实时音视频

视频通话（含 UI）



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
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。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

- 视频通话（含 UI）
  - 组件介绍（TUICallKit）
  - 开通服务（TUICallKit）
  - 跑通 Demo（TUICallKit）
    - Android
    - iOS
    - Web
    - Flutter
    - 微信小程序
    - uni-app（小程序）
    - uni-app（客户端）
    - uni-app（uts 插件）
    - React Native
  - 快速接入（TUICallKit）
    - Android
    - iOS
    - Web&H5 (Vue2/Vue3)
    - Web&H5 (React)
    - uni-app（客户端）
    - uni-app（小程序）
    - 微信小程序
    - Flutter
    - React Native
    - uni-app (uts 插件)
  - 界面定制（TUICallKit）
    - Android
    - iOS
    - Web
    - 微信小程序
    - Flutter
    - uni-app（小程序）
    - uni-app（uts 插件）
      - 替换页面图标
      - 高级定制能力
  - 离线唤醒（TUICallKit）
    - Flutter
      - Notification
      - VoIP
    - Android
      - Notification
      - FCM data message
    - iOS
      - Notification
      - VoIP
    - uni-app (uts 插件)
    - uni-app (客户端)
  - 虚拟背景（TUICallKit）
    - iOS
    - Android

版权所有：腾讯云计算（北京）有限责任公司

- Web
- uniapp（客户端）
- uniapp（小程序）
- 微信小程序
- Flutter

会话聊天（TUICallKit）
- Android
- iOS
- Flutter
- Vue
- React

悬浮窗（TUICallKit）
- Android&iOS&Flutter
- Web&H5&uni-app (小程序)
- uni-app（客户端）
- uni-app (uts 插件)
- 原生小程序

云端录制（TUICallKit）

USB 摄像头（TUICallKit）

更多特性（TUICallKit）
- 大屏设备
- 设置昵称、头像（全平台）
- 设置分辨率、填充模式（Web）
- 体积增量
- 多人通话
  - Android&iOS&Flutter
  - Web&H5&小程序
  - uni-app (uts 插件)
  - uni-app（客户端）
- 美颜特效
  - Android
  - iOS
  - Flutter
  - Web&小程序
- 自定义铃声
  - Android
  - iOS
  - Web&H5&小程序
  - uni-app（客户端）
  - Flutter
- 监听通话状态
  - Android&iOS&Flutter
  - Web&H5
  - uni-app（客户端）
  - uni-app (uts 插件)
  - uniapp（小程序）
  - 微信小程序
- 语言设置
  - Android
  - iOS
  - Web&H5
  - Flutter

版权所有：腾讯云计算（北京）有限责任公司

uni-app（客户端）
体积优化
uni-app (小程序)
服务端 API（TUICallKit）
通话记录简介
通话记录看板
账号系统
账号管理
登录鉴权
生成 UserSig
资料管理
REST API
REST API 简介
REST API 列表
通话记录
通过 CallId 获取记录
通过条件获取记录
获取实时通话状态
第三方回调
第三方回调简介
回调命令列表
回调配置
查询回调配置
设置回调配置
删除回调配置
通话状态回调
开始通话后回调
通话结束后回调
通话成员状态变更后回调
旧版文档
服务端回调
通话状态回调
通话事件回调
回调配置
回调配置 API 列表
创建回调配置
查询回调配置
更新回调配置
删除回调配置
REST API
通话记录
REST API 简介
通过 callId 获取记录
通过条件获取记录
结束通话
客户端 API（TUICallKit）
Android
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义

iOS
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义
Web
API 概览
TUICallKit
TUICallEngine
TUICallEvent
小程序
API 概览
TUICallKit
TUICallEngine
TUICallEvent
uni-app（客户端）
API 概览
TUICallKit
TUICallEngine
TUICallEvent
Flutter
API 概览
TUICallKit
TUICallEngine
TUICallObserver
类型定义
React Native
API 概览
TUICallKit
TUICallEngine
TUICallEvent
关键类型
错误码表（TUICallKit）
发布日志（TUICallKit）
uni-app（小程序）
Web
Android&iOS
uni-app (客户端)
Flutter
微信小程序
常见问题（TUICallKit）
全平台
Web
Android
iOS
uni-app（客户端）
uni-app（小程序）
微信小程序
Flutter

版权所有：腾讯云计算（北京）有限责任公司

视频通话（含 UI）
组件介绍（TUICallKit）

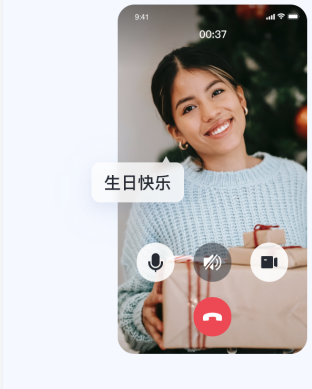
最近更新时间：2025-06-10 15:36:11

组件介绍

TUICallKit是基于通话场景深度定制的一款产品，支持9人同时在线通话，提供微信视频聊天同款 UI 组件，仅需3步，最快1天上线，可节省90%开发时间。支持小程序、 iOS、Android、Web、uni-app、Flutter 等平台全球互通。提供视频通话场景下常见的双人音视频通话、群组音视频通话、中途呼叫第三方、AI 通话降噪、通话卡顿优化等功能，支持标清、高清、超高清等多种画质，配合低延时高质量的音视频服务，帮您快速搭建稳定可靠的音视频通话应用。



应用场景

应用场景	场景展示	功能说明
社交娱乐	 	快速搭建社交娱乐类应用，多端互通，畅享实时通话互动体验。 • 1v1 音视频通话 • 群组通话 • 自定义铃声 • 自定义头像、昵称 • 通话悬浮窗

远程医疗		提供稳定可靠的通话功能，快速实现1v1心理咨询、视频问诊等场景。 • 低延迟实时互动 • 云端录制 • AI降噪 • 虚拟背景
在线客服		快速集成在线客服功能，为您的客户提供更好的沟通服务。 • 1V1音视频通话 • 离线推送 • 通话悬浮窗 • 虚拟背景
在线教育		通过音视频通话连接老师和学生，轻松进行课堂互动。 • 群组通话 • AI 降噪 • 弱网卡顿优化

功能概览

功能说明：
TUICallKit 结合即时通信、实时音视频能力，为您提供含 UI 的低代码组件，您只需要编写几行代码就可以为您的 App 添加音视频通话功能。更多功能与计费说明参见 开通服务 。

基础功能

	功能描述
1v1语音、视频通话	一对一进行语音/视频聊天
群组通话（中途邀请他人、中途加入）	多人同时进行语音/视频通话，并在通话中邀请其他成员加入
自定义铃声	用户可根据自己的需要/喜好设置应用铃声和离线推送铃声

版权所有：腾讯云计算（北京）有限责任公司

设置头像、昵称	用户可以根据自己的需要/喜好设置自己的头像和昵称
通话悬浮窗	通话界面收起后，可以悬浮窗样式悬浮在应用界面之上
小程序通话加速	支持小程序端通话与小程序进房通话加速优化，保障通话效果。

高级功能

	功能描述
离线推送	应用未打开时，也能通过消息推送让用户收到音视频通话的响铃呼叫。
虚拟背景	用户在视频通话时，可设置模糊背景或图片背景，隐藏真实通话环境，保护隐私，让通话更有趣。支持 iOS、Android、Flutter、uin-app 和 Web 端。
云端录制	方便用户对重要通话内容进行存档、审核等。
AI降噪	借助 AI 能力有效过滤背景噪声，还原通话效果，改善用户听感。
同平台多端登录通话	支持同一平台，如多个 iOS 设备登录，来电后任一设备接通，自动停止其他设备的接入请求。
弱网卡顿优化	针对户外等弱网环境下优化卡顿率，秒开速度更快。
通话记录（看板+API）	提供通话数据统计及通话记录可视化看板。服务包含通话次数、通话人数、最大并发通话数等统计数据，以及通话发起时间、通话参与人数、通话持续时间等单条通话记录信息。

支持平台

平台	Android	iOS	Web&H5 (Vue2/Vue3)	Web&H5 (React)	uni-app (小程序)	Flutter	uni-app (客户端)	微信小程序
是否支持	✓	✓	✓	✓	✓	✓	✓	✓
支持语言/框架	• Kotlin • Java	• Swift • Objective-C	• Vue3 • Vue2	• React	• Vue3 • Vue2	Dart	• Vue3 • Vue2	-

在线体验

平台	Web	Android	iOS	微信小程序
在线体验	Available on the Web	GET IT ON Google Play	Download on the App Store	
demo 集成	Github: web	Github: Andorid	Github: iOS	Github: 微信小程序

TRTC 技术交流群

加入 腾讯实时音视频 TRTC 技术交流群，您将获得：

- 可靠的技术支持
- 详细的产品信息
- 紧密的行业交流

开通服务（TUICallKit）

最近更新时间：2025-07-03 12:02:32

本文介绍如何 免费开通体验版 、开通正式版 和 续费正式版 服务，您可以根据 音视频通话能力版本计费说明 选择您需要版本，并参考下述指引免费开通和购买开通正式版音视频通话能力。

说明：

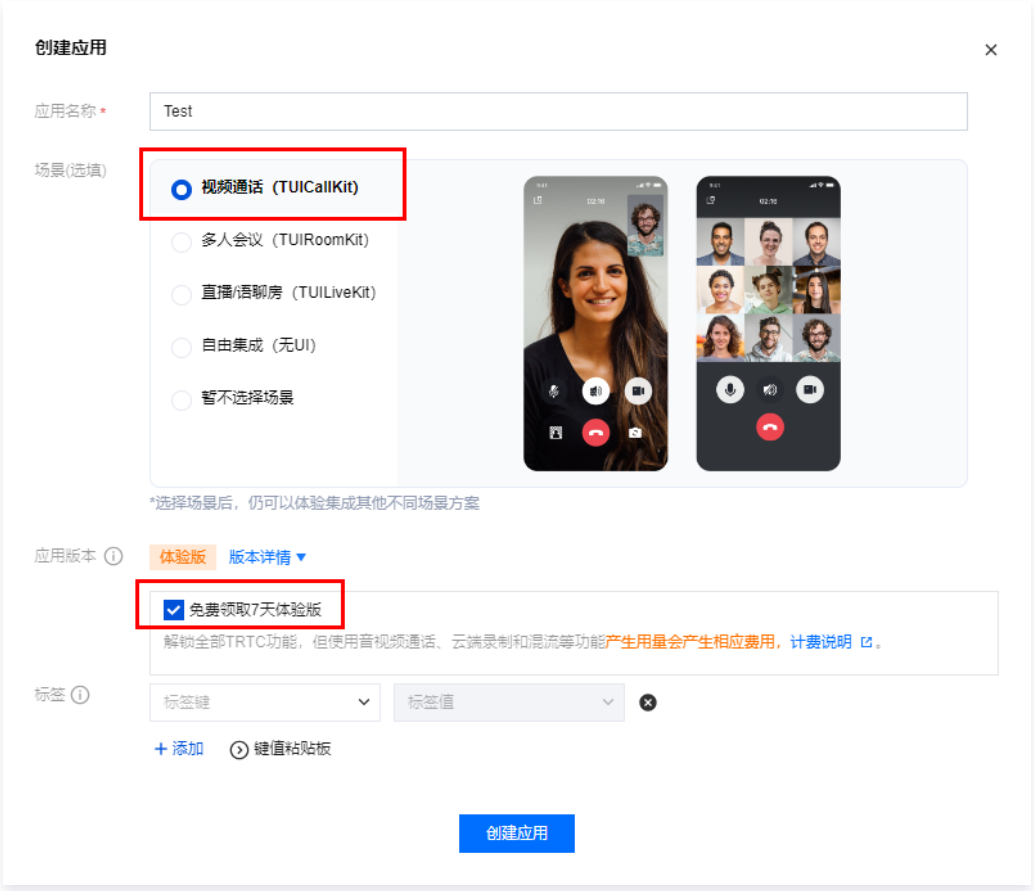
衷心感谢您对本产品的信任与长期支持。为了满足不断变化的市场需求，持续提升用户体验、优化产品性能，我们对产品进行了升级，诚邀您升级至新版本，以体验更好的效果及功能，具体说明如下：

- 全平台新增 calls 接口，支持发起单人通话或多人通话，更加灵活的通话成员管理，支持更强大的 REST API，欢迎使用。
- 全平台新增 join 接口，配合 calls 接口、支持加入已有通话。
- 对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。

开通体验版

TUICallKit 是基于腾讯云 即时通信 IM 和 实时音视频 TRTC 两项付费 PaaS 服务构建出的音视频通信组件。为了您能更好地体验音视频通话功能，我们免费为每个 SDKAppID 提供了音视频通话能力7天体验版（体验版不额外赠送通话时长），每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。

1. 登录 实时音视频 TRTC 控制台，单击进入左侧应用管理页面，单击**创建应用**。在弹出的对话框中输入您的应用名称，并单击**创建应用**。



注意：

若您创建时勾选**免费领取7天体验版**，即可快速创建应用时并领取体验版。**领取体验版后，将解锁全部音视频通话功能。**

2. 单击刚创建的应用，进入应用管理页面，并在页面的右下角找到**含 UI 低代码场景方案 > 视频通话（TUICallKit）**，单击**领取体验**即可**开通 TUICallKit 的 7 天免费试用服务**。领取完成后，可以看到体验版的基本信息，包括服务状态、版本信息和功能详情、到期时间。这里的 `SDKAppID` 、`SDKSecretKey` 会在后续步骤中使用到。



> **注意：**
> - 如果需要**正式应用上线**，可以单击 **场景套餐订阅** 即可进入购买页面。
> - IM 音视频通话能力针对不同的业务需求提供了差异化的付费版本供您选择，您可以在 **IM 购买页** 了解包含功能并选购您适合的版本。

3. 若您当前 SDKAppID 领取的体验版已到期，您可单击**版本详情**，在版本详情弹窗中单击**体验续期**直接使用第二次体验机会，此时体验版有效期将延长7天。

> **注意：**
> 通过**体验续期**延长体验版有效期，将消耗1次免费体验机会。

开通正式版

音视频通话能力为 TRTC 组件能力之一，仅当 IM 应用为专业版或旗舰版时才可购买正式版音视频通话能力。您可参见 **音视频通话能力版本计费说明** 确认所需要使用的版本，前往 **购买正式版** 的音视频通话能力。

1. 登录 **实时音视频购买页** 。
2. 根据实际业务需求选择配置。接入方案选择**含 UI 低代码快速开发**，以及选择需要购买音视频通话能力的 SDKAppID，购买后无法修改。



3. 套餐类型选择**音视频通话**，并在套餐版本中选择所需版本即可，音视频通话套餐包能力详情可参见 音视频通话能力版本计费说明 。



4. 购买完成后，您可回到 实时音视频 TRTC 控制台，在**含 UI 低代码集成接入**功能区查看当前版本信息。开通完成后即可参见 集成指引 进行集成。



功能与计费说明

TUICallKit 由实时音视频 TRTC 和即时通信 IM 提供底层技术支持，因此 TUICallKit 费用包含购买即时通信 IM 的费用及开通实时音视频 TRTC 包月套餐包费用两部分，其中购买 TRTC 包月套餐包会赠送等价格实时音视频 TRTC 音视频时长补充包，赠送时长包按月赠送，仅在当月生效。**您需要购买指定套餐后方可使用 TUICallKit，**其中免费体验版的开通方式可参见 开通 TUICallKit 体验版 。

下表为 TUICallKit 推荐套餐对应的功能和价格，您也可以自由组合 TRTC 包月套餐和 IM 版本以获得对应版本的 TUICallKit。

- TRTC 基础版配合 IM 专业版/旗舰版可使用 TUICallKit 的1v1通话版；
- TRTC 尊享版/尊享版 Plus /旗舰版/旗舰版 Plus 配合 IM 专业版/旗舰版/企业版可使用 TUICallKit 的群组通话版。

套餐包分为**含录制/转码/ AI 套餐**和**不含录制/转码/ AI 套餐**两种类型，请按需选购。

含录制/转码/ AI 套餐（New）

推荐套餐	群组通话版 38万分钟	群组通话版 140万分钟
价格	4374元/月 立即选购	10999元/月 立即选购
包含套餐版本	TRTC 尊享版 Plus +	TRTC 旗舰版 Plus +

		IM 专业版	IM 旗舰版
赠送资源（超出后付费）	音视频时长	380,000分钟/月	1,400,000分钟/月
	日活跃用户数（DAU）	10,000 DAU	10,000 DAU
	峰值群组数	100000个/月	100000个/月
	录制转码时长（new）	260,000分钟/月	600,000分钟/月
	AI 智能识别时长（new）	10,000分钟/月	20,000分钟/月
TUICallKit 专属功能	1v1 视频、音频通话	✓	✓
	微信同款 UI 设计	✓	✓
	通话状态展示	✓	✓
	离线推送（应用未打开时，也能通过消息推送让用户收到音视频通话的响铃呼叫）	✓	✓
	通话悬浮窗（通话界面收起后，可以悬浮窗样式悬浮在应用界面之上）	✓	✓
	自定义呼叫铃声	✓	✓
	呼叫/接听/拒绝/挂断	✓	✓
	视频通话切换语音通话	✓	✓
	群组通话（9人及以下）	✓	✓
	中途呼叫/加入三方通话	✓	✓
	通话记录（看板 + API）	✓	✓
	小程序通话加速（支持小程序端通话与小程序进房通话加速优化，保障通话效果）	✓	✓
	虚拟背景（进行视频通话时，可设置模糊背景或替换背景图片，支持 iOS、Android、Flutter、uin-app 和 Web端）	-	✓
	多端登录通话（任一平台接通，自动停止其他端的接入请求）	✓	✓
	同平台多端登录通话（支持同一平台，如多个 iOS 设备登录，来电后任一设备接通，自动停止其他设备的接入请求）	✓	✓
	AI 降噪（借助 AI 能力有效过滤背景噪声，还原通话效果）	✓	✓
	弱网通话卡顿优化（针对户外等弱网环境下优化卡顿率，秒开速度更快）	✓	✓

即时通信 IM 功能	详见 IM 专业版功能	详见 IM 旗舰版功能
实时音视频 TRTC 功能	详见 TRTC 尊享版 Plus 功能	详见 TRTC 旗舰版 Plus 功能
超出赠送资源部分后付费	• 音视频时长后付费价格详见 音视频时长计费说明 • DAU 与峰值群组数后付费价格详见 IM 计费说明 • 录制转码时长后付费价格详见 云端录制与投递计费说明 和 云端混流转码计费说明 • AI 智能识别时长后付费价格详见 AI 智能识别计费说明	
支持平台	iOS、Android、Web、**微信小程序（仅体验版和群组通话版支持）**、uni-app、Flutter	

不含录制/转码/ AI 套餐

推荐套餐		体验版	1V1通话版	群组通话版 38万分钟	群组通话版 140万分钟
价格		0元/7天	2124元/月 立即选购	4874元/月 立即选购	9249元/月 立即选购
包含套餐版本		TRTC 体验版 + IM 任意版本	TRTC 基础版 + IM 专业版	TRTC 尊享版 + IM 旗舰版	TRTC 旗舰版 + IM 旗舰版
赠送资源 （超出后付费）	音视频时长	–	110,000分钟/月	380,000分钟/月	1,400,000分钟/月
	日活跃用户数（DAU）	根据 IM 版本而定	10,000 DAU	10,000 DAU	10,000 DAU
	峰值群组数	根据 IM 版本而定	100000个/月	100000个/月	100000个/月
TUICallKit 专属功能	1v1 视频、音频通话	✓	✓	✓	✓
	微信同款 UI 设计	✓	✓	✓	✓
	通话状态展示	✓	✓	✓	✓
	离线推送 （应用未打开时，也能通过消息推送让用户收到音视频通话的响铃呼叫）	✓	✓	✓	✓
	通话悬浮窗 （通话界面收起后，可以悬浮窗样式悬浮在应用界面之上）	✓	✓	✓	✓
	自定义呼叫铃声	✓	✓	✓	✓
	呼叫/接听/拒绝/挂断	✓	✓	✓	✓
	视频通话切换语音通话	✓	✓	✓	✓
	群组通话	✓	–	✓	✓
	中途呼叫/加入三方通话	✓	–	✓	✓
	通话记录（看板 + API）	✓	–	✓	✓
	小程序通话加速	✓	–	✓	✓

	（支持小程序端通话与小程序进房通话加速优化，保障通话效果）				
	虚拟背景 （进行视频通话时，可设置模糊背景或替换背景图片，支持 iOS、Android、Flutter、uin-app 和 Web端）	✓	-	-	✓
	多端登录通话 （任一平台接通，自动停止其他端的接入请求）	✓	-	✓	✓
	同平台多端登录通话 （支持同一平台，如多个 iOS 设备登录，来电后任一设备接通，自动停止其他设备的接入请求）	✓ （仅配合 IM 旗舰版使用）	-	✓	✓
	AI 降噪 （借助 AI 能力有效过滤背景噪声，还原通话效果）	✓	-	✓	✓
	弱网通话卡顿优化 （针对户外等弱网环境下优化卡顿率，秒开速度更快）	✓	-	✓	✓
即时通信 IM 功能		根据 IM 版本而定 详见 IM 功能	详见 IM 专业版功能	详见 IM 旗舰版功能	详见 IM 旗舰版功能
实时音视频 TRTC 功能		详见 TRTC 体验版功能	详见 TRTC 基础版功能	详见 TRTC 尊享版功能	详见 TRTC 旗舰版功能
超出赠送资源部分后付费		音视频时长后付费价格详见 音视频时长计费说明，DAU 与峰值群组数后付费价格详见 IM 计费说明			
支持平台		iOS、Android、Web、**微信小程序（仅体验版和群组通话版支持）**、uni-app、Flutter			

说明：

- **包含套餐版本：**TUICallKit 由 TRTC 和 IM 联合提供，使用 TUICallKit 需同时购买 TRTC 包月套餐和 IM 基础套餐包，不同版本套餐提供的 TUICallKit 功能有所不同。
- **DAU：**单个用户当日登录即时通信 IM 计为1个 DAU，同一用户重复登录时，DAU 不累加。
- **峰值群组数：**单个应用（SDKAppID）下所有用户创建和加入的群组数的总和，以自然月中最高峰值进行结算。

续费正式版

您可对正式版应用进行续费，具体操作如下：

1. 登录 实时音视频 TRTC 控制台，单击左侧**应用管理**，选择您需要续期音视频通话（TUICallKit）能力的应用，进入应用管理页面。
2. 在页面的右下角找到**含 UI 低代码场景方案标题 > 视频通话（TUICallKit）**，单击**续期**前往 实时音视频购买页 进行续费。套餐包价格说明请参见 音视频通话能力版本计费说明 。



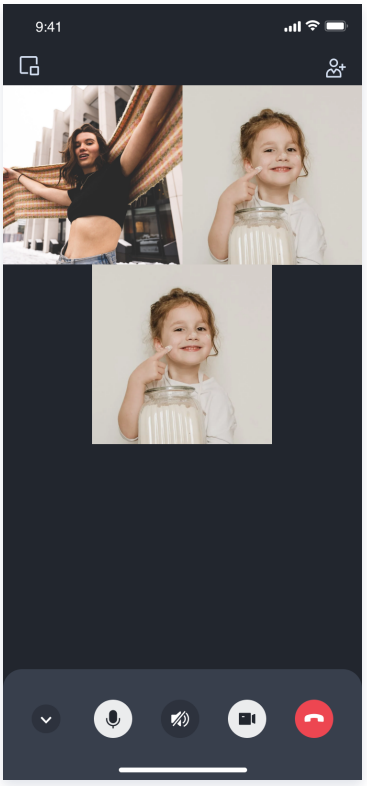
3. 根据已开通的版本进行续费。



跑通 Demo（TUICallKit）
Android

最近更新时间：2024-06-18 15:54:51

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

视频通话	群组通话
	

环境准备

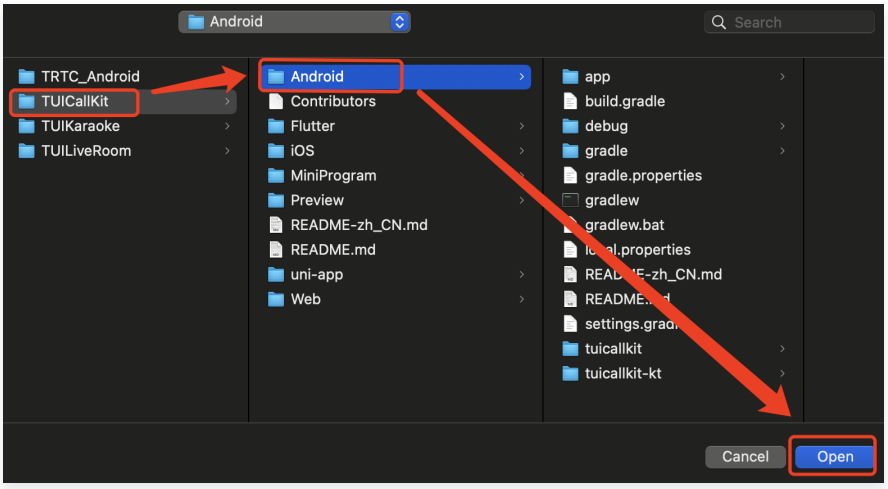
- Android Studio
- 两台 Android 5.0 以上的设备。

步骤一：下载 Demo

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

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

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

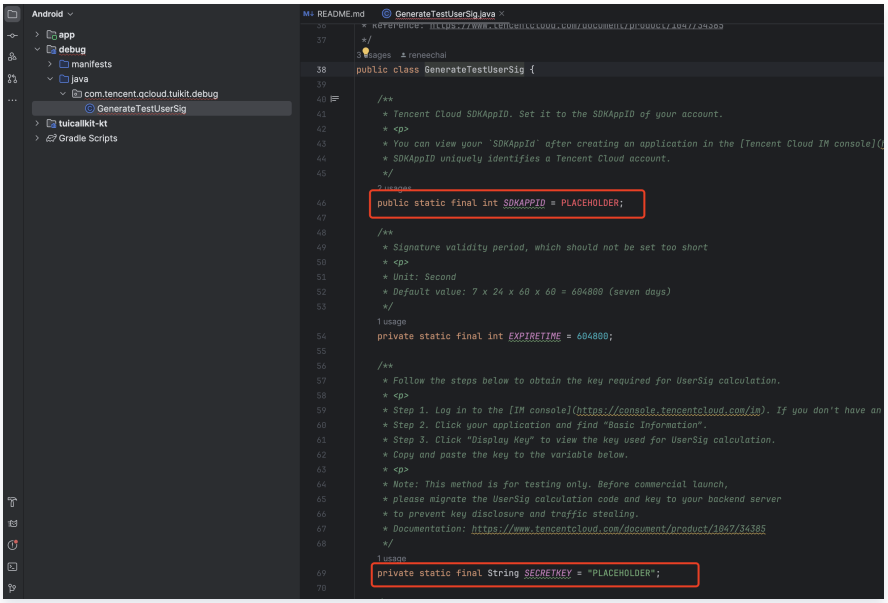


步骤二：配置 Demo

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

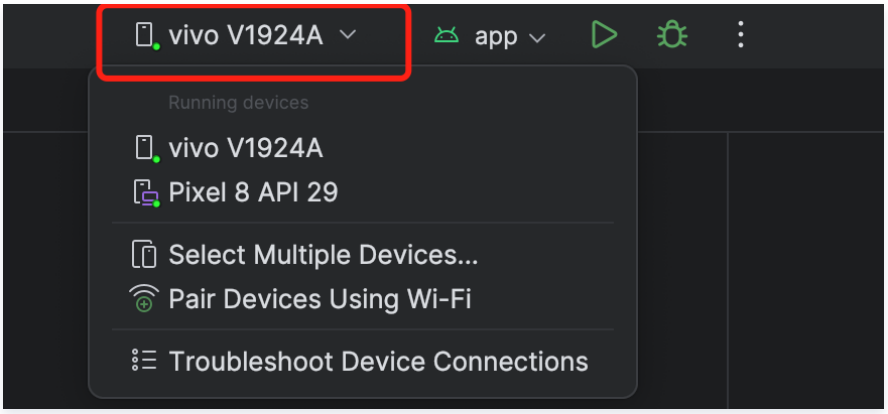


2. 打开 `Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java` 文件，将开通服务时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

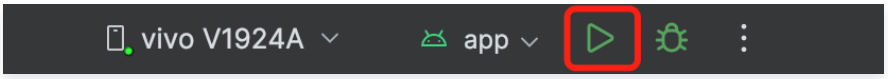


步骤三：跑通 Demo

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



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



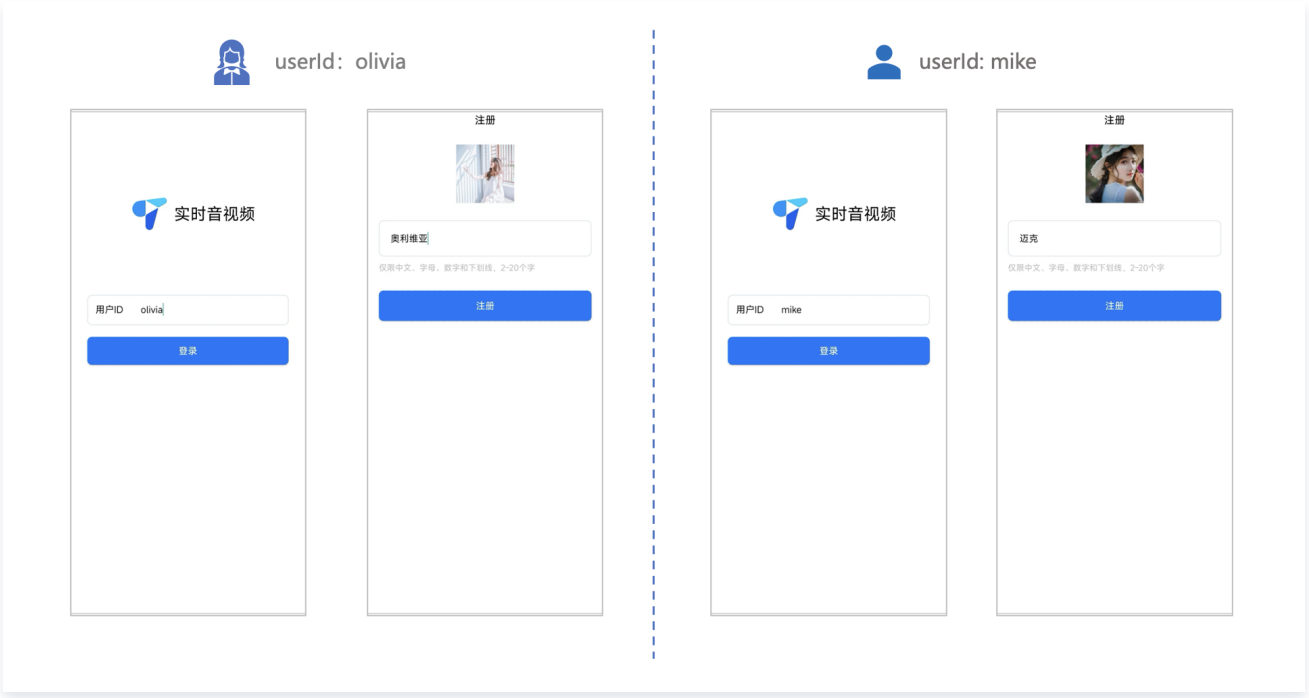
3. 当 Demo 成功运行到设备上后，您可以通过如下步骤来发起一次通话：

拨打第一个电话

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

1. 登录 & 注册

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

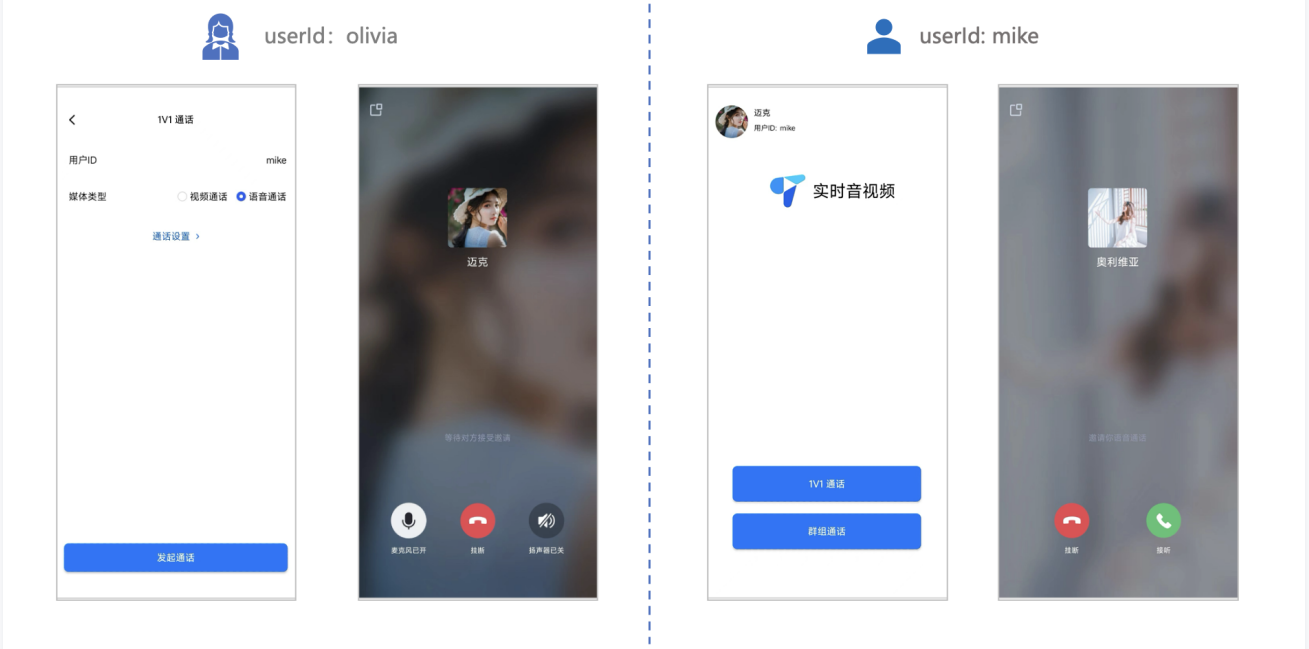


> **注意：**
> 尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

2. 拨打电话

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

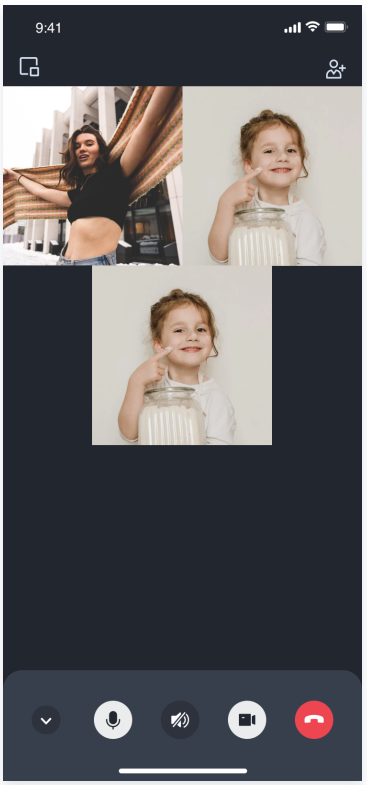
2.2 单击**发起通话**。


userId：olivia
1V1 通话
用户ID
mike
媒体类型
视频通话
语音通话
通话设置 >
发起通话
迈克
userId: mike
迈克
用户ID: mike
实时音视频
1V1 通话
群组通话
奥利维亚

iOS

最近更新时间：2024-06-07 09:42:21

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

视频通话	群组通话
	

环境准备

- Xcode 13 及以上。
- 两台 iOS 13.0 及以上设备。

步骤一：下载 Demo

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

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

2. 在命令行中进入 iOS 项目目录：

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

3. 加载依赖库：

```
pod install
```

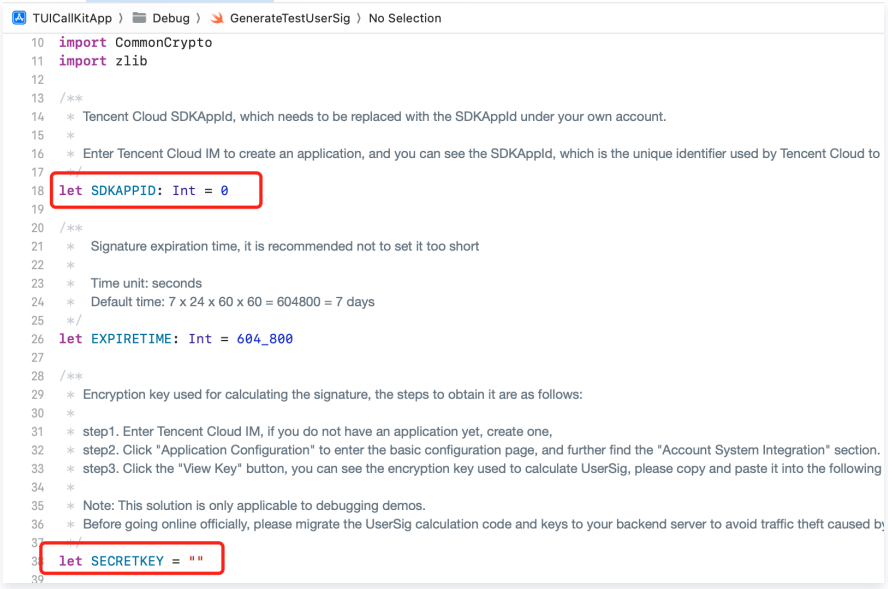
> ⚠ **注意：**
> 如果您尚未安装 CocoaPods，可以参考 此处 了解如何安装。

步骤二：配置 Demo

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

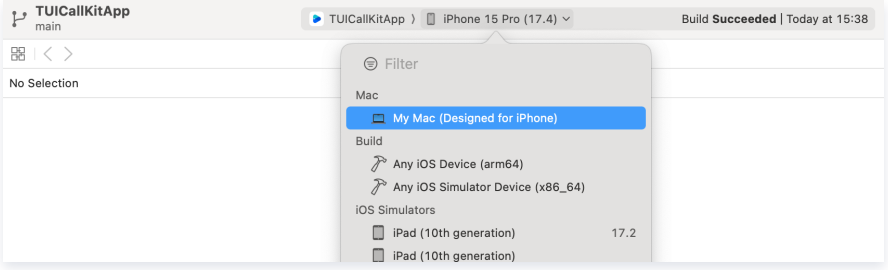


2. 打开 `/iOS/Example/Debug/GenerateTestUserSig.swift` 文件，将 开通服务 时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：



步骤三：跑通 Demo

1. 在 XCode 如下图所示处选择您要将 Demo 运行的设备：



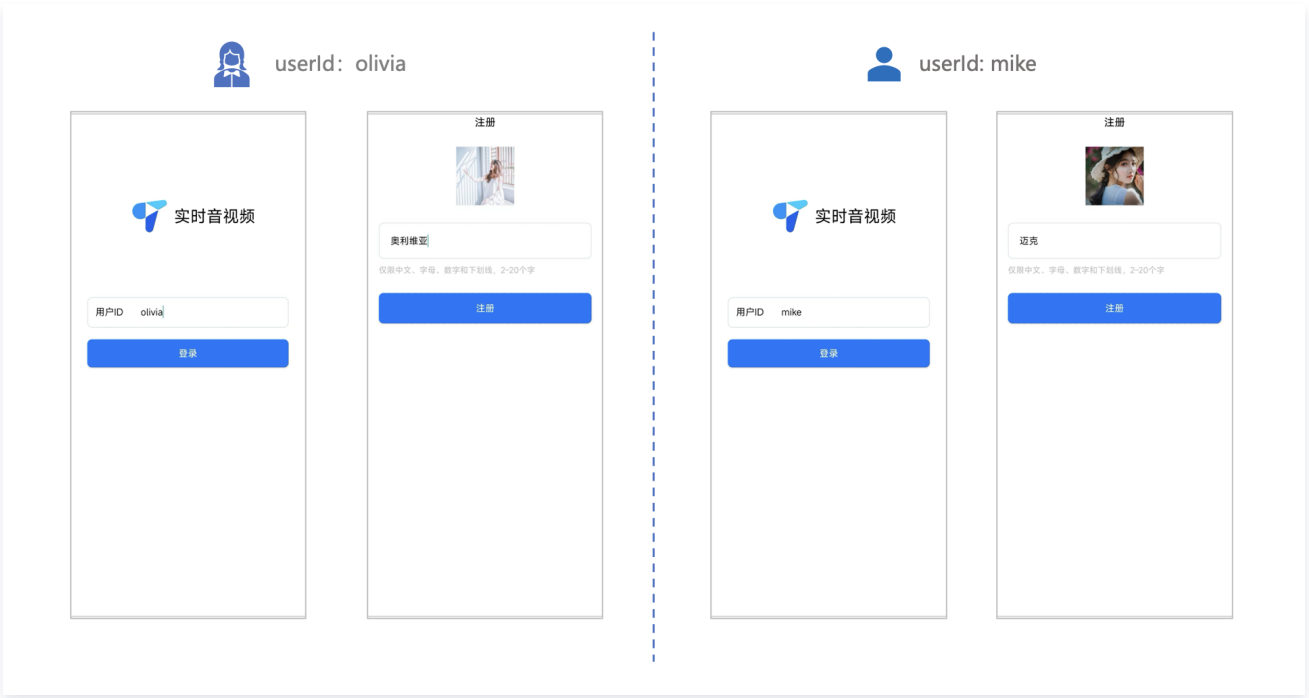
2. 选择完成后点击运行将我们的 TUICallKit Android Demo 运行到目标设备上。

拨打第一个电话

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

1. 登录 & 注册

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

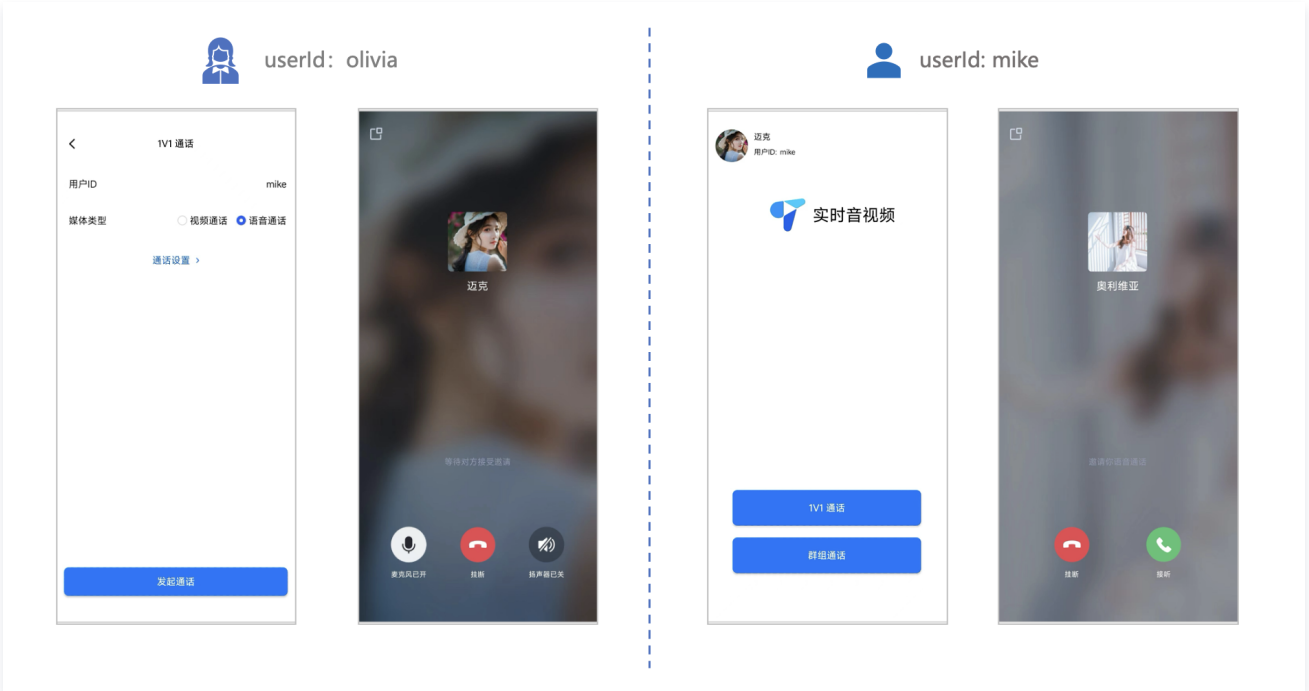


> **注意：**
> 尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

2. 拨打电话

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

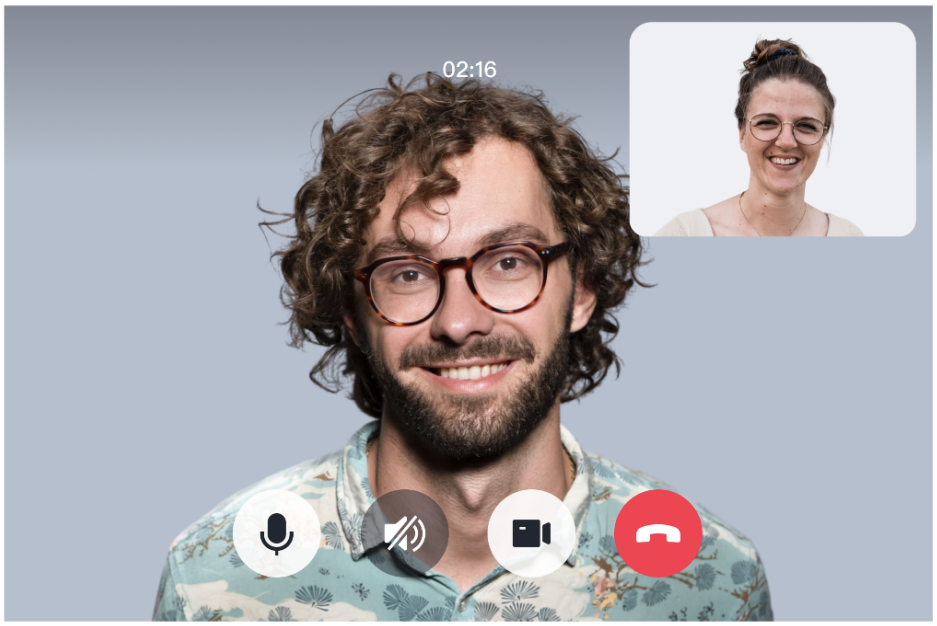
2.2 单击**发起通话**。

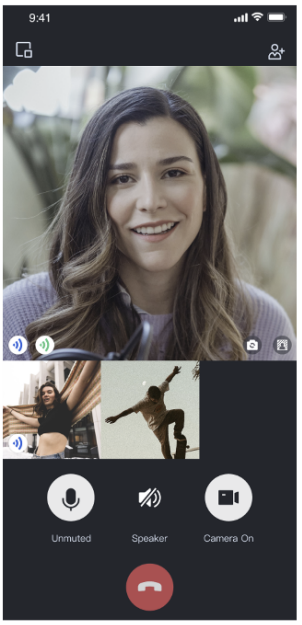


Web

最近更新时间：2024-11-21 09:43:52

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





环境准备

- Node.js, 如果没有，请点击下载。
- Modern browser, supporting WebRTC API.

步骤一：下载 Demo

1. 打开终端，克隆仓库。

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

2. 安装依赖。

React

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

Vue3

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

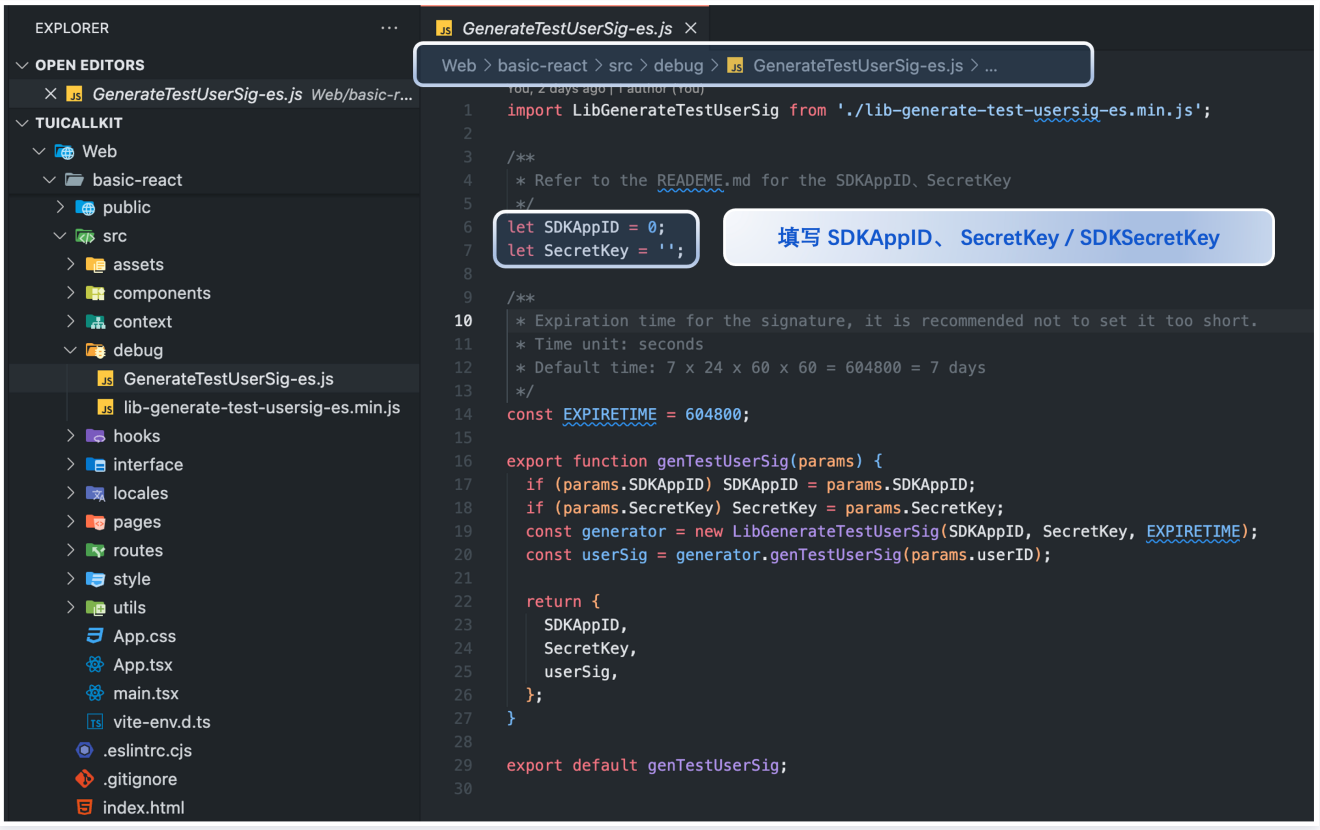
```
npm install
```

步骤二：配置 Demo

1. 点击进入开通服务页面，获取 `SDKAppID`、`SecretKey` 。
2. 填写 `SDKAPPID`、`SecretKey` 。

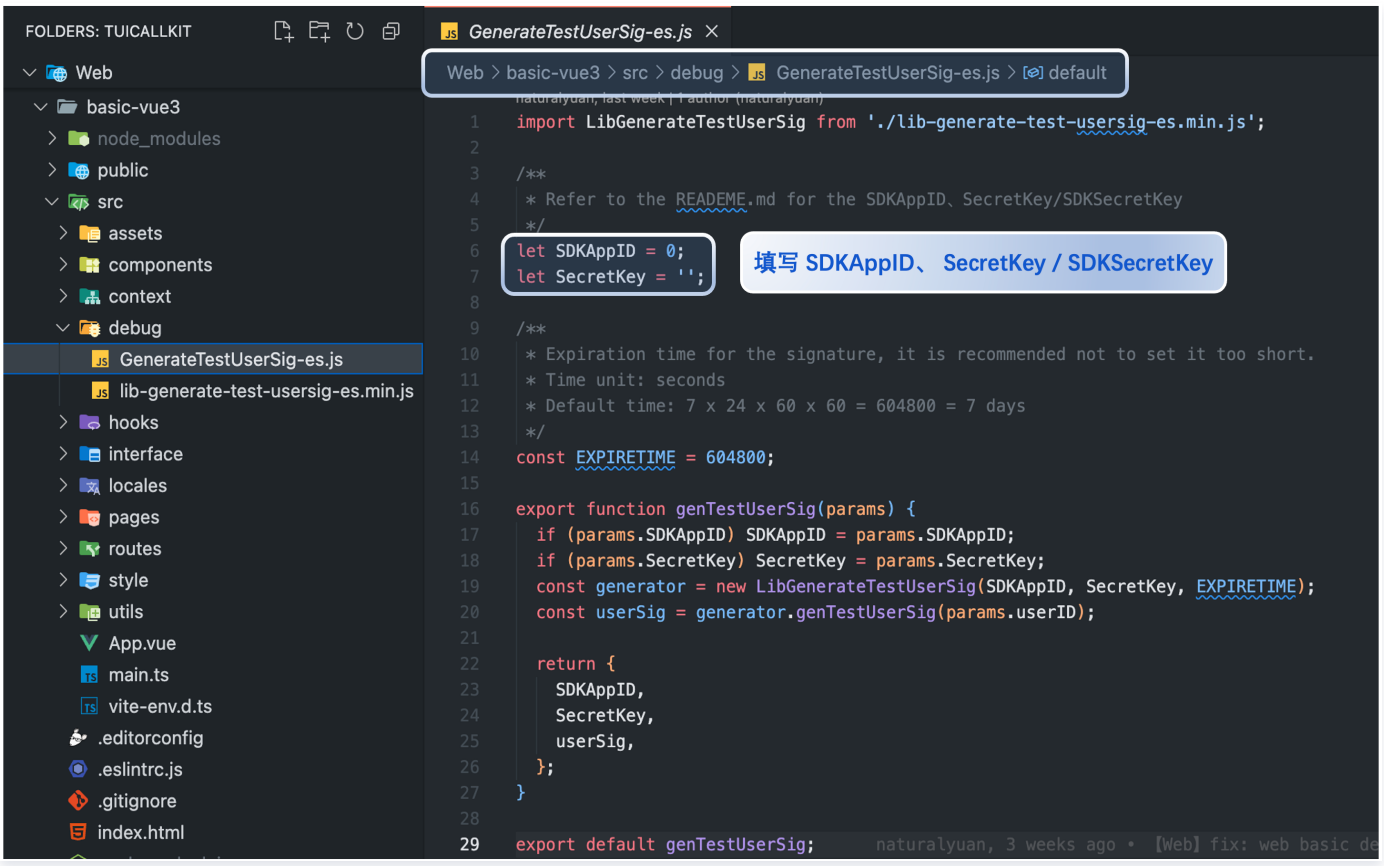
React

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



Vue3

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



步骤三：运行 Demo

打开终端，复制示例命令运行 Demo。

TUICallKit/Web/basic-react

```
npm run dev
```

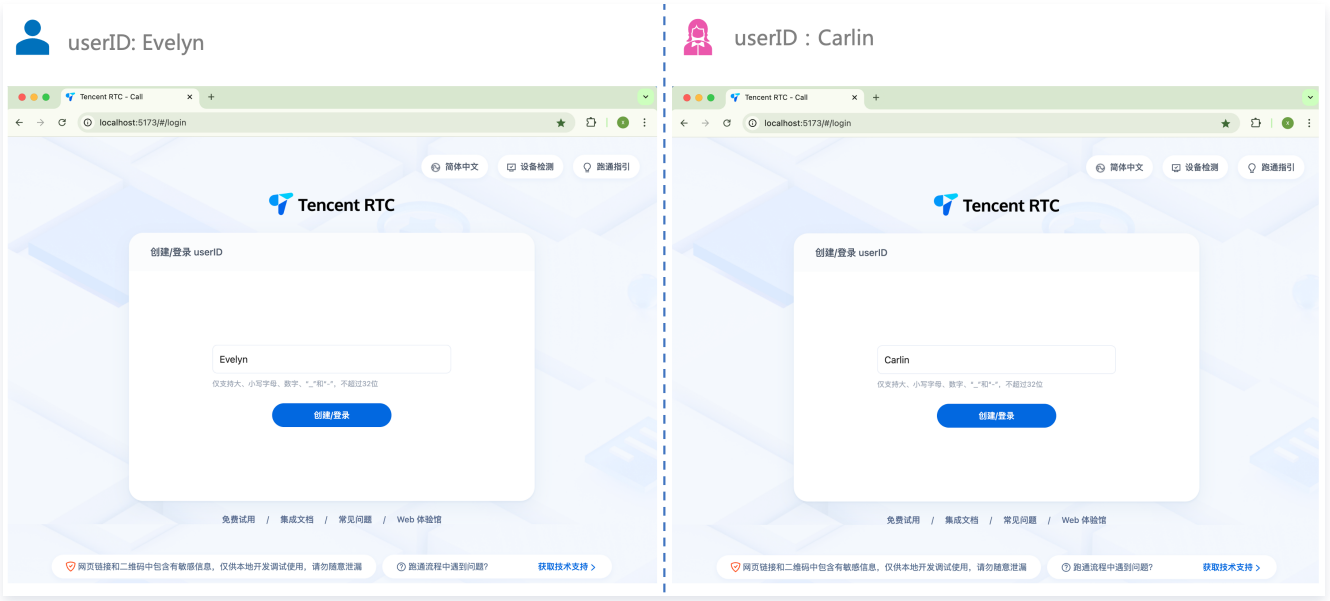
TUICallKit/Web/basic-vue3

```
npm run dev
```

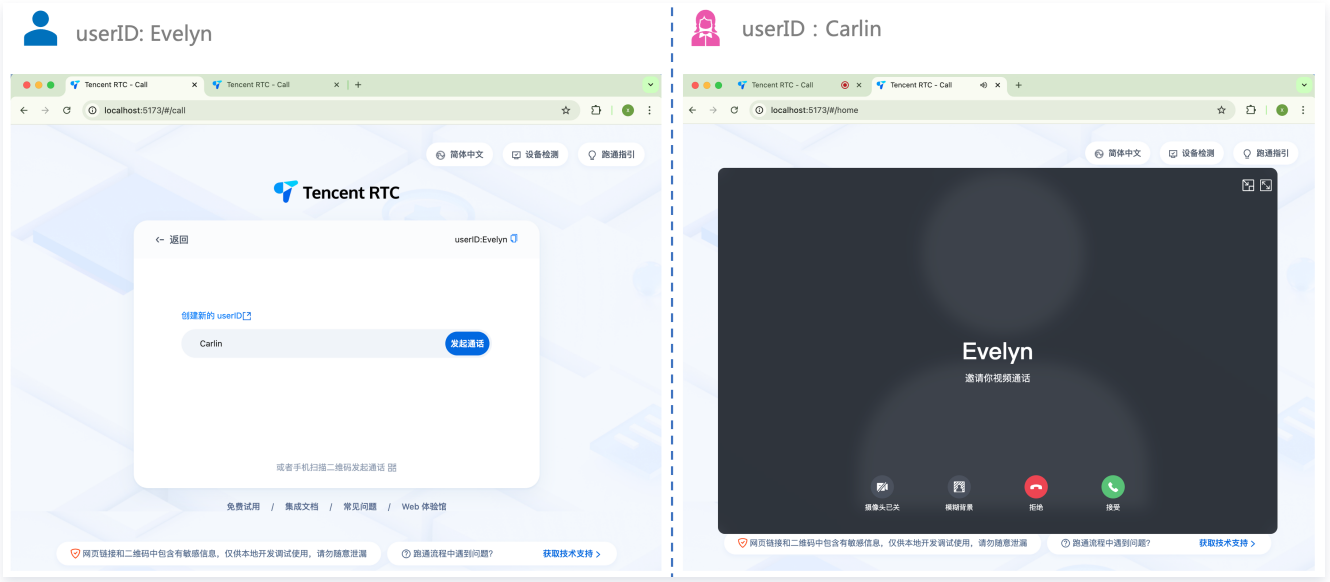
> **警告：**
> **本地环境请在 localhost 协议下访问，公网体验请在 HTTPS 协议下访问，具体参见 网络访问协议说明 。**

步骤四：拨打您的第一通电话

1. 打开浏览器页面，输入项目运行地址，登录 userID（由您定义）。



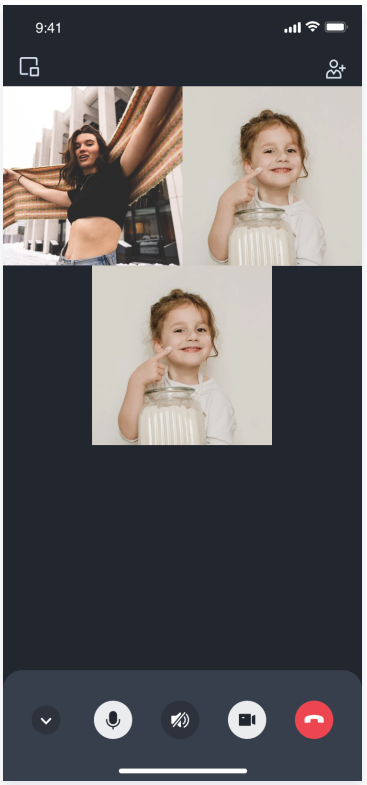
2. 输入被叫方的 userID，单击 `发起通话` 体验您的第一通话。



Flutter

最近更新时间：2024-06-18 15:54:51

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

视频通话	群组通话
	

环境准备

Flutter

Flutter 3.0 及以上版本。

Android

- Android Studio 3.5及以上版本。
- Android 4.1 及以上版本的 Android 设备。

iOS

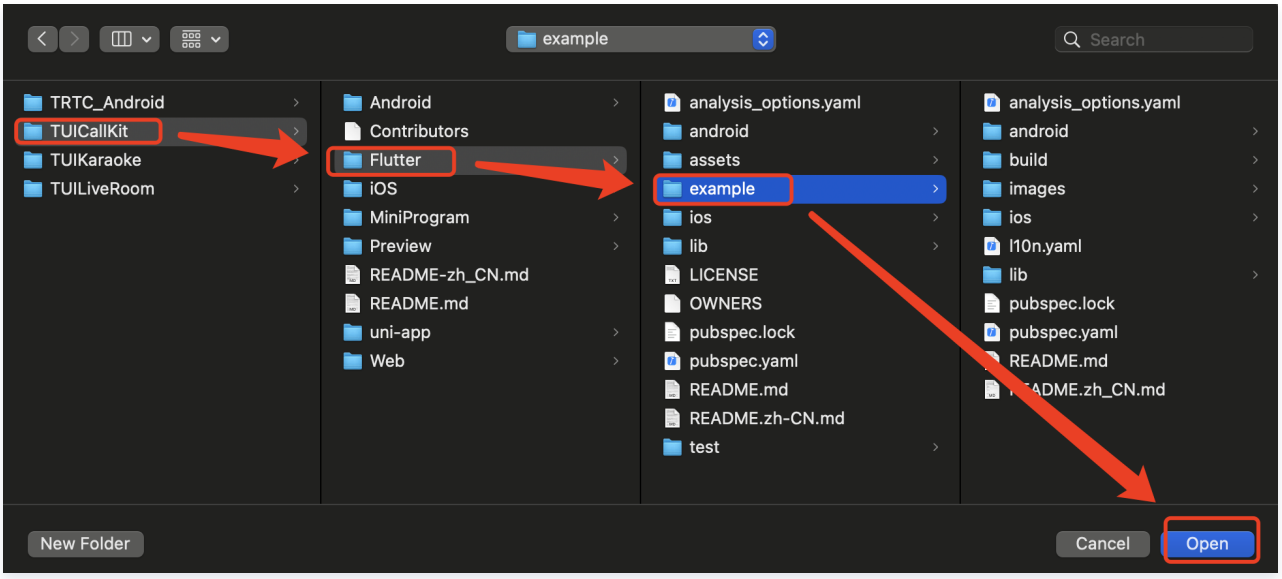
- Xcode 13.0及以上版本。
- 请确保您的项目已设置有效的开发者签名。

步骤一：下载 Demo

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

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

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

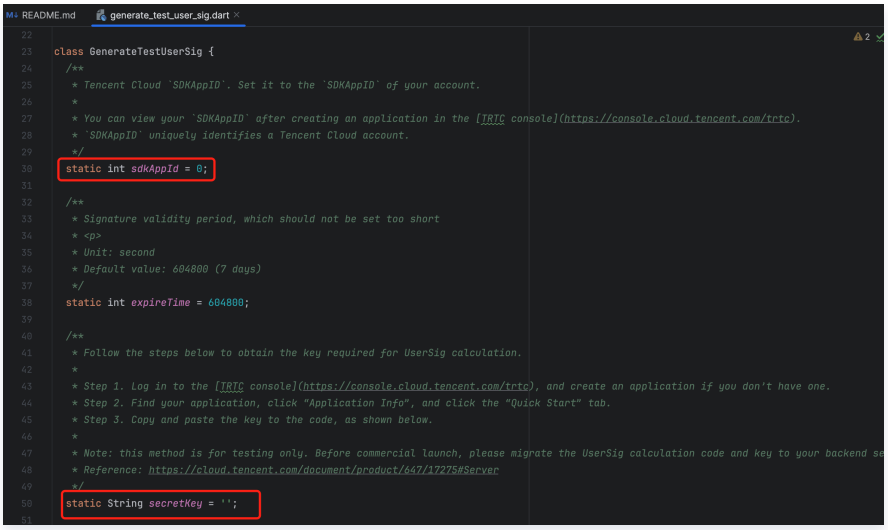


步骤二：配置 Demo

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

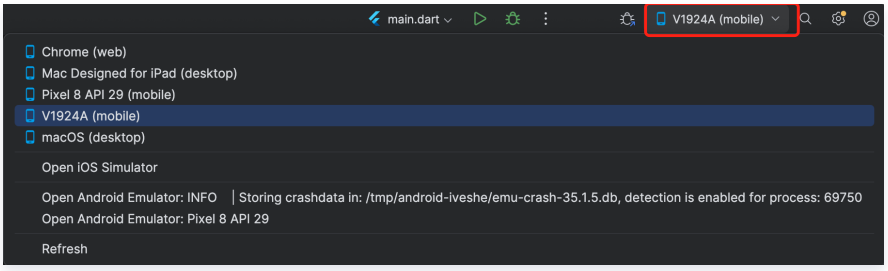


2. 打开 `Flutter/example/lib/debug/generate_test_user_sig.dart` 文件，将开通服务时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

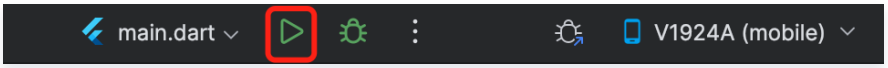


步骤三：跑通 Demo

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



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

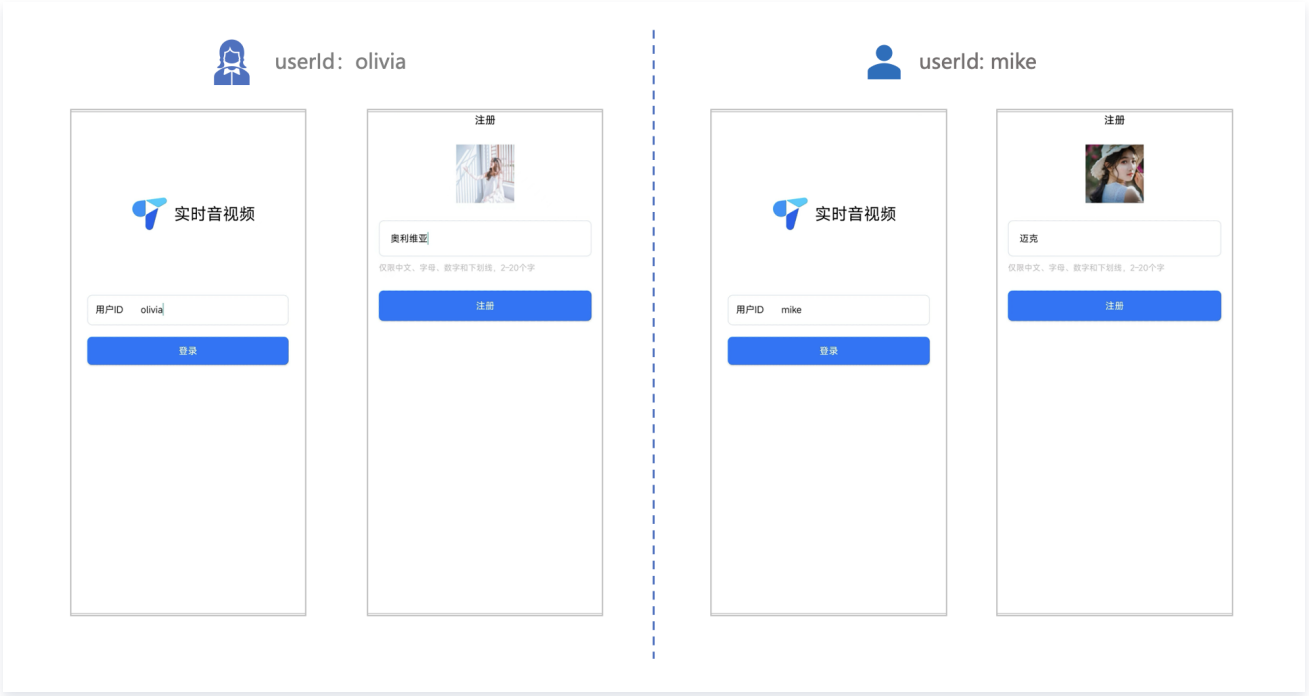


拨打第一个电话

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

1. 登录 & 注册

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

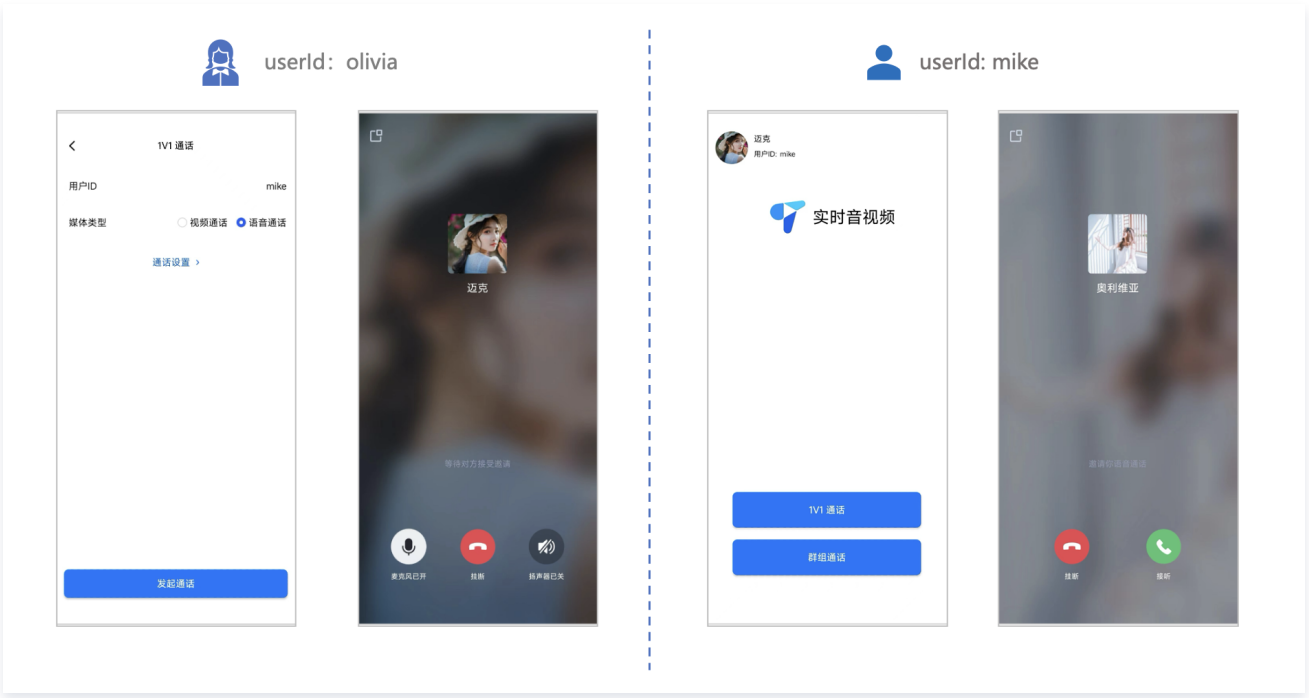


> **注意：**
> 尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

2. 拨打电话

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

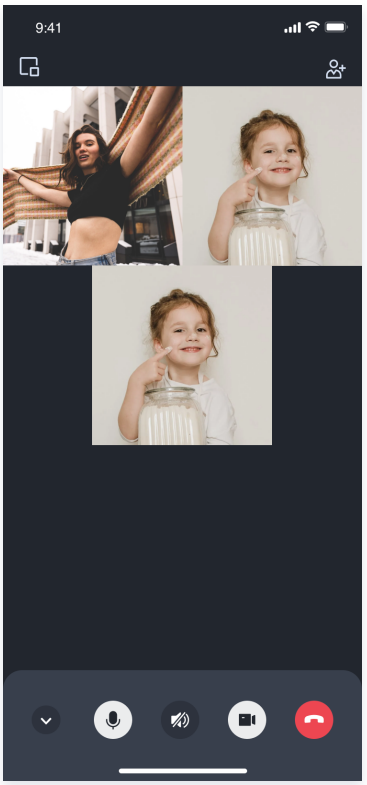
2.2 单击**发起通话**。


userId：olivia
1V1 通话
用户ID
mike
媒体类型
视频通话
语音通话
通话设置
发起通话
迈克
userId: mike
迈克
用户ID: mike
实时音视频
1V1 通话
群组通话
奥利维亚

微信小程序

最近更新时间：2025-05-30 15:27:52

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

视频通话	群组通话
	

开发环境要求

- 微信 App iOS 最低版本要求：8.0.40。
- 微信 App Android 最低版本要求：8.0.40。
- 小程序基础库最低版本要求：2.10.0。

步骤一：下载 Demo

1. 打开终端，克隆仓库。

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

2. 安装依赖。

```
cd ./TUICallKit/MiniProgram/wechat-callkit-demo
```

```
npm install
```

3. 拷贝源码

Windows 端

```
xcopy node_modules\@tencentcloud\call-uikit-wx\ .\TUICallKit /i /e
xcopy node_modules\@tencentcloud\call-engine-wx\RTCCallEngine.wasm.br .\static\
```

Mac 端

```
mkdir -p ./TUICallKit && cp -r node_modules/@tencentcloud/call-uikit-wx/ ./TUICallKit && mkdir -p
./static && cp node_modules/@tencentcloud/call-engine-wx/RTCCallEngine.wasm.br ./static/
```

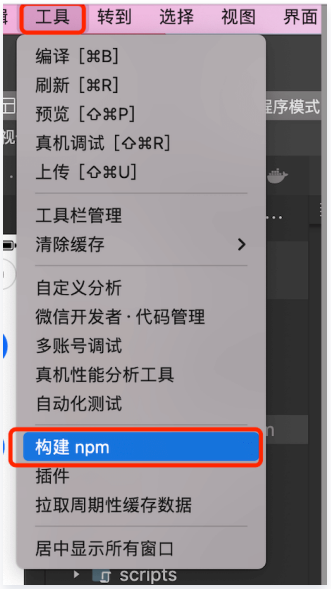
步骤二：配置 Demo

1. 单击 进入开通服务页面，获取 `SDKAppID`、`SDKSecretKey` 。
2. 打开 TUICallKit/MiniProgram/TUICalling/debug/GenerateTestUserSig.js 文件，将开通服务时获取到对应的 SDKAppID 和 SDKSecretKey 填入其中。

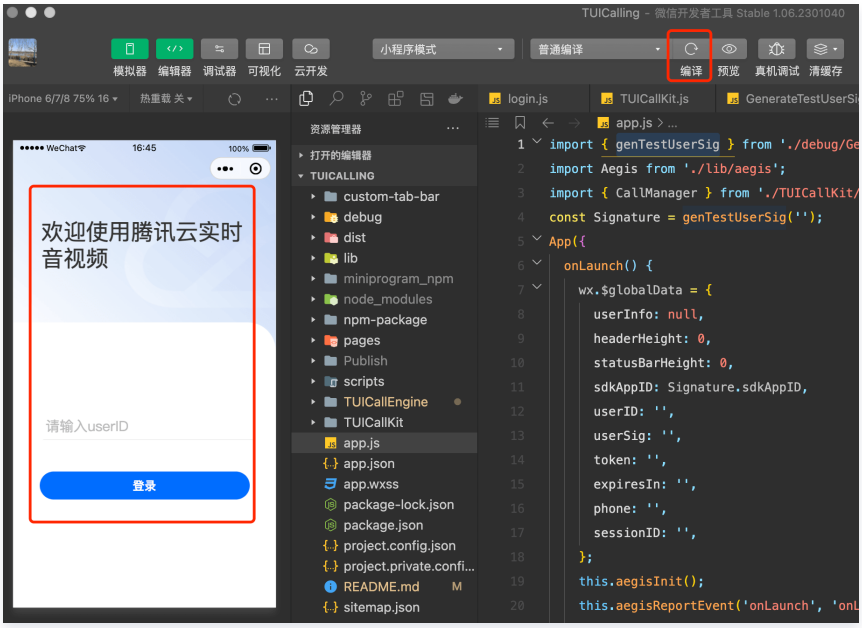
```
function genTestUserSig(userID) {
  /**
   * 腾讯云 SDKAppId，需要替换为您自己账号下的 SDKAppId。
   *
   * 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav ) 创建应用，即可看到 SDKAppId，
   * 它是腾讯云用于区分客户的唯一标识。
   */
  var SDKAPPID    = 0;  将 SDKAppID 粘贴到此处

  /**
   * 签名过期时间，建议不要设置的过短
   * <p>
   * 时间单位：秒
   * 默认时间：7 x 24 x 60 x 60 = 604800 = 7 天
   */
  var EXPIRETIME = 604800;

  /**
   * 计算签名用的加密密钥，获取步骤如下：
   *
   * step1. 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav )，如果还没有应用就创建一个，
   * step2. 单击“应用配置”进入基础配置页面，并进一步找到“帐号体系集成”部分。
   * step3. 点击“查看密钥”按钮，就可以看到计算 UserSig 使用的加密的密钥了，请将其拷贝并复制到如下的变量中
   *
   * 注意：该方案仅适用于调试Demo，正式上线前请将 UserSig 计算代码和密钥迁移到您的后台服务器上，以避免加密密钥泄露导致的流量盗用。
   * 文档：https://cloud.tencent.com/document/product/647/17275#Server
   */
  var SECRETKEY = "";  将 密钥 粘贴到此处
```

步骤三：跑通 Demo

1. 项目导入到微信开发者工具，构建 npm。打开微信开发者工具选择**工具 > 构建 npm**。具体如下图：



2. 在微信开发者工具编译运行。



拨打第一个电话

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

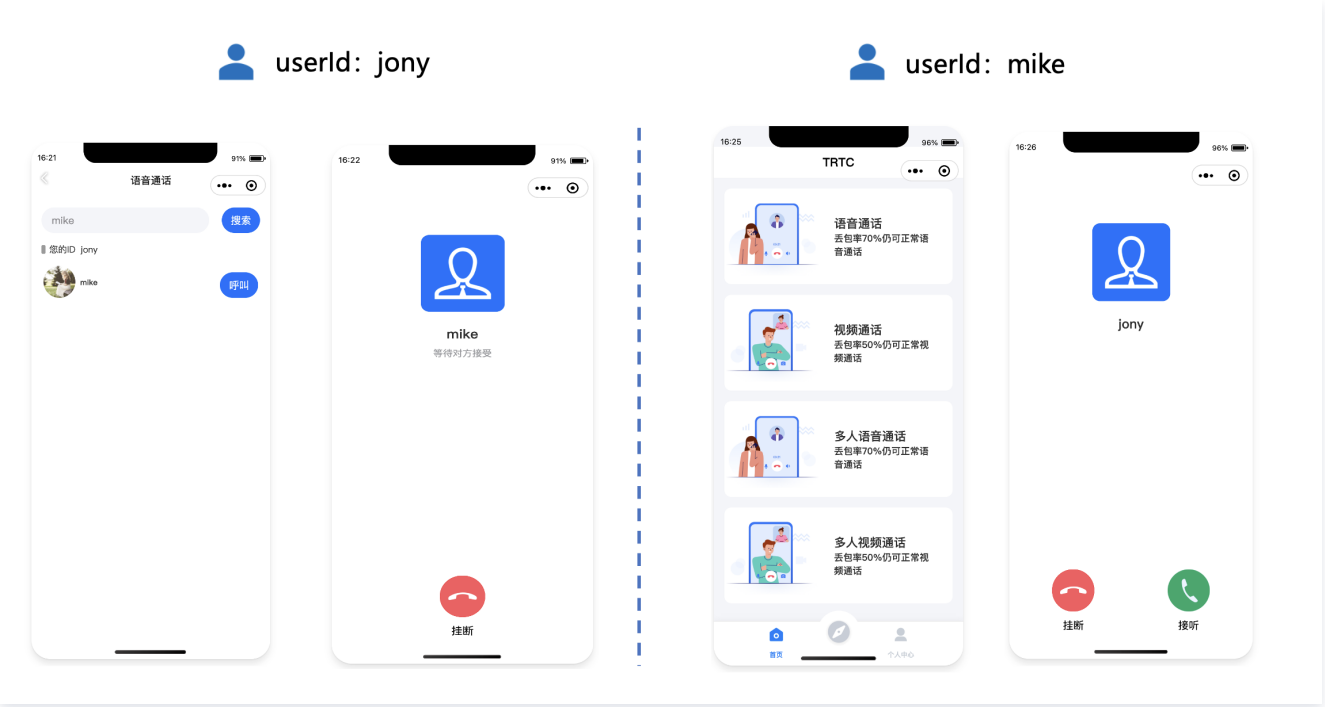
1. 请您在 `用户ID` 处输入 ID



> **注意：**

> 尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

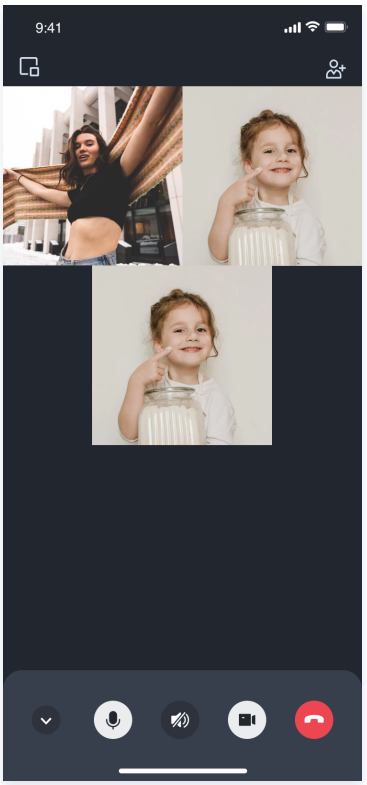
2. 拨打电话



uni-app（小程序）

最近更新时间：2025-05-30 15:27:52

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

视频通话	群组通话
	

开发环境要求

- 微信 App iOS 最低版本要求：8.0.40。
- 微信 App Android 最低版本要求：8.0.40。
- 小程序基础库最低版本要求：2.10.0。

步骤一：下载 Demo

1. 打开终端，克隆仓库。

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

2. 安装依赖。

```
cd ./TUICallKit/uni-app/TUICallKit-Miniprogram/TUICallKit-Vue3
```

```
npm install
```

3. 拷贝源码

window 端

```
xcopy node_modules\@tencentcloud\call-uikit-wx-uniapp\ .\TUICallKit /i /e
xcopy node_modules\@tencentcloud\call-engine-wx\RTCCallEngine.wasm.br .\static
```

mac 端

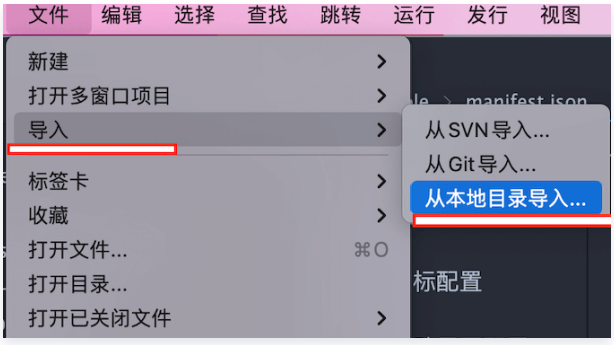
```
mkdir -p ./TUICallKit && cp -r node_modules/@tencentcloud/call-uikit-wx-uniapp/ ./TUICallKit && mkdir -
p ./static && cp node_modules/@tencentcloud/call-engine-wx/RTCCallEngine.wasm.br ./static
```

步骤二：配置 Demo

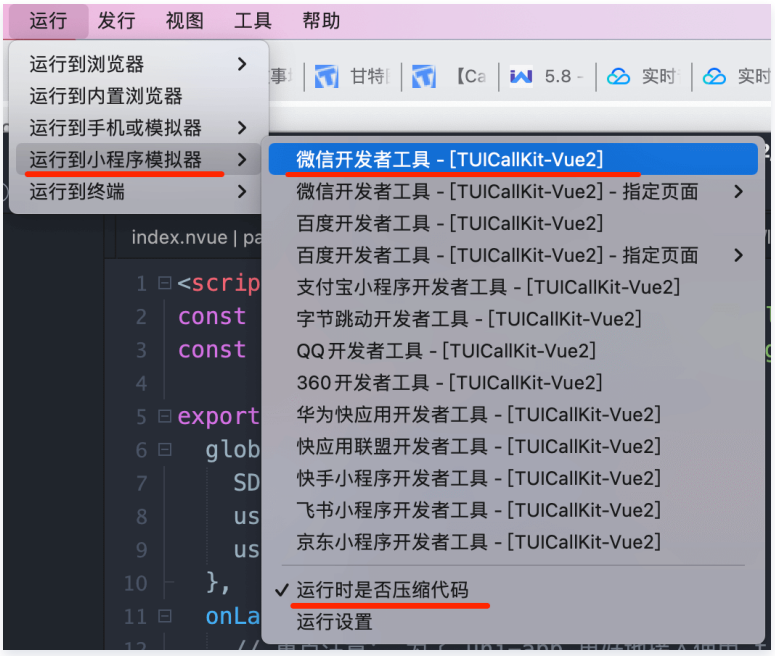
1. 单击 进入开通服务页面，获取 `SDKAppID`、`SDKSecretKey` 。
2. 打开 ./TUICallKit/uni-app/TUICallKit-Miniprogram/TUICallKit-Vue3/TUICallKit/debug/GenerateTestUserSig-es.js 文件，将开通服务时获取到对应的 SDKAppID 和 SDKSecretKey 填入其中。

步骤三：跑通 Demo

1. HBuilder 中导入项目。



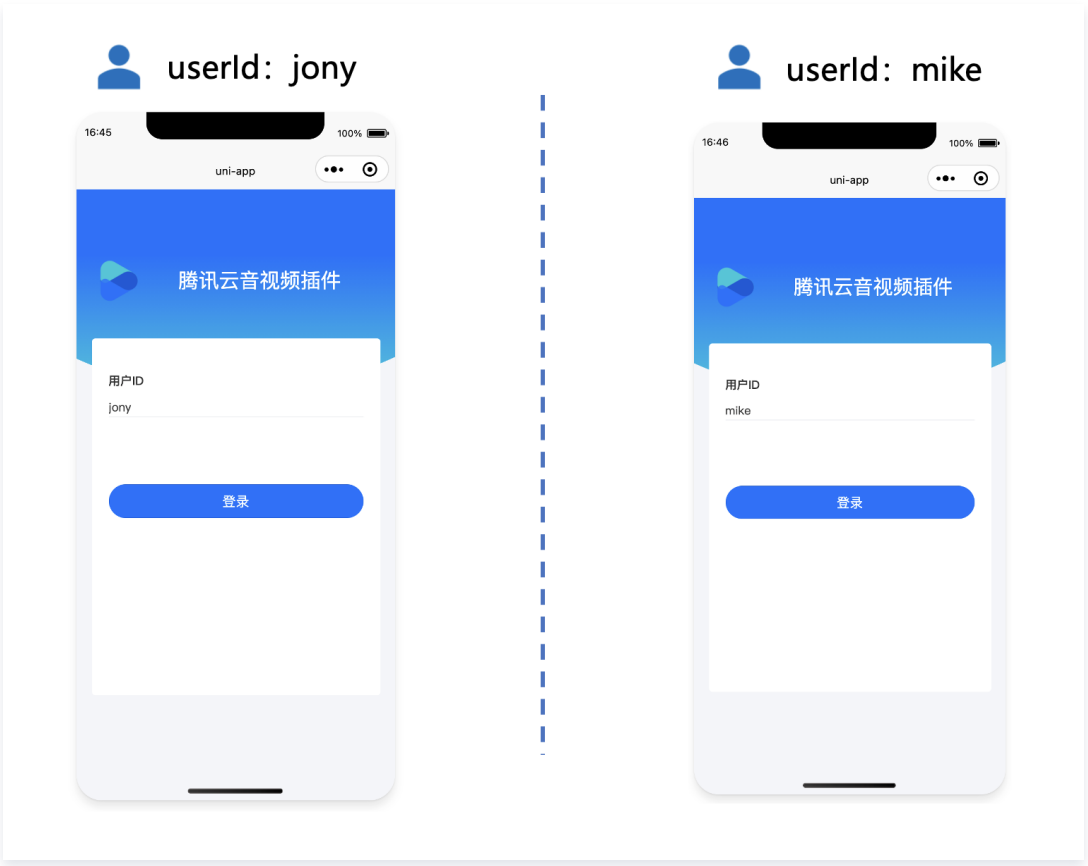
2. 运行到**微信开发者工具**，勾选**运行时是否压缩代码**。



拨打第一个电话

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

1. 请您在 `用户ID` 处输入 ID



> **注意：**
> 尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

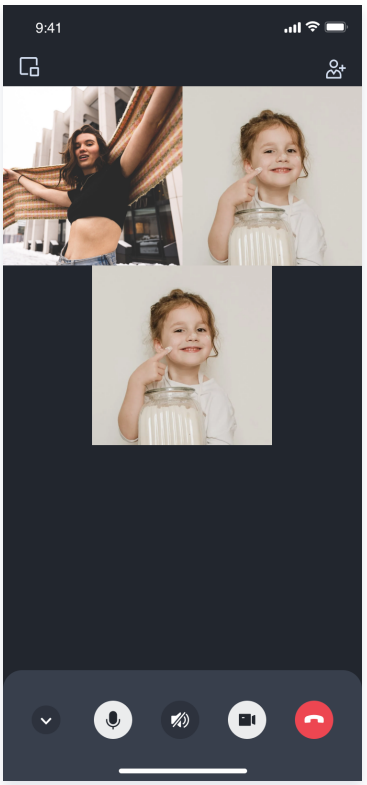
2. 拨打电话



uni-app（客户端）

最近更新时间：2024-10-22 16:39:11

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

视频通话	群组通话
	

环境准备

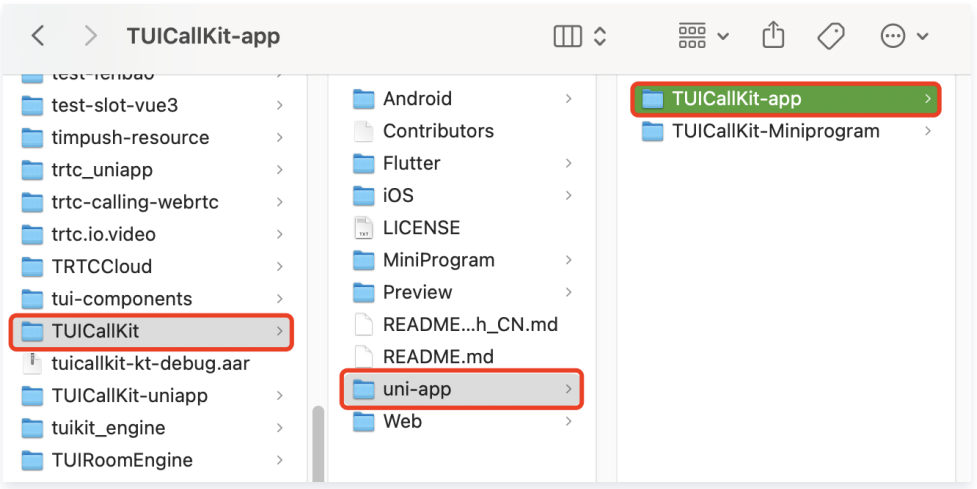
- HBuider X
- 两个移动设备： Android 5.0 以上的设备 / iOS 13.0 及以上设备。

步骤一：下载 Demo

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

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

2. 通过 HbuiderX 打开 TUICallKit **uni-app/TUICallKit-app** 项目：



步骤二：配置 Demo

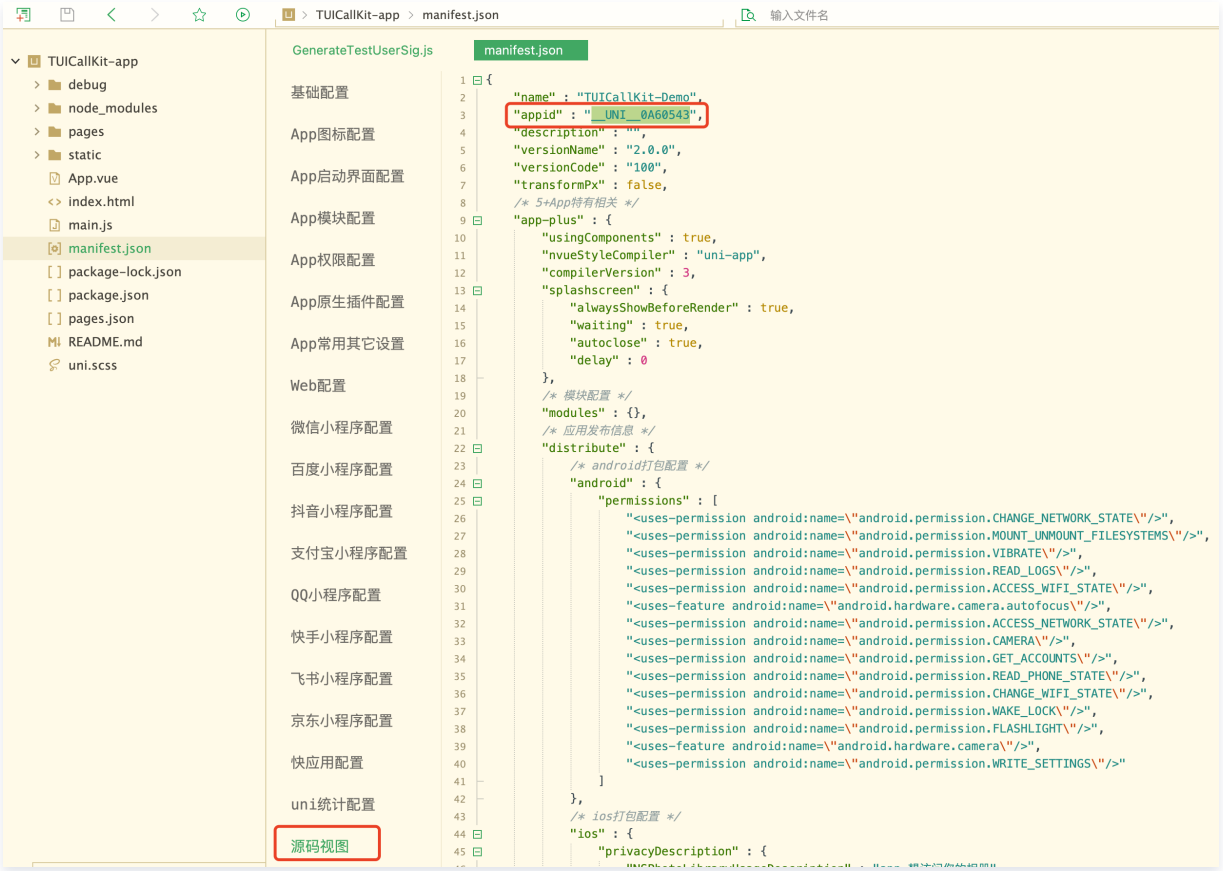
1. 点击进入开通服务页面，获取 `SDKAppID`、`SecretKey` 。
2. 填写 'TUICallKit/uni-app/TUICallKit-app/debug/GenerateTestUserSig-es.js' 文件下的 `SDKAPPID`、`SecretKey` 。



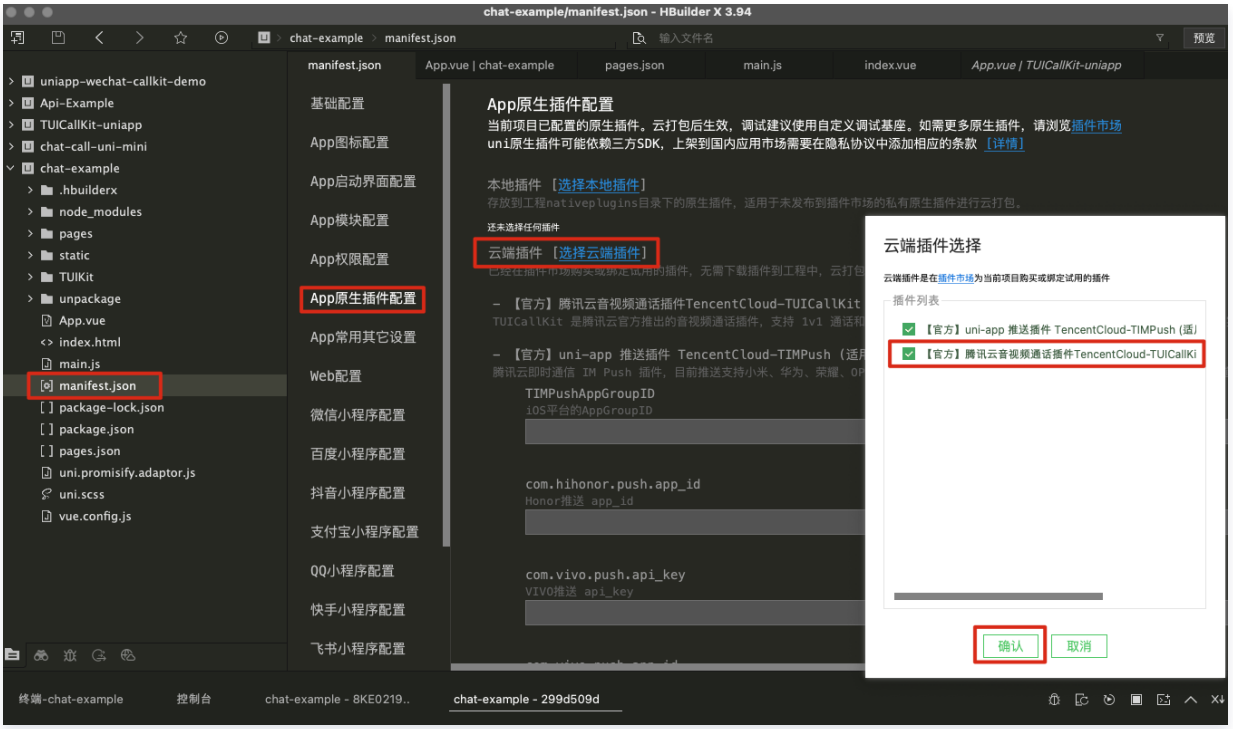
3. 访问 TencentCloud-TUICallKit 插件，在插件详情页中购买插件（免费），购买插件时选择对应的 AppID，绑定正确的包名。



4. 替换 TUICallKit-app/manifest.json 文件下的 appid 为上述绑定过 TencentCloud-TUICallKit 插件的 appid。



5. 在 TUICallKit-app 项目中导入插件。



步骤三：跑通 Demo

1. 制作自定义调试基座，请选择**传统打包**方式进行打包。





2. 自定义调试基座成功后，**使用自定义基座运行项目**。



步骤四：拨打第一个电话

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

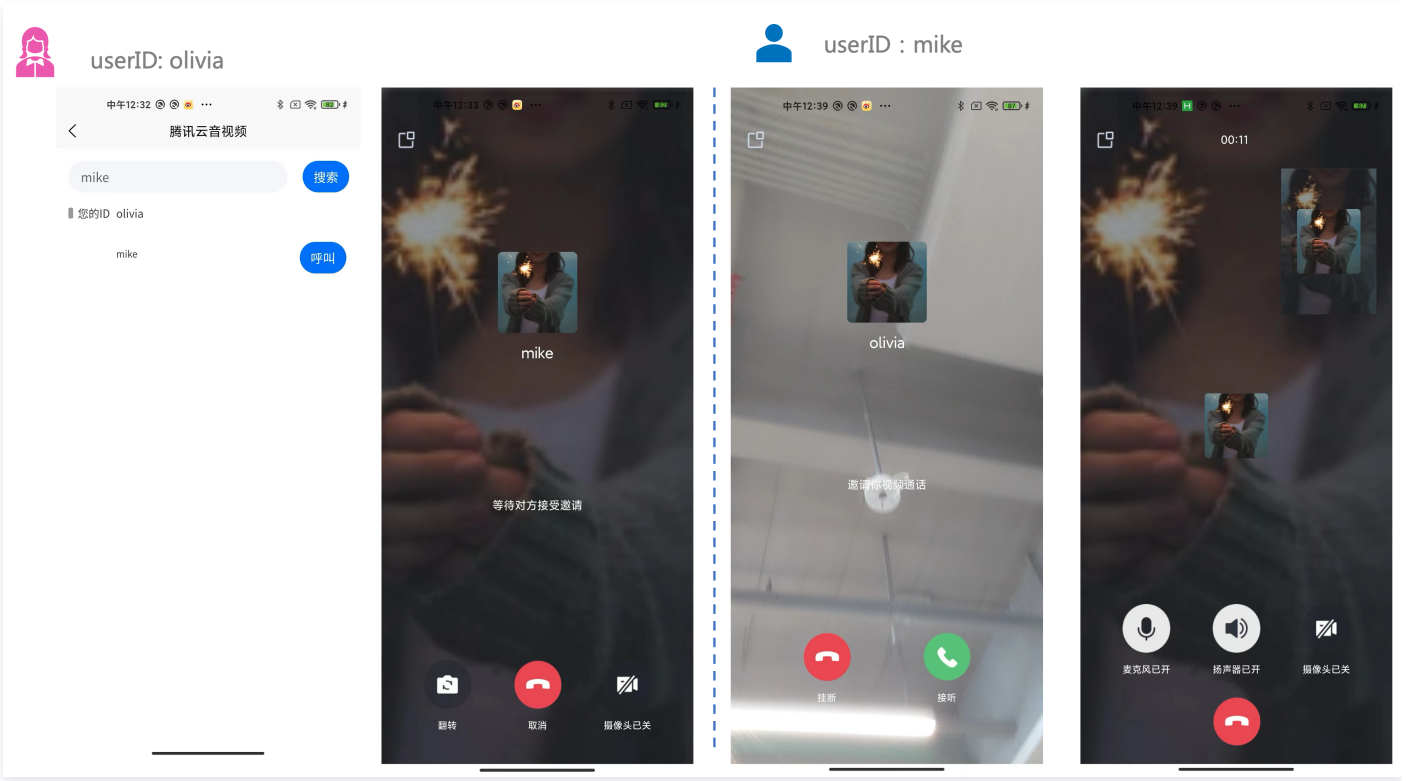
1. 登录/注册 userID（由您定义）。



注意：

尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

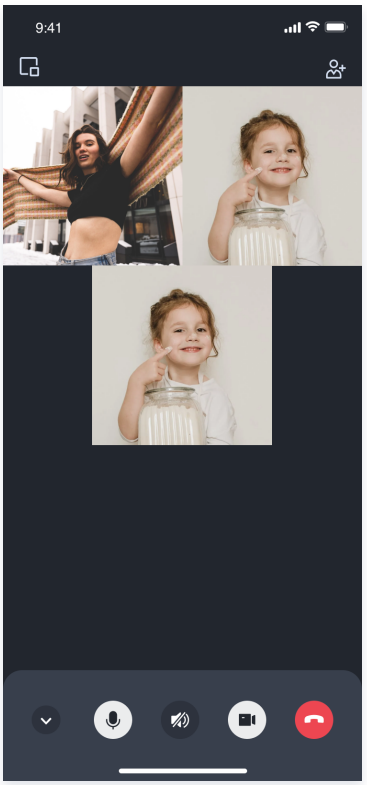
2. 拨打电话


userID: olivia
userID：mike
腾讯云音视频
mike
搜索
您的ID olivia
mike
呼叫
mike
等待对方接受邀请
翻转
取消
摄像头已关
olivia
邀请你视频通话
挂断
接听
00:11
麦克风已开
扬声器已开
摄像头已关

uni-app（uts 插件）

最近更新时间：2025-06-27 17:35:32

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

视频通话	多人通话
	

环境准备

> **说明：**
> HBuilder x 4.64 和 4.65 版本对 uts 插件打包存在兼容性问题，建议使用最新版本。

- HBuilder X
- 两个移动设备： Android 5.0 以上的设备 / iOS 13.0 及以上设备。

步骤1：下载 Demo

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

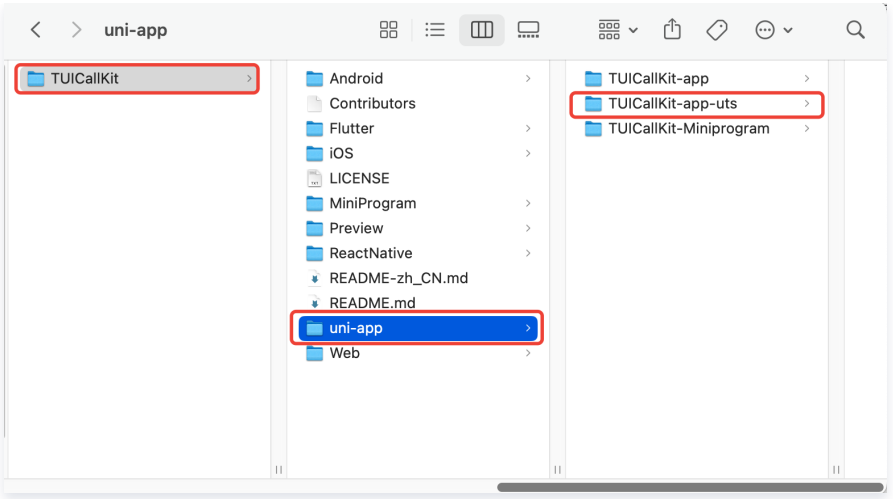
```
git clone https://github.com/Tencent-RTC/TUICallKit.git
```

2. 安装依赖。

```
cd ./TUICallKit/uni-app/TUICallKit-app-uts
```

```
npm install
```

3. 通过 HBuilderX 打开 TUICallKit 的 **uni-app/TUICallKit-app-uts** 项目：



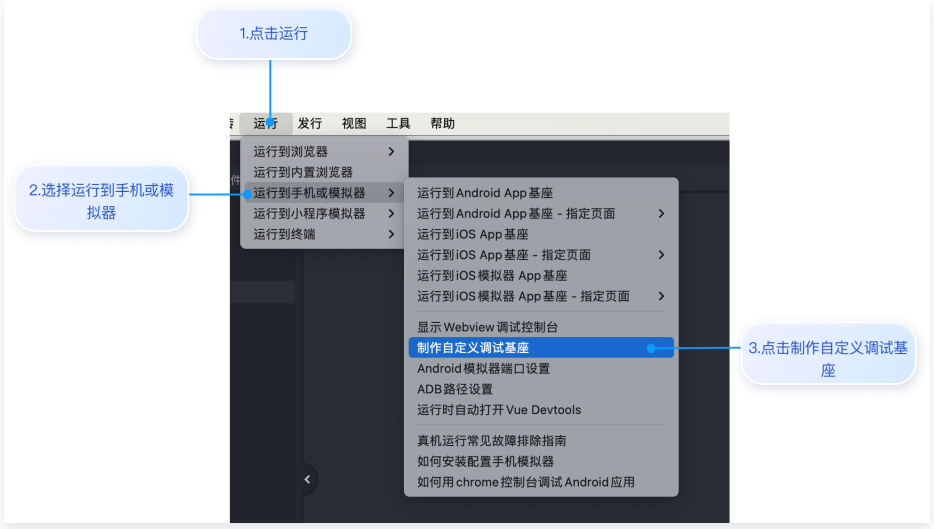
步骤2：配置 Demo

1. 获取 SDKAppID、SecretKey，您需 开通服务 以获取 SDKAppID、SecretKey。
2. 填写 'TUICallKit/uni-app/TUICallKit-uts/debug/GenerateTestUserSig-es.js' 文件下的 `SDKAppID、SECRETKEY` 。

```
import LibGenerateTestUserSig from "./lib-generate-test-usersig-es.min.js";

let SDKAppID = 0;

let SECRETKEY ="";

/**
 * Expiration time for the signature, it is recommended not to set it too short.
 * Time unit: seconds
 * Default time: 7 x 24 x 60 x 60 = 604800 = 7 days
 */
const EXPIRETIME = 604800;

export function genTestUserSig(userID) {
  const generator = new LibGenerateTestUserSig(SDKAppID, SECRETKEY, EXPIRETIME);
  const userSig = generator.genTestUserSig(userID);

  return {
    SDKAppID,
    userSig,
  };
}
```

步骤3：跑通 Demo

1. 制作自定义调试基座，请选择**传统打包**方式进行打包。





2. 自定义调试基座成功后，**使用自定义基座运行**项目。



步骤4：拨打第一通电话

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

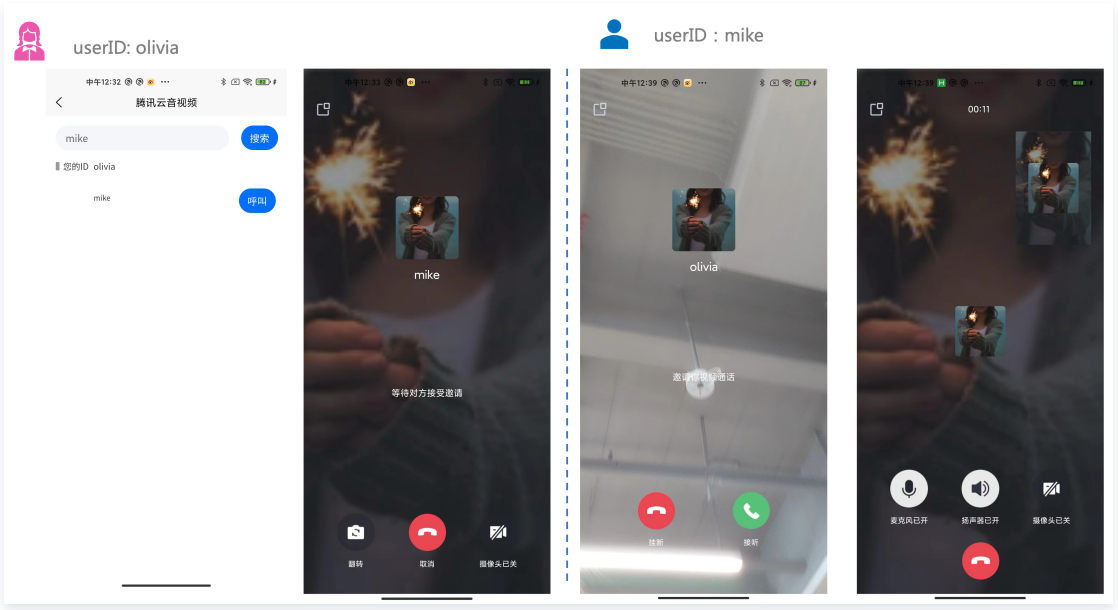
1. 登录/注册 userID（由您定义）。



注意：

尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

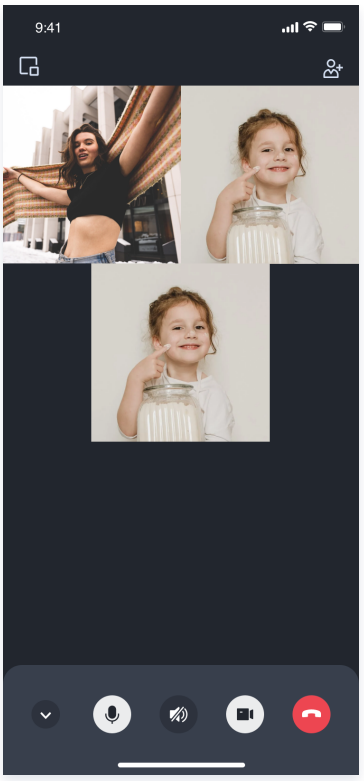
2. 拨打电话



React Native

最近更新时间：2024-12-13 16:58:32

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

1v1 视频通话	群组通话
	

步骤 1: 下载 Demo

1. 从 github 下载源码，或者直接在命令行运行以下命令：

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

2. 进入 `./TUICallKit/ReactNative` 目录，依赖 TUICallKit 组件。

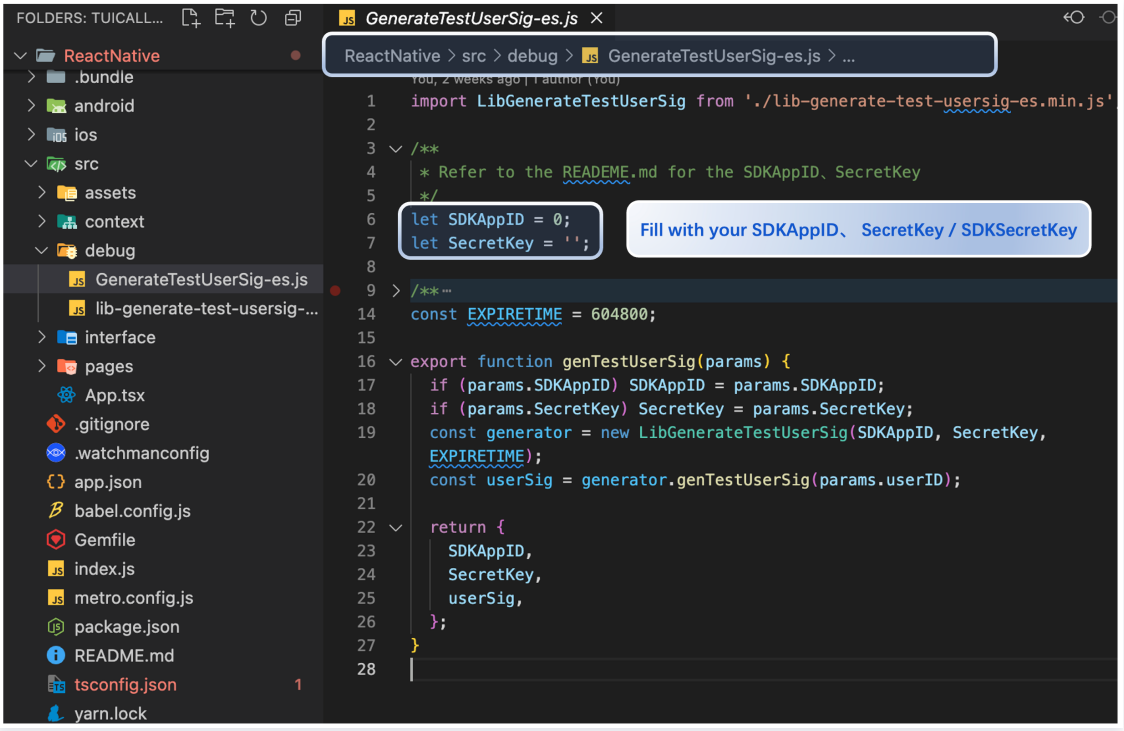
```
cd ./TUICallKit/ReactNative
yarn install
```

步骤 2: 配置 Demo

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



2. 打开 `TUICallKit/ReactNative/src/debug/GenerateTestUserSig-es.js` 文件，填入 `SDKAppID` 和 `SDKSecretKey` 。



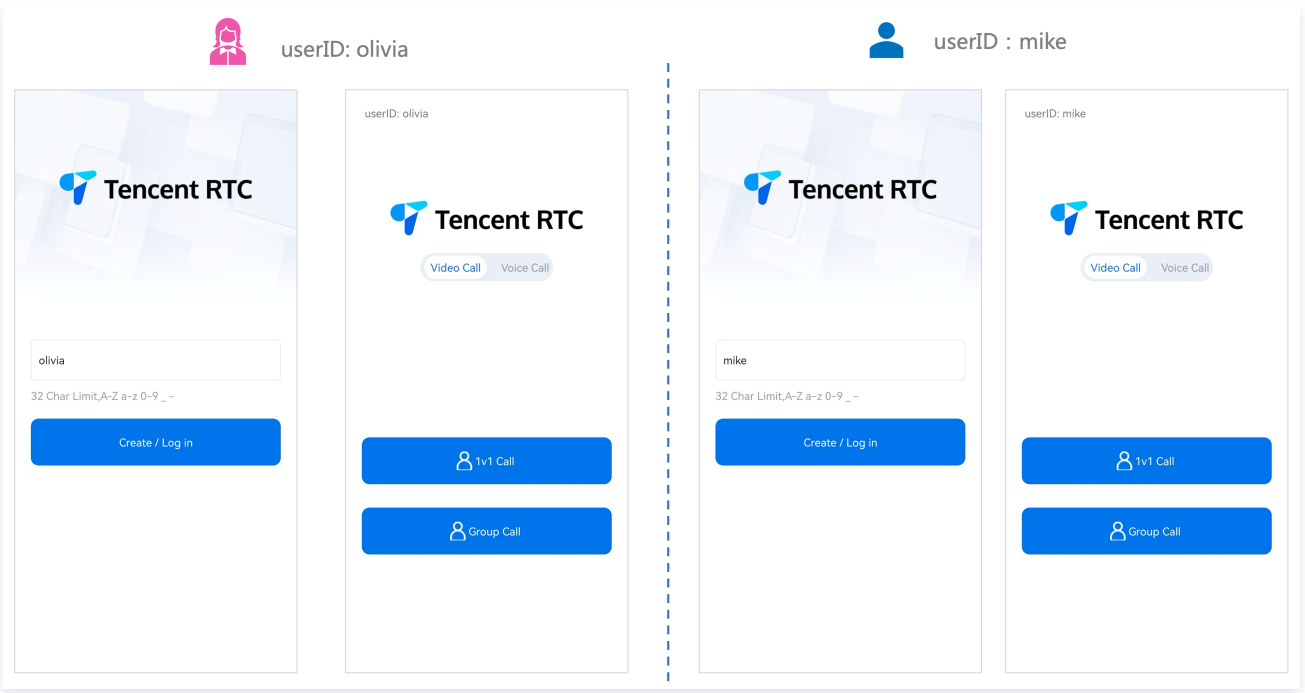
步骤 3: 跑通 Demo

```
# TUICallKit/ReactNative
yarn start
```

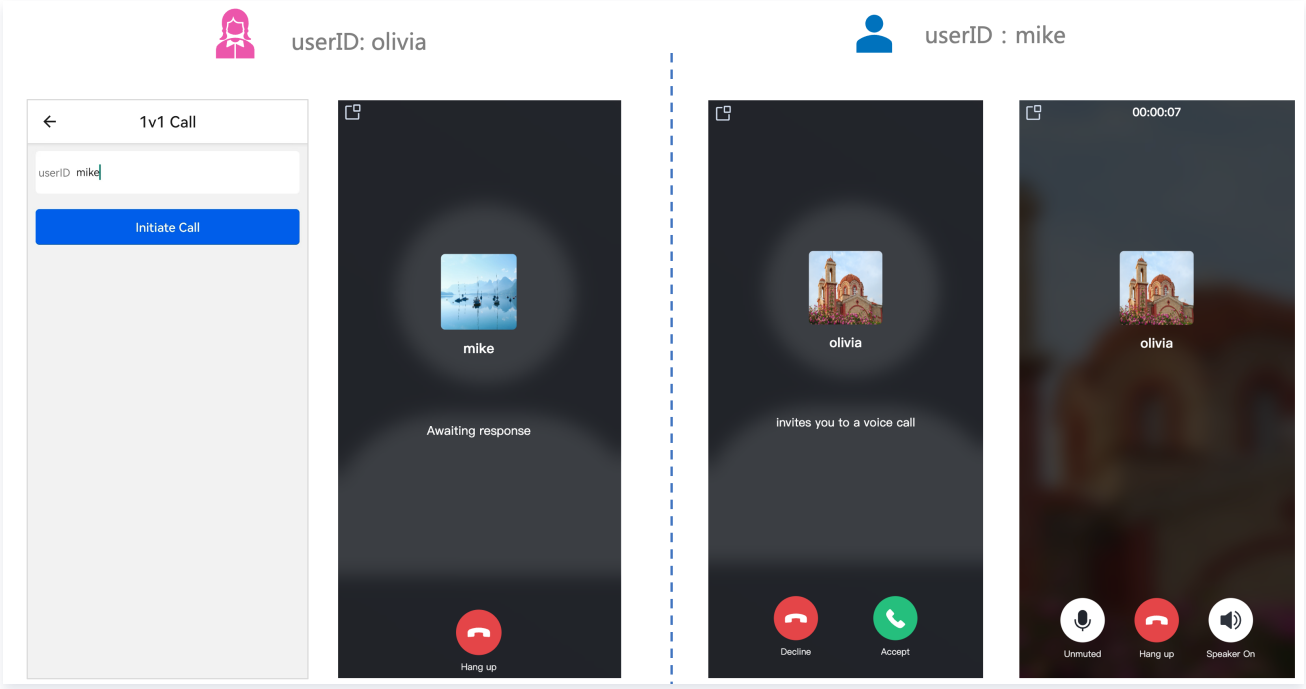
步骤 4: 拨打第一个电话

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

- 登录/注册 userID（由您定义）。



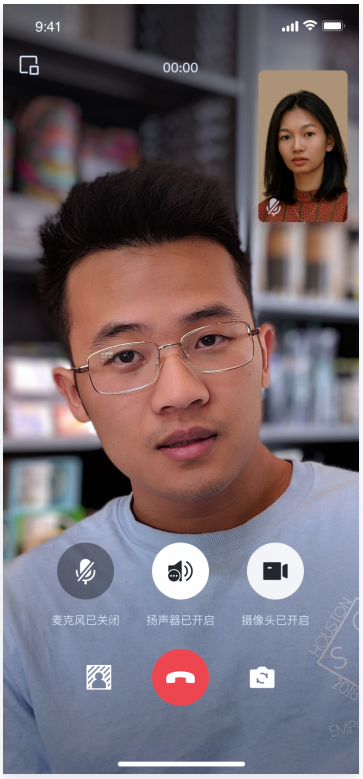
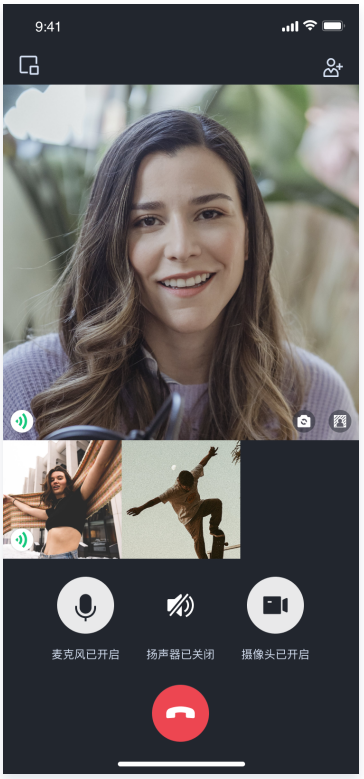
- 输入被叫方的 userID 发起通话。



快速接入（TUICallKit）
Android

最近更新时间：2025-06-23 18:11:32

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在 10 分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

1v1 视频通话	群组通话
	

环境准备

- Android 5.0（SDK API Level 21）及以上版本。
- Gradle 4.2.1 及以上的版本。
- Android 5.0 及以上的手机设备。

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤详见 开通服务（TUICallKit）。开通服务后，请记录 `SDKAppID` 和 `SDKSecretKey`，在后续的步骤中会用到。

步骤二：下载并导入组件

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



步骤三：工程配置

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

setting.gradle.kts

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

settings.gradle

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

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

build.gradle.kts

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

build.gradle

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

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

3. 由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单，因此需要您在 app 目录下的 `proguard-rules.pro` 文件末尾添加如下代码。添加完后，点击右上角的 **Sync Now**，同步代码。

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

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

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

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

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

步骤四：登录 TUI 组件

在您的项目中添加如下代码，它的作用是通过调用 TUICore 中的相关接口完成 TUI 组件的登录。这一步骤至关重要，只有在成功登录之后，您才能正常使用 TUICallKit 提供的各项功能。

Kotlin

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

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

        // begin
        val userId = "denny"      // 请替换为您的UserId
        val sdkAppId = 0          // 请替换为第一步在控制台得到的SDKAppID
        val secretKey = "****"    // 请替换为第一步在控制台得到的SecretKey

        val userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secretKey)

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

Java

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

public class MainActivity extends AppCompatActivity {
    @Override
```

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

        //begin
        String userId = "denny";      // 请替换为您的UserId
        int sdkAppId = 0;             // 请替换为第一步在控制台得到的SDKAppID
        String secretKey = "****";    // 请替换为第一步在控制台得到的SecretKey

        String userSig = GenerateTestUserSig.genTestUserSig(userId, sdkAppId, secretKey);

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

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

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

说明：

- **开发环境：**如果您正在本地开发调试阶段，可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 userSig。该方法中 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。
- **生产环境：**如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤五：拨打您的第一通电话

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

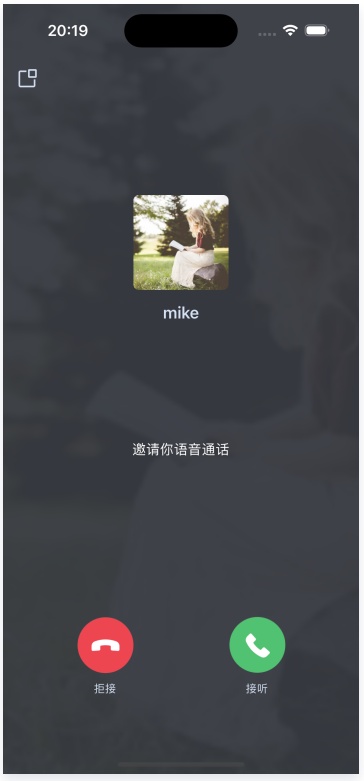
Kotlin

```
import com.tencent.cloud.tuikit.engine.call.TUICallDefine
import com.tencent.qcloud.tuikit.tuicallkit.TUICallKit

val list = mutableListOf<String>()
list.add("mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null, null)
```

Java

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

```
List<String> list = new ArrayList<>();
list.add("mike");
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null, null);
```

主叫方发起音频通话	被叫方收到音频通话请求
	

更多特性

- 设置昵称、头像
- 界面定制
- 离线推送
- 群组通话
- 悬浮窗
- 美颜特效
- 自定义铃声
- 监听通话状态
- 云端录制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

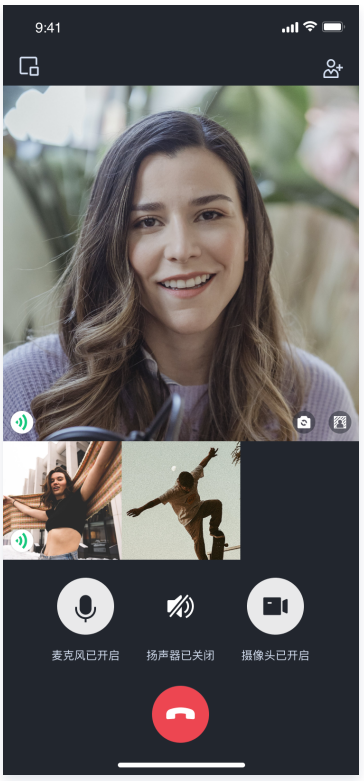
交流与反馈

- 如果您在使用过程中，有什么建议或者意见，可以在这里反馈： TUICallKit 产品反馈问卷 ，感谢您的反馈。
- 如果您是开发者，也欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao ，进行技术交流和产品沟通。

iOS

最近更新时间：2025-06-26 10:31:51

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在 10 分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

1v1 视频通话	群组通话
	

环境准备

- Xcode 13 及以上。
- iOS 13.0 及以上。
- CocoaPods 环境安装，单击 查看 。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

步骤一：开通服务

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

步骤二：导入组件

使用 CocoaPods 导入组件，如果您遇到问题，请先参见 环境准备 。导入组件具体步骤如下：

1. 请在您的 `Podfile` 文件中添加 `pod 'TUICallKit_Swift'` 依赖，建议指定 `Subspec` 为 `Professional`，如果您遇到任何问题，请参见 Example 工程。

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

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

```
pod init
```

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

```
pod install
```

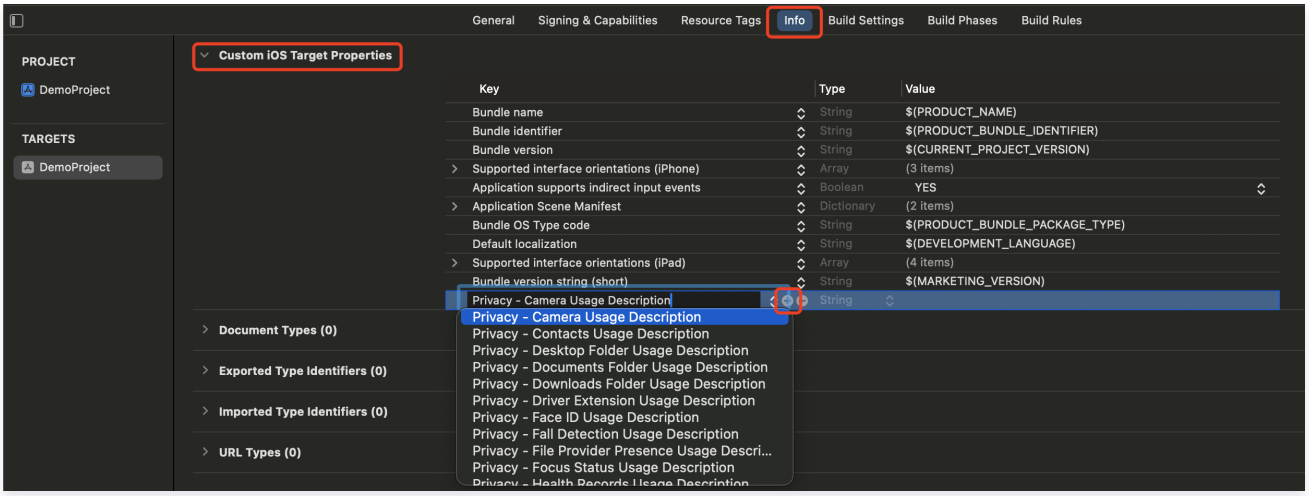
> **说明：**
> 如果无法安装 TUICallKit 最新版本，可以先删除 **Podfile.lock** 和 **Pods**。然后执行以下命令更新本地的 CocoaPods 仓库列表。
>
> ```
> pod repo update
> ```
>
> 之后执行以下命令，更新组件库的 Pod 版本。
>
> ```
> pod update
> ```

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

步骤三：工程配置

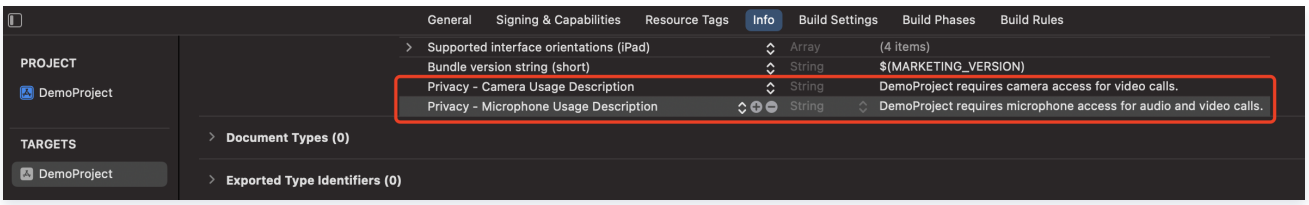
使用音视频功能，需要授权摄像头和麦克风的使用权限，请根据实际项目需要，设置项目所需权限。

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



2. 单击 +，添加摄像头和麦克风权限。

- `Privacy - Camera Usage Description`
- `Privacy - Microphone Usage Description`



步骤四：登录 TUI 组件

在您的项目中添加如下代码，它的作用是通过调用 TUICore 中的相关接口完成 TUI 组件的登录。这一步骤至关重要，只有在成功登录之后，您才能正常使用 TUICallKit 提供的各项功能。

Swift

```
import TUICore
import TUICallKit_Swift

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

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

    let userSig = GenerateTestUserSig.genTestUserSig(userID: userID, sdkAppID: sdkAppID, secretKey:
secretKey)

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

    return true
}
```

Objective-C

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

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {

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

    NSString *userSig = [GenerateTestUserSig genTestUserSigWithUserID:userID sdkAppID:sdkAppID
secretKey:secretKey];

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

    return YES;
}
```

参数	类型	说明
userID	String	客户根据自己的业务自定义用户 ID，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
sdkAppID	Int32	在 实时音视频 TRTC 控制台 创建的音视频应用的唯一标识 SDKAppID。

secretKey	String	在 实时音视频 TRTC 控制台 创建的音视频应用的 SDKSecretKey。
userSig	String	一种安全保护签名，用于对用户进行登录鉴权认证，确认用户是否真实，阻止恶意攻击者盗用您的云服务使用权。

注意：

- **开发环境：**如果您处于本地开发调试阶段，可以使用本地 GenerateTestUserSig.genTestSig 函数生成 userSig。但请注意，该方法中的 secretKey 容易被反编译逆向破解。一旦密钥泄露，攻击者可能盗用您的腾讯云流量。
- **生产环境：**如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤五：拨打您的第一通电话

通过调用 TUICallKit 的 calls 函数并指定通话类型和被叫方的 userId，就可以发起语音或者视频通话。

Swift

```
import TUICallKit_Swift
import RTCRoomEngine

// 发起1对1语音通话(假设 userId 为 mike)
TUICallKit.createInstance().calls(userIdList: ["mike"], callMediaType: .audio, params: nil) {

} fail: { code, message in

}
```

Objective-C

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

// 发起1对1语音通话(假设 userId 为 mike)
[[TUICallKit createInstance] calls:@[@"mike"] callMediaType:TUICallMediaTypeAudio params:NULL succ:^{
} fail:^(int code, NSString * _Nullable errMsg) {
}];
```

主叫方	被叫方





更多特性

- 界面定制
- 离线推送
- 多人通话
- 悬浮窗
- 美颜特效
- 自定义铃声
- 监听通话状态
- 云端录制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

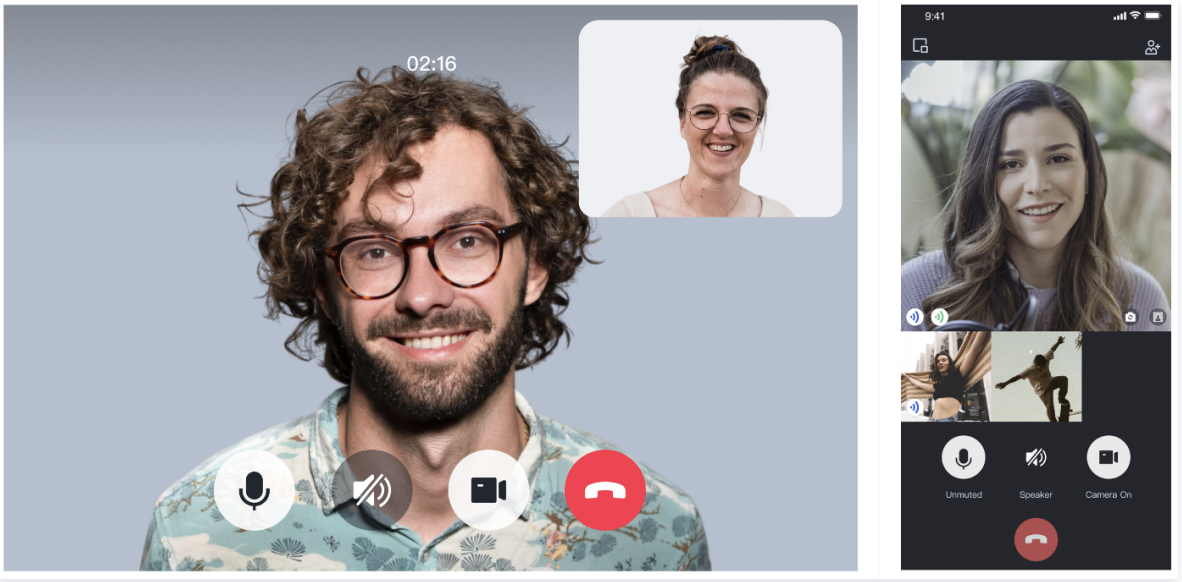
交流与反馈

- 如果您在使用过程中，有什么建议或者意见，可以在这里反馈： TUICallKit 产品反馈问卷 ，感谢您的反馈。
- 如果您是开发者，也欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao ，进行技术交流和产品沟通。

Web&H5 (Vue2/Vue3)

最近更新时间：2025-06-26 10:31:51

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在10分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。



环境准备

- Node.js version 16+.
- Modern browser, supporting WebRTC APIs.
- **npm 包集成**
  - Vue3 开发环境，集成 @tencentcloud/call-uikit-vue NPM 包。
  - Vue2.7 开发环境：集成 @tencentcloud/call-uikit-vue2 NPM 包。
  - Vue2.6 开发环境：集成 @tencentcloud/call-uikit-vue2.6 + @vue/composition-api NPM 包。
- **源码集成**
  - Vue3 + TypeScript 开发环境：从 @tencentcloud/call-uikit-vue NPM 包拷贝源码。
  - Vue2.7 + TypeScript 开发环境：从 @tencentcloud/call-uikit-vue2 NPM 包拷贝源码。
  - Vue2.6 + TypeScript 开发环境：从 @tencentcloud/call-uikit-vue2.6 NPM 包拷贝源码。
    - 需要：Vue2.6 + unplugin-vue2-script-setup + @vue/composition-api 。

> **注意：**
> 1. HBuilderX 中选择 vue2 时，使用的是 vue2.6，因此需要使用：@tencentcloud/call-uikit-vue2.6 。
> 2. HBuilderX 中选择 vue3 时，使用的是 vue3，因此需要使用：@tencentcloud/call-uikit-vue 。

步骤一：开通服务

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

步骤二：下载 TUICallKit 组件

推荐使用 NPM 集成，如果您需要采用源码集成，请参见 TUICallKit 源码集成 。

> **说明：**
> 如果您的工程同时使用了 **TUIChat UIKit**，请直接移步 TUIKit 接入音视频通话 进行接入。

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

Vue3

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

Vue2.7

```
npm install @tencentcloud/call-uikit-vue2
```

Vue2.6

```
npm install @tencentcloud/call-uikit-vue2.6 @vue/composition-api
```

如果是源码集成还需要安装 unplugin-vue2-script-setup 。

```
npm i -D unplugin-vue2-script-setup
```

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

Vue3

MacOS 端

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

Windows 端

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

Vue2.7

MacOS 端

```
cp -r node_modules/@tencentcloud/call-uikit-vue2/debug ./src
```

Windows 端

```
xcopy node_modules\@tencentcloud\call-uikit-vue2\debug .\src\debug /i /e
```

Vue2.6

MacOS 端

```
cp -r node_modules/@tencentcloud/call-uikit-vue2.6/debug ./src
```

Windows 端

```
xcopy node_modules\@tencentcloud\call-uikit-vue2.6\debug .\src\debug /i /e
```

步骤三：初始化 TUICallKit 组件

说明：
TUICallKit 组件需要放到一个 dom 节点里，用于控制 TUICallKit 的位置、宽高等样式。

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

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

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

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

Vue3

您可以选择在 `src/App.vue` **文件引入示例代码，示例代码采用的是 Vue3** `Composition API` **写法。**

```
import { ref } from 'vue';
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-vue";
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to Step 3

const SDKAppID = 0;       // TODO: Replace with your SDKAppID (Notice: SDKAppID is of type number)
const SecretKey = 'xx';   // TODO: Replace with your SecretKey

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

//【2】Initialize the TUICallKit component
const init = async () => {
  const { userSig } = GenerateTestUserSig.genTestUserSig({
    userID: callerUserID.value,
    SDKAppID,
    SecretKey: SecretKey,
  });
  await TUICallKitServer.init({
    userID: callerUserID.value,
    userSig,
    SDKAppID,
    // tim: this.tim     // 如果工程中已有 tim 实例，需在此处传入
  });
  alert('TUICallKit init succeed');
}
</script>
```

Vue2.7

```
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-vue2";
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es";

export default {
  name: 'App',
  data() {
    return {
      callerUserID: '',
      calleeUserID: '',
      SDKAppID: 0,      // TODO: Replace with your SDKAppID
      SecretKey: 'xx',  // TODO: Replace with your SecretKey
    };
  },
  components: {
    TUICallKit
  },
  methods: {
    //【2】Initialize the TUICallKit component
    async init() {
      try {
```

```
        const { userSig } = GenerateTestUserSig.genTestUserSig({
          userID: this.callerUserID,
          SDKAppID: Number(this.SDKAppID),
          SecretKey: this.SecretKey,
        });
        await TUICallKitServer.init({
          SDKAppID: Number(this.SDKAppID),
          userID: this.callerUserID,
          userSig,
          // tim: this.tim     // 如果工程中已有 tim 实例，需在此处传入
        });
        alert("[TUICallKit] Initialization succeeds.");
      } catch (error) {
        alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
      }
    },
  }
}
```

Vue2.6

1. `main.ts` 文件 注册 @vue/composition-api 。

main.ts

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

2. 将下面代码直接复制引用在 `App.vue` 文件中。

```
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-vue2.6";
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es";

export default {
  name: 'App',
  data() {
    return {
      callerUserID: '',
      calleeUserID: '',
      SDKAppID: 0,      // TODO: Replace with your SDKAppID
      SecretKey: 'xx',  // TODO: Replace with your SecretKey
    };
  },
  components: {
    TUICallKit
  },
  methods: {
    //【2】Initialize the TUICallKit component
    async init() {
      try {
        const { userSig } = GenerateTestUserSig.genTestUserSig({
          userID: this.callerUserID,
          SDKAppID: Number(this.SDKAppID),
```

```
          SecretKey: this.SecretKey,
        });
        await TUICallKitServer.init({
          SDKAppID: Number(this.SDKAppID),
          userID: this.callerUserID,
          userSig,
          // tim: this.tim      // 如果工程中已有 tim 实例，需在此处传入
        });
        alert("[TUICallKit] Initialization succeeds.");
      } catch (error) {
        alert(`[TUICallKit] Initialization failed. Reason: ${error}`);
      }
    },
  }
}
```

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

userSig 说明：

- **开发环境**：如果您正在本地跑通 Demo、开发调试，可以采用 debug 文件中的 `genTestUserSig`（参考步骤3.2）函数生成 userSig。该方法中 SDKSecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
- **生产环境**：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤四：拨打您的第一通电话

1. 调用 TUICallKitServer.calls API 拨打通话。

Vue3

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

Vue2.7

```
//【3】Make a 1v1 video call
async call () {
  await TUICallKitServer.calls({
    userIDList: [this.calleeUserID],
    type: TUICallType.VIDEO_CALL,
  });
```

```
}
```

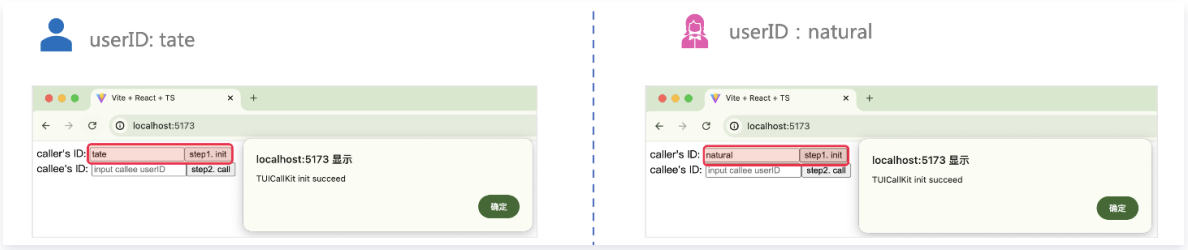
Vue2.6

```
//【3】Make a 1v1 video call
async call () {
  await TUICallKitServer.calls({
    userIDList: [this.calleeUserID],
    type: TUICallType.VIDEO_CALL,
  });
}
```

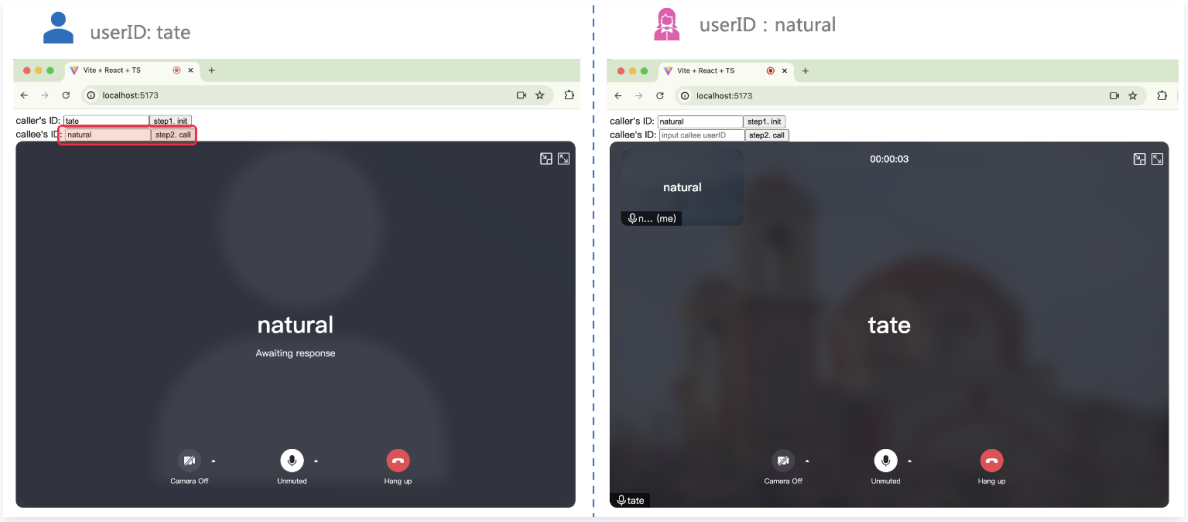
2. 运行项目。

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

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



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



更多特性

- 设置昵称、头像
- 群组通话
- 悬浮窗
- 自定义铃声
- 监听通话状态、组件回调事件
- 设置分辨率、填充模式
- 界面定制

常见问题

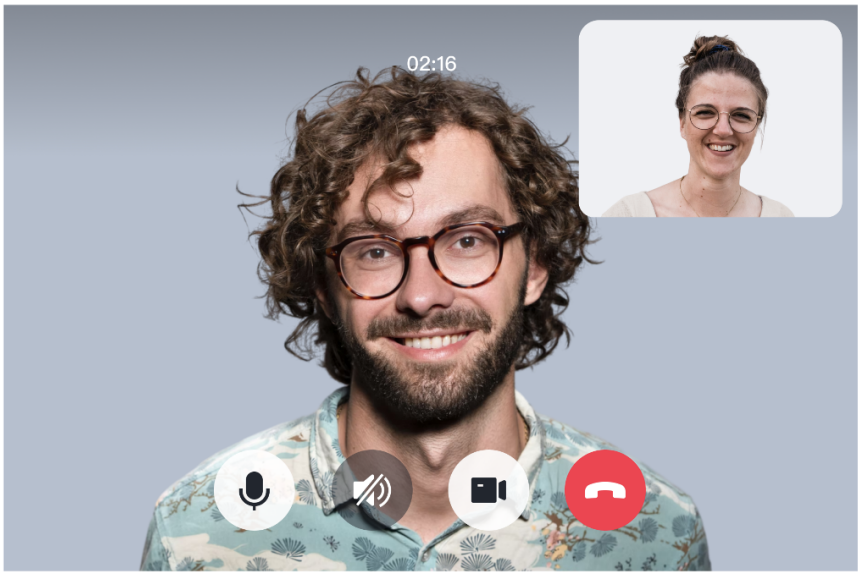
- 如果您的接入和使用中遇到问题，请参见 常见问题 。
- 了解更多详情，请到 腾讯云通信官方社群 进行咨询和反馈。

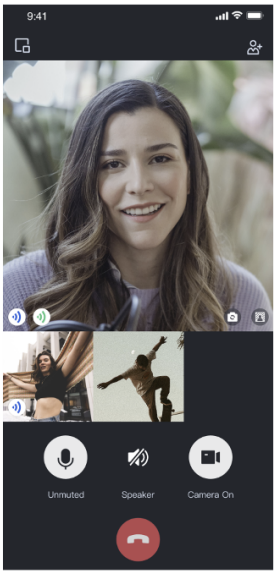
版权所有：腾讯云计算（北京）有限责任公司

Web&H5 (React)

最近更新时间：2025-06-30 20:45:42

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在10分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。





环境准备

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

步骤一：开通服务

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

步骤二：下载 TUICallKit 组件

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

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

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

MacOS

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

Windows

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

步骤三：初始化 TUICallKit 组件

您可以选择在 `/src/App.tsx` 文件引入示例代码。

1. 引入 call-uikit 相关 API 对象。

```
import { useState } from 'react';
```

```
import { TUICallKit, TUICallKitServer, TUICallType } from "@tencentcloud/call-uikit-react";
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to Step 3
```

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

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

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

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

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

const [callerUserID, setCallerUserID] = useState('');
const [calleeUserID, setCalleeUserID] = useState('');

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

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

userSig 说明：

- **开发环境**：如果您正在本地跑通 Demo、开发调试，可以采用 `debug` 文件中的 `genTestUserSig`（参考步骤3.2）函数生成 userSig。该方法中 SDKSecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
- **生产环境**：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤四：拨打您的第一通电话

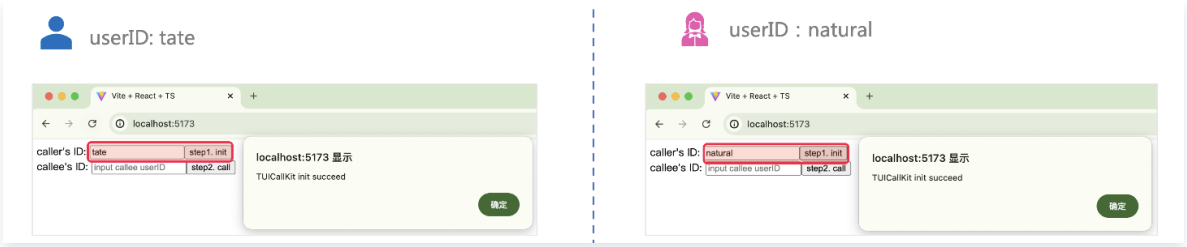
1. 调用 TUICallKitServer.calls API 拨打通话。

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

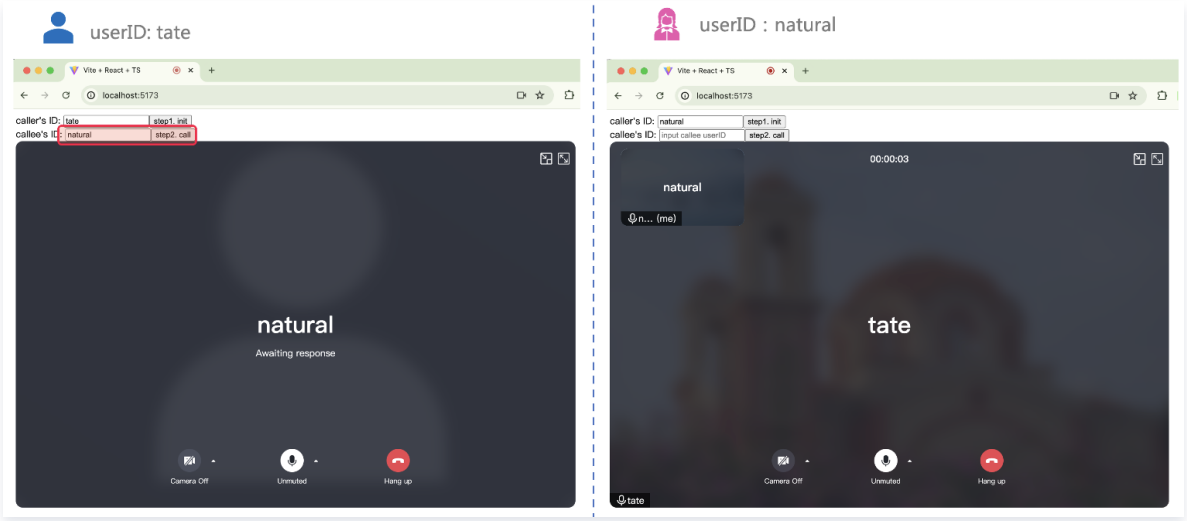
2. 运行项目。

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

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



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



更多特性

- 设置昵称、头像
- 群组通话
- 悬浮窗
- 自定义铃声
- 监听通话状态、组件回调事件

- 设置分辨率、填充模式
- 界面定制

常见问题

- 如果您的接入和使用中遇到问题，请参见 常见问题 。
- 了解更多详情您可 腾讯云通信官方社群 进行咨询和反馈。

uni-app（客户端）

最近更新时间：2025-02-18 19:10:02

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在10分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。



TUICallKit Demo 体验

- TUICallKit 插件地址：TUICallKit 插件链接。
- 如果您想要直接跑通一个新工程，请直接阅读 uni-app Demo 快速跑通。

开发环境要求

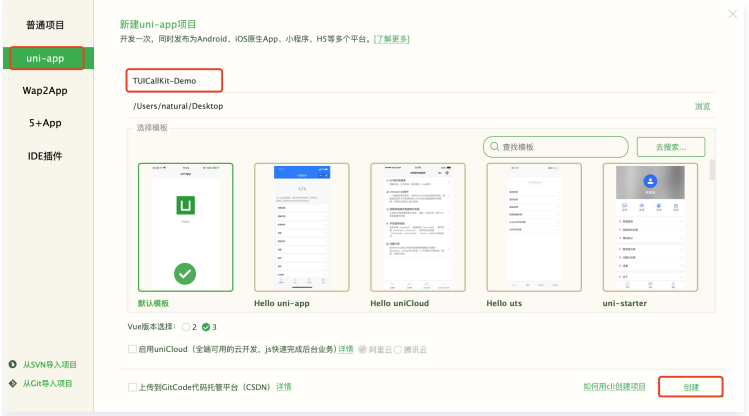
- HbuilderX 版本要求：HbuilderX 版本 ≥ 3.94。
- 插件调试说明：原生插件暂不支持模拟器调试。
- iOS 设备要求：iOS 系统 ≥ 9.0，支持音视频通话的真机设备。
- Android 设备要求：Android 系统 ≥ 5.0（SDK API Level 21），支持音视频的真机设备，允许 USB 调试。

步骤一：开通服务

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

步骤二：创建 uni-app 项目

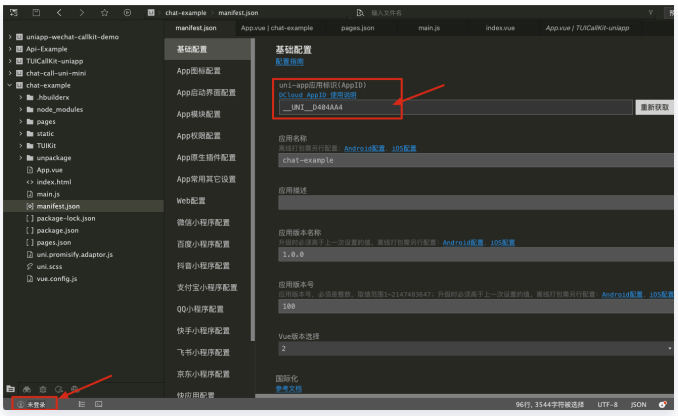
打开 Hbuilderx 开发工具，点击新建 uni-app 项目：项目名称 (TUICallKit-Demo)。



步骤三：下载并导入 TUICallKit 插件

1. 创建项目，生成 uin-app 应用标识（AppID）

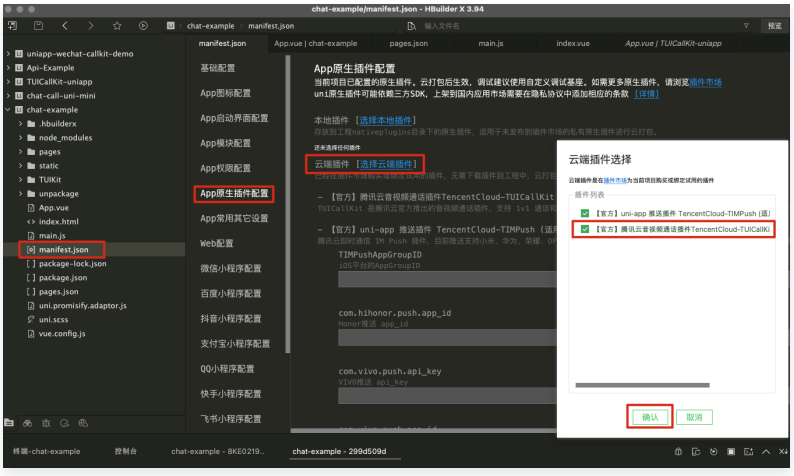
打开 HbuilderX，点击左下角登录 uni-app 账号（无账号请先注册）。完成登录后，点击项目的 manifest.json 文件，生成生成 uni-app 应用标识（AppID）。



2. 访问 TencentCloud-TUICallKit 插件，在插件详情页中购买插件（**免费**），购买插件时选择对应的 AppID，绑定正确的包名。



3. 在 `TUICallKit-Demo` 项目中导入插件。



步骤四：使用 TUICallKit 插件

1. 在 `TUICallKit-Demo/pages/index/index.vue` 中引入下面代码，该段代码功能：登录 TUICallKit 组件、拨打 1v1 视频通话。

```
<template>
    <view class="container">
        <input type="text" v-model="inputID" :placeholder=" isLogin ? 'please enter a caller userID'
: 'please enter your login userID' " />
        <text v-show="isLogin"> your userID: {{ userID }} </text>
        <button v-show="!isLogin" @click="handleLogin"> Login </button>
        <button v-show="isLogin" @click="handleCall"> start call </button>
    </view>
</template>
```

```
<script>
    const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit'); //【1】import TUICallKit
plugin
    uni.$TUICallKit = TUICallKit;
    import { genTestUserSig } from '../../debug/GenerateTestUserSig.js'
    export default {
        data() {
            return {
                inputID: '',
                isLogin: false,
                userID: '',
            }
        },
        methods: {
            handleLogin() {
                this.userID = this.inputID;
                const { userSig, sdkAppID: SDKAppID } = genTestUserSig(this.userID);
                const loginParams = { SDKAppID, userID: this.userID, userSig };    // apply SDKAppID、
userSig
                //【2】Login
                uni.$TUICallKit.login( loginParams, res => {
                        if (res.code === 0) {
                            this.isLogin = true;
                            this.inputID = '';
                            console.log('[TUICallKit] login success.');
                        } else {
                            console.error('[TUICallKit] login failed, failed message = ', res.msg,
params);
                        }
                    }
                );
            },
            handleCall() {
                try {
                    const callParams = {
                        userIDList: [this.inputID],
                        callMediaType: 2,    // 1 -- audio call, 2 -- video call
                        callParams: { roomID: 234, strRoomID: '2323423', timeout:30 },
                    };
                    //【3】start 1v1 video call
                    uni.$TUICallKit.calls( callParams, res => {
                            console.log('[TUICallKit] call params: ', JSON.stringify(res));
                        }
                    );
                    this.inputID = '';
                } catch (error) {
                    console.log('[TUICallKit] call error: ', error);
                }
            }
        }
    }
</script>
<style>
.container {
    margin: 30px;
}
.container input {
    height: 50px;
    border: 1px solid;
```

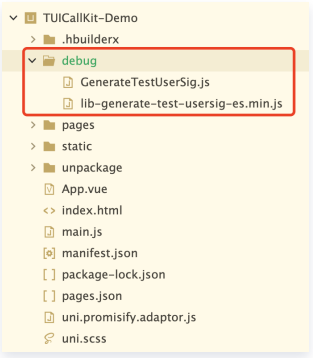
```
}
.container button {
    margin-top: 30px;
}
</style>
```

2. 填写 SDKAppID、SecretKey、userSig 参数。

客户端生成 userSig

由于 UserSig 有时效性，如果您需要长期使用 TUICallKit，推荐您采用该方法。

1. **单击下载 debug 文件夹，将 debug 目录复制到您的项目，如下图所示：**



2. **填写 `TUICallKit-Demo/debug/GenerateTestUserSig.js` 文件的 `SDKAppID` 、 `SecretKey` （参见 开通服务 ）**



控制台生成 userSig

如果您想要快速体验 TUICallKit，您可以通过控制台中的 UserSig生成&校验 生成一个临时可用的 UserSig。



如果您采用的是**控制台生成**，那么需要在 `TUICallKit-Demo/pages/index/index.vue` 文件中，赋值这里的 `SDKAppID`、`userSig` 。



步骤五：拨打您的第一通电话

1. 制作自定义调试基座，请选择**传统打包**方式进行打包。

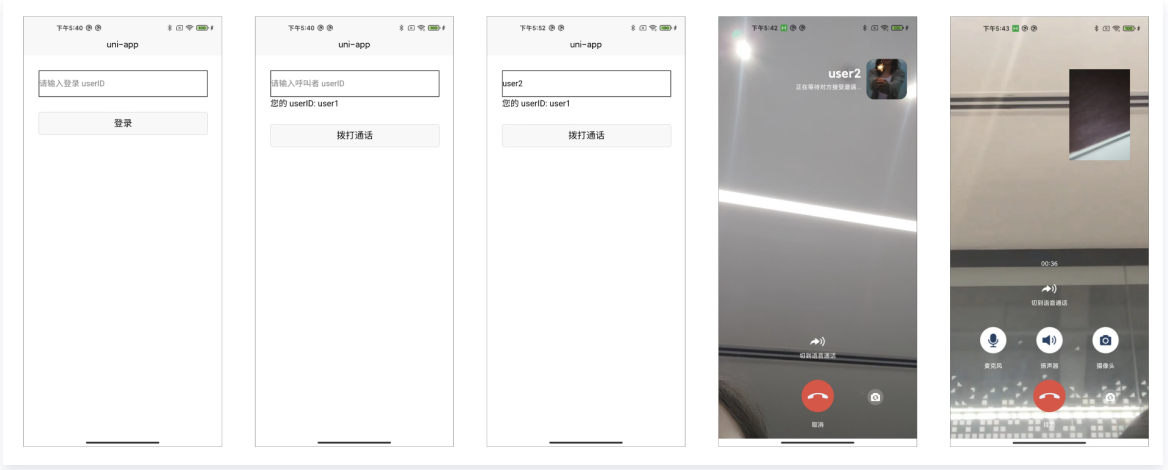




2. 自定义调试基座成功后，**使用自定义基座运行项目**。



3. 拨打 1v1 视频通话具体效果如图所示。



更多特性

- 设置昵称、头像
- 群组通话
- 悬浮窗
- 自定义铃声
- 监听通话状态
- 离线唤醒

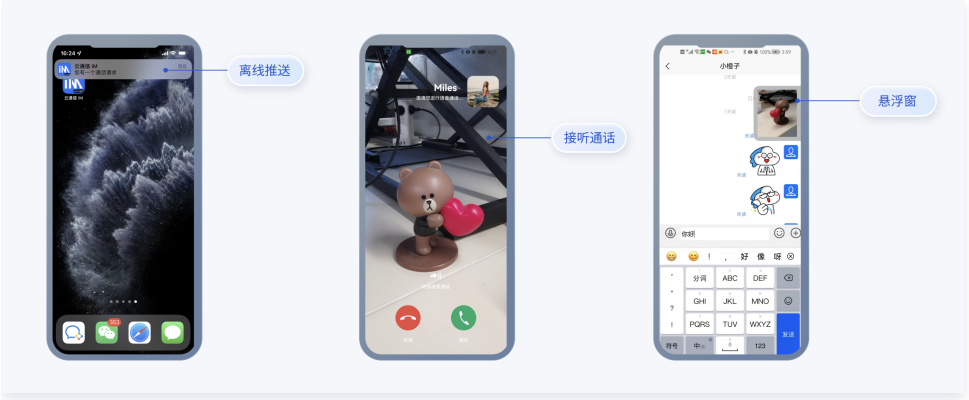
常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

相关案例-在线客服场景

我们提供了**在线客服场景**的相关源码，建议您 下载在线客服场景 并集成体验。该场景提供了示例客服群 + 示例好友的基础模板，实现功能包括：

- 支持发送文本消息、图片消息、语音消息、视频消息等常见消息。
- 支持双人语音、视频通话功能。
- 支持创建群聊会话、群成员管理等。



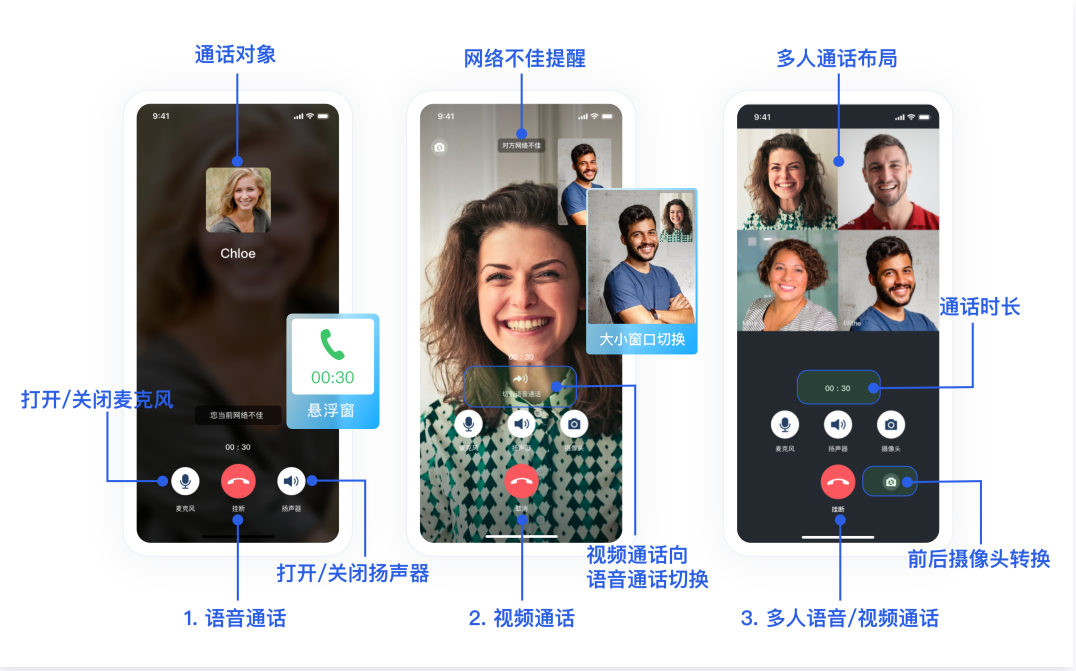
技术咨询

了解更多详情您可加入 腾讯云通信官方社群 进行咨询和反馈。

uni-app（小程序）

最近更新时间：2025-06-12 16:09:02

本文将介绍如何快速完成 TUICallKit 组件的接入，跟随本文档，您将在半小时内得到一个包含完备 UI 界面的视频通话小程序。基本功能如下图所示：



> **注意：**
> cli 脚手架工具创建的项目集成，集成指引请点击 uni-app（小程序）脚手架 查看。

小程序 Demo 体验

- 如果您想要直接体验音视频通话小程序，单击 Demo 体验，扫描小程序二维码。
- 如果您想要直接跑通一个新工程，请直接阅读 uni-app demo 快速跑通 。
- 如果您想要亲自集成 TUICallKit 组件，搭建一个音视频通话小程序，请跟随本文档。

开发环境要求

- 微信 App iOS 最低版本要求：8.0.40。
- 微信 App Android 最低版本要求：8.0.40。
- 小程序基础库最低版本要求：2.10.0。

> **警告：**
> - 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，请使用企业小程序账号申请相关权限进行开发。
> - 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。

小程序开发准备

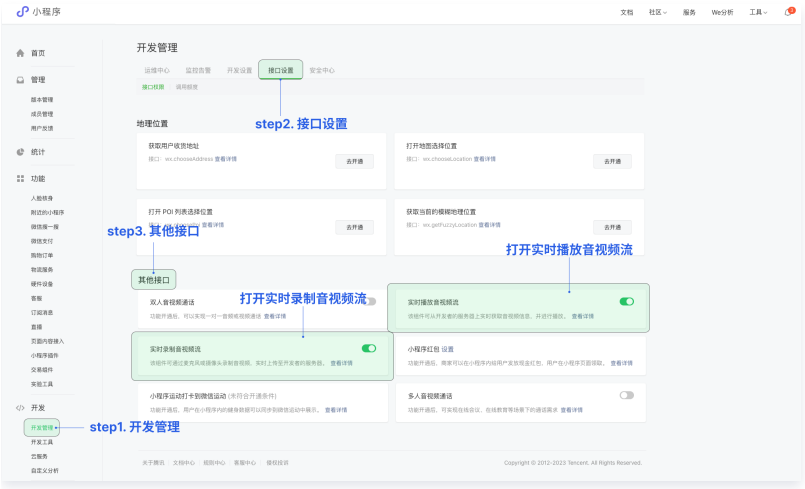
步骤一：开通企业类小程序

小程序推拉流标签不支持个人小程序，只支持企业类小程序。需要在 注册 时填写主体类型为企业，如下图所示：



步骤二：在小程序控制台开启实时音视频接口

- 小程序推拉流标签使用权限暂时只开放给有限类目，具体支持类目参见该地址 。
- 符合类目要求的小程序，需要在 微信公众平台 > 开发 > 开发管理 > 接口设置中自助开通该组件权限。



步骤三：在小程序控制台配置域名

在 微信公众平台 > 开发 > 开发管理 > 开发设置 > 服务器域名中设置 request 合法域名 和 socket 合法域名。

- 将以下域名添加到 socket 合法域名：

域名	说明	是否必须
`wss://${SDKAppID}w4c.my-imcloud.com`	v3.4.6起，SDK 支持独立域名，可更好地保障服务稳定性。 例如您的 SDKAppID 是 1400xxxxxx，则独立域名为： `wss://1400xxxxxxw4c.my-imcloud.com`	必须
`wss://wss.im.qcloud.com`	Web IM 业务域名	必须
`wss://wss.tim.qq.com`	Web IM 业务域名	必须
`wss://wssv6.im.qcloud.com`	Web IM 业务域名	必须

- 将以下域名添加到 request 合法域名：

域名	说明	是否必须

https://web.sdk.qcloud.com	Web IM 业务域名	必须
https://boce-cdn.my-imcloud.com	Web IM 业务域名	必须
https://api.im.qcloud.com	Web IM 业务域名	必须
https://events.im.qcloud.com	Web IM 业务域名	必须
https://webim.tim.qq.com	Web IM 业务域名	必须
https://wss.im.qcloud.com	Web IM 业务域名	必须
https://wss.tim.qq.com	Web IM 业务域名	必须

- 将以下域名添加到 uploadFile 合法域名：

域名	说明	是否必须
https://${SDKAppID}-cn.rich.my-imcloud.com	从 2024年9月10日起，新增应用默认分配 cos 独立域名。 例如您的 SDKAppID 是 1400xxxxxx，则 cos 独立域名为： https://1400xxxxxx-cn.rich.my-imcloud.com	必须
https://cn.rich.my-imcloud.com	文件上传域名	必须
https://cn.imrich.qcloud.com	文件上传域名	必须
https://cos.ap-shanghai.myqcloud.com	文件上传域名	必须
https://cos.ap-shanghai.tencentcos.cn	文件上传域名	必须
https://cos.ap-guangzhou.myqcloud.com	文件上传域名	必须

- 将以下域名添加到 downloadFile 合法域名：

域名	说明	是否必须
https://${SDKAppID}-cn.rich.my-imcloud.com	从 2024年9月10日起，新增应用默认分配 cos 独立域名。 例如您的 SDKAppID 是 1400xxxxxx，则 cos 独立域名为： https://1400xxxxxx-cn.rich.my-imcloud.com	必须
https://cn.rich.my-imcloud.com	文件下载域名	必须
https://cn.imrich.qcloud.com	文件下载域名	必须
https://cos.ap-shanghai.myqcloud.com	文件下载域名	必须
https://cos.ap-shanghai.tencentcos.cn	文件下载域名	必须
https://cos.ap-guangzhou.myqcloud.com	文件下载域名	必须

- 如下图所示，服务器域名配置：



TUICallKit 源码集成

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：创建 uni-app 小程序项目

1. 在 HBuilder 中创建小程序项目。



2. 在终端输入 `npm init -y`，创建 `package.json` 文件。

```
npm init -y
```

步骤三：下载并导入 TUICallKit 组件

MacOS 端

```
npm i @tencentcloud/call-uikit-wx-uniapp
```

```
mkdir -p ./TUICallKit && cp -r node_modules/@tencentcloud/call-uikit-wx-uniapp/ ./TUICallKit && cp
node_modules/@tencentcloud/call-engine-wx/RTCCallEngine.wasm.br ./static
```

Windows 端

```
npm i @tencentcloud/call-uikit-wx-uniapp
```

```
xcopy node_modules\@tencentcloud\call-uikit-wx-uniapp .\TUICallKit /i /e
xcopy node_modules\@tencentcloud\call-engine-wx\RTCCallEngine.wasm.br .\static
```

步骤四：使用精简版 IM SDK 减小体积（可选）

> **警告：**
> 精简版 IM SDK 体积降低 **550KB**，仅适用于独立集成 TUICallKit 的场景。

MacOS 端

```
npm install @tencentcloud/lite-chat && rm -rf node_modules/@tencentcloud/chat && cp -r
node_modules/@tencentcloud/lite-chat node_modules/@tencentcloud/chat
```

Windows 端

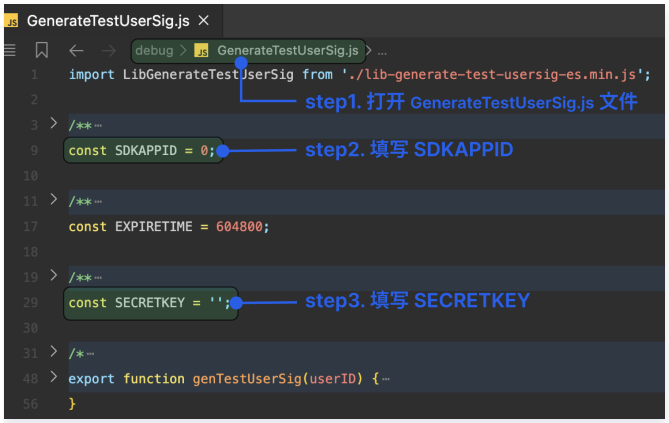
```
npm install @tencentcloud/lite-chat
```

```
rmdir /s /q node_modules\@tencentcloud\chat
```

```
xcopy node_modules\@tencentcloud\lite-chat node_modules\@tencentcloud\chat /i /e
```

步骤五：填写 SDKAPPID 以及 SECRETKEY

修改 TUICallKit/debug/`GenerateTestUserSig-es.js` 文件 的 SDKAPPID 以及 SECRETKEY。



步骤六：调用 TUICallKit 组件

说明：

下面会重点介绍各种接入方式包体积占比，请针对自身项目，选择合适的接入方式。

接入方式	主包接入	分包接入
能否使用全局监听	是	否
是否占用主包体积	是	是
Vue2 项目体积占比	约占用 1.3MB 主包体积，其中 TUICallKit/debug 目录用于生成 userSig，生产环境推荐在服务端生成，可以节省约 150kb 主包体积。	约占用 249 kb主包体积。
Vue3 项目体积占比	约占用 1.3MB 主包体积，其中 TUICallKit/debug 目录用于生成 userSig，生产环境推荐在服务端生成，可以节省约 150kb 主包体积。	参见 体积优化文档，约占用 249 kb主包体积。

全局监听：使小程序具有所有页面都能唤起通话页面的能力。

主包接入

注意：

主包接入的方式默认集成全局监听。

在 pages.json 文件注册全局监听页面。

```
{
    "path": "TUICallKit/src/Components/TUICallKit",
    "style": {
        "navigationBarTitleText": "uni-app"
    }
}
```

Vue3

1. 修改 pages/index/index.vue 文件。

```
<template>
  <view class="loginBox">
```

```
    <input
      class="input-box"
      v-model="userID"
      :placeholder="!isLogin ? '请输入用户ID' : '搜索用户ID'"
      placeholder-style="color:#BBBBBB;"
    />
    <view class="login">
      <button
        class="loginBtn"
        @click="!isLogin ? loginHandler() : callHandler()"
      >
        {{ !isLogin ? "登录" : "呼叫" }}
      </button>
    </view>
  </view>
</template>
<script setup>
import { ref } from "vue";
import * as GenerateTestUserSig from "../../TUICallKit/debug/GenerateTestUserSig-es.js";
// 导入 TUICallKitServer 模块，使您的应用具有全局呼叫的能力
import { TUICallKitServer } from "../../TUICallKit/src/index";
// 导入 CallManager 模块，使您的应用具有全局监听来电的能力
import { CallManager } from "../../TUICallKit/src/TUICallService/serve/callManager";
uni.CallManager = new CallManager();
let userID = ref("");
let isLogin = ref(false);
const loginHandler = async () => {
  if (!userID.value) return;
  const { userSig, SDKAppID } = GenerateTestUserSig.genTestUserSig({
    userID: userID.value,
  });
  await uni.CallManager.init({
    sdkAppID: SDKAppID,
    userID: userID.value,
    userSig: userSig,
    globalCallPagePath: "TUICallKit/src/Components/TUICallKit",
  });
  isLogin.value = true;
  userID.value = "";
};

const callHandler = async () => {
  await TUICallKitServer.calls({
    userIDList: [userID.value],
    type: 2,
  });
};
</script>
<style>
.loginBox {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

input {
  display: flex;
```

```
  font-size: 20px;
}

.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
}

.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

Vue2

1. 下载依赖。

```
npm i unplugin-vue2-script-setup
```

2. 修改 `vue.config.js` 文件。

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

3. 修改 `main.js` 文件。

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

4. 修改 pages/index/index.vue 文件。

```
<template>
<view>
  <view class="loginBox">
    <input
      class="input-box"
      v-model="userID"
      :placeholder="!isLogin ? '请输入用户ID' : '搜索用户ID'"
      placeholder-style="color:#BBBBBB;"
```

```
      />
      <view class="login">
        <button
          class="loginBtn"
          @click="!isLogin ? loginHandler() : callHandler()"
        >
          {{ !isLogin ? "登录" : "呼叫" }}
        </button>
      </view>
    </view>
  </view>
</template>
<script>
import * as GenerateTestUserSig from "../../TUICallKit/debug/GenerateTestUserSig-es.js";
// 导入 TUICallKitServer 模块，使您的应用具有全局呼叫的能力
import { TUICallKitServer } from "../../TUICallKit/src/index";
// 导入 CallManager 模块，使您的应用具有全局监听来电的能力
import { CallManager } from "../../TUICallKit/src/TUICallService/serve/callManager";
uni.CallManager = new CallManager();
export default {
  data() {
    return {
      isLogin: false,
      userID: "",
    };
  },
  methods: {
    async loginHandler() {
      if (!this.userID) return;
      const { userSig, SDKAppID } = GenerateTestUserSig.genTestUserSig({
        userID: this.userID,
      });
      await uni.CallManager.init({
        sdkAppID: SDKAppID,
        userID: this.userID,
        userSig: userSig,
        globalCallPagePath: "TUICallKit/src/Components/TUICallKit",
      });
      this.isLogin = true;
      this.userID = "";
    },

    async callHandler() {
      await TUICallKitServer.calls({
        userIDList: [this.userID],
        type: 2,
      });
    },
  },
};
</script>
<style>
.loginBox {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
```

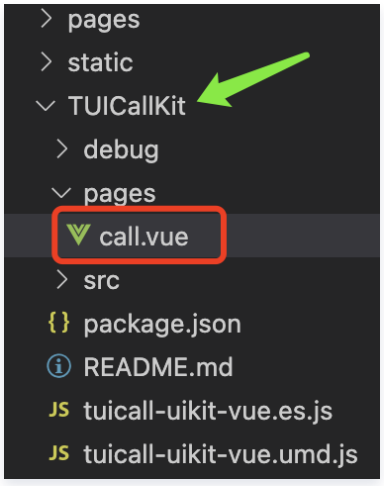
```
input {
  display: flex;
  font-size: 20px;
}

.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
}

.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

分包接入

1. 修改 manifest.json 小程序相关配置。

```
"mp-weixin": {
     "usingComponents": true,
      "optimization": {
         "subPackages": true
      }
},
```

2. 新建 /TUICallkit/pages/call.vue 目录。



3. 修改 pages.json，添加分包页面。

```
"subPackages": [
       {
         "root": "TUICallKit",
         "pages": [
```

```
            {
              "path": "pages/call",
              "style": {
                "navigationBarTitleText": "uni-app"
              }
            }
          ]
        }
  ],
```

4. 集成 TUICallKit。

Vue3 环境

修改 TUICallKit/pages/call.vue 文件。

```
<template>
  <view>
    <TUICallKit></TUICallKit>
    <view class="loginBox">
      <input
        class="input-box"
        v-model="userID"
        :placeholder="!isLogin ? '请输入用户ID' : '搜索用户ID'"
        placeholder-style="color:#BBBBBB;"
      />
      <view class="login">
        <button
          class="loginBtn"
          @click="!isLogin ? loginHandler() : callHandler()"
        >
          {{ !isLogin ? "登录" : "呼叫" }}
        </button>
      </view>
    </view>
  </view>
</template>
<script setup>
import { ref } from "vue";
import * as GenerateTestUserSig from "../debug/GenerateTestUserSig-es.js";
import { TUICallKitServer } from "../src/index";
import TUICallKit from "../src/Components/TUICallKit";
let userID = ref("");
let isLogin = ref(false);
const loginHandler = async () => {
  if (!userID.value) return;
  const { userSig, SDKAppID } = GenerateTestUserSig.genTestUserSig({
    userID: userID.value,
  });
  await TUICallKitServer.init({
    sdkAppID: SDKAppID,
    userID: userID.value,
    userSig: userSig,
  });
  isLogin.value = true;
  userID.value = "";
};
```

```
const callHandler = async () => {
  await TUICallKitServer.calls({
    userIDList: [userID.value],
    type: 2,
  });
};
</script>
<style>
.loginBox {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

input {
  display: flex;
  font-size: 20px;
}

.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
}

.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

Vue2 环境

1. 下载依赖。

```
npm i unplugin-vue2-script-setup
```

2. 修改 `vue.config.js` 文件。

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

```
  },
};
```

3. 修改 `main.js` 文件。

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

4. 修改 TUICallKit/pages/call.vue 文件。

```
<template>
<view>
  <TUICallKit></TUICallKit>
  <view class="loginBox">
    <input
      class="input-box"
      v-model="userID"
      :placeholder="!isLogin ? '请输入用户ID' : '搜索用户ID'"
      placeholder-style="color:#BBBBBB;"
    />
    <view class="login">
      <button
        class="loginBtn"
        @click="!isLogin ? loginHandler() : callHandler()"
      >
        {{ !isLogin ? "登录" : "呼叫" }}
      </button>
    </view>
  </view>
</view>
</template>
<script>
import * as GenerateTestUserSig from "../debug/GenerateTestUserSig-es.js";
import { TUICallKitServer } from "../src/index";
import TUICallKit from "../src/Components/TUICallKit.vue";
export default {
  components: {
    TUICallKit,
  },
  data() {
    return {
      isLogin: false,
      userID: "",
    };
  },
  methods: {
    async loginHandler() {
      if (!this.userID) return;
      const { userSig, SDKAppID } = GenerateTestUserSig.genTestUserSig({
        userID: this.userID,
      });
      await TUICallKitServer.init({
        sdkAppID: SDKAppID,
        userID: this.userID,
        userSig: userSig,
      });
      this.isLogin = true;
      this.userID = "";
```

```
    },

    async callHandler() {
      await TUICallKitServer.calls({
        userIDList: [this.userID],
        type: 2,
      });
    },
  },
};
</script>
<style>
.loginBox {
  margin-top: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

input {
  display: flex;
  font-size: 20px;
}

.login {
  width: 100vw;
  bottom: 5vh;
  margin: 70rpx;
}

.login button {
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
}
</style>
```

5. 修改 pages/index/index.vue 文件。

> **说明：**
> 此处的代码的作用是，主包路由跳转到 TUICallKt 分包页面，注意需要根据您的业务进行调整，此处以空项目举例。

```
<template>
  <view>
    <button @click="jumpUrl">跳转分包</button>
  </view>
</template>
<script>
export default {
  methods: {
    jumpUrl() {
      uni.navigateTo({ url: "/TUICallKit/pages/call" });
    },
  },
```

```
};
</script>
```

警告：

- 如果使用 HBuilder x 4.x 以上版本运行 Vue3 出现如下报错：

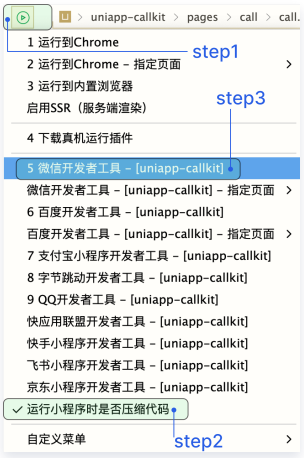
```
11:07:43.690 [plugin:uni:mp-inject] Unexpected token `@`. Expected identifier, string literal, numeric literal or [ for the computed key
11:07:43.700 at TUICallKit/src/TUICallService/serve/callManager.ts:1:0
```

- 可在 tsconfig.json 中添加相关配置，示例：

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

步骤七：运行到微信开发者工具

1. 运行到微信开发者工具。

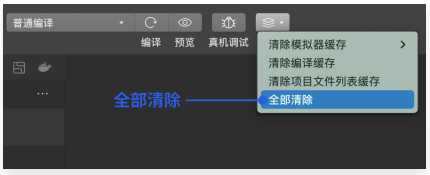


2. 请在本地设置里面勾选上“不校验合法域名、web-view (业务域名)、TLS 版本以及 HTTPS 证书”。

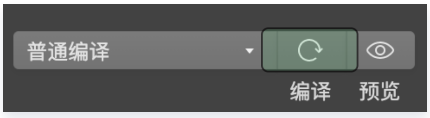


> **警告：**
> 如果不勾选该条目，则会在控制台出现如下错误。
>
> 
>

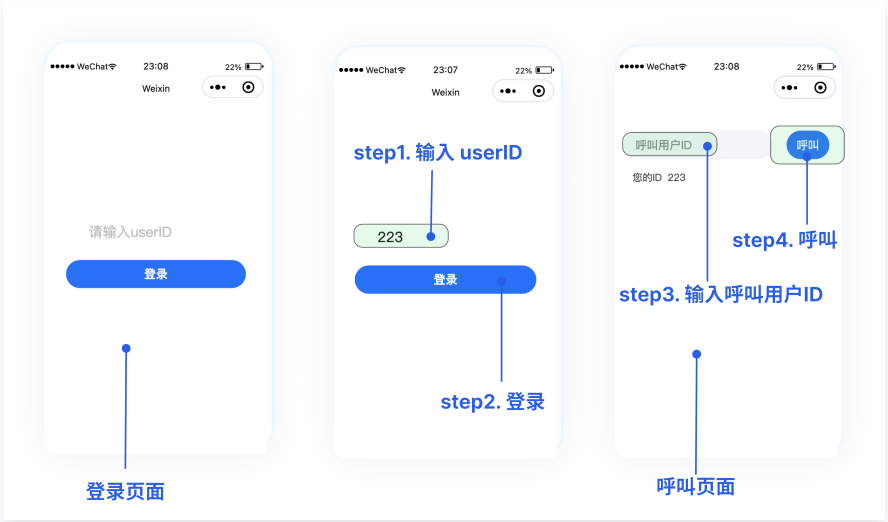
3. 单击清缓存 > 全部清除，避免开发者工具的缓存造成渲染异常。



4. 编译小程序。



5. 该项目快速集成后的预期效果图。



步骤八：拨打您的第一通电话

1. 请单击预览，扫描二维码，在真机环境使用小程序。



2. 登录后，请输入呼叫用户 ID，拨打您的第一通电话。具体效果如下图所示：



> ⚠ **注意：**
> 第一次使用小程序通话，需要获取摄像头和麦克风权限。

更多特性

- 设置昵称、头像
- 自定义铃声
- 群组通话
- 悬浮窗

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

技术咨询

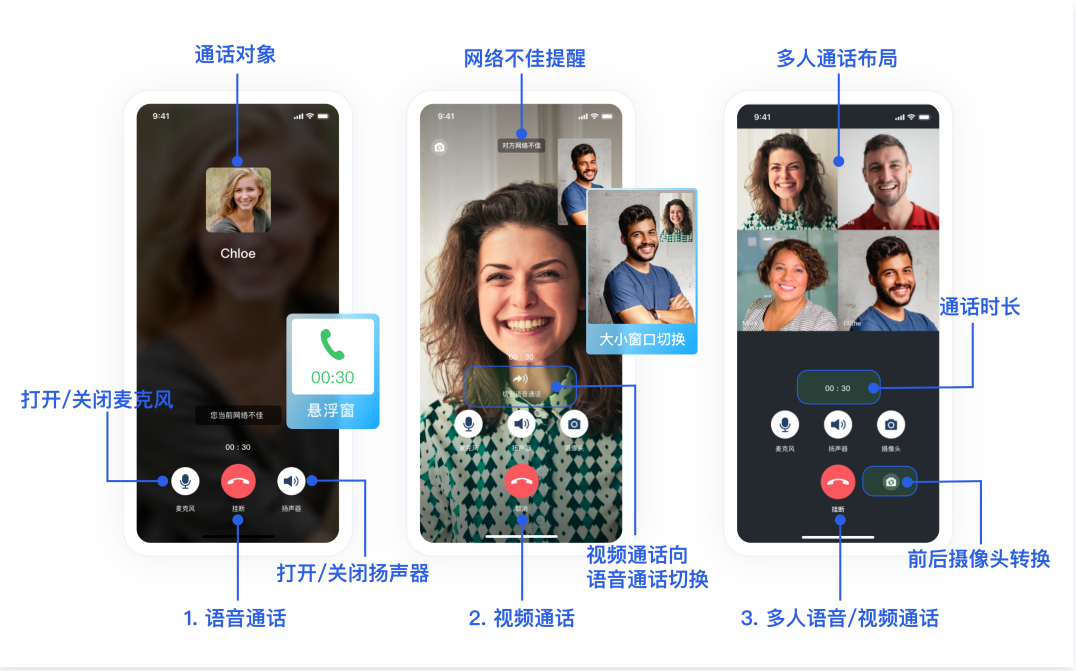
了解更多详情您可前往 腾讯云通信官方社群 进行咨询和反馈。

版权所有：腾讯云计算（北京）有限责任公司

微信小程序

最近更新时间：2025-04-02 17:40:23

本文将介绍如何快速完成 TUICallKit 组件的接入，跟随本文档，您将在半小时内得到一个包含完备 UI 界面的视频通话小程序。基本功能如下图所示：



小程序 Demo 体验

- 如果您想要直接体验音视频通话小程序，单击 Demo 体验，扫描小程序二维码。
- 如果您想要直接跑通一个新工程，请直接阅读 微信小程序 Demo 快速跑通 。

开发环境要求

- 微信 App iOS 最低版本要求：7.0.9。
- 微信 App Android 最低版本要求：7.0.8。
- 小程序基础库最低版本要求：2.10.0。

警告：

- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，请使用**企业小程序账号**申请相关权限进行开发。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在**真机**上运行体验。

小程序开发准备

步骤一：开通企业类小程序

小程序推拉流标签不支持个人小程序，只支持企业类小程序。需要在 注册 时填写**主体类型为企业**，如下图所示：



步骤二：在小程序控制台开启实时音视频接口

- 小程序推拉流标签使用权限暂时只开放给有限类目，具体支持类目参见该地址 。
- 符合类目要求的小程序，需要在 微信公众平台 > 开发 > 开发管理 > 接口设置中自助开通该组件权限。



步骤三：在小程序控制台配置域名

在 微信公众平台 > 开发 > 开发管理 > 开发设置 > 服务器域名中设置 request 合法域名 和 socket 合法域名。

- 将以下域名添加到 socket 合法域名：

域名	说明	是否必须
`wss://${SDKAppID}w4c.my-imcloud.com`	v3.4.6起，SDK 支持独立域名，可更好地保障服务稳定性。 例如您的 SDKAppID 是 1400xxxxxx，则独立域名为： `wss://1400xxxxxxw4c.my-imcloud.com`	必须

wss://wss.im.qcloud.com	Web IM 业务域名	必须
wss://wss.tim.qq.com	Web IM 业务域名	必须
wss://wssv6.im.qcloud.com	Web IM 业务域名	必须

- 将以下域名添加到 **request 合法域名**：

域名	说明	是否必须
https://web.sdk.qcloud.com	Web IM 业务域名	必须
https://boce-cdn.my-imcloud.com	Web IM 业务域名	必须
https://api.im.qcloud.com	Web IM 业务域名	必须
https://events.im.qcloud.com	Web IM 业务域名	必须
https://webim.tim.qq.com	Web IM 业务域名	必须
https://wss.im.qcloud.com	Web IM 业务域名	必须
https://wss.tim.qq.com	Web IM 业务域名	必须

- 将以下域名添加到 **uploadFile 合法域名**：

域名	说明	是否必须
https://${SDKAppID}-cn.rich.my-imcloud.com	**从 2024年9月10日起，新增应用默认分配 cos 独立域名。** 例如您的 SDKAppID 是 1400xxxxxx，则 cos 独立域名为： https://1400xxxxxx-cn.rich.my-imcloud.com	必须
https://cn.rich.my-imcloud.com	文件上传域名	必须
https://cn.imrich.qcloud.com	文件上传域名	必须
https://cos.ap-shanghai.myqcloud.com	文件上传域名	必须
https://cos.ap-shanghai.tencentcos.cn	文件上传域名	必须
https://cos.ap-guangzhou.myqcloud.com	文件上传域名	必须

- 将以下域名添加到 **downloadFile 合法域名**：

域名	说明	是否必须
https://${SDKAppID}-cn.rich.my-imcloud.com	**从 2024年9月10日起，新增应用默认分配 cos 独立域名。** 例如您的 SDKAppID 是 1400xxxxxx，则 cos 独立域名为： https://1400xxxxxx-cn.rich.my-imcloud.com	必须
https://cn.rich.my-imcloud.com	文件下载域名	必须
https://cn.imrich.qcloud.com	文件下载域名	必须
https://cos.ap-shanghai.myqcloud.com	文件下载域名	必须
https://cos.ap-shanghai.tencentcos.cn	文件下载域名	必须
https://cos.ap-guangzhou.myqcloud.com	文件下载域名	必须

- 如下图所示，服务器域名配置：



TUICallKit 源码集成

步骤一：开通服务

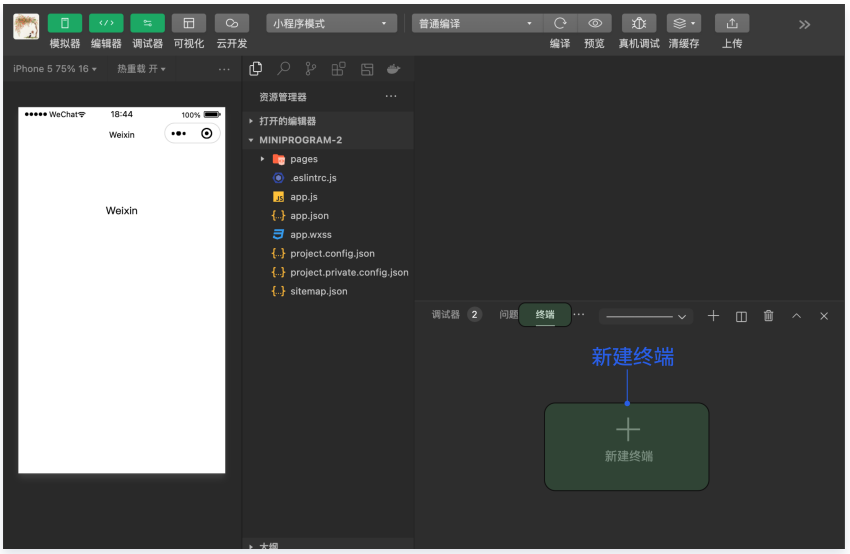
在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务。

步骤二：创建小程序项目

1. 在微信开发者工具上创建一个小程序项目，选择**不使用模板**。

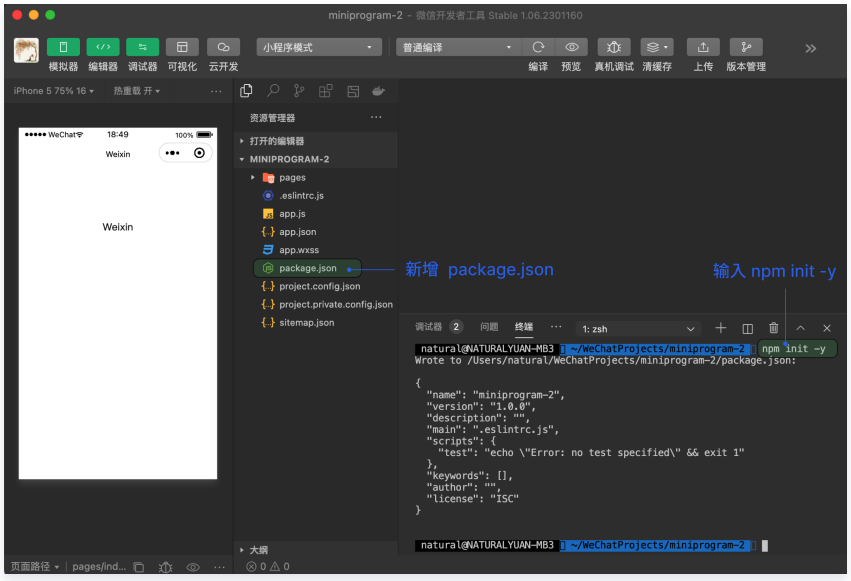


2. 新建终端。



3. 执行 `npm init -y` 命令生成 `package.json` 文件。

```
npm init -y
```



步骤三：下载并导入 TUICallKit 组件

1. 下载 TUICallKit 组件。

```
npm i @tencentcloud/call-uikit-wx
```

MacOS 端

```
mkdir -p ./TUICallKit && cp -r node_modules/@tencentcloud/call-uikit-wx/ ./TUICallKit && mkdir -p ./static && cp node_modules/@tencentcloud/call-engine-wx/RTCCallEngine.wasm.br ./static
```

Windows 端

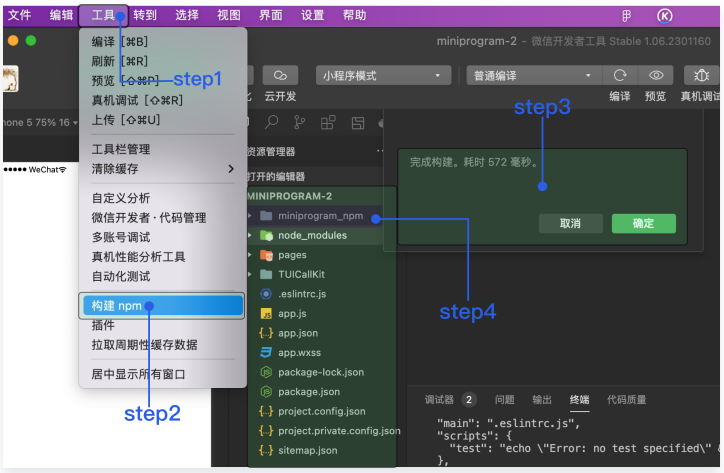
```
xcopy node_modules\@tencentcloud\call-uikit-wx\ .\TUICallKit /i /e
```

```
xcopy node_modules\@tencentcloud\call-engine-wx\RTCCallEngine.wasm.br .\static
```

2. 执行完以上命令后，您的目录下生成 TUICallKit 文件夹，其中包含有 TUICallKit 组件。目录如下：

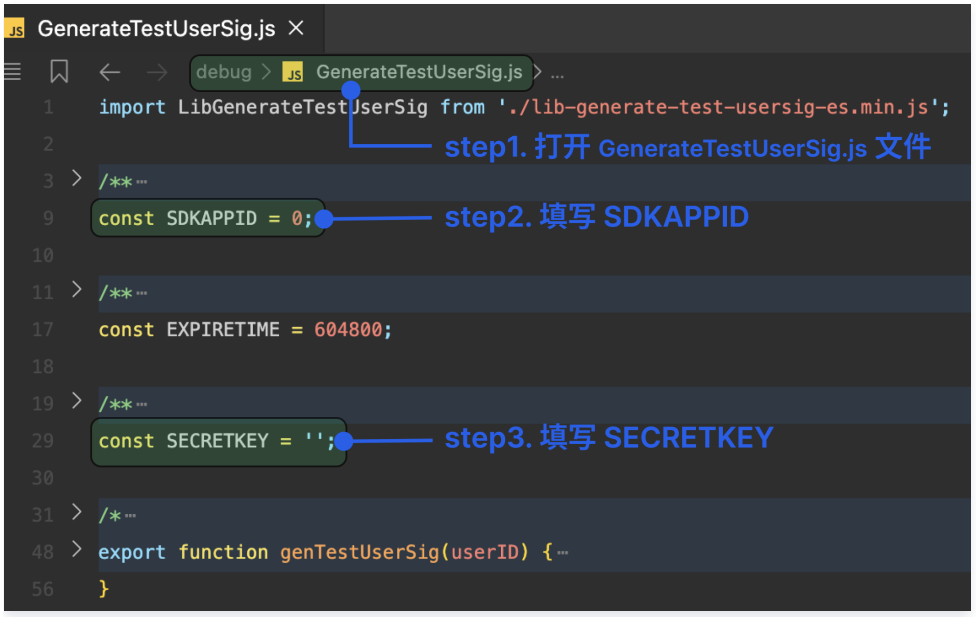


3. 构建 npm，打开微信开发者工具单击工具 > 构建 npm，新增 miniprogram_npm 目录。目录如下：



步骤四：填写 SDKAPPID 以及 SECRETKEY

修改 TUICallKit/debug/`GenerateTestUserSig-es.js` 文件 的 SDKAPPID 以及 SECRETKEY。



步骤五：调用 TUICallKit 组件

1. 修改 `app.json` 文件，添加如下代码，新增全局监听页面。

```
{
  "pages": [
    "pages/index/index",
    "TUICallKit/pages/globalCall/globalCall"
  ],
  "window": {
    "navigationBarTextStyle": "black",
    "navigationStyle": "custom"
  },
  "style": "v2"
}
```

2. 修改 `index` 文件夹下的文件。

index.wxml

```
<view class="container">
  <view class="box">
    <view class="input-box">
      <input type="text" maxlength="20" placeholder="{{isLogin?'请输入呼叫者userID':'请输入登录者userID'
}}" value="{{userID}}" bindinput='bindInputUserID' placeholder-style="color:#BBBBBB;" />
    </view>
    <view class='login'>
      <button class='loginBtn' bindtap="{{isLogin?'call':'login'}}">{{isLogin?'呼叫':'登录'}}</button>
    </view>
  </view>
</view>
```

index.js

```
// 导入 TUICallKitServer 模块，使您的应用具有全局呼叫的能力
import { TUICallKitServer } from "../../TUICallKit/TUICallService/index";
// 导入 CallManager 模块，使您的应用具有全局监听来电的能力
import { CallManager } from "../../TUICallKit/TUICallService/serve/callManager";
import * as GenerateTestUserSig from "../../TUICallKit/debug/GenerateTestUserSig-es.js";
wx.CallManager = new CallManager();
Page({
  data: {
    userID: "",
    isLogin: false,
  },

  bindInputUserID(e) {
    this.setData({
      userID: e.detail.value,
    });
  },

  async login() {
    const userID = this.data.userID;
    if (!userID) return;
    const { userSig, SDKAppID } = GenerateTestUserSig.genTestUserSig({
      userID: userID,
    });
    await wx.CallManager.init({
      sdkAppID: SDKAppID,
      userID: userID,
```

```
      userSig: userSig,
      globalCallPagePath: "TUICallKit/pages/globalCall/globalCall",
    });
    wx.showToast({
      title: "登录成功",
      icon: "error",
    });
    this.setData({
      isLogin: true,
      userID: "",
    });
  },

  async call() {
    await TUICallKitServer.calls({
      userIDList: [this.data.userID],
      type: 2,
    });
  },
});
```

index.wxss

```
.container {
  width: 100vw;
  height: 100vh;
}

.box{
  flex: 1;
  width: 100vw;
  margin-top: -40px;
  background: #ffffff;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

input{
  display: flex;
  font-size: 20px;
  width: 60vw;
}

.login {
  display: flex;
  width: 100vw;
  text-align: center;
  bottom: 5vh;
  margin: 70rpx;
}
.login button{
  width: 80%;
  background-color: #006eff;
  border-radius: 50px;
  color: white;
```

```
}
```

步骤六：编译运行

1. 请在本地设置里面勾选上“不校验合法域名、web-view (业务域名)、 TLS 版本以及 HTTPS 证书”。

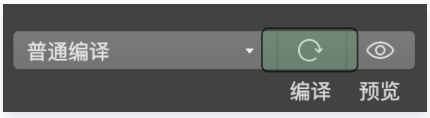


> **警告：**
> **如果不勾选该条目，则会在控制台出现如下错误。**
>
> 
>

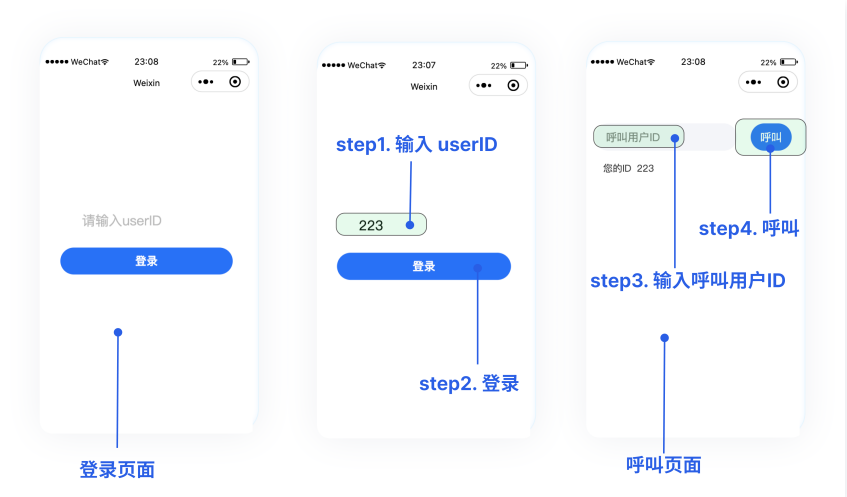
2. 单击**清缓存 > 全部清除**，避免开发者工具的缓存造成渲染异常。



3. 编译小程序。



4. 该项目快速集成后的预期效果图。



步骤七：拨打您的第一通电话

- 请单击预览，扫描二维码，在真机环境使用小程序。



- 登录后，请输入呼叫用户 ID，拨打您的第一通电话。具体效果如下图所示：



> ⚠ **注意：**
> 第一次使用小程序通话，需要获取摄像头和麦克风权限。

更多特性

- 设置昵称、头像
- 自定义铃声
- 群组通话

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

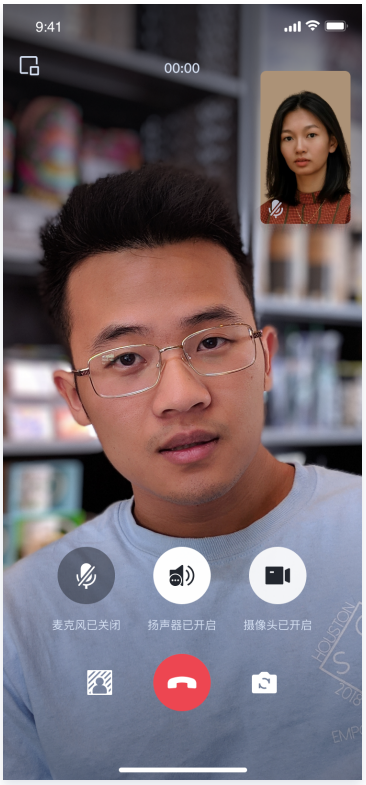
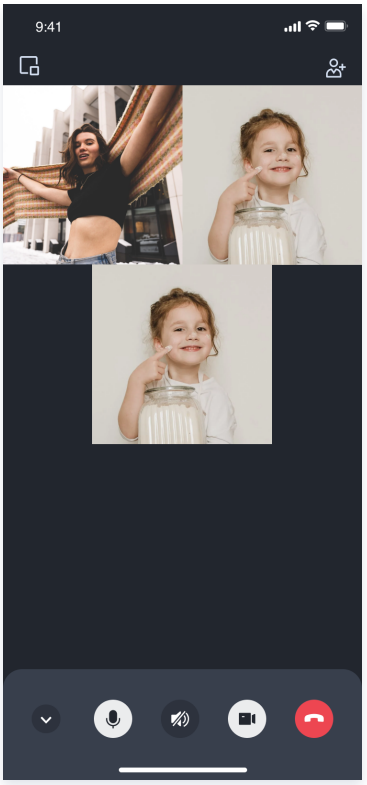
技术咨询

了解更多详情您可 腾讯云通信官方社群 进行咨询和反馈。

Flutter

最近更新时间：2025-02-20 14:18:12

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

视频通话	群组通话
	

环境准备

Flutter 3.0 及更高版本。

步骤一：开通服务

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务，获取 `SDKAppID`、`SDKSecretKey`，它们将在 步骤五 中使用，具体步骤请参见 开通服务。

步骤二：导入 TUICallKit 组件

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

```
flutter pub add tencent_calls_uikit
```

步骤三：完成工程配置

Android

1. 如果您需要编译运行在 Android 平台，由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单。
   - 首先，需要在工程的 `android/app/build.gradle` 文件中配置并开启混淆规则：

```
android {
```

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

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

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

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

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

iOS

由于 TUICallKit 会使用 iOS 的音视频功能，您需要授权麦克风和摄像头的使用权限。
授权操作方法：在您的 iOS 工程的 `Info.plist` 的第一级 `<dict>` 目录下添加以下两项，分别对应麦克风和摄像头在系统弹出授权对话框时的提示信息。

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

步骤四：设置 navigatorObservers

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

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

 ......

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

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

```
  }
}
```

步骤五：登录 TUICallKit 组件

使用 login 接口完成登录，具体使用可参考如下代码：

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

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

void login() async {
    String userSig  = GenerateTestUserSig.genTestSig(userID, sdkAppID, secretKey);
    TUIResult result = await TUICallKit.instance.login(sdkAppID, userID, userSig);
    if (result.code.isEmpty) {
      print('Login success');
    } else {
      print('Login failed: ${result.code} ${result.message}');
    }
}
```

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

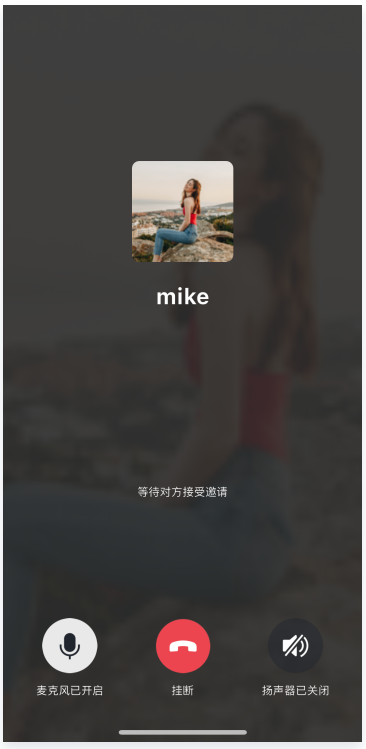
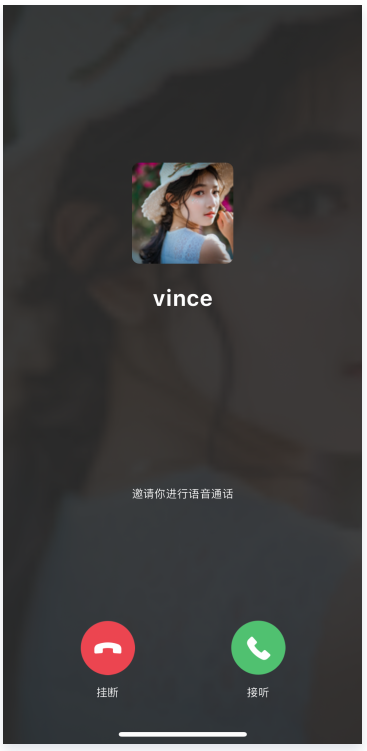
注意：

- **开发环境：**如果您正在本地开发调试阶段，可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 userSig。该方法中 SDKSecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
- **生产环境：**如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤六：拨打您的第一通电话

主叫方与被叫方登录成功后，主叫方通过调用 TUICallKit 的 calls 方法并指定通话类型和被叫方的 userId，就可以发起语音或者视频通话，被叫方此时就可接受到来电邀请。

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

void call() {
    List<String> userIdList = ['vince'];
    TUICallKit.instance.calls(userIdList, TUICallMediaType.audio);
}
```

	
主叫方	被叫方

更多特性

- 界面定制
- 离线推送
- 群组通话
- 悬浮窗
- 美颜特效
- 自定义铃声
- 监听通话状态
- 云端录制

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

交流与反馈

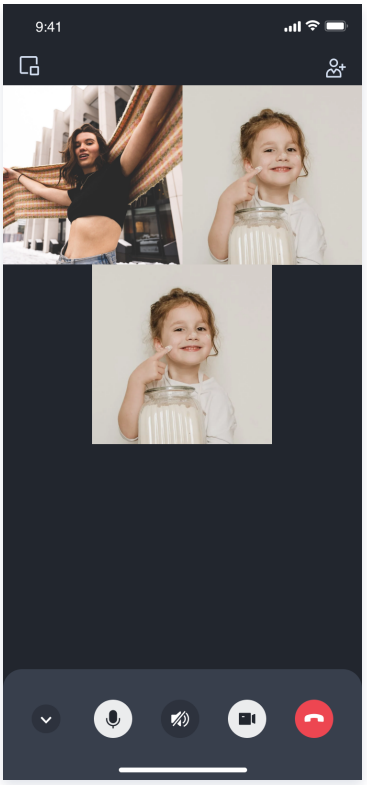
- 如果您在使用过程中，有什么建议或者意见，可以在这里反馈： TUICallKit 产品反馈问卷 ，感谢您的反馈。
- 如果您是开发者，也欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao ，进行技术交流和产品沟通。

版权所有：腾讯云计算（北京）有限责任公司

React Native

最近更新时间：2024-12-13 16:58:32

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

视频通话	群组通话
	

环境搭建

- Node.js 16 及更高版本

步骤 1. 开通服务

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

步骤 2. 下载 TUICallKit

1. 您可通过以下命令下载 @tencentcloud/call-uikit-react-native 组件。

```
yarn add @tencentcloud/call-uikit-react-native
```

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

MacOS

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

Windows

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

步骤 3. 初始化 TUICallKit 组件

您可以选择在 `/src/App.tsx` 文件引入示例代码。

1. 引入 call-uikit 相关 API 对象。

```
import { TUICallKit, MediaType } from '@tencentcloud/call-uikit-react-native';
import * as GenerateTestUserSig from "./debug/GenerateTestUserSig-es"; // Refer to Step 2.2
```

2. 调用 TUICallKit.login API 登录组件，需要在代码中**填写** `SDKAppID、SecretKey` **两个参数。**

```
const handleLogin = async () => {
  const userId = "denny";      // Please replace with your userId
  const SDKAppID = 0;          // Please replace with the SDKAppID obtained from step 1
  const SecretKey = "****";    // Please replace with the SDKSecretKey obtained from step 1

  const { userSig } = genTestUserSig({ userID: userId, SDKAppID, SecretKey });

  TUICallKit.login(
    {
      sdkAppId: SDKAppID,
      userId,
      userSig,
    },
    (res) => {},
    (errCode, errMsg) => {}
  );
};
```

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

userSig 说明：

- **开发环境**：如果您正在本地跑通 Demo、开发调试，可以采用 `debug` 文件中的 `genTestUserSig`（参考步骤3.2）函数生成 userSig。该方法中 SDKSecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
- **生产环境**：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤 4. 拨打您的第一通电话

1. 调用 TUICallKit.call API 拨打电话。

```
//【3】Make a 1v1 video call
const call = async () => {
  await TUICallKit.call({
    userID: 'mike',
    type: MediaType.Video,
```

```
    });
};
```

2. 在成功登录后，进行通话。

呼叫方（语音通话）	被叫方（语音通话）
	

常见问题

- 如果您的接入和使用中遇到问题，请参见 常见问题 。
- 了解更多详情您可 腾讯云通信官方社群 进行咨询和反馈。

uni-app (uts 插件)

最近更新时间：2025-06-29 18:01:02

本文将介绍如何快速完成 TUICallKit 组件的接入，您将在 10 分钟内完成以下几个关键步骤，并最终得到一个包含完备 UI 界面的视频通话功能。

CallKit 原生插件服务终止及迁移指引：

- 自 2025 年 5 月 1 日起，CallKit 原生插件已经终止迭代。为保障业务持续稳定运行并提升功能体验，现针对不同用户群体提供以下方案：
  - 新用户接入：推荐直接集成 CallKit UTS 插件，享受跨平台兼容、开发效率提升等进阶功能。
  - 存量用户使用：不影响已接入客户使用，后续新特性使用建议使用 UTS 插件。
- CallKit UTS 插件核心优势：
  - 支持自定义 UI、深度界面定制。
  - 支持 uni-app 及 uni-app x 项目。
  - 提供插件版本锁定功能，避免因更新导致兼容性问题。



开发环境要求

说明：
HBuilderX 4.64 和 4.65 版本对 uts 插件打包存在兼容性问题，建议使用最新版本。

- HBuilderX 4.45+。
- iOS 设备要求：iOS 系统 ≥ 13.0，支持音视频通话的真机设备。
- Android 设备要求：Android 系统 ≥ 5.0（SDK API Level 21 +），支持音视频的真机设备，允许 USB 调试。

步骤1：开通服务

请参见 开通服务，获取 `SDKAppID、SecretKey`，它们将在 TUICallKit.login 时作为**必填参数**使用。

步骤2：下载并导入 TencentCloud-Call 插件

1. 打开 【官方】腾讯云音视频通话插件 TencentCloud-Call ，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**。



步骤3：注册 TUICallKit

1. 将下面的代码引入到 main.js 文件，目的是为了将 TUICallKit、TUIStore 注册到全局，保证 App 共用一份实例。

```
import { TUICallKit } from '@/uni_modules/TencentCloud-Call/callkit/callServices/services/index';
import { TUIStore } from '@/uni_modules/TencentCloud-Call/callkit/callServices/TUIStore/index';

uni.$TUICallKit = TUICallKit;
uni.$TUIStore = TUIStore; // callkit 内部状态管理
```

2. 在 pages.json 中注册通话页面。

```
{
    "path": "uni_modules/TencentCloud-Call/callkit/callPage",
    "style": {
      "navigationBarTitleText": "",
      "navigationStyle": "custom",
    }
}
```

3. 拷贝 debug 目录，本地生成 userSig 时使用。

MacOS

```
cp -r uni_modules/TencentCloud-Call/debug ./
```

Windows

```
xcopy uni_modules/TencentCloud-Call/debug .\debug /i /e
```

4. 效果如下所示。



步骤4：登录 TUICallKit

调用 uni.$TUICallKit.login 进行登录。

```
import { genTestUserSig } from '../../debug/GenerateTestUserSig.js';

const { userSig, SDKAppID } = genTestUserSig('denny');

uni.$TUICallKit.login({
    SDKAppID: 123456,
    userID: 'denny',
    userSig: 'xxx',
    success: () => {
        console.log(demo login success , data);
    },
    fail: (errCode, errMsg) => {
        console.log(demo login fail , errCode : ${errCode} , errMsg : ${errMsg});
    },
})
```

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

userSig 说明：

- 开发环境：如果您正在本地跑通 Demo、开发调试，可以采用 debug 文件中的 genTestUserSig（参考步骤3.2）函数生成 userSig。该方法中 SDKSecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
- 生产环境：如果您的项目要发布上线，请采用 服务端生成 UserSig 的方式。

步骤5：拨打通话

调用 uni.$TUICallKit.calls 拨打通话。

```
uni.$TUICallKit.calls({
```

```
    userIDList: ['jenny', 'kris'],
    mediaType: 1,
})
```

步骤6：制作自定义基座

制作自定义调试基座，请选择**传统打包**方式进行打包。





步骤7：拨打您的第一通电话

1. 自定义调试基座成功后，**使用自定义基座运行**项目。



2. 拨打 1v1 视频通话具体效果如图所示。

主叫方发起音频通话	被叫方收到音频通话请求
	

更多特性

- 设置昵称、头像
- 多人通话
- 悬浮窗
- 监听通话状态
- 离线唤醒

常见问题

如果您的接入和使用中遇到问题，请参见 常见问题 。

了解更多详情您可加入 腾讯云通信官方社群 进行咨询和反馈。

界面定制（TUICallKit）
Android

最近更新时间：2025-06-26 09:55:52

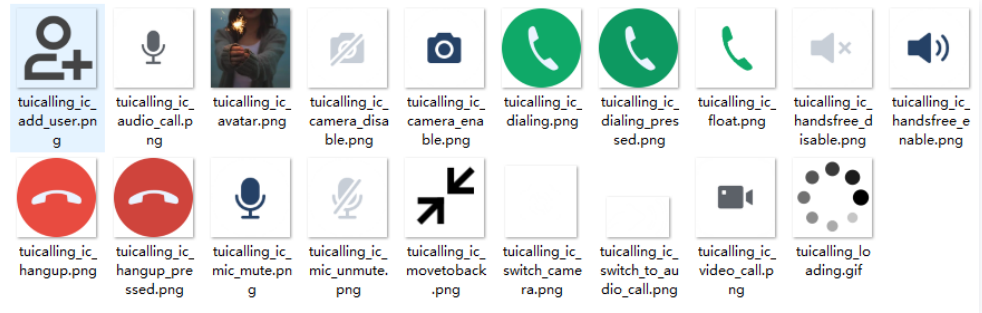
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 GitHub 中的 `Android/tuicallkit` 文件夹下面：

替换图标

您可以直接替换 `res\drawable-xxhdpi` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 `res\raw` 文件夹下的三个音频文件来达到替换铃声的目的：

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

替换文案

您可以通过修改 values-zh 和 values-en 中的 `strings.xml` 文件来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以删掉 tuicallkit 文件夹，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
calls	发起单人或多人通话
accept	接听通话

reject	拒绝通话
hangup	结束通话
ignore	忽略通话
inviteUser	在通话中，邀请其他人加入
join	主动加入通话
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（ 群组通话版套餐 支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallNotConnected	通话未接通的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserInviting	xxxx 用户被追加邀请加入通话时的回调
onUserJoin	xxxx 用户加入通话的回调

onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

关键类型定义

API	描述
TUICallDefine.MediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallDefine.Role	通话的角色，枚举类型：主叫、被叫
TUICallDefine.Status	通话的状态，枚举类型：空闲、待接听、接听中
TUICallDefine.CallParams	通话扩展参数：离线推送消息显示内容
TUICommonDefine.RoomId	音视频房间 ID，支持数字、字符串类型
TUICommonDefine.Camera	摄像头参数，枚举类型：前置摄像头、后置摄像头
TUICommonDefine.AudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUICommonDefine.NetworkQualityInfo	当前的网络质量信息

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2025-06-26 10:31:51

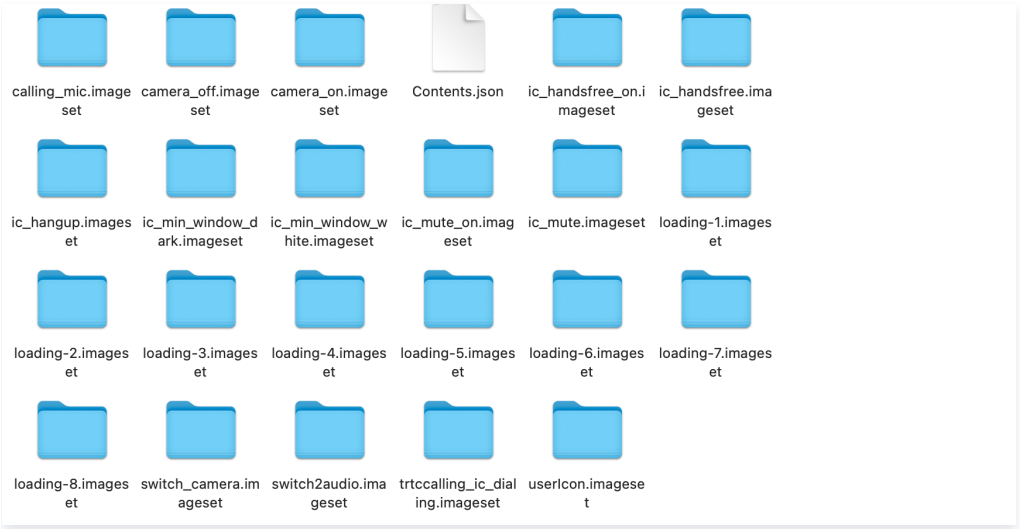
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。

方案一：界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 GitHub 中的 `iOS/TUICallKit` 文件夹下面：

替换图标

您可以直接替换 `Resources\Calling.xcassets` 文件夹下的图标，以确保整个 app 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 `Resources\AudioFile` 文件夹下的三个音频文件来达到替换铃声的目的。

文件名	用途
phone_dialing.m4a	发起呼叫时的声音
phone_hangup.mp3	被挂断的声音
phone_ringing.flac	接到呼叫时的声音

替换文案

您可以通过修改 zh-Hans.lproj 和 en.lproj 中的 `CallingLocalized.strings` 文件来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以删掉 tuicallkit 文件夹，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调

removeObserver	移除回调接口
calls	发起单人或多人通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
inviteUser	在通话中，邀请其他人加入
join	主动加入通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallEnd	通话结束的回调
onCallNotConnected	通话未接通的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserInviting	xxxx 用户被追加邀请加入通话时的回调
onUserJoin	xxxx 用户加入通话的回调

onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

关键类型定义

API	描述
TUICallMediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallRole	通话的角色，枚举类型：主叫、被叫
TUICallStatus	通话的状态，枚举类型：空闲、待接听、接听中
TUIRoomId	音视频房间 ID，支持数字、字符串两种类型
TUICamera	摄像头 ID 参数，枚举类型：前置摄像头、后置摄像头
TUIAudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUINetworkQuality	当前的网络质量信息

版权所有：腾讯云计算（北京）有限责任公司

Web

最近更新时间：2025-06-30 20:45:42

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

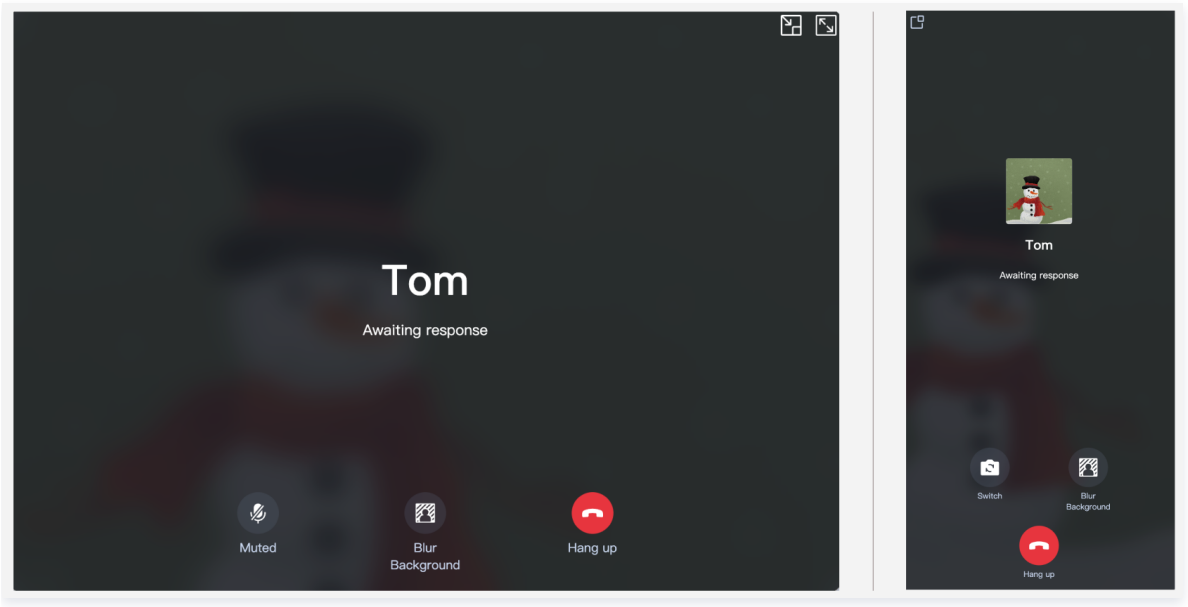
方案一：界面微调方案

按钮隐藏

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

> **注意：**
> v3.2.9+ 支持。

以隐藏**摄像头按钮**为例。



Vue3

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

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

Vue2.7

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

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

Vue2.6

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

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

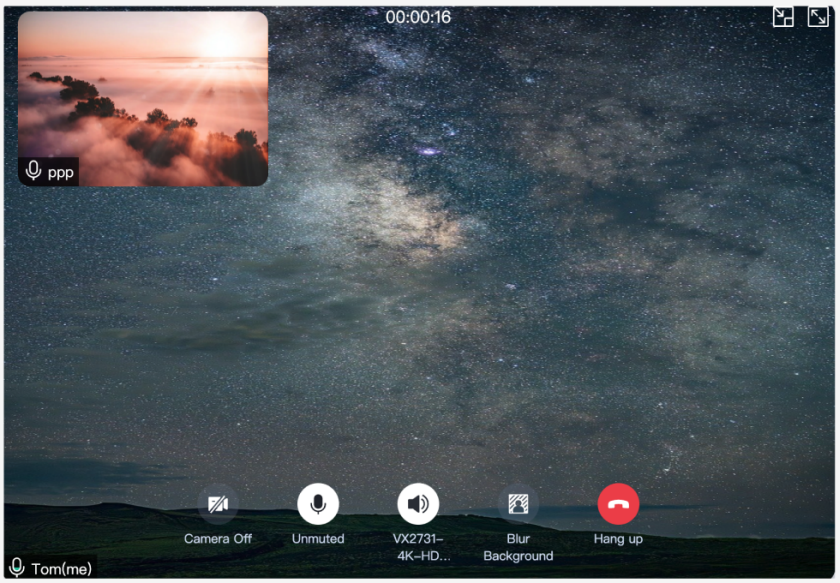
React

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

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

自定义通话背景图

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

> **注意：**
> v3.2.9+ 支持。



Vue3

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

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

Vue2.7

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

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

Vue2.6

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

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

React

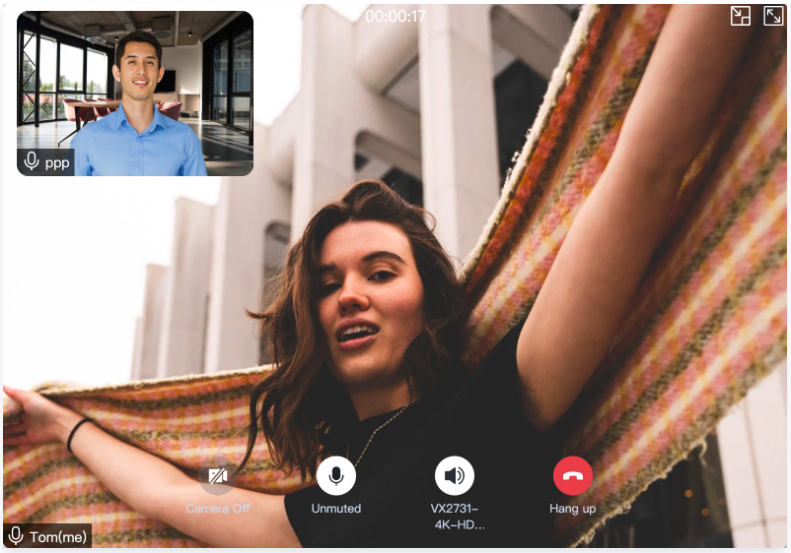
```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";

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

设置布局

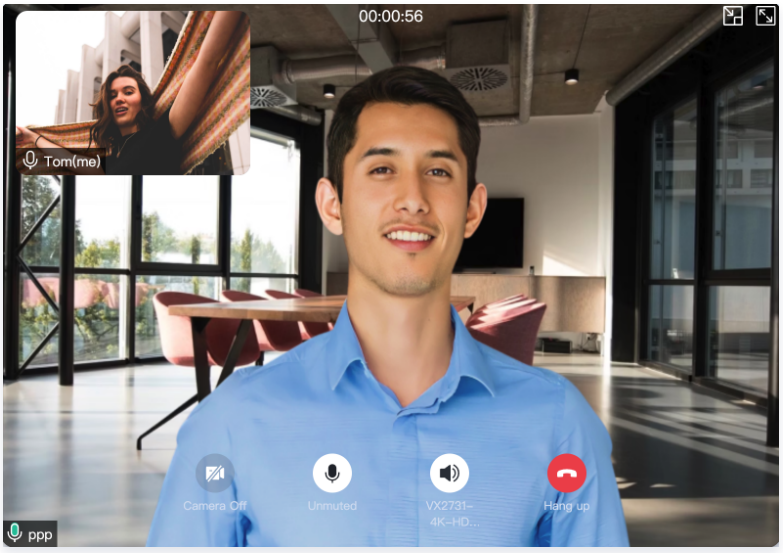
> ⚠ **注意：**
> **仅 1V1 视频通话可用，v3.3.0+ 支持。**

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

1. LocalInLargeView layout，本地用户在大窗：



2. RemoteInLargeView layout，远端用户在大窗：



Vue3

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

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

Vue2.7

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

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

Vue2.6

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

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

React

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

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

设置摄像头初始状态

> ⚠ **注意：**
> **v3.3.0+ 支持。**

调用 setCameraDefaultState 设置摄像头按钮初始状态，目前支持开启和关闭。
以默认关闭摄像头为例：

Vue3

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

TUICallKitServer.setCameraDefaultState(false);
```

Vue2.7

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

TUICallKitServer.setCameraDefaultState(false);
```

Vue2.6

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

TUICallKitServer.setCameraDefaultState(false);
```

React

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

```
TUICallKitServer.setCameraDefaultState(false);
```

替换图标

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

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

1. 下载源码

Vue3

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

Vue2.7

```
npm install @tencentcloud/call-uikit-vue2
```

Vue2.6

```
npm install @tencentcloud/call-uikit-vue2.6
```

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

macOS + Vue3

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

macOS + Vue2.7

```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uikit-vue2/*
./src/components/TUICallKit
```

Windows + Vue3

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

Windows + Vue2.7

```
xcopy .\node_modules\@tencentcloud\call-uikit-vue2  .\src\components\TUICallKit /i /e
```

macOS + Vue2.6

```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uikit-vue2.6/*
./src/components/TUICallKit
```

Window + Vue2.6

```
xcopy .\node_modules\@tencentcloud\call-uikit-vue2.6  .\src\components\TUICallKit /i /e
```

3. **修改引入路径**

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

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

4. **解决源码拷贝可能导致的报错**

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

ESLint 报错

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

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

TypeScript 报错

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

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

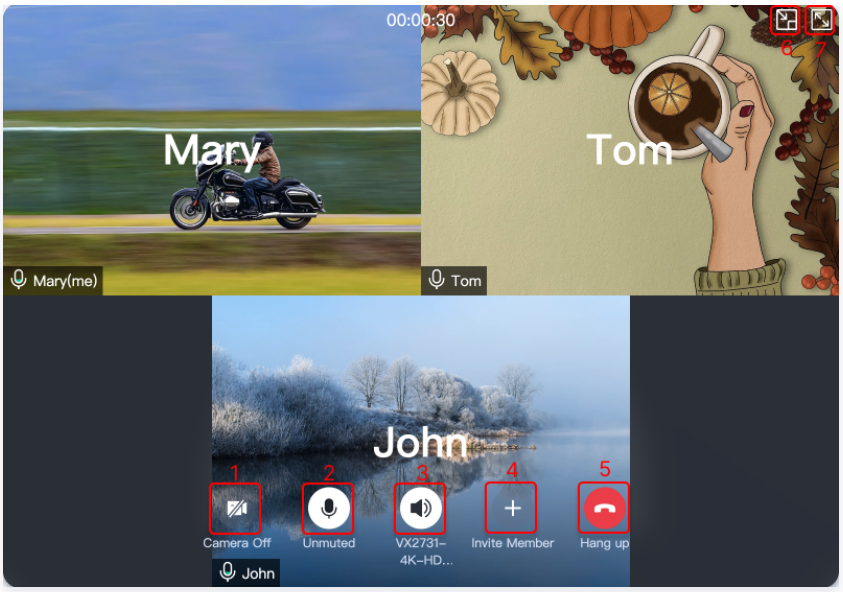
   其他 TSConfig 问题请参见 TSConfig Reference。
2. 如遇 Uncaught SyntaxError: Invalid or unexpected token 报错，是因为 TUICallKit 使用了装饰器，可在 tsconfig.json 中添加相关配置，示例：

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

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

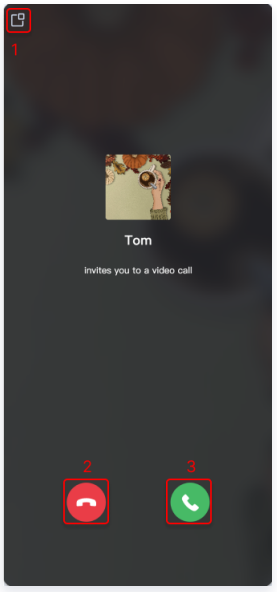
> ⚠ **注意：**
> 为了确保整个应用中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。

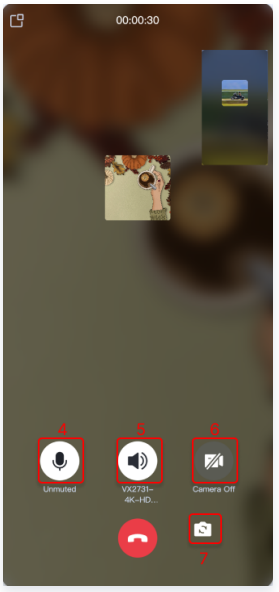
桌面端



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

移动端





版权所有：腾讯云计算（北京）有限责任公司

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

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI SDK 实现的，您可以完全基于 TUICallEngine 实现一套自己的 UI 界面。详情可见：

- TUICallEngine 接入指引
- TUICallEngine API 接口地址

微信小程序

最近更新时间：2025-05-06 17:24:32

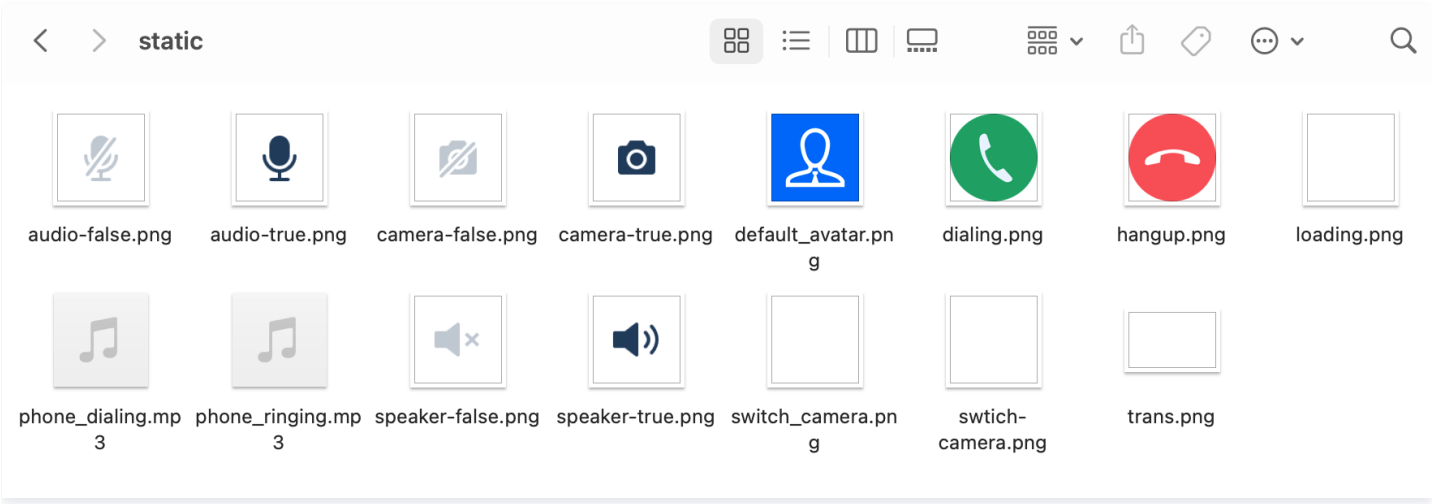
本文将介绍如何定制 TUICallKit 的用户界面，我们提供了**界面微调方案**。

界面微调方案

通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 @tencentcloud/call-uikit-wx 中的 `static` 文件夹下面。

替换图标

您可以直接替换 `TUICallKit\static` 文件夹下的图标，以确保整个小程序中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 `TUICallKit\static` 文件夹下的两个音频文件来达到替换铃声的目的：

文件名	用途
phone_dialing.mp3	发起呼叫时的声音
phone_ringing.mp3	接到呼叫时的声音

Flutter

最近更新时间：2024-10-14 18:02:11

本文将介绍如何定制 TUICallKit 的用户界面，我们提供了两个方案供您选择：**界面微调方案**和**自实现 UI 方案**。
注意：页面定制方案需要使用 tencent_calls_uikit 插件 1.8.0 以上的版本。

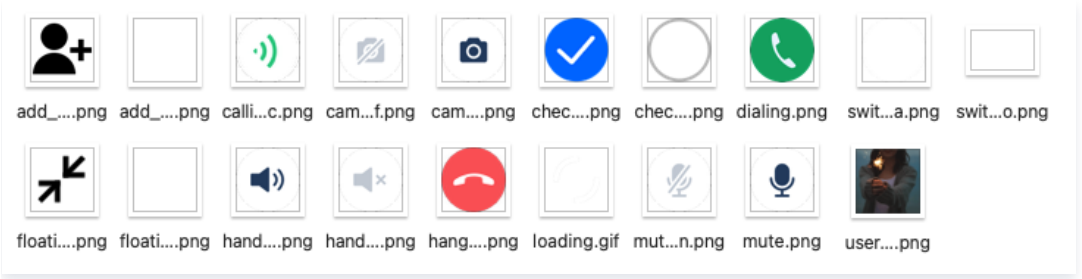
方案一：界面微调方案

您可以下载 tencent_calls_uikit 插件的最新版本到本地，之后在您的工程中使用本地依赖的方式接入插件，本地依赖方式如下：
在工程 **pubspec.yaml** 文件中的 **dependencies** 节点下，添加 **tencent_calls_uikit** 插件依赖，示例如下：

```
dependencies:
    tencent_calls_uikit:
      path: 插件目录
```

替换图标

您可以直接替换 `assets\images` 文件夹下的图标，以确保整个 App 中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



替换铃声

您可以替换 **assets\audios** 文件夹下的三个音频文件来达到替换铃声的目的：

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

替换文案

您可以通过修改 **lib\src\i18n** 目录下 **strings.g.dart** 文件中的字符串来修改视频通话界面中的字符串内容。

方案二：自实现 UI 方案

TUICallKit 的整个通话功能是基于 TUICallEngine 这个无 UI 组件实现的，您可以不依赖 **tencent_calls_uikit** 插件，而直接依赖 tencent_calls_engine 插件，完全基于 TUICallEngine 实现一套自己的 UI 界面。

TUICallEngine

TUICallEngine 是整个通话组件的底层接口，主要提供了1对1音视频通话和群组内通话的发起、接听、拒绝、结束以及设备操作等关键接口。

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，接收自己感兴趣的回调事件。

uni-app（小程序）

最近更新时间：2025-06-30 13:32:21

本文将介绍如何定制 TUICallKit 的用户界面，我们提供了**界面微调方案**。

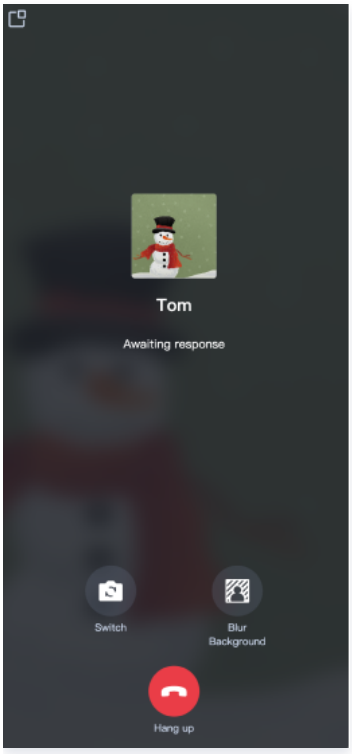
界面微调方案

按钮隐藏

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

> ⚠ **注意：**
> **v3.2.9+ 支持。**

以隐藏摄像头按钮为例。

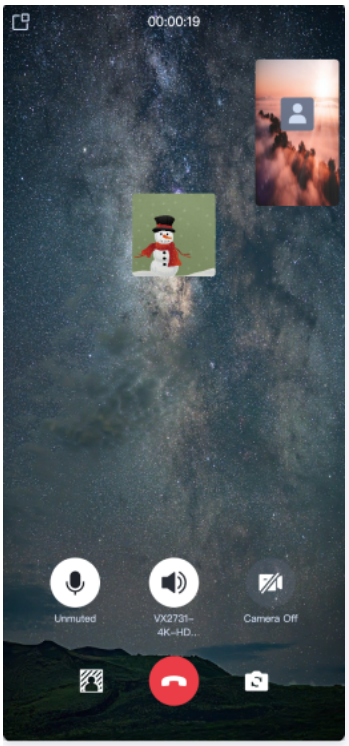



```
// 替换成 TUICallKit 源码中的地址
import { TUICallKitServer, FeatureButton } from "../../TUICallKit/src/index";

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

自定义通话背景图

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

> ⚠ **注意：**
> **v3.2.9+ 支持。**

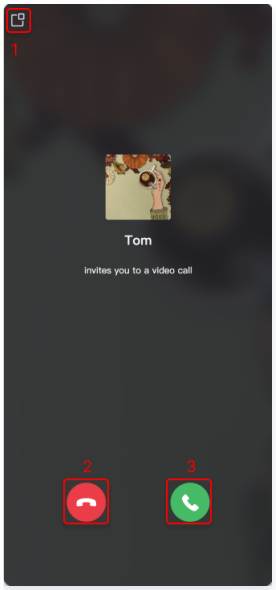


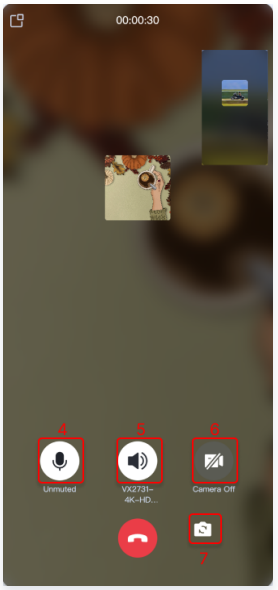

```
import { TUICallKitServer } from "../../TUICallKit/src/index";

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

替换图标

您可以直接修改 `TUICallKit/Components/assets` 文件夹下的图标组件，以确保整个应用中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。

> **注意：**
> v3.2.2+ 支持





序号	资源路径

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

版权所有：腾讯云计算（北京）有限责任公司

uni-app (uts 插件)
替换页面图标

最近更新时间：2025-06-27 17:35:32

功能概述

本文将介绍如何定制 TUICallKit 的用户界面，我们提供了**界面微调方案**。

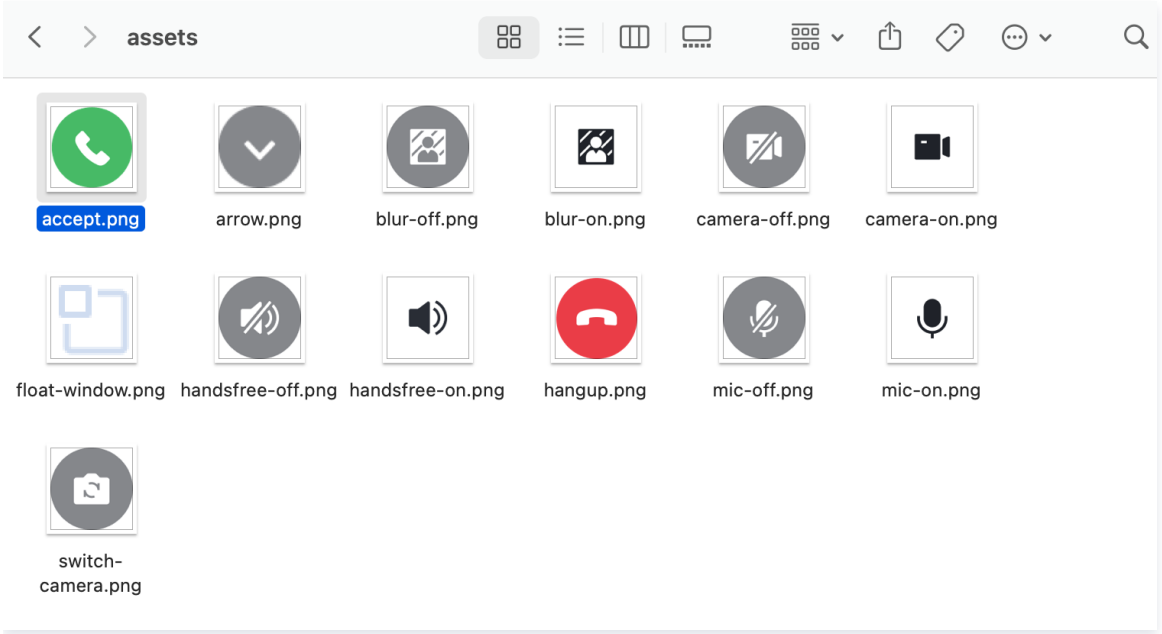
界面微调方案

前提条件

- 已接入 TencentCloud-Call uts 插件，如果没有接入，详细接入方法可参见 快速接入 。
- 通过直接修改我们提供的 UI 源代码，对 TUICallKit 的用户界面进行调整，TUICallKit 的界面源代码位于 uni_modules/TencentCloud-Call/callkit/components。

替换图标

您可以直接修改 uni_modules/TencentCloud-Call/callkit/components/assets 文件夹下的图标组件，以确保您整个应用中的图标色调风格保持一致，请在替换时保持图标文件的名字不变。



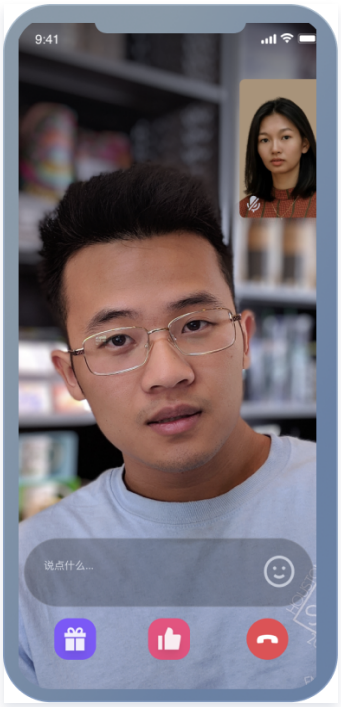
高级定制能力

最近更新时间：2025-06-27 17:35:32

TUICallKit 采用模块化架构设计，核心功能标准化，交互界面定制化，深度解耦音视频通话核心能力与界面呈现层。开发者可基于标准接口保留基础通话功能，同时通过灵活的UI组件替换机制，快速适配各业务场景的差异化界面需求。本文主要介绍，在保留核心通话能力的基础上，如何快速实现功能模块的自定义。

应用场景

若您的业务场景仅需显示用户的视频流，并依赖内部通话状态与设备状态管理，而无需集成其他功能按钮，则此方案尤为适用，能够帮助您实现高度定制化的界面效果。

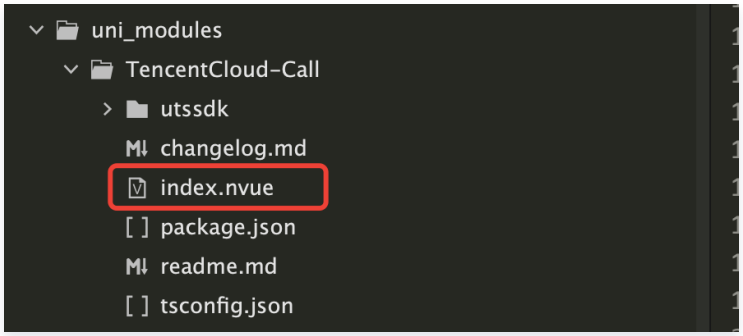
音频通话加载简单按钮	视频通话加载复杂布局
	

实现自定义 UI

新增自定义 UI 页面

1. 在 uni_modules/TencentCloud-Call 目录下新建 index.nvue 页面：

> **注意：**
> 使用原生能力，需要是 nvue 页面，vue 页面无法正常使用。



2. 在 page.json 文件中新增页面配置：

```
{
  "pages": [ //pages数组中第一项表示应用启动页，参考：https://uniapp.dclou
    {
      "path": "pages/index/index",
      "style": {
        "navigationBarTitleText": "uni-app"
      }
    },
    {
      "path": "uni_modules/TencentCloud-Call/index",
      "style": {
        "navigationBarTitleText": "", // 清空标题文字
        "navigationStyle": "custom" // 关键配置：隐藏原生导航栏
      }
    }
  ],
  "globalStyle": {
    "navigationBarTextStyle": "black",
    "navigationBarTitleText": "uni-app",
    "navigationBarBackgroundColor": "#F8F8F8",
    "backgroundColor": "#F8F8F8"
  },
  "uniIdRouter": {}
}
```

引入 callView 组件

> **说明：**
> <call-view/> 是管理显示用户的视频流，内部通话状态与设备状态的原生组件。

在新建的 index.nvue 页面中引入 callView 组件：

```
<template>
  <view>
    <call-view :style="callViewStyle" />
  </view>
</template>
<script setup lang="ts">
const callViewStyle = computed(() => {
  return {
    width: uni.getWindowInfo().windowWidth + 'px',
    height: uni.getWindowInfo().windowHeight + 'px',
  }
})
</script>
```

添加简单按钮

```
<template>
  <view>
    <call-view :style="callViewStyle" />
    <button :style="{position: 'absolute',bottom: '10px' }"></button>
  </view>
</template>
<script setup lang="ts">
const callViewStyle = computed(() => {
  return {
    width: uni.getWindowInfo().windowWidth + 'px',
    height: uni.getWindowInfo().windowHeight + 'px',
  }
})
```

```
</script>
```

添加复杂 UI 也是和上面一致，自实现 UI 引入到这个文件中来进行渲染。

使用基本通话功能

```
<template>
  <view>
    <call-view :style="callViewStyle" />
    <button :style="{position: 'absolute',bottom: '10px' }" @tap="handleStartCall"></button>
  </view>
</template>
<script setup lang="ts">
import { onMounted, onUnMounted, ref } from "vue";
import { CallEngine } from "@/uni_modules/TencentCloud-Call";
const callEngine = new CallEngine();
const callViewStyle = computed(() => {
  return {
    width: uni.getWindowInfo().windowWidth + 'px',
    height: uni.getWindowInfo().windowHeight + 'px',
  }
})

// 可以根据不同的 CallStatus 来进行不同的按钮状态展示，CallStatus 有 IDLE、CALLING、CONNECTED 三种状态
const callStatus = ref('IDLE')
onMounted(() => {
  callEngine.login({
    SDKAppID: 0,
    userID: '12345',
    userSig: '',
  })
  callEngine.on("onCallBegin", (res) => {
    callStatus.value = 'CONNECTED'
  })
  callEngine.on("onCallEnd", (res) => {
    callStatus.value = 'IDLE'
  })
});

onUnMounted(() => {
  callEngine.off("onCallBegin", (res) => {
    console.log(res)
  })
  callEngine.off("onCallEnd", (res) => {
    console.log(res)
  })
});


const handleStartCall = () => {
  callEngine.calls({
    userIDList: ['456'],
    mediaType: 1, // 1 为语音通话，2为视频通话
  })
  callStatus.value = 'CALLING'
}
</script>
```

更多接口使用请参见 call-engine 。

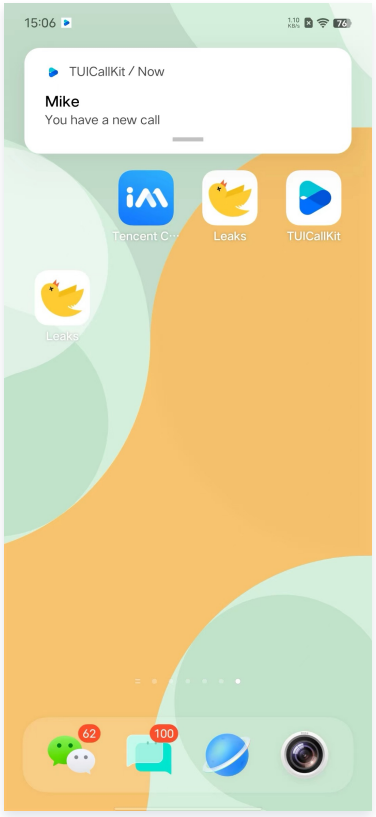
离线唤醒（TUICallKit）Flutter Notification

最近更新时间：2025-02-27 10:30:43

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

- 接入周期短，预计全厂商接入仅需要 30 分钟；
- 支持数据统计和链路追踪，方便您查看推送触达率、点击率和转化率等各类指标；
- 支持全员/标签推送，方便您将营销广告、通知、新闻资讯等内容推送给所有用户或指定群体；
- 支持 uni-app 和 Flutter 等跨平台框架。

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

Android	iOS
	

集成指引

请前往 TIMPush 集成方法 。

拨打离线推送通话

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

```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
 offlinePushInfo.title = "Flutter TUICallKit";
 offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
 offlinePushInfo.ignoreIOSBadge = false;
 offlinePushInfo.iOSSound = "phone_ringing.mp3";
```

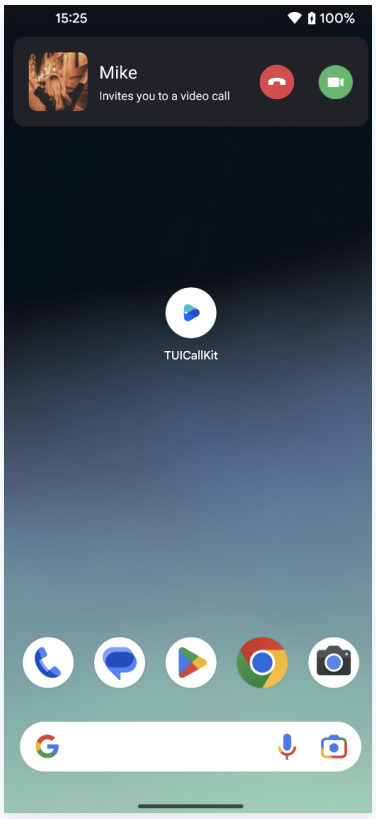
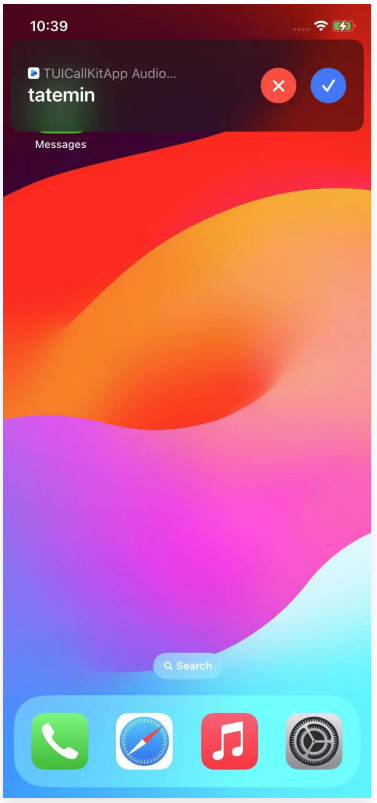
```
offlinePushInfo.androidSound = "phone_ringing";
offlinePushInfo.androidOPPOChannelID = "Flutter TUICallKit";
offlinePushInfo.androidVIVOClassification = 1;
offlinePushInfo.androidFCMChannelID = "fcm_push_channel";
offlinePushInfo.androidHuaWeiCategory = "Flutter TUICallKit";
offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.APNs;

TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

TUICallKit.instance.calls(callUserIdList, TUICallMediaType.audio, params);
```

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

VoIP

最近更新时间：2025-02-27 10:30:43

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

Android	iOS
	

配置VoIP

iOS

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

Android

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

集成消息推送插件

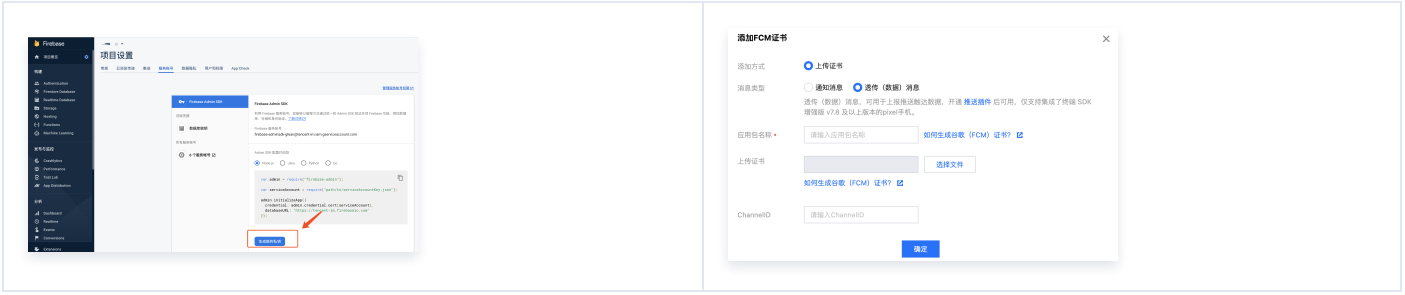
本插件在 pub.dev 的包名为：tencent_cloud_chat_push ，您可以收到将其引入 pubspec.yaml 依赖目录中, 也可以执行下列命令, 自动安装。

```
flutter pub add tencent_cloud_chat_push
```

准备条件

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

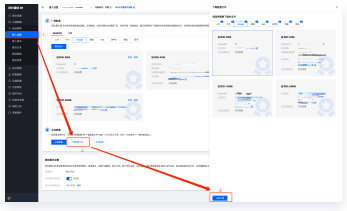
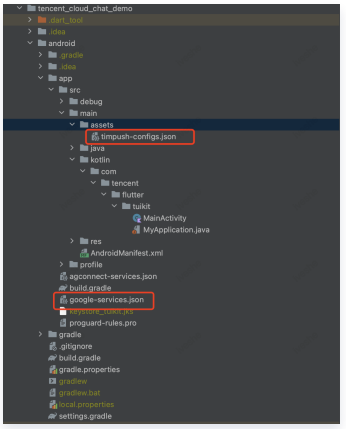
厂商推送平台	IM 控制台配置



快速接入

1. 下载并添加配置文件

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

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

2. 集成 TIMPush 插件

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

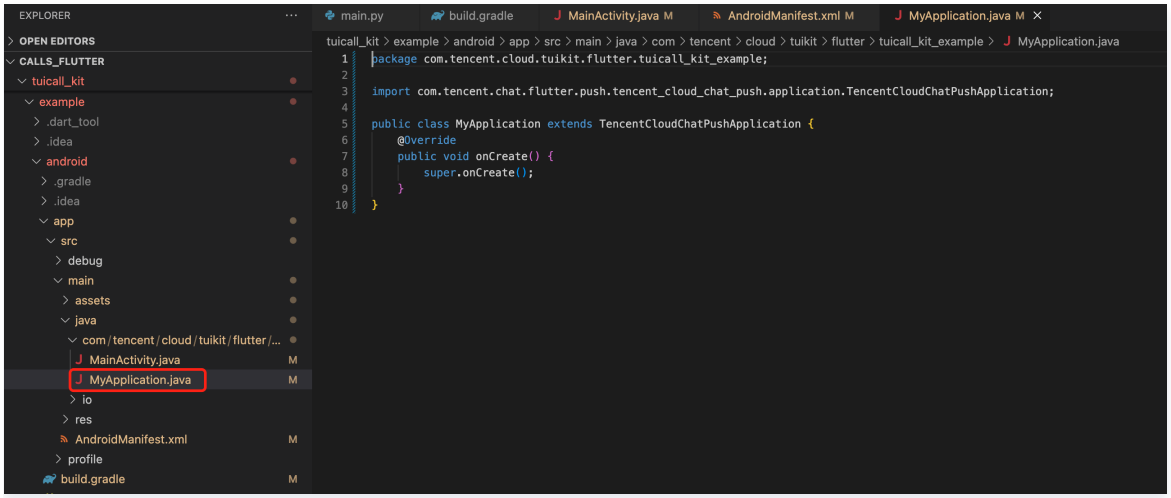
```
implementation "com.tencent.timpush:fcm: xxxxxx"
```

说明：

- TIMPush 需要集成 IM SDK 在7.9.5666版本及以上。TIMPush 的版本号及更新信息，可以在 更新日志 中进行查询。
- 在 `build.gradle` 中添加的依赖的版本需要与 `tencent_cloud_chat_push` 版本对应。

3. 客户端代码配置

在您项目 android 路径下 `MainActivity` 同级目录中，新建一个新的 Application 文件类, 例如可命名为 `MyApplication` 。
如果您已经自定义了一个 Application 类，则可直接复用，不需要再次创建。

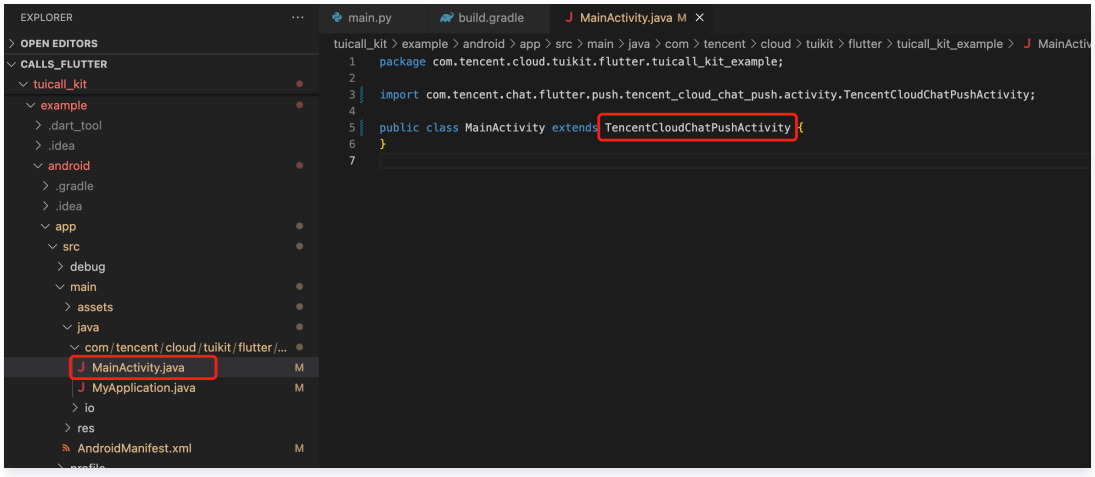


将下列代码粘贴到该文件中，如上图所示：

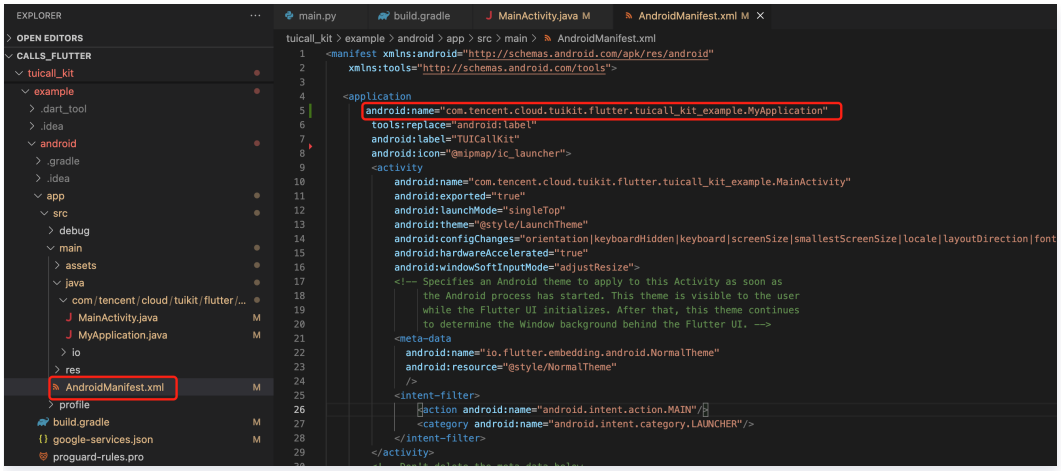
```
package xxxx.xxxx.xx

import
com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentCloudChatPushApplication;

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

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

同时，您还需要修改您的 MainActivity 文件：



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



4. 完成工程配置

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

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

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

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

5. 注册推送插件

请在登录完成后，立即注册推送插件。
调用 `TencentCloudChatPush().registerPush` 方法，需传入上一步定义的点击回调函数。
此外，您还可选传入 `apnsCertificateID` iOS 推送证书 ID 及 `androidPushOEMConfig` Android 推送厂商配置。此二项配置已在前序步骤指定，若无修改必要，可不再传入。

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

6. 实现自动登录

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

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

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

7. 拨打离线推送通话

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

```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
 offlinePushInfo.title = "Flutter TUICallKit";
 offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
 offlinePushInfo.ignoreIOSBadge = false;
 offlinePushInfo.iOSSound = "phone_ringing.mp3";
```

版权所有：腾讯云计算（北京）有限责任公司

```
offlinePushInfo.androidSound = "phone_ringing";
offlinePushInfo.androidOPPOChannelID = "Flutter TUICallKit";
offlinePushInfo.androidVIVOClassification = 1;
offlinePushInfo.androidFCMChannelID = "fcm_push_channel";
offlinePushInfo.androidHuaWeiCategory = "Flutter TUICallKit";
offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.VoIP;

TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

TUICallKit.instance.calls(callUserIdList, TUICallMediaType.audio, params);
```

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

常见问题

应用被终止后，无法显示来电 UI?

- 确认收到了推送，收不到推送需要确认下 IM 控制台是否正确上传证书。参见文档上述快速接入的 第一步，看下是否添加正确。
- 确认控制台选择了 FCM 数据消息，对照上述准备条件的 第二步 。
- 确认收到数据消息，过滤日志（关键字：TIMPush），检查下述日志有打印（如果收不到消息，可以使用 IM 控制台的 排查工具 查看原因）。

```
5 TIMPushFCMMsgService    Message data payload: {IMDesc=You have a new call, IMSound=phone_ringing, IMTitle=258, ext={"timPushFeatures":{"fcmNotificationType":1,"fcmPushType":0},"entity":{"action":2}},
```

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

交流与反馈

如有问题，欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao，进行技术交流和产品沟通。

Android Notification

最近更新时间：2025-05-27 16:09:52

TUICallKit 组件支持通过接入推送插件，实现离线推送的功能。TUICallKit 为您提供了两种集成方式：TIMPush 插件（付费）和自集成推送（免费），我们更推荐您使用 TIMPush 插件来实现离线推送，TIMPush 插件有以下优势：

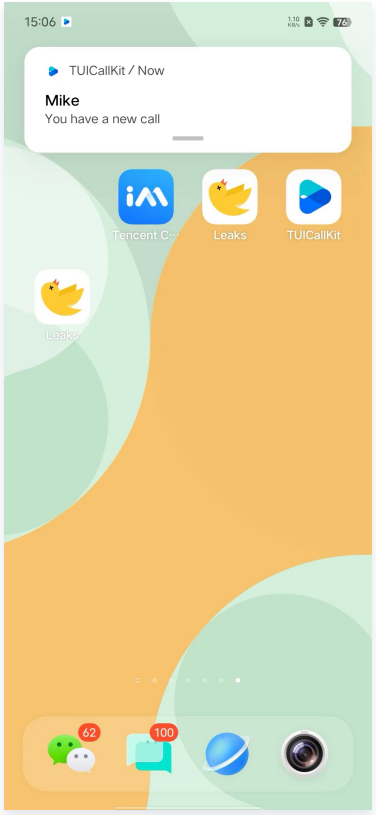
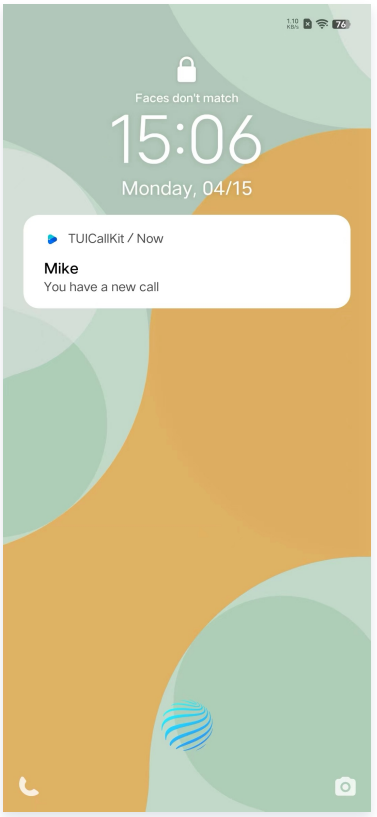
- 接入周期短，预计全厂商接入仅需要 30 分钟。
- 支持数据统计和链路追踪，方便您查看推送触达率、点击率和转化率等各类指标。
- 支持全员/标签推送，方便您将营销广告、通知、新闻资讯等内容推送给所有用户或指定群体。
- 支持 uni-app 和 Flutter 等跨平台框架。

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

> **说明：**
> 如果您想要实现**自集成推送**，请参见：Android-自集成推送。跑通离线推送功能中，步骤7和步骤8不需要配置，TUICallKit 组件内部已完成。

集成效果

TUICallKit 含UI方案集成 TIMPush推送插件 的效果如下（以下是 vivo X60的显示效果，不同厂商的设备可能会呈现出不同的效果）。

应用在后台时或离线时	锁屏时
	

准备条件

1. 开通服务

进入 IM 控制台 > 插件市场，单击**立即购买**或**免费试用**（每个应用可免费试用一次，有效期7天）。



注意：

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 。

2. 厂商配置

在进行离线推送功能的集成前，请确保已按照官方文档完成 厂商配置 ，以确保TUICallKit的离线推送功能正常运行。

说明：

因为 FCM 依赖于 Google Mobile Service 框架且需要能够访问 Google 服务，我们建议：

- 若您的业务运营区域限定于中国大陆境内，则参考该文档，按需集成小米等国内厂商通道即可。
- 若您的业务运营区域不涉及中国大陆境内，则无需集成小米等国内厂商的推送服务，只需参考文档：FCM Data message ，集成 FCM 即可。
- 若您的业务运营区域是涉及中国大陆境内外，则需要同时集成小米等国内厂商的推送服务以及 FCM 推送服务。
- 若您的业务想要强制使用 FCM 通道，可调用以下接口：

```
TIMPushManager.getInstance().forceUseFCMPushChannel(true);
```

功能接入

1. 请参见推送插件 TIMPush 快速接入 文档，完成其中的**步骤1、步骤2、步骤3及步骤5**。

说明：

步骤4 无需单独调用；TUICallKit组件在调用 TUILogin.login() 接口实现登录时，会自动完成注册。
步骤6 和步骤7 在 TUICallKit 组件中发起通话请求和接收到通话请求时已完成，不需要单独配置。

2. 在项目的 app 目录下，找到并打开 `build.gradle` 文件，将应用包名修改为您的实际应用包名。

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

3. 实现自动登录功能。 您可以在自己的业务逻辑中实现自动登录，也可以在应用 Application 的 oncreate() 函数中注册以下通知回调，实现自动登录。

```
TUICore.registerEvent(TUIConstants.TIMPush.EVENT_NOTIFY,
TUIConstants.TIMPush.EVENT_NOTIFY_NOTIFICATION, new ITUINotification() {
```

```
        @Override
        public void onNotifyEvent(String key, String subKey, Map<String, Object> param) {
            if (TUIConstants.TIMPush.EVENT_NOTIFY.equals(key)) {
                if (TUIConstants.TIMPush.EVENT_NOTIFY_NOTIFICATION.equals(subKey)) {
                    // please implement this method by yourself
                    autoLogin();
                }
            }
        }
    });
```

完成上述步骤后，您的项目即具备了TUICallKit的离线唤醒功能。当主叫方发起通话时，被叫方即使在后台或离线状态，也能收到推送通知。用户点击推送通知进入应用并登录后，将自动拉起通话界面。

> **说明：**
> 如果您的应用在接收推送或拉起页面时遇到问题，您可以参见 后台时拉不起通话界面 、 被叫端来电显示策略 对问题进行排查。

高级功能

自定义通知内容

- 在调用 calls 发起通话时，可通过自定义 CallParams 参数来配置 OfflinePushInfo 内容。
- TUICallKit 组件内 OfflinePushInfoConfig.kt 文件提供了默认的通知样式，可以直接使用，或者也可以修改相应代码进行自定义。

Kotlin

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

val pushInfo = TUICallDefine.OfflinePushInfo()
pushInfo.title = "Mike"
pushInfo.desc = "You have a new call"

val params = TUICallDefine.CallParams()
params.offlinePushInfo = pushInfo

val list = mutableListOf<String>()
list.add("Mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, params, null)
```

Java

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

TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setTitle("Mike");
pushInfo.setDesc("You have a new call");

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = pushInfo;

List<String> list = new ArrayList<>();
list.add("Mike");
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, params, null);
```

自定义通知铃声

说明：

- 离线推送铃声不设置时，默认跟随设备的系统通知设置，可以在手机设置中，进入应用详情，手动查看或更改应用的通知铃声，或者按照下述方法自定义通知铃声。
- 离线推送自定义铃声目前仅支持：**华为、小米、FCM、OPPO** 厂商，其他厂商暂不支持。
- 响铃的时长如果不设置自定义铃声默认跟随厂商的系统通知；如果设置了自定义铃声，与铃声资源的时长有关，建议自定义铃声资源的时长和通话超时时长一致。（CallKit 默认通话超时时长是30s，可在调用 calls 接口时通过 CallParams 的 timeout 修改）。
- Andorid 8.0 及以上版本铃声是通知渠道的属性，因此铃声仅在首次创建渠道（设置 sound）有效，后续无法修改，开发阶段，修改后需卸载重装应用。
- 将铃音文件添加到本地 Android Studio 工程 res/raw 目录下。

在调用 calls 发起通话时，可通过自定义 CallParams 参数来配置 OfflinePushInfo 内容以实现离线推送铃音的自定义。

华为

调用接口 setAndroidSound() 和 setIOSSound() 接口设置铃声。

kotlin

```kotlin
val pushInfo = OfflinePushInfo()
pushInfo.title = "Mike"
pushInfo.desc = "You have a new call"
pushInfo.androidSound = "private_ring"
pushInfo.iosSound = "private_ring.mp3"

val params = TUICallDefine.CallParams()
params.offlinePushInfo = pushInfo

val list = mutableListOf<String>()
list.add("Mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, null)
```

java

```java
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.CallParams().offlinePushInfo;
pushInfo.setTitle("Mike")
pushInfo.setDesc("You have a new call")
pushInfo.setAndroidSound("private_ring");
pushInfo.setIOSSound("private_ring.mp3");

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = pushInfo;

List<String> list = new ArrayList<>();
list.add("Mike");
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, null);
```

注意

华为推送的铃声为首次创建通知渠道时的铃声，之后再次设置其他铃声无效，若想更新铃音需卸载重装，详见：华为自定义铃声 。

小米

- Android 8.0 之前，调用 setAndroidSound() 和 setIOSSound() 接口设置铃声。
- Android 8.0 之后，还需要登录小米厂商控制台并 创建 channel 并做好配置，然后在 即时通信 IM 控制台 中配置channelID，或在参数中配置。

kotlin

```
val pushInfo = OfflinePushInfo()
pushInfo.title = "Mike"
pushInfo.desc = "You have a new call"
pushInfo.androidSound = "private_ring"
pushInfo.iosSound = "private_ring.mp3"
pushInfo.androidXiaoMiChannelID = "channelId"

val params = TUICallDefine.CallParams()
params.offlinePushInfo = pushInfo

val list = mutableListOf<String>()
list.add("Mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, callback)
```

java

```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setTitle("Mike");
pushInfo.setDesc("You have a new call");
pushInfo.setAndroidSound("private_ring");
pushInfo.setIOSSound("private_ring.mp3");
pushInfo.setAndroidXiaoMiChannelID("channelId");

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = pushInfo;

List<String> list = new ArrayList<>();
list.add("Mike");
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, callback);
```

3. 在“创建channel”页面下填写信息并提交申请，如下图所示：



FCM（通知消息）

- **Android 8.0 之前，调用setAndroidSound() 和 setIOSSound() 接口设置铃声。**
- **Android 8.0 之后，需要在 即时通信 IM 控制台 中配置 channelID（或在参数中调用 androidFCMChannelID 设置），然后在登录之前配置推送通道。**

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    val nm = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    val notificationChannel =
        NotificationChannel("channelId", "channelName", NotificationManager.IMPORTANCE_HIGH)
    notificationChannel.enableLights(true)
    notificationChannel.enableVibration(true)
    notificationChannel.setShowBadge(true)
    notificationChannel.lockscreenVisibility = Notification.VISIBILITY_PUBLIC

    // "android.resource://包名/raw/private_ring"
    notificationChannel.setSound(Uri.parse("sound"), null)
    nm.createNotificationChannel(notificationChannel)
}
```

```
val pushInfo = OfflinePushInfo()
pushInfo.title = "Mike"
pushInfo.desc = "You have a new call"
pushInfo.androidSound = "private_ring"
pushInfo.iosSound = "private_ring.mp3"
pushInfo.androidFCMChannelID = "channelId"

val params = TUICallDefine.CallParams()
params.offlinePushInfo = pushInfo

val list = mutableListOf<String>()
list.add("Mike")
```

```
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, callback)
```

注意

- FCM 的数据消息可以实现类似VOIP 的通知机制，详情见：FCM VoIP（数据消息），通知消息和数据消息可以相互切换。**TUICallKit 默认所有通话请求以 VoIP 的形式推送。**
- FCM 自定义铃声或者设置 channel id 仅支持证书模式。



OPPO

- Android 8.0 之前，调用setAndroidSound() 和 setIOSSound() 接口设置铃声。
- Android 8.0 之后，先在厂商控制台 创建 OPPO 私信通道，然后在 即时通信 IM 控制台 中配置 channelID（或在参数中调用 androidOPPOChannelID 设置），最后在登录之前配置推送通道。

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    val nm = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    val notificationChannel =
        NotificationChannel("channelId", "channelName", NotificationManager.IMPORTANCE_HIGH)
    notificationChannel.enableLights(true)
    notificationChannel.enableVibration(true)
    notificationChannel.setShowBadge(true)
    notificationChannel.lockscreenVisibility = Notification.VISIBILITY_PUBLIC

    // "android.resource://包名/raw/private_ring"
    notificationChannel.setSound(Uri.parse("sound"), null)
    nm.createNotificationChannel(notificationChannel)
}
```

kotlin

```
val pushInfo = OfflinePushInfo()
pushInfo.title = "Mike"
pushInfo.desc = "You have a new call"
pushInfo.androidSound = "private_ring"
pushInfo.iosSound = "private_ring.mp3"
pushInfo.androidOPPOChannelID = "channelId"

val params = TUICallDefine.CallParams()
params.offlinePushInfo = pushInfo

val list = mutableListOf<String>()
list.add("Mike")
```

```
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, callback)
```

java

```
TUICallDefine.OfflinePushInfo pushInfo = new TUICallDefine.OfflinePushInfo();
pushInfo.setTitle("Mike");
pushInfo.setDesc("You have a new call");
pushInfo.setAndroidSound("private_ring");
pushInfo.setIOSSound("private_ring.mp3");
pushInfo.setAndroidOPPOChannelID("channelID");

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = pushInfo;

List<String> list = new ArrayList<>();
list.add("Mike");
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, callback);
```

自定义角标

目前仅华为支持收到离线消息或通话请求时生成角标，其他厂商暂不支持，详见：自定义角标 。

常见问题

1. 若在集成过程中遇到问题，请务必先查阅 插件推送-常见问题 进行自助排查。
2. 请先检查发起通话（ calls ）时，参数是否传递正确， offlinePushInfo 的 title、desc 为必填项，不能为空。
3. 条件说明：部分厂商要求必须上架应用市场才可以正常使用推送服务，详情参见下表：

厂商通道	是否需要上架	账号说明
小米	是	需要注册企业开发者账号
VIVO	是	需要注册企业开发者账号
OPPO	否	需要注册企业开发者账号
荣耀	否	需要注册企业开发者账号
华为	否	个人开发者账号即可
魅族	否	个人开发者账号即可

交流与反馈

如有问题，欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao ，进行技术交流和产品沟通。

FCM data message

最近更新时间：2025-05-22 16:39:42

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

说明：
- FCM 数据消息功能支持条件：
  - 设备网络可以访问海外网站。
  - 设备支持 GMS（Google Mobile Services）服务且服务可用（FCM是基于GMS服务的）。
  - 应用集成了 TIMPush 7.9.5668 及更高版本。
- TUICallKit 仅支持2.3及以上的版本。

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

集成效果

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

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

说明：
- 为实现上图良好的通话体验，建议您在应用中开启“通知”权限、“显示在其他应用上层”以及“后台拉起界面”权限，具体方法见：相关权限开启。
- 被叫端的来电显示策略详见：被叫端来电显示策略。

准备条件

1. 开通服务

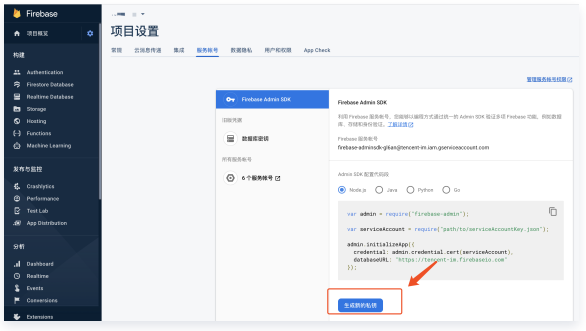
进入 IM 控制台 > 插件市场，单击**立即购买**或**免费试用**（每个应用可免费试用一次，有效期7天）。



> ⚠ **注意：**
> 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 。

2. 厂商配置

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

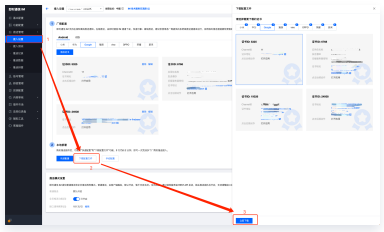
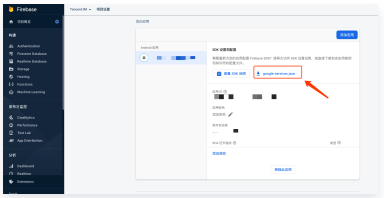
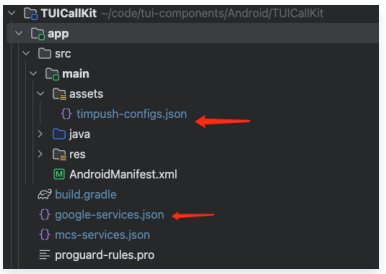
厂商推送平台	IM 控制台配置
	

快速接入

1. 下载并添加配置文件

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

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

2. 集成 TIMPush 插件

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

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

说明：

- TIMPush 需要集成 IM SDK 在**7.9.5666**版本及以上。TIMPush 的版本号及更新信息，可以在 更新日志 中进行查询。
- 您可以在 `tuicallkit-kt/build.gradle` 文件中修改 IM SDK 的版本号。

  `api "com.tencent.imsdk:imsdk-plus:latest.release"`

3. 完成工程配置

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

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

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

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

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

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

- 在项目的 app 目录下，找到并打开 `build.gradle` 文件，将应用包名修改为您的实际应用包名。

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

4. 实现自动登录

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

```
TUICore.registerEvent(TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KEY,
    TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY, new ITUINotification() {
        @Override
        public void onNotifyEvent(String key, String subKey, Map<String, Object> param) {
            if (TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_KEY.equals(key)
```

版权所有：腾讯云计算（北京）有限责任公司

```
                    && TUIConstants.TIMPush.EVENT_IM_LOGIN_AFTER_APP_WAKEUP_SUB_KEY.equals(subKey)) {
                //you need to login again to launch call activity, please implement this method by
yourself
                autoLogin();
            }
        }
    });
```

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

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

高级功能

自定义铃声

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

> **注意：**
> - 替换掉该铃声后，不论应用在前台、后台或者离线，铃声都会是替换后的铃声。
> - 替换后其他厂商手机收到邀请后的铃声也会是替换后的铃声。

常见问题

1. 应用杀死后无法弹出来电UI

- 确认收到了推送，收不到推送需要确认下 IM 控制台是否正确上传证书。参见文档上述**快速接入**的第一步，看下是否添加正确。
- 确认控制台选择了 FCM 数据消息，对照上述**准备条件**的第二步；
- 确认收到数据消息，过滤日志（关键字：TIMPush），检查下述日志有打印（如果收不到消息，可以使用 IM 控制台的 排查工具 查看原因）；

```
5 TIMPushFCMMsgService    Message data payload: {IMDesc=You have a new call, IMSound=phone_ringing, IMTitle=258, ext={"timPushFeatures":{"fcmNotificationType":1,"fcmPushType":0},"entity":{"action":2}},
```

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

交流与反馈

如有问题，欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao，进行技术交流和产品沟通。

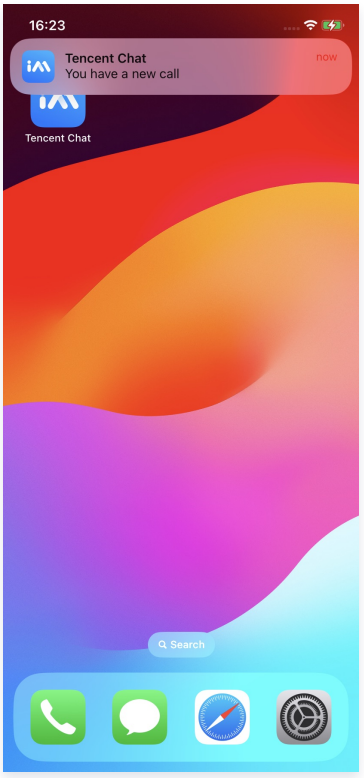
iOS
Notification

最近更新时间：2025-04-07 10:51:22

TUICallKit 组件支持通过接入推送插件，实现离线推送的功能。TUICallKit 为您提供了两种集成方式：TIMPush 插件（付费）和 自集成推送（免费），我们更推荐您使用 TIMPush 推送插件来实现离线推送，TIMPush 推送插件有以下优势：

- 接入周期短，预计全厂商接入仅需要 30 分钟。
- 支持数据统计和链路追踪，方便您查看推送触达率、点击率和转化率等各类指标。
- 支持全员/标签推送，方便您将营销广告、通知、新闻资讯等内容推送给所有用户或指定群体。
- 支持 uni-app 和 Flutter 等跨平台框架。

集成效果

锁屏时的效果	在后台时的效果
	

配置离线推送

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

步骤1: 集成 TIMPush 组件

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

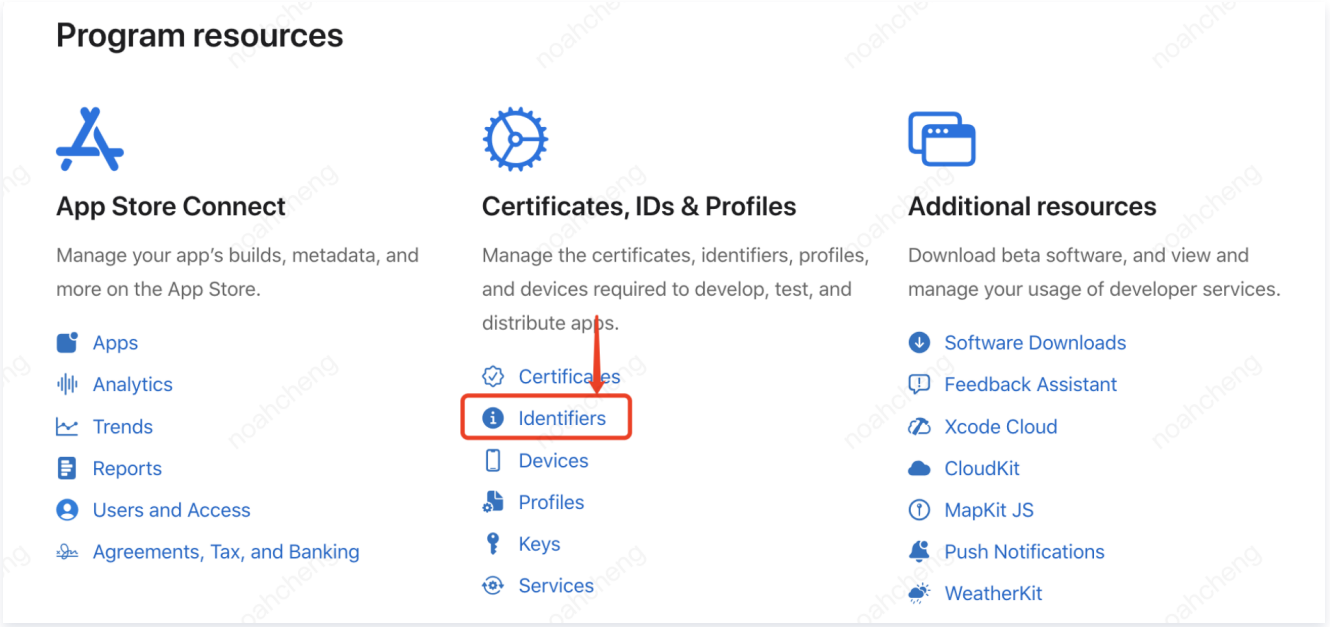
```
target 'YourAppName' do
  # Uncommment the next line if you're using Swift or would like to use dynamic frameworks
  use_frameworks!
  use_modular_headers!

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

2. 执行以下命令，安装 TIMPush 组件。

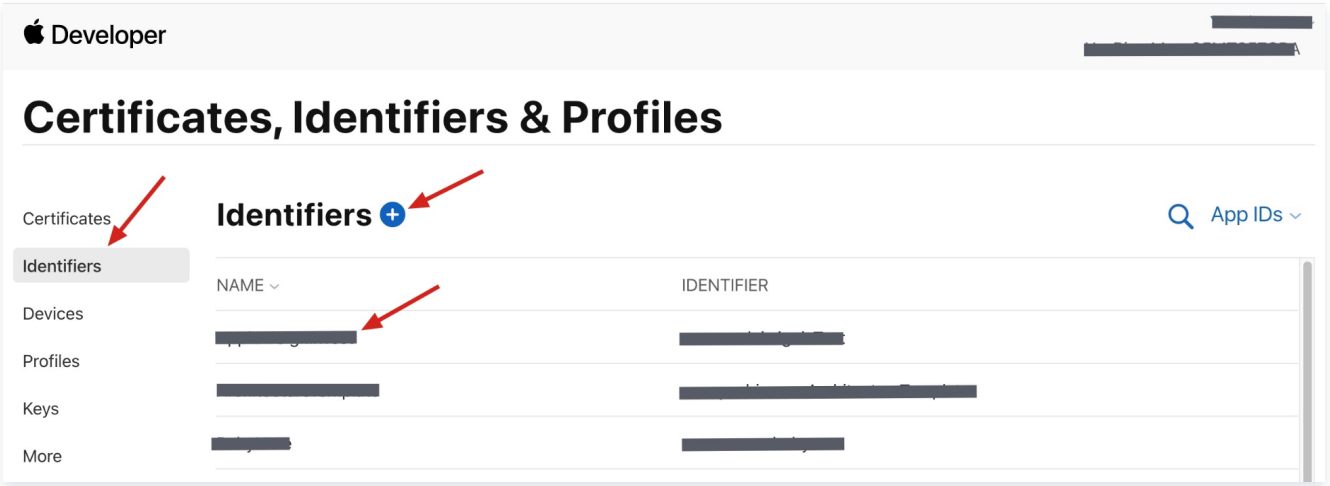
```
pod install
# 如果无法安装 TUIKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。
pod repo update
```

步骤2: 开启 App 远程推送

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



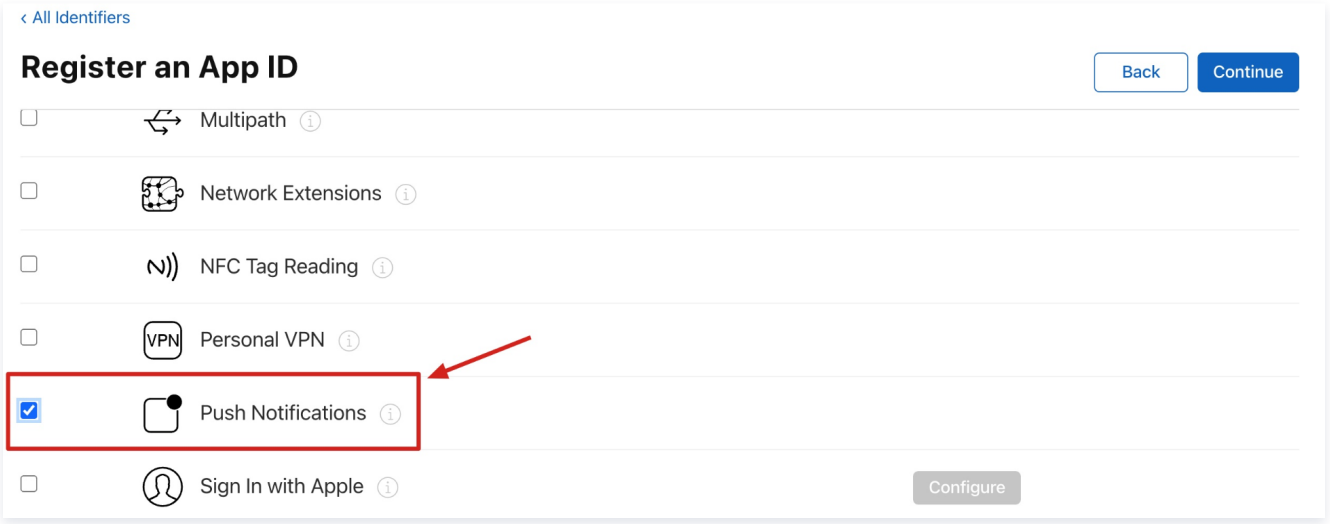
2. 单击 Identifiers 右侧的 “+”。



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

> **说明：**
> 需要注意的是，您 App 的 `Bundle ID` 不能使用通配符 *，否则将无法使用远程推送服务。

4. 勾选 **App IDs**，单击 **Continue** 进行下一步。
5. 选择 **App**，单击 **Continue** 进行下一步。
6. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。
7. 勾选 **Push Notifications**，开启远程推送服务。

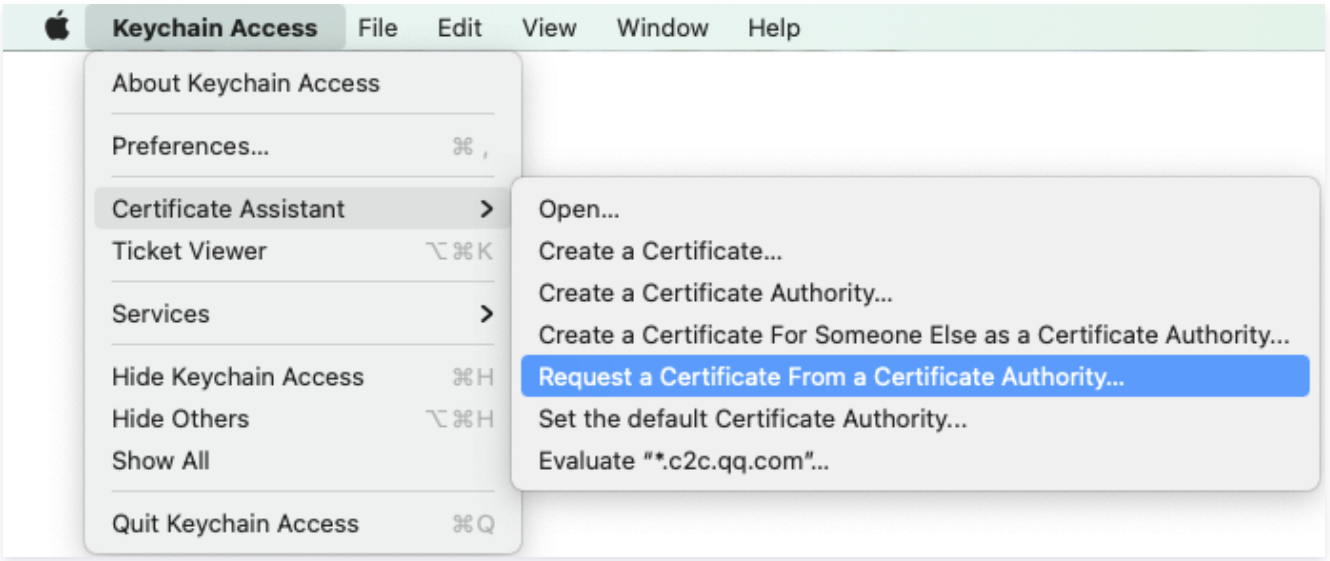


步骤3：生成证书

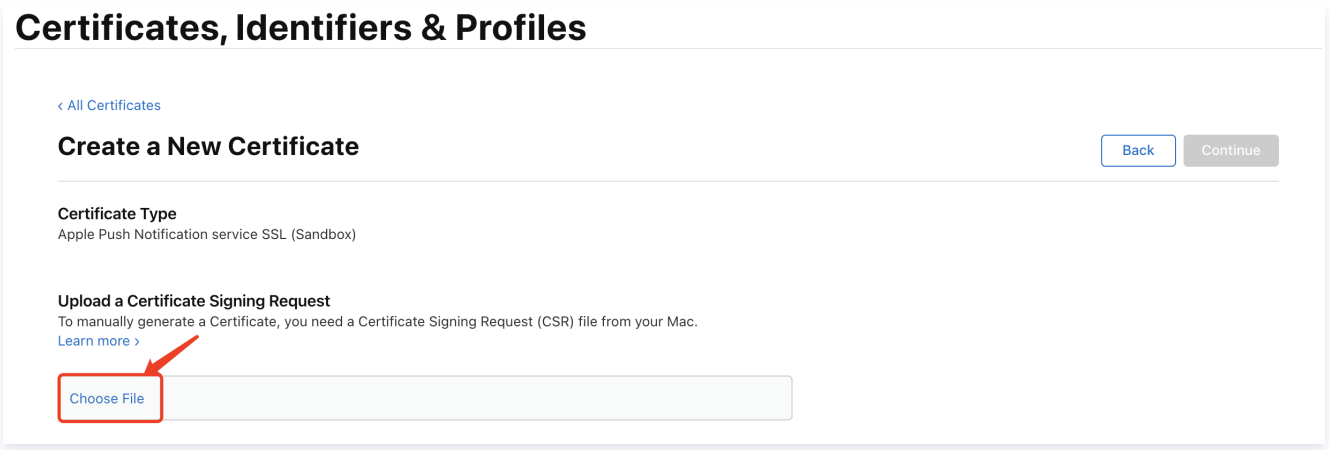
1. 选中您的 AppID，找到 Push Notifications 配置项，选择 **Configure**。



2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate` ，分别用于开发环境（ `Development` ）和生产环境（ `Production` ）的远程推送证书。
3. 我们先选择开发环境（ `Development` ）的 **Create Certificate**，系统将提示我们需要一个 `Certificate Signing Request（CSR）` 。
4. 在 Mac 上打开钥匙串访问工具（Keychain Access），在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（ `Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority` ）。



5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择 “存储到磁盘（Saved to disk）”，单击继续，系统将生成一个 `*.certSigningRequest` 文件。
6. 返回上述 第3步 中 `Apple Developer` 网站刚才的页面，单击 “Choose File” 上传生成的 `*.certSigningRequest` 文件。



7. 单击 **Continue**，即可生成推送证书。
8. 单击 **Download** 下载开发环境的 `Development SSL Certificate` 到本地。
9. 再次按照上述步骤1 – 8，将生产环境的 `Production SSL Certificate` 下载到本地。

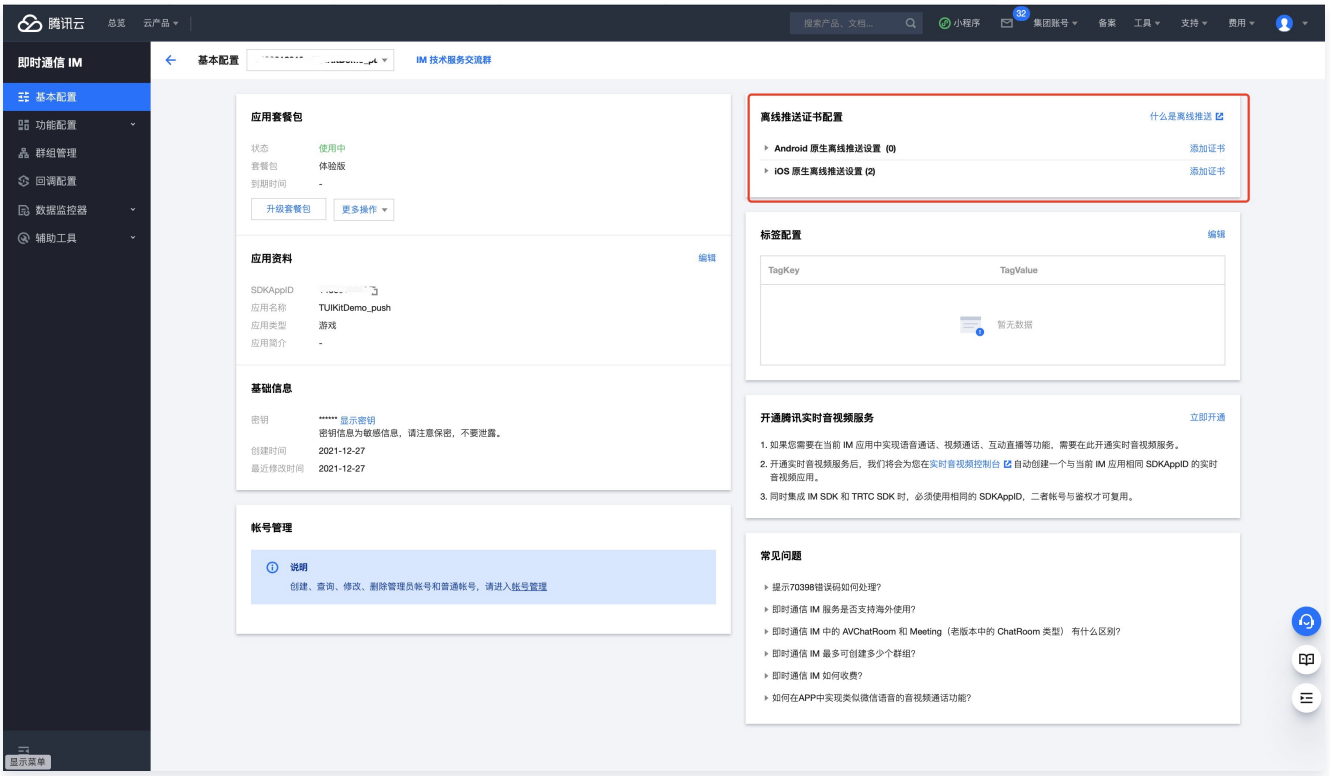
> **说明：**
> 生产环境的证书实际是开发(`Sandbox`)+生产(`Production`)的合并证书，可以同时作为开发环境和生产环境的证书使用。

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

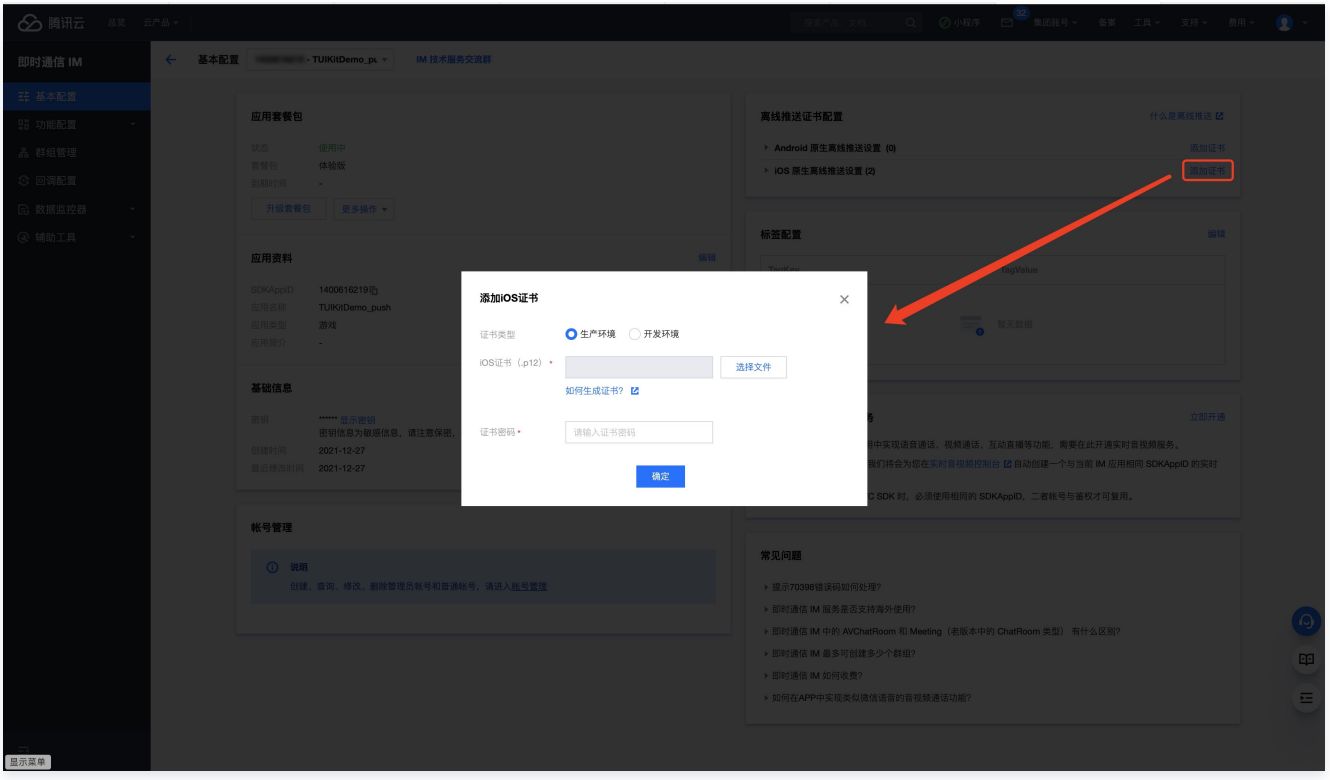
> **说明：**
> 保存 `P12` 文件时，请务必要为其设置密码。

步骤4：上传证书到 IM 控制台

1. 登录 即时通信 IM 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**。

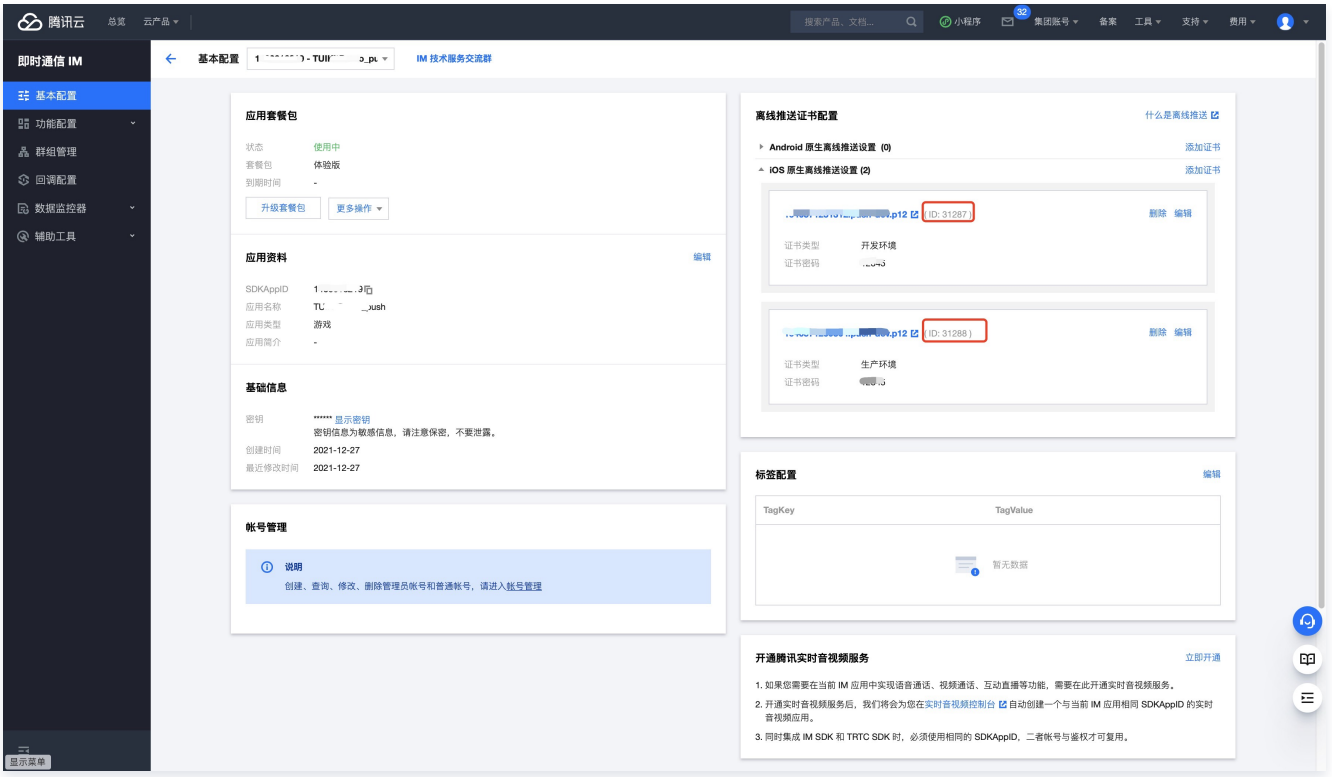


说明：

- 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。

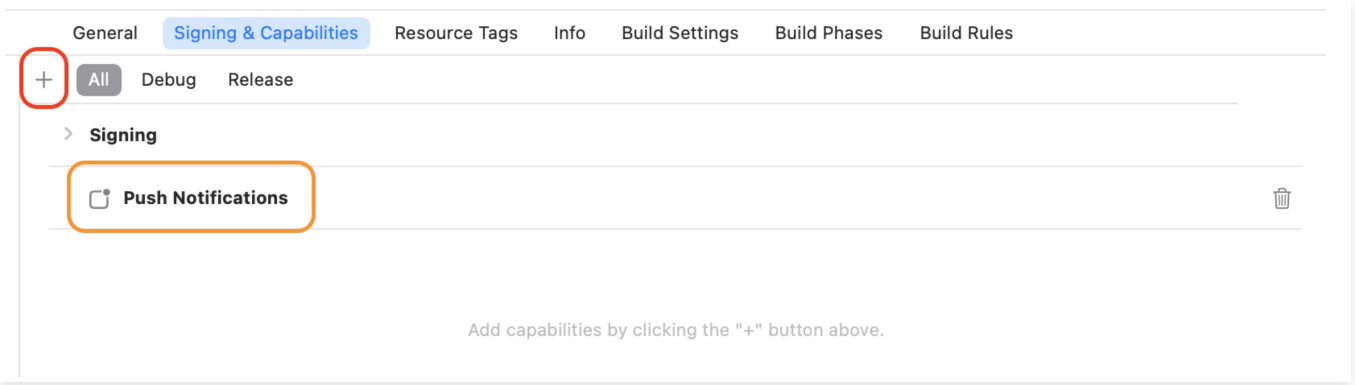
- 上传的 p12 证书必须是自己申请的真实有效的证书。

5. 待推送证书信息生成后，记录证书的 ID。



步骤5：完成工程配置

要在应用程序中添加所需的权限，请在 Xcode 项目中启用推送通知功能。
打开 Xcode 项目，在 **Project** > **Target** > **Capabilities** 页面中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黄框所示：



步骤6：配置推送参数

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

Swift

```
import TIMPush

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

Objective-C

```
#pragma mark - TIMPush

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

步骤7：拨打离线推送电话

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

Swift

```
import TUICallKit_Swift
import RTCRoomEngine

let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "您有一个新的通话"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
pushInfo.androidHuaWeiCategory = "IM"

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

TUICallKit.createInstance().calls(userIdList: ["123456"], callMediaType: .audio, params: params) {

} fail: { code, message in

}
```

Objective-C

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

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
```

```
    callParams.timeout = 30;
    return callParams;
}

- (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = @"您有一个新的通话";
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}

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

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

}];
```

高级功能：持续震动、响铃

APNs（Apple Push Notification service）推送可以指定推送声音文件等功能，却并不能持续的震动、响铃，通知持续时间较短，在音视频通话场景下容易错过来电信息。您可以使用我们的 Demo 体验 APNs 持续震动和响铃功能 。

1. 响铃

APNs 推送可以在调用 call、calls、groupCall 接口拨打电话时设置 params 的 TUIOfflinePushInfo 中的 iOSSound 字段， iOSSound 传语音文件名。

> ⚠ 注意：
> - 如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀名）设置给 iOSSound。
> - 铃声时长应小于30s。

Objective-C

```
[[TUICallKit createInstance] call:@"mike 的 id" params:[self getCallParams]
callMediaType:TUICallMediaTypeVideo];

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
```

```
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}

+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```

Swift

```
let params = TUICallParams()
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = "TUICallKit"
pushInfo.desc = "TUICallKit.have.new.invitation"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
pushInfo.androidHuaWeiCategory = "IM"
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: params) {

} fail: {
    code, message in
}
```

2. 持续震动

Notification Service Extension（简称为 Extension）是 iOS 中用于拦截、修改推送通知内容的 Extension，可以借助 Extension 实现长时间震动。
主应用和 Extension 两独立的进程， 当收到 APNs 推送后主应用不会激活，Extension 会被激活，在 Extension 开启持续震动，当应用切到前台后需要使用进程间通信的方式告知 Extension，关闭震动和铃声。

2.1 开启 mutable-content

使用 `Notification Service Extension` 实现持续的震动，首先需要在控制台开启 `mutable-content` 开关。



2.2 创建 Extension 工程

在 Xcode 项目中，选择 `Editor > Add Target > Notification Service Extension` 来创建一个 Extension。新创建的 Extension Target 需要配置 Apple 证书，具体方法可以 参考这里 。
新建的 Extension 包含名为 **NotificationService** 的源码文件，内容如下：

```
// NotificationService.swift
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler:
@escaping (UNNotificationContent) -> Void) {
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationContent)
        if let bestAttemptContent = bestAttemptContent {
            // Modify the notification content here...
            bestAttemptContent.title = "\(bestAttemptContent.title) [modified]"
            contentHandler(bestAttemptContent)
        }
    }

    override func serviceExtensionTimeWillExpire() {
        // Called just before the extension will be terminated by the system.
```

```
        // Use this as an opportunity to deliver your "best attempt" at modified content, otherwise the
original push payload will be used.
        if let contentHandler = contentHandler, let bestAttemptContent =  bestAttemptContent {
            contentHandler(bestAttemptContent)
        }
    }
}
```

- **didReceive**：当收到推送后 Extension 被激活，执行 didReceive 回调， 用户可以在 didReceive 中完成修改推送信息的工作， 当完成修改后调用 contentHandler() 方法， Extension 执行结束。
- **serviceExtensionTimeWillExpire**：当收到推送后 Extension 被激活， Extension 超过30s没有响应（没有执行 contentHandler() 方法），serviceExtensionTimeWillExpire 回调方法会被调用， 并在该方法内调用 contentHandler() 方法， Extension 执行结束。

2.3 收到推送开启震动

1. 当收到推送后 Extension 被激活，执行 didReceive 回调时，开启震动。
2. 当收到 serviceExtensionTimeWillExpire 后需要关闭震动铃声。

修改后的代码如下：

```
// NotificationService.swift
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler:
@escaping (UNNotificationContent) -> Void) {
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationContent)
        if let bestAttemptContent = bestAttemptContent {
            contentHandler(bestAttemptContent)
        }

        // start ringing, VibratorFeature.start() needs to be implemented by yourself
        VibratorFeature.start()
    }

    override func serviceExtensionTimeWillExpire() {
        if let contentHandler = contentHandler, let bestAttemptContent =  bestAttemptContent {
            /// If there is no response for more than 30 seconds, the vibration and ringing function
will be turned off.
            // stop ringing, RingingFeature.shared.stop() needs to be implemented by yourself
            VibratorFeature.stop()
            contentHandler(bestAttemptContent)
        }
    }
}
```

> ⚠ **注意：**
> 上述代码中使用的开启震动的方法 `VibratorFeature.start()` 、 `VibratorFeature.stop()` 需要客户自行实现。

2.4 在主应用中关闭震动

收到推送，开始震动和响铃，当用户进入到 APP 后应该关闭 Extension 中的震动。主应用和 Extension 分别是两个独立的进程，可以使用 CFNotificationCenterGetDarwinNotifyCenter 发送传递信息。当 APP 进入前台时， 主应用向 Extension 发送一个通知，Extension 监听通知， 当 Extension 收到通知后关闭震动。

修改后的代码如下：

```
// iOS 13 之前的版本通过 AppDelegate applicationWillEnterForeground 监测前后台状态
// AppDelegate.swift
class AppDelegate: UIResponder, UIApplicationDelegate {
    func applicationWillEnterForeground(_ application: UIApplication) {
      ……
      sendStopRingingToExtension()
    }
    func sendStopRingingToExtension() {
        CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter(),
                                             CFNotificationName("APNsStopRinging" as CFString), nil,
nil, true)
    }
}

// iOS 13 及以后版本通过 SceneDelegate sceneWillEnterForeground 监测前后台状态
// SceneDelegate.swift
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
    func sceneWillEnterForeground(_ scene: UIScene) {
        ……
        sendStopRingingToExtension()
    }

    func sendStopRingingToExtension() {
        CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter(),
                                             CFNotificationName("APNsStopRinging" as CFString), nil,
nil, true)
    }
}
```

```
// NotificationService.swift
import UserNotifications

class NotificationService: UNNotificationServiceExtension {
    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler:
@escaping (UNNotificationContent) -> Void) {
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationContent)
        if let bestAttemptContent = bestAttemptContent {
            contentHandler(bestAttemptContent)
        }

        // start ringing, VibratorFeature.start() needs to be implemented by yourself
        VibratorFeature.start()
        registerObserver()
    }

    override func serviceExtensionTimeWillExpire() {
        if let contentHandler = contentHandler, let bestAttemptContent =  bestAttemptContent {
            /// If there is no response for more than 30 seconds, the vibration and ringing function
will be turned off.
            // stop ringing, RingingFeature.shared.stop() needs to be implemented by yourself
            VibratorFeature.stop()
```

```
            contentHandler(bestAttemptContent)
        }
    }

    private func registerObserver() {
        CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
                                        Unmanaged.passUnretained(self).toOpaque(), { center, pointer,
name, _, userInfo in
            VibratorFeature.stop()
        }, "APNsStopRinging" as CFString, nil, .deliverImmediately)
    }
}
```

常见问题

1. 收不到推送，且后台报错 bad devicetoken?

Apple 的 deviceToken 与当前编译环境有关。如果 登录 IMSDK 后上传 deviceToken 到腾讯云所使用的证书 ID 和 token 不一致，就会报错。

- 如果使用的是 Release 环境编译，则 `- application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是发布环境的 token，此时 businessID 需要设置生产环境的 证书 ID 。
- 如果使用的是 Debug 环境编译，则 `- application:didRegisterForRemoteNotificationsWithDeviceToken:` 回调返回的是开发环境的 token，此时 businessID 需要设置开发环境的 证书 ID 。

Swift

```
let config = V2TIMAPNSConfig()
// 用户自己到苹果注册开发者证书，在开发者帐号中下载并生成证书(p12 文件)，将生成的 p12 文件传到腾讯证书管理控制台，
// 控制台会自动生成一个证书 ID，将证书 ID 传入以下 businessID 参数中。
config.businessID = sdkBusiId
config.token = deviceToken
V2TIMManager.sharedInstance().setAPNS(config) {

} fail: { code, message in

}
```

Objective-C

```
V2TIMAPNSConfig *config = [[V2TIMAPNSConfig alloc] init];
// 用户自己到苹果注册开发者证书，在开发者帐号中下载并生成证书(p12 文件)，将生成的 p12 文件传到腾讯证书管理控制台，
// 控制台会自动生成一个证书 ID，将证书 ID 传入以下 businessID 参数中。
config.businessID = sdkBusiId;
config.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:config succ:^{

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

}];
```

2. iOS 开发环境下，注册偶现不返回 deviceToken 或提示 APNs 请求 token 失败?

此问题现象是由于 APNs 服务不稳定导致的，可尝试通过以下方式解决：

1. 给手机插入 SIM 卡后使用 4G 网络测试。
2. 卸载重装、重启 App、关机重启后测试。
3. 打生产环境的包测试。
4. 更换其它 iOS 系统的手机测试。

VoIP

最近更新时间：2025-04-14 15:23:22

VoIP（Voice over IP）Push 是 Apple 提供的用于响应 VoIP calls 的通知机制。结合 Apple 提供的 PushKit.framework 和 CallKit.framework 能够实现系统级的通话效果。

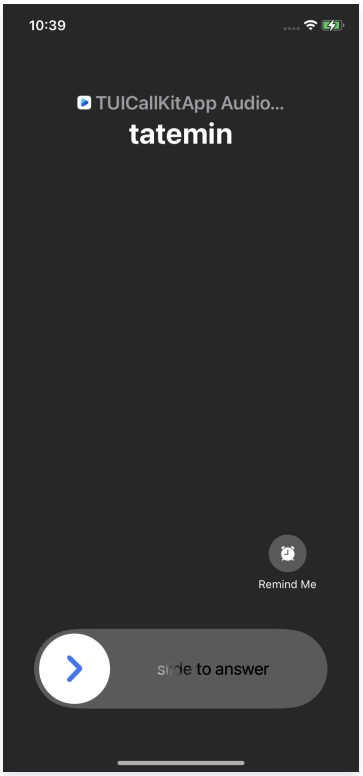
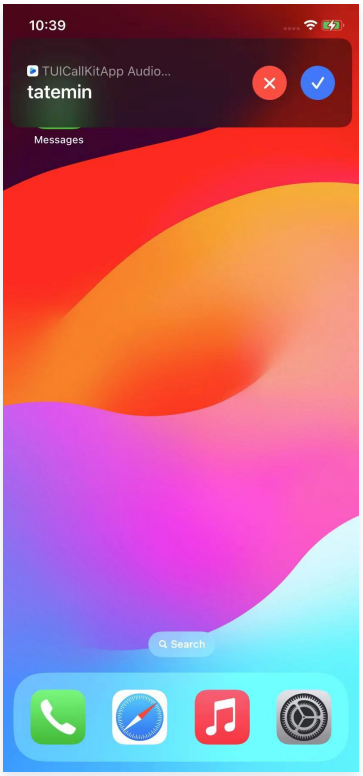
说明：

- Apple 要求，VoIP Push 在 iOS 13.0 以后需要搭配 CallKit.framework 使用，否则 App 运行后会 crash。
- VoIP Push 无法复用 APNs 普通推送证书，需要单独在苹果开发者网站上 申请 VoIP Push 证书 。

禁止：

由于 VoIP 插件使用了 Apple CallKit.framework，无法在 APP Store 的中国大陆地区上架！

集成效果

锁屏时的效果	应用在后台的效果	应用在后台展开的效果
		

准备条件

1. 开通服务

进入 IM 控制台 > 插件市场，单击**立即购买**或**免费试用**（每个应用可免费试用一次，有效期7天）。



注意：

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 。

2. 厂商配置

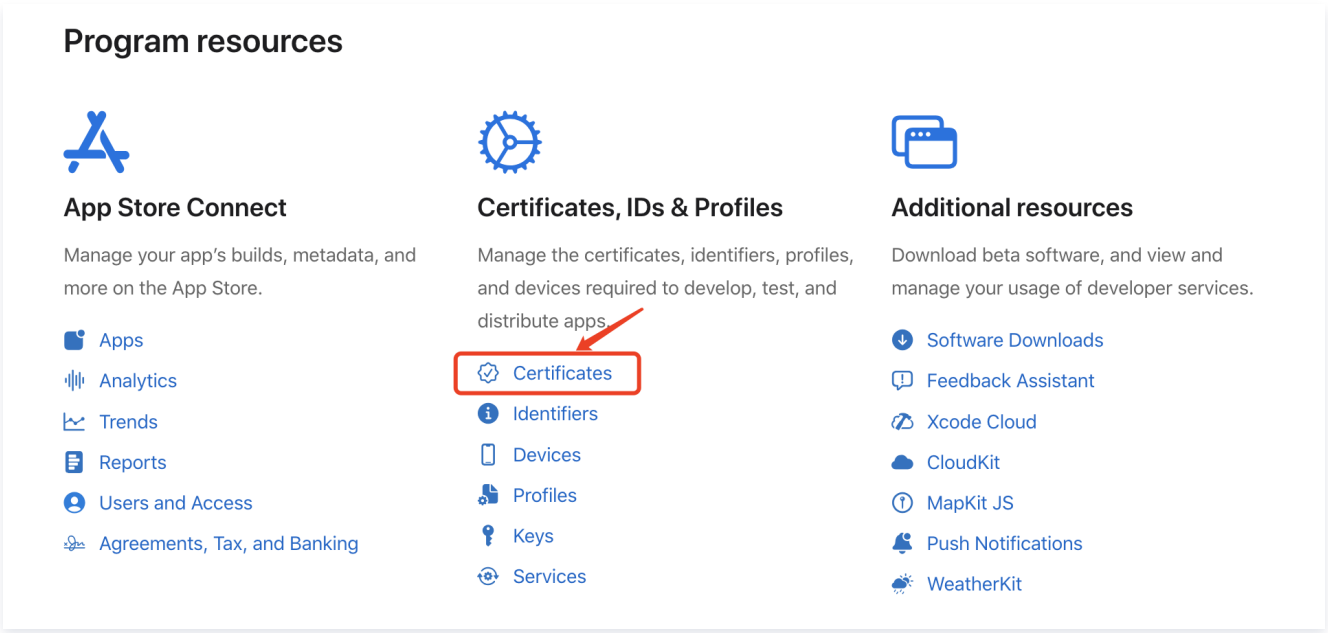
如果想要接收 VoIP Push，需要遵循如下几个步骤：

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

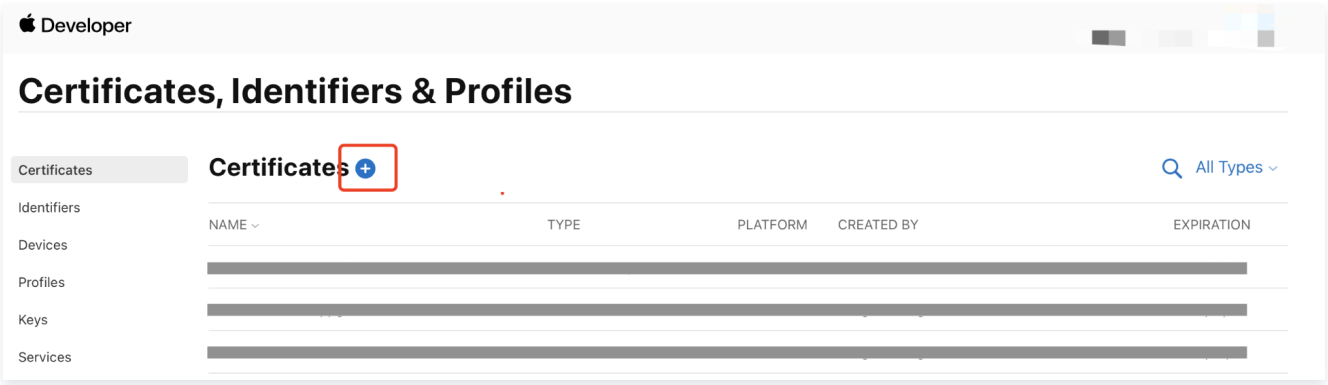
步骤1：申请 VoIP Push 证书

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

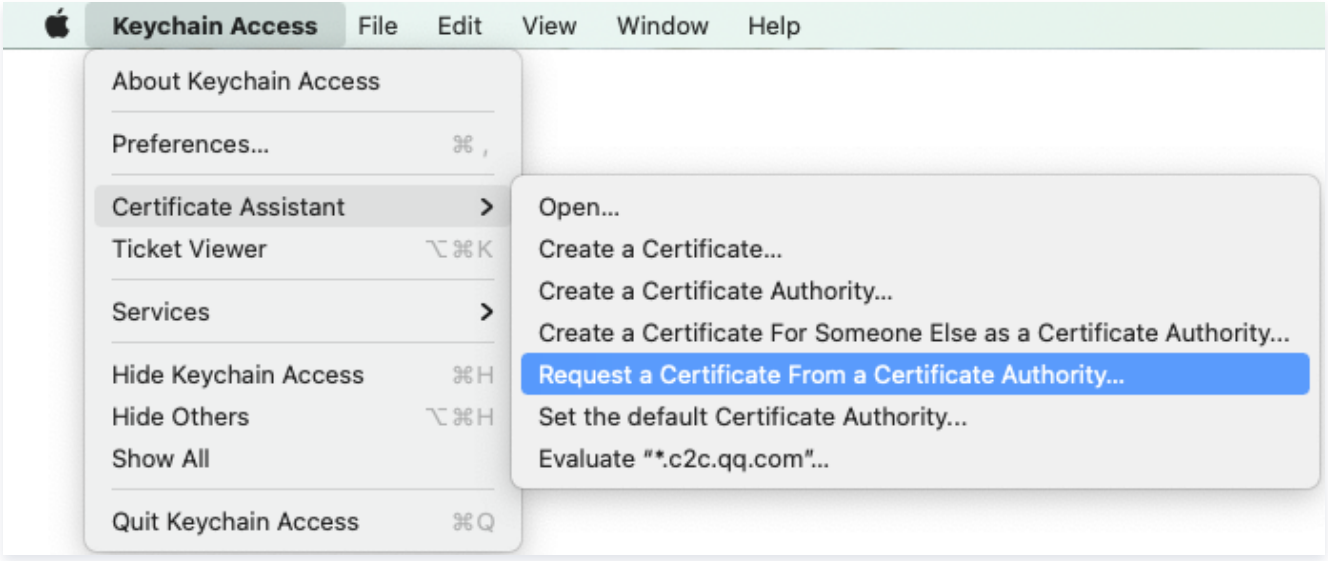
1. 登录 苹果开发者中心 网站，单击 **Certificates, IDs & Profiles** 选项卡中的 **Certificates**，进入 **Certificates, Identifiers & Profiles** 页面。



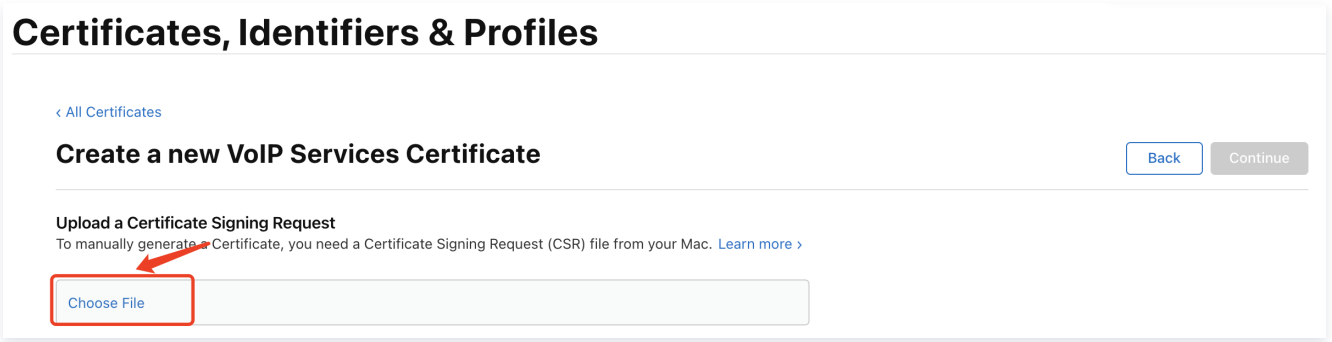
2. 单击 Certificates 右侧的+。



3. 在 **Create a New Certificate** 选项卡中，选择 **VoIP Services Certificate**，并单击 **Continue**。
4. 在 **Select an App ID for your VoIP Service Certificate** 选项卡中，选择您当前的 App 的 BundleID，并单击 **Continue**。
5. 紧接着，系统提示我们需要一个 **Certificate Signing Request（CSR）**。
6. 我们接下来制作 CSR 文件。首先在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



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



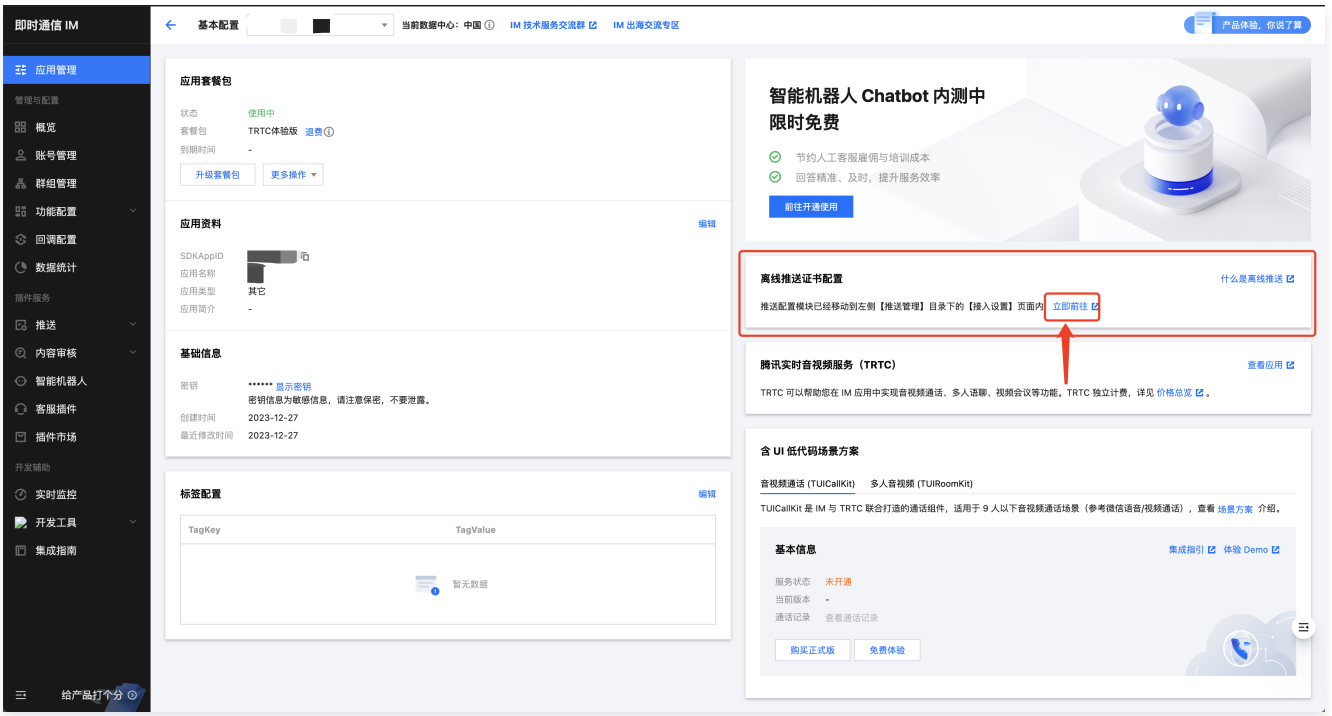
8. 单击 Continue 后生成证书，点击 Download 下载对应的证书到本地。
9. 双击打开刚才下载的 `voip_services.cer`，系统会将其导入钥匙串中。
10. 打开钥匙串应用，在**登录** > **我的证书**，右键导出刚创建的 VoIP Services 的 `P12` 文件。

> **说明：**
> 保存 `P12` 文件时，请务必要为其设置密码。

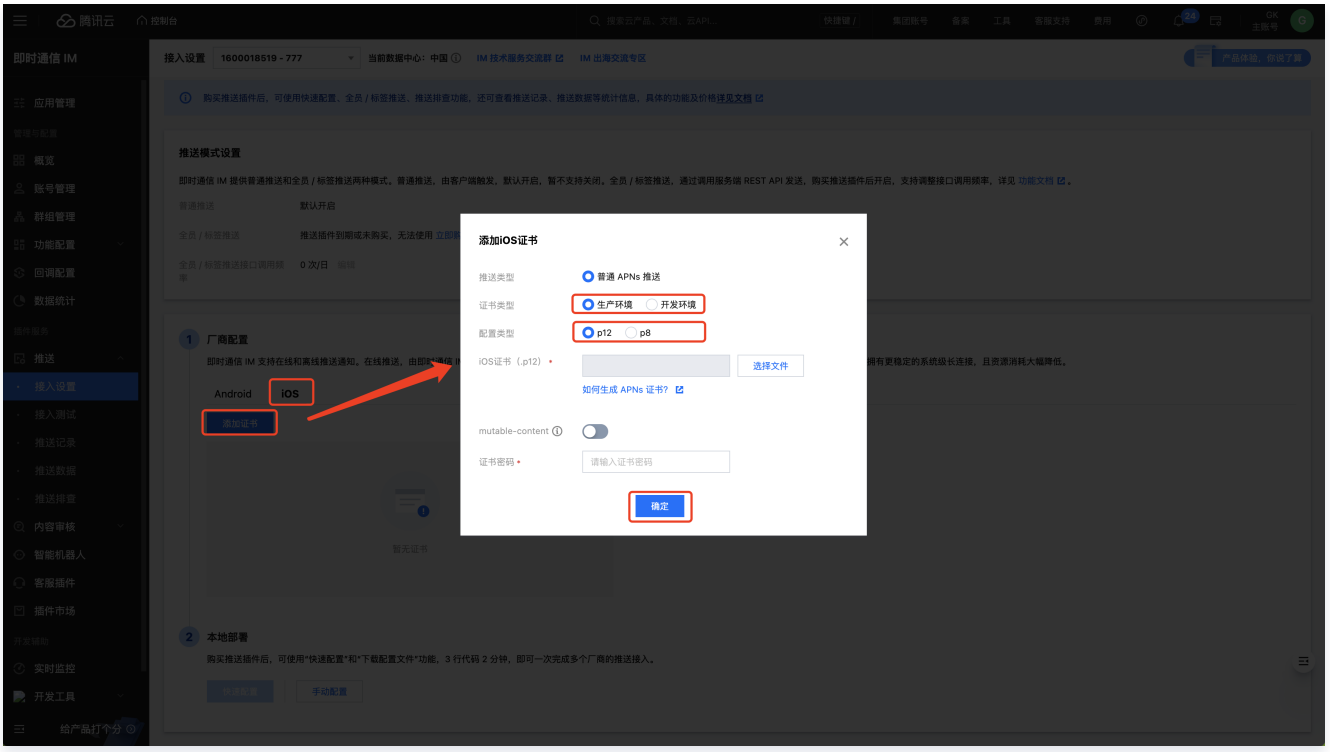
步骤2：上传证书到 IM 控制台

打开 IM 控制台，选择您创建的 IM 应用，并按照如下步骤上传证书：

1. 选择您的 IM 应用，进入应用的基础配置页面，在**离线推送证书配置**选项卡中，单击**立即前往**。



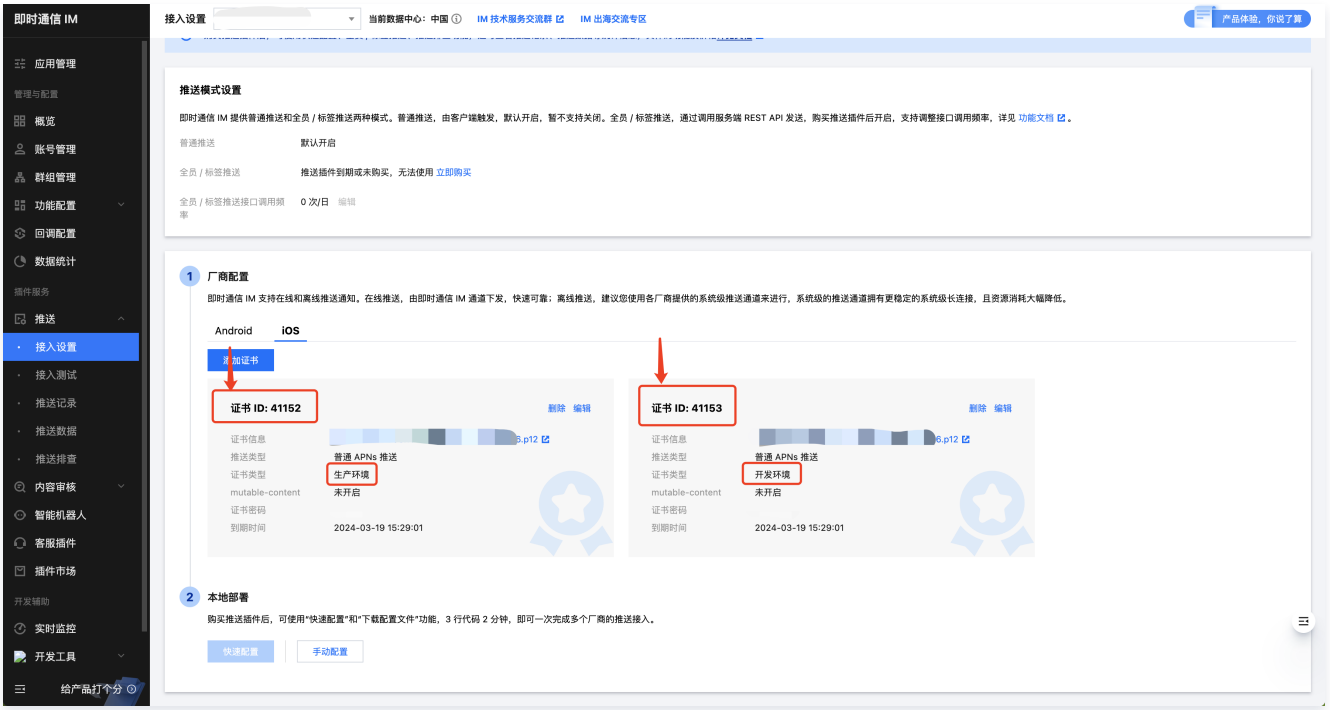
2. 在**厂商配置**中，切换到 **iOS**，点击**添加证书**按钮，然后在悬浮页面中选择证书类型，上传 iOS 证书（p.12），设置证书密码，单击**确认**。



说明：

- 添加证书时，推送类型默认为 APNS，不影响 VoIP 证书的上传和使用。
- VoIP Push 证书本身不区分生产环境和测试环境，生产环境和开发环境使用的是同一份 VoIP Push 证书，请分别上传。
- 上传证书名最好使用全英文（尤其不能使用括号等特殊字符）。
- 上传证书需要设置密码，无密码收不到推送。
- 发布 App Store 的证书需要设置为生产环境，否则无法收到推送。
- 上传的 p12 证书必须是自己申请的真实有效的证书。

3. 上传完成后，记录不同环境下的证书 ID。

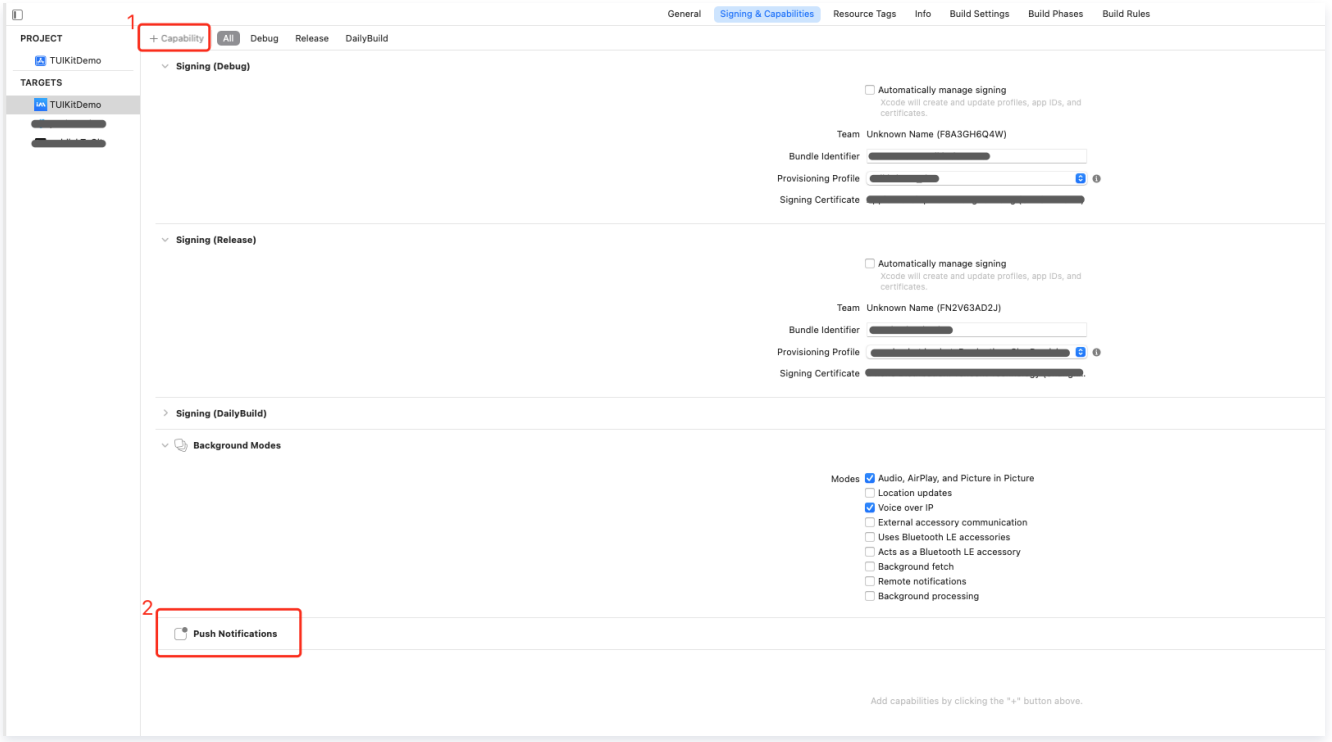


> **说明：**
> 开发环境和生产环境下的证书 ID 要严格区分，并在 步骤4：接入TUICallKitVoIPExtension 组件 时根据实际环境填写。

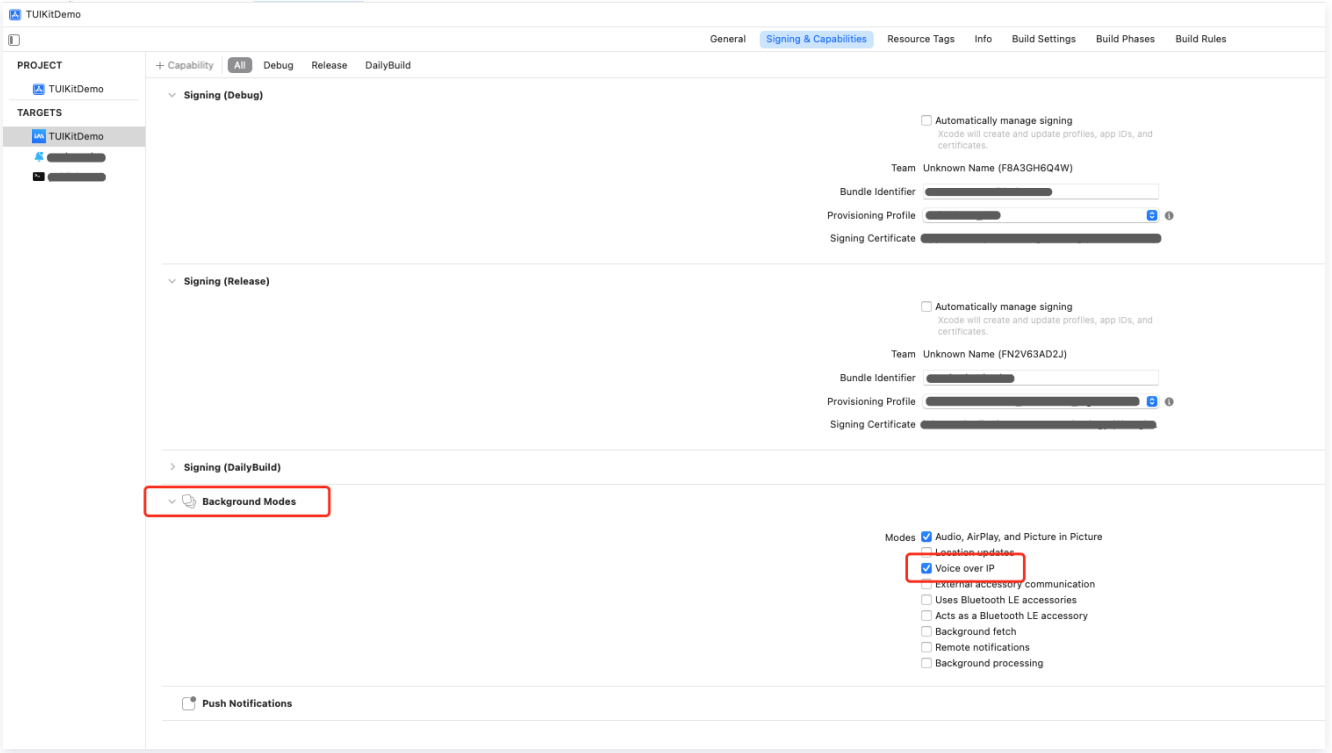
功能接入

步骤1：完成工程配置

1. 如下图，确认您工程的 Capability 中是否添加 Push Notification 能力。



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



步骤2：接入 TUICallKitVoIPExtension 组件

使用 CocoaPods 导入组件，具体步骤如下：

1. 在您的 `Podfile` 文件中添加以下依赖。

```
pod 'TUIVoIPExtension'
```

> ⚠ **注意：**
> 请在您的 `Podfile` 文件中确保为 `TUICallKit_Swift` 和 `TUICallKitVoIPExtension` 组件指定相同的 `Subspec` 。

2. 执行以下命令，安装组件。

```
pod install
```

3. 上报 步骤 2：记录的证书 ID 。

Swift

```
import TUIVoIPExtension

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

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

    return true
}
```

Objective-C

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

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {

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

    return YES;
}
```

> ⓘ **说明：**
> 如果无法安装 TUICallKit 最新版本，执行以下命令更新本地的 CocoaPods 仓库列表。
>
> ```
> pod repo update
> ```

拨打 VoIP 通话

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

Swift

```
import TUICallKit_Swift
import RTCRoomEngine

let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = ""
pushInfo.desc = "您有一个新的通话"
pushInfo.iOSPushType = .voip
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
pushInfo.androidHuaWeiCategory = "IM"

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

TUICallKit.createInstance().calls(userIdList: ["123456"], callMediaType: .audio, params: params) {

} fail: { code, message in

}
```

Java

```
TUICallDefine.OfflinePushInfo offlinePushInfo = new TUICallDefine.OfflinePushInfo();
offlinePushInfo.setTitle("");
offlinePushInfo.setDesc("You have receive a new call");
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
offlinePushInfo.setAndroidOPPOChannelID("tuikit");
offlinePushInfo.setIgnoreIOSBadge(false);
offlinePushInfo.setIOSSound("phone_ringing.mp3");
offlinePushInfo.setAndroidSound("phone_ringing"); //Note: don't add suffix
//VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
offlinePushInfo.setAndroidVIVOClassification(1);
//FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
offlinePushInfo.setAndroidFCMChannelID("fcm_push_channel");
//Huawei message type
offlinePushInfo.setAndroidHuaWeiCategory("IM");
//IOS push type: if you want user VoIP, please modify type to TUICallDefine.IOSOfflinePushType.VoIP
offlinePushInfo.setIOSPushType(TUICallDefine.IOSOfflinePushType.VoIP);

TUICallDefine.CallParams params = new TUICallDefine.CallParams();
params.offlinePushInfo = offlinePushInfo;
```

```
List<String> list = new ArrayList<>();
list.add("mike")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Video, params, null);
```

Dart

```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
 offlinePushInfo.title = "Flutter TUICallKit";
 offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
 offlinePushInfo.ignoreIOSBadge = false;
 offlinePushInfo.iOSSound = "phone_ringing.mp3";
 offlinePushInfo.androidSound = "phone_ringing";
 offlinePushInfo.androidOPPOChannelID = "Flutter TUICallKit";
 offlinePushInfo.androidVIVOClassification = 1;
 offlinePushInfo.androidFCMChannelID = "fcm_push_channel";
 offlinePushInfo.androidHuaWeiCategory = "Flutter TUICallKit";
 offlinePushInfo.iOSPushType = TUICallIOSOfflinePushType.VoIP;

 TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);

 TUICallKit.instance.calls(['vince'], TUICallMediaType.audio, params);
```

Objective-C

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

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}

- (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = @"您有一个新的通话";
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeVoIP;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```

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

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

}];
```

通过系统通话记录拨打通话

如果需要实现点击**系统电话 - 最近通话**列表中的通话记录，直接发起单人音视频通话，您需要在 Application 的生命周期回调函数使用 **TUICallKitVoIPExtension** 组件中的 **callWith** 接口，具体示例如下：

> **说明：**
> - 仅支持单人音视频通话的直接拨打。
> - 登录账户必须为同一个账户。

1. 在iOS 13（及以后版本）上，使用了 **SceneDelegate**，且最低兼容版本到iOS 13以下，需分别在 **AppDelegate** 和 **SceneDelegate** 中实现如下方法。

Swift

```
import TUIVoIPExtension

/// AppDelegate 中的实现，iOS 13 以前版本的处理
func application(_ application: UIApplication, continue userActivity: NSUserActivity,
restorationHandler: @escaping ([any UIUserActivityRestoring]?) -> Void) -> Bool {
    TUIVoIPExtension.call(with: userActivity)
    return true
}

/// SceneDelegate 中的实现
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions:
UIScene.ConnectionOptions) {
    for userActivity in connectionOptions.userActivities {
        TUIVoIPExtension.call(with: userActivity);
     }
}

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

Objective-C

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

/// AppDelegate 中的实现，iOS 13 以前版本的处理
- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUserActivity
*)userActivity restorationHandler:(nonnull void (^)(NSArray<id<UIUserActivityRestoring>> *
_Nullable))restorationHandler {
    [TUIVoIPExtension callWith:userActivity];
    return YES;
}

/// SceneDelegate 中的实现
```

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)sessionptions:
(UISceneConnectionOptions *)connectionOptions {
    [connectionOptions.userActivities enumerateObjectsUsingBlock:^(NSUserActivity * _Nonnull
userActivity, BOOL * _Nonnull stop) {
        [TUIVoIPExtension callWith:userActivity];
    }];
}

- (void)scene:(UIScene *)scene continueUserActivity:(NSUserActivity *)userActivity {
    [TUIVoIPExtension callWith:userActivity];
}
```

2. 在iOS 13（及以后版本）上，未使用**SceneDelegate**，只需要在 **AppDelegate** 实现如下方法。

Swift

```
import TUIVoIPExtension

func application(_ application: UIApplication, continue userActivity: NSUserActivity,
restorationHandler: @escaping ([any UIUserActivityRestoring]?) -> Void) -> Bool {
    TUIVoIPExtension.call(with: userActivity)
    return true
}
```

Objective-C

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

- (BOOL)application:(UIApplication *)application continueUserActivity:(nonnull NSUserActivity
*)userActivity restorationHandler:(nonnull void (^)(NSArray<id<UIUserActivityRestoring>> *
_Nullable))restorationHandler {
    [TUIVoIPExtension callWith:userActivity];
    return YES;
}
```

常见问题

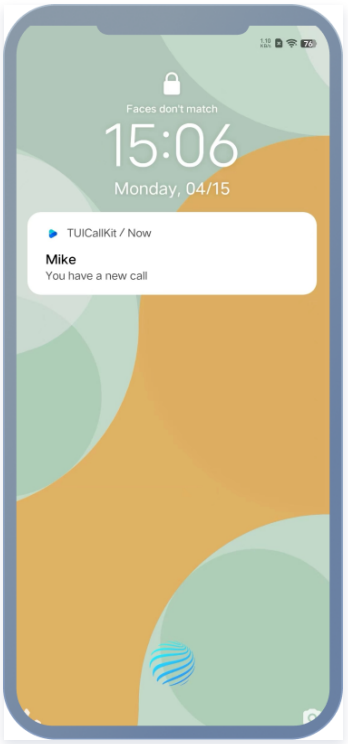
获取不到 VoIP Push，如何处理?

1. 首先检查下 App 的运行环境和证书的环境是否一致，证书 ID 是否匹配，如果不一致，无法收到推送。
2. 请确认当前您登录的账号是否处于离线状态：按 home 键切后台、登录后主动杀进程退出。VoIP Push 目前只支持离线状态下的推送。
3. 检查 步骤3：完成工程配置 是否正确。
4. 尝试重启测试手机来清除系统缓存和内存（很重要）。

uni-app (uts 插件)

最近更新时间：2025-06-27 17:35:32

本文档将详细介绍如何结合 TUICallKit 组件和 uni-app 腾讯云推送服务（Push），以实现音视频通话的离线推送能力。

应用在后台或离线时通知 (海外)	应用在后台或离线时通知 (国内)	锁屏时通知 (国内)
		

步骤1：厂商配置

uni-app 厂商配置包含 Android 厂商配置 和 iOS 厂商配置， 详细参见 厂商配置 - uniapp 。

步骤2：下载 Push 插件并导入 HBuilderX

> **说明：**
> 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。

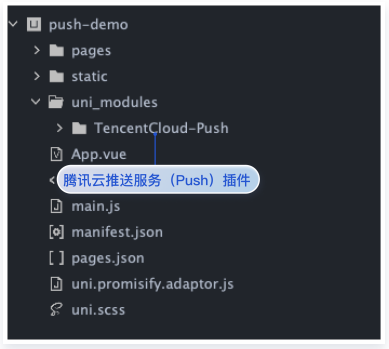
1. 打开 uni-app 腾讯云推送服务（Push），单击下载插件并导入 HBuilderX，将插件导入 HbuilderX 工程中。



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：



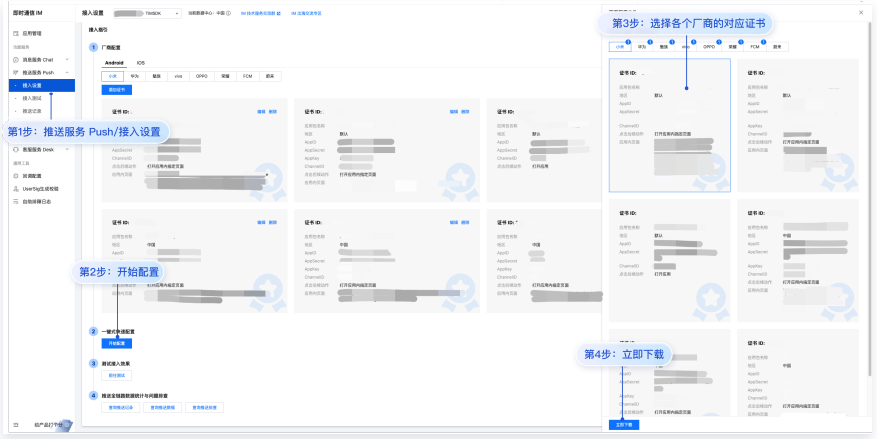
步骤3：离线推送配置

说明：

- 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 uni-app 腾讯云推送服务（Push）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` 。
- 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。
- 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

1. 新建 `nativeResources/android/assets` 目录。
2. 配置 `timpush-configs.json`（在 推送服务 Push > 接入设置 > 一键式快速配置 下载），到 `nativeResources/android/assets/` 目录下。如图所示：





3. 华为、荣耀、vivo、FCM。

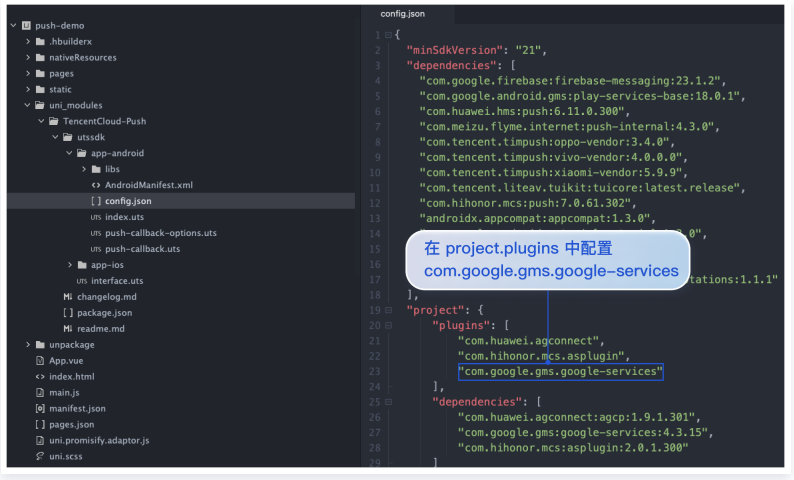
FCM

您需要支持 FCM 推送时，需要配置 google services。

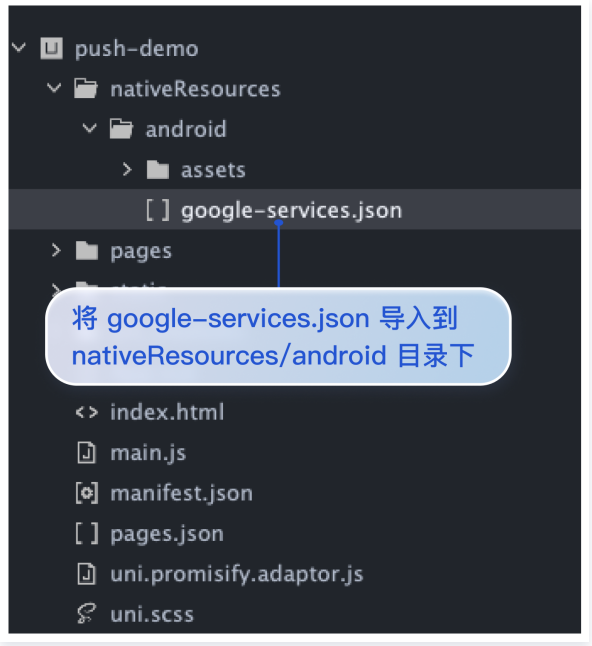
1. 配置 `com.google.gms.google-services` 到 `uni_modules/TencentCloud-Push/app-android/config.json` 的 `project.plugin` 中，如下所示：

```
"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"
    ]
```

```
}
```

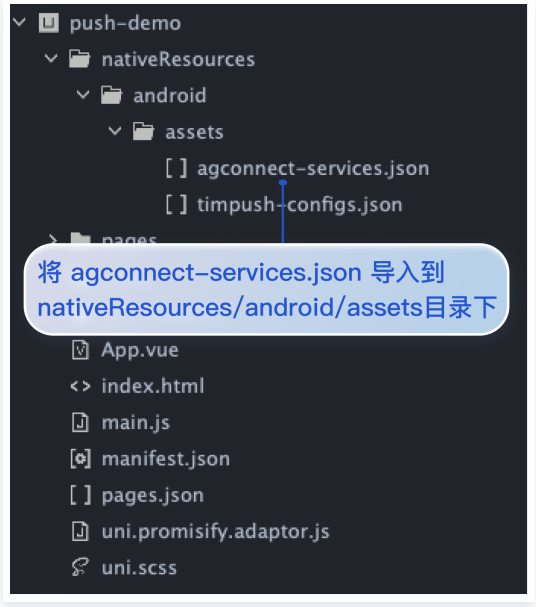


2. 配置 `google-services.json` 文件到 `nativeResources/android/` 目录下。如图所示：



华为

您需要支持华为推送时，需要配置 `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：

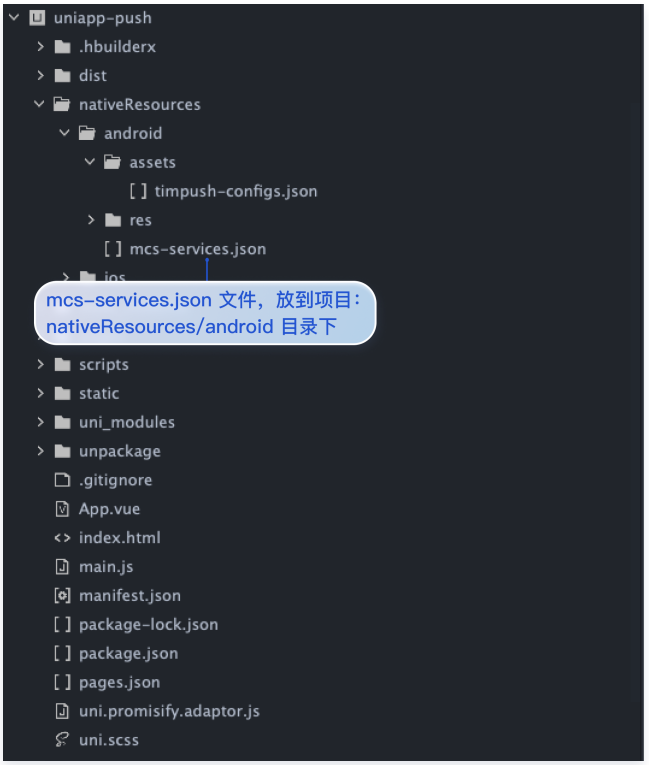


荣耀

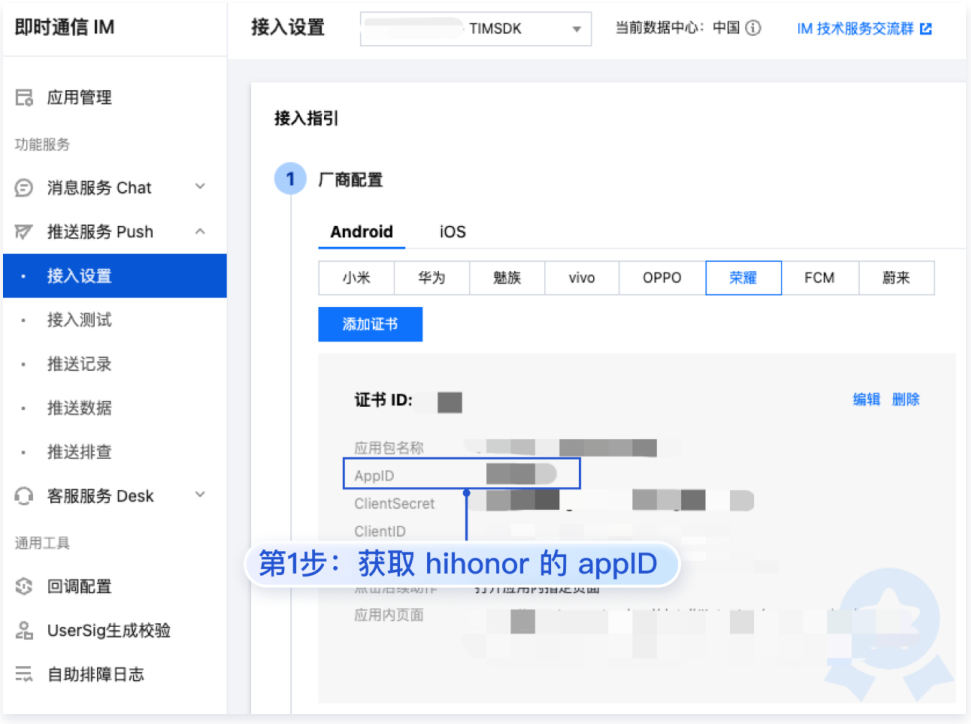
1. 编辑 `uni_modules/TencentCloud-Push/utssdk/app-android/config.json` 的 `dependencies`，添加 `"com.tencent.timpush:honor:8.3.6498"`。

```
{
    ...
    "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"`。




```
{
    "HONOR_APPID": ""
}
```

vivo

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`。




```
{
    "VIVO_APPKEY": "",
    "VIVO_APPID": "",
}
```

iOS

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 IM控制台 > 推送服务 Push > 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"
}
```



步骤4：引入并注册腾讯云推送服务（Push）

将 SDKAppID 和 appKey 替换为您在 IM 控制台 – 推送服务 Push – 接入设置页面 获取的应用的信息。如图所示：




```
// 集成 TencentCloud-Push
import * as Push from '@/uni_modules/TencentCloud-Push';
const SDKAppID = 0;   // 您的 SDKAppID
const appKey = '';    // 客户端密钥
const userID = 'xx'; // 您的登录 userID

Push.setRegistrationID(userID, (res) => {
    console.log('setRegistrationID OK', res);
});
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) => {
    // res 为推送扩展信息
    console.log('notification clicked', res);
});

// 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);
});
```

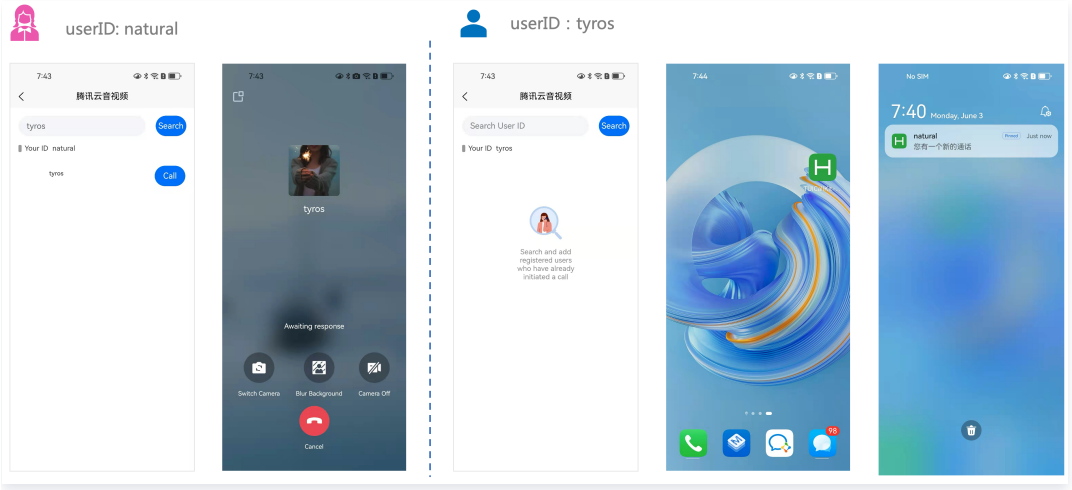
步骤5：使用云端证书，生成自定义基座

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。





步骤6：体验您的第一次推送



高级功能

获取离线推送扩展信息

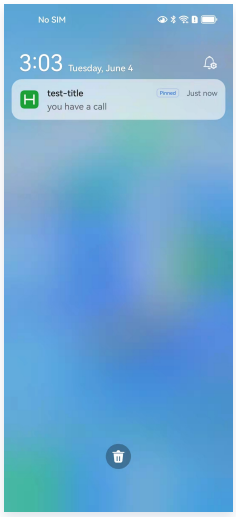
在 App.vue 内处理 `onShow` 回调，当设备收到离线推送，拉起 app 时，业务侧可通过 getNotificationExtInfo 接口获取推送扩展信息。

```
export default {
    onLaunch: function() {
    },
    onShow: function() {
```

```
        console.log('App Show')
        Push.getNotificationExtInfo((extInfo) => {
            console.log('getNotificationExtInfo ok', extInfo);
        })
    },
    onHide: function() {
        console.log('App Hide')
    }
}
```

自定义推送内容

在调用 calls 发起通话时，可通过 offlinePushInfo 字段来自定义对端收到推送时的内容。




```
/ https://cloud.tencent.com/document/product/647/78763#calls
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();
const callParams = {
  userIDList: ['user1'],
  callMediaType: 1,
  callParams: {
    offlinePushInfo: {
      title: "test-title", // 修改 title
      description: "you have a call", // 修改 discription
    }
  },
};

callEngine.calls(callParams);
```

自定义推送铃声

说明：

- 接收端需集成 uni-app 腾讯云推送服务（Push）。
- 发送端 @tencentcloud/chat ⩾ 3.3.2 。
- 支持华为、小米、OPPO、FCM 和 APNS，更详细配置参考 Push-自定义铃音 。

接收端

Android

定制的铃音资源文件，添加到项目 `nativeResources/android/res/raw` 目录下，如图所示：



设置私信通道和自定义铃音（ OPPO、FCM 需要设置）

> **说明：**
> 1. uni-app 腾讯云推送服务（Push）≥ 0.5.0。
> 2. OPPO、FCM 使用自定义铃音需要先设置私信通道和自定义铃音。

- 完成上述步骤后，您可以调用 push.createNotificationChannel 设置 OPPO 、FCM 私信通道和自定义铃音。
- 在 `App.vue` 中设置 OPPO、FCM 私信通道。如下所示：

```
import * as Push from '@/uni_modules/TencentCloud-Push';
Push.createNotificationChannel({
    channelID: '', // 自定义 channel 的 ID。OPPO 为控制台配置的 channelID
    channelName: '自定义channel',// 自定义channel 名称
    channelDesc: '这只是描述', // 自定义 channel 描述
    channelSound: 'private_ring'  // 自定义铃音的名称且不需要后缀名
}, () => {
    console.log('Push | createNotificationChannel ok');
})
```

iOS

> **说明：**
> - iOS 自定义铃音，uni-app 必须为正式包。
> - iOS 自定义铃音长度不能超过 30s。

定制的铃音资源文件，添加到项目 `nativeResources/ios/Resources` 目录下，如图所示：



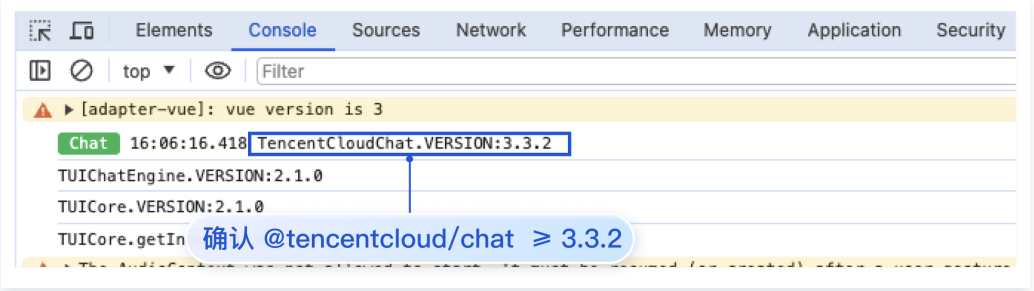
发送端

1. 升级 `@tencentcloud/chat` 到最新版本。

```
npm install @tencentcloud/chat@latest
```

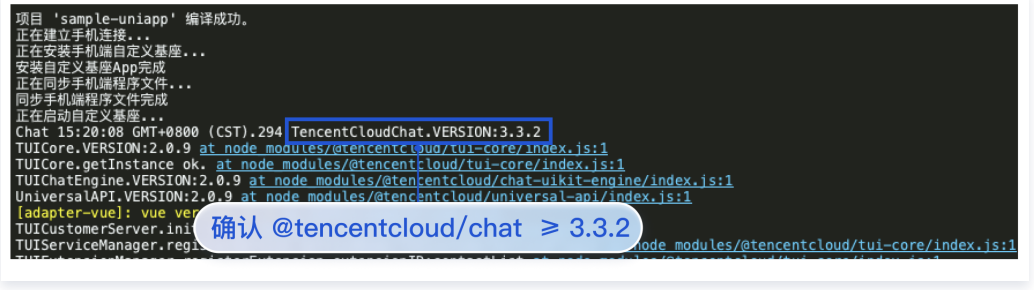
Web

在浏览器控制台查看 TencentCloudChat.VERSION 版本号，来确认 **@tencentcloud/chat ⩾ 3.3.2** 如图所示：



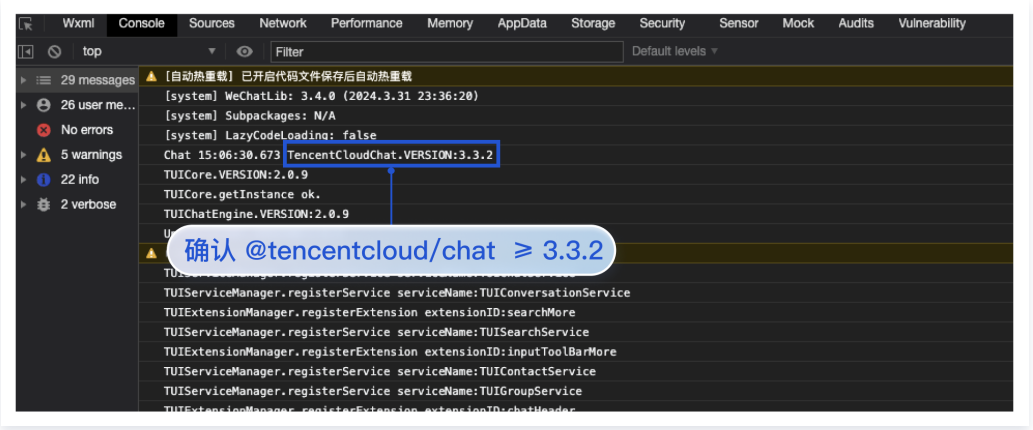
uniapp

在 HBuilderX 日志中查看 TencentCloudChat.VERSION 版本号，来确认 **@tencentcloud/chat ⩾ 3.3.2** 如图所示：



小程序

在小程序开发者工具控制台查看 TencentCloudChat.VERSION 版本号，来确认 @tencentcloud/chat ⩾ 3.3.2 如图所示：



2. 发送消息，设置 offlinePushInfo 自定义铃音的相关参数。

> **说明：**
> - 小米手机在 Android 8.0 及以上版本必须设置 androidInfo.XiaoMiChannelID，请您参见：小米自定义铃声 。
> - 谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须设置 androidInfo.FCMChannelID。
> - androidSound 为铃声文件名，不需要后缀名。
> - iOSSound 为铃声文件名，需要后缀名。

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();

const callParams = {
  userIDList: ['user1'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
  callParams: {
    offlinePushInfo: {
      title: "test-title",
      description: "you have a call",
      androidSound: 'private_ring', // androidSound 为铃声文件名，不需要后缀名
      iOSSound: 'private_ring.mp3', // iOSSound为铃声文件名，需要后缀名
    }
  },
};

callEngine.calls(callParams);
```

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

常见问题

设备通知栏如何设置?

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。
- “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。

厂商推送限制如何设置?

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

将推送消息分为服务与通讯类和资讯营销类，推送效果和策略不同。另外，消息分类还和自分类权益有关：

- 无自分类权益，推送消息厂商还会进行二次智能分类 。
- 有申请自分类权益，消息分类会按照自定义的分类进行推送。 具体请参见 厂商描述 。

vivo

将推送消息分为系统消息类和运营消息类，推送效果和策略不同。系统消息类型还会进行厂商的智能分类二次修正，若智能分类识别出不是系统消息，会自动修正为运营消息，如果误判可邮件申请反馈。另外，消息推送也受日推总数量限制，日推送量由应用在厂商订阅数统计决定。 具体请参见 厂商描述1 或 厂商描述2 。

OPPO

将推送消息分为私信消息类和公信消息类，推送效果和策略不同。其中私信消息是针对用户有一定关注度，且希望能及时接收的信息，私信通道权益需要邮件申请。公信通道推送数量有限制。 具体请参见 厂商描述1 或 厂商描述2 。

小米

将推送消息分为重要消息类和普通消息类，推送效果和策略不同。其中重要消息类型仅允许即时通讯消息、个人关注动态提醒、个人事项提醒、个人订单状态变化、个人财务提醒、个人状态变化、个人资源变化、个人设备提醒这8类消息推送，可以在厂商控制台申请开通。普通消息类型推送数量有限制。

具体请参见 厂商描述1 或 厂商描述2 。

魅族

推送消息数量有限制。 具体请参见 厂商描述 。

FCM

推送上行消息频率有限制。 具体请参见 厂商描述 。

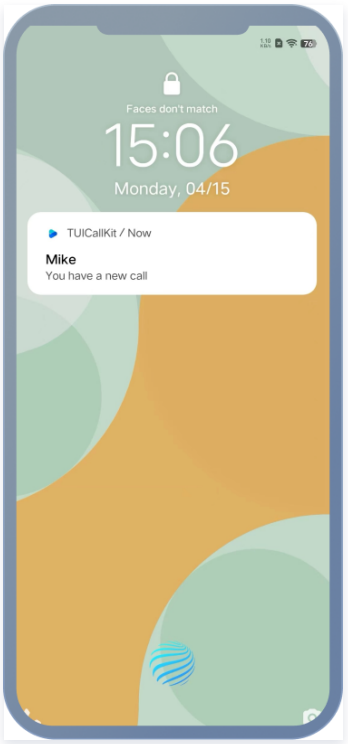
技术咨询

点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

uni-app (客户端)

最近更新时间：2025-02-26 14:16:12

本文档将详细介绍如何结合 TUICallKit 组件和 uni-app 腾讯云推送服务（Push），以实现音视频通话的离线推送能力。

应用在后台或离线时通知 (海外)	应用在后台或离线时通知 (国内)	锁屏时通知 (国内)
		

步骤一：厂商配置

uniapp 厂商配置包含 Android 厂商配置 和 iOS 厂商配置， 详细参见 厂商配置 - uniapp 。

步骤二：下载 Push 插件并导入 Hbuilderx

> **说明：**
> 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。

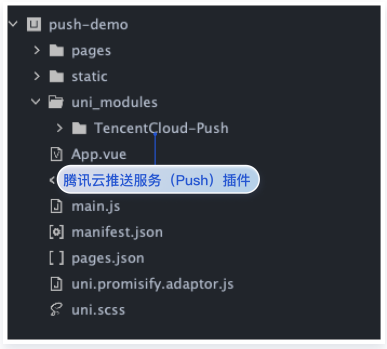
1. 打开 uni-app 腾讯云推送服务（Push），单击下载插件并导入 HbuilderX，将插件导入 HbuilderX 工程中。



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：



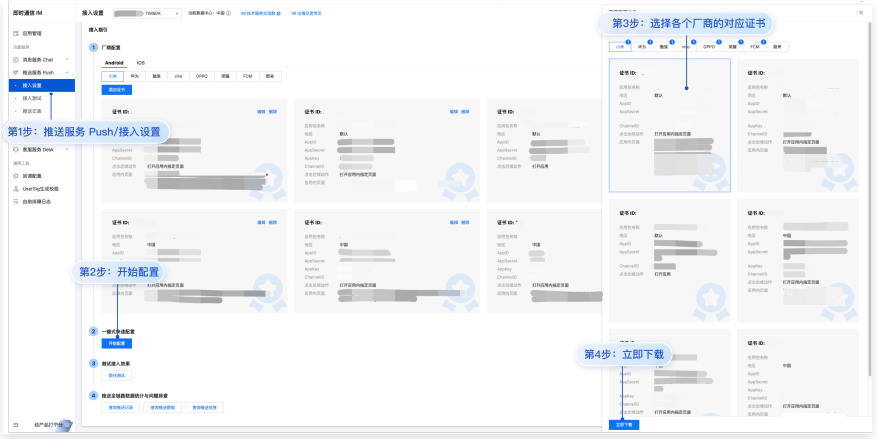
步骤三：离线推送配置

说明：

- 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 uni-app 腾讯云推送服务（Push）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` 。
- 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。
- 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

1. 新建 `nativeResources/android/assets` 目录。
2. 配置 `timpush-configs.json`（在 推送服务 Push > 接入设置 > 一键式快速配置 下载），到 `nativeResources/android/assets/` 目录下。如图所示：





3. 华为、荣耀、vivo、FCM。

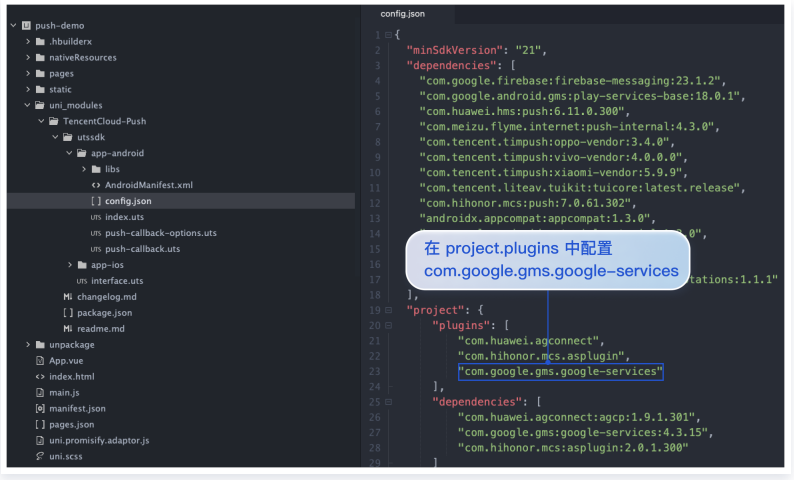
FCM

您需要支持 FCM 推送时，需要配置 google services。

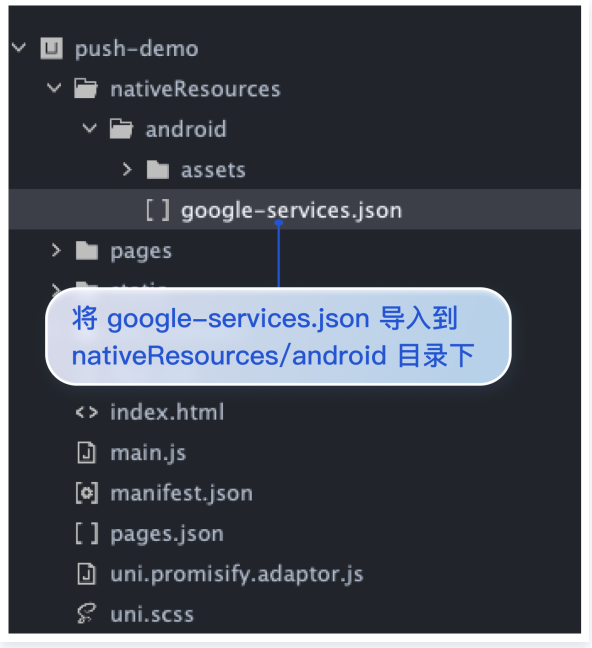
1. 配置 `com.google.gms.google-services` 到 `uni_modules/TencentCloud-Push/app-android/config.json` 的 `project.plugin` 中，如下所示：

```
"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"
    ]
```

```
}
```

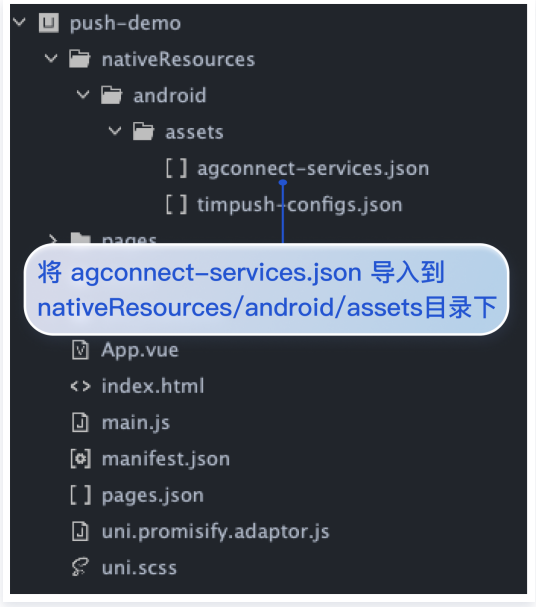


2. 配置 `google-services.json` 文件到 `nativeResources/android/` 目录下。如图所示：



华为

您需要支持华为推送时，需要配置 `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：

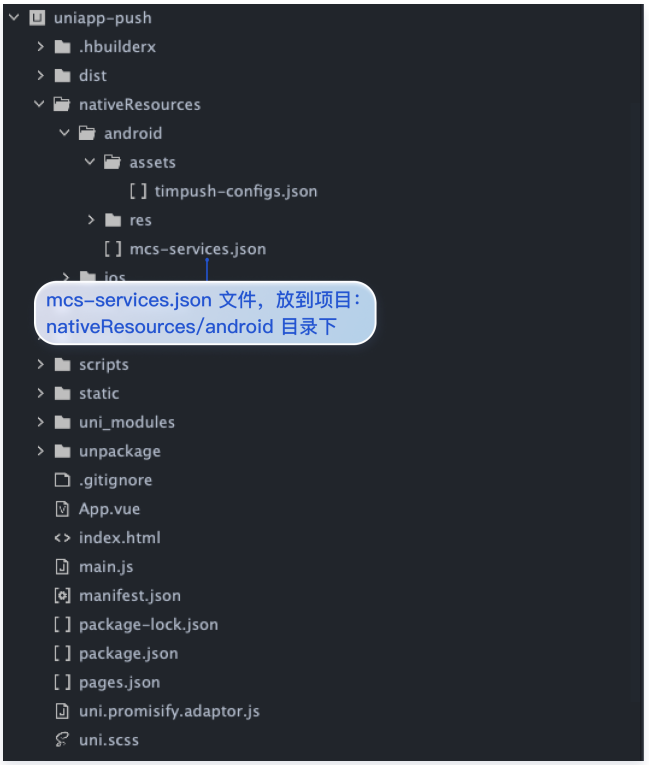


荣耀

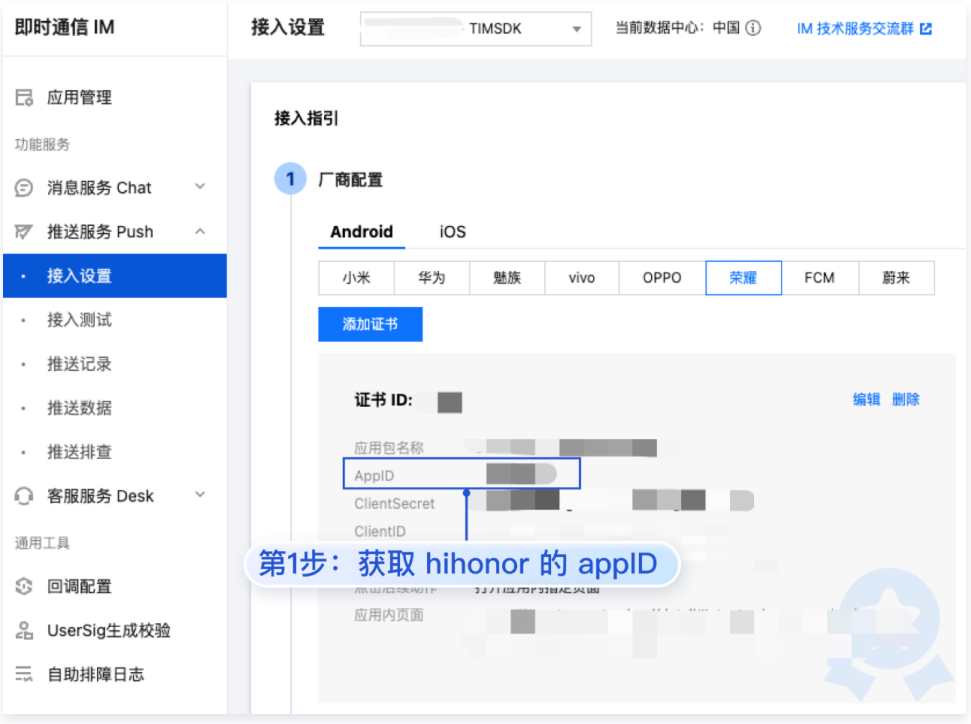
1. 编辑 `uni_modules/TencentCloud-Push/utssdk/app-android/config.json` 的 `dependencies`，添加 `"com.tencent.timpush:honor:8.3.6498"`。

```
{
    ...
    "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"`。




```
{
    "HONOR_APPID": ""
}
```

vivo

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`。




```
{
    "VIVO_APPKEY": "",
    "VIVO_APPID": "",
}
```

iOS

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 IM控制台 > 推送服务 Push > 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"
}
```



步骤四：引入并注册腾讯云推送服务（Push）

将 SDKAppID 和 appKey 替换为您在 IM 控制台 – 推送服务 Push – 接入设置页面 获取的应用的信息。如图所示：




```
// 集成 TencentCloud-Push
import * as Push from '@/uni_modules/TencentCloud-Push';
const SDKAppID = 0;   // 您的 SDKAppID
const appKey = '';    // 客户端密钥
const userID = 'xx'; // 您的登录 userID

Push.setRegistrationID(userID, (res) => {
    console.log('setRegistrationID OK', res);
});
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) => {
    // res 为推送扩展信息
    console.log('notification clicked', res);
});

// 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);
});
```

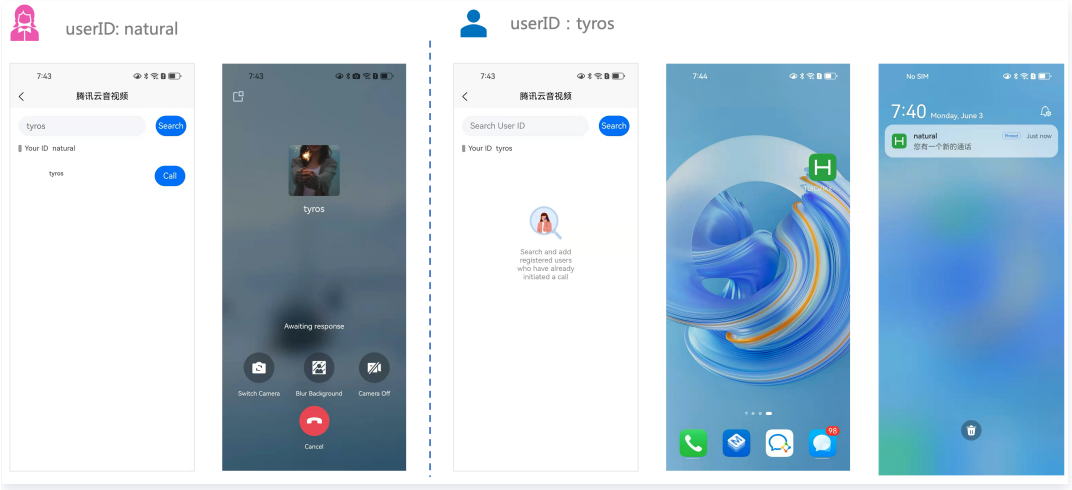
步骤五：使用云端证书，生成自定义基座

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。





步骤六：体验您的第一次推送



高级功能

获取离线推送扩展信息

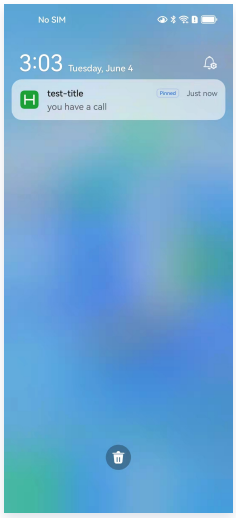
在 App.vue 内处理 `onShow` 回调，当设备收到离线推送，拉起 app 时，业务侧可通过 getNotificationExtInfo 接口获取推送扩展信息。

```
export default {
    onLaunch: function() {
    },
    onShow: function() {
```

```
        console.log('App Show')
        Push.getNotificationExtInfo((extInfo) => {
            console.log('getNotificationExtInfo ok', extInfo);
        })
    },
    onHide: function() {
        console.log('App Hide')
    }
}
```

自定义推送内容

在调用 calls 发起通话时，可通过 offlinePushInfo 字段来自定义对端收到推送时的内容。



CallKit 单独集成场景

uni-app(客户端)

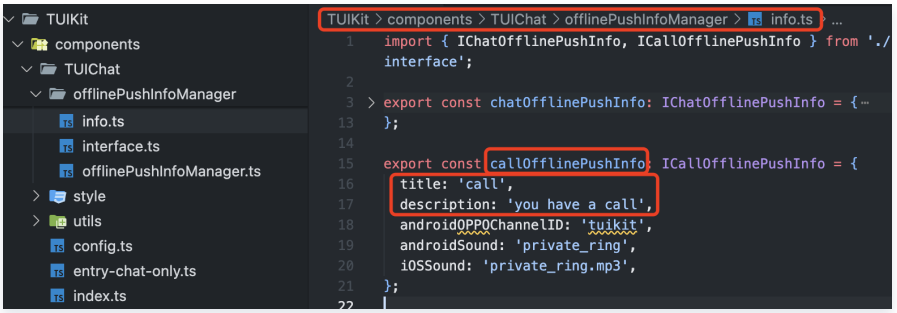
```
// https://cloud.tencent.com/document/product/647/78763#calls
const callParams = {
  userIDList: ['user1'],
  callMediaType: 1,
  callParams: {
    offlinePushInfo: {
      title: "test-title", // 修改 title
      description: "you have a call", // 修改 discription
    }
  },
};

uni.$TUICallKit.calls(callParams, res => {} );
```

Web/小程序

```
// https://cloud.tencent.com/document/product/647/81015#calls
await TUICallKitServer.calls({
  userIDList: ['mike'],
  type: TUICallType.VIDEO_CALL,
  offlinePushInfo: {
    title: "test-title", // 修改 title
    description: "you have a call", // 修改 discription
  }
});
```

Chat + Call 融合场景

修改 `callOfflinePushInfo` 对象中的参数。



自定义推送铃声

说明：

- 接收端需集成 uni-app 腾讯云推送服务（Push）。
- 发送端 @tencentcloud/chat ⩾ 3.3.2 。
- 支持华为、小米、OPPO、FCM 和 APNS，更详细配置参考 Push-自定义铃音 。

接收端

Android

定制的铃音资源文件，添加到项目 `nativeResources/android/res/raw` 目录下，如图所示：



设置私信通道和自定义铃音（ OPPO、FCM 需要设置）

说明：

1. uni-app 腾讯云推送服务（Push）⩾ 0.5.0。
2. OPPO、FCM 使用自定义铃音需要先设置私信通道和自定义铃音。

- 完上述步骤后，您可以调用 push.createNotificationChannel 设置 OPPO 、FCM 私信通道和自定义铃音。
- 在 `App.vue` 中设置 OPPO、FCM 私信通道。如图所示：

```
import * as Push from '@/uni_modules/TencentCloud-Push';
Push.createNotificationChannel({
```

```
    channelID: '', // 自定义 channel 的 ID。OPPO 为控制台配置的 channelID
    channelName: '自定义channel',// 自定义channel 名称
    channelDesc: '这只是描述', // 自定义 channel 描述
    channelSound: 'private_ring'  // 自定义铃音的名称且不需要后缀名
}, () => {
    console.log('Push | createNotificationChannel ok');
})
```

iOS

说明：

- iOS 自定义铃音，uni-app 必须为正式包。
- iOS 自定义铃音长度不能超过 30s。

定制的铃音资源文件，添加到项目 `nativeResources/ios/Resources` 目录下，如图所示：



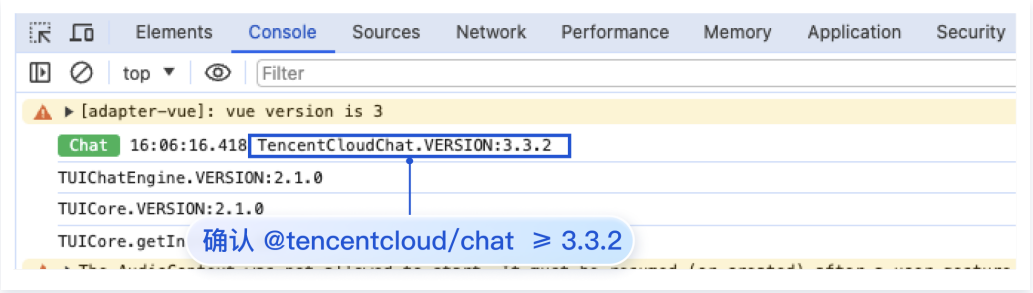
发送端

1. 升级 `@tencentcloud/chat` 到最新版本。

```
npm install @tencentcloud/chat@latest
```

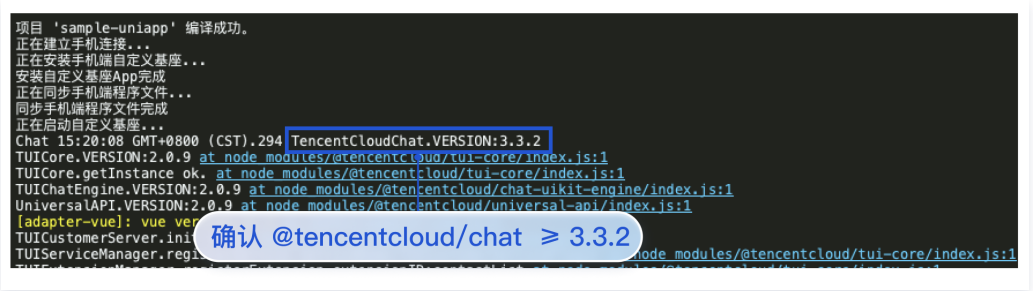
web

在浏览器控制台查看 TencentCloudChat.VERSION 版本号，来确认 **@tencentcloud/chat ⩾ 3.3.2** 如图所示：



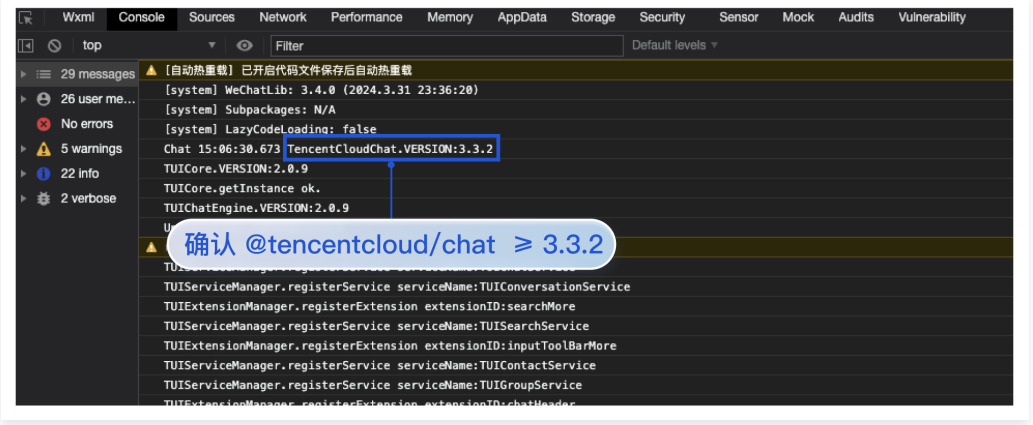
uniapp

在 HBuilder 日志中查看 TencentCloudChat.VERSION 版本号，来确认 **@tencentcloud/chat ≥ 3.3.2** 如图所示：



小程序

在小程序开发者工具控制台查看 TencentCloudChat.VERSION 版本号，来确认 **@tencentcloud/chat ≥ 3.3.2** 如图所示：



2. 发送消息，设置 offlinePushInfo 自定义铃音的相关参数。

> **说明：**
> - 小米手机在 Android 8.0 及以上版本必须设置 androidInfo.XiaoMiChannelID，请您参见：小米自定义铃声 。
> - 谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须设置 androidInfo.FCMChannelID。

TUICallKit 单独集成场景

> **说明：**
> - androidSound 为铃声文件名，不需要后缀名。

- iOSSound 为铃声文件名，需要后缀名。

uni-app(客户端)

```
const callParams = {
  userIDList: ['user1'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
  callParams: {
    offlinePushInfo: {
      title: "test-title",
      description: "you have a call",
      androidSound: 'private_ring', // androidSound 为铃声文件名，不需要后缀名
      iOSSound: 'private_ring.mp3', // iOSSound为铃声文件名，需要后缀名
    }
  },
};

uni.$TUICallKit.calls(callParams, res => {} );
```

Web/小程序

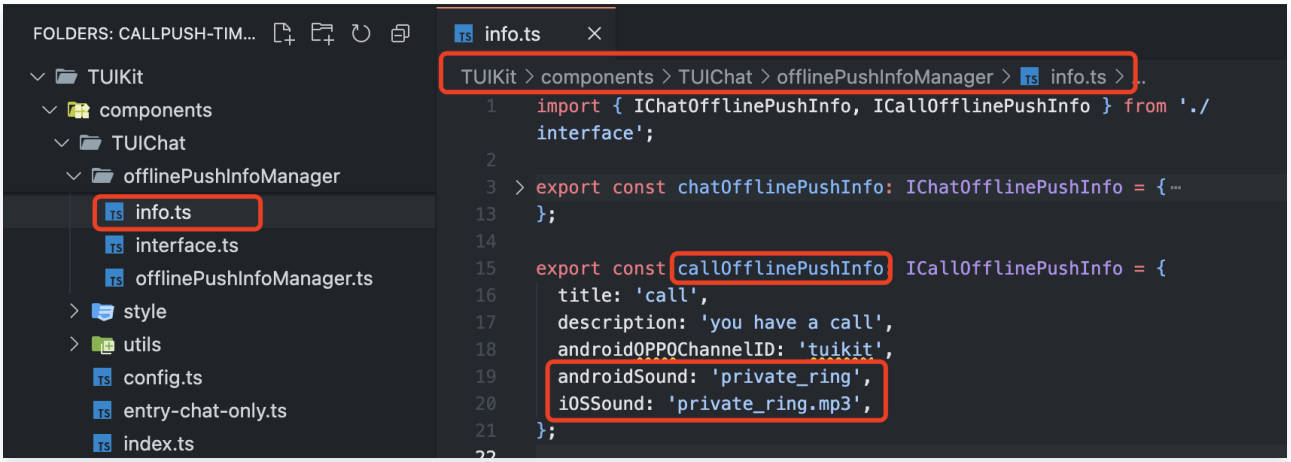
```
// https://cloud.tencent.com/document/product/647/81015#calls
await TUICallKitServer.calls({
  userIDList: ['mike'],
  type: TUICallType.VIDEO_CALL,
  offlinePushInfo: {
    title: 'xxx',
    androidSound: 'private_ring', // androidSound 为铃声文件名，不需要后缀名
    iOSSound: 'private_ring.mp3', // iOSSound为铃声文件名，需要后缀名
  }
});
```

TUIKit + TUICallKit 融合场景

说明：

- androidSound 为铃声文件名，不需要后缀名。
- iOSSound 为铃声文件名，需要后缀名。

修改 `callOfflinePushInfo` 对象中的参数。



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

常见问题

设备通知栏如何设置?

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。
- “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。

厂商推送限制如何设置?

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

将推送消息分为服务与通讯类和资讯营销类，推送效果和策略不同。另外，消息分类还和自分类权益有关：

- 无自分类权益，推送消息厂商还会进行二次智能分类 。
- 有申请自分类权益，消息分类会按照自定义的分类进行推送。 具体请参见 厂商描述 。

vivo

将推送消息分为系统消息类和运营消息类，推送效果和策略不同。系统消息类型还会进行厂商的智能分类二次修正，若智能分类识别出不是系统消息，会自动修正为运营消息，如果误判可邮件申请反馈。另外，消息推送也受日推总数量限制，日推送量由应用在厂商订阅数统计决定。 具体请参见 厂商描述1 或 厂商描述2 。

OPPO

将推送消息分为私信消息类和公信消息类，推送效果和策略不同。其中私信消息是针对用户有一定关注度，且希望能及时接收的信息，私信通道权益需要邮件申请。公信通道推送数量有限制。 具体请参见 厂商描述1 或 厂商描述2 。

小米

将推送消息分为重要消息类和普通消息类，推送效果和策略不同。其中重要消息类型仅允许即时通讯消息、个人关注动态提醒、个人事项提醒、个人订单状态变化、个人财务提醒、个人状态变化、个人资源变化、个人设备提醒这8类消息推送，可以在厂商控制台申请开通。普通消息类型推送数量有限制。具体请参见 厂商描述1 或 厂商描述2 。

魅族

推送消息数量有限制。 具体请参见 厂商描述 。

FCM

推送上行消息频率有限制。 具体请参见厂商描述 。

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。

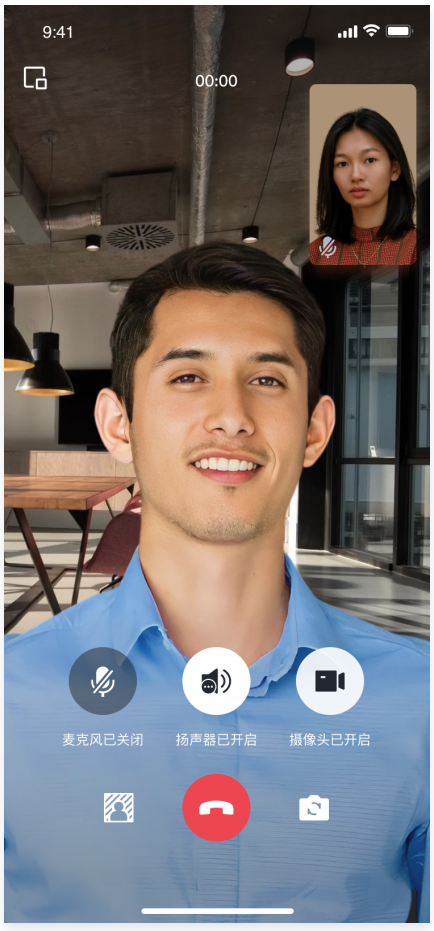
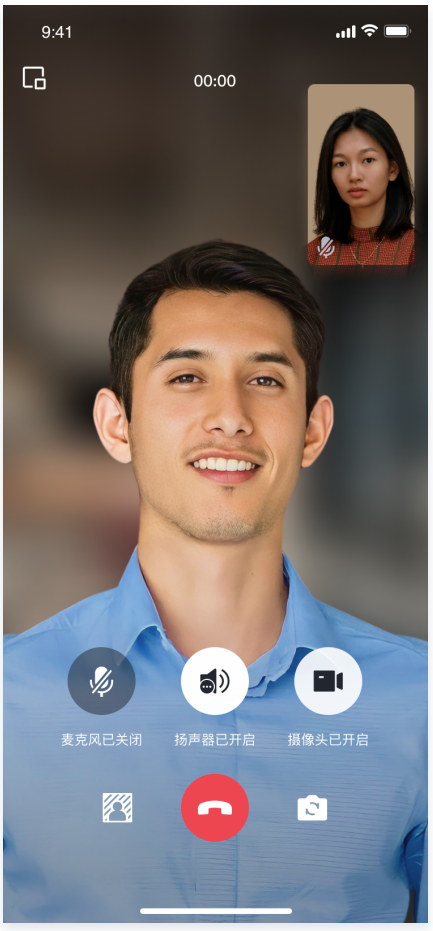
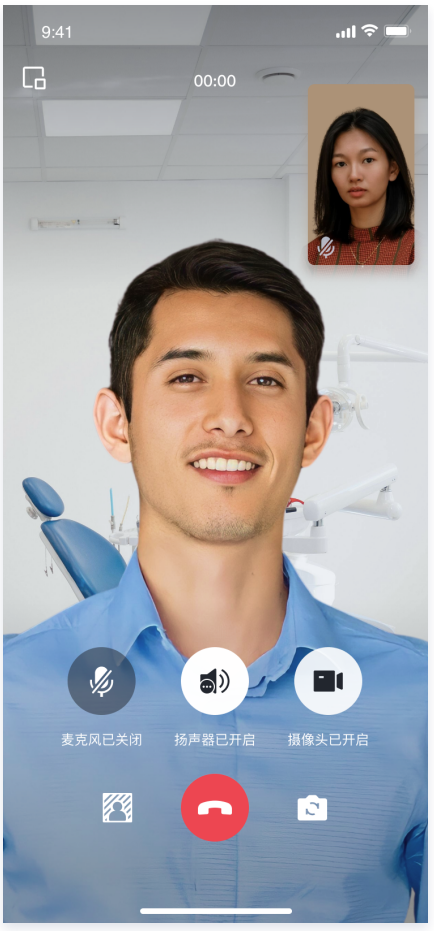
虚拟背景（TUICallKit）
iOS

最近更新时间：2024-05-13 10:03:41

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

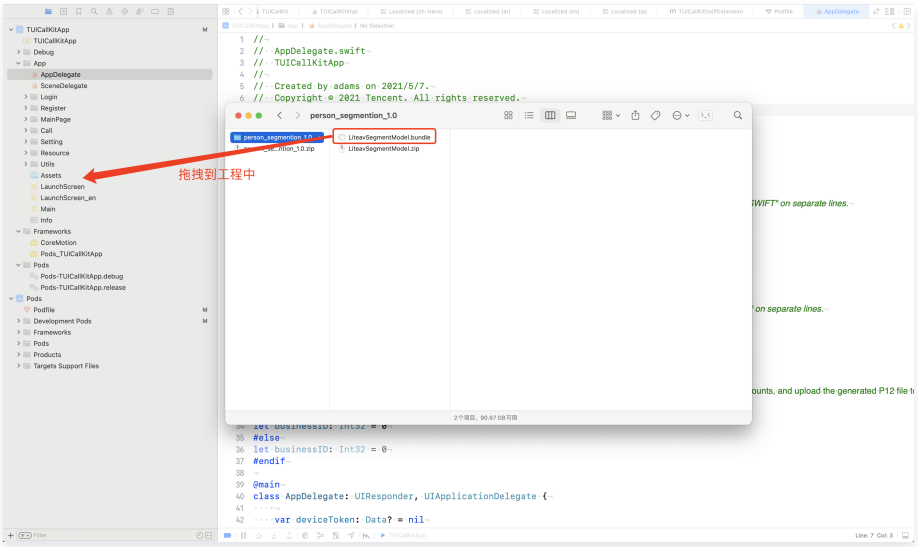
集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

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



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

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

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

开启模糊背景

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

```
import TUICallKit

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

设置图片背景（可选）

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

```
import TUICallEngine

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

常见问题

1. 开启模糊背景无反应或有延迟

- 确保已购买音视频通话 **群组通话版（140万分钟）** 套餐，详见 开通服务 。
- 确保已下载模型文件至本地。
  如果没有添加模型文件到本地路径中，在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。
- 检查模型文件与 SDK 是否匹配。

2. 模型文件与SDK的匹配

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

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

版权所有：腾讯云计算（北京）有限责任公司

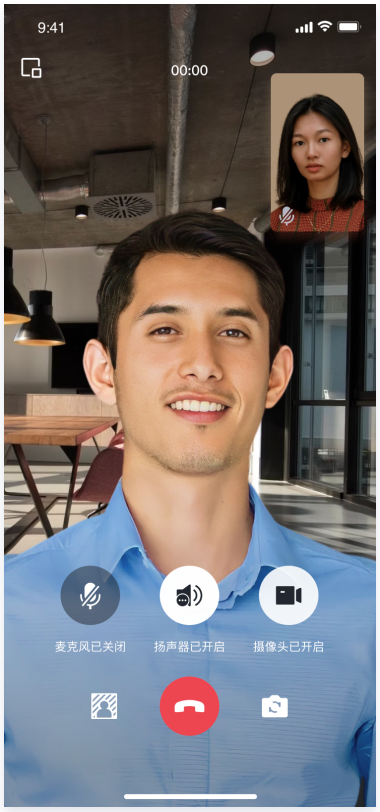
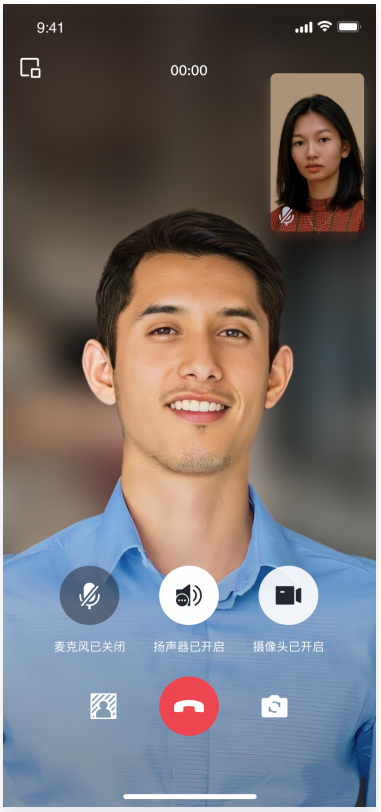
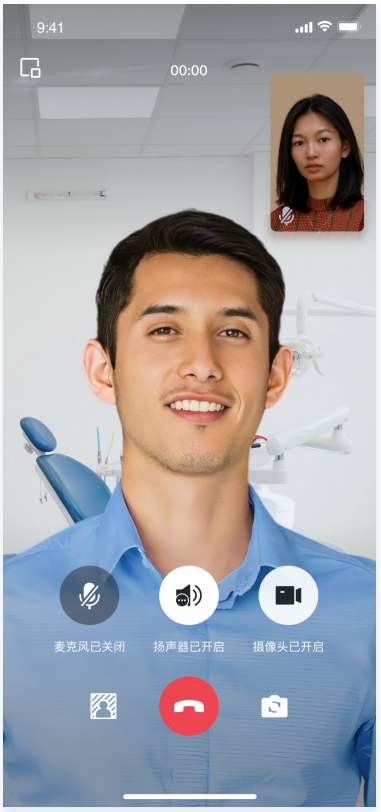
Android

最近更新时间：2024-05-13 11:25:01

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

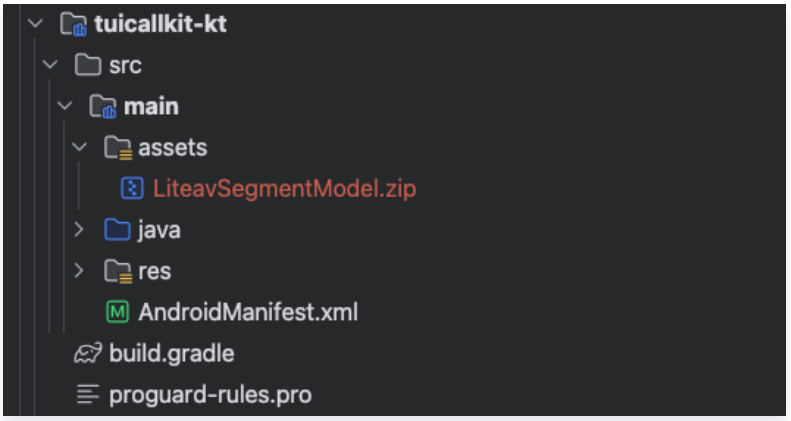
集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

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



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

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

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

开启模糊背景

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

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

设置图片背景（可选）

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

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

常见问题

1. 开启模糊背景无反应或有延迟

- 确保已购买音视频通话 **群组通话版（140万分钟）** 套餐，详见 开通服务。
- 确保已下载模型文件至本地。
  如果没有添加模型文件到本地路径中，在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。
- 检查模型文件与 SDK 是否匹配。

2. 模型文件与 SDK 的匹配

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

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

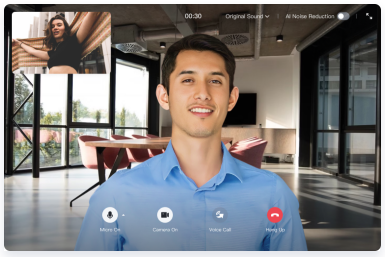
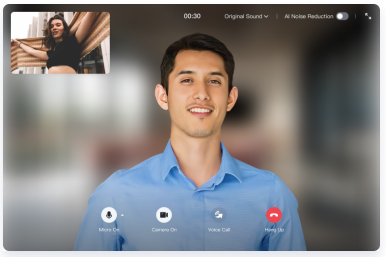
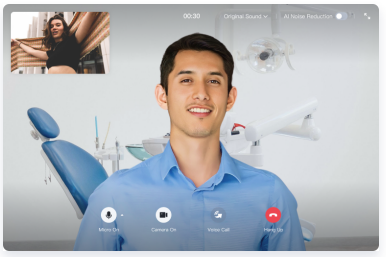
Web

最近更新时间：2024-05-30 17:08:01

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

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

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

开启模糊背景

> **注意：**
> - H5 暂不支持，仅 PC 支持。
> - TUICallKit v3.2.4+ 支持。

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

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

设置图片背景（可选）

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

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

常见问题

开启模糊背景无反应或有延迟？

- 确保已购买音视频通话 **群组通话版（140万分钟）** 套餐，详见 开通服务 。
- 网络较差时，虚拟背景模型文件可能未下载完，从而导致开启虚拟背景失败。

关闭摄像头，是否能开启虚拟背景？

不能。

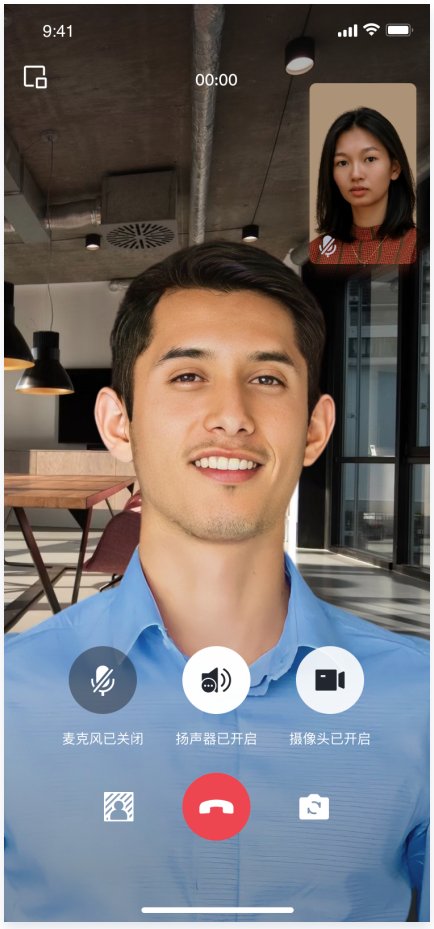
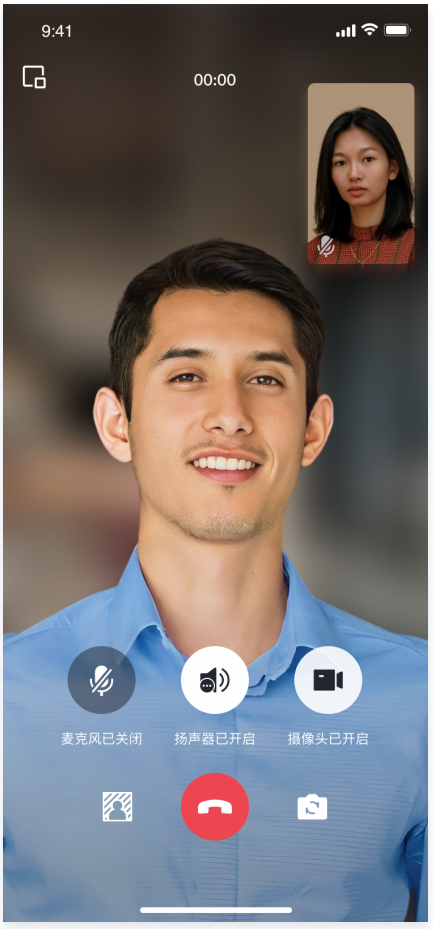
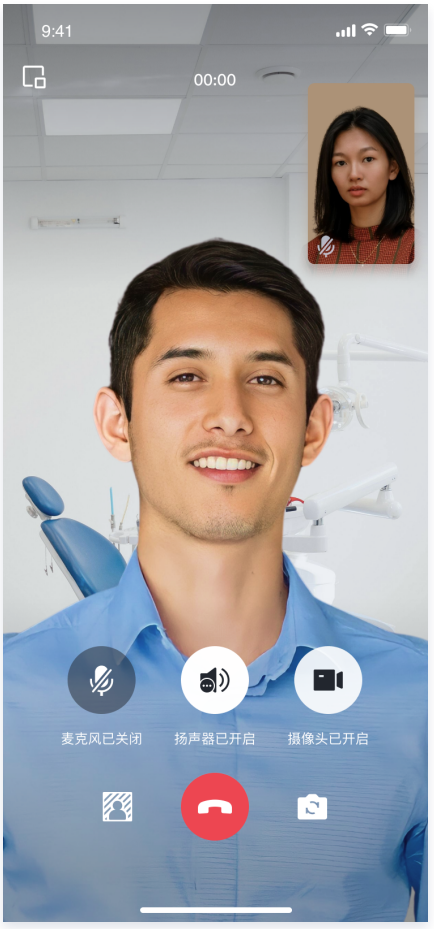
uniapp（客户端）

最近更新时间：2024-04-28 14:49:01

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

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

- 在使用腾讯云提供的虚拟背景功能前，您需要前往控制台，为应用开通音视频服务，购买**群组通话版（140万分钟）**套餐。具体步骤请参见 开通服务。
- HbuilderX 版本要求：HbuilderX 版本 ≥ 3.94。

开启模糊背景

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

> **说明：**
> 首次使用该功能，可能会存在会延迟或无反应，这是因为在开启模糊背景功能的时候，SDK 内部才会去下载模型文件，正常网络情况下，下载耗时1~3s，网络越差耗时越久。

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

```
TUICallKit.enableVirtualBackground(enable);
```

版权所有：腾讯云计算（北京）有限责任公司

uniapp（小程序）

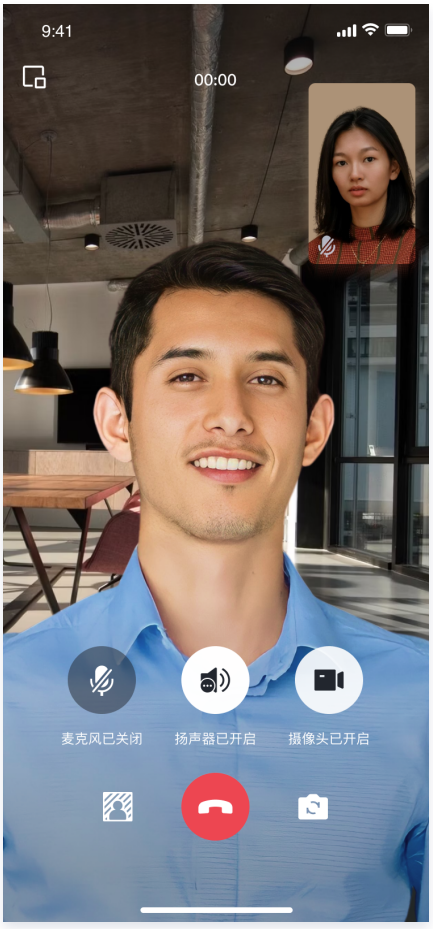
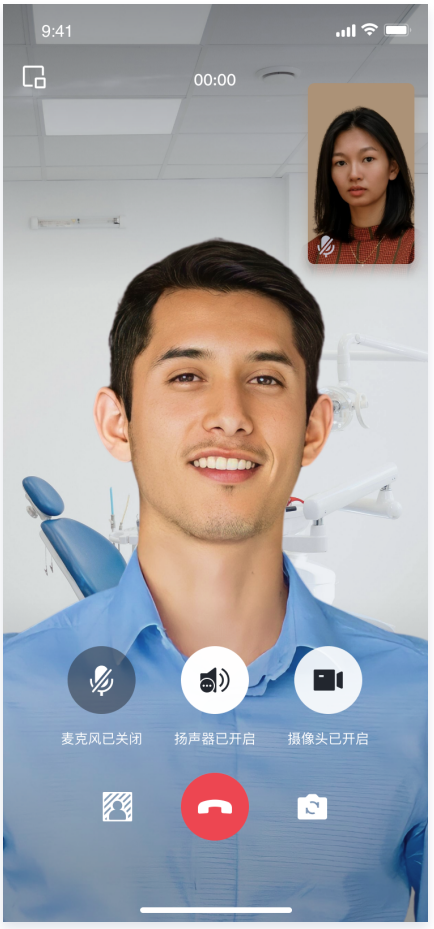
最近更新时间：2024-09-20 21:28:31

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

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

> **注意：**
> 小程序暂时**不支持**模糊背景效果。

原始摄像头	图片背景效果
	

准备条件

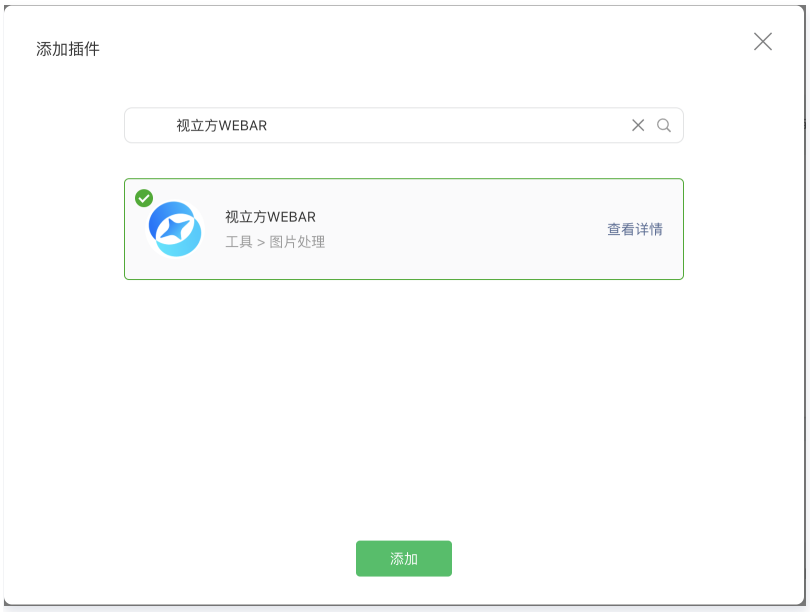
TUICallKit 虚拟背景能力基于 腾讯特效 SDK，请首先申请试用或购买视立方 WEBAR License，详见 腾讯云视立方控制台。

开始使用

> **警告：**
> 在集成虚拟背景能力之前，请阅读 TRTC TUICallKit 快速接入(uniapp-小程序)，推荐使用**分包方式**集成 TUICallKit，并完成调试预览相关操作。

步骤一：添加插件

首先要在小程序管理后台的**设置 > 第三方设置 > 插件管理**中**添加插件**。开发者可登录小程序管理后台，搜索 视立方WEBAR 申请添加插件。



步骤二：引用插件

使用组件前需在 uniapp 工程的 `manifest.json` 中声明要使用的插件：

```
{
  "mp-weixin": {
    "plugins": {
      "webarPlugin": {
        "version": "latest",
        "provider": "wx04445cff065100ed"
      }
    }
  }
}
```

步骤三：下载虚拟背景相关依赖

```
npm install tencentcloud-webar-wx crypto-js
```

步骤四：引入美颜特效 SDK

1. 在 TUICallKit/debug 目录下新增 webar-auth.js，实例代码如下：

```
import crypto from "crypto-js";
/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APP_ID = "";
/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/web)，创建项目即可
获得 LicenseKey
```

```
 */
const LICENSE_KEY = "";
/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提供，前端调用拉取
签名，参考
 * [签名方法]
(https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8D.E6.96.B9.E6.B3.95)
 */
const TOKEN = "";
const sha256 = (str) => {
  return crypto.SHA256(str).toString();
};

// 此方法仅测试时使用，为防止 token 泄露，发布时请使用服务端加密，详见[腾讯云-腾讯特效 SDK 官网文档]
(https://cloud.tencent.com/document/product/616/71364)
function authFunc() {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(
    timestamp + TOKEN + APP_ID + timestamp
  ).toUpperCase();
  return { signature, timestamp };
}
export { LICENSE_KEY, APP_ID, authFunc };
```

2. 修改 TUICallKit/src/Components/components/common/Pusher/weChatPusher/weChatPusher.vue 文件

> **说明：**
> imagePath 表示开启虚拟背景显示的图片背景，可通过修改图片路径来修改显示的背景图片，目前仅支持网络图片(**不支持本地图片**)

```
<template>
<div :class="localClass">
  <div class="stream-box">
    <WebarPusher
      class="stream"
      :custom-effect="true"
      :licenseKey="licenseKey"
      :appId="appid"
      :authFunc="authFunc"
      :background="setBlurBackground ? imagePath : ''"
      :url="pusher.url"
      :mode="pusher.mode"
      :autopush="true"
      :enable-camera="pusher.enableCamera"
      :enable-mic="true"
      :muted="!pusher.enableMic"
      :enable-agc="true"
      :enable-ans="true"
      :enable-ear-monitor="pusher.enableEarMonitor"
      auto-focus="false"
      :zoom="pusher.enableZoom"
      :min-bitrate="pusher.minBitrate"
      :max-bitrate="pusher.maxBitrate"
      :video-width="pusher.videoWidth"
      :video-height="pusher.videoHeight"
      :beauty="pusher.beautyLevel"
      :whiteness="pusher.whitenessLevel"
```

```
        :orientation="pusher.videoOrientation"
        :aspect="pusher.videoAspect"
        :device-position="pusher.frontCamera"
        :remote-mirror="pusher.enableRemoteMirror"
        :local-mirror="pusher.localMirror"
        :background-mute="pusher.enableBackgroundMute"
        :audio-quality="pusher.audioQuality"
        :audio-volume-type="pusher.audioVolumeType"
        :audio-reverb-type="pusher.audioReverbType"
        :waiting-img="pusher.waitingImage"
        :beauty-style="pusher.beautyStyle"
        :filter="pusher.filter"
        @statechange="pusherStateChangeHandler"
        @netstatus="pusherNetStatus"
        @error="pusherErrorHandler"
        @audiovolumenotify="pusherAudioVolumeNotify"
      ></WebarPusher>
    </div>
  </div>
</template>

<script lang="ts">
export default {
  options: {
    virtualHost: true,
  },
};
</script>

<script lang="ts" setup>
import {
  TUICallKitServer,
  StoreName,
  NAME,
  TUIStore,
} from "../../../../../TUICallService/index";
import { onMounted, ref, onUnmounted } from "../../../../../adapter-vue";
import {
  LICENSE_KEY,
  APP_ID,
  authFunc as auth,
} from "../../../../../../debug/webar-auth";
import WebarPusher from "tencentcloud-webar-wx/WebArPusher/WebArPusher.vue";
const pusher = ref(TUIStore.getData(StoreName.CALL, NAME.PUSHER));
const callStatus = ref(TUIStore.getData(StoreName.CALL, NAME.CALL_STATUS));
const licenseKey = ref(LICENSE_KEY);
const appid = ref(APP_ID);
const authFunc = ref(auth);
const setBlurBackground = ref(false);
const imagePath =
  "https://webar-static.tencent-cloud.com/assets/back-new/class.png";

const props = defineProps({
  localClass: {
    type: String,
    required: true,
  },
});
const emit = defineEmits(["toggleViewSize"]);
```

```
const handlePusherChange = (value) => {
  pusher.value = Object.assign({}, pusher.value, value);
};

const handleCallStatusChange = (value) => {
  callStatus.value = value;
};

const handleBlurBackgroundChange = (value) => {
  setBlurBackground.value = value;
};

const watchOptions = {
  [NAME.PUSHER]: handlePusherChange,
  [NAME.CALL_STATUS]: handleCallStatusChange,
  [NAME.ENABLE_VIRTUAL_BACKGROUND]: handleBlurBackgroundChange,
};

onMounted(() => {
  TUIStore.watch(StoreName.CALL, watchOptions, {
    notifyRangeWhenWatch: NAME.MYSELF,
  });
});

onUnmounted(() => {
  TUIStore.unwatch(StoreName.CALL, watchOptions);
});

function pusherStateChangeHandler(e: any) {
  TUICallKitServer._tuiCallEngine._pusherStateChangeHandler(e);
}

function pusherNetStatus(e: any) {
  TUICallKitServer._tuiCallEngine._pusherNetStatus(e);
}

function pusherErrorHandler(e: any) {
  TUICallKitServer._tuiCallEngine._pusherNetStatus(e);
}

function pusherAudioVolumeNotify(e: any) {
  TUICallKitServer._tuiCallEngine._pusherAudioVolumeNotify(e);
}
</script>

<style lang="scss" src="./style/index.scss" scoped>
</style>
```

步骤五：开启模糊背景

通过调用以下接口，您可以在 UI 上显示虚拟背景的功能按钮。

> **说明：**
> WeChat ≥ v3.2.5 支持。

```
TUICallKitServer.enableVirtualBackground(true)
```

微信小程序

版权所有：腾讯云计算（北京）有限责任公司

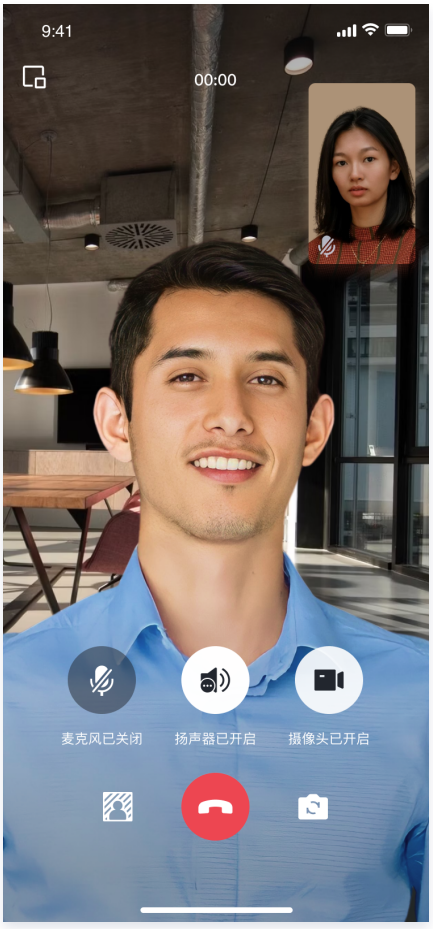
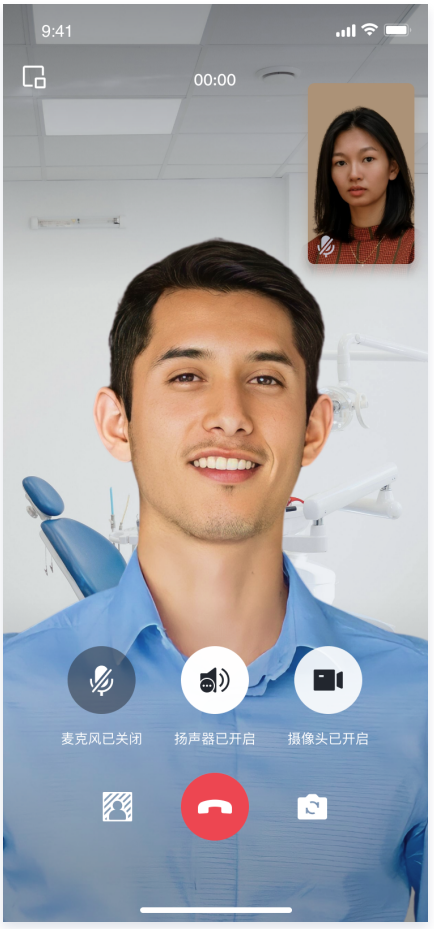
最近更新时间：2024-09-20 21:28:31

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

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

> **注意：**
> 小程序暂时**不支持**模糊背景效果。

原始摄像头	图片背景效果
	

准备条件

TUICallKit 虚拟背景能力基于 腾讯特效 SDK，请首先申请试用或购买视立方 WEBAR License，详见 腾讯云视立方控制台。

开始使用

> **警告：**
> 在集成虚拟背景能力之前，请阅读 TRTC TUICallKit 快速接入(原生小程序)，推荐使用**分包方式**集成 TUICallKit，并完成调试预览相关操作。

步骤一：添加插件

首先要在小程序管理后台的**设置 > 第三方设置 > 插件管理**中**添加插件**。开发者可登录小程序管理后台，搜索 视立方WEBAR 申请添加插件。



步骤二：引用插件

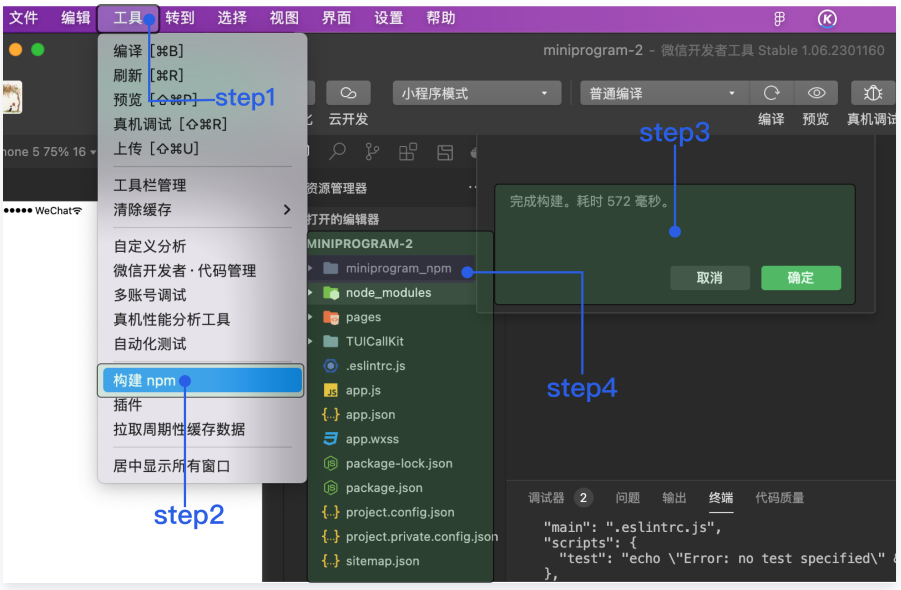
1. 使用组件前需在小程序工程的 app.json 中声明要使用的插件：

```
"plugins": {
  "webarPlugin": {
    "version": "latest",
    "provider": "wx04445cff065100ed"
  }
}
```

2. 在小程序目录安装 npm 包: tencentcloud-webar-wx

```
npm install tencentcloud-webar-wx crypto-js
```

3. 构建 npm，打开微信开发者工具单击工具 > 构建 npm，新增 miniprogram_npm 目录。目录如下：



步骤三：引入美颜特效 SDK

1. 在 TUICallKit/debug 目录下新增 webar-auth.js，实例代码如下：

```
import crypto from "crypto-js";
/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APP_ID = "";
/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/web)，创建项目即可获得 LicenseKey
 */
const LICENSE_KEY = "";
/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提供，前端调用拉取签名，参考
 * [签名方法]
(https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8D.E6.96.B9.E6.B3.95)
 */
const TOKEN = "";
const sha256 = (str) => {
  return crypto.SHA256(str).toString();
};

// 此方法仅测试时使用，为防止 token 泄露，发布时请使用服务端加密，详见[腾讯云-腾讯特效 SDK 官网文档]
(https://cloud.tencent.com/document/product/616/71364)
function authFunc() {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(
    timestamp + TOKEN + APP_ID + timestamp
  ).toUpperCase();
  return { signature, timestamp };
}
export { LICENSE_KEY, APP_ID, authFunc };
```

2. 修改 **TUICallKit/component/SingleCall/SingleCall.json 文件**，引入 `webar-live-pusher` 组件。

```
{
  "usingComponents": {
    "webar-live-pusher": "tencentcloud-webar-wx/webar-live-pusher"
  }
}
```

3. 修改 **TUICallKit/component/SingleCall/SingleCall.wxml** 文件，将 **live-pusher** 替换为 **webar-live-pusher**。

> ⚠ **注意：**
> 组件 webar-live-pusher 的使用方法和微信原生标签 live-pusher 的方法一致，更多属性可参考微信小程序标签 live-pusher 的文档说明，美颜特效相关参数请参见 组件属性说明 。

```
<webar-live-pusher
  custom-effect
```

```
  licenseKey="{{licenseKey}}"
  appId="{{appId}}"
  authFunc="{{authFunc}}"
  background="{{isVirtualBackground ? imagePath : ''}}"
  // 其他 props 同 live-pusher，此处省略
/>
```

4. 修改 TUICallKit/component/SingleCall/SingleCall.js 文件

> **说明：**
> backgroundUrl 表示开启虚拟背景显示的图片背景，可通过修改图片路径来修改显示的背景图片，目前仅支持网络图片(**不支持本地图片**)

```
import { APP_ID, LICENSE_KEY, authFunc } from '../../../debug/webar-auth';

Component({
  data: {
    // 其他的 data 不变，此处忽略
    appId: APP_ID,
    licenseKey: LICENSE_KEY,
    authFunc,
    imagePath: 'https://webar-static.tencent-cloud.com/assets/back-new/class.png'
   },
  });
```

步骤四：开启模糊背景

通过调用以下接口，您可以在 UI 上显示虚拟背景的功能按钮。

> **说明：**
> 3.0.0 > WeChat ≥ v2.1.5 支持。

```
TUICallKitServer.enableVirtualBackground(true)
```

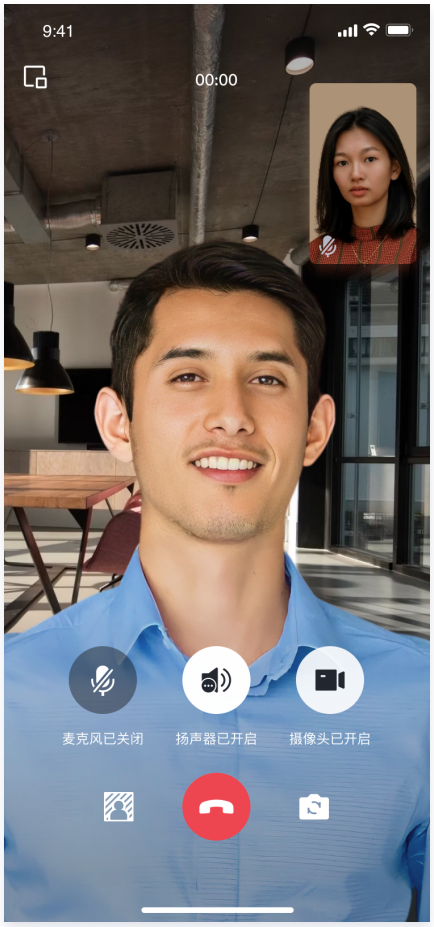
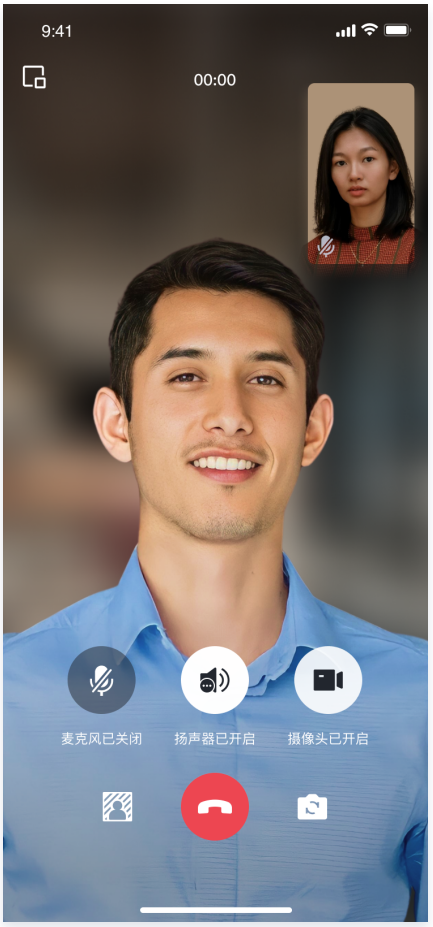
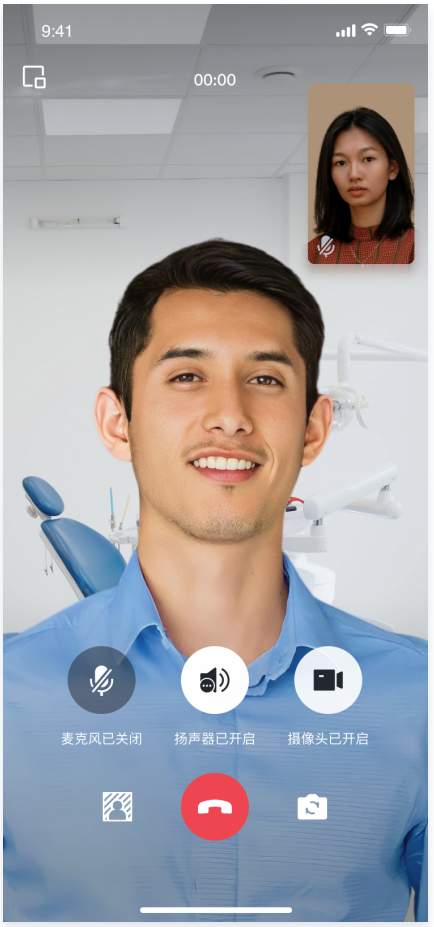
Flutter

最近更新时间：2024-05-17 14:40:41

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

集成效果

TUICallKit 组件接入虚拟背景功能后的显示效果如下：

原始摄像头	模糊背景效果	图片背景效果
		

准备条件

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

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

Android

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

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

iOS

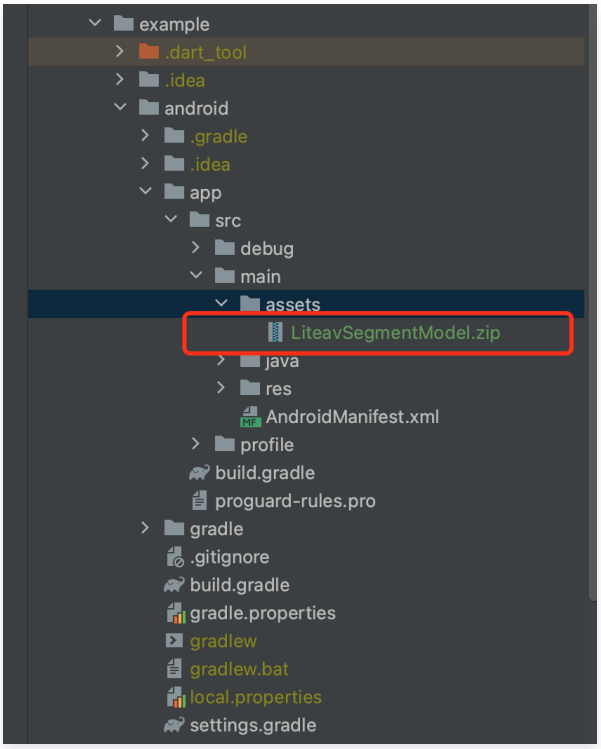
修改您的 Podfile 文件中的依赖，指定 TXLiteAVSDK_Professional 版本， 例如指定为 `11.8.15669`，可根据需求和版本迭代修改。

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

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

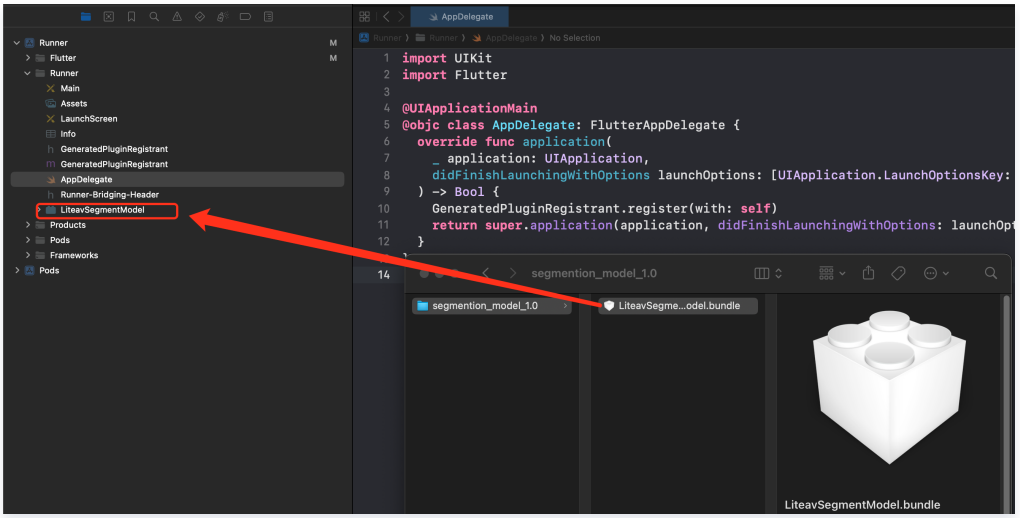
Android

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



iOS

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



开启模糊背景

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

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

设置图片背景（可选）

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

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

常见问题

1. 开启模糊背景无反应或有延迟?

- 确保已购买音视频通话 **群组通话版（140万分钟）** 套餐，详见 开通服务 。
- 确保已下载模型文件至本地。

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

2. 如何将模型文件与 SDK 进行匹配?

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

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

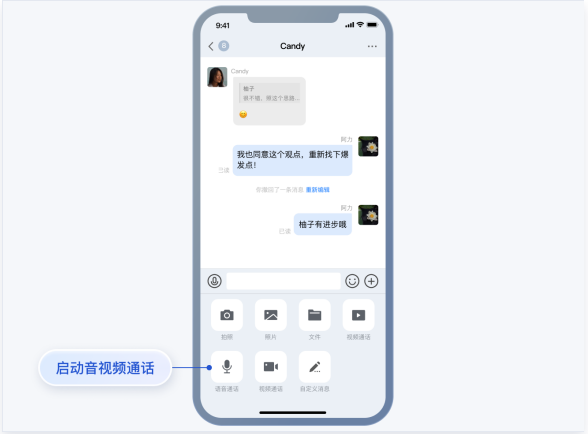
会话聊天（TUICallKit）
Android

最近更新时间：2025-02-18 19:10:02

Chat 提供了多平台的聊天 API、用户界面组件以及服务器端 API，使您能够在十分钟内构建一个功能齐全的聊天应用程序。在聊天应用程序中，您只需编写最多三行代码，即可轻松添加音频和视频通话功能。

集成效果展示

在集成 Chat 和 TUICallKit 后，您可以在聊天消息页面和用户信息页面中轻松发起音视频通话：

消息页发起通话	联系人页发起通话
	

快速集成

准备工作

- 请确保您的应用已经 集成 Chat。

开通服务

1. 登录 即时通信 IM 控制台 ，单击目标应用卡片，进入应用的基础配置页面。
2. 在开通腾讯实时音视频服务功能区，单击**免费体验**即可开通 TUICallKit 的 7 天免费试用服务。
3. 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 IM 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。

集成 TUICallKit

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。

iOS

最近更新时间：2024-12-10 11:16:52

Chat 提供了多平台的聊天 API、用户界面组件以及服务器端 API，使您能够在十分钟内构建一个功能齐全的聊天应用程序。在聊天应用程序中，您只需编写最多三行代码，即可轻松添加音频和视频通话功能。

集成效果展示

在集成 Chat 和 TUICallKit 后，您可以在聊天消息页面和用户信息页面中轻松发起音视频通话：

消息页发起通话	联系人页发起通话
	

快速集成

准备工作

- 请确保您的应用已经 集成 Chat 。

开通服务

1. 登录 即时通信 IM 控制台 ，单击目标应用卡片，进入应用的基础配置页面。
2. 在开通腾讯实时音视频服务功能区，单击**免费体验**即可开通 TUICallKit 的 7 天免费试用服务。
3. 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 IM 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。

集成TUICallKit

在 podfile 文件中添加以下内容：

```
// Integrate the TUICallKit component
pod 'TUICallKit'
```

集成 TUICallKit 组件后，聊天界面和联系人资料界面默认会出现 “视频通话” 和 “语音通话” 两个按钮，当用户点击按钮时，TUICallKit 会自动展示通话邀请 UI，并给对方发起通话邀请请求。

Flutter

最近更新时间：2024-12-10 11:16:52

Chat 提供了多平台的聊天 API、用户界面组件以及服务器端 API，使您能够在十分钟内构建一个功能齐全的聊天应用程序。在聊天应用程序中，您只需编写最多三行代码，即可轻松添加音频和视频通话功能。

集成效果展示

在集成 Chat 和 TUICallKit 后，您可以在聊天消息页面和用户信息页面中轻松发起音视频通话：

消息页发起通话	联系人页发起通话
	

快速集成

准备工作

- 请确保您的应用已经 集成 Chat 。

开通服务

1. 登录 即时通信 IM 控制台 ，单击目标应用卡片，进入应用的基础配置页面。
2. 在开通腾讯实时音视频服务功能区，单击**免费体验**即可开通 TUICallKit 的 7 天免费试用服务。
3. 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 IM 应用相同 SDKAppID 的实时音视频应用，二者账号与鉴权可复用。

集成TUICallKit

1. 在控制台进入您的 Flutter 项目的目录，执行以下命令以安装 tencent_calls_uikit 插件：

```
flutter pub add tencent_calls_uikit
```

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

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

 ......

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

 @override
```

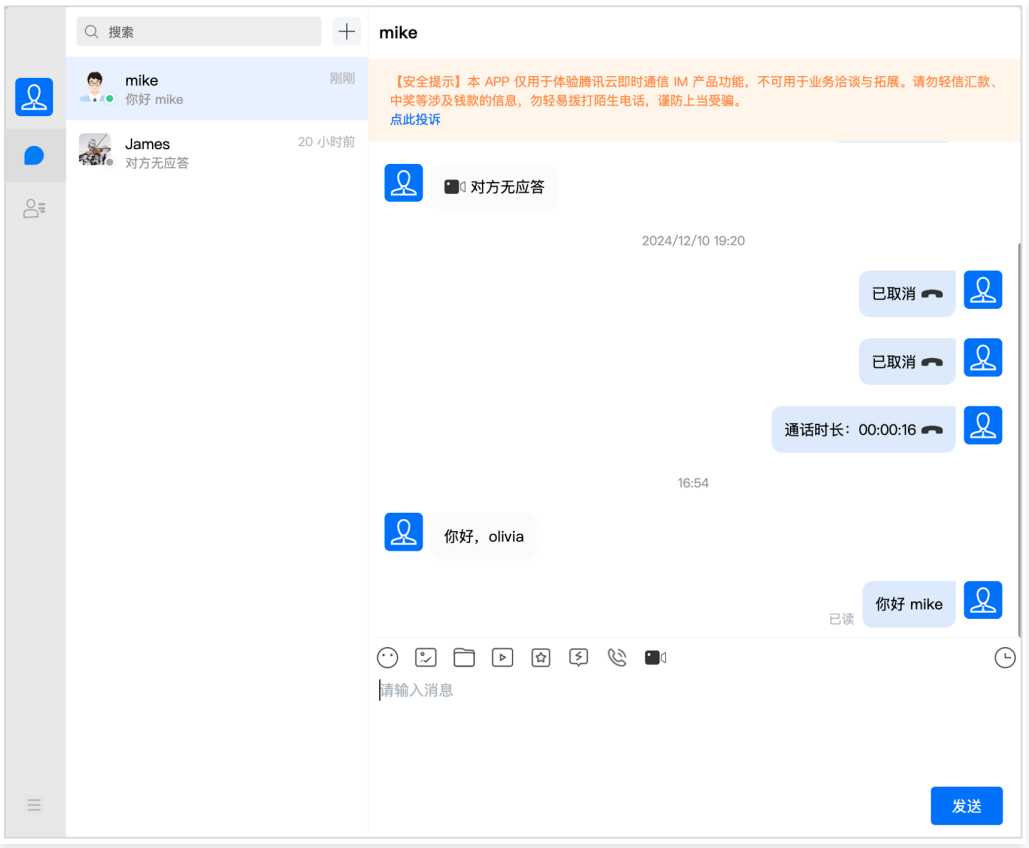
```
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUICallKit.navigatorObserver],
      ......
    );
  }
}
```

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。

Vue

最近更新时间：2025-05-06 17:24:32

通过集成 Chat 和 CallKit 可以实现如下效果，单击 在线体验 。



开发环境要求

- TypeScript
- sass（sass-loader 版本 ⩽ 10.1.1）
- node（node.js ⩾ 16.0.0）
- npm（版本请与 node 版本匹配）

操作步骤

步骤1：集成 Chat

具体步骤请参见 快速集成 Chat 。

步骤2：开通音视频通话能力

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务 。

步骤3：下载 TUICallKit 组件

通过 npm 方式下载 TUICallKit 组件，版本至 v3.3.6 版本及以上：

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

步骤4：引入并调用 TUICallKit 组件

参见以下示例：在引入 `<TUIKit />` 的地方，引入 `<TUICallKit />` 。

```
<template>
  <div id="app">
    <TUIKit :SDKAppID="YOUR_SDKAPPID" userID="YOUR_USERID" userSig="YOUR_USERSIG" />
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScreen="false" />
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue';
</script>
<style lang="scss">
</style>
```

步骤5：启动项目

vite

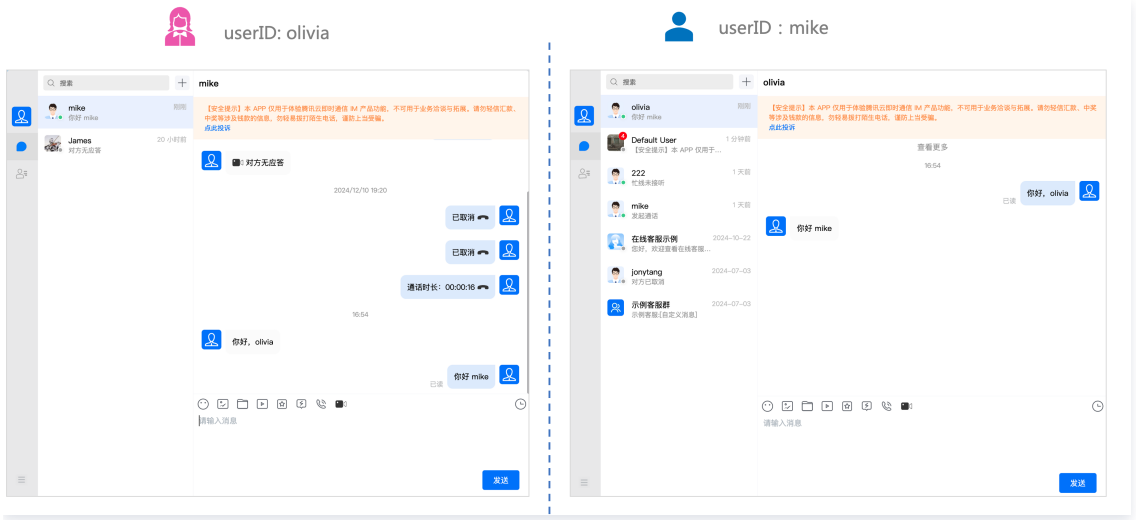
```
npm run dev
```

webpack

```
npm run serve
```

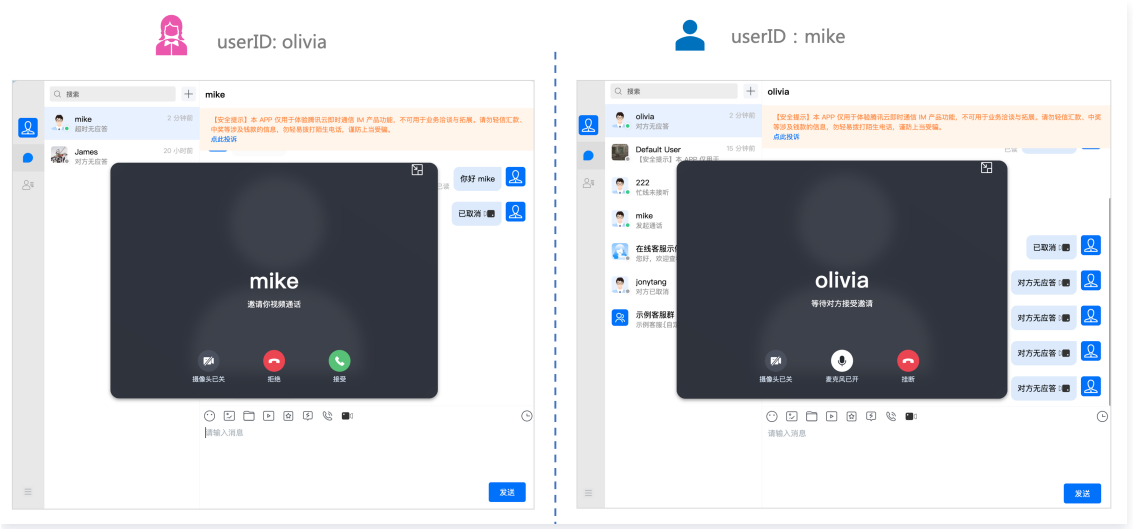
步骤6：发送您的第一条消息

如图，您可以实现与好友之间互发消息。



步骤7：拨打您的第一通电话

如图，点击工具栏的视频/音频 Icon，可以实现通话效果。



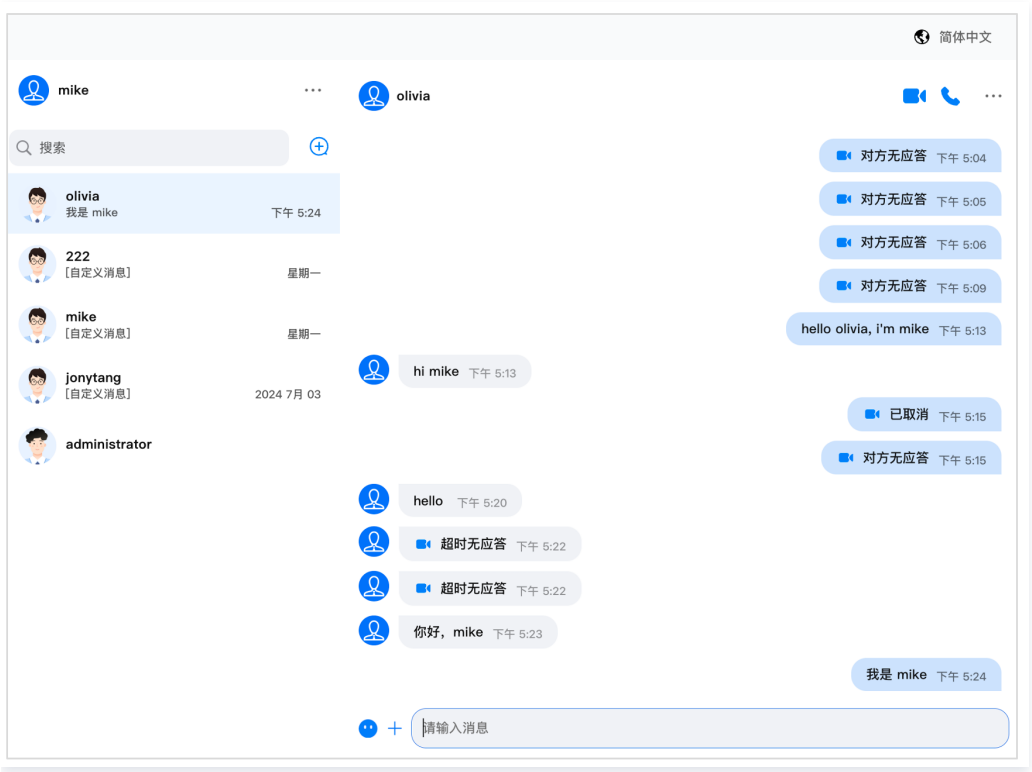
常见问题

点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

React

最近更新时间：2025-01-16 14:47:02

通过集成 Chat 和 CallKit 可以实现如下效果，单击 在线体验 。



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）

步骤一：集成 Chat

具体步骤请参见 快速集成 Chat 。

步骤二：开通音视频通话能力

在使用腾讯云提供的音视频服务前，您需要前往控制台，为应用开通音视频服务。具体步骤请参见 开通服务 。

步骤三：下载 TUICallKit 组件

通过 npm 方式下载 TUICallKit 组件，版本至 v3.3.6 版本及以上:

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

步骤四：引入并调用 TUICallKit 组件

- 仅需添加两行代码，即可体验通话功能。引入 TUICallKit，并调用 TUICallKit 组件。
- 在 <ChatHeader /> 组件中开启 Call，显示 Icon，设置 enableCall 为 ture。

> **说明：**
> 1. 如果您还没有集成 TUIKit，请按照 集成 TUIKit 文档先集成 TUIKit。TUIKit 版本至 v2.2.8 及以上。

2. TUICallKit 组件可以添加 style，用于显示并控制 TUICallKit 的位置、宽高等样式。

```
// 1. 引入 TUICallKit
import { TUICallKit } from '@tencentcloud/call-uikit-react';
```

```
// 2. 如果您是 PC 端展示，请添加 style。初始化 TUICallKit 位置，如果是 H5 展示，则不需要
const callStyle: React.CSSProperties = {
  position: 'fixed',
  top: '50%',
  left: '50%',
  zIndex: '999',
  transform: 'translate(-50%, -50%)',
};
```

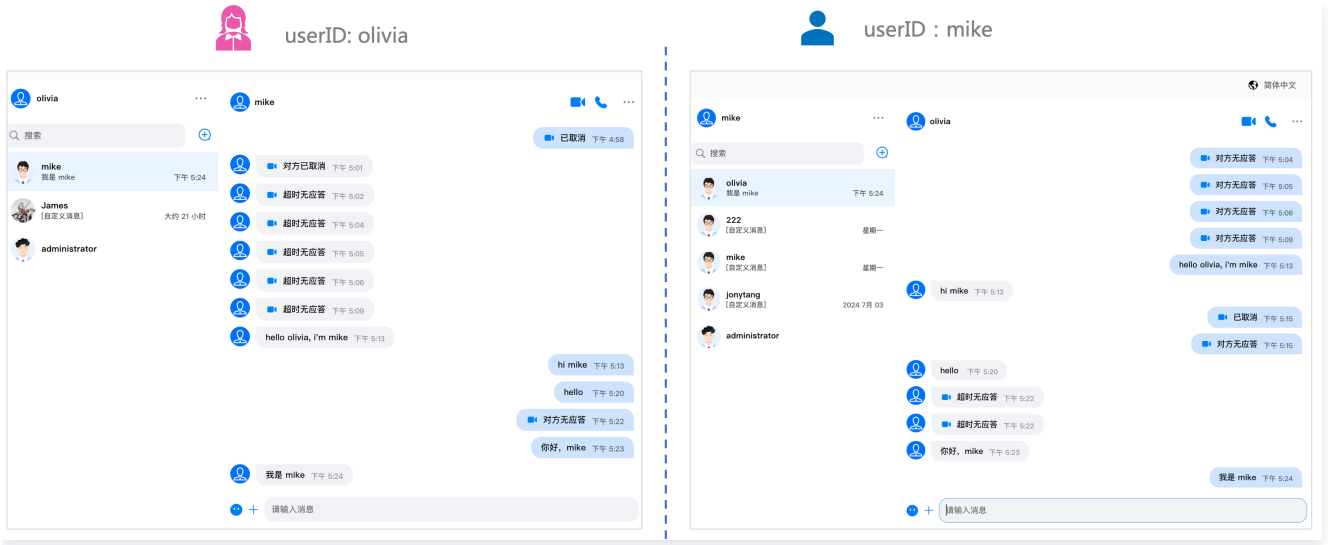
```
// 3. 请在 UIKitProvider 中使用 TUICallKit 组件
return (
  <div style={{display: 'flex', height: '100vh'}}>
    <UIKitProvider language={"en-US"}>
      <TUICallKit style={callStyle} />
      <div style={{maxWidth: '400px'}}>
         <Profile />
         <ConversationList />
      </div>
      <Chat>
        <ChatHeader enableCall={true}/>
        <MessageList />
        <MessageInput />
      </Chat>
      <ChatSetting />
    </UIKitProvider>
  </div>
);
```

步骤五：启动项目

```
npm run start
```

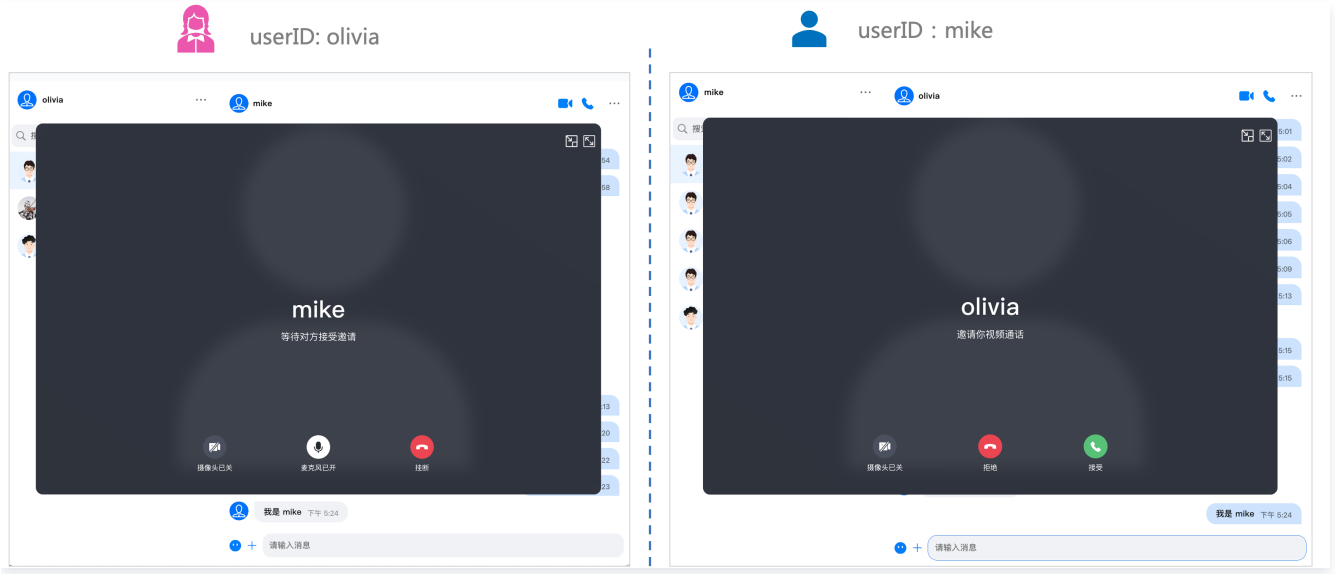
步骤六：发送您的第一条消息

如图，您可以实现与好友之间互发消息。



步骤七：进行您的第一次通话

点击右上角的视频/音频 Icon，拨打您的第一次通话。



常见问题

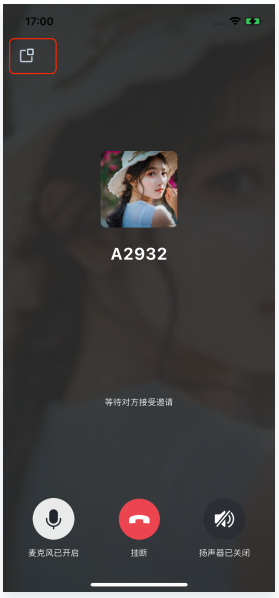
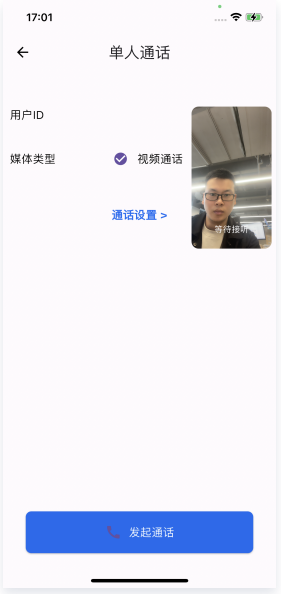
点此进入 IM 社群，享有专业工程师的支持，解决您的难题。

悬浮窗（TUICallKit）
Android&iOS&Flutter

最近更新时间：2024-03-28 15:10:12

本文介绍如何使用悬浮窗功能。

预期效果

开启悬浮窗按钮	语音通话悬浮窗	视频通话悬浮窗
		

悬浮窗功能

TUICallKit 允许用户在通话时使用通话界面左上角的悬浮窗按钮将通话界面缩小为悬浮窗。
您的业务如果需要启用该功能，可以使用 enableFloatWindow 方法，在 TUICallKit 组件初始化时开启该功能：

Android（Kotlin）

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

Android（Java）

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

iOS（Swift）

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

iOS（Objective-C）

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

Flutter（Dart）

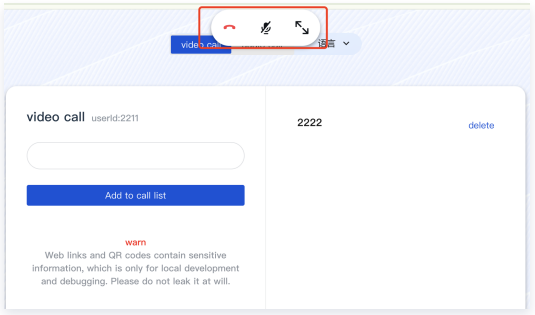
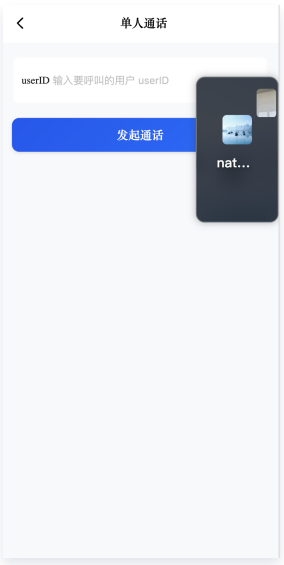
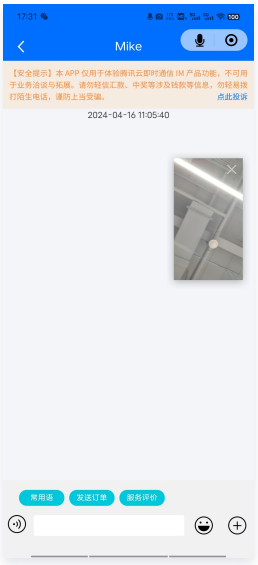
```
TUICallKit.instance.enableFloatWindow(true);
```

Web&H5&uni-app (小程序)

最近更新时间：2024-08-30 15:04:41

本文将介绍悬浮窗功能的使用。

预期效果

Web	H5	uni-app 小程序
		

悬浮窗功能

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

> **说明：**
> Vue&WeChat ⩾ v3.1.0 支持；uni-app 小程序仅有分包使用时支持。

```
TUICallKitServer.enableFloatWindow(true)
```

方式二：通过属性控制悬浮窗、全屏的开启/关闭。

- `allowedMinimized` 属性控制控制悬浮窗开启/关闭。
- `allowedFullScreen` 属性控制全屏的开启/关闭。

> **说明：**
> WeChat 不支持。

React

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

Vue

```
<TUICallKit
```

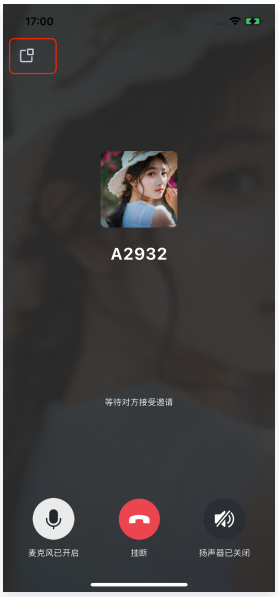
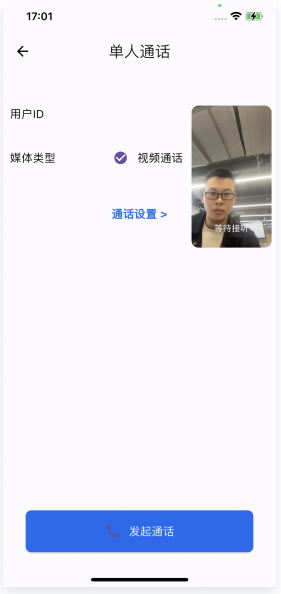
```
    :allowedMinimized="true"
    :allowedFullScreen="true"
  />
```

uni-app（客户端）

最近更新时间：2024-03-28 15:10:12

本文介绍如何使用悬浮窗功能。

预期效果

开启悬浮窗按钮	语音通话悬浮窗	视频通话悬浮窗
		

悬浮窗功能

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

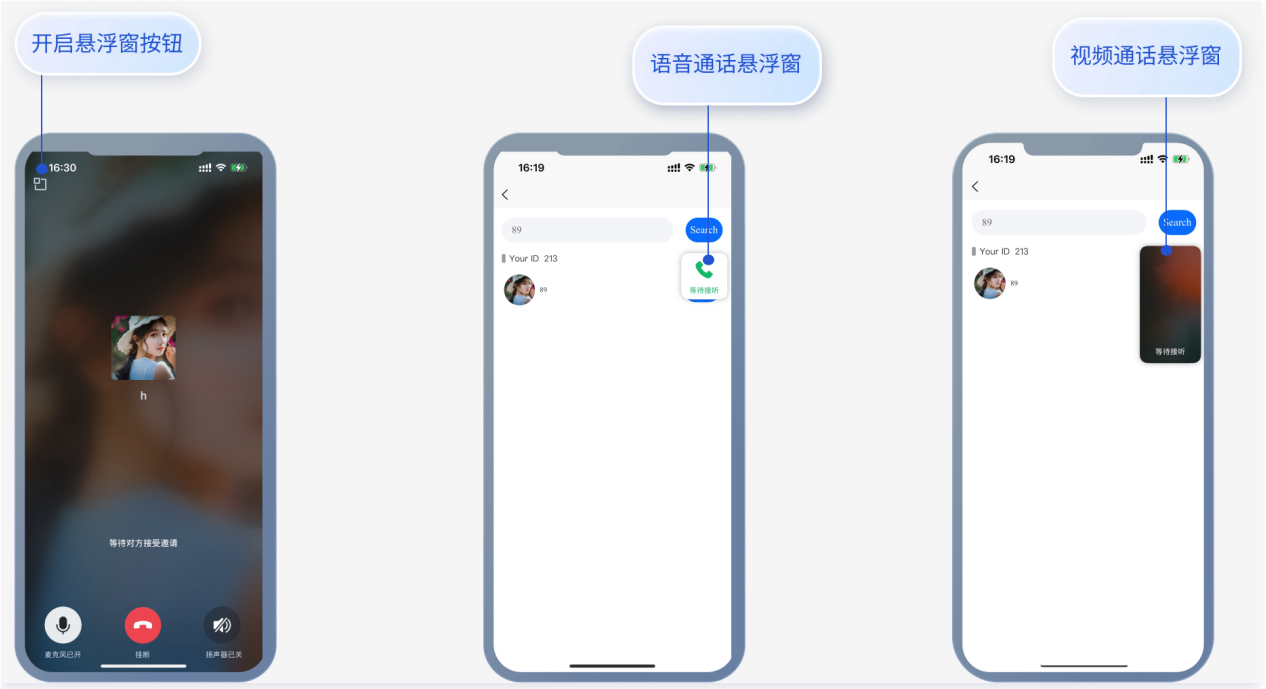
```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const enable = true;
TUICallKit.enableFloatWindow(enable);
```

uni-app (uts 插件)

最近更新时间：2025-06-27 17:35:32

本文介绍如何使用悬浮窗功能。

预期效果



悬浮窗功能

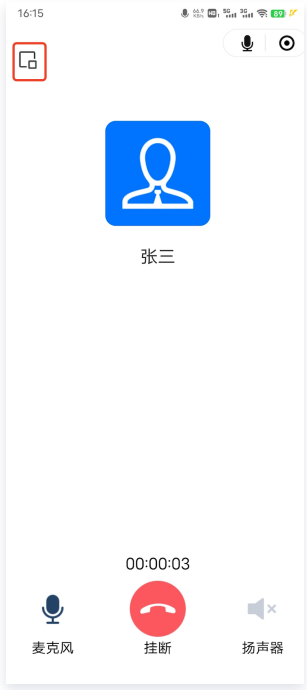
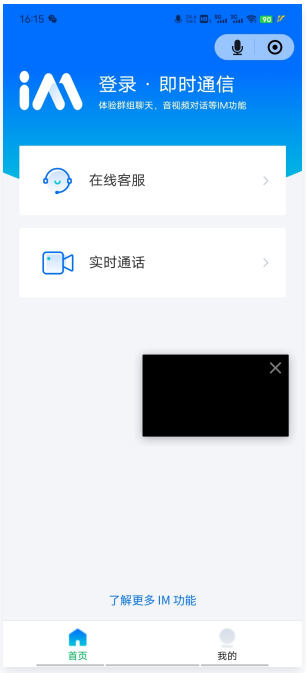
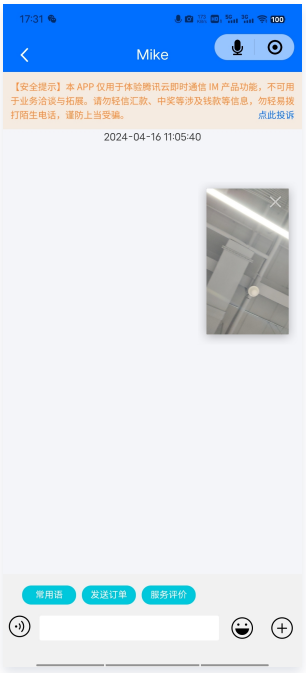
调用 startFloatWindow/stopFloatWindow API 开启/关闭悬浮窗。

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();
// 以开启悬浮窗为例
callEngine.startFloatWindow({
  success: () => {
    console.log(`${NAME.PREFIX} startFloatWindow success `);
  }
  fail: () => {
    console.log(`${NAME.PREFIX} startFloatWindow fail`);
   },
  });
```

原生小程序

最近更新时间：2024-09-26 14:54:01

预期效果

开启悬浮窗按钮	语音通话悬浮窗	视频通话悬浮窗
		

悬浮窗功能

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

说明：

3.0.0 > WeChat ≥ v2.1.3 支持。

```
TUICallKitServer.enableFloatWindow(true)
```

警告：

1. 群组通话/插件接入暂不支持悬浮窗功能。
2. 由于 live-pusher 底层限制，视频通话下的悬浮窗显示本地用户。
3. 悬浮窗 UI 无法自定义，音频通话下的悬浮窗显示黑框。
4. 需要配合全局监听使用。

云端录制（TUICallKit）

最近更新时间：2025-03-07 16:44:32

本文将介绍如何开启 TUICallKit 的云端录制，方便重要通话的存档、审核等，我们提供了两个方案供您选择：自动录制方案 和 REST API 录制方案。

> **说明：**
> TUICallKit 整合有多个腾讯云基础的 PaaS 服务，其中音视频相关能力依赖于实时音视频 TRTC，因为 TUICallKit 的云端录制功能需要进入 实时音视频控制台 进行配置。

方案一：自动录制方案（推荐）

我们推荐您使用自动录制的方案，录制不需业务方来启动和停止，由腾讯云实时音视频后台来管理录制任务，通话过程中有音视频流上行时就自动录制，**接入过程快速、简单**，您可以通过如下几步完成：

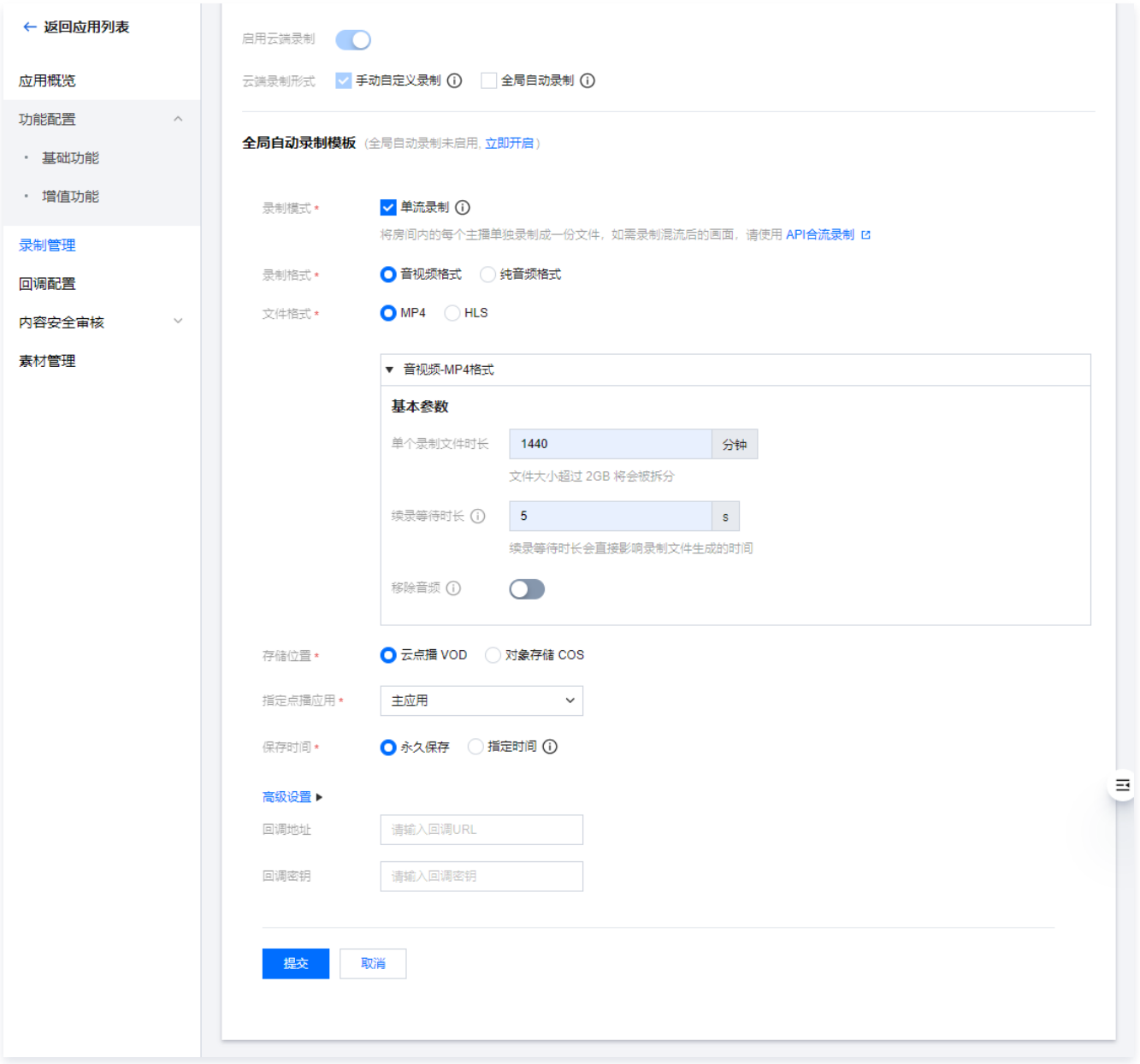
1. 在 实时音视频控制台 > 应用管理 找到对应 SDKAppId 的应用，点击**配置**进入功能配置页，详见下图：



2. 在功能配置页，在左侧选择**录制管理 > 录制模板**，**启用云端录制**功能后，单击**创建全局自动录制模板**。



3. 根据音视频通话的业务场景（1v1通话、群组通话），我们推荐如下配置参数，当然您也可以根据自己的业务需求配置自定义录制模板。



注意：

- **全局录制支持的最大混流人数为8人。如果通话人数超过8人（包括自己），则最后一个用户的流无法被录制。**
- **开启全局自动录制功能后，在通话接听后且有音视频上行，就会触发自动启动录制任务，在通话结束后会自动停止录制，如果因为网络或者其他情况异常退房，我们的录制后台会根据您设置的 MaxIdleTime 值（空闲等待时间，默认5s）自动停止录制任务，避免造成额外的计费损失。**

4. **模板停止进程创建完成后，勾选全局自动录制即可完成。**



方案二：REST API 录制方案

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

1. 在 实时音视频控制台 > 应用管理 找到对应 SDKAppId 的应用，点击**配置**进入功能配置页，详见下图：



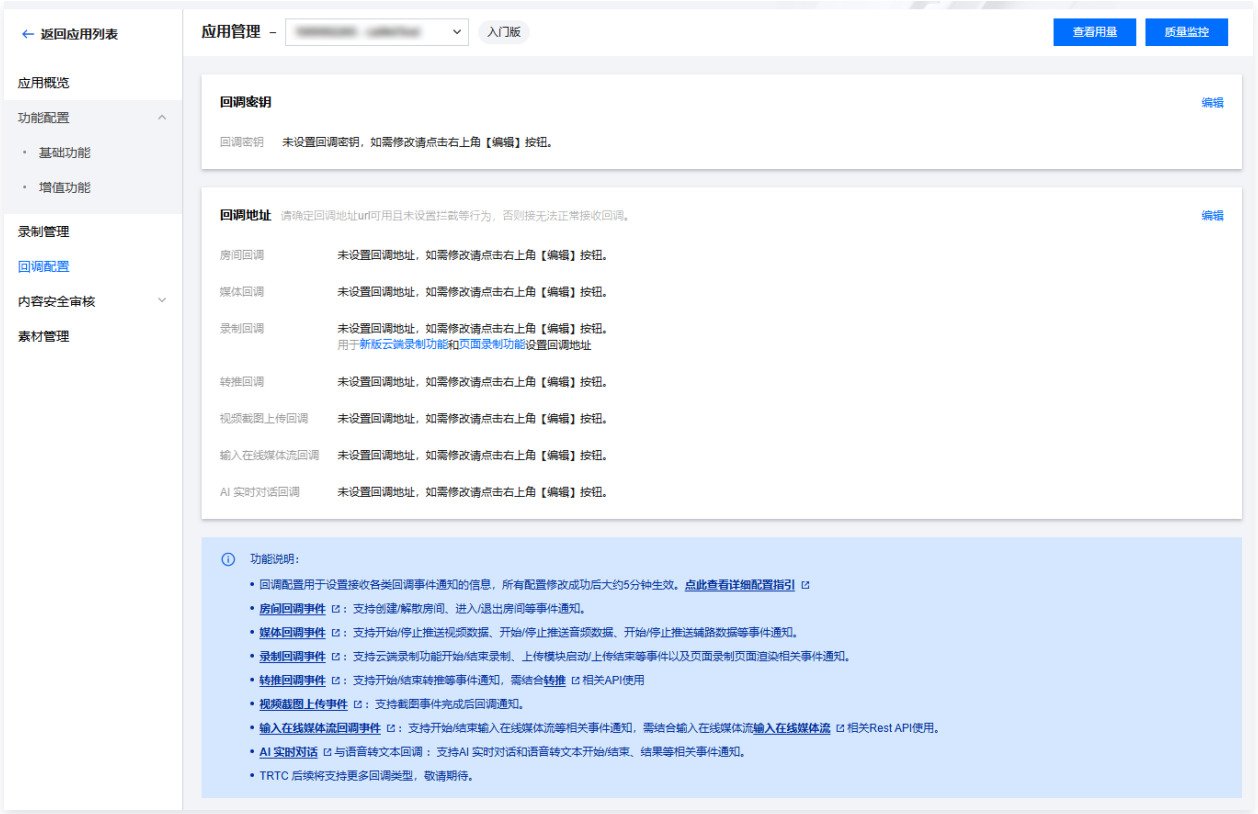
2. 在功能配置页，您可以看到云端录制配置的卡片，**启用云端录制**功能，此时默认勾选**手动自定义录制**，即 REST API 模式。



3. 然后您就可以调用 REST API（CreateCloudRecording）来启动云端的录制，这里建议您可以通过监听 TUICallObserver 的通知事件，在音视频通话开始的时候开启录制，以 Java 代码为例：

```
TUICallEngine.createInstance(context).addObserver(new TUICallObserver() {
    @Override
    public void onCallBegin(TUICommonDefine.RoomId roomId, TUICallDefine.MediaType callMediaType,
TUICallDefine.Role callRole) {
        // 通知您的业务后台，使用REST API 启动录制任务。
    }
});
```

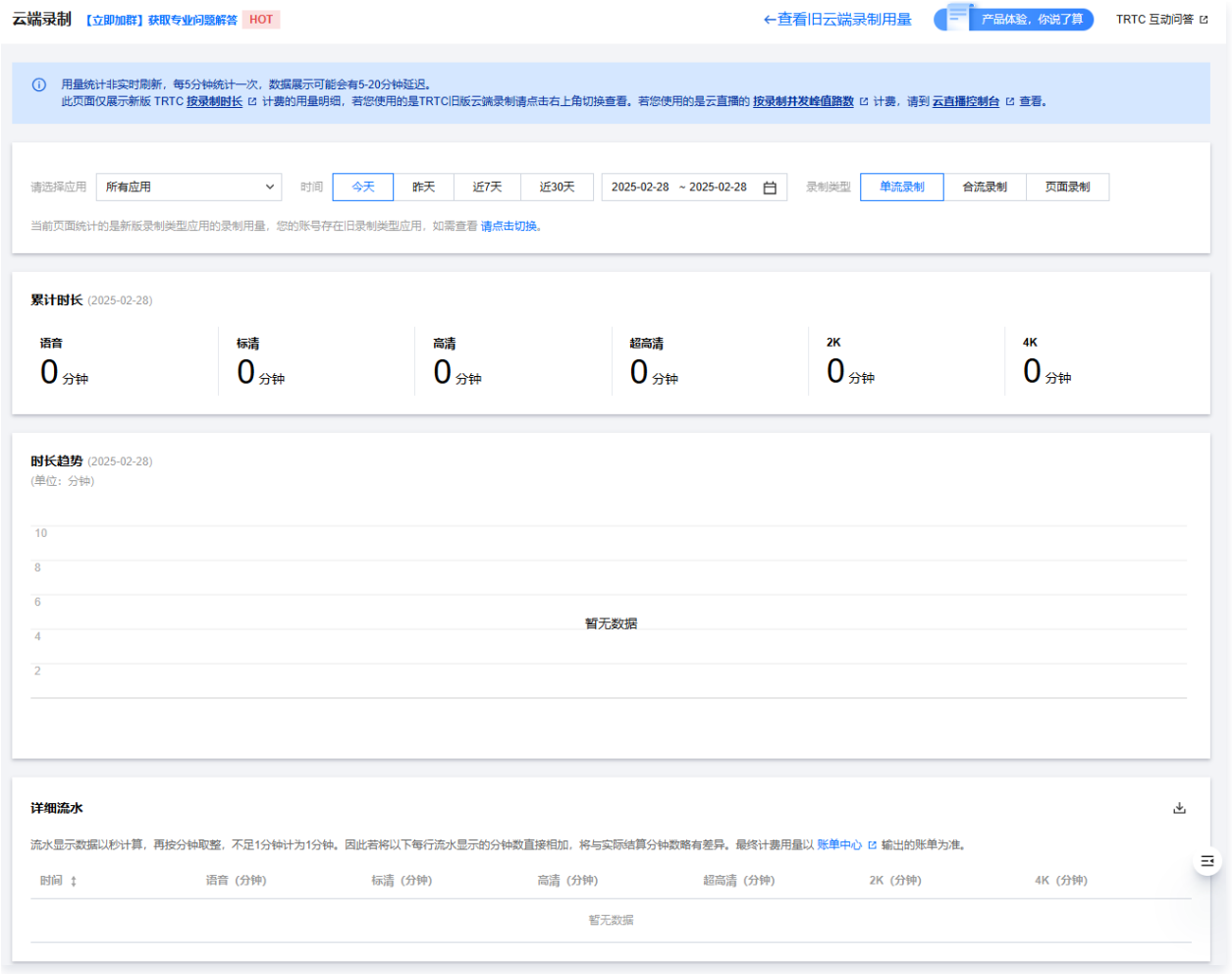
4. 考虑到客户端有可能会存在网络差、停止进程的情况造成的通话异常挂断，所以如何结束录制，建议您可以通过订阅 TRTC 房间状态的回调（详情请参见 监听服务端事件回调），在收到 TRTC 房间状态解散的回调后，调用 REST API（DeleteCloudRecording）来停止云端的录制任务。



常见问题

1. 如何查看录制时长明细?

您可以在 实时音视频控制台 > 云端录制 查看录制的一些时长明细：



2. 如何查看录制的文件?

您可以登录 云点播控制台，在左侧导航栏选择**媒资管理**，单击列表上方的**前缀搜索**，选择**前缀搜索**，在搜索框输入关键词，录制文件命名规则如下：

- 单流录制 MP4 文件名规则：`<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Index>.mp4`
- 合流录制 MP4 文件名规则：`<SdkAppId>_<RoomId>_<Index>.mp4`

更多文档

- 实现云端录制与回放
- 监听服务端事件回调
- 云端录制计费说明

USB 摄像头（TUICallKit）

最近更新时间：2024-09-09 17:00:01

本文介绍了在 Android 系统设备上连接 USB 摄像头后，如何与 `TUICallKit` 组件相结合，以实现音视频通话的功能。目前该功能支持原生 Android TUICallkit 组件和 Flutter（Android）TUICallKit 组件。

集成效果

实物连接图	视频通话效果
	

环境准备

- Android 5.0（SDK API Level 21）及以上版本。
- Gradle 4.2.1 及以上的版本。
- Android 系统设备：手机、平板或其他定制设备。

步骤一：准备条件

1. 在使用腾讯云提供的 USB 摄像头功能前，您需要前往控制台，为应用开通音视频服务，购买群组通话版套餐。具体步骤请参见 开通服务。
2. 本插件需要与 TUICallKit 组件结合使用，请先接入 TUICallKit 组件：
   - Android
   - Flutter

步骤二：集成组件

在工程的 app 目录下的 `build.gradle` 文件中，添加以下依赖代码：

```
implementation "io.trtc.uikit:usb-camera:latest.release"
```

完成上述步骤后，您可以实现在 `TUICallKit` 组件中使用外接摄像头进行视频通话。

常见问题

Android 9 和 Android 10 版本的手机上无法开启摄像头?

问题原因：Android 9 和 Android 10 上申请 USB 权限的时候会检查 Camera 的权限，但是 Android 框架层检查出现异常，即使给了摄像头权限，还是会检查失败，导致无法申请 USB 权限。
解决方式：将应用的 targetSdkVersion 设置为 27 或以下版本。

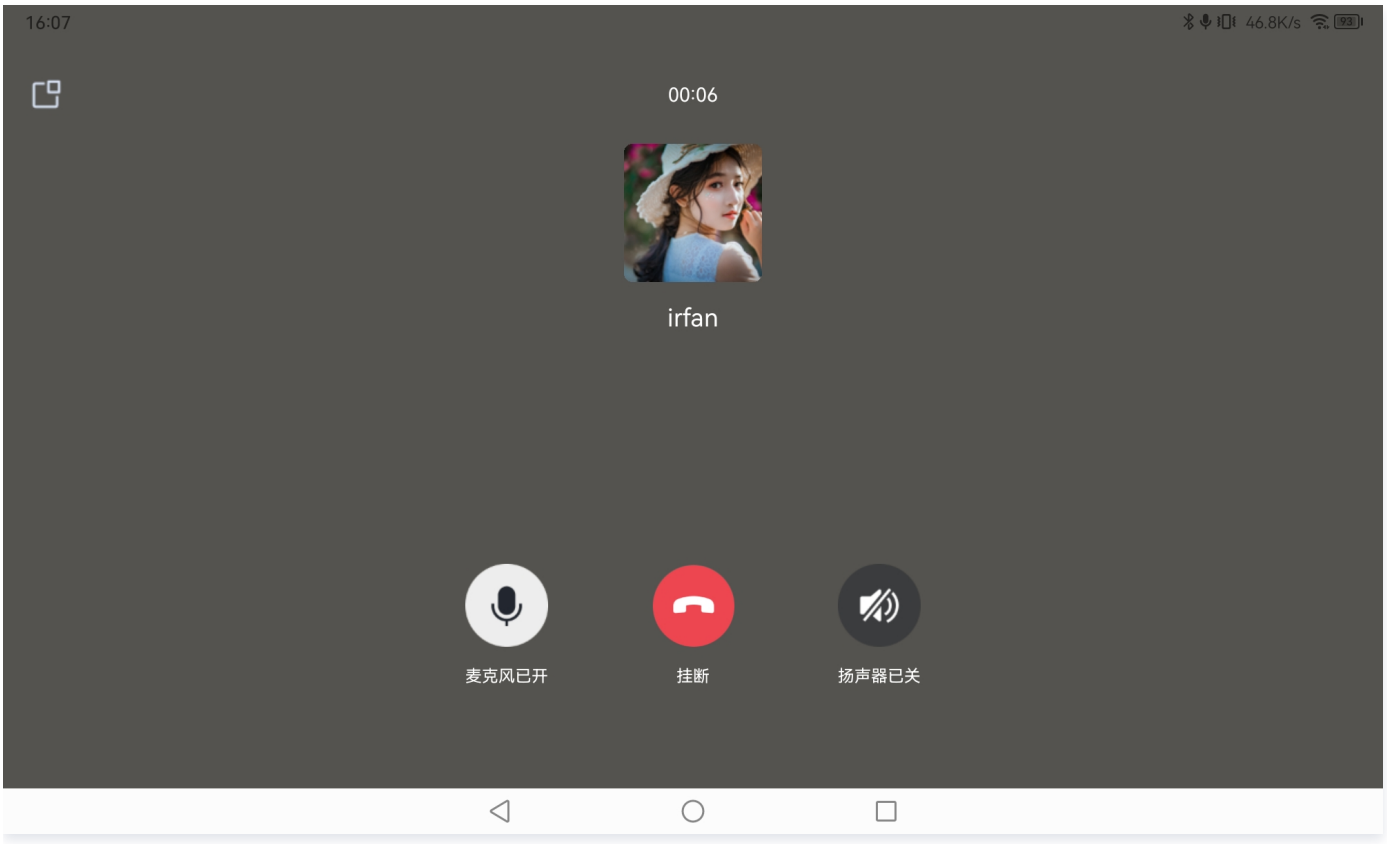
```
defaultConfig {
    targetSdkVersion 27
}
```

更多特性（TUICallKit）
大屏设备

最近更新时间：2025-03-04 09:00:22

在医疗、1v1付费咨询、社区安防等场景中，Android 大屏设备（如会议平板、智能终端）的使用日益普及。为确保视频通话界面在这些设备上完美适配，TUICallKit 提供**屏幕方向控制**及**布局优化**能力，帮助业务方实现全屏沉浸式体验。

预期效果



通话中效果

功能开启

1. Android

```
// Kotlin
// 0-Portrait, 1-LandScape, 2-Auto;   default value: 0
TUICallKit.createInstance(context).setScreenOrientation(1)
```

2. uni-app

```
// Javascrpit
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
// 0-Portrait, 1-LandScape, 2-Auto;   default value: 0
TUICallKit.setScreenOrientation(1);
```

说明：
目前，uni-app（客户端）版本的 TUICallKit 仅支持 Android 平台。如果您需要在 iOS 平台等其他平台上使用，请 联系我们 。

更多设置

- 设置视频渲染参数（uni-app）
- 设置视频渲染参数（Android）

版权所有：腾讯云计算（北京）有限责任公司

设置昵称、头像（全平台）

最近更新时间：2025-06-27 17:35:32

本文介绍如何设置用户的头像和昵称。

设置头像、昵称

如果您需要自定义昵称或头像，可以使用如下接口进行更新：

Android（Kotlin）

```
TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png", callback)
```

Android（Java）

```
TUICallKit.createInstance(context).setSelfInfo("jack", "https:/****/user_avatar.png", callback);
```

iOS（Objective-C）

```
[[TUICallKit createInstance] setSelfInfo:@"" avatar:@"" succ:^{

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

}];
```

iOS（Swift）

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

} fail: { code, message in

}
```

Flutter（Dart）

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

uni-app(Andorid&iOS)

```
const options = {
    nickName: 'jack',
    avatar: 'https:/****/user_avatar.png'
}
TUICallKit.setSelfInfo(options, (res) => {
    if (res.code === 0) {
        console.log('setSelfInfo success');
    } else {
        console.log(`setSelfInfo failed, error message = ${res.msg}`);
    }
});
```

Web&H5

```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "jack", avatar: "http://xxx" });
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`);
}
```

uni-app (uts)

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();
const options = {
    nickName: 'jack',
    avatar: 'https:/****/user_avatar.png'
};
callEngine.setSelfInfo(options);
```

注意：
因为用户隐私限制，非好友之间的通话，被叫的昵称和头像更新可能会有延迟，一次通话成功后就会顺利更新。

设置分辨率、填充模式（Web）

最近更新时间：2024-03-28 15:10:12

本文介绍如何设置分辨率、填充模式。

设置分辨率、填充模式

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

- `videoDisplayMode`：画面显示模式。
- `videoResolution`：通话分辨率。

React

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

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

Vue

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

<TUICallKit
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P" />
```

videoDisplayMode

画面显示模式 `videoDisplayMode` 属性有三个值：

- `VideoDisplayMode.CONTAIN`
- `VideoDisplayMode.COVER`
- `VideoDisplayMode.FILL`，默认值是 `VideoDisplayMode.COVER` 。

属性	值	描述
videoDisplayMode	VideoDisplayMode.CONTAIN	• 优先保证视频内容全部显示。 • 视频尺寸等比缩放，直至视频窗口的一边与视窗边框对齐。 • 如果视频尺寸与显示视窗尺寸不一致，在保持长宽比的前提下，将视频进行缩放后填满视窗，缩放后的视频四周会有一圈黑边。
	VideoDisplayMode.COVER	• 优先保证视窗被填满。 • 视频尺寸等比缩放，直至整个视窗被视频填满。 • 如果视频长宽与显示窗口不同，则视频流会按照显示视窗的比例进行周边裁剪或图像拉伸后填满视窗。
	VideoDisplayMode.FILL	• 保证视窗被填满的同时保证视频内容全部显示，但是不保证视频尺寸比例不变。 • 视频的宽高会被拉伸至和视窗尺寸一致。

videoResolution

分辨率 `videoResolution` 有三个值：

- `VideoResolution.RESOLUTION_480P`
- `VideoResolution.RESOLUTION_720P`

- `VideoResolution.RESOLUTION_1080P`，默认值是`VideoResolution.RESOLUTION_480P`。

分辨率说明：

视频 Profile	分辨率（宽 × 高）	帧率（fps）	码率（kbps）
480p	640 × 480	15	900
720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

常见问题：

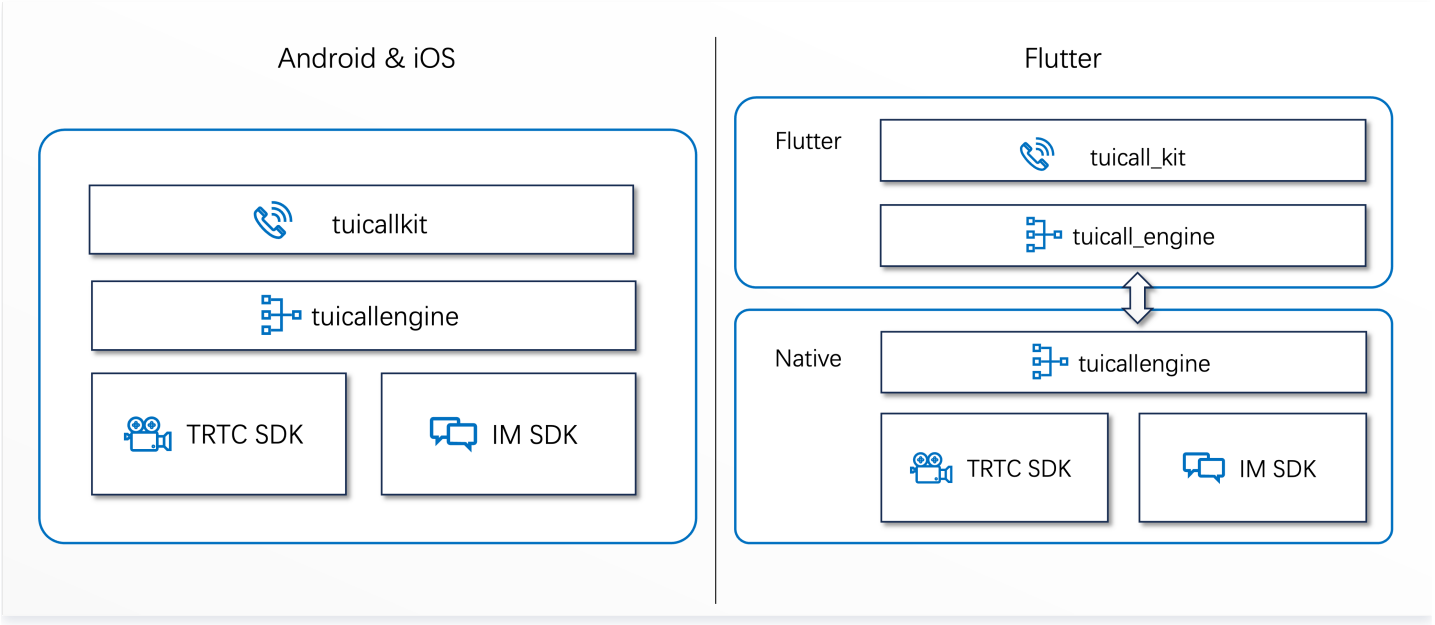
- iOS 13&14不支持编码高于 720P 的视频，建议在这两个系统版本限制最高采集 720P。参见 iOS Safari 已知问题 case 12。
- Firefox 不支持自定义视频帧率（默认为 30fps）。
- 受系统性能占用，摄像头采集能力和浏览器限制等因素的影响，视频分辨率、帧率、码率的实际值不一定能够完全匹配设定值，在这种情况下，浏览器会自动调整 Profile 尽可能匹配设定值。

体积增量

最近更新时间：2024-12-10 11:16:52

本文将详细分析在基于不同开发框架的应用中集成 TUICallKit 可能带来的体积增量。

TUICallKit 组件结构



TUICallKit 体积增量分析

以 TUICallKit 2.6.0 为例，以下是不同开发框架集成之后各平台输出的应用的体积增量：

Android & iOS

系统	CPU 架构	增量
Android	arm64-v8a	16.2 MB
	armeabi-v7a	14.8 MB
iOS	arm64	15.5 MB

Flutter

系统	CPU 架构	增量
Android	arm64-v8a	14.9 MB
	armeabi-v7a	13.7 MB
iOS	arm64	15.5 MB

注意：

- 在 Android 平台的体积测试中，默认启用了 jniLibs 和 dex 的压缩功能。
- 本文档中的体积增量数据仅供参考。实际增量可能因插件版本、依赖项、构建配置、平台和资源文件等因素而有所波动。建议在集成插件后进行实际构建以获取准确数据。

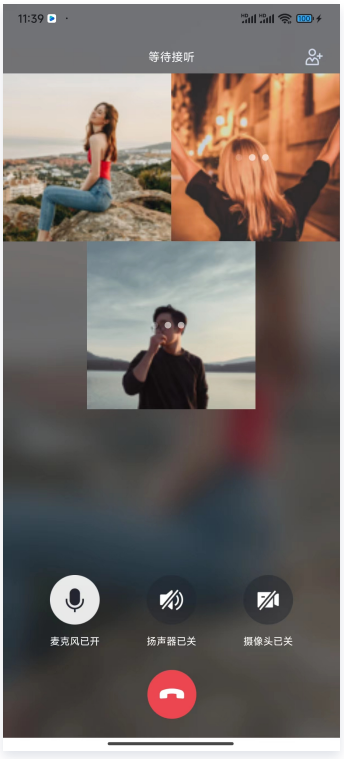
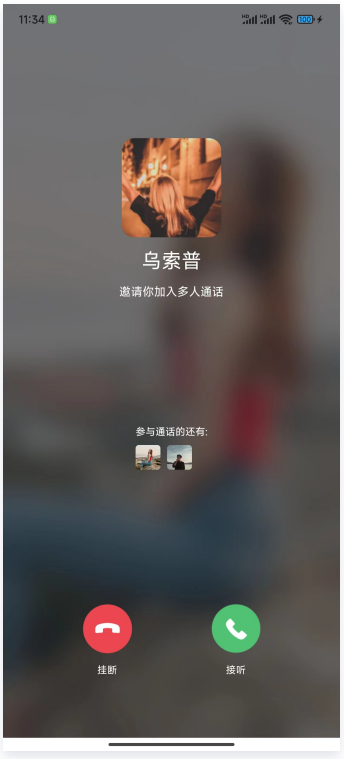
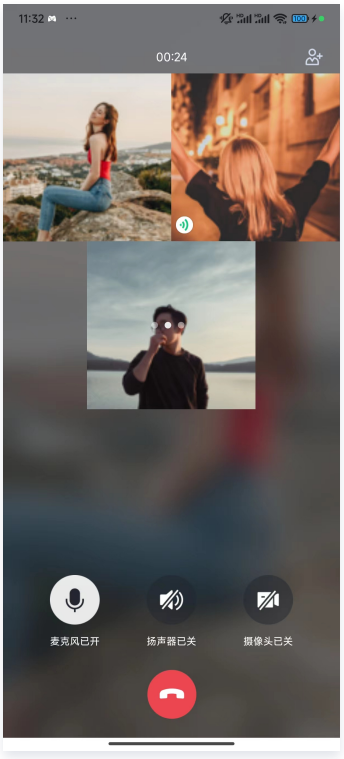
多人通话
Android&iOS&Flutter

最近更新时间：2025-06-10 15:36:11

本文介绍多人通话功能的使用，如发起多人通话、加入通话。

预期效果

TUICallKit 支持多人通话（9人及以下），预期效果见下图。

发起多人通话	收到多人通话请求	接受多人通话请求
		

发起多人通话

调用 groupCall API 发起群通话。

Android（Kotlin）

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

val list = mutableListOf<String>()
list.add("mike")
list.add("tate")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null, null)
```

Android（Java）

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

```
List<String> list = new ArrayList<>();
list.add("mike")
list.add("tate")
TUICallKit.createInstance(context).calls(list, TUICallDefine.MediaType.Audio, null, null);
```

iOS（Swift）

```
import TUICallKit_Swift
import RTCRoomEngine

TUICallKit.createInstance().calls(userIdList: ["mike","tate"], callMediaType: .audio, params: nil) {
} fail: { code, message in
}
```

iOS（Objective-C）

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

[[TUICallKit createInstance] calls:@[@"mike", @"tate"] callMediaType:TUICallMediaTypeAudio params:NULL
succ:^{
} fail:^(int code, NSString * _Nullable errMsg) {
}];
```

Flutter（Dart）

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

void call() {
    List<String> userIdList = ['vince','mike'];
    TUICallKit.instance.call(userIdList, TUICallMediaType.audio);
}
```

加入通话

调用 join API 主动加入已有的音视频通话。

Android（Kotlin）

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

TUICallKit.createInstance(context).join("12345678")
```

Android（Java）

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

TUICallKit.createInstance(context).join("*****");
```

iOS（Swift）

```
import TUICallKit_Swift
```

```
TUICallKit.createInstance().join(callId: "")
```

iOS（Objective-C）

```
#import "TUICallKit_Swift-Swift.h"

[[TUICallKit createInstance] joinWithCallId: @"***"];
```

Flutter（Dart）

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

void join() {
    TUICallKit.instance.join("*****")
}
```

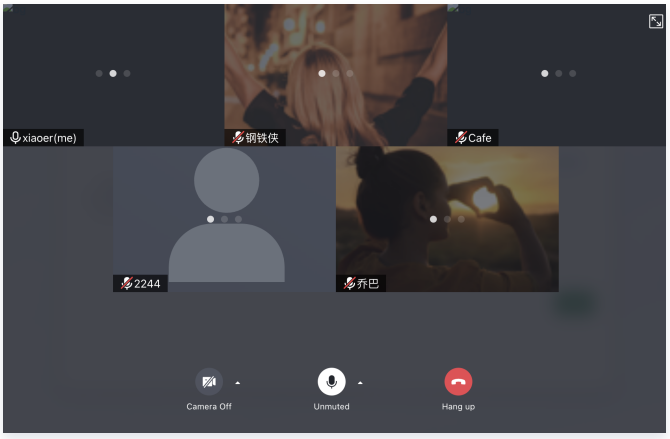
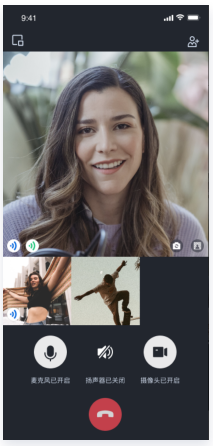
Web&H5&小程序

最近更新时间：2025-05-27 16:21:12

本文介绍多人通话功能的使用，例如发起多人通话、加入多人通话。

预期效果

TUICallKit 支持多人通话，预期效果见下图。

Web 端	移动端
	

发起多人通话

调用 `calls` API 发起群通话。

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

加入多人通话

调用 join API 主动加入已有的音视频通话。

```
try {
  await TUICallKitServer.join({callId: 'xxx'});
} catch (error: any) {
  alert(`[TUICallKit] join failed. Reason: ${error}`);
}
```

uni-app (uts 插件)

最近更新时间：2025-06-27 17:35:32

绍群组通话功能的使用，例如发起多人通话、加入多人通话。

预期效果

TUICallKit 支持多人通话，预期效果见下图。



发起多人通话

调用 calls API 发起群通话。

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();
const options = {
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // voice call(callMediaType = 1)、video call(callMediaType = 2)
};
callEngine.calls(options);
```

加入多人通话

调用 join API 主动加入群组中已有的音视频通话。

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();
const options = {
    callId: '1234',
};
callEngine.join(options);
```

版权所有：腾讯云计算（北京）有限责任公司

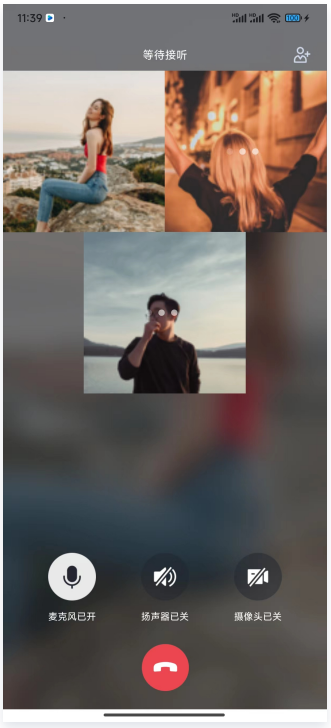
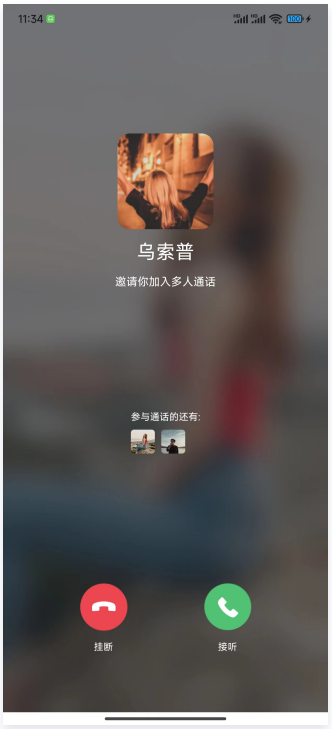
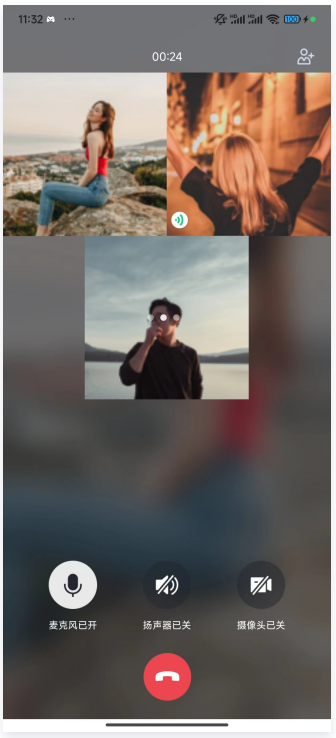
uni-app（客户端）

最近更新时间：2025-05-27 16:21:12

本文介绍群组通话功能的使用，例如发起多人通话、加入多人通话。

预期效果

TUICallKit 支持群组通话，预期效果见下图。

发起多人通话	收到多人通话邀请	接受多人通话邀请
		

发起多人通话

调用 calls API 发起群通话。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // voice call(callMediaType = 1)、video call(callMediaType = 2)
};
TUICallKit.calls(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

加入多人通话

调用 join API 主动加入群组中已有的音视频通话。

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

```
const options = {
    callId: 'xxx',
};
TUICallKit.join(options, (res) => {
  if (res.code === 0) {
    console.log('join success');
  } else {
    console.log(`join failed, error message = ${res.msg}`);
  }
});
```

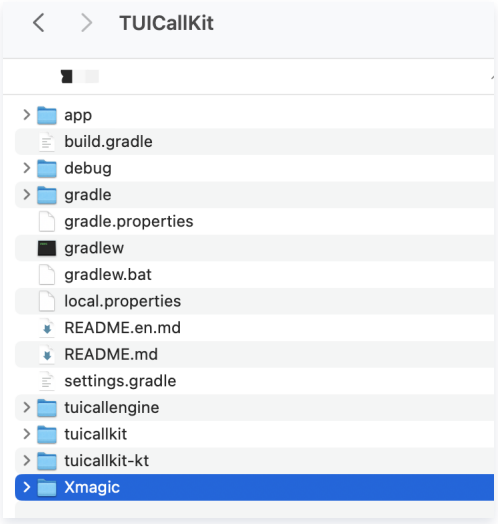
美颜特效
Android

最近更新时间：2023-07-13 15:41:42

TUICallEngine 中提供了 getTRTCCloudInstance() 接口，可以通过该接口调用 TRTC 的高级特性 setLocalVideoProcessListener()，实现接入第三方美颜。
本文将介绍如何在 TUICallKit 中接入 腾讯特效 SDK，其他第三方美颜接入方法类似，请结合第三方 SDK 文档进行接入。

接入准备

按照 腾讯特效 TRTC 接入指引，下载场景化 Demo 的 zip 包，按照文档替换资源，将 `Xmagic` 目录复制到自己的工程中，完成 步骤一；导入后的目录结构如下：



导入 Xmagic

1. 在工程根目录下找到 `settings.gradle` 文件，在其中增加如下代码，将 Xmagic 组件导入到您当前的项目中。

```
include ':Xmagic'
```

2. 在 tuicallkit 目录下找到 `build.gradle` 文件，并在其中增加如下代码，声明 tuicallkit 对新加入的 Xmagic 组件的依赖。

```
api project(':Xmagic')
```

3. 修改 Xmagic 目录下 `build.gradle` 中的 compileSdkVersion、minSdkVersion、TRTC SDK 版本、Glide 版本等与您的工程保持一致；依赖的 Xmagic SDK 版本为您的套餐包版本。

```
api 'com.tencent.liteav:LiteAVSDK_TRTC:11.1.0.13111'
api 'com.github.bumptech.glide:glide:4.12.0'

api 'com.tencent.mediacloud:TencentEffect_S1-04:3.0.0.13'
```

4. 修改 `XMagicImpl.java` 文件，替换自己的 `authKey` 和 `authLicenceUrl`。

```
public static final String authKey = YOUR_LICENSE_KEY;
public static final String authLicenceUrl = YOUR_LICENSE_URL;
```

初始化 Xmagic

Xmagic 提供了美颜面板供使用者调用，TUICallKit 集成的时候，可以在 BaseCallActivity 文件中增加布局，每次拉起通话界面时展示，也可以在显示出通话界面后以 Dialog 的形式展示，注意生命周期即可。可参考 腾讯特效 TRTC 接入指引 中 Demo 程的 `ThirdBeautyActivity` 类。

1. 授权：

```
//鉴权注意事项及错误码详情，请参考
https://cloud.tencent.com/document/product/616/65891#.E6.AD.A5.E9.AA.A4.E4.B8.80.EF.BC.9A.E9.89.B4.E6
.9D.83
XMagicImpl.init(getApplicationContext());

if
(TUICallDefine.MediaType.Video.equals(TUICallingStatusManager.sharedInstance(getApplicationContext())
.getMediaType())) {
    if (XmagicLoadAssetsView.isCopyedRes) {
        initXMagic();
        return;
    }

    XMagicImpl.checkAuth((errorCode, msg) -> {
         if (errorCode == TELicenseCheck.ERROR_OK) {
             showLoadResourceView();
         } else {
             Log.e("Xmagic", "please check url and key" + errorCode + " , " + msg);
         }
    });
  }
```

2. 初始化素材：

```
private void showLoadResourceView() {
     if (XmagicLoadAssetsView.isCopyedRes) {
         XmagicResParser.parseRes(getApplicationContext());
         initXMagic();
     } else {
         loadAssetsView = new XmagicLoadAssetsView(this);
         loadAssetsView.setOnAssetsLoadFinishListener(() -> {
             XmagicResParser.parseRes(getApplicationContext());
             initXMagic();
         });
     }
 }
```

设置纹理回调和添加美颜效果

```
//1. set TRTCVideoFrameListener, refer to: {https://liteav.sdk.qcloud.com/doc/api/zh-
cn/group__TRTCCloud__android.html#a0b565dc8c77df7fb826f0c45d8ad2d85}
TRTCCloud trtcCloud = TUICallEngine.createInstance(this).getTRTCCloudInstance();
trtcCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
        TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoFrameListener() {
            @Override
            public void onGLContextCreated() { //2. GLContext created
              if (isGLContextCreated) {
                  isGLContextCreated = false;
              } else {
                  runOnUiThread(() -> initXMagic());
              }
            }
            @Override
```

```
            public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame,
TRTCCloudDef.TRTCVideoFrame dstFrame) {
            //3. invoke xmagic module, more details refer to
{https://liteav.sdk.qcloud.com/doc/api/zh-
cn/group__TRTCCloudListener__android.html#a22afb08b2a1a18563c7be28c904b166a}
                if (mXMagic != null) {
                    dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId,
srcFrame.width, srcFrame.height);
                }
                return dstFrame.texture.textureId;
            }
            @Override
            public void onGLContextDestory() {
            //4. GLContext destroyed
                if (mXMagic != null) {
                  mXMagic.onDestroy();
                }
            }
});
```

1. **布局中添加 SDK 美颜面板：**

```
 <RelativeLayout
     android:layout_above="@+id/ll_edit_info"
     android:id="@+id/livepusher_bp_beauty_pannel"
     android:layout_width="match_parent"
     android:layout_height="wrap_content" />
```

2. **初始化面板：**

```
   mBeautyPanelView = findViewById(R.id.livepusher_bp_beauty_pannel);

   private void initXMagic() {
       if (mXMagic == null) {
           mXMagic = new XMagicImpl(this, mBeautyPanelView);
       }else {
           mXMagic.onResume();
       }
   }
```

完成上述步骤后，即可在音视频通话中使用第三方美颜。

集成效果

下图是 TUICallKit 含 UI 组件集成腾讯特效 SDK 面板的效果：



常见问题

集成 Xmagic 后，动效、美妆、分割下内容怎么为空?

请先检查套餐是否包含该特效；如果套餐包满足，请检查 /asset/下的资源文件是否包含 MotionRes/2dMotionRes 等目录及文件。

iOS

最近更新时间：2023-07-21 15:51:51

TUICallEngine 中提供了 getTRTCCloudInstance() 接口，可以通过该接口调用 TRTC 的高级特性 setLocalVideoProcessDelegete:pixelFormat:bufferType: 实现接入第三方美颜。
本文将介绍如何在 TUICallKit 中接入 腾讯特效 SDK，其他第三方美颜接入方法类似，请结合第三方 SDK 文档进行接入。

导入组件

您可以使用 CocoaPods 导入组件，具体如下：
1. 在您的 `Podfile` 文件中添加以下依赖。

Objective-C

```
pod 'XMagic'
```

2. 执行以下命令，安装组件。

```
pod install
```

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

```
pod repo update
```

3. 添加美颜资源到实际项目工程中：
   3.1 下载并解压对应套餐的 SDK 和美颜资源，将 resources 文件夹下的除 LightCore.bundle、Light3DPlugin.bundle、LightBodyPlugin.bundle、LightHandPlugin.bundle、LightSegmentPlugin.bundle、audio2exp.bundle 以外的其它 bundle 资源添加到实际工程中。
   3.2 在 Build Settings 中的 Other Linker Flags 添加 `-ObjC`。
4. 将 `Bundle Identifier` 修改成与申请的测试授权一致。

授权

1. 申请授权，得到 LicenseURL 和 LicenseKEY，请参见 License 指引。

> ⚠ **注意：**
> 正常情况下，只要 App 成功联网一次，就能完成鉴权流程，因此您不需要把 License 文件放到工程的工程目录里。但是如果您的 App 在从未联网的情况下也需要使用 SDK 相关功能，那么您可以把 License 文件下载下来放到工程目录，作为保底方案，此时 License 文件名必须是 `v_cube.license`。

2. 在相关业务模块的初始化代码中设置 URL 和 KEY，触发 license 下载，避免在使用前才临时去下载。也可以在 AppDelegate 的 didFinishLaunchingWithOptions 方法里触发下载。其中，LicenseURL 和 LicenseKey 是控制台绑定 License 时生成的授权信息。

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authresult, NSString *
_Nonnull errorMsg) {
    if (authresult == TELicenseCheckOk) {
        NSLog(@"鉴权成功");
    } else {
        NSLog(@"鉴权失败");
    }
}];
```

设置 SDK 素材资源路径

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

- (void)buildBeautySDK:(int)width and:(int)height texture:(unsigned)textureID {
    NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) lastObject];
    beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config.json"];
    NSFileManager *localFileManager=[[NSFileManager alloc] init];
    BOOL isDir = YES;
    NSDictionary * beautyConfigJson = @{};

    if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isDir) {
        NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfigPath
encoding:NSUTF8StringEncoding error:nil];
        NSError *jsonError;
        NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncoding];
        beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
                                                           options:NSJSONReadingMutableContainers
                                                           error:&jsonError];
    }

    NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                                 @"root_path":[[NSBundle mainBundle] bundlePath],
                                 @"tnn_"
                                 @"beauty_config":beautyConfigJson
    };

    // 初始化SDK：width 和 height 分别是 texture 的宽高
    self.xMagicKit = [[XMagic alloc] initWithRenderSize:CGSizeMake(width, height)
assetsDict:assetsDict];
}
```

配置美颜各种效果（详细美颜效果配置请参考 美颜参数说明 ）

```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString
*_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

进行渲染处理

TRTC SDK 设置第三方美颜的视频数据回调：设置该回调之后，TRTC SDK 会把采集到的视频帧通过您设置的 delegate 回调出来，用于第三方美颜组件进行二次处理。

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

[[[TUICallEngine createInstance] getTRTCCloudInstance] setLocalVideoProcessDelegete:self
pixelFormat:TRTCVideoPixelFormat_Texture_2D bufferType:TRTCVideoBufferType_Texture];
```

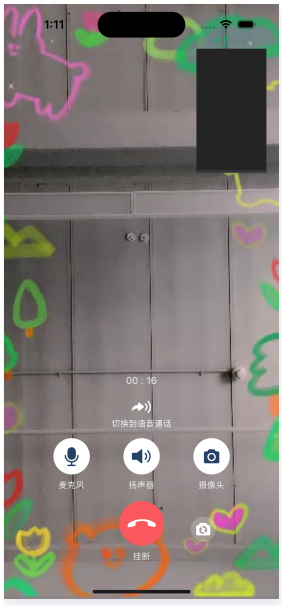
在视频帧回调接口 onProcessVideoFrame:(TRTCVideoFrame *_Nonnull)srcFrame dstFrame:(TRTCVideoFrame *_Nonnull)dstFrame
中，构造 YTProcessInput 传入到 SDK 内做渲染处理。

```
#pragma mark - TRTCVideoFrameDelegate

- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame *_Nonnull)srcFrame dstFrame:(TRTCVideoFrame
*_Nonnull)dstFrame {
    if (!self.xMagicKit) {
        [self buildBeautySDK:srcFrame.width and:srcFrame.height texture:srcFrame.textureId];
```

```
        self.heightF = srcFrame.height;
        self.widthF = srcFrame.width;
    }

    if(self.xMagicKit!=nil && (self.heightF!=srcFrame.height || self.widthF!=srcFrame.width)){
       self.heightF = srcFrame.height;
       self.widthF = srcFrame.width;
       [self.xMagicKit setRenderSize:CGSizeMake(srcFrame.width, srcFrame.height)];
    }

    YTProcessInput *input = [[YTProcessInput alloc] init];
    input.textureData = [[YTTextureData alloc] init];
    input.textureData.texture = srcFrame.textureId;
    input.textureData.textureWidth = srcFrame.width;
    input.textureData.textureHeight = srcFrame.height;
    input.dataType = kYTTextureData;
    YTProcessOutput *output = [self.xMagicKit process:input withOrigin:YtLightImageOriginTopLeft
withOrientation:YtLightCameraRotation0];
    dstFrame.textureId = output.textureData.texture;
    return 0;
}
```

集成效果

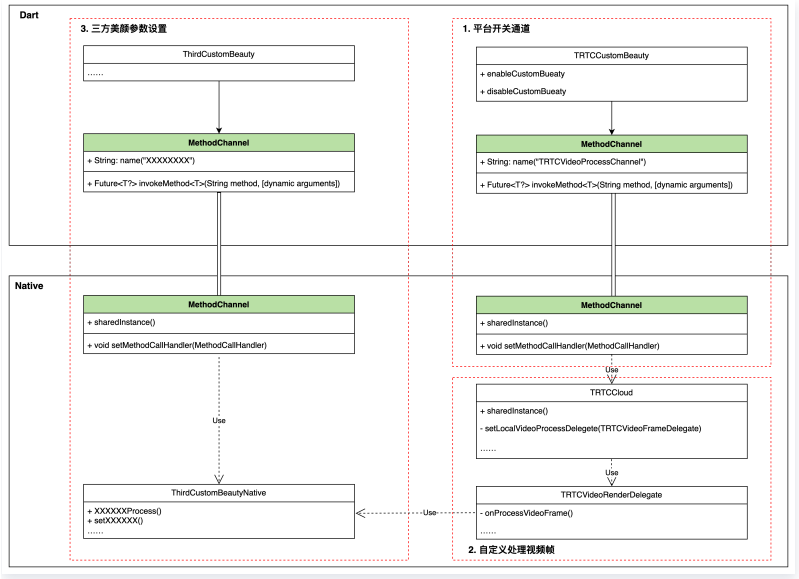
下图是 TUICallKit 含 UI 组件集成腾讯特效 SDK 的效果：



Flutter

最近更新时间：2025-01-17 11:06:22

本篇文档主要介绍在 TUICallKit 中接入美颜特效的方法。
在 Flutter 中完成自定义美颜处理，需要通过 TRTC 自定义视频渲染完成。由于 Flutter 不善于进行大量实时数据传输的特性，涉及 TRTC 的自定义视频渲染部分需要在 Native 部分完成。 具体的方案如下：



接入方案分为 3 步：
第一步：通过 MethodChannel 通道 开启/关闭 TRTC 自定义渲染逻辑；
第二步 ： 在 TRTC 自定义渲染处理逻辑 onProcessVideoFrame() 中使用 `美颜处理模块` 对原始视频帧进行处理；
第三步 : 客户的美颜处理模块也需要在 Dart 通过接口设置当前美颜的参数，客户可以通过 MethodChannel 的方式设置美颜参数。这部分客户可根据自己的需求和使用的美颜自行定制。

接入第三方美颜特效

步骤1：实现 Dart 层到 Native 的“开始/结束”美颜的控制接口

实现 Dart 层接口：

```
final channel = MethodChannel('TUICallKitCustomBeauty');

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

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

实现对应 Native 层接口：

java

```
public class MainActivity extends FlutterActivity {
    private static final String channelName = "TUICallKitCustomBeauty";

    private MethodChannel channel;

    @Override
```

```
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        super.configureFlutterEngine(flutterEngine);

        channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessenger(), channelName);
        channel.setMethodCallHandler(((call, result) -> {
            switch (call.method) {
                case "enableTUICallKitCustomBeauty":
                    enableTUICallKitCustomBeauty();
                    break;
                case "disableTUICallKitCustomBeauty":
                    disableTUICallKitCustomBeauty();
                    break;
                default:
                    break;
            }
            result.success("");
        }));
    }

    public void enableTUICallKitCustomBeauty() {

    }

    public void disableTUICallKitCustomBeauty() {

    }
}
```

swift

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
    var channel: FlutterMethodChannel?

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

        guard let controller = window?.rootViewController as? FlutterViewController else {
            fatalError("Invalid root view controller")
        }
        channel = FlutterMethodChannel(name: "TUICallKitCustomBeauty", binaryMessenger:
controller.binaryMessenger)
        channel?.setMethodCallHandler({ [weak self] call, result in
            guard let self = self else { return }
            switch (call.method) {
            case "enableTUICallKitCustomBeauty":
                self.enableTUICallKitCustomBeauty()
                break
            case "disableTUICallKitCustomBeauty":
                self.disableTUICallKitCustomBeauty()
                break
            default:
                break
            }
        })
        result(nil)
```

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

    func enableTUICallKitCustomBeauty() {

    }

    func disableTUICallKitCustomBeauty() {

    }
}
```

步骤2：在 Native 的 TRTC 自定义渲染逻辑中完成美颜处理

注意：

Android 在接入美颜过程中需要先依赖 LiteAVSDK_Professional，在Android工程的 app/build.gradle中添加一下依赖：

```
dependencies{
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
```

java

```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());

}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
}

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

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
        // 美颜处理逻辑
        mBeautyModel.process(trtcVideoFrame, trtcVideoFrame1);
        ……

        return 0;
    }

    @Override
    public void onGLContextCreated() {
    }

    @Override
    public void onGLContextDestory() {
    }
}
```

swift

```
let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pixelFormat:
._Texture_2D, bufferType: .texture)
}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Texture_2D, bufferType:
.texture)
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
    let bueutyModel = XXXXBeautyModel.shareIntance()
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame) -> UInt32 {
        // 美颜处理逻辑
        bueutyModel.onProcessVideoFrame(srcFrame, dstFrame)
        ……

        return 0
    }
}
```

步骤3：客户自定义第三方美颜参数控制逻辑

此部分您可以根据自己的需求和具体使用的美颜模块，参考 步骤1 种实现美颜参数的设置功能，具体实现根据具体使用情况决定。

接入腾讯美颜特效

腾讯美颜特效的接入方法也同样遵循上述方法，现在以腾讯美颜特效为例，详细介绍接入方法：

步骤1：美颜资源下载与集成

1. 根据您购买的套餐 下载 SDK 。
2. 添加文件到自己的工程中：

Android

1. 在 app 模块下找到 build.gradle 文件，添加您对应套餐的 maven 引用地址，例如您选择的是S1-04套餐，则添加如下：

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

各套餐对应的 maven 地址，请参见文档 。

2. 在 app 模块下找到 src/main/assets 文件夹，如果没有则创建，检查下载的 SDK 包中是否有 MotionRes 文件夹，如果有则将此文件夹拷贝到 `../src/main/assets` 目录下。
3. 在 app 模块下找到 AndroidManifest.xml 文件，在 application 表填内添加如下标签。

```
 <uses-native-library
        android:name="libOpenCL.so"
        android:required="true" />
        //此处的true 表示如果没有此库，则应用将无法正常运行。系统不允许在没有此库的设备上安装应用。
        //false表示应用可以使用此库（如果存在），但专门在没有此库的情况下运行（如果有必要）。系统允许安装应用，即
使不存在此库也是如此。如果您使用 "false"，则需要自行负责妥善处理库不存在的情况。
```

```
        //Android 官网介绍：https://developer.android.com/guide/topics/manifest/uses-native-library-element
```

添加后如下图：

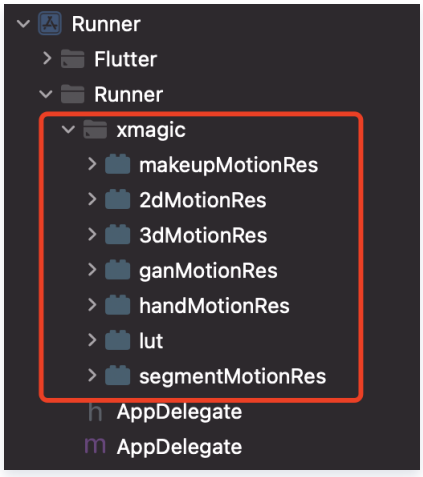


4. 混淆配置
   - 如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。
   - 如果您启用了这样的编译优化，那就要添加这些 keep 规则，防止 xmagic 的代码被裁掉：

```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
-keep class androidx.exifinterface.** { *;}
```

iOS

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



2. 在 demo 中把 demo/lib/producer 里面的4个类：BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid 和 BeautyPropertyProducerIOS 复制添加到自己的 Flutter 工程中，这4个类是用来配置美颜资源，把美颜类型展示在美颜面板中。

步骤2：引用 Flutter 版本 SDK

- **GitHub 引用**：在工程的 pubspec.yaml 文件中添加如下引用：

```
tencent_effect_flutter:
  git:
    url: https://github.com/TencentCloud/tencenteffect-sdk-flutter
```

- **本地引用**：从 tencent_effect_flutter 下载最新版本的 tencent_effect_flutter，然后把文件夹 `android`、`ios`、`lib` 和文件 `pubspec.yaml`、`tencent_effect_flutter.iml` 添加到工程目录下，然后在工程的 pubspec.yaml 文件中添加如下引用：（可参考demo）

```
tencent_effect_flutter:
    path: ../
```

tencent_effect_flutter 只是提供一个桥接，里面依赖的 XMagic 默认是最新版的，真正实现美颜是 XMagic。
如果要使用最新版本的美颜 SDK，您可以通过以下步骤进行 SDK 升级：

Android

在工程目录下执行命令：`flutter pub upgrade` 或者在 `subspec.yaml` 页面的右上角单击 **Pub upgrade**。

iOS

在工程目录下执行命令：flutter pub upgrade，然后在 iOS 目录下执行命令：`pod update`。

步骤3：实现 Dart 层到 Native 的“开始/结束”美颜的控制接口

此部分可以参见 接入第三方美颜特效/步骤1 ，此处不在赘述。

步骤4：在 Native 的 TRTC 自定义渲染逻辑中完成美颜处理

Android

Android 在接入美颜过程中需要先依赖 **LiteAVSDK_Professional，在Android工程的 app/build.gradle中添加一下依赖：**

```
dependencies{
    api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
```

```
void enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new VideoFrameListerer());

}

void disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance(getApplicationContext()).
                setLocalVideoProcessListener(TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
                        TRTC_VIDEO_BUFFER_TYPE_TEXTURE, null);
}
```

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

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame trtcVideoFrame,
                                   TRTCCloudDef.TRTCVideoFrame trtcVideoFrame1) {
       trtcVideoFrame1.texture.textureId = XmagicApiManager.getInstance()
        .process(trtcVideoFrame.texture.textureId, trtcVideoFrame.width, trtcVideoFrame.height);
        return 0;
    }

    @Override
    public void onGLContextCreated() {
      XmagicApiManager.getInstance().onCreateApi();
    }

    @Override
    public void onGLContextDestory() {
      XmagicApiManager.getInstance().onDestroy();
    }
}
```

iOS

```
import TXLiteAVSDK_Professional
import tencent_effect_flutter

let videoFrameListener: TRTCVideoFrameListener = TRTCVideoFrameListener()

func enableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(videoFrameListener, pixelFormat:
._Texture_2D, bufferType: .texture)
}

func disableTUICallKitCustomBeauty() {
    TRTCCloud.sharedInstance().setLocalVideoProcessDelegete(nil, pixelFormat: ._Texture_2D,
bufferType: .texture)
}

class TRTCVideoFrameListener: NSObject, TRTCVideoFrameDelegate {
    func onProcessVideoFrame(_ srcFrame: TRTCVideoFrame, dstFrame: TRTCVideoFrame) -> UInt32 {
        dstFrame.textureId =
GLuint(XmagicApiManager.shareSingleton().getTextureId(ConvertBeautyFrame.convertTRTCVideoFrame(trtc
VideoFrame: srcFrame)))
        return 0
    }
}


public class ConvertBeautyFrame: NSObject {
    public static func convertToTRTCPixelFormat(beautyPixelFormat: ITXCustomBeautyPixelFormat) ->
TRTCVideoPixelFormat {
        switch beautyPixelFormat {
```

```
        case .Unknown:
            return ._Unknown
        case .I420:
            return ._I420
        case .Texture2D:
            return ._Texture_2D
        case .BGRA:
            return ._32BGRA
        case .NV12:
            return ._NV12
        }
    }

    public static func convertTRTCVideoFrame(trtcVideoFrame: TRTCVideoFrame) ->
ITXCustomBeautyVideoFrame {
        let beautyVideoFrame = ITXCustomBeautyVideoFrame()
        beautyVideoFrame.data = trtcVideoFrame.data
        beautyVideoFrame.pixelBuffer = trtcVideoFrame.pixelBuffer
        beautyVideoFrame.width = UInt(trtcVideoFrame.width)
        beautyVideoFrame.height = UInt(trtcVideoFrame.height)
        beautyVideoFrame.textureId = trtcVideoFrame.textureId
        switch trtcVideoFrame.rotation {
        case ._0:
            beautyVideoFrame.rotation = .rotation_0
        case ._90:
            beautyVideoFrame.rotation = .rotation_90
        case ._180:
            beautyVideoFrame.rotation = .rotation_180
        case ._270:
            beautyVideoFrame.rotation = .rotation_270
        default:
            beautyVideoFrame.rotation = .rotation_0
        }
        switch trtcVideoFrame.pixelFormat {
        case ._Unknown:
            beautyVideoFrame.pixelFormat = .Unknown
        case ._I420:
            beautyVideoFrame.pixelFormat = .I420
        case ._Texture_2D:
            beautyVideoFrame.pixelFormat = .Texture2D
        case ._32BGRA:
            beautyVideoFrame.pixelFormat = .BGRA
        case ._NV12:
            beautyVideoFrame.pixelFormat = .NV12
        default:
            beautyVideoFrame.pixelFormat = .Unknown
        }
        beautyVideoFrame.bufferType = ITXCustomBeautyBufferType(rawValue:
trtcVideoFrame.bufferType.rawValue) ?? .Unknown
        beautyVideoFrame.timestamp = trtcVideoFrame.timestamp
        return beautyVideoFrame
    }
}
```

步骤5：开启美颜并设置美颜参数

完成上述配置后，可以通过 `enableTUICallKitCustomBeauty()/disableTUICallKitCustomBeauty()` 开启/关闭美颜。可以通过 腾讯特效美颜 Flutter 接口 设置美颜参数。

Web&小程序

最近更新时间：2025-06-12 15:13:42

`TUICallKitServer` 中提供了 `getTUICallEngineInstance()` 接口，您可以通过该接口调用 `TUICallEngine` 的基础美颜接口 `setBeautyLevel()` 。

Web

```
import { TRTCBeautyStyle } from "@tencentcloud/call-engine-js";

const beautyParams = {
    style: TRTCBeautyStyle.TRTCBeautyStyleSmooth,
    beautyLevel: 5,
    whitenessLevel: 5,
    ruddinessLevel: 5,
};

TUICallKitServer.getTUICallEngineInstance().setBeautyLevel(beautyParams);
```

参数如下表所示：

参数	类型	是否必填	含义
style	TRTCBeautyStyle	是	磨皮算法，有“光滑”(TRTCBeautyStyleSmooth=0)和“自然”(TRTCBeautyStyleNature=1)两种算法, 默认为 TRTCBeautyStyleNature
beautyLevel	Number	是	美颜级别，取值范围 [0 – 9]，0表示关闭，1 – 9值越大，效果越明显。
whitenessLevel	Number	是	美白级别，取值范围 [0 – 9]，0表示关闭，1 – 9值越大，效果越明显。
ruddinessLevel	Number	是	红润级别，取值范围 [0 – 9]，0表示关闭，1 – 9值越大，效果越明显。

小程序

```
import { TRTCBeautyStyle } from "@tencentcloud/call-engine-wx";

const beautyParams = {
    style: TRTCBeautyStyle.TRTCBeautyStyleSmooth,
    beautyLevel: 5,
    whitenessLevel: 5,
};

TUICallKitServer.getTUICallEngineInstance().setBeautyLevel(beautyParams);
```

参数如下表所示：

参数	类型	是否必填	含义
style	TRTCBeautyStyle	是	磨皮算法，有“光滑”(TRTCBeautyStyleSmooth=0)和“自然”(TRTCBeautyStyleNature=1)两种算法, 默认为

			TRTCBeautyStyleNature
beautyLevel	Number	是	美颜级别，取值范围 [0 – 9]，0表示关闭，1 – 9值越大，效果越明显。
whitenessLevel	Number	是	美白级别，取值范围 [0 – 9]，0表示关闭，1 – 9值越大，效果越明显。

自定义铃声
Android

最近更新时间：2025-06-26 10:31:51

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

设置应用铃声

设置应用铃声有两种方式：

1、替换默认的铃声资源

如果您通过源码依赖 TUICallKit 组件，您可以替换 `res\raw` 文件夹下的音频文件来达到替换铃声的目的：

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

2、通过 API 设置来电铃声

您也可以通过 setCallingBell 接口设置被叫端收到的来电铃声。

Kotlin

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

Java

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

设置来电静音模式

如果您的被叫端收到邀请，不需要响铃，您可以通过 enableMuteMode 设置静音模式。

Kotlin

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

Java

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

设置离线推送铃音

离线推送默认铃声是由系统决定的，可以在 手机设置 中，进入应用详情，查看或更改应用的通知铃声。
离线通知铃声仅支持以下厂商自定义：**华为、小米、FCM 和 APNs**，其他厂商暂不支持。离线推送铃声设置详见：

- 华为、小米、FCM（通知消息）自定义推送铃声 。
- FCM（数据消息）自定义来电铃声 。

iOS

最近更新时间：2025-06-30 14:37:32

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

设置应用铃声

设置应用铃声有两种方式：

1、替换默认的铃声资源

如果您通过源码依赖 TUICallKit 组件，您可以替换 `Resources\AudioFile` 文件夹下的音频文件来达到替换铃声的目的：

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

2、设置 API 设置来电铃声

您也可以通过 setCallingBell 接口设置被叫端收到的来电铃声。

Swift

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

Objective-C

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

设置来电静音模式

如果您的被叫端收到邀请，不需要响铃，您可以通过 enableMuteMode 设置静音模式。

Swift

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

Objective-C

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

设置离线推送铃音

VoIP 推送不支持自定义推送铃声。APNs 推送可以在调用 call 接口拨打电话时设置 params 的 offlinePushInfo 中的 iOSSound 字段， iOSSound 传语音文件名。

> **注意：**
> - 离线推送声音设置（仅对 iOS 生效），如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀名）设置给 iOSSound。
> - 铃声时长应小于30s。

Objective-C

```
[[TUICallKit createInstance] call:@"mike 的 id" params:[self getCallParams]
callMediaType:TUICallMediaTypeVideo];

- (TUICallParams *)getCallParams {
    TUIOfflinePushInfo *offlinePushInfo = [self createOfflinePushInfo];
    TUICallParams *callParams = [TUICallParams new];
    callParams.offlinePushInfo = offlinePushInfo;
    callParams.timeout = 30;
    return callParams;
}

+ (TUIOfflinePushInfo *)createOfflinePushInfo {
    TUIOfflinePushInfo *pushInfo = [TUIOfflinePushInfo new];
    pushInfo.title = @"";
    pushInfo.desc = TUICallingLocalize(@"TUICallKit.have.new.invitation");
    pushInfo.iOSPushType = TUICallIOSOfflinePushTypeAPNs;
    pushInfo.ignoreIOSBadge = NO;
    pushInfo.iOSSound = @"phone_ringing.mp3";
    pushInfo.AndroidSound = @"phone_ringing";
    // OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
    // OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
    pushInfo.AndroidOPPOChannelID = @"tuikit";
    // FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
    pushInfo.AndroidFCMChannelID = @"fcm_push_channel";
    // VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
    pushInfo.AndroidVIVOClassification = 1;
    // HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
    pushInfo.AndroidHuaWeiCategory = @"IM";
    return pushInfo;
}
```

Swift

```
let params = TUICallParams()
let pushInfo: TUIOfflinePushInfo = TUIOfflinePushInfo()
pushInfo.title = "TUICallKit"
pushInfo.desc = "TUICallKit.have.new.invitation"
pushInfo.iOSPushType = .apns
pushInfo.ignoreIOSBadge = false
pushInfo.iOSSound = "phone_ringing.mp3"
pushInfo.androidSound = "phone_ringing"
// OPPO必须设置ChannelID才可以收到推送消息，这个channelID需要和控制台一致
// OPPO must set a ChannelID to receive push messages. This channelID needs to be the same as the
console.
pushInfo.androidOPPOChannelID = "tuikit"
// FCM channel ID, you need change PrivateConstants.java and set "fcmPushChannelId"
pushInfo.androidFCMChannelID = "fcm_push_channel"
// VIVO message type: 0-push message, 1-System message(have a higher delivery rate)
pushInfo.androidVIVOClassification = 1
// HuaWei message type: https://developer.huawei.com/consumer/cn/doc/development/HMSCore-
Guides/message-classification-0000001149358835
pushInfo.androidHuaWeiCategory = "IM"
```

版权所有：腾讯云计算（北京）有限责任公司

```
params.userData = "User Data"
params.timeout = 30
params.offlinePushInfo = pushInfo
TUICallKit.createInstance().call(userId: "123456", callMediaType: .audio, params: params) {

} fail: {
    code, message in
}
```

Web&H5&小程序

最近更新时间：2024-09-26 14:54:01

本文介绍自定义铃声、静音来电铃声功能的使用。

自定义来电铃声

设置铃声接口

Web&H5 端	小程序端
• 仅限传入本地 MP3 格式的文件地址，确保该文件可以访问的。 • 如需恢复默认铃声 filePath 传空即可。 • 使用 ES6 import 方式引入铃声文件。	• 传入本地铃声文件应为相对当前小程序项目的绝对路径。 • uni-app 打包 Vue2 项目没有铃声，具体参见： Vue2 uni-app 打包微信小程序项目中，铃声没有声音?

说明：
Vue ⩾ v3.0.0 支持。

Web 端

```
import filePath from './static/ring.mp3';

try {
  await TUICallKitServer.setCallingBell(filePath?: string);
} catch (error: any) {
  alert(`[TUICallKit] setCallingBell API failed. Reason: ${error}`);
}
```

小程序端

```
try {
  await TUICallKitServer.setCallingBell('/static/ring.mp3'); // 相对当前小程序项目的绝对路径
} catch (error: any) {
  alert(`[TUICallKit] setCallingBell API failed. Reason: ${error}`);
}
```

替换音频文件

如果您通过源码集成 TUICallKit 组件，通过替换铃声文件来达到替换铃声的目的。

- uni-app 小程序：替换 `'TUICallKit/src/TUICallService/assets/'` 文件。
- 微信小程序：替换 `'TUICallKit/static/'` 文件。

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

静音来电铃声

- 开启/关闭来电铃声。
- 开启后，收到通话请求时，不会播放来电铃声。

说明：

> Vue&WeChat ⩾ v3.1.2 支持。

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

uni-app（客户端）

最近更新时间：2024-06-12 18:05:42

本文介绍自定义铃声、静音来电铃声功能的使用。

自定义来电铃声

设置自定义来电铃音，这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

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

uni.saveFile({
  tempFilePath: tempFilePath,
  success: (res) => {
    console.warn('保存文件成功 = ', JSON.stringify(res));
    musicFilePath =  res.savedFilePath;
    musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath);
    // Set ringtone
    TUICallKit.setCallingBell(musicFilePath, (res) => {
      if (res.code === 0) {
        console.log('setCallingBell success');
      } else {
        console.log(`setCallingBell failed, error message = ${res.msg}`);
      }
    });
  }
});
```

静音来电铃声

- 开启/关闭来电铃声。
- 开启后，收到通话请求时，不会播放来电铃声。

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

Flutter

最近更新时间：2024-09-02 17:35:11

本文介绍如何替换 TUICallKit 的来电铃声，来电铃声分为**应用铃声**和**离线推送铃声**。

设置应用铃声

设置应用铃声有两种方式：

1、替换默认的铃声资源

如果您通过源码依赖 TUICallKit 组件，您可以替换 **assets\audios** 文件夹下的音频文件来达到替换铃声的目的：

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

2、通过 API 设置来电铃声

您也可以通过 setCallingBell 接口设置被叫端收到的来电铃声。

```
TUICallKit.instance.setCallingBell('flie path');
```

设置静音模式

如果您的被叫端收到邀请，不需要响铃，您可以通过 enableMuteMode 设置静音模式。

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

设置离线推送铃音

1、iOS

VoIP 推送不支持自定义推送铃声。APNs 推送可以修改 `call` 和 `groupcall` 接口中的参数 `params` 中的 `TUIOfflinePushInfo.iOSSound` 设置 `iOS` 平台上的离线消息铃声。

2、Android

> **注意：**
> 接口支持华为、小米、FCM。

FCM 的推送铃声为设置的应用铃声。
华为、小米 和 APNS 推送铃声设置请在调用 Call 和 GroupCall 的时候设置 `TUIOfflinePushInfo.iOSSound` 的 `iOSSound` 及 `androidSound` 字段。具体详情请参见 自定义铃声 。

监听通话状态
Android&iOS&Flutter

最近更新时间：2025-06-30 21:57:42

回调说明

本文介绍 TUICallKit 组件通话状态回调的使用，您可以通过注册通话回调来监听通话的开始、结束，通话成员的加入、离开等事件。

> **注意：**
> 在 Android 平台设置 TUICallObserver 监听回调时，需要确保回调所在的类不会被销毁。例如，不建议在 LoginActivity 中添加监听，因为 LoginActivity 销毁时，回调也会随之清除；建议将监听设置在应用的 Application 类或应用的主界面中。

通话状态回调

用于感知整个通话生命周期中的关键状态变化。

Kotlin

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

private val observer: TUICallObserver = object : TUICallObserver() {
    override fun onCallReceived(callId: String?, callerId: String?, calleeIdList: MutableList<String>?,
mediaType: TUICallDefine.MediaType?, info: TUICallDefine.CallObserverExtraInfo?) {
    }

    override fun onCallBegin(roomId: TUICommonDefine.RoomId?, callMediaType: TUICallDefine.MediaType?,
callRole: TUICallDefine.Role?) {
    }

    override fun onCallEnd(roomId: TUICommonDefine.RoomId?, callMediaType: TUICallDefine.MediaType?,
callRole: TUICallDefine.Role?, totalTime: Long) {
    }

    override fun onCallNotConnected(callId: String?, mediaType: TUICallDefine.MediaType?, reason:
TUICallDefine.CallEndReason?, userId: String?, info: TUICallDefine.CallObserverExtraInfo?) {
    }
    ……
}

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

Swift

```
import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

func onCallReceived(_ callId: String, callerId: String, calleeIdList: [String], mediaType:
TUICallMediaType, info: TUICallObserverExtraInfo) {
}
```

```
func onCallBegin(callId: String, mediaType: TUICallMediaType, info: TUICallObserverExtraInfo) {
}

func onCallEnd(callId: String, mediaType: TUICallMediaType, reason: TUICallEndReason, userId: String,
totalTime: Float, info: TUICallObserverExtraInfo) {
}

func onCallNotConnected(callId: String, mediaType: TUICallMediaType, reason: TUICallEndReason, userId:
String, info: TUICallObserverExtraInfo) {
}
```

Dart

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

TUICallObserver observer = TUICallObserver(
    onCallReceived: (String callId, String callerId, List<String> calleeIdList, TUICallMediaType
mediaType, CallObserverExtraInfo info) {

    }, onCallBegin: (String callId, TUICallMediaType mediaType, CallObserverExtraInfo info) {

    }, onCallEnd: (String callId, TUICallMediaType mediaType, CallEndReason reason, String userId,
double totalTime, CallObserverExtraInfo info) {

    }, onCallNotConnected: (String callId, TUICallMediaType mediaType, CallEndReason reason, String
userId, CallObserverExtraInfo info) {

    }
    ……
)

void addObserver() {
    TUICallEngine.instance.addObserver(observer);
}
```

通话成员动态回调

追踪参与通话用户的个体行为或状态变更，主要用于 UI 实时展示成员动态（例如加入、离开、拒绝等）。

Kotlin

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

private val observer: TUICallObserver = object : TUICallObserver() {
    override fun onUserJoin(userId: String?) {
    }

    override fun onUserLeave(userId: String?) {
    }

    override fun onUserInviting(userId: String?) {
    }
```

版权所有：腾讯云计算（北京）有限责任公司

```
    override fun onUserReject(userId: String?) {
    }
    ……
}

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

Swift

```
import TUICallEngine

TUICallEngine.createInstance().addObserver(self)

func onUserJoin(userId: String) {
}

func onUserLeave(userId: String) {
}

func onUserInviting(userId: String) {
}

func onUserReject(userId: String) {
}
……
```

Dart

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

TUICallObserver observer = TUICallObserver(
    onUserJoin: (String userId) {

    }, onUserLeave: (String userId) {

    }, onUserInviting: (String userId) {

    }, onUserReject: (String userId) {

    },
    ……
)

void addObserver() {
    TUICallEngine.instance.addObserver(observer);
}
```

Web&H5

最近更新时间：2025-06-30 20:45:42

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详细参见 TUICallEvent ），可以参见如下代码：

```
import { TUICallEvent } from '@tencentcloud/call-engine-js';

let handleUserEnter = function(event) {
    console.log('TUICallEvent.USER_ENTER: ', event);
};

TUICallKitServer.getTUICallEngineInstance().on(TUICallEvent.USER_ENTER, handleUserEnter);
TUICallKitServer.getTUICallEngineInstance().off(TUICallEvent.USER_ENTER, handleUserEnter);
```

组件回调事件

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

- `beforeCalling`：通话开始前会执行。
- `afterCalling`：通话完成后执行。

> **说明：** WeChat 暂不支持。

React

```
function App() {
  const handleBeforeCalling = () => {
    console.log("[TUICallKit Demo] beforeCalling");
  };
  const handleAfterCalling = () => {
    console.log("[TUICallKit Demo] afterCalling");
  };
  return (
    <TUICallKit
      beforeCalling={handleBeforeCalling}
      afterCalling={handleAfterCalling} />
  )
}
```

Vue

```
<template>
  <TUICallKit
    :beforeCalling="handleBeforeCalling"
    :afterCalling="handleAfterCalling" />
</template>
<script setup>
function handleBeforeCalling() {
  console.log("[TUICallKit Demo] beforeCalling");
}
function handleAfterCalling() {
  console.log("[TUICallKit Demo] afterCalling");
```

```
}
</script>
```

uni-app（客户端）

最近更新时间：2024-04-30 09:51:21

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 TUICallEvent ），请参见如下代码：

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

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

uni-app (uts 插件)

最近更新时间：2025-06-27 17:35:32

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 TUICallEvent ），请参见如下代码：

```
import { CallEngine } from "@/uni_modules/TencentCloud-Call";

const callEngine = new CallEngine();

callEngine.on("onCallBegin", (res) => {
   console.log(`onCallBegin, data: ${JSON.stringify(res)}`);
 });
callEngine.on("onCallEnd", (res) => {
   console.log(`onCallEnd, data: ${JSON.stringify(res)}`);
 });
```

uniapp（小程序）

最近更新时间：2025-05-06 15:20:02

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 TUICallEvent），可以参见如下代码：

Vue3

```
<script setup>
// 替换成 TUICallKit 源码中的地址
import { TUICallKitServer } from "../../TUICallKit/src/index";
import { TUICallEvent } from "@tencentcloud/call-engine-wx";

// 绑定监听的函数
const bindEvent = () => {
  // setup 语法糖没有this，需要绑定回调函数的执行上下文
  const context = {
      handleNewInvitationReceived,
   };
   TUICallKitServer.getTUICallEngineInstance().on(
      TUICallEvent.INVITED,
      handleNewInvitationReceived,
      context
   );
};

// 回调函数
const handleNewInvitationReceived = (event) => {
  console.log("收到新的邀请", event);
};
</script>
```

Vue2

```
<script>
// 替换成 TUICallKit 源码中的地址
import { TUICallKitServer } from "../../TUICallKit/src/index";
import { EVENT } from "tuicall-engine-wx";

// 绑定监听的函数
bindEvent() {
  TUICallKitServer.getTUICallEngineInstance().on(
    EVENT.INVITED,
    this.handleNewInvitationReceived,
    this
  );
},

// 回调函数
handleNewInvitationReceived(event) {
```

```
    console.log("收到新的邀请", event);
},
</script>
```

微信小程序

最近更新时间：2025-05-06 15:20:02

本文介绍 TUICallKit 组件通话状态回调的使用。

通话状态监听

如果您的业务需要**监听通话的状态**，例如通话开始、结束等通话过程中的事件（详见 TUICallEvent），可以参见如下代码：

```
// 替换成 TUICallKit 源码中的地址
import { TUICallKitServer } from "../../TUICallKit/TUICallService/index";
import { TUICallEvent } from '@tencentcloud/call-engine-wx';

// 绑定监听的函数
bindEvent() {
    TUICallKitServer.getTUICallEngineInstance().on(
        TUICallEvent.INVITED,
        this.handleNewInvitationReceived,
        this
    );
},

// 回调函数
handleNewInvitationReceived(event) {
    console.log("收到新的邀请",event)
},
```

语言设置
Android

最近更新时间：2024-08-06 09:43:21

支持的语言

目前支持简体中文、英文、日文和阿拉伯语。

切换语言

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

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

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

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

添加新的语言

第一步：源码集成

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



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

setting.gradle.kts

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

settings.gradle

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

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

build.gradle.kts

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

build.gradle

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

第二步：新增语言包

以西班牙语为例：

1. 新增西班牙语文件。
   进入到 `TUICallKit` 源码文件目录下的 `src/main/res` 目录下，新增 `value-es/strings.xml` 文件 。
2. 将 `src/main/res/values-en/strings.xml` 中的内容复制到新增的 `src/main/res/values-es/strings.xml` 文件中。
3. 将 `src/main/res/values-es/strings.xml` 中的英文翻译为西班牙语。
4. 新增语言。

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

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

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

iOS

最近更新时间：2024-08-06 09:43:21

支持的语言

目前支持简体中文、英文、日文和阿拉伯语。

切换语言

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

```
……
import TUICore

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

添加新的语言

第一步：源码集成

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

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

end
```

3. 执行 `pod update` 命令，更新依赖。

第二步：新增语言包

以西班牙语为例：

1. 新增西班牙语文件。
   进入到 `TUICallKit` 源码文件目录下的 iOS/TUICallKit-Swift/Resources 目录下，新增 `es.lproj/Localized.strings` 文件 。
2. 将 `iOS/TUICallKit-Swift/Resources/en.lproj/Localized.strings` 中的内容复制到新增的 `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` 文件中。
3. 将 `iOS/TUICallKit-Swift/Resources/es.lproj/Localized.strings` 中的英文翻译为西班牙语。
4. 进入到 Application 工程的 Podfile 所在的目录文件夹下执行 `pod install` 命令，更新依赖。

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

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

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

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

Web&H5

最近更新时间：2025-06-30 20:45:42

支持的语言

目前支持简体中文、英文和日文。

切换语言

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

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

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

添加新的语言

如果您有其他语言需要支持，可以通过源码集成方式修改语言源文件实现。

第一步：源码集成

> ⚠ **注意：**
> 源码集成适用于 `Vue + Typescript` 项目且 `TUICallKit` 版本号 ≥ 3.2.2。

1. **下载源码**

Vue3

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

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

macOS + Vue3

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

Windows + Vue3

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

3. **修改引入路径**

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

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

4. **解决源码拷贝可能导致的报错**

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

ESLint 报错

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

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

TypeScript 报错

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

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

其他 TSConfig 问题请参见 TSConfig Reference。

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

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

第二步：新增语言包

以新增越南语为例：

1. 创建目标语言源文件。

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

```
export const vi = {    // 注意这里的导出变量
  'hangup': '挂断',
  'reject': '拒绝',
  'accept': '接受',
  'camera': '摄像头',
  'microphone': '麦克风',
  'speaker': '扬声器',
  'open camera': '打开摄像头',
  'close camera': '关闭摄像头',
  'open microphone': '打开麦克风',
  'close microphone': '关闭麦克风',
  'video-to-audio': '转语音通话',
  'virtual-background': '模糊背景',
  'other side reject call': '对方已拒绝',
  'reject call': '拒绝通话',
  'cancel': '取消通话',
  ...
};
```

2. 从 index.ts 导出

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

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

.....

export const languageData: languageDataType = {
  en,
  'zh-cn': zh,
  ja_JP,
  vi,        // 新增语言文件导出
};
```

3. 新增 LanguageType 枚举。

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

```
export enum LanguageType {
  EN = 'en',
  'ZH-CN' = 'zh-cn',
  JA_JP = 'ja_JP',
  VI = 'vi',    // 新增枚举类型
}
```

4. 切换语言

   在项目中通过调用 setLanguage 接口切换语言。

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

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

Flutter

最近更新时间：2024-08-06 09:43:21

支持的语言

目前支持简体中文、英文和日文，默认语言为英文 。

切换语言

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

添加新的语言

第一步：源码集成

1. 下载源码

   进入 `https://pub.dev/packages/tencent_calls_uikit` 下载最新 `TUICallKit` 源码。
2. 依赖本地源码

   在 `Application` 工程的 `pubspec.yaml` 文件中修改 `TUICallKit` 为本地依赖：

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

第二步：新增语言包

以西班牙语为例：

1. 新增西班牙语文件。

   进入到 `TUICallKit` 源码文件目录下的 `lib/src/i18n` 目录下，新增 `strings_es.i18n.json` 。
2. 将 `lib/src/i18n/strings.i18n.json` 中的内容复制到新增的 `lib/src/i18n/strings_es.i18n.json` 文件中。
3. 将 `lib/src/i18n/strings_es.i18n.json` 中的英文翻译为西班牙语。
4. 更新翻译包

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

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

5. 更新 `TUICallKit` 自适应设置语言方法。

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

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

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

uni-app（客户端）

最近更新时间：2024-08-06 09:49:51

支持的语言

目前支持**简体中文、英文、日文和阿拉伯语。**

切换语言

TUICallKit 语言与手机系统保持一致 。

添加新的语言

uni-app（客户端）暂不支持新增语言。

如果您有新增语言的需求，请通过 zhiliao 平台和技术、产品进行沟通。

体积优化
uni-app (小程序)

最近更新时间：2024-08-27 15:30:22

警告：
影响包体积的通常原因是项目中的静态资源过大，其他分包中的第三方依赖被打入主包中，具体可以参见 通用解决方案 。
特别注意：单独集成 TUIKit 和 TUIKit 与 TUICallKit 的融合场景，大约占用主包体积约800KB，需要预留对应的主包体积。

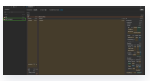
如何查看小程序包体积占比

在微信开发者工具中，顶部操作栏右侧 > 详情 > 代码依赖分析，可以查看编译后的包的情况与各文件大小。



包体积占比参照

注意：
这里是统一使用**分包集成**方式并经过下面的优化手段后的体积占用，可以通过对比下表，快速分析项目中的包体积占用，**并预留相对应的主包体积**。

场景	环境	需预留主包体积	分包体积	截图	说明
单独集成 TUICallKit	Vue2	**不占用主包体积**	1.48 MB		主包仅包含 uniapp 相关打包产物
	Vue3	**不占用主包体积**	1.37 MB		主包仅包含 uniapp 相关打包产物

单独集成 TUIKit	Vue2	约 800 KB	1.48 MB		主包中包含 Chat SDK 和 TUICore 约800KB
	Vue3	约 800 KB	764 KB		主包中包含 Chat SDK 和 TUICore 约800KB
TUIKit 与 TUICallKit 的融合场景	Vue2	约 800 KB	2.07 MB		主包中包含 Chat SDK 和 TUICore 约800KB
	Vue3	约 800 KB	1.22 MB		主包中包含 Chat SDK 和 TUICore 约800KB

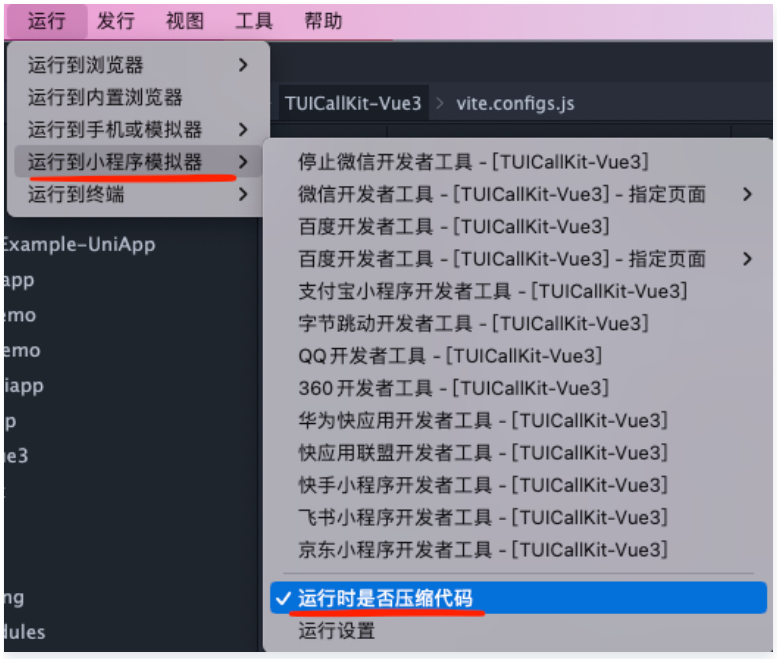
说明：
在单独集成 TUIKit和融合场景下，Chat SDK 和 TUICore 位于主包中，大约**占用主包体积约 800 KB，需要预留对应的主包体积。**

IDE 配置优化方案

Hbuilder 运行时相关配置

顶部操作栏 > 运行 > 运行到小程序模拟器 > 运行时是否压缩代码（勾选）

说明：
该步骤能有效减少第三方插件或组件库的代码体积，但也仅限代码，静态资源不会被压缩。

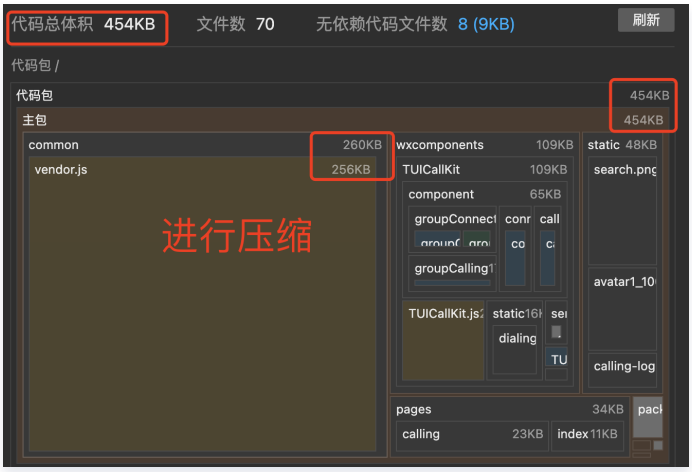


下图给出了是否勾选运行时是否压缩代码的效果对比（**851KB -> 454KB，降低 50%**）

- 不进行压缩：



- 进行压缩



微信开发者工具上传代码配置

> **说明：**
> 顶部操作栏 > 详情 > 上传代码时自动压缩样式文件 / 上传代码时自动压缩脚本 / 上传代码时自动压缩wxml文件 / 上传代码时过滤无依赖文件（勾选）



使用发行模式

在 HBuilderX 的工具栏中依次单击发行 > 小程序-微信。

> **说明：**
> 小程序发布前使用 HBuilderX 的发行功能进行打包。使用发行方式进行打包会删除代码中的 Source Map 等信息，打包体积最小。



然后单击**发行**，会在项目目录中的 `unpackage/dist/build/mp-weixin` 文件夹中生成微信小程序源码，再使用微信开发者工具打开该文件夹即可。

通用优化方案

删除 Debug 脚本

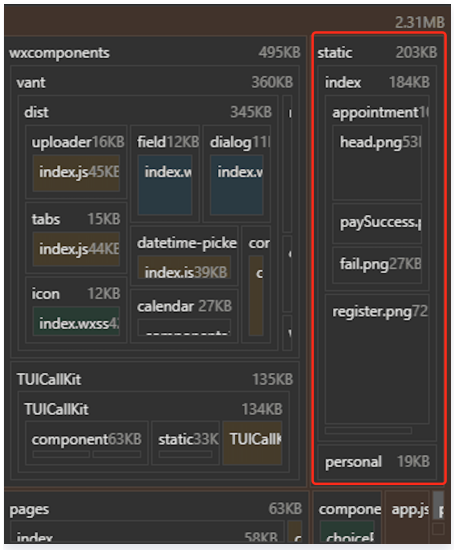
> **说明：**
> 出于体积和安全双重因素考虑，请在发布前删除项目目录下 `debug` 文件夹。可以节省约**150kb包体积**，生产环境推荐在服务端生成。

TUICallKit 的脚本文件存放于 `TUICallKit/debug` 目录。
TUIKit 的脚本文件存放于 `TUIKit/debug` 目录。

托管项目中的静态资源

> **警告：**
> 这一步往往是最影响包体积的一部分。

下面通过一个典型案例来说明：



从上图中的包体积分析中可以看出来， staic 中的静态资源达到了203KB，这些图片资源完全可以托管到服务器上，为主包节约203KB的大小。

公共样式统一加载

公共样式需要统一进行加载。不要每次进行引入，公共样式统一加载的好处不仅体现在减少代码重复，还有利于减少小程序包体积。

针对第三方 UI 库按需引入

使用 UI 库时只引入自己使用到的部分，按需引入只加载项目中实际使用的组件和样式，避免了整个 UI 库被打包，从而减小了最终产物的体积。

使用外部依赖优化包体积

> **警告：**
> **Vue3** 项目其他分包中的第三方依赖也会被打入主包中，导致主包体积膨胀，可以使用这种方式避免将分包中其他第三方依赖被打入主包中。

具体可以参见 如何使用外部依赖 。

单独集成 TUICallKit 体积优化

Vue2 项目

> **说明：**
> Vue2 使用 **webpack** 打包，独立分包中都会有单独的 vendor.js ，分包中的相关依赖不会打到主包里，不存在包体积问题。

TUICallKit 作为独立分包之后，**没有占有任何主包体积**，用户只需在 TUICallKit 分包中完成相关业务即可，包体积问题可以参考上述的**通用体积优化方案**。

Vue3 项目

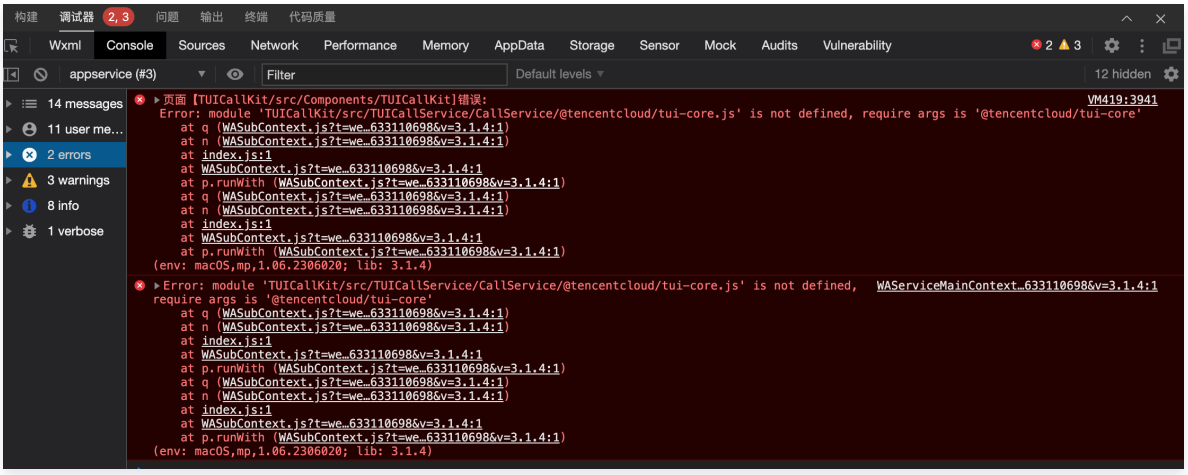
> **说明：**
> Vue3 使用 vite 打包，分包中的依赖会被打入主包中，造成**主包体积过大**问题。

1. 在根目录下新建 vite.config.js 文件

```
import { defineConfig } from "vite";
import uni from "@dcloudio/vite-plugin-uni";

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [uni()],
  optimizeDeps: {
    // 这里的 include 以 TUICallKit 为例，可以使用这种方式避免将分包中其他第三方依赖被打入主包中
    include: ["tuicall-engine-wx",'@tencentcloud/tui-core', '@tencentcloud/chat'],
  },
  build: {
    rollupOptions: {
      external: ["tuicall-engine-wx",'@tencentcloud/tui-core', '@tencentcloud/chat'],
    },
  },
});
```

2. 构建 npm

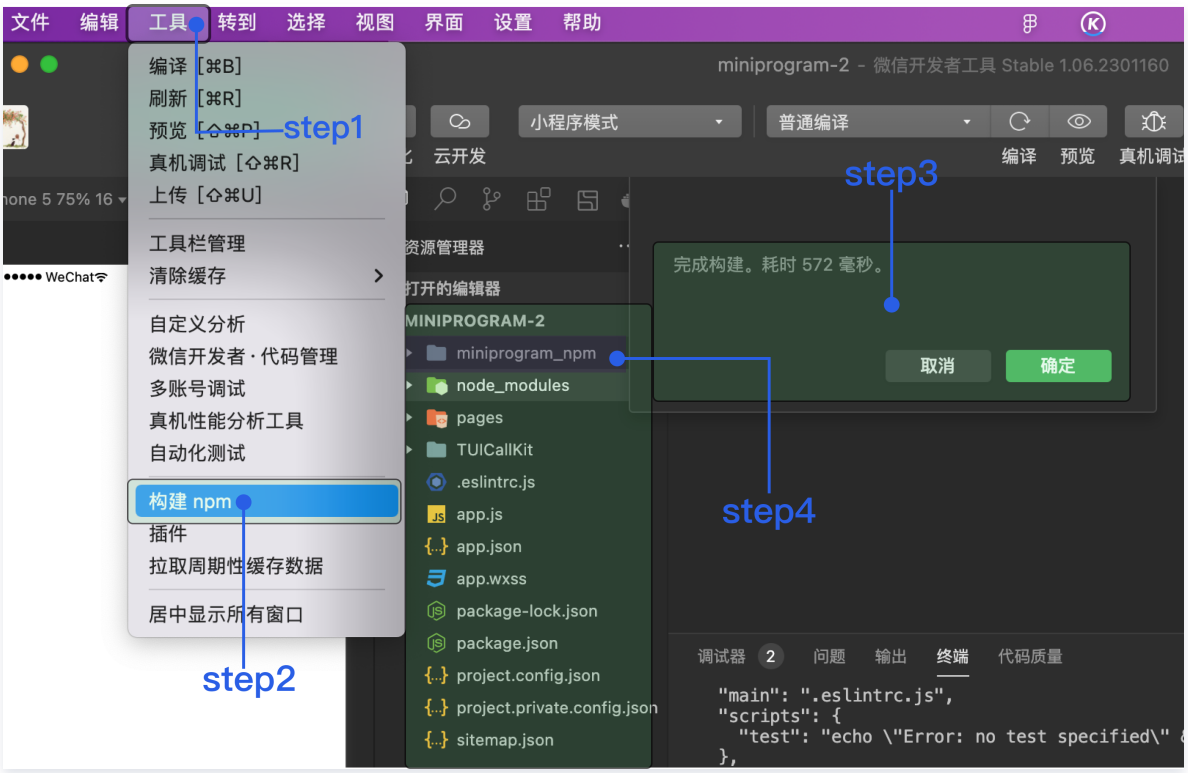
因为 HBuilder 运行到小程序后，项目中的 package.json、node_modules 都不存在了。需要微信开发者工具的终端里，重新安装依赖。



3. 进入分包页面，重新下载依赖。

```
cd TUICallKit && npm init -y && npm i @tencentcloud/call-uikit-wechat
```

4. 构建 npm。



单独集成 TUIKit 体积优化

> **说明：**
> 包体积问题可以参考上述的**通用体积优化方案**。

由于要保证和 uniapp 打包 App 保证一致性，登录逻辑需要位于主包，主包中包含 Chat SDK 和 TUICore，大约占用约 800 KB，需要预留相关的主包体积。

TUIKit 和 TUICallKit 融合场景体积优化

Vue2 项目

> **说明：**
> 包体积问题可以参考上述的**通用体积优化方案**。

由于 TUICallKit 和 TUIKit 处于同一个分包下，分包体积会超出2MB， 需要将 Chat SDK 和 TUICore 转移到主包中，大约占用主包体积约800KB，需要预留相关的主包体积。

Vue3 项目

使用外部依赖的方式优化包体积

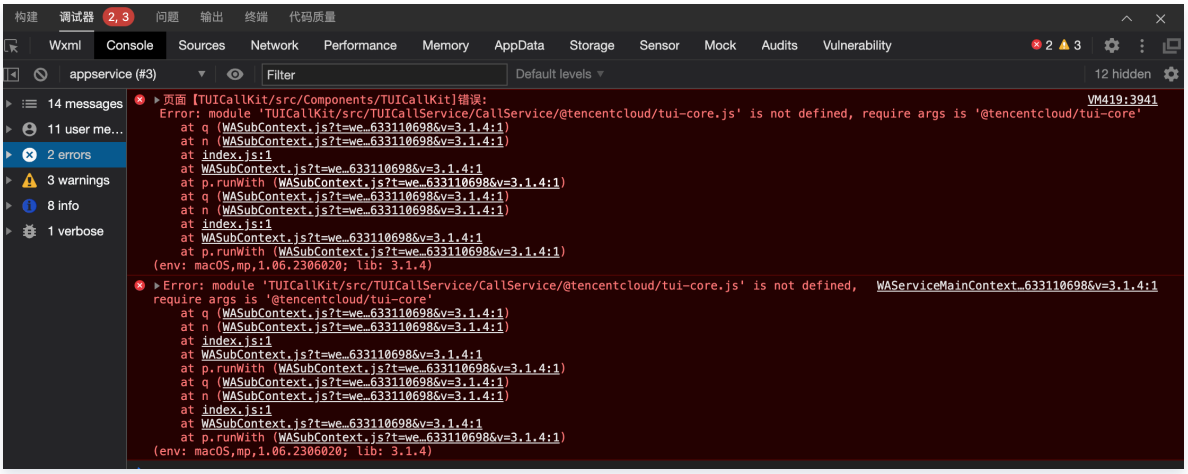
1. 在根目录下新建 vite.config.js 文件

```
import { defineConfig } from "vite";
import uni from "@dcloudio/vite-plugin-uni";

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [uni()],
  optimizeDeps: {
    // 这里的 include 以 TUICallKit 为例，可以使用这种方式避免将分包中其他第三方依赖被打入主包中
    include: ["tuicall-engine-wx",'@tencentcloud/tui-core', '@tencentcloud/chat'],
  },
```

```
  build: {
    rollupOptions: {
      external: ["tuicall-engine-wx",'@tencentcloud/tui-core', '@tencentcloud/chat'],
    },
  },
});
```

2. 构建 npm

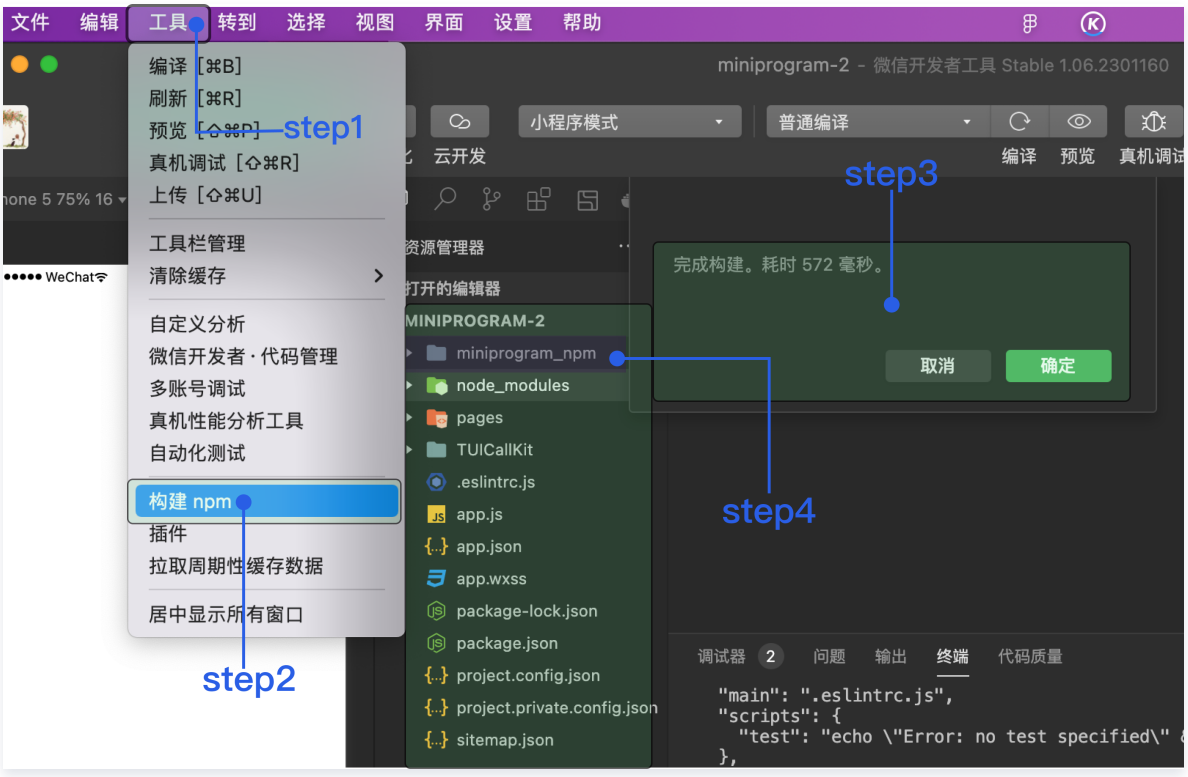
因为 HBuilder 运行到小程序后，项目中的 package.json、node_modules 都不存在了。需要微信开发者工具的终端里，重新安装依赖。



3. 进入分包页面，重新下载依赖。

```
cd TUIKit && npm init -y && npm i @tencentcloud/call-uikit-wechat
```

4. 构建 npm。



高级用法

> **警告:**

下面这一部分，通过移除 UIKit 的某些您可能未使用到的组件来降低包体积，适用于上述的方法都未解决包体积问题时，请谨慎抉择！

移除客服插件

说明：
注释以下五段代码，大约可以减小约 50 KB 的包体积。

1. TUIKit/plugins/plugin-components/index.ts。

```
import TUIChatEngine, { IMessageModel } from '@tencentcloud/chat-uikit-engine';
// import { isCustomerServicePluginMessage } from './message-customer/index';
import { JSONToObject } from '../../utils/type-check';
```

```
export function isPluginMessage(message: IMessageModel): boolean {
  return (
    message.type === TUIChatEngine.TYPES.MSG_CUSTOM
    && (isCallMessage(message)
    // || isCustomerServicePluginMessage(message as any)
    || isRoomCardMessage(message)
    || isRoomSignalingMessage(message))
  );
}
```

2. TUIKit/plugins/plugin-components/message-plugin.vue。

```
    <template #messageBubble>
      <MessageCallC2C
        v-if="pluginMessageType.pluginType === 'call'"
        :message="props.message"
        :signalingInfo="messageSignalingInfo"
        :customContent="messageCustomContent"
      />
      <!-- <MessageCustomerService
        v-if="pluginMessageType.pluginType === 'customer'"
        :message="props.message"
      /> -->
      <MessageRoom
        v-if="pluginMessageType.pluginType === 'room'"
        :message="props.message"
      />
    </template>
  </MessagePluginLayout>
</template>
```

移除群通话组件

说明：

- 单独集成 TUICallKit，位于 TUICallKit/src/Components/TUICallKit.vue。
- 融合场景下，位于 TUIKit/TUICallKit/src/Components/TUICallKit.vue。
- 注释以下代码，大约可以减小约 40 KB 的包体积。

```vue
<template>
  <div v-if="callInfoContextValue.callStatus !== CallStatus.IDLE" :style="[{ visibility: floatWindowContextValue.isFloatWindow ?
    <SingleCall v-if="!callInfoContextValue.isGroupCall" class="singCall"></SingleCall>
    <GroupCall v-else class="singCall"></GroupCall>
  </div>
</template>

<script lang="ts" setup>
import { ref, onMounted, reactive, onUnmounted, toRefs, watchEffect, provide } from '../adapter-vue';
import SingleCall from './components/SingleCall/SingleCall.vue';
import GroupCall from './components/GroupCall/GroupCall.vue';
import { TUIGlobal, TUIStore, StoreName, TUICallKitServer, NAME, CallStatus, CallMediaType } from '../TUICallService/index';
import {
  CallInfoContextKey,
  CallerUserInfoContextKey,
  UserInfoExcludeVolumeContextKey,
  FloatWindowContextKey,
  TCallerUserInfoValue,
  TFloatWindowContextValue,
  CustomUIConfigContextKey,
} from './context';
```

服务端 API（TUICallKit）
通话记录简介

最近更新时间：2025-02-26 17:34:12

通话记录是音视频通话 SDK（TUICallKit）的配套服务，提供 TUICallKit 产生的通话记录数据查看、查询等功能，具体包含 通话记录看板 、通话状态回调 和 REST API 三种服务内容，您可按需取用。

服务内容

服务类型	描述	版本要求
通话记录看板	提供 TUICallKit 的通话数据统计及通话记录可视化看板，登录控制台即可查看相关数据。	开通**群组通话版**的应用（SDKAppId）可使用
通话状态回调	提供实时通话结果回调	开通**群组通话版**的应用（SDKAppId）可使用
REST API	提供一种便捷获取通话记录的方法	**目前属于内测期**，内测期间开通**群组通话版**的应用（SDKAppId）可使用

使用条件

目前各平台仅在特定版本的 TUICallKit 可使用通话记录服务，详见下表：

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

注意：
使用 calls 接口发起的通话记录，目前正在加急开发中，预计在3月中旬上线。目前，如果需要查询通话记录，可以通过 通话记录 RestAPI 进行查询。

通话记录看板

最近更新时间：2025-06-17 12:12:12

通话记录看板是音视频通话能力（TUICallKit）的配套服务，提供 TUICallKit 的通话数据统计及通话记录可视化看板。服务包含通话次数、通话人数、最大并发通话数等统计数据，以及通话发起时间、通话参与人数、通话持续时间等单条通话记录信息。

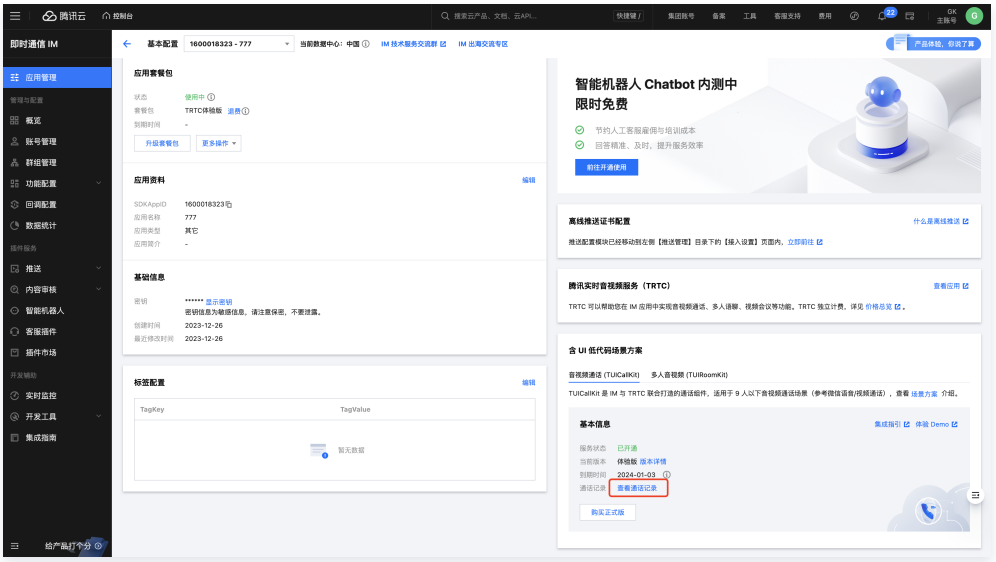
使用条件

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

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

使用指引

1. 登录 即时通信IM 控制台，单击目标应用卡片，进入应用的基础配置页面。
2. 在页面的右下角找到**含 UI 低代码场景方案 > 音视频通话（TUICallKit）**功能区。
3. 您可通过点击**查看通话记录**，进入通话记录页面。



4. 若您为首次使用通话记录，需阅读并确认服务开通界面说明，勾选**我已知晓并同意《腾讯云视立方 · 音视频通话 SDK 个人信息保护规则》**，随后单击 **立即开通**。



数据指标及功能

统计数据



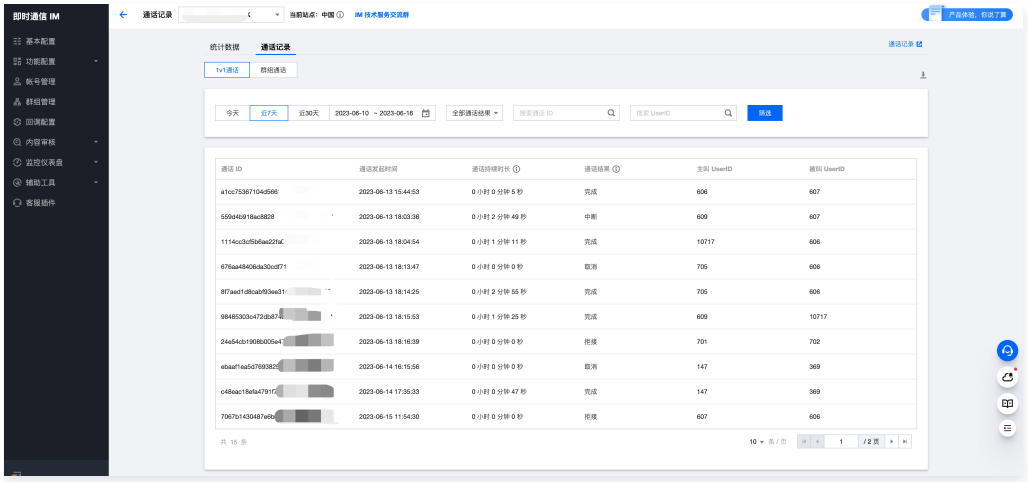
数据指标

指标	定义
通话次数	发起通话的总次数，不论是否接通均统计在内。
通话人数	成功加入通话的 UserID 去重后的总数。 如收到了通话邀请但未接听的 UserID 不会统计在内；如一个 UserID 在统计时间范围内参与多场通话，在通话人数中也只记录为1。
最大并发通话数	同时进行通话的最大通话数量。 在今日通话总览和本月通话总览中，最大并发通话数是1v1通话和群组通话2种方式的最大值，如统计时间范围内1v1通话的最大并发通话数是20，群组通话的最大并发通话数是10，那么对应数据将为20。

数据筛选条件

筛选条件	定义
时间	最多可查询近一年统计数据，最小时间粒度为天。
通话类型	分为1v1通话、群组通话和全部通话三个统计维度。 今日通话总览和本月通话总览均为全部通话统计结果。

通话记录



数据指标

通话类型	指标	定义
1v1通话	通话 ID	每一条通话记录的唯一标识。
	通话发起时间	发起通话的时刻，时间粒度到秒，如：2023-5-20 10:28:08。
	通话持续时长	通话接通后的持续时长。
	通话结果	1. 取消：主叫方在接通前取消通话。 2. 拒接：被叫方拒接。 3. 未接听：被叫方超时未接听。 4. 完成：通话接通并正常结束。 5. 忙线：通话忙线。 6. 中断：网络等原因导致通话中断。 未知原因：未知原因。如本地网络、进程异常中断等原因导致的数据上报失败。
	主叫 UserID	发起通话的 UserID。
	被叫 UserID	接听通话的 UserID。
群组通话	通话 ID	每一条通话记录的唯一标识。
	通话发起时间	发起通话的时刻，时间粒度到秒，如：2023-5-20 10:28:08。
	通话持续时长	通话参与人数大于1人的持续时长。
	同时通话人数峰值	同时在线通话人数的峰值。
	参与人数	成功加入过群组通话的人数，以 UserID 去重，点击对应数字可查看详情，包含： 1. UserID：成功加入通话的 UserID。 2. 通话时长：通话参与人数大于1人时，对应 UserID 的通话持续时长。 3. 平台：接入通话的平台，分为iOS、Android、Web 和微信小程序。

数据筛选条件

筛选条件	指标
通话类型	1v1通话和群组通话。
时间	最多可查询近90天通话记录，最小查询时间粒度为天。
通话结果	仅1v1通话支持，通话结果分类及定义参见 通话记录 > 数据指标 。

通话 ID	输入通话 ID 进行查询。
UserID	输入 UserID 进行查询。

版权所有：腾讯云计算（北京）有限责任公司

账号系统
账号管理

最近更新时间：2025-06-26 17:11:51

TUICallKit、TUIRoomKit、TUILiveKit 的系统和即时通信 IM 系统共用一套账号体系，账号即用户名(UserId)，分为普通账号和管理员账号，登录 即时通信 IM 控制台，在左侧导航栏选择**消息服务 Chat > 账号管理**，您可以根据实际业务需求管理账号。

> **说明：**
> TUICallKit、TUIRoomKit、TUILiveKit 的系统和即时通信 IM 系统共用一套账号体系。

新建账号

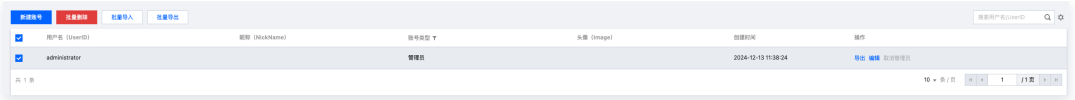
1. 在账号管理页面，单击**新建账号**。
2. 在弹出的新建账号对话框中，配置以下参数：



   - 账号类型：区分普通账号和管理员账号。“App管理员”是对 App 具有最高管理权限的角色，可调用 REST API 接口，进行用户管理，房间管理，房间成员管理等操作。每个应用最多支持配置 10 个管理员。
   - 用户名：需输入用户名（UserID），必填。
   - 用户昵称：可输入用户昵称，非必填。
   - 头像：可输入用户头像链接 URL，非必填。
3. 单击**确定**保存配置。
4. 账号新建完成后，在账号列表中可以查看用户名、昵称、账号类型、头像和创建时间。

删除账号

1. 在账号管理页面，在列表中选中需要删除的账号，单击**批量删除**。



2. 在弹出的确认对话框中，单击**确定**确认删除。删除后数据无法恢复，请谨慎操作。



编辑账号

1. 在账号管理页面，在列表中希望编辑的账号操作列，单击**编辑**。
2. 在弹出的编辑账号对话框中，配置以下参数：



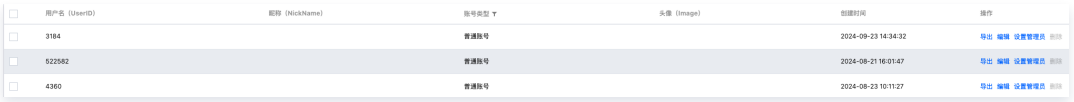
- 用户名：不可编辑。
- 昵称：可编辑用户昵称，非必填。
- 账号类型：不可编辑。
- 性别：可编辑性别，非必填。
- 生日：可编辑生日，非必填。
- 所在地：可编辑所在地，非必填。
- 个性签名：可编辑个性签名，非必填。
- 语言：可编辑语言，非必填。

- 头像：可编辑头像，非必填。
- 等级：可编辑等级，非必填。
- 角色：可编辑角色，非必填。

2.1 单击**确定**保存配置。

导出账号

1. 单独导出：在账号管理页面，在列表中希望导出的账号操作列，单击**导出**。



2. 批量导出：在账号管理页面，在列表中选中需要导出的账号，单击**批量导出**。

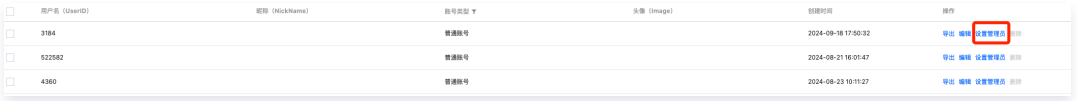


3. 在弹出的导出对话框中提示导出成功后，单击**下载**，即完成导出。



设置/取消管理员账号

1. 设置/取消管理员：在账号管理页面，在列表中找到希望设置或取消管理员的普通账号，单击**设置/取消管理员**。



> **注意：**
> administrator 账号不支持取消其管理员身份。

2. 在弹出的确认对话框中，单击**确定**确认即可。



登录鉴权

最近更新时间：2025-06-26 17:11:51

登录鉴权简介

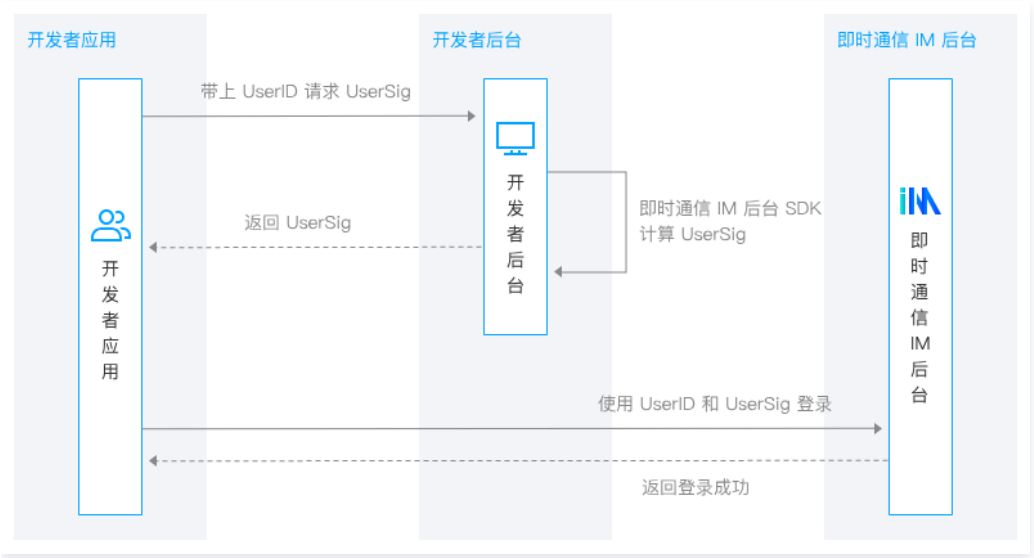
TUICallKit、TUIRoomKit、TUILiveKit 与即时通信 IM 共用同一套登录鉴权体系，即时通信 IM 的前身是 QQ 的即时通信系统，我们抽离 QQ 的通用模块，并将其整合成适合终端接入的 IM SDK 及后台服务。您可以把 IM SDK 理解为一个没有用户交互界面的 QQ，把 IM SDK 集成到您的 App 里，就相当于把一个 QQ 的内核集成在您的 App 内部。
QQ 可以用来收发消息，但前提是您必须先登录才能使用。登录 QQ 用的是 QQ 号和密码，登录 IM SDK 需要使用您指定的用户名（`UserID`）和密码（`UserSig`）。

- **UserID**：旧称为 Identifier，用户登录即时通信 IM 时使用的用户名，即您 App 里的用户 ID。
- 例如，App 里有一个用户，该用户的 ID 是27149 ，那么您可以用27149作为登录即时通信 IM 的 UserID。
- **UserSig**：用户登录即时通信 IM 时使用的密码，其本质是 App Server 用密钥对 UserID 等信息加密后的数据。具体生成方法请参见 生成 UserSig。

> **说明：**
> TUICallKit、TUIRoomKit、TUILiveKit 的系统和即时通信 IM 系统共用一套登录鉴权体系。

App 登录流程

推荐的 App 登录即时通信 IM 流程如下：



> **注意：**
> - 即时通信 IM 后台完全信赖 UserSig，为避免数据和业务受影响，请务必确保私钥的安全。
> - 即时通信 IM 后台 SDK 默认接口生成的 UserSig 有效期为180天，可以使用含有效期参数的接口自行设定有效期，开发者需要在 UserSig 过期前到开发者后台获取新的 UserSig。
> - 用于生成 UserSig 的即时通信 IM 后台 SDK 详细请参阅 生成 UserSig。

App 管理员

即时通信 IM 的部分服务需要管理员权限，例如调用 REST API 接口、解散群以及全员推送等。App 管理员是对 App 具有最高管理权限的角色，与普通账号相比，其区别如下：

- 读取权限更高。例如，获取 App 内部的所有群组、获取任意群组的任意资料。
- 操作权限更高。例如，给任意用户发消息、在任意群组中增删成员。

设置 App 管理员仅能通过登录控制台操作，具体操作请参见 配置账号管理员。

生成 UserSig

最近更新时间：2025-06-29 18:01:02

UserSig 是用户登录的密码，也是使用RestAPI的签名，其本质是对 UserID 等信息加密后得到的密文，本文将指导您如何生成 UserSig。以下视频将帮助您快速了解如何生成 UserSig：

> **说明：**
> TUICallKit、TUIRoomKit、TUILiveKit 的系统和即时通信 IM 系统共用同一套 Usersig 的生成方式。

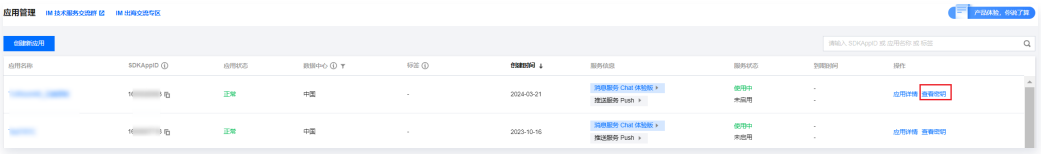
观看视频

获取密钥

1. 登录即时通信 IM 控制台 。

> **说明：**
> 如果您还没有应用，请先通过 开通服务 创建一个应用。

2. 单击**应用管理**，进入应用管理列表。
3. 在目标应用的操作列，单击**查看密钥**。



4. 在对话框中单击**显示密钥**。



5. 单击复制图标即可复制并储存密钥信息。



> **注意：**
> 请妥善保管密钥信息，谨防泄露。

客户端计算 UserSig

IM SDK 示例代码中提供的 `GenerateTestUserSig` 的开源模块可以帮忙您快速生成 UserSig。您只需设置 SDKAPPID（应用 SDKAppID）、EXPIRETIME（UserSig 过期时间）和 SECRETKEY（密钥信息）三个成员变量的取值，然后调用 genTestUserSig() 函数即可快速获取 UserSig。
为了简化实现过程，我们提供了下列语言或平台的 UserSig 计算源码，您可以直接下载并集成到您的客户端。

编程语言	所属平台	GenerateTestUserSig 源代码
Java	Android	GenerateTestUserSig.java
Objective-C	iOS	GenerateTestUserSig.h
Objective-C	Mac	GenerateTestUserSig.h
C++	Windows	GenerateTestUserSig.h
Javascript	Web	GenerateTestUserSig.js
Javascript	小程序	GenerateTestUserSig.js
Dart	Flutter	GenerateTestUserSig.dart

注意：
该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通 Demo 和功能调试**。
正确的 UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向 App 的接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig 。

服务端计算 UserSig

采用服务端计算 UserSig，可以最大限度地保障计算 UserSig 所用的密钥信息不被泄露。您只需将计算代码部署在您的服务器上，并提供面向 App 的服务端接口，在需要 UserSig 时由您的 App 向业务服务器发起请求获取动态 UserSig。
为了简化实现过程，我们提供了下列语言或平台的 UserSig 计算源码，您可以直接下载并集成到您的服务端。

语言版本	关键函数	下载链接
Java	HMAC-SHA256	genSig
GO	HMAC-SHA256	GenSig
PHP	HMAC-SHA256	genSig
Nodejs	HMAC-SHA256	genSig
Python	HMAC-SHA256	gen_sig
C#	HMAC-SHA256	GenSig
C++	HMAC-SHA256	gen_sig

UserSig 计算函数中主要包括 SDKAppID、UserID 以及 UserSig 有效期等关键参数，关键参数的详细介绍如下表所示：

说明：
下表中的字段名以 Java 语言的源码为例，其他语言略有差异，请以实际字段名为准。

字段名示例	参数说明
sdkappid	应用 SDKAppID，可在即时通信 IM 控制台 的应用卡片中获取。
userId	用户 ID，旧称为 Identifier。

expire	UserSig 的有效期，单位为秒。建议 UserSig 有效期最短不小于24小时，最长不超过50年。为了您的账号安全，建议将 UserSig 有效期设置为两个月。
userbuf	即时通信 IM 中均默认使用不带 UserBuf 的接口，即该参数默认填写为 `null` 。
key	密钥信息，可在即时通信 IM 控制台 的应用详情页面中获取。

老版本算法

为了简化签名计算难度，方便客户更快速地使用腾讯云服务，即时通信 IM 服务自2019.07.19开始启用新的签名算法，从之前的 ECDSA-SHA256 升级为 HMAC-SHA256。 2019.07.19以后创建的 SDKAppID 均会采用新的 HMAC-SHA256 算法。
如果您的 SDKAppID 是2019.07.19之前创建的，建议升级为 HMAC-SHA256 算法，升级过程不会影响现网业务。您也可以继续使用老版本的签名算法，ECDSA-SHA256 算法的源码下载链接如下：

语言版本	签名算法	下载链接
Java	ECDSA-SHA256	Github
GO	ECDSA-SHA256	Github
PHP	ECDSA-SHA256	Github
Nodejs	ECDSA-SHA256	Github
Python	ECDSA-SHA256	Github
C#	ECDSA-SHA256	Github
C++	ECDSA-SHA256	Github

版权所有：腾讯云计算（北京）有限责任公司

资料管理

最近更新时间：2025-06-26 17:11:51

TUICallKit、TUIRoomKit、TUILiveKit 的系统和即时通信 IM 系统共用同一套账号体系，资料属于账号体系里用户绑定的一些信息。

资料系统简介

即时通信 IM 开放了用户资料托管能力，提供资料相关的一整套完整解决方案。如果您想让每个用户都有自己的资料，并且轻松实现资料的设置和拉取功能，那么您可以选择即时通信 IM 的资料托管服务：

- 即时通信 IM 提供资料的存储能力，确保您的数据具备异地容灾、多地部署和自动扩容/缩容的能力，帮助您从服务器宕机、多拷贝主从复制和扩容缩容等复杂处理流程中得到完全地解放。
- 即时通信 IM 提供业界通用的业务处理流程，帮助您在用户资料的业务逻辑上彻底地解放。
- 即时通信 IM 提供专业的运营流程和运营团队，全年99.99%的稳定服务质量，帮助您为用户提供具有稳定口碑的服务。
- 即时通信 IM 提供简单易用的服务接口和快捷接入的帮助指引，全程为您提供星级服务。

使用即时通信 IM 的资料托管服务，可以得到以下能力：

- 标配资料字段的存储、读写能力。
- 自定义资料字段的存储、读写能力。

资料字段

资料是用来描述用户属性的一组数据，即时通信 IM 资料系统支持标配资料字段和自定义资料字段。资料字段有如下特性：

- 资料字段通过 Key-Value 来表示。
- Key 为 String 类型，其命名仅支持英文大小写字母、数字、下划线。
- Value 有以下几种类型：
  - a. uint32_t 类型的整数（自定义资料字段不支持）。
  - b. uint64_t 类型的整数（自定义资料字段不支持）。
  - c. string 类型的字符串（string 的长度不得超过500字节）。
  - d. bytes 类型的一段 buffer（buffer 的长度不得超过500字节）。

标配资料字段

目前即时通信 IM 支持的标配资料字段如下：

字段名称	类型	描述	更新时有 Push	备注
Tag_Profile_IM_Nick	String	昵称	有	长度不得超过500个字节
Tag_Profile_IM_Image	String	头像 URL	有	长度不得超过500个字节
Tag_Profile_IM_Level	uint32	等级	有	通常一个 UINT-8 数据即可保存一个等级信息您可以考虑拆分保存，从而实现多种角色的等级信息
Tag_Profile_IM_Role	uint32	角色	有	通常一个 UINT-8 数据即可保存一个角色信息，您可以考虑拆分保存，从而保存多种角色信息

自定义资料字段

自定义资料字段是各 App 根据各自业务需要而设置的用户数据。通过自定义资料，各 App 可以将一些额外数据附加到用户资料上，并通过现有接口进行读写操作。

自定义资料字段的申请

App 管理员可以通过即时通信 IM 控制台 > **应用配置** > **功能配置** 申请自定义资料字段，申请提交后，自定义资料字段将在10分钟内生效。
申请自定义资料字段时，需要为每一个自定义资料字段提交如下资料：

- 自定义资料字段的名称（Key）：详情可参见 自定义资料字段的命名规范 。
- 自定义资料字段的类型（Value）：详情可参见 资料字段 。

自定义资料字段的命名规范

自定义资料字段的命名规则如下：

- 自定义资料字段的名称包括前缀和关键字两部分。
- 自定义资料字段的前缀是：Tag_Profile_Custom。
- 关键字：必须是英文字母，且长度不得超过8字节，建议用一个英文单词或该英文单词的缩写。
- 示例：某 App 要申请的自定义字段的关键字是 Test，则自定义资料字段的名称是：Tag_Profile_Custom_Test。

版权所有：腾讯云计算（北京）有限责任公司

REST API
REST API 简介

最近更新时间：2025-06-30 21:41:41

REST API 是 CallKit SDK 的后台 HTTP 管理接口，其主要目的在于为开发者提供一套简单的管理入口，为了安全性，REST API 仅提供 HTTPS 接口。

前提条件

调用 REST API，您必须购买或领取音视频通话能力（TUICallKit）群组通话版的应用（SDKAppId），您也可以开通体验版免费试用，版本说明及开通指引参见 音视频通话能力版本 。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们 。

调用方法

请求 URL

REST API 的 URL 格式如下：

```
https://xxxxxx/$ver/$servicename/$command?
sdkappid=$SDKAppID&identifier=$identifier&usersig=$usersig&random=99999999&contenttype=json
```

其中各个参数的含义以及取值如下（参数名称及其取值均区分大小写）：

参数	含义	取值
xxxxxx	SDKAppID 所在国家/地区对应的专属域名	• 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
https	请求协议	请求协议为 HTTPS，请求方式为 POST
console.tim.qq.com	请求域名	固定为 `console.tim.qq.com`
ver	协议版本号	固定为 `v4`
servicename	内部服务名，不同的 servicename 对应不同的服务类型	示例：`v4/call_engine_http_srv/get_call_info`，其中 `call_engine_http_srv` 为 `servicename` 更多详情请参见 REST API 列表
command	命令字，与 servicename 组合用来标识具体的业务功能	示例：`v4/call_engine_http_srv/get_call_info`，其中 `get_call_info` 为 `command` 更多详情请参见 REST API 列表
sdkappid	App 在即时通信 IM 控制台获取的应用标识	在申请接入时获得
identifier	用户名，调用 REST API 时必须为 App 管理员账号	参见 App 管理员
usersig	用户名对应的密码	参见 生成 UserSig
random	标识当前请求的随机数参数	32位无符号整数随机数，取值范围0 - 4294967295
contenttype	请求格式	固定值为 `json`

> ⚠ 注意：
> - App 服务端在调用 REST API 时，identifier 必须为 App 管理员账号。
> - App 可以在每次调用 REST API 时都生成管理员账号的 UserSig，亦可生成一个固定的 UserSig 重复使用，但请特别注意 UserSig 的有效期。

HTTP 请求包体格式

- REST API 仅支持 POST 方法，其请求包体为 JSON 格式，具体的包体格式参见每个 API 的详细描述。
- 需要特别注意的是，POST 包体不能为空，即使某条协议包体中不需要携带任何信息，也需要携带一个空的 JSON 对象，即 `{}`。

HTTP 返回码

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

HTTP 应答包体格式

REST API 的应答包体也是 JSON 格式，其格式符合如下特征：

```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0,
    "RequestId": "Id-70e312f1de024af5a36714b7b71da224-O-Seq-63504"
    // REST API 其他应答内容
}
```

应答包体中必然包含 ActionStatus、ErrorInfo、ErrorCode、RequestId 这三个属性，其含义如下：

字段	类型	说明
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败，如果为 FAIL，ErrorInfo 带上失败原因
ErrorInfo	String	失败原因
ErrorCode	Integer	错误码，0为成功，其他为失败，可查询 错误码表 得到具体的原因
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId

调用示例

以下为通过 REST API 来 获取 App 中所有群组 示例。
HTTPS 请求：

```
POST /v4/group_open_http_svc/get_appid_group_list?
usersig=xxx&identifier=admin&sdkappid=88888888&random=99999999&contenttype=json HTTP/1.1
Host: console.tim.qq.com
Content-Length: 22
{
    "Limit": 2
}
```

HTTPS 应答：

```
HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Fri, 09 Oct 2015 02:59:55 GMT
Content-Length: 156
Connection: keep-alive
```

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Requested-With
Access-Control-Allow-Methods: POST

{
    "ActionStatus": "OK",
    "ErrorCode": 0,
    "GroupIdList": [
        {
            "GroupId": "@TGS#1YTTZEAEG"
        },
        {
            "GroupId": "@TGS#1KVTZEAEZ"
        }
    ],
    "TotalCount": 58530
}
```

REST API 公共错误码

错误码	含义说明
60002	HTTP 解析错误，请检查 HTTP 请求 URL 格式。
60003	HTTP 请求 JSON 解析错误，请检查 JSON 格式。
60004	请求 URL 或 JSON 包体中账号或签名错误。
60005	请求 URL 或 JSON 包体中账号或签名错误。
60006	SDKAppID 失效，请核对 SDKAppID 有效性。
60007	REST 接口调用频率超过限制，请降低请求频率。
60008	服务请求超时或 HTTP 请求格式错误，请检查并重试。
60009	请求资源错误，请检查请求 URL。
60010	请求需要 App 管理员权限。
60011	SDKAppID 请求频率超限，请降低请求频率。
60012	REST 接口需要带 SDKAppID，请检查请求 URL 中的 SDKAppID。
60013	HTTP 响应包 JSON 解析错误。
60014	置换账号超时。
60015	请求包体账号类型错误，请确认账号为字符串格式。
60016	SDKAppID 被禁用。
60017	请求被禁用。
60018	请求过于频繁，请稍后重试。
60019	请求过于频繁，请稍后重试。
60020	您的专业版套餐包已到期并停用，请登录 即时通信 IM 购买页面 重新购买套餐包。购买后，将在5分钟后生效。
60021	RestAPI 调用来源 IP 非法。

常见问题

REST API 请求有概率超时，收不到任何响应怎么办?

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

> **说明:**
> 由于 HTTPS 短连接建连耗时比较大，每次请求都有 TCP + tls 握手开销，所以建议 REST API 长连接接入。

使用标准 HTTP 库的场景：HTTP1.0 需要指定请求头部 Connection: keep-alive，HTTP1.1 默认支持长连接；基于 TCP 封装 HTTPS 请求的场景，可以复用 TCP 连接来收发请求。

REST API 列表

最近更新时间：2025-02-26 17:34:12

通话状态

功能说明	接口
获取实时通话状态	call_engine_http_srv/get_call_info

通话记录

功能说明	接口
通过 CallId 获取记录	call_record_http_srv/get_record_by_callid
通过条件获取记录	call_record_http_srv/get_record_by_filter

版权所有：腾讯云计算（北京）有限责任公司

通话记录
通过 CallId 获取记录

最近更新时间：2025-06-30 21:41:41

功能说明

App 管理员可以通过该接口获取指定 CallId 的通话记录信息。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/call_record_http_srv/get_record_by_callid?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： • 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
v4/call_record_http_srv/get_record_by_callid	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式

```
{
  "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b"
}
```

请求包字段说明

字段	类型	属性	说明

CallId	String	必填	通话 ID

应答包体示例

基础形式


```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-da9b3ee8bece466d951a9d93965c3d2c-0-Seq-324151555",
    "Response": {
        "CallRecord": {
            "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b",
            "Caller_Account": "user1",
            "MediaType": "Audio",
            "CallType": "MultiCall",
            "StartTime": 1739868165,
            "EndTime": 1740064224,
            "AcceptTime": 1739872743,
            "CallResult": "NormalEnd",
            "CalleeList_Account": [
                "user1",
                "user5",
                "user2"
            ],
            "RoomId": "roomid-1434",
            "RoomIdType": 2
        }
    }
}
```


应答包字段说明

字段	类型	说明
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId
CallRecord	Struct	通话记录信息
CallId	String	通话 ID
Caller_Account	String	主叫用户 ID
MediaType	String	媒体类型： • `Video` 视频通话 • `Audio` 音频通话
CallType	String	通话类型： • `SingleCall` 一对一通话 • `MultiCall` 多人通话
StartTime	Integer	通话发起的时间戳（秒级）

EndTime	Integer	通话结束的时间戳（秒级）
AcceptTime	Integer	通话接通的时间戳（秒级）
CallResult	Integer	通话结果： • `Cancel` 取消：主叫方在接通前取消通话 • `Reject` 拒接：被叫方拒接 • `NotAnswer` 未接听：被叫方超时未接听 • `NormalEnd` 完成：通话接通并正常结束 • `CallBusy` 忙线：通话忙线 • `Interrupt` 中断：网络等原因导致通话中断
CalleeList_Account	Array	通话成员列表
RoomId	String	TRTC 房间 ID
RoomIdType	Integer	RoomId 类型： • `1` 数字房间号 • `2` 字符串房间号

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
101001	服务器内部错误，请重试
101050	通话记录不存在

通过条件获取记录

最近更新时间：2025-06-30 21:41:41

功能说明

App 管理员可以通过该接口获取指定条件的7天内通话记录。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/call_record_http_srv/get_record_by_filter?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： • 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
v4/call_record_http_srv/get_record_by_filter	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式

查询7天内所有通话记录，默认查询第一页前10条。

```
{

}
```

指定条件形式

```
{
    "StartTime":0,          // 默认为当前时间减7天
```

```
    "EndTime":1740531683,  // 默认为当前时间
    "CallResult":"NormalEnd",
    "CallType":"SingleCall",
    "NumberPerPage":2,
    "Page":2
}
```

请求包字段说明

字段	类型	属性	说明
StartTime	Integer	选填	查找通话记录起始时间，默认为当前时间减7天。如果填写时间小于默认时间，后台会自动调整为默认时间。
EndTime	Integer	选填	查找通话记录结束时间（秒级），默认为当前时间
NumberPerPage	Integer	选填	每页查询数量，默认为：10
Page	Integer	选填	查询页数，不填默认为第一页

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-d3d6aa216dcc4cf4a4eee19c4942e740-O-Seq-2556133",
    "Response": {
        "TotalNum": 141,
        "Page":2,
        "CallRecordList": [
            {
                "CallId": "04c9a0ac-8e38-4a19-be45-349c5ce7911b",
                "Caller_Account": "147",
                "MediaType": "Audio",
                "CallType": "SingleCall",
                "StartTime": 1739960242,
                "EndTime": 1739960245,
                "AcceptTime": 1739960244,
                "CallResult": "NormalEnd",
                "CalleeList_Account": [
                    "147",
                    "369"
                ],
                "RoomId": "roomid-1434",
                "RoomIdType": 2
            },
            {
                "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
                "Caller_Account": "3423",
                "MediaType": "Video",
                "CallType": "SingleCall",
                "StartTime": 1739960500,
                "EndTime": 1739960507,
                "AcceptTime": 1739960503,
                "CallResult": "NormalEnd",
                "CalleeList_Account": [
```

```
                    "3423",
                    "3243"
                ],
                "RoomId": "roomid-1434",
                "RoomIdType": 2
            }
        ]
    }
}
```

应答包字段说明

字段	类型	说明
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId
TotalNum	Integer	此次查询的总数量
Page	Integer	当 Page > 0 时 说明还有数据，将 Page+1 可继续请求之后的数据
CallId	String	通话 ID
Caller_Account	String	主叫 用户 ID
MediaType	String	媒体类型： • `Video` 视频通话 • `Audio` 音频通话
CallType	String	通话类型： • `SingleCall` 一对一通话 • `MultiCall` 多人通话
StartTime	Integer	通话发起的时间戳（秒级）
EndTime	Integer	通话结束的时间戳（秒级）
AcceptTime	Integer	通话接通的时间戳（秒级）
CallResult	Integer	通话结果： • `Cancel` 取消：主叫方在接通前取消通话 • `Reject` 拒接：被叫方拒接 • `NotAnswer` 未接听：被叫方超时未接听 • `NormalEnd` 完成：通话接通并正常结束 • `CallBusy` 忙线：通话忙线 • `Interrupt` 中断：网络等原因导致通话中断
CalleeList_Account	Array	通话成员列表
RoomId	String	TRTC 房间 ID
RoomIdType	Integer	RoomId 类型： • `1` 数字房间号 • `2` 字符串房间号

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
101001	服务器内部错误，请重试
101050	通话记录不存在

获取实时通话状态

最近更新时间：2025-04-17 15:13:02

功能说明

App 管理员可以通过该接口获取实时通话状态。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://console.tim.qq.com/v4/call_engine_http_srv/get_call_info?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
v4/call_engine_http_srv/get_call_info	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为`json`

最高调用频率

200次/秒。

请求包示例

基础形式

```
{
  "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58"
}
```

请求包字段说明

字段	类型	属性	说明
CallId	String	必填	通话 ID

应答包体示例

基础形式

```
{
```

```
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-431454f25a44462d8155bdff4fed38cc-O-Seq-19029769",
    "Response": {
        "CallInfo": {
            "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
            "MediaType": "Audio",
            "ChatGroupId": "",
            "RoomId": "",
            "RoomIdType": 0
        },
        "UserList": [{
                "User_Account": "user1",
                "Status": "Calling"
            },
            {
                "User_Account": "user2",
                "Status": "Waiting"
            }
        ]
    }
}
```


应答包字段说明

字段	类型	说明
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId
CallId	String	通话 ID
MediaType	String	媒体类型： • `Video` 视频通话 • `Audio` 音频通话
ChatGroupId	String	IM 群 ID
RoomId	String	TRTC 房间 ID
RoomIdType	Integer	TRTC 房间 ID 类型： • `1` 数字房间号 • `2` 字符串房间号
UserList	Array	通话成员列表
User_Account	String	通话用户 ID
Status	String	通话状态： `Calling` 通话中 `Waiting` 等待接通

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
101001	服务器内部错误，请重试
101002	请参数非法，请根据错误描述检查请求是否正确
101004	通话不存在，或者曾经存在过，但是目前已经结束

第三方回调
第三方回调简介

最近更新时间：2025-04-17 15:13:02

为了方便您控制 App 的功能形态，CallKit 提供了回调能力。

功能说明

用户可通过 REST API 方式配置指定 Url 的回调，当执行的 CallbackCommand 在配置列表中，将触发回调。

注意事项

- 一个 sdkAppId 仅允许配置一个回调。
- 确保回调的 URL 是可以正常访问。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

回调命令列表

最近更新时间：2025-04-14 14:18:32

通话状态

回调类型	回调命令字
开始通话后回调	Call.CallbackAfterStartCall
通话结束后回调	Call.CallbackAfterEndCall
通话成员状态变更后回调	Call.CallbackAfterMemberChanged

版权所有：腾讯云计算（北京）有限责任公司

回调配置
查询回调配置

最近更新时间：2025-06-30 21:41:41

功能说明

App 管理员可以通过该接口查询回调配置。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/call_config/get_callback?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： • 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
v4/call_config/get_callback	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式

```
{
}
```

请求包字段说明

无。

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-1cc8828fd3d84795ac866ced43b15b5c-0-Seq-61309",
  "Response": {
      "Url": "http://www.example.com/callback",
      "CallbackCommandList": [
        "Call.CallbackAfterStartCall",
        "Call.CallbackAfterEndCall"
      ]
  }
}
```

应答包字段说明

字段	类型	说明
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId
Url	String	回调的地址，需要以 `http/https` 开头，建议使用更为安全的 `https`
CallbackCommandList	Array	需要触发回调的命令字，参见 回调命令字列表

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
100001	服务器内部错误，请重试
100002	参数非法，请根据错误描述检查请求是否正确。
100301	回调配置不存在，可以时间创建回调配置接口进行创建

设置回调配置

最近更新时间：2025-06-30 21:41:41

功能说明

App 管理员可以通过该接口设置回调配置。如非第一次设置配置，新配置会覆盖旧配置。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/call_config/set_callback?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： • 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
v4/call_config/set_callback	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式

```
{
  "Url":"http://www.example.com/callback",
  "CallbackCommandList":[
    "Call.CallbackAfterStartCall",
    "Call.CallbackAfterEndCall",
    "Call.CallbackAfterMemberChanged"
  ]
}
```

请求包字段说明

字段	类型	属性	说明
Url	String	必填	回调的地址，需要以 `http/https` 开头，建议使用更为安全的 `https`
CallbackCommandList	Array	必填	需要触发回调的命令字，参见 回调命令字列表

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720"
}
```

应答包字段说明

字段	类型	说明
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
ErrorInfo	String	错误信息
ErrorCode	Integer	错误码，0表示成功，非0表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
100001	服务器内部错误，请重试
100002	参数非法，请根据错误描述检查请求是否正确。

删除回调配置

最近更新时间：2025-06-30 21:41:41

功能说明

App 管理员可以通过该接口删除回调配置。

> **警告：**
> 如果您使用的是 `TUICallKit.call()` 或者 `TUICallKit.groupCall()` 接口发起通话的，请点击此处跳转至 旧版文档，如果您还有任何疑问，欢迎 联系我们。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/call_config/delete_callback?
sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： • 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com`
v4/call_config/delete_callback	请求接口
sdkappid	创建应用时即时通信 IM 控制台分配的 SDKAppID
identifier	必须为 App 管理员账号，更多详情请参见 App 管理员
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式

```
{
}
```

请求包字段说明

无。

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-1cc8828fd3d84795ac866ced43b15b5c-O-Seq-61309"
}
```

应答包字段说明

字段	类型	说明
ActionStatus	String	请求处理的结果，OK 表示处理成功，FAIL 表示失败
ErrorInfo	String	错误信息
ErrorCode	Integer	错误码，0表示成功，非0表示失败
RequestId	String	唯一请求 ID，每次请求都会返回，定位问题时需要提供该次请求的 RequestId

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。真正的错误码，错误信息是通过应答包体中的 ErrorCode、ErrorInfo 来表示的。
公共错误码（60000到79999）参见 错误码 文档。
本 API 私有错误码如下：

错误码	含义说明
100001	服务器内部错误，请重试
100002	参数非法，请根据错误描述检查请求是否正确。

版权所有：腾讯云计算（北京）有限责任公司

通话状态回调
开始通话后回调

最近更新时间：2025-03-31 09:03:02

功能说明

App 后台可以通过该回调实时查看通话的开始动态。

注意事项

- 要启用回调，必须通过 REST API 配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 设置回调配置 文档。
- 回调的方向是 Call 后台向 App 后台发起 HTTP POST 请求。
- App 后台在收到回调请求之后，务必校验请求 URL 中的参数 SDKAppID 是否是自己的 SDKAppID。

可能触发该回调的场景

App 用户通过客户端发起通话成功。

回调发生时机

通话发起之后。

接口说明

请求 URL 示例

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

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$ClientIP&OptPlatform=$Op
tPlatform
```

请求参数说明

参数	说明
https	请求协议为 HTTPS，请求方式为 POST
www.example.com	回调 URL
SdkAppid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
CallbackCommand	固定为 `Call.CallbackAfterStartCall`
contenttype	固定值为 `json`
ClientIP	客户端 IP，格式如：`127.0.0.1`
OptPlatform	客户端平台，取值参见 第三方回调简介：回调协议 中 OptPlatform 的参数含义

请求包示例

```
{
    "CallbackCommand":"Call.CallbackAfterStartCall",
    "CallInfo":{
        "CallId":"055662e1-bc8a-469c-a334-1126c8c17d58",
        "CallMediaType":"Audio",
```

```
        "ChatGroupId":"",
        "RoomId":"",
        "RoomIdType":1
    },
    "UserList":[
        {
            "User_Account":"user1",
            "Status":"Calling"
        },
        {
            "User_Account":"user2",
            "Status":"Waiting"
        }
    ],
    "EventTime":1740464128807
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
CallId	String	通话 ID
MediaType	String	媒体类型： • `Video` 视频通话 • `Audio` 音频通话
ChatGroupId	String	IM 群 ID
RoomId	String	TRTC 房间 ID
RoomIdType	Integer	TRTC 房间 ID 类型： • `1` 数字房间号 • `2` 字符串房间号
UserList	Array	通话成员列表
User_Account	String	通话用户 ID
Status	String	通话状态： `Calling` 通话中 `Waiting` 等待接通
EventTime	Integer	事件触发的毫秒级别时间戳

应答包示例

App 后台同步数据后，发送回调应答包。

```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 忽略回调结果
}
```

应答包字段说明

字段	类型	属性	说明
ActionStatus	String	必填	请求处理的结果，OK 表示处理成功，FAIL 表示失败
ErrorInfo	String	必填	错误信息
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果

参考

- 第三方回调简介
- REST API：获取实时通话状态

通话结束后回调

最近更新时间：2025-06-06 10:20:01

功能说明

App 后台可以通过该回调实时查看通话的结束动态。通话结束5秒后，可以通过查询通话记录接口查询通话详情。

注意事项

- 要启用回调，必须通过 REST API 配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 设置回调配置 文档。
- 回调的方向是 Call 后台向 App 后台发起 HTTP POST 请求。
- App 后台在收到回调请求之后，务必校验请求 URL 中的参数 SDKAppID 是否是自己的 SDKAppID。

可能触发该回调的场景

通话结束。

回调发生时机

通话成员数小于等于1人时，触发通话结束回收。

接口说明

请求 URL 示例

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

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$ClientIP&OptPlatform=$Op
tPlatform
```

请求参数说明

参数	说明
https	请求协议为 HTTPS，请求方式为 POST
www.example.com	回调 URL
SdkAppid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
CallbackCommand	固定为 `Call.CallbackAfterEndCall`
contenttype	固定值为 `json`
ClientIP	客户端 IP，格式如：`127.0.0.1`
OptPlatform	客户端平台，取值参见 第三方回调简介：回调协议 中 OptPlatform 的参数含义

请求包示例

```
{
    "CallbackCommand":"Call.CallbackAfterEndCall",
    "CallRecord": {
        "CallId": "055662e1-bc8a-469c-a334-1126c8c17d58",
        "Caller_Account": "10001",
        "MediaType": "Audio",
        "CallType": "SingleCall",
        "StartTime": 1741231146,
```

```
        "EndTime": 1741231296,
        "AcceptTime": 0,
        "CallResult": "NotAnswer",
        "CalleeList_Account": ["10001", "user2"],
        "RoomId": "roomid-1434",
        "RoomIdType": 2
    }
    "EventTime":1703589922780
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
CallRecord	Struct	通话记录信息
CallId	String	通话 ID
Caller_Account	String	主叫用户 ID
MediaType	String	媒体类型： • `Video` 视频通话 • `Audio` 音频通话
CallType	String	通话类型： • `SingleCall` 一对一通话 • `MultiCall` 多人通话
StartTime	Integer	通话发起的时间戳（秒级）
EndTime	Integer	通话结束的时间戳（秒级）
AcceptTime	Integer	通话接通的时间戳（秒级）
CallResult	Integer	通话结果： • `Cancel` 取消：主叫方在接通前取消通话 • `Reject` 拒接：被叫方拒接 • `NotAnswer` 未接听：被叫方超时未接听 • `NormalEnd` 完成：通话接通并正常结束 • `CallBusy` 忙线：通话忙线 • `Offline` 离线：服务端检测到心跳过期导致通话结束
CalleeList_Account	Array	通话成员列表
RoomId	String	TRTC 房间 ID
RoomIdType	Integer	RoomId 类型： • `1` 数字房间号 • `2` 字符串房间号
EventTime	Integer	事件触发的毫秒级别时间戳

应答包示例

App 后台同步数据后，发送回调应答包。

```
{
```

```
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 忽略回调结果
}
```

应答包字段说明

字段	类型	属性	说明
ActionStatus	String	必填	请求处理的结果，OK 表示处理成功，FAIL 表示失败
ErrorInfo	String	必填	错误信息
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果

参考

- 第三方回调简介
- REST API：通过 CallId 获取记录

通话成员状态变更后回调

最近更新时间：2025-04-14 14:18:32

功能说明

App 后台可以通过该回调实时查看通话成员状态变更。

注意事项

- 要启用回调，必须通过 REST API 配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 设置回调配置 文档。
- 回调的方向是 Call 后台向 App 后台发起 HTTP POST 请求。
- App 后台在收到回调请求之后，务必校验请求 URL 中的参数 SDKAppID 是否是自己的 SDKAppID。

可能触发该回调的场景

App 用户通过客户端 接通、挂断 等操作导致的成员状态变更都会触发该回调。

回调发生时机

通话成员状态变更之后。

接口说明

请求 URL 示例

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

```
https://www.example.com?
SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&contenttype=json&ClientIP=$ClientIP&OptPlatform=$Op
tPlatform
```

请求参数说明

参数	说明
https	请求协议为 HTTPS，请求方式为 POST
www.example.com	回调 URL
SdkAppid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
CallbackCommand	固定为 `Call.CallbackAfterMemberChanged`
contenttype	固定值为 `json`
ClientIP	客户端 IP，格式如：`127.0.0.1`
OptPlatform	客户端平台，取值参见 第三方回调简介：回调协议 中 OptPlatform 的参数含义

请求包示例

```
{
    "CallbackCommand":"Call.CallbackAfterMemberChanged",
    "CallId":"055662e1-bc8a-469c-a334-1126c8c17d58",
    "UserList":[
        {
            "User_Account":"user1",
            "Status":"Calling"
        },
```

```
        {
            "User_Account":"user2",
            "Status":"Calling"
        }
    ],
    "ChangedUserList":[
        {
            "User_Account":"user2",
            "ActionType":"Accept"   // Accept:接听; Reject:拒绝; NotAnswer:未接听; Hangup:挂断; Join:主动加入; Offline:离线(心跳过期); Invite:邀请; Cancel_Invite:取消邀请;
        }
    ]
    "EventTime":1740464128807
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
CallId	String	通话 ID
UserList	Array	通话成员列表
User_Account	String	通话用户 ID
Status	String	通话状态： • `Calling`：通话中 • `Waiting`：等待接通
ChangedUserList	Array	变成成员列表
User_Account	String	发生状态变更成员的用户 ID
ActionType	String	导致变更的操作： • `Accept`：接听 • `Reject`：拒绝 • `NotAnswer`：未接听 • `HangUp`：挂断 • `Join`：主动加入通话 • `Offline`：离线 • `Invite`：被邀请加入通话 • `Cancel_Invite`：主叫取消邀请
EventTime	Integer	事件触发的毫秒级别时间戳

应答包示例

App 后台同步数据后，发送回调应答包。

```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 忽略回调结果
}
```

应答包字段说明

字段	类型	属性	说明
ActionStatus	String	必填	请求处理的结果，OK 表示处理成功，FAIL 表示失败
ErrorInfo	String	必填	错误信息
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果

参考

- 第三方回调简介
- REST API：获取实时通话状态

旧版文档
服务端回调
通话状态回调

最近更新时间：2024-12-13 21:04:52

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

使用条件

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

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

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

注意事项

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

可能触发该回调的场景

App 用户通过客户端进行通话所产生的动作，例如挂断等。

回调的发生时机

通话结束后。

可能发生的回调结果

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

接口说明

请求URL

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

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

请求参数说明

参数	说明
http	请求协议为 HTTPS 或 HTTP，请求方式为 POST
www.example.com	回调 URL
sdkappid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
command	请参见：回调命令字列表
contenttype	固定值为 json
clientip	客户端 IP，格式如：127.0.0.1
optplatform	客户端平台，可能是 iOS、Android、Web、miniProgram

具体的回调内容在 HTTP 请求包体中，参见下文回调示例。

回调示例

回调请求示例：

```
POST /?sdkappid=8888888&command=call_end&contenttype=json&clientip=127.0.0.1&optplatform=iOS HTTP/1.1
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
    "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
    "CallResult": "normal_end",
    "EventTime": 170485688,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

请求包字段说明

字段	类型	说明
UserId	String	操作的用户 ID
RoomId	String	操作的房间 ID
TotalNum	Integer	参与通话人数
MediaType	String	媒体类型： • `audio` 音频通话

		• `video` 视频通话
CallType	String	通话类型： • `single` 1v1通话 • `group` 群组通话
CallId	String	通话唯一 ID
Role	String	角色： • `caller` 主叫 userId • `callee` 被叫 userId
CallResult	String	通话结果，仅在结束事件中填写，其余为空： • `cancel` 取消：主叫方在接通前取消通话 • `reject` 拒接：被叫方拒接 • `not_answer` 未接听：被叫方超时未接听 • `normal_end` 完成：通话接通并正常结束 • `call_busy` 忙线：通话忙线 • `interrupt` 中断：网络等原因导致通话中断
EventTime	Integer	操作发生的时间戳（秒级）
StartCallTs	Integer	通话发起的时间戳（秒级）只有在normal_end才会回调此值
AcceptTs	Integer	通话接听的时间戳（秒级）只有在normal_end才会回调此值
EndTs	Integer	通话结束的时间戳（秒级）只有在normal_end才会回调此值

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

回调应答示例

App 后台同步数据后，发送回调应答包。

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

应答包字段说明

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

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

重试机制

- 触发重试的时机

  ErrorCode = 0 视为回调成功，否则会触发重试。
  向您的 appServer 发 http 请求报错也会触发重试。

- 重试间隔

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

错误码

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

通话事件回调

最近更新时间：2024-12-13 21:04:52

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

使用条件

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

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

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

注意事项

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

可能触发该回调的场景

App 用户通过客户端进行通话所产生的动作，例如接听、挂断等。

回调的发生时机

接听、挂断等用户操作后。

可能发生的回调结果

请参见 回调命令字-通话事件回调 。

接口说明

请求URL

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

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

请求参数说明

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

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

具体的回调内容在 HTTP 请求包体中，参见下文回调示例。

回调示例

回调请求示例：

```
POST /?sdkappid=8888888&command=caller_start_call&contenttype=json&clientip=127.0.0.1&optplatform=iOS
HTTP/1.1
Host: www.example.com
Content-Length: 337
{
    "UserId": "Alice",
    "RoomId": "Alice's Room",
    "TotalNum": 2,
    "MediaType": "audio",
    "CallType": "single",
    "CallId": "aheahfo-eqwnknoihfsd-qweqf",
    "Role": "caller",
    "Event": "start_call",
    "CallResult": "",
    "EventTime": 1704695566,
    "StartCallTs": 1704856873,
    "AcceptTs": 1704856876,
    "EndTs": 1704856885
}
```

请求包字段说明

字段	类型	说明
UserId	String	操作的用户 ID
RoomId	String	操作的房间 ID
TotalNum	Integer	参与通话人数
MediaType	String	通话类型： • `audio` 音频通话 • `video` 视频通话
CallType	String	通话类型： • `single` 1v1通话 • `group` 群组通话
CallId	String	通话唯一 ID
Role	String	角色： • `caller` 主叫 userId • `callee` 被叫 userId

Event	String	通话事件： • `start_call` 主叫发起通话 • `call_accepted` 主叫接听通话 • `call_missed` 主叫未接听 • `call_rejected` 主叫拒绝通话 • `call_busy` 主叫通话忙线 • `cancel_call` 主叫取消通话 • `call_failed` 主叫发起通话失败 • `call_end` 主叫/被叫通话正常结束 • `call_interrupted` 主叫/被叫通话中断 • `receive_call` 被叫接收通话 • `accept_call` 被叫接听通话 • `not_answer_call` 被叫未应答 • `reject_call` 被叫拒接 • `ignore_call` 被叫忽略通话 • `call_canceled` 被叫取消通话 • `invite_user` 中途邀请用户 • `join_in_group_call` 中途加入通话
StartCallTs	Integer	通话发起的时间戳（秒级）只有在normal_end才会回调此值
AcceptTs	Integer	通话接听的时间戳（秒级）只有在normal_end才会回调此值
EndTs	Integer	通话结束的时间戳（秒级）只有在normal_end才会回调此值

说明：
CallResult 字段仅一对一通话可能显示所有类型，群聊仅有 `normal_end` 。
Event 字段中列举了所有可能，但部分事件概率出现较小。
TotalNum 通话总人数可能在事件同步时有一定的误差。

回调应答示例

App 后台同步数据后，发送回调应答包。

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

应答包字段说明

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

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

重试机制

- 触发重试的时机

ErrorCode = 0 视为回调成功，否则会触发重试。
向您的 appServer 发 http 请求报错也会触发重试。

- 重试间隔

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

错误码

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

回调配置
回调配置 API 列表

最近更新时间：2024-01-11 15:42:52

回调信息配置

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

回调命令字列表

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

创建回调配置

最近更新时间：2024-10-29 16:05:01

功能说明

管理员可通过该接口创建回调。

接口说明

> **注意：**
> 接口多次调用以最后一次的结果为准。

请求URL示例

```
https://xxxxxx/v1/callback/set?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国：`callkit.trtc.tencent-cloud.com`
v1/callback/set	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 IM App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

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

请求字段说明

字段	类型	属性	说明

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

应答包体示例

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

应答包字段说明

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

错误码说明

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

查询回调配置

最近更新时间：2024-10-29 16:05:01

功能说明

管理员可通过该接口查询已配置的回调。

接口说明

请求URL示列

```
https://xxxxxx/v1/callback/get?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国：`callkit.trtc.tencent-cloud.com`
v1/callback/get	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 IM App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

传空即可。

```
{

}
```

应答包体示例

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

```
    }
}
```

应答包字段说明

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

错误码说明

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

更新回调配置

最近更新时间：2024-10-29 16:05:01

功能说明

管理员可通过该接口更新回调。

接口说明

请求 URL 示例

```
https://xxxxxx/v1/callback/update?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国：`callkit.trtc.tencent-cloud.com`
v1/callback/update	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 IM App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

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

请求字段说明

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

应答包体示例

```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "5b0fa500064397cad3554506e27e18e1",
    "data": "https://www.example2.com/callback"
}
```

应答包字段说明

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

错误码说明

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

删除回调配置

最近更新时间：2024-10-29 16:05:01

功能说明

管理员可通过该接口删除回调。

接口说明

请求 URL 示例

```
https://xxxxxx/v1/callback/delete?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国：`callkit.trtc.tencent-cloud.com`
v1/callback/delete	请求接口
sdkappid	创建应用时控制台分配的 sdkappid
identifier	必须为 IM App 管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

10次/秒。

请求包示例

传空即可。

```
{

}
```

应答包体示例

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

应答包字段说明

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

错误码说明

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

REST API
通话记录
REST API 简介

最近更新时间：2024-12-13 21:04:52

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

使用条件

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

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

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

REST API 接口列表

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

调用方法

请求 URL

REST API 的 URL 格式如下：

```
https://xxxxxx/$version/$kind/$command?
sdkappid=$SDKAppID&identifier=$identifier&usersig=$usersig&random=99999999&contenttype=json
```

其中各个参数的含义以及取值如下（参数名称及其取值均区分大小写）：

参数	含义	取值
https	请求协议	请求协议为 HTTPS，请求方式为 POST
xxxxxx	专属域名	中国 `callkit.trtc.tencent-cloud.com`
version	协议版本号	固定为 `v1`
kind	管理分类	示列：

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

注意：
- 领取或购买套餐包后会在 IM 账号系统中创建 administrator 管理员账号，在请求中的 `identifier` 参数中请使用 administrator，如果您在 IM 账号管理 中取消或删除管理员，请正确指定管理员账号以及对应的 userSig。
- App 可以在每次调用 REST API 时都生成管理员账号的 UserSig，亦可生成一个固定的 UserSig 重复使用，但需特别注意 UserSig 的有效期。
- 在创建房间以及进入房间等操作时，系统会在 IM 系统中自动导入 IM 账号，敬请知悉。

HTTP 请求包体格式

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

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

HTTP 返回码

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

HTTP 应答包体格式

REST API 的应答包体也是 JSON 格式，其格式符合如下特征：

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

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

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

调用示例

以下为通过 REST API 获取指定 callId 的通话记录示例。

HTTPS 请求：

```
POST /records/get_record_by_callId?
usersig=xxx&identifier=admin&sdkappid=88888888&random=99999999&contenttype=json HTTP/2
Host: callkit.trtc.tencent-cloud.com
Content-Length: 36
{
    "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582"
}
```

HTTPS 应答：

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

{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "9f01db503f2006ad10c45f4f4609e38d",
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```

REST API 公共错误码

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

版权所有：腾讯云计算（北京）有限责任公司

未知错误码	系统内部错误，请 提交工单 联系技术人员

常见问题

REST API 请求有概率超时，收不到任何响应?

您可以从以下几个方面确认：

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

通过 callId 获取记录

最近更新时间：2024-10-29 16:05:01

功能说明

管理员可通过该接口获取指定 callId 的通话记录。

> **说明：**
> REST API 目前处于内测期，您可以查询创建时间在7天内的通话数据。

接口调用说明

请求URL示列

```
https://xxxxxx/v1/records/get_record_by_callId?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国站：`callkit.trtc.tencent-cloud.com`
v1/records/get_record_by_callId	请求接口
sdkappid	创建应用时控制台分配的 SDKAppID
identifier	使用 IM 的管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

20次/秒。

请求包示例

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

请求包字段说明

字段	类型	属性	说明
callId	String	必填	通话唯一 ID

应答包体示例

```
{
```

```
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "3fdc344c7a67461c6af3cfc3e77744b5",
    "data": {
        "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
        "sdkAppId": 88888888,
        "mediaType": "video",
        "roomId": "123456",
        "startCallTs": 1688705638,
        "acceptTs": 1688705641,
        "endTs": 1688705668,
        "caller": "alice",
        "totalUserNumber": 2,
        "callType": "single",
        "callResult": "normal_end",
        "callees": [
            "bob1",
            "bob2"
        ]
    }
}
```


应答包字段说明

字段	类型	说明
errorCode	Integer	错误码，0为成功
errorMessage	String	错误信息
requestId	String	请求的唯一 ID
callId	String	通话的唯一 ID
sdkAppId	Integer	您的 sdkAppId
mediaType	String	媒体类型 • `video` 视频通话 • `audio` 音频通话
roomId	String	通话的房间 ID
startCallTs	Integer	通话发起的时间戳（秒级）
acceptTs	Integer	通话接通的时间戳（秒级）
endTs	Integer	通话结束的时间戳（秒级）
caller	String	主叫 userId
totalUserNumber	Integer	参与通话的总人数
callType	String	通话类型： • `single` 一对一通话 • `group` 群组通话
callResult	String	通话结果： • `cancel` 取消：主叫方在接通前取消通话。 • `reject` 拒接：被叫方拒接。

		• `not_answer` 未接听：被叫方超时未接听。 • `normal_end` 完成：通话接通并正常结束。 • `call_busy` 忙线：通话忙线。 • `interrupt` 中断：网络等原因导致通话中断。
callees	Array	被叫且参与通话的 userId 列表

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

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。实际的错误码、错误信息是通过应答包体中的 errorCode、errorMessage 来表示的。
公共错误码（70000到79999）请参见 错误码 。

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

通过条件获取记录

最近更新时间：2024-10-29 16:05:01

管理员可通过该接口获取指定条件的通话记录。

> **说明：**
> REST API 目前处于内测期，您可以查询创建时间在七天内的通话数据。

接口说明

请求URL示列

```
https://xxxxxx/v1/records/get_records_by_filter?
sdkappid=88888888&identifier=administrator&usersig=xxx&random=99999999&contenttype=json
```

请求参数说明

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

参数	说明
xxxxxx	SDKAppID 所在国家/地区对应的专属域名： 中国：`callkit.trtc.tencent-cloud.com`
v1/records/get_records_by_filter	请求接口
sdkappid	创建应用时控制台分配的 SDKAppID
identifier	使用 IM 的管理员账号
usersig	App 管理员账号生成的签名，具体操作请参见 生成 UserSig
random	请输入随机的32位无符号整数，取值范围 0 - 4294967295
contenttype	请求格式固定值为 `json`

最高调用频率

200次/秒。

请求包示例

基础形式
创建房间

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

请求包字段说明

字段	类型	属性	说明

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

应答包字段示列


```
{
    "errorCode": 0,
    "errorMessage": "Success",
    "requestId": "8ba14f64a1d92d38705eec775e1f3deb",
    "data": {
        "totalNum": 3,
        "page": 2,
        "callRecordList": [
            {
                "callId": "2ae7d549-c441-4a9b-87c0-61810fe19581",
                "sdkAppId": 88888888,
                "mediaType": "video",
                "roomId": "456",
                "startCallTs": 1688709003,
                "acceptTs": 1688709003,
                "endTs": 1689150030,
                "caller": "123",
                "totalUserNumber": 0,
                "callType": "single",
                "callResult": "offline",
                "callees": [
                    "111",
                    "123"
                ]
            },
            {
                "callId": "2ae7d549-c441-4a9b-87c0-61810fe19582",
                "sdkAppId": 88888888,
                "mediaType": "video",
                "roomId": "456",
                "startCallTs": 1688709303,
                "acceptTs": 1688709303,
                "endTs": 1689150030,
                "caller": "123",
                "totalUserNumber": 0,
                "callType": "single",
                "callResult": "offline",
                "callees": [
                    "111",
                    "123"
                ]
            },
            {
                "callId": "2ae7d549-c441-4a9b-87c0-61810fe19583",
                "sdkAppId": 88888888,
                "mediaType": "video",
```

```
                "roomId": "456",
                "startCallTs": 1688709903,
                "acceptTs": 1688709903,
                "endTs": 1689150030,
                "caller": "123",
                "totalUserNumber": 0,
                "callType": "single",
                "callResult": "offline",
                "callees": [
                    "111",
                    "123"
                ]
            }
        ]
    }
}
```

应答包字段说明

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

单条通话记录说明

字段	类型	说明
callId	String	通话的唯一 ID
sdkAppId	Integer	您的 sdkAppId
mediaType	String	媒体类型 • `video` 视频通话 • `audio` 音频通话
roomId	String	通话的房间 ID
startCallTs	Integer	通话发起的时间戳（秒级）
acceptTs	Integer	通话接通的时间戳（秒级）
endTs	Integer	通话结束的时间戳（秒级）
caller	String	主叫 userId
totalUserNumber	Integer	参与通话的总人数
callType	String	通话类型 • `single` 一对一通话 • `group` 群组通话
callResult	String	通话结果：

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

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

错误码说明

除非发生网络错误（例如502错误），否则该接口的 HTTP 返回码均为200。实际的错误码、错误信息是通过应答包体中的 errorCode、errorMessage 来表示的。
公共错误码（70000到79999）请参见 错误码 。

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

结束通话

最近更新时间：2024-08-19 18:23:51

本文主要介绍音视频通话过程中，如何在服务端主动结束通话，以便于更好的管理通话时长，增强在医疗问诊、1V1社交等不同场景下的服务体验。

接口说明

接口请求域名： **trtc.tencentcloudapi.com** 。
接口说明：把房间所有用户从音视频房间移出，结束通话。
默认接口请求频率限制：**20次/秒**。

请求示例

```
https://trtc.tencentcloudapi.com/?Action=DismissRoom
&SdkAppId=1400000001
&RoomId=1234
&<Common request parameters>
```

下表仅列出调用本接口时涉及修改的参数及其说明，更多详情见：实时音视频 API 请求 。

参数	必选	类型	说明
trtc.tencentcloudapi.com	是	String	请求接口
Action	是	String	公共参数，本接口取值：DismissRoom。
Version	是	String	公共参数，本接口取值：2019-07-22。
Region	是	String	公共参数，详见产品支持的 地域列表，本接口仅支持其中的：ap-beijing、ap-guangzhou、ap-mumbai 。
SdkAppId	是	Integer	通话的 SDKAppId。 示例值：1400188366
RoomId	是	Integer	音视频房间号。
Common request parameters	是	/	详情见：公共参数

说明：
如需解散字符串房间号，Action 取值为：DismissRoomByStrRoomId， RoomId 类型为：String。

应答示例

```
{
    "Response": {
        "RequestId": "eac6b301-a322-493a-8e36-83b295459397"
    }
}
```

参数名称	类型	描述
RequestId	String	唯一请求 ID，由服务端生成，每次请求都会返回（若请求因其他原因未能抵达服务端，则该次请求不会获得 RequestId）。定位问题时需要提供该次请求的 RequestId。

调试接口

API Explorer： 点击调试

API Explorer 提供了在线调用、签名验证、SDK 代码生成和快速检索接口等能力。您可查看每次调用的请求内容和返回结果以及自动生成 SDK 调用示例。

错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见 公共错误码 。

错误码	描述
FailedOperation.RoomNotExist	房间不存在。
InternalError	内部错误。
InternalError.GetRoomCacheIpError	查询房间失败。
InvalidParameter.RoomId	RoomId 参数错误。
InvalidParameter.SdkAppId	SdkAppId 参数错误。
InvalidParameterValue.RoomId	RoomId 值错误。
MissingParameter.RoomId	缺少 RoomId 参数。
MissingParameter.SdkAppId	缺少 SdkAppId 参数。
UnauthorizedOperation.SdkAppId	没有操作 SdkAppId 的权限。

版权所有：腾讯云计算（北京）有限责任公司

客户端 API（TUICallKit）
Android
API 概览

最近更新时间：2025-06-26 10:03:12

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
createInstance	创建 TUICallKit 实例（单例模式）。
setSelfInfo	设置用户的头像、昵称。
calls	发起单人或多人通话。
join	主动加入通话。
setCallingBell	设置自定义来电铃音。
enableMuteMode	开启/关闭静音模式。
enableFloatWindow	开启/关闭悬浮窗功能。
enableIncomingBanner	开启/关闭来电横幅显示。
setScreenOrientation	设置通话界面显示方向。
enableVirtualBackground	设置模糊背景。

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
init	完成音视频通话基础能力的鉴权。
addObserver	增加事件回调。
removeObserver	移除回调接口。
calls	发起单人或多人通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
ignore	忽略通话。
inviteUser	在通话中，邀请其他人加入。
join	主动加入通话。
switchCallMediaType	切换通话媒体类型，例如视频通话切音频通话。

版权所有：腾讯云计算（北京）有限责任公司

startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
closeCamera	关闭摄像头。
switchCamera	切换前后摄像头。
openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）。
setSelfInfo	设置用户的昵称、头像。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
getTRTCCloudInstance	高级接口，用于接入第三方美颜等。
setBeautyLevel	设置美颜级别，支持关闭默认美颜。

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调。
onCallReceived	通话请求的回调。
onCallBegin	通话接通的回调。
onCallEnd	通话结束的回调。
onCallNotConnected	通话未接通的回调。
onUserReject	xxxx 用户拒绝通话的回调。
onUserNoResponse	xxxx 用户不响应的回调。
onUserLineBusy	xxxx 用户忙线的回调。
onUserInviting	xxxx 用户被追加邀请加入通话时的回调。
onUserJoin	xxxx 用户加入通话的回调。
onUserLeave	xxxx 用户离开通话的回调。
onUserVideoAvailable	xxxx 用户是否有视频流的回调。
onUserAudioAvailable	xxxx 用户是否有音频流的回调。
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调。
onUserNetworkQualityChanged	所有用户网络质量的反馈回调。
onKickedOffline	当前用户被踢下线。
onUserSigExpired	在线时票据过期。

关键类型定义

API	描述
TUICallDefine.MediaType	通话的媒体类型，枚举类型：视频通话、语音通话。
TUICallDefine.Role	通话的角色，枚举类型：主叫、被叫。
TUICallDefine.Status	通话的状态，枚举类型：空闲、待接听、接听中。
TUICallDefine.CallParams	通话扩展参数：离线推送消息显示内容。
TUICommonDefine.RoomId	音视频房间 ID，支持数字、字符串类型。
TUICommonDefine.Camera	摄像头参数，枚举类型：前置摄像头、后置摄像头。
TUICommonDefine.AudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒。
TUICommonDefine.NetworkQualityInfo	当前的网络质量信息。

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2025-06-25 10:09:12

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入TUICallKit 。

API 概览

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的头像、昵称
calls	发起单人或多人通话
join	主动加入通话
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
enableIncomingBanner	开启/关闭来电横幅显示
setScreenOrientation	设置通话界面显示方向
enableVirtualBackground	设置模糊背景

API 详情

createInstance

创建 TUICallKit 的单例。

Kotlin

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

Java

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

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

Kotlin

```
fun setSelfInfo(nickname: String?, avatar: String?, callback: TUICommonDefine.Callback?)
```

Java

```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
nickname	String	目标用户的昵称
avatar	String	目标用户的头像

calls

发起单人或多人通话。

kotlin

```
fun calls(
    userIdList: List<String?>?, mediaType: TUICallDefine.MediaType?,
    params: TUICallDefine.CallParams?, callback: TUICommonDefine.Callback?
)
```

Java

```
void calls(List<String> userIdList, TUICallDefine.MediaType mediaType,
           TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

join

主动加入通话。

Kotlin

```
fun join(callId: String?, callback: TUICommonDefine.Callback?)
```

Java

```
void join(String callId, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID

setCallingBell

设置自定义来电铃音。
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

- 铃声设置后与设备绑定，更换用户，铃声依旧会生效。
- 如需恢复默认铃声，`filePath` 传空即可。

Kotlin

```
fun setCallingBell(filePath: String?)
```

Java

```
void setCallingBell(String filePath);
```

enableMuteMode

开启/关闭静音模式。
开启后，收到通话请求，不会播放来电铃声。

Kotlin

```
fun enableMuteMode(enable: Boolean)
```

Java

```
void enableMuteMode(boolean enable);
```

enableFloatWindow

开启/关闭悬浮窗功能。
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Kotlin

```
fun enableFloatWindow(enable: Boolean)
```

Java

```
void enableFloatWindow(boolean enable);
```

enableIncomingBanner

开启/关闭来电横幅显示。
默认为 `false`，被叫端收到邀请后默认弹出全屏通话等待界面。开启后先展示一个横幅，然后根据需要拉起全屏通话界面。

Kotlin

```
fun enableIncomingBanner(enable: Boolean)
```

Java

```
void enableIncomingBanner(boolean enable);
```

setScreenOrientation

设置通话界面显示方向。
默认为竖屏，orientation：0-Portrait、1-LandScape、2-Auto。

Kotlin

```
fun setScreenOrientation(orientation: Int)
```

Java

```
void setScreenOrientation(int orientation);
```

enableVirtualBackground

设置模糊背景。
默认值为：false。

```
fun enableVirtualBackground(enable: Boolean)
```

废弃接口

call

拨打电话（1v1通话）。**注意：v2.9+版本废弃，建议使用 calls 接口。**

Kotlin

```
fun call(userId: String, mediaType: TUICallDefine.MediaType)
```

Java

```
 void call(String userId, TUICallDefine.MediaType mediaType)
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

call

拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。**注意：v2.9+版本废弃，建议使用 calls 接口。**

Kotlin

```
fun call(
    userId: String, mediaType: TUICallDefine.MediaType,
    params: CallParams?, callback: TUICommonDefine.Callback?
)
```

Java

```
void call(String userId, TUICallDefine.MediaType mediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

groupCall

发起群组通话。注意：v2.9+版本废弃，建议使用 calls 接口。

> ⚠ **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Kotlin

```
fun groupCall(groupId: String, userIdList: List<String?>?, mediaType: TUICallDefine.MediaType)
```

Java

```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType mediaType);
```

参数如下表所示：

参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List	目标用户的 userId 列表
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

groupCall

发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等。注意：v2.9+版本废弃，建议使用 calls 接口。

> ⚠ **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Kotlin

```
fun groupCall(
    groupId: String, userIdList: List<String?>?, mediaType: TUICallDefine.MediaType,
    params: CallParams?, callback: TUICommonDefine.Callback?
)
```

Java

```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType mediaType,
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
groupId	String	此次群组通话的群 ID
userIdList	List	目标用户的 userId 列表
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

joinInGroupCall

加入群组中已有的音视频通话。**注意：v2.9+版本废弃，建议使用 join 接口。**

> ⚠ **注意：**
> 加入群组中已有的音视频通话前，需要提前创建或加入 IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Kotlin

```
fun joinInGroupCall(roomId: RoomId?, groupId: String?, mediaType: TUICallDefine.MediaType?)
```

Java

```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId, TUICallDefine.MediaType
callMediaType);
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID
groupId	String	此次群组通话的群 ID
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话

TUICallEngine

最近更新时间：2025-06-26 09:55:52

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
init	完成音视频通话基础能力的鉴权。
addObserver	增加事件回调。
removeObserver	移除回调接口。
calls	发起单人或多人通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
ignore	忽略通话。
inviteUser	在通话中，邀请其他人加入。
join	主动加入通话。
switchCallMediaType	切换通话媒体类型，例如视频通话切音频通话。
startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
closeCamera	关闭摄像头。
switchCamera	切换前后摄像头。
openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）。
setSelfInfo	设置用户的昵称、头像。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐支持）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
getTRTCCloudInstance	高级接口，用于接入第三方美颜等。
setBeautyLevel	设置美颜级别，支持关闭默认美颜。

API 详情

createInstance

创建 TUICallEngine 的单例。

```
TUICallEngine createInstance(Context context);
```

destroyInstance

销毁 TUICallEngine 的单例。

```
void destroyInstance();
```

init

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。

```
void init(int sdkAppId, String userId, String userSig, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在**实时音视频控制台** > **应用管理** > **应用信息**中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig 。
callback	TUICommonDefine.Callback	初始化回调，`onSuccess` 表示初始化成功。

addObserver

添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。

```
void addObserver(TUICallObserver observer);
```

removeObserver

移除回调接口。

```
void removeObserver(TUICallObserver observer);
```

calls

发起单人或多人通话。

```
void calls(List<String> userIdList, TUICallDefine.MediaType mediaType,
           TUICallDefine.CallParams params, TUICommonDefine.Callback callback)
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表。

mediaType	TUICallDefine.MediaType	通话的媒体类型，例如视频通话、语音通话。
params	TUICallDefine.CallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

accept

接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。

```
void accept(TUICommonDefine.Callback callback);
```

reject

拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。

```
void reject(TUICommonDefine.Callback callback);
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
void hangup(TUICommonDefine.Callback callback);
```

inviteUser

邀请用户加入此次通话。
使用场景：一个通话中的用户主动邀请其他人时使用。

```
void inviteUser(List<String> userIdList, TUICallDefine.CallParams params,
                TUICommonDefine.ValueCallback callback);
```

参数如下表所示：

参数	类型	含义
userIdList	List	目标用户的 userId 列表。
params	TUICallDefine.CallParams	通话参数扩展字段，例如：通话邀请超时时间，离线推送自定义内容等。

> **注意：**
> 邀请他人加入当前通话，该场景下，自定义 RoomId 无效，SDK 会邀请他人加入当前邀请者所在的房间。

join

主动加入通话。

```
void join(String callId, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID。

startRemoteView

开始订阅远端用户的视频数据。

```
void startRemoteView(String userId, TUIVideoView videoView, TUICommonDefine.PlayCallback callback);
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId。
videoView	TUIVideoView	待渲染的视图。

stopRemoteView

停止订阅远端用户的视频数据。

```
void stopRemoteView(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId。

openCamera

开启摄像头。

```
void openCamera(TUICommonDefine.Camera camera, TUIVideoView videoView, TUICommonDefine.Callback
callback);
```

参数如下表所示：

参数	类型	含义
camera	TUICommonDefine.Camera	前置/后置 摄像头。
videoView	TUIVideoView	待渲染的视图。

closeCamera

关闭摄像头。

```
void closeCamera();
```

switchCamera

切换前后摄像头。

```
void switchCamera(TUICommonDefine.Camera camera);
```

参数如下表所示：

参数	类型	含义
camera	TUICommonDefine.Camera	前置/后置摄像头。

版权所有：腾讯云计算（北京）有限责任公司

openMicrophone

打开麦克风。

```
void openMicrophone(TUICommonDefine.Callback callback);
```

closeMicrophone

关闭麦克风。

```
void closeMicrophone();
```

selectAudioPlaybackDevice

选择音频播放设备。
目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。

```
void selectAudioPlaybackDevice(TUICommonDefine.AudioPlaybackDevice device);
```

参数如下表所示：

参数	类型	含义
device	TUICommonDefine.AudioPlaybackDevice	听筒/扬声器。

setSelfInfo

设置用户昵称、头像。
用户昵称不能超过500字节，用户头像必须是 URL 格式。

```
void setSelfInfo(String nickname, String avatar, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
nickname	String	用户昵称。
avatar	String	用户头像（格式为 URL）。

enableMultiDeviceAbility

开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐支持）。

```
void enableMultiDeviceAbility(boolean enable, TUICommonDefine.Callback callback);
```

setVideoRenderParams

设置用户视频画面的渲染模式。

```
void setVideoRenderParams(String userId, TUICommonDefine.VideoRenderParams params,
TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义

userId	String	目标用户的 userId。
params	TUICommonDefine.VideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式。

setVideoEncoderParams

设置视频编码的编码参数。
该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。

```
void setVideoEncoderParams(TUICommonDefine.VideoEncoderParams params, TUICommonDefine.Callback
callback);
```

参数如下表所示：

参数	类型	含义
params	TUICommonDefine.VideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高比。

getTRTCCloudInstance

高级接口。
用于接入第三方美颜等。

```
TRTCCloud getTRTCCloudInstance();
```

setBeautyLevel

设置美颜级别，支持关闭默认美颜。

```
void setBeautyLevel(float level, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
level	float	美颜级别，取值范围 0 - 9； 0 表示关闭，9 表示效果最明显。

废弃接口

call

拨打电话（1v1通话）。**注意：v2.9+版本废弃，建议使用 calls 接口。**

```
void call(String userId, TUICallDefine.MediaType mediaType,
          TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId。
mediaType	TUICallDefine.MediaType	通话的媒体类型，例如：视频通话、语音通话。
params	TUICallDefine.CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

版权所有：腾讯云计算（北京）有限责任公司

ignore

忽略当前通话，当您作为被叫收到 onCallReceived() 的回调时，可以调用该函数忽略来电，此时主叫会收到 onUserLineBusy 的回调。**注意：v2.9+版本废弃，针对 calls 发起的通话不生效，建议使用 reject 接口。**

```
void ignore(TUICommonDefine.Callback callback);
```

groupCall

发起群组通话。**注意：v2.9+版本废弃，建议使用 calls 接口。**

> ⚠ **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

```
void groupCall(String groupId, List<String> userIdList, TUICallDefine.MediaType mediaType,
               TUICallDefine.CallParams params, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
groupId	String	此次群组通话的群 ID。
userIdList	List	目标用户的 userId 列表。
mediaType	TUICallDefine.MediaType	通话的媒体类型，例如：视频通话、语音通话。
params	TUICallDefine.CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

joinInGroupCall

主动加入此次群组通话。**注意：v2.9+版本废弃，建议使用 join 接口。**
使用场景：群组内用户主动加入此次群组通话使用。

```
void joinInGroupCall(TUICommonDefine.RoomId roomId, String groupId,
                     TUICallDefine.MediaType mediaType, TUICommonDefine.Callback callback);
```

参数如下表所示：

参数	类型	含义
roomId	TUICommonDefine.RoomId	此次通话的音视频房间 ID。
groupId	String	此次群组通话的群 ID。
mediaType	TUICallDefine.MediaType	通话的媒体类型，例如视频通话、语音通话。

switchCallMediaType

切换视频通话到语音通话。

```
void switchCallMediaType(TUICallDefine.MediaType mediaType);
```

参数如下表所示：

参数	类型	含义
mediaType	TUICallDefine.MediaType	通话的媒体类型，例如视频通话、语音通话。

TUICallObserver

最近更新时间：2025-06-18 17:26:52

TUICallObserver API 简介

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallNotConnected	通话未接通的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserInviting	xxxx 用户被追加邀请加入通话时的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

回调事件详情

onError

错误回调。

> **说明：**
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
void onError(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码

message	String	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
void onCallReceived(String callId, String callerId, List<String> calleeIdList,
                    TUICallDefine.MediaType mediaType, TUICallDefine.CallObserverExtraInfo info);
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID
callerId	String	主叫 ID（邀请方）
calleeIdList	List<String>	被叫 ID 列表（被邀请方）
mediaType	TUICallDefine.MediaType	通话的媒体类型，比如视频通话、语音通话
info	TUICallDefine.CallObserverExtraInfo	其他信息

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
void onCallBegin(String callId, TUICallDefine.MediaType mediaType, TUICallDefine.CallObserverExtraInfo
info);
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID
mediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
info	TUICallDefine.CallObserverExtraInfo	其他信息

onCallEnd

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
void onCallEnd(String callId, TUICallDefine.MediaType mediaType, TUICallDefine.CallEndReason reason,
               String userId, long totalTime, TUICallDefine.CallObserverExtraInfo info);
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的音视频房间 ID
mediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
reason	TUICallDefine.CallEndReason	通话结束原因
userId	String	结束通话的用户 ID

totalTime	long	此次通话的时长，单位：秒
info	TUICallDefine.CallObserverExtraInfo	其他信息

> **注意：**
> 客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallNotConnected

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallNotConnected 回调（userId 是自己的 ID）；被叫收到 onCallNotConnected 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallNotConnected 回调（userId 是自己的 ID）；被叫收到 onCallNotConnected 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallNotConnected 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
void onCallNotConnected(String callId, TUICallDefine.MediaType mediaType, TUICallDefine.CallEndReason
reason,
                        String userId, TUICallDefine.CallObserverExtraInfo info)
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的音视频房间 ID
mediaType	TUICallDefine.MediaType	通话的媒体类型，视频通话、语音通话
reason	TUICallDefine.CallEndReason	通话未连接原因
userId	String	导致通话未连接的用户 ID
info	TUICallDefine.CallObserverExtraInfo	其他信息

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。

```
void onUserReject(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。

```
void onUserNoResponse(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	无响应用户的 ID

onUserInviting

用户被追加邀请加入通话时的回调。

```
void onUserInviting(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	被追加邀请用户的 ID

onUserLineBusy

通话忙线回调。

```
void onUserLineBusy(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	忙线用户的 ID

onUserJoin

有用户进入此次通话的回调。

```
void onUserJoin(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。

```
void onUserLeave(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。

```
void onUserVideoAvailable(String userId, boolean isVideoAvailable);
```

参数如下表所示：

参数	类型	含义
userId	String	通话用户 ID
isVideoAvailable	boolean	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。

```
void onUserAudioAvailable(String userId, boolean isAudioAvailable);
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID
isAudioAvailable	boolean	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。

```
void onUserVoiceVolumeChanged(Map<String, Integer> volumeMap);
```

参数如下表所示：

参数	类型	含义
volumeMap	Map<String, Integer>	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。

```
void onUserNetworkQualityChanged(List<TUICallDefine.NetworkQualityInfo> networkQualityList);
```

参数如下表所示：

参数	类型	含义
networkQualityList	List	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。

```
void onKickedOffline();
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。

```
void onUserSigExpired();
```

废弃回调

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
void onCallCancelled(String userId);
```

参数如下表所示：

参数	类型	含义
userId	String	用户的 ID

onCallMediaTypeChanged

表示通话的媒体类型发生变化。

```
void onCallMediaTypeChanged(TUICallDefine.MediaType oldCallMediaType,TUICallDefine.MediaType
newCallMediaType);
```

参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallDefine.MediaType	旧的通话类型
newCallMediaType	TUICallDefine.MediaType	新的通话类型

版权所有：腾讯云计算（北京）有限责任公司

类型定义

最近更新时间：2025-06-30 20:45:42

常用结构

TUICallDefine

类型	描述
CallParams	通话参数
OfflinePushInfo	离线推送厂商配置信息
CallObserverExtraInfo	扩展信息

TUICommonDefine

类型	描述
RoomId	通话中音视频房间 ID
NetworkQualityInfo	网络质量
VideoRenderParams	视频画面的渲染参数
VideoEncoderParams	视频编码参数

枚举定义

TUICallDefine

类型	说明
MediaType	通话类型
Role	通话角色
Status	通话状态
Scene	通话场景
IOSOfflinePushType	iOS 离线推送类型
CallEndReason	通话结束原因

TUICommonDefine

类型	说明
AudioPlaybackDevice	音频设备
Camera	摄像头类型
NetworkQuality	网络质量
FillMode	视频画面填充模式
Rotation	视频画面旋转方向
ResolutionMode	视频宽高比模式
Resolution	视频分辨率

版权所有：腾讯云计算（北京）有限责任公司

CallParams

通话参数

类型	类型	描述
roomId	TUICommonDefine.RoomId	通话中音视频房间ID
offlinePushInfo	OfflinePushInfo	厂商离线推送配置信息
timeout	int	通话超时时间，默认：30s，单位：秒。设置范围：10s ~ 600s。如果传入0，默认30s
userData	String	发起通话时自定义扩展字段，被叫端在 onCallReceived 中回调
chatGroupId	String	与 Chat 结合使用时，Chat 群组的群 ID

OfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参见 离线唤醒 。

类型	类型	描述
title	String	离线推送展示通知栏标题（必填）
desc	String	离线推送展示通知栏内容（必填）
ignoreIOSBadge	boolean	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加。
iOSSound	String	离线推送声音设置（仅对 iOS 生效）。当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	String	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。详细参见：推送自定义铃声 。
androidOPPOChannelID	String	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidVIVOClassification	int	VIVO 推送消息分类 (待废弃接口，VIVO 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1
androidXiaoMiChannelID	String	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	String	FCM 通道手机 8.0 系统及以上的渠道 ID
androidHuaWeiCategory	String	华为推送消息分类，详细参见：华为消息分类标准
isDisablePush	boolean	是否关闭推送（默认开启推送）
iOSPushType	IOSOfflinePushType	iOS 离线推送类型，默认：APNs

CallObserverExtraInfo

扩展信息

参数	类型	描述
roomId	TUICommonDefine.RoomId	通话中音视频房间 ID

role	Role	当前登录用户的通话角色
userData	NSString	发起通话时自定义的扩展字段
chatGroupId	NSString	IM 群组 ID

RoomId

通话中音视频房间 ID

- intRoomId 与 strRoomId 是互斥的，若您选用 strRoomId，则 intRoomId 需要填写为 0。若两者都填，SDK 将优先选用 intRoomId。
- 不要混用 intRoomId 和 strRoomId，因为它们之间是不互通的，比如数字123和字符串123是两个完全不同的房间。

类型	类型	描述
intRoomId	int	数字房间号 **取值范围：** 1 – 2147483647(2^31-1)
strRoomId	String	字符串房间号 **推荐取值：** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a-zA-Z）； • 数字（0-9）； • 空格、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<`、`=`、`.`、`>`、`?`、`@`、`[`、`]`、`^`、`_`、`{`、`}`、`|`、`~`、`,`。

NetworkQualityInfo

用户网络质量

类型	类型	描述
userId	String	用户 ID
quality	NetworkQuality	网络质量

VideoRenderParams

视频画面的渲染参数

类型	类型	描述
fillMode	VideoRenderParams.FillMode	视频画面填充模式
rotation	VideoRenderParams.Rotation	视频画面旋转方向

VideoEncoderParams

视频编码参数

类型	类型	描述
resolution	VideoEncoderParams.Resolution	视频分辨率
resolutionMode	VideoEncoderParams.ResolutionMode	视频宽高比

MediaType

通话类型

类型	取值	描述
Unknown	0	未知类型

Audio	1	语音通话
Video	2	视频通话

Role

通话角色

类型	取值	描述
None	0	未知类型
Caller	1	主叫（邀请方）
Called	2	被叫（被邀请方）

Status

通话状态

类型	取值	描述
None	0	未定义
Waiting	1	通话等待中
Accept	2	通话已接听

Scene

通话场景

类型	取值	描述
SINGLE_CALL	0	单人通话
GROUP_CALL	1	群组通话
MULTI_CALL	2	多人通话

IOSOfflinePushType

iOS 离线推送类型

类型	取值	描述
APNs	0	普通推送
VoIP	1	VoIP 推送

CallEndReason

通话结束原因

类型	取值	描述
UNKNOWN	0	未知
HANGUP	1	挂断
REJECT	2	拒绝
NO_RESPONSE	3	无响应

版权所有：腾讯云计算（北京）有限责任公司

OFFLINE	4	离线
LINE_BUSY	5	忙线
CANCELED	6	取消通话
OTHER_DEVICE_ACCEPTED	7	其他设备接听
OTHER_DEVICE_REJECT	8	其他设备拒绝
END_BY_SERVER	9	后台结束

AudioPlaybackDevice

音频设备

类型	取值	描述
Speakerphone	0	扬声器
Earpiece	1	听筒

Camera

前置/后置摄像头

类型	取值	描述
Front	0	前置摄像头
Back	1	后置摄像头

NetworkQuality

网络质量

类型	取值	描述
UNKNOWN	0	未定义
EXCELLENT	1	当前网络非常好
GOOD	2	当前网络比较好
POOR	3	当前网络一般
BAD	4	当前网络较差
VERY_BAD	5	当前网络很差
DOWN	6	当前网络不满足通话的最低要求

FillMode

视频画面填充模式

类型	取值	描述
Fill	0	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。
Fit	1	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

Rotation

视频画面旋转方向

类型	取值	描述
Rotation_0	0	不旋转
Rotation_90	1	顺时针旋转90度
Rotation_180	2	顺时针旋转180度
Rotation_270	3	顺时针旋转270度

ResolutionMode

视频宽高比模式

类型	取值	描述
Landscape	0	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360。
Portrait	1	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640。

Resolution

视频分辨率

类型	取值	描述
Resolution_640_360	108	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps
Resolution_960_540	110	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps
Resolution_1280_720	112	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps
Resolution_1920_1080	114	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps

版权所有：腾讯云计算（北京）有限责任公司

iOS
API 概览

最近更新时间：2025-06-26 10:31:51

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的昵称、头像
calls	发起单人或多人通话
join	主动加入通话
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
enableIncomingBanner	开启/关闭来电横幅显示

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
calls	发起单人或多人通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
inviteUser	在通话中，邀请其他人加入
join	主动加入通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头

版权所有：腾讯云计算（北京）有限责任公司

openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

TUICallObserver

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallEnd	通话结束的回调
onCallNotConnected	通话未接通的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserInviting	xxxx 用户被追加邀请加入通话时的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

关键类型定义

API	描述
TUICallMediaType	通话的媒体类型，枚举类型：视频通话、语音通话
TUICallRole	通话的角色，枚举类型：主叫、被叫

TUICallStatus	通话的状态，枚举类型：空闲、待接听、接听中
TUIRoomId	音视频房间 ID，支持数字、字符串两种类型
TUICamera	摄像头 ID 参数，枚举类型：前置摄像头、后置摄像头
TUIAudioPlaybackDevice	声音的播放设备，枚举类型：扬声器、听筒
TUINetworkQuality	当前的网络质量信息

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2025-06-30 14:57:51

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入TUICallKit 。

API 概览

API	描述
createInstance	创建 TUICallKit 实例（单例模式）
setSelfInfo	设置用户的昵称、头像
calls	发起单人或多人通话
join	主动加入通话
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
enableIncomingBanner	开启/关闭来电横幅显示

API 详情

createInstance

创建 TUICallKit 的单例。

Objective-C

```
- (instancetype)createInstance;
```

Swift

```
public static func createInstance() -> TUICallKit
```

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

Objective-C

```
- (void)setSelfInfo:(NSString * _Nullable)nickname avatar:(NSString * _Nullable)avatar succ:
(TUICallSucc)succ fail:(TUICallFail)fail;
```

Swift

```
public func setSelfInfo(nickname: String, avatar: String, succ:@escaping TUICallSucc, fail: @escaping
TUICallFail)
```

参数	类型	含义
nickname	NSString	用户昵称
avatar	NSString	用户头像（格式为 URL）
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

calls

发起通话。

Swift

```
func calls(userIdList: [String], callMediaType: TUICallMediaType, params: TUICallParams,
                       succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

Objective-C

```
- (void)calls:(NSArray<NSString *> *)userIdList callMediaType:(TUICallMediaType)callMediaType params:
(TUICallParams *)params succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数如下表所示：

参数	类型	含义
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

join

主动加入通话。

Swift

```
func join(callId: String)
```

Objective-C

```
- (void)join:(NSString *)callId;
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的唯一 ID

setCallingBell

设置自定义来电铃音。
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

- 铃声设置后与设备绑定，更换用户，铃声依旧会生效。
- 如需恢复默认铃声，`filePath` 传空即可。

Objective-C

```
- (void)setCallingBell:(NSString *)filePath;
```

Swift

```
public func setCallingBell(filePath: String)
```

enableMuteMode

开启后，收到通话请求，不会播放来电铃声。

Objective-C

```
- (void)enableMuteMode:(BOOL)enable;
```

Swift

```
public func enableMuteMode(enable: Bool)
```

enableFloatWindow

开启/关闭悬浮窗功能。
默认为 `false`，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

Objective-C

```
- (void)enableFloatWindow:(BOOL)enable;
```

Swift

```
public func enableFloatWindow(enable: Bool)
```

enableIncomingBanner

开启/关闭来电横幅显示。
默认为 `false`，被叫端收到邀请后默认弹出全屏通话等待界面，开启后先展示一个横幅，然后根据需要拉起全屏通话界面。

```
public func enableIncomingBanner(enable: Bool)
```

废弃接口

call

拨打电话（1v1通话）。**注意：建议使用 calls 接口。**

Objective-C

```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType;
```

Swift

```
public func call(userId: String, callMediaType: TUICallMediaType)
```

参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

call

拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。**注意：建议使用 calls 接口。**

Objective-C

```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType params:(TUICallParams
*)params succ:(TUICallSucc __nullable)succ fail:(TUICallFail __nullable)fail;
```

Swift

```
public func call(userId: String, callMediaType: TUICallMediaType, params: TUICallParams,
                 succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

groupCall

发起群组通话。**注意：建议使用 calls 接口。**

> ⚠ **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList callMediaType:
(TUICallMediaType)callMediaType;
```

Swift

```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICallMediaType)
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

groupCall

发起群组通话，支持自定义房间号、通话邀请超时时间，离线推送内容等。注意：建议使用 calls 接口。

> ⚠ **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray<NSString *> *)userIdList callMediaType:
(TUICallMediaType)callMediaType params:(TUICallParams *)params succ:(TUICallSucc __nullable)succ fail:
(TUICallFail __nullable)fail;
```

Swift

```
public func groupCall(groupId: String, userIdList: [String], callMediaType: TUICallMediaType, params:
TUICallParams, succ: @escaping TUICallSucc, fail: @escaping TUICallFail)
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

joinInGroupCall

加入群组中已有的音视频通话。注意：建议使用 join 接口。

> ⚠ **注意：**
> 加入群组中已有的音视频通话前，需要提前创建或加入 IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。
> 群组的创建详见：IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

Objective-C

```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaType:
(TUICallMediaType)callMediaType;
```

Swift

```
public func joinInGroupCall(roomId: TUIRoomId, groupId: String, callMediaType: TUICallMediaType)
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID
groupId	NSString	此次群组通话的群 ID
callMediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话

TUICallEngine

最近更新时间：2025-06-30 15:01:12

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）
destroyInstance	销毁 TUICallEngine 实例（单例模式）
init	完成音视频通话基础能力的鉴权
addObserver	增加事件回调
removeObserver	移除回调接口
calls	发起单人或多人通话
accept	接听通话
reject	拒绝通话
hangup	结束通话
inviteUser	在通话中，邀请其他人加入
join	主动加入通话
startRemoteView	开始订阅远端用户视频流
stopRemoteView	停止订阅远端用户视频流
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的昵称、头像
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数
getTRTCCloudInstance	高级接口，用于接入第三方美颜等
setBeautyLevel	设置美颜级别，支持关闭默认美颜

API 详情

createInstance

创建 TUICallEngine 的单例。

```
- (TUICallEngine *)createInstance;
```

destroyInstance

销毁 TUICallEngine 的单例。

```
- (void)destroyInstance;
```

init

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。

```
- (void)init:(NSString *)sdkAppID userId:(NSString *)userId userSig:(NSString *)userSig succ:
(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数如下表所示：

参数	类型	含义
sdkAppID	NSString	您可以在**实时音视频控制台** > **应用管理** > **应用信息**中查看 SDKAppID
userId	NSString	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）
userSig	NSString	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

addObserver

添加回调接口，您可以通过这个接口，监听 TUICallObserver 相关的事件回调。

```
- (void)addObserver:(id<TUICallObserver>)observer;
```

removeObserver

移除回调接口。

```
- (void)removeObserver:(id<TUICallObserver>)observer;
```

calls

发起通话。

```
- (void)calls:(NSArray<NSString *> *)userIdList callMediaType:(TUICallMediaType)callMediaType params:
(TUICallParams *)params succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数如下表所示：

参数	类型	含义
userIdList	NSArray	目标用户的 userId 列表

callMediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话
params	TUICallParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等

accept

接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。

```
- (void)accept:(TUICallSucc)succ fail:(TUICallFail)fail;
```

reject

拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。

```
- (void)reject:(TUICallSucc)succ fail:(TUICallFail)fail;
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
- (void)hangup:(TUICallSucc)succ fail:(TUICallFail)fail;
```

inviteUser

邀请用户加入此次群组通话。
使用场景：一个群组通话中的用户主动邀请其他人时使用。

```
- (void)inviteUser:(NSArray<NSString *> *)userIdList params:(TUICallParams *)params succ:(void(^)
(NSArray <NSString *> *userIdList))succ fail:(TUICallFail)fail
```

参数	类型	含义
userIdList	NSArray	目标用户的 userId 列表
params	TUICallParams	通话参数扩展字段，例如：离线推送自定义内容
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

join

主动加入通话。

```
- (void)join:(NSString *)callId succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的唯一 ID

startRemoteView

设置显示视频画面的 View 对象

```
- (void)startRemoteView:(NSString *)userId videoView:(TUIVideoView *)videoView onPlaying:(void(^)
(NSString *userId))onPlaying onLoading:(void(^)(NSString *userId))onLoading onError:(void(^)(NSString
```

```
*userId, int code, NSString *errMsg))onError;
```

参数	类型	含义
userId	NSString	目标用户的 userId
videoView	TUIVideoView	待渲染的视图

stopRemoteView

停止订阅远端用户的视频数据。

```
- (void)stopRemoteView:(NSString *)userId;
```

参数	类型	含义
userId	NSString	目标用户的 userId

openCamera

开启摄像头。

```
- (void)openCamera:(TUICamera)camera videoView:(TUIVideoView *)videoView succ:(TUICallSucc)succ fail:
(TUICallFail)fail;
```

参数	类型	含义
camera	TUICamera	前置/后置 摄像头
videoView	TUIVideoView	待渲染的视图
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

closeCamera

关闭摄像头。

```
- (void)closeCamera;
```

switchCamera

切换前后摄像头。

```
- (void)switchCamera:(TUICamera)camera;
```

参数	类型	含义
camera	TUICamera	前置/后置摄像头

openMicrophone

打开麦克风。

```
- (void)openMicrophone:(TUICallSucc)succ fail:(TUICallFail)fail;
```

closeMicrophone

关闭麦克风。

```
- (void)closeMicrophone;
```

selectAudioPlaybackDevice

选择音频播放设备。
目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。

```
- (void)selectAudioPlaybackDevice:(TUIAudioPlaybackDevice)device;
```

参数	类型	含义
device	TUIAudioPlaybackDevice	听筒/扬声器

setSelfInfo

设置用户昵称、头像。
用户昵称不能超过500字节，用户头像必须是 URL 格式。

```
- (void)setSelfInfo:(NSString *_Nullable)nickname avatar:(NSString *_Nullable)avatar succ:
(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数	类型	含义
nickname	NSString	用户昵称
avatar	NSString	用户头像（格式为 URL）
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

enableMultiDeviceAbility

开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）。

```
- (void)enableMultiDeviceAbility:(BOOL)enable succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

setVideoRenderParams

设置用户视频画面的渲染模式。

```
- (void)setVideoRenderParams:(NSString *)userId params:(TUIVideoRenderParams *)params succ:
(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数	类型	含义
userId	NSString	目标用户的 userId
params	TUIVideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式
succ	TUICallSucc	成功回调

fail	TUICallFail	失败回调

setVideoEncoderParams

设置视频编码的参数。
该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。

```
- (void)setVideoEncoderParams:(TUIVideoEncoderParams *)params succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数	类型	含义
params	TUIVideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高比
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

getTRTCCloudInstance

高级接口。
用于接入第三方美颜等。

```
- (TRTCCloud *)getTRTCCloudInstance;
```

setBeautyLevel

设置美颜级别，支持关闭默认美颜。

```
- (void)setBeautyLevel:(CGFloat)level succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数如下表所示：

参数	类型	含义
level	CGFloat	美颜级别，美颜级别，取值范围 0 - 9； 0 表示关闭，9 表示效果最明显。
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

废弃接口

call

拨打电话（1v1通话）。**注意：v2.9+版本废弃，建议使用 calls 接口。**

> ⚠ **注意：**
> 字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！

```
- (void)call:(NSString *)userId callMediaType:(TUICallMediaType)callMediaType params:(TUICallParams *)params succ:(TUICallSucc)succ fail:(TUICallFail)fail
```

参数如下表所示：

参数	类型	含义
userId	NSString	目标用户的 userId
callMediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

ignore

忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。注意：v2.9+版本废弃，针对 calls 发起的通话不生效，建议使用 reject 接口。

```
- (void)ignore:(TUICallSucc)succ fail:(TUICallFail)fail;
```

groupCall

发起群组通话。注意：v2.9+版本废弃，建议使用 calls 接口。

> **注意：**
> 使用群组通话前需要创建 IM 群组，如果已经创建，请忽略；
> 字符串房间号目前只支持 Android 和 iOS 平台，Web、小程序、Flutter、Uniapp 等其他平台后续支持，敬请期待！

```
- (void)groupCall:(NSString *)groupId userIdList:(NSArray <NSString *> *)userIdList callMediaType:
(TUICallMediaType)callMediaType params:(TUICallParams *)params succ:(TUICallSucc)succ fail:
(TUICallFail)fail
```

参数	类型	含义
groupId	NSString	此次群组通话的群 ID
userIdList	NSArray	目标用户的 userId 列表
callMediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

joinInGroupCall

主动加入此次群组通话。注意：v2.9+版本废弃，建议使用 join 接口。
使用场景：群组内用户主动加入此次群组通话使用。

```
- (void)joinInGroupCall:(TUIRoomId *)roomId groupId:(NSString *)groupId callMediaType:
(TUICallMediaType)callMediaType succ:(TUICallSucc)succ fail:(TUICallFail)fail;
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID

groupId	NSString	此次群组通话的群 ID
callMediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话
succ	TUICallSucc	成功回调
fail	TUICallFail	失败回调

switchCallMediaType

切换视频通话到语音通话。注意：v2.9+版本废弃，针对 calls 发起的通话不生效；

```
- (void)switchCallMediaType:(TUICallMediaType)newType;
```

参数	类型	含义
callMediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话

TUICallObserver

最近更新时间：2025-06-30 14:37:32

TUICallObserver API 简介

TUICallObserver 是 TUICallEngine 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallNotConnected	通话未接通的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserInviting	xxxx 用户被追加邀请加入通话时的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxxx 用户是否有视频流的回调
onUserAudioAvailable	xxxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期

回调事件详情

onError

错误回调。

> **说明：**
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
- (void)onError:(int)code message:(NSString * _Nullable)message;
```

参数如下表所示：

参数	类型	含义
code	int	错误码

message	NSString	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
- (void)onCallReceived:(NSString *)callId callerId:(NSString *)callerId calleeIdList:(NSArray<NSString
*> *)calleeIdList mediaType:(TUICallMediaType)mediaType info:(TUICallObserverExtraInfo *)info;
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的唯一 ID
callerId	NSString	主叫 ID（邀请方）
calleeIdList	NSArray	被叫 ID 列表（被邀请方）
mediaType	TUICallMediaType	通话的媒体类型，比如视频通话、语音通话
info	TUICallObserverExtraInfo	其他信息

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
- (void)onCallBegin:(NSString *)callId mediaType:(TUICallMediaType)mediaType info:
(TUICallObserverExtraInfo *)info;
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的唯一 ID
mediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
info	TUICallObserverExtraInfo	其他信息

onCallEnd

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
- (void)onCallEnd:(NSString *)callId mediaType:(TUICallMediaType)mediaType reason:
(TUICallEndReason)reason userId:(NSString *)userId totalTime:(float)totalTime info:
(TUICallObserverExtraInfo *)info;
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的音视频房间 ID
mediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
reason	TUICallEndReason	通话结束原因
userId	NSString	结束通话的用户 ID
totalTime	float	此次通话的时长，单位：秒

版权所有：腾讯云计算（北京）有限责任公司

info	TUICallObserverExtraInfo	其他信息

注意：
客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallNotConnected

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallNotConnected 回调（userId 是自己的 ID）；被叫收到 onCallNotConnected 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallNotConnected 回调（userId 是自己的 ID）；被叫收到 onCallNotConnected 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallNotConnected 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
- (void)onCallNotConnected:(NSString *)callId mediaType:(TUICallMediaType)mediaType reason:(TUICallEndReason)reaso userId:(NSString *)userId info:(TUICallObserverExtraInfo *)info
```

参数如下表所示：

参数	类型	含义
callId	NSString	此次通话的音视频房间 ID
mediaType	TUICallMediaType	通话的媒体类型，视频通话、语音通话
reason	TUICallEndReason	通话未连接原因
userId	NSString	导致通话未连接的用户 ID
info	TUICallObserverExtraInfo	其他信息

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。

```
- (void)onUserReject:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。

```
- (void)onUserNoResponse:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	无响应用户的 ID

onUserLineBusy

通话忙线回调。

```
- (void)onUserLineBusy:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	忙线用户的 ID

onUserInviting

用户被追加邀请加入通话时的回调。

```
- (void)onUserInviting:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	被追加邀请用户的 ID

onUserJoin

有用户进入此次通话的回调。

```
- (void)onUserJoin:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。

```
- (void)onUserLeave:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。

```
- (void)onUserVideoAvailable:(NSString *)userId isVideoAvailable:(BOOL)isVideoAvailable;
```

参数如下表所示：

参数	类型	含义

版权所有：腾讯云计算（北京）有限责任公司

userId	NSString	通话用户 ID
isVideoAvailable	BOOL	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。

```
- (void)onUserAudioAvailable:(NSString *)userId isAudioAvailable:(BOOL)isAudioAvailable;
```

参数如下表所示：

参数	类型	含义
userId	NSString	通话用户 ID
isAudioAvailable	BOOL	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。

```
- (void)onUserVoiceVolumeChanged:(NSDictionary <NSString *, NSNumber *> *)volumeMap;
```

参数如下表所示：

参数	类型	含义
volumeMap	NSDictionary	音量表，根据每个 userId 可以获取对应的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。

```
- (void)onUserNetworkQualityChanged:(NSArray<TUINetworkQualityInfo *> *)networkQualityList;
```

参数如下表所示：

参数	类型	含义
networkQualityList	NSArray	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

当前用户被踢下线：此时可以 UI 提示用户，并重新调用初始化。

```
- (void)onKickedOffline;
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并重新调用初始化。

```
- (void)onUserSigExpired;
```

废弃回调

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
- (void)onCallCancelled:(NSString *)userId;
```

参数如下表所示：

参数	类型	含义
userId	NSString	用户的 ID

onCallMediaTypeChanged

表示通话的媒体类型发生变化。

```
- (void)onCallMediaTypeChanged:(TUICallMediaType)oldCallMediaType newCallMediaType:
(TUICallMediaType)newCallMediaType;
```

参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallMediaType	旧的通话类型
newCallMediaType	TUICallMediaType	新的通话类型

类型定义

最近更新时间：2025-06-30 20:45:42

常用结构

TUICallDefine

类型	描述
TUICallParams	通话参数
TUIOfflinePushInfo	离线推送厂商配置信息
TUICallObserverExtraInfo	扩展信息

TUICommonDefine

类型	描述
TUIRoomId	通话中音视频房间 ID
TUIVideoRenderParams	视频画面的渲染参数
TUIVideoEncoderParams	视频编码参数

枚举定义

TUICallDefine

类型	说明
TUICallMediaType	通话类型
TUICallRole	通话角色
TUICallStatus	通话状态
TUICallScene	通话场景
TUICallIOSOfflinePushType	iOS 离线推送类型
TUICallEndReason	通话结束原因

TUICommonDefine

类型	说明
TUIAudioPlaybackDevice	音频设备
TUICamera	摄像头类型
TUINetworkQuality	网络质量
TUIVideoRenderParamsFillMode	视频画面填充模式
TUIVideoRenderParamsRotation	视频画面旋转方向
TUIVideoEncoderParamsResolutionMode	视频宽高比模式
TUIVideoEncoderParamsResolution	视频分辨率

TUICallParams

通话参数

参数	类型	描述
offlinePushInfo	TUIOfflinePushInfo	厂商离线推送配置信息
timeout	int	通话超时时间，默认：30s，单位：秒。设置范围：10s ~ 600s。如果传入0，默认30s
userData	NSString	发起通话时自定义扩展字段，被叫端在 onCallReceived 中回调
roomId	TUIRoomId	通话中音视频房间 ID
chatGroupId	NSString	IM 群组 ID

TUIOfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参考：离线唤醒 。

参数	类型	描述
title	NSString	离线推送展示通知栏标题
desc	NSString	离线推送展示通知栏内容
ignoreIOSBadge	BOOL	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加
iOSSound	NSString	离线推送声音设置（仅对 iOS 生效）。当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	NSString	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声 另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效
androidOPPOChannelID	NSString	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidVIVOClassification	NSInteger	VIVO 推送消息分类 (待废弃接口，VIVO 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1
androidXiaoMiChannelID	NSString	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	NSString	FCM 通道手机 8.0 系统及以上的渠道 ID
androidHuaWeiCategory	NSString	华为推送消息分类，详见：华为消息分类标准
isDisablePush	BOOL	是否关闭推送（默认开启推送）
iOSPushType	TUICallIOSOfflinePushType	iOS 离线推送类型，默认：APNs

TUICallObserverExtraInfo

扩展信息

参数	类型	描述

roomId	TUIRoomId	通话中音视频房间 ID
role	TUICallRole	当前登录用户的通话角色
userData	NSString	发起通话时自定义的扩展字段
chatGroupId	NSString	IM 群组 ID

TUIRoomId

通话中音视频房间ID

- intRoomId 与 strRoomId 是互斥的，若您选用 strRoomId，则 intRoomId 需要填写为 0。若两者都填，SDK 将优先选用 intRoomId。
- 不要混用 intRoomId 和 strRoomId，因为它们之间是不互通的，比如数字 123 和字符串 123 两个完全不同的房间。

参数	类型	描述
intRoomId	UInt32	数字房间号，**取值范围**：1 – 2147483647(2^31-1)
strRoomId	NSString	字符串房间号，**推荐取值：** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： 大小写英文字母（a-zA-Z）； 数字（0-9）； 空格、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<`、`=`、`.`、`>`、`?`、`@`、`[`、`]`、`^`、`_`、`{`、`}`、`|`、`~`、`,`。

TUIVideoRenderParams

视频画面的渲染参数

参数	类型	描述
fillMode	TUIVideoRenderParamsFillMode	视频画面填充模式
rotation	TUIVideoRenderParamsRotation	视频画面旋转方向

TUIVideoEncoderParams

视频编码参数

参数	类型	描述
resolution	TUIVideoEncoderParamsResolution	视频分辨率
resolutionMode	TUIVideoEncoderParamsResolutionMode	视频宽高比

TUICallMediaType

通话类型

类型	取值	描述
TUICallMediaTypeUnknown	0	未知类型
TUICallMediaTypeAudio	1	语音通话
TUICallMediaTypeVideo	2	视频通话

TUICallRole

通话角色

类型	取值	描述
TUICallRoleNone	0	未知类型
TUICallRoleCall	1	主叫（邀请方）
TUICallRoleCalled	2	被叫（被邀请方）

TUICallStatus

通话状态

类型	取值	描述
TUICallStatusNone	0	未定义
TUICallStatusWaiting	1	通话等待中
TUICallStatusAccept	2	通话已接听

TUICallScene

通话场景

类型	取值	描述
TUICallSceneGroup	0	群组通话
TUICallSceneMulti	1	匿名群组通话（暂不支持，敬请期待）
TUICallSceneSingle	2	单人通话

TUICallIOSOfflinePushType

iOS 离线推送类型

类型	取值	描述
TUICallIOSOfflinePushTypeAPNs	0	普通推送
TUICallIOSOfflinePushTypeVoIP	1	VoIP 推送

TUICallEndReason

通话结束原因

类型	取值	描述
TUICallEndReasonUnknown	0	未知
TUICallEndReasonHangup	1	挂断
TUICallEndReasonReject	2	拒绝
TUICallEndReasonNoResponse	3	无响应
TUICallEndReasonOffline	4	离线
TUICallEndReasonLineBusy	5	忙线
TUICallEndReasonCanceled	6	取消通话
TUICallEndReasonOtherDeviceAccepted	7	其他设备接听

版权所有：腾讯云计算（北京）有限责任公司

TUICallEndReasonOtherDeviceReject	8	其他设备拒绝
TUICallEndReasonEndByServer	9	后台结束

TUIAudioPlaybackDevice

音频设备

类型	取值	描述
TUIAudioPlaybackDeviceSpeakerphone	0	扬声器
TUIAudioPlaybackDeviceEarpiece	1	听筒

TUICamera

前置/后置摄像头

类型	取值	描述
TUICameraFront	0	前置摄像头
TUICameraBack	1	后置摄像头

TUINetworkQuality

网络质量

类型	取值	描述
TUINetworkQualityUnknown	0	未定义
TUINetworkQualityExcellent	1	当前网络非常好
TUINetworkQualityGood	2	当前网络比较好
TUINetworkQualityPoor	3	当前网络一般
TUINetworkQualityBad	4	当前网络较差
TUINetworkQualityVbad	5	当前网络很差
TUINetworkQualityDown	6	当前网络不满足通话的最低要求

TUIVideoRenderParamsFillMode

视频画面填充模式

类型	取值	描述
TUIVideoRenderParamsFillModeFill	0	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整
TUIVideoRenderParamsFillModeFit	1	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边

TUIVideoRenderParamsRotation

视频画面旋转方向

类型	取值	描述

TUIVideoRenderParamsRotation_0	0	不旋转
TUIVideoRenderParamsRotation_90	1	顺时针旋转90度
TUIVideoRenderParamsRotation_180	2	顺时针旋转180度
TUIVideoRenderParamsRotation_270	3	顺时针旋转270度

TUIVideoEncoderParamsResolutionMode

视频宽高比模式

类型	取值	描述
TUIVideoEncoderParamsResolutionMode Landscape	0	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360
TUIVideoEncoderParamsResolutionMode Portrait	1	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640

TUIVideoEncoderParamsResolution

视频分辨率

类型	取值	描述
TUIVideoEncoderParamsResolution_640_360	1	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps
TUIVideoEncoderParamsResolution_960_540	2	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps
TUIVideoEncoderParamsResolution_1280_720	3	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps
TUIVideoEncoderParamsResolution_1920_1080	4	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps

版权所有：腾讯云计算（北京）有限责任公司

Web
API 概览

最近更新时间：2025-06-19 17:48:02

TUICallKit（含 UI 组件）

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
init	初始化 TUICallKit 组件实例。
calls	发起单人或多人通话。
join	主动加入通话。
setCallingBell	设置自定义来电铃音。
setSelfInfo	设置用户昵称和头像。
enableMuteMode	开启/关闭来电铃声。
enableFloatWindow	开启/关闭悬浮窗功能。
enableVirtualBackground	开启/关闭模糊背景的功能按钮。
setLanguage	设置 TUICallKit 组件通话语言。
hideFeatureButton	隐藏按钮。
setLocalViewBackgroundImage	设置本地用户通话界面背景图。
setRemoteViewBackgroundImage	设置远端用户通话界面背景图。
setLayoutMode	设置通话界面布局模式。
setCameraDefaultState	设置摄像头是否默认打开。
destroyed	销毁 TUICallKit 组件实例。
getTUICallEngineInstance	获取 TUICallEngine 实例。

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 CallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
on	监听事件。
off	取消监听事件。
login	登录接口。
logout	登出接口。
setSelfInfo	设置用户昵称和头像。

版权所有：腾讯云计算（北京）有限责任公司

calls	发起单人或多人通话。
join	主动加入通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
startRemoteView	启动远端画面渲染。
stopRemoteView	停止远端画面渲染。
openCamera	开启摄像头。
closeCamara	关闭摄像头。
switchCamera	切换前后置摄像头，仅移动端支持。
openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
setVideoQuality	设置视频分辨率。
getDeviceList	获取设备列表。
switchDevice	切换摄像头或麦克风设备。
enableAIVoice	开启/关闭 AI 降噪。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式。
setBlurBackground	开关/设置背景虚化。
setVirtualBackground	开关/设置图片背景虚化。

TUICallEvent

TUICallEvent 是 TUICallEngine 对应的事件类，您可以通过监听如下事件，来监听自己感兴趣的事件。

EVENT	描述
TUICallEvent.ERROR	通话过程中错误事件
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该事件
TUICallEvent.KICKED_OUT	用户重复登录，被踢下线
TUICallEvent.USER_ACCEPT	xxxx 用户接通的事件
TUICallEvent.USER_ENTER	xxxx 用户加入通话的事件
TUICallEvent.USER_LEAVE	xxxx 用户离开通话的事件
TUICallEvent.REJECT	xxxx 用户拒绝通话的事件
TUICallEvent.NO_RESP	xxxx 用户不响应的事件
TUICallEvent.LINE_BUSY	xxxx 用户忙线的事件
TUICallEvent.USER_VIDEO_AVAILABLE	xxxx 用户是否有视频流的事件
TUICallEvent.USER_AUDIO_AVAILABLE	xxxx 用户是否有音频流的事件
TUICallEvent.USER_VOICE_VOLUME	所有用户音量大小的反馈事件

版权所有：腾讯云计算（北京）有限责任公司

TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该事件
TUICallEvent.ON_CALL_BEGIN	通话接通的事件，**v1.4.6+ 支持**
TUICallEvent.INVITED	通话请求的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_RECEIVED**
TUICallEvent.ON_CALL_RECEIVED	通话请求的事件，**v1.4.6+ 支持**
TUICallEvent.CALLING_CANCEL	通话取消的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_CANCELED	通话取消的事件，**v1.4.6+ 支持**
TUICallEvent.ON_CALL_BEGIN	通话接通时抛出的事件
TUICallEvent.CALLING_END	通话结束的事件
TUICallEvent.DEVICED_UPDATED	设备列表更新的事件
TUICallEvent.CALL_TYPE_CHANGED	通话的媒体类型发生改变的回调
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件，**v3.0.7+ 支持**

TUICallKit

最近更新时间：2025-06-30 15:22:42

TUICallKit 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景；更详细的接入步骤，详见：快速接入 TUICallKit 。

说明：

- 为提供更优质的音视频通信能力， TUICallKit npm 包 @tencentcloud/call-uikit-react 、@tencentcloud/call-uikit-vue 、@tencentcloud/call-uikit-vue2 、@tencentcloud/call-uikit-vue2.6 在 4.0.0 版本进行了重大升级。
- v4.0.0+ 对接口进行了优化调整，废弃了部分接口，同步提供更简洁、稳定的新接口方案。

API 概览

API	描述
<TUICallKit />	TUICallKit 通话 UI 组件。
init	初始化 TUICallKit 组件实例。
calls	发起单人或多人通话。
join	主动加入通话。
setCallingBell	设置自定义来电铃音。
setSelfInfo	设置用户昵称和头像。
enableMuteMode	开启/关闭来电铃声。
enableFloatWindow	开启/关闭悬浮窗功能。
enableVirtualBackground	开启/关闭模糊背景的功能按钮。
setLanguage	设置 TUICallKit 组件通话语言。
hideFeatureButton	隐藏按钮。
setLocalViewBackgroundImage	设置本地用户通话界面背景图。
setRemoteViewBackgroundImage	设置远端用户通话界面背景图。
setLayoutMode	设置通话界面布局模式。
setCameraDefaultState	设置摄像头是否默认打开。
destroyed	销毁 TUICallKit 组件实例。
getTUICallEngineInstance	获取 TUICallEngine 实例。

< TUICallKit /> 组件

属性概览

属性	描述	类型	是否必填	默认值	支持 vue	支持 react
allowedMinimized	是否允许悬浮窗	boolean	否	false	√	√

allowedFullScreen	是否允许通话界面全屏	boolean	否	true	√	√
videoDisplayMode	通话界面显示模式	VideoDisplayMode	否	VideoDisplayMode.COVER	√	√
videoResolution	通话分辨率	VideoResolution	否	VideoResolution.RESOLUTION_480P	√	√
beforeCalling	拨打电话前与收到通话邀请前会执行此函数	function(type, error)	否	-	√	√
afterCalling	结束通话后会执行此函数	function()	否	-	√	√
onMinimized	组件切换最小化状态时会执行此函数，STATUS 值说明	function(oldStatus, newStatus)	否	-	√	√
kickedOut	组件抛出的事件，当前登录用户被踢出登录时会触发该事件，通话也会自动结束	function()	否	-	√	×
statusChanged	组件抛出的事件，当通话状态发生变化时，会触发该事件，通话状态种类详见，STATUS 值说明	function({oldStatus, newStatus})	否	-	√	×

示例代码

React

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

<TUICallKit
    videoDisplayMode={VideoDisplayMode.CONTAIN}
    videoResolution={VideoResolution.RESOLUTION_1080P}
    beforeCalling={handleBeforeCalling}
    afterCalling={handleAfterCalling}
/>

function handleBeforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] handleBeforeCalling:", type, error);
}
function handleAfterCalling() {
  console.log("[TUICallkit Demo] handleAfterCalling");
}
```

Vue3

```
<template>
   <TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P"
    :beforeCalling="beforeCalling"
    :afterCalling="afterCalling"
    :onMinimized="onMinimized"
```

```
    :kickedOut="handleKickedOut"
    :statusChanged="handleStatusChanged"
  />
</template>

<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS } from
"@tencentcloud/call-uikit-vue";

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

Vue2.7

```
<template>
   <TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P"
    :beforeCalling="beforeCalling"
    :afterCalling="afterCalling"
    :onMinimized="onMinimized"
    :kickedOut="handleKickedOut"
    :statusChanged="handleStatusChanged"
  />
</template>

<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS } from
"@tencentcloud/call-uikit-vue2";

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
```

```
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

Vue2.6

```
<template>
   <TUICallKit
    :allowedMinimized="true"
    :allowedFullScreen="true"
    :videoDisplayMode="VideoDisplayMode.CONTAIN"
    :videoResolution="VideoResolution.RESOLUTION_1080P"
    :beforeCalling="beforeCalling"
    :afterCalling="afterCalling"
    :onMinimized="onMinimized"
    :kickedOut="handleKickedOut"
    :statusChanged="handleStatusChanged"
  />
</template>

<script lang="ts" setup>
import { TUICallKit, TUICallKitServer, VideoDisplayMode, VideoResolution, STATUS } from
"@tencentcloud/call-uikit-vue2.6";

function beforeCalling(type: string, error: any) {
  console.log("[TUICallkit Demo] beforeCalling:", type, error);
}
function afterCalling() {
  console.log("[TUICallkit Demo] afterCalling");
}
function onMinimized(oldStatus: string, newStatus: string) {
  console.log("[TUICallkit Demo] onMinimized: " + oldStatus + " -> " + newStatus);
}
function kickedOut() {
  console.log("[TUICallkit Demo] kickedOut");
}
function statusChanged(args: { oldStatus: string; newStatus: string; }) {
  const { oldStatus, newStatus } = args;
  if (newStatus === STATUS.CALLING_C2C_VIDEO) {
    console.log(`[TUICallkit Demo] statusChanged: ${oldStatus} -> ${newStatus}`);
  }
}
</script>
```

TUICallKitServer API 详情

React

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

Vue3

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

Vue2.7

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

Vue2.6

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

init

初始化 TUICallKit。

```
try {
  await TUICallKitServer.init({ SDKAppID, userID, userSig });
  // If you already have a tim instance in your project, you need to pass it in here.
  // await TUICallKitServer.init({ tim, SDKAppID, userID, userSig});
  console.log("[TUICallKit] init succeeds.");
} catch (error: any) {
  console.error(`[TUICallKit] init failed. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
SDKAppID	Number	是	应用的 SDKAppID，您可以在实时音视频控制台中找到您的 SDKAppID。具体详见：开通服务
userID	String	是	即用户名，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。 **注意：**不支持同一个 userID 在两台不同的设备上同时进入同一个房间通话，否则会相互干扰。
userSig	String	是	使用 SecretKey 对 SDKAppID、UserID 等信息进行加密，就可以得到 userSig。 它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务，获取方式请参见 UserSig
tim	TencentCloudChat	否	tim 是 TencentCloudChat SDK 的实例

calls

发起单人或多人通话。

> **说明：**
> v4.0.0+ 支持。

```
try {
  await TUICallKitServer.calls({
    userIDList: ['jack', 'tom'],
```

```
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userIDList	Array<String>	是	被呼叫的用户列表
type	TUICallType	是	通话的媒体类型，参数值说明参见 TUICallType 通话类型 兼容旧版本：语音通话(type = 1)、视频通话(type = 2)
chatGroupID	String	是	与 chat 结合使用时， chat 群组的群 ID
roomID	Number	否	数字房间号，范围 [1, 2147483647]
strRoomID	String	否	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，例如数字 123 和字符串 "123" 是两个完全不同的房间。
timeout	Number	否	通话超时时间，默认：30s，单位：秒。设置范围：10s ~ 600s。如果传入0，默认30s。
userData	String	否	发起通话时自定义扩展字段，被呼叫用户在 ON_CALL_RECEIVED 事件中有该参数
offlinePushInfo	Object	否	自定义离线消息推送参数

join

加入群组中已有的音视频通话。

> **说明：**
> **v4.0.0+ 支持。**

```
try {
  await TUICallKitServer.join({callId: xx});
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the join API. Reason: ${error}`);
}
```

参数列表：

参数	类型	是否必填	含义
callId	string	是	此次通话的唯一 ID。

setLanguage

设置语言，目前支持：中文、英文、日文。

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

参数如下表所示：

参数	类型	是否必填	含义
lang	String	是	语言类型 `en` 、 `zh-cn` 和 `ja_JP` 。

setSelfInfo

设置当前用户昵称和头像。

> ⚠ **注意：**
> v2.2.0+ 支持。通话中使用该接口修改用户信息，UI 不会立即更新，需要等到下次通话才能看到变化。

```
try {
  await TUICallKitServer.setSelfInfo({ nickName: "xxx", avatar: "http://xxx" });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the setSelfInfo API. Reason: ${error}`;
}
```

参数如下表所示：

参数	类型	是否必填	含义
nickName	String	是	自己的昵称
avatar	String	是	自己头像地址

setCallingBell

- 自定义用户的来电铃音。
- 仅限传入本地 MP3 格式的文件地址，需要确保该文件目录是应用可以访问的。
- 使用 import 方式引入铃声文件。
- 如需恢复默认铃声， `filePath` 传空即可。

```
try {
  await TUICallKitServer.setCallingBell(filePath?: string)
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the setCallingBell API. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
filePath	String	是	铃声文件地址

enableFloatWindow

开启/关闭悬浮窗功能。默认为 false，通话界面左上角的悬浮窗按钮隐藏，设置为 true 后显示。

```
try {
  await TUICallKitServer.enableFloatWindow(enable: Boolean)
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableFloatWindow API. Reason: ${error}`);
}
```

enableMuteMode

开启/关闭来电铃声。开启后，收到通话请求时，不会播放来电铃声。

```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean)
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
enable	Boolean	是	开启/关闭来电铃声。默认 false。

enableVirtualBackground

开启/关闭模糊背景功能。如果想设置图片背景模糊参见 Web 。通过调用接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

> ⚠ 注意：
> v3.2.4+ 支持。

```
import { TUICallKitServer } from "@tencentcloud/call-uikit-react";
const enable = true;
TUICallKitServer.enableVirtualBackground(enable);
```

参数列表：

参数	类型	是否必填	含义
enable	boolean	是	• enable = true 显示模糊背景按钮 • enable = false 不显示模糊背景按钮

destroyed

销毁 TUICallKit 实例。
该方法不会自动退出 `tim` ，需要手动退出 `tim.logout();` 。

```
try {
  await TUICallKitServer.destroyed();
} catch (error: any) {
  alert(`[TUICallKit] Failed to call the destroyed API. Reason: ${error}`);
}
```

hideFeatureButton

隐藏功能按钮，目前仅支持 摄像头，麦克风和切换前后置按钮。

> ⚠ 注意：
> v3.2.9+ 支持。

Vue

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

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

React

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

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

参数列表：

参数	类型	是否必填	含义
buttonName	FeatureButton	是	按钮名称

setLocalViewBackgroundImage

设置本地用户通话背景。

> ⚠ 注意：
> v3.2.9+ 支持。

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

参数列表：

参数	类型	是否必填	含义
url	string	是	图片地址（支持本地路径和网络地址）

setRemoteViewBackgroundImage

设置远端用户通话背景。

> ⚠ 注意：
> v3.2.9+ 支持。

```
const userId = 'xxx';
const url = 'http://xxx.png';
TUICallKitServer.setRemoteViewBackgroundImage(userId, url);
```

参数列表：

参数	类型	是否必填	含义
userId	string	是	远端用户 userId，设置为 '*' 表示对所有远端用户生效
url	string	是	图片地址（支持本地路径和网络地址）

setLayoutMode

设置通话界面布局模式。

> ⚠ 注意：
> v3.3.0+ 支持。

Vue

```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-vue";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

React

```
import { TUICallKitServer, LayoutMode } from "@tencentcloud/call-uikit-react";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

参数列表：

参数	类型	是否必填	含义
layoutMode	LayoutMode	是	用户流的布局模式

setCameraDefaultState

设置摄像头是否默认打开。

> ⚠ **注意：**
> **v3.3.0+ 支持。**

```
TUICallKitServer.setCameraDefaultState(true);
```

参数列表：

参数	类型	是否必填	含义
isOpen	boolean	是	是否开启摄像头

getTUICallEngineInstance

获取 TUICallEngine 实例。

> ⚠ **注意：**
> **v2.4.3+ 支持。**

```
TUICallKitServer.getTUICallEngineInstance();
```

TUICallKit 类型定义

videoDisplayMode

画面显示模式 `videoDisplayMode` **属性有三个值：**

- `VideoDisplayMode.CONTAIN`
- `VideoDisplayMode.COVER`
- `VideoDisplayMode.FILL`，**默认值是** `VideoDisplayMode.COVER` 。

属性	值	描述
videoDisplayMode	VideoDisplayMode.CONTAIN	• 优先保证视频内容全部显示。 • 视频尺寸等比缩放，直至视频窗口的一边与视窗边框对齐。 • 如果视频尺寸与显示视窗尺寸不一致，在保持长宽比的前提下，将视频进行缩放后填满视窗，缩放后的视频四周会有一圈黑边。

	VideoDisplayMode.COVER	• 优先保证视窗被填满。 • 视频尺寸等比缩放，直至整个视窗被视频填满。 • 如果视频长宽与显示窗口不同，则视频流会按照显示视窗的比例进行周边裁剪或图像拉伸后填满视窗。
	VideoDisplayMode.FILL	• 保证视窗被填满的同时保证视频内容全部显示，但是不保证视频尺寸比例不变。 • 视频的宽高会被拉伸至和视窗尺寸一致。

videoResolution

分辨率 `videoResolution` 有三个值：

- `VideoResolution.RESOLUTION_480P`
- `VideoResolution.RESOLUTION_720P`
- `VideoResolution.RESOLUTION_1080P`，**默认值是** `VideoResolution.RESOLUTION_480P` 。

分辨率说明：

视频 Profile	分辨率（宽 × 高）	帧率（fps）	码率（kbps）
480p	640 × 480	15	900
720p	1280 × 720	15	1500
1080p	1920 × 1080	15	2000

常见问题：

- iOS 13&14不支持编码高于 720P 的视频，建议在这两个系统版本限制最高采集 720P。参见 iOS Safari 已知问题 case 12。
- Firefox 不支持自定义视频帧率（默认为 30fps）。
- 受系统性能占用，摄像头采集能力和浏览器限制等因素的影响，视频分辨率、帧率、码率的实际值不一定能够完全匹配设定值，在这种情况下，浏览器会自动调整 Profile 尽可能匹配设定值。

STATUS

属性值	描述
STATUS.IDLE	闲置状态
STATUS.BE_INVITED	收到通话邀请
STATUS.DIALING_C2C	正在 1v1 呼叫
STATUS.DIALING_GROUP	正在群组呼叫
STATUS.CALLING_C2C_AUDIO	正在 1v1 语音通话
STATUS.CALLING_C2C_VIDEO	正在 1v1 视频通话
STATUS.CALLING_GROUP_AUDIO	正在群组语音通话
STATUS.CALLING_GROUP_VIDEO	正在群组视频通话

TUICallType

TUICallType 类型	描述
TUICallType.AUDIO_CALL	语音通话
TUICallType.VIDEO_CALL	视频通话

offlinePushInfo

参数	类型	是否必填	含义
offlinePushInfo.title	String	否	离线推送标题（选填）
offlinePushInfo.description	String	否	离线推送内容（选填）
offlinePushInfo.androidOPPOChannelID	String	否	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID（选填）
offlinePushInfo.extension	String	否	离线推送透传内容，可以用于设置 Android Notification 模式 和 VoIP 模式。**默认：**Notification 模式，会是系统的通知；VoIP 模式是需要传字段。
offlinePushInfo.ignoreIOSBadge	Boolean	否	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加 **v3.3.0+ 支持**
offlinePushInfo.iOSSound	String	否	离线推送声音设置（仅对 iOS 生效）。**v3.3.0+ 支持**
offlinePushInfo.androidSound	String	否	离线推送声音设置（仅对 Android 生效）。**v3.3.0+ 支持**
offlinePushInfo.androidVIVOClassification	Number	否	VIVO 推送消息分类。**v3.3.0+ 支持**
offlinePushInfo.androidXiaoMiChannelID	String	否	小米手机 8.0 系统及以上的渠道 ID。**v3.3.0+ 支持**
offlinePushInfo.androidFCMChannelID	String	否	FCM 通道手机 8.0 系统及以上的渠道 ID。**v3.3.0+ 支持**
offlinePushInfo.androidHuaWeiCategory	String	否	华为推送消息分类。**v3.3.0+ 支持**
offlinePushInfo.isDisablePush	Boolean	否	是否关闭推送（默认开启推送）。**v3.3.0+ 支持**
offlinePushInfo.iOSPushType	Number	否	iOS 离线推送类型。0-普通推送；1-VoIP推送。默认：0。**v3.3.0+ 支持**

Android Notification 模式

```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, VoIP; 1, notification
  }
};
offlinePushInfo.extension = JSON.stringify(extension);
```

Android VoIP 模式

```
const extension = {
  timPushFeatures: {
    fcmPushType: 0, // 0, data; 1, notification
    fcmNotificationType: 1, // 0, TIMPush implementation;  1, business implementation after transparent transmission
  }
};
offlinePushInfo.extension = JSON.stringify(extension);
```

FeatureButton

FeatureButton 类型	描述
FeatureButton.Camera	摄像头按钮
FeatureButton.Microphone	麦克风按钮
FeatureButton.SwitchCamera	切换前后置摄像头按钮
FeatureButton.InviteUser	邀请他人按钮

LayoutMode

LayoutMode 类型	描述
LayoutMode.LocalInLargeView	本地用户在大窗显示
LayoutMode.RemoteInLargeView	远端用户在大窗显示

废弃接口

<TUICallKitMini/>

`<TUICallKitMini/>`：TUICallKit UI 通话悬浮窗，当 `<TUICallKit :allowedMinimized="true"/>` 时，`<TUICallKitMini/>` 必须被放置在页面中。

说明：

- `<TUICallKitMini/>` UI 通话悬浮窗，≥ v2.3.2 废弃，不再提供该组件。
- 由 `<TUICallKit/>` UI 通话组件主体，实现悬浮窗样式，减少使用复杂度。

属性	描述	类型	是否必填	默认值	支持 vue	支持 react
@kicked-out	注意：≥ v2.3.2 废弃 组件抛出的事件，当前登录用户被踢出登录时会触发该事件，通话也会自动结束	function()	否	-	√	×
@status-changed	注意：≥ v2.3.2 废弃 组件抛出的事件，当通话状态发生变化时，会触发该事件，通话状态种类详见，STATUS 值说明	function({oldStatus, newStatus})	否	-	√	×

call

拨打 1v1 通话，**建议使用 calls 接口。**

```
import { TUICallKitServer, TUICallType } from '@tencentcloud/call-uikit-react';
try {
  await TUICallKitServer.call({
    userID: 'mike', // username being called
    type: TUICallType.VIDEO_CALL,
  });
```

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

参数如下表所示：

参数	类型	是否必填	含义
userID	String	是	被呼叫用户的用户名
type	TUICallType	是	通话的媒体类型，参数值说明参见 TUICallType 通话类型 兼容旧版本：语音通话(type = 1)、视频通话(type = 2)
roomID	Number	否	数字房间号，范围 [1, 2147483647]
strRoomID	String	否	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，例如数字 123 和字符串 "123" 是两个完全不同的房间。
timeout	Number	否	通话超时时间，默认：30s，单位：秒。timeout = 0，设置为不超时
userData	String	否	发起通话时自定义扩展字段，被呼叫用户在 ON_CALL_RECEIVED 事件中有该参数
offlinePushInfo	Object	否	自定义离线消息推送参数

groupCall

发起群组通话，建议使用 calls 接口。

```
try {
  await TUICallKitServer.groupCall({
    userIDList: ['jack', 'tom'],
    groupID: "xxx",
    type: TUICallType.VIDEO_CALL
  });
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the groupCall API. Reason:${error}`);
}
```

参数如下表所示：

参数	类型	是否必填	含义
userIDList	Array<String>	是	被呼叫的用户列表
type	TUICallType	是	通话的媒体类型，参数值说明参见 TUICallType 通话类型 兼容旧版本：语音通话(type = 1)、视频通话(type = 2)
groupID	String	是	通话群组 ID，groupID 的创建可参见 chat-createGroup API
roomID	Number	否	数字房间号，范围 [1, 2147483647]
strRoomID	String	否	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，例如数字 123 和字符串 "123" 是两个完全不同的房间。

版权所有：腾讯云计算（北京）有限责任公司

timeout	Number	否	通话超时时间，默认：30s，单位：秒。timeout = 0，设置为不超时
userData	String	否	发起通话时自定义扩展字段，被呼叫用户在 ON_CALL_RECEIVED 事件中有该参数
offlinePushInfo	Object	否	自定义离线消息推送参数

joinInGroupCall

加入群组中已有的音视频通话，建议使用 join 接口。

> **注意：**
> **v3.1.2+ 支持。**

> **说明：**
> - **加入群组中已有的音视频通话前，需要提前创建或加入IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。**
> - **群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。**

```
try {
  const params = {
    type: TUICallType.VIDEO_CALL,
    groupID: "xxx",
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

参数列表：

参数	类型	是否必填	含义
type	TUICallType	是	通话的媒体类型，例如视频通话、语音通话
groupID	String	是	此次群组通话的群 ID
roomID	number	是	此次通话的音视频房间 ID

TUICallEngine

最近更新时间：2025-04-01 09:57:33

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

说明：

- 为提供更优质的音视频通信能力，原 tuicall-engine-webrtc 包已正式停止维护，建议开发者迁移至 @tencentcloud/call-engine-js 新包。
- 新包对接口进行了优化调整，部分旧版接口不再兼容，同步提供更简洁、稳定的新接口方案。

API 概览

API	描述
createInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
on	监听事件。
off	取消监听事件。
login	登录接口。
logout	登出接口。
setSelfInfo	设置用户昵称和头像。
calls	发起单人或多人通话。
join	主动加入通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
startRemoteView	启动远端画面渲染。
stopRemoteView	停止远端画面渲染。
openCamera	开启摄像头。
closeCamara	关闭摄像头。
switchCamera	切换前后置摄像头，仅移动端支持。
openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
setVideoQuality	设置视频分辨率。
getDeviceList	获取设备列表。
switchDevice	切换摄像头或麦克风设备。
enableAIVoice	开启/关闭 AI 降噪。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式。
setBlurBackground	开关/设置背景虚化。

版权所有：腾讯云计算（北京）有限责任公司

setVirtualBackground	开关/设置图片背景虚化。

废弃接口

API	描述
call	拨打电话（1v1通话），建议使用 calls 接口。
groupCall	发起群组通话，建议使用 calls 接口。
startLocalView	启动本地画面渲染，建议使用 openCamera。
stopLocalView	停止本地画面渲染，建议使用 closeCamera。
joinInGroupCall	主动加入此次群组通话，建议使用 join 接口。
switchCallMediaType	通话类型切换。

TUICallEvent

最近更新时间：2025-06-30 15:14:22

TUICallEvent API 简介

TUICallEvent API 是音视频通话组件的**事件接口**。

事件列表

EVENT	描述
TUICallEvent.ERROR	通话过程中错误事件
TUICallEvent.SDK_READY	SDK 进入 ready 状态时收到该事件
TUICallEvent.KICKED_OUT	用户重复登录，被踢下线
TUICallEvent.USER_ACCEPT	xxxx 用户接通的事件
TUICallEvent.USER_ENTER	xxxx 用户加入通话的事件
TUICallEvent.USER_LEAVE	xxxx 用户离开通话的事件
TUICallEvent.REJECT	xxxx 用户拒绝通话的事件
TUICallEvent.NO_RESP	xxxx 用户不响应的事件
TUICallEvent.LINE_BUSY	xxxx 用户忙线的事件
TUICallEvent.USER_VIDEO_AVAILABLE	xxxx 用户是否有视频流的事件
TUICallEvent.USER_AUDIO_AVAILABLE	xxxx 用户是否有音频流的事件
TUICallEvent.USER_VOICE_VOLUME	所有用户音量大小的反馈事件
TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE	群聊更新邀请列表收到该事件
TUICallEvent.ON_CALL_BEGIN	通话接通的事件，**v1.4.6+ 支持**
TUICallEvent.INVITED	通话请求的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_RECEIVED**
TUICallEvent.ON_CALL_RECEIVED	通话请求的事件，**v1.4.6+ 支持**
TUICallEvent.CALLING_CANCEL	通话取消的事件， **后续废弃，建议使用 TUICallEvent.ON_CALL_CANCELED**
TUICallEvent.ON_CALL_CANCELED	通话取消的事件，**v1.4.6+ 支持**
TUICallEvent.ON_CALL_BEGIN	通话接通时抛出的事件
TUICallEvent.CALLING_END	通话结束的事件
TUICallEvent.DEVICED_UPDATED	设备列表更新的事件
TUICallEvent.CALL_TYPE_CHANGED	通话的媒体类型发生改变的回调
TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件，**v3.0.7+ 支持**

ERROR

通话过程中错误事件，可以通过监听该事件，捕获通话过程中的内部错误。

```
let onError = function(error) {
  console.log(error.code, error.msg);
};
tuiCallEngine.on(TUICallEvent.ERROR, onError);
```

参数如下表所示：

参数	类型	含义
code	Number	错误码
msg	String	错误信息

SDK_READY

TUICallEngine 依赖 @tencentcloud/chat SDK，登录成功后才会触发 SDK_READY 事件，然后才能使用 SDK 的各项功能。

```
let onSDKReady = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.SDK_READY, onSDKReady);
```

KICKED_OUT

当前用户被踢下线，此时可以 UI 提示用户，并再次重新调用登录。

```
let handleOnKickedOut = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.KICKED_OUT, handleOnKickedOut);
```

USER_ACCEPT

如果有用户接通，其他用户都会抛出该事件，userID 就是接通的用户名。

1. 1v1 通话下：被叫接通，主叫会抛出该事件。
2. 群组通话下：A 呼叫 B/C，B 接通，A/C 都会抛出该事件，事件的 userID 都是 B。同理 C 接通，A、B 都会抛出该事件，事件的 userID 都是 C。

```
let handleUserAccept = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, handleUserAccept);
```

参数如下表所示：

参数	类型	含义
userID	String	接通通话的用户 ID

USER_ENTER

如果有用户进入通话，其他用户都会抛出该事件，userID 就是进入通话的用户名。

```
let handleUserEnter = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_ENTER, handleUserEnter);
```

参数如下表所示：

参数	类型	含义
userID	String	进入通话的用户 ID

USER_LEAVE

有用户离开此次通话时，**通话中**的其他用户都会抛出该事件，userID 就是离开通话的用户名。

```
let handleUserLeave = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.USER_LEAVE, handleUserLeave);
```

参数如下表所示：

参数	类型	含义
userID	String	离开当前通话的用户 ID

REJECT

通话被拒绝时抛出该事件

1. 在 1v1 通话中，只有主叫方会收到拒绝事件，userID 就是被叫用户名。
2. 在群组通话中，当有被邀请者拒绝通话时，群组通话的其他人都会抛出该事件，userID 就是拒绝通话的用户名。

```
let handleInviteeReject = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.REJECT, handleInviteeReject);
```

参数如下表所示：

参数	类型	含义
userID	String	拒绝通话的用户 ID

NO_RESP

被呼叫者没有回应时，其他通话用户都会抛出该事件。

- 在 1v1 通话中，只有主叫方会抛出无人应答的事件。例如 A 邀请 B，B 不应答，A 抛出该事件。
- 在群组通话中，当有被邀请者不回应时，群组通话的其他人都会抛出该事件。例如 A 邀请 B、C 进入通话，B 不应答，A、C 均会抛出该事件。

```
let handleNoResponse = function(event) {
console.log(event.sponsor, event.userIDList);
};
tuiCallEngine.on(TUICallEvent.NO_RESP, handleNoResponse);
```

参数如下表所示：

参数	类型	含义
sponsor	String	发起方的用户 ID
userIDList	Array<String>	无响应的用户 ID 列表

LINE_BUSY

通话忙线事件。例如：B 正在通话时，此时 A 呼叫 B，A 会抛出事件。

```
let handleLineBusy = function(event) {
  console.log(event);
};
tuiCallEngine.on(TUICallEvent.LINE_BUSY, handleLineBusy);
```

参数如下表所示：

参数	类型	含义
userID	String	忙线用户 ID

USER_VIDEO_AVAILABLE

视频通话中一个用户开启/关闭了摄像头，通话的其他用户都会抛出该事件。例如：A 和 B 在进行视频通话，A 开/关摄像头，B 会抛出该事件。

```
let handleUserVideoChange = function(event) {
  console.log(event.userID, event.isVideoAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_VIDEO_AVAILABLE, handleUserVideoChange);
```

参数如下表所示：

参数	类型	含义
userID	String	开/关摄像头的用户 ID
isVideoAvailable	Boolean	true 用户打开摄像头；false 用户关闭摄像头

USER_AUDIO_AVAILABLE

音视频通话中一个用户开启/关闭了麦克风，通话的其他用户都会抛出该事件。例如：A 和 B 在进行音视频通话，A 开/关麦克风，B 会抛出该事件。

```
let handleUserAudioChange = function(event) {
  console.log(event.userID, event.isAudioAvailable);
};
tuiCallEngine.on(TUICallEvent.USER_AUDIO_AVAILABLE, handleUserAudioChange);
```

参数如下表所示：

参数	类型	含义
userID	String	开/关麦克风的用户 ID
isAudioAvailable	Boolean	true 用户打开麦克风；false 用户关闭麦克风

USER_VOICE_VOLUME

音视频通话中用户音量变化时，通话的其他用户都会抛出该事件。例如：A 和 B 在进行音视频通话，A 音量变化，B 会抛出该事件。

```
let handleUserVoiceVolumeChange = function(event) {
  console.log(event.volumeMap);
};
tuiCallEngine.on(TUICallEvent.USER_VOICE_VOLUME, handleUserVoiceVolumeChange);
```

参数如下表所示：

参数	类型	含义
volumeMap	Array<Object>	音量表，根据每个 userid 可以获取对应的音量大小，音量范围：[0, 100]

GROUP_CALL_INVITEE_LIST_UPDATE

群聊更新邀请列表收到该事件。

```
let handleGroupInviteeListUpdate = function(event) {
  console.log(event.userIDList);
};
tuiCallEngine.on(TUICallEvent.GROUP_CALL_INVITEE_LIST_UPDATE, handleGroupInviteeListUpdate);
```

参数如下表所示：

参数	类型	含义
userIDList	Array<String>	群组更新邀请列表

INVITED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

> ⚠ 注意：
> 计划后续版本废弃，建议使用：ON_CALL_RECEIVED 。

ON_CALL_RECEIVED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

> ⚠ 注意：
> v1.4.6+ 版本支持。

```
let handleOnCallReceived = function(event) {
    console.log(event);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_RECEIVED, handleOnCallReceived);
```

参数如下表所示：

参数	类型	含义
callerId	String	邀请者
calleeIdList	Array<String>	同时还被邀请的人
isFromGroup	Boolean	是否是群组通话
inviteData	Object	呼叫数据
inviteID	String	邀请 ID，标识一次邀请
userData	String	扩展字段：用于在邀请信令中增加扩展信息
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话

callRole	String	角色，枚举类型：主叫、被叫

CALLING_CANCEL

如果通话未建立，都会抛出该事件。您可以通过监听该事件来实现类似未接来电、重置 UI 状态等显示逻辑。通话未建立的场景如下：

> ⚠ **注意：**
> **计划后续版本废弃，建议使用：** ON_CALL_CANCELED 。

ON_CALL_CANCELED

如果通话未建立，都会抛出该事件。您可以通过监听该事件来实现类似未接来电、重置 UI 状态等显示逻辑。通话未建立的场景如下：

- 主叫取消：主叫抛出该事件，userID 为主叫；被叫也会抛出该事件，userID 为被叫；
- 被叫超时：主叫会同时抛出 NO_RESP 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；
- 被叫拒接：主叫会同时抛出 REJECT 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；
- 被叫忙线：主叫会同时抛出 LINE_BUSY 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

> ⚠ **注意：**
> **v1.4.6+ 版本支持** 。

```
let handleOnCallCanceled = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_CANCELED, handleOnCallCanceled);
```

参数如下表所示：

参数	类型	含义
userID	String	通话取消的用户 ID
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫、被叫

ON_CALL_BEGIN

表示通话接通，主叫和被叫都会抛出该事件，您可以通过监听这个事件来开启云端录制、内容审核等流程。

> ⚠ **注意：**
> **v1.4.6+ 版本支持。**

```
let handleOnCallBegin = function(event) {
    console.log(event);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_BEGIN, handleOnCallBegin);
```

参数如下表所示：

参数	类型	含义

callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，类型：主叫、被叫

CALLING_END

通话结束时，主叫和被叫都会抛出该事件。您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
let handleCallingEnd = function(event) {
  console.log(event.userID, event.);
};
tuiCallEngine.on(TUICallEvent.CALLING_END, handleCallingEnd);
```

参数如下表所示：

参数	类型	含义
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫（'inviter'）、被叫（'invitee'）、未知（''）
totalTime	Number	此次通话的时长，单位: 秒
userID	String	通话结束的 userID。
callId	String	本次通话的唯一 ID。**v1.4.6+ 版本支持**
callEnd	Number	此次通话的时长（**后续将废弃，请使用 totalTime**），单位: 秒

DEVICED_UPDATED

设备列表更新时抛出该事件。

```
let handleDeviceUpdated = function({ microphoneList, cameraList, currentMicrophoneID, currentCameraID
}) {
  console.log(microphoneList, cameraList, currentMicrophoneID, currentCameraID);
};
tuiCallEngine.on(TUICallEvent.DEVICED_UPDATED, handleDeviceUpdated);
```

CALL_TYPE_CHANGED

通话类型切换时抛出该事件。

```
let handleCallTypeChanged = function({ oldCallType, newCallType }) {
  console.log(oldCallType, newCallType);
};
tuiCallEngine.on(TUICallEvent.CALL_TYPE_CHANGED, handleDeviceUpdated);
```

参数如下表所示：

参数	类型	含义
oldCallType	Number	旧通话类型
newCallType	Number	新通话类型

ON_USER_NETWORK_QUALITY_CHANGED

所有用户网络质量的事件

> ⚠ **注意：**
> v3.0.7+ 支持。

```
let handleOnUserNetworkQualityChange = function(event) {
  console.log(event.networkQualityList);
};
tuiCallEngine.on(TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED, handleOnUserNetworkQualityChange);
```

参数如下表所示：

参数	类型	含义
networkQualityList	Array<Object>	网络状态，根据 userID 可以获取对应用户当前的网络质量（仅表示本地的上下行）。例如：`networkQualityList: [{ userId: quality }]` 。 **网络质量说明：** • quality = 0，网络状况未知 • quality = 1，网络状况极佳 • quality = 2，网络状况较好 • quality = 3，网络状况一般 • quality = 4，网络状况差 • quality = 5，网络状况极差 • quality = 6，网络链接已断开

小程序
API 概览

最近更新时间：2025-06-25 10:09:12

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
init	初始化 TUICallKit。
calls	发起单人或多人通话。
join	主动加入通话。
setSelfInfo	设置用户的头像、昵称。
setCallingBell	设置自定义来电铃声。
setLogLevel	设置日志级别。
enableFloatWindow	开启/关闭悬浮窗功能。
enableMuteMode	开启/关闭来电铃声。
hideFeatureButton	隐藏按钮。
setLocalViewBackgroundImage	设置本地用户通话界面背景图。
setRemoteViewBackgroundImage	设置远端用户通话界面背景图。
setLayoutMode	设置通话界面布局模式。
setCameraDefaultState	设置摄像头是否默认打开。
destroyed	销毁 TUICallKit。
getTUICallEngineInstance	获取 TUICallEngine 实例。

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
createInstance	创建 TUICallEngine 实例（静态方法）。
destroyInstance	销毁 TUICallEngine 实例（静态方法）。
on	增加事件监听
off	取消事件监听
calls	发起单人或多人通话。
join	主动加入通话。
accept	接听通话
reject	拒绝通话
hangup	结束通话

版权所有：腾讯云计算（北京）有限责任公司

openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）
setSelfInfo	设置用户的头像、昵称

TUICallEvent

TUICallEvent 是 TUICallEngine 对应的事件类，您可以通过监听如下事件，来监听自己感兴趣的事件。

事件名	说明
ON_CALL_RECEIVED	通话请求的事件
USER_ACCEPT	用户接听
USER_ENTER	用户进入通话
USER_LEAVE	用户离开通话
USER_UPDATE	用户更新
REJECT	用户拒绝通话
NO_RESP	用户无响应
LINE_BUSY	用户忙线
ON_CALL_CANCELED	通话未建立时，通话的各端都会抛出该事件
ON_CALL_BEGIN	通话接通时抛出的事件
CALL_END	通话结束
SDK_READY	SDK Ready 回调
KICKED_OUT	被踢下线
CALL_MODE	切换通话模式
ERROR	错误信息
ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件

关键常量定义

CALL_STATUS

参数	说明
CALL_STATUS.IDLE	默认
CALL_STATUS.CALLING	呼叫中/被呼叫中
CALL_STATUS.CONNECTED	接通中

CallMediaType

参数	说明
CallMediaType.AUDIO	音频
CallMediaType.VIDEO	视频

CALL_MEDIA_TYPE

参数	说明
CALL_MEDIA_TYPE.AUDIO_CALL	音频
CALL_MEDIA_TYPE.VIDEO_CALL	视频

AUDIO_PLAYBACK_DEVICE

参数	说明
AUDIO_PLAYBACK_DEVICE.EAR	听筒
AUDIO_PLAYBACK_DEVICE.SPEAKER	扬声器

offlinePushInfo

参数	类型	是否必填	含义
offlinePushInfo.title	String	否	离线推送标题（选填）
offlinePushInfo.description	String	否	离线推送内容（选填）
offlinePushInfo.androidOPPOChannelID	String	否	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID（选填）
offlinePushInfo.extension	String	否	离线推送透传内容（选填）（tsignaling 版本 ≥ 0.9.0）

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2025-06-30 13:45:02

TUICallKit 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景；更详细的接入步骤，详见 快速接入TUICallKit 。

> **说明：**
> - 为提供更优质的音视频通信能力，原 @tencentcloud/call-uikit-wechat 包已正式停止维护，建议开发者迁移至 @tencentcloud/call-uikit-wx-uniapp、@tencentcloud/call-uikit-wx 新包。
> - 新包对接口进行了优化调整，部分旧版接口不再兼容，同步提供更简洁、稳定的新接口方案。

API 概览

API	描述
<TUICallKit />	TUICallKit 通话 UI 组件。
init	初始化 TUICallKit
calls	发起单人或多人通话。
join	主动加入通话。
setSelfInfo	设置用户的头像、昵称
setCallingBell	设置自定义来电铃声
setLogLevel	设置日志级别
enableFloatWindow	开启/关闭悬浮窗功能
enableMuteMode	开启/关闭来电铃声
hideFeatureButton	隐藏按钮。
setLocalViewBackgroundImage	设置本地用户通话界面背景图。
setRemoteViewBackgroundImage	设置远端用户通话界面背景图。
setLayoutMode	设置通话界面布局模式。
setCameraDefaultState	设置摄像头是否默认打开。
destroyed	销毁 TUICallKit。
getTUICallEngineInstance	获取 TUICallEngine 实例。

API 详情

uni-app 小程序

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

微信小程序

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

init

初始化 TUICallKit。

```
TUICallKitServer.init({
      sdkAppID: 0, // 替换为您自己账号下的 SDKAppId
      userID: 'jane', // 填写当前用的 userID
      userSig: 'xxxxxxxxxxx',
      tim: null, // 如果您不需要 TIM 实例，可忽略
})
```

参数如下表所示：

参数	类型	说明	是否必填
sdkAppID	Number	您可以在**实时音视频控制台** > 应用管理 > **应用信息**中查看 SDKAppID	是
userID	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）	是
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见：如何计算及使用 UserSig	是
tim	ChatSDK	TIM 实例	否

这里详细介绍一下 init 中的几个参数：

- **userSig**：使用步骤三中获取的 SecretKey 对 sdkAppID、userID 等信息进行加密，就可以得到 userSig，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务，更多信息请参见 如何计算及使用 UserSig 。
- **tim** 可以将外部的 tim 实例通过 init 透传给 callkit ，tim 参数适用于业务中已存在 TIM 实例，为保证 TIM 实例唯一性。

calls

发起群组通话。

```
TUICallKitServer.calls({
    userIDList: ["jane", "mike", "tommy"],
    type: 1,
})
```

参数如下表所示：

参数	类型	含义	是否必填
userIDList	Array<String>	目标用户的 userId 列表，示例：["jane", "mike", "tommy"]	是
type	CallMediaType	通话的媒体类型，参数值说明参见 CallMediaType 通话类型	是
chatGroupID	String	与 chat 结合使用时， chat 群组的群ID	是
roomID	Number	数字房间号，范围 [1, 2147483647]	否
userData	String	扩展字段：用于在邀请信令中增加扩展信息	否
timeout	Number	• 通话的超时时间，0为不超时，单位 s(秒)（选填）。设置范围：10s ~ 600s。如果传入0，默认30s。 • 被叫需要在30s内处于登录状态，timeout 值才会有效，具体参见 timeout 字段设置无效 。	否
offlinePushInfo	Object	自定义离线消息推送（选填）	否

join

发起群组通话。

```
TUICallKitServer.join({callId: xx});
```

参数如下表所示：

参数	类型	含义	是否必填
callId	string	此次通话的唯一 ID。	是

setSelfInfo

设置用户头像、昵称。

> ⚠ **注意：**
> **通话中使用该接口修改用户信息，UI 不会立即更新，需要等到下次通话才能看到变化。**

```
TUICallKitServer.setSelfInfo({ nickName: "xxx", avatar: "http://xxx" });
```

参数如下表所示：

参数	类型	含义
nickName	String	设置昵称
avatar	String	头像地址

setCallingBell

设置自定义来电铃声。
这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

- 传入路径应为本地铃声文件的绝对地址。
- 如需恢复默认铃声，`filePath` 传空即可。
- 支持的铃声文件格式：

格式	iOS	Android
m4a	√	√
mp3	√	√
wav	√	√
aac	√	√

```
TUICallKitServer.setCallingBell("filePath")
```

参数如下表所示：

参数	类型	含义
filePath	String	铃声地址

setLogLevel

设置日志级别，低于 level 的日志将不会输出。

```
TUICallKitServer.setLogLevel(level)
```

参数如下表所示：

参数	值	含义
level	0	普通级别，日志量较多，接入时建议使用
	1	release 级别，SDK 输出关键信息，生产环境时建议使用
	2	告警级别，SDK 只输出告警和错误级别的日志
	3	错误级别，SDK 只输出错误级别的日志
	4	无日志级别，SDK 将不打印任何日志

enableFloatWindow

开启/关闭悬浮窗功能。
默认为 `false` ，通话界面左上角的悬浮窗按钮隐藏，设置为 `true` 后显示。

```
TUICallKitServer.enableFloatWindow(enable?: Boolean)
```

enableMuteMode

开启/关闭来电铃声。
开启后，收到通话请求时，不会播放来电铃声。

```
try {
  await TUICallKitServer.enableMuteMode(enable: boolean)
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

hideFeatureButton

隐藏功能按钮，目前仅支持 摄像头，麦克风和切换前后置按钮。

```
TUICallKitServer.hideFeatureButton(buttonName: FeatureButton);
```

参数列表：

参数	类型	是否必填	含义
buttonName	FeatureButton	是	按钮名称

setLocalViewBackgroundImage

设置本地用户通话背景。

```
TUICallKitServer.setLocalViewBackgroundImage(url: string);
```

参数列表：

参数	类型	是否必填	含义
url	string	是	图片地址（支持本地路径和网络地址）

版权所有：腾讯云计算（北京）有限责任公司

			注意： 如果使用网络地址，需要将网络地址添加到 request 合法域名。

setRemoteViewBackgroundImage

设置远端用户通话背景。

```
TUICallKitServer.setRemoteViewBackgroundImage(userId: string, url: string);
```

参数列表：

参数	类型	是否必填	含义
userId	string	是	远端用户 userId，设置为 '*' 表示对所有远端用户生效
url	string	是	图片地址（支持本地路径和网络地址） **注意：** 如果使用网络地址，需要将网络地址添加到 request 合法域名。

setLayoutMode

设置通话界面布局模式。

```
import { TUICallKitServer, LayoutMode } from "../../TUICallKit/src/index";
TUICallKitServer.setLayoutMode(LayoutMode.LocalInLargeView);
```

参数列表：

参数	类型	是否必填	含义
layoutMode	LayoutMode	是	用户流的布局模式

setCameraDefaultState

设置摄像头是否默认打开。

```
TUICallKitServer.setCameraDefaultState(true);
```

参数列表：

参数	类型	是否必填	含义
isOpen	boolean	是	是否开启摄像头

destroyed

销毁 TUICallKit。

```
TUICallKitServer.destroyed()
```

getTUICallEngineInstance

获取 TUICallEngine 实例。

```
TUICallKitServer.getTUICallEngineInstance();
```

TUICallKit 类型定义

offlinePushInfo

参数	类型	是否必填	含义
offlinePushInfo.title	String	否	离线推送标题（选填）
offlinePushInfo.description	String	否	离线推送内容（选填）
offlinePushInfo.androidOPPOChannelID	String	否	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID（选填）
offlinePushInfo.extension	String	否	离线推送透传内容（选填）（tsignaling 版本 ≥ 0.9.0）

FeatureButton

FeatureButton 类型	描述
FeatureButton.Camera	摄像头按钮
FeatureButton.Microphone	麦克风按钮
FeatureButton.SwitchCamera	切换前后置摄像头按钮
FeatureButton.InviteUser	邀请他人按钮

LayoutMode

LayoutMode 类型	描述
LayoutMode.LocalInLargeView	本地用户在大窗显示
LayoutMode.RemoteInLargeView	远端用户在大窗显示

废弃接口

call

拨打电话（1v1 通话）。

```
TUICallKitServer.call({
    userID: "jane",
    type: 1
})
```

参数如下表所示：

参数	类型	含义	是否必填
userID	String	被叫用户的 ID	是
type	Number	通话的媒体类型：1：语音通话　2：视频通话	是
roomID	Number	数字房间号, 范围 [1, 2147483647]	否
strRoomID	String	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。	否

		2. 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，例如数字 123 和字符串 "123" 是两个完全不同的房间。	
userData	String	扩展字段：用于在邀请信令中增加扩展信息	否
timeout	Number	• 通话的超时时间，0为不超时，单位 s(秒)（选填），默认30s • 被叫需要在30s内处于登录状态，timeout 值才会有效，具体参见 timeout 字段设置无效	否
offlinePushInfo	Object	自定义离线消息推送（选填）	否

groupCall

发起群组通话。

```
TUICallKitServer.groupCall({
    userIDList: ["jane", "mike", "tommy"],
    type: 1,
    groupID: "12345678"
})
```

参数如下表所示：

参数	类型	含义	是否必填
userIDList	Array<String>	目标用户的 userId 列表，示例：["jane", "mike", "tommy"]	是
type	CallMediaType	通话的媒体类型，参数值说明参见 CallMediaType 通话类型	是
groupID	String	此次群组通话的群 ID	是
roomID	Number	数字房间号, 范围 [1, 2147483647]	否
strRoomID	String	字符串房间号。**v3.3.1+ 支持** 1. roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。 2. 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，例如数字 123 和字符串 "123" 是两个完全不同的房间。	否
userData	String	扩展字段: 用于在邀请信令中增加扩展信息	否
timeout	Number	• 通话的超时时间，0为不超时，单位 s(秒)（选填），默认30s • 被叫需要在30s内处于登录状态，timeout 值才会有效，具体参见 timeout 字段设置无效	否
offlinePushInfo	Object	自定义离线消息推送（选填）	否

joinInGroupCall

加入群组中已有的音视频通话。

> **说明：**
> • 加入群组中已有的音视频通话前，需要提前创建或加入IM 群组，并且群组中已有用户在通话中，如果已经创建，请忽略。
> • 群组的创建详见：IM 群组管理，或者您也可以直接使用 IM TUIKIT，一站式集成聊天、通话等场景。

```
try {
  const params = {
    type: 2, // 视频通话
```

```
    groupID: "xxx",
    roomID: 0,
  };
  await TUICallKitServer.joinInGroupCall(params);
} catch (error: any) {
  console.error(`[TUICallKit] Failed to call the enableMuteMode API. Reason: ${error}`);
}
```

参数列表：

参数	类型	是否必填	含义
type	CallMediaType	是	通话的媒体类型，参数值说明参见 CallMediaType 通话类型
groupID	string	是	此次群组通话的群 ID
roomID	number	是	此次通话的音视频房间 ID

TUICallEngine

最近更新时间：2025-06-30 12:41:02

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。由于小程序开发的限制，需要先给 live-pusher 绑定事件，具体请参见 TUICallKit 。

> **说明：**
> - 为提供更优质的音视频通信能力，原 tuicall-engine-wx 包已正式停止维护，建议开发者迁移至 @tencentcloud/call-engine-wx 新包。
> - 新包对接口进行了优化调整，部分旧版接口不再兼容，同步提供更简洁、稳定的新接口方案。

API 概览

创建实例和事件回调

API	描述
createInstance	创建 TUICallEngine 实例（静态方法）
destroyInstance	销毁 TUICallEngine 实例（静态方法）
on	增加事件监听
off	取消事件监听

通话操作相关接口函数

API	描述
calls	发起 1v1 通话
inviteUser	在群组通话中，邀请其他人加入
join	主动加入当前的群组通话中
accept	接听通话
reject	拒绝通话
hangup	结束通话

设备控制相关接口函数

API	描述
openCamera	开启摄像头
closeCamera	关闭摄像头
switchCamera	切换前后摄像头
openMicrophone	打开麦克风
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）

其他接口函数

API	描述
setSelfInfo	设置用户的头像、昵称

setLogLevel	设置日志等级
setBeautyLevel	设置美颜级别，支持关闭默认美颜

API 详情

```
import TUICallEngine, { TUICallEvent, CALL_MEDIA_TYPE } from '@tencentcloud/tuicall-engine-wx';
```

createInstance

创建 TUICallEngine 的单例。

```
wx.$TUICallEngine = TUICallEngine.createInstance({
   tim,
   sdkAppID,
});
```

说明

参数	类型	说明	是否必传
sdkAppID	Number	SDKAppId 是腾讯云用于区分客户的唯一标识	是
tim	ChatSDK	TIM 实例	否

destroyInstance

销毁 TUICallEngine 的单例。

```
TUICallEngine.destroyInstance();
```

on

通过 on 方法可以监听 TUICallEngine 所有的事件。

```
let handleNewInvitationReceived = function(event) {
    // 收到邀请
};
wx.$TUICallEngine.on(TUICallEvent.INVITED, handleNewInvitationReceived, this);
```

说明

参数	类型	说明	是否必传
eventName	TUICallEvent	事件名	是
handler	Function	事件的回调函数	是
context	Any	当前执行上下文（this）	是

off

移除事件监听。

```
let handleNewInvitationReceived = function(event) {
    // 收到邀请
};
```

```
wx.$TUICallEngine.off(TUICallEvent.INVITED, handleNewInvitationReceived);
```

说明

参数	类型	说明	是否必传
eventName	TUICallEvent	事件名	是
handler	Function	事件的回调函数	是

login

登录接口。

```
const params = {
    userID: 'john', // 您的用户 ID
    userSig: 'xxxx', // 签名
};
let promise = wx.$TUICallEngine.login(params);
promise.then(() => {
    //success
}).catch(error => {
    console.warn('login error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
userID	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）	是
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参考 如何计算UserSig。	是

logout

登出接口。

```
let promise = wx.$TUICallEngine.logout();
promise.then(() => {
    // logout success
}).catch(error => {
    console.warn('logout error:', error);
});
```

calls

发起单人或多人通话。

> **注意**
> 离线推送仅适用于终端（Android 或 iOS），Web 和 微信小程序不支持。

```
import { MEDIA_TYPE } from 'tuicall-engine-wx';
let promise = wx.$TUICallEngine.groupCall({
    userIDList: ['user1', 'user2'],
    type: CALL_MEDIA_TYPE.AUDIO_CALL,
});
promise.then(() => {
```

```
    // groupCall success
}).catch(error => {
    console.warn('groupCall error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
userIDList	Array<String>	被呼叫者的 userID 列表	是
type	MEDIA_TYPE	通话的媒体类型	是
chatGroupID	String	与 chat 结合使用时， chat 群组的群ID	是
timeout	Number	超时时长，单位：秒	否
roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号	否
userData	String	扩展字段: 用于在邀请信令中增加扩展信息	否

inviteUser

邀请用户加入此次群组通话。
使用场景：一个群组通话中的用户主动邀请其他人时使用。

```
const userIDList = ['jack', 'john'];
const params = {
    userIDList
};
wx.$TUICallEngine.inviteUser(params).then(() => {
    // success
}).catch(error => {
    console.error('inviteUser error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
params	Object	邀请参数	是
params.userIDList	Array<String>	被叫用户的 userID 列表	是

join

主动加入此次群组通话。

```
const params = { callId: xxx };
wx.$TUICallEngine.join(params).then(() => {
    // success
}).catch(error => {
    console.error('join error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传

params	Object	主动加入参数	是
params.callId	String	是	此次通话的唯一 ID。

accept

接受当前通话，当您作为被叫方收到 `EVENT.INVITED` 事件的回调时，可以调用该接口接听来电。

```
wx.$TUICallEngine.on(TUICallEvent.INVITED, () => {
    wx.$TUICallEngine.accept().promise.then(() => {
        // accept success
    }).catch(error => {
        console.warn('accept error:', error);
    });
});
```

reject

拒绝当前通话，当您作为被叫收到 `EVENT.INVITED` 的回调时，可以调用该函数拒绝来电。

```
wx.$TUICallEngine.on(TUICallEvent.INVITED, () => {
    wx.$TUICallEngine.reject().then(() => {
        // reject success
    }).catch(error => {
        console.warn('reject error:', error);
    });
});
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
wx.$TUICallEngine.hangup().then(() => {
     // hangup success
 }).catch(error => {
     console.warn('hangup error:', error);
 });
```

openCamera

您可以调用该函数开启摄像头。

```
wx.$TUICallEngine.openCamera().then(() => {
    // openCamera success
}).catch(error => {
    console.warn('openCamera error:', error);
});
```

closeCamera

您可以调用该函数关闭摄像头，处于通话中的用户会收到回调。

```
wx.$TUICallEngine.closeCamera().then(() => {
    // closeCamera success
}).catch(error => {
    console.warn('closeCamera error:', error);
});
```

版权所有：腾讯云计算（北京）有限责任公司

switchCamera

切换前后摄像头。

```
wx.$TUICallEngine.switchCamera().then(() => {
    // switchCamera success
}).catch(error => {
    console.warn('openMicrophone error:', error);
});
```

openMicrophone

您可以调用该函数打开麦克风，处于通话中的用户会收到回调。

```
wx.$TUICallEngine.openMicrophone().then(() => {
    // openMicrophone success
}).catch(error => {
    console.warn('openMicrophone error:', error);
});
```

closeMicrophone

您可以调用该函数关闭麦克风，处于通话中的用户会收到回调。

```
wx.$TUICallEngine.closeMicrophone().then(() => {
    // closeMicrophone success
}).catch(error => {
    console.warn('closeMicrophone error:', error);
});
```

selectAudioPlaybackDevice

选择音频播放设备，目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。

```
wx.$TUICallEngine.selectAudioPlaybackDevice(AUDIO_PLAYBACK_DEVICE.EAR).then(() => {
    // selectAudioPlaybackDevice success
}).catch(error => {
    console.warn('selectAudioPlaybackDevice error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
type	AUDIO_PLAYBACK_DEVICE	扬声器：AUDIO_PLAYBACK_DEVICE.SPEAKER 听筒：AUDIO_PLAYBACK_DEVICE.EAR	是

setSelfInfo

设置用户头像、昵称。

```
wx.$TUICallEngine.setSelfInfo("昵称", "头像 URL").then(() => {
    // setSelfInfo success
}).catch(error => {
    console.warn('setSelfInfo error:', error);
});
```

参数如下表所示：

参数	类型	含义
nickName	String	设置昵称
avatar	String	头像地址

setLogLevel

设置日志级别，低于 level 的日志将不会输出。

```
wx.$TUICallEngine.setLogLevel(level)
```

参数如下表所示：

参数	值	含义
level	0	普通级别，日志量较多，接入时建议使用
	1	release 级别，SDK 输出关键信息，生产环境时建议使用
	2	告警级别，SDK 只输出告警和错误级别的日志
	3	错误级别，SDK 只输出错误级别的日志
	4	无日志级别，SDK 将不打印任何日志

setBeautyLevel

> **注意：**
> ≥ v2.1.0 支持

设置美颜级别，支持关闭默认美颜。

```
wx.$TUICallEngine.setBeautyLevel(beautyLevel: number);
```

参数如下表所示：

参数	类型	含义
beautyLevel	number	美颜级别，取值范围 0 - 9； 0 表示关闭，9 表示效果最明显

废弃接口

API	描述
call	拨打电话（1v1通话），建议使用 calls 接口。
groupCall	发起群组通话，建议使用 calls 接口。
startLocalView	启动本地画面渲染，建议使用 openCamera。
stopLocalView	停止本地画面渲染，建议使用 closeCamera。
joinInGroupCall	主动加入此次群组通话，建议使用 join 接口。
switchCallMediaType	切换通话媒体类型，比如视频通话切音频通话

call

拨打电话（1v1通话）。

> ⚠ **注意：**
> 离线推送仅适用于终端（Android 或 iOS），Web 和 微信小程序不支持。

```
let promise = wx.$TUICallEngine.call({
    userID: 'user1',
    type: MEDIA_TYPE.AUDIO,
});
promise.then(() => {
    // call success
}).catch(error => {
    console.warn('call error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
userID	String	被呼叫者的 userID	是
type	MEDIA_TYPE	通话的媒体类型 MEDIA_TYPE.AUDIO：语音通话 MEDIA_TYPE.VIDEO：视频通话	是
roomID	Number	数字房间号, 范围 [1, 2147483647]	否
userData	String	扩展字段: 用于在邀请信令中增加扩展信息	否

groupCall

发起群组通话。

> ⚠ **注意：**
> 离线推送仅适用于终端（Android 或 iOS），Web 和 微信小程序不支持。

```
import { MEDIA_TYPE } from 'tuicall-engine-wx';
let promise = wx.$TUICallEngine.groupCall({
    userIDList: ['user1', 'user2'],
    type: MEDIA_TYPE.AUDIO,
    groupID: '123345',
});
promise.then(() => {
    // groupCall success
}).catch(error => {
    console.warn('groupCall error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
userIDList	Array<String>	被呼叫者的 userID 列表	是
type	MEDIA_TYPE	通话的媒体类型 MEDIA_TYPE.AUDIO：语音通话 MEDIA_TYPE.VIDEO：视频通话	是

groupID	String	此次群组通话的群 ID	是
timeout	Number	超时时长，单位：秒	否
roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号	否
userData	String	扩展字段: 用于在邀请信令中增加扩展信息	否

joinInGroupCall

主动加入此次群组通话。
使用场景：群组内用户主动加入此次群组通话使用。

```
import { MEDIA_TYPE } from 'tuicall-engine-wx';
const params = {
    roomID: 123,
    type: MEDIA_TYPE.AUDIO,
    groupID: 111
};
wx.$TUICallEngine.joinInGroupCall(params).then(() => {
    // success
}).catch(error => {
    console.error('joinInGroupCall error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
params	Object	主动加入参数	是
roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号	是
groupID	String	此次群组通话的群 ID	是
type	MEDIA_TYPE	通话的媒体类型 MEDIA_TYPE.AUDIO：语音通话 MEDIA_TYPE.VIDEO：视频通话	是

switchCallMediaType

当前通话类型切换。
仅支持 1v1 通话过程中使用

```
wx.$TUICallEngine.switchCallMediaType(MEDIA_TYPE.VIDEO).then(() => {
  // switchCallMediaType success
}).catch(error => {
  console.warn('switchCallMediaType error:', error);
});
```

参数如下表所示：

参数	类型	含义	是否必传
type	MEDIA_TYPE	通话的媒体类型 MEDIA_TYPE.AUDIO：语音通话 MEDIA_TYPE.VIDEO：视频通话	是

TUICallEvent

最近更新时间：2025-06-25 10:09:12

TUICallEvent API 是音视频通话组件的事件接口。

> **说明：**
> - 为提供更优质的音视频通信能力，原 tuicall-engine-webrtc 包已正式停止维护，建议开发者迁移至 @tencentcloud/tuicall-engine-js 新包。
> - 新包对接口进行了优化调整，部分旧版接口不再兼容，同步提供更简洁、稳定的新接口方案。

TUICallEvent 概览

事件名	说明
ON_CALL_RECEIVED	通话请求的事件
USER_ACCEPT	用户接听
USER_ENTER	用户进入通话
USER_LEAVE	用户离开通话
USER_UPDATE	用户更新
REJECT	用户拒绝通话
NO_RESP	用户无响应
LINE_BUSY	用户忙线
ON_CALL_CANCELED	通话未建立时，通话的各端都会抛出该事件
ON_CALL_BEGIN	通话接通时抛出的事件
CALL_END	通话结束
SDK_READY	SDK Ready 回调
KICKED_OUT	被踢下线
CALL_MODE	切换通话模式
ERROR	错误信息
ON_USER_NETWORK_QUALITY_CHANGED	所有用户网络质量的事件

TUICallEvent 详情

```
import TUICallEngine, { TUICallEvent } from '@tencentcloud/tuicall-engine-wx';
```

ON_CALL_RECEIVED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
let handleOnCallReceived = function(event) {
    console.log(event)
};
tuiCallEngine.on(TUICallEvent.ON_CALL_RECEIVED, handleOnCallReceived, this);
```

参数如下表所示：

参数	类型	含义
sponsor	String	邀请者
userIDList	Array<String>	同时还被邀请的人
isFromGroup	Boolean	是否是群组通话
inviteData	Object	呼叫数据
inviteID	String	邀请 ID，标识一次邀请
userData	String	扩展字段：用于在邀请信令中增加扩展信息
callId	String	本次通话的唯一 ID。
roomID	Number	此次通话的音视频房间 ID。
callMediaType	Number	通话的媒体类型，视频通话、语音通话。
callRole	String	角色，枚举类型：主叫、被叫。

USER_ACCEPT

如果有用户接听，其他用户会收到该事件。

```
let handleUserAccept = function(event) {
    console.log('用户接听')
}
tuiCallEngine.on(TUICallEvent.USER_ACCEPT, this.handleUserAccept, this);
```

参数如下表所示：

参数	类型	含义
userID	String	接听的用户 ID
userList	Array	未接听用户的信息（**后续将废弃**）

USER_ENTER

如果有用户同意进入通话，其他用户会收到该事件。

```
let handleUserEnter = function(event) {
    console.log('用户进入通话')
}
tuiCallEngine.on(TUICallEvent.USER_ENTER, this.handleUserEnter, this);
```

参数如下表所示：

参数	类型	含义
userID	String	进房的用户 ID
playerList	Array	房间中用户的推流信息

USER_LEAVE

有用户离开此次通话时，通话中的其他用户会收到该事件。

版权所有：腾讯云计算（北京）有限责任公司

```
let handleUserLeave = function(event) {
    console.log('用户离开通话')
}
tuiCallEngine.on(TUICallEvent.USER_LEAVE, this.handleUserLeave, this);
```

参数如下表所示：

参数	类型	含义
userID	String	退房的用户 ID
playerList	Array	房间中用户的推流信息

USER_UPDATE

用户更新。

```
let handleUserUpdate = function(event) {
    console.log('用户更新')
}
tuiCallEngine.on(TUICallEvent.USER_UPDATE, this.handleUserUpdate, this)
```

参数如下表所示：

参数	类型	含义
pusher	String	更新后的自身推流信息
playerList	Array	更新后其他用户的推流信息

REJECT

通话被拒绝事件，在 1v1 通话中，只有主叫方会收到拒绝事件，在群组通话中，所有被邀请者都可以收到该事件。

```
let handleInviteeReject = function(event) {
    console.log('用户拒绝通话')
}
tuiCallEngine.on(TUICallEvent.REJECT, this.handleInviteeReject, this);
```

参数如下表所示：

参数	类型	含义
userID	String	拒绝通话的用户 ID
invitee	String	拒绝通话的用户 ID（**后续将废弃**）
inviteID	String	邀请 ID, 标识一次邀请
reason	String	reject 表示拒绝

NO_RESP

邀请用户无应答。

- 在C2C通话中，只有发起方会收到无人应答的回调 例如 A 邀请 B、C 进入通话，B不应答，A可以收到该回调，但C不行。
- 在IM群组通话中，所有被邀请人均能收到该回调 例如 A 邀请 B、C 进入通话，B不应答，A、C均能收到该回调。

```
let handleNoResponse = function(event) {
    console.log('用户无响应')
```

```
}
tuiCallEngine.on(TUICallEvent.NO_RESP, this.handleNoResponse, this);
```

参数如下表所示：

参数	类型	含义
groupID	String	群ID, 唯一标识
sponsor	String	发起方的用户 ID
userIDList	Array<String>	触发超时无响应的用户列表
timeoutUserList	Array<String>	触发超时无响应的用户列表（**后续将废弃**）
inviteID	String	邀请 ID, 标识一次邀请

LINE_BUSY

用户忙线。

```
let handleLineBusy = function(event) {
    console.log('用户忙线')
}
tuiCallEngine.on(TUICallEvent.LINE_BUSY, this.handleLineBusy, this);
```

参数如下表所示：

参数	类型	含义
userID	String	忙线用户 ID
invitee	String	忙线用户 ID（**后续将废弃**）
inviteID	String	邀请 ID, 标识一次邀请
reason	String	line busy 表示忙线

ON_CALL_CANCELED

如果通话未建立，都会抛出该事件。您可以通过监听该事件来实现类似未接来电、重置 UI 状态等显示逻辑。通话未建立的场景如下：

- 主叫取消：主叫抛出该事件，userID 为主叫；被叫也会抛出该事件，userID 为被叫；
- 被叫超时：主叫会同时抛出 NO_RESP 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；
- 被叫拒接：主叫会同时抛出 REJECT 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；
- 被叫忙线：主叫会同时抛出 LINE_BUSY 和 CALLING_CANCEL 事件，userID 为主叫；被叫抛出 CALLING_CANCEL 事件，userID 为被叫；

```
let handleOnCallCanceled = function(event) {
  console.log(event.userID);
};
tuiCallEngine.on(TUICallEvent.ON_CALL_CANCELED, handleOnCallCanceled, this);
```

参数如下表所示：

参数	类型	含义
userID	String	通话取消的用户 ID
callId	String	本次通话的唯一 ID

roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫、被叫

ON_CALL_BEGIN

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
let handleOnCallBegin = function(event) {
    console.log(event)
};
tuiCallEngine.on(TUICallEvent.ON_CALL_BEGIN, handleOnCallBegin, this);
```

参数如下表所示：

参数	类型	含义
callId	String	本次通话的唯一 ID
roomID	Number	此次通话的音视频房间 ID
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，类型：主叫、被叫

CALL_END

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
let handleCallingEnd = function(event) {
    console.log('通话结束')
}
tuiCallEngine.on(TUICallEvent.CALL_END, this.handleCallingEnd, this);
```

参数如下表所示：

参数	类型	含义
roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
callMediaType	Number	通话的媒体类型，视频通话、语音通话
callRole	String	角色，枚举类型：主叫（'inviter'）、被叫（'invitee'）、未知（''）
totalTime	Number	此次通话的时长，单位: 秒
userID	String	表示引起通话结束 userID
callId	String	本次通话的唯一 ID。

SDK_READY

SDK Ready 回调。

```
let handleSDKReady = function(event) {
    console.log('SDK Ready 回调')
}
tuiCallEngine.on(TUICallEvent.SDK_READY, this.handleSDKReady, this);
```

KICKED_OUT

被踢下线。

```
let handleKickedOut = function(event) {
    console.log('被踢下线')
}
tuiCallEngine.on(TUICallEvent.KICKED_OUT, this.handleKickedOut, this);
```

CALL_MODE

切换通话模式。

```
let handleCallMode = function(event) {
    console.log('切换通话模式')
}
tuiCallEngine.on(TUICallEvent.CALL_MODE, this.handleCallMode, this);
```

ERROR

监听 SDK 的错误信息。

```
let handleError = function(event) {
    console.log('错误信息')
}
tuiCallEngine.on(TUICallEvent.ERROR, this.handleError, this);
```

ON_USER_NETWORK_QUALITY_CHANGED

所有用户网络质量的事件。

```
let handleOnUserNetworkQualityChange = function(event) {
  console.log(event.networkQualityList);
};
tuiCallEngine.on(TUICallEvent.ON_USER_NETWORK_QUALITY_CHANGED, this.handleOnUserNetworkQualityChange,
this);
```

参数如下表所示：

参数	类型	含义
networkQualityList	Array<Object>	网络状态，根据 userID 可以获取对应用户当前的网络质量。例如：`networkQualityList: [{ userId: quality }]` 。 **网络质量说明：** • quality = 0，网络状况未知 • quality = 1，网络状况极佳 • quality = 2，网络状况较好 • quality = 3，网络状况一般 • quality = 4，网络状况差 • quality = 5，网络状况极差 • quality = 6，网络链接已断开

废弃事件

INVITED

收到一个新的来电请求事件，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
let handleNewInvitationReceived = function(event){
    console.log('被邀请通话')
  }
tuiCallEngine.on(EVENT.INVITED, this.handleNewInvitationReceived, this);
```

参数如下表所示：

参数	类型	含义
sponsor	String	邀请者
inviteeList	Array<String>	同时还被邀请的人（**后续将废弃**）
userIDList	Array<String>	同时还被邀请的人
isFromGroup	Boolean	是否是群组通话
inviteData	Object	邀请数据
inviteID	String	邀请 ID，标识一次邀请
userData	String	扩展字段：用于在邀请信令中增加扩展信息
callId	String	本次通话的唯一 ID。**v1.5.1+ 版本支持**
roomID	Number	此次通话的音视频房间 ID。**v1.5.1+ 版本支持**
callMediaType	Number	通话的媒体类型，视频通话、语音通话。**v1.5.1+ 版本支持**
callRole	String	角色，枚举类型：主叫、被叫。**v1.5.1+ 版本支持**

CALLING_CANCEL

表示此次通话被主叫取消（取消原因有可能是主叫主动取消、也有可能是来自于通话超时取消），被叫会收到，您可以通过监听这个事件来实现类似未接来电等显示逻辑。

1. A 呼叫 B，A 挂断，A 会抛出该事件，userID 就是 A。
2. A 呼叫 B，B 一直接通超时，B 会抛出该事件，userID 是 B。
3. A 呼叫 B、C、D，B 接通后，A 和 B 都挂断，此时 C 拒绝，此时 B 也会抛出该事件，事件里的 userID 为 C。

```
let handleCallingCancel = function(event) {
    console.log('通话被取消')
}
tuiCallEngine.on(EVENT.CALLING_CANCEL, this.handleCallingCancel, this);
```

参数如下表所示：

参数	类型	含义
userID	String	通话取消的用户 ID
invitee	String	通话取消的用户 ID（**后续将废弃**）
inviteID	String	邀请 ID, 标识一次邀请
callId	String	本次通话的唯一 ID。**v1.5.1+ 版本支持**
roomID	Number	此次通话的音视频房间 ID。**v1.5.1+ 版本支持**
callMediaType	Number	通话的媒体类型，视频通话、语音通话。**v1.5.1+ 版本支持**
callRole	String	角色，枚举类型：主叫、被叫。**v1.5.1+ 版本支持**

版权所有：腾讯云计算（北京）有限责任公司

uni-app（客户端）API 概览

最近更新时间：2025-06-25 15:45:22

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的头像、昵称
calls	发起 1v1 通话
join	发起群组通话
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
enableIncomingBanner	开启/关闭来电横幅显示
setScreenOrientation	设置通话界面显示方向

TUICallEvent

TUICallEvent 是 TUICallEngine 对应的事件类，您可以通过监听如下事件，来监听自己感兴趣的事件。

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调

版权所有：腾讯云计算（北京）有限责任公司

onUserNetworkQualityChanged	所有用户网络质量的反馈回调

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，使用 TUICallEngine API，您可以做一些额外的操作（目前仅支持以下接口）。

API	描述
hangup	结束通话
accept	接听通话
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数

TUICallKit

最近更新时间：2025-06-30 13:14:13

TUICallKit API 是音视频通话组件的含 UI 接口，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入 TUICallKit 。

API 概览

API	描述
login	登录
logout	登出
setSelfInfo	设置用户的昵称、头像
calls	发起单人或多人通话
join	主动加入通话
setCallingBell	设置自定义来电铃音
enableMuteMode	开启/关闭静音模式
enableFloatWindow	开启/关闭悬浮窗功能
enableIncomingBanner	开启/关闭来电横幅显示
setScreenOrientation	设置通话界面显示方向

API 详情

login

登录

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  SDKAppID: 0,
  userID: 'mike',
  userSig: '',
};
TUICallKit.login(options, (res) => {
  if (res.code === 0) {
    console.log('login success');
  } else {
    console.log(`login failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
options	Object	初始化参数
options.SDKAppID	Number	用户 SDKAppID
options.userID	String	用户 ID

options.userSig	String	用户签名 userSig
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

logout

登出

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.logout((res) => {
  if (res.code === 0) {
    console.log('logout success');
  } else {
    console.log(`logout failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是 URL 格式。

> **注意：**
> 通话中使用该接口修改用户信息，UI 不会立即更新，需要等到下次通话才能看到变化。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  nickName: 'jack',
  avatar: 'https:/****/user_avatar.png'
}
TUICallKit.setSelfInfo(options, (res) => {
  if (res.code === 0) {
    console.log('setSelfInfo success');
  } else {
    console.log(`setSelfInfo failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
options	Object	初始化参数
options.nickName	String	目标用户的昵称，非必填
options.avatar	String	目标用户的头像，非必填
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

calls

拨打电话（1v1通话）。

> **说明：**
> v2.9.0 支持。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userIDList: ['user1'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.calls(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

参数如下表所示：

参数	类型	含义
options	Object	初始化参数
options.userIDList	String[]	被呼叫的用户列表
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.callParams	Object	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

join

主动加入通话。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
    callId: xxx
};
TUICallKit.join(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
options	Object	初始化参数
options.callId	String	此次通话的唯一 ID。
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

setCallingBell

设置自定义来电铃音，这里仅限传入本地文件地址，需要确保该文件目录是应用可以访问的。

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

// 【1】通过 uni.saveFile 保存音频文件到本地，具体参考 saveFile 接口：
https://zh.uniapp.dcloud.io/api/file/file.html#savefile
```

```
const tempFilePath = './static/rain.mp3'; // 本地存放的音频文件
let musicFilePath = '';
uni.saveFile({
    tempFilePath: tempFilePath,
    success: (res) => {
        console.warn('保存文件成功 = ', JSON.stringify(res)); // 获取的是相对路径
        musicFilePath =  res.savedFilePath;

        // 【2】相对路径转绝对路径，否则访问不到
        musicFilePath = plus.io.convertLocalFileSystemURL(musicFilePath); // 转绝对路径

        // 【3】设置铃声
        TUICallKit.setCallingBell(musicFilePath, (res) => {
          if (res.code === 0) {
            console.log('setCallingBell success');
          } else {
            console.log(`setCallingBell failed, error message = ${res.msg}`);
          }
        });
    },
    fail: (err) => {
      console.error('保存文件失败');
    },
});
```

参数	类型	含义
filePath	String	来电铃音本地文件地址
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

enableMuteMode

开启/关闭静音模式。
开启后，收到通话请求，不会播放来电铃声。

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

参数	类型	含义
enable	Boolean	开启、关闭静音；true 表示开启静音

enableFloatWindow

开启/关闭悬浮窗功能，设置为 false 后，通话界面左上角的悬浮窗按钮会隐藏。

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

参数	类型	含义
enable	Boolean	开启、关闭悬浮窗功能；true 表示开启浮窗

enableIncomingBanner

开启/关闭来电横幅显示。
默认为 `false`，被叫端收到邀请后默认弹出全屏通话等待界面，开启后先展示一个横幅，然后根据需要拉起全屏通话界面。

> ⚠ 注意：
> v2.3.1+ 支持

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

setScreenOrientation

设置通话界面显示方向。
默认为竖屏，orientation：0-Portrait、1-LandScape、2-Auto。

> ⚠ 注意：
> v2.6.0+ 支持，仅支持 Android 设备。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
TUICallKit.setScreenOrientation(1);
```

废弃接口

TUICallKit 类型定义

CallParams

通话参数

> ⚠ 注意：
> - roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。
> - 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，比如数字 123 和字符串 123 是两个完全不同的房间。

参数	类型	描述
roomID	Number	数字房间号 **取值范围** 1 - 2147483647(2^31-1)
strRoomID	String	字符串房间号 **推荐取值** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a-zA-Z） • 数字（0-9） • 空格、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`-`、`:`、`;`、`<`、`=`、`.`、`>`、`?`、`@`、`[`、`]`、`^`、`_`、`{`、`}`、`|`、`~`、`,`
chatGroupId	String	与 chat 结合使用时， chat 群组的群ID
offlinePushInfo	OfflinePushInfo	厂商离线推送配置信息
timeout	Number	通话超时时间，默认：30s，单位：秒。设置范围：10s ~ 600s。如果传入0，默认30s。
userData	String	发起通话时自定义扩展字段，被叫端在 onCallReceived 中回调

OfflinePushInfo

> ⚠ 注意：

- title、description 字段值不能传入空字符串，必须传入有效值。
- v2.3.2+ 支持。

参数	类型	描述
title	String	离线推送展示通知栏标题
description	String	离线推送展示通知栏内容
iOSSound	String	离线推送声音设置（仅对 iOS 生效）。当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound
androidSound	String	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声。另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效
androidOPPOChannelID	String	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID
androidXiaoMiChannelID	String	小米手机 8.0 系统及以上的渠道 ID
androidFCMChannelID	String	FCM 通道手机 8.0 系统及以上的渠道 ID
ignoreIOSBadge	boolean	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 APP 的应用图标未读计数增加。
isDisablePush	boolean	是否关闭推送（默认开启推送）

废弃接口

call

拨打电话（1v1通话）

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  userID: 'mike',
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.call(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

参数如下表所示：

参数	类型	含义
options	Object	初始化参数
options.userID	String	目标用户的 userID
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.callParams	Object	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

groupCall

发起群组通话，注意：使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  groupID: 'myGroup',
  userIDList: ['mike', 'tom'],
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.groupCall(options, (res) => {
  if (res.code === 0) {
    console.log('call success');
  } else {
    console.log(`call failed, error message = ${res.msg}`);
  }
});
```

参数	类型	含义
options	Object	初始化参数
options.groupID	String	此次群组通话的群 ID
options.userIDList	Array<string>	目标用户的 userId 列表
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
options.callParams	Object	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

joinInGroupCall

加入群组中已有的音视频通话。

```
const TUICallKit = uni.requireNativePlugin('TencentCloud-TUICallKit');
const options = {
  roomID: 9898,
  groupID: 'myGroup',
  callMediaType: 1, // 语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
};
TUICallKit.joinInGroupCall(options, (res) => {
  if (res.code === 0) {
    console.log('joinInGroupCall success');
  } else {
    console.log(`joinInGroupCall failed, error message = ${res.msg}`);
  }
});
```

注意：

- roomID 与 strRoomID 是互斥的，若您选用 strRoomID，则 roomID 需要填写为 0。若两者都填，SDK 将优先选用 roomID。
- 不要混用 roomID 和 strRoomID，因为它们之间是不互通的，比如数字 123 和字符串 123 是两个完全不同的房间。

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
options	Object	初始化参数
options.roomID	Number	数字房间号 **取值范围** 1 – 2147483647(2^31–1)
options.strRoomID	String	字符串房间号 **推荐取值** 限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a–zA–Z） • 数字（0–9） • 空格、`!`、`#`、`$`、`%`、`&`、`(`、`)`、`+`、`–`、`:`、`;`、`<`、`=`、`.`、`>`、`?`、`@`、`[`、`]`、`^`、`_`、`{`、`}`、`|`、`~`、`,`
options.groupID	String	此次群组通话的群 ID
options.callMediaType	Number	通话的媒体类型，比如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
callback	Function	回调函数，code = 0 表示调用成功；code != 0 表示调用失败，失败原因见 msg

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2025-06-25 10:09:12

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口实现交互。**目前仅提供如下接口，如有需求可到这里 反馈 。**

> **注意：**
> 需要结合 腾讯云音视频通话插件 TencentCloud-TUICallKit 插件使用，不能单独使用。下述接口也是结合客户有自动挂断、设置渲染模式的需求下，才支持的。

API 概览

API	描述
hangup	结束通话
accept	接听通话
setVideoRenderParams	设置用户视频画面的渲染模式
setVideoEncoderParams	设置视频编码的编码参数

API 详情

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
TUICallEngine.hangup();
```

accept

接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。

```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
TUICallEngine.accept();
```

setVideoRenderParams

设置用户视频画面的渲染模式。

```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
const params = {
    userID: '234',
    fillMode: 0, // 0-填充模式，1一适应模式
    rotation: 1, // 0~3分别对应值：Rotation_0, Rotation_90, Rotation_180, Rotation_270;
};
TUICallEngine.setVideoRenderParams(params, (res) => {
    console.warn('res = ', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
userID	String	目标用户的 userID
params	Object	视频画面渲染参数，例如：画面旋转角度、填充模式

setVideoEncoderParams

设置视频编码的编码参数。
该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。

```
const TUICallEngine = uni.requireNativePlugin('TencentCloud-TUICallKit-TUICallEngine');
const params = {
    resolution: 108,
    resolutionMode: 0, // 0--横屏，1--竖屏
};
TUICallEngine.setVideoEncoderParams(params, (res) => {
    console.warn('res = ', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
resolution	Number	视频分辨率 • 62 -- 宽高比 16:9；分辨率 640x360； • 64 -- 宽高比 4:3；分辨率 960x720； • 108 -- 宽高比 16:9；分辨率 640x360； • 110 -- 宽高比 16:9；分辨率 960x540； • 112 -- 宽高比 16:9；分辨率 1280x720； • 114 -- 宽高比 16:9；分辨率 1920x1080；
resolutionMode	Number	分辨率模式（横屏分辨率 or 竖屏分辨率） • 0-Landscape（横屏） • 1-Portrait（竖屏）

版权所有：腾讯云计算（北京）有限责任公司

TUICallEvent

最近更新时间：2025-06-25 15:45:22

事件简介

可以监听 TUICallKit 对应的事件，进行提示或其它交互。

回调事件概览

API	描述
onError	通话过程中错误回调
onCallReceived	通话请求的回调
onCallCancelled	通话取消的回调
onCallBegin	通话接通的回调
onCallEnd	通话结束的回调
onCallMediaTypeChanged	通话媒体类型发生改变的回调
onUserReject	xxxx 用户拒绝通话的回调
onUserNoResponse	xxxx 用户不响应的回调
onUserLineBusy	xxxx 用户忙线的回调
onUserJoin	xxxx 用户加入通话的回调
onUserLeave	xxxx 用户离开通话的回调
onUserVideoAvailable	xxx 用户是否有视频流的回调
onUserAudioAvailable	xxx 用户是否有音频流的回调
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调
onUserNetworkQualityChanged	所有用户网络质量的反馈回调
onKickedOffline	当前用户被踢下线
onUserSigExpired	在线时票据过期
CustomViewClickEvent	点击新增自定义按钮时会触发该事件。v2.7.0+

回调事件详情

通过 globalEvent 监听原生插件抛出的事件。

```
const TUICallKitEvent = uni.requireNativePlugin('globalEvent');
```

onError

错误回调。

> **说明**
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
TUICallKitEvent.addEventListener('onError', (res) => {
```

```
  console.log('onError', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	错误回调参数
res.code	Number	错误码
res.msg	String	错误信息

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
TUICallKitEvent.addEventListener('onCallReceived', (res) => {
  console.log('onCallReceived', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.callerId	String	主叫 ID（邀请方）
res.calleeIdList	Array<String>	被叫 ID 列表（被邀请方）
res.groupId	String	群通话 ID。在群组通话时返回，1v1 返回 groupId 为空字符串
res.callMediaType	Number	通话的媒体类型，例如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.userData	String	用户增加的扩展字段

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
TUICallKitEvent.addEventListener('onCallCancelled', (res) => {
  console.log('onCallCancelled', res);
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数

res.userId	String	用户的 ID

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
TUICallKitEvent.addEventListener('onCallBegin', (res) => {
  console.log('onCallBegin', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
res.callMediaType	Number	通话的媒体类型，例如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)

onCallEnd

表示通话结束，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
TUICallKitEvent.addEventListener('onCallEnd', (res) => {
  console.log('onCallEnd', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.roomID	Number	此次通话的音视频房间 ID，目前仅支持数字房间号，后续版本会支持字符串房间号
res.callMediaType	Number	通话的媒体类型，例如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.callRole	Number	角色，枚举类型：主叫(callRole = 1)、被叫(callRole = 2)
res.totalTime	Number	此次通话的时长，单位：秒

> **注意**
> 客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。

```
TUICallKitEvent.addEventListener('onCallMediaTypeChanged', (res) => {
  console.log('onCallMediaTypeChanged', JSON.stringify(res));
});
```

参数如下表所示：

版权所有：腾讯云计算（北京）有限责任公司

参数	类型	含义
res	Object	回调参数
res.oldCallMediaType	Number	旧的通话类型，例如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)
res.newCallMediaType	Number	新的通话类型，例如：语音通话(callMediaType = 1)、视频通话(callMediaType = 2)

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。

```
TUICallKitEvent.addEventListener('onUserReject', (res) => {
  console.log('onUserReject', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	拒绝用户的 ID

onUserNoResponse

对方无回应的回调。

```
TUICallKitEvent.addEventListener('onUserNoResponse', (res) => {
  console.log('onUserNoResponse', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	无响应用户的 ID

onUserLineBusy

通话忙线回调。

```
TUICallEngine.addEventListener('onUserLineBusy', (res) => {
  console.log('onUserLineBusy', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	忙线用户的 ID

onUserJoin

有用户进入此次通话的回调。

```
TUICallEngine.addEventListener('onUserJoin', (res) => {
  console.log('onUserJoin', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	加入当前通话的用户 ID

onUserLeave

有用户离开此次通话的回调。

```
TUICallEngine.addEventListener('onUserLeave', (res) => {
  console.log('onUserLeave', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	离开当前通话的用户 ID

onUserVideoAvailable

用户是否开启视频上行回调。

```
TUICallEngine.addEventListener('onUserVideoAvailable', (res) => {
  console.log('onUserVideoAvailable', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.userId	String	通话用户 ID
res.isVideoAvailable	boolean	用户视频是否可用

onUserAudioAvailable

用户是否开启音频上行回调。

```
TUICallEngine.addEventListener('onUserAudioAvailable', (res) => {
  console.log('onUserAudioAvailable', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义

res	Object	回调参数
res.userId	String	用户 ID
res.isAudioAvailable	boolean	用户音频是否可用

onUserVoiceVolumeChanged

用户通话音量的回调。

```
TUICallEngine.addEventListener('onUserVoiceVolumeChanged', (res) => {
  console.log('onUserVoiceVolumeChanged', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.volumeMap	Map<String, Integer>	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100

onUserNetworkQualityChanged

用户网络质量的回调。

```
TUICallEngine.addEventListener('onUserNetworkQualityChanged', (res) => {
  console.log('onUserNetworkQualityChanged', JSON.stringify(res));
});
```

参数如下表所示：

参数	类型	含义
res	Object	回调参数
res.networkQualityList	List	网络状态，根据每个 userId 可以获取对应用户当前的网络质量

onKickedOffline

当前用户被踢下线：此时可以 UI 提示用户，并再次重新调用初始化。

```
TUICallingEvent.addEventListener('onKickedOffline', (res) => {
  console.log('onKickedOffline', JSON.stringify(res));
});
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并再次重新调用初始化。

```
TUICallEngine.addEventListener('onUserSigExpired', (res) => {
  console.log('onUserSigExpired', JSON.stringify(res));
});
```

CustomViewClickEvent

点击新增自定义按钮时会触发该事件。

```
uni.$TUICallKitEvent.addEventListener('CustomViewClickEvent', (res) => {
    console.log(res.viewId);
});
```

参数	类型	含义
res	Object	回调参数
res.viewId	String	触发的点击事件的自定义按钮 viewId

版权所有：腾讯云计算（北京）有限责任公司

Flutter
API 概览

最近更新时间：2025-06-25 15:45:22

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	描述
login	登录。
logout	登出。
setSelfInfo	设置用户的昵称、头像。
calls	发起通话。
join	主动加入通话。
enableMuteMode	开启/关闭静音模式。
enableFloatWindow	开启/关闭悬浮窗功能。
setCallingBell	设置自定义来电铃音。
enableVirtualBackground	开启/关闭虚拟背景功能。

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	描述
init	初始化 SDK 函数，完成包含通话服务鉴权在内初始化动作。
unInit	反初始化函数，释放 TUICallEngine 使用到的相关资源。
addObserver	增加事件回调。
removeObserver	移除回调接口。
calls	发起通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
join	主动加入通话
inviteUser	邀请其他人加入通话。
startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
closeCamera	关闭摄像头。
switchCamera	切换前后摄像头。

版权所有：腾讯云计算（北京）有限责任公司

openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）。
setSelfInfo	设置用户的头像、昵称。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（尊享版套餐支持）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
setBeautyLevel	设置美颜级别。
setBlurBackground	设置视频模糊效果。
setVirtualBackground	设置虚拟背景图片。

TUICallObserver

TUICallObserver 是回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	描述
onError	通话过程中错误回调。
onUserInviting	当用户被邀请加入通话时的回调。
onCallReceived	通话请求的回调。
onCallNotConnected	当前设备未接通通话。
onCallBegin	通话接通的回调。
onCallEnd	通话结束的回调。
onUserReject	xxxx 用户拒绝通话的回调。
onUserNoResponse	xxxx 用户不响应的回调。
onUserLineBusy	xxxx 用户忙线的回调。
onUserJoin	xxxx 用户加入通话的回调。
onUserLeave	xxxx 用户离开通话的回调。
onUserVideoAvailable	xxxx 用户是否有视频流的回调。
onUserAudioAvailable	xxxx 用户是否有音频流的回调。
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调。
onUserNetworkQualityChanged	所有用户网络质量的反馈回调。
onKickedOffline	当前用户被移下线。
onUserSigExpired	在线时票据过期。

关键类型定义

类型	描述
VideoEncoderParams	视频编码参数。

VideoRenderParams	视频画面的渲染参数。
TUICallMediaType	通话类型。
TUICallParams	通话拓展参数。
TUICamera	摄像头类型。
TUIAudioPlaybackDevice	音频播放路由。
TUIRoomId	房间 ID。

版权所有：腾讯云计算（北京）有限责任公司

TUICallKit

最近更新时间：2025-06-25 15:45:22

TUICallKit API 简介

TUICallKit API 是音视频通话组件的含 UI 接口，使用TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入（TUICallKit）。

API 概览

API	描述
login	登录。
logout	登出。
setSelfInfo	设置用户的昵称、头像。
calls	发起通话。
join	主动加入通话。
enableMuteMode	开启/关闭静音模式。
enableFloatWindow	开启/关闭悬浮窗功能。
setCallingBell	设置自定义来电铃音。
enableVirtualBackground	开启/关闭虚拟背景功能。

API 详情

login

登录。

```
Future<TUIResult> login(int sdkAppId, String userId, String userSig)
```

参数	类型	含义
sdkAppId	int	用户 SDKAppID。
userId	String	用户 ID。
userSig	String	用户签名 userSig。
返回值	TUIResult	包含 code 和 message 信息：code 为空 ("") 表示调用成功；code 不为空 ("") 表示调用失败，失败原因见 message。

logout

登出。

```
Future<void> logout()
```

setSelfInfo

设置用户昵称、头像。用户昵称不能超过500字节，用户头像必须是URL格式。

```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

参数	类型	含义
nickName	String	目标用户的昵称，非必填。
avatar	String	目标用户的头像，非必填。
返回值	TUIResult	包含 code 和 message 信息：code 为空 ("") 表示调用成功；code 不为空 ("") 表示调用失败，失败原因见 message。

calls

发起通话。v2.9+ 版本支持。

```
Future<TUIResult> calls(List<String> userIdList, TUICallMediaType mediaType, TUICallParams params)
```

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表。
callMediaType	TUICallMediaType	通话的媒体类型，例如：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
params	TUICallParams	可选通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

join

主动加入通话。v2.9+ 版本支持。

```
Future<void> join(String callId)
```

参数	类型	含义
callId	String	此次通话的唯一 ID。

enableMuteMode

开启后，收到通话请求，不会播放来电铃声。

```
Future<void> enableMuteMode(bool enable)
```

参数	类型	含义
enable	bool	开启、关闭静音；true 表示开启静音。

enableFloatWindow

开启/关闭悬浮窗功能，设置为false后，通话界面左上角的悬浮窗按钮会隐藏。

```
Future<void> enableFloatWindow(bool enable)
```

参数	类型	含义
enable	bool	开启、关闭悬浮窗功能；true 表示开启浮窗。

setCallingBell

自定义来电铃声：将铃声文件添加至主工程的 `assets` 资源中，传入资源文件名称即可。

```
Future<void> setCallingBell(String assetName)
```

enableVirtualBackground

开启/关闭虚拟背景功能，开启虚拟背景功能后，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

```
Future<void> enableVirtualBackground(bool enable)
```

参数	类型	含义
enable	bool	开启、关闭静音；true 表示开启静音。

废弃接口

call

拨打电话（1v1通话）。

> ⚠ 注意：
> 该接口已在 v2.9+ 版本废弃，建议使用 calls 接口替代。

```
Future<void> call(String userId, TUICallMediaType callMediaType, [TUICallParams? params])
```

参数	类型	含义
userId	String	目标用户的 userID。
callMediaType	TUICallMediaType	通话的媒体类型，例如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`。

groupCall

发起群组通话，注意：使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

> ⚠ 注意：
> 该接口已在 v2.9+ 版本废弃，建议使用 calls 接口替代。

```
Future<void> groupCall(String groupId, List<String> userIdList, TUICallMediaType callMediaType,
[TUICallParams? params])
```

参数	类型	含义
groupId	String	此次群组通话的群 ID。
userIdList	List<String>	目标用户的 userId 列表。
callMediaType	TUICallMediaType	通话的媒体类型，例如：`TUICallMediaType.video` 或 `TUICallMediaType.audio`。

joinInGroupCall

加入群组中已有的音视频通话。

> ⚠ 注意：

该接口已在 v2.9+ 版本废弃，建议使用 join 接口替代。

```
Future<void> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType callMediaType)
```

参数	类型	含义
roomId	TUIRoomID	此次通话的音视频房间 ID。
groupId	String	此次群组通话的群 ID。
callMediaType	TUICallMediaType	通话的媒体类型，例如：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。

TUICallEngine

最近更新时间：2025-06-25 15:45:22

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

API 概览

API	描述
init	初始化 SDK 函数，完成包含通话服务鉴权在内初始化动作。
unInit	反初始化函数，释放 TUICallEngine 使用到的相关资源。
addObserver	增加事件回调。
removeObserver	移除回调接口。
calls	发起通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
join	主动加入通话
inviteUser	邀请其他人加入通话。
startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
closeCamera	关闭摄像头。
switchCamera	切换前后摄像头。
openMicrophone	打开麦克风。
closeMicrophone	关闭麦克风。
selectAudioPlaybackDevice	选择音频播放设备（听筒/免提）。
setSelfInfo	设置用户的头像、昵称。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（尊享版套餐支持）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
setBeautyLevel	设置美颜级别。
setBlurBackground	设置视频模糊效果。
setVirtualBackground	设置虚拟背景图片。

API 详情

init

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。

```
Future<TUIResult> init(int sdkAppID, String userId, String userSig)
```

unInit

反初始化函数，释放 `TUICallEngine` 使用到的相关资源。

```
Future<TUIResult> unInit()
```

addObserver

添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。

```
Future<void> addObserver(TUICallObserver observer)
```

removeObserver

移除回调接口。

```
Future<void> removeObserver(TUICallObserver observer)
```

calls

发起通话。**v2.9+ 版本支持。**

```
Future<TUIResult> calls(List<String> userIdList, TUICallMediaType mediaType, TUICallParams params)
```

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表。
callMediaType	TUICallMediaType	通话的媒体类型，例如：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
params	TUICallParams	可选通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

accept

接听当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。

```
Future<TUIResult> accept()
```

reject

拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。

```
Future<TUIResult> reject()
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
Future<TUIResult> hangup()
```

inviteUser

邀请用户加入此次群组通话。
使用场景：一个群组通话中的用户主动邀请其他人时使用。

```
Future<void> iniviteUser(List<String> userIdList, TUICallParams params, TUIValueCallback callback)
```

参数	类型	含义
userIdList	List<String>	目标用户的 userId 列表。
params	TUICallParams	通话参数扩展字段，例如：离线推送自定义内容。

join

主动加入通话。v2.9+ 版本支持。

```
Future<void> join(String callId)
```

参数	类型	含义
callId	String	此次通话的唯一 ID。

startRemoteView

设置显示视频画面的 View 对象。

```
Future<void> startRemoteView(String userId, intviewId)
```

参数	类型	含义
userId	String	目标用户的 userId。
intviewId	int	显示视频画面的 widget 对应的 Id。

stopRemoteView

停止订阅远端用户的视频数据。

```
Future<void> stopRemoteView(String userId)
```

参数	类型	含义
userId	String	目标用户的 userId。

openCamera

开启摄像头。

```
Future<TUIResult> openCamera(TUICamera camera, int? viewId)
```

参数	类型	含义
camera	TUICamera	前置/后置摄像头。
viewId	int	显示视频画面的 widget 对应的 ID。

版权所有：腾讯云计算（北京）有限责任公司

closeCamera

关闭摄像头。

```
Future<void> closeCamera()
```

switchCamera

切换前后摄像头。

```
Future<void> switchCamera(TUICamera camera)
```

参数	类型	含义
camera	TUICamera	前置 / 后置摄像头。

openMicrophone

打开麦克风。

```
Future<TUIResult> openMicrophone()
```

closeMicrophone

关闭麦克风。

```
Future<void> closeMicrophone()
```

selectAudioPlaybackDevice

选择音频播放设备。
目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。

```
Future<void> selectAudioPlaybackDevice(TUIAudioPlaybackDevice device)
```

参数	类型	含义
device	TUIAudioPlaybackDevice	听筒 / 扬声器。

setSelfInfo

设置用户昵称、头像。
用户昵称不能超过500字节，用户头像必须是 URL 格式。

```
Future<TUIResult> setSelfInfo(String nickname, String avatar)
```

参数	类型	含义
nickname	String	用户昵称。
avatar	String	用户头像（格式为 URL）。

enableMultiDeviceAbility

开启/关闭 TUICallEngine 的多设备登录模式（尊享版套餐支持）。

版权所有：腾讯云计算（北京）有限责任公司

```
Future<TUIResult> enableMultiDeviceAbility(bool enable)
```

setVideoRenderParams

设置用户视频画面的渲染模式。

```
Future<TUIResult> setVideoRenderParams(String userId, VideoRenderParams params)
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId。
params	VideoRenderParams	视频画面渲染参数，例如：画面旋转角度、填充模式。

setVideoEncoderParams

设置视频编码的编码参数。
该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。

```
Future<TUIResult> setVideoEncoderParams(VideoEncoderParams params)
```

参数如下表所示：

参数	类型	含义
params	VideoEncoderParams	视频编码的参数，例如：分辨率、视频宽高比。

setBeautyLevel

设置美颜级别。

```
Future<TUIResult> setBeautyLevel(double level)
```

参数如下表所示：

参数	类型	含义
level	double	美颜等级，使用范围0.0~9.0。

setBlurBackground

设置视频模糊效果。

```
void setBlurBackground(int level, Function(int code, String message)? errorCallback)
```

参数如下表所示：

参数	类型	含义
level	int	• 0：关闭 • 1：低 • 2：中 • 3：高

setVirtualBackground

设置虚拟背景图片。

```
void setVirtualBackground(String imagePath, Function(int code, String message)? errorCallback)
```

参数	类型	含义
imagePath	String	图片文件名，需要将文件添加至主工程的 assets 资源中。

废弃接口

call

拨打电话（1v1通话）。

> ⚠ 注意：
> 该接口已在 v2.9+ 版本废弃，建议使用 calls 接口替代。

```
Future<TUIResult> call(String userId, TUICallMediaType mediaType, TUICallParams params)
```

参数如下表所示：

参数	类型	含义
userId	String	目标用户的 userId。
mediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话。
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

ignore

忽略当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数忽略来电，此时主叫会收到 `onUserLineBusy` 的回调。
备注：如果您的业务中存在直播、会议等场景，在直播/会议中的情况时，也可以调用这个函数来忽略此次来电。

> ⚠ 注意：
> 注意：v2.9+版本废弃，针对 calls 发起的通话不生效，建议使用 reject 接口。

```
Future<TUIResult> ignore()
```

groupCall

发起群组通话。
使用群组通话前需要创建 IM 群组，如果已经创建，请忽略。

> ⚠ 注意：
> 该接口已在 v2.9+ 版本废弃，建议使用 calls 接口替代。

```
Future<TUIResult> groupCall(String groupId, List<String> userIdList, TUICallMediaType mediaType,
TUICallParams params)
```

参数	类型	含义
groupId	String	此次群组通话的群 ID。

userIdList	List<String>	目标用户的 userId 列表。
mediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话。
params	TUICallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

joinInGroupCall

主动加入此次群组通话。
使用场景：群组内用户主动加入此次群组通话使用。

> ⚠ **注意：**
> 该接口已在 v2.9+ 版本废弃，建议使用 calls 接口替代。

```
Future<TUIResult> joinInGroupCall(TUIRoomId roomId, String groupId, TUICallMediaType mediaType)
```

参数	类型	含义
roomId	TUIRoomId	此次通话的音视频房间 ID。
groupId	String	此次群组通话的群 ID。
mediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话。

switchCallMediaType

切换视频通话到语音通话。

```
Future<void> switchCallMediaType(TUICallMediaType mediaType)
```

参数	类型	含义
mediaType	TUICallMediaType	通话的媒体类型，例如视频通话、语音通话。

TUICallObserver

最近更新时间：2025-06-25 15:45:22

TUICallObserver API 简介

TUICallObserver 是 TUICallKit 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

回调事件概览

API	描述
onError	通话过程中错误回调。
onUserInviting	当用户被邀请加入通话时的回调。
onCallReceived	通话请求的回调。
onCallNotConnected	当前设备未接通通话。
onCallBegin	通话接通的回调。
onCallEnd	通话结束的回调。
onUserReject	xxxx 用户拒绝通话的回调。
onUserNoResponse	xxxx 用户不响应的回调。
onUserLineBusy	xxxx 用户忙线的回调。
onUserJoin	xxxx 用户加入通话的回调。
onUserLeave	xxxx 用户离开通话的回调。
onUserVideoAvailable	xxxx 用户是否有视频流的回调。
onUserAudioAvailable	xxxx 用户是否有音频流的回调。
onUserVoiceVolumeChanged	所有用户音量大小的反馈回调。
onUserNetworkQualityChanged	所有用户网络质量的反馈回调。
onKickedOffline	当前用户被移下线。
onUserSigExpired	在线时票据过期。

回调事件详情

通过 addObserver 监听Flutter插件抛出的事件。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {

    },
    onUserInviting: (String userId) {

    },
    onCallReceived: (String callId, String callerId, List<String> calleeIdList,
        TUICallMediaType mediaType, CallObserverExtraInfo info) {

    },
    onCallNotConnected: (String callId, TUICallMediaType mediaType, CallEndReason reason,
        String userId, CallObserverExtraInfo info) {
```

```
    },
    onCallBegin: (String callId, TUICallMediaType mediaType, CallObserverExtraInfo info) {

    },
    onCallEnd: (String callId, TUICallMediaType mediaType, CallEndReason reason,
        String userId, double totalTime, CallObserverExtraInfo info) {

    },
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType newCallMediaType) {

    },
    onUserReject: (String userId) {

    },
    onUserNoResponse: (String userId) {

    },
    onUserLineBusy: (String onUserLineBusy) {

    },
    onUserJoin: (String userId) {

    },
    onUserLeave: (String userId) {

    },
    onUserVideoAvailable: (String userId, bool isVideoAvailable) {

    },
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {

    },
    onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList) {

    },
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {

    },
    onKickedOffline: () {

    },
    onUserSigExpired: () {

    }
));
```

onError

错误事件回调。

> **说明**
> **SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。**

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onError: (int code, String message) {
      // TODO
    }
```

```
));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onUserInviting

当用户被邀请加入通话时的回调

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserInviting: (String userId) {
      // TODO
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	被邀请的用户 ID。

onCallReceived

收到一个新的来电请求回调，被叫会收到，您可以通过监听这个事件，来决定是否显示通话接听界面。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallReceived: (String callId, String callerId, List<String> calleeIdList, TUICallMediaType
mediaType, CallObserverExtraInfo info) {
      // TODO
    }
));
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID。
callerId	String	主叫 ID（邀请方）。
calleeIdList	List<String>	被叫 ID 列表（被邀请方）。
mediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio`。
info	CallObserverExtraInfo	扩展信息。

onCallNotConnected

通话取消的回调

```
TUICallEngine.instance.addObserver(TUICallObserver(
  onCallNotConnected: (String callId, TUICallMediaType mediaType, CallEndReason reason,
    String userId, CallObserverExtraInfo info) {
```

```
    // TODO
  }
));
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID。
mediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
reason	CallEndReason	通话结束的原因。
userId	String	结束通话的用户 ID。
info	CallObserverExtraInfo	扩展信息。

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallBegin: (String callId, TUICallMediaType mediaType, CallObserverExtraInfo info) {
    // TODO
  }
));
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID。
mediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
info	CallObserverExtralnfo	扩展信息。

onCallEnd

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallEnd: (String callId, TUICallMediaType mediaType, CallEndReason reason,
        String userId, double totalTime, CallObserverExtraInfo info) {
    // TODO
  }
));
```

参数如下表所示：

参数	类型	含义
callId	String	此次通话的唯一 ID。

mediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
reason	CallEndReason	通话结束的原因。
userId	String	结束通话的用户 ID。
totalTime	double	此次通话的时长，单位 ms。
info	CallObserverExtralnfo	扩展信息。

注意：
客户端的事件一般都会随着杀进程等异常事件丢失掉，如果您需要通过监听通话时长来完成计费等逻辑，建议可以使用 REST API 来完成这类流程。

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserReject: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
res.userId	String	拒绝用户的 ID。

onUserNoResponse

对方无回应的回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserNoResponse: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	无响应用户的 ID。

onUserLineBusy

通话忙线回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLineBusy: (String onUserLineBusy) {
      //您的回调处理逻辑
    },
));
```

参数如下表所示：

参数	类型	含义
userId	String	忙线用户的 ID。

onUserJoin

有用户进入此次通话的回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserJoin: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	加入当前通话的用户 ID。

onUserLeave

有用户离开此次通话的回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserLeave: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	离开当前通话的用户 ID。

onUserVideoAvailable

用户是否开启视频上行回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVideoAvailable: (String userId, bool isVideoAvailable) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	通话用户 ID。
isVideoAvailable	bool	用户视频是否可用。

onUserAudioAvailable

用户是否开启音频上行回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserAudioAvailable: (String userId, bool isAudioAvailable) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID。
isAudioAvailable	bool	用户音频是否可用。

onUserVoiceVolumeChanged

用户通话音量的回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserVoiceVolumeChanged: (Map<String, int> volumeMap) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
volumeMap	Map<String, int>	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100。

onUserNetworkQualityChanged

用户网络质量的回调。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserNetworkQualityChanged: (List<TUINetworkQualityInfo> networkQualityList) {
      //您的回调处理逻辑
    }
));

//TUINetworkQualityInfo的定义如下：
class TUINetworkQualityInfo {
    String userId;
    TUINetworkQuality quality;
    TUINetworkQualityInfo({required this.userId, required this.quality});
}
// TUINetworkQuality的定义如下：
enum TUINetworkQuality {
  unknown,
  excellent,
  good,
  poor,
  bad,
  vBad,
  down
}
```

参数如下表所示：

参数	类型	含义
networkQualityList	List<TUINetworkQualityInfo>	网络状态，根据每个 userId 可以获取对应用户当前的网络质量。

onKickedOffline

当前用户被踢下线：此时可以 UI 提示用户，并重新调用初始化。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onKickedOffline: () {
      //您的回调处理逻辑
    }
));
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并重新调用初始化。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onUserSigExpired: () {
      //您的回调处理逻辑
    }
));
```

废弃回调

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景，您可以通过监听这个事件来实现类似未接来电、重置 UI 状态等显示逻辑。

- 主叫取消：主叫收到该回调（userId 为自己）；被叫收到该回调（userId 为**主叫的 ID**）。
- 被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（userId 是自己的 ID）；被叫收到 onCallCancelled 回调（userId 是自己的 ID）。
- 被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（userId 是自己的 ID）。
- 异常中断：被叫接收通话失败，收到该回调（userId 是自己的 ID）。

```
TUICallEngine.instance.addObserver(TUICallObserver(
    onCallCancelled: (String userId) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
userId	String	用户的 ID。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。

```
TUICallEngine.instance.addObserver(TUICallObserver(
```

```
    onCallMediaTypeChanged: (TUICallMediaType oldCallMediaType, TUICallMediaType newCallMediaType) {
      //您的回调处理逻辑
    }
));
```

参数如下表所示：

参数	类型	含义
oldCallMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。
newCallMediaType	TUICallMediaType	通话的媒体类型，示例：`TUICallMediaType.video` 或 `TUICallMediaType.audio` 。

类型定义

最近更新时间：2025-07-01 09:47:12

常用结构

TUIResult

调用 API 的返回值

类型	类型	描述
code	String	code 为空 "" 表示调用成功，code 不为空 "" 表示调用失败。
message	String	失败原因。

TUIRoomId

房间 ID

类型	类型	描述
intRoomId	int	数字房间号。
strRoomId	String	字符串房间号。

VideoRenderParams

视频画面的渲染参数

类型	类型	描述
fillMode	FillMode	视频画面填充模式。
rotation	Rotation	视频画面旋转方向。

VideoEncoderParams

视频编码参数

类型	类型	描述
resolution	Resolution	视频分辨率。
resolutionMode	ResolutionMode	视频宽高比模式。

TUICallParams

通话参数

类型	类型	描述
roomId	TUIRoomId	房间 ID。
offlinePushInfo	TUIOfflinePushInfo	厂商离线推送配置信息。
timeout	String	超时时常。
userData	String	用户自定义数据。
chatGroupId	String	群组 ID。

TUIOfflinePushInfo

厂商离线推送配置信息，详情配置步骤请参考：离线唤醒 。

类型	类型	描述
title	String	离线推送展示通知栏标题。
desc	String	离线推送展示通知栏内容。
ignoreIOSBadge	bool	离线推送忽略 badge 计数（仅对 iOS 生效），如果设置为 true，在 iOS 接收端，这条消息不会使 App 的应用图标未读计数增加。
iOSSound	String	离线推送声音设置（仅对 iOS 生效）。当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound。
androidSound	String	离线推送声音设置（仅对 Android 生效, IMSDK 6.1 及以上版本支持）。只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参照：小米自定义铃声 。另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效。
androidOPPOChannelID	String	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID。
androidVIVOClassification	int	VIVO 推送消息分类 (待废弃接口，VIVO 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。0：运营消息 1：系统消息，默认取值为 1。
androidXiaoMiChannelID	String	小米手机 8.0 系统及以上的渠道 ID。
androidFCMChannelID	String	FCM 通道手机 8.0 系统及以上的渠道 ID。
androidHuaWeiCategory	String	华为推送消息分类，详见：华为消息分类标准 。
isDisablePush	bool	是否关闭推送（默认开启推送）。
iOSPushType	TUICallIOSOfflinePushType	iOS 离线推送类型，默认：APNs。

TUICallRecords

通话记录信息

类型	类型	描述
callId	String	通话记录 ID。
inviter	String	邀请者 ID。
inviteList	List<String>	被邀请用户 ID 列表。
scene	TUICallScene	通话场景。
mediaType	TUICallMediaType	通话媒体类型。
groupId	String	群组 ID。
role	TUICallRole	通话角色。
result	TUICallResultType	通话结果类型。
beginTime	int	开始时间。
totalTime	int	总时间。

TUICallRecentCallsFilter

通话记录过滤条件

类型	类型	描述
begin	double	开始时间。
end	double	结束时间。
resultType	TUICallResultType	通话记录类型。

CallObserverExtraInfo

回调扩展信息

类型	类型	描述
roomId	TUIRoomId	房间 ID。
role	TUICallRole	通话角色。
userData	String	发起通话时自定义扩展字段，详情见 TUICallParams 。
chatGroupId	String	群组 ID。

枚举定义

TUICallMediaType

通话类型

类型	描述
none	未知类型。
audio	语音通话。
video	视频通话。

TUICallRole

通话角色

类型	描述
none	未知类型。
caller	主叫（邀请方）。
called	被叫（被邀请方）。

TUICallStatus

通话状态

类型	描述
none	未知类型。
waiting	通话等待中。
accept	通话已接听。

TUICallScene

通话场景

类型	描述
singleCall	单人通话。
groupCall	群组通话。

TUINetworkQuality

网络质量

类型	描述
unknown	未定义。
excellent	当前网络非常好。
good	当前网络比较好。
poor	当前网络一般。
bad	当前网络较差。
vBad	当前网络很差。
down	当前网络不满足通话的最低要求。

FillMode

视频画面填充模式

类型	描述
fill	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。
fit	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

Rotation

视频画面旋转方向

类型	描述
rotation_0	不旋转。
rotation_90	顺时针旋转90度。
rotation_180	顺时针旋转180度。
rotation_270	顺时针旋转270度。

ResolutionMode

视频宽高比模式

类型	描述
landscape	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360。
portrait	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640。

Resolution

视频分辨率

版权所有：腾讯云计算（北京）有限责任公司

类型	描述
resolution_640_360	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps。
resolution_960_540	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps。
resolution_1280_720	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps。
resolution_1920_1080	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps。

TUICallIOSOfflinePushType

iOS 离线推送类型

类型	描述
APNs	普通推送。
VoIP	VoIP 推送。

TUICamera

摄像头类型

类型	描述
front	前置摄像头。
back	后置摄像头。

TUIAudioPlaybackDevice

音频播放路由

类型	描述
speakerphone	扬声器。
earpiece	耳麦。

TUICallResultType

通话记录类型

类型	描述
unknown	未知。
missed	未接。
incoming	接入。
outgoing	拨出。

CallEndReason

通话结束原因

类型	描述
unknown	未知。
hangup	挂断。

reject	拒绝。
noResponse	无响应。
offline	离线。
lineBusy	忙线。
canceled	取消通话。
otherDeviceAccepted	其他设备接听。
otherDeviceReject	其他设备拒绝。
endByServer	后台结束。

版权所有：腾讯云计算（北京）有限责任公司

React Native API 概览

最近更新时间：2025-06-25 15:58:52

TUICallKit (含 UI 接口)

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景。

API	含义
login	登录。
logout	退出登录。
calls	发起单人或多人通话。
call	进行一对一通话，支持自定义房间ID、通话超时、离线推送内容等。
groupCall	进行群组通话，支持自定义房间ID、通话超时、离线推送内容等。
joinInGroupCall	加入群组通话。
setCallingBell	自定义用户的铃声。
enableMuteMode	开启/关闭静音模式。
enableVirtualBackground	开启/关闭虚拟背景功能。
setScreenOrientation	设置屏幕方向。
on	监听 TUICallKit 事件
off	取消监听 TUICallKit 事件

TUICallEvent

TUICallEvent 是 TUICallKit 对应的回调事件类，您可以通过此回调，来监听自己感兴趣的回调事件。

API	含义
TUICallEvent.onError	通话过程中错误回调。
TUICallEvent.onCallReceived	通话请求的回调。
TUICallEvent.onCallCancelled	通话取消的回调。
TUICallEvent.onCallBegin	通话接通的回调。
TUICallEvent.onCallEnd	通话结束的回调。
TUICallEvent.onUserReject	xxxx 用户拒绝通话的回调。
TUICallEvent.onUserNoResponse	xxxx 用户不响应的回调。
TUICallEvent.onUserLineBusy	xxxx 用户忙线的回调。
TUICallEvent.onUserJoin	xxxx 用户加入通话的回调。
TUICallEvent.onUserLeave	xxxx 用户离开通话的回调。
TUICallEvent.onCallMediaTypeChanged	通话的媒体类型发生改变的回调。
TUICallEvent.onKickedOffline	当前用户被移下线。

版权所有：腾讯云计算（北京）有限责任公司

TUICallEvent.onUserSigExpired	在线时票据过期。
TUICallEvent.onUserVideoAvailable	xxx 用户是否有视频流的回调。
TUICallEvent.onUserAudioAvailable	xxx 用户是否有音频流的回调。
TUICallEvent.onUserVoiceVolumeChanged	所有用户音量大小的反馈回调。
TUICallEvent.onUserNetworkQualityChanged	所有用户网络质量的反馈回调。

TUICallEngine (无 UI 接口)

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套 API 根据您的业务需求自定义封装。

API	含义
CreateInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
login	完成音视频通话基础能力的鉴权。
on	增加事件回调。
off	移除回调接口。
call	发起 1v1 通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
switchCallMediaType	切换通话媒体类型，例如视频通话切音频通话。
startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
switchCamera	切换前后摄像头。
closeCamera	关闭摄像头。
openMicrophone	打开麦克风。。
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
setBeautyLevel	设置美颜级别，支持关闭默认美颜。
setSelfInfo	设置用户的昵称、头像。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）。

TUICallKit

最近更新时间：2025-06-25 15:45:22

TUICallKit API 简介

TUICallKit API 是音视频通话组件的**含 UI 接口**，使用 TUICallKit API，您可以通过简单接口快速实现一个类微信的音视频通话场景，更详细的接入步骤，详情请参见 快速接入TUICallKit 。

API 概览

API	含义
login	登录。
logout	退出登录。
calls	发起单人或多人通话。
call	进行一对一通话，支持自定义房间ID、通话超时、离线推送内容等。
groupCall	进行群组通话，支持自定义房间ID、通话超时、离线推送内容等。
joinInGroupCall	加入群组通话。
setCallingBell	自定义用户的铃声。
enableMuteMode	开启/关闭静音模式。
enableVirtualBackground	开启/关闭虚拟背景功能。
setScreenOrientation	设置屏幕方向。
on	监听 TUICallKit 事件
off	取消监听 TUICallKit 事件

API 详情

login

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

```
TUICallKit.login(
  {
    sdkAppId: 0,
    userId: '',
    userSig: '',
  },
  (res) => {
    console.log('login success');
  },
   (errCode, errMsg) => {
     console.log('login error');
   }
);
```

参数	类型	含义
sdkAppId	Number	在 实时音视频 TRTC 控制台 创建的音视频应用的唯一标识 SDKAppID。

userId	String	客户根据自己的业务自定义用户 ID，只允许包含大小写英文字母(a-z A-Z)、数字(0-9)及下划线和连词符。
userSig	String	在 实时音视频 TRTC 控制台 创建的音视频应用的 SDKSecretKey。

logout

退出登录，退出登录后将不再监听任何 TUICallKit 的事件。

```
TUICallKit.login(
  (res) => {
    console.log('login success', res);
  },
   (errCode, errMsg) => {
     console.log('login error', errCode, errMsg);
   }
);
```

calls

发起单人或多人通话。

```
TUICallKit.calls(
  {
    userIdList: userIDList,
    mediaType: MediaType.Audio,
    callParams: {
      offlinePushInfo: {
        title: '',
        desc: '',
      },
    },
  },
  () => {
    console.log('calls success');
  },
  () => {
    console.log('calls error');
  }
);
```

参数	类型	含义
userIdList	Array<String>	目标用户的 userId 列表
mediaType	MediaType	通话的媒体类型，例如视频通话、语音通话。 • MediaType.Audio：语音通话。 • MediaType.Video：视频通话。
callParams	callParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

call

拨打电话（1v1通话），支持自定义房间号、通话邀请超时时间，离线推送内容等。

```
TUICallKit.call(
  {
    userId: calleeID,
```

```
    mediaType: MediaType.Audio,
    callParams: {
      offlinePushInfo: {
        title: '',
        desc: '',
      },
    },
  },
  () => {
    console.log('call success');
  },
  () => {
    console.log('call error');
  }
);
```

参数	类型	含义
userId	String	目标用户的 userId。
mediaType	MediaType	通话的媒体类型，例如视频通话、语音通话。 • MediaType.Audio：语音通话。 • MediaType.Video：视频通话。
callParams	callParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

groupCall

发起群组通话。

```
TUICallKit.groupCall(
  {
    userIdList: userIDList,
    mediaType: MediaType.Audio,
    groupId: '',
  },
  (res) => {
    console.log('groupCall success', res);
  },
  (errCode, errMsg) => {
    console.log('groupCall error', errCode, errMsg);
  }
);
```

参数	类型	含义
groupId	String	此次群组通话的群 ID。
userIdList	Array<String>	被呼叫的用户列表。
mediaType	MediaType	通话的媒体类型，例如视频通话、语音通话。 • MediaType.Audio：语音通话。 • MediaType.Video：视频通话。
callParams	callParams	通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

joinInGroupCall

加入群组中已有的音视频通话。

```
TUICallKit.joinInGroupCall(
  {
    roomId: '',
    groupId: '',
    mediaType: '',
  }
);
```

参数	类型	含义
roomId	RoomId	此次通话的音视频房间 ID。
groupId	String	此次群组通话的群 ID。
mediaType	MediaType	通话的媒体类型，例如视频通话、语音通话。

setCallingBell

设置自定义来电铃音。

- **仅限传入本地 MP3 格式的文件地址，需要确保该文件目录是应用可以访问的。**
- **使用 import 方式引入铃声文件。**
- **如需恢复默认铃声，`filePath` 传空即可。**

```
String filePath = '';
TUICallKit.setCallingBell(filePath);
```

参数	类型	含义
filePath	String	铃声文件地址。

enableMuteMode

开启/关闭来电铃声。开启后，收到通话请求时，不会播放来电铃声。

```
Boolean enable = true
TUICallKit.enableMuteMode(enable);
```

参数	类型	含义
enable	Boolean	开启/关闭来电铃声。默认 false。

enableVirtualBackground

开启/关闭模糊背景功能。通过调用接口，您可以在 UI 上显示模糊背景的功能按钮，点击按钮可直接启用模糊背景功能。

```
Boolean enable = true
TUICallKit.enableVirtualBackground(enable);
```

参数	类型	含义
enable	Boolean	• enable = true 显示模糊背景按钮。 • enable = false 不显示模糊背景按钮。

setScreenOrientation

设置屏幕显示模式。

```
Number orientation = 0
TUICallKit.setScreenOrientation(orientation);
```

参数	类型	含义
orientation	Number	• orientation = 0：竖屏显示。 • orientation = 1：横屏显示。 • orientation = 2：根据当前设备的状态自动选择最佳的显示方式。

on

您可以通过以下示例代码监听 TUICallKit 事件（事件详情请参见 TUICallEvent ）。

```
TUICallKit.on(TUICallEvent.onCallReceived, (res: any) => {
  console.log('onUserReject userId=' + res.userId);
});
```

参数	类型	含义
type	String	TUICallKit 您监听的事件，事件列表请参见 TUICallEvent。
params	Any	事件所携带的信息，详情请参见 TUICallEvent。

off

您可以通过以下示例代码，取消监听 TUICallKit 事件（事件详情请参见 TUICallEvent ）。

```
TUICallKit.off(TUICallEvent.onCallReceived);
```

版权所有：腾讯云计算（北京）有限责任公司

TUICallEngine

最近更新时间：2025-06-25 15:45:22

TUICallEngine API 简介

TUICallEngine API 是音视频通话组件的**无 UI 接口**，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。

Overview

API	含义
CreateInstance	创建 TUICallEngine 实例（单例模式）。
destroyInstance	销毁 TUICallEngine 实例（单例模式）。
login	完成音视频通话基础能力的鉴权。
on	增加事件回调。
off	移除回调接口。
call	发起 1v1 通话。
accept	接听通话。
reject	拒绝通话。
hangup	结束通话。
switchCallMediaType	切换通话媒体类型，例如视频通话切音频通话。
startRemoteView	开始订阅远端用户视频流。
stopRemoteView	停止订阅远端用户视频流。
openCamera	开启摄像头。
switchCamera	切换前后摄像头。
closeCamera	关闭摄像头。
openMicrophone	打开麦克风。。
closeMicrophone	关闭麦克风
selectAudioPlaybackDevice	选择音频播放设备（听筒/扬声器）。
setVideoRenderParams	设置用户视频画面的渲染模式。
setVideoEncoderParams	设置视频编码的编码参数。
setBeautyLevel	设置美颜级别，支持关闭默认美颜。
setSelfInfo	设置用户的昵称、头像。
enableMultiDeviceAbility	开启/关闭 TUICallEngine 的多设备登录模式（群组通话版套餐 支持）。

API 详情

CreateInstance

创建 TUICallEngine 的单例。

```
TUICallKit.CreateInstance();
```

destroyInstance

销毁 TUICallEngine 的单例。

```
TUICallKit.destroyInstance();
```

login

初始化函数，请在使用所有功能之前先调用该函数，以便完成包含通话服务鉴权在内初始化动作。

```
login(
  sdkAppId: number;
  userId: string;
  userSig: string;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
sdkAppId	Number	您可以在**实时音视频控制台** > **应用管理** > **应用信息**中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。

on

添加回调接口，您可以通过这个接口，监听 `TUICallObserver` 相关的事件回调。

```
on(type: TUICallEvent, listener: (params: any) => void): void {}
```

参数	类型	含义
type	TUICallEvent	监听 TUICallkit 的某个事件。

off

移除回调接口。

```
off(type: TUICallEvent): void {}
```

参数	类型	含义
type	TUICallEvent	注销 TUICallkit 的某个事件。

call

发起 1v1 或 多人通话。

```
call(
  userId: string;
  mediaType: MediaType;
  callParams?: CallParams;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
```

```
): void {}
```

参数	类型	含义
userId	string	目标用户的 userId。
mediaType	MediaType	通话的媒体类型，例如：视频通话、语音通话。
callParams	CallParams	通话参数扩展字段，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

accept

接受当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数接听来电。

```
accept(
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

reject

拒绝当前通话，当您作为被叫收到 `onCallReceived()` 的回调时，可以调用该函数拒绝来电。

```
reject(
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

hangup

挂断当前通话，当您处于通话中，可以调用该函数结束通话。

```
hangup(
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

switchCallMediaType

切换视频通话到语音通话。

```
switchCallMediaType(mediaType: MediaType): void {}
```

参数	类型	含义
mediaType	MediaType	通话的媒体类型，例如视频通话、语音通话。

startRemoteView

开始订阅远端用户的视频数据。

```
startRemoteView(
  userId: string;
  viewId: string;
  onPlaying: (params: any) => void;
  onLoading: (params: any) => void;
): void {}
```

参数	类型	含义
userId	string	目标用户的 userId。
viewId	string	需要渲染的 view ID。

stopRemoteView

停止订阅远端用户的视频数据。

```
stopRemoteView(userId: string;): void {}
```

参数	类型	含义
userId	string	目标用户的 userId。

openCamera

开启摄像头。

```
openCamera(
  camera?: Camera;
  viewId: string;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;): void {}
```

参数	类型	含义
camera	Camera	前置/后置摄像头 • Camera.Front：前置摄像头 • Camera.Back：后置摄像头
viewId	string	需要渲染画面的 view ID

switchCamera

切换前后摄像头。

```
switchCamera(camera: Camera): void {}
```

参数	类型	含义
camera	Camera	前置/后置 摄像头。 • Camera.Front：前置摄像头。 • Camera.Back：后置摄像头。

closeCamera

关闭摄像头。

```
closeCamera(): void {}
```

openMicrophone

打开麦克风。

```
openMicrophone(
```

```
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

closeMicrophone

关闭麦克风。

```
closeMicrophone(
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

selectAudioPlaybackDevice

选择音频播放设备。
目前支持听筒、扬声器，在通话场景中，可以使用这个接口来开启/关闭免提模式。

```
selectAudioPlaybackDevice(
  device: AudioPlaybackDevice;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
device	AudioPlaybackDevice	听筒/扬声器。 • AudioPlaybackDevice.Earpiece：听筒。 • AudioPlaybackDevice.Speakerphone：扬声器。

setVideoRenderParams

设置用户视频画面的渲染模式。

```
setVideoRenderParams(
  userId: string;
  videoRender?: {
    fillMode?: FillMode;
    rotation?: Rotation;
  };
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
userId	string	目标用户的 userId。
fillMode	FillMode	视频画面渲染参数，填充模式。详情请参见 FillMode 。
rotation	Rotation	视频画面渲染参数，画面旋转角度。 • Rotation.Rotation_0：不旋转。 • Rotation.Rotation_90：顺时针旋转90度。 • Rotation.Rotation_180：顺时针旋转180度。 • Rotation.Rotation_270：顺时针旋转270度。

setVideoEncoderParams

设置视频编码的编码参数。
该设置能够决定远端用户看到的画面质量，同时也能决定云端录制出的视频文件的画面质量。

```
setVideoEncoderParams(
  videoEncoder?: {
    resolution?: Resolution;
    resolutionMode?: ResolutionMode;
  };
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
resolution	Resolution	视频编码的参数，分辨率。详情请参见 Resolution。
resolutionMode	ResolutionMode	视频编码的参数，视频宽高比。详情请参见 ResolutionMode。

setBeautyLevel

设置美颜级别，支持关闭默认美颜。

```
setBeautyLevel(
  level: number;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
level	number	美颜级别，取值范围 0 – 9； 0 表示关闭，9 表示效果最明显。

setSelfInfo

设置用户昵称、头像。
用户昵称不能超过500字节，用户头像必须是 URL 格式。

```
setSelfInfo(
  nickname: string;
  avatar?: string;
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
nickname	string	用户昵称。
avatar	string	用户头像（格式为 URL）。

enableMultiDeviceAbility

开启/关闭 TUICallEngine 的多设备登录模式（ 群组通话版套餐 支持）。

```
enableMultiDeviceAbility(
  enable: boolean;
  onSuccess?: (data: String) => void;
```

```
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

参数	类型	含义
enable	boolean	设置 TUICallEngine 是否开启多设备登录。

废弃接口

ignore

忽略当前通话，当您作为被叫收到 onCallReceived() 的回调时，可以调用该函数忽略来电，此时主叫会收到 onUserLineBusy 的回调。注意：v2.9+版本废弃，针对 calls 发起的通话不生效，建议使用 reject 接口。

```
ignore(
  onSuccess?: (data: String) => void;
  onError?: (errCode: number, errMsg: string) => void;
): void {}
```

TUICallEvent

最近更新时间：2025-06-25 15:58:52

TUICallEvent API 简介

TUICallEvent API 是音视频通话组件的**事件接口**。

API	含义
TUICallEvent.onError	通话过程中错误回调。
TUICallEvent.onCallReceived	通话请求的回调。
TUICallEvent.onCallCancelled	通话取消的回调。
TUICallEvent.onCallBegin	通话接通的回调。
TUICallEvent.onCallEnd	通话结束的回调。
TUICallEvent.onUserReject	xxxx 用户拒绝通话的回调。
TUICallEvent.onUserNoResponse	xxxx 用户不响应的回调。
TUICallEvent.onUserLineBusy	xxxx 用户忙线的回调。
TUICallEvent.onUserJoin	xxxx 用户加入通话的回调。
TUICallEvent.onUserLeave	xxxx 用户离开通话的回调。
TUICallEvent.onCallMediaTypeChanged	通话的媒体类型发生改变的回调。
TUICallEvent.onKickedOffline	当前用户被移下线。
TUICallEvent.onUserSigExpired	在线时票据过期。
TUICallEvent.onUserVideoAvailable	xxx 用户是否有视频流的回调。
TUICallEvent.onUserAudioAvailable	xxx 用户是否有音频流的回调。
TUICallEvent.onUserVoiceVolumeChanged	所有用户音量大小的反馈回调。
TUICallEvent.onUserNetworkQualityChanged	所有用户网络质量的反馈回调。

事件详情

onError

错误回调。

```
TUICallKit.on(TUICallEvent.onError, (res: any) => {
  console.log('onError code=' + res.code + ',message=' + res.message);
});
```

参数	类型	含义
errorCode	Number	错误码。
message	String	错误信息。

onCallReceived

收到一个新的来电请求回调。

```
TUICallKit.on(TUICallEvent.onCallReceived, (res: any) => {
  console.log('onCallReceived callerId=' + res.callerId);
});
```

参数	类型	含义
callerId	String	主叫 ID（邀请方）。
calleeIdList	Array<String>	被叫 ID 列表（被邀请方）。
groupId	String	群组通话 ID。
callMediaType	Number	通话的媒体类型，例如视频通话、语音通话。 • params.callMediaType = 0：语音通话。 • params.callMediaType = 1：视频通话。
userData	String	用户增加的扩展字段。

onCallCancelled

表示此次通话主叫取消、被叫超时、拒接等，涉及多个场景。

- **主叫取消：主叫收到该回调（callerId 为自己）；被叫收到该回调（callerId 为主叫的 ID）**
- **被叫超时：主叫会同时收到 onUserNoResponse 和 onCallCancelled 回调（callerId 是自己的 ID）；被叫收到 onCallCancelled 回调（callerId 是自己的 ID）**
- **被叫拒接：主叫会同时收到 onUserReject 和 onCallCancelled 回调（callerId 是自己的 ID）；被叫收到 onCallCancelled 回调（callerId 是自己的 ID）**
- **被叫忙线：主叫会同时收到 onUserLineBusy 和 onCallCancelled 回调（callerId 是自己的 ID）**
- **异常中断：被叫接收通话失败，收到该回调（callerId 是自己的 ID）**

```
TUICallKit.on(TUICallEvent.onCallCancelled, (res: any) => {
  console.log('onCallCancelled userId=' + res.callerId);
});
```

参数	类型	含义
callerId	String	主叫 ID（邀请方）。

onCallBegin

表示通话接通，主叫和被叫都可以收到，您可以通过监听这个事件来开启云端录制、内容审核等流程。

```
TUICallKit.on(TUICallEvent.onCallBegin, (res: any) => {
  console.log('onCallBegin strRoomId=' + res.roomId.strRoomId + ', callMediaType=' + res.callMediaType
+ ',callRole=' + res.callRole);
});
```

参数	类型	含义
roomId	RoomId	• roomId.intRoomId：此次通话的音视频房间 ID (int 类型)。 • roomId.strRoomId：此次通话的音视频房间 ID (String 类型)。
callMediaType	Number	通话的媒体类型，视频通话、语音通话。 • params.callMediaType = 0：语音通话。 • params.callMediaType = 1：视频通话。
callRole	Number	角色，枚举类型：主叫、被叫。 • params.callRole = 0：未知类型。

		• params.callRole = 1：主叫（邀请方）。 • params.callRole = 2：被叫（被邀请方）。

onCallEnd

表示通话挂断，主叫和被叫都可以收到，您可以通过监听这个事件来显示通话时长、通话类型等信息，或者来停止云端的录制流程。

```
TUICallKit.on(TUICallEvent.onCallEnd, (res: any) => {
  console.log('onCallEnd strRoomId=' + res.roomId.strRoomId
  + ',callMediaType=' + res.callMediaType
  + ',callRole=' + res.callRole
  + ',totalTime=' + res.totalTime);
});
```

参数	类型	含义
roomId	RoomId	• roomId.intRoomId: 此次通话的音视频房间 ID (int 类型)。 • roomId.strRoomId: 此次通话的音视频房间 ID (String 类型)。
callMediaType	Number	通话的媒体类型，视频通话、语音通话。 • params.callMediaType = 0：语音通话。 • params.callMediaType = 1：视频通话。
callRole	Number	角色，枚举类型：主叫、被叫。 • params.callRole = 0：未知类型。 • params.callRole = 1：主叫（邀请方）。 • params.callRole = 2：被叫（被邀请方）。
totalTime	Number	此次通话的时长，单位：秒。

onUserReject

通话被拒绝的回调，在1v1 通话中，只有主叫方会收到拒绝回调，在群组通话中，所有被邀请者都可以收到该回调。

```
TUICallKit.on(TUICallEvent.onUserReject, (res: any) => {
  console.log('onUserReject userId=' + res.userId);
});
```

参数	类型	含义
userId	String	拒绝用户的 ID。

onUserNoResponse

对方无回应的回调。

```
TUICallKit.on(TUICallEvent.onUserNoResponse, (res: any) => {
  console.log('onUserNoResponse userId=' + res.userId);
});
```

参数	类型	含义
userId	String	无响应用户的 ID。

onUserLineBusy

通话忙线回调。

```
TUICallKit.on(TUICallEvent.onUserLineBusy, (res: any) => {
  console.log('onUserLineBusy userId=' + res.userId);
});
```

参数	类型	含义
userId	String	拒绝用户的 ID。

onUserJoin

有用户进入此次通话的回调。

```
TUICallKit.on(TUICallEvent.onUserJoin, (res: any) => {
  console.log('onUserJoin userId=' + res.userId);
});
```

参数	类型	含义
userId	String	加入当前通话的用户 ID。

onUserLeave

有用户离开此次通话的回调。

```
TUICallKit.on(TUICallEvent.onUserLeave, (res: any) => {
  console.log('onUserLeave userId=' + res.userId);
});
```

参数	类型	含义
userId	String	离开当前通话的用户 ID。

onCallMediaTypeChanged

表示通话的媒体类型发生变化。

```
TUICallKit.on(TUICallEvent.onCallMediaTypeChanged, (res: any) => {
  console.log('onCallMediaTypeChanged oldCallMediaType=' + res.oldCallMediaType + ',newCallMediaType='
+ res.newCallMediaType);
});
```

参数	类型	含义
oldCallMediaType	MediaType	旧的通话类型。
newCallMediaType	MediaType	新的通话类型。

onKickedOffline

当前用户被移下线：此时可以 UI 提示用户，并重新调用初始化。

```
TUICallKit.on(TUICallEvent.onKickedOffline, (res: any) => {
  console.log('onKickedOffline');
});
```

onUserSigExpired

在线时票据过期：此时您需要生成新的 userSig，并重新调用初始化。

```
TUICallKit.on(TUICallEvent.onUserSigExpired, (res: any) => {
  console.log('onUserSigExpired');
});
```

onUserVideoAvailable

用户是否开启视频上行回调。

```
TUICallKit.on(TUICallEvent.onUserVideoAvailable, (res: any) => {
  console.log('onUserVideoAvailable userId=' + res.userId + 'isVideoAvailable=' +
res.isVideoAvailable);
});
```

参数	类型	含义
userId	String	通话用户 ID。
isVideoAvailable	boolean	用户视频是否可用。

onUserAudioAvailable

用户是否开启音频上行回调。

```
TUICallKit.on(TUICallEvent.onUserAudioAvailable, (res: any) => {
  console.log('onUserAudioAvailable userId=' + res.userId + 'isAudioAvailable=' +
res.isAudioAvailable);
});
```

参数	类型	含义
userId	String	用户 ID。
isAudioAvailable	boolean	用户音频是否可用。

onUserVoiceVolumeChanged

用户通话音量的回调。

```
TUICallKit.on(TUICallEvent.onUserVoiceVolumeChanged, (res: any) => {
  console.log('onUserVoiceVolumeChanged', res)
});
```

参数	类型	含义
volumeMap	any	音量表，根据每个 userId 可以获取对应用户的音量大小，音量最小值为0，音量最大值为100。

onUserNetworkQualityChanged

用户网络质量的回调。

```
TUICallKit.on(TUICallEvent.onUserNetworkQualityChanged, (networkQuality: any) => {
 for (const [key, value] of networkQuality) {
    console.log(`onUserNetworkQualityChanged userId: ${key}, network quality: ${value}`);
  }
```

```
});
```

参数	类型	含义
networkQuality	any	网络状态，根据每个 userId 可以获取对应用户当前的网络质量。 • value = Unknown • value = Excellent • value = Good • value = Poor • value = Bad • value = Vbad • value = Down

关键类型

最近更新时间：2025-06-25 15:45:22

CallParams

在 Tencent RTC 控制台通话扩展参数，例如：房间号、通话邀请超时时间，离线推送自定义内容等。

参数	类型	是否必填	含义
roomId	RoomId	否	房间 ID，详情请参考 RoomId。
timeout	Number	否	通话超时时间，默认：40s，单位：秒。timeout = 0，设置为不超时。
userData	String	否	发起通话时自定义扩展字段。
offlinePushInfo	OfflinePushInfo	否	自定义离线消息推送参数。
chatGroupId	String	否	与 chat 结合使用时， chat 群组的群 ID。

RoomId

房间 ID，分为 String 类型和 Number 类型。

参数	类型	是否必填	含义
strRoomId	String	否	房间 ID，String 类型。
intRoomId	Number	否	房间 ID，Number 类型。

OfflinePushInfo

厂商离线推送配置信息。

参数	类型	是否必填	含义
title	String	否	离线推送展示通知栏标题。
desc	String	否	离线推送展示通知栏内容。
ignoreIOSBadge	boolean	否	离线推送忽略 badge 计数（仅对 iOS 生效）， 如果设置为 true，在 iOS 接收端，这条消息不会使 App 的应用图标未读计数增加。
iOSSound	String	否	离线推送声音设置（仅对 iOS 生效）。 • 当 sound = IOS_OFFLINE_PUSH_NO_SOUND，表示接收时不会播放声音。 • 当 sound = IOS_OFFLINE_PUSH_DEFAULT_SOUND，表示接收时播放系统声音。如果要自定义 iOSSound，需要先把语音文件链接进 Xcode 工程，然后把语音文件名（带后缀）设置给 iOSSound。
androidSound	String	否	离线推送声音设置（仅对 Android 生效，IMSDK 6.1 及以上版本支持）。只有华为和谷歌手机支持设置声音提示，小米手机设置声音提示，请您参见 小米自定义铃声 。另外，谷歌手机 FCM 推送在 Android 8.0 及以上系统设置声音提示，必须调用 setAndroidFCMChannelID 设置好 channelID，才能生效。
androidOPPOChannelID	String	否	离线推送设置 OPPO 手机 8.0 系统及以上的渠道 ID。
androidVIVOClassification	Number	否	VIVO 推送消息分类 (待废弃接口，vivo 推送服务于 2023 年 4 月 3 日优化消息分类规则，推荐使用 setAndroidVIVOCategory 设置消息类别)。 • 0：运营消息。

			• 1：系统消息，默认取值为 1。
androidXiaoMiChannelID	String	否	小米手机 8.0 系统及以上的渠道 ID。
androidFCMChannelID	String	否	FCM 通道手机 8.0 系统及以上的渠道 ID。
androidHuaWeiCategory	String	否	华为推送消息分类，详细参见：华为消息分类标准 。
isDisablePush	boolean	否	是否关闭推送（默认开启推送）。
iOSPushType	IOSOfflinePushType	否	iOS 离线推送类型，默认：APNs。

IOSOfflinePushType

iOS 离线推送类型。

类型	含义
APNs	普通推送。
VoIP	VoIP 推送。

MediaType

通话类型。

类型	含义
Audio	语音通话。
Video	视频通话。

Camera

前置/后置 摄像头

类型	含义
Front	前置摄像头。
Back	后置摄像头。

AudioPlaybackDevice

音频设备

类型	含义
Earpiece	听筒。
Speakerphone	扬声器。

FillMode

视频画面的渲染参数

类型	含义
Fill	填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。

Fit	适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

Rotation

视频画面旋转方向

类型	含义
Rotation_0	不旋转。
Rotation_90	顺时针旋转90度。
Rotation_180	顺时针旋转180度。
Rotation_270	顺时针旋转270度。

Resolution

视频分辨率

类型	含义
Resolution_640_360	宽高比 16:9；分辨率 640x360；建议码率（VideoCall）500kbps。
Resolution_640_480	宽高比 4:3；分辨率 640x480；建议码率（VideoCall）600kbps。
Resolution_960_540	宽高比 16:9；分辨率 960x540；建议码率（VideoCall）850kbps。
Resolution_960_720	宽高比 4:3；分辨率 960x720；建议码率（VideoCall）1000kbps。
Resolution_1280_720	宽高比 16:9；分辨率 1280x720；建议码率（VideoCall）1200kbps。
Resolution_1920_1080	宽高比 16:9；分辨率 1920x1080；建议码率（VideoCall）2000kbps。

ResolutionMode

视频宽高比模式

类型	含义
Landscape	横屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Landscape = 640 × 360。
Portrait	竖屏分辨率，例如：Resolution.Resolution_640_360 + ResolutionMode.Portrait = 360 × 640。

版权所有：腾讯云计算（北京）有限责任公司

错误码表（TUICallKit）

最近更新时间：2025-02-26 17:34:12

通知用户在使用音视频通话过程中出现的警告和错误。

TUICallDefine 错误码

定义	取值	描述
ERROR_PACKAGE_NOT_PURCHASED	-1001	您当前未购买音视频通话能力套餐，请前往 IM 控制台 开通免费体验 或 加购正式版 。
ERROR_PACKAGE_NOT_SUPPORTED	-1002	您当前购买的音视频通话能力套餐不支持该功能，建议您前往 IM 控制台 升级套餐类型 。
ERROR_TIM_VERSION_OUTDATED	-1003	IM SDK 版本过低，请升级 IM SDK 版本 ≥ 2.21.2（仅 web、小程序平台支持）。
ERROR_PERMISSION_DENIED	-1101	获取权限失败，当前未授权音/ 视频权限，请查看是否开启设备权限。
ERROR_INIT_FAIL	-1201	未调用 init，TUICallEngine API 使用需在 init 之后, 初始化失败。
ERROR_PARAM_INVALID	-1202	参数错误（参数格式、类型等错误）。
ERROR_REQUEST_REFUSED	-1203	当前状态不支持调用。例如： 需要在空闲状态才可使用的 API，在呼叫中/通话中时调用。
ERROR_REQUEST_REPEATED	-1204	当前方法正在执行中，请勿重复调用。
ERROR_SCENE_NOT_SUPPORTED	-1205	当前通话场景，不支持该功能。
ERROR_SIGNALING_SEND_FAIL	-1401	信令发送失败，具体失败原因可查看具体方法回调中的 errorMsg。
FAILURE_SWITCH_TO_AUDIO	60001	视频切语音失败（仅 web 平台支持）。
FAILURE_SWITCH_TO_VIDEO	60002	语音切视频失败（仅 web 平台支持）。
NOT_MICROPHONE_LIST	60003	没有可用的麦克风设备（仅 web 平台支持）。
NOT_CAMERA_LIST	60004	没有可用的摄像头设备（仅 web 平台支持）。
NOT_SUPPORT	60005	当前页面不具备 webRTC 能力（仅 web 平台支持）。
UNKNOWN	65535	未知错误（仅 web、小程序平台支持）。

IM 错误码

音视频通话使用了腾讯云 IM SDK 作为通信的基础服务，例如通话拨打信令、通话忙线信令等核心逻辑。常见错误码如下：

错误码	描述
6014	IM SDK 未登录，请先登录，成功回调之后重试，或者已被移下线，可使用 TIMManager getLoginUser 检查当前是否在线。

6017	参数无效，请检查参数是否符合要求，具体可查看错误信息进一步定义哪个字段。
6206	UserSig 过期，请重新获取有效的 UserSig 后再重新登录，获取方法请参见 生成 UserSig 。
7013	套餐包不支持该接口的使用，请升级到旗舰版套餐。
8010	信令请求 ID 无效或已经被处理过。
8021	呼叫的用户还没有登录过 TIM，导致呼叫失败。

说明：
更多 IM SDK 错误码详见：IM 错误码 。

TRTC 错误码

音视频通话使用了腾讯云 TRTC SDK 作为音频和视频的基础服务，例如开关摄像头、开关麦克风等核心逻辑。常见错误码如下（Web TRTC SDK 错误码详见：TRTC 错误码 ）：

枚举	取值	描述
ERR_CAMERA_START_FAIL	-1301	打开摄像头失败，例如在 Windows 或 Mac 设备，摄像头的配置程序（驱动程序）异常，禁用后重新启用设备，或者重启机器，或者更新配置程序。
ERR_CAMERA_NOT_AUTHORIZED	-1314	摄像头设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。
ERR_CAMERA_SET_PARAM_FAIL	-1315	摄像头参数设置出错（参数不支持或其它）。
ERR_CAMERA_OCCUPY	-1316	摄像头正在被占用中，可尝试打开其他摄像头。
ERR_MIC_START_FAIL	-1302	打开麦克风失败，例如在 Windows 或 Mac 设备，麦克风的配置程序（驱动程序）异常，禁用后重新启用设备，或者重启机器，或者更新配置程序。
ERR_MIC_NOT_AUTHORIZED	-1317	麦克风设备未授权，通常在移动设备出现，可能是权限被用户拒绝了。
ERR_MIC_SET_PARAM_FAIL	-1318	麦克风设置参数失败。
ERR_MIC_OCCUPY	-1319	麦克风正在被占用中，例如移动设备正在通话时，打开麦克风会失败。
ERR_TRTC_ENTER_ROOM_FAILED	-3301	进入房间失败，请查看 onError 中的 -3301 对应的 msg 提示确认失败原因。

说明：
更多 TRTC SDK 错误码详见：TRTC 错误码 。Web TRTC SDK 错误码详见：TRTC 错误码 。

Callkit 服务端错误码

新版本 Callkit RestAPI 错误码，例如获取通话状态和通话记录。常见错误码如下：

错误码	描述
101001	服务器内部错误，请重试
101002	参数无效，请检查参数是否符合要求，具体可查看错误信息进一步定义哪个字段。
101004	通话不存在，或者曾经存在过，但是目前已经结束
101050	通话记录不存在

发布日志（TUICallKit）
uni-app（小程序）

最近更新时间：2025-06-30 13:32:21

> **注意：**
> 1. **新版本（4.0.2）涉及 包名调整为** @tencentcloud/call-uikit-wx-uniapp **：**
>   - **对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。**
> 2. @tencentcloud/call-uikit-wechat **≥ V3.0.4 采用新架构进行重构，不再使用微信原生进行开发，而是采用 Vue（支持 Vue2.6、Vue3）开发的组件。目前暂时无法在微信开发者工具中引入组件直接使用，需要在 uniapp 项目中打包小程序使用。**
> 3. 如果想使用原生小程序 @tencentcloud/call-uikit-wechat 组件，需要低于 V3.0.4。低版本的组件会针对发现的 bug 继续升级更新维护。

Version 4.0.7 @2025.06.06

新增

- 支持 REST API 结束通话。
- 邀请他人时，将通话中的用户通过 inviteeList 通知给被叫。
- 增加对通过 REST API 销毁房间的处理。
- 增加 calls 网络中断恢复后的兼容性处理。

优化

- 优化和 chat tuikit 融合时的中途加入 join 功能。
- 优化端上长时间连续心跳失败（断网/超时）结束通话逻辑。
- 优化异常中断（关闭浏览器/刷新浏览器/小程序左滑/右上角退出）等操作，立刻结束通话。
- 优化被叫 reject 接口逻辑，移除退房操作。
- 优化排障日志，增加在线日志。
- 优化 callkit 事件监听，将监听 USER_ACCEPT 调整为 ON_CALL_BEGIN。
- 中途加入，默认关闭摄像头。
- 多人通话下，主叫默认大屏。

修复

- 修复中途加入未更新 groupId 问题。
- 修复通话记录上报 groupId 缺失问题。
- 修复记录在中途加入和邀请他人场景下，参与通话成员不准确问题。
- 修复主动加入通话时，异常回调 onUserLeave 问题。

Version 4.0.4 @2025.03.31

新增

- 新增 onCallNotConnected 事件。
- 新增 onUserInviting 事件。
- 新增 chatGroupID 字段在 ON_CALL_RECEIVED、CALLING_END、ON_CALL_BEGIN 事件。
- 增加在线排障日志。

优化

- 优化多端登录问题。
- 优化通过 trtc restAPI 解散房间的逻辑。
- 优化本地日志打印。

版权所有：腾讯云计算（北京）有限责任公司

修复

- 修复日志上报问题（字段缺失导致 kibana 没有记录）。
- 修复 setSelfInfo 接口无效问题。
- 修复 calls 能力位校验码异常问题。
- 修复被叫都忙线时，主叫呼叫被叫未抛出 onCallCanceled 事件问题。
- 移除 trtc 退房抛出的 userLeave 事件。

Version 4.0.3 @2025.03.13

修复

当所有的被叫都忙线时，engine 没有抛出 cancel 事件时，callkit 根据远端用户列表结束通话。

Version 4.0.2 @2025.02.18

新增

- 新增 calls 接口，支持发起单人通话或多人通话，更加灵活的通话成员管理，支持更强大的 REST API，欢迎使用。
- 新增 join 接口，配合 calls 接口、支持加入已有通话。

Version 3.3.9 @2024.10.21

优化

- 切换语言后能够动态改变组件文案。
- 小程序体积优化（清除多语言支持；图片压缩等）。

修复

- 修复选人组件未清除上次选中用户导致重复邀请的问题。
- 修复中途邀请他人 inviteUser 报错时，通话界面消失问题。

Version 3.3.6 @2024.09.13

优化

优化上屏消息展示逻辑，对齐微信消息上屏。

Version 3.3.4 @2024.08.22

修复

修复 TUICallKit 在与 TUIKit 融合分包场景下无法初始化的问题。

Version 3.3.3 @2024.08.06

优化

- 优化进房流程：进房成功后发信令。
- 优化铃声实例的创建、销毁时机。
- 优化关闭摄像头后，翻转摄像头和虚拟背景按钮的交互。
- 优化用户未 init 时，直接调用 call/groupCall API 时的报错内容。
- 优化用户未 init 时，在页面引入 TUICallKit 组件，控制台的 setVideoResolution 警告现象。

修复

- 修复铃声被中断(切后台、滑动页面等)后的异常响铃问题。
- 修复微信 App 无权限时的进房失败问题。

Version 3.3.2 @2024.07.12

新增

- 新增支持邀请他人按钮显示隐藏。
- 网络状态较差时的提示。
- 和 chat 融合，callkit 控制自身显隐。
- 增加 点击 “接听” 的日志上报。

优化

- 样式作用域优化，避免影响用户样式。
- 优化虚拟背景默认虚化等级，默认虚化等级调整：1 -> 3。

修复

- 修复中途加入的已知问题。
- 修复 roomId、roomIdType 更新问题。

Version 3.3.1 @2024.06.25

修复

- 修复中途加入功能的一些问题。
- 删除 debug 文件的注释。

Version 3.3.0 @2024.06.14

新增

- 新增支持自定义大小窗显示和设置摄像头初始状态功能。
- 新增没有 live-pusher 推流能力时的弹窗提示。

优化

- 扩展 offlinePushInfo 参数，支持离线推送声音设置等功能。
- 优化全局监听下初始化失败后的异常抛出。

修复

- 修复被叫接通后，立刻自己超时后通话异常问题。
- 修复群通话被叫等待接听页面远端用户昵称显示问题。

Version 3.2.9 @2024.05.29

优化

UI 定制化接口增加日志上报。

Version 3.2.8 @2024.05.27

修复

修复 SDK 引入 ref 路径错误问题。

Version 3.2.7 @2024.05.17

新增

- 新增中途加入 UI 组件。
- 新增自定义 UI 接口，支持设置通话背景和按钮隐藏。
- 调整发起通话时的参数校验，支持字符串房间号。

Version 3.2.5 @2024.05.10

新增

视频通话支持背景虚化，详细参见：虚拟背景 。

Version 3.2.4 @2024.05.06

修复

- 修复群通话切换摄像头按钮问题。
- 优化按钮点击后，应用卡住，再次点击按钮引起的异常问题。

Version 3.2.3 @2024.04.19

优化

优化异常退出时（左滑、后退等）的逻辑处理。

Version 3.2.2 @2024.03.25

新增

新增全新的 UI 视效，功能更清晰，体验更好。

优化

优化使用 TUICallKit 的数据上报。

Version 3.2.1 @2024.03.08

新增

language 日志上报。

Version 3.2.0 @2024.02.23

新增

增加默认的离线推送参数。

修复

修复群组通话没有昵称问题。

Version 3.1.9 @2024.01.30

修复

- 修复群组通话没有展示用户信息问题。
- 修复选人组件在没有可选成员时，“确认”按钮仍然可点击的问题。
- 修复小程序在切换到后台后通话计时暂停，导致通话时长不准确的问题。

Version 3.1.8 @2024.01.19

修复

修复选人组件样式对页面的影响。

Version 3.1.7 @2024.01.12

修复

- 修复远端关闭摄像头和麦克风之后的报错问题。
- 修复群组通话中，进房用户没有流时的报错问题。
- 修复悬浮窗偏移及层级样式问题。

Version 3.1.6 @2023.12.29

优化

- 优化群组通话下的提示信息。
- 优化昵称过长时的显示问题。

修复

- 修复小程序 setSelfInfo 接口传参问题。
- 修复 destroyed 存在的问题。
- 修复不同呼叫场景下悬浮窗里的挂断问题。
- 修复主叫呼叫状态下显示远端问题。
- 修复 pc 上填充不全的样式问题。

Version 3.1.4 @2023.12.01

新增

融合到 Chat 中增加 isFromChat 上报。

修复

修复按钮 loading 下可点击的问题。

Version 3.1.3 @2023.11.17

新增

接口增加参数校验。

Version 3.1.2 @2023.11.03

新增

- 新增邀请他人 inviteUser 功能。
- 新增中途加人 joinInGroupCall 功能。
- 新增被叫来电铃声静音 enableMuteMode 功能。

修复

修复远端流麦克风状态展示不对问题。

Version 3.1.1 @2023.10.27

修复

修复部分 iOS 机型上按钮被隐藏问题。

Version 3.1.0 @2023.10.20

新增

- 增加悬浮窗功能。
- 增加 enableFloatWindow 接口，用于开启/关闭悬浮窗功能。
- 增加呼叫黑名单用户失败提示信息。
- 增加对日语的支持。

优化

视频通话下，大屏默认显示远端用户。

版权所有：腾讯云计算（北京）有限责任公司

Version 3.0.8 @2023.10.10

新增

增加版本号、框架等信息上报。

Version 3.0.7 @2023.10.08

新增

增加桌面端视频通话下，通话时长展示。

优化

- 优化桌面端上视频流预览圆角黑边问题。
- 优化远端流用户信息显示优先级：备注 > 昵称 > userId。
- 优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 3.0.6 @2023.09.19

修复

修复融合到 TUIKit 中的消息上屏问题。

Version 3.0.5 @2023.09.15

优化

优化 TUICallKit 之间的相互引用，避免在 uniapp 打包小程序出现的栈溢出问题。

新增

- 新增无套餐的弹框提示。
- 增加全局监听功能。

修复

- 修复 setCallingBell 被叫铃声被主叫铃声覆盖，导致铃声重复问题。
- 修复移动端上的样式问题。

Version 3.0.4 @2023-09-01

Breaking Change

升级底层依赖 tuicall-engine-wx 至 ^2.0.0，不再支持 tim-wx-sdk 创建 tim 实例，如需创建 tim 实例，请使用 @tencentcloud/chat 创建。

新增

支持在 uniapp 项目中以分包的方式集成。

Version 1.4.4 @2023-07-28

新增

新增 init 错误抛出处理。

Version 1.4.3 @2023-07-14

修复

调整事件监听时机，修复 SDK_READY 未抛出的问题。

Version 1.4.2 @2023-06-09

修复

- 修复全局监听卸载不彻底的问题。
- 修复全局监听本地预览取值不当的问题。

Version 1.4.1 @2023-05-25

新增

新增 call、groupCall 错误抛出处理。

Version 1.4.0 @2023-05-17

新增

新增在 tabBar 页面接入时的兼容性处理。

优化

- 优化在页面跳转情况下的初始化以及卸载逻辑。
- 优化全局监听逻辑，防止重复监听。

Version 1.3.9 @2023-05-11

新增

- 新增 call、groupCall 支持自定义数字房间号。
- 新增 call、groupCall 允许用户自定义 userData 数据。

修复

修复全局监听模式下，视频通话被叫预览黑屏导致无法接听的问题。

Version 1.3.8 @2023-04-27

> ⚠ **注意：**
> 需要依赖 tuicall-engine-wx 的版本 ⩾ v1.4.7。

新增

新增通话页面出现时的动画效果。

优化

- 优化通话界面唤起的时间。
- 优化用户异常退出的逻辑，如右滑退出以及返回退出的情况。

修复

修复用户重复点击，导致接口被重复调用的问题。以下接口增加节流处理： call，groupCall，accept，reject。

Version 1.3.7 @2023-04-14

新增

新增 设置日志级别接口 。

修复

修复铃声路径错误问题。

Version 1.3.6 @2023-03-31

新增

新增 index.js 入口文件，解决 npm 构建时的警告问题。

优化

- 优化用户没有 live-pusher 推流标签使用权限时，给予弹窗提示。
- 优化针对在微信开发者工具运行时，给予弹窗提示。

Version 1.3.5 @2023-03-24

新特性

支持自定义铃声。

修复

优化 reset 函数在生命周期中的触发时机，修复全局监听下被叫方的白屏问题。

Version 1.3.4 @2023-03-16

修复

全局监听下的视频通话，被叫方使用 iOS 设备预览黑屏问题。

优化

优化常见错误提示，方便客户快速定位问题。

Version 1.3.3 @2023-03-02

优化

- 弹窗昵称显示优先级（昵称 > userID）。
- 打印显示 TUICallKit 版本号。

Version 1.3.2 @2023-02-17

修复

打包路径错误。

Version 1.3.1 @2023-02-17

修复

修复远端关闭麦克风，没有及时更新远端流导致后续调用音量变化接口报错。

Version 1.3.0 @2023-02-13

优化

优化依赖引入，组件包体积由 900KB 降低至 110KB 左右。

Version 1.2.7 @2023-02-03

修复

去除 aegis.js 文件。

Version 1.2.6 @2023-01-03

修复

未及时回收 wx.$TUICallEngine 变量引发的 init 报错。

Version 1.2.5 @2023-01-31

新特性

支持 uniapp 小程序全局监听

修复

升级 tsignaling-wx，解决和客户端的互通问题。

Version 1.2.4 @2023-01-12

修复

- 打包路径错误。
- callManager 兼容 sdkAppID 与 SDKAppID 传参。
- callkit 生命周期初始化时 移除 wx.CallManager 监听报错问题。

Version 1.2.3 @2023-01-10

修复

TUICallKit 样式布局错误。

Version 1.2.1 @2023-01-09

修复

打包问题引起的 globalCall 页面引用错误。

Version 1.2.0 @2023-01-06

新特性

支持全局监听

- 主包方案详见【TUICallKit】小程序全局监听（主包方案）。
- 分包方案详见【TUICallKit】小程序全局监听（分包方案）。

Version 1.1.4 @2022-12-23

修复

- init 方法支持外部传入 tim。
- 修复 uni-app 打包小程序存在的栈溢出问题。

Version 1.1.3 @2022-12-14

新增

新增设备权限弹出窗口。

优化

群通话中的昵称显示优先级。

> **说明：**
> 群通话显示优先级为 **昵称 > userID**。

修复

群通话中 player 推流过程中 trtc-wx 报错问题。

Version 1.1.2 @2022-11-30

优化

优化接入流程。

Version 1.1.1 @2022-11-30

优化

兼容 chat，支持 TUICallKit 在 init 时传值。

Version 1.1.0 @2022.11.16

> ⚠ **注意：**
> 升级至该版本需注意：升级指引 。

优化

- 提高多人通话稳定性
- 抛出事件的数据结构调整，具体请参见 TUICallEvent 。

Web

最近更新时间：2025-07-01 09:51:42

说明：

1. **新版本（4.0.2）涉及 TUICallEngine 接口变更：**
   - **对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。**
2. TUICallKit Vue3 Github 版本地址：Github TUICallKit Web 。

@tencentcloud/call-uikit-react 发布日志从 **Version 3.3.0** 开始查看。

Version 4.0.7 @2025.06.06

新增

- 支持 REST API 结束通话。
- 邀请他人时，将通话中的用户通过 inviteeList 通知给被叫。
- 增加对通过 REST API 销毁房间的处理。
- 增加 calls 网络中断恢复后的兼容性处理。

优化

- 优化和 chat tuikit 融合时的中途加入 join 功能。
- 优化端上长时间连续心跳失败（断网/超时）结束通话逻辑。
- 优化异常中断（关闭浏览器/刷新浏览器/小程序左滑/右上角退出）等操作，立刻结束通话。
- 优化被叫 reject 接口逻辑，移除退房操作。
- 优化排障日志，增加在线日志。
- 优化 callkit 事件监听，将监听 USER_ACCEPT 调整为 ON_CALL_BEGIN。
- 中途加入，默认关闭摄像头。
- 多人通话下，主叫默认大屏。

修复

- 修复中途加入未更新 groupId 问题。
- 修复通话记录上报 groupId 缺失问题。
- 修复记录在中途加入和邀请他人场景下，参与通话成员不准确问题。
- 修复主动加入通话时，异常回调 onUserLeave 问题。

Version 4.0.4 @2025.03.31

新增

- 新增 onCallNotConnected 事件。
- 新增 onUserInviting 事件。
- 新增 chatGroupID 字段在 ON_CALL_RECEIVED、CALLING_END、ON_CALL_BEGIN 事件。
- 增加在线排障日志。

优化

- 优化多端登录问题。
- 优化通过 trtc restAPI 解散房间的逻辑。
- 优化本地日志打印。

修复

- 修复日志上报问题（字段缺失导致 kibana 没有记录）。

- 修复 setSelfInfo 接口无效问题。
- 修复 calls 能力位校验码异常问题。
- 修复被叫都忙线时，主叫呼叫被叫未抛出 onCallCanceled 事件问题。
- 移除 trtc 退房抛出的 userLeave 事件。

Version 4.0.3 @2025.03.13

修复

当所有的被叫都忙线时，engine 没有抛出 cancel 事件时，callkit 根据远端用户列表结束通话。

Version 4.0.2 @2025.02.18

新增

- 新增 calls 接口，支持发起单人通话或多人通话，更加灵活的通话成员管理，支持更强大的 REST API，欢迎使用。
- 新增 join 接口，配合 calls 接口、支持加入已有通话。

Version 3.3.9 @2024.10.21

优化

切换语言后能够动态改变组件文案。

修复

- 修复关闭摄像头重置前后置摄像头问题。
- 修复中途邀请他人 inviteUser 报错时，通话界面消失问题。

Version 3.3.8 @2024.09.27

新增

react 新增主题切换功能。

优化

react 优化设备列表挂载的父元素。

Version 3.3.7 @2024.09.13

修复

修复首次通话中关闭扬声器，下次通话扬声器仍处于关闭状态的问题。

Version 3.3.6 @2024.09.13

新增

react 增加 CallMessage 组件，用于和 chat 融合上屏消息展示。

优化

优化上屏消息展示逻辑，对齐微信消息上屏。

Version 3.3.4 @2024.09.10

新增

react 新增 onMinimized 回调事件。

优化

- react 支持悬浮窗拖动。

- 优化移动端点击扬声器按钮交互，调整为开启/关闭扬声器。
- 优化 react 设备列表层级。

Version 3.3.3 @2024.08.06

新增

新增开启/关闭扬声器功能。

优化

- react：优化按钮 loading 效果。
- 优化铃声实例的创建、销毁时机。
- 优化关闭摄像头后，翻转摄像头和虚拟背景按钮的交互。
- 优化用户未 init 时，直接调用 call/groupCall API 时的报错内容。
- 优化用户未 init 时，在页面引入 TUICallKit 组件，控制台的 setVideoResolution 警告现象。

修复

- react：修复开虚拟背景按钮间距。
- react：修复昵称没有截断问题。

Version 3.3.2 @2024.07.12

新增

- 新增支持邀请他人按钮显示隐藏。
- 网络状态较差时的提示。

修复

- 设备选择后重新拨打，设备选择列表选中设备与当前设备不符。
- 设备列表为空时，不显示设备选择列表。
- 修复中途加入的已知问题。

Version 3.3.1 @2024.06.25

修复

- 修复中途加入功能的一些问题。
- 删除 debug 文件的注释。

Version 3.3.0 @2024.06.14

新增

- 新增支持自定义大小窗显示和设置摄像头初始状态功能。
- Vue&React：新增支持传入自定义字符串房间号。
- React：新增 react TUICallKit 虚拟背景功能。

优化

扩展 offlinePushInfo 参数，支持离线推送声音设置等功能。

修复

- 修复被叫接通后，立刻自己超时后通话异常问题。
- 修复群通话被叫等待接听页面远端用户昵称显示问题。

优化

UI 定制化接口增加日志上报。

Version 3.2.9 @2024.05.29

优化

UI 定制化接口增加日志上报。

Version 3.2.8 @2024.05.27

修复

修复 SDK 引入 ref 路径错误问题。

Version 3.2.7 @2024.05.17

新增

- 新增中途加入 UI 组件。
- 新增自定义 UI 接口，支持设置通话背景和按钮隐藏。
- 调整发起通话时的参数校验，支持字符串房间号。

Version 3.2.4 @2024.05.06

新增

视频通话支持背景虚化。

修复

- 修复 uni-app 打包 web 项目，image 加载 icon 失败问题。
- 修复群通话切换摄像头按钮问题。
- 优化按钮点击后，应用卡住，再次点击按钮引起的异常问题。

Version 3.2.3 @2024.04.19

新增

新增群通话支持前后置摄像头切换。

优化

优化 TUICallKit SDK Readme。

Version 3.2.2 @2024.03.25

新增

新增全新的 UI 视效，功能更清晰，体验更好。

优化

优化使用 TUICallKit 的数据上报。

新增

language 日志上报。

Version 3.2.1 @2024.03.08

新增

language 日志上报。

Version 3.2.0 @2024.02.23

新增

增加默认的离线推送参数。

修复

修复群组通话没有昵称问题。

Version 3.1.9 @2024.01.30

修复

- 修复群组通话没有展示用户信息问题。
- 修复选人组件在没有可选成员时，“确认”按钮仍然可点击的问题。
- 修复通话中关闭麦克风，再次通话不上行音频流问题（升级 trtc-cloud-js-sdk 到 v2.2.7+）。

Version 3.1.8 @2024.01.19

修复

修复选人组件样式对页面的影响。

Version 3.1.7 @2024.01.12

修复

- 增加接口的重试机制，修复播放时因找不到 dom 节点导致播放失败问题。
- 修复 PC 上设备列表选择样式问题。

Version 3.1.6 @2023.12.29

优化

- 优化群组通话下的提示信息。
- 优化昵称过长时的显示问题。

修复

- 修复语音通话请求摄像头权限问题。
- 修复 destroyed 存在的问题。
- 修复不同呼叫场景下悬浮窗里的挂断问题。
- 修复主叫呼叫状态下显示远端问题。
- 修复 pc 上填充不全的样式问题。

Version 3.1.5 @2023.12.15

新增

优化获取设备权限的时机，初始化时不再获取设备权限，使用 call 时才获取。

修复

修复 @tencentcloud/call-uikit-vue2 、@tencentcloud/call-uikit-vue2.6 组件没有声明文件问题。

Version 3.1.4 @2023.12.01

新增

融合到 Chat 中增加 isFromChat 上报。

修复

修复按钮 loading 下可点击的问题。

Version 3.1.3 @2023.11.17

新增

接口增加参数校验。

Version 3.1.2 @2023.11.03

新增

- 新增邀请他人 inviteUser 功能。
- 新增中途加人 joinInGroupCall 功能。
- 新增被叫来电铃声静音接口 enableMuteMode 功能。

修复

修复远端流麦克风状态展示不对问题。

Version 3.1.0 @2023.10.20

新增

- 增加悬浮窗功能。
- 增加 enableFloatWindow 接口，用于开启/关闭悬浮窗功能。
- 桌面端支持摄像头、麦克风切换设备。
- 增加呼叫黑名单用户失败提示信息。
- 增加对日语的支持。

优化

视频通话下，大屏默认显示远端用户。

Version 3.0.8 @2023.10.10

新增

增加版本号、框架等信息上报。

Version 3.0.7 @2023.10.08

新增

增加桌面端视频通话下，通话时长展示。

优化

- 优化桌面端上视频流预览圆角黑边问题。
- 优化远端流用户信息显示优先级：备注 > 昵称 > userId。
- 优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 3.0.6 @2023.09.19

修复

修复融合到 TUIKit 中的消息上屏问题。

Version 3.0.5 @2023.09.15

优化

优化 TUICallKit 之间的相互引用，避免在 uniapp 打包小程序出现的栈溢出问题。

新增

新增桌面端设备无权限时的提示，引导客户如何对设备进行授权。

修复

- 修复 setCallingBell 被叫铃声被主叫铃声覆盖，导致铃声重复问题。
- 修复移动端上的样式问题。

Version 3.0.4 @2023.09.01

修复

- 修复 setCallingBell 作用对象为来电铃声（被叫铃声）。
- 修复 destroyed 报错问题。
- 修复错误弹框提示没有中英文。
- 修复 1v1 通话关闭摄像头后, 无法切换大小屏。

Version 3.0.3 @2023.8.25

新增

新增 @tencentcloud/call-uikit-vue2.6 适配 Vue2.6 版本。

优化

- 优化组件默认语言，调整为系统默认的语言。
- 优化打印的日志信息。
- 优化 tuicall-engine-webrtc 抛出错误的信息提示。
- 优化组件销毁后，对所用资源的清理。

修复

- 修复 videoDisplayMode 、videoResolution 挂断后再次呼叫时，未生效问题。
- 修复通话中， statusChanged 未抛出问题。
- 修复 init 多次调用问题。
- 修复通话中，关闭摄像头无法切换大小屏问题。

Version 3.0.2 @2023.8.14

修复

- 修复 H5 端被叫组件的样式问题。
- 修复切换小窗后，再次通话出现的样式问题。

Version 3.0.1 @2023.8.8

修复

修复群通话主叫方本地预览失败的问题，修改组件层对数据层默认读取方式。

Version 3.0.0 @2023.8.4

Breaking Change

升级底层依赖 tuicall-engine-webrtc 至 ^2.0.0，不再支持 tim-js-sdk 创建 tim 实例，如需创建 tim 实例，请使用 @tencentcloud/chat 创建。

新增

新增自定义铃声功能 setCallingBell 。

Version 2.4.2 @2023.11.03

新增

- 新增邀请他人 inviteUser 功能。
- 新增中途加人 joinInGroupCall 功能。
- 新增被叫来电铃声静音接口 enableMuteMode 功能。

修复

修复远端流麦克风状态展示不对问题。

Version 2.4.0 @2023.10.20

新增

- 增加悬浮窗功能。
- 增加 enableFloatWindow 接口，用于开启/关闭悬浮窗功能。
- 桌面端支持摄像头、麦克风切换设备。
- 增加呼叫黑名单用户失败提示信息。
- 增加对日语的支持。

优化

视频通话下，大屏默认显示远端用户。

Version 2.3.9 @2023.10.10

新增

增加版本号、框架等信息上报。

Version 2.3.8 @2023.10.08

新增

增加桌面端视频通话下，通话时长展示。

优化

- 优化桌面端上视频流预览圆角黑边问题。
- 优化远端流用户信息显示优先级：备注 > 昵称 > userId。
- 优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 2.3.6 @2023.09.15

优化

优化 TUICallKit 之间的相互引用，避免在 uniapp 打包小程序出现的栈溢出问题。

新增

新增桌面端设备无权限时的提示，引导客户如何对设备进行授权。

修复

- 修复 setCallingBell 被叫铃声被主叫铃声覆盖，导致铃声重复问题。
- 修复移动端上的样式问题。

Version 2.3.5 @2023.9.5

修复

修复进房前摄像头、麦克风按钮默认打开的问题。

Version 2.3.4 @2023.9.1

新增

新增视频通话接听前可以关闭或者开启摄像头功能。

修复

- 修复 1v1 通话关闭摄像头后, 无法切换大小屏问题。
- 修复从视频通话切换到语音通话 statusChanged 未抛出问题。

Version 2.3.3 @2023.8.22

修复

- 修复 videoDisplayMode 、videoResolution 挂断后再次呼叫时，未生效问题。
- 修复通话中， statusChanged 未抛出问题。

Version 2.3.2 @2023.7.26

Breaking Change

- 移除悬浮窗 TUICallKitMini 组件，合并到 TUICallKit 组件中。
- 抛出的 @kicked-out 事件调整为绑定回调 `:kickedOut` 的方式。
- 抛出的 @status-changed 事件调整为绑定回调 `:statusChanged` 的方式。

新增

- 新增通话页面出现时的动画效果。
- 新增 H5 上群组通话布局。

优化

- 优化通话中的问题提示信息、提示方式。
- 优化在 H5 页面的支持、交互。
- 优化通话界面唤起的时间。
- 优化 @tencentcloud/call-uikit-vue 包目录结构。

修复

- 修复接通后立刻挂断等边界操作下的通话问题。
- 修复 H5 在部分机型、浏览器上的样式问题。
- 修复重复点击引起的通话异常问题。

Version 2.2.1 @2023.7.7

新增

@tencentcloud/call-uikit-vue2 新增 Vue 版本的检测及提示。

修复

- 修复来电页面重复点击“接听”按钮导致接听失败。

Version 2.2.0 @2023.6.30

新增

- call 、groupCall 支持自定义数字房间号 roomID 参数。
- call 、groupCall 支持自定义扩展字段 userData 参数（用于在邀请信令中增加扩展信息）。
- 新增 setSelfInfo 接口，支持用户设置昵称和头像。

Version 2.1.0 @2023.4.14

新增

- 在 H5 模式语音通话下，呼叫时支持显示对方昵称。
- 发起通话失败时，会在呼叫页面显示“发起通话失败”。
- 接听通话失败时，会在来电页面显示“接听失败”。
- 支持监听当前用户是否被踢出（如被挤出登录），详见 TUICallKit 方法 - @kicked-out。
- 支持监听 TUICallKit 通话状态，详见 TUICallKit 方法 - @status-changed。
- 支持业务侧代码控制通话的接听、取消和挂断，详见 更多特性 - 通过接口设置自动接听。
- Vue2 版本新增 TypeScript 类型声明文件，在 TypeScript 项目中可以正常编译类型。

修复

- 修复在初始化组件时，控制台出现的更新个人资料接口的警告。
- 修复在 H5 模式下，被叫接听按钮出现背景图片错位的问题。

接口变更

`TUICallKitServer.destroy()` 增加调用限制，仅支持在未通话状态时调用。

Version 2.0.1 @2023.03.31

新增

- 优化 1v1 通话与多人通话视频的渲染逻辑，提升性能与稳定性。
- 优化 UI 展示，支持在 `TUICallKitServer.call()` 执行过程中显示对应 UI，即可以实现点击通话按钮立刻展示 `<TUICallKit/>` UI 组件。

修复

- 修复多人通话中昵称显示错误的问题。
- 修复 CSS 未限制生效范围导致污染全局样式的问题。

Version 2.0.0 @2023.03.21

新增

- 支持从 npm 引入已打包的 CallKit 文件。
- 支持 JavaScript 版本的 Vue 项目。
- 支持 Vue2 所有版本的 Vue 项目，适用于 Vue2 的 npm 包：call-uikit-vue2。

修复

修复无摄像头设备或权限导致通话发起异常的问题。

Version 1.4.2 @2023.03.03

新增

- 支持设置通话分辨率，详见 API 文档。
- 支持修改画面显示模式，详见 API 文档。
- 优化了接入步骤。
- 优化了错误抛出。

Version 1.4.1 @2023.02.13

新增

- 优化了本地摄像头的预览逻辑。
- 优化了远端视频流的渲染逻辑。

Version 1.4.0 @2023.01.06

新增

- 支持 Vue2.7+ 项目引入。
- 通话界面默认显示昵称，设置昵称请参考 TIM#updateMyProfile。

Version 1.3.3 @2022.12.27

新增

- 在 Basic Demo 中拨打电话时，新增了通话列表的空值检测。
- 在拨打电话时，Basic demo 中增加了 loading 图标。
- 优化了 Basic Demo 中设备检测的逻辑，手动跳过后不再主动弹出。
- 优化了组件图标的引用方式。
- 修改了默认的包管理工具为 npm。
- 优化了视频的渲染方式，减少了重复渲染的次数。

修复

修复了由于 vue-CLI 过时依赖导致的 Basic Demo 报错。

Version 1.3.2 @2022.12.07

新增

- 支持语言切换，接口详情可见 setLanguage 。
- 优化 Basic Demo 设备检测逻辑，手动跳过后不会再主动弹出。

修复

修复引入 `defineProps` 导致的警告。

Version 1.3.1 @2022.11.29

> ⚠ **注意：**
> 本版本依赖 SDK tuicall-engine-webrtc@1.2.1 版本，请及时更新。

新增

- 优化样式细节。
- 支持未接听对方通话时，监听对方修改通话类型。
- basic demo 增加设备检测功能。

修复

修复在挂断电话时内部逻辑导致的报错。

Version 1.3.0 @2022.11.14

新增

- 支持手机 H5 使用时，自动切换为竖屏样式。
- 支持在拨打电话时预览本地摄像头。
- basic demo 增加拨打电话前的设备检测。

修复

- 修复调用 TUICallKitServer.destroyed() 之后 tim 实例未完全退出的问题。
- 修复对方忙线时显示无应答的问题。
- 修复 vite 环境下 typescript 类型未成功打包的问题。

接口变更

主动调用 TUICallKitServer.call() 或 TUICallKitServer.groupCall() 时，如果遇到报错，不会再调用 beforeCalling 回调。请直接使用 try catch 捕获错误。

Version 1.2.0 @2022.11.03

新增

适配新版本 TUICallEngine SDK。

Version 1.1.0 @2022.10.21

新增

- 通话过程中，通话页面可以全屏。
- 通话过程中，可以使用 `<TUICallKitMini/>` 进行最小化。

修复

修复已知问题，提升稳定性。

Version 1.0.3 @2022.10.14

新增

basic demo 增加快捷复制 UserID、一键打开新窗口。

Version 1.0.2 @2022.09.30

新增

优化接入文档，增加演示图片和详细指引。

修复

- 修复首次进入房间时，设备状态位失效的问题。
- 修复在 webpack 工具打包时，图标偶现加载失败的问题。
- 修复已知样式问题。

Version 1.0.1 @2022.09.26

新增

在拨打电话时，隐藏对方麦克风图标。

修复

修复 basic demo SDKAppID 输入框应为数字的问题。

Version 1.0.0 @2022.09.23

- TUICallKit demo 快速跑通
- TUICallKit 快速接入
- TUICallKit API
- TUICallKit 界面定制指引
- TUICallKit (Web) 常见问题

Android&iOS

最近更新时间：2025-06-26 10:12:42

新版本（2.9.0.1230）涉及 TUICallEngine 接口变更：

- 如果您在开发过程中，更新Android代码到最新后，出现编译错误，可以使用如下方式解决：
  - 替换 "com.tencent.qcloud.tuikit.tuicallengine" 为 "com.tencent.cloud.tuikit.engine.call"。
  - 替换 "com.tencent.qcloud.tuikit.TUICommonDefine" 为 "com.tencent.cloud.tuikit.engine.common.TUICommonDefine"。
  - 升级后的文件名没有变化，如还有其他文件报错，请自行重新导入。
- 如果您在开发过程中，因为执行 pod update 升级后，出现编译错误，可以使用如下方式解决：
  - 替换 "import TUICallEngine" 为 "import RTCRoomEngine"。
- 对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。

Version 2.9.0.1230 @ 2025.2.18

新特性

- Android&iOS：新增 calls 接口，支持发起单人通话或多人通话，更加灵活的通话成员管理，支持更强大的 REST API，欢迎使用。
- Android&iOS：新增 join 接口，配合 calls 接口、支持加入已有通话。
- Android&iOS：支持来电震动。

缺陷修复

Android： 修复悬浮窗导致的内存泄漏问题。

Version 2.7.0.1130 @ 2024.11.27

新特性

Android&iOS：新增繁体中文，优化使用体验。

缺陷修复

- Android：修复收到FCM推送消息无法弹出通知的问题。
- Android：修复中途加入语音通话，无法拉取已开摄像头的用户的画面问题。
- iOS： 修复iOS主叫在接听前进入悬浮窗，接听后对端画面异常的问题。
- iOS： 修复在等待接听页面关闭摄像头以后，接听后摄像头状态不一致的问题。

Version 2.6.0.1080 @ 2024.9.29

功能优化

- Android：大屏设备支持横屏布局，增强使用体验。
- iOS：适配 iOS 16以上的版本，支持应用外悬浮窗。
- Android：有悬浮窗权限时，支持应用外悬浮窗。

缺陷修复

- Android&iOS：修复极少场景下，发起通话后无回调问题。
- Android&iOS：修复 hangup 接口回调两次问题。
- iOS：修复 VoIP 使用过程中，切换音频播放设备不可用问题。
- iOS：修复后台活跃状态收到来电后，进入 APP 后铃声不响问题。

Version 2.5.0.1025 @ 2024.8.7

功能优化

Android：优化中途加入的逻辑，避免内存泄漏。

缺陷修复

- Android&iOS：修复发送群通话邀请给 web 端用户，web 用户收不到通话邀请的问题。
- Android：修复群组通话中 A 语音呼叫 B，B 收到推送通知后点击拉起界面，显示是扬声器而不是听筒问题。

Version 2.4.0.970 @ 2024.6.11

功能优化

- Android&iOS：弱网时提示用户，增强用户使用体验。
- Android：优化被叫端的锁屏时的来电策略。
- iOS：修复群通话内存异常增长问题。

缺陷修复

Android&iOS：修复主动加入通话界面显示异常问题。

Version 2.3.0.915 @ 2024.4.15

功能优化

- Android&iOS：支持在 TUIChat 的群组顶部显示通话状态，并允许群成员主动加入通话。
- Android&iOS：优化来电弹出逻辑，默认展示横幅接听框。
- Android&iOS：视频通话支持背景虚化，详见：虚拟背景 。

缺陷修复

- Android：修复通话记录编辑界面中，点击删除按钮无响应的问题。
- iOS：修复在群通话中点击成员视图时，切换过程中出现重影的问题 。
- iOS：修复音视频界面特定场景下不显示的问题。
- Android&iOS：修复在呼叫忙线中的用户时，通话结束后缺少提示的问题。

Version 2.2.0.860 @ 2024.2.1

功能优化

Android&iOS：全新的 UI 视效，功能更清晰，体验更好。

缺陷修复

Android&iOS：修复在会议、直播场景下，接听来电后，麦克风和摄像头设备互相占用问题。

Version 2.1.0.810 @ 2023.12.19

功能优化

- Android：优化未登录时调用 TUICallKit 接口异常的提示。
- Android： 优化针对 Android 14平台(API 34) 的兼容性，详见：Android 14 行为变更 。
- Android&iOS：优化用户昵称的显示，按照如下顺序显示：用户备注 > 用户昵称 > 用户 ID，默认为 userId。

缺陷修复

- iOS：修复群通话头像显示重叠问题。
- iOS：修复视频通话时开启悬浮窗发消息再回到通话界面键盘无法收起问题。
- iOS：修复视频通话中，先切换摄像头、关闭摄像头，再切换回原摄像头并重新打开时，无法切换和移动摄像头的问题。
- Android：修复在阿拉伯语等从右至左布局模式(RTL 模式)下单人视频通话的小窗口无法移动的问题。

Version 2.0.0.750 @ 2023.11.03

功能优化

- Android&iOS：UIKit 支持日语。
- Android&iOS：优化通话昵称显示。
- Android&iOS：调整铃声的默认音量，由60%调整到100%。

缺陷修复

- iOS：修复Swift 版本加载图片过慢问题。
- iOS：修复主叫端发起通话后退后台，通话邀请自动取消问题。

Version 1.9.0.680 @ 2023.09.27

功能优化

- Android&iOS：支持阿拉伯语言。
- Android&iOS：优化套餐购买提示，支持根据链接跳转到对应的套餐购买页。
- Android&iOS：优化不同分辨率下的默认码率，高分辨率下画面更清晰，详见 Resolution 。
- Android&iOS：调整视频通话的默认码率为600kbps，美颜等级为4级。

缺陷修复

- Android&iOS：对黑名单用户发起通话，通话拒绝提示与发送单聊消息拒绝提示不一致。
- iOS：4人群视频通话界面，一名成员拒绝接听，发起人视频位置画面异常。
- iOS：修复登录成功后立即设置美颜，分辨率会被重置问题。

Version 1.8.0.620 @ 2023.08.14

功能优化

- Android&iOS：通话消息默认不计入未读数。
- Android：优化小米手机悬浮窗权限跳转页面。

缺陷修复

- iOS：修复被移下线后回调接口 onKickOffline 无效问题。
- iOS： 修复通话记录未接界面清除通话后，回到所有通话界面，列表为空问题。

Version 1.7.2.570 @ 2023.07.20

功能优化

Android：默认关闭重力感应，优化在大屏、定制设备的通话体验。

缺陷修复

- Android&iOS：A 呼叫 B（离线）后取消，A 再次呼叫 B，B 登录上线，B 的云端通话记录异常问题。
- Android： 升级 TRTC SDK 版本到11.3后，TUICallKit 的崩溃问题。

Version 1.7.0.460 @ 2023.06.25

功能优化

- Android&iOS：含 UI 集成方案，优化示例工程，完善通话设置项。
- Android：降低通话时系统保活的级别，只在状态栏显示保活提示，去掉通知和震动。

Version 1.6.1.410 @ 2023.05.22

新特性

- Android&iOS：含 UI 接口 call 、groupCall 支持自定义房间号。

- Android&iOS：支持发起通话时传入字符串型房间号，详见 CallParams 。

缺陷修复

- Android：修复通过 Arrays.asList 方式生成列表参数时，groupCall 报错问题。
- Android：修复视频通话显示异常问题。
- iOS：修复与 TUIRoom 组件冲突问题。
- iOS：修复登录成功回调中，立刻发起通话崩溃问题。
- iOS：修复点击通知消息进入 APP，偶现不显示邀请页面问题。

Version 1.6.0.360 @ 2023.04.27

新特性

- Android：TUICallKit 新增 Kotlin 语言版本。
- iOS：TUICallKit 新增 Swift 语言版本。
- Android&iOS：新增通话记录（本地）的显示页面。

功能优化

- Android：优化单人视频通话头像显示。
- Android&iOS：群组通话中，默认支持邀请其他群成员加入通话。

缺陷修复

- Android：修复 Android 12及以上设备连接蓝牙后无声的问题。
- Android：修复被叫端偶现静音设置不生效问题。
- iOS：修复重新登录后偶现设备接收不到来电邀请的问题。
- iOS：修复 TUICallKit 中 enableCustomViewRoute 接口失效问题。
- iOS：修复 VoIP 推送页面昵称显示错误的问题。

Version 1.5.1.310 @ 2023.04.17

新特性

- Android&iOS：新增 VoIP 消息推送功能，提供更好的通话接听体验。
- Android&iOS：支持发起通话时自定义扩展字段，详见 call 方法的 TUICallDefine.CallParams 参数。

功能优化

- Android&iOS：优化华为、小米、FCM 等厂商的离线推送能力，增加厂商消息分类及 channelID 设置。

缺陷修复

- Android&iOS：修复 IM 自定义属性发起通话后被覆盖的问题。
- Android：修复回调 onCallEnd 中 totalTime 的时间单位错误的问题；

Version 1.5.0.305 @ 2023.03.09

功能优化

- Android&iOS：优化上屏消息显示。
- Android：贴耳息屏功能默认关闭。
- Android：升级 Gradle 插件及版本。
- Android：优化铃声播放类，支持循环播放。

缺陷修复

- Android&iOS：修复被叫接听通话失败，没有 onCallCancel 回调问题。
- Android：修复被叫接听通话失败，主叫异常问题。

版权所有：腾讯云计算（北京）有限责任公司

- Android：修复首次通话检查权限时，主叫端取消通话，被叫端再次拉起界面问题。
- Android：修复回调上层的网络质量 userId 为空问题。

Version 1.4.0.255 @ 2023.01.06

新特性

Android&iOS：支持自定义来电超时时间，详见 call方法 TUICallDefine.CallParams 参数。

缺陷修复

Android&iOS：修复主动加入房间（joinInGroupCall）时通话异常结束问题。
Android：修复在音视频通话接听中退后台，再回到前台时，通话状态异常问题。

Version 1.3.0.205 @ 2022.11.30

新特性：

Android&iOS：新增美颜设置接口 setBeautyLevel，支持关闭默认美颜。

功能优化：

iOS: 优化 TUICallKit 的 Framework 体积。

缺陷修复：

- Android&iOS：修复服务端解散房间或踢出用户时，通话界面不消失问题。
- Android：修复A呼叫离线用户B，然后取消；A 再次呼叫 B，B 上线后通话界面不显示问题。

Version 1.2.0.153 @ 2022.11.14

新特性：

- Android&iOS：支持视频自定义编码分辨率。
- Android&iOS：支持设置视频的渲染参数：渲染方向、填充模式。
- Android&iOS：支持第三方美颜的接入。
- Android&iOS：TUICallKit 新增重载接口 call()、groupCall()，支持自定义离线消息（详情请参见 API文档）。

功能优化：

- Android&iOS：优化部分 TUICallObserver 回调异常问题。
- Android&iOS：优化视频切换语音功能，支持离线状态下切换。
- Android&iOS：完善 TUICallKit 的错误码及错误提示。
- iOS：规范 TUICallEngine、TUICallKit Swift API 名称。

缺陷修复：

- Android&iOS：修复重新登录账号后，仍会收到之前已拒绝的来电问题。
- Android：修复单聊中收到群聊邀请，通话异常挂断问题。
- Android：多场景退出直播间无法发起通话异常问题。
- Android：修复 A 呼叫 B，同时 B 呼叫 C，C 概率性进入 A 的房间问题。

Version 1.1.0.103 @ 2022.09.30

- Android&iOS：优化群组通话过程中，邀请群成员加入功能。
- Android&iOS：优化通话流程，避免通话接听前产生录制、审核等费用。
- Android&iOS：支持自定义音视频通话的离线推送消息。
- Android&iOS：部分 **TUICallEngine** 接口参数变更，详见 call()、groupCall()、inviteUser()、以及回调接口onCallReceived()。
- Android&iOS：修复群组通话过程中，偶现的少量回调异常问题。
- Android&iOS：修复重复登录以及UserSig过期造成的状态异常问题。

版权所有：腾讯云计算（北京）有限责任公司

- iOS：修复 TUICallKit Objective-C 和 Swift 混合编译时，`init` 接口报错的问题。
- Android：修复 Kotlin 集成悬浮窗功能时编译报错问题。

Version 1.0.0.53 @ 2022.08.15

全新发布：

- Android&iOS：支持1v1 音视频通话、群组音视频通话。
- Android&iOS：支持主流厂商的离线唤醒。
- Android&iOS：支持自定义头像、自定义昵称。
- Android&iOS：支持通话过程中开启悬浮窗。
- Android&iOS：支持设置自定义铃音。
- Android&iOS：支持多平台登录状态下的来电服务。

版权所有：腾讯云计算（北京）有限责任公司

uni-app (客户端)

最近更新时间：2025-02-27 16:11:32

uni-app 的 CallKit 插件完整更新日志，请访问 uni-app 插件市场的 changelog 。

2.9.0 版本升级须知：
对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。

Flutter

最近更新时间：2025-02-27 16:11:32

Flutter 的完整更新日志，请访问 pub.dev changelog 。

> **2.9.0版本升级须知：**
> 对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。

版权所有：腾讯云计算（北京）有限责任公司

微信小程序

最近更新时间：2025-06-30 15:36:52

> **注意：**
> 1. **新版本（4.0.2）涉及 包名调整为 @tencentcloud/call-uikit-wx：**
>   - **对于需要升级到 calls 接口的用户，请您注意版本灰度，使用 calls 接口呼叫，如果对方的版本未升级，会导致通话失败。**

Version 4.0.7 @2025.06.06

新增

- 支持 REST API 结束通话。
- 邀请他人时，将通话中的用户通过 inviteeList 通知给被叫。
- 增加对通过 REST API 销毁房间的处理。
- 增加 calls 网络中断恢复后的兼容性处理。

优化

- 优化和 chat tuikit 融合时的中途加入 join 功能。
- 优化端上长时间连续心跳失败（断网/超时）结束通话逻辑。
- 优化异常中断（关闭浏览器/刷新浏览器/小程序左滑/右上角退出）等操作，立刻结束通话。
- 优化被叫 reject 接口逻辑，移除退房操作。
- 优化排障日志，增加在线日志。
- 优化 callkit 事件监听，将监听 USER_ACCEPT 调整为 ON_CALL_BEGIN。
- 中途加入，默认关闭摄像头。
- 多人通话下，主叫默认大屏。

修复

- 修复中途加入未更新 groupId 问题。
- 修复通话记录上报 groupId 缺失问题。
- 修复记录在中途加入和邀请他人场景下，参与通话成员不准确问题。
- 修复主动加入通话时，异常回调 onUserLeave 问题。

Version 4.0.4 @2025.03.31

新增

- 新增 onCallNotConnected 事件。
- 新增 onUserInviting 事件。
- 新增 chatGroupID 字段在 ON_CALL_RECEIVED、CALLING_END、ON_CALL_BEGIN 事件。
- 增加在线排障日志。

优化

- 优化多端登录问题。
- 优化通过 trtc REST API 解散房间的逻辑。
- 优化本地日志打印。

修复

- 修复日志上报问题（字段缺失导致 Kibana 没有记录）。
- 修复 setSelfInfo 接口无效问题。
- 修复 calls 能力位校验码异常问题。

- 修复被叫都忙线时，主叫呼叫被叫未抛出 onCallCanceled 事件问题。
- 移除 TRTC 退房抛出的 userLeave 事件。

Version 4.0.3 @2025.03.13

修复

当所有的被叫都忙线时，engine 没有抛出 cancel 事件时，callkit 根据远端用户列表结束通话。

Version 4.0.2 @2025.02.18

新增

- 新增 calls 接口，支持发起单人通话或多人通话，更加灵活的通话成员管理，支持更强大的 REST API，欢迎使用。
- 新增 join 接口，配合 calls 接口、支持加入已有通话。

Version 2.2.2 @2024.08.09

新增

新增弱网时 toast 提示。

优化

- 优化群组通话支持进房前预览。
- 优化铃声实例的创建、销毁时机。
- 优化关闭摄像头后，翻转摄像头和虚拟背景按钮的交互。
- 优化用户未 init 时，直接调用 call/groupCall API 时的报错内容。
- 优化用户未 init 时，在页面引入 TUICallKit 组件，控制台的 setVideoResolution 警告现象。

修复

- 修复铃声的异常响铃问题。
- 修复微信 App 无权限时的进房失败问题。

Version 2.2.1 @2024.07.12

新增

- 和 chat 融合，callkit 控制自身显隐。
- 增加点击 “接听” 的日志上报。

优化

样式作用域优化，避免影响用户样式。

修复

- 修复中途加入的已知问题。
- 修复 roomId、roomIdType 更新问题。

Version 2.1.9 @2024.06.14

新增

新增没有 live-pusher 推流能力时的弹窗提示。

优化

- 扩展 offlinePushInfo 参数，支持离线推送声音设置等功能。
- 优化全局监听下初始化失败后的异常抛出。

修复

修复被叫接通后，立刻自己超时后通话异常问题。

Version 2.1.8 @2024.05.17

修复

修复开启画中画，对端开关摄像头会导致退房的问题。

Version 2.1.5 @2024.05.10

新增

视频通话支持背景虚化，详细参见： 虚拟背景 。

Version 2.1.4 @2024.05.06

修复

修复事件监听移除问题。

Version 2.1.3 @2024.04.19

新增

- 新增悬浮窗、拖动功能。
- 新增对小程序插件接入的支持，见TRTC 音视频通话插件 。

优化

优化异常退出时（左滑、后退等）的逻辑处理。

Version 2.1.2 @2024.03.08

新增

增加 TRTC 事件对应的日志上报 Kibana。

修复

音频通话推流异常问题。

Version 2.1.1 @2023.12.29

修复

- 修复小程序 setSelfInfo 接口传参问题。
- 修复 destroyed 存在的问题。

Version 2.1.0 @2023.12.01

新增

融合到 Chat 中增加 isFromChat 上报。

修复

修复按钮 loading 下可点击的问题。

Version 2.0.0 @2023-11-17

Breaking Change

升级底层依赖 tuicall-engine-wx 至 ^2.0.0，不再支持 tim-wx-sdk 创建 tim 实例，如需创建 tim 实例，请使用 @tencentcloud/chat 创建。

新增

- 增加主叫、被叫的铃声。
- 增加被叫来电铃声静音接口 enableMuteMode 。
- 增加呼叫黑名单用户失败提示信息。
- 增加对日语的支持。
- 增加 TUICallKit 框架上报。
- 新增无套餐的弹框提示。

优化

- 视频通话下，大屏默认显示远端用户。
- 优化远端流用户信息显示优先级：备注 > 昵称 > userId。
- 优化 TUICallKit 组件包体积（删除无用图片及代码）。

Version 1.4.4 @2023-07-28

新增

新增 init 错误抛出处理。

Version 1.4.3 @2023-07-14

修复

调整事件监听时机，修复 SDK_READY 未抛出的问题。

Version 1.4.2 @2023-06-09

修复

- 修复全局监听卸载不彻底的问题。
- 修复全局监听本地预览取值不当的问题。

Version 1.4.1 @2023-05-25

新增

新增 call 、groupCall 错误抛出处理。

Version 1.4.0 @2023-05-17

新增

新增在 tabBar 页面接入时的兼容性处理。

优化

- 优化在页面跳转情况下的初始化以及卸载逻辑。
- 优化全局监听逻辑，防止重复监听。

Version 1.3.9 @2023-05-11

新增

- 新增 call 、groupCall 支持自定义数字房间号。
- 新增 call 、groupCall 允许用户自定义 userData 数据。

修复

修复全局监听模式下，视频通话被叫预览黑屏导致无法接听的问题。

Version 1.3.8 @2023-04-27

> ⚠ 注意：

> 需要依赖 tuicall-engine-wx 的版本 ≥ v1.4.7。

新增

新增通话页面出现时的动画效果。

优化

- 优化通话界面唤起的时间。
- 优化用户异常退出的逻辑，如右滑退出以及返回退出的情况。

修复

修复用户重复点击，导致接口被重复调用的问题。以下接口增加节流处理： call，groupCall，accept，reject。

Version 1.3.7 @2023-04-14

新增

新增 设置日志级别接口 。

修复

修复铃声路径错误问题。

Version 1.3.6 @2023-03-31

新增

新增 index.js 入口文件，解决 npm 构建时的警告问题。

优化

- 优化用户没有 live-pusher 推流标签使用权限时，给予弹窗提示。
- 优化针对在微信开发者工具运行时，给予弹窗提示。

Version 1.3.5 @2023-03-24

新特性

支持自定义铃声。

修复

优化 reset 函数在生命周期中的触发时机，修复全局监听下被叫方的白屏问题。

Version 1.3.4 @2023-03-16

修复

全局监听下的视频通话，被叫方使用 iOS 设备预览黑屏问题。

优化

优化常见错误提示，方便客户快速定位问题。

Version 1.3.3 @2023-03-02

优化

- 弹窗昵称显示优先级（昵称 > userID）。
- 打印显示 TUICallKit 版本号。

Version 1.3.2 @2023-02-17

修复

打包路径错误。

Version 1.3.1 @2023-02-17

修复

修复远端关闭麦克风，没有及时更新远端流导致后续调用音量变化接口报错。

Version 1.3.0 @2023-02-13

优化

优化依赖引入，组件包体积由900KB降低至110KB左右。

Version 1.2.7 @2023-02-03

修复

去除 aegis.js 文件。

Version 1.2.6 @2023-01-03

修复

未及时回收 wx.$TUICallEngine 变量引发的 init 报错。

Version 1.2.5 @2023-01-31

新特性

支持 uniapp 小程序全局监听。

修复

升级 tsignaling-wx，解决和客户端的互通问题。

Version 1.2.4 @2023-01-12

修复

- 打包路径错误。
- callManager 兼容 sdkAppID 与 SDKAppID 传参。
- callkit 生命周期初始化时 移除 wx.CallManager 监听报错问题。

Version 1.2.3 @2023-01-10

修复

TUICallKit 样式布局错误。

Version 1.2.1 @2023-01-09

修复

打包问题引起的 globalCall 页面引用错误。

Version 1.2.0 @2023-01-06

新特性

支持全局监听

- 主包方案详见【TUICallKit】小程序全局监听（主包方案）。
- 分包方案详见【TUICallKit】小程序全局监听（分包方案）。

Version 1.1.4 @2022-12-23

修复

- init 方法支持外部传入 tim。
- 修复 uni-app 打包小程序存在的栈溢出问题。

Version 1.1.3 @2022-12-14

新增

新增设备权限弹出窗口。

优化

群通话中的昵称显示优先级。

> **说明：**
> 群通话显示优先级为**昵称 > userID**。

修复

群通话中 player 推流过程中 trtc-wx 报错问题。

Version 1.1.2 @2022-11-30

优化

优化接入流程。

Version 1.1.1 @2022-11-30

优化

兼容 chat，支持 TUICallKit 在 init 时传值。

Version 1.1.0 @2022.11.16

> **注意：**
> 升级至该版本需注意：升级指引 。

优化

- 提高多人通话稳定性
- 抛出事件的数据结构调整，具体请参见 TUICallEvent 。

常见问题（TUICallKit）
全平台

最近更新时间：2025-06-26 10:12:42

> **说明：**
> 如果并没有解决您的问题，也欢迎您加入我们的 TUICallKit 技术交流 QQ 群： **605115878** 来进行交流讨论。

音视频通话功能出现了未开通的错误提示?

您可以选择体验版（7 天试用）或者购买正式版。

对于体验版

进入 即时通信 IM 控制台 的配置页面，并在页面的右下角找到开通腾讯实时音视频服务功能区，单击**免费体验**，选择**领取7天试用**，即可开通 TUICallKit 的 7 天免费试用服务，成为体验版。



对于正式版

如果您体验过 7 天体验期后，需要购买正式套餐。IM 音视频通话能力针对不同的业务需求提供了差异化的付费版本供您选择，您可以在 IM 购买页 了解包含功能并选购您适合的版本。

在 TUIChat + TUICallKit 融合场景下，升级 TUICallKit 版本后出现新版本呼叫旧版本，旧版本无法收到通话问题?

TUICallKit 3.1 版本起，在 TUIChat + TUICallKit 融合场景下，通过 TUIChat 中的通话按钮发起通话默认使用 calls 新接口，所以如果您的业务已经上线，您可以通过高级接口将通话接口切换为 call、groupCall，用来保持与线上版本的用户互通。
您需要在登录成功后调用一次高级接口，各个平台的具体使用为：

iOS(Swift)

```
let jsonParams: [String: Any] = ["api": "forceUseV2API",
                                 "params": ["enable": true,],]
guard let data = try? JSONSerialization.data(withJSONObject: jsonParams, options:
JSONSerialization.WritingOptions(rawValue: 0)) else { return }
guard let paramsString = NSString(data: data, encoding: String.Encoding.utf8.rawValue) as? String else
{ return }

TUICallKit.createInstance().callExperimentalAPI(jsonStr: paramsString)
```

iOS(Objective-C)

```
NSDictionary *jsonParams = @{@"api": @"forceUseV2API",
                             @"params": @{@"enable": @YES}};
NSError *error = nil;
NSData *data = [NSJSONSerialization dataWithJSONObject:jsonParams options:0 error:&error];
if (error || !data) { return; }
```

```
NSString *paramsString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
if (!paramsString) { return;}
[[TUICallKit createInstance] callExperimentalAPIWithJsonStr:paramsString];
```

Android(java)

```
 try {
    JSONObject params = new JSONObject();
    params.put("enable", true);
    JSONObject jsonObject = new JSONObject();
    jsonObject.put("api", "forceUseV2API");
    jsonObject.put("params", params);
    TUICallKit.createInstance(context).callExperimentalAPI(jsonObject.toString());
} catch (Exception e) {
    e.printStackTrace();
}
```

Android(kotlin)

```
try {
    val params = JSONObject()
    params.put("enable", true)

    val jsonObject = JSONObject()
    jsonObject.put("api", "forceUseV2API")
    jsonObject.put("params", params)
    TUICallKit.createInstance(context).callExperimentalAPI(jsonObject.toString())
} catch (e: Exception) {
    e.printStackTrace()
}
```

Flutter(Dart)

```
final jsonParams = {'api': 'forceUseV2API',
                    'params': {'enable': true} };

try {
    final jsonString = json.encode(jsonParams);
    TUICallKit.instance.callExperimentalAPI(jsonStr: jsonString);
} catch (e) {
    return;
}
```

Web&小程序

```
const jsonObject = {
    api: "forceUseV2API",
    params: {
      enable: true,
    }
};
```

```
TUICallKitServer.getTUICallEngineInstance().callExperimentalAPI(JSON.stringfy(jsonObject));
```

TUICalling 如何升级到 TUICallKit?

TUICallKit 是腾讯云推出一款新的音视频通话 UI 组件，**是 TUICalling 的升级版本**，支持群组通话、AI 降噪等更多功能特性、支持全平台间互通呼叫，功能更加稳定。从 TUICalling 升级到 TUICallKit 您可以参考：

- 微信小程序升级方案
- Web 升级方案
- Android 升级方案
- iOS 升级方案
- uni-app (客户端) 升级方案
- Flutter 升级方案
- uni-app (小程序) 升级方案

报错 inviteID 无效或邀请已处理

该 inviteID 已经被主叫方取消，或者该邀请已经被接受。由于信令发送受网络带宽等诸多因素影响，在极端情况下(accept 与 reject 接口重复调用)可能会导致 inviteID 无效的错误。

消息发送方或接收方 UserID 无效或不存在

您呼叫的用户还没有登录过 TIM，导致呼叫失败，请尝试呼叫已经登录成功过的用户。

TUICallKit 是否可以不引入 IM SDK，只使用 TRTC?

不可以，TUIKit 全系组件都使用了腾讯云 IM SDK 作为通信的基础服务，比如通话拨打信令、通话忙线信令等核心逻辑，如果您已经购买有其他 IM 产品，也可以参照 TUICallKit 逻辑进行适配。

如何购买音视频通话套餐?

请参考购买链接 音视频通话 SDK 价格总览，如有其他问题，请点击页面右侧，进行售前套餐咨询；或者加入 QQ 群：**605115878**，进行咨询和反馈。

如何获取通话的房间号 Roomid?

通话接通后，您可以通过 onCallBegin 返回 roomid 字段进行获取。

查询用户在线状态

TUICallKit 底层通过 IM 管理用户，可以通过 IM 查询用户的在线状态，方式有以下两种：

- **方式一：**通过 getUserStatus 高级接口查询

  详情可参考：查询用户状态（Android&iOS&Windows&Mac 、web&小程序&uniapp ），您可以调用不同平台的接口去查询。

> ⚠ **注意：**
>
> - 查询其他用户状态需要升级 TUICallKit 套餐包到**群组通话版本**。
> - 查询其他用户状态需要提前在 即时通信 IM 控制台 开启 “用户状态查询及状态变更通知”。不开启，调用 `getUserStatus` 会报错。



getUserStatus 接口的返回结果有以下几种：

- 未知（UNKNOWN）
- 在线（ONLINE）：当前用户已登录上线，通常情况下，该用户可以正常收到通话请求。
- 离线（OFFLINE）：用户未主动调用 `logout` 退出登录，但长连接中断的状态。通常情况下，若接入了离线推送，该用户可以收到离线推送的消息。
- 未登录（UNLOGINED）：用户注册账号后从未登录过，或者用户主动调用 `logout` 退出登录，该用户收不到通话请求。

- **方式二：**通过 REST API 进行查询

  详情可参考：查询账号在线状态 。

音视频通话信令里各字段常见问题?

问题	说明
如何判断 “发起通话”	actionType = 1 且 cmd = 'videoCall' 或 'audioCall'
如何判断 “已接听”	actionType = 3
如何判断 “通话已结束”，“通话时长是多少”	actionType = 1 且 cmd = 'hangup'，通话时长取 call_end 字段，单位：秒
如何判断 “取消通话”	actionType = 2
如何判断 “拒绝通话”	actionType = 4
如何判断对端忙线产生的 “拒绝通话”	actionType = 4 且 cmd = "line_busy"
如何被叫判断 “超时未接听”	actionType = 5

Web

最近更新时间：2025-05-14 14:39:02

> **说明：**
> - 如果并没有解决您的问题，也欢迎您加入我们的 TUICallKit 技术交流 QQ 群： **605115878** 来进行交流讨论。
> - 我们团队即将在 Web 端丰富更多不同的组件，以满足您的开发需求。
>
> 希望您能抽出几分钟时间，为您自己的需求投上一票，我们将优先开发！
> 问卷地址： TUICallKit Web 问卷调查

一、基础环境问题

> **说明：**
> 以下问题对于 TUICallKit 与 TUICallEngine SDK 同样适用。

TUICallEngine 和 TUICallKit 分别是什么?

TUICallKit 是含 UI 音视频通话组件，底层是用 TUICallEngine SDK，目前支持 Typescript+Vue2 / Typescript+Vue3，可直接将组件放到页面中，调用简单的接口即可直接实现音视频通话，开源地址为：TUICallKit/Web。
TUICallEngine SDK 是音视频通话组件的无 UI SDK，如果 TUICallKit 的交互并不满足您的需求，您可以使用这套接口自己封装交互。npm 地址为 @tencentcloud/call-engine-js 。

Web 端支持哪些浏览器? 报错“获取设备权限失败”？

请先确保页面已被授权使用麦克风或摄像头，参见 设备授权说明。
对浏览器的详细支持度，请参见 浏览器兼容信息。
对于上述没有列出的环境，您可以在需要检测的浏览器打开 能力测试 测试是否完整的支持 WebRTC 的功能。

为什么本地开发测试能正常使用，但是部署到线上用 IP 访问后无法正常视频/语音通话?

- **对网站域名协议的要求**

  出于对用户安全、隐私等问题的考虑，浏览器限制网页在 HTTPS 协议下才能正常使用本文档中所对接组件的全部功能。为确保生产环境中的用户能够顺畅体验产品功能，请将您的网站部署在 `https://` 协议的域名下。更多请参见 页面访问协议说明。
- **对网络环境的要求**

  在使用 TUICallKit 时，用户可能因防火墙限制导致无法正常进行音视频通话，请参考 应对防火墙策略 将相应端口及域名添加至防火墙白名单中。

在接通过程中报："is not included in the current tim's package"?

TUICallKit（含 TUICallEngine SDK）依赖的 `tim-js-sdk` 版本需要 ≥ 2.21.2。
如果 `tim-js-sdk` 依赖包版本正确，则可能是 SDKAppID 未购买音视频套餐或套餐包不支持所调用的功能，请访问 音视频通话功能出现了未开通的错误提示。

TUICallKit（含 TUICallEngine）是否支持接收离线消息?

不支持接收离线消息。
支持离线消息推送，可以通过 call / groupCall 中的 offlinePushInfo 添加需要推送的消息。

TUICallEngine init 未完成，需要在 init 完成后使用此API

未调用 login 接口，所有功能需要先进行登录完成后才能使用，具体参见 TUICallEngine login 。

> **注意：**
> TUICallEngine login 是一个异步接口，避免在 login 还未完成，直接调用 TUICallEngine 的接口。

当前通话状态无法使用该 API

API 与通话状态对照表：

API	idle	calling	connected
call	✓	×	×
groupCall	✓	×	×
accept	×	✓	×
reject	×	✓	×
hangup	×	✓	✓
inviteUser	×	✓	✓
joinInGroupCall	✓	×	×

二、TUICallKit 问题（含 UI）

1. TUICallKit Web 支持什么框架？支持 H5 吗？

TUICallKit 适用于 `Vue2.7 + Typescript` 或者 `Vue3 + Typescript` 项目，若您采用其他语言或者技术栈，请访问 界面定制指引。
TUICallKit 支持 H5，通过页面 UA 自动修改适应移动端的布局，为了移动端的良好体验，推荐通过修改 CSS 在 H5 时将 `<TUICallKit/>` 组件放大至全屏。

2. TUICallKit 打包失败？

对于 Vite 项目，您需要在 `vite.config.js` 中添加 `base: "./"` 。
对于 Vue-CLI 创建的 webpack 项目，您需要在 `vue.config.js` 中添加 `publicPath: "./"` 。

3. TUICallKit 报错“获取设备权限失败”？

请先确保页面已被授权使用麦克风或摄像头，参见 设备授权说明。
可以尝试 Github Demo 是否可以正常通话，然后在 设备检测页面 检查是否支持 webrtc。

4. timeout 字段设置无效原因？

该字段目前在 call / groupcall API 中会被使用到。
目前全平台 TUICallKit 的策略是被叫登录后，仅拉取 30s 内的历史消息。所以被叫登录后无法拉到主叫 30s 前的呼叫信息，从而导致被叫无法拉起邀请页面进行通话。

5. 视频通话切换大小屏出现闪屏现象？

ios 17 设备和 WebRTC 底层兼容性问题，等待后续修复。

6. safari 浏览器省电模式出现暂停键？

浏览器层面针对省电模式的视频播放策略，SDK 无法处理。

7.如何生成 UserSig？

UserSig 是腾讯云为其云服务设计的一种安全保护签名，是一种登录凭证，由 SDKAppID 与 SecretKey 等信息组合加密得到。

- **方式一：**控制台获取，参考 获取临时 userSig。
- **方式二：**部署临时生成脚本。

> **警告：**
> 此方式是在前端代码中配置 SecretKey，该方法中 SecretKey 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通功能调试**，生产环境请看方式三。

为方便初期调试，userSig 可临时使用 `GenerateTestUserSig-es.js` 中 `genTestUserSig(params)` 函数来计算，例如：

```
import { genTestUserSig } from "@tencentcloud/call-uikit-vue/debug/GenerateTestUserSig-es.js";
const { userSig } = genTestUserSig({ userID: "Alice", SDKAppID: 0, SecretKey: "YOUT_SECRETKEY" });
```

- **方式三：正式环境使用。**

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

8.如何创建 userID?

- 通过 userID 与 UserSig 登录过一次，会默认创建该用户。
- 通过 即时通信 IM 控制台 进行创建和获取，单击目标应用卡片，进入应用的账号管理页面，也可创建账号并获取 userID。



9. 解决源码拷贝可能导致的报错

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

ESLint 报错

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

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

TypeScript 报错

如遇 `Cannot find module '../package.json'` 报错，是因为 TUICallKit 内引用了 JSON 文件，可在 `tsconfig.json` 中添加相关配置。其他 TSConfig 问题请参见 TSConfig Reference。

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

10. 如何源码集成 TUICallKit?

步骤一：下载 TUICallKit 组件源码及相关插件

Vue3

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

Vue2.7

```
npm install @tencentcloud/call-uikit-vue2
```

Vue2.6

```
npm install @tencentcloud/call-uikit-vue2.6 @vue/composition-api
```

源码集成需要安装 unplugin-vue2-script-setup 插件。

```
npm i -D unplugin-vue2-script-setup
```

修改构建工具配置文件。

Vue Cli

```
// vue.config.js
const ScriptSetup = require('unplugin-vue2-script-setup/webpack').default

module.exports = {
  parallel: false, // disable thread-loader, which is not compactible with this plugin
  configureWebpack: {
    plugins: [
      ScriptSetup({ /* options */ }),
    ],
  },
}
```

vite

```
// vite.config.ts
import { defineConfig } from 'vite'
import { createVuePlugin as Vue2 } from 'vite-plugin-vue2'
import ScriptSetup from 'unplugin-vue2-script-setup/vite'

export default defineConfig({
  plugins: [
    Vue2(),
    ScriptSetup({ /* options */ }),
  ],
})
```

Webpack

```
// webpack.config.js
const ScriptSetup = require('unplugin-vue2-script-setup/webpack').default

module.exports = {
  /* ... */
```

```
  plugins: [
    ScriptSetup({ /* options */ }),
  ]
}
```

步骤二：源码集成 TUICallKit 组件包

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

macOS + Vue3

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

macOS + Vue2

```
mkdir -p ./src/components/TUICallKit && cp -r ./node_modules/@tencentcloud/call-uikit-vue2/*
./src/components/TUICallKit
```

Windows + Vue3

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

Windows + Vue2

```
xcopy .\node_modules\@tencentcloud\call-uikit-vue2  .\src\components\TUICallKit /i /e
```

步骤三：修改 TUICallKit 引入路径

修改 TUICallKit 引入路径为本地文件引入。

> ⚠ **注意：**
> 此方法可能与您的 ESLint、TypeScript 配置冲突，若出现报错，可参见 解决源码拷贝导致的报错 。

```
import { TUICallKit, TUICallKitServer } from "./components/TUICallKit/src/index";
```

接下来请参见上面的步骤及说明，使用 TUICallKit 组件提供的功能。

三、套餐问题

1. 错误提示“The package you purchased does not support this ability”？

由于您当前应用的音视频通话能力包过期或未开通，请参见 开通服务，领取或者开通音视频通话能力，进而继续使用 TUICallKit 组件。

2. 如何购买套餐?

请参考购买链接 音视频通话 SDK 价格总览，如有其他问题，请点击页面右侧，进行售前套餐咨询，或 进入 IM 社群 进行咨询和反馈。

四、内网代理

内网的环境下如何使用 callkit?

1. 通过 getTRTCCloudInstance 获取 TRTCCloud 实例。

> ⚠ 注意：
> v3.1.3+ 支持。

```
const trtcCloud = TUICallKitServer.getTUICallEngineInstance().getTRTCCloudInstance();
```

2. 通过 TRTCCloud 实例调用 callExperimentalAPI 完成代理服务器的设置（注意：需要在通话前调用）。具体如下：

```
trtcCloud.callExperimentalAPI(JSON.stringify({
   api: 'setNetworkProxy',
   params: {
      websocketProxy: 'wss://proxy.example.com/ws/',
      turnServer: [{
         url: '14.3.3.3:3478',
         username: 'turn',
         credential: 'turn',
      }],
      iceTransportPolicy: 'relay',
   },
}));
```

> ⚠ 注意：
> TRTC 应对防火墙受限参见 设置代理服务器 。

五、其它

1. 如何关闭美颜?

- web 端默认无美颜。
- 小程序端调用 setBeautyLevel API，传入参数为0时，会关闭默认美颜。

```
TUICallKitServer.getTUICallEngineInstance().setBeautyLevel(0);
```

Android

最近更新时间：2025-05-27 16:09:52

TUICallKit 是否可以不引入 IM SDK，只使用 TRTC?

不可以，TUIKit 全系组件都使用了腾讯云 IM SDK 作为通信的基础服务，例如通话拨打信令、通话忙线信令等核心逻辑，如果您已经购买有其他 IM 产品，也可以参照 TUICallKit 逻辑进行适配。

错误提示“The package you purchased does not support this ability”？

如遇以上错误提示，是由于您当前应用的音视频通话能力包过期或未开通，请参见 开通服务，领取或者开通音视频通话能力，进而继续使用 TUICallKit 组件。

如何购买音视频通话套餐?

请参考购买链接 音视频通话 SDK 价格总览，如有其他问题，请点击页面右侧，进行售前套餐咨询；或者加入 QQ 群：**605115878**，进行咨询和反馈。

TUICallKit 崩溃，崩溃日志："No implementation found for xxxx"?

堆栈信息如下：

```
java.lang.UnsatisfiedLinkError: No implementation found for void
com.tencent.liteav.base.Log.nativeWriteLogToNative(int, java.lang.String, java.lang.String) (tried
Java_com_tencent_liteav_base_Log_nativeWriteLogToNative and
Java_com_tencent_liteav_base_Log_nativeWriteLogToNative__ILjava_lang_String_2Ljava_lang_String_2)
        at com.tencent.liteav.base.Log.nativeWriteLogToNative(Native Method)
        at com.tencent.liteav.base.Log.i(SourceFile:177)
        at com.tencent.liteav.basic.log.TXCLog.i(SourceFile:36)
        at com.tencent.liteav.trtccalling.model.impl.base.TRTCLogger.i(TRTCLogger.java:15)
        at com.tencent.liteav.trtccalling.model.impl.ServiceInitializer.init(ServiceInitializer.java:36)
        at
com.tencent.liteav.trtccalling.model.impl.ServiceInitializer.onCreate(ServiceInitializer.java:101)
        at android.content.ContentProvider.attachInfo(ContentProvider.java:2097)
        at android.content.ContentProvider.attachInfo(ContentProvider.java:2070)
        at android.app.ActivityThread.installProvider(ActivityThread.java:8168)
        at android.app.ActivityThread.installContentProviders(ActivityThread.java:7709)
        at android.app.ActivityThread.handleBindApplication(ActivityThread.java:7573)
        at android.app.ActivityThread.access$2600(ActivityThread.java:260)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2435)
        at android.os.Handler.dispatchMessage(Handler.java:110)
        at android.os.Looper.loop(Looper.java:219)
        at android.app.ActivityThread.main(ActivityThread.java:8668)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:513)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1109)
```

> ⚠ **注意：**
> 上述异常，是因为 TUICallKit 依赖的 TRTC 等 SDK 部分接口使用了 JNI 实现，Android Studio 在某些条件进行编译 APK 的时候可能并不会把 Native 的 .so 库打包进去，导致出现这类报错，重新 Clean 一下工程，卸载重装即可。

allowBackup 异常?

- **报错信息：** `Manifest merger failed : Attribute application@allowBackup value=(true) from AndroidManifest.xml`
- **问题原因：** TUICallKit 依赖的 IM SDK 中默认 allowBackup 的值为 false ，表示关闭应用的备份和恢复功能。
- **解决方法：** 您可以在您工程的 `AndroidManifest.xml` 文件中删除 allowBackup 属性或将该属性改为 false，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。

TUICallKit 是否支持离线推送呢?

支持，Android 离线推送采用腾讯云的**TIMPush推送插件**以及**自集成推送**，接入方式详见：离线唤醒 。

在通话邀请超时时间内，被邀请者如果离线再上线，能否弹出通话界面?

单人通话时，如果在超时时间内上线，会触发来电邀请；群组通话，如果在超时时间内上线，会拉起未处理的20条群消息，如果存在通话邀请，则触发来电邀请。在不同的版本上来电的显示策略有所不同（详见下方：被叫端来电显示策略 ）。

应用在后台时，不能自动将通话界面拉取到前台

1. TUICallKit 2.3 及以上版本（ Android&iOS 发布日志 ）调整了被叫端的来电显示策略，见下方：被叫端来电显示策略 。
2. TUICallKit 2.3 版本之前，将应用从后台自动拉取到前台，需要检查 App 是否开启了“后台自启动”或“悬浮窗”权限。

   不同厂商、甚至同一厂商不同 Android 版本，其对于应用开放的权限以及权限名称也会存在不一致。例如，小米6只需要开启后台弹出界面权限，而红米需要同时打开后台弹出界面和显示悬浮窗权限。如何开启权限，详见下方：相关权限开启 。
3. 如果遇到以下场景拉不起通话界面，原因是：应用的启动堆栈变化，导致 CallKitActivity 界面被遮挡移除了。

   场景一：接通后退到后台，点击桌面图标进入应用，原通话界面消失；

   场景二：应用在后台时，开启banner的情况下，收到通知View，不点击View，点击桌面图标进入应用，拉不起通话界面且通知消失；

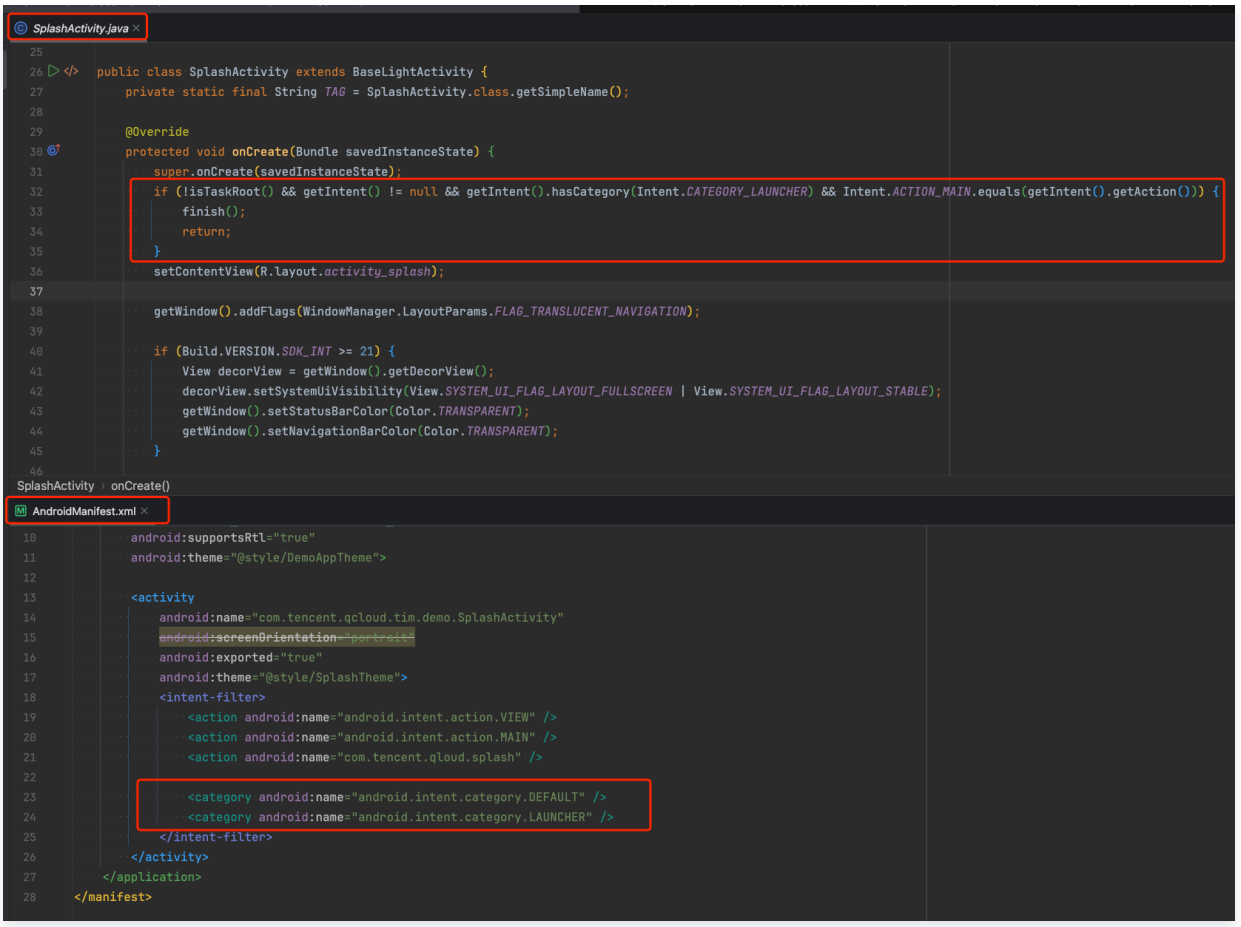
   场景三：应用在后台时，收到离线推送，不点击推送通知，点击桌面图标进入应用，无法拉起通话界面或通话界面闪了一下；

以上几种情况，需要在您自己业务的默认启动的主 Activity 中添加以下代码，每个应用默认启动的 Activity 都不一样，详见 AndroidManifest.xml 配置，以目前大部分应用的启动页 SplashActivity 为例：

添加代码

```
if (!isTaskRoot() && getIntent() != null && getIntent().hasCategory(Intent.CATEGORY_LAUNCHER)
        && Intent.ACTION_MAIN.equals(getIntent().getAction())) {
    finish();
    return;
}
```

具体添加位置

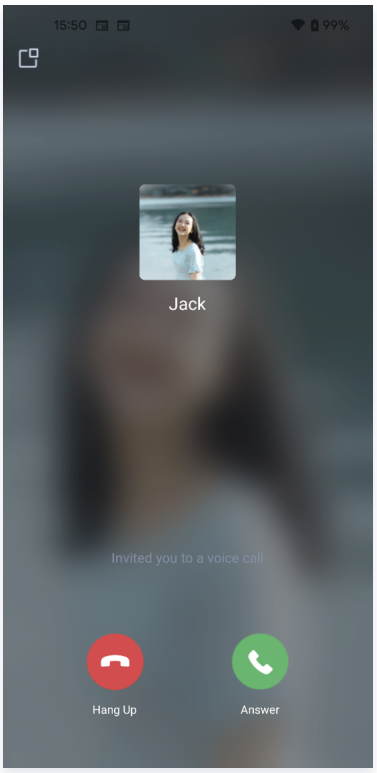


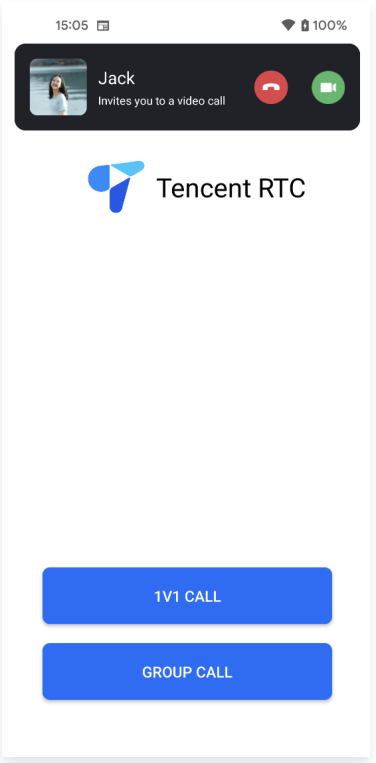
说明：
SplashActivity 的启动模式不能是 singleTask；singleTask 模式下 isTaskRoot 是 true，上述方法不生效。
如果您在开发中开启了所有权限或做了上述尝试，依然无法自动拉起通话界面到前台，请加入 TUICallKit 技术交流平台 zhiliao，联系我们协助处理。

被叫端来电显示策略

为使 TUICallKit 适应不同的业务需求，增加产品特色，提升用户的使用体验，TUICallKit **2.3 及以上版本**（发布日志），优化收到来电后的通话页面显示弹出策略，详情如下所示：

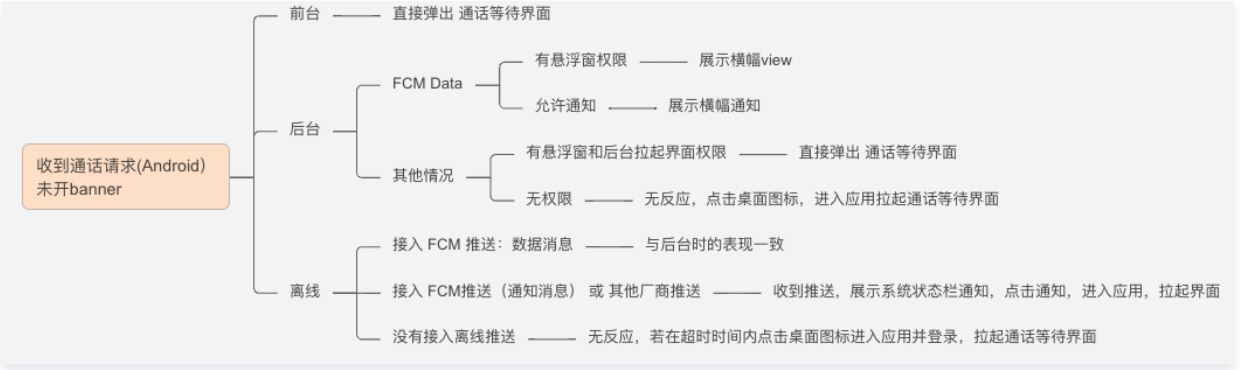
全屏显示通话等待界面	横幅显示来电请求





2.3.0.920版本及之后的版本

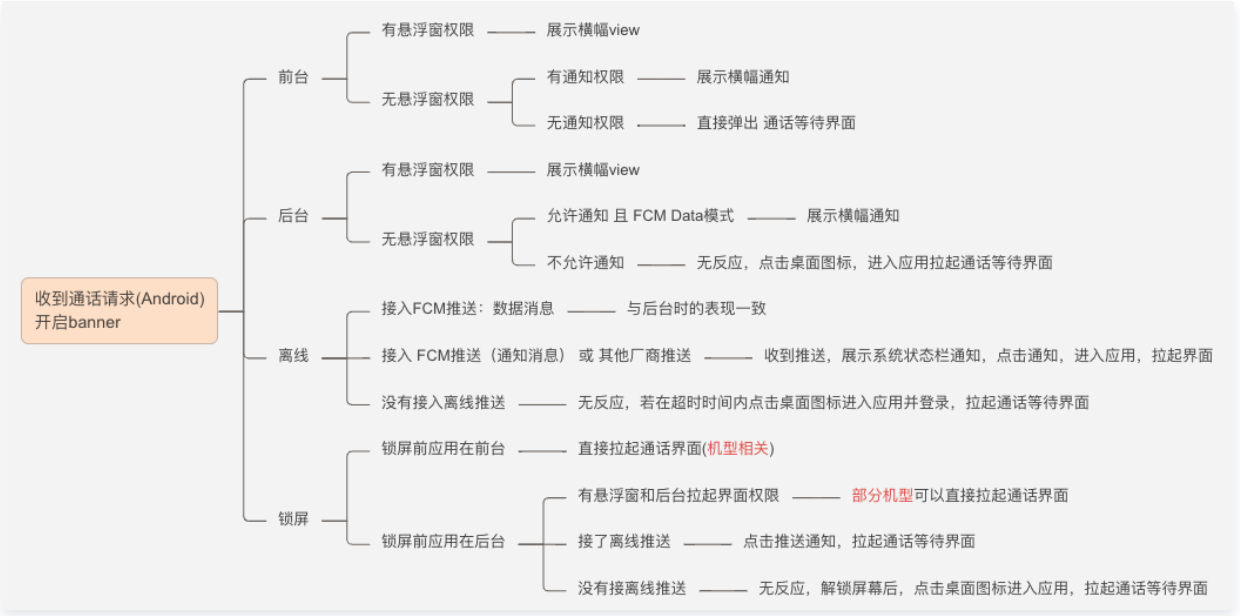
1. 如果您想要被叫端收到邀请时，尽量去拉起全屏通话界面，那么您可以更新 tuicallkit-kt 代码到最新。该情况下，被叫端的来电显示策略如下：



2. 如果您想在被叫端收到通话的时候，先展示一个横幅，然后根据需要拉起全屏通话界面，那么您可以调用以下接口开启该功能。

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

开启后，被叫端的来电策略如下所示，只要开启**悬浮窗权限**，就能尽可能的展示来电横幅。



2.3版本之前



说明：
相关权限如何开启，详见下方：相关权限开启 。

如果没有按照上述策略显示来电界面，请过滤 `onCallReceived` 日志，检查是否收到通话邀请，如果没有该日志的打印，请加入我们的 TUICallKit 技术交流平台 zhiliao ，联系我们协助处理。

相关权限开启

为实现良好的通话体验，建议您在应用中开启“通知”权限、“显示在其他应用上层（悬浮窗）”以及“后台拉起界面”权限，具体方法如下：

代码指引

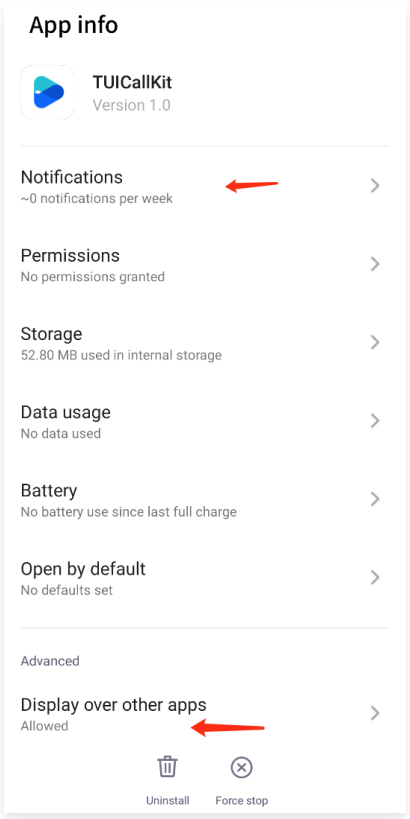
- **通知权限**：便于展示推送通知：请参见 通知运行时权限 和 请求运行时权限 根据业务需求自行实现。
- **悬浮窗权限**：用于展示自定义的来电横幅，以及通话悬浮窗。
- **后台拉起界面权限**：用于当应用在后台时拉起界面（例如：vivo手机）。

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

```
}
```

手动开启

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

Pixel 4a	小米	vivo
		

iOS

最近更新时间：2025-06-30 16:43:13

错误提示“The package you purchased does not support this ability”？

如遇以上错误提示，是由于您当前应用的音视频通话能力包过期或未开通，请参见 开通服务，领取或者开通音视频通话能力，进而继续使用TUICallKit组件。

如何购买音视频通话套餐?

请参见购买链接 开通正式版，如有其他问题，请点击页面右侧，进行售前套餐咨询；也可以加入我们的 TUICallKit 技术交流平台 zhiliao，进行咨询和反馈。

如何修改 TUICallKit 源码?

使用 `CocoaPods` 导入组件，具体步骤如下：

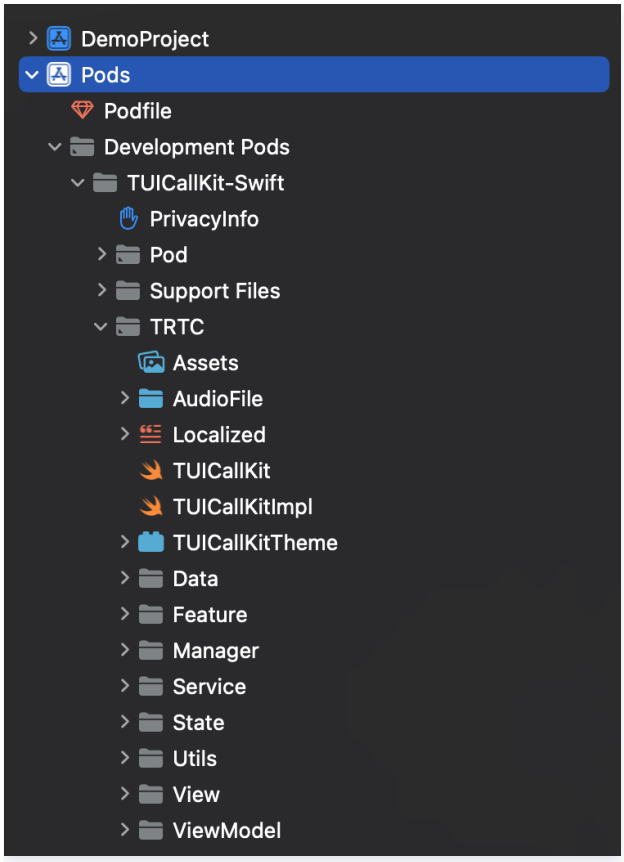
1. 在您的工程 `Podfile` 文件同一级目录下创建 `TUICallKit` 文件夹。
2. 单击进入 Github/TUICallKit ，选择克隆/下载代码，然后将 iOS 目录下的 TUICallKit_Swift 文件夹和 TUICallKit_Swift.podspec 文件拷贝到您在 `步骤1` 创建的 `TUICallKit` 文件夹下。
3. 在您的 `Podfile` 文件中添加以下依赖。

```
# :path => "指向 TUICallKit_Swift.podspec 的相对路径"
pod 'TUICallKit_Swift', :path => "TUICallKit/TUICallKit_Swift.podspec"
```

4. 执行 `pod install` 命令，完成导入。

> ⚠ **注意：**
> `TUICallKit_Swift` 文件夹和 `TUICallKit_Swift.podspec` 文件必须在同一目录下。

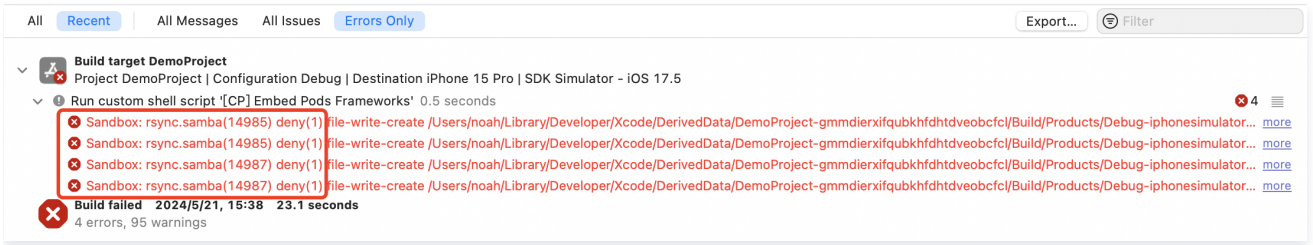
TUICallKit_Swift 组件集成后效果：



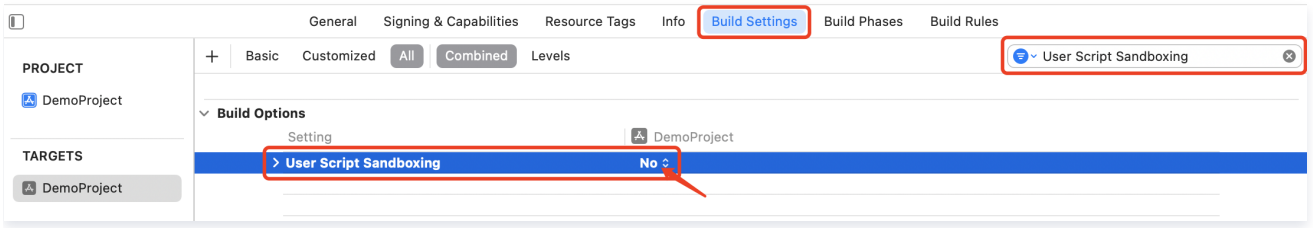
> **说明：**
> TUICallKit_Swift 组件集成后支持文件夹分层显示，方便您阅读和修改源代码。

Xcode 15 编译报错?

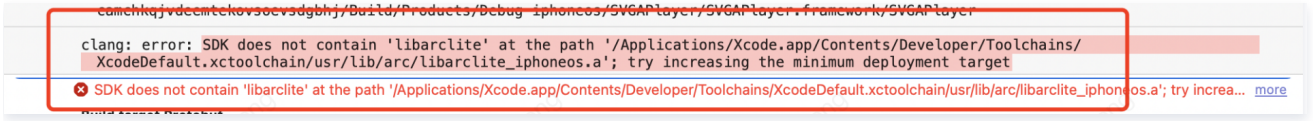
1、出现 Sandbox: rsync 编译报错



可以在 Build Settings 中把 User Script Sandboxing 设置为 NO：



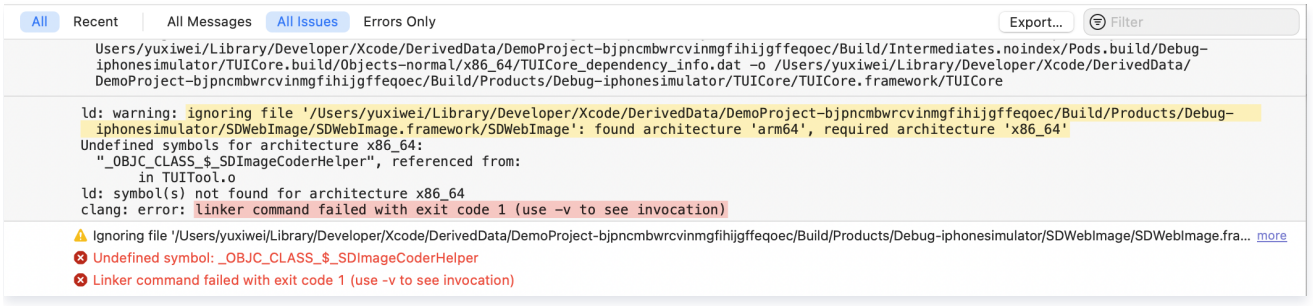
2、出现 SDK does not contain 编译报错



可以在 Podfile 添加如下代码：

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

post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '13.0'
    end
  end
end
```

3、如果在 M 系列电脑上运行模拟器，可能会出现 Linker command failed with exit code 1 (use -v to see invocation) 编译报错



可以在 Podfile 添加如下代码：

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

post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['EXCLUDED_ARCHS[sdk=iphonesimulator*]'] = "arm64"
    end
  end
end
```

TUICallKit 和自己集成的音视频库冲突了?

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

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

```
pod 'TUICallKit_Swift'
```

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

```
pod 'TUICallKit_Swift/Professional'
```

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

TUICallKit 是否可以不引入 IM SDK，只使用 TRTC?

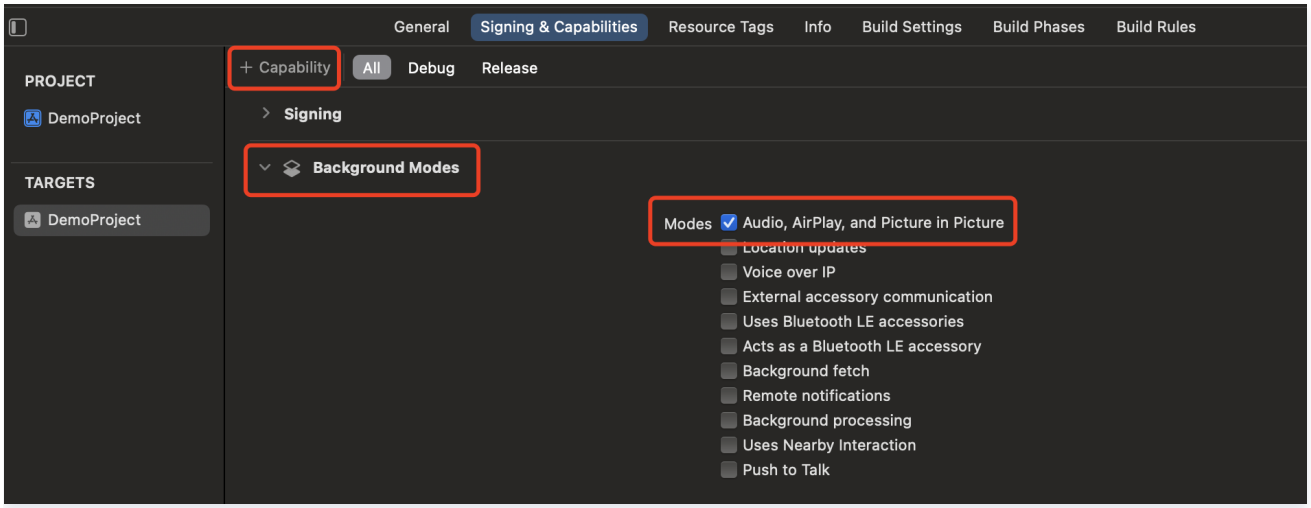
不可以。`TUIKit` 全系组件都使用了腾讯云 IM SDK 作为通信的基础服务，比如通话拨打信令、通话忙线信令等核心逻辑，如果您已经购买有其他 `IM` 产品，也可以参照 `TUICallKit` 逻辑进行适配。

TUICallKit 组件支持自定义铃声吗?

支持，调用 setCallingBell 即可。

TUICallKit 是否支持后台运行?

支持，如需要进入后台仍然运行相关功能，可选中当前工程项目，在 Capabilities 下的 Background Modes 模块中勾选 Audio，AirPlay and Picture in Picture ，如下图所示：



如何查看 TRTC 日志?

TRTC 的日志默认压缩加密，后缀为 .xlog。日志是否加密是可以通过 setLogCompressEnabled 来控制，生成的文件名里面含 C(compressed) 的就是加密压缩的，含 R(raw) 的就是明文的。
iOS：`sandbox的Documents/log`

> **说明：**
> - 查看 .xlog 文件需要下载 解密工具，在 Python 2.7环境中放到 xlog 文件同目录下直接使用 `python decode_mars_log_file.py` 运行即可。
> - 查看 .clog 文件（9.6 版本以后新的日志格式）需要下载 解密工具，在 Python 2.7 环境中放到 clog 文件同目录下直接使用 `python decompress_clog.py` 运行即可。

uni-app（客户端）

最近更新时间：2024-08-09 17:40:21

uni-app 集成 TUICallKit 如何做离线唤醒?

建议您结合 TUIPush 插件 实现离线唤醒。

应用在后台时，不能自动将通话界面拉取到前台?

详情请参见：（Android）应用在后台时，收到通话不能到前台 。

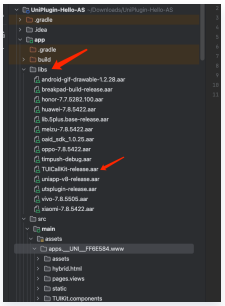
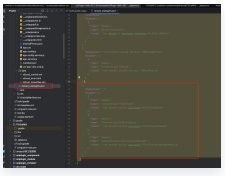
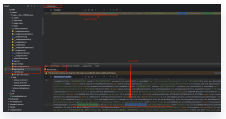
支持云端打包、本地打包，以及离线打包三种方式

TUICallKit 插件支持以下三种打包方式，前两种在 HBuilderX 开发工具中，适用于前端开发人员；离线打包适用于原生开发人员。
方式一：云端打包，（**推荐**采用该方式打包，方便后续升级）。
方式二：本地打包。下载插件到本地，在 HBuilderX 工程中创建 **nativeplugins** 目录，将下载的插件放到该目录下；点击 **manifest.json** 文件，点击 APP 原生插件配置，选择本地插件。



方式三：离线打包。Android 离线打包 uniapp 官网文档较分散，可按照下述步骤进行：

- 按照官网文档 Android 离线打包，集成 Android 工程，直接使用 uniapp 官网提供的方式二，跑通官网 demo，减少踩坑。其中“配置工程”的第三步“**资源配置**”很重要，一定要仔细按照文档完成。
- 从官网下载 TUICallKit 插件，拷贝 TUICallKit-release.aar 放到 AndroidStudio工程的 **libs** 目录下。
- 在 app/build.gradle 文件中增加依赖（下载的插件的 package.json 文件有依赖项）。
- 在 **dcloud_uniplugins.json** 文件中添加 module 的引入。
- 检查 js 上层引用了 TUICallKit 插件，并有相关的调用。完成这些步骤，插件已引入，可以直接打包生成 apk。

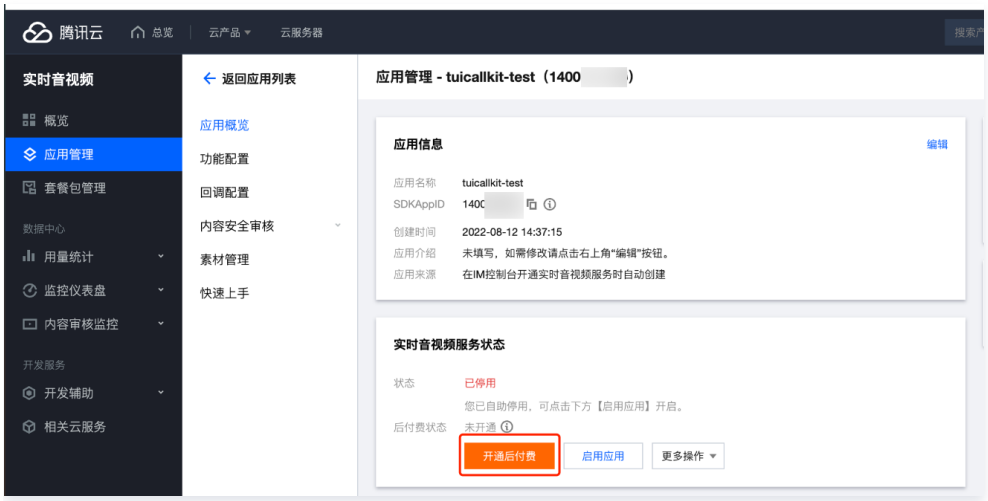
拷贝插件到 libs 目录下	在 app/build.gradle 中增加依赖	在 dcloud_uniplugins.json 文件中添加 module 的引入	上层调用
			

提示“[-100013]:TRTC service is suspended...”？

单击**免费体验**以后，部分之前使用过 实时音视频 TRTC 服务的用户会提示：

```
[-100013]:TRTC service is  suspended. Please check if the package balance is 0 or the Tencent Cloud
accountis in arrears
```

因为新的 IM 音视频通话能力是整合了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础的 PaaS 服务，所以当 实时音视频 TRTC 的免费额度（10000分钟）已经过期或者耗尽，就会导致开通此项服务失败，这里您可以单击 TRTC 控制台，找到对应 SDKAppID 的应用管理页，示例如图，开通付费功能后，再次**启用应用**即可正常体验音视频通话能力。



SDKAppID、sercretKey 参数说明

- **SDKAppID**：在 步骤一：开通 TRTC 服务 中的最后一步中您已经获取到，这里不再赘述。
- **userID**：当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
- **userSig**：使用 步骤一：开通 TRTC 服务 中获取的 SecretKey 对 SDKAppID、userID 等信息进行加密，就可以得到 userSig，它是一个鉴权用的票据，用于腾讯云识别当前用户是否能够使用 TRTC 的服务。您可以通过控制台中的 **辅助工具** 生成一个临时可用的 userSig。
- 更多信息请参见 如何计算及使用 userSig 。

> ⚠ **注意**
>
> - **这个步骤也是目前我们收到的开发者反馈最多的步骤，常见问题如下：**
>   - SDKAppID 设置错误，国内站的 SDKAppID 一般是以140开头的10位整数。
>   - userSig 被错配成了加密密钥（Secretkey），userSig 是用 SecretKey 把 SDKAppID、userID 以及过期时间等信息加密得来的，而不是直接把 Secretkey 配置成 userSig。
>   - userID 被设置成“1”、“123”、“111”等简单字符串，由于 **TRTC 不支持同一个 UserID 多端登录**，所以在多人协作开发时，形如“1”、“123”、“111”这样的 userID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 userID。
> - Github 中的示例代码使用了 genTestUserSig 函数在本地计算 userSig 是为了更快地让您跑通当前的接入流程，但该方案会将您的 SecretKey 暴露在 App 的代码当中，这并不利于您后续升级和保护您的 SecretKey，所以我们强烈建议您将 userSig 的计算逻辑放在服务端进行，并由 App 在每次使用 TUICallKit 组件时向您的服务器请求实时计算出的 userSig。

自定义调试基座注意事项

- 什么是自定义基座：参见 uni-app 官网自定义基座说明 。
- 自定义基座不是正式版，真正发布时，需要再打正式包。使用自定义基座是无法正常升级替换 APK 的。
- 请尽量不要使用本地插件，插件包超过自定义基座的限制，可能导致调试收费。
- 使用自定义基座打包 uni 原生插件（**请使用真机运行自定义基座**）
- 使用 uni 原生插件必须先提交云端打包才能生效，购买插件后在应用的 `manifest.json` 页面的 **App原生插件配置** 项下单击**选择云端插件**，选择**腾讯云原生音视频插件**。
- 直接云端打包后无法打 log，无法排查问题，需要自定义基座调试原生插件。

如何创建群组通话的 groupID

群组的创建详见： IM 群组管理 ，或者您也可以直接使用 IM TUIKit，一站式集成聊天、通话等场景。

虚拟背景设置失效

请升级 HbuilderX 版本 ⩾ 3.94。

和 livepusher 推流模块同时使用，打包冲突?

结论：
目前两者无法同时使用。
原因：
uni-app 官网文档介绍 第三方 SDK 使用 LiteAVSDK（weex_livepusher-release.aar）做直播推流，和 uniapp TUICallKit 使用的 LiteAVSDK_TRTC 类冲突。和 uni-app 技术沟通后，对方反馈不清楚 SDK 属于哪个版本类型，因此 uni-app 他们无法对 livepusher 进行升级。
解决方案：



uni-app（小程序）

最近更新时间：2024-09-02 17:35:11

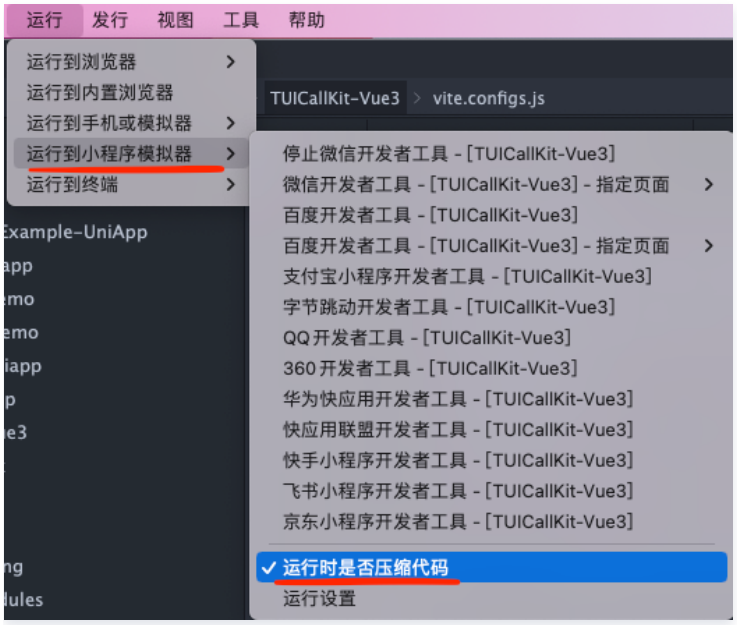
vue2 和 vue3 打包小程序包体积常见问题和方案

1、是否压缩代码?

以 TUICallKit-Vue2 demo 举例，下图给出了是否勾选**运行时是否压缩代码**的效果对比（**851KB -> 454KB，降低 50%**）：



因此：HBuilder 中**运行 > 运行到小程序模拟器 > 勾选运行时是否压缩代码**，具体参考下图：



≤ v1.4.4

2、依赖包如何不打入到主包的 `common/vendor.js`？

在使用 **微信开发者工具** 里 **构建 npm** 时，我们希望 HBuilder 打包小程序时，不将 `node_module` 中的依赖打入到主包的 `common/vendor.js` 中（**这会导致依赖存在多份，影响主包体积**）。

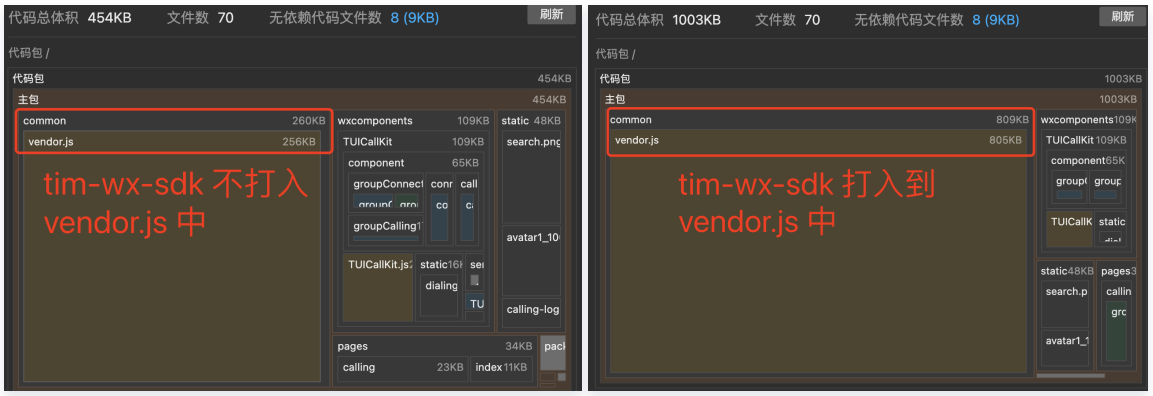
Vue2

推荐通过配置 `vue.config.js` 实现。

- `vue.config.js` **可选配置文件需放置在项目根目录下。**

- `vue.config.js` 一般用于配置 `webpack` 等编译选项。HBuilder 采用 vue2 开发小程序，构建、打包时 `vue.config.js` 会被自动加载。

以 TUICallKit-Vue2 demo 举例，下图给出依赖包不打入主包的 `common/vendor.js` 的效果对比（805KB -> 256KB，降低 68%）：



具体使用如下：

1. 页面中通过 `require` 引入包依赖：

```
// 页面中引入方式必须使用 require，不能使用 import 的方式
const TIM = require('tim-wx-sdk');
```

2. `vue.config.js` 配置如下：

```
// vue.config.js 配置
module.exports = {
  configureWebpack: {
      externals: {
          'tim-wx-sdk': 'commonjs tim-wx-sdk', // 550KB 左右
      },
  }
}
```

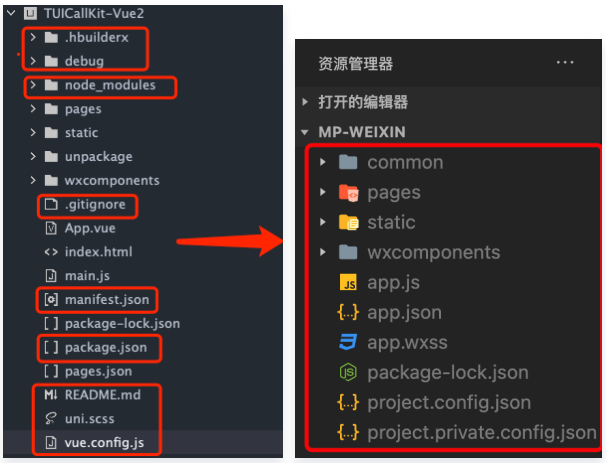
具体目录结构如下截图：



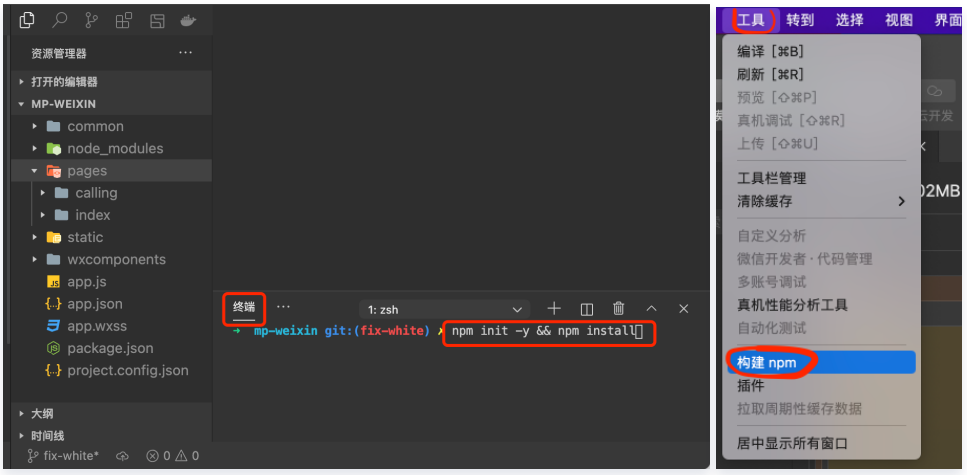
> **注意：**
> **页面中引入依赖包的方式必须使用 require，不能使用 import 方式！**
> 原因：通过 `import` 引入时，HBuilder 打包小程序后，会将 `import` 引入方式进行编译。从而导致无法使用 `node_modules` 中的 `tim-wx-sdk` 。而 `require` 的方式则不会被编译。

3、如何构建 npm?

1. 通过和 uni-app 的技术人员沟通了解到，目前 HBuilder 中不支持类似**微信开发者工具**中的**构建 npm**。而且他们也没有支持的计划。
2. HBuilder 中**运行 > 运行到小程序模拟器**生成的小程序项目，会清除 `package.json` 、 `node_modules` 等文件。具体如下图：



如果想在**微信开发者工具 > 工具 > 构建 npm**，就必须在**微信开发者工具的终端里**创建 `package.json` 、 `npm install` 安装依赖。然后才能**构建 npm**。具体如下：

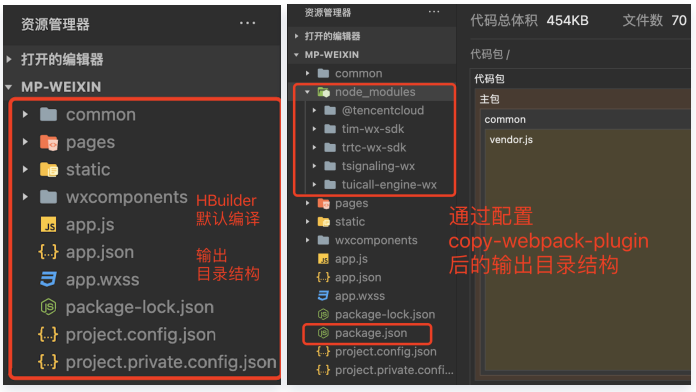



```
npm init -y && npm install xx
```

4、如何使用 `copy-webpack-plugin` 提升效率?

上面 3. 如何构建 npm 中，需要在 **微信开发者工具** 的终端里创建 `package.json` ，然后安装依赖才行。通过 `copy-webpack-plugin` 可减少该步骤，并提升效率。

以 TUICallKit-Vue2 demo 举例，下图给出使用 `copy-webpack-plugin` 的前后对比：



具体使用如下：

1. 安装 `copy-webpack-plugin`：

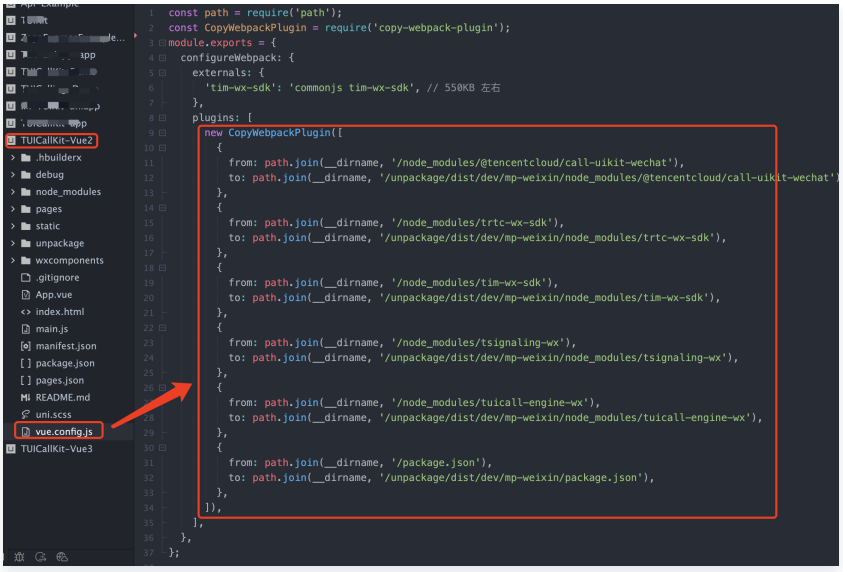
> ⚠ **注意：**
> **最新版本 copy-webpack-plugin 插件暂不兼容，推荐 v5.0.0（锁死版本）**

```
npm install -D copy-webpack-plugin@5.0.0
```

2. 在 `vue.config.js` 中使用 `copy-webpack-plugin` 插件，具体如下：

```
// vue.config.js 配置
const path = require('path');
const CopyWebpackPlugin = require('copy-webpack-plugin');
module.exports = {
  configureWebpack: {
      plugins: [
          new CopyWebpackPlugin([
              {
                  from: path.join(__dirname, '/node_modules/@tencentcloud/call-uikit-
wechat'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-
weixin/node_modules/@tencentcloud/call-uikit-wechat')
              },
              {
                  from: path.join(__dirname, '/node_modules/trtc-wx-sdk'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-weixin/node_modules/trtc-
wx-sdk')
              },
              {
                  from: path.join(__dirname, '/node_modules/tim-wx-sdk'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-weixin/node_modules/tim-
wx-sdk')
              },
              {
                  from: path.join(__dirname, '/node_modules/tsignaling-wx'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-
weixin/node_modules/tsignaling-wx')
              },
              {
                  from: path.join(__dirname, '/node_modules/tuicall-engine-wx'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-
weixin/node_modules/tuicall-engine-wx')
              },
              {
                  from: path.join(__dirname, '/package.json'),
                  to: path.join(__dirname, '/unpackage/dist/dev/mp-weixin/package.json')
              }
          ])
      ]
  }
}
```

具体目录结构如下截图：



Vue3

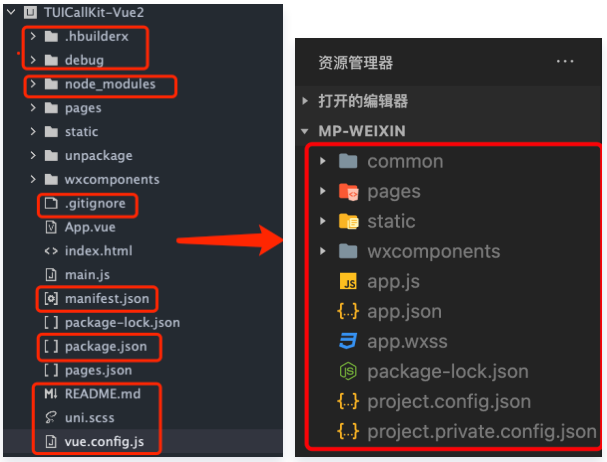
不用配置 `vite.config.js`，只需通过 `require` 引入即可。具体如下：

```
// 页面中引入方式必须使用 require，不使用 import
const TIM = require('tim-wx-sdk');
```

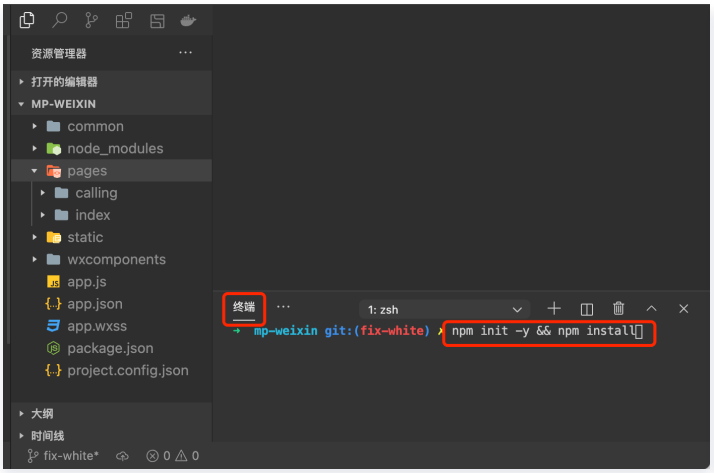
具体可参考：TUICallKit-Vue3 demo 。

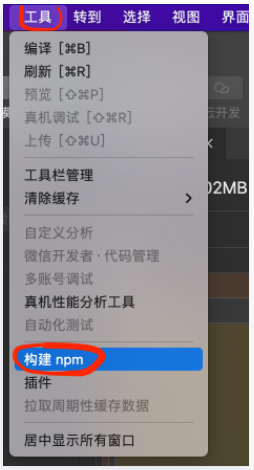
如何构建 npm?

1. 通过和 uni-app 的技术人员沟通了解到，目前 HBuilder 中不支持类似**微信开发者工具**中的**构建 npm**。而且他们也没有支持的计划。
2. HBuilder 中**运行 > 运行到小程序模拟器**生成的小程序项目，会清除 `package.json` 、 `node_modules` 等文件。具体如下图：



如果想在**微信开发者工具 > 工具 > 构建 npm**，就必须在**微信开发者工具的终端里**创建 `package.json` 、 `npm install` 安装依赖。然后才能**构建 npm**。具体如下：





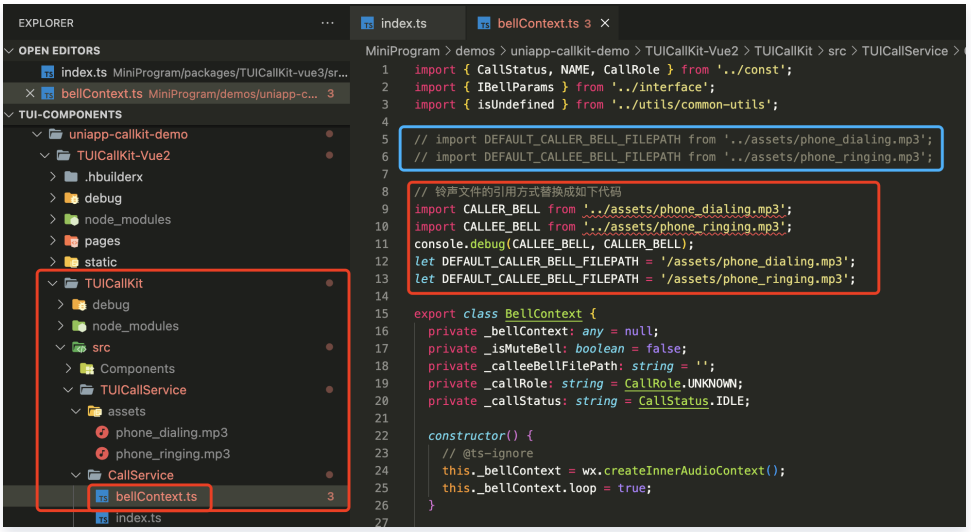

```
npm init -y && npm install tuicall-engine-wx@1.5.6
```

Vue2 uni-app 打包微信小程序项目中，铃声没有声音?

这个是因为在 vue2 中，uni-app 打包，将铃声文件编码为 base64 编码，这个编码在播放铃声的 api 那里是不支持的。
您可采取如下步骤解决该问题：

- 第一步，在项目的 `TUICallKit` 源码中，修改铃声的引入方式。

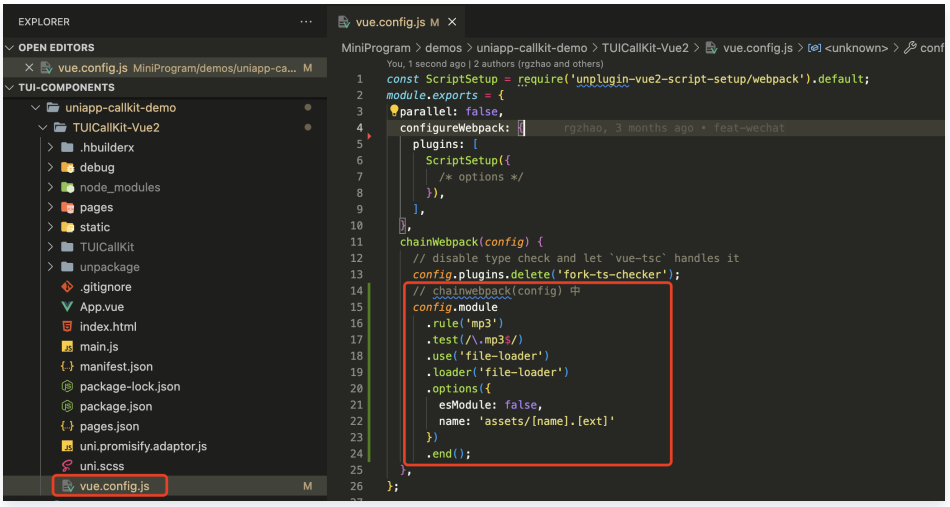
```
// 铃声文件的引用方式替换成如下代码
import CALLER_BELL from '../assets/phone_dialing.mp3';
import CALLEE_BELL from '../assets/phone_ringing.mp3';
console.debug(CALLEE_BELL, CALLER_BELL);
let DEFAULT_CALLER_BELL_FILEPATH = '/assets/phone_dialing.mp3';
let DEFAULT_CALLEE_BELL_FILEPATH = '/assets/phone_ringing.mp3';
```



- 第二步，修改项目中的 `vue.config.js` 文件。

```
// chainwebpack(config) 中
config.module
      .rule('mp3')
      .test(/\.mp3$/)
```

```
        .use('file-loader')
        .loader('file-loader')
        .options({
          esModule: false,
          name: 'assets/[name].[ext]'
        })
        .end();
```



包体积过大?

> **注意:**
> TUICallKit 版本须 ≥ 3.1.2。

- 体积优化文档。

什么是 SDKAppID、SecretKey?

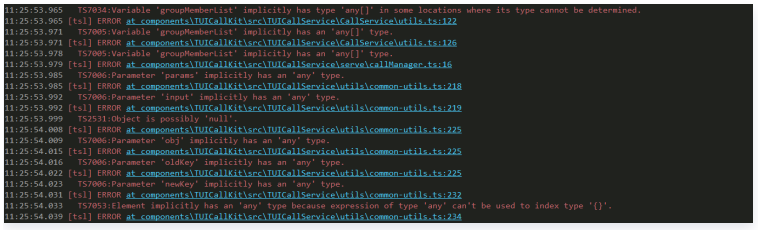
- **SDKAppID**：IM 的应用 ID，用于业务隔离，即不同的 SDKAppID 的通话彼此不能互通。
- **Secretkey**：IM 的应用密钥，需要和 SDKAppID 配对使用，用于签出合法使用 IM 服务的鉴权用票据 UserSig。

什么是 UserSig，如何生成 UserSig?

- UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。
- UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig。

如何处理 TUICallKit 的 ts 语法校验错误?

如果出现如下所示 error, 请在项目中添加如下配置。



.eslintignore

```
TUICallKit
```

tsconfig.json

```
{
  "compilerOptions": {
      "noImplicitAny": true
   }
}
```

如何关闭美颜?

- web 端默认无美颜。
- 小程序端调用 setBeautyLevel API，传入参数为0时，会关闭默认美颜。

```
TUICallKitServer.getTUICallEngineInstance().setBeautyLevel(0);
```

微信小程序

最近更新时间：2024-09-02 17:35:11

> **说明：**
> 如果并没有解决您的问题，也欢迎您加入我们的 TUICallKit 技术交流 QQ 群： **605115878**，来进行交流讨论。

是否支持 uniapp、Taro 等开发环境?

Taro 暂不支持，uniapp 请参见 快速集成方案（uni-app）。

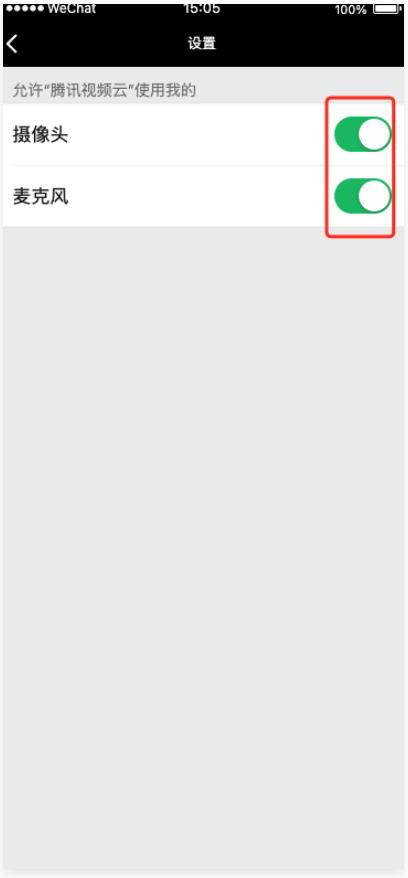
当摄像头/麦克风权限被拒绝，如何处理?

首次进入小程序需要默认是**没有设备权限**的，需要手动开启权限。

1. 打开小程序的设置。



2. 开启摄像头/麦克风权限。



报错 "authorize:fail:appid privacy api banned"，如何处理?

由于 TRTC 涉及到麦克风与摄像头的权限问题，需要用户在小程序控制台中更新隐私保护政策，具体步骤如下：

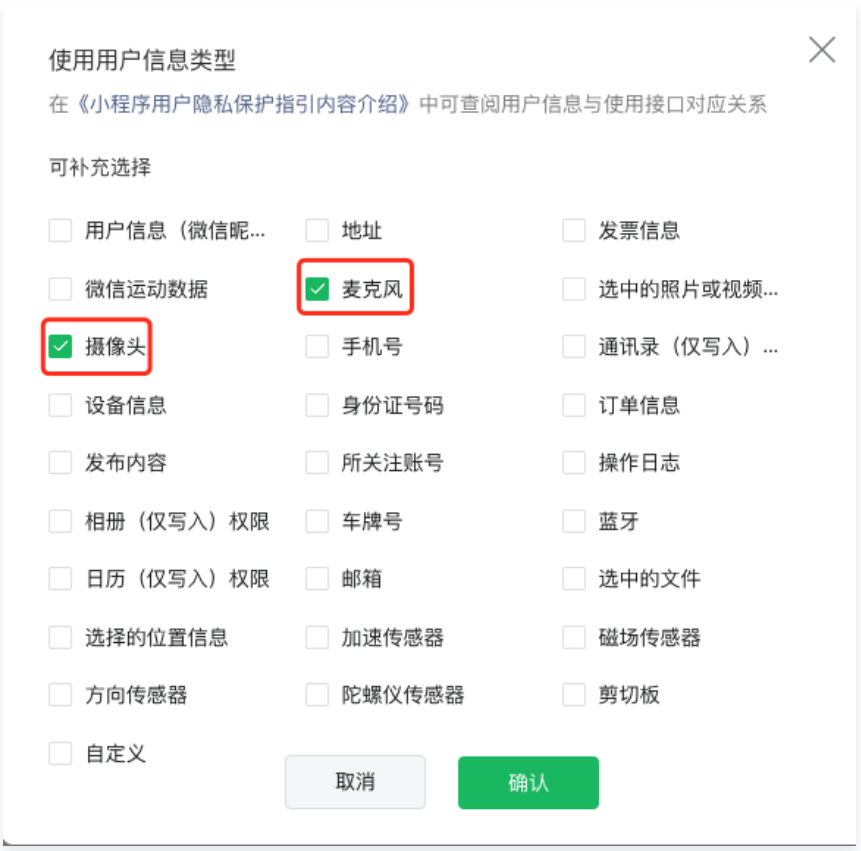
1. 登录腾讯平台的小程序后台，点击设置栏目。



2. 选择用户隐私保护指引设置功能，点击更新。



3. 在弹出的表格中，勾选小程序会涉及到的类型。



出现推流黑屏，样式错乱怎么回事?

警告:

微信小程序 Skyline 渲染引擎与 live-pusher 存在兼容性问题，如果您使用微信开发者工具 1.06.2307260+ 版本以上的新创建的项目，请在 app.json 中删除 Skyline 的相关配置。

```
"renderer": "skyline",
"rendererOptions": {
  "skyline": {
    "defaultDisplayBlock": true,
    "disableABTest": true,
    "sdkVersionBegin": "3.0.0",
    "sdkVersionEnd": "15.255.255"
  }
},
"componentFramework": "glass-easel",
"sitemapLocation": "sitemap.json",
"lazyCodeLoading": "requiredComponents"
```

什么是 SDKAppID、SecretKey?

- **SDKAppID**：IM 的应用 ID，用于业务隔离，即不同的 SDKAppID 的通话彼此不能互通。
- **Secretkey**：IM 的应用密钥，需要和 SDKAppID 配对使用，用于签出合法使用 IM 服务的鉴权用票据 UserSig。

什么是 UserSig，如何生成 UserSig?

- UserSig 是用户登录即时通信 IM 的密码，其本质是对 UserID 等信息加密后得到的密文。
- UserSig 签发方式是将 UserSig 的计算代码集成到您的服务端，并提供面向项目的接口，在需要 UserSig 时由您的项目向业务服务器发起请求获取动态 UserSig。更多详情请参见 服务端生成 UserSig 。

错误提示“The package you purchased does not support this ability”，如何处理?

如遇以上错误提示，是由于您当前应用的音视频通话能力包过期或未开通，领取或者开通音视频通话能力，进而继续使用 TUICallKit 组件。

注意：

单击**免费体验**以后，部分之前使用过 实时音视频 TRTC 服务的用户会提示：

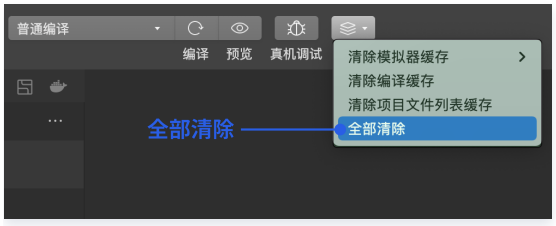
```
[-100013]:TRTC service is  suspended. Please check if the package balance is 0 or the Tencent Cloud accountis in arrears
```

当实时音视频 TRTC 的免费额度（10000分钟）已经过期或者耗尽，就会导致开通此项服务失败，这里您可以单击 TRTC 控制台，找到对应 SDKAppID 的应用管理页，开通后付费功能后，再次**启用应用**即可正常体验音视频通话能力。具体参见下图：



微信开发者工具有路径报错，如何处理?

如果微信开发者工具有路径保存，请清除缓存，避免开发者工具的缓存造成渲染异常。具体参见下图：



如何关闭美颜?

- web 端默认无美颜。
- 小程序端调用 setBeautyLevel API，传入参数为0时，会关闭默认美颜。

```
TUICallKitServer.getTUICallEngineInstance().setBeautyLevel(0);
```

Flutter

最近更新时间：2024-08-06 17:37:31

同时集成 tencent_calls_uikit 和 tencent_trtc_cloud，或同时集成 tencent_calls_uikit 和 live_flutter_plugin，出现符号冲突报错，怎么解决?

问题详情：当引入 flutter "tencent_calls_uikit" 进我们现有的 project 后，Android 构建 APK 时出现如下报错：

```
Duplicate class com.tencent.liteav.LiveSettingJni found in modules jetified-LiteAVSDK_Professional-
10.7.0.13053-runtime
(com.tencent.liteav:LiteAVSDK_Professional:10.7.0.13053) and jetified-LiteAVSDK_TRTC-10.3.0.11225-
runtime
(com.tencent.liteav:LiteAVSDK_TRTC:10.3.0.11225)
```

iOS 在执行 `pod install` 出现如下报错：

```
[!] The 'Pods-Runner' target has frameworks with conflicting names: txsoundtouch.xcframework and
txffmpeg.xcframework.
```

这个问题原因是您使用的 tencent_calls_uikit 和 tencent_trtc_cloud 分别依赖于 TRTC Android SDK 的专业版和精简版，这个问题我们已经在最新的版本中解决，只需要您将 tencent_calls_uikit 和 tencent_trtc_cloud 升级到最新版本即可。

Flutter Android未添加混淆设置，怎么设置?

如果您需要编译运行在 Android 平台，由于我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 中的部分类加入不混淆名单，因此需要您在 proguard-rules.pro 文件中添加如下代码：

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

从1.8.0以下的版本升级到1.8.0及以上的版本，导致编译报错或者拉不起页面问题如何修复?

如果是从1.8.0以下升级到1.8.0及以上的版本，需要您检查下面几个步骤是否正常：

1. 将 navigatorObservers 添加到 MateriaApp。目的是在收到呼叫邀请时导航到 TUICallKit 页面。示例代码如下：

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

MaterialApp(
    navigatorObservers: [TUICallKit.navigatorObserver],
    ...
)
```

2. tencent_calls_engine 插件中的导入文件统一替换为新的。

```
import 'package:tencent_calls_engine/tuicall_engine.dart';
import 'package:tencent_calls_engine/tuicall_observer.dart';
import 'package:tencent_calls_engine/tuicall_define.dart';
```

上图内容替换为下图所示：

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

3. 登录 API 调整更规范，不需要再指定参数。

```
return await callsUIKitPlugin.login(
    sdkAppId: GenerateTestUserSig.sdkAppId,
    userId: userInfo.userId,
    userSig: GenerateTestUserSig.genTestSig(userInfo.userId));
```

```
return await callsUIKitPlugin.login(GenerateTestUserSig.sdkAppId,
    userInfo.userId, GenerateTestUserSig.genTestSig(userInfo.userId));
}
```

1.8.0以上版本不需要指定参数

4. 离线推送参数构造优化。

```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
offlinePushInfo.title = "Flutter TUICallKit";
offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
TUICallParams params = TUICallParams(offlinePushInfo: offlinePushInfo);
callsUIKitPlugin.call(callUserId, TUICallMediaType.audio, params);
}
```

```
TUIOfflinePushInfo offlinePushInfo = TUIOfflinePushInfo();
offlinePushInfo.title = "Flutter TUICallKit";
offlinePushInfo.desc = "This is an incoming call from Flutter TUICallkit";
TUICallParams params = TUICallParams();
params.offlinePushInfo = offlinePushInfo;
callsUIKitPlugin.call(callUserId, TUICallMediaType.audio, params);
```