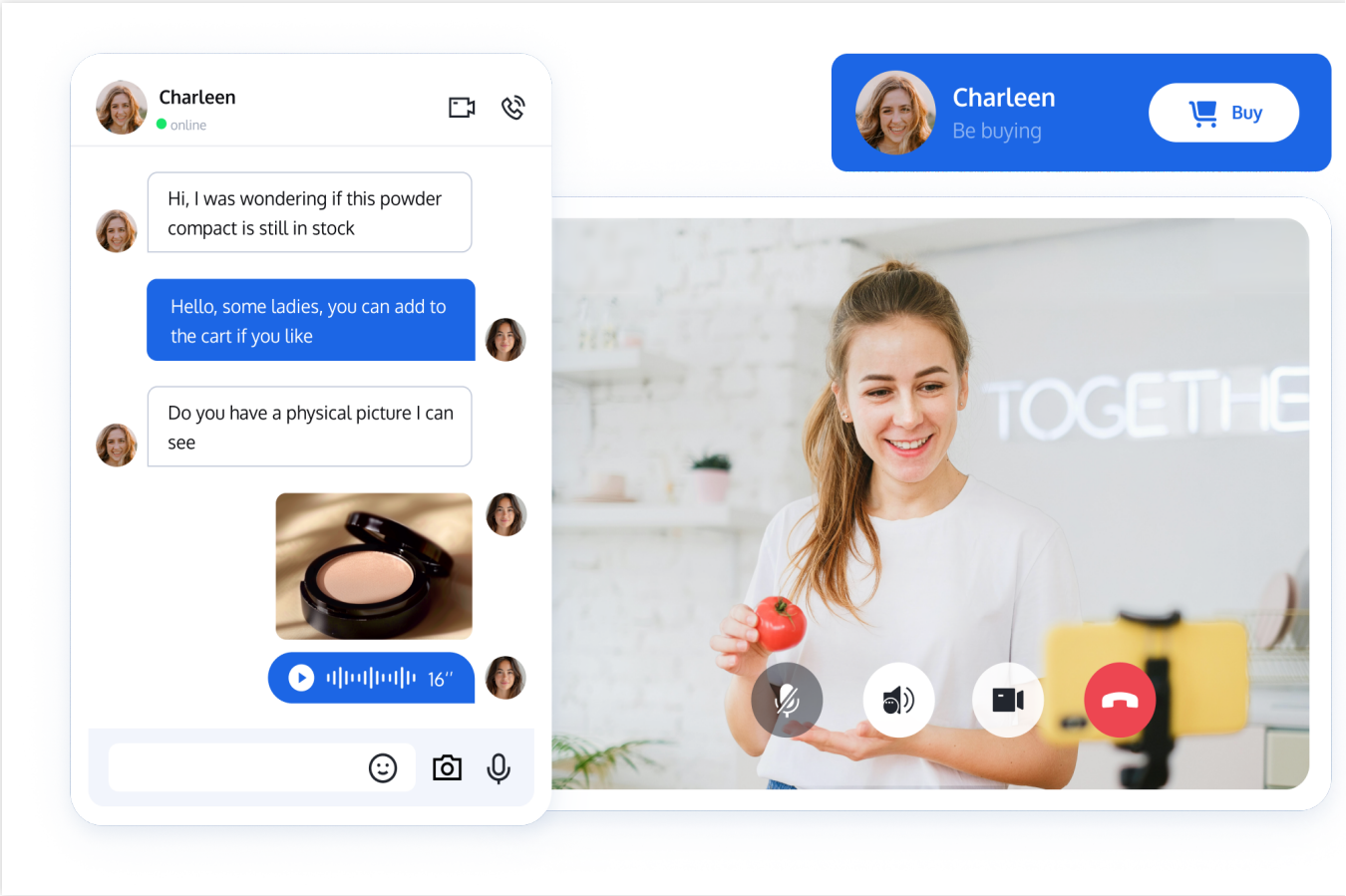
Chat 提供多平台聊天 API、 UI 组件、服务端 API 和 webhook，使您能够在十分钟内构建一个功能齐全的聊天应用程序。
您可以直接体验下面的 Chat UIKit Demo。同时，您可以通过 **体验沙箱** 快速体验在线代码实现。

应用场景

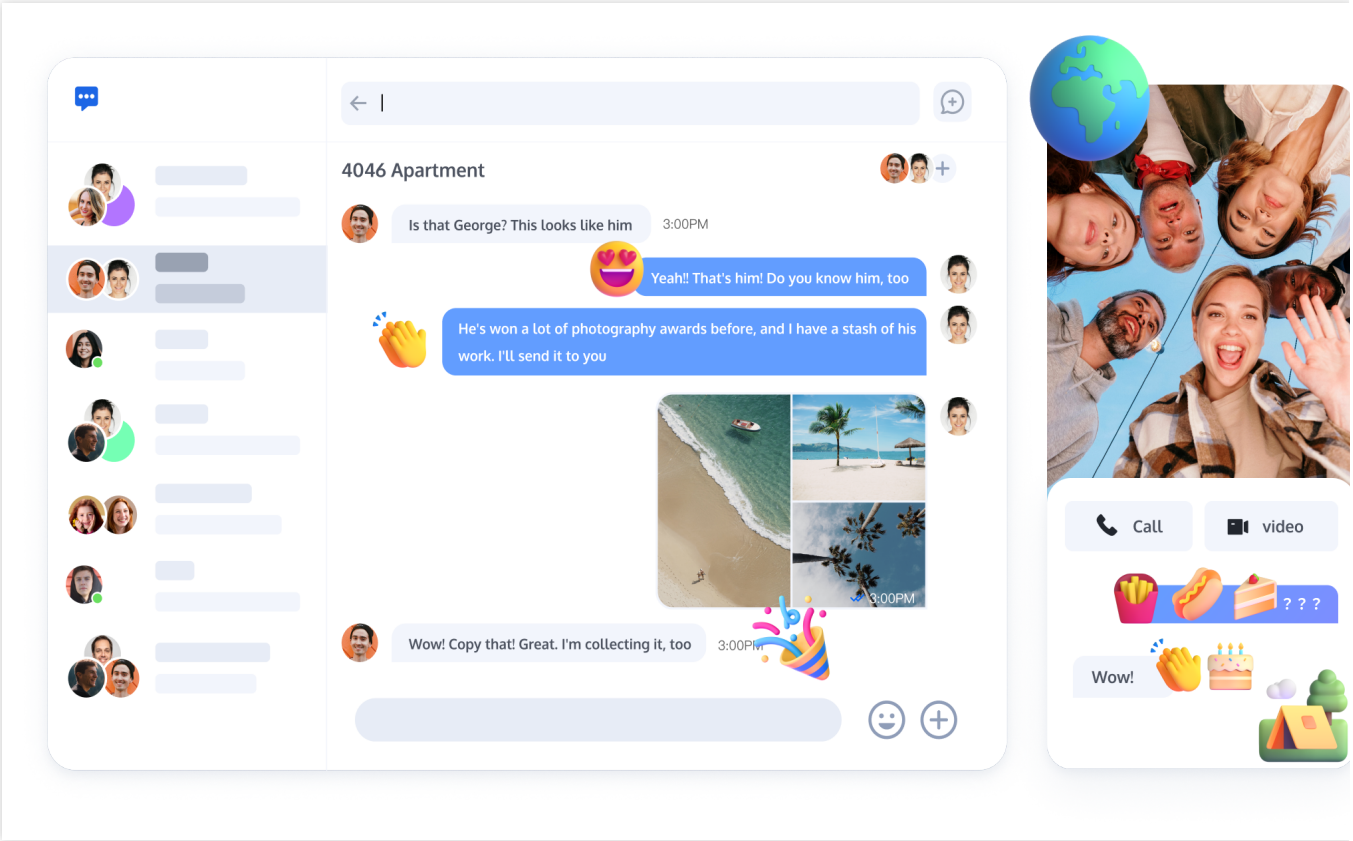
直播互动
零售电商
社交娱乐
对话式 AI
点播服务
游戏开黑
在线教育
医疗服务

©2013-2025 Tencent Cloud International Pte. Ltd.

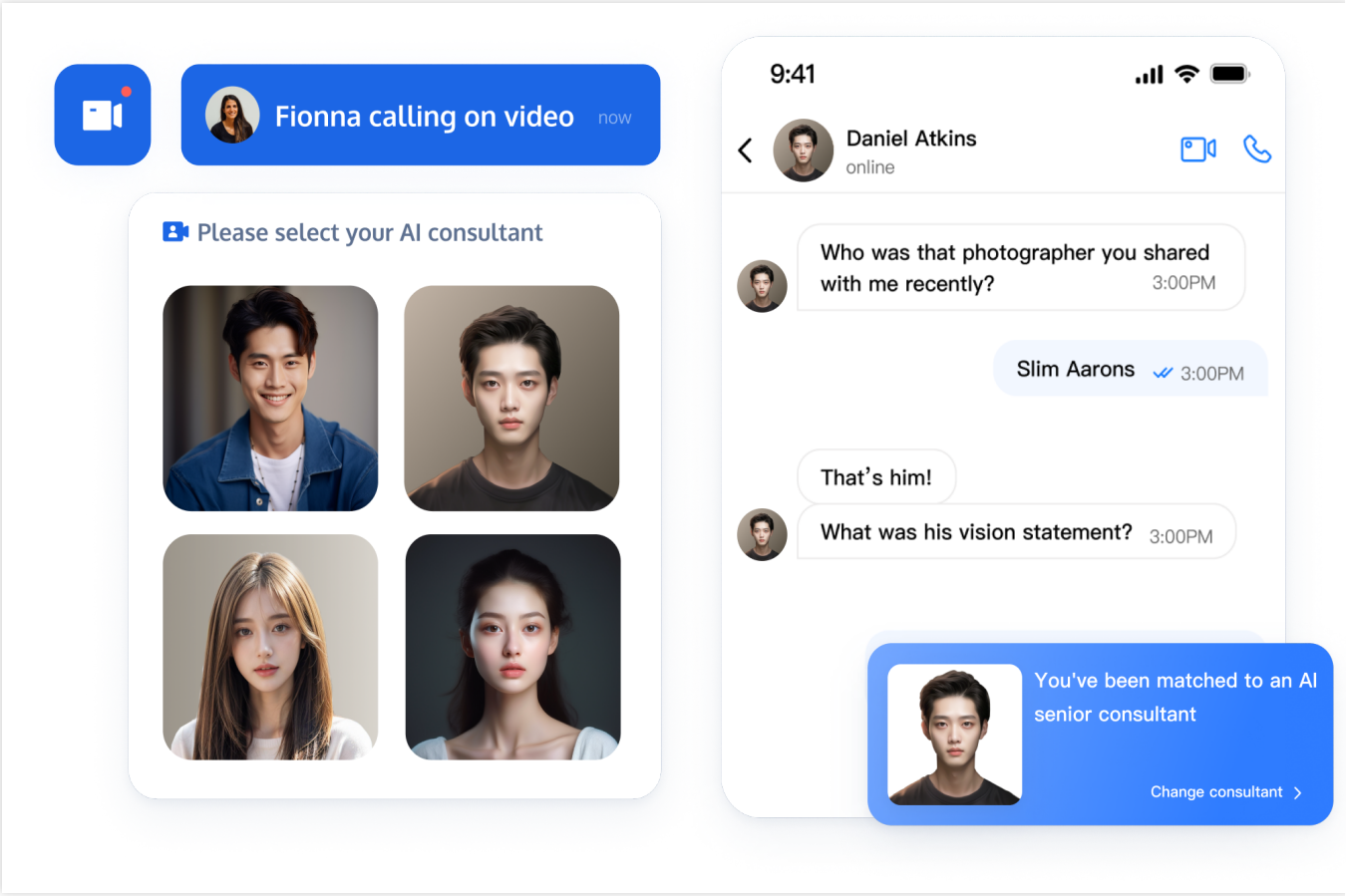

320
Emma
Sophia
Charlotte
Ava
Ryan
Spektral0...
Sent a Sports car
Nightbot : EbilDoggies cast their line into the water and caught a cod !
Ghost_Beretta Enter the room
Spektral0_0 : modCheck
Let's talk
Stream with t...
Follow
abey



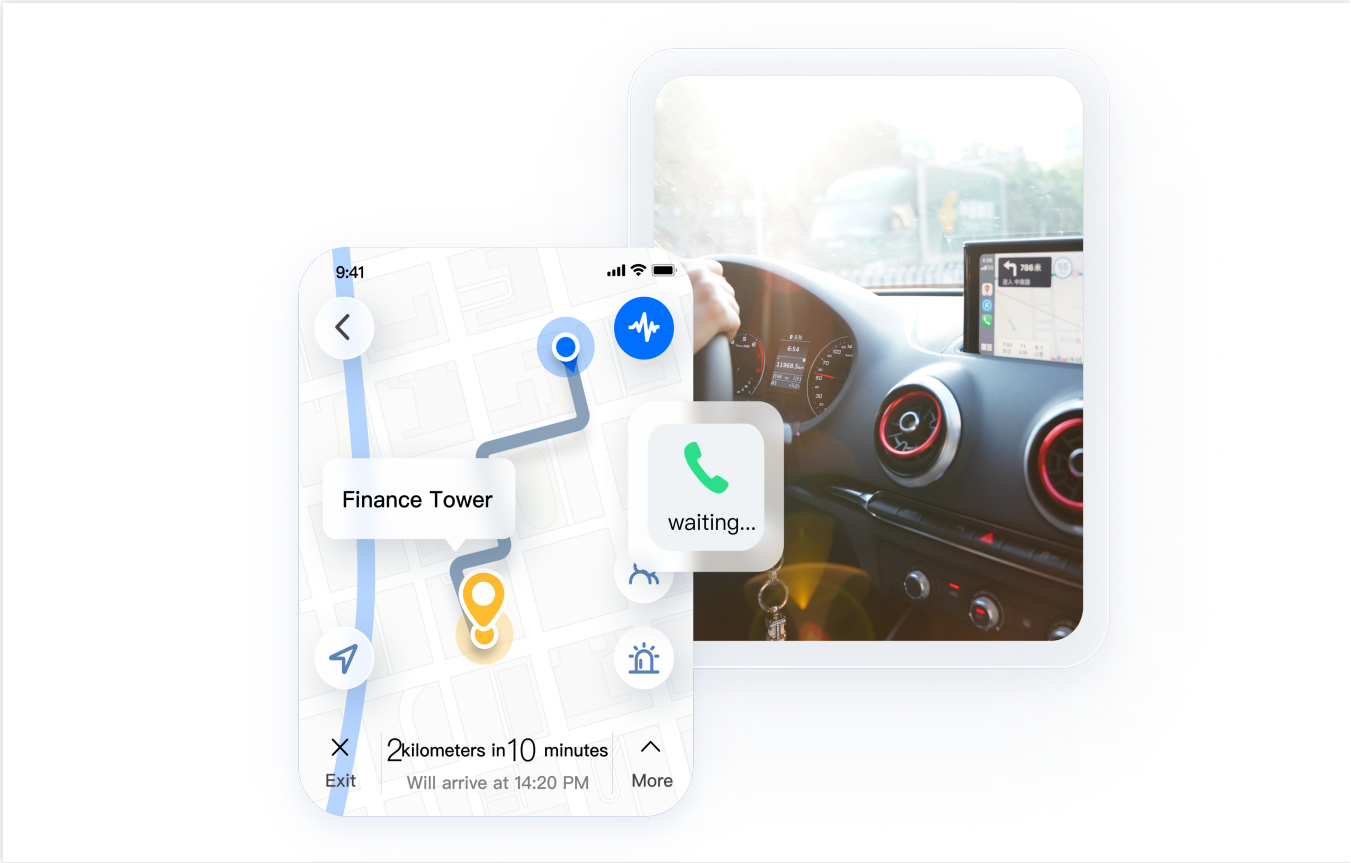
©2013-2025 Tencent Cloud International Pte. Ltd.



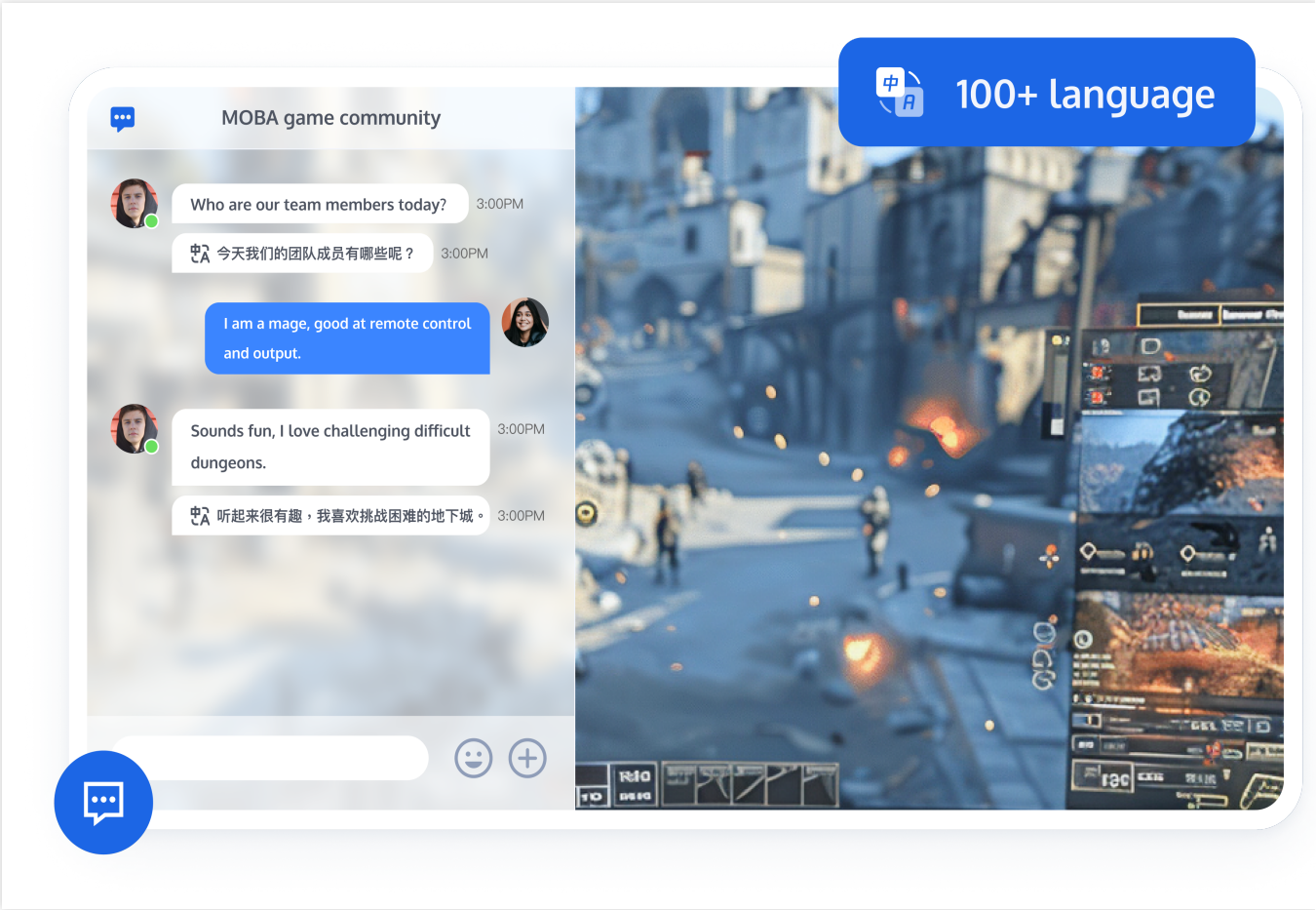
©2013-2025 Tencent Cloud International Pte. Ltd.



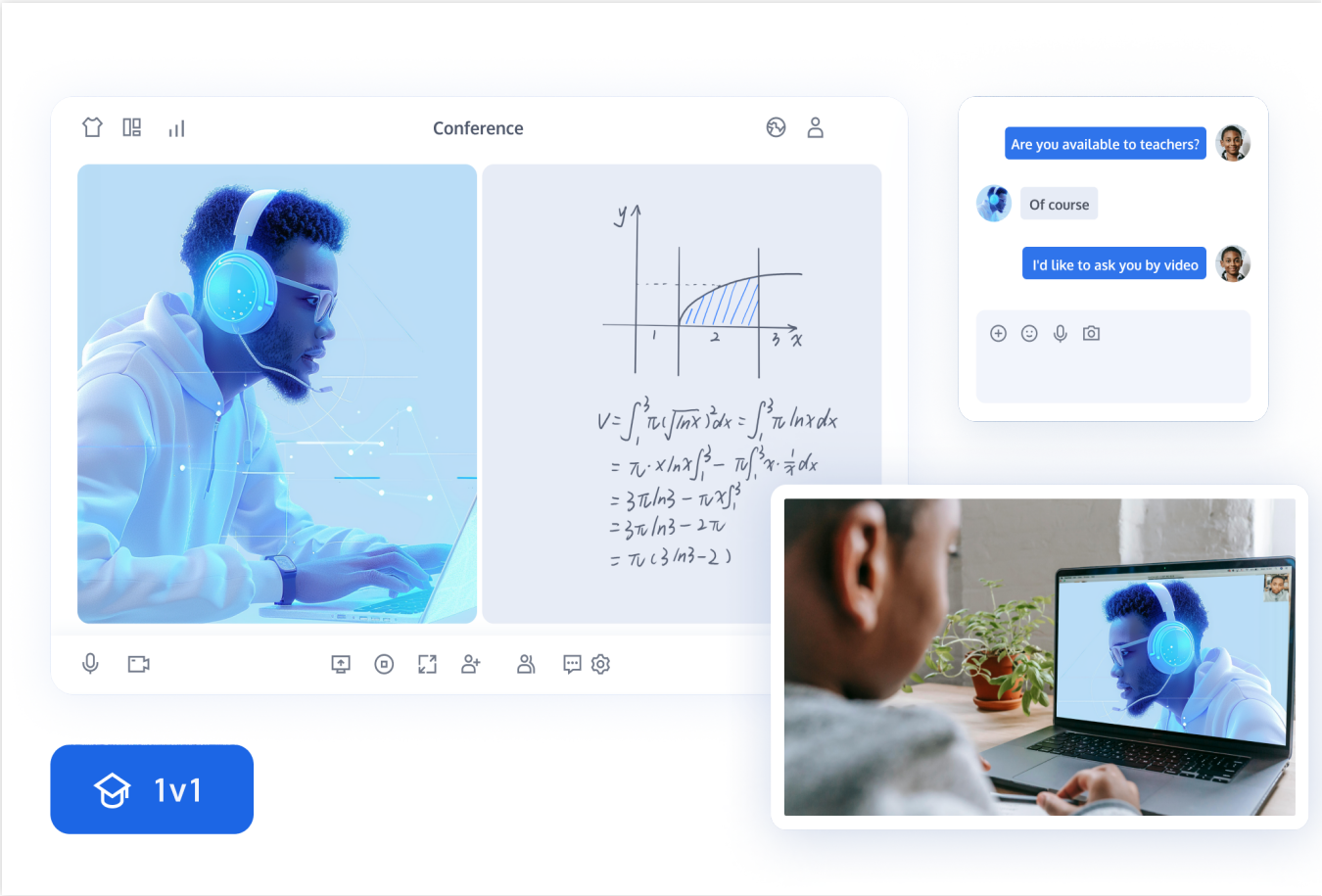
©2013-2025 Tencent Cloud International Pte. Ltd.



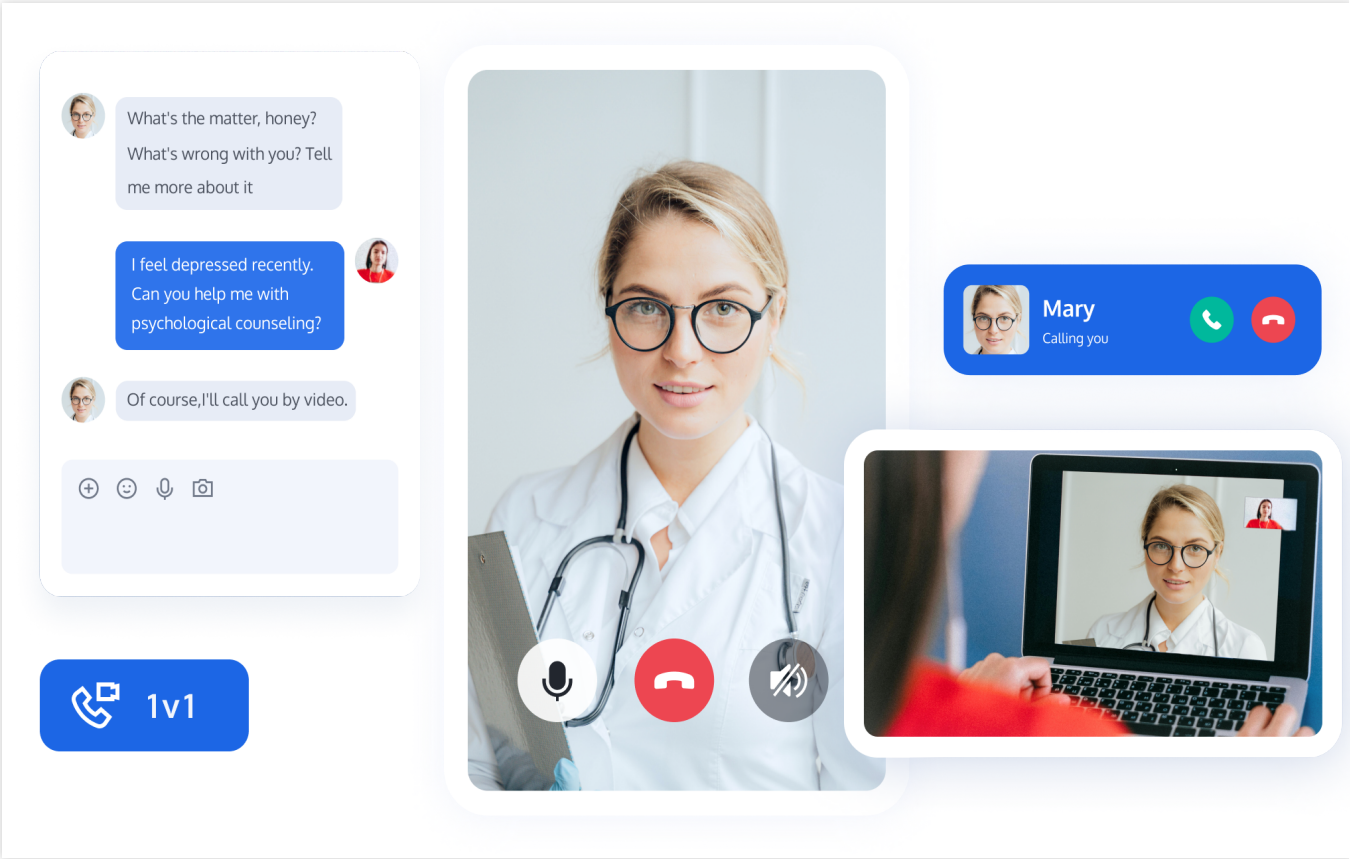
©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.



UIKit

Chat

ConversationList

ChatSetting

Contact

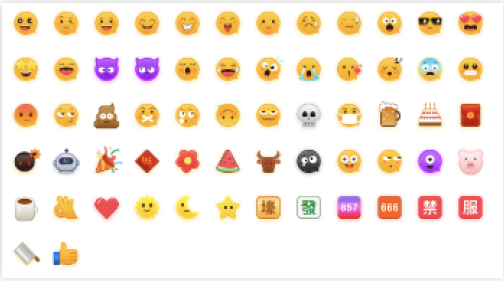
Profile

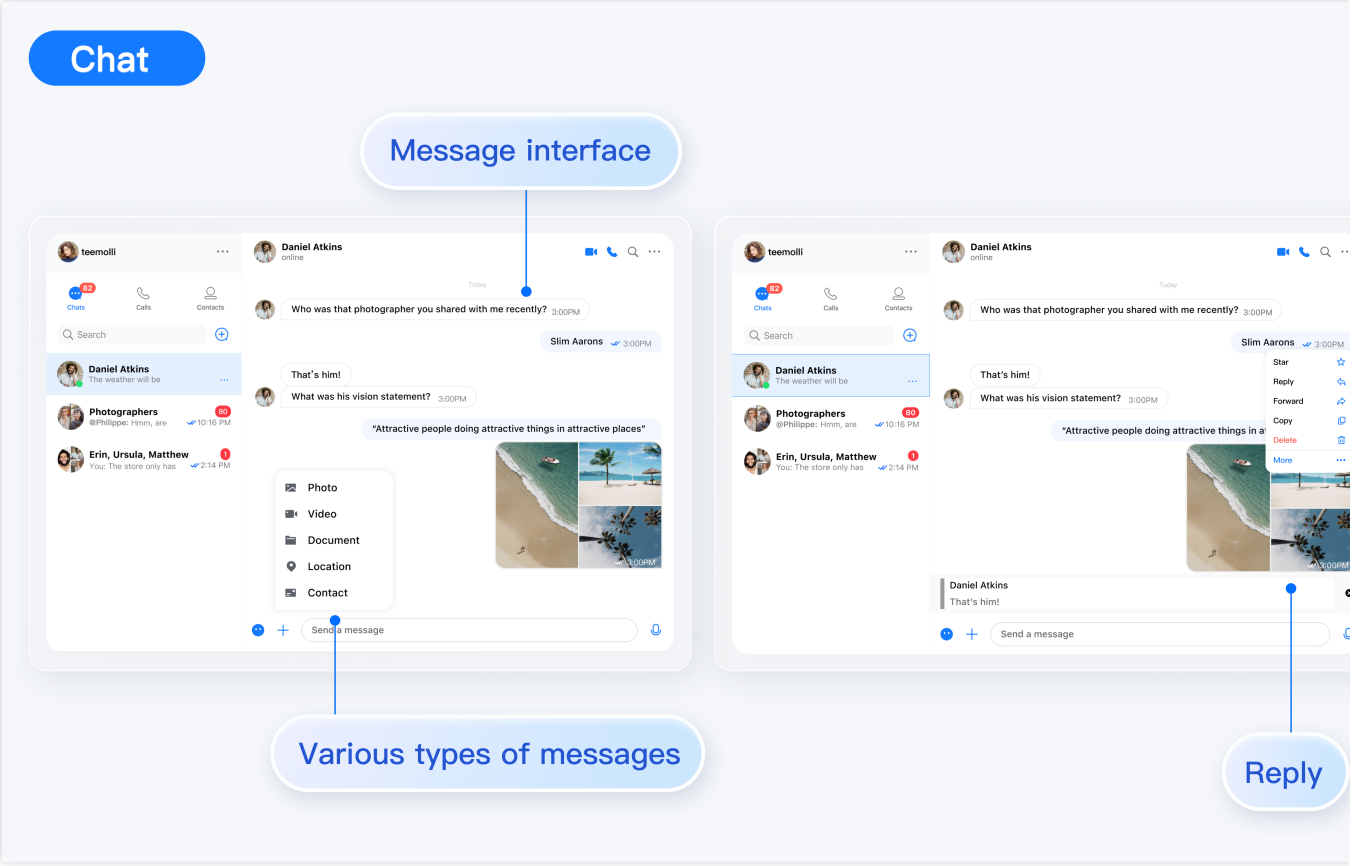
Chat 组件主要负责消息界面的展示。您可以利用它直接发送不同类型的消息，支持文字/表情/图片/视频/文件/自定义等消息类型，同时支持消息转发/撤回/引用、查询、已读回执等功能。

说明：

为尊重表情设计版权，IM Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

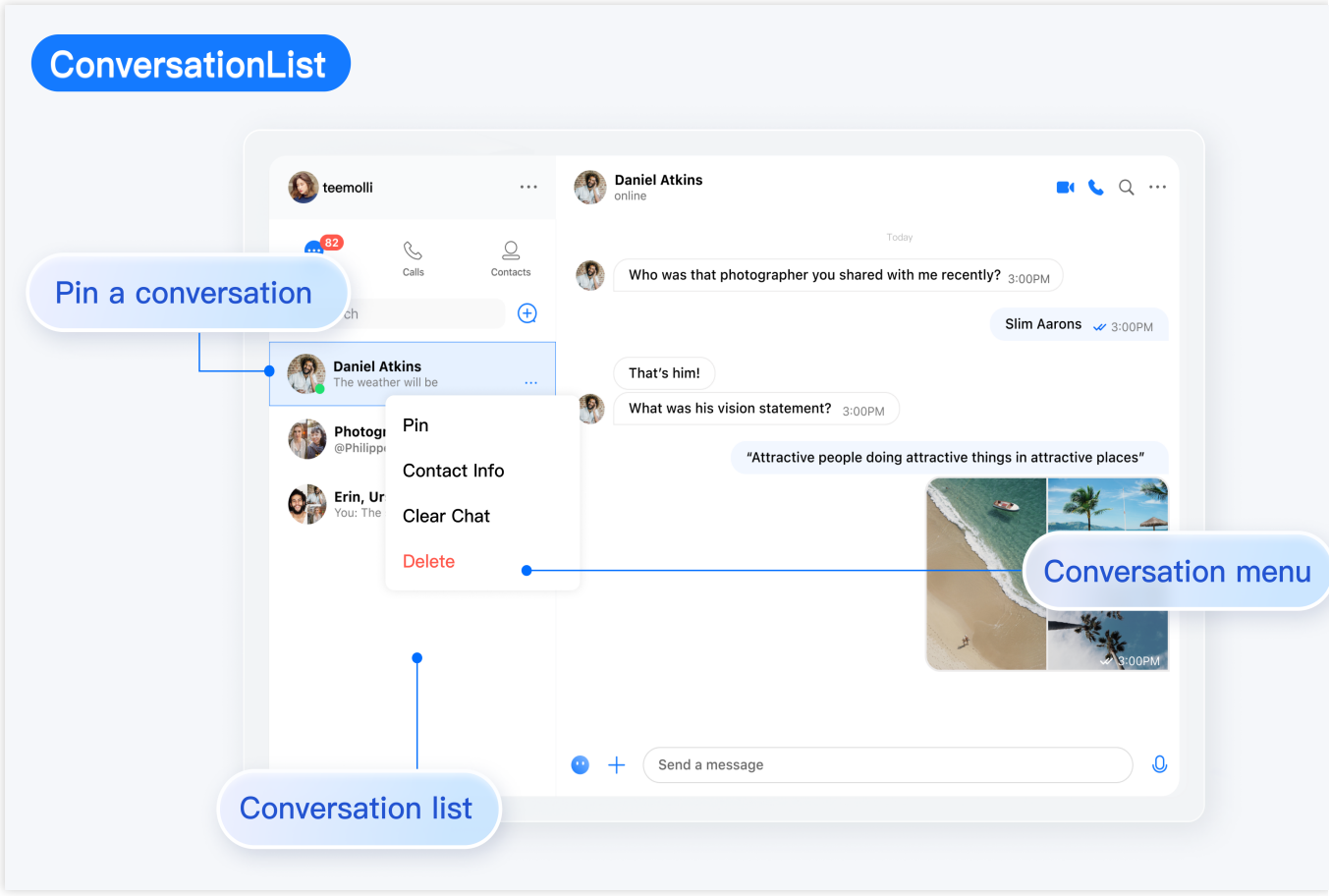
©2013-2025 Tencent Cloud International Pte. Ltd.





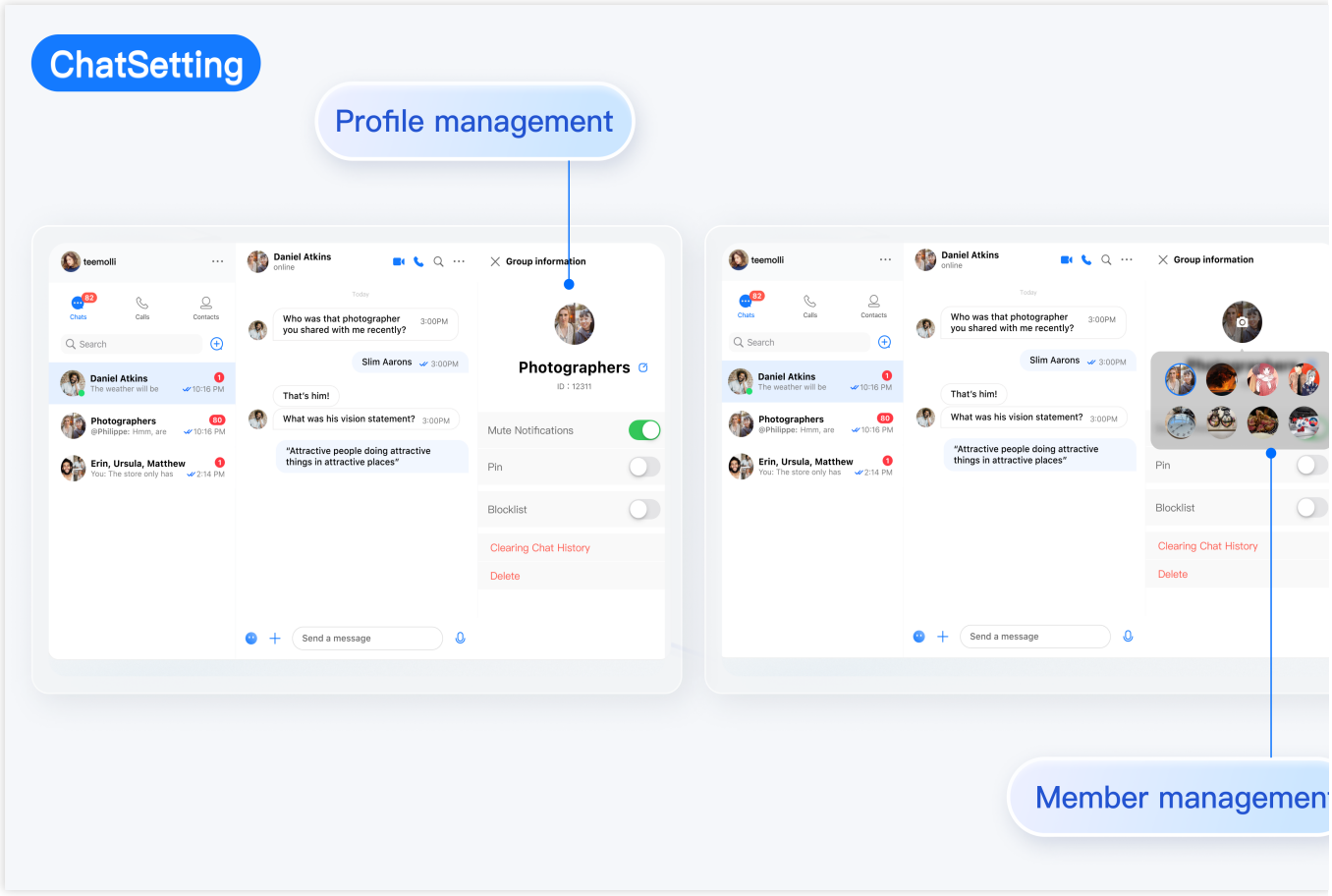
ConversationList 组件主要负责会话列表的展示和编辑，包含会话置顶、会话删除等功能。

©2013-2025 Tencent Cloud International Pte. Ltd.



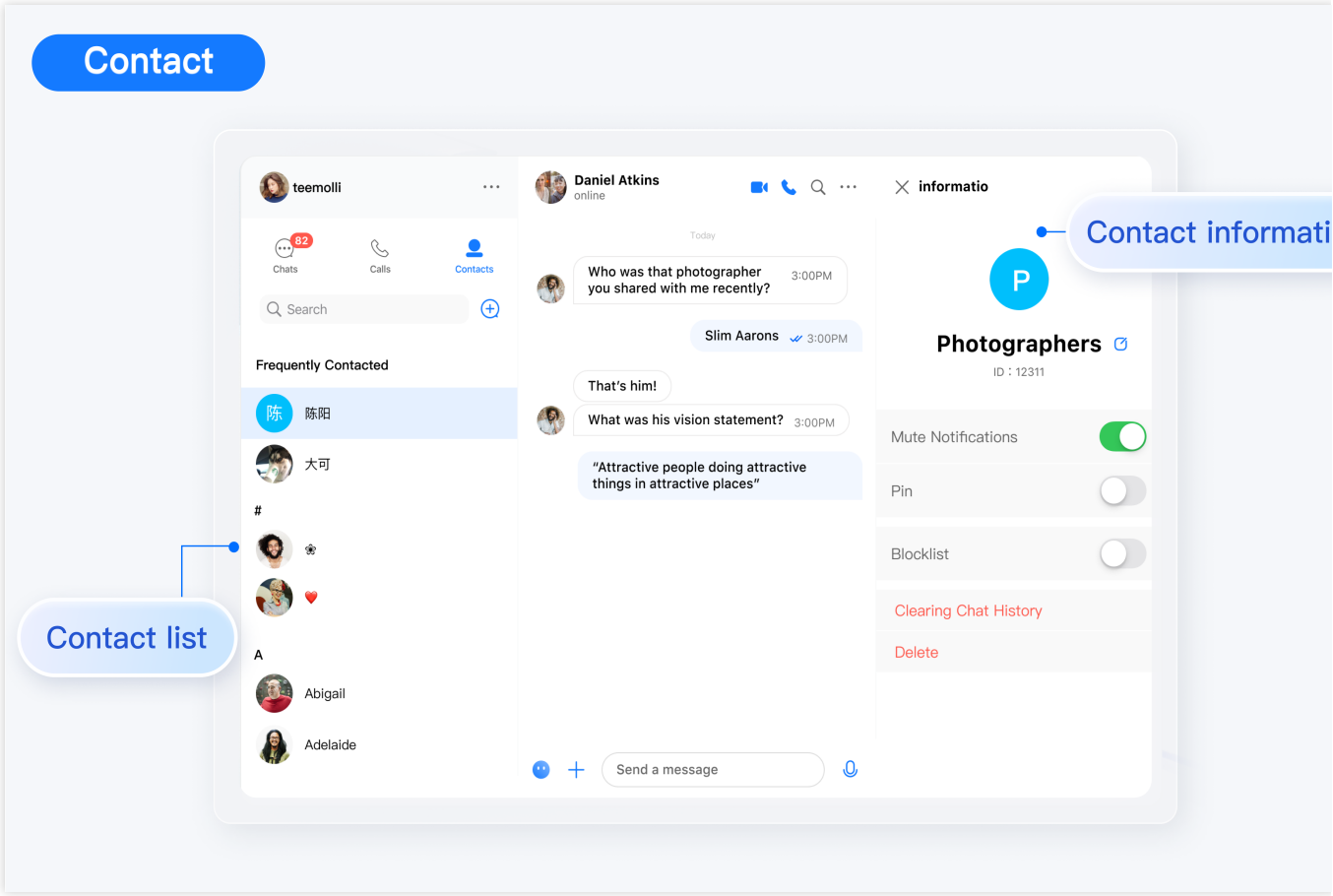
ChatSetting 组件主要负责 Chat 相关设置。

©2013-2025 Tencent Cloud International Pte. Ltd.



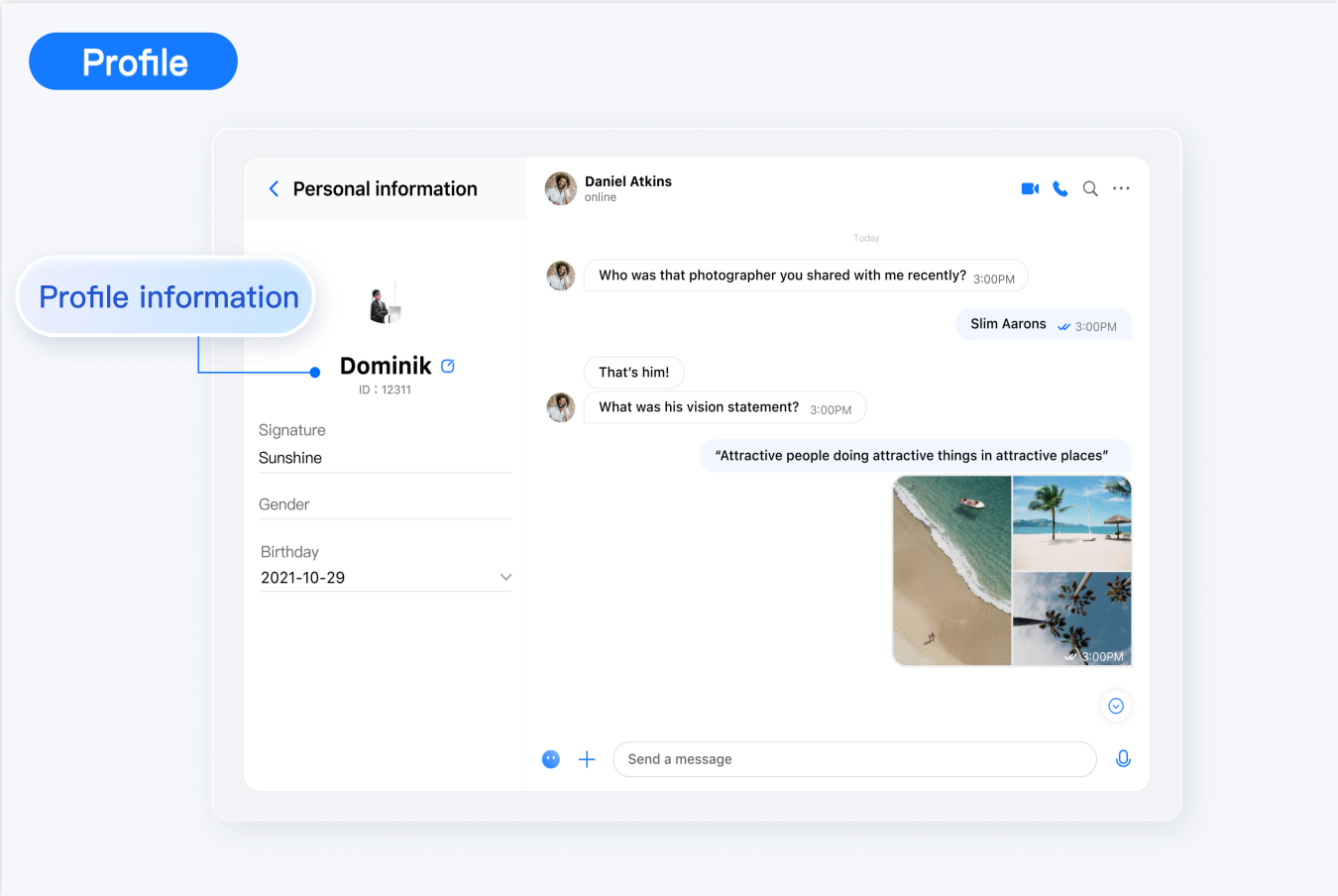
Contact 组件主要负责关系链展示以及创建会话等操作。

©2013-2025 Tencent Cloud International Pte. Ltd.



Profile 组件主要负责用户资料的管理。

©2013-2025 Tencent Cloud International Pte. Ltd.



平台	React	React Native	Vue	Android	iOS and macO
是否支持					
体验 Demo	点击体验		点击体验		
跑通 Github Demo	React	React Native	Vue	Android	iOS and macO
快速集成 UI	React	React Native	Vue	Android	iOS and macO

©2013-2025 Tencent Cloud International Pte. Ltd.

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 未读计数 免打扰 消息翻译 对方正在输入	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

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

H5

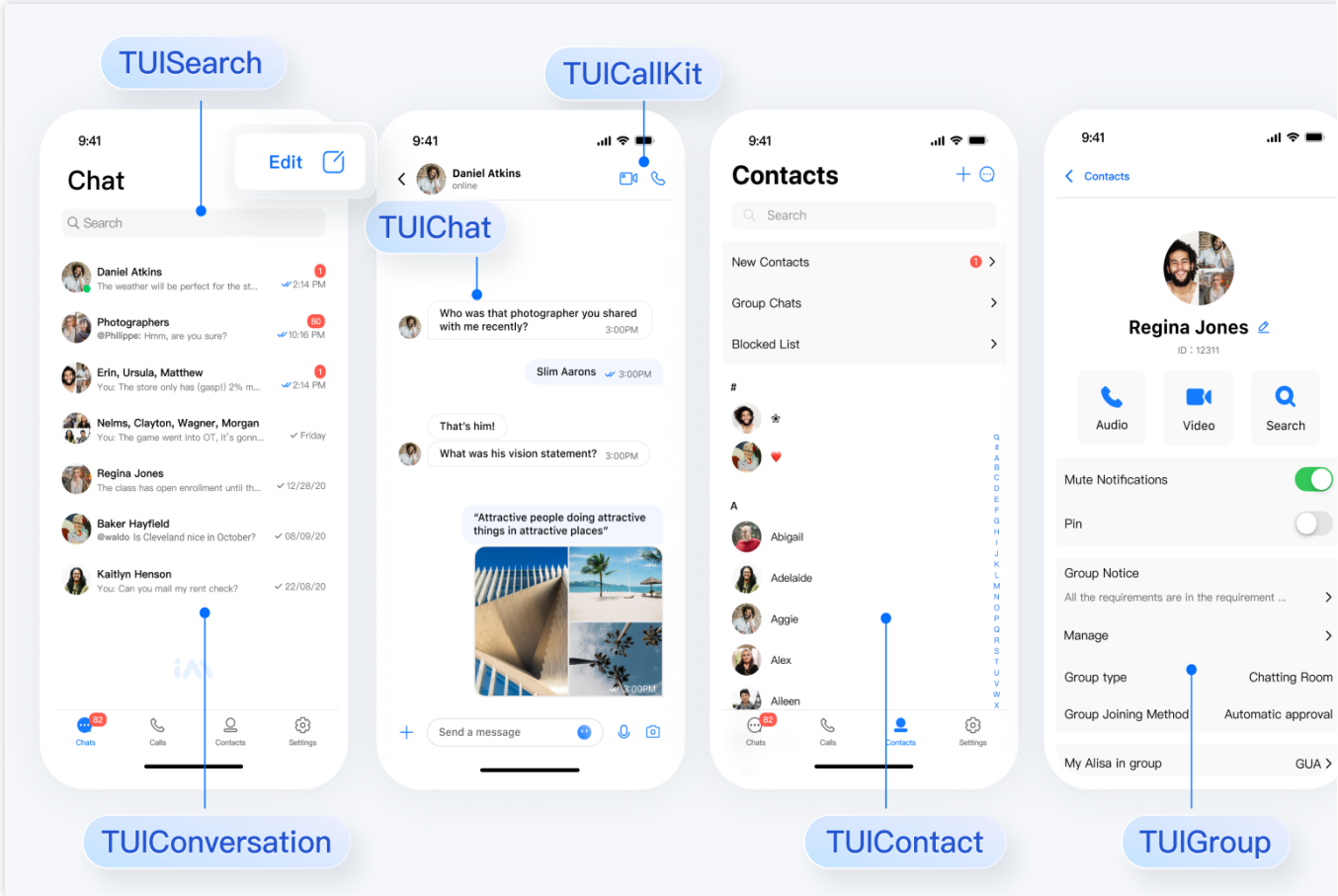
最近更新时间：2025-03-03 11:25:44

TUIKit 介绍

TUIKit 是基于腾讯云 Chat 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 Chat SDK 相应的接口实现 Chat 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

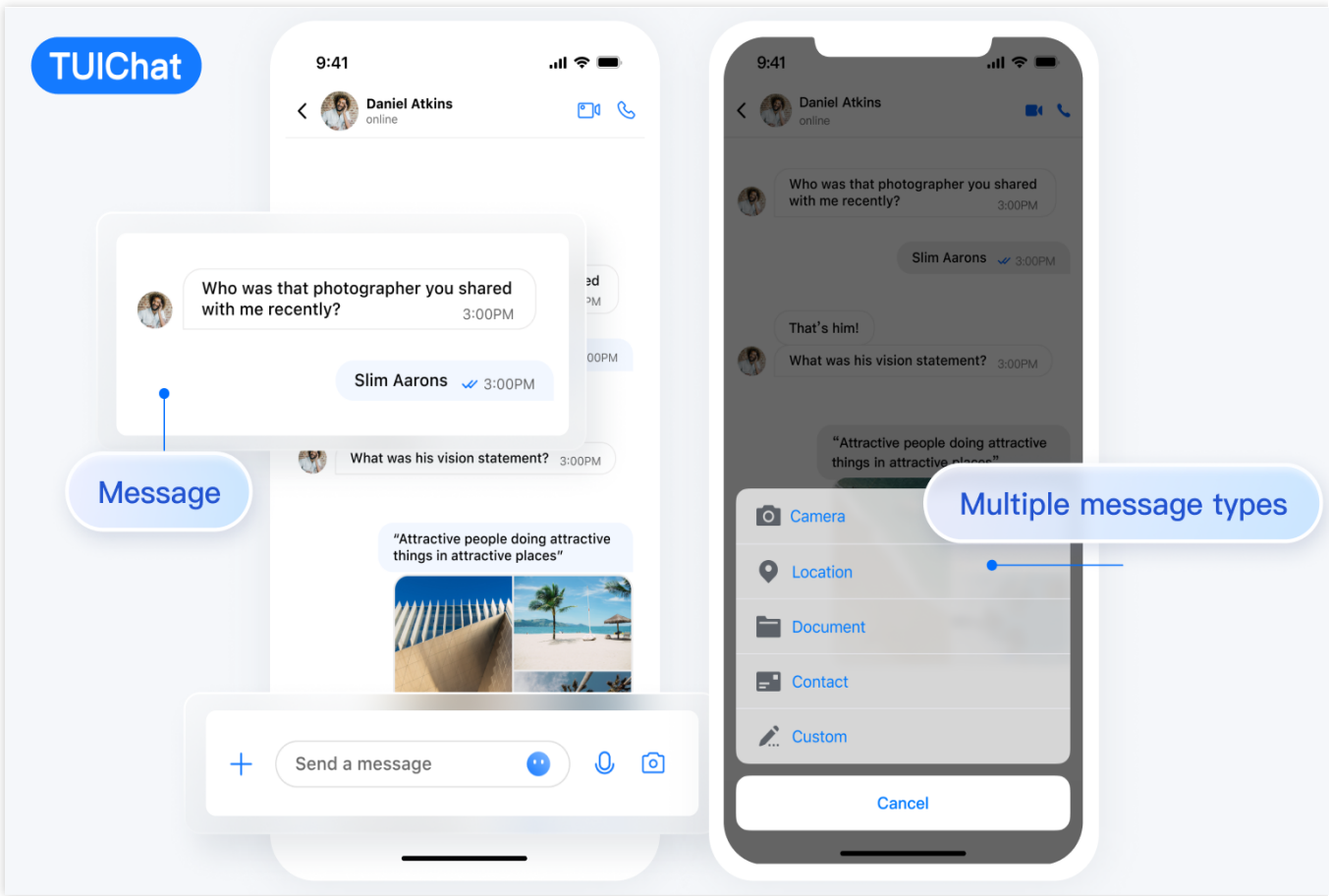
TUIKit 主要分为 TUIChat、TUIConversation、TUIGroup、TUIContact、TUIProfile几个 UI 子组件，每个 UI 组件负责展示不同的内容。



©2013-2025 Tencent Cloud International Pte. Ltd.

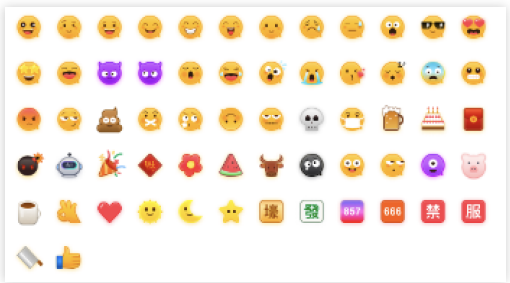
TUIChat 重点功能介绍

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。



说明：

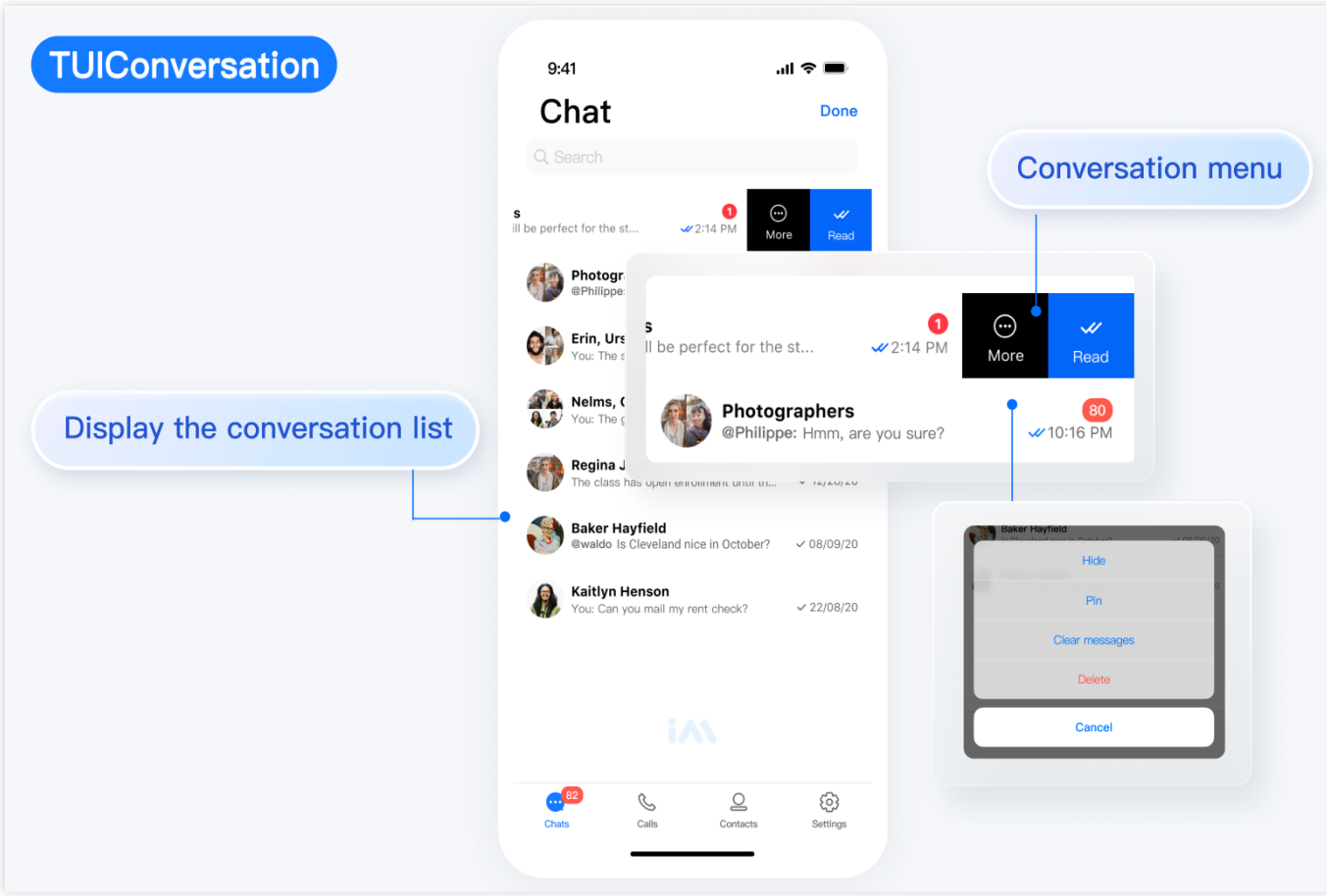
为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。



©2013-2025 Tencent Cloud International Pte. Ltd.

TUIConversation 重点功能介绍

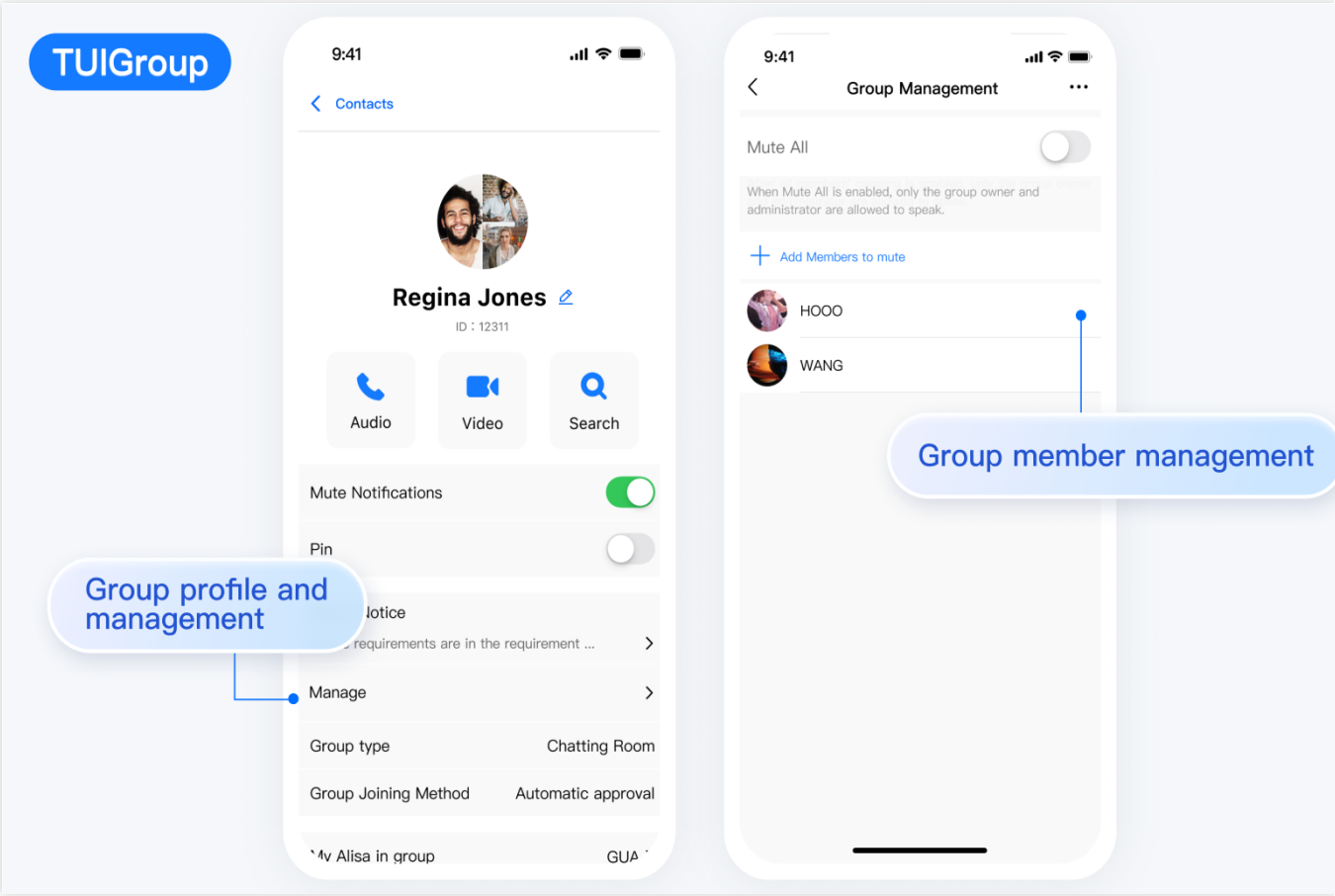
TUIConversation 主要负责会话列表的展示和编辑，包括会话置顶、会话消息免打扰、会话删除等功能。



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。

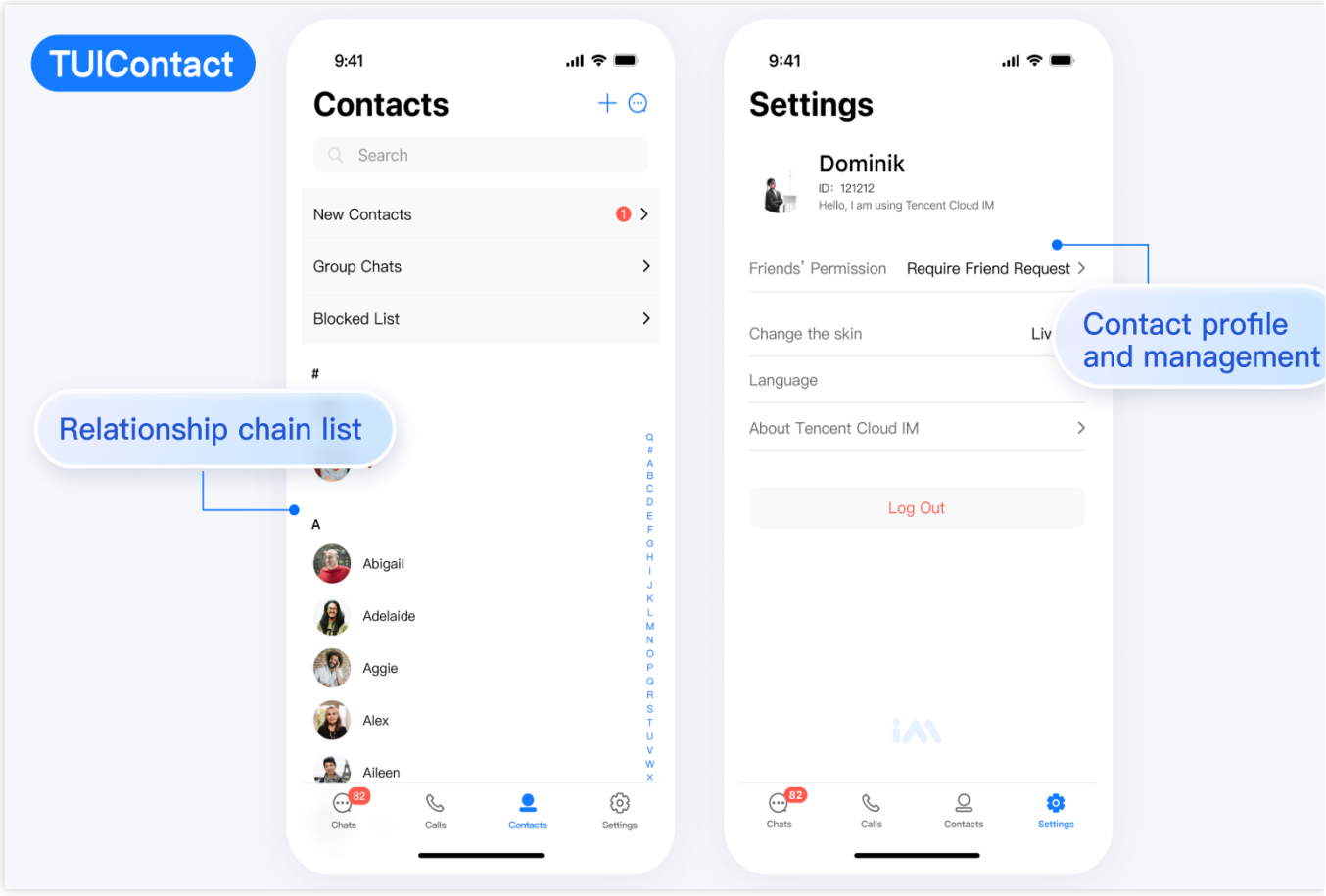
©2013-2025 Tencent Cloud International Pte. Ltd.



TUIContact 重点功能介绍

TUIContact 主要负责联系人的展示、权限设置等。

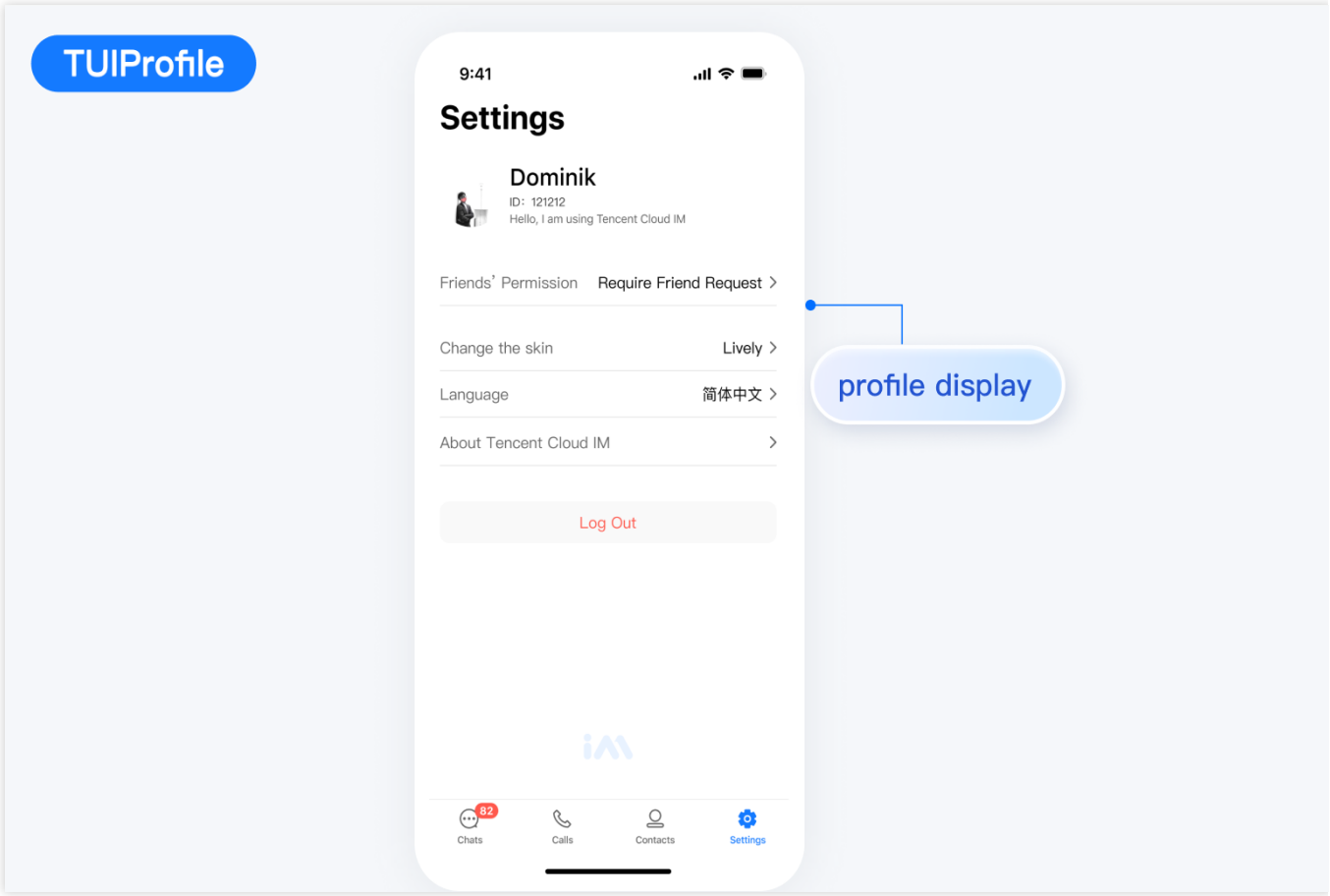
©2013-2025 Tencent Cloud International Pte. Ltd.



TUIProfile 重点功能介绍

TUIProfile 主要负责用户资料的管理。

©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

最近更新时间：2025-03-03 11:24:23

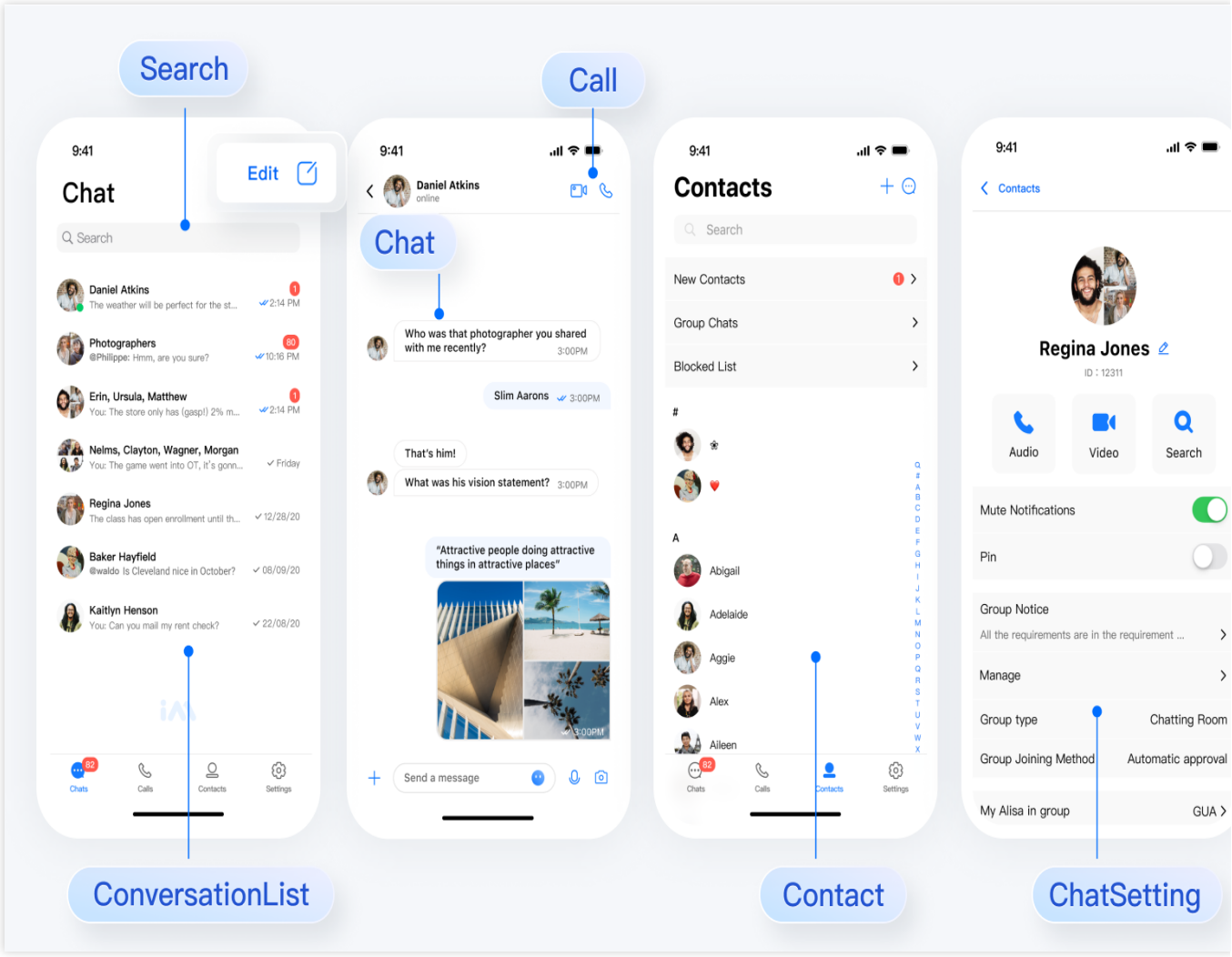
TUIKit 介绍

TUIKit 是一款基于腾讯云 Chat SDK 的 UI 组件库，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。基于 React Native 开发的 UIKit 界面风格更契合海外客户的使用习惯，而且支持国际化，欢迎接入。

TUIKit 主要功能介绍

TUIKit 主要分为 ConversationList、Chat、ChatSetting、Contact 几个 UI 子组件，每个 UI 组件负责展示不同的内容。

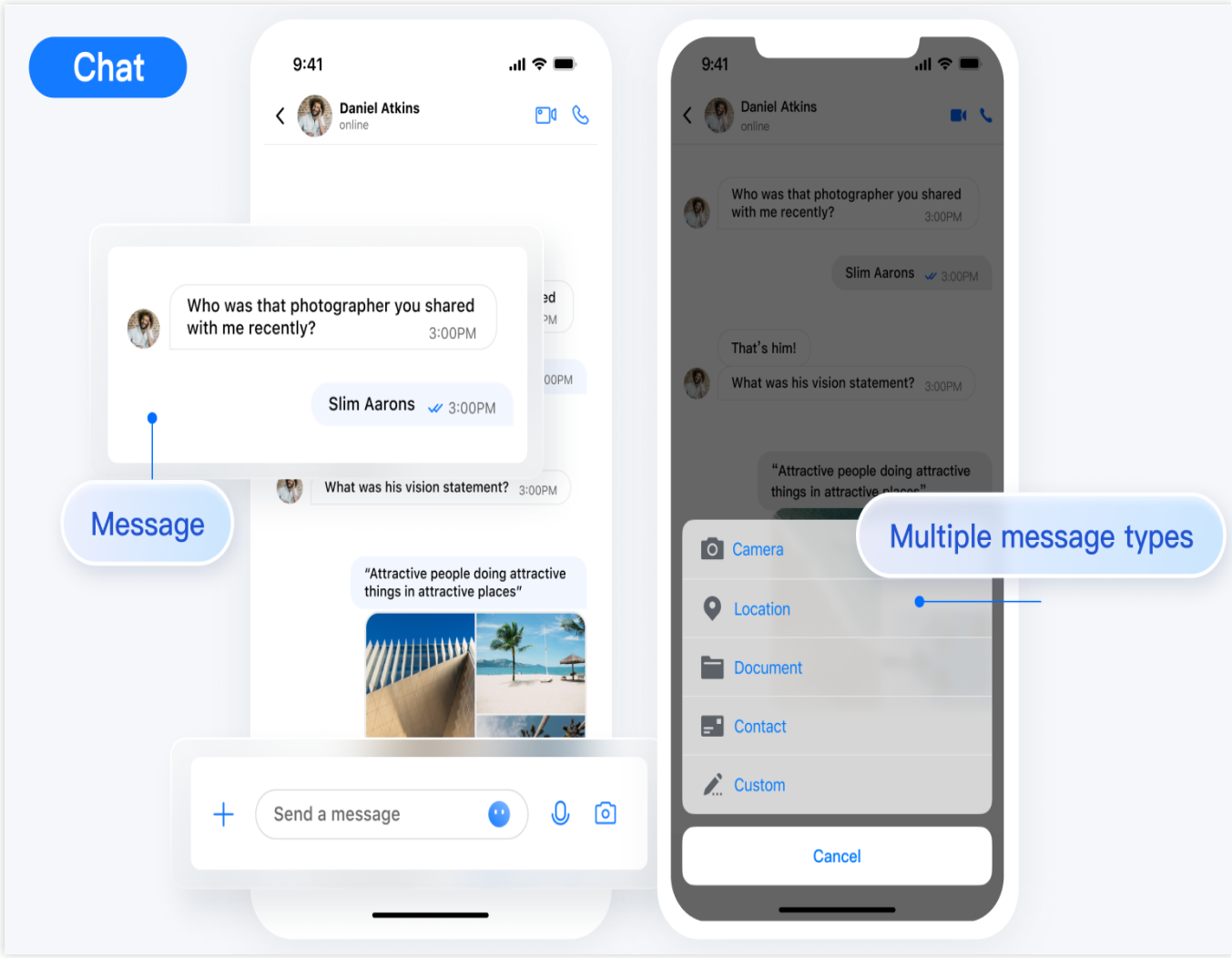
©2013-2025 Tencent Cloud International Pte. Ltd.



Chat 重点功能介绍

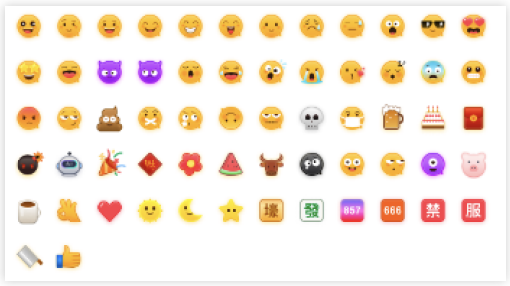
Chat 主要负责消息界面的展示。您可以利用它直接发送不同类型的消息，支持文字/表情/图片/视频/文件/自定义等消息类型，同时您还可以利用它对消息长按进行转发/撤回/引用、查询消息已读回执详情等。

©2013-2025 Tencent Cloud International Pte. Ltd.



说明：

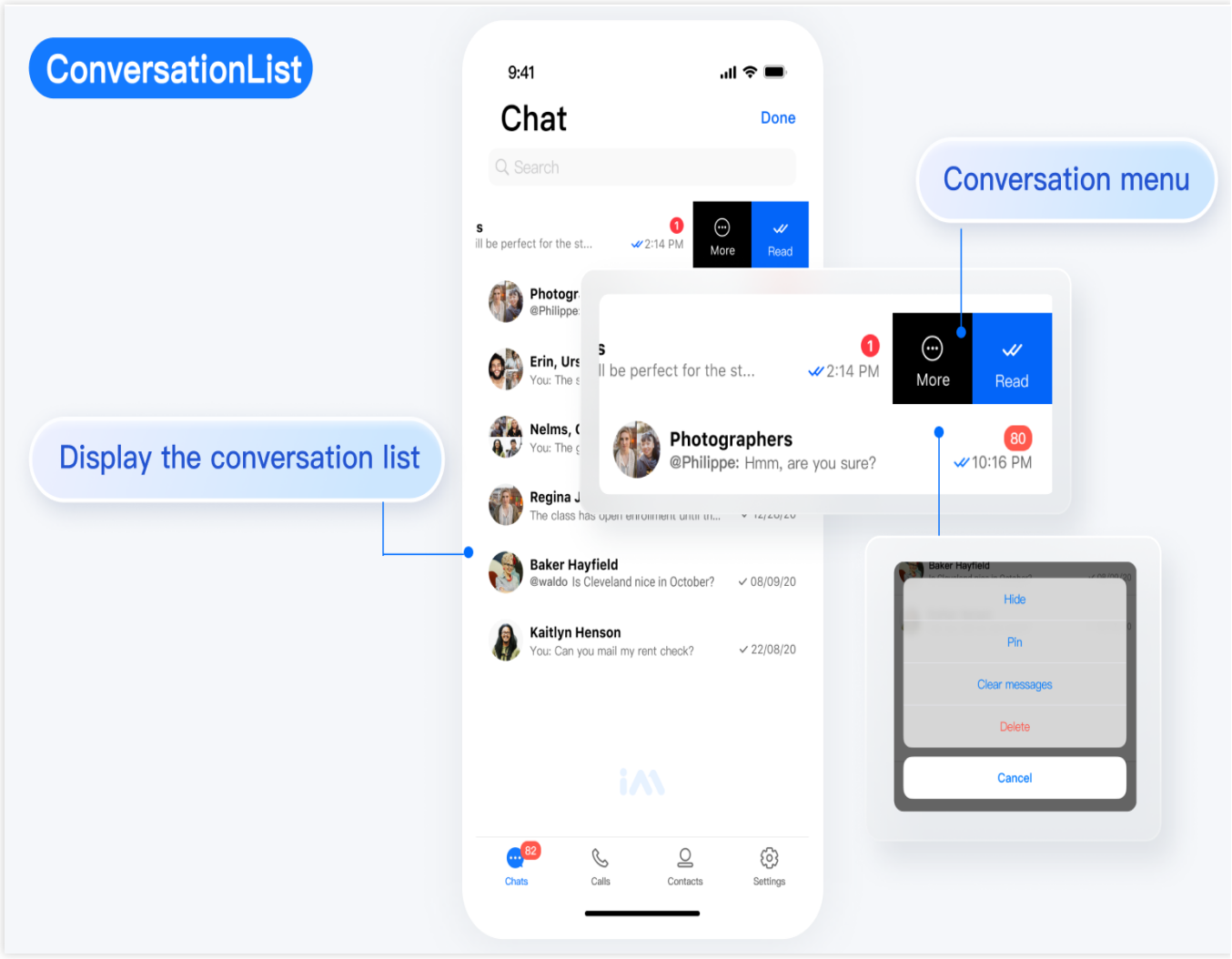
为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。



ConversationList 重点功能介绍

ConversationList 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。

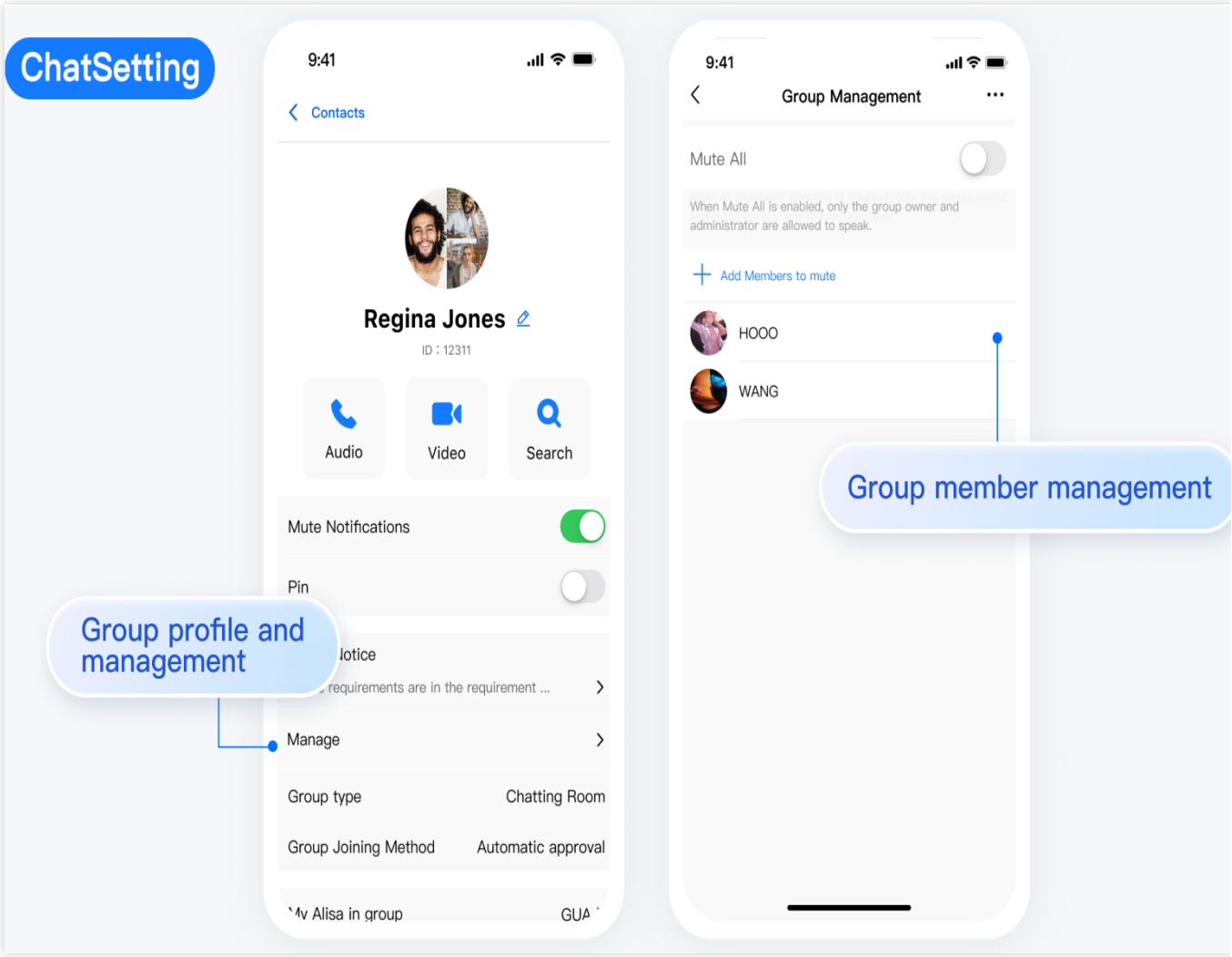
©2013-2025 Tencent Cloud International Pte. Ltd.



ChatSetting 重点功能介绍

ChatSetting 主要负责群资料、群成员、群组权限的管理。

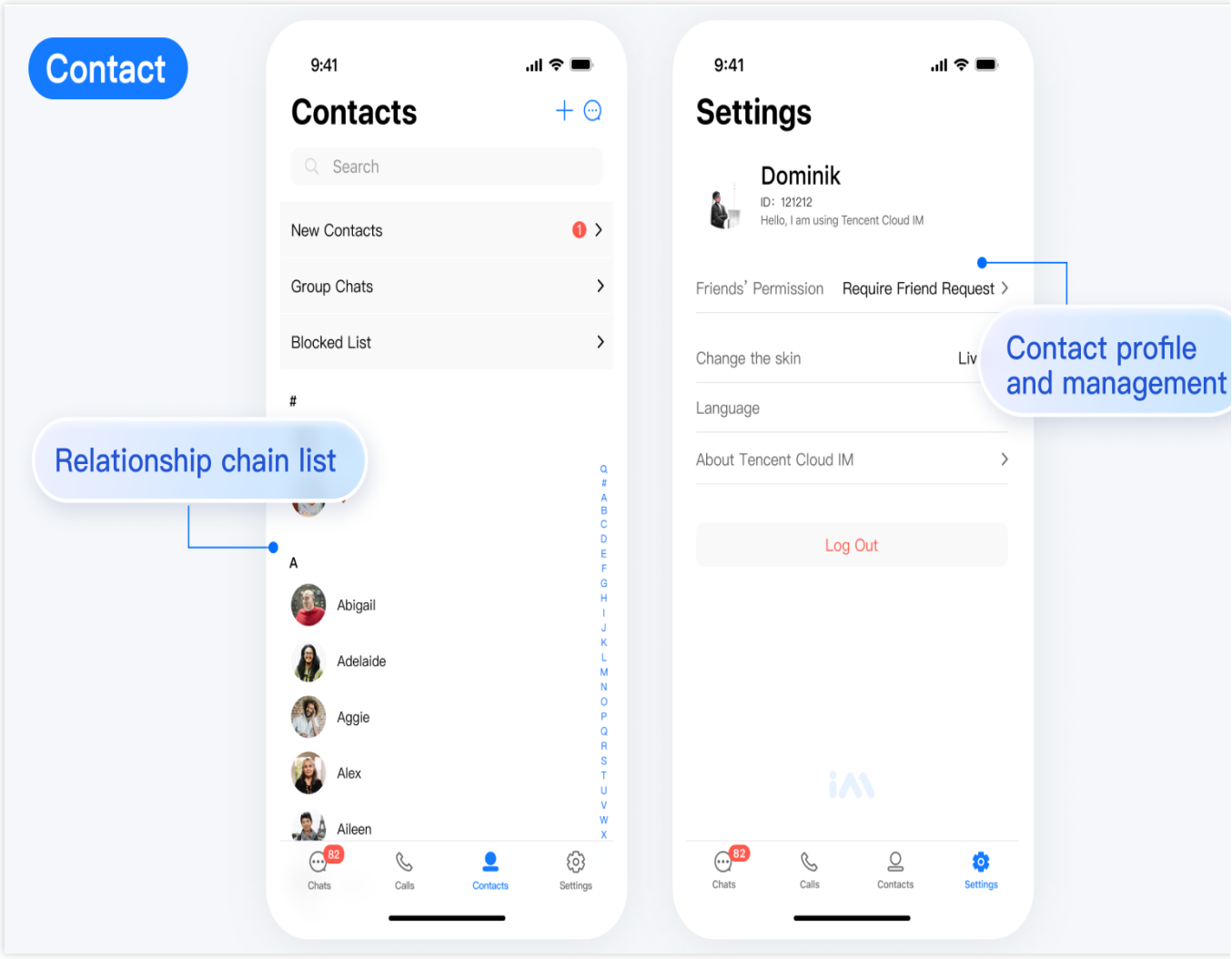
©2013-2025 Tencent Cloud International Pte. Ltd.



Contact 重点功能介绍

Contact 主要负责联系人的展示、权限设置等。

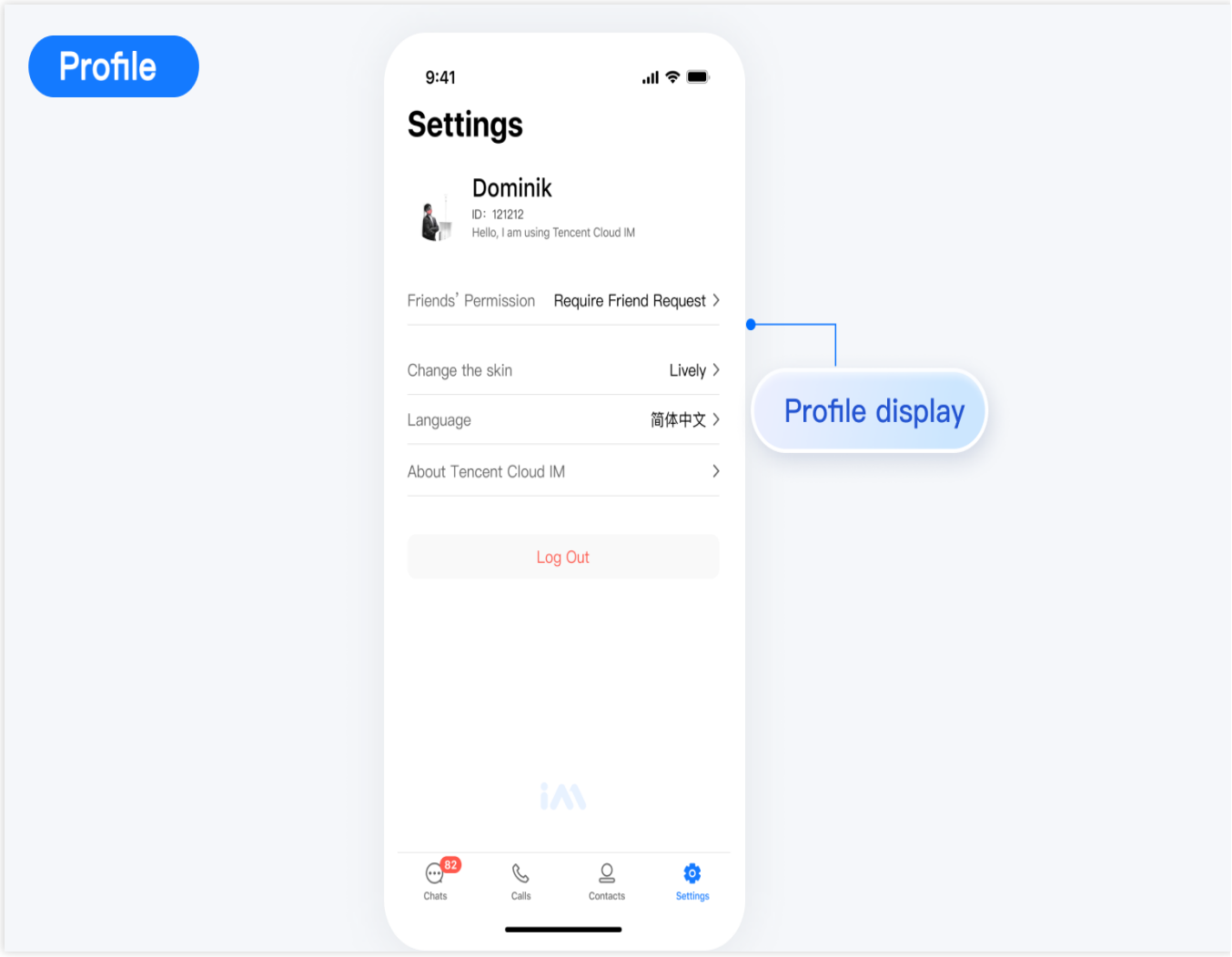
©2013-2025 Tencent Cloud International Pte. Ltd.



Profile 重点功能介绍

Profile 主要负责用户资料的管理。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

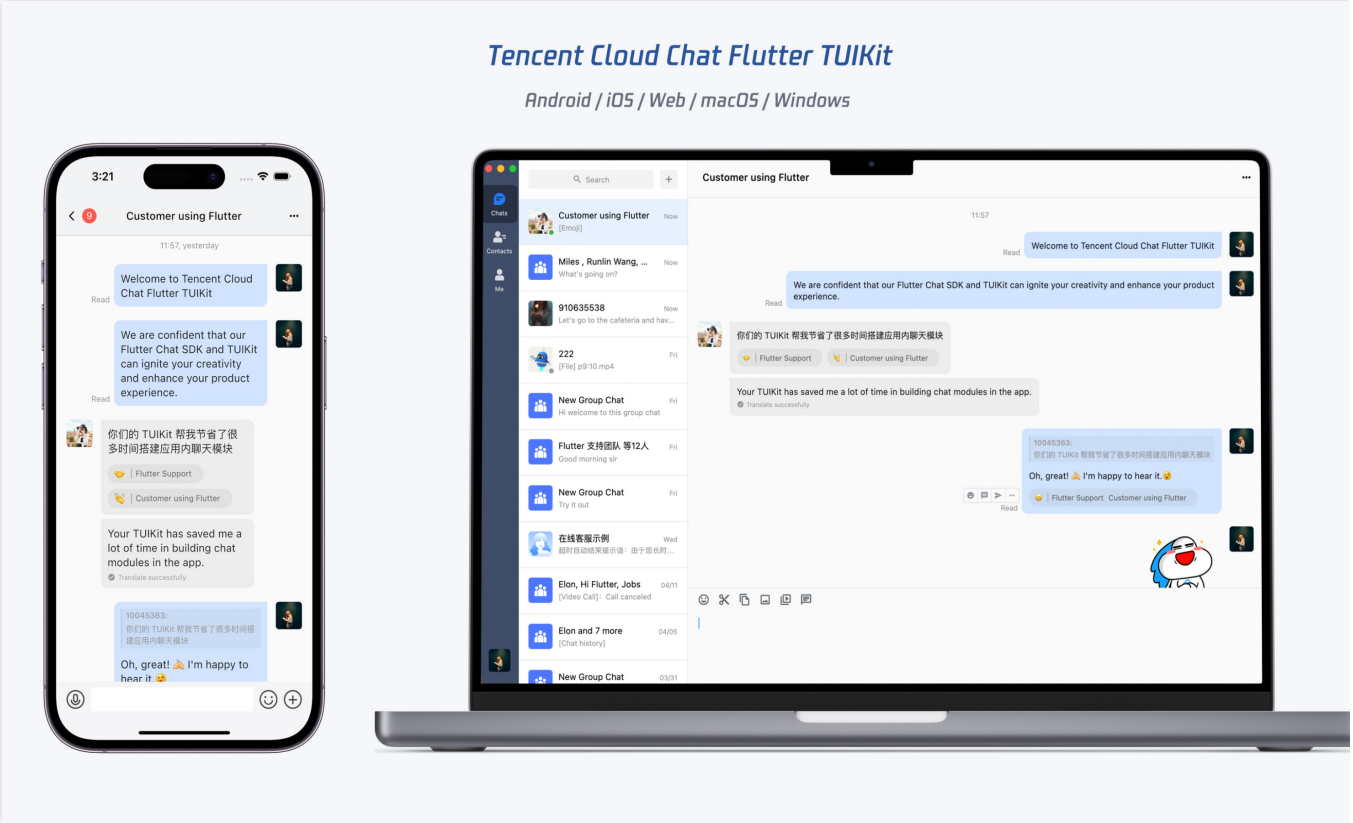
©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-03-03 11:25:44

TUIKit 介绍

TUIKit 是基于 腾讯云 Chat 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 Chat 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 Chat 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

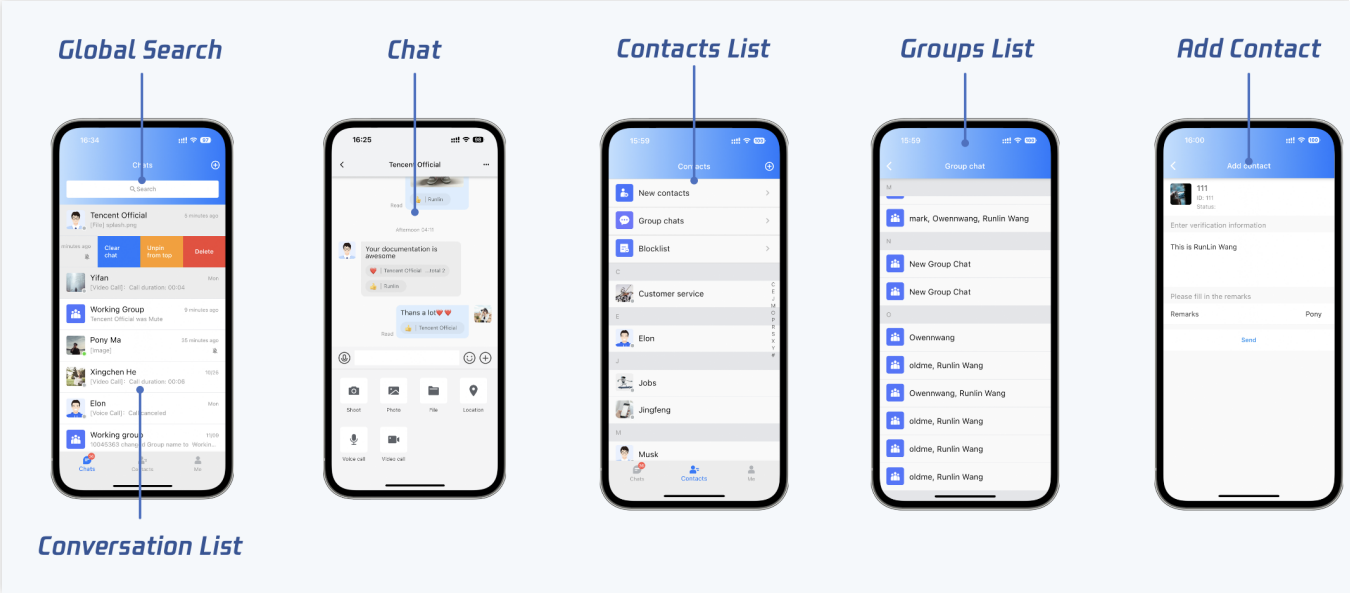
说明：
TUIKit 拥有自适应 UI 界面，可同时用于 **移动端 及 桌面端。**并支持以下所有平台，针对不同平台有不同的特色能力。
方便您**一套代码开发全平台应用**。

©2013-2025 Tencent Cloud International Pte. Ltd.

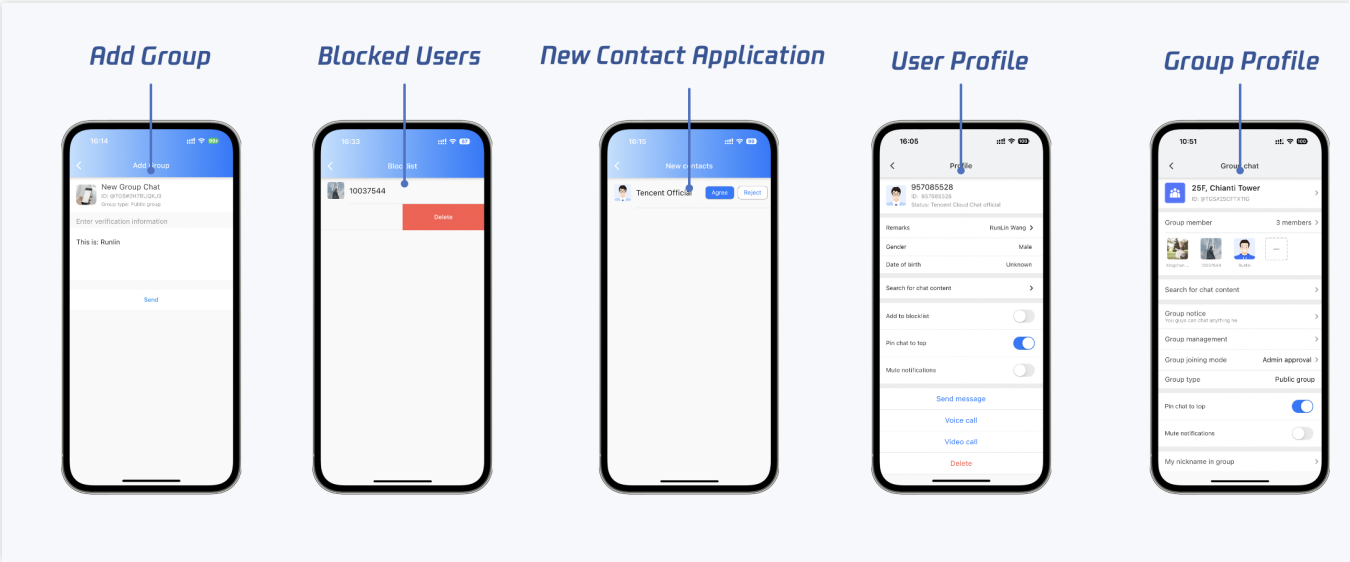
iOS / **Android** / **Web** (二维码) / **macOS** / **Windows** / 混合开发（将 Flutter Chat SDK 添加至现有原生应用）
单击跳转至对应平台 Demo 下载体验。以上各端 Demo 均由**同一项目代码**引入 TUIKit 开发编译打包而成。

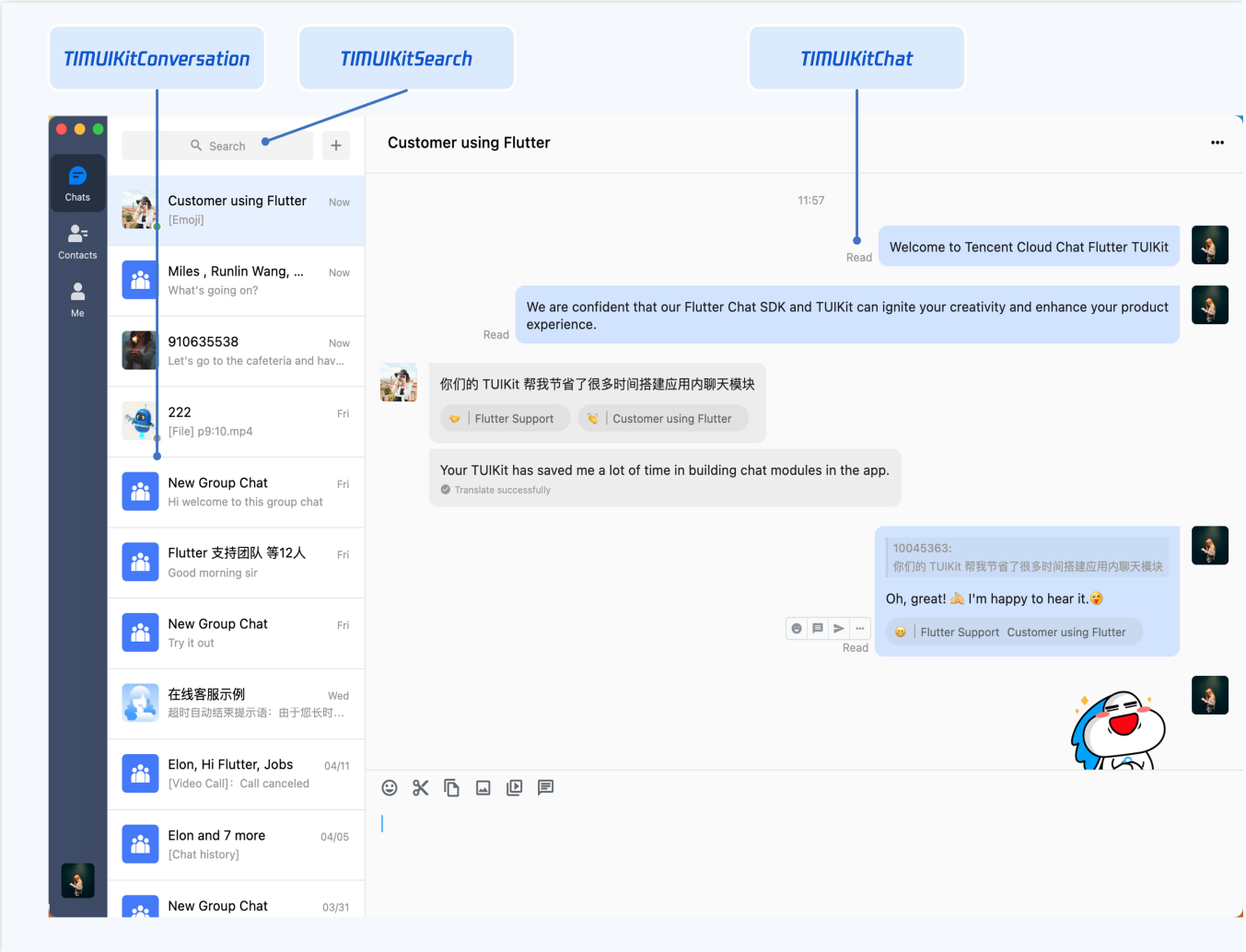
TUIKit 主要功能介绍

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

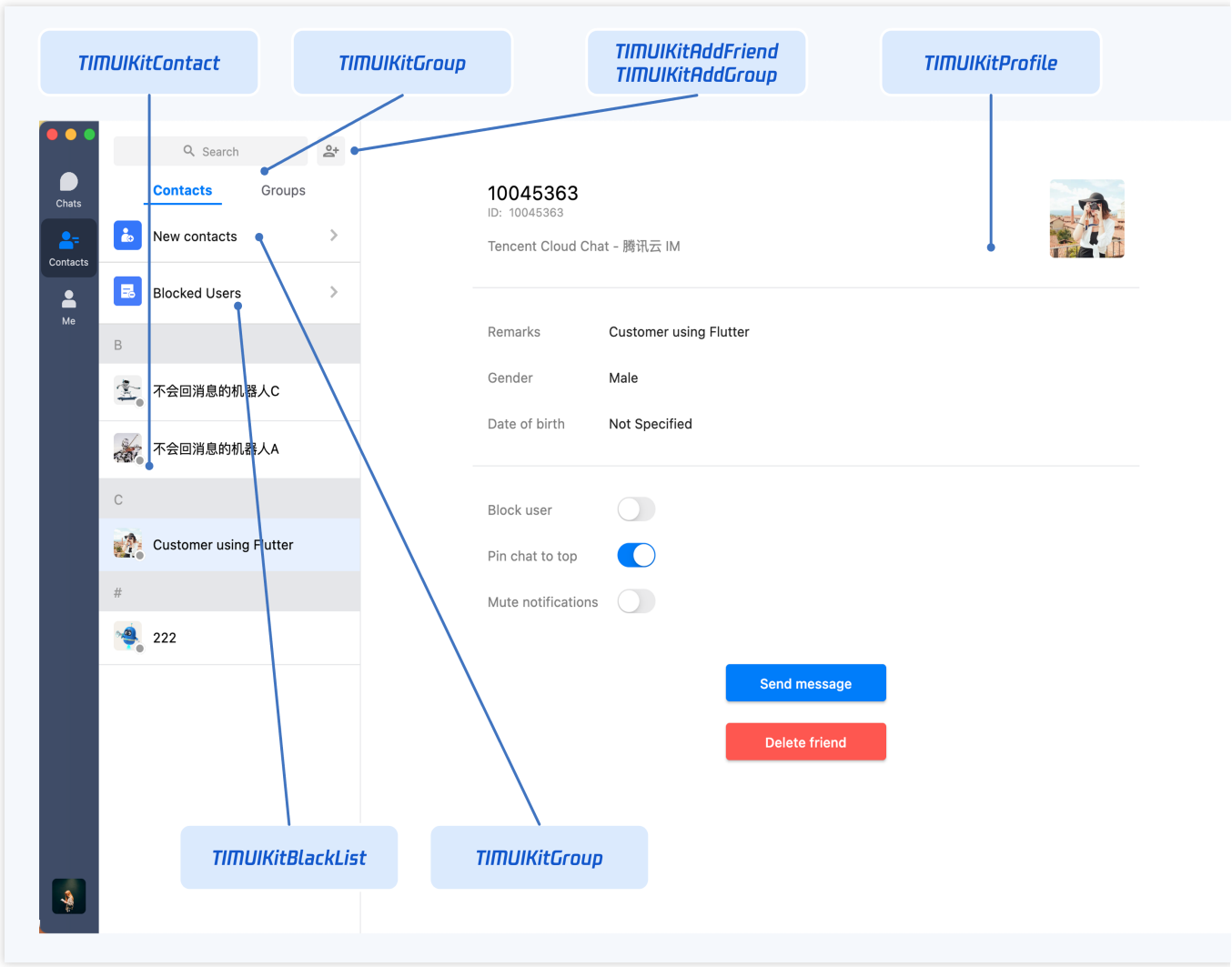


©2013-2025 Tencent Cloud International Pte. Ltd.





©2013-2025 Tencent Cloud International Pte. Ltd.



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。

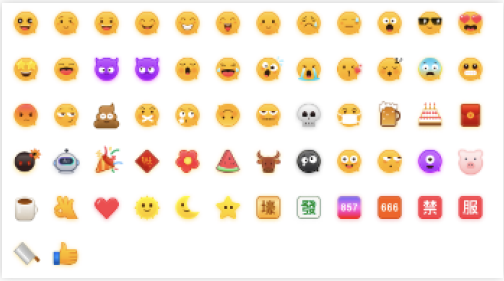
在桌面端，还包括 文件拖入发送 / 截图 / 粘贴图片发送 / 在目录中打开文件 等桌面端特色能力。

界面效果如下图所示：

说明：

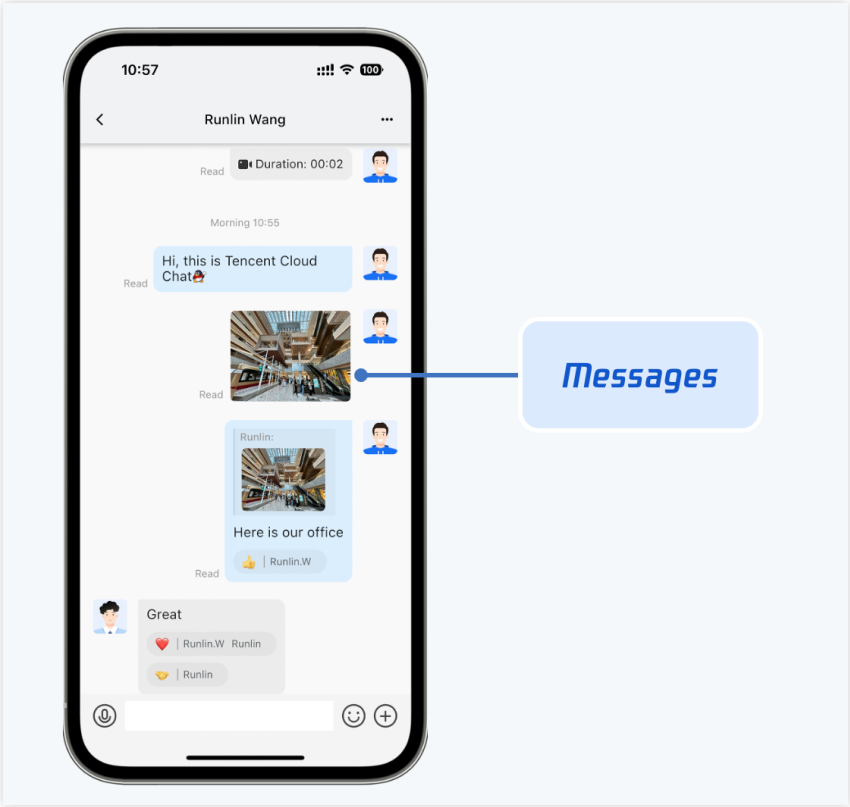
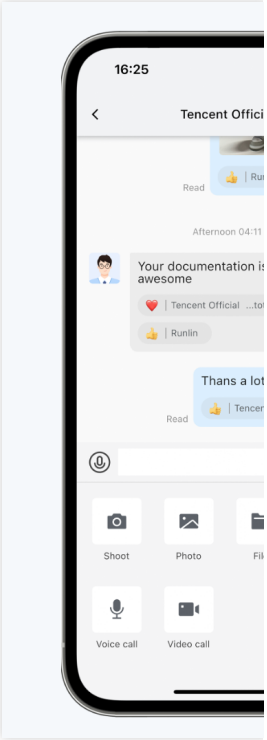
为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

©2013-2025 Tencent Cloud International Pte. Ltd.



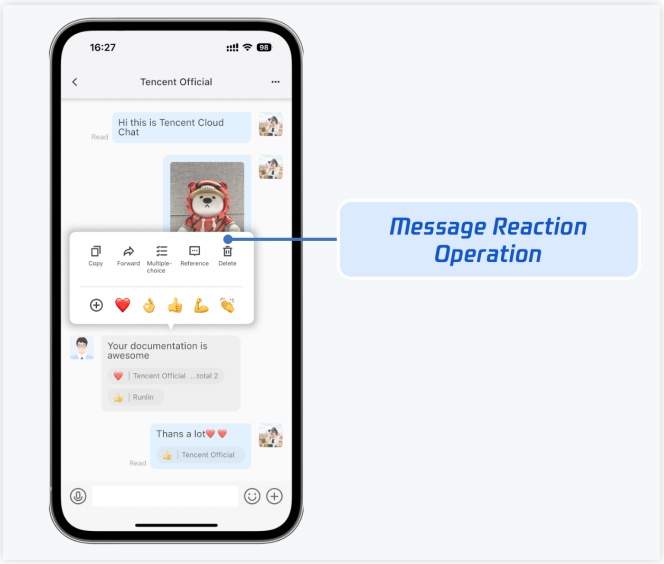
移动端

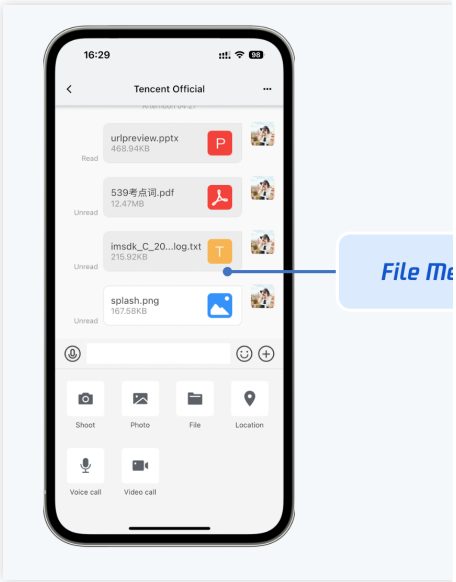
桌面端

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

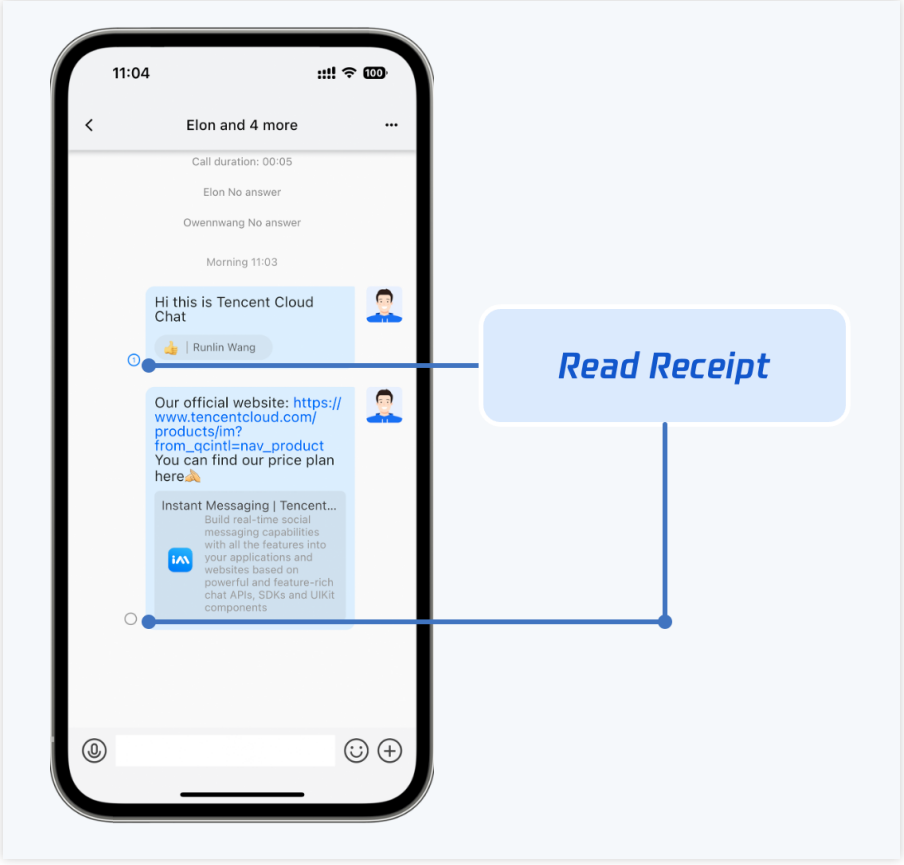
©2013-2025 Tencent Cloud International Pte. Ltd.

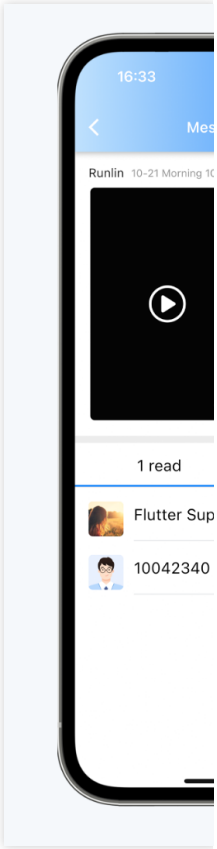




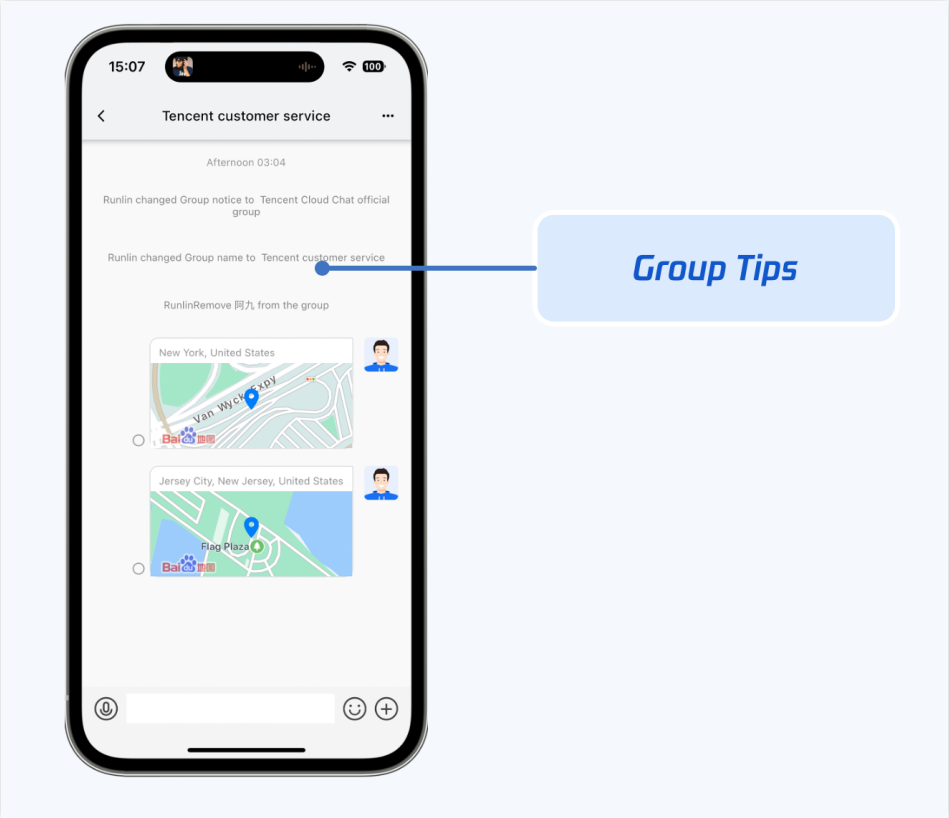
消息已读回执

已读回执详情



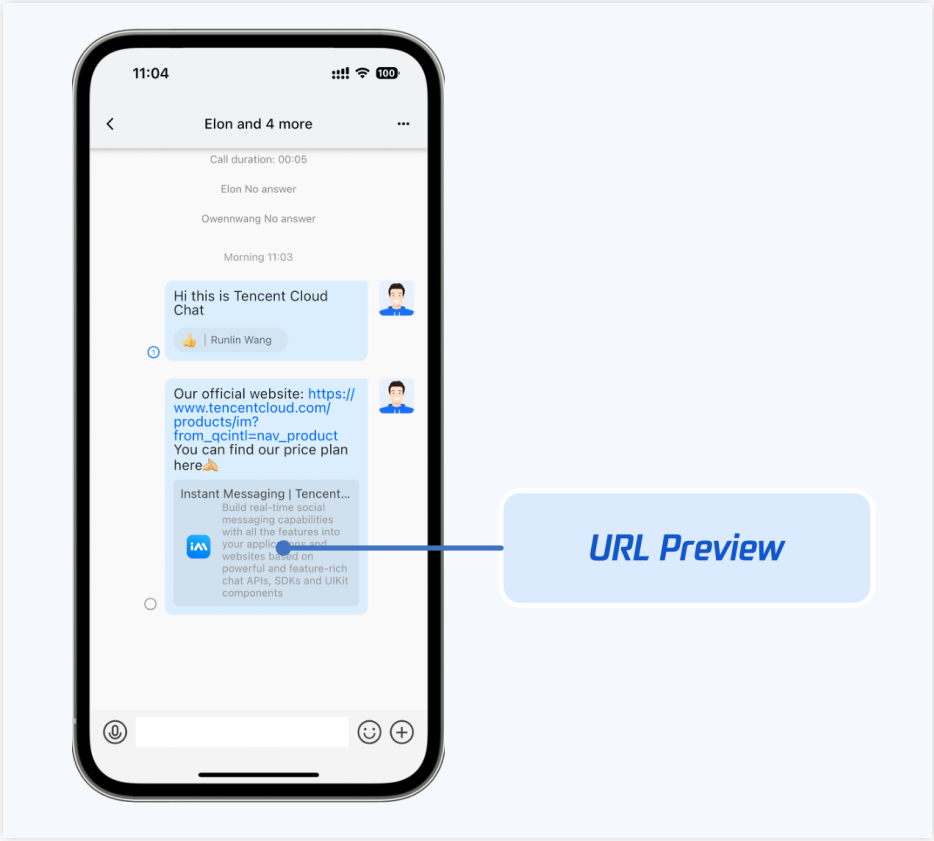


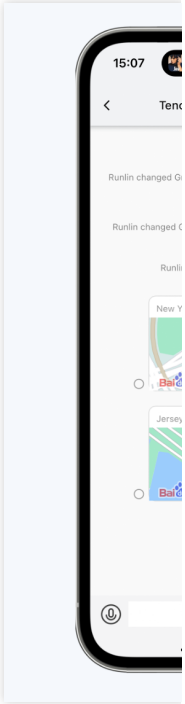
©2013-2025 Tencent Cloud International Pte. Ltd.

群 tips 消息	入群申请审批
	

链接解析预览	地理位置消息

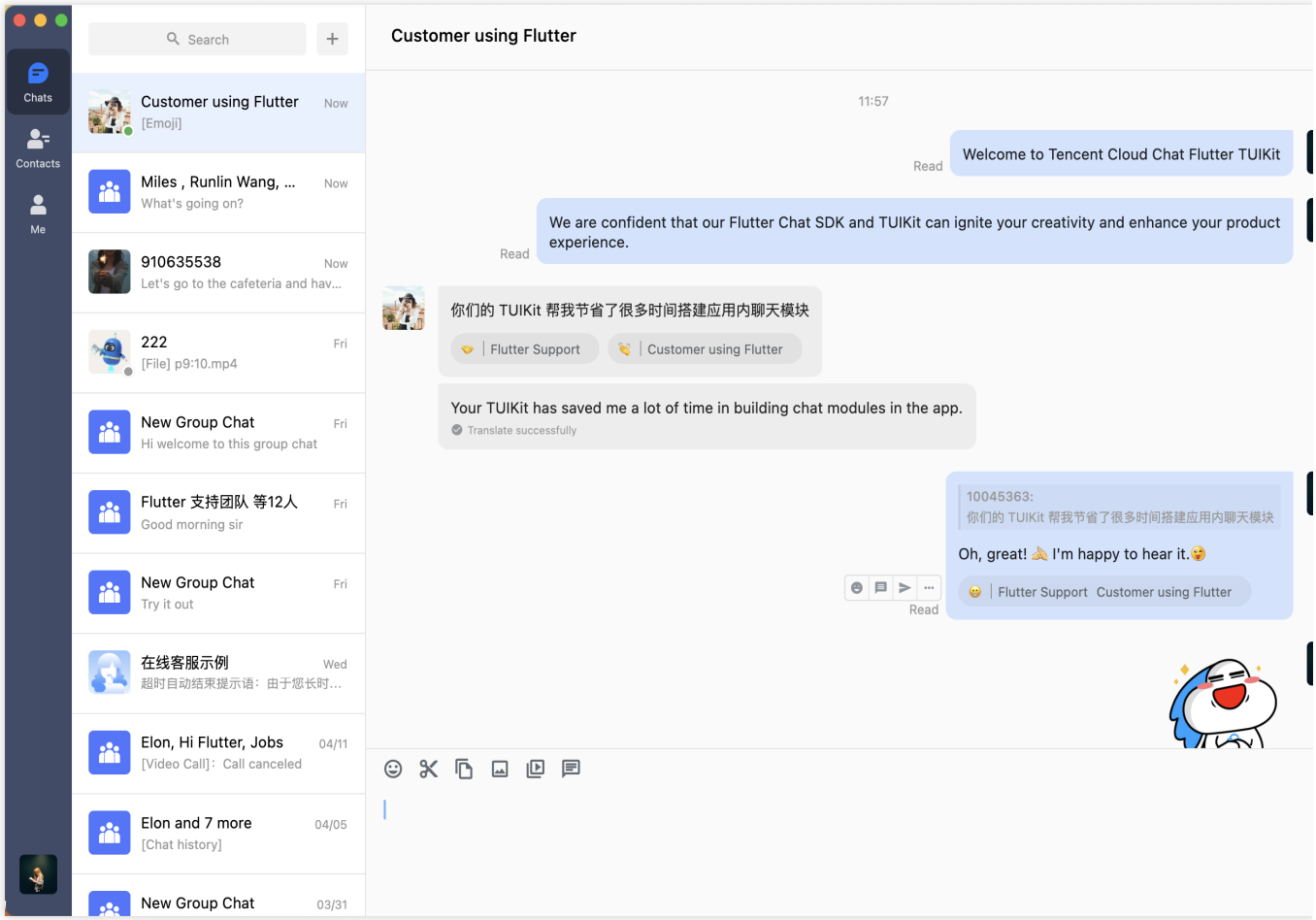
©2013-2025 Tencent Cloud International Pte. Ltd.





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。

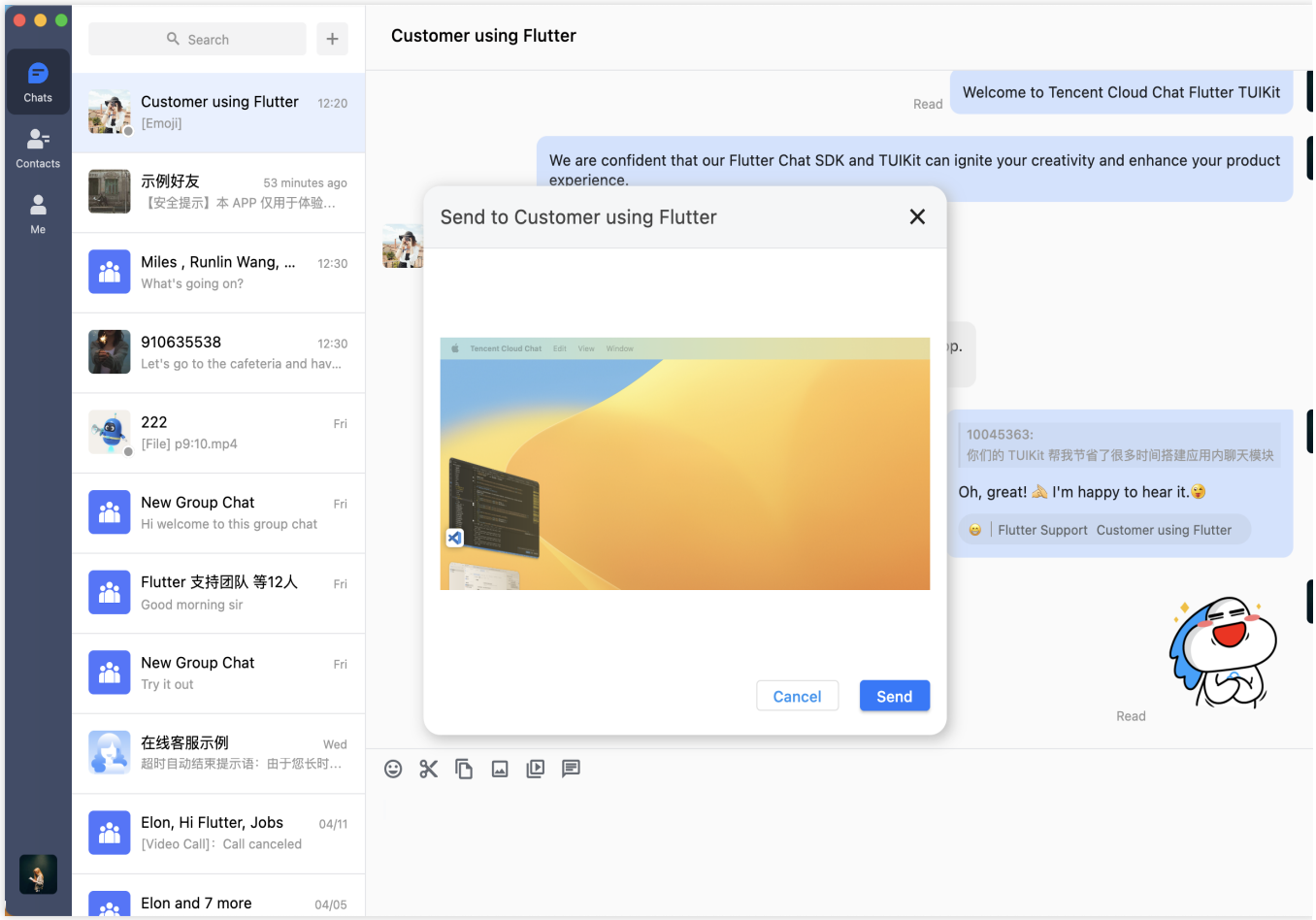
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

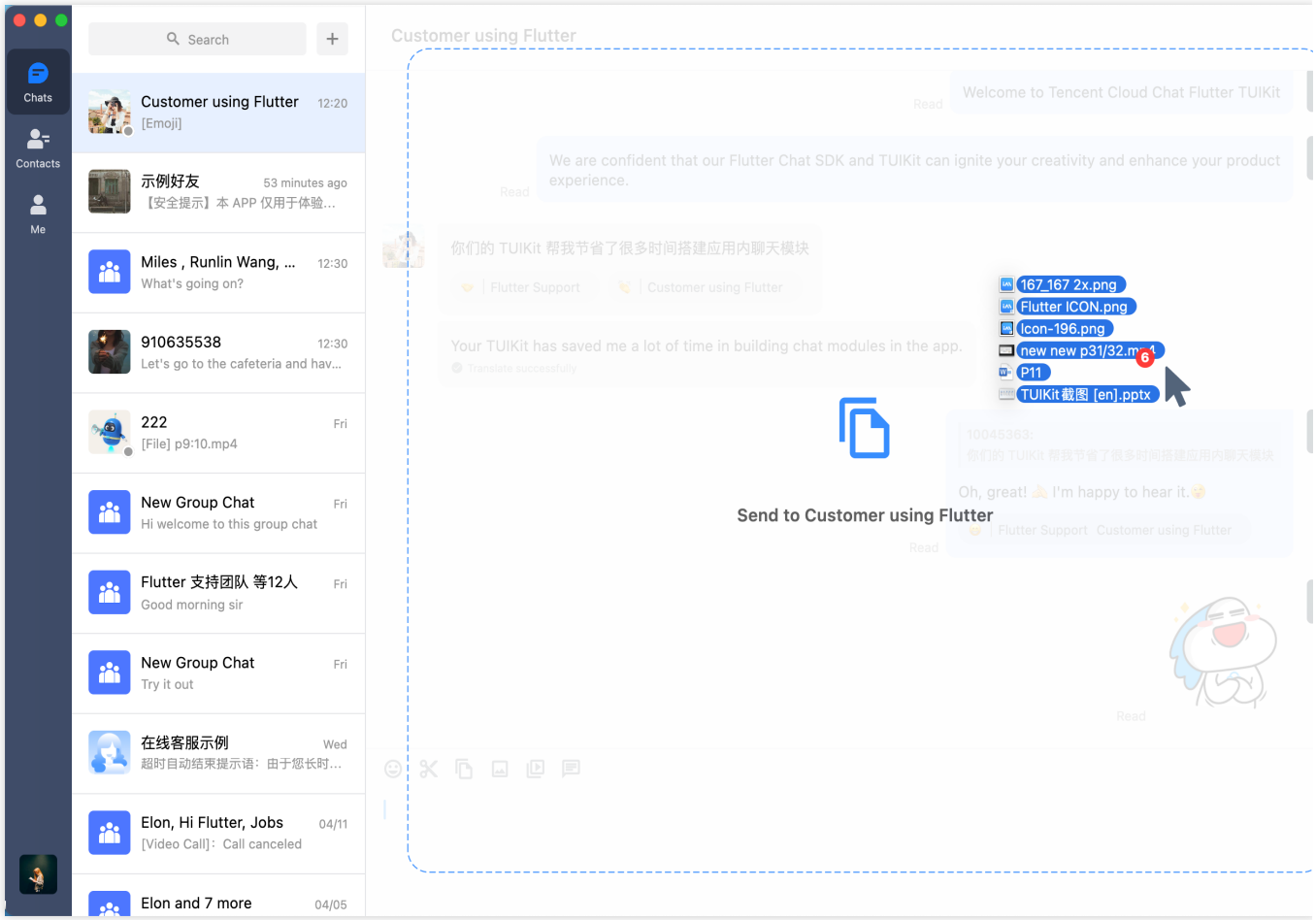
截图 或 粘贴图片至发送区域 直接发送图片。

©2013-2025 Tencent Cloud International Pte. Ltd.

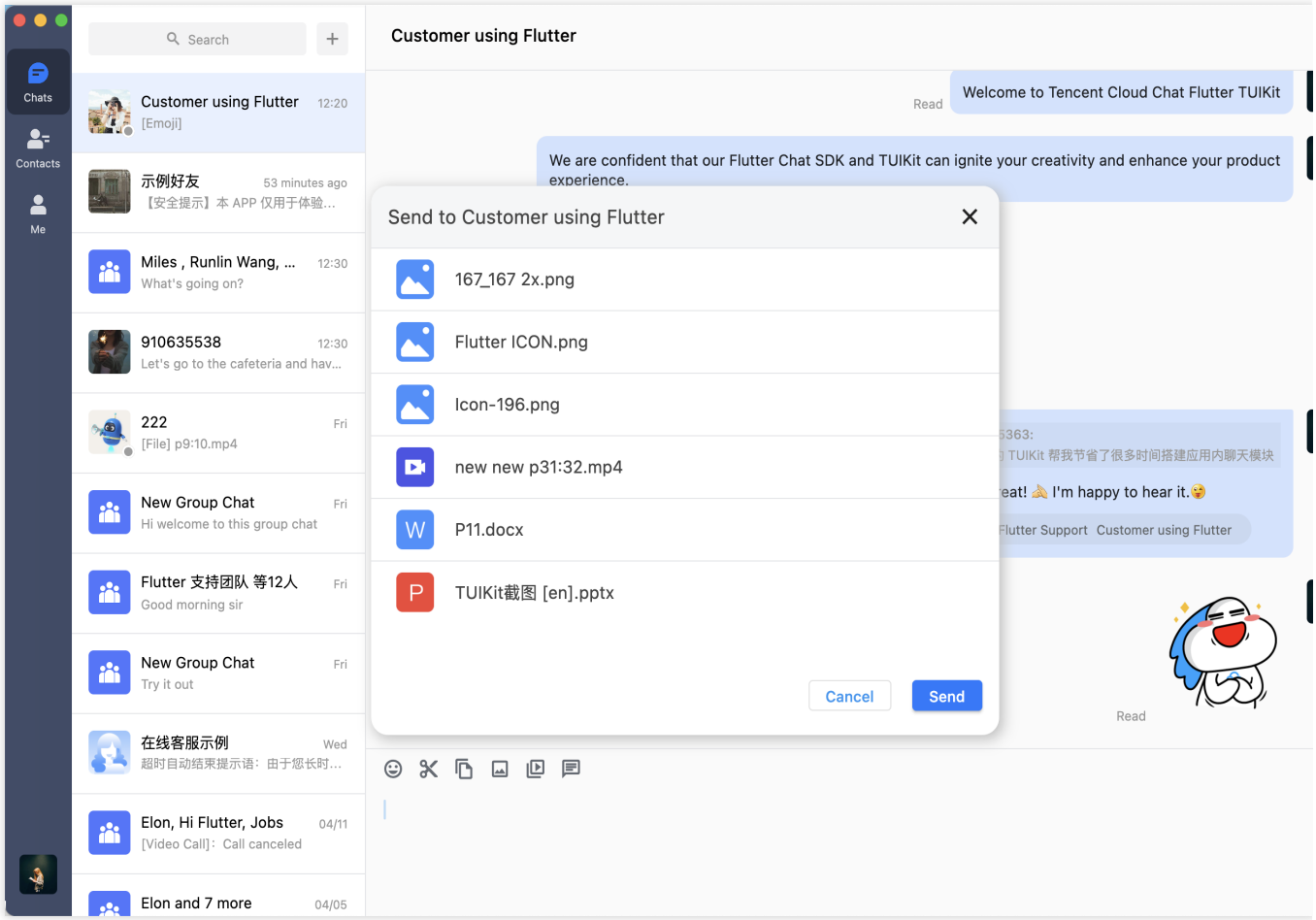


拖入多个文件直接发送。

©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.

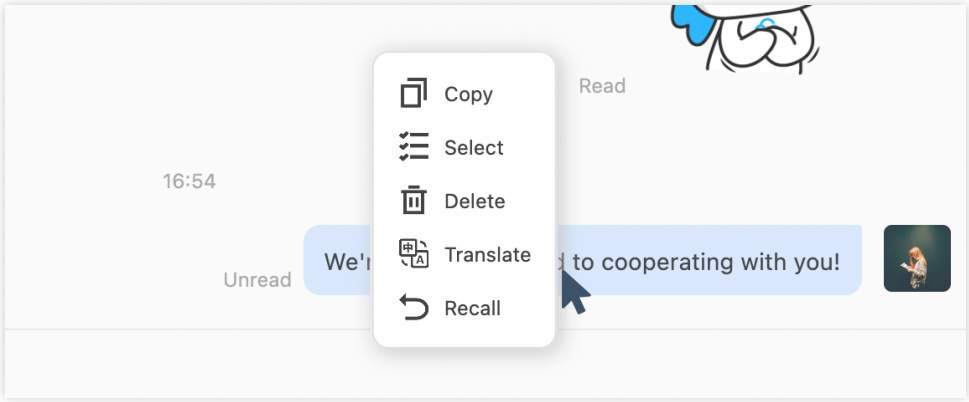


消息上，鼠标 Hover 菜单。包含 表情回应 / 消息回复 / 转发 及更多能力。

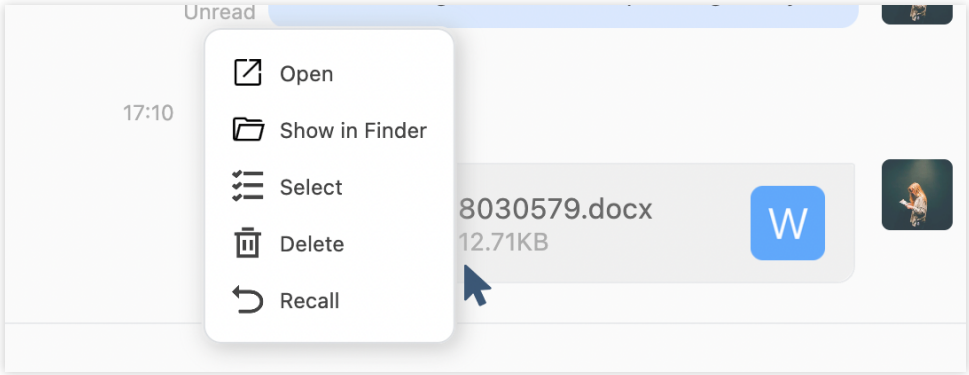


消息上，鼠标右键菜单。包含更多消息操作，如 复制 / 多选 / 删除 / 翻译 / 撤回 等。

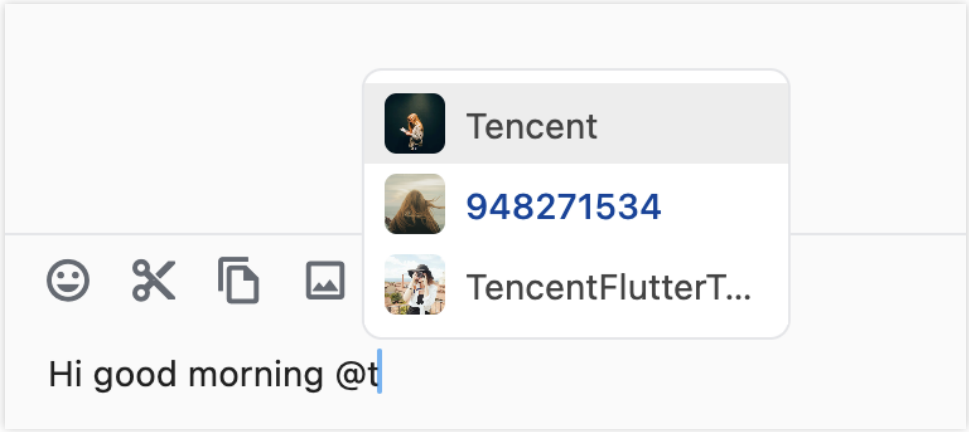
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

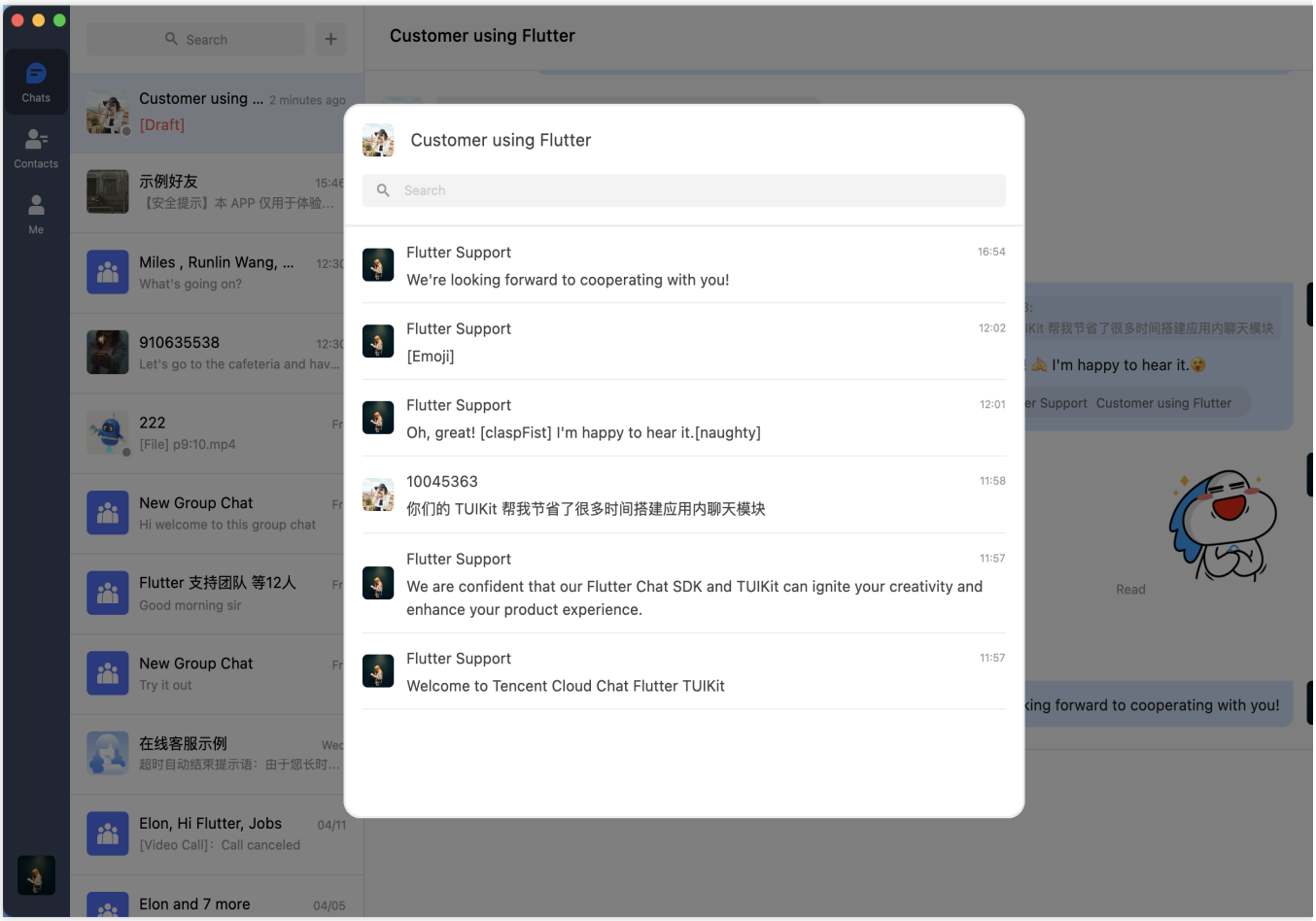


群聊中 @ 其他群成员。在群成员选择面板中，可渐进式输入搜索并选择群成员。被 @ 的群成员可收到特别提醒。



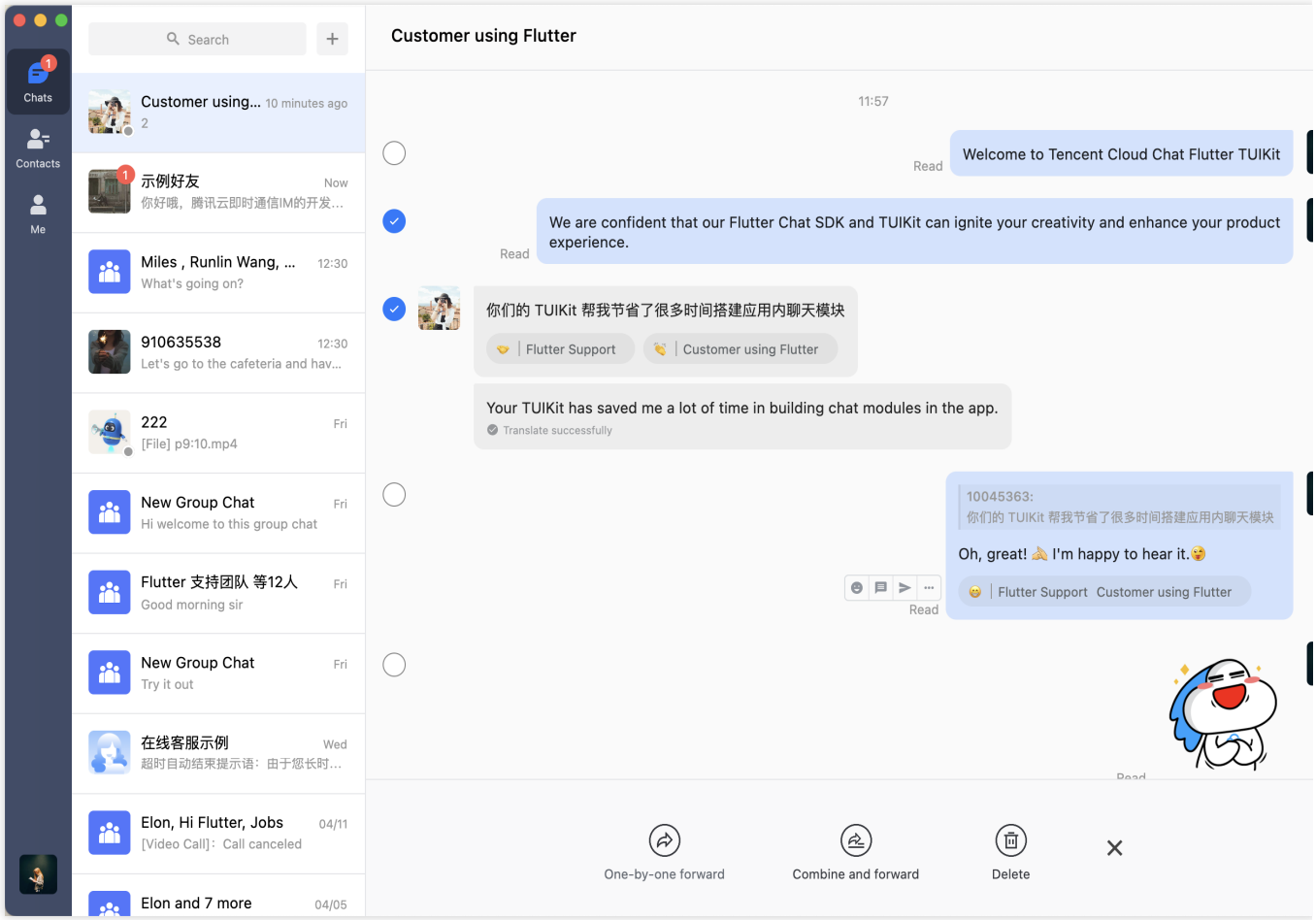
历史消息盒子。支持关键词搜索。

©2013-2025 Tencent Cloud International Pte. Ltd.



消息多选。

©2013-2025 Tencent Cloud International Pte. Ltd.



关系链系列组件

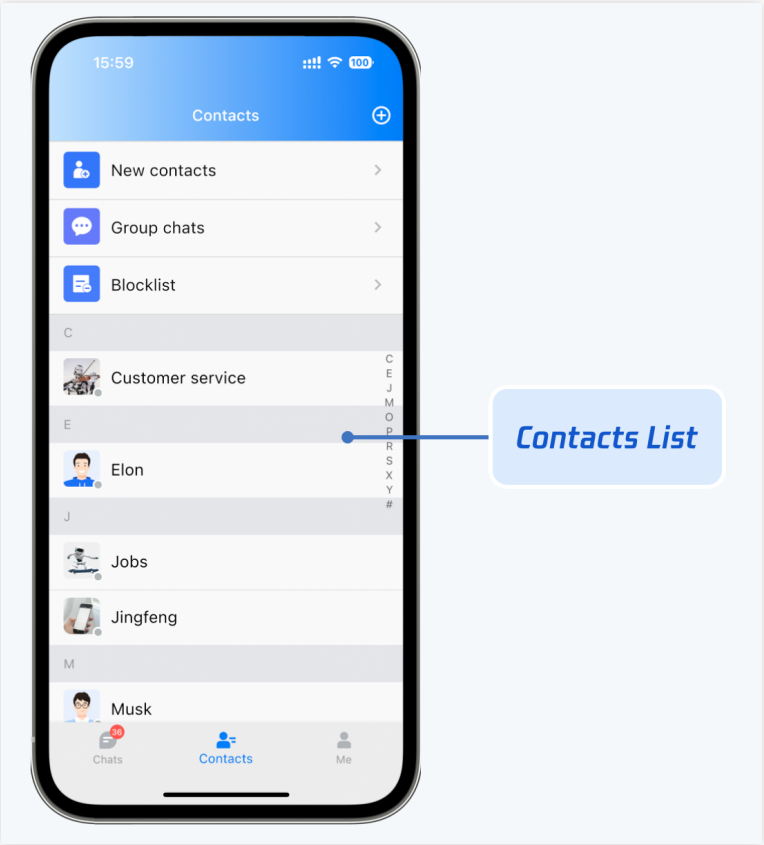
关系链主要负责当前用户的联系人、群聊、黑名单的信息展示。界面效果如下图所示：

移动端

桌面端

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

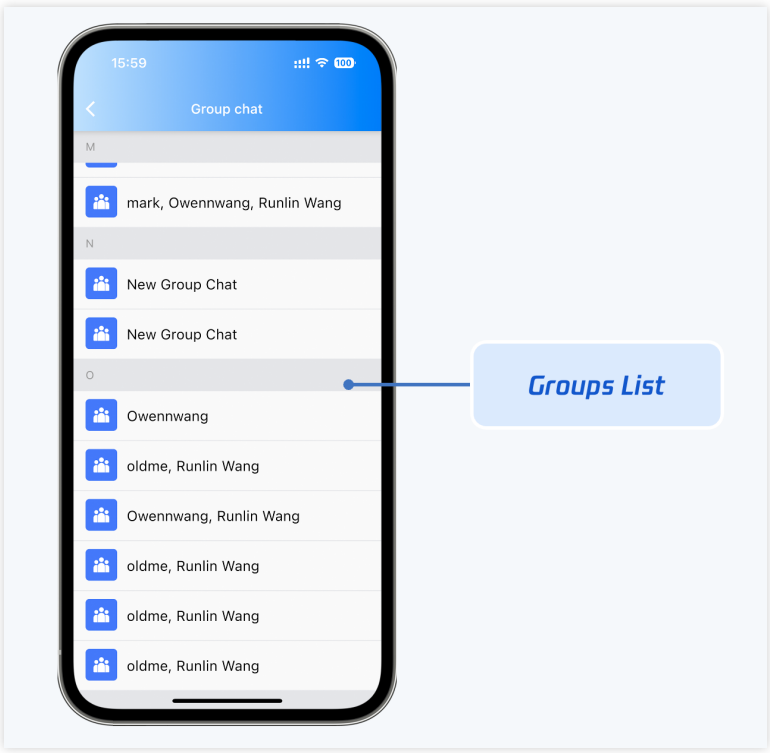
©2013-2025 Tencent Cloud International Pte. Ltd.

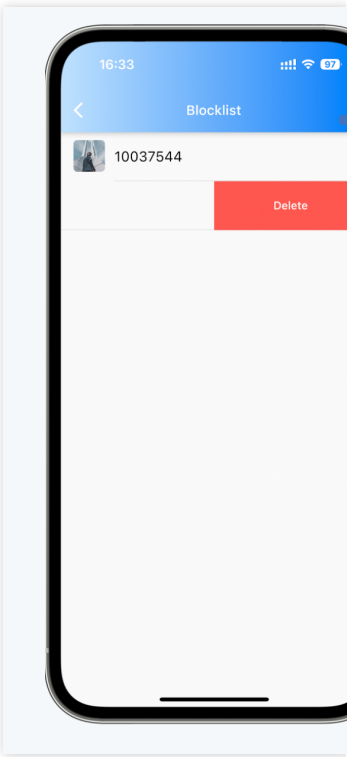




参与的群聊列表(TIMUIKitGroup)	黑名单列表(TIMUIKitBlackList)

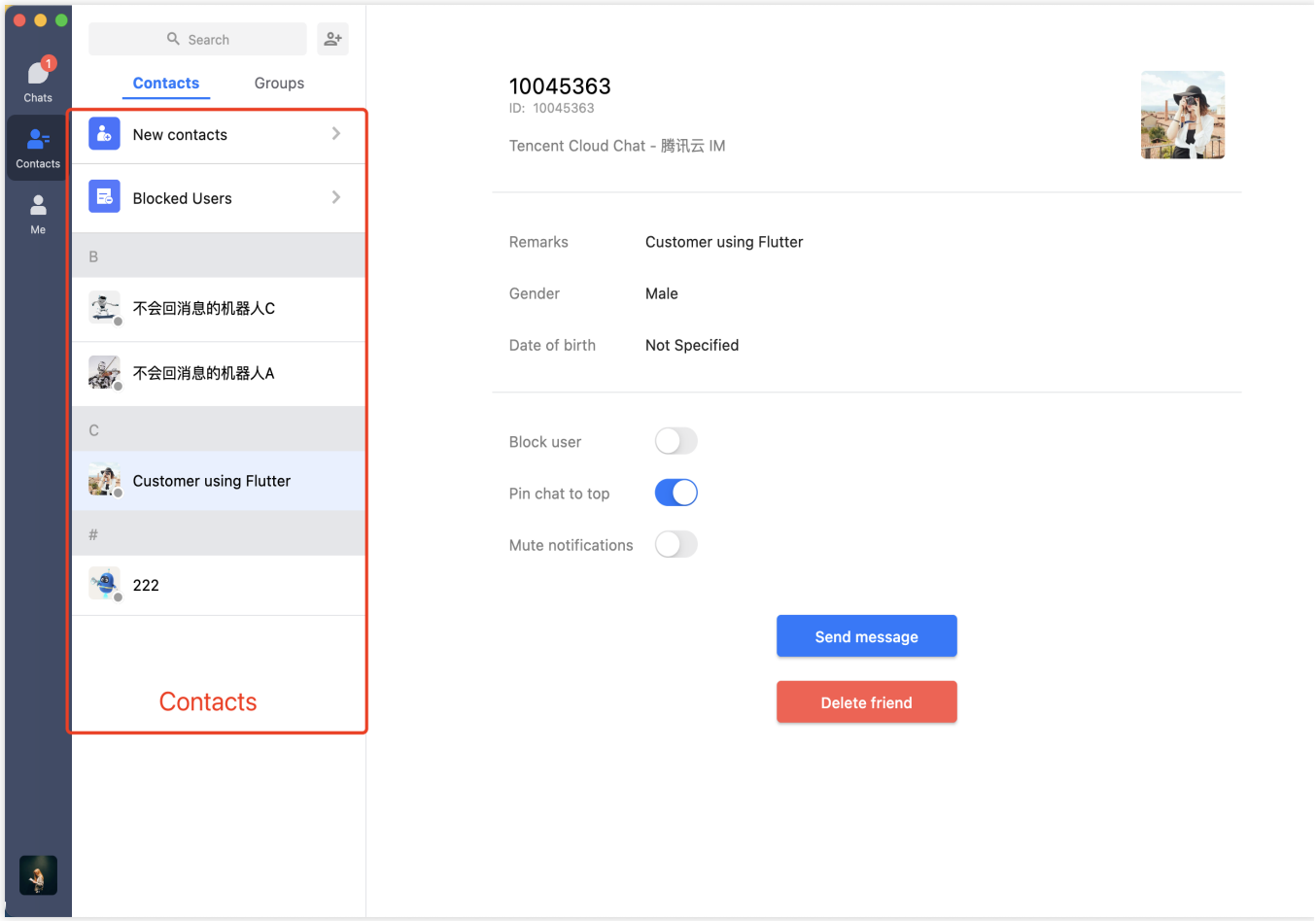
©2013-2025 Tencent Cloud International Pte. Ltd.





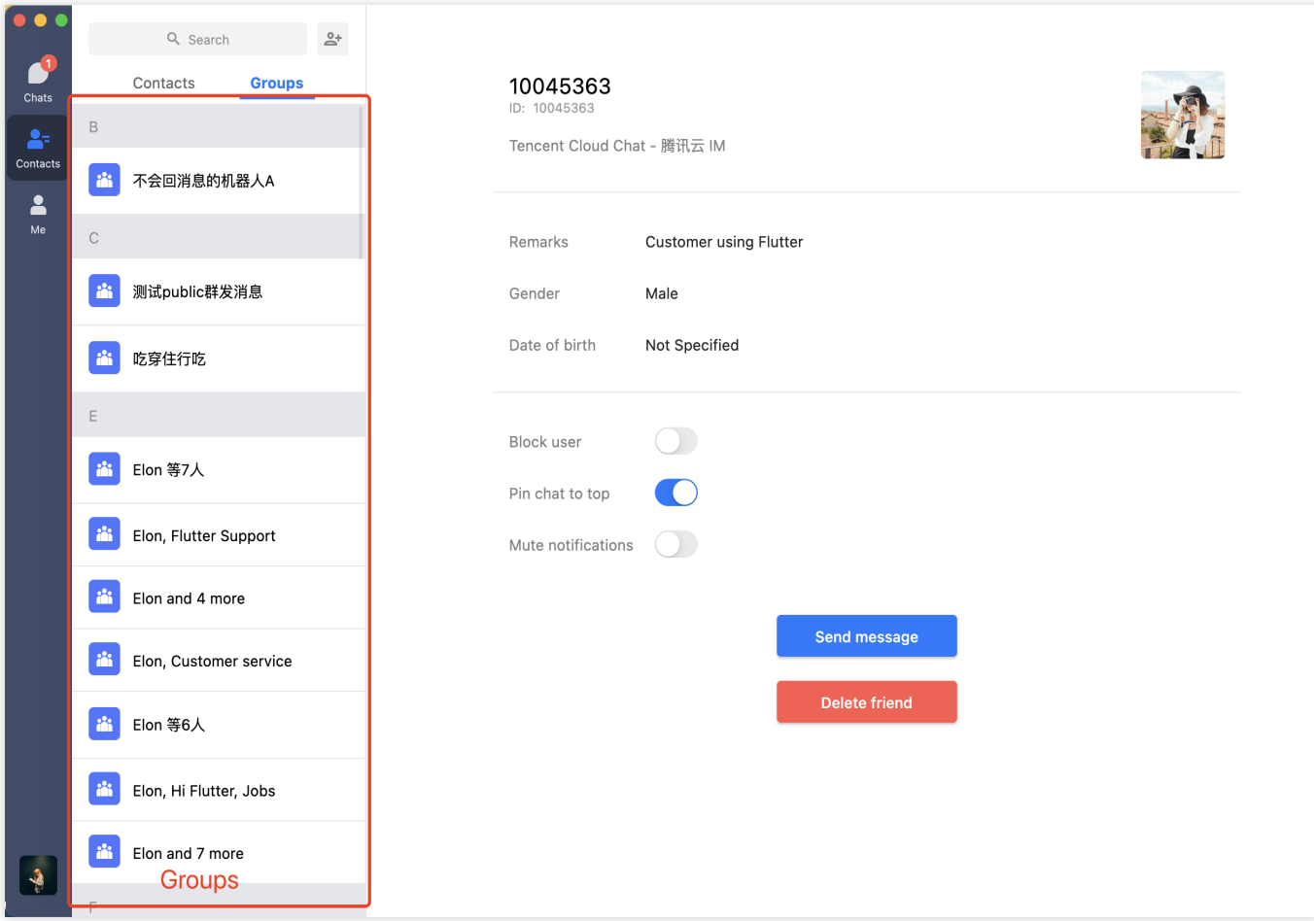
联系人列表 - TIMTUIKitContact

©2013-2025 Tencent Cloud International Pte. Ltd.



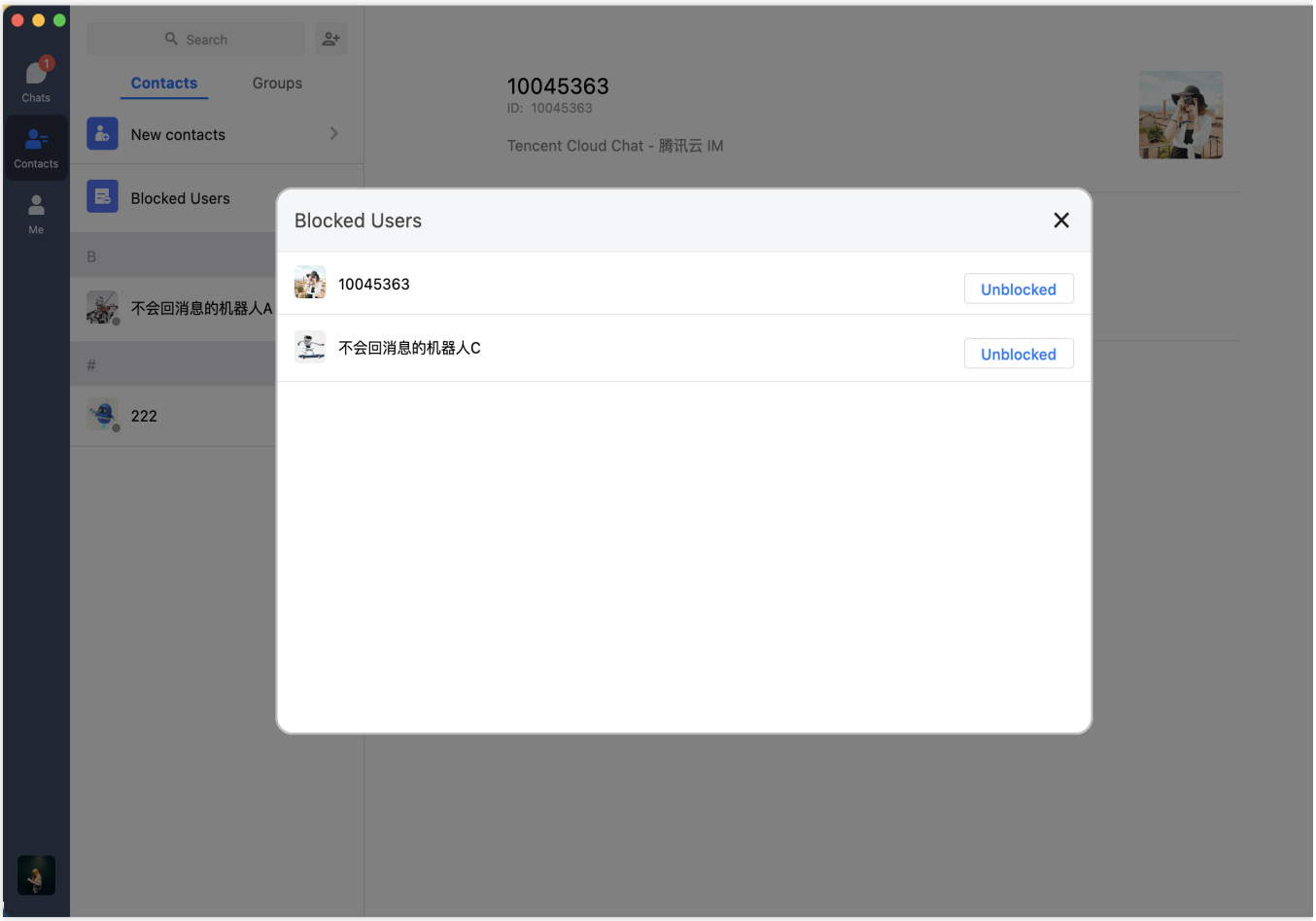
群组列表 - TIMUIKitGroup

©2013-2025 Tencent Cloud International Pte. Ltd.



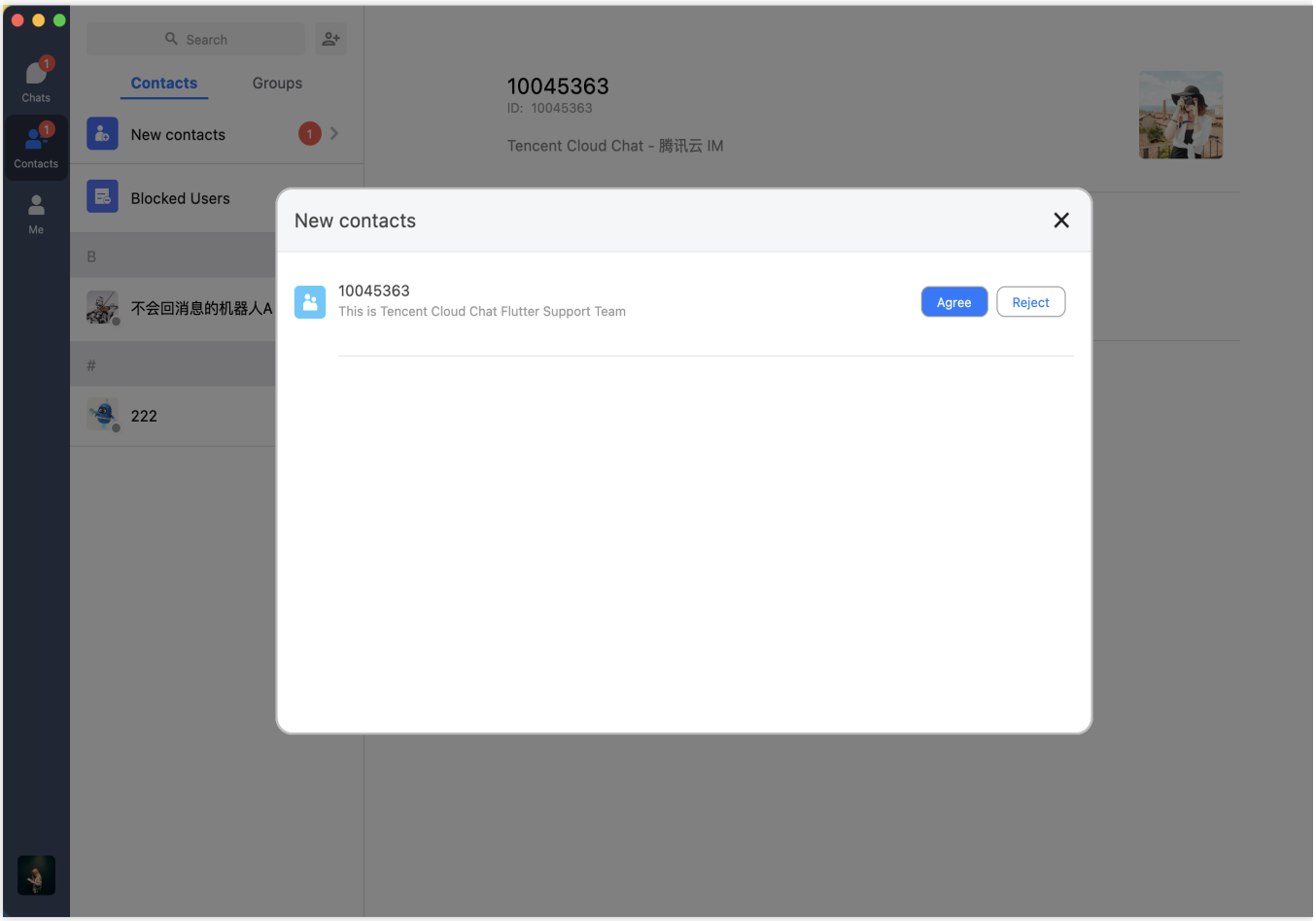
黑名单列表 - TIMUIKitBlackList

©2013-2025 Tencent Cloud International Pte. Ltd.



好友申请列表 - TIMUIKitNewContact

©2013-2025 Tencent Cloud International Pte. Ltd.



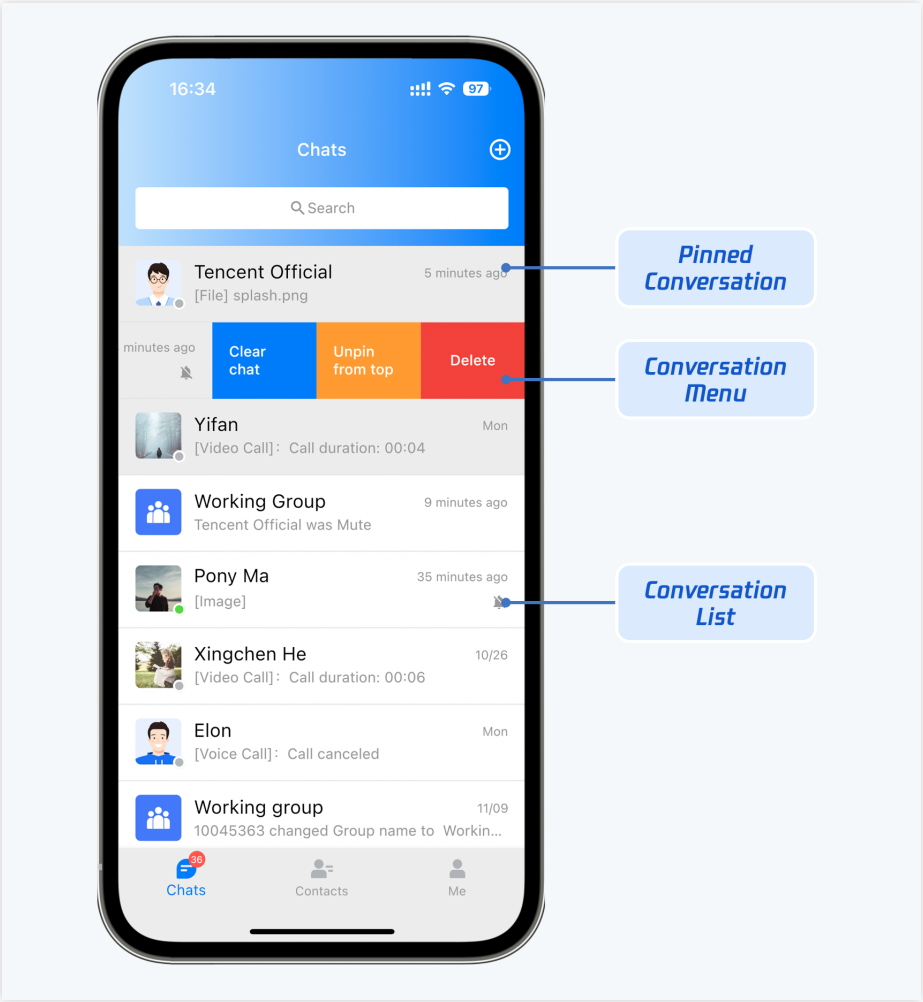
会话列表组件

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

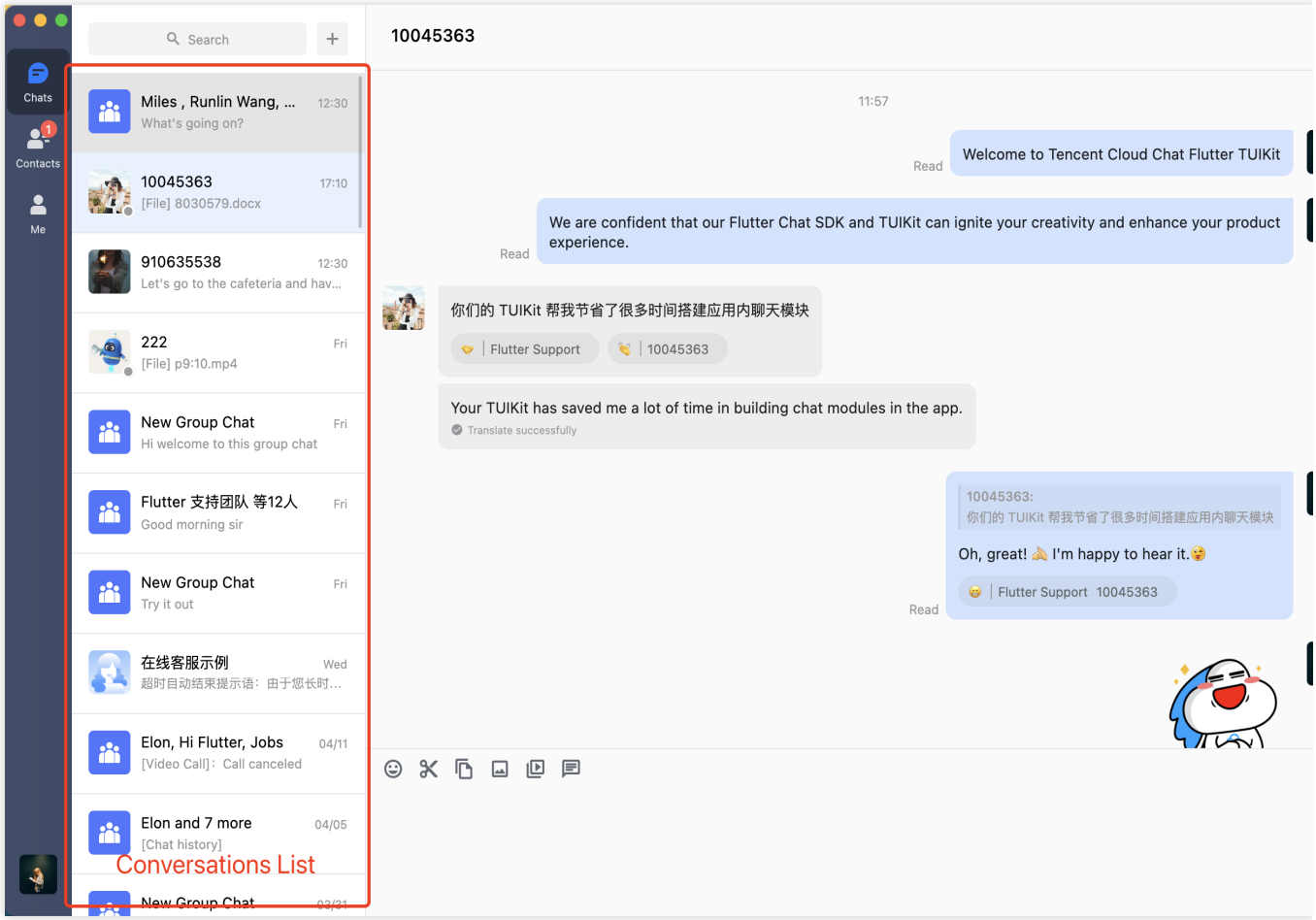
桌面端

©2013-2025 Tencent Cloud International Pte. Ltd.

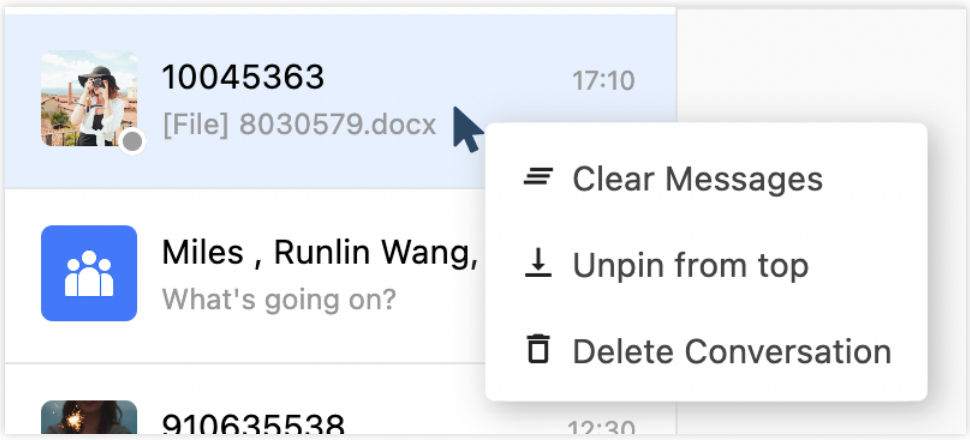


会话列表。当前选中会话 / 置顶的会话 / 未选中会话 分别用不同颜色展示。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。



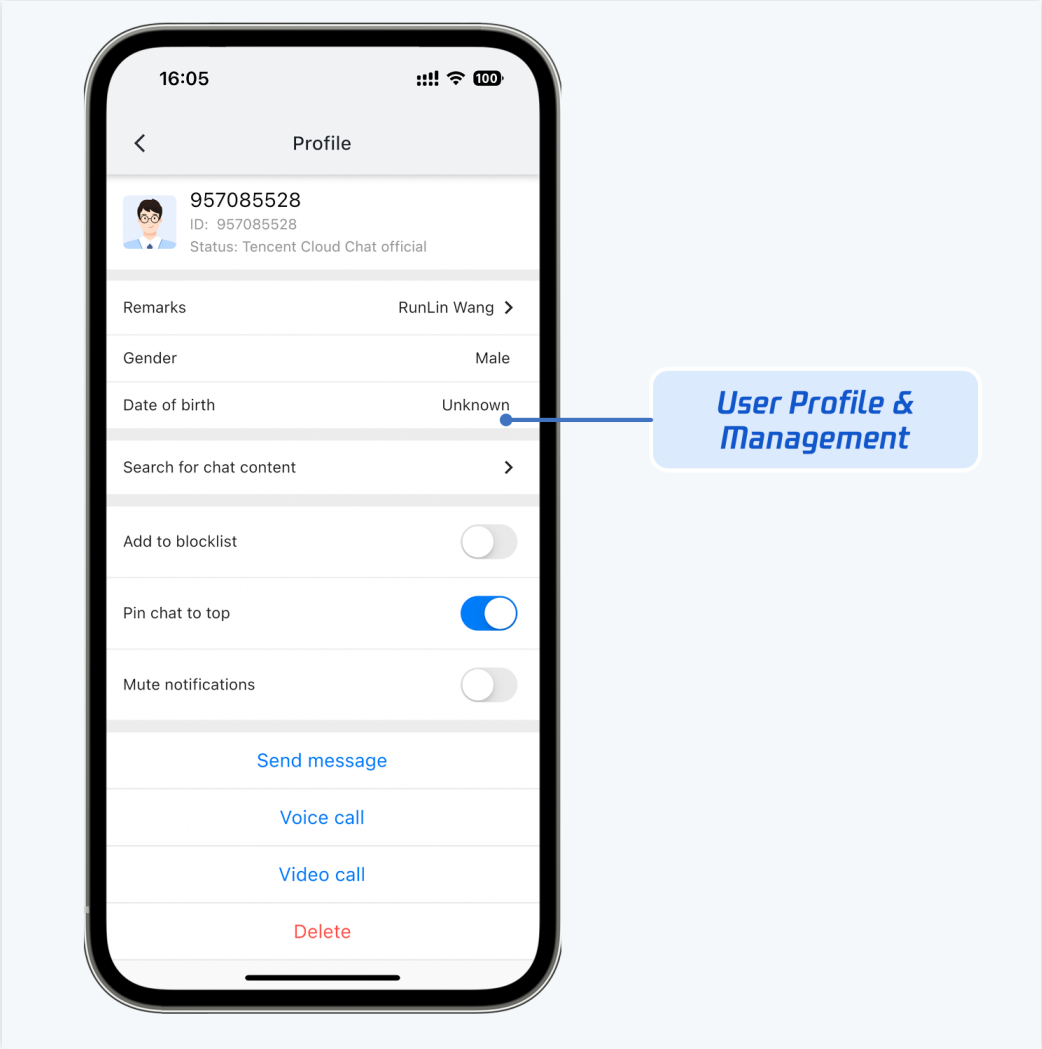
用户资料与管理组件

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

移动端

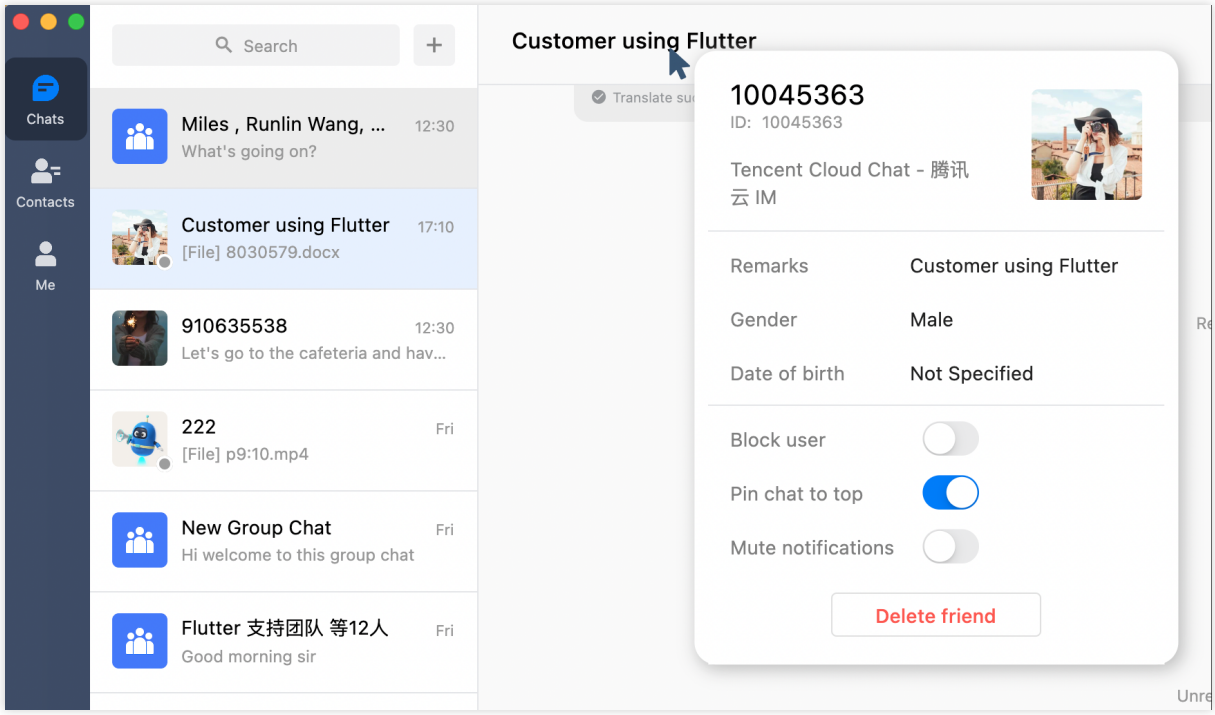
桌面端



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。

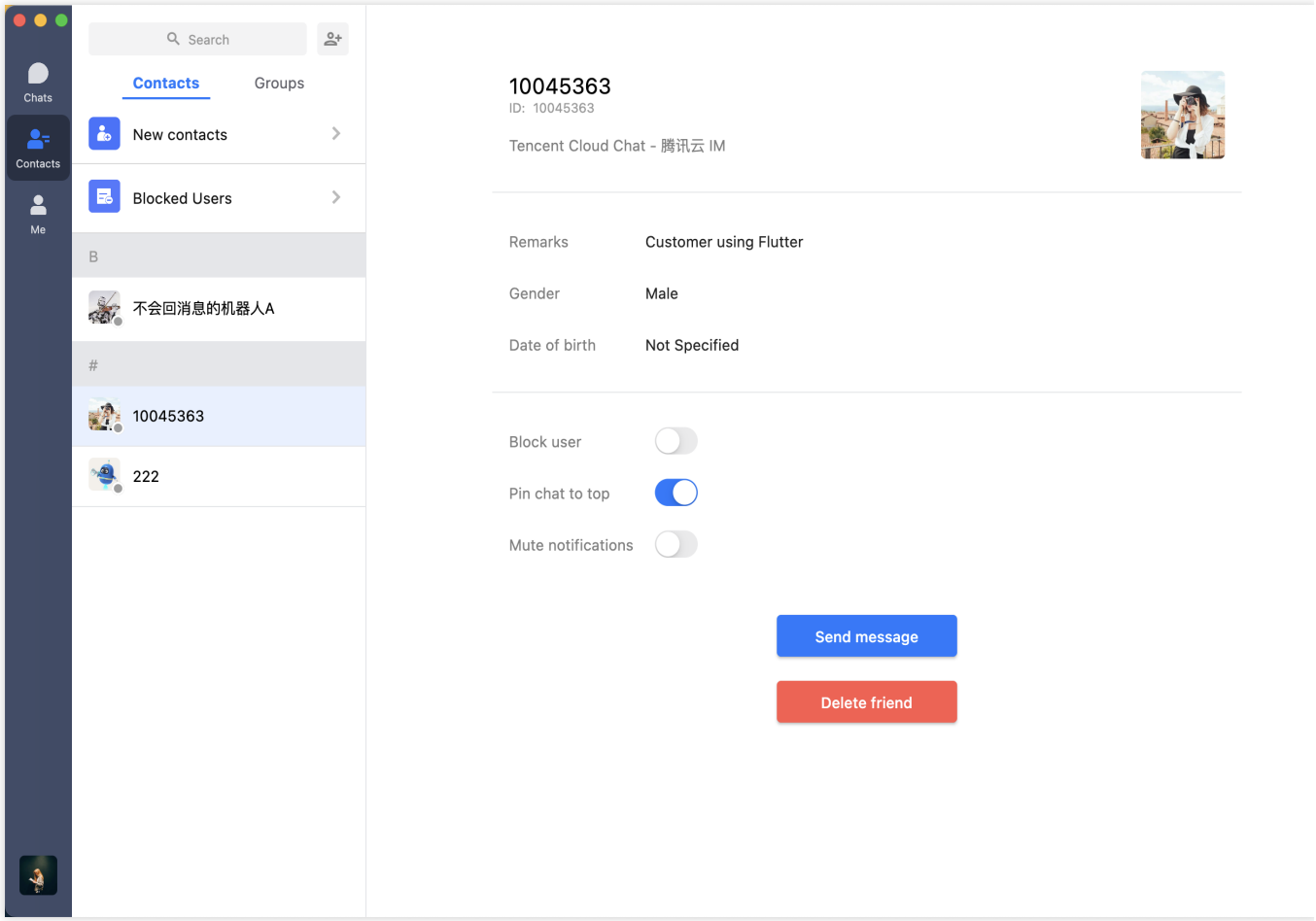
小卡片展示。用于 单聊标题点击 / 群聊其他成员头像点击 等场景。

©2013-2025 Tencent Cloud International Pte. Ltd.



联系人详情页。

©2013-2025 Tencent Cloud International Pte. Ltd.



添加好友与群组系列组件

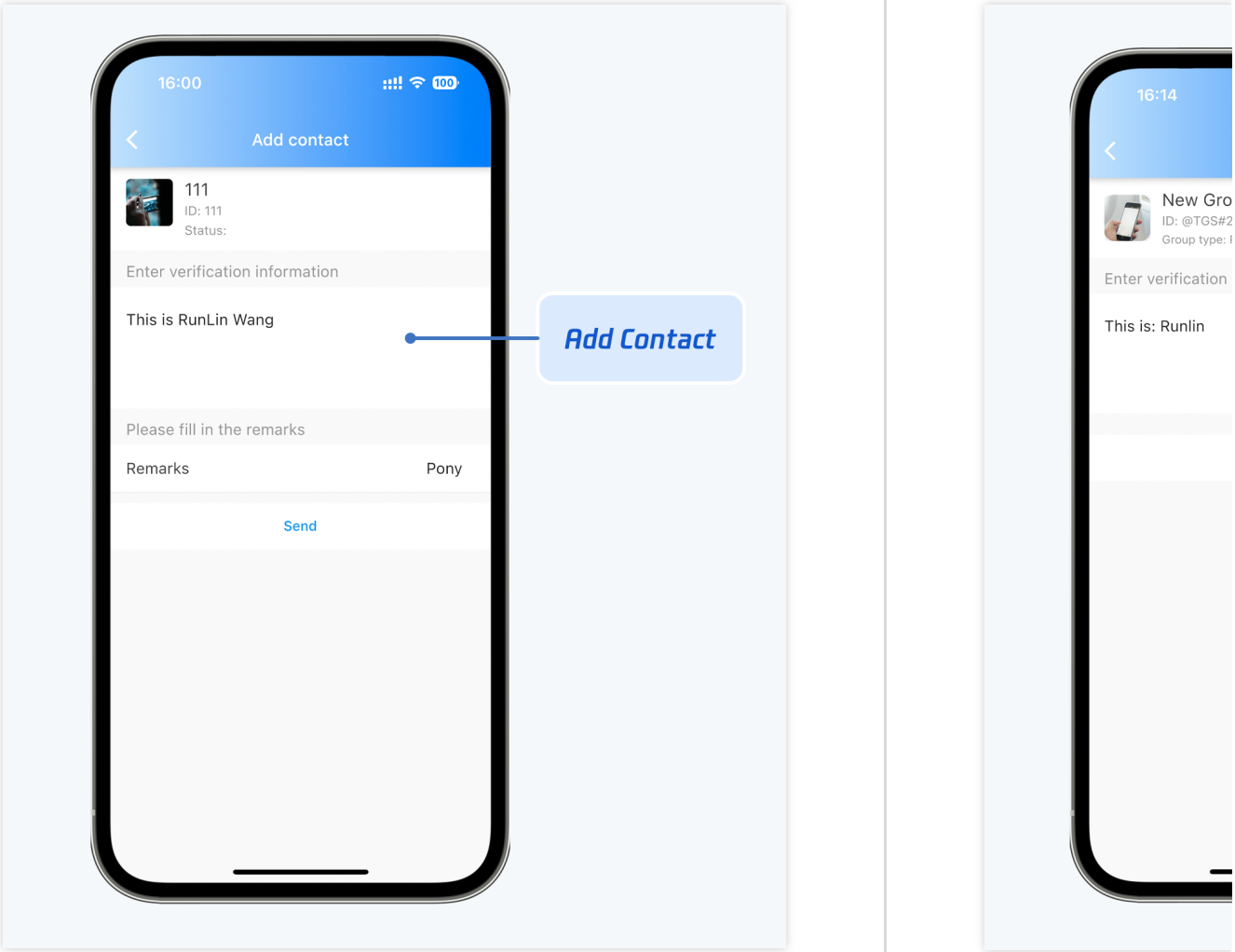
TIMUIKitAddFriend 为添加好友页面。**TIMUIKitAddGroup** 为添加群组页面。界面效果如下图所示：

移动端

桌面端

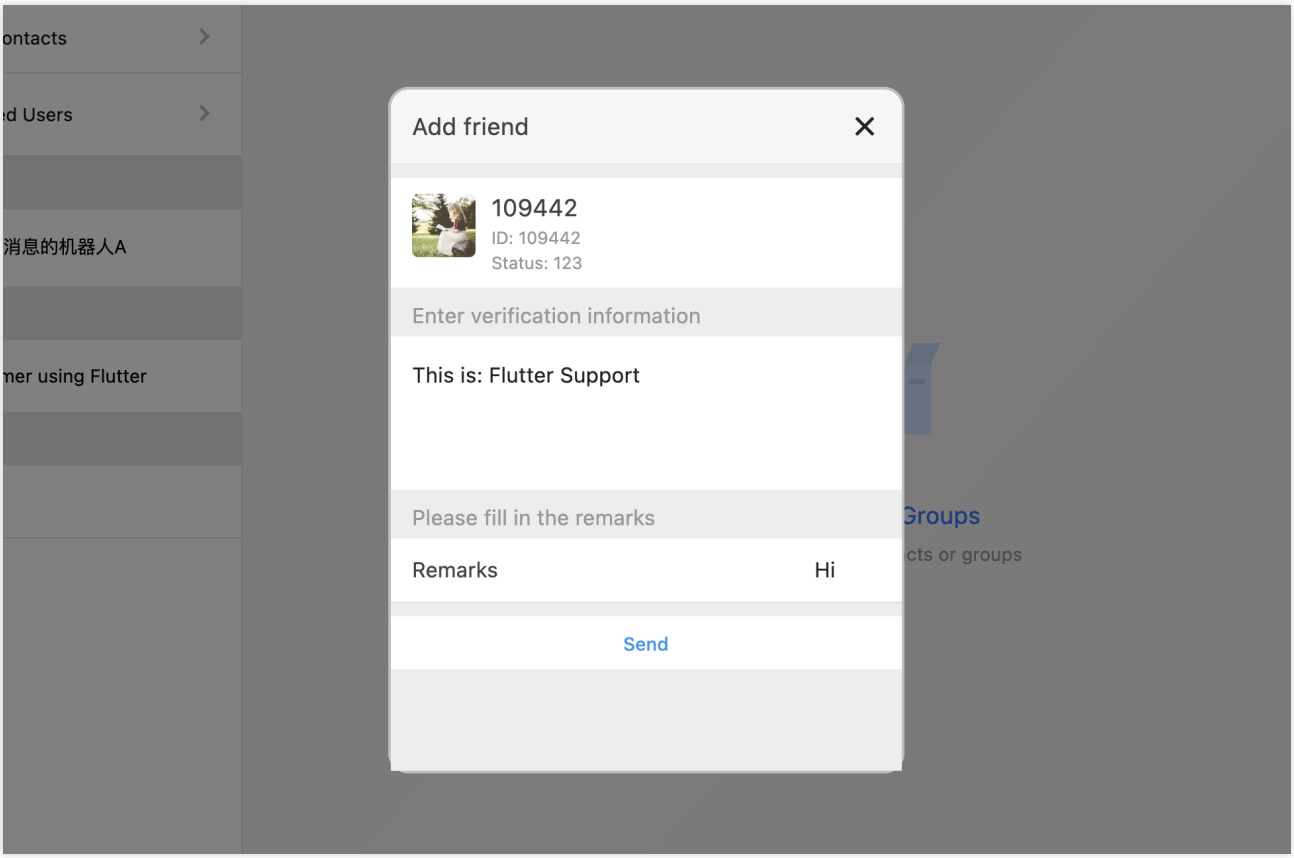
添加好友页面(TIMUIKitAddFriend)	添加群组页面(TIMUIKitAddG

©2013-2025 Tencent Cloud International Pte. Ltd.



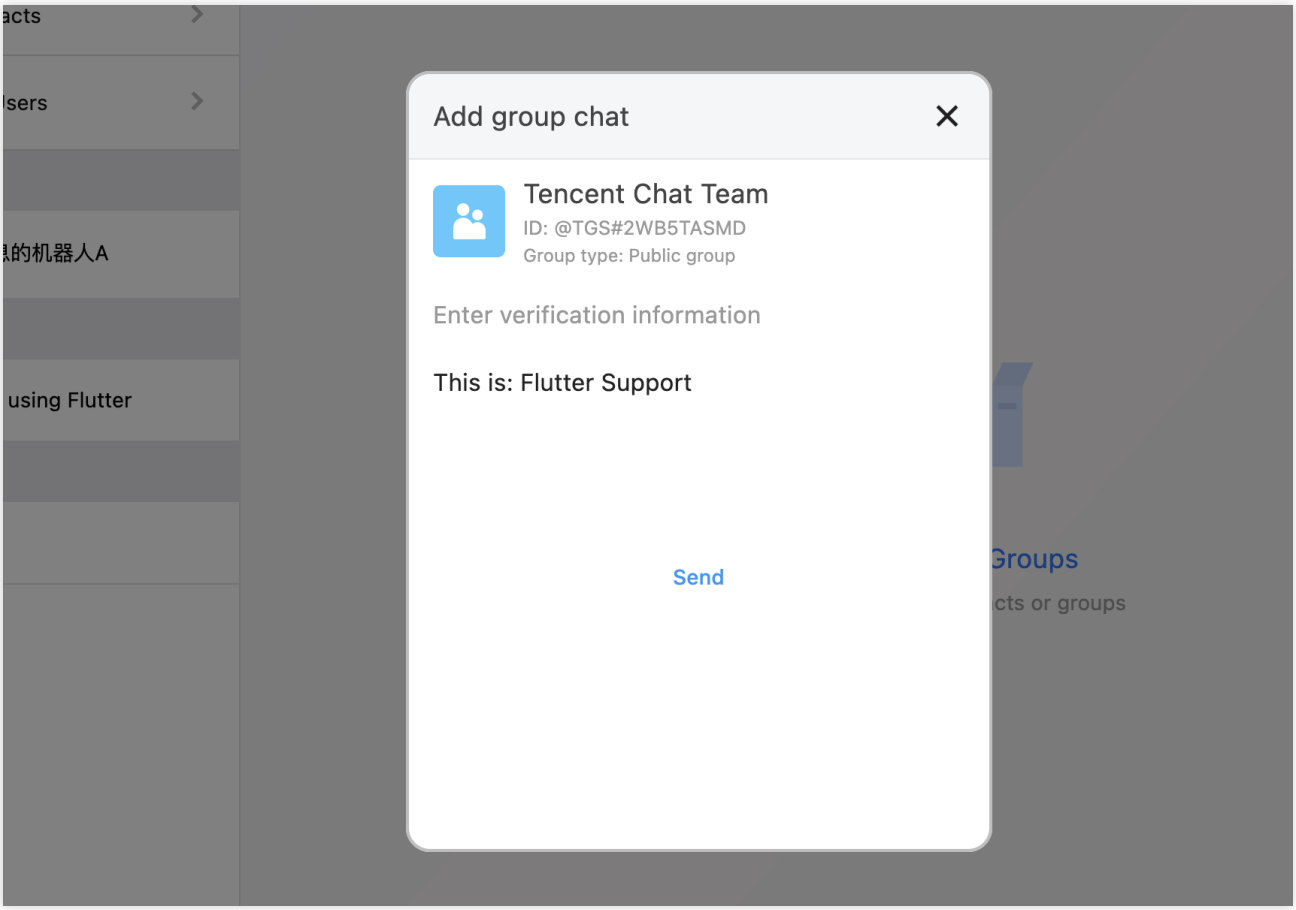
添加好友 - TIMUIKitAddFriend

©2013-2025 Tencent Cloud International Pte. Ltd.



添加群组 - TIMUIKitAddGroup

©2013-2025 Tencent Cloud International Pte. Ltd.



群资料与管理组件

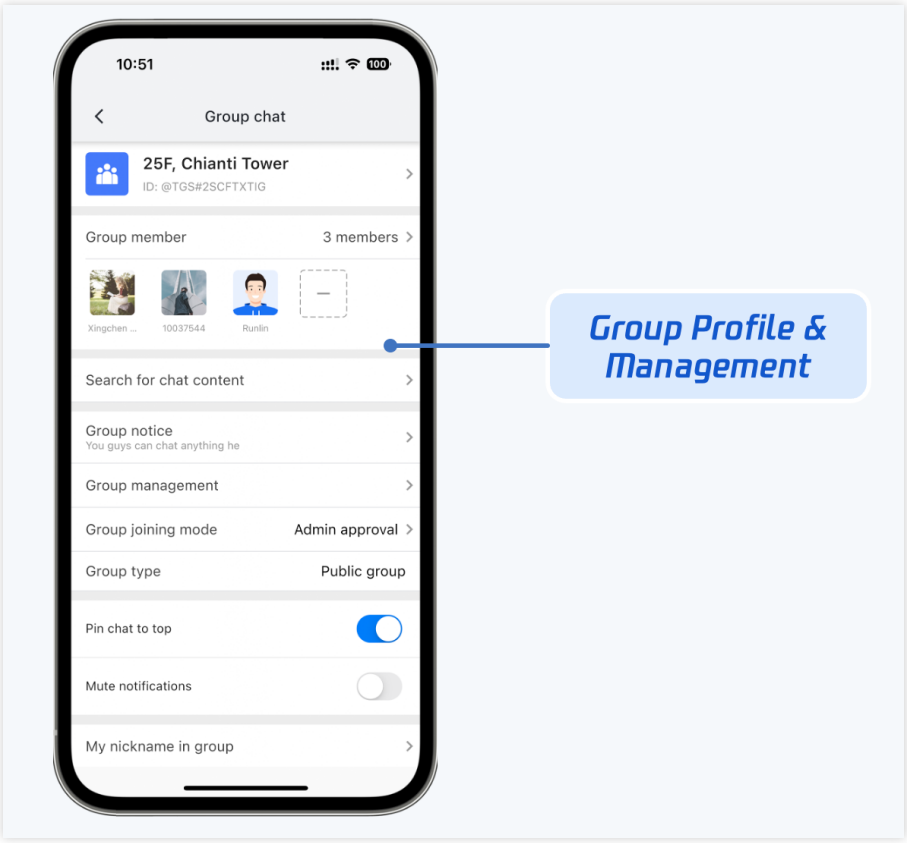
TIMUIKitGroupProfile 主要负责群资料、群成员、群组权限的展示与管理。界面效果如下图所示：

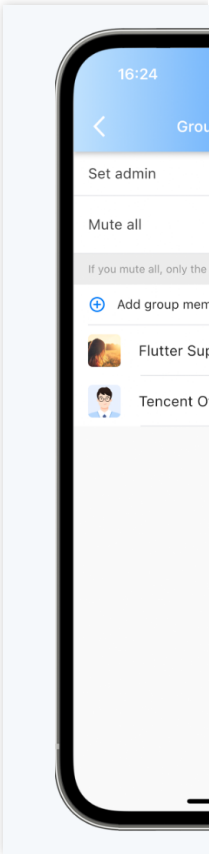
移动端

桌面端

群资料及管理	群成员管理

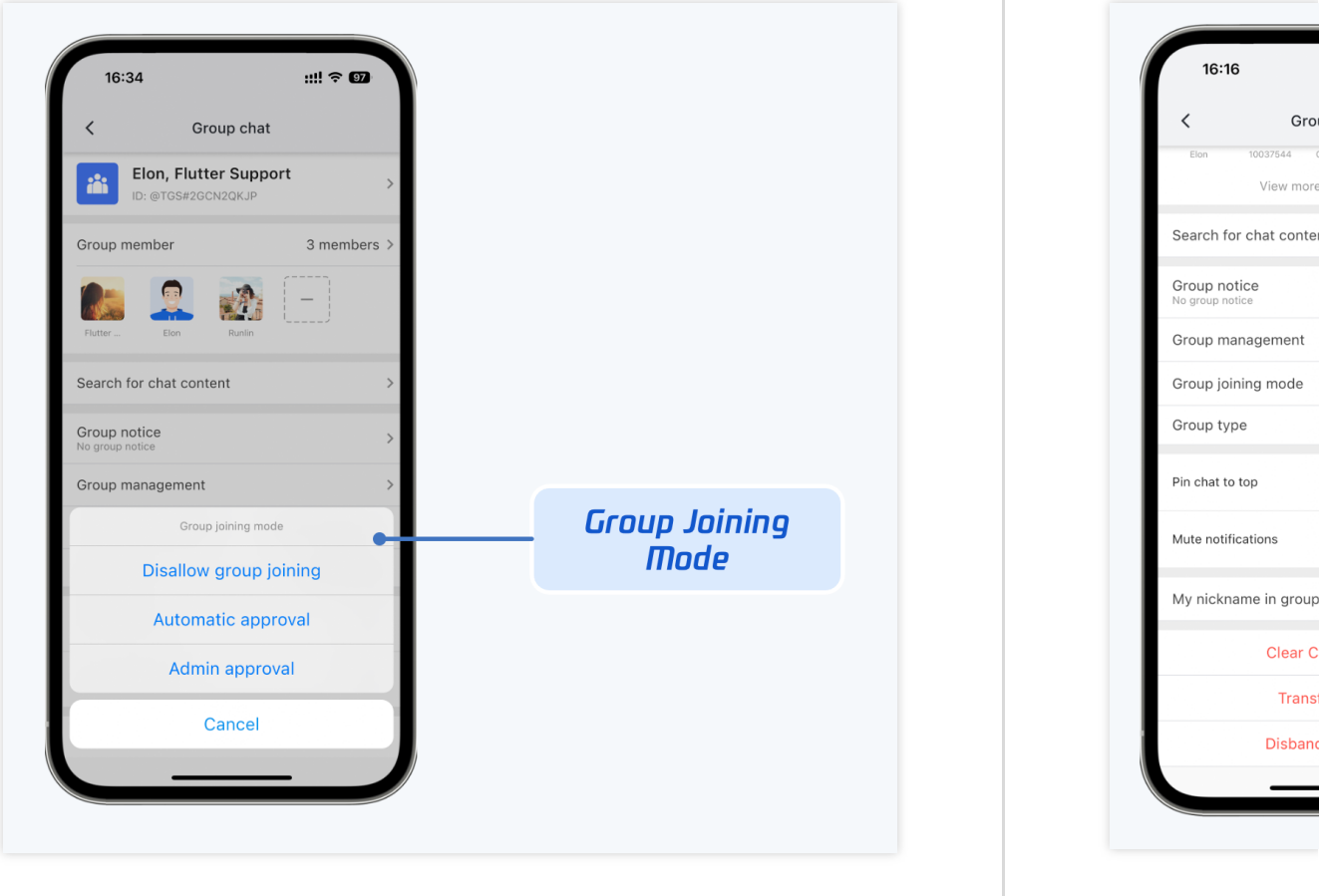
©2013-2025 Tencent Cloud International Pte. Ltd.





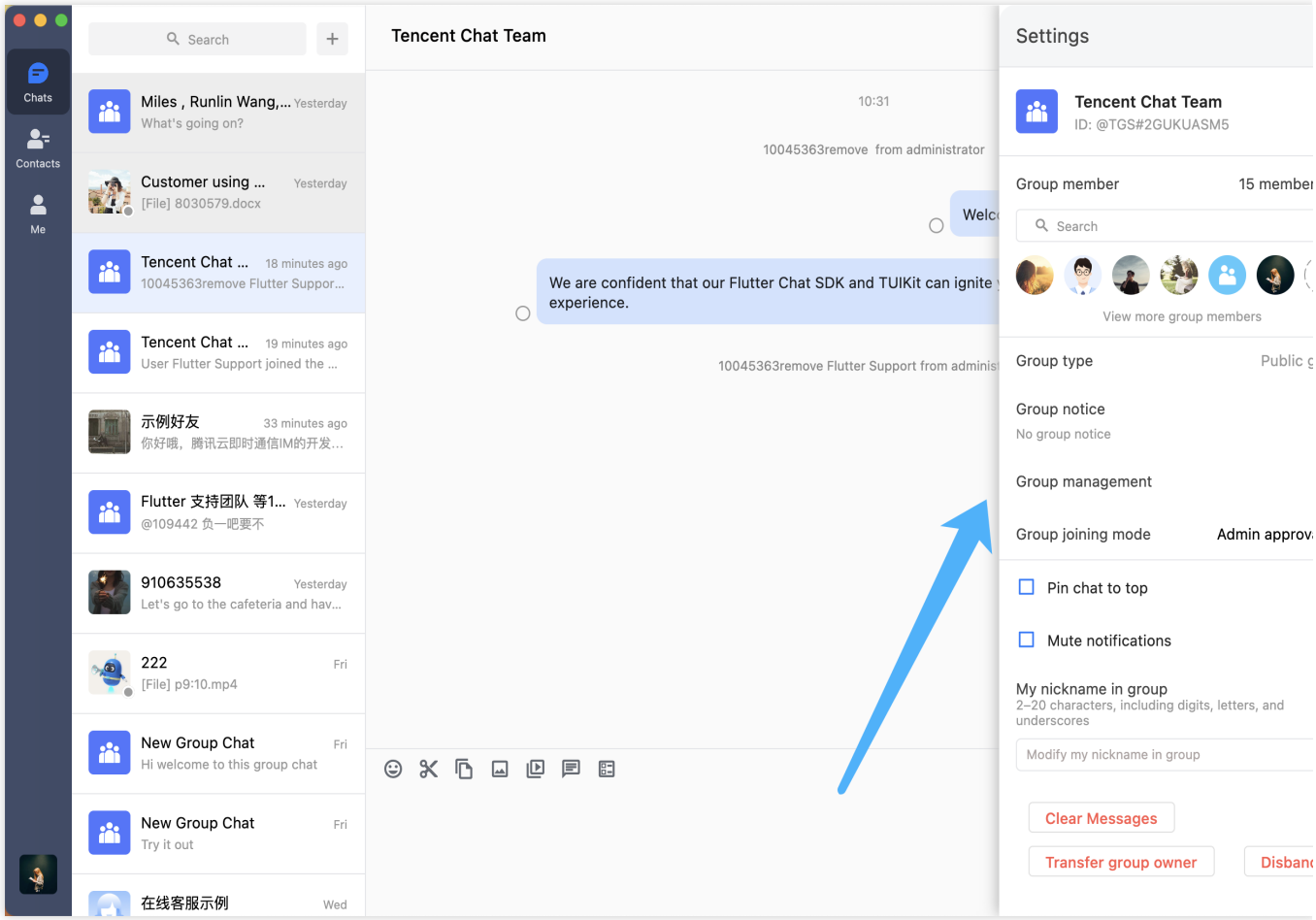
加群方式管理	群操作

©2013-2025 Tencent Cloud International Pte. Ltd.



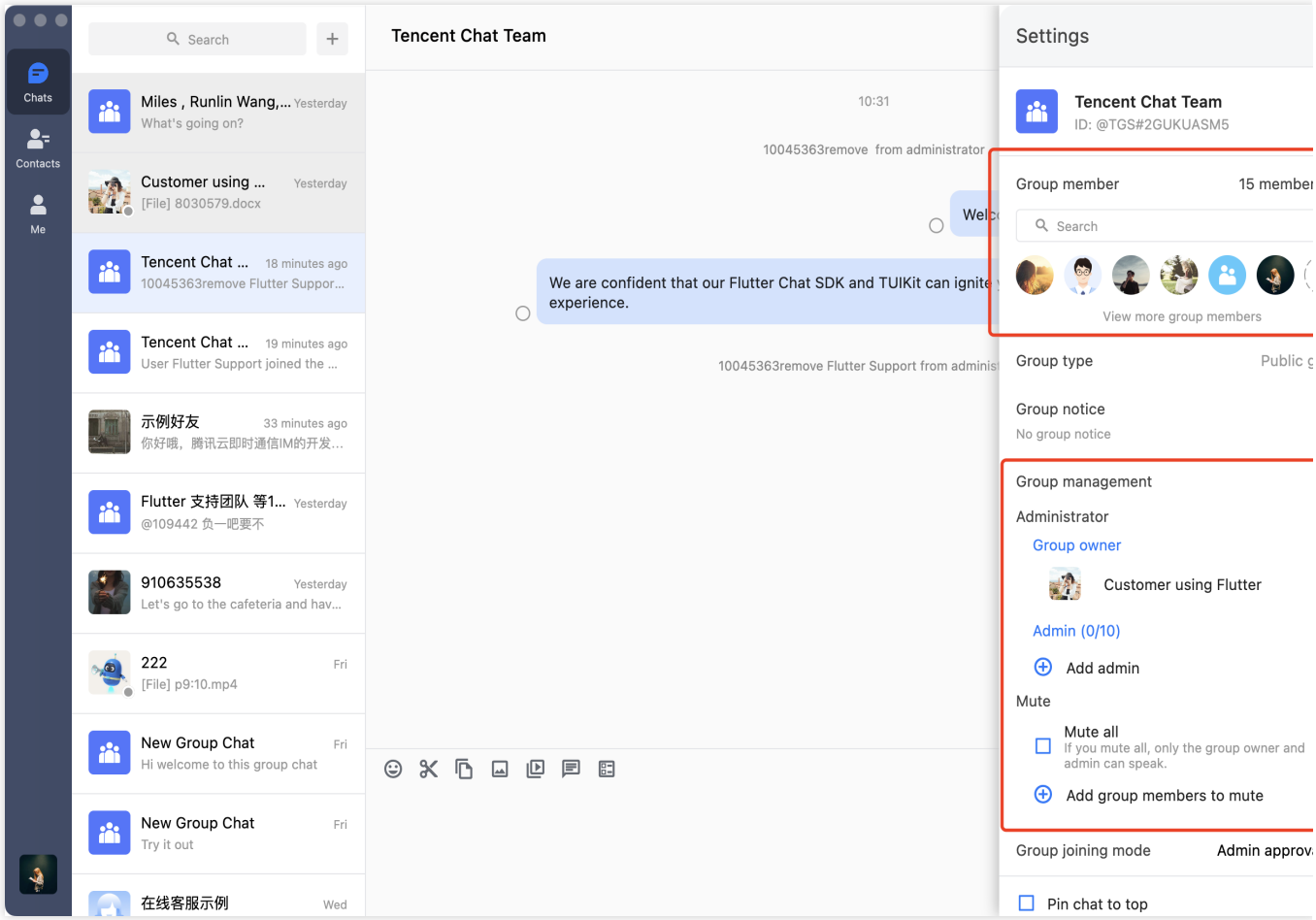
群资料及管理。可展示在群聊右侧，和移动端功能保持一致，只是界面 UI 不同。

©2013-2025 Tencent Cloud International Pte. Ltd.



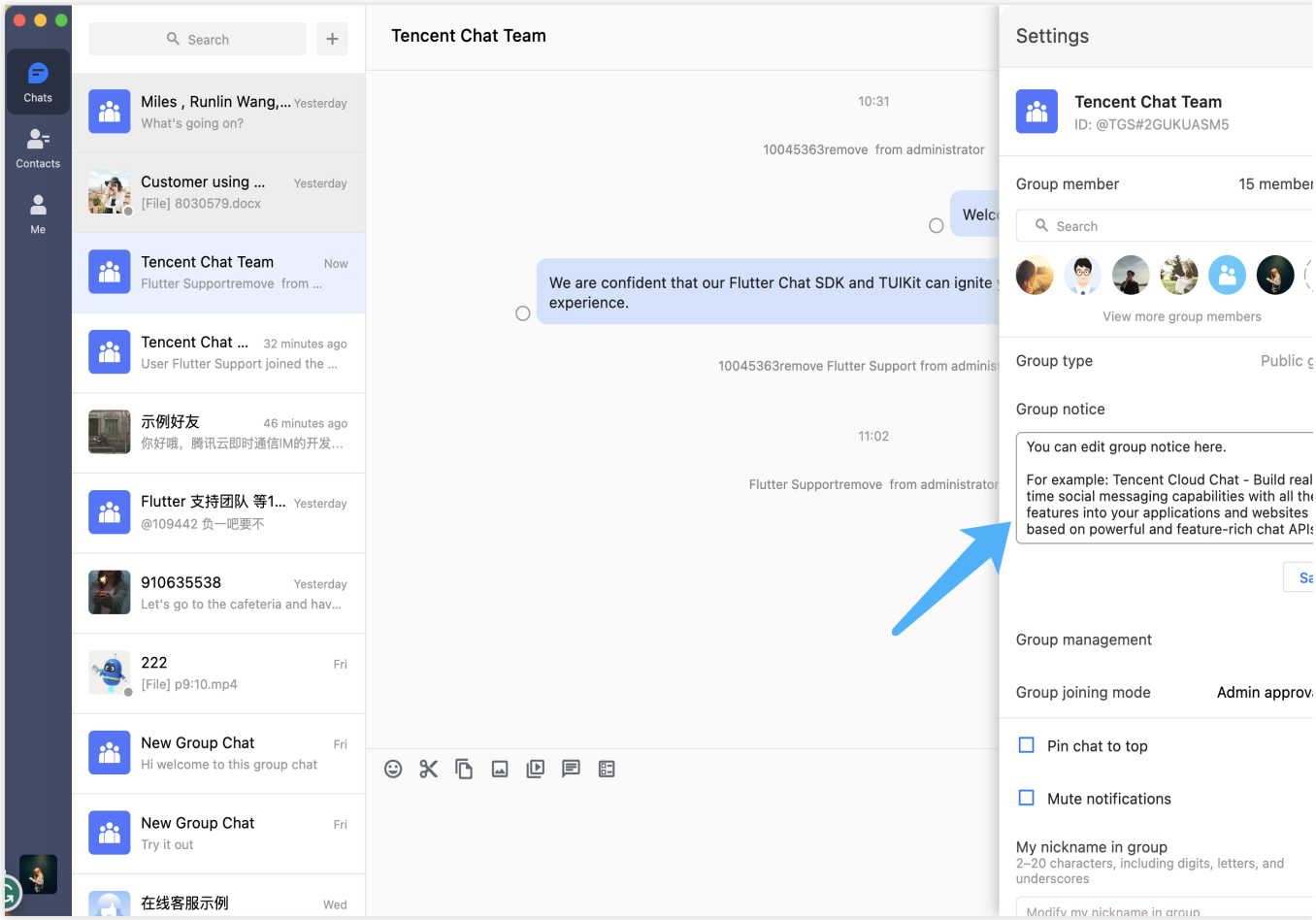
群成员管理。可在群成员卡片中查看全部群成员，并操作群成员增减。点开群成员管理面板后，可配置 管理员 / 群成员禁言 / 全员禁言 等信息。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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**。

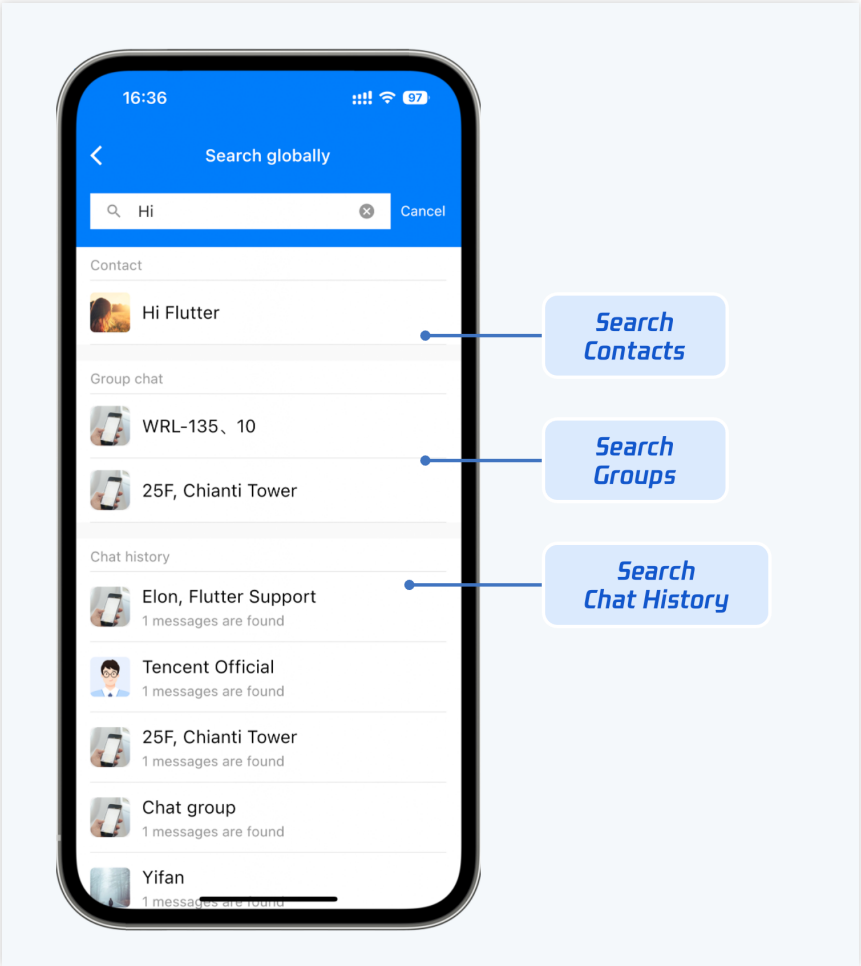
TIMUIKitSearch 为本地全局搜索，支持同时搜索联系人、群聊及聊天记录；**TIMUIKitSearchMsgDetail** 为会话内聊天历史记录搜索。

移动端

桌面端

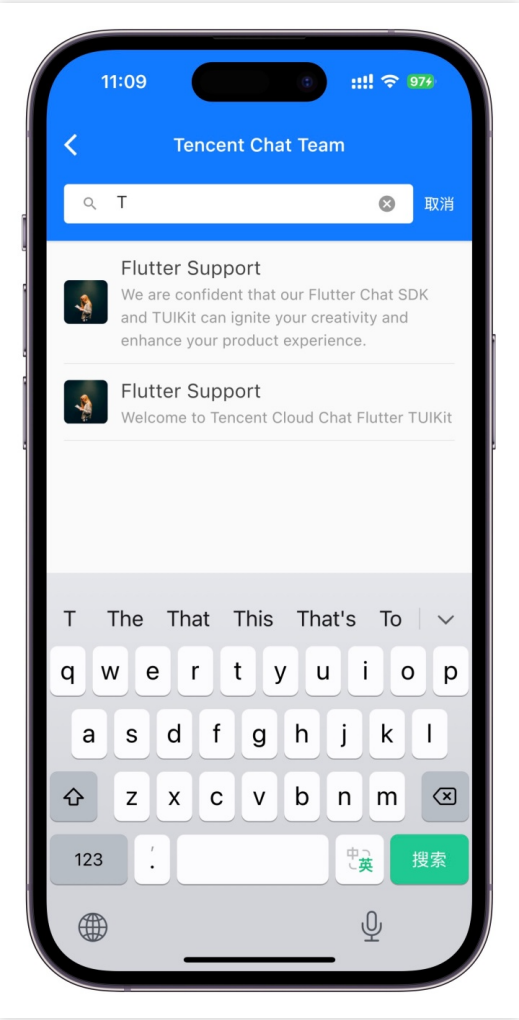
全局搜索 - TIMUIKitSearch

©2013-2025 Tencent Cloud International Pte. Ltd.



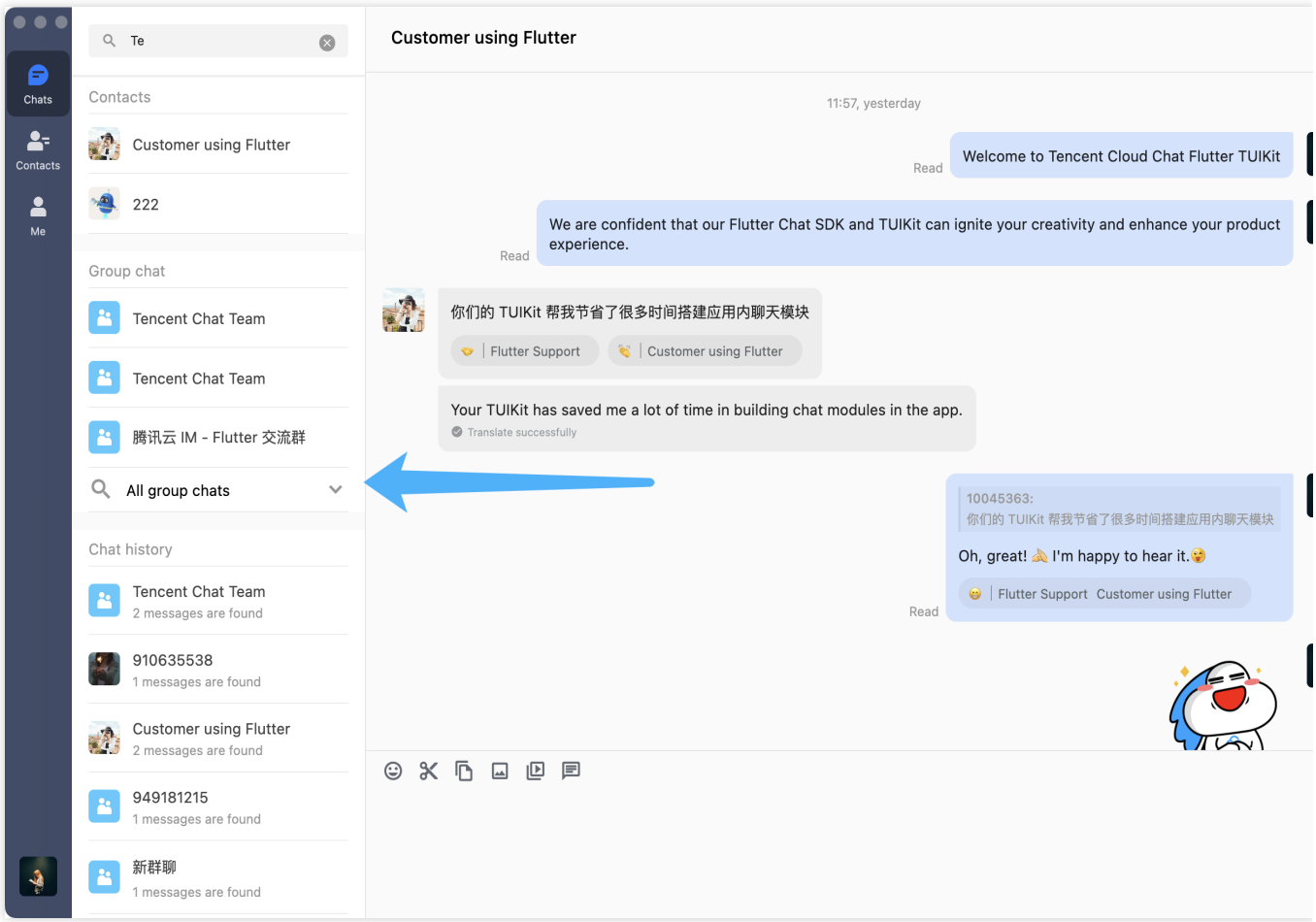
会话内搜索 - TIMUIKitSearchMsgDetail

©2013-2025 Tencent Cloud International Pte. Ltd.



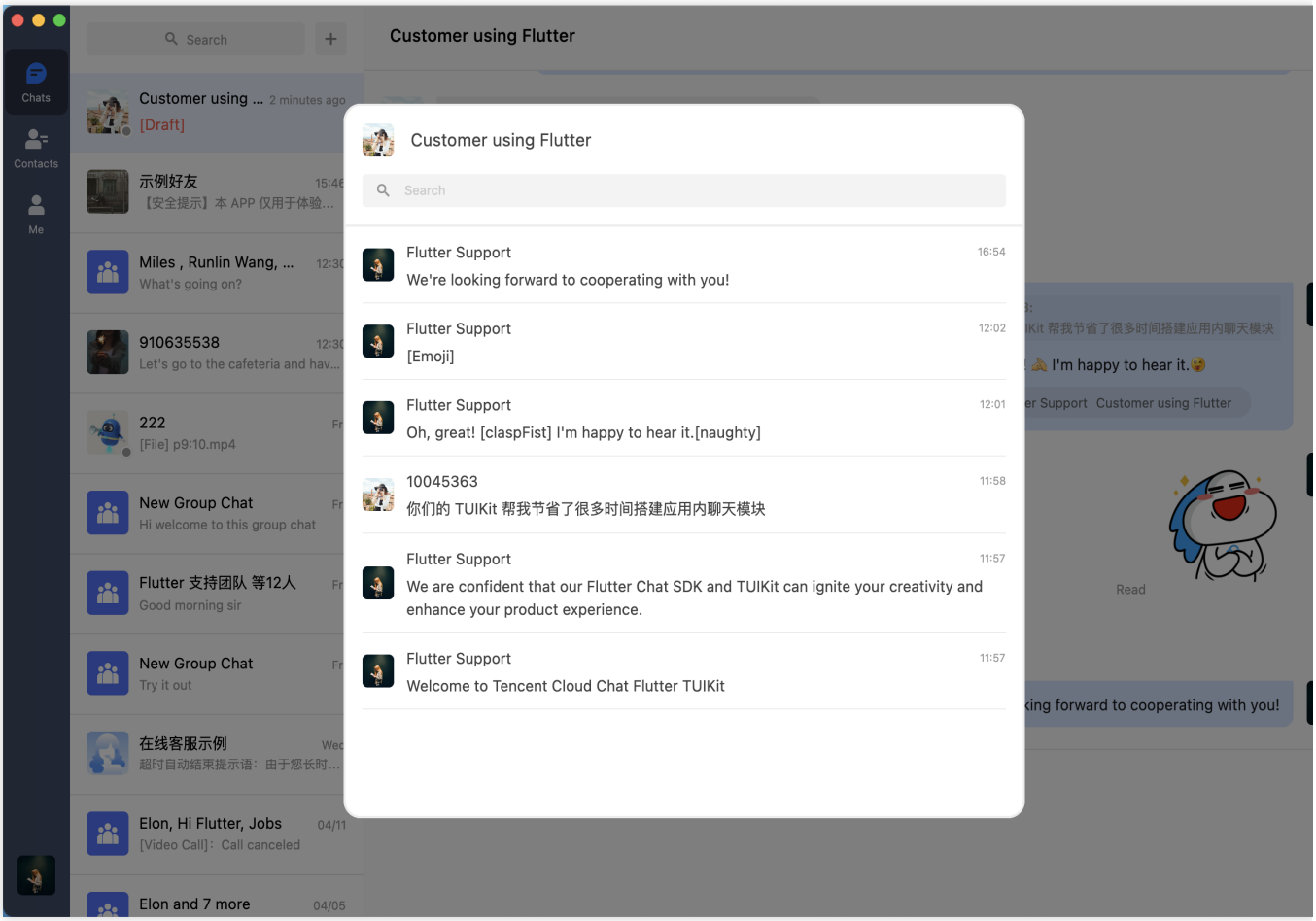
全局搜索 - TIMUIKitSearch

©2013-2025 Tencent Cloud International Pte. Ltd.



会话内搜索 - TIMUIKitSearchMsgDetail

©2013-2025 Tencent Cloud International Pte. Ltd.



音视频通话

TUICallKit插件 用于语音、视频通话。目前仅支持移动端。

©2013-2025 Tencent Cloud International Pte. Ltd.



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：

©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.

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。

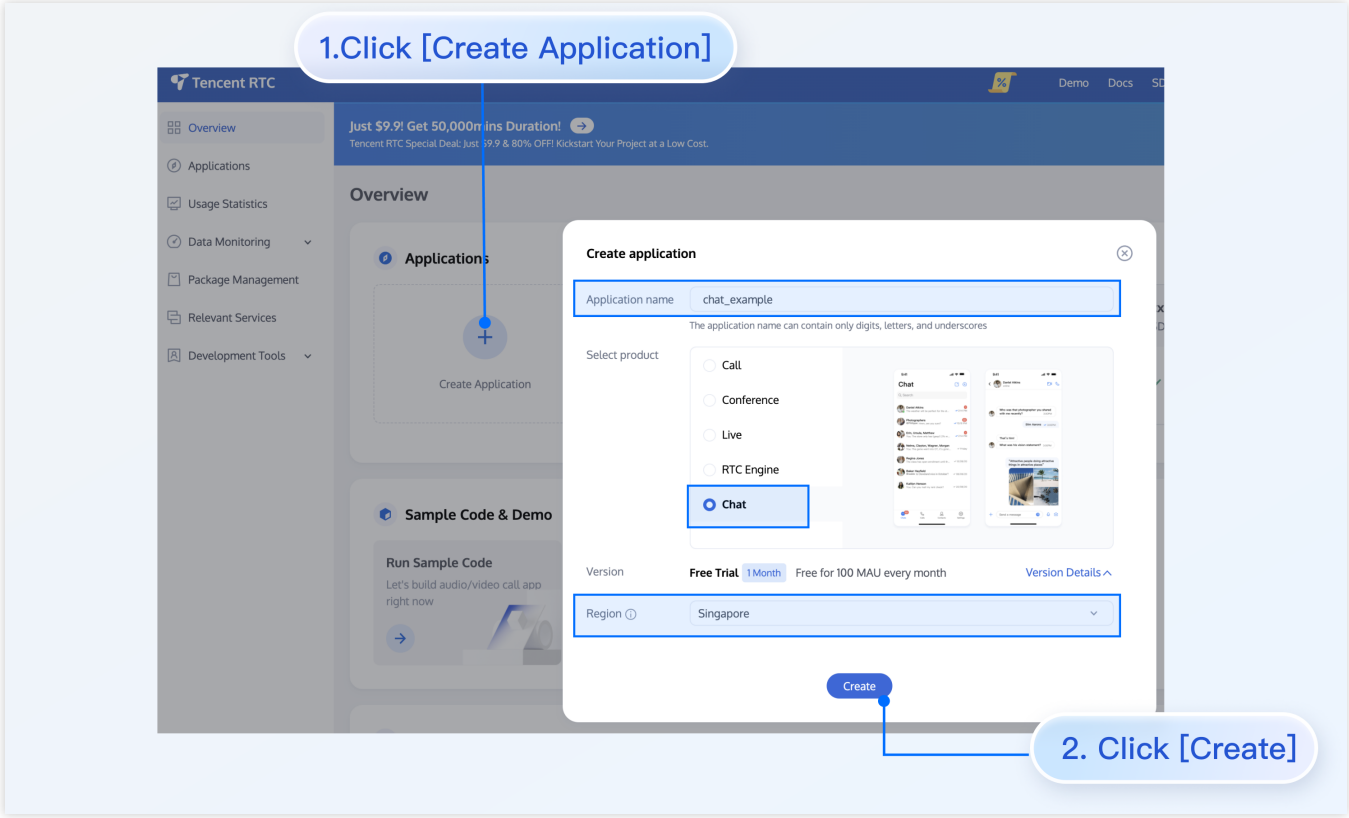
开发环境要求

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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：

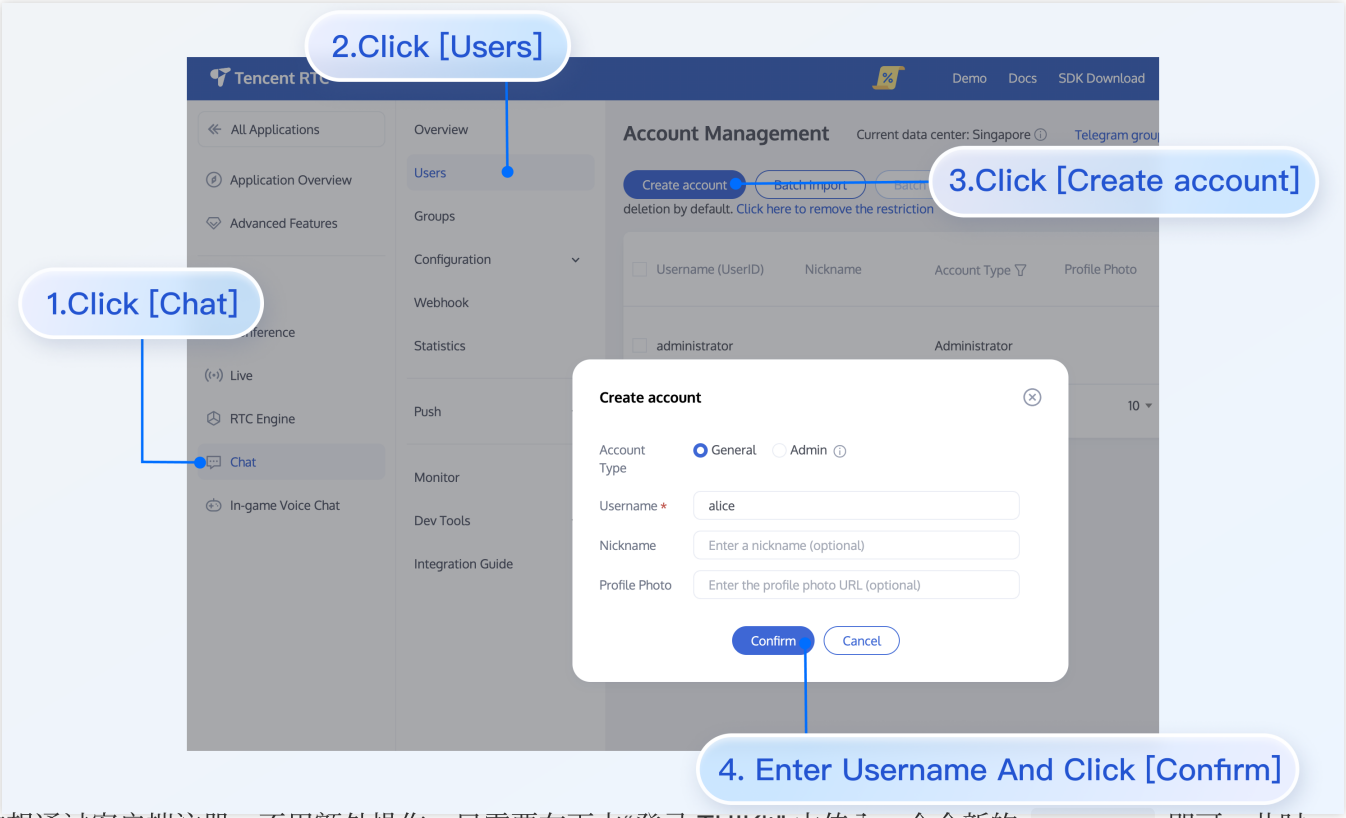
发消息至少是两个用户之间进行，因此您在此环节至少要创建 2 个账号。请记录下这 2 个账号的 userID，后续步骤会使用到。

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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：

```
// 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() {
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

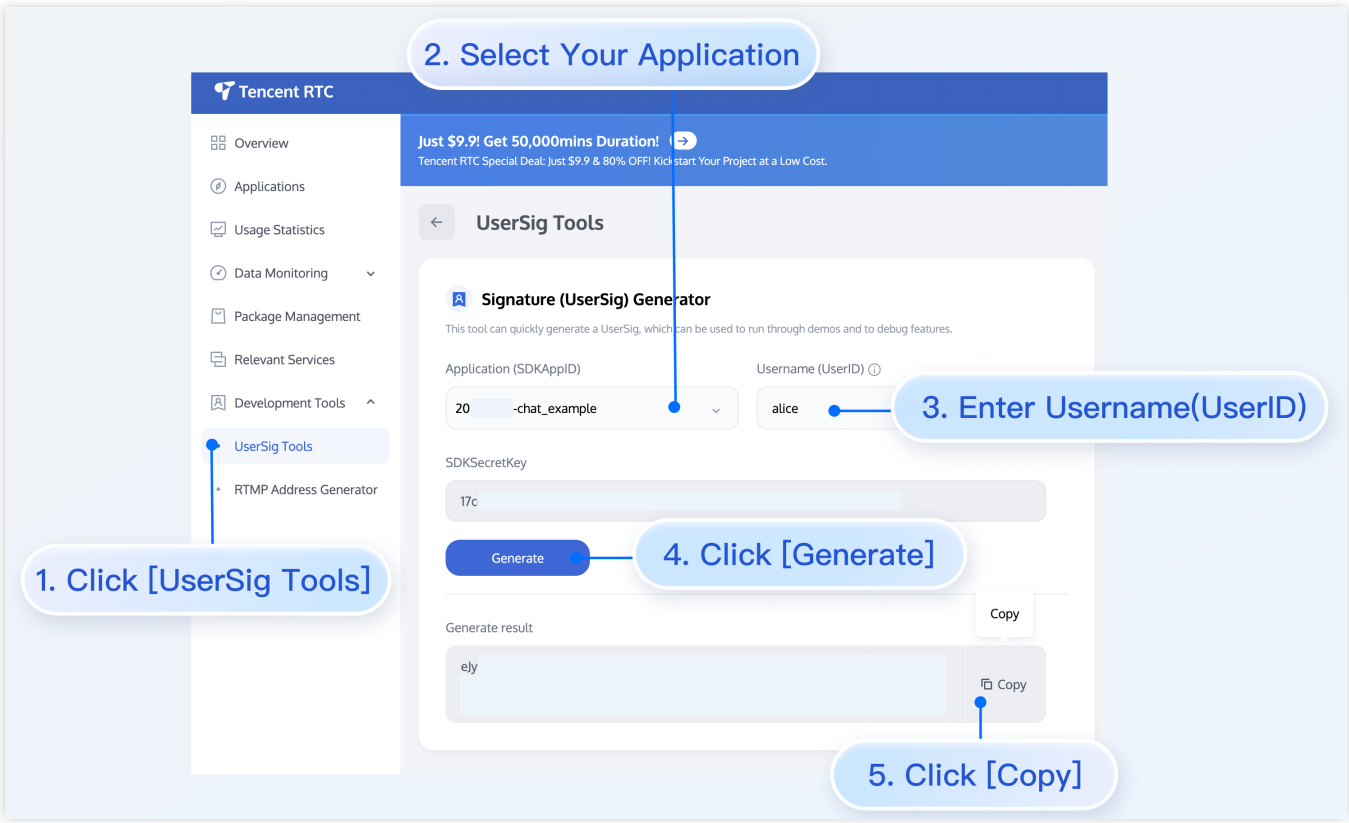
```
    @Override
    public void onError(final int code, final String desc) {
    }
});
```

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。

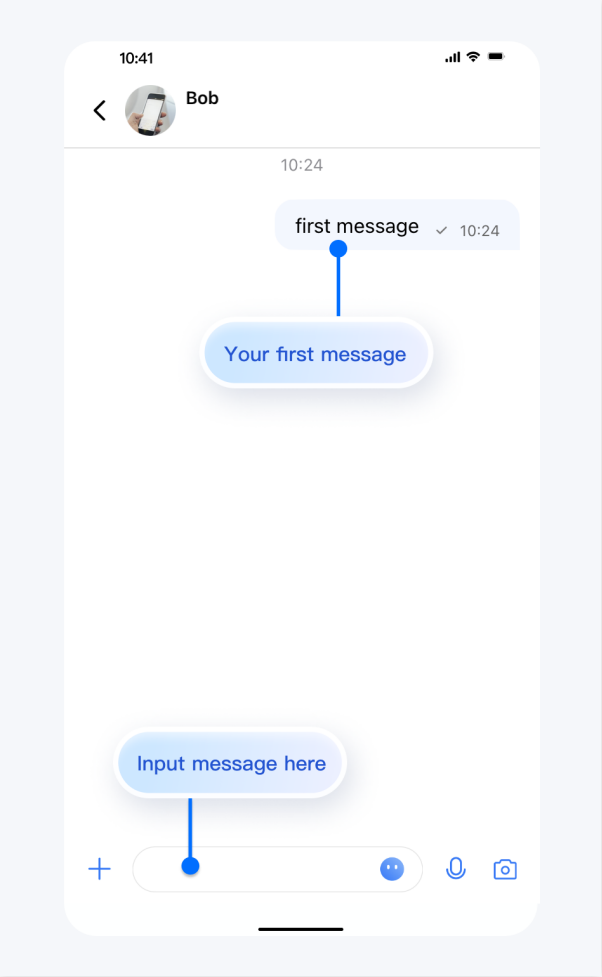
示例代码如下，其中 chatID 需要传入聊天对象 user2 的 userID。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:17

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

©2013-2025 Tencent Cloud International Pte. Ltd.

`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：

```
// 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
// Pass userID for 1v1 conversation.
func pushToChatViewController(groupID: String?, userID: String?) {
    // Create conversationData.
    let conversationData = TUIChatConversationModel()
    conversationData.userID = userID

    // Create c2c chatVC.
    let chatVC = TUIC2CChatViewController_Minimalist()
    chatVC.conversationData = conversationData

    // Option 1: navigate to chatVC.
    navigationController?.pushViewController(chatVC, animated: true)
    // Option 2: add chatVC as a childVC to your parent VC.
    // addChild(chatVC)
    // view.addSubview(chatVC.view)
}
```

```
// 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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-06-23 10:47:59

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、文本翻译、语音转文字等。

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。
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：3.24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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

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_common。

```
flutter pub add tencent_cloud_chat_common
```

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_sticker
flutter pub add tencent_cloud_chat_message_reaction
flutter pub add tencent_cloud_chat_text_translate
```

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

```
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,
          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。

```
pages = [
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      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。

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` 。
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

集成 TUIKit
Android

最近更新时间：2025-04-03 11:45:28

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// 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 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-kt'
project(':tuicallkit-kt').projectDir = new File(settingsDir, '../TUIKit/TUICallKit/

// 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 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')
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  api project(':tuisearch')
  api project(':tuicommunity')
  api project(':tuicallkit-kt')
  api project(':tuiroomkit')
  // Integrate speech-to-text plugin, supported from version 7.5
  api project(':tuivoicetotextplugin')
  // 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:8.4.6667'
  // Integrate group voting plugin, supported from version 7.1
  api 'com.tencent.imsdk:tuipoll-plugin:8.4.6667'
  // Integrate session grouping plugin, supported from version 7.3
  api 'com.tencent.imsdk:tuiconversationgroup-plugin:8.4.6667'
  // Integrate session tagging plugin, supported from version 7.3
  api 'com.tencent.imsdk:tuiconversationmark-plugin:8.4.6667'
  // Integrate message push plugin, supported from version 7.6
  api 'com.tencent.timpush:timpush:8.4.6667'
  // Integrate the corresponding manufacturer's push package as needed
  api 'com.tencent.timpush:fcm:8.4.6667'
  api 'com.tencent.timpush:xiaomi:8.4.6667'
  api 'com.tencent.timpush:meizu:8.4.6667'
  api 'com.tencent.timpush:oppo:8.4.6667'
  api 'com.tencent.timpush:vivo:8.4.6667'
  api 'com.tencent.timpush:huawei:8.4.6667'
  api 'com.tencent.timpush:honor:8.4.6667'
}
```

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"
  }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
```

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。

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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？

Chat 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 Chat 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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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。

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

注意：

从 8.5.6870 版本开始，TUIKit 支持 Swift 语言组件。

您可以在 Podfile 中按需添加组件库：

简约版

经典版

Swift

Objective-C

```
# 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
  use_frameworks!
  use_modular_headers!

  # Integrate the chat feature
  pod 'TUIChat_Swift/UI_Minimalist'
  # Integrate the conversation list feature
  pod 'TUIConversation_Swift/UI_Minimalist'
  # Integrate the relationship chain feature
  pod 'TUIContact_Swift/UI_Minimalist'
  # Integrate the search feature (To use this feature, you need to purchase the
Pro edition 、Pro Plus edition or Enterprise edition)
  pod 'TUISearch_Swift/UI_Minimalist'
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'
  # Integrate Translation Plugin(Value-added feature activation required,
please contact Tencent Cloud Business to activate)
  pod 'TUITranslationPlugin_Swift'
  # Integrate Session Tagging Plugin
  pod 'TUIConversationMarkPlugin_Swift'
  # Integrate Speech-to-Text Plugin
  pod 'TUIVoiceToTextPlugin_Swift'
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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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
mask the configuration.
  # use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers,
the Pod module can be imported using @import.
  # 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 search feature (To use this feature, you need to purchase the
Pro edition 、Pro Plus edition or Enterprise edition)
  pod 'TUISearch/UI_Minimalist'
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'
  # Integrate Translation Plugin, supported starting from version 7.2 (Value-
added feature activation required, please contact Tencent Cloud Business to
activate)
  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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Swift

Objective-C

```
# 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
  use_frameworks!
  use_modular_headers!

  # Integrate the chat feature
  pod 'TUIChat_Swift/UI_Classic'
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  # Integrate the conversation list feature
  pod 'TUIConversation_Swift/UI_Classic'
  # Integrate the relationship chain feature
  pod 'TUIContact_Swift/UI_Classic'
  # Integrate the search feature (To use this feature, you need to purchase the
Pro edition 、Pro Plus edition or Enterprise edition)
  pod 'TUISearch_Swift/UI_Classic'
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'
  # Integrate Voting Plugin
  pod 'TUIPollPlugin_Swift'
  # Integrate Group Chain Plugin
  pod 'TUIGroupNotePlugin_Swift'
  # Integrate Translation Plugin (Value-added feature activation required,
please contact Tencent Cloud Business to activate)
  pod 'TUITranslationPlugin_Swift'
  # Integrate Session Grouping Plugin
  pod 'TUIConversationGroupPlugin_Swift'
  # Integrate Session Tagging Plugin
  pod 'TUIConversationMarkPlugin_Swift'
  # Integrate Speech-to-Text Plugin
  pod 'TUIVoiceToTextPlugin_Swift'
  # Integrate Customer Service Plugin
  pod 'TUICustomerServicePlugin_Swift'

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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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
mask the configuration.
  # use_frameworks!

  # Enable modular headers as needed. Only after you enable modular headers,
the Pod module can be imported using @import.
  # 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 search feature (To use this feature, you need to purchase the
Pro edition 、Pro Plus edition or Enterprise edition)
  pod 'TUISearch/UI_Classic'
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'
  # Integrate Voting Plugin, supported starting from version 7.1
  pod 'TUIPollPlugin'
  # Integrate Group Chain Plugin, supported starting from version 7.1
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  pod 'TUIGroupNotePlugin'
  # Integrate Translation Plugin, supported starting from version 7.2 (Value-
added feature activation required, please contact Tencent Cloud Business to
activate)
  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 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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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` 组件搭配使用。同理，简约版 `TUIChat` 组件必须与简约版 `TUIConversation`、`TUIContact` 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。

```
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，将 TUIKit 目录直接拖入您的工程目录下:
Swift TUIKit 源码 Github 地址：

Objective-C TUIKit 源码 Github 地址：

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"`

©2013-2025 Tencent Cloud International Pte. Ltd.

以 TUIKit 文件夹位于您工程 Podfile 文件父目录为例：

DevelopmentPodfile

Swift

Objective-C

```
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
frameworks
  use_frameworks!
  use_modular_headers!

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

  # Integrate TUIKit components (optional)
  # Integrate the chat feature
  pod 'TUIChat_Swift', :path => "../TUIKit/TUIChat"
  # Integrate the conversation list feature
  pod 'TUIConversation_Swift', :path => "../TUIKit/TUIConversation"
  # Integrate the relationship chain feature
  pod 'TUIContact_Swift', :path => "../TUIKit/TUIContact"
  # Integrate the search feature (To use this feature, you need to purchase the
Ultimate edition)
  pod 'TUISearch_Swift', :path => "../TUIKit/TUISearch"
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'

  # Integrate the TUIKitPlugin plugin (optional)
  # Integrate translation plugin (Value-added feature activation is required.
Please contact Tencent Cloud sales)
  pod 'TUITranslationPlugin_Swift'

  # Other Pods
  pod 'MJRefresh'
  pod 'SnapKit'
end
#Pods config
post_install do |installer|
    installer.pods_project.targets.each do |target|
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

```
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
frameworks
  use_frameworks!
  use_modular_headers!

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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  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 search feature (To use this feature, you need to purchase the
Ultimate edition)
  pod 'TUISearch', :path => "../TUIKit/TUISearch"
  # Integrate the audio/video call feature
  pod 'TUICallKit_Swift'
  # 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/TUICore.spec".

  # Integrate the voting plugin, supported from version 7.1
  pod 'TUIPollPlugin'
  # Integrate the group chain plugin, supported from version 7.1
  pod 'TUIGroupNotePlugin'
  # Integrate translation plugin, supported from version 7.2 (Value-added
feature activation is required. Please contact Tencent Cloud sales)
  pod 'TUITranslationPlugin'
  # Integrate the session grouping plugin, supported from version 7.3
  pod 'TUIConversationGroupPlugin'
  # Integrate the session tagging plugin, supported from version 7.3
  pod 'TUIConversationMarkPlugin'
  # 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'] = ""
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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。

Swift 组件

Objective-C 组件

©2013-2025 Tencent Cloud International Pte. Ltd.

```
- MJExtension (3.4.1)
- MJRefresh (3.7.5)
- SnapKit (5.7.1)
- SSZipArchive (2.4.3)
- SDWebImage (5.18.11)
```

```
- 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

Xcode 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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

Xcode 16 不支持 Framework 开启 bitcode 问题

解决方案1：升级 SDK

如果您使用的是包含 Bitcode 的旧版本 SDK（如 TXIMSDK_iOS），建议您按照本文档的指引，将 SDK 升级至 TXIMSDK_Plus_iOS_XCFramework。

解决方式2: 修改 Podfile 配置

在您的 Podfile 末尾新增如下配置，重新 Pod install。

```
post_install do |installer|
  bitcode_strip_path = 'xcrun --find bitcode_strip'.chop!
  def strip_bitcode_from_framework(bitcode_strip_path, framework_relative_path)
    framework_path = File.join(Dir.pwd, framework_relative_path)
    command = "#{bitcode_strip_path} #{framework_path} -r -o #{framework_path}"
    puts "Stripping bitcode: #{command}"
    system(command)
  end
  framework_paths = [
    "/Pods/TXIMSDK_iOS/ImSDK.framework/ImSDK",
  ]
  framework_paths.each do |framework_relative_path|
    strip_bitcode_from_framework(bitcode_strip_path, framework_relative_path)
   end
end
```

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 ，

©2013-2025 Tencent Cloud International Pte. Ltd.

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
    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_Swift'
```

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_Swift/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_Swift/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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5（react)

最近更新时间：2025-03-03 11:27:23

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。

```
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 i @tencentcloud/chat-uikit-react @tencentcloud/uikit-base-component-react
```

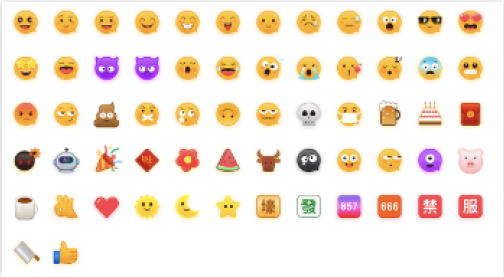
步骤3：引入 chat-uikit-react 组件

注意：

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。



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 React from 'react';
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import { Profile, ConversationList, Chat, ChatHeader, MessageList, MessageInput, Ch

import '@tencentcloud/chat-uikit-react/dist/esm/index.css'

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

TUILogin.login({
  ...config,
  useUploadPlugin: true
}).then(() => {
  openDefaultChat();
}).catch(() => {});

function openDefaultChat() {
  // 1v1 chat: conversationID = `C2C${userID}`
  // group chat: conversationID = `GROUP${groupID}`
  const userID = 'administrator'; // userID: Recipient of the Message userID, Get i
  const conversationID = `C2C${userID}`;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  TUIConversationService.switchConversation(conversationID);
}

export default function App() {
  // language support en-US(default) / zh-CN / ja-JP / ko-KR / zh-TW
  // theme support light(default) / dark
  return (
    <div style={{display: 'flex', height: '100vh'}}>
      <UIKitProvider language='en-US' theme='light'>
        <div style={{ display: 'flex', flexDirection: 'column', maxWidth: '30%' }}>
          <Profile />
          <ConversationList />
        </div>
        <Chat>
          <ChatHeader />
          <MessageList />
          <MessageInput />
        </Chat>
        <ChatSetting />
      </UIKitProvider>
    </div>
  );
}
```

→ 在沙箱中体验

```
import React from 'react';
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import { Chat, ChatHeader, MessageList, MessageInput, ChatSetting } from '@tencentc

import '@tencentcloud/chat-uikit-react/dist/esm/index.css';

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

TUILogin.login({
  ...config,
  useUploadPlugin: true
}).then(() => {
  openDefaultChat();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}).catch(() => {});

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

export default function App() {
  // language support en-US(default) / zh-CN / ja-JP / ko-KR / zh-TW
  // theme support light(default) / dark
  return (
    <div style={{display: 'flex', height: '100vh'}}>
      <UIKitProvider language='en-US' theme='light'>
        <Chat>
          <ChatHeader />
          <MessageList />
          <MessageInput />
        </Chat>
        <ChatSetting />
      </UIKitProvider>
    </div>
  );
}
```

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。

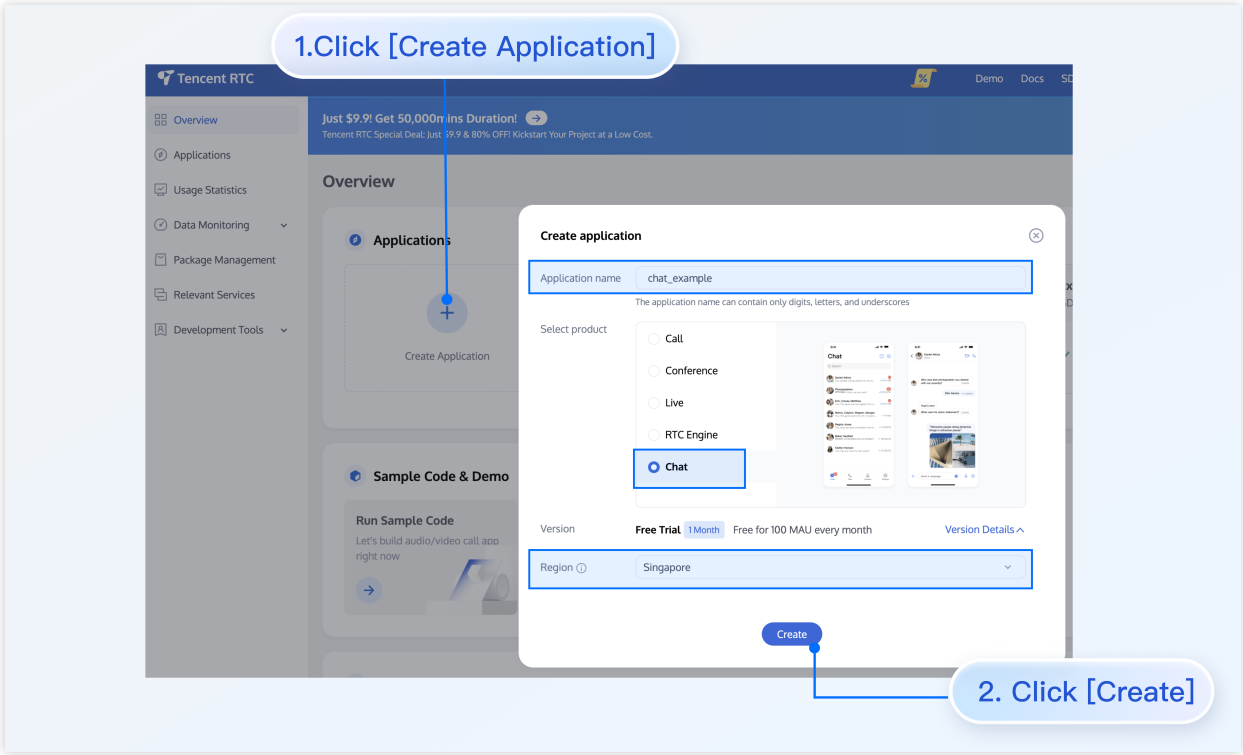
进入聊天

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

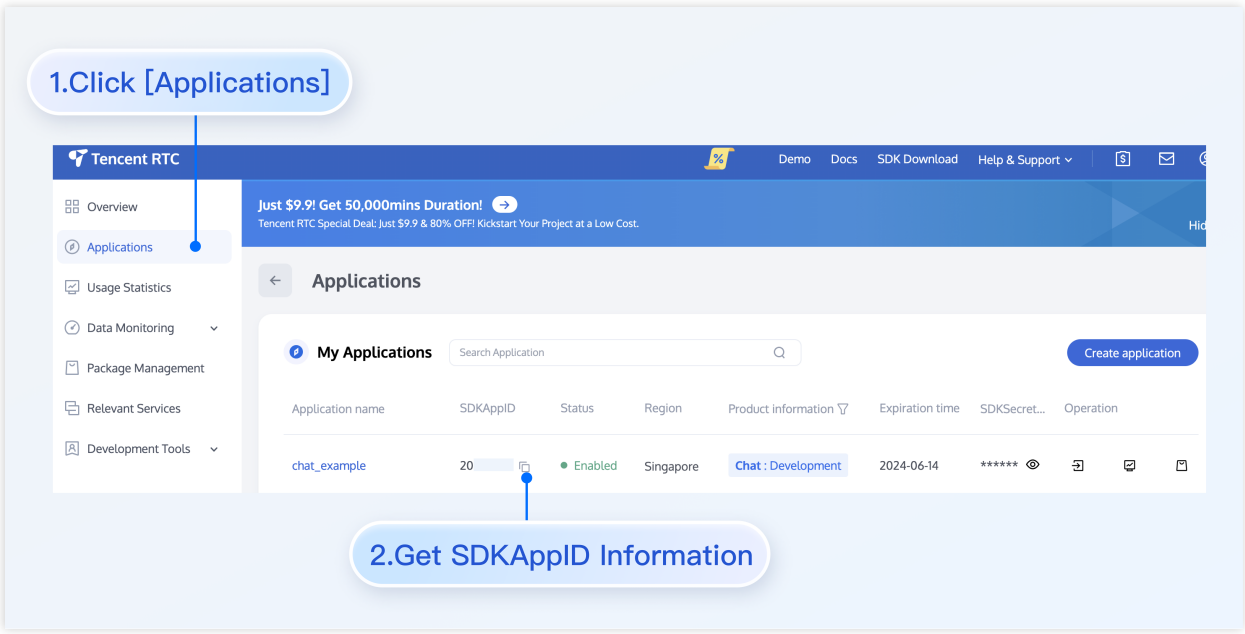
步骤4：创建一个应用

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**。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

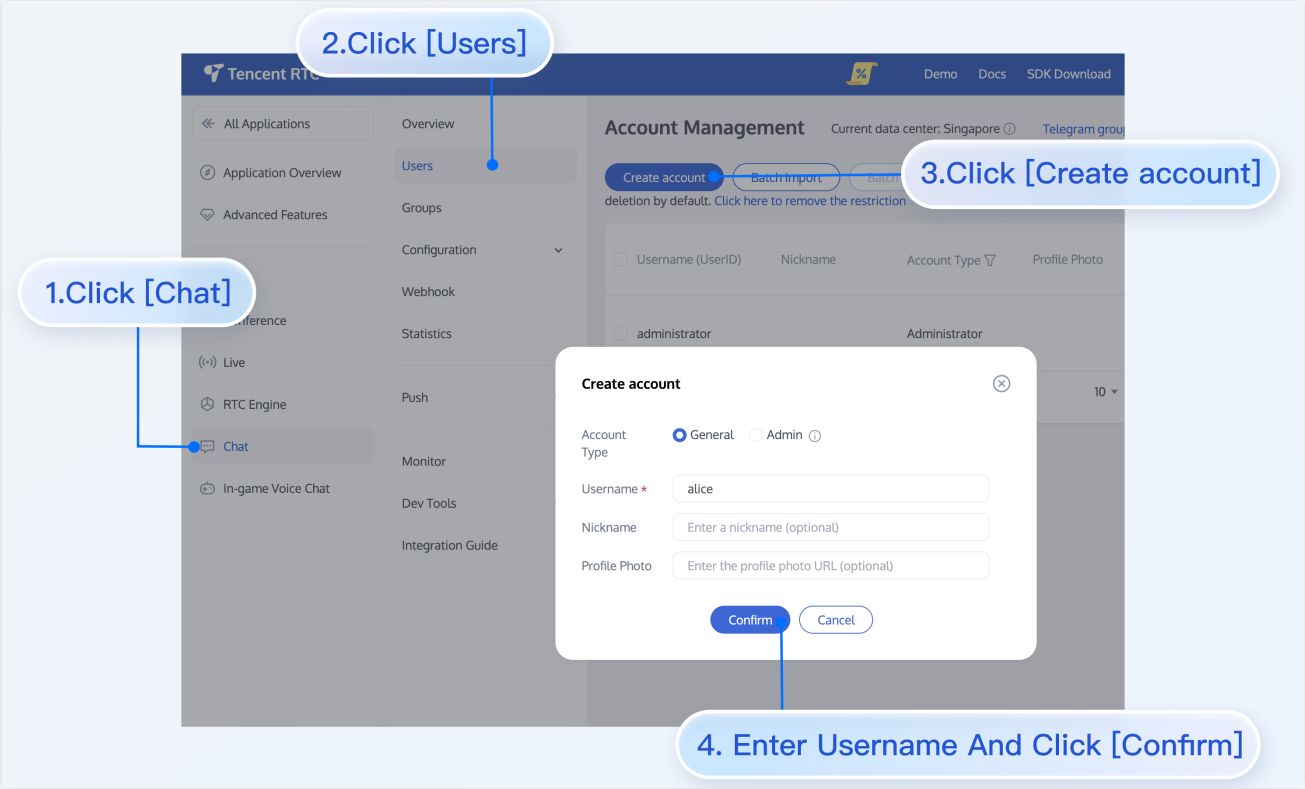
`userID`

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。

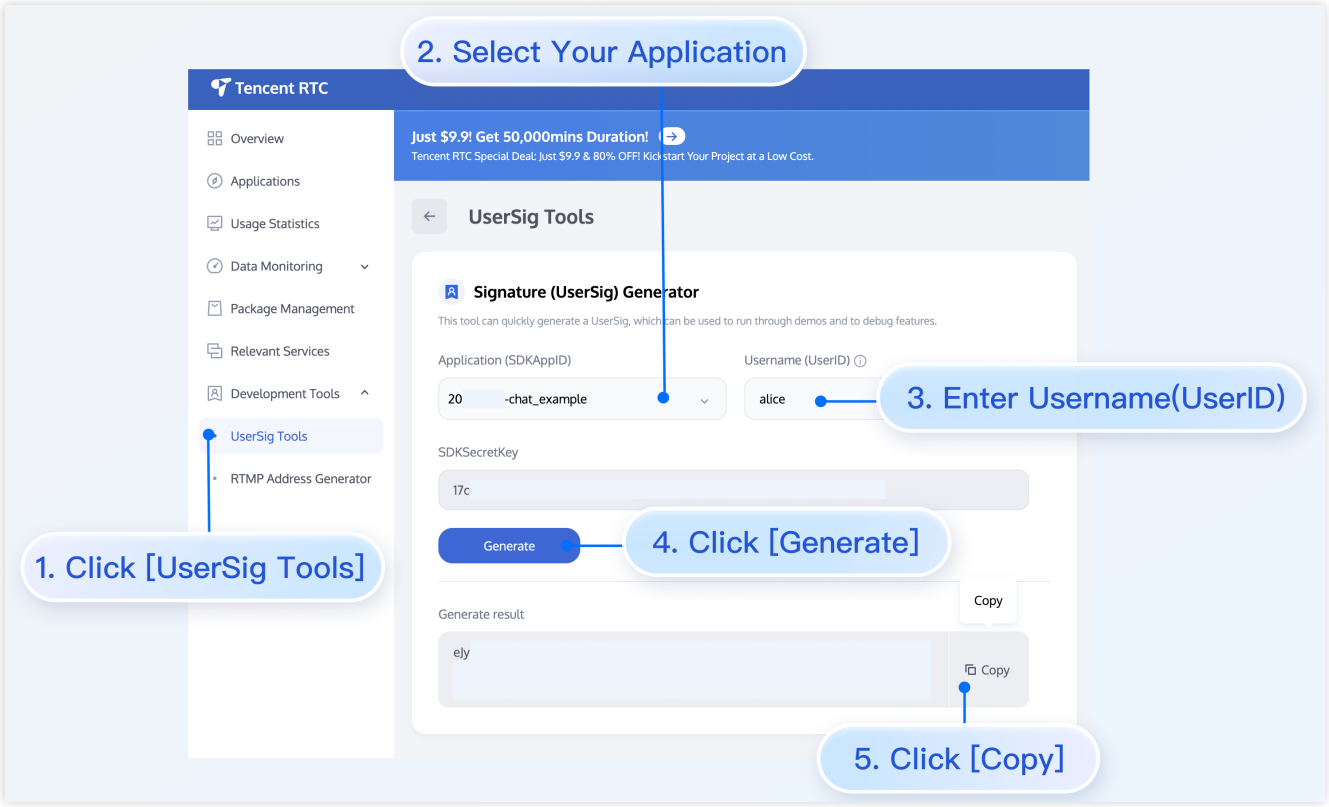
©2013-2025 Tencent Cloud International Pte. Ltd.

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 。

©2013-2025 Tencent Cloud International Pte. Ltd.



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
```

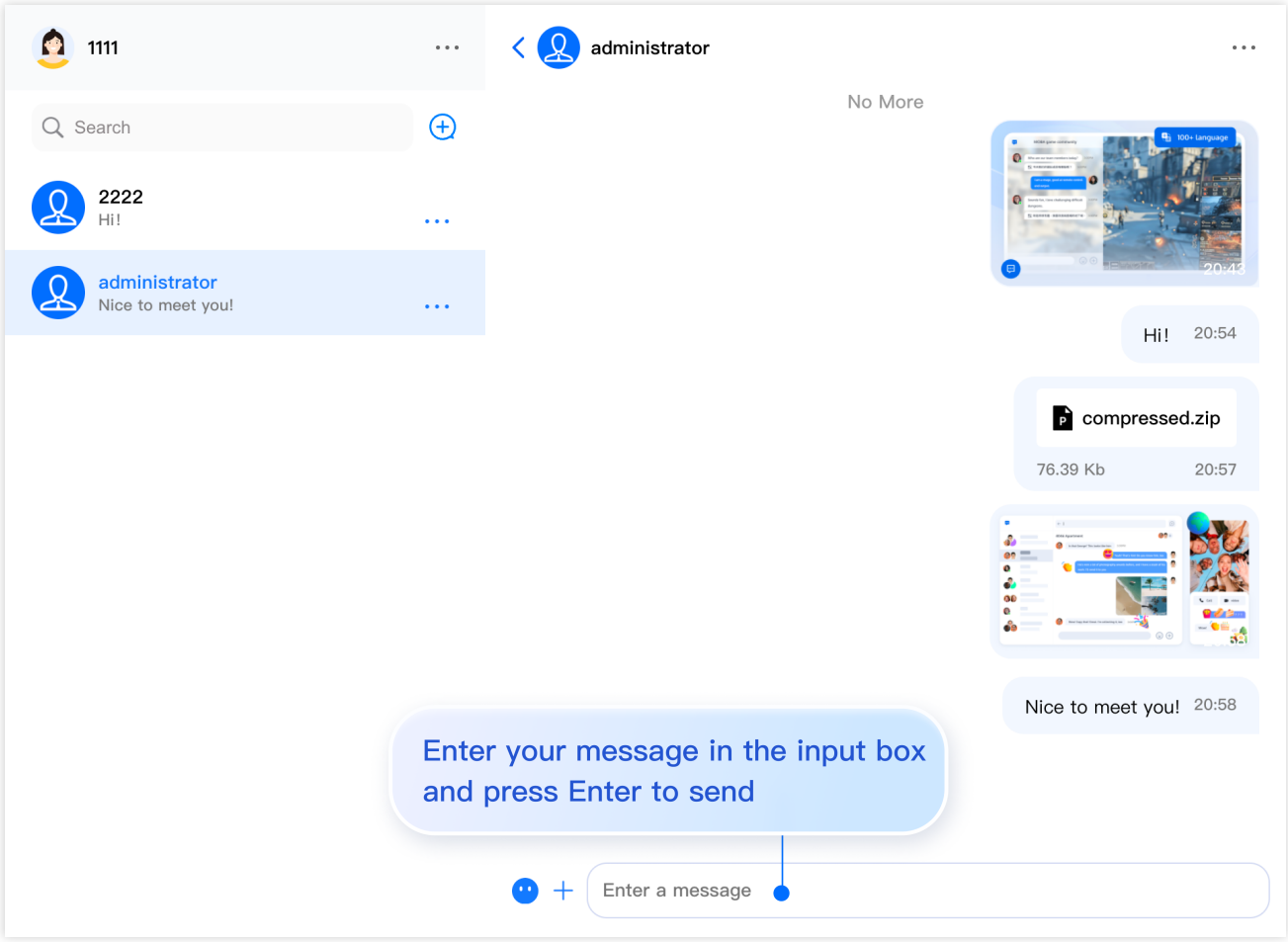
注意：

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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？

我们建议您优先采用 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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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/src/** ./src/TUIKit
```

```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-react\\src\\ .\\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 { ConversationList } from 'tencentcloud/chat-uikit-react';
// after
import { ConversationList } from './TUIKit';
```

3. 若运行报错，可参考修改以下配置，支持 create-react-app 源码导入。

3.1 安装 sass 参考 create-react-app 文档

3.2 关闭 tsconfig.json 中的 `isolatedModules`

3.3 配置 eslint 以关闭对 src/TUIKit 源码的检查

3.4 修改文件 src/TUIKit/styles/fonts/icon-font.scss 中的字体资源路径，相对定位到 `../../assets/fonts/*`

3.5 配置 webpack.config.js 中主题颜色导入的别名路径 `resolve.alias`

```
module.exports = {
    resolve: {
      alias: {
        '~@tencentcloud/uikit-base-component-react/dist/styles/theme/util': path.re
      }
    }
}
```

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 （Vue）

最近更新时间：2025-03-03 11:29:56

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。



©2013-2025 Tencent Cloud International Pte. Ltd.





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)

©2013-2025 Tencent Cloud International Pte. Ltd.

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 版本 ≤ 1.77.4，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@1.77.0 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：



©2013-2025 Tencent Cloud International Pte. Ltd.

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
```

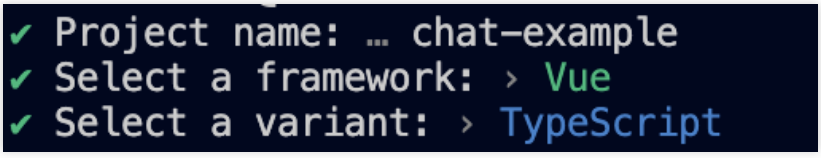
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compiler@2.6.14
```

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@1.77.0 sass-loader@10.1.1
```

说明：

如果您的 tsconfig.json 已有 references 配置，比如 vite 官方自动配置的 "./tsconfig.app.json"，"./tsconfig.node.json"；则请在这两个规则对应的文件 tsconfig.app.json 和 tsconfig.node.json 的 compilerOptions 分别添加以下规则。

typescript ≥ 5.0.0

typescript < 5.0.0

```
{
  ...
  "compilerOptions": {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    "verbatimModuleSyntax": false,
  }
}
```

```
{
  ...
  "compilerOptions": {
    "importsNotUsedAsValues": "preserve",
  }
}
```

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','excluded-list.txt'}
./node_modules/@tencentcloud/chat-uikit-vue/ ./src/TUIKit
```

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

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

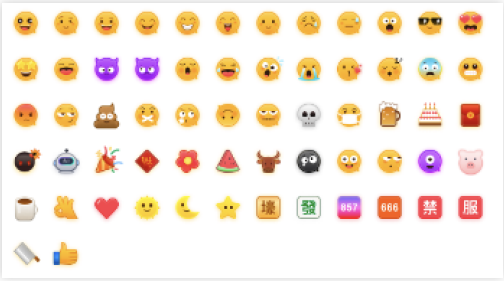
步骤3：引入 TUIKit 组件

在需要展示的页面，引入 TUIKit 的组件即可使用。

说明：

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

```
<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>
```

vue2.7

```
<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>
```

vue2.6及以下

```
<template>
  <div id="app">
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    <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：

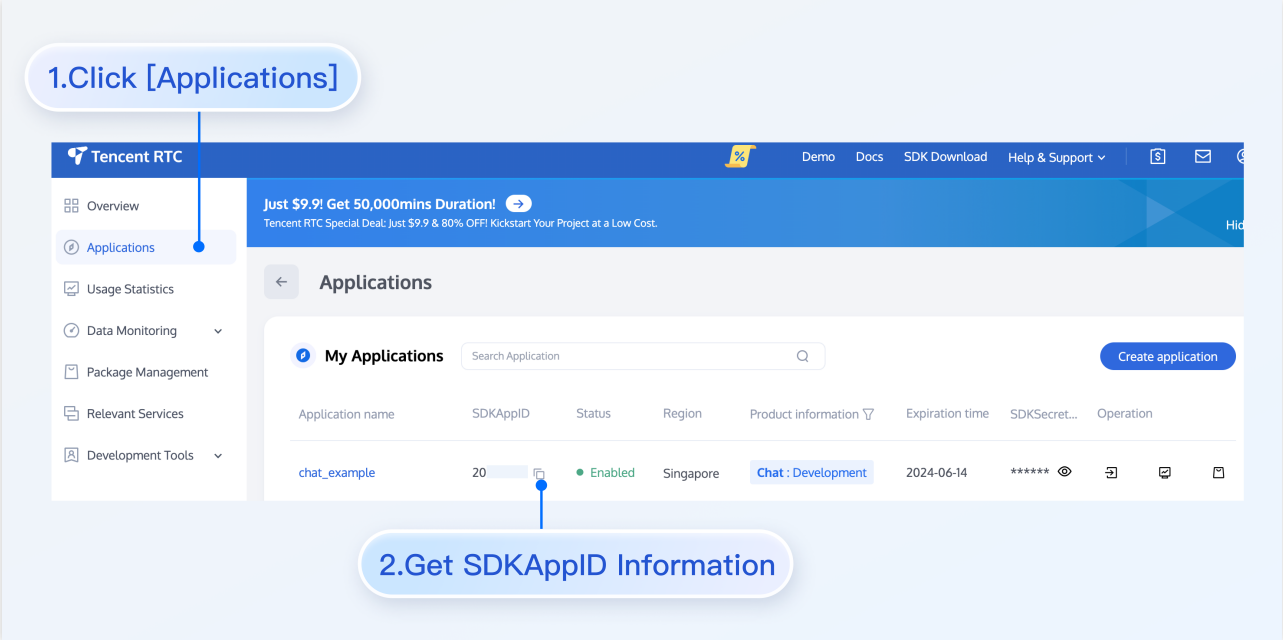
```
// 初始写法
export * from "vue";
// 替换为
export * from "@vue/composition-api";
```

步骤4： 获取 SDKAppID 、userID 与 userSig

©2013-2025 Tencent Cloud International Pte. Ltd.

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`

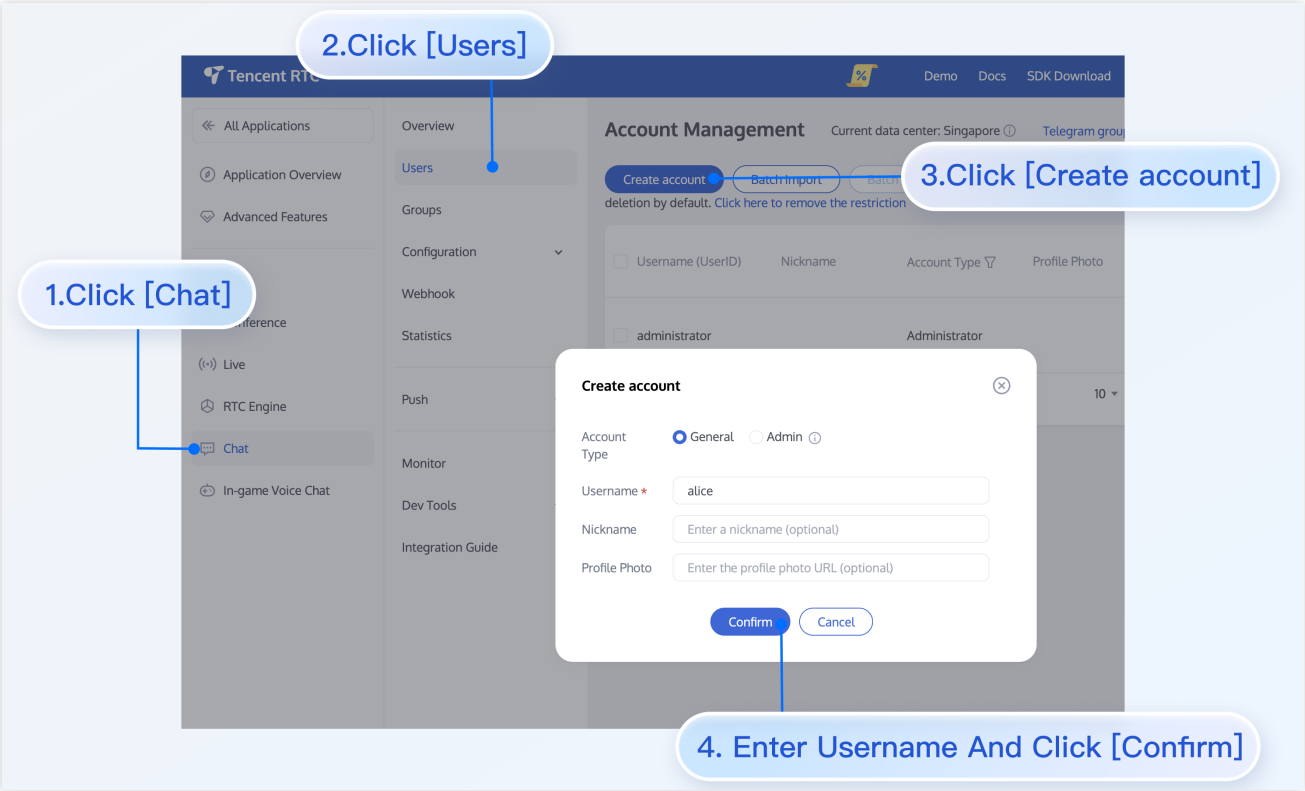
单击进入您上面创建的 Application，会在左侧边栏看到 `Chat` 产品入口，单击进入。

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。

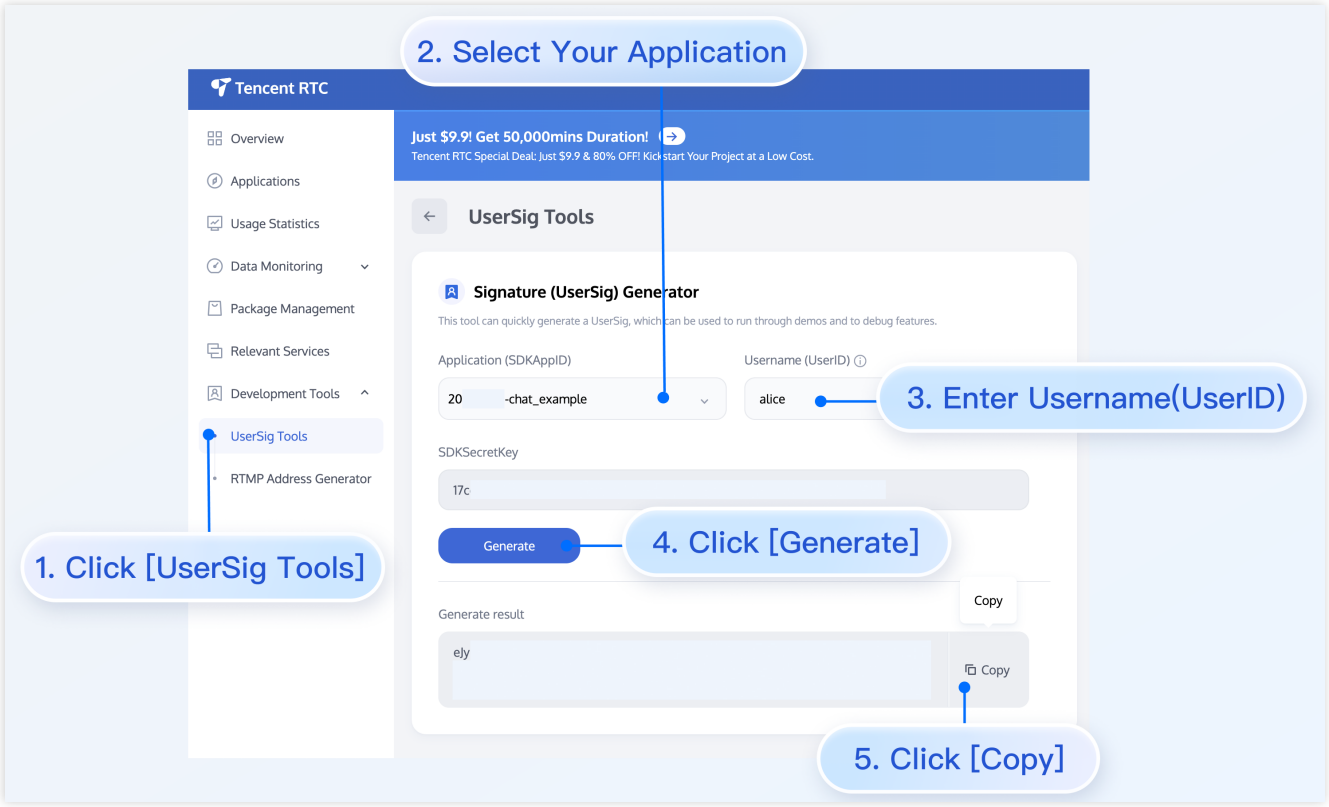
为了您更好的体验消息收发等功能，建议您创建两个 userID。

©2013-2025 Tencent Cloud International Pte. Ltd.



`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 。

©2013-2025 Tencent Cloud International Pte. Ltd.



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,
  },
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
};
```

```
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。

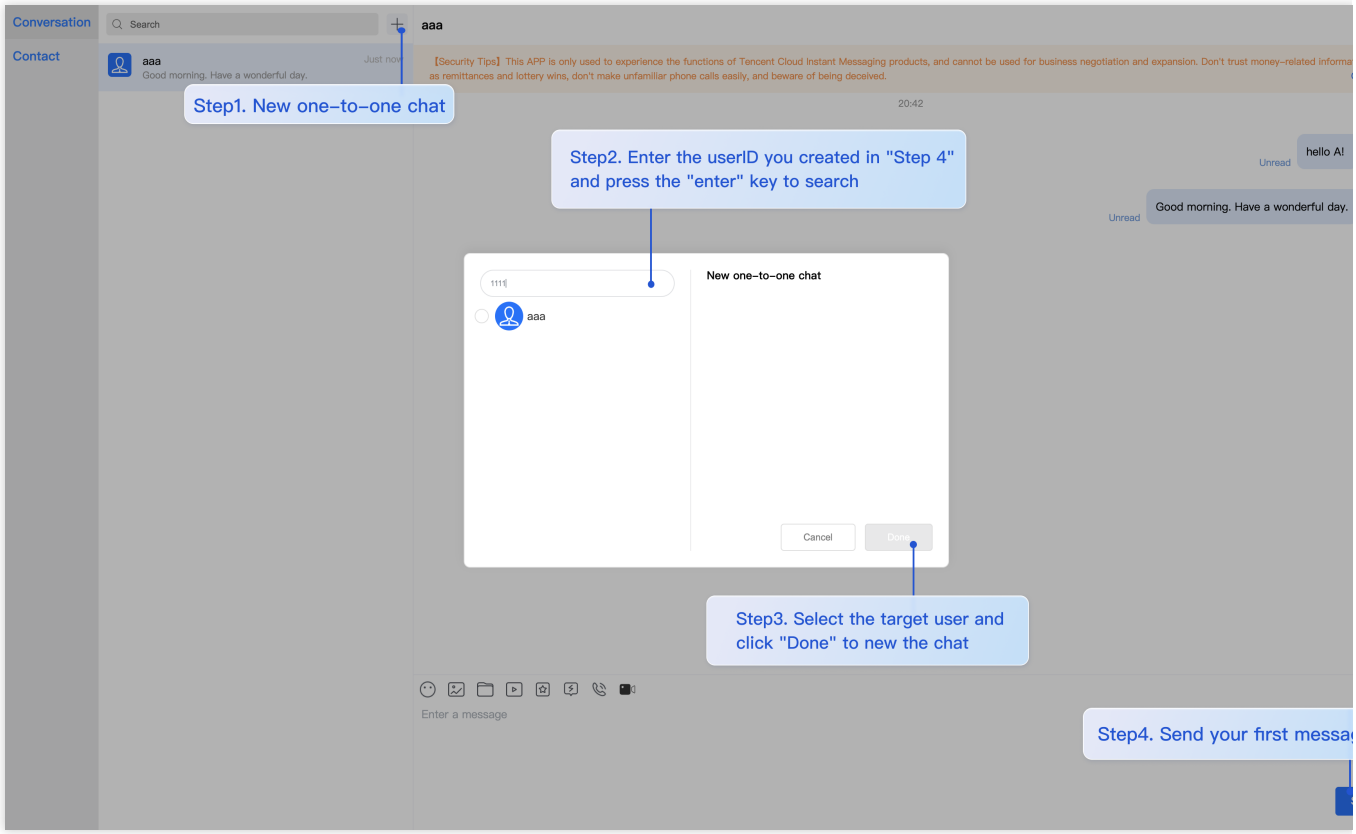
```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

步骤6：发送您的第一条消息

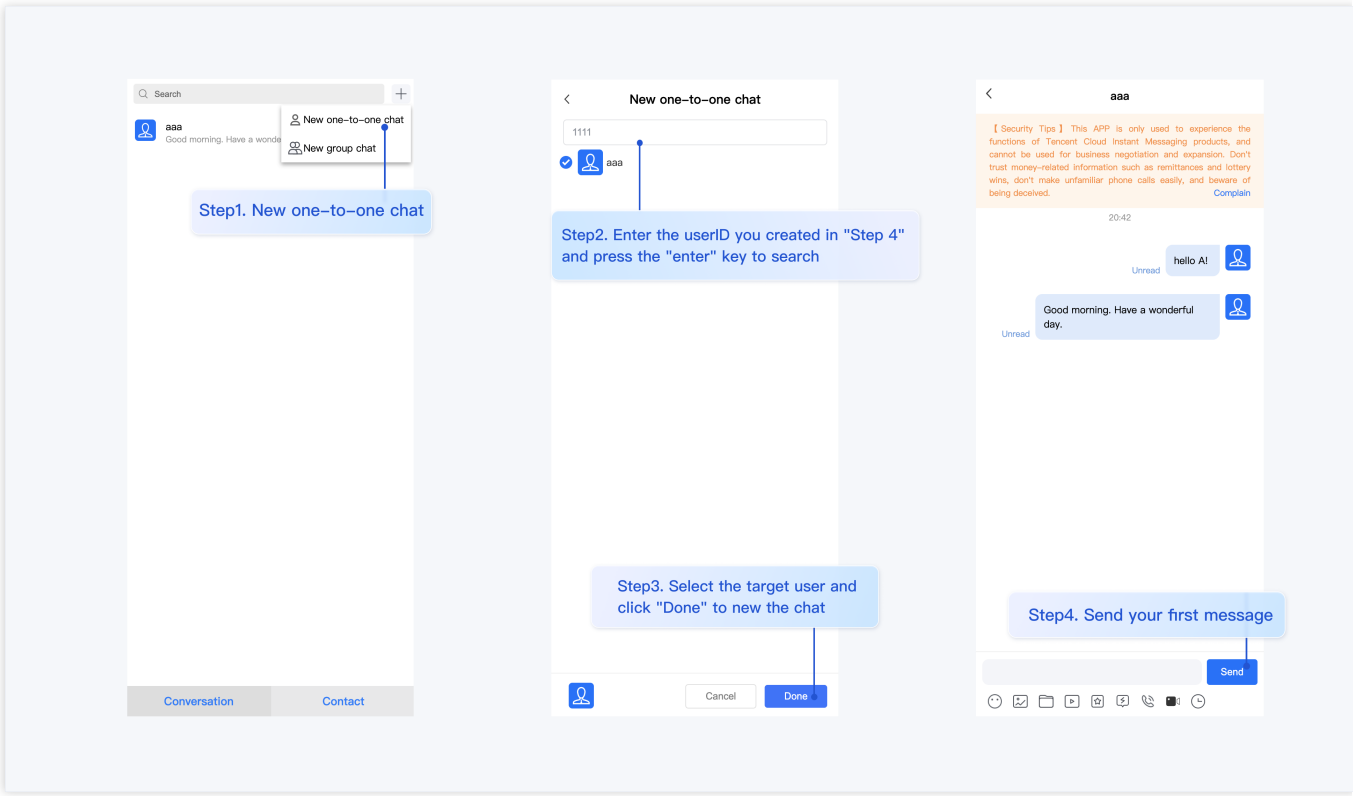
1. 项目启动之后单击左上角**发起单聊**。
2. 进入**发起单聊**弹窗。在搜索栏输入 步骤4 中创建的 userID，选中后单击**完成**。
3. 在输入框中输入消息并单击**发送**。

Web 端 “发送您的第一条消息” 具体步骤示例：

©2013-2025 Tencent Cloud International Pte. Ltd.

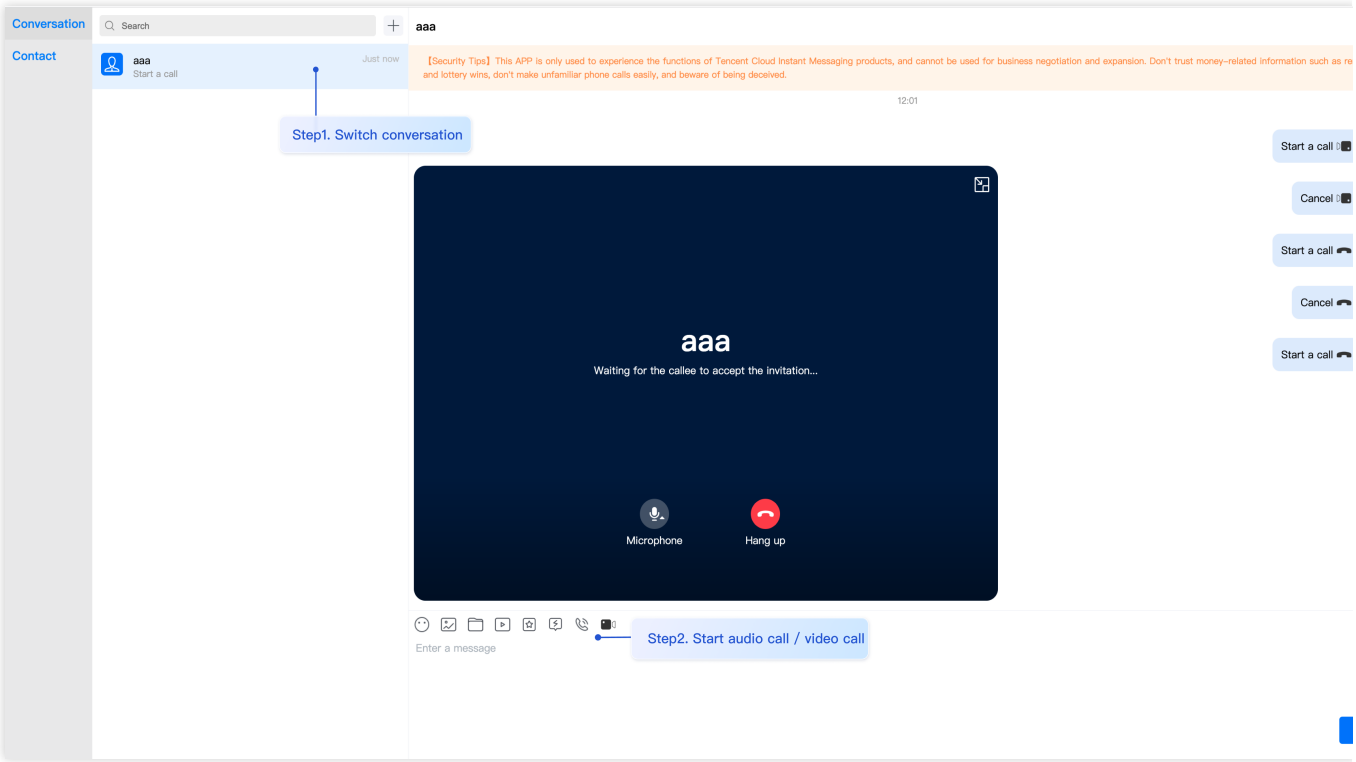


H5 端 “发送您的第一条消息” 具体步骤示例：



©2013-2025 Tencent Cloud International Pte. Ltd.

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？

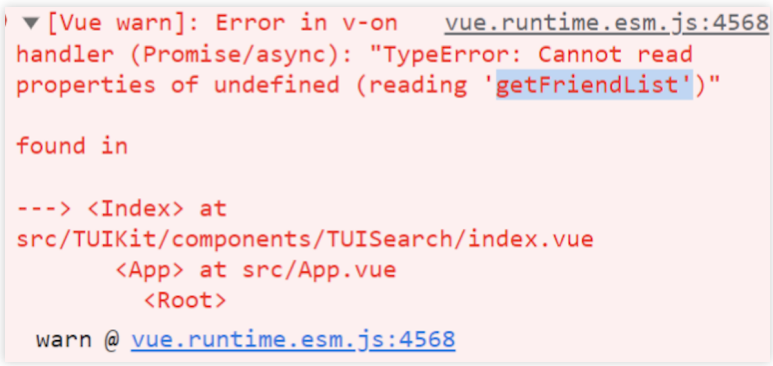
UserSig 是用户登录 Chat 的密码，其本质是对 UserID 等信息加密后得到的密文。
UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。
注意：
本文示例代码采用的获取 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**。正确的 UserSig 签发方式请参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

接入报错类问题

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

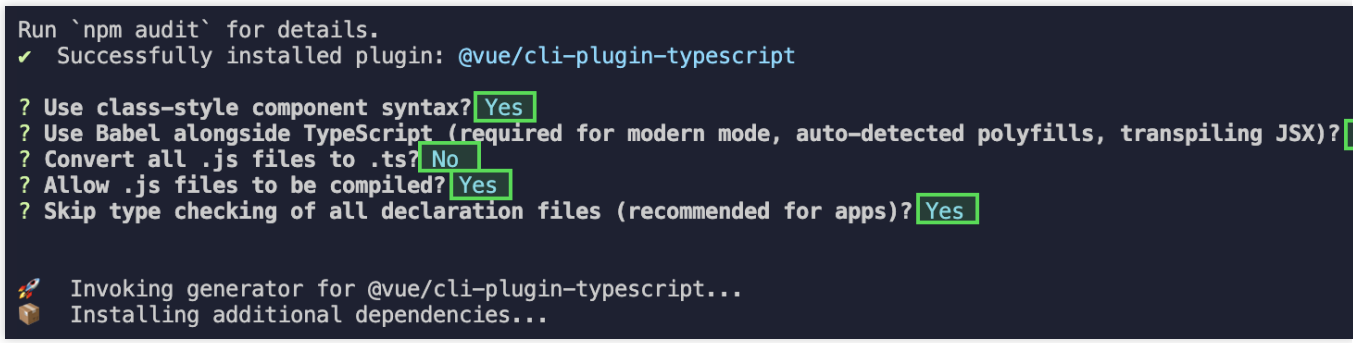
vue-cli

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

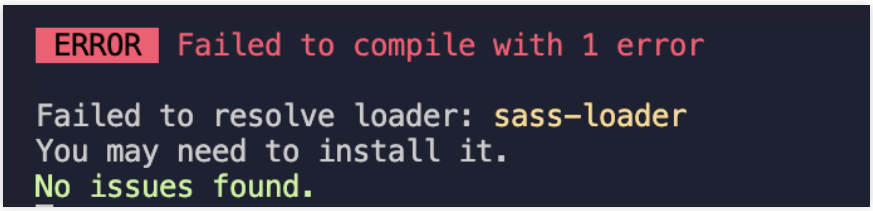
3. 运行时报错：/chat-example/src/TUIKit/components/TUIChat /message-input/message-input-editor.vue.ts（8，23）TS1005: expected.



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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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？

Chat TUIKit web 所使用的 ESLint 版本为 v6.7.2 ，对于模块名的驼峰式格式并不进行严格校验。
如果您出现此问题，您可以在 .eslintrc.js 文件中进行如下配置：

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

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

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

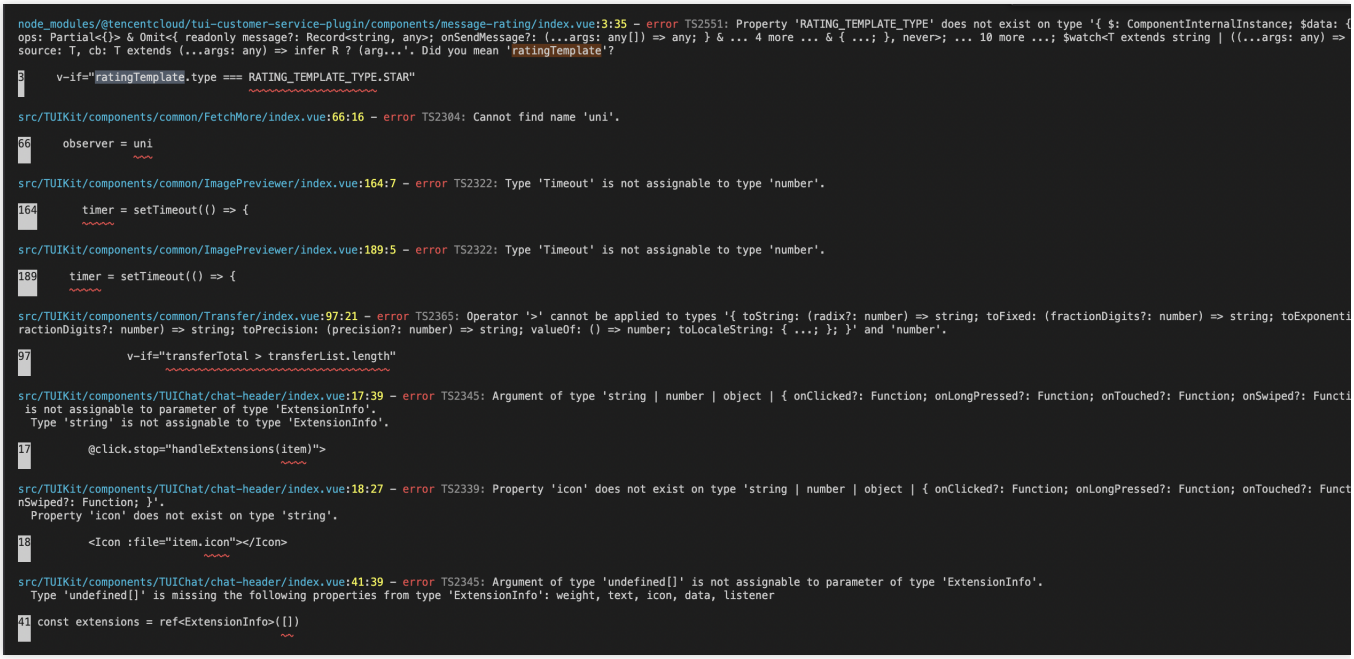
```
npm install --legacy-peer-deps
```

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

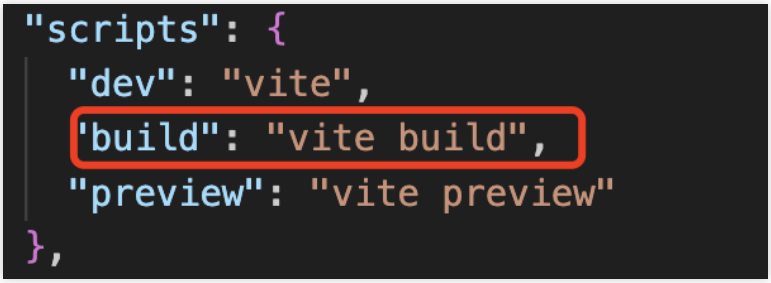
10. vite 项目 npm run build 之后 ts 报错?



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"



©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

Unity

最近更新时间：2025-06-10 11:59:53

Chat for Unity on iOS or Android.
此 Chat(Instant Messaging) Unity UIKit & UIKit Demo 是基于Tencent Cloud Chat 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。
Chat Unity UIKit & UIKit Demo 链接
即时通信 Chat 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 Chat 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 Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

说明：

为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

初始化并登录

初始化并登录 Chat 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 Chat, 注意 Chat 应用只需初始化一次即可。如若在现有 Chat 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);
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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 Chat 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

©2013-2025 Tencent Cloud International Pte. Ltd.

`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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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 Chat 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);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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 Chat 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;
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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
  },
  }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  };
```

4. 注册表情包给 UIKit：

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

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

语言包

Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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 Chat Chat SDK 文档链接

Tencent Cloud Chat Chat SDK 官网链接

Tencent Cloud Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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

©2013-2025 Tencent Cloud International Pte. Ltd.

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Chat社群，享有专业工程师的支持，解决您的难题。

©2013-2025 Tencent Cloud International Pte. Ltd.

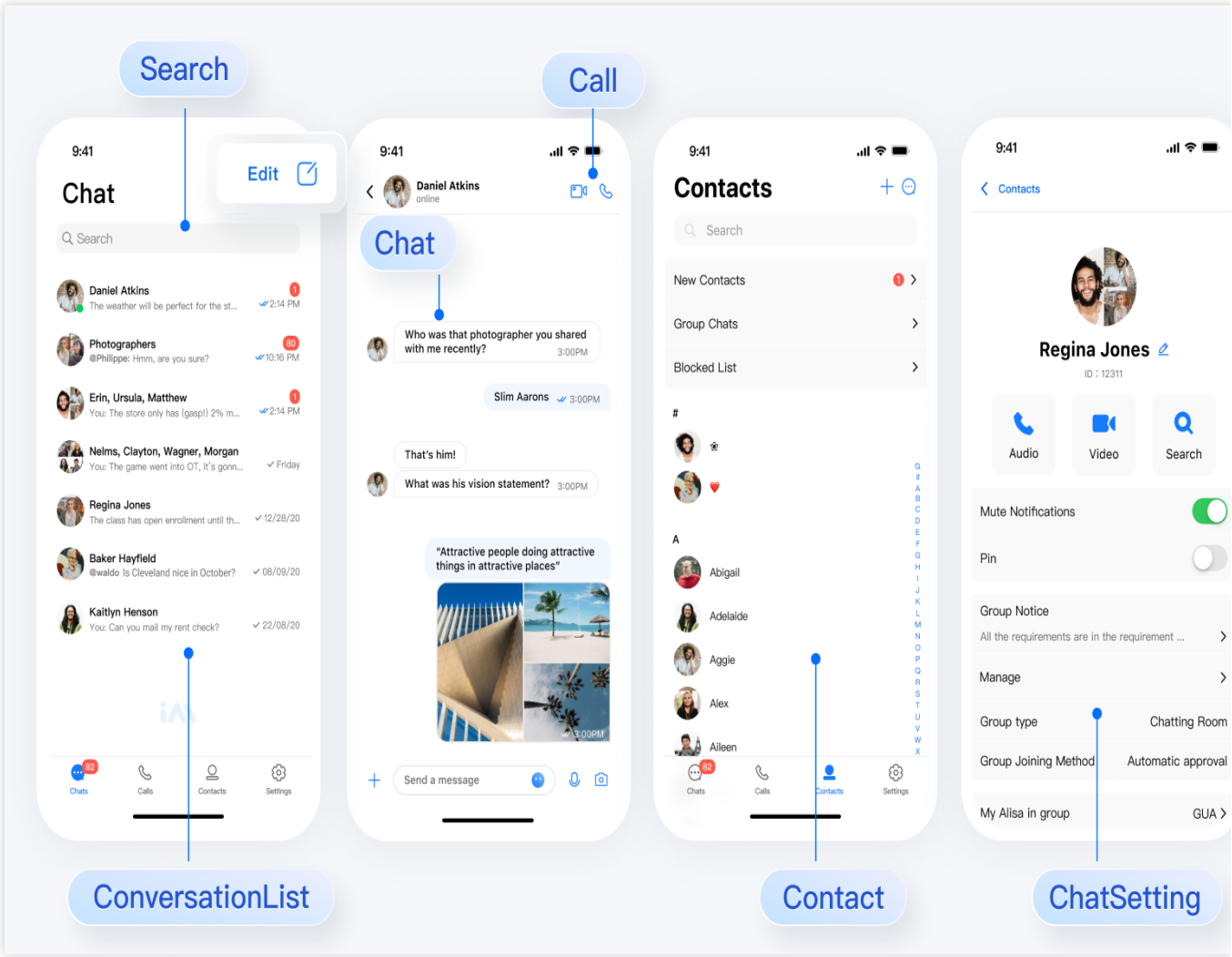
React Native

最近更新时间：2024-11-26 15:19:10

chat-uikit-react-native 介绍

chat-uikit-react-native 是一款基于腾讯云 Chat SDK 的 React Native UI 组件库，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。基于 React Native 开发的 UIKit 界面风格更契合海外客户的使用习惯，而且支持国际化，欢迎接入。

chat-uikit-react-native 界面效果如下图所示：



开发环境要求

React Native 0.75.0

Node.js version 18+

©2013-2025 Tencent Cloud International Pte. Ltd.

JDK 17
Xcode 版本 14.0 或更高版本
Android Studio

配置开发环境

如果您是首次开发 React Native 项目，请参见 React Native 官网步骤 set-up-your-environment 配置开发环境。
在创建项目或编译项目过程中如果遇到环境问题，可以运行 `npx react-native doctor` 进行环境诊断。

集成 chat-uikit-react-native

步骤1：创建项目（已有项目可忽略此步骤）

1. 创建一个新的 React Native 项目。

```
npx @react-native-community/cli@latest init MyChatApp --version 0.75.0
```

2. 创建项目完成后，切换到项目所在目录。

```
cd MyChatApp
```

步骤2：集成 chat-uikit-react-native

通过 npm / yarn 方式下载 chat-uikit-react-native 并在项目中使用，您也可在此基础上进行二次开发。

npm

yarn

```
npm install @tencentcloud/chat-uikit-react-native react-native-image-picker
react-native-document-picker react-native-video
```

```
yarn add @tencentcloud/chat-uikit-react-native react-native-image-picker react-
native-document-picker react-native-video
```

添加设备权限

Android

iOS

将以下权限添加到 android/app/src/main/AndroidManifest.xml 文件。

```
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
<uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

将以下权限使用描述添加到 info.plist 文件中。

```
<key>NSCameraUsageDescription</key>
<string> we would like to use your camera</string>
<key>NSPhotoLibraryUsageDescription</key>
<string> we would like to use your photo library</string>
<key>NSMicrophoneUsageDescription</key>
<string>we would like to use your microphone</string>
```

步骤3：设置导航

请安装 React Navigation 相关依赖，参考文档 React Navigation 。

npm

yarn

```
npm install @react-navigation/native react-native-screens react-native-safe-area-context @react-navigation/native-stack
```

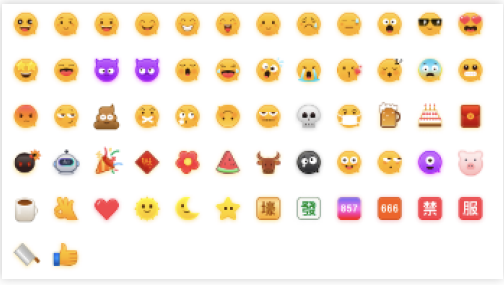
```
yarn add @react-navigation/native react-native-screens react-native-safe-area-context @react-navigation/native-stack
```

步骤4：引入 chat-uikit-react-native

说明：

以下代码中未填入 SDKAppID、userID 和 userSig，需在步骤5 中获取相关信息后进行替换。

为尊重表情设计版权，IM Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的小黄脸表情包版权归腾讯云所有，可有偿授权使用，如需获得授权可 提交工单 联系我们。



App.tsx

Screens.tsx

©2013-2025 Tencent Cloud International Pte. Ltd.

说明：

以下代码中未填入 `SDKAppID` 、 `userID` 和 `userSig` ，需在**步骤5**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。

```
import React from 'react';
import {
  View,
  TouchableOpacity,
  Text,
  Image,
  StyleSheet,
} from 'react-native';
import { NavigationContainer, useNavigation } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { UIKitProvider } from '@tencentcloud/chat-uikit-react-native';
import resources from '@tencentcloud/chat-uikit-react-native/i18n';
import { TUITranslateService } from '@tencentcloud/chat-uikit-engine';
import { TUILogin } from '@tencentcloud/tui-core';
import { ConversationListScreen, ChatScreen, ChatSettingScreen } from './Screens';

const LoginScreen = () => {
  const navigation = useNavigation<any>();
  // Init localization
  TUITranslateService.provideLanguages(resources);
  TUITranslateService.useI18n('en-US');
  // Login
  const Login = () => {
    TUILogin.login({
      SDKAppID: 0, // Your SDKAppID
      userID: 'test_1', // Login UserID
      userSig: '', // Login userSig
      useUploadPlugin: true,
      framework: 'rn',
    }).then(() => {
      navigation.navigate('ConversationList');
    });
  }

  return (
    <View style={styles.container}>
      <Image
        style={styles.logo}
        source={{uri:'https://web.sdk.qcloud.com/im/assets/images/tencent_rtc_logo.
      />
      <TouchableOpacity style={styles.buttonContainer} onPress={Login}>
        <Text style={styles.buttonText}>Log in</Text>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      </TouchableOpacity>
    </View>
  );
};

const Navigation = () => {
  const Stack = createNativeStackNavigator();
  return (
    <NavigationContainer>
      <Stack.Navigator
        screenOptions={{ headerShown: false }}
        initialRouteName="Login">
        <Stack.Screen
          name="Login"
          component={LoginScreen} />
        <Stack.Screen
          name="ConversationList"
          component={ConversationListScreen} />
        <Stack.Screen
          name="Chat"
          component={ChatScreen} />
        <Stack.Screen
          name="ChatSetting"
          component={ChatSettingScreen}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
};

const App = () => {
  return (
    <UIKitProvider>
      <Navigation />
    </UIKitProvider>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#FFFFFF',
  },
  logo: {
    width: 232,
    height: 80,
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  },
  buttonContainer: {
    width: '80%',
    justifyContent: 'center',
    alignItems: 'center',
    paddingVertical: 11,
    borderRadius: 5,
    backgroundColor: '#2F80ED',
  },
  buttonText: {
    fontSize: 18,
    lineHeight: 24,
    color: '#FFFFFF',
  },
});

export default App;
```

在与 App.tsx 文件相同目录中创建一个新的 Screens.tsx 文件。

```
import React from 'react';
import { useNavigation } from '@react-navigation/native';
import { ConversationList, Chat, ChatSetting } from '@tencentcloud/chat-uikit-react

export const ConversationListScreen = () => {
  const navigation = useNavigation<any>();
  const onPressConversation = () => {
    navigation.navigate('Chat');
  };
  return (
    <ConversationList onPressConversation={onPressConversation} />
  );
};

export const ChatScreen = () => {
  const navigation = useNavigation<any>();
  const navigateBack = () => {
    navigation.goBack();
  };
  const navigateToChatSetting = () => {
    navigation.navigate('ChatSetting');
  };
  return (
    <Chat
      navigateBack={navigateBack}
      navigateToChatSetting={navigateToChatSetting}
    />
```

©2013-2025 Tencent Cloud International Pte. Ltd.

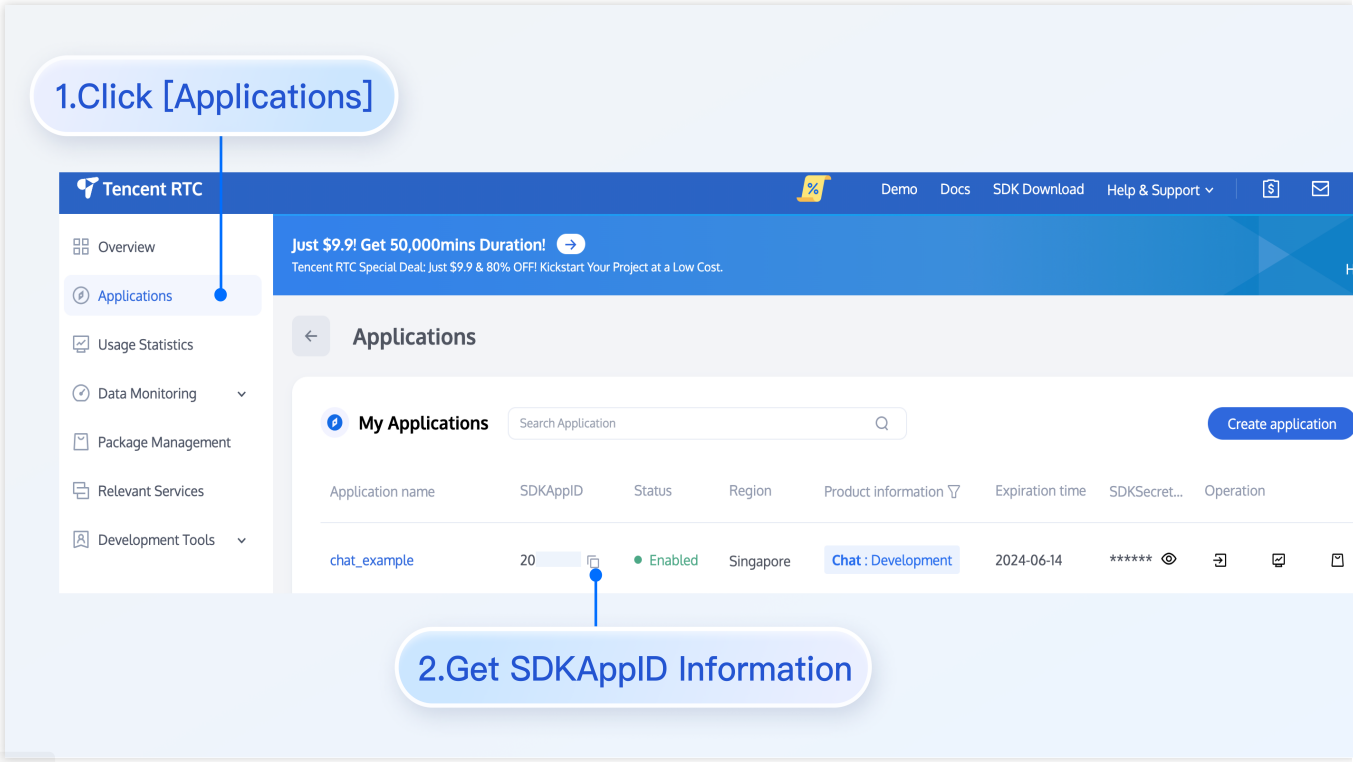
```
  );
};

export const ChatSettingScreen = () => {
  const navigation = useNavigation<any>();
  // Navigate to Chat when you click header back button.
  const navigateBack = () => {
    navigation.goBack();
  };
  // Navigate to Chat when you click the send message button.
  const navigateToChat = () => {
    navigation.goBack();
  };
  // Navigate to ConversationList when you disband group or leave group.
  const navigateToConversationList = () => {
    navigation.navigate('ConversationList');
  };
  return (
    <ChatSetting
      navigateBack={navigateBack}
      navigateToChat={navigateToChat}
      navigateToConversationList={navigateToConversationList}
    />
  );
};
```

步骤5： 获取 SDKAppID 、userID 与 userSig

获取 `Login` 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:

©2013-2025 Tencent Cloud International Pte. Ltd.



`userID`

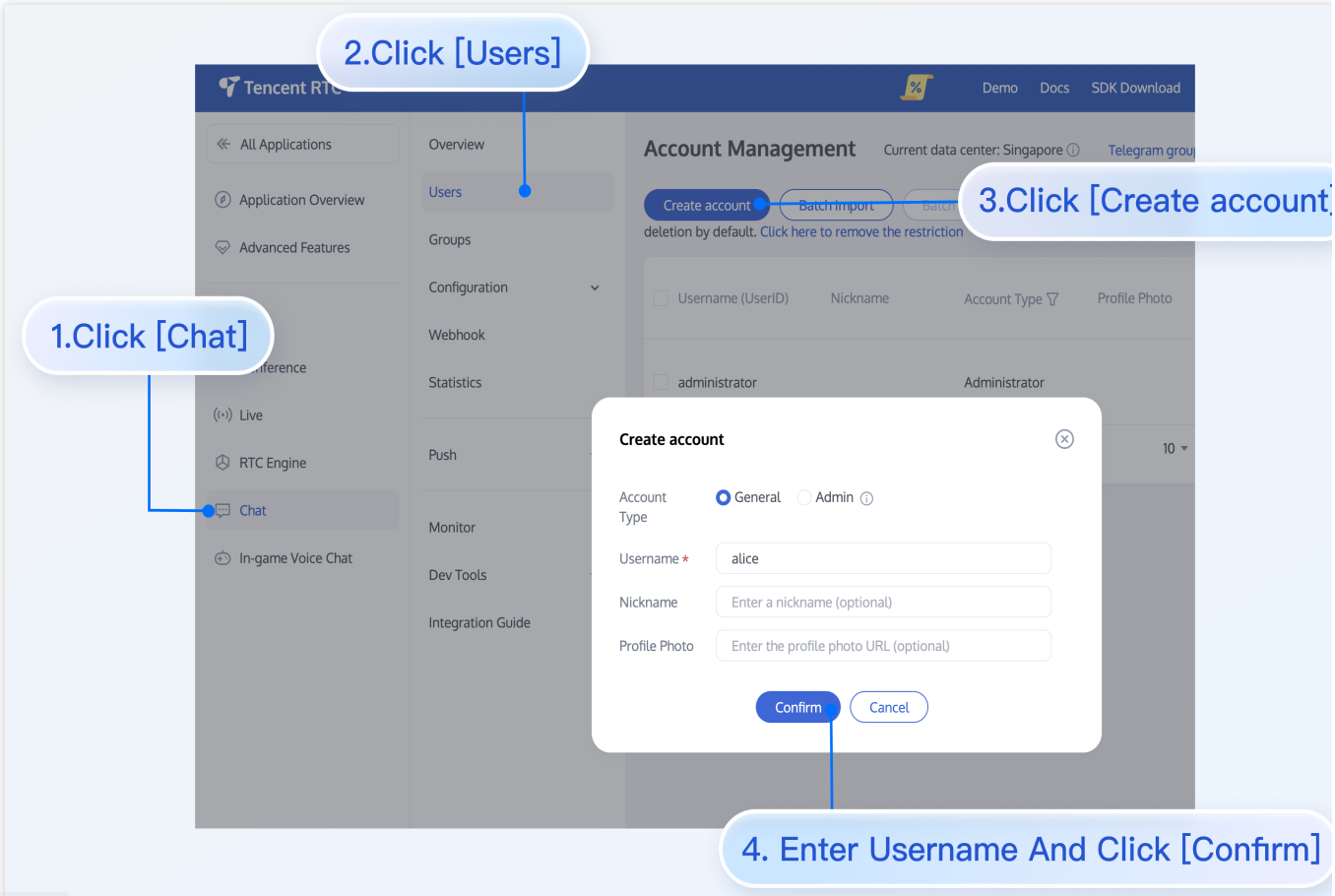
单击进入您上面创建的 Application，会在左侧边栏看到 `Chat` 产品入口，单击进入。

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。

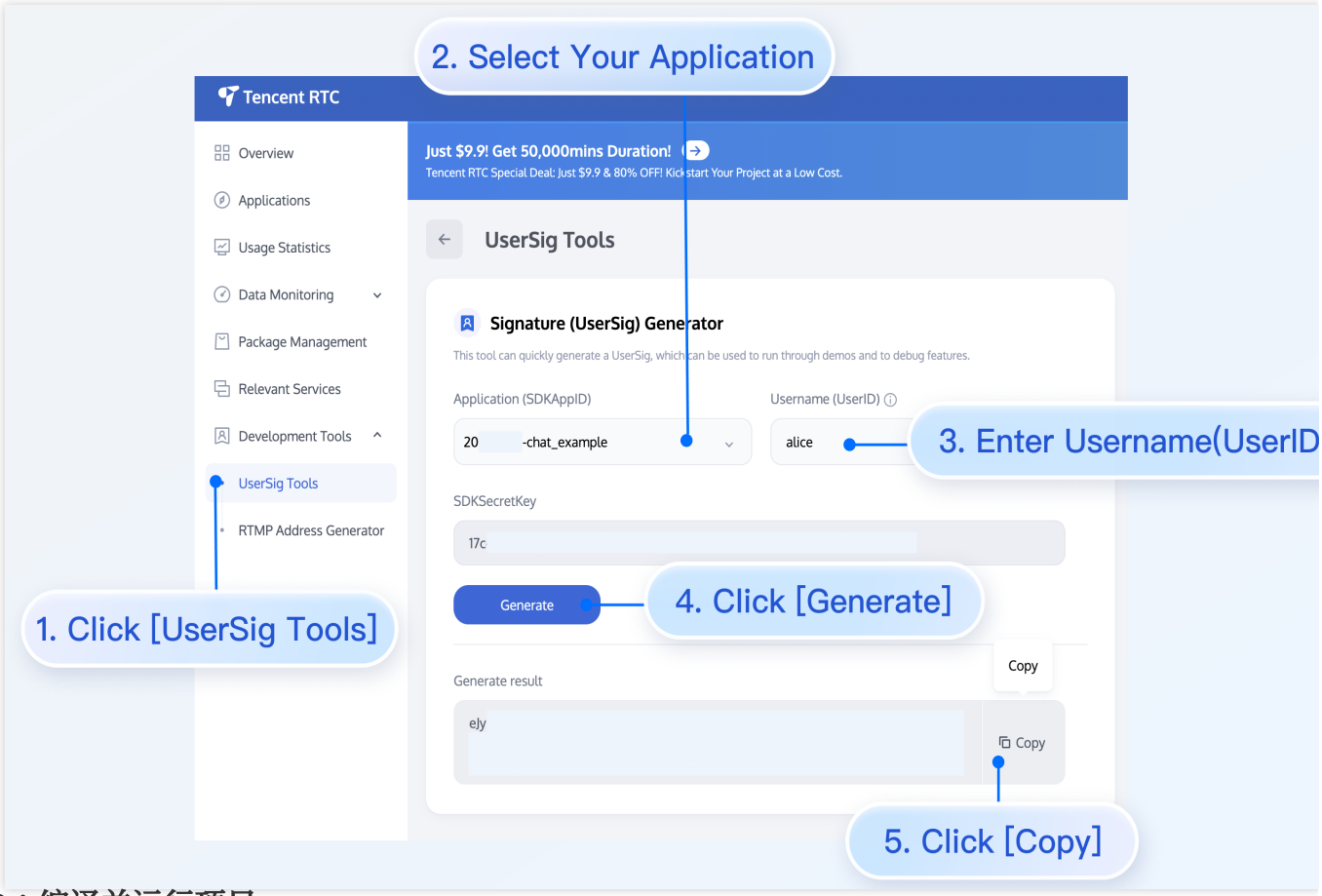
为了您更好的体验消息收发等功能，建议您创建两个 userID(test_1, test_2)。

©2013-2025 Tencent Cloud International Pte. Ltd.



`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 。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤6：编译并运行项目

编译运行项目您需要使用真机或模拟器，推荐使用真机运行。您可以参见 React Native 官网 running-on-device 连接真机进行调试。

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

Android

iOS

1. 手机开启开发者模式，打开 **USB 调试**开关。
2. 用 USB 连接手机，推荐选择 **传输文件** 选项，**不要选择 仅充电 选项**。
3. 确认手机连接成功后，执行 `npm run android` 编译运行项目。

```
npm run android
```

1. 用 USB 连接手机，用 Xcode 打开工程的 ios 目录。
2. 按照 React Native 官网 running-on-device 配置签名信息。
3. 进入 ios 目录下安装依赖。

```
cd ios
pod install
```

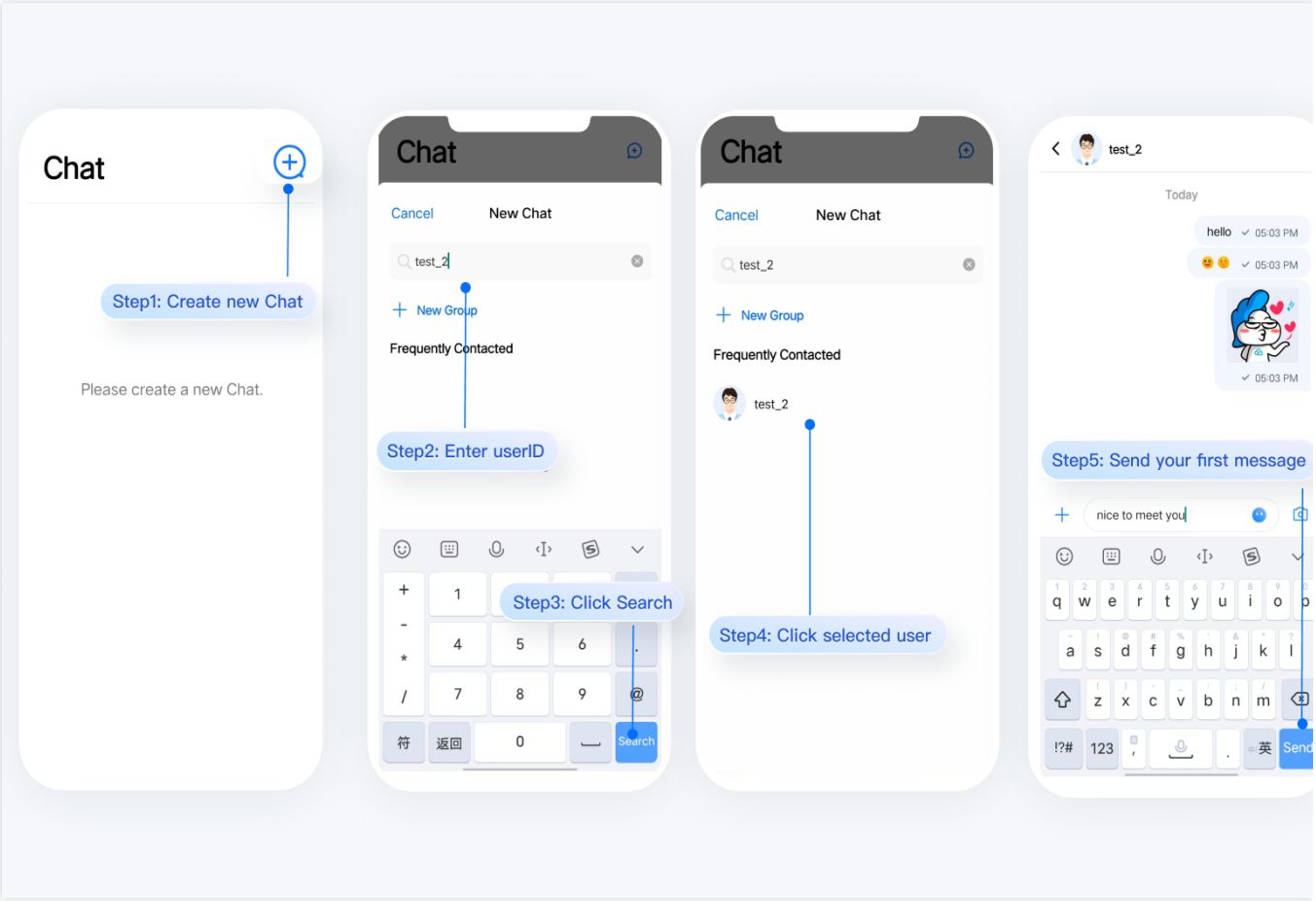
4. 回退到根目录，执行 npm run ios 编译运行项目。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
cd ../
npm run ios
```

步骤7：发送第一条消息

1. 项目启动之后单击左上角发起会话。
2. 进入发起会话窗口。在搜索栏输入 步骤5 中创建的 userID(**test_2**)，选中后打开会话。
3. 在输入框中输入消息并点击发送。



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。

常见问题

运行时环境报错怎么办？

©2013-2025 Tencent Cloud International Pte. Ltd.

可以执行以下命令，进行环境诊断。

```
npx react-native doctor
```

相关文档

UIKit 相关：

chat-uikit-react-native npm

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-05-27 12:42:15

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。

我们的 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、文本翻译、语音转文字等。

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）。

©2013-2025 Tencent Cloud International Pte. Ltd.

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：3.24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。
说明：
为尊重表情设计版权，Chat Demo/TUIKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，可有偿授权使用，如需获得授权可 提交工单 联系我们。

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_common。

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_sticker
tencent_cloud_chat_message_reaction
tencent_cloud_chat_text_translate
tencent_cloud_chat_sound_to_text
tencent_cloud_chat_search *（内测中）*
下面展示了我们的 UIKit 的架构：

©2013-2025 Tencent Cloud International Pte. Ltd.

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 Chat 配置信息，例如 sdkappid、测试 userID、userSig 等。您可以参见 跑通 Demo。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

```
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` 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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(),
);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

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 Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// 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;
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    );
```

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

结语

我们希望这份文档能帮助您理解我们新的 Flutter Chat 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。

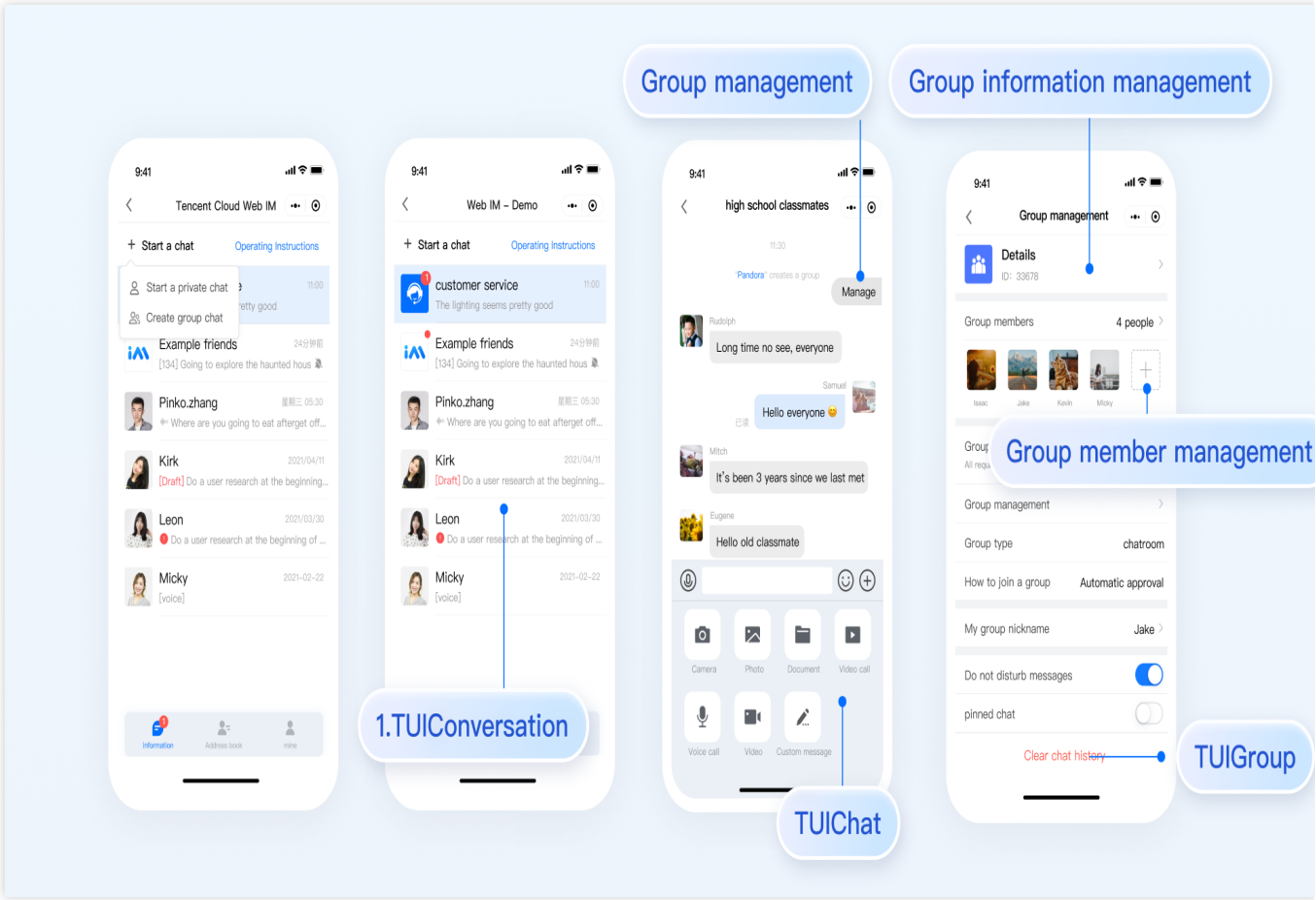
©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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）

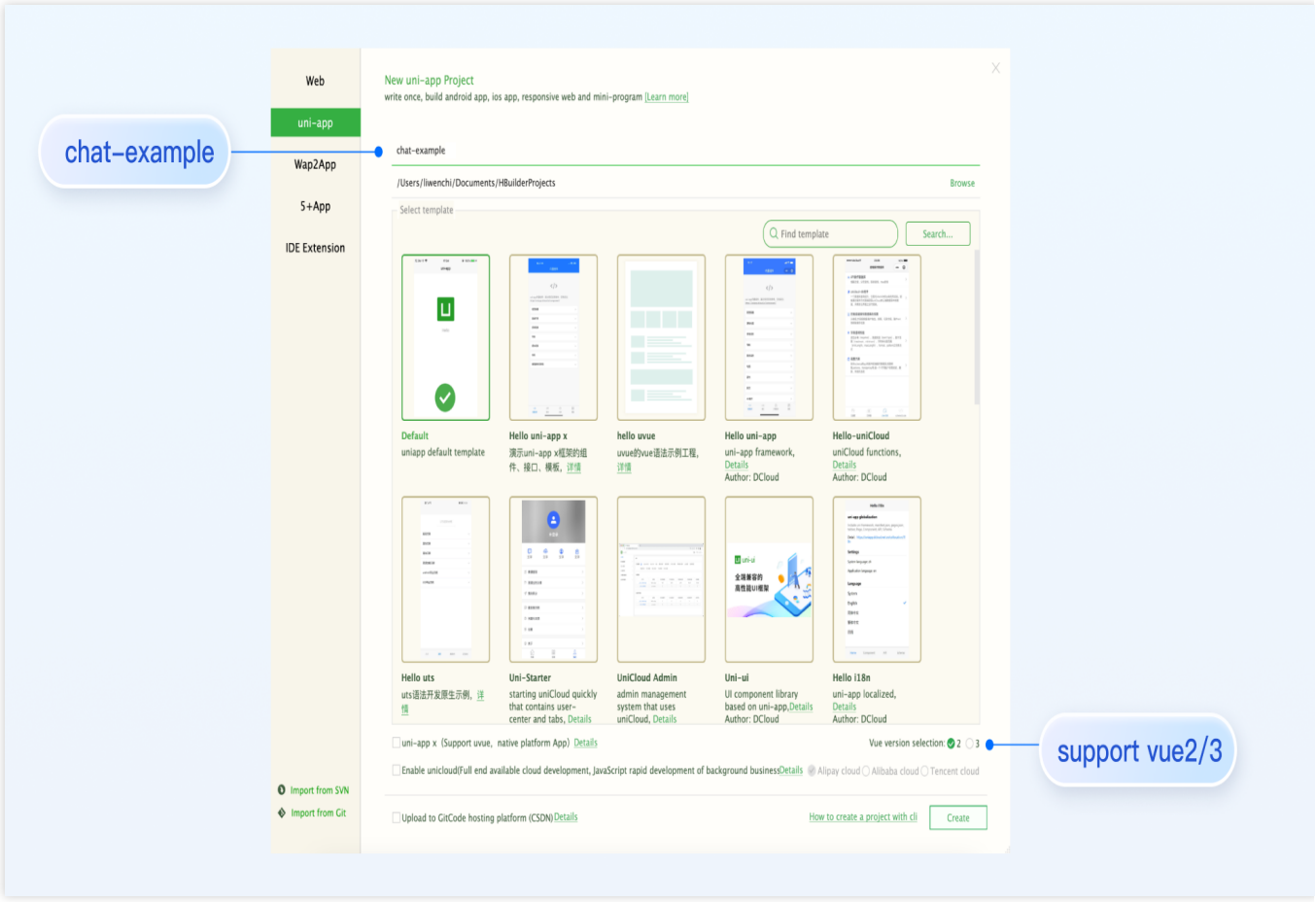
TUIKit 源码集成

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤2：下载 TUIKit 组件

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

```
npm init -y
```

下载 TUIKit 并拷贝至源码中：

macOS 端

Windows 端

通过 npm 方式下载 TUIKit 组件：

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

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

```
mkdir -p ./TUIKit && rsync -av --exclude=
{'node_modules','package.json','excluded-list.txt'}
./node_modules/@tencentcloud/chat-uikit-uniapp/ ./TUIKit
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

通过 npm 方式下载 TUIKit 组件：

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

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

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

```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin
.\\TUIKit\\tui-customer-service-plugin /i /e
```

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        }
    },
    "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 {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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",
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

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

// #ifdef APP-PLUS || H5
uni.$chat_userSig = genTestUserSig(config).userSig;
// TUIChatKit 初始化
TUIChatKit.init();
// #endif
export default {
  onLaunch: function () {
    // #ifdef APP-PLUS || H5
    // TUICore login
    TUILogin.login({
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        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";
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// 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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<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>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
.index {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  &-button {
    width: 180px;
        padding: 10px 40px;
        color: #fff;
        background-color: #006eff;
        font-size: 16px;
        margin-top: 65px;
        border-radius: 30px;
        text-align: center;
  }
}
</style>
```

步骤4：获取 SDKAppID、secretKey、userID

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

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

获取 SDKAppID、secretKey

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

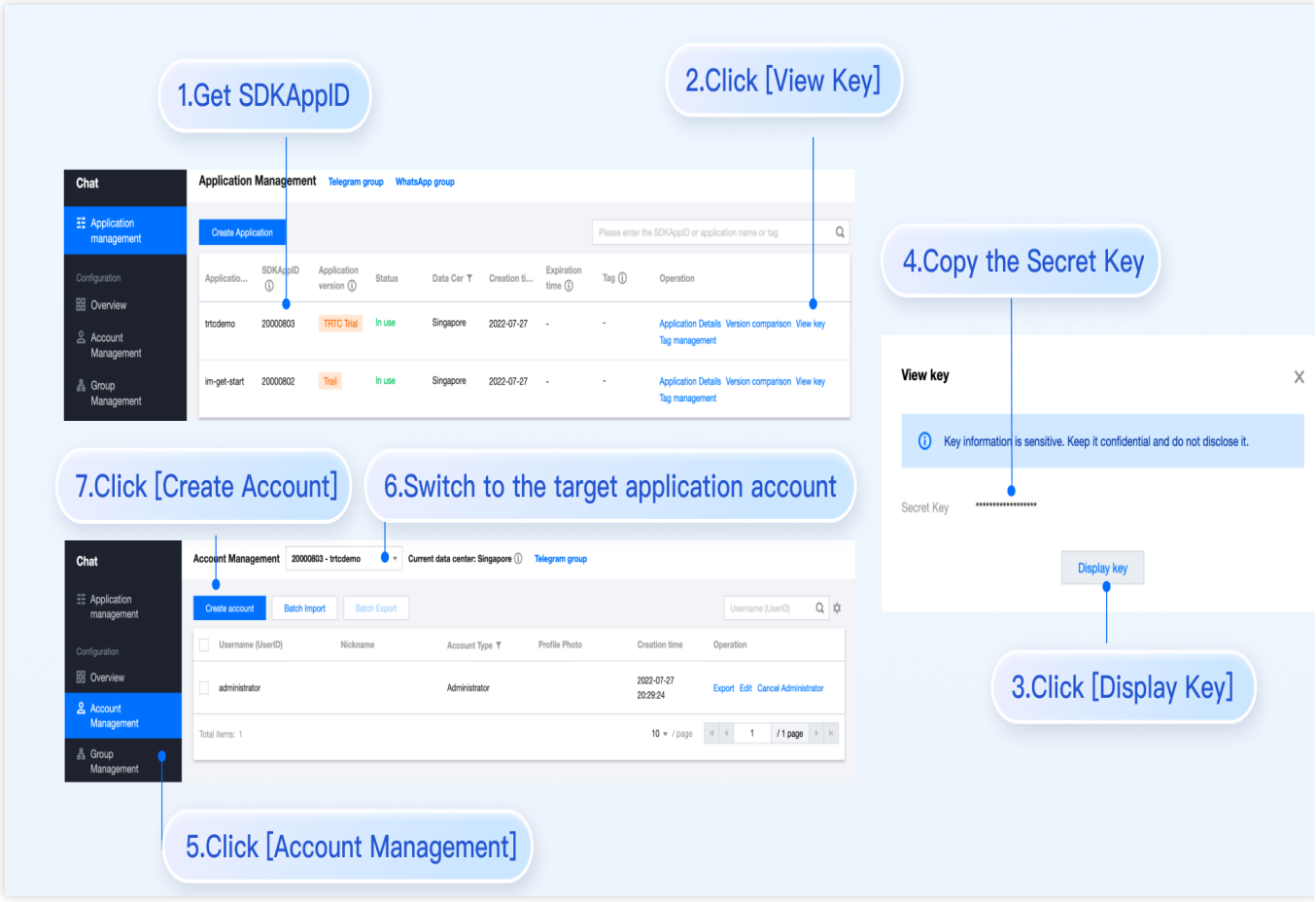
创建 userID 为 test-user1 的账户

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

说明：

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

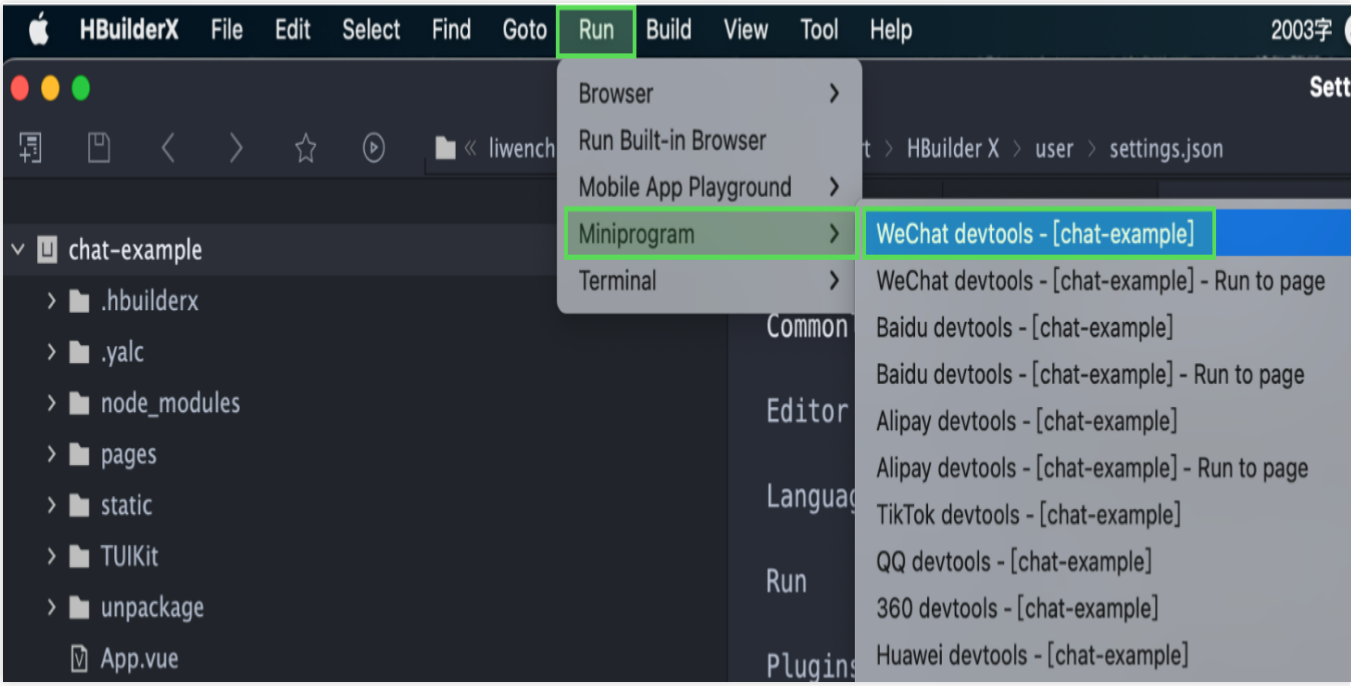
©2013-2025 Tencent Cloud International Pte. Ltd.



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”。

©2013-2025 Tencent Cloud International Pte. Ltd.

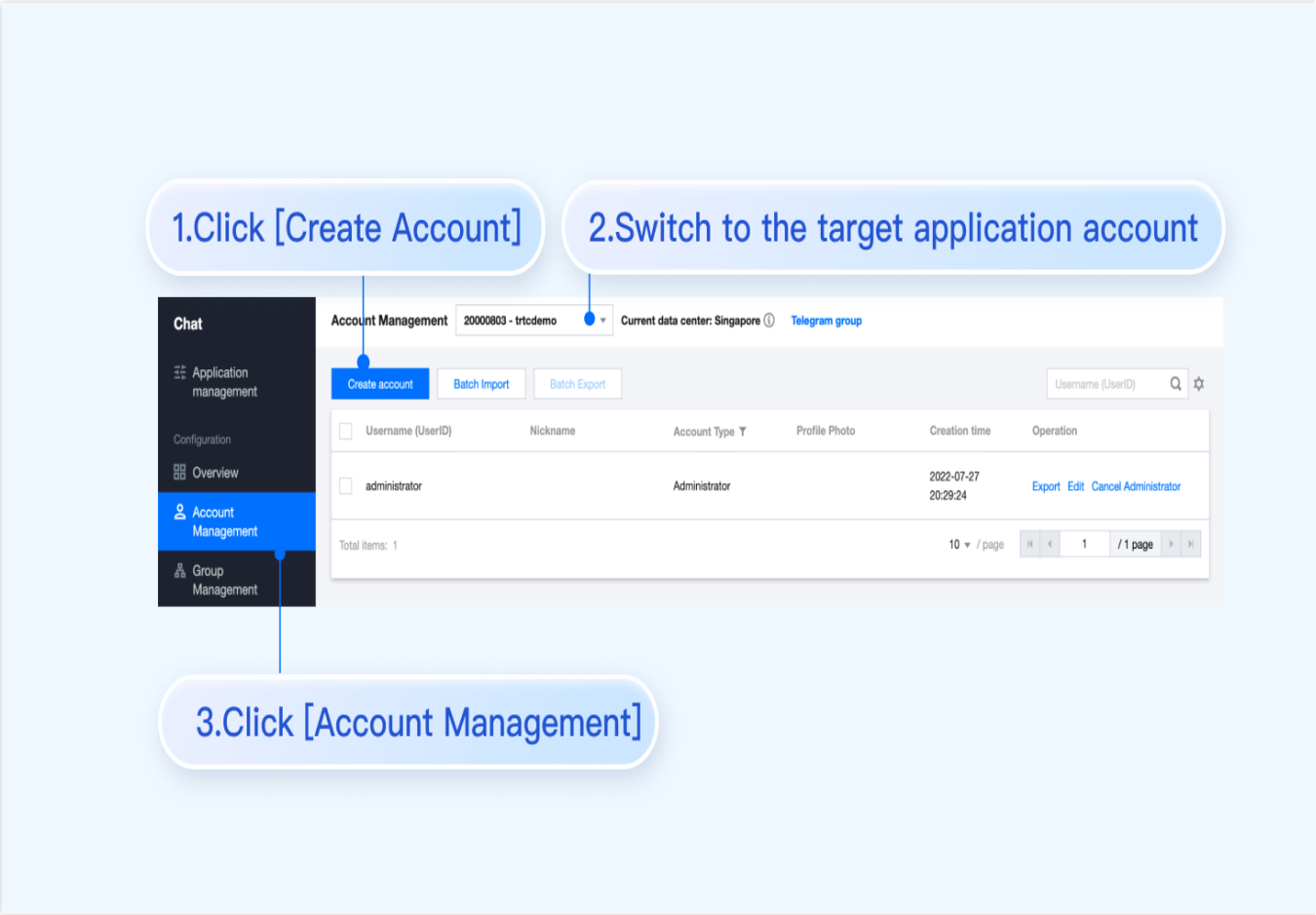


2. 如果 HBuilderX 没有自动拉起微信开发者工具，请使用微信开发者工具手动打开编译后的项目。
使用微信开发者工具打开项目根目录下的 `unpackage/dist/dev/mp-weixin` 即可。
3. 打开项目后，在微信开发者工具 “详情-本地设置” 中勾选 “不校验合法域名、web-view（业务域名）、TLS版本以及 HTTPS 证书”。

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。

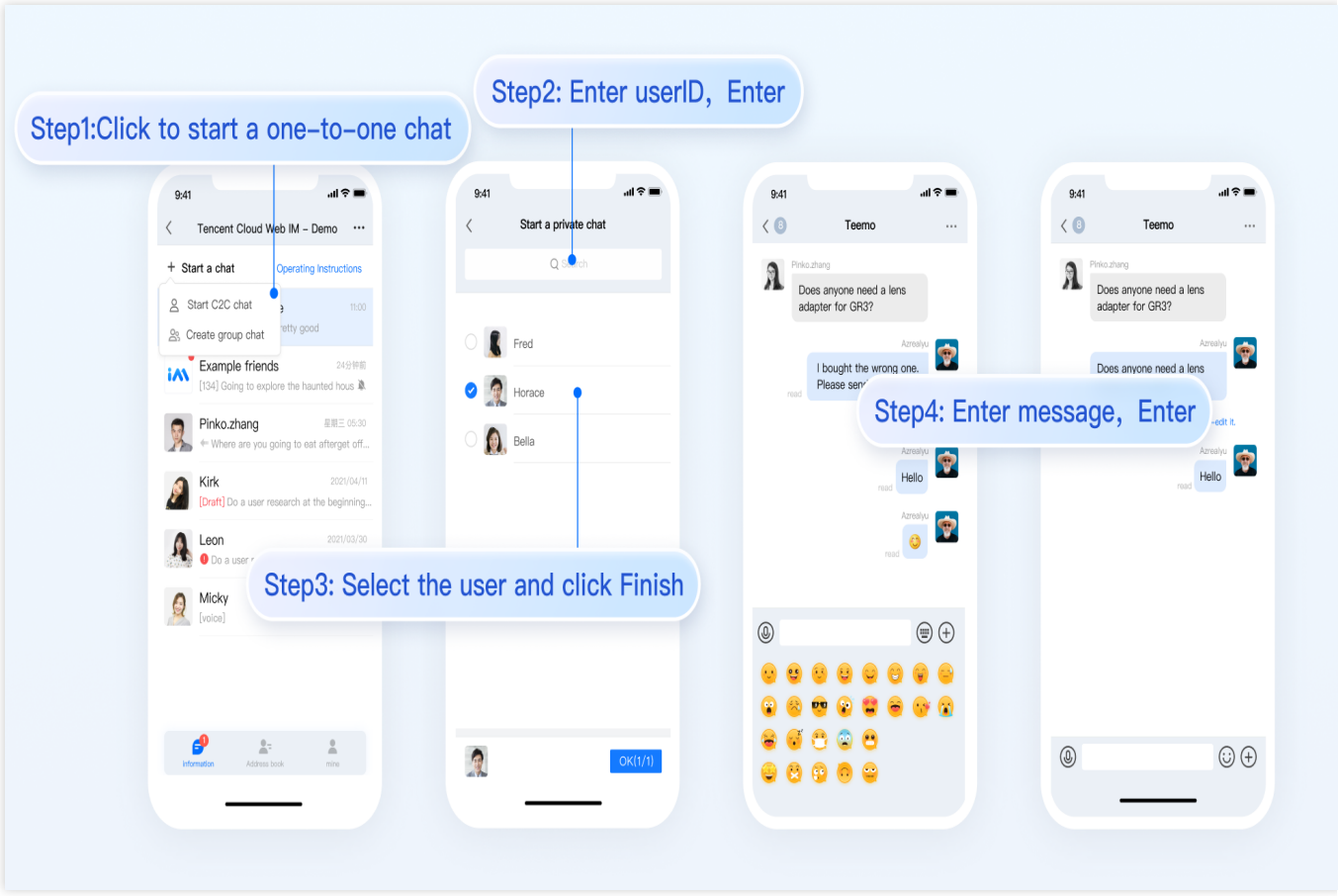
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

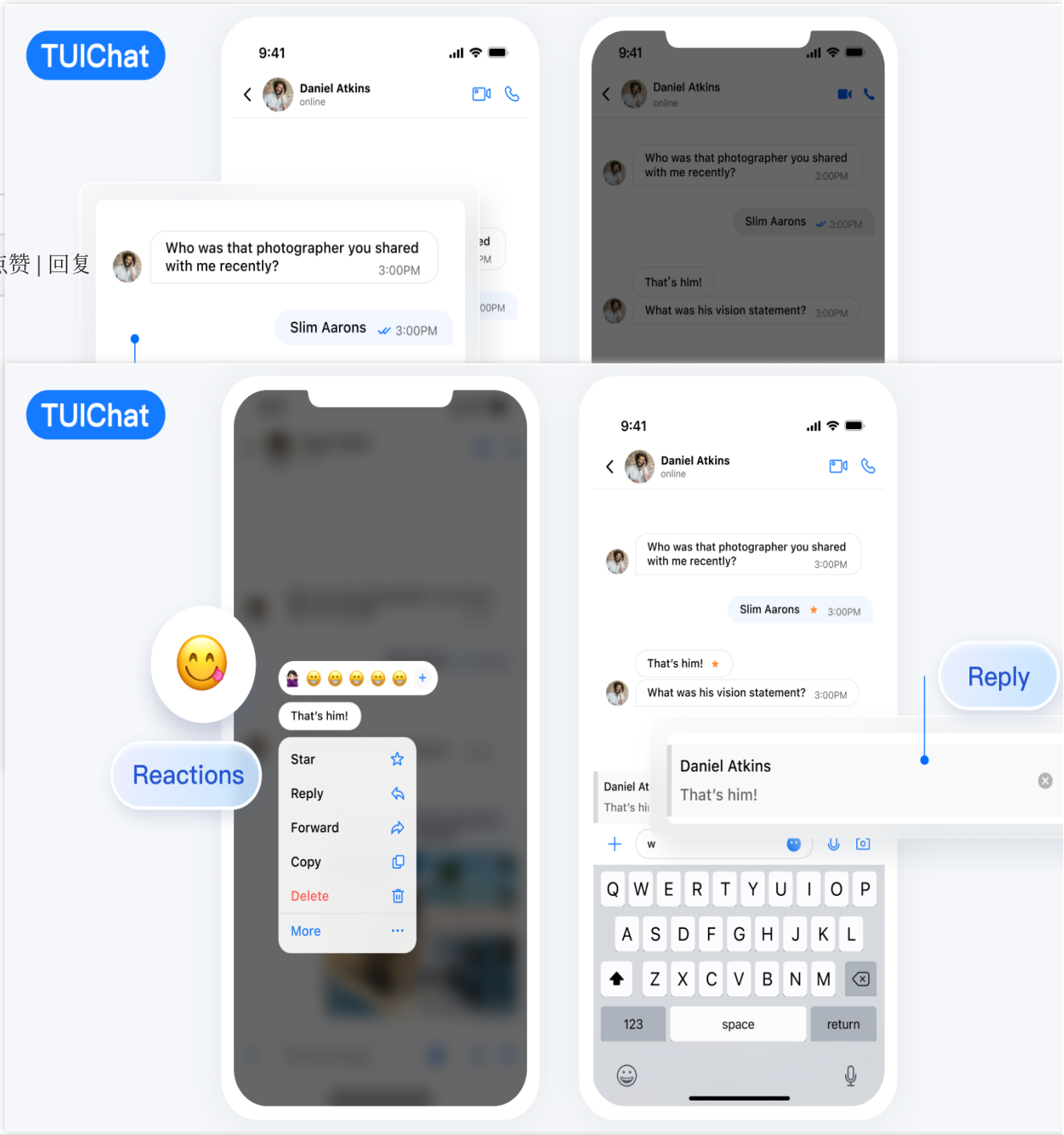
RTL语言

经典版

消息界面 | 发送多种类型消息

©2013-2025 Tencent Cloud International Pte. Ltd.

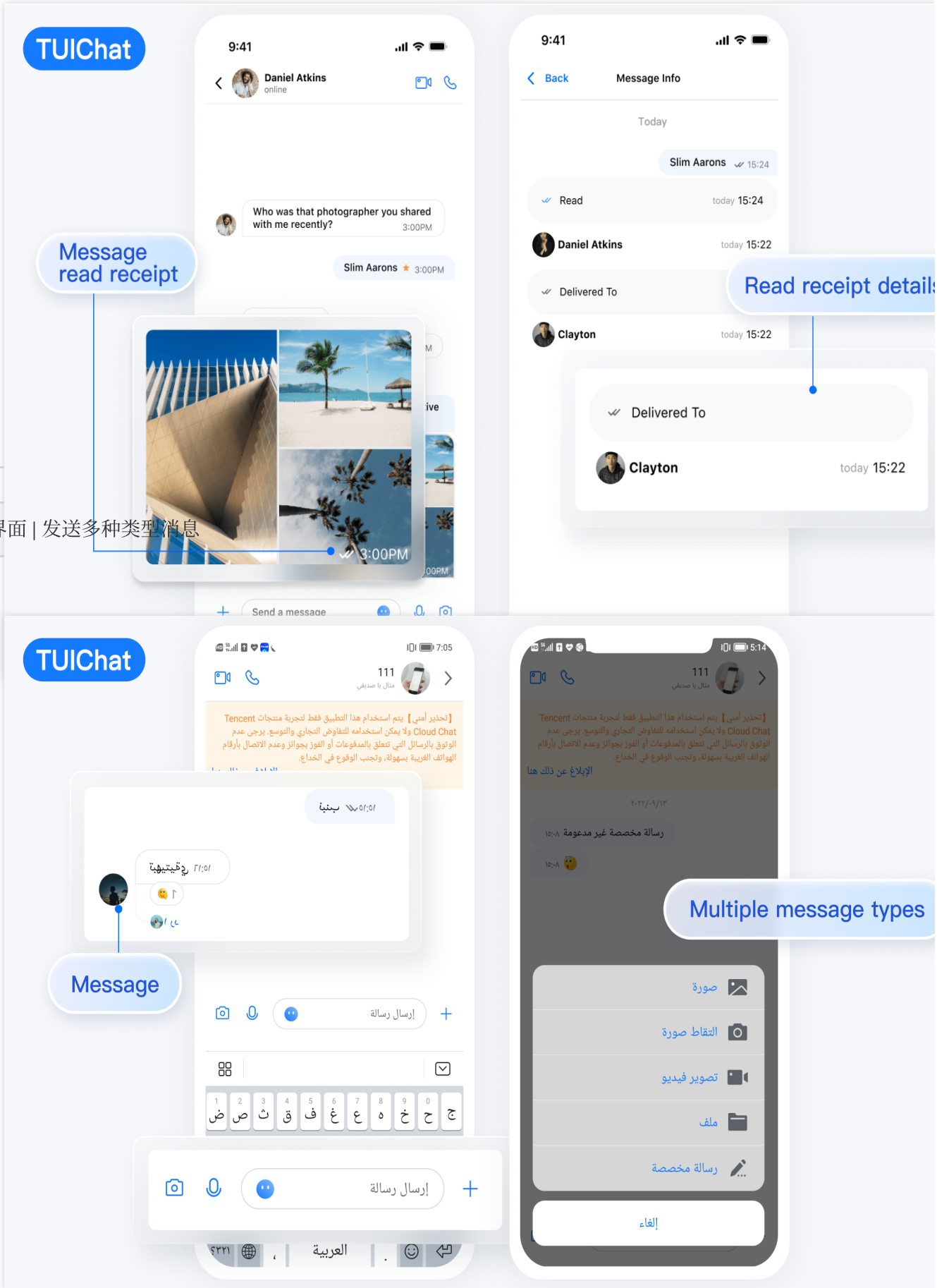
消息点赞 | 回复



消息已读回执 | 已读回执详情

©2013-2025 Tencent Cloud International Pte. Ltd.

消息界面 | 发送多种类型消息



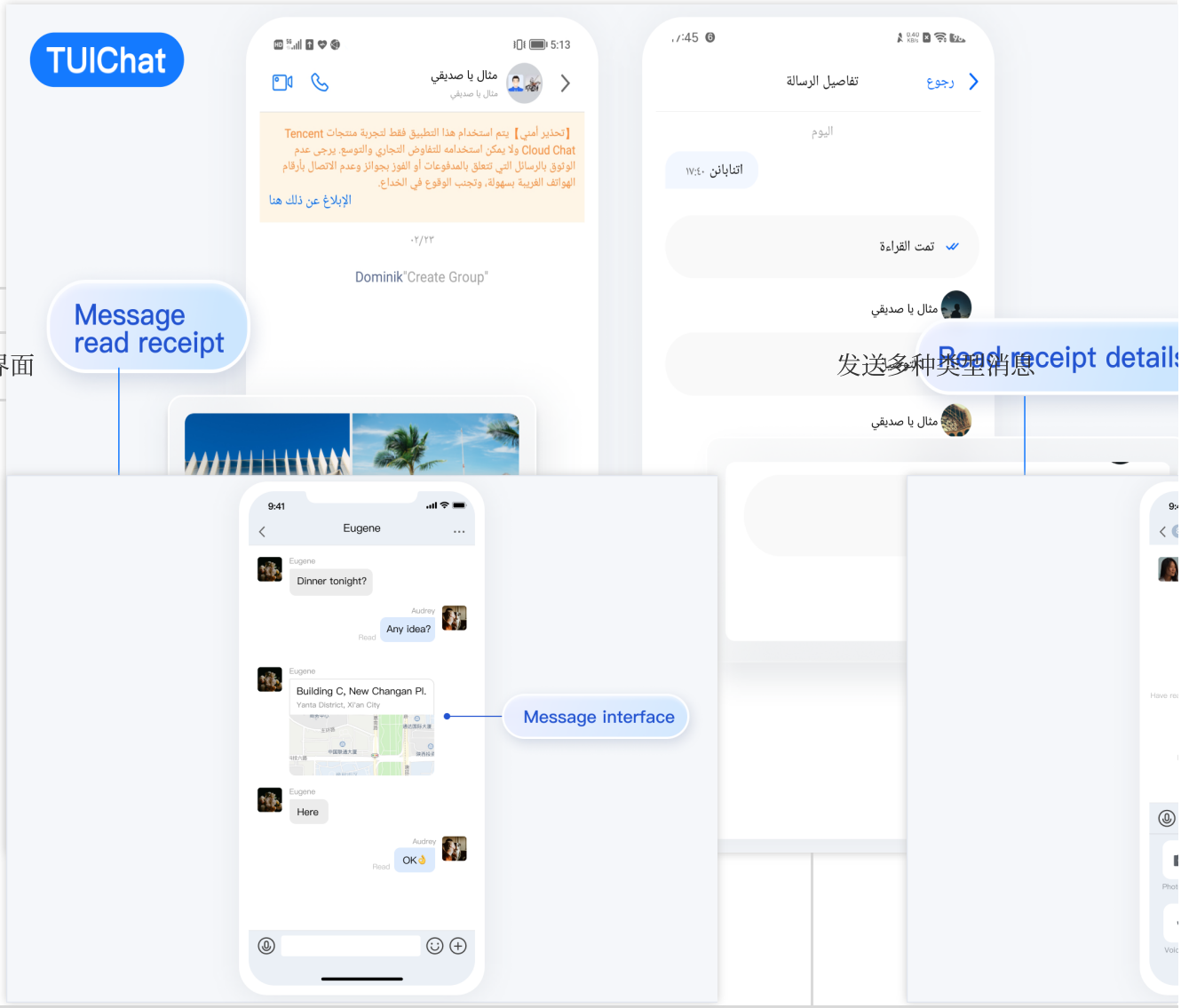
©2013-2025 Tencent Cloud International Pte. Ltd.

消息点赞 | 回复



消息已读回执 | 已读回执详情

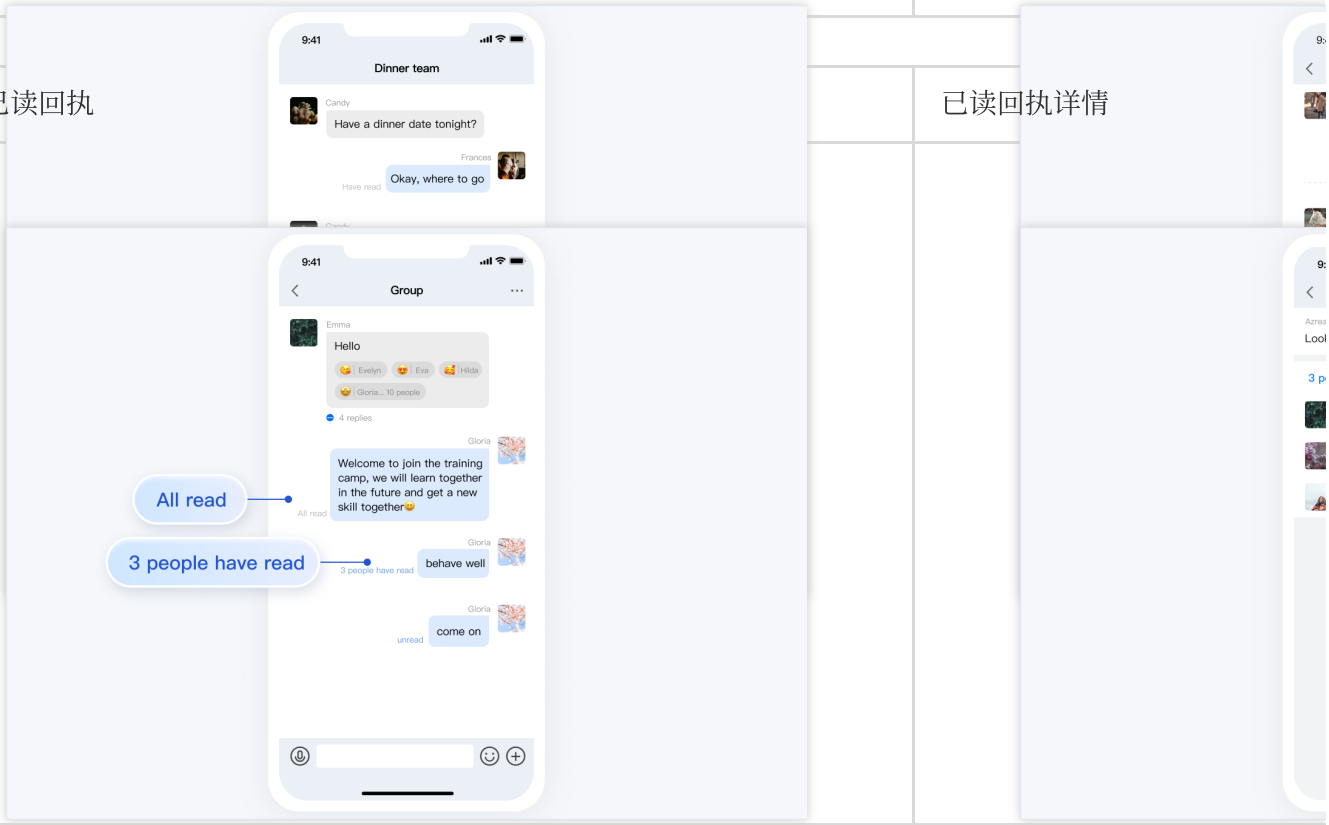
©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

消息已读回执	已读回执详情



开发环境要求

Android Studio-Giraffe

Gradle-7.2

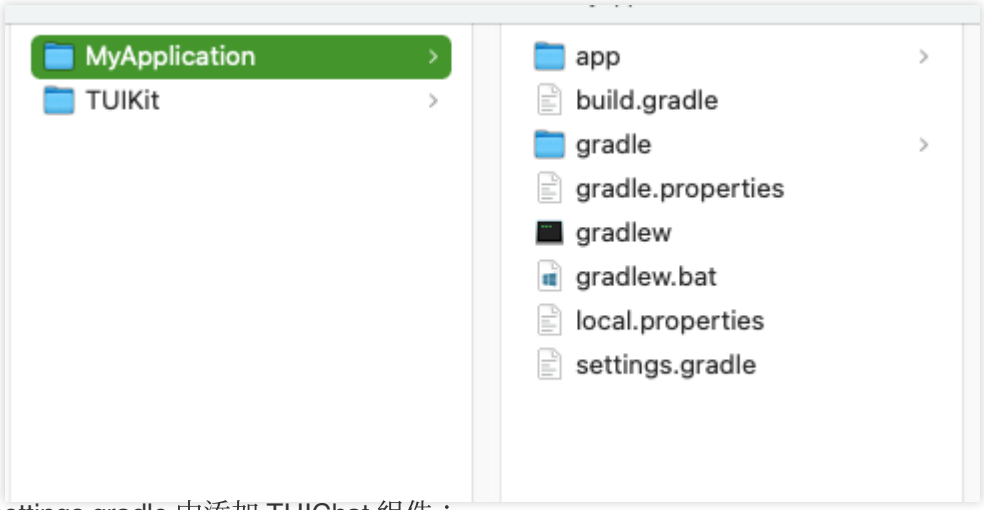
Android Gradle Plugin Version-7.0.0

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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" />
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
                <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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
# 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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

©2013-2025 Tencent Cloud International Pte. Ltd.

Message Read Receipt | Read Receipt Details

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

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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 TUIChat 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 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

Swift

Objective-C

```
# Uncomment the next line to define a global platform for your project
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
# 防止 TUIChat 组件里的 *.xcassets 与您项目里面冲突。
install! 'cocoapods', :disable_input_output_paths => true

# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
  use_frameworks!
  use_modular_headers!

  # 集成聊天功能
  pod 'TUIChat_Swift/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'] = ""
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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'
# 防止 TUIChat 组件里的 *.xcassets 与您项目里面冲突。
install! 'cocoapods', :disable_input_output_paths => true

# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
  use_frameworks!

  # 开启 modular headers。请按需开启，开启后 Pod 模块才能使用 @import 导入。
  # use_modular_headers!

  # 集成聊天功能
  pod 'TUIChat/UI_Minimalist'
end
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Swift

Objective-C

```
# Uncomment the next line to define a global platform for your project
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '13.0'
# 防止 TUIChat 组件里的 *.xcassets 与您项目里面冲突。
install! 'cocoapods', :disable_input_output_paths => true

# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
  use_frameworks!
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  use_modular_headers!

  # 集成聊天功能
  pod 'TUIChat_Swift/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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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'
# 防止 TUIChat 组件里的 *.xcassets 与您项目里面冲突。
install! 'cocoapods', :disable_input_output_paths => true
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
  use_frameworks!

  # 开启 modular headers。请按需开启，开启后 Pod 模块才能使用 @import 导入。
  # use_modular_headers!

  # 集成聊天功能
  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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
            end
        end
    end
end
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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` 。

Swift TUIKit 源码 Github 地址：

Objective-C TUIKit 源码 Github 地址：

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

Swift

Objective-C

```
# 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

# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  # Uncomment the next line if you're using Swift or would like to use dynamic
frameworks
  use_frameworks!
  use_modular_headers!

  # 集成基础库（必选）
  pod 'TUICore', :path => "../TUIKit/TUICore"
  pod 'TIMCommon_Swift', :path => "../TUIKit/TIMCommon"

  # 集成聊天功能
  pod 'TUIChat_Swift', :path => "../TUIKit/TUIChat"

  # 其他 Pod
  pod 'MJRefresh'
  pod 'SnapKit'
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`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                end
              end
              File.open(xcconfig_path, "w") { |file| file << xcconfig }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            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'
install! 'cocoapods', :disable_input_output_paths => true

# 请使用您的真实项目名称替换 your_project_name
target 'your_project_name' do
  # Uncomment the next line if you're using Swift or would like to use dynamic
frameworks
  use_frameworks!
  use_modular_headers!

  # 注意:使用本地集成方案时，如需升级时需要从
https://github.com/TencentCloud/TIMSDK/tree/master/iOS/TUIKit/TUIChat
  # 获取最新的组件代码，放置在本地指定目录下，如/TIMSDK/ios/TUIKit/TUIChat
  # 注意:当私有化修改和远端有冲突时，需要手动合并，处理冲突。

  # 集成基础库（必选）
  pod 'TUICore', :path => "../TUIKit/TUICore"
  pod 'TIMCommon', :path => "../TUIKit/TIMCommon"

  # 集成聊天功能
  pod 'TUIChat', :path => "../TUIKit/TUIChat"

  # 其他 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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            xcode_version = `xcrun xcodebuild -version | grep Xcode | cut -d' '
-f2`.to_f
            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 =
$(inherited)")
                end
                if xcconfig.include?("-ld64") == false
                  xcconfig = xcconfig.sub("OTHER_LDFLAGS = $(inherited)",
'OTHER_LDFLAGS = $(inherited) "-ld64"')
                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，如需升级时可从 Github 更新 TUIChat 到本地升级。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
    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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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)

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

React

最近更新时间：2024-11-22 11:55:28

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 i @tencentcloud/chat-uikit-react @tencentcloud/uikit-base-component-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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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 from 'react';
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import { Chat, ChatHeader, MessageList, MessageInput } from '@tencentcloud/chat-uik

import '@tencentcloud/chat-uikit-react/dist/esm/index.css';

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

TUILogin.login({
  ...config,
  useUploadPlugin: true
}).then(() => {
  openDefaultChat();
}).catch(() => {});

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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
export default function App() {
  // language support en-US(default) / zh-CN / ja-JP / ko-KR / zh-TW
  // theme support light(default) / dark
  return (
    <div style={{display: 'flex', height: '100vh'}}>
      <UIKitProvider language='en-US' theme='light'>
        <Chat>
          <ChatHeader />
          <MessageList />
          <MessageInput />
        </Chat>
      </UIKitProvider>
    </div>
  );
}
```

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/src/** ./src/TUIKit
```

```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-react\\src\\ .\\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 from 'react';
import { TUILogin } from '@tencentcloud/tui-core';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import { Chat, ChatHeader, MessageList, MessageInput } from './TUIKit';

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

TUILogin.login({
  ...config,
  useUploadPlugin: true
}).then(() => {
  openDefaultChat();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}).catch(() => {});

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

export default function App() {
  // language support en-US(default) / zh-CN / ja-JP / ko-KR / zh-TW
  // theme support light(default) / dark
  return (
    <div style={{display: 'flex', height: '100vh'}}>
      <UIKitProvider language='en-US' theme='light'>
        <Chat>
          <ChatHeader />
          <MessageList />
          <MessageInput />
        </Chat>
      </UIKitProvider>
    </div>
  );
}
```

3. 若运行报错，可参考修改以下配置，支持 create-react-app 源码导入。

3.1 安装 sass 参考 create-react-app 文档

3.2 关闭 tsconfig.json 中的 `isolatedModules`

3.3 配置 eslint 以关闭对 src/TUIKit 源码的检查

3.4 修改文件 src/TUIKit/styles/fonts/icon-font.scss 中的字体资源路径，相对定位到 `../../assets/fonts/*`

3.5 配置 webpack.config.js 中主题颜色导入的别名路径 `resolve.alias`

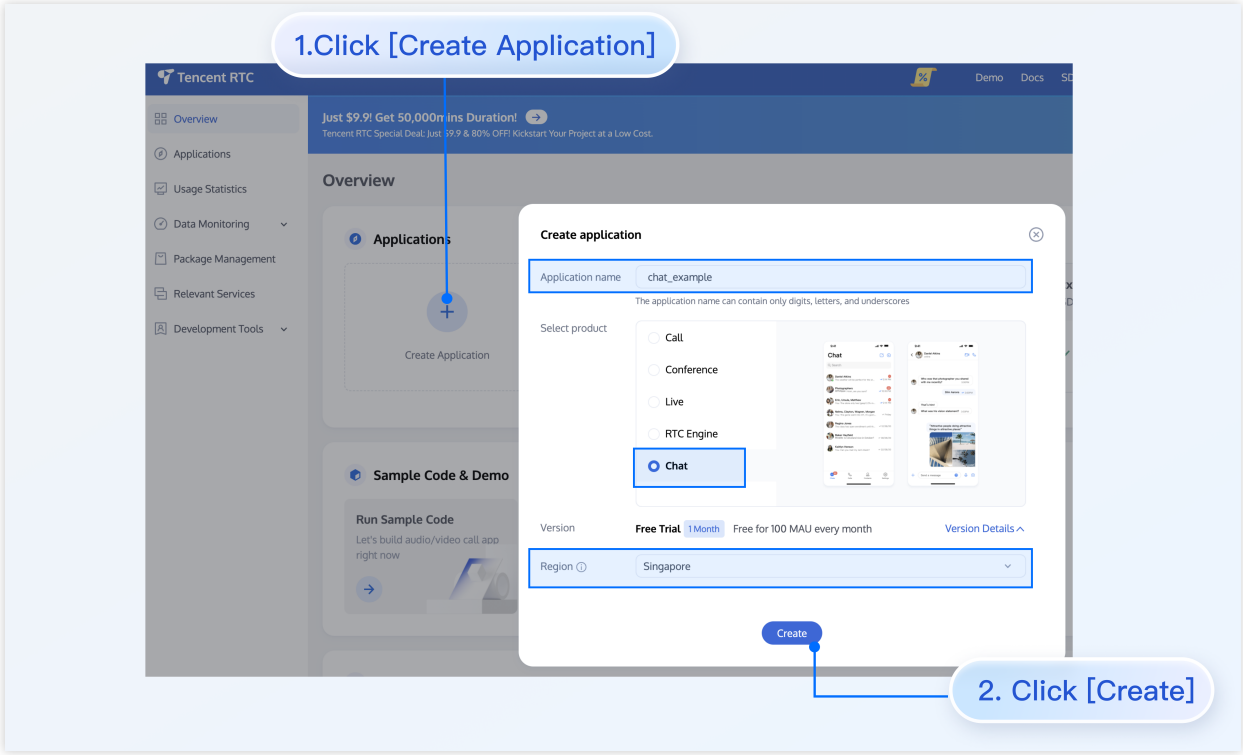
```
module.exports = {
    resolve: {
      alias: {
        '~@tencentcloud/uikit-base-component-react/dist/styles/theme/util': path.re
      }
    }
}
```

步骤4：创建一个应用

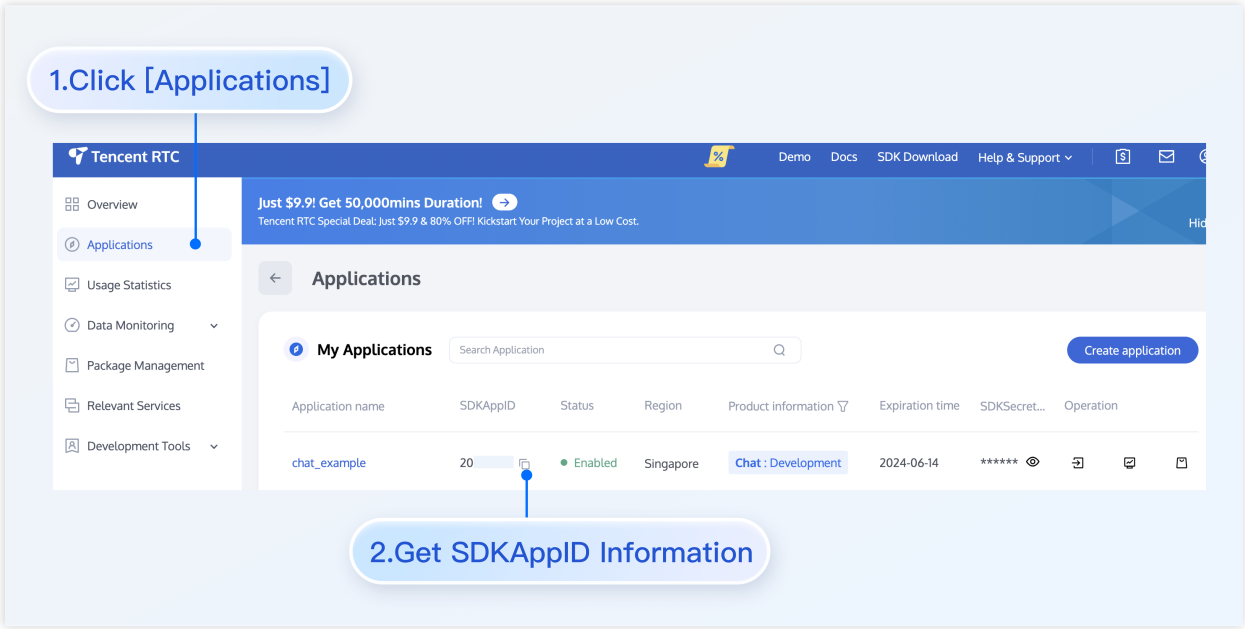
1. 登录 Chat Console 。

2. 单击“Create Application”，输入您的应用程序名称，然后单击“Create”。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

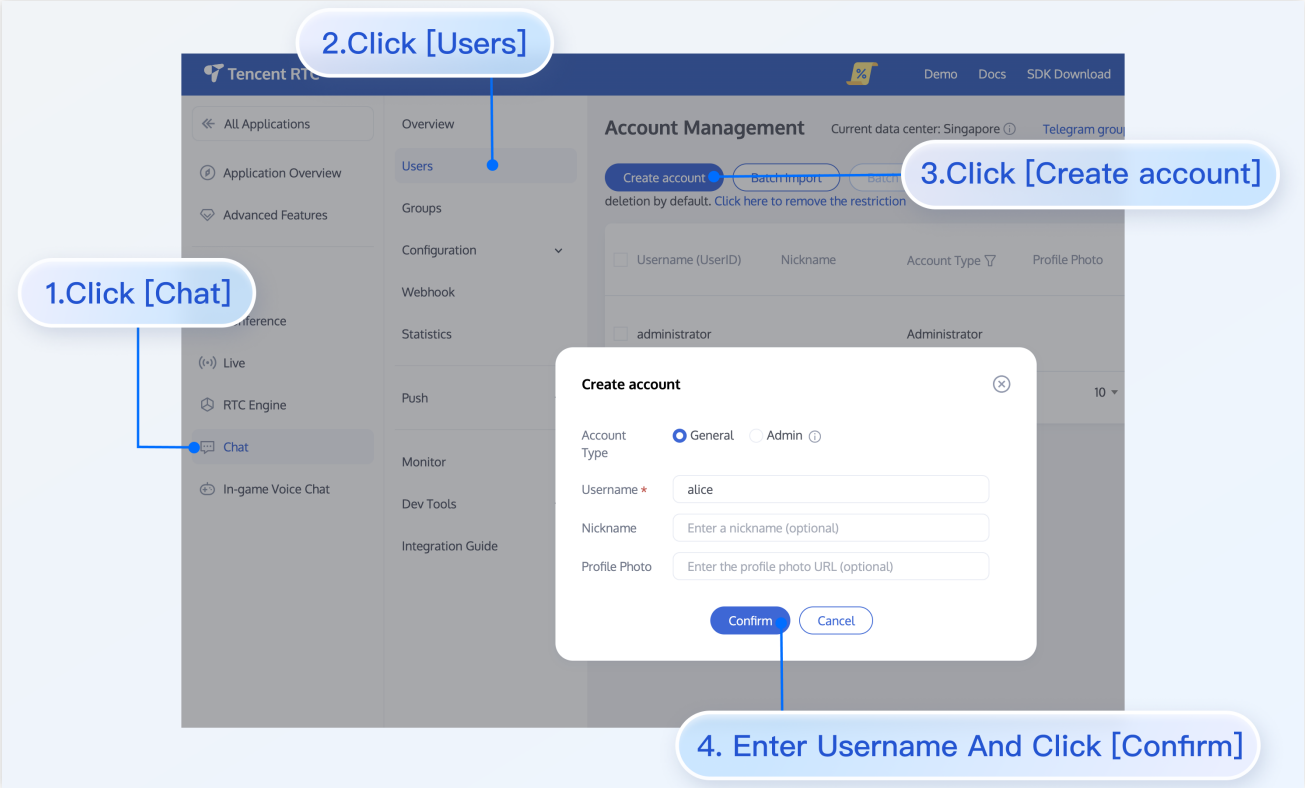


步骤5：获取 userID 和 userSig

`userID`

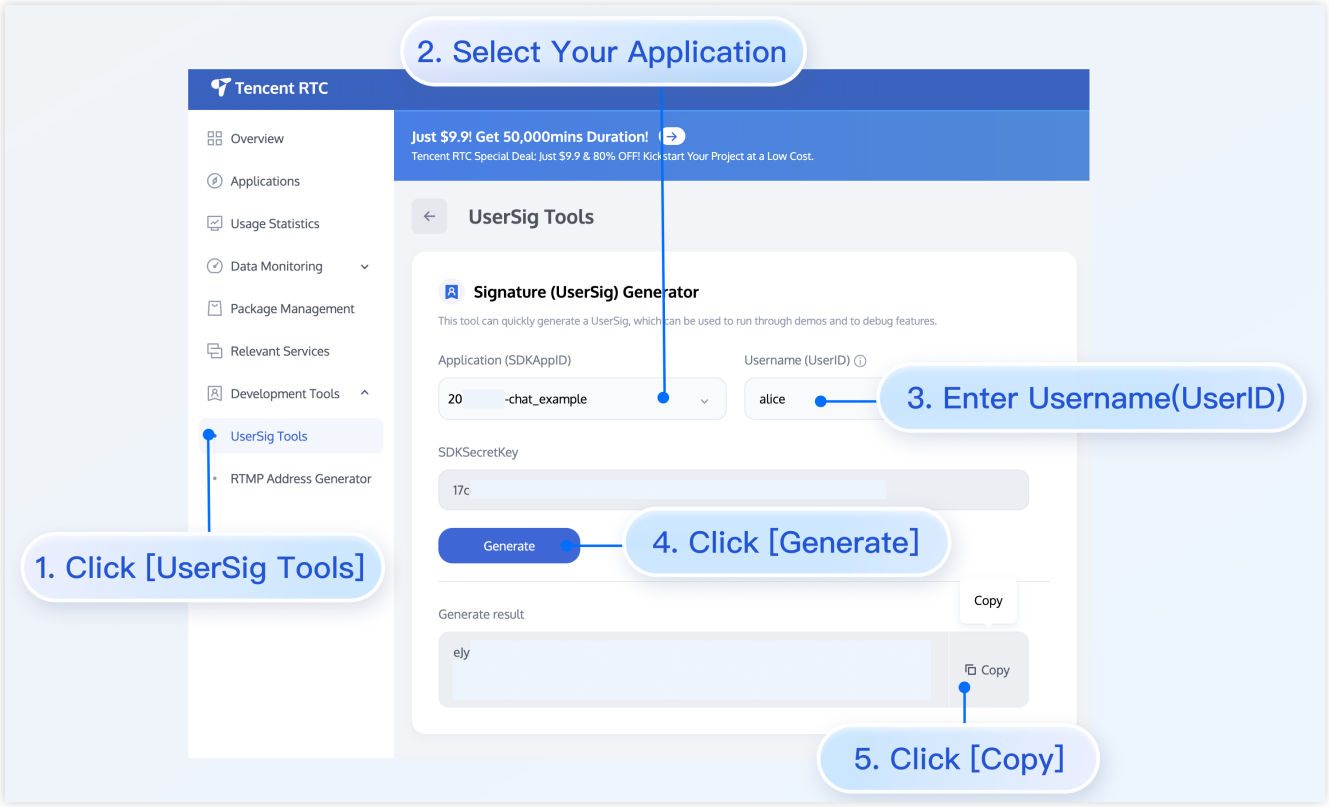
©2013-2025 Tencent Cloud International Pte. Ltd.

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 。

©2013-2025 Tencent Cloud International Pte. Ltd.



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？

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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）

©2013-2025 Tencent Cloud International Pte. Ltd.

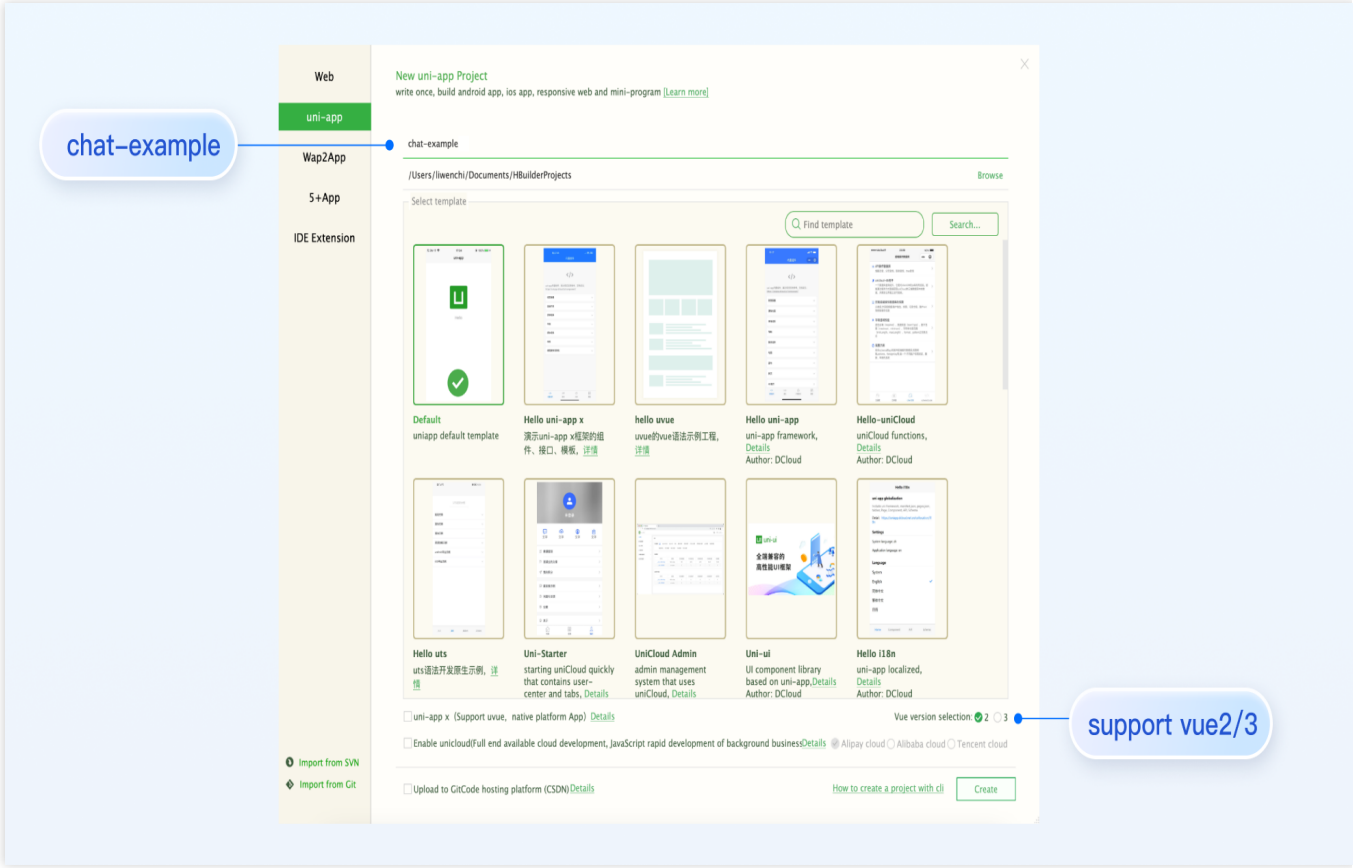
node（12.13.0 ≤ node 版本 ≤ 17.0.0, 推荐使用 Node.js 官方 LTS 版本 16.17.0）
npm（版本请与 node 版本匹配）

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

©2013-2025 Tencent Cloud International Pte. Ltd.

Windows 端

通过 npm 方式下载 TUIKit 组件：

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

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

```
mkdir -p ./TUIKit && rsync -av --exclude=
{'node_modules','package.json','excluded-list.txt'}
./node_modules/@tencentcloud/chat-uikit-uniapp/ ./TUIKit
```

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

通过 npm 方式下载 TUIKit 组件：

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

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

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

```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin
.\\TUIKit\\tui-customer-service-plugin /i /e
```

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 */
      }),
    ],
  },
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  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);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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",
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    "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"
 }
}
```

```
<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();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// #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";
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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/product/269/32688
    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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

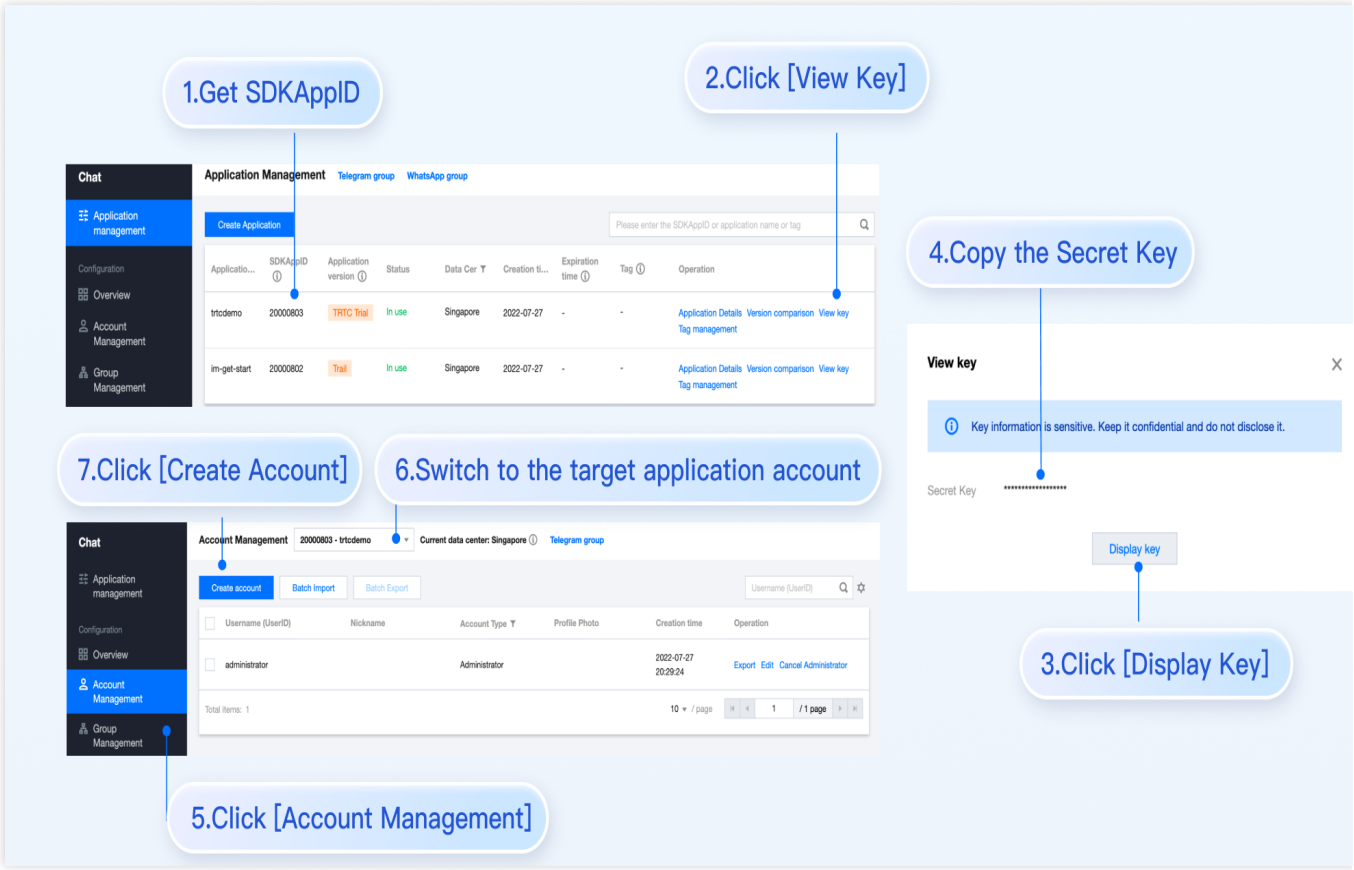
创建 userID 为 test-user1 的账户

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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";
```

©2013-2025 Tencent Cloud International Pte. Ltd.

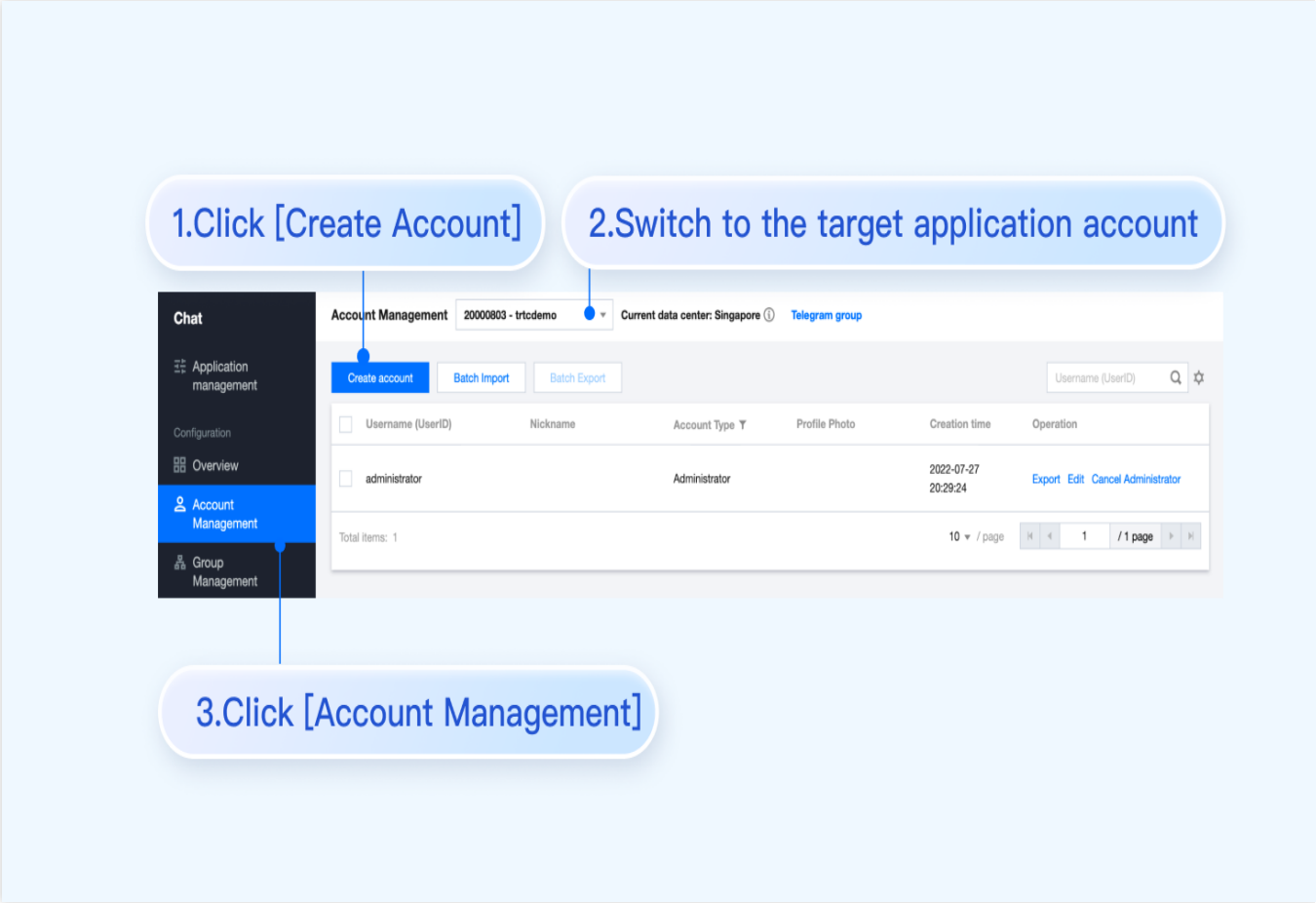
```
export default {
  components: {},
  data() {
    return {
      userID: "test-user2", // 请填写对端 userID 参考步骤6
      groupID: "", // 通过调用 API createGroup 可获取 groupID，具体可参考：https://web.s
    };
  },
  methods: {
    // 打开 1v1 聊天
    openChat() {
      // 切换会话进入聊天
      const conversationID = `C2C${this.userID}`;
      // #ifdef APP-PLUS || H5
      TUIConversationService.switchConversation(conversationID);
      // #endif
      uni.navigateTo({
        url: `/TUIKit/components/TUIChat/index?conversationID=${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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    text-align: center;
  }
}
</style>
```

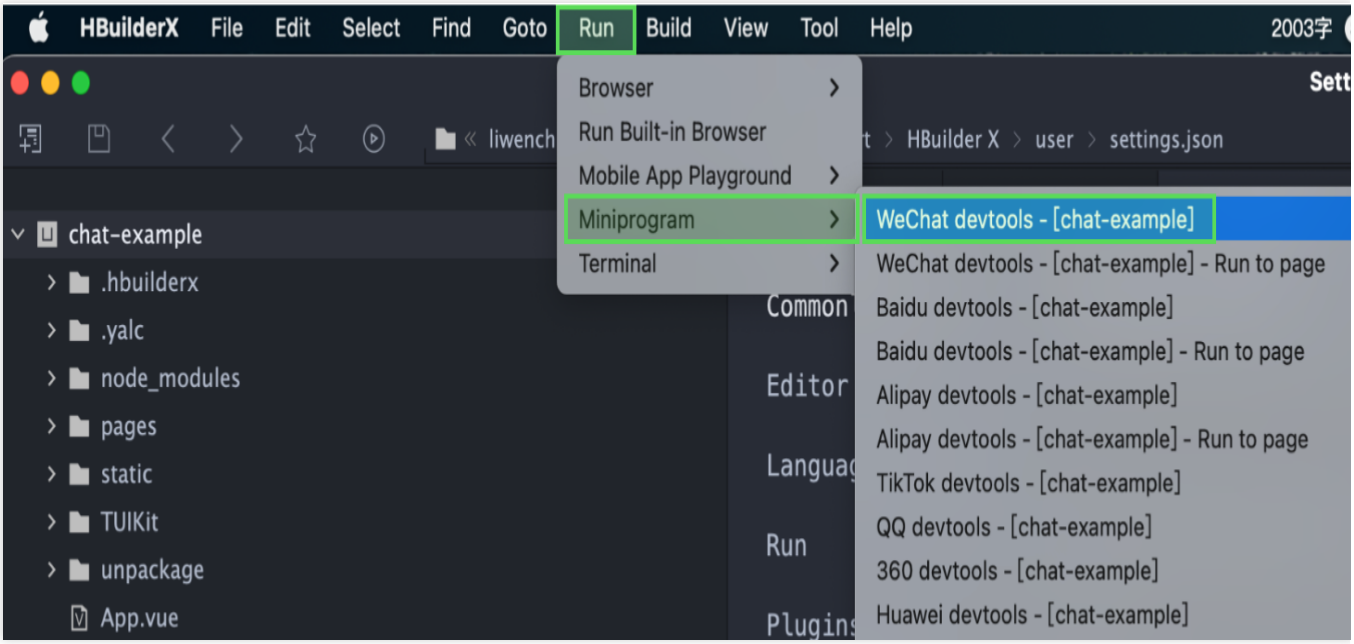
步骤6：发送您的第一条消息

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**。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

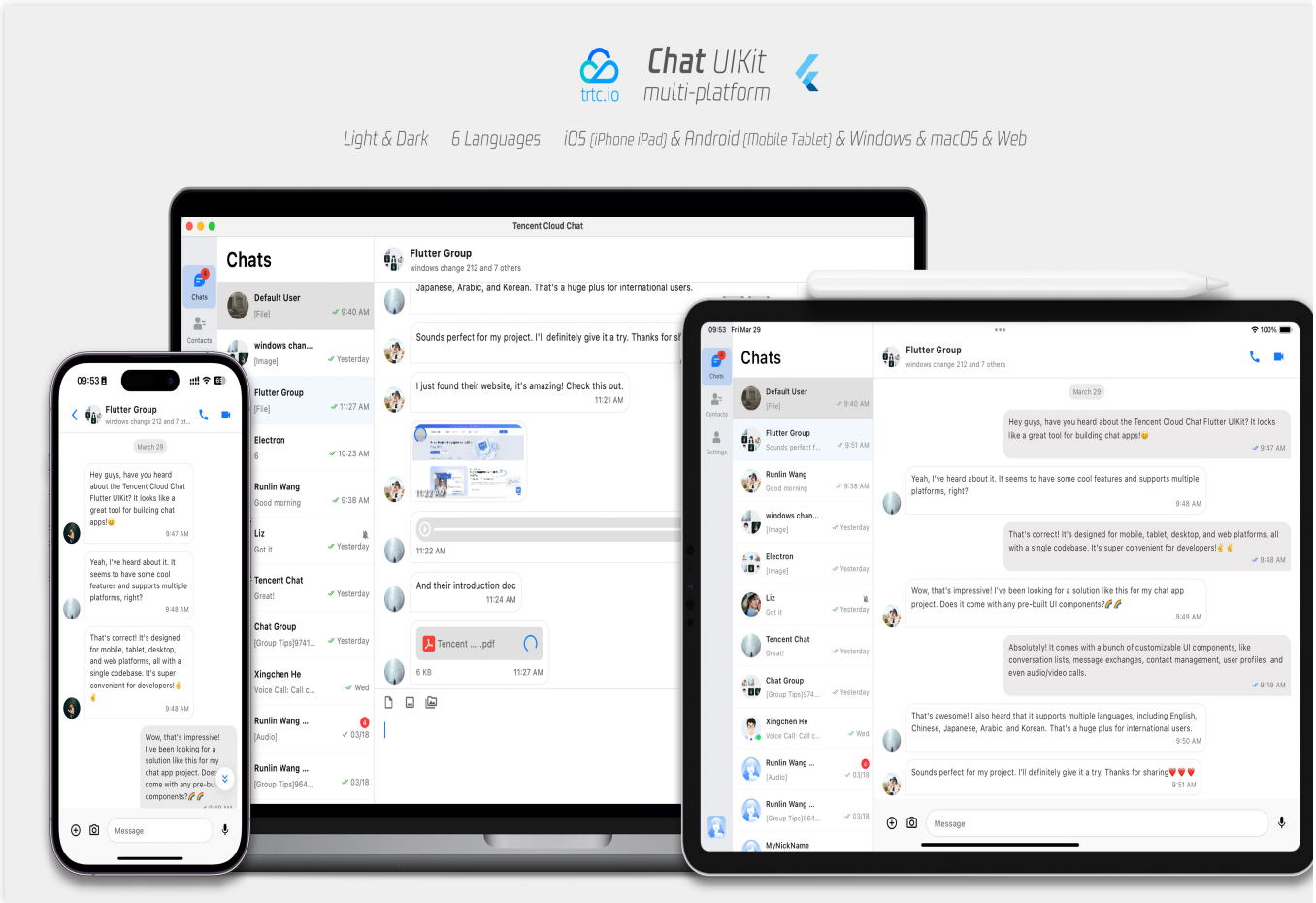
ChatEngine npm

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-03-17 15:35:32

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：

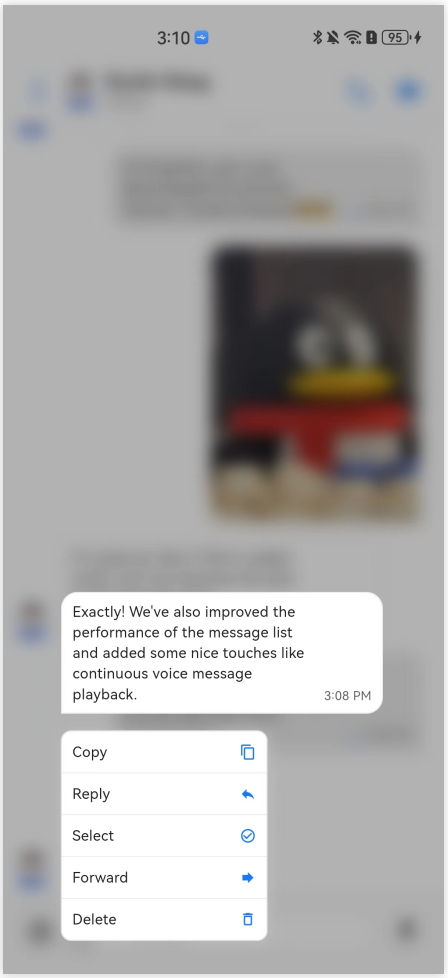


手机端
电脑端 & 平板端

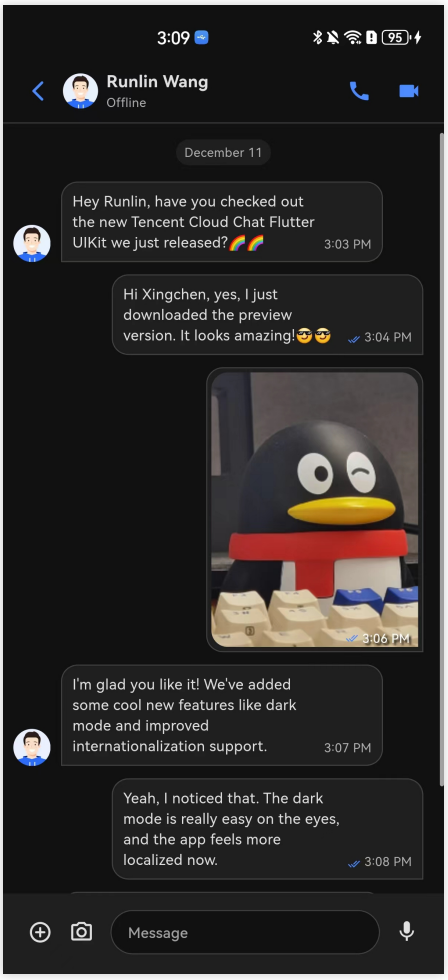
©2013-2025 Tencent Cloud International Pte. Ltd.



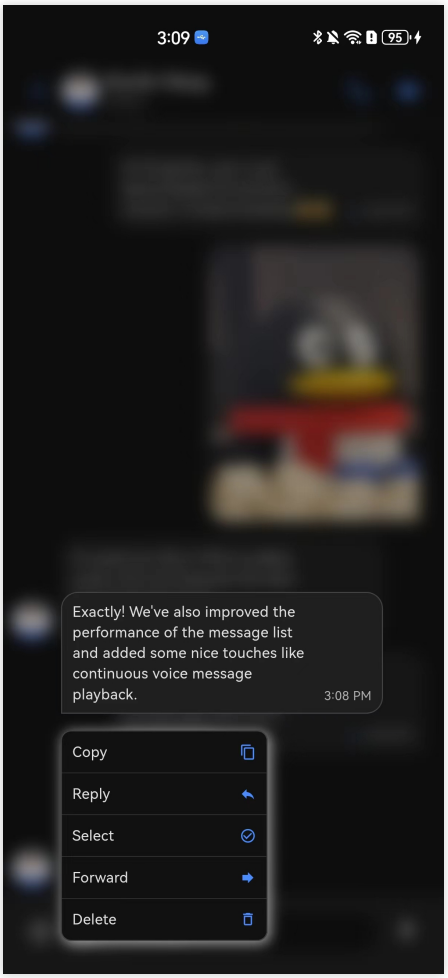
©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.

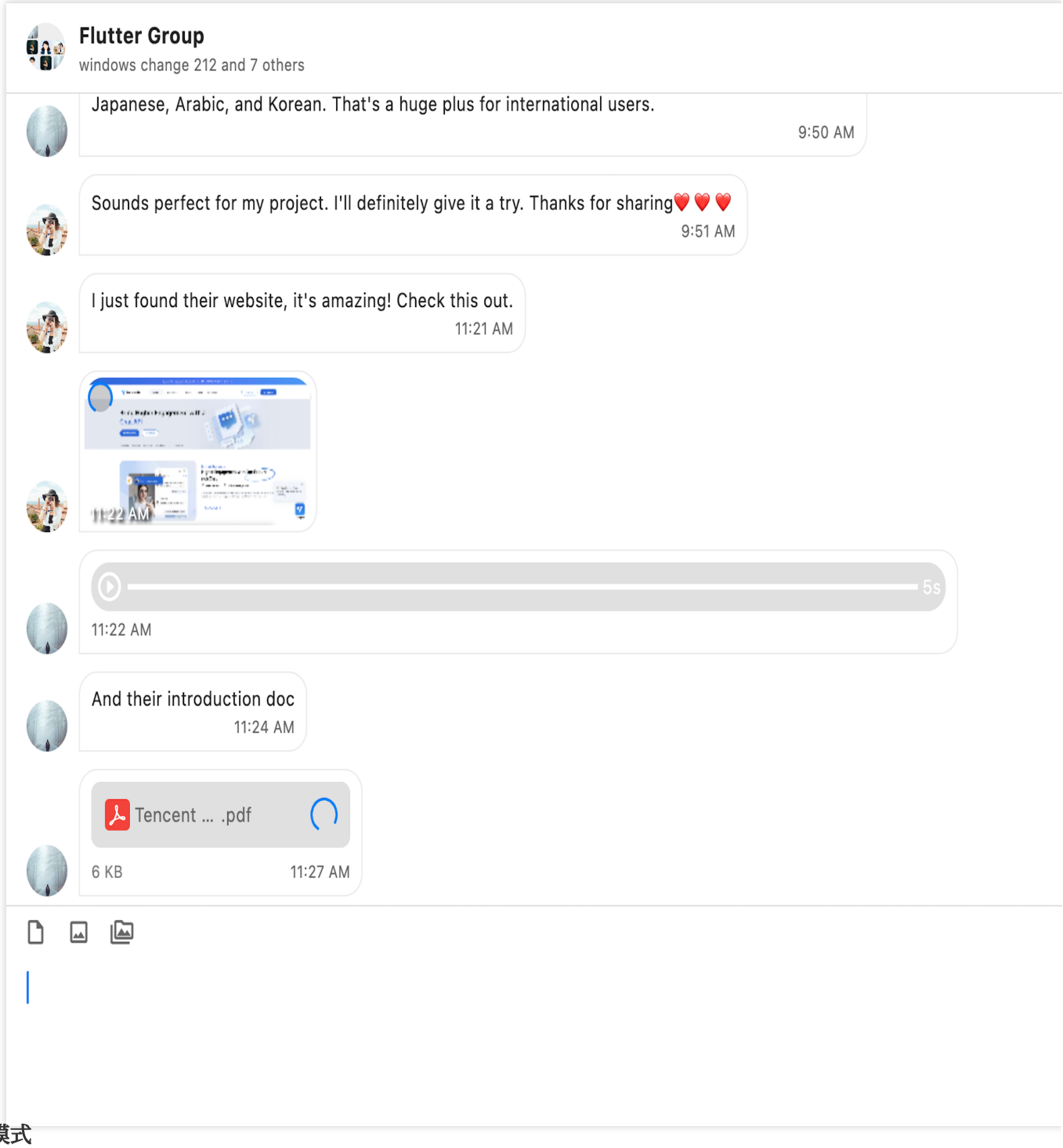


©2013-2025 Tencent Cloud International Pte. Ltd.



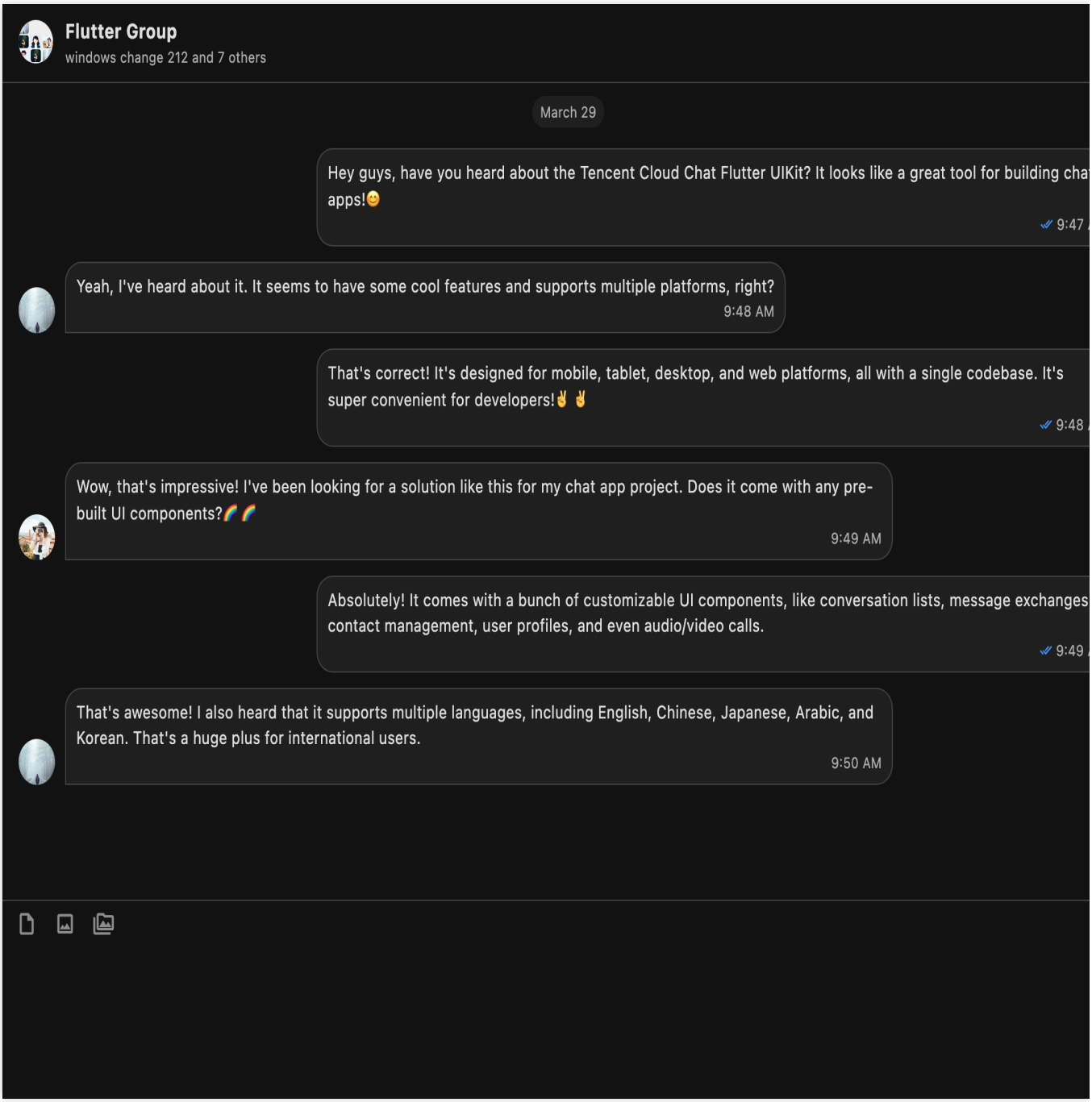
日间模式

©2013-2025 Tencent Cloud International Pte. Ltd.



黑暗模式

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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：3.24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_common。

```
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 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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 Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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>?`

©2013-2025 Tencent Cloud International Pte. Ltd.

`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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

```
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

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）。

©2013-2025 Tencent Cloud International Pte. Ltd.

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**。

©2013-2025 Tencent Cloud International Pte. Ltd.

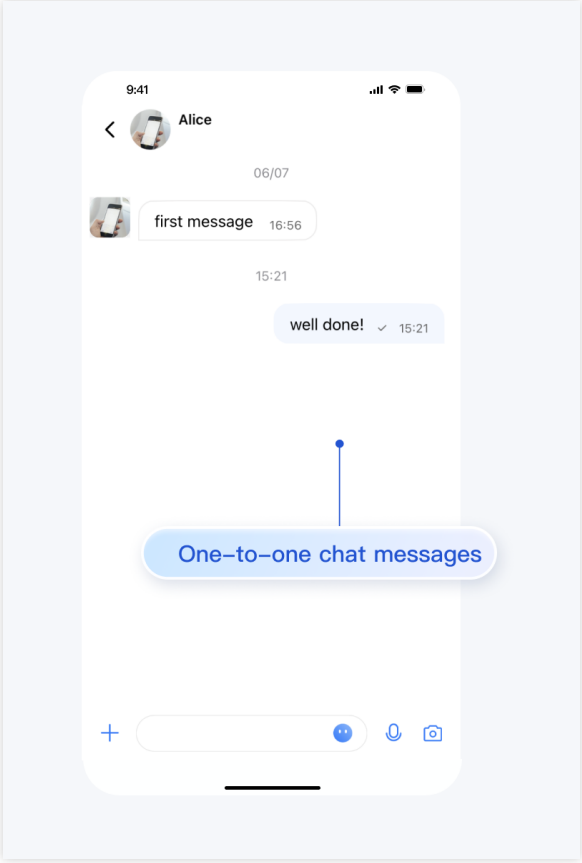
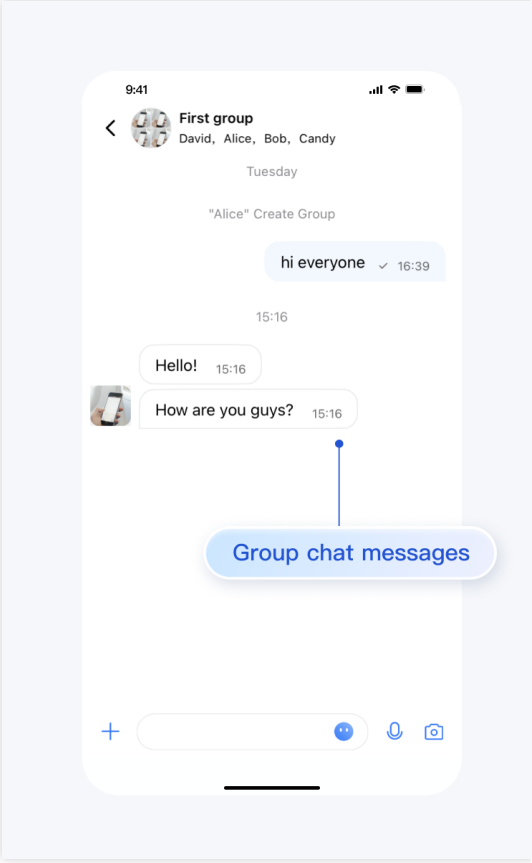
构建基础界面
聊天界面
Android

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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");
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
intent.putExtra(TUIConstants.TUIChat.CHAT_TYPE, isGroup ? V2TIMConversation.V2TIM_G
startActivity(intent);
```

```
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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
} 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-06-20 16:13:28

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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
Swift
Objective-C

```
import UIKit

// ChatViewController is your own ViewController
class ChatViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // Create conversation data.
        let conversationData = TUIChatConversationModel()
        // Pass userID for 1v1 chat, while groupID for group chat.
        conversationData.userID = "userID"
        conversationData.groupID = "groupID"

        // Create chatVC by groupID or userID.
        var chatVC: TUIBaseChatViewController_Minimalist?

        if let groupID = conversationData.groupID, !groupID.isEmpty {
            chatVC = TUIGroupChatViewController_Minimalist()
        } else if let userID = conversationData.userID, !userID.isEmpty {
            chatVC = TUIC2CChatViewController_Minimalist()
        }

        chatVC?.conversationData = conversationData

        // Option 1: push chatVC.
        navigationController?.pushViewController(chatVC!, animated: true)
        // Option 2: add chatVC to your own ViewController.
        // addChild(chatVC!)
        // view.addSubview(chatVC!.view)
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
```

```
#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
```

Swift

Objective-C

```
import UIKit

// ChatViewController is your own ViewController
class ChatViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // Create conversation data.
        let conversationData = TUIChatConversationModel()
        // Pass userID for 1v1 chat, while groupID for group chat.
        conversationData.userID = "userID"
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        conversationData.groupID = "groupID"

        // Create chatVC by groupID or userID.
        var chatVC: TUIBaseChatViewController?

        if let groupID = conversationData.groupID, !groupID.isEmpty {
            chatVC = TUIGroupChatViewController()
        } else if let userID = conversationData.userID, !userID.isEmpty {
            chatVC = TUIC2CChatViewController()
        }

        chatVC?.setConversationData(conversationData)

        // Option 1: push chatVC.
        navigationController?.pushViewController(chatVC!, animated: true)
        // Option 2: add chatVC to your own ViewController.
        // addChild(chatVC!)
        // view.addSubview(chatVC!.view)
    }
}
```

```
#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.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    // [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。

©2013-2025 Tencent Cloud International Pte. Ltd.

会话列表

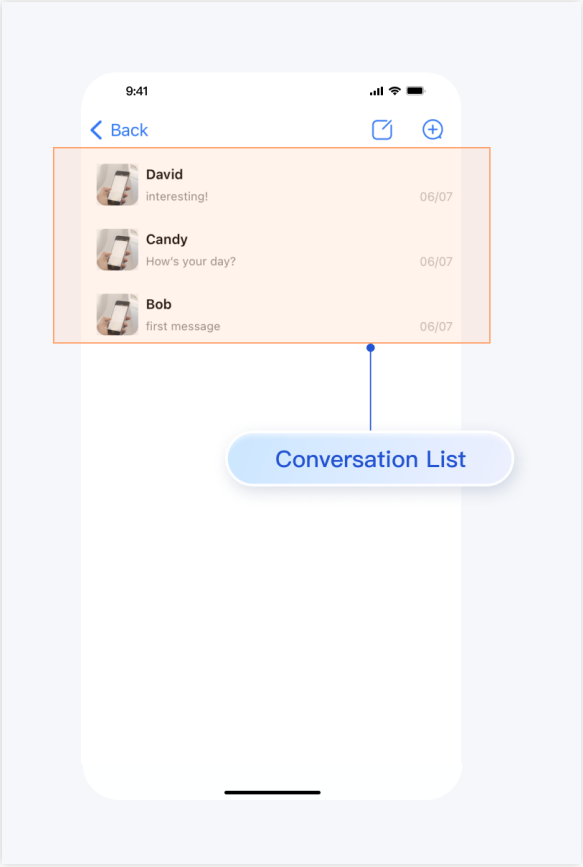
Android

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Swift

Objective-C

```
import UIKit

// ConversationController is your own ViewController
class ConversationController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // TUIConversationListController_Minimalist
        let vc = TUIConversationListController_Minimalist()
        vc.delegate = self

        // Option 1: push vc.
        navigationController?.pushViewController(vc, animated: true)
        // Option 2: Add TUIConversationListController_Minimalist to your own ViewC
        // addChild(vc)
        // view.addSubview(vc.view)
    }
}

extension ConversationController: TUIConversationListControllerListener {
    func conversationListController(_ conversationController: UIViewController, did
 {
        // Conversation list click event, typically, opening the chat UI
        let conversationData = TUIChatConversationModel()
        conversationData.userID = conversation.userID
        conversationData.title = conversation.title
        conversationData.faceUrl = conversation.faceUrl

        // Create chatVC by groupID or userID.
        var chatVC: TUIBaseChatViewController_Minimalist?

        if let groupID = conversationData.groupID, !groupID.isEmpty {
            chatVC = TUIGroupChatViewController_Minimalist()
        } else if let userID = conversationData.userID, !userID.isEmpty {
            chatVC = TUIC2CChatViewController_Minimalist()
        }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        chatVC?.conversationData = conversationData

        // Option 1: push chatVC.
        navigationController?.pushViewController(chatVC!, animated: true)
        // Option 2: add chatVC to your own ViewController.
        // addChild(chatVC!)
        // view.addSubview(chatVC!.view)
    }
}
```

```
#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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    // 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
```

Swift

Objective-C

```
import UIKit

// ConversationController is your own ViewController
class ConversationController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // TUIConversationListController
        let vc = TUIConversationListController()
        vc.delegate = self

        // Option 1: push vc.
        navigationController?.pushViewController(vc, animated: true)
        // Option 2: Add TUIConversationListController to your own ViewController
        // addChild(vc)
        // view.addSubview(vc.view)
    }
}

extension ConversationController: TUIConversationListControllerListener {
    func conversationListController(_ conversationController: UIViewController, did
 {
        // Conversation list click event, typically, opening the chat UI
        let conversationData = TUIChatConversationModel()
        conversationData.userID = conversation.userID
        conversationData.title = conversation.title
        conversationData.faceUrl = conversation.faceUrl

        // Create chatVC by groupID or userID.
        var chatVC: TUIBaseChatViewController?

        if let groupID = conversationData.groupID, !groupID.isEmpty {
            chatVC = TUIGroupChatViewController()
        } else if let userID = conversationData.userID, !userID.isEmpty {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            chatVC = TUIC2CChatViewController()
        }

        chatVC?.conversationData = conversationData

        // Option 1: push chatVC.
        navigationController?.pushViewController(chatVC!, animated: true)
        // Option 2: add chatVC to your own ViewController.
        // addChild(chatVC!)
        // view.addSubview(chatVC!.view)
    }
}
```

```
#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) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        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。

©2013-2025 Tencent Cloud International Pte. Ltd.

联系人界面

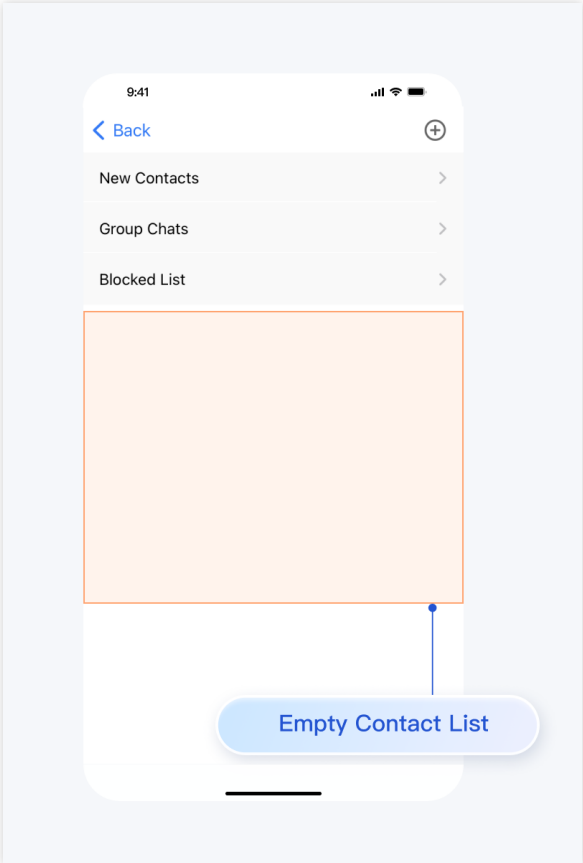
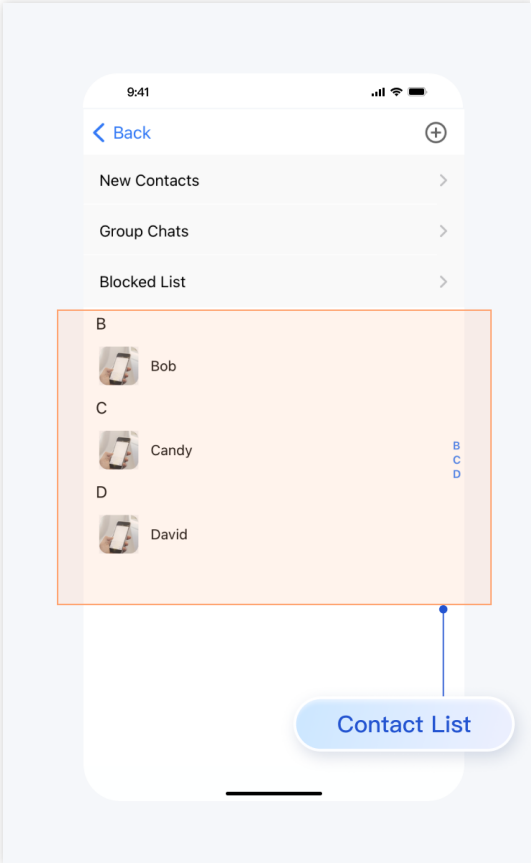
Android

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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 {
```

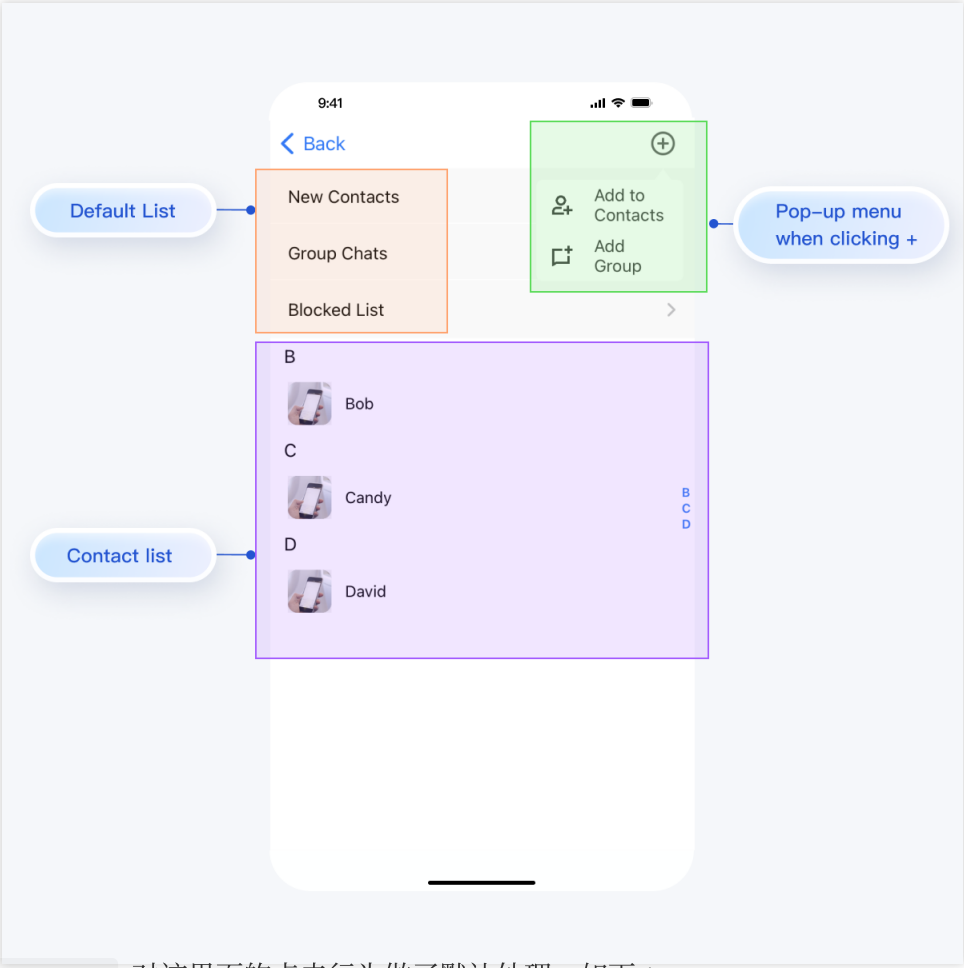
©2013-2025 Tencent Cloud International Pte. Ltd.

```
    @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：

©2013-2025 Tencent Cloud International Pte. Ltd.



`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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Swift

Objective-C

```
import UIKit

// ContactController is your own ViewController
class ContactController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // Create TUIContactController_Minimalist
        let vc = TUIContactController_Minimalist()

        // Option 1: push vc.
        navigationController?.pushViewController(vc, animated: true)
        // Option 2: add vc to your own ViewController.
        // addChild(vc)
        // view.addSubview(vc.view)
    }
}
```

```
#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
```

Swift

Objective-C

```
import UIKit
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// ContactController is your own ViewController
class ContactController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // Create TUIContactController
        let vc = TUIContactController()

        // Option 1: push vc.
        navigationController?.pushViewController(vc, animated: true)
        // Option 2: add vc to your own ViewController.
        // addChild(vc)
        // view.addSubview(vc.view)
    }
}
```

```
#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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Swift

Objective-C

```
protocol TUIContactControllerListener_Minimalist: AnyObject {
    func onSelectFriend(_ cell: TUICommonContactCell_Minimalist) -> Bool
    func onAddNewFriend(_ cell: TUICommonTableViewCell) -> Bool
    func onGroupConversation(_ cell: TUICommonTableViewCell) -> Bool
}
```

```
@protocol TUIContactControllerListener_Minimalist <NSObject>
@optional
- (void)onSelectFriend:(TUICommonContactCell *)cell;
- (void)onAddNewFriend:(TUICommonTableViewCell *)cell;
- (void)onGroupConversation:(TUICommonTableViewCell *)cell;
@end
```

Swift

Objective-C

```
protocol TUIContactControllerListener: NSObjectProtocol {
    func onSelectFriend(_ cell: TUICommonContactCell) -> Bool
    func onAddNewFriend(_ cell: TUICommonTableViewCell) -> Bool
    func onGroupConversation(_ cell: TUICommonTableViewCell) -> Bool
}
```

```
@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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

添加联系人

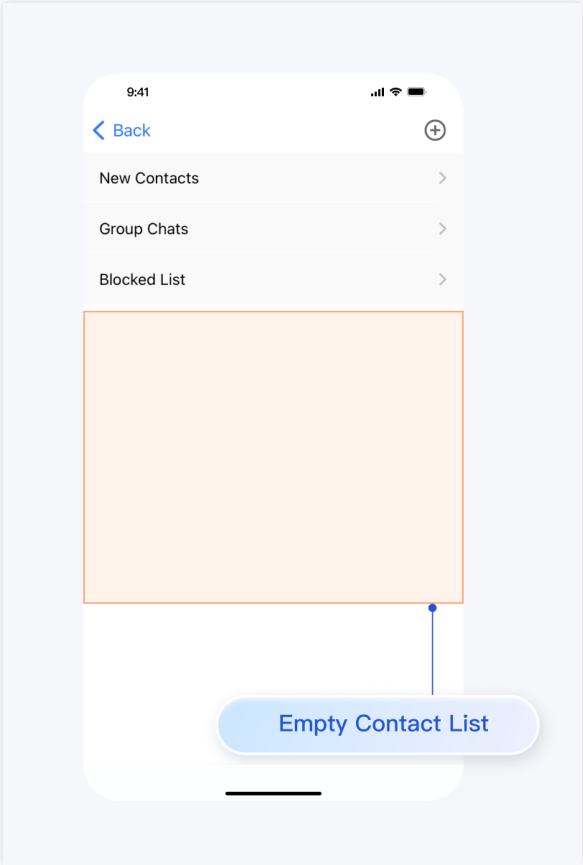
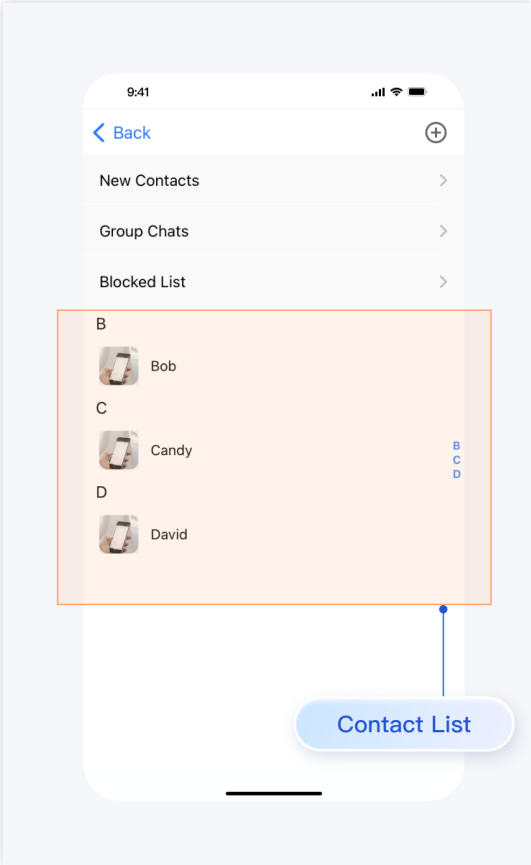
Android

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

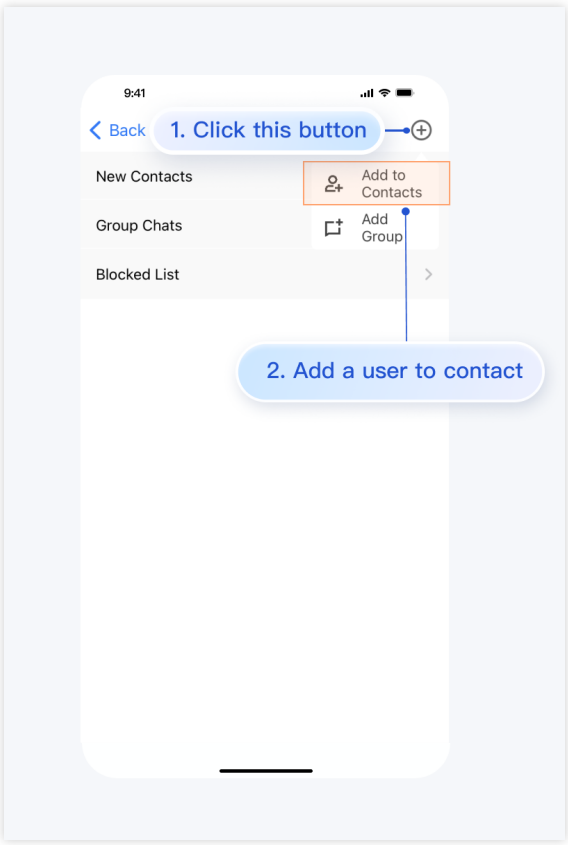
注意：
1. 所有组件都是这个登录接口。每次启动应用，登录一次即可。
2. 请确保登录成功，我们建议您在登录成功的回调里进行下文的操作。
如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

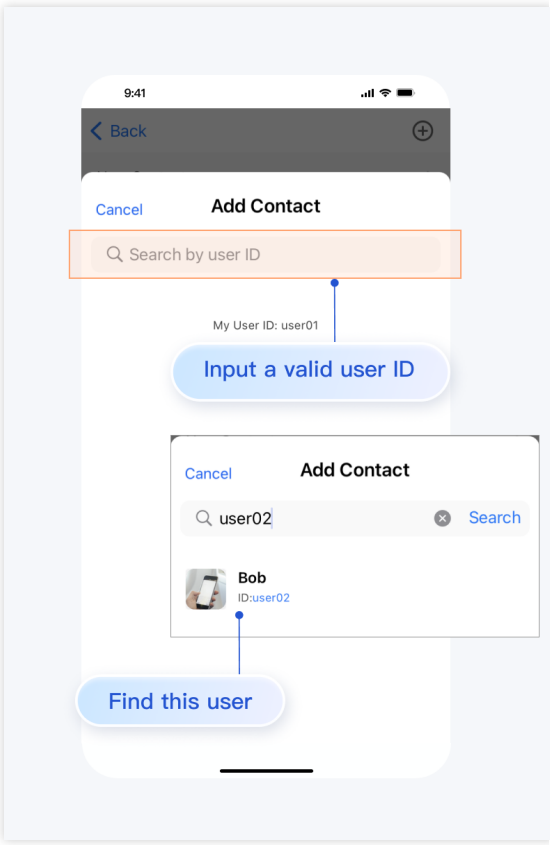
步骤说明

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

©2013-2025 Tencent Cloud International Pte. Ltd.





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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

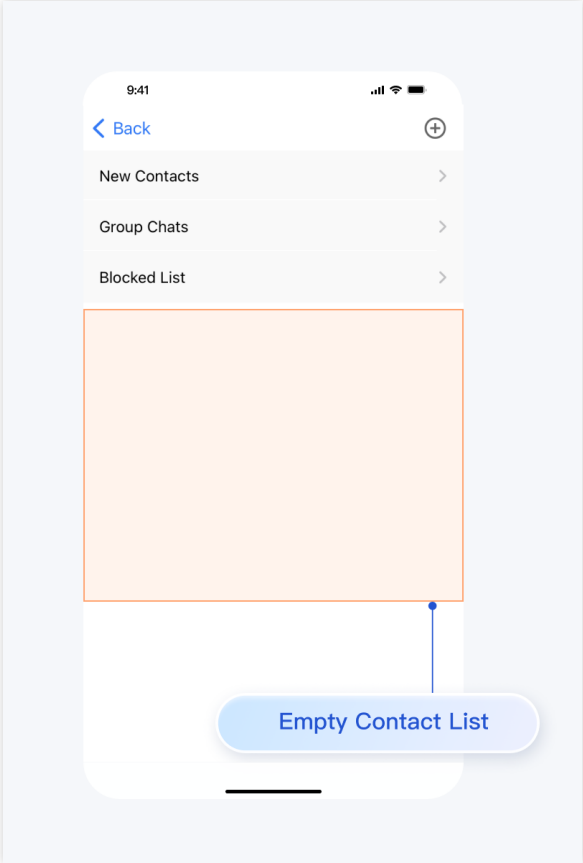
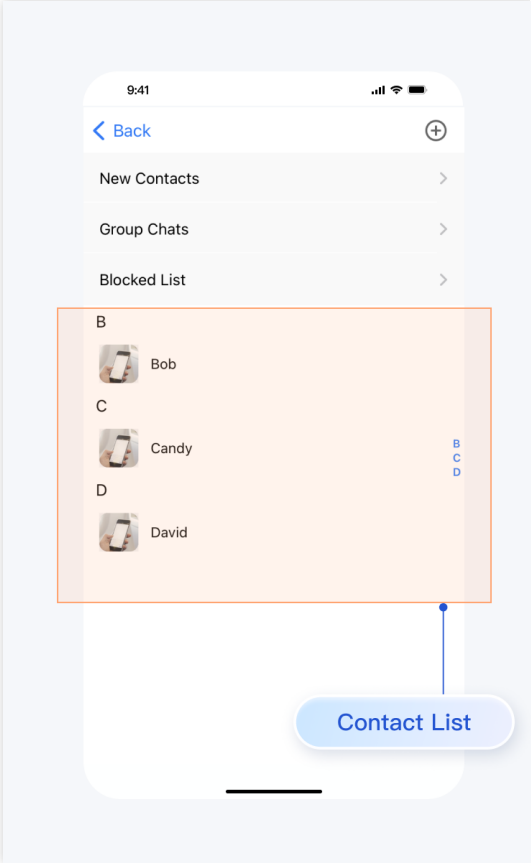
iOS

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

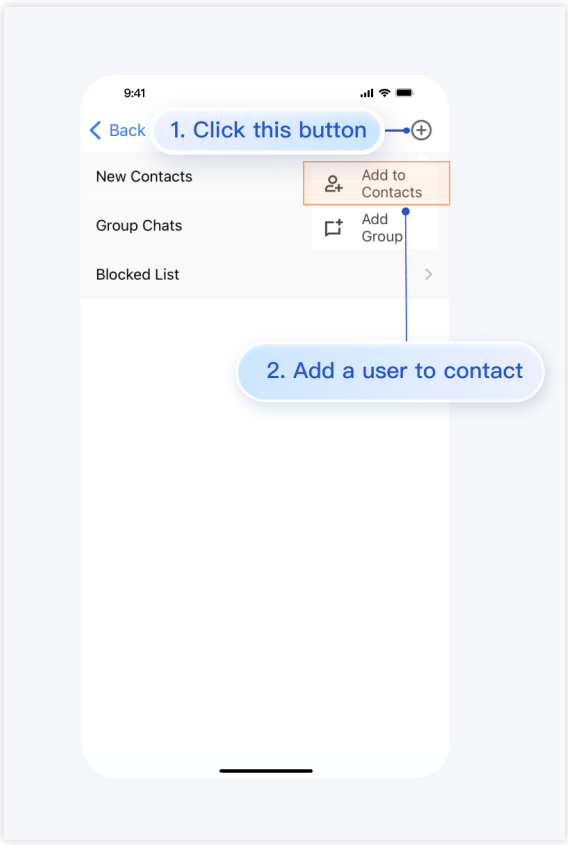
步骤说明

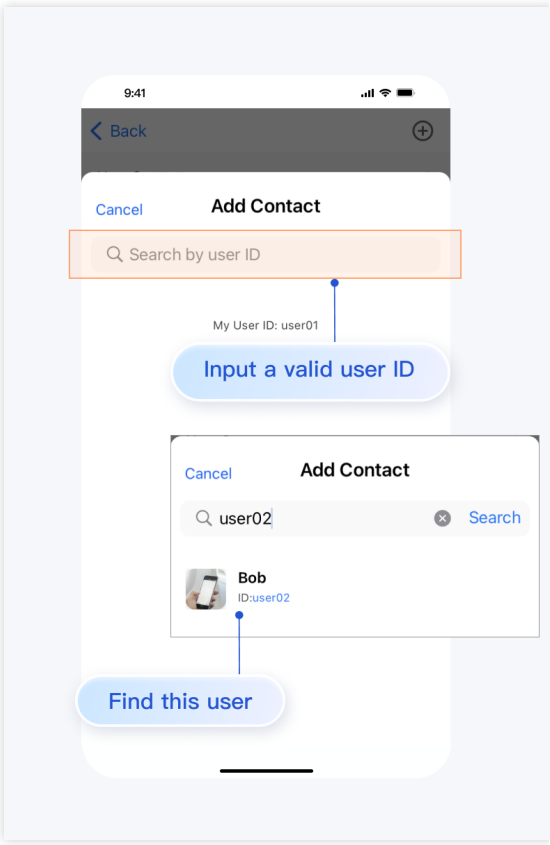
我们推荐您直接在 `TUKit` 中操作，手动添加联系人，步骤如下：
1. 按照 构建联系人界面 中的介绍，将联系人界面展示出来。
2. 点击该界面右上角的 `+` 按钮，在子菜单中选择 `Add to Contacts` 。
3. 输入有效的 userID，搜索出用户。您可以去控制台 `Account Managerment` 页面获取有效的 userID。页面路径：Applications > Your App > Chat > Users > Account Management。
4. 添加用户为联系人。

效果如下图所示：

点击 `Add to Contacts`	搜索用户

©2013-2025 Tencent Cloud International Pte. Ltd.





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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

创建群组

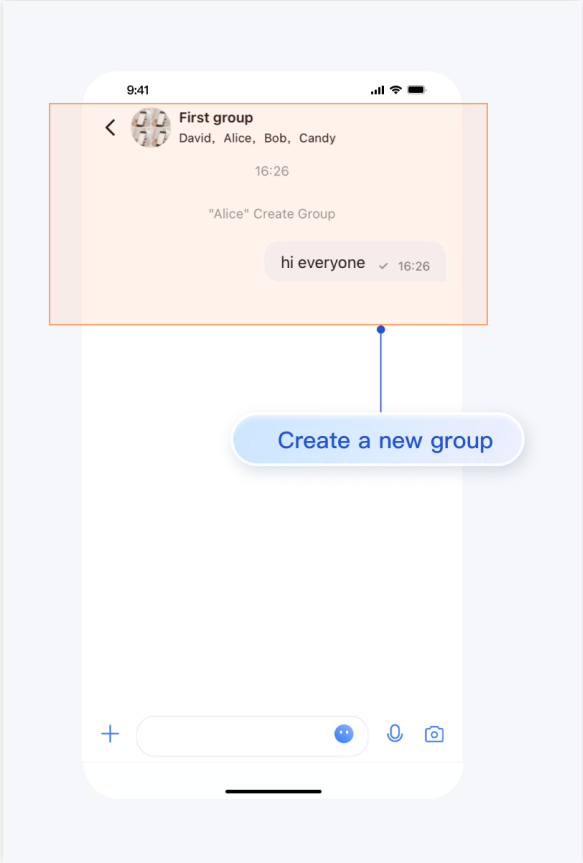
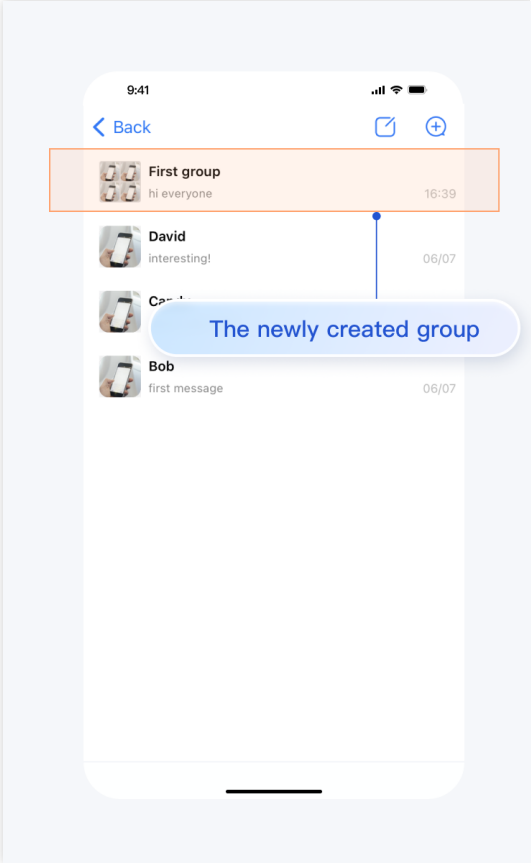
Android

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

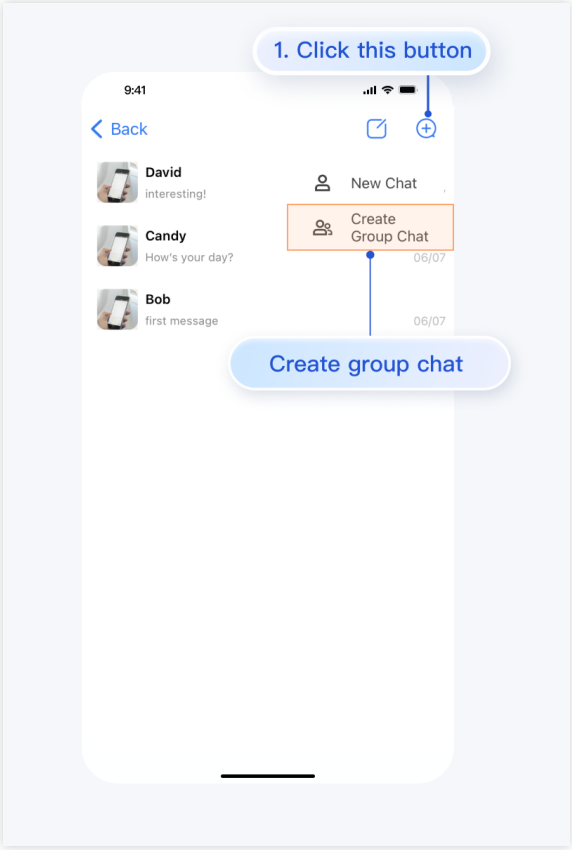
在 `TUIKit` 上手动创建群组有两个前提：
1. 加载出会话列表。请参考文档：构建会话列表。
2. 添加过一些联系人。请参考文档：添加联系人。

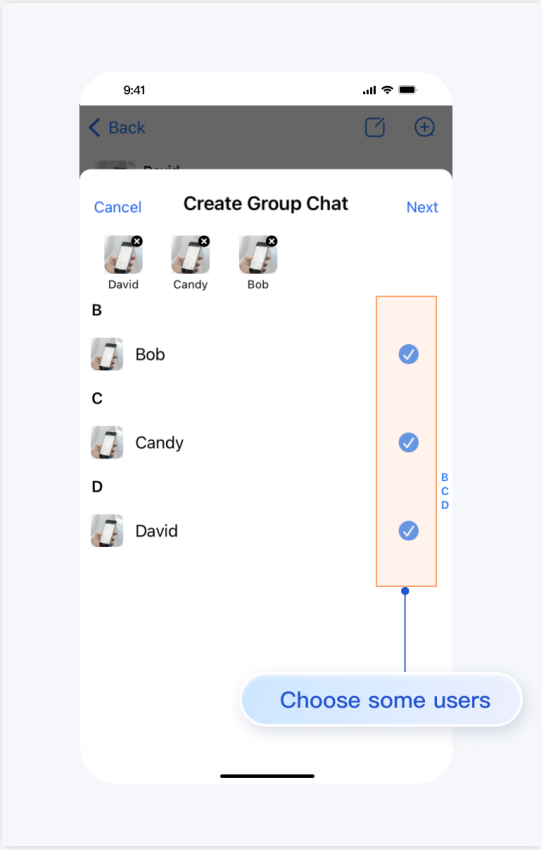
接下来还需要 3 步：
1. 在加载出的会话列表界面，点击右上方的 `+` 号，弹出子菜单，选择 `Create Group Chat` 。
2. 选择若干群组成员。这些成员是您添加的联系人。如果您之前没有添加过任何联系人，此界面将无人可选。
3. 设置群聊的名称、类型、头像等。

效果如下图所示：

单击 Create Group Chat	选择群组成员

©2013-2025 Tencent Cloud International Pte. Ltd.





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。

©2013-2025 Tencent Cloud International Pte. Ltd.

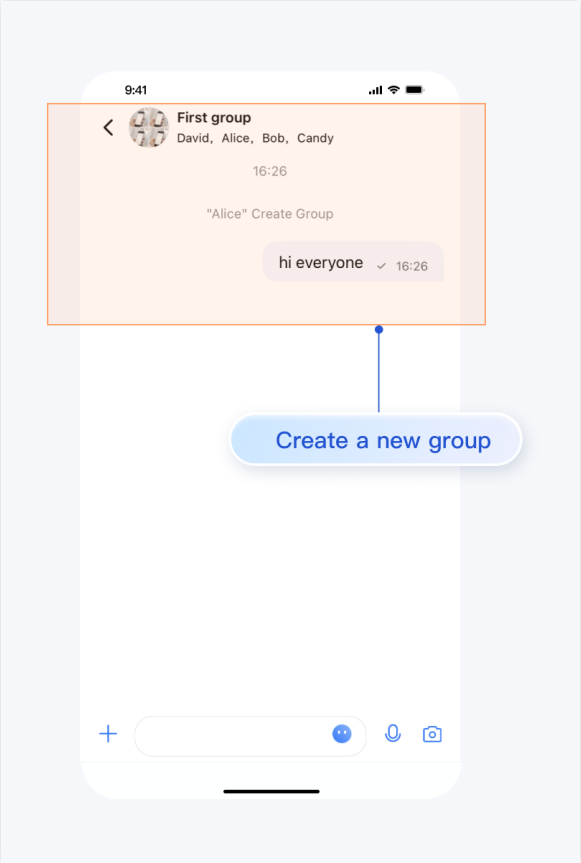
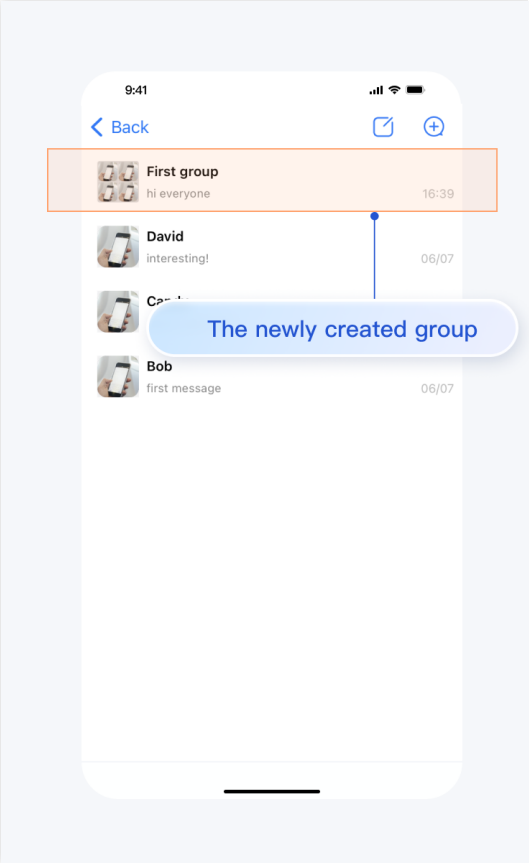
iOS

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

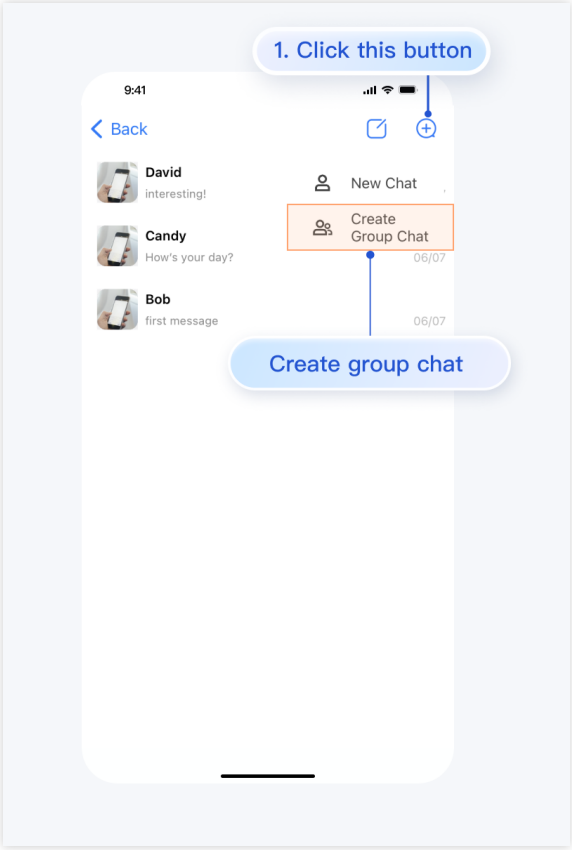
在 `TUIKit` 上手动创建群组有两个前提：
1. 加载出会话列表。请参考文档：构建会话列表。
2. 添加过一些联系人。请参考文档：添加联系人。

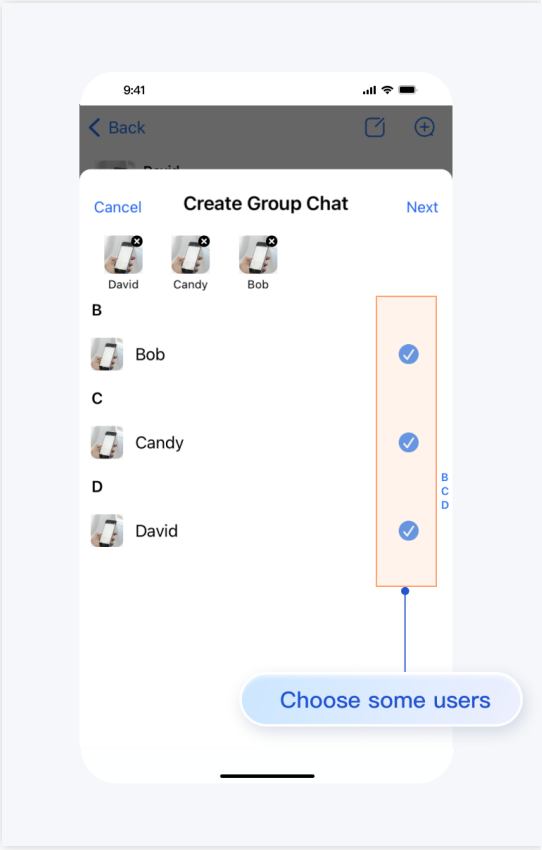
接下来还需要 3 步：
1. 在加载出的会话列表界面，点击右上方的 `+` 号，弹出子菜单，选择 `Create Group Chat` 。
2. 选择若干群组成员。这些成员是您添加的联系人。如果您之前没有添加过任何联系人，此界面将无人可选。
3. 设置群聊的名称、类型、头像等。

效果如下图所示：

点击 `Create Group Chat`	选择群组成员

©2013-2025 Tencent Cloud International Pte. Ltd.





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。

©2013-2025 Tencent Cloud International Pte. Ltd.

音视频通话功能

Android

最近更新时间：2025-02-24 10:00:38

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

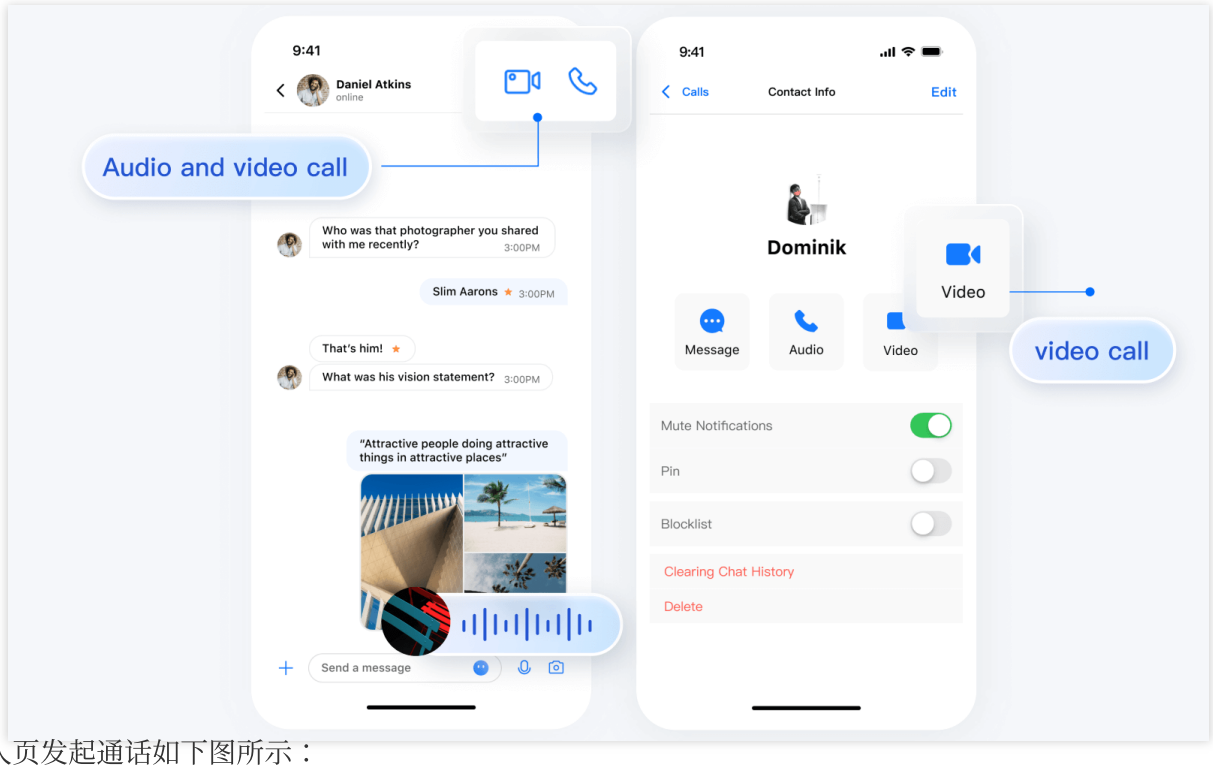
如果您尚未完成以上 4 步，请先参考 快速开始 中的对应步骤完成，否则在实现下文功能时可能遭遇阻碍。
如果您已经完成，请继续阅读下文。

集成步骤

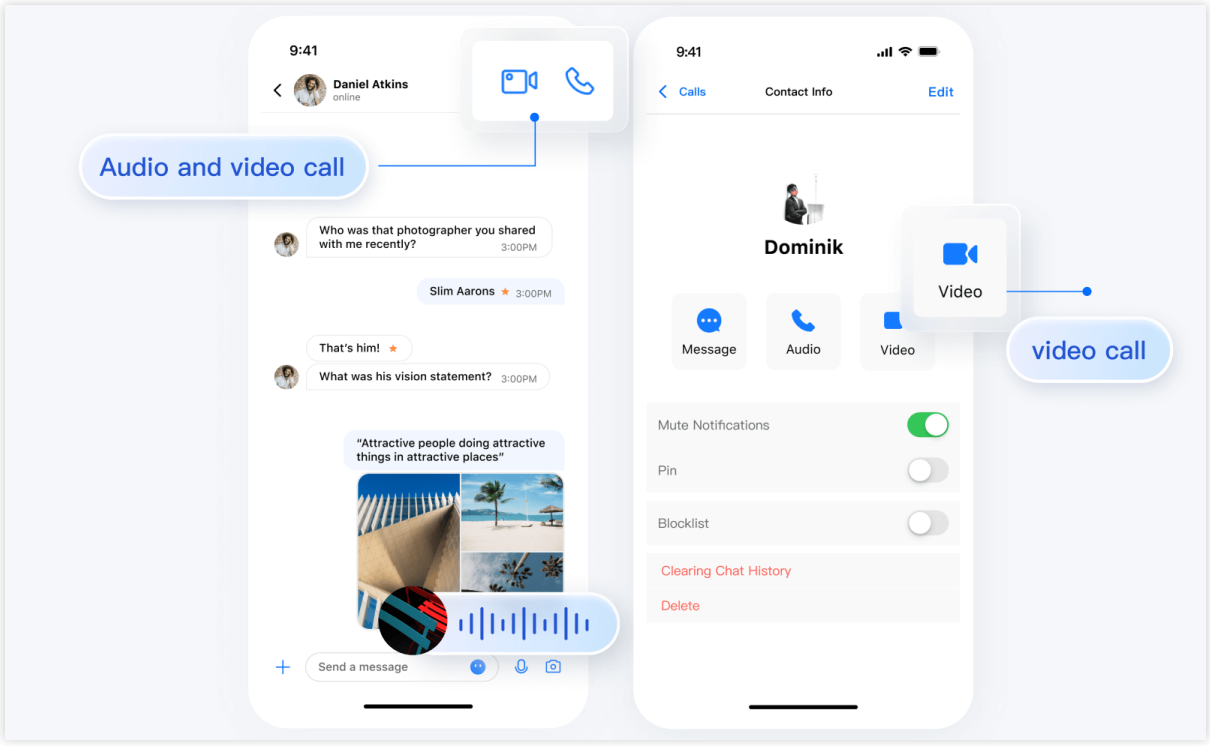
1. 登录 Chat 控制台 开通音视频服务。具体步骤可参考文档：开通服务。
2. 在弹出的开通实时音视频 `TRTC` 服务对话框中，单击“确认”，系统将为您在 实时音视频控制台 创建一个与当前 Chat 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。
3. 集成 `TUICallKit` 组件。在 App 的 build.gradle 文件中添加对 `TUICallKit` 的依赖：

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

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专业版、专业版plus或企业版及以上套餐。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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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_Swift'
```

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专业版、专业版Plus或企业版套餐。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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

修改界面主题

Android

最近更新时间：2025-01-21 18:06:01

概述

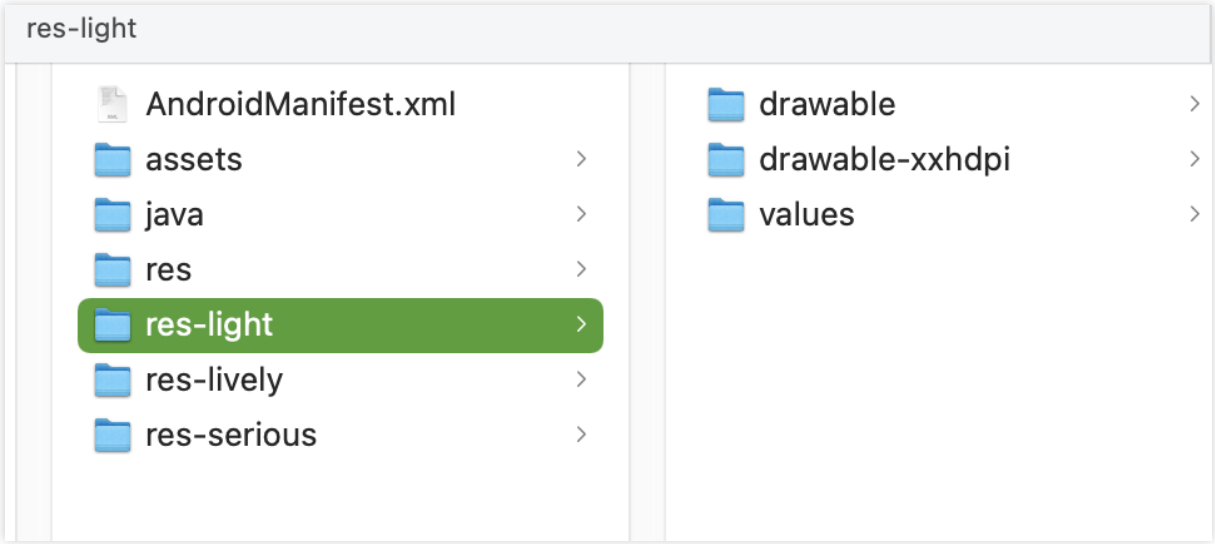
TUI 组件默认内置了：轻量、活泼、深沉共三套主题。您可以任意切换或者修改内置主题，也可以按需新增主题。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

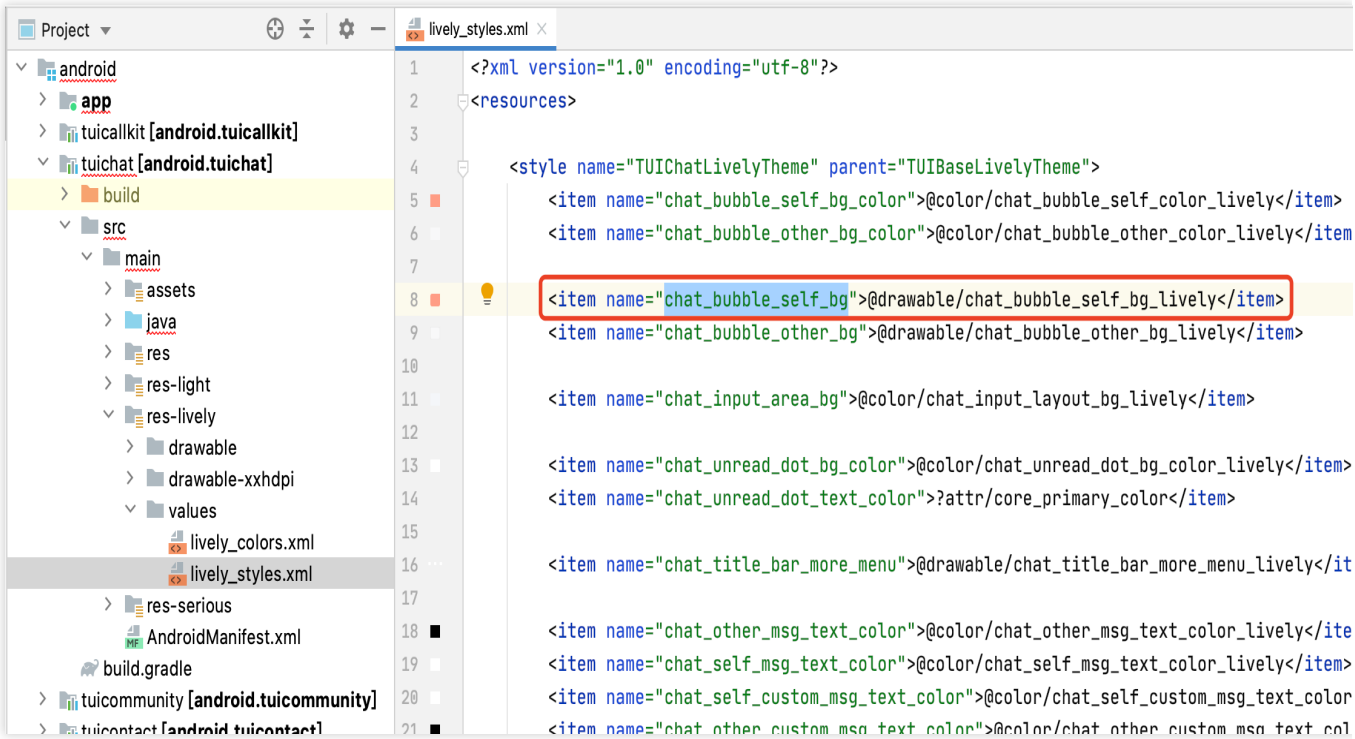
该背景色在内置的 “活泼” 主题下的色值为 #FF9D85，现在想要修改成 #EA286C，您只需要按照如下步骤操作即可：

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：

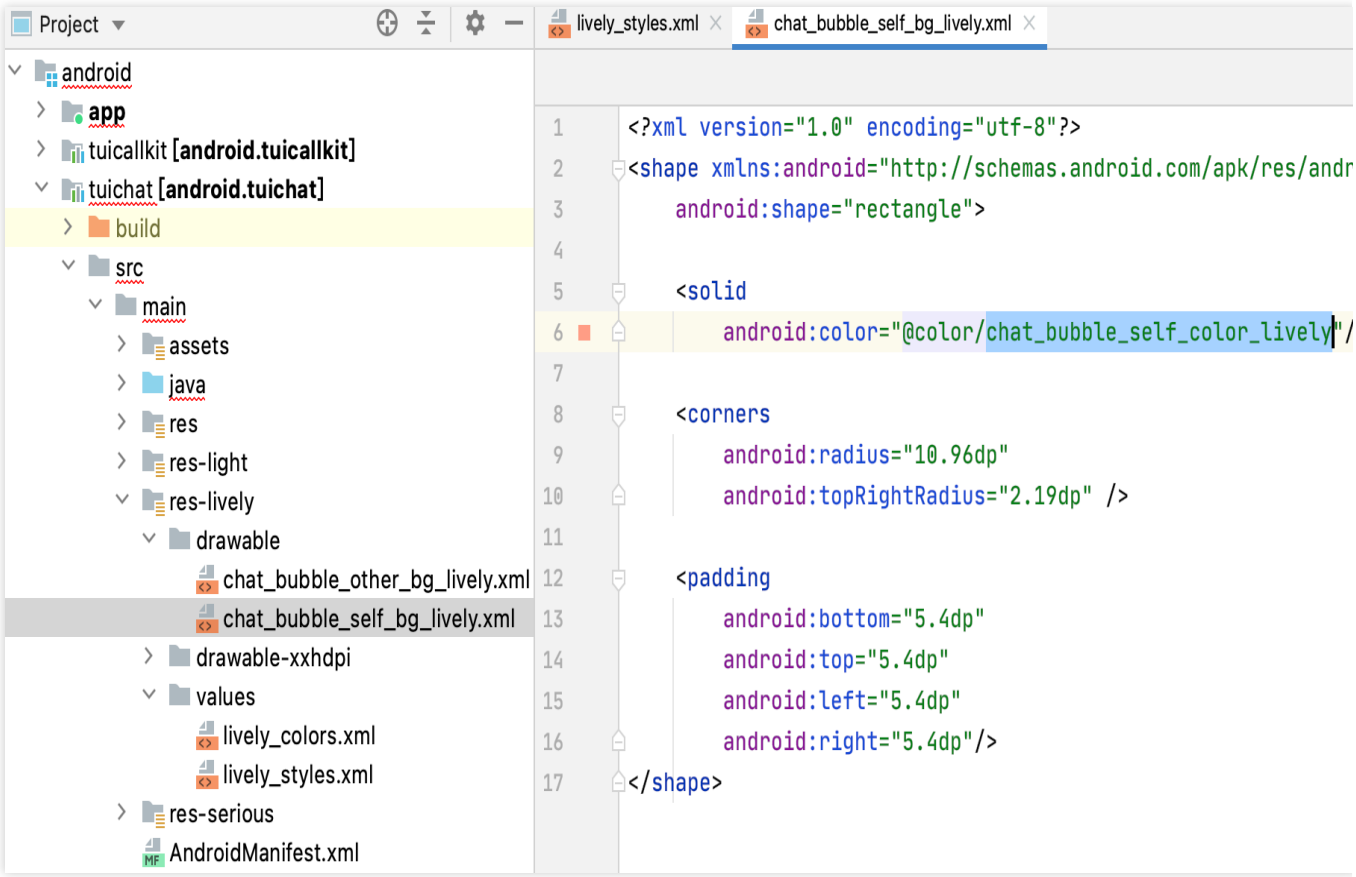
```
if (msg.isSelf()) {
    if (properties.getRightBubble() != null && properties.getRightBubble().getConstantState() != null) {
        msgArea.setBackground(properties.getRightBubble().getConstantState().newDrawable());
    } else {
        msgArea.setBackgroundResource(TUIThemeManager.getAttrResId(itemView.getContext(), R.attr.chat_bubble_self_bg
    }
} else {
    if (properties.getLeftBubble() != null && properties.getLeftBubble().getConstantState() != null) {
        msgArea.setBackground(properties.getLeftBubble().getConstantState().newDrawable());
    } else {
        msgArea.setBackgroundResource(TUIThemeManager.getAttrResId(itemView.getContext(), R.attr.chat_bubble_other_b
    }
}
```

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` ：

©2013-2025 Tencent Cloud International Pte. Ltd.

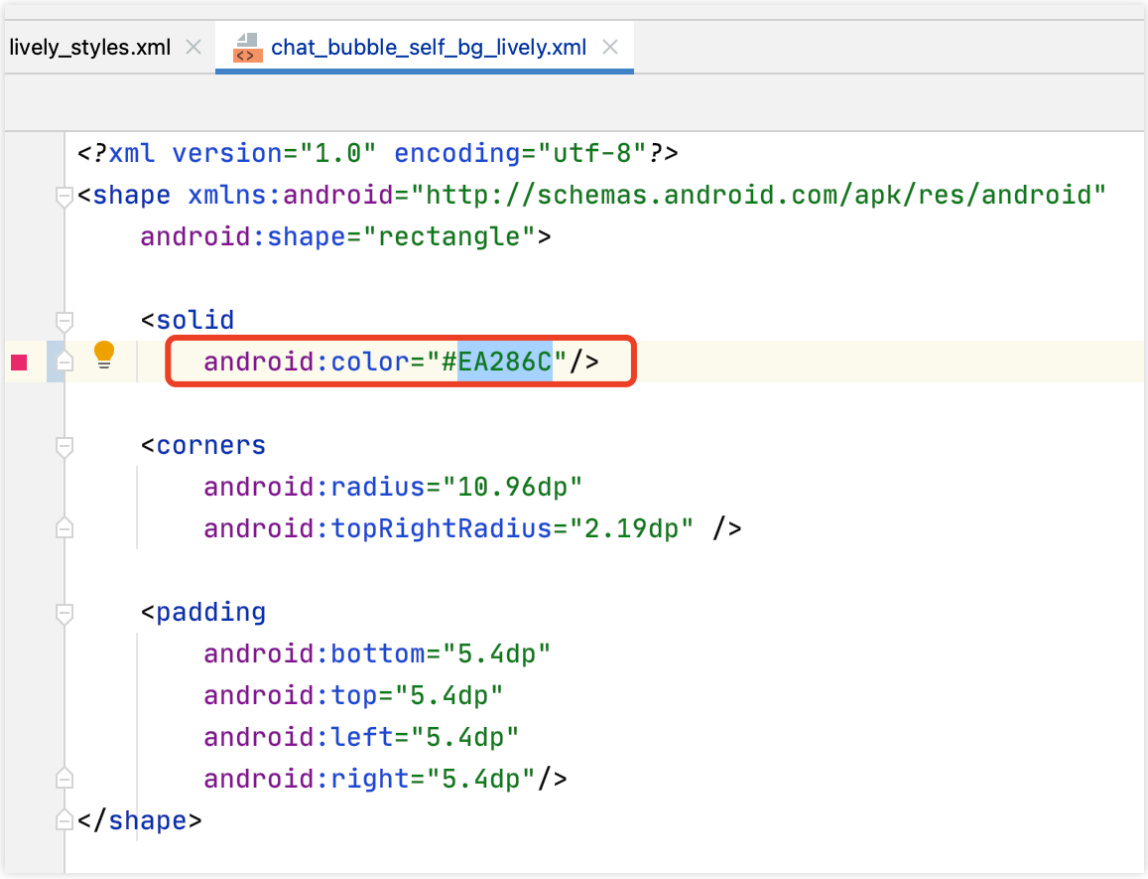


打开对应的资源文件，发现背景色是 `@color/chat_bubble_self_color_lively` ：



©2013-2025 Tencent Cloud International Pte. Ltd.

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` ：

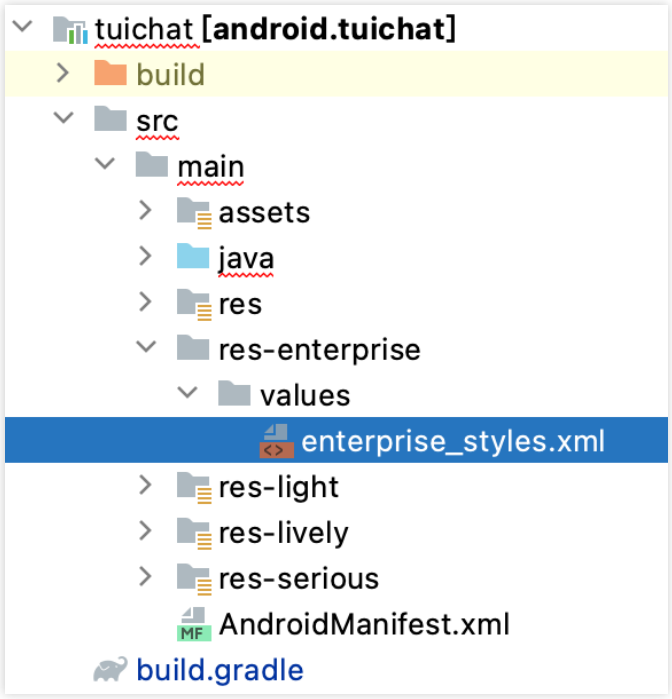


3. 保存文件，重新编译安装应用，切换主题为活泼版主题，即可看到效果。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

        ...

    </style>
</resources>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

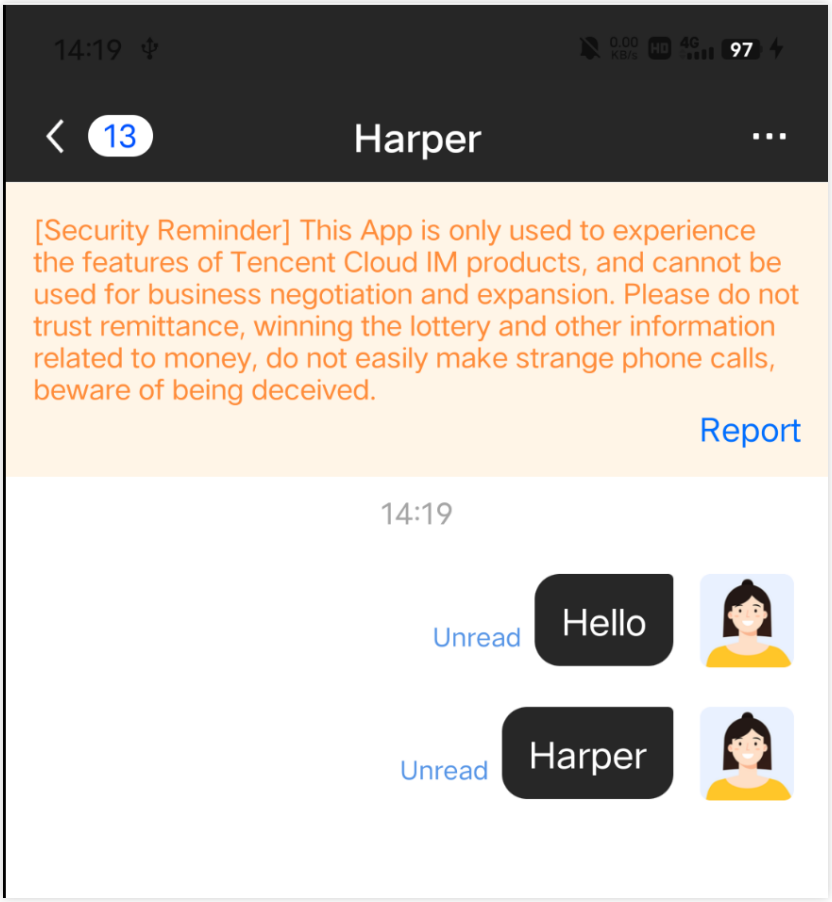
注意

`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);
    }
}
```

5. 切换到新增的 `Enterprise` 主题，即可看到新增加的主题风格：

©2013-2025 Tencent Cloud International Pte. Ltd.



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

©2013-2025 Tencent Cloud International Pte. Ltd.



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
user_status_offline	用户离线状态图标

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Chat 页面样式

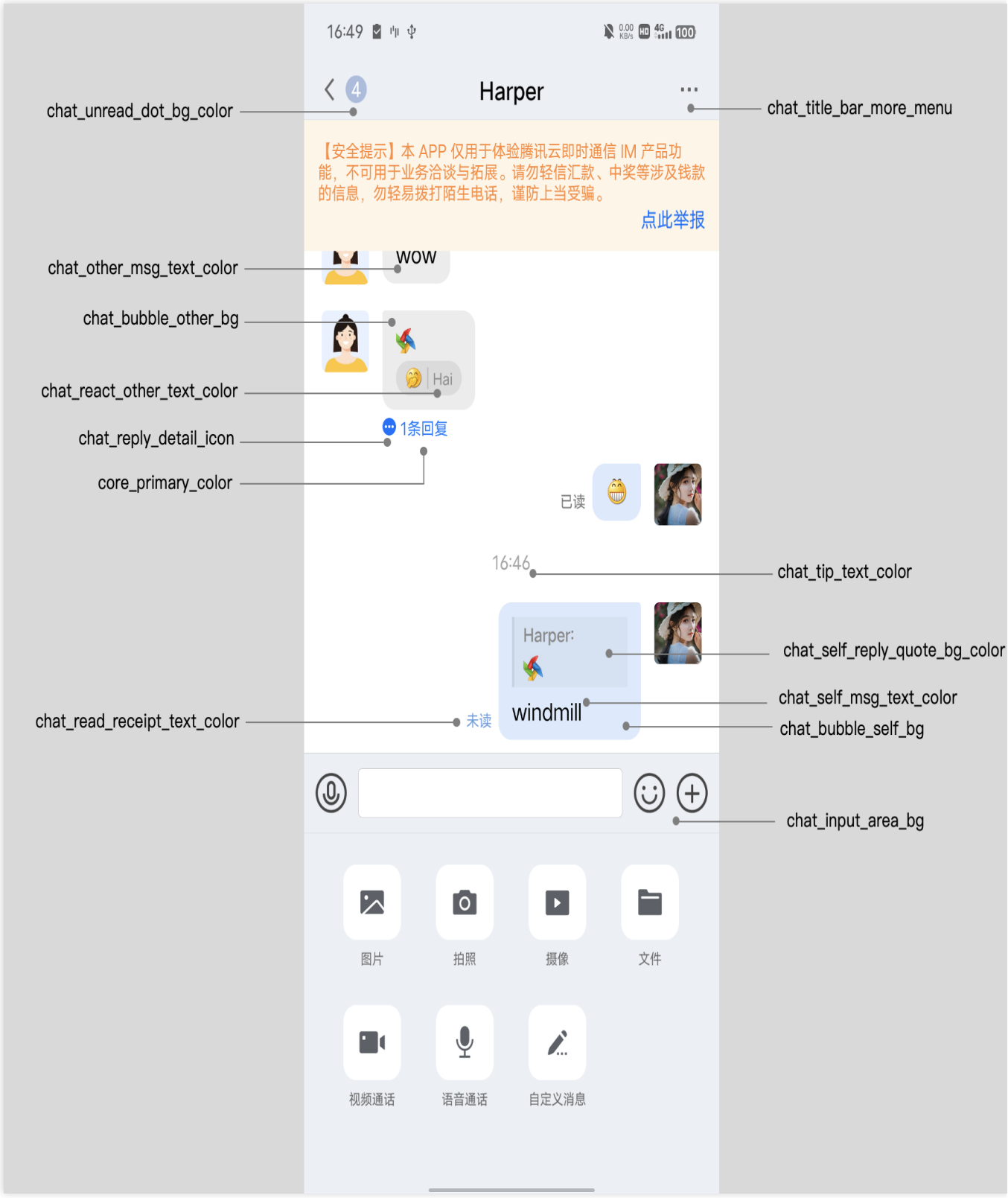
©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.



图标

属性名称	属性说明
chat_title_bar_more_menu	标题栏菜单图标

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Contact 页面样式

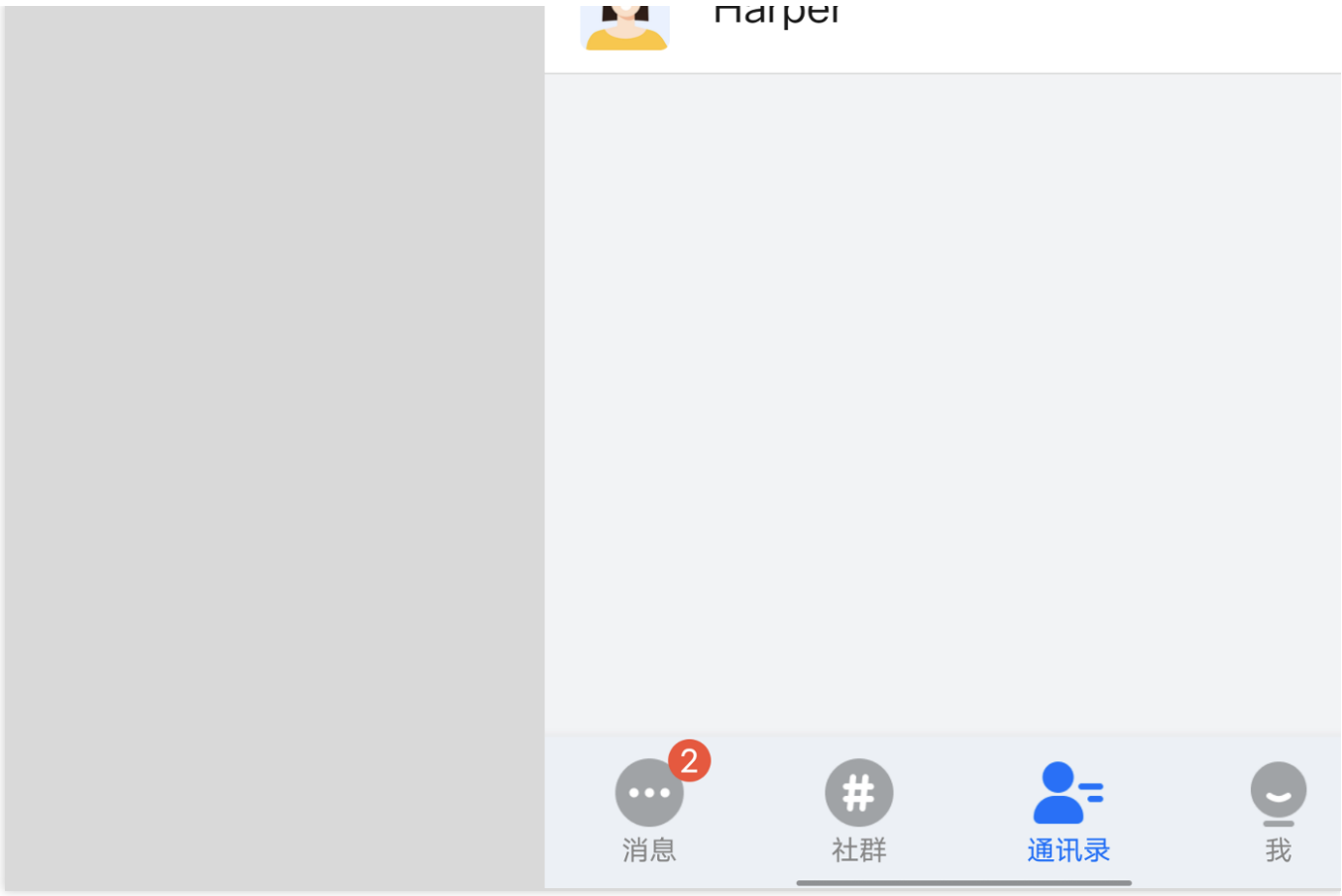
存储位置

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



©2013-2025 Tencent Cloud International Pte. Ltd.



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

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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。
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
public static func applyTheme(_ themeID: String?) {
    var lastThemeID = getCacheThemeID()
    if let themeID = themeID, !themeID.isEmpty {
        lastThemeID = themeID
    }

    if lastThemeID.isEmpty || lastThemeID == "system" {
        TUIThemeManager.share().unApplyTheme(for: .all)
    } else {
        TUIThemeManager.share().applyTheme(lastThemeID, for: .all)
    }

    if gDisableFollowSystemStyle {
        return
    }

    DispatchQueue.main.async {
        if #available(iOS 13.0, *) {
            if lastThemeID.isEmpty || lastThemeID == "system" {
                TUITool.applicationKeywindow()?.overrideUserInterfaceStyle = .unspe
            } else if lastThemeID == "dark" {
                TUITool.applicationKeywindow()?.overrideUserInterfaceStyle = .dark
            } else {
                TUITool.applicationKeywindow()?.overrideUserInterfaceStyle = .light
            }
        }
    }
}
```

```
+ (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];
    } else {
          // 为所有组件应用主题
        [TUIShareThemeManager applyTheme:lastThemeID forModule:TUIThemeModuleAll];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }

    // 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

©2013-2025 Tencent Cloud International Pte. Ltd.

参考 TUIKitDemo 的 AppDelegate 文件。

Swift

Objective-C

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
    AppDelegate.sharedInstance = self

    lastLoginResultCode = 0

    NSSetUncaughtExceptionHandler { exception in
        print("CRASH: \\(exception)")
        print("Stack Trace: \\(exception.callStackSymbols)")
    }

    TUISwift.tuiRegisterThemeResourcePath(TUISwift.tuiBundlePath("TUIDemoTheme", ke
    TUIThemeSelectController.applyLastTheme()

    setupListener()
    setupGlobalUI()
    setupConfig()
    tryPreloadMainVC()
    tryAutoLogin()

    return true
}
```

```
- (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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
```

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

Swift

Objective-C

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
    // Some other code

    let customChatThemePath = Bundle.main.path(forResource: "TUIChatCustomTheme.bun
    TUISwift.tuiRegisterThemeResourcePath(customChatThemePath, themeModule: TUIThem
    TUIThemeManager.share().applyTheme("enterprise", for: TUIThemeModule.chat)

    return true
}
```

```
- (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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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
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

©2013-2025 Tencent Cloud International Pte. Ltd.

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
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

©2013-2025 Tencent Cloud International Pte. Ltd.

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
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

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

UI 样式表

©2013-2025 Tencent Cloud International Pte. Ltd.

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
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

Common 页面样式

Common 页面样式均存在于 TIMCommon 组件中，属于IM 组件公共模块，由各个组件引用。
您可以在 TIMCommon 组件的 TIMCommon/Resources/TIMCommonTheme.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，TIMCommon 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  // 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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  /// 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

©2013-2025 Tencent Cloud International Pte. Ltd.

设置界面风格

Android

全局

最近更新时间：2024-10-30 10:54:32

下文将向您展示如何设置全局自定义选项及其效果。

切换 TUIKit 语言

API 作用：切换 TUIKit 语言。
方法原型：

```
// TUIConfigMinimalist.java
/**
 * Switch the language of TUIKit.
 * The currently supported languages are "en", "zh", and "ar".
 */
public static void switchLanguageToTarget(Context context, String targetLanguage)
```

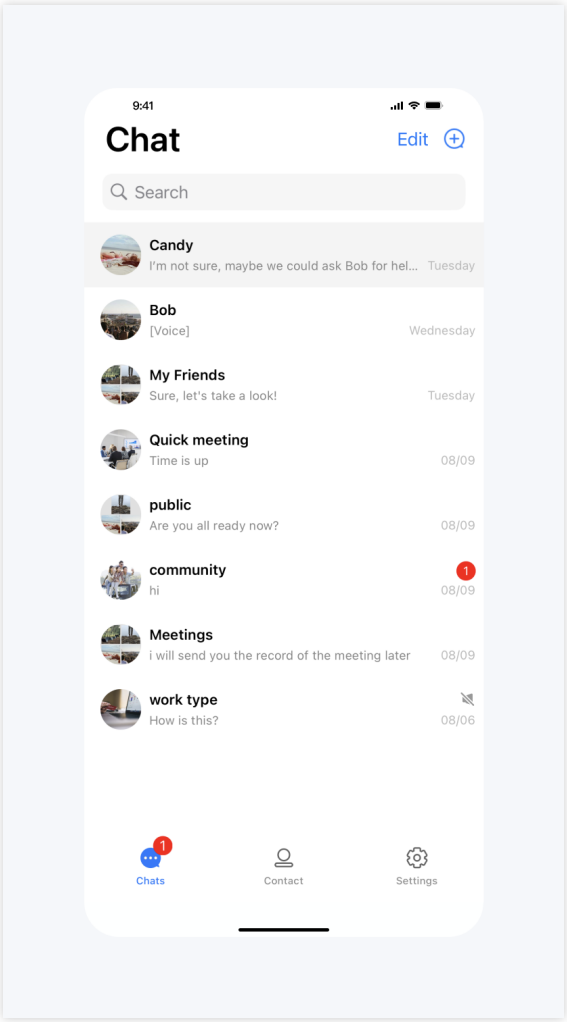
示例代码：

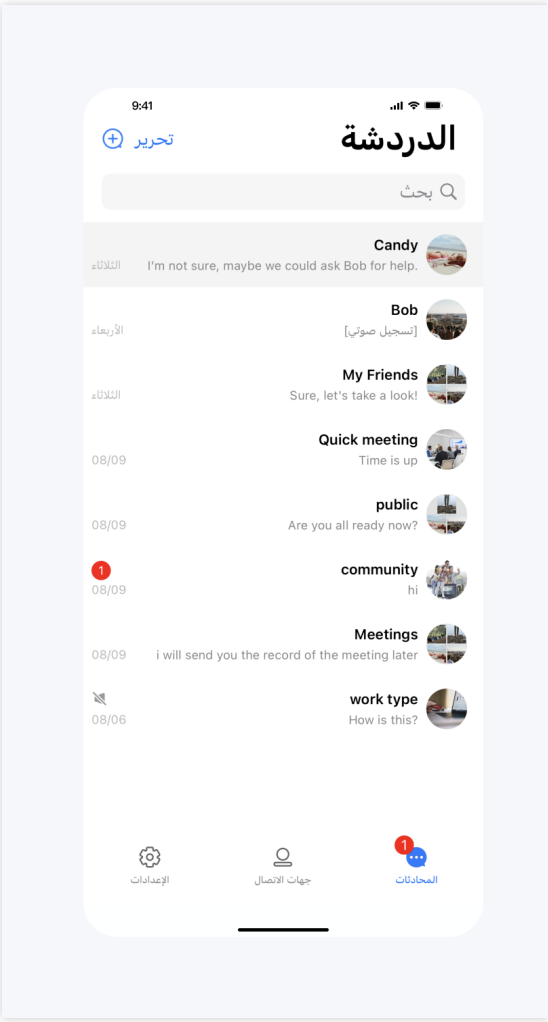
```
// When to call: In the application initialization phase.
TUIConfigMinimalist.switchLanguageToTarget(context, "en");
```

设置效果：

设置为英文	设置为阿拉伯语

©2013-2025 Tencent Cloud International Pte. Ltd.





开启 TUIKit 内建提示

API 作用：设置是否开启 TUIKit 内建 toast 提示。默认开启，TUIKit 组件会显示内建的提示。

方法原型：

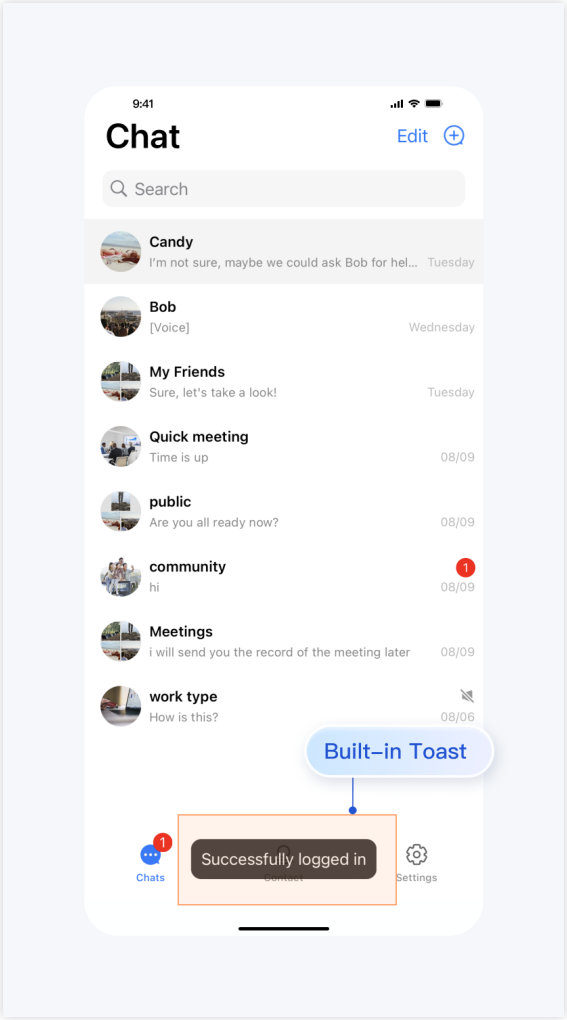
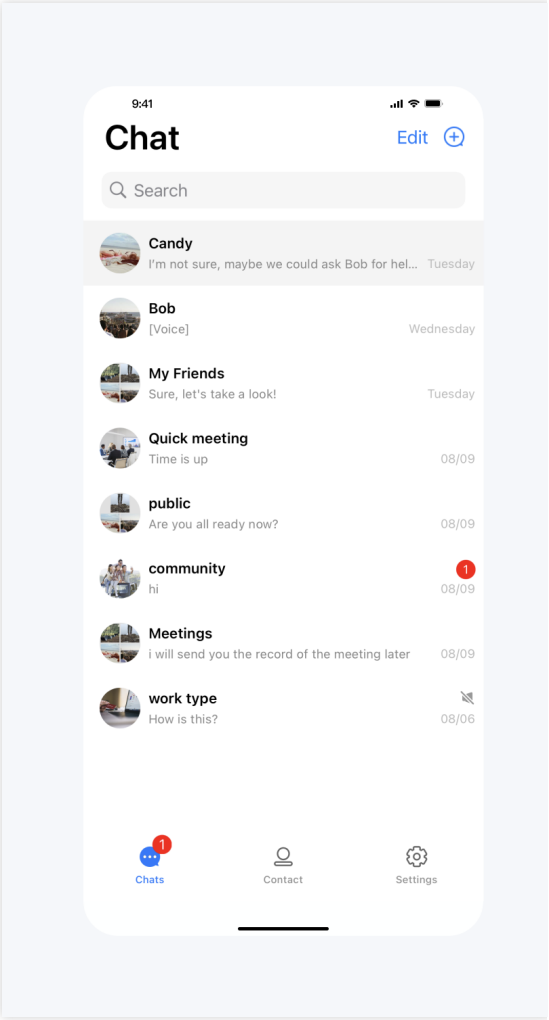
```
// TUIConfigMinimalist.java
/**
 *  Show the toast prompt built in TUIKit.
 *  The default value is true.
 */
public static void enableToast(boolean enableToast)
```

示例代码：

```
// When to call: Before initializing the TUIKit interface.
TUIConfigMinimalist.enableToast(false);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

设置效果：

开启 Toast 提示	关闭 Toast 提示
	

©2013-2025 Tencent Cloud International Pte. Ltd.

聊天界面

最近更新时间：2024-10-30 10:56:24

下文将向您展示如何设置聊天界面自定义选项及其效果。

消息列表相关

聊天界面背景色、背景图片

API 作用：设置聊天界面消息列表背景色、背景图片，针对所有聊天界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Customize the backgroud of message list interface.
 * This configuration takes effect in all message list interfaces.
 */
public static void setBackground(Drawable background)
```

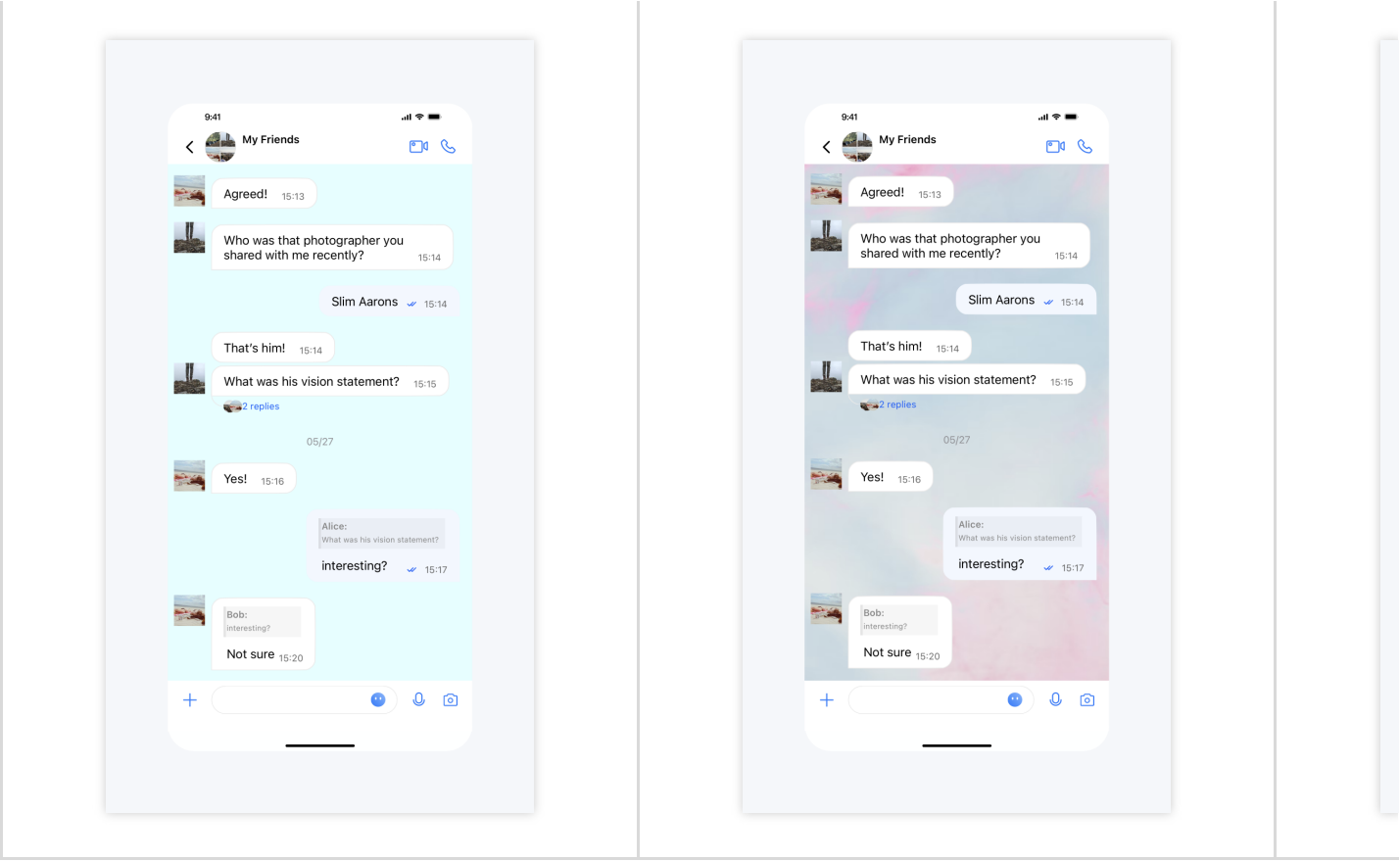
示例代码：

```
// When to call: Before initializing the message list interface.
TUIChatConfigMinimalist.setBackground(new ColorDrawable(0xFFE1FFFF));
TUIChatConfigMinimalist.setBackground(context.getDrawable(R.drawable.your_backgroun
```

设置效果：

背景色	设置背景图片	默认

©2013-2025 Tencent Cloud International Pte. Ltd.



消息列表用户头像大小、圆角半径

API 作用：设置消息列表用户头像大小、圆角半径。

API 原型：

```
// TUIConfigMinimalist.java

/**
 *  Customize the size of avatar.
 *  This configuration takes effect in message list.
 */
public static void setMessageListAvatarSize(int size)

/**
 *  Customize the corner radius of the avatar.
 *  This configuration takes effect in message list.
 */
public static void setMessageListAvatarRadius(int radius)
```

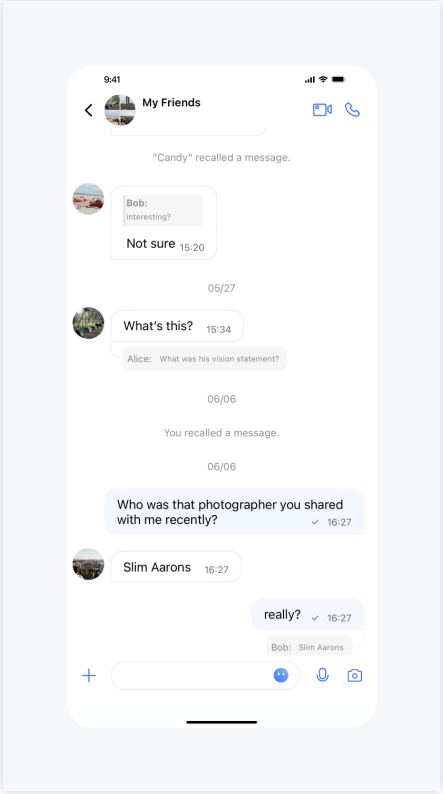
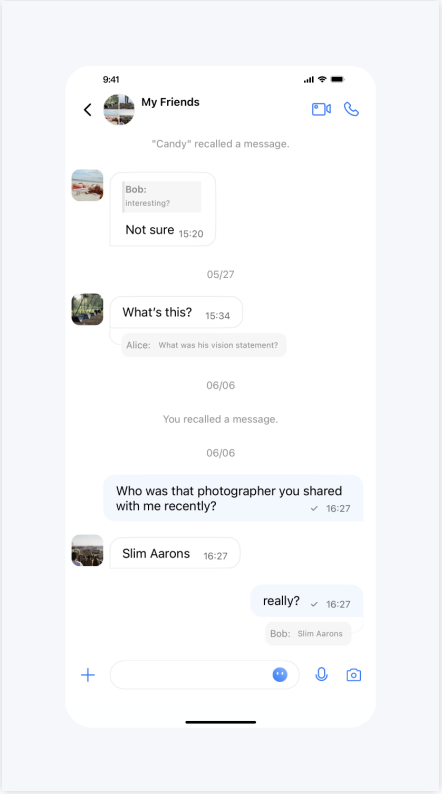
示例代码：

```
// When to call: Before initializing the TUIKit interfaces.
// Set size
TUIConfigMinimalist.setMessageListAvatarSize(50);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// Set cornerRadius
TUIConfigMinimalist.setMessageListAvatarRadius(10);
```

设置效果：

默认圆形头像	设置圆角矩形头像	设置矩形
		

设置群头像展示九宫格

API 作用：设置群头像展示九宫格。针对消息列表、会话列表和联系人列表生效。

API 原型：

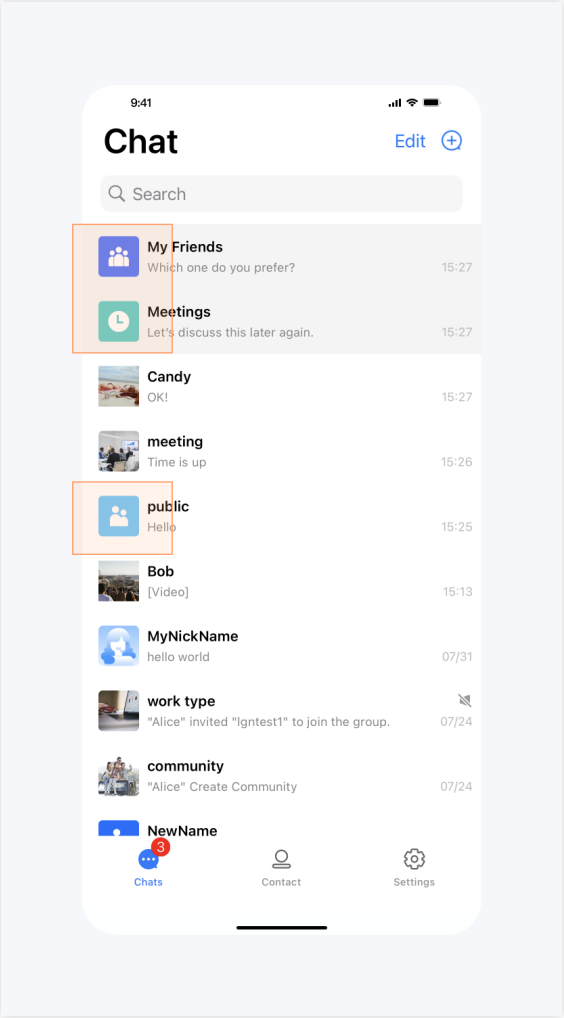
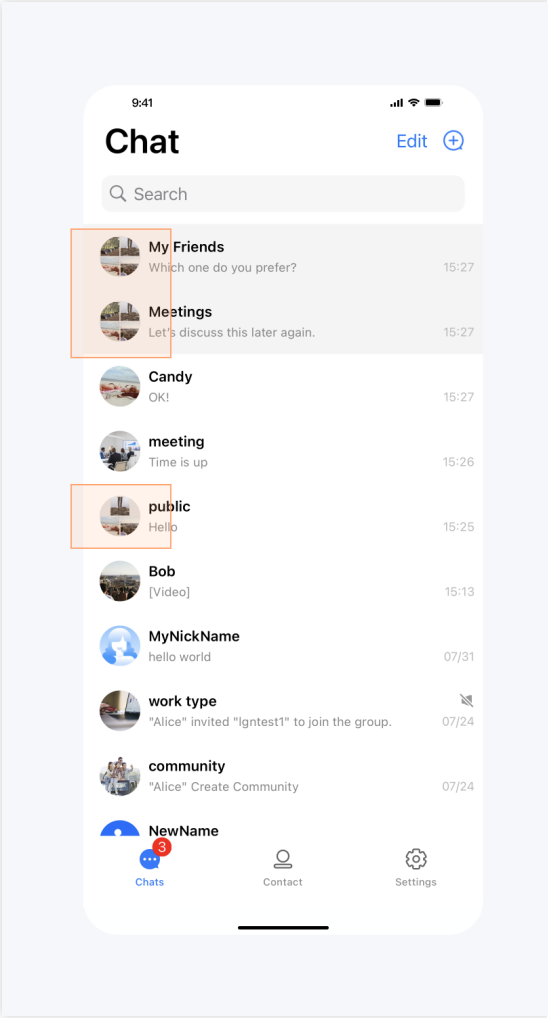
```
// TUIConfigMinimalist.java
/**
 * Display the group avatar in the nine-square grid style.
 * The default value is true.
 * This configuration takes effect in all groups.
 */
public static void setEnableGroupGridAvatar(boolean enableGroupGridAvatar)
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: Before initializing the TUIKit interfaces.
TUIConfigMinimalist.setEnableGroupGridAvatar(false);
```

设置效果：

设置群头像不展示九宫格	默认
	

开启正在输入状态指示

API 作用：开启“正在输入”状态指示。针对所有 1v1 聊天消息界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 *  Enable the display "Alice is typing..." on one-to-one chat interface.
```

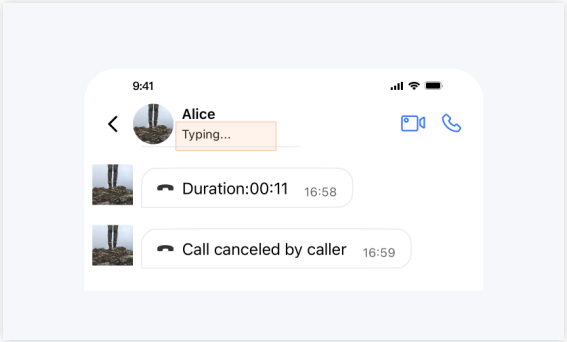
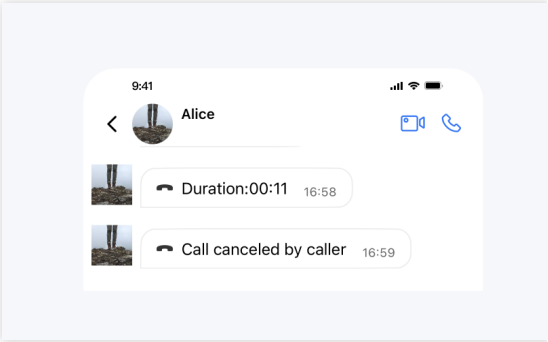
©2013-2025 Tencent Cloud International Pte. Ltd.

```
 *  The default value is true.
 *  This configuration takes effect in all one-to-one chat message list interfaces.
 */
public static void setEnableTypingIndicator(boolean enableTypingIndicator)
```

示例代码：

```
// When to call: Before initializing the message list interface.
TUIChatConfigMinimalist.setEnableTypingIndicator(false);
```

设置效果：

开启“正在输入”	不开启“正在输入”
	

开启消息已读回执

API 作用：开启消息已读回执，开启后可以在消息详情中查看已读信息。针对所有消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 *  When sending a message, set this flag to require message read receipt.
 *  The default value is false.
 *  This configuration takes effect in all chat message list interfaces.
 */
public static void setMessageReadReceiptNeeded(boolean messageReadReceiptNeeded)
```

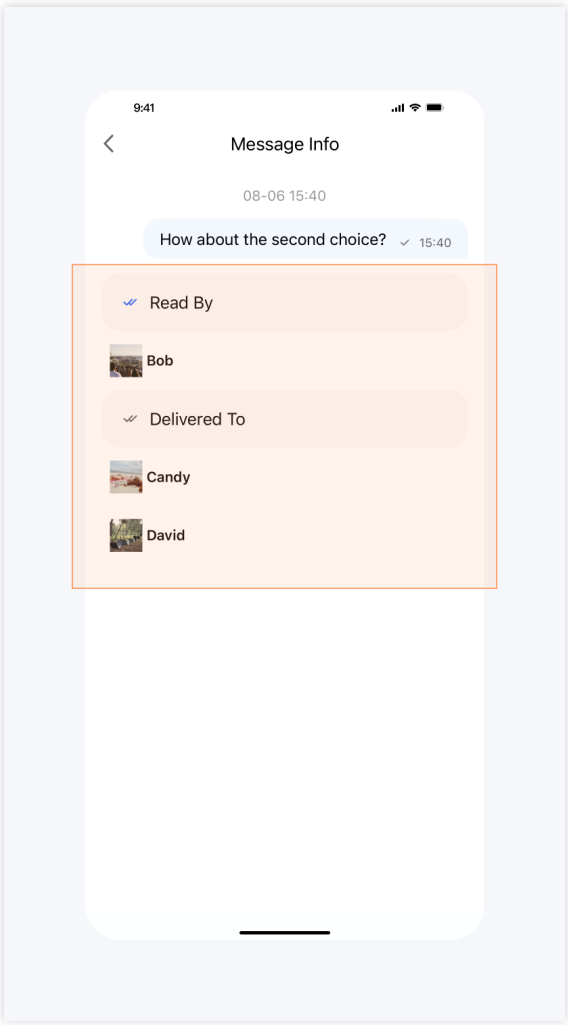
示例代码：

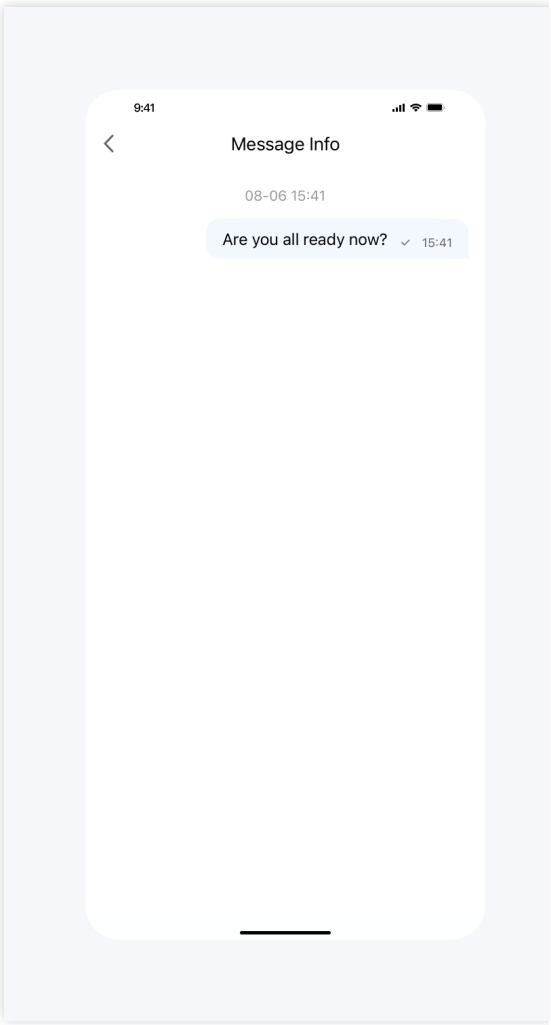
```
// When to call: Before sending messages.
TUIChatConfigMinimalist.setMessageReadReceiptNeeded(true);
```

设置效果：

开启消息已读回执	不开启消息已读回执

©2013-2025 Tencent Cloud International Pte. Ltd.





隐藏长按消息菜单按钮

API 作用：隐藏长按消息菜单中的指定按钮，针对所有聊天消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
public static final int REPLY = 1;
public static final int QUOTE = 2;
public static final int EMOJI_REACTION = 3;
public static final int PIN = 4;
public static final int RECALL = 5;
public static final int TRANSLATE = 6;
public static final int CONVERT = 7;
public static final int FORWARD = 8;
public static final int SELECT = 9;
public static final int COPY = 10;
public static final int DELETE = 11;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
public static final int INFO = 12;
public static final int SPEAKER_MODE_SWITCH = 13;

@IntDef({REPLY, QUOTE, EMOJI_REACTION, PIN, RECALL, TRANSLATE, CONVERT, FORWARD, SE
public @interface LongPressPopMenuItem {}

/**
 * Hide the items in the pop-up menu when user presses the message.
 */
public static void hideItemsWhenLongPressMessage(@LongPressPopMenuItem int... items
```

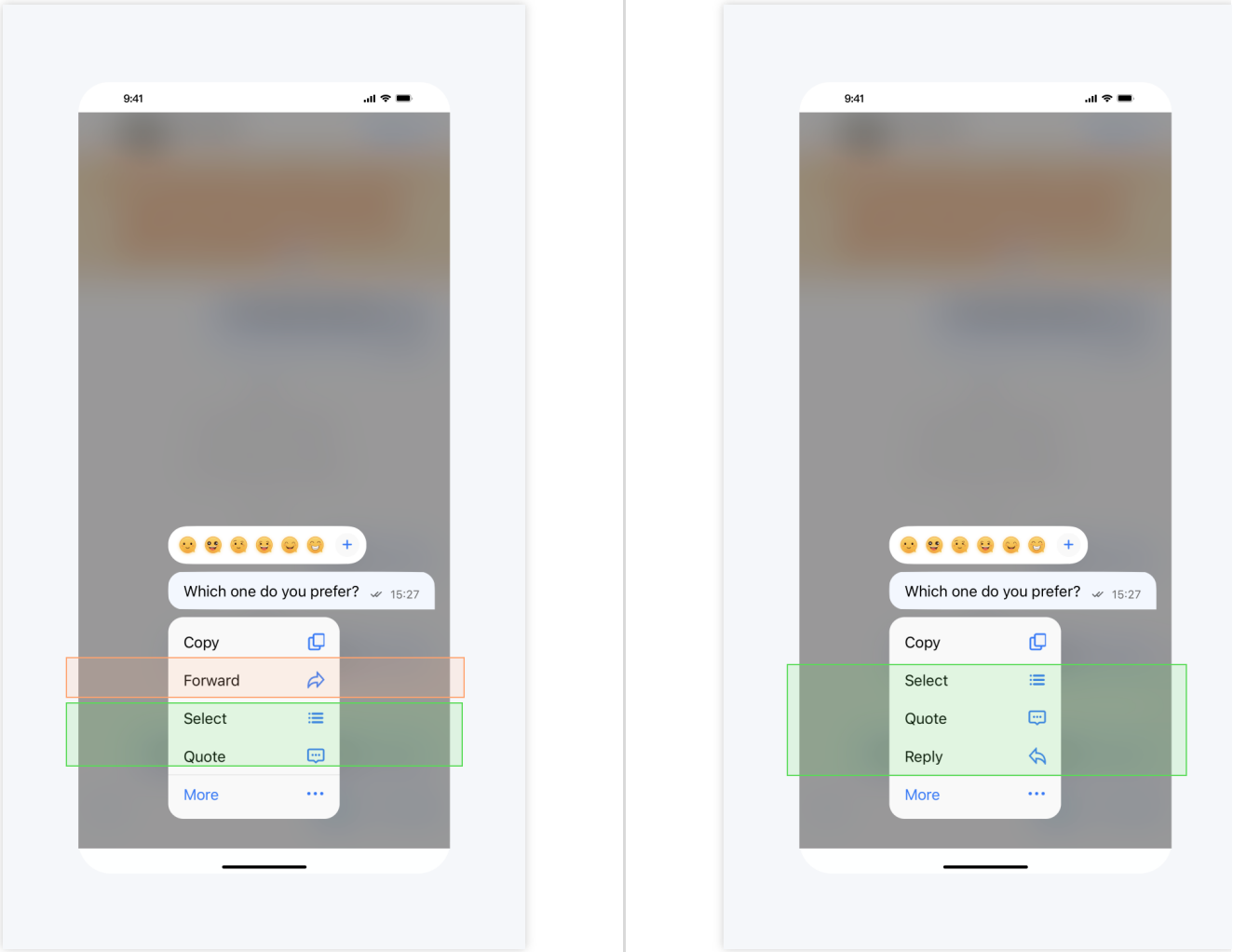
示例代码：

```
// When to call: Before displaying the pop-up menu when user presses the message.
TUIChatConfigMinimalist.hideItemsWhenLongPressMessage(TUIChatConfigMinimalist.FORWA
                                                      TUIChatConfigMinimalist.INFO)
```

设置效果：

不隐藏任何按钮	隐藏 Forward 按钮

©2013-2025 Tencent Cloud International Pte. Ltd.



开启音视频通话浮窗

API 作用：开启音视频通话浮窗。开启后，您可以将音视频通话界面以小窗口的形式漂浮在聊天界面。针对所有音视频通话界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Turn on audio and video call floating windows,
 * The default value is true.
 */
public static void setEnableFloatWindowForCall(boolean enableFloatWindowForCall)
```

示例代码：

```
// When to call: Before entering the message list interface.
TUIChatConfigMinimalist.setEnableFloatWindowForCall(false);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

开启音视频通话多端登录

API 作用：开启音视频通话多端登录。针对所有音视频通话生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Enable multi-terminal login function for audio and video calls
 * The default value is false.
 */
public static void setEnableMultiDeviceForCall(boolean enableMultiDeviceForCall)
```

示例代码：

```
// When to call: Before entering the message list interface.
TUIChatConfigMinimalist.setEnableMultiDeviceForCall(true);
```

设置消息列表顶部自定义 View

API 作用：设置聊天界面顶部自定义 View，针对所有聊天消息界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Add a custom view at the top of the chat interface.
 * This view will be displayed at the top of the message list and will not slide up
 */
public static void setCustomTopView(View customTopView)
```

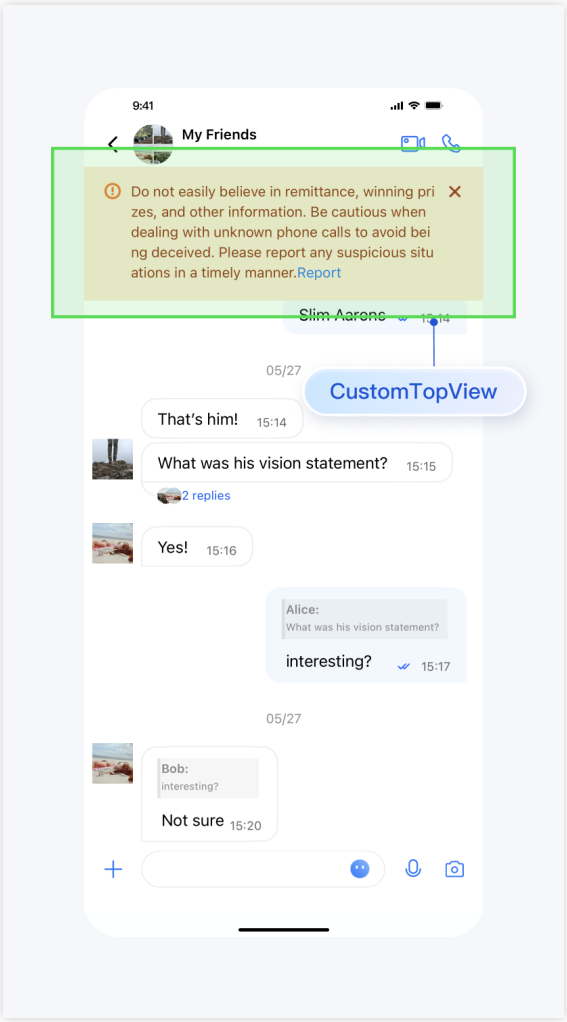
示例代码：

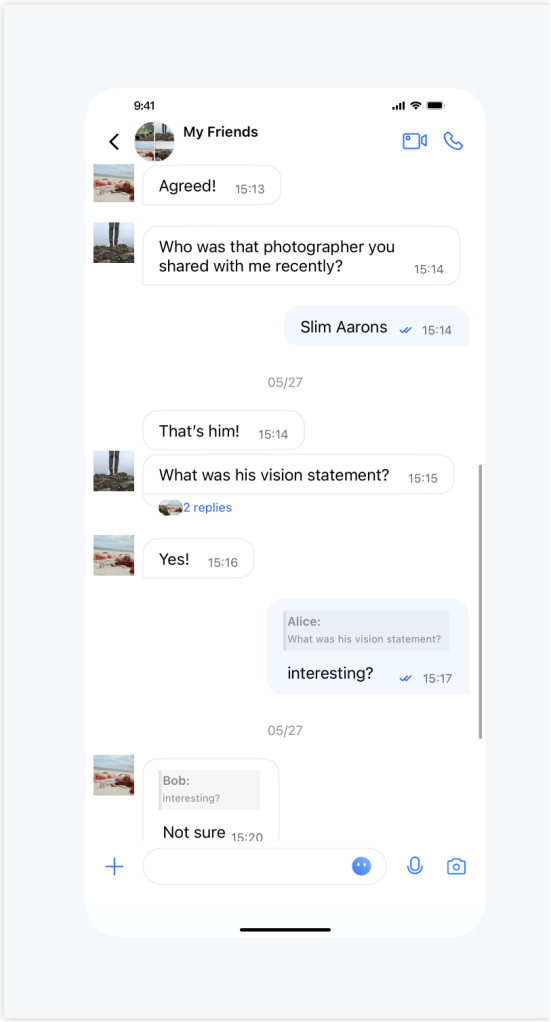
```
// When to call: Before initializing the message list interface.
// tipsView is your customized view.
TUIChatConfigMinimalist.setCustomTopView(tipsView);
```

设置效果：

设置自定义view	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





设置消息是否不计入未读

API 作用：设置即将发送的消息不更新会话未读数，针对所有消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is false.
 */
public static void setExcludedFromUnreadCount(boolean excludedFromUnreadCount)
```

示例代码：

```
// When to call: Before sending messages.
TUIChatConfigMinimalist.setExcludedFromUnreadCount(true);
```

设置消息是否不更新会话 lastMsg

©2013-2025 Tencent Cloud International Pte. Ltd.

API 作用：设置即将发送的消息不更新会话 lastMsg，针对所有消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is false.
 */
public static void setExcludedFromLastMessage(boolean excludedFromLastMessage)
```

示例代码：

```
// When to call: Before sending messages.
TUIChatConfigMinimalist.setExcludedFromLastMessage(true);
```

消息撤回时间间隔

API 作用：设置消息撤回时间，针对所有消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Time interval within which a message can be recalled after being sent.
 * The default value is 120 seconds.
 * If you want to adjust this configuration, please modify the setting on Chat Cons
 */
public static void setTimeIntervalForAllowedMessageRecall(int timeIntervalForAllowe
```

示例代码：

```
// When to call: Before sending messages.
TUIChatConfigMinimalist.setTimeIntervalForAllowedMessageRecall(90);
```

语音、视频消息最长录制时长

API 作用：设置语音、视频消息最长录制时长，针对所有语音、视频消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Maximum audio recording duration, no more than 60s.
 * The default value is 60 seconds.
 */
public static void setMaxAudioRecordDuration(int maxAudioRecordDuration)
/**
 * Maximum video recording duration, no more than 15s.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 * The default value is 15 seconds.
 */
public static void setMaxVideoRecordDuration(int maxVideoRecordDuration)
```

示例代码：

```
// When to call: Before recording audio or video messages.
TUIChatConfigMinimalist.setMaxAudioRecordDuration(10);
TUIChatConfigMinimalist.setMaxVideoRecordDuration(10);
```

开启自定义铃声

API 作用：设置 Android 设备收到消息时的铃声为内置的自定义铃声，针对所有消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Enable custom ringtone.
 * This config takes effect only for Android devices.
 */
public static void setEnableAndroidCustomRing(boolean enableAndroidCustomRing)
```

示例代码：

```
// When to call: Before sending messages.
TUIChatConfigMinimalist.setEnableAndroidCustomRing(true);
```

开启语音消息扬声器播放

API 作用：设置播放语音消息时默认使用扬声器而不是听筒播放。针对所有语音消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Call this method to use speakers instead of handsets by default when playing voi
 */
public static void setPlayingSoundMessageViaSpeakerByDefault(boolean playingSoundMe
```

示例代码：

```
// When to call: Before initializing the Message interface.
TUIChatConfigMinimalist.setPlayingSoundMessageViaSpeakerByDefault(true);
```

注册自定义消息

API 作用：注册自定义消息。使用场景请参考文档《添加自定义消息》。

©2013-2025 Tencent Cloud International Pte. Ltd.

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Register custom message.
 * - Parameters:
 *   - businessID: Customized message‘s businessID, which is unique.
 *   - messageBeanClass: Customized message's MessagBean class.
 *   - messageViewHolderClass: Customized message's MessagViewHolder class.
 *   - isUseEmptyViewGroup: If true, the user avatar and message bubble will not be
 */
public static void registerCustomMessage(String businessID, Class<? extends TUIMess
    Class<? extends RecyclerView.ViewHolder> messageViewHolderClass, boolean isUseE
```

示例代码：

```
// When to call: Before initializing the Message List interface.
TUIChatConfigMinimalist.registerCustomMessage(CUSTOM_LINK_MESSAGE_BUSINESS_ID,
                                                                    CustomLinkMessage
                                                                    CustomLinkMessage
                                                false);
```

点击、长按消息列表里的用户头像

API 作用：用户点击、长按了消息列表里的用户头像的事件回调。

API 原型：

```
// TUIChatConfigMinimalist.java
public interface ChatEventListener {
    /**
    * Tells the listener a user's avatar in the chat list is clicked.
    * Returning true indicates this event has been intercepted, and Chat will not p
    * Returning false indicates this event is not intercepted, and Chat will contin
    */
    default boolean onUserIconClicked(View view, TUIMessageBean messageBean) {
        return false;
    }

    /**
    * Tells the listener a user's avatar in the chat list is long pressed.
    * Returning true indicates that this event has been intercepted, and Chat will
    * Returning false indicates that this event is not intercepted, and Chat will c
    */
    default boolean onUserIconLongClicked(View view, TUIMessageBean messageBean) {
        return false;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }
}
```

示例代码：

```
TUIChatConfigMinimalist.setChatEventListener(new ChatEventListener() {
    @Override
    public boolean onUserIconClicked(View view, TUIMessageBean messageBean) {
        // Customize your own action when user avatar is clicked.
        ToastUtil.toastShortMessage("onUserIconClicked");
        return true;
    }

    @Override
    public boolean onUserIconLongClicked(View view, TUIMessageBean messageBean) {
        // Customize your own action when user avatar is long pressed.
        ToastUtil.toastShortMessage("onUserIconLongClicked");
        return true;
    }
}
```

点击、长按消息列表里的消息

API 作用：用户点击、长按了消息列表里的消息的事件回调。

API 原型：

```
// TUIChatConfigMinimalist.java
public interface ChatEventListener {
     /**
      * Tells the listener a message in the chat list is clicked.
      * Returning true indicates that this event has been intercepted, and Chat will
      * Returning false indicates that this event is not intercepted, and Chat will
      */
    default boolean onMessageClicked(View view, TUIMessageBean messageBean) {
        return false;
    }

    /**
     * Tells the listener a message in the chat list is long pressed.
     * Returning true indicates that this event has been intercepted, and Chat will
     * Returning false indicates that this event is not intercepted, and Chat will
     */
    default boolean onMessageLongClicked(View view, TUIMessageBean messageBean) {
        return false;
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

示例代码：

```
TUIChatConfigMinimalist.setChatEventListener(new ChatEventListener() {
    @Override
    public boolean onMessageClicked(View view, TUIMessageBean messageBean) {
        // Customize your own action when message is clicked.
        ToastUtil.toastShortMessage("onMessageClicked");
        return true;
    }

    @Override
    public boolean onMessageLongClicked(View view, TUIMessageBean messageBean) {
        // Customize your own action when message is long pressed.
        ToastUtil.toastShortMessage("onMessageLongClicked");
        return true;
    }
}
```

消息样式相关

文本消息的颜色、字体

API 作用：设置发送、接收的文本消息的文字颜色和字体。针对所有的文本消息生效。

API 原型：

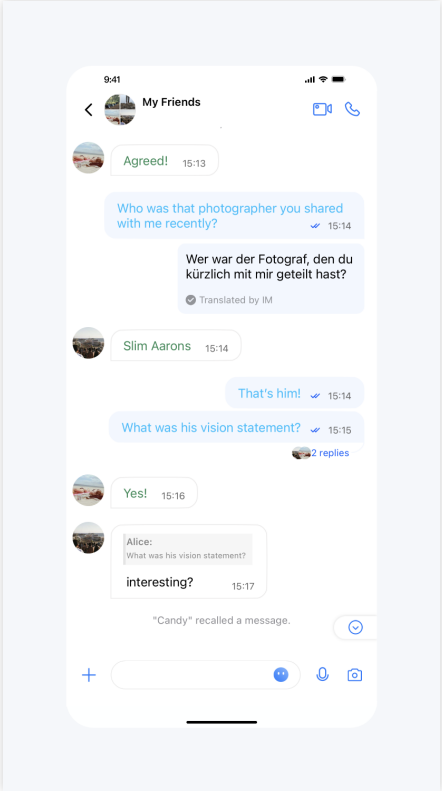
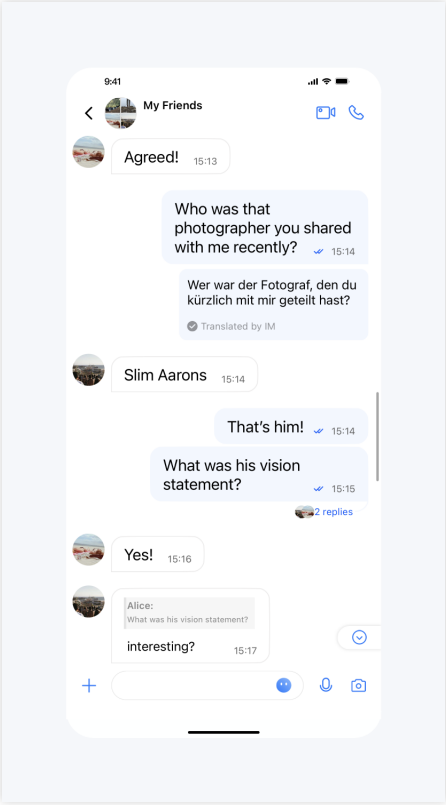
```
// TUIChatConfigMinimalist.java
/**
 * The color of send text message.
 */
public static void setSendTextMessageColor(int color)
/**
 * The font size of send text message.
 */
public static void setSendTextMessageFontSize(int size)
/*
 * The color of receive text message.
 */
public static void setReceiveTextMessageColor(int color)
/**
 * The font size of receive text message.
 */
public static void setReceiveTextMessageFontSize(int size)
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: After initializing the message list interface and before entering
TUIChatConfigMinimalist.setSendTextMessageColor(0xFF00BFFF);
TUIChatConfigMinimalist.setSendTextMessageFontSize(20);
TUIChatConfigMinimalist.setReceiveTextMessageColor(0xFF2E8B57);
TUIChatConfigMinimalist.setReceiveTextMessageFontSize(23);
```

设置效果：

设置文本消息文字颜色	设置文本消息字体	默认
		

系统通知消息字体、颜色和背景色

API 作用：设置系统通知消息文字的字体、颜色和背景色，针对所有系统通知消息生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * The text color of system message.
 */
public static void setSystemMessageTextColor(int color)
/**
 * The font size of system message.
```

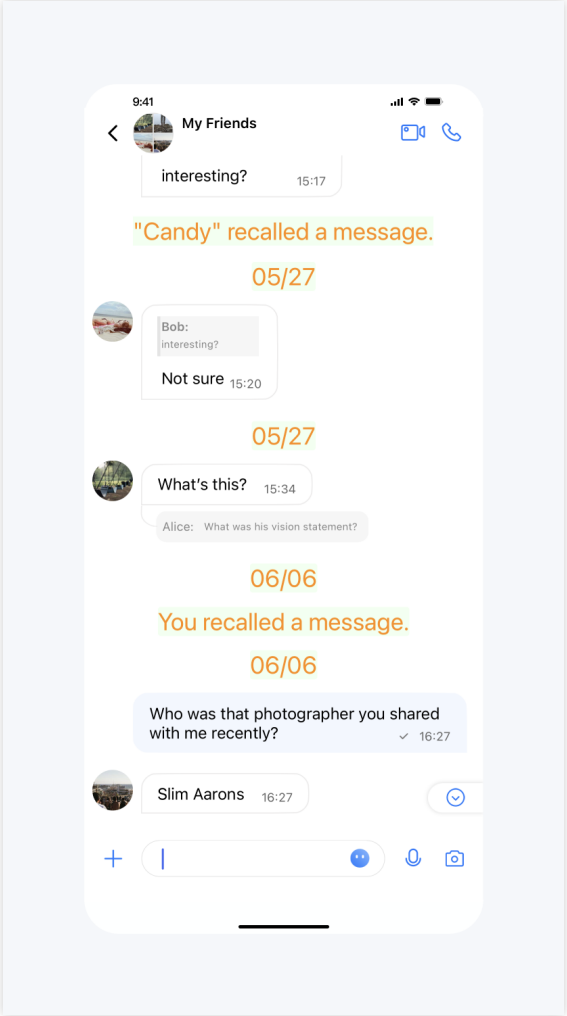
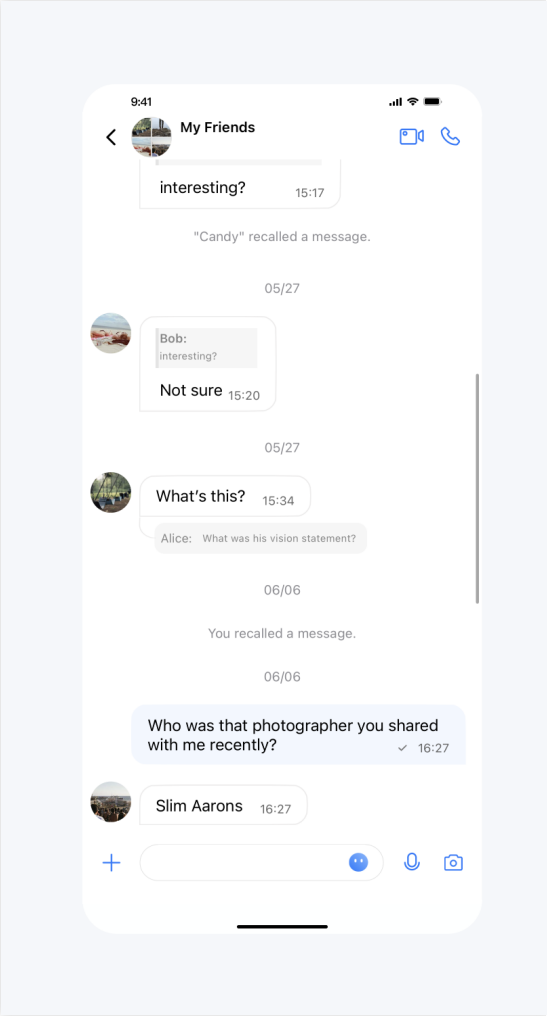
©2013-2025 Tencent Cloud International Pte. Ltd.

```
 */
public static void setSystemMessageFontSize(int size)
/**
 * The background of system message.
 */
public static void setSystemMessageBackground(Drawable drawable)
```

示例代码：

```
// When to call: After initializing the message list interface and before entering
TUIChatConfigMinimalist.setSystemMessageTextColor(0xFFFF8C00);
TUIChatConfigMinimalist.setSystemMessageFontSize(23);
TUIChatConfigMinimalist.setSystemMessageBackground(new ColorDrawable(0xFFF0FFF0));
```

设置效果：

设置系统通知消息文字的字体、颜色和背景色	默认
	

©2013-2025 Tencent Cloud International Pte. Ltd.

消息气泡相关

开启消息气泡展示

API 作用：开启消息气泡展示，针对所有聊天界面生效。

API 原型：

```
// TUIConfigMinimalist.java
/**
 * Enable the message display in the bubble style.
 * The default value is true.
 */
public static void setEnableMessageBubbleStyle(boolean enable)
```

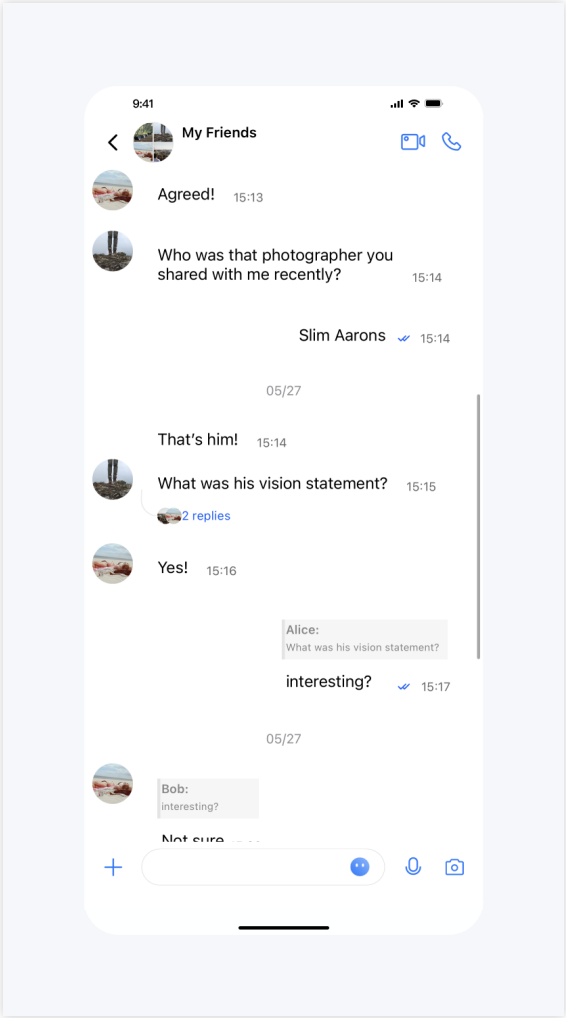
示例代码：

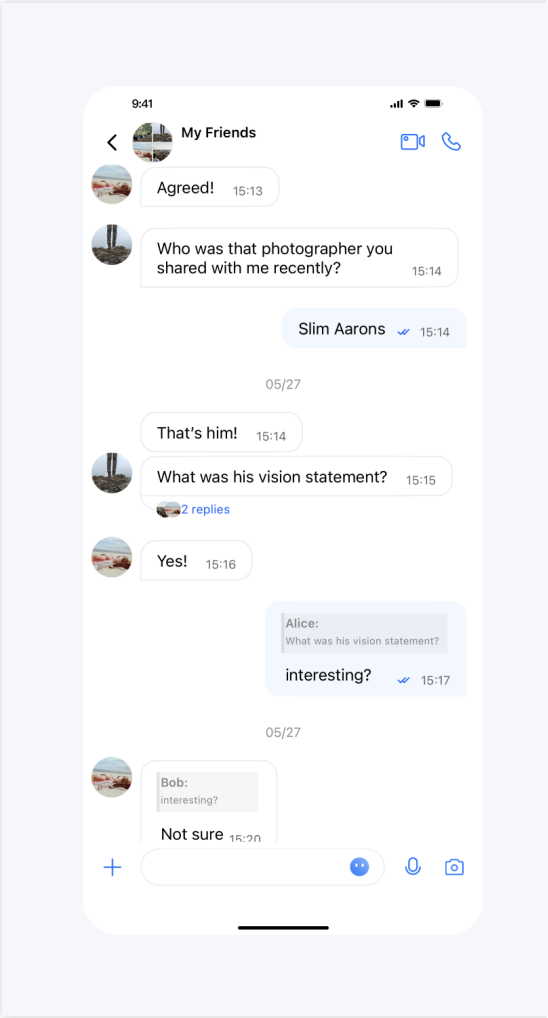
```
// When to call: After initializing the message list interface and before entering
TUIConfigMinimalist.setEnableMessageBubbleStyle(false);
```

设置效果：

不显示消息气泡	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





气泡背景图设置

API 作用：设置气泡背景图，针对所有聊天界面生效。

API 原型：

```
// TUIConfigMinimalist.java
/**
 * Set the background of the last sent message bubble in consecutive messages.
 */
public static void setSendLastBubbleBackground(Drawable drawable)
/**
 * Set the background of the non-last sent message bubble in consecutive message.
 */
public static void setSendBubbleBackground(Drawable drawable)
/**
 * Set the background of the last received message bubble in consecutive message.
 */
public static void setReceiveLastBubbleBackground(Drawable drawable)
/**
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 * Set the background of the non-last received message bubble in consecutive messag
 */
public static void setReceiveBubbleBackground(Drawable drawable)
/**
 * Set the light color when the message bubble needs to flicker.
 */
public static void setBubbleHighlightLightColor(int color)
/**
 * Set the dark color when the message bubble needs to flicker.
 */
public static void setBubbleHighlightDarkColor(int color)
```

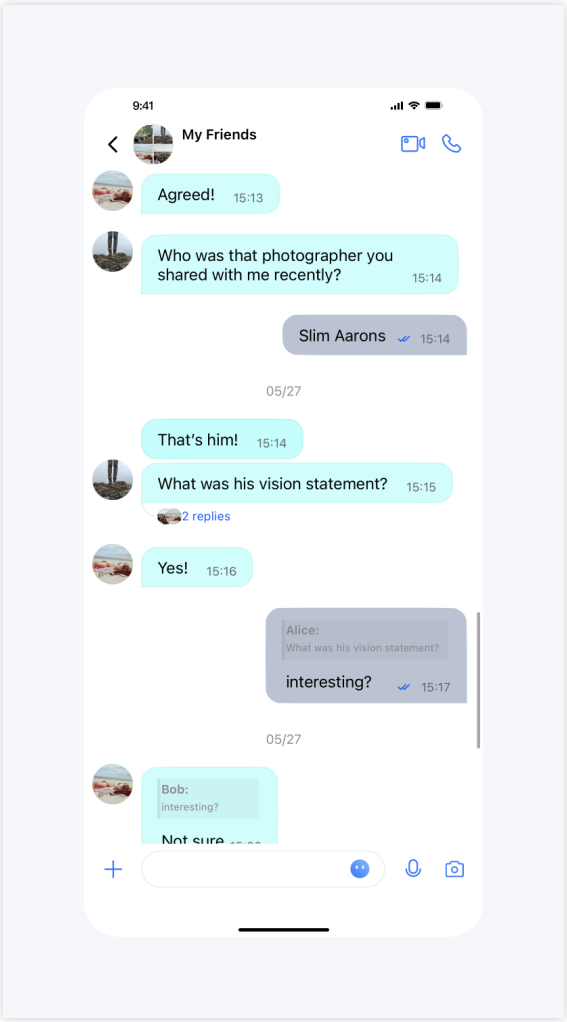
示例代码：

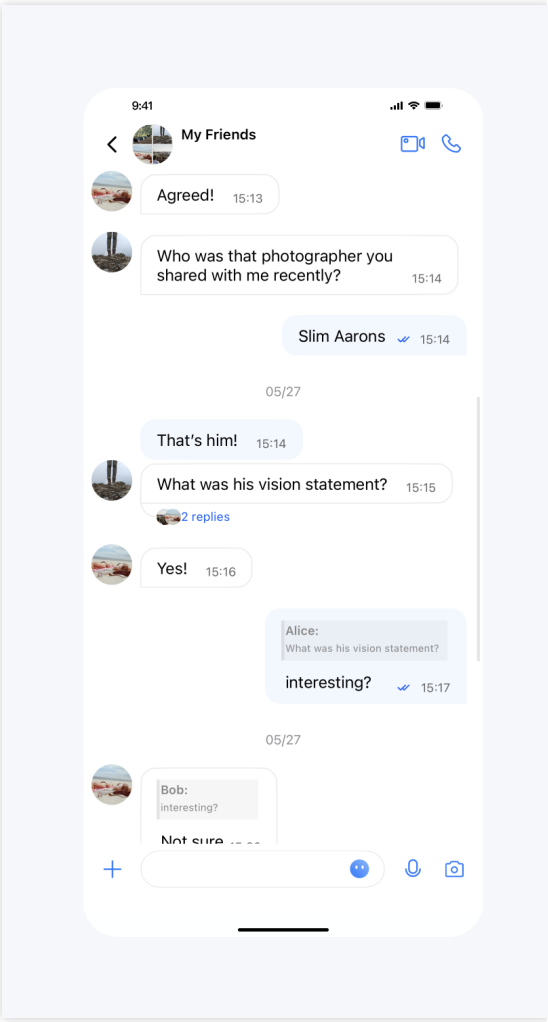
```
// When to call: After initializing the message list interface and before entering
TUIConfigMinimalist.setSendLastBubbleBackground(context.getDrawable(R.drawable.Send
TUIConfigMinimalist.setSendBubbleBackground(context.getDrawable(R.drawable.SenderTe
TUIConfigMinimalist.setReceiveLastBubbleBackground(context.getDrawable(R.drawable.R
TUIConfigMinimalist.setReceiveBubbleBackground(context.getDrawable(R.drawable.Recei
```

设置效果：

设置气泡背景图	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





输入栏相关

展示聊天界面输入框

API 作用：展示聊天界面输入框，针对所有聊天界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 *  Show the input bar in the message list interface.
 *  The default value is true.
 */
public static void setShowInputBar(boolean showInputBar)
```

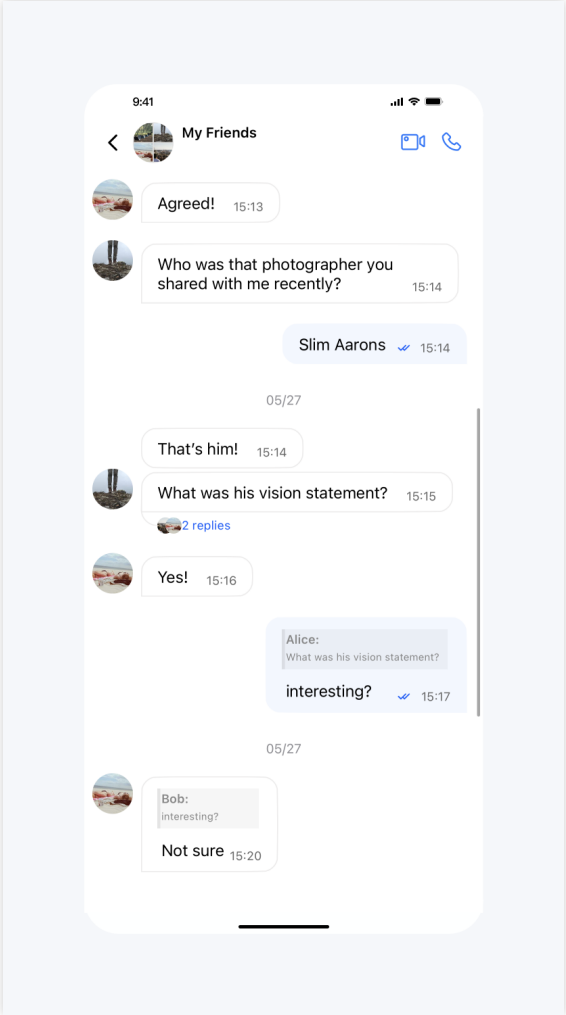
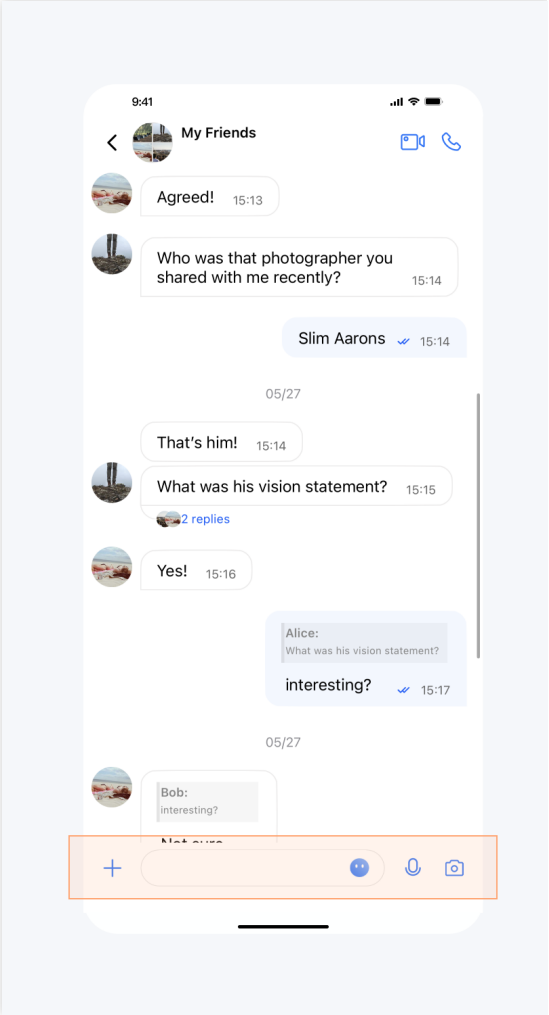
示例代码：

```
// When to call: After initializing the message list interface and before entering
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
TUIChatConfigMinimalist.setShowInputBar(false);
```

设置效果：

隐藏输入框	默认
	

隐藏更多菜单中选项（全局）

API 作用：隐藏更多菜单中的按钮，针对所有聊天界面生效。

API 原型：

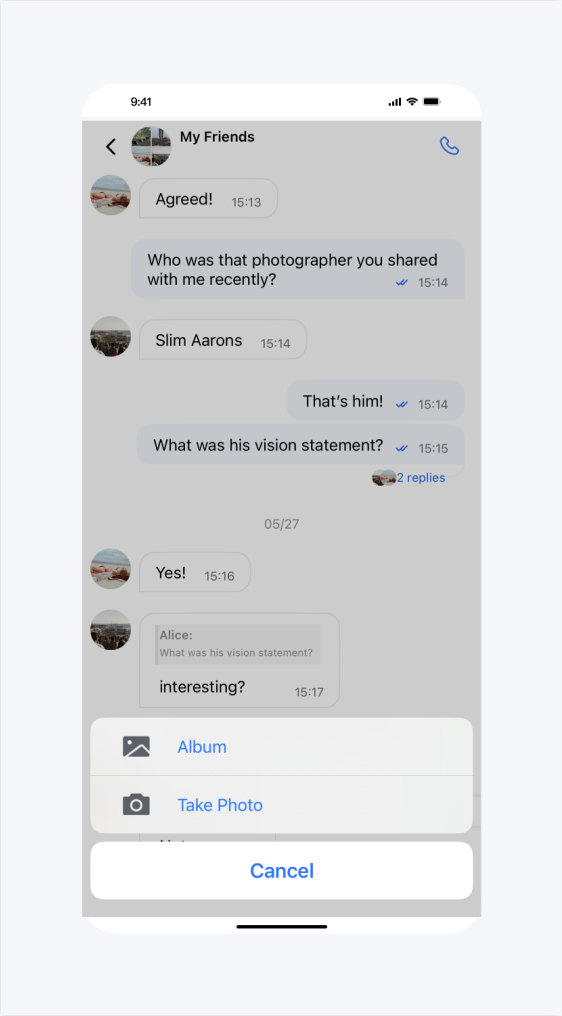
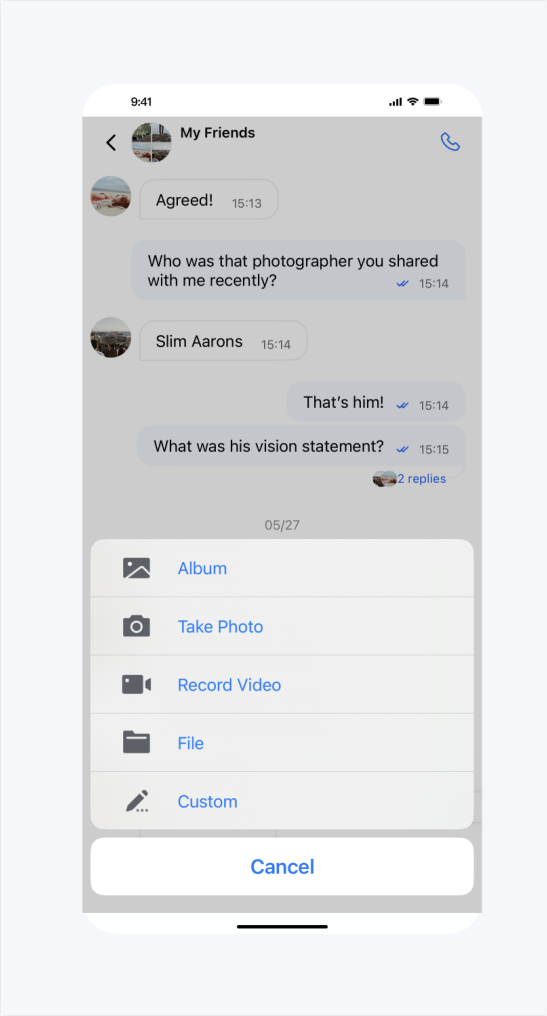
```
// TUIChatConfigMinimalist.java
/**
 *  Hide items in more menu.
 */
public static void hideItemsInMoreMenu(@InputMoreMenuItem int... items)
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: After initializing the message list interface and before entering
TUIChatConfigMinimalist.hideItemsInMoreMenu(TUIChatConfigMinimalist.CUSTOM,
                                            TUIChatConfigMinimalist.RECORD_VIDEO,
                                            TUIChatConfigMinimalist.FILE);
```

设置效果：

隐藏部分 item	默认
	

隐藏更多菜单中选项（局部）

API 作用：隐藏更多菜单中的按钮，针对指定聊天界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
public static final int CUSTOM = 1;
public static final int RECORD_VIDEO = 2;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
public static final int TAKE_PHOTO = 3;
public static final int ALBUM = 4;
public static final int FILE = 5;

@IntDef({CUSTOM, RECORD_VIDEO, TAKE_PHOTO, ALBUM, FILE})
public @interface InputMoreMenuItem {}

public interface ChatInputMoreDataSource {
    default @InputMoreMenuItem List<Integer> inputBarShouldHideItemsInMoreMenuOfInf
        return new ArrayList<>();
    }
}
```

示例代码：

```
// When to call: After initializing the message list interface and before entering
TUIChatConfigMinimalist.setChatInputMoreDataSource(new TUIChatConfigMinimalist.Chat
    @Override
    public List<Integer> inputBarShouldHideItemsInMoreMenuOfInfo(ChatInfo chatInfo)
        return Arrays.asList(TUIChatConfigMinimalist.CUSTOM, TUIChatConfigMinimalis
                             TUIChatConfigMinimalist.TAKE_PHOTO, TUIChatConfigMinim
                             TUIChatConfigMinimalist.FILE);
        }
});
```

更多菜单添加选项（局部）

API 作用：向更多菜单添加选项，针对指定聊天界面生效。

API 原型：

```
// TUIChatConfigMinimalist.java
public interface ChatInputMoreDataSource {
    default List<InputMoreItem> inputBarShouldAddNewItemToMoreMenuOfInfo(ChatInfo c
        return new ArrayList<>();
    }
}
```

示例代码：

```
// When to call: After initializing the message list interface and before entering
TUIChatConfigMinimalist.setChatInputMoreDataSource(new TUIChatConfigMinimalist.Chat
    @Override
    public List<InputMoreItem> inputBarShouldAddNewItemToMoreMenuOfInfo(ChatInfo ch
        InputMoreItem inputMoreItem = new InputMoreItem() {
            @Override
            public void onAction(String chatInfoId, int chatType) {
              ToastUtil.toastShortMessage("on click");
```

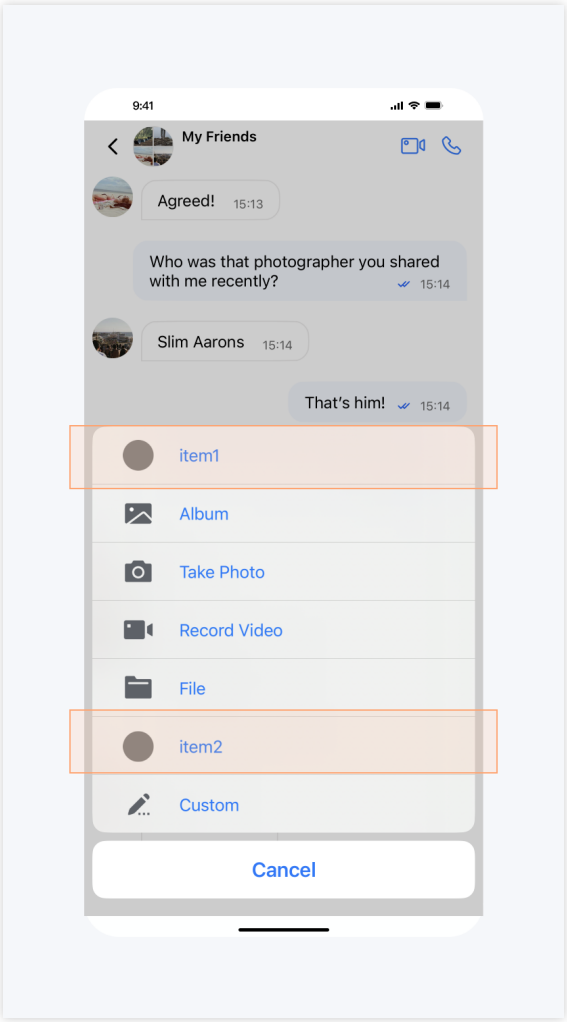
©2013-2025 Tencent Cloud International Pte. Ltd.

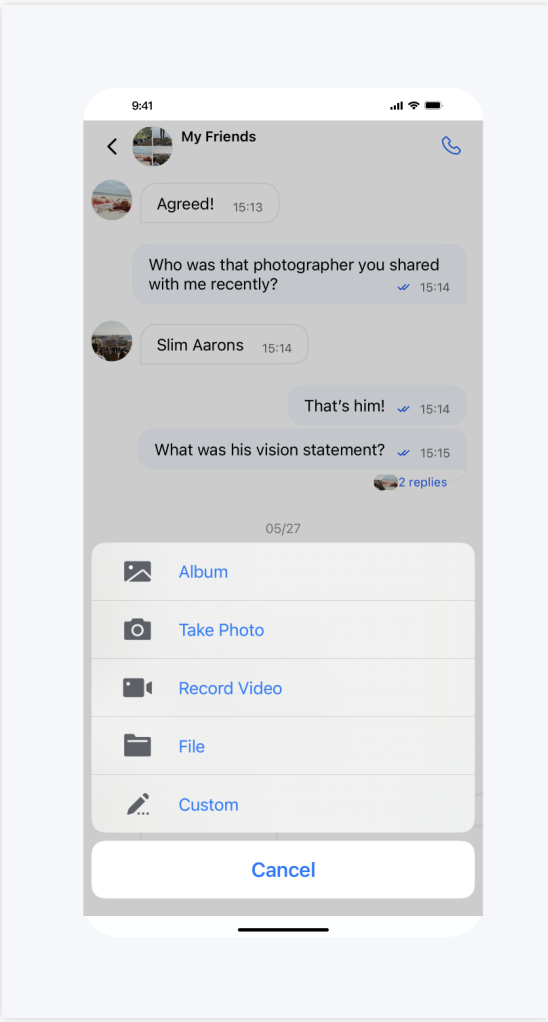
```
            }
        };
        inputMoreItem.setName("item1");
        inputMoreItem.setPriority(10000);
        inputMoreItem.setIconResId(R.drawable.ic_launcher);

        InputMoreItem inputMoreItem2 = new InputMoreItem() {
            @Override
            public void onAction(String chatInfoId, int chatType) {
              ToastUtil.toastShortMessage("on click");
            }
        };
        inputMoreItem2.setName("item1");
        inputMoreItem2.setPriority(8000);
        inputMoreItem2.setIconResId(R.drawable.ic_launcher);

        return Arrays.asList(inputMoreItem, inputMoreItem2);
    }
});
```

设置效果：

添加 item	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





添加表情组

API 作用：向表情菜单中添加表情组，针对所有聊天界面生效。使用场景请参考文档《添加自定义表情》。

API 原型：

```
// TUIChatConfigMinimalist.java
/**
 * Add sticker group.
 */
public static void addStickerGroup(int groupID, FaceGroup<? extends ChatFace> faceG
```

示例代码：

```
// When to call: After initializing the message list interface and before entering
FaceGroup faceGroup1 = new FaceGroup();
faceGroup1.setPageColumnCount(5);
faceGroup1.setPageRowCount(2);
faceGroup1.setGroupID(1);
faceGroup1.setFaceGroupIconUrl("file:///android_asset/4350/yz00@2x.gif");
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
faceGroup1.setGroupName("4350");
for (int i = 0; i <= 17; i++) {
      CustomFace customFace = new CustomFace();
      String index = "" + i;
      if (i < 10) {
          index = "0" + i;
      }
      customFace.setAssetPath("4350/yz" + index + "@2x.gif");
      String faceKey = "yz" + index;
      customFace.setFaceKey(faceKey);
      customFace.setWidth(170);
      customFace.setHeight(170);
      faceGroup1.addFace(faceKey, customFace);
}
TUIChatConfigMinimalist.addStickerGroup(1, faceGroup1);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

会话列表

最近更新时间：2024-10-30 10:59:38

下文将向您展示如何设置会话列表界面自定义选项及其效果。

设置会话列表、cell 背景色

API 作用：设置会话列表、cell、置顶 cell 的背景色。

API 原型：

```
// TUIConversationConfigMinimalist.java
/**
 *  Background color of conversation list.
 */
public static void setListBackground(Drawable listBackground)
/**
 *  Background color of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public static void setCellBackground(Drawable cellBackground)
/**
 *  Background color of pinned cell in conversation list.
 *  This configuration takes effect in all pinned cells.
 */
public static void setPinnedCellBackground(Drawable pinnedCellBackground)
```

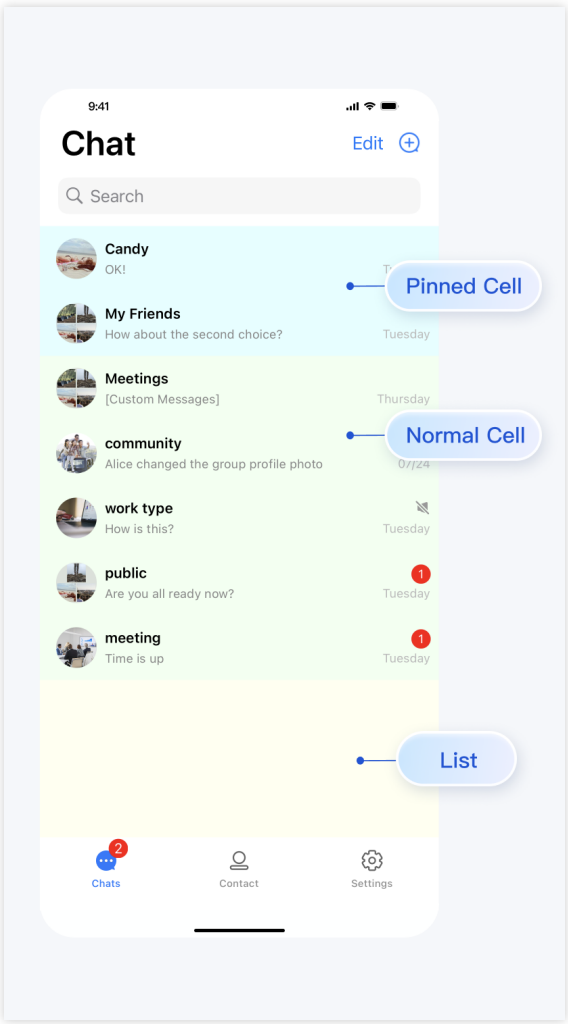
示例代码：

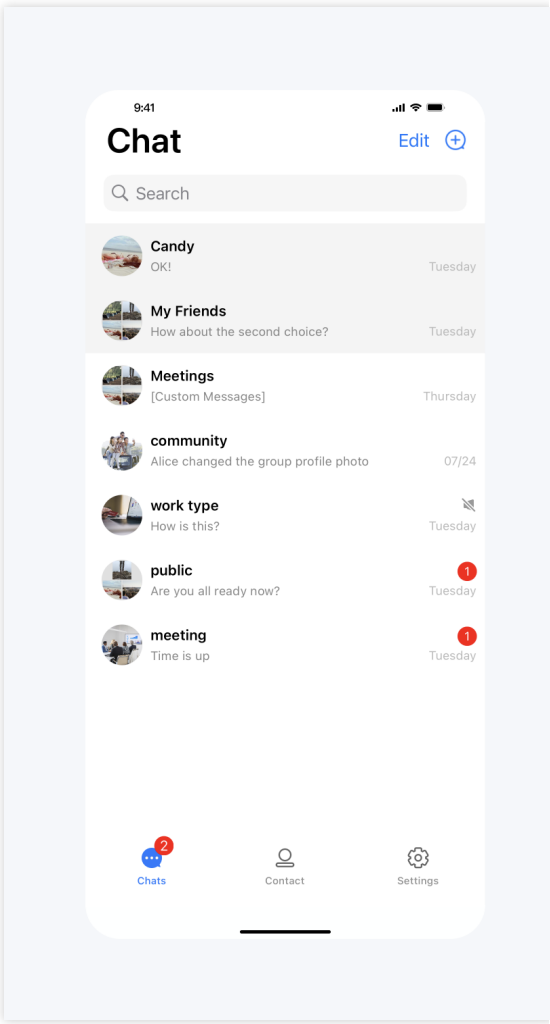
```
// When to call: Before initializing conversation list.
TUIConversationConfigMinimalist.setListBackground(new ColorDrawable(Color.BLUE));
TUIConversationConfigMinimalist.setCellBackground(new ColorDrawable(Color.LTGRAY));
TUIConversationConfigMinimalist.setPinnedCellBackground(new ColorDrawable(Color.GRA
```

设置效果：

设置背景色	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





设置会话列表 cell 字体

API 作用：设置会话列表 cell 上的标题、副标题、时间文字的字体。针对所有 cell 生效。

API 原型：

```
// TUIConversationConfigMinimalist.java
/**
 *  Font of title label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public static void setCellTitleLabelFontSize(int cellTitleLabelFontSize)
/**
 *  Font of subtitle label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public static void setCellSubtitleLabelFontSize(int cellSubtitleLabelFontSize)
/**
```

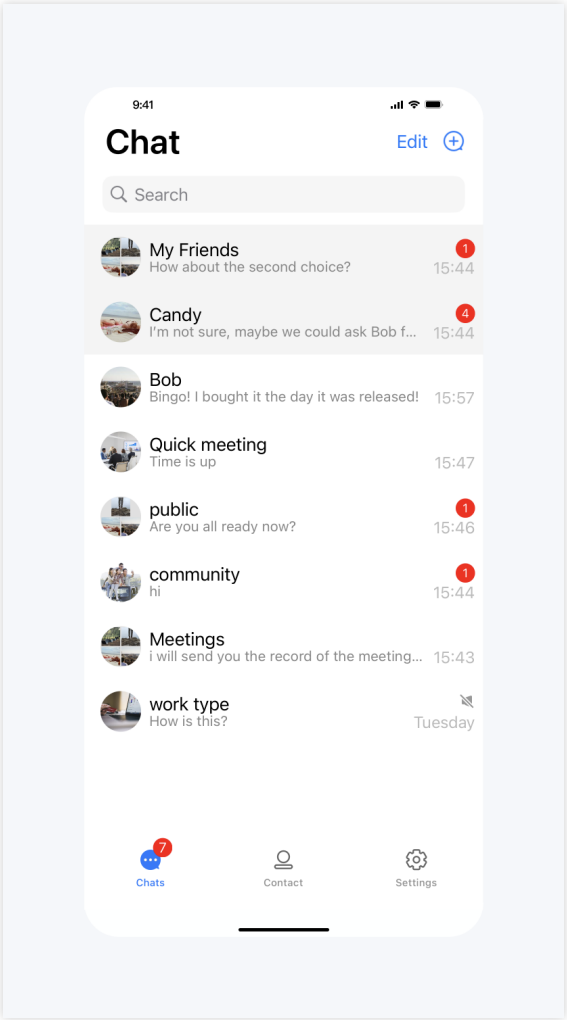
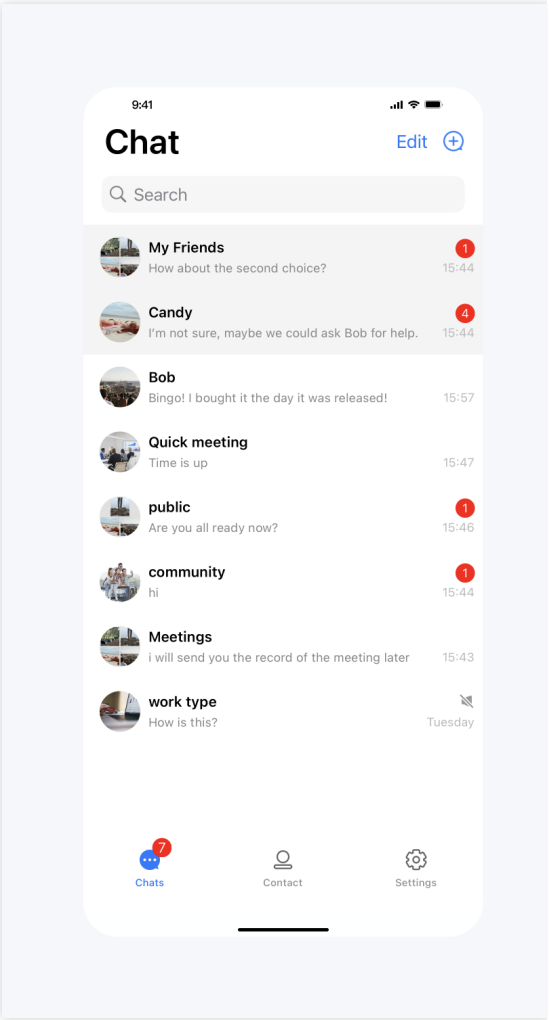
©2013-2025 Tencent Cloud International Pte. Ltd.

```
 *  Font of time label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public static void setCellTimeLabelFontSize(int cellTimeLabelFontSize)
```

示例代码：

```
// When to call: Before initializing conversation list.
TUIConversationConfigMinimalist.setCellTitleLabelFontSize(18);
TUIConversationConfigMinimalist.setCellSubtitleLabelFontSize(14);
TUIConversationConfigMinimalist.setCellTimeLabelFontSize(16);
```

设置效果：

设置字体	默认
	

©2013-2025 Tencent Cloud International Pte. Ltd.

展示未读红点

API 作用：展示 cell 上的未读消息红点 icon。针对所有 cell 生效。

API 原型：

```
// TUIConversationConfigMinimalist.java
/**
 *  Display unread count icon in each conversation cell.
 *  The default value is true.
 */
public static void setShowCellUnreadCount(boolean showCellUnreadCount)
```

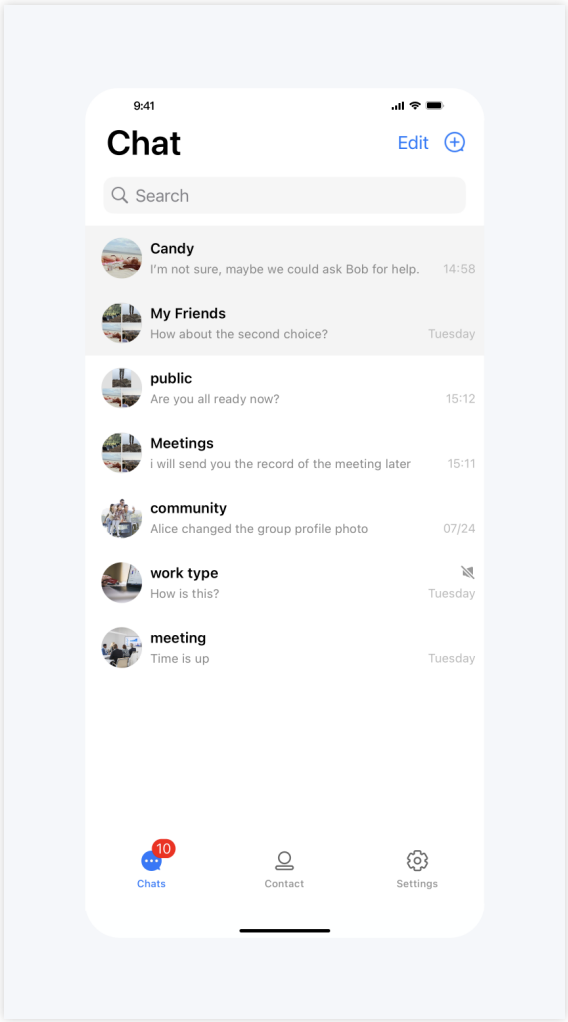
示例代码：

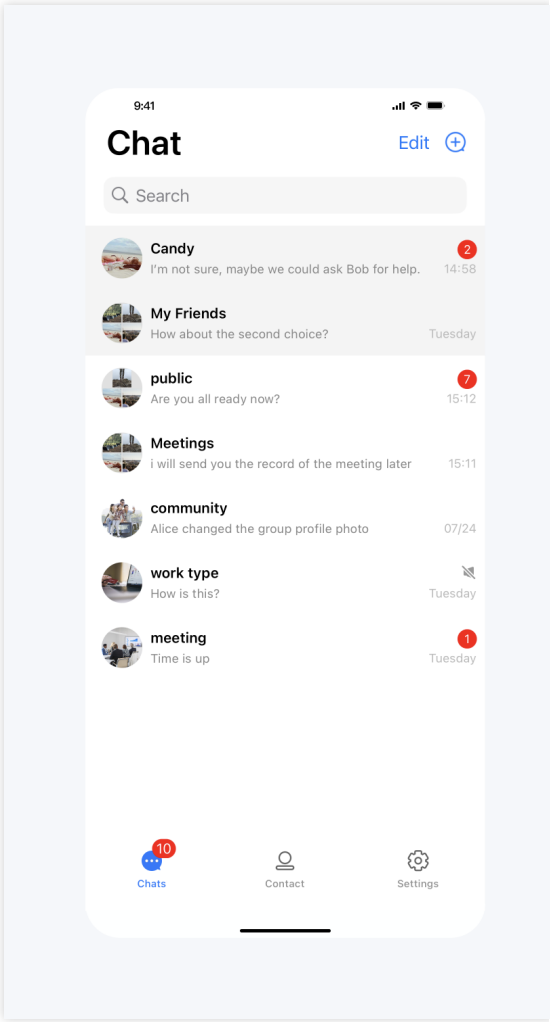
```
// When to call: Before initializing conversation list.
TUIConversationConfigMinimalist.setShowCellUnreadCount(false);
```

设置效果：

不展示会话 cell 上的未读红点	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





展示在线状态

API 作用：展示 cell 里用户头像上的在线状态 icon。针对所有 cell 生效。

API 原型：

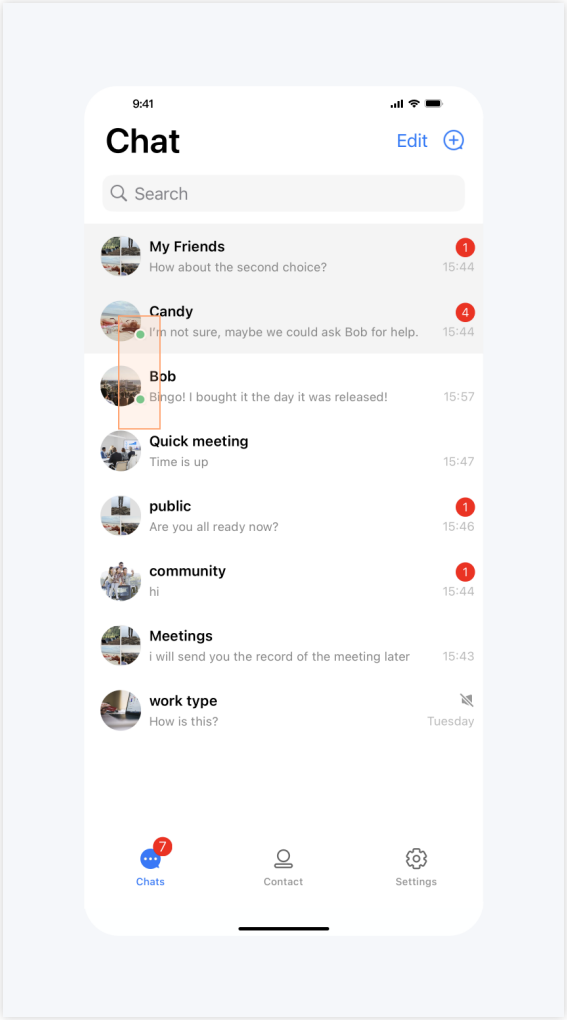
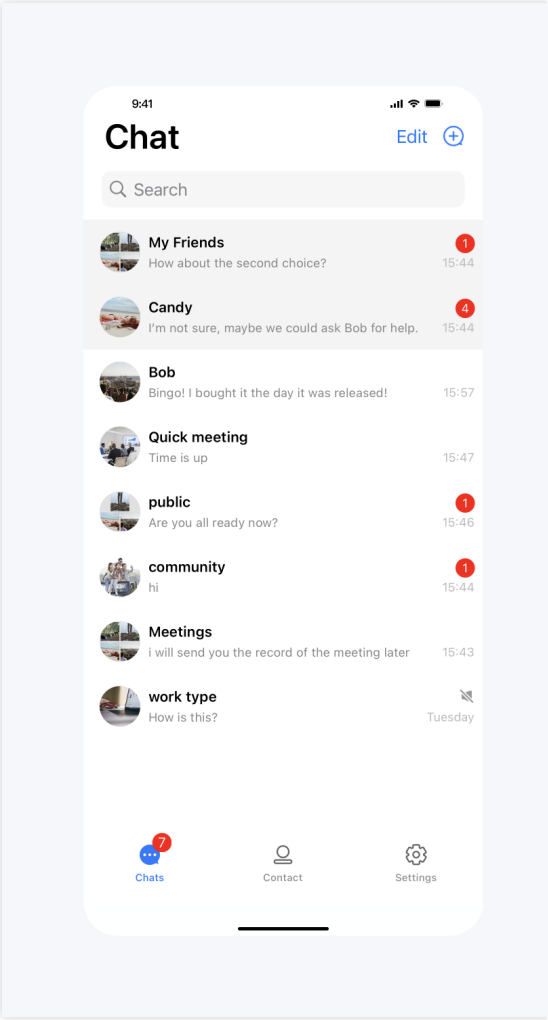
```
// TUIConversationConfigMinimalist.java
/**
 *  Display user's online status icon in conversation list.
 *  The default value is false.
 */
public static void setShowUserOnlineStatusIcon(boolean showUserOnlineStatusIcon)
```

示例代码：

```
// When to call: Before initializing conversation list.
TUIConversationConfigMinimalist.setShowUserOnlineStatusIcon(true);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

设置效果：

展示在线状态	默认
	

会话更多菜单选项自定义

API 作用：隐藏会话更多菜单选项、向会话更多菜单添加选项。针对指定会话生效。

API 原型：

```
// TUIConversationConfigMinimalist.java
public interface ConversationMenuItemDataSource {
    /**
    * Implement this method to add new items.
    */
    default List<ConversationPopMenuItem> conversationShouldAddNewItemsToMoreMenu(C
        return new ArrayList<>();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }

    /**
    * Implement this method to hide items in more menu.
    */
    default @ConversationMenuItem List<Integer> conversationShouldHideItemsInMoreMe
        return new ArrayList<>();
    }
}
```

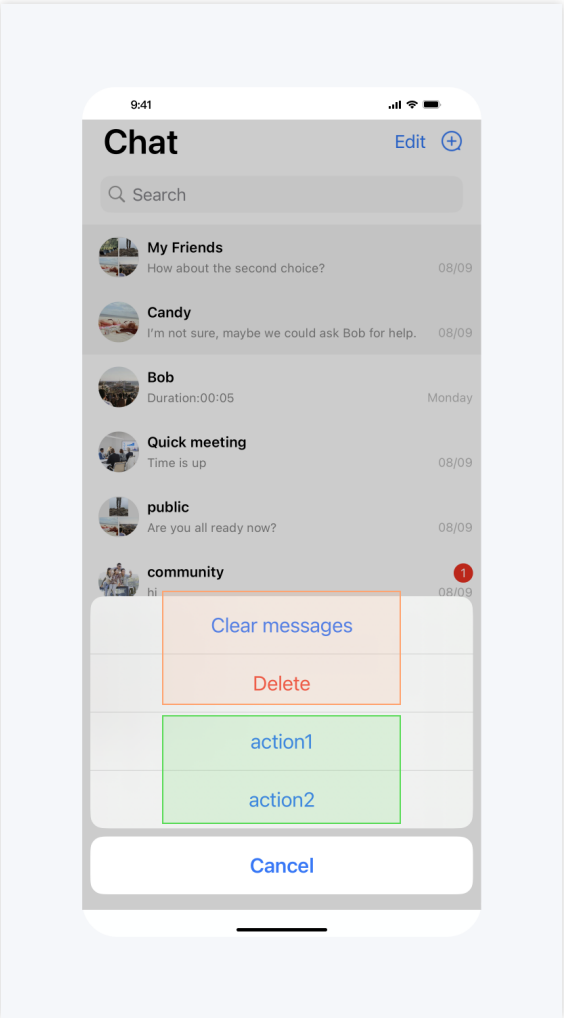
示例代码：

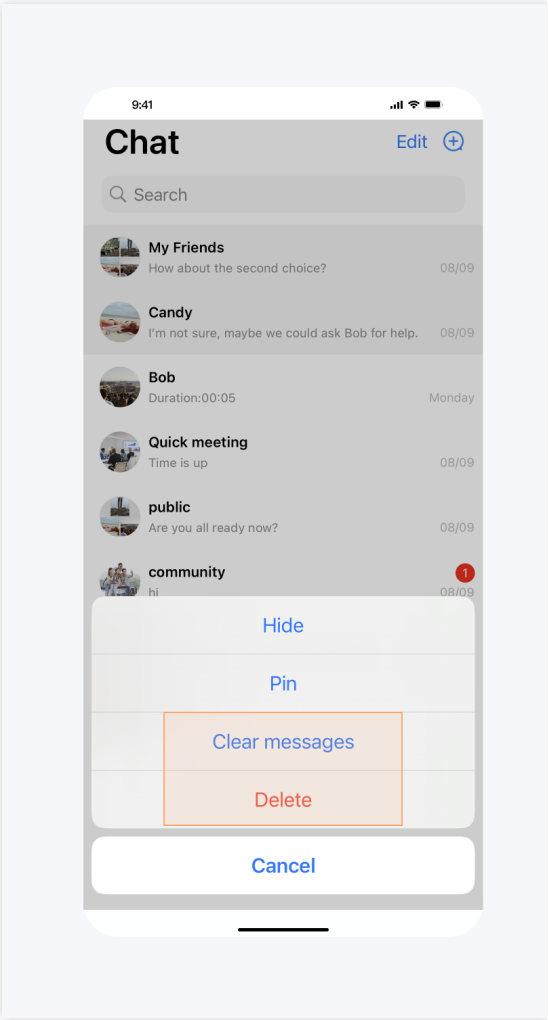
```
// When to call: Before initializing conversation list.
TUIConversationConfigMinimalist.setConversationMenuItemDataSource(new TUIConversati
    @Override
    public List<Integer> conversationShouldHideItemsInMoreMenu(ConversationInfo con
        return Arrays.asList(TUIConversationConfigMinimalist.HIDE,
                             TUIConversationConfigMinimalist.PIN);
    }

    @Override
    public List<ConversationPopMenuItem> conversationShouldAddNewItemsToMoreMenu(Co
        ConversationPopMenuItem item = new ConversationPopMenuItem();
        item.text = "action1";
        item.iconResId = R.drawable.ic_launcher;
        item.onClickListener = new View.OnClickListener() {
            @Override
                public void onClick(View v) {
                    ToastUtil.toastShortMessage("action1 clicked");
                }
        };

        ConversationPopMenuItem item2 = new ConversationPopMenuItem();
                item2.text = "action2";
                item2.iconResId = R.drawable.ic_launcher;
                item2.onClickListener = new View.OnClickListener() {
            @Override
                public void onClick(View v) {
                                ToastUtil.toastShortMessage("action2 clicked");
                }
                };
        return Arrays.asList(item, item2);
        }
});
```

设置效果：

隐藏、添加选项	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





©2013-2025 Tencent Cloud International Pte. Ltd.

群组设置

最近更新时间：2025-01-21 18:08:22

下文将向您展示如何隐藏群设置选项及其效果。

隐藏群设置选项

API 作用：隐藏群组设置选项。针对所有群组生效。

API 原型：

```
// GroupConfig.java
public static final int MUTE_AND_PIN = 1;
public static final int MANAGE = 2;
public static final int ALIAS = 3;
public static final int BACKGROUND = 4;
public static final int MEMBERS = 5;
public static final int CLEAR_CHAT_HISTORY = 6;
public static final int DELETE_AND_LEAVE = 7;
public static final int TRANSFER = 8;
public static final int DISMISS = 9;

@IntDef({MUTE_AND_PIN, MANAGE, ALIAS, BACKGROUND, MEMBERS, CLEAR_CHAT_HISTORY, DELE
public @interface Items {}

/**
 * Hide items in group config interface.
 */
public static void hideItemsInGroupConfig(@Items int... items)
```

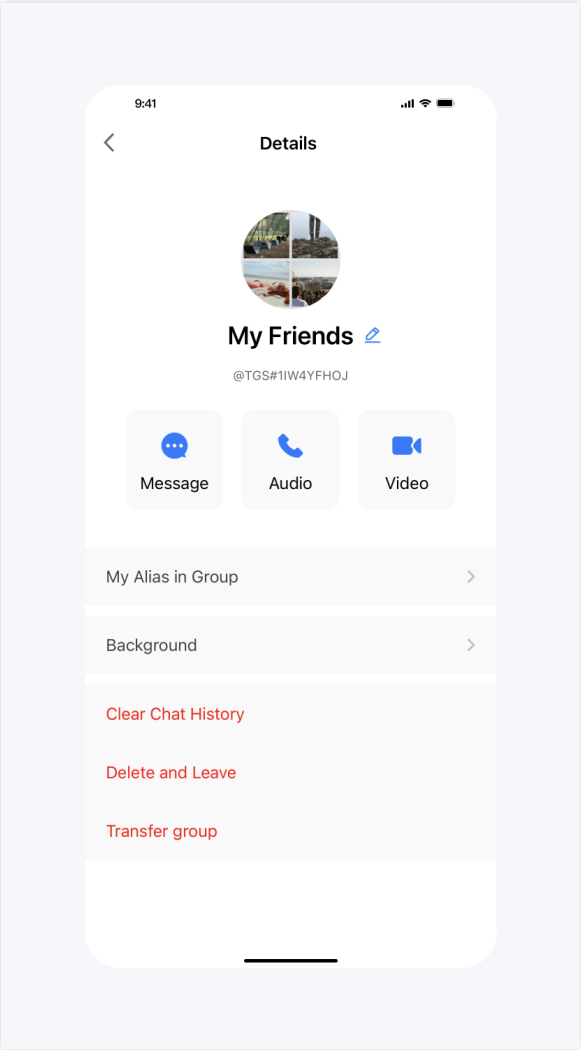
示例代码：

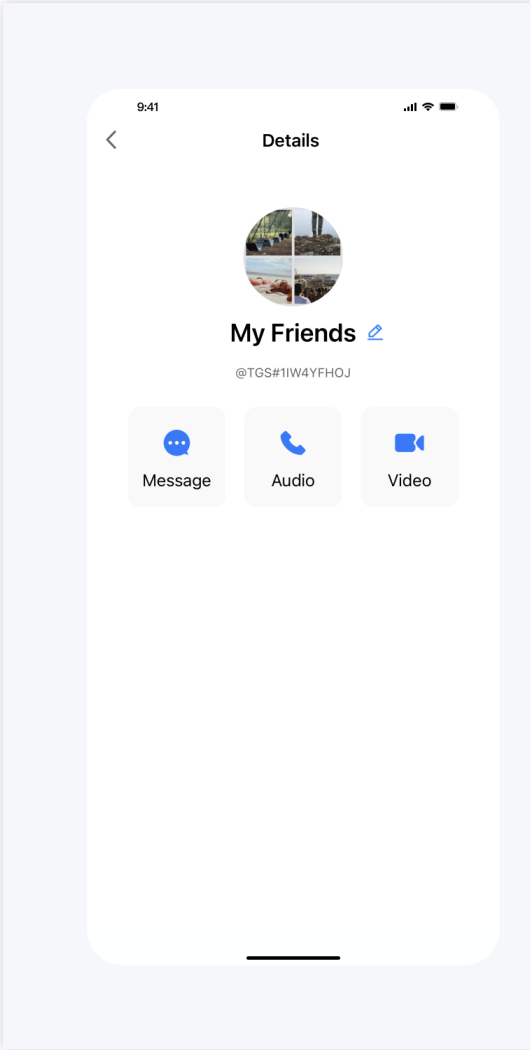
```
// When to call: Before initializing group setting interface.
GroupConfig.hideItemsInGroupConfig(GroupConfig.MUTE_AND_PIN,
                                   GroupConfig.MANAGE);
```

设置效果：

隐藏部分选项	隐藏全部选项

©2013-2025 Tencent Cloud International Pte. Ltd.





©2013-2025 Tencent Cloud International Pte. Ltd.

联系人

最近更新时间：2025-01-21 18:08:22

下文将向您展示如何隐藏联系人设置选项及其效果。

隐藏联系人设置选项

API 作用：隐藏联系人设置选项。针对所有联系人生效。

API 原型：

```
// FriendConfig.java
public static final int ALIAS = 1;
public static final int MUTE_AND_PIN = 2;
public static final int BACKGROUND = 3;
public static final int BLOCK = 4;
public static final int CLEAR_CHAT_HISTORY = 5;
public static final int DELETE = 6;
public static final int ADD_FRIEND = 7;

@IntDef({ALIAS, MUTE_AND_PIN, BACKGROUND, BLOCK, CLEAR_CHAT_HISTORY, DELETE, ADD_FR
private @interface Items {}

/**
 * Hide items in contact config interface.
 */
public static void hideItemsInContactConfig(@Items int... items)
```

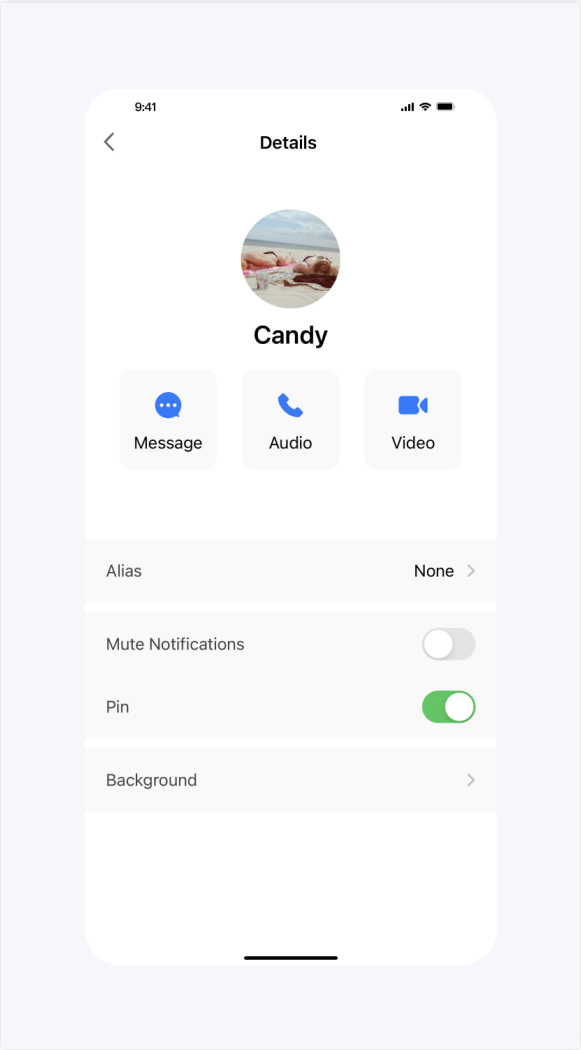
示例代码：

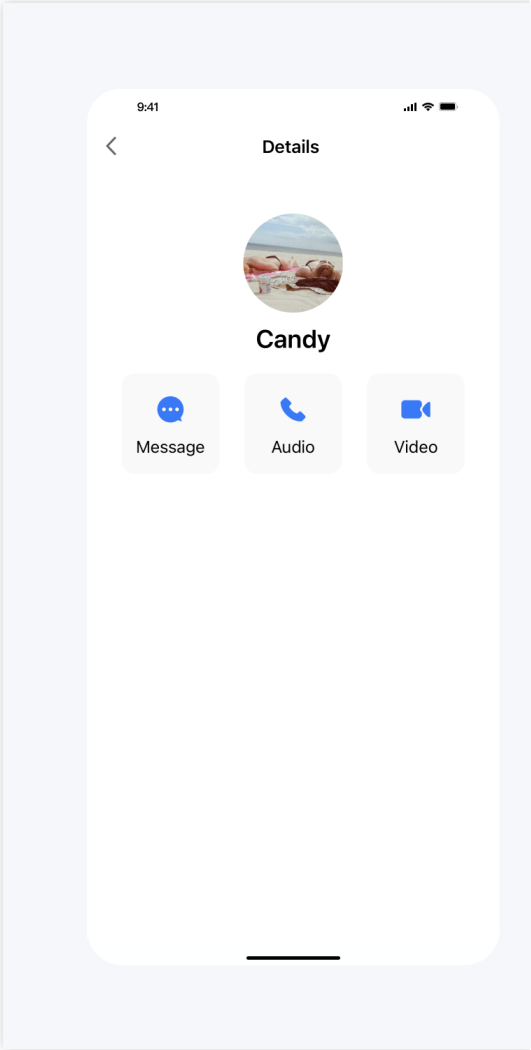
```
// When to call: Before initializing contact setting interface.
// Valid for contacts.
FriendConfig.hideItemsInContactConfig(FriendConfig.BLOCK,
                                      FriendConfig.CLEAR_CHAT_HISTORY,
                                      FriendConfig.DELETE);
// Valid for strange users who have not been added to the contact.
FriendConfig.hideItemsInContactConfig(FriendConfig.ADD_FRIEND);
```

对联系人设置效果：

隐藏部分选项	隐藏全部选项

©2013-2025 Tencent Cloud International Pte. Ltd.

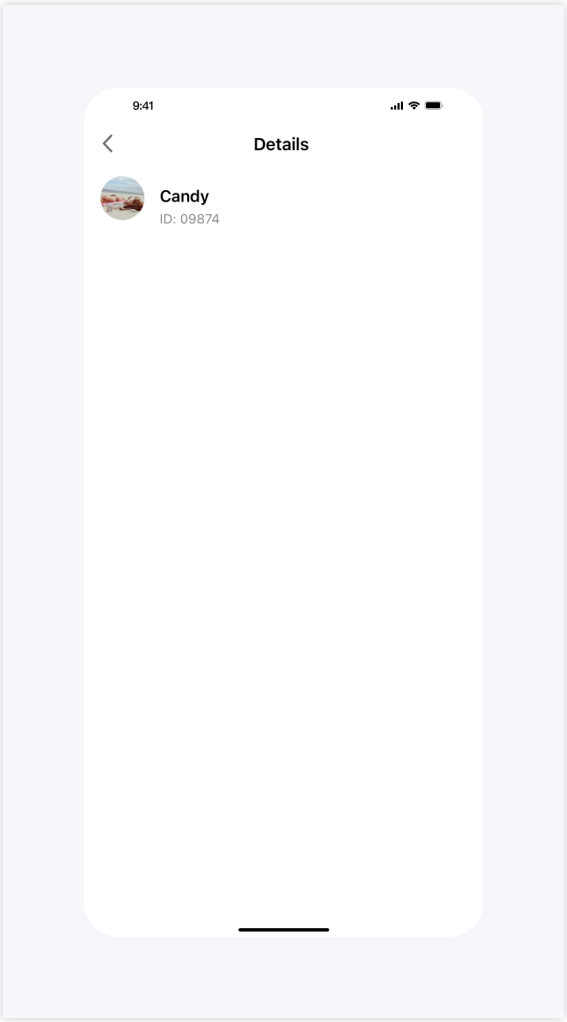


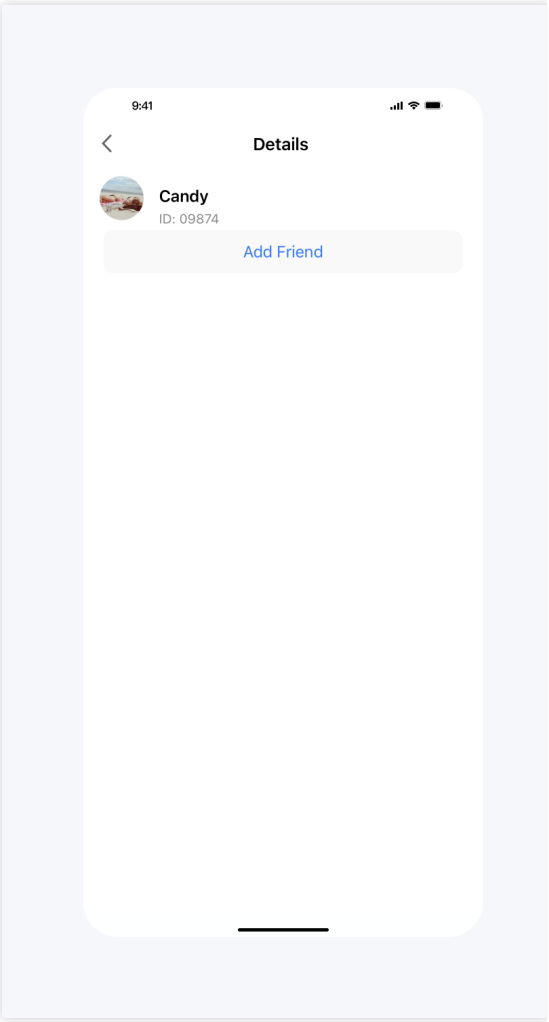


对尚未添加到联系人的陌生用户设置效果：

隐藏添加好友	默认

©2013-2025 Tencent Cloud International Pte. Ltd.





©2013-2025 Tencent Cloud International Pte. Ltd.

iOS
全局

最近更新时间：2025-05-23 12:02:31

下文将向您展示如何设置全局自定义选项及其效果。

切换 TUIKit 语言

API 作用：切换 TUIKit 语言。
方法原型：
Swift
Objective-C

```
// TUIConfig_Minimalist.swift
/**
 * Switch the language of TUIKit.
 * The currently supported languages are "en", "zh-Hans", and "ar".
 */
static func switchLanguage(to targetLanguage: String) {
    TUIGlobalization.setPreferredLanguage(targetLanguage)
}
```

```
// TUIConfig_Minimalist.h
/**
 * Switch the language of TUIKit.
 * The currently supported languages are "en", "zh-Hans", and "ar".
 */
+ (void)switchLanguageToTarget:(NSString *)targetLanguage;
```

示例代码：
Swift
Objective-C

```
// When to call: Before initializing the TUIKit interface.
TUIConfig_Minimalist.switchLanguage(to: "en")
```

```
// When to call: Before initializing the TUIKit interface.
[TUIConfig_Minimalist switchLanguageToTarget:@"en"];
```

设置效果：

©2013-2025 Tencent Cloud International Pte. Ltd.

设置为英文	设置为阿拉伯语

开启 TUIKit 内建提示

API 作用：设置是否开启 TUIKit 内建 toast 提示。默认开启，TUIKit 组件会显示内建的提示。
方法原型：
Swift
Objective-C

```
// TUIConfig_Minimalist.swift
/**
* Show the toast prompt built in TUIKit.
* The default value is YES.
*/
static func enableToast(_ enable: Bool) {
    TUIConfig.default().enableToast = enable
}
```

```
// TUIConfig_Minimalist.h
/**
 *  Show the toast prompt built in TUIKit.
 *  The default value is YES.
 */
+ (void)enableToast:(BOOL)enable;
```

示例代码：
Swift
Objective-C

```
// When to call: Before initializing the TUIKit interface.
TUIConfig_Minimalist.enableToast(false)
```

```
// When to call: Before initializing the TUIKit interface.
[TUIConfig_Minimalist enableToast:NO];
```

设置效果：

开启 Toast 提示	关闭 Toast 提示

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

聊天界面

最近更新时间：2025-05-23 12:02:32

下文将向您展示如何设置聊天界面自定义选项及其效果。

消息列表相关

聊天界面背景色、背景图片

API 作用：设置聊天界面消息列表背景色、背景图片，针对所有聊天界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
var backgroundColor: UIColor? {
    get {
        return TUIChatConfig.shared.backgroudColor
    }
    set {
        TUIChatConfig.shared.backgroudColor = newValue ?? .black
    }
}
var backgroundImage: UIImage? {
    get {
        return TUIChatConfig.shared.backgroudImage
    }
    set {
        TUIChatConfig.shared.backgroudImage = newValue ?? UIImage()
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Customize the backgroud color of message list interface.
 * This configuration takes effect in all message list interfaces.
 */
@property (nonatomic, strong) UIColor *backgroudColor;
/**
 * Customize the backgroud image of message list interface.
 * This configuration takes effect in all message list interfaces.
 */
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
@property (nonatomic, strong) UIImage *backgroudImage;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the message list interface.
TUIChatConfig_Minimalist.shared.backgroundColor = UIColor.tui_color(withHex: "#E1FF
TUIChatConfig_Minimalist.shared.backgroundImage = UIImage.init(named: "your_backgro
```

```
// When to call: Before initializing the message list interface.
[TUIChatConfig_Minimalist sharedConfig].backgroudColor = [UIColor tui_colorWithHex:
[TUIChatConfig_Minimalist sharedConfig].backgroudImage = [UIImage imageNamed:@"your
```

设置效果：

背景色	设置背景图片	默认

用户头像类型、圆角半径

API 作用：设置用户头像类型、圆角半径。目前支持的类型有矩形、圆形、圆角矩形，其中只有圆角矩形类型会使用到圆角半径。针对消息列表、会话列表和联系人列表生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 *  Customize the style of avatar.
 *  The default value is TUIAvatarStyleCircle.
 *  This configuration takes effect in all avatars.
 */
public var avatarStyle: TUIAvatarStyle_Minimalist {
    get {
        return TUIAvatarStyle_Minimalist(rawValue: TUIConfig.default().avatarType.r
    }
    set {
        TUIConfig.default().avatarType = TUIKitAvatarType(rawValue: newValue.rawVal
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
/**
 *  Customize the corner radius of the avatar.
 *  This configuration takes effect in all avatars.
 */
public var avatarCornerRadius: CGFloat {
    get {
        return TUIConfig.default().avatarCornerRadius
    }
    set {
        TUIConfig.default().avatarCornerRadius = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
typedef NS_ENUM(NSInteger, TUIAvatarStyle) {
    TUIAvatarStyleRectangle,
    TUIAvatarStyleCircle,
    TUIAvatarStyleRoundedRectangle,
};

/**
 *  Customize the style of avatar.
 *  The default value is TUIAvatarStyleCircle.
 *  This configuration takes effect in all avatars.
 */
@property (nonatomic, assign) TUIAvatarStyle avatarStyle;
/**
 *  Customize the corner radius of the avatar.
 *  This configuration takes effect in all avatars.
 */
@property (nonatomic, assign) CGFloat avatarCornerRadius;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the TUIKit interfaces.
TUIChatConfig_Minimalist.shared.avatarStyle = .rectangle
// Set cornerRadius
TUIChatConfig_Minimalist.shared.avatarStyle = .roundedRectangle
TUIChatConfig_Minimalist.shared.avatarCornerRadius = 10
```

```
// When to call: Before initializing the TUIKit interfaces.
[TUIChatConfig_Minimalist sharedConfig].avatarStyle = TUIAvatarStyleRectangle;
// Set cornerRadius
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
[TUIChatConfig_Minimalist sharedConfig].avatarStyle = TUIAvatarStyleRoundedRectangl
[TUIChatConfig_Minimalist sharedConfig].avatarCornerRadius = 10;
```

设置效果：

默认圆形头像	设置圆角矩形头像	设置矩形头像

设置群头像展示九宫格

API 作用：设置群头像展示九宫格。针对消息列表、会话列表和联系人列表生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Display the group avatar in the nine-square grid style.
 * The default value is YES.
 * This configuration takes effect in all groups.
 */
public var enableGroupGridAvatar: Bool {
    get {
        return TUIConfig.default().enableGroupGridAvatar
    }
    set {
        TUIConfig.default().enableGroupGridAvatar = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Display the group avatar in the nine-square grid style.
 * The default value is YES.
 * This configuration takes effect in all groups.
 */
@property (nonatomic, assign) BOOL enableGroupGridAvatar;
```

示例代码：

Swift

Objective-C

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: Before initializing the TUIKit interfaces.
TUIChatConfig_Minimalist.shared.enableGroupGridAvatar = false
```

```
// When to call: Before initializing the TUIKit interfaces.
[TUIChatConfig_Minimalist sharedConfig].enableGroupGridAvatar = NO;
```

设置效果：

设置群头像不展示九宫格	默认

开启正在输入状态指示

API 作用：开启“正在输入”状态指示。针对所有 1v1 聊天消息界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 *  Enable the display "Alice is typing..." on one-to-one chat interface.
 *  The default value is YES.
 *  This configuration takes effect in all one-to-one chat message list interfaces.
 */
public var enableTypingIndicator: Bool {
    get {
        return TUIChatConfig.shared.enableTypingStatus
    }
    set {
        TUIChatConfig.shared.enableTypingStatus = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 *  Enable the display "Alice is typing..." on one-to-one chat interface.
 *  The default value is YES.
 *  This configuration takes effect in all one-to-one chat message list interfaces.
 */
@property (nonatomic, assign) BOOL enableTypingIndicator;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the message list interface.
TUIChatConfig_Minimalist.shared.enableTypingIndicator = false
```

```
// When to call: Before initializing the message list interface.
[TUIChatConfig_Minimalist sharedConfig].enableTypingIndicator = NO;
```

设置效果：

开启“正在输入”	不开启“正在输入”

开启消息已读回执

API 作用：开启消息已读回执，开启后可以在消息详情中查看已读信息。针对所有消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 *  When sending a message, set this flag to require message read receipt.
 *  The default value is NO.
 *  This configuration takes effect in all chat message list interfaces.
 */
public var isMessageReadReceiptNeeded: Bool {
    get {
        return TUIChatConfig.shared.msgNeedReadReceipt
    }
    set {
        TUIChatConfig.shared.msgNeedReadReceipt = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 *  When sending a message, set this flag to require message read receipt.
 *  The default value is NO.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 *  This configuration takes effect in all chat message list interfaces.
 */
@property (nonatomic, assign) BOOL isMessageReadReceiptNeeded;
```

示例代码：

Swift

Objective-C

```
// When to call: Before sending messages.
TUIChatConfig_Minimalist.shared.isMessageReadReceiptNeeded = true
```

```
// When to call: Before sending messages.
[TUIChatConfig_Minimalist sharedConfig].isMessageReadReceiptNeeded = YES;
```

设置效果：

开启消息已读回执	不开启消息已读回执

隐藏长按消息菜单按钮

API 作用：隐藏长按消息菜单中的指定按钮，针对所有聊天消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Hide the items in the pop-up menu when user presses the message.
 */
public class func hideItemsWhenLongPressMessage(_ items: TUIChatItemWhenLongPressMe
    let value = items.rawValue
    TUIChatConfig.shared.enablePopMenuReplyAction = (value & TUIChatItemWhenLongPre
    TUIChatConfig.shared.enablePopMenuEmojiReactAction = (value & TUIChatItemWhenLo
    TUIChatConfig.shared.enablePopMenuReferenceAction = (value & TUIChatItemWhenLon
    TUIChatConfig.shared.enablePopMenuPinAction = (value & TUIChatItemWhenLongPress
    TUIChatConfig.shared.enablePopMenuRecallAction = (value & TUIChatItemWhenLongPr
    TUIChatConfig.shared.enablePopMenuTranslateAction = (value & TUIChatItemWhenLon
    TUIChatConfig.shared.enablePopMenuConvertAction = (value & TUIChatItemWhenLongP
    TUIChatConfig.shared.enablePopMenuForwardAction = (value & TUIChatItemWhenLongP
    TUIChatConfig.shared.enablePopMenuSelectAction = (value & TUIChatItemWhenLongPr
    TUIChatConfig.shared.enablePopMenuCopyAction = (value & TUIChatItemWhenLongPres
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    TUIChatConfig.shared.enablePopMenuDeleteAction = (value & TUIChatItemWhenLongPr
    TUIChatConfig.shared.enablePopMenuInfoAction = (value & TUIChatItemWhenLongPres
}
```

```
// TUIChatConfig_Minimalist.h
typedef NS_OPTIONS(NSInteger, TUIChatItemWhenLongPressMessage_Minimalist) {
    TUIChatItemWhenLongPressMessage_Minimalist_None = 0,
    TUIChatItemWhenLongPressMessage_Minimalist_Reply = 1 << 0,
    TUIChatItemWhenLongPressMessage_Minimalist_EmojiReaction = 1 << 1,
    TUIChatItemWhenLongPressMessage_Minimalist_Quote = 1 << 2,
    TUIChatItemWhenLongPressMessage_Minimalist_Pin = 1 << 3,
    TUIChatItemWhenLongPressMessage_Minimalist_Recall = 1 << 4,
    TUIChatItemWhenLongPressMessage_Minimalist_Translate = 1 << 5,
    TUIChatItemWhenLongPressMessage_Minimalist_Convert = 1 << 6,
    TUIChatItemWhenLongPressMessage_Minimalist_Forward = 1 << 7,
    TUIChatItemWhenLongPressMessage_Minimalist_Select = 1 << 8,
    TUIChatItemWhenLongPressMessage_Minimalist_Copy = 1 << 9,
    TUIChatItemWhenLongPressMessage_Minimalist_Delete = 1 << 10,
    TUIChatItemWhenLongPressMessage_Minimalist_Info = 1 << 11,
};

/**
 * Hide the items in the pop-up menu when user presses the message.
 */
+ (void)hideItemsWhenLongPressMessage:(TUIChatItemWhenLongPressMessage_Minimalist)i
```

示例代码：

Swift

Objective-C

```
// When to call: Before displaying the pop-up menu when user presses the message.
TUIChatConfig_Minimalist.hideItemsWhenLongPressMessage([.reply, .recall, .select])
```

```
// When to call: Before displaying the pop-up menu when user presses the message.
[TUIChatConfig_Minimalist hideItemsWhenLongPressMessage:TUIChatItemWhenLongPressMes
```

设置效果：

不隐藏任何按钮	隐藏 Forward 按钮

隐藏视频通话、视频通话按钮

©2013-2025 Tencent Cloud International Pte. Ltd.

API 作用：隐藏消息列表顶部的音频、视频通话按钮，针对所有聊天消息界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 *  Hide the "Video Call" button in the message list header.
 *  The default value is NO.
 */
public var hideVideoCallButton: Bool {
    get {
        return !TUIChatConfig.shared.enableVideoCall
    }
    set {
        TUIChatConfig.shared.enableVideoCall = !newValue
    }
}

/**
 *  Hide the "Audio Call" button in the message list header.
 *  The default value is NO.
 */
public var hideAudioCallButton: Bool {
    get {
        return !TUIChatConfig.shared.enableAudioCall
    }
    set {
        TUIChatConfig.shared.enableAudioCall = !newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 *  Hide the "Video Call" button in the message list header.
 *  The default value is NO.
 */
@property (nonatomic, assign) BOOL hideVideoCallButton;
/**
 *  Hide the "Audio Call" button in the message list header.
 *  The default value is NO.
 */
@property (nonatomic, assign) BOOL hideAudioCallButton;
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
// When to call: Before entering the message list interface.
TUIChatConfig_Minimalist.shared.hideVideoCallButton = false
TUIChatConfig_Minimalist.shared.hideAudioCallButton = false
```

```
// When to call: Before entering the message list interface.
[TUIChatConfig_Minimalist sharedConfig].hideVideoCallButton = YES;
[TUIChatConfig_Minimalist sharedConfig].hideAudioCallButton = YES;
```

设置效果：

隐藏视频通话按钮	隐藏音频通话按钮	默认

开启音视频通话浮窗

API 作用：开启音视频通话浮窗。开启后，您可以将音视频通话界面以小窗口的形式漂浮在聊天界面。针对所有音视频通话界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Turn on audio and video call floating windows,
 * The default value is YES.
 */
public var enableFloatWindowForCall: Bool {
    get {
        return TUIChatConfig.shared.enableFloatWindowForCall
    }
    set {
        TUIChatConfig.shared.enableFloatWindowForCall = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 * Turn on audio and video call floating windows,
 * The default value is YES.
 */
@property (nonatomic, assign) BOOL enableFloatWindowForCall;
```

示例代码：

Swift

Objective-C

```
// When to call: Before entering the message list interface.
TUIChatConfig_Minimalist.shared.enableFloatWindowForCall = false
```

```
// When to call: Before entering the message list interface.
[TUIChatConfig_Minimalist sharedConfig].enableFloatWindowForCall = NO;
```

开启音视频通话多端登录

API 作用：开启音视频通话多端登录。针对所有音视频通话生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Enable multi-terminal login function for audio and video calls
 * The default value is NO.
 */
public var enableMultiDeviceForCall: Bool {
    get {
        return TUIChatConfig.shared.enableMultiDeviceForCall
    }
    set {
        TUIChatConfig.shared.enableMultiDeviceForCall = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Enable multi-terminal login function for audio and video calls
 * The default value is NO.
 */
@property (nonatomic, assign) BOOL enableMultiDeviceForCall;
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
// When to call: Before entering the message list interface.
TUIChatConfig_Minimalist.shared.enableMultiDeviceForCall = true
```

```
// When to call: Before entering the message list interface.
[TUIChatConfig_Minimalist sharedConfig].enableMultiDeviceForCall = YES;
```

设置消息列表顶部自定义 View

API 作用：设置聊天界面顶部自定义 View，针对所有聊天消息界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Add a custom view at the top of the chat interface.
 * This view will be displayed at the top of the message list and will not slide up
 */
public class func setCustomTopView(_ view: UIView) {
    TUIBaseChatViewController_Minimalist.customTopView = view
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Add a custom view at the top of the chat interface.
 * This view will be displayed at the top of the message list and will not slide up
 */
+ (void)setCustomTopView:(UIView *)view;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the message list interface.
// tipsView is your customized view.
TUIChatConfig_Minimalist.shared.setCustomTopView(tipsView)
```

```
// When to call: Before initializing the message list interface.
// tipsView is your customized view.
[TUIChatConfig_Minimalist setCustomTopView:tipsView];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

设置效果：

设置自定义view	默认

设置消息是否不计入未读

API 作用：设置即将发送的消息不更新会话未读数，针对所有消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is NO.
 */
public var isExcludedFromUnreadCount: Bool {
    get {
        return TUIConfig.default().isExcludedFromUnreadCount
    }
    set {
        TUIConfig.default().isExcludedFromUnreadCount = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is NO.
 */
@property (nonatomic, assign) BOOL isExcludedFromUnreadCount;
```

示例代码：

Swift

Objective-C

```
// When to call: Before sending messages.
TUIChatConfig_Minimalist.shared.isExcludedFromUnreadCount = true
```

```
// When to call: Before sending messages.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
[TUIChatConfig_Minimalist sharedConfig].isExcludedFromUnreadCount = YES;
```

设置消息是否不更新会话 lastMsg

API 作用：设置即将发送的消息不更新会话 lastMsg，针对所有消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is NO.
 */
public var isExcludedFromLastMessage: Bool {
    get {
        return TUIConfig.default().isExcludedFromLastMessage
    }
    set {
        TUIConfig.default().isExcludedFromLastMessage = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Set this parameter when the sender sends a message, and the receiver will not up
 * The default value is NO.
 */
@property (nonatomic, assign) BOOL isExcludedFromLastMessage;
```

示例代码：

Swift

Objective-C

```
// When to call: Before sending messages.
TUIChatConfig_Minimalist.shared.isExcludedFromLastMessage = true
```

```
// When to call: Before sending messages.
[TUIChatConfig_Minimalist sharedConfig].isExcludedFromLastMessage = YES;
```

消息撤回时间间隔

API 作用：设置消息撤回时间，针对所有消息生效。

API 原型：

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Time interval within which a message can be recalled after being sent.
 * The default value is 120 seconds.
 * If you want to adjust this configuration, please modify the setting on Chat Cons
 */
public var timeIntervalForAllowedMessageRecall: UInt {
    get {
        return TUIChatConfig.shared.timeIntervalForMessageRecall
    }
    set {
        TUIChatConfig.shared.timeIntervalForMessageRecall = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Time interval within which a message can be recalled after being sent.
 * The default value is 120 seconds.
 * If you want to adjust this configuration, please modify the setting on Chat Cons
 */
@property (nonatomic, assign) NSUInteger timeIntervalForAllowedMessageRecall;
```

示例代码：

Swift

Objective-C

```
// When to call: Before sending messages.
TUIChatConfig_Minimalist.shared.timeIntervalForAllowedMessageRecall = 90
```

```
// When to call: Before sending messages.
[TUIChatConfig_Minimalist sharedConfig].timeIntervalForAllowedMessageRecall = 90;
```

语音、视频消息最长录制时长

API 作用：设置语音、视频消息最长录制时长，针对所有语音、视频消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
/**
 * Maximum audio recording duration, no more than 60s.
 * The default value is 60 seconds.
 */
public var maxAudioRecordDuration: TimeInterval {
    get {
        return TUIChatConfig.shared.maxAudioRecordDuration
    }
    set {
        TUIChatConfig.shared.maxAudioRecordDuration = newValue
    }
}

/**
 * Maximum video recording duration, no more than 15s.
 * The default value is 15 seconds.
 */
public var maxVideoRecordDuration: TimeInterval {
    get {
        return TUIChatConfig.shared.maxVideoRecordDuration
    }
    set {
        TUIChatConfig.shared.maxVideoRecordDuration = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Maximum audio recording duration, no more than 60s.
 * The default value is 60 seconds.
 */
@property (nonatomic, assign) CGFloat maxAudioRecordDuration;
/**
 * Maximum video recording duration, no more than 15s.
 * The default value is 15 seconds.
 */
@property (nonatomic, assign) CGFloat maxVideoRecordDuration;
```

示例代码：

Swift

Objective-C

```
// When to call: Before recording audio or video messages.
TUIChatConfig_Minimalist.shared.maxAudioRecordDuration = 10
TUIChatConfig_Minimalist.shared.maxVideoRecordDuration = 10
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: Before recording audio or video messages.
[TUIChatConfig_Minimalist sharedConfig].maxAudioRecordDuration = 10;
[TUIChatConfig_Minimalist sharedConfig].maxVideoRecordDuration = 10;
```

开启自定义铃声

API 作用：设置 Android 设备收到消息时的铃声为内置的自定义铃声，针对所有消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Enable custom ringtone.
 * This config takes effect only for Android devices.
 */
public var enableAndroidCustomRing: Bool {
    get {
        return TUIConfig.default().enableCustomRing
    }
    set {
        TUIConfig.default().enableCustomRing = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Enable custom ringtone.
 * This config takes effect only for Android devices.
 */
@property (nonatomic, assign) BOOL enableAndroidCustomRing;
```

示例代码：

Swift

Objective-C

```
// When to call: Before sending messages.
TUIChatConfig_Minimalist.shared.enableAndroidCustomRing = true
```

```
// When to call: Before sending messages.
[TUIChatConfig_Minimalist sharedConfig].enableAndroidCustomRing = YES;
```

开启语音消息扬声器播放

©2013-2025 Tencent Cloud International Pte. Ltd.

API 作用：设置播放语音消息时默认使用扬声器而不是听筒播放。针对所有语音消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Call this method to use speakers instead of handsets by default when playing voi
 */
public static func setPlayingSoundMessageViaSpeakerByDefault() {
    if TUIVoiceMessageCellData.getAudioplaybackStyle() == .handset {
        TUIVoiceMessageCellData.changeAudioPlaybackStyle()
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Call this method to use speakers instead of handsets by default when playing voi
 */
+ (void)setPlayingSoundMessageViaSpeakerByDefault;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the Message interface.
TUIChatConfig_Minimalist.setPlayingSoundMessageViaSpeakerByDefault()
```

```
// When to call: Before initializing the Message interface.
[TUIChatConfig_Minimalist setPlayingSoundMessageViaSpeakerByDefault];
```

注册自定义消息

API 作用：注册自定义消息。使用场景请参考文档《添加自定义消息》。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Register custom message.
 * - Parameters:
 *   - businessID: Customized message‘s businessID, which is unique.
 *   - messageCellClassName: Customized message's MessagCell class name.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
     *    - messageCellDataClassName: Customized message's MessagCellData class name.
     */
    public func registerCustomMessage(businessID: String, messageCellClassName: String,
        TUIChatConfig.shared.registerCustomMessage(businessID: businessID, messageCellC
    }
```

```
    // TUIChatConfig_Minimalist.h
    /**
     * Register custom message.
     * - Parameters:
     *    - businessID: Customized message‘s businessID, which is unique.
     *    - cellName: Customized message's MessagCell class name.
     *    - cellDataName: Customized message's MessagCellData class name.
     */
    - (void)registerCustomMessage:(NSString *)businessID
                 messageCellClassName:(NSString *)cellName
             messageCellDataClassName:(NSString *)cellDataName;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the Message List interface.
TUIChatConfig_Minimalist.shared.registerCustomMessage(businessID: "text_link", mess
```

```
// When to call: Before initializing the Message List interface.
[[TUIChatConfig_Minimalist sharedConfig] registerCustomMessage:BussinessID_TextLink
                                          messageCellClassName:@"TUILinkCell"
                                      messageCellDataClassName:@"TUILinkCellData"];
```

本地插入系统提示消息

API 作用：在本地插入一条系统提示消息。

API 原型：

Swift

Objective-C

```
    // TUIChatBaseDataProvider.swift
    /**
     * Insert local tips message.
     * - Parameters:
     *    - content: tips message content
     *    - chatID: if is group chat,chatID is group id,if is single chat,chatID is user
     *    - isGroup: whether is group chat
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 */
class func insertLocalTipsMessage(_ content: String, chatID: String, isGroup: Bool)
```

```
// TUIChatBaseDataProvider.h
/**
 * Insert local tips message.
 * - Parameters:
 *   - content: tips message content
 *   - chatID: if is group chat,chatID is group id,if is single chat,chatID is user
 *   - isGroup: whether is group chat
 */
+ (void)insertLocalTipsMessage:(NSString *)content chatID:(NSString *)chatID isGrou
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing the Message List interface.
TUIChatBaseDataProvider.insertLocalTipsMessage("Test Tips", chatID: "100480", isGro
```

```
// When to call: Before initializing the Message List interface.
[TUIChatBaseDataProvider insertLocalTipsMessage:@"Test Tips" chatID:@"100480" isGro
```

点击、长按消息列表里的用户头像

API 作用：用户点击、长按了消息列表里的用户头像的事件回调。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
public protocol TUIChatConfigDelegate_Minimalist: NSObjectProtocol {
    /**
     * Tells the delegate a user's avatar in the chat list is clicked.
     * Returning YES indicates this event has been intercepted, and Chat will not p
     * Returning NO indicates this event is not intercepted, and Chat will continue
     */
    func onUserAvatarClicked(view: UIView, messageCellData: TUIMessageCellData) ->
    /**
     * Tells the delegate a user's avatar in the chat list is long pressed.
     * Returning YES indicates that this event has been intercepted, and Chat will
     * Returning NO indicates that this event is not intercepted, and Chat will con
     */
    func onUserAvatarLongPressed(view: UIView, messageCellData: TUIMessageCellData)
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
```

```
// TUIChatConfig_Minimalist.h
@protocol TUIChatConfigDelegate_Minimalist <NSObject>
/**
 * Tells the delegate a user's avatar in the chat list is clicked.
 * Returning YES indicates this event has been intercepted, and Chat will not proce
 * Returning NO indicates this event is not intercepted, and Chat will continue to
 */
- (BOOL)onUserAvatarClicked:(UIView *)view messageCellData:(TUIMessageCellData *)ce
/**
 * Tells the delegate a user's avatar in the chat list is long pressed.
 * Returning YES indicates that this event has been intercepted, and Chat will not
 * Returning NO indicates that this event is not intercepted, and Chat will continu
 */
- (BOOL)onUserAvatarLongPressed:(UIView *)view messageCellData:(TUIMessageCellData
@end
```

示例代码：

Swift

Objective-C

```
TUIChatConfig_Minimalist.shared.delegate = self

func onUserAvatarClicked(view: UIView, messageCellData: TUIMessageCellData) -> Bool
    // Customize your own action when user avatar is clicked.
    print("onUserAvatarClicked")
    return true
}

func onUserAvatarLongPressed(view: UIView, messageCellData: TUIMessageCellData) ->
    // Customize your own action when user avatar is long pressed.
    print("onUserAvatarLongPressed")
    return true
}
```

```
[TUIChatConfig_Minimalist sharedConfig].delegate = self;

// TUIChatConfigDelegate_Minimalist
- (BOOL)onUserAvatarClicked:(UIView *)view messageCellData:(TUIMessageCellData *)ce
    // Customize your own action when user avatar is clicked.
    NSLog(@"onUserAvatarClicked, cellData: %@", celldata);
    return YES;
}
- (BOOL)onUserAvatarLongPressed:(UIView *)view messageCellData:(TUIMessageCellData
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    // Customize your own action when user avatar is long pressed.
    NSLog(@"onUserAvatarLongPressed, cellData: %@", celldata);
    return YES;
}
```

点击、长按消息列表里的消息

API 作用：用户点击、长按了消息列表里的消息的事件回调。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
public protocol TUIChatConfigDelegate_Minimalist: NSObjectProtocol {
    /**
     * Tells the delegate a message in the chat list is clicked.
     * Returning YES indicates that this event has been intercepted, and Chat will
     * Returning NO indicates that this event is not intercepted, and Chat will con
     */
    func onMessageClicked(view: UIView, messageCellData: TUIMessageCellData) -> Boo
    /**
     * Tells the delegate a message in the chat list is long pressed.
     * Returning YES indicates that this event has been intercepted, and Chat will
     * Returning NO indicates that this event is not intercepted, and Chat will con
     */
    func onMessageLongPressed(view: UIView, messageCellData: TUIMessageCellData) ->
}
```

```
// TUIChatConfig_Minimalist.h
@protocol TUIChatConfigDelegate_Minimalist <NSObject>
/**
 * Tells the delegate a message in the chat list is clicked.
 * Returning YES indicates that this event has been intercepted, and Chat will not
 * Returning NO indicates that this event is not intercepted, and Chat will continu
 */
- (BOOL)onMessageClicked:(UIView *)view messageCellData:(TUIMessageCellData *)celld
/**
 * Tells the delegate a message in the chat list is long pressed.
 * Returning YES indicates that this event has been intercepted, and Chat will not
 * Returning NO indicates that this event is not intercepted, and Chat will continu
 */
- (BOOL)onMessageLongPressed:(UIView *)view messageCellData:(TUIMessageCellData *)c
@end
```

示例代码：

©2013-2025 Tencent Cloud International Pte. Ltd.

Swift

Objective-C

```
TUIChatConfig_Minimalist.sharedConfig.delegate = self

func onMessageClicked(view: UIView, messageCellData: TUIMessageCellData) -> Bool {
    // Customize your own action when message is clicked.
    print("onMessageClicked")
    return true
}

func onMessageLongPressed(view: UIView, messageCellData: TUIMessageCellData) -> Boo
    // Customize your own action when message is long pressed.
    print("onMessageLongPressed")
    return true
}
```

```
[TUIChatConfig_Minimalist sharedConfig].delegate = self;

// TUIChatConfigDelegate_Minimalist
- (BOOL)onMessageClicked:(UIView *)view messageCellData:(TUIMessageCellData *)celld
    // Customize your own action when message is clicked.
    NSLog(@"onMessageClicked, cellData: %@", celldata);
    return YES;
}
- (BOOL)onMessageLongPressed:(UIView *)view messageCellData:(TUIMessageCellData *)c
    // Customize your own action when message is long pressed.
    NSLog(@"onMessageLongPressed, cellData: %@", celldata);
    return YES;
}
```

消息样式相关

文本消息的颜色、字体

API 作用：设置发送、接收的文本消息的文字颜色和字体。针对所有的文本消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * The color of send text message.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
     */
    public var sendTextMessageColor: UIColor? {
        get {
            return TUITextMessageCell_Minimalist.outgoingTextColor
        }
        set {
            TUITextMessageCell_Minimalist.outgoingTextColor = newValue
        }
    }

    /**
     * The font of send text message.
     */
    public var sendTextMessageFont: UIFont? {
        get {
            return TUITextMessageCell_Minimalist.outgoingTextFont ?? UIFont()
        }
        set {
            TUITextMessageCell_Minimalist.outgoingTextFont = newValue
        }
    }

    /**
     * The color of receive text message.
     */
    public var receiveTextMessageColor: UIColor? {
        get {
            return TUITextMessageCell_Minimalist.incommingTextColor ?? UIColor()
        }
        set {
            TUITextMessageCell_Minimalist.incommingTextColor = newValue
        }
    }

    /**
     * The font of receive text message.
     */
    public var receiveTextMessageFont: UIFont? {
        get {
            return TUITextMessageCell_Minimalist.incommingTextFont ?? UIFont()
        }
        set {
            TUITextMessageCell_Minimalist.incommingTextFont = newValue
        }
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// TUIChatConfig_Minimalist.h
/**
 * The color of send text message.
 */
@property(nonatomic, assign) UIColor *sendTextMessageColor;
/**
 * The font of send text message.
 */
@property(nonatomic, assign) UIFont *sendTextMessageFont;
/*
 * The color of receive text message.
 */
@property(nonatomic, assign) UIColor *receiveTextMessageColor;
/**
 * The font of receive text message.
 */
@property(nonatomic, assign) UIFont *receiveTextMessageFont;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.sendTextMessageColor = UIColor.tui_color(withHex: "
TUIChatConfig_Minimalist.shared.sendTextMessageFont = UIFont.systemFont(ofSize: 20)
TUIChatConfig_Minimalist.shared.receiveTextMessageColor = UIColor.tui_color(withHex
TUIChatConfig_Minimalist.shared.receiveTextMessageFont = UIFont.systemFont(ofSize:
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].sendTextMessageColor = [UIColor tui_colorWi
[TUIChatConfig_Minimalist sharedConfig].sendTextMessageFont = [UIFont systemFontOfS
[TUIChatConfig_Minimalist sharedConfig].receiveTextMessageColor = [UIColor tui_colo
[TUIChatConfig_Minimalist sharedConfig].receiveTextMessageFont = [UIFont systemFont
```

设置效果：

设置文本消息文字颜色	设置文本消息字体	默认

系统通知消息字体、颜色和背景色

API 作用：设置系统通知消息文字的字体、颜色和背景色，针对所有系统通知消息生效。

©2013-2025 Tencent Cloud International Pte. Ltd.

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * The text color of system message.
 */
public var systemMessageTextColor: UIColor? {
    get {
        return TUISystemMessageCellData.textColor
    }
    set {
        TUISystemMessageCellData.textColor = newValue
    }
}

/**
 * The font of system message.
 */
public var systemMessageTextFont: UIFont? {
    get {
        return TUISystemMessageCellData.textFont
    }
    set {
        TUISystemMessageCellData.textFont = newValue
    }
}

/**
 * The background color of system message.
 */
public var systemMessageBackgroundColor: UIColor? {
    get {
        return TUISystemMessageCellData.textBackgroundColor
    }
    set {
        TUISystemMessageCellData.textBackgroundColor = newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * The text color of system message.
 */
@property (nonatomic, strong) UIColor *systemMessageTextColor;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
/**
 * The font of system message.
 */
@property (nonatomic, strong) UIFont *systemMessageTextFont;
/**
 * The background color of system message.
 */
@property (nonatomic, strong) UIColor *systemMessageBackgroundColor;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.systemMessageTextColor = UIColor.tui_color(withHex:
TUIChatConfig_Minimalist.shared.systemMessageTextFont = UIFont.systemFont(ofSize: 2
TUIChatConfig_Minimalist.shared.systemMessageBackgroundColor = UIColor.tui_color(wi
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].systemMessageTextColor = [UIColor tui_color
[TUIChatConfig_Minimalist sharedConfig].systemMessageTextFont = [UIFont systemFontO
[TUIChatConfig_Minimalist sharedConfig].systemMessageBackgroundColor = [UIColor tui
```

设置效果：

设置系统通知消息文字的字体、颜色和背景色	默认

消息布局相关

消息 layout

API 作用：设置各种类型消息 layout，针对指定的消息生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Text message cell layout of my sent message.
 */
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
public func sendTextMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .text, isSender: true)
}

/**
 * Text message cell layout of my received message.
 */
public func receiveTextMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .text, isSender: false)
}

/**
 * Image message cell layout of my sent message.
 */
public func sendImageMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .image, isSender: true)
}

/**
 * Image message cell layout of my received message.
 */
public func receiveImageMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .image, isSender: false)
}

/**
 * Voice message cell layout of my sent message.
 */
public func sendVoiceMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .voice, isSender: true)
}

/**
 * Voice message cell layout of my received message.
 */
public func receiveVoiceMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .voice, isSender: false)
}

/**
 * Video message cell layout of my sent message.
 */
public func sendVideoMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .video, isSender: true)
}

/**
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 * Video message cell layout of my received message.
 */
public func receiveVideoMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .video, isSender: false)
}

/**
 * Other message cell layout of my sent message.
 */
public func sendMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .other, isSender: true)
}

/**
 * Other message cell layout of my received message.
 */
public func receiveMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .other, isSender: false)
}

/**
 * System message cell layout.
 */
public func systemMessageLayout() -> TUIMessageCellLayout {
    return getMessageLayout(ofType: .system, isSender: false)
}
```

```
// TUIMessageCellLayout.h
@interface TUIMessageCellLayout : NSObject
/**
 * The insets of message
 */
@property(nonatomic, assign) UIEdgeInsets messageInsets;
/**
 * The insets of bubble content.
 */
@property(nonatomic, assign) UIEdgeInsets bubbleInsets;
/**
 * The insets of avatar
 */
@property(nonatomic, assign) UIEdgeInsets avatarInsets;
/**
 * The size of avatar
 */
@property(nonatomic, assign) CGSize avatarSize;
@end
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// TUIChatConfig_Minimalist.h
/**
 * Text message cell layout of my sent message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *sendTextMessageLayout;
/**
 * Text message cell layout of my received message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *receiveTextMessageLayo
/**
 * Image message cell layout of my sent message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *sendImageMessageLayout
/**
 * Image message cell layout of my received message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *receiveImageMessageLay
/**
 * Voice message cell layout of my sent message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *sendVoiceMessageLayout
/**
 * Voice message cell layout of my received message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *receiveVoiceMessageLay
/**
 * Video message cell layout of my sent message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *sendVideoMessageLayout
/**
 * Video message cell layout of my received message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *receiveVideoMessageLay
/**
 * Other message cell layout of my sent message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *sendMessageLayout;
/**
 * Other message cell layout of my received message.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *receiveMessageLayout;
/**
 * System message cell layout.
 */
@property(nonatomic, assign, readonly) TUIMessageCellLayout *systemMessageLayout;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.receiveTextMessageLayout().bubbleInsets = UIEdgeIns
TUIChatConfig_Minimalist.shared.sendTextMessageLayout().avatarInsets = UIEdgeInsets
TUIChatConfig_Minimalist.shared.sendTextMessageLayout().bubbleInsets = UIEdgeInsets
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].receiveTextMessageLayout.bubbleInsets = UIE
[TUIChatConfig_Minimalist sharedConfig].sendTextMessageLayout.avatarInsets = UIEdge
[TUIChatConfig_Minimalist sharedConfig].sendTextMessageLayout.bubbleInsets = UIEdge
```

设置效果：

设置头像尺寸	设置头像边距	设置气泡内边距

消息气泡相关

开启消息气泡展示

API 作用：开启消息气泡展示，针对所有聊天界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Enable the message display in the bubble style.
 * The default value is YES.
 */
public var enableMessageBubbleStyle: Bool {
    get {
        return TIMConfig.shared.enableMessageBubble
    }
    set {
        TIMConfig.shared.enableMessageBubble = newValue
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// TUIChatConfig_Minimalist.h
/**
 * Enable the message display in the bubble style.
 * The default value is YES.
 */
@property(nonatomic, assign) BOOL enableMessageBubbleStyle;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.enableMessageBubbleStyle = false
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].enableMessageBubbleStyle = NO;
```

设置效果：

不显示消息气泡	默认

气泡背景图设置

API 作用：设置气泡背景图，针对所有聊天界面生效。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 * Set the background image of the last sent message bubble in consecutive messages
 */
public var sendLastBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.outgoingBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.outgoingBubble = newValue ?? UIImage()
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
/**
 * Set the background image of the non-last sent message bubble in consecutive mess
 */
public var sendBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.outgoingSameBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.outgoingSameBubble = newValue ?? UIImage()
    }
}

/**
 * Set the background image of the sent message bubble in highlight status.
 */
public var sendHighlightBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.outgoingHighlightedBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.outgoingHighlightedBubble = newValue ?? UII
    }
}

/**
 * Set the light background image when the sent message bubble needs to flicker.
 */
public var sendAnimateLightBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.outgoingAnimatedHighlightedAlpha20
    }
    set {
        TUIBubbleMessageCell_Minimalist.outgoingAnimatedHighlightedAlpha20 = newVal
    }
}

/**
 * Set the dark background image when the sent message bubble needs to flicker.
 */
public var sendAnimateDarkBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.outgoingAnimatedHighlightedAlpha50
    }
    set {
        TUIBubbleMessageCell_Minimalist.outgoingAnimatedHighlightedAlpha50 = newVal
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}

/**
 * Set the background image of the last received message bubble in consecutive mess
 */
public var receiveLastBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.incommingBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.incommingBubble = newValue ?? UIImage()
    }
}

/**
 * Set the background image of the non-last received message bubble in consecutive
 */
public var receiveBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.incommingSameBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.incommingSameBubble = newValue ?? UIImage()
    }
}

/**
 * Set the background image of the received message bubble in highlight status.
 */
public var receiveHighlightBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.incommingHighlightedBubble
    }
    set {
        TUIBubbleMessageCell_Minimalist.incommingHighlightedBubble = newValue ?? UI
    }
}

/**
 * Set the light background image when the received message bubble needs to flicker
 */
public var receiveAnimateLightBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.incommingAnimatedHighlightedAlpha20
    }
    set {
        TUIBubbleMessageCell_Minimalist.incommingAnimatedHighlightedAlpha20 = newVa
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }
}

/**
 * Set the dark background image when the received message bubble needs to flicker.
 */
public var receiveAnimateDarkBubbleBackgroundImage: UIImage? {
    get {
        return TUIBubbleMessageCell_Minimalist.incommingAnimatedHighlightedAlpha50
    }
    set {
        TUIBubbleMessageCell_Minimalist.incommingAnimatedHighlightedAlpha50 = newVa
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Set the background image of the last sent message bubble in consecutive messages
 */
@property (nonatomic, strong) UIImage *sendLastBubbleBackgroundImage;
/**
 * Set the background image of the non-last sent message bubble in consecutive mess
 */
@property (nonatomic, strong) UIImage *sendBubbleBackgroundImage;
/**
 * Set the background image of the sent message bubble in highlight status.
 */
@property (nonatomic, strong) UIImage *sendHighlightBubbleBackgroundImage;
/**
 * Set the light background image when the sent message bubble needs to flicker.
 */
@property (nonatomic, strong) UIImage *sendAnimateLightBubbleBackgroundImage;
/**
 * Set the dark background image when the sent message bubble needs to flicker.
 */
@property (nonatomic, strong) UIImage *sendAnimateDarkBubbleBackgroundImage;
/**
 * Set the background image of the last received message bubble in consecutive mess
 */
@property (nonatomic, strong) UIImage *receiveLastBubbleBackgroundImage;
/**
 * Set the background image of the non-last received message bubble in consecutive
 */
@property (nonatomic, strong) UIImage *receiveBubbleBackgroundImage;
/**
 * Set the background image of the received message bubble in highlight status.
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 */
@property (nonatomic, strong) UIImage *receiveHighlightBubbleBackgroundImage;
/**
 * Set the light background image when the received message bubble needs to flicker
 */
@property (nonatomic, strong) UIImage *receiveAnimateLightBubbleBackgroundImage;
/**
 * Set the dark background image when the received message bubble needs to flicker.
 */
@property (nonatomic, strong) UIImage *receiveAnimateDarkBubbleBackgroundImage;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.sendLastBubbleBackgroundImage = [UIImage imageNamed
TUIChatConfig_Minimalist.shared.sendBubbleBackgroundImage = [UIImage imageNamed:@"S
TUIChatConfig_Minimalist.shared.receiveLastBubbleBackgroundImage = [UIImage imageNa
TUIChatConfig_Minimalist.shared.receiveBubbleBackgroundImage = [UIImage imageNamed:
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].sendLastBubbleBackgroundImage = [UIImage im
[TUIChatConfig_Minimalist sharedConfig].sendBubbleBackgroundImage = [UIImage imageN
[TUIChatConfig_Minimalist sharedConfig].receiveLastBubbleBackgroundImage = [UIImage
[TUIChatConfig_Minimalist sharedConfig].receiveBubbleBackgroundImage = [UIImage ima
```

设置效果：

设置气泡背景图	默认

输入栏相关

展示聊天界面输入框

API 作用：展示聊天界面输入框，针对所有聊天界面生效。

API 原型：

Swift

Objective-C

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// TUIChatConfig_Minimalist.swift
/**
 *  Show the input bar in the message list interface.
 *  The default value is YES.
 */
public var showInputBar: Bool {
    get {
        return !TUIChatConfig.shared.enableMainPageInputBar
    }
    set {
        TUIChatConfig.shared.enableMainPageInputBar = !newValue
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 *  Show the input bar in the message list interface.
 *  The default value is YES.
 */
@property(nonatomic, assign) BOOL showInputBar;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.showInputBar = false
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].showInputBar = NO;
```

设置效果：

隐藏输入框	默认

隐藏更多菜单中选项（全局）

API 作用：隐藏更多菜单中的按钮，针对所有聊天界面生效。

API 原型：

Swift

©2013-2025 Tencent Cloud International Pte. Ltd.

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
 *  Hide items in more menu.
 */
public class func hideItemsInMoreMenu(_ items: TUIChatInputBarMoreMenuItem) {
    let value = items.rawValue
    TUIChatConfig.shared.enableWelcomeCustomMessage = (value & TUIChatInputBarMoreM
    TUIChatConfig.shared.showRecordVideoButton = (value & TUIChatInputBarMoreMenuIt
    TUIChatConfig.shared.showTakePhotoButton = (value & TUIChatInputBarMoreMenuItem
    TUIChatConfig.shared.showAlbumButton = (value & TUIChatInputBarMoreMenuItem.alb
    TUIChatConfig.shared.showFileButton = (value & TUIChatInputBarMoreMenuItem.file
}
```

```
// TUIChatConfig_Minimalist.h
/**
 *  Hide items in more menu.
 */
+ (void)hideItemsInMoreMenu:(TUIChatInputBarMoreMenuItem_Minimalist)items;
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.hideItemsInMoreMenu([.customMessage, .recordVideo, .file])
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist hideItemsInMoreMenu:TUIChatInputBarMoreMenuItem_Minimalis
```

设置效果：

隐藏部分 item	默认

隐藏更多菜单中选项（局部）

API 作用：隐藏更多菜单中的按钮，针对指定聊天界面生效。

API 原型：

Swift

©2013-2025 Tencent Cloud International Pte. Ltd.

Objective-C

```
// TUIChatConfig.swift
public protocol TUIChatInputBarConfigDataSource: AnyObject {
/**
*  Implement this method to add new items to the more list of the specified model o
*/
func shouldHideItems(of model: TUIChatConversationModel) -> TUIChatInputBarMoreMenu
}
```

```
// TUIChatConfig.h
@protocol TUIChatInputBarConfigDataSource <NSObject>
/**
*  Implement this method to add new items to the more list of the specified model o
*/
- (TUIChatInputBarMoreMenuItem)inputBarShouldHideItemsInMoreMenuOfModel:(TUIChatCon
@end
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.inputBarDataSource = self

func shouldHideItems(of model: TUIChatConversationModel) -> TUIChatInputBarMoreMenu
    if model.groupID == "your target groupID" {
        return [.customMessage, .recordVideo, .file]
    }
    return [.none]
}
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].inputBarDataSource = self;

- (TUIChatInputBarMoreMenuItem)inputBarShouldHideItemsInMoreMenuOfModel:(TUIChatCon
    if ([model.groupID isEqualToString:@"your target groupID"]) {
        return TUIChatInputBarMoreMenuItem_CustomMessage|TUIChatInputBarMoreMenuIte
    }
    return TUIChatInputBarMoreMenuItem_None;
}
```

更多菜单添加选项（局部）

API 作用：向更多菜单添加选项，针对指定聊天界面生效。

©2013-2025 Tencent Cloud International Pte. Ltd.

API 原型：

Swift

Objective-C

```
// TUIChatConfig.swift
public protocol TUIChatInputBarConfigDataSource: AnyObject {
    /**
     *  Implement this method to add new items to the more menu of the specified mo
     */
    func shouldAddNewItemsToMoreList(of model: TUIChatConversationModel) -> [TUICus
}
```

```
// TUIChatConfig.h
@protocol TUIChatInputBarConfigDataSource <NSObject>
/**
 *  Implement this method to add new items to the more list of the specified model
 */
- (NSArray<TUICustomActionSheetItem *> *)inputBarShouldAddNewItemsToMoreListOfModel
@end
```

示例代码：

Swift

Objective-C

```
// When to call: After initializing the message list interface and before entering
TUIChatConfig_Minimalist.shared.inputBarDataSource = self

func shouldAddNewItemsToMoreList(of model: TUIChatConversationModel) -> [TUICustomA
    let item1 = TUICustomActionSheetItem(title: "item1", leftMark: UIImage(named: "
        print("item1 is clicked")
    }

    let item2 = TUICustomActionSheetItem(title: "item2", leftMark: UIImage(named: "
        print("item1 is clicked")
    }

    return [item1, item2]
}
```

```
// When to call: After initializing the message list interface and before entering
[TUIChatConfig_Minimalist sharedConfig].inputBarDataSource = self;

- (NSArray<TUICustomActionSheetItem *> *)inputBarShouldAddNewItemsToMoreListOfModel
    TUICustomActionSheetItem *item1 = [[TUICustomActionSheetItem alloc] initWithTit
        NSLog(@"item1 is clicked");
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }];

    TUICustomActionSheetItem *item2 = [[TUICustomActionSheetItem alloc] initWithTit
        NSLog(@"item2 is clicked");
    }];

    return @[item1, item2];
}
```

设置效果：

添加 item	默认

添加表情组

API 作用：向表情菜单中添加表情组，针对所有聊天界面生效。使用场景请参考文档《添加自定义表情》。

API 原型：

Swift

Objective-C

```
// TUIChatConfig_Minimalist.swift
/**
* Add sticker group.
*/
public func addStickerGroup(_ group: TUIFaceGroup) {
    if let service = TIMCommonMediator.shared.getObject(for: TUIEmojiMeditorProtoco
        service.appendFaceGroup(group)
    } else {
        print("Failed to get TUIEmojiMeditorProtocol service")
    }
}
```

```
// TUIChatConfig_Minimalist.h
/**
 * Add sticker group.
 */
- (void)addStickerGroup:(TUIFaceGroup *)group;
```

示例代码：

Swift

Objective-C

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// When to call: After initializing the message list interface and before entering
let group4350 = TUIFaceGroup()
group4350.groupIndex = 1
group4350.groupPath = bundlePath + "/4350/"
group4350.faces = faces4350
group4350.rowCount = 2
group4350.itemCountPerRow = 5
group4350.menuPath = bundlePath + "/4350/menu"
TUIChatConfig_Minimalist.shared.addStickerGroup(group4350)
```

```
// When to call: After initializing the message list interface and before entering
TUIFaceGroup *group4350 = [[TUIFaceGroup alloc] init];
group4350.groupIndex = 1;
group4350.groupPath = [bundlePath stringByAppendingPathComponent:@"4350/"];
group4350.faces = faces4350;
group4350.rowCount = 2;
group4350.itemCountPerRow = 5;
group4350.menuPath = [bundlePath stringByAppendingPathComponent:@"4350/menu"];
[[TUIChatConfig_Minimalist sharedConfig] addStickerGroup:group4350];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

会话列表

最近更新时间：2025-05-23 12:02:32

下文将向您展示如何设置会话列表界面自定义选项及其效果。

设置会话列表、cell 背景色

API 作用：设置会话列表、cell、置顶 cell 的背景色。

API 原型：

Swift

Objective-C

```
// TUIConversationConfig.swift
/**
 *  Background color of conversation list.
 */
public var listBackgroundColor: UIColor?
/**
 *  Background color of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public var cellBackgroundColor: UIColor?
/**
 *  Background color of pinned cell in conversation list.
 *  This configuration takes effect in all pinned cells.
 */
public var pinnedCellBackgroundColor: UIColor?
```

```
// TUIConversationConfig.h
/**
 *  Background color of conversation list.
 */
@property (nonatomic, strong) UIColor *listBackgroundColor;
/**
 *  Background color of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
@property (nonatomic, strong) UIColor *cellBackgroundColor;
/**
 *  Background color of pinned cell in conversation list.
 *  This configuration takes effect in all pinned cells.
 */
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
@property (nonatomic, strong) UIColor *pinnedCellBackgroundColor;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing conversation list.
TUIConversationConfig.shared.listBackgroundColor = UIColor.tui_color(withHex: "#FFF
TUIConversationConfig.shared.cellBackgroundColor = UIColor.tui_color(withHex: "#F0F
TUIConversationConfig.shared.pinnedCellBackgroundColor = UIColor.tui_color(withHex:
```

```
// When to call: Before initializing conversation list.
[TUIConversationConfig sharedConfig].listBackgroundColor = [UIColor tui_colorWithHe
[TUIConversationConfig sharedConfig].cellBackgroundColor = [UIColor tui_colorWithHe
[TUIConversationConfig sharedConfig].pinnedCellBackgroundColor = [UIColor tui_color
```

设置效果：

设置背景色	默认

设置会话列表 cell 字体

API 作用：设置会话列表 cell 上的标题、副标题、时间文字的字体。针对所有 cell 生效。

API 原型：

Swift

Objective-C

```
// TUIConversationConfig.swift
/**
 *  Font of title label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public var cellTitleLabelFont: UIFont?
/**
 *  Font of subtitle label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public var cellSubtitleLabelFont: UIFont?
/**
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
 *  Font of time label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
public var cellTimeLabelFont: UIFont?
```

```
// TUIConversationConfig.h
/**
 *  Font of title label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
@property (nonatomic, strong) UIFont *cellTitleLabelFont;
/**
 *  Font of subtitle label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
@property (nonatomic, strong) UIFont *cellSubtitleLabelFont;
/**
 *  Font of time label of cell in conversation list.
 *  This configuration takes effect in all cells.
 */
@property (nonatomic, strong) UIFont *cellTimeLabelFont;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing conversation list.
TUIConversationConfig.shared.cellTitleLabelFont = UIFont.systemFont(ofSize: 18)
TUIConversationConfig.shared.cellSubtitleLabelFont = UIFont.systemFont(ofSize: 14)
TUIConversationConfig.shared.cellTimeLabelFont = UIFont.systemFont(ofSize: 16)
```

```
// When to call: Before initializing conversation list.
[TUIConversationConfig sharedConfig].cellTitleLabelFont = [UIFont systemFontOfSize:
[TUIConversationConfig sharedConfig].cellSubtitleLabelFont = [UIFont systemFontOfSi
[TUIConversationConfig sharedConfig].cellTimeLabelFont = [UIFont systemFontOfSize:1
```

设置效果：

设置字体	默认

©2013-2025 Tencent Cloud International Pte. Ltd.

展示未读红点

API 作用：展示 cell 上的未读消息红点 icon。针对所有 cell 生效。

API 原型：

Swift

Objective-C

```
// TUIConversationConfig.swift
/**
 *  Display unread count icon in each conversation cell.
 *  The default value is YES.
 */
public var showCellUnreadCount: Bool = true
```

```
// TUIConversationConfig.h
/**
 *  Display unread count icon in each conversation cell.
 *  The default value is YES.
 */
@property(nonatomic, assign) BOOL showCellUnreadCount;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing conversation list.
TUIConversationConfig.shared.showCellUnreadCount = false
```

```
// When to call: Before initializing conversation list.
[TUIConversationConfig sharedConfig].showCellUnreadCount = NO;
```

设置效果：

不展示会话 cell 上的未读红点	默认

展示在线状态

API 作用：展示 cell 里用户头像上的在线状态 icon。针对所有 cell 生效。

©2013-2025 Tencent Cloud International Pte. Ltd.

API 原型：

Swift

Objective-C

```
// TUIConversationConfig.swift
/**
 *  Display user's online status icon in conversation and contact list.
 *  The default value is NO.
 */
public var showUserOnlineStatusIcon: Bool {
    get {
        return TUIConfig.default().displayOnlineStatusIcon
    }
    set {
        TUIConfig.default().displayOnlineStatusIcon = newValue
    }
}
```

```
// TUIConversationConfig.h
/**
 *  Display user's online status icon in conversation and contact list.
 *  The default value is NO.
 */
@property(nonatomic, assign) BOOL showUserOnlineStatusIcon;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing conversation list.
TUIConversationConfig.shared.showUserOnlineStatusIcon = false
```

```
// When to call: Before initializing conversation list.
[TUIConversationConfig sharedConfig].showUserOnlineStatusIcon = YES;
```

设置效果：

展示在线状态	默认

©2013-2025 Tencent Cloud International Pte. Ltd.

会话更多菜单选项自定义

API 作用：隐藏会话更多菜单选项、向会话更多菜单添加选项。针对指定会话生效。

API 原型：

Swift

Objective-C

```
// TUIConversationConfig.swift
public protocol TUIConversationConfigDataSource: AnyObject {
    /**
     * Implement this method to hide items in more menu.
     */
    func conversationShouldHideItemsInMoreMenu(_ data: TUIConversationCellData) ->
    /**
     * Implement this method to add new items.
     */
    func conversationShouldAddNewItemsToMoreMenu(_ data: TUIConversationCellData) -
}
```

```
// TUIConversationConfig.h
@protocol TUIConversationConfigDataSource <NSObject>
/**
 * Implement this method to hide items in more menu.
 */
- (NSInteger)conversationShouldHideItemsInMoreMenu:(TUIConversationCellData *)data;
/**
 * Implement this method to add new items.
 */
- (NSArray *)conversationShouldAddNewItemsToMoreMenu:(TUIConversationCellData *)dat
@end
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing conversation list.
TUIConversationConfig.shared.moreMenuDataSource = self

func conversationShouldHideItemsInMoreMenu(_ data: TUIConversationCellData) -> TUIC
    if let groupID = data.groupID, !groupID.isEmpty {
        return [.Hide, .Pin]
    }
    return [.None]
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
func conversationShouldAddNewItemsToMoreMenu(_ data: TUIConversationCellData) -> [A
    if let groupID = data.groupID, !groupID.isEmpty {
        let action1 = UIAlertAction.init(title: "action1", style: .default) { _ in
            print("action1 is clicked")
        }
        let action2 = UIAlertAction.init(title: "action2", style: .default) { _ in
            print("action2 is clicked")
        }
        return [action1, action2]
    }
    return []
}
```

```
// When to call: Before initializing conversation list.
[TUIConversationConfig sharedConfig].moreMenuDataSource = self;

- (NSInteger)conversationShouldHideItemsInMoreMenu:(TUIConversationCellData *)data
    if (data.groupID != nil) {
        return TUIConversationItemInMoreMenu_Hide | TUIConversationItemInMoreMenu_P
    }
    return 0;
}
- (NSArray *)conversationShouldAddNewItemsToMoreMenu:(TUIConversationCellData *)dat
    if (data.groupID != nil) {
        UIAlertAction *action1 = [UIAlertAction actionWithTitle:@"action1"
                                                          style:UIAlertActionStyleD
                                                        handler:^(UIAlertAction *_N
            NSLog(@"action1 is clicked");
                                                      }];
        UIAlertAction *action2 = [UIAlertAction actionWithTitle:@"action2"
                                                          style:UIAlertActionStyleD
                                                        handler:^(UIAlertAction *_N
            NSLog(@"action2 is clicked");
                                                      }];
        return @[action1, action2];
    }
    return nil;
}
```

设置效果：

隐藏、添加选项	默认

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

群组设置

最近更新时间：2025-05-23 12:02:32

下文将向您展示如何隐藏群设置选项及其效果。

隐藏群设置选项

API 作用：隐藏群组设置选项。针对所有群组生效。

API 原型：

Swift

Objective-C

```
public struct TUIGroupConfigItem: OptionSet {
    public let rawValue: Int

    public init(rawValue: Int) {
        self.rawValue = rawValue
    }

    public static let none = TUIGroupConfigItem([])
    public static let members = TUIGroupConfigItem(rawValue: 1 << 0)
    public static let notice = TUIGroupConfigItem(rawValue: 1 << 1)
    public static let manage = TUIGroupConfigItem(rawValue: 1 << 2)
    public static let alias = TUIGroupConfigItem(rawValue: 1 << 3)
    public static let muteAndPin = TUIGroupConfigItem(rawValue: 1 << 4)
    public static let background = TUIGroupConfigItem(rawValue: 1 << 5)
    public static let clearChatHistory = TUIGroupConfigItem(rawValue: 1 << 6)
    public static let deleteAndLeave = TUIGroupConfigItem(rawValue: 1 << 7)
    public static let transfer = TUIGroupConfigItem(rawValue: 1 << 8)
    public static let dismiss = TUIGroupConfigItem(rawValue: 1 << 9)
    public static let report = TUIGroupConfigItem(rawValue: 1 << 10)
}

/**
 * Hide items in group config interface.
 */
public func hideItemsInGroupConfig(_ items: TUIGroupConfigItem) {
    hideGroupMuteAndPinItems = items.contains(.muteAndPin)
    hideGroupManageItems = items.contains(.manage)
    hideGroupAliasItem = items.contains(.alias)
    hideGroupBackgroundItem = items.contains(.background)
    hideGroupMembersItems = items.contains(.members)
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    hideGroupClearChatHistory = items.contains(.clearChatHistory)
    hideGroupDeleteAndLeave = items.contains(.deleteAndLeave)
    hideGroupTransfer = items.contains(.transfer)
    hideGroupDismiss = items.contains(.dismiss)
    hideGroupReport = items.contains(.report)
}
```

```
// TUIGroupConfig.h
typedef NS_OPTIONS(NSInteger, TUIGroupConfigItem) {
    TUIGroupConfigItem_None = 0,
    TUIGroupConfigItem_Members = 1 << 0,
    TUIGroupConfigItem_Notice = 1 << 1,
    TUIGroupConfigItem_Manage = 1 << 2,
    TUIGroupConfigItem_Alias = 1 << 3,
    TUIGroupConfigItem_MuteAndPin = 1 << 4,
    TUIGroupConfigItem_Background = 1 << 5,
    TUIGroupConfigItem_ClearChatHistory = 1 << 6,
    TUIGroupConfigItem_DeleteAndLeave = 1 << 7,
    TUIGroupConfigItem_Transfer = 1 << 8,
    TUIGroupConfigItem_Dismiss = 1 << 9,
    TUIGroupConfigItem_Report = 1 << 10,
};

/**
 * Hide items in group config interface.
 */
- (void)hideItemsInGroupConfig:(TUIGroupConfigItem)items;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing group setting interface.
TUIGroupConfig.shared.hideItemsInGroupConfig([.muteAndPin, .manage, .members])
```

```
// When to call: Before initializing group setting interface.
[[TUIGroupConfig sharedConfig] hideItemsInGroupConfig:TUIGroupConfigItem_MuteAndPin
```

设置效果：

隐藏部分选项	隐藏全部选项	默认

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

联系人

最近更新时间：2025-05-23 12:02:32

下文将向您展示如何隐藏联系人设置选项及其效果。

隐藏联系人设置选项

API 作用：隐藏联系人设置选项。针对所有联系人生效。

API 原型：

Swift

Objective-C

```
// TUIContactConfig.swift
public struct TUIContactConfigItem: OptionSet {
    public let rawValue: Int

    public init(rawValue: Int) {
        self.rawValue = rawValue
    }

    public static let none = TUIContactConfigItem([])
    public static let alias = TUIContactConfigItem(rawValue: 1 << 0)
    public static let muteAndPin = TUIContactConfigItem(rawValue: 1 << 1)
    public static let background = TUIContactConfigItem(rawValue: 1 << 2)
    public static let block = TUIContactConfigItem(rawValue: 1 << 3)
    public static let clearChatHistory = TUIContactConfigItem(rawValue: 1 << 4)
    public static let delete = TUIContactConfigItem(rawValue: 1 << 5)
    public static let addFriend = TUIContactConfigItem(rawValue: 1 << 6)
}

/**
 * Hide items in contact config interface.
 */
public func hideItemsInContactConfig(_ items: TUIContactConfigItem) {
    hideContactAlias = items.contains(.alias)
    hideContactMuteAndPinItems = items.contains(.muteAndPin)
    hideContactBackgroundItem = items.contains(.background)
    hideContactBlock = items.contains(.block)
    hideContactClearChatHistory = items.contains(.clearChatHistory)
    hideContactDelete = items.contains(.delete)
    hideContactAddFriend = items.contains(.addFriend)
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// TUIContactConfig.h
typedef NS_OPTIONS(NSInteger, TUIContactConfigItem) {
    TUIContactConfigItem_None = 0,
    TUIContactConfigItem_Alias = 1 << 0,
    TUIContactConfigItem_MuteAndPin = 1 << 1,
    TUIContactConfigItem_Background = 1 << 2,
    TUIContactConfigItem_Block = 1 << 3,
    TUIContactConfigItem_ClearChatHistory = 1 << 4,
    TUIContactConfigItem_Delete = 1 << 5,
    TUIContactConfigItem_AddFriend = 1 << 6,
};
/**
 * Hide items in contact config interface.
 */
- (void)hideItemsInContactConfig:(TUIContactConfigItem)items;
```

示例代码：

Swift

Objective-C

```
// When to call: Before initializing contact setting interface.
// Valid for contacts.
TUIContactConfig.shared.hideItemsInContactConfig([.block, .clearChatHistory, .delet
// Valid for strange users who have not been added to the contact.
TUIContactConfig.shared.hideItemsInContactConfig([.addFriend])
```

```
// When to call: Before initializing contact setting interface.
// Valid for contacts.
[[TUIContactConfig sharedConfig] hideItemsInContactConfig:TUIContactConfigItem_Bloc
// Valid for strange users who have not been added to the contact.
[[TUIContactConfig sharedConfig] hideItemsInContactConfig:TUIContactConfigItem_AddF
```

对联系人设置效果：

隐藏部分选项	隐藏全部选项	默认

对尚未添加到联系人的陌生用户设置效果：

隐藏添加好友	默认

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

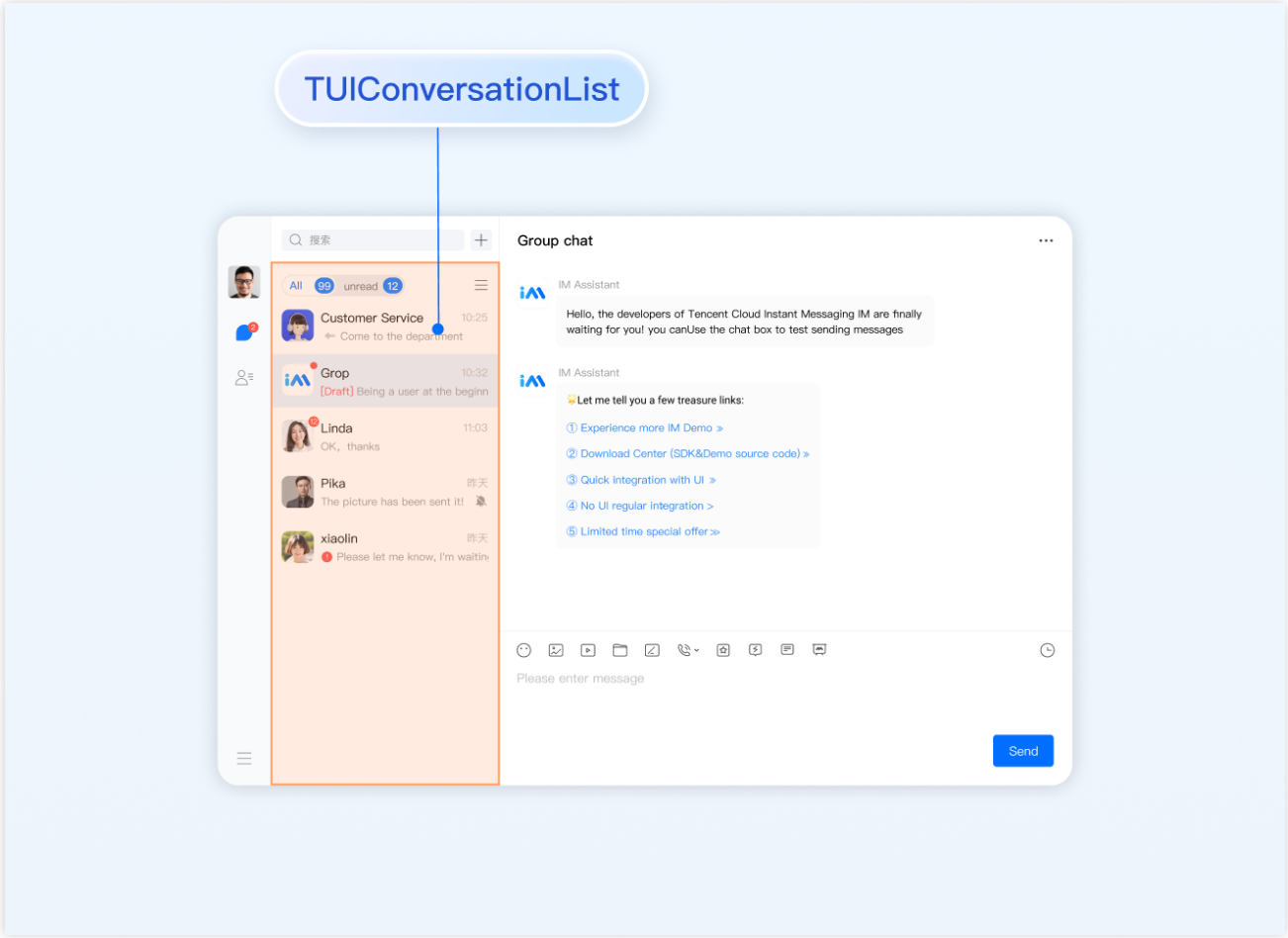
Web(Vue)

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-`

©2013-2025 Tencent Cloud International Pte. Ltd.

`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。

©2013-2025 Tencent Cloud International Pte. Ltd.

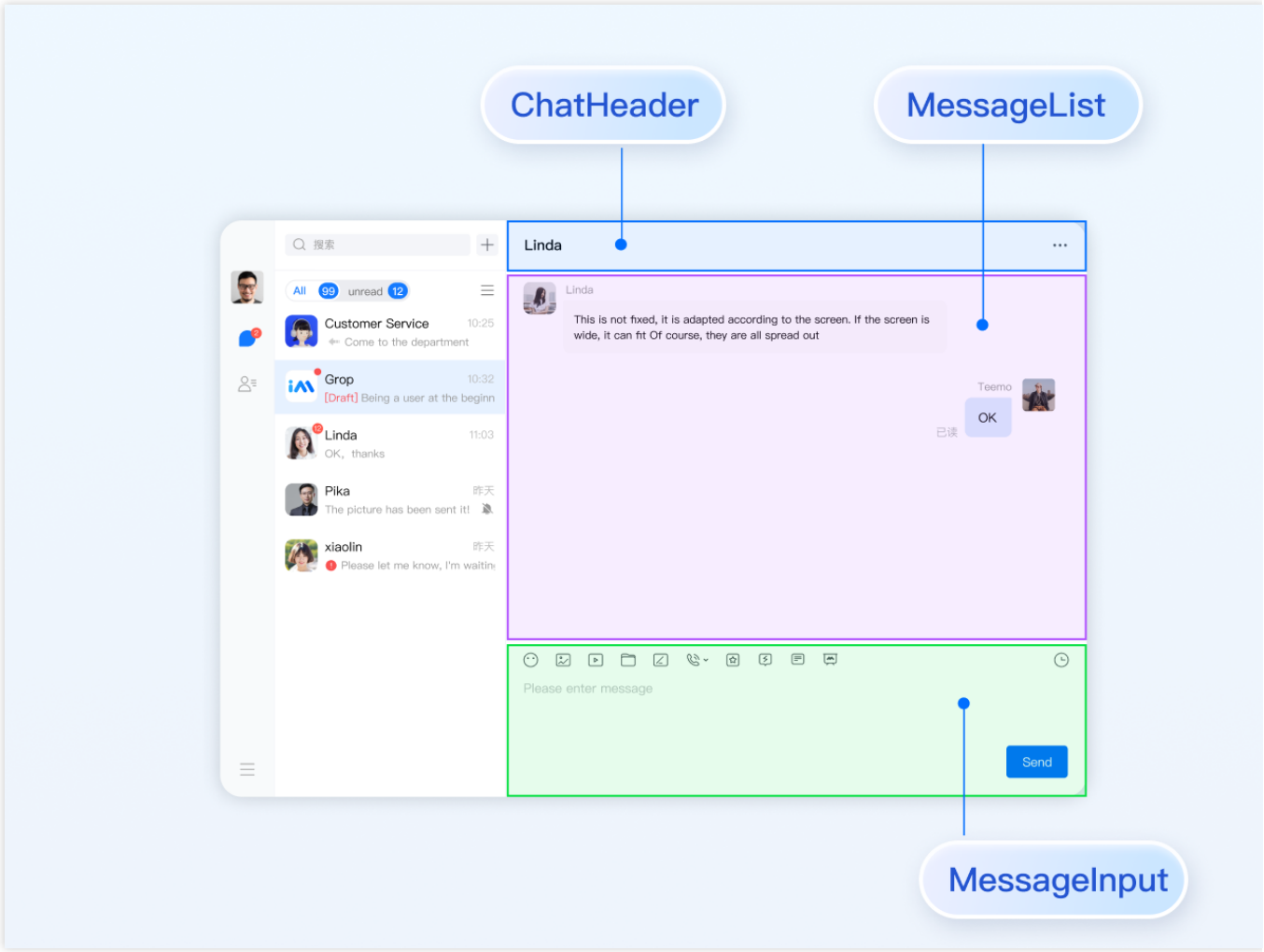
设置会话列表中头像样式示例代码如下：

```
.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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      </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。

©2013-2025 Tencent Cloud International Pte. Ltd.

<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：

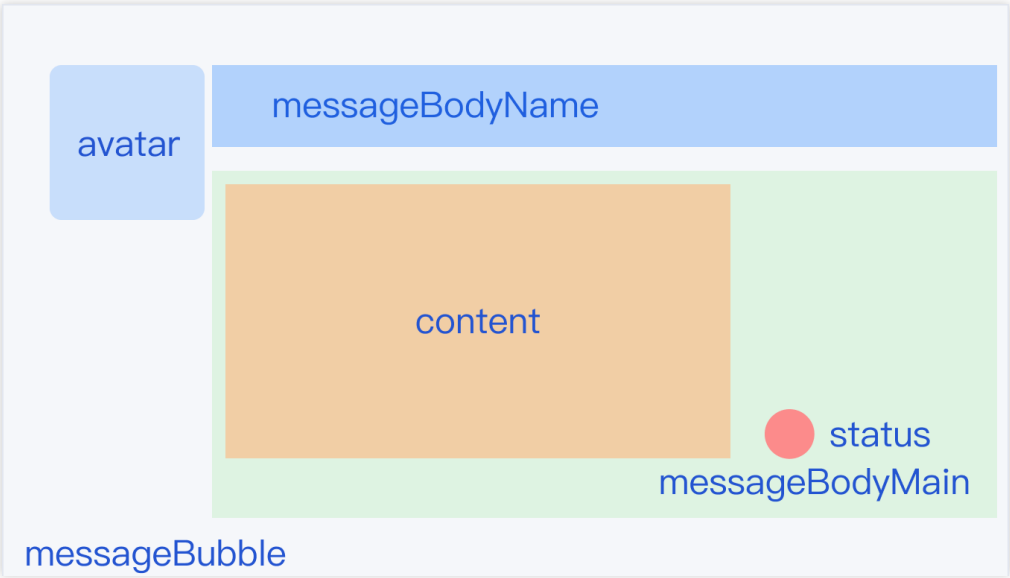
```
<Avatar
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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;
        }
      }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }
  }
}
```

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：

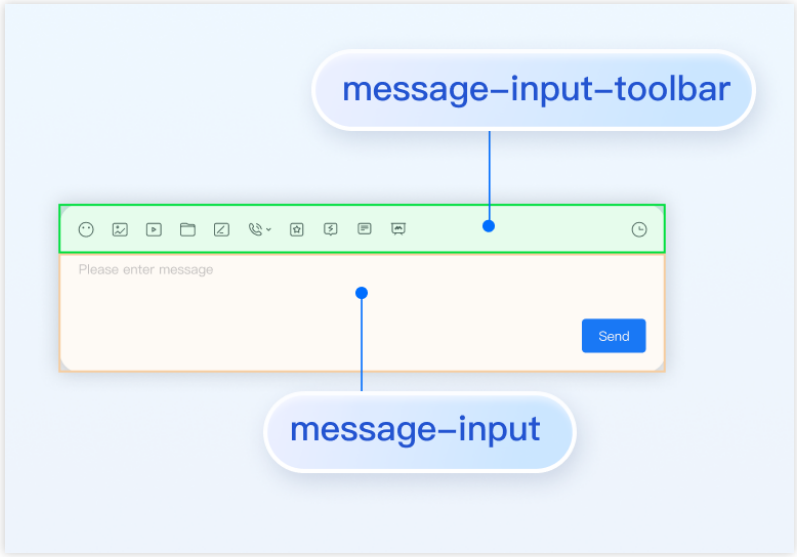
```
.message-tip {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      <!-- 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<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：

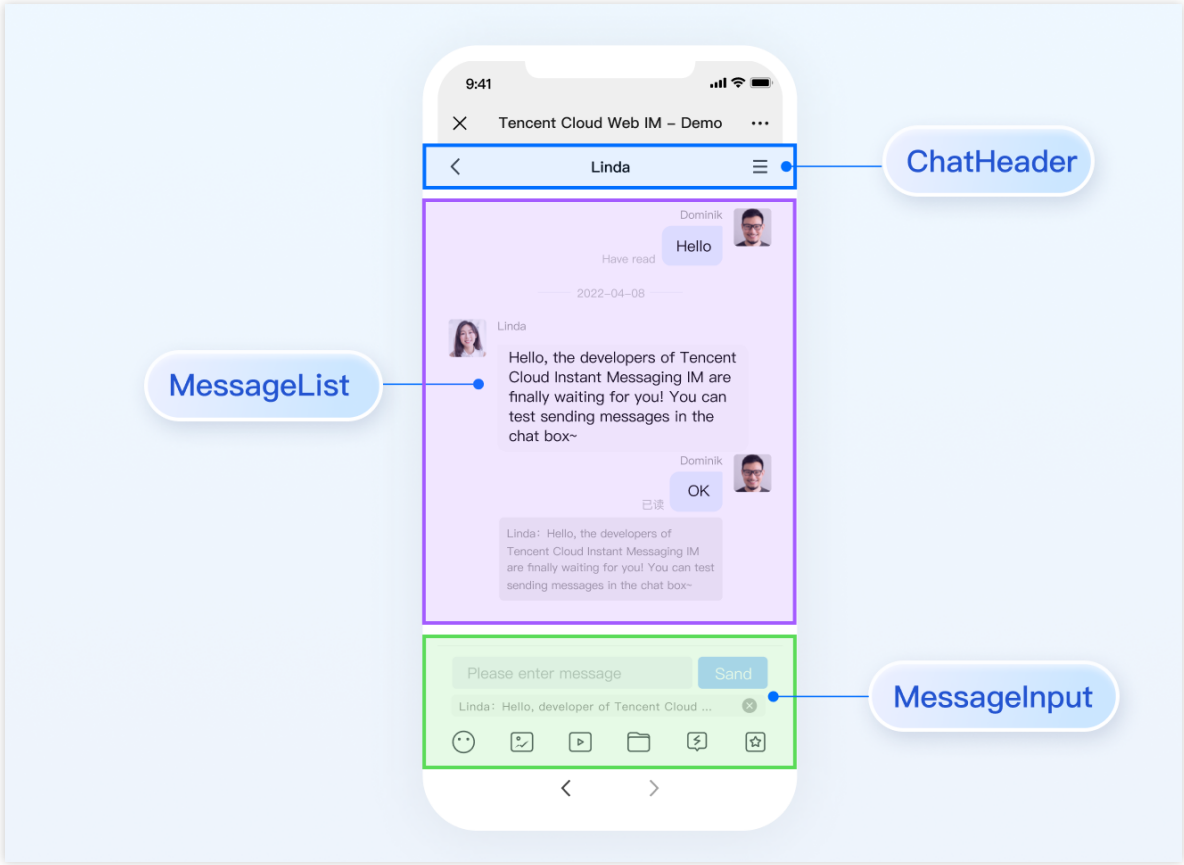
```
.TUI-conversation-content {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  .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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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="handleExtensions(item)">
        <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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
.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"

©2013-2025 Tencent Cloud International Pte. Ltd.

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 || ''"
/>
```

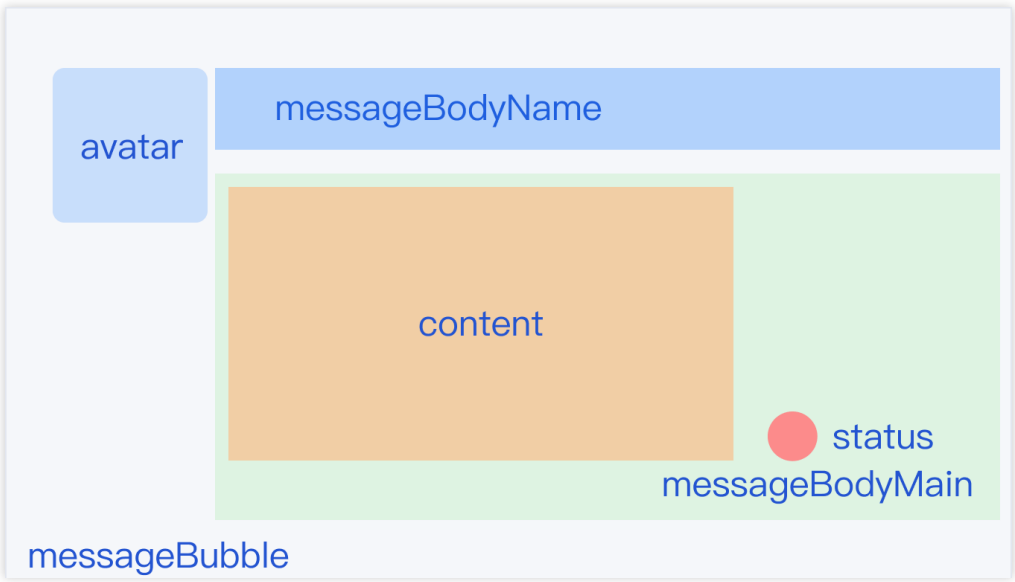
设置头像形状、大小示例代码如下：

```
<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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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 {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    .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

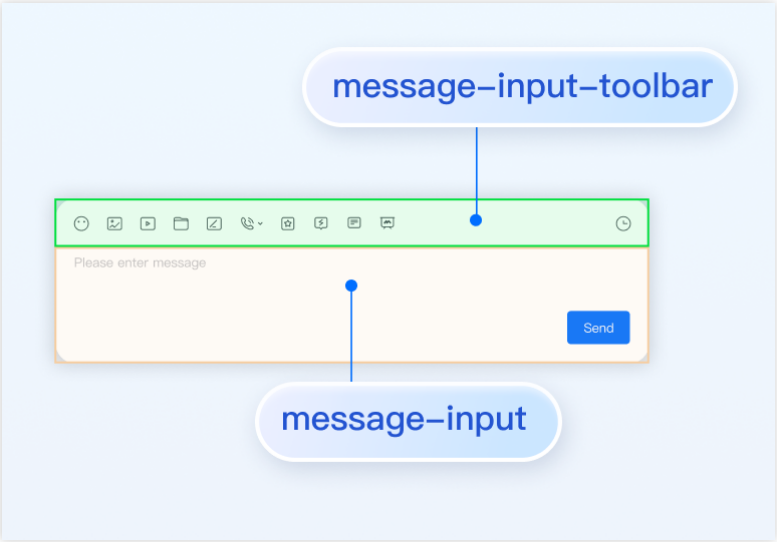
您可以在路径 `src/TUIKit/components/TUIChat/message-list/message-elements/message-tip.vue` 文件中自定义设置提示信息的背景、字体大小以及字体颜色等。
示例代码如下：

```
.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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

交流与反馈

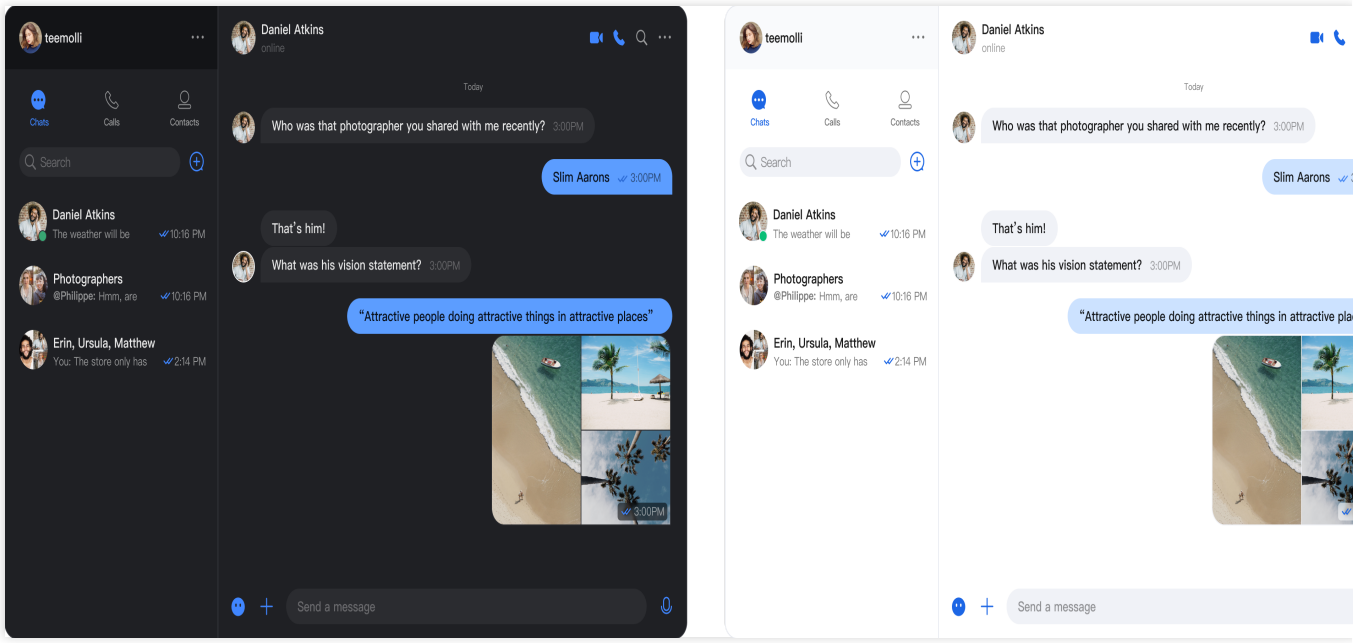
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

©2013-2025 Tencent Cloud International Pte. Ltd.

React

最近更新时间：2024-11-22 11:55:28

React UIKit 目前提供了明亮和黑暗两种预置主题。您可以通过传递 theme 属性到根组件 `<UIKitProvider>` 上来切换主题。另外您也可以完全自定义颜色来适应您自己的品牌颜色。



预置主题

React UIKit 目前提供了明亮和黑暗两种预置主题。您可以在根组件 `<UIKitProvider>` 上设置 theme 属性 `'light'`（默认）或 `'dark'`，以切换当前主题。

说明：

`<UIKitProvider>` 如果不传递 theme 属性则默认使用明亮主题。

使用黑暗主题示例：

```
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import '@tencentcloud/chat-uikit-react/dist/esm/index.css';

export default function App() {
  // language support en-US(default) / zh-CN / ja-JP / ko-KR / zh-TW
  // theme support light(default) / dark
  return (
    <div style={{display: 'flex', height: '100vh'}}>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      <UIKitProvider language='en-US' theme='dark'>

      </UIKitProvider>
    </div>
  );
}
```

使用 useState 管理主题示例：

```
import { useState } from 'react';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';
import '@tencentcloud/chat-uikit-react/dist/esm/index.css';

export default function App() {
  const [theme, setTheme] = useState('light');

  return (
    <div style={{display: 'flex', height: '100vh'}}>
      <UIKitProvider theme={theme}>
        {/* ... */}
      </UIKitProvider>
    </div>
  );
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  // 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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  /// 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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
/// 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

©2013-2025 Tencent Cloud International Pte. Ltd.

实现本地搜索

Android

最近更新时间：2025-03-03 11:36:57

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 Chat 专业版、专业版Plus、企业版功能，购买专业版、专业版Plus、企业版 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 购买专业版、专业版Plus、企业版。

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.
    }
```

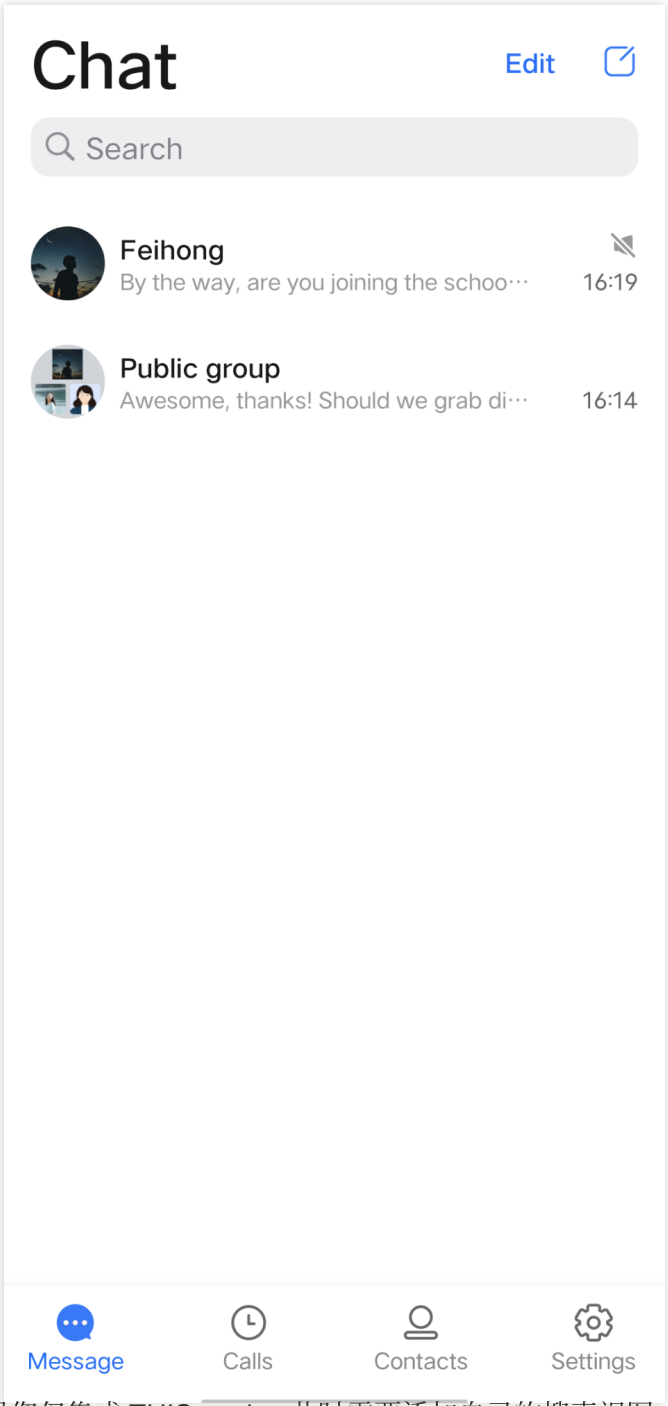
©2013-2025 Tencent Cloud International Pte. Ltd.

```
    @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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 14:43:18

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 Chat 专业版、专业版Plus、企业版功能，购买专业版、专业版Plus、企业版 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 购买专业版、专业版Plus、企业版。

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。

Swift

Objective-C

```
TUILogin.login(SDKAppID, userID: userID, userSig: userSig) {
    // Login succeeded
    print("login succeeded")
} fail: { code, desc in
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    // Login failed
    print("login failed, \\(code): \\(desc)")
}
```

```
[TUILogin login:SDKAPPID userID:userID userSig:userSig succ:^{
   // Login succeeded
} fail:^(int code, NSString *msg) {
   // Login failed
}];
```

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：
Swift
Objective-C

```
// init
let searchBar = TUISearchBar()
// self.containerView is your own view
self.containerView.addSubview(searchBar)
```

```
// init
TUISearchBar *searchBar = [[TUISearchBar alloc] init];
// self.containerView is 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp（Vue）

最近更新时间：2024-07-10 16:26:41

功能体验



©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.





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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<TUISearch searchType="global" />
```





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",
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      "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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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。

©2013-2025 Tencent Cloud International Pte. Ltd.

接入离线推送
Android
厂商配置

最近更新时间：2025-03-26 19:34:39

操作步骤

步骤1：注册应用到厂商推送平台

离线推送需要将您自己的应用注册到各个厂商的推送平台，得到 AppID 和 AppKey 等参数，来实现离线推送功能。
目前支持的手机厂商有：小米、华为、荣耀、OPPO、vivo、魅族、 Google FCM。

步骤2：Chat 控制台配置

登录腾讯云 即时通信 Chat 控制台 ，在**推送管理 > 接入设置**功能栏添加各个厂商推送证书，并将您在步骤一中获取的各厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。
关于**点击后续动作**选项的说明：
打开应用：单击通知栏后拉起应用，默认启动应用的 Launcher 界面；
打开网页：单击通知栏会跳转到配置的网页链接；
打开应用内指定界面：单击通知栏会根据配置自定义跳转界面，详见 自定义点击跳转。
小米
华为
OPPO
vivo
魅族
荣耀
Google FCM

厂商推送平台	Chat 控制台配置

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.

	说明： Client ID 对应 AppID，Client Secret 对应 AppSecret。

厂商推送平台	Chat 控制台配置

厂商推送平台	Chat 控制台配置

回执配置请参见：消息触达统计配置-vivo。

厂商推送平台	Chat 控制台配置

回执配置请参考：消息触达统计配置-魅族。

厂商推送平台	Chat 控制台配置

厂商推送平台	Chat 控制台配置

注意：

©2013-2025 Tencent Cloud International Pte. Ltd.

关于**点击后续动作**支持上报统计功能：

1. 如果选择打开应用和打开网页，购买插件后会默认支持上报统计。
2. 如果选择打开应用内指定界面：

新增证书情况，请直接使用自动填写的默认值即可支持点击上报统计。

如果之前有证书且已配置，继续使用旧证书需要修改为默认值，才可以支持上报统计，或者重新生成新的证书。

关于 FCM 数据消息

FCM 提供两种推送方式是通知消息和数据消息。

通知消息，样式简单不区分设备，成功集成即可进行离线推送。

数据消息，样式定制丰富，特定设备有效，支持触达和点击上报，需要集成后在设备上做好测试开放上线。

控制台默认选择为通知消息，两种模式切换可在 Chat 控制台操作：

注意：

FCM 数据消息能力仅支持 TIMPush 7.8 及以上版本的 pixel 手机，其他厂商设备需自测支持情况。

©2013-2025 Tencent Cloud International Pte. Ltd.

快速接入

最近更新时间：2025-06-23 09:56:05

步骤1：下载并添加配置文件

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

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

步骤2：集成 TIMPush 插件

```
// 版本号 “VERSION” 请前往 更新日志 中获取配置。
// 1. 推送主包必须要集成
implementation 'com.tencent.timpush:timpush:VERSION'
// 2. 集成 FCM 推送包
implementation 'com.tencent.timpush:fcm:VERSION'
// 3. 如果只需要 FCM 通道，以下包不需要集成；如果想优先走 FCM 通道，请调用接口 forceUseFCMPush
implementation 'com.tencent.timpush:huawei:VERSION'
implementation 'com.tencent.timpush:xiaomi:VERSION'
implementation 'com.tencent.timpush:oppo:VERSION'
implementation 'com.tencent.timpush:vivo:VERSION'
implementation 'com.tencent.timpush:honor:VERSION'
implementation 'com.tencent.timpush:meizu:VERSION'
```

说明：

1. TIMPush 需要集成 ChatSDK 在 7.6.5011 版本及以上。
2. 无 UI 或者没有集成其他插件的用户，需要增加集成 TUICore，支持源码和 Maven 集成，方式如下：

```
def projects = this.rootProject.getAllprojects().stream().map { project -> project.
api projects.contains("tuicore") ? project(':tuicore') : "com.tencent.liteav.tuikit
```

vivo 和荣耀配置

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 {
    ...
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    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 begin
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.api_key"
    android:value="您应用分配的证书 APPKEY" />
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.app_id"
    android:value="您应用分配的证书 APPID" />
// vivo end

// honor begin
<meta-data tools:replace="android:value"
    android:name="com.hihonor.push.app_id"
    android:value="您应用分配的证书 APPID" />
// honor end
```

华为、荣耀和 Google FCM 适配

按照厂商方法，集成对应的 plugin 和 json 配置文件。

注意：

以下荣耀的适配仅 7.7.5283 及以上版本需要配置。

1.1 下载配置文件添加到工程根目录：

华为

荣耀

Google FCM

操作路径

©2013-2025 Tencent Cloud International Pte. Ltd.

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

Gradle 7.1 及以上版本

Gradle 7.0 版本

Gradle 7.0 以下版本

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

```
buildscript {
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

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

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

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

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

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

```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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

1.3 在应用级 build.gradle 文件中添加下方配置：

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

以上步骤完成后，就可以收到离线推送通知了。

注意：

1. 如果您想尽可能简单地接入 TIMPush 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 TIMPush 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，您需要在完成登录操作后，手动调用 TIMPushManager 的接口 registerPush。
2. 如果您仅想支持免登录推送的功能，可以使用带 appKey 参数注册推送，步骤 5 的发送消息详细参见 REST API 接口 - 发起全员/标签推送。

步骤3：设置混淆规则

在 proguard-rules.pro 文件，将 TIMPush 相关类加入不混淆名单：

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

步骤4：消息触达统计配置

如果您需要统计触达数据，请按照如下完成配置：

华为

荣耀

vivo

魅族

回执地址：

新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/huawei

©2013-2025 Tencent Cloud International Pte. Ltd.

韩　国：https://apikr.im.qcloud.com/v3/offline_push_report/huawei
美　国：https://apiusa.im.qcloud.com/v3/offline_push_report/huawei
德　国：https://apiger.im.qcloud.com/v3/offline_push_report/huawei
印　尼：https://apiidn.im.qcloud.com/v3/offline_push_report/huawei
中　国：https://api.im.qcloud.com/v3/offline_push_report/huawei

注意：

华为推送证书 ID <= 11344 ，使用华为推送 v2 版本接口，不支持触达和点击回执，请重新生成更新证书 ID 。

回执地址：
新加坡：https://apisgp.im.qcloud.com/v3/offline_push_report/honor
韩　国：https://apikr.im.qcloud.com/v3/offline_push_report/honor
美　国：https://apiusa.im.qcloud.com/v3/offline_push_report/honor
德　国：https://apiger.im.qcloud.com/v3/offline_push_report/honor
印　尼：https://apiidn.im.qcloud.com/v3/offline_push_report/honor
中　国：https://api.im.qcloud.com/v3/offline_push_report/honor

回调地址配置	回执 ID 配置 IM 控制台
回执地址： 新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/vivo 韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/vivo 美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/vivo 德　国： https://apiger.im.qcloud.com/v3/offline_push_report/vivo 印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/vivo 中　国： https://api.im.qcloud.com/v3/offline_push_report/vivo	

打开回执开关	配置回执地址

©2013-2025 Tencent Cloud International Pte. Ltd.

回执地址：

新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/meizu

韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/meizu

美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/meizu

德　国： https://apiger.im.qcloud.com/v3/offline_push_report/meizu

印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/meizu

中　国： https://api.im.qcloud.com/v3/offline_push_report/meizu

注意：

打开回执开关后，请务必确保回执地址正确配置。不配置或者配置地址错误，都会影响推送功能。

说明：

其他支持厂商无需进行消息触达统计配置。

FCM 暂不支持推送统计功能。

步骤5：发消息时设置离线推送参数

调用 sendMessage 发送消息时，您可以通过 V2TIMOfflinePushInfo 设置离线推送参数。调用 V2TIMOfflinePushInfo 的setExt 设置自定义 ext 数据，当用户收到离线推送启动 App 的时候，可以在单击通知跳转的回调中获取到 ext 字段，然后根据 ext 字段内容跳转到指定的 UI 界面。可以参见 ChatProvider 的 sendMessage() 方法：

```
V2TIMOfflinePushInfo v2TIMOfflinePushInfo = new V2TIMOfflinePushInfo();
v2TIMOfflinePushInfo.setTitle("推送标题");
v2TIMOfflinePushInfo.setDesc("推送内容");

OfflinePushExtInfo offlinePushExtInfo = new OfflinePushExtInfo();
offlinePushExtInfo.getBusinessInfo().setSenderId("senderID");
offlinePushExtInfo.getBusinessInfo().setSenderNickName("senderNickName");
if (chatInfo.getType() == V2TIMConversation.V2TIM_GROUP) {
    offlinePushExtInfo.getBusinessInfo().setChatType(V2TIMConversation.V2TIM_GROUP)
    offlinePushExtInfo.getBusinessInfo().setSenderId("groupID");
}
v2TIMOfflinePushInfo.setExt(new Gson().toJson(offlinePushExtInfo).getBytes());

// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
v2TIMOfflinePushInfo.setAndroidOPPOChannelID("tuikit");
v2TIMOfflinePushInfo.setAndroidHuaWeiCategory("IM");
v2TIMOfflinePushInfo.setAndroidVIVOCategory("IM");

final V2TIMMessage v2TIMMessage = message.getTimMessage();
String msgID = V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, isGroup ?
    V2TIMMessage.V2TIM_PRIORITY_DEFAULT, false, v2TIMOfflinePushInfo, new V2TIMSend
        @Override
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        public void onProgress(int progress) {

        }

        @Override
        public void onError(int code, String desc) {
            TUIChatUtils.callbackOnError(callBack, TAG, code, desc);
        }

        @Override
        public void onSuccess(V2TIMMessage v2TIMMessage) {
            TUIChatLog.v(TAG, "sendMessage onSuccess:" + v2TIMMessage.getMsgID());
            message.setMsgTime(v2TIMMessage.getTimestamp());
            TUIChatUtils.callbackOnSuccess(callBack, message);
        }
    });
```

步骤6：解析离线推送消息

收到推送消息后点击通知栏，组件会回调该点击事件和透传离线消息。

自定义点击跳转实现

自定义点击跳转实现（旧方案）

注意：

1. 注册回调时机建议放在应用 Application 的 oncreate() 函数中。
2. 控制台配置点击后续动作按如下配置，选择 **打开应用内指定界面,** 请勿修改使用默认值。

```
TIMPushManager.getInstance().addPushListener(new TIMPushListener() {
    @Override
    public void onNotificationClicked(String ext) {
        Log.d(TAG, "onNotificationClicked =" + ext);
        // 获取 ext 自定义跳转

        // 示例：跳转到对应聊天界面
        OfflinePushExtInfo offlinePushExtInfo = null;
        try {
            offlinePushExtInfo = new Gson().fromJson(extString, OfflinePushExtInfo.
            if (offlinePushExtInfo.getBusinessInfo().getChatAction() == OfflinePush
                String senderId = offlinePushExtInfo.getBusinessInfo().getSenderId(
                if (TextUtils.isEmpty(senderId)) {
                    return;
                }
                TUIUtils.startChat(senderId, offlinePushExtInfo.getBusinessInfo().g
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            }
        } catch (Exception e) {
            Log.e(TAG, "getOfflinePushExtInfo e: " + e);
        }
    }
});
```

组件会以回调或者广播形式通知应用，应用在回调中配置 App 的跳转页面即可。

注意：

1. 注册回调时机建议放在应用 Application 的 oncreate() 函数中。
2. 控制台配置点击后续动作按如下配置，选择 **打开应用内指定界面,** 请勿修改使用默认值。

1. 回调方式如下：

```
TUICore.registerEvent(TUIConstants.TIMPush.EVENT_NOTIFY, TUIConstants.TIMPush.EVENT
        @Override
        public void onNotifyEvent(String key, String subKey, Map<String, Object> pa
            Log.d(TAG, "onNotifyEvent key = " + key + "subKey = " + subKey);
            if (TUIConstants.TIMPush.EVENT_NOTIFY.equals(key)) {
                if (TUIConstants.TIMPush.EVENT_NOTIFY_NOTIFICATION.equals(subKey))
                    if (param != null) {
                        String extString = (String)param.get(TUIConstants.TIMPush.N
                        // 获取 ext 自定义跳转

                        // 示例：跳转到对应聊天界面
                        OfflinePushExtInfo offlinePushExtInfo = null;
                        try {
                            offlinePushExtInfo = new Gson().fromJson(extString, Off
                            if (offlinePushExtInfo.getBusinessInfo().getChatAction(
                                String senderId = offlinePushExtInfo.getBusinessInf
                                if (TextUtils.isEmpty(senderId)) {
                                    return;
                                }
                                TUIUtils.startChat(senderId, offlinePushExtInfo.get
                            }
                        } catch (Exception e) {
                            Log.e(TAG, "getOfflinePushExtInfo e: " + e);
                        }
                    }
                }
            }
        }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    });
```

2. 广播方式如下：

```
/ 动态注册广播
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(TUIConstants.TIMPush.NOTIFICATION_BROADCAST_ACTION);
LocalBroadcastManager.getInstance(context).registerReceiver(localReceiver, intentFi

//广播接收者
public class OfflinePushLocalReceiver extends BroadcastReceiver {
    public static final String TAG = OfflinePushLocalReceiver.class.getSimpleName()

    @Override
    public void onReceive(Context context, Intent intent) {
        DemoLog.d(TAG, "BROADCAST_PUSH_RECEIVER intent = " + intent);
        if (intent != null) {
            String ext = intent.getStringExtra(TUIConstants.TIMPush.NOTIFICATION_EX
            // 获取 ext 自定义跳转

            // 示例：跳转到对应聊天界面
            OfflinePushExtInfo offlinePushExtInfo = null;
            try {
                offlinePushExtInfo = new Gson().fromJson(extString, OfflinePushExtI
                if (offlinePushExtInfo.getBusinessInfo().getChatAction() == Offline
                    String senderId = offlinePushExtInfo.getBusinessInfo().getSende
                    if (TextUtils.isEmpty(senderId)) {
                        return;
                    }
                    TUIUtils.startChat(senderId, offlinePushExtInfo.getBusinessInfo
                }
             } catch (Exception e) {
                 Log.e(TAG, "getOfflinePushExtInfo e: " + e);
             }
        } else {
            Log.e(TAG, "onReceive ext is null");
        }
    }
}
```

恭喜您已经完成了推送插件的接入，需要提醒您：推送插件**试用或购买到期后，将自动停止提供推送服务（包括普通消息离线推送、全员/标签推送等服务）**。为避免影响您业务正常使用，请提前购买/续费。

说明：

1. 厂商离线通道都有 消息分类机制，不同类型也会有不同的推送策略。

如果推送需求属于 IM 类型推送，想要推送及时触达，需要按照厂商规则设置自己应用为对应的推送类型，会归类为高优先级的系统消息类型或者重要消息类型。

©2013-2025 Tencent Cloud International Pte. Ltd.

反之，离线推送会有数量和频次的限制，可能不会及时推送到设备。

2. 接入完成收不到推送，请先自助使用 排查工具 查看下具体原因。推送指标数据查看，请使用 数据统计 查询。

3. 全员/标签推送功能请参见：REST API 接口 - 发起全员/标签推送。

©2013-2025 Tencent Cloud International Pte. Ltd.

客户端 API

最近更新时间：2025-03-24 11:22:47

TIMPush - TIMPushManager

public abstract class TIMPushManager：推送插件接口类。

接口概览

注册/反注册推送服务接口

API	描述
registerPush	注册推送服务，推送信息读取工程中的配置文件 timpush-configs.json (必须在 App 用户同意了隐私政策后，再调用该接口使用推送服务)。
unRegisterPush	反注册关闭推送服务。
setRegistrationID	RegistrationID 是推送接收设备的唯一标识 ID。默认情况下，注册推送服务成功时自动生成该 ID，同时也支持您自定义设置。您可根据 RegistrationID 向指定设备推送消息。需要注意的是，卸载并重新安装设备会更改 RegistrationID，因此需要在注册推送服务之前调用 setRegistrationID 接口。
getRegistrationID	在成功注册推送服务后，可通过调用 getRegistrationID 接口获取推送接收设备的唯一标识 ID，即RegistrationID。您可根据 RegistrationID 向指定设备推送消息。

Push 全局监听接口

API	描述
addPushListener	添加 Push 监听器。
removePushListener	移除 Push 监听器。

自定义配置接口

API	描述
forceUseFCMPushChannel	指定设备离线推送使用 FCM 通道，需要在注册推送服务之前调用。
disablePostNotificationInForeground	关闭 App 在前台时弹出通知栏。

©2013-2025 Tencent Cloud International Pte. Ltd.

接口详情

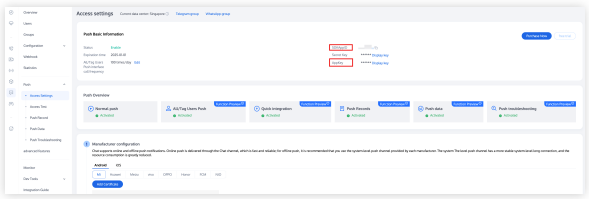
静态 Public 成员函数

static TIMPushManager getInstance()：获取 TIMPushManager 管理器实例。

成员函数说明

abstract void registerPush(Context context, int sdkAppId, String appKey, TIMPushCallback callback)

注册离线推送服务：请正确传递 sdkAppId 和 appKey 两个参数，即可注册推送服务。
注意：如果您已经集成 Chat 产品，请在 Chat 登录成功后调用该接口，将 appKey 参数设置为 null，再接入离线推送能力。
参数说明：

参数	描述	获取路径
sdkAppId	Chat 控制台为您分配的应用 ID。	
appKey	Chat 控制台为您分配的客户端密钥。	

注意：
您需要使用 TUICore 组件中的 TUILogin 提供的 login 接口登录，插件会自动感知并注册推送服务。
如果您不想使用 TUILogin 提供的接口，您需要在完成登录操作后，手动调用该接口注册服务。

abstract void unRegisterPush(TIMPushCallback callback)

反注册关闭离线推送服务，Chat 账号登出前调用。
注意：
如果您不想使用推送服务，手动调用该接口反注册服务即可。
如果您使用 TUICore 组件中的 TUILogin 提供的 logout 接口登出，插件会自动感知并反注册推送服务。

abstract void setRegistrationID(String registrationID, TIMPushCallback callback)

设置注册推送服务使用的推送 ID 标识, 即 RegistrationID，需要在注册推送服务之前调用。
参数说明：

参数	描述
registrationID	设备的推送唯一标识 ID，卸载重装会改变。

abstract void getRegistrationID(TIMPushCallback callback)

©2013-2025 Tencent Cloud International Pte. Ltd.

注册推送服务成功后，获取推送 ID 标识，即 RegistrationID。

abstract void addPushListener(TIMPushListener listener)

添加 Push 监听器

abstract void removePushListener(TIMPushListener listener)

移除 Push 监听器

abstract void forceUseFCMPushChannel(boolean enable)

指定设备离线推送使用 FCM 通道，需要在注册推送服务之前调用。
参数说明：

参数	描述
enable	true：使用 FCM 通道。 false：使用本机通道。

abstract void disablePostNotificationInForeground(boolean disable)

关闭 App 在前台时弹出通知栏，默认关闭。
参数说明：

参数	描述
disable	true：关闭。 false：开启。

TIMPush - TIMPushListener

public abstract class TIMPushListener：Push 监听器类

接口概览

API	描述
onRecvPushMessage	收到 Push 消息。
onRevokePushMessage	收到 Push 消息撤回的通知。
onNotificationClicked	点击通知栏消息回调。

接口详情

成员函数说明

©2013-2025 Tencent Cloud International Pte. Ltd.

void onRecvPushMessage(TIMPushMessage message)

收到 Push 消息，message 消息。

void onRevokePushMessage(String messageID)

收到 Push 消息撤回的通知，messageID 消息唯一标识。

void onNotificationClicked(String ext)

点击通知栏消息回调。

注意：

控制台推送证书需要配置为"打开应用内指定界面"，并使用默认填充值生效。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS
厂商配置

最近更新时间：2025-03-24 11:23:26

本文将在您拥有**开发者账号**和**开发/分发证书**的前提下，引导您创建和使用**推送证书**及**描述文件**。
如果您对“开发者账号”或“开发/分发证书”有任何疑问，可以参考附录中的相关概念。

一、推送证书配置（APNs）

集成 TIMPush 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 Chat 控制台 。之后按照快速接入步骤接入即可。
Apple 厂商配置目前有两种主流的证书，p12 证书和 p8 证书。两种证书各有优劣，您可按需要选择其中的一种。

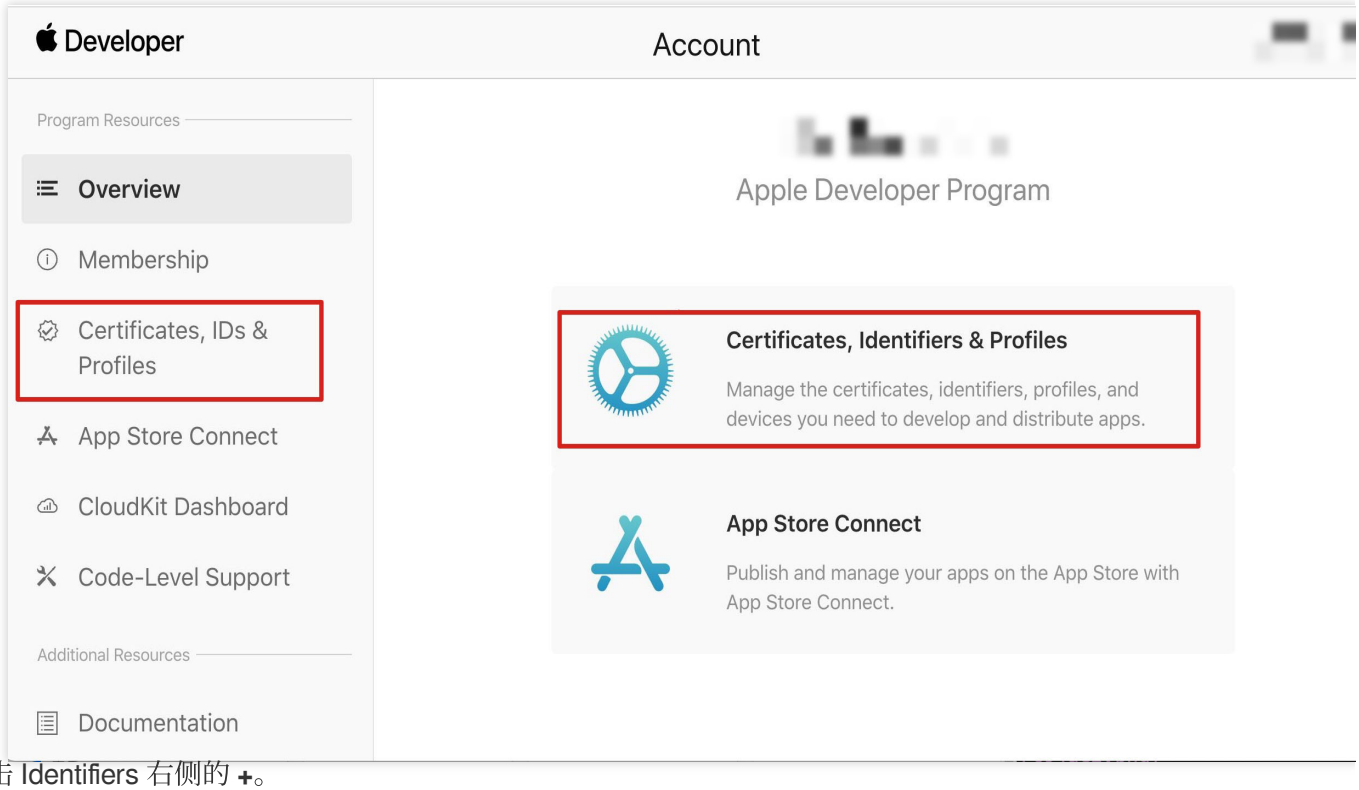
	证书类型	有效期和管理	安全性	灵动岛
p12 证书	p12 证书是一个包含公钥和私钥的二进制文件，用于基于证书的身份验证。它将公钥证书和私钥捆绑在一个文件中，扩展名为 .p12 或 .pfx。	p12 证书通常有一年的有效期，过期后需要重新生成和部署。每个应用程序都需要单独的 P12 证书来处理推送通知。	证书：p12 证书使用基于证书的身份验证，需要在服务器上存储私钥。这可能会增加安全风险，因为私钥可能会被未经授权的用户访问。	不支持
p8 证书	p8 证书是一个 Auth Key（授权密钥），用于基于令牌的身份验证。它是一个包含私钥的文本文件，扩展名为 .p8。	p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 P8 证书可以简化证书管理，因为您可以使用一个 p8 证书为多个应用程序提供推送通知服务。	p8 证书使用基于令牌的身份验证，这意味着您的服务器会周期性地生成一个 JSON Web Token（JWT）来与 APNs 建立连接。这种方法更安全，因为它不需要在服务器上存储私钥。	支持灵动岛推送

方式一：使用 p12 证书 （传统APNs推送证书）

步骤1：申请 APNs 证书

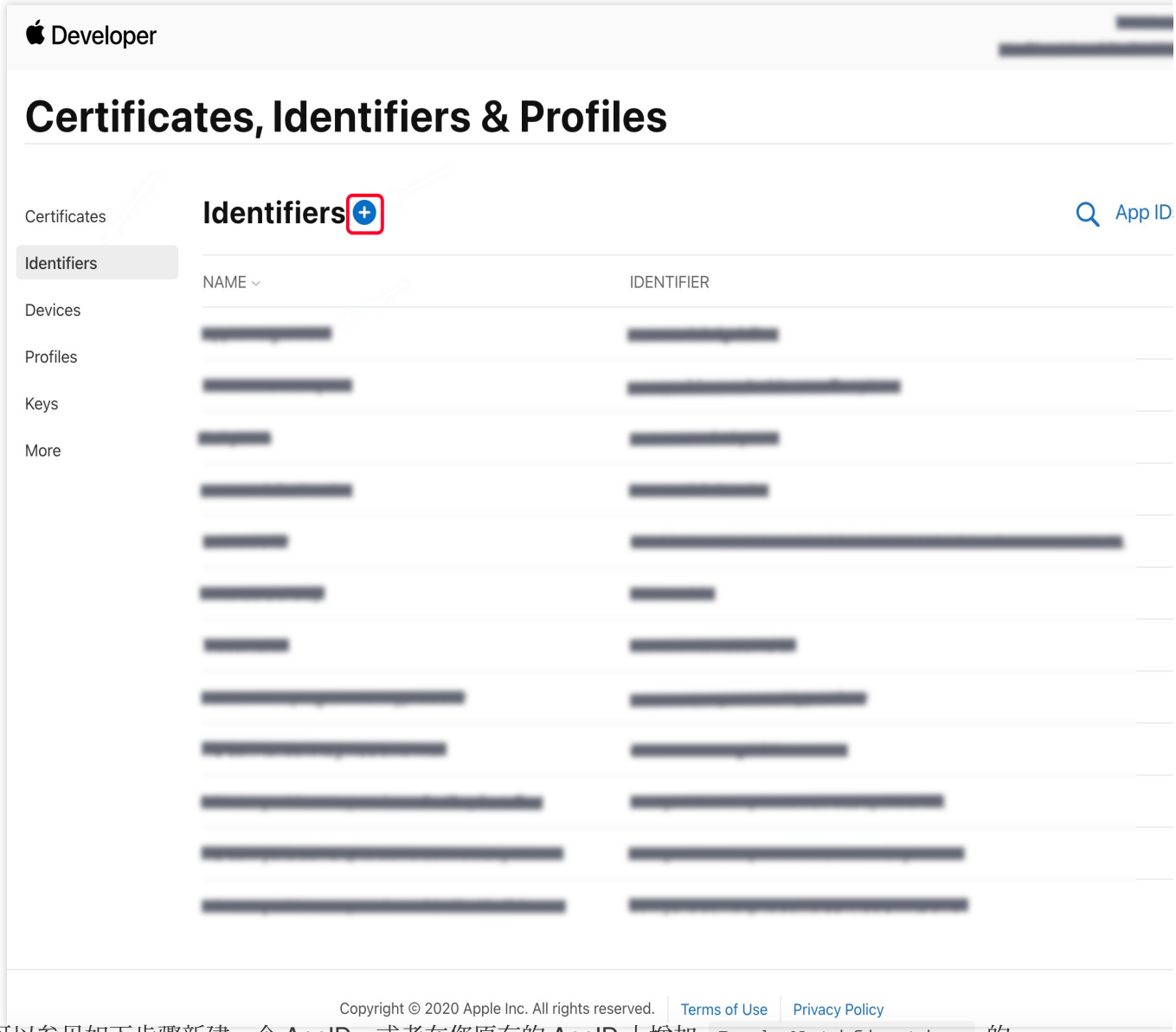
1. 登录 苹果开发者中心 网站，单击 **Certificates,Identifiers & Profiles** 或者侧栏的 **Certificates,IDs & Profiles**，进入 **Certificates, IDS & Profiles** 页面。

©2013-2025 Tencent Cloud International Pte. Ltd.



2. 单击 Identifiers 右侧的 +。

©2013-2025 Tencent Cloud International Pte. Ltd.



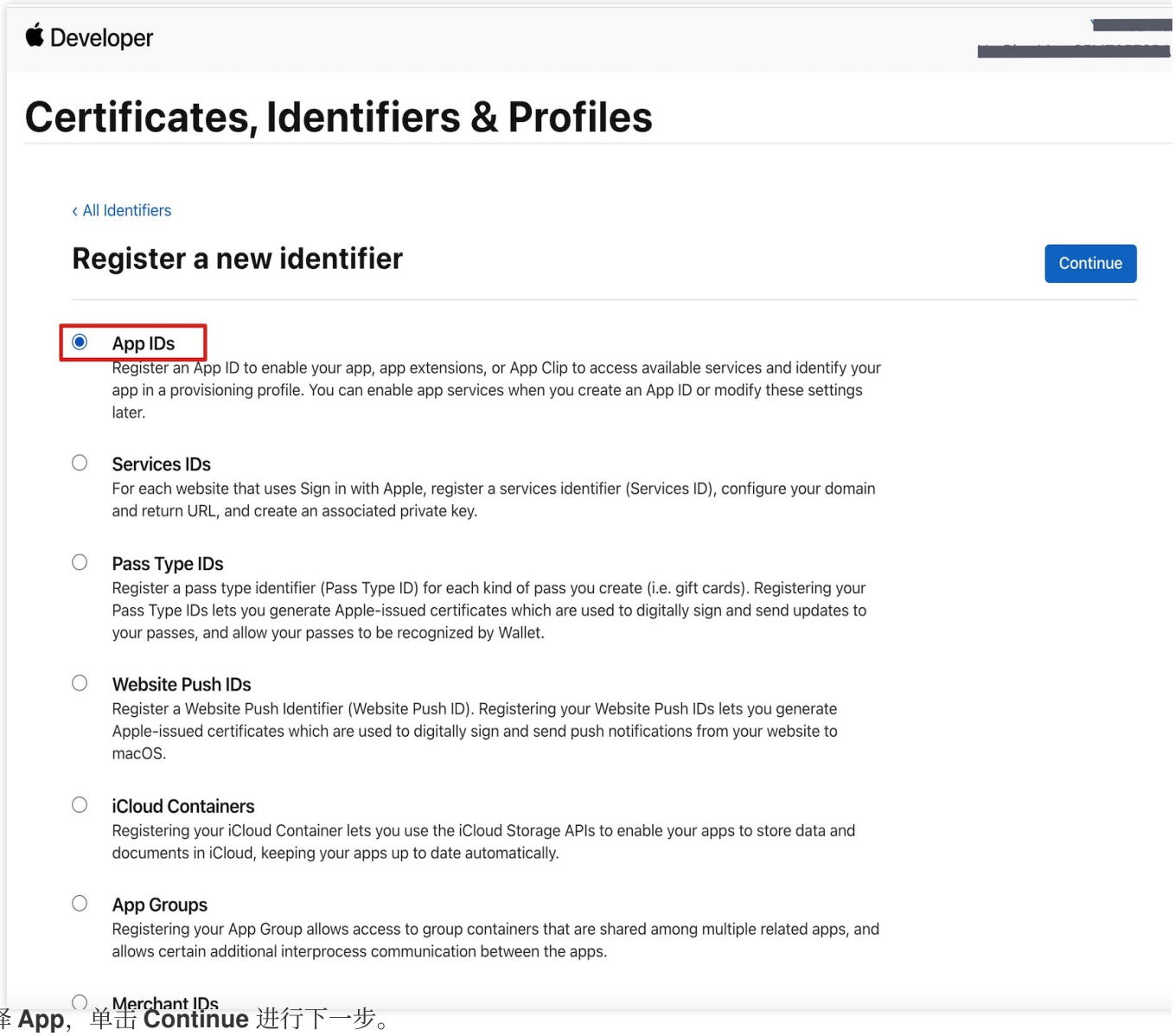
3. 您可以参见如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service` 。

说明：

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

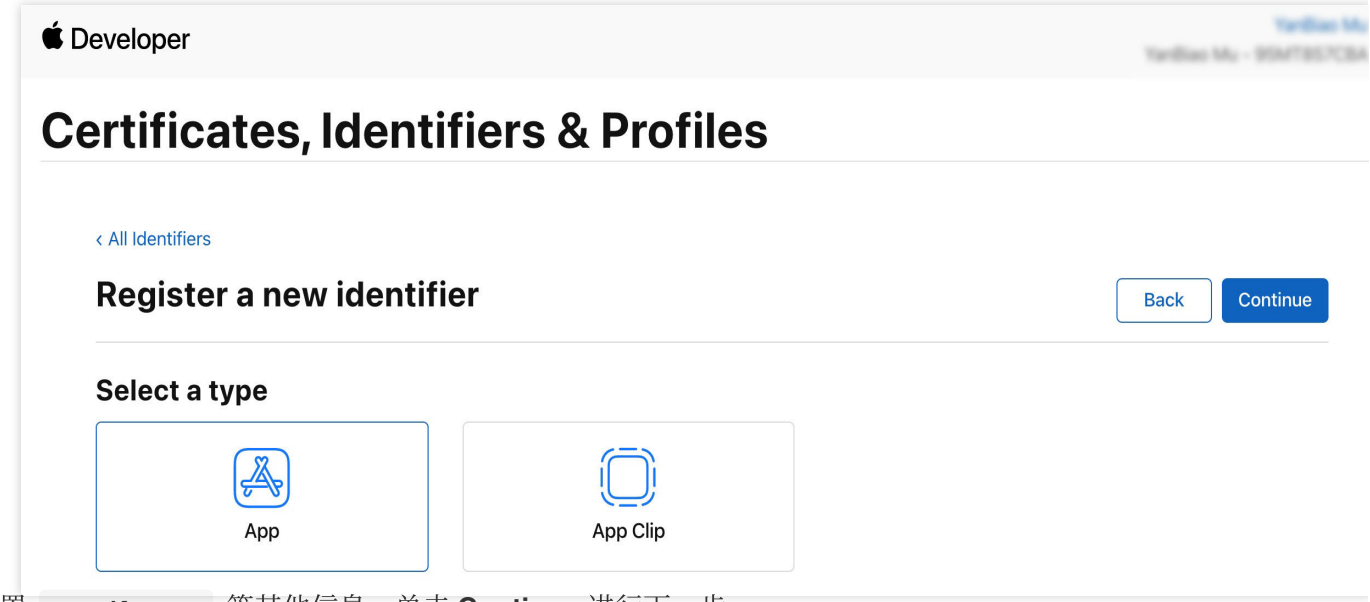
4. 勾选 **App IDs**，单击 **Continue** 进行下一步。

©2013-2025 Tencent Cloud International Pte. Ltd.

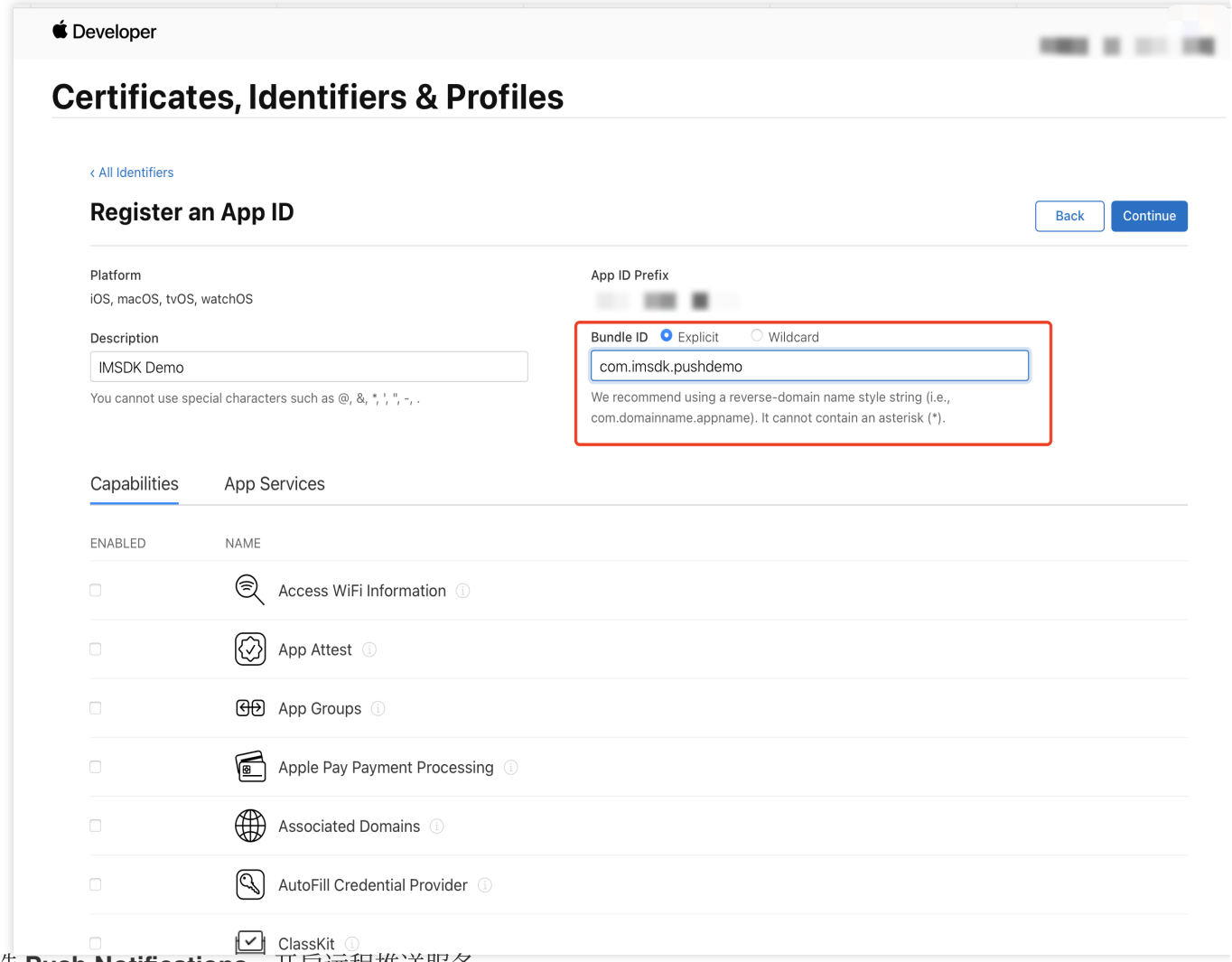


5. 选择 **App**，单击 **Continue** 进行下一步。

©2013-2025 Tencent Cloud International Pte. Ltd.

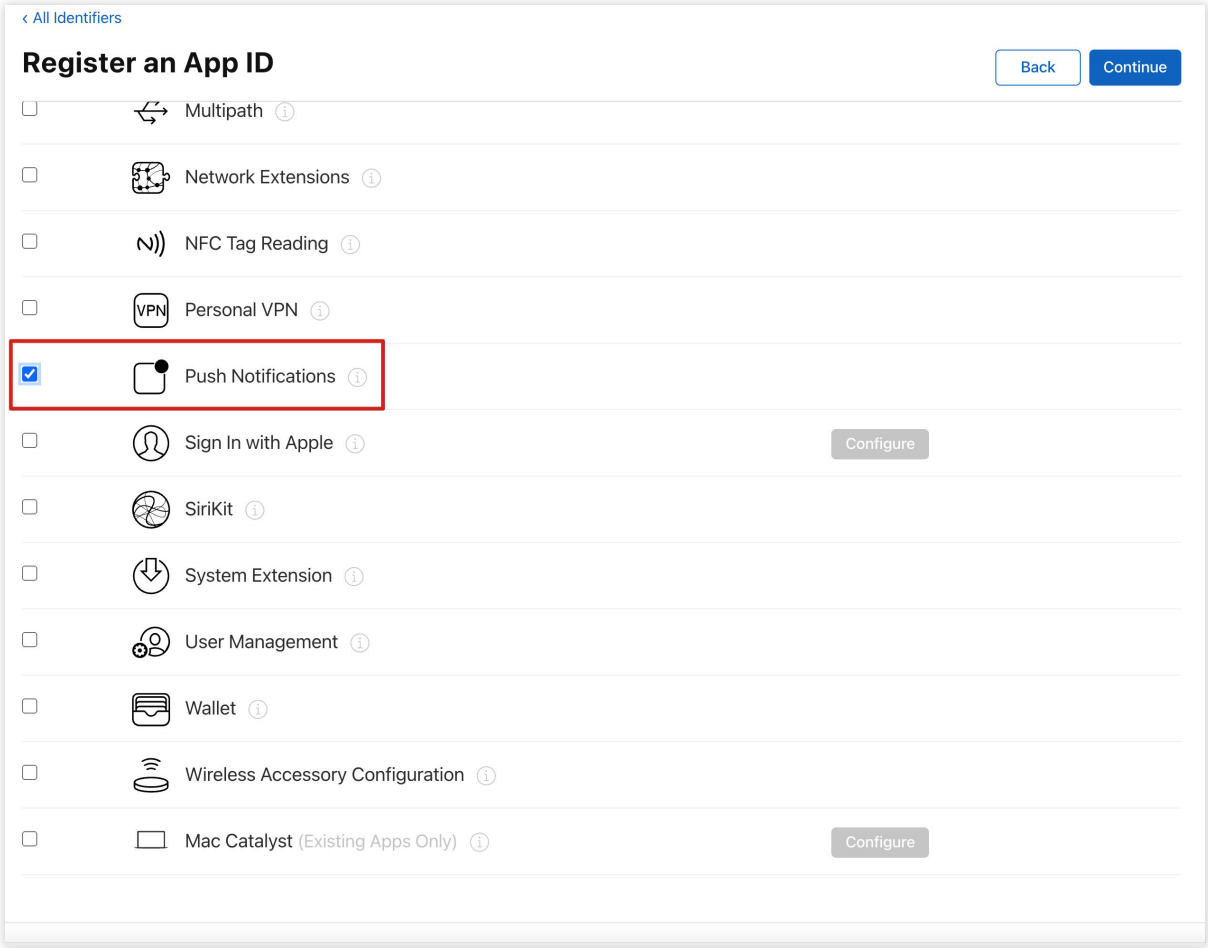


6. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。



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

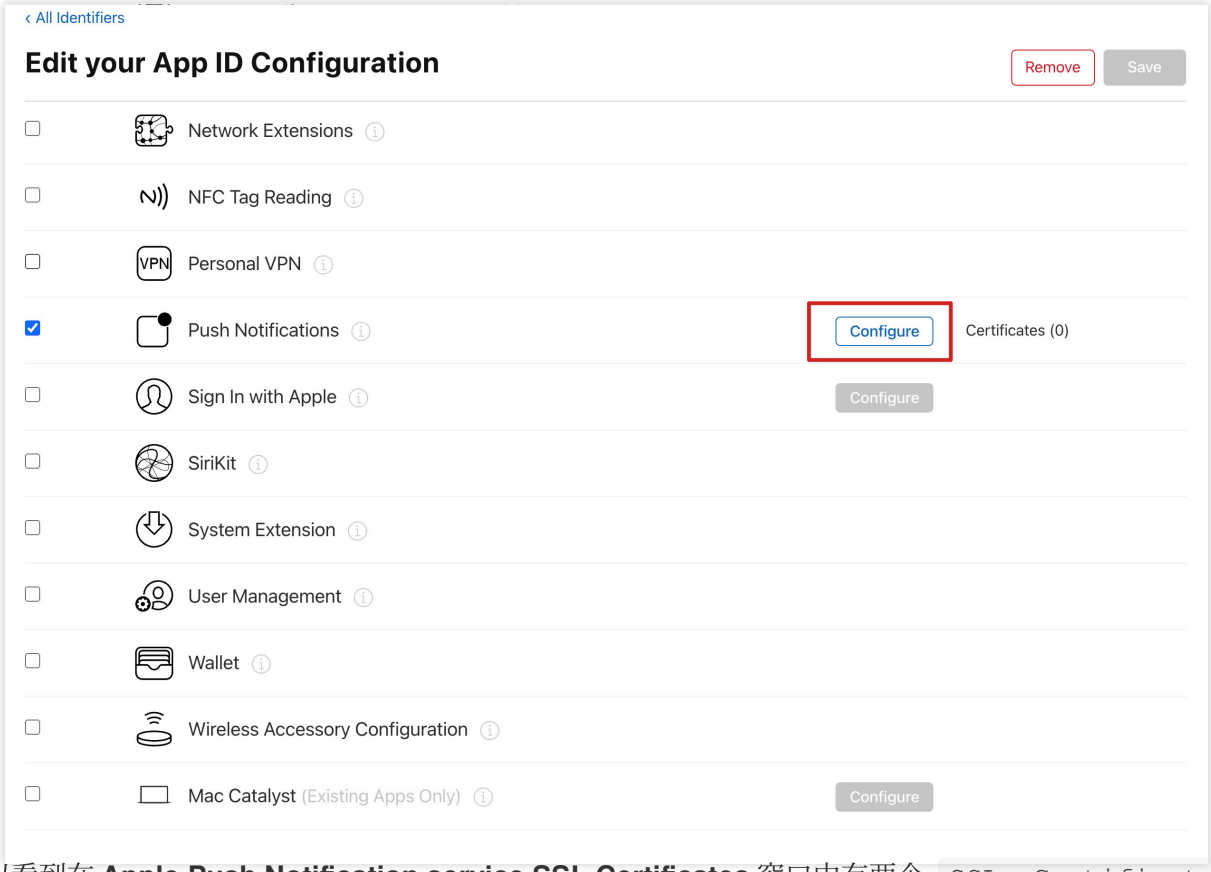
©2013-2025 Tencent Cloud International Pte. Ltd.



生成证书

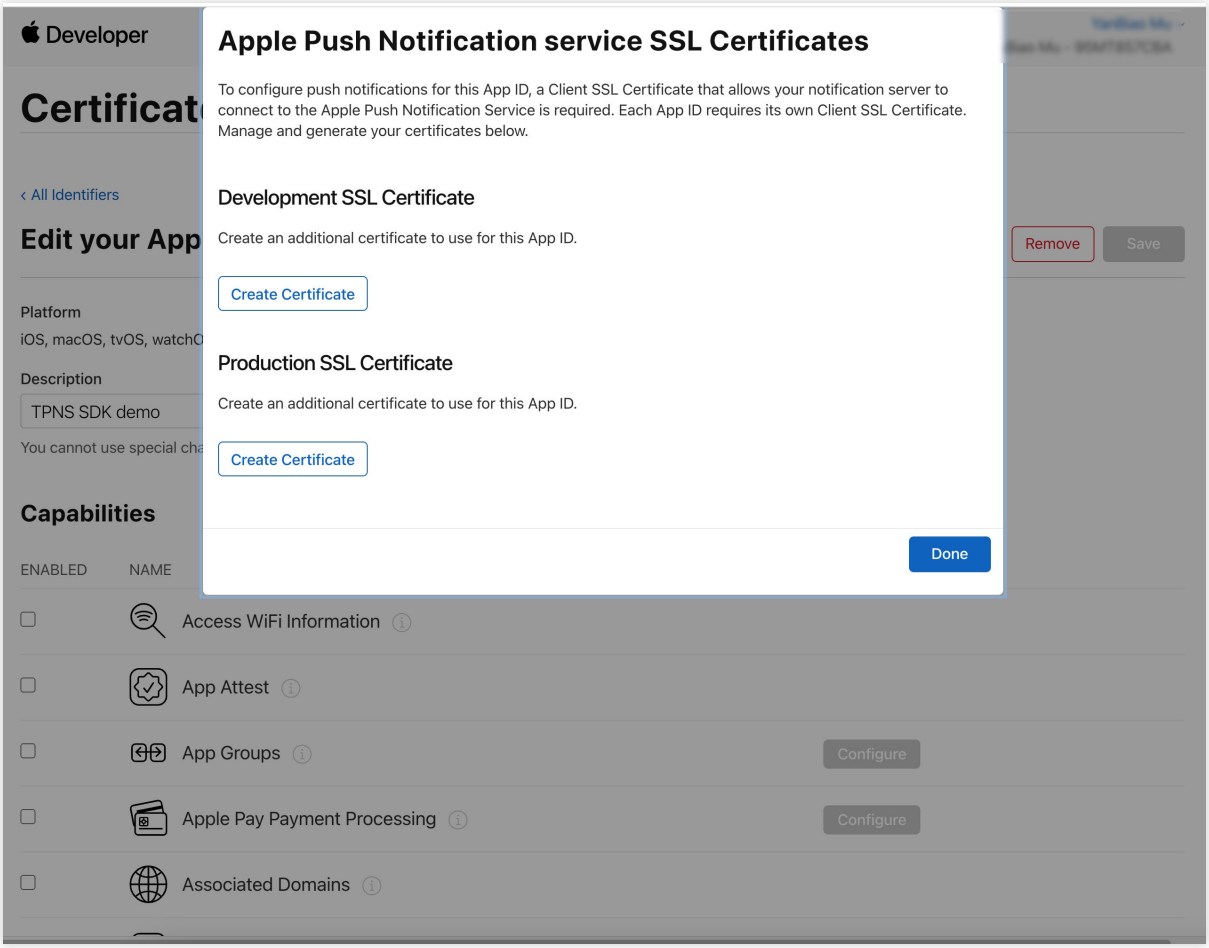
1. 选中您的 AppID，选择 **Configure**。

©2013-2025 Tencent Cloud International Pte. Ltd.



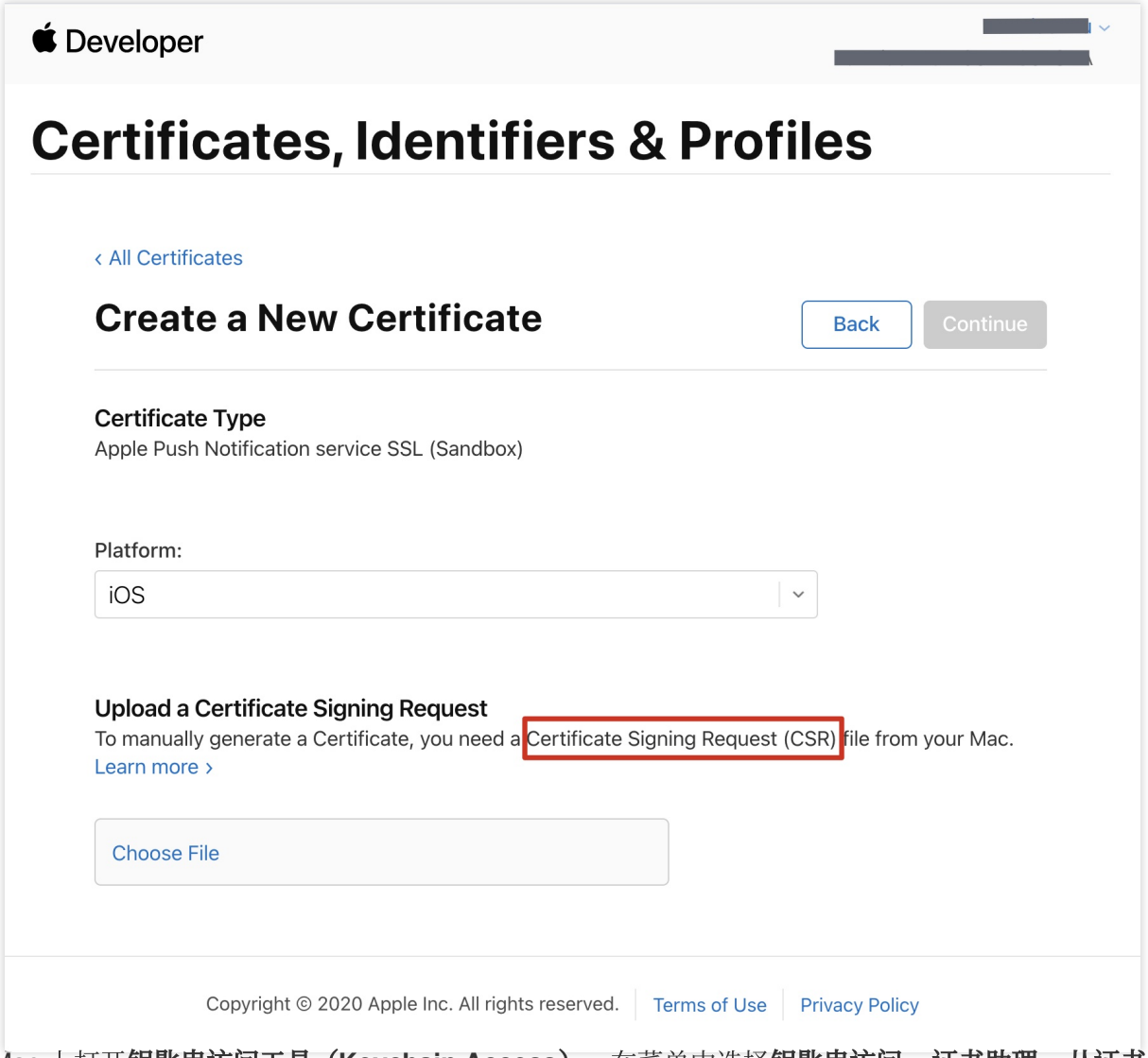
2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate`，分别用于开发环境（Development）和生产环境（Production）的远程推送证书，如下图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.

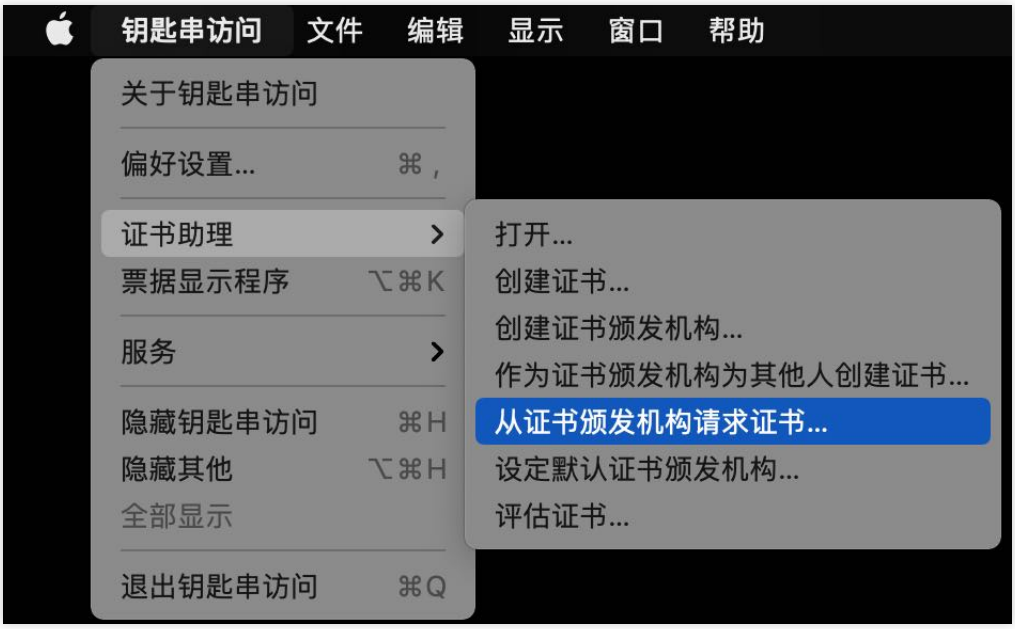


3. 我们先选择开发环境（Development）的 **Create Certificate**，系统将提示我们需要一个 Certificate Signing Request（CSR）。

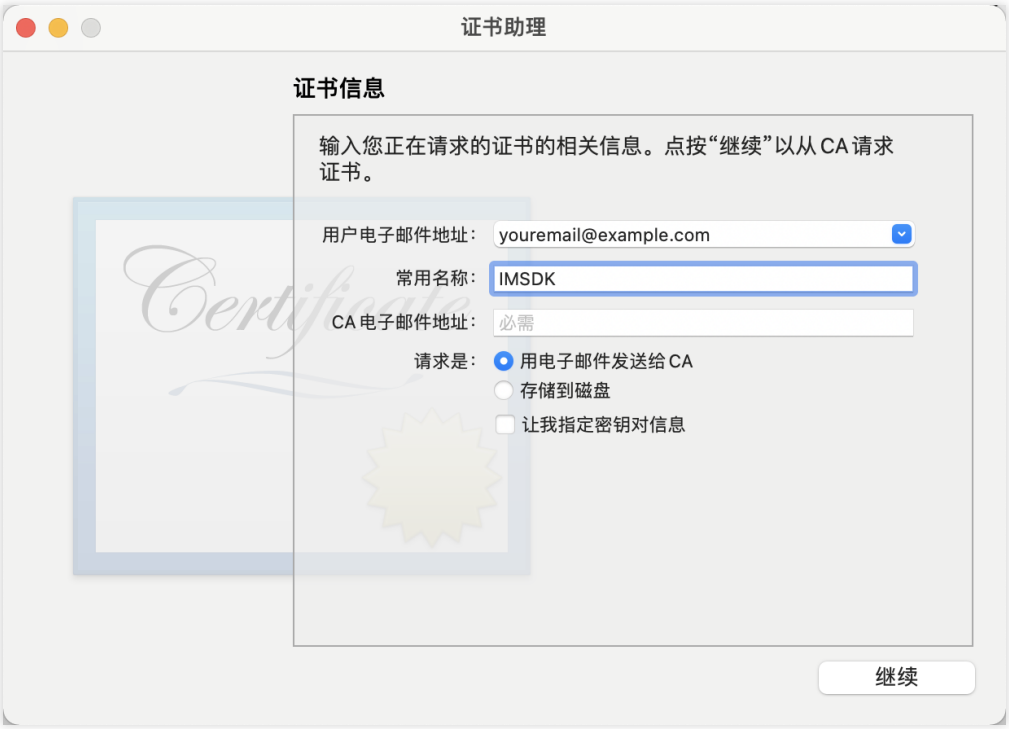
©2013-2025 Tencent Cloud International Pte. Ltd.



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

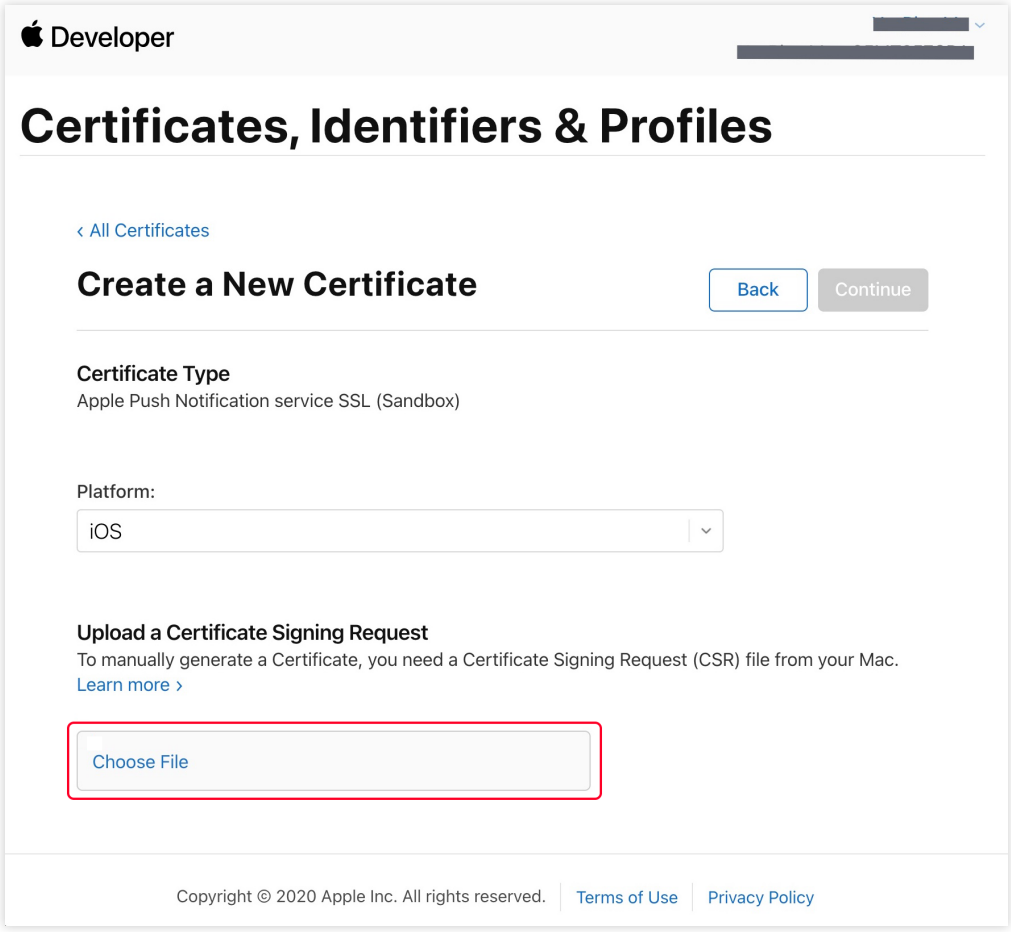


5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。



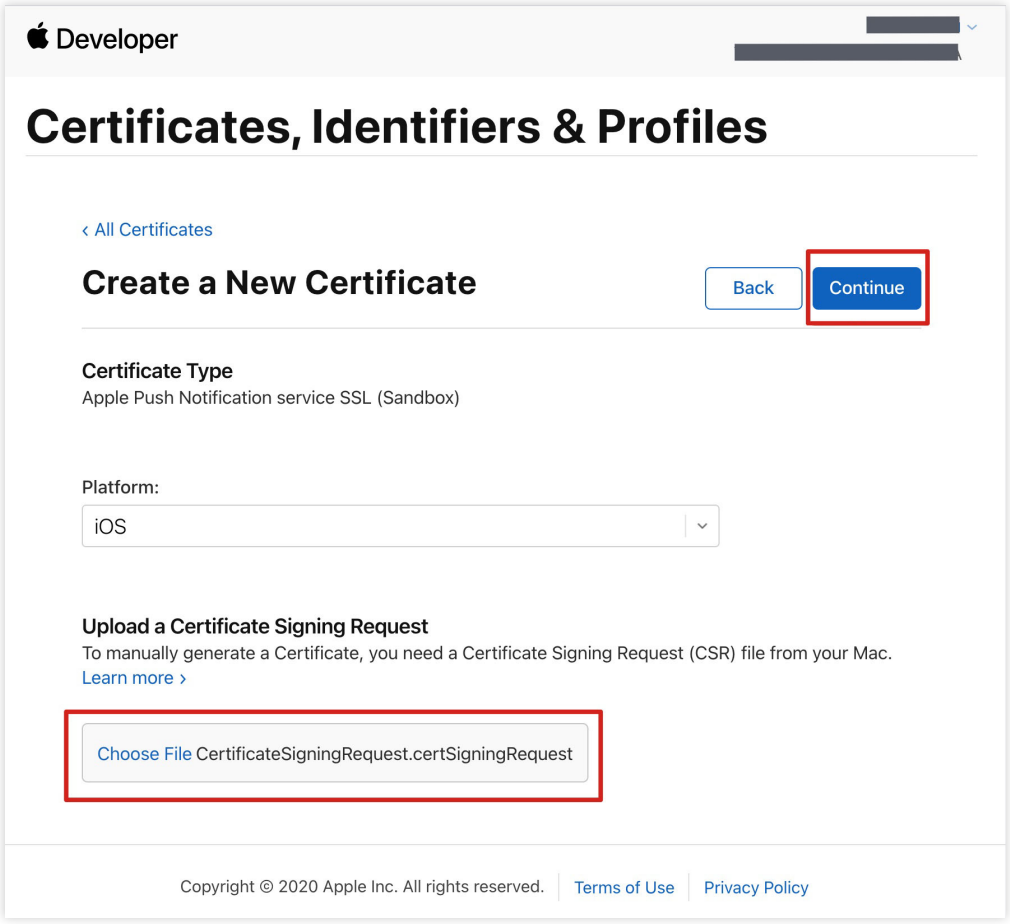
6. 返回上述 步骤3 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

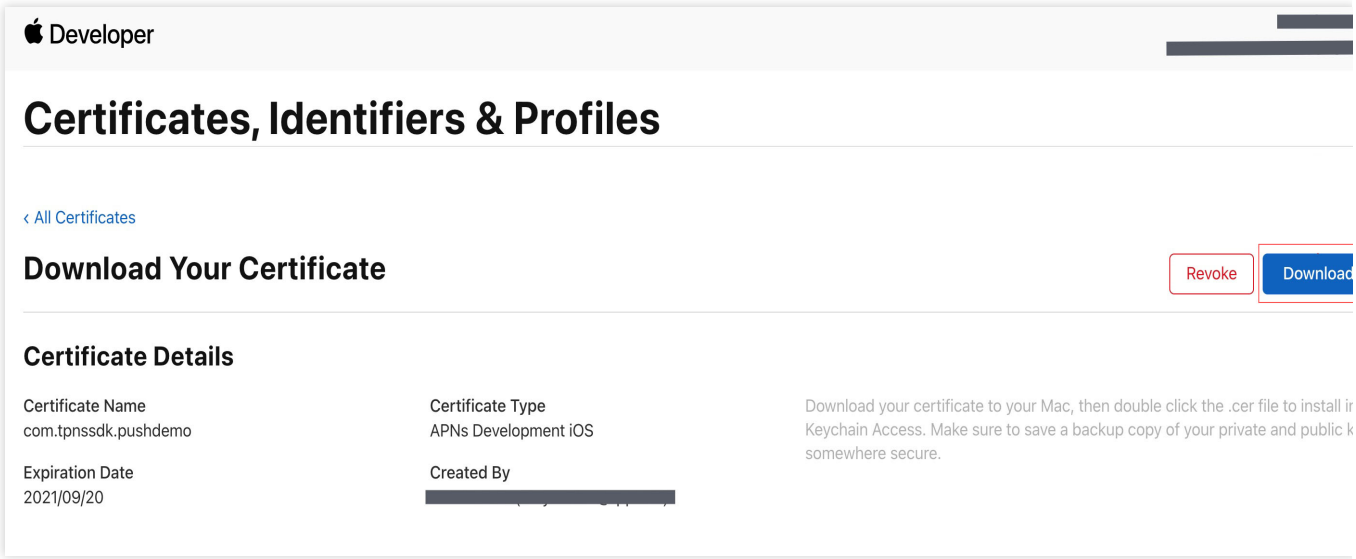


7. 单击 **Continue**，即可生成推送证书。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

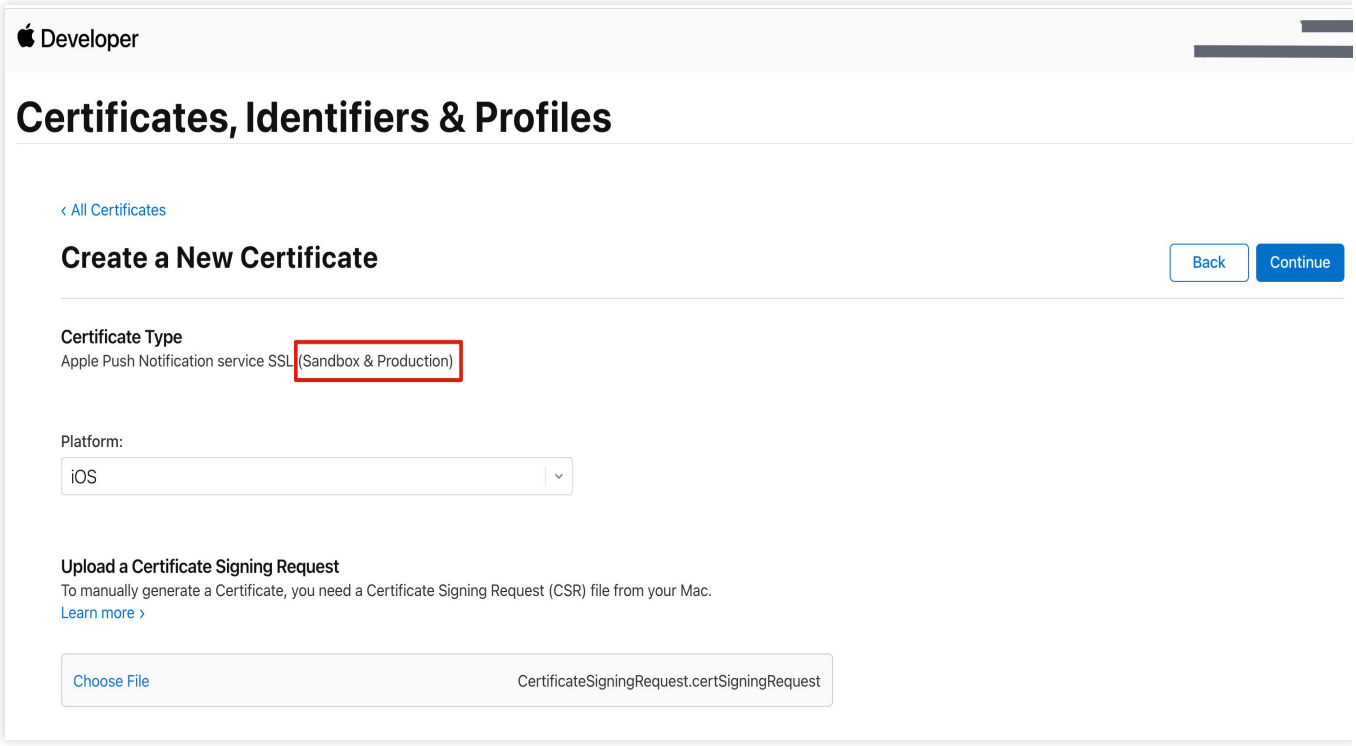


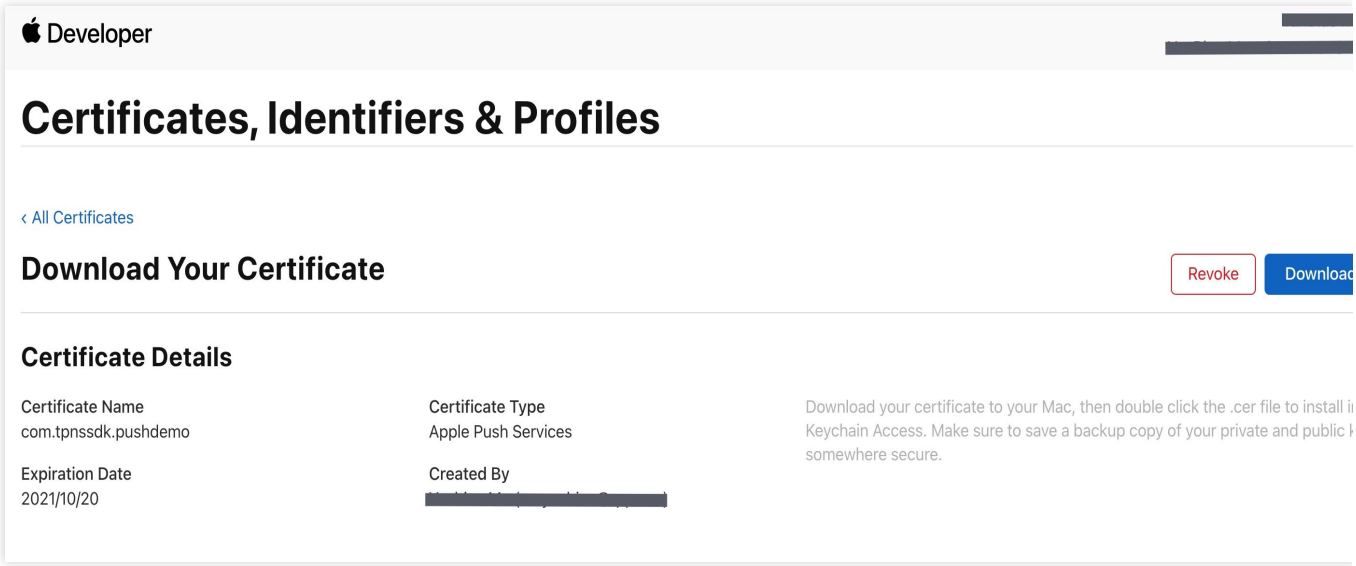
9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

说明

©2013-2025 Tencent Cloud International Pte. Ltd.

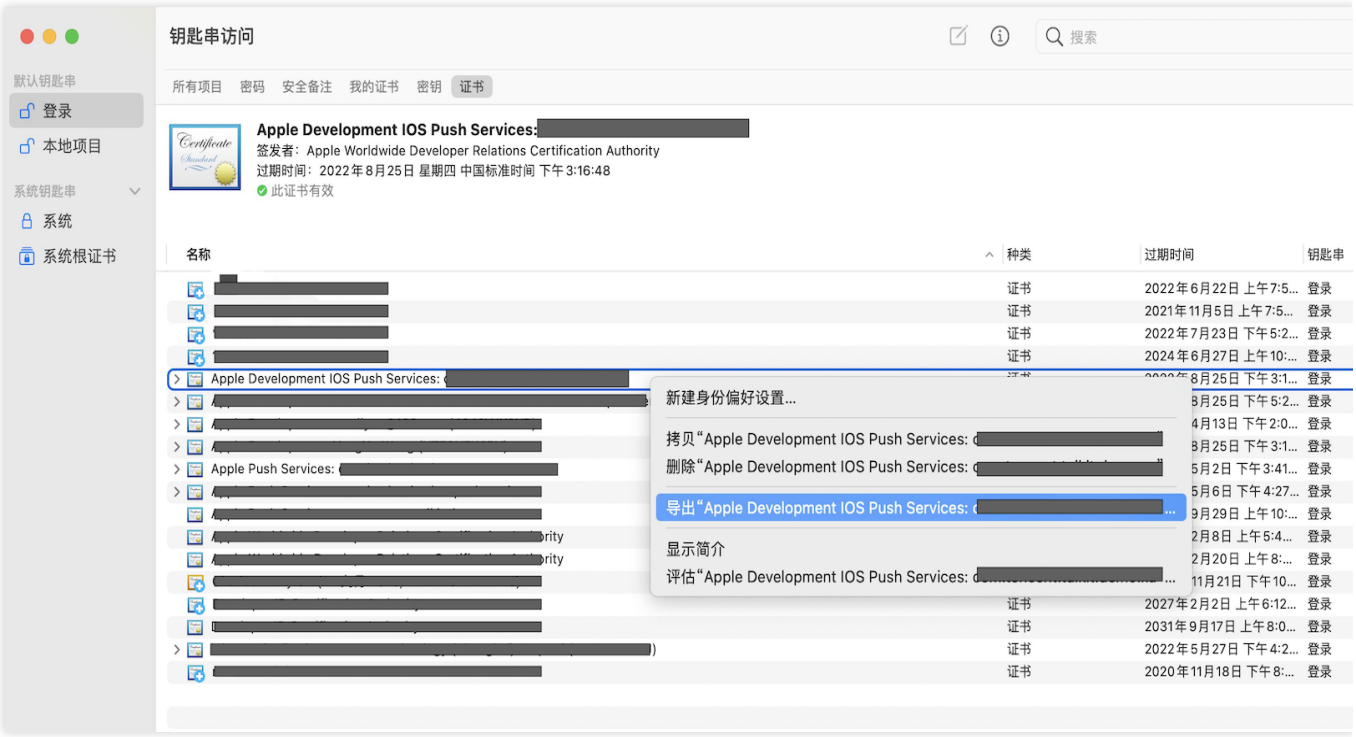
生产环境的证书实际是开发（Sandbox）+生产（Production）的合并证书，可以同时作为开发环境和生产环境的证书使用。





10. 双击打开下载的开发环境和生产环境的 `SSL Certificate`，系统会将其导入钥匙串中。
11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（`Apple Development IOS Push Service`）和生产环境（`Apple Push Services`）的 `p12` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

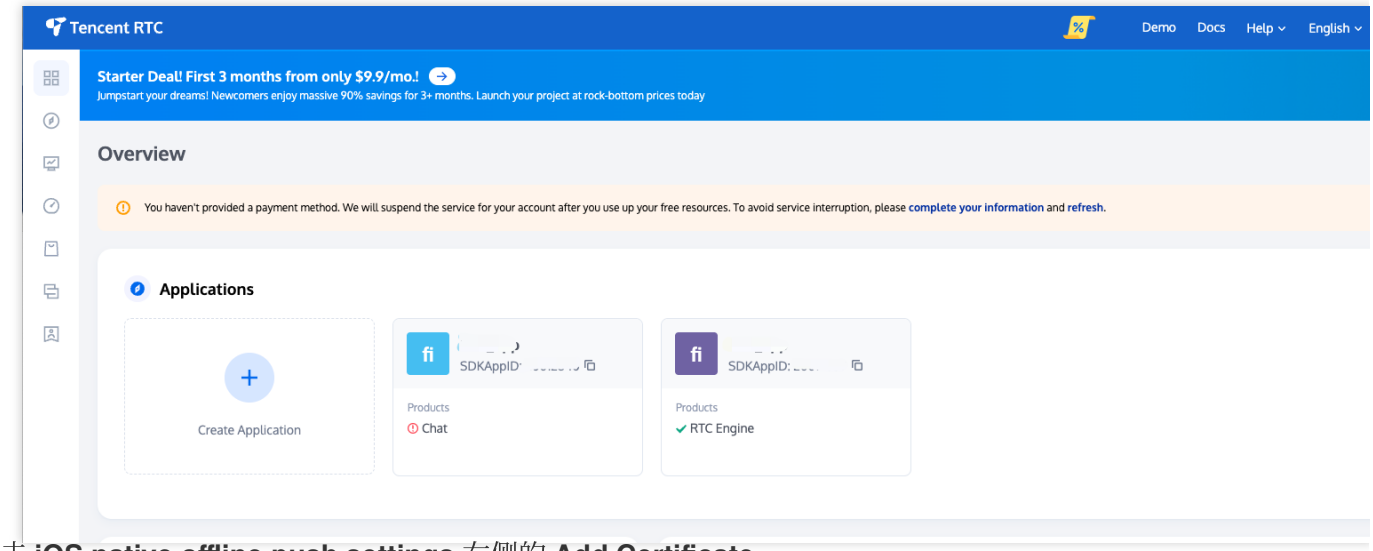


注意

保存 `.p12` 文件时，请务必要为其设置密码。步骤2：上传证书到控制台。

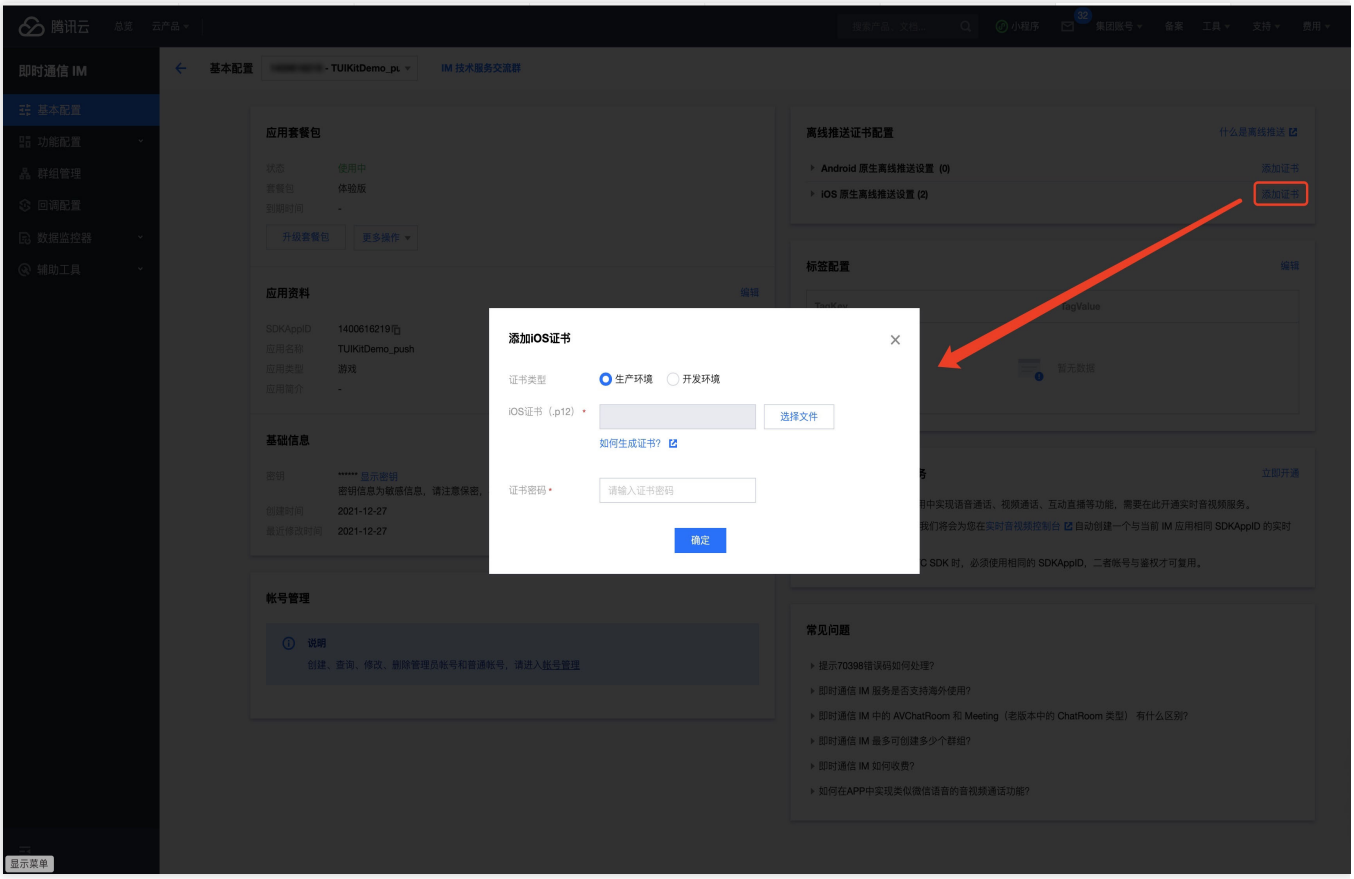
步骤2：上传 p12 证书到 Chat 控制台

1. 登录 Chat 控制台。
2. 单击目标应用卡片，进入应用的基础配置页面。



3. 单击 **iOS native offline push settings** 右侧的 **Add Certificate**。
4. 选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击 **Confirm**。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意：

上传的推送证书时需要确认下 Bundle ID 与您的主 App 的Bundle ID一致。

上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

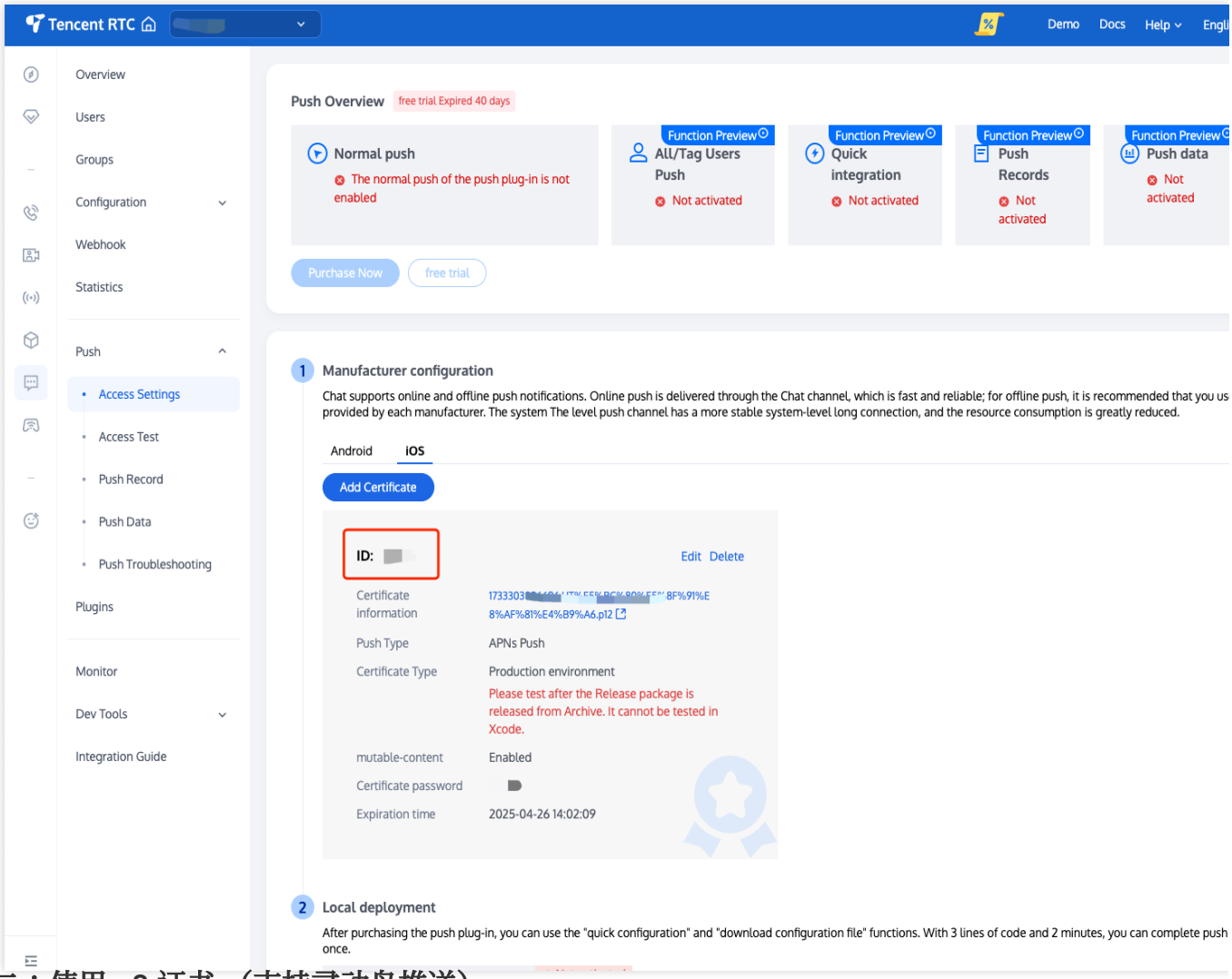
上传证书需要设置密码，无密码收不到推送。

发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。

5. 待推送证书信息生成后，记录证书的 **ID**。

©2013-2025 Tencent Cloud International Pte. Ltd.



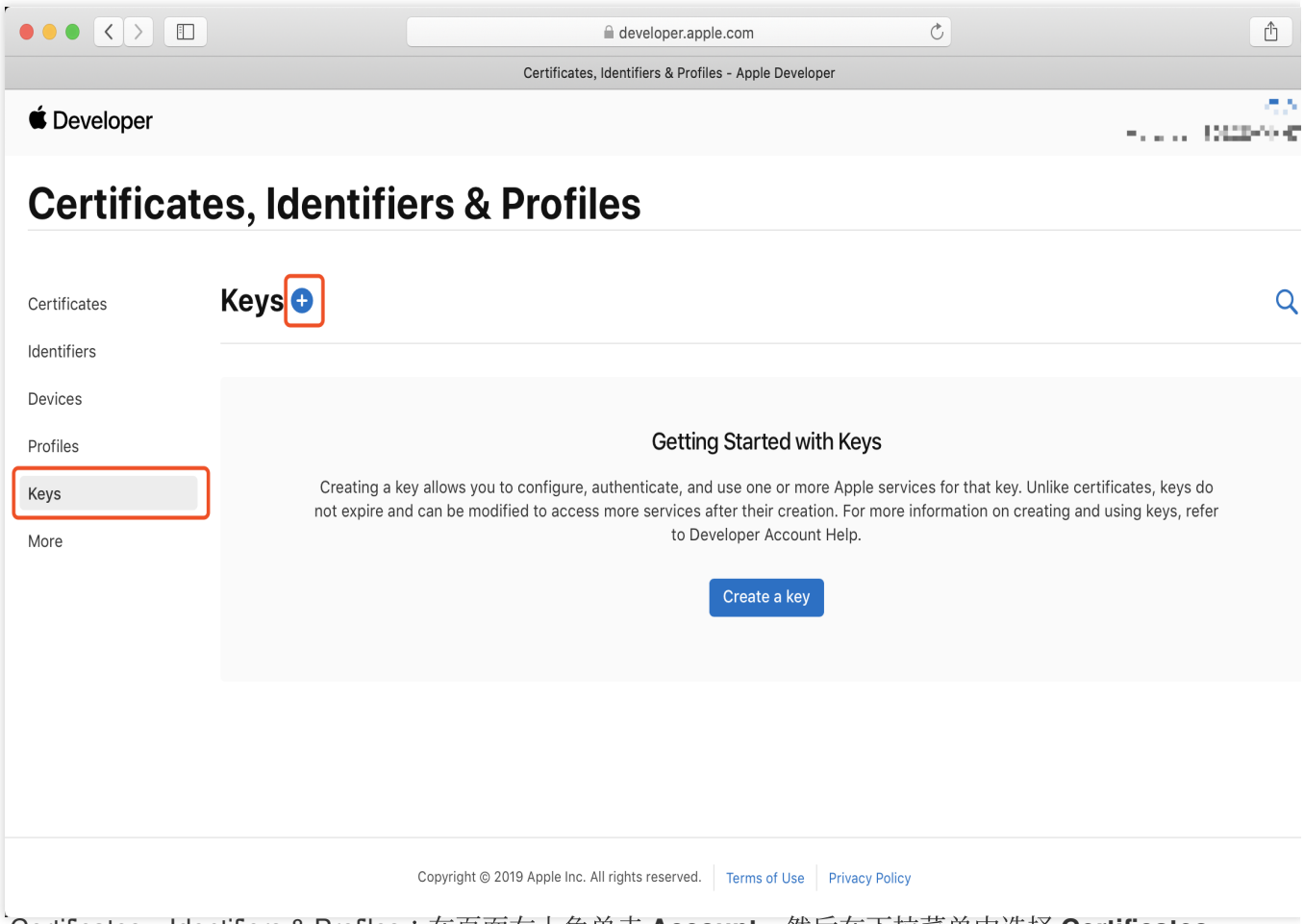
方式二：使用 p8 证书 （支持灵动岛推送）

p8 证书：p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 p8 证书可以简化证书管理，因为您可以使用一个p8 证书为多个应用程序提供推送通知服务。另外，p8 证书支持灵动岛推送。

步骤1：申请 APNs 证书

要创建 p8 证书文件，首先需要登录 苹果开发者中心。

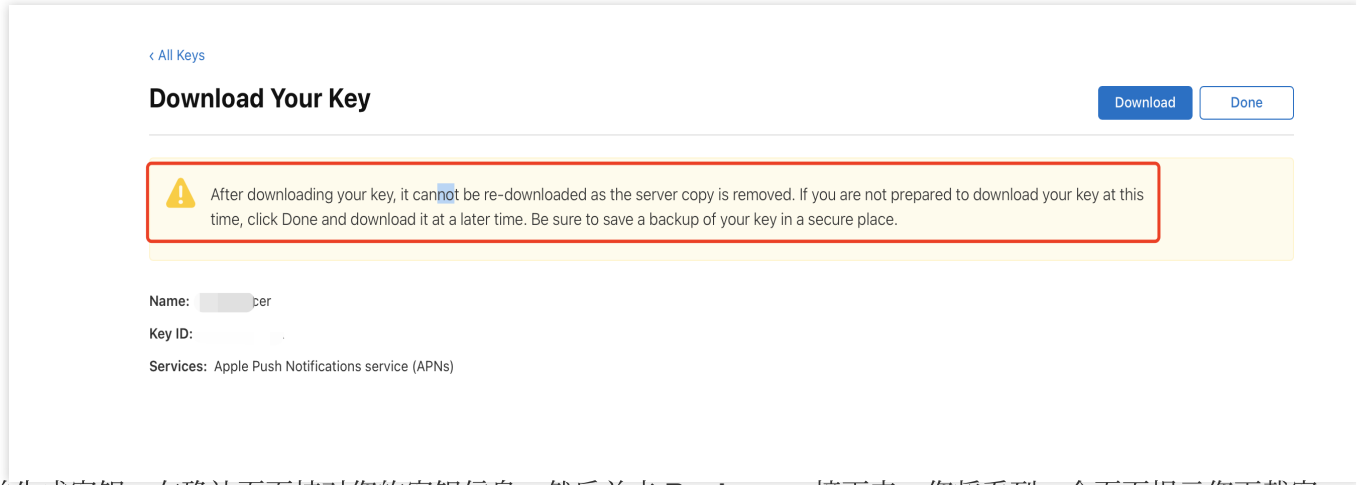
©2013-2025 Tencent Cloud International Pte. Ltd.



1. 进入Certificates，Identifiers & Profiles：在页面右上角单击 **Account**，然后在下拉菜单中选择 **Certificates, Identifiers & Profiles** 。

2. 创建一个新的 App ID：在左侧菜单中单击 **Identifiers** ，然后单击右侧的 **+** 创建一个新的 App ID。填写相应的信息并单击 **Continue** 。

3. 创建一个新的密钥：在左侧菜单中单击 **Keys**，然后单击右侧的 **+** 创建一个新的密钥。输入密钥的名称，然后勾选 **Apple Push Notifications service (APNs)**，单击 **Continue** 。



确认并生成密钥：在确认页面核对您的密钥信息，然后单击 **Register** 。接下来，您将看到一个页面提示您下载密钥。单击 **Download**，将生成的 .p8 文件保存到您的计算机上。

©2013-2025 Tencent Cloud International Pte. Ltd.

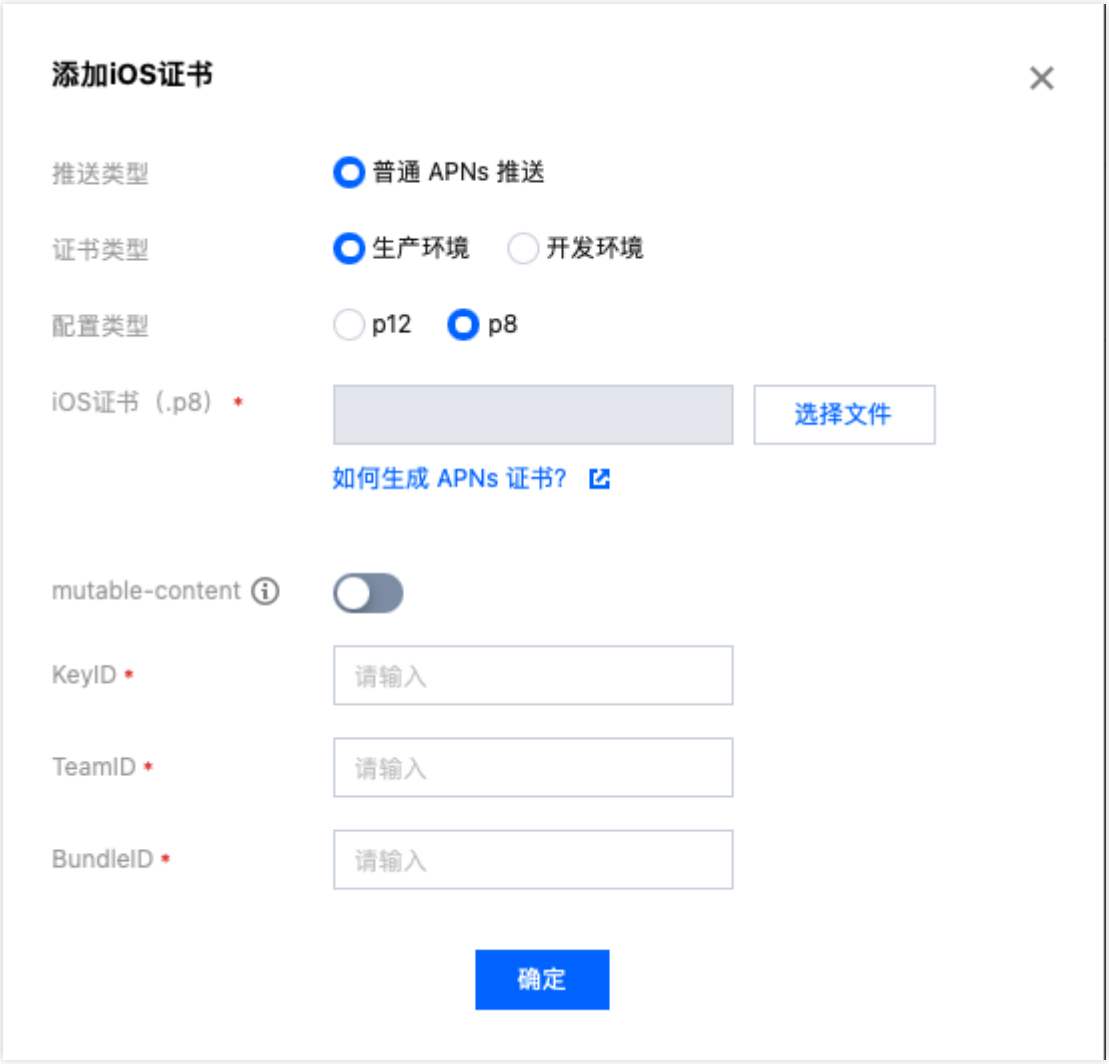
注意：

p8 证书只可以下载一次，请妥善保存。

请妥善保管下载的 p8 文件，因为您将无法再次下载该文件。您可以使用此P8证书配置您的iOS应用程序以接收推送通知。

步骤2：上传 p8 证书到 Chat 控制台

1. 登录 Chat 控制台。
2. 单击目标应用卡片，进入应用的基础配置页面。
3. 单击 **iOS native offline push settings** 右侧的 **Add Certificate**。
4. 选择 .p8 证书
5. 选择生产环境、开发环境各上传一份（您需要准备的材料一致，仅由后端为您分配 APNs 的推送环境）



说明：

KeyID：这是您的 APNs Auth Key 的唯一标识符。当您在 Apple Developer Center 创建一个新的 APNs Auth Key 时，系统会为您生成一个 Key ID。您可以在 "Certificates, Identifiers & Profiles" 部分的 "Keys" 中找到它。

TeamID：这是您的开发者账户的唯一标识符。您可以在 Apple Developer Center 的账户详情页面找到它。点击右上角的 "Membership"，在 "Membership Details" 部分可以找到您的 Team ID。

©2013-2025 Tencent Cloud International Pte. Ltd.

BundleID：这是您的应用程序的唯一标识符，也称为应用程序 ID。您可以在 Apple Developer Center 的 "Certificates, Identifiers & Profiles" 部分找到它。选择 "Identifiers"，然后在您的应用程序列表中找到对应的 Bundle ID。

二、创建描述文件

步骤 1: 创建 App ID

1. **登录到 Apple Developer Center:**
访问 Apple Developer Center 并登录您的开发者账户。
2. **创建 App ID:**
在 "Certificates, Identifiers & Profiles" 部分，选择 "Identifiers"。
点击 "+" 按钮以创建新的 App ID。
选择 "App IDs" 并点击 "Continue"。
输入您的 App ID 名称和 Bundle ID（例如 **com.yourcompany.yourapp**）。
确保选择适当的功能（如 Push Notifications、App Groups 等），然后点击 "Continue" 并确认。

步骤 2: 创建 Service Extension 的 App ID （可选）

重复上述步骤，创建一个新的 App ID，用于您的 Service Extension。
Bundle ID 通常是主应用程序的 Bundle ID 后面加上扩展的标识符，例如 com.yourcompany.yourapp.extension。

步骤 3: 创建描述文件

1. **创建描述文件:**
在 "Certificates, Identifiers & Profiles" 部分，选择 "Profiles"。
点击 "+" 按钮以创建新的描述文件。
选择 "iOS App Development" 或 "App Store"（根据您的需求），然后点击 "Continue"。
选择您刚才创建的 App ID（主应用程序的 App ID），然后点击 "Continue"。
选择您的开发证书和设备（如果是开发描述文件），然后点击 "Continue"。
输入描述文件的名称，然后点击 "Generate"。
2. **为 Service Extension 创建描述文件（可选）：**
重复上述步骤，为您的 Service Extension 创建一个新的描述文件，确保选择对应的 App ID。

步骤 4: 下载和安装描述文件

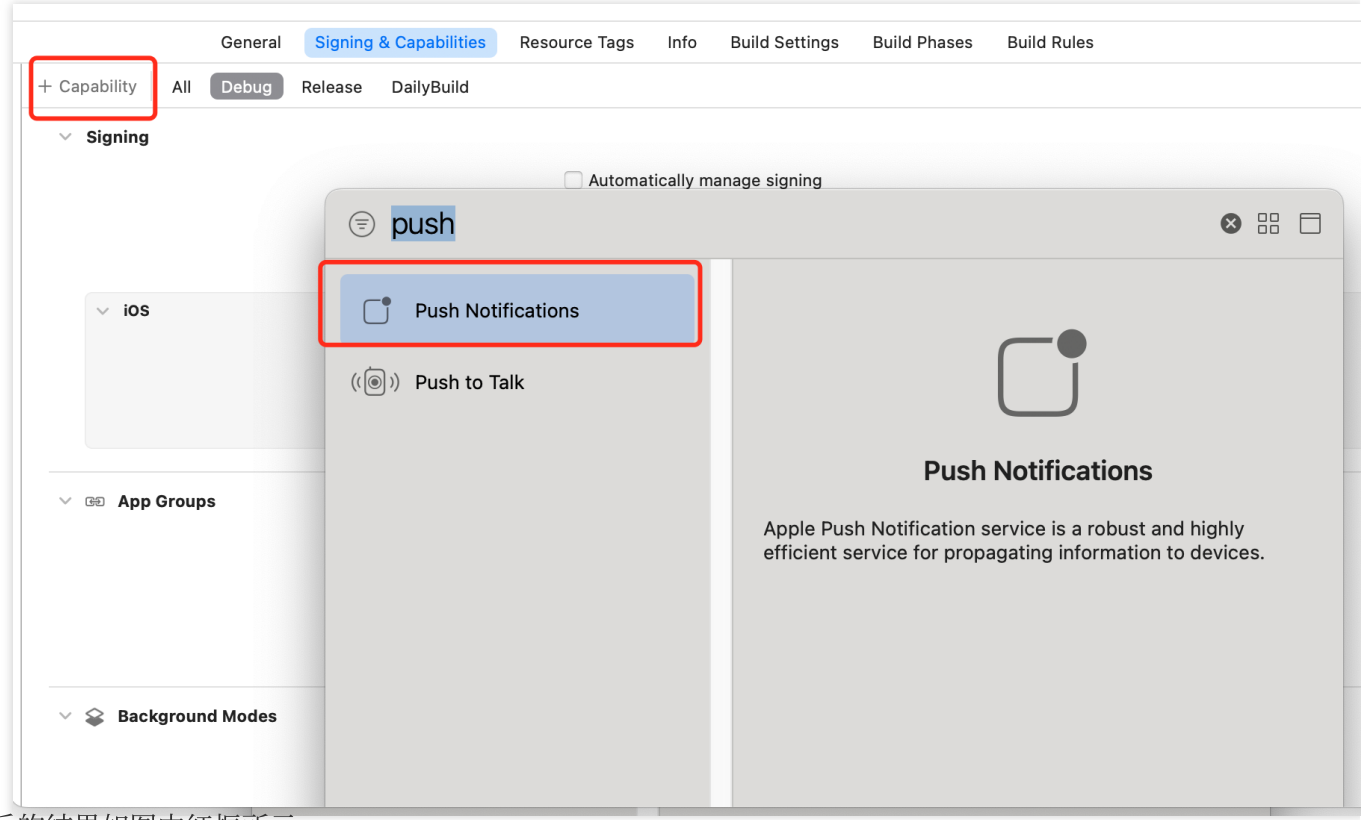
1. **下载描述文件:**
在描述文件创建完成后，点击 "Download" 按钮下载描述文件。
2. **安装描述文件:**
双击下载的描述文件，它会自动在 Xcode 中安装。

©2013-2025 Tencent Cloud International Pte. Ltd.

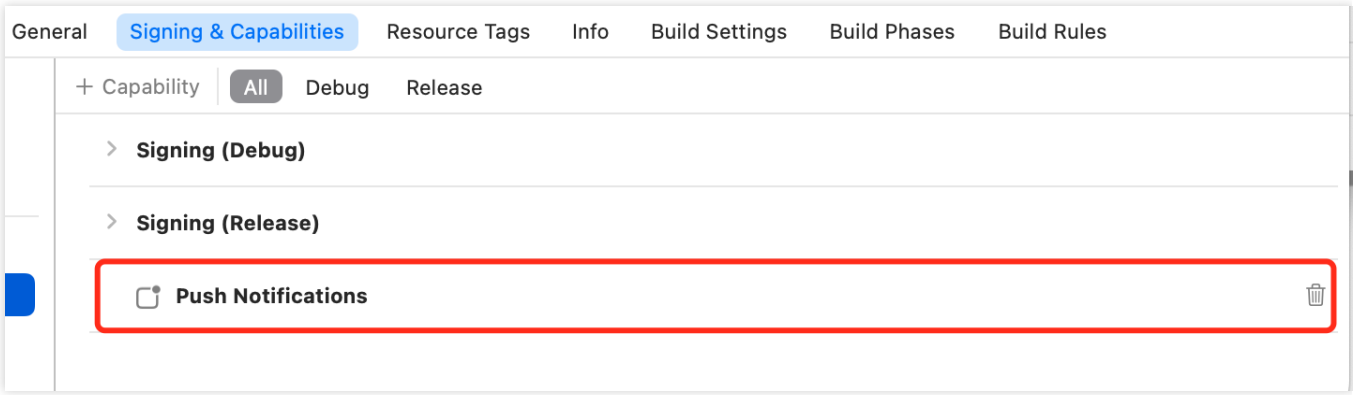
三、Xcode添加推送权限

要在App中添加推送权限，请在 Xcode 项目中启用推送通知功能。

打开 **Xcode** 项目，在 Project > Target > Capabilities 页面中点击红框中的加号按钮，然后选择并添加 **Push Notifications** 。



添加后的结果如图中红框所示。



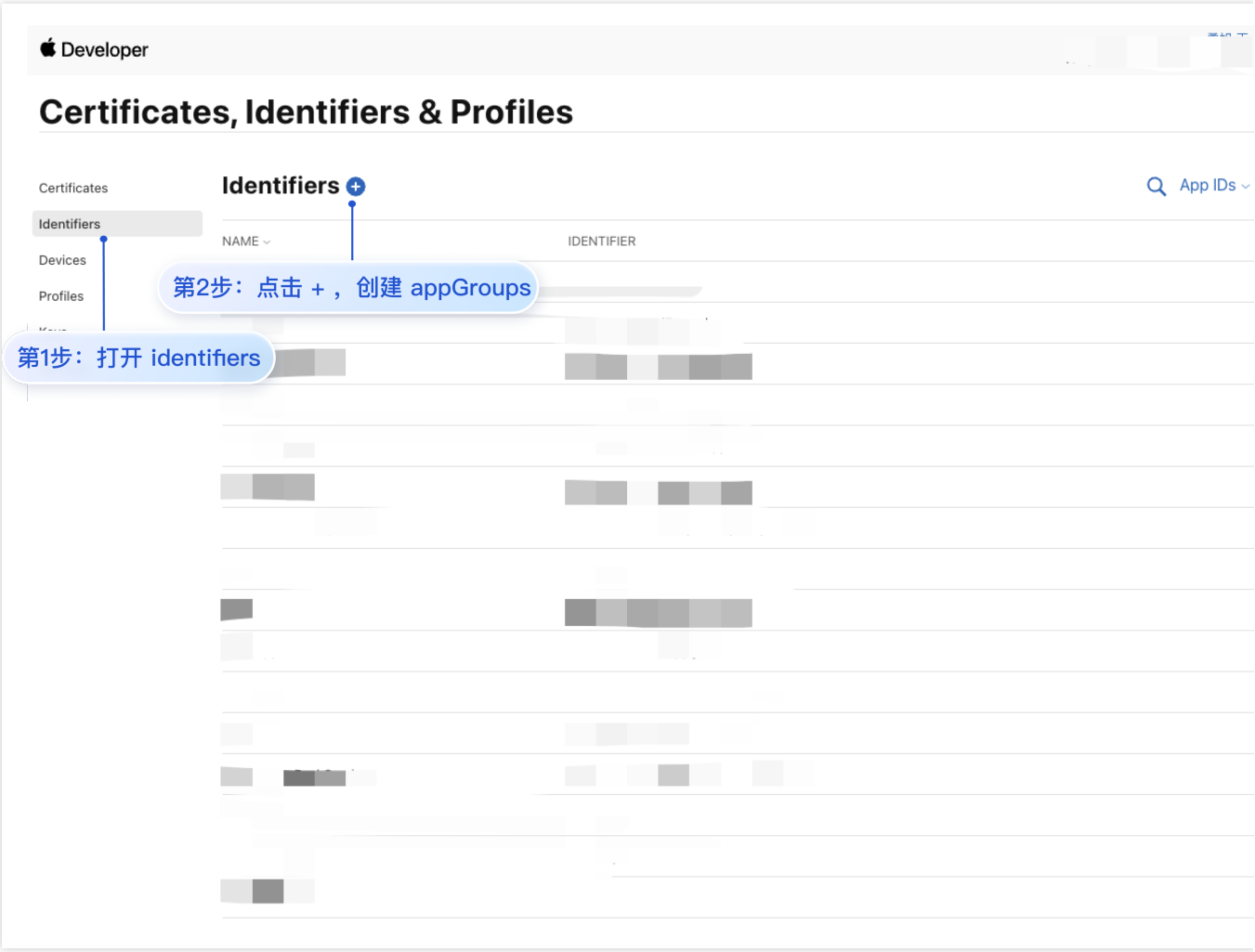
四、生成 App GroupID (可选)

当您需要使用 TIMPush 组件统计推送抵达率时，推荐您配置 TIMPushAppGroupID ，之后按照快速接入进行使用。

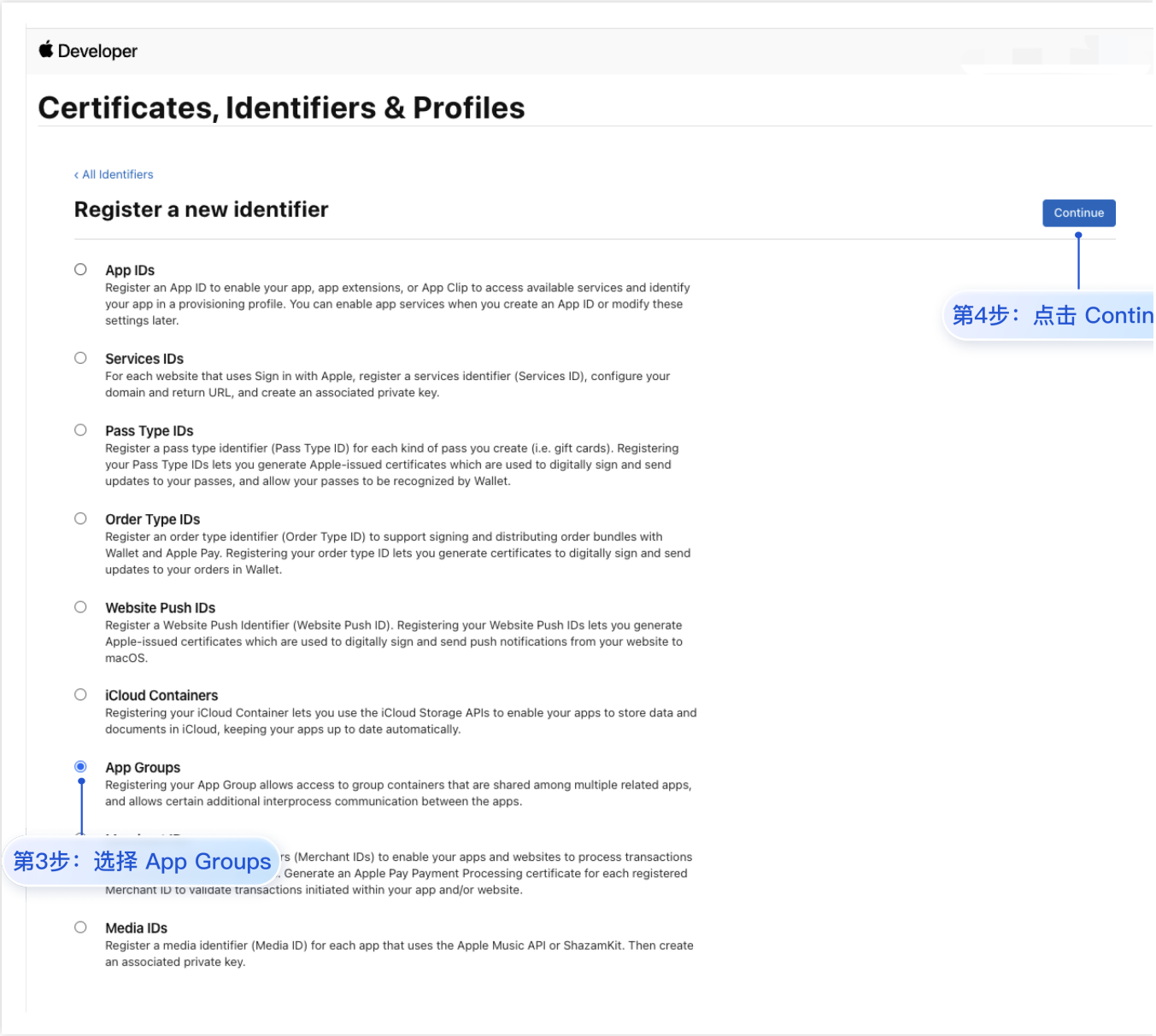
©2013-2025 Tencent Cloud International Pte. Ltd.

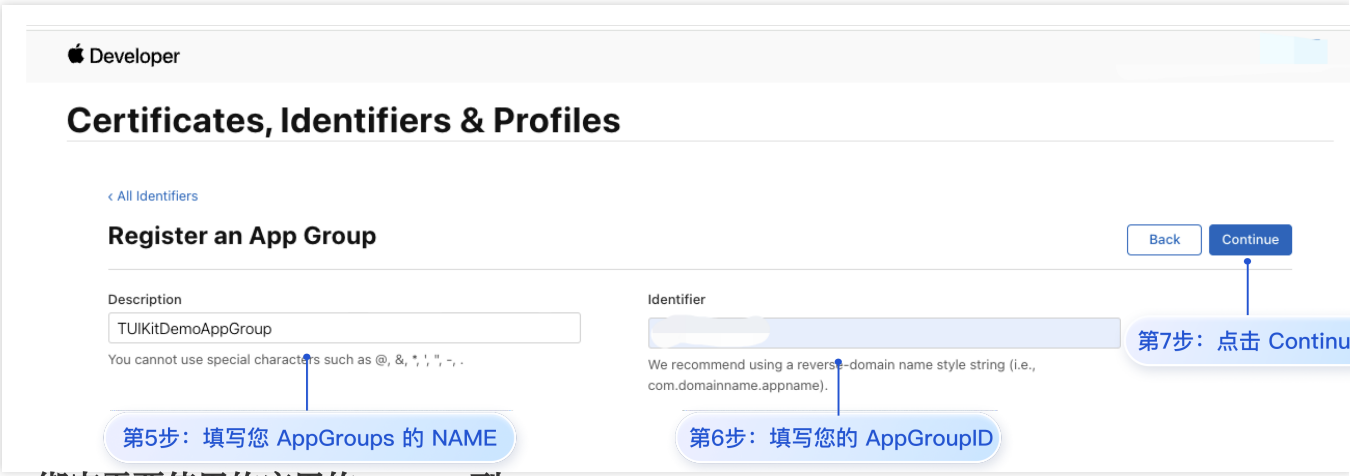
App GroupID 标识当前主 App 和 Extension 之间共享的 App Group，需要在主 App 的 Capability 中配置 App Groups 能力。

步骤1：登录 苹果开发者中心 网站，进入【identifiers】 ->【App Groups】创建 AppGroups。



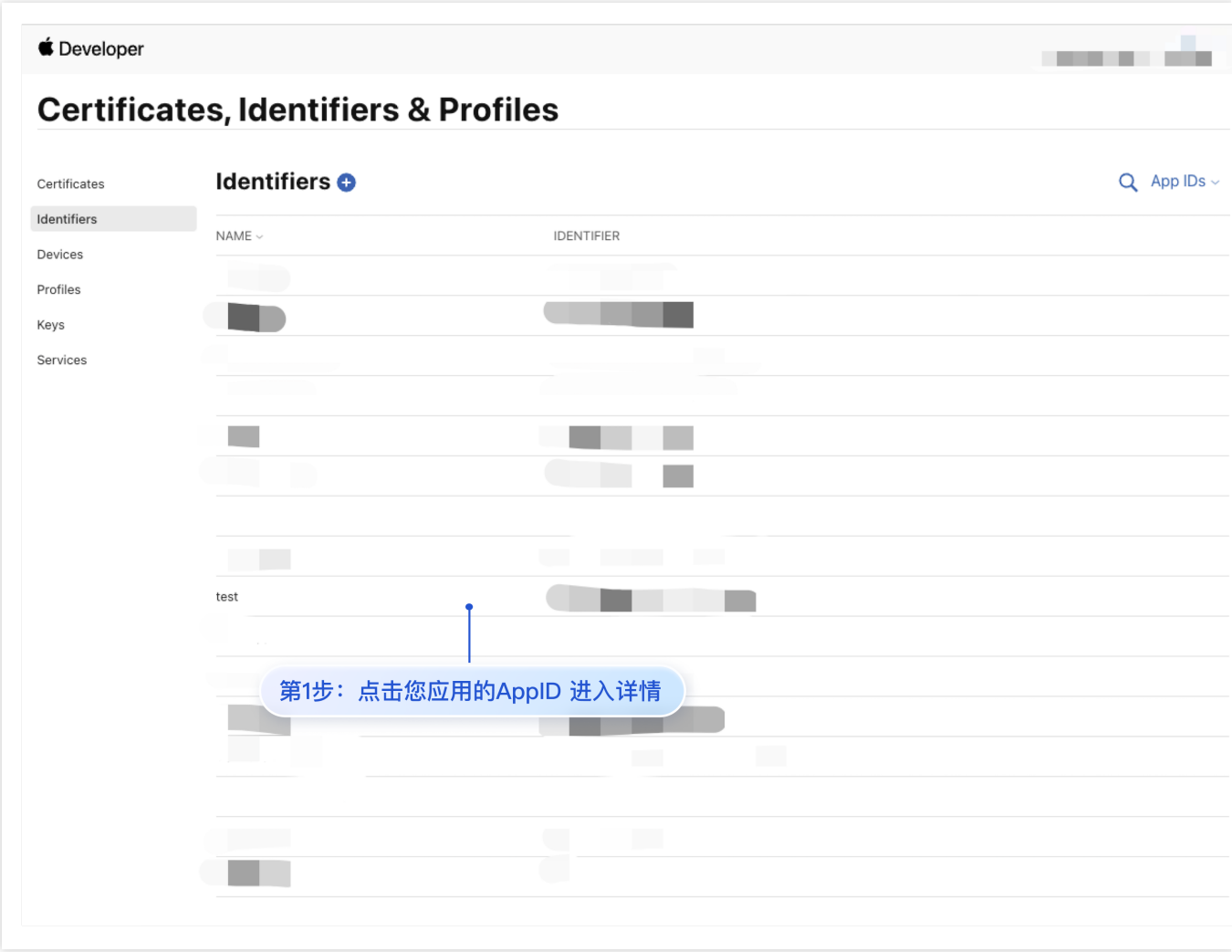
©2013-2025 Tencent Cloud International Pte. Ltd.



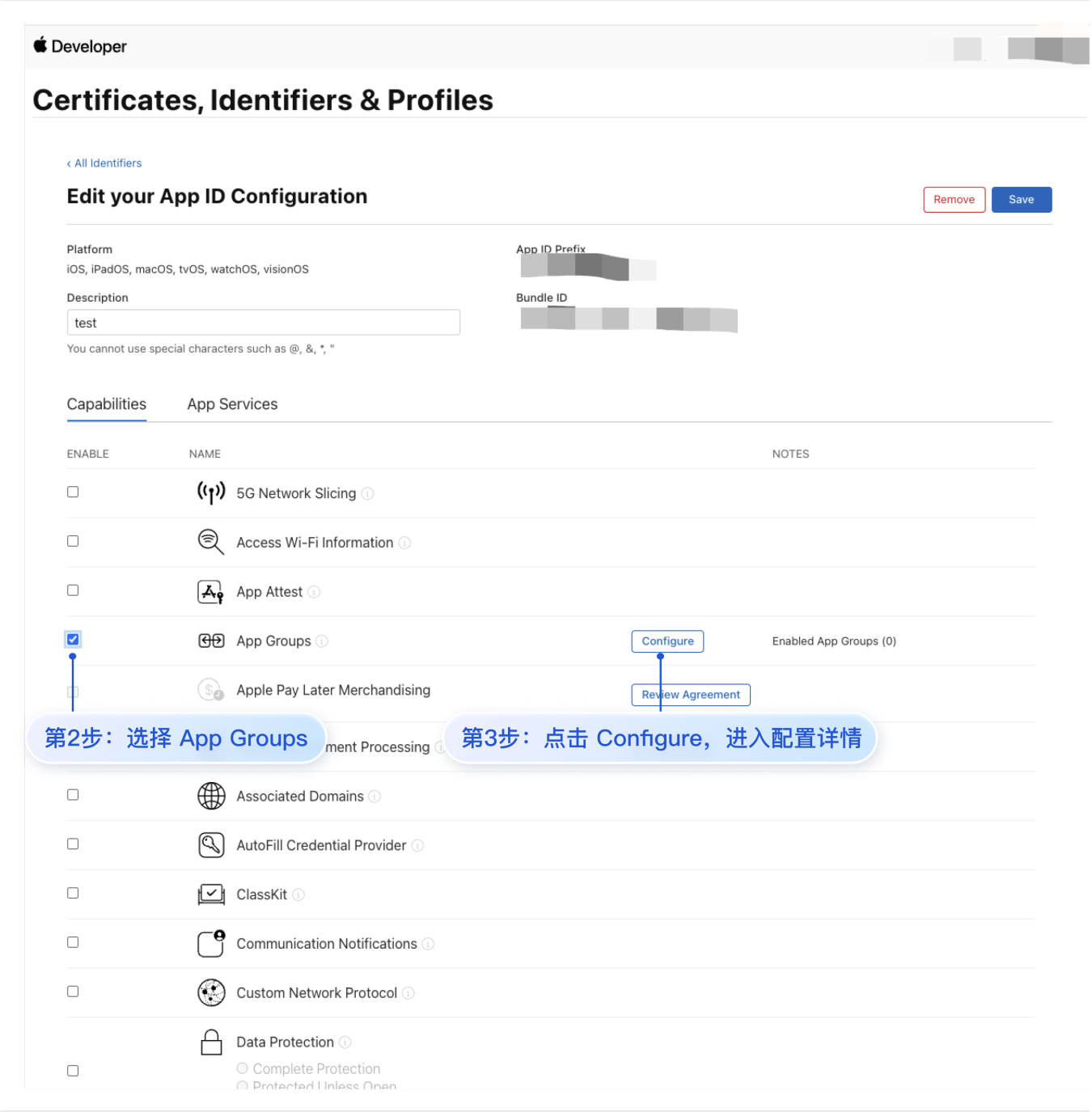


步骤2：绑定需要使用的应用的 AppID 到 AppGroups。

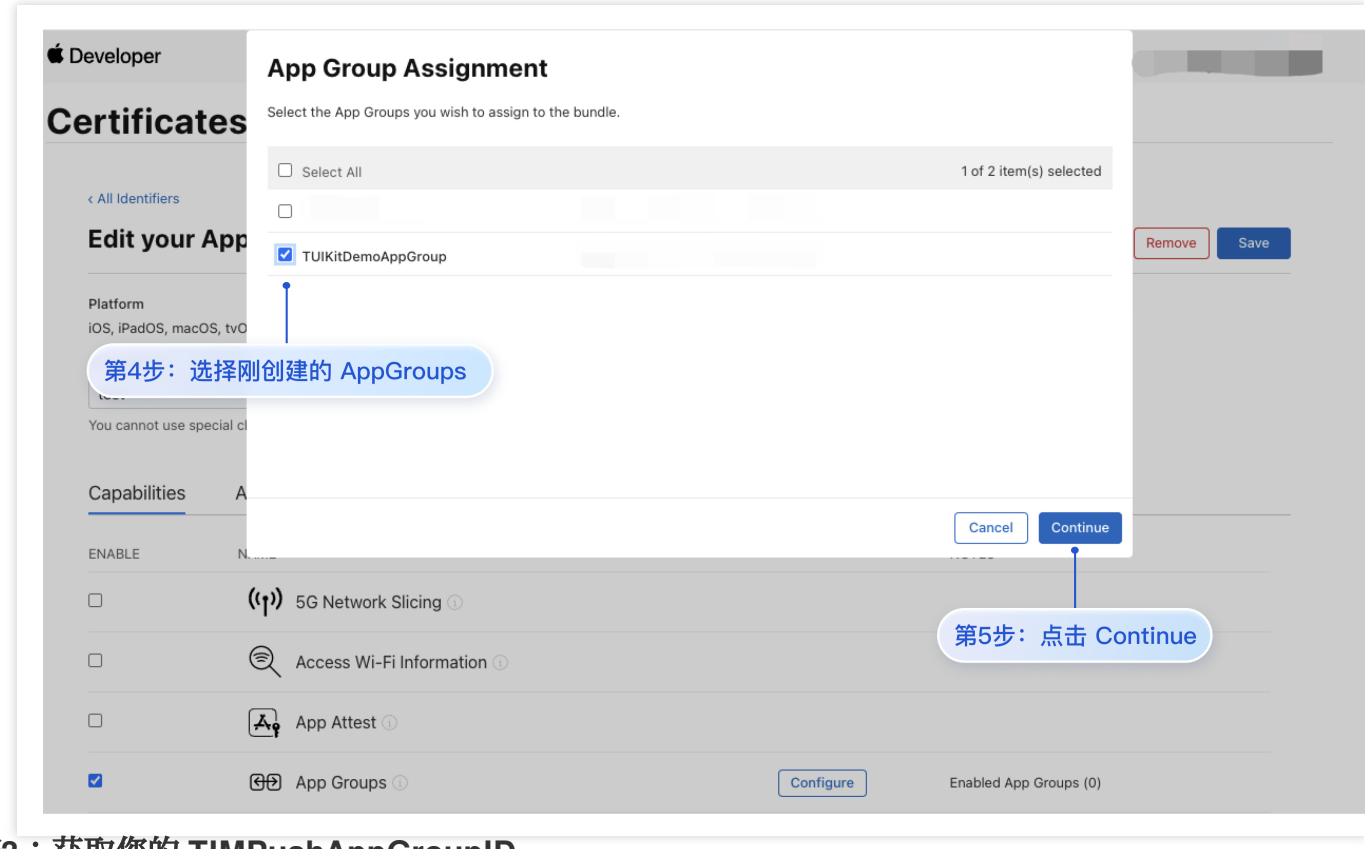
©2013-2025 Tencent Cloud International Pte. Ltd.



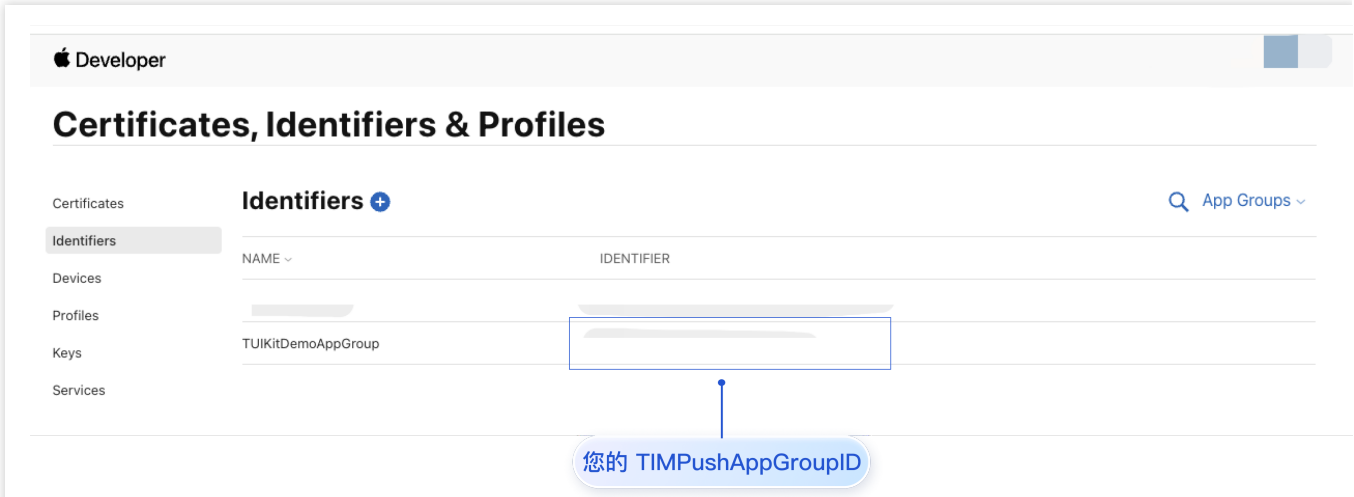
©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.



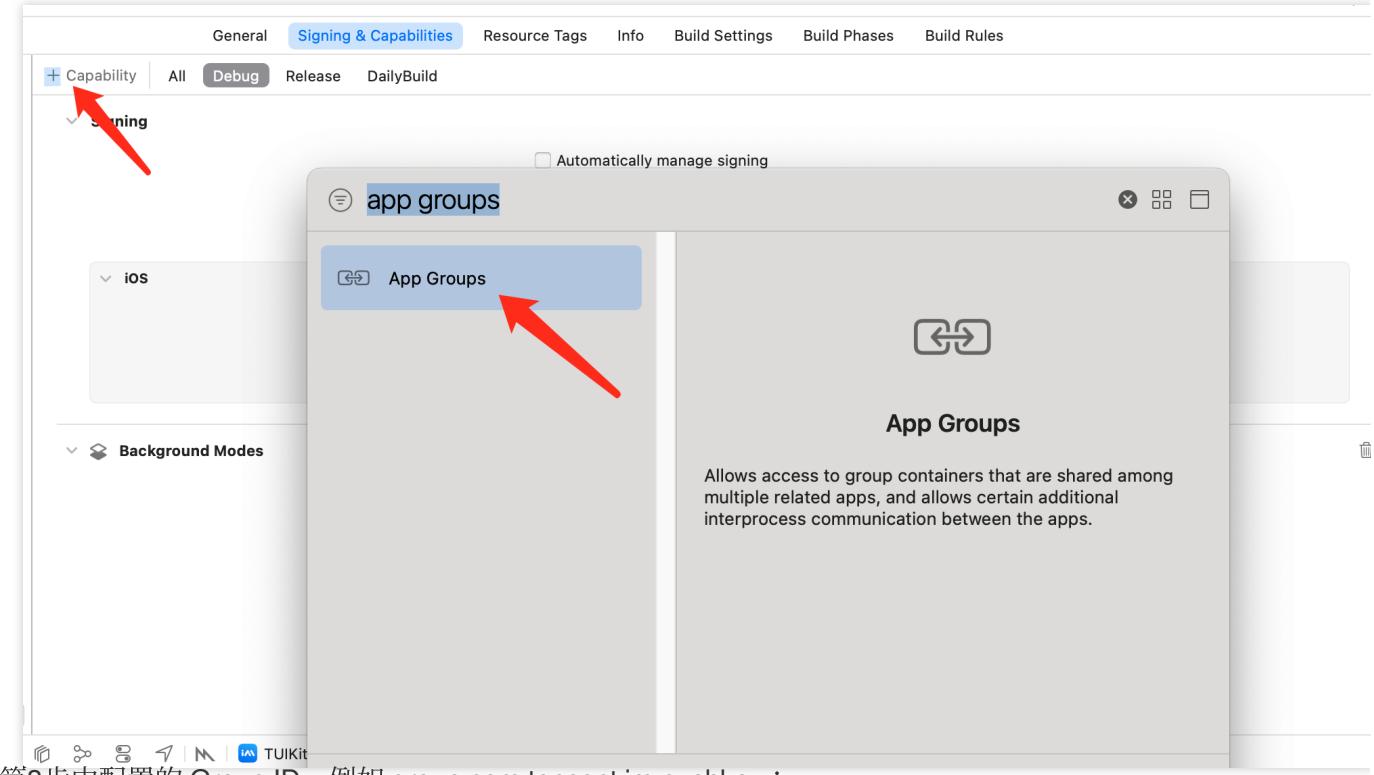
步骤3：获取您的 TIMPushAppGroupID。



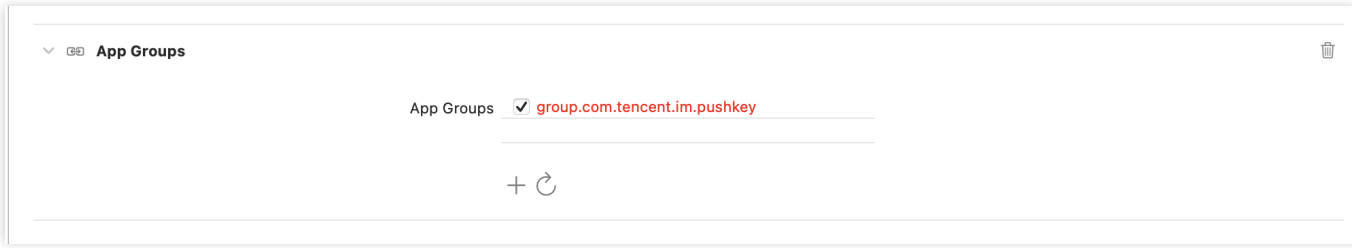
步骤4：在Xcode中配置 TIMPushAppGroupID。

打开 **Xcode** 项目，在 Project > Target > Capabilities 页面中点击红框中的加号按钮，然后选择并添加 **App Groups**。

©2013-2025 Tencent Cloud International Pte. Ltd.



填入第3步中配置的 Group ID，例如 group.com.tencent.im.pushkey：



附录: 开发者账号、证书类型与描述文件

苹果开发者账号:

创建 iOS 证书和 Profile 需使用付费的苹果账号，苹果开发者账号有三种，个人开发者、公司开发者、企业开发者，用途各有不同。

个人开发者和公司开发者是 99 美元/年，企业开发者是 299 美元/年。

企业开发者一般是大企业开发内部应用时使用，不能上架 App Store 的。

请先确保您已经有一个个人开发者或公司开发者账号。

证书介绍:

证书有多种类型，用于不同的目的。以下是 iOS 证书的分类：

证书类型	区别
开发者证书（Developer Certificate）	用于在开发阶段对应用程序进行签名和调试。开发者证书由苹果开发者中心颁发，需要使用 Xcode 或者其他开发工具进行申请和管理。

©2013-2025 Tencent Cloud International Pte. Ltd.

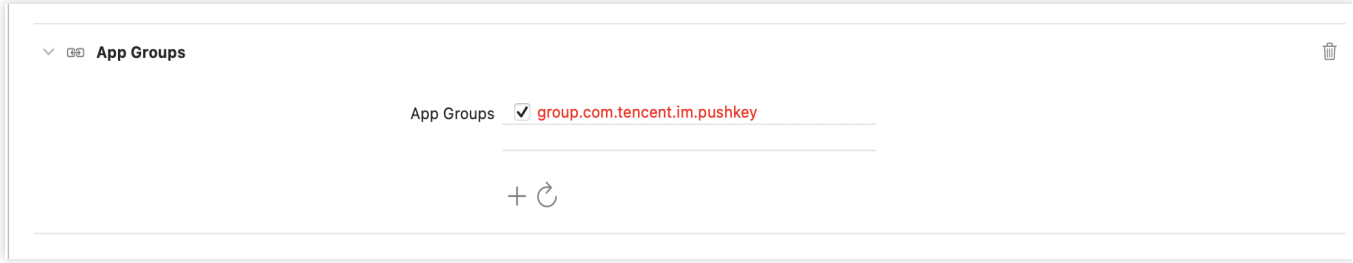
分发证书（Distribution Certificate）	用于将应用程序分发给其他用户或者上传到 App Store 进行审核。分发证书由苹果开发者中心颁发，需要使用 Xcode 或者其他开发工具进行申请和管理。
推送证书（Push Certificate）	用于实现 APNs 远程推送功能，可以让应用程序接收来自服务器的推送通知。推送证书由苹果开发者中心颁发，需要使用 Xcode 或者其他开发工具进行申请和管理。
企业证书（Enterprise Certificate）	用于将应用程序分发给企业内部员工或者客户。企业证书由苹果开发者中心颁发，需要使用 Xcode 或者其他开发工具进行申请和管理。

描述文件介绍：

描述文件（Provisioning Profiles）同样也分两种，分为开发（Development）和发布（Distribution），配置文件中包含了证书、App ID、设备（Devices），

后缀名为 .mobileprovision。它在开发者账号体系中扮演着配置和验证的角色，是真机调试和打包上架必须的文件。

描述文件（Provisioning Profile）	**作用**
开发（Development）	一个 Provisioning Profile 对应一个 App ID (Bundle ID) Provisioning Profile 决定 Xcode 用哪个证书（公钥）/私钥组合（Key Pair/Signing Identity）来签名应用程序（Signing Product），将在应用程序打包时嵌入到 .ipa 包里。 Provisioning Profile 把这些信息全部打包在一起，方便我们在调试和发布程序打包时使用。这样，只要在不同的情况下选择不同的 Provisioning Profile 文件就可以了。 Provisioning Profile 也分为 Development 和 Distribution 两类，有效期同 Certificate 一样。Development 版本的 ProvisioningProfile 用于开发调试，Distribution 版本的 ProvisioningProfile 主要用于提交 App Store 审核，其不指定开发测试的 Devices。
发布（Distribution）	



©2013-2025 Tencent Cloud International Pte. Ltd.

快速接入

最近更新时间：2025-03-24 11:24:15

如果您想尽可能简单地接入 TIMPush 组件，您需要使用 TUILogin 的 login/logout 接口进行 Chat 账号的登入/登出操作，TIMPush 组件能自动感知登入/登出事件。

步骤1：集成 TIMPush 组件

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

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

  # Pods for Example
  # 版本号 “VERSION” 请前往 更新日志 中获取配置。
  pod 'TIMPush', 'VERSION'
end
```

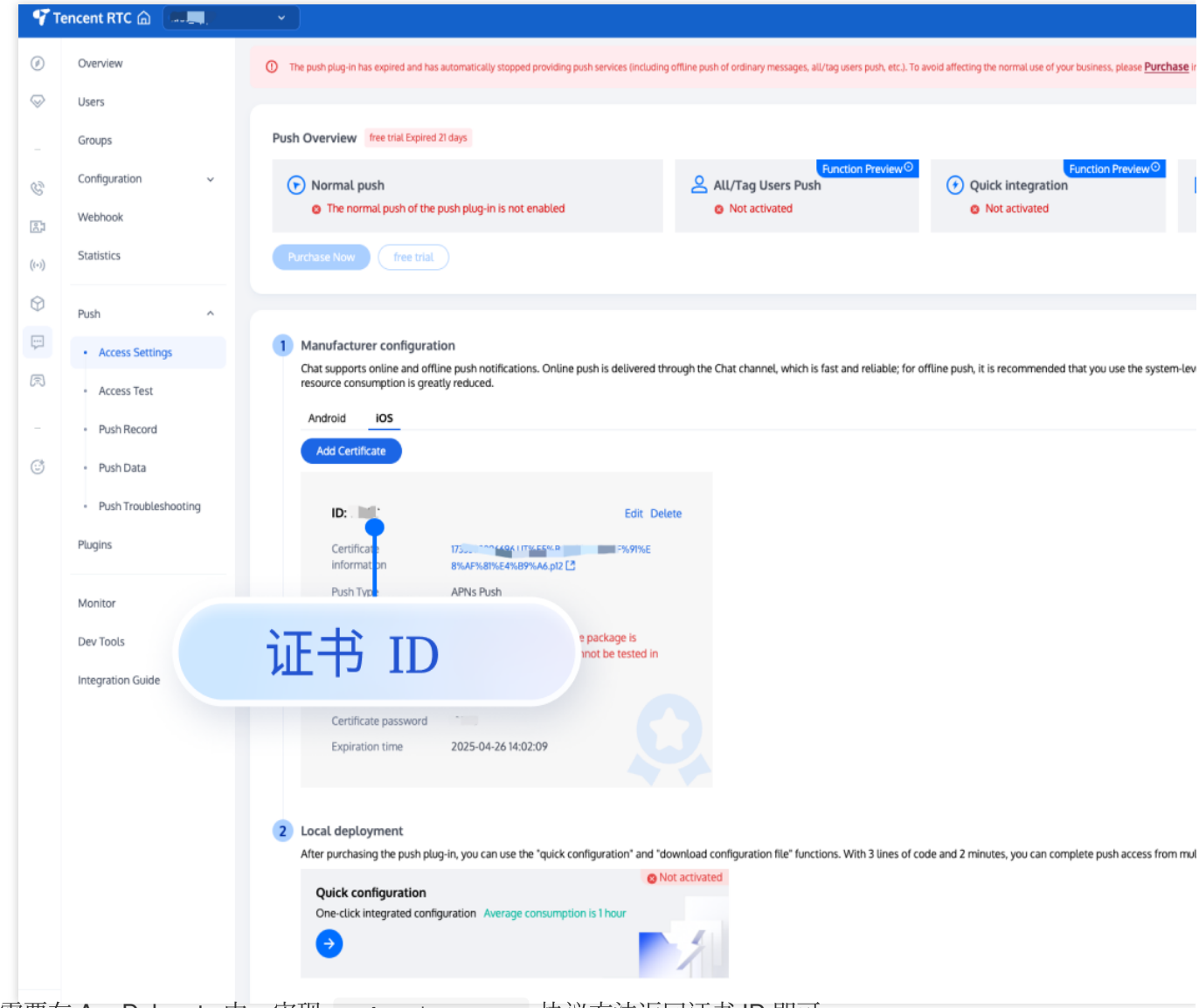
2. 执行以下命令，安装 TIMPush 组件。

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

步骤2：配置推送参数

1. 当您上传证书到 Chat 控制台后，Chat 控制台会为您分配一个证书 ID，见下图：

©2013-2025 Tencent Cloud International Pte. Ltd.



2. 您需要在 AppDelegate 中，实现 `- businessID` 协议方法返回证书 ID 即可。

Objective-C

Swift

```
#pragma mark - TIMPush
- (int)businessID {
    //上一步控制台给的证书ID，如 1234567
    int  kBusinessID = 0;
    return kBusinessID;
}

- (NSString *)applicationGroupID {
    //AppGroup ID
    return kTIMPushAppGroupKey;
}

- (BOOL)onRemoteNotificationReceived:(NSString *)notice {
    // custom navigate
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    return NO;
}
```

```
#pragma mark - TIMPush
//Swift 务必携带 @objc 关键字
@objc func businessID() -> Int32 {
    //上一步控制台给的证书ID
    return 0
}
@objc func applicationGroupID() -> String {
    //AppGroup ID
    return "group.com.yourcompony.pushkey"
}
@objc func onRemoteNotificationReceived(_ notice: String?) -> Bool {
    // custom navigate
    return false
}
```

至此，您已经完成了基本推送功能的接入。

注意：

1. 当您登录后，在控制台上看到 APNs configuration success 日志打印时，即表示已成功接入。
2. 如果您的 App 已经获取到了推送权限，此时退入后台或者停止进程，即可收到远程推送通知。
3. 如果您未接入 TUICore 组件，不需要使用 TUILogin 的登录/登出，但依然想实现离线推送，您只需：
在您的 App/Chat 登录完成后，主动调用 registerPush 方法注册推送。
退出登录时，主动调用 unRegisterPush 方法 反注册推送。
4. 如果您仅想支持免登录推送的功能，可以切换注册接口为：调用 TIMPushManager 的接口 registerPush，步骤 3 的发送消息详见 - restApi 接口。

步骤3：发消息时设置离线推送参数（含 UI 集成时 TUIChat 已默认添加，可跳过此步骤）

调用 sendMessage 发送消息时，您可以通过 V2TIMOfflinePushInfo 设置离线推送参数。调用 V2TIMOfflinePushInfo 的ext 设置自定义 ext 数据，当用户收到离线推送启动 App 的时候，可以在单击通知跳转的回调中获取到 ext 字段，然后根据 ext 字段内容跳转到指定的 UI 界面。可以参见 TUIMessageBaseDataProvider 的 sendMessage: 方法：

Objective-C

Swift

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

V2TIMOfflinePushInfo *pushInfo = [[V2TIMOfflinePushInfo alloc] init];
pushInfo.title = @"推送标题";
pushInfo.desc  = @"推送内容";
BOOL isGroup = groupID.length > 0;
NSString *senderId = isGroup ? (groupID) : ([TUILogin getUserID]);
NSString *nickName = isGroup ? (conversationData.title) : ([TUILogin getNickName] ?
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
OfflinePushExtInfo *extInfo = [[OfflinePushExtInfo alloc] init];
OfflinePushExtBusinessInfo * entity = extInfo.entity;
entity.action = 1;
entity.content = @"推送内容";
entity.sender = senderId;
entity.nickname = nickName;
entity.faceUrl = [TUILogin getFaceUrl] ?: @"";
entity.chatType = [isGroup ? @(V2TIM_GROUP) : @(V2TIM_C2C) integerValue];
entity.version = kOfflinePushVersion;
pushInfo.ext = [extInfo toReportExtString];
//以下是兼容安卓的字段，需要填写
pushInfo.AndroidOPPOChannelID = @"tuikit";
pushInfo.AndroidSound = TUIConfig.defaultConfig.enableCustomRing ? @"private_ring"
pushInfo.AndroidHuaWeiCategory = @"IM";
pushInfo.AndroidVIVOCategory = @"IM";
```

```
import TIMPush
import TUICore
import ImSDK_Plus

let pushInfo = V2TIMOfflinePushInfo()
pushInfo.title = "推送标题"
pushInfo.desc = "推送内容"

let isGroup = groupID!.count > 0
let senderId = isGroup ? groupID : TUILogin.getUserID()
let nickName = isGroup ? "conversationData Title" : (TUILogin.getNickName() ?? TUIL

let extInfo = OfflinePushExtInfo()
let entity = extInfo.entity
entity.action = 1
entity.content = "推送内容"
entity.sender = senderId ?? ""
entity.nickname = nickName ?? ""
entity.faceUrl = TUILogin.getFaceUrl() ?? ""
entity.chatType = isGroup ? Int(V2TIMConversationType.GROUP.rawValue) : Int(V2TIMCo
entity.version = 1
pushInfo.ext = extInfo.toReportExtString()

// 以下是兼容安卓的字段，需要填写
pushInfo.androidOPPOChannelID = "tuikit"
pushInfo.androidSound = TUIConfig.default().enableCustomRing ? "private_ring" : nil
pushInfo.androidHuaWeiCategory = "IM"
pushInfo.androidVIVOCategory = "IM"
```

©2013-2025 Tencent Cloud International Pte. Ltd.

步骤4: 点击离线推送后自定义跳转

如果您需要自定义解析收到的远程推送，您可按照如下方法实现：

自定义点击跳转实现

自定义点击跳转实现（旧方案）

注意：

注册回调时机建议放在应用 AppDelegate 的 didFinishLaunchingWithOptions 函数中。

Objective-C

Swift

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    [TIMPushManager addPushListener:self];
    return YES;
}
#pragma mark - TIMPushListener
- (void)onNotificationClicked:(NSString *)ext {
        // 获取 ext 自定义跳转
}
```

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
    // Override point for customization after application launch.
    TIMPushManager.addPushListener(listener: self)
    return true
}

@objc func onNotificationClicked(_ ext: String) {
    //Clicked
}
@objc func onRecvPushMessage(_ message: TIMPushMessage) {
    //onRecvPushMessage
}
@objc func onRevokePushMessage(_ messageID: String) {
    //onRevokePushMessage
}
```

1. 如果您需要自定义解析收到的远程推送，您需要在 AppDelegate.m 文件中遵守 TIMPushDelegate 协议并实现 -onRemoteNotificationReceived 方法。

Objective-C

Swift

```
#pragma mark - TIMPush
- (BOOL)onRemoteNotificationReceived:(NSString *)notice {
    //- 如果返回 YES， TIMPush 将不在执行内置的 TUIKit 离线推送解析逻辑，完全交由您自行处理；
    //NSString *ext = notice;
    //OfflinePushExtInfo *info = [OfflinePushExtInfo createWithExtString:ext];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    //return YES;

    //- 如果返回 NO，TIMPush 将继续执行内置的 TUIKit 离线推送解析逻辑，继续回调 - navigateTo
    return NO;
}
```

```
@objc func onRemoteNotificationReceived(_ notice: String) -> Bool {
    // - 如果返回 true， TIMPush 将不在执行内置的 TUIKit 离线推送解析逻辑，完全交由您自行处理
    // let ext = notice
    // let info = OfflinePushExtInfo.create(withExtString: ext)
    // return true

    // - 如果返回 false，TIMPush 将继续执行内置的 TUIKit 离线推送解析逻辑，继续回调 - naviga
    return false
}
```

2. 如果您想使用内置的 TUIChat 推送解析逻辑，并跳转到 TUIChat 的聊天页面，您可以实现 `-navigateToBuiltInChatViewController` 方法。

Objective-C

Swift

```
#pragma mark - TIMPush

- (void)navigateToBuiltInChatViewController:(NSString *)userID groupID:(NSString *)

    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")]
        && ![vc isKindOfClass:NSClassFromString(@"ConversationController_Minimalist
        return;
    }
    if ([vc respondsToSelector:NSSelectorFromString(@"pushToChatViewController:user
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        [vc performSelector:NSSelectorFromString(@"pushToChatViewController:userID:
    }
```

```
@objc func navigateToBuiltInChatViewController(userID: String?, groupID: String?) {
    // 在这里实现导航到内置聊天视图控制器的逻辑
    print(userID,groupID)
}
```

步骤5: 统计推送抵达率

1. 如果您需要统计推送的抵达和点击数据，您需要在 AppDelegate.m 文件中实现 `- applicationGroupID` 方法，返回 App Group ID （生成方式可参见 厂商配置-生成 App GroupID）。
2. 在 Notification Service Extension 的 `-didReceiveNotificationRequest:withContentHandler:` 方法中调用推送抵达率统计函数:

Objective-C

Swift

```
@implementation NotificationService
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentH
    //appGroup 标识当前主 APP 和 Extension 之间共享的 APP Group，需要在主 APP 的 Capabili
    //格式为 group + [主bundleID]+ key
    //如 group.com.tencent.im.pushkey
    NSString * appGroupID = kTIMPushAppGroupKey;
    __weak typeof(self) weakSelf = self;
    [TIMPushManager handleNotificationServiceRequest:request appGroupID:appGroupID
        weakSelf.bestAttemptContent = [content mutableCopy];
        // Modify the notification content here...
        // self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [modifie
        weakSelf.contentHandler(weakSelf.bestAttemptContent);
    }];
}
@end
```

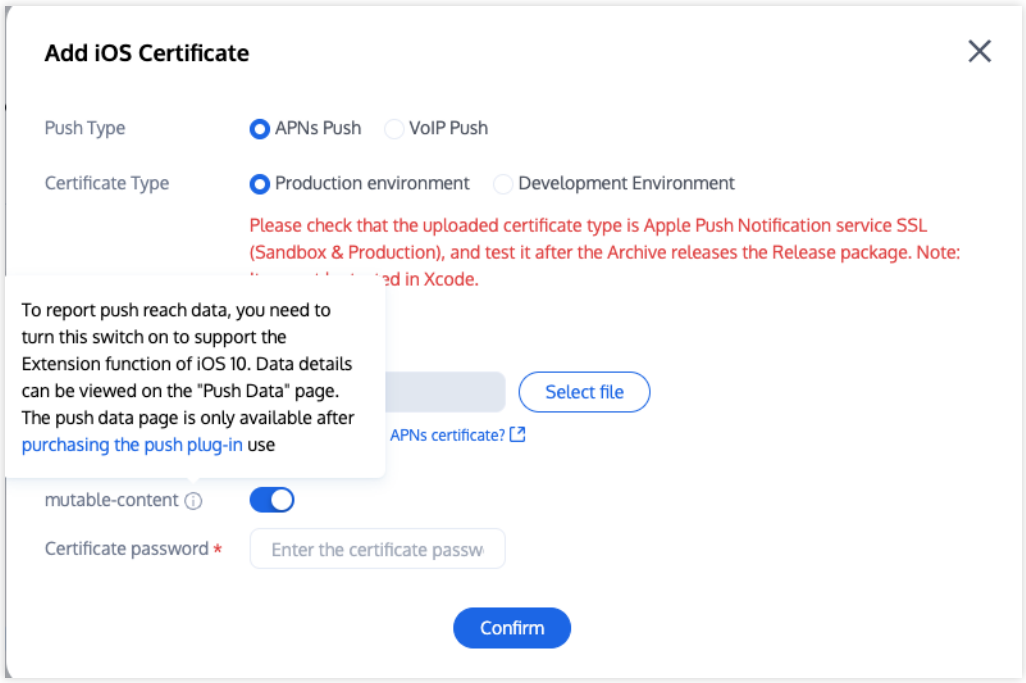
```
override func didReceive(_ request: UNNotificationRequest, withContentHandler conte
    self.contentHandler = contentHandler
    bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationCo
    //appGroup 标识当前主 APP 和 Extension 之间共享的 APP Group，需要在主 APP 的 Capabili
    //推荐格式为 group + [主bundleID]
    //如 group.com.主bundleID.pushkey
    TIMPushManager.handleNotificationServiceRequest(request: request, appGroupID: "
        [weak self] content  in
        if let bestAttemptContent = self?.bestAttemptContent {
            // Modify the notification content here...
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            bestAttemptContent.title = "\\(bestAttemptContent.title) [modified]"
            contentHandler(bestAttemptContent)
        }
    }
}
```

注意：

上报推送触达数据，需要开启 mutable-content 开关来支持 iOS 10 的 Extension 功能。



数据详情可在推送数据页面查看，推送数据页面仅限 购买推送插件 后使用。

关于合规

TIMPush 在您未主动调用 registerPush 之前，不会有其他任何操作，符合相关规定。

如果您使用了 TUILogin 的登录登出，TIMPush 会在内部自动调用 registerPush 或 unRegisterPush。

恭喜您已经完成了推送插件的接入，需要提醒您：消息推送插件**试用或购买到期后，将自动停止提供推送服务（包括普通消息离线推送、全员推送等服务）**。为避免影响您业务正常使用，请提前购买/续费。

关于全员/标签推送

全员/标签推送支持发送特定内容，还可根据标签、属性，针对特定用户群体发送个性化内容，例如会员活动、区域通知等，助力拉新、转化、促活等各个阶段运营工作的有效进行，同时支持推送送达报表，自助推送故障排查工具，具体效果请参见 效果展示。

更多详细内容建议查阅全员/标签推送

恭喜您已经完成了推送插件的接入，需要提醒您：消息推送插件**试用或购买到期后，将自动停止提供推送服务（包括普通消息离线推送、全员推送等服务）**。为避免影响您业务正常使用，请提前购买/续费。

©2013-2025 Tencent Cloud International Pte. Ltd.

说明：

接入完成收不到推送，请先自助使用 排查工具 查看下具体原因，也需要关注 厂商消息分类机制。

推送指标数据查看，请使用 数据统计 查询。

全员/标签推送功能请参见：REST API 接口 - 发起全员/标签推送。

©2013-2025 Tencent Cloud International Pte. Ltd.

客户端 API

最近更新时间：2025-03-24 11:24:48

TIMPush - TIMPushManager

@interface TIMPushManager : NSObject ：推送插件接口类。

接口概览

注册/反注册推送服务接口

API	描述
registerPush	注册推送服务 (必须在 App 用户同意了隐私政策后，再调用该接口使用推送服务)。
unRegisterPush	反注册关闭推送服务。
setRegistrationID	RegistrationID 是推送接收设备的唯一标识 ID。默认情况下，注册推送服务成功时自动生成该 ID，同时也支持您自定义设置。您可根据 RegistrationID 向指定设备推送消息。需要注意的是，卸载并重新安装设备会更改 RegistrationID，因此需要在注册推送服务之前调用 setRegistrationID 接口。
getRegistrationID	在成功注册推送服务后，可通过调用 getRegistrationID 接口获取推送接收设备的唯一标识 ID，即RegistrationID。您可根据 RegistrationID 向指定设备推送消息。

Push 全局监听接口

API	描述
addPushListener	添加 Push 监听器。
removePushListener	移除 Push 监听器。

自定义配置接口

设置应用前台是否展示推送

API	描述
disablePostNotificationInForeground	关闭 App 在前台时弹出通知栏。

©2013-2025 Tencent Cloud International Pte. Ltd.

统计 TIMPush 的推送抵达率

如果您需要统计推送的抵达和点击数据，您需要在 Notification Service Extension 中主动调用本函数。

API	描述
handleNotificationServiceRequest:appGroupID:callback:	仅支持在 Notification Service Extension 的 '-didReceiveNotificationRequest:withContentHandler:' 方法中调用。 appGroup 标识当前主 App 和 Extension 之间共享的 App Group，需要在主 App 的 Capability 中配置 App Groups 能力。

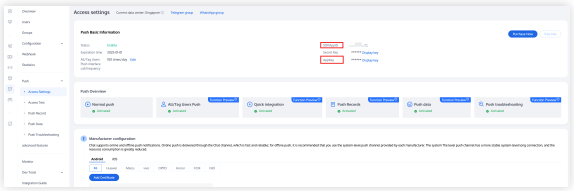
接口详情

函数说明

+ (void)registerPush:(int) sdkAppId appKey:(NSString *) appKey succ:(TIMPushSuccessCallback) successCallback fail:(TIMPushFailedCallback) failedCallback;

注册推送服务，请正确传递 sdkAppId 和 appKey 两个参数，即可注册推送服务。

参数说明：

参数	描述	获取路径
sdkAppId	Chat 控制台为您分配的应用 ID。	
appKey	Chat 控制台为您分配的客户端密钥。	

代码示例：

```
const int sdkAppId = 您的 sdkAppId;
static const NSString *appKey = @"客户端密钥";

[TIMPushManager registerPush:sdkAppId appKey:appKey succ:^(NSData * _Nonnull devic
    //
 } fail:^(int code, NSString * _Nonnull desc) {
   //error
}];
```

+ (void)unRegisterPush:(TIMPushCallback) successCallback fail:(TIMPushFailedCallback) failedCallback;

©2013-2025 Tencent Cloud International Pte. Ltd.

反注册关闭推送服务

代码示例：

```
[TIMPushManager unRegisterPush:^{
     //success
   } fail:^(int code, NSString * _Nonnull desc) {
     //error
 }];
```

+ (void)setRegistrationID:(NSString *)registrationID callback: (TIMPushCallback) callback;

设置注册推送服务使用的推送 ID 标识, 即 RegistrationID，需要在注册推送服务之前调用。

参数说明：

参数	描述
registrationID	设备的推送标识 ID，卸载重装会改变。

+ (void)getRegistrationID:(TIMPushValueCallback) callback;

注册推送服务成功后，获取推送 ID 标识, 即 RegistrationID 。

+ (void)addPushListener:(id<TIMPushListener>)listener

添加 Push 监听器。

+ (void)removePushListener:(id<TIMPushListener>)listener

移除 Push 监听器。

+ (void)disablePostNotificationInForeground:(BOOL)disable;

关闭 App 在前台时弹出通知栏。推送 SDK 收到在线推送时，会自动向通知栏增加 Notification 提示，如果您想自己处理在线推送消息，可以调用该接口关闭自动弹通知栏提示的特性。

参数说明：

参数	描述
disable	true:关闭 false:开启

+ (void)handleNotificationServiceRequest:(UNNotificationRequest *)request appGroupID:(NSString *)appGroupID callback: (TIMPushNotificationExtensionCallback)callback

统计 TIMPush 的推送抵达率

1. 您需要在 AppDelegate.m 文件中实现 `- applicationGroupID` 方法，返回 App Group ID。

2. 并在 Notification Service Extension 的 '- didReceiveNotificationRequest:withContentHandler:' 方法中调用本函数。

注意：

©2013-2025 Tencent Cloud International Pte. Ltd.

appGroup 标识当前主 App 和 Extension 之间共享的 App Group，需要在主 App的 Capability 中配置 App Groups 能力。

参数说明：

request	UNNotificationServiceExtension 回调携带的参数
appGroupID	appGroup 标识当前主 App和 Extension 之间共享的 App Group，需要在主 App 的 Capability 中配置 App Groups 能力。
callback	typedef void(^TIMPushNotificationExtensionCallback)(UNNotificationContent *content) 统计函数 Callback，携带 content 信息。

示例代码：

```
@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentH
    //appGroup 标识当前主 App 和 Extension 之间共享的 App Group，需要在主 App 的 Capabili
    //格式为group + [主bundleID]+ key
    //如group.com.tencent.im.pushkey
    NSString * appGroupID = kTIMPushAppGorupKey;
    __weak typeof(self) weakSelf = self;
    [TIMPushManager handleNotificationServiceRequest:request appGroupID:appGroupID
        weakSelf.bestAttemptContent = [content mutableCopy];
        // Modify the notification content here...
        // self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [modifie
        weakSelf.contentHandler(weakSelf.bestAttemptContent);
    }];
}

@end
```

TIMPush - TIMPushListener

@protocol TIMPushListener <NSObject> ：Push 监听器协议

接口概览

API	描述
onRecvPushMessage	收到 Push 消息。
onRevokePushMessage	收到 Push 消息撤回的通知。

©2013-2025 Tencent Cloud International Pte. Ltd.

onNotificationClicked	点击通知栏消息回调

接口详情

成员函数说明

- (void)onRecvPushMessage:(TIMPushMessage *)message;

收到 Push 消息，message 消息。

- (void)onRevokePushMessage:(NSString *)messageID;

收到 Push 消息撤回的通知，messageID 消息唯一标识。

- (void)onNotificationClicked:(NSString *)ext;

点击通知栏消息回调。

©2013-2025 Tencent Cloud International Pte. Ltd.

uni-app
厂商配置

最近更新时间：2025-03-03 14:28:47

uni-app 腾讯云推送服务（Push） 目前推送支持小米、华为、荣耀、OPPO、vivo、魅族、APNs、一加、realme、iQOO 、FCM 和 苹果等厂商通道。

Android

iOS

注册应用到厂商推送平台

推送需要将您自己的应用注册到各个厂商的推送平台，得到 AppID 和 AppKey 等参数，来实现推送功能。目前国内支持的手机厂商有：小米、华为、荣耀、OPPO、vivo、魅族，境外支持 Google FCM。

小米

华为

OPPO

vivo

魅族

荣耀

Google FCM

注意：

通知栏推送：应用需在小米软件商店上架。

步骤1：注册小米开发者账号

进入 小米开放平台，注册小米开发者账号，详情请参见 企业开发者账号注册流程。

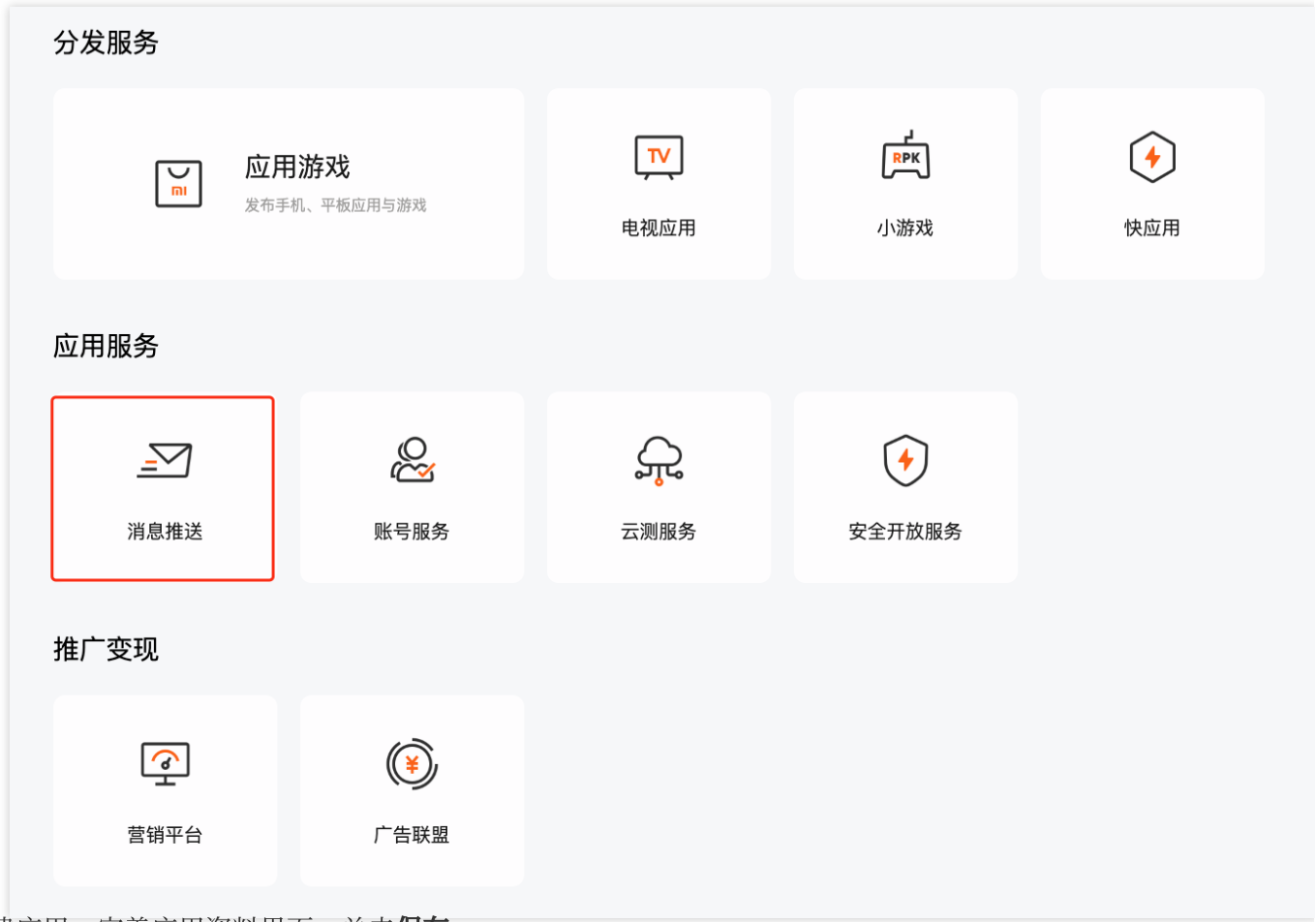
步骤2：创建应用

1. 在管理控制台单击**消息推送**。

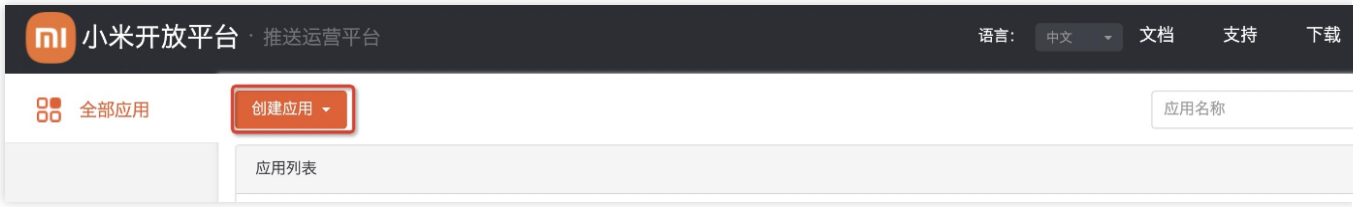
注意：

若使用个人账号登录，会显示“抱歉，您当前没有推送/审核权限”。

©2013-2025 Tencent Cloud International Pte. Ltd.



2. 创建应用，完善应用资料界面，单击**保存**。



注意：

应用包名与插件应用包名保持一致。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤3：启用推送

进入推送运营平台的**应用列表**页面，在对应的应用名称单击**启用推送**，确定启用。

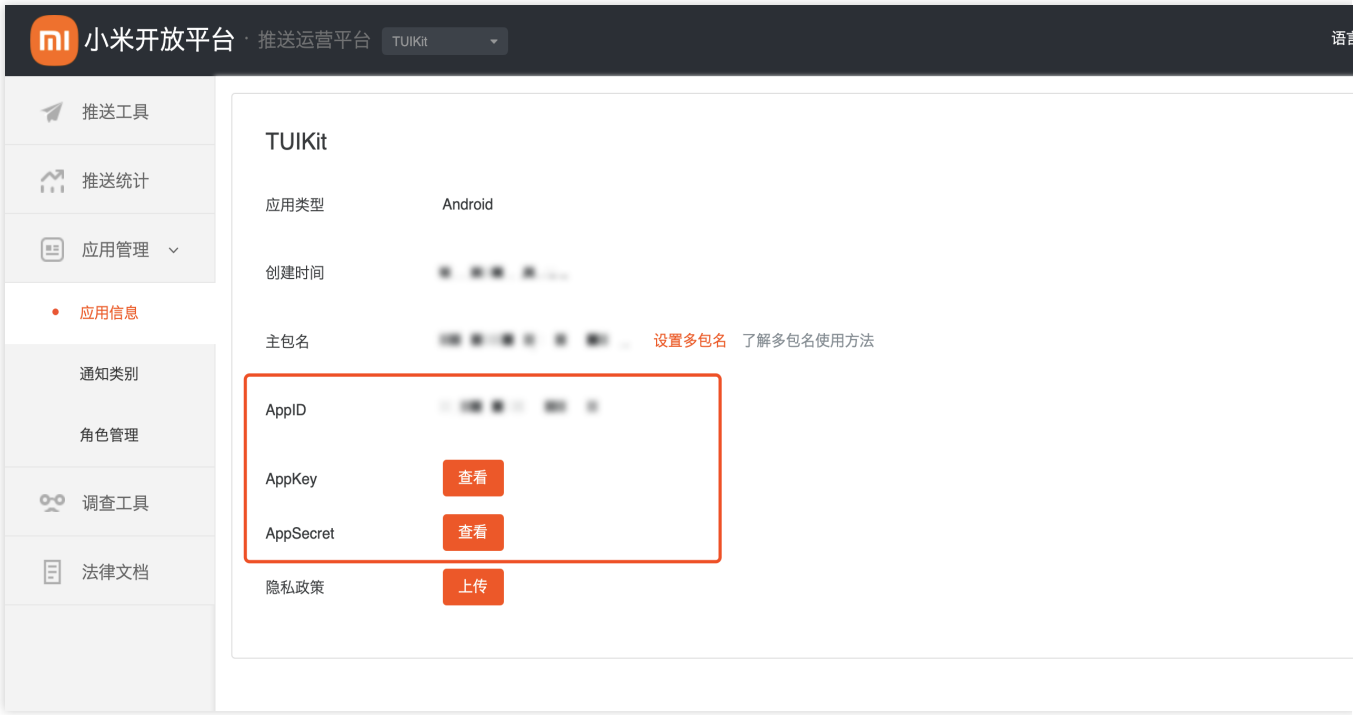


步骤4：查看获取应用信息

进入推送运营平台的**应用信息**页面，查看**应用信息**。

©2013-2025 Tencent Cloud International Pte. Ltd.



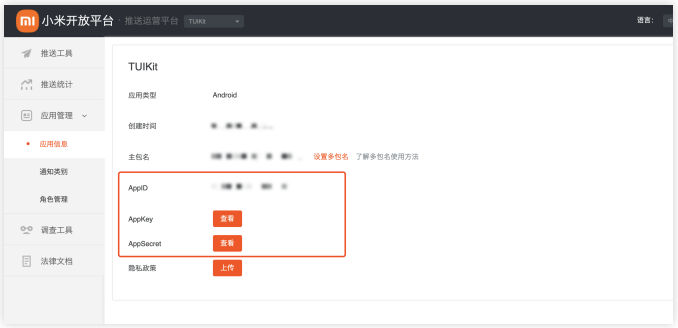


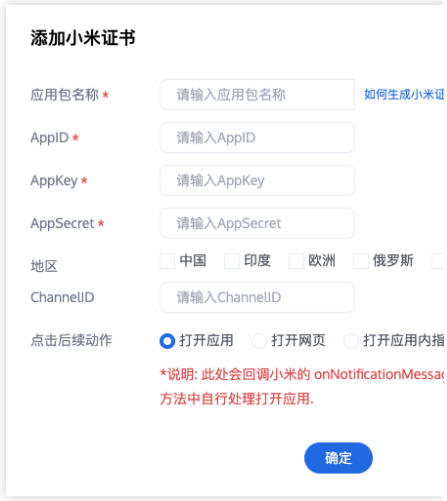
步骤5：配置推送证书

登录腾讯云 即时通信 Chat 控制台，在**推送 > 接入设置**功能栏添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.



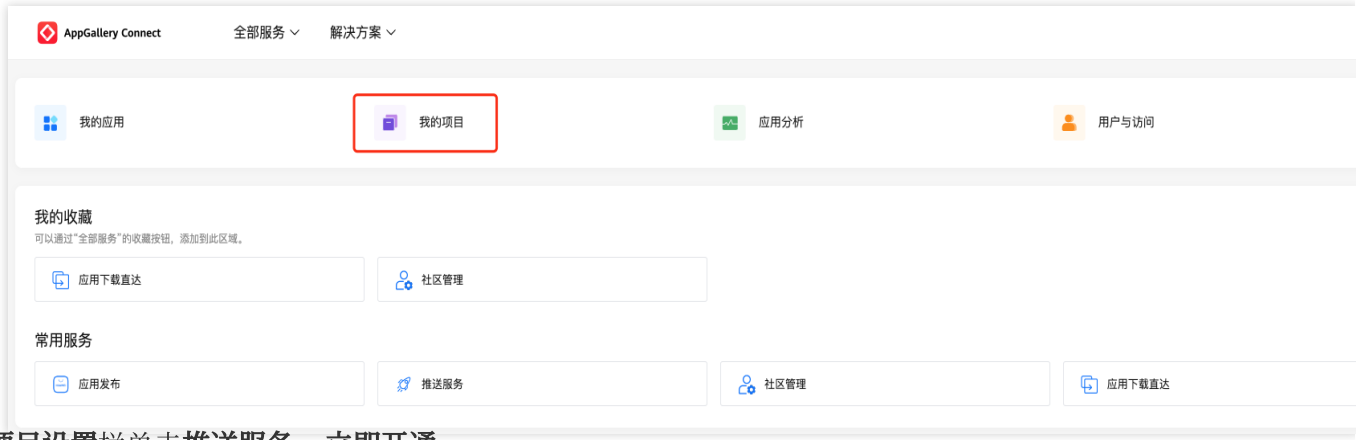


步骤1：注册华为开发者账号

进入 华为开发者联盟，注册华为开发者账号，详情请参见 注册账号。

步骤2：创建应用

1. 在管理中心单击**我的项目**，添加一个新的项目。



2. 在**项目设置**栏单击**推送服务 > 立即开通**。

©2013-2025 Tencent Cloud International Pte. Ltd.



3.单击**项目设置 > API 管理**，开启推送服务的权限。



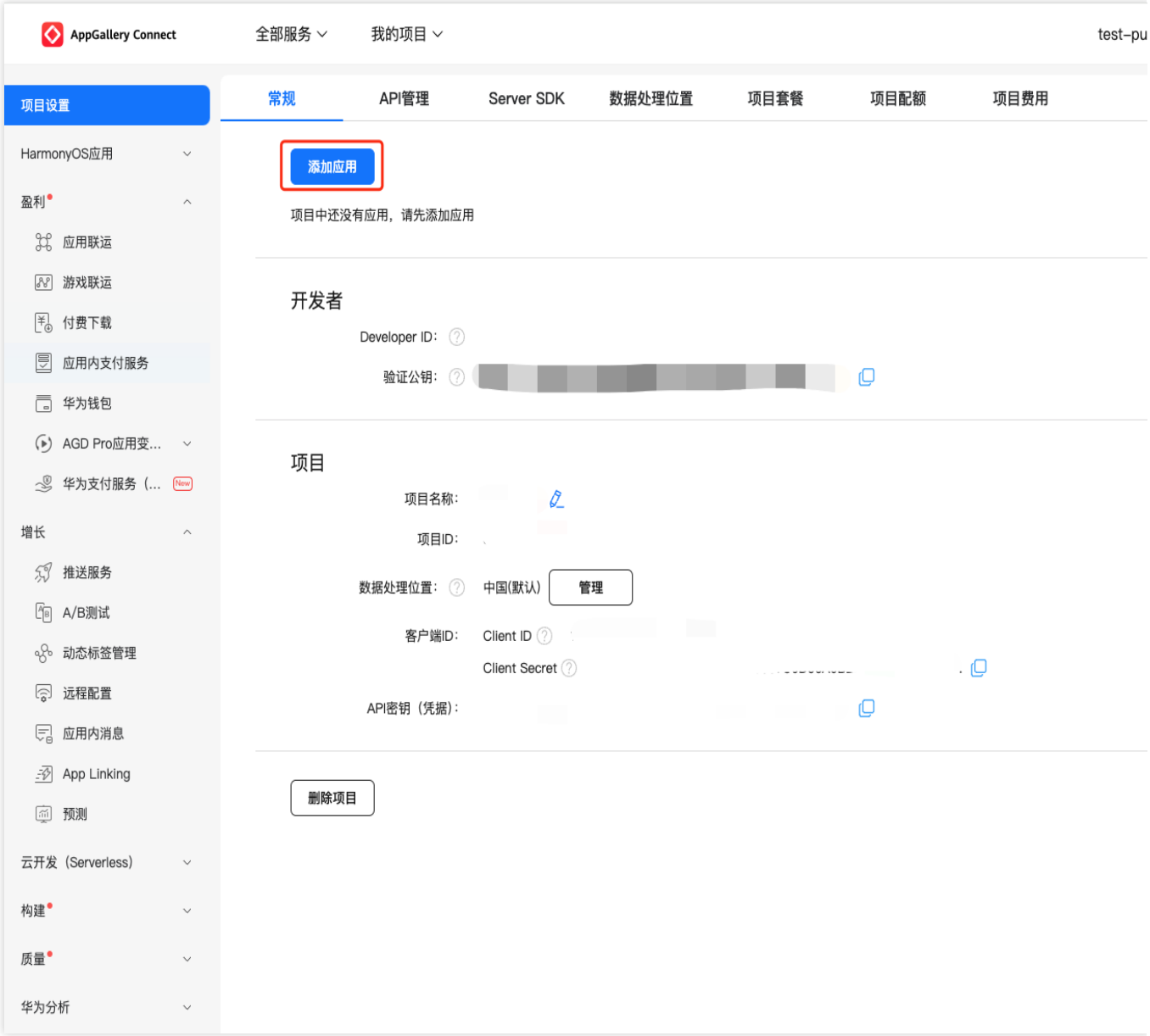
步骤3. 添加应用

单击**项目设置 > 常规**，添加应用。

注意：

应用包名与插件应用包名保持一致。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤4：获取应用信息

单击**项目设置 > 常规**，获取应用信息。

说明：

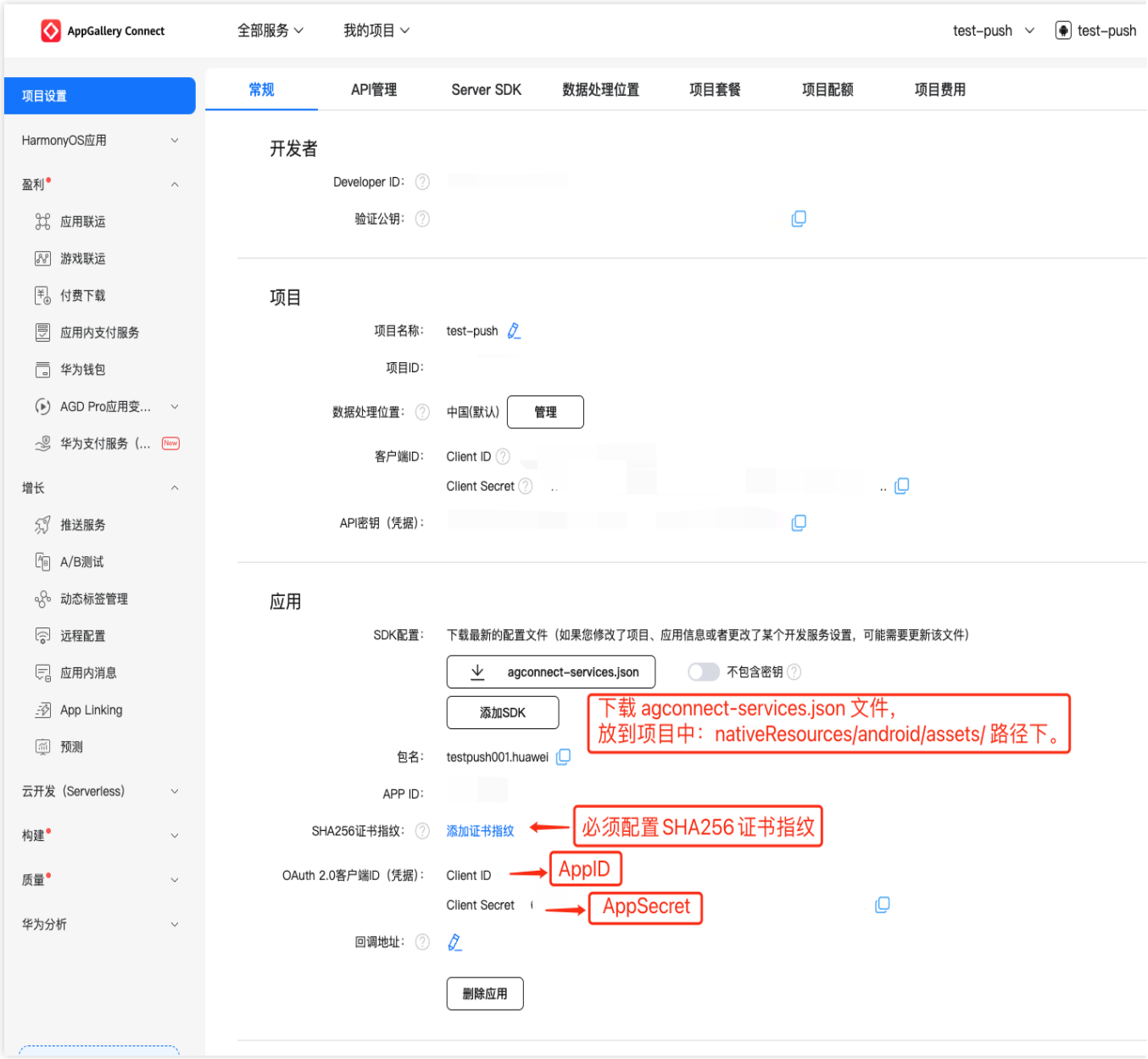
常规页面包含项目和应用的 Client ID 和 Client Secret，两者对应的参数不一致，请下拉至页面底部，获取应用的 Client ID 和 Client Secret。

必须添加打包的 SHA256证书指纹，SHA256 证书指纹需与自己的打包证书一致。

下载 agconnect-services.json 文件，放到项目中：nativeResources/android/assets/ 路径下。

修改了项目、应用信息、开发服务设置，都需要重新下载配置 agconnect-services.json 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤5：添加推送证书

登录腾讯云 即时通信 Chat 控制台，单击**推送** > **接入设置**添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置
	注意： Client ID 对应 AppID，Client

©2013-2025 Tencent Cloud International Pte. Ltd.





回执配置

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/huawei

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/huawei

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/huawei

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/huawei

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/huawei

China: https://api.im.qcloud.com/v3/offline_push_report/huawei

注意：

通知栏推送：应用需在 OPPO 软件商店上架；

通知栏推送测试权限：每天仅可推送1000条消息，限测试使用。应用上架后需重新申请“通知栏推送”权限，以获得正常消息推送数量；

平台将会在1个工作日内返回审核结果，开发者可以在申请页面查看审核结果，其他问题可咨询开放平台客服。

步骤1：注册 OPPO 开发者账号

进入 OPPO开放平台，注册 OPPO 开发者账号，详情参见 OPPO 企业开发者账号注册。

步骤2：创建应用

进入 OPPO 开放平台，单击**产品 > 应用分发> OPPO 软件商店 > 发布应用**进入管理中心，创建应用。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤3：开通 PUSH 服务

1. 进入 OPPO 开放平台，单击**产品** > **移动服务** > **推送服务**进入推送主页，单击**申请接**入开通推送服务。



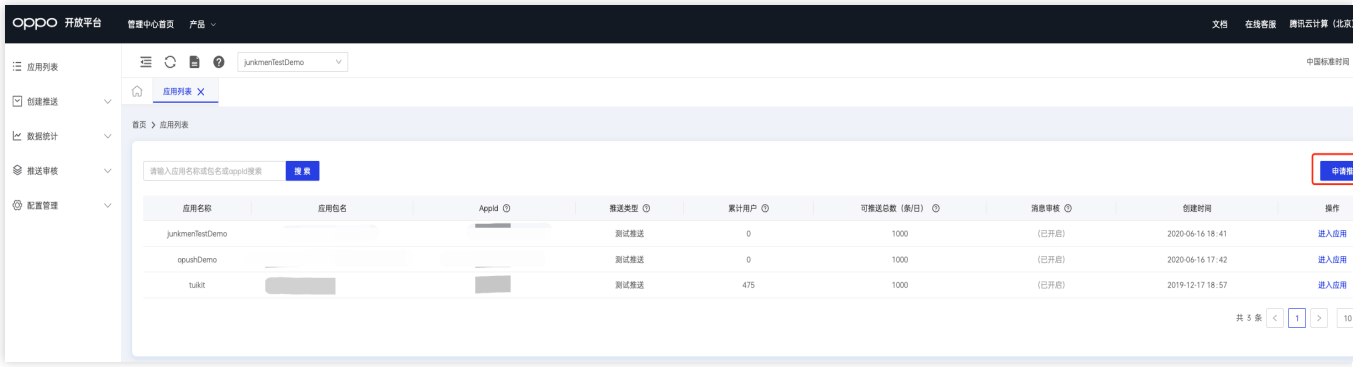
2. 单击进入**管理中心** > **应用列表** > **申请推送服务**界面，为未开启服务的应用申请推送权限。

说明：

已开启服务：已申请 PUSH 权限并通过的应用。

未开启服务：可申请 PUSH 权限的应用。

©2013-2025 Tencent Cloud International Pte. Ltd.





3. 单击**申请开通**。在未开启服务中单击需要申请 PUSH 权限的应用，进入 PUSH 服务并点击**申请开通**。



步骤4：添加推送证书

登录腾讯云 即时通信 Chat 控制台 ，在**推送 > 接入设置**功能栏添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.





注意：

若应用没有上架应用市场，推送权限受限，不可在 vivo 官网的 Web 界面和 API 后台发送正式消息，可在 API 后台向设置的测试设备发送测试消息进行测试。

步骤1：注册 vivo 开发者账号

进入 vivo开放平台，注册 vivo 开发者账号，详情参见 vivo 企业开发者账号注册。

步骤2：新建应用

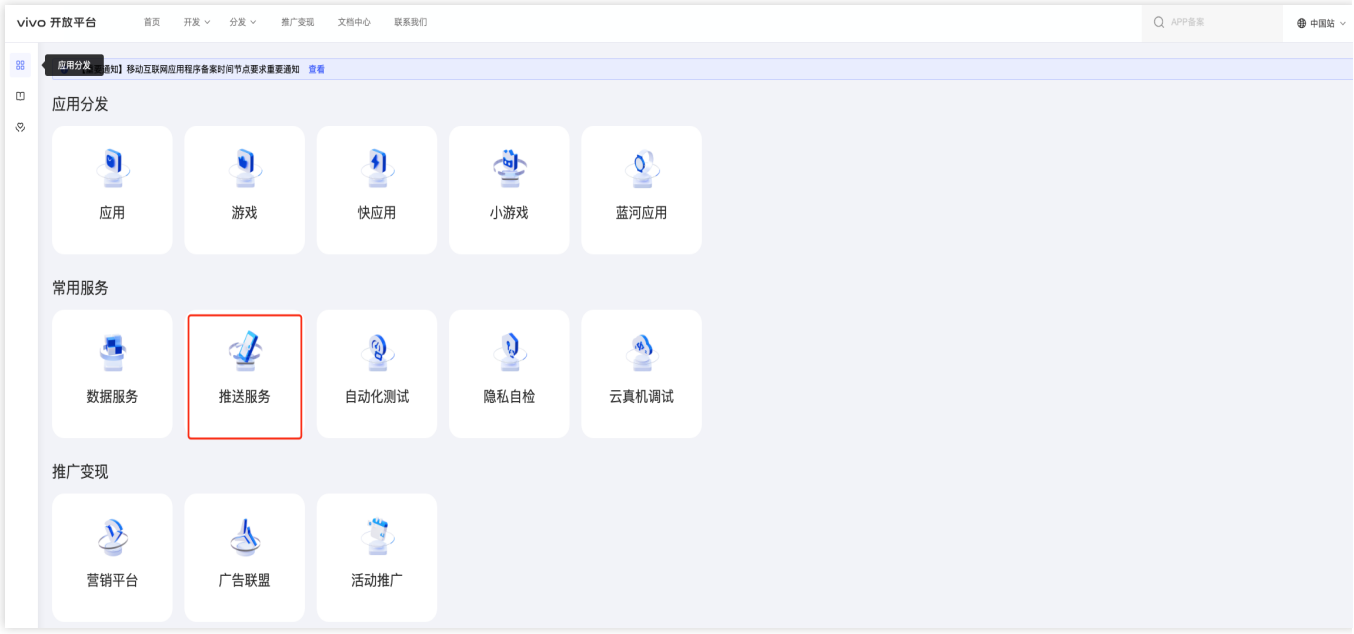
进入**vivo 开放平台**，单击**分发 > 应用分发 > 应用商店 > 上传应用**来新建您的应用。



步骤3：开通推送

进入管理中心单击**推送服务 > 推送申请**为新建的应用申请开通推送。

©2013-2025 Tencent Cloud International Pte. Ltd.





步骤4：获取应用信息

进入推送运营平台，单击**应用管理 > 应用信息**，获取应用信息。



步骤5：添加推送证书

登录腾讯云 即时通信 Chat 控制台 ，单击**推送 > 接入设置**添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.





回执配置

回执地址:

Singapore :https://apisgp.im.qcloud.com/v3/offline_push_report/vivo

Korea:https://apikr.im.qcloud.com/v3/offline_push_report/vivo

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/vivo

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/vivo

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/vivo

China: https://api.im.qcloud.com/v3/offline_push_report/vivo

步骤1：注册魅族开发者账号

注册魅族开发者账号，详情参见 开发者注册。

步骤2：创建应用

1. 单击**控制台** > **Flyme 推送**。

©2013-2025 Tencent Cloud International Pte. Ltd.



2. 填写应用信息后，创建应用。

注意：

应用包名与插件应用包名保持一致。



步骤3：获取应用信息

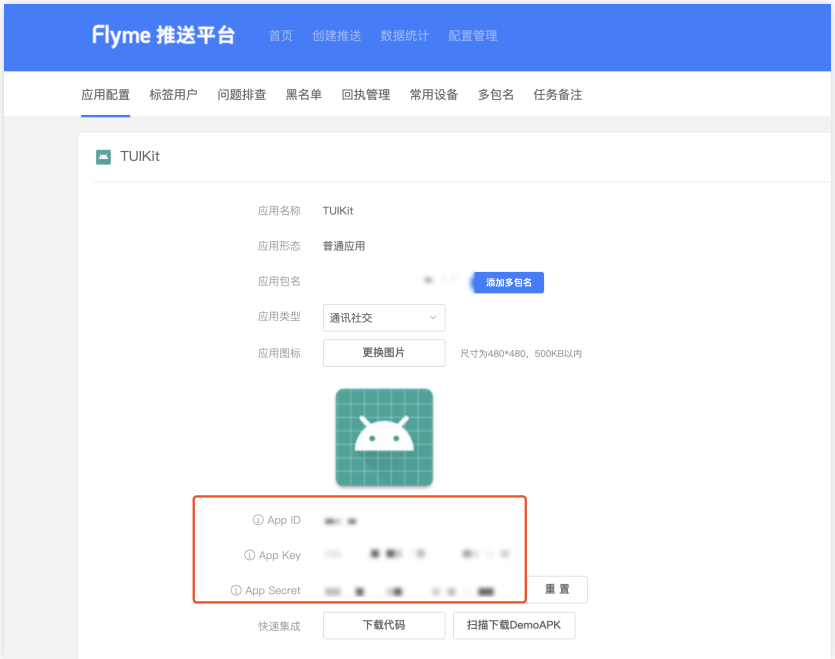
在应用列表中单击**打开应用**。进入配置管理页面，获取应用信息。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤4：添加推送证书

登录腾讯云 即时通信 Chat 控制台，单击**推送** > **接入设置**功能栏添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置
	

回执配置

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/meizu

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/meizu

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/meizu

©2013-2025 Tencent Cloud International Pte. Ltd.

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/meizu

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/meizu

China: https://api.im.qcloud.com/v3/offline_push_report/meizu

步骤1. 注册荣耀开发者账号

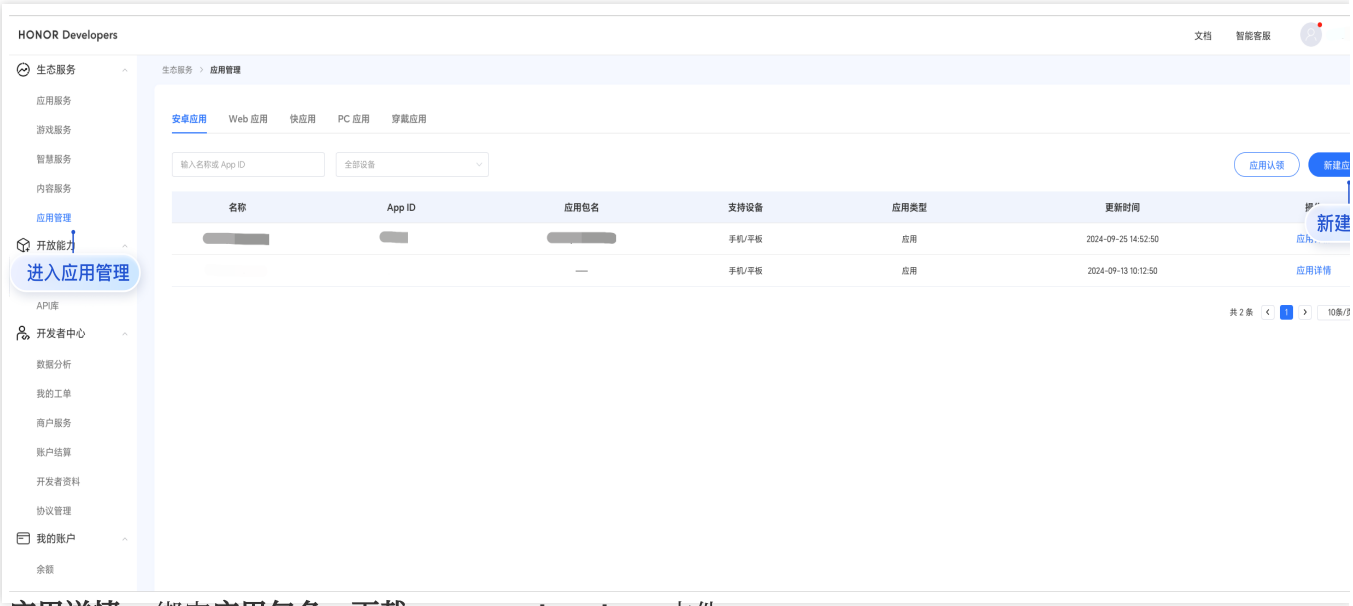
注册荣耀开发者账号，详情参见 开发者注册。

步骤2: 进入管理中心页面



步骤3: 创建应用

1. 进入**应用管理**，点击**新建应用**创建新应用。



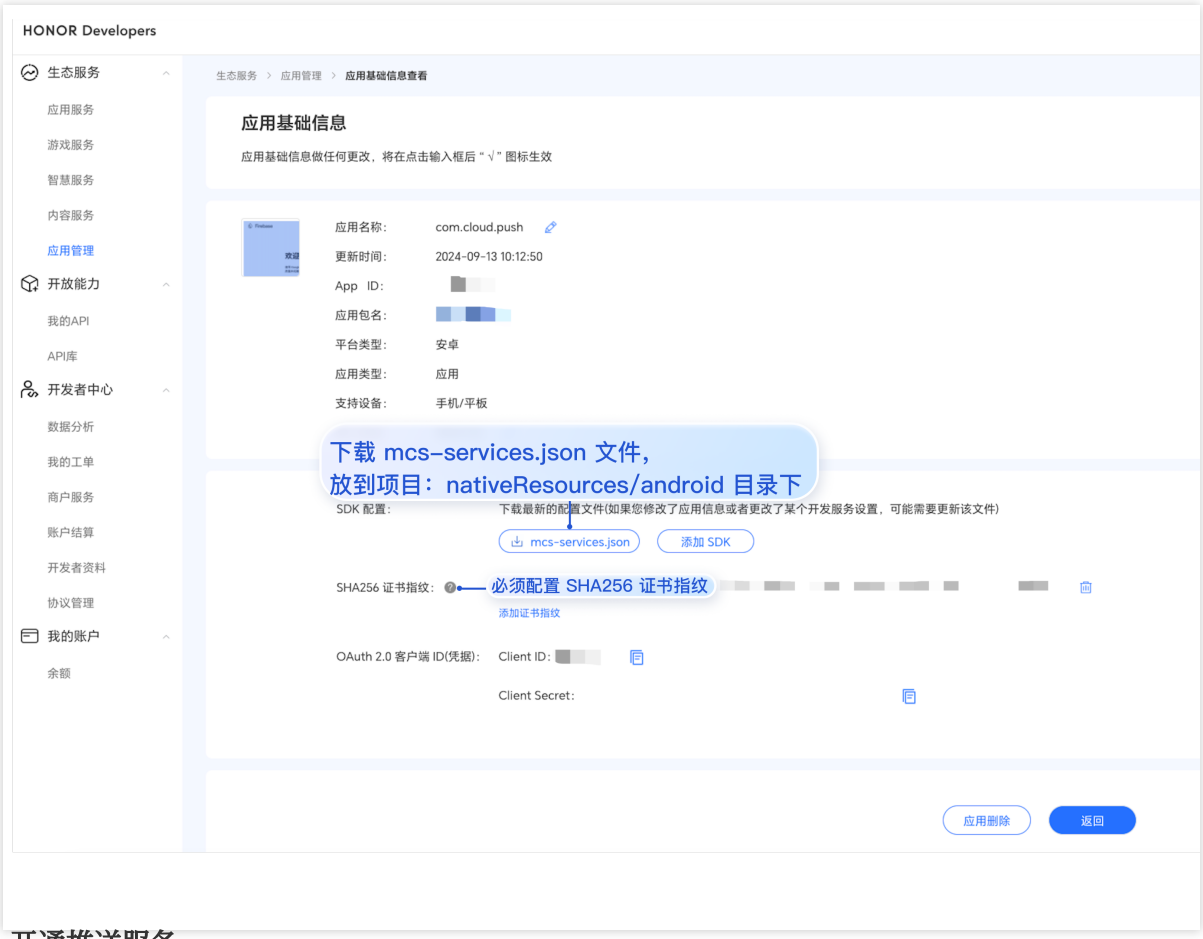
2. 进入**应用详情**，绑定**应用包名**，**下载 mcs-services.json** 文件。

说明：

必须添加打包的 SHA256证书指纹，SHA256 证书指纹需与自己的打包证书一致。

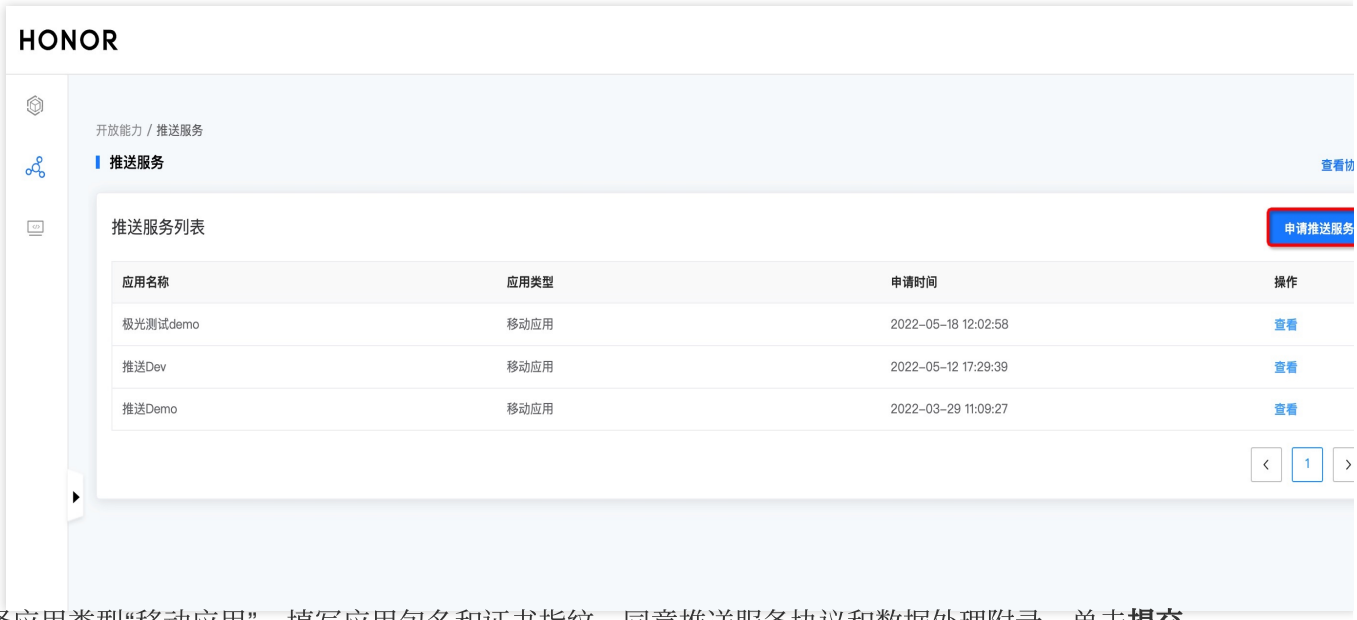
©2013-2025 Tencent Cloud International Pte. Ltd.

下载 mcs-services.json 文件，放到项目中：nativeResources/android/ 路径下。
修改了项目、应用信息、开发服务设置，都需要重新下载配置 mcs-services.json 文件。



步骤4: 开通推送服务

1. 单击**申请推送服务**进入应用申请页面。

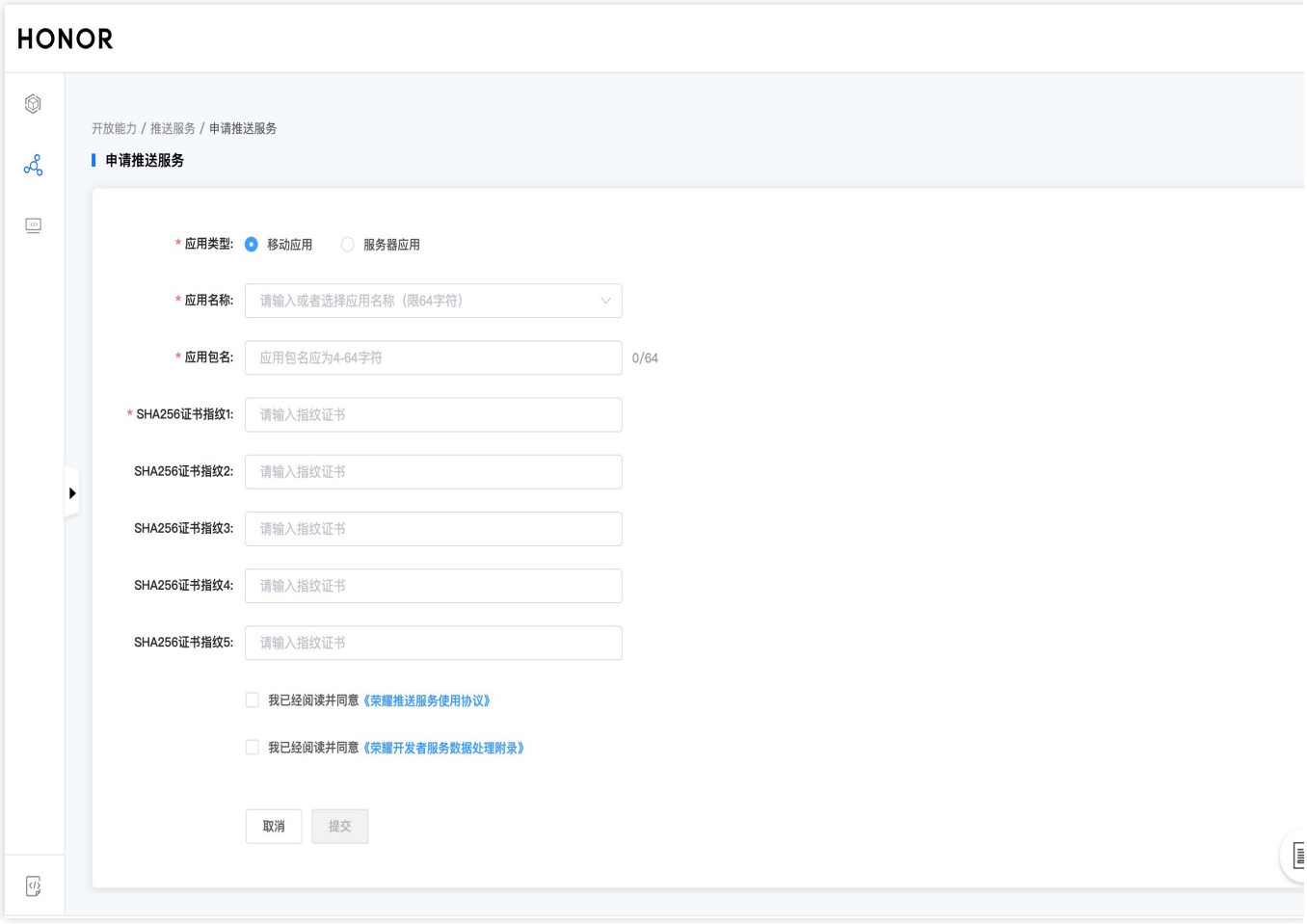


2. 选择应用类型“移动应用”，填写应用包名和证书指纹、同意推送服务协议和数据处理附录，单击**提交**。

©2013-2025 Tencent Cloud International Pte. Ltd.

注意：

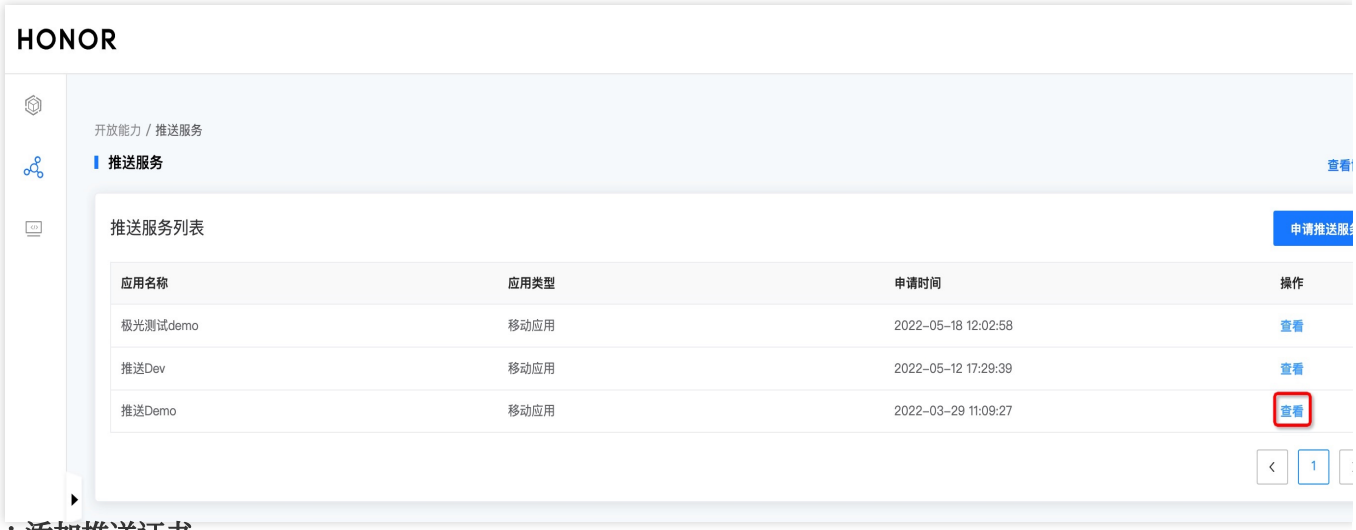
需要添加打包的 SHA256 证书指纹，SHA256 证书指纹需与自己的打包证书一致。



步骤5: 获取应用信息

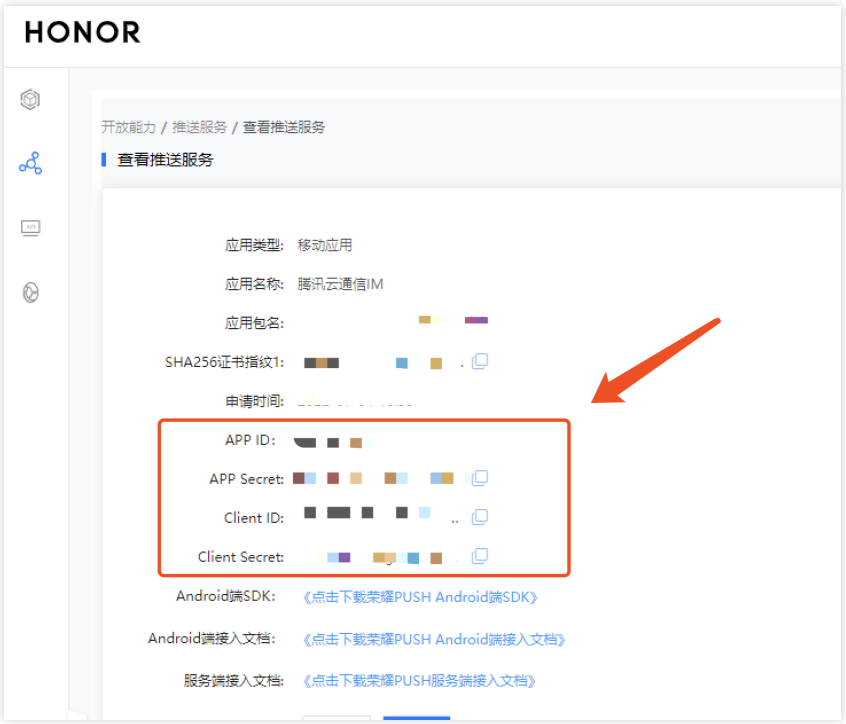
在**推送服务**列表中，单击**查看**，获取应用信息。

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤6：添加推送证书

登录腾讯云 即时通信 Chat 控制台，单击**推送** > **接入设置**，添加各个厂商推送证书，并将您获取的厂商的 AppID、AppKey、AppSecret 等参数配置给添加的推送证书。

厂商推送平台	Chat 控制台配置
	

回执配置

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/honor

©2013-2025 Tencent Cloud International Pte. Ltd.

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/honor

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/honor

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/honor

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/honor

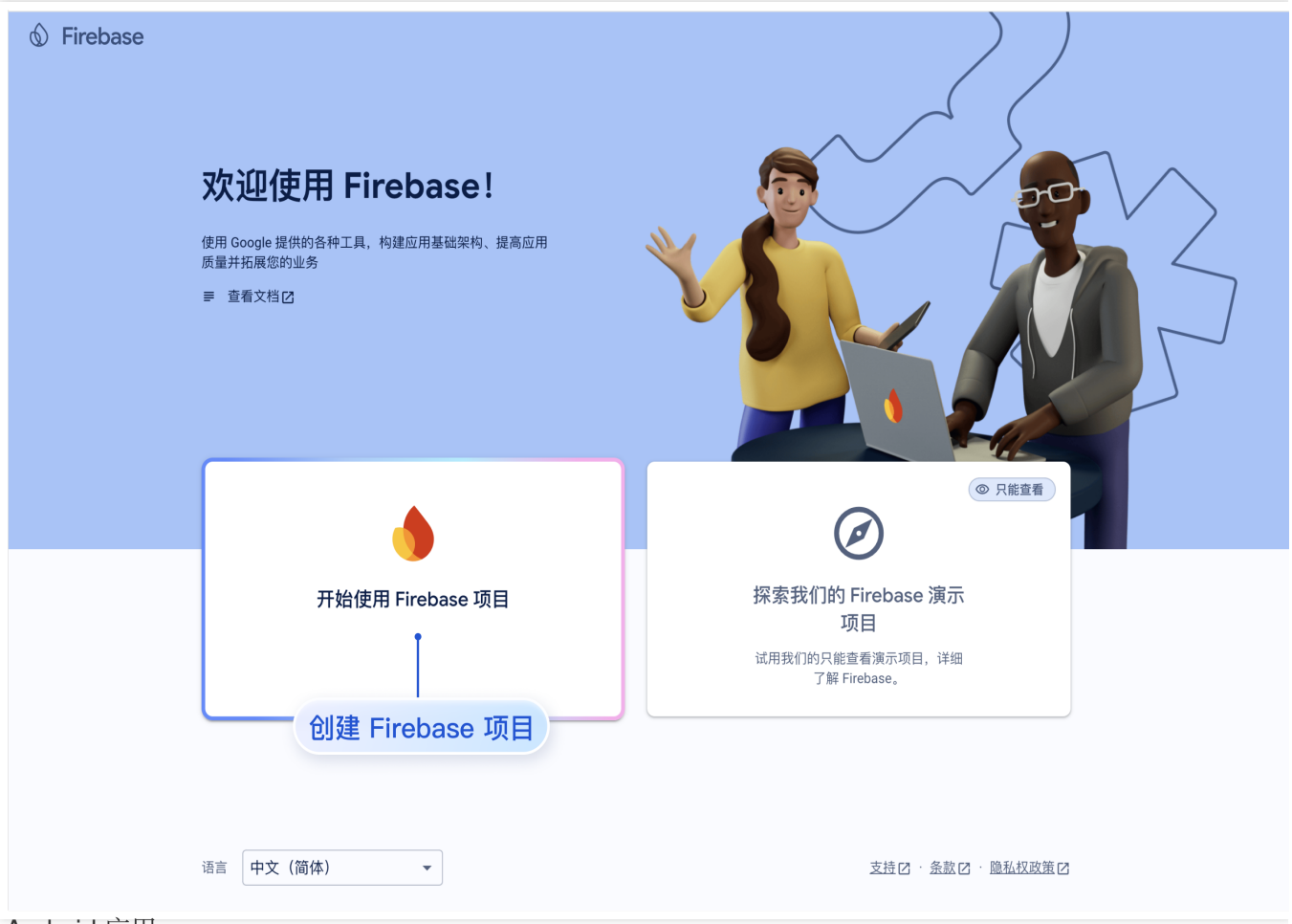
China: https://api.im.qcloud.com/v3/offline_push_report/honor

步骤1：进入Firebase 控制台

进入 Firebase 控制台，登录谷歌账号。

步骤2：创建应用

1. 单击**创建项目**，添加一个新的项目。



2. 进入Android 应用。

©2013-2025 Tencent Cloud International Pte. Ltd.



3. 输入应用信息，注册应用。

©2013-2025 Tencent Cloud International Pte. Ltd.



4. 下载并添加配置文件并完成注册。

©2013-2025 Tencent Cloud International Pte. Ltd.



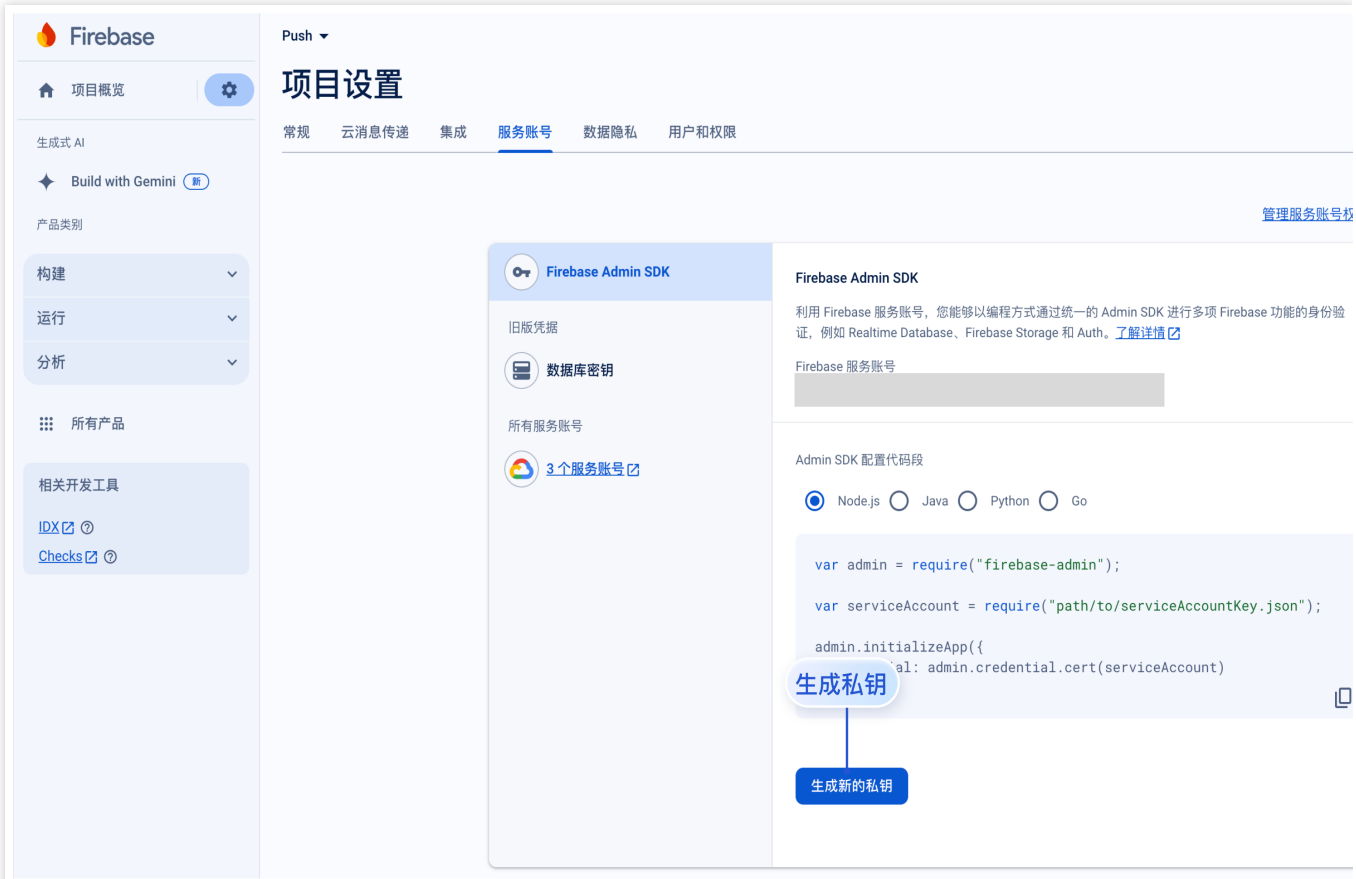
步骤3. 生成证书私钥。

1. 点击应用，进入应用管理。

©2013-2025 Tencent Cloud International Pte. Ltd.



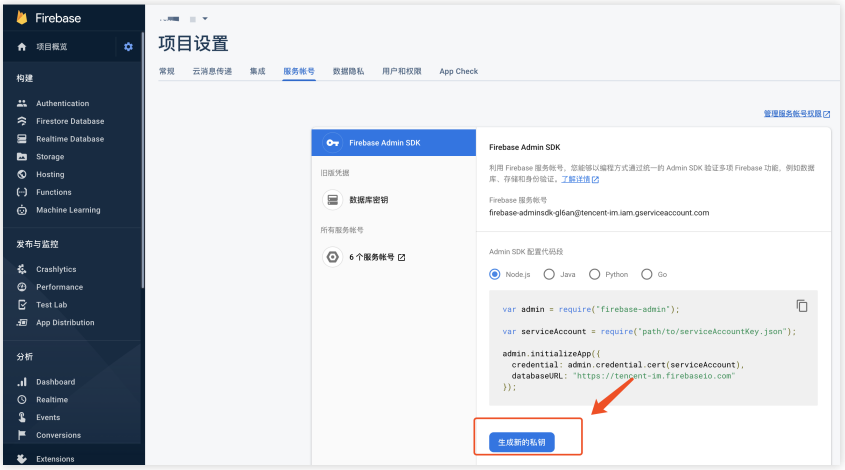
2. 在**项目设置**单击**服务账号 > 生成新的密钥**。



步骤4. 配置推送证书。

登录腾讯云 即时通信 Chat 控制台，在**推送 > 接入设置**功能栏添加各个厂商推送证书，并将您获取的厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

©2013-2025 Tencent Cloud International Pte. Ltd.

厂商推送平台	Chat 控制台配置
	

集成 uni-app 腾讯云推送服务（Push）之前，需要先向 Apple 申请 APNs 推送证书，然后上传推送证书到 Chat 控制台 。之后按照快速接入步骤接入即可。

Apple 厂商配置目前有两种主流的证书，p12 证书和 p8 证书。两种证书各有优劣，您可按需要选择其中的一种。

	证书类型	有效期和管理	安全性	灵动岛
p12 证书	p12 证书是一个包含公钥和私钥的二进制文件，用于基于证书的身份验证。它将公钥证书和私钥捆绑在一个文件中，扩展名为 .p12 或 .pfx。	p12 证书通常有一年的有效期，过期后需要重新生成和部署。每个应用程序都需要单独的 P12 证书来处理推送通知。	证书：p12 证书使用基于证书的身份验证，需要在服务器上存储私钥。这可能会增加安全风险，因为私钥可能会被未经授权的用户访问。	不支持
p8 证书	p8 证书是一个 Auth Key（授权密钥），用于基于令牌的身份验证。它是一个包含私钥的文本文件，扩展名为 .p8。	p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 P8 证书可以简化证书管理，因为您可以使用一个 p8 证书为多个应用程序提供推送通知服务。	p8 证书使用基于令牌的身份验证，这意味着您的服务器会周期性地生成一个 JSON Web Token（JWT）来与 APNs 建立连接。这种方法更安全，因为它不需要在服务器上存储私钥。	支持灵动岛推送

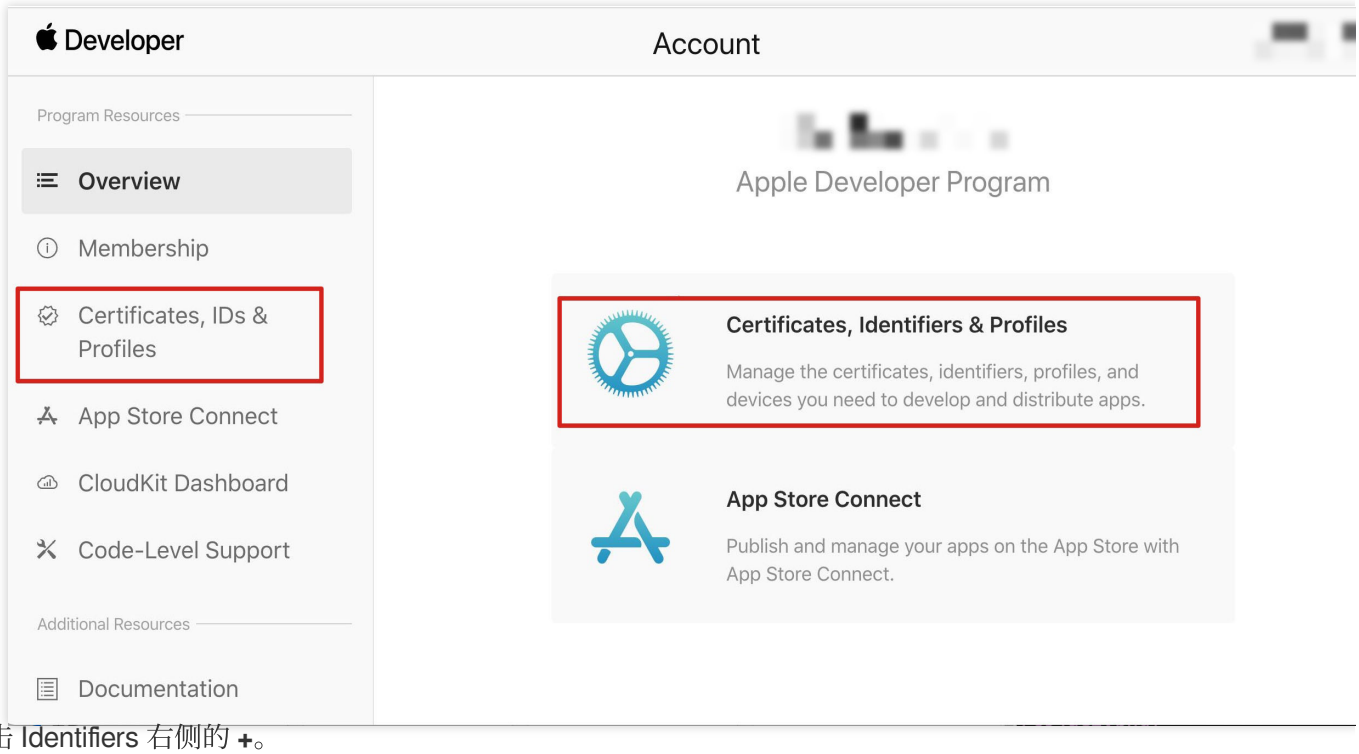
©2013-2025 Tencent Cloud International Pte. Ltd.

一、使用 p12 证书 （传统推送证书）

步骤1：申请 APNs 证书

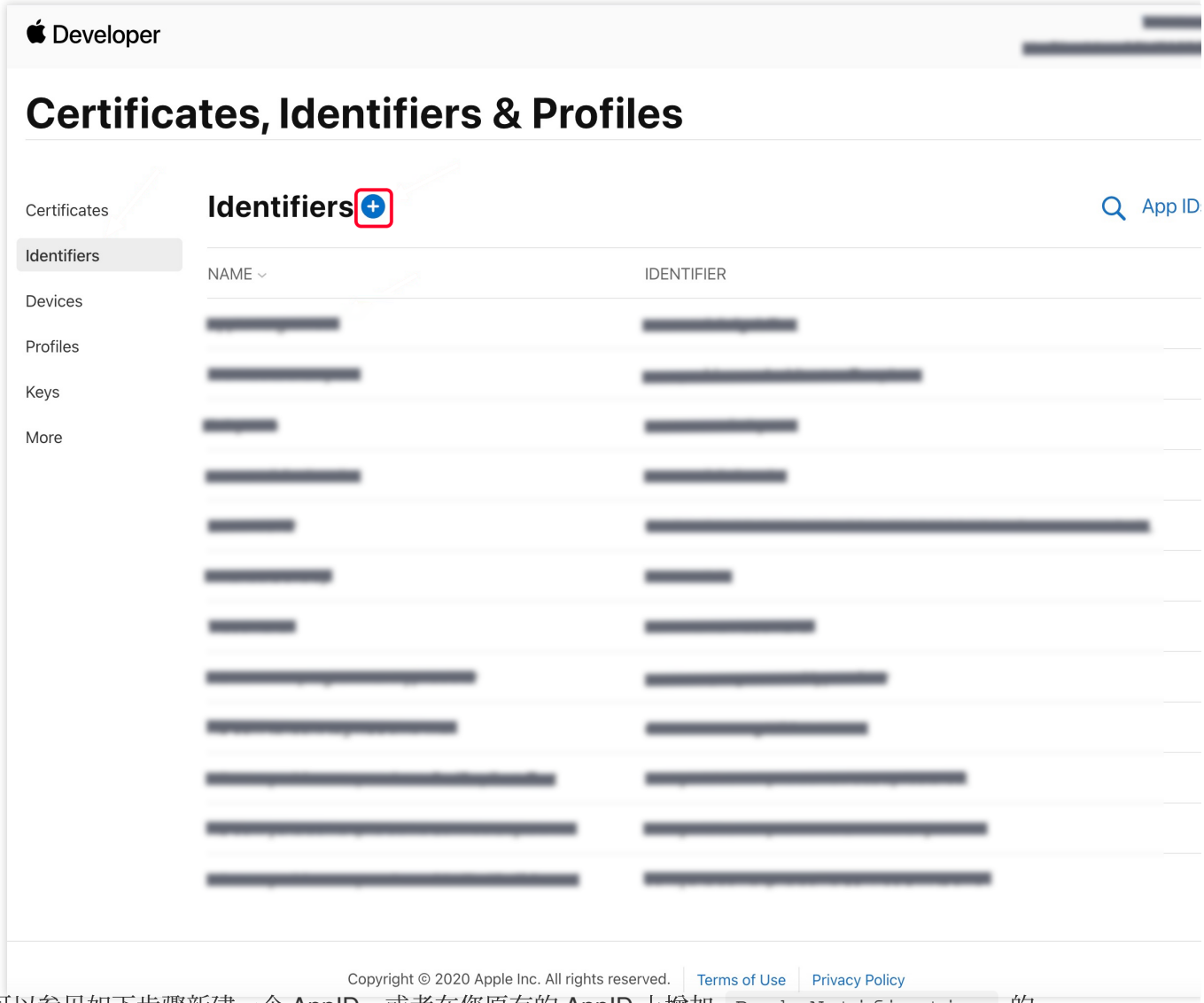
开启 App 远程推送

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



2. 单击 Identifiers 右侧的 +。

©2013-2025 Tencent Cloud International Pte. Ltd.



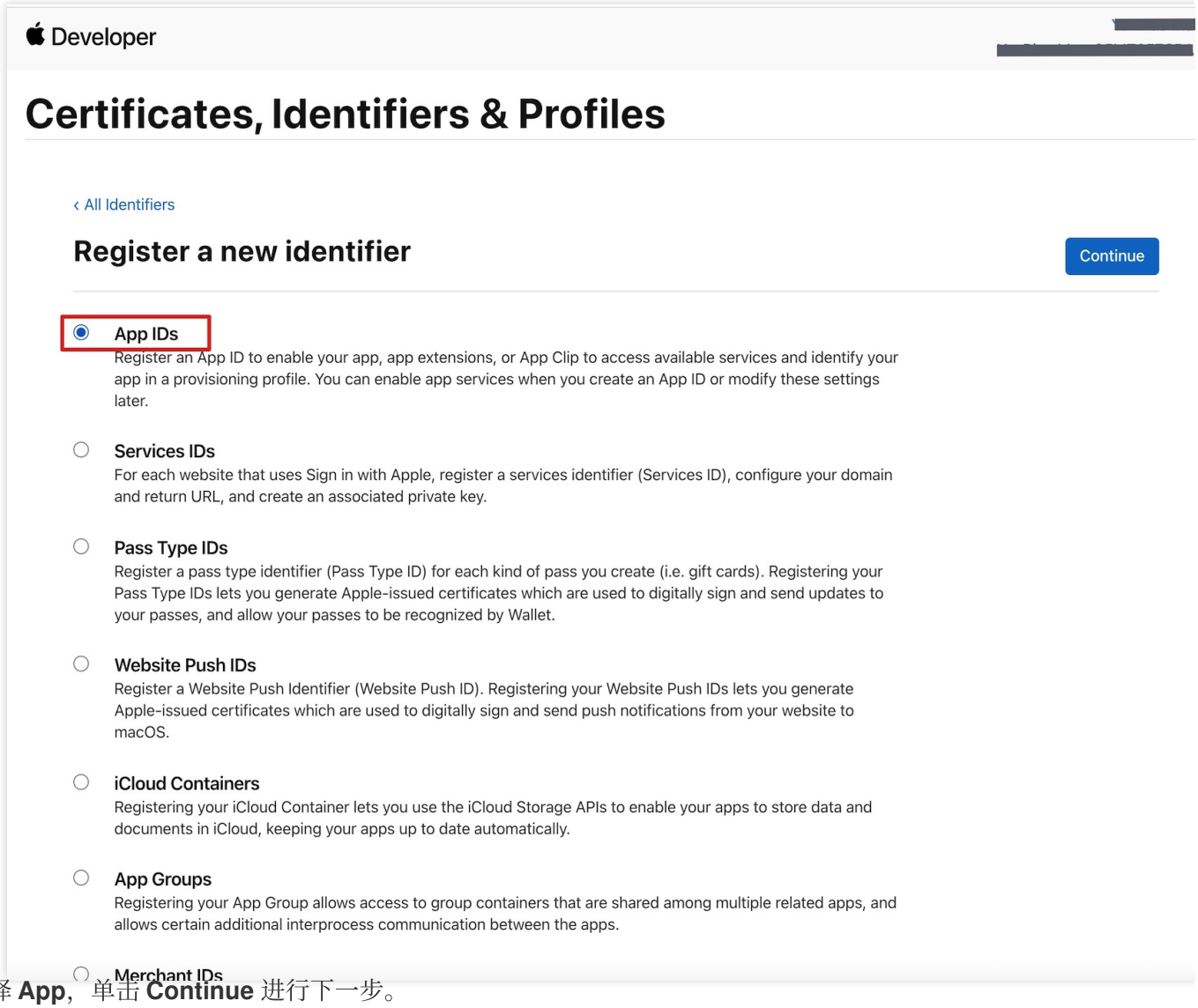
3. 您可以参见如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service` 。

说明：

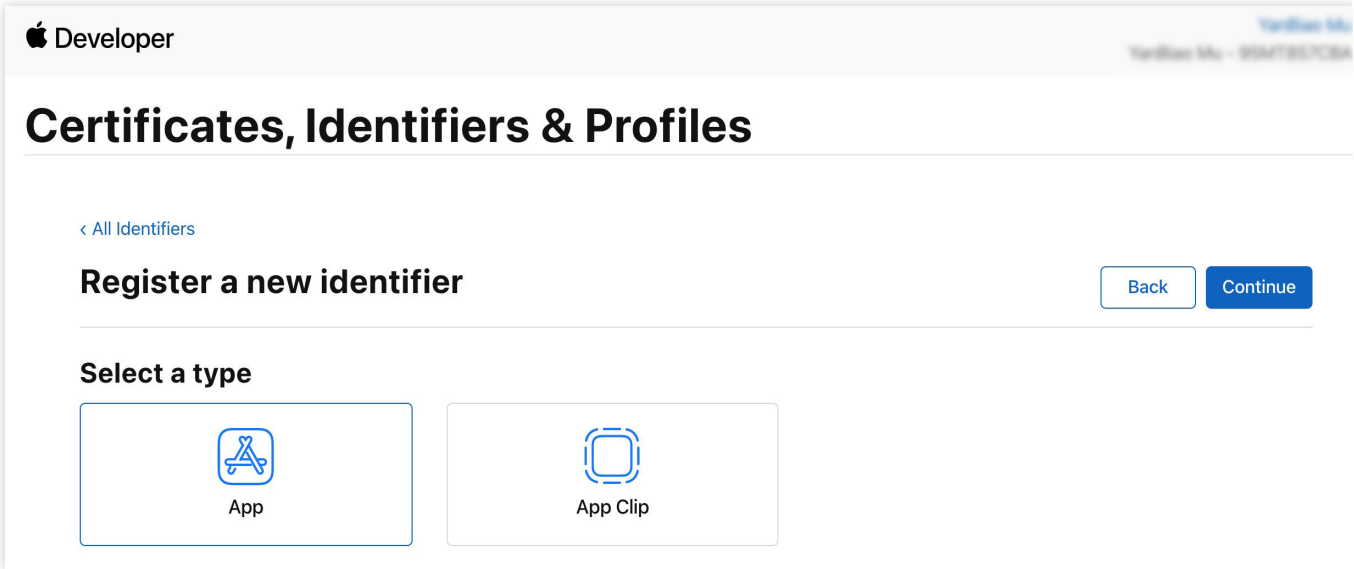
您 App 的 `Bundle ID` 不能使用通配符 `*` ，否则将无法使用远程推送服务。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

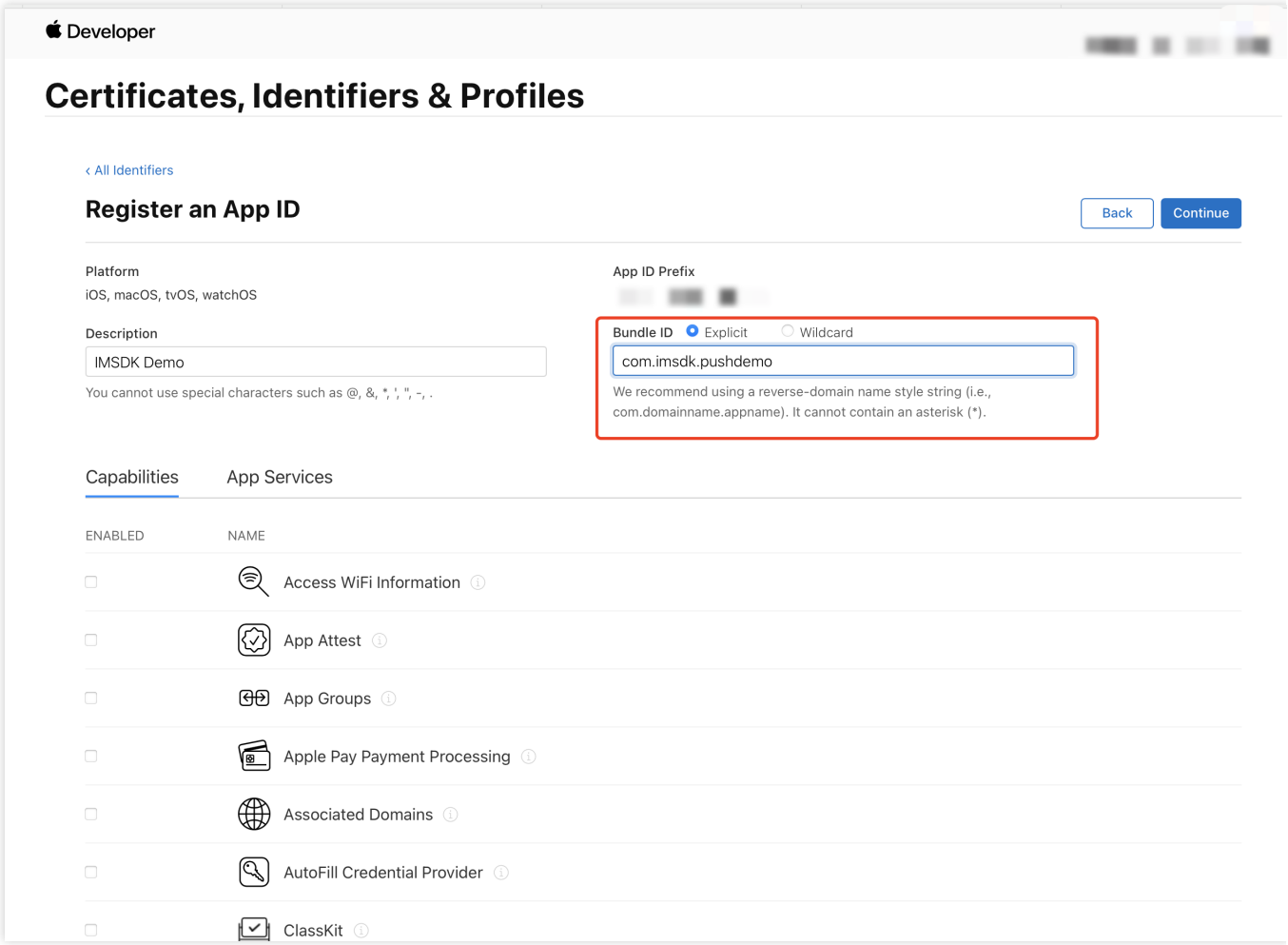


5. 选择 **App**，单击 **Continue** 进行下一步。



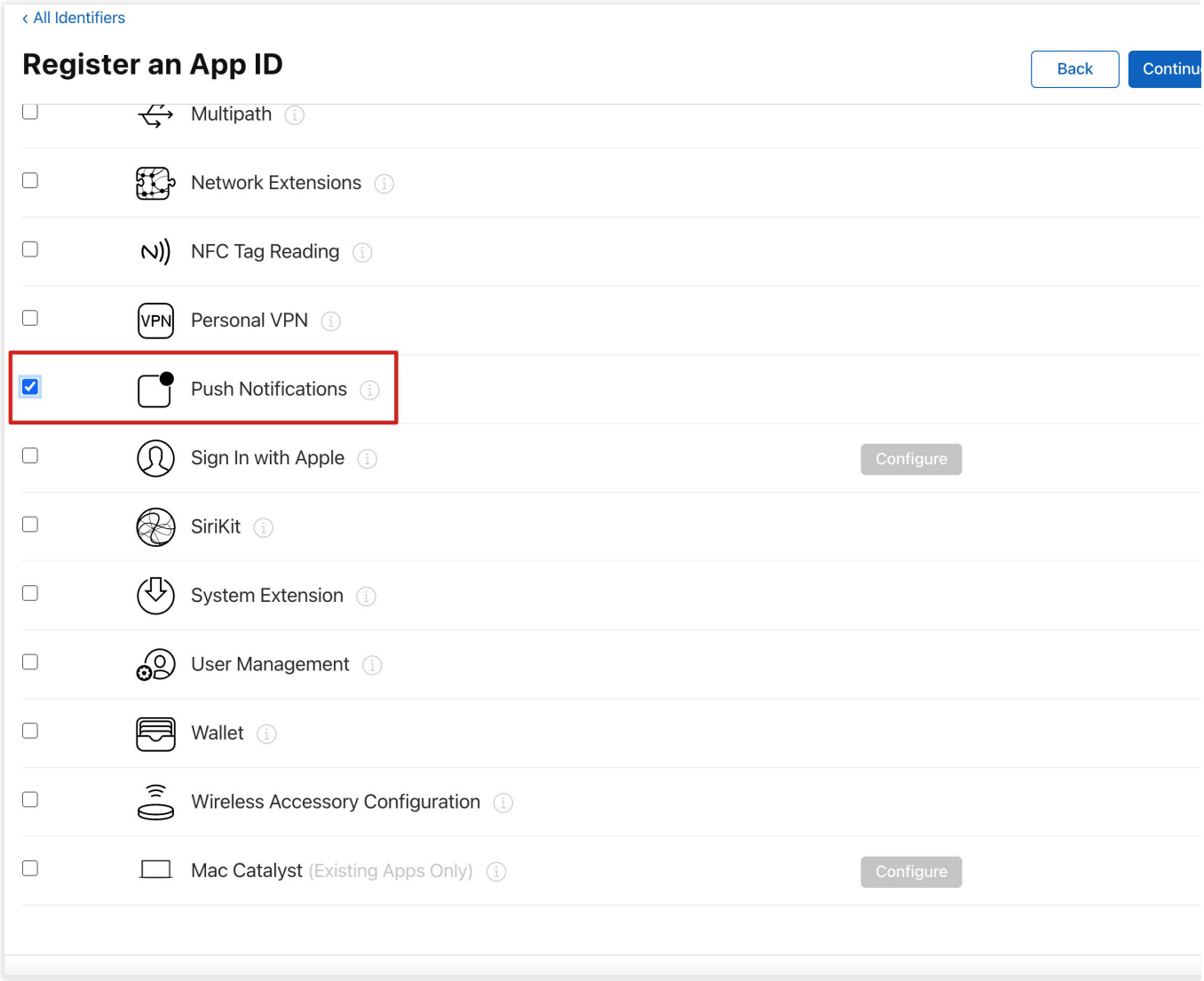
©2013-2025 Tencent Cloud International Pte. Ltd.

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



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

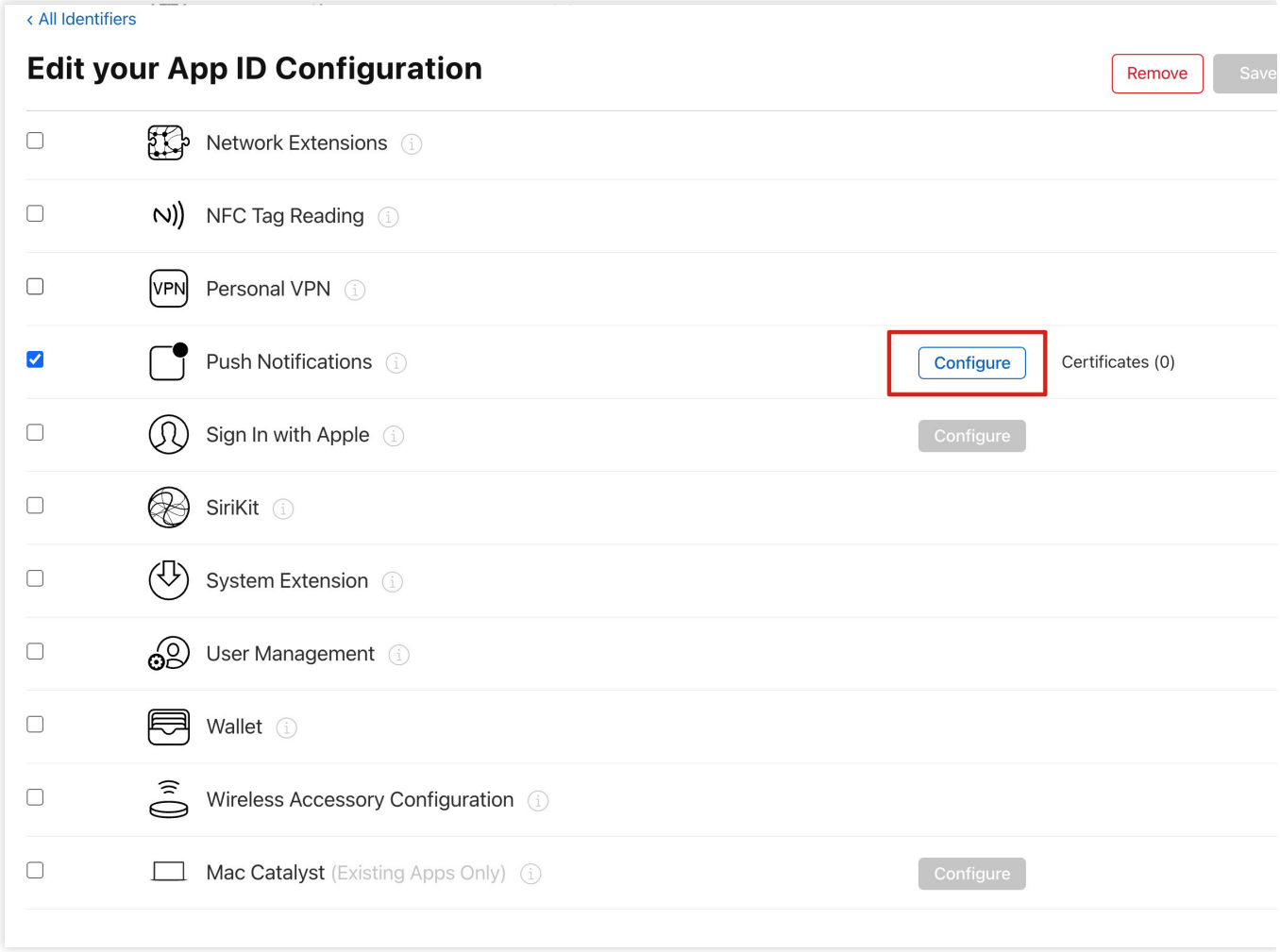
©2013-2025 Tencent Cloud International Pte. Ltd.



生成证书

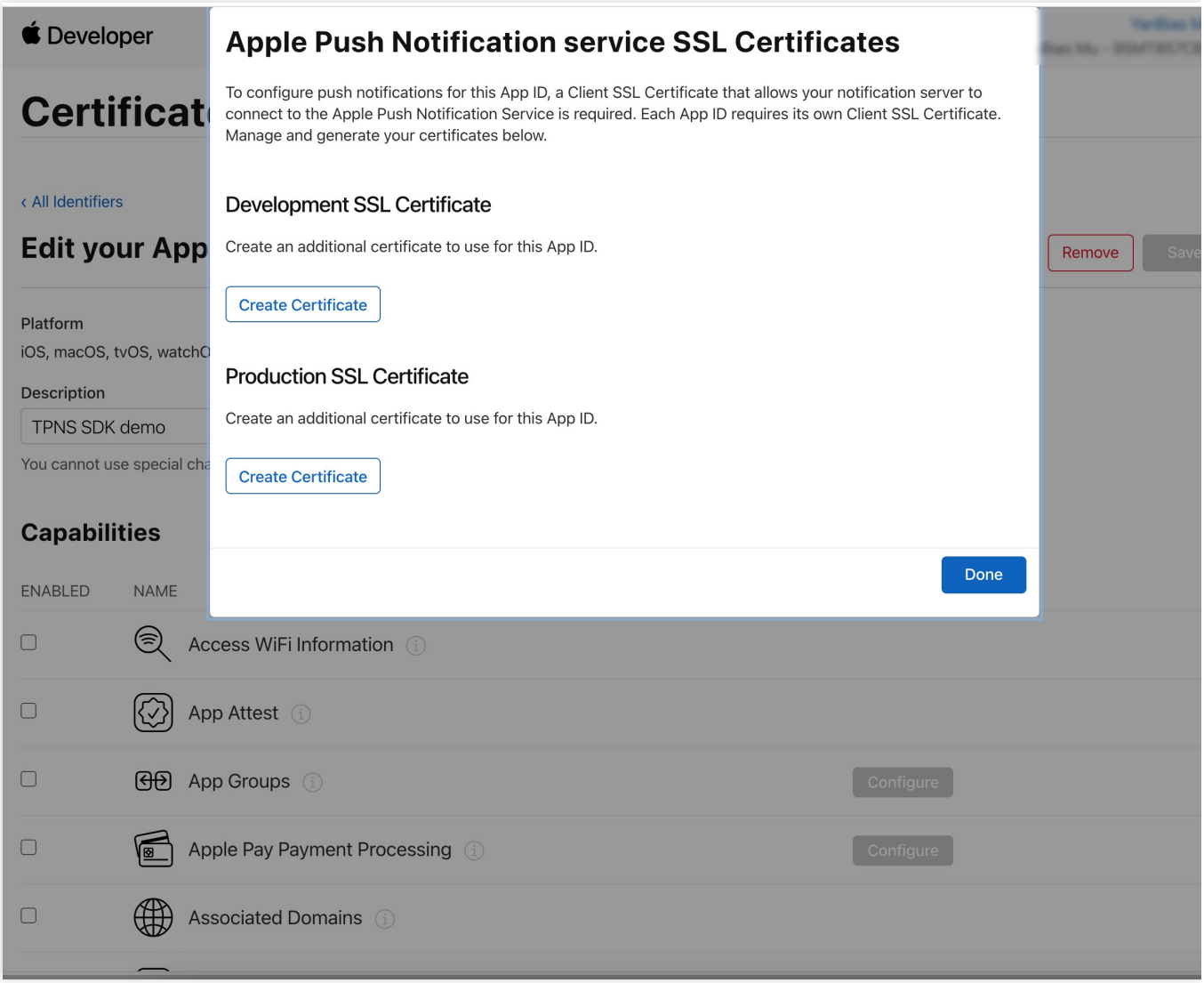
1. 选中您的 AppID，选择 **Configure**。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

©2013-2025 Tencent Cloud International Pte. Ltd.

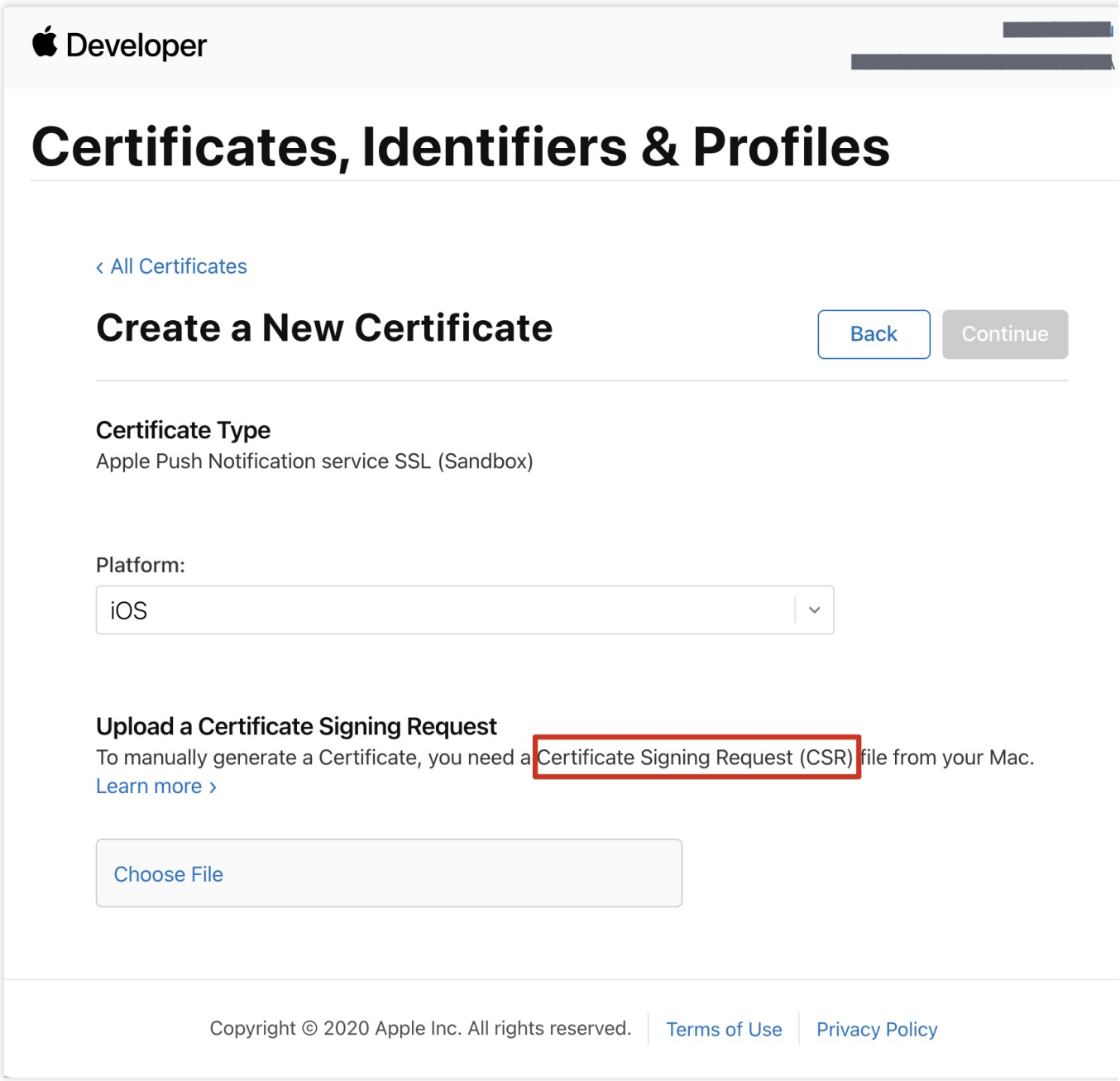


3.

我

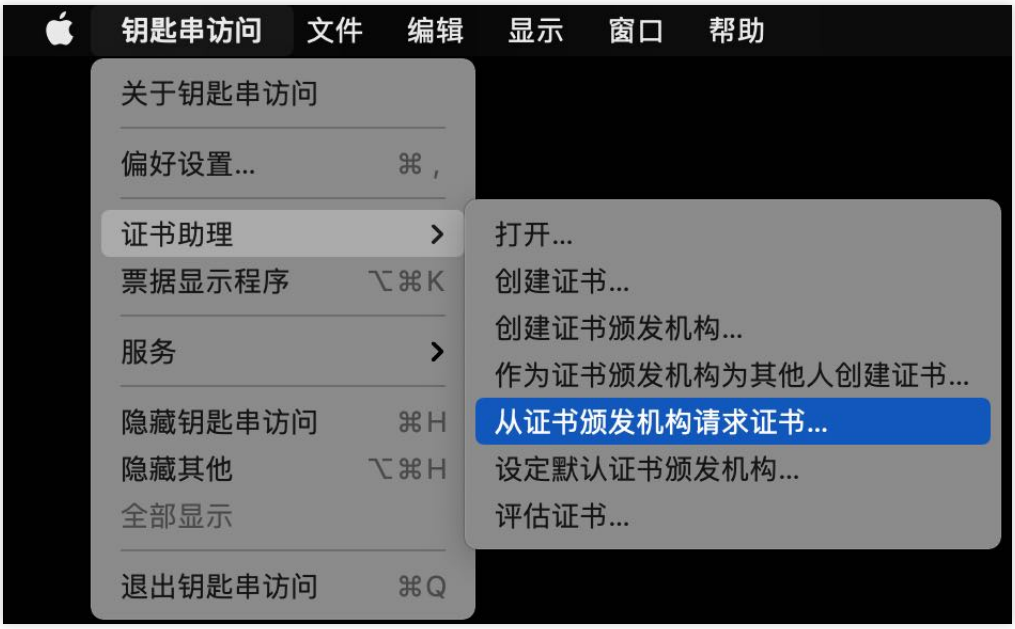
们先选择开发环境（Development）的 **Create Certificate**，系统将提示我们需要一个 Certificate Signing Request（CSR）。

©2013-2025 Tencent Cloud International Pte. Ltd.

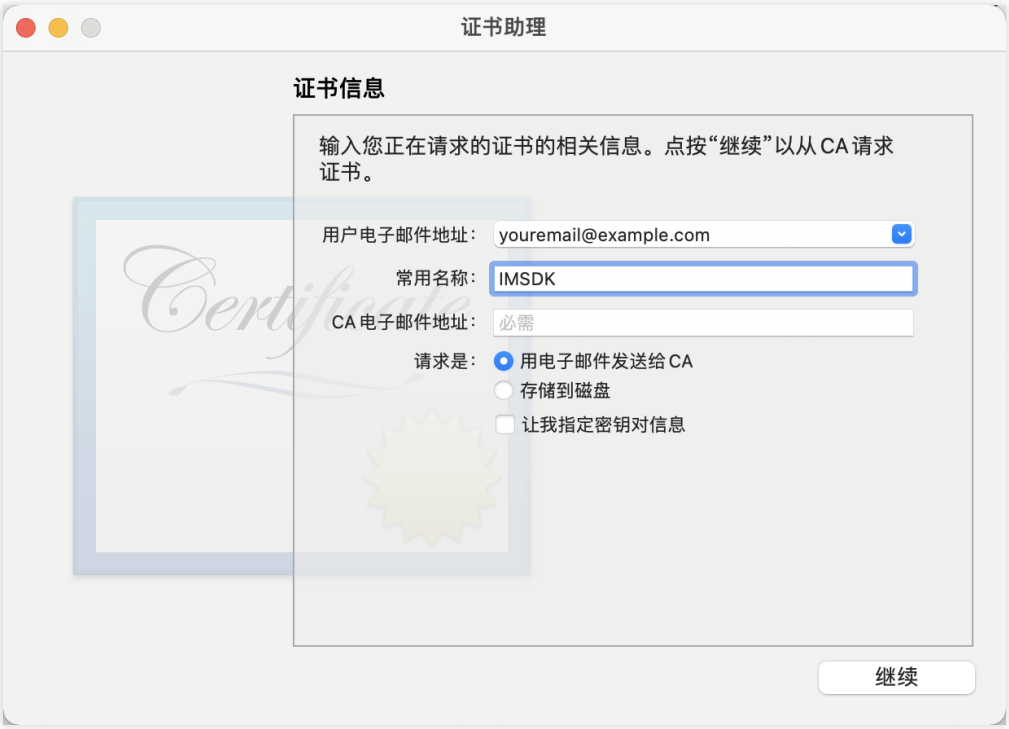


4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。

©2013-2025 Tencent Cloud International Pte. Ltd.

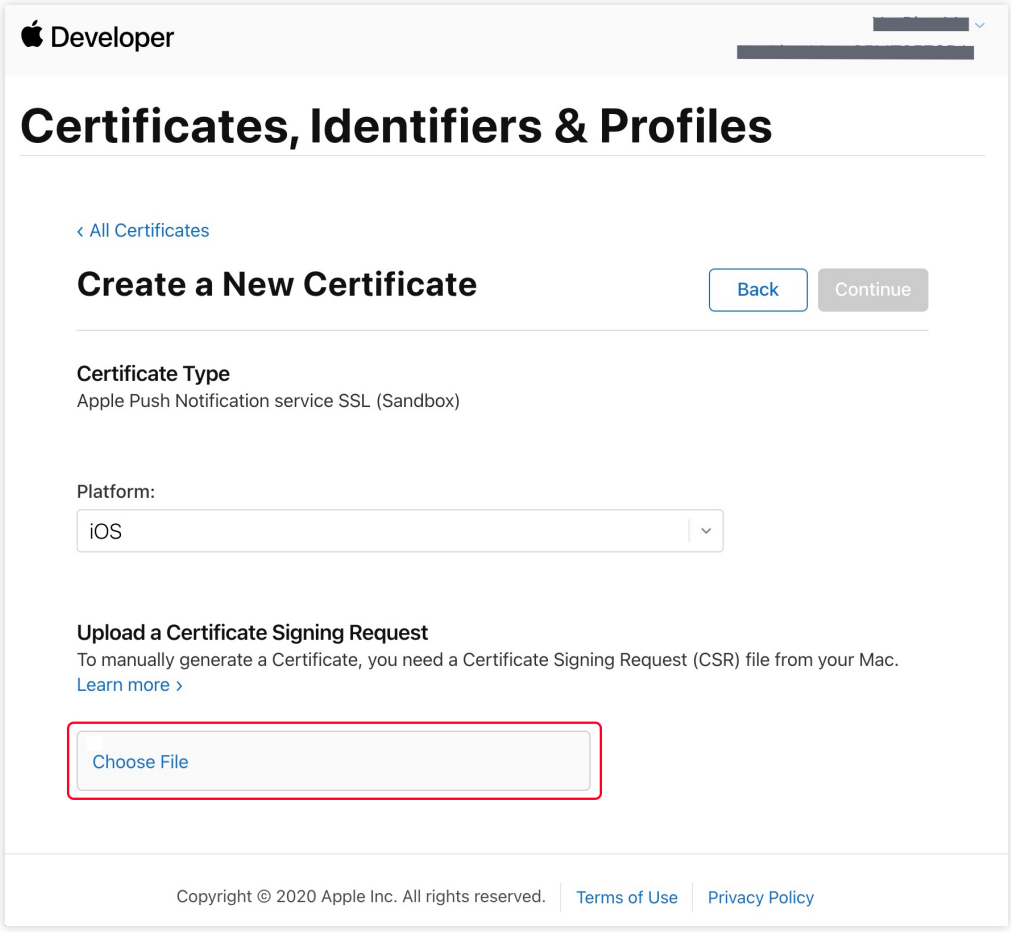


5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。



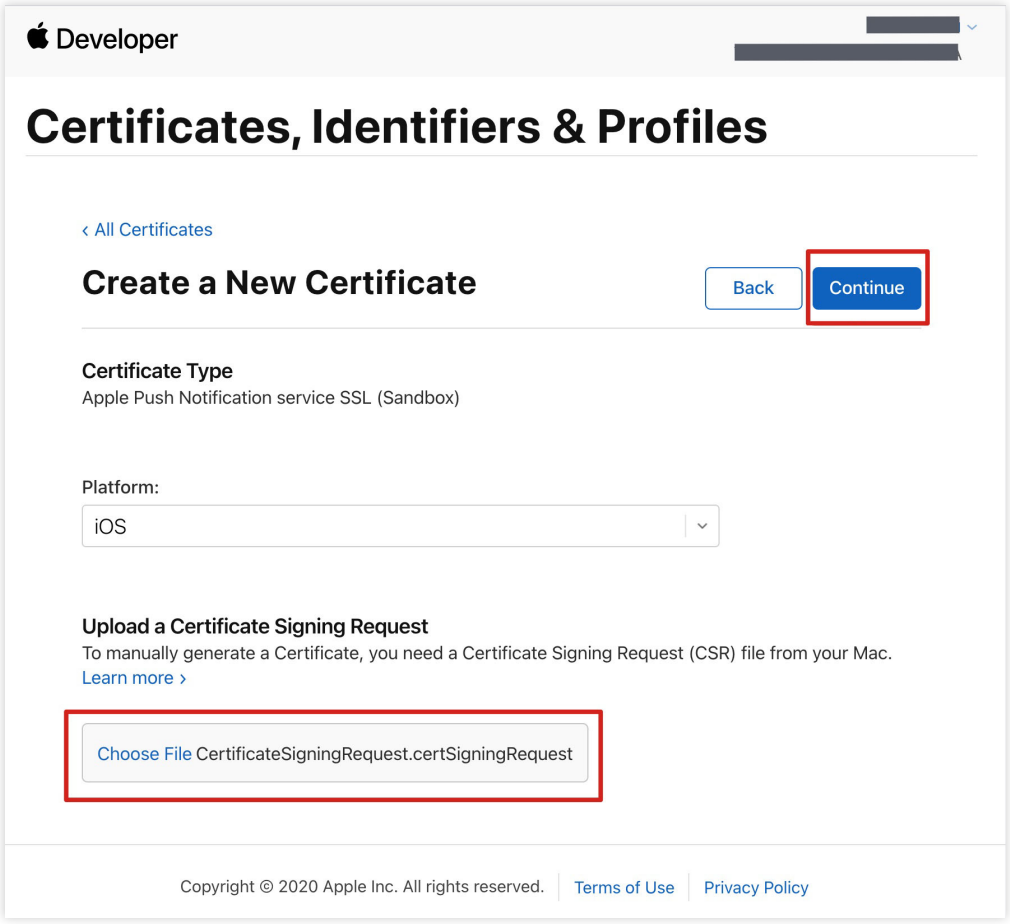
6. 返回上述 步骤3 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

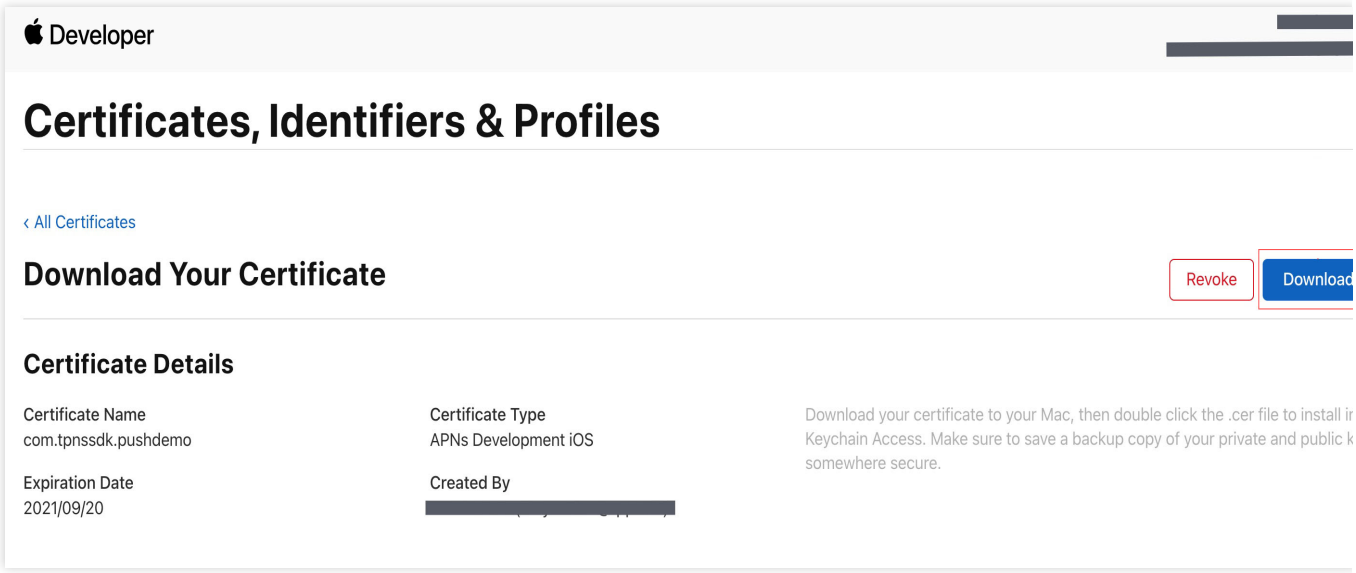


7. 单击 **Continue**，即可生成推送证书。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

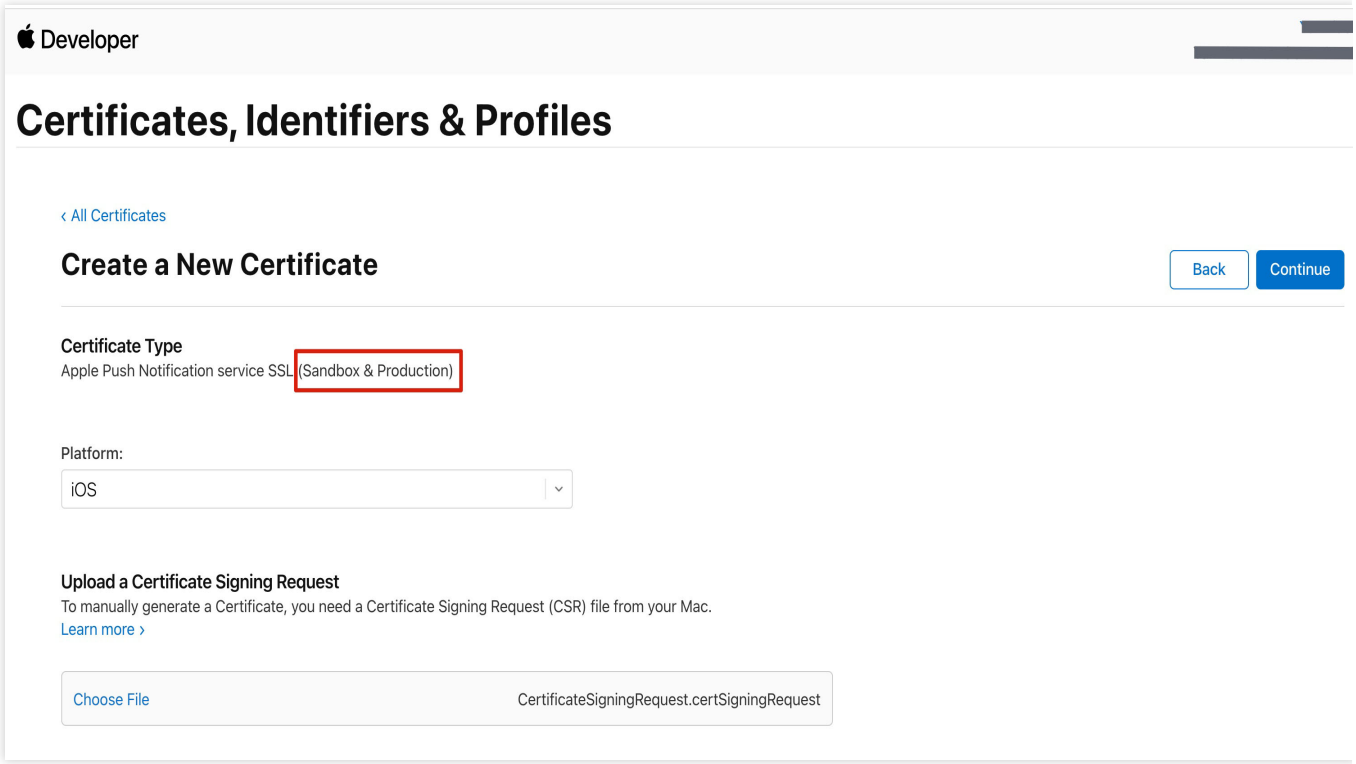


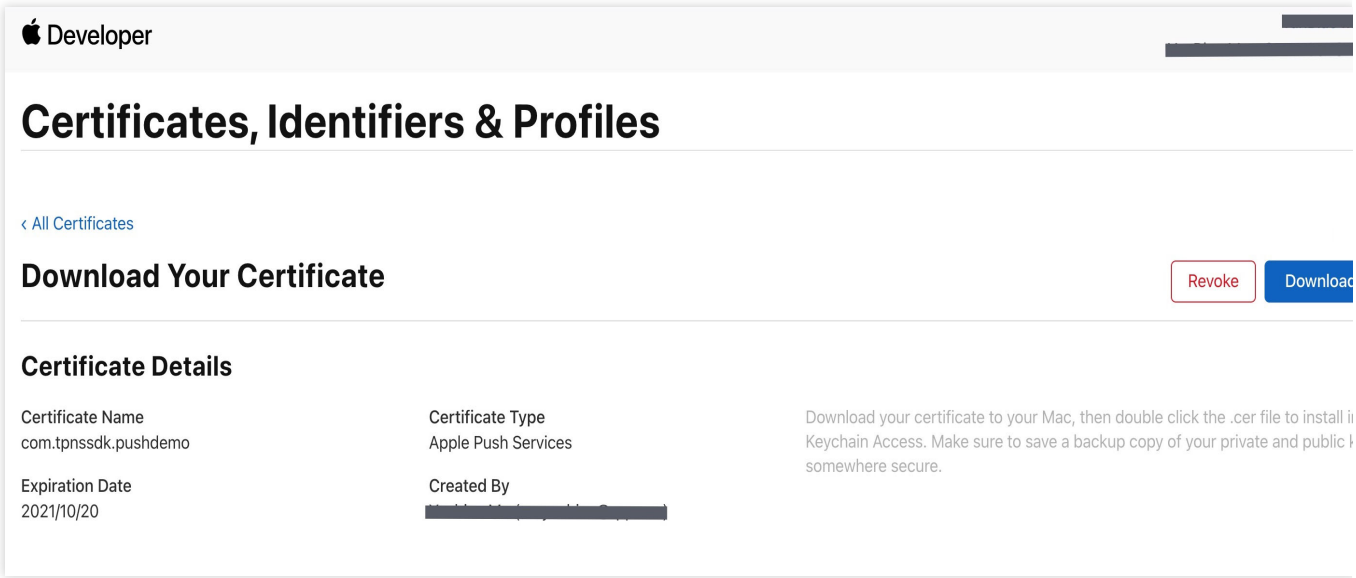
9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

说明

©2013-2025 Tencent Cloud International Pte. Ltd.

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





10. 双击打开下载的开发环境和生产环境的 `SSL Certificate`，系统会将其导入钥匙串中。
11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（`Apple Development IOS Push Service`）和生产环境（`Apple Push Services`）的 `p12` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意

保存 `.p12` 文件时，请务必为其设置密码。

步骤2：上传证书到控制台

1. 登录 即时通信 Chat 控制台。
2. 进入 **推送设置 > 厂商管理 > iOS**。
3. 点击**添加证书**。
4. 选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击**确认**。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意：

上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

上传证书需要设置密码，无密码收不到推送。

发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。

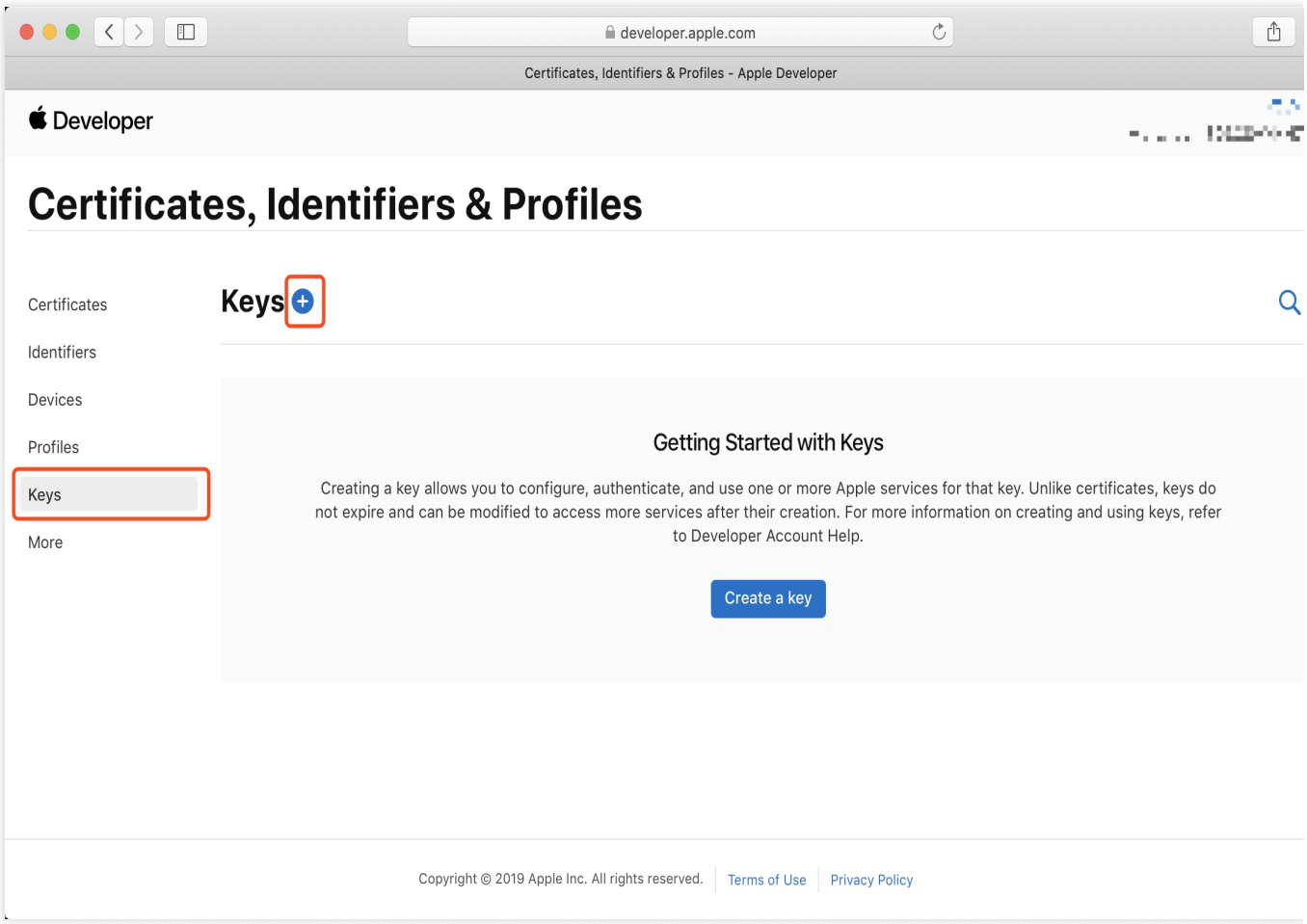
二、使用 p8 证书 （支持灵动岛推送）

p8 证书：p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 p8 证书可以简化证书管理，因为您可以使用一个p8 证书为多个应用程序提供推送通知服务。另外，p8 证书支持灵动岛推送。

步骤1：申请 APNs 证书

要创建 p8 证书文件，首先需要登录 苹果开发者中心。

©2013-2025 Tencent Cloud International Pte. Ltd.

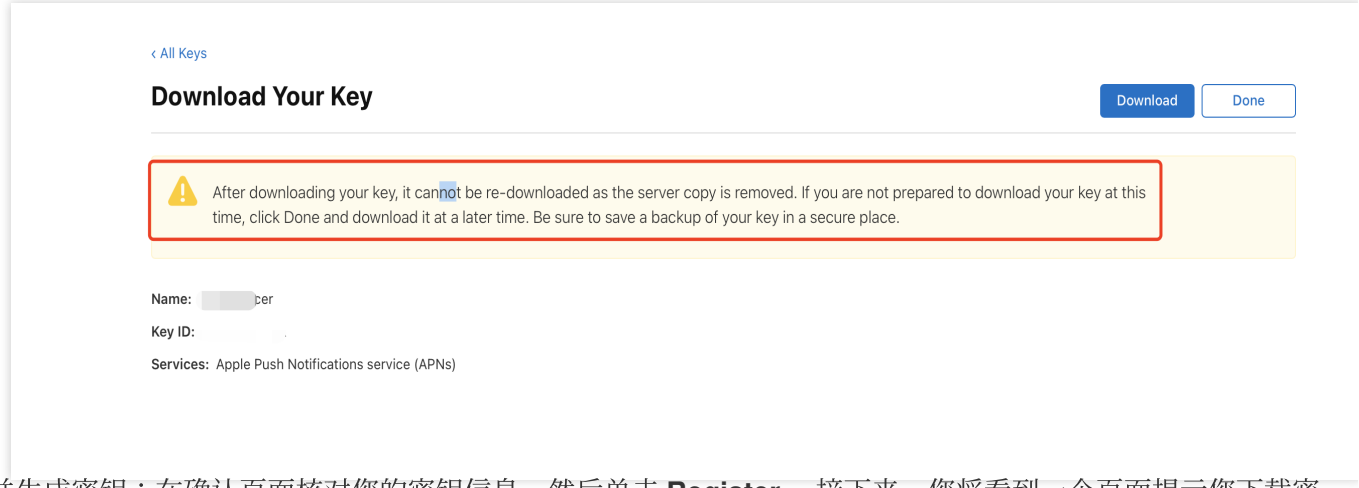


1. 进入Certificates，Identifiers & Profiles：在页面右上角单击 **Account**，然后在下拉菜单中选择 **Certificates, Identifiers & Profiles** 。

2. 创建一个新的 App ID：在左侧菜单中单击 **Identifiers** ，然后单击右侧的 **+** 创建一个新的 App ID。填写相应的信息并单击 **Continue** 。

3. 创建一个新的密钥：在左侧菜单中单击 **Keys**，然后单击右侧的 **+** 创建一个新的密钥。输入密钥的名称，然后勾选 **Apple Push Notifications service (APNs)**，单击 **Continue** 。

©2013-2025 Tencent Cloud International Pte. Ltd.



确认并生成密钥：在确认页面核对您的密钥信息，然后单击 **Register** 。接下来，您将看到一个页面提示您下载密钥。单击 **Download**，将生成的 .p8 文件保存到您的计算机上。

注意：

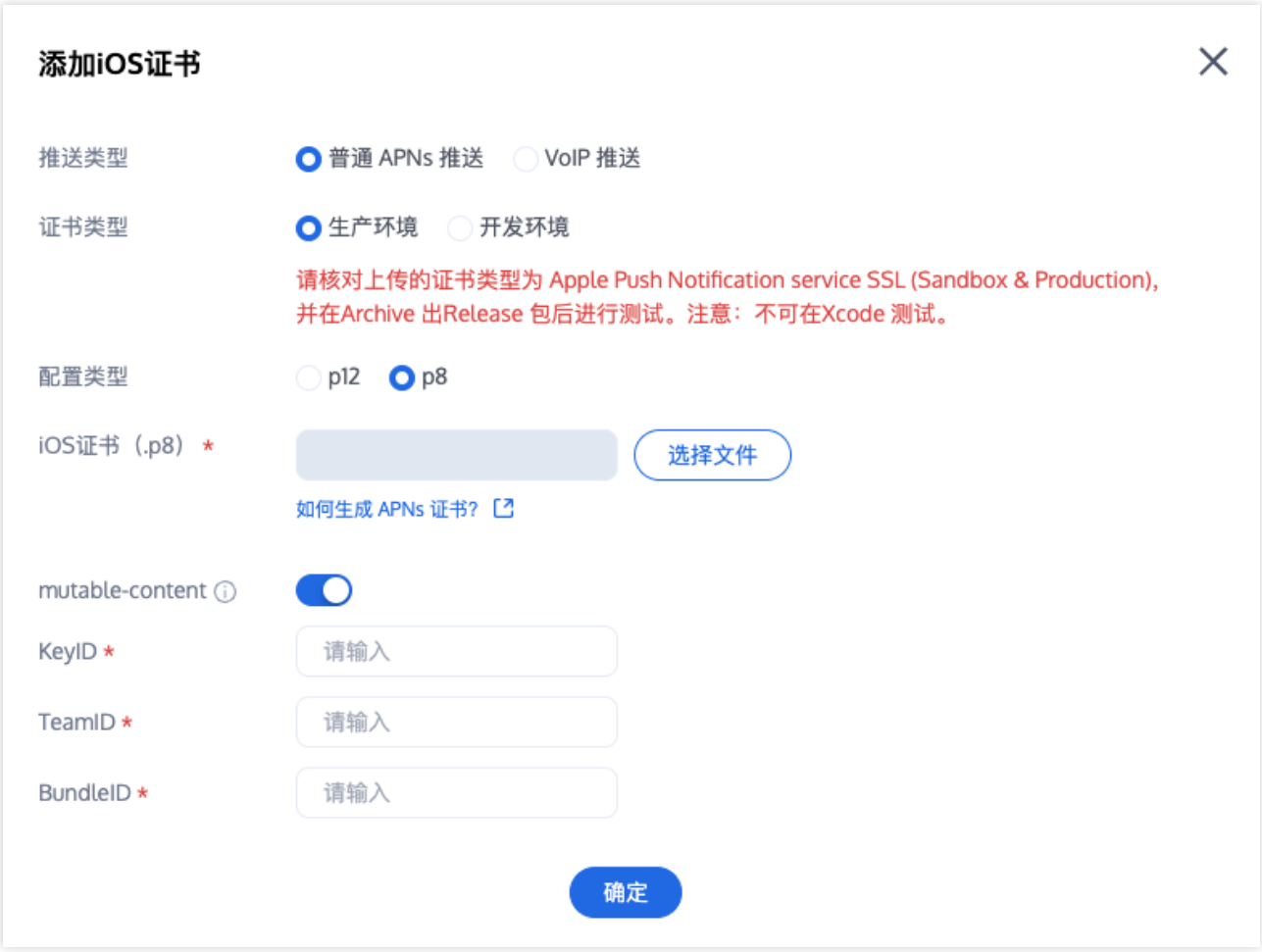
p8 证书只可以下载一次，请妥善保存。

请妥善保管下载的 p8 文件，因为您将无法再次下载该文件。您可以使用此P8证书配置您的iOS应用程序以接收推送通知。

步骤2：上传 p8 证书到Chat控制台

1. 登录 即时通信 Chat 控制台。
2. 进入 **推送设置 > 厂商管理 > iOS**。
3. 点击**添加证书**。
4. 选择 .p8 证书，上传 iOS 证书（.p8），设置 **KeyID、TeamID、BundleID**，单击**确认**。

©2013-2025 Tencent Cloud International Pte. Ltd.



说明：

KeyID：这是您的 APNs Auth Key 的唯一标识符。当您在 Apple Developer Center 创建一个新的 APNs Auth Key 时，系统会为您生成一个 Key ID。您可以在 "Certificates, Identifiers & Profiles" 部分的 "Keys" 中找到它。

TeamID：这是您的开发者账户的唯一标识符。您可以在 Apple Developer Center 的账户详情页面找到它。点击右上角的 "Membership"，在 "Membership Details" 部分可以找到您的 Team ID。

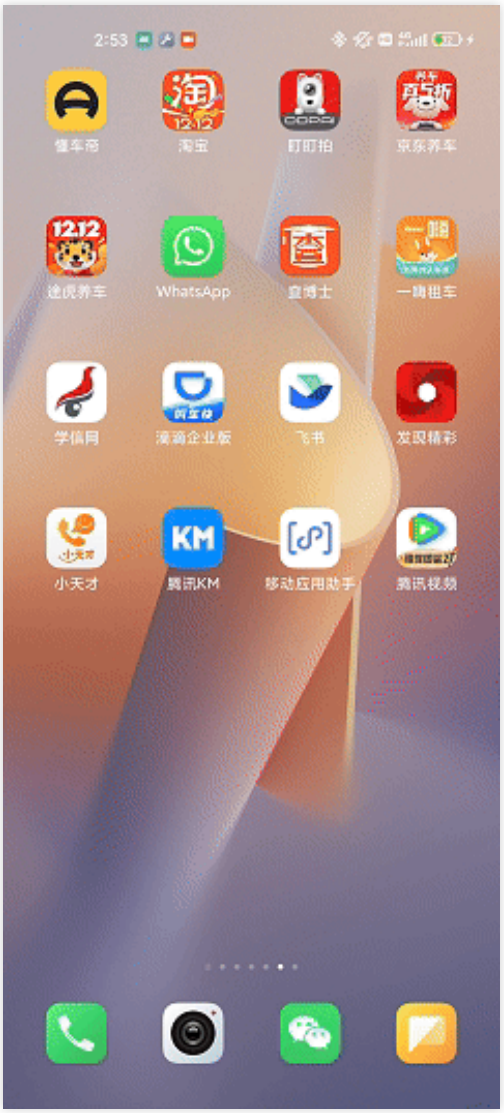
BundleID：这是您的应用程序的唯一标识符，也称为应用程序 ID。您可以在 Apple Developer Center 的 "Certificates, Identifiers & Profiles" 部分找到它。选择 "Identifiers"，然后在您的应用程序列表中找到对应的 Bundle ID。

©2013-2025 Tencent Cloud International Pte. Ltd.

快速接入

最近更新时间：2025-01-02 11:38:44

效果展示

小米 11 Pro（与 CallKit 融合）	iPhone 13
	

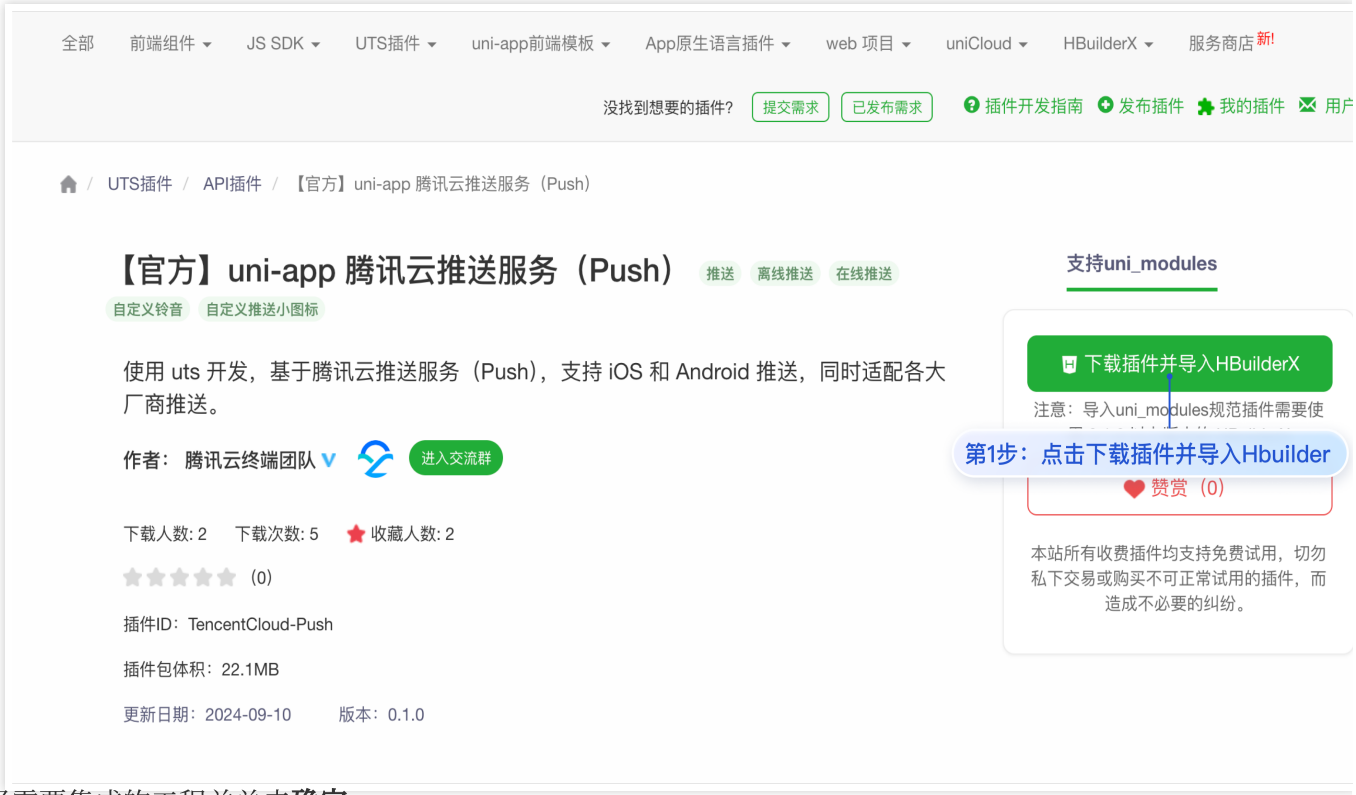
集成 TencentCloud-Push

HBuilderX 4.29 有 bug，请使用 HBuilderX 4.36 或更高版本，并升级 uni-app 腾讯云推送服务（Push）到 1.1.0 或更高版本。

步骤1：下载插件并导入 HBuilderX

©2013-2025 Tencent Cloud International Pte. Ltd.

1. 打开 uni-app 腾讯云推送服务（Push），单击**下载插件并导入HBuilderX**，将插件导入 HBuilderX 工程中。

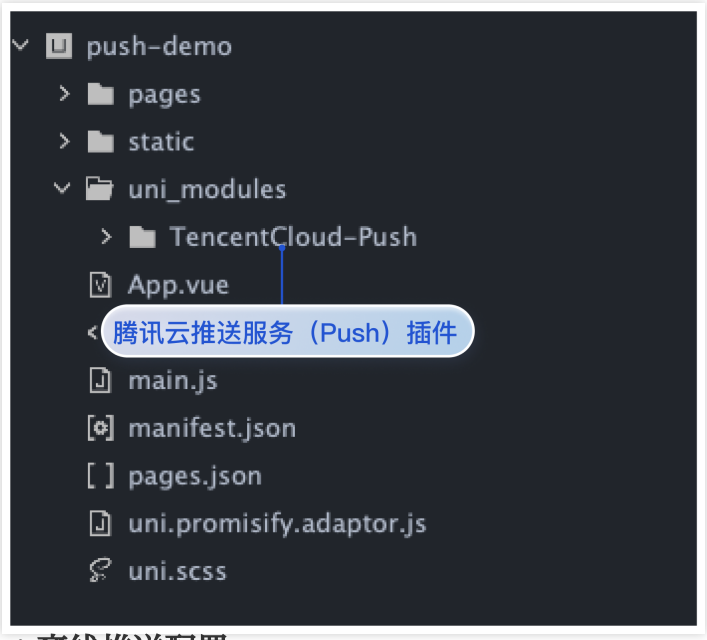


2. 选择需要集成的工程并单击**确定**。



3. 集成后效果如下图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤2：离线推送配置

说明：

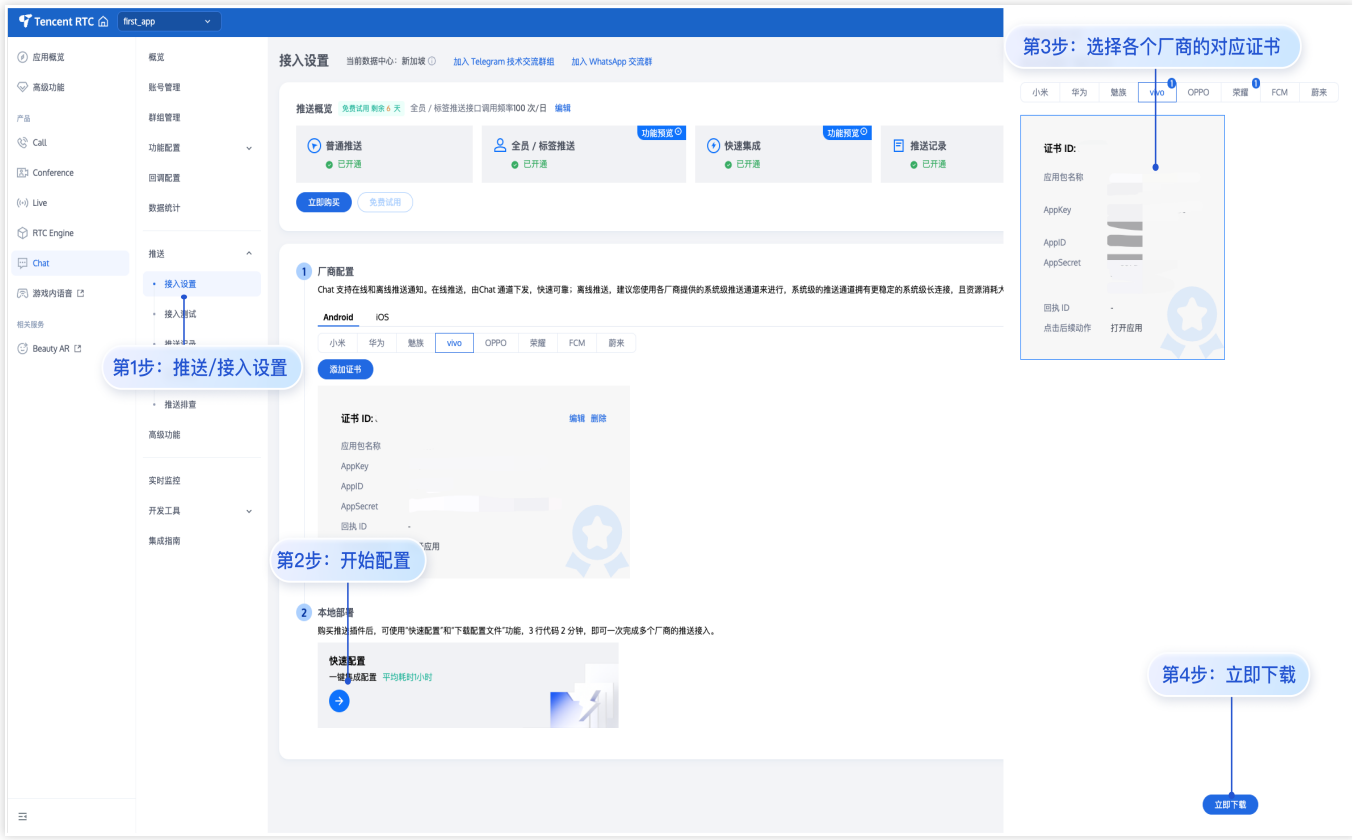
1. HBuilderX 4.36 发布了不向下兼容的更新，如果您使用的是 HBuilderX 4.36 或者更高版本，且需要 vivo/荣耀 的厂商推送，请升级推送版本到 1.1.0 或更高版本，并参考文档正确配置 `manifestPlaceholders.json` 和 `mcs-services.json` 。
2. 您需在 `nativeResources` 目录下进行推送配置。若项目根目录尚未创建该文件夹，请新建一个名为 `nativeResources` 的文件夹。
3. 确保您用 HBuilderX 打开的项目中 `nativeResources` 目录存在，且与 `uni_modules` 目录平级。

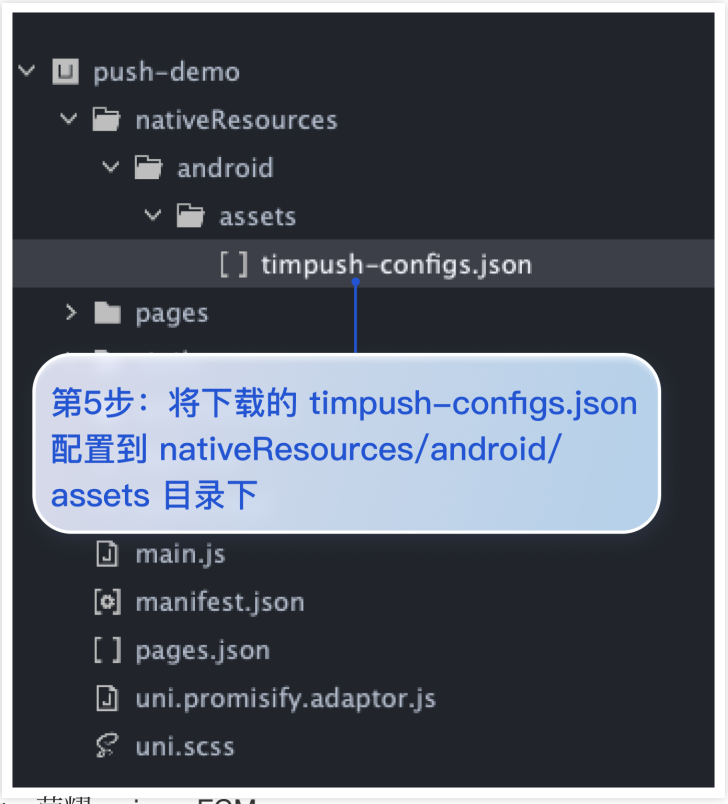
Android

iOS

1. 新建 `nativeResources/android/assets` 目录。
2. 配置 `timpush-configs.json` （在 推送 > 接入设置 > 一键式快速配置 下载），到 `nativeResources/android/assets/` 目录下。如图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.





3. 华为、荣耀、vivo、FCM。

FCM

华为

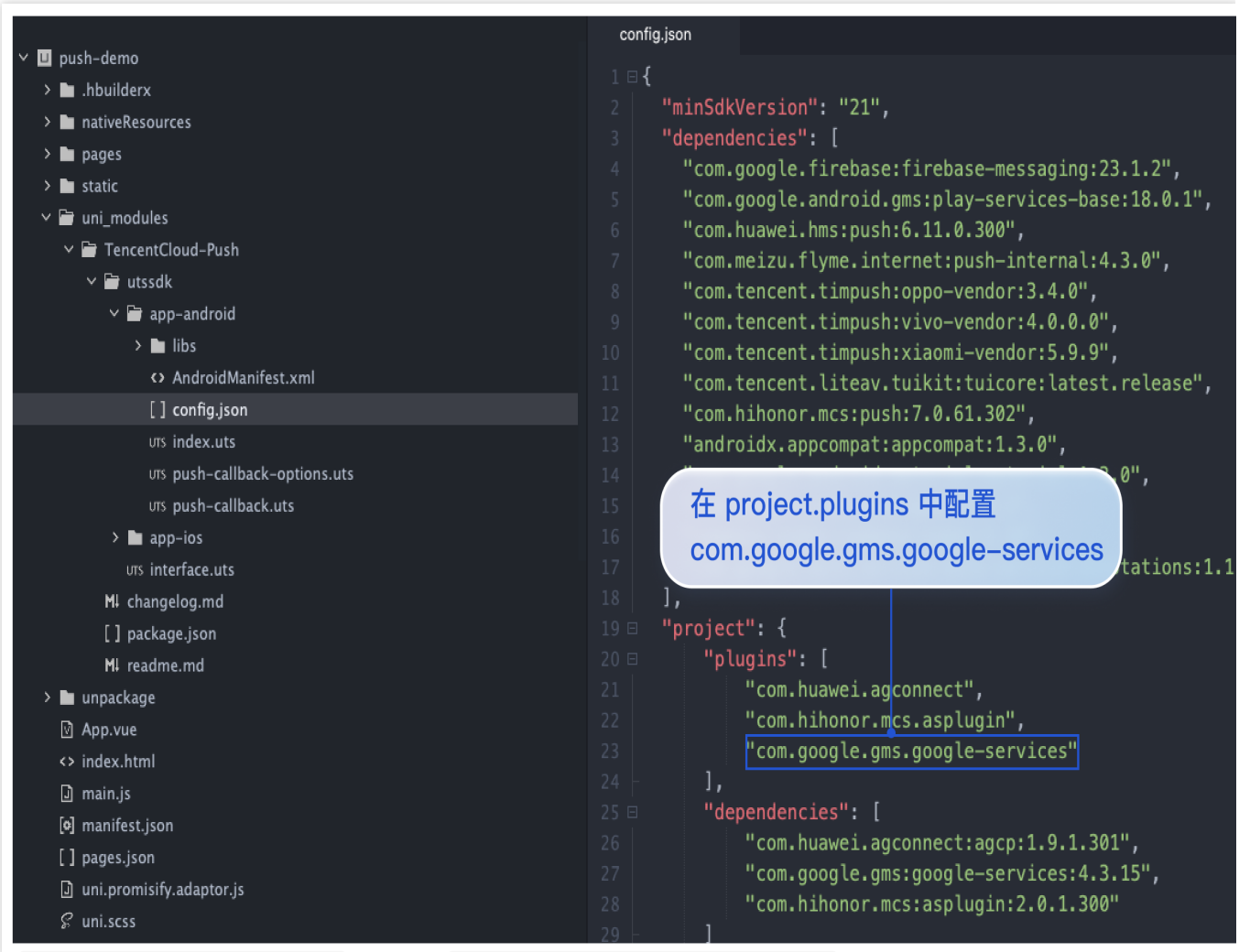
©2013-2025 Tencent Cloud International Pte. Ltd.

荣耀

vivo

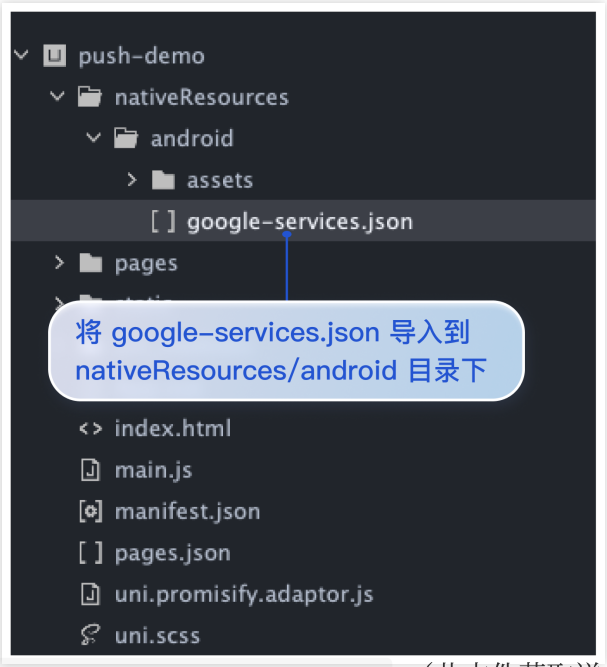
1. 配置 `com.google.gms.google-services` 到 `uni_modules/TencentCloud-Push/utssdk/app-android/config.json` 的 `project.plugins` 中，如下所示：

```
"project": {
        "plugins": [
        ...
                "com.google.gms.google-services"
        ],
        "dependencies": [
                "com.huawei.agconnect:agcp:1.9.1.301",
                "com.google.gms:google-services:4.3.15",
                "com.hihonor.mcs:asplugin:2.0.1.300"
        ]
}
```



2. 配置 `google-services.json` 文件到 **`nativeResources/android/`** 目录下（注意！请勿配置到 `nativeResources/android/asstes` 目录下）。如图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.



配置 `agconnect-services.json` （此文件获取详见 厂商配置 > uniapp > 华为 > 步骤4：获取应用信息）到 **`nativeResources/android/assets/`** 目录下。如图所示：



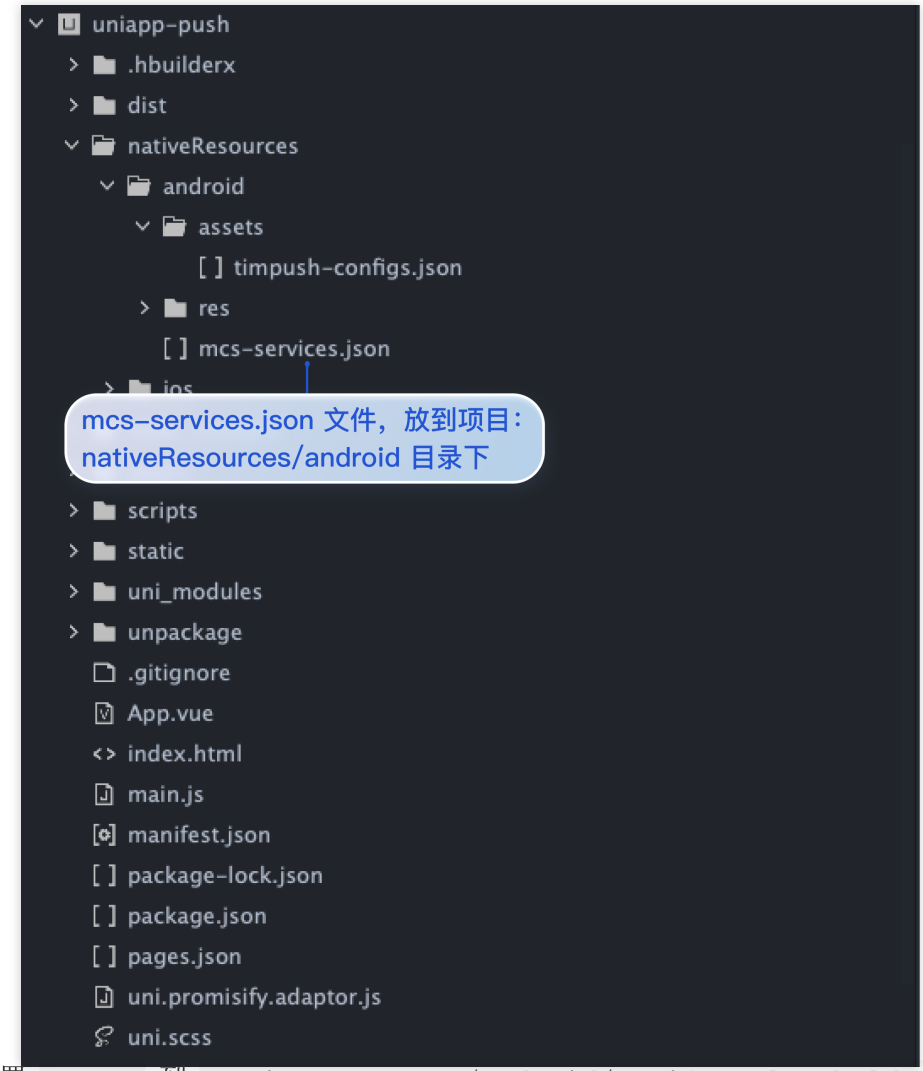
1. 编辑 `uni_modules/TencentCloud-Push/utssdk/app-android/config.json` 的 `dependencies` ，添加 `"com.tencent.timpush:honor:8.3.6498"` 。

```
{
    ...
```

©2013-2025 Tencent Cloud International Pte. Ltd.

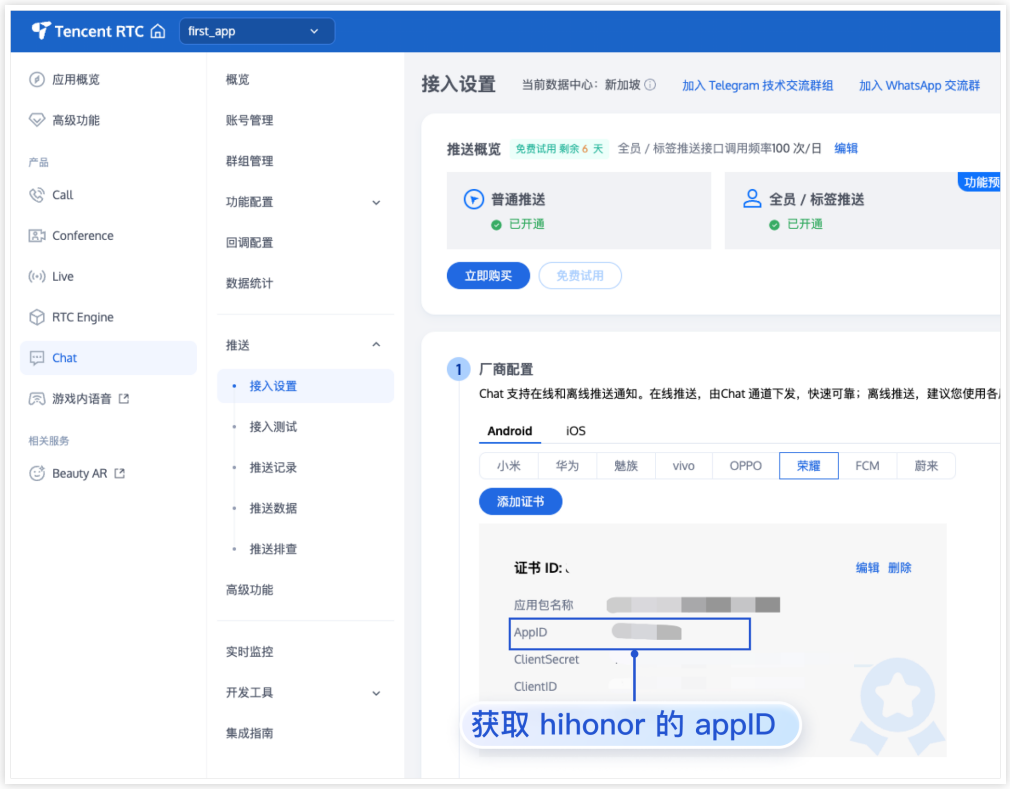
```
        "dependencies": [
            ...
            "com.tencent.timpush:honor:8.3.6498"
        ]
}
```

2. 配置 `mcs-services.json` 文件到 `nativeResources/android` （此文件获取详见 厂商配置 > uniapp > 荣耀 > 步骤3.2：进入应用详情， 绑定应用包名， 下载 mcs-services.json 文件）目录下。如图所示：



3. 配置 `appID` 到 `nativeResources/android/manifestPlaceholders.json` 中的 `"HONOR_APPID"` 。

©2013-2025 Tencent Cloud International Pte. Ltd.




```
{
      "HONOR_APPID": ""
}
```

1. 编辑 `uni_modules/TencentCloud-Push/utssdk/app-android/config.json` 的 `dependencies`，添加 `"com.tencent.timpush:vivo:8.3.6498"`。

```
{
      ...
      "dependencies": [
          ...
          "com.tencent.timpush:vivo:8.3.6498"
      ]
}
```

2. 配置 `appID` 和 `appKey` 到 `nativeResources/android/manifestPlaceholders.json` 中的 `VIVO_APPKEY` 和 `VIVO_APPID`。

©2013-2025 Tencent Cloud International Pte. Ltd.




```
{
      "VIVO_APPKEY": "",
      "VIVO_APPID": "",
}
```

1. 新建 `nativeResources/ios/Resources` 目录；
2. 在 `nativeResources/ios/Resources` 中新建 `timpush-configs.json` 文件；
3. 并将在 控制台 > 推送 > 接入设置 获取的证书ID，补充到 `timpush-configs.json` 文件中。如下所示：

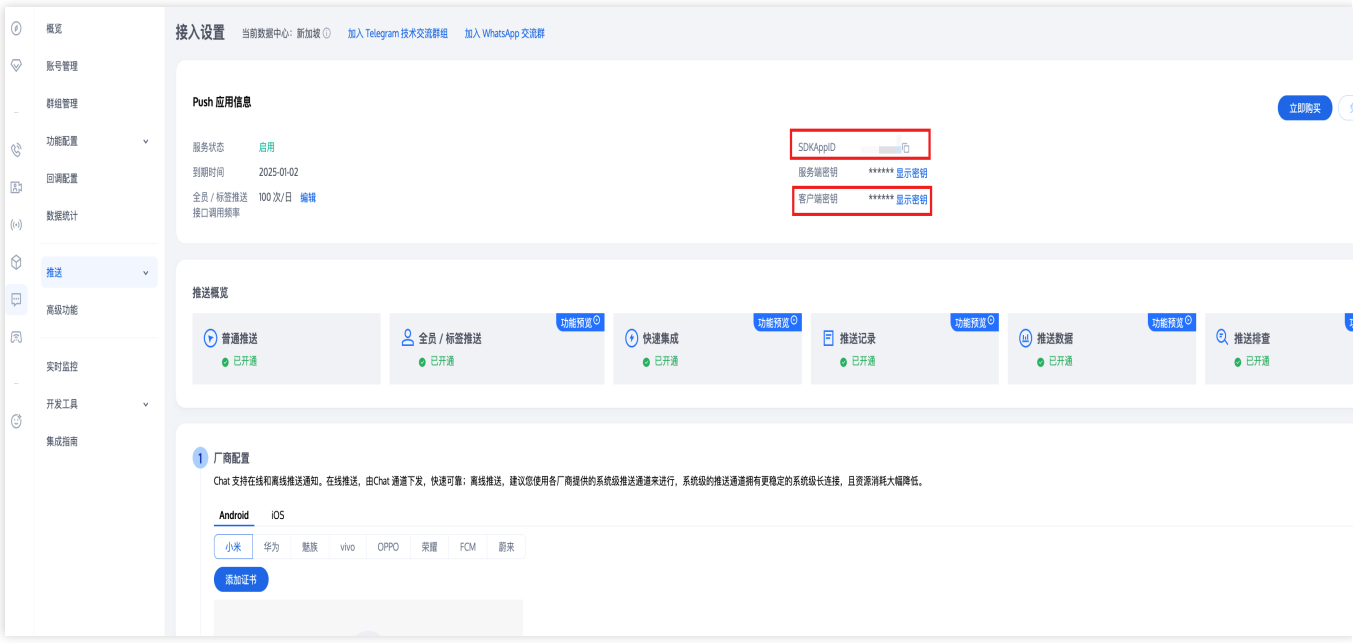
```
{
     "businessID":"xxx"
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤3：引入并注册腾讯云推送服务（Push）

将 `SDKAppID` 和 `appKey` 替换为您在 控制台 获取的应用的信息。如图所示：



©2013-2025 Tencent Cloud International Pte. Ltd.

```
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import * as Push from '@/uni_modules/TencentCloud-Push';

const SDKAppID = 0; // 您的 SDKAppID
const appKey = ''; // 客户端密钥
const userID = ''; // 您的 userID
const userSig = ''; // 您 userID 的密钥

let vueVersion = 2;
// #ifdef VUE3
vueVersion = 3;
// #endif

TUILogin.login({
  SDKAppID,
  userID,
  userSig,
  useUploadPlugin: true,
  framework: `vue${vueVersion}`,
}).then(() => {
  if (Push) {
    Push.setRegistrationID(userID, () => {
      console.log('setRegistrationID ok', userID);

      Push.registerPush(SDKAppID, appKey, (data) => {
          console.log('registerPush ok', data);
          Push.getRegistrationID((registrationID) => {
            console.log('getRegistrationID ok', registrationID);
          });
        }, (errCode, errMsg) => {
          console.error('registerPush failed', errCode, errMsg);
        }
      );
    });

    // 监听通知栏点击事件，获取推送扩展信息
    Push.addPushListener(Push.EVENT.NOTIFICATION_CLICKED, (res) => {
      console.log('notification clicked', res);
      // 解析扩展信息，跳转到相应的会话（代码仅供参考，发布前需要完善）
      try {
        const data = JSON.parse(res.data);
        const conv_type = data?.entity?.chatType === 1 ? 'C2C' : 'GROUP';
        // 根据推送信息拼的 conversationID
        const conversationID = `${conv_type}${data.entity.sender}`;
        // 切换会话
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        TUIConversationService.switchConversation(conversationID);
        const chatPath = '/TUIKit/components/TUIChat/index';
        uni.navigateTo({ url: chatPath });
      } catch (error) {
        console.log('error', error);
      }
    });

    // 监听在线推送
    Push.addPushListener(Push.EVENT.MESSAGE_RECEIVED, (res) => {
      // res 为消息内容
      console.log('message received', res);
    });

    // 监听在线推送被撤回
    Push.addPushListener(Push.EVENT.MESSAGE_REVOKED, (res) => {
      // res 为被撤回的消息 ID
      console.log('message revoked', res);
    });
  }
});
```

步骤4: 使用云端证书，生成自定义基座

单击 HBuilderX 的**运行 > 运行到手机或模拟器 > 制作自定义调试基座**，使用云端证书制作 Android 或 iOS 自定义调试基座。



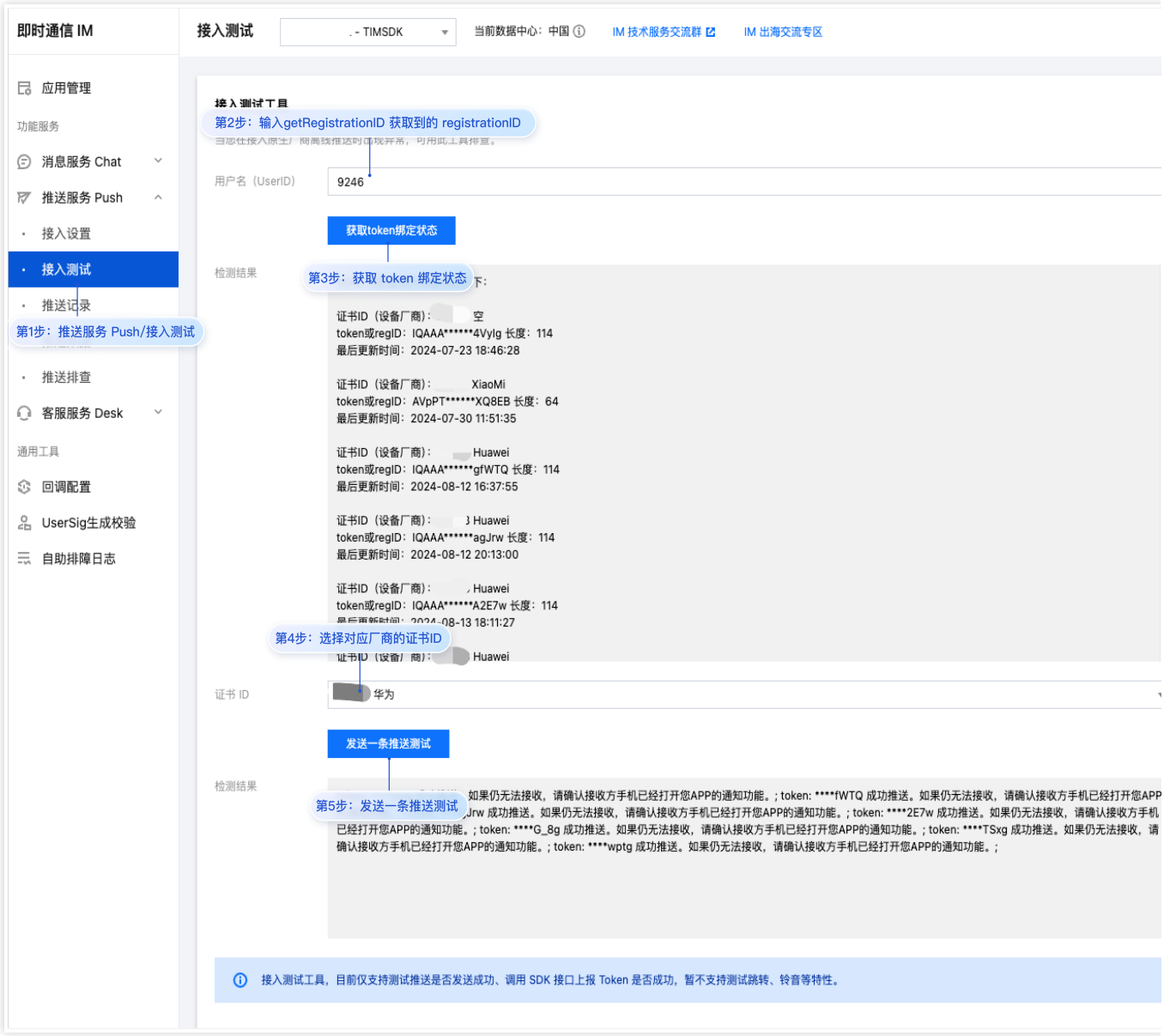
©2013-2025 Tencent Cloud International Pte. Ltd.



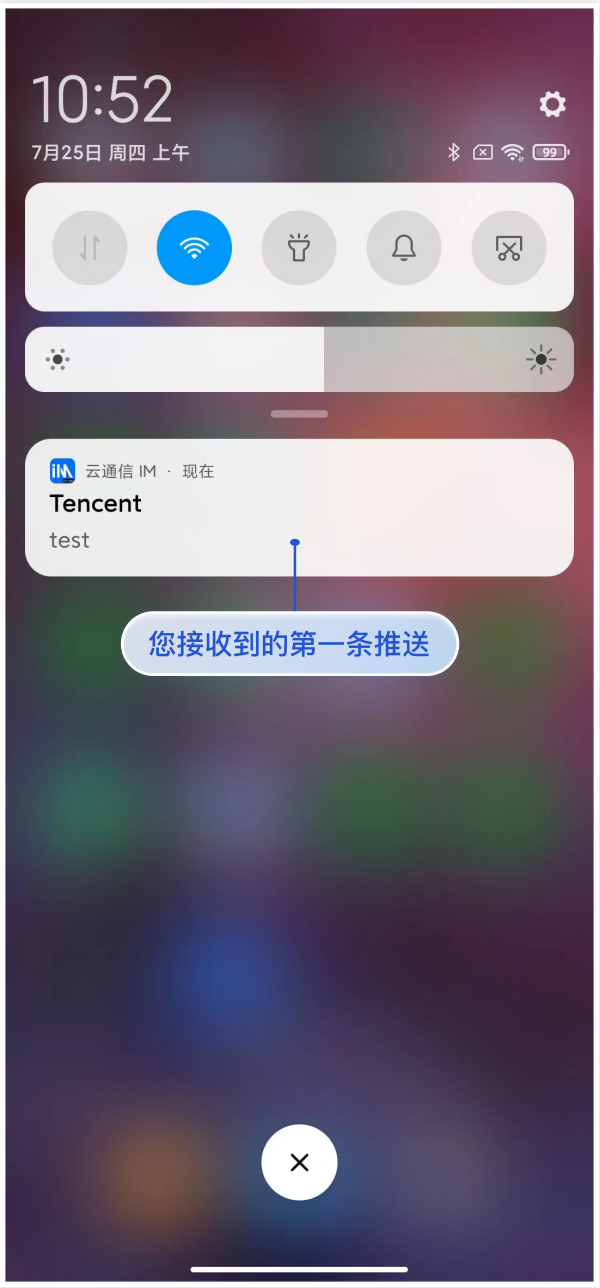
步骤5. 体验您的第一次推送

在测试推送前，请务必打开通知和状态栏。进入 推送服务 Push > 接入测试 发送您的第一条推送。

©2013-2025 Tencent Cloud International Pte. Ltd.



©2013-2025 Tencent Cloud International Pte. Ltd.



推送结果回调

开启推送服务后，推送结果可以通过配置基础回调的方式，将结果转发给 App 后台，详见：
普通推送结果回调
全员推送结果回调

设备通知栏设置

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”，会影响锁屏状态下推送通知显示。

©2013-2025 Tencent Cloud International Pte. Ltd.

“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”，会影响推送通知铃音的效果。



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，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
vivo
OPPO
小米
魅族
FCM

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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。
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。
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。
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。
推送消息数量有限制。 具体请参见 厂商描述。
推送上行消息频率有限制。 具体请参见厂商描述。

©2013-2025 Tencent Cloud International Pte. Ltd.

客户端 API

最近更新时间：2025-01-02 11:38:14

接口概览

API	描述
`registerPush`	注册推送服务，(必须在 App 用户同意了隐私政策后，再调用该接口使用推送服务)。
`unRegisterPush`	反注册关闭推送服务。
`setRegistrationID`	设置推送设备标识 ID。 RegistrationID 是推送接收设备的唯一标识 ID。默认情况下，注册推送服务成功时自动生成该 ID，同时也支持您自定义设置。**请注意！如果您调用此接口，请务必在 `registerPush` 前调用。** 适用场景举例： 若您同时集成了消息服务（Chat）和推送服务（Push），在某个设备上，用户登录 Chat 的 userID 假设为 "user123"，如果您想指定向 "user123" 推送消息，则需要调用此接口设置设备标识 ID，如下： `Push.setRegistrationID("user123", () => {});`
`getRegistrationID`	在成功注册推送服务后，调用此接口可获取推送接收设备的唯一标识 ID，即 RegistrationID。
`getNotificationExtInfo`	收到离线推送时，点击通知栏拉起 App，调用此接口可获取推送扩展信息。
`addPushListener`	添加 Push 监听器。
`removePushListener`	移除 Push 监听器。
`disablePostNotificationInForeground`	应用在前台时，开/关通知栏通知（默认开）。
`createNotificationChannel`	创建客户端通知 channel。此接口可在 Android 平台上实现自定义铃音功能。

接口详情

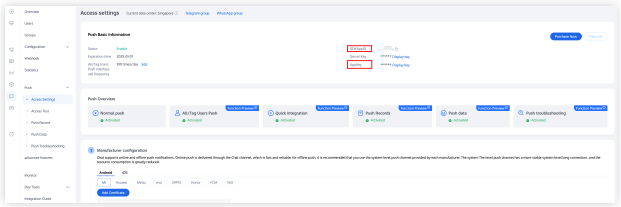
©2013-2025 Tencent Cloud International Pte. Ltd.

注册推送服务

接口

```
registerPush(SDKAppID, appKey, onSuccess, onError)
```

参数说明：

参数	类型	说明	获取路径
SDKAppID	Number	推送服务 Push 的 SDKAppID。	Chat 控制台 
appKey	String	推送服务 Push 的客户端密钥。	
onSuccess	Function	注册推送成功的回调。	-
onError	Function | undefined	注册推送失败的回调。	-

反注册关闭推送服务

接口

```
unRegisterPush(onSuccess, onError)
```

参数说明：

参数	类型	说明
onSuccess	Function	反注册推送成功的回调。
onError	Function | undefined	反注册推送失败的回调。

设置推送 ID 标识 RegistrationID

说明：

©2013-2025 Tencent Cloud International Pte. Ltd.

1. 如果您调用此接口，请务必在 `registerPush` 前调用。

接口

```
setRegistrationID(registrationID, onSuccess)
```

参数说明：

参数	类型	说明
registrationID	String	自定义的推送 ID 标识。
onSuccess	Function	设置自定义推送 ID 标识成功后的回调。

获取推送 ID 标识 RegistrationID

说明：

1. 若您调用过 `setRegistrationID` 接口设置标识 ID，此接口将返回您设置的标识 ID，否则返回由 Push SDK 生成的随机值。

接口

```
getRegistrationID(onSuccess)
```

参数说明：

参数	类型	说明
onSuccess	Function	获取推送 ID 标识成功的回调。

获取推送扩展信息

接口

```
getNotificationExtInfo(onSuccess)
```

参数说明：

参数	类型	说明
onSuccess	Function	获取点击透传的内容成功的回调。

添加 Push 监听器

©2013-2025 Tencent Cloud International Pte. Ltd.

接口

```
addPushListener(eventName: string, listener: (data: any) => void);
```

参数说明：

参数	类型	说明
eventName	String	推送事件类型。
listener	Function	推送事件处理方法。

移除 Push 监听器

接口

```
removePushListener(eventName: string, listener?: (data: any) => void);
```

参数说明：

参数	类型	说明
eventName	String	推送事件类型。
listener	Function | undefined	推送事件处理方法。

应用在前台时，开/关通知栏通知

接口

```
disablePostNotificationInForeground(disable: boolean);
```

参数说明：

参数	类型	说明
disable	boolean	应用在前台时，开/关通知栏通知，默认开： true: 应用在前台时，关闭通知栏通知。 false: 应用在前台时，开启通知栏通知。

创建客户端通知 channel

©2013-2025 Tencent Cloud International Pte. Ltd.

接口

```
createNotificationChannel(options: any, onSuccess: (data: any) => void);
```

参数说明：

参数	类型	说明
options.channelID	String	自定义 channel 的 ID。 OPPO：控制台-> 接入配置 中 channelID。
options.channelName	String	自定义 channel 的名称。
options.channelDesc	String | undefined	自定义 channel 的描述。
options.channelSound	String | undefined	自定义 channel 的铃音，音频文件名，不带后缀，音频文件需要放到 `xxx/nativeResources/android/res/raw` 中。 例如： `options.channelSound = private_ring`，即设置 `xxx/nativeResources/android/res/raw/private_ring.mp3` 为自定义铃音
onSuccess	Function	接口调用成功的回调函数。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter
厂商配置

最近更新时间：2025-03-03 14:30:48

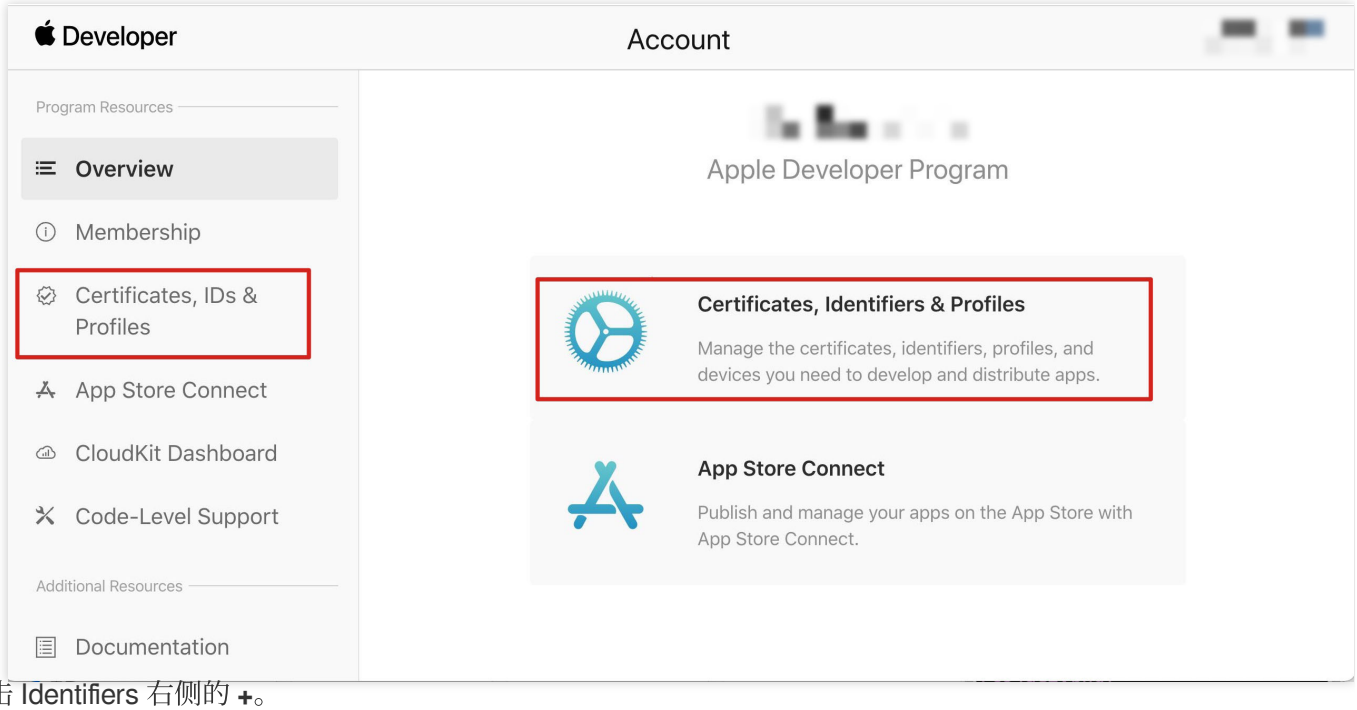
目前消息推送插件, 在 Flutter 使用中, 仅支持推送至 Android (含各厂商通道) 和 iOS 设备.
iOS
Android
集成 消息推送 插件之前，需要先向 Apple 申请 APNs 推送证书，然后上传推送证书到 Chat 控制台 。之后按照快速接入步骤接入即可。

操作步骤

步骤1：申请 APNs 证书

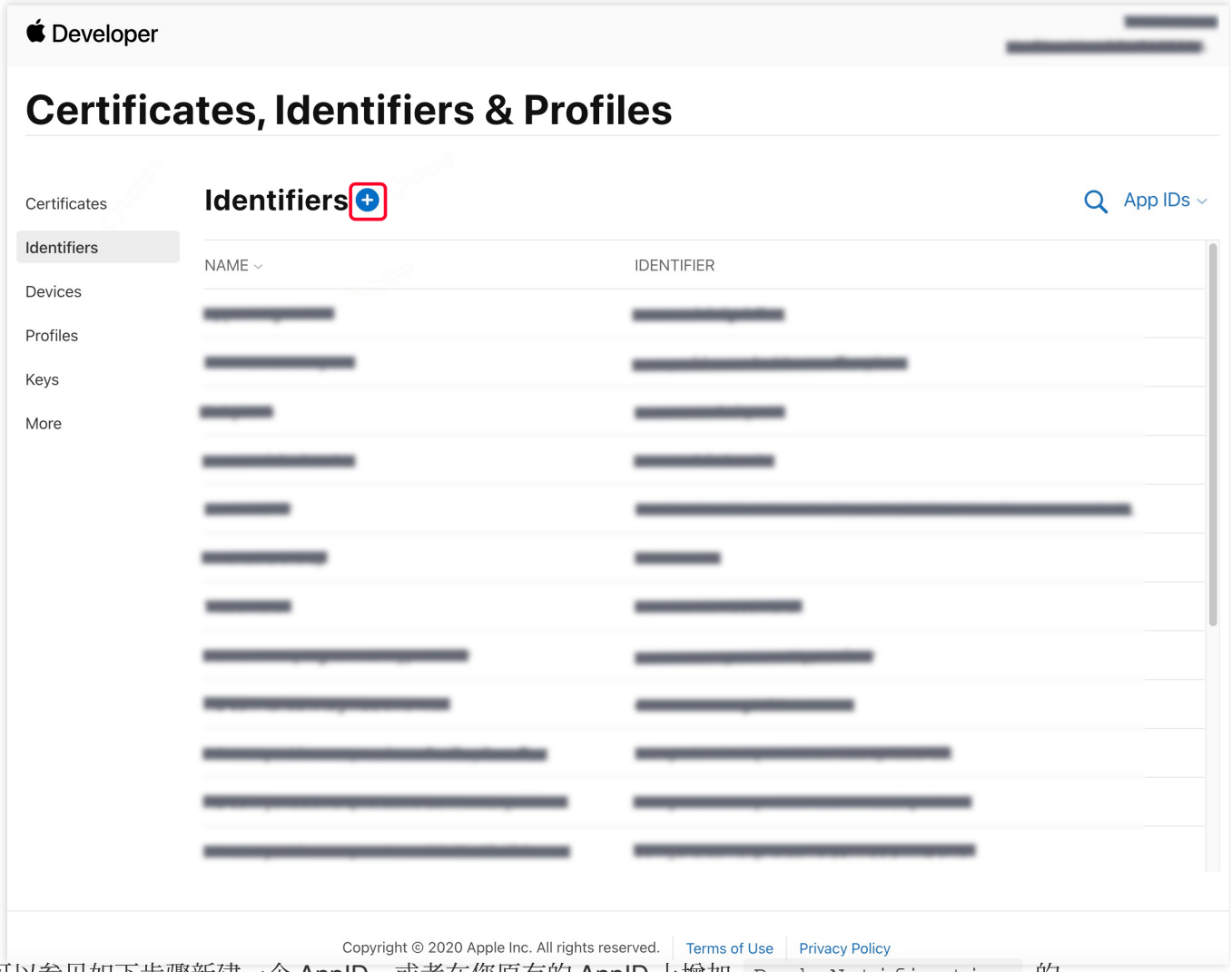
开启 App 远程推送

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



2. 单击 Identifiers 右侧的 +。

©2013-2025 Tencent Cloud International Pte. Ltd.



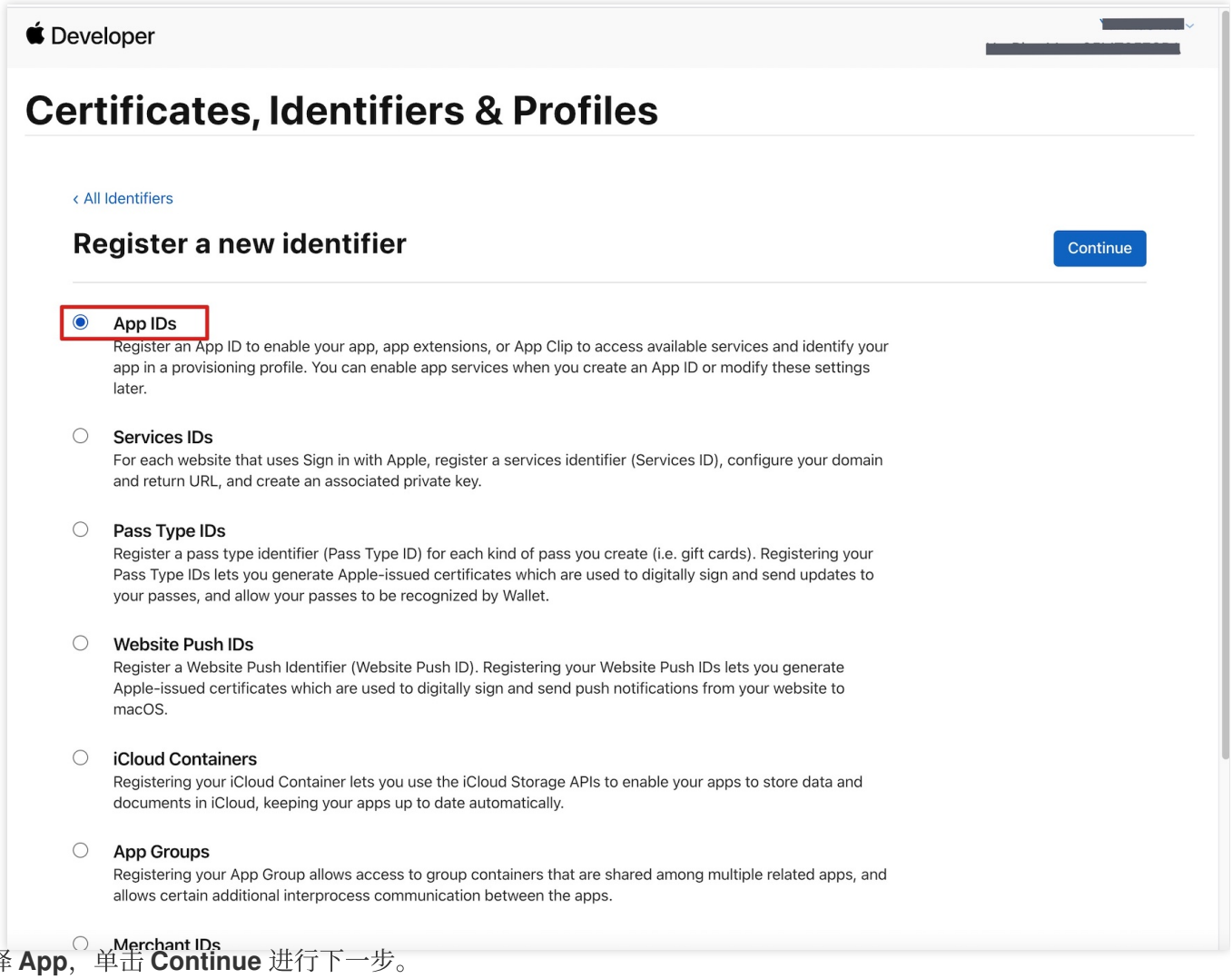
3. 您可以参见如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service` 。

说明

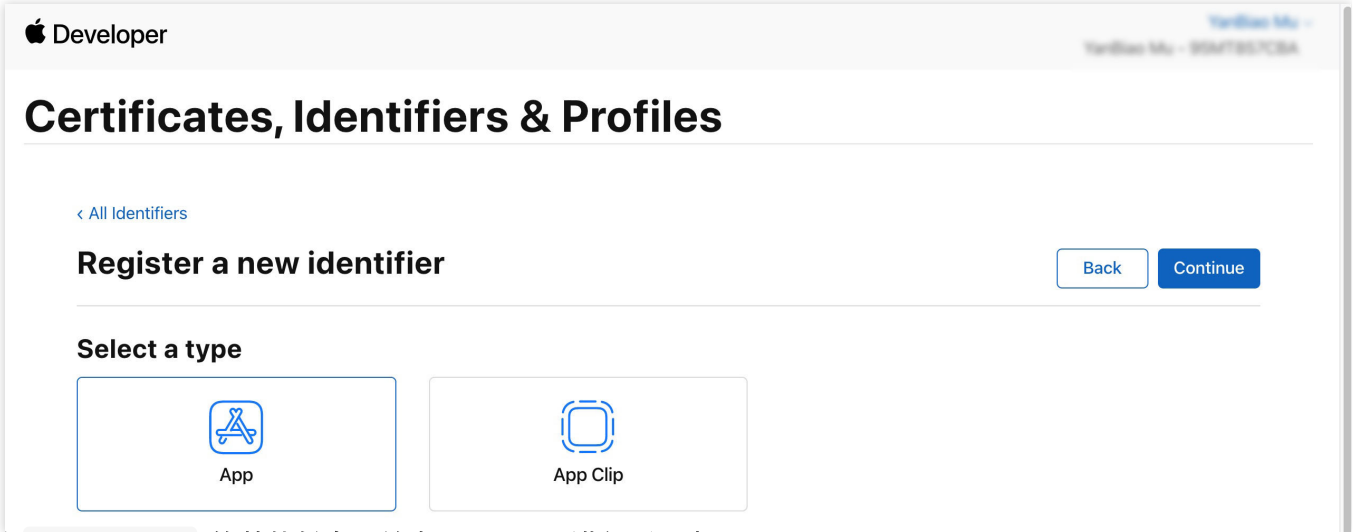
您 App 的 `Bundle ID` 不能使用通配符 `*` ，否则将无法使用远程推送服务。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

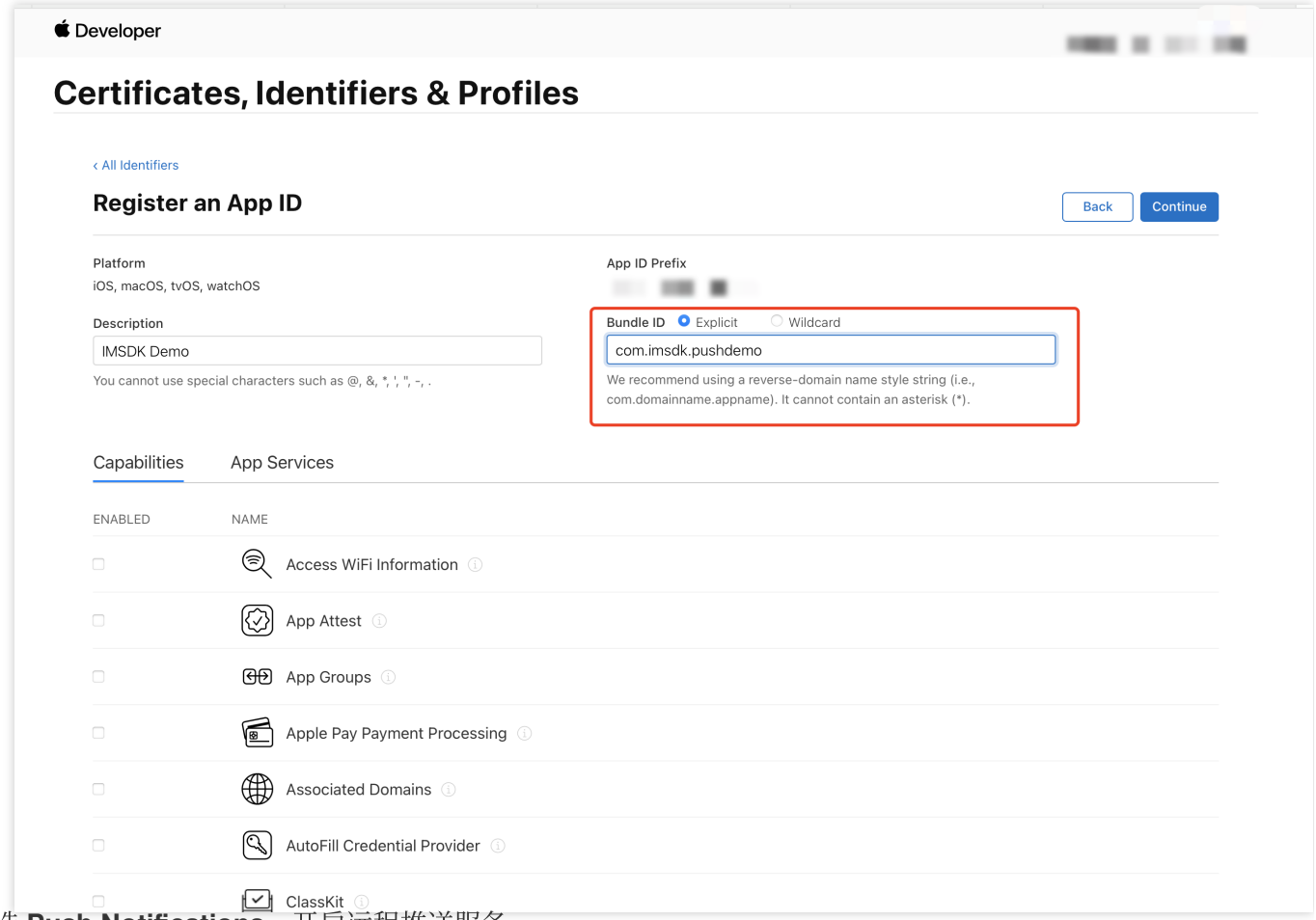


5. 选择 **App**，单击 **Continue** 进行下一步。



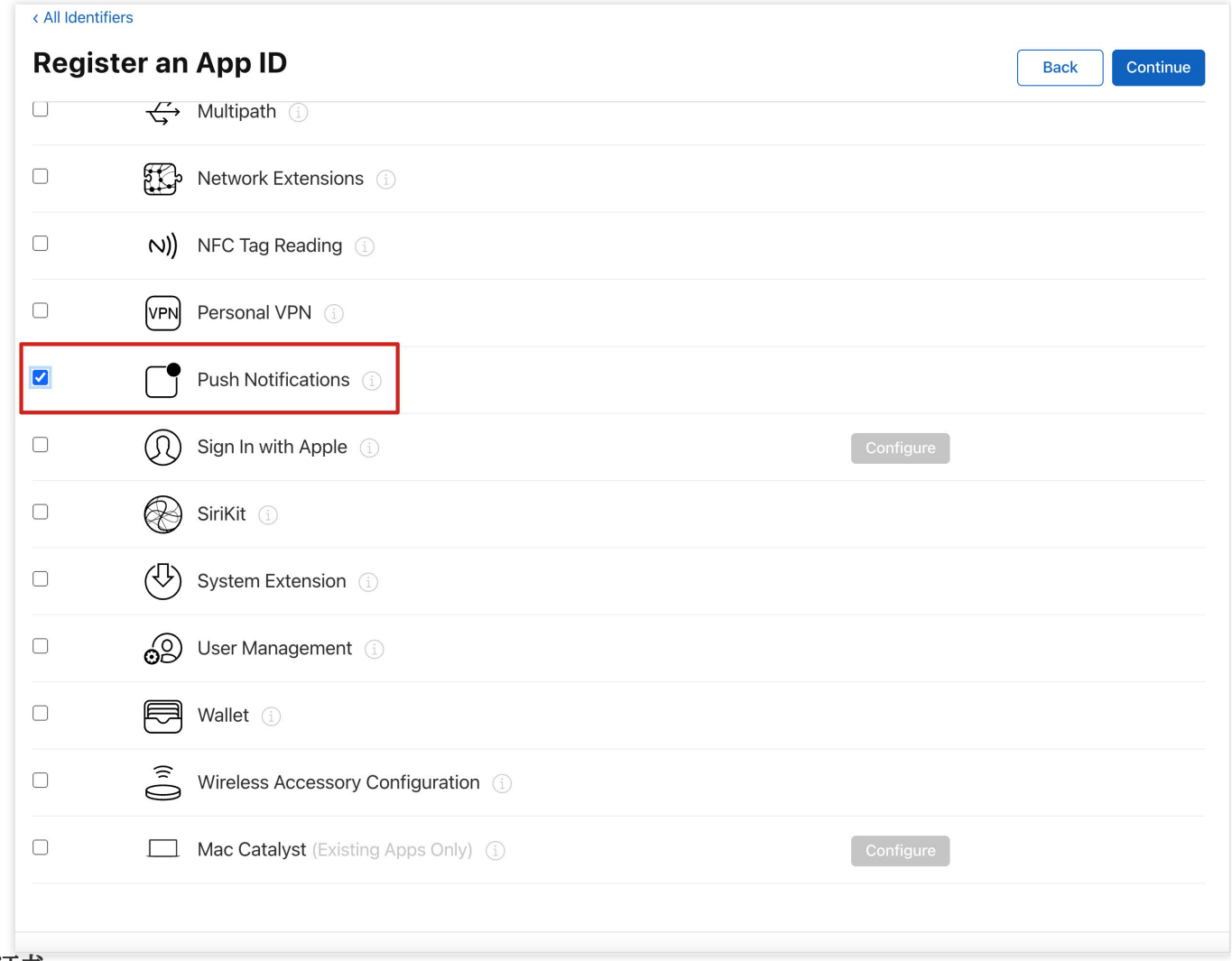
6. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

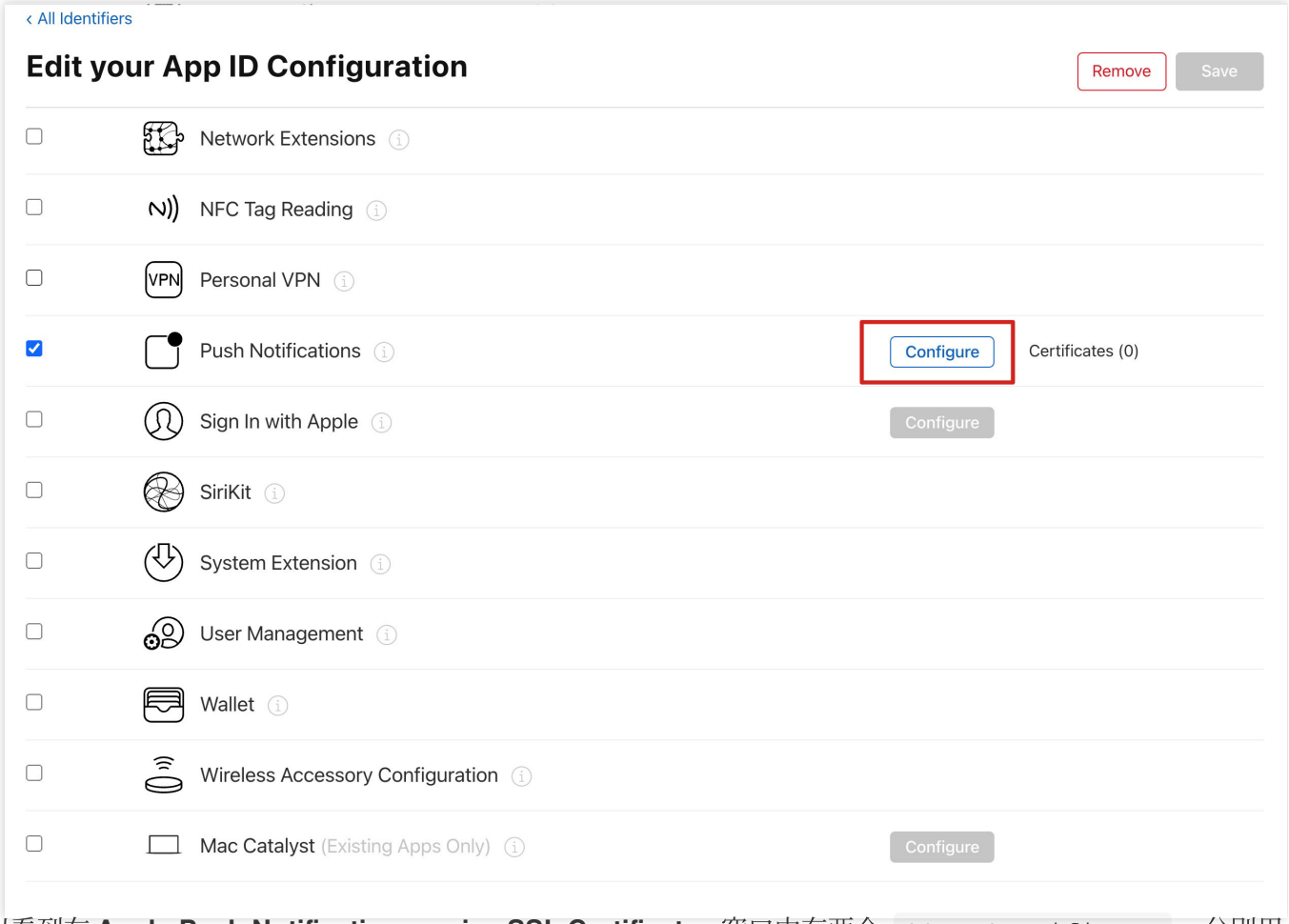
©2013-2025 Tencent Cloud International Pte. Ltd.



生成证书

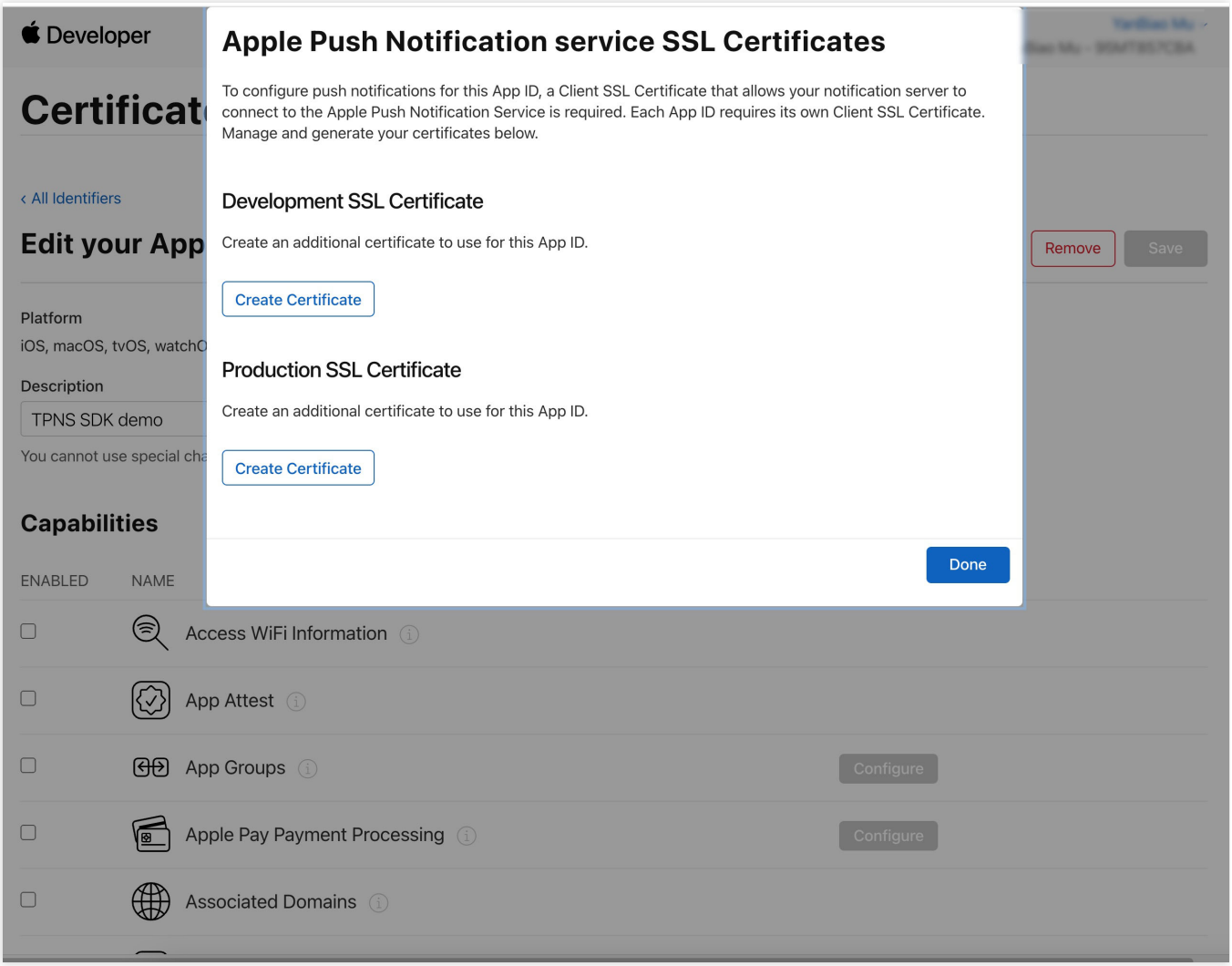
1. 选中您的 AppID，选择 **Configure**。

©2013-2025 Tencent Cloud International Pte. Ltd.



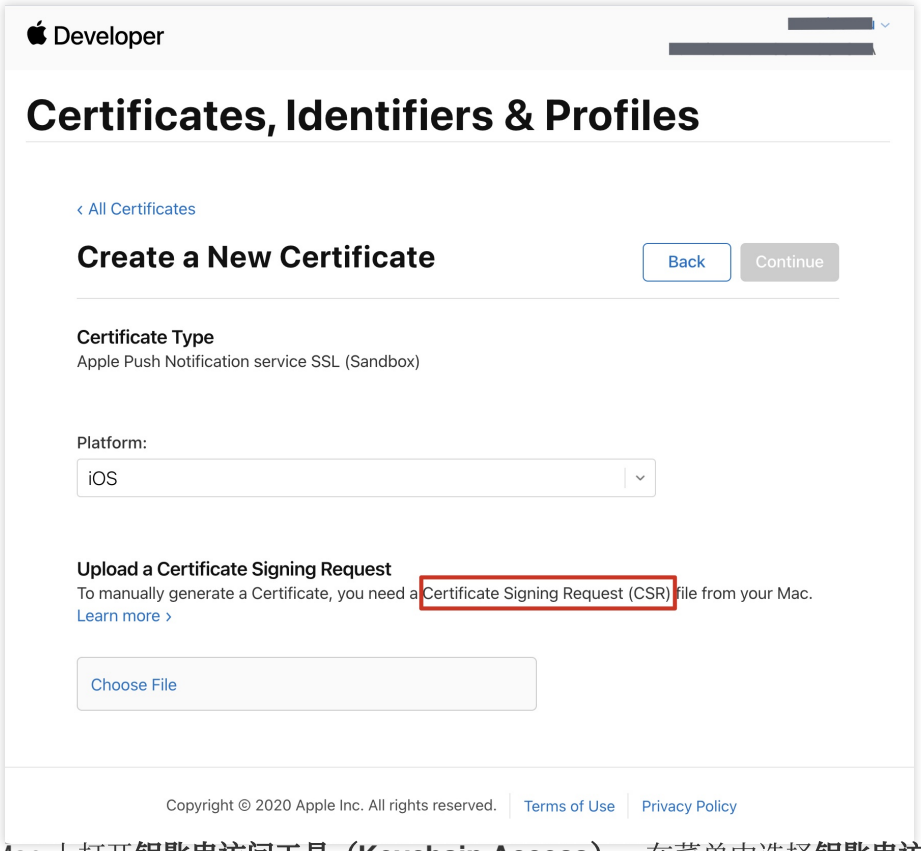
2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate`，分别用于开发环境（Development）和生产环境（Production）的远程推送证书，如下图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.



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

©2013-2025 Tencent Cloud International Pte. Ltd.

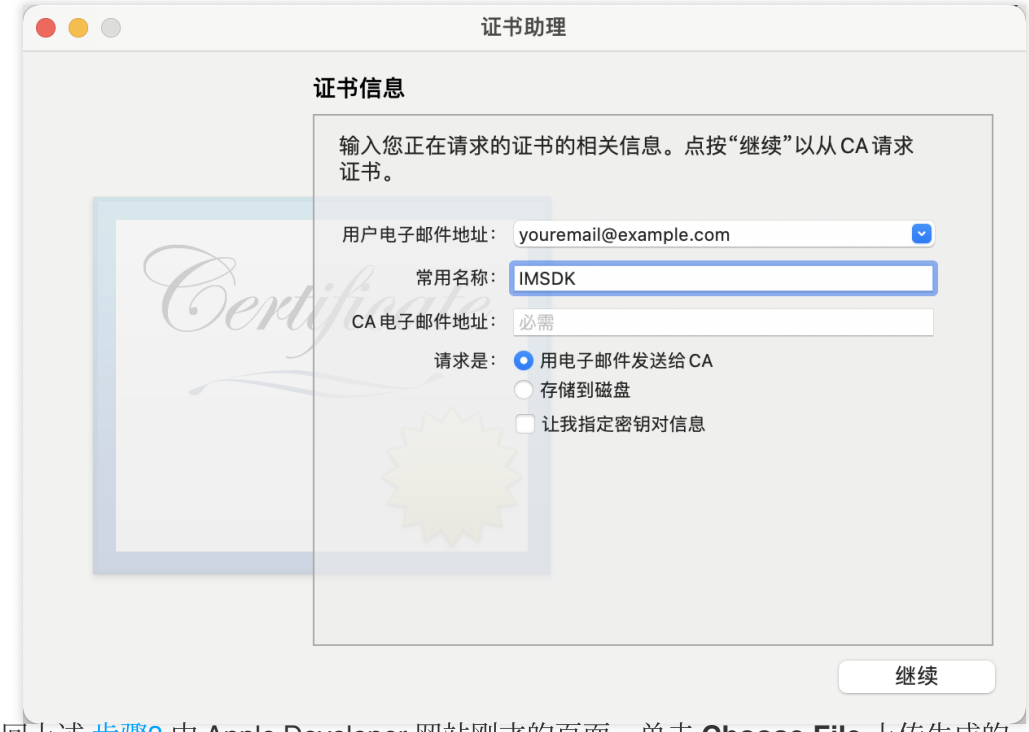


4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。

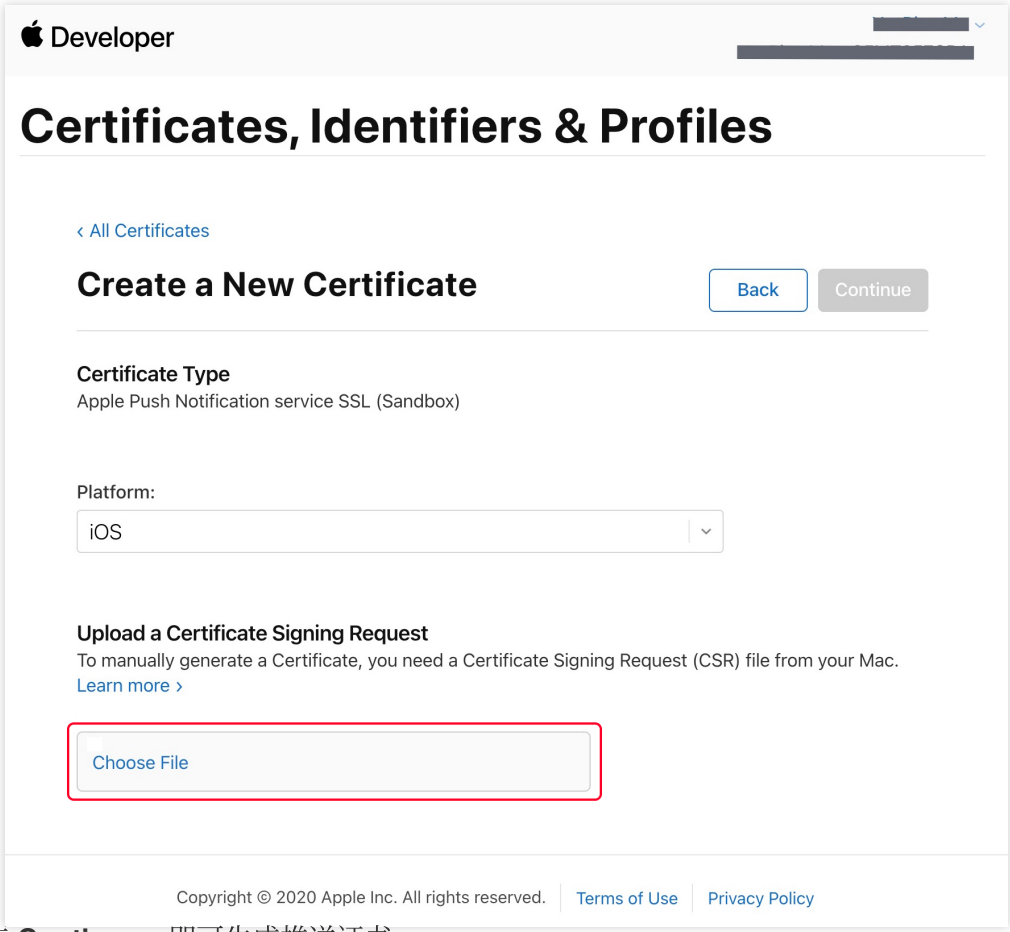


5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

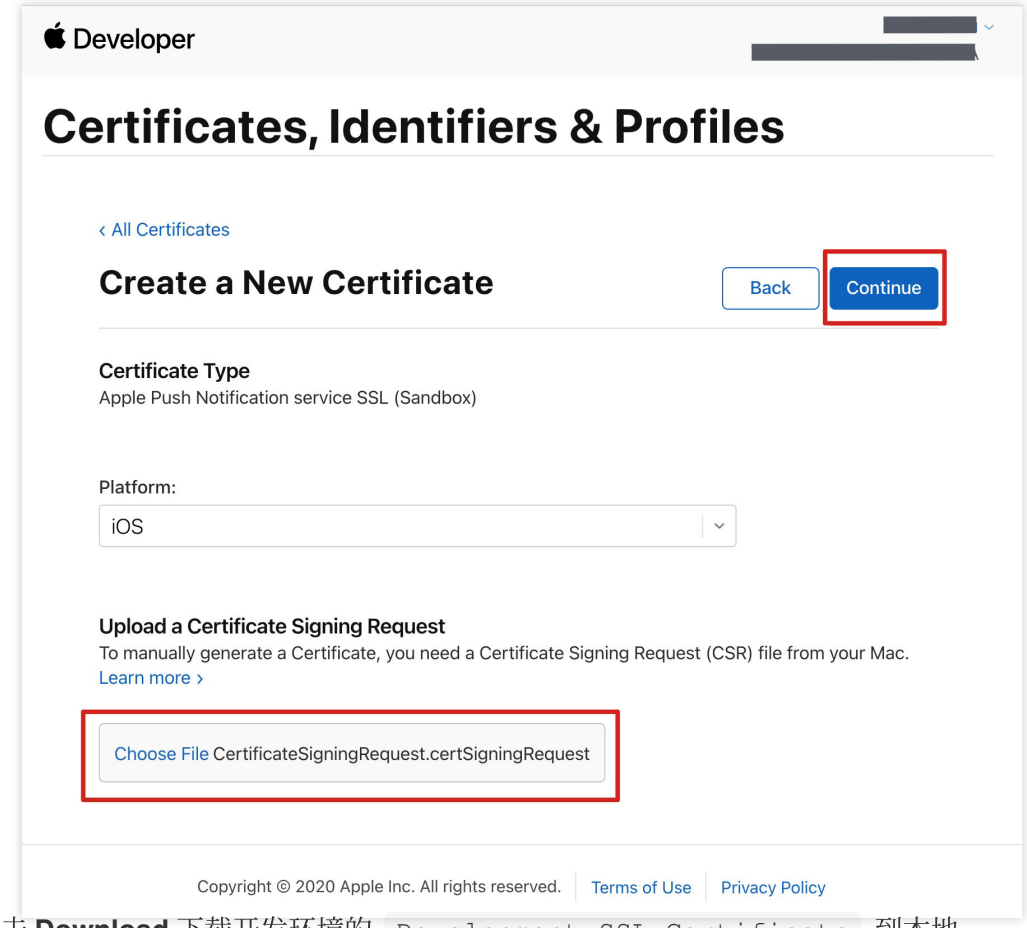


6. 返回上述 步骤3 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。

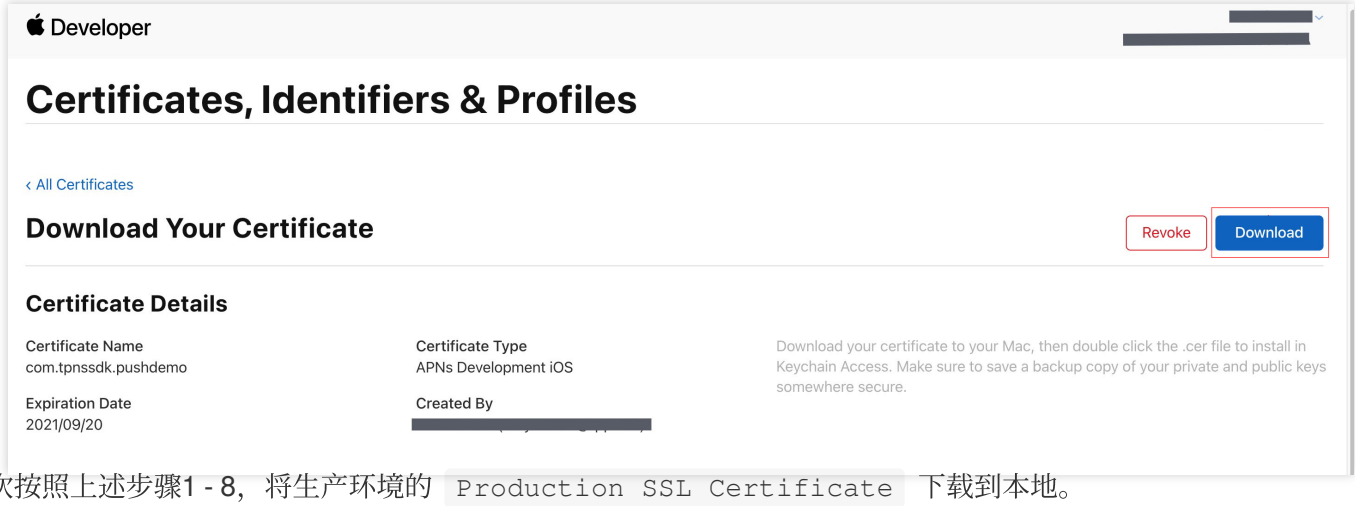


7. 单击 **Continue**，即可生成推送证书。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

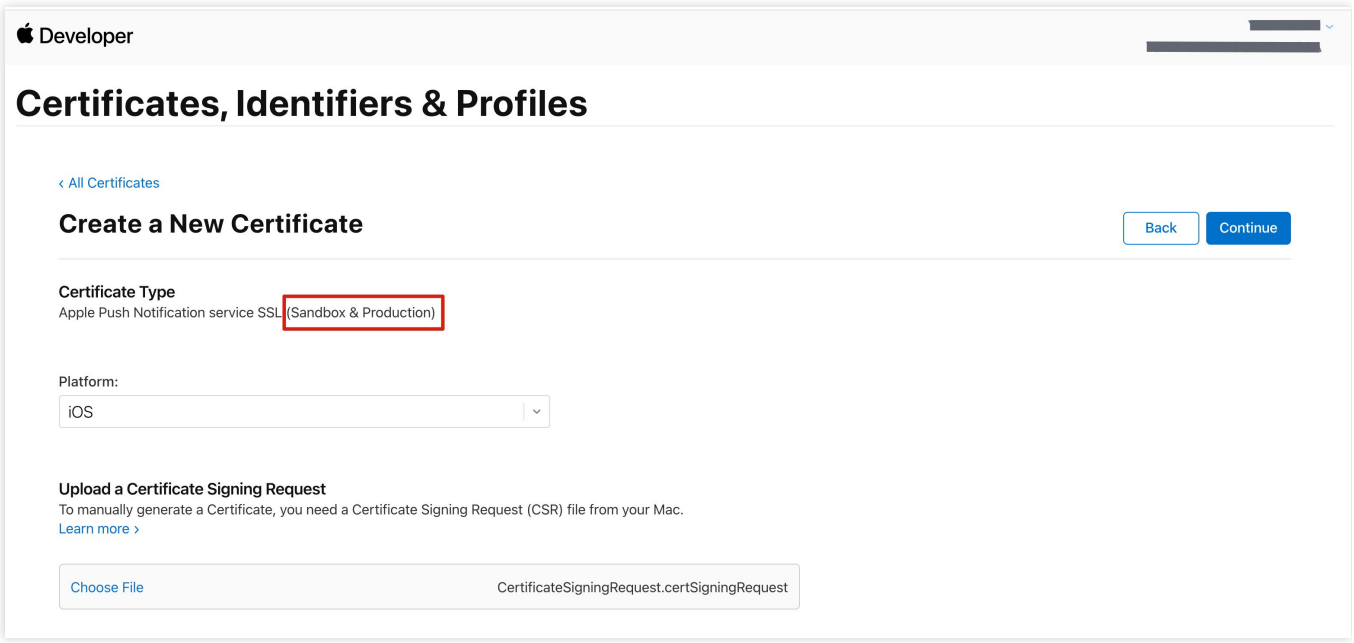


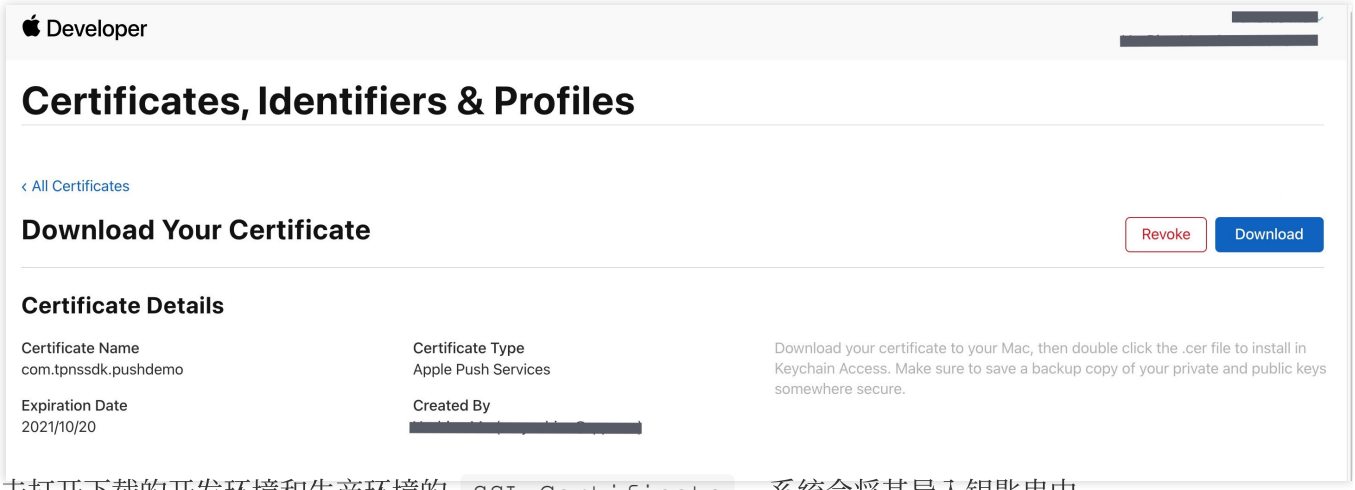
9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

说明

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

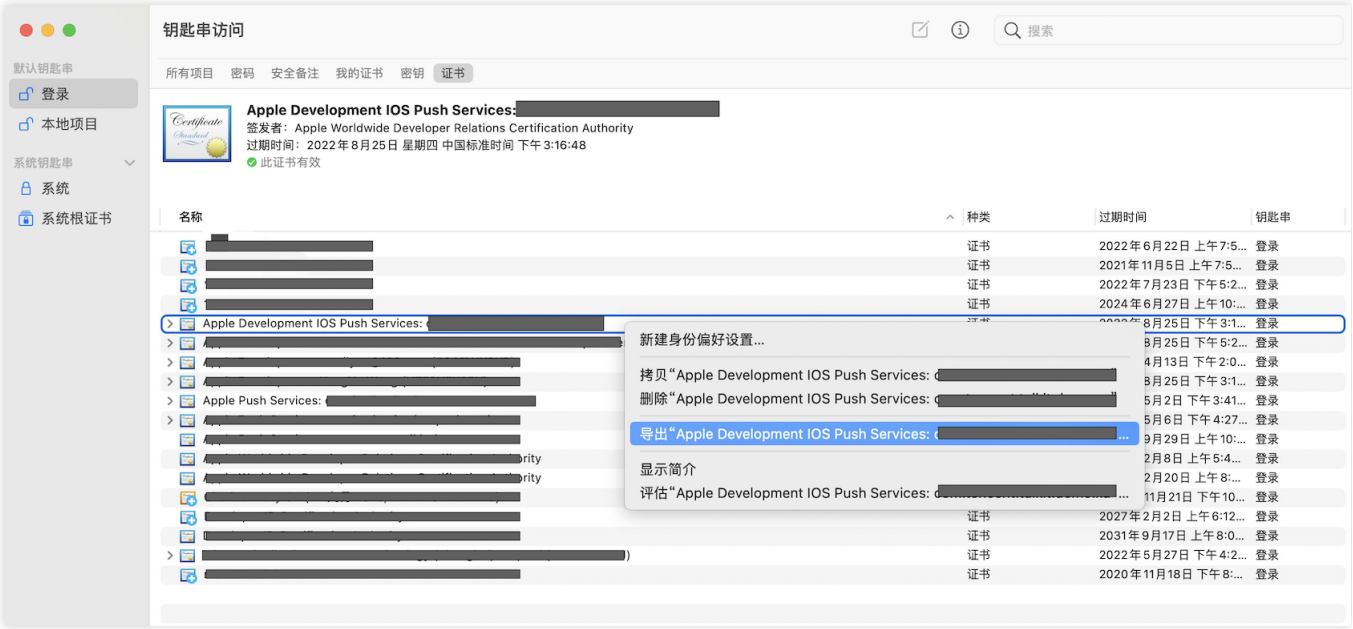
©2013-2025 Tencent Cloud International Pte. Ltd.





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

©2013-2025 Tencent Cloud International Pte. Ltd.

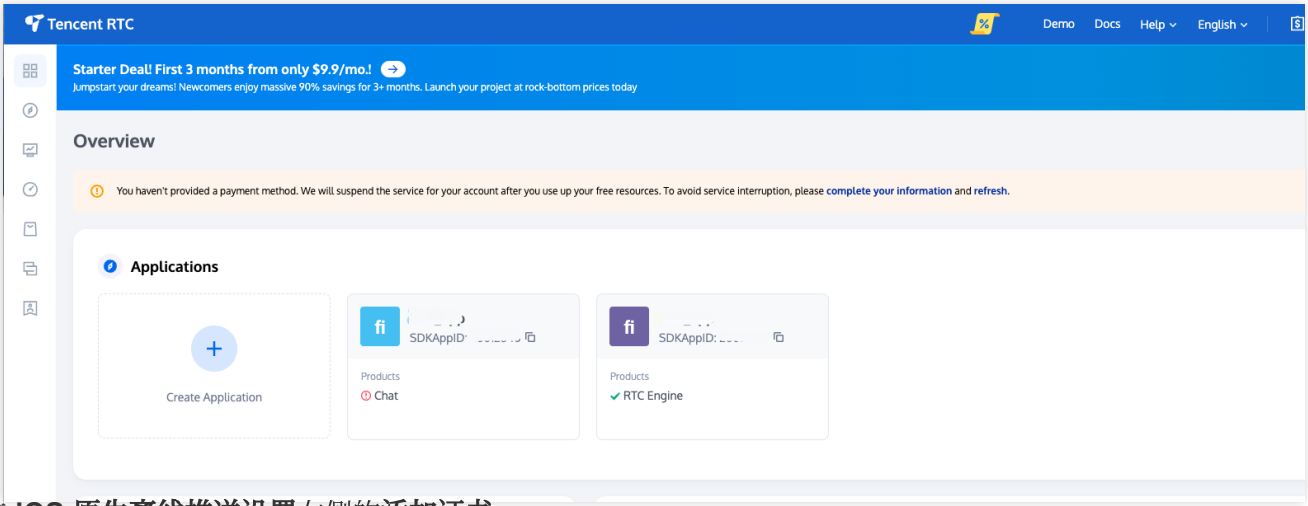


注意

保存 `P12` 文件时，请务必要为其设置密码。

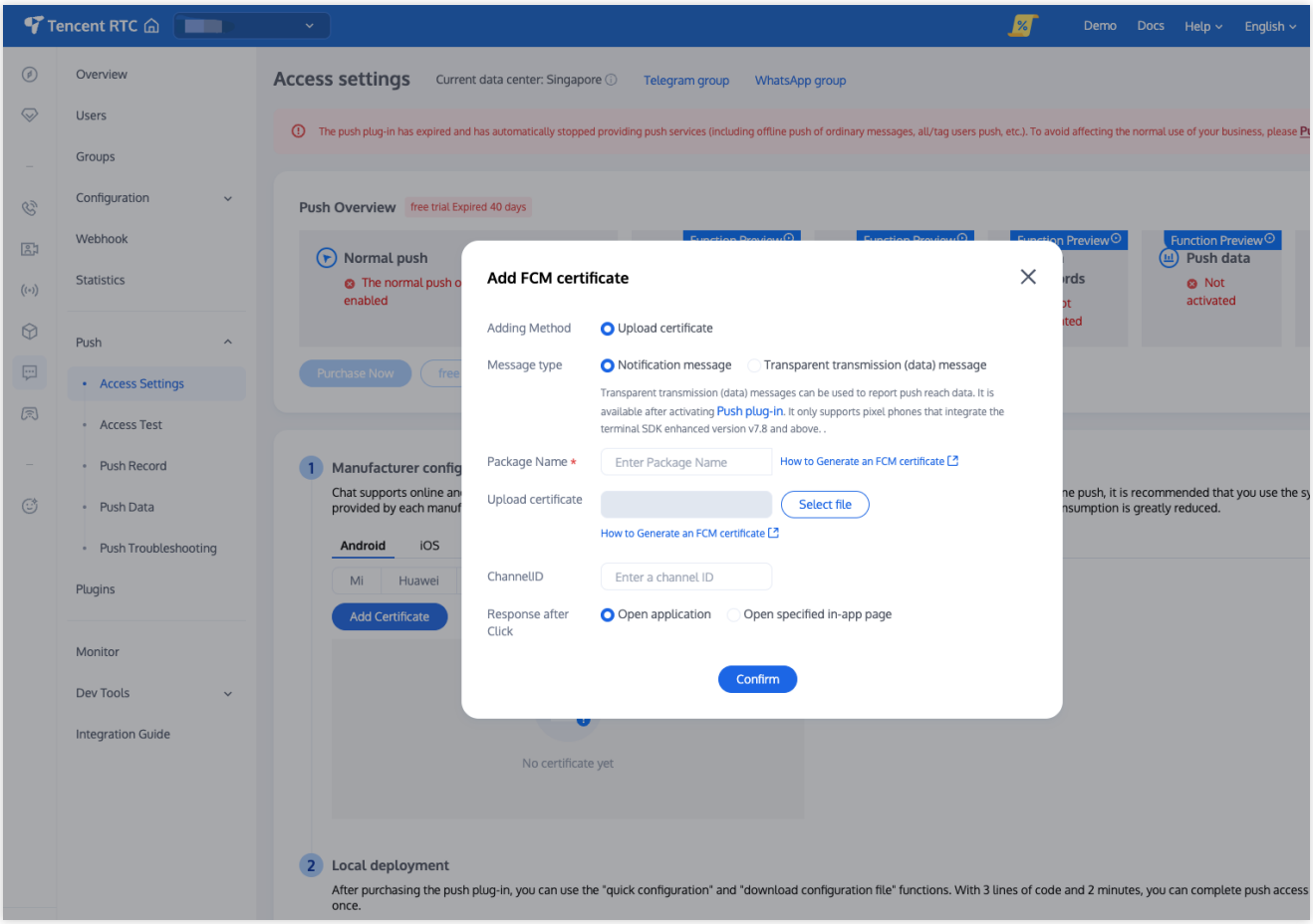
步骤2：上传证书到控制台

1. 登录 即时通信 Chat 控制台。
2. 单击目标应用卡片，进入应用的基础配置页面。



3. 单击 **iOS 原生离线推送设置**右侧的**添加证书**。
4. 选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击**确认**。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意:

上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

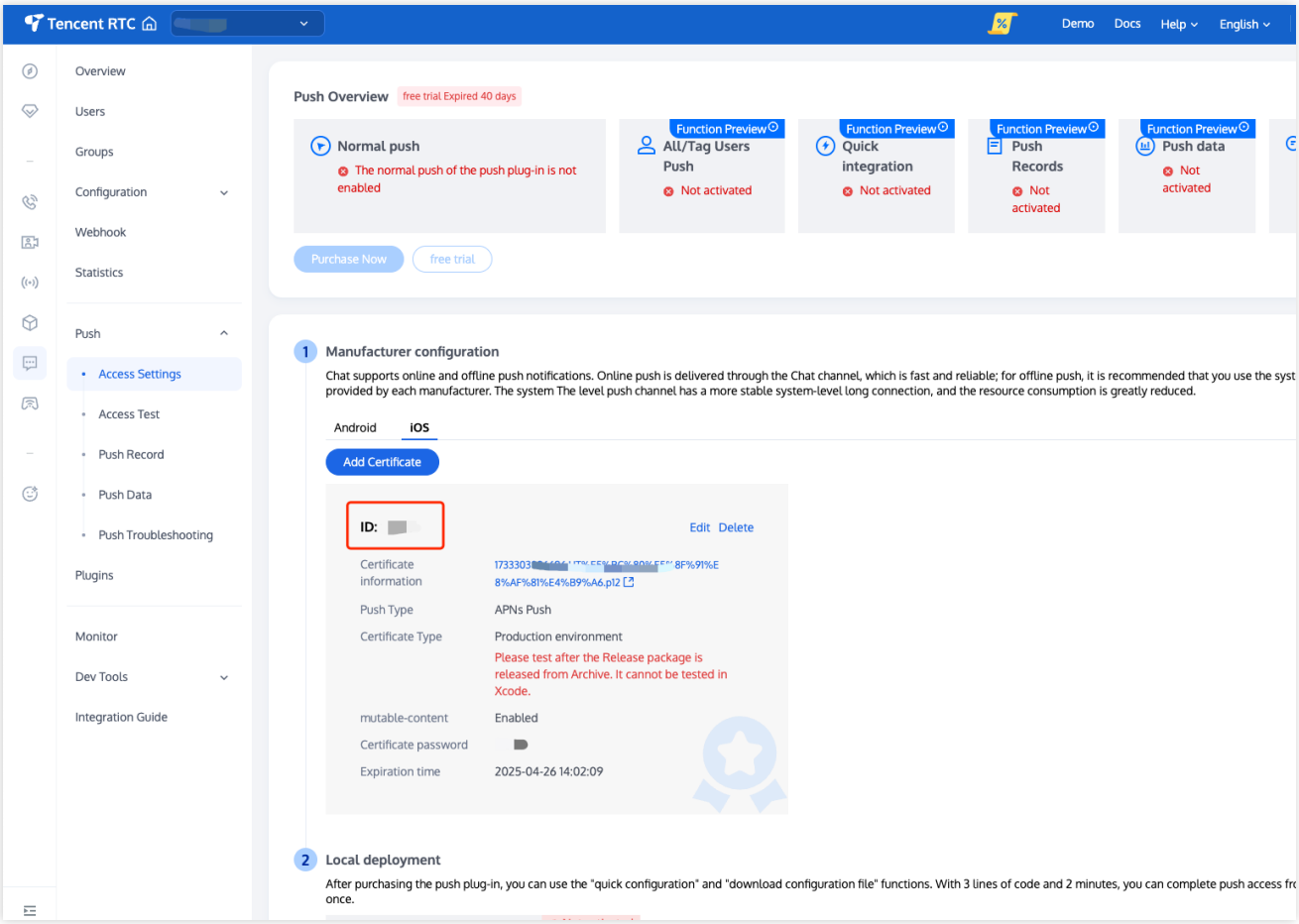
上传证书需要设置密码，无密码收不到推送。

发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。

5. 待推送证书信息生成后，记录证书的 ID。

©2013-2025 Tencent Cloud International Pte. Ltd.



操作步骤

步骤1：注册应用到厂商推送平台

离线推送需要将您自己的应用注册到各个厂商的推送平台，得到 AppID 和 AppKey 等参数，来实现离线推送功能。目前支持的手机厂商有：小米、华为、荣耀、OPPO、vivo、魅族、 Google FCM。

步骤2：Chat 控制台配置

登录腾讯云 即时通信 Chat 控制台 ，在**推送管理 > 接入设置**功能栏添加各个厂商推送证书，并将您在步骤一中获取的各厂商的 AppId、AppKey、AppSecret 等参数配置给添加的推送证书。

关于**点击后续动作**选项的说明：

打开应用：单击通知栏后拉起应用，默认启动应用的 Launcher 界面；

打开网页：单击通知栏会跳转到配置的网页链接；

打开应用内指定界面：单击通知栏会根据配置自定义跳转界面，详见 自定义点击跳转。

小米

华为

OPPO

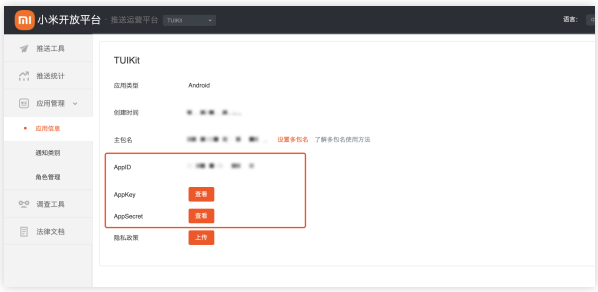
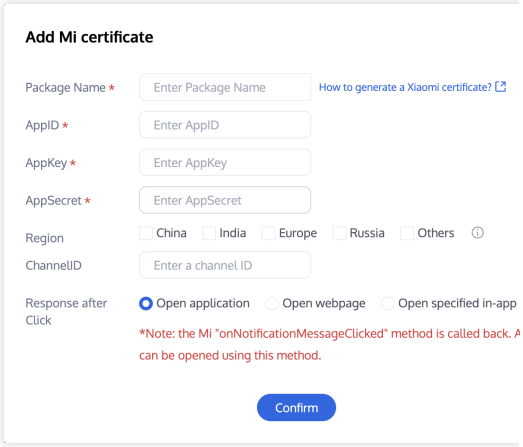
vivo

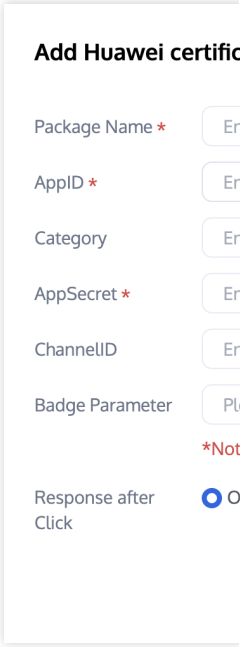
©2013-2025 Tencent Cloud International Pte. Ltd.

魅族

荣耀

Google FCM

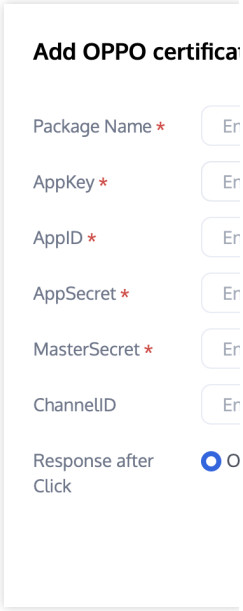
厂商推送平台	Chat 控制台配置
	

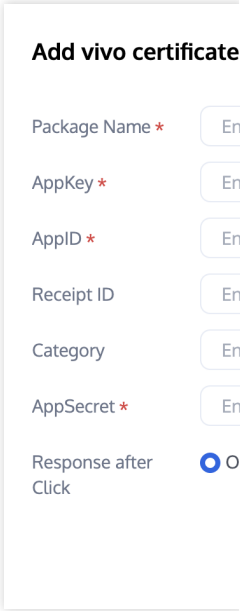
厂商推送平台	Chat 控制台配置
	

说明：

Client ID 对应 AppID，Cli

©2013-2025 Tencent Cloud International Pte. Ltd.

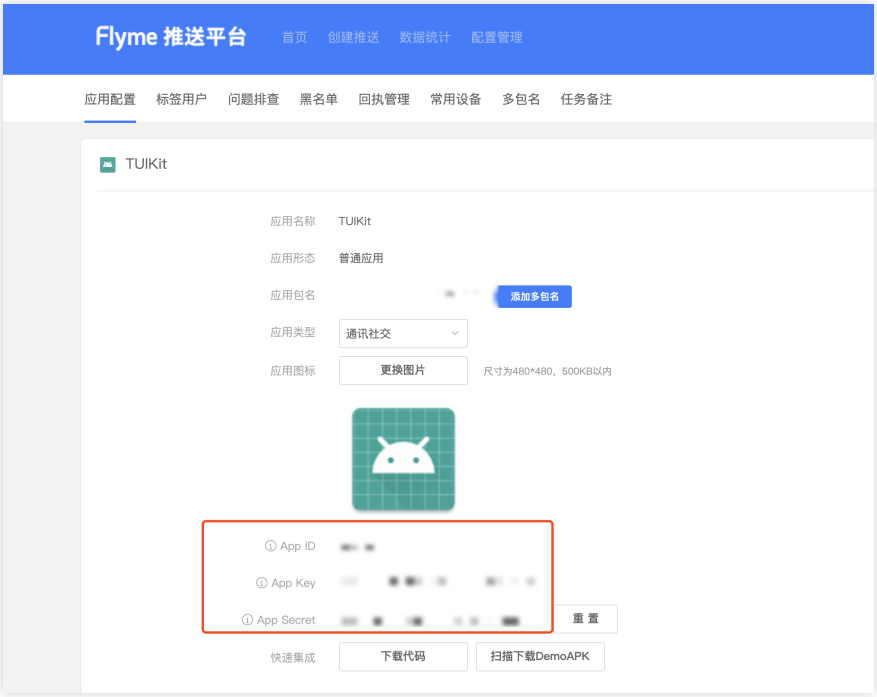
厂商推送平台	Chat 控制台配置
	

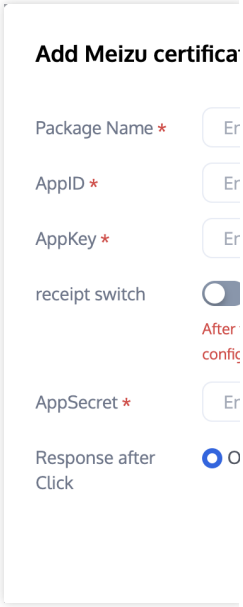
厂商推送平台	Chat 控制台配置
	

回执配置请参见：消息触达统计配置-vivo。

厂商推送平台	Chat 控制台配置

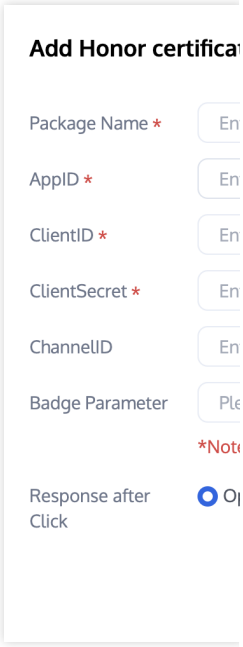
©2013-2025 Tencent Cloud International Pte. Ltd.



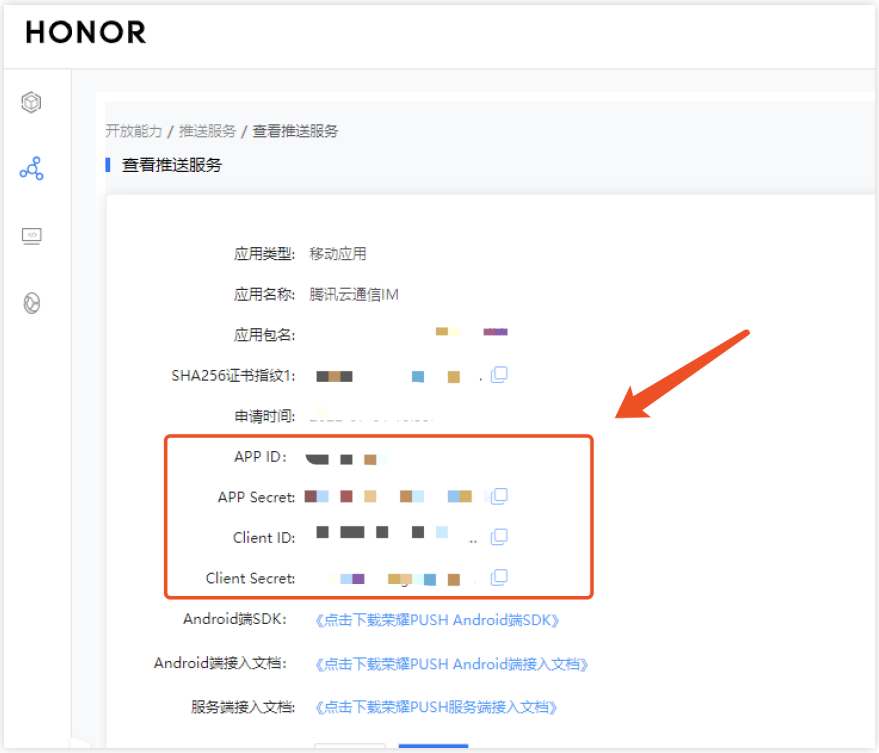


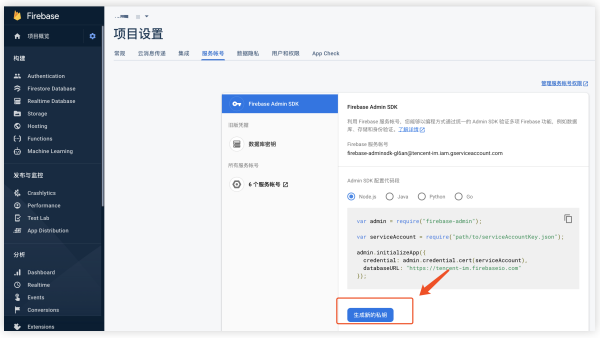
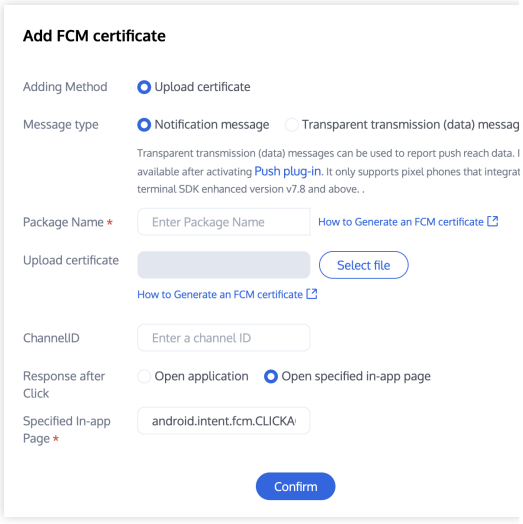
回执配置请参考：消息触达统计配置-魅族。

厂商推送平台	Chat 控制台配置



©2013-2025 Tencent Cloud International Pte. Ltd.



厂商推送平台	Chat 控制台配置
	

注意：

关于**点击后续动作**支持上报统计功能：

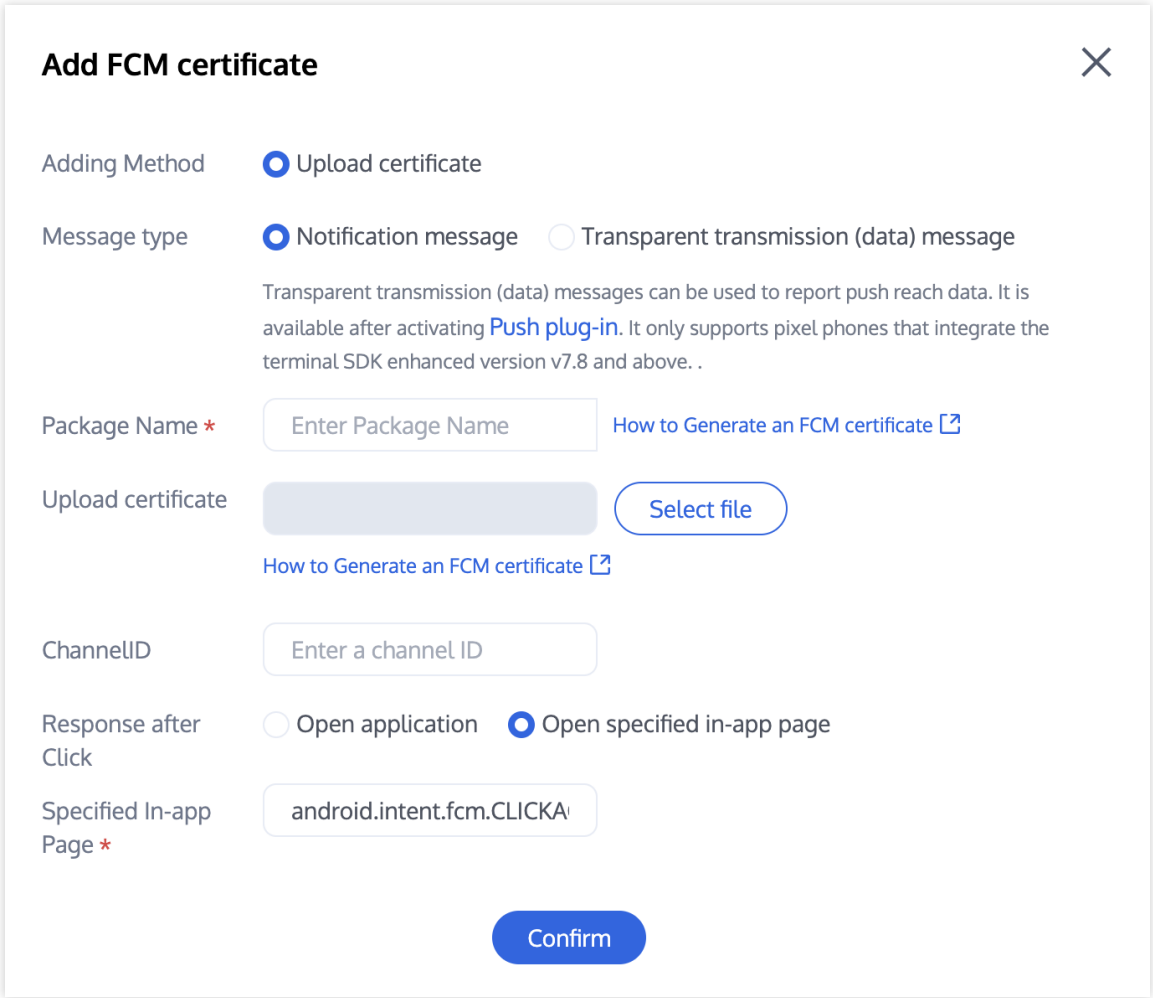
1. 如果选择打开应用和打开网页，购买插件后会默认支持上报统计。
2. 如果选择打开应用内指定界面：

©2013-2025 Tencent Cloud International Pte. Ltd.

新增证书情况，请直接使用自动填写的默认值即可支持点击上报统计。
如果之前有证书且已配置，继续使用旧证书需要修改为默认值，才可以支持上报统计，或者重新生成新的证书。

关于 FCM 数据消息

FCM 提供两种推送方式是通知消息和数据消息。
通知消息，样式简单不区分设备，成功集成即可进行离线推送。
数据消息，样式定制丰富，特定设备有效，支持触达和点击上报，需要集成后在设备上做好测试开放上线。
控制台默认选择为通知消息，两种模式切换可在 Chat 控制台操作：



注意：

FCM 数据消息能力仅支持 TChatPush 7.8 及以上版本的 pixel 手机，其他厂商设备需自测支持情况。

©2013-2025 Tencent Cloud International Pte. Ltd.

快速接入

最近更新时间：2025-04-16 10:58:10

步骤1: 集成消息推送插件

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

```
flutter pub add tencent_cloud_chat_push
```

步骤2: 推送参数配置

iOS

Android

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

Chat 控制台会为您分配一个证书 ID，见下图：

注册推送需要将此证书 ID（apnsCertificateID）传入：

```
TencentCloudChatPush().registerPush(apnsCertificateID: 您配置的证书 ID);
```

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

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

步骤3: 客户端代码配置

本步骤需要编写若干原生代码，例如：Swift、 Java、XML 等。

请不要担心，您可直接根据说明，复制我们提供的代码到指定文件即可。

iOS

Android

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

```
import UIKit
import Flutter

// Add these two import lines
import TIMPush
import tencent_cloud_chat_push

// Add `, TIMPushDelegate` to the following line
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate, TIMPushDelegate {
    override func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKe
    ) -> Bool {
        GeneratedPluginRegistrant.register(with: self)
        return super.application(application, didFinishLaunchingWithOptions: launch
    }

    // To be deprecated
    @objc func offlinePushCertificateID() -> Int32 {
        return TencentCloudChatPushFlutterModal.shared.offlinePushCertificateID();
    }

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

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

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

建议使用 Android Studio 完成本部分编辑。

©2013-2025 Tencent Cloud International Pte. Ltd.

在您项目 android 路径下 `MainActivity` 同级目录中，新建一个新的 Application 文件类，例如可命名为 `MyApplication.java` 。
如果您已经自定义了一个 Application 类，则可直接复用，不需要再次创建。

将下列代码粘贴到该文件中如上图所示：

```
package 替换成您自己的，一般 Android Studio 会自动生成;

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

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

说明：
如果您已经创建了自己的 Application 为了其他用途，请直接 `extends TencentCloudChatPushApplication` 并保证 `onCreate()` 函数中，调用了 `super.onCreate();` 即可。
打开 `android/app/src/main/AndroidManifest.xml` 文件，为 `<application>` 标签，新增指定一个 `android:name` 参数即可，指向刚制作的自定义 `Application` 类。如图所示：

步骤4: 客户端厂商配置

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

```
dependencies {
      // 版本号 “VERSION” 请前往 更新日志 中获取配置。
      // Huawei
      implementation 'com.tencent.timpush:huawei:VERSION'
      // XiaoMi
      implementation 'com.tencent.timpush:xiaomi:VERSION'
      // OPPO
      implementation 'com.tencent.timpush:oppo:VERSION'
      // vivo
      implementation 'com.tencent.timpush:vivo:VERSION'
      // Honor
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        implementation 'com.tencent.timpush:honor:VERSION'
        // Meizu
        implementation 'com.tencent.timpush:meizu:VERSION'
        // Google Firebase Cloud Messaging (Google FCM)
        implementation 'com.tencent.timpush:fcm:VERSION'
}
```

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。

方法1

方法2

```
// android/app/build.gradle

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"
        ]
    }
}
```

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

// Vivo begin
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.api_key"
    android:value="您应用分配的证书 APPKEY" />
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.app_id"
    android:value="您应用分配的证书 APPID" />
// Vivo end

// Honor begin
<meta-data tools:replace="android:value"
    android:name="com.hihonor.push.app_id"
    android:value="您应用分配的证书 APPID" />
// Honor end
```

华为、荣耀和 Google FCM 适配

按照厂商方法，集成对应的 plugin 和 json 配置文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

注意：

以下荣耀的适配仅 7.7.5283 及以上版本需要配置。

1.1 下载配置文件添加到工程根目录/Android/app。

华为

荣耀

Google FCM

操作路径

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

Gradle 7.1 及以上版本

Gradle 7.0 版本

Gradle 7.0 以下版本

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

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

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

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

```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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

```
buildscript {
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // 配置HMS Core SDK的Maven仓地址。
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}

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

1.3 在应用级 build.gradle 文件中添加下方配置：

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

步骤5: 处理消息点击回调，并解析参数

如果您需要自定义解析收到的远程推送，您可按照如下方法实现：

自定义点击跳转实现

自定义点击跳转实现（旧方案）

注意：

1. 注册回调时机建议放在程序入口 main() 函数中。
2. 控制台配置点击后续动作按如下配置，选择 **打开应用内指定界面**，请勿修改使用默认值。

```
TIMPushListener timPushListener = TIMPushListener(
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        onNotificationClicked: (String ext) {
          debugPrint("ext: $ext");
          // 获取 ext 自定义跳转

        }
      );
  tencentCloudChatPush.addPushListener(listener: timPushListener);
```

请定义一个函数，用于接受推送消息点击回调事件.

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

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

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

您可定义一个函数来接收该回调，并据此跳转至对应会话页面或您的业务页面。

示例如下:

```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  if (userID != null || groupID != null) {
    // 根据 userID 或 groupID 跳转至对应 Message 页面.
  } else {
    // 根据 ext 字段, 自己写解析方式, 跳转至对应页面.
  }
}
```

步骤6: 注册推送插件

请在 Chat 登录完成后，且在其他插件 (如 CallKit) 使用前，立即注册推送插件。请注意，不要在 Flutter 程序入口的 main 方法中调用。

调用 `TencentCloudChatPush().registerPush` 方法成功后，就可以收到离线推送通知了。

```
TencentCloudChatPush().registerPush(
    onNotificationClicked: _onNotificationClicked,
    sdkAppId: 您的sdkAppId,
    appKey: "客户端密钥",
    apnsCertificateID: 您配置的证书 ID);
```

步骤7: 消息推送触达统计

如果您需要统计触达数据，请按照如下完成配置：

华为

©2013-2025 Tencent Cloud International Pte. Ltd.

荣耀

vivo

魅族

iOS

回执地址：

新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/huawei

韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/huawei

美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/huawei

德　国： https://apiger.im.qcloud.com/v3/offline_push_report/huawei

印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/huawei

中　国： https://api.im.qcloud.com/v3/offline_push_report/huawei

注意：

华为推送证书 ID <= 11344 ，使用华为推送 v2 版本接口，不支持触达和点击回执，请重新生成更新证书 ID。

回执地址：

新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/honor

韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/honor

美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/honor

德　国： https://apiger.im.qcloud.com/v3/offline_push_report/honor

印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/honor

中　国： https://api.im.qcloud.com/v3/offline_push_report/honor

回调地址配置	回执 ID 配置 IM 控制台
回执地址： 新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/vivo 韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/vivo 美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/vivo 德　国： https://apiger.im.qcloud.com/v3/offline_push_report/vivo 印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/vivo 中　国： https://api.im.qcloud.com/v3/offline_push_report/vivo	

©2013-2025 Tencent Cloud International Pte. Ltd.

打开回执开关	配置回执地址

回执地址：

新加坡： https://apisgp.im.qcloud.com/v3/offline_push_report/meizu

韩　国： https://apikr.im.qcloud.com/v3/offline_push_report/meizu

美　国： https://apiusa.im.qcloud.com/v3/offline_push_report/meizu

德　国： https://apiger.im.qcloud.com/v3/offline_push_report/meizu

印　尼： https://apiidn.im.qcloud.com/v3/offline_push_report/meizu

中　国： https://api.im.qcloud.com/v3/offline_push_report/meizu

注意：

打开回执开关后，请务必确保回执地址正确配置。不配置或者配置地址错误，都会影响推送功能。

iOS 端推送触达统计配置，请参见 **统计推送抵达率**。

其余支持厂商不需要配置，FCM 暂不支持推送统计功能。

恭喜您已经完成了推送插件的接入，需要提醒您：推送插件**试用或购买到期后，将自动停止提供推送服务（包括普通消息离线推送、全员/标签推送等服务）**。为避免影响您业务正常使用，请提前购买/续费。

说明：

接入完成收不到推送，请先自助使用 排查工具 查看下具体原因，也需要关注 厂商消息分类机制。

推送指标数据查看，请使用 数据统计 查询。

全员/标签推送功能请参见：REST API 接口 - 发起全员/标签推送。

©2013-2025 Tencent Cloud International Pte. Ltd.

客户端 API

最近更新时间：2025-03-03 14:30:49

TencentCloudChatPush

class TencentCloudChatPush：推送插件接口类。

接口概览

注册/反注册推送服务接口

API	描述
registerPush	注册推送服务，可选覆盖推送信息来自接口参数 json。
unRegisterPush	反注册推送服务。
setRegistrationID	RegistrationID 是推送接收设备的唯一标识 ID。默认情况下，注册推送服务成功时自动生成该 ID，同时也支持您自定义设置。您可根据 RegistrationID 向指定设备推送消息。需要注意的是，卸载并重新安装设备会更改 RegistrationID，因此需要在注册推送服务之前调用 setRegistrationID 接口。
getRegistrationID	在成功注册推送服务后，可通过调用 getRegistrationID 接口获取推送接收设备的唯一标识 ID，即RegistrationID。您可根据 RegistrationID 向指定设备推送消息。

Push 全局监听接口

API	描述
addPushListener	添加 Push 监听器。
removePushListener	移除 Push 监听器。

自定义配置接口接口

API	描述
forceUseFCMPushChannel	指定设备离线推送使用 FCM 通道，需要在注册推送服务之前调用。

©2013-2025 Tencent Cloud International Pte. Ltd.

disablePostNotificationInForeground	关闭 App 在前台时弹出通知栏。

接口详情

推送插件类

TencentCloudChatPush()：获取 TencentCloudChatPush 推送插件实例，是一个静态单例。后续步骤, 均通过此单例实例, 进行方法调用。

成员函数说明

registerPush

注册推送服务，Chat 账号登录成功后调用。

示例代码：

```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  /// 自定义处理
}
TencentCloudChatPush().registerPush(
    onNotificationClicked: _onNotificationClicked,
    sdkAppId: 您的sdkAppId,
    appKey: "客户端密钥",
    apnsCertificateID: 您配置的证书 ID);
```

参数说明：

参数		类型	说明
onNotificationClicked	ext	String	为该消息所携带的完整 ext 信息，由发送方指定, 如果未指定，则有默认值。您可根据解析该字段，跳转至对应页面。
	userID	String?	本参数对应 userID，自动尝试解析 ext Json String， 获取里面携带的单聊对方 userID 。 **说明：** 如果您未自定义 ext 字段，ext 字段由 SDK 或 UIKit 默认指定，则可使用此处的默认解析。如果尝试解析失败，则为 null 空。
	groupID	String?	本参数对应 groupID，自动尝试解析 ext Json String， 获取里面携带的群聊 groupID 信息。 **说明：** 如果您未自定义 ext 字段，ext 字段由 SDK 或 UIKit 默认指定，则可使用此处的默认解析。如果尝试解析失败，则为 null 空。

©2013-2025 Tencent Cloud International Pte. Ltd.

sdkAppId	int?	Chat 控制台为您分配的应用 ID	
appKey	String?	Chat 控制台为您分配的客户端密钥	
apnsCertificateID	int?	如单独调用 setApnsCertificateID 方法已配置, 此项可不传。	

unRegisterPush

反注册离线推送服务。

示例代码：

```
TencentCloudChatPush().unRegisterPush();
```

setRegistrationID

设置注册离线推送服务使用的推送 ID 标识, 即 RegistrationID，需要在注册推送服务之前调用。

参数说明：

参数	描述
registrationID	设备的推送标识 ID，卸载重装会改变。

示例代码：

```
TencentCloudChatPush().setRegistrationID(registrationID: registrationID);
```

getRegistrationID

注册离线推送服务成功后，获取推送 ID 标识，即 RegistrationID。

示例代码：

```
TencentCloudChatPush().getRegistrationID();
```

addPushListener

添加 Push 监听器

示例代码：

```
TIMPushListener timPushListener = TIMPushListener(
  onRecvPushMessage: (TimPushMessage message) {
      String messageLog = message.toLogString();
      debugPrint(
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            "message: $messageLog");
        },

    onRevokePushMessage: (String messageId) {
      debugPrint(
          "message: $messageId");
    },

    onNotificationClicked: (String ext) {
      debugPrint(
          "ext: $ext");
    }
);
TencentCloudChatPush.addPushListener(listener: timPushListener);
```

removePushListener

移除 Push 监听器

示例代码：

```
TIMPushListener timPushListener = TIMPushListener(
  onRecvPushMessage: (TimPushMessage message) {
      String messageLog = message.toLogString();
      debugPrint(
          "message: $messageLog");
      },

  onRevokePushMessage: (String messageId) {
    debugPrint(
        "message: $messageId");
  },

  onNotificationClicked: (String ext) {
    debugPrint(
        "ext: $ext");
  }
);
TencentCloudChatPush.removePushListener(listener: timPushListener);
```

forceUseFCMPushChannel

指定设备离线推送使用 FCM 通道，需要在注册推送服务之前调用。

参数说明：

参数	描述
enable	true：使用 FCM 通道。

©2013-2025 Tencent Cloud International Pte. Ltd.

	false：使用本机通道。

示例代码：

```
TencentCloudChatPush.forceUseFCMPushChannel(enable: true);
```

disablePostNotificationInForeground

关闭 App 在前台时弹出通知栏。推送 SDK 收到在线推送时，会自动向通知栏增加 Notification 提示，如果您想自己处理在线推送消息，可以调用该接口关闭自动弹通知栏提示的特性。

参数说明：

参数	描述
disable	true：关闭 false：开启

示例代码：

```
TencentCloudChatPush.disablePostNotificationInForeground(disable: true);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native 厂商配置

最近更新时间：2025-03-03 14:34:31

React Native 推送目前支持小米、华为、荣耀、OPPO、vivo、魅族、APNs、一加、realme、iQOO 、FCM 和 苹果等厂商通道。

iOS

Android

集成 **@tencentcloud/react-native-push** 之前，需要先向 Apple 申请 APNs 推送证书，然后上传推送证书到 Chat 控制台 。之后按照快速接入步骤接入即可。

Apple 厂商配置目前有两种主流的证书，p12 证书和 p8 证书。两种证书各有优劣，您可按需要选择其中的一种。

	证书类型	有效期和管理	安全性	灵动岛
p12 证书	p12 证书是一个包含公钥和私钥的二进制文件，用于基于证书的身份验证。它将公钥证书和私钥捆绑在一个文件中，扩展名为 .p12 或 .pfx。	p12 证书通常有一年的有效期，过期后需要重新生成和部署。每个应用程序都需要单独的 P12 证书来处理推送通知。	p12 证书使用基于证书的身份验证，需要在服务器上存储私钥。这可能会增加安全风险，因为私钥可能会被未经授权的用户访问。	不支持
p8 证书	p8 证书是一个 Auth Key（授权密钥），用于基于令牌的身份验证。它是一个包含私钥的文本文件，扩展名为 .p8。	p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 P8 证书可以简化证书管理，因为您可以使用一个 p8 证书为多个应用程序提供推送通知服务。	p8 证书使用基于令牌的身份验证，这意味着您的服务器会周期性地生成一个 JSON Web Token（JWT）来与 APNs 建立连接。这种方法更安全，因为它不需要在服务器上存储私钥。	支持灵动岛推送

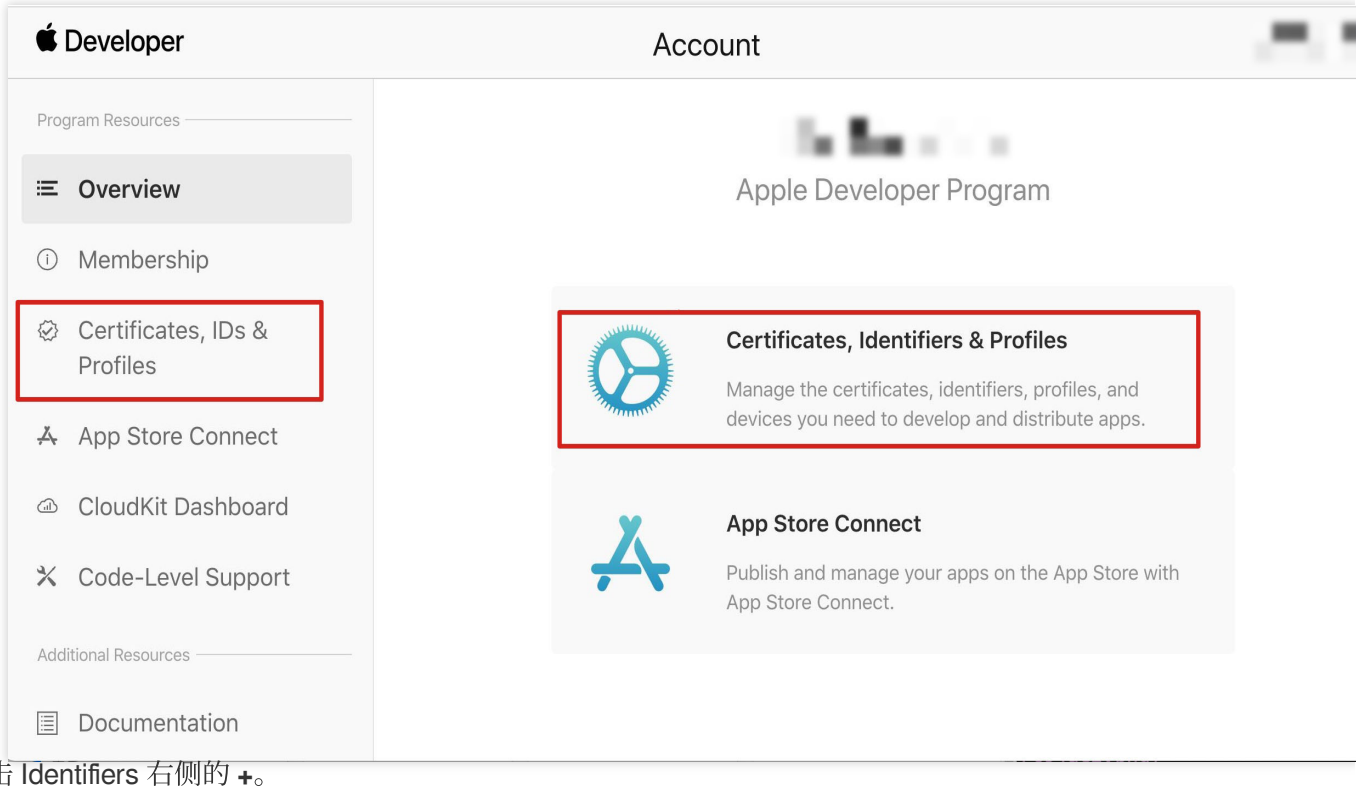
一、使用 p12 证书 （传统推送证书）

步骤1：申请 APNs 证书

开启 App 远程推送

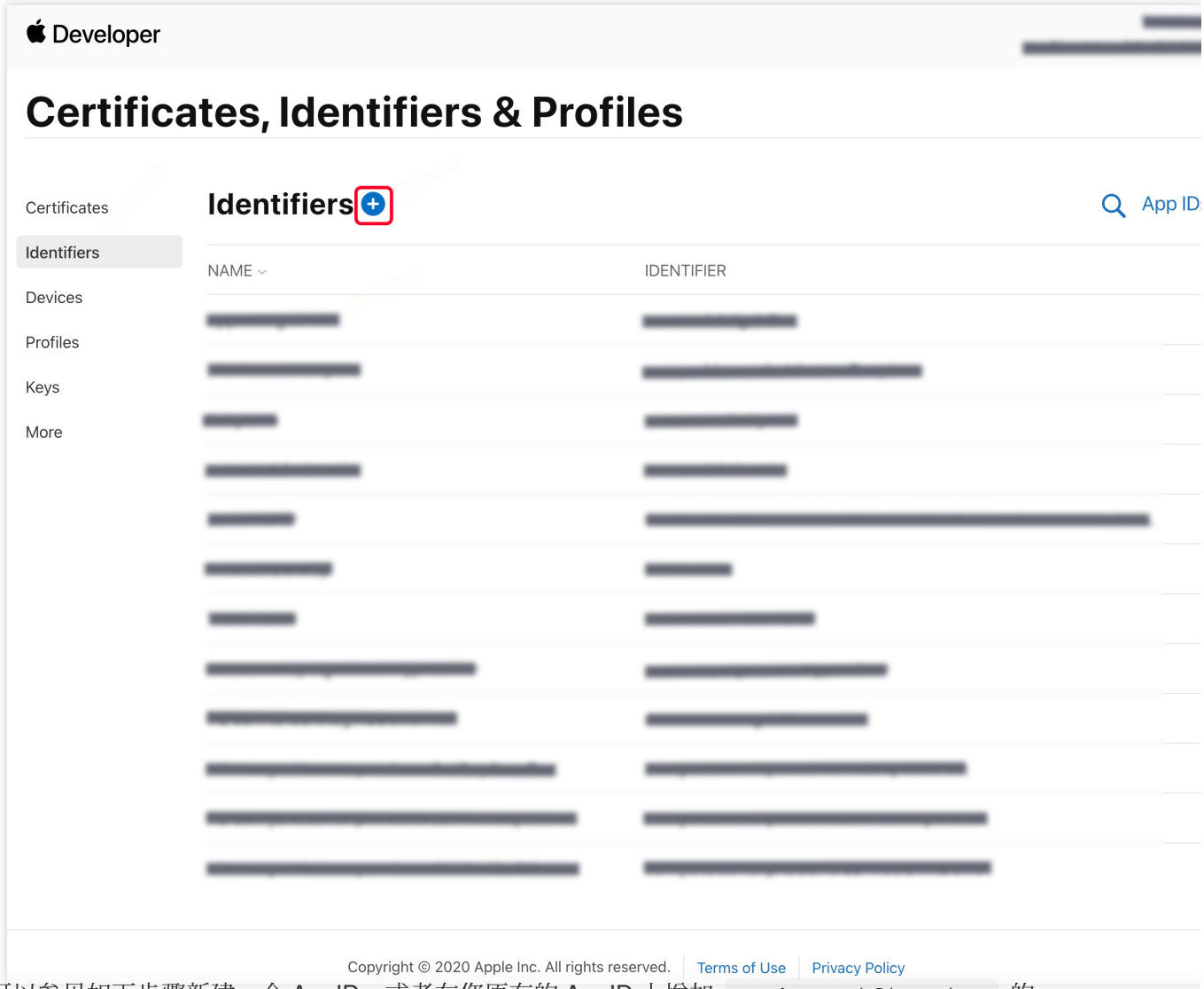
1. 登录 苹果开发者中心 网站，单击 **Certificates,Identifiers & Profiles** 或者侧栏的 **Certificates,IDs & Profiles**，进入 **Certificates, IDS & Profiles** 页面。

©2013-2025 Tencent Cloud International Pte. Ltd.



2. 单击 Identifiers 右侧的 +。

©2013-2025 Tencent Cloud International Pte. Ltd.



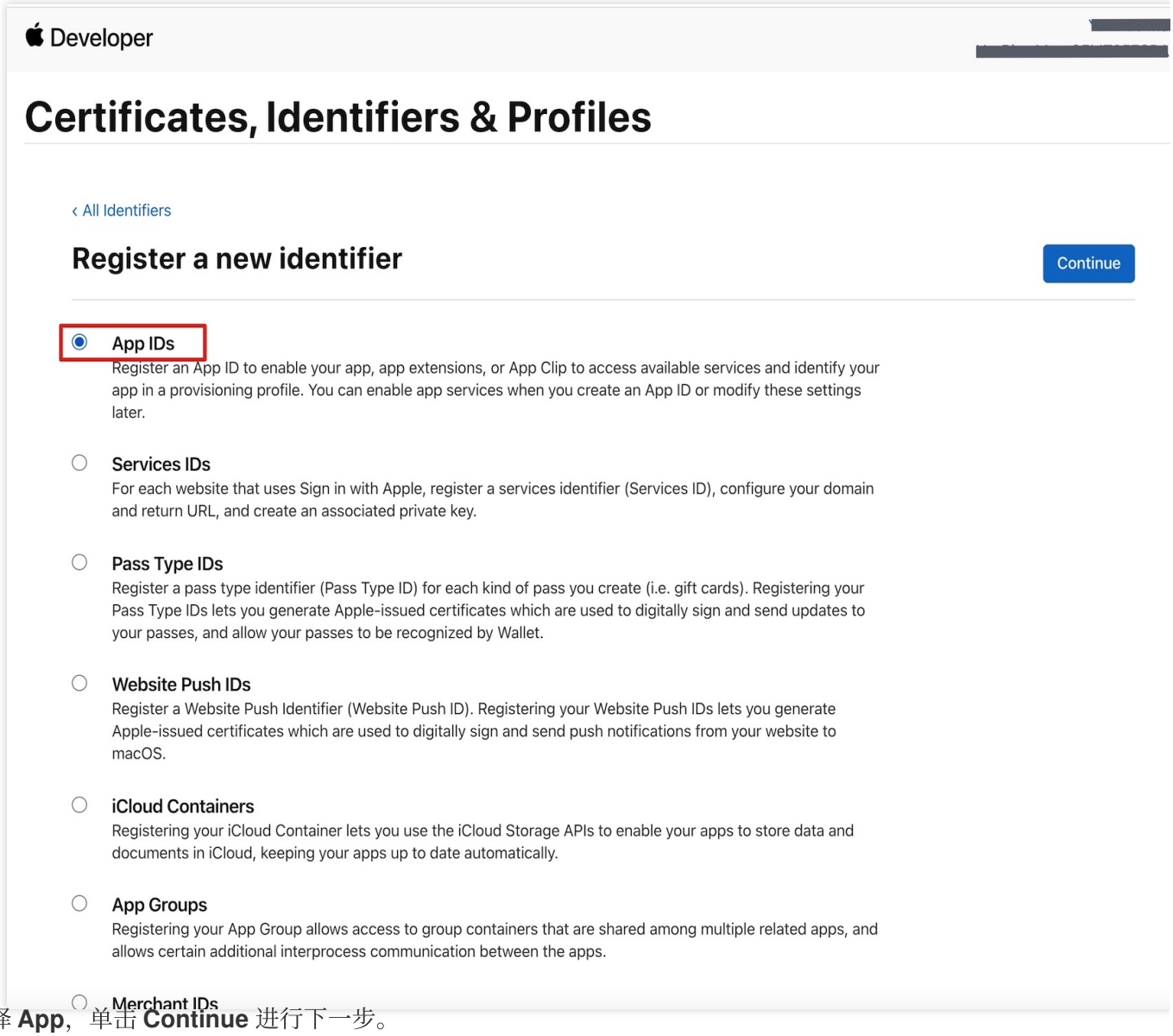
3. 您可以参见如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification Service` 。

说明：

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

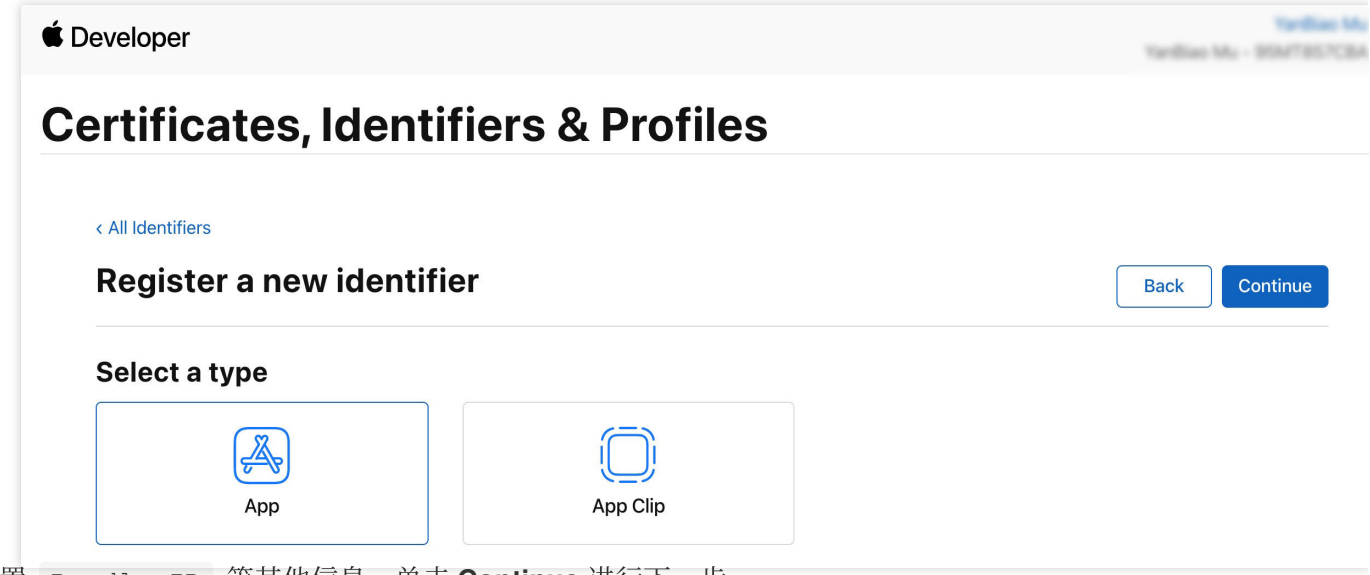
4. 勾选 **App IDs**，单击 **Continue** 进行下一步。

©2013-2025 Tencent Cloud International Pte. Ltd.

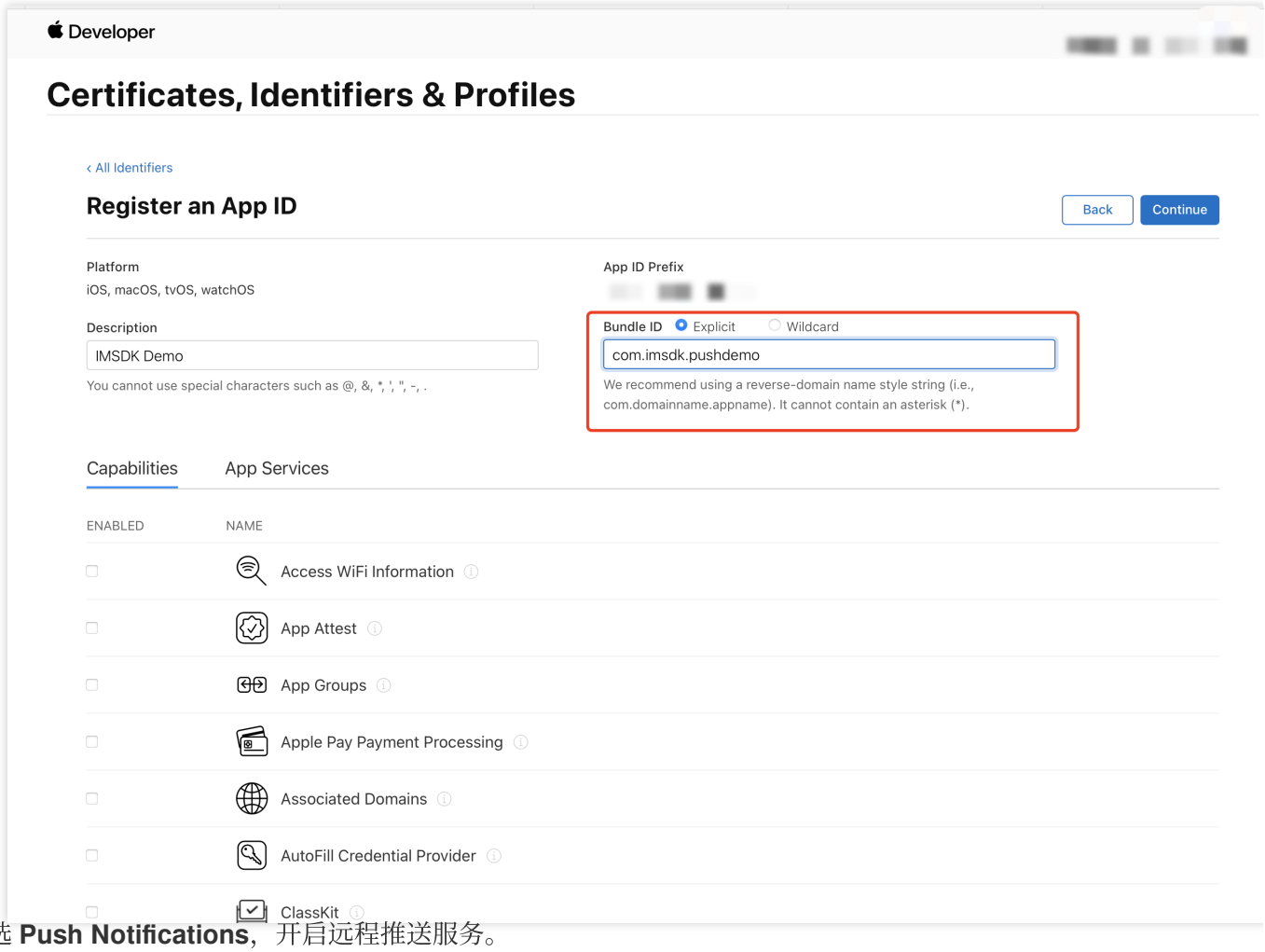


5. 选择 **App**，单击 **Continue** 进行下一步。

©2013-2025 Tencent Cloud International Pte. Ltd.

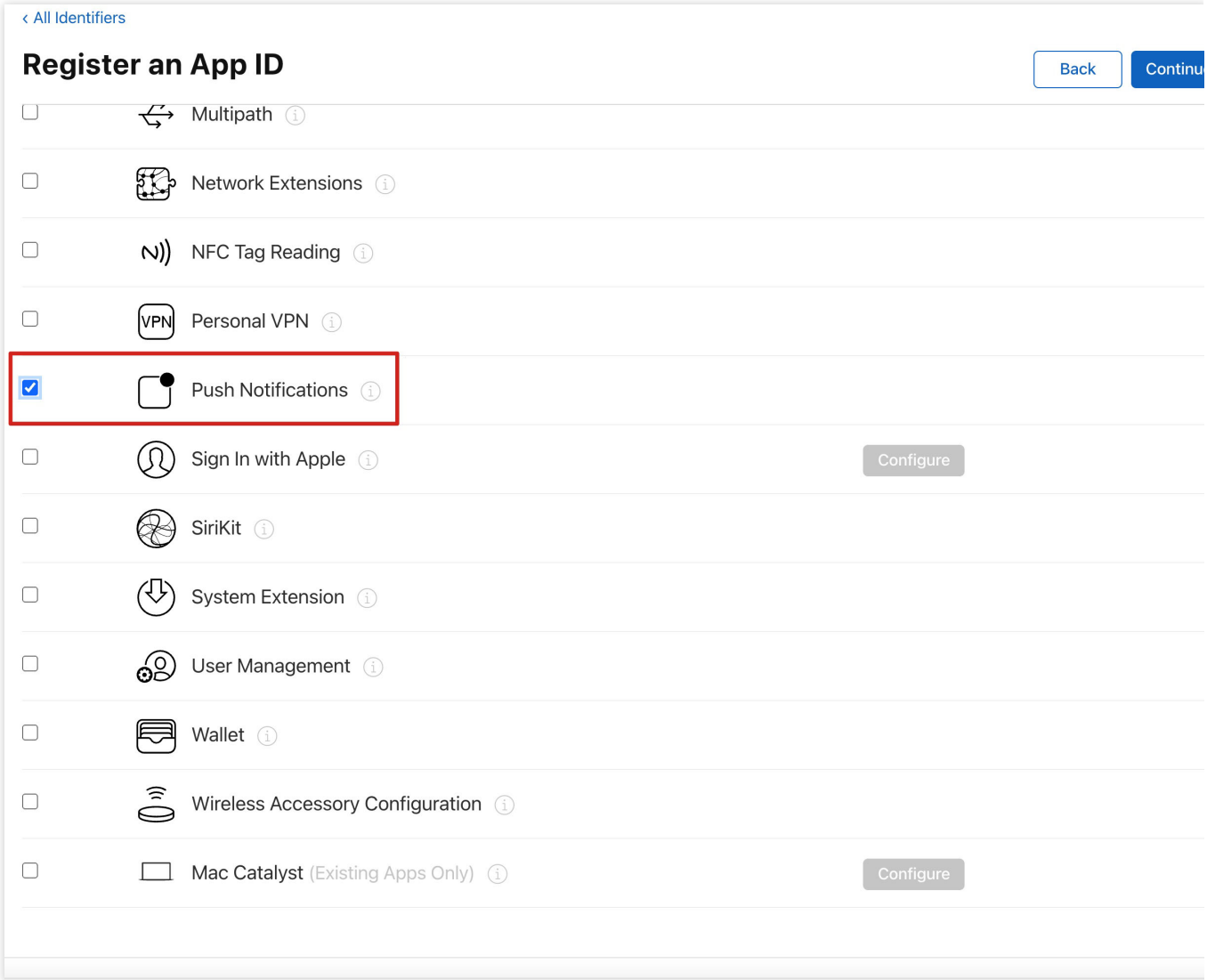


6. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。



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

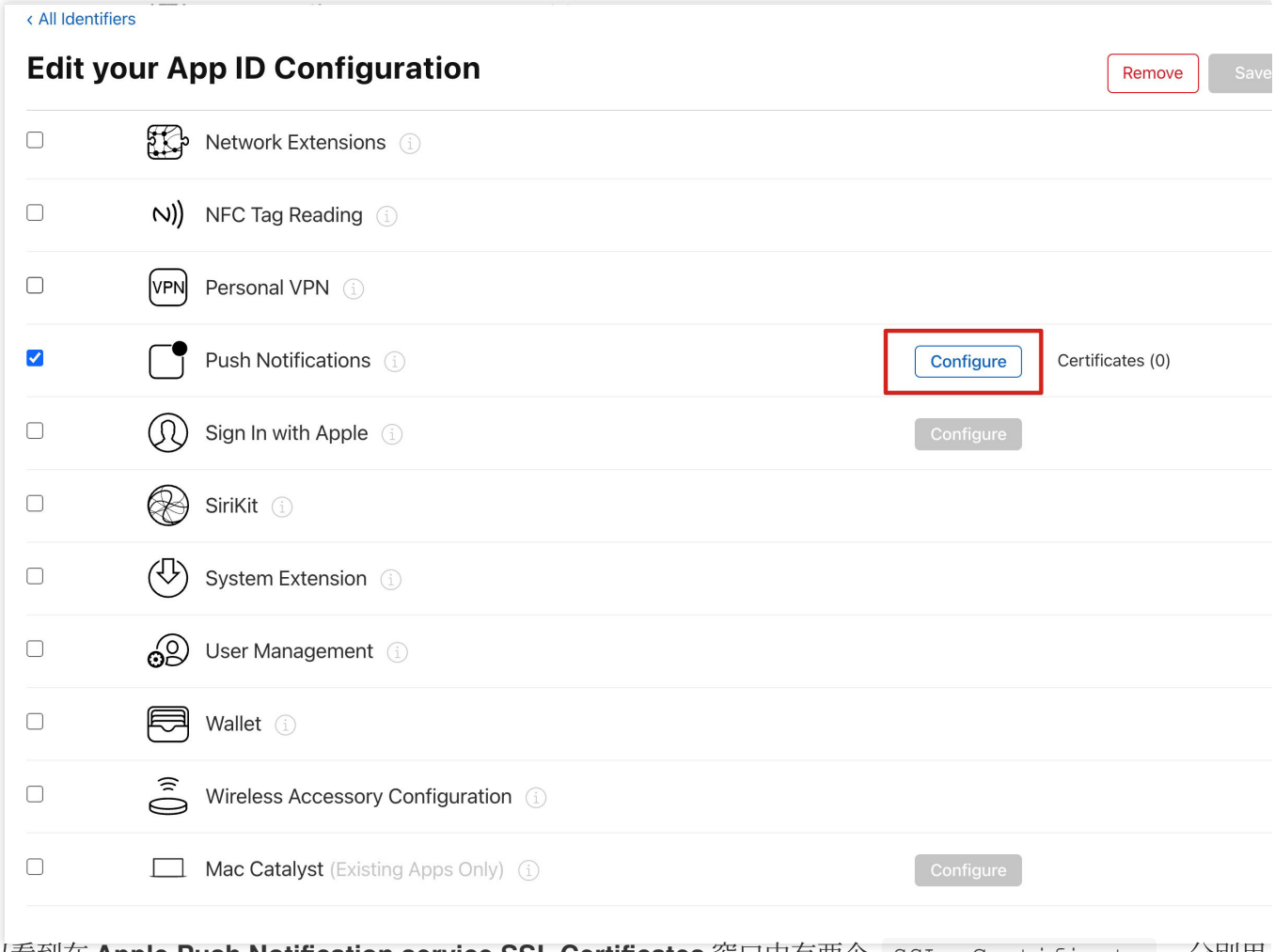
©2013-2025 Tencent Cloud International Pte. Ltd.



生成证书

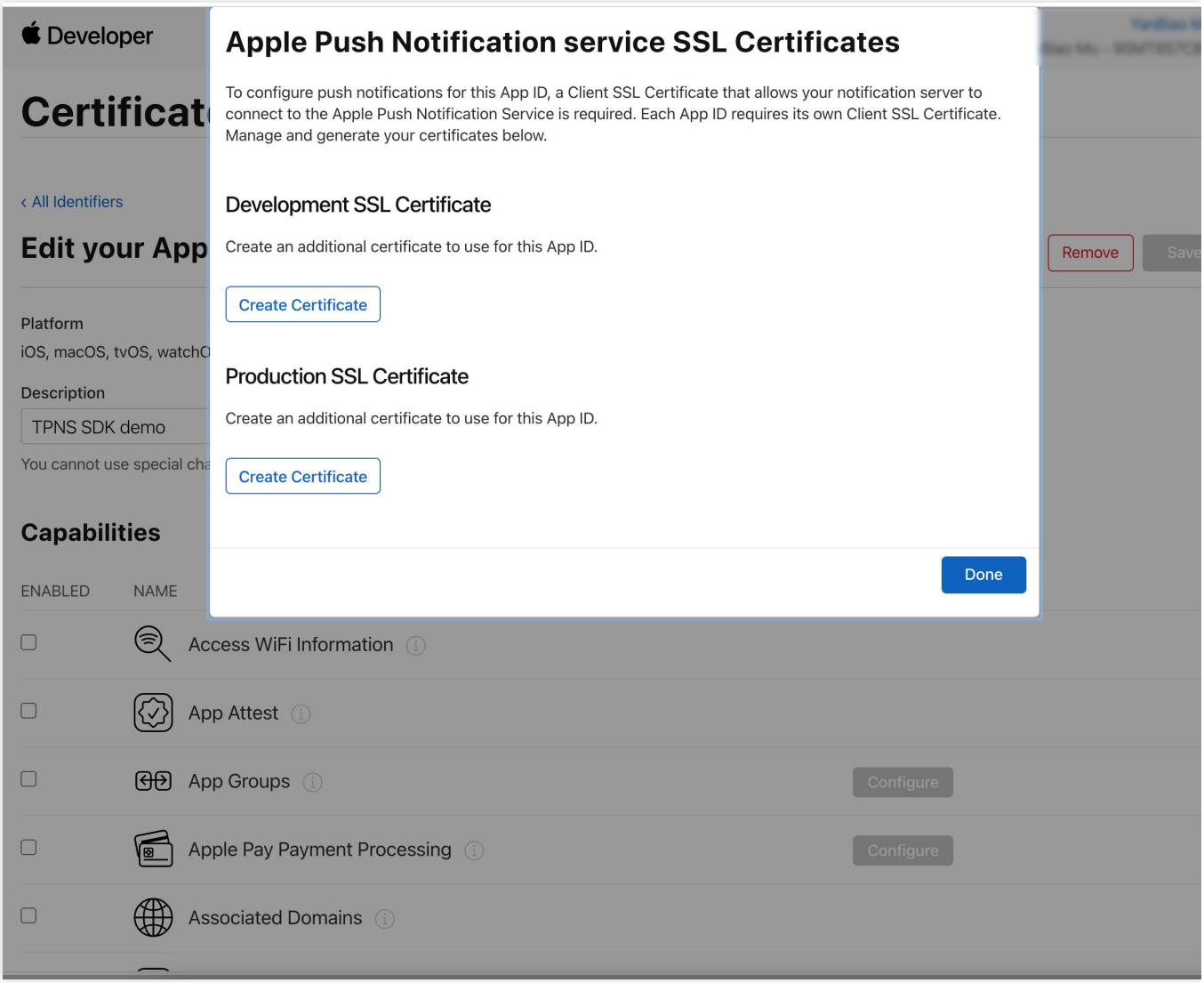
1. 选中您的 AppID，选择 **Configure**。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

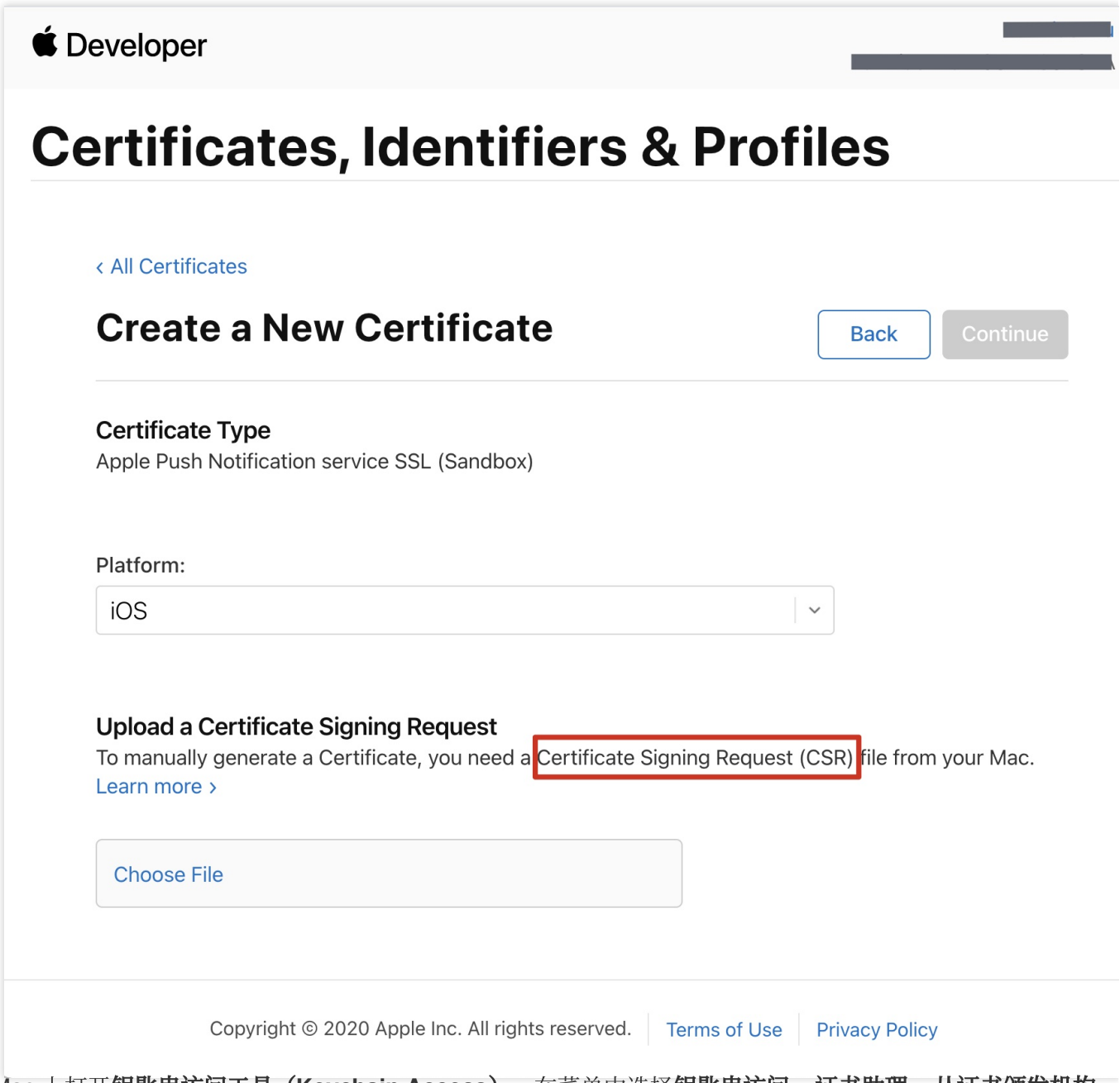
©2013-2025 Tencent Cloud International Pte. Ltd.



3.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

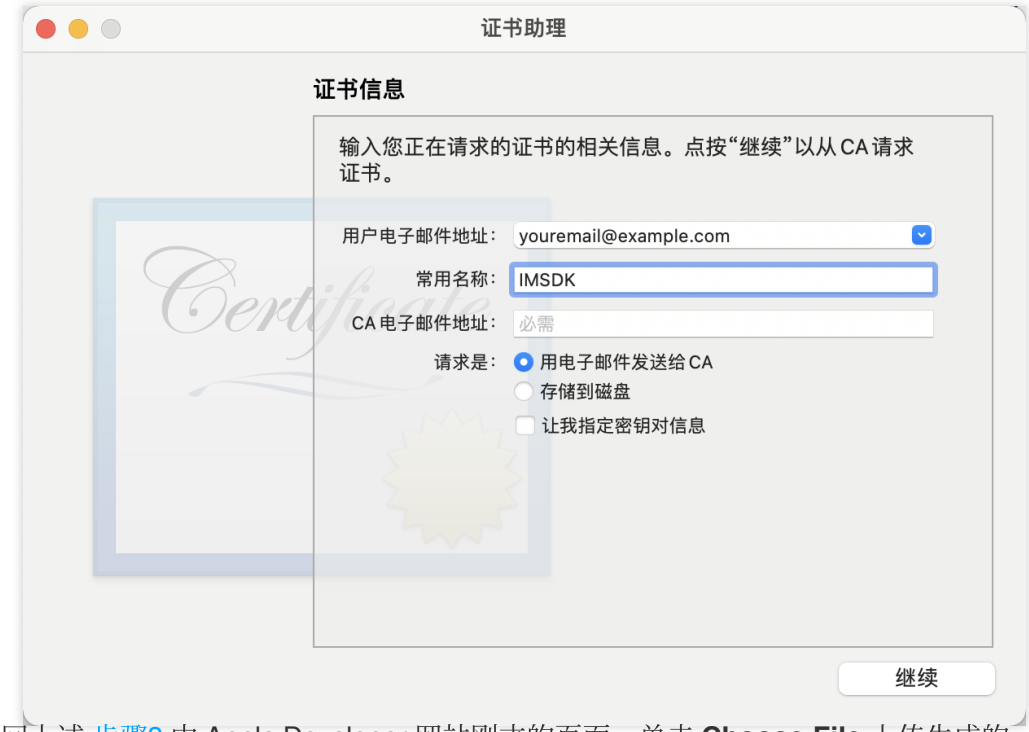


4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。

©2013-2025 Tencent Cloud International Pte. Ltd.

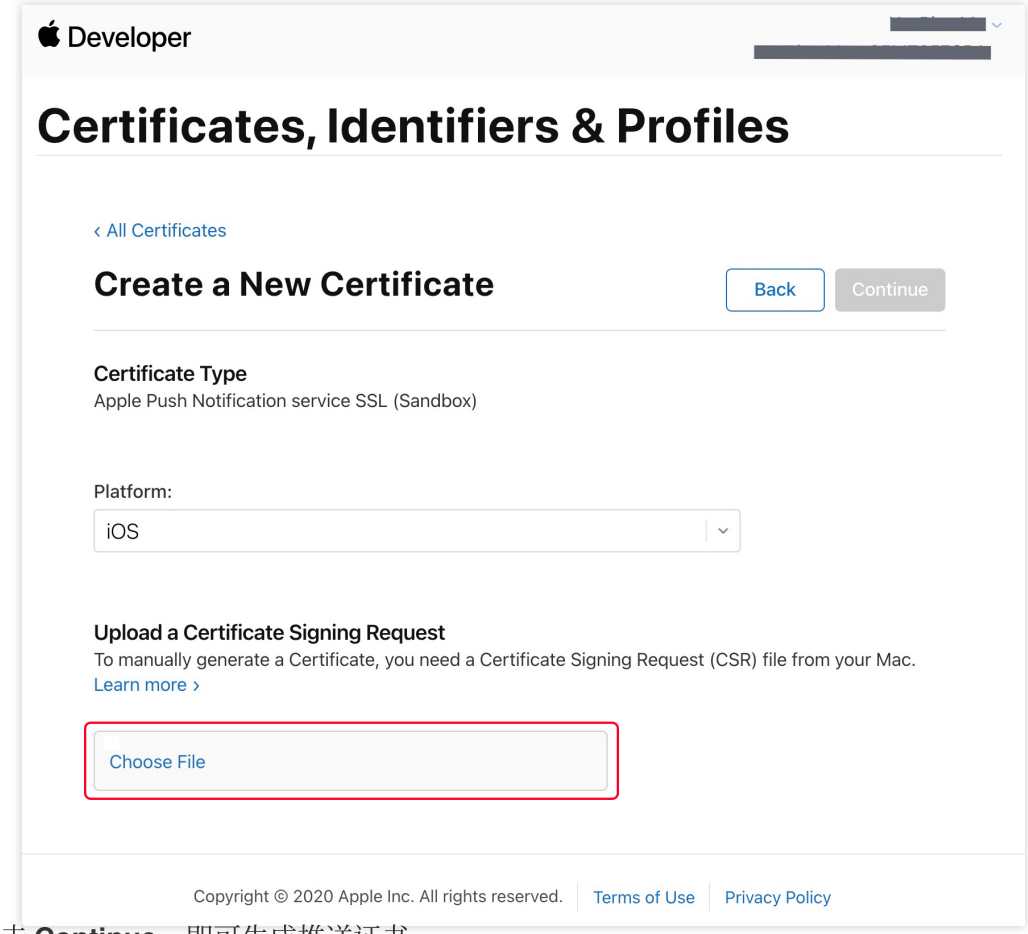


5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击**继续**，系统将生成一个 `*.certSigningRequest` 文件。



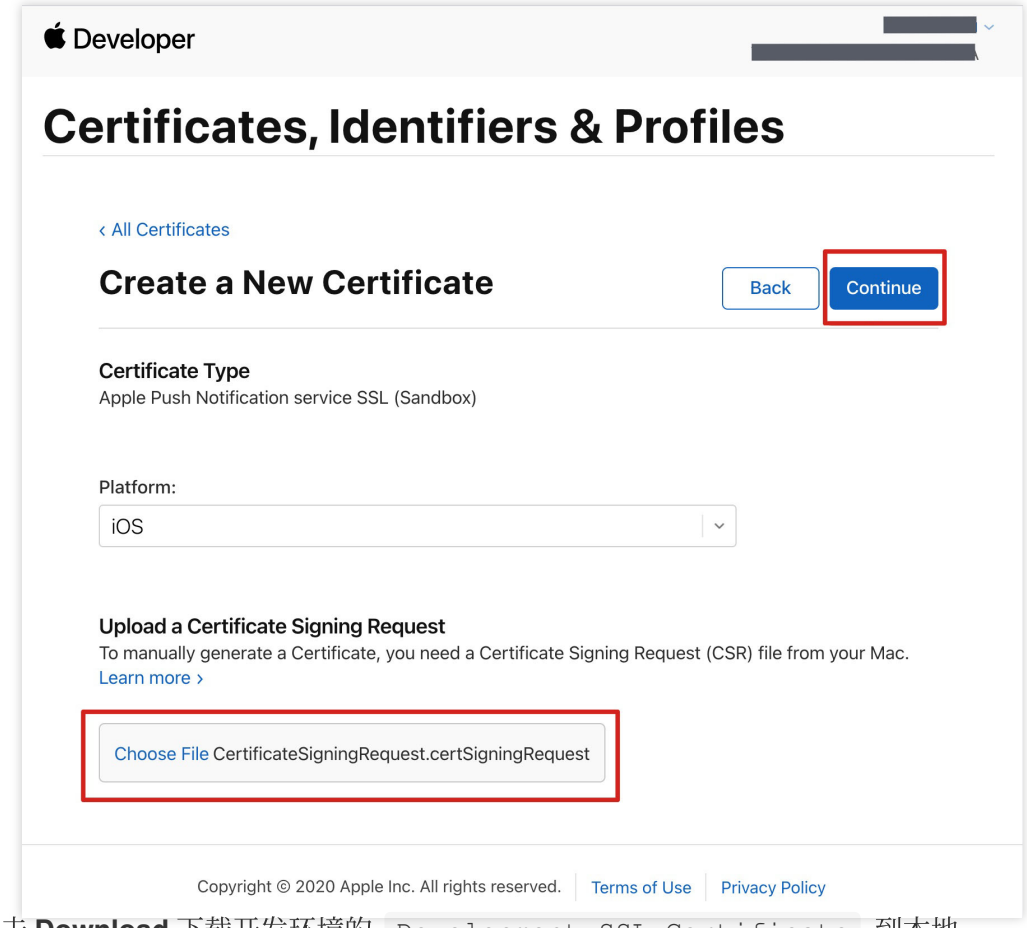
6. 返回上述 步骤3 中 Apple Developer 网站刚才的页面，单击 **Choose File** 上传生成的 `*.certSigningRequest` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.

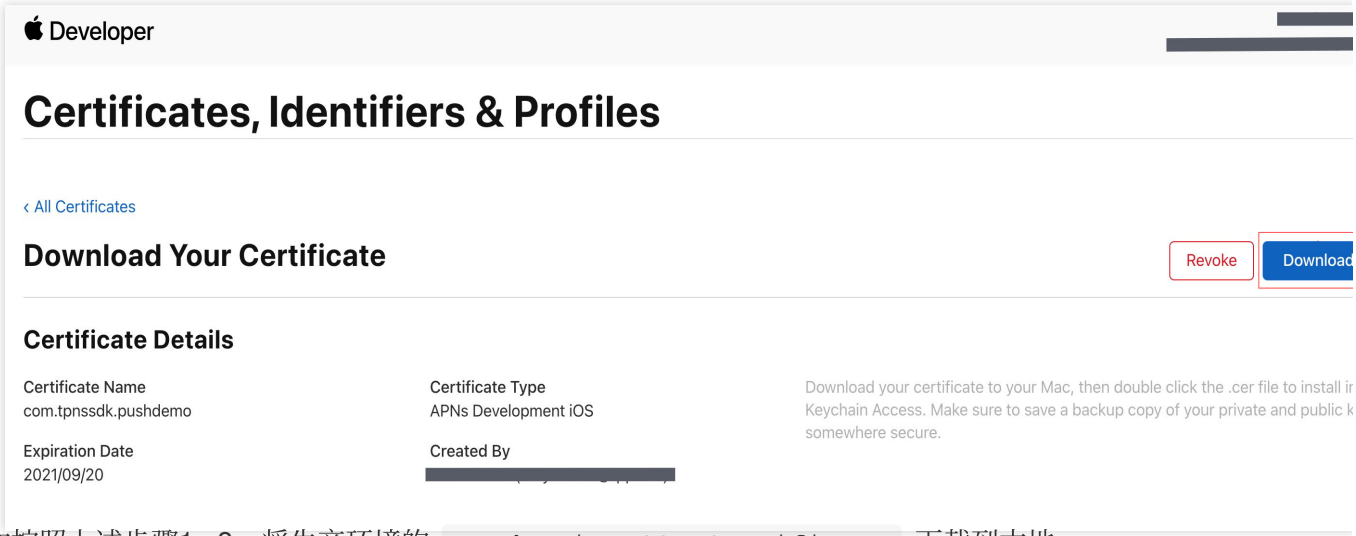


7. 单击 **Continue**，即可生成推送证书。

©2013-2025 Tencent Cloud International Pte. Ltd.



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

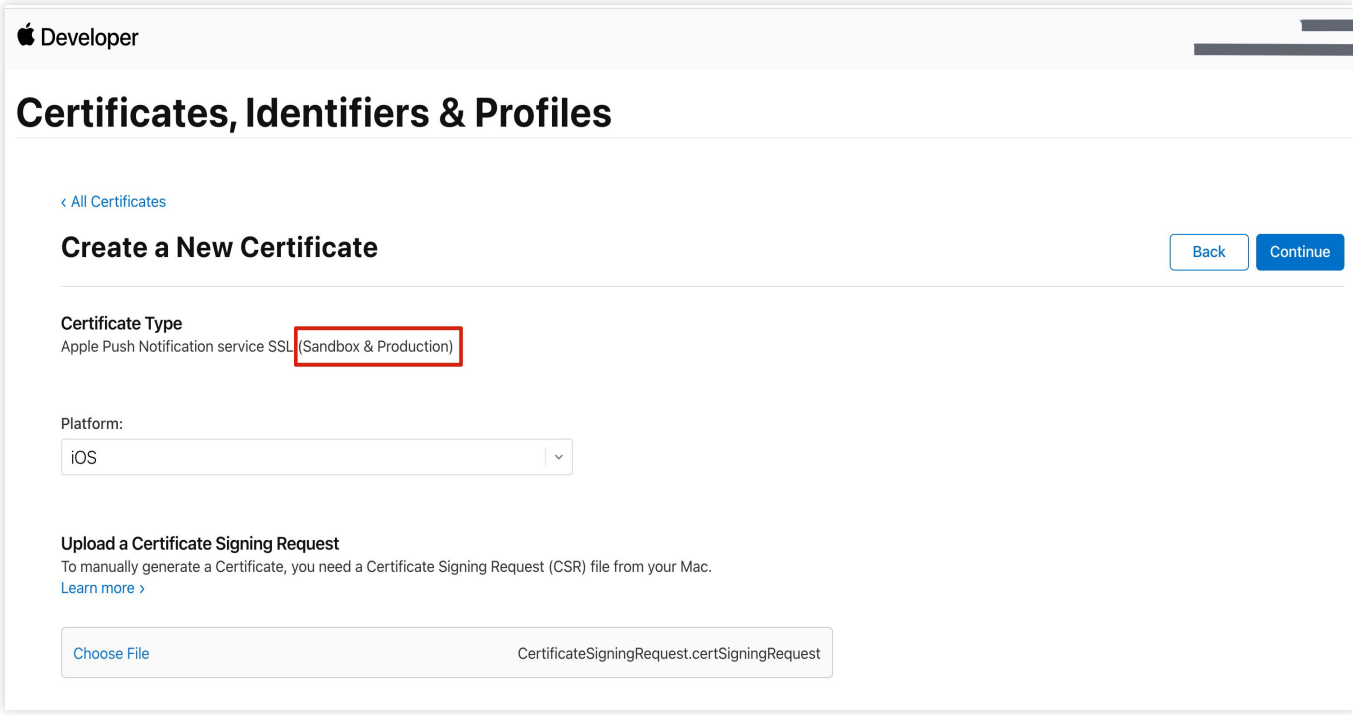


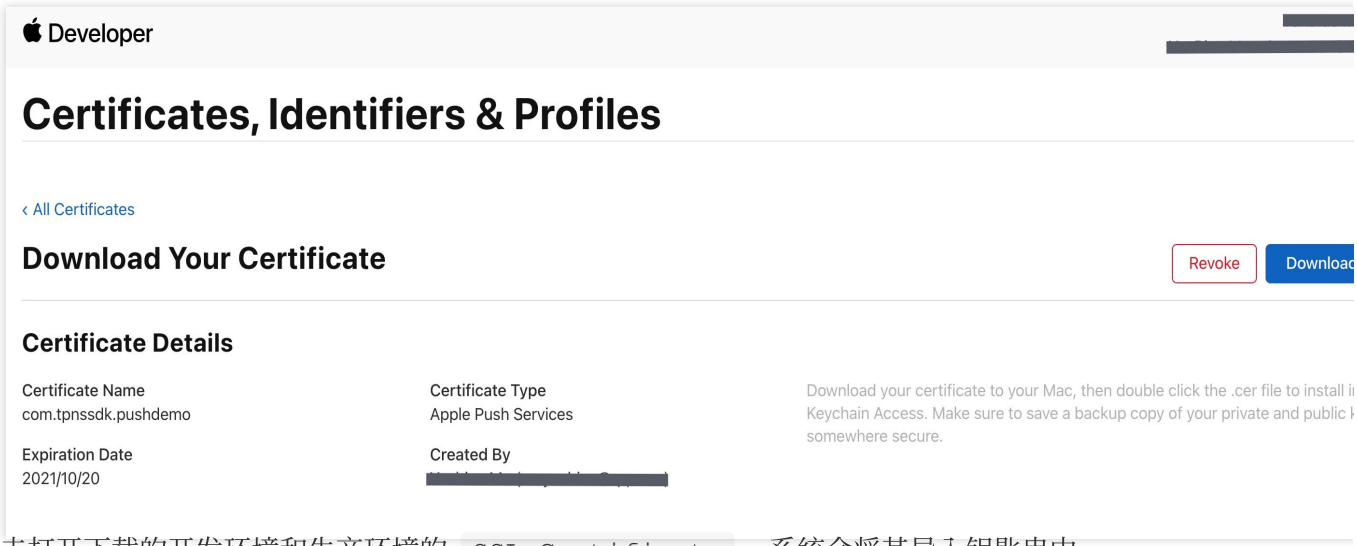
9. 再次按照上述步骤1 - 8，将生产环境的 `Production SSL Certificate` 下载到本地。

说明：

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

©2013-2025 Tencent Cloud International Pte. Ltd.

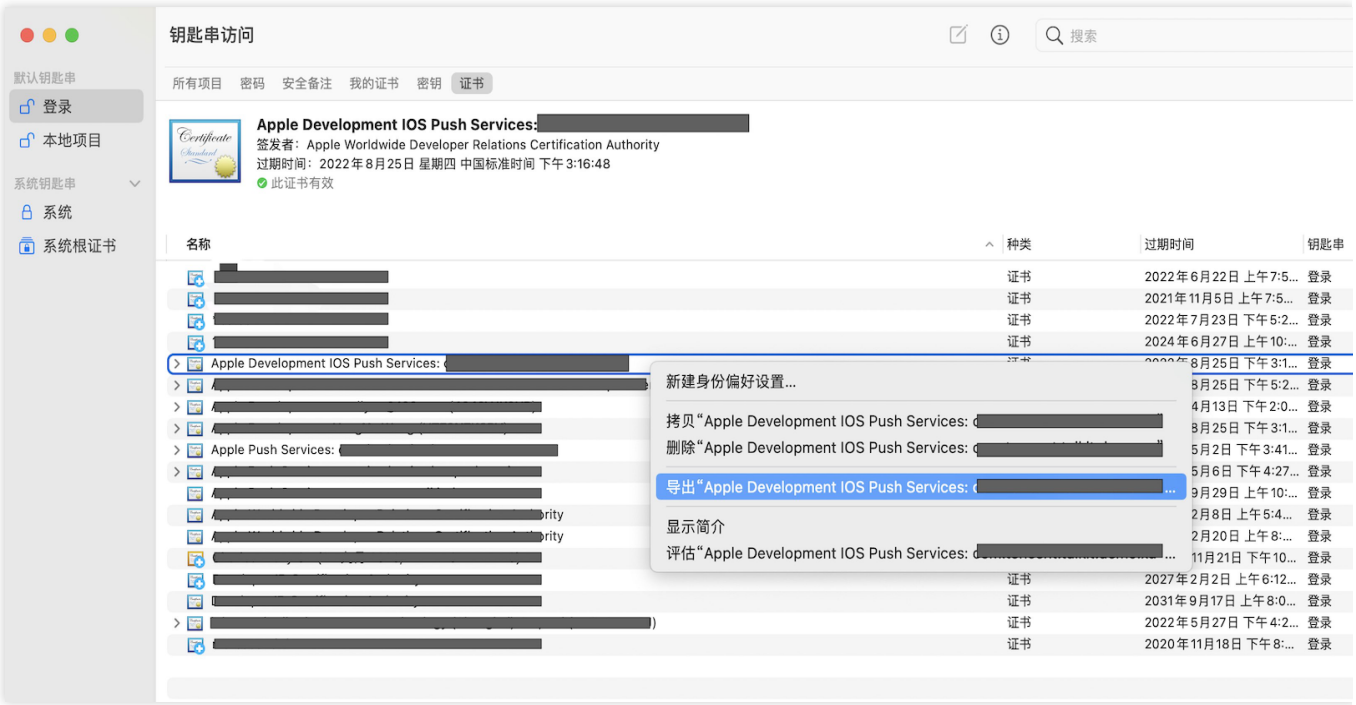




10. 双击打开下载的开发环境和生产环境的 `SSL Certificate`，系统会将其导入钥匙串中。

11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出刚创建的开发环境（`Apple Development IOS Push Service`）和生产环境（`Apple Push Services`）的 `p12` 文件。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意

保存 `.p12` 文件时，请务必为其设置密码。

步骤2：上传证书到控制台

1. 登录 即时通信 Chat 控制台。
2. 进入 **推送设置 > 厂商管理 > iOS**。
3. 点击**添加证书**。
4. 选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击**确认**。

©2013-2025 Tencent Cloud International Pte. Ltd.



注意：

上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。

上传证书需要设置密码，无密码收不到推送。

发布 App Store 的证书需要设置为生产环境，否则无法收到推送。

上传的 p12 证书必须是自己申请的真实有效的证书。

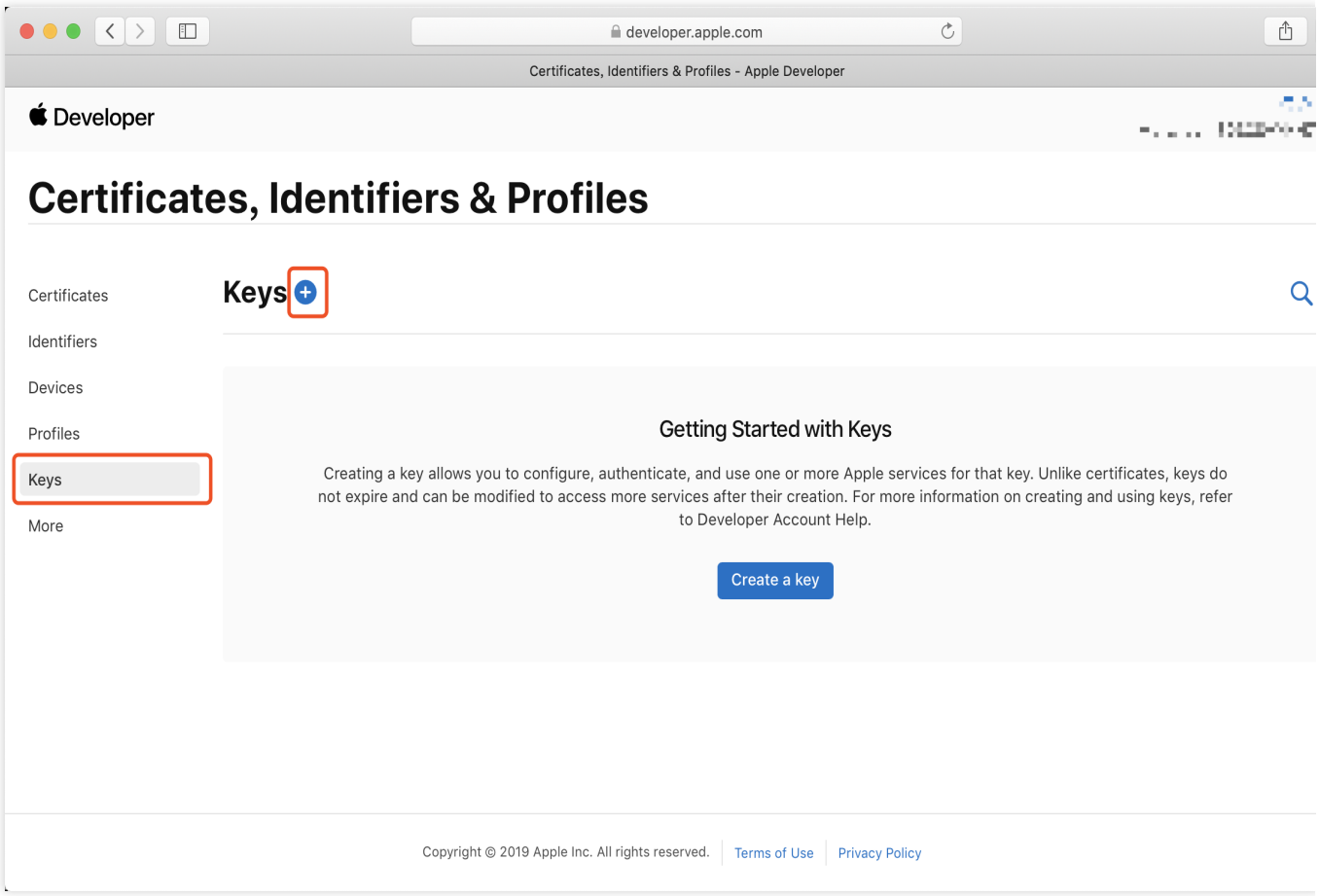
二、使用 p8 证书 （支持灵动岛推送）

p8 证书：p8 证书没有到期日期，因此您无需担心证书过期。此外，使用 p8 证书可以简化证书管理，因为您可以使用一个p8 证书为多个应用程序提供推送通知服务。另外，p8 证书支持灵动岛推送。

步骤1：申请 APNs 证书

要创建 p8 证书文件，首先需要登录 苹果开发者中心。

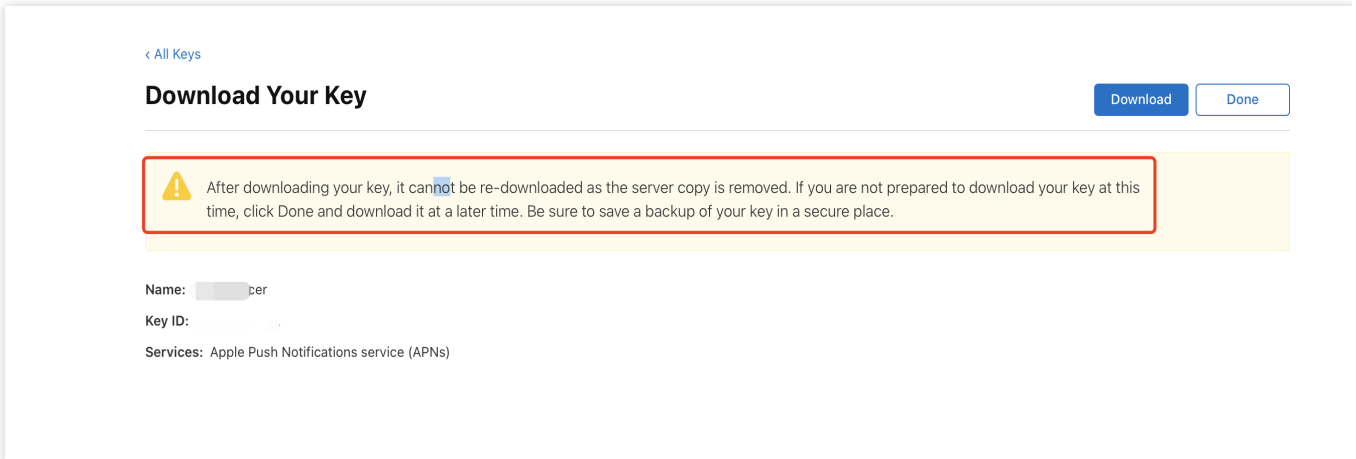
©2013-2025 Tencent Cloud International Pte. Ltd.



1. 进入Certificates，Identifiers & Profiles：在页面右上角单击 **Account**，然后在下拉菜单中选择 **Certificates, Identifiers & Profiles** 。

2. 创建一个新的 App ID：在左侧菜单中单击 **Identifiers** ，然后单击右侧的 **+** 创建一个新的 App ID。填写相应的信息并单击 **Continue** 。

3. 创建一个新的密钥：在左侧菜单中单击 **Keys**，然后单击右侧的 **+** 创建一个新的密钥。输入密钥的名称，然后勾选 **Apple Push Notifications service (APNs)**，单击 **Continue** 。



确认并生成密钥：在确认页面核对您的密钥信息，然后单击 **Register** 。接下来，您将看到一个页面提示您下载密钥。单击 **Download**，将生成的 .p8 文件保存到您的计算机上。

©2013-2025 Tencent Cloud International Pte. Ltd.

注意：

p8 证书只可以下载一次，请妥善保存。

请妥善保管下载的 p8 文件，因为您将无法再次下载该文件。您可以使用此P8证书配置您的iOS应用程序以接收推送通知。

步骤2：上传 p8 证书到Chat控制台

1. 登录 即时通信 Chat 控制台。
2. 进入 **推送设置 > 厂商管理 > iOS**。
3. 点击**添加证书**。
4. 选择 .p8 证书，上传 iOS 证书（.p8），设置 **KeyID、TeamID、BundleID**，单击**确认**。



说明：

KeyID：这是您的 APNs Auth Key 的唯一标识符。当您在 Apple Developer Center 创建一个新的 APNs Auth Key 时，系统会为您生成一个 Key ID。您可以在 "Certificates, Identifiers & Profiles" 部分的 "Keys" 中找到它。

TeamID：这是您的开发者账户的唯一标识符。您可以在 Apple Developer Center 的账户详情页面找到它。点击右上角的 "Membership"，在 "Membership Details" 部分可以找到您的 Team ID。

©2013-2025 Tencent Cloud International Pte. Ltd.

BundleID：这是您的应用程序的唯一标识符，也称为应用程序 ID。您可以在 Apple Developer Center 的 "Certificates, Identifiers & Profiles" 部分找到它。选择 "Identifiers"，然后在您的应用程序列表中找到对应的 Bundle ID。

操作步骤

步骤1：注册应用到厂商推送平台

离线推送需要将您自己的应用注册到各个厂商的推送平台，得到 AppID 和 AppKey 等参数，来实现离线推送功能。目前国内支持的手机厂商有：小米、华为、荣耀、OPPO、VIVO、魅族，境外支持 Google FCM。

步骤2：Chat 控制台配置

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

说明：

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

如需使用上报统计功能, 也请保持此项默认值不变,

小米

华为

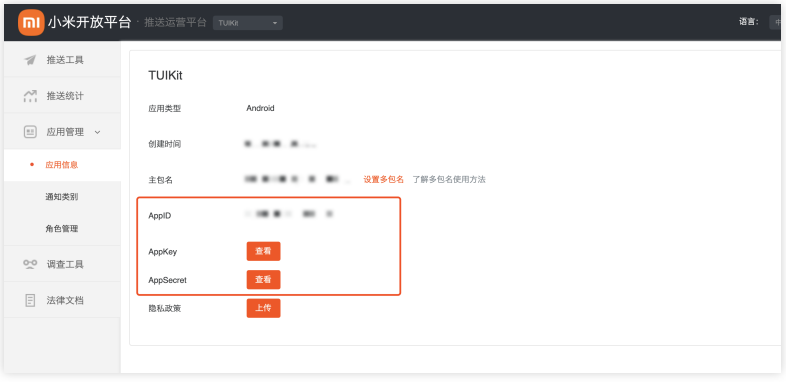
OPPO

vivo

魅族

荣耀

Google FCM

厂商推送平台	Chat 控制台配置
	

©2013-2025 Tencent Cloud International Pte. Ltd.



厂商推送平台	Chat 控制台配置
	

说明：

Client ID 对应 AppID，Client Secr

回执配置

©2013-2025 Tencent Cloud International Pte. Ltd.

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/huawei

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/huawei

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/huawei

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/huawei

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/huawei

China: https://api.im.qcloud.com/v3/offline_push_report/huawei

厂商推送平台	Chat 控制台配置
	

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.





回执配置

回执地址:

Singapore :https://apisgp.im.qcloud.com/v3/offline_push_report/vivo

Korea:https://apikr.im.qcloud.com/v3/offline_push_report/vivo

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/vivo

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/vivo

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/vivo

China: https://api.im.qcloud.com/v3/offline_push_report/vivo

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.





回执配置

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/meizu

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/meizu

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/meizu

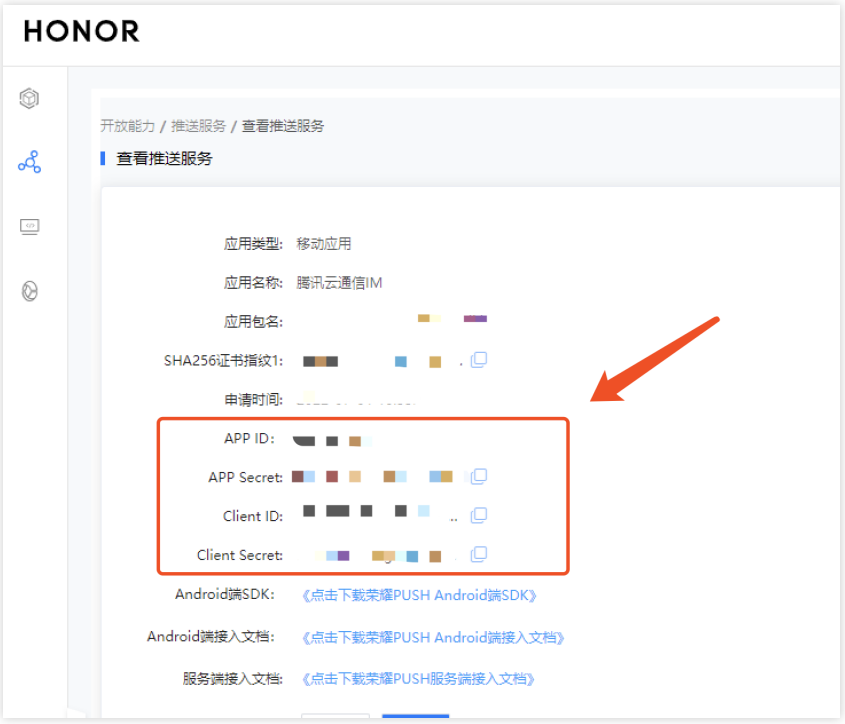
Germany: https://apiger.im.qcloud.com/v3/offline_push_report/meizu

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/meizu

China: https://api.im.qcloud.com/v3/offline_push_report/meizu

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.





回执配置

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/honor

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/honor

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/honor

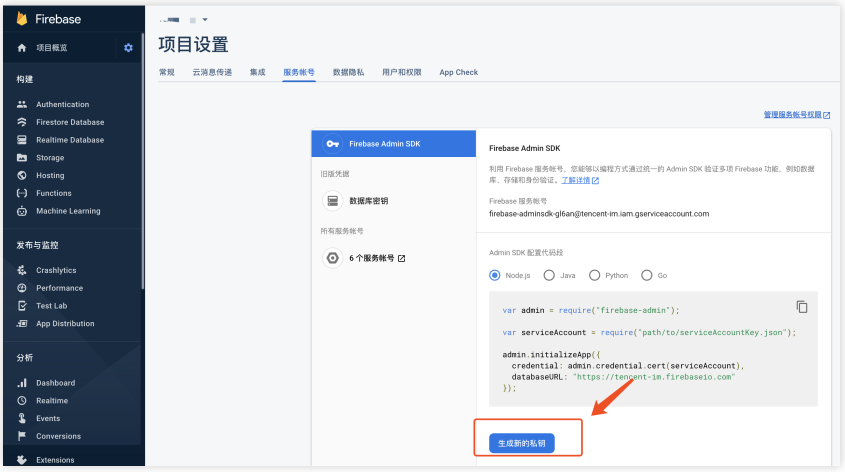
Germany: https://apiger.im.qcloud.com/v3/offline_push_report/honor

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/honor

China: https://api.im.qcloud.com/v3/offline_push_report/honor

厂商推送平台	Chat 控制台配置

©2013-2025 Tencent Cloud International Pte. Ltd.



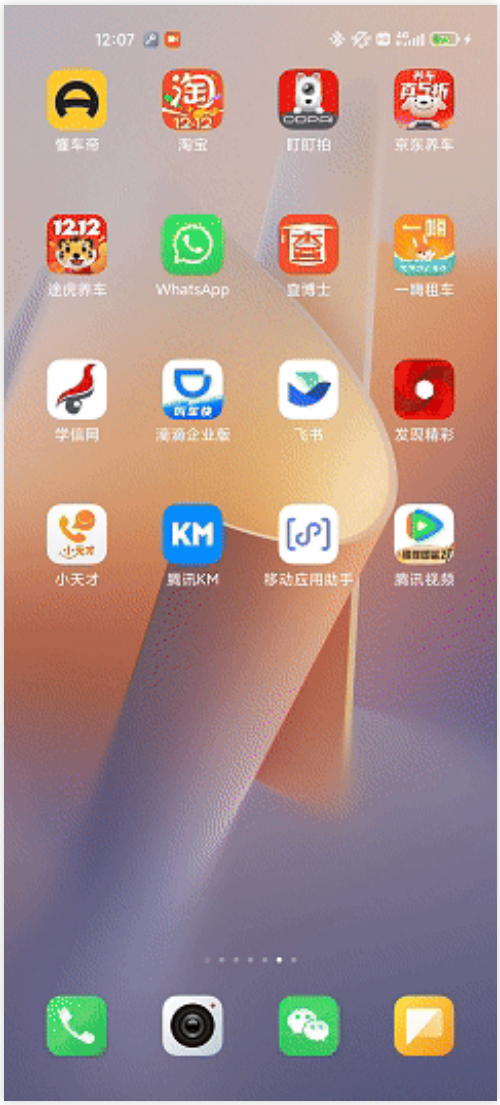


©2013-2025 Tencent Cloud International Pte. Ltd.

快速接入

最近更新时间：2025-03-03 14:34:31

效果展示

小米 11 Pro	iPhone 13
	

步骤1：创建一个 React Native 项目（已有项目可忽略此步骤）

```
npx @react-native-community/cli@latest init MyReactNativeApp --version 0.75.0
```

步骤2：进入 MyReactNativeApp 目录，集成 @tencentcloud/react-native-push

©2013-2025 Tencent Cloud International Pte. Ltd.

npm

yarn

```
npm install @tencentcloud/react-native-push --save
```

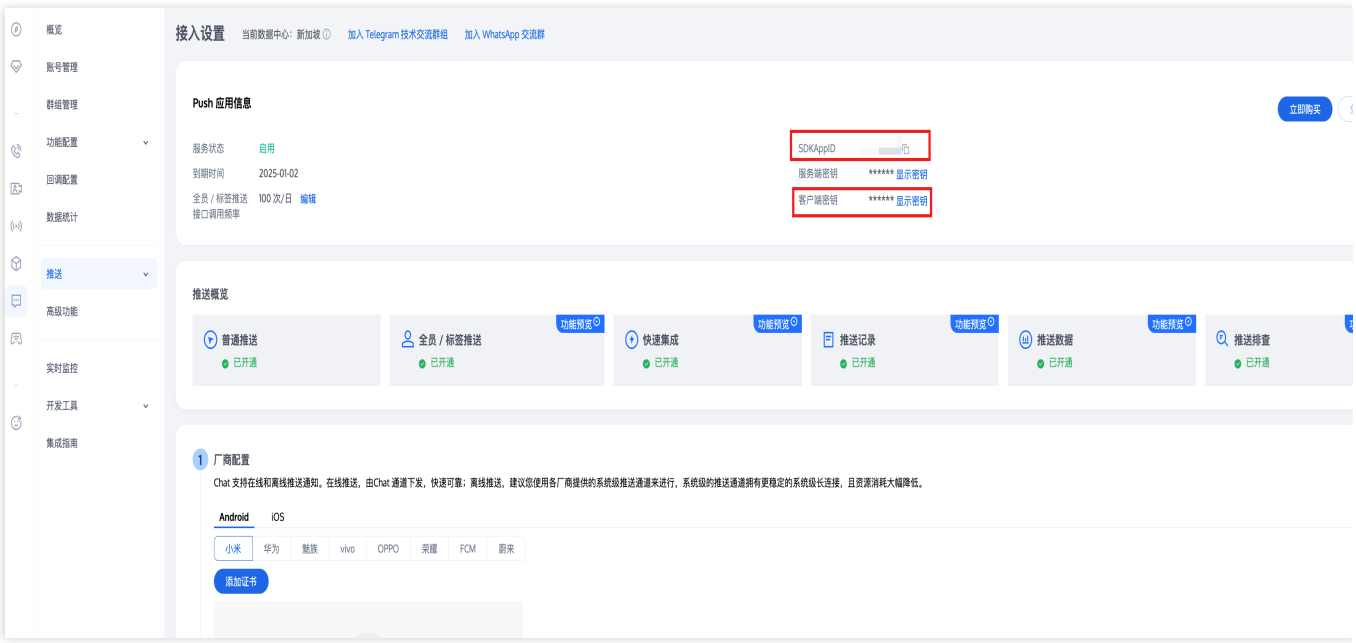
```
yarn add @tencentcloud/react-native-push
```

步骤3：注册推送

复制下面的代码到 `App.tsx`，并将 `SDKAppID` 和 `appKey` 替换为您的应用的信息。

说明：

`registerPush` 注册推送服务成功后，您可通过 `getRegistrationID` 获取推送 ID 标识, 即 RegistrationID。您可以向指定的 RegistrationID 推送消息。




```
import { TUILogin } from '@tencentcloud/tui-core';
import { TUIConversationService } from '@tencentcloud/chat-uikit-engine';
import Push from '@tencentcloud/react-native-push';

const SDKAppID = 0; // 您的 SDKAppID
const appKey = ''; // 客户端密钥
const userID = ''; // 您的 userID
const userSig = ''; // 对 UserID 加密后生成的签名串

TUILogin.login({
  SDKAppID,
  userID,
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  userSig,
  useUploadPlugin: true,
  framework: 'rn',
}).then(() => {
  if (Push) {
    Push.setRegistrationID(userID, () => {
      console.log('setRegistrationID ok', userID);
    });

    Push.registerPush(SDKAppID, appKey, (data) => {
        console.log('registerPush ok', data);
        Push.getRegistrationID((registrationID) => {
          console.log('getRegistrationID ok', registrationID);
        });
      }, (errCode, errMsg) => {
        console.error('registerPush failed', errCode, errMsg);
      }
    );

    // 监听通知栏点击事件，获取推送扩展信息
    Push.addPushListener(Push.EVENT.NOTIFICATION_CLICKED, (res) => {
      console.log('notification clicked', res);
      // 解析扩展信息，跳转到相应的会话
      try {
        const data = JSON.parse(res);
        const conv_type = data?.entity?.chatType === 1 ? 'C2C' : 'GROUP';
        TUIConversationService.switchConversation(`${conv_type}${data.entity.sender
        navigation.navigate('Chat');
      } catch (error) {
        console.log('error', error);
      }
    });

    // 监听在线推送
    Push.addPushListener(Push.EVENT.MESSAGE_RECEIVED, (res) => {
      // res 为消息内容
      console.log('message received', res);
    });

    // 监听在线推送被撤回
    Push.addPushListener(Push.EVENT.MESSAGE_REVOKED, (res) => {
      // res 为被撤回的消息 ID
      console.log('message revoked', res);
    });
  }
});
```

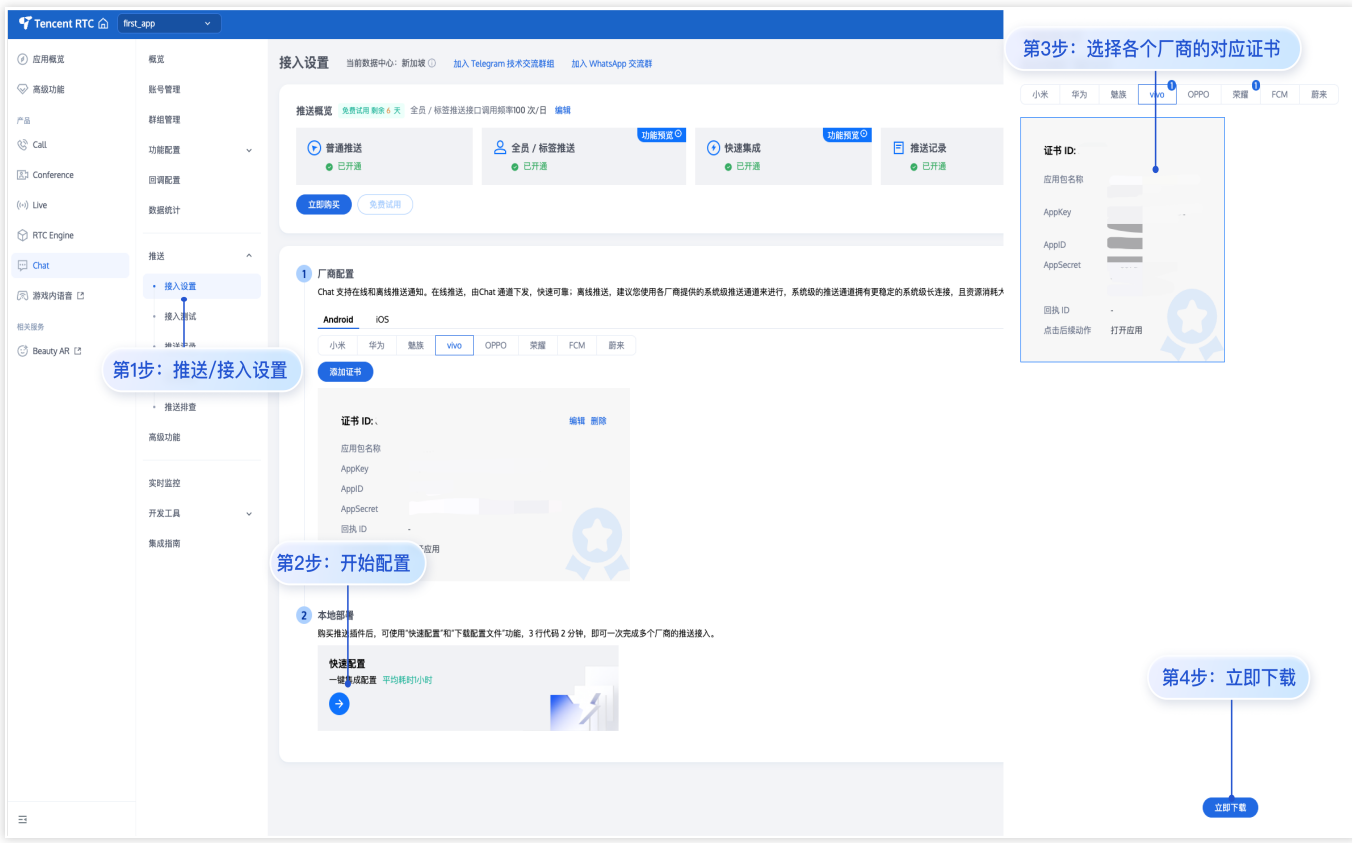
©2013-2025 Tencent Cloud International Pte. Ltd.

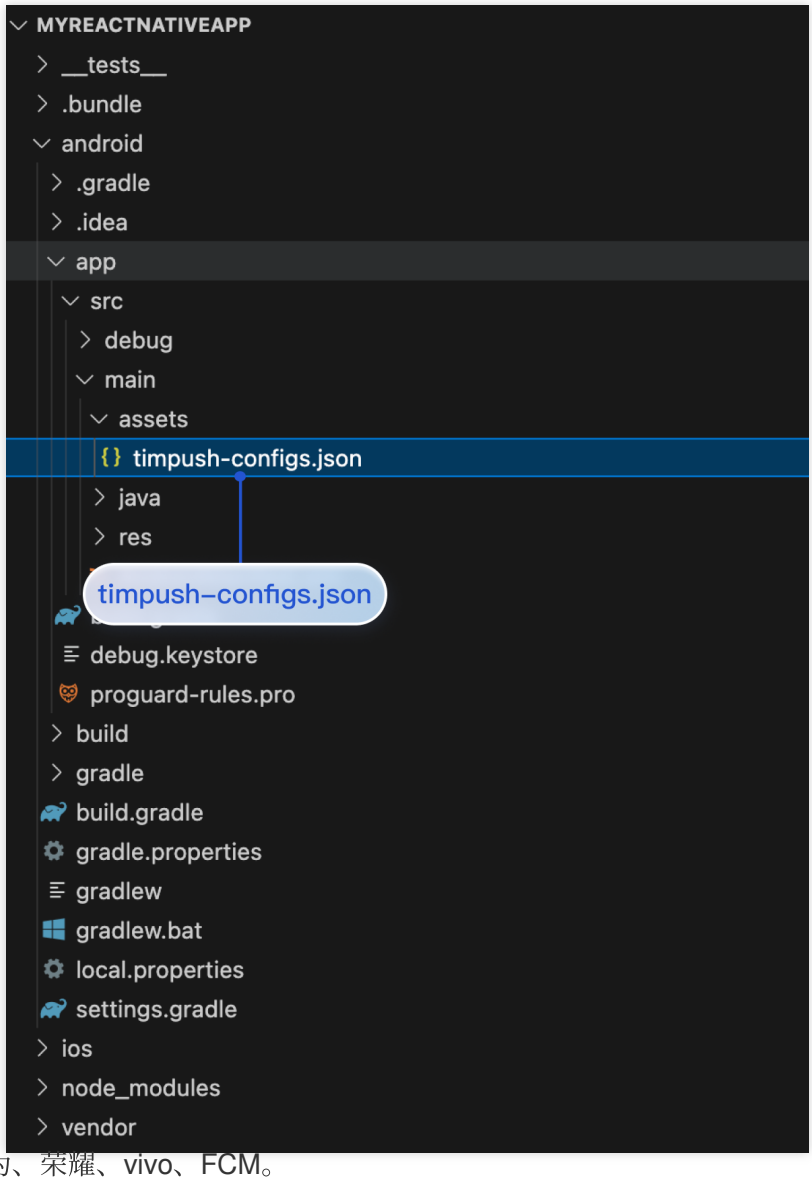
步骤4：配置厂商推送

Android

iOS

1. 请您完成控制台厂商推送信息填写后，从控制台下载 `timpush-configs.json` 文件，并添加到项目的 `MyReactNativeApp/android/app/src/main/assets` 目录下，如果该目录不存在，请手动创建。如图所示：





2. 华为、荣耀、vivo、FCM。

FCM

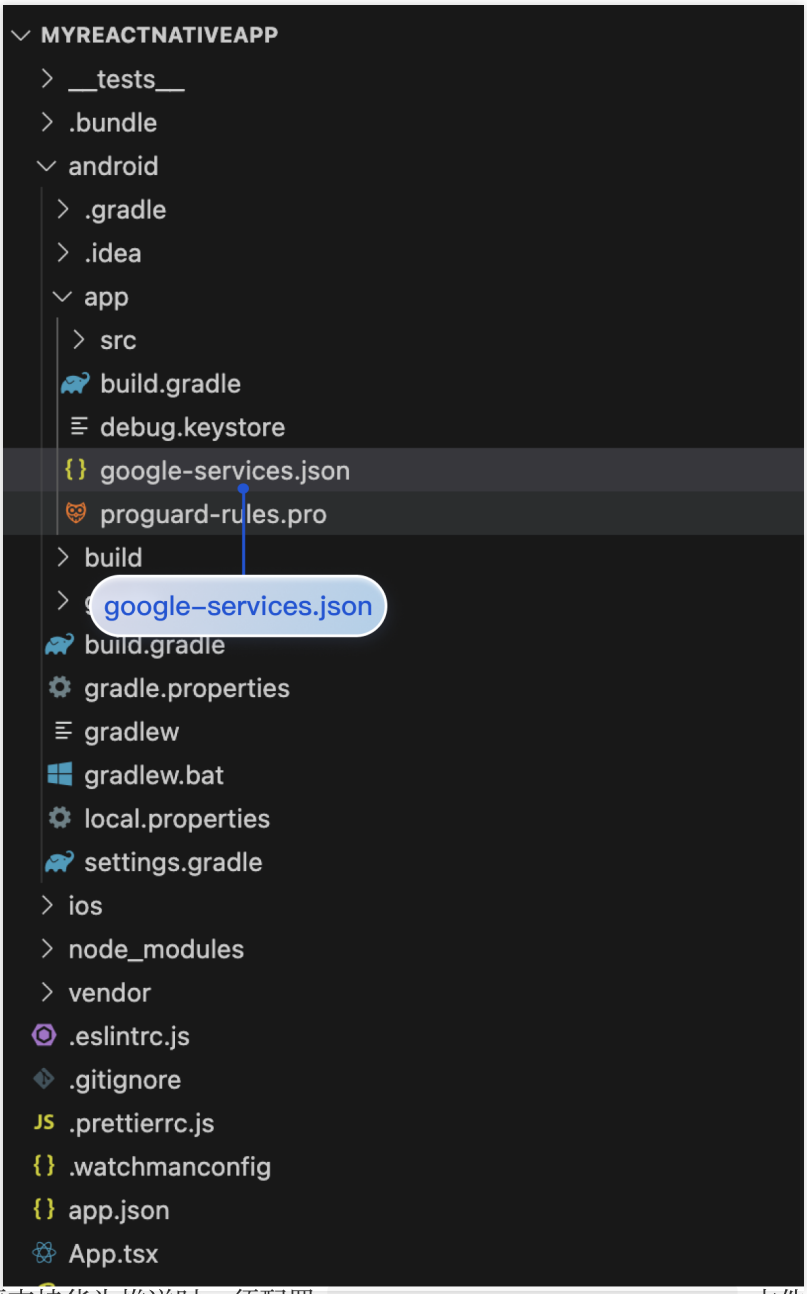
华为

荣耀

vivo

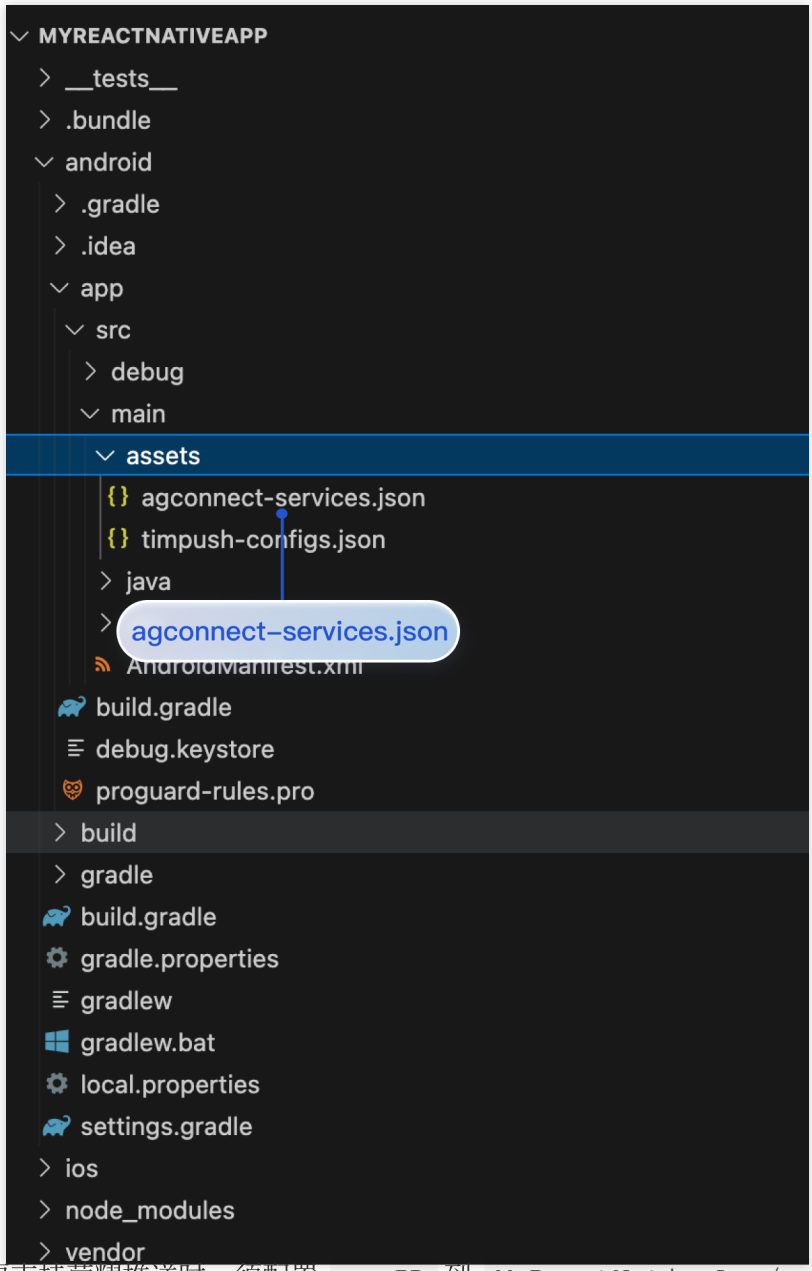
您需要支持 FCM 推送时，须配置 `google-services.json` 文件到 `MyReactNativeApp/android/app` 目录下（**请注意！不是 `MyReactNativeApp/android/app/src/main/assets` 目录**）。如图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.



您需要支持华为推送时，须配置 `agconnect-services.json` 文件到 `MyReactNativeApp/android/app/src/main/assets/` 目录下。如图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.

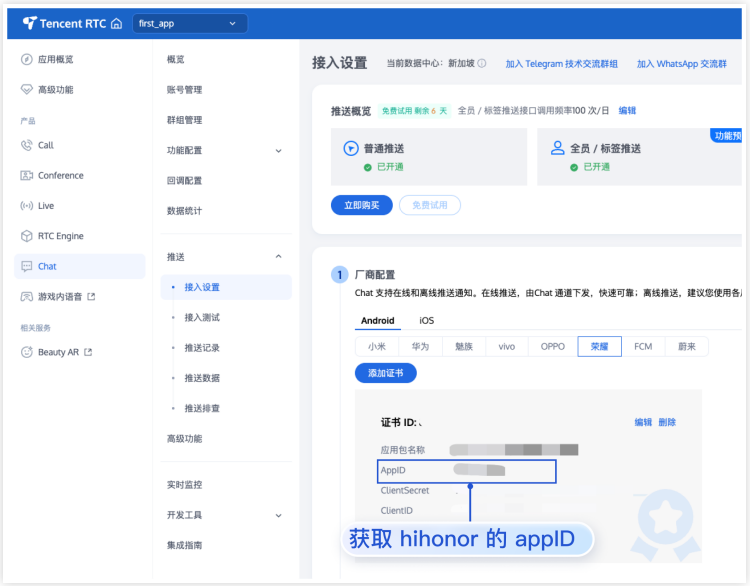


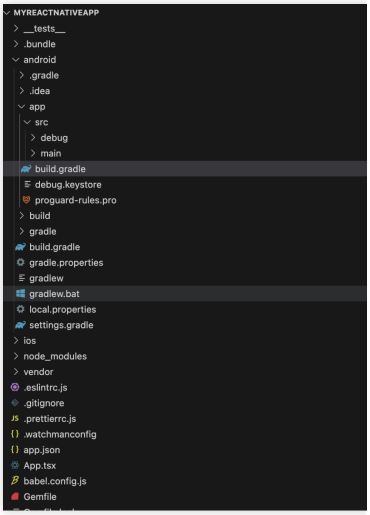
您需要支持荣耀推送时，须配置 `appID` 到 `MyReactNativeApp/android/app/build.gradle` 文件中。如图所示：

```
......
android {
    ......
    defaultConfig {
        ......
        manifestPlaceholders = [
          "HONOR_APPID" : ""
        ]
    }
}
```

获取荣耀的 appID	配置荣耀的 appID

©2013-2025 Tencent Cloud International Pte. Ltd.





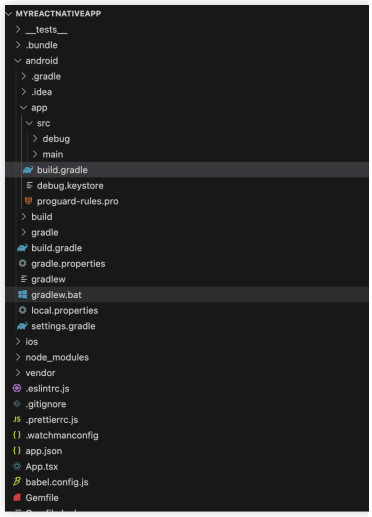
您需要支持 vivo 推送时，须配置 `appID` 和 `appKey` 到 `MyReactNativeApp/android/app/build.gradle` 文件中。如图所示：

```
......
android {
    ......
    defaultConfig {
        ......
        manifestPlaceholders = [
          "VIVO_APPKEY" : "0",
          "VIVO_APPID" : "0",
        ]
    }
}
```

获取 vivo 的 appID && appKey	配置 vivo 的 appID && appKey

©2013-2025 Tencent Cloud International Pte. Ltd.





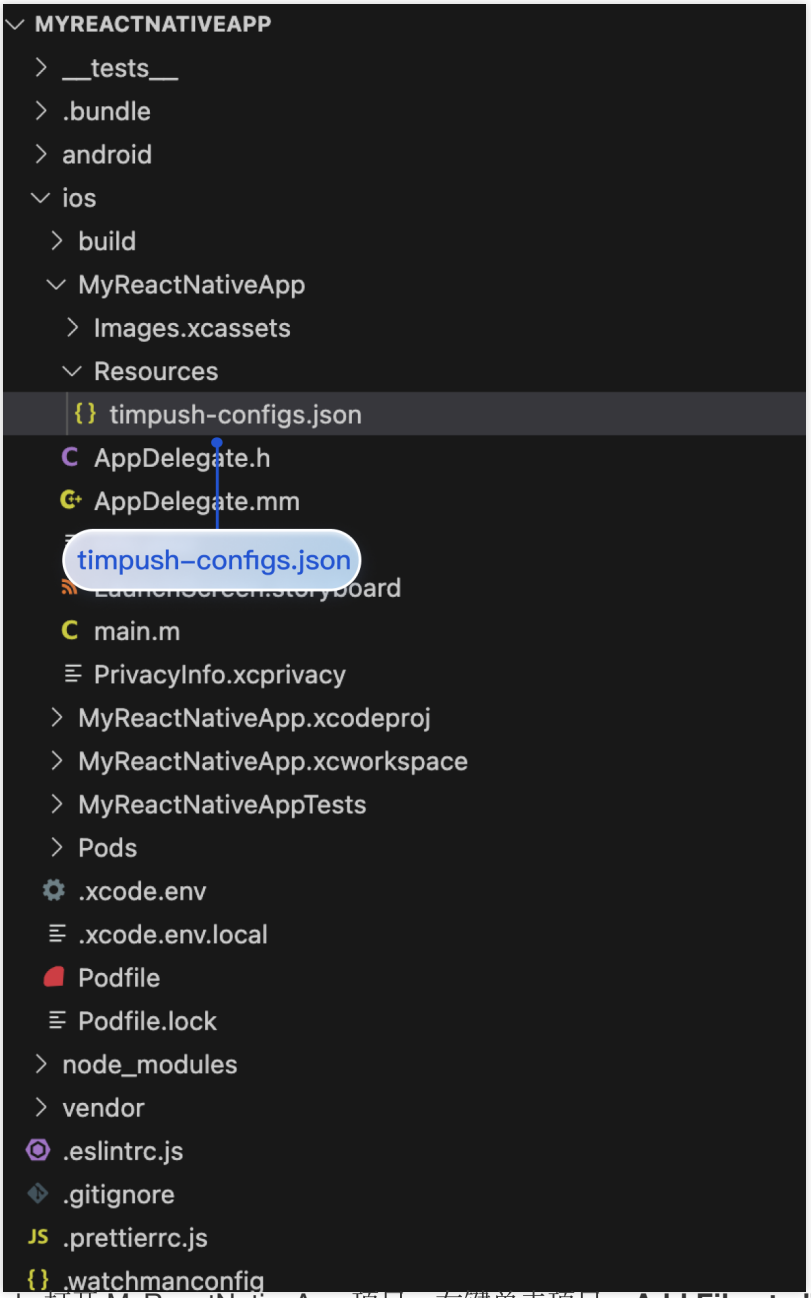
1. 请将您在厂商配置步骤中, 获取到的 iOS APNs 推送证书，上传至 Chat 控制台。Chat 控制台会为您分配一个证书 ID，如图所示：



©2013-2025 Tencent Cloud International Pte. Ltd.

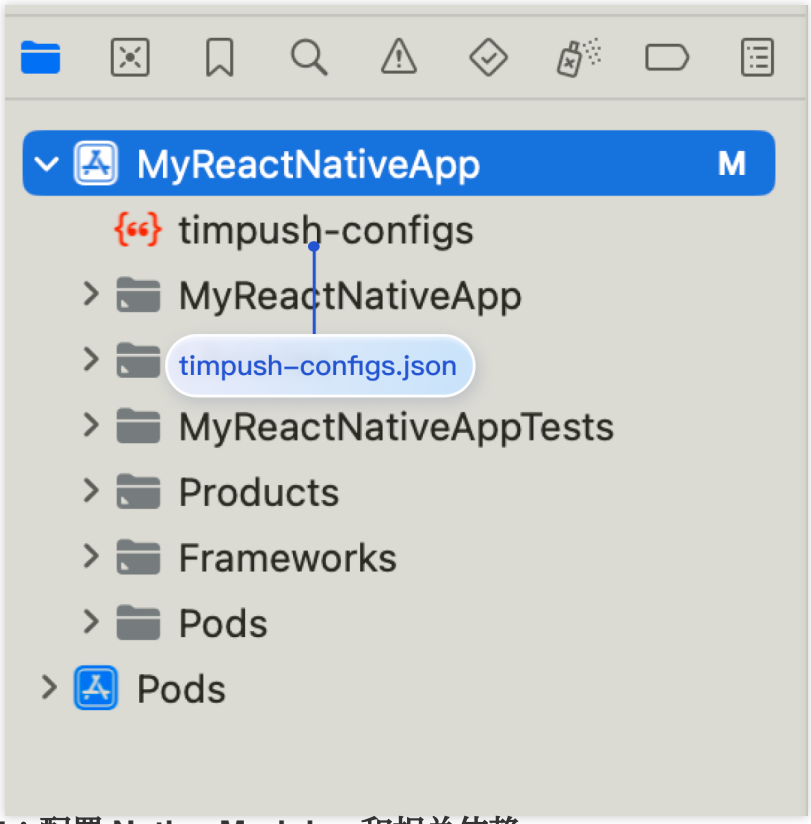
2. 在 `MyReactNativeApp/ios/MyReactNativeApp` 目录下，新建 `Resources` 文件夹，并新建 `timpush-configs.json` 文件。编辑 `timpush-configs.json`，填入控制台获取的证书 ID，如下所示：

```
{
  "businessID": "您的证书ID"
}
```



3. XCode 打开 MyReactNativeApp 项目，右键单击项目 > **Add Files to "MyReactNativeApp"**，将 `timpush-configs.json` 目录添加到工程。如图所示：

©2013-2025 Tencent Cloud International Pte. Ltd.



步骤5：配置 Native Modules 和相关依赖

Android

iOS

说明：

请确保 `timpush-configs.json` 文件内的包名和 `MyReactNativeApp/android/app/build.gradle` 文件内的 `applicationId` 值一致，不一致则会导致离线推送不可用。

1. 使用 Android Studio 打开 `MyReactNativeApp/android` 目录。

2. 修改项目入口文件。

项目入口文件为：MainApplication.kt

项目入口文件为：MainApplication.java

```
...
import com.tencent.qcloud.rntimpush.TencentCloudPushApplication

// Replace Application with TencentCloudPushApplication
class MainApplication : TencentCloudPushApplication(), ReactApplication {
  ...
  //  add TencentCloudPushPackage to the list of packages returned in ReactNativeHo
  override fun getPackages(): List<ReactPackage> =
    PackageList(this).packages.apply {
        // Packages that cannot be autolinked yet can be added manually here, for e
        // add(MyReactNativePackage())
    }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
```

```
...
import com.tencent.qcloud.rntimpush.TencentCloudPushApplication;

// Replace Application with TencentCloudPushApplication
public class MainApplication extends TencentCloudPushApplication implements ReactAp
  ...
  // add TencentCloudPushPackage to the list of packages returned in ReactNativeHos
  @Override
  protected List<ReactPackage> getPackages() {
    List<ReactPackage> packages = new PackageList(this).getPackages();
    // Packages that cannot be autolinked yet can be added manually here, for examp
    // packages.add(new MyReactNativePackage());
    return packages;
  }
  ...
```

3. 编辑 `android/build.gradle` 文件，更新 `repositories`， `dependencies` 和 `allprojects`。

```
buildscript {
    ...
    repositories {
        ...
        google()
        mavenCentral()
        maven { url 'https://mirrors.tencent.com/nexus/repository/maven-public/' }
        // 配置 HMS Core SDK 的 Maven 仓地址。
        maven { url 'https://developer.huawei.com/repo/' }
        maven { url 'https://developer.hihonor.com/repo' }
    }
    dependencies {
        ...
        // 如果您创建的项目 com.android.tools.build:gradle 未带版本号，请设置为 8.5.0
        // classpath("com.android.tools.build:gradle:8.5.0")
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.9.1.301'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
allprojects {
    repositories {
        mavenCentral()
        maven { url 'https://mirrors.tencent.com/nexus/repository/maven-public/' }
        // 配置 HMS Core SDK 的 Maven 仓地址。
        maven { url 'https://developer.huawei.com/repo/' }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        maven { url 'https://developer.hihonor.com/repo' }
    }
}
...
```

4. 编辑 `android/app/build.gradle` 文件，更新 `plugin` 和 `defaultConfig`。

```
...
// 如果您的 APP 需要 FCM 推送，请取消下面的注释
// apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
apply plugin: 'com.hihonor.mcs.asplugin'
...
android {
    ...
    defaultConfig {
        ...
        manifestPlaceholders = [
            "VIVO_APPKEY" : "0",
            "VIVO_APPID" : "0",
            "HONOR_APPID" : ""
        ]
    }
}
```

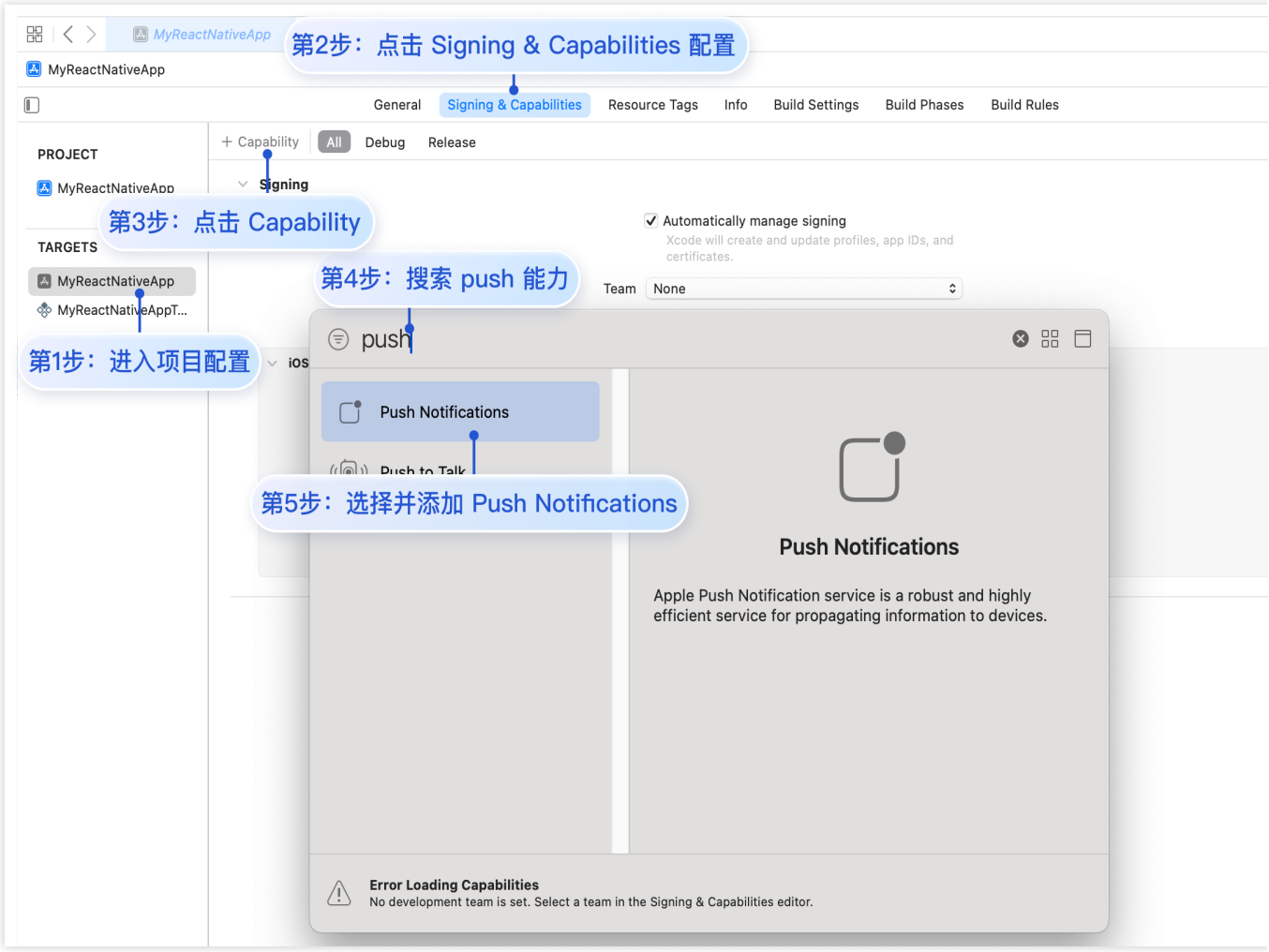
5. 以上操作都完成后，选择 **File > Sync Project with Gradle Files**。

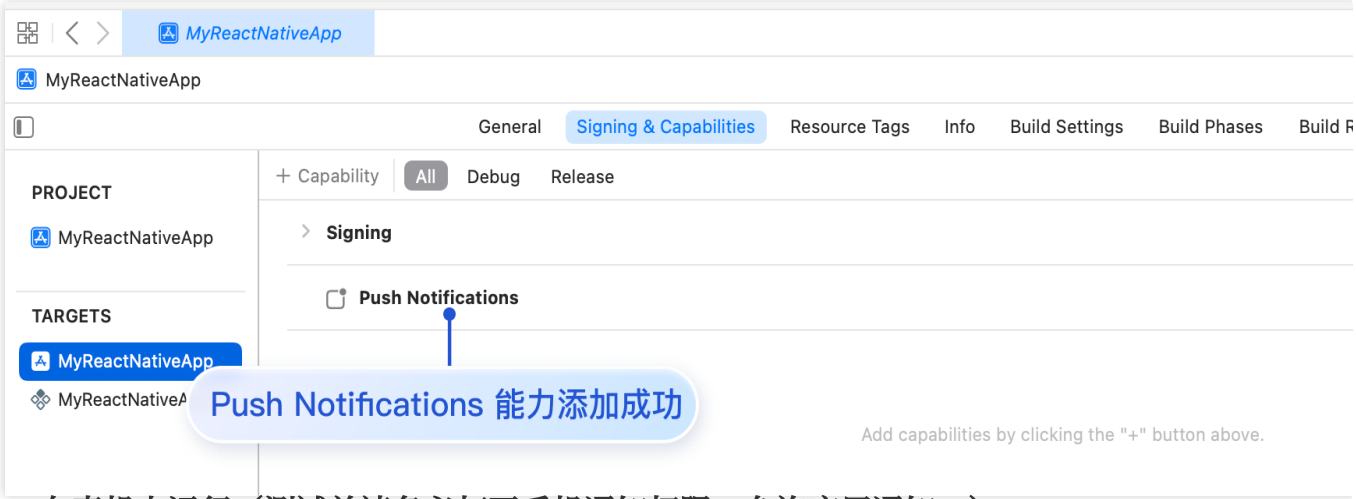
1. 使用 XCode 打开 **MyReactNativeApp/ios/MyReactNativeApp.xcworkspace**。
2. 进入 `MyReactNativeApp/ios` 目录，安装 TChatPush。

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

3. 在 App 中启用推送通知功能。打开 Xcode 项目，在 **Project > Target > Capabilities** 页面选择并添加 **Push Notifications**。

©2013-2025 Tencent Cloud International Pte. Ltd.





步骤6：在真机上运行（测试前请务必打开手机通知权限，允许应用通知。）

从项目根目录开始，在命令提示符中运行以下命令，在设备上安装并启动您的应用程序：

Android

iOS

```
npm run android
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
npm run ios
```

步骤7：消息推送触达统计

如果您需要统计触达数据，请按照如下完成配置：

华为

荣耀

vivo

魅族

iOS



回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/huawei

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/huawei

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/huawei

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/huawei

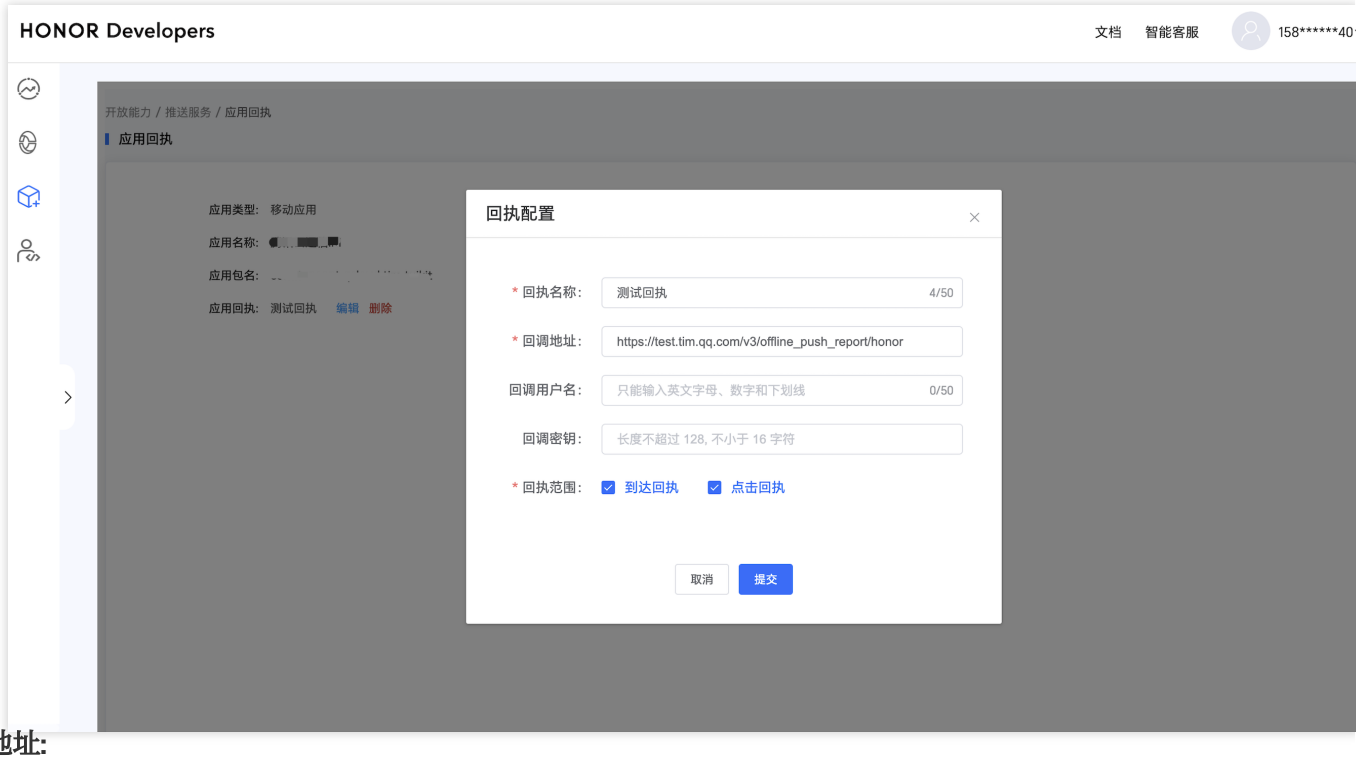
Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/huawei

China: https://api.im.qcloud.com/v3/offline_push_report/huawei

说明：

华为推送证书 ID <= 11344 ，使用华为推送 v2 版本接口，不支持触达和点击回执，请重新生成更新证书 ID。

©2013-2025 Tencent Cloud International Pte. Ltd.



回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/honor

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/honor

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/honor

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/honor

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/honor

China: https://api.im.qcloud.com/v3/offline_push_report/honor

回调地址配置	回执 ID 配置 Chat 控制台
	

回执地址:

Singapore :https://apisgp.im.qcloud.com/v3/offline_push_report/vivo

©2013-2025 Tencent Cloud International Pte. Ltd.

Korea:https://apikr.im.qcloud.com/v3/offline_push_report/vivo USA: https://apiusa.im.qcloud.com/v3/offline_push_report/vivo Germany: https://apiger.im.qcloud.com/v3/offline_push_report/vivo Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/vivo China: https://api.im.qcloud.com/v3/offline_push_report/vivo	

打开回执开关	配置回执地址
	

回执地址:

Singapore : https://apisgp.im.qcloud.com/v3/offline_push_report/meizu

Korea: https://apikr.im.qcloud.com/v3/offline_push_report/meizu

USA: https://apiusa.im.qcloud.com/v3/offline_push_report/meizu

Germany: https://apiger.im.qcloud.com/v3/offline_push_report/meizu

Indonesia: https://apiidn.im.qcloud.com/v3/offline_push_report/meizu

China: https://api.im.qcloud.com/v3/offline_push_report/meizu

说明：

打开回执开关后，请务必确保回执地址正确配置。不配置或者配置地址错误，都会影响推送功能。

iOS 端推送触达统计配置，详细请参见 **统计推送抵达率**。

其余支持厂商不需要配置，FCM 暂不支持推送统计功能。

©2013-2025 Tencent Cloud International Pte. Ltd.

客户端 API

最近更新时间：2025-03-03 14:34:31

接口概览

API	描述
`registerPush`	注册推送服务，(必须在 App 用户同意了隐私政策后，再调用该接口使用推送服务)。
`unRegisterPush`	反注册关闭推送服务。
`setRegistrationID`	设置推送设备标识 ID。 RegistrationID 是推送接收设备的唯一标识 ID。默认情况下，注册推送服务成功时自动生成该 ID，同时也支持您自定义设置。**请注意！如您调用此接口，请务必在 `registerPush` 前调用。** 适用场景举例： 若您同时集成了消息服务（Chat）和推送服务（Push），在某个设备上，用户登录 Chat 的 userID 假设为 "user123"，如果您想指定向 "user123" 推送消息，则需要调用此接口设置设备标识 ID，如下： `Push.setRegistrationID("user123", () => {});`
`getRegistrationID`	在成功注册推送服务后，调用此接口可获取推送接收设备的唯一标识 ID，即 RegistrationID。
`getNotificationExtInfo`	收到离线推送时，点击通知栏拉起 App，调用此接口可获取推送扩展信息。
`addPushListener`	添加 Push 监听器。
`removePushListener`	移除 Push 监听器。
`disablePostNotificationInForeground`	应用在前台时，开/关通知栏通知（默认开）。
`createNotificationChannel`	创建客户端通知 channel。此接口可在 Android 平台上实现自定义铃音功能。

接口详情

©2013-2025 Tencent Cloud International Pte. Ltd.

注册推送服务

接口

```
registerPush(SDKAppID: number, appKey: string, onSuccess: (data: string) => void, o
```

参数说明：

参数	类型	说明	获取路径
SDKAppID	Number	推送服务 Push 的 SDKAppID。	Chat 控制台 
appKey	String	推送服务 Push 的客户端密钥。	
onSuccess	Function	注册推送成功的回调。	-
onError	Function | undefined	注册推送失败的回调。	-

反注册关闭推送服务

接口

```
unRegisterPush(onSuccess: () => void, onError?: (errCode: number, errMsg: string) =
```

参数说明：

参数	类型	说明
onSuccess	Function	反注册推送成功的回调。
onError	Function | undefined	反注册推送失败的回调。

设置推送 ID 标识 RegistrationID

说明：

©2013-2025 Tencent Cloud International Pte. Ltd.

需要在注册推送服务之前调用。

接口

```
setRegistrationID(registrationID: string,  onSuccess: () => void): void;
```

参数说明：

参数	类型	说明
registrationID	String	自定义的推送 ID 标识。
onSuccess	Function	设置自定义推送 ID 标识成功后的回调。

获取推送 ID 标识 RegistrationID

说明：

需要在注册推送服务成功之后调用。

若您调用过 `setRegistrationID` 接口设置标识 ID，此接口将返回您设置的标识 ID，否则返回由 Push SDK 生成的随机值。

接口

```
getRegistrationID(onSuccess: (registrationID: string) => void): void;
```

参数说明：

参数	类型	说明
onSuccess	Function	获取推送 ID 标识成功的回调。

获取点击透传的内容

说明：

收到离线推送时，点击通知栏拉起 App，调用此接口可获取推送扩展信息。

接口

```
getNotificationExtInfo(onSuccess: (extInfo: string) => void): void;
```

参数说明：

参数	类型	说明

©2013-2025 Tencent Cloud International Pte. Ltd.

onSuccess	Function	获取点击透传的内容成功的回调。

添加 Push 监听器。

接口

```
addPushListener(eventName: string, listener: (data: any) => void);
```

参数说明：

参数	类型	说明
eventName	String	推送事件类型。
listener	Function	推送事件处理方法。

移除 Push 监听器。

接口

```
removePushListener(eventName: string, listener?: (data: any) => void);
```

参数说明：

参数	类型	说明
eventName	String	推送事件类型。
listener	Function | undefined	推送事件处理方法。

应用在前台时，开/关通知栏通知。

接口

```
disablePostNotificationInForeground(disable: boolean);
```

参数说明：

参数	类型	说明
disable	boolean	应用在前台时，开/关通知栏通知，默认开 true: 应用在前台时，关闭通知栏通知。

©2013-2025 Tencent Cloud International Pte. Ltd.

		false: 应用在前台时，开启通知栏通知。

创建客户端通知 channel。

接口

```
createNotificationChannel(options: any, onSuccess: (data: any) => void);
```

参数说明：

参数	类型	说明
options.channelID	String	自定义 channel 的 ID。 OPPO：控制台-> 接入配置 中 channelID。
options.channelName	String	自定义 channel 的名称。
options.channelDesc	String | undefined	自定义 channel 的描述。
options.channelSound	String | undefined	自定义 channel 的铃音，音频文件名，不带后缀，音频文件需要放到 `MyReactNativeApp/android/app/src/main/res/raw` 中。 例如： `options.channelSound = private_ring`，即设置 `MyReactNativeApp/android/app/src/main/res/raw/private` 为自定义铃音
onSuccess	Function	接口调用成功的回调函数。

©2013-2025 Tencent Cloud International Pte. Ltd.

用户在线状态

Android

最近更新时间：2025-06-20 16:13:28

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”功能仅专业版套餐、体验版套餐、专业版plus和企业版套餐支持，使用前请确认已开通专业版、专业版plus或企业版套餐。
“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。
会话列表里只有好友才支持显示在线状态。如果您需要支持非好友显示在线状态，需要主动订阅非好友状态并调用 API 自行实现 UI，API 参考：用户状态。

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 。

```
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，首先请开通专业版、专业版Plus或企业版套餐包，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”

©2013-2025 Tencent Cloud International Pte. Ltd.

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，首先请开通专业版、专业版Plus或企业版套餐包，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，请升级到专业版、专业版plus或企业版套餐。

“用户在线状态”功能仅专业版、体验版、专业版plus和企业版套餐包支持，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 开通专业版、专业版plus或企业版套餐包进行体验。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-06-20 16:13:28

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”功能仅专业版、专业版plus或企业版套餐支持，使用前请确认已开通专业版、专业版plus或企业版套餐。
“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。
会话列表里只有好友才支持显示在线状态。如果您需要支持非好友显示在线状态，需要主动订阅非好友状态并调用 API 自行实现 UI，API 参考：用户状态。

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 。
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。
Swift
Objective-C

```
TUIConfig.default().displayOnlineStatusIcon = true
```

```
[TUIConfig defaultConfig].displayOnlineStatusIcon = YES;
```

效果展示

会话列表

Enabling "Show User Online Status"	Disabling "Show User Online Status"

©2013-2025 Tencent Cloud International Pte. Ltd.

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，请升级到专业版、专业版plus或企业版套餐。

“用户在线状态”功能仅专业版、专业版plus或企业版套餐包支持，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 开通专业版、专业版plus或企业版套餐包进行体验。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp (Vue)

最近更新时间：2025-03-20 21:18:45

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”功能仅体验版、专业版、专业版plus和企业版支持，使用前请确认已开通体验版、专业版、专业版plus或企业版套餐。

“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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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。

`TUIConversation` 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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

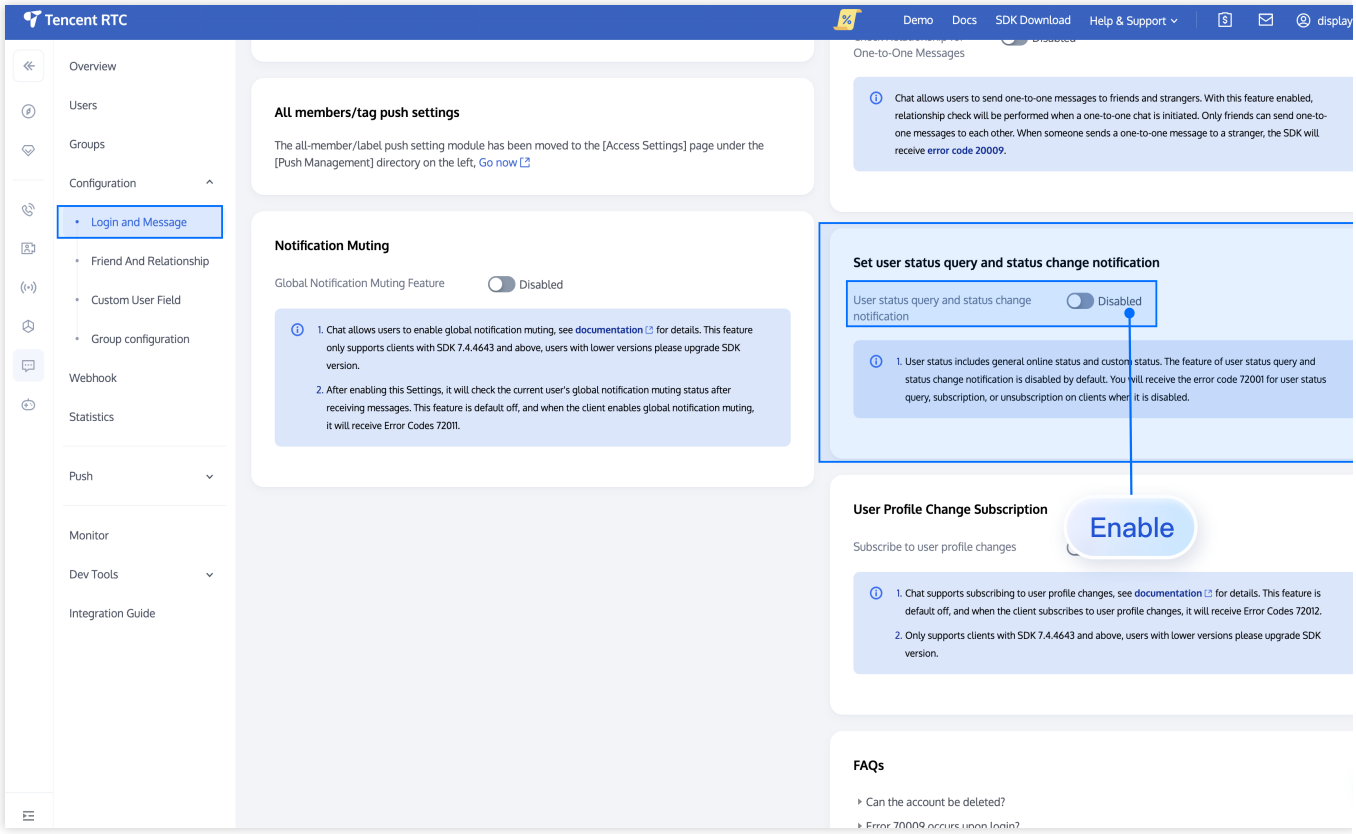
```
<div
  v-if="props.displayOnlineStatus"
  :class="{
    'online-status': true,
    'online-status-online': isOnline,
    'online-status-offline': !isOnline
  }"
></div>
```

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



Error: 套餐包不支持该接口的使用，请升级到专业版、专业版Plus或企业版套餐

“用户在线状态”功能仅专业版、专业版Plus或企业版套餐支持，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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-02-18 17:33:39

功能描述

支持在会话列表及联系人列表，展示用户在线状态。
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。
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：

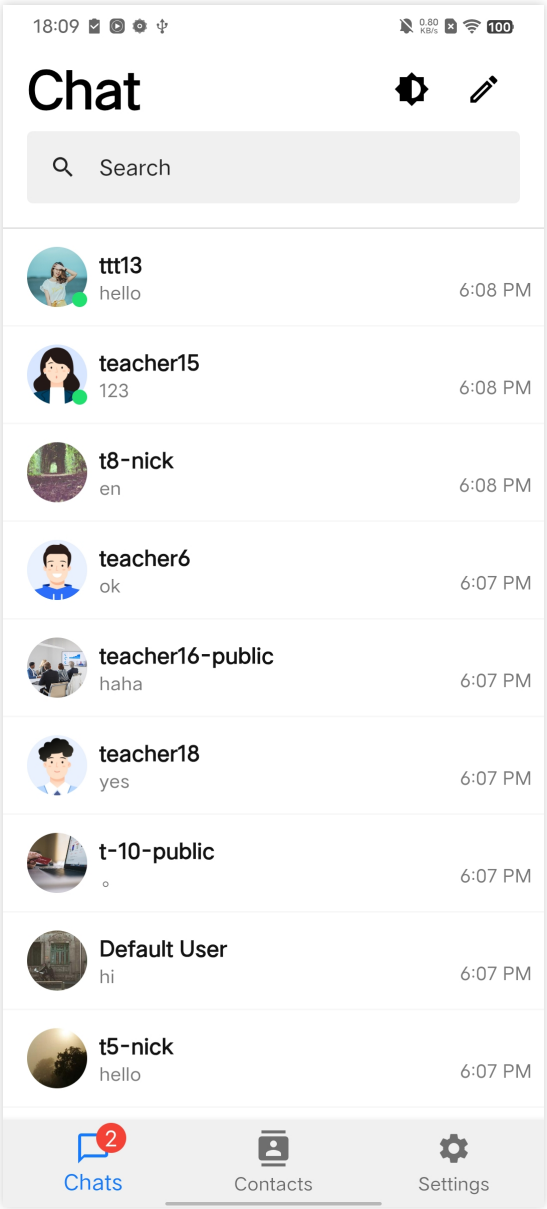
1. 仅专业版、专业版 plus 和企业版套餐支持，使用前请购买专业版、专业版plus或企业版。
2. 需要在 Chat 控制台 打开用户状态开关，使用前请确认开关已经打开。

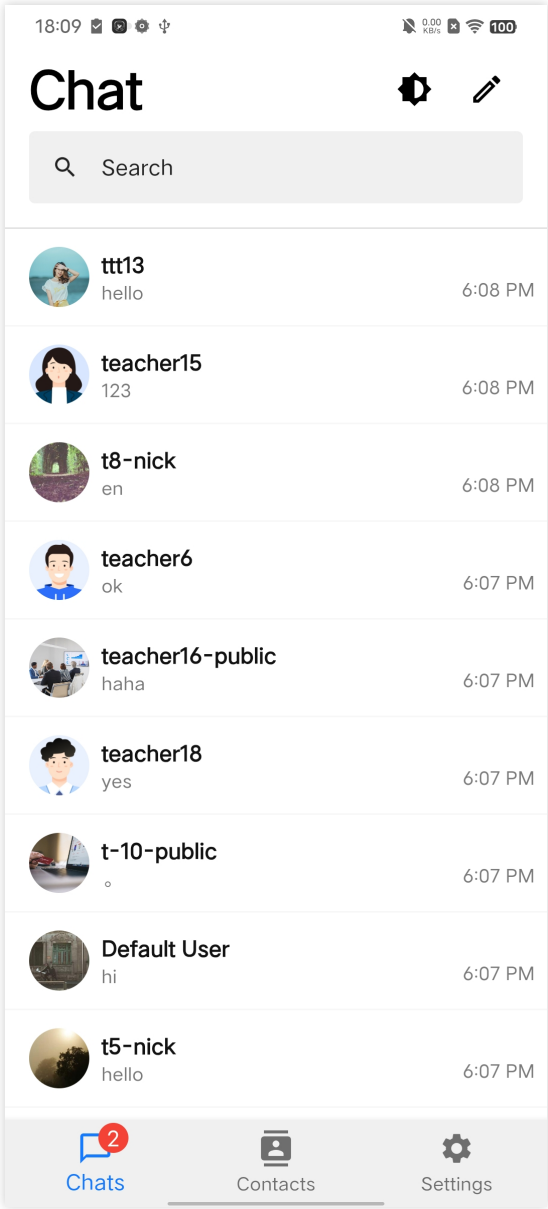
效果展示

会话列表

开启“显示用户在线状态”	关闭“显示用户在线状态”

©2013-2025 Tencent Cloud International Pte. Ltd.

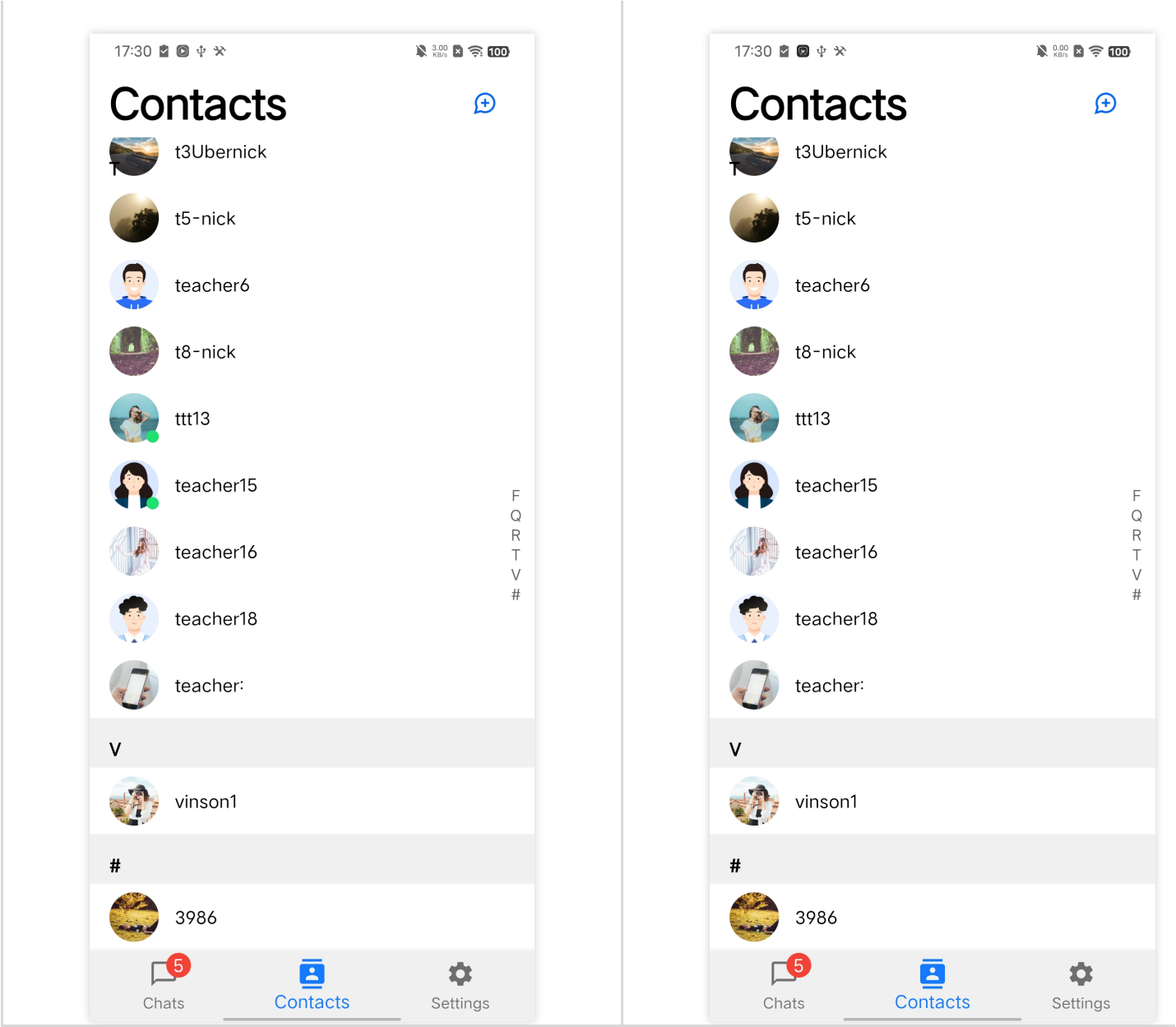




联系人列表

开启“显示用户在线状态”	关闭“显示用户在线状态”

©2013-2025 Tencent Cloud International Pte. Ltd.



功能说明

请在初始化 UIKit 时，通过配置全局 `TencentCloudChatUserConfig` 中在线状态功能字段 `useUserOnlineStatus`，来控制此功能开启或关闭。

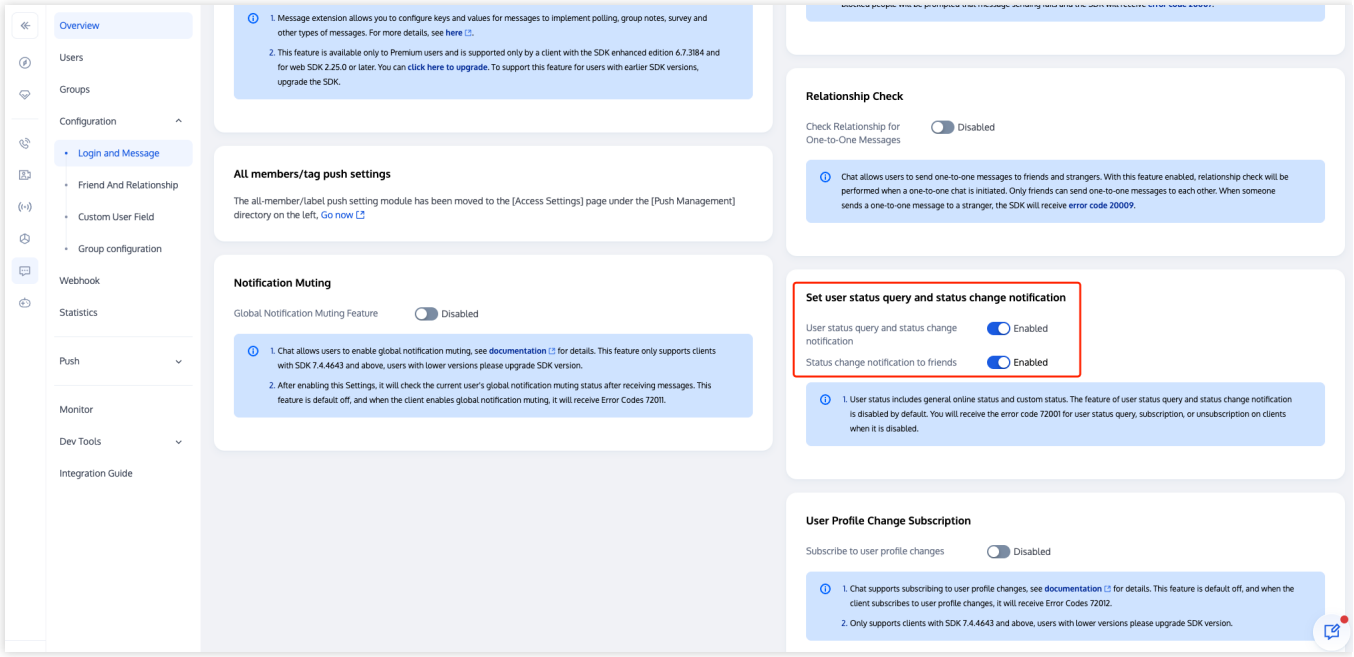
```
final bool initRes = await TencentCloudChat.controller.initUIKit(
  config: TencentCloudChatConfig(
    userConfig: TencentCloudChatUserConfig(
      useUserOnlineStatus: true,
      // ... 其他配置
    ),
  ),
);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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，请升级到专业版、专业版plus或企业版套餐。

本功能仅专业版、专业版plus或企业版套餐包支持，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 开通专业版、专业版plus或企业版套餐包进行体验。

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

©2013-2025 Tencent Cloud International Pte. Ltd.

对方正在输入

Android

最近更新时间：2025-03-03 14:36:24

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 Chat 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”

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 10:22:06

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 ChatSDK 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` 。
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。
Swift
Objective-C

```
TUIChatConfig.shared.enableTypingStatus = true
```

```
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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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）。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

消息已读回执
Android

最近更新时间：2025-03-21 21:45:35

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. 该功能仅专业版、专业版plus和企业版支持，请购买专业版、专业版plus或企业版后使用。
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。

©2013-2025 Tencent Cloud International Pte. Ltd.



群聊消息已读回执

消息上显示消息阅读情况：

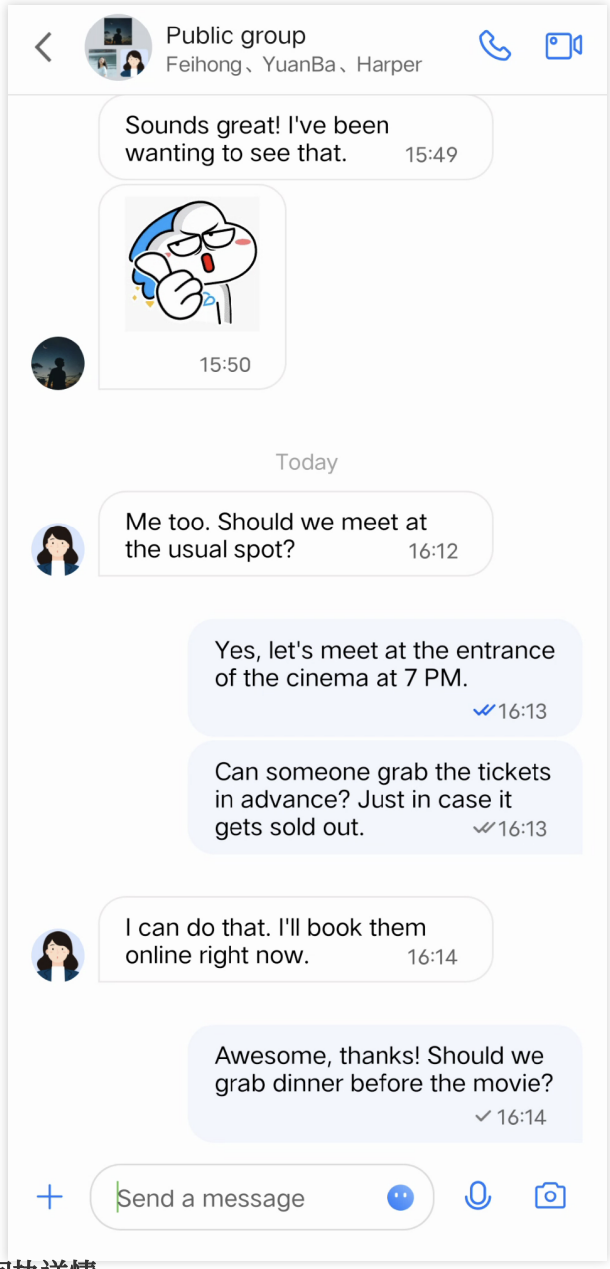
无人阅读时，显示 ✔ ；

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

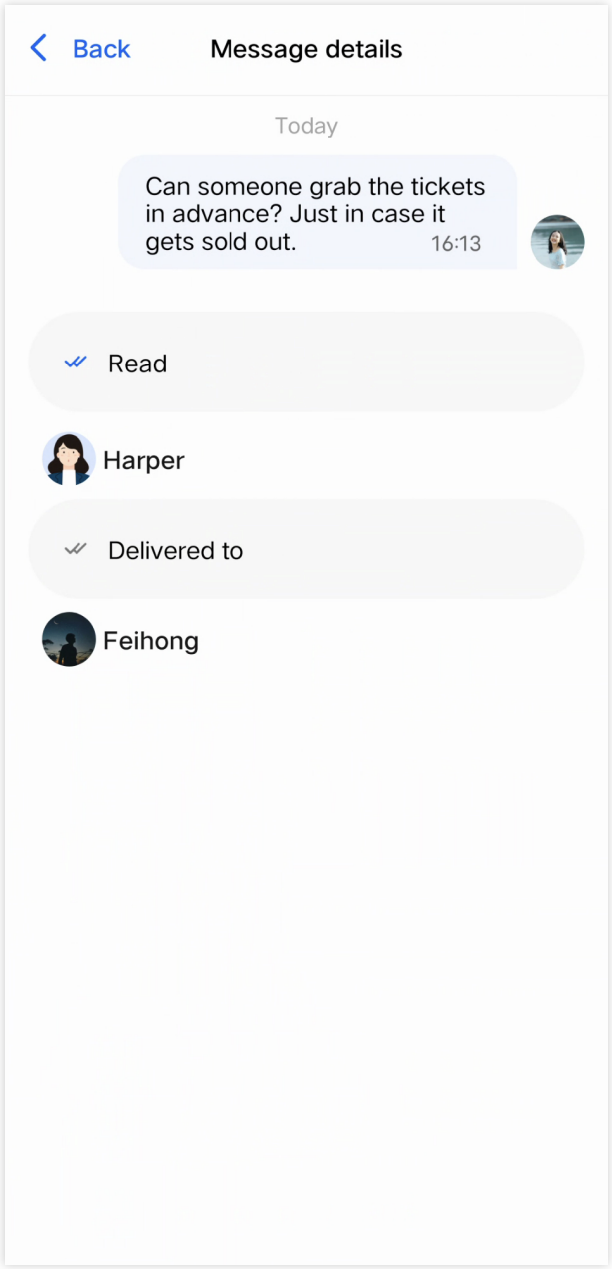
©2013-2025 Tencent Cloud International Pte. Ltd.



已读回执详情

点击已读状态即可进入已读回执详情页面。

©2013-2025 Tencent Cloud International Pte. Ltd.



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，首先请开通专业版、专业版plus和企业版套餐包，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);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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” 功能仅专业版、专业版plus和企业版支持，该报错信息表示您当前的套餐包不支持此能力，请 购买专业版 、专业版plus或企业版后使用。

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 10:22:06

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. 该功能仅专业版、专业版plus和企业版支持，请购买专业版、专业版plus或企业版后使用。
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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` 。
如果想开启消息已读回执功能，首先请开通专业版、专业版Plus或企业版套餐包，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。

Swift

Objective-C

```
TUIChatConfig.shared.msgNeedReadReceipt = true
```

```
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” 功能仅专业版、专业版plus和企业版支持，该报错信息表示您当前的套餐包不支持此能力，请购买专业版、专业版plus或企业版后使用。

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp（Vue）

最近更新时间：2025-05-27 18:02:12

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，表情回应功能仅专业版、专业版plus和企业版套餐支持，请 购买专业版、专业版plus或企业版后使用。

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
```

©2013-2025 Tencent Cloud International Pte. Ltd.

扩展资料

说明：
以下内容仅为辅助阅读资料，已读回执功能已在专业版、专业版Plus或企业版 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，请升级到专业版、专业版Plus或企业版套餐

“群消息已读回执”功能仅专业版、专业版Plus或企业版套餐支持，该报错信息表示您当前的套餐包不支持此能力，请开通专业版、专业版Plus或企业版进行体验。

2. 如何关闭已读功能？

请参考本文档 用户侧关闭已读回执 的内容，通过 `TUIUserService.switchMessageReadStatus(isDisplay: boolean)` 关闭已读功能。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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。

©2013-2025 Tencent Cloud International Pte. Ltd.

消息表情回应

Android

最近更新时间：2025-04-28 17:45:34

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：

表情回应功能仅专业版plus和企业版套餐支持，请 购买专业版 plus 或企业版 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

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-04-28 17:45:34

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：

表情回应功能仅专业版plus或企业版套餐支持，请 购买专业版 plus 或企业版 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

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp（Vue）

最近更新时间：2025-04-28 17:45:34

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，表情回应功能仅专业版plus或企业版套餐支持，请 购买专业版 plus 或企业版 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-04-28 17:45:34

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。
注意：
表情回应功能仅专业版plus和企业版套餐支持，请 购买专业版 plus 或企业版 后使用。

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

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

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

消息引用
Android & iOS

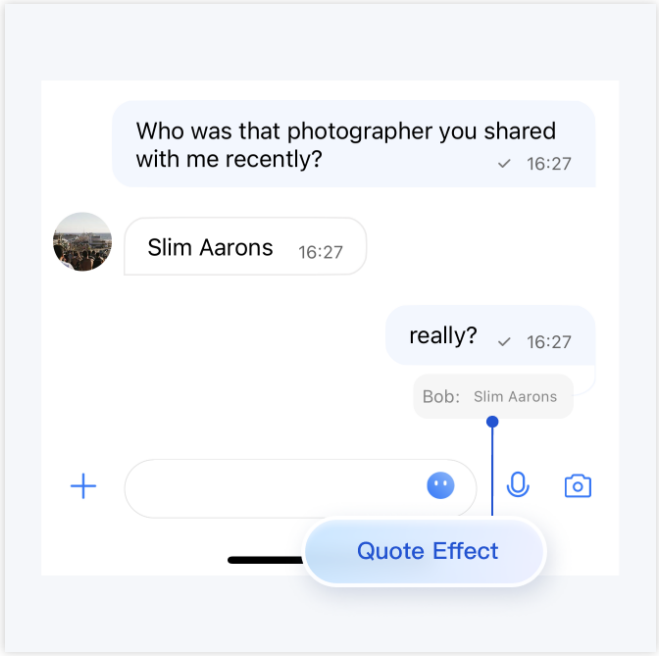
最近更新时间：2024-06-14 10:29:49

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：

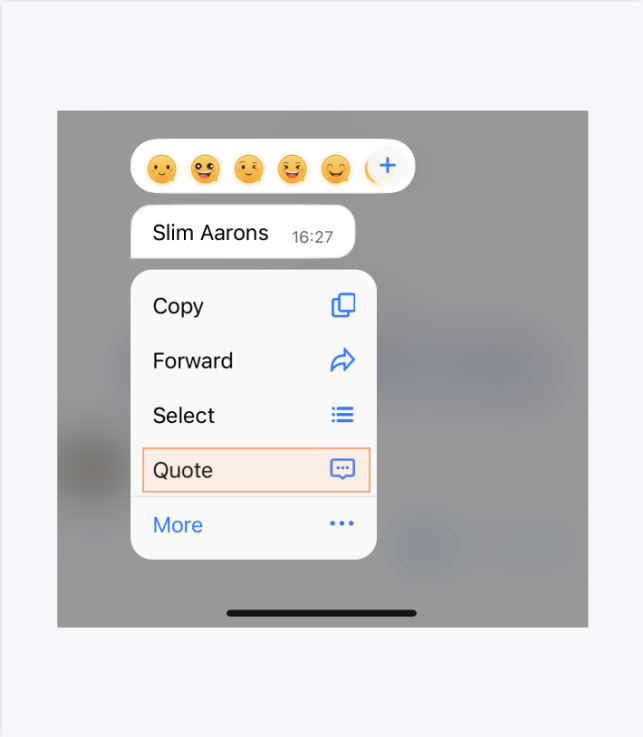


功能说明

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。

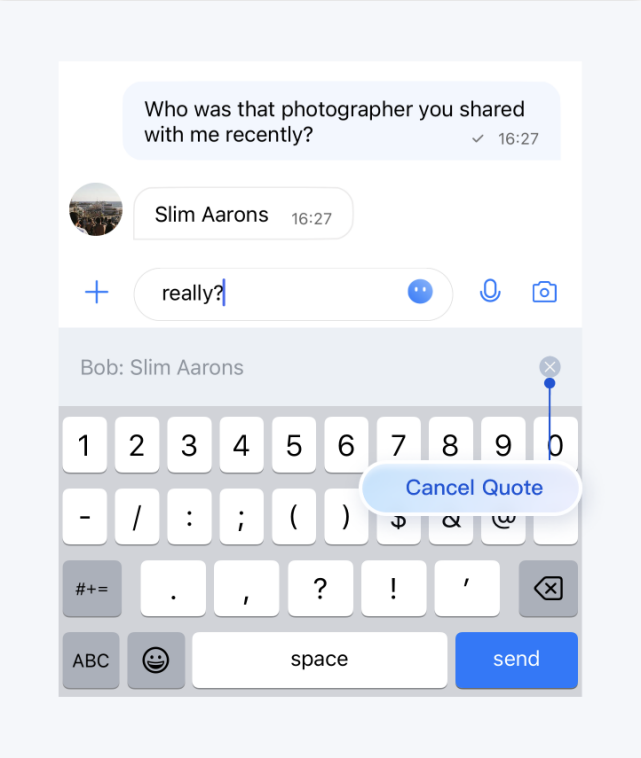
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

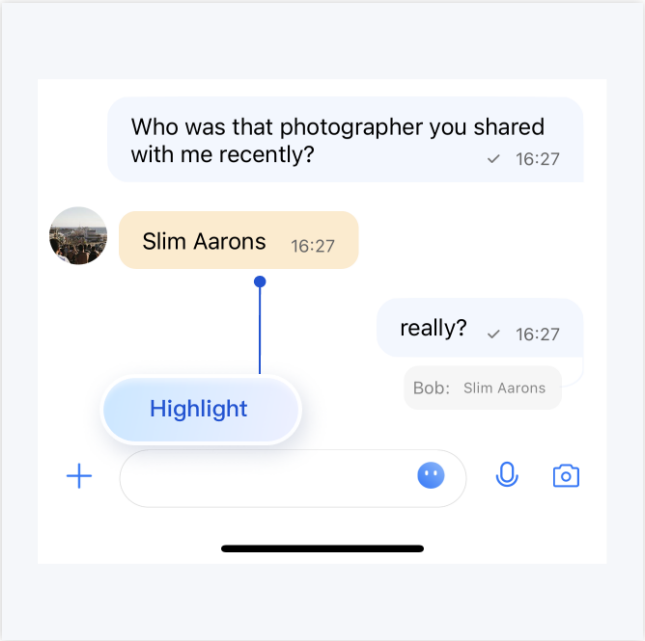
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



消息引用

引用一条消息

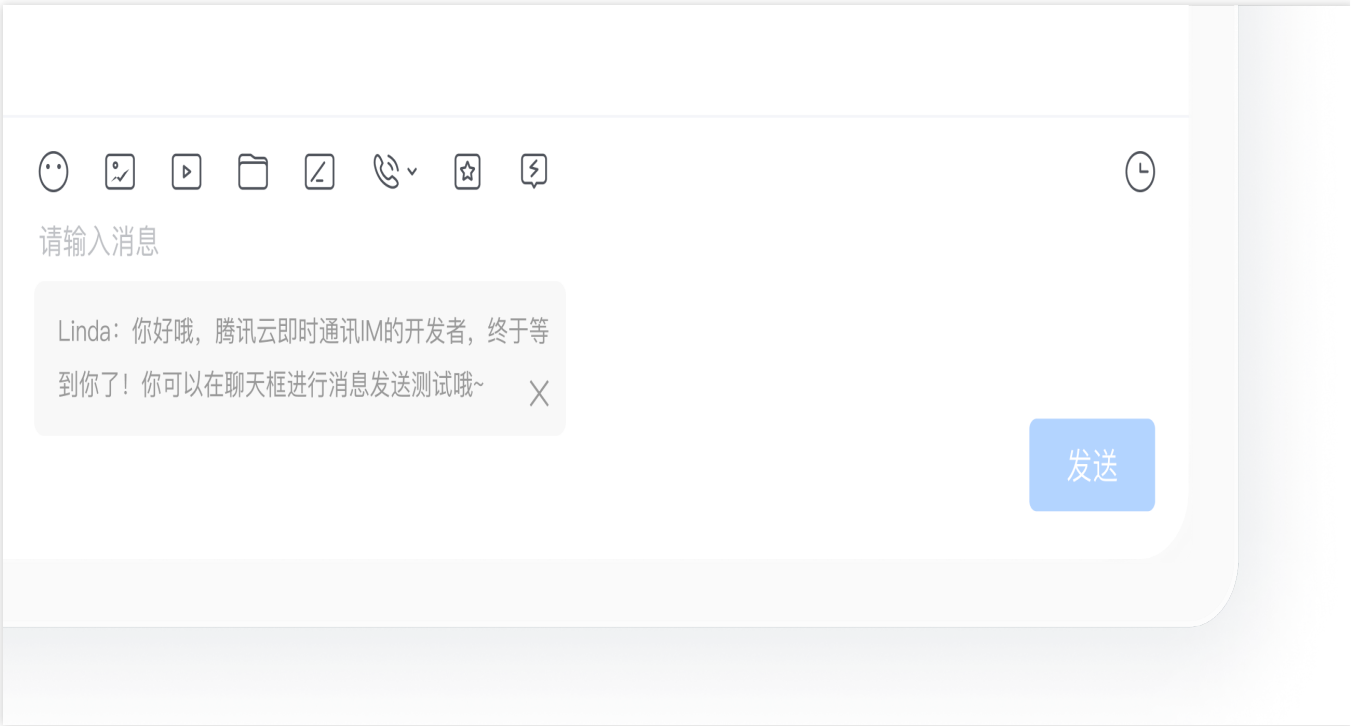
Web 端右键消息，移动端长按消息，在消息上会弹出消息工具栏，点击工具栏中的引用按钮，即可对该消息进行引用。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

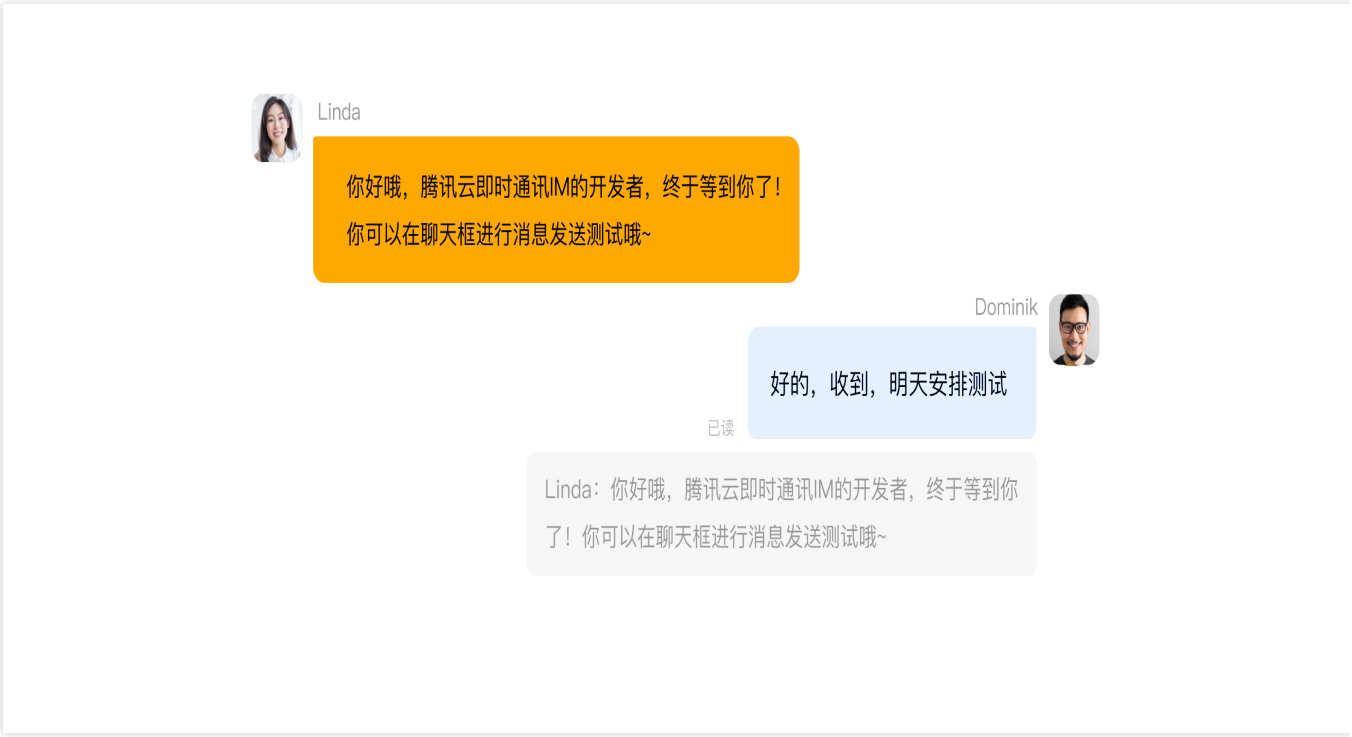


查看被引用消息

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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,
  // }
]
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

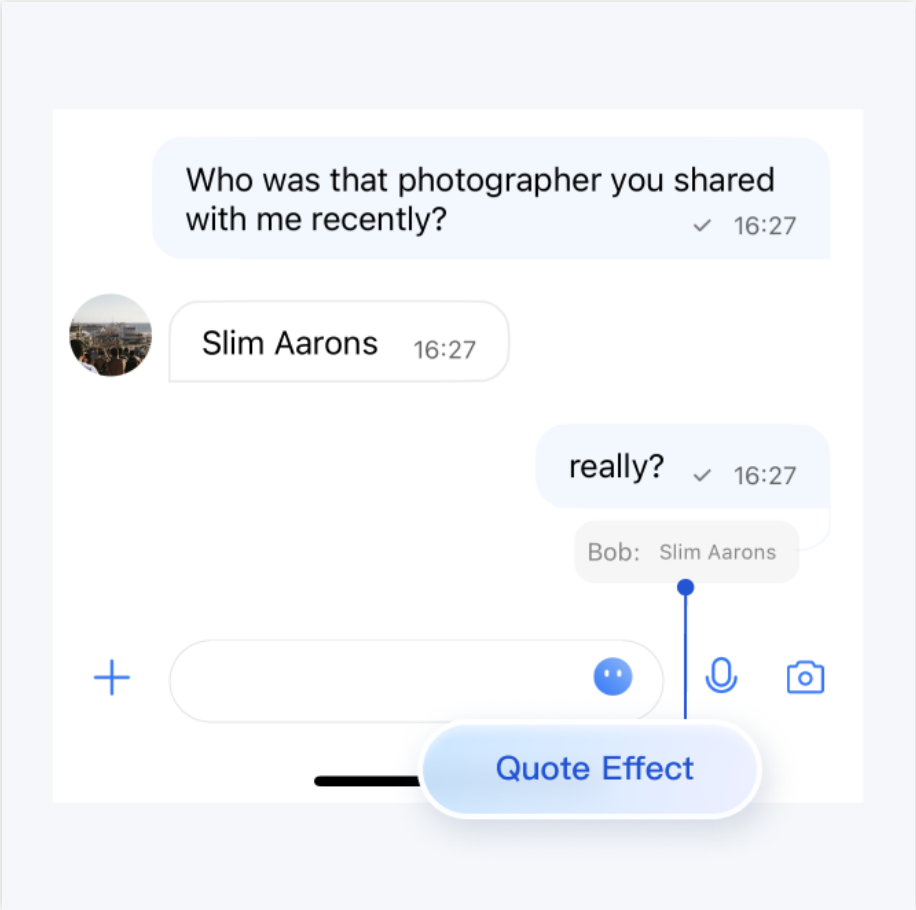
©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

最近更新时间：2024-12-13 17:38:14

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。

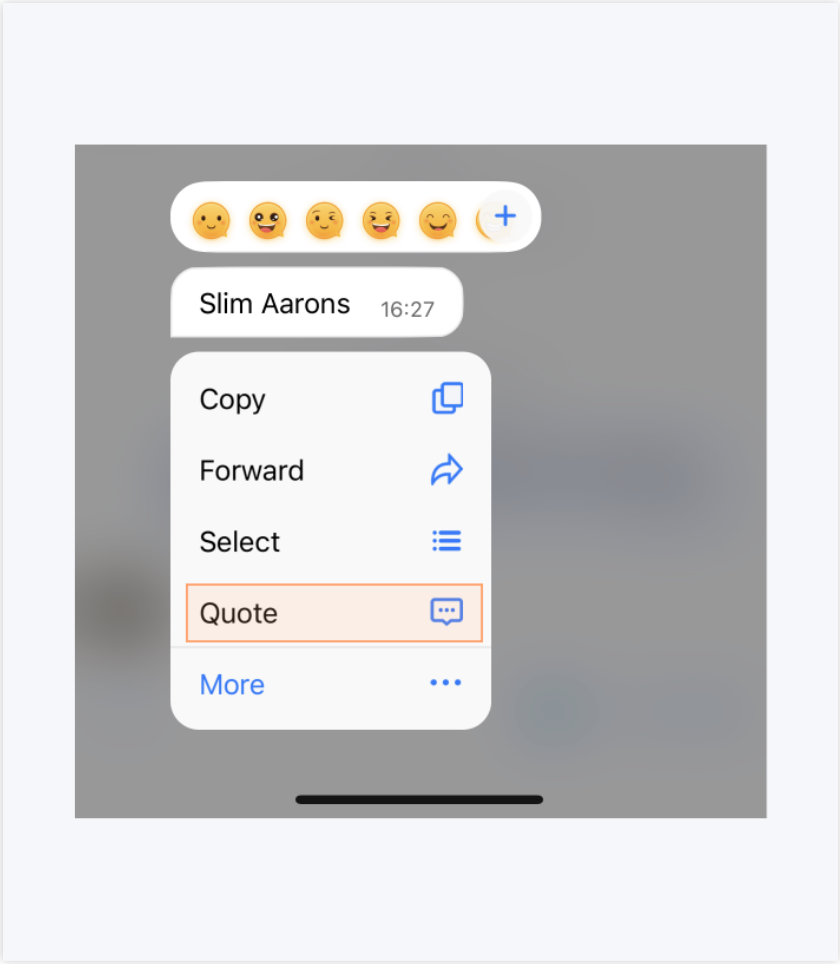


消息引用

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。

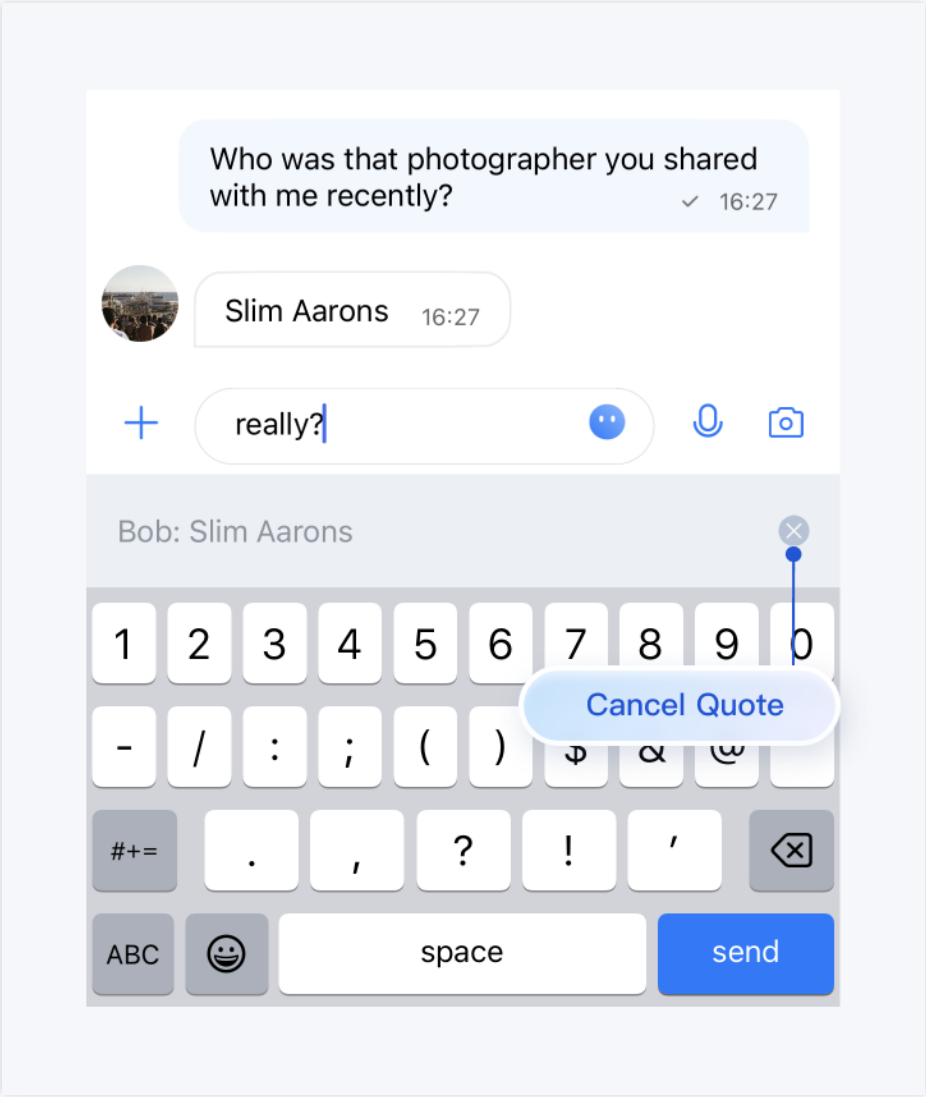
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

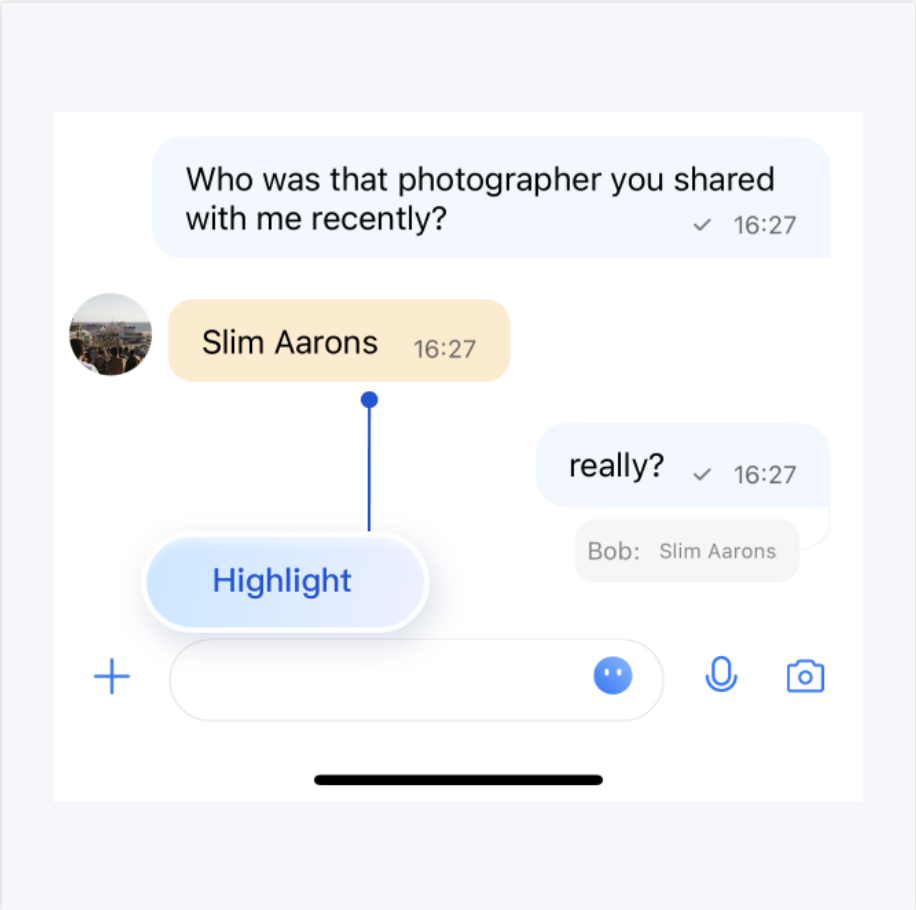
©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

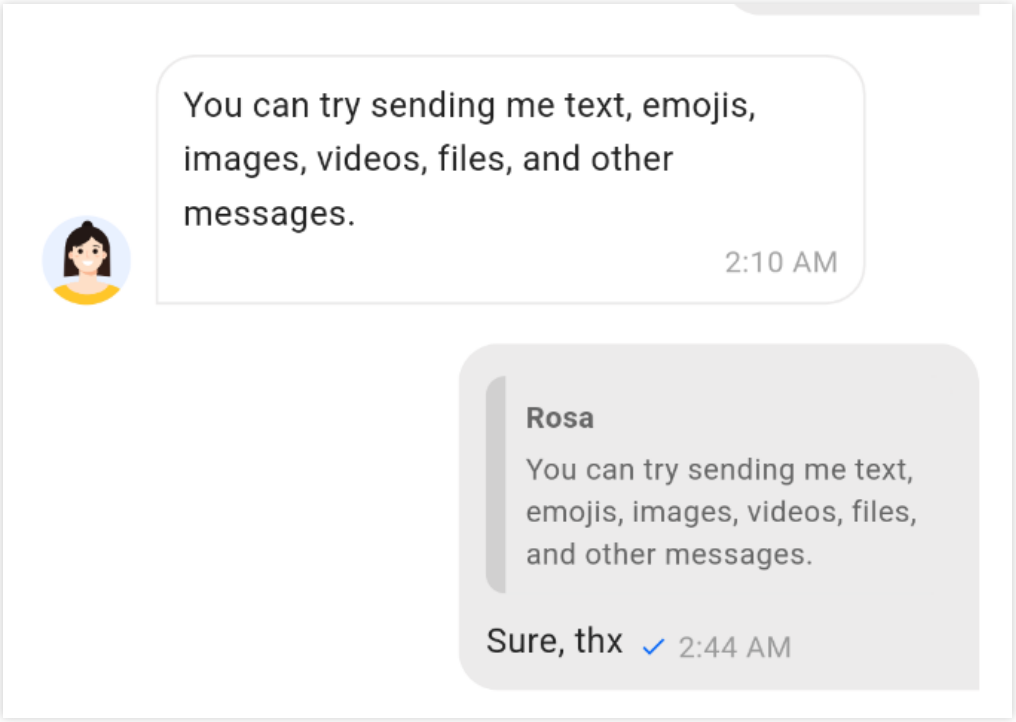
最近更新时间：2024-07-08 16:09:44

描述

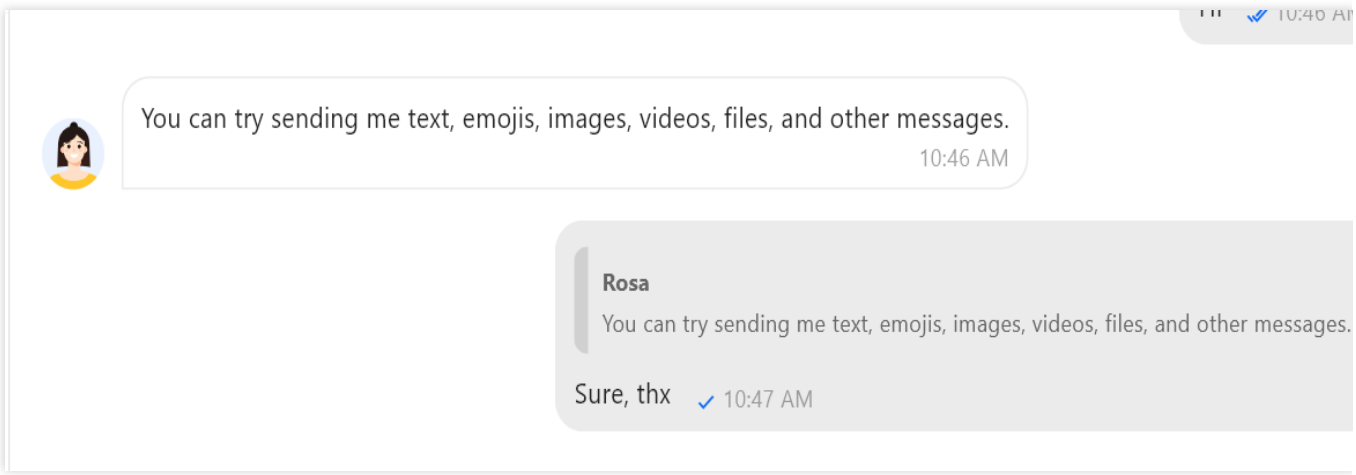
在 TencentCloudChatMessage 的消息列表中，您可以通过引用特定的先前消息进行回复。回复后，点击被引用的消息将跳转到原始消息，并将其高亮显示。
引用消息有两种模式：“消息引用”和“消息回复”。
消息引用：仅引用消息。消息上下文菜单中显示的文本为“引用”。
消息回复：引用并回复消息，在群聊中提及消息发送者。消息上下文菜单中显示的文本为“回复”。

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



©2013-2025 Tencent Cloud International Pte. Ltd.



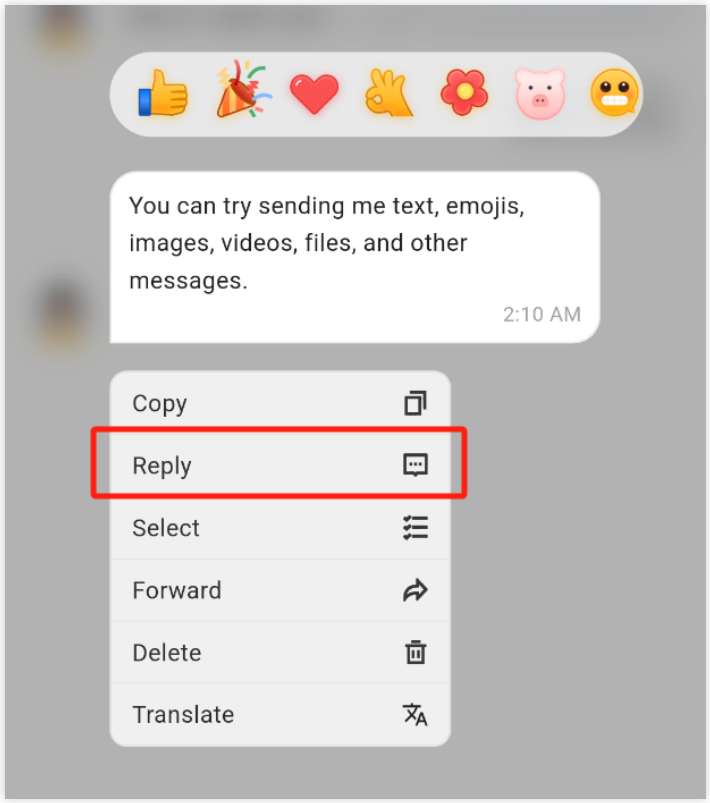
功能说明

引用一条消息

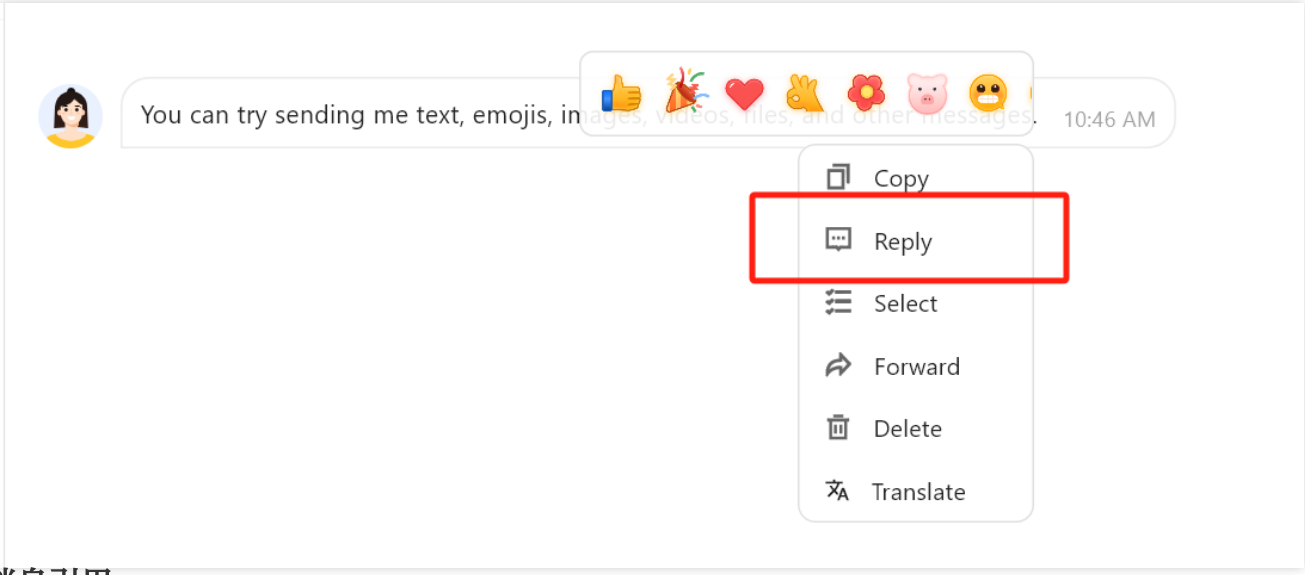
长按消息，消息上会弹出消息工具栏。单击工具栏中的引用按钮，对该消息进行引用。

移动端

桌面端



©2013-2025 Tencent Cloud International Pte. Ltd.

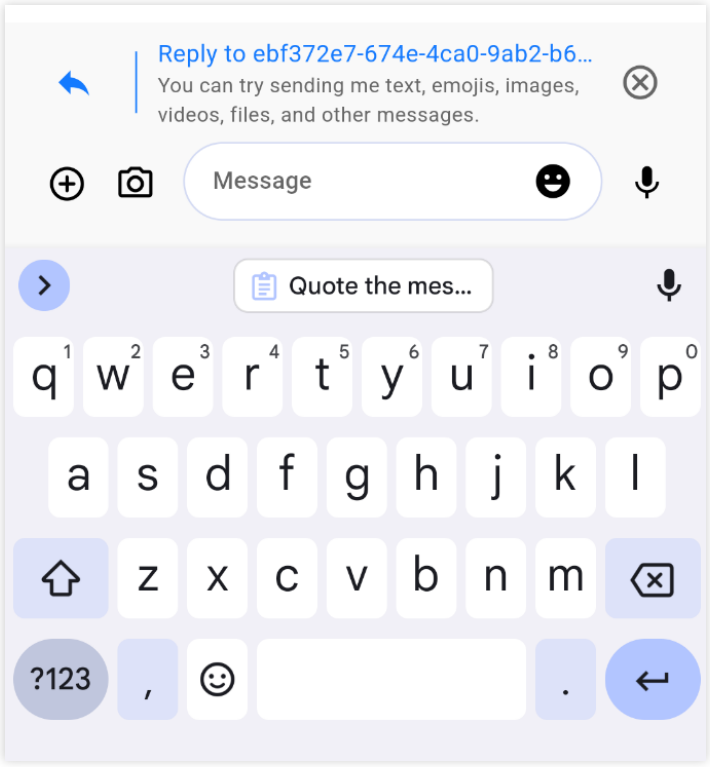


取消消息引用

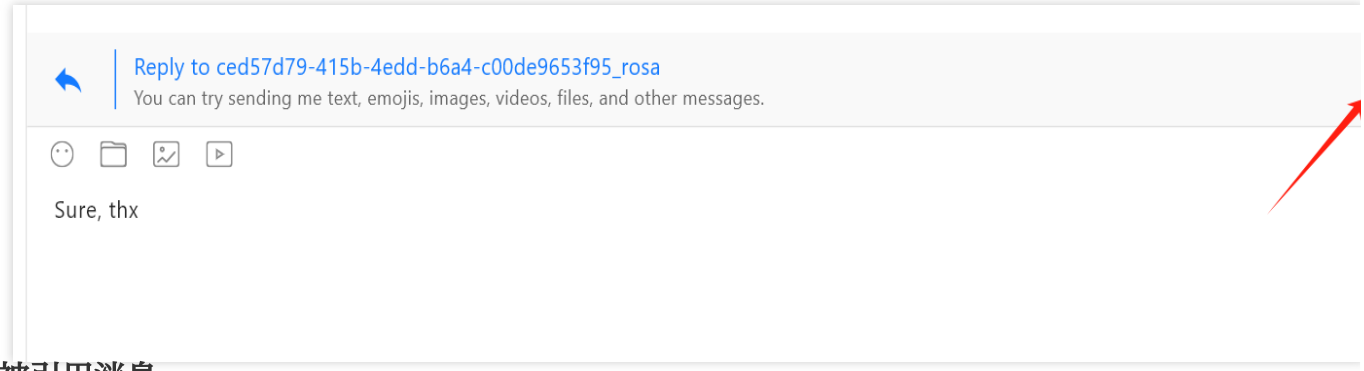
在消息被引用但还未发出时，通过单击引用之后的关闭按钮，可以取消消息引用。

移动端

桌面端

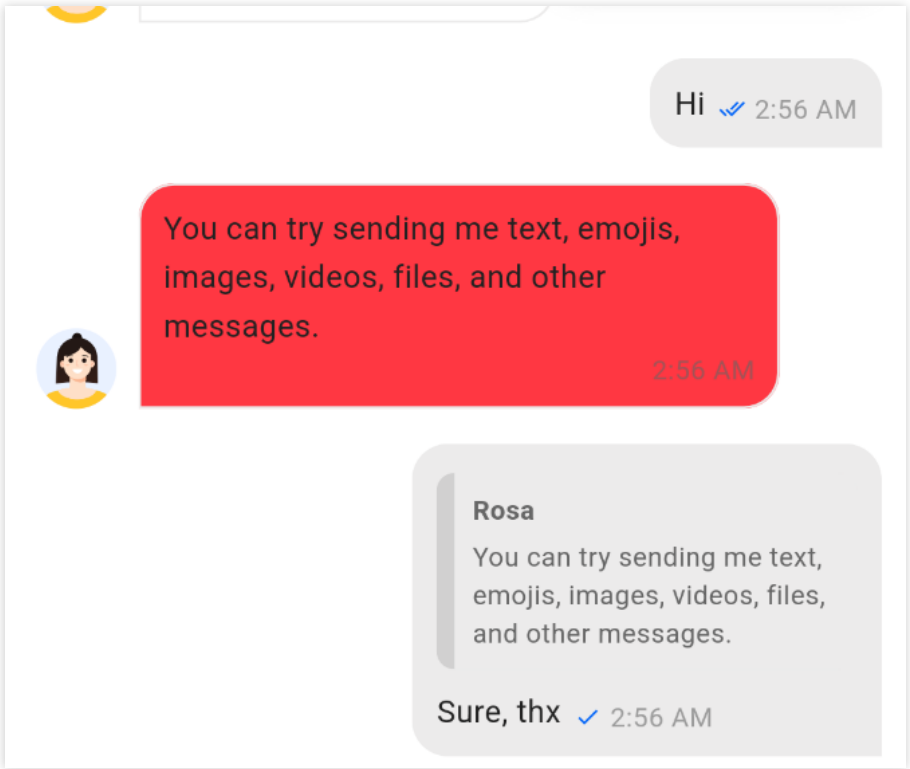


©2013-2025 Tencent Cloud International Pte. Ltd.

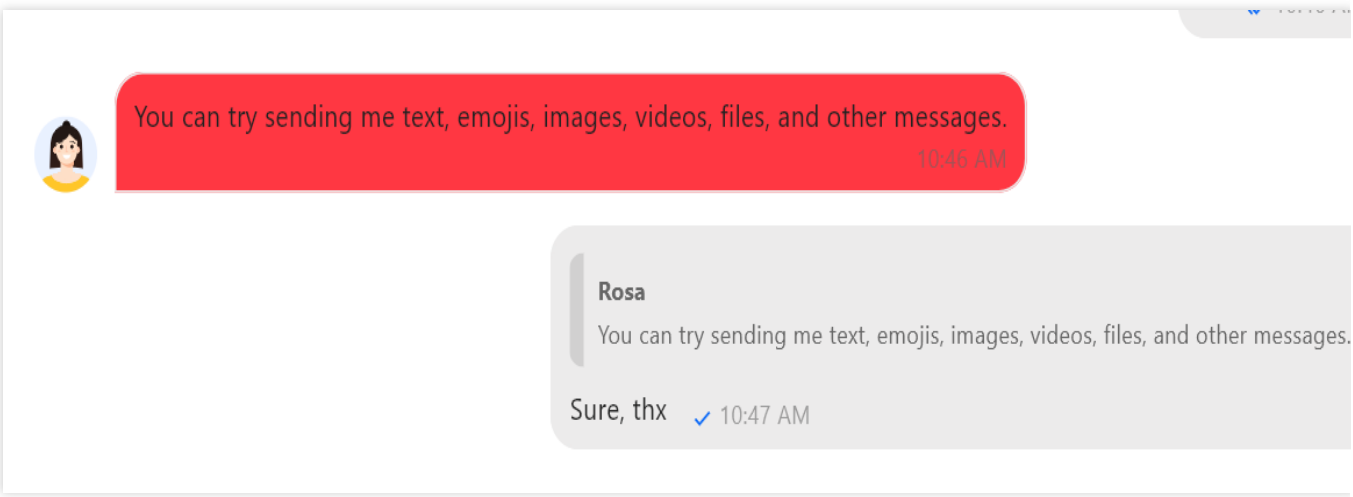


查看被引用消息

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



©2013-2025 Tencent Cloud International Pte. Ltd.



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

文本消息翻译

Flutter

最近更新时间：2025-03-27 15:32:17

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。

文本翻译功能仅对**专业版 Plus 和企业版**客户开放，购买专业版 Plus 和企业版 后可使用；体验版支持一定额度免费试用，有效期一个月。

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	文本消息翻译效果

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	文本消息翻译效果

功能概览

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

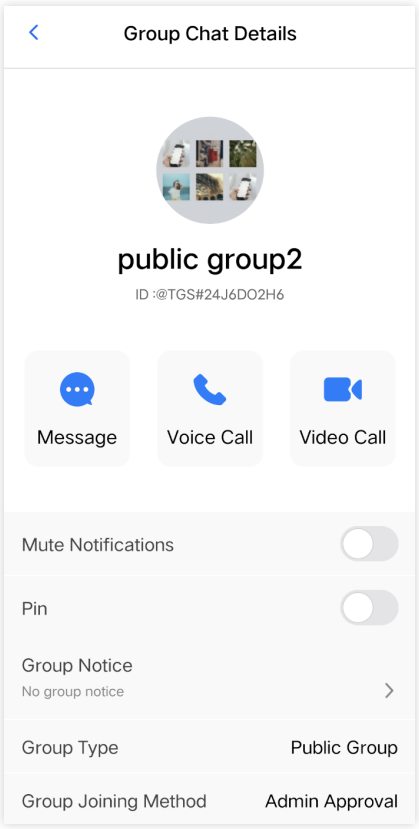
国际化界面语言

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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`。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

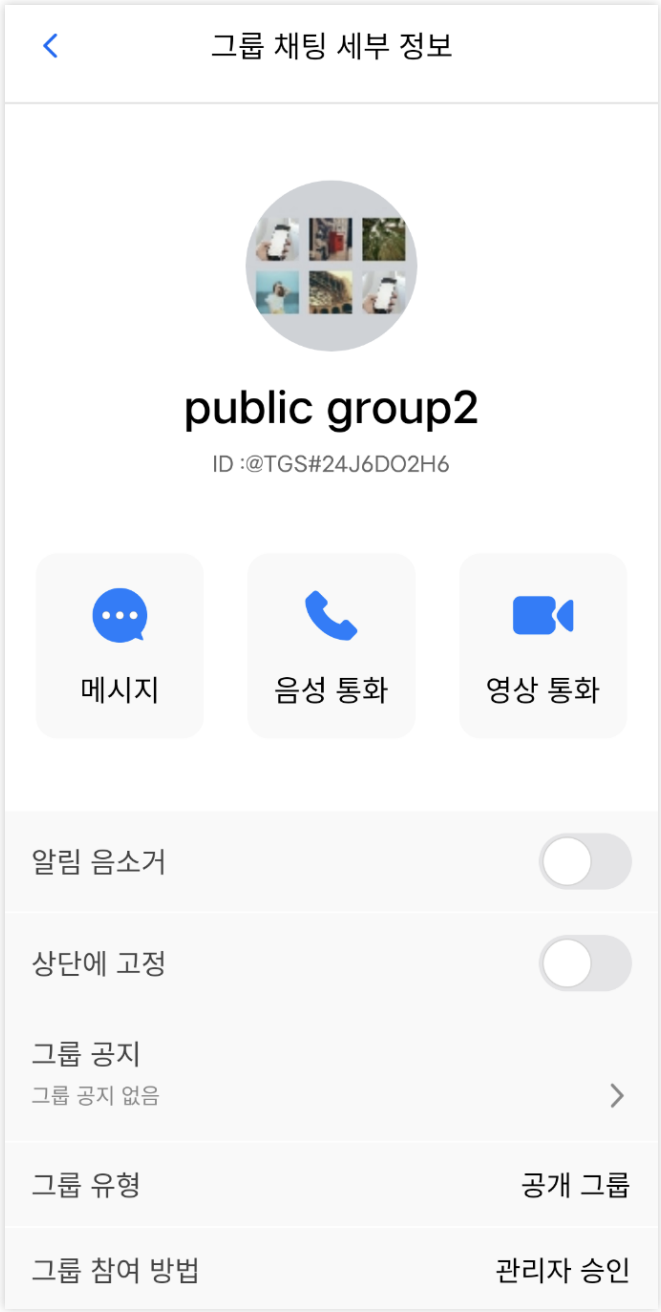
指定显示的语言

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



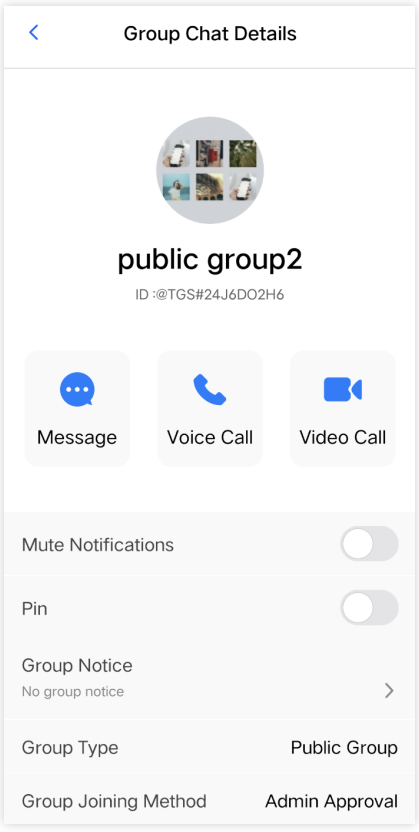
©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

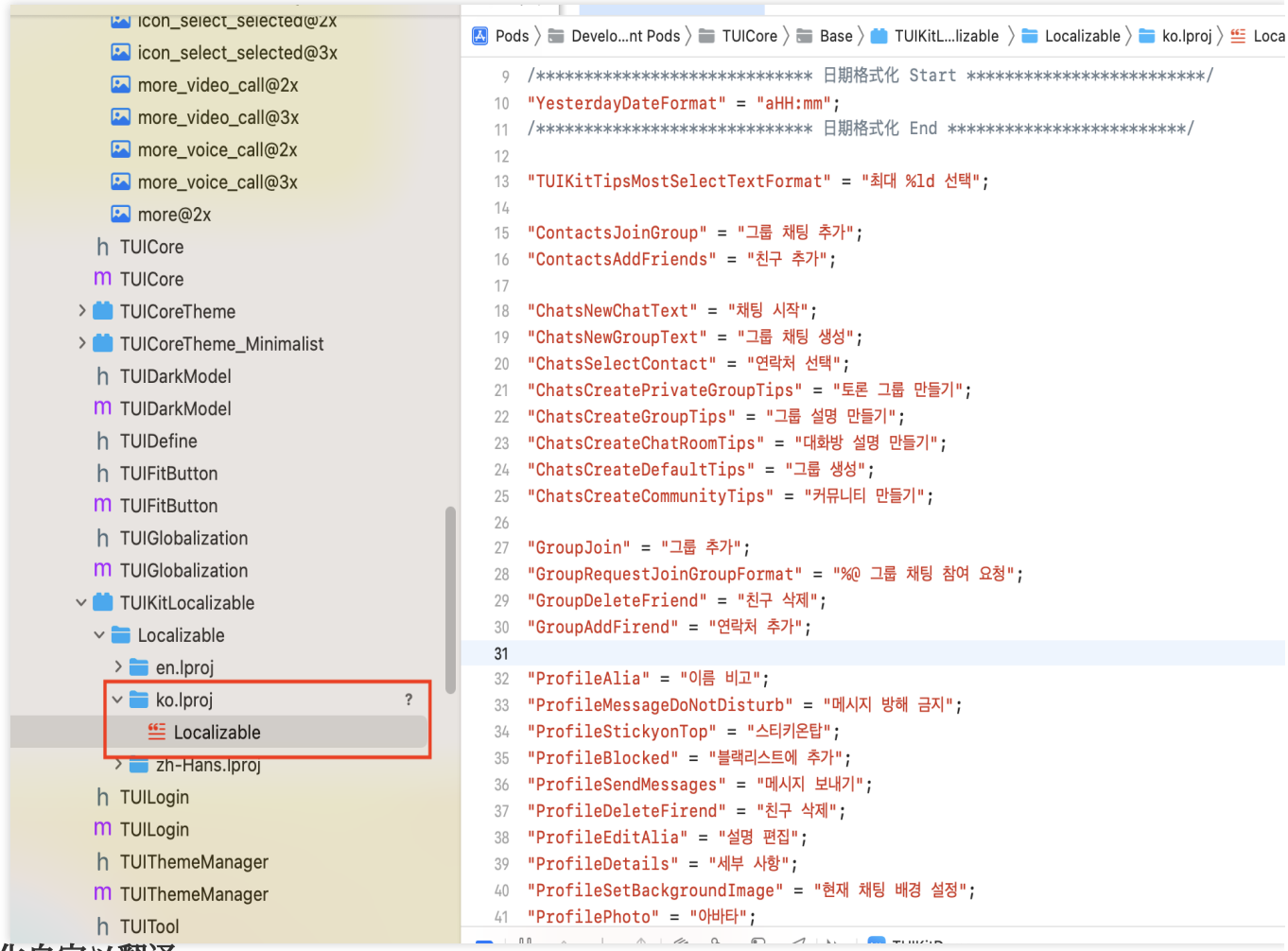
使用更多语言/自定义翻译表述

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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

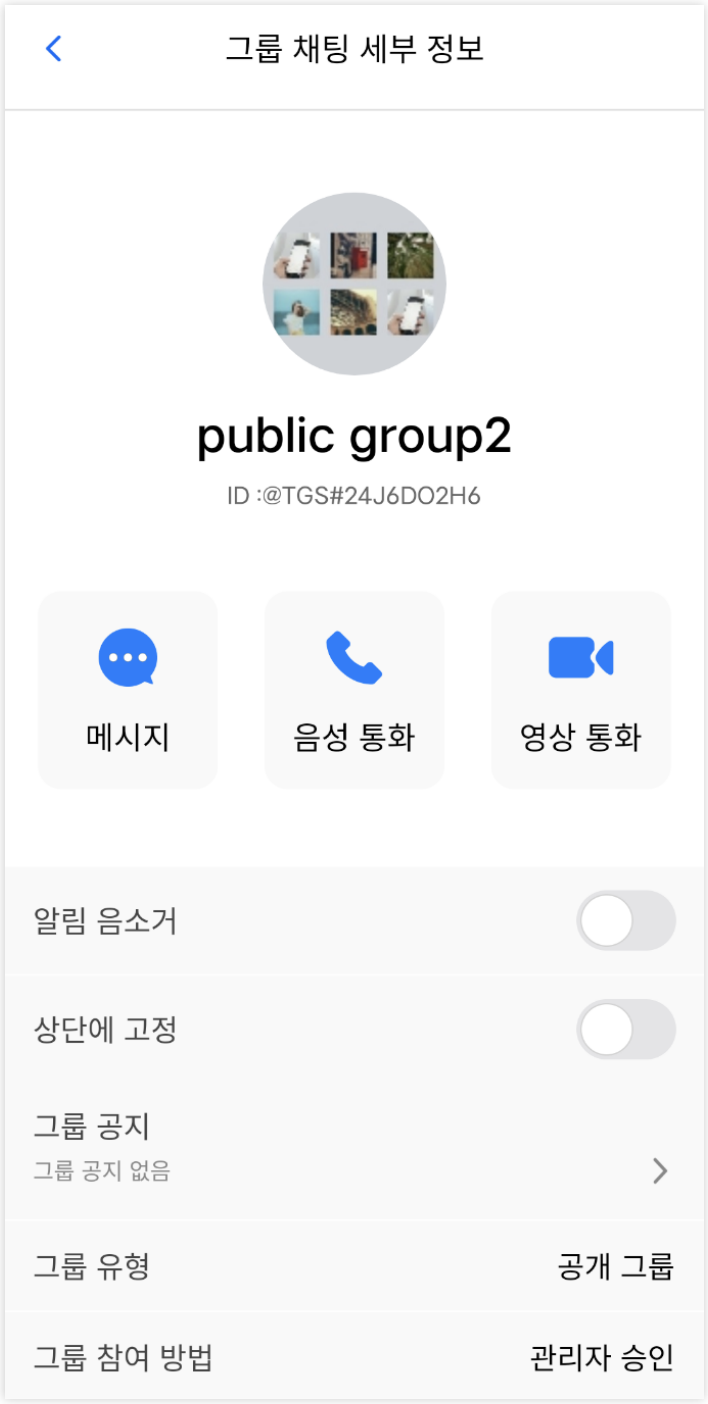
语言代码清单见 ISO 639-1 语言代码。

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 (Vue)

最近更新时间：2024-06-12 17:44:13

功能描述

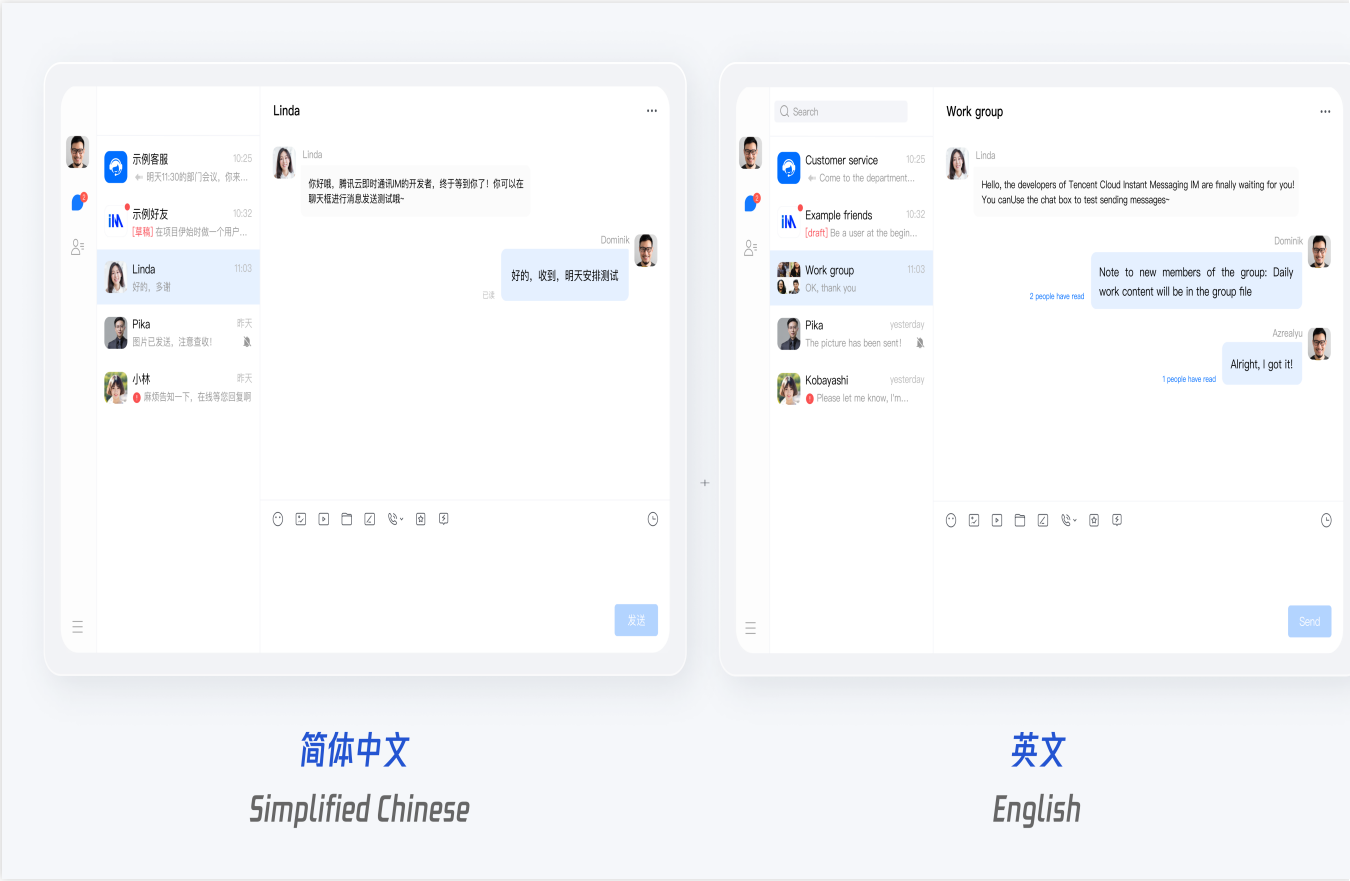
说明：

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！

©2013-2025 Tencent Cloud International Pte. Ltd.

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 {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        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

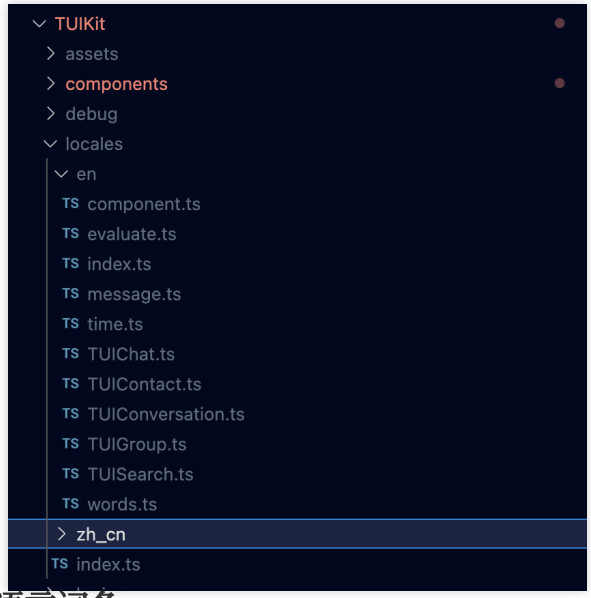
如果您需要对现有的 **简体中文、英语** 语言包词条进行扩充或修改，可以在 `src/TUIKit/locales` 目录下进行新增或修改词条。

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>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
</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。

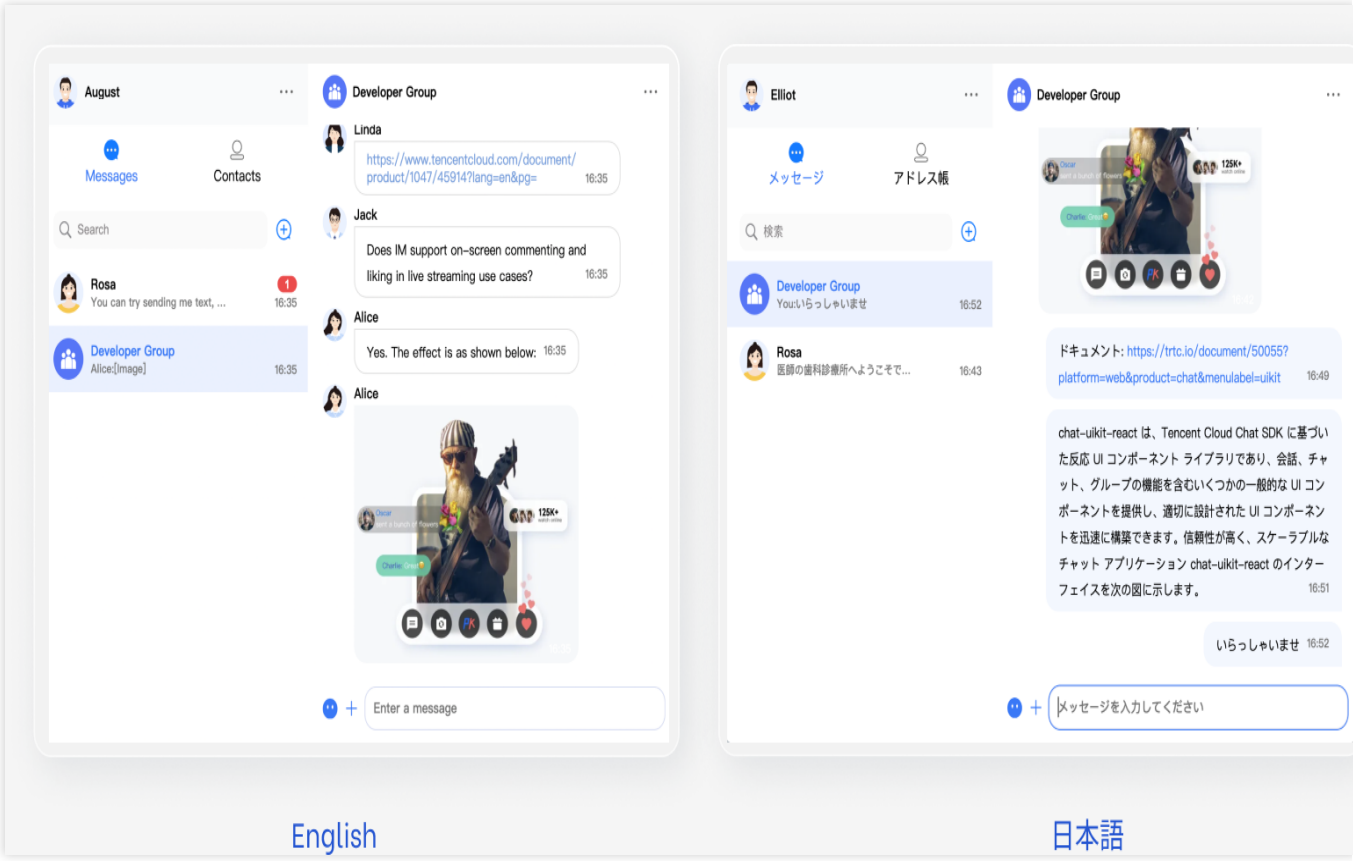
©2013-2025 Tencent Cloud International Pte. Ltd.

React

最近更新时间：2024-11-22 11:55:28

功能描述

React UIKit 默认自带 **英语、日语、韩语、中文简体/ 中文繁体** 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 / 中文简体/ 中文繁体** ，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

©2013-2025 Tencent Cloud International Pte. Ltd.

如果您需要指定界面的语言，需要在引入 `UIKitProvider` 时设置 `language` 。

```
  // language support en-US / zh-CN / ja-JP / ko-KR / zh-TW
  <UIKitProvider language={'en-US'}>...</UIKitProvider>
```

动态切换语言

在 `UIKitProvider` 之外，动态切换 React UIKit 语言：

```
import React, { useState } from 'react';
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';

// language support en-US / zh-CN / ja-JP / ko-KR / zh-TW
const languageList = ['en-US','zh-CN','ja-JP','ko-KR', 'zh-TW'];

export default function App() {
  // language setting
  const [currentLanguage, setCurrentLanguage] = useState('en-US');
  const changeLanguage = (language) => {
    setCurrentLanguage(language);
  };

  return (
    // select language
    // <div @click="changeLanguage('en-US')">English</div>
    <UIKitProvider language={currentLanguage}>
      ...
    </UIKitProvider>
  );
}
```

在 `UIKitProvider` 之内，动态切换 React UIKit 语言：

```
import React from 'react';
import { UIKitProvider, useUIKit } from '@tencentcloud/uikit-base-component-react';

// language support en-US / zh-CN / ja-JP / ko-KR / zh-TW
const languageList = ['en-US','zh-CN','ja-JP','ko-KR', 'zh-TW'];

export default function App() {
  return (
    // init language
    <UIKitProvider language='en-US'>
      <Child />
    </UIKitProvider>
  );
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
function Child() {
    const { language, setLanguage } = useUIKit();
    setLanguage('zh-CN');
    return <div>current language is {language}</div>
}
```

使用语言词条

使用 hook `useUIKit` 导出翻译函数 `t`， `useUIKit` 作为 Hook 只能在 `UIKitProvider` 的子组件中使用。

```
import React from 'react';
import { UIKitProvider, useUIKit } from '@tencentcloud/uikit-base-component-react';

export default function SampleChat() {
  return (
    // init language
    <UIKitProvider language='en-US'>
      <Child />
    </UIKitProvider>
  );
}

function Child() {
    const { t } = useUIKit();
    return <div>{t('TUIChat.No More')}</div>
}
```

自定义语言词条

新增或修改语言词条

如果您需要对现有的 **英语 / 日语 / 韩语 / 中文简体/ 中文繁体** 语言包词条进行扩充或修改，参考下面的示例代码。具体词条详情请参考 Github - @tencentcloud/chat-uikit-react/locales。

```
import { UIKitProvider } from '@tencentcloud/uikit-base-component-react';

const additionalLanguageResource = [
  {
    lng: 'de',
    translation: {
      'TUIChat': {
        'No More': 'Nicht mehr',
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      },
    },
  },
];

export default function App() {
  return (
    <UIKitProvider language="de" additionalLanguageResources={additionalLanguageRes
      <Child />
    </UIKitProvider>
  );
}

function Child() {
    const { t, language, setLanguage } = useUIKit();
    return <div>current language is {language}. {t('TUIChat.No More')}</div>
}
```

交流与反馈

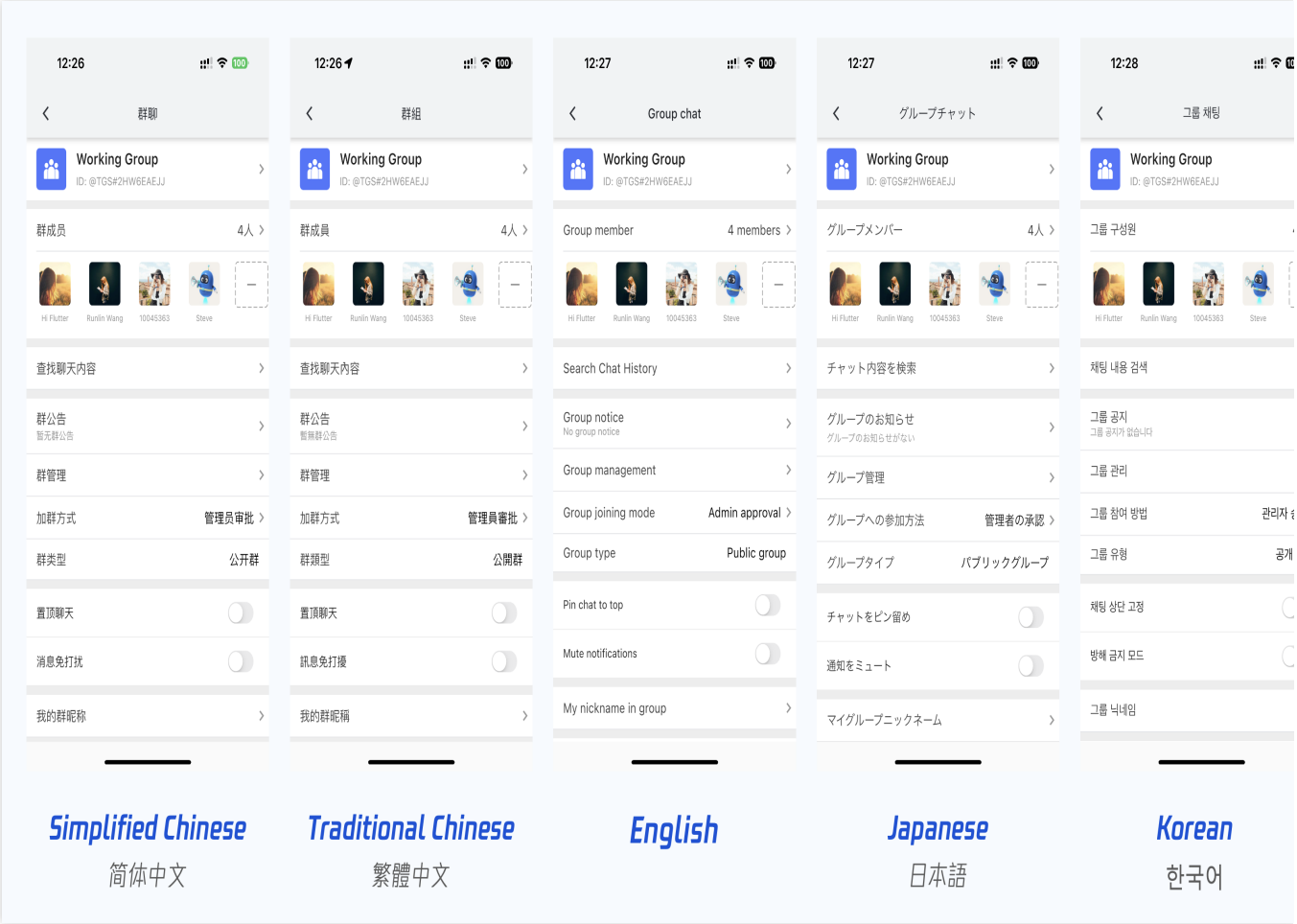
加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

最近更新时间：2025-03-03 14:39:15

腾讯云 Chat 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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

最近更新时间：2024-12-19 14:28:51

功能描述

React Native UIKit 默认自带 **英语**、**简体中文** 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，也可以根据您的需要实现定制化的国际化能力，包括新增语言，更新词条翻译内容。



使用默认语言包

©2013-2025 Tencent Cloud International Pte. Ltd.

初始化语言包

App 初始化时您需要在 `App.tsx` 文件中给翻译引擎注册语言包并进行初始化。

```
import { TUITranslateService } from '@tencentcloud/chat-uikit-engine';
import uikitResources from '@tencentcloud/chat-uikit-react-native/i18n';

 // Init localization
 TUITranslateService.provideLanguages(uikitResources);
 TUITranslateService.useI18n('en-US');
```

切换语言

如果您的 App 中有切换语言入口，在您切换语言时可以调用 `TUITranslateService.changeLanguage` 实现语言切换功能。

```
import { TUITranslateService } from '@tencentcloud/chat-uikit-engine';

// language 是您切换后的目标语言
TUITranslateService.changeLanguage(language)
```

添加 App 语言资源

如果您的 App 中除了 UIKit，还需要对其他页面（比如：Login 页、Setting 页）进行翻译，您可以在 App 根目录中创建 i18n 资源目录添加相应的词条，通过 `TUITranslateService` 来翻译，可以按照以下示例代码进行初始化，示例代码默认在您项目的根目录下有 `i18n` 目录。

```
import { TUITranslateService } from '@tencentcloud/chat-uikit-engine';
import uikitResources from '@tencentcloud/chat-uikit-react-native/i18n';
import appResources from './i18n';

// Init localization
TUITranslateService.provideLanguages({
    'en-US': {
      ...appResources['en-US'],
      ...uikitResources['en-US'],
    },
    'zh-CN': {
      ...appResources['zh-CN'],
      ...uikitResources['zh-CN'],
    },
});

TUITranslateService.useI18n('en-US');
```

©2013-2025 Tencent Cloud International Pte. Ltd.

更新词条翻译

如果默认的词条翻译不满足您的需求，您可以更新词条翻译，可以按照如下步骤进行更新。
从 `node_modules/@tencentcloud/chat-uikit-react-native/` 中复制 `i18n` 放到您项目的根目录下，并重命名为 `i18n-uikit` ，根据需要对翻译内容进行修改。修改完成后， `uikitResources` 修改为从您本地 `import` 。

```
import uikitResources from './i18n-uikit';
```

新增语言类型

如果您需要新增一种语言，可以从 `node_modules/@tencentcloud/chat-uikit-react-native/i18n` 中复制 `en-US` 放到您项目的根目录下的 `i18n` 中，并命名为新语言的名字（比如：'zh-TW'），并对翻译内容进行修改即可。

使用翻译接口

如果您的 App 需要使用翻译引擎提供的翻译能力，可以按照如下方式使用。
比如您新增了一个 Login 模块的翻译词条，需要对 【用户名】实现国际化，Login 模块词条定义如下：
`en-US` ：

```
export const Login = {
  USER_NAME: 'UserName',
};
```

`zh-CN` ：

```
export const Login = {
  USER_NAME: '用户名',
};
```

在 Login 页面中使用翻译函数：

```
import { TUITranslateService } from '@tencentcloud/chat-uikit-engine';
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
TUITransalteService.t('Login.USER_NAME')
```

关于切换语言如何实现，您可以参见 UIKit Demo 源码 。

©2013-2025 Tencent Cloud International Pte. Ltd.

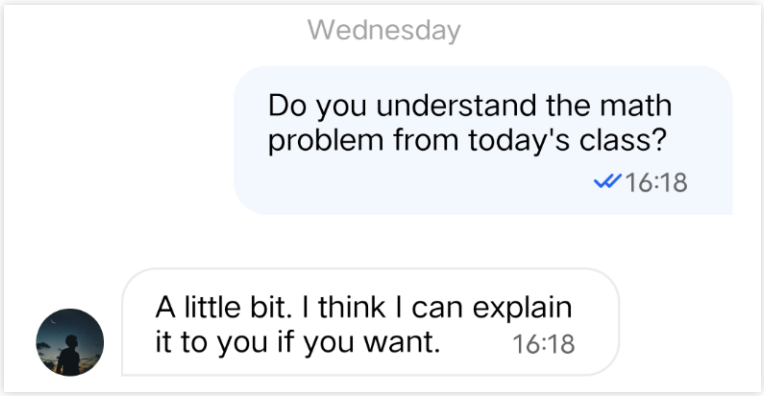
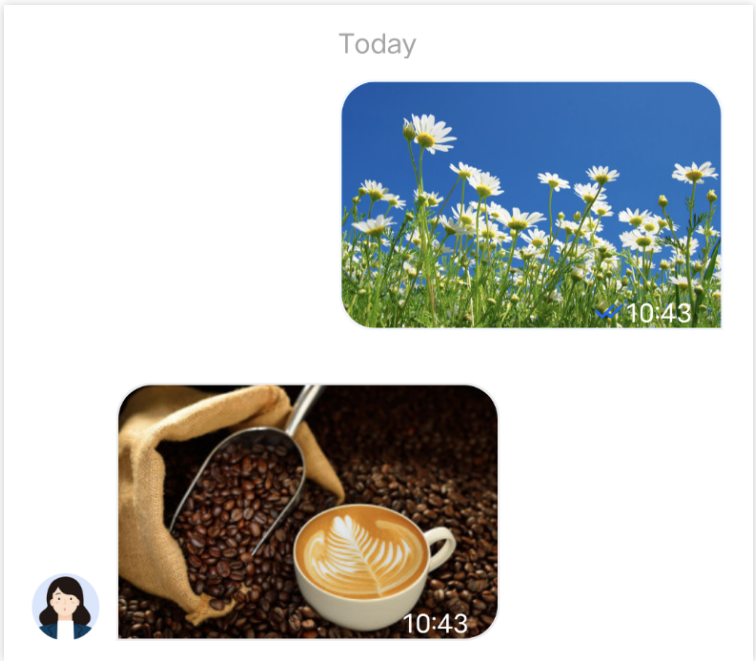
添加自定义消息

Android

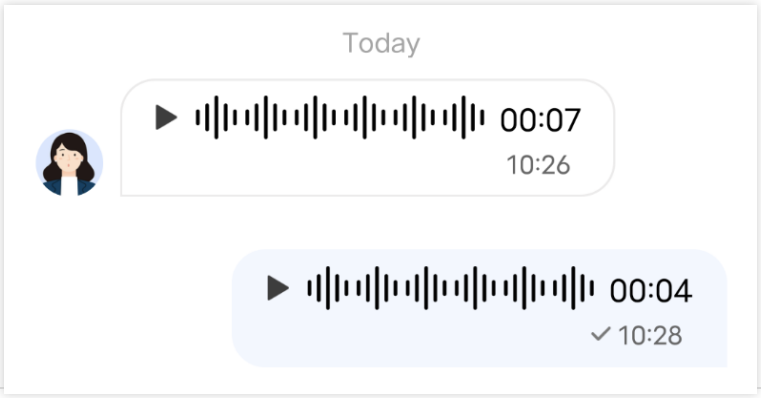
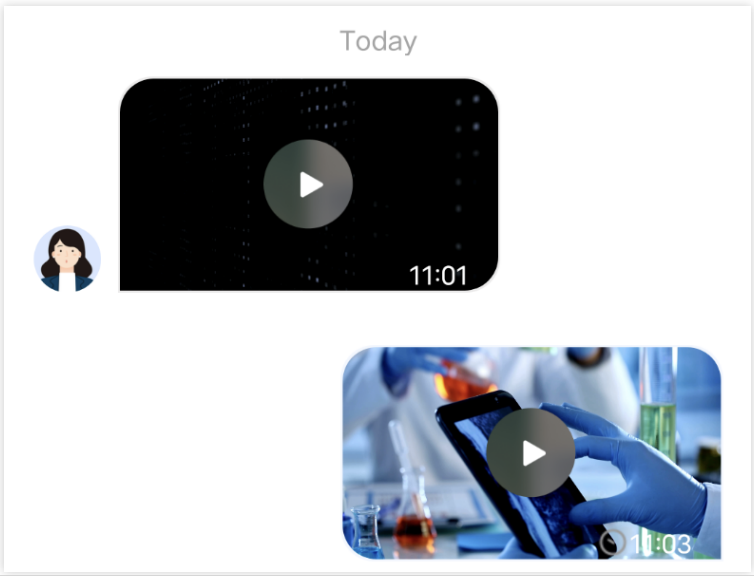
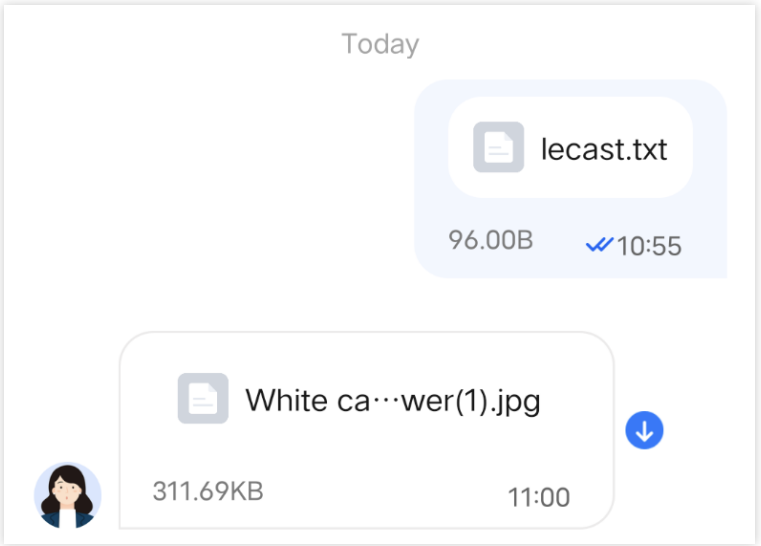
最近更新时间：2024-05-20 15:17:21

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	

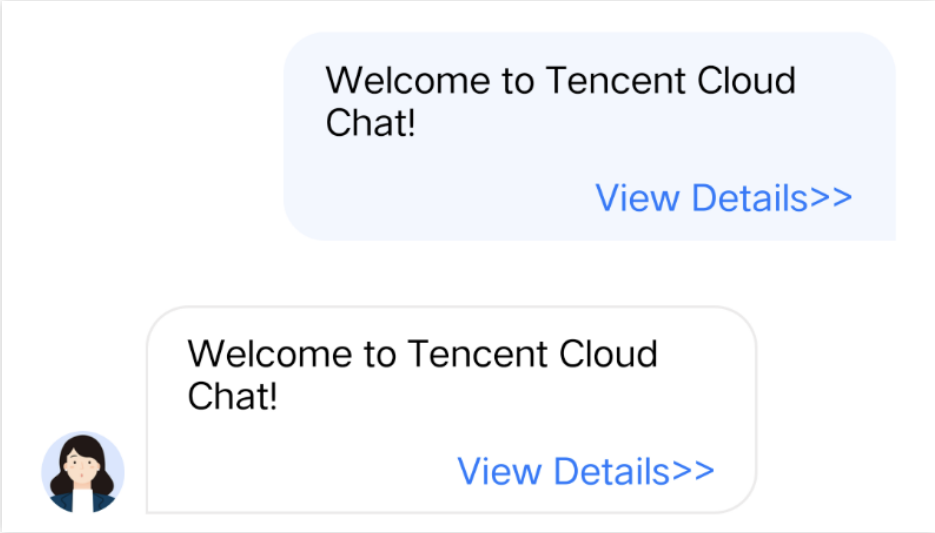
©2013-2025 Tencent Cloud International Pte. Ltd.

语音类消息	
视频类消息	
文件类消息	

自定义消息

©2013-2025 Tencent Cloud International Pte. Ltd.

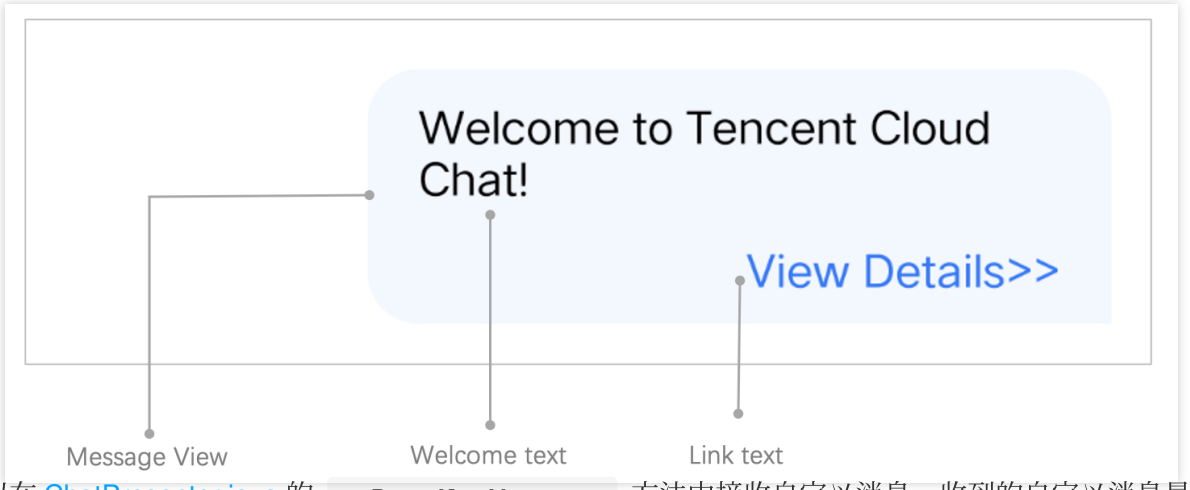
如果基本消息类型不能满足您的需求，您可以根据实际业务需求自定义消息。下文以发送一条可跳转至浏览器的超文本作为自定义消息为例，帮助您快速了解实现流程。
TUIKit 内置的自定义消息样式如下图所示：



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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

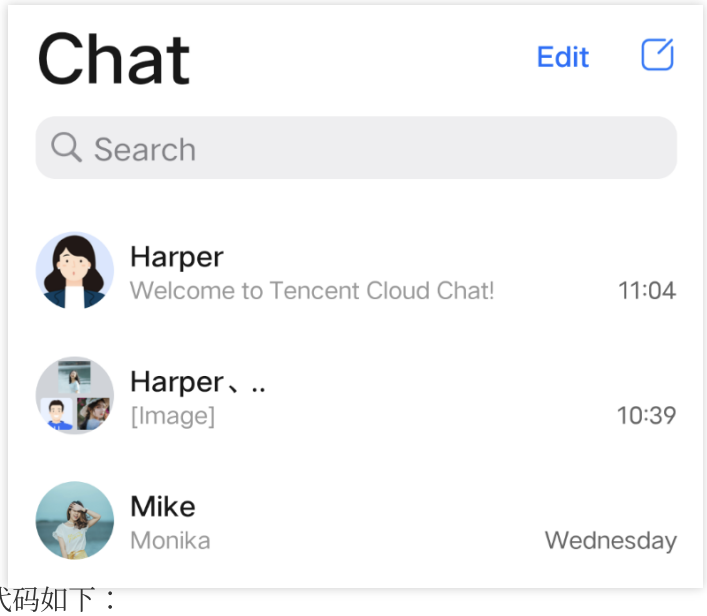
示例代码如下：

```
@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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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：

©2013-2025 Tencent Cloud International Pte. Ltd.

```
@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();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            link = ((CustomLinkMessageBean) msg).getLink();
        }
        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);
```

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

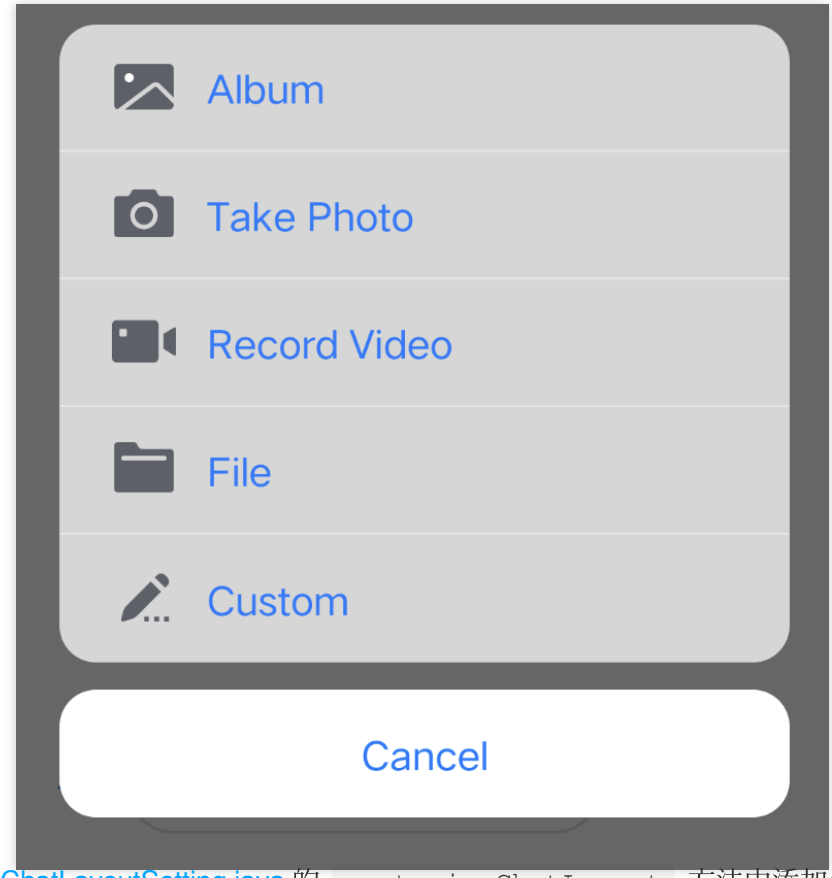
©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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);
    }
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 10:22:07

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` 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` 。

©2013-2025 Tencent Cloud International Pte. Ltd.

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`，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：

Swift

Objective-C

```
public class TUILinkCellData: TUIBubbleMessageCellData {
    var text: String = ""
    var link: String = ""

    // ...
}
```

```
@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：

Swift

Objective-C

```
public class TUILinkCellData: TUIBubbleMessageCellData {
    override public class func getCellData(message: V2TIMMessage) -> TUIMessageCell
      guard let data = message.customElem?.data,
      let param = try? JSONSerialization.jsonObject(with: data, options: .allowFrag
      else {
        return TUILinkCellData(direction: .incoming)
      }

      let cellData = TUILinkCellData(direction: message.isSelf ? .outgoing : .incom
      cellData.msgID = message.msgID
      cellData.text = param["text"] as? String ?? ""
      cellData.link = param["link"] as? String ?? ""
      cellData.avatarUrl = URL(string: message.faceURL ?? "")
      return cellData
   }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
@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：

Swift

Objective-C

```
public class TUILinkCellData: TUIBubbleMessageCellData {
    override public class func getDisplayString(message: V2TIMMessage) -> String {
        guard let data = message.customElem?.data,
                let param = try? JSONSerialization.jsonObject(with: data, options:
        else {
            return ""
        }
        return param["text"] as? String ?? ""
    }
}
```

```
@implementation TUILinkCellData

+ (NSString *)getDisplayString:(V2TIMMessage *)message {
    NSDictionary *param = [NSJSONSerialization JSONObjectWithData:message.customEle
    return param[@"text"];
}

@end
```

创建自定义 Cell

©2013-2025 Tencent Cloud International Pte. Ltd.

1. 在 `TUIChat/UI_Minimalist/Cell/Custom` 文件夹下新建 `TUILinkCell_Minimalist` 文件，继承自 `TUIBubbleMessageCell_Minimalist` ，用于绘制 `TUILinkCellData` 数据。

示例代码如下：

Swift

Objective-C

```
class TUILinkCell_Minimalist: TUIBubbleMessageCell_Minimalist {
    var myTextLabel: UILabel!
    var myLinkLabel: UILabel!
    var customData: TUILinkCellData?
    override func fill(with data: TUICommonCellData) {
        super.fill(with: data)
        if let data = data as? TUILinkCellData {
            customData = data
            myTextLabel.text = data.text

            setNeedsUpdateConstraints()
            updateConstraintsIfNeeded()
            layoutIfNeeded()
        }
    }
}
```

```
@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：

Swift

Objective-C

```
class TUILinkCell_Minimalist: TUIBubbleMessageCell {
     override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        setupViews()
    }

     private func setupViews() {
        myTextLabel = UILabel()
        myTextLabel.numberOfLines = 0
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        myTextLabel.font = UIFont.systemFont(ofSize: 15)
        myTextLabel.textAlignment = TUISwift.isRTL() ? .right : .left
        myTextLabel.textColor = TUISwift.tuiChatDynamicColor("chat_link_message_tit
        container.addSubview(myTextLabel)

        myLinkLabel = UILabel()
        myLinkLabel.text = TUISwift.timCommonLocalizableString("TUIKitMoreLinkDetai
        myLinkLabel.font = UIFont.systemFont(ofSize: 15)
        myLinkLabel.textAlignment = TUISwift.isRTL() ? .right : .left
        myLinkLabel.textColor = TUISwift.tuiChatDynamicColor("chat_link_message_sub
        container.addSubview(myLinkLabel)
    }
}
```

```
@implementation TUILinkCell_Minimalist
- (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：

Swift

Objective-C

```
class TUILinkCell_Minimalist: TUIBubbleMessageCell_Minimalist {
    override func fill(with data: TUICommonCellData) {
        super.fill(with: data)
        if let data = data as? TUILinkCellData {
            customData = data
            myTextLabel.text = data.text

            setNeedsUpdateConstraints()
            updateConstraintsIfNeeded()
            layoutIfNeeded()
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        }
    }
}
```

```
@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：

Swift

Objective-C

```
override func layoutSubviews() {
    super.layoutSubviews()
    // Custimization
}
```

```
// 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：

Swift

Objective-C

```
override class func getContentSize(_ data: TUIMessageCellData) -> CGSize {
    guard let linkCellData = data as? TUILinkCellData else {
        assertionFailure("data must be kind of TUILinkCellData")
        return .zero
    }

    let textMaxWidth = 245
    let font = UIFont.systemFont(ofSize: 15)
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    let attributes: [NSAttributedString.Key: Any] = [NSAttributedString.Key.font: f
    var size = CGSize(width: textMaxWidth, height: Int.max)

    let rect = linkCellData.text.boundingRect(with: size,
                                              options: [.usesLineFragmentOrigin,
                                              attributes: attributes,
                                              context: nil)
    size = CGSize(width: textMaxWidth + 15, height: Int(rect.size.height) + 56)
    return size
}
```

```
+ (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：

Swift

Objective-C

```
public class TUIMessageCellConfig_Minimalist: NSObject {
    // ...

    static func registerExternalCustomMessageInfo() {
        /*
         Insert your own custom message UI here, your businessID can not be same wi
         Example:
         registerCustomMessageCell(#your message cell#, messageCellData: #your mess
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
         */
    }
}
```

```
@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` 的 `registerCustomMessage` 函数里主动注册 `cell` 和 `cellData` 信息。

示例代码如下：

Swift

Objective-C

```
// 自定义消息 businessID 不能重复
TUIChatConfig_Minimalist.shared.registerCustomMessage(businessID: "text_link", mess
```

```
// 自定义消息 businessID（注意不能重复）
#define BussinessID_TextLink @"text_link"

/** 向 TUIChat 注册自定义消息。三个参数分别为
* @param businessID 自定义消息 businessID
* @param messagellClass 自定义消息 NSString 类型
* @param messageCellDataClassName 自定义消息 NSString 类型
*/
- (void)registerCustomMessageCell {
    [TUIChatConfig.defaultConfig registerCustomMessage:BussinessID_TextLink
                                    messageCellClassName:@"TUILinkCell"
                               messageCellDataClassName:@"TUILinkCellData"
    ];
}
```

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

©2013-2025 Tencent Cloud International Pte. Ltd.

`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：
Swift
Objective-C

```
class TUIChatDataProvider: TUIChatBaseDataProvider {
    // ...

    private func createCustomInputMoreActionItemList(model: TUIChatConversationMode
        if self.customInputMoreActionItemList.isEmpty {
            var arrayM: [TUICustomActionSheetItem] = []

            let showCustom = TUIChatConfig.shared.enableWelcomeCustomMessage && mod
            if showCustom {
                let link = TUICustomActionSheetItem(title: TUISwift.timCommonLocali
                    guard let self else { return }
                    let text = TUISwift.timCommonLocalizableString("TUIKitWelcome")
                    var homePageLink = TUITencentCloudHomePageEN
                    let language = TUIGlobalization.getPreferredLanguage() ?? ""
                    if language.contains("zh-") {
                        homePageLink = TUITencentCloudHomePageCN
                    }
                    do {
                        let param: [String: Any] = ["businessID": "text_link", "tex
                        let data = try JSONSerialization.data(withJSONObject: param
                        let message = TUIMessageDataProvider.getCustomMessageWithJs
                        self.delegate?.dataProvider(self, sendMessage: message)
                    } catch {
                        print("[\\(self)] Post Json Error")
                    }
                }
                link.priority = 100
                arrayM.append(link)
            }

            if let items = model.customizedNewItemsInMoreMenu as? [TUICustomActionS
                arrayM.append(contentsOf: items)
            }
            self.customInputMoreActionItemList = arrayM
        }
        return self.customInputMoreActionItemList
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }

    // For Minimalist Edition.
    func getInputMoreActionItemList(userID: String, groupID: String, conversationMo
        var result: [TUICustomActionSheetItem] = []
        result.append(contentsOf: self.createBuiltInInputMoreActionItemList(model:
        result.append(contentsOf: self.createCustomInputMoreActionItemList(model: c

        // ...
    }
}
```

```
@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;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
@end
```

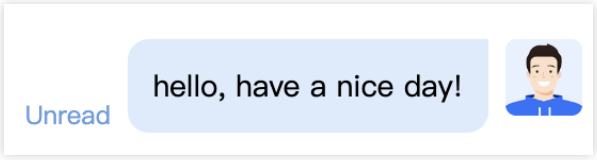
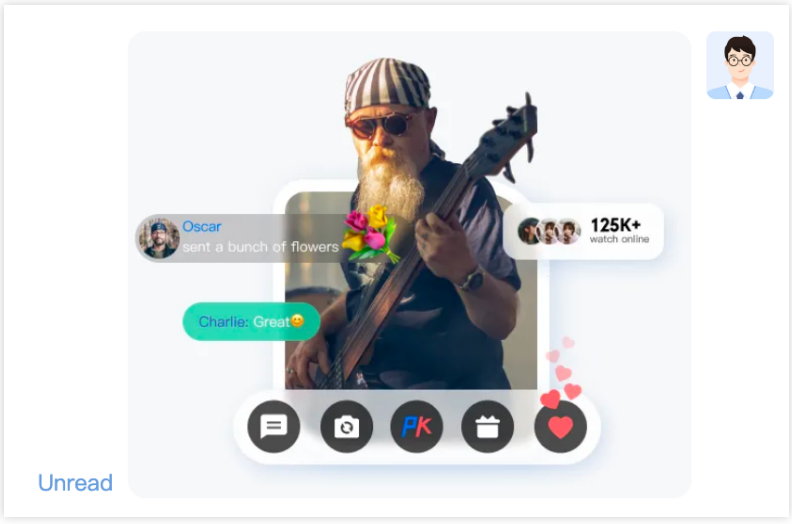
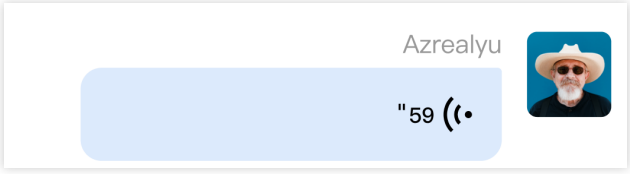
©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp (Vue)

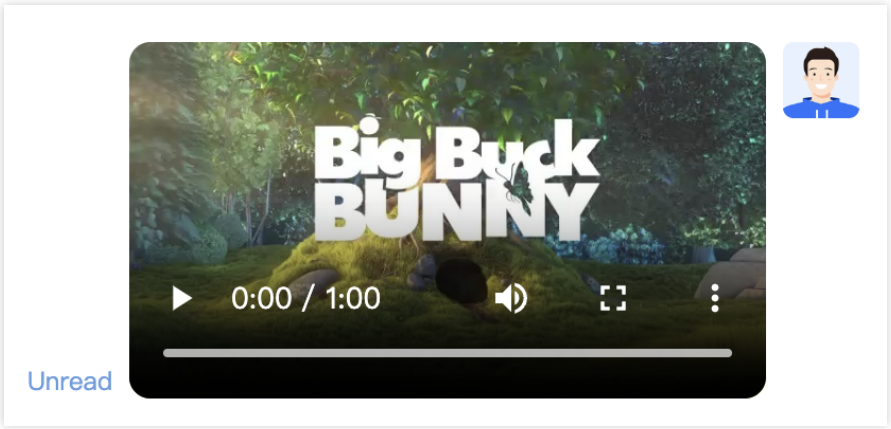
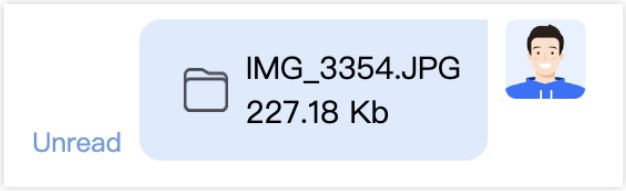
最近更新时间：2024-01-31 17:20:35

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	

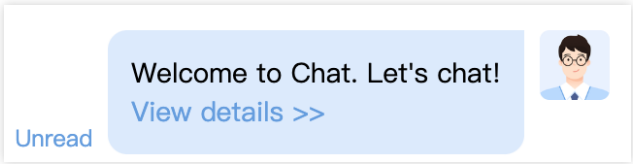
©2013-2025 Tencent Cloud International Pte. Ltd.

视频类消息	
文件类消息	

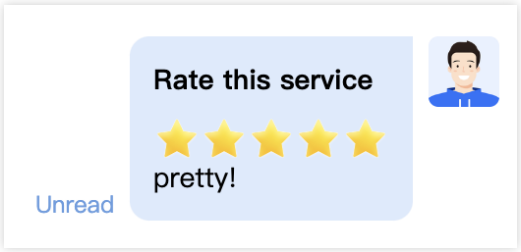
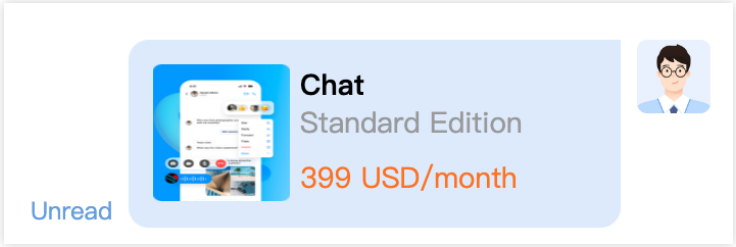
自定义消息

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	

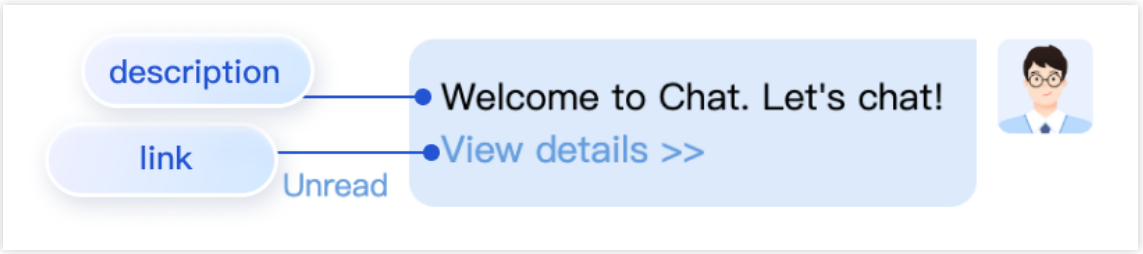
©2013-2025 Tencent Cloud International Pte. Ltd.

	
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-`

©2013-2025 Tencent Cloud International Pte. Ltd.

`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";
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
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>`

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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

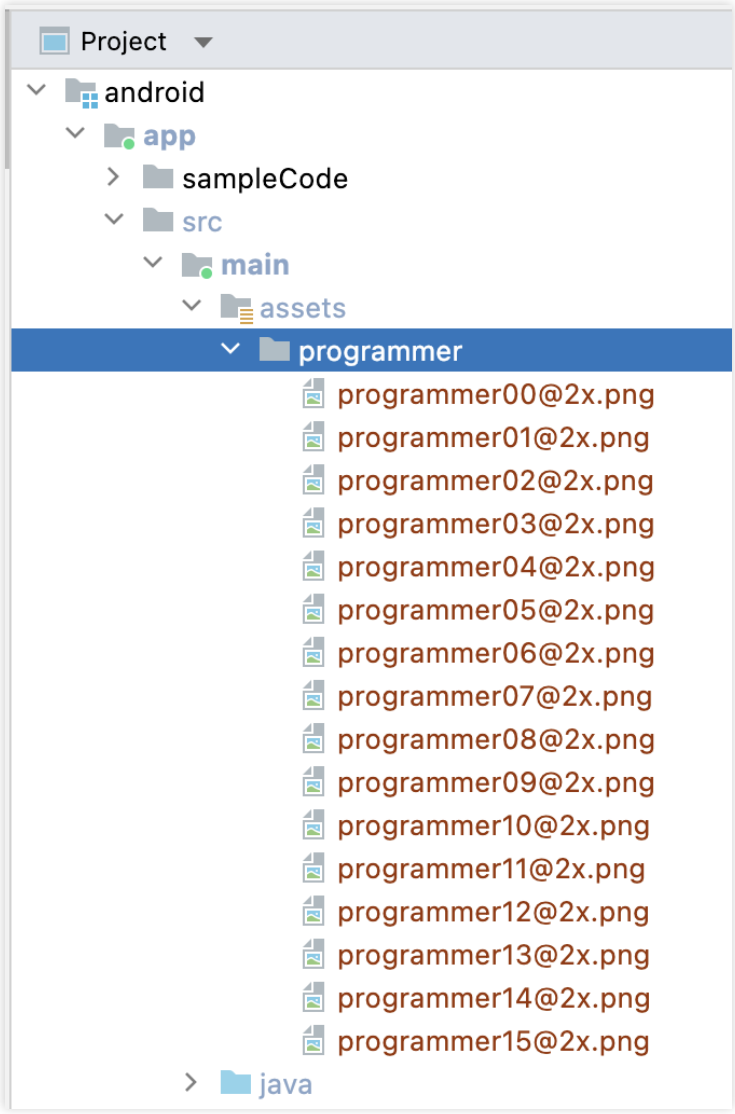
TUIChat 支持从沙盒、assets 目录以及网络路径加载自定义表情。

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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++) {
```

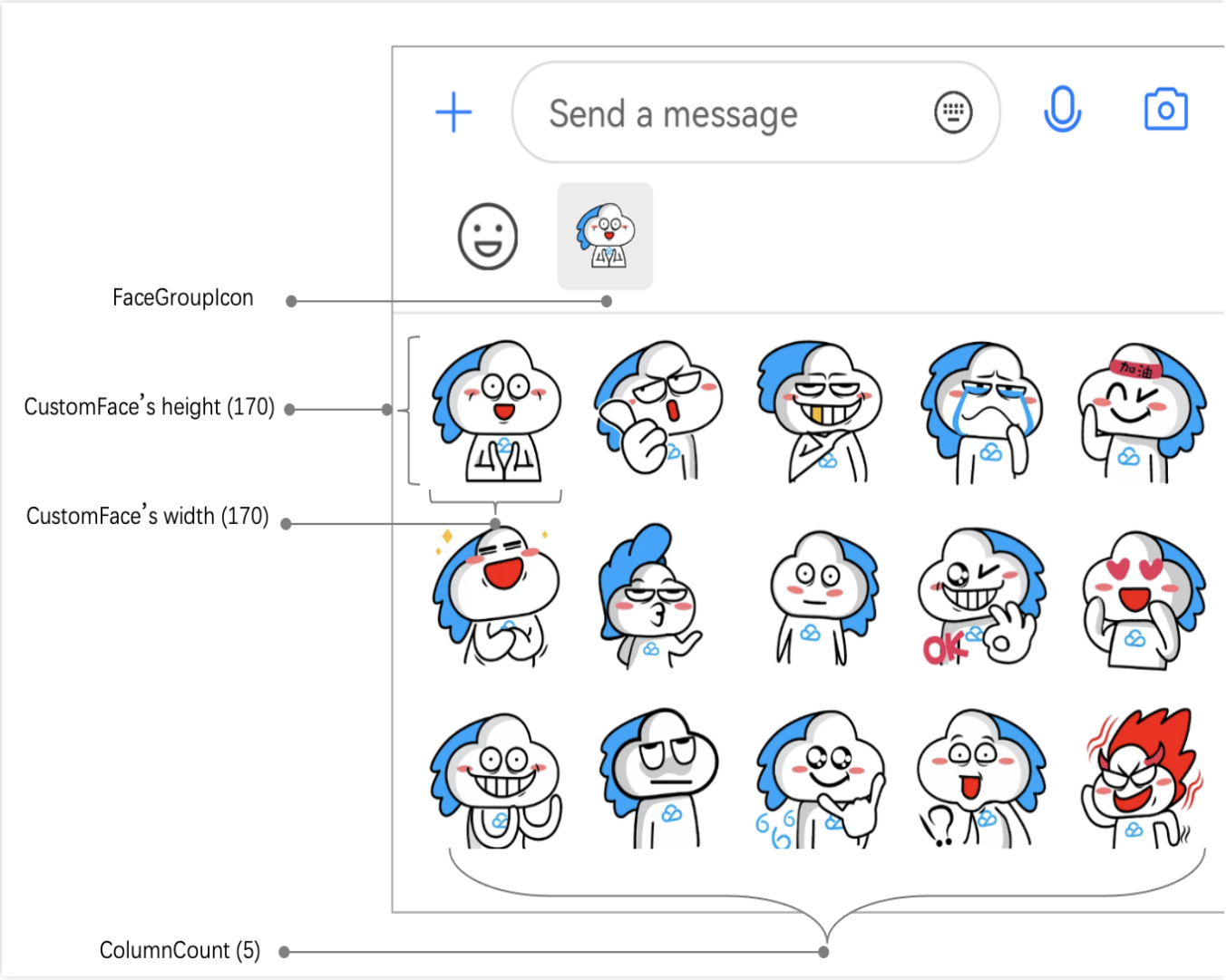
©2013-2025 Tencent Cloud International Pte. Ltd.

```
            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：

©2013-2025 Tencent Cloud International Pte. Ltd.



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,
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        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;
}
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：

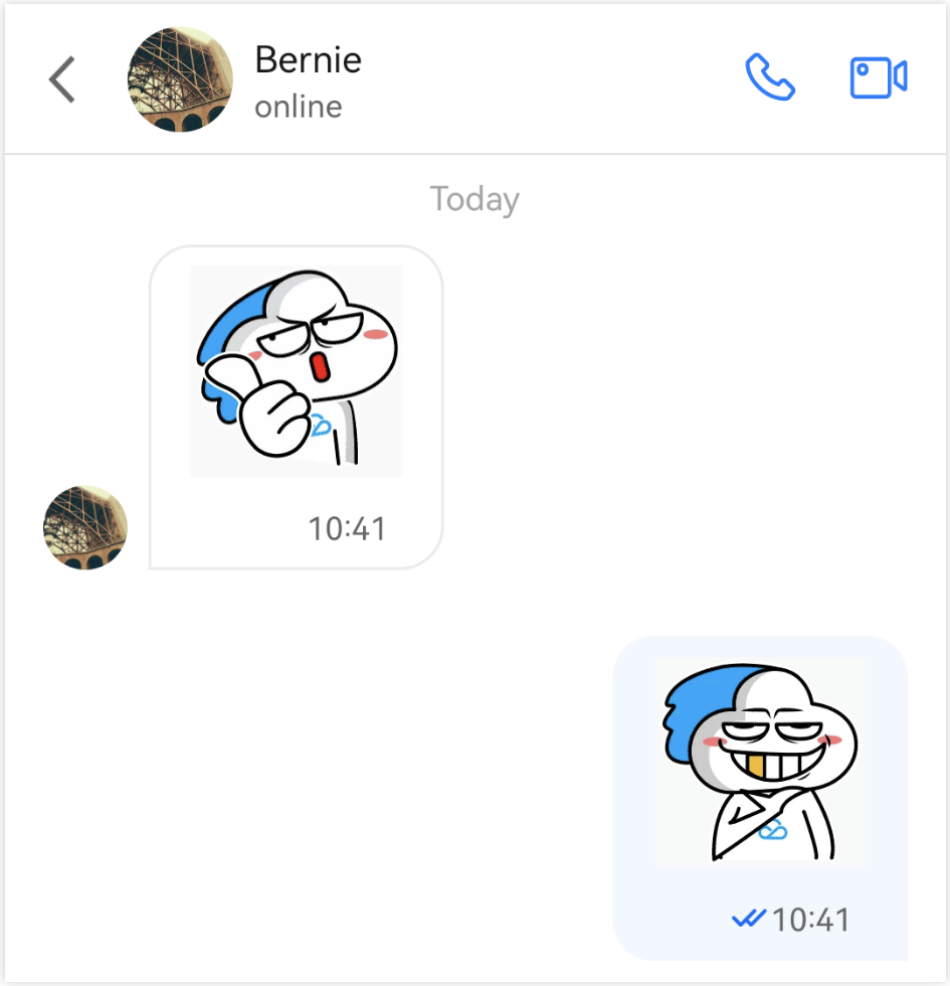
```
String faceUrl = "";
List<FaceGroup> faceGroupList = FaceManager.getFaceGroupList();
for(FaceGroup faceGroup : faceGroupList) {
    if (faceGroup.getGroupID() == faceGroupID) {
        ChatFace face = faceGroup.getFace(faceKey);
        if (face != null) {
            faceUrl = face.getFaceUrl();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
            }
        }
    }

    // load faceUrl into view
```

渲染效果

渲染效果如图所示：



©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

最近更新时间：2025-05-29 10:22:07

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。

Swift
Objective-C

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    // ...
    self.setupCustomSticker()
    return YES
}

func setupCustomSticker() {
    guard let service = TIMCommonMediator.shared.getObject(for: TUIEmojiMeditorProt
        assertionFailure("There's not any object implement TUIEmojiMeditorProtocol"
        return
    }
    let bundlePath = TUISwift.tuiBundlePath("CustomFaceResource", key: "TIMAppKit.T

    // 4350 group
    var faces4350 = [TUIFaceCellData]()
    for i in 0...17 {
        let data = TUIFaceCellData()
        let name = String(format: "yz%02d", i)
        let path = "4350/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4350.append(data)
    }
    if faces4350.count > 0 {
        let group4350 = TUIFaceGroup()
        group4350.groupIndex = 1
        group4350.groupPath = bundlePath + "/4350/"
        group4350.faces = faces4350
        group4350.rowCount = 2
        group4350.itemCountPerRow = 5
        group4350.menuPath = bundlePath + "/4350/menu"
        service.appendFaceGroup(group4350)
    }

    // 4351 group
    var faces4351 = [TUIFaceCellData]()
    for i in 0...15 {
        let data = TUIFaceCellData()
        let name = String(format: "ys%02d", i)
        let path = "4351/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4351.append(data)
    }
    if faces4351.count > 0 {
        let group4351 = TUIFaceGroup()
        group4351.groupIndex = 2
        group4351.groupPath = bundlePath + "/4351/"
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        group4351.faces = faces4351
        group4351.rowCount = 2
        group4351.itemCountPerRow = 5
        group4351.menuPath = bundlePath + "/4351/menu"
        service.appendFaceGroup(group4351)
    }

    // 4352 group
    var faces4352 = [TUIFaceCellData]()
    for i in 0...16 {
        let data = TUIFaceCellData()
        let name = String(format: "gcs%02d", i)
        let path = "4352/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4352.append(data)
    }
    if faces4352.count > 0 {
        let group4352 = TUIFaceGroup()
        group4352.groupIndex = 3
        group4352.groupPath = bundlePath + "/4352/"
        group4352.faces = faces4352
        group4352.rowCount = 2
        group4352.itemCountPerRow = 5
        group4352.menuPath = bundlePath + "/4352/menu"
        service.appendFaceGroup(group4352)
    }
}
```

```
- (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];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        // 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）。

表情面板高级配置

调整表情面板顺序

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

Swift

Objective-C

```
func setupCustomSticker() {
    guard let service = TIMCommonMediator.shared.getObject(for: TUIEmojiMeditorProt
        assertionFailure("There's not any object implement TUIEmojiMeditorProtocol"
        return
    }
    let bundlePath = TUISwift.tuiBundlePath("CustomFaceResource", key: "TIMAppKit.T

    // 4350 group
    var faces4350 = [TUIFaceCellData]()
    for i in 0...17 {
        let data = TUIFaceCellData()
        let name = String(format: "yz%02d", i)
        let path = "4350/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4350.append(data)
    }
    if faces4350.count > 0 {
        let group4350 = TUIFaceGroup()
        group4350.groupIndex = 1
        group4350.groupPath = bundlePath + "/4350/"
        group4350.faces = faces4350
        group4350.rowCount = 2
        group4350.itemCountPerRow = 5
        group4350.menuPath = bundlePath + "/4350/menu"
        service.appendFaceGroup(group4350)
    }

    // 4351 group
    var faces4351 = [TUIFaceCellData]()
    for i in 0...15 {
        let data = TUIFaceCellData()
        let name = String(format: "ys%02d", i)
        let path = "4351/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        faces4351.append(data)
    }
    if faces4351.count > 0 {
        let group4351 = TUIFaceGroup()
        group4351.groupIndex = 2
        group4351.groupPath = bundlePath + "/4351/"
        group4351.faces = faces4351
        group4351.rowCount = 2
        group4351.itemCountPerRow = 5
        group4351.menuPath = bundlePath + "/4351/menu"
        service.appendFaceGroup(group4351)
    }

    // 4352 group
    var faces4352 = [TUIFaceCellData]()
    for i in 0...16 {
        let data = TUIFaceCellData()
        let name = String(format: "gcs%02d", i)
        let path = "4352/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4352.append(data)
    }
    if faces4352.count > 0 {
        let group4352 = TUIFaceGroup()
        group4352.groupIndex = 3
        group4352.groupPath = bundlePath + "/4352/"
        group4352.faces = faces4352
        group4352.rowCount = 2
        group4352.itemCountPerRow = 5
        group4352.menuPath = bundlePath + "/4352/menu"
        service.appendFaceGroup(group4352)
    }
}
```

```
- (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];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        // 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。

Swift

Objective-C

```
func setupCustomSticker() {
    // ...

    // 4350 group
    var faces4350 = [TUIFaceCellData]()
    for i in 0...17 {
        let data = TUIFaceCellData()
        let name = String(format: "yz%02d", i)
        let path = "4350/\\(name)"
        data.name = name
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        data.path = bundlePath + "/" + path
        faces4350.append(data)
    }
    if faces4350.count > 0 {
        let group4350 = TUIFaceGroup()
        group4350.groupIndex = 1
        group4350.groupPath = bundlePath + "/4350/"
        group4350.faces = faces4350
        group4350.rowCount = 2
        group4350.itemCountPerRow = 5
        group4350.menuPath = bundlePath + "/4350/menu"
        service.appendFaceGroup(group4350)
    }
    // ...
}
```

```
- (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。

Swift

Objective-C

```
func setupCustomSticker() {
    // ...

    // 4350 group
    var faces4350 = [TUIFaceCellData]()
    for i in 0...17 {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        let data = TUIFaceCellData()
        let name = String(format: "yz%02d", i)
        let path = "4350/\\(name)"
        data.name = name
        data.path = bundlePath + "/" + path
        faces4350.append(data)
    }
    if faces4350.count > 0 {
        let group4350 = TUIFaceGroup()
        group4350.groupIndex = 1
        group4350.groupPath = bundlePath + "/4350/"
        group4350.faces = faces4350
        group4350.rowCount = 2
        group4350.itemCountPerRow = 5
        group4350.menuPath = bundlePath + "/4350/menu"
        service.appendFaceGroup(group4350)
    }
    // ...
}
```

```
- (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。

©2013-2025 Tencent Cloud International Pte. Ltd.

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。

Swift

Objective-C

```
public func faceVerticalView(_ faceView: TUIFaceVerticalView, didSelectItemAtIndexP
    let group = faceView.faceGroups[indexPath.section]
    if let face = group.faces?[indexPath.row] as? TUIFaceCellData {
        if group.isNeedAddInInputBar {
            inputBar?.addEmoji(face)
            updateRecentMenuQueue(face.name ?? "")
        } else {
            let message = V2TIMManager.sharedInstance().createFaceMessage(index: In
            delegate?.inputController(self, didSendMessage: message)
        }
    }
}
```

```
- (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` 。

©2013-2025 Tencent Cloud International Pte. Ltd.

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 。

Swift

Objective-C

```
override class func getCellData(message: V2TIMMessage) -> TUIMessageCellData {
    guard let elem = message.faceElem else { return TUIFaceMessageCellData(directio
    let faceData = TUIFaceMessageCellData(direction: message.isSelf ? .outgoing : .
    faceData.groupIndex = elem.index
    if let data = elem.data {
        faceData.faceName = String(data: data, encoding: .utf8)
    }

    if let groups = TIMConfig.shared.faceGroups {
        for group in groups {
            if group.groupIndex == faceData.groupIndex {
                if let url = URL(string: group.groupPath ?? "") {
                    let path = url.appendingPathComponent(faceData.faceName ?? "").
                    faceData.path = path
                }
                break
            }
        }
    }

    faceData.reuseId = "TFaceMessageCell"
    return faceData
}
```

```
+ (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;
        }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    }
    faceData.reuseId = TFaceMessageCell_ReuseId;
    return faceData;
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Web & H5 & Uniapp

最近更新时间：2024-11-04 17:22:40

说明：

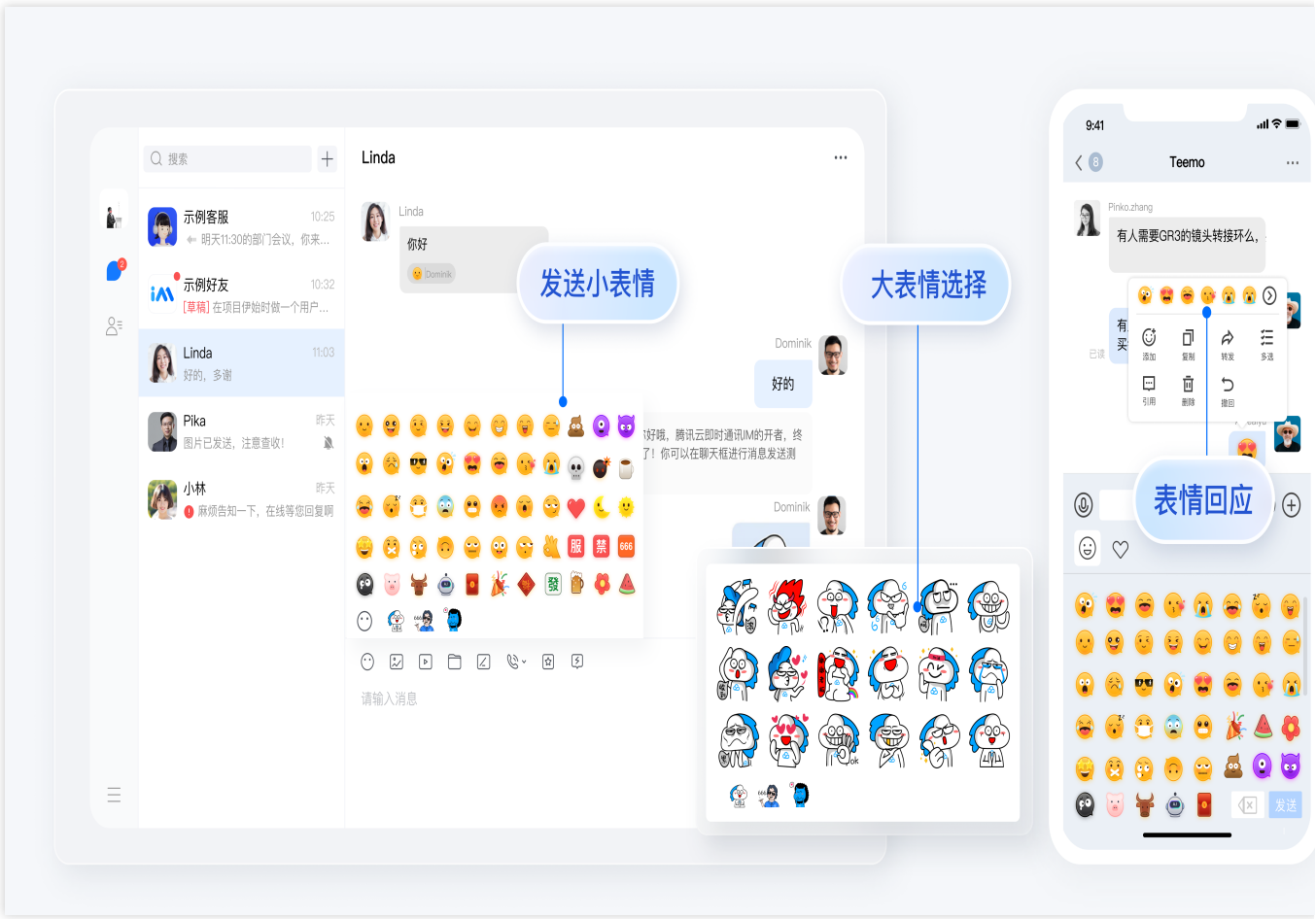
本文所述的自定义表情能力用法适用于 **@tencentcloud/chat-uikit-vue ≥ 2.2.0** 或 **@tencentcloud/chat-uikit-uniapp ≥ 2.2.0**。

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：

©2013-2025 Tencent Cloud International Pte. Ltd.



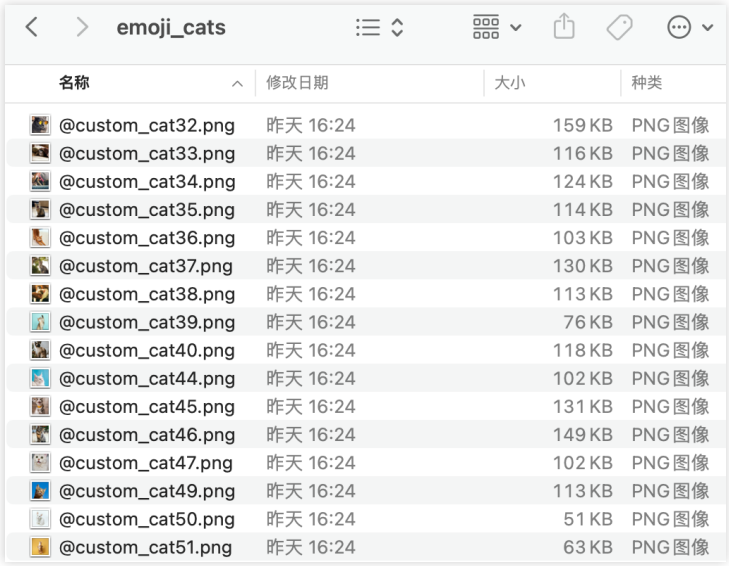
自定义大表情

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。

©2013-2025 Tencent Cloud International Pte. Ltd.



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：

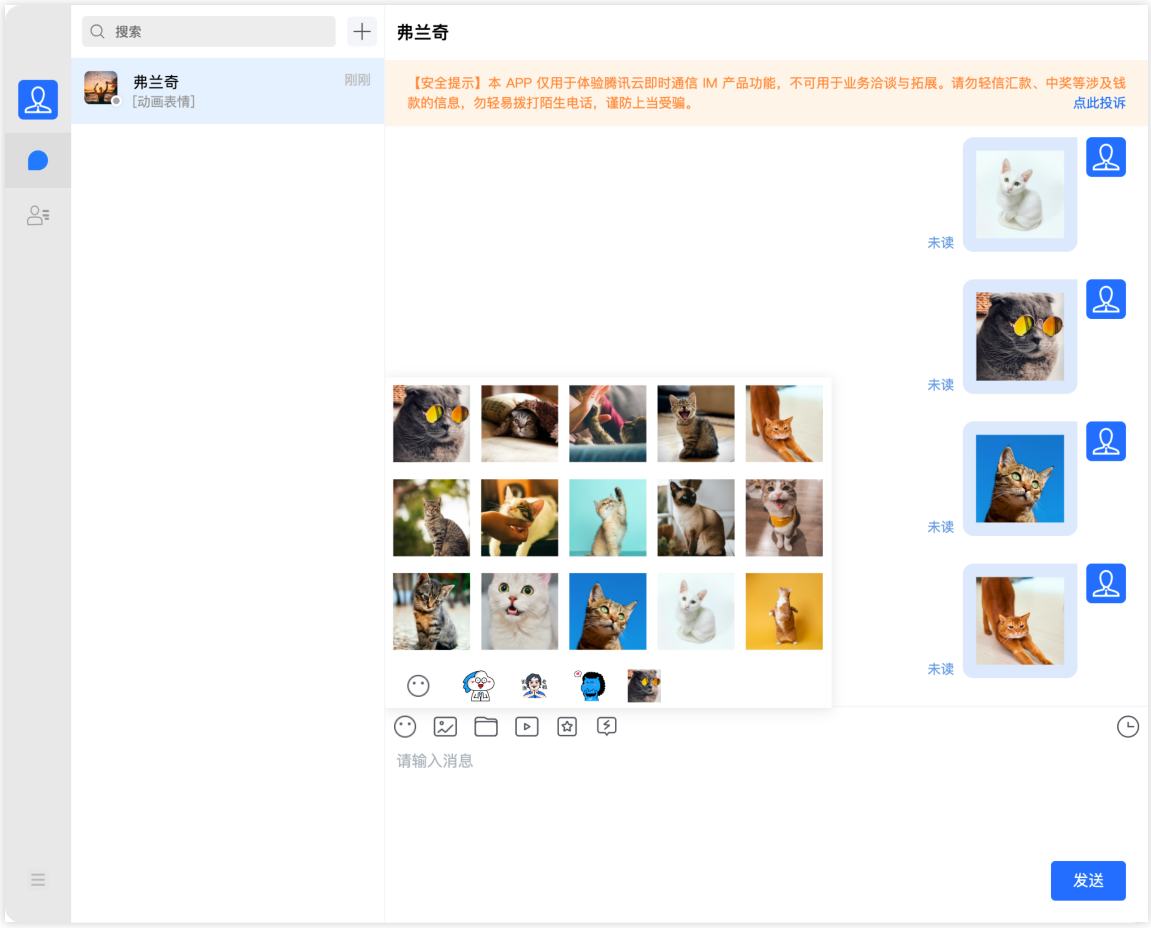
如您有多端互通需求，请您务必保证每组自定义表情的 emojiGroupID 与其中每个表情的 key 在所有端定义一致。

```
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：

©2013-2025 Tencent Cloud International Pte. Ltd.

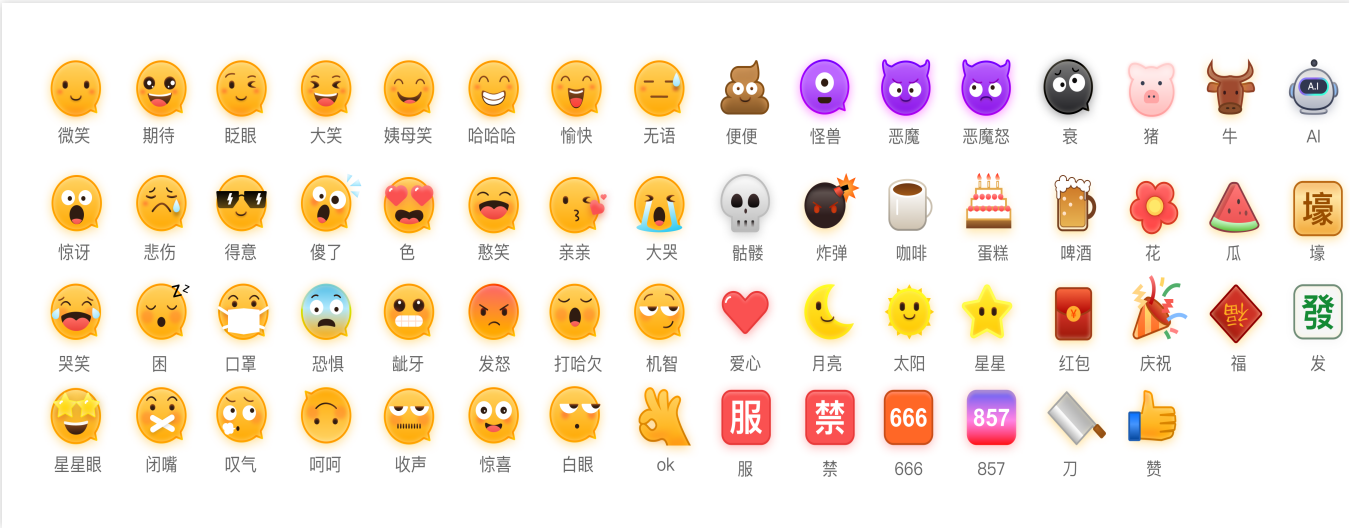


自定义小表情

内置小表情资源

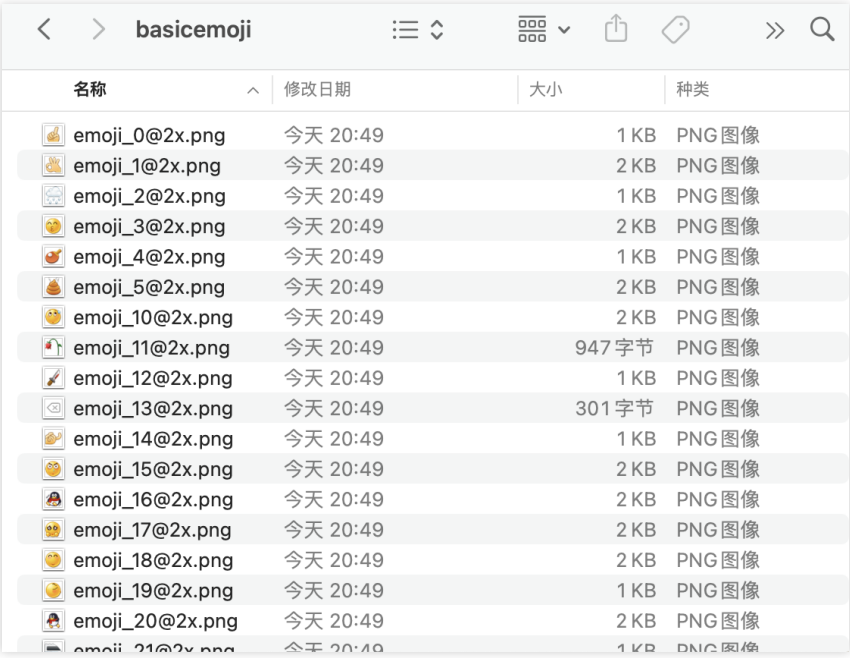
TUIKit 默认包含一套黄脸表情资源。黄脸表情版权归腾讯云所有，如需授权使用，请您联系我们。黄脸表情图片与含义对照如下：

©2013-2025 Tencent Cloud International Pte. Ltd.



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。**

©2013-2025 Tencent Cloud International Pte. Ltd.

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
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

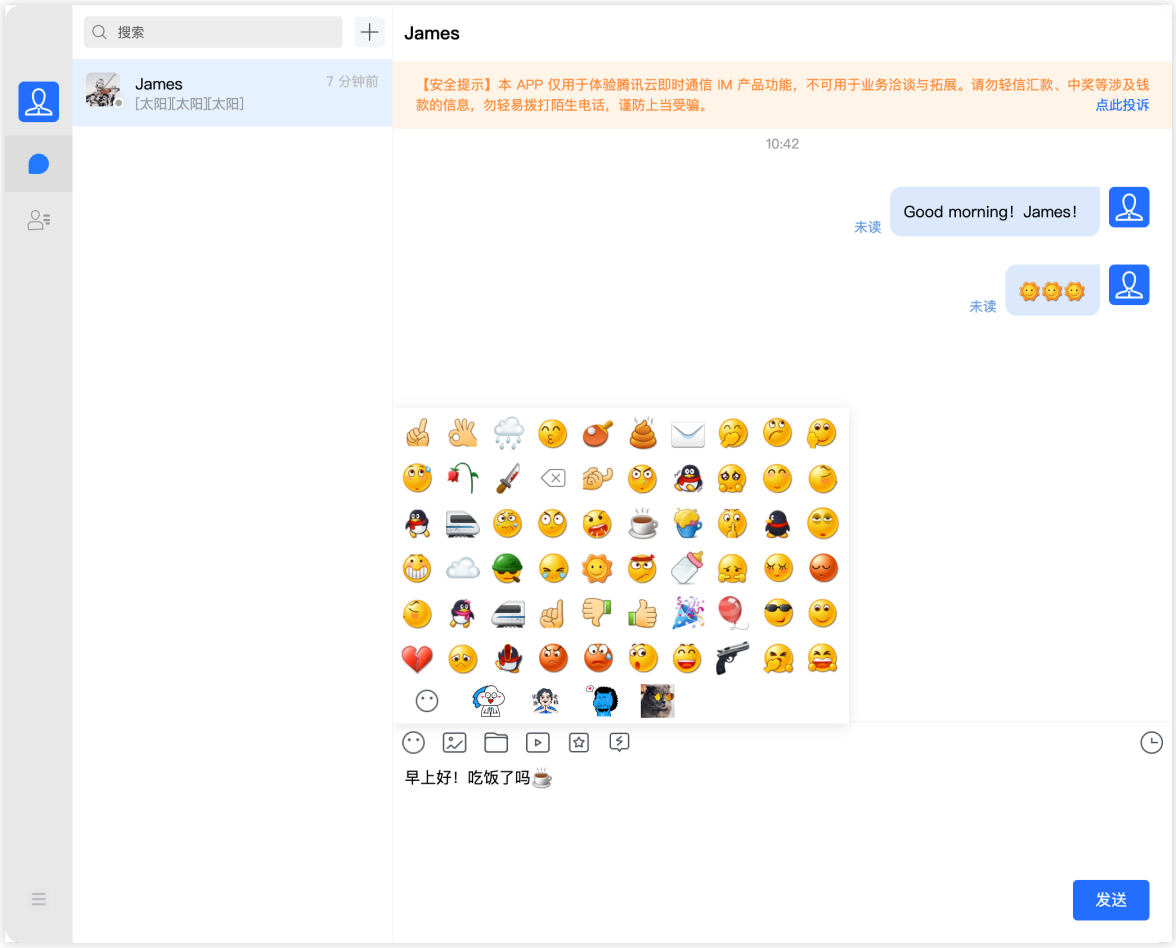
目标文件目录：`TUIKit/components/TUIChat/emoji-config/locales/en.ts`

```
// 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

©2013-2025 Tencent Cloud International Pte. Ltd.

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：

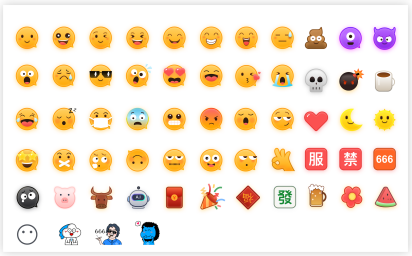


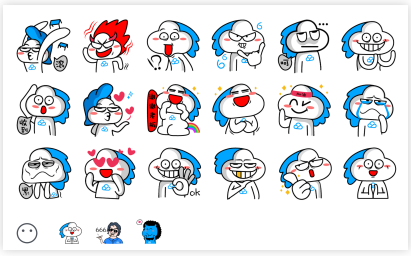
删除系统自带表情

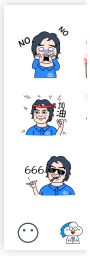
系统自带表情 emojiGroupID 对应表情包如下：

emojiGroupID = 0	emojiGroupID = 1	emojiGroupID

©2013-2025 Tencent Cloud International Pte. Ltd.







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。

```
// 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'],
  // },
  ...
];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

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		

©2013-2025 Tencent Cloud International Pte. Ltd.



图1

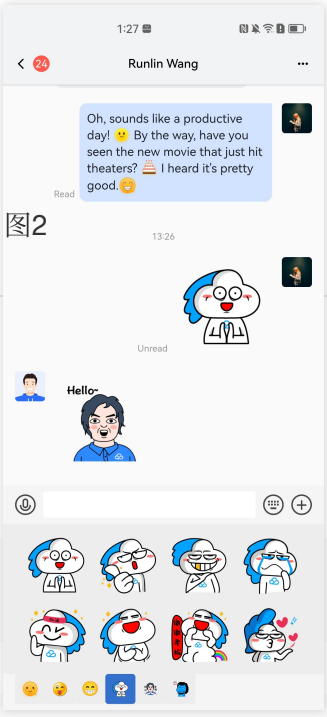


图2

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,
  ),
)
```

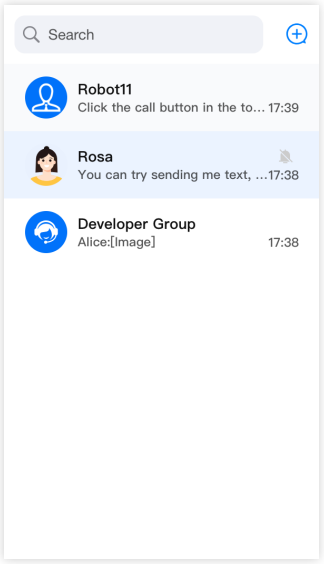
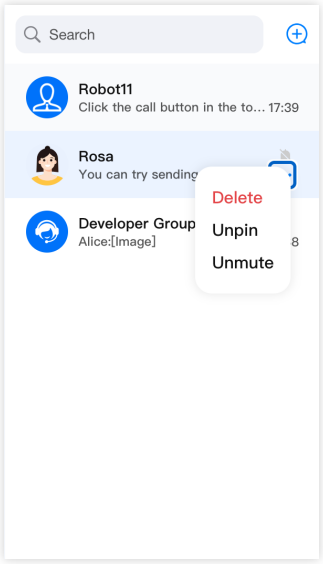
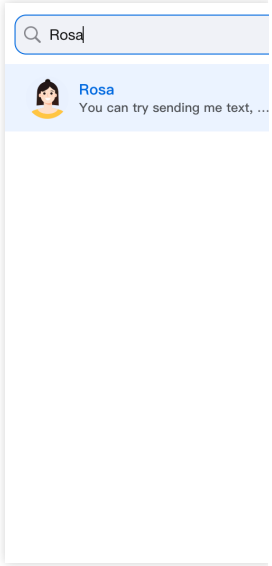
在 `TencentCloudChatStickerInitData` 对象中，您可以选择启用哪些默认的表情包，并根据需要添加更多的表情包。

©2013-2025 Tencent Cloud International Pte. Ltd.

自定义 UI 组件
Conversation List（React）
ConversationList

最近更新时间：2024-10-09 15:11:35

`ConversationList` 组件主要负责会话列表功能，它包含了 Header 部分和 List 部分，具有搜索会话、创建会话、会话置顶、会话删除等功能。
本文档将详细介绍该组件的基础使用，组件定制，自由组合以及组件的 props 参数列表。

会话列表	会话操作	会话搜索
		

基础使用

`ConversationList` 组件没有任何必须属性，您可以通过以下代码使用 `ConversationList` 。

```
import { UIKitProvider, ConversationList } from '@tencentcloud/chat-uikit-react';

const App = () => {
  return (
    <UIKitProvider>
      <ConversationList />
    </UIKitProvider>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  );
};
```

Props

参数名	类型	默认值
enableSearch	`Boolean`	`true`
enableCreate	`Boolean`	`true`
enableActions	`Boolean`	`true`
actionsConfig	IConversationActionsConfig	-
Header	`ReactElement`	`Header`
List	`ReactElement`	`List`
Preview	`ReactElement`	ConversationPreview

©2013-2025 Tencent Cloud International Pte. Ltd.

ConversationCreate	`ReactElement`	`ConversationCreate`
ConversationSearch	`ReactElement`	ConversationSearch
ConversationActions	`ReactElement`	ConversationActions
Avatar	`ReactElement`	`Avatar`
PlaceholderEmptyList	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.NO_CONVERSATIONS} />`
PlaceholderLoading	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.LOADING} />`
PlaceholderLoadError	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.WRONG} />`
filter	(conversationList: IConversationModel[]) => IConversationModel[]	-
sort	(conversationList: IConversationModel[]) =>	-

©2013-2025 Tencent Cloud International Pte. Ltd.

	IConversationModel[]	
onConversationSelect	(conversation: IConversationModel) => void;	-
onBeforeCreateConversation	(params: CreateParams) => CreateParams;	-
onConversationCreate	(conversation: IConversationModel) => void;	-
className	`String`	-
style	`React.CSSProperties`	-

©2013-2025 Tencent Cloud International Pte. Ltd.

自定义组件

`ConversationList` 为用户自定义提供了丰富且多维度的 Props 接口，允许用户自定义功能、UI、模块等。`ConversationList` 组件提供了多个可替换的子组件，允许用户自定义 `Header`，`List`，`ConversationPreview`，`ConversationCreate`，`ConversationSearch`，`ConversationActions`，`Avatar`，`Placeholder` 等。同时，用户还可以利用默认子组件进行二次开发定制。

基础功能开关

通过设置 `enableSearch`，`enableCreate` 和 `enableActions` 参数，你可以灵活地控制 `ConversationList` 中的搜索会话、创建会话和会话操作功能的显示。

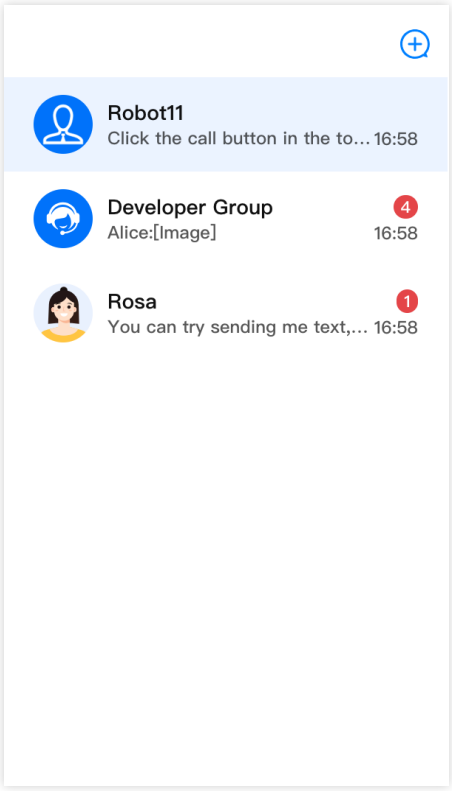
```
<ConversationList enableSearch={false} />
```

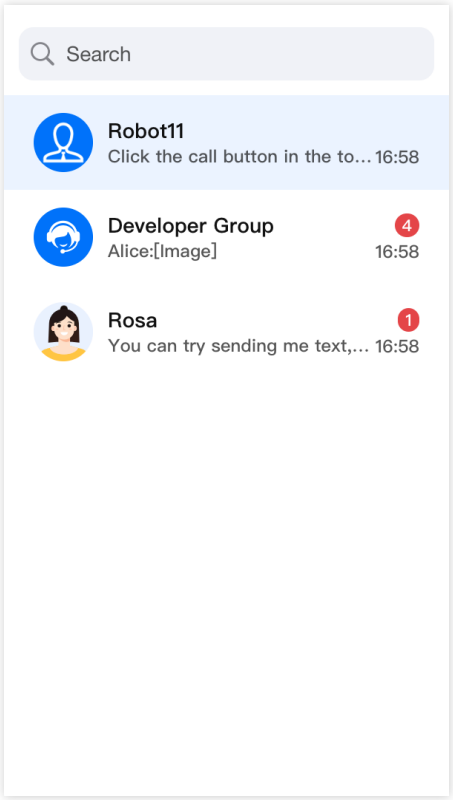
```
<ConversationList enableCreate={false} />
```

```
<ConversationList enableActions={false} />
```

`enableSearch={false}`	`enableCreate={false}`	`enab`

©2013-2025 Tencent Cloud International Pte. Ltd.





数据筛选和排序

ConversationList 组件提供了 `filterConversation` 和 `sortConversation` 属性，可以让你对会话列表数据进行筛选和排序。

筛选会话

要筛选会话列表数据，您可以给 `filterConversation` 属性传递一个筛选函数。这个函数接收一个 IConversationModel 数组作为参数，然后应该返回一个新数组，只包含符合你筛选条件的会话。
下面是一个使用 `filterConversation` 属性来只显示“包含未读消息”的会话的例子：

```
<ConversationList
  filter={(conversationList: IConversationModel[]) =>
    conversationList.filter(conversation => conversation.unreadCount > 0)}
/>
```

排序会话

要排序会话列表数据，您可以给 `sortConversation` 属性传递一个排序函数。这个函数接收一个 IConversationModel 数组作为参数，然后应该返回一个新数组，按照你的排序标准对会话进行排序。
下面是一个使用 `sortConversation` 属性来按照“最新消息时间倒序”排序会话列表的例子：

```
<ConversationList
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  sort={(conversationList: IConversationModel[]) =>
    conversationList.sort(
      (a, b) => (+(b?.lastMessage?.lastTime || 0)) - (+(a?.lastMessage?.lastTime ||
    )}
/>
```

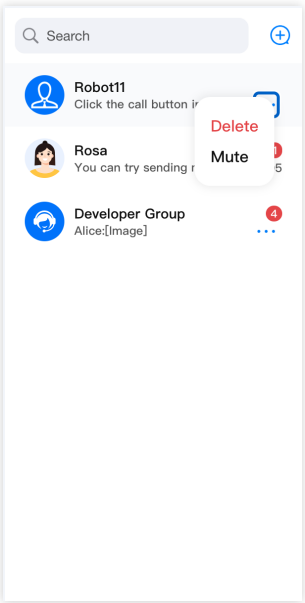
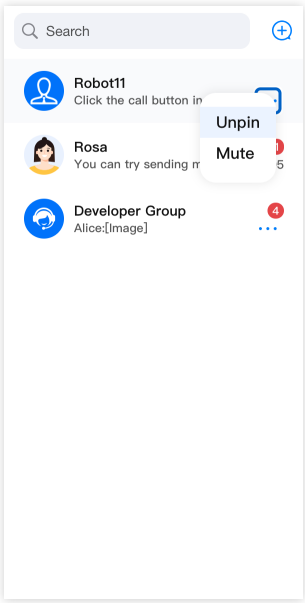
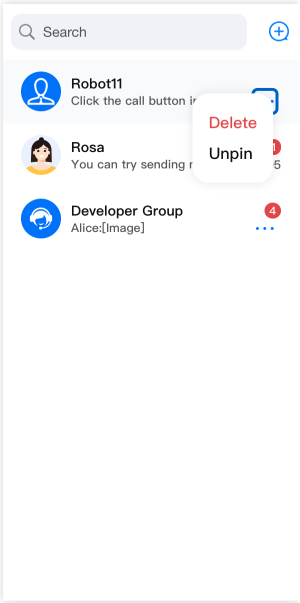
通过使用 `filter` 和 `sort` 属性，你可以根据你的需求有效地筛选和排序会话列表数据。

自定义 actionsConfig

利用 actionsConfig 对 ConversationActions 的基础功能进行控制。

更多定制请详情参见 ConversationActions 章节。

```
<ConversationList
  actionsConfig={{
    enablePin: false,
    onConversationDelete: (conversation: IConversationModel) => { console.log('Dele
    customConversationActions: {
      'custom-actions-1': {
        label: 'custom-actions',
        onClick: (conversation: IConversationModel) => { console.log(conversation);
      },
    },
  }}
/>
```

`enablePin: false`	`enableDelete: false`	`enableMute: false`
		

©2013-2025 Tencent Cloud International Pte. Ltd.

自定义 Placeholder

您可以通过传入 `PlaceholderEmptyList` 、 `PlaceholderLoading` 以及 `PlaceholderLoadError` 来定制不同状态下的列表展示。
以下是定制一个新的 `PlaceholderLoading` 示例：

```
<ConversationList
  PlaceholderEmptyList={<div>Empty List!!!</div>}
/>
```

自定义 Header

`ConversationListHeader` 负责 ConversationList Header 部分的渲染工作，作为包裹层渲染默认 `ConversationSearch` 与 `ConversationCreate` 。您可以通过传入 left、right 等属性进行自定义，同时，您也可以自定义整个组件。

Props

参数名	类型	默认值	说明
children	`ReactNode`	-	自定义会话列表头部中心组件。 在 `<ConversationList>` 中使用时，会默认传入 `<ConversationSearch>` 和 `<ConversationCreate>` 。
left	`ReactElement`	-	自定义会话列表头部左侧组件。
right	`ReactElement`	-	自定义会话列表头部右侧组件。
className	`String`	-	指定根元素类的 CSS 自定义名称。
style	`React.CSSProperties`	-	指定根元素样式的自定义样式。

基础定制

以下是在 Header 组件右侧添加一个新的功能按钮示例。

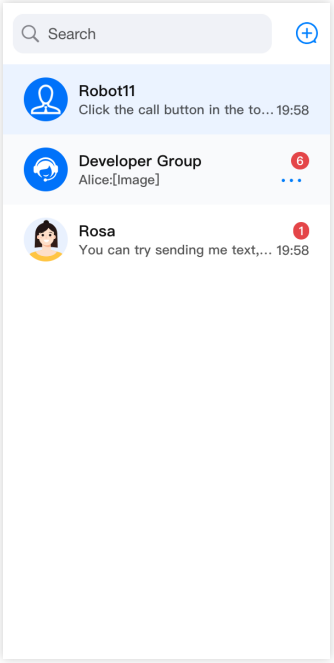
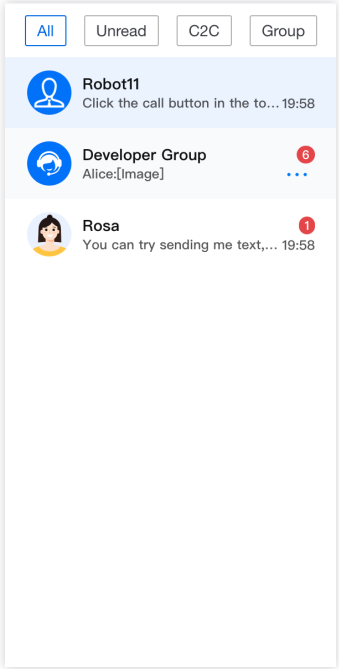
```
import { ConversationList, ConversationListHeader, Icon, IconTypes, IConversationLi

const CustomConversationListHeader = (props: IConversationListHeaderProps) => {
    const CustomIcon = <Icon type={IconTypes.ADD} onClick={() => { console.log('cli
    return (
      <ConversationListHeader {...props} right={CustomIcon} />
    );
};
```

©2013-2025 Tencent Cloud International Pte. Ltd.

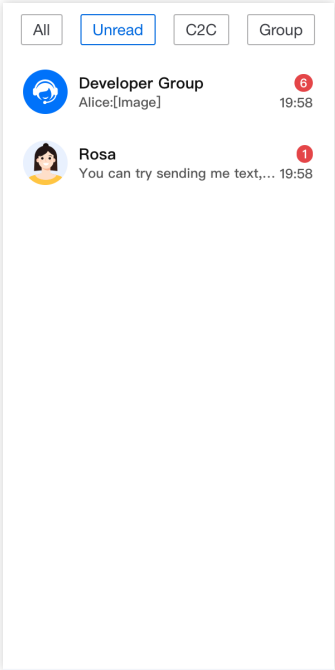
```
<ConversationList Header={CustomConversationListHeader} />
```

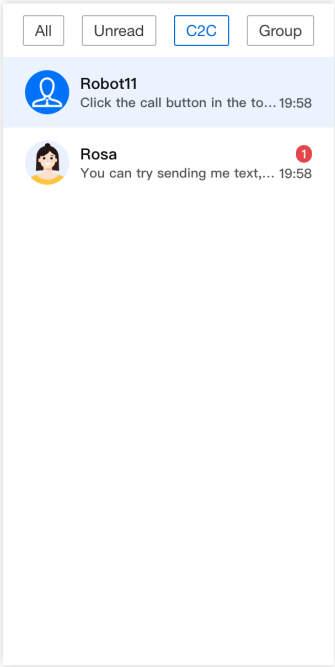
高阶定制

以下是实现一个简易的会话分组功能，按照全部会话、未读会话、单聊会话、群会话四个维度进行区分，通过点击不同分组按钮执行不同的筛选规则。

修改前	修改后
	

©2013-2025 Tencent Cloud International Pte. Ltd.





React

CSS

```
import { useState } from 'react';
import TUIChatEngine, { IConversationModel } from '@tencentcloud/chat-uikit-engine'
import { ConversationList, IConversationListHeaderProps, UIKitProvider } from '@ten

const App = () => {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  const [currentFilter, setCurrentFilter] = useState<string>('all');
  const conversationGroupFilter: Record<string, (conversationList: IConversationMod
    all: (conversationList: IConversationModel[]) => conversationList,
    unread: (conversationList: IConversationModel[]) => conversationList?.filter((i
    c2c: (conversationList: IConversationModel[]) => conversationList?.filter((item
    group: (conversationList: IConversationModel[]) => conversationList?.filter((it
  };

  const CustomConversationListHeader = (props: IConversationListHeaderProps) => {
    return (
      <div className="conversation-group-wrapper">
        <button className={currentFilter === 'all' ? 'btn-active' : 'btn-default'}
        <button className={currentFilter === 'unread' ? 'btn-active' : 'btn-default
        <button className={currentFilter === 'c2c' ? 'btn-active' : 'btn-default'}
        <button className={currentFilter === 'group' ? 'btn-active' : 'btn-default'
      </div>
    );
  };

  return (
    <UIKitProvider>
      <ConversationList
        style={{ maxWidth: '300px', height: '600px' }}
        Header={CustomConversationListHeader}
        filter={conversationGroupFilter[currentFilter]}
      />
    </UIKitProvider>
  );
};
```

```
.conversation-group-wrapper {
  display: flex;
  justify-content: space-around;
  align-items: center;
  margin: 10px;
  font-size: 14px;
  .btn-default{
    display: flex;
    padding: 5px 10px;
    border: 1px solid #b3b3b4;
    color: #3b3d43;
    background-color: transparent;
    border-radius: 2px;
  }
  .btn-active{
    display: flex;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    padding: 5px 10px;
    border: 1px solid #1c66e5;
    color: #1c66e5;
    background-color: transparent;
    border-radius: 2px;
  }
}
```

自定义 List

`ConversationListContent` 负责 ConversationList 主列表部分的渲染工作。
作为包裹层默认渲染 Context 中计算好的当前会话列表显示数据 `filteredAndSortedConversationList` 。

Props

参数名	类型	默认值	说
children	`ReactNode`	-	自在时filt的
empty	`Boolean`	`false`	空<会filt==
loading	`Boolean`	`false`	会<使取
error	`Boolean`	`false`	会<使取
PlaceholderEmptyList	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.NO_CONVERSATIONS} />`	自
PlaceholderLoading	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.LOADING} />`	自

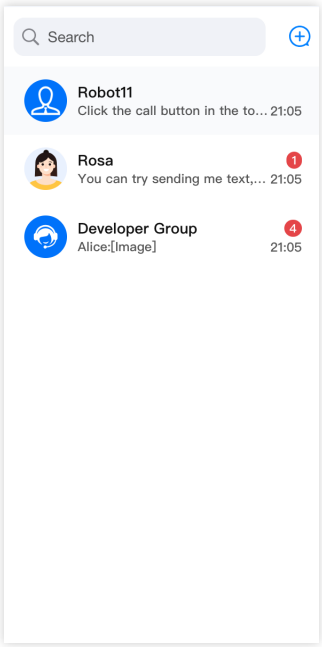
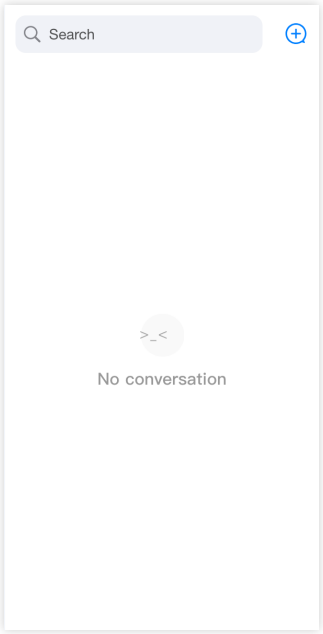
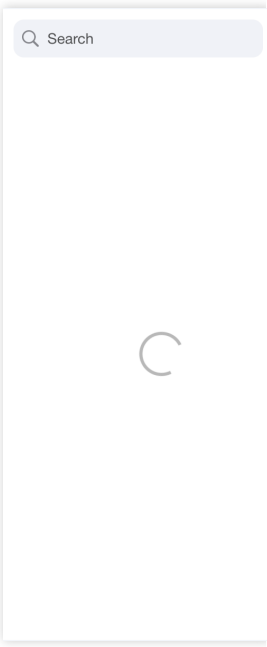
©2013-2025 Tencent Cloud International Pte. Ltd.

PlaceholderLoadError	`ReactNode`	`<PlaceHolder type={PlaceHolderTypes.WRONG} />`	自 素
className	`String`	-	指
style	`React.CSSProperties`	-	指

基础定制

`List` 组件不同状态 UI 效果如下，在 `ConversationList` 内部已处理好每种状态的触发时机。
同时，您可以通过自定义传入 `empty` 、 `loading` 、 `error` 来控制组件状态。

```
import { ConversationList, ConversationListContent, IConversationListContentProps }

const CustomConversationListContent = (props: IConversationListContentProps) => {
    return <ConversationListContent {...props} loading={true} />;
};

<ConversationList  List={CustomConversationListContent} />
```

`default`	`empty={true}`	`loading={true}`
		

自定义 ConversationPreview

详情参见 ConversationPreview 章节。

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<ConversationList ConversationPreview={CustomConversationPreview} />
```

自定义 ConversationActions

详情参见 ConversationActions 章节。

```
<ConversationList ConversationActions={CustomConversationActions} />
```

自定义 ConversationSearch

详情参见 ConversationSearch 章节。

```
<ConversationList ConversationSearch={CustomConversationSearch} />
```

自定义 ConversationCreate

```
<ConversationList ConversationCreate={CustomConversationCreate} />
```

自定义 Avatar

```
<ConversationList Avatar={CustomAvatar} />
```

©2013-2025 Tencent Cloud International Pte. Ltd.

ConversationListContext

最近更新时间：2024-10-09 15:11:35

ConversationListProvider

`ConversationListProvider` 作为整个 `ConversationList` 的包裹层，为 `ConversationList` 内部的所有组件提供上下文。
`ConversationListProvider` 内部对 `ChatEngine` 中 `Conversation` 相关的数据进行监听处理，并传递给整个 `ConversationList UI` 。

Props

`ConversationListProvider` 接收两个可选参数 `filter` 和 `sort` 用于处理要展示的 `conversationList` 数据：

参数名	类型	默认值	说明
filter	(conversationList: IConversationModel[]) => IConversationModel[]	-	用于筛选会话列表的函数。
sort	(conversationList: IConversationModel[]) => IConversationModel[]	-	用于排序会话列表的函数。

基础使用

```
import { ConversationListProvider, IConversationListProps } from '@tencentcloud/cha

function CustomConversationList(props: IConversationListProps) {
  <ConversationListProvider filter={props.filter} sort={props.sort}>
    <div>Custom ConversationList UI</div>
  </ConversationListProvider>;
}
```

useConversationList

`useConversationList` 基于 React useContext Hook 实现。
`ConversationListProvider` 内的所有组件，均可以通过 `useConversationList` 读取和订阅 `context` 中的值。

©2013-2025 Tencent Cloud International Pte. Ltd.

Value

`useConversationList` 提供的 context value 如下：

参数名	类型	说明
conversationList	IConversationModel[]	ChatEngine 中获取的原始的对话列表。
filteredAndSortedConversationList	IConversationModel[]	ConversationList 要渲染的对话列表。这是经过 ConversationListProvider 过滤和排序后的对话列表。
currentConversation	IConversationModel	当前打开的会话。
setCurrentConversation	(conversation: IConversationModel) => void;	设置当前打开的会话。
isLoading	`Boolean`	当前会话列表是否正在获取加载中。
isLoadError	`Boolean`	当前会话列表是否获取错误。

基础使用

```
import { useConversationList } from '@tencentcloud/chat-uikit-react';

const {
    conversationList,
    filteredAndSortedConversationList,
    currentConversation,
    setCurrentConversation
} = useConversationList();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

ConversationPreview

最近更新时间：2024-09-29 16:59:17

`ConversationPreview` 用于对于会话列表中单条会话内容进行预览，组件负责显示会话信息、未读计数以及提供会话操作功能。
借助原子化的信息展示组件，您可以自由地设计和组合出您所期望的 `ConversationPreview` 布局。
同时，您还可以通过 `onConversationSelect` 函数来自定义选中会话时的行为。

基础使用

您可以使用 `ConversationList` 的 `Preview` 属性来自定义会话列表中每个单独会话的预览项，如果没有指定 `Preview` 属性，系统将自动采用 `ConversationPreviewUI` 组件作为默认值。

```
import { ConversationList, ConversationPreviewUI, IConversationPreviewUIProps } fro

const CustomConversationPreview = (props: IConversationPreviewUIProps) => {
  const { Title } = props;
  return (
    <ConversationPreviewUI {...props}>
      {Title}
      <div>Your custom preview UI</div>
    </ConversationPreviewUI>
  );
};

<ConversationList Preview={CustomConversationPreviewUI}></ConversationList>
```

Props

参数名	类型	默认值	说明
conversation*(Required)*	IConversationModel	-	必选参数，标识当前渲染的会话列表项。
isSelected	`Boolean`	`false`	控制

©2013-2025 Tencent Cloud International Pte. Ltd.

			会话列表项 UI 是否处于选中状态
enableActions	`Boolean`	`true`	控制会话操作功能是否显示
actionsConfig	IConversationActionsConfig	-	用于自定义会话操作配置
highlightMatchString	`String`	-	会话列表项 Title 高亮匹配关键词，常用于会话搜索结果高亮。
Title	`String` \\ `JSX.Element`	`ConversationPreviewTitle`	渲染会话列表项标题区域。
LastMessageAbstract	`String` \\	`ConversationPreviewAbstract`	渲染

©2013-2025 Tencent Cloud International Pte. Ltd.

	`JSX.Element`		会话列表项最新消息摘要区域。
LastMessageTimestamp	`String` \\ `JSX.Element`	`ConversationPreviewTimestamp`	渲染会话列表项最新消息时间戳区域。
Unread	`String` \\ `JSX.Element`	`ConversationPreviewUnread`	渲染会话列表项消息未读标识区域。
ConversationActions	`ReactElement`	`ConversationActions`	渲染会话列表项会话操作区域。
Avatar	`ReactElement`	`Avatar`	渲染会话列表项头像区域。
onConversationSelect	(conversation: IConversationModel) => void;	-	指定在选择对话列

©2013-2025 Tencent Cloud International Pte. Ltd.

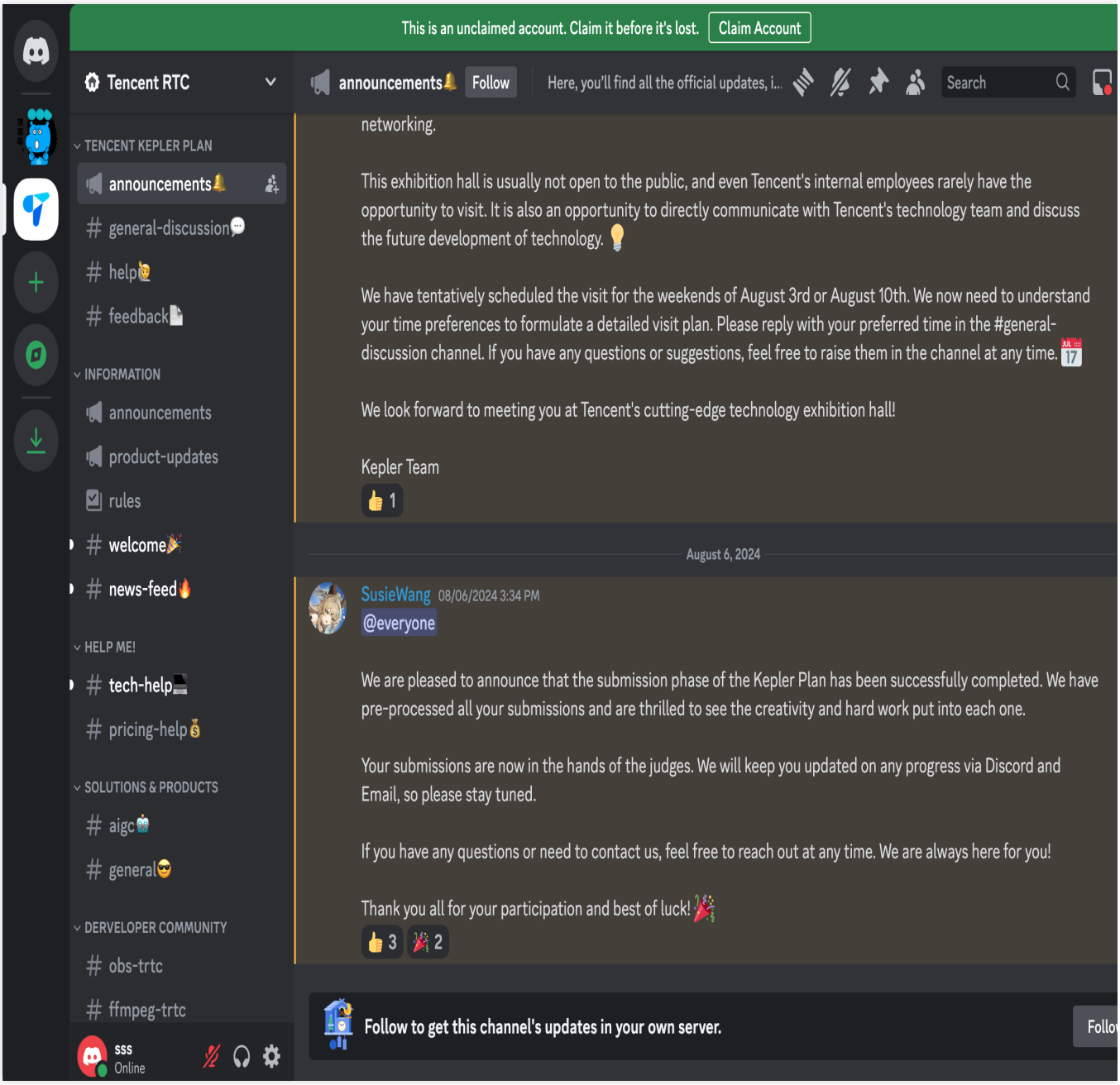
			表中的对话时接收回调的属性。
className	`String`	-	指定根元素类的 CSS 自定义名称。
style	`React.CSSProperties`	-	指定根元素样式的自定义样式。

自定义案例

类 Discord 风格

Discord 是一个流行的聊天应用程序，类似于 Skype 或 Telegram。Discord 中的聊天内容如下图所示:

©2013-2025 Tencent Cloud International Pte. Ltd.



通过对 `ConversationPreview` 布局、功能以及样式的定制，我们可以快速实现类 Discord 效果。

React

CSS

1. 定制 `ConversationListPreview`

2. 切换主题到深色模式

```
import { UIKitProvider, ConversationList, ConversationPreviewUI, IConversationPrevi

const CustomConversationPreview = (props: IConversationPreviewUIProps) => {
  const { Title } = props;
  return (
    <ConversationPreviewUI {...props}>
```

©2013-2025 Tencent Cloud International Pte. Ltd.

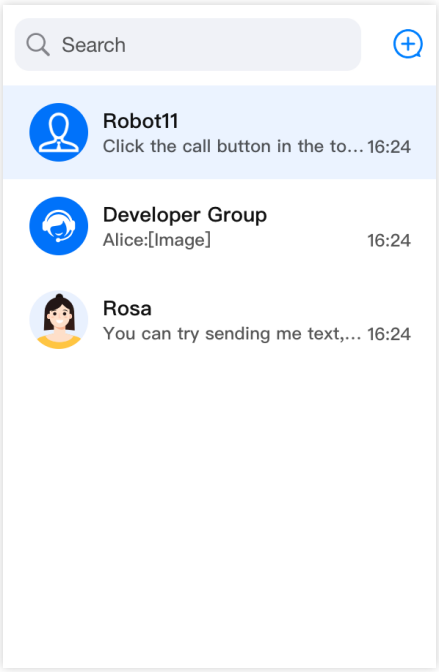
```
      <span> # </span>
      <span>{Title}</span>
    </ConversationPreviewUI>
  );
};

const App = () => {
    <UIKitProvider theme={'dark'}>
      <ConversationList
        style={{ maxWidth: '300px', height: '600px' }}
        Preview={CustomConversationPreviewUI}
       />
      ...
    </UIKitProvider>
}
```

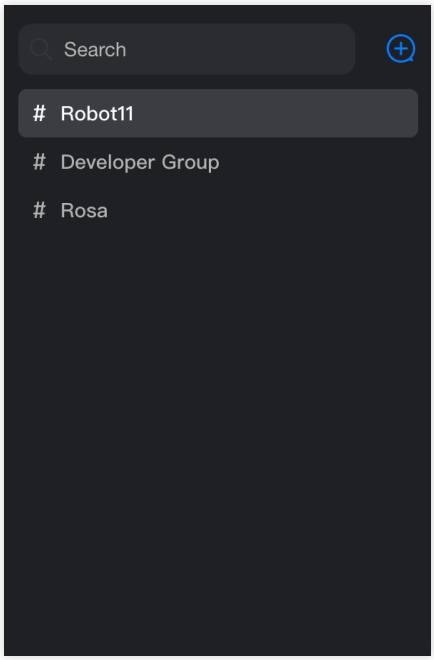
```
.custom-preview-ui {
  height: 34px;
  border-radius: 6px;
  padding: 10px;
  margin: 0 10px;
  .custom-preview-ui__tag {
    margin-right: 10px;
    font-size: 16px;
    color: #b3b3b4;
  }
  .custom-preview-ui__title {
    font-size: 14px;
    color: #b3b3b4;
  }
  &.uikit-conversation-preview--active {
    background-color: #3b3d43;
    .custom-preview-ui__tag {
      color: #ffffff;
    }
    .custom-preview-ui__title {
      .uikit-conversation-preview__title {
        color: #ffffff;
      }
    }
  }
}
```

`ConversationListPreview` 定制后效果如下：

修改前	修改后

©2013-2025 Tencent Cloud International Pte. Ltd.





©2013-2025 Tencent Cloud International Pte. Ltd.

ConversationSearch

最近更新时间：2025-05-27 18:09:46

`ConversationSearch` is mainly used to search the conversation list, consisting of `ConversationSearchInput` and `ConversationSearchResult`.

Basic Usage

Used in ConversationList

Used independently

```
import { UIKitProvider, ConversationList, ConversationSearch, IConversationSearchPr
import { IConversationModel } from '@tencentcloud/chat-uikit-engine';

const CustomConversationSearch = (props: IConversationSearchProps) => {
  const onSelectResult = (conversation: IConversationModel) => {
    console.warn(`Select Search Result: ${conversation.conversationID}`);
  };
  return (
    <ConversationSearch {...props} onSelectResult={onSelectResult} />
  );
};

const App = () => {
  return (
    <UIKitProvider>
      <ConversationList
        style={{ maxWidth: '300px', height: '600px' }}
        ConversationSearch={CustomConversationSearch}
       />
     </UIKitProvider>
  );
};
```

```
import { UIKitProvider, ConversationListProvider, useConversationList, Conversation

const CustomConversationSearch = () => {
  const { conversationList } = useConversationList();
  return (
    <ConversationSearch
      style={{ maxWidth: '300px', maxHeight: '500px' }}
      conversationList={conversationList}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    />
  );
};

const App = () => {
  return (
    <UIKitProvider>
      <ConversationListProvider>
        <CustomConversationSearch />
      </ConversationListProvider>
    </UIKitProvider>
  );
};
```

Props

Parameter Name	Type	Default Value
conversationList ***(Required)***	IConversationModel	-
Avatar	`ReactElement`	`Avatar`
ResultPreview	`ReactElement`	`ConversationPreview`
ConversationSearchInput	`ReactElement`	`ConversationSearchInput`
ConversationSearchResult	`ReactElement`	`ConversationSearchResult`
searchFn	(searchValue: string, conversationList: IConversationModel[]) => IConversationModel[]	defaultSearchFn

©2013-2025 Tencent Cloud International Pte. Ltd.

onSearchChange	(searchValue: string) => void	-
onFocus	(event: React.FocusEvent<HTMLInputElement>) => void	-
onBlur	(event: React.FocusEvent<HTMLInputElement>) => void	-
onSelectResult	(conversation: IConversationModel) => void	-
visible	`Boolean`	`true`
className	`String`	-
style	`React.CSSProperties`	-

Search rules

Default search rules

When `searchFn` is not provided, `searchFn` will use the default value `defaultSearchFn` as the search rule.

`defaultSearchFn` defaults to searching and matching the conversation title in the conversation list, case insensitive. `defaultSearchFn` Definition is as follows:

```
function defaultSearchFn(searchValue: string, conversationList: IConversationModel[
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  if (!searchValue || conversationList?.length === 0) return [];
  return conversationList.filter(item => item.getShowName().toLocaleLowerCase().inc
}
```

Custom Search Rules

You can implement different search rules by passing searchFn props. Here is an example of searching based on conversationID information in the conversation list:

```
function customSearchFn(searchValue: string, conversationList: IConversationModel[]
  if (!searchValue || conversationList?.length === 0) return [];
  return conversationList.filter(item => item.conversationID.includes(searchValue))
}

<ConversationSearch {...props} searchFn={customSearchFn} />
```

Event Handling

`ConversationSearch` provides a rich set of props to help you customize responses to various search states.

`onSearchChange` is called when the search input changes.

`onFocus` is called when the search input gains focus.

`onBlur` is called when the search input gains focus.

`onSelectResult` is called when a search result is selected.

Custom Component

ConversationSearchInput

Parameter Name	Type	Default Value	Indication
placeholder	`String`	-	Input box placeholder text.
clearable	`Boolean`	`false`	Whether the input content can be cleared.
prefix	`ReactNode`	<Icon type={IconTypes.SEARCH}	Input box prefix content.

©2013-2025 Tencent Cloud International Pte. Ltd.

		height={16} width={16} />	
value	`String`	-	Value of the input box.
onChange	(event: React.ChangeEvent<HTMLInputElement>) => void	-	Specify a function to be called when the search input changes.
onFocus	(event: React.FocusEvent<HTMLInputElement>) => void	-	Specify a function to be called when the search input gains focus.
onBlur	(event: React.FocusEvent<HTMLInputElement>) => void	-	Specify a function to be called when the search input loses focus.
className	`String`	-	Custom Class Name.

Below is a new definition of a `placeholder` and `prefix` example:

```
const CustomConversationSearchInput = (props: IConversationSearchInputProps) => {
  return <ConversationSearchInput {...props} placeholder="this is a custom placehol
};

const CustomConversationSearch = (props: IConversationSearchProps) => {
  return <ConversationSearch {...props} ConversationSearchInput={CustomConversation
};
```

before	**after**

ConversationSearchResult

Parameter Name	Type	Default Value	Description
visible	`Boolean`	false	Specify whethe

©2013-2025 Tencent Cloud International Pte. Ltd.

			display the sear results compon
searchResult	IConversationModel[]	-	Search results.
searchValue	`String`	-	Search input va
ResultPreview	`ReactElement`	`ConversationPreview`	From the Definit `ResultPrev` component.
Avatar	`ReactElement`	`Avatar`	From the Definit `Avatar` component.
PlaceholderNoResult	`ReactNode`	<PlaceHolder type={PlaceHolderTypes.NO_RESULTS} searchString={searchValue} />	Placeholder component fron Definition for no results.
onSelectResult	(conversation: IConversationModel) => void	-	Callback functic when `ResultPrev` is clicked.
className	`String`	-	Custom Class Name.
style	`React.CSSProperties`	-	Custom Definiti CSS style.

Here is an example of a new no search results interface `PlaceholderNoResult` :

```
const CustomConversationSearchResult = (props: IConversationSearchResultProps) => {
  const CustomPlaceholder = (
    <div style={{ display: 'flex', flex: '1', justifyContent: 'center', alignItems:
      <img src="https://web.sdk.qcloud.com/im/assets/emoji/emoji_76@2x.png" alt="te
    </div>
  );
  return <ConversationSearchResult {...props} PlaceholderNoResult={CustomPlaceholde
};

const CustomConversationSearch = (props: IConversationSearchProps) => {
  return <ConversationSearch {...props} ConversationSearchResult={CustomConversatio
};
```

©2013-2025 Tencent Cloud International Pte. Ltd.

before	after

ResultPreview

`ResultPreview` parameters are used to display a single search result in `ConversationSearchResult`. When no value is passed, `ResultPreview` will default to displaying the `ConversationPreview` component as the single search result component.

You can go to ConversationPreview to view more Custom Tutorials.

©2013-2025 Tencent Cloud International Pte. Ltd.

ConversationActions

最近更新时间：2024-09-29 16:58:34

`ConversationActions` 组件负责对于单条会话进行操作，默认支持会话删除、会话置顶/取消置顶、会话免打扰/取消免打扰功能。

基础使用

当在 `ConversationList` 使用 `ConversationActions` 时，您可以通过 `ConversationList` 顶层 `actionsConfig` 参数直接进行自定义， `actionsConfig` 支持对于默认会话操作功能开关、事件响应、新增自定义操作项、基础 UI 定制等。
如您需要更高阶的定制，您可以通过 `ConversationActions` 自定义新的组件。

使用 actionsConfig 基础定制

```
import { UIKitProvider, ConversationList } from "@tencentcloud/chat-uikit-react";

const App = () => {
  return (
    <UIKitProvider>
      <ConversationList
        actionsConfig={{
          enablePin: false,
          onConversationDelete: (conversation: IConversationModel) => { console.log
          customConversationActions: {
            'custom-actions-1': {
               label: 'custom-actions',
               onClick: (conversation: IConversationModel) => { console.log(convers
             },
          },
      }}/>
    </UIKitProvider>
  );
}
```

自定义 ConversationActions 组件

```
import { UIKitProvider, ConversationList, ConversationActions, IConversationActions

const CustomConversationActions = (props: IConversationActionsProps) => {
  return <ConversationActions {...props} enableDelete={false} />;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
};

const App = () => {
  return (
    <UIKitProvider>
      <ConversationList
        style={{ maxWidth: '300px', height: '600px' }}
        ConversationActions={CustomConversationActions}
      />
    </UIKitProvider>
  );
}
```

Props

`ConversationActions` 组件的接口类型 `IConversationActionsProps` 基于 `IConversationActionsConfig` 接口进行扩展。

IConversationActionsProps

参数名	类型	默认值	说明
conversation*(Required)*	IConversationModel	-	必选参数，表示当前渲染会话操作对应的会话。
className	`String`	-	自定义根元素的类名。
style	`React.CSSProperties`	-	自定义根元素的样式。

IConversationActionsConfig

参数名	类型	默认值	说明
enablePin	`Boolean`	`true`	是否显示置顶功能按钮。
enableMute	`Boolean`	`true`	是否显示免打扰功能按钮。
enableDelete	`Boolean`	`true`	是否显示删除功

©2013-2025 Tencent Cloud International Pte. Ltd.

			能按钮。
onConversationPin	(conversation: IConversationModel, e?: React.MouseEvent) => void	-	自定义置顶/取消置顶会话的行为。
onConversationMute	(conversation: IConversationModel, e?: React.MouseEvent) => void	-	自定义免打扰/取消免打扰会话的行为。
onConversationDelete	(conversation: IConversationModel, e?: React.MouseEvent) => void	-	自定义删除会话的行为。
customConversationActions	Record<string, IConversationActionItem>	-	自定义会话操作项。
PopupIcon	`ReactElement`	-	自定义动作弹窗的图标。
PopupElements	`ReactElement[]`	-	自定义动作弹窗的内容。
onClick	(e: React.MouseEvent, key?: string, conversation?: IConversationModel) => void	-	点击动作项时的回调函数。

自定义组件

基础功能开关

通过设置 `enablePin` , `enableDelete` 和 `enableMute` 参数，你可以灵活地控制 `ConversationActions` 中的会话置顶、会话免打扰和会话删除的显示。

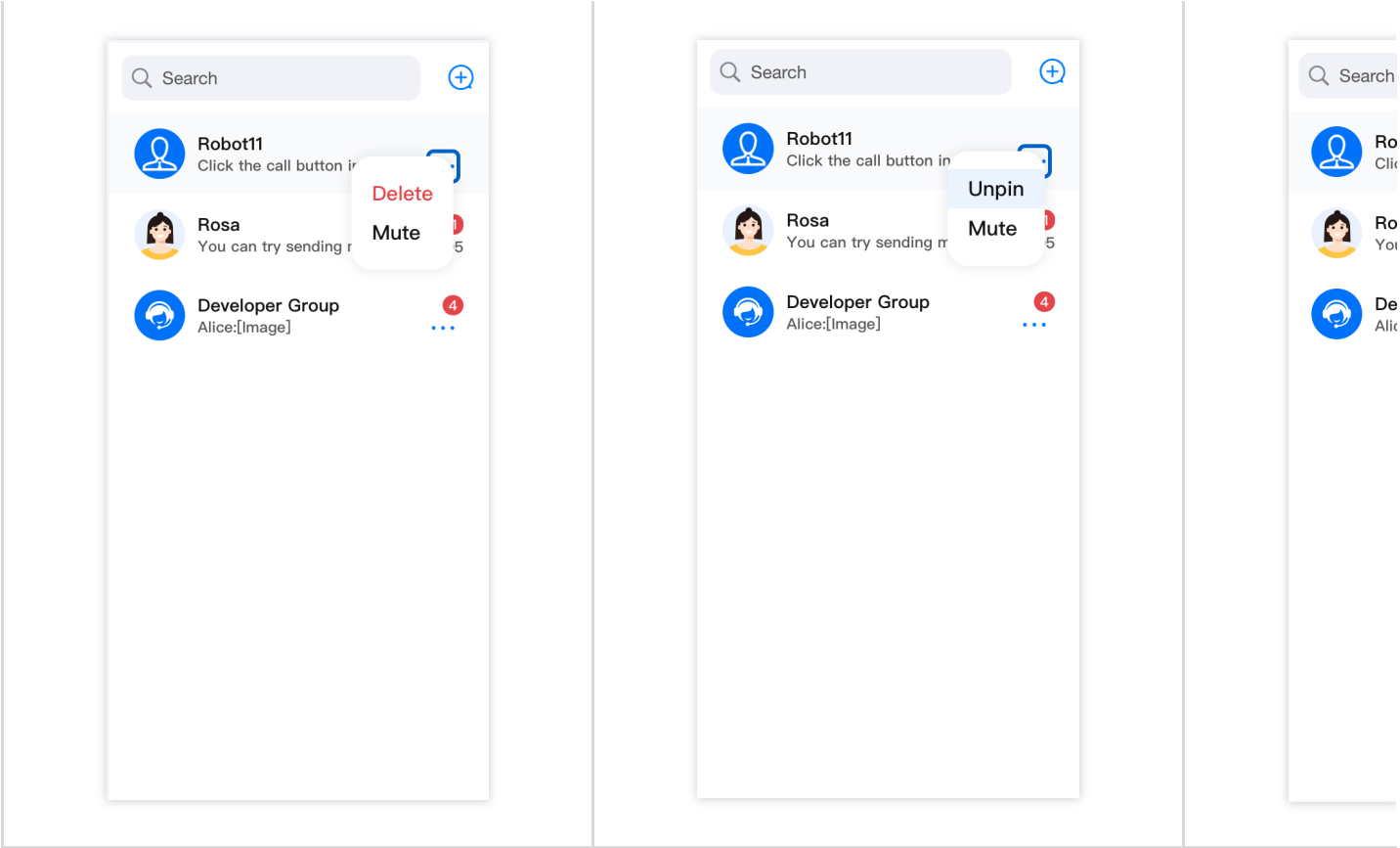
```
<ConversationActions enablePin={false} />
```

```
<ConversationActions enableDelete={false} />
```

```
<ConversationActions enableMute={false} />
```

`enablePin={false}`	`enableDelete={false}`	`enableMute=`

©2013-2025 Tencent Cloud International Pte. Ltd.



事件响应

`ConversationActions` 组件默认支持会话删除、会话置顶/取消置顶、会话免打扰/取消免打扰功能，如已有功能的事件响应不符合您的需求，您可以自定义事件响应处理函数并进行覆盖; 除了支持对于功能事件响应的自定义外，您还可以通过 onClick 获取基础点击事件响应。

```
import { ConversationList, ConversationActions, IConversationActionsProps, Toast }
import { IConversationModel } from '@tencentcloud/chat-uikit-engine';

const CustomConversationActions = (props: IConversationActionsProps) => {
  return (
    <ConversationActions
      {...props}
      onConversationDelete={(conversation: IConversationModel) => {
        conversation.deleteConversation().then(() => {
          Toast({ text: 'delete conversation successfully!', type: 'info' });
        }).catch(() => {
          Toast({ text: 'delete conversation failed!', type: 'error' });
        });
      }}
    />
  );
};
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
<ConversationList ConversationActions={CustomConversationActions} />
```

customConversationActions

`customConversationActions` 用于在 ConversationActions 上新增自定义操作项列表。

Record<string, IConversationActionItem>

IConversationActionItem

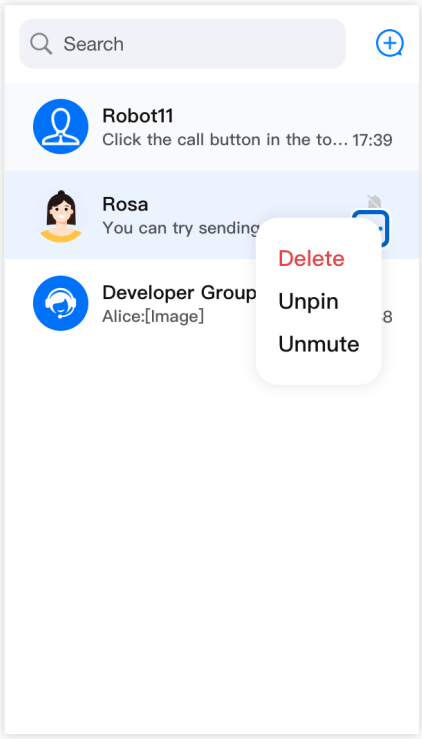
参数名	类型	默认值	说明
enable	`Boolean`	`true`	是否启用自定义操作项。
label	`String`	-	自定义操作项的展示内容。
onClick	(conversation: IConversationModel, e?: React.MouseEvent) => void	-	点击自定义操作项时的回调函数。

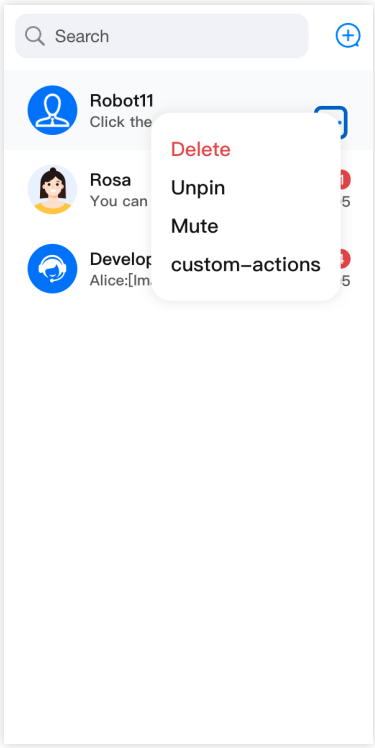
以下是使用 `customConversationActions` 新增自定义操作项的示例：

```
import { ConversationList, ConversationActions, IConversationActionsProps } from '@

const CustomConversationActions = (props: IConversationActionsProps) => {
  return (
    <ConversationActions
      {...props}
      customConversationActions={{
        'custom-actions-1': {
           label: 'custom-actions',
           onClick: (conversation: IConversationModel) => { console.log(conversatio
         },
      }}
    />
  );
};

<ConversationList ConversationActions={CustomConversationActions} />
```

修改前	修改后

©2013-2025 Tencent Cloud International Pte. Ltd.



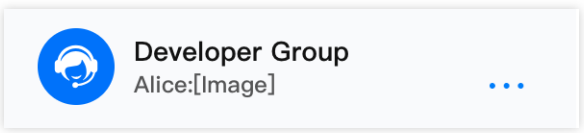
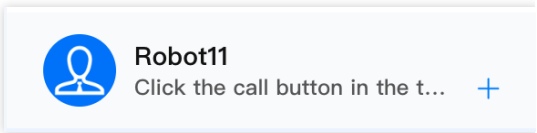


UI 界面定制

您可以通过 `PopupIcon` 参数定制唤醒弹出按钮样式，通过 `PopupElements` 定制弹出内容。

以下是在默认的 `ConversationActions` 组件的基础上进行二次开发，为其定制新的唤起按钮样式代码示例：

```
import { ConversationList, ConversationActions, IConversationActionsProps, Icon, Ic

const CustomConversationActions = (props: IConversationActionsProps) => {
  const customIcon = <Icon type={IconTypes.ADD} width={16} height={16} />;
  return (
    <ConversationActions {...props} PopupIcon={customIcon} />
  );
};

<ConversationList ConversationActions={CustomConversationActions} />
```

修改前	修改后
	

©2013-2025 Tencent Cloud International Pte. Ltd.

©2013-2025 Tencent Cloud International Pte. Ltd.