即时通信 IM

在线直播（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

在线直播（含 UI）
- 组件介绍（TUILiveKit）
- 开通服务（TUILiveKit）
- 跑通 Demo（TUILiveKit）
  - Android
  - iOS
  - Web
  - Flutter
  - Electron
- 快速接入（TUILiveKit）
  - Android
  - iOS
  - Web
  - Flutter
  - Electron
- 界面定制（TUILiveKit）
  - Android&iOS
  - Web
  - Flutter
- 开播和观看（TUILiveKit）
  - Android
  - iOS
- 悬浮窗（TUILiveKit）
  - Android
  - iOS
- 房间列表（TUILiveKit）
- 关注主播（TUILiveKit）
- 主播连线（TUILiveKit）
- 主播 PK（TUILiveKit）
- 观众连麦（TUILiveKit）
- 至臻画质（TUILiveKit）
- 互动礼物（TUILiveKit）
- 礼物特效（TUILiveKit）
- 美颜特效（TUILiveKit）
- 互动弹幕（TUILiveKit）
- 客户端 API（TUILiveKit）

版权所有：腾讯云计算（北京）有限责任公司

Android
VideoLiveKit
LiveCoreView
LiveCoreViewDefine
iOS
VideoLiveKit
LiveCoreView
LiveCoreViewDefine
Web
TUILiveKit API
服务端 API（TUILiveKit）
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
用户管理
账号管理
导入单个账号
导入多个账号
查询账号
用户资料
设置资料
拉取资料
房间管理
创建房间
更新房间
获取房间信息
获取直播列表
解散房间
获取房间自定义信息
设置房间自定义信息
删除房间自定义信息
房间成员管理
成员禁言
获取禁言列表

成员封禁
解除成员封禁
获取封禁列表
修改管理员
获取管理员列表
获取房间成员列表
连麦管理
获取麦位列表
用户上麦
用户下麦
锁定麦位
PK 相关
发起 PK
修改 PK 分数
获取 PK 信息
云端搜索
房间成员搜索
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
更新回调配置
删除回调配置
房间相关
创建房间之后回调
解散房间之后回调
更新房间信息之后回调
设置房间自定义信息后回调
删除房间自定义信息后回调
创建房间达到70%预警回调
用户相关
房间成员状态变更回调
连麦相关
麦位列表变更之后回调
PK 相关
创建 PK 之后回调
PK 开启之后回调

版权所有：腾讯云计算（北京）有限责任公司

PK 结束之后回调
错误码（TUILiveKit）
发布日志（TUILiveKit）
常见问题（TUILiveKit）

在线直播（含 UI）组件介绍（TUILiveKit）

最近更新时间：2025-06-26 16:22:22

组件介绍

TUILiveKit 是一款适用于社交娱乐、购物、健身等互动直播场景的产品，通过集成该产品，仅需三步，30分钟内就可以为您的 App 添加互动连麦、送礼、房间管理等功能，快速上线业务。基本功能展示如下图：



支持平台

平台	Android	iOS	桌面端	Flutter
是否支持	✓	✓	✓	✓
支持语言/框架	• Java • Kotlin	• Swift • Objective-C	• Electron（只支持 Windows）	• Dart

功能介绍

基础功能

	功能描述
高清直播	主播可以在直播前预览画面，支持开启和关闭直播。
直播观看	支持多人同时观看直播。
单房间连麦	支持单房间内与多位观众连线，以及跨房与其他多位主播连线。
跨房连线/PK	支持双人、多人跨房 PK。
在线观众列表	观众列表组件支持展示直播间最新进房的前100位观众及总人数，提供简化版和详情版视图展示功能。
聊天弹幕	用户可发送表情和文本消息弹幕，实时展示于直播间内。
心动点赞	支持点赞功能，并统计基础点赞数据。
礼物特效	提供弹幕动画、全屏动画（支持 SVGA 格式）及点赞动画的播放功能。

高级功能

	功能描述
成员管理：拉黑禁言	支持主播对观众进行拉黑和禁言操作。
直播预览	观众可以不进入直播间，在直播间外预览直播间的内容。
直播瀑布流	观众可以通过上下滑动屏幕的方式查看不同的直播间，支持单列和双列两种形式的瀑布流 UI。
礼物系统	展示礼物面板，支持配置礼物名称、价格、等级等信息。
关注	观众可以在直播间关注主播及其他观众。
直播间搜索	支持搜索直播间内的成员。
至臻画质	在保证画质的前提下显著降低码率，或在相同比特率下提升画质，优化带宽受限环境下的观看体验。
PC推流助手	支持多机位画面采集和本地音视频混流，适用于 PC 端专业推流。
RTMP推流/输入在线媒体流	通过RTMP协议将本地视频和音频推送至直播间，支持主播OBS开播、虚拟人直播等多元场景。

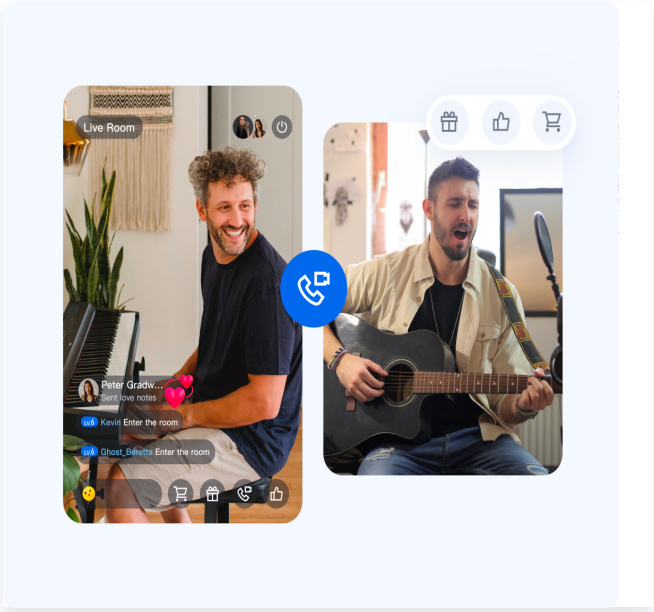
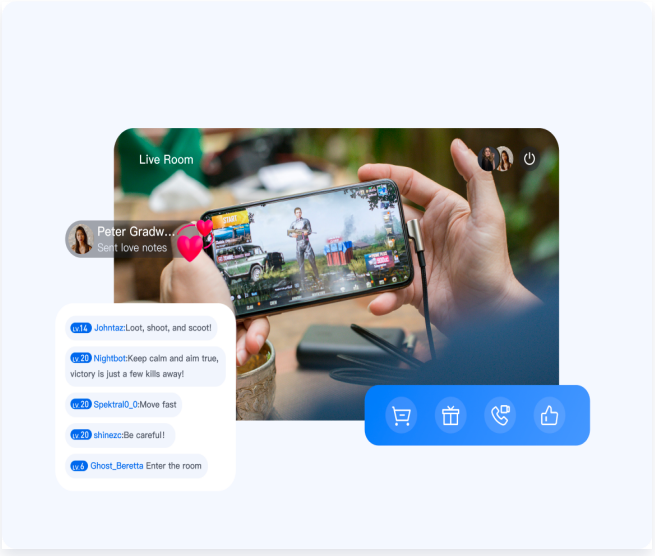
版权所有：腾讯云计算（北京）有限责任公司

QUIC弱网稳定传输	可以根据终端的网络状况，自动选择切换网络链路或多链路传输，从而在弱网环境下提升传输速度。
腾讯特效	在默认美颜基础上，支持集成 腾讯特效，提供更全面高级的美颜功能。
音效变声	支持开播前设置音量、变声和混响效果。

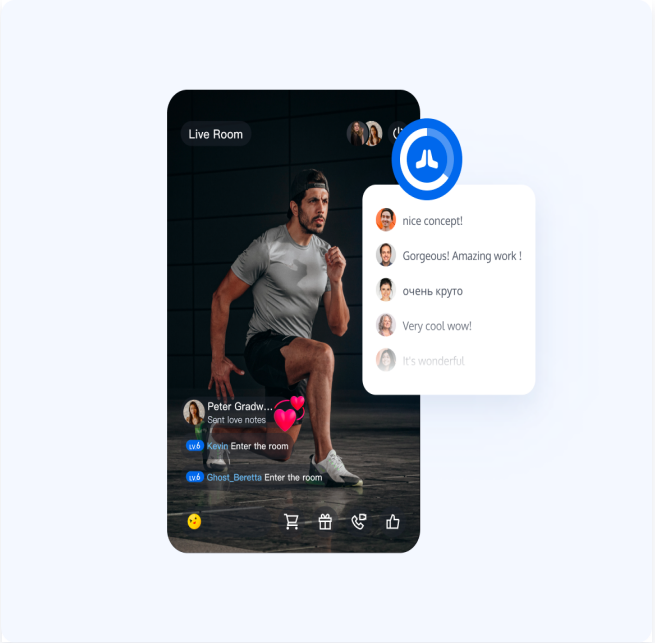
说明：
更多功能与计费说明参见 开通服务 。

适用场景

TUILiveKit 适用于秀场直播、电商直播、赛事直播、新品发布会、路演、在线拍卖等各类高并发大规模直播观看的终端场景。

社交娱乐	游戏互动
	
电商购物	健身交流





在线体验

平台	Android	iOS	桌面端	Flutter
demo 集成	GitHub: Android	GitHub: iOS	GitHub: Electron	GitHub: Flutter

交流与反馈

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

开通服务（TUILiveKit）

最近更新时间：2025-06-26 16:22:22

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

开通体验版

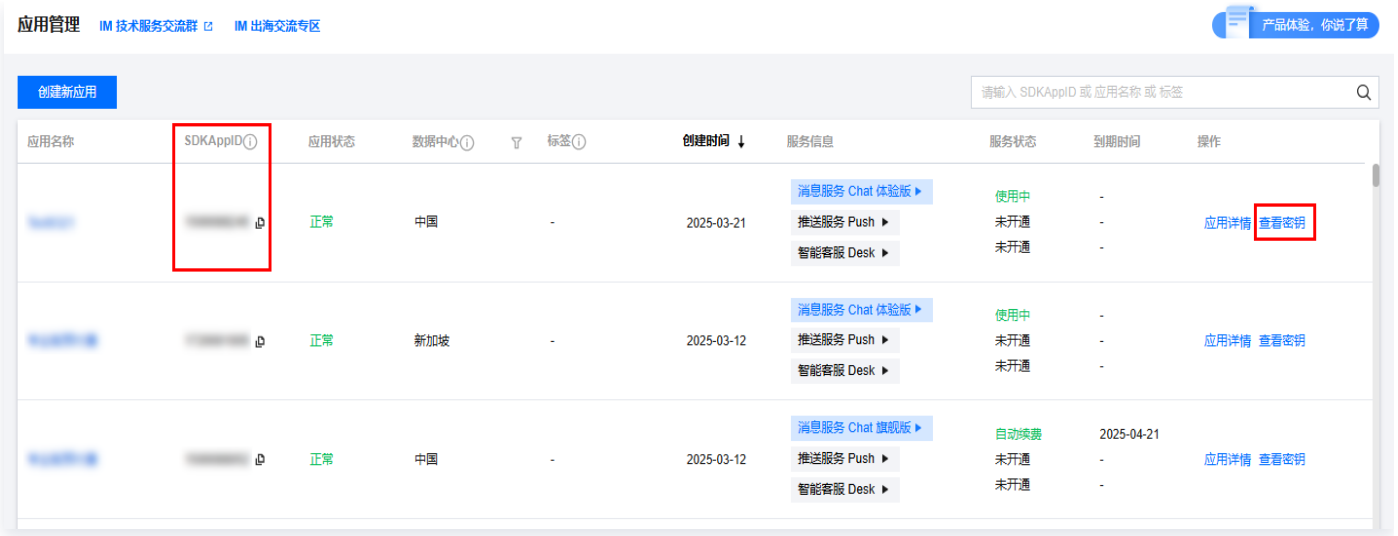
TUILiveKit 是基于腾讯云 即时通信 IM 和 实时音视频 TRTC 两项付费 PaaS 服务构建出的在线直播组件。为了您能更好地体验在线直播功能，我们免费为每个 SDKAppID 提供了在线直播能力7天体验版（体验版不额外赠送通话时长），每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。

1. 登录 即时通信 IM 控制台，单击**创建新应用**，在弹出的对话框中输入您的应用名称，并单击**确定**。

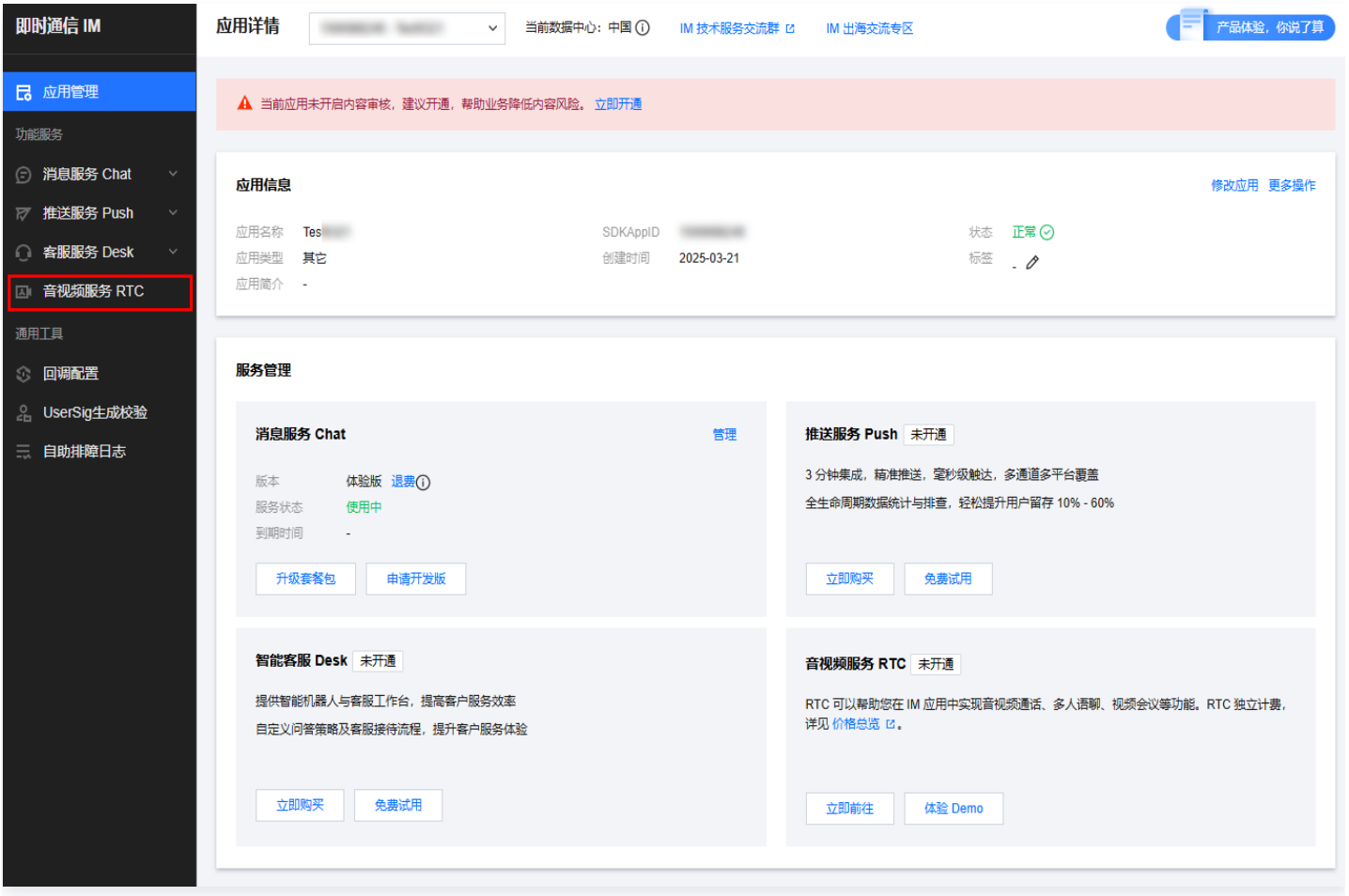
> **说明：**
> Live 目前仅支持选择中国、新加坡和硅谷数据中心，如需使用其他部署区域请 联系我们 。



2. 创建后即可在应用列表查看所有应用，这里的 `SDKAppID` 、**密钥** 会在后续步骤中使用到。



3. 在应用列表单击刚创建的应用，进入应用详情页面，再点击左侧导航栏**音视频服务 RTC**，进入刚创建应用的音视频服务管理页面。



4. 勾选协议，点击**立即开通**，即可开通实时音视频 TRTC 服务。

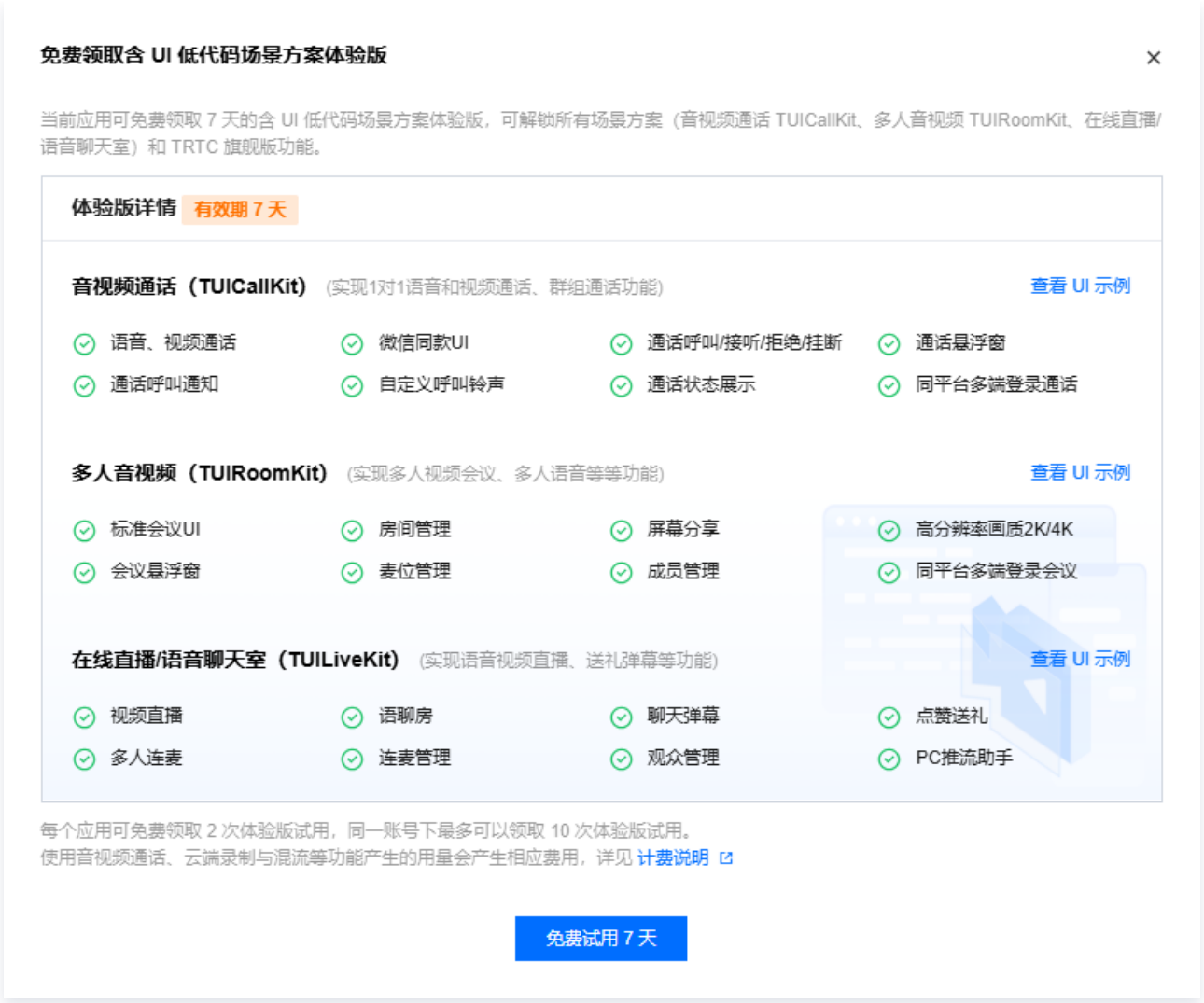


5. **在含 UI 低代码场景方案 > 直播/语聊房（TUILiveKit），单击 免费体验 即可开通 TUILiveKit 的 7 天免费试用服务。**

> **说明：**
> - 领取体验版后，将解锁全部 TRTC 功能。
> - 如果需要**正式应用上线**，可以单击 购买正式版 即可进入购买页面。
> - IM 音视频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。



6. 在弹窗中单击**免费试用7天**，即可成功开通在线直播免费体验版。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。开通完成后即可参见 集成指引 进行集成。



注意：

- 领取体验版后仅开通 TUILiveKit 7天的体验资格，测试过程中所产生的音视频时长等资源消耗，仍会按照实时音视频 TRTC 标准计费规则计费。
- 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。
- 新账号首次可前往 试用中心 免费领取10000分钟音视频时长。
- 如果所选 SDKAppID 体验版领取次数已达上限，需要购买 TUILiveKit 包月套餐才能开通服务，请点击前往 购买页 购买。

7. 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。



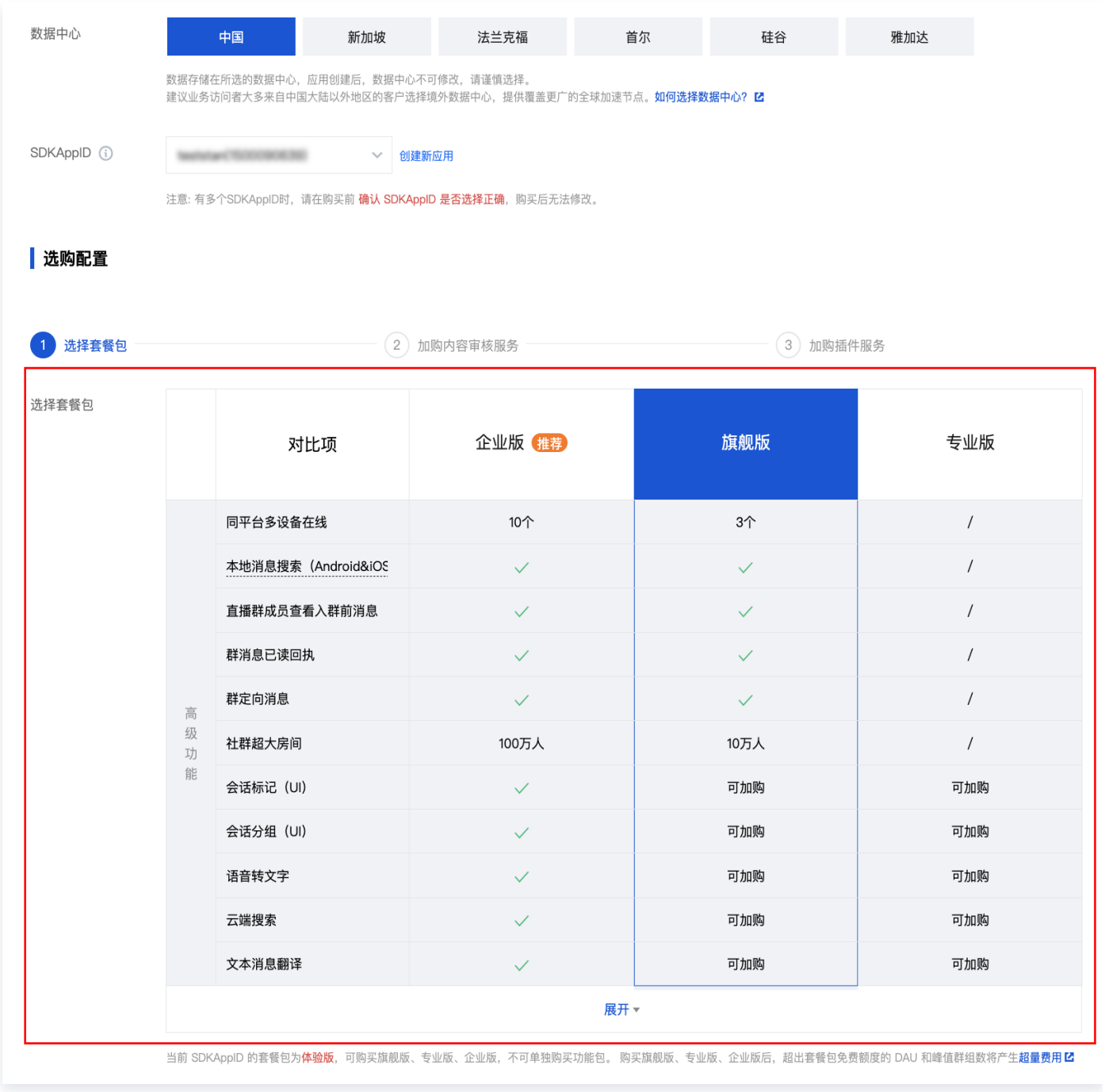
> **注意：**
> 通过**体验续期**延长体验版有效期，将消耗1次免费体验机会。

开通正式版

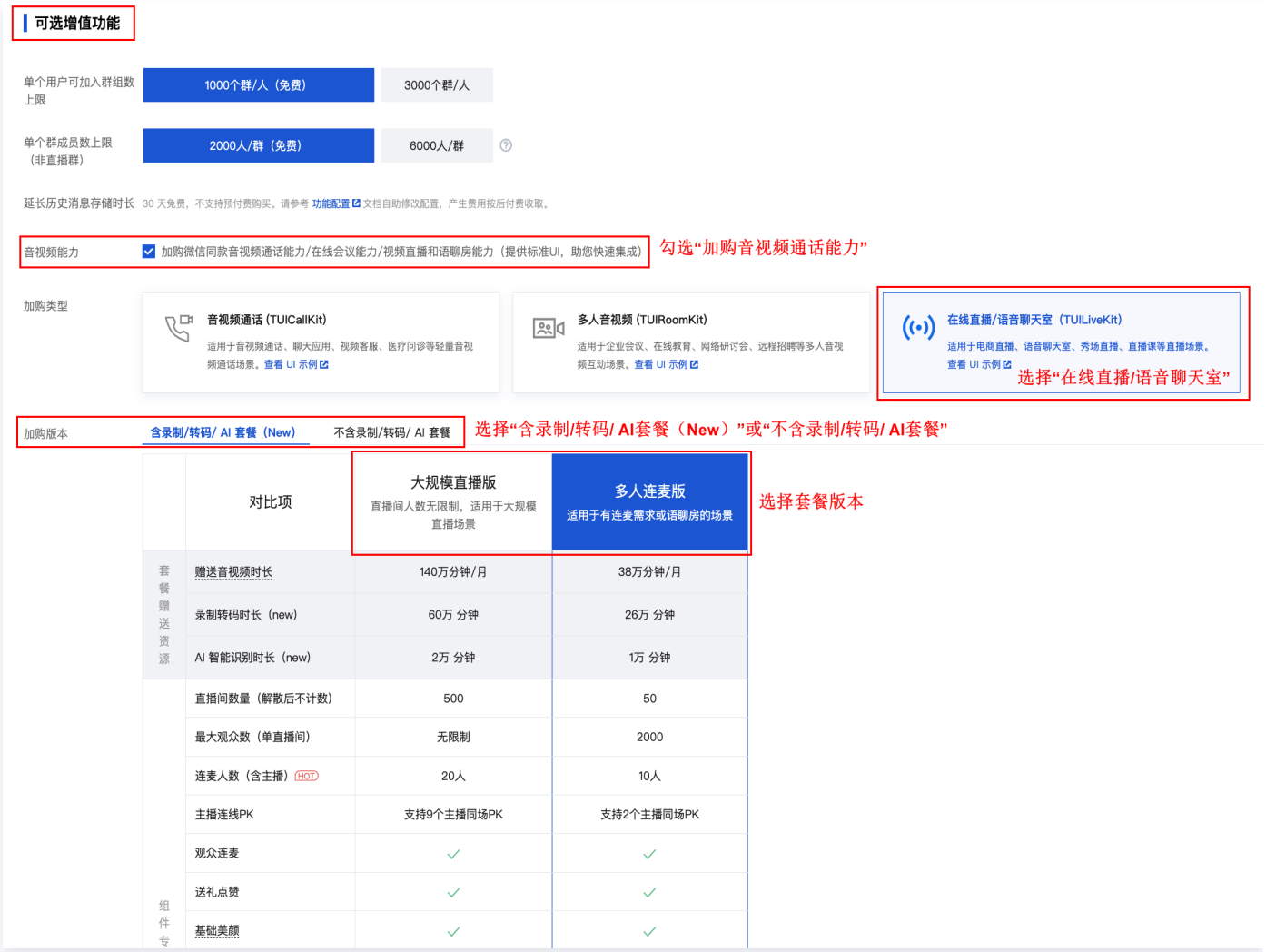
您可参见 TUILiveKit 能力版本计费说明 确认所需要使用的版本，并参见下述内容购买和开通 TUILiveKit 正式版。

1. 登录 即时通信 IM 购买页 。
2. 根据实际业务需求选择数据中心，以及需要购买在线直播能力的 SDKAppID，购买后无法修改。
3. 若您当前 IM 应用已为专业版、旗舰版或企业版，可跳过该步骤；否则您需要先选择您所需要的套餐包，IM 基础套餐包区别可参见 基础服务计费说明 。

> **说明：**
> Live 目前仅支持选择中国、新加坡和硅谷数据中心，如需使用其他部署区域请 联系我们 。



4. 在可选增值功能中勾选**加购音视频通话能力**，选择**在线直播/语音聊天室（TUILiveKit）**类型，并在加购版本中选择所需版本即可。在线直播套餐包能力详情可参见 TUILiveKit 能力版本计费说明 。



5. 购买完成后，您可回到 即时通信 IM 控制台 > 音视频服务 RTC，在含 UI 低代码场景方案 > 直播/语聊房（TUILiveKit）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。



> **注意：**
> - 购买 TUILiveKit 相当于订阅对应版本的 TRTC 包月套餐，若您已经订阅过付费版的 TRTC 包月套餐，仅需购买 IM 付费版后即可。
> - 若您在满足前述付费条件后， TUILiveKit 版本状态显示**待激活**，可参见 激活组件 进行激活。
> - 一个 SDKAppID 只可激活一个低代码组件，若您之前在使用其他组件（TUICallKit 或 TUIRoomKit），可参见 组件切换 变更为TUILiveKit；若您想同时使用多款组件，可参见 如何同时使用多款组件 。

功能与计费说明

TUILiveKit 由实时音视频 TRTC 和即时通信 IM 提供底层技术支持，**您需要购买指定套餐后方可使用 TUILiveKit**，其中免费体验版的开通方式可参见 开通 TUILiveKit 体验版 。
在线直播能力为 IM 增值能力之一，仅当 IM 应用为专业版、旗舰版或企业版时才可购买正式在线直播能力。

不含录制/转码/ AI 套餐

对比项		体验版	单主播版	多人连麦版	大规模直播版
价格		0元/7天	625元/月 立即选购	1875元/月 立即选购	6250元/月 立即选购
开通条件		IM 为任意版本均可开通	IM 为专业版、旗舰版或企业版时可开通		
对应 TRTC 包月套餐		TRTC 基础版	TRTC 基础版	TRTC 尊享版	TRTC 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	100,000个/月	100,000个/月	100,000个/月
	录制转码时长（new）	根据 IM 版本而定	–	–	–
	AI 智能识别时长（new）	根据 IM 版本而定	–	–	–

TUILiveKit 专属能力	直播间数量（解散后不计数）	10	30	50	500
	最大观众数（单直播间）	100	1000	2000	无限制
	单房间连麦人数（含主播）	16人	1人（仅主播）	10人	16人
	跨房连线 / PK	支持9个房间同场 PK	/	支持2个房间同场 PK	支持9个房间同场 PK
	直播预览	✓	/	/	✓
	直播瀑布流	✓	/	/	✓
	关注	✓	/	/	✓
	直播间搜索	✓	/	/	✓
	礼物系统	✓	/	/	✓
	礼物特效	✓	✓	✓	✓
	基础美颜	✓	✓	✓	✓
	观众列表	✓	✓	✓	✓
	PC 推流助手（Electron）	✓	✓（限时体验）	✓（限时体验）	✓
	RTMP 推流/输入在线媒体流	✓	/	✓	✓
	QUIC 稳定传输	✓	/	/	✓
	房间内连麦	✓	/	✓	✓
	AI 降噪	✓	/	✓	✓
	高分辨率画质 2K/4K	✓	/	/	✓
	至臻画质（内测）	/	/	/	✓（申请内测）

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、Flutter、Electron（体验版和大规模直播版支持，单主播版和多人连麦版限时提供）			

含录制/转码/ AI 套餐（New）

对比项		多人连麦版	大规模直播版
价格		2875元/月 立即选购	8000元/月 立即选购
开通条件		IM 为专业版、旗舰版或企业版时可开通	
对应 TRTC 包月套餐		尊享版 Plus	旗舰版 Plus
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	100,000个/月	100,000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
TUILiveKit 专属能力	直播间数量（解散后不计数）	50	500
	最大观众数（单直播间）	2000	无限制

版权所有：腾讯云计算（北京）有限责任公司

	单房间连麦人数（含主播）	10人	20人
	跨房连线/PK	支持10个房间同场 PK	支持20个房间同场 PK
	直播预览	/	✓
	直播瀑布流	/	✓
	关注	/	✓
	直播间搜索	/	✓
	礼物系统	/	✓
	礼物特效	✓	✓
	基础美颜	✓	✓
	观众列表	✓	✓
	PC 推流助手（Electron）	✓（限时体验）	✓
	RTMP 推流/输入在线媒体流	✓	✓
	QUIC 弱网稳定传输	/	✓
	房间内连麦	✓	✓
	AI 降噪	✓	✓
	高分辨率画质 2K/4K	/	✓
	至臻画质（内测）	/	✓（申请内测）
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、Flutter、Electron（体验版和大规模直播版支持，单主播版和多人连麦版限时提供）

说明：

1. **开通条件**：TUILiveKit 为 IM 和 TRTC 联合提供的服务，因此在开通 TUILiveKit 需首先保证应用已有对应版本的 IM 基础套餐包。可了解更多详情请 单击购买 。
2. **对应 TRTC 包月套餐**：TUILiveKit 为 IM 和 TRTC 联合提供的服务，同时 TUILiveKit 为 TRTC 包月套餐提供的功能项之一，因此购买 TUILiveKit 相当于订阅对应版本 TRTC 包月套餐，**仅收取一次费用**，更多 TRTC 包月套餐功能及版本说明可参见 TRTC 版本功能 。 示例：您在 IM 为专业版/旗舰版或企业版的情况下，购买了上表中价格为 9249元/月的**大规模直播版**的版本，则：您订阅了 TRTC 旗舰版包月套餐，同时开通了 TUILiveKit 大规模直播版。
3. **赠送音视频时长**：购买 TUILiveKit 相当于订阅对应版本 TRTC 包月套餐， TRTC 包月套餐将赠送音视频时长用量，有效期一个月，仅可用于抵扣订阅套餐服务的应用（SDKAppId）产生的实时音视频音视频时长用量， 超额用量计费说明请参见 音视频时长计费说明 。

计费说明

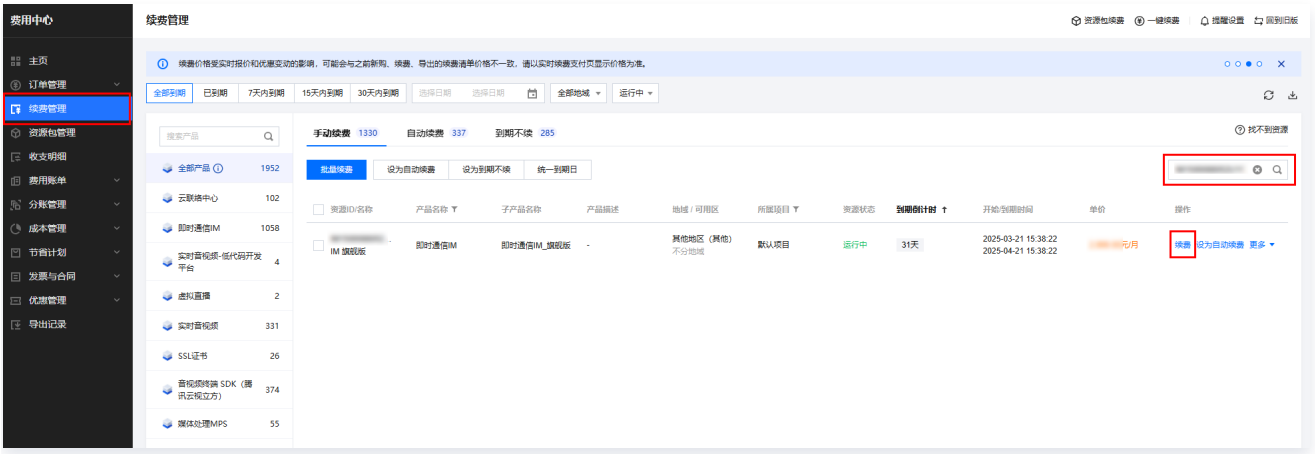
- 在线直播能力 TUILiveKit 由即时通信 IM 与实时音视频 TRTC 两个底层 PaaS 服务提供支持，**使用期间请务必保证您的 IM 应用和 TRTC 应用处于可用状态，否则在线直播相关功能将无法正常使用**。
- 在线直播能力购买后立即生效，体验版默认7天，付费版默认1个月。
- 在线直播能力暂不支持版本升降级。若您需要使用其他版本的在线直播能力，可在当前版本到期后重新购买，或在满足 退款规则 的前提下，退还当前版本在线直播能力，重新购买新的版本。
- 在线直播能力不支持更换 SDKAppId，您在选购时请务必确认您需要购买的 SDKAppId。

续费正式版

在线直播能力为 IM 增值服务之一，在进行续费时同样需要保证 IM 基础套餐包为专业版、旗舰版或企业版，因此续费时建议您同时对 IM 套餐包和在线直播能力进行续费，若您仅对在线直播能力进行续费，IM 套餐包到期后 IM 应用停用将导致在线直播能力无法正常使用。

具体操作如下：

1. 登录您的腾讯云 费用中心 ，在左侧导航中，选择 续费管理 。
2. 分别为 IM 套餐包和音视频通话能力进行续费：
   - 右侧搜索框中搜索“IM + SDKAppID”，续费即时通信 IM 套餐包。
   - 右侧搜索框中搜索“TRTC + SDKAppID”，续费音视频通话能力。



3. 进入续费界面，选择续费时长，单击**下一步**。套餐包价格说明请参见 **价格总览** 。

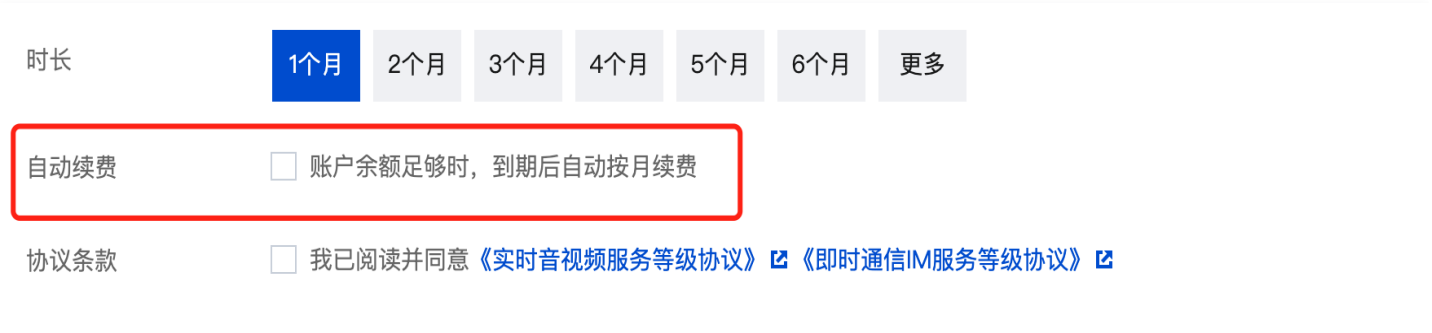


4. 在购买确认界面确认商品信息，**提交订单**并完成支付。



开启自动续期

由于 TUILiveKit 由实时音视频 TRTC 与即时通信 IM 联合提供服务，开启 TRTC 包月套餐和 IM 套餐包的自动续期后，TUILiveKit 的自动续期即可开启。您可以在购买页内下单时即勾选开启自动续期。



若您已购买完成，可通过**控制台管理自动续费**和**费用中心管理自动续费** 2种方式管理自动续费，具体操作如下：

控制台管理自动续费

登录 即时通信 IM 控制台 > 音视频服务 RTC，在**含 UI 低代码场景方案 > 直播/语聊房（TUILiveKit）**功能区点击**开通自动续费**。

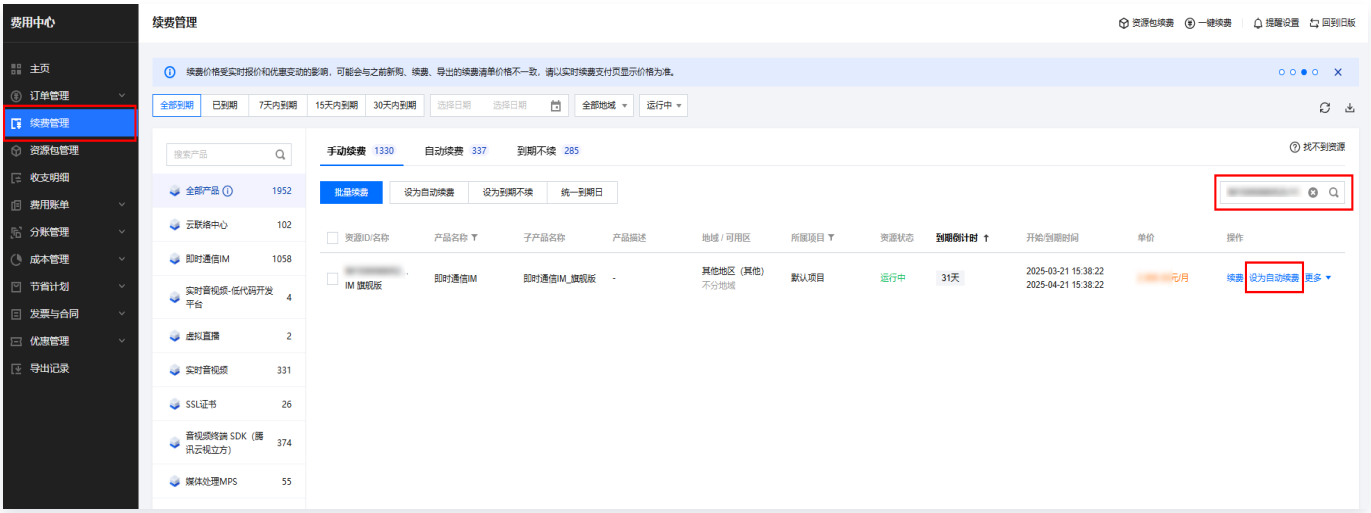


费用中心管理自动续费

1. 登录您的腾讯云 **费用中心**，在左侧导航中，选择 **续费管理**。
2. 将 TRTC 包月套餐和 IM 套餐包设置为自动续费：
   - 右侧搜索框中搜索“IM + SDKAppID”，将即时通信 IM 套餐包设为自动续费。
   - 产品类别选择实时音视频，右侧搜索框中搜索“TRTC + SDKAppID”，将 TRTC 包月套餐设为自动续费。



   - 产品类别选择即时通信，右侧搜索框中搜索“IM + SDKAppID”，将即时通信 IM 套餐包设为自动续费。



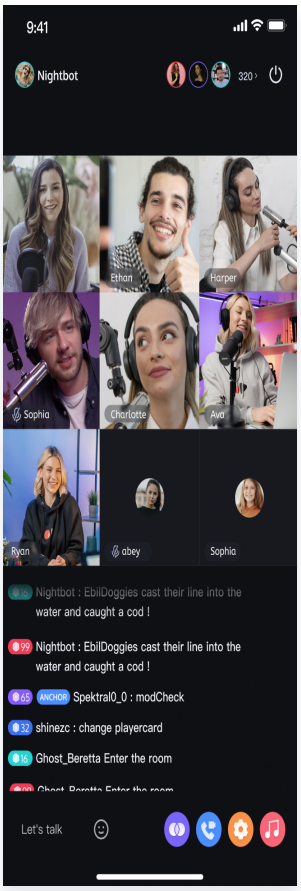
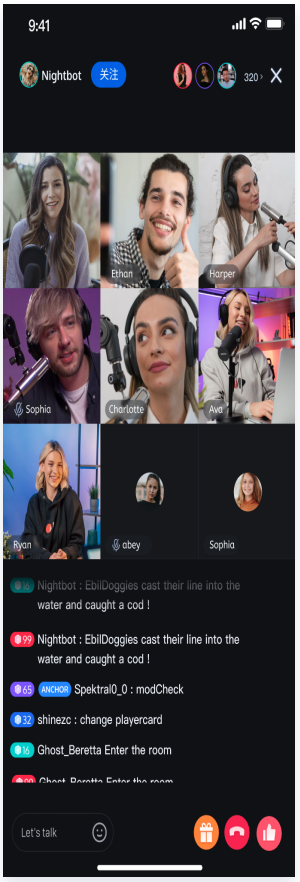
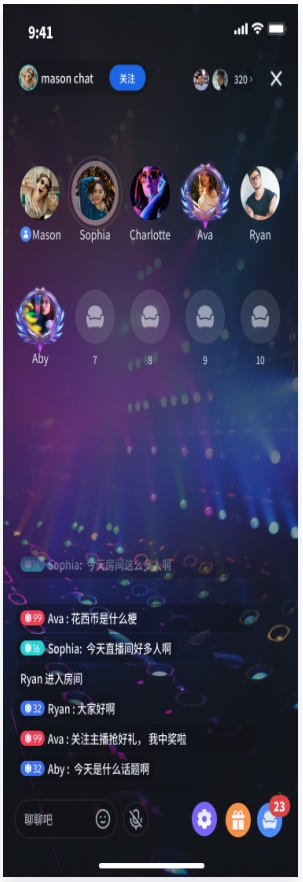
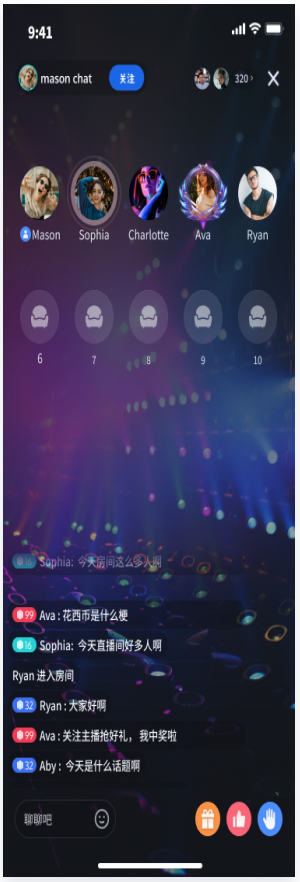
3. 操作完成您可登录 即时通信 IM 控制台，查看自动续期开启状态。



跑通 Demo（TUILiveKit）Android

最近更新时间：2025-04-14 10:13:52

本文将为您介绍如何快速跑通视频直播/语音聊天室 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 界面的视频直播/语音聊天室 功能。

主播（视频直播）	观众（视频直播）	主播（语音直播）	观众（语音直播）
			

环境准备

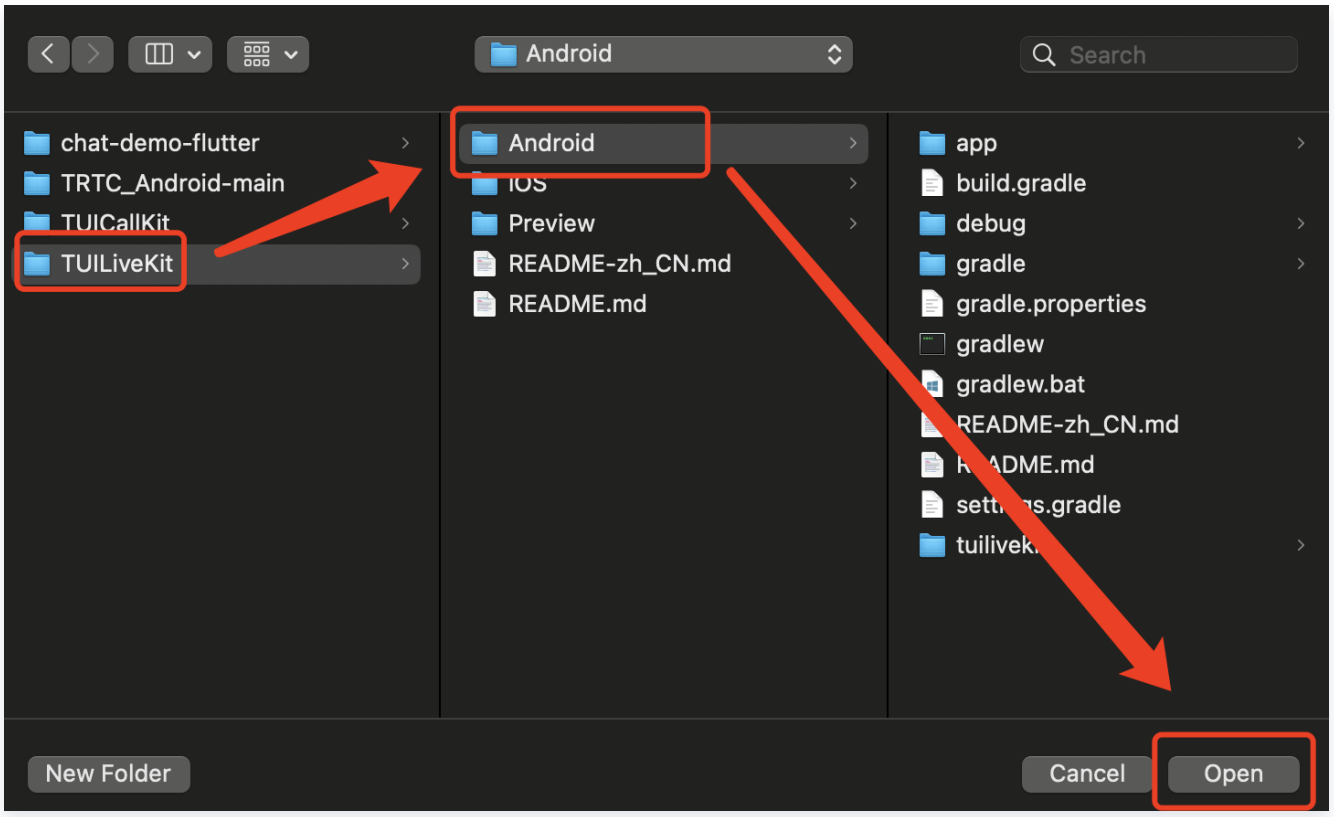
- Android 5.0 (SDK API level 21)及以上。
- Gradle 8.0 及以上。
- 两台 Android 5.0 以上的设备。

步骤一：下载 Demo

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

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

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

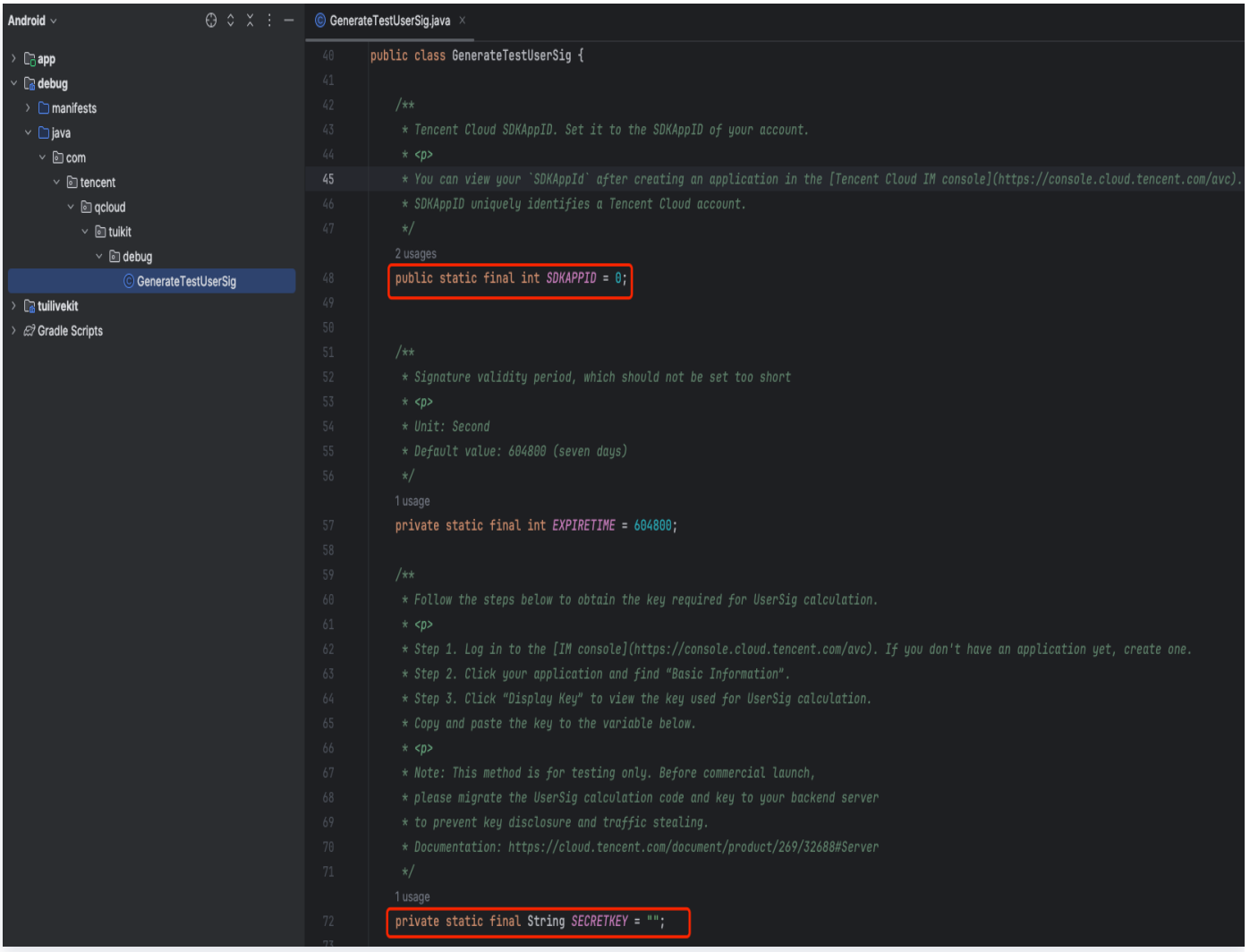


步骤二：配置 Demo

1. 请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。成功后即可获得14天的体验版，可免费体验 TUILiveKit 的全部功能，并获取 SDKAppID 和 SDKSecretKey 。
2. 打开

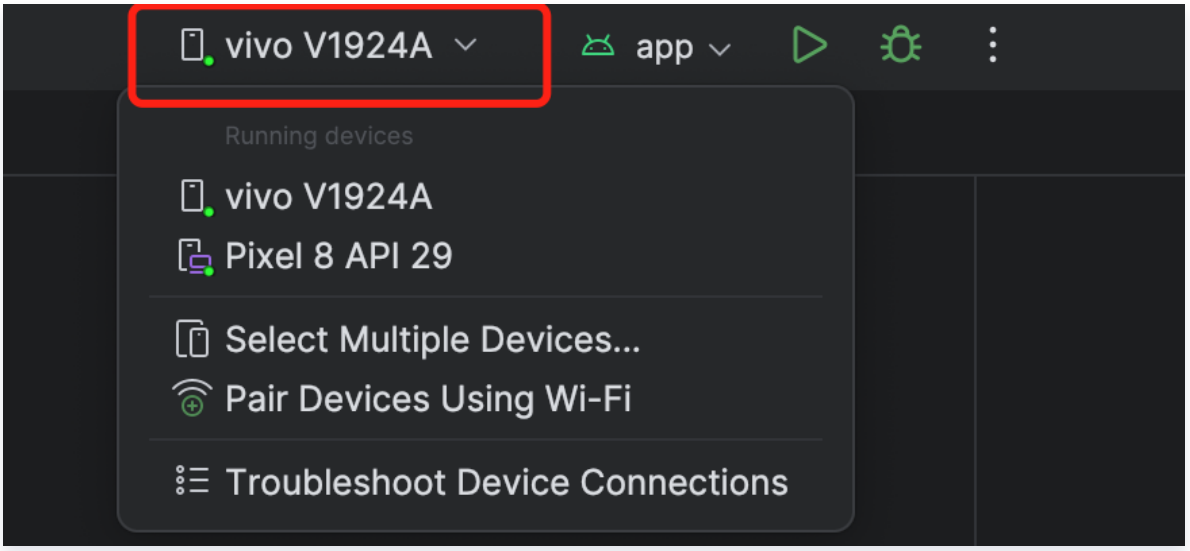
```
Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java
```

文件，将 提交申请 时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

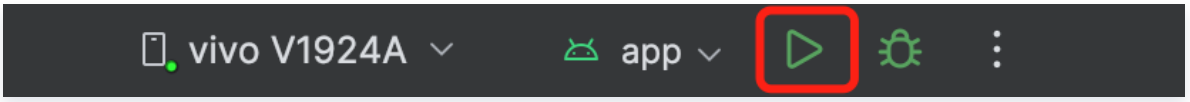


步骤三：跑通 Demo

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 TUILiveKit 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直播和观看直播。

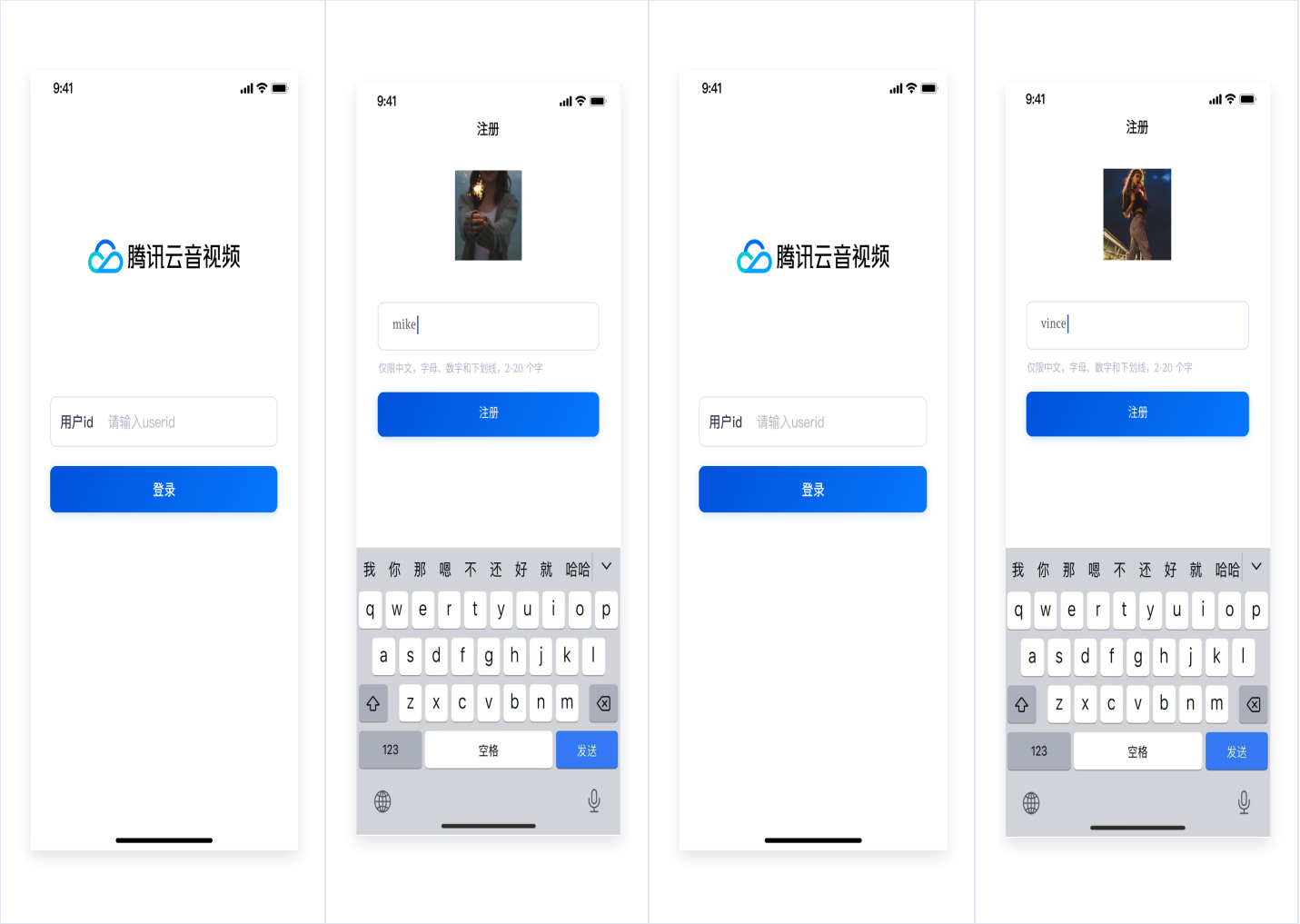
步骤四：开始直播

> **注意：**
> 为了使您可以体验完整的视频直播流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主播，一方作为观众。

1. 登录 & 注册

请您在 `用户ID` 处输入的 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。

主播端（mike）	观众端（vince）



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. 主播开启直播

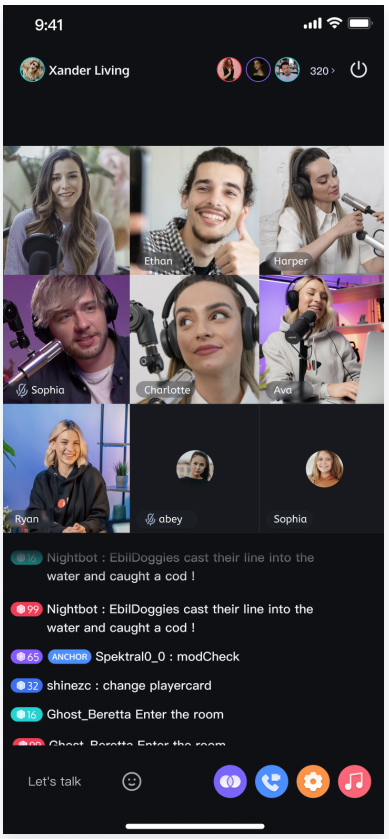
点击主页底部中间的**开播**按钮，即可进入开播预览页面，再点击**开始直播**即可开始直播了。

视频直播：

主播（进房前）	主播（预览）	主播（直播中）

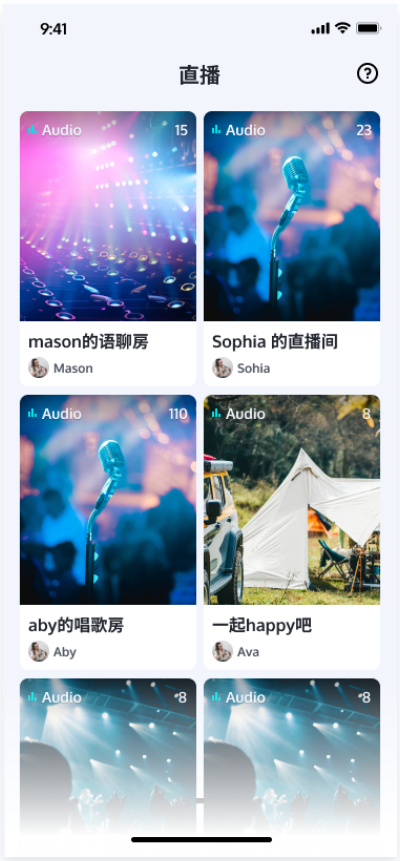


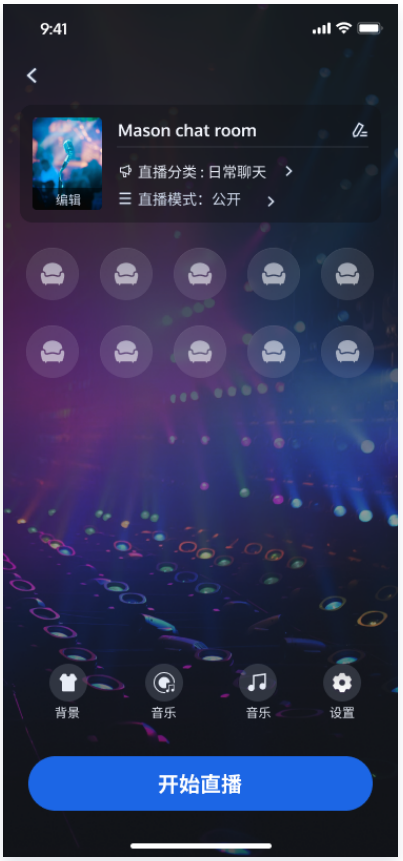


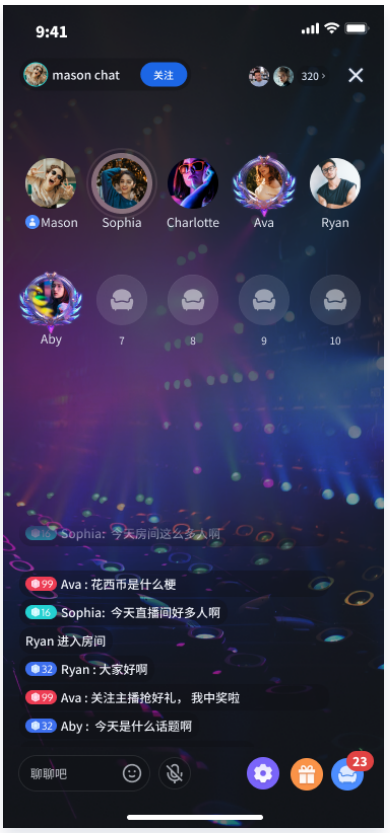


语音直播：

主播（预览）	主播（预览）	主播（直播中）



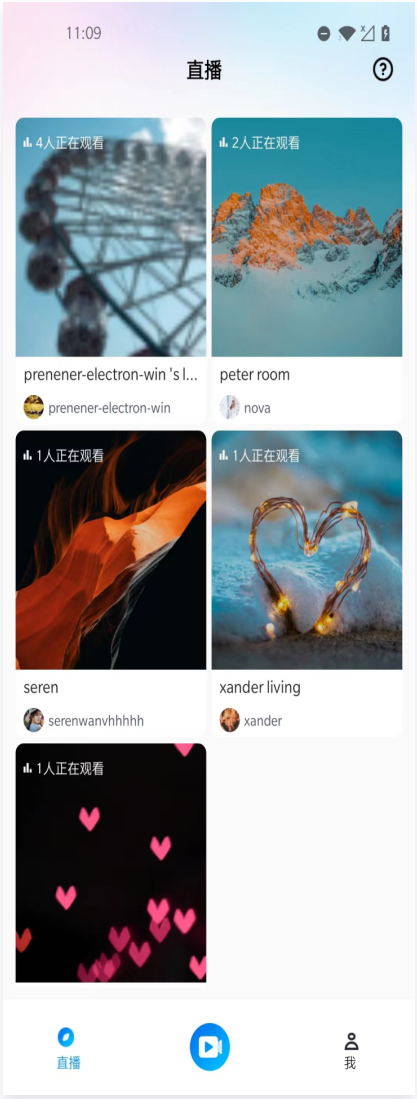


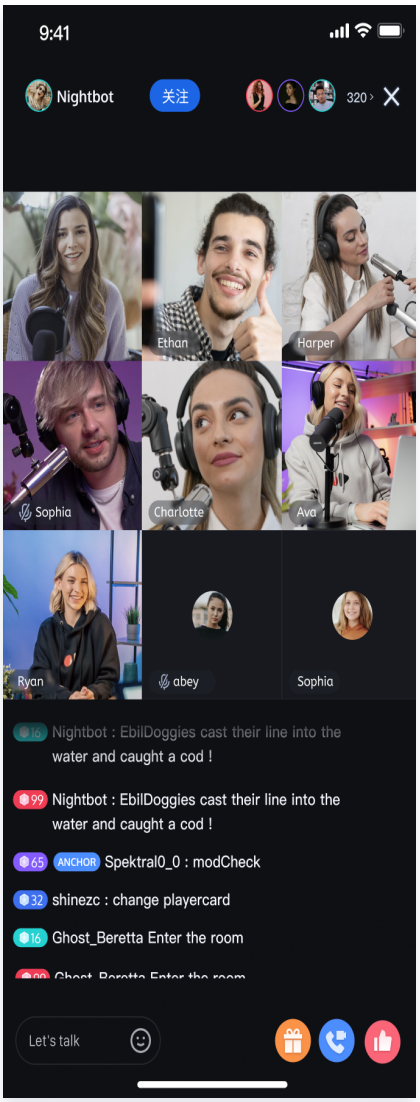


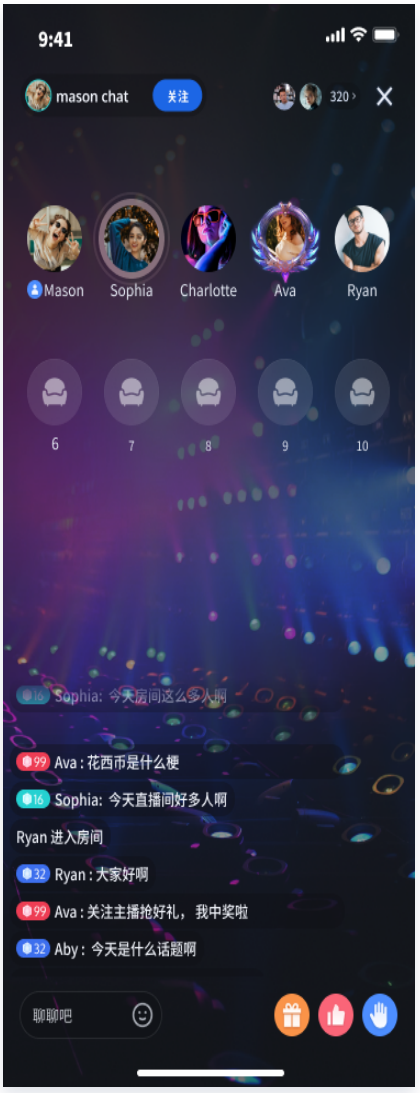
3. 观众加入直播间

点击直播列表里任意一个房间，即可进入直播间。

观众（进房前）	观众（进房后 视频直播）	观众（进房后 语音直播）


11:09
直播
4人正在观看
2人正在观看
prenener-electron-win 's l...
prenener-electron-win
peter room
nova
1人正在观看
1人正在观看
seren
serenwanvhhhhh
xander living
xander
1人正在观看
直播
我

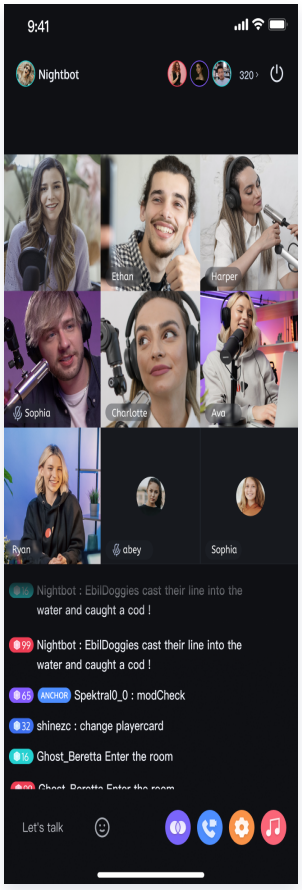
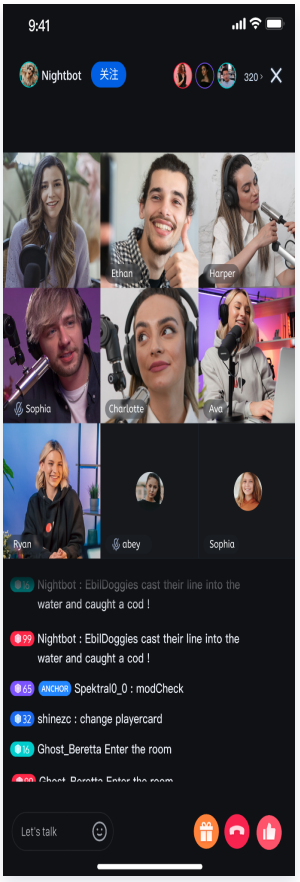
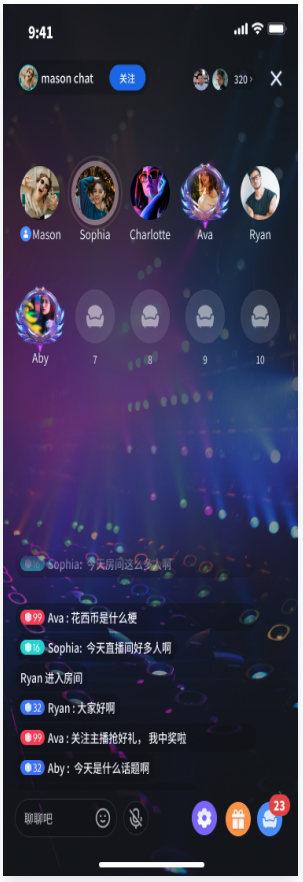
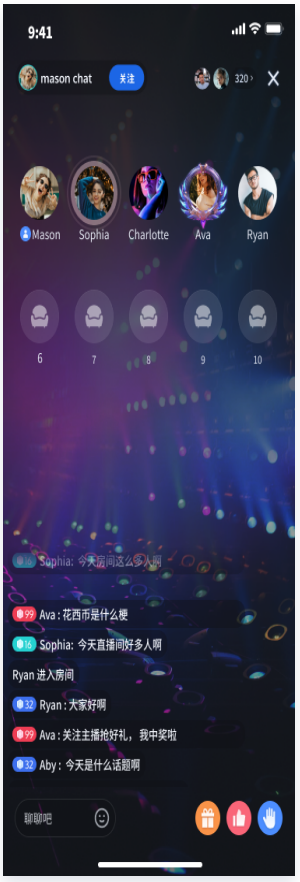

9:41
Nightbot
关注
320
Ethan
Harper
Sophia
Charlotte
Ava
Ryan
abey
Sophia
Nightbot : EbilDoggies cast their line into the water and caught a cod !
Nightbot : EbilDoggies cast their line into the water and caught a cod !
ANCHOR Spektral0_0 : modCheck
shinezc : change playercard
Ghost_Beretta Enter the room
Let's talk


9:41
mason chat
关注
320
Mason
Sophia
Charlotte
Ava
Ryan
6
7
8
9
10
Sophia: 今天房间这么多人啊
Ava : 花西币是什么梗
Sophia: 今天直播间好多人啊
Ryan 进入房间
Ryan : 大家好啊
Ava : 关注主播抢好礼，我中奖啦
Aby : 今天是什么话题啊
聊聊吧

iOS

最近更新时间：2025-04-14 10:13:52

本文将为您介绍如何快速跑通视频直播/语音聊天室 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 界面的视频直播/语音聊天室功能。

主播（视频直播）	观众（视频直播）	主播（语音聊天室）	观众（语音聊天室）
			

环境准备

- Xcode 15 及以上。
- iOS 13.0 及以上。
- CocoaPods 环境安装，点击查看 。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

步骤一：下载 Demo

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

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

2. 在命令行中进入 iOS 项目目录：

```
cd TUILiveKit/iOS/Example
```

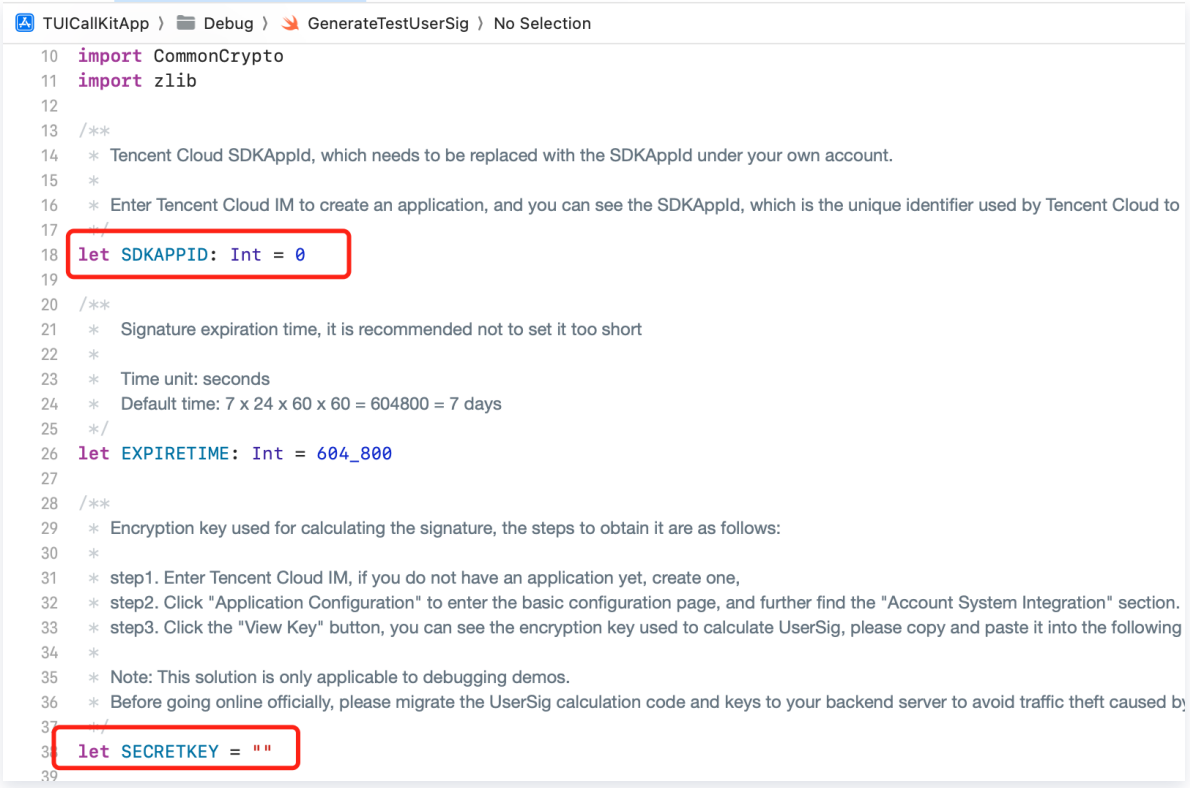
3. 加载依赖库：

```
pod install
```

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

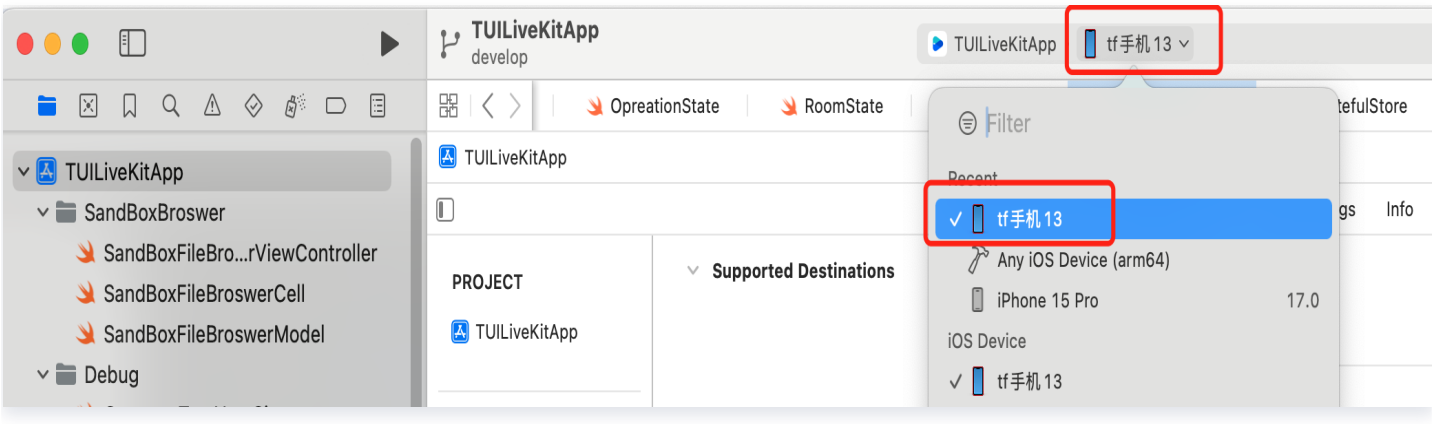
步骤二：配置 Demo

1. 请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。成功后即可获得14天的体验版，可免费体验 TUILiveKit 的全部功能，并获取 SDKAppID 和 SDKSecretKey 。
2. 打开 `/iOS/Example/Debug/GenerateTestUserSig.swift` 文件，将 提交申请 时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：



步骤三：跑通 Demo

打开 `TUILiveKit/iOS/Example/TUILiveKitApp.xcworkspace` 工程， 如下图所示在 XCode 中选择您要将运行的设备：



1. 选择完成后，点击运行将我们的 Demo 运行到目标设备上。
2. 当 Demo 成功运行到设备上后，您可以通过如下步骤来发起直播和观看直播。

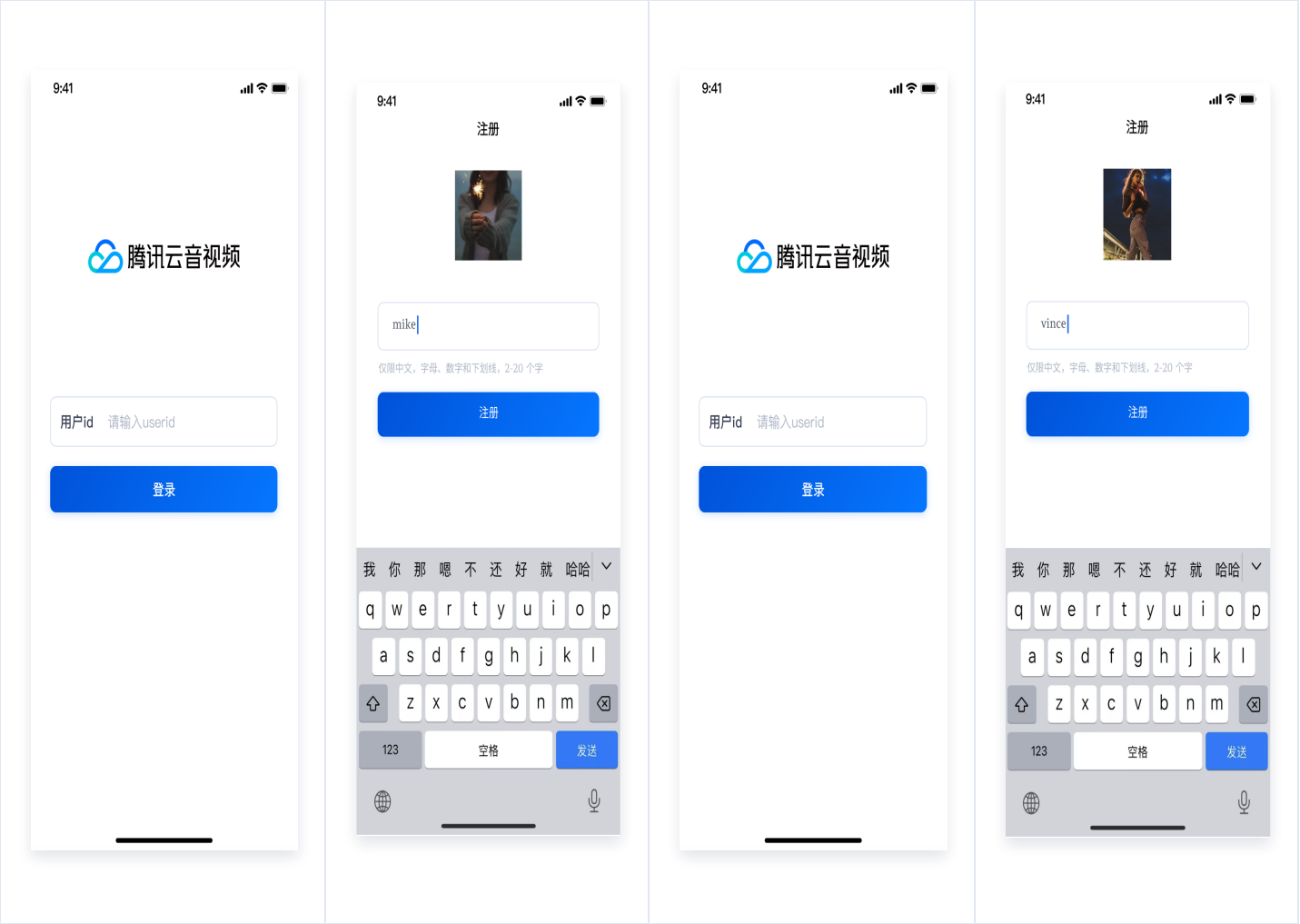
步骤四：开始直播

> **注意：**
> 为了使您可以体验完整的视频直播流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主播，一方作为观众。

1. 登录 & 注册

请您在 `用户ID` 处输入的 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。

主播端（mike）	观众端（vince）



注意：

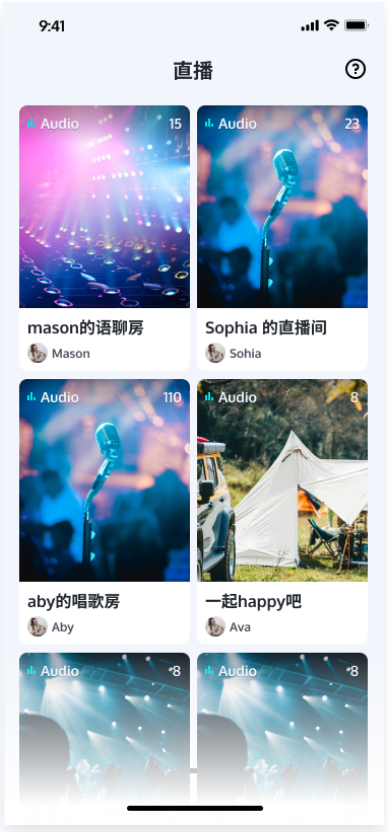
尽量避免使您的 UserID 被设置成“1”、“123”、“111”等简单字符串，由于 TRTC 不支持同一个 UserID 多端登录，所以在多人协作开发时，形如 “1”、“123”、“111” 这样的 UserID 很容易被您的同事占用，导致登录失败，因此我们建议您在调试的时候设置一些辨识度高的 UserID。

2. 主播开启直播

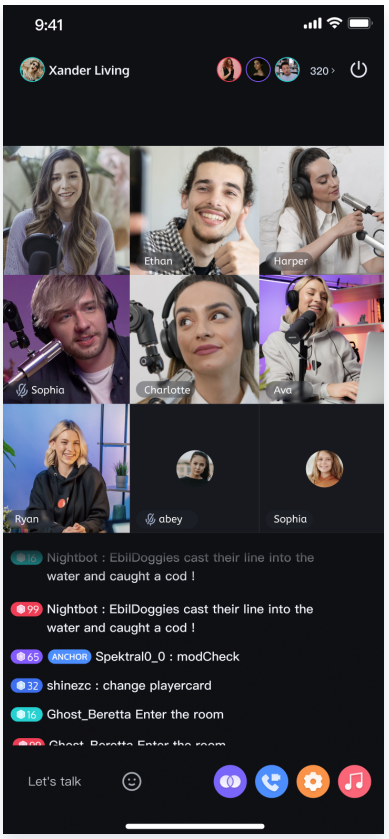
点击主页底部中间的**开播**按钮，即可进入开播预览页面，再点击**开始直播**即可开始直播了。

视频直播：

主播（进房前）	主播（预览）	主播（直播中）

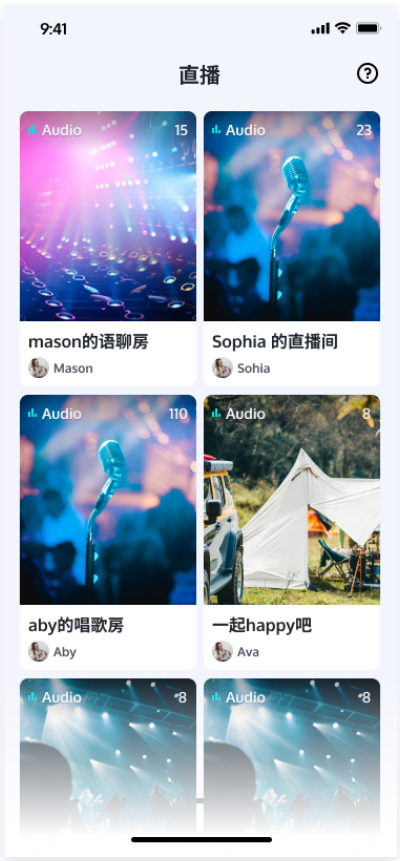


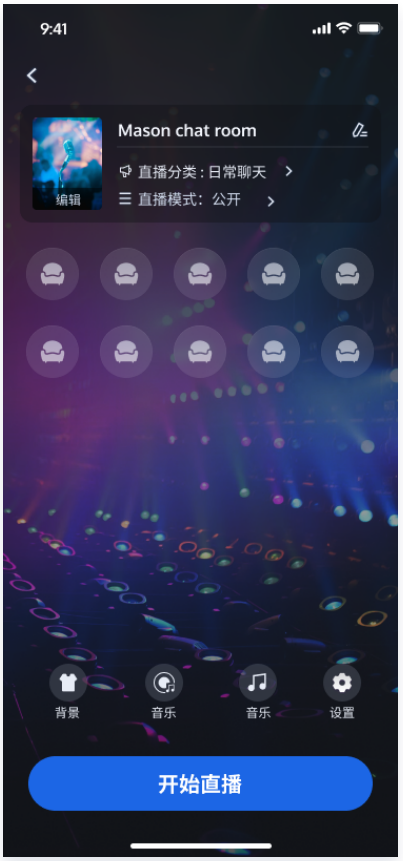


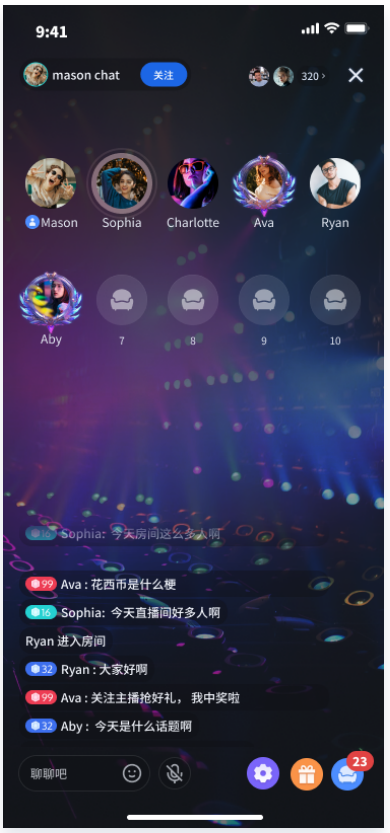


语音聊天室：

主播（预览）	主播（预览）	主播（直播中）



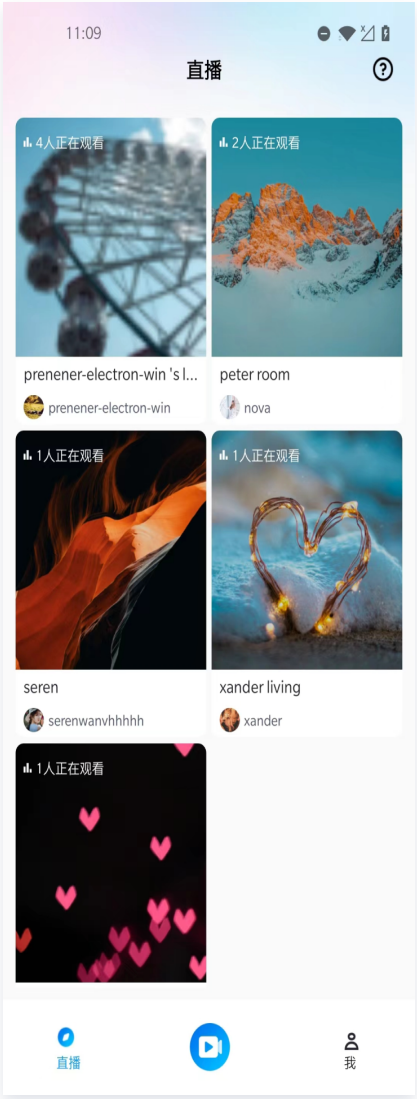


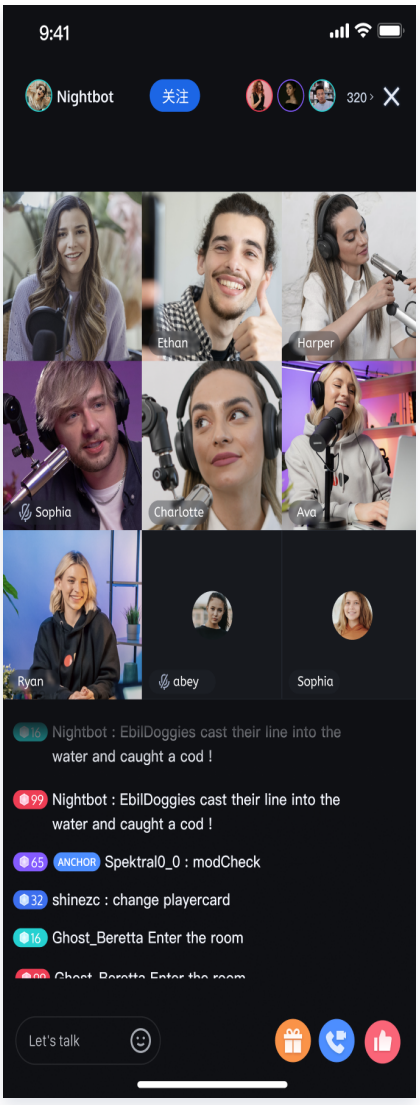


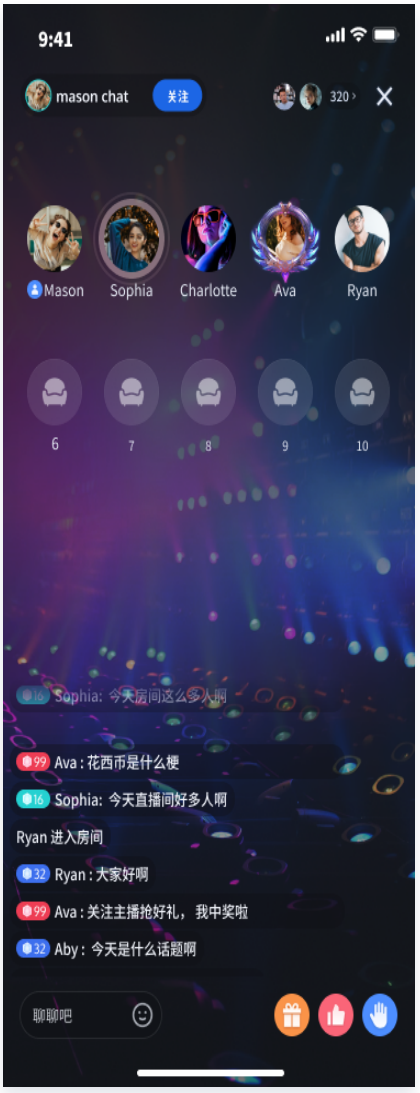
3. 观众加入直播间

点击直播列表里任意一个房间，即可进入直播间。

观众（进房前）	观众（进房后 视频直播）	观众（进房后 语音聊天室）


11:09
直播
4人正在观看
2人正在观看
prenener-electron-win 's l...
prenener-electron-win
peter room
nova
1人正在观看
1人正在观看
seren
serenwanvhhhhh
xander living
xander
1人正在观看
直播
我

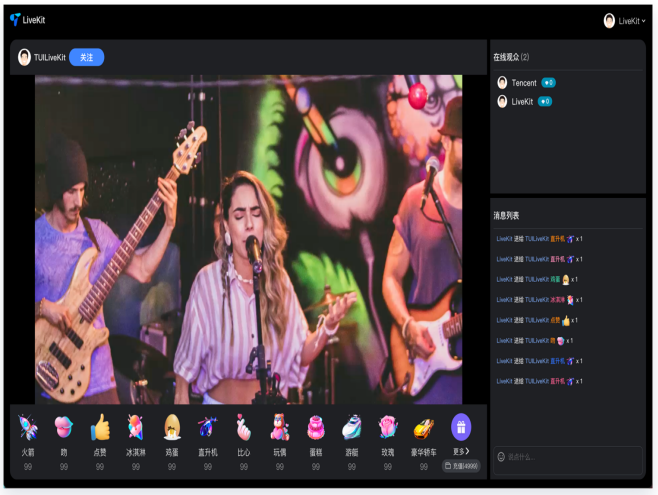

9:41
Nightbot
关注
320
Ethan
Harper
Sophia
Charlotte
Ava
Ryan
abey
Sophia
Nightbot : EbilDoggies cast their line into the water and caught a cod !
Nightbot : EbilDoggies cast their line into the water and caught a cod !
ANCHOR Spektral0_0 : modCheck
shinezc : change playercard
Ghost_Beretta Enter the room
Let's talk


9:41
mason chat
关注
320
Mason
Sophia
Charlotte
Ava
Ryan
6
7
8
9
10
Sophia: 今天房间这么多人啊
Ava : 花西币是什么梗
Sophia: 今天直播间好多人啊
Ryan 进入房间
Ryan : 大家好啊
Ava : 关注主播抢好礼，我中奖啦
Aby : 今天是什么话题啊
聊聊吧

Web

最近更新时间：2025-04-14 18:02:42

本文档主要介绍如何快速跑通 TUILiveKit 示例工程，体验高质量在线直播。跟随本文档，您可以在 10 分钟内跑通 Demo，并开启一场属于您自己的直播。





环境准备

- Node.js 版本：Node.js ⩾ 16.19.1（推荐使用官方 LTS 版本，npm 版本请与 node 版本匹配）。
- 现代浏览器，支持 WebRTC APIs 。

下载 Demo

1. 打开终端复制输入示例命令克隆 仓库 。

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

2. 安装依赖

```
cd TUILiveKit/Web/web-vite-vue3
```

```
npm install
```

配置 Demo

1. 激活 TUILiveKit 服务 。获取 SDKAppID 和 SDKSecretKey。



2. 打开 TUILiveKit/Web/web-vite-vue3/src/config/basic-info-config.js 文件，并输入激活服务时获得的 SDKAppID 和 SDKSecretKey：

```
/*
 * @Description: Basic information configuration for TUIRoomKit applications
 */

import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min';

/**
 * Tencent Cloud SDKAppId, which should be replaced with user's SDKAppId.
 * Enter Tencent Cloud TRTC [Console] (https://console.cloud.tencent.com/trtc ) to create an application,
 * and you will see the SDKAppId.
 * It is a unique identifier used by Tencent Cloud to identify users.
 *
 */

export const SDKAPPID = 0;

/**
 * Encryption key for calculating signature, which can be obtained in the following steps:
 *
 * Step1. Enter Tencent Cloud TRTC [Console](https://console.cloud.tencent.com/rav ),
 * and create an application if you don't have one.
 * Step2. Click your application to find "Quick Start".
 * Step3. Click "View Secret Key" to see the encryption key for calculating UserSig,
 * and copy it to the following variable.
 *
 * Notes: this method is only applicable for debugging Demo. Before official launch,
 * please migrate the UserSig calculation code and key to your backend server to avoid
 * unauthorized traffic use caused by the leakage of encryption key.
 * Document: https://intl.cloud.tencent.com/document/product/647/35166#Server
 *
 */
export const SDKSECRETKEY = '';
```

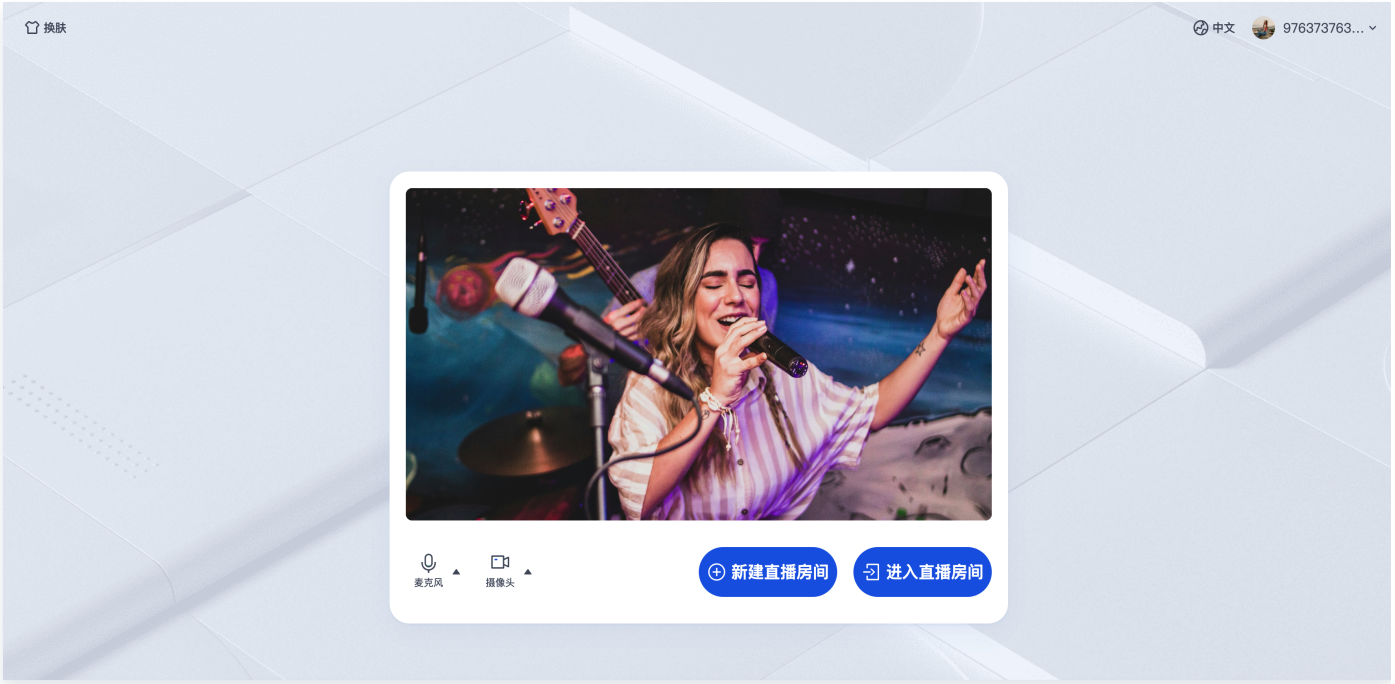
跑通 Demo

1. 在终端中输入命令，运行 Demo。

```
#cd TUILiveKit/Web/web-vite-vue3
npm run dev
```

> **注意：**
> 本地环境请在 localhost 协议下访问，具体参见 网络访问协议说明 。

2. 自动打开浏览器窗口。

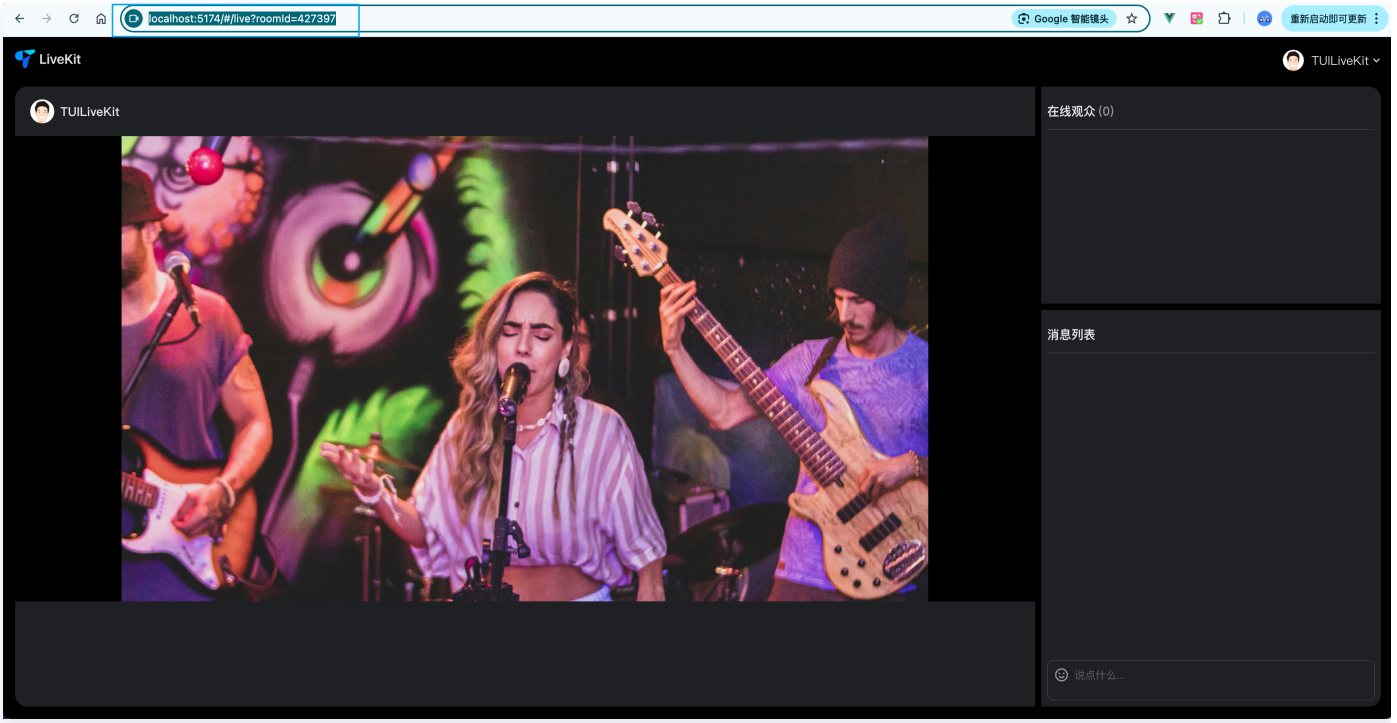


3. 点击**新建直播房间**即可开启您的第一场直播啦。

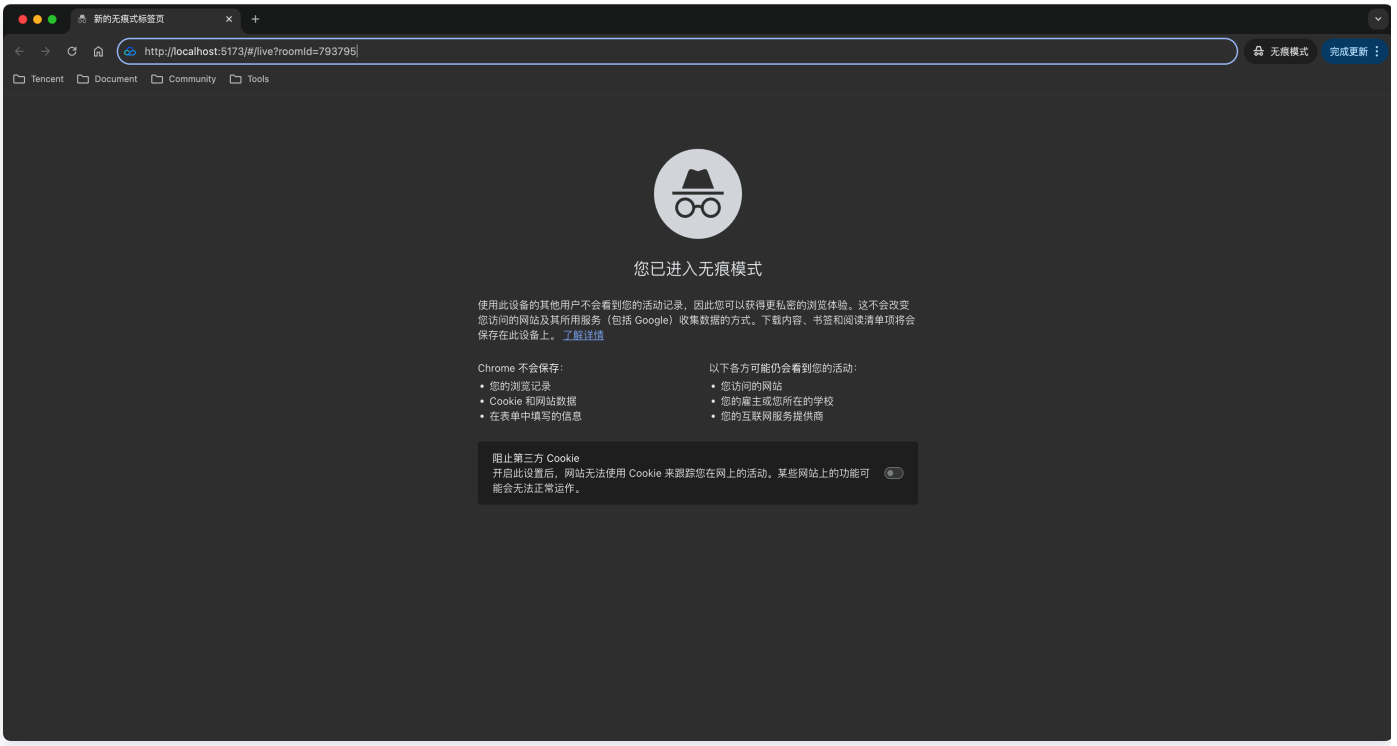


4. 复制导航栏的 URL 地址获取房间链接，观众可在浏览器输入链接进入直播间。

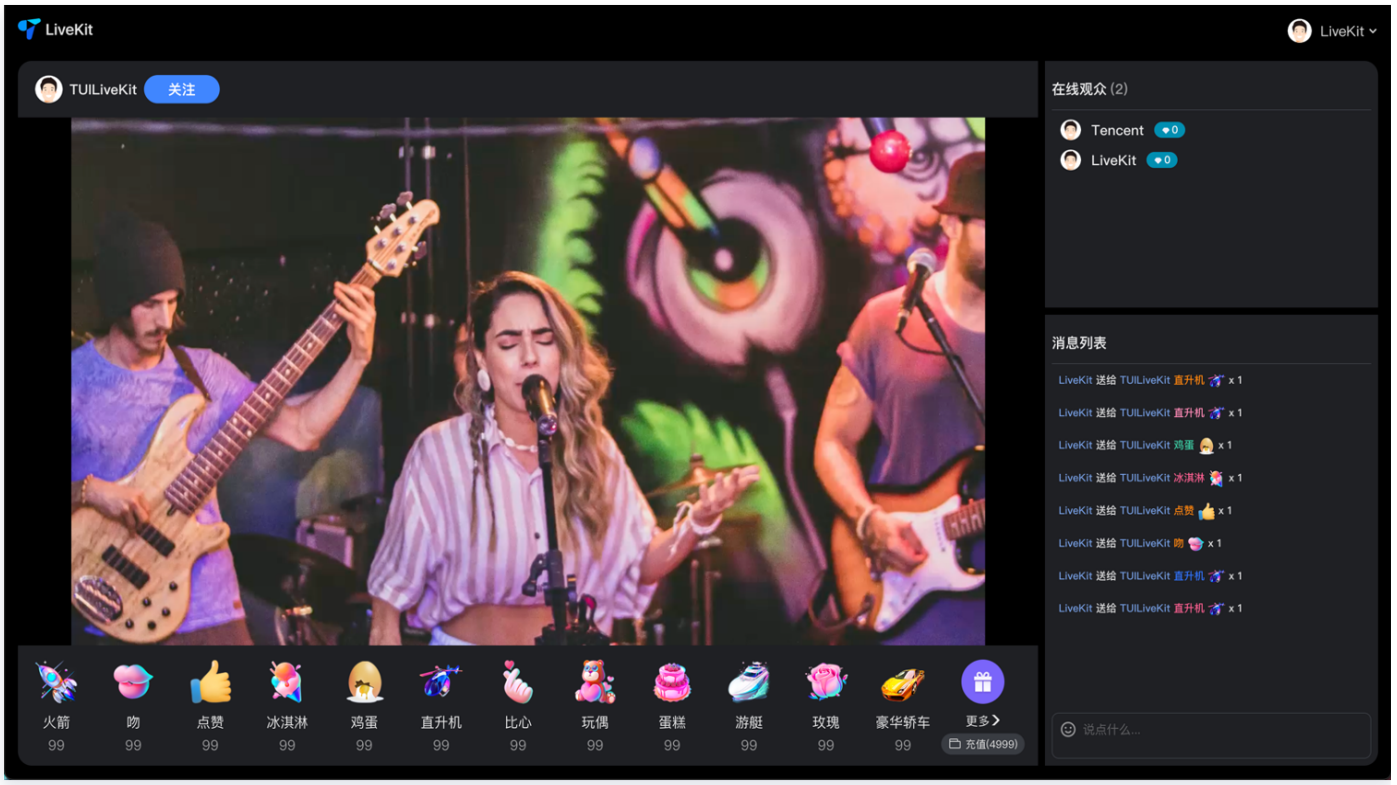
版权所有：腾讯云计算（北京）有限责任公司



5. 观众在浏览器输入链接，如果没有登录信息会跳转至登录页。



6. 登录成功后会自动跳转至直播页。



其他文档

- TUILiveKit 组件介绍
- TUILiveKit 开通服务
- TUILiveKit Web 快速接入
- 客户端 API 文档
- 常见问题

交流与反馈

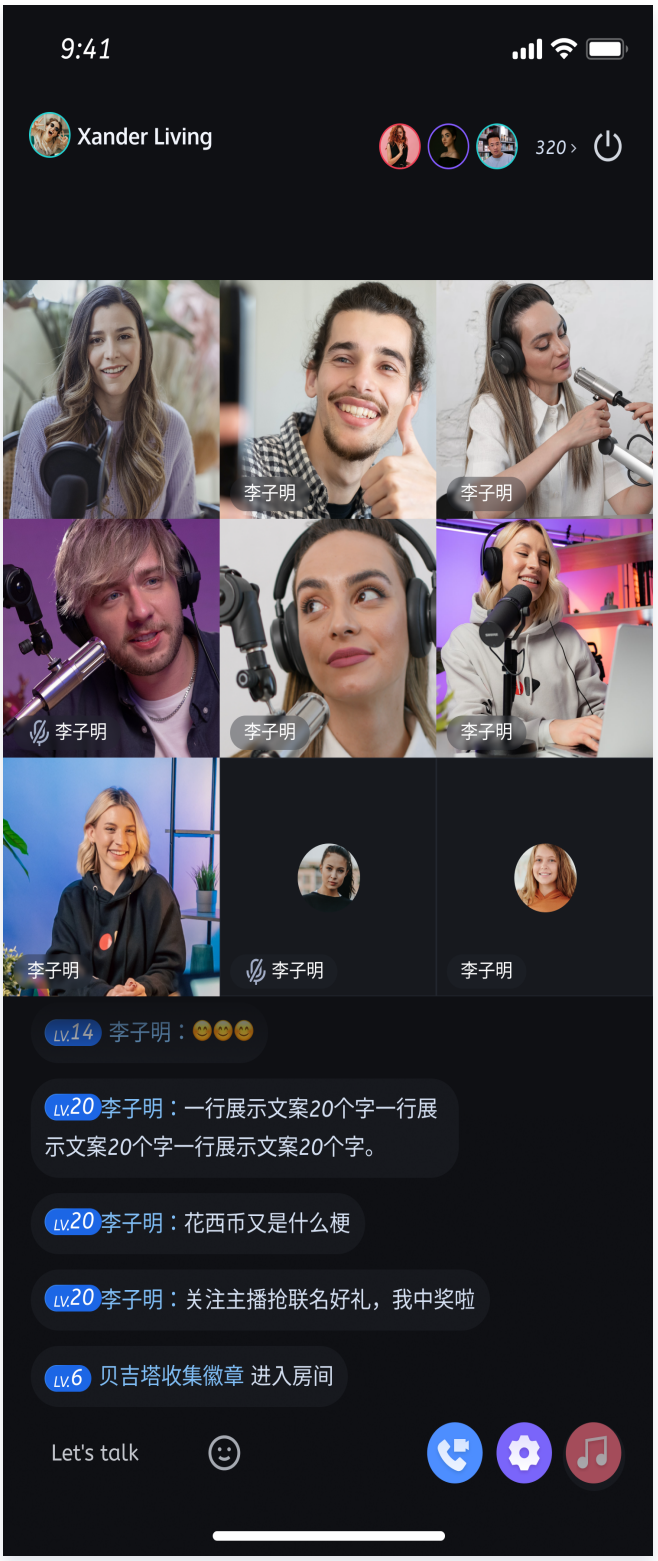
您在接入或使用过程有任何需要或者反馈，欢迎加入 腾讯云实时音视频 TRTC 技术交流群 进行技术交流和问题反馈。

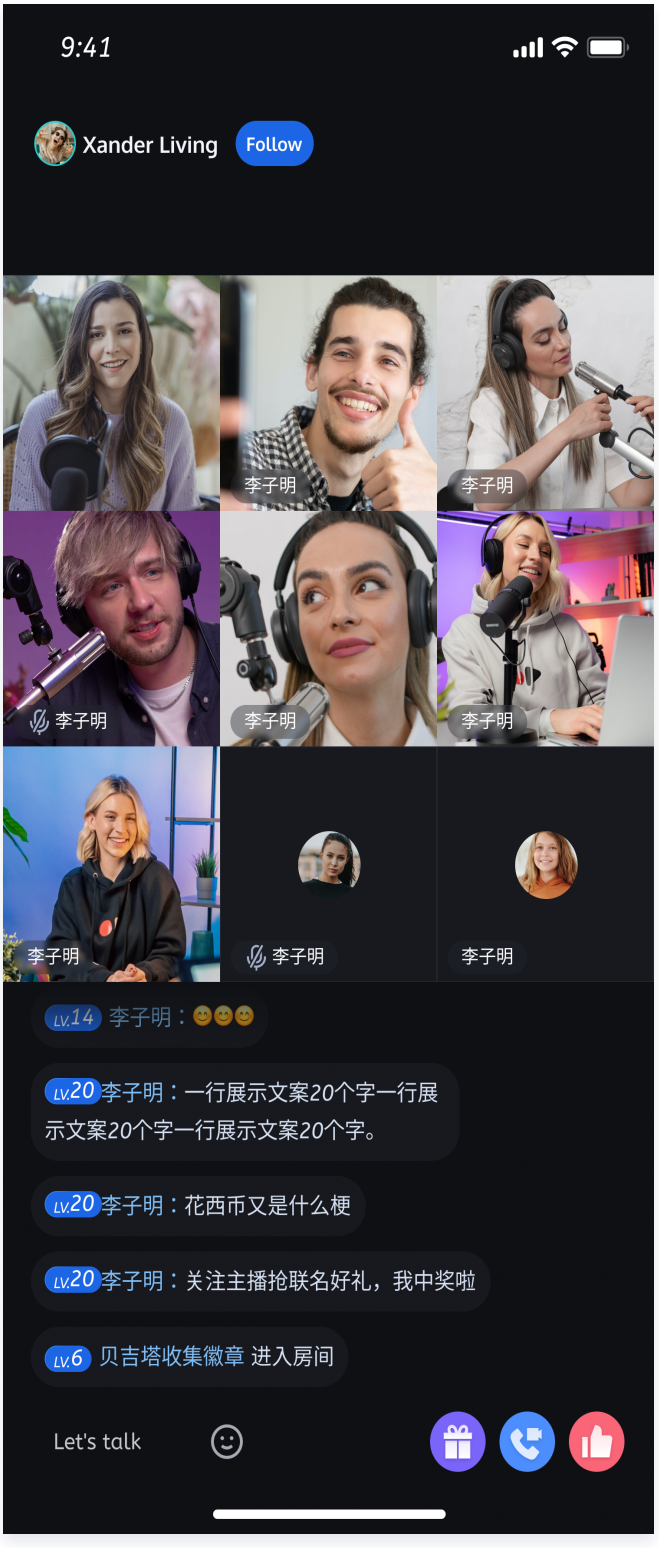
Flutter

最近更新时间：2025-04-14 09:19:31

本文将为您介绍如何快速跑通视频直播 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 界面的视频直播功能。

主播：直播中	观众：连麦中


9:41
Xander Living
320 ›
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
LV.14 李子明：😊😊😊
LV.20 李子明：一行展示文案20个字一行展示文案20个字一行展示文案20个字。
LV.20 李子明：花西币又是什么梗
LV.20 李子明：关注主播抢联名好礼，我中奖啦
LV.6 贝吉塔收集徽章 进入房间
Let's talk


9:41
Xander Living
Follow
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
LV.14 李子明：😊😊😊
LV.20 李子明：一行展示文案20个字一行展示文案20个字一行展示文案20个字。
LV.20 李子明：花西币又是什么梗
LV.20 李子明：关注主播抢联名好礼，我中奖啦
LV.6 贝吉塔收集徽章 进入房间
Let's talk

环境准备

平台	版本

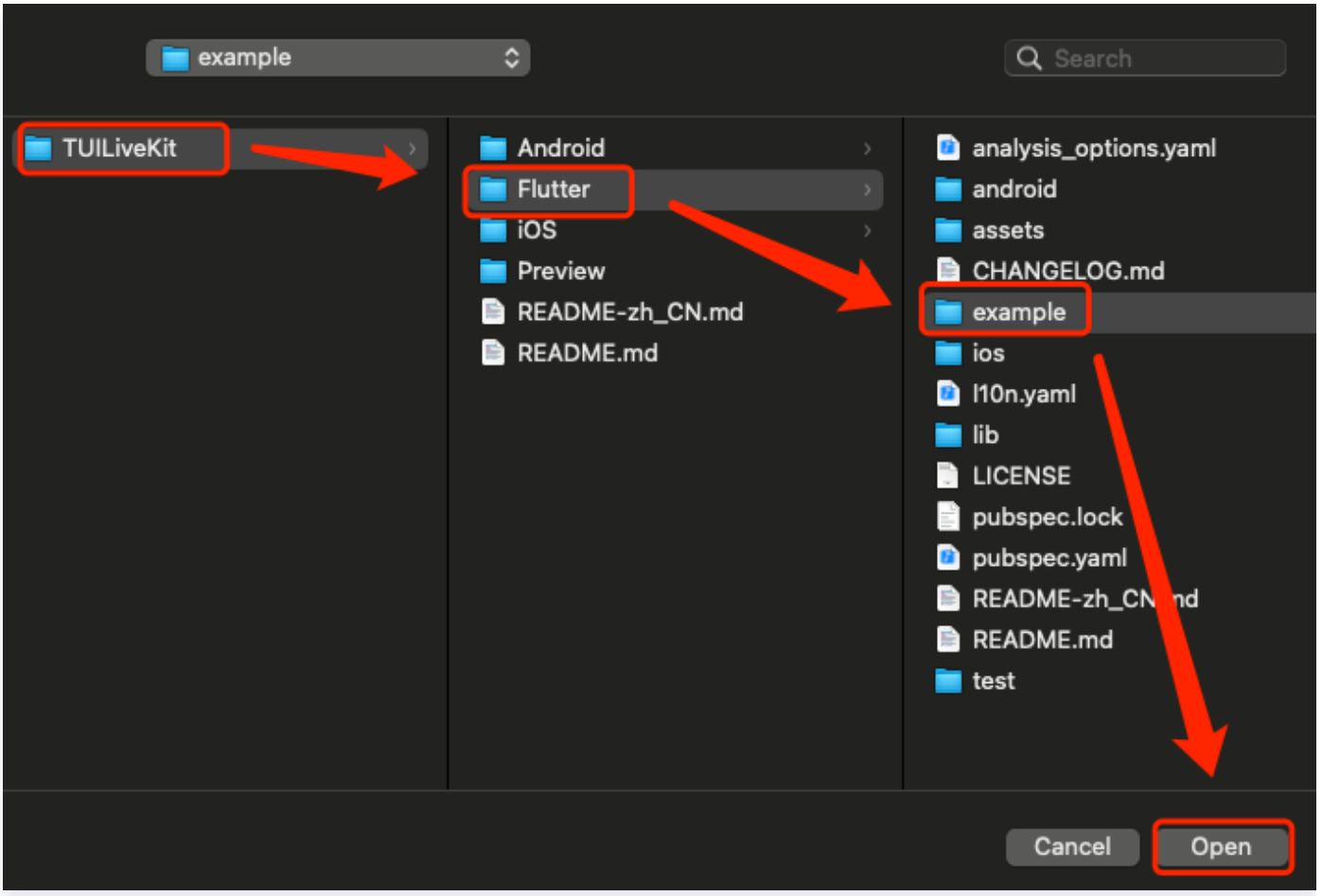
Flutter	• Flutter 3.27.4 及更高版本。 • Dart 3.6.2 或更高的版本。
Android	• Android Studio 3.5及以上版本。 • Android 5.0 及以上版本的 Android 设备。
iOS	• Xcode 15.0及以上版本。 • 请确保您的项目已设置有效的开发者签名。

步骤一：下载 Demo

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

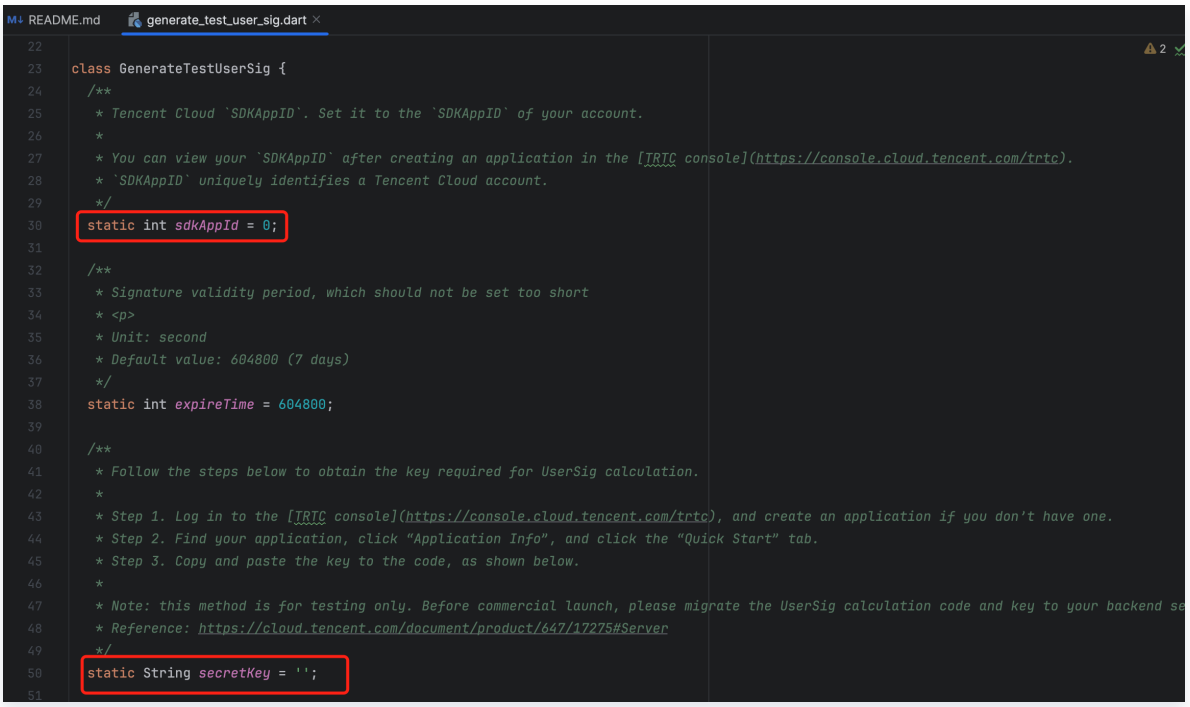
```
git clone https://github.com/Tencent-RTC/TUILiveKit.git
```

2. 通过 Android Studio 打开 TUILiveKit/Flutter/example 目录：



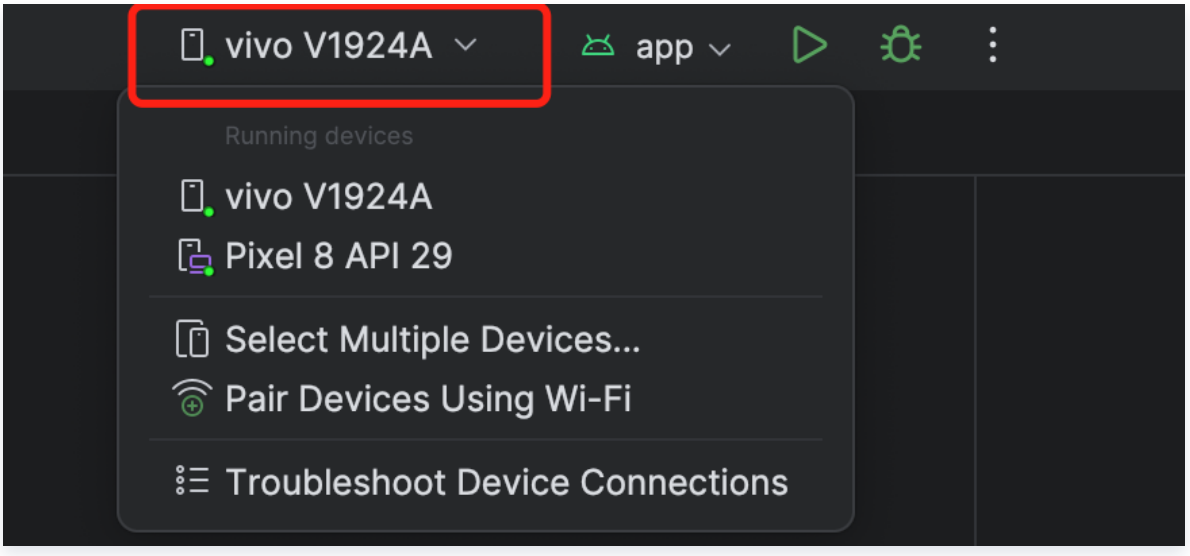
步骤二：配置 Demo

1. 请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。申请成功后即可获得14天的体验版，可免费体验 TUILiveKit 的全部功能，并获取 SDKAppID 和 SDKSecretKey 。
2. 打开 `Flutter/example/lib/debug/generate_test_user_sig.dart` 文件，将 提交申请 时获取到的对应的 SDKAppID 和 SDKSecretKey 填入其中：

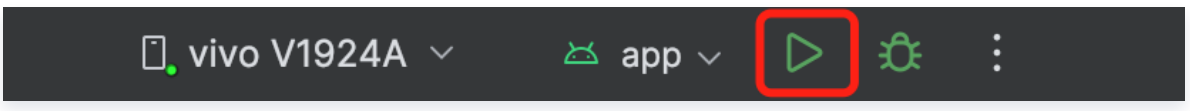


步骤三：跑通 Demo

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 TUILiveKit Flutter Demo 运行到目标设备上。



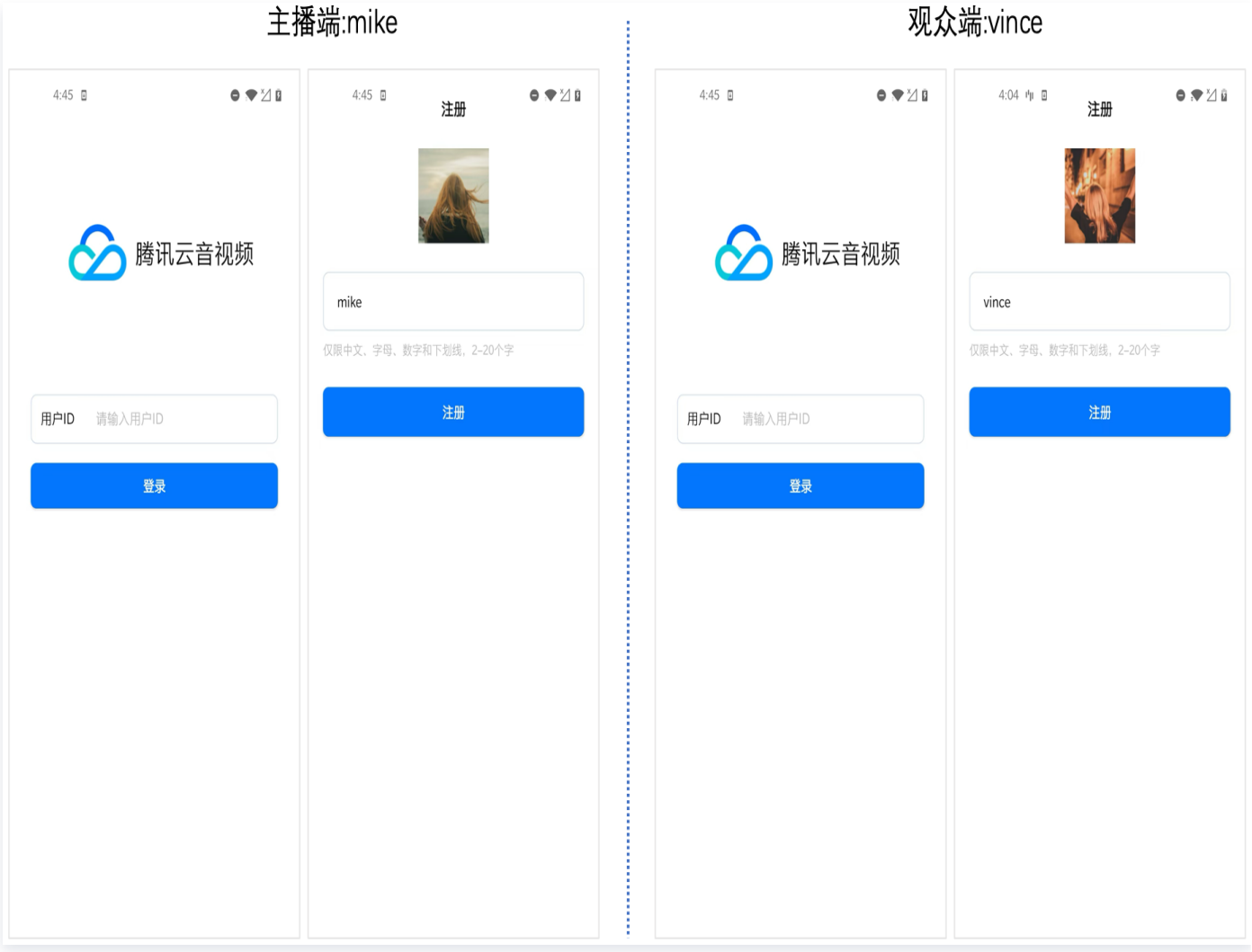
3. 当 Demo 成功运行到设备上后，您可以通过如下步骤来发起直播和观看直播。

开始直播

> **注意：**
> 为了使您可以体验完整的视频直播流程，请将 Demo 分别在两台设备上登录两个用户，一方作为主播，一方作为观众。

1. 登录 & 注册

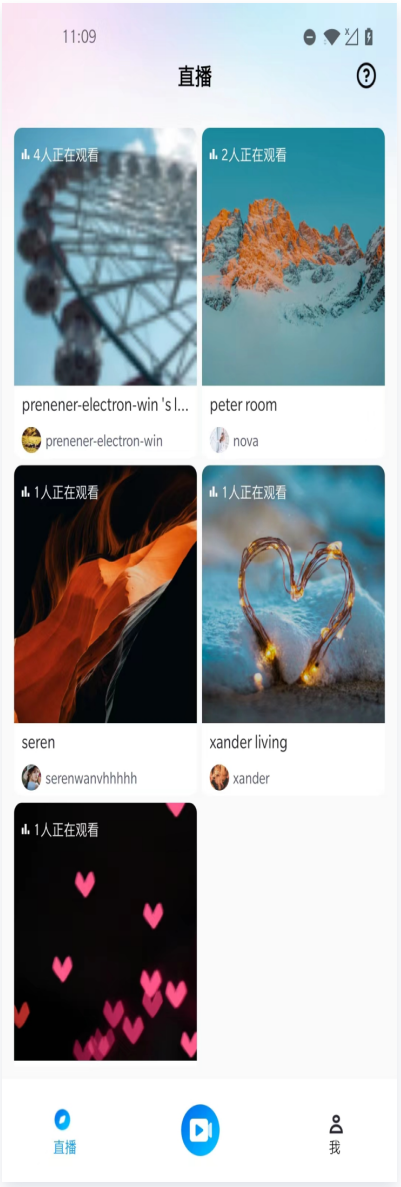
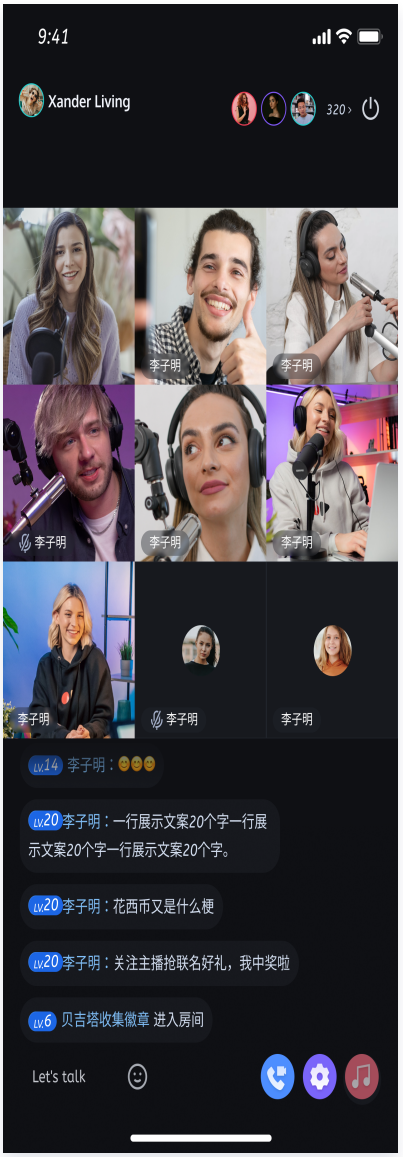
请您在 `用户ID` 处输入的 ID 。如果您当前的 UserID 未曾被使用过，此时会进入到**注册**界面，您可以在该界面为自己设置头像及昵称。



> **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. 主播开启直播

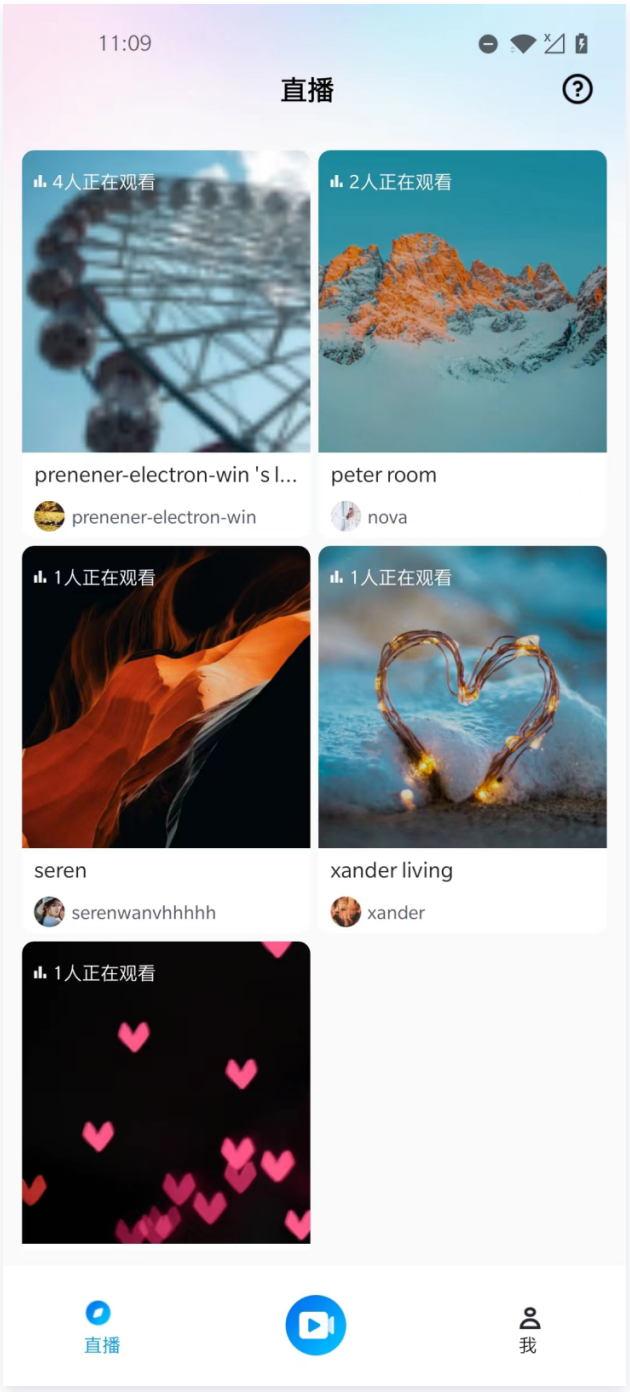
点击主页底部中间的“开播”按钮，即可进入开播预览页面，再点击“开始直播”即可开始直播了。

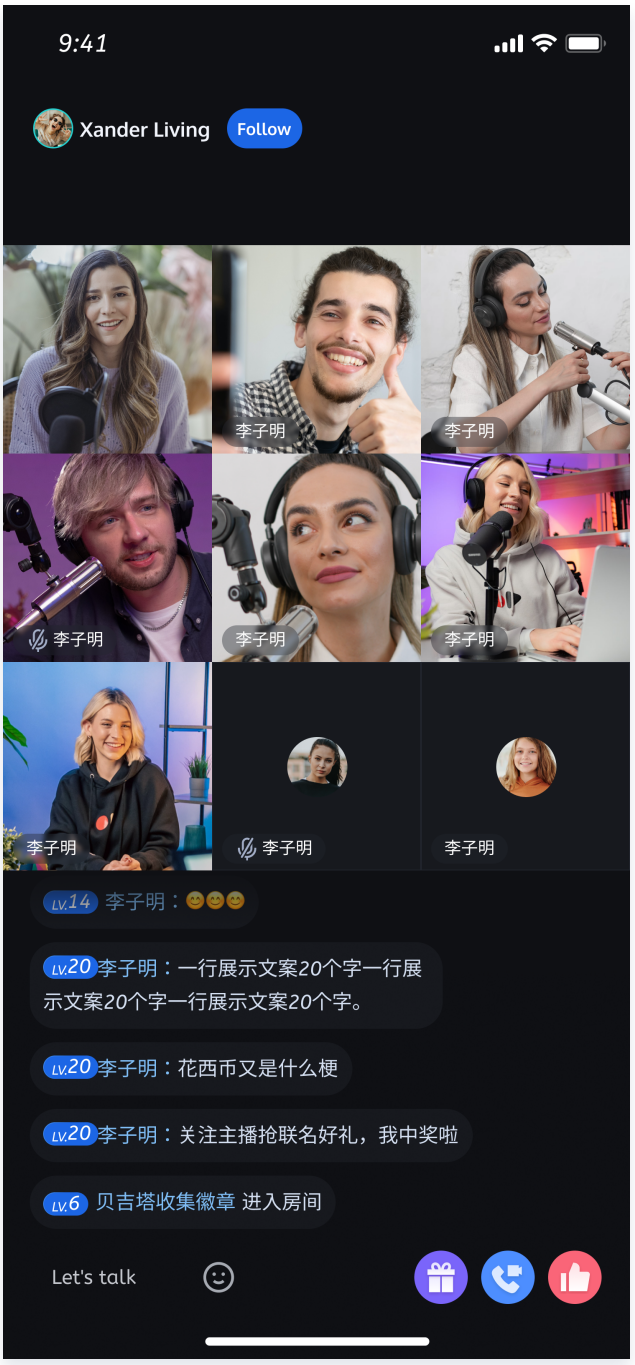
主播：进房前	主播：预览	主播：直播中
		

3. 观众加入直播间

点击直播列表里任意一个房间，即可进入直播间。

观众：进房前	观众：进房后

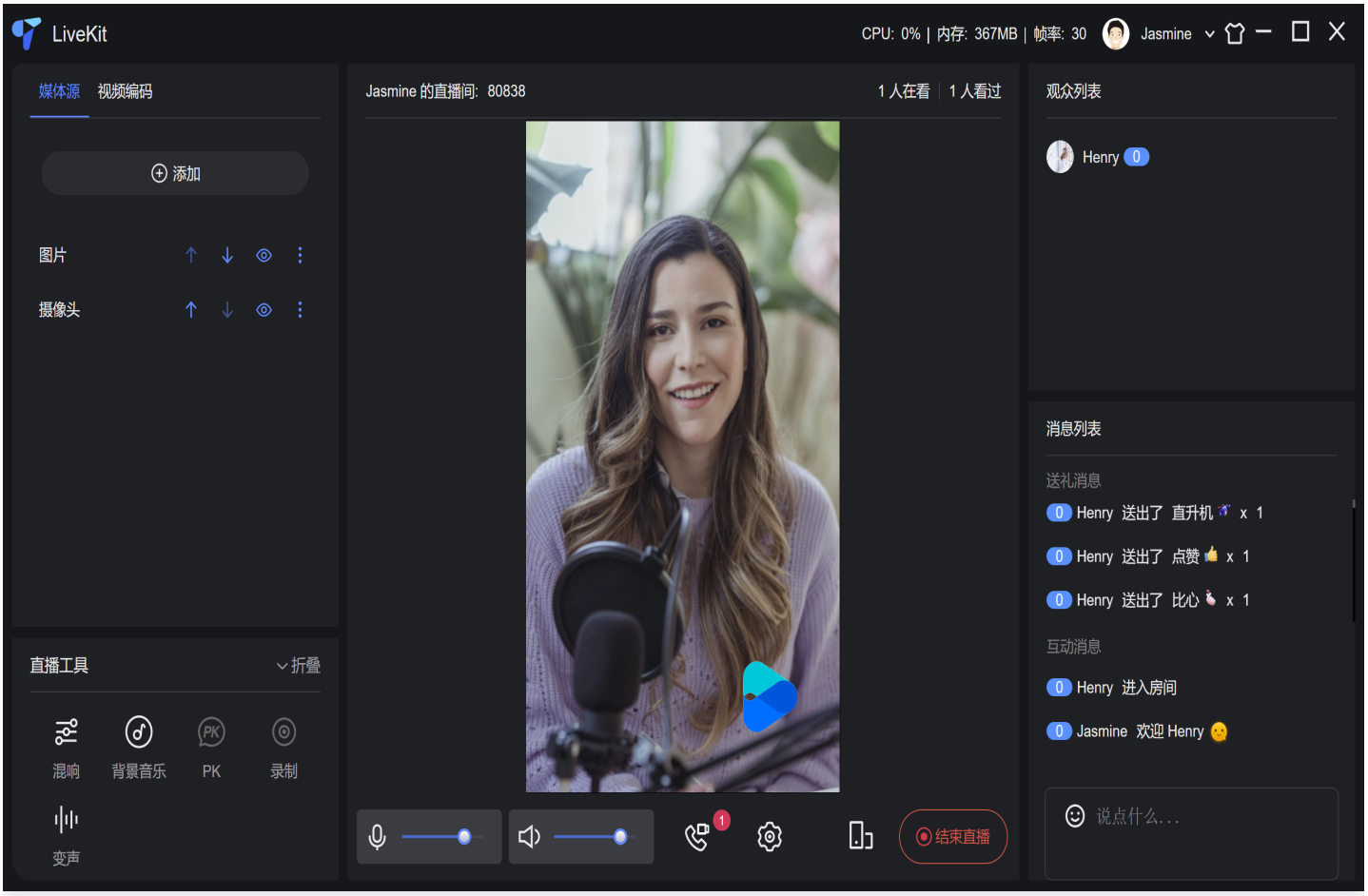

11:09
直播
4人正在观看
2人正在观看
prenener-electron-win 's l...
prenener-electron-win
peter room
nova
1人正在观看
1人正在观看
seren
serenwanvhhhhh
xander living
xander
1人正在观看
直播
我


9:41
Xander Living
Follow
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
李子明
lv.14 李子明：😊😊😊
lv.20 李子明：一行展示文案20个字一行展示文案20个字一行展示文案20个字。
lv.20 李子明：花西币又是什么梗
lv.20 李子明：关注主播抢联名好礼，我中奖啦
lv.6 贝吉塔收集徽章 进入房间
Let's talk

Electron

最近更新时间：2025-04-14 10:13:52

本文将为您介绍如何快速跑通桌面端高清视频直播应用程序。跟随本文档，您将在10 分钟内开始一场直播。桌面端高清视频直播支持将各种多媒体资源合并成一路视频流，支持视频源的编辑、合成、自定义布局。



环境准备

- 操作系统：Windows 10+ 。
- 设备要求：摄像头、麦克风、扬声器。

步骤一：下载源码

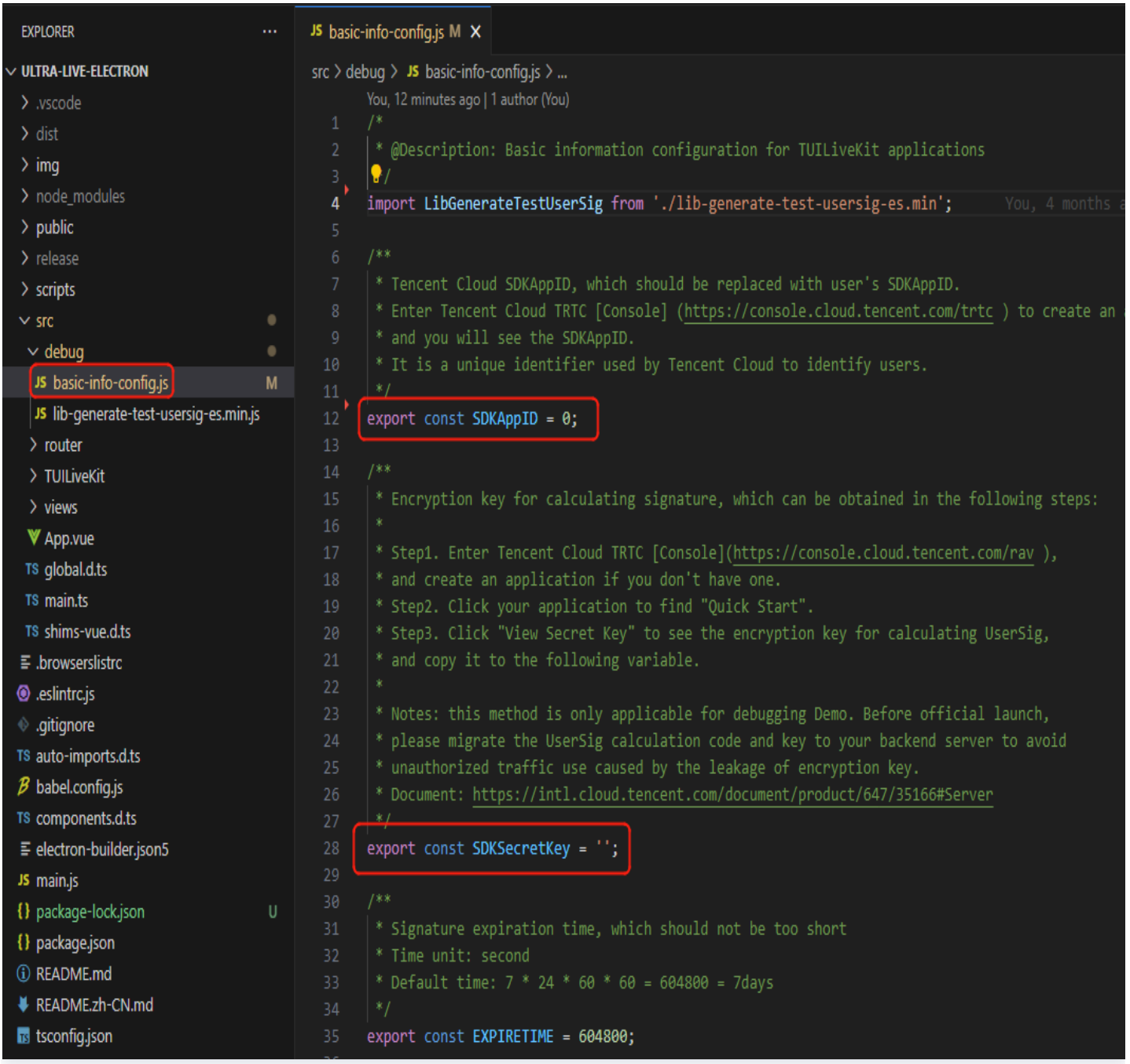
从 Github 下载 项目源码，或者执行以下命令下载：

```
git clone https://github.com/Tencent-RTC/ultra-live-electron.git

cd ultra-live-electron
```

步骤二：配置

1. 请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。申请成功后即可获得14天的体验版，可免费体验 TUILiveKit 的全部功能，并获取 SDKAppID 和 SDKSecretKey 。
2. 打开 `src/debug/basic-info-config.js` 文件，将 提交申请 时获取到的 `SDKAppID` 和 `SDKSecretKey` 填入其中。



步骤三：运行

在代码目录下，执行以下命令，以开发模式启动应用程序：

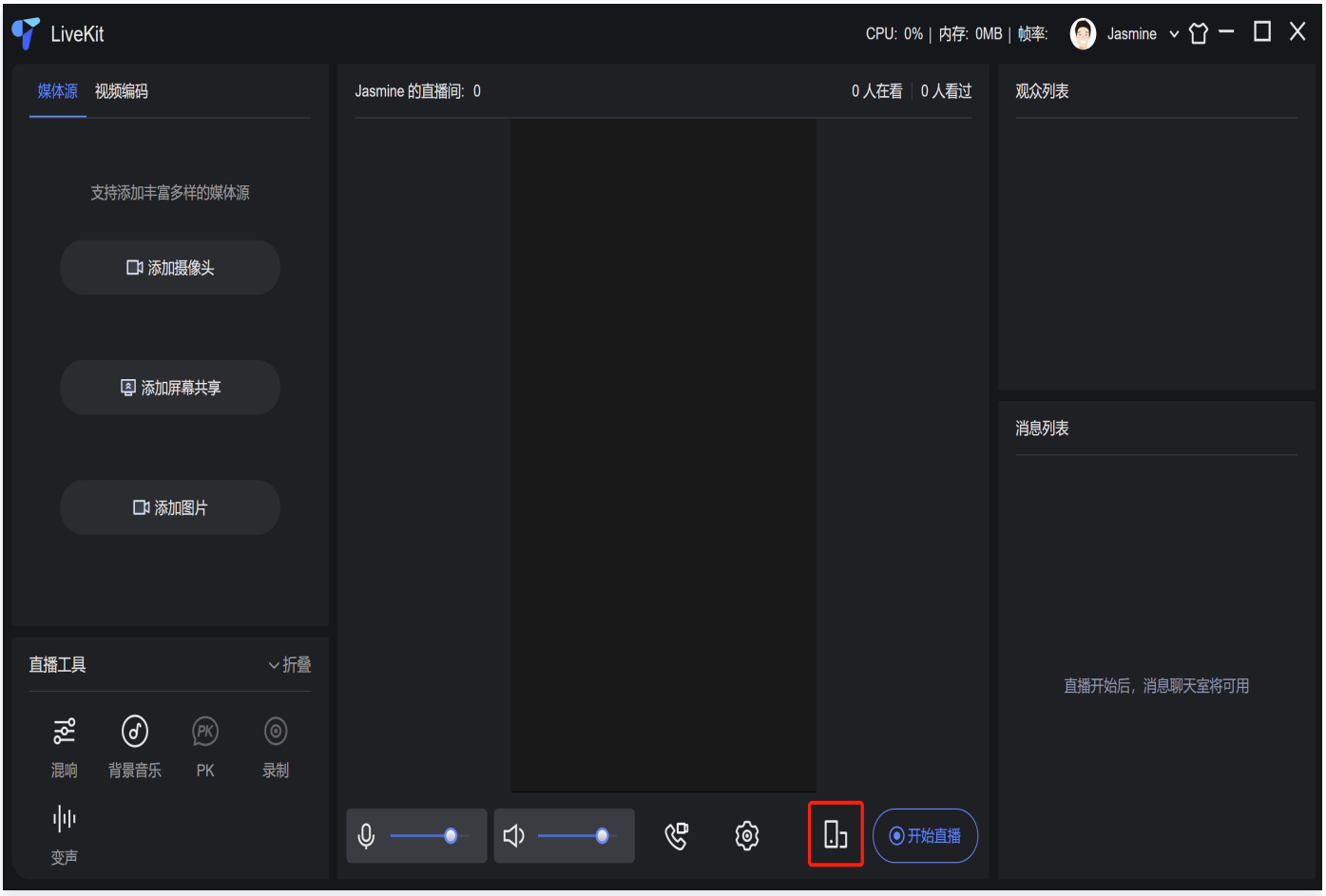
```
npm install

npm run start
```

开始第一场直播

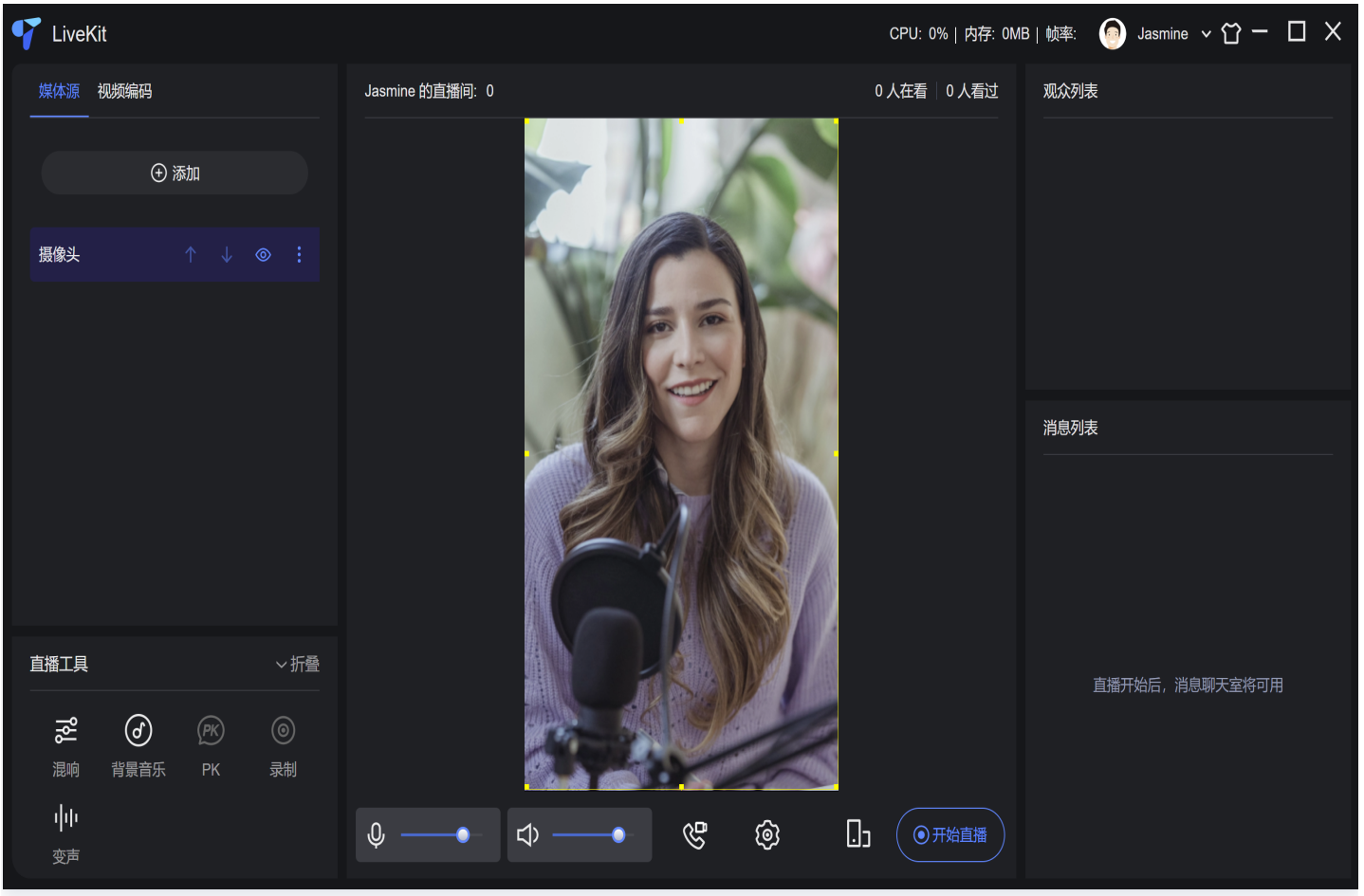
1. 切换竖屏

   切换竖屏开播，手机端观看效果更佳。



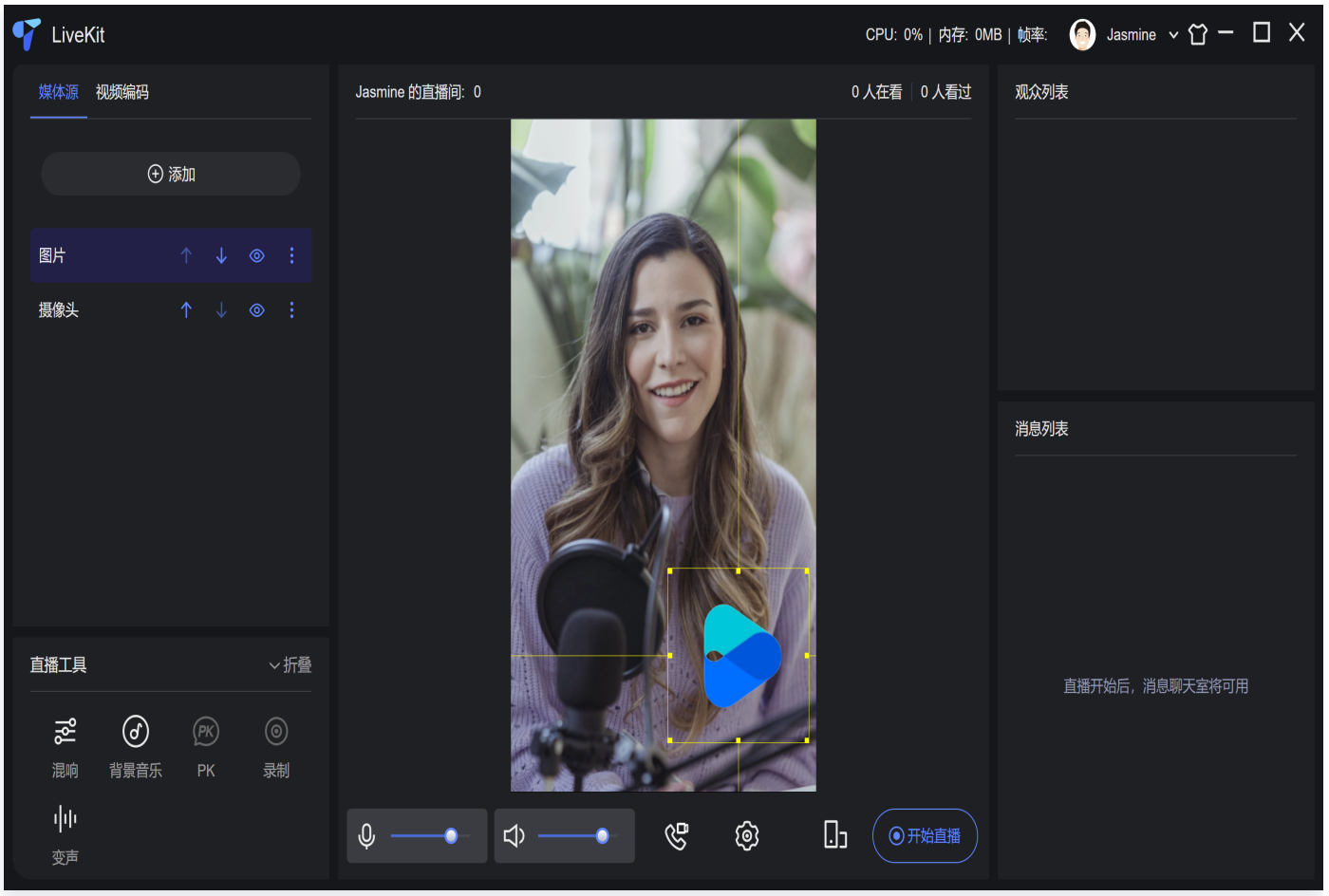
2. 添加摄像头

   在开播前，需要先添加画面源，目前支持：摄像头、屏幕分享、窗口分享和图片，如下图是添加一个摄像头后的效果。



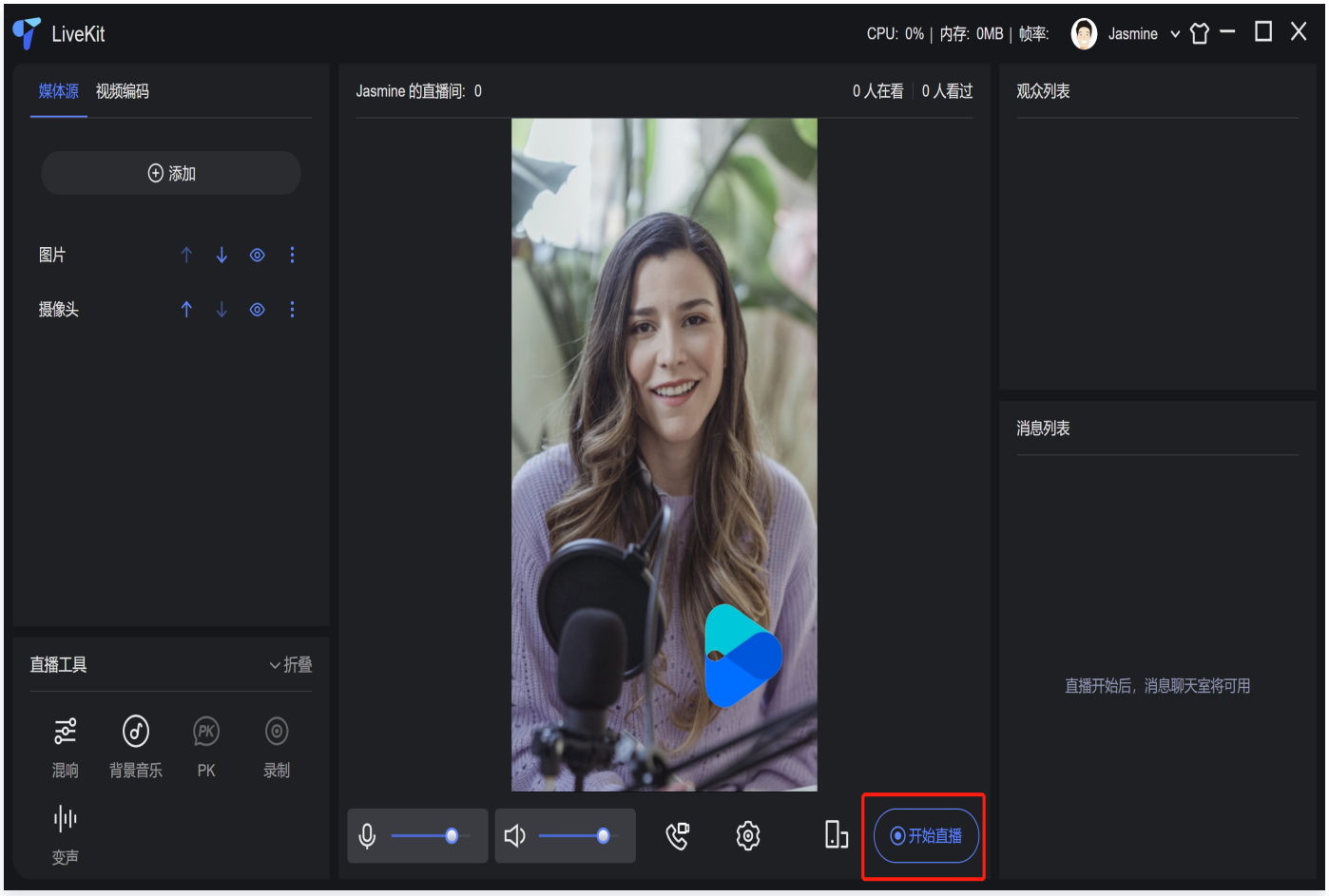
3. 添加 Logo 图片

如果您在直播时，需要添加自己的品牌 Logo，可以添加 Logo 图片。如下图，是添加一个背景透明的 Logo 图片后的效果，新添加的图片周围会有黄色边框，表示当前处于选中状态。处于选中状态的多媒体源，可以使用鼠标移动位置、调整大小，使用右键菜单对多媒体源进行旋转、修改显示层级等。



4. 开始直播

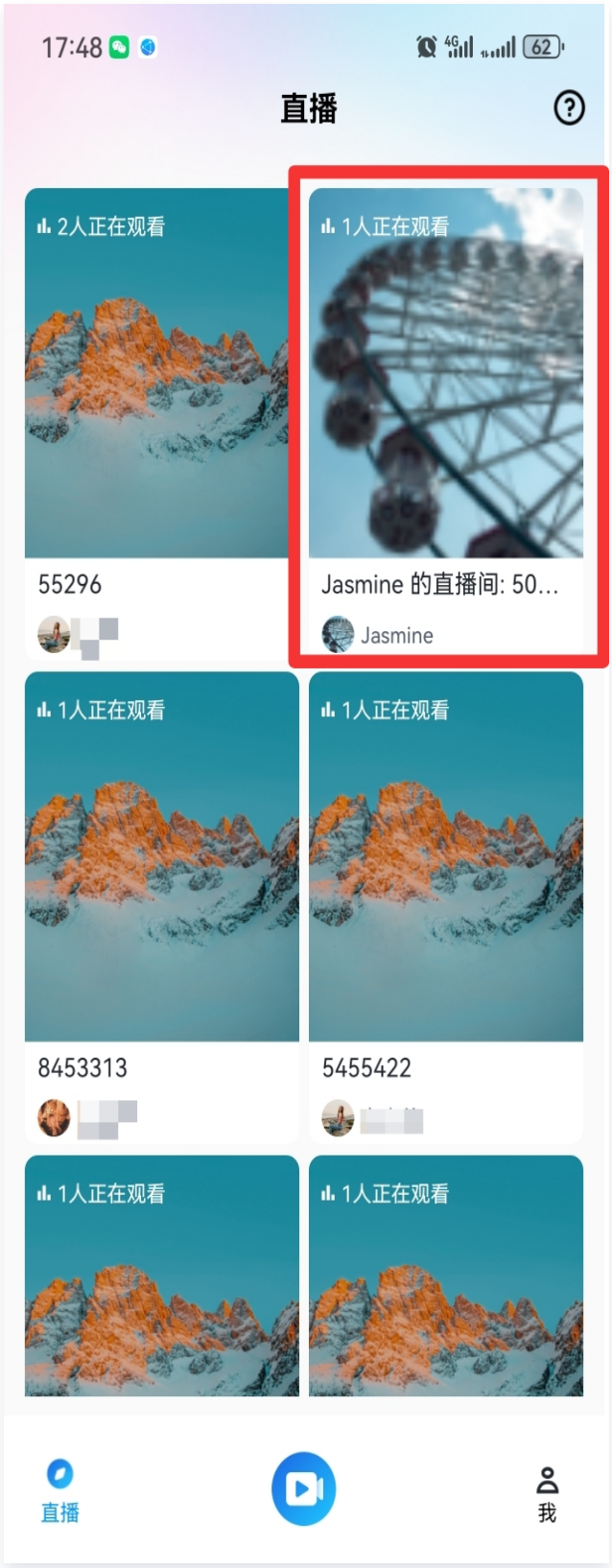
单击**开始直播**，将开启直播推流，成功开播后，**开始直播**按钮会变成**结束直播**，单击可结束直播推流。

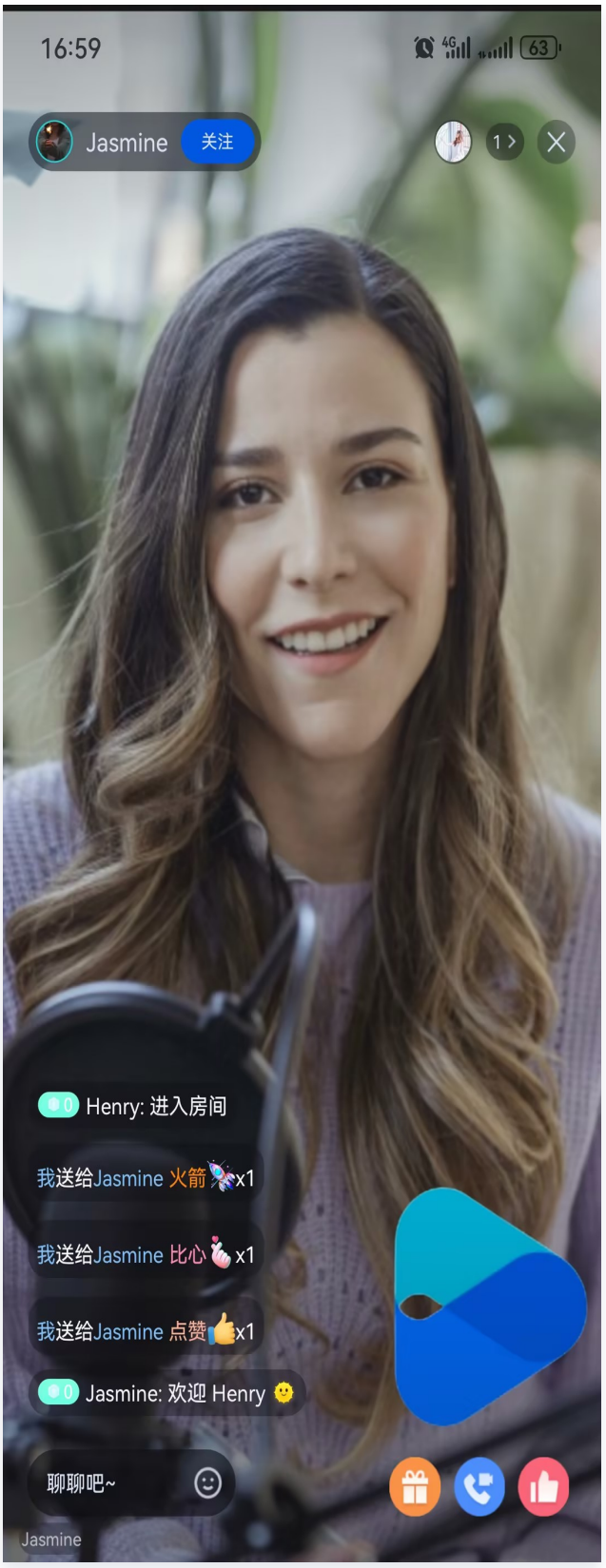


5. 观看直播

桌面端只支持主播开播，观看需要使用手机端 App，在手机端 App 的直播列表中找到对应直播间，进入直播。手机端 App 的使用，可参照 iOS 和 Android 的跑通文档。

观众：浏览直播列表	观众：进入直播间

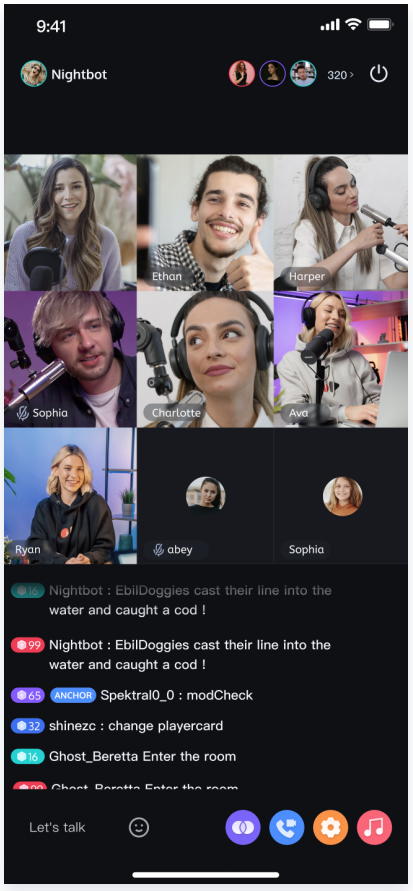
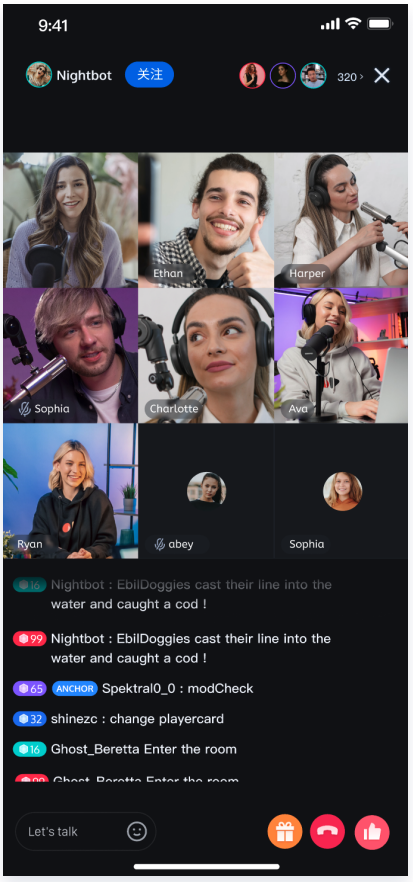

17:48
直播
2人正在观看
55296
1人正在观看
Jasmine 的直播间: 50...
Jasmine
1人正在观看
8453313
1人正在观看
5455422
1人正在观看
1人正在观看
直播
我


16:59
Jasmine
关注
Henry: 进入房间
我送给Jasmine 火箭x1
我送给Jasmine 比心x1
我送给Jasmine 点赞x1
Jasmine: 欢迎 Henry
聊聊吧~
Jasmine

快速接入（TUILiveKit）
Android

最近更新时间：2025-05-23 09:49:51

本文将介绍如何在短时间内完成 TUILiveKit 组件的接入，跟随本文档，您可以在10分钟内完成接入工作，并最终得到一个包含完备 UI 界面的直播功能。

主播（视频直播）	观众（视频直播）
	

环境准备

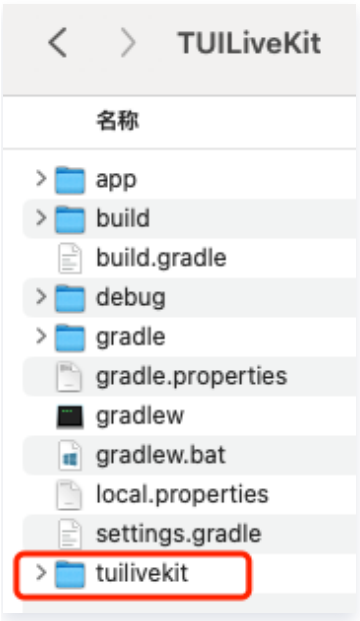
- Android 5.0（SDK API Level 21）及以上版本。
- Gradle 7.0 及以上的版本。
- Android 5.0 及以上的手机设备。
- 环境配置或编译运行期间，如有问题，请参见 常见问题 。

步骤1：开通服务

请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。

步骤2：下载 TUILiveKit 组件

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



步骤3：工程配置

1. 在工程根目录下的 `settings.gradle.kts`（或`settings.gradle`）文件中添加 jitpack 仓库地址：中添加 jitpack 仓库依赖（下载播放礼物 svg 动画的三方库 SVGAPlayer）：

settings.gradle.kts

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()

        // 添加 jitpack 仓库地址
        maven { url = uri("https://jitpack.io") }
    }
}
```

settings.gradle

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
```

```
        google()
        mavenCentral()

        // 添加 jitpack 仓库地址
        maven { url 'https://jitpack.io' }
    }
}
```

2. 在工程根目录的 `settings.gradle.kts`（或`settings.gradle`）文件中增加如下代码，它的作用是将 步骤二 中下载的 tuilivekit 组件导入到您当前的项目中：

settings.gradle.kts

```
include(":tuilivekit:livekit")
include(":tuilivekit:component:barrage")
```

settings.gradle

```
include ':tuilivekit:livekit'
include ':tuilivekit:component:barrage'
```

3. 在 app 目录下找到 `build.gradle.kts`（或`build.gradle`）文件，并在其中增加如下代码，它的作用是声明当前 app 对新加入的 tuilivekit 组件的依赖：

build.gradle.kts

```
api(project(":tuilivekit:livekit"))
```

build.gradle

```
api project(':tuilivekit:livekit')
```

> **说明：**
> TUILiveKit 工程内部已经默认依赖：`TRTC SDK`、`IM SDK`、`tuiroomengine` 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 `tuilivekit/build.gradle` 文件即可。

4. 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.** { *; }
-keep class com.trtc.uikit.livekit.livestreamcore.** { *; }
-keep class com.trtc.uikit.livekit.component.gift.store.model.** { *; }
-keep class com.squareup.wire.** { *; }
-keep class com.opensource.svgaplayer.proto.** { *; }

-keep class com.tcmediax.** { *; }
-keep class com.tencent.** { *; }
-keep class com.tencent.xmagic.** { *; }
-keep class androidx.exifinterface.** {*;}
-keep class com.google.gson.** { *;}
# Tencent Effect SDK - beauty
-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.** { *;}
```

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

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

<application
  ...

  // 添加如下配置覆盖 依赖的 sdk 中的配置
  android:allowBackup="false"
  tools:replace="android:allowBackup">
```

步骤4：登录

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 TUILiveKit 提供的各项功能。

Kotlin

```
//登录
TUILogin.login(applicationContext,
    1400000001,  // 请替换为步骤一取到的 SDKAppID
    "denny",  // 请替换为您的 UserID
    "xxxxxxxxxx",  // 您可以在控制台中计算一个 UserSig 并填在这个位置
    object : TUICallback() {
        override fun onSuccess() {
            Log.i(TAG, "login success")
        }

        override fun onError(errorCode: Int, errorMessage: String) {
            Log.e(TAG, "login failed, errorCode: $errorCode
msg:$errorMessage")
        }
    })
```

Java

```
//登录
TUILogin.login(context,
    1400000001,      // 请替换为步骤一取到的 SDKAppID
    "denny",         // 请替换为您的 UserID
    "xxxxxxxxxx",  // 您可以在控制台中计算一个 UserSig 并填在这个位置
    new TUICallback() {
    @Override
    public void onSuccess() {
        Log.i(TAG, "login success");
    }

    @Override
    public void onError(int errorCode, String errorMessage) {
        Log.e(TAG, "login failed, errorCode: " + errorCode + " msg:" +
errorMessage);
    }
});
```

参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

参数	类型	说明
SDKAppID	int	在步骤一中的最后一步中您已经获取到，这里不再赘述。
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	使用 步骤一 的第3步中获取的 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 。

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 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：开始直播

在上述登录方法调用返回成功后，调用 VideoLiveKit 的 startLive 方法，指定房间 ID，开启您的直播页面。

Kotlin

```
import com.trtc.uikit.livekit.VideoLiveKit

VideoLiveKit.createInstance(applicationContext).startLive("roomId")
```

Java

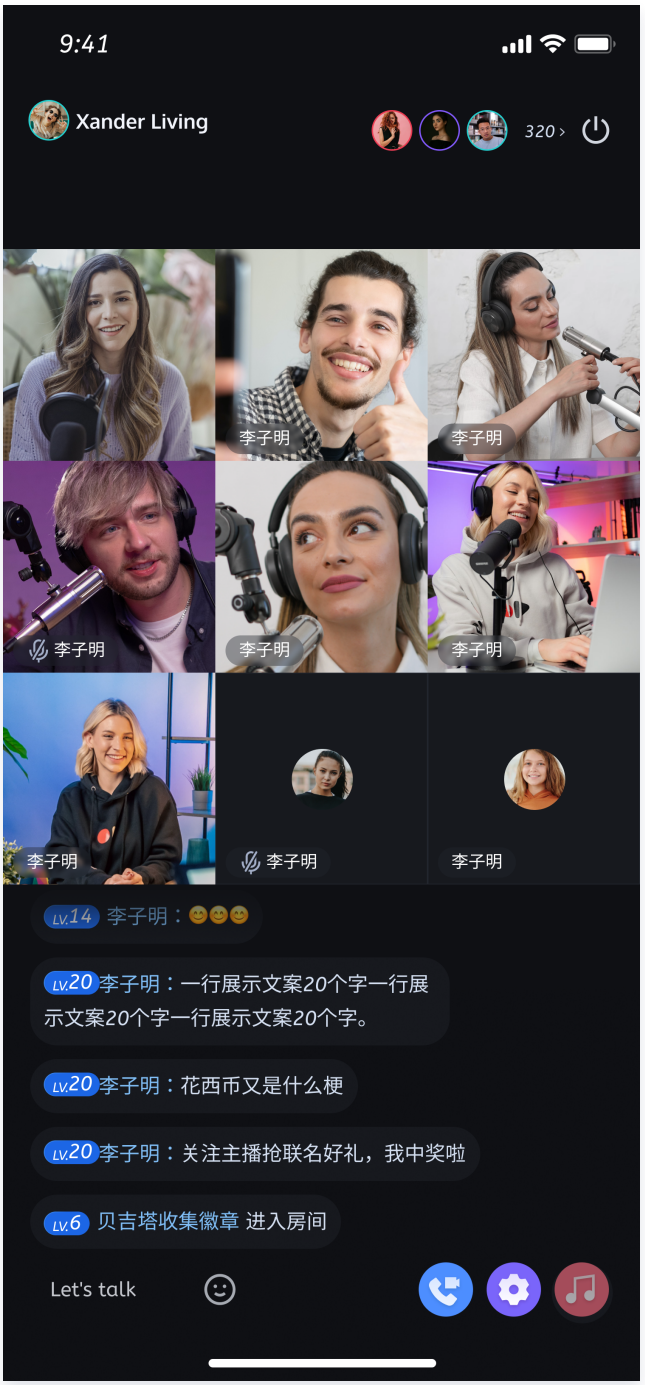
```
import com.trtc.uikit.livekit.VideoLiveKit;

VideoLiveKit.createInstance(getApplicationContext()).startLive("roomId");
```

主播开播预览页	主播直播中



视频直播预览画面



视频直播中画面

步骤6：观看直播

在上述登录方法调用返回成功后，调用 VideoLiveKit 的 joinLive 方法，指定房间 ID，开启您的直播页面。

Kotlin

```
import com.trtc.uikit.livekit.VideoLiveKit
```

```
VideoLiveKit.createInstance(applicationContext).joinLive("roomId")
```

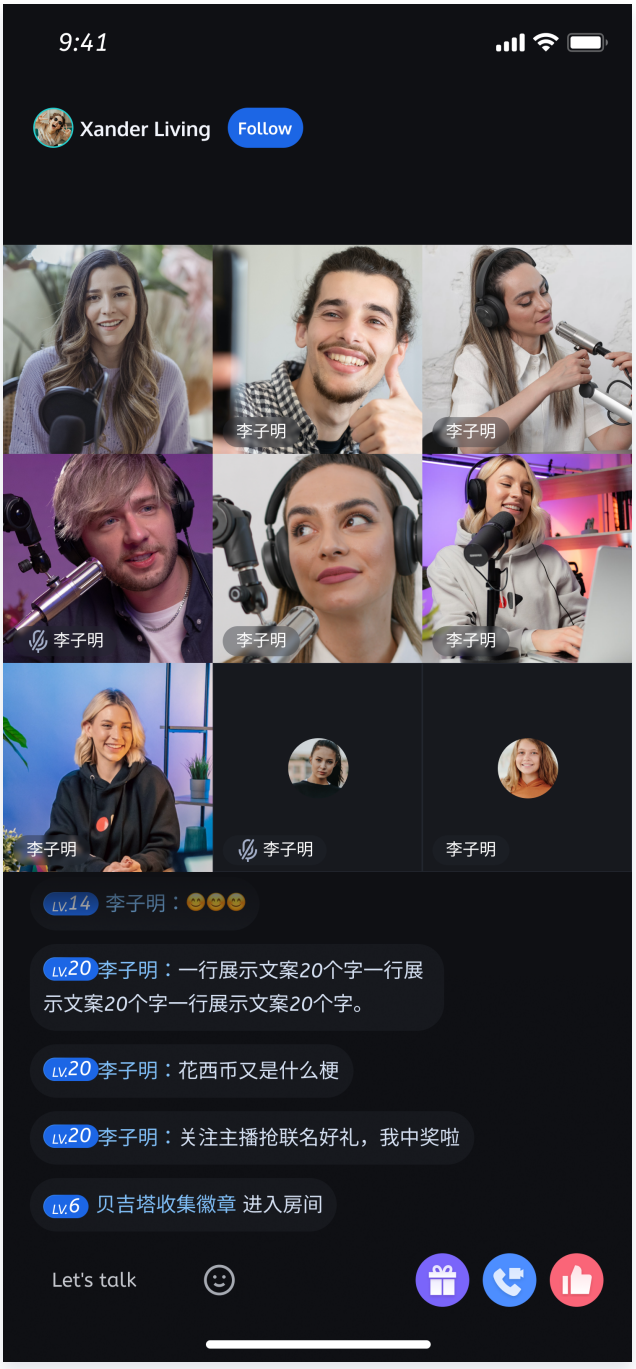
Java

```
import com.trtc.uikit.livekit.VideoLiveKit;

VideoLiveKit.createInstance(getApplicationContext()).joinLive("roomId");
```

观看直播	和主播连麦





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

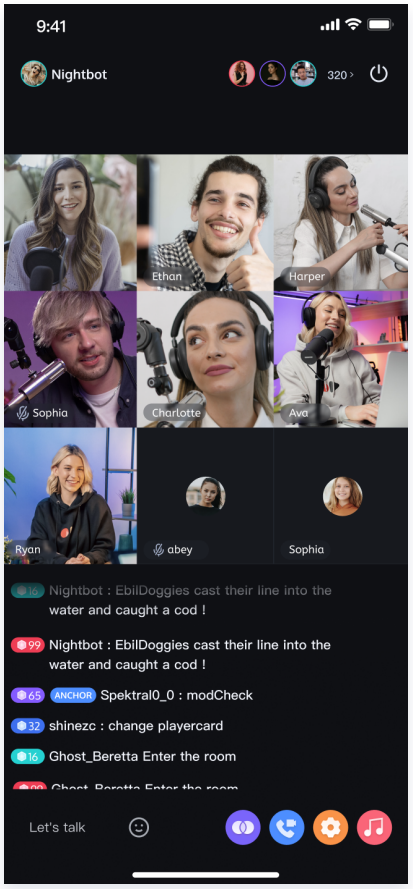
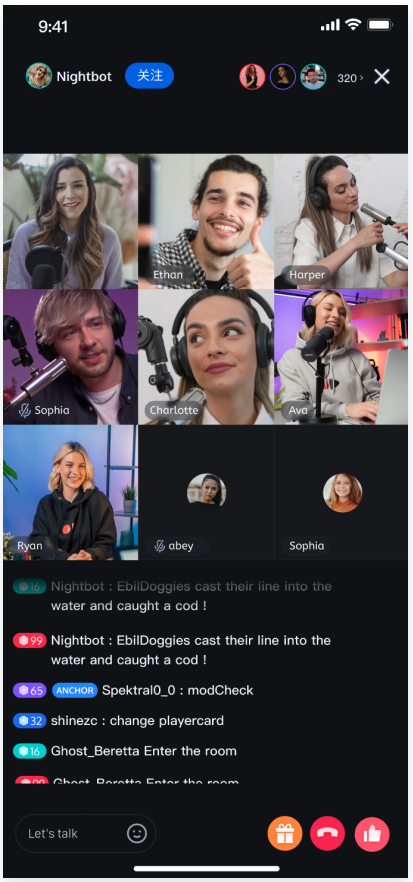
如果您在接入和使用过程有任何需要或者反馈，欢迎加入我们的 TUILiveKit 技术交流平台 zhiliao ，进行技术交流和问题反馈。

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2025-05-20 18:01:52

本文将介绍如何在短时间内完成 TUILiveKit 组件的接入，跟随本文档，您可以在10分钟内完成接入工作，并最终得到一个包含完备 UI 界面的直播功能。

主播（视频直播）	观众（视频直播）
	

环境准备

- Xcode 15 及以上。
- iOS 13.0 及以上。
- CocoaPods 环境安装，点击查看 。
- 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（TUILiveKit），领取体验版或者开通付费版。

步骤二：导入 TUILiveKit 组件

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

1. 请在您的 `Podfile` 文件中添加 `pod 'TUILiveKit'` 依赖，如果您遇到任何问题，请参见 Example 工程。

```
target 'xxxx' do
  ...
  ...
  pod 'TUILiveKit'
end
```

如果您没有 `Podfile` 文件，首先终端 `cd` 到 `xxxx.xcodeproj` 目录，然后通过以下命令创建：

```
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
```

如果无法安装 TUILiveKit 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。

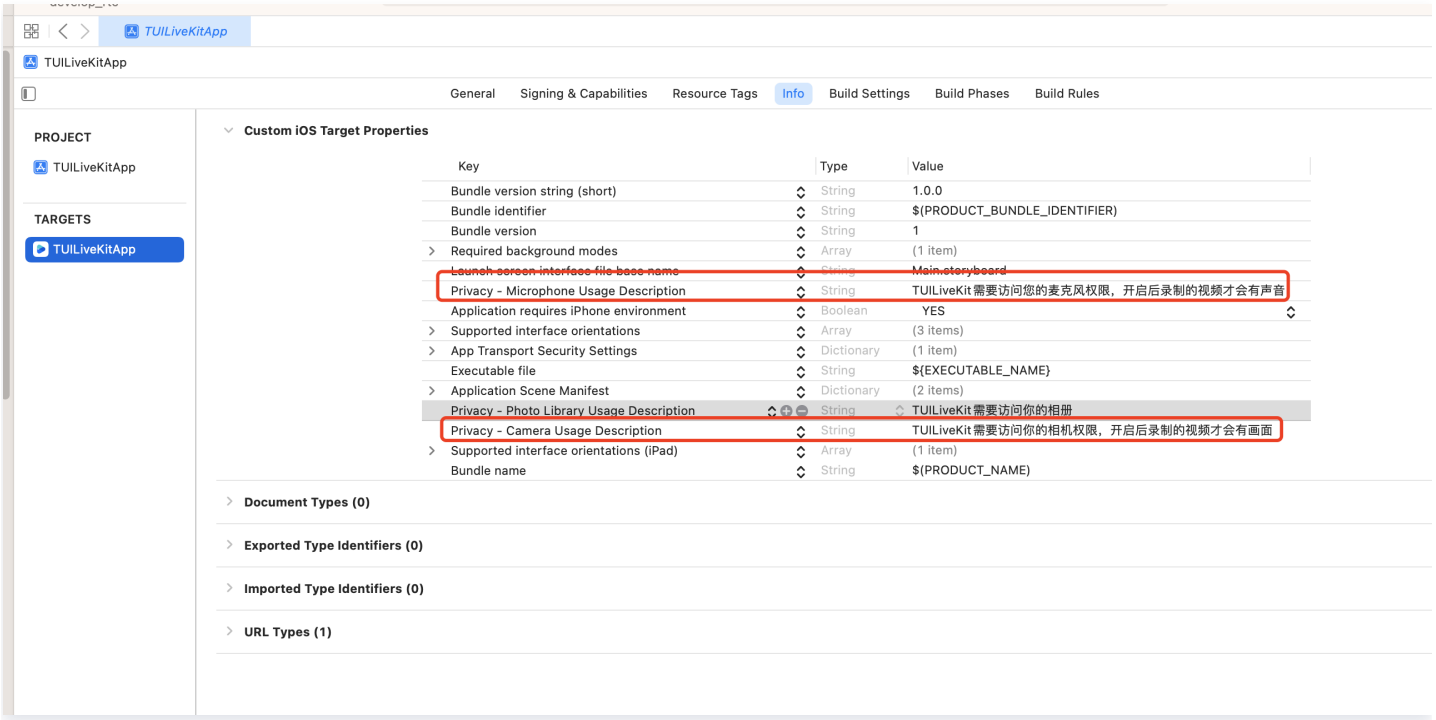
```
pod update
```

3. 可以先编译运行一下，如果遇到问题，请参见 常见问题 。问题如果依然无法解决，可以先去跑一下我们的 Example 工程。您在接入和使用过程中遇到的任何问题，欢迎给我们 反馈 。

步骤三：工程配置

使用音视频功能，需要授权麦克风和摄像头的使用权限。在 App 的 Info 中添加以下两项，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>TUILiveKit需要访问你的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>TUILiveKit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：登录

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。这个步骤异常关键，因为只有在登录成功后才能正常使用 TUILiveKit 的各项功能，故请您耐心检查相关参数是否配置正确：

Swift

```
//
//  AppDelegate.swift
//

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

    TUILogin.login(1400000001,                  // 请替换为步骤一取到的 SDKAppID
            userID: "denny",                    // 请替换为您的 UserID
            userSig: "xxxxxxxxxx") {            // 您可以在控制台中计算一个
UserSig 并填在这个位置
      print("login success")
    } fail: { (code, message) in
      print("login failed, code: \(code), error: \(message ?? "nil")")
    }

    return true
```

```
}
```

Objective-C

```
//
//  AppDelegate.m
//

#import <TUICore/TUILogin.h>
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [TUILogin login:1400000001          // 请替换为步骤一取到的 SDKAppID
             userID:@"denny"            // 请替换为您的 UserID
            userSig:@"xxxxxxxxxxx"      // 您可以在控制台中计算一个 UserSig 并填
在这个位置
               succ:^{
        NSLog(@"login success");
    } fail:^(int code, NSString * _Nullable msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }];
    return YES;
}
```

参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

- **SDKAppID**：在步骤一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 的第3步中获取的 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 TUILiveKit 组件时向您的服务器请求实时计算出的 userSig。

步骤五：开始直播

在上述登录方法调用返回成功后，调用 VideoLiveKit 的 startLive 方法，指定房间 ID，开启您的直播页面。

Swift

```
import TUILiveKit

VideoLiveKit.createInstance().startLive(roomId: "your room id")
```

Objective-C

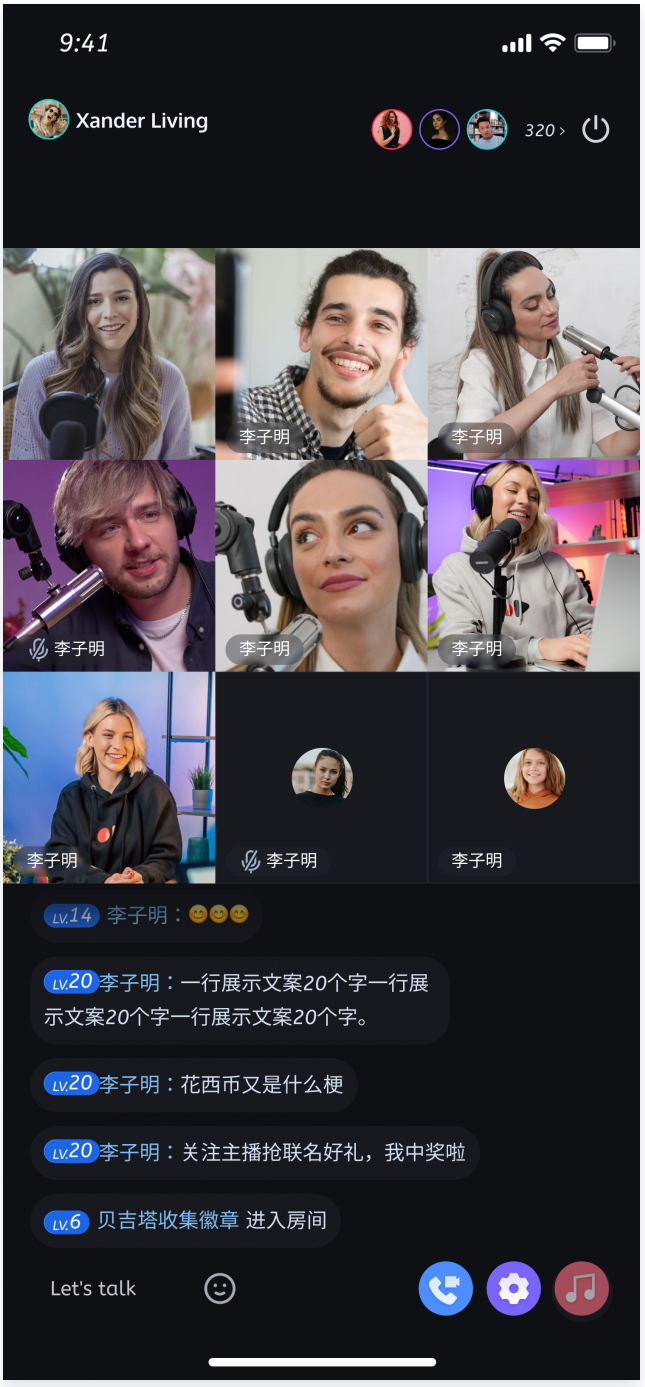
```
#import <TUILiveKit/TUILiveKit-Swift.h>

[[VideoLiveKit createInstance] startLiveWithRoomId:@"your room id"];
```

主播开播预览页	主播直播中



视频直播预览画面



视频直播中画面

步骤六：观看直播

在上述登录方法调用返回成功后，调用 VideoLiveKit 的 joinLive 方法，指定房间 ID，开启您的直播页面。

Swift

```
import UIKit
import TUILiveKit
```

```
VideoLiveKit.createInstance().joinLive(roomId: "your room id")
```

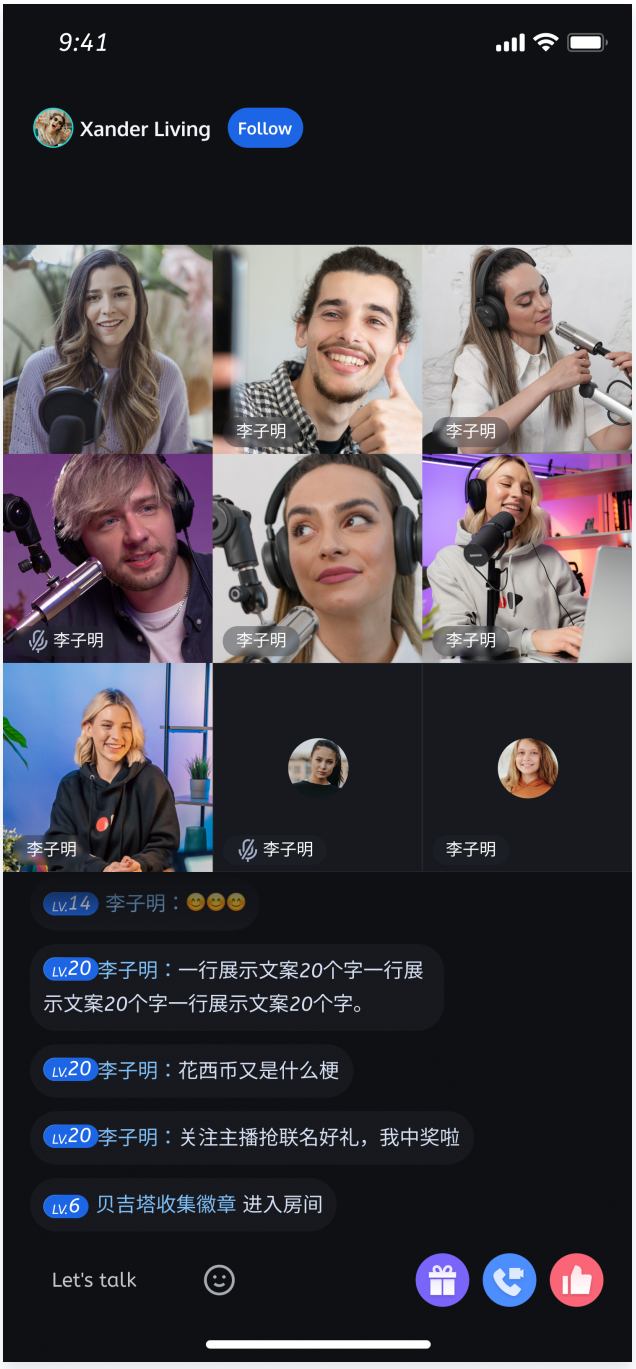
Objective-C

```
#import <TUILiveKit/TUILiveKit-Swift.h>

[[VideoLiveKit createInstance] joinLiveWithRoomId:@"your room id"];
```

观看直播	和主播连麦





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 。

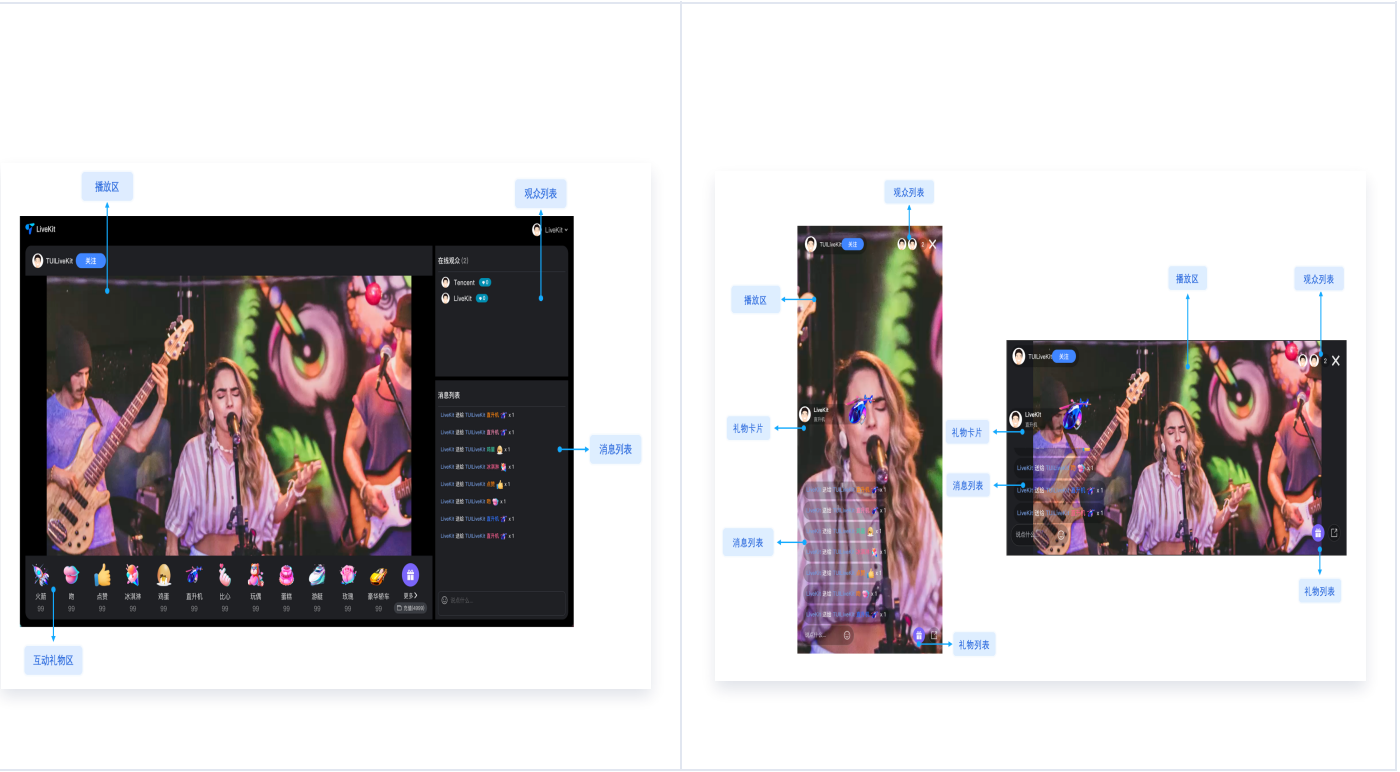
交流与反馈

如果您在接入和使用过程有任何需要或者反馈，欢迎加入我们的 TUILiveKit 技术交流平台 zhiliao ，进行技术交流和问题反馈。

Web

最近更新时间：2025-04-14 18:02:42

TUILiveKit 是腾讯云推出的在线直播 UI 组件。本文将指导您如何快速地将 TUILiveKit Web 端组件集成到您的项目中，从而为您的应用程序提供直播功能。



环境准备

- Node.js 版本：Node.js ⩾ 16.19.1（推荐使用官方 LTS 版本，npm 版本请与 node 版本匹配）。
- 现代浏览器，支持 WebRTC APIs。

集成 TUILiveKit 组件

> **说明：**
> 如果您当前尚无 Vue 项目，建议您前往 Github 下载 TUILiveKit Demo 项目，参考项目 README.md 文档运行示例工程，并根据自身需求增加业务逻辑。

如果需要在已有项目中集成，请按照以下步骤进行接入。

步骤一：安装依赖

```
npm install @tencentcloud/livekit-web-vue3 pinia --save
```

> **说明：**
> TUILiveKit npm 包提供的功能样式若无法满足您的业务需求，您可以参见 LiveKit 源码导出 方案接入 TUILiveKit 源码。

步骤二：项目工程配置

注册 pinia：TUILive 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia。项目入口文件为 src/main.ts 文件。

```
// src/main.ts 文件
import { createPinia } from 'pinia';

const app = createApp(App);
// 注册 pinia
app.use(createPinia());
app.mount('#app')
```

步骤三：引入 TUILiveKit 组件

```
<template>
    <live-main-view :user-info="basicInfo"></live-main-view>
</template>

<script setup>
import { reactive } from 'vue';
import { LiveMainView } from '@tencentcloud/livekit-web-vue3';

const basicInfo = reactive({
  userId: '',
  userName: '',
  avatarUrl: '',
});
</script>
<style>
#app {
  width: 100%;
  height: 100%;
}
</style>
```

步骤四：登录 TUILiveKit 组件

开启直播前需要调用 login 接口进行登录。获取 sdkAppId 、 userId 、 userSig 可参见 开通服务 。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';

liveRoom.login({
    // 获取 sdkAppId 可参考文档开通服务部分，
https://cloud.tencent.com/document/product/647/104842
    sdkAppId: 0,     // 用户在您业务中的唯一标示 Id
    userId: '',     // 本地开发调试可在
https://console.cloud.tencent.com/trtc/usersigtool 页面快速生成 userSig，注
意 userSig 与 userId 为一一对应关系
    userSig: '',
});
```

参数说明 这里详细介绍一下 login 函数中所需要用到的几个关键参数：

- **sdkAppId**：在 开通服务 中的最后一步中获取。
- **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**：通过对在 开通服务 中获取的 SDKSecretKey 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，并在每次使用 TUILiveKit 组件时向您的服务器请求实时计算出的 UserSig。

步骤五：开启直播

主播可以通过调用 start 接口来开启直播，观众可以调用 join 接口加入直播间。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';

const startLive = async () => {
    await liveRoom.login({
        sdkAppId: 0,
        userId: '',
        userSig: '',
    });
    await liveRoom.start('123456', {
      roomName: 'TestRoom',
      isOpenCamera: false,
      isOpenMicrophone: false,
    });
}
startLive()
```

步骤六：进入直播间

观众可以通过调用 join 接口，填写对应的 `roomId` 参数来加入主播在 步骤五 中创建的直播间。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';

const joinLive = async () => {
    await liveRoom.login({
        sdkAppId: 0,
        userId: '',
        userSig: '',
    });
    await liveRoom.join('123456', {
      isOpenCamera: false,
      isOpenMicrophone: false,
    });
}
joinLive()
```

开发环境运行

1. 执行开发环境命令。（此处以 vue3 + vite 默认项目为例，不同项目 dev 指令可能不同，请根据您自己的项目进行调整）

```
npm run dev
```

2. 根据控制台提示，在浏览器中打开页面 ，如： `http://localhost:5173/` 。
3. 体验 TUILiveKit 组件功能。

生产环境部署

1. 打包 dist 文件。

```
npm run build
```

2. 部署 dist 文件到服务器上。

> **注意：**
> 生产环境要求使用 HTTPS 域名，参见下图：

页面访问协议说明

浏览器厂商出于对用户安全、隐私等问题的考虑，限制网页在 https 协议下才能正常使用 TRTC Web SDK（WebRTC）的全部功能。为确保生产环境用户顺畅接入和体验 TRTC Web SDK 的全部功能，请使用 https 协议访问音视频应用页面。

注：本地开发可以通过 http://localhost 或者 file:// 协议进行访问。

URL域名及协议支持情况请参考如下表格：

应用场景	协议	接收（拉流）	发送（推流）	屏幕分享	备注
生产环境	https协议	支持	支持	支持	推荐
生产环境	http协议	支持	不支持	不支持	
本地开发环境	http://localhost	支持	支持	支持	推荐
本地开发环境	http://127.0.0.1	支持	支持	支持	
本地开发环境	http://[本机IP]	支持	不支持	不支持	
本地开发环境	file://	支持	支持	支持	

其他文档

- TUILiveKit 组件介绍
- TUILiveKit 开通服务
- TUILiveKit Web 快速跑通 Demo
- 客户端 API 文档
- 常见问题

交流与反馈

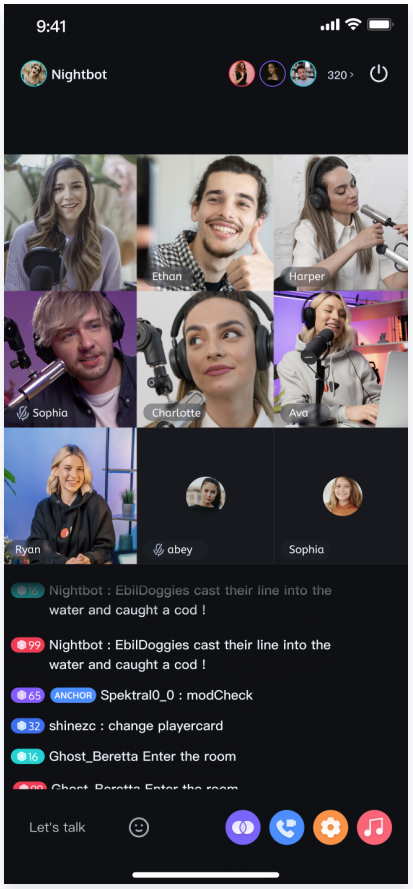
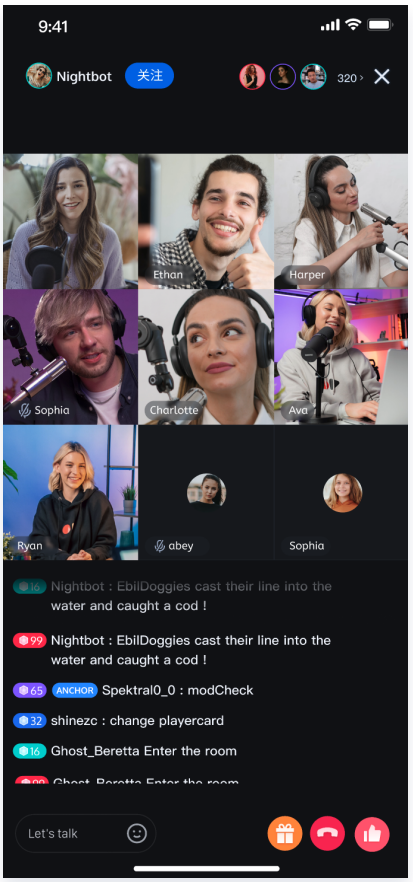
您在接入或使用过程有任何需要或者反馈，欢迎加入 腾讯云实时音视频 TRTC 技术交流群 进行技术交流和问题反馈。

版权所有：腾讯云计算（北京）有限责任公司

Flutter

最近更新时间：2025-05-23 09:49:51

本文将介绍如何在短时间内完成 TUILiveKit 组件的接入，跟随本文档，您可以在10分钟内完成接入工作，并最终得到一个包含完备 UI 界面的直播功能。

主播（视频直播）	观众（视频直播）
	

环境准备

平台	版本
Flutter	• Flutter 3.27.4 及更高版本。 • Dart 3.6.2 或更高的版本。
Android	• Android Studio 3.5及以上版本。 • Android 5.0 及以上版本的 Android 设备。
iOS	• Xcode 15.0及以上版本。 • 请确保您的项目已设置有效的开发者签名。

版权所有：腾讯云计算（北京）有限责任公司

步骤1：开通服务

请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。

步骤2：导入 TUILiveKit 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_live_uikit 插件。

```
flutter pub add tencent_live_uikit
```

步骤3：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

1. 使用 Xcode 打开您的工程，选择**项目 > Building Settings > Deployment**，将其下的 **Strip Style** 设置为 `Non-Global Symbols` ，以保留所需要的全局符号信息。
2. **可选**如果您需要在 iOS 模拟器上调试，并且您使用的 Mac 电脑使用的是 Intel 芯片，您需要在工程的 `ios/Podfile` 文件中添加以下代码：

```
target 'xxxx' do
  ......
end
......

post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
      config.build_settings['VALID_ARCHS'] = 'arm64 arm64e x86_64'
      config.build_settings['VALID_ARCHS[sdk=iphonesimulator*]'] =
'x86_64'
    end
  end
end
```

3. 由于 TUILiveKit 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>
```

完成以上添加后，在您的 **ios/Podfile** 中添加以下预处理器定义，用于启用相机与麦克风权限。

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

步骤4：设置 navigatorObservers 和 localizationsDelegates

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

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

 ......

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

 @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUILiveKitNavigatorObserver.instance],
      localizationsDelegates:
[...LiveKitLocalizations.localizationsDelegates],
      ......
    );
  }
}
```

步骤5：登录 TUILiveKit 组件

在调用 TUILiveKit 组件各项功能之前，先要执行 TUI 组件的登录。在您的项目中，建议在您的业务登录场景里或者 App 的首个启动 Activity 里，添加执行如下登录代码，它的作用是通过调用 TUICore 中的相关接口完成 TUI 组件

的登录。这个步骤异常关键，因为只有在登录成功后才能正常使用 TUILiveKit 的各项功能，故请您耐心检查相关参数是否配置正确：

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

login() async {
  await TUILogin.instance.login(
    1400000001,      // 请替换为步骤1取到的 SDKAppID
    "denny",         // 请替换为您的 UserID
    "xxxxxxxxxxx",   // 您可以在控制台中计算一个 UserSig 并填在这个位置
    TUICallback(
      onError: (code, message) {
        print("TUILogin login fail, {code:$code, message:$message}");
      },
      onSuccess: () async {
        print("TUILogin login success");
      },
    ),
  );
}
```

参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

- **SDKAppID**：在步骤2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 的第3步中获取的 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 TUILiveKit 组件时向您的服务器请求实时计算出的 UserSig。

步骤6：进入直播预览画面

注意：
请务必确保已经按照 步骤五 完成登录操作。只有 `TUILogin.login` 登录成功后才能正常进入直播预览画面。

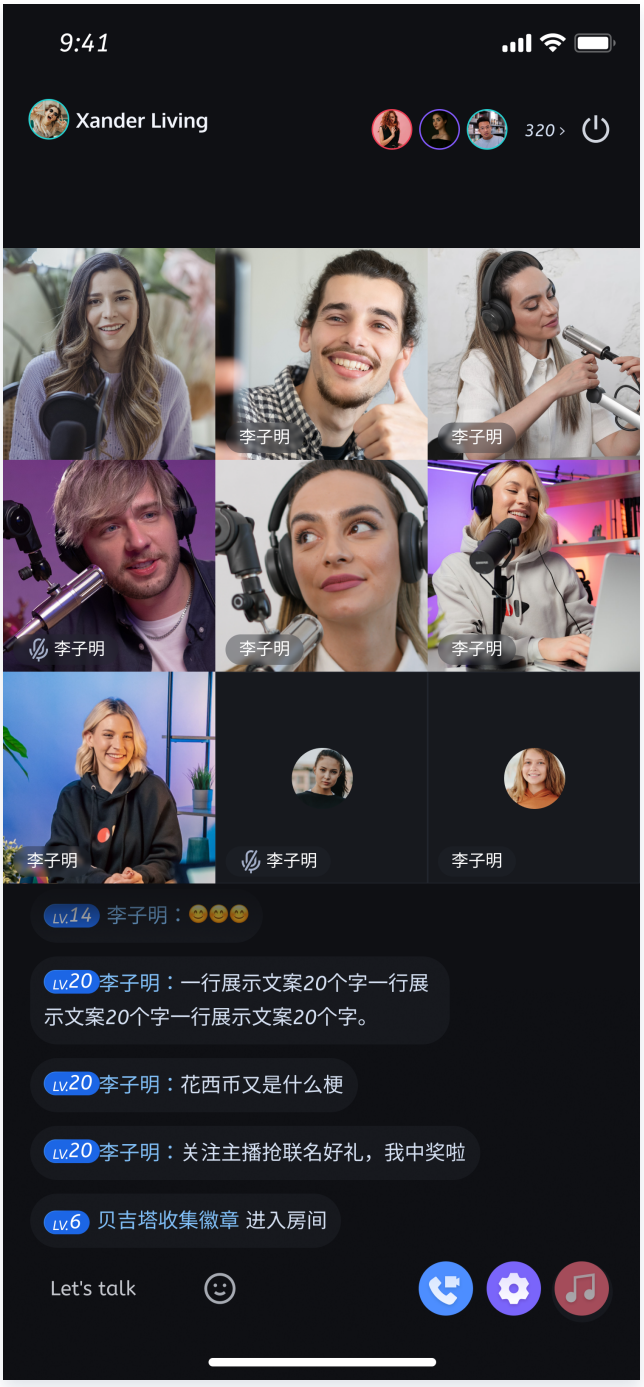
在您需要开启直播的地方（具体由您的业务决定，在其点击事件里执行），执行如下操作，拉起主播开播页面：

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

Navigator.push(context, MaterialPageRoute(
  builder: (context) {
    return TUILiveRoomAnchorWidget(
        roomId: LiveIdentityGenerator.instance.generateId(AppStore.userId,
RoomType.live));
  },
));
```



视频直播预览画面



视频直播中画面

步骤7：拉取房间列表

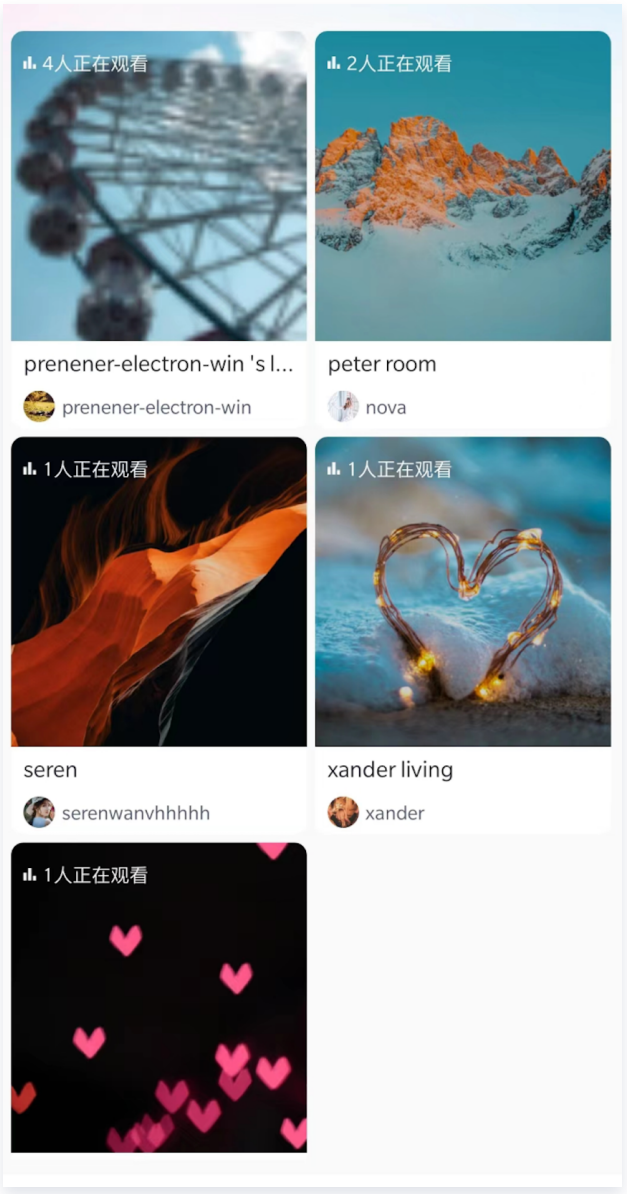
> **注意：**
> 请务必确保已经按照 步骤五 完成登录操作。只有 `TUILogin.login` 登录成功后才能正常进入直播预览画面。

在您的 widget 中，通过加载 TUILiveKit 的 LiveListWidget 组件，就可以展示房间列表。

Java

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


return Scaffold(
  body: SizedBox(
    width: _screenWidth,
    height: double.infinity,
    child: LiveListWidget(),     // 在自己的 widget树中 添加 TUILiveKit 的 房间列表组件 LiveListWidget
  ),
);
```



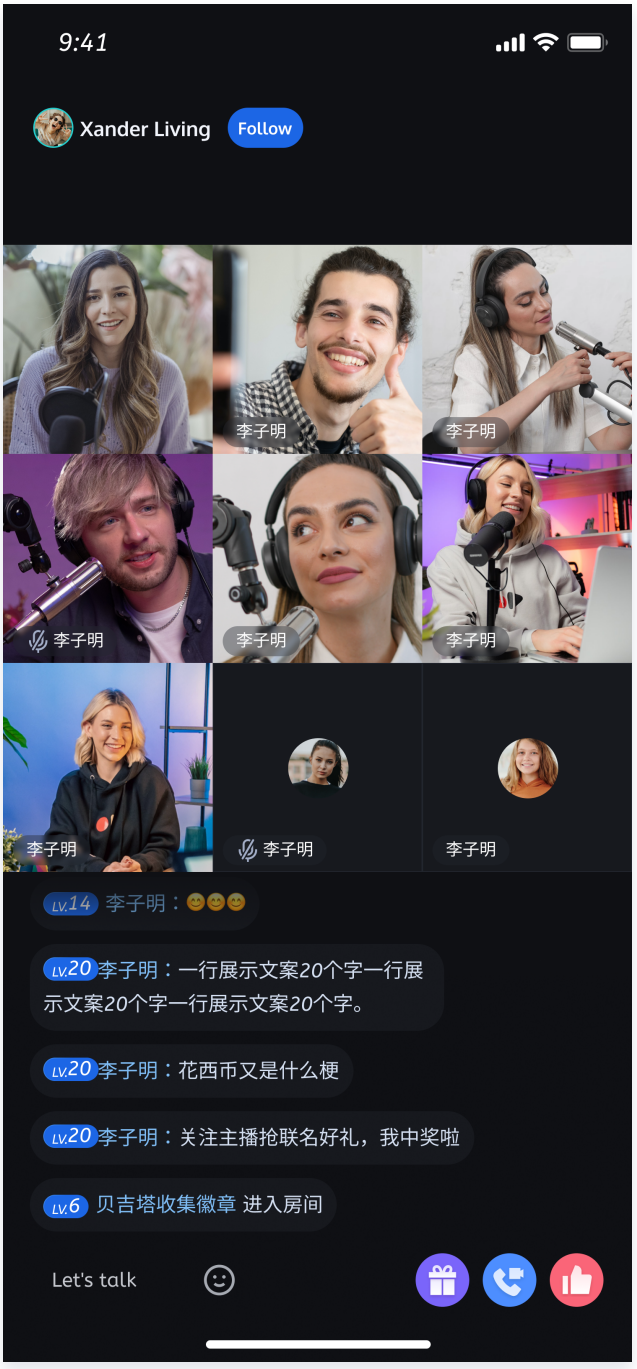
步骤8：观众进入直播间

在 步骤七 的房间列表界面，点击任意一个房间即可自动进入直播间。

版权所有：腾讯云计算（北京）有限责任公司



视频直播间



视频直播间

交流与反馈

如果您在接入和使用过程有任何需要或者反馈，欢迎加入我们的 TUILiveKit 技术交流平台 zhiliao，进行技术交流和问题反馈。

Electron

最近更新时间：2025-04-14 10:13:52

本文将指导您如何快速地将桌面端 TUILiveKit 组件集成到您的项目中，从而为您的应用程序提供直播推流功能。

环境准备

- Windows 电脑：操作系统 Windows 10 或者 11 版本。
- Node.js 版本：Node.js ⩾ 16.19.1（推荐使用官方 LTS 版本，npm 版本请与 node 版本匹配）。

步骤一：开通服务

请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。

步骤二：下载 TUILiveKit 源码

从 Github 下载 项目源码，或者执行以下命令下载：

```
git clone https://github.com/Tencent-RTC/ultra-live-electron.git
```

步骤三：集成 TUILiveKit

当 TUILiveKit 在桌面端运行时，需要创建两个 Electron 窗口，分别用于承载 TUILiveKit 的主界面和设置界面。我们将这两个窗口分别称为 TUILiveKit 主窗口和子窗口。集成后的最终效果是：在您现有的 Electron 应用中，通过点击一个按钮，即可创建并打开 TUILiveKit 主窗口，开始使用 TUILiveKit 的全部功能。

前置依赖

您的代码工程需要包含以下技术栈：

- Vue3
- Webpack
- TypeScript
- Electron

> ⚠ **注意：**
> 如果您没有符合接入条件的工程，可以参阅文档下面 常见问题，获得指引。

复制 TUILiveKit 代码到您的工程

1. **复制 TUILiveKit 组件**
   复制 `ultra-live-electron/src/TUILiveKit` 目录到您工程的 `src` 目录下。
2. **复制 TUILiveKit 窗口创建代码**

版权所有：腾讯云计算（北京）有限责任公司

复制 `ultra-live-electron/TUILiveKit.main.js` 和 `ultra-live-electron/TUILiveKit.preload.js` 文件到您工程的根目录下。

3. 复制 TUILiveKit 主窗口和子窗口页面及其页面路由定义

- 复制 `ultra-live-electron/src/views/TUILiveKitChild.vue` 和 `ultra-live-electron/src/views/TUILiveKitMain.vue` 文件到您工程的 `src/views` 目录下。
- 在您工程的页面路由文件 `src/router/index.ts` 中，新增如下两条路由配置：

```
// src/router/index.ts
import { createRouter, createWebHashHistory, RouteRecordRaw } from
'vue-router';
import HomeView from '../views/HomeView.vue';

const routes: Array<RouteRecordRaw> = [
  { // 您应用的当前路由页面，页面内有触发打开 TUILiveKit 主窗口的按钮
    path: '/',
    name: 'home',
    component: HomeView
  },
  // ********************************* TUILiveKit 相关代码开始
********************************* //
  { // TUILiveKit 主窗口页面
    path: '/tui-live-kit-main',
    name: 'tui-live-kit-main',
    component: () => import(/* webpackChunkName: "TUILiveKitMain" */
'../views/TUILiveKitMain.vue')
  },
  { // TUILiveKit 子窗口页面
    path: '/tui-live-kit-child',
    name: 'tui-live-kit-child',
    component: () => import(/* webpackChunkName: "TUILiveKitChild" */
'../views/TUILiveKitChild.vue'),
  },
  // ********************************* TUILiveKit 相关代码结束
********************************* //
];

// ********************************* TUILiveKit 相关代码开始
********************************* //
// 监听来自主进程的窗口类型通知，进行路由跳转
window.ipcRenderer.on('window-type', (event: any, type: string) => {
  console.log(`[router] window type:${type}`);
  console.log(`[router] current href:${window.location.href}`);
  router.replace({ name: `tui-live-kit-${type}` });
```

```
});
// ********************************* TUILiveKit 相关代码结束
********************************* //

const router = createRouter({
  history: createWebHashHistory(),
  routes
});

export default router;
```

4. **复制美颜相关代码、资源和配置。**

- 复制 `ultra-live-electron/public/assets` 目录和文件 `public/avatar.png` 到您工程的 `public` 目录下。
- 复制 `ultra-live-electron/scripts/prestart.js` 文件到您工程的 `scripts` 目录下，并在您工程的 `package.json` 文件中的 `scripts` 部分新增如下命令。

```
{
  "scripts": {
    "prestart": "node ./scripts/prestart.js"
  }
}
```

- 这里我们先不开启美颜功能，以上配置、代码、资源复制，可以保证工程运行不出错。美颜功能的开启参见 如何开启美颜功能 。

5. **修改 vue.config.js 配置**

在您工程的 `vue.config.js` 文件中，新增如下配置：

```
// vue.config.js
const { defineConfig } = require('@vue/cli-service')
// ********************************* TUILiveKit 相关代码开始
********************************* //
const os = require("os");
const isProduction = process.env.NODE_ENV === "production";
const platform = os.platform();
// ********************************* TUILiveKit 相关代码结束
********************************* //

module.exports = defineConfig({
  transpileDependencies: true,
```

```
// ********************************** TUILiveKit 相关代码开始
********************************** //
  publicPath: "./",
  configureWebpack: {
    devtool: isProduction ? "source-map" : "inline-source-map",
    target: "electron-renderer",
    module: {
      rules: [
        {
          test: /\.node$/,
          loader: "native-ext-loader",
          options: {
            rewritePath: isProduction
              ? platform === "win32"
                ? "./resources"
                : "../Resources"
              : "./node_modules/trtc-electron-sdk/build/Release",
          },
        },
      ],
    },
  }
// ********************************** TUILiveKit 相关代码结束
********************************** //
});
```

6. 复制 `src/debug` 目录，配置 `SDKAppID` 和 `SDKSecretKey` 。

   复制 `ultra-live-electron/src/debug` 文件夹到您工程的 `src` 目录下，打开复制过来的 `basic-info-config.js` 文件，填入 步骤一：开通服务 时获取的 `SDKAppID` 和 `SDKSecretKey` 。

```
// basic-info-config.js
/**
 * 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。
 */
export const SDKAppID = 0;

/**
 * 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
 */
export const SDKSecretKey = '';
```

7. 启用 pinia 状态管理并引入主题样式

打开您工程的 `src/main.ts` 文件，增加 pinia 状态管理支持：

```
// main.ts
import { createApp } from 'vue';
// ********************************** TUILiveKit 相关代码开始
********************************** //
import { createPinia } from 'pinia';
import "./TUILiveKit/assets/theme.css";
// ********************************** TUILiveKit 相关代码结束
********************************** //
import App from './App.vue'
import router from './router'

createApp(App)
// ********************************** TUILiveKit 相关代码开始
********************************** //
  .use(createPinia())
// ********************************** TUILiveKit 相关代码结束
********************************** //
  .use(router)
  .mount('#app');
```

8. 修改 global.d.ts

在您工程的 `src/global.d.ts` 文件中，针对 Window 类型增加以下属性声明：

```
// src/global.d.ts
export {}
declare global {
  interface Window {
```

```
// ********************************* TUILiveKit 相关代码开始
********************************* //
    ipcRenderer: any;
    nativeWindowHandle: Uint8Array;
    mainWindowPort: MessagePort | null;
  }
// ********************************* TUILiveKit 相关代码结束
********************************* //
}
```

增加入口，打开 TUILiveKit 主窗口

Vue 组件

在您项目的某个页面中，新增一个按钮，点击后，通知 Electron 主进程打开 TUILiveKit 主窗口。如下所示，是我们实现的示例代码。

```
// HomeView.vue
<template>
  <div class="home">
    <button @click="openTUILiveKit">开始直播</button>
  </div>
</template>

<script setup lang="ts">
import { ref } from 'vue';
import type { Ref } from 'vue';
import { getBasicInfo } from '../debug/basic-info-config';

const isOpen:Ref<boolean> = ref(false); // 记录状态，避免重复打开
TUILiveKit 主窗口

const openTUILiveKit = async () => {
  if (!isOpen.value) {
    const currentUserInfo = await getBasicInfo();
    if (currentUserInfo) {
      window.ipcRenderer.send('openTUILiveKit', {
        userInfo: currentUserInfo // 注意打开 TUILiveKit 主窗口需要带上用
户信息作为参数
      });
      isOpen.value = true;
    } else {
      console.error('Error: cannot get current user info');
```

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

Electron preload 脚本

在您 Electron 工程的 preload 脚本中，将 Electron 的 `ipcRenderer` 对象放到全局对象 `window` 上，方便在 vue 组件和 JavaScript/TypeScript 脚本中，与 Electron 主进程通信。

```
// preload.js
const { ipcRenderer } = require("electron");

// Enable `ipcRenderer` can be used in vue and Javascript module
window.ipcRenderer = ipcRenderer;
```

Eletron 主进程

在您 Electron 工程的主进程脚本中，接收来自 vue 组件发送的 `openTUILiveKit` 请求，打开 TUILiveKit 主窗口。

```
// main.js
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');
// ********************************** TUILiveKit 相关代码开始
********************************** //
const { TUILiveKitMain } = require("./TUILiveKit.main");
// ********************************** TUILiveKit 相关代码结束
********************************** //

let mainWindow = null;
const createWindow = () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
// ********************************** TUILiveKit 相关代码开始
********************************** //
    webPreferences: {
```

```
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: true,
      contextIsolation: false,
    }
// ********************************* TUILiveKit 相关代码结束
********************************* //
  });

// ********************************* TUILiveKit 相关代码开始
********************************* //
  bindIPCMainEvent();
// ********************************* TUILiveKit 相关代码结束
********************************* //

  if (app.isPackaged) {
    mainWindow.loadFile("dist/index.html");
  } else {
    mainWindow.loadURL('http://localhost:8080');
  }
}

// ********************************* TUILiveKit 相关代码开始
********************************* //
function bindIPCMainEvent() {
  ipcMain.on("openTUILiveKit", (event, args) => {
    console.log(`[main] open live kit`, args);
    TUILiveKitMain.open(args); // Open TUILiveKit main window
  });
}
// ********************************* TUILiveKit 相关代码结束
********************************* //


app.whenReady().then(() => {
  createWindow();
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  });
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
});
```

package.json

在您工程的 package.json 文件中，新增如下配置，支持以开发模式启动 Electron 应用程序。

```
{
  "main": "main.js",
  "scripts": {
      "start": "electron ."
  }
}
```

安装依赖

```
npm install --save pinia movable-resizable-js
npm install --save @tencentcloud/tuiroom-engine-electron
# TUILiveKit 直播间引擎
npm install --save trtc-electron-plugin-xmagic
# TUILiveKit 集成的美颜插件
npm install --save-dev native-ext-loader electron-devtools-installer
electron-builder
```

步骤四：开发模式启动

1. 进入您的项目目录，打开命令行终端 `cmd.exe` ，执行以下命令，以开发模式启动 Web 项目。

```
npm run serve
```

如果您在启动时遇到了 error Component name "Index" should always be multi-word vue/multi-word-component-names 报错，表示您的项目工程 eslint 配置与 TUILiveKit 有差异，打开您项目的 .eslintrc.js 文件或者在 package.json 文件的 eslintConfig 部分，增加如下所示代码中的 vue/multi-word-component-names 校验规则，再重新执行 “npm run serve” 命令。

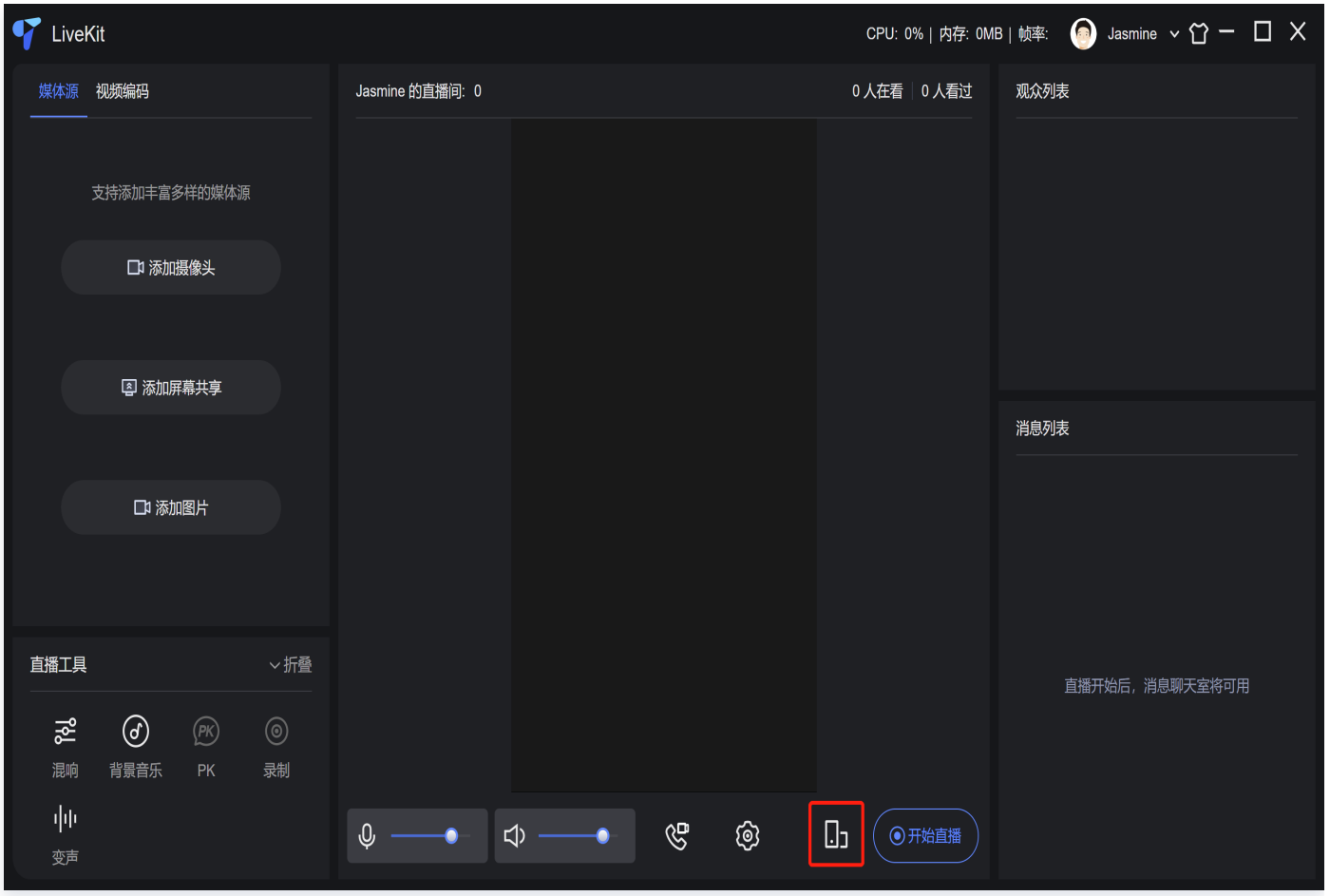
```
// .eslintrc.js
module.exports = {
  root: true,
  env: {
    node: true
  },
```

```
    'extends': [
      'plugin:vue/vue3-essential',
      'eslint:recommended',
      '@vue/typescript/recommended'
    ],
    parserOptions: {
      ecmaVersion: 2020
    },
    rules: {
      'no-console': process.env.NODE_ENV === 'production' ? 'warn' :
'off',
      'no-debugger': process.env.NODE_ENV === 'production' ? 'warn' :
'off',
// ********************************* TUILiveKit 相关代码开始
********************************* //
      "vue/multi-word-component-names": process.env.NODE_ENV ===
'production' ? 'warn' : 'off',
// ********************************* TUILiveKit 相关代码结束
********************************* //
    }
}
```

2. 进入您的项目目录，再打开一个新的命令行终端 `cmd.exe` ，执行以下命令，以开发模式启动 Electron 应用程序。

```
npm run start
```

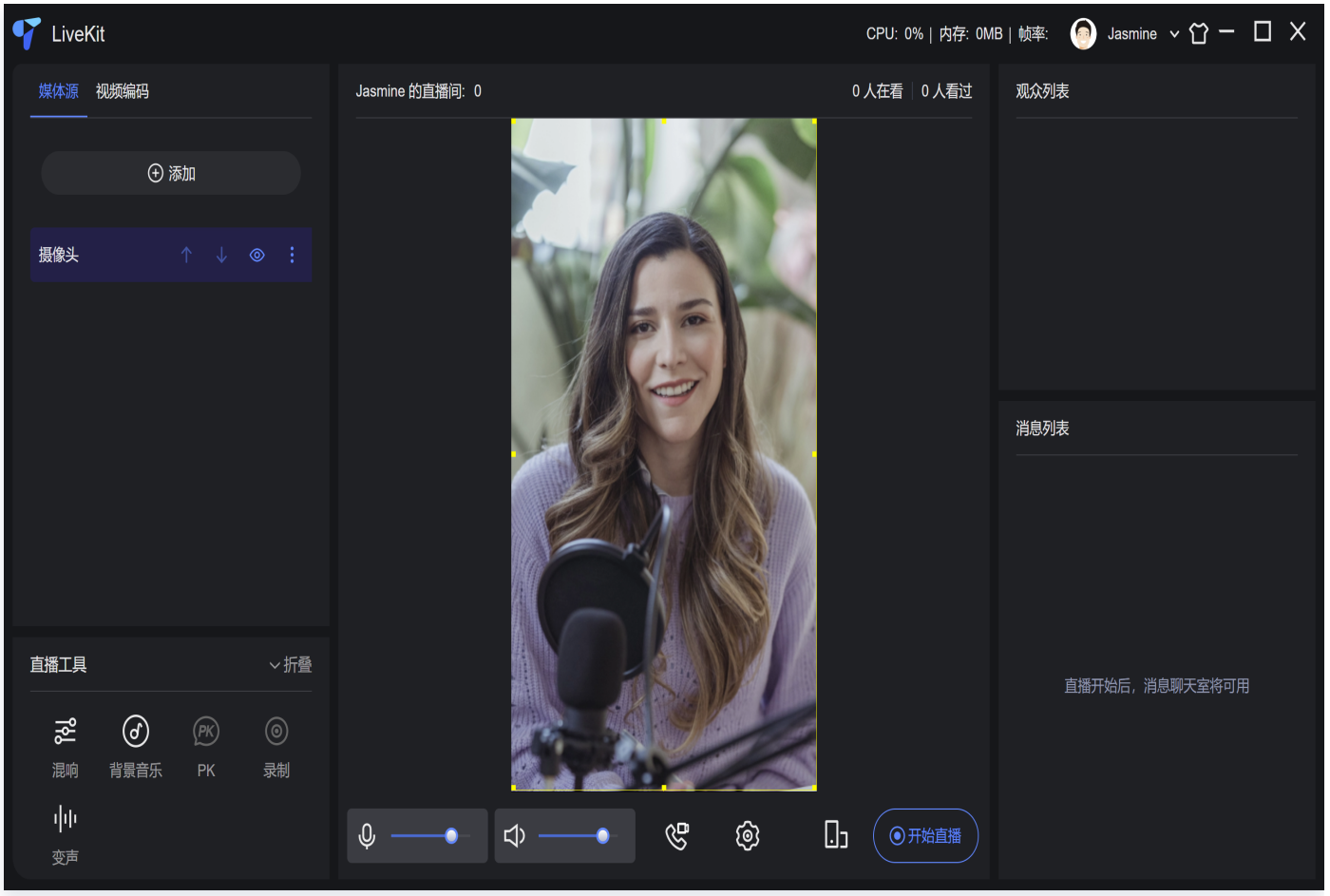
启动成功后，点击您在应用界面添加的 “开始直播” 按钮，就可以打开 TUILiveKit 主窗口，打开后效果如下图：



步骤五：开启一场直播

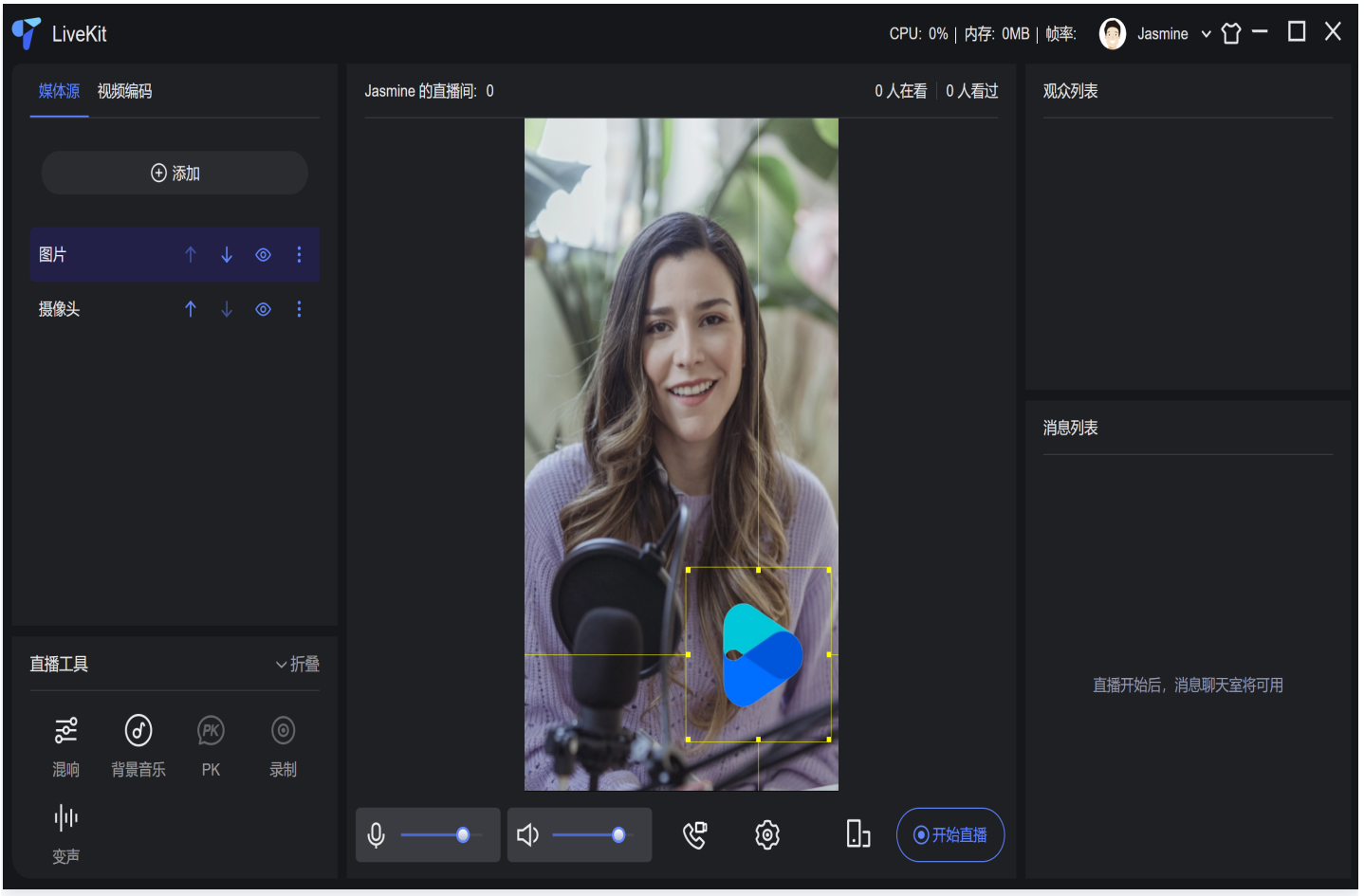
1. 添加摄像头

在开播前，需要先添加画面源，目前支持：摄像头、屏幕分享、窗口分享和图片，如下图是添加一个摄像头后的效果。



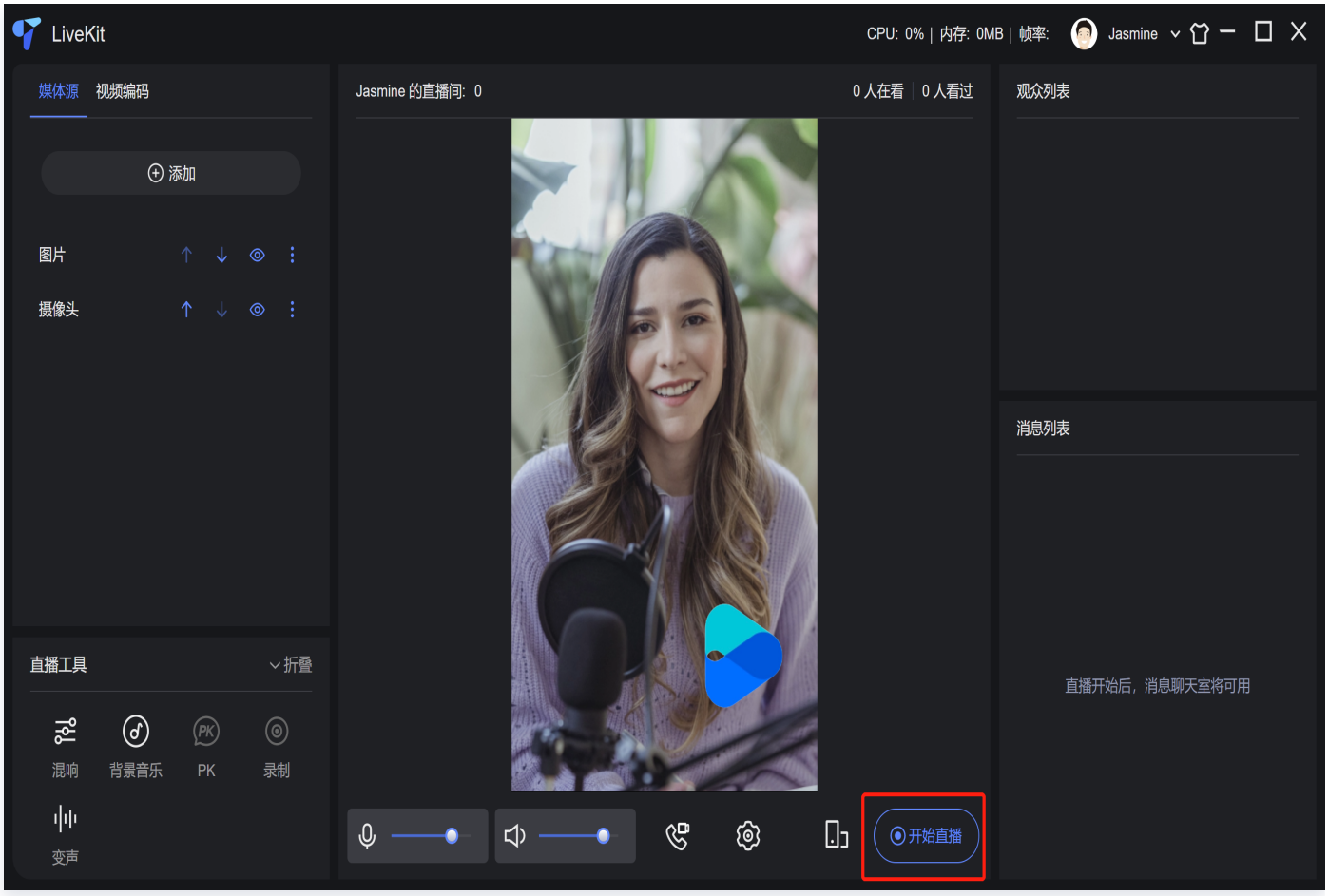
2. 添加 Logo 图片

如果您在直播时，需要添加自己的品牌 Logo，可以添加 Logo 图片。如下图，是添加一个背景透明的 Logo 图片后的效果，新添加的图片周围会有黄色边框，表示当前处于选中状态。处于选中状态的多媒体源，可以使用鼠标移动位置、调整大小，使用右键菜单对多媒体源进行旋转、修改显示层级等。



3. 开始直播

单击**开始直播**，将开启直播推流，成功开播后，**开始直播**按钮会变成**结束直播**，单击可结束直播推流。



4. **观看直播**

   桌面端 TUILiveKit 只支持直播推流，如果您想观看，请接入我们 TUILiveKit 手机端，通过手机端观看：安卓、iOS。

步骤六：构建安装包

1. 复制 `ultra-live-electron/electron-builder.json5` 文件到您项目的根目录下，文件中的 `productName` 和 `appId` 请改成您应用的相应信息。
2. 在您工程的 package.json 文件中新增以下命令，支持构建应用程序安装包。

```
{
  "scripts": {
    "build:win64": "electron-builder --win --x64",
    "pack:win64": "npm run build && npm run build:win64"
  }
}
```

3. 构建应用程序安装包。

   进入您的项目目录，打开命令行终端 `cmd.exe`，执行以下命令即可启动构建，构建好的安装包在您工程的 `release` 目录下。

```
npm run pack:win64
```

4. 安装构建的应用程序安装包，并运行。

常见问题

没有符合接入条件的项目工程，如何解决?

根据您的具体情况，建议采用不同的途径：

- 如果您还没有项目工程，建议您直接使用我们开源的 Github 项目 为基础，克隆一份作为您的项目工程。
- 如果您的项目与我们要求的接入条件不一致，可以参照我们开源的 模板工程: trtc-electron-template-vue3-webpack ，对照调整您的项目。
- 如果您的项目是 JavaScript 工程，不支持 TypeScript ，可以参见常见问题： 如何在 JavaScript 工程中集成 。

是否支持 Vite 构建、接入?

暂不支持，如果您有诉求，可以通过下方 交流与反馈 章节提供的渠道，寻求帮助。

如何在 JavaScript 工程中集成?

JavaScript 工程无法直接集成 TUILiveKit，需要改造您的工程，使其支持 TypeScript。步骤如下：

1. 安装依赖

```
npm install --save-dev typescript@4.5.5 @typescript-eslint/eslint-plugin@5.4.0 @typescript-eslint/parser@5.4.0 @vue/cli-plugin-typescript@5.0.0 @vue/eslint-config-typescript@9.1.0
```

2. 复制 `ultra-live-electron/tsconfig.json` 文件到您工程的根目录下。
3. 复制 `ultra-live-electron/src/global.d.ts` 文件到您工程的根目录下。

如何开启美颜功能?

TUILiveKit 使用的美颜功能基于 腾讯特效 SDK 实现，如需使用，需要 购买相应服务 ，获取 `licenseURL` 和 `licenseKey` ，填入 `src/TUILiveKit/utils/beauty.ts` 文件即可。

> ⚠ **注意：**
> 生产项目需要通过调用后台服务获取 `licenseURL` 和 `licenseKey` ，此处写入 JavaScript 文件可以快速启用，但是 `licenseURL` 和 `licenseKey` 泄露风险极高，仅适合快速接入、体验。

```
// beauty.ts
export const XmagicLicense = {
```

```
    licenseURL: "",
    licenseKey: "",
};
```

交流与反馈

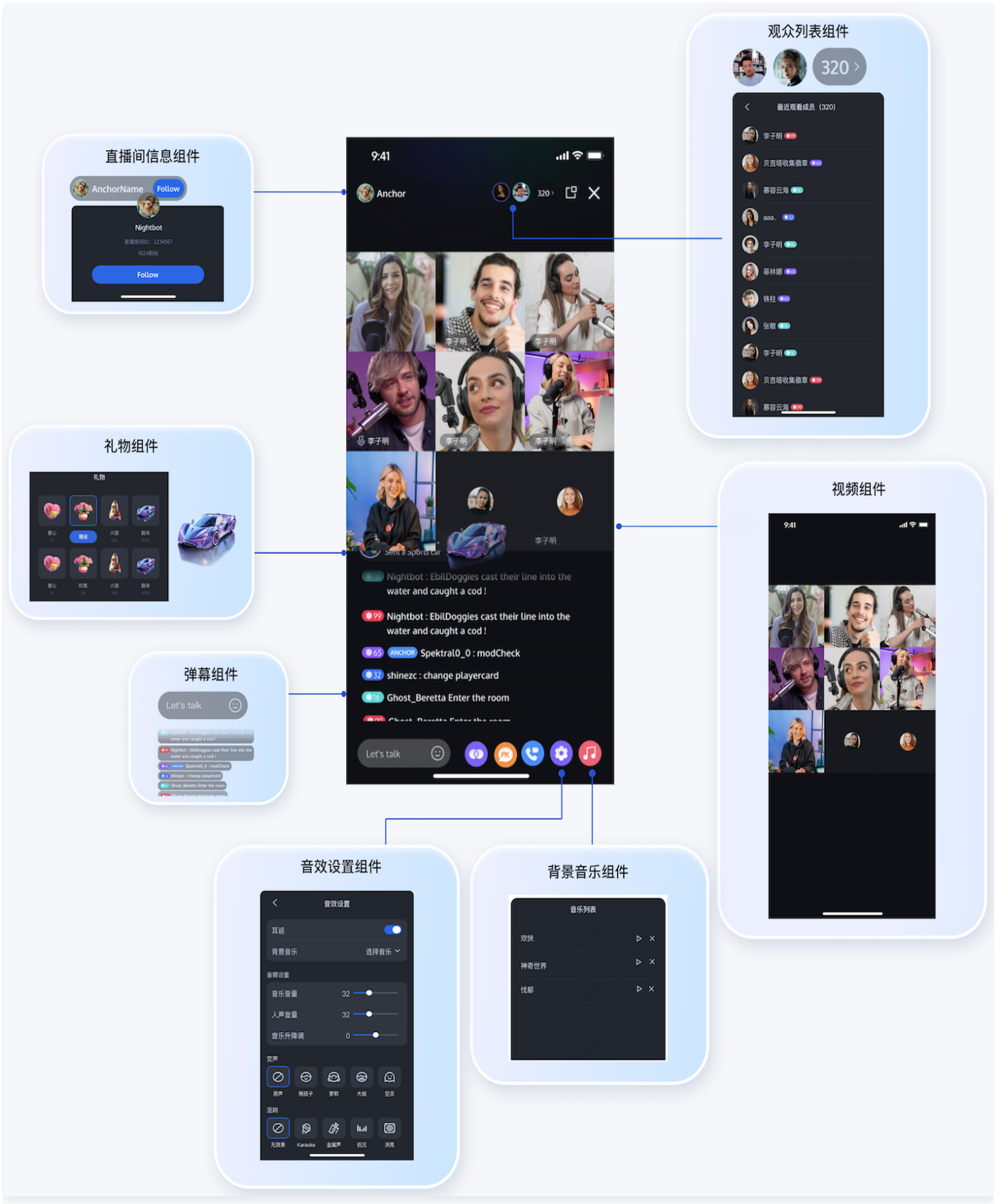
如果您在接入和使用过程有任何需要或者反馈，欢迎加入我们的 TUILiveKit 技术交流平台 zhiliao，进行技术交流和问题反馈。

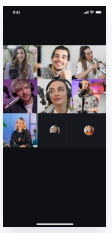
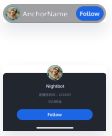
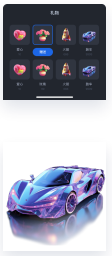
界面定制（TUILiveKit）Android&iOS

最近更新时间：2025-05-07 16:49:52

TUILiveKit 组件集合介绍

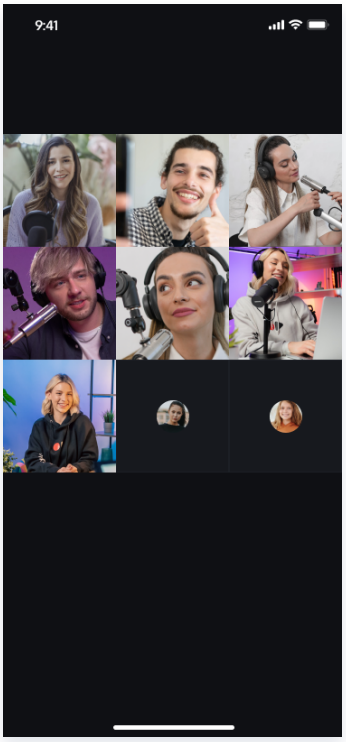
TUILiveKit 是基于腾讯云 RoomEngine SDK 实现直播场景下一系列的 UI 组件库，它提供了一些通用的 UI 组件，包含音视频核心组件、直播间信息组件、观众列表组件、弹幕组件、礼物组件等。
基于这些 UI 组件您可以像搭积木一样快速搭建起自己的直播业务逻辑。


观众列表组件
320
最近观看成员（320）
直播间信息组件
AnchorName
Follow
Nightbot
Follow
9:41
Anchor
320
礼物组件
礼物
视频组件
9:41
弹幕组件
Let's talk
Nightbot : EbilDoggies cast their line into the water and caught a cod !
Nightbot : EbilDoggies cast their line into the water and caught a cod !
Spektral0_0 : modCheck
shinezc : change playercard
Ghost_Beretta Enter the room
Let's talk
音效设置组件
音效设置
耳返
背景音乐
选择音乐
音频设置
音乐音量
32
人声音量
32
音乐升降调
0
变声
混响
背景音乐组件
音乐列表
欢快
神奇世界
忧郁

组件英文名	组件中文名	作用	缩略图
LiveCoreView	直播核心组件	直播核心组件提供了 开播前画面预览、开启视频直播、关闭视频直播，直播间内和观众连线，跨房和其他主播连线等丰富的 API。	
LiveInfoView	直播间信息组件	直播间信息组件会展示**主播信息**和**直播间信息**，并且支持观众最自己喜欢的主播进行关注、取消关注等功能。	
AudienceListView	观众列表组件	观众列表组件会展示直播间最新进房的前100位观众和观众人数，您可以在您的直播界面展示观众列表的简化版视图，当简化版试图被点击时，可以拉起观众列表详情视图。	
BarrageView	互动弹幕组件	• 互动弹幕组件支持主播和观众相互发送文本消息或者表情消息。 • 互动弹幕组件提供了 BarrageInputView (弹幕发送 View) 和 BarrageStreamView (弹幕显示 View)。	
GiftView	互动礼物组件	• 互动礼物组件支持直播间的观众向自己欣赏的直播主播送上虚拟礼物，其中礼物支持动效。 • 互动礼物组件提供了 GiftButton（礼物发送按钮）和 GiftPlayView (礼物播放 View)。	
MusicView	背景音乐组件	音乐列表组件允许您为直播间添加背景音乐。您还可以暂停、重新播放或删除背景音乐。	
AudioEffectView	音效设置组件	音效设置组件提供了设置您的直播音量、背景音乐音量、变声等效果。	

LiveCoreView 组件介绍

LiveCoreView 是我们为视频直播 UIKit 开发的一个基础控件。这个核心控件提供了丰富的 API，包括开播前画面预览、开启视频直播、关闭视频直播、直播间内与观众连线以及与其他主播跨房连线等功能。您只需调用 LiveCoreView 组件的几行 API，即可实现以下直播画面。
具体接入 LiveCoreView 组件详见 LiveCoreView 接入文档。

主播直播中	主播连线中-九宫格布局	主播连线中-浮窗布局
		

重点组件介绍

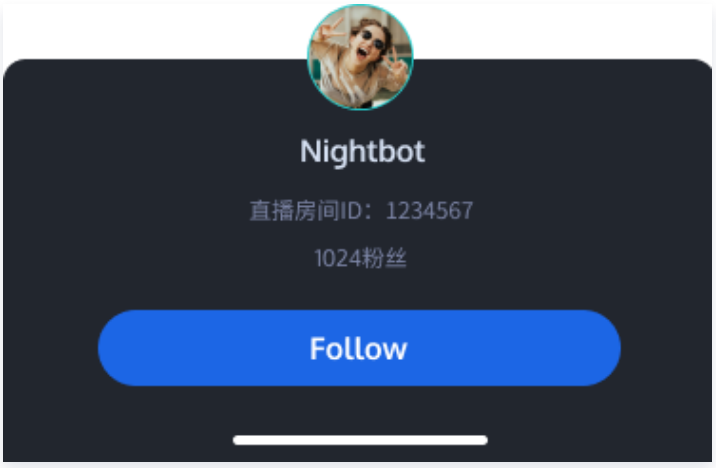
LiveInfoView 组件

LiveInfoView（直播间信息组件）中展示了主播和直播间的信息，默认情况下展示主播信息视图。当您点击主播信息视图时，将显示直播间信息的详细视图。

- 主播信息视图：



- 直播间信息视图：

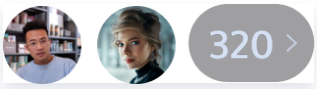


- 直播间信息组件即将上线，敬请期待。

AudienceListView 组件

AudienceListView（观众列表组件）展示了直播间最新进入的前100位观众及观众总人数。默认情况下展示观众列表视图，点击后将展开显示观众列表的详细视图。

- 观众列表视图



- 观众列表详情

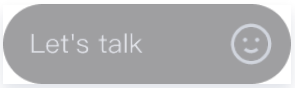


- 观众列表组件即将上线，敬请期待。

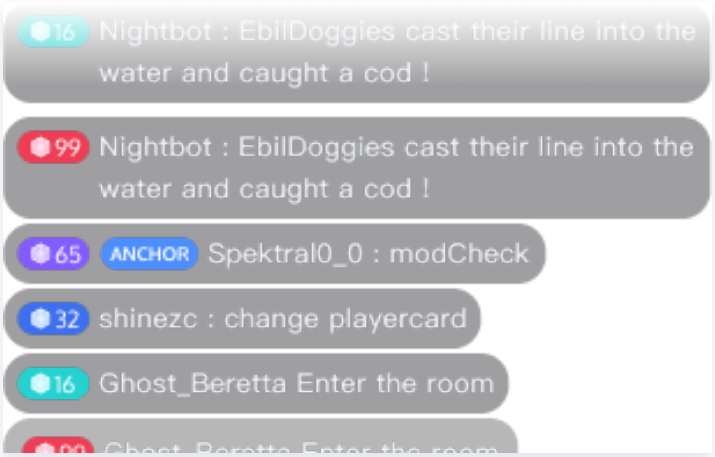
BarrageView 组件

BarrageView（互动弹幕组件）是一项重要的实时交流工具，支持多种互动方式。用户可以在弹幕中输入表情，增强消息的趣味性，从而使互动体验更加愉快和生动。通过这一功能，观众能够在直播过程中与主播和其他观众进行更为丰富的交流，提升整体的参与感和乐趣。您可以在您想要发送弹幕的布局上添加弹幕发送组件，并在想要显示弹幕的布局上添加弹幕显示组件。

- 弹幕发送组件



- 弹幕显示组件



- 如果您想接入弹幕组件请参见 互动弹幕 组件接入文档。

GiftView 组件

GiftView（互动礼物组件）是一个虚拟礼物互动平台，旨在为用户的社交体验增添更多乐趣。通过这个组件，用户可以向自己喜欢的直播主播送出虚拟礼物，以此表达他们的赞赏、喜爱和支持。互动礼物组件支持设置**礼物素材、余额显示、普通礼物播放与全屏礼物播放、充值按钮**等。

- 礼物发送组件



- 礼物显示组件



- 如果您想接入礼物组件请参见 互动礼物 组件接入文档。

MusicView 组件

MusicView（音乐列表组件）能够为您的直播间添加背景音乐。您只需在需要背景音乐时展示我们的音乐列表组件，随后便可在我们提供的音乐组件上播放您喜欢的音乐。

- 音乐列表组件



- 背景音乐组件即将上线，敬请期待。

AudioEffectView 组件

AudioEffectView（音效设置组件）允许您设置直播音量、背景音乐音量和变声效果。只需在需要设置这些效果时展示音效设置组件，便可轻松调整至所需效果。

- 音效设置组件



- 音效设置组件即将上线，敬请期待。

Web

最近更新时间：2025-05-07 16:49:52

本文将详细介绍如何通过导出源码定制 TUILiveKit 的用户界面。

方案一：TUILiveKit 源码导出修改

您可以直接修改我们提供的 UI 源代码，以便根据需求调整 TUILiveKit 的用户界面。

步骤一：导出 TUILiveKit 源码

Vue3

1. 执行源码导出脚本，默认拷贝路径为 `./src/components/TUILive`

```
  node ./node_modules/@tencentcloud/livekit-web-vue3/scripts/eject.js
```

2. 根据脚本提示确认是否要将 TUILiveKit 源码拷贝到 `./src/components/TUILive` 目录。如您需要自定义拷贝目录请输入 'y', 否则输入'n'。

```
fengch@FengChdeMacBook-Pro web-vite-vue3 % node node_modules/@tencentcloud/livekit-web-vue3/scripts/eject.js
This script copies the TUILiveKit source code to the ./src/components/TUILive path, do you want to change the copy path? (y|n)y
Please enter your copy path:
【SUCCESS】The TUILiveKit source code has been successfully copied to the ./src/components/TUILive path!
fengch@FengChdeMacBook-Pro web-vite-vue3 %
```

3. 源码导出后，在您指定的项目路径中会新增 TUILiveKit 源码。此时，您需要手动将 liveMainView 组件，liveRoom 对象的引用从 npm 包地址更改为 TUILive 源码的相对路径地址。

```
- import { liveMainView, liveRoom } from '@tencentcloud/livekit-web-vue3';
// 替换引用路径为 TUILiveKit 源码的真实路径
+ import { liveMainView, liveRoom } from './src/components/TUILive/index.ts';
```

步骤二：配置 UIKIT 源码开发环境

配置 Vue3 + Vite + TS 开发环境

1. 安装开发环境依赖

```
npm install typescript -S -D
```

2. **注册 Pinia**

   TUILive 使用 Pinia 进行房间数据管理，您需要在项目入口文件中注册 Pinia，项目入口文件为 `src/main.ts` 文件。

```
// src/main.ts 文件
import { createPinia } from 'pinia';

const app = createApp(App);
// 注册 pinia
app.use(createPinia());
app.mount('#app');
```

3. **配置 esLint 校验**

   如果您不希望 TUILiveKit 组件的 esLint 规则和您本地的规则出现冲突导致运行报错，您可以在 `.eslintignore` 中添加忽略 TUILive 文件夹。

```
// 请添加 TUILive 源码真实路径
src/components/TUILive
```

   如果您在 build 项目的过程，仍出现 ts 报错，可以检查该工程的 `package.json` 文件，删除对应 build 指令里的 `vue-tsc` 部分，如下：

```
// package.json
{
    "scripts": {
        // "build": "vue-tsc --noEmit --skipLibCheck && vite
build",
        "build": "vite build"
    }
}
```

4. **此时您可以运行项目查看源码导入效果**

```
npm run dev
```

步骤三：根据需求修改源码

1. 替换图标

您可以直接修改 `/TUILive/components/common/icons` 文件夹下的图标组件，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。

2. 调整 UI 布局

您可以通过修改 `TUILive/components/` 路径下的不同组件来调整 TUILive 界面的 UI 布局

```
- components/
    - Chat                聊天
    - common              公共组件
    - ManageMember        成员管理
    - RoomContent         Room 视频
    - RoomFooter          Room 页面 Footer部分
    - RoomHeader          Room 页面 Header部分
    - RoomHome            Home 页面
    - RoomInvite          邀请成员
    - RoomLogin           登录页面
    - RoomMore            更多
    - RoomSetting         设置
```

方案二：自实现 UI 方法

TUILiveKit 的整体功能是基于 RTC Room Engine SDK 实现的，您可以完全基于 RTC Room Engine SDK 实现一套自己的 UI 界面。详情可见 RTC Room Engine SDK 。

Flutter

最近更新时间：2025-05-23 09:49:51

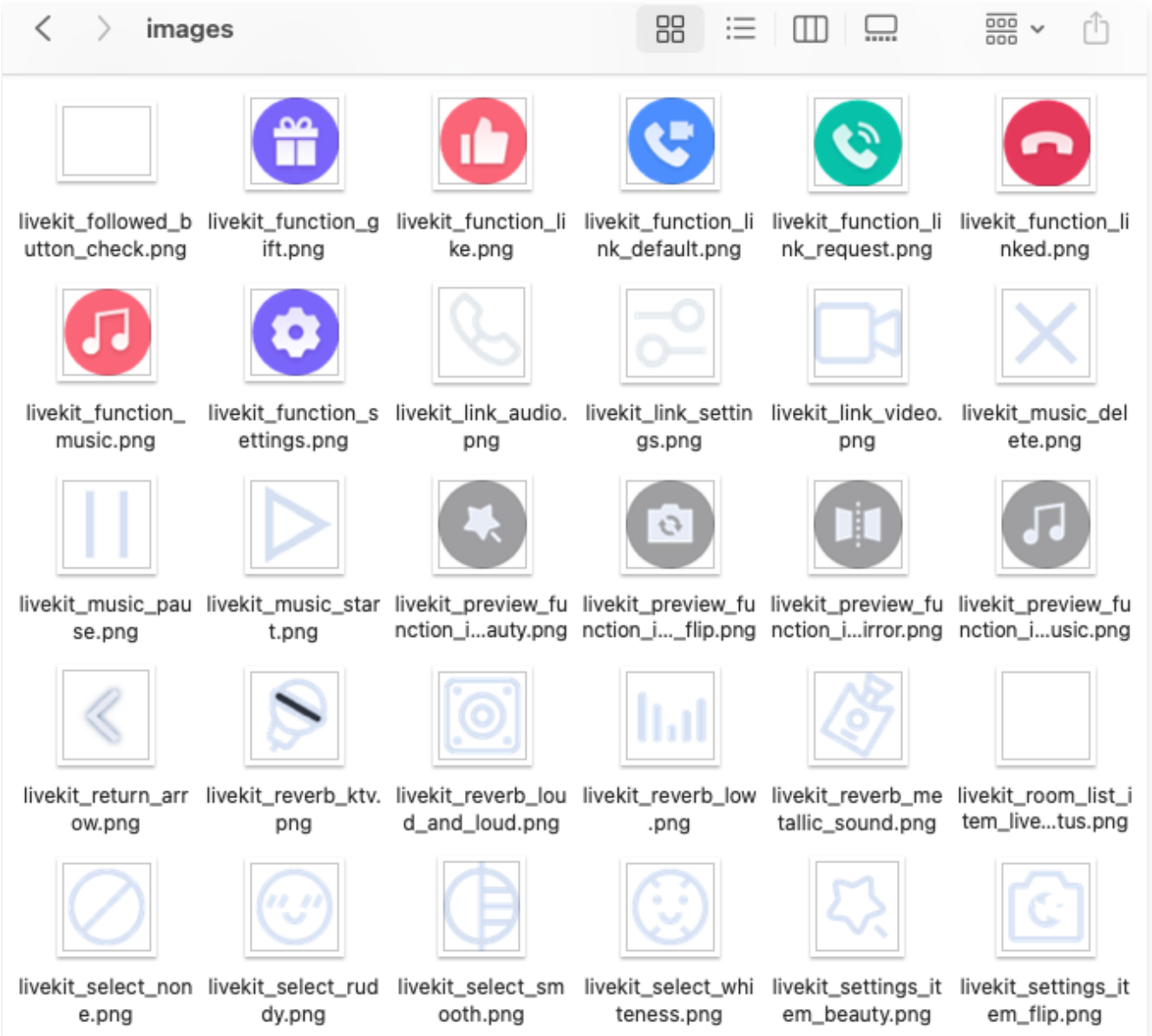
本文将介绍如何定制 TUILiveKit 的用户界面，我们提供了三个方案供您选择：界面微调方案、自定义部分 UI 方案、自定义全部 UI 方案。

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 TUILiveKit 的用户界面进行调整，TUILiveKit 的界面源代码位于 Github 中的 Flutter 文件夹下面：

替换图标

您可以直接替换 `TUILiveKit/Flutter/livekit/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。



替换文案

您可以通过修改 `TUILiveKit/Flutter/livekit/lib/common/language/i10n/` 目录下的 `livekit_en.arb` 和 `livekit_zh.arb` 文件来修改TUILiveKit 界面的字符串内容。

```
// 修改完字符串之后，需要在当前功能目录下执行下面的命令让其生效
flutter gen-l10n
```

方案二：自定义部分 UI 方案

TUILiveKit 的 UI 主要分为两部分，公共的 UI 组件 和 非公共的 UI 组件，您可以通过修改文件来适配您自己 UI 风格，这里对我们代码中的 UI 介绍如下：

公共的 UI 组件

文件位置： TUILiveKit / Flutter / livekit / lib / component

```
component
  ├── audience_list                                  // 主播推流页面和观众拉流页面右上角的 观众列表UI目录
  |    ├── audience_list_panel_widget.dart           // 观众列表 被点击，弹出的观众列表面板视图
  |    └── audience_list_widget.dart                 // 观众列表 视图
  ├── audioe_ffect                                   // 主播推流页面功能区的音效设置面板目录
  |    ├── service                                   // 音效设置面板 的服务层目录
  |        ├── audio_effect_service.dart             // 音效设置面板 的服务层，封装了音效设置相关的api
  |    ├── store                                     // 音效设置面板 的数据层目录
  |        ├── audio_effect_state_factory.dart       // 音效设置面板 生成数据层数据的工厂类实现
  |        └── audio_effect_state.dart               // 音效设置面板 数据的具体封装类
  |    └── widget                                    // 音效设置面板 的视图层目录
  |        ├── audio_effect_panel_widget.dart        // 音效设置面板 的视图具体实现
  |        ├── change_voice_widget.dart              // 音效设置面板 视图中的变声视图
  |        └── reverb_widget.dart                    // 音效设置面板 视图中的混响视图
  ├── dashboard                                      // 仪表盘视图的目录
  |    ├── anchor_dashboard_widget.dart              // 主播结束直播后，显示主播仪表盘视图的实现
```

```
    |   └── audience_dashboard_widget.dart          // 观众收到主播结束直播后，
显示观众端仪表盘视图的实现
    ├── live_info                                   // 直播间信息面板的目录
    |   ├── live_info_detail_widget.dart            // 直播间信息面板 被点击后，
弹出的直播间信息详情页面板
    |   └── live_info_widget.dart                   // 直播间信息面板 的视图实现
    ├── music                                       // 音乐面板功能 实现的目录
    |   ├── service                                 // 音乐设置面板 的服务层目录
    |       ├── music_service.dart                  // 音乐面板服务层的具体实
现，封装了音乐播放相关的 api
    |   ├── store                                   // 音乐设置面板 的数据层目录
    |       ├── music_state_factory.dart            // 音乐设置面板 生成数据层数
据的工厂类实现
    |       └── music_state.dart                    // 音乐设置面板 数据的具体封
装类
    |   └── widget                                  // 音乐设置面板 的视图层目录
    |       └── music_panel_widget.dart             // 音乐设置面板 的视图层的具
体实现
    ├── preview                                     // 主播开播预览页面，直播间
信息编辑区域的视图目录
    |   ├── live_cover_select_widget.dart           // 直播间信息编辑区域的直播
封面选择的视图
    |   └── live_info_edit_widget.dart              // 直播间信息编辑区域的视图
实现
    └── room_list                                   // 直播间列表组件的实现目录
        ├── service                                 // 直播间列表组件 的服务层目
录
            └── room_list_service.dart              // 直播间列表组件服务层的具
体实现，封装了直播间列表相关的 api
        ├── store                                   // 直播间列表组件 的数据层目
录
            └── room_list_state.dart                // 直播间列表组件 数据的具体
封装类
        └── widget                                  // 直播间列表组件 的视图层目
录
            └── room_list_widget.dart               // 直播间列表组件 直播间列表
视图的实现
```

非公共的 UI 组件

文件位置： TUILiveKit / Flutter / lib / live_stream / widget / live_room

```
live_room
```

```
├── anchor                                                    // 主播端 视图层
目录
│   ├── common                                                // 主播端 视图层
预览界面 和 直播界面公共的视图目录
│   │   ├── settings_panel_widget.dart                        // 设置面板的实
现
│   │   └── video_params_panel_widget.dart                    // 选择超清、高
清、标清的面板
│   ├── live_streaming                                        // 直播推流状态
的视图目录
│   │   ├── link                                              // 观众连麦相关
的视图目录
│   │   │   ├── anchor_link_mic_manage_panel_widget.dart      // 连麦管理面板,
支持同意或者拒绝观众连麦等逻辑
│   │   │   └── apply_link_mic_float_widget.dart              // 有观众申请连
麦时，悬浮在顶层的提示连麦的视图
│   │   ├── anchor_living_function_widget.dart                // 直播中底部功
能区的视图
│   │   └── anchor_living_widget.dart                         // 直播中整体的
布局视图
│   ├── preview                                               // 直播预览状态
的视图目录
│   │   ├── anchor_preview_function_widget.dart               // 直播预览时底
部功能区的视图
│   │   └── anchor_preview_widget.dart                        // 直播预览整体
的布局视图
│   ├── video                                                 // 主播端推流画
面视图目录
│   │   └── anchor_video_widget.dart                          // 主播端推流画
面视图的实现
│   └── anchor_widget.dart                                    // 主播推流整体
的视图布局
├── audience                                                  // 观众视图层目
录
│   ├── live_streaming                                        // 观众拉流的视
图目录
│   │   ├── link                                              // 观众连麦相关
的视图目录
│   │   │   ├── select_link_mic_type_panel_widget.dart        // 观众连麦弹出
的选择语音连麦还是视频连麦的视图
│   │   │   └── video_link_settings_panel_widget.dart         // 视频连麦时相
关参数设置面板视图
│   │   ├── audience_function_widget.dart                     // 观众拉流页面
底部功能区的视图
```

```
    |       ├── audience_living_widget.dart                    // 观众拉流页面
子组建的布局视图
    |       └── audience_waiting_pass_widget.dart              // 观众申请连麦
时，等待主播统一的动画视图
    |   ├── video                                              // 观众端拉流画
面视图目录
    |       ├── audience_video_widget.dart                     // 观众端拉流画
面视图的实现
    |       └── link_request_place_holder_widget.dart.         // 观众申请连麦
时，用来占位的麦上用户窗口
    |   └── audience_widget.dart                               // 观众拉流整体
的视图布局
    └── video                                                  // 摄像头采集的
视频画面视图目录
        ├── nine_grid_layout.dart                              // 连麦用户的九
宫格视图布局
        ├── video_item_widget.dart                             // 一路视频流的
封装视图
        └── video_widget_factory.dart                          // 生成
VideoView 的工厂类
```

方案三：自定义全部 UI 方案

TUILiveKit 的整体功能是基于 RTC Room Engine SDK 实现的，您可以完全基于 RTC Room Engine SDK 实现一套自己的 UI 界面。详情可见 RTC Room Engine SDK 。

开播和观看（TUILiveKit）Android

最近更新时间：2025-04-07 15:22:32

适用场景

我们的**主播开播**和**观众观看**功能主要依赖于**我们视频直播的核心控件(LiveCoreView)**，该核心控件提供了 开播前画面预览、开启视频直播、关闭视频直播，直播间内和观众连线，跨房和其他主播连线等丰富的 API。
当您通过 快速接入 接入视频直播 UIKit 后，如果 UI 风格和您的理想中 UI 风格有出入，您可以使用我们的核心控件半小时内快速搭建视频直播的主流程。然后在其之上添加您自己的业务 UI 视图。

开发环境要求

- Android 5.0（SDK API Level 21）及以上版本。
- Gradle 7.0 及以上的版本。
- Android 5.0 及以上的手机设备。
- 环境配置或编译运行期间，如有问题，请参见 常见问题 。

步骤一：开通服务

请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。

步骤二：工程配置

1. 在 app 目录下找到 `build.gradle.kts(或build.gradle)` 文件，并在其中增加如下代码，加入对 LiveCoreView 组件的依赖：

build.gradle.kts

```
api("io.trtc.uikit:live-stream-core:latest.release")
```

build.gradle

```
api 'io.trtc.uikit:live-stream-core:latest.release'
```

2. 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.** { *; }
```

```
-keep class com.trtc.uikit.livekit.livestreamcore.** { *; }
-keep class com.google.gson.** { *;}
```

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

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

<application
  ...

  // 添加如下配置覆盖 依赖的 sdk 中的配置
  android:allowBackup="false"
  tools:replace="android:allowBackup">
```

步骤三：登录

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 LiveCoreView 提供的各项功能。

Kotlin

```
TUIRoomEngine.login(applicationContext,
    1400000001,       // 请替换为步骤一取到的 SDKAppID
    "denny",          // 请替换为您的 UserID
    "xxxxxxxxxxx",    // 您可以在控制台中计算一个 UserSig 并填在这个位置
    object : TUIRoomDefine.ActionCallback() {
        override fun onSuccess() {
            Log.i(TAG, "login success")
        }

        override fun onError(errorCode: Int, errorMessage: String) {
            Log.e(TAG, "login failed, errorCode: $errorCode
msg:$errorMessage")
        }
    })
```

Java

```
TUIRoomEngine.login(context,
```

```
    1400000001,     // 请替换为步骤一取到的 SDKAppID
    "denny",        // 请替换为您的 UserID
    "xxxxxxxxxx",   // 您可以在控制台中计算一个 UserSig 并填在这个位置
        new TUIRoomDefine.ActionCallback() {
            @Override
            public void onSuccess() {
                Log.i(TAG, "login success");
            }

            @Override
            public void onError(TUICommonDefine.Error error, String
message) {
                Log.e(TAG, "login failed, errorCode: " + errorCode + "
msg:" + errorMessage);
            }
        });
```

参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

参数	类型	说明
SDKAppID	int	在步骤一中的最后一步中您已经获取到，这里不再赘述。
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	使用 **步骤一** 的第3步中获取的 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** 。

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 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：使用核心控件实现直播功能

创建核心控件并开启预览

创建核心控件：您可以在您实现推流的 Activity 中通过 java 代码或者 xml 方式加载我们的核心控件，其中代码方式示例如下（XML 方式也类似）：

kotlin

```kotlin
val livecoreView = LiveCoreView(this)
```

java

```java
LiveCoreView liveCoreView = new LiveCoreView(this);
```

开启直播预览：本地摄像头预览，并未真正开启直播间。

kotlin

```kotlin
livecoreView.startCamera(true, null)
```

java

```java
liveCoreView.startCamera(true, null);
```

主播开启直播间和观众加入直播间

主播开启直播间：开启一个直播间，并将本地摄像头采集的数据和麦克风采集的数据推流到直播间。

kotlin

```kotlin
val roomInfo = TUIRoomDefine.RoomInfo()
roomInfo.roomId = "123456"
livecoreView.startLiveStream(roomInfo, null)

livecoreView.startMicrophone(null)
```

java

```java
TUIRoomDefine.RoomInfo roomInfo = new TUIRoomDefine.RoomInfo();
roomInfo.roomId = "roomId_123456";
livecoreView.startLiveStream(roomInfo, null);
```

版权所有：腾讯云计算（北京）有限责任公司

```
livecoreView.startMicrophone(null);
```

观众加入直播间：观众进入直播间，并拉取直播间主播的视频流和音频流。

kotlin

```
livecoreView.joinLiveStream("roomId_123456", null)
```

java

```
livecoreView.joinLiveStream("roomId_123456", null);
```

主播开启直播间开始直播	观众加入直播间观看直播





观众连麦

您如果需要实现观众连麦功能，可参见 观众连麦 文档。

主播连线

您如果需要实现主播连线功能，可参见 主播连线 文档。

设置连线布局

您可以通过以下方式快速设置您与连线主播或您与连麦观众的布局。

kotlin

```
// 设置宫格布局
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.GRID_LAYOU
T, "")

// 设置浮窗布局
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.FLOAT_LAYO
UT, "")

// 设置自定义布局
var layoutJson = ""
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.FREE_LAYOU
T, layoutJson)
```

java

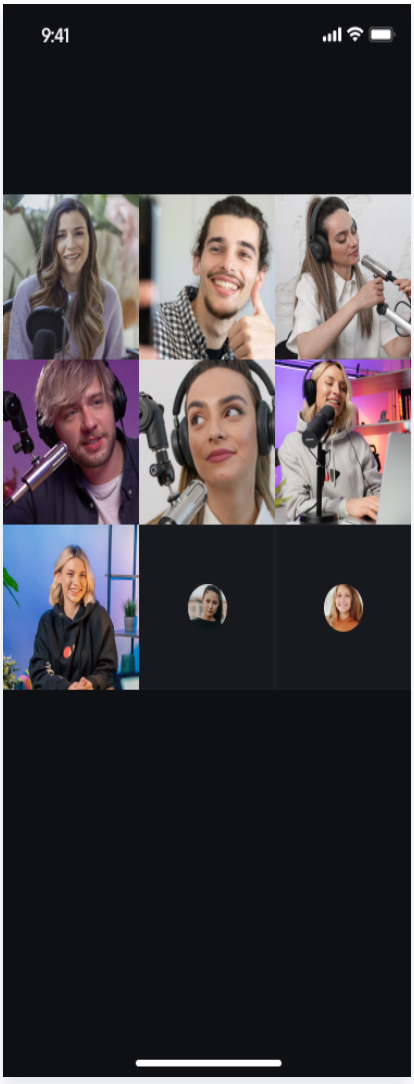
```
// 设置宫格布局
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.GRID_LAYOU
T, "");

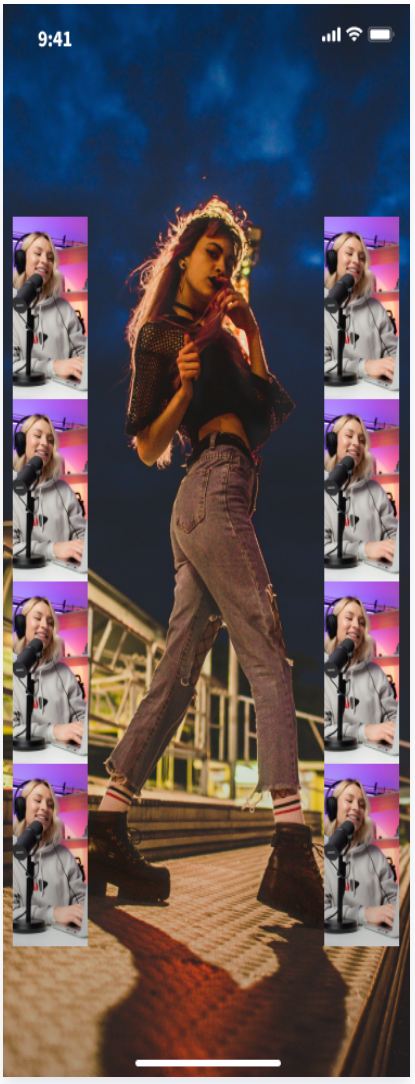
// 设置浮窗布局
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.FLOAT_LAYO
UT, "");

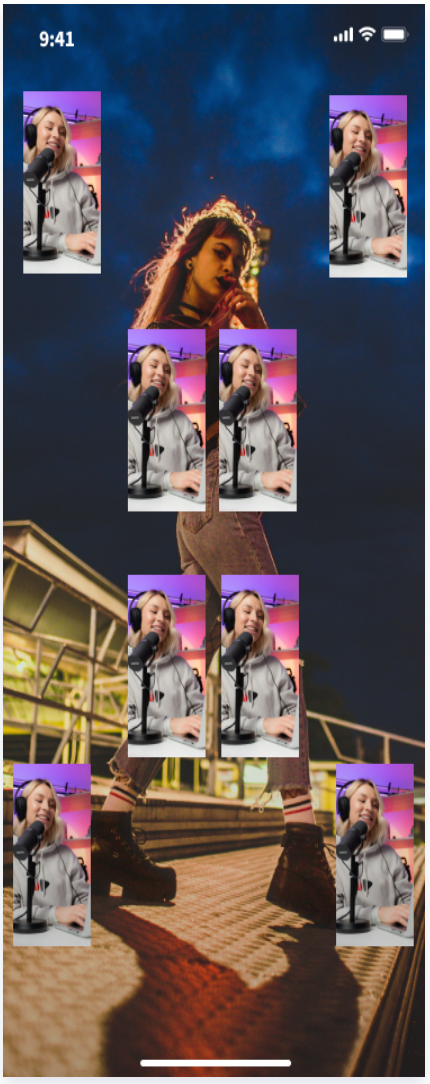
// 设置自定义布局
String layoutJson = "";
mLiveStreamListView.setLayoutMode(LiveCoreViewDefine.LayoutMode.FREE_LAYOU
T, layoutJson);
```

说明：
自定义布局 layoutJson 是一个 json 格式的字符串，详细内容请查看 自定义布局 json 。

主播连线中-九宫格布局	主播连线中-浮窗布局	主播连线中-自定义布局







自定义布局 json

自定义布局的 json 结构说明如下：

```
{
    "1": {                                  // 视频视图的数量
        "backgroundColor": "#000000",       // 画布的背景颜色，采用 RGB 十六进制格式
        "viewInfoList": [{                  // 每个视频视图的布局信息和背景颜色
            "x": 0,                         // 水平偏移与屏幕宽度的比例，取值范围 [0, 1]
            "y": 0,                         // 垂直偏移与屏幕宽度的比例，取值范围 [0, 1]
```

```
            "width": 1,                         // 视频视图的宽度与屏幕宽度的比例，
取值范围 [0, 1]
            "height": -1,                       // 视频视图的高度与屏幕宽度的比例，
取值范围 [0, 1] 或 -1；-1 表示视图高度与屏幕高度相同
            "zOrder": 0,                        // 视频视图的层级顺序，数值越大，视
图越靠上
            "backgroundColor": "#000000"    // 当前视频视图的背景颜色，采用 RGB
十六进制格式
        }]
    }
}
```

iOS

最近更新时间：2025-05-23 09:49:51

适用场景

我们的**主播开播**和**观众观看**功能主要依赖于**我们核心控件(LiveCoreView)**，该核心控件提供了 开播前画面预览、开启视频直播、关闭视频直播，直播间内和观众连线，跨房和其他主播连线等丰富的 API。
当您通过 快速接入 接入视频直播 UIKit 后，如果 UI 风格和您的理想中 UI 风格有出入，您可以使用我们的核心控件半小时内快速搭建视频直播的主流程。然后在其之上添加您自己的业务 UI 视图。

环境准备

- Xcode 15 及以上。
- iOS 13.0 及以上。
- CocoaPods 环境安装，点击查看 。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

步骤1：开通服务

请参见 开通服务（TUILiveKit），领取体验版或者开通付费版。

步骤2：导入 LiveStreamCore 组件

使用 CocoaPods 导入组件，如果您遇到问题，请先参见 环境准备 。导入组件具体骤如下：
1. 请在您的 `Podfile` 文件中添加 `pod 'LiveStreamCore'` 依赖。

Swift

```
target 'xxxx' do
  ...
  ...
  pod 'LiveStreamCore'
end
```

如果您没有 `Podfile` 文件，首先终端 `cd` 到 `xxxx.xcodeproj` 目录，然后通过以下命令创建：

```
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
```

如果无法安装 LiveCoreView 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. 可以先编译运行一下，如果遇到问题，请参见 常见问题 。问题如果依然无法解决，可以先去跑一下我们的 Example 工 程。您在接入和使用过程中遇到的任何问题，欢迎给我们 反馈 。

步骤3：登录

在您的项目中添加如下代码，它的作用是通过调用 RTCRoomEngine 中的登录相关接口完成 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 LiveCoreView 提供的各项功能。

swift

```
//
//  AppDelegate.swift
//

import RTCRoomEngine

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

    TUIRoomEngine.login(sdkAppId: 1400000001,        // 请替换为步骤一取到的
SDKAppID
                        userId: "denny",             // 请替换为您的 UserID
                       userSig: "xxxxxxxxxxx") {   // 您可以在控制台中计算一
个 UserSig 并填在这个位置
      print("login success")
    } onError: { code, message in
      print("login failed, code: \(code), error: \(message ?? "nil")")
    }
```

```
    return true
}
```

参数说明

这里详细介绍一下 login 函数中所需要用到的几个关键参数：

参数	类型	说明
SDKAppID	int	在步骤一中的最后一步中您已经获取到，这里不再赘述。
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	使用 步骤一 的第3步中获取的 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 。

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 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：使用核心控件实现直播功能

创建核心控件并开启预览

创建核心控件：

swift

```
import LiveStreamCore

let liveCoreView = LiveCoreView()
```

开启直播预览：本地摄像头预览，并未真正开启直播间。

swift

```
import LiveStreamCore

liveCoreView.startCamera(useFrontCamera: true) {
} onError: { code, message in
}
```

主播开启直播间和观众加入直播间

主播开启直播间：开启一个直播间，并将本地摄像头采集的数据和麦克风采集的数据推流到直播间。

swift

```
import LiveStreamCore
import RTCRoomEngine

let roomInfo = TUIRoomInfo()
roomInfo.roomId = "123456"
roomInfo.seatMode = .applyToTake

liveCoreView.startLiveStream(roomInfo: roomInfo) { roomInfo in
} onError: { code, message in
}

liveCoreView.startMicrophone {
} onError: { code, message in
}
```

观众加入直播间：观众进入直播间，并拉取直播间主播的视频流和音频流。

swift

```
import LiveStreamCore

liveCoreView.joinLiveStream(roomId: "roomId_123456") { roomInfo in
} onError: { code, message in
}
```

主播开启直播间开始直播	观众加入直播间观看直播





观众连麦

您如果需要实现观众连麦功能，可参见 观众连麦 文档。

主播连线

您如果需要实现主播连线功能，可参见 主播连线 文档。

设置连线布局

您可以通过以下方式快速设置您与连线主播或您与连麦观众的布局。

swift

```
import LiveStreamCore

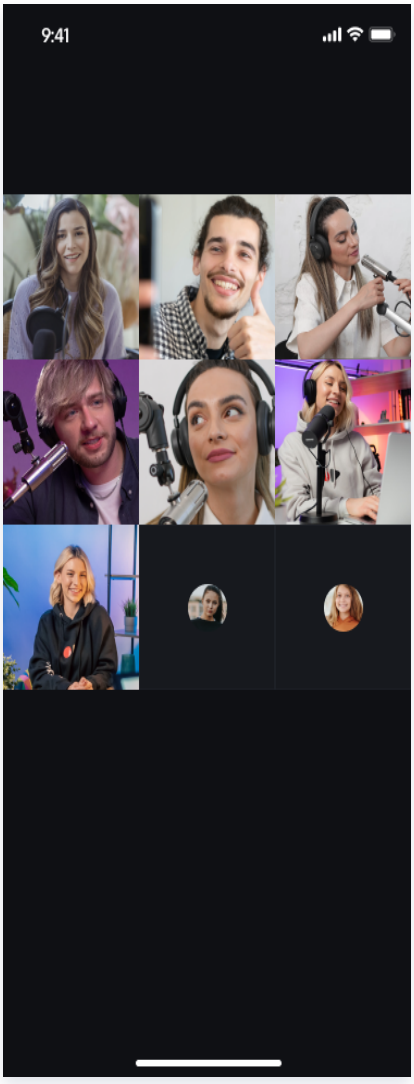
// 设置宫格布局
liveCoreView.setLayoutMode(layoutMode: .gridLayout)

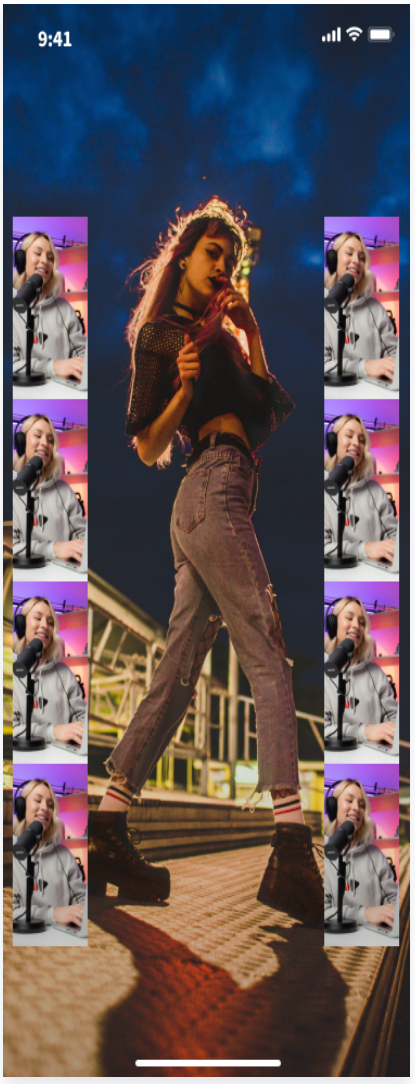
// 设置浮窗布局
liveCoreView.setLayoutMode(layoutMode: .floatLayout)

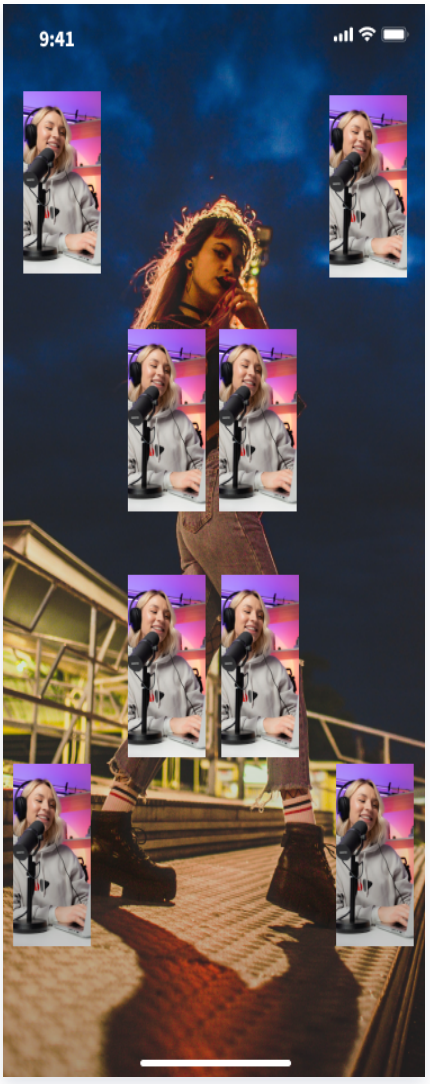
// 设置自定义布局
let layoutJson = ""
liveCoreView.setLayoutMode(layoutMode: .freeLayout, layoutJson:
layoutJson)
```

说明：
自定义布局 layoutJson 是一个 json 格式的字符串，详细内容请查看 自定义布局 json 。

主播连线中-九宫格布局	主播连线中-浮窗布局	主播连线中-自定义布局







自定义布局 json

自定义布局的 json 结构说明如下：

```
{
    "1": {                                  // 视频视图的数量
        "backgroundColor": "#000000",       // 画布的背景颜色，采用 RGB 十六进制格式
        "viewInfoList": [{                  // 每个视频视图的布局信息和背景颜色
            "x": 0,                         // 水平偏移与屏幕宽度的比例，取值范围 [0, 1]
            "y": 0,                         // 垂直偏移与屏幕宽度的比例，取值范围 [0, 1]
```

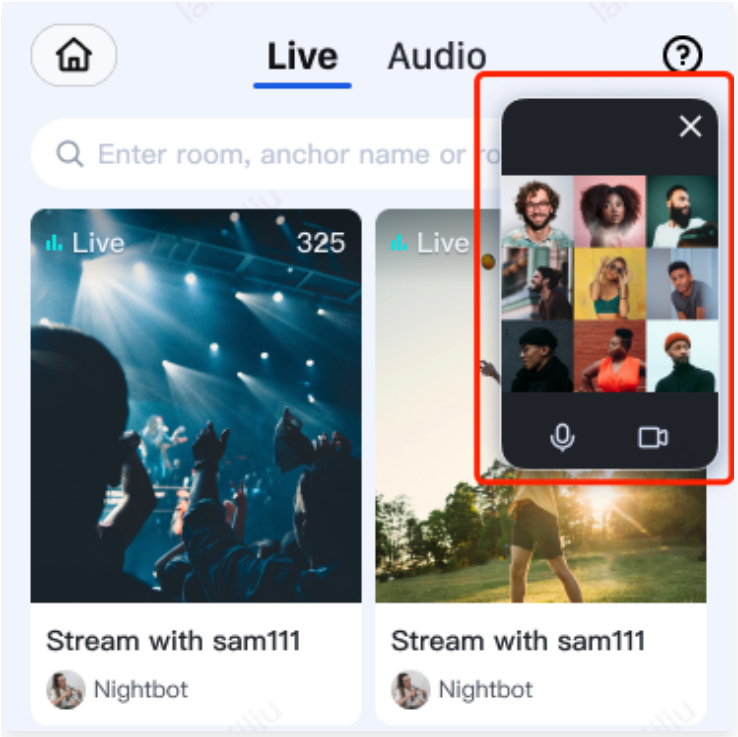
```
            "width": 1,                         // 视频视图的宽度与屏幕宽度的比例，
取值范围 [0, 1]
            "height": -1,                       // 视频视图的高度与屏幕宽度的比例，
取值范围 [0, 1] 或 -1；-1 表示视图高度与屏幕高度相同
            "zOrder": 0,                        // 视频视图的层级顺序，数值越大，视
图越靠上
            "backgroundColor": "#000000"    // 当前视频视图的背景颜色，采用 RGB
十六进制格式
        }]
    }
}
```

悬浮窗（TUILiveKit）
Android

最近更新时间：2025-04-03 16:12:02

功能介绍

悬浮窗功能允许主播和观众在不退出直播间的情况下，进入到小窗口模式，小窗口悬浮在屏幕上，您可以自由拖拽到合适的位置，悬浮窗内可以显示直播画面。在悬浮窗模式下，App 可以切换到别的场景或者退后台，点击悬浮窗还可以恢复到正常全屏直播界面。



使用说明

原理简介

悬浮窗的原理是，在 WindowManager 上添加自定义视图作为悬浮窗的内容。在Android6.0及以上系统，需要申请悬浮窗权限。

快速接入

悬浮窗功能核心代码：

```
// 文件位置：
Android/tuilivekit/src/main/java/com/trtc/uikit/livekit/component/floatwindow/core
```

```
core                                // 悬浮窗核心功能的实现目录
   ├── impl                         // 悬浮窗的具体实现
   ├── FloatWindow.java             // 悬浮窗功能对外接口
   └── FloatWindowObserver.java     // 悬浮窗监听器
```

如果您没有使用 `Android/tuilivekit`，您可以将该目录拷贝到您的工程中使用。
快速实现悬浮窗的示例代码：

```
// contentView 是您要放在悬浮窗上展示的内容视图，请需要替换成自己的视图。
// contentView 上的状态、事件和业务逻辑等由您自己负责。FloatWindow 只是负责把 contentView 展示出来。
TextView contentView = new TextView(this);
contentView.setBackgroundColor(0xE022262E);
contentView.setTextColor(Color.YELLOW);
contentView.setGravity(Gravity.CENTER);
contentView.setText("This is a float window");

FloatWindow floatWindow = FloatWindow.getInstance();
floatWindow.setView(contentView);
floatWindow.show();
```

示例悬浮窗的效果图如下：



申请悬浮窗权限：
首先在 `AndroidManifest.xml` 中注册 `android.permission.SYSTEM_ALERT_WINDOW` 权限：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"
/>
```

```
</manifest>
```

然后动态申请权限：

```
FloatWindow floatWindow = FloatWindow.getInstance();
if (floatWindow.hasPermission()) {
    floatWindow.setView(contentView);
    floatWindow.show()
} else {
    floatWindow.requestPermission();
}
```

> **注意：**
> 在 `Android6.0` 及以上系统，需要申请悬浮窗权限。您也可以自己实现申请悬浮窗权限的逻辑，只要在获得悬浮窗权限后调用 `FloatWindow` 的 show 方法即可。

监听悬浮窗事件：

```
FloatWindow floatWindow = FloatWindow.getInstance();
floatWindow.setObserver(new FloatWindowObserver() {
    @Override
    public void onClickWindow() {
        // 收到点击悬浮窗的通知，通常您需要在此实现的业务逻辑是：切换到正常全屏模式，
并关闭悬浮窗。
    }

    @Override
    public void onMove(int x, int y) {
        // 收到悬浮窗移动的通知
    }
});
```

自定义悬浮窗设置：

```
FloatWindow floatWindow = FloatWindow.getInstance();
floatWindow.setSize(width, height);
floatWindow.setLocation(x, y);
floatWindow.setMarginToEdge(margin);
```

模式切换

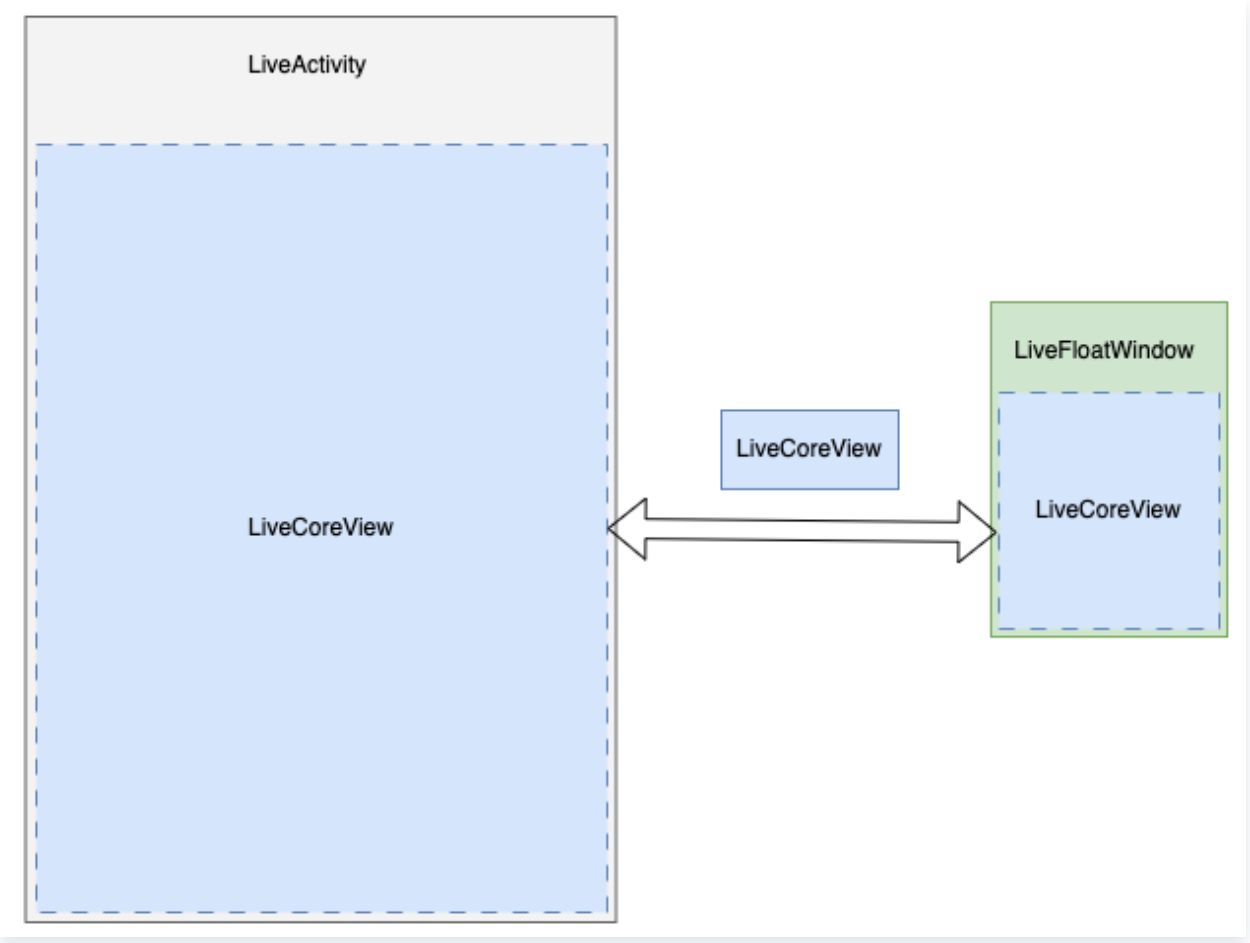
通常，悬浮窗模式与正常模式需要互相切换。相关步骤如下：

正常模式切换到悬浮窗模式	悬浮窗模式切换到正常模式
• 申请悬浮窗权限（如果有权限那么无需申请）； • 创建内容视图，并设置给悬浮窗； • 退出 `Activity`，但要保留相关业务的数据和状态。 • 弹出悬浮窗。	• 监听悬浮窗事件； • 在收到 `onClickWindow` 通知时，关闭悬浮窗，然后跳转回到原来的 `Activity`，并恢复相关业务的数据和状态。

注意：

- 悬浮窗功能本身只需要您传入一个内容视图即可，与您的业务没有任何关联。
- 模式切换过程需要保存/恢复相关业务数据和状态，这部分功能取决于您的代码架构和具体业务，所以需要您自行实现。

直播场景里，模式切换过程中，您需要使用同一个 `LiveCoreView` 对象，用于在两种模式下展示直播内容，这样不但可以共享视图，还可以共享直播数据和状态。



如上图所示，正常模式是 `LiveActivity` 场景，悬浮窗模式是 `LiveFloatWindow` 场景，在模式切换过程中，可以使用同一个 `LiveCoreView` 对象，以保证直播过程正常进行。由于 `LiveCoreView` 要在 `LiveActivity` 销毁

后依然能在 `LiveFloatWindow` 使用，所以创建 `LiveCoreView` 时使用的 `context` 是 `ApplicationContext` 而非 `Activity` 。

```
LiveCoreView liveCoreView = new
LiveCoreView(context.getApplicationContext());
```

参考实现

`Android/tuilivekit` 直播场景默认实现了悬浮窗功能，可以供您参考：

```
// 文件位置：
Android/tuilivekit/src/main/java/com/trtc/uikit/livekit/component/floatwin
dow

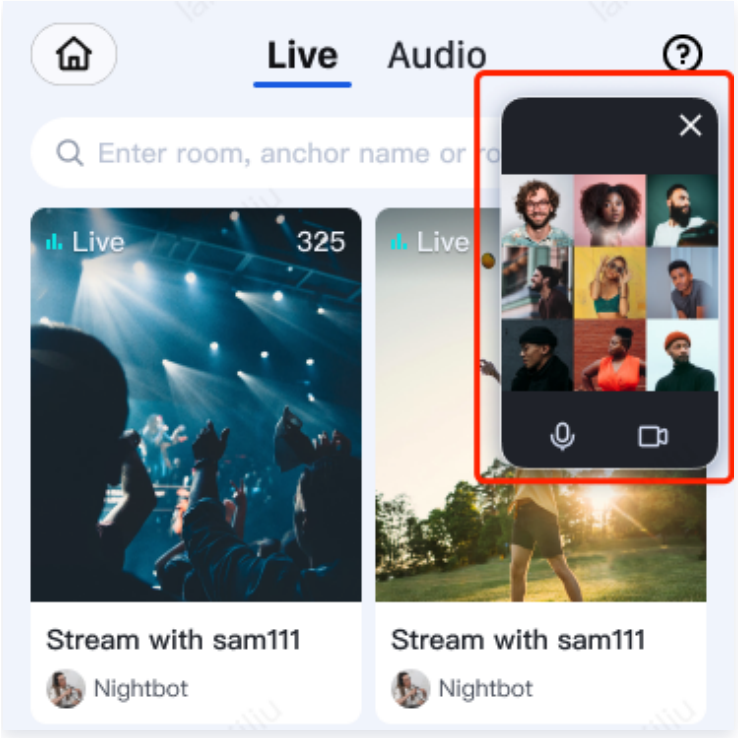
floatwindow                                  // 悬浮窗功能的实现目录
  ├── service                                // LiveKit 悬浮窗业务相关目录
  |       └── FloatWindowManager.java        // 负责实现悬浮窗模式与正常模式切换等业务
  ├── store                                  // LiveKit 悬浮窗状态相关目录
  |       └── FloatWindowStore.java
  └── view                                   // LiveKit 悬浮窗视图相关目录
        ├── VideoFloatView.java              // 直播场景悬浮窗内容视图
        └── VoiceFloatView.java              // 语聊场景悬浮窗内容视图。语聊暂未支持悬
浮窗。
```

iOS

最近更新时间：2025-04-03 16:12:02

功能介绍

悬浮窗功能允许主播和观众在不退出直播间的情况下，进入到小窗口模式，小窗口悬浮在屏幕上，您可以自由拖拽到合适的位置，悬浮窗内可以显示直播画面。在悬浮窗模式下，App 可以切换到别的场景或者退后台，点击悬浮窗还可以恢复到正常全屏直播界面。



使用说明

原理简介

悬浮窗的原理是，在 rootWindow 上添加视频直播的视图作为悬浮窗的内容。

快速接入

悬浮窗功能核心代码：

```
// 文件位置：iOS/TUILiveKit/Sources/Component/FloatWindow
FloatWindow                          // 悬浮窗核心功能的实现目录
  ├── FloatWindow.swift              // 悬浮窗功能对外接口
  └── FloatView.swift                // 悬浮窗View
```

悬浮窗数据源：

展示悬浮窗的 ViewController 需要实现如下数据源，以便悬浮窗可以拿到正确的视图数据。

```
protocol FloatWindowDataSource: {
    func getRoomId() -> String                  // 当前直播间的Id
    func getCoreView() -> LiveCoreView()        // 悬浮窗上展示的视频View
    func relayoutCoreView()                     // 当从悬浮窗状态恢复到视图原
ViewController时会触发该函数，在此重新设置视频View的布局
}
```

使用场景及示例：

展示悬浮窗：

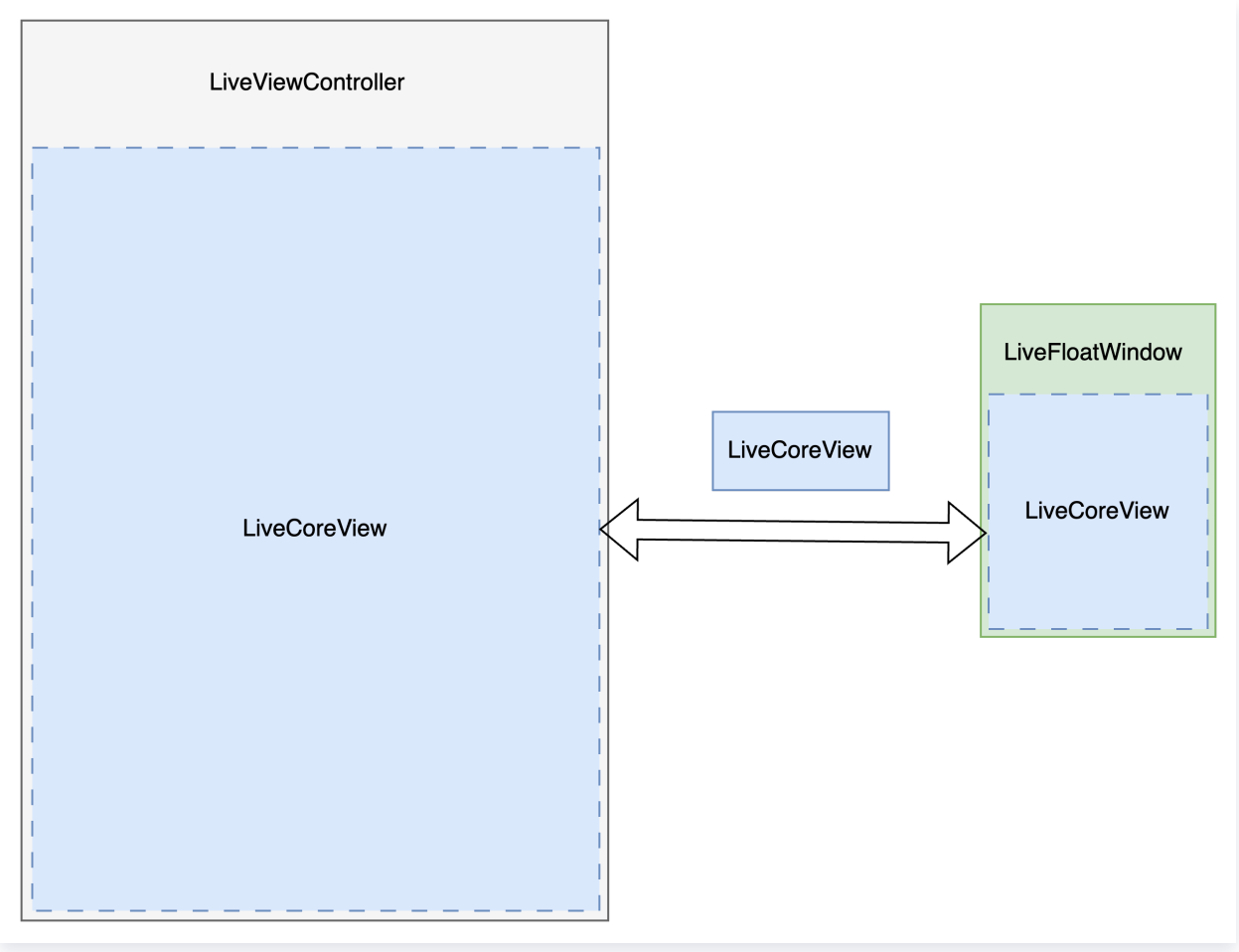
```
class ExampleViewController: UIViewController {
    override func touchesBegan(_ touches: Set<UITouch>, with event:
UIEvent?) {
        // 展示悬浮窗，当前ViewController需要实现FloatWindowDataSource
        // 在展示悬浮窗时，会退出当前ViewController
        FloatWindow.shared.showFloatWindow(controller: self)
    }
}
```

观众点击进入房间列表的某直播间：

```
if FloatWindow.shared.isShowingFloatWindow() {
    if FloatWindow.shared.getCurrentRoomId() == liveInfo.roomInfo.roomId {
      // 切回到悬浮窗所在的视频直播间
      FloatWindow.shared.resumeLive(atViewController:
self.navigationController ?? self)
      return
    } else {
      // 切换到其他的视频直播间
      FloatWindow.shared.releaseFloatWindow()
    }
}
```

模式切换

直播场景里，模式切换过程中，您需要使用同一个 `LiveCoreView` 对象，用于在两种模式下展示直播内容，这样不但可以共享视图，还可以共享直播数据和状态。



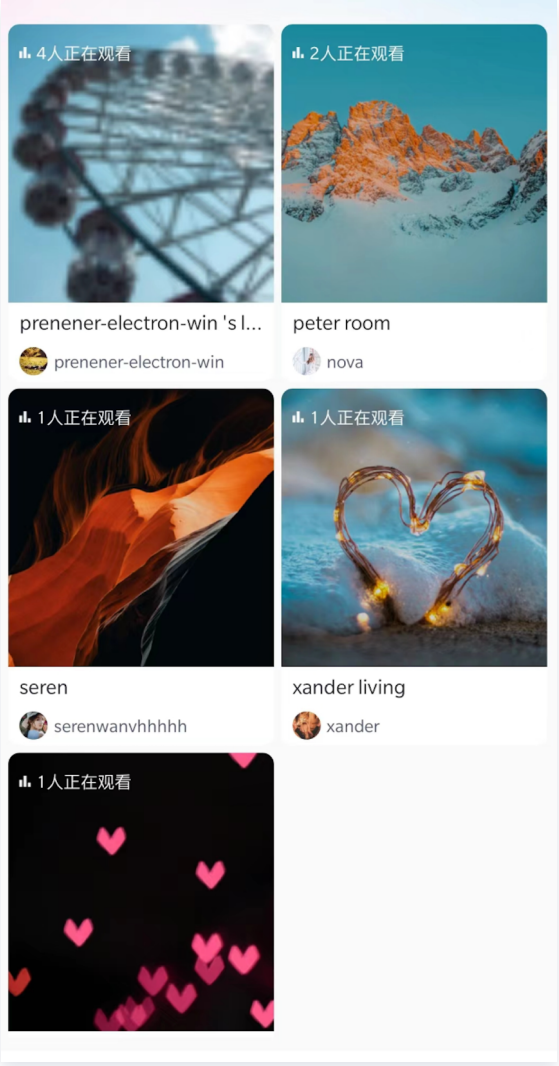
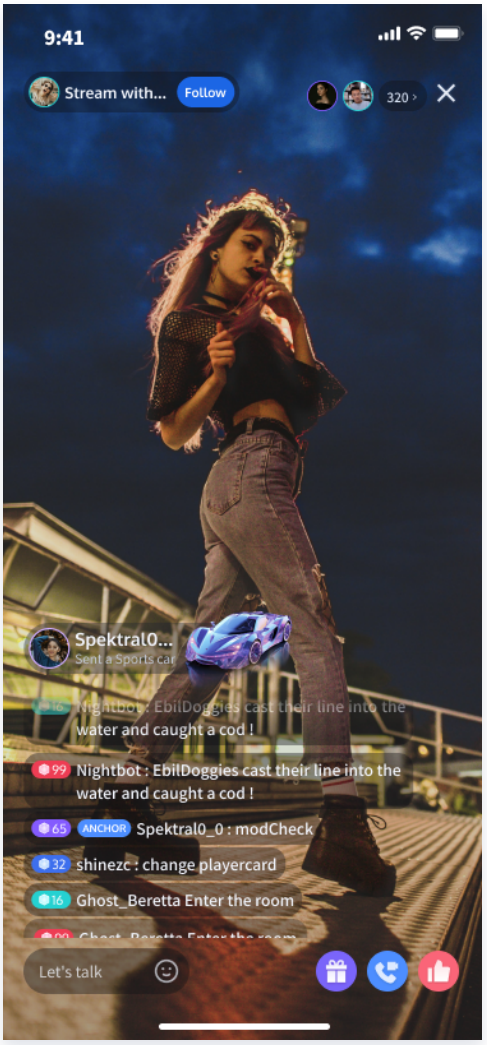
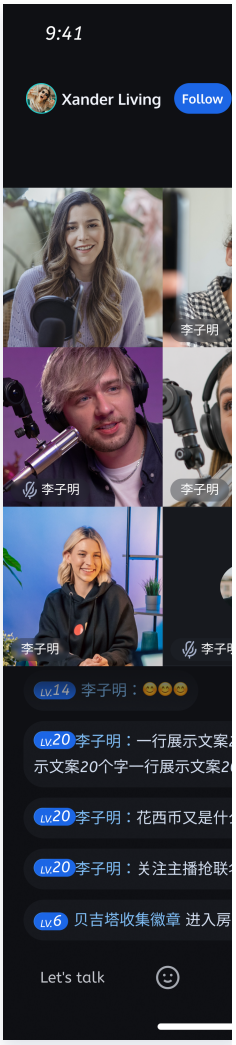
如上图所示，正常模式是 `LiveViewController` 场景，悬浮窗模式是 `LiveFloatWindow` 场景，在模式切换过程中，可以使用同一个 `LiveCoreView` 对象，以保证直播过程正常进行。

房间列表（TUILiveKit）

最近更新时间：2025-04-14 18:02:42

功能介绍

TUILiveKit 支持房间列表 UI 组件，房间列表组件可以帮您展示当前应用下所有的在线直播和语音聊天室的直播间，当您接入房间列表 UI 组件后，只需要点击列表中的某个直播间，就可以实时观看当前主播的直播内容。进入直播间后您还可以通过弹幕、礼物、连麦等功能和主播进行实时互动。

房间列表组件	观看直播	和主
		

功能接入

Android

> **说明：**
> 请务必确保已经按照 快速接入（TUILiveKit）完成登录操作。只有 `TUILogin.login` 登录成功后才能正常进入直播预览画面。

1. 在需要显示房间列表 UI 组件的 xml 中添加 FrameLayout 布局。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/fl_live_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

2. 通过加载 TUILiveKit 的 `TUILiveListFragment` 页面到 xml 中定义的布局上，就可以展示房间列表。

Java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_main);

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
        TUILiveListFragment listFragment = new TUILiveListFragment();
        fragmentTransaction.add(R.id.fl_live_list, listFragment);
        fragmentTransaction.commit();
    }
}
```

iOS

说明：
请务必确保已经按照 快速接入 完成登录操作。只有 `TUILogin.login` 登录成功后才能正常进入直播预览画面。

Swift

```
//
//  MainViewController.swift
//

import UIKit
import TUILiveKit

@objc private func buttonTapped(_ sender: UIButton) {
    // 进入房间列表
    let liveListViewController = TUILiveListViewController()
    self.navigationController?.pushViewController(viewController,
animated: true)
}
```

Objective-C

```
//
//  MainViewController.m
//

#import <TUILiveKit/TUILiveKit-Swift.h>

- (void)buttonTapped:(UIButton *)sender {
    // 进入房间列表
    TUILiveListViewController *liveListViewController =
[[TUILiveListViewController alloc] init];
    [self.navigationController
pushViewController:liveListViewController animated:true];
}
```

Flutter

> **说明：**
> 请务必确保已经按照 快速接入（TUILiveKit）完成登录操作。只有登录成功后才能正常进入直播预览画面。

在您的 widget 中，通过加载 TUILiveKit 的 `LiveListWidget` 组件，就可以展示房间列表。

Dart

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


return Scaffold(
  body: SizedBox(
    width: _screenWidth,
    height: double.infinity,
    child: LiveListWidget(),     // 在自己的 widget树中 添加 TUILiveKit 的房间列表组件 LiveListWidget
  ),
);
```

功能定制

如果当前的 UI 不满足您的需求，您可以通过修改如下目录中的源代码，来实现您满意的 UI 效果。为了您更方便的定制 UI，这里对房间列表相关的文件做了介绍。

Android

```
// 文件位置：
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/common/uicomponent/

roomlist                                          // 直播间列表组件的实现目录
  ├── service                                     // 直播间列表组件 的服务层目录
  |    └── RoomListService.java                   // 直播间列表组件服务层的具体实现，封装了直播间列表相关的 api
```

```
    ├── store                                    // 直播间列表组件 的数据层目
录
    │   └── RoomListState.java                   // 直播间列表组件 数据的具体
封装类
    └── view                                     // 直播间列表组件 的视图层目
录
        ├── adapter                              // 直播间列表组件 的视图层的
适配器目录
        │   ├── LoadMoreAdapterWrapper.java      // 直播间列表组件 对直播间列
表适配器扩展了下拉加载功能的适配器
        │   └── RoomListAdapter.java             // 直播间列表组件 的直播间列
表适配器
        ├── ListAudienceActivity.java            // 直播间列表组件 中点击某一
个直播间，拉起的直播拉流页面
        └── RoomListView.java                    // 直播间列表组件 直播间列表
视图的实现
```

iOS

```
// 文件位置：iOS/TUILiveKit/Sources/Component/LiveList/

├── LiveList                                    // 直播间列表组件的实现目录
│ ├── Service                                   // 直播间列表组件 的服务层目录
│ │ └── LiveListService.swift                   // 直播间列表组件服务层的具体
实现，封装了直播间列表相关的 api
│ ├── Store                                     // 直播间列表组件 的数据层目录
│ │ ├── LiveListActions.swift                   // 直播间列表组件 的事件定义
类，定义了所有直播列表相关事件
│ │ ├── LiveListReducer.swift                   // 直播间列表组件 的事件响应
类，当有事件发生时会触发，用于监听修改直播列表相关数据
│ │ ├── LiveListSelectors.swift                 // 直播间列表组件 的数据选择
类，获取数据源的值
│ │ ├── LiveListState.swift                     // 直播间列表组件 的数据定义
类，定义了所有与直播列表有关的数据模型
│ │ └── LiveListStore.swift                     // 直播间列表组件 的数据驱动和
事件发送协议类
│ └── View                                      // 直播间列表组件 的视图层目录
│     ├── LiveListCell.swift                    // 直播间列表组件 的自定义
Cell
```

```
|       └── LiveListRootView.swift              // 直播间列表组件 的根视图
```

Flutter

```
// 文件位置: TUILiveKit/Flutter/livekit/lib/component/

room_list                                       // 直播间列表组件的实现目
录
  ├── service                                   // 直播间列表组件 的服务
层目录
  |   └── room_list_service.dart                // 直播间列表组件服务层的
具体实现，封装了直播间列表相关的 api
  ├── store                                     // 直播间列表组件 的数据
层目录
  |   └── room_list_state.dart                  // 直播间列表组件 数据的
具体封装类
  └── view                                      // 直播间列表组件 的视图
层目录
      └── room_list_view.dart                   // 直播间列表组件 直播间
列表视图的实现
```

关键代码

获取直播间列表插件

Android

Java

```
// 文件位置:
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/common/uicompo
nent/roomlist/service/RoomListService.java

private final TUILiveListManager mTUILiveListManager;
```

```
mTUILiveListManager = (TUILiveListManager)
TUIRoomEngine.sharedInstance().getExtension(LIVE_LIST_MANAGER);
```

iOS

Swift

```
// 文件位置:
iOS/TUILiveKit/Source/Component/LiveList/Service/LiveListService.swift

let listManager = roomEngine.getExtension(extensionType:
.liveListManager) as? TUILiveListManager
```

Flutter

Dart

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/component/room_list/service/room_list_
service.dart

late final TUILiveListManager _liveListManager =
TUIRoomEngine.sharedInstance().getExtension(TUIExtensionType.liveListM
anager);
```

获取直播间列表

Android

Java

```
// 文件位置：
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/common/uicompo
nent/roomlist/service/RoomListService.java

private static final int FETCH_LIST_COUNT = 20;
public String cursor = "";
mTUILiveListManager.fetchLiveList(cursor, FETCH_LIST_COUNT, new
LiveInfoListCallback() {
    @Override
    public void onSuccess(LiveInfoListResult result) {
    }

    @Override
    public void onError(TUICommonDefine.Error error, String s) {
    }
});
```

iOS

Swift

```
// 文件位置：
iOS/TUILiveKit/Source/Component/LiveList/Service/LiveListService.swift

func getLiveList(cursor: String, count: Int = 20) ->
AnyPublisher<(String, [TUILiveInfo]), InternalError> {
    return Future<(String,[TUILiveInfo]), InternalError> { [weak self]
promise in
      guard let self = self else { return }
      guard let listManager = roomEngine.getExtension(extensionType:
.liveListManager) as? TUILiveListManager else {
        promise(.failure(InternalError(error:TUIError.failed, message:
"getRoomListFailed")))
        return
      }
      listManager.fetchLiveList(cursor: cursor, count: count) {
cursor, liveInfoList in
        promise(.success((cursor, liveInfoList)))
      } onError: { error, message in
```

```
        promise(.failure(InternalError(error: error, message:
message)))
      }
    }.eraseToAnyPublisher()
}
```

Flutter

Dart

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/component/room_list/service/room_list_
service.dart

Future<void> _fetchLiveList() async {
  final String cursor = roomListState.cursor;
  TUIValueCallBack<TUILiveListResult> result = await
_liveListManager.fetchLiveList(cursor, fetchListCount);
  if (result.code != TUIError.success) {
    ErrorHandler.onError(result.code);
    roomListState.loadStatus.value = false;
    roomListState.refreshStatus.value = false;
    roomListState.isHaveMoreData.value = false;
  } else {
    final liveListResult = result.data as TUILiveListResult;
    roomListState.liveInfoList.value = liveListResult.liveInfoList;
    roomListState.cursor = liveListResult.cursor;
    roomListState.loadStatus.value = false;
    roomListState.refreshStatus.value = false;
    roomListState.isHaveMoreData.value =
liveListResult.cursor.isNotEmpty;
  }
```

关注主播（TUILiveKit）

最近更新时间：2025-04-03 16:12:02

功能介绍

在直播场景下，关注主播功能是一种重要的互动方式，它允许观众选择关注特定的主播，从而在直播平台上建立长期的互动和关注关系。TUILiveKit 中已经通过即时通信 IM 实现了关注主播的功能。

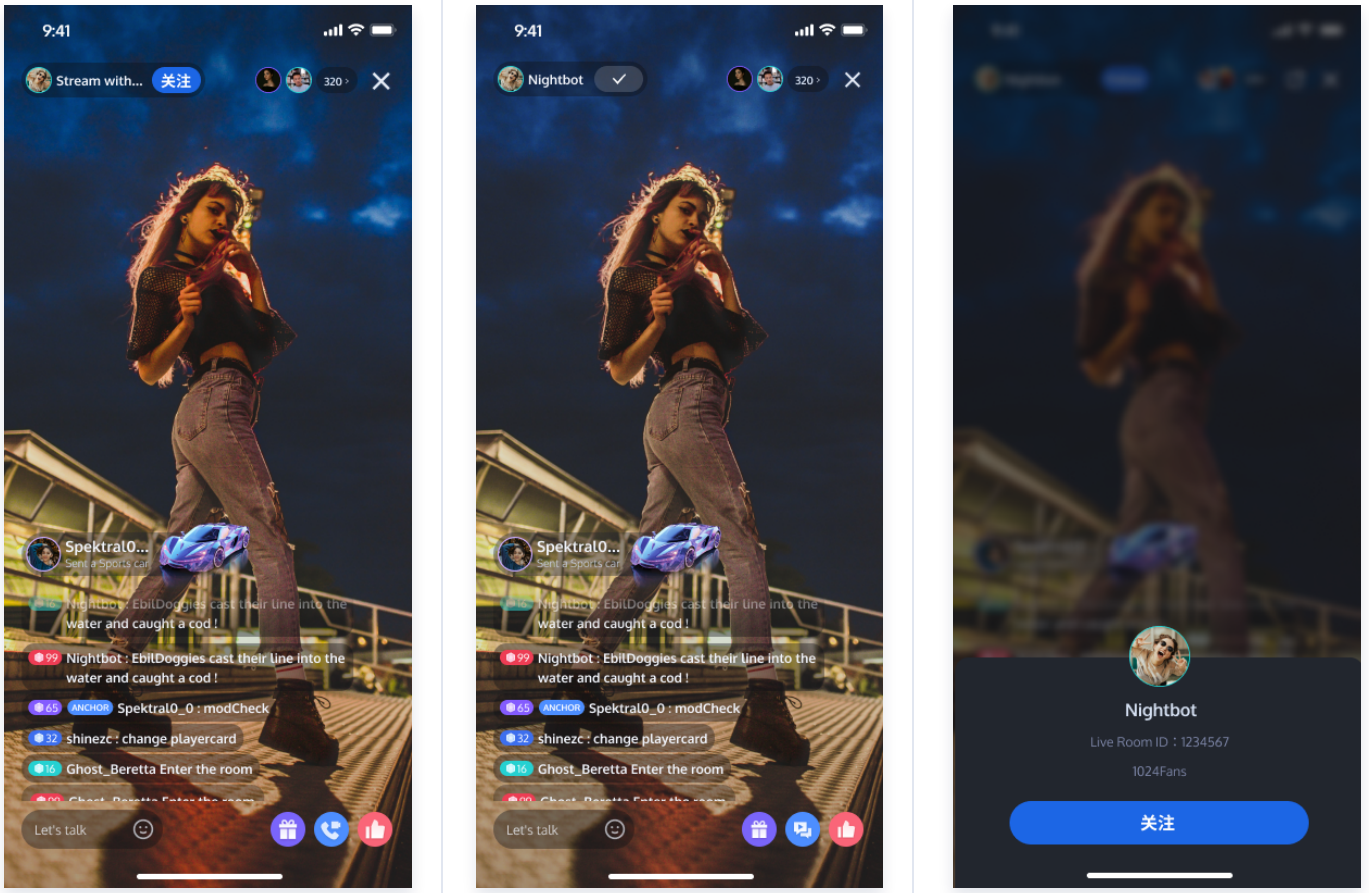
- **关注** 功能允许用户选择他们感兴趣的其他用户，以便及时获取这些用户最新的动态、发布或活动信息，您可以根据用户的关注列表提供个性化的内容推荐。
- **粉丝** 功能是指用户被其他人关注的状态，当用户 A 关注用户 B 时，A 就成为了 B 的粉丝。

通过关注和粉丝功能，社交应用和网站能够创建一个活跃的、相互连接的用户网络，促进信息的传播和社区的构建。

使用说明

> **说明：**
> TUILiveKit 的**关注主播功能**依赖于**即时通信 IM 的关注&粉丝相关的 API**，所以需要开通即时通信 IM 相关的套餐包。具体可参见 关注&粉丝 文档。

未关注状态	关注状态	主播信息面板



- **关注主播**：您可以在直播界面的左上角的直播间信息区域点击 关注 按钮关注主播
- **取消关注**：您可以在直播界面的左上角的直播间信息区域点击 ✓ 按钮可以取消关注主播
- **查看主播粉丝数**：您可以点击直播界面左上角的直播间信息区域弹出直播间详情面板，该面板会展示主播的**粉丝数量**。

功能定制

如果当前的 UI 不满足您的需求，您可以通过修改如下目录中的源代码，来实现您满意的 UI 效果。为了您更方便的定制 UI，这里对关注功能相关的文件做了介绍。

Android

```
// 文件位置：
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/common/uicomponent/
```

```
roominfo                              // 直播间信息UI组件的实现目录
  ├── RoomInfoDetailView.java         // 直播间信息详情面板视图的具体实现
  └── RoomInfoView.java               // 直播间信息视图的具体实现，点击该视图，会展
示详情面板视图
```

iOS

```
// 文件位置：TUILiveKit/iOS/TUILiveKit/Sources/Component/LiveInfo/

LiveInfo                                  // 直播间信息UI组件的实现目录
  ├── LiveInfoView.swift                  // 直播间信息视图的具体实现，点击该视
图，会展示详情面板视图
  └── View
      └── LiveInfoPanelView.swift         // 直播间信息详情面板视图的具体实现
```

关键代码

关注主播

Android

Java

```
// 文件位置：
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/service/impl/
LiveServiceImpl.java

List<String> userIDList = new ArrayList<>();
userIDList.add("userId");

V2TIMManager.getFriendshipManager().followUser(userIDList,
        new V2TIMValueCallback<List<V2TIMFollowOperationResult>>() {
            @Override
            public void onSuccess(List<V2TIMFollowOperationResult>
results) {
            }
```

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

iOS

Swift

```
// 文件位置:
TUILiveKit/iOS/TUILiveKit/Sources/Component/LiveInfo/RoomInfoService

public func followUser(userId: String) {
    let userInfo = TUIUserInfo()
    userInfo.userId = userId
    V2TIMManager.sharedInstance().followUser([userId]) { [weak self]
followResultList in
        guard let self = self, let result = followResultList?.first
else { return }
        if result.resultCode == 0,
!self.state.followingList.contains(where: { $0.userId == userId }) {
            self.state.followingList.insert(userInfo)
            self.getFansNumber()
        }
    } fail: { code, message in
        debugPrint("[RoomInfo] followUser failed, error:\(code),
message:\(String(describing: message))")
    }
}
```

取消关注主播

Android

Java

```
// 文件位置：
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/service/impl/
LiveServiceImpl.java

List<String> userIDList = new ArrayList<>();
userIDList.add("userId");

V2TIMManager.getFriendshipManager().unfollowUser(userIDList,
        new V2TIMValueCallback<List<V2TIMFollowOperationResult>>() {
            @Override
            public void onSuccess(List<V2TIMFollowOperationResult>
results) {
            }

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

iOS

Swift

```
// 文件位置：
TUILiveKit/iOS/TUILiveKit/Sources/Component/LiveInfo/RoomInfoService

public func unfollowUser(userId: String) {
    V2TIMManager.sharedInstance().unfollowUser([userId]) { [weak self]
followResultList in
        guard let self = self, let result = followResultList?.first
else { return }
        if result.resultCode == 0 {
            self.state.followingList = state.followingList.filter {
$0.userId != userId}
        }
```

```
    } fail: { code, message in
        debugPrint("[RoomInfo] unfollowUser failed, error:\(code),
message:\(String(describing: message))")
    }
}
```

查看主播关注状态

Android

Java

```
// 文件位置:
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/service/impl/
LiveServiceImpl.java

List<String> userIDList = new ArrayList<>();
userIDList.add("userId");

V2TIMManager.getFriendshipManager().checkFollowType(userIDList,
        new V2TIMValueCallback<List<V2TIMFollowTypeCheckResult>>() {
            @Override
            public void onSuccess(List<V2TIMFollowTypeCheckResult>
results) {
            }

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

IOS

```
// 文件位置:
TUILiveKit/iOS/TUILiveKit/Sources/Component/LiveInfo/RoomInfoService
```

```
public func isFollow(userId: String) {
    let userInfo = TUIUserInfo()
    userInfo.userId = userId
    V2TIMManager.sharedInstance().checkFollowType([state.ownerId]) {
[weak self] checkResultList in
        guard let self = self, let result = checkResultList?.first
else { return }
        if result.followType == .FOLLOW_TYPE_IN_BOTH_FOLLOWERS_LIST ||
result.followType == .FOLLOW_TYPE_IN_MY_FOLLOWING_LIST {
            if !self.state.followingList.contains(where: { $0.userId
== userId }) {
                self.state.followingList.insert(userInfo)
            }
        } else {
            self.state.followingList = state.followingList.filter {
$0.userId != userId}
        }
    } fail: { code, message in
        debugPrint("[RoomInfo] isFollow failed, error:\(code),
message:\(String(describing: message))")
    }
}
```

获取粉丝数量

Android

Java

```
// 文件位置:
Android/tuilivekit/src/Main/java/com/trtc/uikit/livekit/service/impl/
LiveServiceImpl.java

List<String> userIDList = new ArrayList<>();
userIDList.add("userId");

V2TIMManager.getFriendshipManager().getUserFollowInfo(userIDList,
        new V2TIMValueCallback<List<V2TIMFollowInfo>>() {
            @Override
            public void onSuccess(List<V2TIMFollowInfo> results) {
```

```
            }

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

iOS

```
// 文件位置：
TUILiveKit/iOS/TUILiveKit/Sources/Component/LiveInfo/RoomInfoService

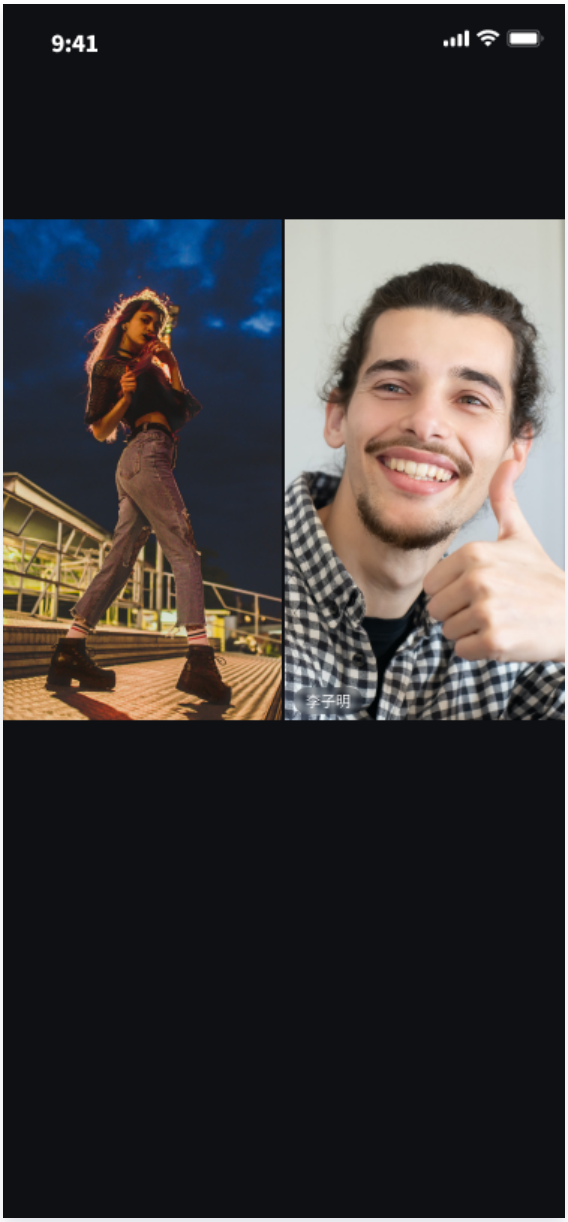
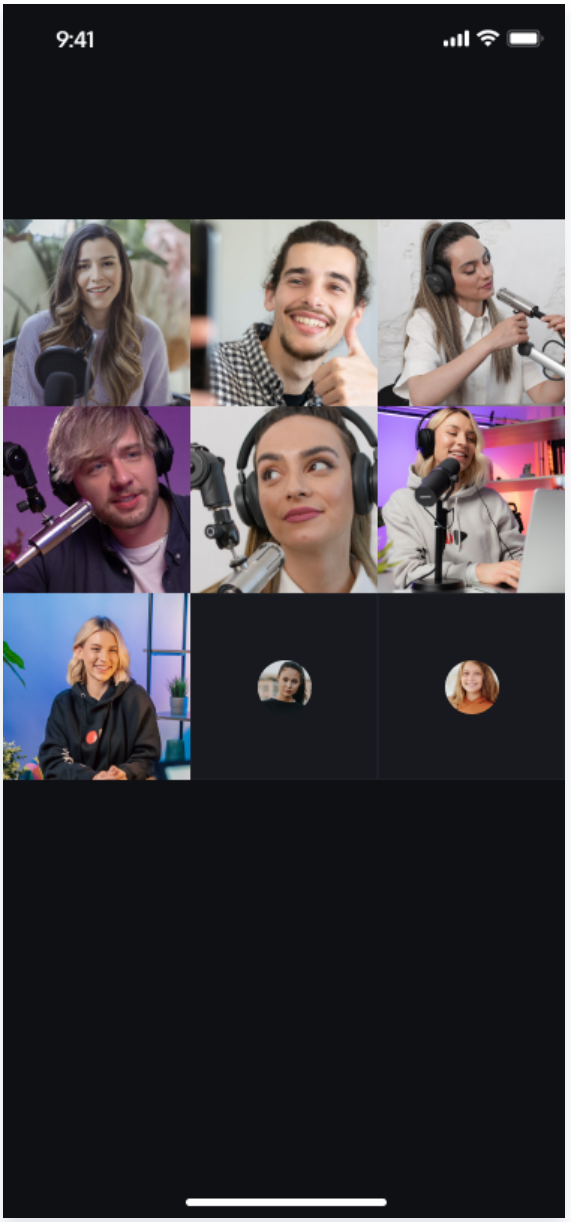
public func getFansNumber() {
    V2TIMManager.sharedInstance().getUserFollowInfo([state.ownerId]) {
[weak self] followInfoList in
        guard let self = self, let followInfo = followInfoList?.first
else { return }
        self.state.fansNumber = Int(followInfo.followersCount)
    } fail: { code, message in
        debugPrint("[RoomInfo] getFansNumber failed, error:\(code),
message:\(String(describing: message))")
    }
}
```

主播连线（TUILiveKit）

最近更新时间：2025-04-03 16:12:02

功能介绍

连线功能是一种实时互动交流方式，专门为主播量身定制。它让来自不同房间的主播在直播过程中实现实时互动与交流，TUILiveKit 的连线功能可支持单房间内**多达9人**同时连线，为主播互动、知识分享、文化交流、电商培训等多种场景提供了强大的技术支持，不仅为主播提供多样化的互动方式，也为观众带来更加丰富和深入的观看体验，从而为双方创造更多的惊喜和价值，使直播更具吸引力。

双人连线	多人连线
	

接入流程

主播连线流程

TUILiveKit 主播连线功能主要是基于 LiveCoreView 实现的，您可以按如下 API 来完成主播连线功能。这里以主播 A 与主播 B 连线为例实现如下。

> **说明：**
> - 以下是由 LiveCoreView 提供的主动调用方法。
> - 所有的回调方法是指 LiveCoreView 设置的 ConnectionObserver 对象中的回调方法。

主播 A 发起连线

- 主播 A 通过调用 `requestCrossRoomConnection` 发起连线，在参数 roomId 中传入需要连线的主播 B 房间 id。

Android

Kotlin

```
val roomId = "anchorBRoomId"
mLiveViewList.requestCrossRoomConnection(roomId, 10, null)
```

java

```
String roomId = "anchorBRoomId";
mLiveViewList.requestCrossRoomConnection(roomId, 10, null);
```

iOS

Swift

```
let roomId = "anchorRoomId"

liveCoreView.requestCrossRoomConnection(roomId: roomId, timeOut: 60) {
} onError: { code, message in
```

```
}
```

OC

```
NSString *roomId = @"anchorRoomId";

[liveCoreView requestCrossRoomConnection:roomId
                                 timeOut:60
                               onSuccess:^(void) {
                               } onError:^(NSInteger code, NSString *
_Nonnull message) {
                               }];
```

- 主播 A 可通过 `onCrossRoomConnectionAccepted` 接收请求同意回调。

主播 B 收到连线请求

- 主播 B 通过 `onCrossRoomConnectionRequest` 接收连线请求回调。

Android

Kotlin

```
override fun onCrossRoomConnectionRequest(inviterUser: UserInfo) {
    Log.i(TAG, "收到主播A连线房间请求：${inviterUser.userId}")
}
```

java

```
@Override
public void onCrossRoomConnectionRequest(LiveStreamDefine.RoomInfo
roomInfo) {
    Log.i(TAG, "收到主播A连线房间请求：" + roomInfo.roomId);
}
```

iOS

Swift

```
func onCrossRoomConnectionRequest(hostUser: TUIConnectionUser) {
    print("收到主播A连线房间请求：\(hostUser.userId)")
}
```

OC

```
- (void)onCrossRoomConnectionRequest:(TUIConnectionUser *)hostUser {
    NSLog(@"收到主播A连线房间请求：%@", hostUser.userId);
}
```

- 主播 B 通过调用 `respondToCrossRoomConnection` 响应连线请求。

Android

Kotlin

```
liveCoreView.respondToCrossRoomConnection(roomId, true, null)
```

java

```
// 同意连线请求
liveCoreView.respondToCrossRoomConnection(roomId, true, null);
```

iOS

Swift

```
liveCoreView.respondToCrossRoomConnection(roomId: roomId, isAccepted:
true) {
} onError: { code, message in
}
```

OC

```
[liveCoreView respondToCrossRoomConnection:roomId
                                isAccepted:true
                                 onSuccess:^(void) {
                                 } onError:^(NSInteger code, NSString
* _Nonnull message) {
                                 }];
```

- **主播 A, B以及房间内观众收到 `onConnectedRoomsUpdated` 回调，收到接收连线列表发生变化通知。**

Android

Kotlin

```
override fun onConnectedRoomsUpdated(inviterUser: UserInfo) {
    Log.i(TAG, "主播连线房间列表更新")
}
```

java

```
@Override
public void onConnectedRoomsUpdated(List<LiveStreamDefine.RoomInfo>
roomList) {
    Log.i(TAG, "主播连线房间列表更新");
}
```

iOS

Swift

```
func onConnectedRoomsUpdated(hostUserList: [TUIConnectionUser]) {
    print("主播连线房间列表更新")
}
```

OC

```
- (void)onConnectedRoomsUpdated:(NSArray<TUIConnectionUser *>
*)hostUserList {
    NSLog(@"主播连线房间列表更新");
}
```

退出连线流程

- 主播 B 调用 `terminateIntraRoomConnection` 退出连线。

Android

kotlin

```
public void disconnect(TUIRoomDefine.ActionCallback callback) {
    mTUILiveConnectionManager.disconnect(callback);
}
```

java

```
liveCoreView.terminateIntraRoomConnection()
```

iOS

Swift

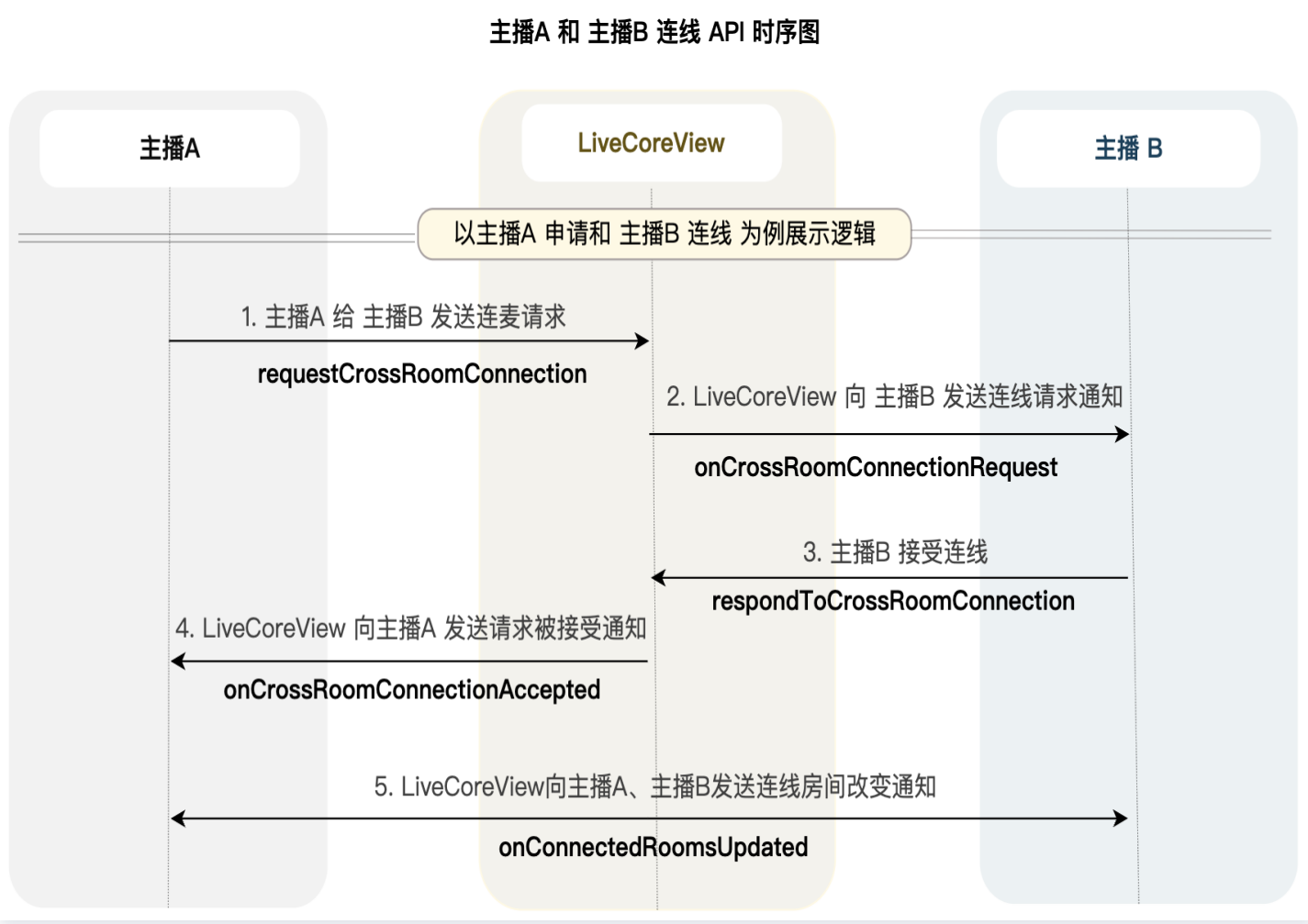
```
liveCoreView.terminateCrossRoomConnection()
```

OC

```
[liveCoreView terminateCrossRoomConnection];
```

> **说明：**
> 连线功能是基于 LiveCoreView 实现，若您需要扩展连线功能，可参见 LiveCoreView 文档。

主播连线时序图



版权所有：腾讯云计算（北京）有限责任公司

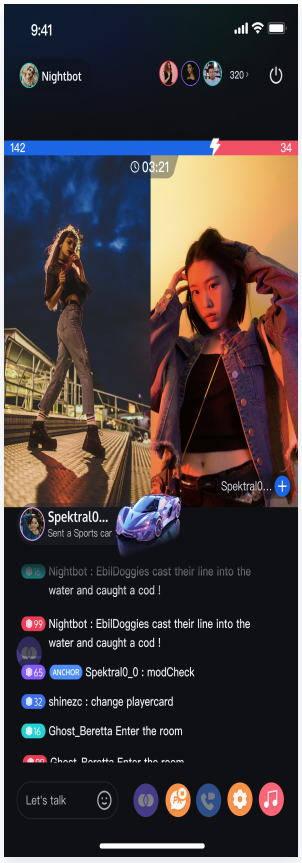
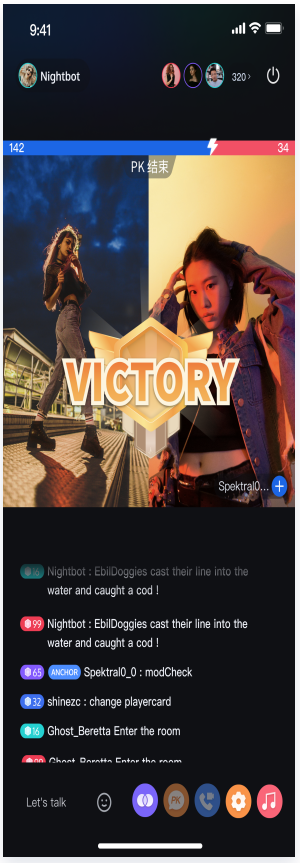
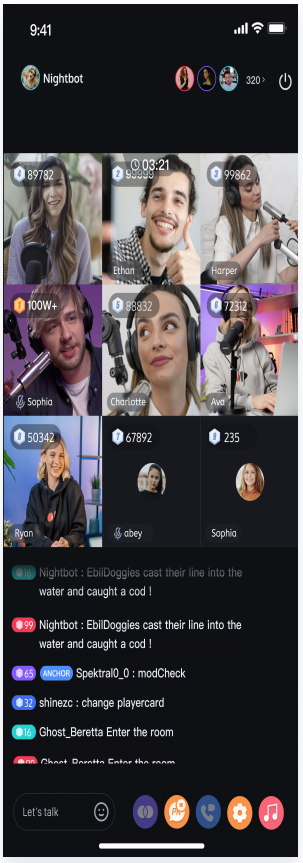
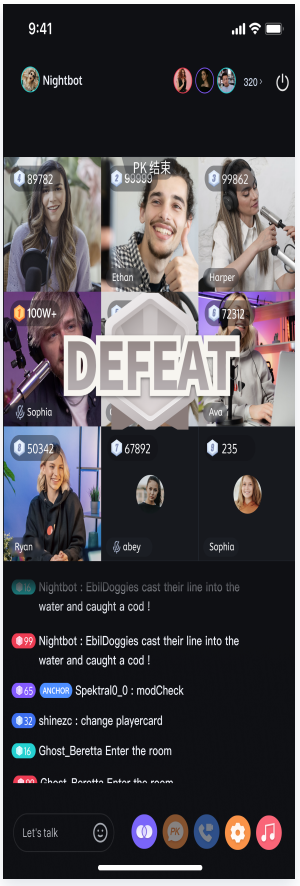
主播 PK（TUILiveKit）

最近更新时间：2025-04-03 16:12:02

> **说明：**
> PK 上新，限时免费体验。

功能介绍

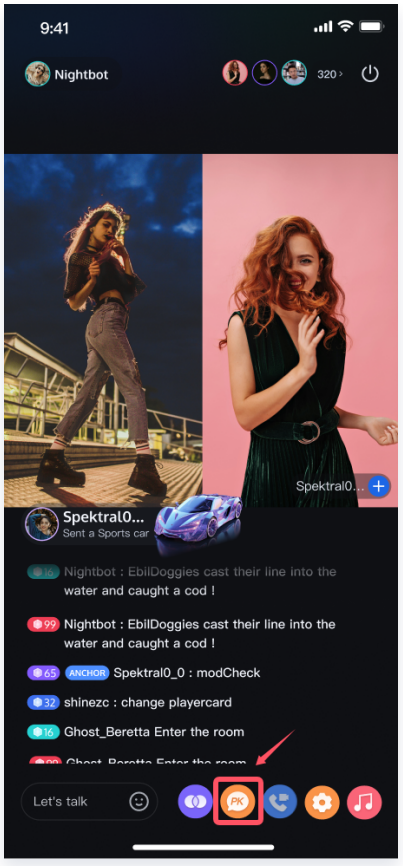
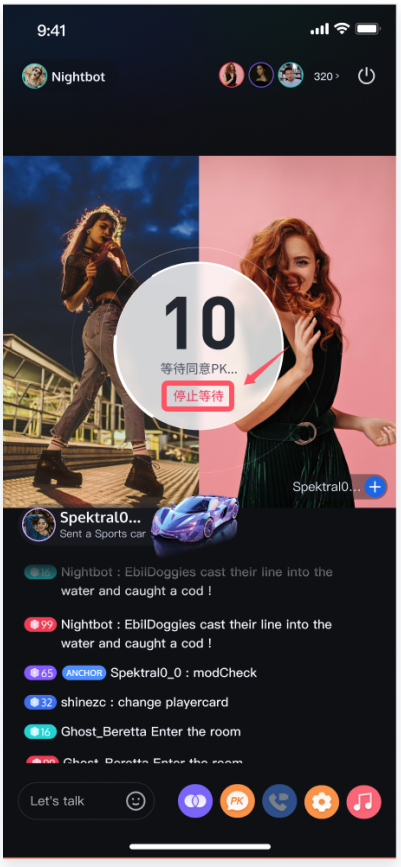
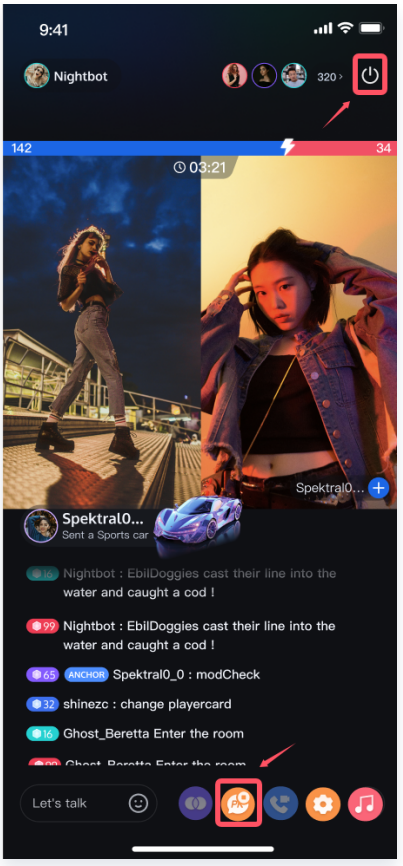
PK 功能是一种激烈而富有趣味的实时互动方式，专为主播和观众设计。它允许来自不同房间的主播进行实时对抗，增加了直播的竞争性和观赏性。TUILiveKit 的 PK 功能支持**多达9位**主播同时参与对战，为主播提供了一个展示才华、比拼魅力的平台，同时也为观众带来了紧张刺激的观看体验。无论是才艺比拼、知识问答还是游戏竞技，PK 功能都能为主播和观众创造更多的互动机会，**激发观众的参与热情**，**提升直播的趣味性和吸引力**，从而为双方带来更多的惊喜和价值，推动直播内容的多样化发展。

双人 PK 中	双人 PK 结果	多人 PK 中	多人 PK 结果
			

使用说明

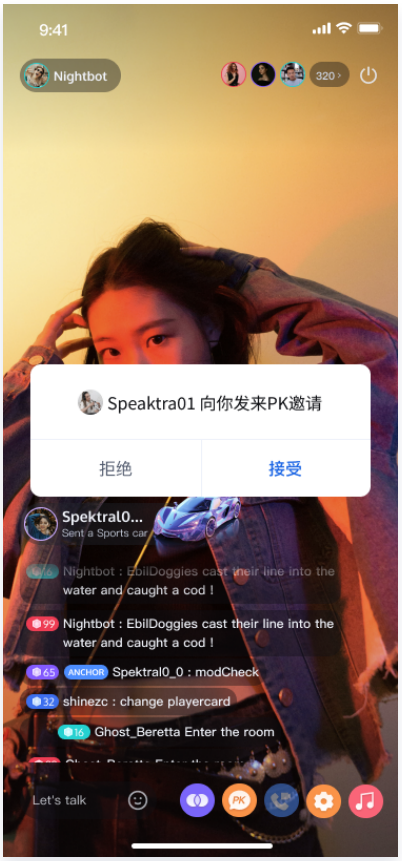
> **说明：**
> 在您准备发起 PK 或接受 PK 前，请确保您处于连线状态中。

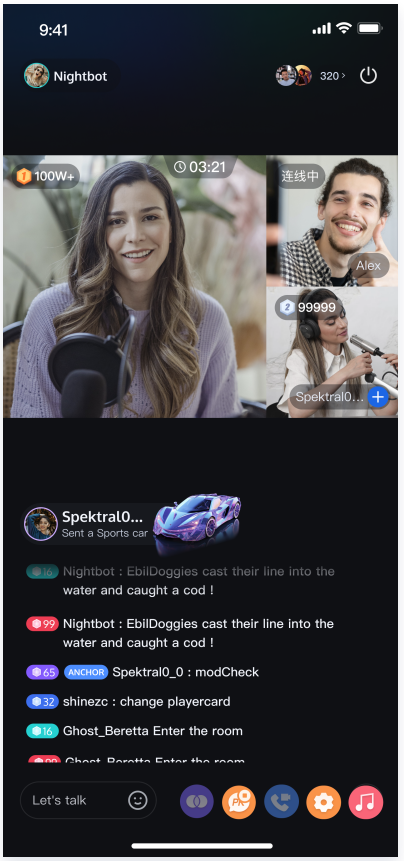
主播发起 PK

点击 PK 按钮	停止 PK 等待	断开 PK
		

主播接收 PK

主播收到 PK 邀请	主播接受 PK





功能定制

自定义 PK 等待倒计时样式

如您需要自定义 PK 等待倒计时样式，请参考以下路径更改：

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/view/liveroom/view/anchor/component/livestreaming/battle/

├── BattleCountdownBackView.java     // PK 等待倒计时背景样式
└── BattleCountdownView.java         // PK 等待倒计时前景样式
```

iOS

```
// 文件位置：iOS/TUILiveKit/Sources/LiveStream/HostBattle/View

View
  ├── BattleCountDownBackgroundView.swift     // PK 等待倒计时背景样式
  └── BattleCountDownView.swift               // PK 等待倒计时前景样式
```

自定义双人 PK 比分样式

如您需要自定义双人 PK 比分样式，请参考以下路径更改：

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/view/liveroom/view/com
mon/battle/SingleBattleScoreView.java

public class SingleBattleScoreView extends FrameLayout {
    ...
}
```

iOS

```
// 文件位置：
iOS/TUILiveKit/Sources/LiveStream/HostBattle/View/SingleBattleScoreVie
w.swift

class SingleBattleScoreView: UIView {
    ...
    func constructViewHierarchy() {
        // 视图层级构建
    }

    func activateConstraints() {
        // 视图layout布局
    }
```

```
}
```

自定义多人 PK 比分样式

如您需要自定义多人 PK 比分样式，请参考以下路径更改：

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/view/liveroom/view/com
mon/battle/BattleMemberInfoView.java

public class BattleMemberInfoView extends FrameLayout {
    ...
}
```

iOS

```
// 文件位置：
iOS/TUILiveKit/Sources/LiveStream/HostBattle/View/BattleMemberInfoView
.swift

class BattleMemberInfoView: UIView {
    ...
    func constructViewHierarchy() {
        // 视图层级构建
    }

    func activateConstraints() {
        // 视图layout布局
    }
}
```

自定义 PK 比分结果样式

如您需要自定义 PK 比分结果样式，请参考以下路径更改：

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/view/liveroom/view/common/battle/BattleInfoView.java

public class BattleInfoView extends BasicView {
    ...
    private void showBattleResult(int type) {
        // PK结果展示
    }
}
```

iOS

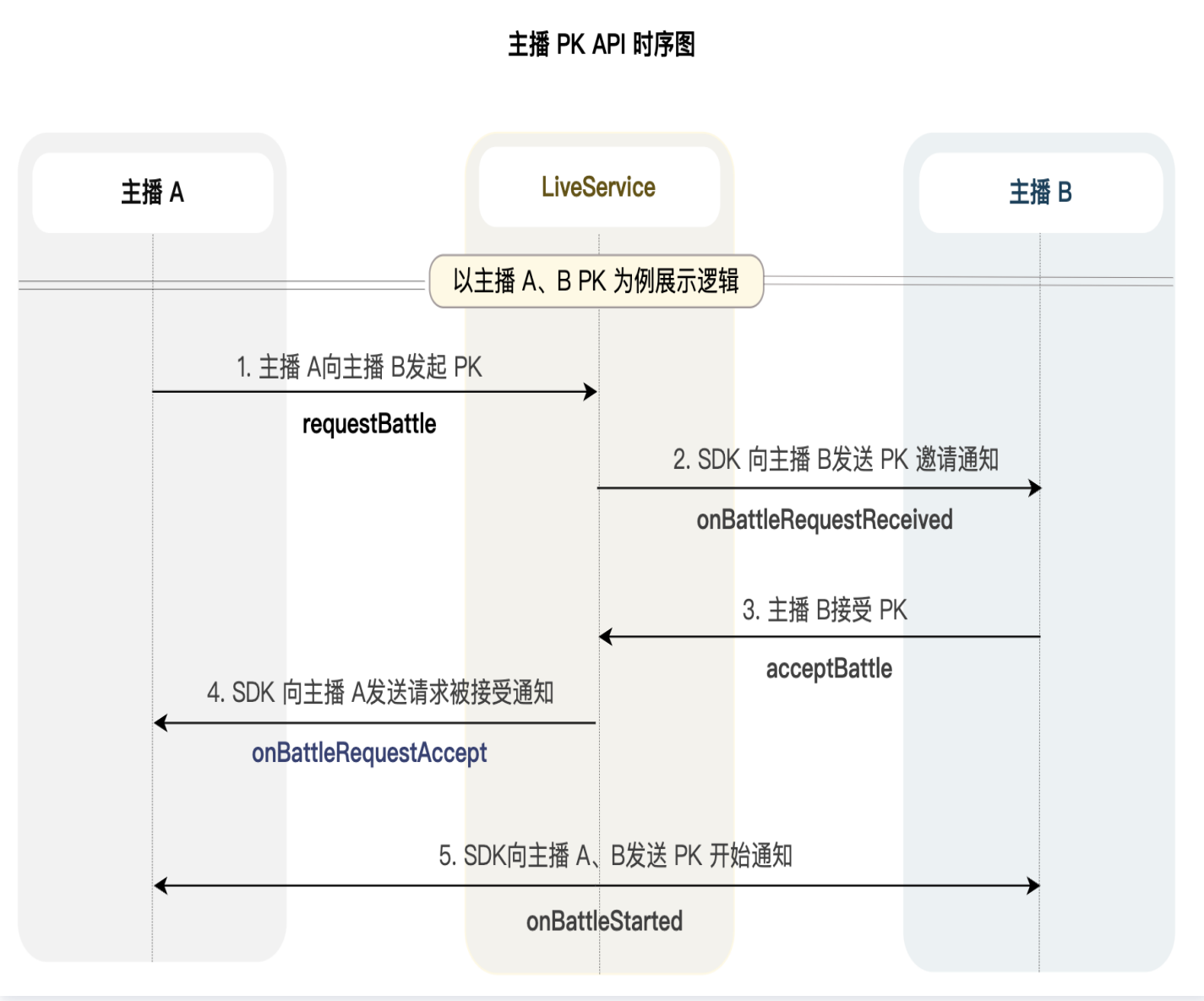
```
// 文件位置：
iOS/TUILiveKit/Sources/LiveStream/HostBattle/View/BattleInfoView.swift

class BattleInfoView: UIView {
    ...
    func showBattleResult(store: LiveStore) {
        // PK结果展示
    }
}
```

关键代码

主播 PK

TUILiveKit 主播 PK 功能主要是基于 `LiveService` 实现的，在 `LiveService` 中，您可通过 `mTUIRoomEngine.getLiveBattleManager()` 获取到 PK 管理类对象，进而调用 PK 相关 API 函数，实现 PK 功能。以主播 A 和主播 B 的 PK 为例，具体交互时序可参考下图。



说明：
邀请多人参加 PK 时，若被邀请方中有人接受了 PK，则仅 PK 发起方和 PK 邀请接受方和对应房间内观众会收到 `onBattleStarted` 回调。

主播 A 发起 PK

- 主播 A 通过调用 `requestBattle` 发起 PK，在参数 `config` 中传入 PK 最大时长、是否需要邀请方回复同意/拒绝，在参数 `userIdList` 中传入主播 B 的 userId，在参数 `timeout` 中传入 PK 邀请等待时长。

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/controller/Bat
tleController.java

public void requestBattle(List<String> roomIdList, int timeout) {
    TUILiveBattleManager.BattleConfig config = new
TUILiveBattleManager.BattleConfig();
    config.duration = BattleState.BATTLE_DURATION;
    config.needResponse = mBattleState.mNeedResponse;
    config.extensionInfo = "";
    mLiveService.requestBattle(config, roomIdList, timeout, new
TUILiveBattleManager.BattleRequestCallback() {
        @Override
        public void onSuccess(TUILiveBattleManager.BattleInfo
battleInfo,
                              Map<String,
TUILiveBattleManager.BattleCode> map) {
            mBattleState.mBattleId = battleInfo.battleId;
            mBattleState.mBattleConfig.copy(config);
            List<BattleState.BattleUser> sendRequests =
mBattleState.mSentBattleRequests.get();
            for (Map.Entry<String, TUILiveBattleManager.BattleCode>
entry : map.entrySet()) {
                String key = entry.getKey();
                TUILiveBattleManager.BattleCode code =
entry.getValue();
                if (code == TUILiveBattleManager.BattleCode.SUCCESS) {
                    for (ConnectionState.ConnectionUser user :
mConnectionState.connectedUsers.get()) {
                        if (TextUtils.equals(user.userId, key)) {
                            sendRequests.add(new
BattleState.BattleUser(user));
                            break;
                        }
                    }
                } else {

notifyToast(convertCodeToString(entry.getValue()));
                }
            }
            mBattleState.mSentBattleRequests.set(sendRequests);
        }
```

```
        @Override
        public void onError(TUICommonDefine.Error error, String s) {
            ErrorHandler.onError(error);
            mBattleState.mSentBattleRequests.clear();
        }
    });
}
```

iOS

```
// 文件位置:
iOS/TUILiveKit/Sources/LiveStream/HostBattle/API/LSBattleService.swift

func requestBattle(config: TUIBattleConfig, userIdList: [String],
timeout: TimeInterval) async throws -> (battleInfo: TUIBattleInfo,
resultMap: [String : TUIBattleCode]) {
    return try await withCheckedThrowingContinuation { [weak self]
continuation in
        guard let self = self else { return }
        battleManager.requestBattle(config: config, userIdList:
userIdList, timeout: timeout) { battleInfo, resultMap in
            var battleResult: [String: TUIBattleCode] = [:]
            resultMap.forEach { (key: String, value: NSNumber) in
                battleResult[key] = TUIBattleCode(rawValue:
value.intValue) ?? .unknown
            }
            continuation.resume(returning: (battleInfo, battleResult))
        } onError: { err, message in
            let error = InternalError(error: err, message: message)
            continuation.resume(throwing: error)
        }
    }
}
```

- 主播 A 可通过 `onBattleRequestAccept` 接收请求同意回调。

主播收到 PK 请求

- 主播 B 通过 `onBattleRequestReceived` 接收 PK 请求回调。

Android

```
// 文件位置:
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/observer/LiveB
attleManagerObserver.java

@Override
public void onBattleRequestReceived(BattleInfo battleInfo, BattleUser
inviter, BattleUser invitee) {
    LiveKitLog.info(mTag + " onBattleRequestReceived:[battleInfo:" +
new Gson().toJson(battleInfo)
        + ", inviter:" + new Gson().toJson(inviter) + ", invitee:" +
new Gson().toJson(invitee) + "]");
    mBattleController.onBattleRequestReceived(battleInfo, inviter);
}
```

iOS

```
// 文件位置:
iOS/TUILiveKit/Sources/LiveStream/HostBattle/Module/LSBattleManagerObs
erver

func onBattleRequestReceived(battleInfo: TUIBattleInfo, inviter:
TUIBattleUser, invitee: TUIBattleUser) {
    manager?.onBattleRequestReceived(battleInfo: battleInfo, inviter:
inviter, invitee: invitee)
}
```

- 主播 B 通过调用 `accept` 接受 PK 请求。

Android

```
// 文件位置:
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/controller/Bat
tleController.java

public void accept() {
    mLiveService.acceptBattle(mBattleState.mBattleId, new
TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {

        }

        @Override
        public void onError(TUICommonDefine.Error error, String s) {

        }
    });
}
```

iOS

```
// 文件位置:
iOS/TUILiveKit/Sources/LiveStream/HostBattle/API/LSBattleService.swift

func acceptBattle(battleId: String) async throws {
        return try await withCheckedThrowingContinuation { [weak self]
continuation in
        guard let self = self else { return }
        battleManager.acceptBattle(battleId: battleId) {
            continuation.resume()
        } onError: { err, message in
            let error = InternalError(error: err, message: message)
            continuation.resume(throwing: error)
        }
    }
}
```

- 主播 A，B 以及房间内观众可通过 `onBattleStarted` 接收 PK 开始回调。

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/observer/LiveBattleManagerObserver.java

@Override
public void onBattleStarted(BattleInfo battleInfo) {
    LiveKitLog.info(mTag + " onBattleStarted:[battleInfo:" + new Gson().toJson(battleInfo) + "]");
    mBattleController.onBattleStarted(battleInfo);
}
```

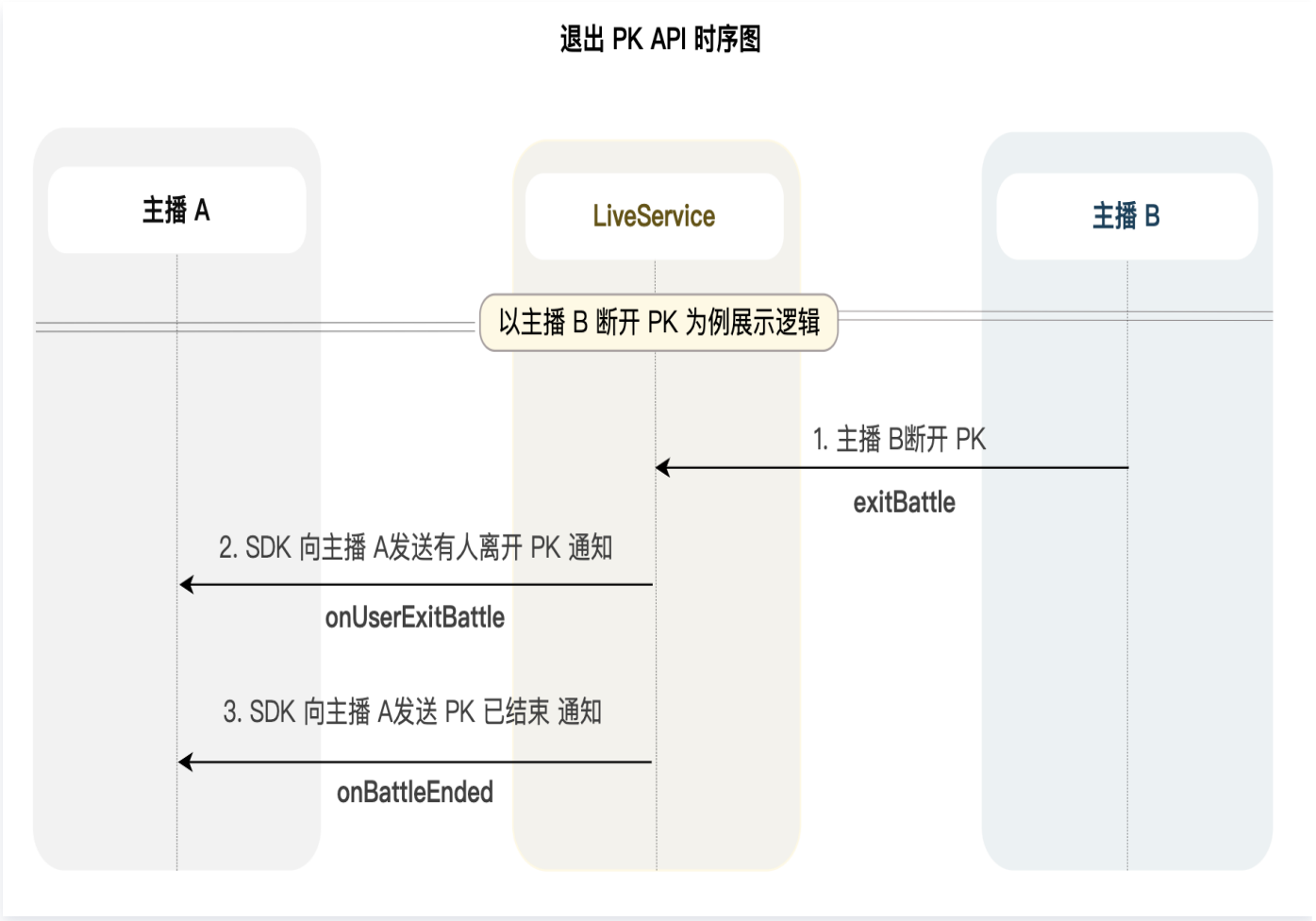
iOS

```
// 文件位置：
iOS/TUILiveKit/Sources/LiveStream/HostBattle/Module/LSBattleManagerObserver

func onBattleStarted(battleInfo: TUIBattleInfo) {
    manager?.onBattleStarted(battleInfo: battleInfo)
}
```

主播退出 PK

- 以主播 B 退出 PK 为例，交互时序可参考下图。



说明：

多人PK 时：

- 当有主播退出 PK，则其余 PK 中主播和对应房间内观众会收到 `onUserExitBattle` 回调。
- 当 PK 到达预设 PK 时间，则 PK 中主播和对应房间内观众会收到 `onBattleEnded` 回调。
- 当 PK 中主播人数为2，且有主播退出 PK 时， PK 中主播和对应房间内观众会收到 `onBattleEnded` 回调。

- 主播 B 调用 `exitBattle` 退出 PK。

Android

```
// 文件位置：
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/controller/BattleController.java
```

```
public void exitBattle() {
    mLiveService.exitBattle(mBattleState.mBattleId, new
TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {
            mBattleState.mSentBattleRequests.clear();
            mBattleState.mBattledUsers.clear();
            removeBattleRequestReceived();
        }

        @Override
        public void onError(TUICommonDefine.Error error, String s) {

        }
    });
}
```

iOS

```
// 文件位置：
iOS/TUILiveKit/Sources/LiveStream/HostBattle/API/LSBattleService.swift

func exitBattle(battleId: String) async throws {
    return try await withCheckedThrowingContinuation { [weak self]
continuation in
        guard let self = self else { return }
        battleManager.exitBattle(battleId: battleId) {
            continuation.resume()
        } onError: { err, message in
            let error = InternalError(error: err, message: message)
            continuation.resume(throwing: error)
        }
    }
}
```

- 主播 A，B 以及房间内观众收到 `onBattleEnded` 回调，收到 PK 结束通知。

Android

```
// 文件位置:
tuilivekit/src/main/java/com/trtc/uikit/livekit/manager/observer/Li
veBattleManagerObserver.java

@Override
public void onBattleEnded(BattleInfo battleInfo,
BattleStoppedReason reason) {
    LiveKitLog.info(mTag + " onBattleEnded:[battleInfo:"
        + new Gson().toJson(battleInfo) + ", reason:" + reason +
"]");
    mBattleController.onBattleEnded(battleInfo);
}
```

iOS

```
// 文件位置:
iOS/TUILiveKit/Sources/LiveStream/HostBattle/Module/LSBattleManagerObs
erver

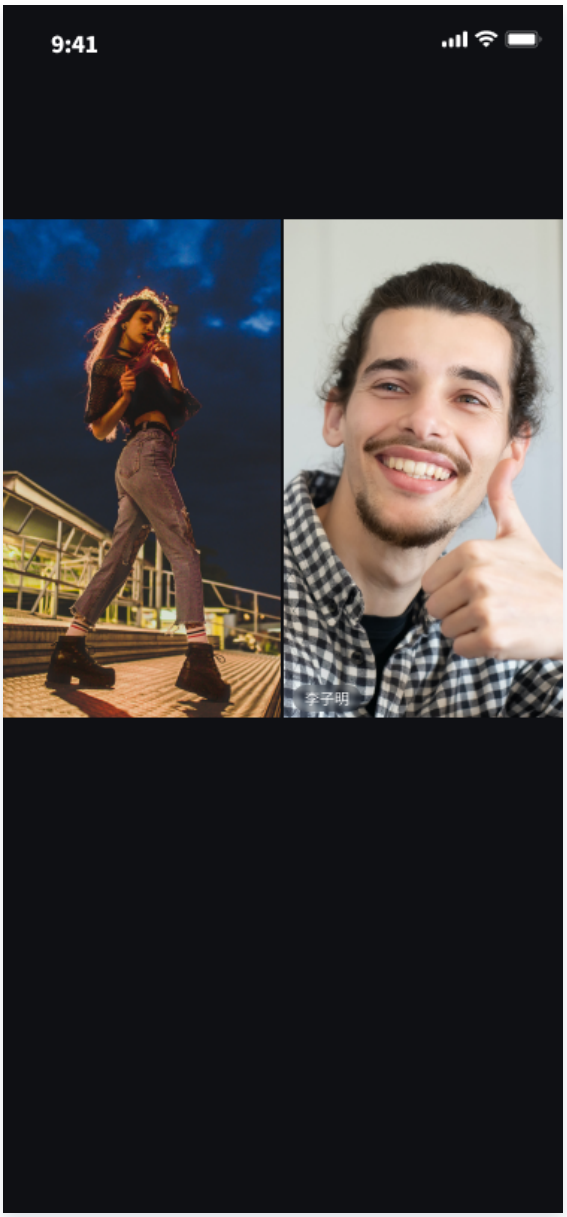
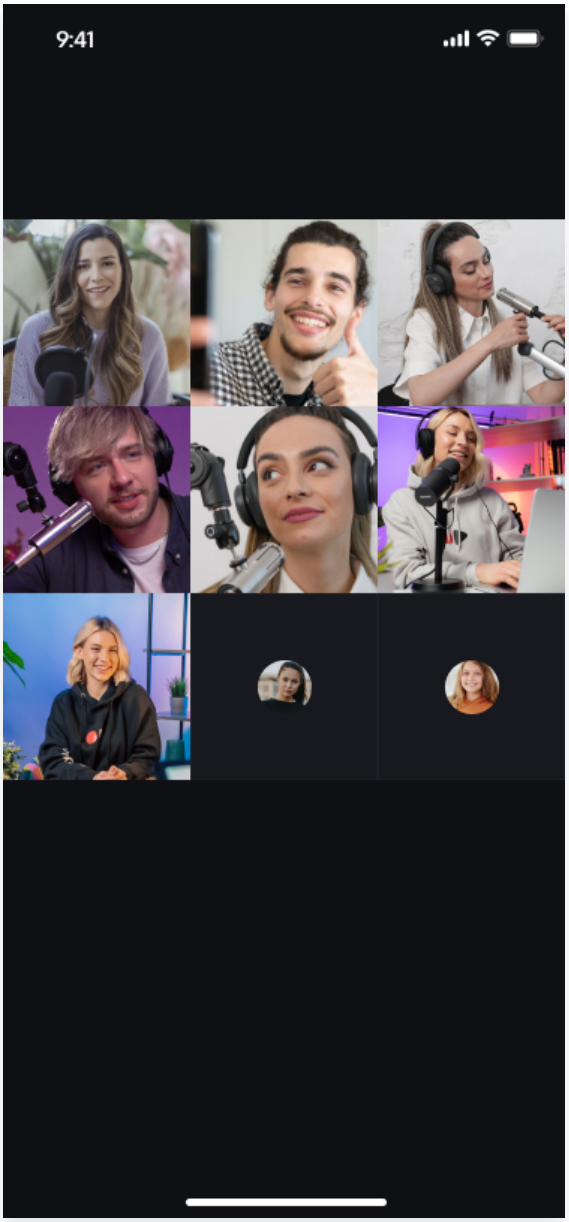

func onBattleEnded(battleInfo: TUIBattleInfo, reason:
TUIBattleStoppedReason) {
    manager?.onBattleEnded(battleInfo: battleInfo, reason: reason)
}
```

观众连麦（TUILiveKit）

最近更新时间：2025-04-14 18:02:42

功能介绍

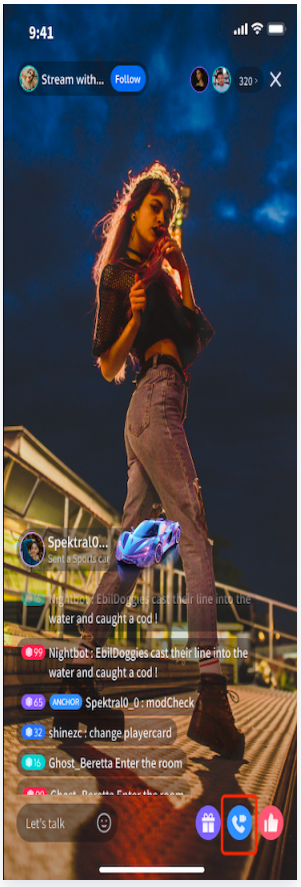
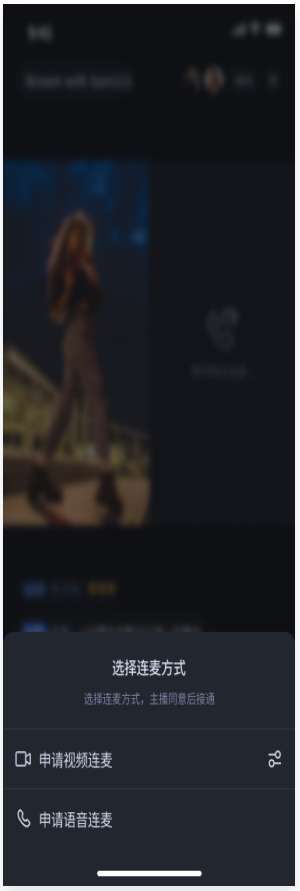
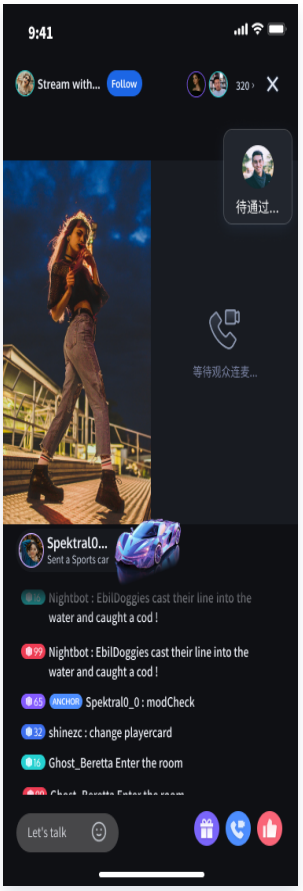
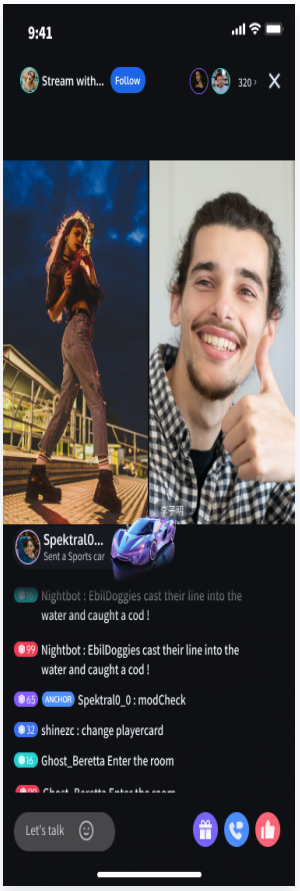
观众连麦功能是一种实时互动交流方式，通过观众连麦功能，主播可以与**多达9个**观众进行实时互动，无论是解答问题、分享经验还是进行娱乐互动，都能极大地提升观众的**参与感**和**满意度**。这种直接的互动和交流为商业化运营提供了更加便捷和高效的渠道，同时也为观众提供了更加个性化和定制化的购物体验。观众连麦功能适用于**多个场景**，包括**电商直播、娱乐直播、在线教学**等。

单人连麦	多人连麦
	

使用说明

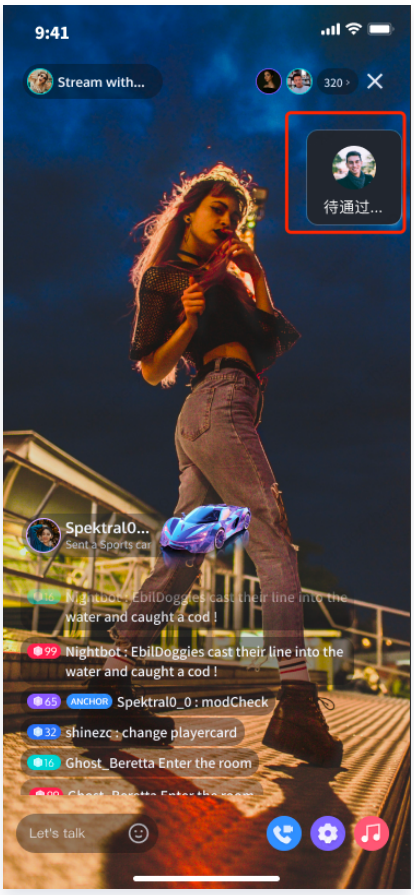
> **说明：**
> 此说明介绍在TUILiveKit中的观众连麦功能使用说明，如您未使用TUILiveKit而直接使用了 LiveCoreView 或 RTCRoomEngine，可忽略此步骤，直接跳转至接入流程或相应的API文档。

观众发起连麦申请

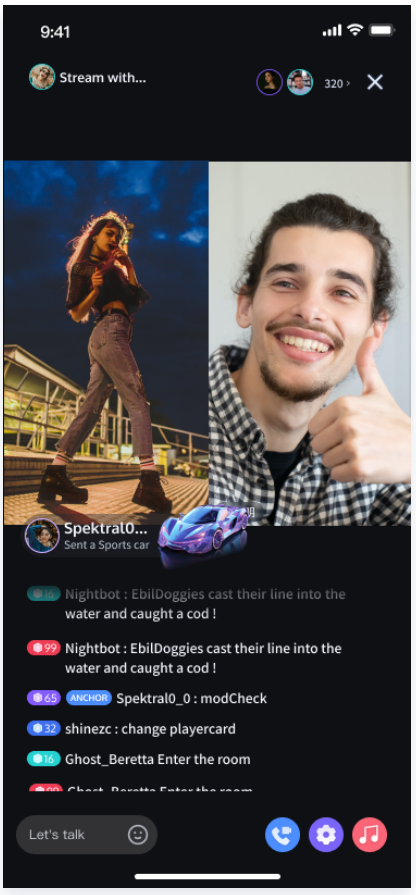
点击连麦请求按钮	选择连麦方式	发送连麦请求，等待主播同意	主播同意后，连麦成功
			

主播处理观众连麦请求

收到观众的连麦请求	点击连麦用户，打开连麦面板	点击同意后，连麦成功







接入流程

观众连麦流程

TUILiveKit 观众连麦功能 主要是通过 LiveCoreView 实现（除Flutter），您可以调用如下 API 函数，实现观众连麦功能。以观众 B 申请和主播 A 连麦为例实现如下。

> **说明：**
> - 以下是由 LiveCoreView 提供的主动调用方法。
> - 所有的回调方法是指 LiveCoreView 设置的 ConnectionObserver 对象中的回调方法。

在Flutter TUILiveKit中，观众连麦功能直接通过 RTCRoomEngine 实现。以下所有Flutter相关内容在是TUILiveKit中，调用由RTCRoomEngine提供的方法以实现对应功能供您参考。

观众发送连麦请求

观众 B 给主播 A 发送连麦请求。

Android

Kotlin

```
val userId = "anchorUserId";    // 修改为房主的UserId,传入空字符串时默认为房主UserId
val timeout = 60;
liveCoreView.requestIntraRoomConnection(userId, 10, null)
```

Java

```
String userId = "anchorUserId";  // 修改为房主的UserId,传入空字符串时默认为房主UserId
int timeout = 60;
liveCoreView.requestIntraRoomConnection(userId, timeout, true, null);
```

iOS

Swift

```
let timeout = 60
let userId = "anchorUserId"  // 修改为房主的UserId,传入空字符串时默认为房主UserId
liveCoreView.requestIntraRoomConnection(userId: userId, timeOut: timeOut, openCamera: true) {
} onError: { code, message in
}
```

OC

```
NSInteger timeout = 60;
NSString userId = "anchorUserId"  // 修改为房主的UserId,传入空字符串时默认为房主UserId
[liveCoreView requestIntraRoomConnection:""
                                 timeOut:timeout
```

```
                                onSuccess:^(void) {
                                } onError:^(NSInteger code, NSString *
_Nonnull message) {
                                }];
```

Flutter

Dart

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/live_stream/service/impl/room_engine_se
rvice.dart

@override
TUIRequest takeSeat(int seatIndex, int timeout, TUIRequestCallback?
requestCallback) {
  return roomEngine.takeSeat(seatIndex, timeout, requestCallback);
}
```

主播端收到连麦请求

主播 A 会在 onUserConnectionRequest 回调方法中收到 观众B 的连麦请求。

Android

Kotlin

```
override fun onUserConnectionRequest(inviterUser: UserInfo) {
    Log.i(TAG, "收到观众连线请求: ${inviterUser.userId}")
}
```

Java

```
@Override
```

```
public void onUserConnectionRequest(LiveStreamDefine.LiveUser
inviterUser) {
    Log.i(TAG, "收到观众连线请求：" + inviterUser.userId);
}
```

iOS

Swift

```
func onUserConnectionRequest(inviterUser: TUIUserInfo) {
    print("收到观众连线请求：\(inviterUser.userId)")
}
```

OC

```
- (void)onUserConnectionRequest:(TUIUserInfo *)inviterUser {
    NSLog(@"收到观众连线请求：%@", hostUser.userId);
}
```

Flutter

在Flutter中，主播 A 会在 `RTCRoomEngine` 的 onRequestReceived 回调中收到观众 B 的连麦请求。

Dart

```
// 文件位置：
TUILiveKit/Flutter/livekit/lib/live_stream/manager/observer/live_obser
ver.dart

super.onRequestReceived = (request) {
  LiveKitLogger.info("$tag onRequestReceived:[request:$request");
  liveController.seatController.onRequestReceived(request);
};
```

主播响应连麦请求

主播 A 收到观众的连麦请求后，可以调用 respondIntraRoomConnection 响应观众 B 是否同意连麦。

Android

Kotlin

```
// 主播同意连麦
liveCoreView.respondIntraRoomConnection(audienceBUserId, true, null)
```

Java

```
// 主播同意连麦
liveCoreView.respondIntraRoomConnection(userId, true, null);

// 主播拒绝连麦
liveCoreView.respondIntraRoomConnection(userId, false, null);
```

iOS

Swift

```
// 主播同意连麦
liveCoreView.respondIntraRoomConnection(userId: audienceBUserId,
isAccepted: true) {
} onError: { code, message in
}
```

OC

```
// 主播同意连麦
[liveCoreView respondIntraRoomConnection:audienceBUserId
                              isAccepted:YES
                               onSuccess:^(void) {
```

```
                                } onError:^(NSInteger code, NSString *
_Nonnull message) {
                                }];
```

Flutter

Dart

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/live_stream/service/impl/room_engine_se
rvice.dart

mTUIRoomEngine.responseRemoteRequest(requestId, true, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {}

    @Override
    public void onError(TUICommonDefine.Error error, String message)
{}
});
```

观众收到主播响应回调

主播 A 同意观众 B 连麦请求后， 观众 B 会通过 onUserConnectionAccepted 回调收到主播 A 同意连麦的回调。

Android

Kotlin

```
override fun onUserConnectionAccepted(inviterUser: UserInfo) {
    Log.i(TAG, "观众同意连线: ${inviterUser.userId}")
}
```

Java

```
@Override
public void onUserConnectionAccepted(LiveStreamDefine.LiveUser
liveUser) {
    Log.i(TAG, "观众同意连线：" + liveUser.userId);
}

@Override
public void onUserConnectionRejected(LiveStreamDefine.LiveUser
liveUser) {
    Log.i(TAG, "观众拒绝连线：" + liveUser.userId);
}
```

iOS

Swift

```
func onUserConnectionAccepted(userId: String) {
    print("观众同意连线：\(userId)")
}
```

OC

```
- (void)onUserConnectionAccepted:(NSString *)userId {
    NSLog(@"观众同意连线：%@", userId);
}
```

Flutter

在Flutter中，观众收到主播响应连麦申请的回调由takeSeat接口中的`requestCallback`给出。具体示例可以参考以下代码：

```
// 文件位置：
TUILiveKit/Flutter/livekit/lib/live_stream/manager/controller/seat_con
```

```
troller.dart

TUIRequest request = liveService.takeSeat(
        index,
        timeout,
        TUIRequestCallback(
          onAccepted: (String requestId, String userId) {
// 连麦申请被同意
          },
          onRejected: (String requestId, String userId, String
message) {   // 连麦申请被拒绝
          },
          onCancelled: (String requestId, String userId) {
// 连麦申请取消
          },
          onTimeout: (String requestId, String userId) {
// 连麦申请超时
          },
          onError: (String requestId, String userId, TUIError error,
String message) {  // 错误回调
          },
));
```

连麦用户列表发生变化的回调

主播 A 同意观众 B 连麦请求后， LiveCoreView 会同时给主播 A 和观众 B 发送连麦用户发生变化。

Android

Kotlin

```
override fun onConnectedUsersUpdated(inviterUser: UserInfo) {
    Log.i(TAG, "连麦用户列表发生变化")
}
```

Java

```
@Override
```

```
public void onConnectedUsersUpdated(List<UserInfo> userList,
List<UserInfo> joinList, List<UserInfo> leaveList) {
    Log.i(TAG, "连麦用户列表发生变化");
}
```

iOS

Swift

```
func onConnectedUsersUpdated(userList: [TUIUserInfo], joinList:
[TUIUserInfo], leaveList: [TUIUserInfo]) {
   print("连麦用户列表发生变化")
}
```

OC

```
- (void)onConnectedUsersUpdated:(NSArray<TUIUserInfo *> *)userList
                       joinList:(NSArray<TUIUserInfo *> *)joinList
                      leaveList:(NSArray<TUIUserInfo *> *)leaveList {
                          NSLog(@"连麦用户列表发生变化");         // 如果
需要，您可以在这里处理 userList、joinList 和 leaveList
                       }
```

Flutter

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/live_stream/manager/observer/live_obser
ver.dart

super.onSeatListChanged = (seatList, seatedList, leftList) {
      LiveKitLogger.info(
          "$tag($hashCode) onSeatListChanged:[seatList:$seatList
,seatedList:$seatedList,leftList:$leftList]");

liveController.target?.seatController.onSeatListChanged(seatList,
```

```
seatedList, leftList);
};
```

断开连麦流程

观众连麦成功后，主播挂断观众的连麦

观众 B 和主播 A 连麦成功后，主播 A 断开和观众 B 的连麦。

Android

Kotlin

```
val userId = "audienceBUserId"
liveCoreView.disconnectUser(userId, null)
```

Java

```
String userId = "audienceUserId";
liveCoreView.disconnectUser(userId, null);
```

iOS

Swift

```
let userId = "audienceBUserId"
liveCoreView.disconnectUser(userId: userId) {
} onError: { code, message in
}
```

OC

```
NSString *userId = @"audienceBUserId";
[liveCoreView disconnectUser:userId
```

```
                    onSuccess:^{
                    } onError:^(NSInteger code, NSString * _Nonnull
message) {
                    }];
```

Flutter

Dart

```
// 文件位置:
TUILiveKit/Flutter/livekit/lib/live_stream/service/impl/room_engine_se
rvice.dart

mLiveService.kickUserOffSeatByAdmin(0, userId, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {}

    @Override
    public void onError(TUICommonDefine.Error error, String message)
{}
});
```

观众收到主播断开连麦的回调

主播 A 断开和观众 B 的连麦请求后，观众 B 会收到 onUserConnectionTerminated 回调。

Android

Kotlin

```
override fun onUserConnectionTerminated(inviterUser: UserInfo) {
   Log.i(TAG, "主播关闭连线")
}
```

Java

```
@Override
public void onUserConnectionTerminated() {
    Log.i(TAG, "主播关闭连线");
}
```

iOS

Swift

```
func onUserConnectionTerminated() {
    print("主播关闭连线")
}
```

OC

```
- (void)onUserConnectionTerminated {
    NSLog(@"主播关闭连线");
}
```

Flutter

Flutter中，主播 A 断开和观众 B 的连麦请求后，观众 B 会收到在 `RTCRoomEngine` 的 onKickedOffSeat 回调。

```
// 文件位置：
TUILiveKit/Flutter/livekit/lib/live_stream/manager/observer/live_obser
ver.dart

super.onKickedOffSeat = (seatIndex, userInfo) {
      LiveKitLogger.info("$tag($hashCode) onKickedOffSeat:
[seatIndex:$seatIndex,userInfo:$userInfo");
      liveController.target?.seatController.onKickedOffSeat(seatIndex,
userInfo);
```

```
};
```

观众连麦成功后，观众结束连麦

观众 B 和主播 A 连麦成功后，观众 B 主动断开连麦可以调用 terminateIntraRoomConnection。

Android

Kotlin

```
liveCoreView.terminateIntraRoomConnection()
```

Java

```
liveCoreView.terminateIntraRoomConnection();
```

iOS

Swift

```
liveCoreView.terminateIntraRoomConnection()
```

OC

```
[liveCoreView terminateIntraRoomConnection]
```

Flutter

Dart

```
// 文件位置：
TUILiveKit/Flutter/livekit/lib/live_stream/service/impl/room_engine_se
rvice.dart

mLiveService.leaveSeat(new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {}

    @Override
    public void onError(TUICommonDefine.Error error, String message)
{}
});
```

主播收到观众断开连线的回调

当观众 B 主动断开连麦时，主播会收到 onUserConnectionExited 回调。

Android

Kotlin

```
override fun onUserConnectionExited(inviterUser:
LiveStreamDefine.LiveUser) {
   Log.i(TAG, "观众退出连线")
}
```

Java

```
@Override
public void onUserConnectionExited(UserInfo liveUser) {
    Log.i(TAG, "观众退出连线：${liveUser.userId}");
}
```

iOS

Swift

```
func onUserConnectionExited(userInfo: TUIUserInfo) {
    print("观众退出连线")
}
```

OC

```
- (void)onUserConnectionExited:(TUIUserInfo *)userInfo {
    NSLog(@"观众退出连线");
}
```

Flutter

Flutter中，您可以在onSeatListChanged回调给出的leftList中，查看断开连麦的观众。

```
// 文件位置：Flutter/lib/manager/observer/live_observer.dart

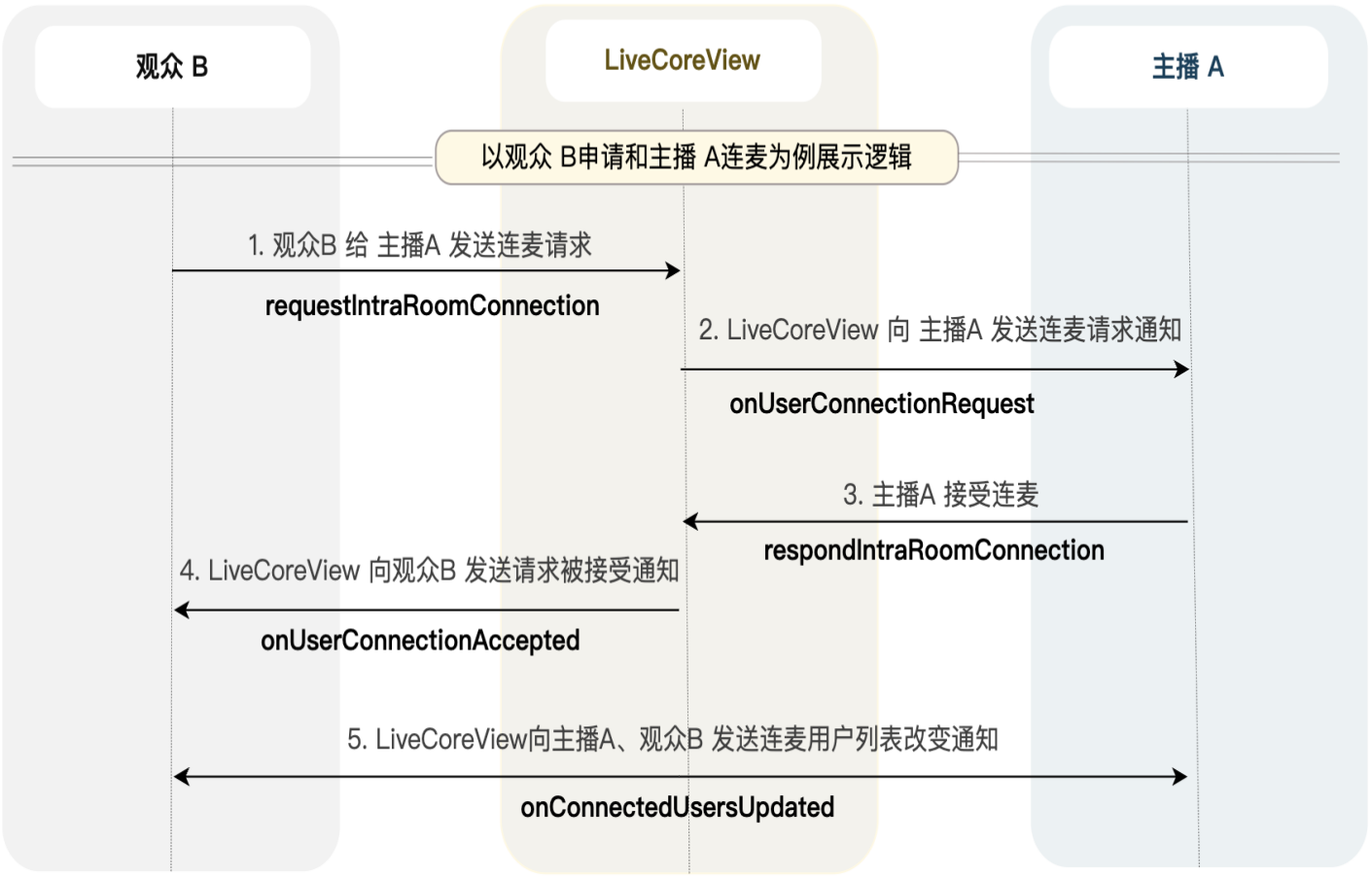
super.onSeatListChanged = (seatList, seatedList, leftList) {
      LiveKitLogger.info(
          "$tag($hashCode) onSeatListChanged:[seatList:$seatList
,seatedList:$seatedList,leftList:$leftList]");

liveController.target?.seatController.onSeatListChanged(seatList,
seatedList, leftList);
};
```

说明：
连线功能是基于 LiveCoreView 实现，若您需要扩展连线功能，可参见 LiveCoreView 文档。

观众连麦时序图

观众申请连麦 API 时序图



至臻画质（TUILiveKit）

最近更新时间：2025-05-07 16:49:52

功能介绍

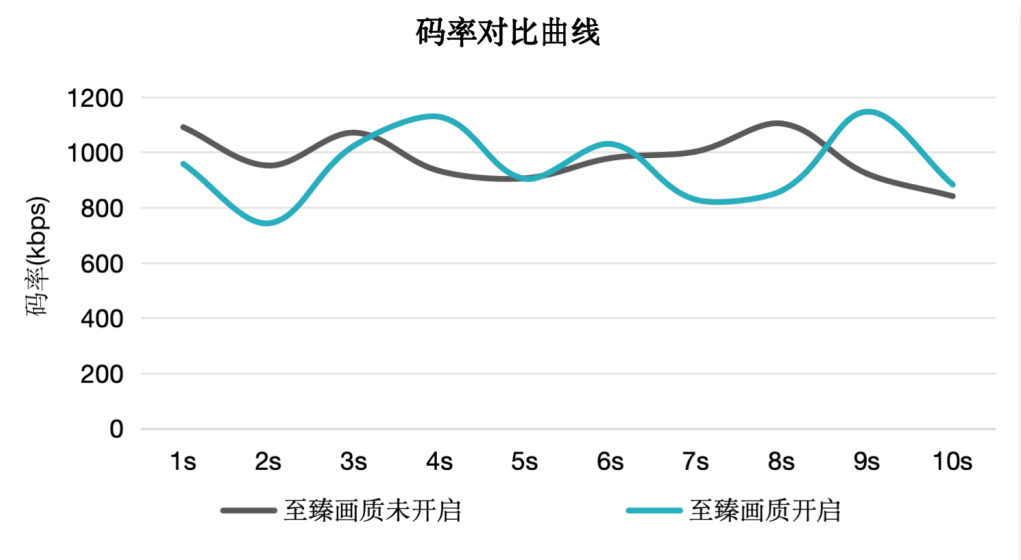
TUILiveKit 为了满足客户对直播画质清晰度、流畅度的高标准需求，推出了“**至臻画质**”功能。该功能实现了在相同画质下显著降低码率，在相同码率下画质得到显著提升，适合对画质要求高或带宽受限的环境。致力于为每个用户、每台设备提供清晰有质感、真实有美感的观看和互动体验。

效果展示

1. 码率相同，画质明显提升

在推流端视频参数相同的情况下，开启至臻画质后，拉流端的画面清晰度显著提升。

观看视频



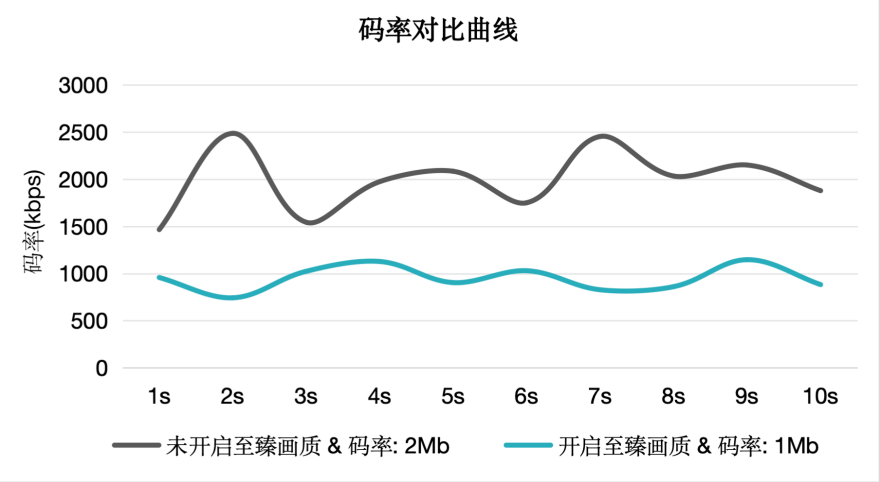
适用场景

至臻画质适用于对画质要求较高的场景。比如在电商直播、秀场直播等场景中，至臻画质不仅能让画面的色彩细节更加丰富，还能基于 AI模型自动为人脸或商品区域分配更多的码率，确保这些关键内容显示得更清晰。

2. 画质相同，编码码率更低

开启至臻画质后，使用较低的码率，依然能实现接近甚至超越高码率视频的画质表现。

观看视频



适用场景

在设备性能不足、带宽受限的弱网环境中，至臻画质能够使用更低的分辨率和码率，来减少视频播放时的卡顿率，和提高视频的清晰度。

开通方式

TUILiveKit 至臻画质功能开通无需 API 调用，目前仅支持**云端开启**，可通过点击 至臻画质内测申请 联系开通，相关申请会在一个工作日内处理。本次内测申请仅对 TUILiveKit 大规模直播版应用（SDKAppID）开放，申请前请确认您的应用满足上述条件，否则申请将不予通过。

互动礼物（TUILiveKit）

最近更新时间：2025-05-20 17:57:42

功能介绍

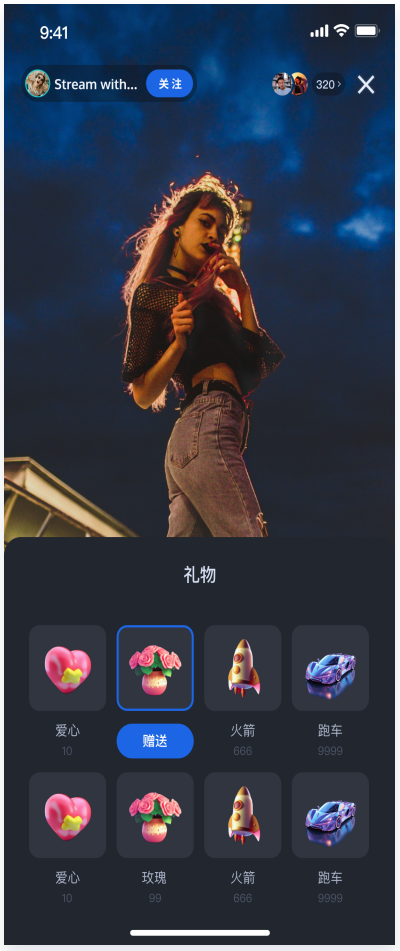
互动礼物组件是一款虚拟礼物互动平台，旨在为用户的社交体验增添更多乐趣。借助这一组件，用户能够向自己欣赏的直播主播送上虚拟礼物，以此展示他们的赞赏、喜爱以及支持。
互动礼物组件支持设置**点赞、礼物发送面板、发送礼物、播放普通礼物动画、播放 svga 礼物动画**等。

Flutter

互动礼物组件提供 2 个视图组件：

- `GiftSendWidget`：发送礼物消息视图，点击后可以拉起礼物面板。
- `GiftDisplayWidget`：接收礼物消息视图，会显示礼物消息到该视图上。

效果如下图所示：

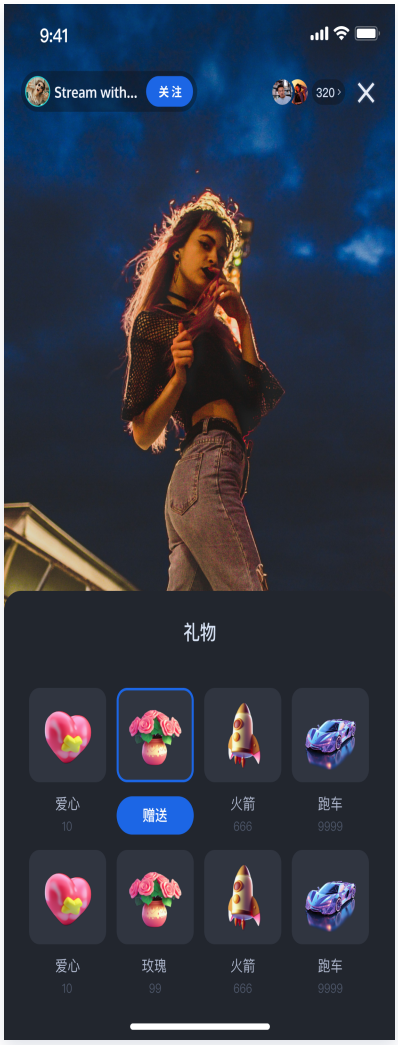
礼物展示面板	普通礼物播放效果	全屏礼物播放效果
		

Android

礼物组件主要提供2个 API：

- `GiftListPanelView`：**礼物面板，呈现礼物列表，发送礼物及充值。**
- `GiftPlayView`：**播放礼物的面板，自动监听礼物消息。**

效果如下图所示：

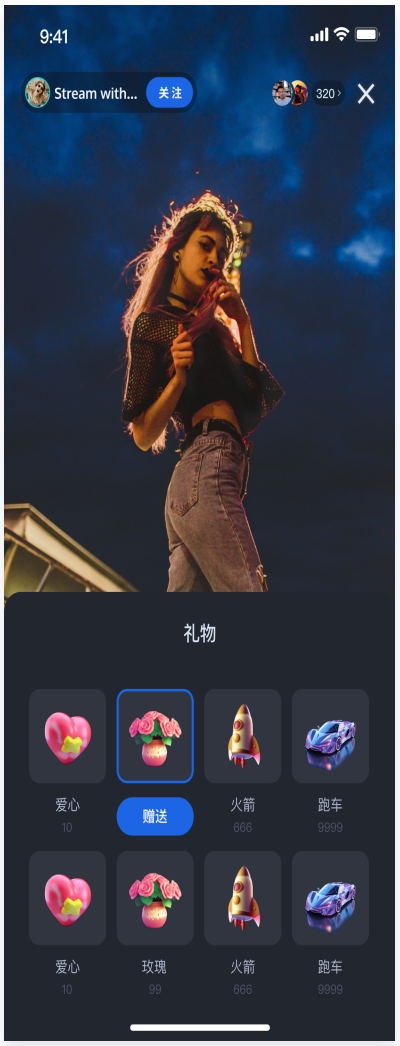
礼物展示面板	普通礼物播放效果	全屏礼物播放效果
		

iOS

礼物组件主要提供2个 API：

- `GiftListView`：礼物面板，呈现礼物列表，发送礼物。
- `GiftPlayView`：播放礼物的面板，自动监听礼物消息。

效果如下图所示：

礼物展示面板	普通礼物播放效果	全屏礼物播放效果
		

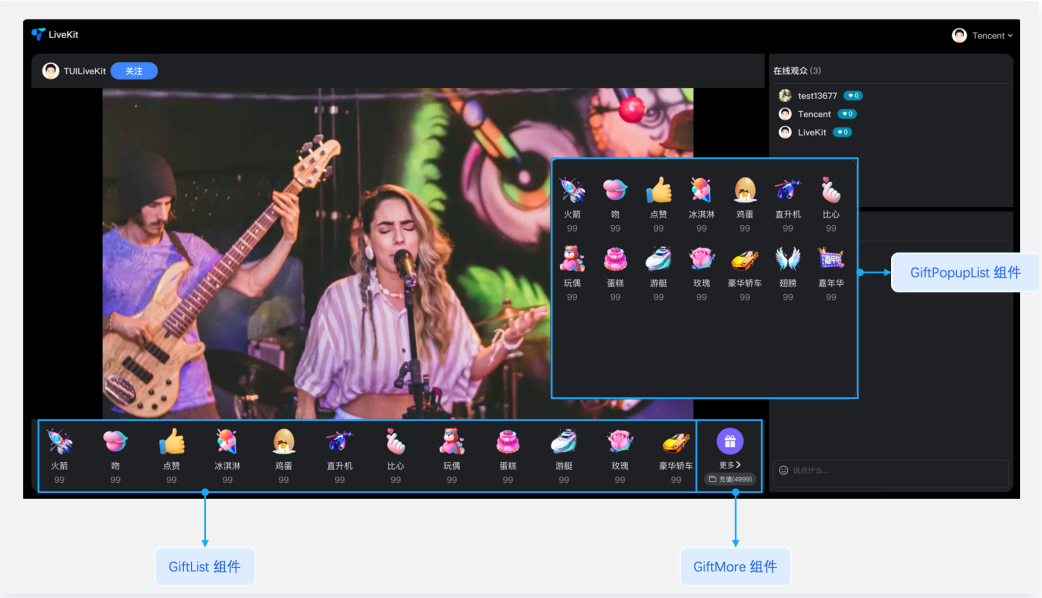
Web

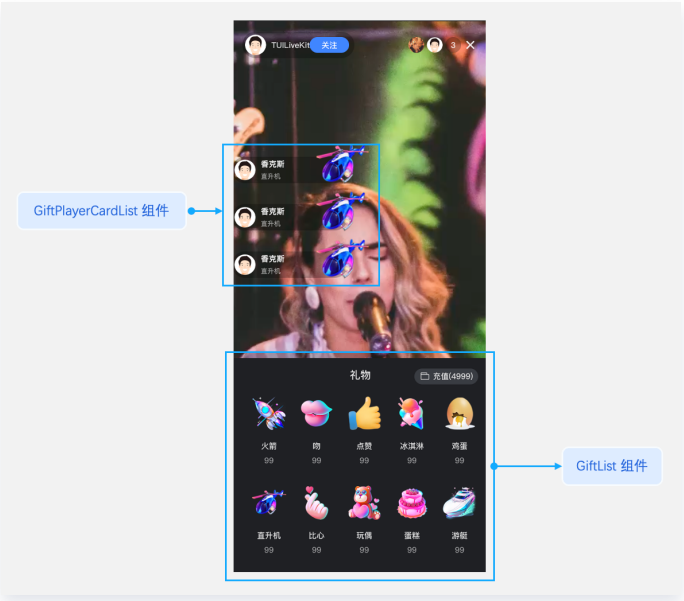
互动礼物主要提供4个组件：

- `GiftList`：礼物面板，呈现礼物列表，发送礼物，H5 上面板右上角还含有充值功能。

- `GiftMore`：更多礼物按钮，PC 上含有充值功能。
- `GiftPopupList`：PC 上当 `GiftList` 没能完全展示所有礼物时，此时可以点击更多礼物弹出 `GiftPopupList` 组件展示所有礼物。
- `GiftPlayerCardList`：播放礼物的面板，自动监听礼物消息。

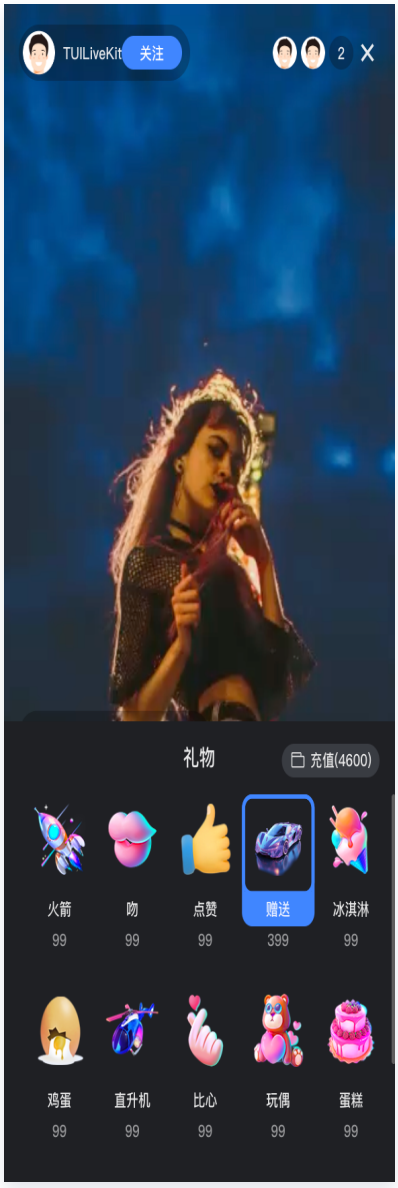
组件介绍如下图所示：

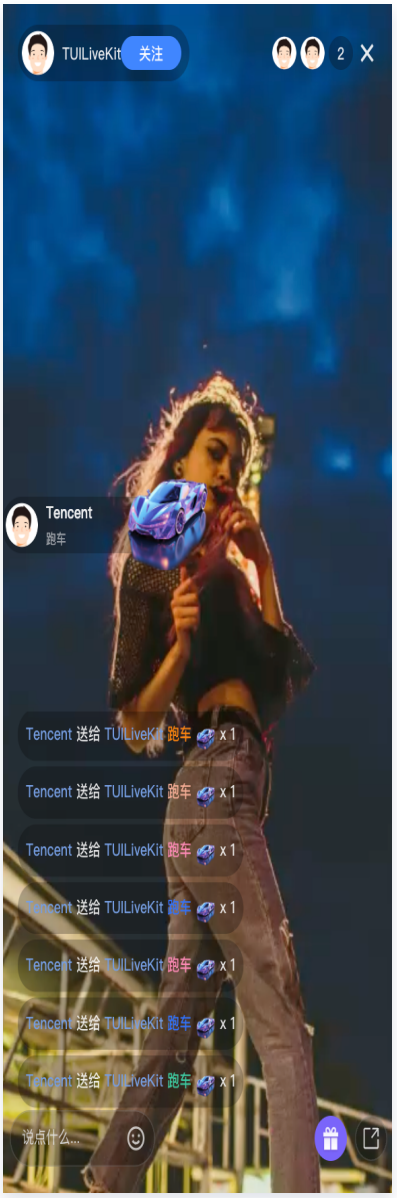




效果如下图所示：

礼物展示面板	普通礼物播放效果	全屏礼物播放效果







快速接入

> **说明：**
> 如果您已经集成了 LiveKit 组件，我们在 LiveKit 中默认已经接入了 gift 组件，您可以直接在 LiveKit 中体验互动礼物功能。

添加 Gift 组件到项目中

Flutter

1. 在项目工程的 `pubspec.yaml` 文件的 `dependencies` 节点中，添加 gift 的依赖。

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  intl: ^0.19.0
  # 添加 gift 依赖
  live_uikit_gift: ^1.0.0
```

2. 执行 `fluter pub get` 命令
3. 配置多语言支持，在您应用 `MaterialApp` 的 `localizationsDelegates` 和 `supportedLocales` 属性上添加 gift 组件的多语言支持。

```
MaterialApp(localizationsDelegates: const [
  ...GiftLocalizations.localizationsDelegates,
], supportedLocales: const [
  ...GiftLocalizations.supportedLocales,
],
// ...
);
```

Android

集成 LiveKit 组件即可正常使用，详情见 GitHub 。

iOS

集成 LiveKit 组件即可正常使用，详情见 GitHub 。

Web

集成 LiveKit 组件即可正常使用，详情见 NPM 目录：`src/TUILive/components/Gift/` 。

接入发送礼物组件

Flutter

在您需要接入发送礼物消息的地方构建 `GiftSendController` 和 `GiftSendWidget` 对象，并将构建的 `GiftSendWidget` 对象添加到您的 `Widget` 树上。示例代码如下：

```
GiftSendController _giftSendController = GiftSendController(
  roomId: "liveRoomId",              /// liveRoomId    替换为您的直播间ID
  owner: ownerInfo,                  /// ownerInfo     替换为您的直播间主播
信息
  self: selfInfo,                    /// selfInfo      替换为您的登录用户信
息
);

GiftSendWidget(controller: _giftSendController);
```

Android

实现 `GiftListPanelView` 的 `OnSendGiftListener` 中的 `onSendGift` 回调，获取礼物个数和礼物信息，在预处理完后可调用 `GiftListPanelView` 的 `sendGift` 函数用于礼物的实际发送。

```
public void onSendGift(GiftListPanelView giftListView, Gift gift, int
giftCount) {
   //...此处为预处理，如校验当前用户的余额等操作
   GiftUser receiver = new GiftUser();
   //...此处设置礼物接受者信息
   giftListView.sendGift(gift, giftCount, receiver);
}
```

iOS

创建 `GiftListView` 对象，并将其添加到您的视图上，同时还需要实现 `GiftListViewDelegate` 中的 `onSendGift` 回调方法。
当您点击 `GiftListView` 面板上的发送按钮时，您会收到该回调方式，您可在该方法中获取准备发送的礼物个数和礼物信息，在预处理完后可调用 `GiftListView` 的 `sendGift` 函数用于礼物的实际发送。

```
lazy var giftListView = GiftListView(roomId: roomId, delegate: self)
// ...此处将giftListView添加到您的父视图上并调整布局
```

```
func onSendGift(gift model: TUIGift, giftCount: Int) {
    //...此处为预处理，如校验当前用户的余额等操作
    let receiver = TUIGiftUser()
    //...此处设置礼物接受者信息
    giftListView.sendGift(model: giftModel, giftCount: giftCount,
receiver: receiver, completion: completion)
}
```

Web

在您需要发送礼物的组件内引入 `useGiftList` hook 并执行，从执行结果中解构出 `handleSendGift` 方法，当您点击 `GiftList` 面板上的发送按钮时，您只需将准备发送的礼物信息作为参数传递给 `handleSendGift` 方法即可发送成功，可参考 `GiftList` 组件的礼物发送逻辑。

```
import { ref } from "vue";
import { useGiftList, type Gift } from
"src/TUILive/components/Gift/useGiftList.ts";

const { handleSendGift } = useGiftList();
const selectedGift = ref<Gift>({} as Gift);

// 该方法绑定为 GiftList 面板上发送按钮的点击事件
const handleSendSelectedGift = async (gift: Gift) => {
  if (selectedGift.value !== gift) return;
  try {
    // 调用 handleSendGift 发送礼物
    await handleSendGift(gift);
    selectedGift.value = {} as Gift;
  } catch (error) {
    selectedGift.value = {} as Gift;
    throw error;
  }
}
```

接入显示礼物组件

Flutter

在您需要接入显示礼物消息的地方构建 `GiftDisplayController` 和 `GiftDisplayWidget` 对象，并将构建的 `GiftDisplayWidget` 对象添加到您的 `Widget` 树上。示例代码如下：

```
GiftDisplayController _giftDisplayController = GiftDisplayController(
  rroomId: "liveRoomId",              /// liveRoomId    替换为您的直播间ID
  owner: ownerInfo,                   /// ownerInfo     替换为您的直播间主
播信息
  self: selfInfo,                     /// selfInfo      替换为您的登录用户
信息
);

GiftDisplayWidget(controller: _giftDisplayController!);
```

Android

礼物展示组件 `GiftPlayView` 自身会接收并播放礼物消息。

```
GiftPlayView giftPlayView = new GiftPlayView(mContext, roomId);
```

iOS

礼物展示组件 `GiftPlayView` 自身会接收并播放礼物消息。
您只需要创建 `GiftPlayView` 对象并将其添加到您的视图上即可。

```
let giftPlayView = GiftPlayView(groupId: roomId)
// ...此处将giftPlayView添加到您的父视图上并调整布局
```

Web

礼物展示组件 `GiftPlayerCardList` 自身会接收并播放礼物消息。
您只需将其引入并添加到需要展示的组件内即可。

```
<template>
    // 将 GiftPlayerCardList 组件放置后可根据您具体的业务场景调整布局
```

```
    <GiftPlayerCardList/>
</template>

<script setup lang="ts">
import GiftPlayerCardList from
'src/TUILive/components/Gift/GiftPlayerCardList.vue';
</script>
```

监听礼物收发消息

Flutter

若需要获取收发礼物的回调信息，您可以调用 `GiftDisplayController` 的 `setGiftCallback` 方法，示例代码如下所示：

```
_giftDisplayController?.setGiftCallback(
  onReceiveGiftCallback: _onReceiveGiftCallback,     ///
_onReceiveGiftCallback 可以替换为您的自定义处理方法
  onSendGiftCallback: _onSendGiftCallback,           ///
_onSendGiftCallback    可以替换为您的自定义处理方法
);
```

Android

若需要获取接收礼物的回调信息，可实现 `GiftPlayView` 的 `TUIGiftPlayViewListener` 中的 `onReceiveGift` 函数。

```
public interface TUIGiftPlayViewListener {
    void onReceiveGift(Gift gift, int giftCount, GiftUser
sender, GiftUser receiver);
    //...
}
```

iOS

若需要获取接收礼物的回调信息，可实现 `GiftPlayView` 的 代理 `GiftPlayViewDelegate` 中的 `giftPlayView:onReceiveGift:gift:giftCount:sender:receiver` 函数。

```
func giftPlayView(_ giftPlayView: GiftPlayView, onReceiveGift gift:
TUIGift, giftCount: Int, sender: TUIGiftUser, receiver: TUIGiftUser)
    // 自定义处理
}
```

Web

若需要查看接收到的礼物信息，可以进入目录 `src/TUILive/components/Chat/MessageList/useMessageListHook.ts` ，在 `onReceiveMessage` 函数中查看。

```
// 文件位置:
src/TUILive/components/Chat/MessageList/useMessageListHook.ts

const onReceiveMessage = (options: { data: any }) => {
  if (!options || !options.data) {
    return;
  }
  const currentConversationId = `GROUP${roomId.value}`;
  options.data.forEach((message: any) => {
    // ..
    if (isGiftMsg(message.payload?.data)) {
      // 进入此判断则为礼物消息，可根据您具体的业务做自定义调整
    }
    // ..
  });
};
```

自定义礼物面板

Flutter

如果您需要修改礼物面板的礼物列表，您可以调用 `GiftSendController` 的 `setGiftList` 方法，示例如下：

```
        List<GiftModel> giftList = [      /// 自定义您的礼物列表数据源
  GiftModel(giftId: "1", giftName: "鸡蛋", imageUrl: "giftImageUrl",
price: 10),
  GiftModel(giftId: "2", giftName: "星星", imageUrl: "giftImageUrl",
price: 10),
];

_giftSendController.setGiftList(giftList);
```

Android

如果您需要修改礼物面板的礼物列表：

```
mGiftCloudServer.queryGiftInfoList((error, result) -> post(() -> {
    if (error == Error.NO_ERROR) {
        mGiftListView.setGiftList(result);
    } else {
        ToastUtil.toastLongMessage("query gift list error, code = " +
error);
    }
}));
```

说明：

- 客户自行实现 `queryGiftInfoList` 方法的逻辑，得到一个自定义的礼物列表 `List<` Gift `>` ，通过 `GiftListView.setGiftList` 设置礼物列表即可。
- 礼物的 `animationUrl` 要求是一个SVGA动画。

iOS

如果您需要修改礼物面板的礼物列表：

```
// 文件位置：
iOS/TUILiveKit/Sources/Component/Gift/Store/TUIGiftStore.swift

class TUIGiftStore {
    static let shared = TUIGiftStore()
```

```
    private init() {
        giftCloudServer.queryGiftInfoList { [weak self] error,
giftList in
            guard let self = self else { return }
            if error == .noError {
                self.giftList = giftList
            }
        }
    }
    var giftList: [TUIGift] = []
    let giftCloudServer: IGiftCloudServer = GiftCloudServer()
}


//  文件位置:
iOS/TUILiveKit/Sources/LiveStream/Main/Manager/LSRouterControlCenter.s
wift
private func getRouteDefaultView(route: LSRoute) -> UIView?
 {
    switch route {
    case .giftView:
      let giftPanel = GiftListPanel(roomId: manager.roomState.roomId,
dataSource: manager)
      giftPanel.setGiftList(TUIGiftStore.shared.giftList)
      view = giftPanel
    }
}
```

说明:

- 客户自行实现 `queryGiftInfoList` 方法的逻辑，得到一个自定义的礼物列表 `List<TUIGift>` ，通过 `GiftListView.setGiftList` 设置礼物列表即可。
- 礼物的 `animationUrl` 要求是一个 SVGA 动画。

Web

如果您需要修改礼物面板的礼物列表:

```
// 文件位置: src/TUILive/components/Gift/useGiftList.ts
```

```
// 礼物备份数据
const giftListDataBackup = {
    "giftList": [
        {
          "giftId": "1",
          "imageUrl": "礼物图片url地址",
          "animationUrl": "礼物特效地址", // 要求是一个 svga 动画
          "price": 99,
          "giftName": "比心",
          "giftName_en": "Heart gesture",
          "type": 0
        },
        // ...
    ]
}

// 在 getGiftList 方法中替换礼物数据的 url
const getGiftList = async (): Promise<Gift[]> => {
  const giftDataUrl = '您的礼物数据 url 地址';
  const { giftList } = await safelyFetchGiftData(giftDataUrl);
  // 浏览器请求资源时容易出现跨域问题导致拿不到接口数据，此时可使用礼物备份数据
  return giftList || giftListDataBackup;
}
```

常见问题

礼物组件初始化时机

由于礼物组件的 `controller` 需要接入一些直播间信息参数 和 自己的用户信息，所以需要在**观众进入直播间**或者**主播创建直播间**后，再加载礼物组件。

礼物特效（TUILiveKit）

最近更新时间：2025-04-03 16:12:02

TUILiveKit 提供了两种礼物特效播放器：基础特效播放器和高级特效播放器。默认情况下，集成了基础特效播放器，若您对播放器有更高的性能要求，或期望支持更多的动画文件格式，我们还提供了高级特效播放器供您使用。

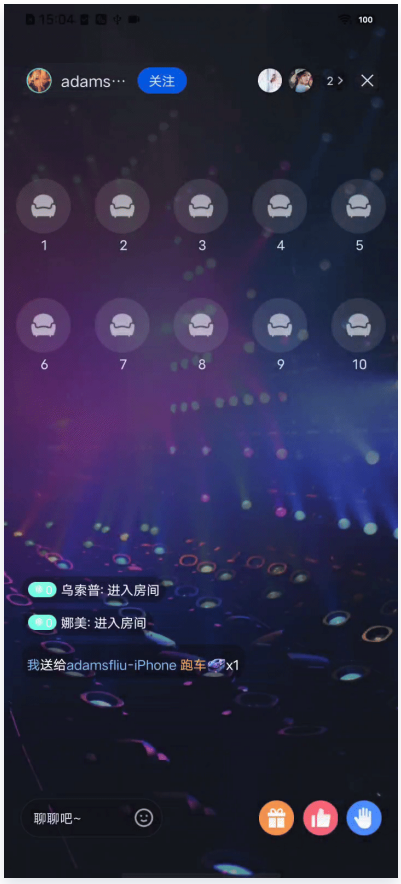
基础特效播放器

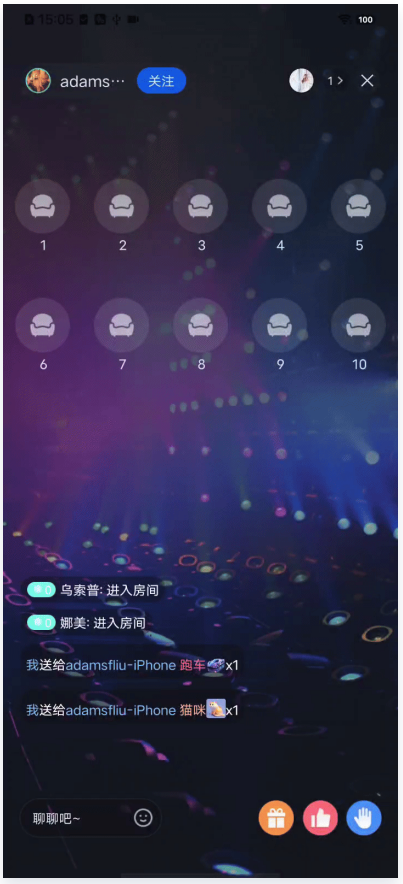
基础特效播放器基于 `SVGA` 实现，特效动画仅支持 `SVGA` 格式文件。
使用基础特效播放器时，有如下3个特效动画：

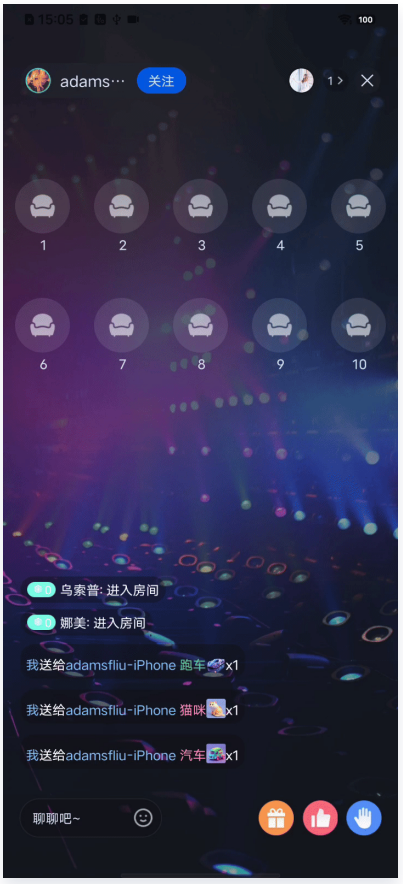


效果展示

跑车	猫咪	汽车







高级特效播放器

`TUILiveKit` 高级特效播放器采用腾讯特效播放器，支持多种格式的特效动画（如：vap、Lottie、mp4、svga 等）。

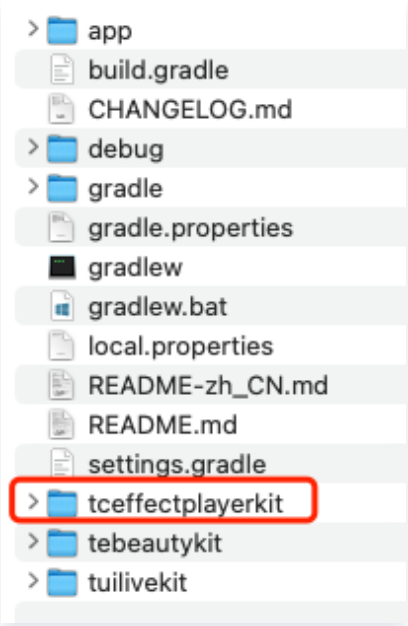
> **说明：**
> 高级特效播放器需单独付费，详情请查看腾讯特效播放器。

开始集成

步骤一：集成礼物特效组件

Android

1. 下载并解压 TUILiveKit，把 `Android/tceffectplayerkit` 文件夹拷贝到自己的工程中，和 app 同级目录。



2. 编辑您自己工程的 `settings.gradle` 文件，添加下面的代码：

```
include ':tceffectplayerkit'
```

3. 如果您只接入了高级特效播放器，没有接入高级美颜，那么请在 tceffectplayerkit/build.gradle 文件中注释一行代码：

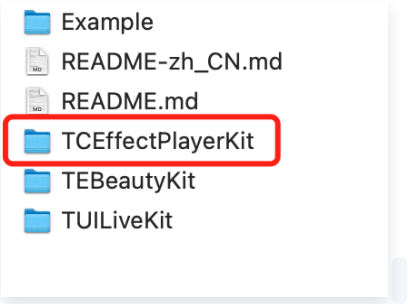
```
dependencies {
    api("com.tencent.mediacloud:TCEffectPlayer:3.0.0.250") {
       // exclude group: "com.tencent.mediacloud", module:
"TCXMagicAuth"
    }
    ...
}
```

4. 如果您的 App 开启了混淆，请在您的混淆配置里加上如下配置：

```
-keep class com.tcmediax.** { *; }
-keep class com.tencent.** { *; }
-keep class com.tencent.xmagic.** { *; }
-keep class androidx.exifinterface.** {*;}
```

iOS

1. 下载并解压 TUILiveKit，把 `iOS/TCEffectPlayerKit` 文件夹拷贝到自己的工程中，和 `Podfile` 同级目录。



2. 编辑 `podfile` 文件，添加下面的代码：
   - 若您未集成 高级美颜，参考如下：

```
  pod 'TCEffectPlayerKit', :path =>
'../TCEffectPlayerKit/TCEffectPlayerKit.podspec'
  pod 'TCMediaX', :podspec => 'https://mediacloud-
76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCMediaX.podspec'
  pod 'TCEffectPlayer', :podspec => 'https://mediacloud-
76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCEffectPlayer.pods
pec'
  pod 'YTCommonXMagic', :podspec => 'https://mediacloud-
76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/YTCommonXMagic_1.1/YTCommon
XMagic.podspec'
```

   - 若您集成了 高级美颜，参考如下：

```
  pod 'TCEffectPlayerKit/Beauty', :path =>
'../TCEffectPlayerKit/TCEffectPlayerKit.podspec'
  pod 'TCMediaX', :podspec => 'https://mediacloud-
76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCMediaX.podspec'
  pod 'TCEffectPlayer', :podspec => 'https://mediacloud-
76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCEffectPlayer.pods
pec'
```

3. 终端执行 `pod install` 命令。

步骤二：鉴权

1. 申请授权，得到 `LicenseUrl` 和 `LicenseKey` 。详情请发送邮件咨询： TRTC_helper@tencent.com。
2. 在您的业务初始化的地方，添加如下鉴权代码，并替换您申请的 `LicenseUrl` 和 `LicenseKey` ：
   如：

Android

```
  TCMediaXBase.getInstance().setLicense(context,
        "LicenseUrl", // 请替换 LicenseUrl
        "LicenseKey", // 请替换 LicenseKey
        new ILicenseCallback() {
            @Override
            public void onResult(int error, String message) {
                Log.i("TCMediaXBase", "setLicense result: " + error +
" " + message);
            }
        });
```

iOS

iOS 可在 `AppDelegate` 的 `didFinishLaunchingWithOptions` 方法中设置相关内容。

```
//
//  AppDelegate.swift
//

import TCMediaX

func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    TCMediaXBase.getInstance().setDelegate(self)
    TCMediaXBase.getInstance().setLicenceURL("LicenseURL", key:
"LicenseKEY")
    return true
}
```

```
func onLicenseCheckCallback(_ errcode: Int32, withParam param:
[AnyHashable : Any]) {
    debugPrint("[TCMediaXBase] setLicense result: errcode:\(errcode),
param:\(param)")
}
```

美颜特效（TUILiveKit）

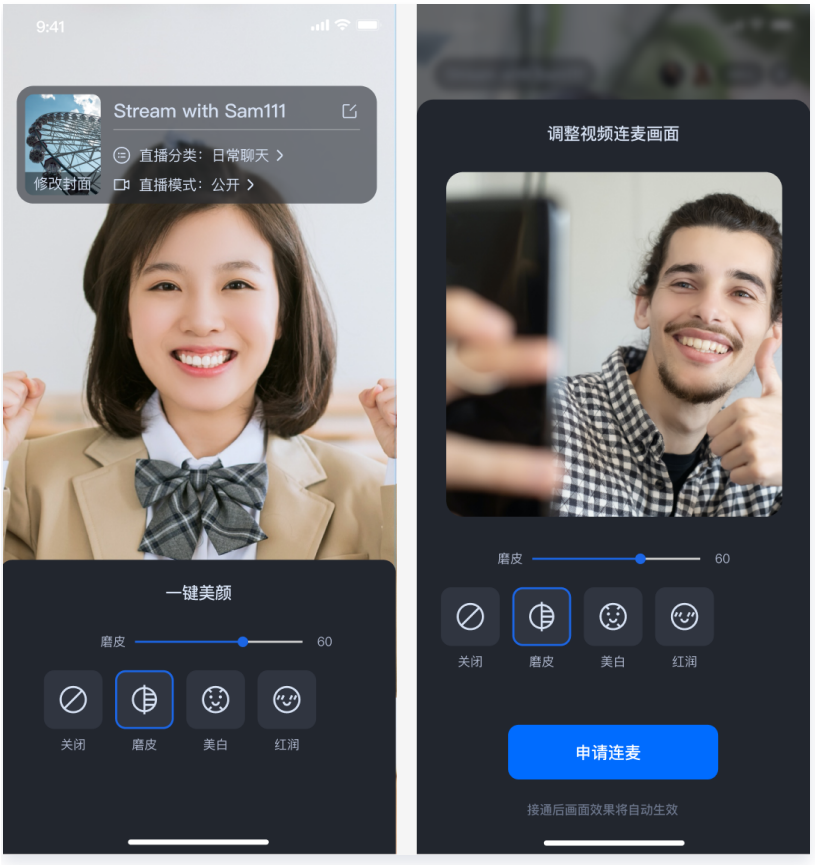
最近更新时间：2025-04-03 16:12:02

TUILiveKit 提供了两种美颜特效：基础美颜和高级美颜。如果您对基础美颜的效果不满意，可以选择集成高级美颜来满足更高级的美颜需求。

基础美颜

TUILiveKit 默认已经集成了基础美颜功能。基础美颜包括美白、磨皮和红润效果，并且可以通过调节美颜强度来满足不同的需求。这些功能已经内置在TUILiveKit 中，**无需额外的配置或集成**。

面板展示

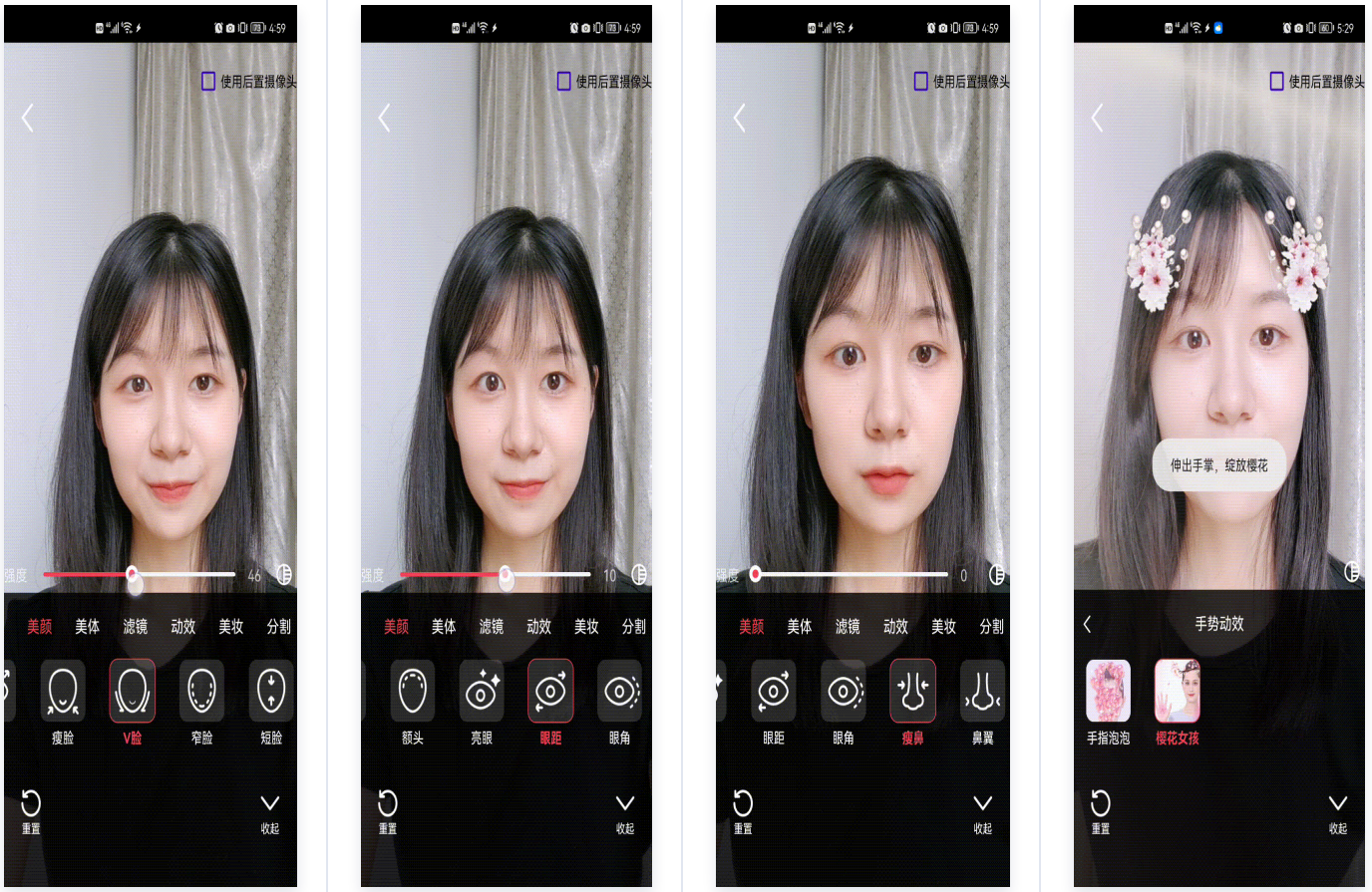


高级美颜

TUILiveKit 高级美颜采用 腾讯特效美颜 。

效果展示

v脸	眼距	瘦鼻	3D贴纸



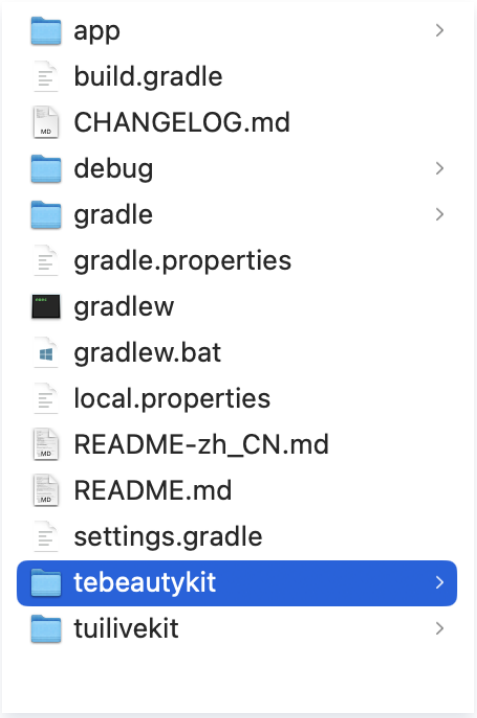
说明：
高级美颜需要单独付费，详情请查看 腾讯特效 SDK 。

开始集成

步骤一：集成 `tebeautykit`

Android

1. 下载并解压 TUILiveKit，把 `Android/tebeautykit` 文件夹拷贝到自己的工程中，和 app 同级目录。

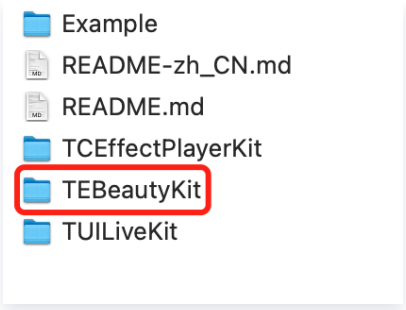


2. 编辑您自己工程的 `settings.gradle` 文件，添加下面的代码：

```
include ':tebeautykit'
```

iOS

1. 下载并解压 TUILiveKit，把 `iOS/TEBeautyKit` 文件夹拷贝到自己的工程中，和 `Podfile` 同级目录。



2. 编辑 `podfile` 文件，添加下面的代码：

```
  pod 'TEBeautyKit',:podspec => './TEBeautyKit/TEBeautyKit.podspec'
```

3. 终端执行 `pod install` 命令。

步骤二： 鉴权 & 设置美颜资源

1. 申请授权，得到 `LicenseUrl` 和 `LicenseKey`，请参见 License 指引。
2. 在您的业务初始化的地方（默认可在 登录 的相同位置），添加如下鉴权代码，并替换您申请的 `美颜套餐编号`、`LicenseUrl` 和 `LicenseKey`：

Android

```
TEBeautySettings.getInstance().initBeautySettings(context,
                                    S1_07,              // 将S1_07替换为
您购买的美颜套餐编号
                                    "LicenseUrl",       // 请替换
LicenseUrl
                                    "LicenseKey");      // 请替换
LicenseKey
```

iOS

iOS 可在 `AppDelegate` 的 `didFinishLaunchingWithOptions` 方法中设置相关内容。

```
//
//  AppDelegate.swift
//

import TEBeautyKit

func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        TEBeautyKit.setTELicense("LicenseURL", key: "LicenseKEY") {
code, message in
            TEUIConfig.shareInstance().setPanelLevel(.S1_07) // 将
S1_07替换为您购买的美颜套餐编号
        }
        return true
}
```

说明：

- 如果您不清楚美颜编号，可点此 美颜套餐编号总览 查看。
- **完成上述步骤即可集成高级美颜。**

版权所有：腾讯云计算（北京）有限责任公司

互动弹幕（TUILiveKit）

最近更新时间：2025-05-20 17:57:42

功能介绍

互动弹幕功能是一项重要的实时交流工具，支持多种互动方式。用户可以在弹幕中输入表情和文本消息，增强消息的趣味性，使得互动体验更加愉快和生动。通过这一功能，观众能够在直播过程中与主播和其他观众进行更为丰富的交流，提升整体的参与感和乐趣。TUILiveKit 中已经通过即时通信 IM 实现了互动弹幕的功能。

iOS

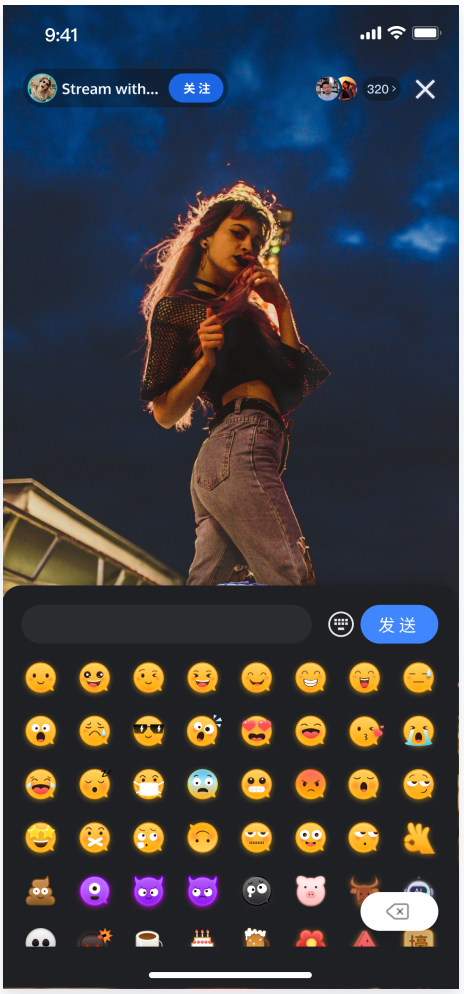
弹幕组件提供 2 个视图组件：

- `BarrageInputView`：发送弹幕消息视图，点击后可以拉起输入界面。
- `BarrageStreamView`：接收弹幕消息视图，会显示弹幕消息到该视图上。

效果如下图所示：

展示	输入





Flutter

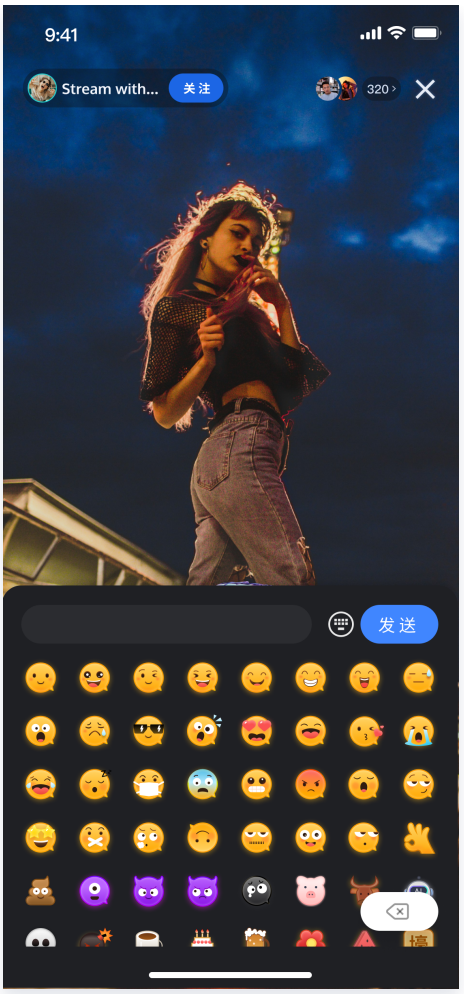
弹幕组件主要包提供2个 Widget 组件，发送弹幕消息 Widget（ `BarrageSendWidget` ）和显示弹幕消息的 Widget（ `BarrageDisplayWidget` ）：

- `BarrageSendWidget`：**发送弹幕消息 Widget，点击后可以拉起输入界面。**
- `BarrageDisplayWidget`：**接收到弹幕消息后，会显示弹幕消息到该 Widget 上。**

效果如下图所示：

展示	输入





Android

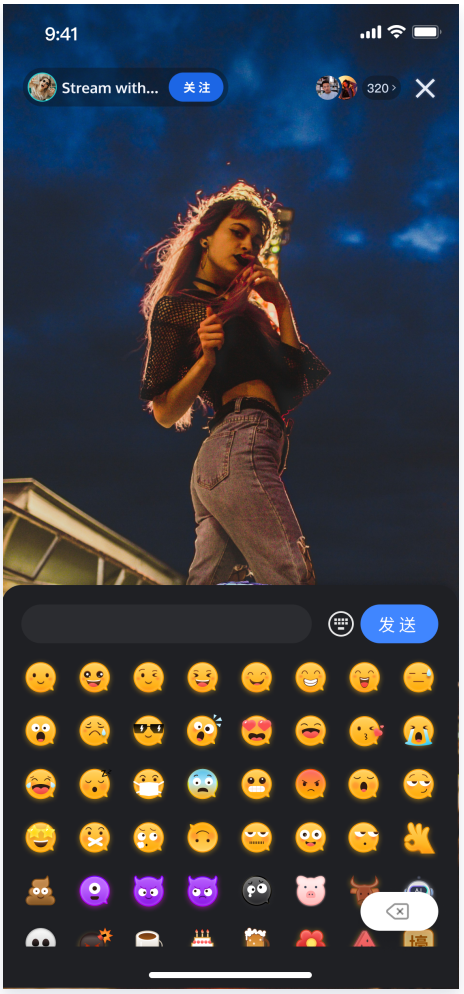
弹幕组件提供 2 个视图组件：

- `BarrageInputView`：**发送弹幕消息视图，点击后可以拉起输入界面。**
- `BarrageStreamView`：**接收弹幕消息视图，会显示弹幕消息到该视图上。**

效果如下图所示：

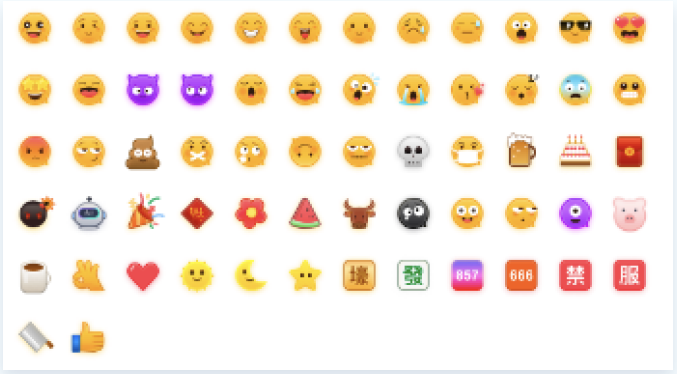
展示	输入





说明：

1. 支持**系统键盘**和**表情键盘**切换。
2. 为尊重表情设计版权，TUILiveKit 工程中不包含大表情元素切图，正式上线商用前请您替换为自己设计或拥有版权的其他表情包。下图所示默认的**小黄脸表情包版权归腾讯云所有**，您可以通过升级至 IM企业版套餐 免费使用该表情包。



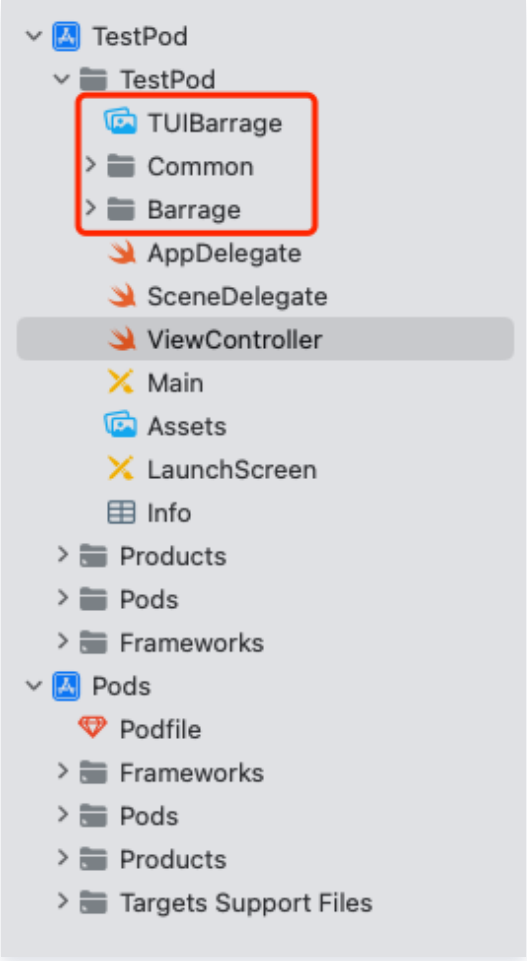
快速接入

> **说明：**
> 如果您已经集成了 LiveKit 组件，我们在 LiveKit 中默认已经接入了 barrage 组件，您可以直接在 LiveKit 中体验弹幕功能。

添加 barrage 组件到项目中

iOS

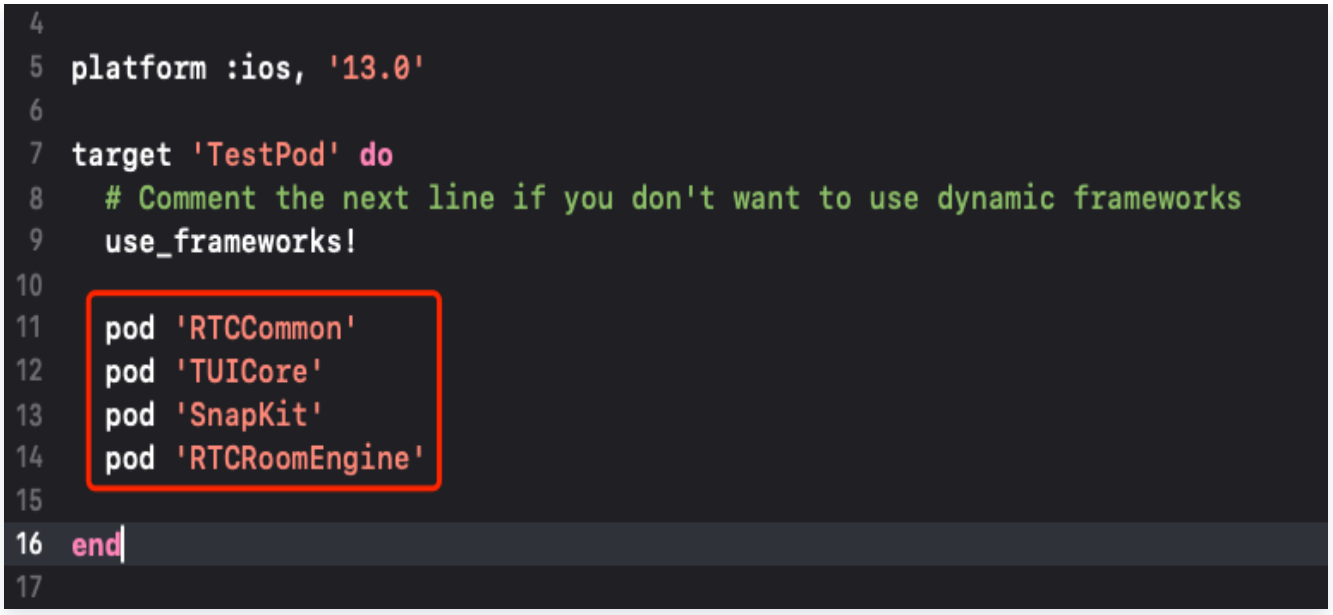
1. 从 GitHub 下载 Barrage 组件到本地，并将 Barrage (TUILiveKit/Sources/Component/Barrage) 目录拷贝到您的工程中；
2. 拷贝依赖组件 Common (TUILiveKit/Sources/Common) 到您的工程中；
3. 拷贝资源文件 TUIBarrage.xcassets (TUILiveKit/Resources/TUIBarrage.xcassets) 到您的工程中。
4. 拷贝语言文件 Localized (TUILiveKit/Resources/Localized) 到您的工程中。
   目录如下图所示：



5. 在您的Podfile文件中，添加如下依赖：
   - pod 'RTCCommon'

- pod 'TUICore'
- pod 'SnapKit'
- pod 'RTCRoomEngine'

如图：



```
platform :ios, '13.0'

target 'TestPod' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  pod 'RTCCommon'
  pod 'TUICore'
  pod 'SnapKit'
  pod 'RTCRoomEngine'

end
```

6. 执行 pod install

Flutter

1. 在项目工程的 pubspec.yaml 文件的 dependencies 节点中，添加 **barrage** 的依赖。

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  intl: ^0.19.0
  # 添加 barrage 依赖
  live_uikit_barrage: ^1.0.0
```

2. 执行 `flutter pub get` 命令
3. 配置多语言支持，在您应用 `MaterialApp` 的 `localizationsDelegates` 和 `supportedLocales` 属性上添加 **gift** 组件的多语言支持。

版权所有：腾讯云计算（北京）有限责任公司

```
MaterialApp(localizationsDelegates: const [
  ...BarrageLocalizations.localizationsDelegates,
], supportedLocales: const [
  ...BarrageLocalizations.supportedLocales,
],
// ...
);
```

Android

1. 从 GitHub 下载 TUILiveKit 到本地。
2. 将 Android/tuilivekit/component/barrage 目录拷贝到您的工程中。
3. 将 Android/tuilivekit/component/common 目录拷贝到您的工程中。
4. 并在您工程的settings.gradle文件中增加配置：

```
include ':common'
include ':barrage'
```

5. 在您需要引入的工程的build.gradle文件中，增加依赖配置：

```
api project(':common')
api project(':barrage')
```

接入发送弹幕消息组件

iOS

在需要发送弹幕的位置，创建 `BarrageInputView` ，点击后可以拉起输入界面：

```
let barrageInputView = BarrageInputView(roomId: roomId)
addSubView(barrageInputView)
```

Flutter

在您需要接入发送弹幕消息的地方构建 BarrageSendController 和 BarrageSendWidget 对象，并将构建的 BarrageSendWidget 对象添加到您的 Widget 树上。示例代码如下：

```
BarrageSendController _sendController = BarrageSendController(
                roomId: "liveRoomId",              /// liveRoomId   替换为您的直播间ID
                ownerId:  "liveOwnerId",           /// liveOwnerId  替换为您的直播间主播ID
                selfUserId: "selfUserId",          /// selfUserId   替换为您的当前登录的用户ID
                selfName: "selfUserName";          /// selfUserName 替换为您的当前登录的用户昵称
BarrageSendWidget(controller: _sendController);
```

Android

在需要发送弹幕的位置，创建 `BarrageInputView` ，点击后可以拉起输入界面：

```
BarrageInputView barrageInputView = new BarrageInputView(mContext);
barrageInputView.init(roomId);
mBarrageInputContainer.addView(barrageInputView);
```

接入显示弹幕消息组件

iOS

在需要展示弹幕的位置，使用 `BarrageStreamView` 来展示弹幕消息：

```
let barrageDisplayView = BarrageStreamView(roomId: roomId)
addSubView(barrageDisplayView)
```

在获取到房主信息后，设置 ownerId 用来区分房主与观众的显示效果。

```
barrageDisplayView.setOwnerId(ownerId)
```

Flutter

在您需要接入显示弹幕消息的地方构建 BarrageDisplayController 和 BarrageDisplayWidget 对象，并将构建的 BarrageDisplayWidget 对象添加到您的 Widget 树上。示例代码如下：

```
BarrageDisplayController _displayController =
BarrageDisplayController(
                roomId: "liveRoomId",              /// liveRoomId   替换为您的直播间ID
                ownerId:  "liveOwnerId",           /// liveOwnerId  替换为您的直播间主播ID
                selfUserId: "selfUserId",          /// selfUserId   替换为您的当前登录的用户ID
                selfName: "selfUserName";          /// selfUserName 替换为您的当前登录的用户昵称
BarrageDisplayWidget(controller: _displayController);
```

Android

在需要展示弹幕的位置，使用 `BarrageStreamView` 来展示弹幕消息：

```
BarrageStreamView barrageStreamView = new BarrageStreamView(mContext);
// ownerId 表示房主的 ID，用来区分房主与观众的显示效果
barrageStreamView.init(roomId, ownerId);
mLayoutBarrageContainer.addView(barrageStreamView);
```

插入本地弹幕消息

iOS

弹幕展示组件 BarrageStreamView 对外提供 `insertBarrages` 接口方法，用于（批量）插入自定义消息（如 礼物消息、直播间公告），通常自定义消息配合自定义样式，可以实现不一样的展示效果。
示例代码：

```
// 示例1：在弹幕区插入一条礼物消息
let barrage = TUIBarrage()
barrage.content = "gift"
```

```
barrage.user.userId = sender.userId
barrage.user.userName = sender.userName
barrage.user.avatarUrl = sender.avatarUrl
barrage.user.level = sender.level

barrage.extInfo["TYPE"] = AnyCodable("GIFTMESSAGE")
barrage.extInfo["gift_name"] = AnyCodable(gift.giftName)
barrage.extInfo["gift_count"] = AnyCodable(giftCount)
barrage.extInfo["gift_icon_url"] = AnyCodable(gift.imageUrl)
barrage.extInfo["gift_receiver_username"] =
AnyCodable(receiver.userName)

barrageDisplayView.insertBarrages([barrage])

// 示例2:发送弹幕后回显消息
// 实现 BarrageInputView 的代理 BarrageInputViewDelegate
extension MyViewController: BarrageInputViewDelegate {
    func barrageInputViewOnSendBarrage(_ barrage: TUIBarrage) {
        barrageDisplayView.insertBarrages([barrage])
    }
}
```

说明：
`TUIBarrage` 的 `extInfo` 是一个 `Map`，用于存放自定义数据。

Flutter

当您需要插入本地的弹幕消息时，可以调用 BarrageDisplayWidget 的 insertMessage 方法，插入本地消息。例如，LiveKit 中监听到 观众进房后，可以插入一条 观众进房的 弹幕消息，示例代码如下所示：

```
BarrageUser barrageUser = BarrageUser();
barrageUser.userId = "enterRoomUserId";             /// 弹幕上展示的用户
Id信息
barrageUser.userName = "enterRoomUserName";         /// 弹幕上展示的用户昵
称信息
barrageUser.avatarUrl = "enterRoomUserAvatar";      /// 弹幕上展示的用户头
像信息
barrageUser.level = "66";                           /// 弹幕上展示的用户等
级信息
```

```
Barrage barrage = Barrage();
barrage.user = barrageUser;
barrage.content = "进入房间";                                    /// 弹幕上展示的文本内
容
_displayController.insertMessage(barrage);
```

Android

弹幕展示组件 BarrageStreamView 对外提供 `insertBarrages` 接口方法，用于（批量）插入自定义消息（如 礼物消息、直播间公告），通常自定义消息配合自定义样式，可以实现不一样的展示效果。
示例代码：

```
// 在弹幕区插入一条礼物消息
Barrage barrage = new Barrage();
barrage.content = "gift";
barrage.user.userId = sender.userId;
barrage.user.userName = sender.userName;
barrage.user.avatarUrl = sender.avatarUrl;
barrage.user.level = sender.level;
barrage.extInfo.put(Constants.GIFT_VIEW_TYPE, GIFT_VIEW_TYPE_1);
barrage.extInfo.put(GIFT_NAME, barrage.giftName);
barrage.extInfo.put(GIFT_COUNT, giftCount);
barrage.extInfo.put(GIFT_ICON_URL, barrage.imageUrl);
barrage.extInfo.put(GIFT_RECEIVER_USERNAME, receiver.userName);

mBarrageStreamView.insertBarrages(barrage);
```

说明：
`Barrage` 的 `extInfo` 是一个 `Map`，用于存放自定义数据。

自定义弹幕消息

iOS

弹幕消息样式默认有两种：普通弹幕消息样式 和 自定义消息样式。

如果您需要更多消息样式（如**礼物发送回显**），可实现 BarrageStreamView 的代理方法 BarrageStreamViewDelegate：

```
class MyViewController: BarrageStreamViewDelegate {
    func barrageDisplayView(_ barrageDisplayView: BarrageStreamView,
createCustomCell barrage: TUIBarrage) -> UIView? {
        // 是否需要使用自定义UI，如不需要返回 nil 即可
        guard let type = barrage.extInfo["TYPE"], type.value as?
String == "GIFTMESSAGE" else {
            return nil
        }
        // 返回自定义消息样式UI（礼物回显）
        return CustomBarrageCell(barrage: barrage)
    }
}
```

Flutter

当您需要针对特定的弹幕消息，展示自定义的弹幕 item 时，您可以通过 BarrageDisplayWidget 的 setCustomBarrageBuilder 方法实现。例如，自定义一个显示红色文本的弹幕示例代码如下所示：

```
/// 1、定义一个自定义弹幕 item 构建器
class GiftBarrageItemBuilder extends CustomBarrageBuilder {
  @override
  Widget buildWidget(BuildContext context, Barrage barrage) { /// 当
shouldCustomizeBarrageItem 返回true时，自定义Widget
    return const Text(
      barrage.content,
      style: TextStyle(fontSize: 18, fontWeight: FontWeight.w700,
color: Colors.red),
    );
  }

  @override
  bool shouldCustomizeBarrageItem(Barrage barrage) {     /// 当弹幕消息的
数据模型，判断当前弹幕消息是否需要自定义
    if (barrage.extInfo.keys.isNotEmpty) {
      return true;
    }
    return false;
  }
```

```
}

/// 2、给 BarrageDisplayWidget 设置 setCustomBarrageBuilder
_displayController.setCustomBarrageBuilder(GiftBarrageItemBuilder());
```

Android

弹幕消息样式默认有两种：普通弹幕消息样式 和 自定义消息样式。具体样式使用整数表示，普通弹幕消息样式是0。
如果您需要更多消息样式（如礼物发送回显），需要重写 BarrageStreamView 的代理 BarrageItemTypeDelegate，并实现新增样式 BarrageItemAdapter。

- **重写 BarrageItemTypeDelegate 代理，支持新的样式 GIFT_VIEW_TYPE_1。**

```
public static final int    GIFT_VIEW_TYPE_1        = 1;
public class BarrageViewTypeDelegate implements
BarrageItemTypeDelegate {

    @Override
    public int getItemViewType(int position, Barrage barrage) {
        if (barrage.extInfo != null &&
barrage.extInfo.containsKey(GIFT_VIEW_TYPE)) {
            String viewTypeString =
String.valueOf(barrage.extInfo.get(GIFT_VIEW_TYPE));
            if
(String.valueOf(GIFT_VIEW_TYPE_1).equals(viewTypeString)) {
                return GIFT_VIEW_TYPE_1;
            }
        }
        return 0;
    }
}

mBarrageStreamView.setItemTypeDelegate(new
BarrageViewTypeDelegate());
```

- **实现自定义样式的 adapter，并设置给样式 GIFT_VIEW_TYPE_1 。**

```
public class GiftBarrageAdapter implements BarrageItemAdapter {
    @Override
```

```
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup
parent) {
        LinearLayout ll = new LinearLayout(mContext);
        ll.addView(new TextView(mContext));
        return new GiftViewHolder(ll, mDefaultGiftIcon);
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder,
int position, Barrage barrage) {
        GiftViewHolder viewHolder = (GiftViewHolder) holder;
        viewHolder.bind(barrage);
    }
    // GiftViewHolder
    ...
}

mBarrageStreamView.setItemAdapter(GIFT_VIEW_TYPE_1, new
GiftBarrageAdapter(mContext));
```

常见问题

弹幕组件初始化时机

由于弹幕组件的 controller 需要接入一些直播间信息参数，所以需要在**观众进入直播间**或者**主播创建直播间**后，再加载弹幕组件。

客户端 API（TUILiveKit）
Android
VideoLiveKit

最近更新时间：2025-05-07 16:49:52

API 简介

VideoLiveKit 是在线直播组件的 **UI 使用相关的接口**，使用 VideoLiveKit API，您可以通过简单接口快速实现一个在线直播场景，如果您想要体验并调试互动直播效果，请阅读 Demo 快速跑通 。如果您想把我们的功能直接集成到您的项目中，请阅读 快速接入(TUILiveKit) 。

> **说明：**
> 如果您有自己的 UI 设计，想结合 TUILiveKit 业务自行开发，我们同时提供有灵活性更好，功能更全面的 RTC Room Engine SDK，您可以通过查看我们的 RTC Room Engine SDK API 文档，了解更多功能。

API 概览

API	描述
createInstance	创建 VideoLiveKit 实例（单例模式）
startLive	通过 roomId 开启一个直播间。
joinLive	通过 roomId 加入一个直播间。

API 详情

createInstance

获取一个 VideoLiveKit 对象实例。

```
static VideoLiveKit createInstance(Context context)
```

参数：

参数	类型	说明	默认值	含义

context	Context	必填	-	android 上下文对象

返回值： VideoLiveKit

startLive

通过 roomId 开启一个直播间。

```
void startLive(String roomId);
```

参数：

参数	类型	说明	默认值	含义
roomId	String	必填	-	直播间 ID

返回值： void

joinLive

通过 roomId 加入一个直播间。

```
void joinLive(String roomId);
```

参数：

参数	类型	说明	默认值	含义
roomId	String	必填	-	直播间 ID

返回值： void

LiveCoreView

最近更新时间：2025-04-03 16:12:02

API 简介

LiveCoreView 是我们开发视频直播 UIKit 的一个基础控件，该核心控件提供了 开播前画面预览、开启视频直播、关闭视频直播，直播间内和观众连线，跨房和其他主播连线等丰富的 API。

API 概览

API	描述
LiveCoreView	创建 LiveCoreView 对象，支持**代码创建**和 **xml 加载**两种方式。
startCamera	开启摄像头采集，并将采集到的画面显示到 LiveCoreView 上。
startMicrophone	打开本地麦克风
muteMicrophone	是否暂停发布本地的音频流
stopCamera	关闭本地摄像头
stopMicrophone	关闭本地麦克风
startLiveStream	主播创建直播间并开始推流
stopLiveStream	主播停止推流并销毁直播间
joinLiveStream	观众加入某个主播的直播间
leaveLiveStream	观众离开某个主播的直播间
requestIntraRoomConnection	观众请求和主播连线
cancelIntraRoomConnection	观众取消和主播连线的请求
respondIntraRoomConnection	主播响应观众连线的请求
disconnectUser	主播断开连线的观众
terminateIntraRoomConnection	观众自己停止和主播的连线

版权所有：腾讯云计算（北京）有限责任公司

requestCrossRoomConnection	主播请求和另外一个直播间的主播连线
cancelCrossRoomConnection	主播取消和另外一个直播间的主播连线的请求
respondToCrossRoomConnection	主播响应连线请求
terminateCrossRoomConnection	主播断开连线
registerConnectionObserver	注册一个连线事件回调
unregisterConnectionObserver	反注册一个连线事件回调
setLayoutMode	设置连线主播视频画面的布局模式
setVideoViewAdapter	设置为连线主播视频画面上添加挂件的视图适配器

API 详情

LiveCoreView

创建一个 LiveCoreView 对象实例。同时支持**代码创建**和 **xml 加载**两种方式创建。

```
public LiveCoreView(Context context)
```

参数：

参数	类型	含义
context	Context	Android 上下文对象

返回值：LiveCoreView

startCamera

开始摄像头采集并将采集到的画面显示到 LiveCoreView 视图上。

```
public void startCamera(boolean useFrontCamera, ActionCallback callback)
```

参数：

参数	类型	含义
useFrontCamera	boolean	true：使用前置摄像头；false：使用后置摄像头
callback	ActionCallback	操作的回调

返回值：void

startMicrophone

打开本地麦克风。

```
void startMicrophone(ActionCallback callback)
```

参数：

参数	类型	含义
callback	ActionCallback	操作的回调

返回值：void

muteMicrophone

是否暂停发布本地的音频流。

```
void muteMicrophone(boolean mute)
```

参数：

参数	类型	含义
mute	boolean	true：暂停发布视频流，false：正常发布视频流

返回值：void

stopCamera

关闭本地摄像头。

```
void stopCamera()
```

参数：
返回值： void

stopMicrophone

关闭本地麦克风。

```
void stopMicrophone()
```

参数：
返回值： void

startLiveStream

主播创建直播间并开始推流。

```
void startLiveStream(RoomInfo roomInfo, GetRoomInfoCallback callback)
```

参数：

参数	类型	含义
roomInfo	RoomInfo	创建直播间的信息
callback	ActionCallback	操作的回调

返回值： void

stopLiveStream

主播停止推流并销毁直播间。

```
void stopLiveStream(ActionCallback callback)
```

参数：

参数	类型	含义
callback	ActionCallback	操作的回调

返回值： void

joinLiveStream

观众加入某个主播的直播间。

```
void joinLiveStream(String roomId, GetRoomInfoCallback callback)
```

参数：

参数	类型	含义
roomId	String	直播间 ID
callback	ActionCallback	操作的回调

返回值： void

leaveLiveStream

观众离开某个主播的直播间。

```
void leaveLiveStream(ActionCallback callback)
```

参数：

参数	类型	含义
callback	ActionCallback	操作的回调

返回值： void

requestIntraRoomConnection

观众请求和主播连线。

```
void requestIntraRoomConnection(String userId, int timeout, boolean
openCamera, ActionCallback callback)
```

参数：

参数	类型	含义
userId	String	用户 ID，如果不填，则代表主播的用户 ID。

timeout	int	请求超时时长，单位： 秒。
openCamera	boolean	连麦成功后是否打开摄像头。true:视频连麦， false：语音连麦。
callback	ActionCallback	操作的回调。

返回值： void

cancelIntraRoomConnection

观众取消和主播连线的请求。

```
void cancelIntraRoomConnection(String userId, ActionCallback callback)
```

参数：

参数	类型	含义
userId	String	取消连麦的用户 ID，如果不填，则代表主播的用户 ID。
callback	ActionCallback	操作的回调

返回值： void

respondIntraRoomConnection

主播响应观众连线的请求。

```
void respondIntraRoomConnection(String userId, boolean isAccepted,
ActionCallback callback)
```

参数：

参数	类型	含义
userId	String	响应用户的用户 ID
isAccepted	isAccepted	是否接受连麦请求，true：同意连麦请求，false：拒绝连麦请求
callback	ActionCallback	操作的回调

返回值： void

disconnectUser

主播断开连线的观众。

```
void disconnectUser(String userId, ActionCallback callback)
```

参数：

参数	类型	含义
userId	String	主播需要断开连麦的用户 ID
callback	ActionCallback	操作的回调

返回值： void

terminateIntraRoomConnection

观众自己停止和主播的连线。

```
void terminateIntraRoomConnection()
```

参数:无
返回值： void

requestCrossRoomConnection

主播请求和另外一个直播间的主播连线。

```
void requestCrossRoomConnection(String roomId, int timeout, ActionCallback
callback)
```

参数：

参数	类型	含义
roomId	String	请求跨房连线的房间 ID。
timeout	int	请求的超时时长，单位：秒。
callback	ActionCallback	操作的回调。

返回值： void

cancelCrossRoomConnection

主播取消和另外一个直播间的主播连线的请求。

```
void cancelCrossRoomConnection(String roomId, ActionCallback callback)
```

参数：

参数	类型	含义
roomId	String	取消连线的房间 ID
callback	ActionCallback	操作的回调

返回值： void

respondToCrossRoomConnection

主播响应连线请求。

```
void respondToCrossRoomConnection(String roomId, boolean isAccepted,
ActionCallback callback)
```

参数：

参数	类型	含义
roomId	String	响应连线的房间 ID
isAccepted	boolean	是否同意连线，true：同意连线，false：拒绝连线
callback	ActionCallback	操作的回调

返回值： void

terminateCrossRoomConnection

主播断开连线。

```
void terminateCrossRoomConnection()
```

参数：无
返回值：void

registerConnectionObserver

注册一个连线事件回调。

```
void registerConnectionObserver(ConnectionObserver observer)
```

参数：

参数	类型	含义
observer	ConnectionObserver	连线事件的回调对象

返回值：void

unregisterConnectionObserver

反注册一个连线事件回调。

```
void unregisterConnectionObserver(ConnectionObserver observer)
```

参数：

参数	类型	含义
observer	ConnectionObserver	连线事件的回调对象

返回值：void

setLayoutMode

设置连线主播视频画面的布局模式。

```
void setLayoutMode(LayoutMode layoutModel, String layoutJson)
```

参数：

参数	类型	含义
layoutMod	LayoutM	连线时的布局模式，支持宫格布局、浮窗布局、自定义布局。

el	ode	
layoutJson	String	布局的 json 字符串

返回值： void

setVideoViewAdapter

设置为连线主播视频画面上添加挂件的视图适配器。

```
void setVideoViewAdapter(LiveCoreViewDefine.VideoViewAdapter viewAdapter)
```

参数：

参数	类型	含义
viewAdapter	VideoViewAdapter	连线主播视频画面上添加挂件的视图适配器

返回值： void

LiveCoreViewDefine

最近更新时间：2025-04-03 16:12:03

类型定义

类型	描述
ConnectionObserver	为核心控件设置连线的回调事件。
LayoutMode	连线时的布局模式，支持宫格布局、浮窗布局、自定义布局。
VideoViewAdapter	连线视图适配器接口，您可以通过实现该接口来向您的每个音视频流视图上添加挂件。

ConnectionObserver

类型	说明
onConnectedUsersUpdated	连麦的用户列表发生改变的回调。
onUserConnectionRequest	收到连麦请求的回调。
onUserConnectionCancelled	收到取消连麦请求的回调。
onUserConnectionAccepted	连麦请求被同意的回调。
onUserConnectionRejected	连麦请求被拒绝的回调。
onUserConnectionTimeout	连麦请求超时的回调。
onUserConnectionTerminated	主播断开和此观众连麦的回调。
onUserConnectionExited	观众主动断开连线的回调。
onConnectedRoomsUpdated	跨房连线的房间列表发生改变回调。
onCrossRoomConnectionRequest	收到跨房连线请求的回调。
onCrossRoomConnectionCancelled	收到取消跨房连线请求的回调。
onCrossRoomConnectionAccepted	收到同意跨房连线的回调
onCrossRoomConnectionReject	收到拒绝跨房连线的回调

ed	
onCrossRoomConnectionTimeout	收到跨房连线超时的回调。
onCrossRoomConnectionExited	收到断开跨房连线的回调。
onRoomDismissed	收到房间销毁的回调。

LayoutMode

连线时的布局模式

类型	描述
GRID_LAYOUT	宫格布局。
FLOAT_LAYOUT	浮窗布局。
FREE_LAYOUT	自定义布局。

VideoViewAdapter

连线视图适配器接口，您可以通过实现该接口来向您的每个音视频流视图上添加挂件。

API	说明
createCoGuestView	创建连麦观众视图时回调，您通过该 API 创建的 View 会被显示到连麦观众的视图上。
updateCoGuestView	更新连麦观众视图时回调。
createCoHostView	创建连线主播视图时回调，您通过该 API 创建的 View 会被显示到连线主播的视图上。
updateCoHostView	更新连线主播视图时回调。

回调事件详情

onConnectedUsersUpdated

连麦的用户列表发生改变的回调。

```
void onConnectedUsersUpdated(List<UserInfo> userList, List<UserInfo>
joinList, List<UserInfo> leaveList);
```

参数：

参数	类型	描述
userList	List<UserInfo>	连麦的用户列表
joinList	List<UserInfo>	新加入的连麦用户
leaveList	List<UserInfo>	离开的连麦用户

返回值： void

onUserConnectionRequest

收到连麦请求的回调。

```
void onUserConnectionRequest(UserInfo inviterUser);
```

参数：

参数	类型	描述
inviterUser	UserInfo	申请连麦的用户信息

返回值： void

onUserConnectionCancelled

收到取消连麦请求的回调。

```
void onUserConnectionCancelled(UserInfo inviterUser);
```

参数：

参数	类型	描述
inviterUser	UserInfo	取消连麦的用户信息

返回值： void

onUserConnectionAccepted

连麦请求被同意的回调。

```
void onUserConnectionAccepted(UserInfo userInfo);
```

参数：

参数	类型	描述
userInfo	UserInfo	同意连麦的用户信息

返回值： void

onUserConnectionRejected

连麦请求被拒绝的回调。

```
void onUserConnectionRejected(UserInfo userInfo);
```

参数：

参数	类型	描述
userInfo	UserInfo	拒绝连麦的用户信息

返回值： void

onUserConnectionTimeout

连麦请超时的回调。

```
void onUserConnectionTimeout(UserInfo userInfo);
```

参数：

参数	类型	描述
userInfo	UserInfo	连麦请求超时的用户信息

返回值： void

onUserConnectionTerminated

主播断开和此观众连麦的回调。

```
vvoid onUserConnectionTerminated();
```

版权所有：腾讯云计算（北京）有限责任公司

参数：暂无
返回值：void

onUserConnectionExited

连麦的用户断开连麦的回调。

```
void onUserConnectionExited(UserInfo userInfo);
```

参数：

参数	类型	描述
userInfo	UserInfo	断开连麦的用户信息

返回值：void

onConnectedRoomsUpdated

连线的房间列表发生改变的回调。

```
void onConnectedRoomsUpdated(List<RoomInfo> roomList);
```

参数：

参数	类型	描述
roomList	List<RoomInfo>	连线的房间列表

返回值：void

onCrossRoomConnectionRequest

收到跨房连线的请求的回调。

```
void onCrossRoomConnectionRequest(RoomInfo roomInfo);
```

参数：

参数	类型	描述
roomInfo	RoomInfo	申请连线的直播间信息

返回值：void

版权所有：腾讯云计算（北京）有限责任公司

onCrossRoomConnectionCancelled

收到取消跨房连线请求的回调。

```
void onCrossRoomConnectionCancelled(RoomInfo roomInfo);
```

参数：

参数	类型	描述
roomInfo	RoomInfo	取消申请连线的直播间信息

返回值： void

onCrossRoomConnectionAccepted

收到同意跨房连线请求的回调。

```
void onCrossRoomConnectionAccepted(RoomInfo roomInfo);
```

参数：

参数	类型	描述
roomInfo	RoomInfo	同意连线的直播间信息

返回值： void

onCrossRoomConnectionRejected

收到拒绝跨房连线请求的回调。

```
void onCrossRoomConnectionRejected(RoomInfo roomInfo);
```

参数：

参数	类型	描述
roomInfo	RoomInfo	拒绝连线的直播间信息

返回值： void

onCrossRoomConnectionTimeout

收到房连线请求超时的回调。

```
void onCrossRoomConnectionTimeout(RoomInfo inviter, RoomInfo invitee);
```

参数：

参数	类型	描述
inviter	RoomInfo	申请连线的直播间信息
invitee	RoomInfo	被邀请连线的直播间信息

返回值： void

onCrossRoomConnectionExited

连麦的用户列表发生改变的回调。

```
void onCrossRoomConnectionExited(RoomInfo roomInfo);
```

参数：

参数	类型	描述
roomInfo	RoomInfo	退出连线的直播间信息

返回值： void

onRoomDismissed

直播间被销毁的回调。

```
void onRoomDismissed(String roomId);
```

参数：

参数	类型	描述
roomId	String	房间 ID

返回值： void

createCoGuestView

创建观众连麦的挂件视图，该视图会被添加到 该用户的视频流画面上面。

```
View createCoGuestView(TUIRoomDefine.UserInfo userInfo);
```

参数：

参数	类型	描述
userInfo	UserInfo	连麦用户的用户信息

返回值：View

updateCoGuestView

更新挂件视图的回调，一般情况下，您不需要关系此回调，您可以根据您自己的挂件数据状态来更新您设置的挂件视图，除非您的挂件依赖于userInfo的改变。

```
void updateCoGuestView(TUIRoomDefine.UserInfo userInfo, View coGuestView);
```

参数：

参数	类型	描述
userInfo	UserInfo	连麦用户的用户信息
coGuestView	View	您通过 createCoGuestView api 创建的挂件视图

返回值：void

createCoHostView

创建主播连线的挂件视图，该视图会被添加到 该连线主播的视频流画面上面。

```
View createCoHostView(TUILiveConnectionManager.ConnectionUser
connectionUser);
```

参数：

参数	类型	描述
connectionUser	ConnectionUser	连线主播的用户信息

返回值：View

updateCoHostView

更新挂件视图的回调，一般情况下，您不需要关系此回调，您可以根据您自己的挂件数据状态来更新您设置的挂件视图，除非您的挂件依赖于connectionUser的改变。

```
void updateCoHostView(TUILiveConnectionManager.ConnectionUser
connectionUser, View coHostView);
```

参数：

参数	类型	描述
connectionUser	ConnectionUser	连线主播的信息
coHostView	View	您通过 createCoHostView api 创建的挂件视图

返回值： void

iOS VideoLiveKit

最近更新时间：2025-05-07 16:49:52

API 简介

VideoLiveKit 是在线直播组件的 **UI 使用相关的接口**，使用 VideoLiveKit API，您可以通过简单接口快速实现一个在线直播场景，如果您想要体验并调试互动直播效果，请阅读 Demo 快速跑通 。如果您想把我们的功能直接集成到您的项目中，请阅读 快速接入(TUILiveKit) 。

> **说明：**
> 如果您有自己的 UI 设计，想结合 TUILiveKit 业务自行开发，我们同时提供有灵活性更好，功能更全面的 RTC Room Engine SDK，您可以通过查看我们的 RTC Room Engine SDK API 文档，了解更多功能。

API 概览

API	描述
createInstance	创建 VideoLiveKit 实例（单例模式）
startLive	通过 roomId 开启一个直播间
stopLive	结束直播
joinLive	通过 roomId 加入一个直播间
leaveLive	离开直播

API 详情

createInstance

获取一个 VideoLiveKit 对象实例。

```
static func createInstance() -> VideoLiveKit
```

返回值： VideoLiveKit

startLive

通过 roomId 开启一个直播间。

```
func startLive(roomId: String)
```

参数：

参数	类型	说明	含义
roomId	String	必填	直播间 ID

stopLive

通过 roomId 加入一个直播间。

```
func stopLive(onSuccess: TUISuccessBlock?, onError: TUIErrorBlock?)
```

参数：

参数	类型	说明	含义
onSuccess	TUISuccessBlock?	选填	成功回调。具体类型：()->Void?
onError	TUIErrorBlock?	选填	失败回调。具体类型：(TUIError, String)?

joinLive

通过 roomId 加入一个直播间。

```
func joinLive(roomId: String)
```

参数：

参数	类型	说明	含义
roomId	String	必填	直播间 ID

leaveLive

通过 roomId 加入一个直播间。

```
func leaveLive(onSuccess: TUISuccessBlock?, onError: TUIErrorBlock?)
```

参数：

参数	类型	说明	含义
onSuccess	TUISuccessBlock?	选填	成功回调。具体类型：()->Void?
onError	TUIErrorBlock?	选填	失败回调。具体类型：(TUIError, String)?

LiveCoreView

最近更新时间：2025-04-03 16:12:03

API 简介

LiveCoreView 是我们开发视频直播 UIKit 的一个基础控件，该核心控件提供了 开播前画面预览、开启视频直播、关闭视频直播，直播间内和观众连线，跨房和其他主播连线等丰富的 API。

API 概览

API	描述
LiveCoreView	创建 LiveCoreView 对象。
startCamera	开启摄像头采集，并将采集到的画面显示到 LiveCoreView 上。
startMicrophone	打开本地麦克风
muteMicrophone	是否暂停发布本地的音频流
stopCamera	关闭本地摄像头
stopMicrophone	关闭本地麦克风
startLiveStream	主播创建直播间并开始推流
stopLiveStream	主播停止推流并销毁直播间
joinLiveStream	观众加入某个主播的直播间
leaveLiveStream	观众离开某个主播的直播间
requestIntraRoomConnection	观众请求和主播连线
cancelIntraRoomConnection	观众取消和主播连线的请求
respondIntraRoomConnection	主播响应观众连线的请求
disconnectUser	主播断开连线的观众
terminateIntraRoomConnection	观众自己停止和主播的连线

版权所有：腾讯云计算（北京）有限责任公司

requestCrossRoomConnection	主播请求和另外一个直播间的主播连线
cancelCrossRoomConnection	主播取消和另外一个直播间的主播连线的请求
respondToCrossRoomConnection	主播响应连线请求
terminateCrossRoomConnection	主播断开连线
registerConnectionObserver	注册一个连线事件回调
unregisterConnectionObserver	反注册一个连线事件回调
setLayoutMode	设置连线主播视频画面的布局模式
delegate	设置为连线主播视频画面上添加挂件的视图适配器

API 详情

LiveCoreView

创建一个 LiveCoreView 对象实例。

```
LiveCoreView()
```

startCamera

开始摄像头采集并将采集到的画面显示到 LiveCoreView 视图上。

```
func startCamera(useFrontCamera: Bool, onSuccess: @escaping
TUISuccessBlock, onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
useFrontCamera	Bool	true：使用前置摄像头；false：使用后置摄像头
onSucces	TUISucce	操作成功的回调

s	ssBlock	
onError	TUIErrorBlock	操作失败的回调

startMicrophone

打开本地麦克风。

```
func startMicrophone(onSuccess: @escaping TUISuccessBlock, onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

muteMicrophone

是否暂停发布本地的音频流。

```
func muteMicrophone(mute: Bool)
```

参数：

参数	类型	含义
mute	Bool	true：暂停发布视频流，false：正常发布视频流

stopCamera

关闭本地摄像头。

```
func stopCamera()
```

stopMicrophone

关闭本地麦克风。

```
func stopMicrophone()
```

startLiveStream

主播创建直播间并开始推流。

```
func startLiveStream(roomInfo: TUIRoomInfo,
                     onSuccess: @escaping TUIRoomInfoBlock,
                     onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
roomInfo	RoomInfo	创建直播间的信息
onSuccess	TUIRoomInfoBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

stopLiveStream

主播停止推流并销毁直播间。

```
func stopLiveStream(onSuccess: @escaping TUISuccessBlock, onError:
@escaping TUIErrorBlock)
```

参数：

参数	类型	含义
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

joinLiveStream

观众加入某个主播的直播间。

```
func joinLiveStream(roomId: String,
```

```
                    onSuccess: @escaping TUIRoomInfoBlock,
                    onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
roomId	String	直播间 ID
onSuccess	TUIRoomInfoBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

leaveLiveStream

观众离开某个主播的直播间。

```
func leaveLiveStream(onSuccess: @escaping TUISuccessBlock, onError:
@escaping TUIErrorBlock)
```

参数：

参数	类型	含义
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

requestIntraRoomConnection

观众请求和主播连线。

```
func requestIntraRoomConnection(userId: String, timeOut: Int, openCamera:
Bool,
                                onSuccess: @escaping TUISuccessBlock,
onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义

userId	String	用户 ID，如果传入主播ID，则代表观众请求和主播连线（传入空字符串时默认代表主播UserID）。为其他用户的UserID时，则代表主播邀请对应userId的用户连线。
timeout	int	请求超时时长，单位： 秒。
openCamera	Bool	连麦成功后是否打开摄像头。true：视频连麦， false：语音连麦。
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

cancelIntraRoomConnection

观众取消和主播连线的请求。

```
func cancelIntraRoomConnection(userId: String,
                               onSuccess: @escaping TUISuccessBlock,
                               onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
userId	String	取消连麦的用户 ID，如果传入主播ID，则代表观众取消和主播连线的请求（传入空字符串时默认代表主播UserID）。为其他用户的UserID时，则代表主播取消邀请对应 userId 的用户连线。
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

respondIntraRoomConnection

主播响应观众连线的请求。

```
func respondIntraRoomConnection(userId: String, isAccepted: Bool,
                                onSuccess: @escaping TUISuccessBlock,
onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
userId	String	主播响应观众连线的观众user ID。若传入主播的User ID，则代表观众响应主播的邀请连线（传入空字符串时默认代表主播UserID）。
isAccepted	isAccepted	是否接受连麦请求，true：同意连麦请求，false：拒绝连麦请求
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

disconnectUser

主播断开连线的观众。

```
func disconnectUser(userId: String, onSuccess: @escaping TUISuccessBlock,
onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
userId	String	主播需要断开连麦的用户 ID
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

terminateIntraRoomConnection

观众自己停止和主播的连线。

```
func terminateIntraRoomConnection()
```

requestCrossRoomConnection

主播请求和另外一个直播间的主播连线。

```
func requestCrossRoomConnection(roomId: String, timeOut: Int,
                                onSuccess: @escaping TUISuccessBlock,
onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
roomId	String	请求跨房连线的房间 ID。
timeout	int	请求的超时时长，单位：秒。
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

cancelCrossRoomConnection

主播取消和另外一个直播间的主播连线的请求。

```
func cancelCrossRoomConnection(roomId: String, onSuccess: @escaping
TUISuccessBlock, onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
roomId	String	取消连线的房间 ID
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

respondToCrossRoomConnection

主播响应连线请求。

```
func respondToCrossRoomConnection(roomId: String, isAccepted: Bool,
                                  onSuccess: @escaping TUISuccessBlock,
onError: @escaping TUIErrorBlock)
```

参数：

参数	类型	含义
roomId	String	响应连线的房间 ID
isAccepted	boolean	是否同意连线，true：同意连线，false：拒绝连线
onSuccess	TUISuccessBlock	操作成功的回调
onError	TUIErrorBlock	操作失败的回调

terminateCrossRoomConnection

主播断开连线。

```
func terminateCrossRoomConnection()
```

registerConnectionObserver

注册一个连线事件回调。

```
func registerConnectionObserver(observer: ConnectionObserver)
```

参数：

参数	类型	含义
observer	Connection Observer	连线事件的回调对象

unregisterConnectionObserver

反注册一个连线事件回调。

```
func unregisterConnectionObserver(observer: ConnectionObserver)
```

参数：

参数	类型	含义

observer	ConnectionObserver	连线事件的回调对象

setLayoutMode

设置连线主播视频画面的布局模式。

```
func setLayoutMode(layoutMode: LayoutMode, layoutJson: String? = nil)
```

参数：

参数	类型	含义
layoutModel	LayoutMode	连线时的布局模式，支持宫格布局、浮窗布局、自定义布局。
layoutJson	String	布局的 json 字符串。具体设置用法可参考 设置自定义布局 。

delegate

设置为连线主播视频画面上添加挂件的视图适配器。

```
public weak var videoViewDelegate: VideoViewDelegate?
```

参数：

参数	类型	含义
viewAdapter	VideoViewDelegate	连线主播视频画面上添加挂件的视图适配器

LiveCoreViewDefine

最近更新时间：2025-04-03 16:12:03

类型定义

类型	描述
ConnectionObserver	为核心控件设置连线的回调事件。
LayoutMode	连线时的布局模式，支持宫格布局、浮窗布局、自定义布局。
VideoViewDelegate	连线视图适配器接口，您可以通过实现该接口来向您的每个音视频流视图上添加挂件。

ConnectionObserver

类型	说明
onConnectedUsersUpdated	连麦的用户列表发生改变的回调。
onUserConnectionRequest	收到连麦请求的回调。
onUserConnectionCancelled	收到取消连麦请求的回调。
onUserConnectionAccepted	连麦请求被同意的回调。
onUserConnectionRejected	连麦请求被拒绝的回调。
onUserConnectionTimeout	连麦请求超时的回调。
onUserConnectionTerminated	主播断开和此观众连麦的回调。
onUserConnectionExited	观众主动断开连线的回调。
onConnectedRoomsUpdated	跨房连线的房间列表发生改变回调。
onCrossRoomConnectionRequest	收到跨房连线请求的回调。
onCrossRoomConnectionCancelled	收到取消跨房连线请求的回调。
onCrossRoomConnectionAccepted	收到同意跨房连线的回调
onCrossRoomConnectionReject	收到拒绝跨房连线的回调

ed	
onCrossRoomConnectionTimeout	收到跨房连线超时的回调。
onCrossRoomConnectionExited	收到断开跨房连线的回调。
onRoomDismissed	收到房间销毁的回调。

LayoutMode

连线时的布局模式

类型	描述
gridLayout	宫格布局。
floatLayout	浮窗布局。
freeLayout	自定义布局。

VideoViewDelegate

连线视图适配器接口，您可以通过实现该接口来向您的每个音视频流视图上添加挂件。

API	说明
createCoGuestView	创建连麦观众视图时回调，您通过该 API 创建的 View 会被显示到连麦观众的视图上。
updateCoGuestView	更新连麦观众视图时回调。
createCoHostView	创建连线主播视图时回调，您通过该 API 创建的 View 会被显示到连线主播的视图上。
updateCoHostView	更新连线主播视图时回调。

回调事件详情

onConnectedUsersUpdated

连麦的用户列表发生改变的回调。

```
func onConnectedUsersUpdated(userList: [TUIUserInfo], joinList:
[TUIUserInfo], leaveList: [TUIUserInfo])
```

参数：

参数	类型	描述
userList	TUIUserInfo	连麦的用户列表
joinList	TUIUserInfo	新加入的连麦用户
leaveList	TUIUserInfo	离开的连麦用户

onUserConnectionRequest

收到连麦请求的回调。

```
func onUserConnectionRequest(inviterUser: TUIUserInfo)
```

参数：

参数	类型	描述
inviterUser	TUIUserInfo	申请连麦的用户信息

onUserConnectionCancelled

收到取消连麦请求的回调。

```
func onUserConnectionCancelled(inviterUser: TUIUserInfo)
```

参数：

参数	类型	描述
inviterUser	TUIUserInfo	取消连麦的用户信息

onUserConnectionAccepted

连麦请求被同意的回调。

```
func onUserConnectionAccepted(userInfo: TUIUserInfo)
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	同意连麦的用户信息

版权所有：腾讯云计算（北京）有限责任公司

onUserConnectionRejected

连麦请求被拒绝的回调。

```
func onUserConnectionRejected(userInfo: TUIUserInfo)
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	拒绝连麦的用户信息

onUserConnectionTimeout

连麦请超时的回调。

```
func onUserConnectionTimeout(userInfo: TUIUserInfo)
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	连麦请求超时的用户信息

onUserConnectionTerminated

主播断开和此观众连麦的回调。

```
func onUserConnectionTerminated()
```

onUserConnectionExited

连麦的用户断开连麦的回调。

```
func onUserConnectionExited(userInfo: TUIUserInfo)
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	断开连麦的用户信息

onConnectedRoomsUpdated

连线的房间列表发生改变的回调。

```
func onConnectedRoomsUpdated(hostUserList: [TUIConnectionUser])
```

参数：

参数	类型	描述
hostUserList	TUIConnectionUser	连线房间的房主列表

onCrossRoomConnectionRequest

收到跨房连线的请求的回调。

```
func onCrossRoomConnectionRequest(hostUser: TUIConnectionUser)
```

参数：

参数	类型	描述
hostUser	TUIConnectionUser	申请连线的直播间房主信息

onCrossRoomConnectionCancelled

收到取消跨房连线请求的回调。

```
func onCrossRoomConnectionCancelled(hostUser: TUIConnectionUser)
```

参数：

参数	类型	描述
hostUser	TUIConnectionUser	取消申请连线的直播间房主信息

onCrossRoomConnectionAccepted

收到同意跨房连线请求的回调。

```
func onCrossRoomConnectionAccepted(hostUser: TUIConnectionUser)
```

参数：

参数	类型	描述
hostUser	TUIConnectionUser	同意连线的直播间房主信息

onCrossRoomConnectionRejected

收到拒绝跨房连线请求的回调。

```
func onCrossRoomConnectionRejected(hostUser: TUIConnectionUser)
```

参数：

参数	类型	描述
hostUser	TUIConnectionUser	拒绝连线的直播间房主信息

onCrossRoomConnectionTimeout

收到房连线请求超时的回调。

```
func onCrossRoomConnectionTimeout(inviter: TUIConnectionUser, invitee:
TUIConnectionUser)
```

参数：

参数	类型	描述
inviter	TUIConnectionUser	申请连线的直播间房主信息
invitee	TUIConnectionUser	被邀请连线的直播间房主信息

onCrossRoomConnectionExited

连麦的用户列表发生改变的回调。

```
func onCrossRoomConnectionExited(hostUser: TUIConnectionUser)
```

参数：

参数	类型	描述
hostUser	TUIConnectionUser	退出连线的直播间信息

onRoomDismissed

直播间被销毁的回调。

```
func onRoomDismissed(roomId: String)
```

参数：

参数	类型	描述
roomId	String	房间 ID

createCoGuestView

创建观众连麦的挂件视图，该视图会被添加到 该用户的视频流画面上面。

```
func createCoGuestView(userInfo: TUIUserInfo) -> UIView?
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	连麦用户的用户信息

返回值：UIView?

updateCoGuestView

更新挂件视图的回调，一般情况下，您不需要关系此回调，您可以根据您自己的挂件数据状态来更新您设置的挂件视图，除非您的挂件依赖于userInfo的改变。

```
func updateCoGuestView(userInfo: TUIUserInfo, coGuestView: UIView)
```

参数：

参数	类型	描述
userInfo	TUIUserInfo	连麦用户的用户信息

coGuestView	UIView	您通过 createCoGuestView api 创建的挂件视图

createCoHostView

创建主播连线的挂件视图，该视图会被添加到 该连线主播的视频流画面上面。

```
func createCoHostView(connectionUser: TUIConnectionUser) -> UIView?
```

参数：

参数	类型	描述
connectionUser	TUIConnectionUser	连线主播的用户信息

返回值：UIView?

updateCoHostView

更新挂件视图的回调，一般情况下，您不需要关系此回调，您可以根据您自己的挂件数据状态来更新您设置的挂件视图，除非您的挂件依赖于connectionUser的改变。

```
func updateCoHostView(connectionUser: TUIConnectionUser, coHostView:
UIView)
```

参数：

参数	类型	描述
connectionUser	TUIConnectionUser	连线主播的信息
coHostView	UIView	您通过 createCoHostView api 创建的挂件视图

Web
TUILiveKit API

最近更新时间：2025-04-03 16:12:03

API 简介

TUILiveKit API 是直播组件的**含 UI 接口**，使用 TUILiveKit API，您可以通过简单接口快速实现一个直播场景，更详细的接入步骤，详见：快速接入 TUILiveKit 。

API 概览

- `<LiveMainView />`: TUILiveKit UI 组件主体。
- liveRoom：依赖 `LiveMainView` 提供的 API。

API	描述
getRoomEngine	获取 roomEngine 实例。如果 roomEngine 不存在，则返回 null。
on	监听指定类型的事件。当事件发生时，将调用回调函数。
off	取消监听指定类型的事件。
login	登录直播系统。
logout	登出直播系统。
start	开始一个新的直播。
join	加入一个已经存在的直播。
leave	离开当前直播。
dismiss	解散当前直播。
setSelfInfo	设置自己的用户信息。
setLanguage	设置界面语言。
setTheme	设置界面主题。

liveRoom API 详情

liveRoom 提供了一系列用于管理和控制在线直播功能的方法。通过实现这个接口，开发者可以方便地在他们的应用中集成在线直播的功能。

getRoomEngine

获取 roomEngine 实例。如果 roomEngine 不存在，则返回 null。

```
import { liveRoom, TUIRoomEngine } from '@tencentcloud/livekit-web-vue3';
TUIRoomEngine.once('ready', () => {
    const roomEngine = liveRoom.getRoomEngine();
});
```

Returns: *TUIRoomEngine | null*

on

监听指定类型的事件。当事件发生时，将调用回调函数。
参数：

参数	类型	默认值	含义
eventType	RoomEvent	-	要监听的事件类型
callback	() => void	-	事件发生时调用的回调函数

```
import { liveRoom, RoomEvent } from '@tencentcloud/livekit-web-vue3';
liveRoom.on(RoomEvent.RoomStart, () => {
  console.log('[liveRoom] The live has already started.')
});
liveRoom.on(RoomEvent.ROOM_DISMISS, () => {
  console.log('[liveRoom] The live has been dismissed')
});
```

Returns: *void*

off

取消监听指定类型的事件。
参数：

参数	类型	默认值	含义
eventT	RoomEvent	-	要取消监听的事件类型

ype			
callbac k	() => void	-	之前添加的回调函数

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.off('event', callback);
```

Returns: *void*

login

登录直播系统。
参数：

参数	类型	默认值	含义
params	{sdkAppId: number; userId: string; userSig: string; tim?: ChatSDK}	-	登录参数对象
sdkAppId	number	-	在 实时音视频控制台 单击**应用管理 > 创建应用**，创建新应用之后，即可 **应用信息**中获取 sdkAppId 信息。
userId	string	-	用户ID建议限制长度为32字节，只允许包含大小写英文字母(a-zA-Z)、数字(0-9)及下划线和连词符。
userSig	string	-	userSig 签名 计算 userSig 的方式请参见 UserSig 相关。
tim	ChatSDK (可选)	-	若您在接入 roomEngine 的同时想使用即时通信 SDK 的更多能力，您可以将创建的 tim 实例传入 TUIRoomEngine。tim 实例的创建方式请参见 TIM.create 。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.login({
  sdkAppId: 123456,
  userId: 'testUser',
  userSig: 'testSig'
```

```
});
```

Returns：*Promise<void>*

logout

登出直播系统。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.logout();
```

Returns：*Promise<void>*

start

开始一个新的直播。
参数：

参数	类型	默认值	含义
roomId	string	-	直播房间 ID
params	StartParams	-	开始直播的参数

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.start('123456', {
  roomName: 'TestRoom',
  isOpenCamera: false,
  isOpenMicrophone: false,
});
```

Returns：*Promise<void>*

join

加入一个已经存在的直播。
参数：

参数	类型	默认值	含义
roomId	string	-	直播房间 ID
params	JoinParams	-	加入直播的参数

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.join('123456', {
  isOpenCamera: false,
  isOpenMicrophone: false,
});
```

Returns：*Promise<void>*

leave

离开当前直播。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.leave();
```

Returns：*Promise<void>*

dismiss

解散当前直播。

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.dismiss();
```

Returns：*Promise<void>*

setSelfInfo

设置自己的用户信息。
参数：

参数	类型	默认值	含义
options	{userName: string; avatarUrl: string}	-	用户信息对象
userName	string（可选）	-	用户昵称
avatarUrl	string（可选）	-	用户头像

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.setSelfInfo({
  userName: 'test-name',
  avatarUlr: 'https://avatar.png'
```

```
});
```

Returns：*Promise<void>*

setLanguage

设置界面语言。
参数：

参数	类型	默认值	含义
language	'zh-CN' | 'en-US'	-	语言类型

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.setLanguage('en-US');
```

Returns：*void*

setTheme

设置界面主题。
参数：

参数	类型	默认值	含义
theme	'LIGHT' | 'DARK'	-	主题类型

```
import { liveRoom } from '@tencentcloud/livekit-web-vue3';
liveRoom.setTheme('DARK');
```

Returns：*void*

类型定义

RoomEvent（枚举值）

参数	类型	描述
ROOM_START	string	创建直播
ROOM_JOIN	string	加入直播
ROOM_LEAVE	string	离开直播

ROOM_DISMISS	string	直播解散
KICKED_OFFLINE	string	用户被移下线
KICKED_OUT	string	用户被移出直播
USER_LOGOUT	string	用户登出
ROOM_ERROR	string	直播错误
ROOM_NEED_PASSWORD	string	直播间密码

FeatureButton（枚举值）

参数	类型	描述
SwitchTheme	string	切换主题功能按钮
SwitchLanguage	string	切换语言功能按钮

StartParams

参数	类型	描述	默认值
roomName	string（可选）	直播间名称	-
isOpenCamera	boolean（可选）	是否开启摄像头	false
isOpenMicrophone	boolean（可选）	是否开启麦克风	false
defaultCameraId	string（可选）	默认摄像头 ID	-
defaultMicrophoneId	string（可选）	默认麦克风 ID	-
defaultSpeakerId	string（可选）	默认扬声器 ID	-

JoinParams

参数	类型	描述	默认值
isOpenCamera	boolean（可选）	是否开启摄像头	false
isOpenMicrophone	boolean（可	是否开启麦克风	false

版权所有：腾讯云计算（北京）有限责任公司

	选）		
defaultCameraId	string（可选）	默认摄像头 ID	-
defaultMicrophoneId	string（可选）	默认麦克风 ID	-
defaultSpeakerId	string（可选）	默认扬声器 ID	-

服务端 API（TUILiveKit）
账号系统
账号管理

最近更新时间：2025-06-27 17:58:42

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

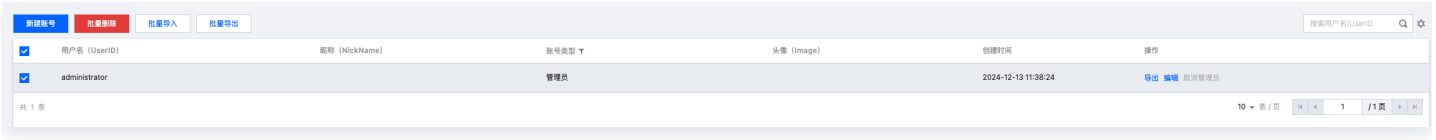
1. 在账号管理页面，单击**新建账号**。
2. 在弹出的新建账号对话框中，配置以下参数：



   - 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：



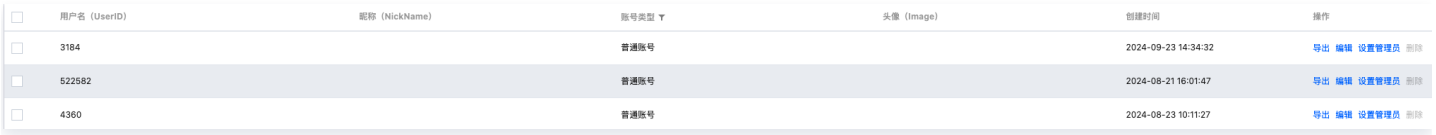
- 用户名：不可编辑。
- 昵称：可编辑用户昵称，非必填。
- 账号类型：不可编辑。
- 性别：可编辑性别，非必填。
- 生日：可编辑生日，非必填。
- 所在地：可编辑所在地，非必填。
- 个性签名：可编辑个性签名，非必填。
- 语言：可编辑语言，非必填。
- 头像：可编辑头像，非必填。
- 等级：可编辑等级，非必填。
- 角色：可编辑角色，非必填。

版权所有：腾讯云计算（北京）有限责任公司

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**。

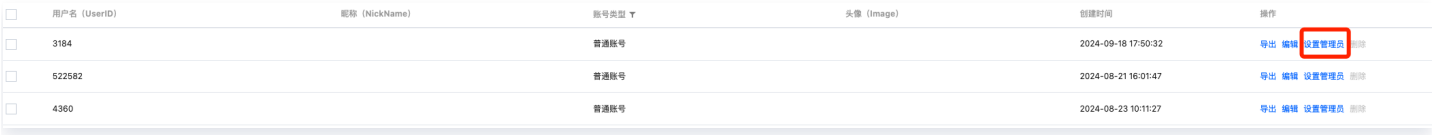


3. 在弹出的导出对话框中提示导出成功后，单击**下载**，即完成导出。



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：2025-06-27 18:04:11

登录鉴权简介

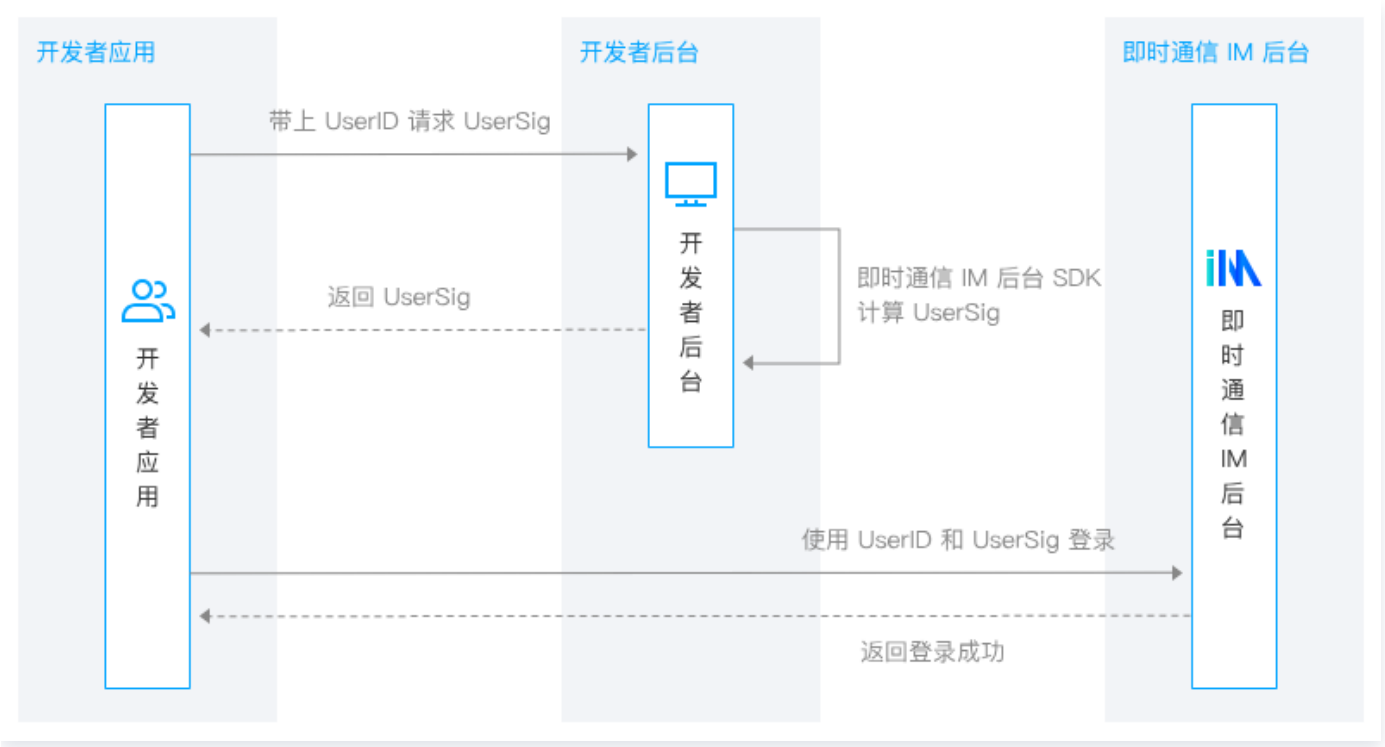
TUICallKit、TUIRoomKit、TUILiveKit 与即时通信 IM 共用同一套登录鉴权体系，即时通信 IM 的前身是 QQ 的即时通信系统，我们抽离 QQ 的通用模块，并将其整合成适合终端接入的 IM SDK 及后台服务。您可以把 IM SDK 理解为一个没有用户交互界面的 QQ，把 IM SDK 集成到您的 App 里，就相当于把一个 QQ 的内核集成在您的 App 内部。
QQ 可以用来收发消息，但前提是您必须先登录才能使用。登录 QQ 用的是 QQ 号和密码，登录 IM SDK 需要使用您指定的用户名（ `UserID` ）和密码（ `UserSig` ）。

- **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：2025-06-27 17:58:42

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。

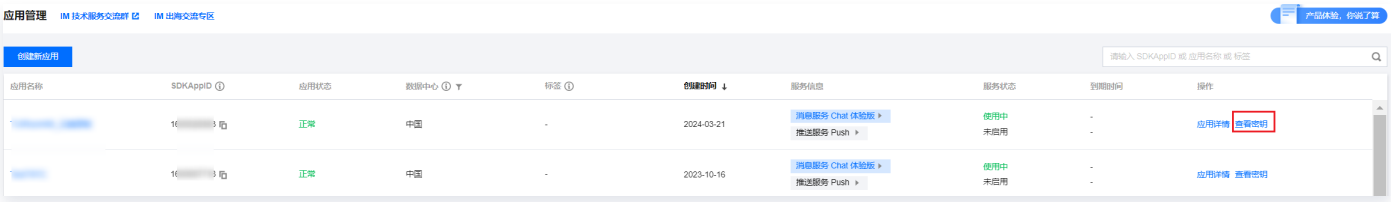
观看视频

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：2025-06-27 17:58:42

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_I	String	昵称	有	长度不得超过500个字节

M_Nick				
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。

REST API
REST API 简介

最近更新时间：2025-04-03 16:12:03

REST API 是在线直播房间 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

要调用 REST API，您必须开通 TUILiveKit 服务 。

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
https	请求协议	请求协议为 HTTPS，请求方式为 POST
xxxxxx	请求域名	• 中国：`console.tim.qq.com` • 新加坡：`adminapisgp.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.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/live_engine_http_srv/create_room`，其中 `room_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/live_engine_http_srv/create_room`，其中 `create_room` 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-70e312f1de024af5a36714b7b71da224-0-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	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

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

版权所有：腾讯云计算（北京）有限责任公司

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](https://buy.cloud.tencent.com/avc) 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。

FAQ

REST API 请求有概率超时，收不到任何响应?

1. Live 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：2025-04-03 16:12:03

房间管理

功能说明	接口
创建房间	live_engine_http_srv/create_room
解散房间	live_engine_http_srv/destroy_room
更新房间信息	live_engine_http_srv/update_room_info
获取房间信息	live_engine_http_srv/get_room_info
获取直播列表	live_engine_http_srv/get_room_list
获取房间自定义信息	live_engine_http_srv/get_room_metadata
设置房间自定义信息	live_engine_http_srv/set_room_metadata
删除房间自定义信息	live_engine_http_srv/del_room_metadata

房间成员管理

功能说明	接口
获取管理员列表	live_engine_http_srv/get_admin_list
成员封禁	live_engine_http_srv/ban_member
成员解封	live_engine_http_srv/unban_member
获取封禁列表	live_engine_http_srv/get_banned_member_list
修改管理员	live_engine_http_srv/modify_admin
获取管理员列表	live_engine_http_srv/get_admin_list
成员禁言	live_engine_http_srv/mute_member
获取禁言列表	live_engine_http_srv/get_muted_member_list
获取房间成员列表	live_engine_http_srv/get_member_list

账号管理

版权所有：腾讯云计算（北京）有限责任公司

功能说明	接口
导入单个账号	im_open_login_svc/account_import
导入多个账号	im_open_login_svc/multiaccount_import
查询账号	im_open_login_svc/account_check
设置用户资料	profile/portrait_set
拉取用户资料	profile/portrait_get

版权所有：腾讯云计算（北京）有限责任公司

用户管理
账号管理
导入单个账号

最近更新时间：2025-04-03 16:12:03

功能说明

本接口用于将 App 自有账号导入 Live 账号系统，为该账号创建一个对应的内部 ID，使该账号能够使用 Live 服务。

> **注意：**
> 同一个账号重复导入仅会创建1个内部 ID，Live 和即时通信 IM 共用一套底层账号系统。

以下视频将帮助您快速了解如何将 App 自有账号导入 Live 账号系统：
观看视频

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/im_open_login_svc/account_import?
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` • 首尔：`adminapikr.im.qcloud.com` • 法兰克福：`adminapiger.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.com` • 印尼：`adminapiidn.im.qcloud.com`
v4/im_open_login_svc/acco	请求接口

unt_import	
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

```
{
    "UserID":"test",
    "Nick": "Nick_001",
    "FaceUrl": "http://www.qq.com"
}
```

请求包字段说明

字段	类型	属性	说明
UserID	String	必填	用户名，长度不超过32字节
Nick	String	选填	用户昵称，对应于 资料字段 里的 Tag_Profile_IM_Nick
FaceUrl	String	选填	用户头像，对应于 资料字段 里的 Tag_Profile_IM_Image

应答包体示例

```
{
    "ActionStatus":"OK",
    "ErrorInfo":"",
    "ErrorCode":0
}
```

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

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
40006	设置资料时服务器内部错误，请稍后重试
40601	资料字段的 Value 长度超过500字节
70169	服务端内部超时，请稍后重试
70398	账号数超限。
70399	非 IM 体验版应用，账号被删除后，三个月内不允许重新导入
70402	参数非法，请检查必填字段是否填充，或者字段的填充是否满足协议要求
70403	请求失败，需要 App 管理员权限
70500	服务器内部错误，请稍后重试

接口调试工具

通过 REST API 在线调试 工具调试本接口。

版权所有：腾讯云计算（北京）有限责任公司

导入多个账号

最近更新时间：2025-04-03 16:12:03

功能说明

本接口用于批量将 App 自有账号导入 Live 账号系统，为该账号创建一个对应的内部 ID，使该账号能够使用 Live 服务。

说明：

- 本接口单次最多支持导入100个账号，同一个账号重复导入仅会创建1个内部 ID。
- 批量导入账号时，如果账号已存在，会按新传入的参数（用户昵称、头像 URL 等）进行更新。
- Live 系统和 即时通信 IM系统共用一套账号体系。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/im_open_login_svc/multiaccount_import?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` • 首尔：`adminapikr.im.qcloud.com` • 法兰克福：`adminapiger.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.com` • 印尼：`adminapiidn.im.qcloud.com`
v4/im_open_login_svc/multiaccount_import	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

100次/秒。

新版本请求包示例

```
{
  "AccountList": [
    {
      "UserID": "UserID_001",
      "Nick": "Nick_001",
      "FaceUrl": "http://www.qq.com"
    },
    {
      "UserID": "UserID_002",
      "Nick": "Nick_002",
      "FaceUrl": "http://www.qq.com"
    },
    {
      "UserID": "UserID_002",
      "Nick": "Nick_001",
      "FaceUrl": "http://www.qq.com"
    }
  ]
}
```

新版本请求包字段说明

字段	类型	属性	说明
AccountList	Array	必填	待导入的用户列表，单次最多导入100个用户
UserID	String	必填	用户的 UserID，UserID 长度不超过32字节
Nick	String	选填	用户昵称，对应于 资料字段 里的 Tag_Profile_IM_Nick

FaceUrl	String	选填	用户头像，对应于 资料字段 里的 Tag_Profile_IM_Image

应答包体示例

```
{
    "ActionStatus": "OK",
    "ErrorCode": 0,
    "ErrorInfo": "",
    "FailAccounts": [
        "UserID_002",
        "UserID_003"
    ]
}
```

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
ErrorCode	Integer	错误码： • 0表示成功 • 非0表示失败
ErrorInfo	String	错误信息
FailAccounts	Array	导入失败的账号列表

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
40006	设置资料时服务器内部错误，请稍后重试

40601	资料字段的 Value 长度超过500字节
70169	服务端内部超时，请稍后重试
70202	服务端内部超时，请稍后重试
70398	账号数超限
70399	非 IM 体验版应用，账号被删除后，三个月内不允许重新导入
70402	参数非法，请检查必填字段是否填充，或者字段的填充是否满足协议要求
70403	请求失败，需要 App 管理员权限
70500	服务器内部错误，请稍后重试

接口调试工具

通过 REST API 在线调试 工具调试本接口。

查询账号

最近更新时间：2025-04-03 16:12:03

功能说明

用于查询自有账号是否已导入 Live 系统，支持批量查询。

> **说明：**
> Live 系统和即时通信 IM 系统共用一套账号体系。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/im_open_login_svc/account_check?
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` • 首尔：`adminapikr.im.qcloud.com` • 法兰克福：`adminapiger.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.com` • 印尼：`adminapiidn.im.qcloud.com`
v4/im_open_login_svc/account_check	请求接口
sdkappid	创建应用时控制台分配的 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

版权所有：腾讯云计算（北京）有限责任公司

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

100次/秒。

请求包示例

```
{
  "CheckItem":
  [
      {
          "UserID":"UserID_1"
      },
      {
          "UserID":"UserID_2"
      }
  ]
}
```

请求包字段说明

字段	类型	属性	说明
CheckItem	Array	必填	请求检查的账号对象数组，单次请求最多支持100个账号
UserID	String	必填	请求检查的账号的 UserID

应答包体示例

```
{
    "ActionStatus": "OK",
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ResultItem": [
        {
            "UserID": "UserID_1",
            "ResultCode": 0,
            "ResultInfo": "",
            "AccountStatus": "Imported"
        },
```

```
        {
            "UserID": "UserID_2",
            "ResultCode": 0,
            "ResultInfo": "",
            "AccountStatus": "NotImported"
        }
    ]
}
```

应答包字段说明

字段	类型	说明
ActionStatus	String	请求的处理结果： • OK：表示处理成功 • FAIL：表示失败
ErrorCode	Integer	错误码： • 0：表示成功 • 非0：表示失败
ErrorInfo	String	请求处理失败时的错误信息
ResultItem	Array	单个账号的检查结果对象数组
UserID	String	请求检查的账号的 UserID
ResultCode	Integer	单个账号的检查结果： • 0：表示成功 • 非0：表示失败
ResultInfo	String	单个账号检查失败时的错误描述信息
AccountStatus	String	单个账号的导入状态：Imported 表示已导入，NotImported 表示未导入

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）请参见 错误码 。
本 API 私有错误码如下：

错误码	含义说明

70169	服务端内部超时，请稍后重试
70202	服务端内部超时，请稍后重试
70402	参数非法，请检查必填字段是否填充，或者字段的填充是否符合协议要求
70403	请求失败，需要 App 管理员权限
70500	服务器内部错误，请稍后重试

接口调试工具

通过 REST API 在线调试 工具调试本接口。

用户资料
设置资料

最近更新时间：2025-04-03 16:12:03

功能说明

资料是绑定到账号上的信息，支持 标配资料字段 和 自定义资料字段 的设置。

> **说明：**
> Live 和 即时通信 IM 共用同一套账号体系，因此账号附带的资料也是共用的。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/profile/portrait_set?
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` • 首尔：`adminapikr.im.qcloud.com` • 法兰克福：`adminapiger.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.com` • 印尼：`adminapiidn.im.qcloud.com`
v4/profile/portrait_set	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`

请求频率限制

默认接口请求频率限制：200次/秒。

请求包示例

```
{
    "From_Account":"id",
    "ProfileItem":
    [
        {
            "Tag":"Tag_Profile_IM_Nick",
            "Value":"MyNickName"
        }
    ]
}
```

请求包字段说明

字段	类型	属性	说明
From_Account	String	必填	需要设置该 UserID 的资料
ProfileItem	Array	必填	待设置的用户的资料对象数组，数组中每一个对象都包含了 Tag 和 Value
Tag	String	必填	指定要设置的资料字段的名称，资料字段
Value	uint64_t/string/bytes	必填	待设置的资料字段的值

应答包体示例

```
{
    "ActionStatus":"FAIL",
    "ErrorCode":40001,
    "ErrorInfo":"Err_Profile_Comm_Decode_Fail",
    "ErrorDisplay":""
```

```
}
```

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
ErrorCode	Integer	错误码： • 0：表示成功 • 非0：表示失败
ErrorInfo	String	详细错误信息
ErrorDisplay	String	详细的客户端展示信息

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	描述
40001	请求参数错误，请根据错误描述检查请求参数
40003	请求的用户账号不存在
40004	请求需要 App 管理员权限
40006	服务器内部错误，请稍后重试
40008	没有资料字段的写权限
40009	资料字段的 Tag 不存在
40601	资料字段的 Value 长度超过500字节
40605	标配资料字段的 Value错误，详情可参见 标配资料字段
40610	资料字段的 Value 类型不匹配，详情可参见 标配资料字段

接口调试工具

通过 REST API 在线调试 工具调试本接口。

拉取资料

最近更新时间：2025-04-03 16:12:03

功能说明

- 支持拉取 标配资料字段 和 自定义资料字段 。
- 建议每次拉取的用户数不超过100，避免因回包数据量太大导致回包失败。
- 请确保请求中的所有账号都已导入即时通信 IM，如果请求中含有未导入即时通信 IM 的账号，即时通信 IM 后台将会提示错误。

> **说明：**
> Live 和 即时通信 IM 共用同一套账号体系，因此账号附带的资料也是共用的。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/profile/portrait_get?
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` • 首尔：`adminapikr.im.qcloud.com` • 法兰克福：`adminapiger.im.qcloud.com` • 美国：`adminapiusa.im.qcloud.com` • 印尼：`adminapiidn.im.qcloud.com`
v4/profile/portrait_get	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`

请求频率限制

默认接口请求频率限制：200次/秒。

请求包示例

- **读一个用户的一个资料字段**

```
{
  "To_Account":["id1"],
  "TagList":
  [
      "Tag_Profile_IM_Nick"
  ]
}
```

- **读一个用户的多个资料字段**

```
{
  "To_Account":["id1"],
  "TagList":
  [
      "Tag_Profile_IM_Nick",
      "Tag_Profile_IM_AllowType",
      "Tag_Profile_IM_SelfSignature",
      "Tag_Profile_Custom_Test"
  ]
}
```

- **读多个用户的一个资料字段**

```
{
  "To_Account":["id1","id2","id3"],
  "TagList":
  [
      "Tag_Profile_IM_Nick"
  ]
```

```
}
```

- **读多个用户的多个资料字段**

```
{
  "To_Account":["id1","id2","id3","id4"],
  "TagList":
  [
      "Tag_Profile_IM_Nick",
      "Tag_Profile_IM_AllowType",
      "Tag_Profile_IM_SelfSignature",
      "Tag_Profile_Custom_Test"
  ]
}
```

请求包字段说明

字段	类型	属性	说明
To_Account	Array	必填	需要拉取这些 UserID 的资料。注意：每次拉取的用户数不得超过100，避免因回包数据量太大以致回包失败。
TagList	Array	必填	指定要拉取的资料字段的 Tag，资料字段

应答包体示例

- **读一个用户的一个资料字段**

```
{
  "UserProfileItem":
  [
      {
          "To_Account":"id1",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest1"
              }
          ],
          "ResultCode":0,
          "ResultInfo":""
      }
```

```
    ],
    "ActionStatus":"OK",
    "ErrorCode":0,
    "ErrorInfo":"",
    "ErrorDisplay":""
}
```

- **读一个用户的多个资料字段**

```
{
    "UserProfileItem":
    [
        {
            "To_Account":"id1",
            "ProfileItem":
            [
                {
                    "Tag":"Tag_Profile_IM_Nick",
                    "Value":"NickNameTest1"
                },
                {
                    "Tag":"Tag_Profile_IM_AllowType",
                    "Value":"AllowType_Type_NeedConfirm"
                },
                {
                    "Tag":"Tag_Profile_IM_SelfSignature",
                    "Value":"I'm Test1"
                },
                {
                    "Tag":"Tag_Profile_Custom_Test",
                    "Value":"Custom Data1"
                }
            ],
            "ResultCode":0,
            "ResultInfo":""
        }
    ],
    "ActionStatus":"OK",
    "ErrorCode":0,
    "ErrorInfo":"",
    "ErrorDisplay":""
}
```

- **读多个用户的一个资料字段**


```
{
  "UserProfileItem":
  [
      {
          "To_Account":"id1",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest1"
              }
          ],
          "ResultCode":0,
          "ResultInfo":""
      },
      {
          "To_Account":"id2",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest2"
              }
          ],
          "ResultCode":0,
          "ResultInfo":""
      },
      {
          "To_Account":"id3",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest3"
              }
          ],
          "ResultCode":0,
          "ResultInfo":""
      }
  ],
  "ActionStatus":"OK",
  "ErrorCode":0,
```

```
  "ErrorInfo":"",
  "ErrorDisplay":""
}
```

- **读多个用户的多个资料字段**

```
{
  "UserProfileItem":
  [
      {
          "To_Account":"id1",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest1"
              },
              {
                  "Tag":"Tag_Profile_IM_AllowType",
                  "Value":"AllowType_Type_NeedConfirm"
              },
              {
                  "Tag":"Tag_Profile_IM_SelfSignature",
                  "Value":"I'm Test1"
              },
              {
                  "Tag":"Tag_Profile_Custom_Test",
                  "Value":"Custom Data1"
              }
          ],
          "ResultCode":0,
          "ResultInfo":""
      },
      {
          "To_Account":"id2",
          "ProfileItem":
          [
              {
                  "Tag":"Tag_Profile_IM_Nick",
                  "Value":"NickNameTest2"
              },
              {
                  "Tag":"Tag_Profile_IM_AllowType",
```

```
                    "Value":"AllowType_Type_DenyAny"
                },
                {
                    "Tag":"Tag_Profile_IM_SelfSignature",
                    "Value":"I'm Test2"
                },
                {
                    "Tag":"Tag_Profile_Custom_Test",
                    "Value":"Custom Data2"
                }
            ],
            "ResultCode":0,
            "ResultInfo":""
        },
        {
            "To_Account":"id3",
            "ProfileItem":
            [
                {
                    "Tag":"Tag_Profile_IM_Nick",
                    "Value":"NickNameTest3"
                },
                {
                    "Tag":"Tag_Profile_IM_AllowType",
                    "Value":"AllowType_Type_AllowAny"
                },
                {
                    "Tag":"Tag_Profile_IM_SelfSignature",
                    "Value":"I'm Test3"
                },
                {
                    "Tag":"Tag_Profile_Custom_Test",
                    "Value":"Custom Data3"
                }
            ],
            "ResultCode":0,
            "ResultInfo":""
        },
        {
            "To_Account":"id4",
            "ResultCode":40006,
            "ResultInfo":"Err_Profile_PortraitGet_Read_Custom_Data_Fail"
        }
    ],
```

```
  "Fail_Account":["id4"],
  "ActionStatus":"OK",
  "ErrorCode":0,
  "ErrorInfo":"",
  "ErrorDisplay":""
}
```

应答包字段说明

字段	类型	说明
UserProfileItem	Array	返回的用户资料结构化信息
To_Account	String	返回的用户的 UserID
ProfileItem	Array	返回的用户的资料对象数组，数组中每一个对象都包含了 Tag 和 Value
Tag	String	返回的资料对象的名称：资料字段
Value	uint64_t/string/bytes	拉取的资料对象的值
ResultCode	Integer	To_Account 的处理结果： • 0：表示成功 • 非0：表示失败
ResultInfo	String	To_Account 的错误描述信息，成功时该字段为空
Fail_Account	Array	返回处理失败的用户列表，仅当存在失败用户时才返回该字段
ActionStatus	String	请求处理的结果： • OK：表示处理成功 • FAIL：表示失败
ErrorCode	Integer	错误码： • 0：表示成功 • 非0：表示失败
ErrorInfo	String	详细错误信息
ErrorDisplay	String	详细的客户端展示信息

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	描述
40001	请求参数错误，请根据错误描述检查请求参数
40002	请求参数错误，没有指定 To_Account
40003	请求的用户账号不存在
40004	请求需要 App 管理员权限
40006	服务器内部错误，请稍后重试
40007	没有资料字段的读权限
40009	资料字段的 Tag 不存在

接口调试工具

通过 REST API 在线调试 工具调试本接口。

房间管理
创建房间

最近更新时间：2025-05-30 16:34:02

功能说明

App 管理员可以通过该接口创建房间。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/create_room?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/create_room	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

```
{
  "RoomInfo":{
              "RoomId":"live-room",
              "RoomType":"Live",
              "RoomName":"live name",
              "Owner_Account":"administrator",
              "MaxMemberCount":400,
              "CoverURL":"cover url",
              "IsMessageDisabled":true,
              "Category":[1, 2, 3],
              "IsPublicVisible": true,
              "IsSeatEnabled": true,
              "TakeSeatMode":"ApplyToTake",
              "MaxSeatCount":8,
              "ActivityStatus":1,
              "IsUnlimitedRoomEnabled":true
              }
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID，最长48个字节
RoomName	String	选填	房间名称，默认为房间 ID，最长100个字节
RoomType	String	必填	房间类型：Live（直播房间）
Owner_Account	String	选填	房主 ID（需是 已导入 的账号），默认为接口调用者的用户 ID
MaxMemberCount	Integer	选填	最大房间成员数量，缺省时的默认值：付费套餐包上限，例如体验版是20, 如果升级套餐包，需按照修改房间资料修改这个字段
CoverURL	String	选填	房间封面，最大200字节
IsMessageDisabled	Bool	选填	全体禁音，默认 false

Category	Array	选填	房间类型标志, 业务可以用来作为房间类型标识，例如游戏、音乐等，最多支持三个
IsPublicVisible	Bool	选填	房间公开标识
IsSeatEnabled	Bool	选填	是否支持麦位, 默认不支持
TakeSeatMode	String	选填	当使用麦位能力时候，需要填写麦位模式：FreeToTake（自由上麦），ApplyToTake（申请上麦）
MaxSeatCount	Integer	选填	最大麦位数，当使用麦位能力时候，默认为套餐包上限
ActivityStatus	Integer	选填	直播间活跃状态：用户自定义标记
IsUnlimitedRoomEnabled	Bool	选填	开启房间混流，支持高并发场景，默认为 false。当设置为 true 时候对应于 sdk 侧 TUILiveKit 视频房

注意：

- 用户需要在直播结束以后，手动调用 解散房间 接口，结束本场会议。
- Live 房间自动解散机制 。

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720",
  "RoomId": "live-room"
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

ErrorInfo	String	错误信息
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
RoomId	String	房间 ID

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	请参数非法，请根据错误描述检查请求是否正确
100003	房间 ID 已存在，请选择其他房间 ID
100007	无付费信息，需在控制台购买套餐包
100010	房间 ID 已被使用，并且操作者为房主，可以直接使用
100011	房间 ID 已被 IM 占用，可以换一个房间 ID 使用，或者先通过 IM 接口解散该群
100012	创建房间超过频率超限，同一房间 ID 1秒内只能创建一次

可能触发的回调

- 创建房间之后回调

更新房间

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口更新房间信息。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/update_room_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
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/update_room_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

```
{
    "RoomInfo" : {
      "RoomId":"live-room",
      "RoomName" : "rname-123",
      "MaxMemberCount" : 300,
      "IsMessageDisabled" : false,
      "CoverURL": "https://xxxx.png",
      "Category": [1,2,3],
      "ActivityStatus":1,
      "IsPublicVisible": false,
      "TakeSeatMode": "ApplyToTake",
      "CustomInfo": "custom123"
  }
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
RoomName	String	选填	房间名字
MaxMemberCount	Integer	选填	房间人数上限
IsMessageDisabled	Bool	选填	房间全员禁言开关，管理员和房主除外
CoverURL	String	选填	房间封面
Category	Array	选填	房间分类标签，数组长度上限为3
ActivityStatus	Integer	选填	房间自定义状态
IsPublicVisible	Bool	选填	房间开放开关，主要用于获取房间列表使用
TakeSeatMode	String	选填	麦位模式，自由上麦 FreeToTake，申请上麦 ApplyToTake
CustomInfo	String	选填	自定义字段

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-0-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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了
100006	没有权限解散，例如操作者不是房主也不是root管理员或者解散的不是live房间等

版权所有：腾讯云计算（北京）有限责任公司

获取房间信息

最近更新时间：2025-05-20 17:56:32

功能说明

App 管理员可以通过该接口获取房间信息。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_room_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
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_room_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

```
{
     "RoomId":"live-room"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720",
  "Response":  {// 响应体
    "RoomInfo" : {
      "RoomId" : "rid-123",
      "RoomName" : "rname-123",
      "RoomType" : "Live",
      "Owner_Account" : 144115233775727695,
      "CreateTime": 1693271354,
      "IsSeatEnabled" : true,
       "TakeSeatMode":"ApplyToTake",
       "MaxSeatCount" : 8,
       "MaxMemberCount" : 300,
       "IsMessageDisabled" : false,
       "CoverURL": "https://xxxx.png",
       "Category": [1,2,3],
       "ActivityStatus":1,
       "ViewCount":10,
       "IsPublicVisible": true
    }
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
RoomInfo	Object	房间详情
RoomId	String	房间 ID
RoomName	String	房间名字
RoomType	String	房间类型：Live（直播房间）
Owner_Account	String	房间群主 ID
IsSeatEnabled	Bool	麦位能力开关
TakeSeatMode	String	麦位模式，自由上麦 FreeToTake，申请上麦 ApplyToTake
MaxSeatCount	Integer	麦位数量，受套餐包限制
MaxMemberCount	Integer	房间最大人数上限
IsMessageDisabled	String	群内全员禁言开关
CoverURL	String	房间封面
Category	Array	直播间分类标签，数组大小上限为3
ActivityStatus	Integer	直播间活跃状态：用户自定义标记
IsPublicVisible	Bool	房间是否公开可见, 用于获取直播间列表使用
CreateTime	Integer	房间创建时间
ViewCount	Integer	用户进房总次数

错误码说明

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

版权所有：腾讯云计算（北京）有限责任公司

获取直播列表

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口获取直播列表。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_room_list?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_room_list	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

10次/秒。

请求包示例

基础形式

```
{
    "Next":"",
    "Count":20
}
```

注意: 列表是按照ViewCount字段降序排序, 分页拉取

请求包字段说明

字段	类型	属性	说明
Next	String	必填	分页请求的分页参数，第一次为空字符串，后续使用响应包体的 Next，当响应包体的 Next 为空字符串时候，表示拉取完毕
Count	Integer	必填	单次拉取的房间数量，默认为20，上限为20

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720",
  "Response" : {
    "Next":"1693271355-123123123123",
    "RoomList":[
      {
        "RoomName":"name",
        "RoomId":"room_id0",
        "Owner_Account":"bren",
        "CoverURL":"cover_url",
        "Category":[1,2,3],
        "CreateTime":1693271354,
        "ActivityStatus":1,
        "ViewCount":1000
      },
      ...
      {
        "RoomName":"name",
```

```
            "RoomId":"room_id19",
            "Owner_Account":"bren",
            "CoverURL":"cover_url",
            "Category":[1,2,3],
            "CreateTime":1693271354,
            "ActivityStatus":1,
            "ViewCount":900
        }
    ]
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
RoomList	Array	房间 ID 列表，默认按照 view count 降序排序
RoomName	String	房间名字
RoomId	String	房间 ID
Owner_Account	String	房间群主 ID
CoverURL	String	房间封面
Category	Array	直播间分类标签，数组大小上限为3
CreateTime	Integer	房间创建时间
ActivityStatus	Integer	直播间活跃状态：用户自定义标记
ViewCount	Integer	用户进房总次数
Next	String	当房间列表拉完后，返回空字符串，否则返回下次请求的 Next

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确

解散房间

最近更新时间：2025-06-17 16:46:52

功能说明

App 管理员可以通过该接口解散房间。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/destroy_room?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/destroy_room	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

```
{
    "RoomId":"live-room"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-0-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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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

房间自动解散机制

Live 房间在以下三种情况会触发房间自动解散逻辑, 支持 sdkappid 维度的配置，开启条件和时间的调整配置请联系商务或者 提交工单 进行变更。

1. 当有用户进入过房间之后，房间空无一人并且持续一段时间内(默认30分钟)没人进房，就会触发自动解散机制，配置 机器人 的房间不会自动解散。2025年05月20日之前的 Live 客户维持原来的6小时解散机制，该时间支持调整，配置区间为[10分钟，24小时]，**该机制不支持关闭**。

2. 房主调用接口离开房间并且持续一段时间内(默认10分钟)没有重新进房，就会触发自动解散机制，如果房主是 机器人，则不会触发该机制，该机制**默认关闭**，该时间支持调整，配置区间为[2分钟，12小时]。
3. 房主超过心跳保活时间掉线后，并且持续一段时间内(默认10分钟)没有恢复心跳进房，就会触发自动解散机制，如果房主是 机器人，则不会触发该机制，该机制**默认关闭**，该时间支持调整，配置区间为[2分钟，12小时]。

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了
100006	没有权限解散，例如操作者不是房主也不是 root 管理员或者解散的不是 live 房间等

获取房间自定义信息

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口设置房间自定义信息。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_room_metadata?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_room_metadata	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

```
{
  "RoomId": "live-room",
  "Keys":["key1", "key2"] //如果为空数组表示获取该房间的所有的自定义信息，否则就是获取指定Key列表的自定义信息
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Keys	Array	必填	指定的 Key 列表，如果为空，则表示获取房间内所有的自定义信息

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-O-Seq-52720",
  "Response": {
        "Metadata": [
            {
                "Key": "key1",
                "Value": "value1"
            },
            {
                "Key": "key2",
                "Value": "value2"
            }
        ]
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
Response	Object	房间自定义信息
Metadata	Array	返回的有效的房间自定义信息

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	请参数非法，请根据错误描述检查请求是否正确
100003	房间 ID 已存在，请选择其他房间 ID
100006	无权限操作房间自定义信息
100007	无付费信息，需在控制台购买套餐包
100500	房间自定义信息key数量超过上限10
100501	房间自定义信息单个key对应的value字节数超过上限2KB
100502	房间自定义信息所有key对应的value字节数总和超过上限1KB
100503	删除自定义信息时候指定的key没有一个有效的

版权所有：腾讯云计算（北京）有限责任公司

设置房间自定义信息

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口设置房间自定义信息。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/set_room_metadata?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/set_room_metadata	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

```
{
  "RoomId": "live-room",
  "Metadata": [ //key数量不超过10,  key 的长度不超过50字节，单个value 字节数上限为 2K，所有value总字节数上限为16K
                {
                    "Key":"key1",  //当key存在时候则是更新，否则就是新增
                    "Value":"value1"
                },
                {
                    "Key":"key2",
                    "Value":"value2"
                }
            ]
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Metadata	Array	必填	房间自定义信息列表，key 数量不超过10，key 的长度不超过50字节，单个 value 字节数上限为2K，所有 value 总字节数上限为16K

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

ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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	请参数非法，请根据错误描述检查请求是否正确
100003	房间 ID 已存在，请选择其他房间 ID
100006	无权限操作房间自定义信息
100007	无付费信息，需在控制台购买套餐包
100500	房间自定义信息 key 数量超过上限10
100501	房间自定义信息单个 key 对应的 value 字节数超过上限2KB
100502	房间自定义信息所有 key 对应的 value 字节数总和超过上限16KB
100503	删除自定义信息时候指定的 key 没有一个有效的

版权所有：腾讯云计算（北京）有限责任公司

删除房间自定义信息

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口删除房间内指定的间自定义信息。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/del_room_metadata?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/del_room_metadata	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

```
{
  "RoomId": "live-room",
  "Keys": ["key1", "key2"]
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Keys	Array	必填	房间自定义信息指定删除的 Key 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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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。

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100003	房间 ID 已存在，请选择其他房间 ID
100006	无权限操作房间自定义信息
100007	无付费信息，需在控制台购买套餐包
100500	房间自定义信息 key 数量超过上限10
100501	房间自定义信息单个 key 对应的 value 字节数超过上限2KB
100502	房间自定义信息所有 key 对应的 value 字节数总和超过上限16KB
100503	删除自定义信息时候指定的key没有一个有效的

版权所有：腾讯云计算（北京）有限责任公司

房间成员管理
成员禁言

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口针成员进行房间内禁言和取消禁言。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/mute_member?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/mute_member	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

```
{
      "RoomId": "test",
      "MemberList_Account": [
             "user1",
             "user2"
             ],
     "MuteTime":3600
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
MemberList_Account	Array	必填	禁言的成员列表
MuteTime	integer	必填	禁言时间，如果是0，表示取消禁言

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-0-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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息

版权所有：腾讯云计算（北京）有限责任公司

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（TUILiveKit） 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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

版权所有：腾讯云计算（北京）有限责任公司

获取禁言列表

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口针对成员进行禁言和取消禁言。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_muted_member_list?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_muted_member_list	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

```
{
    "RoomId": "test"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-9a9fe01d96164827b49dacb6b3cdadba-0-Seq-126318",
    "Response": {
        "MutedAccountList": [
            {
                "Member_Account": "user1",
                "MutedUntil": 1719293107
            },
            {
                "Member_Account": "brennanli",
                "MutedUntil": 1719293107
            }
        ]
    }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

版权所有：腾讯云计算（北京）有限责任公司

ErrorInfo	String	错误信息
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
MutedAccountList	Array	禁言列表
Member_Account	String	禁言用户 ID
MutedUntil	Integer	禁言截止时间

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

成员封禁

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口封禁成员，被封禁的用户在封禁有效期内无法再进房间。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/ban_member?
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/ban_member	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

```
{
    "RoomId": "test",
     "MemberList_Account": [
            "user1",
            "user2"
            ],
    "Reason":"",
    "Duration":3600
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
MemberList_Account	Array	必填	被封禁的成员列表，单次最多20个
Reason	String	选填	封禁原因
Duration	Integer	必填	封禁时间

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

版权所有：腾讯云计算（北京）有限责任公司

ErrorInfo	String	错误信息
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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了
100006	没有权限转移房主
100016	新的房主被封禁，请先解禁

解除成员封禁

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口解除成员封禁。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/unban_member?
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/unban_member	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

```
{
    "RoomId": "test",
     "MemberList_Account": [
            "user1",
            "user2"
            ]
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
MemberList_Account	Array	必填	解除封禁的成员列表，单次最多20个

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-0-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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

获取封禁列表

最近更新时间：2025-05-20 17:56:32

功能说明

App 管理员可以通过该接口获取封禁成员列表。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_banned_member_list?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_banned_member_list	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

```
{
    "RoomId": "test",
    "Offset":0,
    "Limit":10
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Offset	Integer	选填	偏移量，初次拉取从0开始，后续请求以响应包 NextOffset 为准
Limit	Integer	选填	单次分页拉取的数量，最大为100

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-138c94122acc4ca485c8c7bf270686ac-0-Seq-135511",
    "Response": { //响应体
        "BannedAccountList": [
            {
                "Member_Account": "user2",
                "BannedUntil": 1719300627
            },
            {
                "Member_Account": "user1",
                "BannedUntil": 1719300627
            }
        ],
        "NextOffset": 0 // 分页标识
    }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
BannedAccountList	Array	封禁成员列表
Member_Account	String	封禁成员账号 ID
BannedUntil	Integer	封禁截止时间

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

修改管理员

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口进行设置和取消房间管理员。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/modify_admin?
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/modify_admin	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

```
{
    "RoomId": "test",
     "Admin_Account": [
            "user1",
            "user2"
            ],
    "OpType":"add"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Admin_Account	Array	必填	修改的管理员列表，最大支持 5 个管理员
OpType	String	必填	操作类型，add 表示添加管理员，delete 表示删除管理员

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了
100020	房间管理员超过上限, 最大上限为5

获取管理员列表

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口获取房间管理员列表。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_admin_list?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_admin_list	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

```
{
    "RoomId": "test"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID

应答包体示例

基础形式

```
{
  "ActionStatus": "OK",
  "ErrorInfo": "",
  "ErrorCode": 0,
  "RequestId": "Id-8c9858f01e954611ae2d4c1b1ed7d583-0-Seq-52720",
  "Response": { // 响应体
        "Admin_Account": [
            "user2",
            "user1"
        ]
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
Admin_Account	Array	管理员列表

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了

获取房间成员列表

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口获取房间成员列表。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/get_member_list?sdkappid=88888888&identifier=admin&usersig=xxx&random=99999999&contenttype=json
```

注意：

- 在线成员列表总体更新粒度为40s。
- 当房间超过1000人时，接口仅返回1000人。

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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/get_member_list	请求接口
sdkappid	创建应用时分配的 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

```
{
    "RoomId":"room_id"
    "Next": ""    // 首次为空字符串，下次请求为之前响应里的Next，直到为空字符串表示已经获取完毕
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Next	String	必填	分页获取参数

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "requestId-123-456",
  "Response" : {
    "Next": "",  //返回空表示所有成员获取完毕
    "MemberCount": 2, //房间总人数
    "MemberList":[
      {
        "Member_Account" : "brennan1"
      },
      {
        "Member_Account" : "tandy"
      },
   ]
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
MemberList	Array	房间成员列表
Next	String	下次请求参数里的Next值
MemberCount	Integer	房间总人数

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（TUILiveKit） 文档。
本 API 私有错误码如下：

错误码	含义说明
100001	系统错误，重试
100004	房间不存在，原因可能是房间没被创建过，或者已经被解散了
100006	表示此房间不是 live 房间，无权限使用 live 接口查询

版权所有：腾讯云计算（北京）有限责任公司

连麦管理
获取麦位列表

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口获取麦位列表。

接口调用说明

请求 URL 示例

```
https://console.tim.qq.com/v4/room_engine_http_mic/get_seat_list?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/room_engine_http_mic/get_seat_list	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

```
{
```

```
  "RoomId":"room-test"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-7c1680be52734bdc8d6de398ab9505e7-O-Seq-57717",
  "Response": {
      "SeatList": [
          {
              "Index": 0,
              "Member_Account": "user2",
              "IsTakenDisabled": false,
              "IsVideoDisabled": false,
              "IsAudioDisabled": false
          },
          {
              "Index": 1,
              "IsTakenDisabled": false,
              "IsVideoDisabled": false,
              "IsAudioDisabled": false
          }
      ]
  }
}
```

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

ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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
SeatList	Array	麦位列表
Index	Integer	麦位号
Member_Account	String	麦位上的用户 ID，空表示麦上没有用户
IsTakenDisabled	Bool	麦位锁定
IsVideoDisabled	Bool	禁止麦位视频流
IsAudioDisabled	Bool	禁止麦位音频流

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，或者曾经存在过，但是目前已经被解散

用户上麦

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口让指定成员上麦。

接口调用说明

请求 URL 示例

```
https://console.tim.qq.com/v4/room_engine_http_mic/pick_user_on_seat?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/room_engine_http_mic/pick_user_on_seat	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

```
{
  "RoomId":"room-test",
  "Member_Account":"user2",
```

```
  "Index":1
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Index	Integer	必填	麦位号
Member_Account	String	必填	上麦用户 ID，需要为房间成员

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-7c1680be52734bdc8d6de398ab9505e7-O-Seq-57717"
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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。

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，或者曾经存在过，但是目前已经被解散
100005	上麦用户非房间成员
100200	麦位已锁定，可以尝试换一个麦位
100202	已经处于排麦状态
100203	已经处于麦上状态
100205	麦位已满
100210	已有用户在麦位上
100211	该房间不支持连麦

可能触发的回调

- 麦位列表变更之后回调

用户下麦

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口让指定成员下麦。

接口调用说明

请求 URL 示例

```
https://console.tim.qq.com/v4/room_engine_http_mic/kick_user_off_seat?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/room_engine_http_mic/kick_user_off_seat	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

```
{
  "RoomId":"room-test",
  "Member_Account":"user2",
```

```
  "Data":"time over"
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Member_Account	String	必填	麦上用户 ID
Data	String	必填	自定义信息，该信息踢人下麦下行通知时下发

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-7c1680be52734bdc8d6de398ab9505e7-O-Seq-57717"
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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。

版权所有：腾讯云计算（北京）有限责任公司

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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，或者曾经存在过，但是目前已经被解散

可能触发的回调

- 麦位列表变更之后回调

锁定麦位

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口锁定麦位，麦位锁定后，用户无法再上此麦位。锁定麦位时，如果麦上有人，会先将用户下麦，再锁定。

接口调用说明

请求 URL 示例

```
https://console.tim.qq.com/v4/room_engine_http_mic/lock_seat?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/room_engine_http_mic/lock_seat	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

```
{
```

```
  "RoomId":"room-test",
  "Index":2,
  "ForbidTaken":true,
  "ForbidVideo":true,
  "ForbidAudio":true
}
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Index	Integer	必填	麦位号
ForbidTaken	Bool	选填	锁定麦位
ForbidVideo	Bool	选填	禁止视频流
ForbidAudio	Bool	选填	禁止音频流

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-7c1680be52734bdc8d6de398ab9505e7-O-Seq-57717"
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

ErrorInfo	String	错误信息
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	请参数非法，请根据错误描述检查请求是否正确
100004	房间不存在，或者曾经存在过，但是目前已经被解散
100211	该房间不支持连麦

可能触发的回调

- 麦位列表变更之后回调

PK 相关
发起 PK

最近更新时间：2025-05-20 17:56:32

功能说明

App 管理员可以通过该接口发起 PK。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_PK/create_battle?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_pk/create_battle	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。

版权所有：腾讯云计算（北京）有限责任公司

请求包示例

基础形式

```
{
    "FromRoomId":"pk-1",   //pk的主叫房间id
    "ToRoomIdList":["pk-2","pk-3"], //pk的被叫房间id列表
    "Duration":60000, //pk时长
    "ExtensionInfo":"extension pk"
}
```

请求包字段说明

字段	类型	属性	说明
FromRoomId	String	必填	PK 主叫房间 ID
ToRoomIdList	String	必填	PK 被叫房间 ID 列表
Duration	Integer	必填	PK 时长
ExtensionInfo	String	选填	PK 信息扩展字段

> **注意：**
> 发起 PK 时候，业务侧需要确认端上参与 PK 的房间处于连线中。发起 PK 的时候会通知参与 PK 的房间房主。

应答包体示例

基础形式

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "requestId-123-456",
  "Response":{
    "BattleId": "battleid1",
    "Result": [
          {
              "RoomId": "pk-2",
              "ResultCode": 0,
              // 0 表示发起成功，
```

```
                // 3 表示已经处于其他PK中,
                // 5 系统错误
                "ResultMessage": "",
            },
            {
                "RoomId": "pk-3",
                "ResultCode": 0,
                "ResultMessage": "",
            }
        ],
    }
}
```

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
BattleId	String	此次 PK 的 ID
Result	Array	被叫的 PK 结果
RoomId	String	房间 ID
ResultCode	Integer	结果，0表示成功， 3表示正处于 pk 中，5表示系统错误
ResultMessage	String	结果信息

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	请参数非法，请根据错误描述检查请求是否正确
100003	房间 ID 已存在，请选择其他房间 ID
100412	被叫房间都处于 PK 中，请稍后再试
100415	主叫方处于 PK 中，请稍后再试

版权所有：腾讯云计算（北京）有限责任公司

修改 PK 分数

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口修改 pk 里各房间分数。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_pk/set_battle_score?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。

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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_pk/set_battle_score	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

```
{

    "BattleId":"HuYM036AMk2JJeSUgnv5H0KDJpc+HXfqTAtGuYWzRPs=",
    "OpType":1, // 1表示累加，2表示重置
    "ScoreList":[
                {
                    "RoomId":"pk-live1",
                    "Score":100
                },
                {
                    "RoomId":"pk-live2",
                    "Score":100
                }
            ]
}
```

请求包字段说明

字段	类型	属性	说明
BattleId	String	必填	pk 场次对应的 ID，可通过获取 pk 详情获取到
OpType	String	必填	设置类型，1为累加，2为覆盖设置
ScoreList	Array	必填	设置分数列表，如果请求里的房间不处于该 pk 中，则该列表忽略此房间
RoomId	String	必填	房间 ID
Score	Integer	必填	房间分数

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-e7625183568d4058bd48c9924d4b1ed7-O-Seq-125382",
    "Response": {
        "Result": [
```

```
            {
                "RoomId": "pk-3",
                "ResultCode": 0
            },
            {
                "RoomId": "pk-4",
                "ResultCode": 0
            }
        ]
    }
}
```

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
Result	Array	操作结果
RoomId	String	房间 id
ResultCode	Integer	0表示成功，1表示房间不存在

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	请参数非法，请根据错误描述检查请求是否正确
100006	当前 pk 尚未开始或者已经结束，无权再设置分数

版权所有：腾讯云计算（北京）有限责任公司

获取 PK 信息

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口获取 pk 详情。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_pk/get_battle_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
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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_pk/get_battle_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

```
{
    "BattleId":"HuYM036AMk2JJeSUgnv5H0KDJpc+HXfqTAtGuYWzRPs=", //业务后台创建pk时候可以保存好该BattleId。
    "RoomId":"room_pk1" //当业务侧不知道具体的BattleId时候，可以填写RoomId，这样就可以获取该RoomId正在进行的BattleId信息
}
```

请求包字段说明

字段	类型	属性	说明
BattleId	String	必填	pk 场次的 ID
RoomId	String	选填	房间 ID，当不知道 BattleId 时候，可以填写 RoomId，响应包返回该房间正在进行的 PK 详情

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-e1fffbc751dc44bc95ec960d1fdfbefe-0-Seq-126652",
    "Response": {
        "BattleId": "HuYM036AMk2JJeSUgnv5H0KDJpc+HXfqTAtGuYWzRPs=",
        "Duration": 120000,
        "StartTime": 1739508625,
        "EndTime": 1739508746,
        "Status": 3,
        "FromRoomInfo": {
            "RoomId": "pk-3",
            "Score": 100,
            "Owner_Account": "brennanli1"
        },
        "ToRoomList": [
            {
                "RoomId": "pk-4",
                "Score": 100,
                "Owner_Account": "brennanli2"
```

```
                }
            ]
        }
}
```

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
BattleId	String	pk ID
Duration	Integer	pk 时长
StartTime	Integer	pk 正式开始时间
EndTime	Integer	pk 正式结束时间
Status	Integer	pk 状态，1表示未开始，2表示进行中，3表示已结束，4表示 pk 未开始就结束
FromRoomInfo	Object	pk 主叫房间信息
ToRoomList	Array	pk 被叫房间信息
RoomId	String	房间 ID
Score	Integer	pk 中的房间分数
Owner_Account	String	pk 中的房间对应的房主账号 ID

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	请参数非法，请根据错误描述检查请求是否正确
100411	当前pk不存在，或者结束时间过长信息已被清理

版权所有：腾讯云计算（北京）有限责任公司

云端搜索
房间成员搜索

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口在房间里**模糊搜索**成员。

请求 URL 示例

```
https://xxxxxx/v4/live_engine_http_srv/search_room_member?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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_engine_http_srv/search_room_member	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

2次/秒。

请求包示例

基础形式

```
 {
        "RoomId":"search_test", //房间id
        "Keyword":"brennanli", //关键字搜索，用户的账号昵称或者账号id，账号昵称支持模糊搜索，账号id支持精确匹配
        "Next":"" //用于分页查询的参数，用于指定下一页数据的起始位置，首次请求时，Next 字段为空，每次请求后，服务器返回响应中的 Next 字段值。客户端需将该值作为下一次请求的 Next 参数，直到 Next 为空才停止
 }
```

请求包字段说明

字段	类型	属性	说明
RoomId	String	必填	房间 ID
Keyword	String	必填	关键字匹配，可填写账号昵称或者账号 ID，其中账号昵称支持模糊搜索
Next	String	必填	用于分页查询的参数，用于指定下一页数据的起始位置

> **注意：**
> 该接口能力支持版本为：体验版 + LiveKit 大规模直播版，当前需要提单联系平台手动打开。

应答包体示例

基础形式

```
{
    "ErrorCode": 0,
    "ErrorInfo": "",
    "ActionStatus": "OK",
    "RequestId": "Id-1ea25224fab7403f939986c2f7266ce2-O-Seq-1645192760",
    "Response": {
        "Next": "",
        "MemberList": [
            {
                "Member_Account": "brennanli1",
                "NickName":
"brennanliqbrennanliqbrennanliqbrennanliqbrennanliqbrennanliq"
            },
```

```
            {
                "Member_Account": "brennanli",
                "NickName": "brennanli_nick"
            }
        ]
    }
}
```

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
Next	String	分页列表的参数，为空时候表示结束
MemberList	Array	搜索到的成员信息列表
Member_Account	String	成员的账号 ID
NickName	String	成员的账号昵称

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	请参数非法，请根据错误描述检查请求是否正确

100007	当前付费套餐包不支持该能力

版权所有：腾讯云计算（北京）有限责任公司

第三方回调
第三方回调简介

最近更新时间：2025-04-29 16:54:12

为了方便您控制 App 的功能形态，LiveKit 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。

回调协议

第三方回调基于 HTTP/HTTPS 协议，App 后台需要向 live平台 提供回调 URL，即时通信 IM 使用 POST 请求的方式来向 App 后台发起回调请求。即时通信 IM 在发起回调时，会在 App 提供的 URL 之后增加如下几个参数：

参数	含义
SdkAppid	App 在即时通信 IM 分配的应用标识
CallbackCommand	回调命令字
contenttype	可选，通常值为 JSON
ClientIP	客户端 IP 地址
OptPlatform	客户端平台，对应不同的平台类型，可能的取值有：RESTAPI（使用 REST API 发送请求）、Web（使用 Web SDK 发送请求）、Android、iOS、Windows、Mac、iPad、Unknown（使用未知类型的设备发送请求）

安全问题

安全性问题：

1. HTTP 是明文传输，数据的保密性无法保证，建议使用 HTTPS。
2. 无法验证回调请求是否真正来自于 Live。

为了解决请求来源的安全性问题，我们为您提供**回调应用层签名认证**，启用步骤如下：

1. 首先使用 设置回调配置 设置回调。
2. 在 设置回调鉴权 Token 中，开启鉴权，并配置鉴权 Token，配置完成后，回调请求 URL 参数会增加签名 Sign 以及签名时间戳 RequestTime，签名算法：**Sign=sha256(TokenRequestTime)** 。

3. 在 App 后台支持针对回调请求进行鉴权，通过本地鉴权 Token 以及回调 URL 参数中的签名时间戳 RequestTime 计算 sha256，比较签名是否匹配。并对 RequestTime 校验时效性，RequestTime 和当前时间相差超过1分钟即可认为是无效请求，以防止重放攻击。

签名算法示例：

```
Token=xxxxyyyy
RequestTime=1669872112
Sign=sha256(xxxxyyyy1669872112)=17773bc39a671d7b9aa835458704d2a6db81360a59
40292b587d6d760d484061
```

回调

```
URL=URL&Sign=17773bc39a671d7b9aa835458704d2a6db81360a5940292b587d6d760d484
061&RequestTime=1669872112
```

回调不通的常见原因

如果遇到回调不通的情况，App 先依照如下清单排查一下设置的回调服务是否存在问题。

回调不通的现象	可能存在的原因
回调 URL 访问超时	1. Live无法完成 DNS 解析，请确认该域名是否在公网生效。（例如，回调 HOST 为 http://notexist.com，该域名不存在，Live无法完成 DNS 解析。） 2. Live无法访问到回调 URL 中配置的 IP，请确认该 IP 是否公网可达。（例如，回调 HOST 为 http://10.0.0.1，该域名为 App 内网 IP，Live无法访问到该 IP。） 3. App 回调服务防火墙策略限制，请检查防火墙配置。（例如，App 回调服务器拒绝了所有到达 80 端口的请求。）
回调服务拒绝访问	Live 可以访问到 HOST，但链接建立失败，请确认 WebServer 已经正确启动。（例如：App 回调服务器的 WebServer 并未启动，或者端口配置错误。）
回调服务 HTTP 返回码非200	回调请求成功，但应答报文中的 HTTP 返回码非200。
回调应答包体解析失败	回调请求包体非 JSON 格式。

回调命令列表

最近更新时间：2025-06-20 15:11:32

房间相关

回调类型	回调命令字
创建房间之后回调	Live.CallbackAfterCreateRoom
解散房间之后回调	Live.CallbackAfterDestroyRoom
更新房间信息之后回调	Live.CallbackAfterUpdateRoomInfo
设置房间自定义信息之后回调	Live.CallbackAfterSetMetadata
删除房间自定义信息之后回调	Live.CallbackAfterDelMetadata
创建房间达到70%预警回调	Live.CallbackAfterCreateRoomReachingThreshold

用户相关

回调类型	回调命令字
房间成员状态变更回调	Live.CallbackAfterMemberStateChanged

连麦相关

回调类型	回调命令字
麦位列表变更之后回调	Mic.CallbackAfterSeatInfoChanged

PK 相关

回调类型	回调命令字
创建 PK 之后回调	Live.CallbackAfterCreateBattle
PK 开启之后回调	Live.CallbackAfterStartBattle
PK 结束之后回调	Live.CallbackAfterEndBattle

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：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口创建回调。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_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，更多参数详情请参见 REST API 简介 。

参数	说明
xxxxxxx	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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_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**
  创建房间

```
{
}
```

请求包字段说明

应答包体示例

- **基础形式**

```
{
  "ErrorCode": 0,
  "ErrorInfo": "",
  "ActionStatus": "OK",
  "RequestId": "Id-1cc8828fd3d84795ac866ced43b15b5c-O-Seq-61309",
  "Response": {
      "Url": "http://www.example.com/callback",
      "CallbackCommandList": [
        "Live.CallbackAfterCreateRoom",
        "Live.CallbackAfterDestroyRoom"
      ]
  }
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口设置回调配置。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_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，更多参数详情请参见 REST API 简介 。

参数	说明
v4/live_config/set_callback	请求接口
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` • 美国：`adminapiusa.im.qcloud.com`
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**

  创建房间

```
{
  "Url":"http://www.example.com/callback",
  "CallbackCommandList":[
      "Live.CallbackAfterCreateRoom",
      "Live.CallbackAfterDestroyRoom",
      "Live.CallbackAfterMemberStateChanged"
  ]
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

ErrorInfo	String	错误信息
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。
100300	回调配置已存在，可以使用更新回调配置接口进行更新

更新回调配置

最近更新时间：2025-04-03 16:12:03

功能说明

App 管理员可以通过该接口创建回调。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_config/update_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` • 美国：`adminapiusa.im.qcloud.com`
v4/live_config/update_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**

  创建房间

```
{
  "Url":"http://www.example.com/callback",
  "CallbackCommandList":[
    "Live.CallbackAfterCreateRoom",
    "Live.CallbackAfterDestroyRoom"
  ]
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败

ErrorInfo	String	错误信息
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。
100301	回调配置不存在，可以时间创建回调配置接口进行创建

删除回调配置

最近更新时间：2025-04-03 16:12:04

功能说明

App 管理员可以通过该接口删除回调配置。

接口调用说明

请求 URL 示例

```
https://xxxxxx/v4/live_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，更多参数详情请参见 REST API 简介 。

参数	说明
v4/live_config/delete_callback	请求接口
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` • 美国：`adminapiusa.im.qcloud.com`
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**

  创建房间

```
{
}
```

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
ErrorCode	Integer	错误码，0表示成功，非0表示失败
ErrorInfo	String	错误信息
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。

房间相关
创建房间之后回调

最近更新时间：2025-06-20 15:11:32

功能说明

App 后台可以通过该回调实时查看用户创建房间的信息，包括：App 后台有房间创建成功通知时，可以据此进行数据同步等操作。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端创建房间成功
- App 管理员通过 REST API 创建房间成功

回调发生时机

房间创建成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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.c	回调 URL

om	
SdkAppid	创建应用时在即时通信 IM 控制台分配的 SDKAppID
CallbackCommand	固定为 Live.CallbackAfterCreateRoom
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":"Live.CallbackAfterCreateRoom",
    "Operator_Account":"admin",
    "RoomInfo":{
        "RoomId":"tandy-test-rest",
        "RoomName":"tandy-test-rest",
        "RoomType":"Live",
        "Owner_Account":"user3",
        "MaxMemberCount":300,
        "IsMessageDisabled":true,
        "CustomInfo":"custom123",
        "IsSeatEnabled":true,
        "TakeSeatMode":"ApplyToTake",
        "MaxSeatCount":16,
        "CreateTime":1703589922,
        "IsPublicVisible":true,
        "CoverURL":"cover url",
        "Category":[1,2]
    }
    "EventTime":1703589922780
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令

Operator_Account	String	发起创建群组请求的操作者 UserID
RoomId	String	房间 ID
RoomName	String	房间名称
RoomType	String	房间类型：Meeting（会议房间）
Owner_Account	String	房主 ID
MaxMemberCount	Integer	最大房间成员数量
IsMessageDisabled	Bool	禁止所有成员发文本消息
CustomInfo	String	自定义字段
IsSeatEnabled	Bool	是否支持麦位
MaxSeatCount	Integer	最大麦位数
TakeSeatMode	String	麦位模式：None（关闭），FreeToTake（自由上麦），ApplyToTake（申请上麦）
CreateTime	Integer	预定会议开始时间
IsPublicVisible	Bool	房间是否公开
CoverURL	String	房间封面 url
Category	Array	房间类型标识，整数数组类型
EventTime	Integer	事件触发的毫秒级别时间戳

版权所有：腾讯云计算（北京）有限责任公司

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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介
- REST API：创建房间

解散房间之后回调

最近更新时间：2025-04-29 16:54:12

功能说明

App 后台可以通过该回调实时查看房间的解散动态，包括：对房间的解散实时记录（例如记录日志，或者同步到其他系统）。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端解散房间成功。
- App 管理员通过 REST API 解散房间成功。

回调发生时机

房间销毁成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 Live.CallbackAfterDestroyRoom
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":"Live.CallbackAfterDestroyRoom",
    "Operator_Account":"admin",
    "RoomId":"tandy-test-rest",
    "EventType":"DestroyByUser", //"DestroyByUser", "DestroyBySystem" 分为用户主动解散和房间无人后系统自动解散两种
    "EventTime":1703589922780,
     "RoomInfo": {
        "RoomName": "live name",
        "RoomType": "Live",
        "Owner_Account": 144115216631667826,
        "IsSeatEnabled": true,
        "TakeSeatMode": "FreeToTake",
        "MaxMemberCount": 400,
        "MaxSeatCount": 4,
        "Category": [1, 2, 3],
        "CustomInfo": "",
        "IsMessageDisabled": false,
        "CoverURL": "cover url",
        "ActivityStatus": 0,
        "IsPublicVisible": false,
        "ViewCount": 0,
        "BackgroundURL": "background url",
        "IsUnlimitedRoomEnabled": true
    }
}
```

请求包字段说明

字段	类型	说明

CallbackCommand	String	回调命令
Operator_Account	String	发起销毁房间请求的操作者 UserID
RoomId	String	房间 ID
EventType	String	解散类型：分为主动解散(DestroyByUser)和系统自动解散(DestroyBySystem)两种
EventTime	Integer	事件触发的毫秒级别时间戳
RoomName	String	房间名称
RoomType	String	房间类型：Meeting（会议房间）
Owner_Account	String	房主 ID
IsSeatEnabled	Bool	是否支持麦位
TakeSeatMode	String	麦位模式：None（关闭），FreeToTake（自由上麦），ApplyToTake（申请上麦）
MaxMemberCount	Integer	最大房间成员数量
MaxSeatCount	Integer	最大麦位数
Category	Array	房间类型标识，整数数组类型
CustomInfo	String	自定义字段
IsMessageDisabled	Bool	禁止所有成员发文本消息
CoverURL	String	房间封面 url

ActivityStatus	Integer	直播间活跃状态：用户自定义标记
IsPublicVisible	Bool	房间是否公开
ViewCount	Integer	用户进房总次数
BackgroundURL	String	房间背景图url
IsUnlimitedRoomEnabled	Bool	是否开启房间混流，支持高并发场景，默认为 false。当为 true 时对应于 sdk 侧 TUILiveKit 视频房

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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介
- REST API：解散房间

更新房间信息之后回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调实时查看房间信息的变更，包括对变动房间信息的实时记录（例如记录日志，或者同步到其他系统）。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端变动房间信息。
- App 管理员通过 REST API 变动房间信息。

回调发生时机

房间信息变动成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 `Live.CallbackAfterUpdateRoomInfo`
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":"Live.CallbackAfterUpdateRoomInfo",
    "Operator_Account":"bob",
    "RoomInfo" : {
        "RoomId":"live-room",
        "RoomName" : "rname-123",
        "MaxMemberCount" : 300,
        "IsMessageDisabled" : false,
        "CoverURL": "https://xxxx.png",
        "Category": [1,2,3],
        "ActivityStatus":1,
        "IsPublicVisible": false,
        "TakeSeatMode": "ApplyToTake",
        "CustomInfo": "custom123"
     }
    "EventTime":1670574414123//毫秒级别，事件触发时间戳
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
Operator_Account	String	发起创建群组请求的操作者 UserID
RoomId	String	房间 ID
RoomName	String	房间名称
MaxMemberCount	Integer	最大房间成员数量

IsMessageDisabled	Bool	禁止所有成员发文本消息
CustomInfo	String	自定义字段
TakeSeatMode	String	麦位模式：None（关闭），FreeToTake（自由上麦），ApplyToTake（申请上麦）
IsPublicVisible	Bool	房间是否公开
CoverURL	String	房间封面 url
Category	Array	房间类型标识，整数数组类型
ActivityStatus	Integer	直播间活跃状态：用户自定义标记
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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介
- REST API：更新房间

设置房间自定义信息后回调

最近更新时间：2025-06-20 15:11:32

功能说明

App 后台可以通过该回调实时查看房间 metadata 数据的设置。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端设置房间自定义信息。
- App 管理员通过 REST API 设置房间自定义信息。

回调发生时机

设置房间自定义信息成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 `Live.CallbackAfterSetMetadata`
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": "Live.CallbackAfterSetMetadata",
    "Operator_Account": "brennanli",
    "RoomId": "live-room1111",
    "Metadata": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": "value2"
        }
    ],
    "EventTime": 1739965885831
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
Operator_Account	String	发起销毁房间请求的操作者 UserID
RoomId	String	房间 ID
Metadata	Array	metadata 数据
Key	String	metadata 对应的 key
Value	String	metadata 对应的 value
EventTime	Integer	事件触发的毫秒级别时间戳

版权所有：腾讯云计算（北京）有限责任公司

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
- 设置房间自定义信息

删除房间自定义信息后回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调实时查看房间自定义信息的删除。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端删除 metadata。
- App 管理员通过 REST API 删除 metadata。

回调发生时机

房间 metadata 数据删除成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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	`Live.CallbackAfterDelMetadata`
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": "Live.CallbackAfterDelMetadata",
    "Operator_Account": "brennanli",
    "RoomId": "live-room1111",
    "Keys": [
        "key1", "key2"
    ],
    "EventTime": 1739966441121
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
Operator_Account	String	发起销毁房间请求的操作者 UserID
RoomId	String	房间 ID
Keys	Array	metadata 数据里对应的 Key
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
- 删除房间自定义信息

创建房间达到70%预警回调

最近更新时间：2025-06-20 15:11:32

功能说明

App 后台可以通过该回调查看当前房间创建数量。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端创建房间。
- App 管理员通过 REST API 创建房间。

回调发生时机

创建房间成功之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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	`Live.CallbackAfterCreateRoomReachingThreshold`。
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": "Live.CallbackAfterCreateRoomReachingThreshold",
    "Operator_Account": "brennanli",
    "EventTime": 1739966441121,
    "CurRoomCount": 400,
    "MaxRoomCount": 500
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令。
Operator_Account	String	发起创建房间请求的操作者 UserID。
CurRoomCount	String	房间当前创建数。
MaxRoomCount	Array	房间最大创建数。
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

用户相关
房间成员状态变更回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调实时查看房间用户上下线状态变更。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户通过客户端创建房间成功。
- App 管理员通过 REST API 创建房间成功。

回调发生时机

房间创建成功之后。

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
```

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 Live.CallbackAfterMemberStateChanged
contenttype	固定值为 JSON

请求包示例

```
{
    "CallbackCommand":"Live.CallbackAfterMemberStateChanged",
    "RoomId":"room_id",
    "EventType":"Online", //Online, Offline 上下线两种
    "EventCause":"Enter", //分为以下四种 进房 Enter, 退房 Leave, 心跳断线 HeartbeatInterrupt, 心跳恢复 HeartbeatRecover
    "MemberList":[
        {
            "Member_Account": "jared"
        },
        {
            "Member_Account": "tommy"
        }
    ],
    "EventTime":1703589922780
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
RoomId	String	房间 ID
EventType	String	事件类型，分为房间成员上线和下线两种：Online，Offline
EventCause	String	事件缘由，分为以下四种 进房 Enter，退房 Leave，心跳断线 HeartbeatInterrupt，心跳恢复 HeartbeatRecover
MemberList	Array	涉及的成员列表
Member_Account	String	账号 id

EventTime	Interger	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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介

连麦相关
麦位列表变更之后回调

最近更新时间：2025-05-20 17:56:32

功能说明

App 后台可以通过该回调实时查看麦位列表变更消息。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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 用户上下麦。
- 操作麦位锁定。

回调发生时机

麦位信息变更之后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 Mic.CallbackAfterSeatInfoChanged
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":"Mic.CallbackAfterSeatInfoChanged",
    "RoomId":"rid-123",
    "SeatList":[
        {
            // 麦位编号
            "Index": 1,
            // 如果当前麦位有人占用，则返回对应用户的 ID，否则该字段不会设置
            "Member_Account": "user2",
             // false:允许上麦 true:禁止上麦
             "IsTakenDisabled": false,
             // false:允许推视频流 true:禁止推视频流
             "IsVideoDisabled": false,
             // false:允许推音频流 true:禁止推音频流
             "IsAudioDisabled": false
         }
    ],
    "EventTime":1670574414123//毫秒级别，事件触发时间戳
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
RoomId	String	房间 ID
SeatList	Array	麦位列表
Index	Integer	麦位号

Member_Account	String	麦位上的用户 ID，如果该麦没有用户，则没此字段
IsTakenDisabled	Bool	麦位锁定
IsVideoDisabled	Bool	禁止麦位视频流
IsAudioDisabled	Bool	禁止麦位音频流
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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介

PK 相关
创建 PK 之后回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调实时查看用户创建 Battle 的信息。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是Live 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 用户通过客户端创建 battle 成功。
- App 管理员通过 REST API 创建 battle 成功。

回调发生时机

创建 battle 成功之后

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 `Live.CallbackAfterCreateBattle`
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": "Live.CallbackAfterCreateBattle",
    "Operator_Account": "brennanli",
    "BattleId": "4siHsNsWN/T3aub9zKraqPbSEGRX2z6gs3LDFi+d/3M=",  //battle
id
    "FromRoomId": "pk-3", //发起battle的主叫方房间id
    "ToRoomIdList": [ //参与battle的被叫方房间id列表
        "pk-5"
    ],
    "Timeout": 30000, //如果NeedResponse为true，那么最多等待被叫方处理时长，如果
被叫方不处理表示放弃参加battle
    "Duration": 60000, //battle开启后的时长，单位毫秒
    "NeedResponse": false,  //为false表示直接开始battle,如果为true得等被叫方处
理后才能开始battle
    "ExtensionInfo": "extension pk",
    "EventTime":1739954005000
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
Operator_Account	String	发起 battle 请求的操作者 UserID
BattleId	String	PK 对应的 battle id
FromRoomId	String	battle 的主叫方房间 id
ToRoomIdList	Array	参与 battle 的被叫方房间 id 列表

Timeout	Integer	创建 battle 后, 等待被叫方处理的最大等待时长，单位 ms
Duration	Integer	battle 的时长
NeedResponse	Bool	创建 battle 后是否需要被叫方房主同意加入
ExtensionInfo	String	battle 的扩展信息
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
ErrorCode	Integer	必填	错误码，此处填0表示忽略应答结果
ErrorInfo	String	必填	错误信息

参考

- 第三方回调简介

PK 开启之后回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调及时知晓 battle 开启。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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

- battle 创建后，无需等待直接进入开启状态。
- battle 创建后，所有被叫方处理完 battle 请求后或者整体等待时间结束后，只要同意加入 battle 的被叫方房主大于等于1人，battle 正式开始。

回调发生时机

battle 正式开启后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 `Live.CallbackAfterStartBattle`
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": "Live.CallbackAfterStartBattle",
    "BattleId": "4siHsNsWN/T3aub9zKraqI4zZAyPRpXQhdtKv1q4HOs=", // battle
id
    "Duration": 60000, // battle时长
    "CreateTime": 1739954005, // battle创建时间，单位秒
    "StartTime": 1739954005, // battle开启时间，单位秒
    "FromRoomInfo": {  // battle 主叫方信息
        "RoomId": "pk-3",
        "Owner_Account": "brennan"
    },
    "ToRoomList": [ // battle 被叫方信息
        {
            "RoomId": "pk-5",
            "Owner_Account": "tandy"
        }
    ],
    "EventTime":1739954005000 //回调触发时间，单位毫秒
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令
BattleId	String	battle ID
Duration	Integer	battle 的时长

CreateTime	Integer	battle 创建时间
StartTime	Integer	battle 开启时间
FromRoomInfo	String	battle 的主叫方信息
ToRoomList	Array	参与 battle 的被叫方信息
RoomId	String	房间 ID
Owner_Account	String	房主账号 ID
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

PK 结束之后回调

最近更新时间：2025-04-03 16:12:04

功能说明

App 后台可以通过该回调及时知晓 battle 结束。

注意事项

- 要启用回调，必须配置回调 URL，并打开本条回调协议对应的开关，配置方法详见 第三方回调配置 文档。
- 回调的方向是 Live 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

- battle 正常到时间结束。
- battle 进行期间，所有房主退出 battle。
- battle 创建好后，因没有房主同意 battle 而无法进入开启模式直接结束。

回调发生时机

battle 结束后。

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&Clien
tIP=$ClientIP&OptPlatform=$OptPlatform
```

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 Live.CallbackAfterEndBattle
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": "Live.CallbackAfterEndBattle",
    "BattleId": "4siHsNsWN/T3aub9zKraqPfGQAimPhdFoe6VWhtz9lY=",
    "Duration": 30000,
    "CreateTime": 1739956115,
    "StartTime": 1739956115,
    "EndTime": 1739956145,
    "OpType": 0, // 0表示时间到了正常结束， 1表示battle开启后， 所有主播都退出了导致的结束， 2表示battle创建后，因无主播加入导致直接未开启就结束
    "FromRoomInfo": {  //主叫方信息
        "RoomId": "pk-3",
        "Score": 0,
        "Owner_Account": 144115245353757792
    },
    "ToRoomList": [ //被叫方信息
        {
            "RoomId": "pk-5",
            "Score": 0,
            "Owner_Account": 144115245442327522
        }
    ],
    "EventTime": 1739956146119
}
```

请求包字段说明

字段	类型	说明
CallbackCommand	String	回调命令

BattleId	String	battle ID
Duration	Integer	battle 的时长
CreateTime	Integer	battle 创建时间
StartTime	Integer	battle 开启时间
EndTime	Integer	battle 结束时间
OpType	Integer	0表示时间到了正常结束；1表示battle开启后，所有主播都退出了导致的结束；2表示 battle 创建后，因无主播加入导致直接未开启就结束
FromRoomInfo	String	battle 的主叫方信息
ToRoomList	Array	参与 battle 的被叫方信息
RoomId	String	房间 ID
Score	Integer	房间对应的 PK 分数
Owner_Account	String	房主 ID
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

错误码（TUILiveKit）

最近更新时间：2025-04-08 14:37:02

客户端错误码

通用错误码

错误码	描述
0	操作成功
-1	暂未归类的通用错误
-2	请求被限频，请稍后重试
-3	重复操作
-1000	未找到 SDKAppID，请在 腾讯云视立方 SDK 控制台 确认应用信息
-1001	调用 API 时，传入的参数不合法，检查入参是否合法
-1002	未登录,请调用 Login 接口
-1003	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
-1004	该功能需要开通额外的套餐，请在 腾讯云视立方 SDK 控制台 按需开通对应套餐

本地用户渲染，视频管理，音频管理 API 回调错误定义

错误码	描述
-1100	系统问题，打开摄像头失败。检查摄像头设备是否正常
-1101	摄像头没有系统授权, 检查系统授权
-1102	摄像头被占用，检查是否有其他进程使用摄像头
-1103	当前无摄像头设备，请插入摄像头设备解决该问题
-1104	系统问题，打开麦克风失败。检查麦克风设备是否正常
-1105	麦克风没有系统授权，检查系统授权
-1106	麦克风被占用
-1107	当前无麦克风设备

版权所有：腾讯云计算（北京）有限责任公司

-1108	获取屏幕分享对象失败，检查屏幕录制权限
-1109	开启屏幕分享失败，检查房间内是否有人正在屏幕分享

房间管理相关 API 回调错误定义

错误码	描述
-2101	需要进房后才可使用此功能
-2102	房主不支持退房操作，Conference（会议）房间类型: 可以先转让房主，再退房。 LivingRoom（直播）房间类型: 房主只能解散房间
-2103	当前房间类型下不支持该操作
-2105	创建房间 ID 非法，自定义 ID 必须为可打印 ASCII 字符（0x20-0x7e），最长48个字节
-2107	房间名称非法，名称最长30字节，字符编码必须是 UTF-8 ，如果包含中文
-2108	当前用户已在别的房间内，单个 roomEngine 实例只支持用户进入一个房间，如果要进入不同的房间请先退房或者使用新的 roomEngine 实例

房间内用户信息 API 回调错误定义

错误码	描述
-2200	未找到该用户

房间内用户发言管理 API 回调错误定义&房间内麦位管理 API 回调错误定义

错误码	描述
-2300	需要房主权限才能操作
-2301	需要房主或者管理员权限才能操作
-2310	信令请求无权限，例如取消非自己发起的邀请
-2311	信令请求 ID 无效或已经被处理过
-2312	信令请求重复
-2340	最大麦位超出套餐包数量限制
-2344	麦位编号不存在
-2360	当前麦位音频被锁

版权所有：腾讯云计算（北京）有限责任公司

-2361	需要向房主或管理员申请后打开麦克风
-2370	当前麦位视频被锁, 需要由房主解锁麦位后，才能打开摄像头
-2371	需要向房主或管理员申请后打开摄像头
-2372	当前麦位视频被锁, 需要由房主解锁麦位后，才能打开屏幕分享
-2373	需要向房主或管理员申请后打开屏幕分享
-2380	当前房间已开启全员禁言
-2381	当前房间内，您已被已禁言

房间预加载API回调错误定义

错误码	描述
-4001	当前房间不支持预加载

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

版权所有：腾讯云计算（北京）有限责任公司

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](https://buy.cloud.tencent.com/avc) 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。

服务端错误码

说明：

- 以下错误码适用于2.0版本 SDK，公共错误码（60000到79999）参见 错误码 文档。
- 1.x版本 SDK 的服务端错误码请参见 服务端错误码（旧）文档。

错误码	含义说明
100001	服务器内部错误，请重试
100002	请参数非法，请根据错误描述检查请求是否正确
100003	房间ID 已被使用，请选择别的房间ID
100004	房间不存在，或者曾经存在过，但是目前已经被解散。
100005	非房间成员

100006	操作权限不足
100007	无付费信息，需在控制台购买套餐包
100008	房间成员已满
100009	标签数量超上限
100010	房间 ID 已被使用，并且操作者为房主，可以直接使用
100011	房间 ID 已被 IM 占用，可以换一个房间 ID 使用，或者先通过 IM 接口解散该群
100012	频率超过限制，例如创建房间超过频率超限，同一房间 ID， 1秒内只能创建一次
100013	超过付费上限，例如麦位数，pk场次房间数量等超过付费限制
100015	无效的房间类型
100016	该成员已经被封禁
100017	该成员已经被禁言
100018	当前房间需要密码才能进入
100019	进房密码错误
100020	管理员数量超过上限
100102	信令请求冲突
100200	麦位已锁定，可以尝试换一个麦位
100201	当前麦位已经有人了
100202	已经处于排麦状态
100203	已经处于麦上状态
100204	没有在排麦列表中

版权所有：腾讯云计算（北京）有限责任公司

100205	麦位已满
100206	未在麦上
100210	已经有用户在麦位上
100211	该房间不支持连麦
100251	连麦列表为空
100400	当前连线不存在或结束
100401	该房间已经在连线中
100402	该房间存在待处理的连线请求
100403	当前房间与其他房间连线中
100404	超过连线和 battle 房间数量上限
100405	短时间内连线过于频繁，稍等一会再试
100411	该场次 battle 不存在或已结束
100412	发起的 battle 里没有一个有效的房间
100413	短时间内频繁发起 battle, 稍等一会再试
100414	该房间已经不在 battle 中
100415	该房间处于其他的 battle 场次中
100416	该房间存在待处理的 battle 请求
100419	该房间处于 battle 中
100420	该 battle 场次还未开始
100421	该 battle 场次已经结束
10050	房间 meta 数据中的 key 数量超过上限

版权所有：腾讯云计算（北京）有限责任公司

0	
100501	房间 meta 数据中单个 key 对应的 val 超过最大字节数限制
100502	房间 meta数据中所有 key 对应的 val 总和超过最大字节数限制
100503	删除房间 meta 数据时候，被删除的 key 没有一个存在
100504	房间 meta 数据中的 key 大小超过了最大字节数限制

发布日志（TUILiveKit）

最近更新时间：2025-06-12 16:00:12

2025年04月

发布动态	描述	发布时间
Version 3.1.0	**TUILiveKit** 新功能 • Android & iOS：新增组件瀑布流列表组件。 • Android & iOS：新增组件主播准备页组件。 功能增强 • 直播创建房间支持设置房主上麦后常驻麦位。 • 直播/观众端界面增加渐变色蒙层。 流程优化 • 优化观众列表显示逻辑。 • 优化进房拉取房间信息数据流程。 问题修复 • iOS：优化断网状态下，连线功能 UI 异常问题。 • iOS：修复异常退出 UI 导致的房间状态异常问题。 • iOS：修复弱网环境下麦位请求 UI 异常/黑屏问题。 • iOS：修复 iPhone 7 语聊房 UI 显示不全问题。 • Android：修复偶现的观众数量与列表不一致问题。 • Android：修复 PK 图标显示异常问题。 **RTCRoomEngine SDK** 问题修复 • 调整客户端能力位的加载时机，确保 RoomEngine callback 成功之前能力位加载完成。 • 新增消息接口，支持使用 RoomEngine 收到 IM 消息，并支持忽略禁言。 • 修复 TRTC 进房后返回 result 为 0 导致进房失败的问题。 • SSO 通道的通知增加用户资料映射，减轻 SDK 并发补齐资料对后台的压力。 • 增加云控配置变更回调，及时更新配置中心和限频参数。 • 设置客户端码率上限，对齐官网。 • 修复 RoomEngine 与 TIMPush 混用时，账号无法登录的问题。 **LiveStreaCore** 新增接口 • 垫片推流接口。 功能优化	2025.06.10

	• createRoom 新增 `keepOwnerOnSeat` 字段，支持房主永不下麦。 • `OnUserNetworkQualityChanged` 回调新增观众拉流时本地网络质量检测。 • `onKickedOutOfRoom` 回调增加被踢原因。 • 优化禁言列表、管理员列表的同步策略，降低后台请求频次。 • 优化直播连线的状态同步逻辑，降低后台请求频次。 • 优化观众未进房时预览拉流逻辑。 问题修复 • 修复未进房时的错误码未能正确返回的问题。 • 修复上麦之后没有清空本地的 layout 通知记录，导致下麦后拉混流时 layout 不再回调的问题。 • 修复重复退房时参数无效的问题。 • 修复网络异常回复后，未能及时同步麦位信息导致的黑屏问题。 • 修复被拉黑封禁后，再次音画秒开，依然漏声音和画面的问题。 • 修复网络异常恢复后，本地用户下麦未能及时关闭采集的问题。 • 修复网络异常情况下，`onRequestTimeout` 回调中 TUIUserInfo 不准确的问题。	
Version 3.0.0	**TUILiveKit** • Flutter：新增语聊房场景。 • Flutter：统一国际化文案。 • Flutter：修复已知问题。 • iOS&Android：高级特效播放器鉴权问题修复。 • iOS：端高级美颜支持使用历史效果。 • iOS&Android：新增直播间用户管理功能（禁言、踢出房间、踢下麦、禁止音视频等）。 • iOS&Android：删除未完整实现的功能（用户等级、音乐播放、直播类型）。 • iOS&Android：修复频繁刷礼物导致直播间异常的问题。 • iOS&Android：统一国际化文案。 • iOS：端直播列表信息支持定制化修改。 • Android：优化一些特殊机型导致的问题。 • iOS&Android：连麦观众进房后自动下麦。 • iOS&Android：修复已知问题。 **LiveStreamCore** • Flutter：新增语聊组件。	2025.04.03

	• iOS：静态库发布。 • iOS&Android：State状态可访问，可订阅变更。 • iOS&Android：增加接口 switchCamera&enableMirror。 • iOS&Android：统一国际化文案。 • iOS&Android：修复已知问题。 **RoomEngine SDK** 新增功能 • 直播类型房间新增支持模糊搜索成员。 • LayoutManager 新增自定义布局接口。 • 新增屏幕共享暂停接口。 • Web 端补齐直播列表接口。 • Web 端增加房间 metadata 接口。 接口变更 • 错误码变更。 • OnRemoteUserEnterRoom/OnRemoteUserLeaveRoom 参数 userInfo 补齐禁言字段。 • iOS 端禁言 API 的 swift API 声明变更。 修复问题 • 优化屏幕分享异常问题。 • 修复 Android 偶现崩溃问题。 • 修复进房后可能无法打开麦克风问题。	

2025年02月

发布动态	描述	发布时间
Version 2.9.0	**TUILiveKit** 新功能： • Android&iOS：VideoLiveKit 接口增加：禁用关注功能，结束、退出直播间。 • Android&iOS：悬浮窗支持主播端。 • Web：支持自动拉快直播、UI 适配手机端浏览器、支持收发文字和表情消息。 • Electron：优化混流排版能力。 Bug 修复： • Android&iOS：修复一些已知问题。 • Electron：修复一些已知问题。 **RoomEngine SDK** • Flutter：新增连线、PK 功能支持。	2025.02.12

	• 全平台：`onRoomUserCountChanged` 房间人数变更回调优化。	

2024年12月

发布动态	描述	发布时间
Version 2.8.0	稳定性优化： • iOS： 优化了弹幕组件的稳定性。 Bug 修复： • Android&iOS：修复一些已知问题。	2024.12.18

2024年11月

发布动态	描述	发布时间
Version 2.4.3	新功能 • 新增 LiveCoreView 组件。 • 支持悬浮窗功能。	2024.11.20

2024年10月

发布动态	描述	发布时间
Version 2.4.2	新功能 • 新增 VideoLiveKit、VoiceRoomKit 接口。	2024.10.12

2024年08月

发布动态	描述	发布时间
Version 2.4.1	新功能 • 新增直播连线功能。	2024.08.29

2024年07月

发布动态	描述	发布时间
Version 2.1.0	新功能 • Android&iOS：演示交互升级。 Bug 修复 • Android&iOS：修复了一些运行时崩溃的问题。	2024.07.22

版权所有：腾讯云计算（北京）有限责任公司

2024年06月

发布动态	描述	发布时间
Version 2.0.0	新功能 • Android&iOS：UI2.0交互更新。 • Android&iOS：支持房间列表。 • Android：支持滑动切换直播间。	2024.06.26

2024年05月

发布动态	描述	发布时间
Version 1.0.1	新功能 • Android&iOS：支持语音聊天室场景。 • Android&iOS：支持主播发弹幕。 Bug 修复 • Android：修复麦克风连接问题和仪表盘显示问题。 • Android：修复锁定座位时麦克风状态问题。	2024.05.24

2024年04月

发布动态	描述	发布时间
Version 1.0.0	新功能 • Android&iOS：支持主播开播。 • Android&iOS：支持观众进入直播间。 • Android&iOS：支持观众话筒连接。 • Android&iOS：支持观众列表。 • Android&iOS：支持弹幕组件。 • Android&iOS：支持礼物组件。	2024.04.22

版权所有：腾讯云计算（北京）有限责任公司

常见问题（TUILiveKit）

最近更新时间：2025-04-03 16:12:04

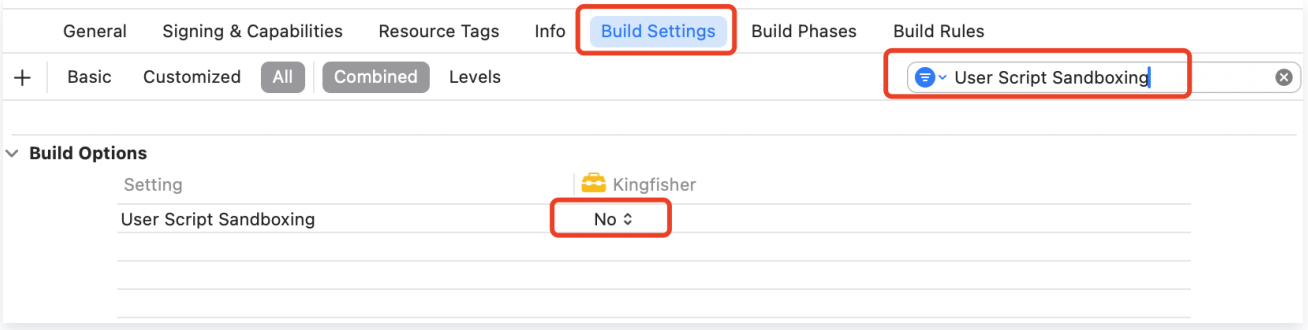
iOS

Xcode 15 编译报错?

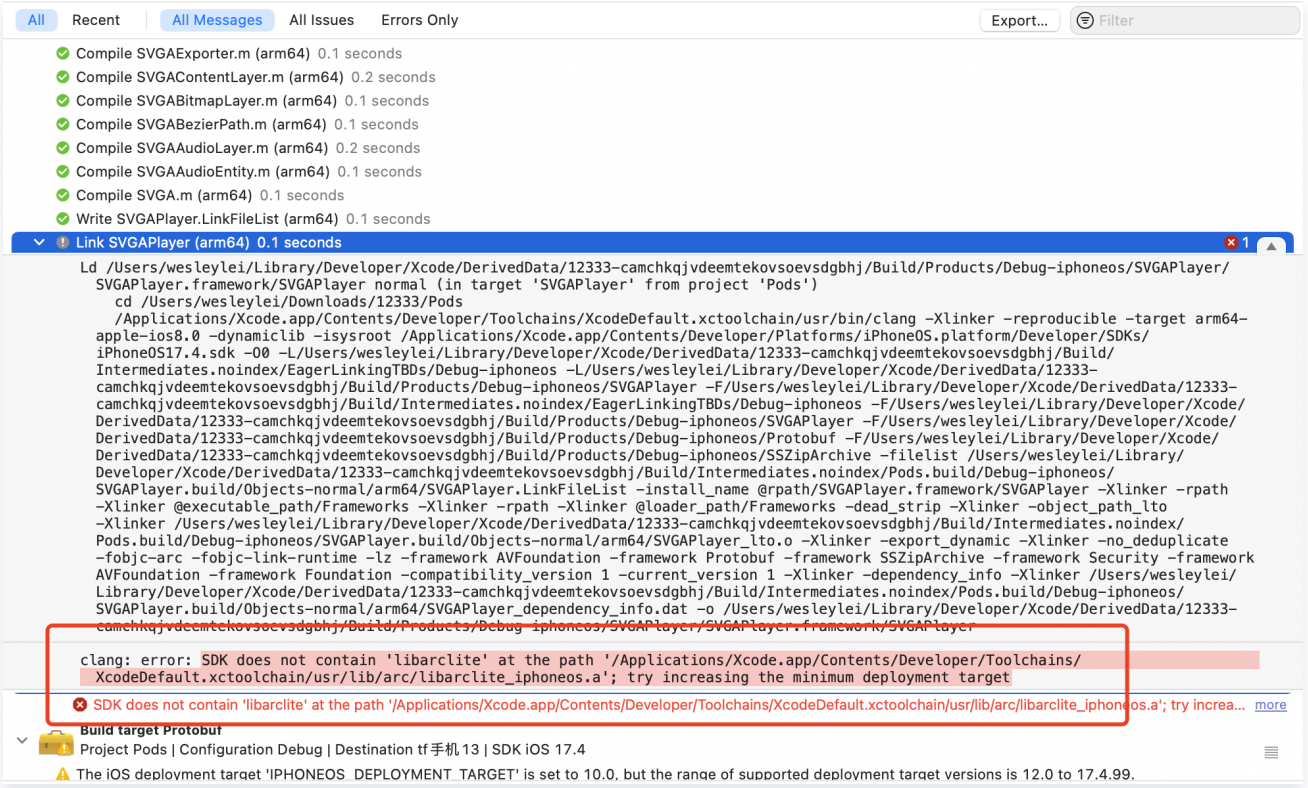
1. 出现Sandbox: rsync，编译报错截图：



可以在“Build Settings”中把"User Script Sandboxing"设置为NO



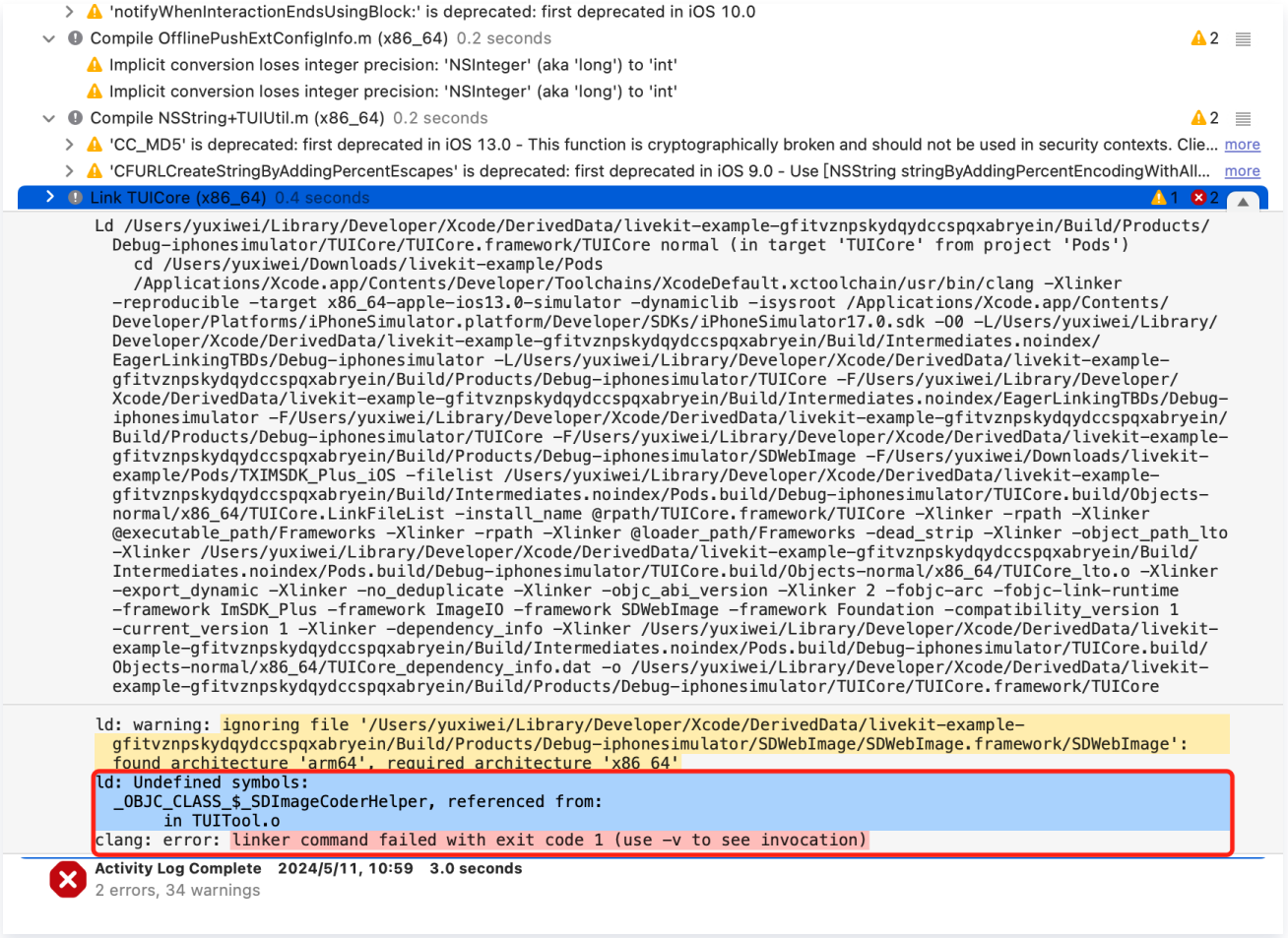
2. 如果出现 SDK does not contain，编译报错截图：



请在 Podfile 添加如下代码：

```
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)，编译报错截图：



可以在 Podfile 添加如下代码

```
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
```

TUILiveKit 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 'TUILiveKit'
```

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 'TUILiveKit/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 `TUILiveKit/Professional` 。

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。
> - 更多日志相关设置参考： 日志输出配置 。

Android

TUILiveKit 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，例如创建房间信令、连麦信令等核心逻辑都使用 IM服务，如果您已经购买有其他 IM 产品，也可以参照 TUILiveKit 逻辑进行适配。

allowBackup 异常，如何处理?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-36
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(true).
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xml:5:5-53:19 to override.
```

- **问题原因**：多个模块的 `AndroidManifest.xml` 中都配置了 `allowBackup` 属性，造成冲突。
- **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；并在 `AndroidManifest.xml` 文件的 application 节点中添加 `tools:replace="android:allowBackup"` 表示覆盖其他模块的设置，使用您自己的设置。修复示例如图所示：

```
oginActivity.java | AndroidManifest.xml | VideoViewFactory.java | Ar
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        tools:replace="android:allowBackup"
        android:allowBackup="false"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

Activity 主题问题：Activity need to use a Theme.AppCompat theme，如何处理?

```
FATAL EXCEPTION: main
Process: com.trtc.uikit.livekit.example, PID: 15190
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.trtc.uikit.livekit.example/com.trtc.uikit.livekit.example.login
.LoginActivity}: java.lang.IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) with this activity.
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3730)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3885)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:101)
    at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2332)
    at android.os.Handler.dispatchMessage(Handler.java:107)
    at android.os.Looper.loop(Looper.java:230)
    at android.app.ActivityThread.main(ActivityThread.java:8115) <1 internal line>
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:526)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1034)
```

- **问题原因：由于** `LoginActivity` **继承自** `AppCompatActivity` **，所以要给** `LoginActivity` **设置一个** `Theme.AppCompat` **主题。**
- **解决方法：您可以在您工程的** `AndroidManifest.xml` **文件中** `LoginActivity` **的配置里，增加一个** `Theme.AppCompat` **主题。您也可以使用自己的** `Theme.AppCompat` **主题。修复示例如图所示：**

```
<activity
    android:name=".login.LoginActivity"
    android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
</activity>
```

跳转浏览器打开网页地址失败?

解决办法：您可以在您工程的 `AndroidManifest.xml` **文件中，新增如下配置：**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <queries>
        <intent>
            <action android:name="android.intent.action.VIEW" />
            <data android:scheme="https" />
        </intent>
    </queries>
```